



**BITS** Pilani  
Pilani|Dubai|Goa|Hyderabad

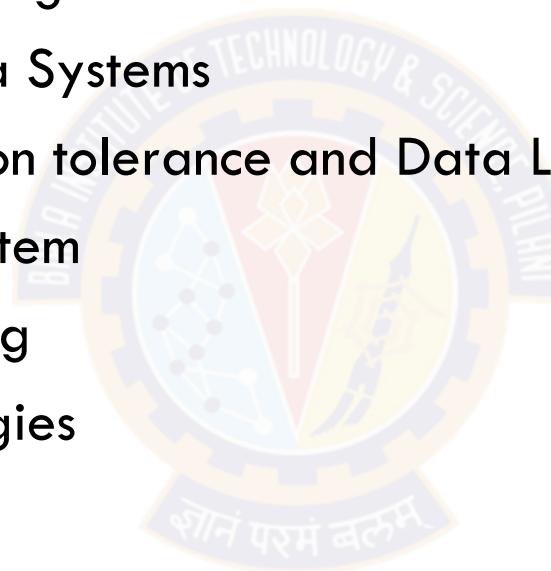
# DSECL ZG 522: Big Data Systems

## Session 1.1: Introduction to Big Data

Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Course Outline

- S1: Introduction to Big Data and data locality
- S2: Parallel and Distributed Processing
- S3: Big Data Analytics and Big Data Systems
- S4: Consistency, Availability, Partition tolerance and Data Lifecycle
- S5: Hadoop architecture and filesystem
- S6: Distributed Systems Programming
- S7-S9: Hadoop ecosystem technologies
- S10: NoSQL Databases
- S11-14: In-memory and streaming - Spark
- S15: Big Data on Cloud
- S16: Amazon storage services



# Books

T1	Seema Acharya and Subhashini Chellappan. <i>Big Data and Analytics</i> . Wiley India Pvt. Ltd. Second Edition
T2	Raj Kamal and Preeti Saxena, <i>Big Data Analytics</i> . McGraw Hill Education (India) Pvt.Ltd
R1	DT Editorial Services. <i>Big Data - Black Book</i> . DreamTech. Press. 2016
R2	Kai Hwang, Jack Dongarra, and Geoffrey C. Fox. <i>Distributed and Cloud Computing: From Parallel Processing to the Internet of Things</i> . Morgan Kauffman 2011
AR	Additional Reading (As per topic)

# Transformations of Applications and Database Technologies

(Additional presentation)

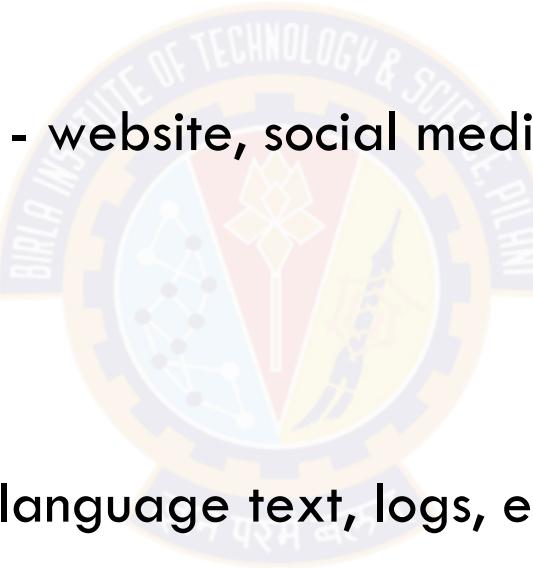
# Topics for today

- Motivation
  - ✓ Why do modern Enterprises need to work with volume data
  - ✓ What is Big Data and data classification
  - ✓ Scaling RDBMS
- What is a Big Data System
  - ✓ Characteristics
  - ✓ Design challenges
- Architecture
  - ✓ High level architecture of Big Data solutions
  - ✓ Technology ecosystem
  - ✓ Case studies



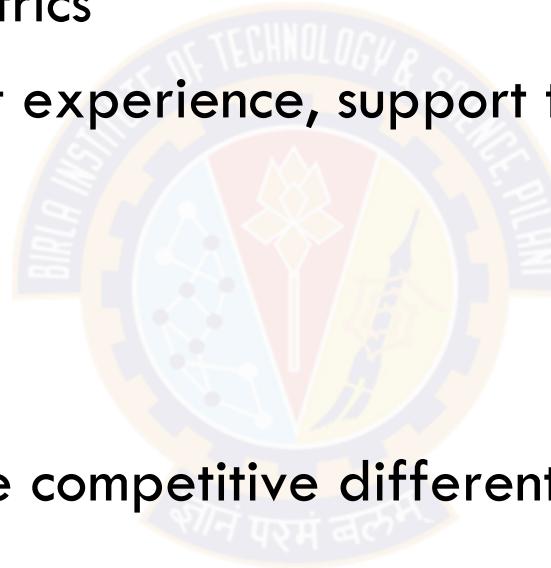
## Example of a data-driven Enterprise: A large online retailer (1)

- What data is collected
  - ✓ Millions of transactions and browsing clicks per day across products, users
  - ✓ Delivery tracking
  - ✓ Reviews on multiple channels - website, social media, customer support
  - ✓ Support emails, logged calls
  - ✓ Ad click and browsing data
  - ✓ ...
- Data is a mix of metrics, natural language text, logs, events, videos, images etc.

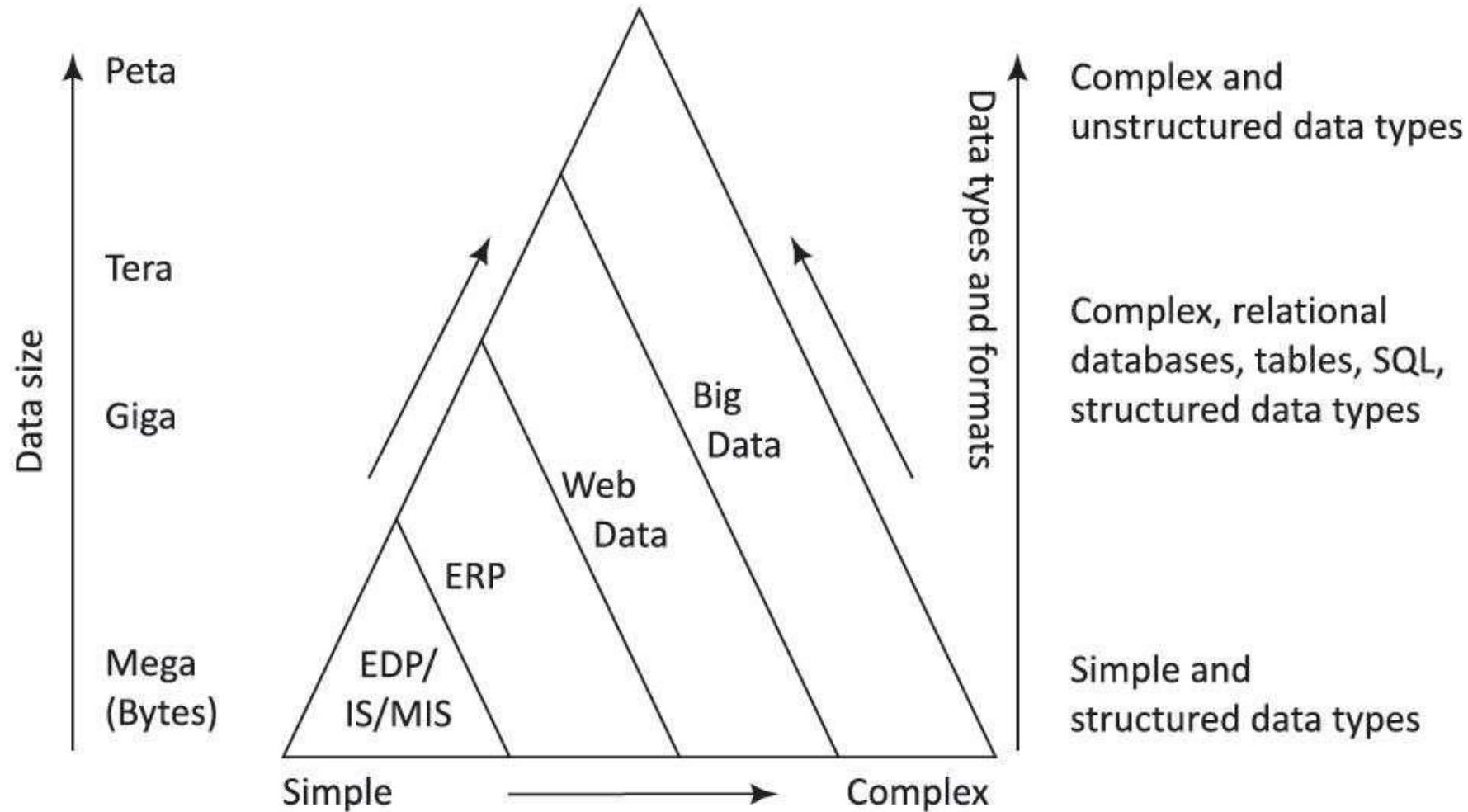


## Example of a data-driven Enterprise: A large online retailer (2)

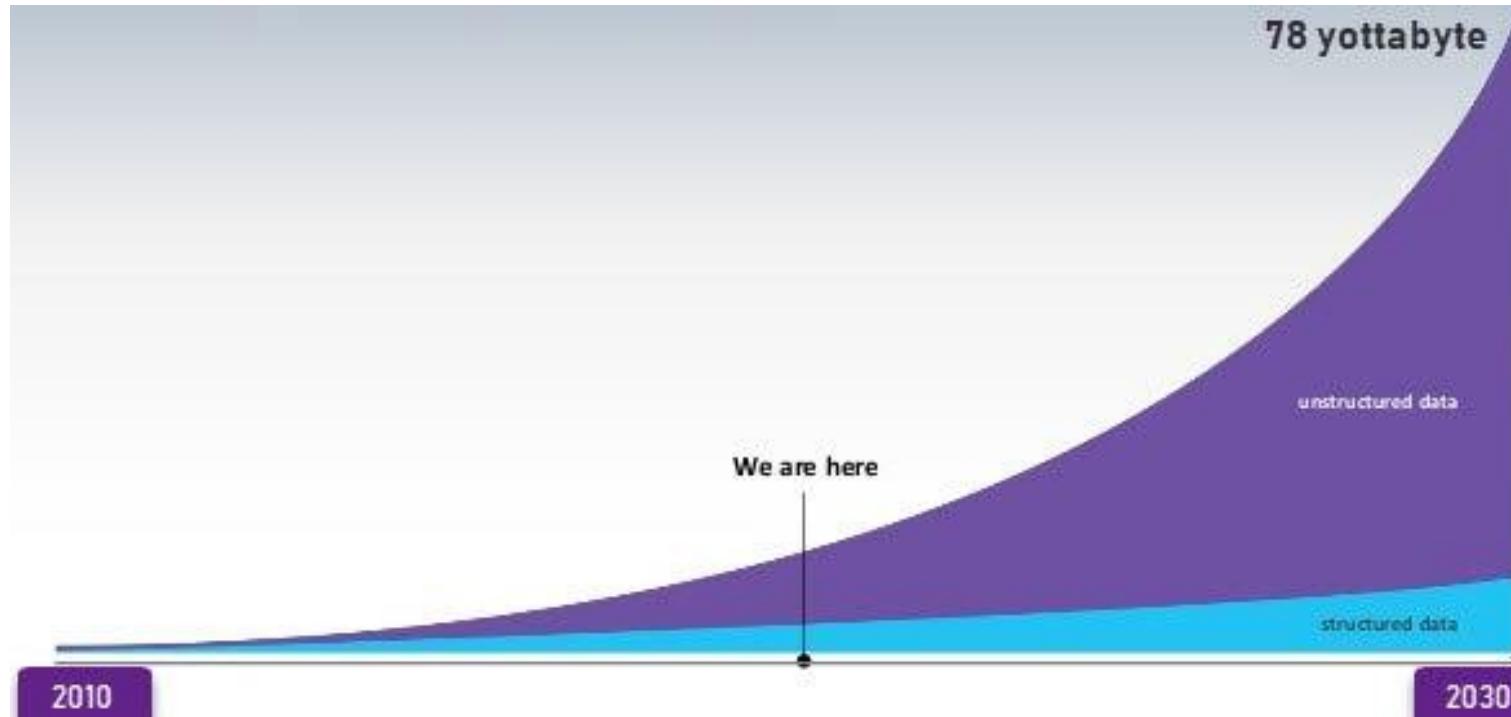
- What is this data used for
  - ✓ User profiling for better shopping experience
  - ✓ Operations efficiency metrics
  - ✓ Improve customer support experience, support training
  - ✓ Demand forecasting
  - ✓ Product marketing
  - ✓ ...
- Data is the only way to create competitive differentiators, retain customers and ensure growth



# Evolution of Bigdata and their characteristics



# Data volume growth



- Facebook: 500+ TB/day of comments, images, videos etc.
- NYSE: 1TB/day of trading data
- A Jet Engine: 20TB / hour of sensor / log data

Source : What is big data?

# Variety of data sources



[Source : What is Big Data?](#)

# Big Data Characteristics

- How big is the Big Data?
- What is big today maybe not big tomorrow
- One's Big Data may be small Data for another

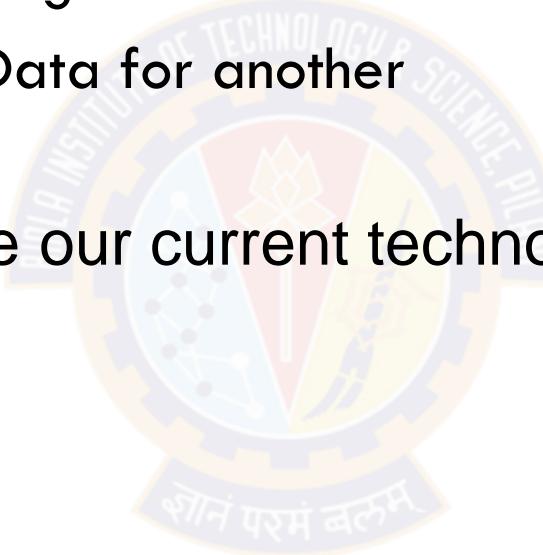
Any data that can challenge our current technology in some manner can be considered as Big Data

Volume

Communication

Speed of Generating

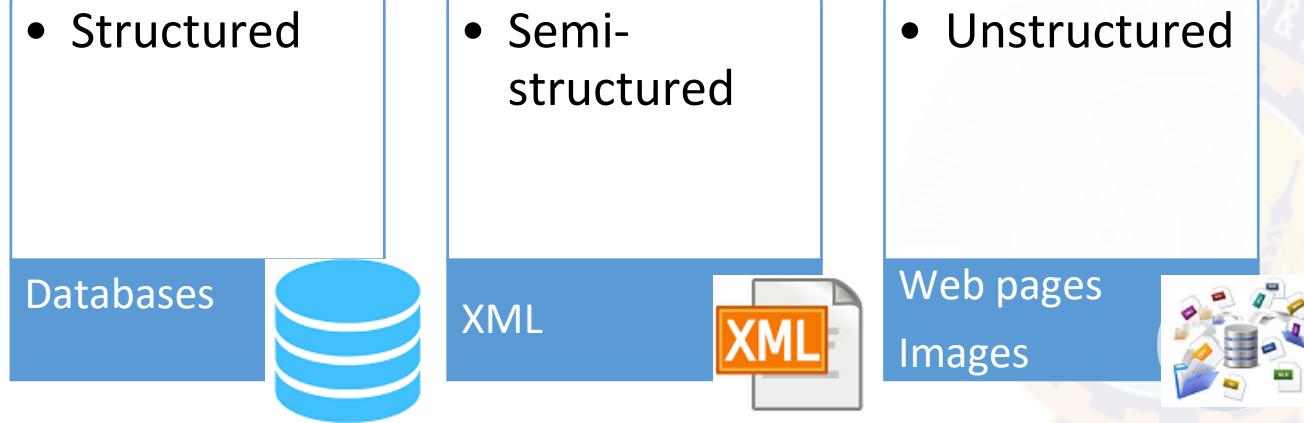
Meaningful Analysis



# Big Data Challenge

- In the past, the most difficult problem for businesses was how to store all the data.
- The challenge now is no longer to store large amounts of information, but to understand and analyze this data.
- By making sense out of this data through sophisticated analytics, and by presenting the key findings in an easily discernable fashion, we can derive value out of Big data
- Big data creates a new **Digital divide** – Those who can process Big data and those who cannot
- **Data is only as useful as the decisions it enables**

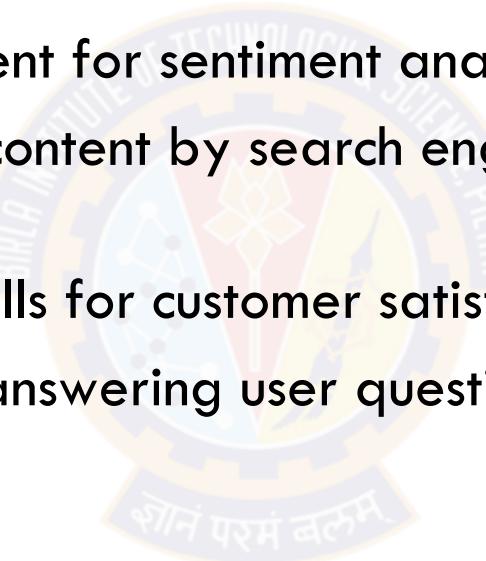
# Data classification



- Structured data is metrics, events that can be put in RDBMS with fixed schema
- Semi-structured data are XML, JSON structure where traditional RDBMS have support with varying efficiency but needs new kind of NoSQL databases
- New applications produce unstructured data which could be natural language text and multi-media content

# Data usage pattern

- Higher demand now of analysing unstructured data to glean insights
- Examples
  - ✓ analysis of social media content for sentiment analysis
  - ✓ analysis of unstructured text content by search engines on the web as well as within enterprise
  - ✓ analysis of support emails, calls for customer satisfaction
  - ✓ NLP / Conversational AI for answering user questions from backend data



# Structured Data

- Data is transformed and stored as per pre-defined schema
- Traditionally stored in RDBMS
- CRUD operations on records
- ACID semantics (Atomicity, Consistency, Isolation, Durability)
- Fine grain security and authorisation
- Known techniques on scaling RDBMS - more on this later
- Typically used by Systems of Record, e.g. OLTP systems, with strong consistency requirements and read / write workloads



```
TABLE Employee (
    emp_id int PRIMARY KEY,
    name varchar (50),
    designation varchar(25),
    salary int,
    dept_code int FOREIGN KEY
)
```

# Semi-Structured Data

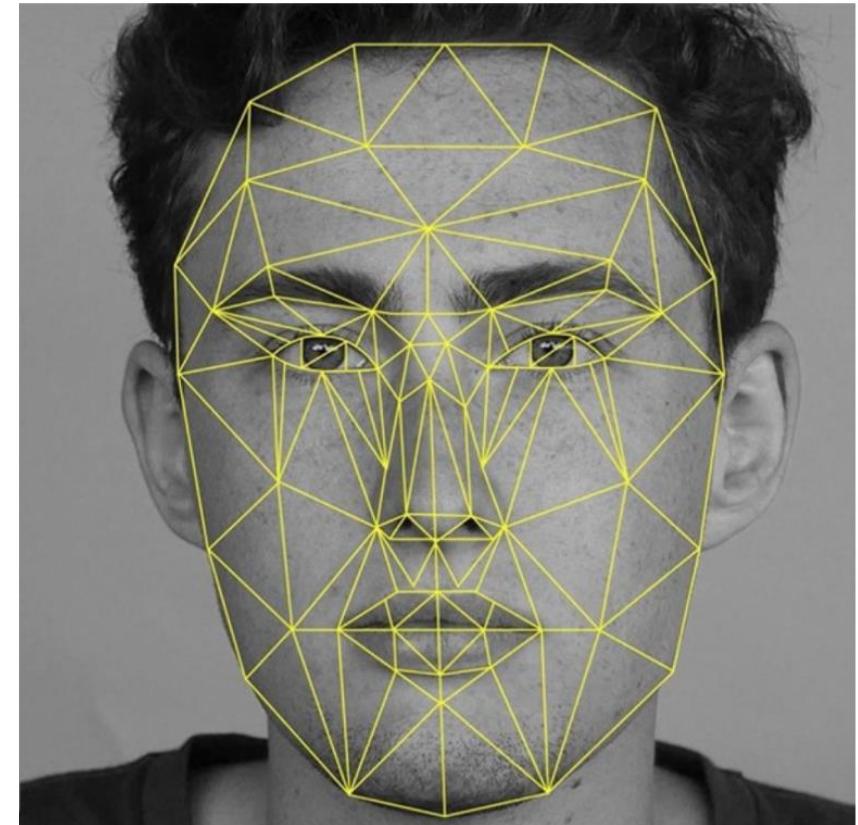
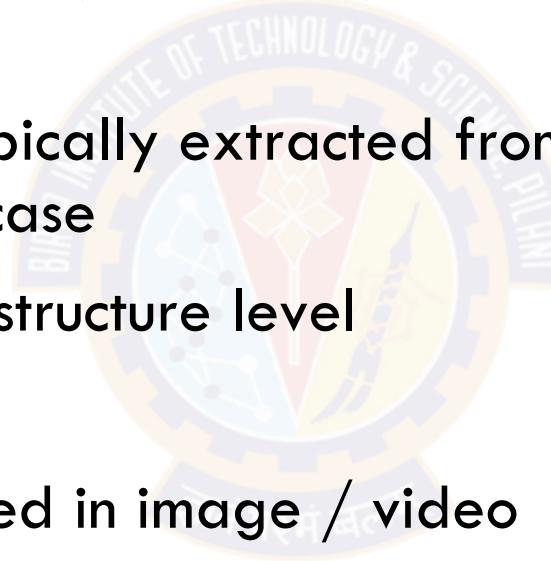
- No explicit data and schema separation
- Models real life situations better because attributes for every record could be different
- Easy to add new attributes
- XML, JSON structures
- Databases typically support flexible ACID properties, esp consistency of replicas
- Typically used by Systems of Engagement, e.g. social media



```
{  
  "title": "Sweet fresh strawberry",  
  "type": "fruit",  
  "description": "Sweet fresh strawberry",  
  "image": "1.jpg",  
  "weight": 250,  
  "expiry": 30/5/2021,  
  "price": 29.45,  
  "avg_rating": 4  
  "reviews": [  
    { "user": "p1", "rating": 2, "review": "....." }]
```

# Unstructured Data (1)

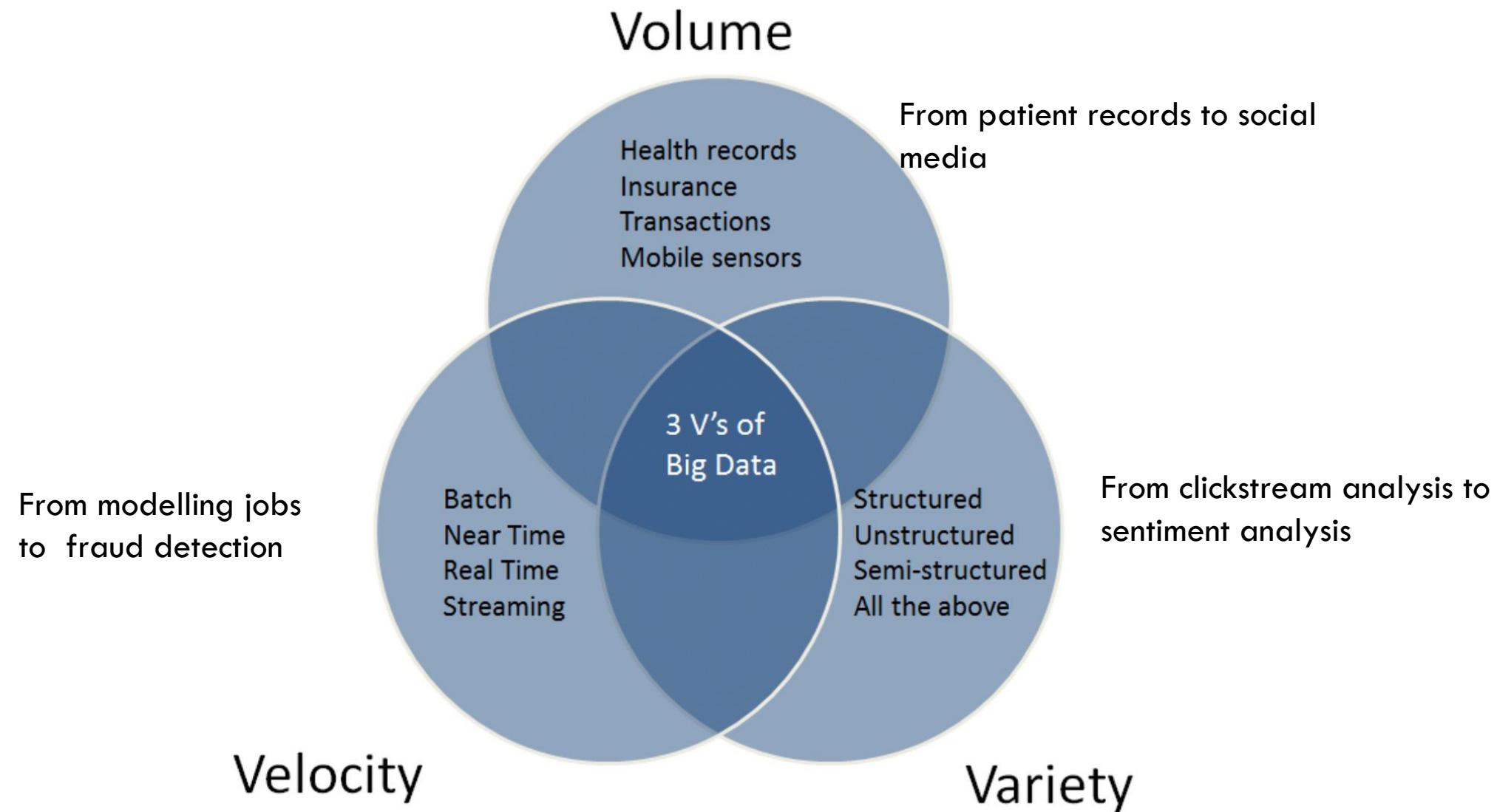
- More real-life data
  - ✓ video, voice, text, emails, chats, comments, reviews, blogs ...
- There is some structure that is typically extracted from the data depending on the use case
  - ✓ image adjustments at pixel structure level
  - ✓ face recognition from video
  - ✓ tagging of features extracted in image / video
  - ✓ annotation of text



# Unstructured Data (2)

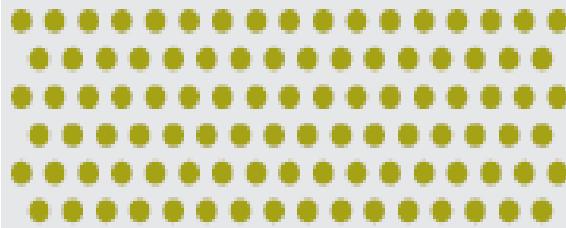
- What can we do with it ?
  - ✓ Data mining
    - Association rule mining, e.g. market basket or affinity analysis
    - Regression, e.g. predict dependent variable from independent variables
    - Collaborative filtering, e.g. predict a user preference from group preferences
  - ✓ NLP - e.g. Human to Machine interaction, conversational systems
  - ✓ Text Analytics - e.g. sentiment analysis, search
  - ✓ Noisy text analytics - e.g. spell correction, speech to text

# Define Big Data – 3Vs



# Big Data – More Vs

Volume



Data at scale

Terabytes to petabytes of data

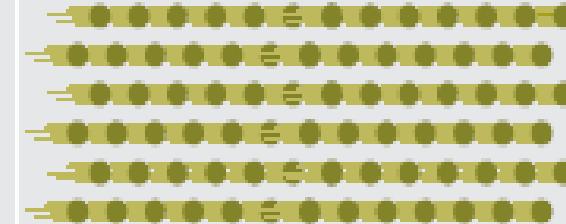
Variety



Data in many forms

Structured, unstructured,  
text, multimedia

Velocity



Data in motion

Analysis of streaming data  
to enable decisions within  
fraction of a second

Velocity

Machine data as well as data coming from new sources is being  
ingested at speeds not even imagined a few years ago.

Veracity

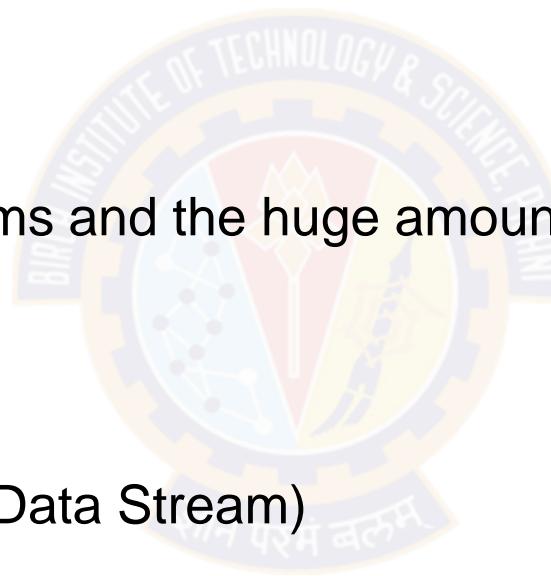


Data uncertainty

Managing the reliability and predictability  
of inherently imprecise data types

# Velocity – Data streams

- . What are Data Streams?
  - Continuous streams
  - Huge, Fast, and Changing
  - Scan the data only once
- . Why Data Streams?
  - The arriving speed of streams and the huge amount of data are beyond our capability to store them.
  - “Real-time” processing
- . Window Models
  - Landscape window (Entire Data Stream)
  - Sliding Window
  - Damped Window
- . Mining Data Stream



# Some make it 4Vs

## Volume

Terabytes to Exabytes of Existing Data to be processed

**Data at Rest**

## Velocity

Streaming Data, Milliseconds to seconds to respond

**Data in Motion**

## Variety

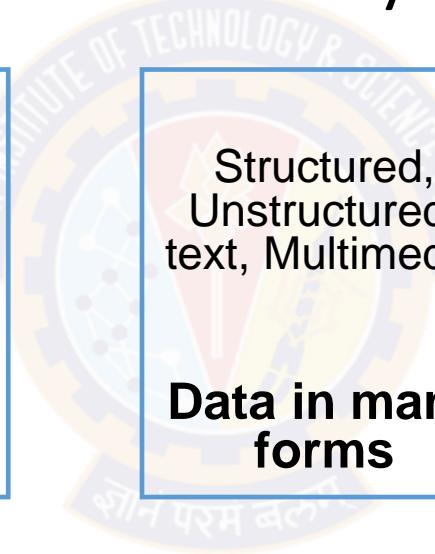
Structured, Unstructured, text, Multimedia

**Data in many forms**

## Veracity

Uncertainty due to data inconsistency, incompleteness, ambiguity, latency, deception etc

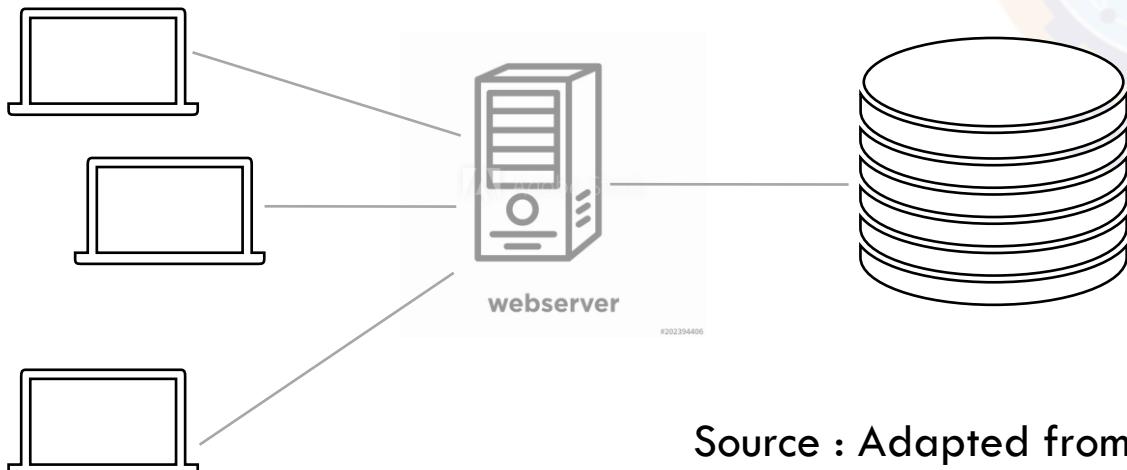
**Data in Doubt**



# Isn't a traditional RDBMS good enough ?

## Example Web Analytics Application

- Designing an application to monitor the page hits for a portal
- Every time a user visiting a portal page in browser, the server side keeps track of that visit
- Maintains a simple database table that holds information about each page hit
- If user visits the same page again, the page hit count is increased by one
- Uses this information for doing analysis of popular pages among the users



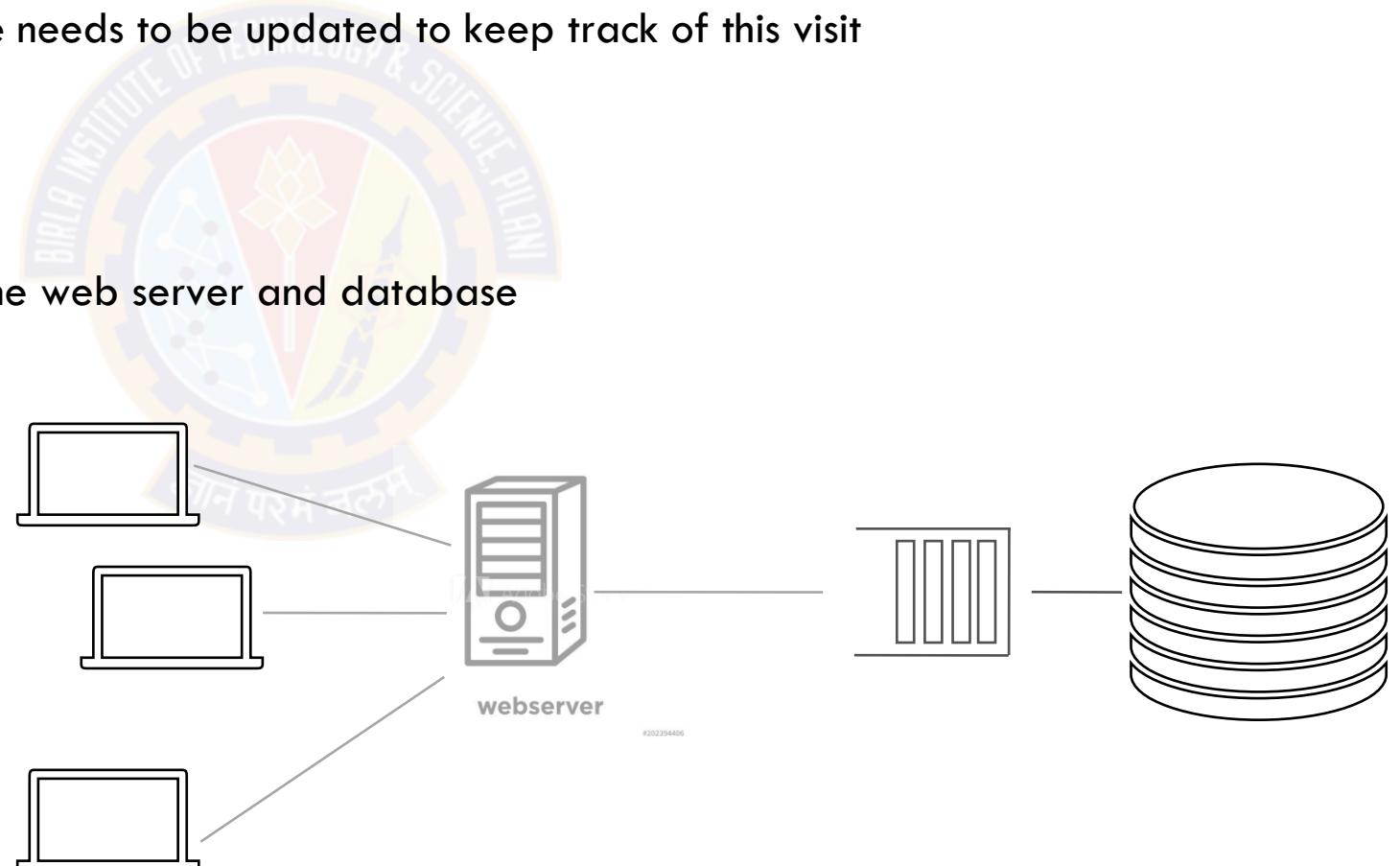
Column	Data Type
Id	Integer
User_ID	Integer
Page_URL	Varchar
Page_count	Long

Source : Adapted from Big Data by Nathan Marz

# Scaling with intermediate layer

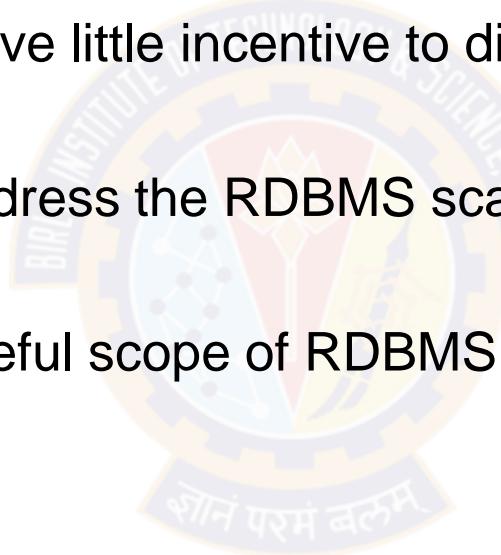
## Using a queue

- Portal is very popular, lot of users visiting it
  - ✓ Many users are concurrently visiting the pages of portal
  - ✓ Every time a page is visited, database needs to be updated to keep track of this visit
  - ✓ Database write is heavy operation
  - ✓ Database write is now a bottleneck !
- Solution
  - ✓ Use an intermediate queue between the web server and database
  - ✓ Queue will hold messages
  - ✓ Message will not be lost



# Road Blocks to RDBMS Scaling

- RDBMS technology is a forced fit for modern interactive software systems
- RDBMS is incredibly complex internally, and changes are difficult
- Vendors of RDBMS technology have little incentive to disrupt a technology generating billions of dollars for them annually
- It requires huge investments to address the RDBMS scaling issue and find out a viable solution
- Techniques used to extend the useful scope of RDBMS technology fight symptoms but not the disease itself

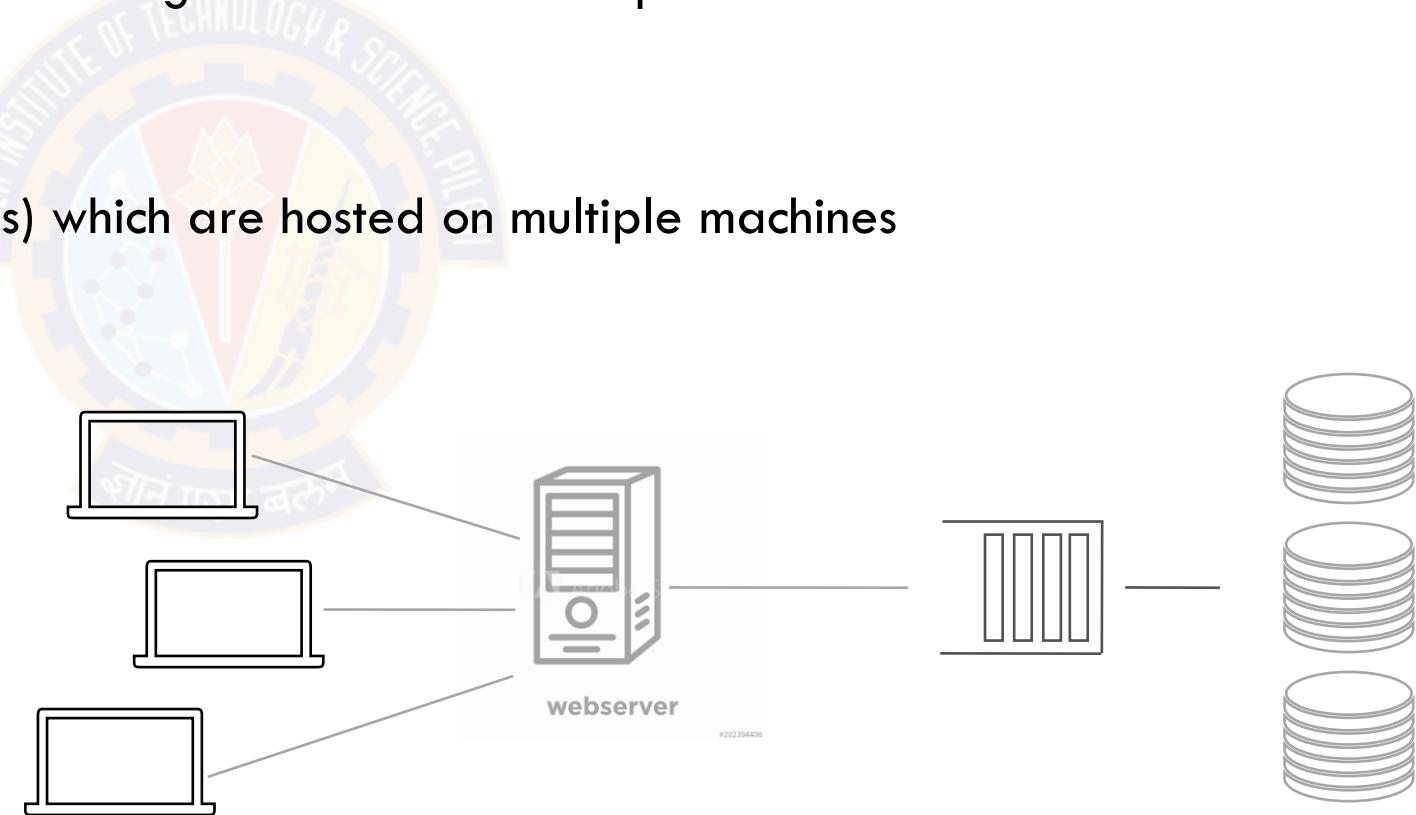


Provocative statement:

*The relational database will be a footnote in history, because of fundamental flaws in the RDBMS approach to managing relational data of the present era*

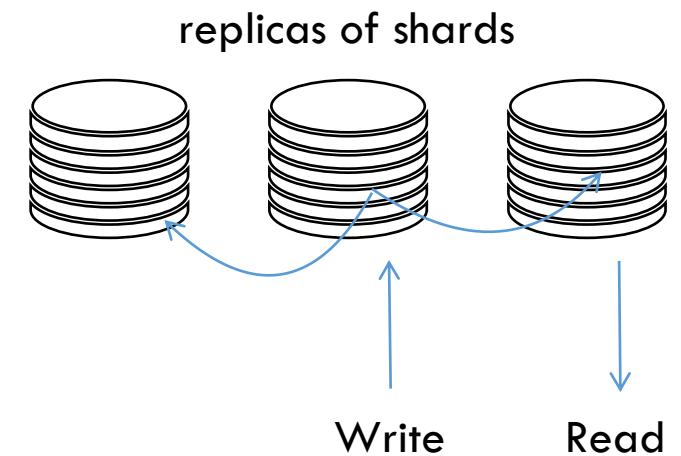
# Scaling with Database Partitions (Sharding)

- Application is too popular
  - ✓ Users are using it very heavily, increasing the load on application
  - ✓ Maintaining the page view count is becoming difficult even with queue
- Solution
  - ✓ Use database partitions
  - ✓ Data is divided into partitions (shards) which are hosted on multiple machines
  - ✓ Database writes are parallelized
  - ✓ Scalability increasing
  - ✓ Also complexity increasing!



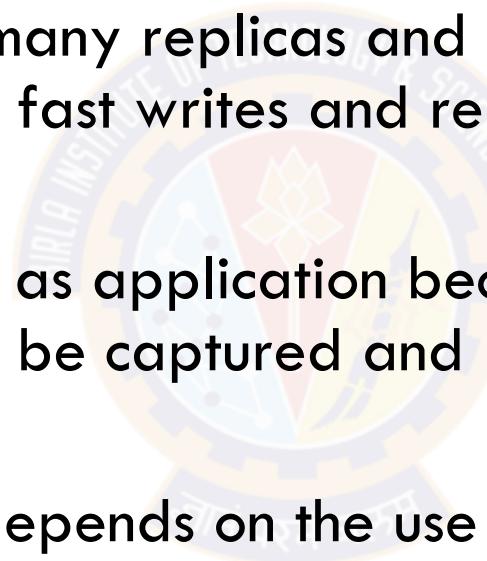
# Issues with RDBMS sharding

- With too many shards some disk is bound to fail
- Fault tolerance needs shard replicas - so more things to manage
- Complex logic to read / write because need to locate the right shard - human errors can be devastating
- Keep re-sharding and balancing as data grows or load increases
- What is the consistency semantics of updating replicas ? Should a read on a replica be allowed before it is updated ?
- Is it optimised when data is written once and read many times or vice versa ?

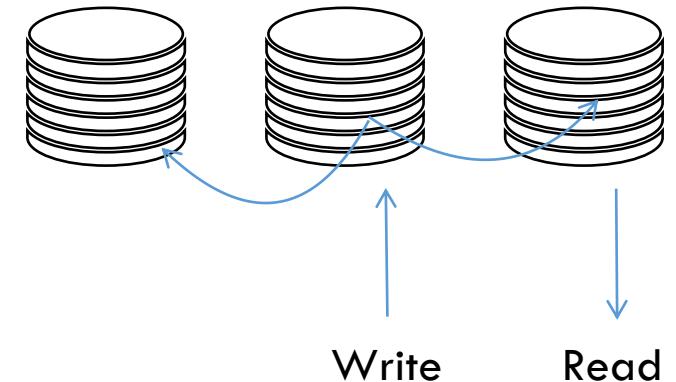


# Issues with RDBMS (1)

- Not all BigData use cases need strong ACID semantics, esp Systems of Engagement
  - ✓ Becomes a bottleneck with many replicas and many attributes - need to optimize fast writes and reads with less updates
- Fixed schema is not sufficient ... as application becomes popular more attributes need to be captured and DB modelling becomes an issue.
  - ✓ Which attributes are used depends on the use case.



replicas of shards in a social site DB



```
{  
    "title": "Sweet fresh strawberry",  
    "type": "fruit",  
    "description": "Sweet fresh strawberry",  
    "image": "1.jpg",  
    "weight": 250,  
    "expiry": 30/5/2021,  
    "price": 29.45,  
    "avg_rating": 4  
    "reviews": [  
        {"user": "p1", "rating": 2, "review": "....."}, ...  
    ]  
}
```

want to add field applicable to some products

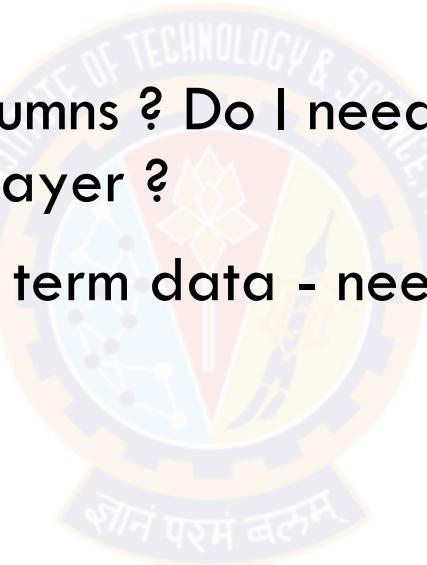
ACID to BASE Transformation in NoSQL era

[ACID vs BASE Concepts - Data Science Blog \(data-science-blog.com\)](https://data-science-blog.com/acid-vs-base-concepts/)

[ACID to BASE Transformation - DataScienceCentral.com](https://www.datasciencecentral.com/acid-to-base-transformation/)

# Issues with RDBMS (2)

- Very wide de-normalized attribute sets
- Data layout formats - column or row major - depends on use case
  - ✓ What if we query only few columns ? Do I need to touch the entire row in storage layer ?
- Expensive to retain and query long term data - need low cost solution



```
{  
  "title": "Sweet fresh strawberry",  
  "type": "fruit",  
  "description": "Sweet fresh strawberry",  
  "image": "1.jpg",  
  "weight": 250,  
  "expiry": 30/5/2021,  
  "price": 29.45,  
  "avg_rating": 4  
  "reviews": [  
    { "user": "p1", "rating": 2, "review": "....." }, ...  
  ]  
}
```

what if a JSON had 1000+ attributes  
demographic records  
for millions of users

# Topics for today

- Motivation
  - ✓ Why do modern Enterprises need to work with data
  - ✓ What is Big Data and data classification
  - ✓ Scaling RDBMS
- What is a Big Data System
  - ✓ Characteristics
  - ✓ Design challenges
- Architecture
  - ✓ High level architecture of Big Data solutions
  - ✓ Technology ecosystem
  - ✓ Case studies



# Characteristics of Big Data Systems (1)

- Application does not need to bother about common issues like sharding, replication
  - ✓ Developers more focused on application logic rather than data management
- Easier to model data with flexible schema
  - ✓ Not necessary that every record has same set of attributes
- If possible, treat data as immutable
  - ✓ Keep adding timestamped versions of data values
  - ✓ Avoid human errors by not destroying a good copy

replicated / partitioned storage



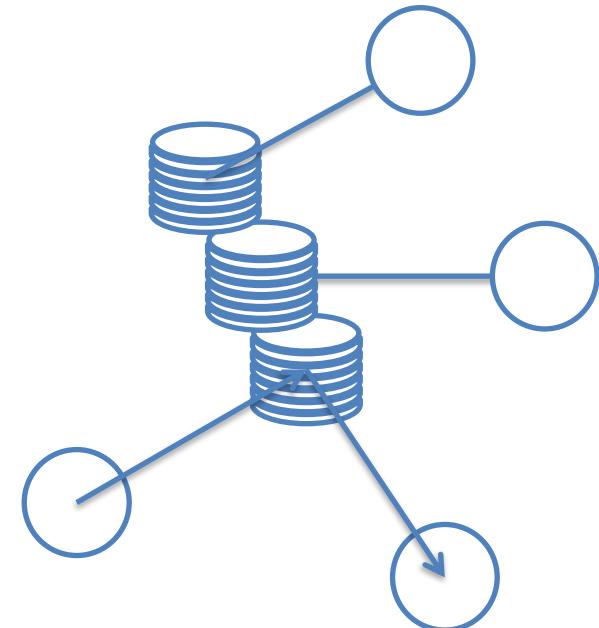
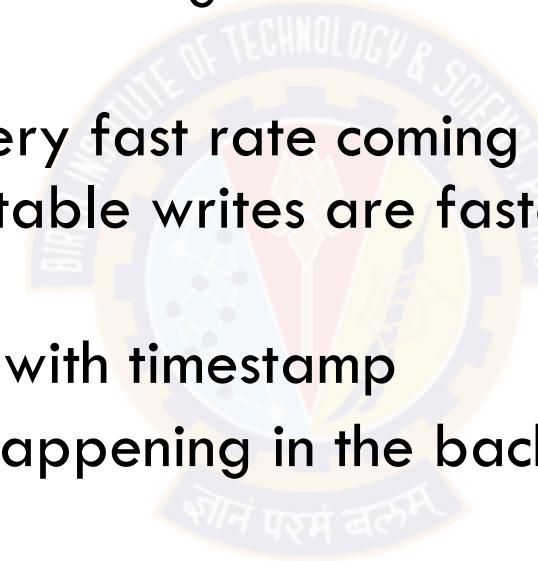
key-value      document

graph

t1, k    t2, k    t3, k    ...

# Characteristics of Big Data Systems (2)

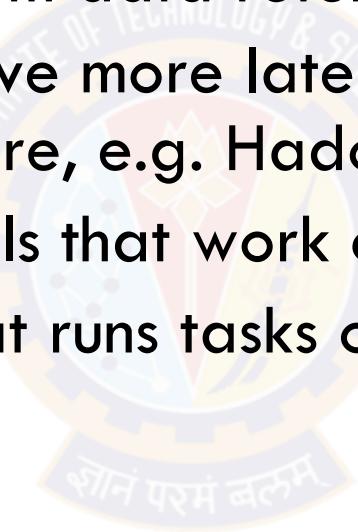
- Application specific consistency models
  - ✓ a reader may read a replica that's has not been updated yet as in “read preference” options in MongoDB
  - ✓ e.g. comments on social media
- Handles high data volume, at very fast rate coming from variety of sources because immutable writes are faster with flexible consistency models
  - ✓ Keep adding data versions with timestamp
  - ✓ Replica updates can keep happening in the background



Cassandra is a NoSQL database for write-heavy workload and eventual consistency

# Characteristics of Big Data Systems (3)

- Built as distributed and incrementally scalable systems
  - ✓ add new nodes to scale as in a Hadoop cluster
- Options to have cheaper long term data retention
  - ✓ long term data reads can have more latency and can be less expensive to store on commodity hardware, e.g. Hadoop file system (HDFS)
- Generalized programming models that work close to the data
  - ✓ e.g. Hadoop map-reduce that runs tasks on data nodes



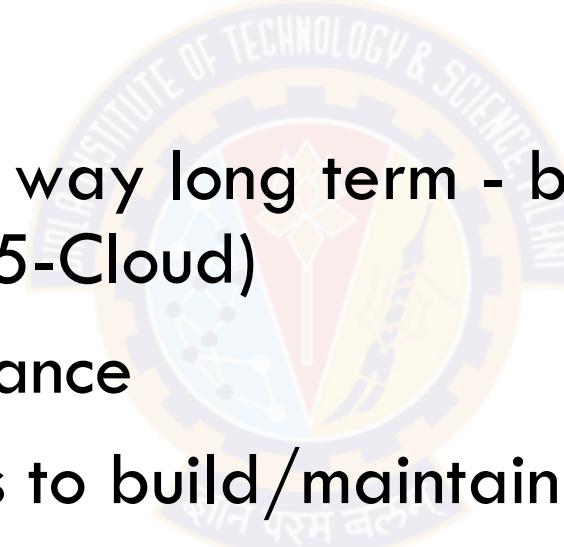
# Challenges in Big Data Systems (1)

- Latency issues in algorithms and data storage working with large data sets (S1\*-LoR)
- Basic design considerations of Distributed and Parallel systems - reliability, availability, consistency (S2, S3, S4)
- What data to keep and for how long - depends on analysis use case
- Cleaning / Curation of data (S4-Lifecycle)
- Overall orchestration involving large volumes of data (S9)
- Choose the right technologies from many options, including open source, to build the Big Data System for the use cases (S6 onwards)

\* Si: Topic is discussed in session i

# Challenges in Big Data Systems (2)

- Programming models for analysis (S5, S6 - MapReduce, S13-Spark)
- Scale out for high volume (S6-Hadoop, S10-NoSQL, S13-Spark)
- Search (S10-NoSQL)
- Cloud is the cost effective way long term - but need to host Big Data outside the Enterprise (S15-Cloud)
- Data privacy and governance
- Skilled coordinated teams to build/maintain Big Data Systems and analyse data



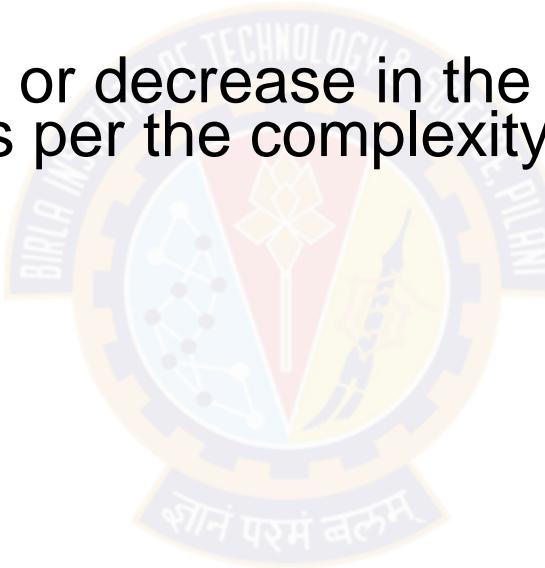
# Topics for today

- Motivation
  - ✓ Why do modern Enterprises need to work with data
  - ✓ What is Big Data and data classification
  - ✓ Scaling RDBMS
- What is a Big Data System
  - ✓ Characteristics
  - ✓ Design challenges
- Architecture
  - ✓ High level architecture of Big Data solutions
  - ✓ Technology ecosystem
  - ✓ Case studies
- Summary



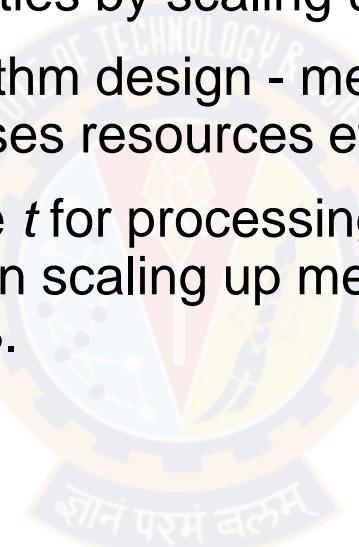
# Needs of Big Data Systems

- Processing of large data volume
- Intensive computations
- Scalability enables increase or decrease in the capacity of data storage, processing and analytics, as per the complexity of computations and volume of data
- Types of Scalability
  - ✓ Vertical
  - ✓ Horizontal
  - ✓ Elastic



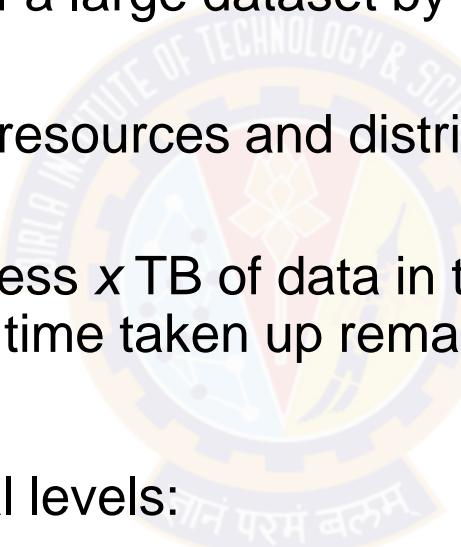
# Vertical Scalability (Scaling Up)

- Scaling up the given system's resources and increasing the system's analytics, reporting and visualization capabilities
- Solve problems of greater complexities by scaling up
- Demands architecture-aware algorithm design - means designing the algorithm according to the architecture that uses resources efficiently
- For example,  $x$  TB of data take time  $t$  for processing, code size with increasing complexity increase by factor  $n$ , then scaling up means that processing takes equal, less or much less than  $(nxt)$  for  $x$  TB.
- Server changes
  - More powerful CPU
  - More memory
  - Product companies
    - Enjoys Free Performance Lunch facilitated by Moore's law
    - Exploited by RDBMS aka SQL Databases by new releases



# Horizontal Scalability (Scaling Out)

- Horizontal scalability means increasing the number of systems working in coherence and scaling out the workload
- Processing different datasets of a large dataset by increasing number of systems running in parallel.
- Scaling out means using more resources and distributing the processing and storage tasks in parallel
- If  $r$  resources in a system process  $x$  TB of data in time  $t$ , then the  $(pxx)$  TB on  $p$  parallel distributed nodes such that the time taken up remains  $t$  or is slightly more than  $t$
- Parallelization of jobs at several levels:
  - (i) distributing separate tasks onto separate threads on the same CPU,
  - (ii) distributing separate tasks onto separate CPUs on the same computer and
  - (iii) distributing separate tasks onto separate computers



# Elastic Scaling – Cloud computing

## Cloud computing

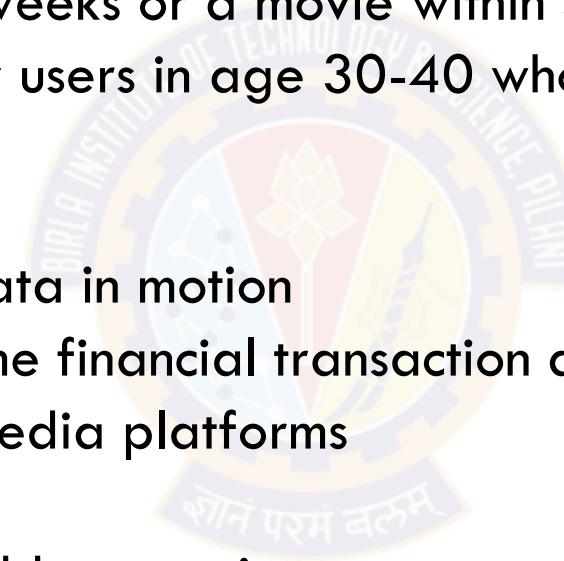
- (i) on-demand service
- (ii) resource pooling,
- (iii) scalability,
- (iv) accountability, and
- (v) broad network access.



Elastic scaling (scaling up and scaling down) by dynamic provisioning based on computational need

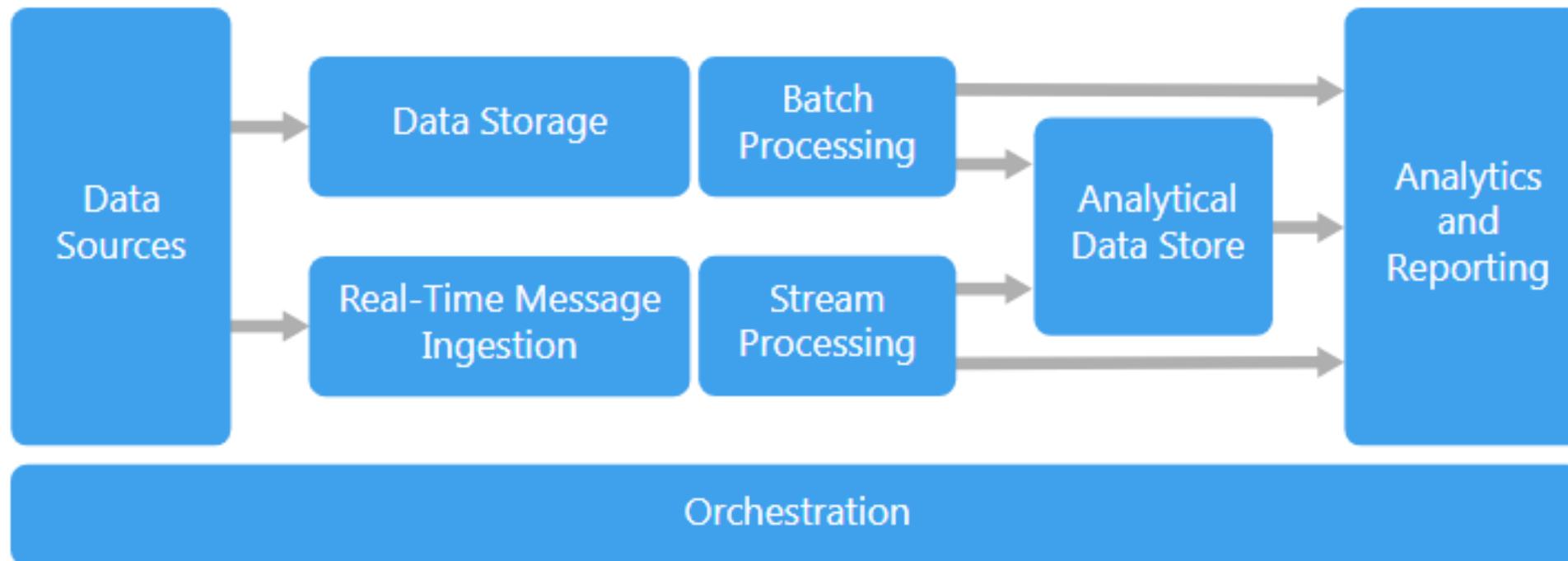
# Types of Big Data solutions

1. Batch processing of big data sources at rest
  - ✓ Building ML models, statistical aggregates
  - ✓ “What percentage of users in US last year watched shows starring Kevin Spacey and completed a season within 4 weeks or a movie within 4 hours”
  - ✓ “Predict number of US family users in age 30-40 who will buy a Kelloggs cereal if they purchase milk”
2. Real-time processing of big data in motion
  - ✓ Fraud detection from real-time financial transaction data
  - ✓ Detect fake news on social media platforms
3. Interactive exploration with ad-hoc queries
  - ✓ Which region and product has least sales growth in last quarter



# Big data architecture style

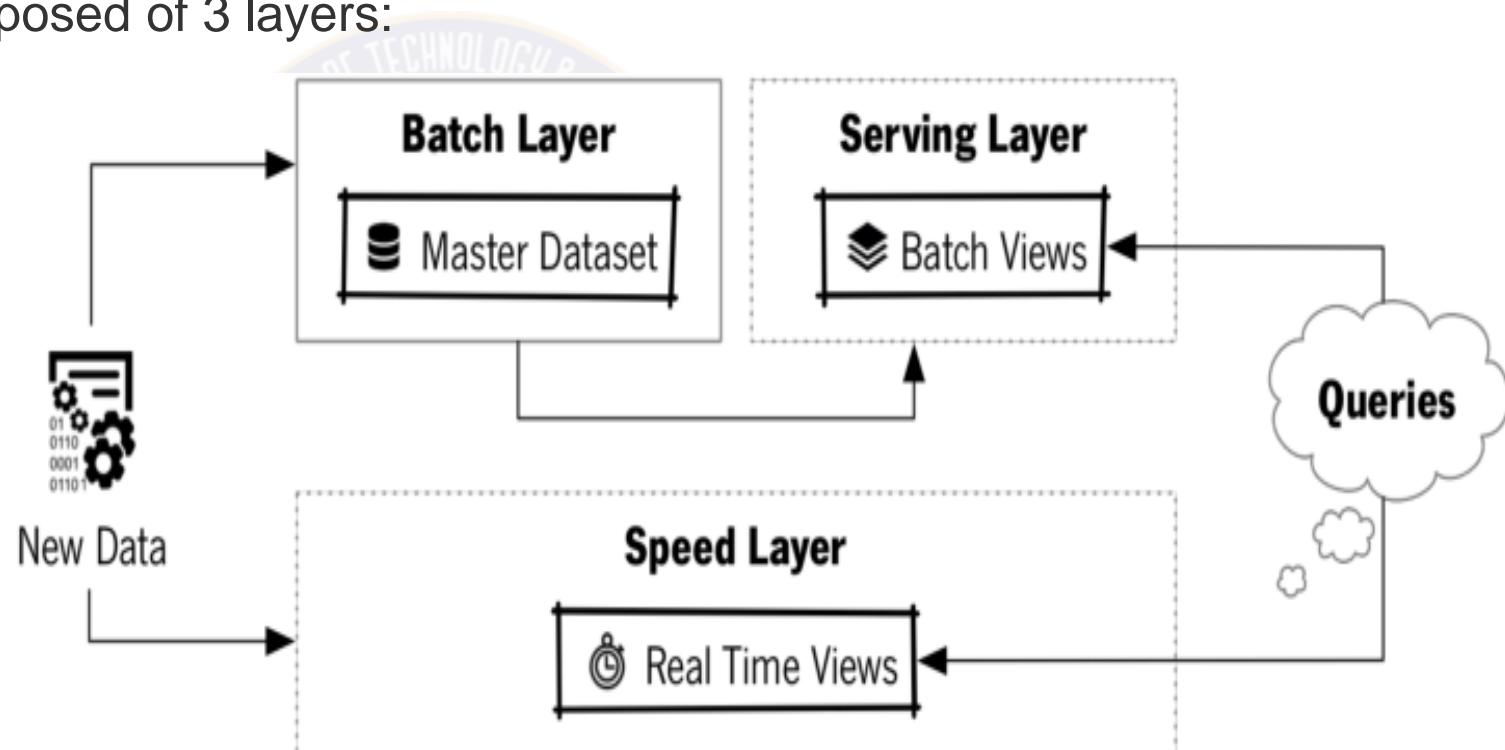
- Designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



[Source : Microsoft Big Data Architecture](#)

# Lambda Architecture

- Lambda architecture is a way of processing massive quantities of data (i.e. “Big Data”) that provides access to batch-processing and stream-processing methods with a hybrid approach .
- Lambda architecture is used to solve the problem of computing arbitrary functions in real time.
- The lambda architecture is composed of 3 layers:
  1. Batch Layer
  2. Serving Layer
  3. Speed Layer(Stream Layer)



Source: <https://www.databricks.com/glossary/lambda-architecture>

Source: <http://vda-lab.github.io/2019/10/lambda-architecture>

# Big Data Systems Components (1)

## 1. Data sources

- ✓ One or more data sources like databases, docs, files, IoT devices, images, video etc.

## 2. Data Storage

- ✓ Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats.
- ✓ Data can also be stored in key-value stores.

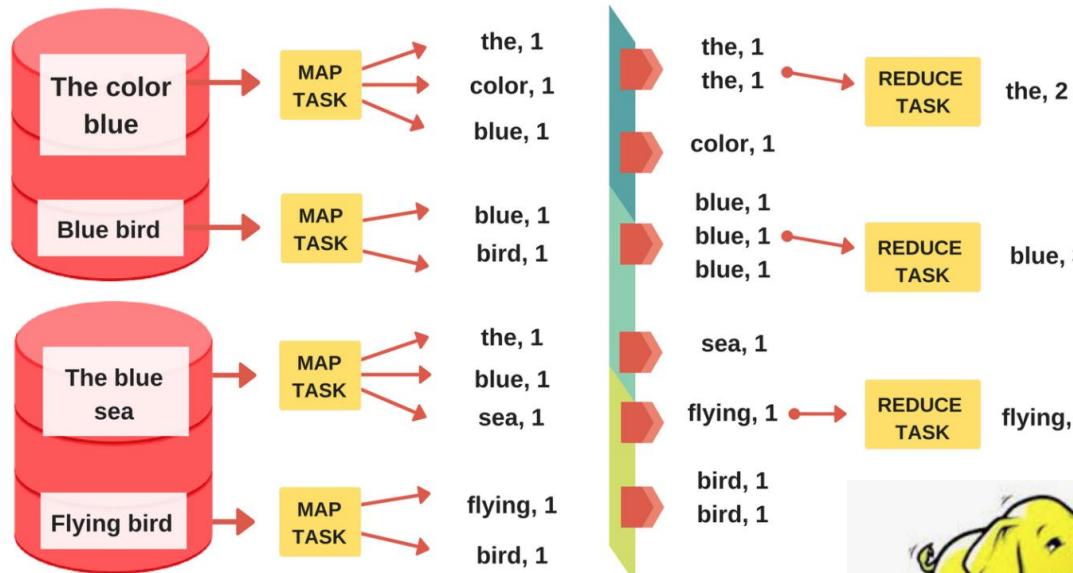


e.g. social data



e.g. medical images

# Big Data Systems Components (2)



e.g. search scans on unindexed docs



e.g. credit card transactions for fraud detection

## 3. Batch processing

- ✓ Process data files using long-running parallel batch jobs to filter, sort, aggregate or prepare the data for analysis.
- ✓ Usually these jobs involve reading source files, processing them, and writing the output to new files.

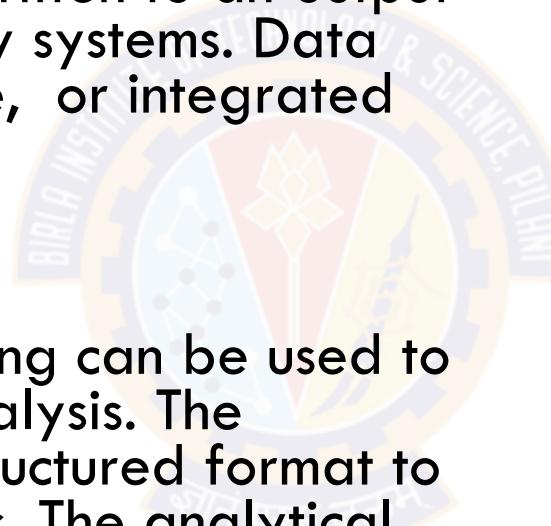
## 4. Real-time message ingestion

- ✓ Capture data from real-time sources and integrate with stream processing. Typically these are in-memory systems with optional storage backup for resiliency.

# Big Data Systems Components (3)

## 5. Stream processing

Real-time in-memory filtering, aggregating or preparing the data for further analysis. The processed stream data is then written to an output sink. These are mainly in-memory systems. Data can be written to files, database, or integrated with an API.



## 6. Analytical data store

Real-time or batch processing can be used to prepare the data for further analysis. The processed data is stored in a structured format to be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse or BigData warehouse like Hive. There may be also NoSQL stores such as MongoDB, HBase.



e.g. fraud detection logic



e.g. financial transaction history across clients for spend analysis

# Big Data Systems Components (4)



e.g. weekly management report



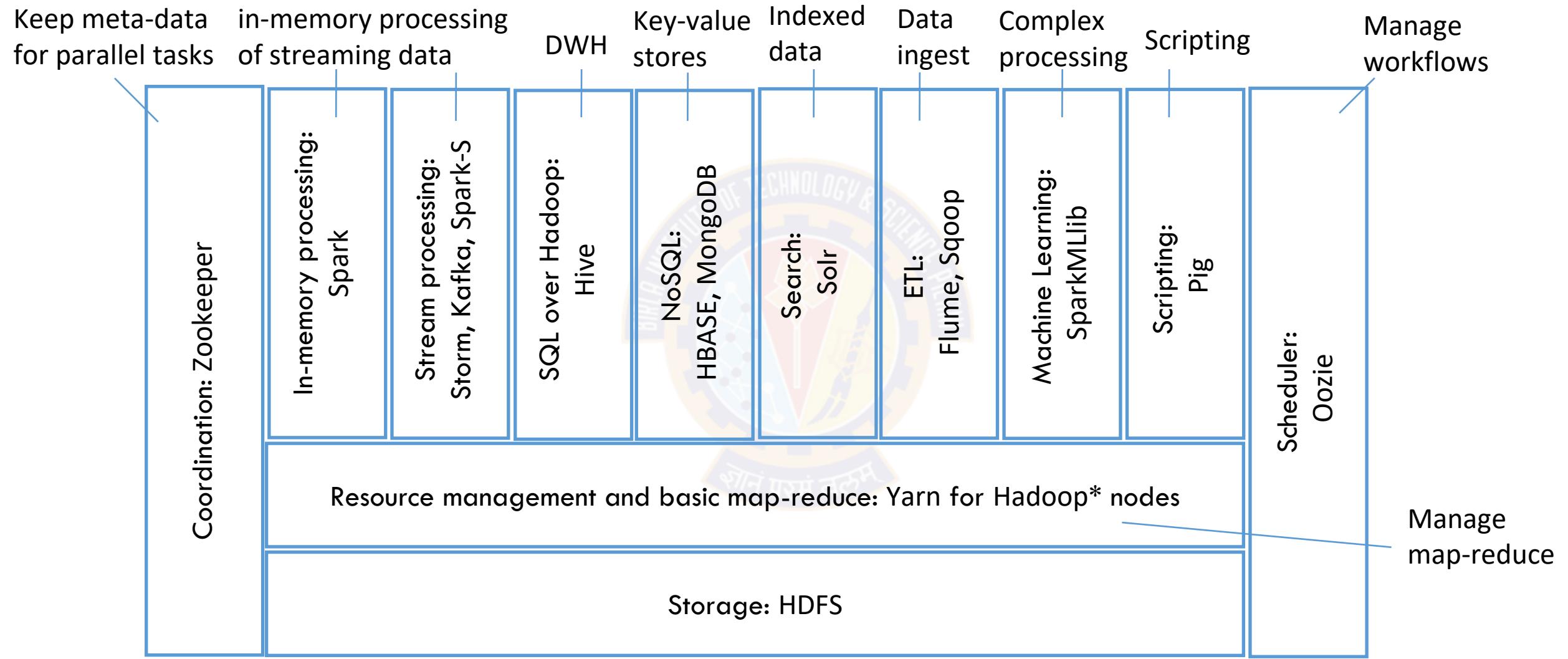
## 7. Analysis and reporting

The goal of most big data solutions is to provide insights into the data through analysis and reporting. These can be various OLAP, search and reporting tools.

## 8. Orchestration and ETL

Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory, Apache Oozie or Sqoop.

# Technology Ecosystem (showing mostly Apache projects)



\* nodes run **map-reduce** jobs (more on this later)

\*\* we'll cover technologies in **bold**



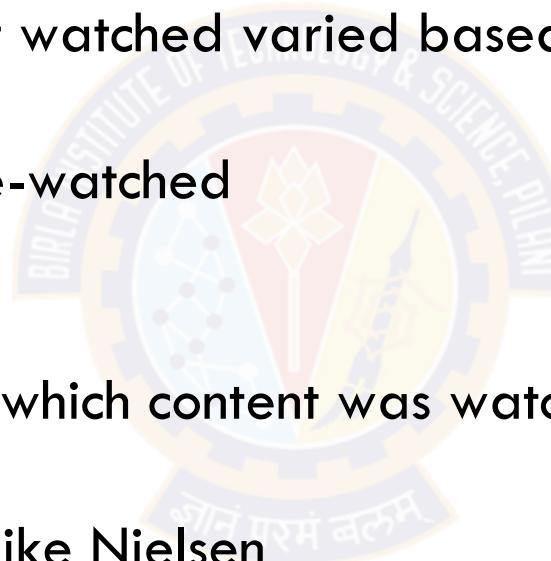
# Case Study: Netflix

How Netflix uses Big Data for exponential user growth and retention

(reference: <https://www.clickz.com/how-netflix-uses-big-data-content/228201/>)

# Data collected from 150+ million users

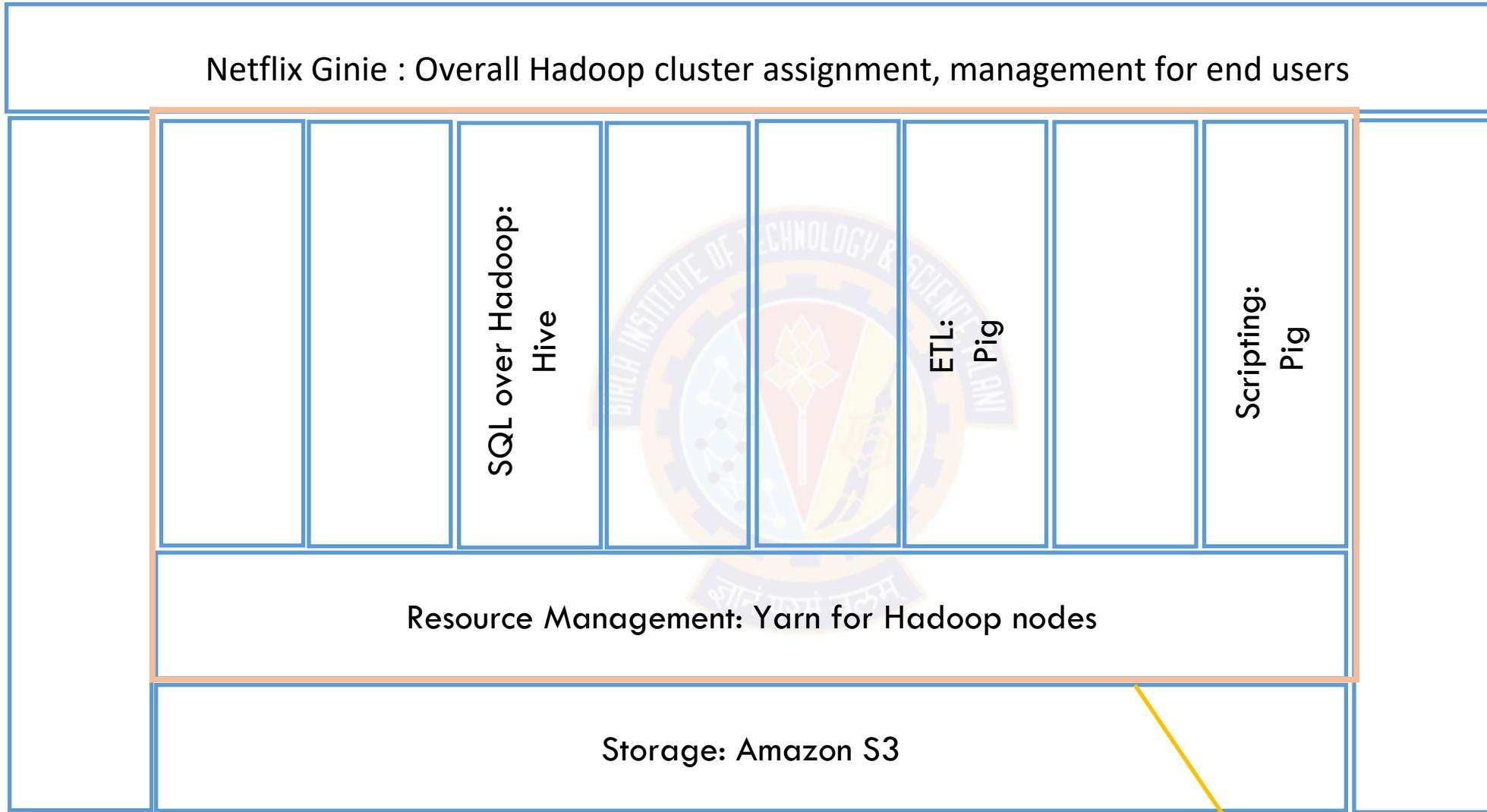
- Date content watched
- The device on which the content was watched
- How the nature of the content watched varied based on the device
- Searches on its platform
- Portions of content that got re-watched
- Whether content was paused
- User location data
- Time of the day and week in which content was watched and how it influences the kind of content watched
- Metadata from third parties like Nielsen
- Social media data from Facebook and Twitter



# Recommendation System

- Personalized context ranking
- Content promotion influenced by personal interests and not just what's popular or needs promotion
- Intelligent prioritization of viewed content that users are expected to watch / re-watch - never bore the user
- What type of content is likely to be popular to decide new shows
  - ✓ E.g. “House of Cards” was contracted for 2 seasons without a pilot because analysis said that show will be popular based on analysis of historical viewership data across regions for past content by cast and crew

# Big Data System



Amazon EMR (Elastic Map Reduce) - Hadoop on Cloud

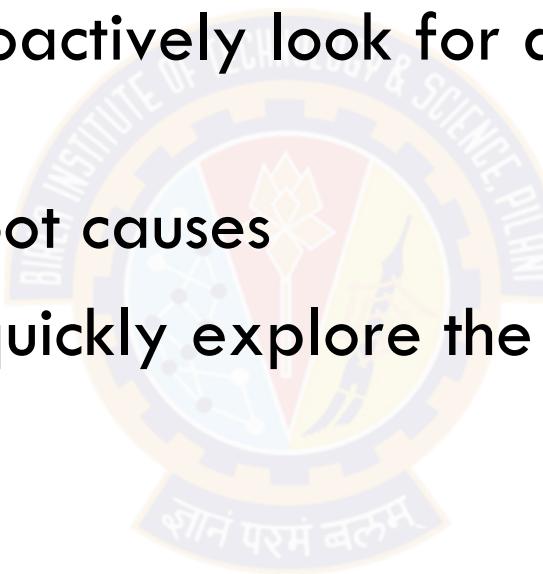


# Case Study: IT Ops

Using Big Data tools and architecture for managing IT

# IT Operations Analytics

- IT systems generate large volumes of monitoring, logging and event data
- Can we use this data to proactively look for anomalous patterns and predict an issue
- Can we localise possible root causes
- Can we help an engineer quickly explore the data to confirm the specific root cause



# IT Operations Analytics

- IT systems generate large volumes of monitoring, logging and event data
- Can we use this data to proactively look for anomalous patterns and predict an issue
- Can we localise possible symptoms and possible causes
- Can we help an engineer quickly explore the data to confirm the specific root cause



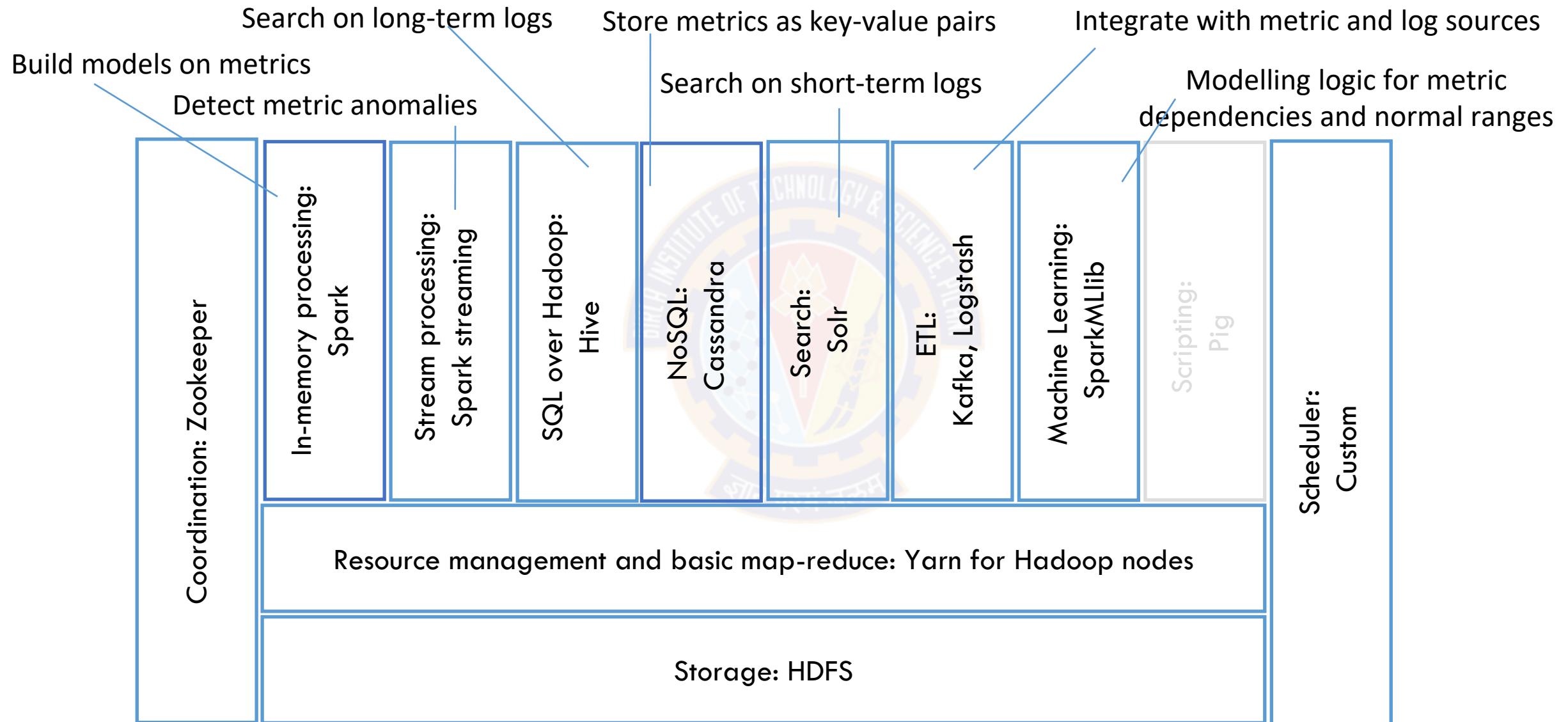
Real-time streaming analysis of metrics

- in-memory fast compute
- real-time model updates and model lookups

Interactive time-sensitive search and exploration of log and metric data

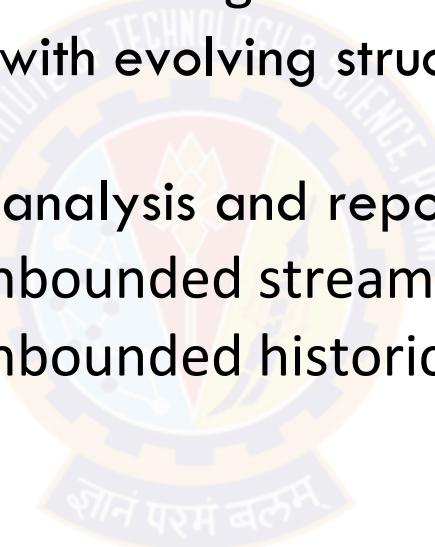
- older data may take time (has this happened earlier in last month)
- but new data should be fast (what happened few minutes back)

# Big Data Platform



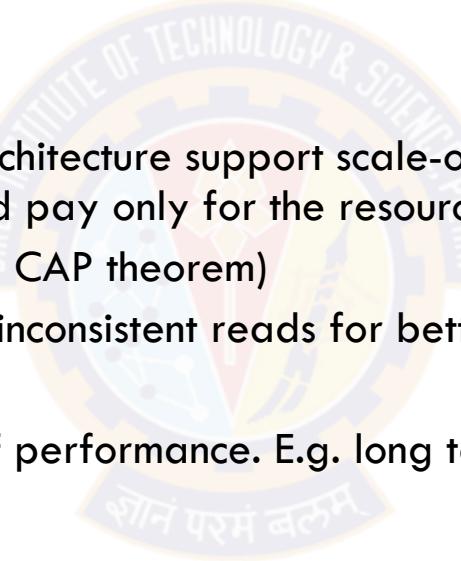
# Where will you apply this architecture style

- Consider this architecture style when you need to:
  - ✓ Store and process data in volumes too large for a traditional database
  - ✓ Handle semi-structured or data with evolving structure - e.g. demographic data with hundreds of attributes
  - ✓ Transform unstructured data for analysis and reporting
  - ✓ Capture, process, and analyze unbounded streams of data with low latency
  - ✓ Capture, process, and analyze unbounded historical data cost effectively



# Big data architecture benefits

- Technology choices
  - ✓ Variety of technology options in open source and from vendors are available
- Performance through parallelism
  - ✓ Big data solutions take advantage of data or task parallelism, enabling high-performance solutions that scale to large volumes of data.
- Elastic scale
  - ✓ All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.
- Flexibility with consistency semantics (more in CAP theorem)
  - ✓ E.g. Cassandra or MongoDB can make inconsistent reads for better scale and fault tolerance
- Good cost performance ratio
  - ✓ Ability to reduce cost at the expense of performance. E.g. long term data storage in commodity HDFS nodes.
- Interoperability with existing solutions
  - ✓ The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads. e.g. Hadoop can work with data in Amazon S3.



# Big data architecture challenges

- Complexity
  - ✓ Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes.
- Skillset
  - ✓ Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages.
- Technology maturity
  - ✓ Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release.



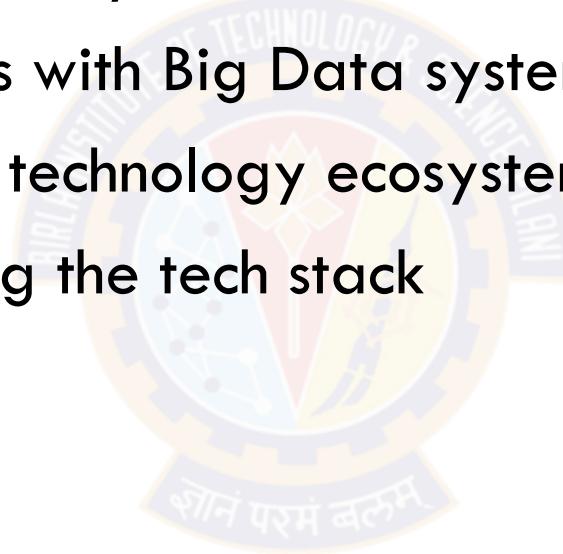
# Topics for today

- Motivation
  - ✓ Why do modern Enterprises need to work with data
  - ✓ What is Big Data and data classification
  - ✓ Scaling RDBMS
- What is a Big Data System
  - ✓ Characteristics
  - ✓ Design challenges
- Architecture
  - ✓ High level architecture of Big Data solutions
  - ✓ Technology ecosystem
  - ✓ Case studies



# Summary

- Why modern Enterprises and new age applications are data-centric
- Challenges with existing data systems
- Advantages and challenges with Big Data systems
- High level architecture and technology ecosystem
- Some real applications using the tech stack





## Next Session: Locality of Reference (LOR)



**BITS** Pilani  
Pilani|Dubai|Goa|Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 1.2 : Locality of Reference

Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)



# Locality of Reference (LoR)

Spatial and temporal locality principles for processing and storage

# Context

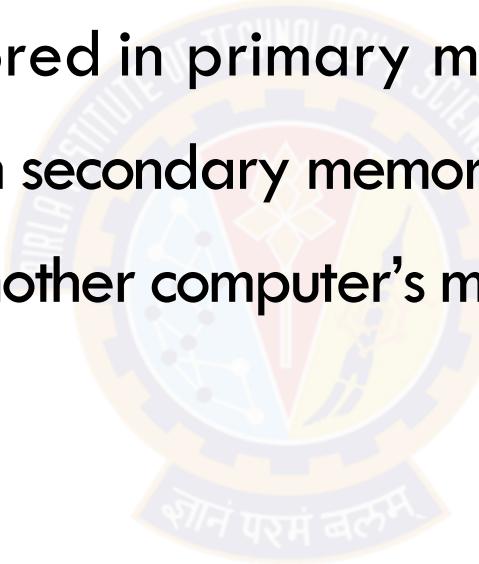
- Big Data Systems need to transport large volumes of data
  - ✓ e.g. browse and stream content on Netflix or answer analytical queries on e-commerce transaction data in Amazon
- Is there a way to reduce latency of data requests using locality of reference in the data and request origin



# Levels of storage

## Data Location – Memory vs Storage vs Network

- Computational Data is stored in primary memory aka memory
- Persistent Data is stored in secondary memory aka Storage
- Remote data access from another computer's memory or storage is done over network



# Cost of access: Memory vs. Storage vs. Network

## Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
<u>Main memory reference</u>	<u>100 ns</u>
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
<u>Disk seek</u>	<u>10,000,000 ns</u>
Read 1 MB sequentially from disk	20,000,000 ns
<u>Send packet CA-&gt;Netherlands-&gt;CA</u>	<u>150,000,000 ns</u>

Reference: <http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf>

# Memory hierarchy – motivation

- A memory hierarchy amortizes cost in computer architecture:
  - ✓ fast (and therefore costly) but small-sized memory to
  - ✓ large-sized but slow (and therefore cheap) memory

In a BigData search, recent Solr indexed data can be a cache for long term HDFS data.

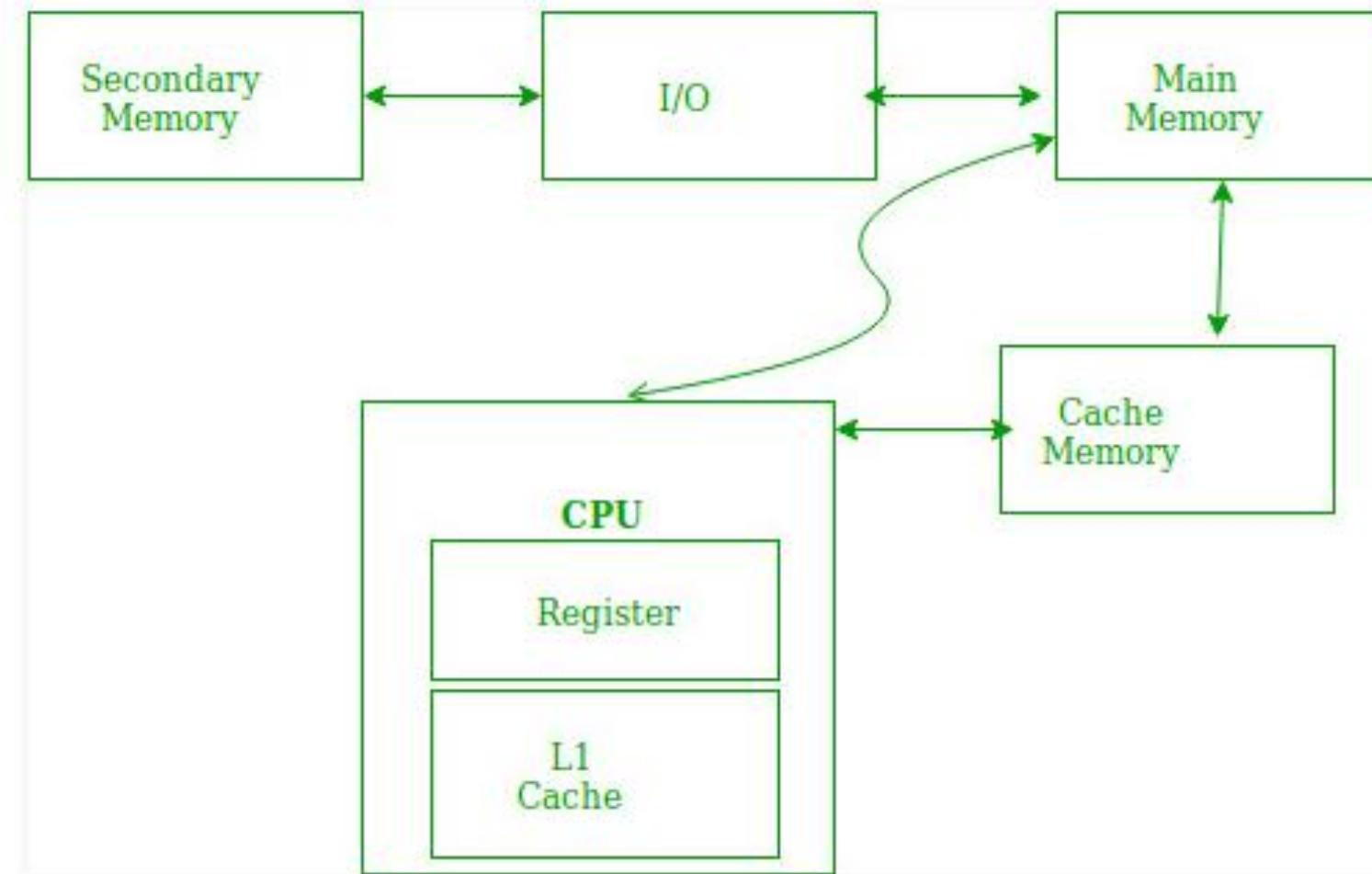
- **Classic:**



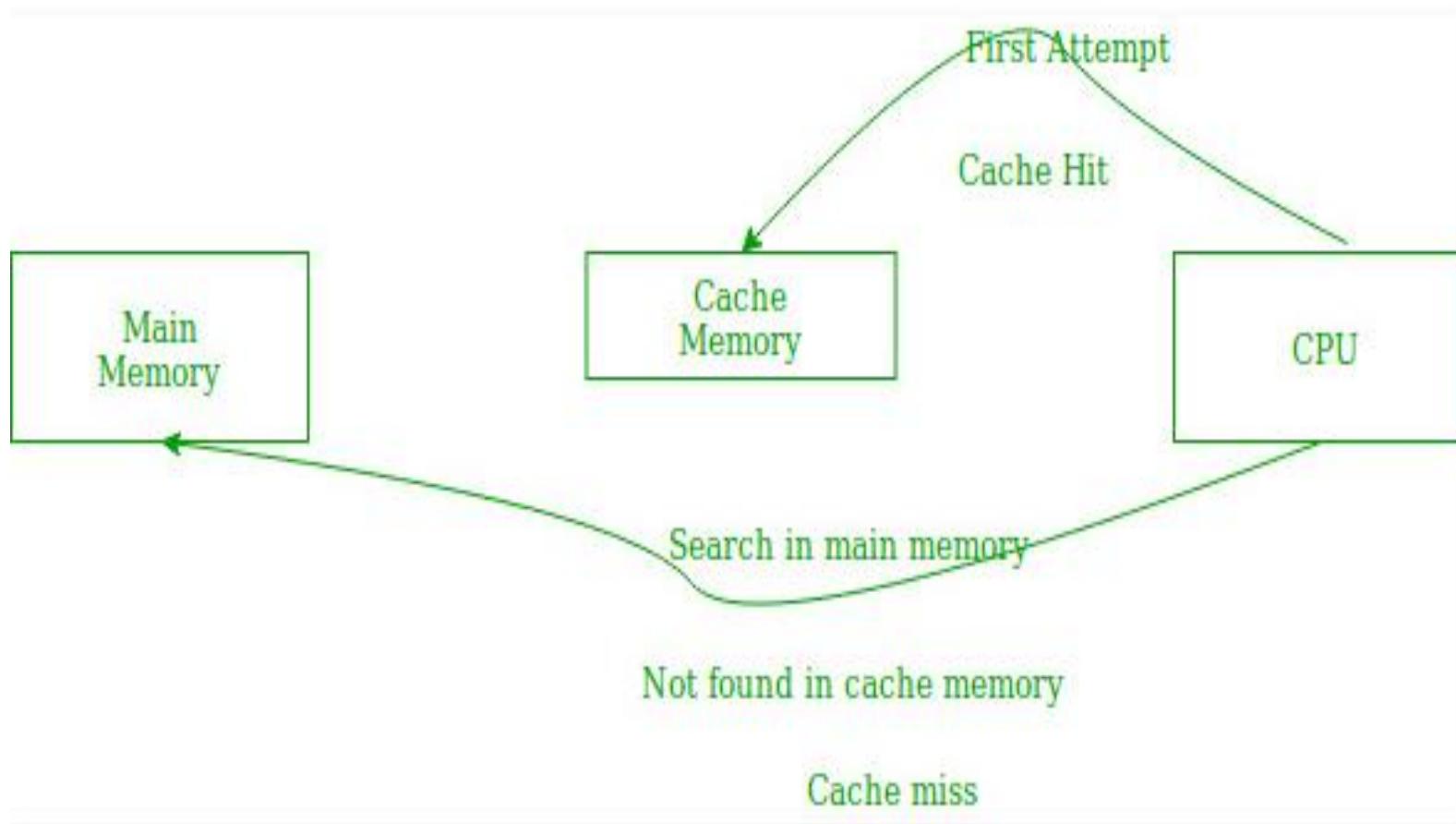
- **Modern:**



# Memory hierarchy - access



# Memory hierarchy - cache hit/miss



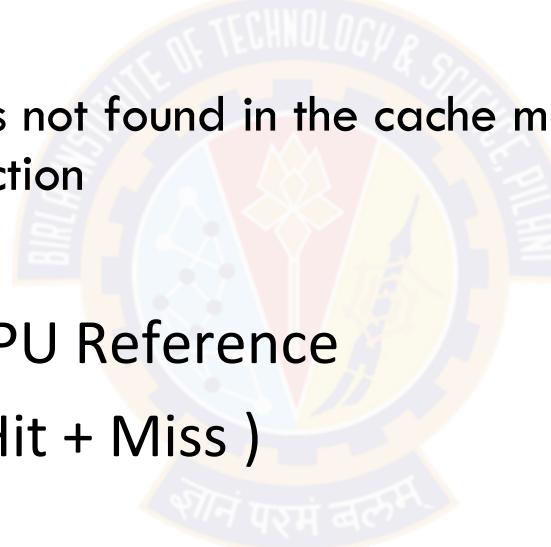
# Cache performance

## Hit Ratio - The performance of the cache

- Cache hit
  - ✓ When CPU refers to memory and find the data or instruction within the Cache Memory
- Cache miss
  - ✓ If the desired data or instruction is not found in the cache memory and CPU refers to the main memory to find that data or instruction

Hit + Miss = Total CPU Reference

Hit Ratio  $h = \text{Hit} / (\text{Hit} + \text{Miss})$



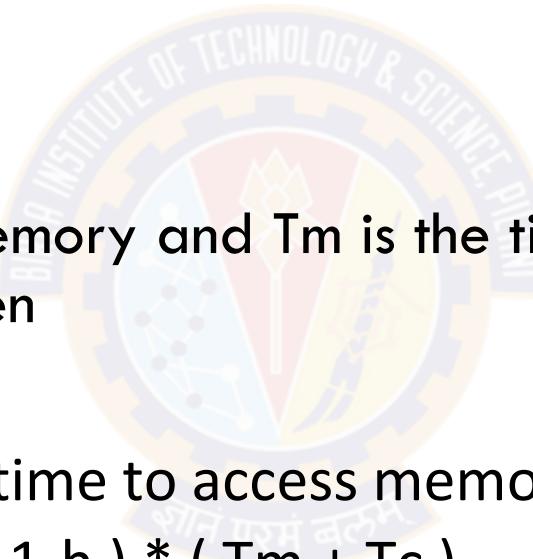
- One can generalise this to any form of cache e.g. movie request from user to nearest Netflix content cache

# Access time of memories

- Average access time of any memory system consists of two levels:
  - ✓ Cache Memory
  - ✓ Main Memory
- If  $T_c$  is time to access cache memory and  $T_m$  is the time to access main memory and  $h$  is the cache hit ratio, then

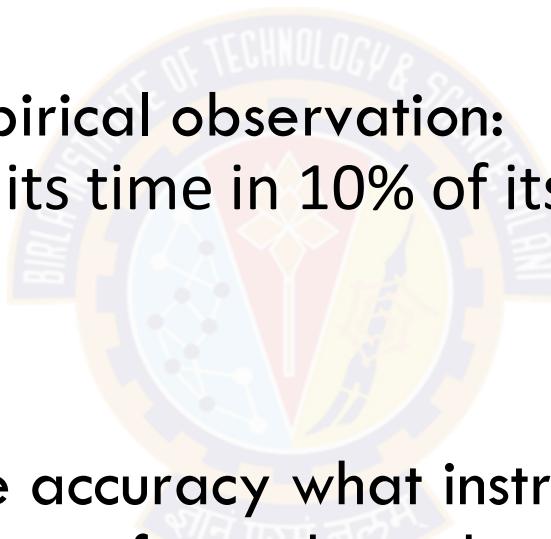
$T_{avg}$  = Average time to access memory

$$T_{avg} = h * T_c + (1-h) * (T_m + T_c)$$



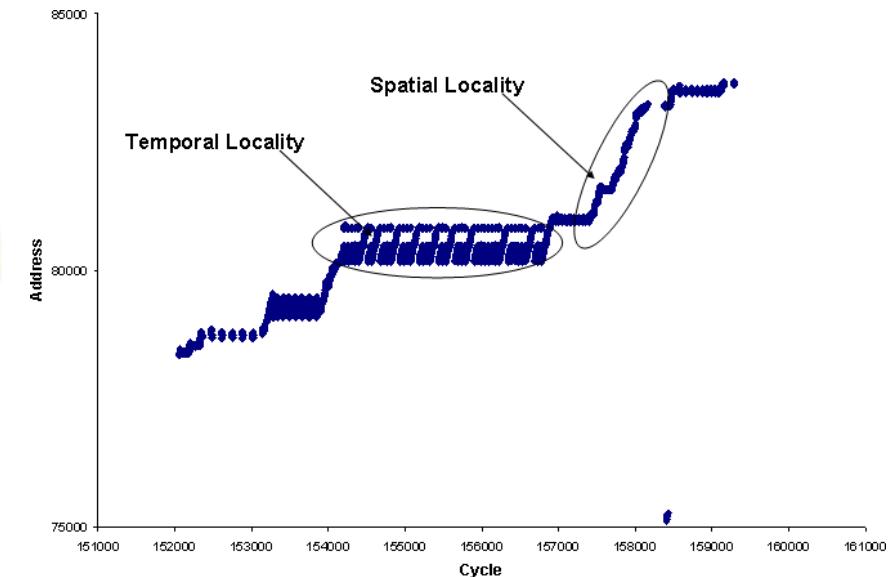
# Data reference empirical evidence

- Programs tend to reuse data and instructions they have used recently.
- 90/10 rule comes from empirical observation:  
"A program spends 90% of its time in 10% of its code"
- Implication:
  - ✓ Predict with reasonable accuracy what instructions and data a program will use in the near future based on the recent past



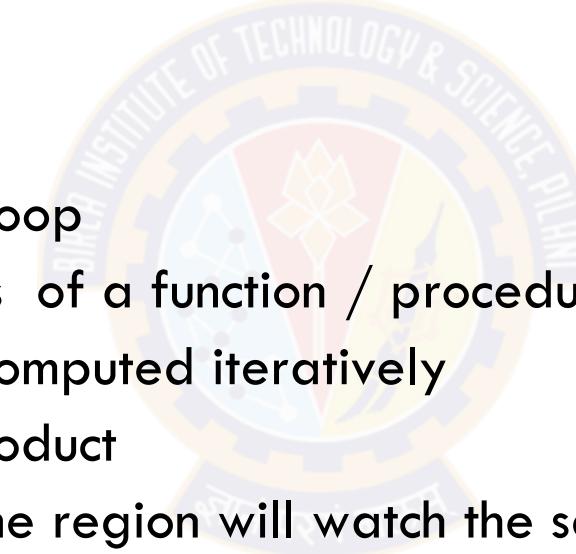
# Principle of Locality of Reference (LoR)

1. The locus of data access – and hence that of memory references – is small at any point during program execution
2. It is the tendency of a processor to access the same set of memory locations repetitively over a short period of time
3. Locality is a type of predictable behavior that occurs in computer systems
4. Systems that exhibit strong locality of reference are great candidates for performance optimization through the use of techniques such as the caching, prefetching for memory and advanced branch predictors at the pipelining stage of a processor core.



# Locality of Reference - Temporal locality

- Data that is accessed (at a point in program execution) is likely to be accessed again in the near future:  
i.e. data is likely to be repeatedly accessed in a short span of time during execution
- Examples
  1. Instructions in the body of a loop
  2. Parameters / Local variables of a function / procedure
  3. Data (or a variable) that is computed iteratively
    - e.g. a cumulative sum or product
  4. Another user in Netflix in same region will watch the same episode
  5. A recent social media post will be viewed soon by other users



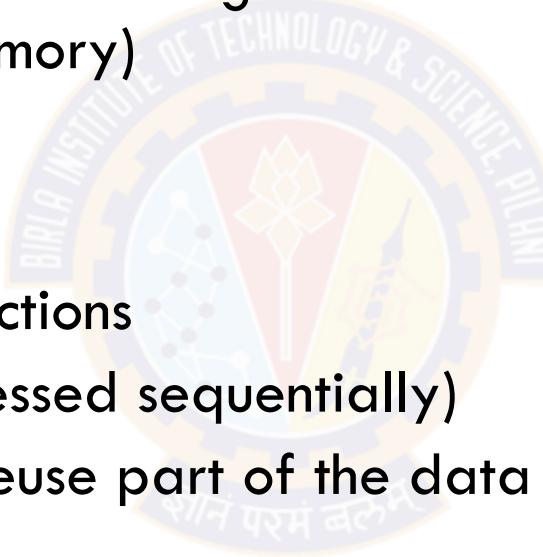
```
void swap(int x,  
         int y)  
{  
    t = x;  
    x = y;  
    y = t;  
}
```

# Locality of Reference - Spatial locality

- Data accessed (at a point in program execution) is likely located adjacent to data that is to be accessed in near future:
  - ✓ data accessed in a short span during execution is likely to be within a small region (in memory)

- Examples

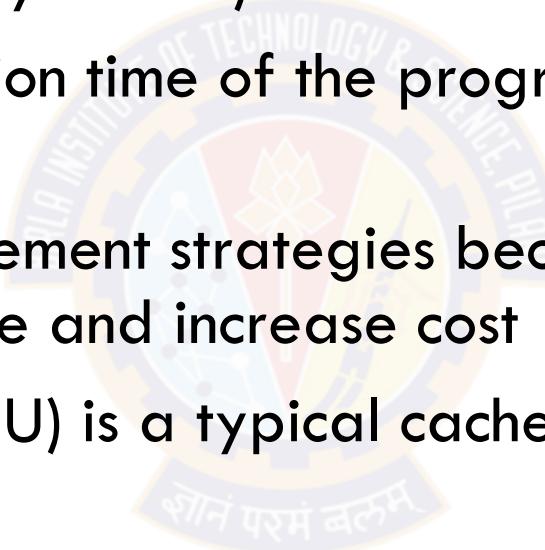
1. A linear sequence of instructions
2. Elements of an Array (accessed sequentially)
3. Another user's query will reuse part of the data file brought in for current user's query



```
int sum_array_rows(int  
marks[8]) {  
    int i, sum = 0;  
    for (i = 0; i < 8; i++)  
        sum = sum +  
    marks[i];  
    return sum;  
}
```

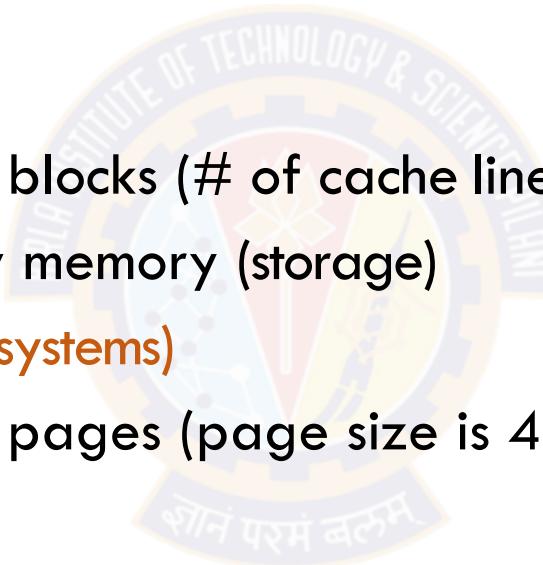
# Memory hierarchy and locality of reference (1)

- A memory hierarchy is effective only due to locality exhibited by programs (and the data they access)
- Longer the range of execution time of the program, larger is the locus of data accesses
- Need to have cache replacement strategies because increasing cache size will also increase access time and increase cost
  - ✓ Least Recently Used (LRU) is a typical cache entry replacement strategy
    - This is an example of temporal locality



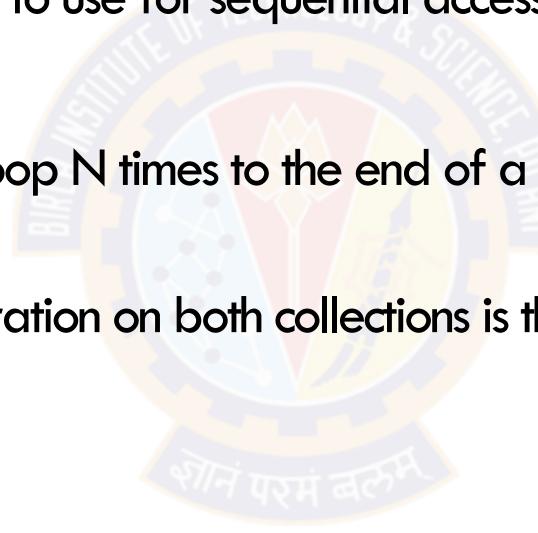
# Memory hierarchy and locality of reference (2)

- LOR leads to memory hierarchy at two main interface levels:
  - ✓ Processor - Main memory
    - **Introduction of caches**
    - Unit of data transfer is blocks (# of cache lines)
  - ✓ Main memory - Secondary memory (storage)
    - **Virtual memory (paging systems)**
    - Unit of data transfer is pages (page size is 4 or 8 KB)
- Fetching larger chunks of data enables spatial locality



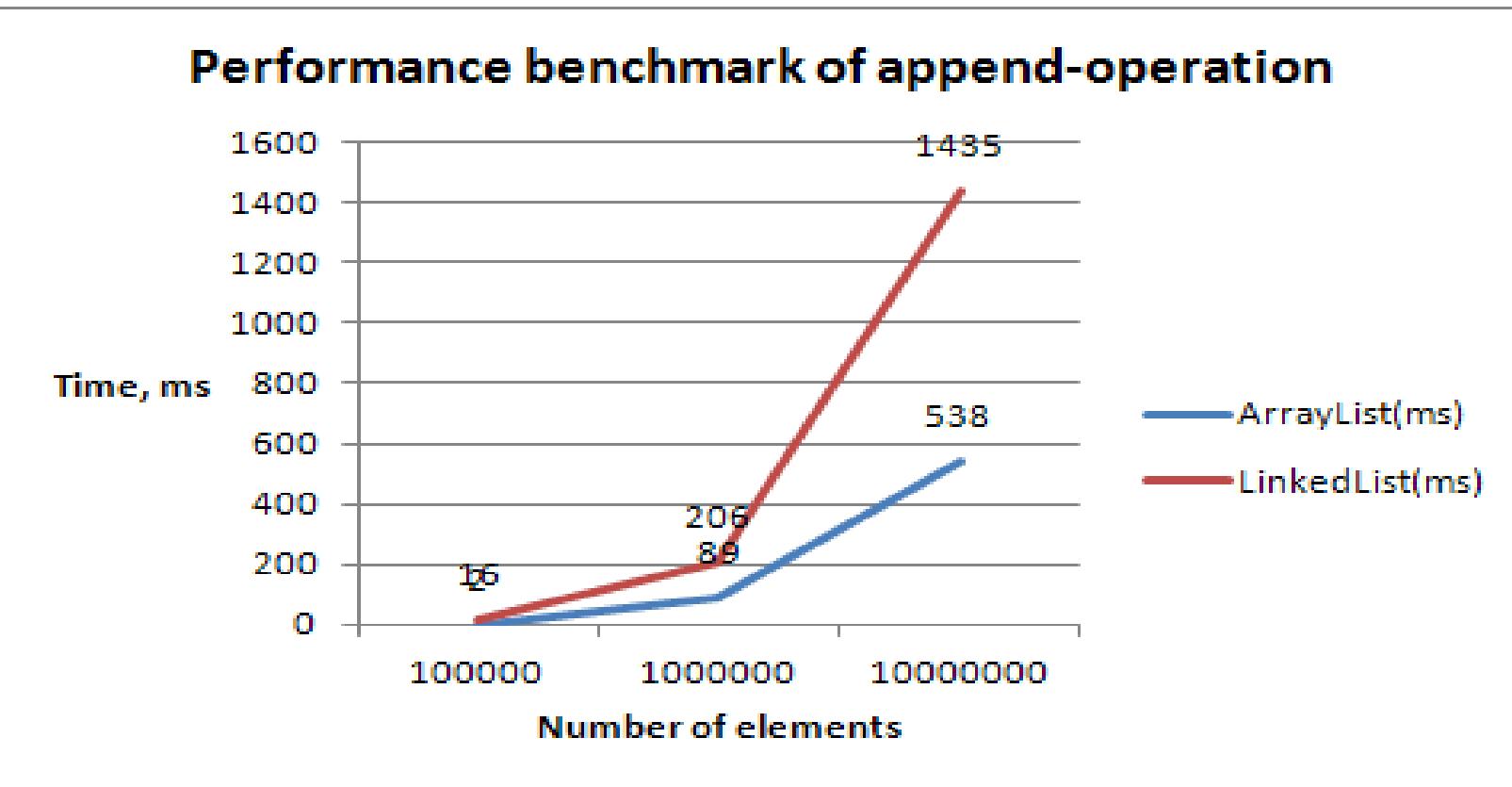
# LoR and data structure choices

- In Java, the following 2 classes are available in util package
  - ✓ `LinkedList<Integer>` QuizMarks – a dynamic list of objects
  - ✓ `ArrayList<Integer>` QuizMarks - a dynamic array of objects
  - ✓ Which is a better data structure to use for sequential access performance ?
- Test
  - ✓ Append a single element in a loop N times to the end of a `LinkedList`. Repeat with `ArrayList`.
  - ✓ Take average for 100 runs.
  - ✓ The time complexity of the operation on both collections is the same.
- Which one works faster ?



<https://dzone.com/articles/performance-of-array-vs-linked-list-on-modern-comp#:~:text=But%20when%20we%20need%20to,have%20better%20performance%20than%20array>.

# LoR and data structure choices (2)

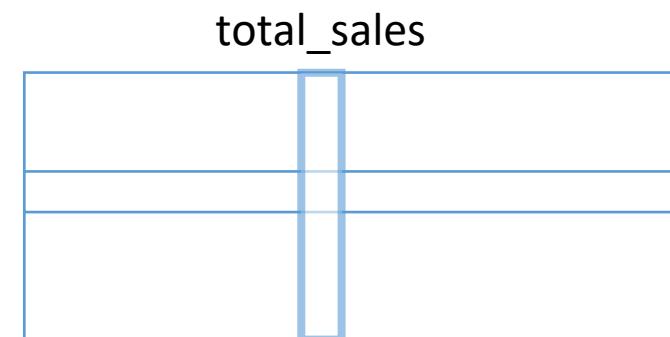


- Analysis - sequential access in arrays is faster than on linked lists on many machines, because they have a very good spacial locality of reference and thus make good use of data caching.

# Big Data: Storage organisation matters

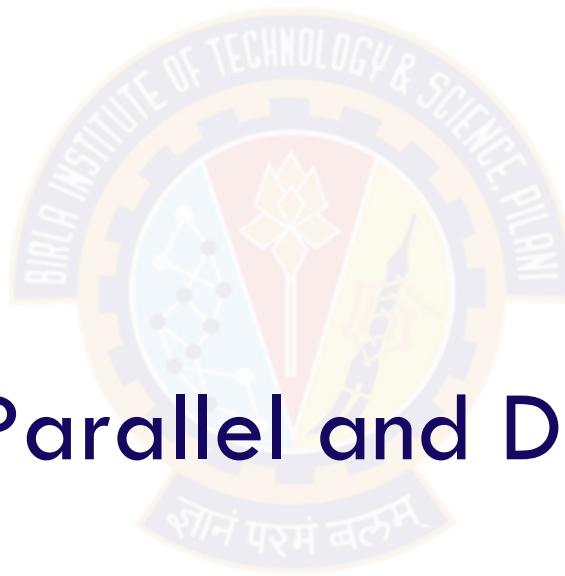
## Example

- We need to build a prediction model of sales for various regions
- There are many attributes for a region and “total sales” is the metric used
- Suppose the database is “columnar” (Column major data organisation)
  - ✓ Will exhibit high spatial locality and hit rate because data blocks will fetch blocks of total sales column and not other unnecessary columns
  - ✓ Will improve speed of modelling logic
- Columnar storage is common in most Big Data Systems to run analysis and queries that focus on specific attributes at a time for searching, aggregating, modelling etc.



# Distributed Cache in Hadoop

- Copy small files from HDFS to local disks of worker nodes
- Saves network bandwidth during run, since these files are locally available
- These files get tagged as localized
- Hadoop NodeManager maintains a count of number of tasks using the localized files
- The file is selected for deletion only when there are no tasks using it
- The file gets deleted when the cache occupancy exceeds a specified limit
- Hadoop deletes localized files to make room for other files getting used
- Files selected for deletion using least recently used algorithm
- One can change the cache size by setting the property  
**yarn.nodemanager.localizer.cache.target-size-mb** (default 10240)



## Next Session: Parallel and Distributed Processing



**BITS** Pilani

Pilani-Dubai-Goa-Hyderabad

# **.DSECL ZG 522: Big Data Systems**

## **.Session 1A: Transformations of Applications and Database Technologies**

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Evolution of Applications

---

- Online systems of 1970s were designed for direct human interaction using direct attached terminals
- Interactive software has undergone fundamental changes over last 50 years
- They **evolved** into today's web and mobile applications
- Now, these systems need to handle millions of concurrent interactive users
- The architecture of software systems has also transformed drastically

# Interactive Software - Transformations

Category	Year 1975 Online Applications	Year 2011 Interactive Web Applications
Users	2000 online users is the end point	2000 online users is the starting point
	Static User Population	Dynamic User Population
Applications	Business Process Automation	Business Process Innovation
	Highly structured data records	Structured, Semi-structured and Unstructured data
Infrastructure	Data networking in its infancy	Universal high-speed data networking
	Centralized computing (Mainframes and mini-computers)	Distributed and Parallel computing ( Clusters, Multi-Core Processors)
	Memory scarce and expensive	Memory Plentiful and cheap

# RDBMS and Web applications

---

- Most enterprise solutions have RDBMS back-end
- Very little change to RDBMS between 1980 and 2000
- Many application servers, one database
  - Response slows down when DB is concurrently accessed by applications and Analytics tools
  - Response of SQL queries depend on size of data base
  - Easy to parallelize application servers to 100s of servers
  - Harder to parallelize databases to same scale
- Most data originates from devices and volume of data grows at very high rate
- Web-based applications shows spikes in usage
  - ❑ Especially true for public-facing e-Commerce sites
- RDBMS becomes a point of contention

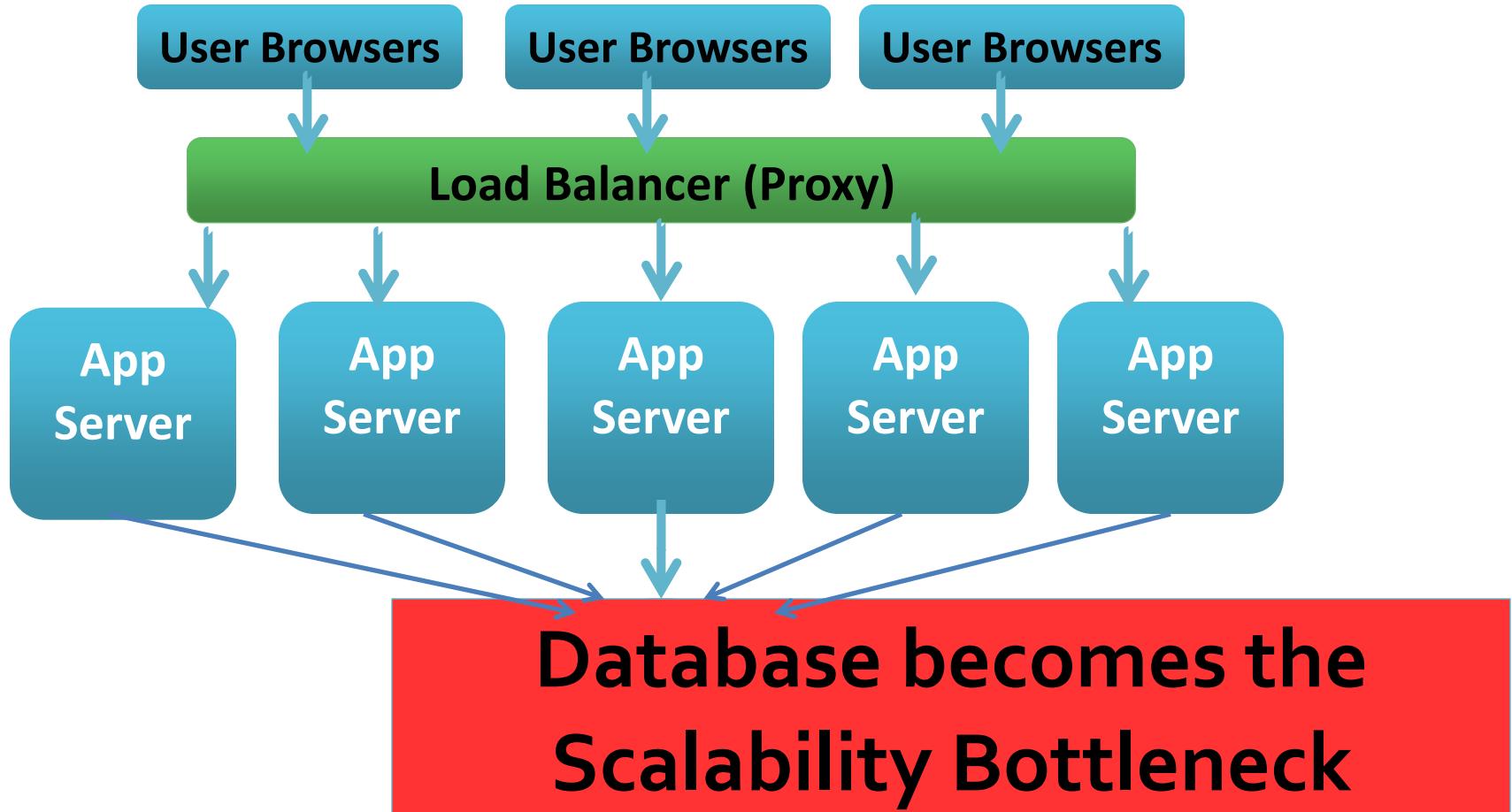
# Changes in Application Architecture

---

- Modern Web applications are built to scale out – simply add more commodity Web servers behind a load-balancer to support more number of concurrent users
- Applications are designed for high availability, fault tolerance and disaster tolerance
- By distributing the load across many servers, the system is inherently fault-tolerant, supporting continuous operations and guarantees consistent levels of performance
- As application needs change, new software can be gradually rolled out across subsets of the overall server pool.

# Scalability of Web Application

- Need to handle 1 to 1 million concurrent users
- Applications should provide uniform response



# Changes in Data used in Web applications

---

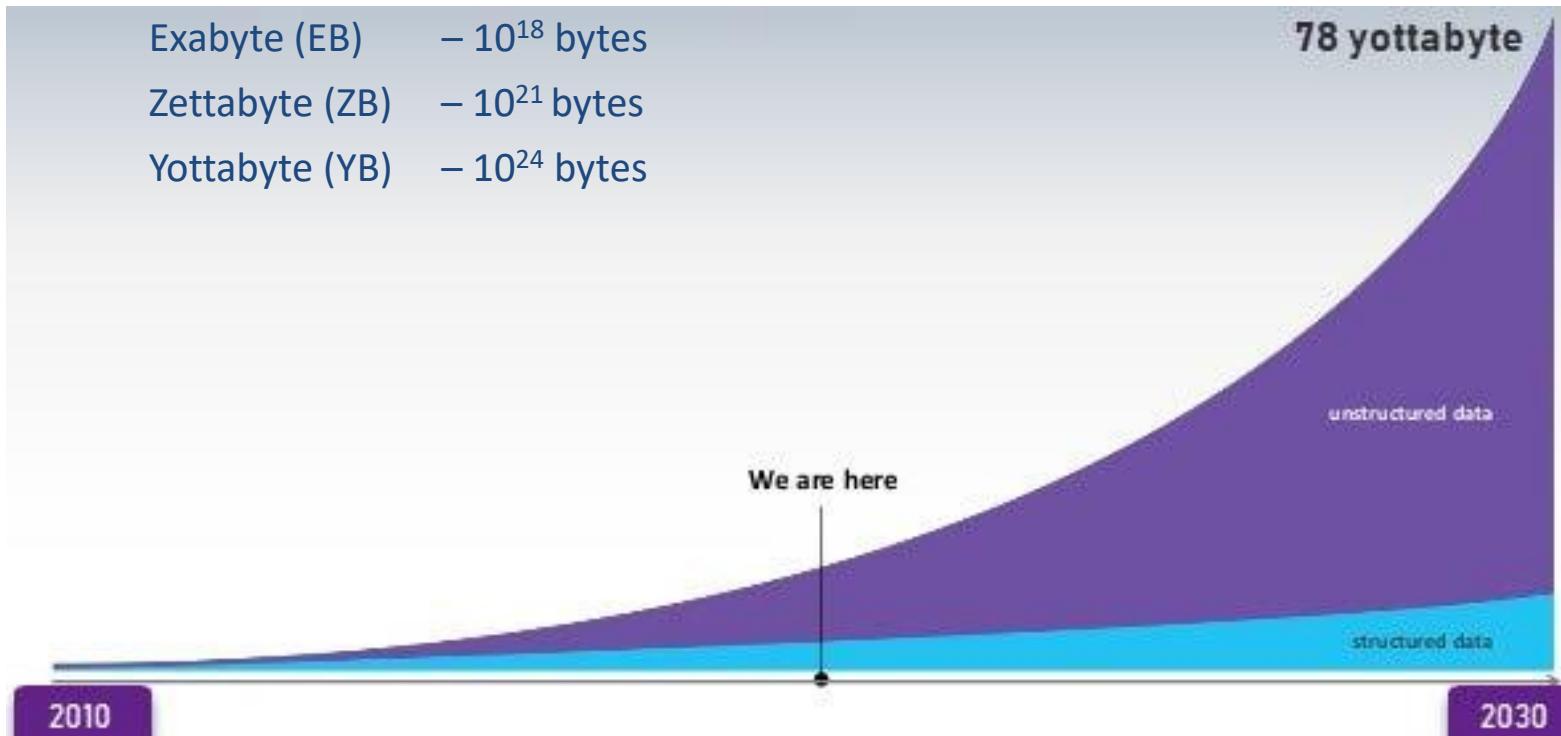
.Web applications generate lot of temporary data that do not really belong to the main structured data store. Eg:

- shopping carts
- retained searches
- site personalization
- incomplete user questionnaires.
- Data set consists of large quantities of unstructured data in the form of Text, Images, Videos etc.
- Binary large objects in RDBMS (BLOB, CLOB) cannot handle this properly.
- Local data transactions that do not have to be very durable.  
Eg: – "liking" items on website
- Need to run queries against data that do not involve simple hierarchical relations  
Eg: "all people in a social network who have not purchased book this year "

# Growth of Unstructured data

---

Kilobyte (KB)	- $10^3$ bytes
Megabyte (MB)	- $10^6$ bytes
Gigabyte (GB)	- $10^9$ bytes
Terabyte (TB)	- $10^{12}$ bytes
Petabyte (PB)	- $10^{15}$ bytes
Exabyte (EB)	- $10^{18}$ bytes
Zettabyte (ZB)	- $10^{21}$ bytes
Yottabyte (YB)	- $10^{24}$ bytes



# RDBMS is optimized for space- Not for speed

---

- Normalization removes data duplication and ensures data consistency
- RDBMS schemas are highly normalized to minimize the data storage and to speed up inserts, update and deletes
- High degree of normalization is a disadvantage when it comes to retrieving data, as multiple tables may have to be joined to get all the desired information
- Creating these joins and reading from multiple tables can have a severe impact on performance, as multiple reads to disk may be required
- Analytical queries need to access large portions of the whole database, resulting in long run times

# Fixed Table - No flexibility

---

- RDBMS Tables are designed and fixed once for all
- Number of columns in a table cannot be changed without stopping a running application
- Schema definition of tables cannot be changed on the fly
- Any change to the table definition involves recreation of the table lasting for several hours/days
- Growth of number of rows in a table is not unlimited.
- Time taken for processing queries varies with size of table
- Application performance degrades as the number of rows in a table increase (even with indexing)
- For example, aggregation of values in a table of 1 billion entries may take hours together

# RDBMS Bottleneck in Application scalability

---

- RDBMS engines are monolithic software systems
  - Originally designed to run on single CPU systems
  - Not core aware – not fully multi-threaded to take advantage of all cores
- For more performance, upgrade servers - Enjoy free performance lunch
- Upgrading a server is an exercise that requires application downtime
- CPU clock speed has hit the thermal barrier
- Processors moving to multi-core
- Multi-core adaptation needs software redesign (No free lunch)
- Given the relatively unpredictable user growth rate of modern software systems, there is either over or under provisioning of resources
- Evolution of In Memory Data Grids and NoSQL helps in overcoming the limitations of traditional RDBMS used in modern web applications

---

# How Data base technology is changing ?

# How Database Technology changes

---

- Relational database technology, invented in the 1980s are still in widespread use in today's web applications
- Relational database technology was optimized for the applications, users and infrastructure that existed in 1980s.
- Relational database technology has not changed over 30 years and it is not able to scale out in web applications
- In response to the lack of commercially available alternatives, organizations such as Google and Amazon were, out of necessity, forced to invent new approaches to data management.
- Non-relational database technologies are a better match for the needs of modern interactive web based software systems.

# Scaling of RDBMS

---

- RDBMS can scale up (Vertical scaling) on a bigger server – Enjoys free performance lunch provided by Moore's law
- When the capacity of single server is reached, solution is to scale out and distribute the load across multiple servers
- RDBMS were designed for single CPU systems – scale out bottleneck
- This is when the complexity of relational databases starts to rub against their potential to scale
- Began to look at multi-node database solutions known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

# Scaling out RDBMS – Master/Slave

---

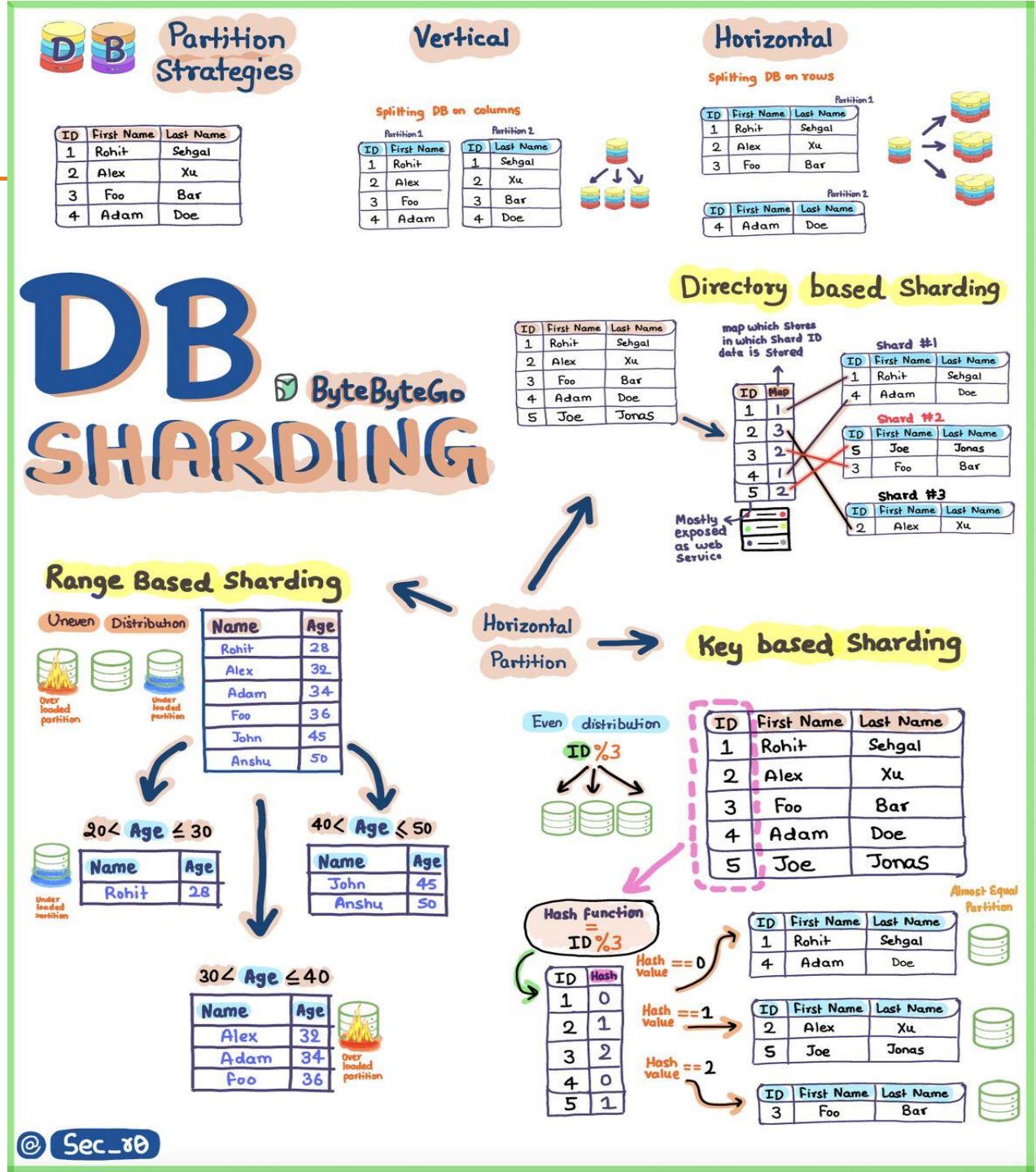
- All writes are written to the master.
- All reads performed against the replicated slave databases
- Good for mostly read, very few update applications
- Critical reads may be incorrect as writes may not have been propagated down
- Large data sets can pose problems as master needs to duplicate data to slaves

# Scaling out RDBMS sharding

---

- Data is divided into partitions (shards) hosted on multiple machines.  
E.g. Using hash or range partitioning
- Divide data amongst many cheap database (MySQL/PostgreSQL)
- Manage parallel access in the application
- Scales well for both reads and writes
- Not transparent, application needs to be partition-aware

# Database Sharding methods



# Road-blocks to RDBMS scaling

---

- RDBMS technology is a forced fit for modern interactive software systems
- RDBMS is incredibly complex internally, and changes are difficult
- Vendors of RDBMS technology have little incentive to disrupt a technology generating billions of dollars for them annually
- It requires huge investments to address the RDBMS scaling issue and find out a viable solution
- Techniques used to extend the useful scope of RDBMS technology fight symptoms but not the disease itself

# Enterprises gradually move out of RDBMS

---

- Now, we have high volume, variety data
- Volume of data keeps on increasing
- RDBMS is not capable of handling unlimited volume of data
- RDBMS scalability is limited
- In response to the lack of commercially available alternatives, organizations such as Google and Amazon were, out of necessity, forced to invent new approaches to data management.
- Non-relational database technologies are a better match for the needs of modern interactive web-based software systems.

---

End of  
Session 1A



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

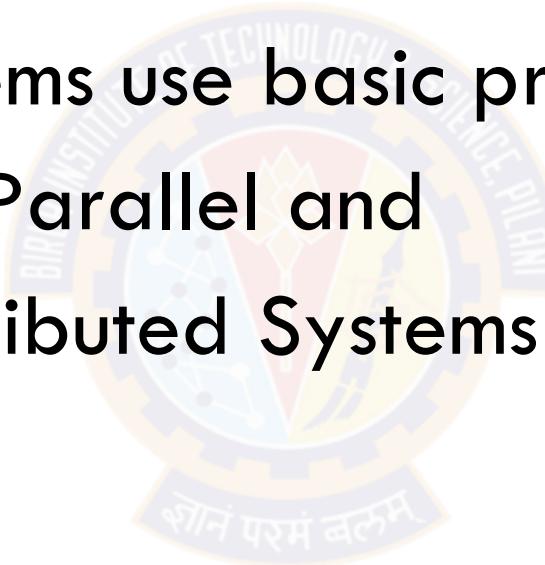
# DSECL ZG 522: Big Data Systems

## Session 2: Parallel and Distributed Systems

---

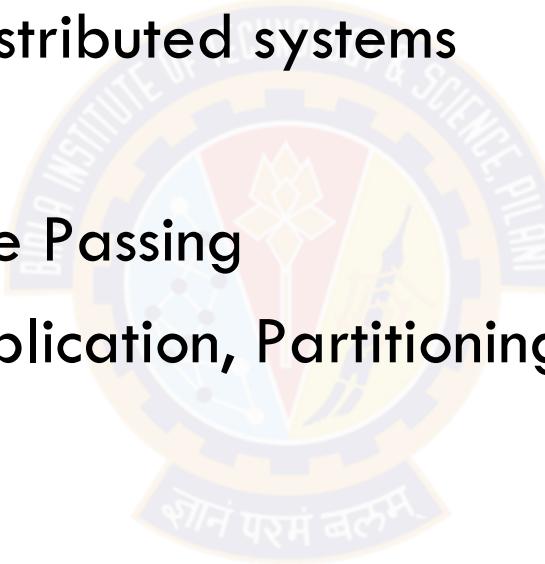
Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

Big Data Systems use basic principles of  
Parallel and  
Distributed Systems



# Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



# Roadblocks of Processor Scaling

- The Frequency Wall
  - ✓ Not much headroom
- The Power Wall
  - ✓ Dynamic and static power dissipation
- The Memory Wall
  - ✓ Gap between compute bandwidth and memory bandwidth



# Frequency wall

- More CPU Power -> More CPU hungry applications
- More Disk space -> New requirements to fill it up
- Applications enjoyed free performance benefits from latest processors
- 500 MHz -> 1 GHz -> 2 GHz -> 3.4 GHz -> 4 GHz ?  
( No need to rewrite the S/W or even new release. Sometimes rebuilding is required)

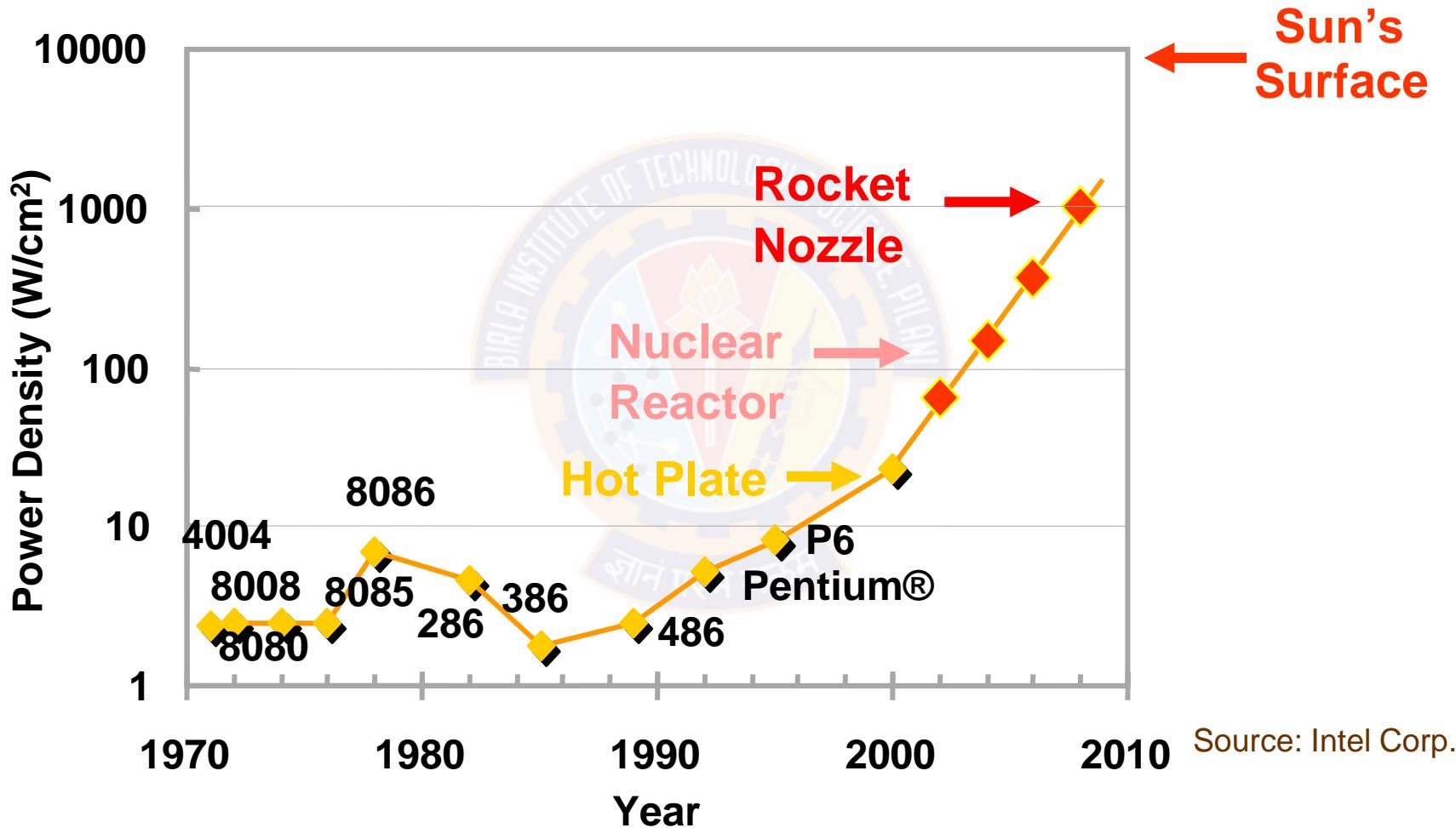
**Why we do not have a 10GHz processor now?**

Chip designers are under pressure to deliver faster CPUs that will risk changing the meaning of your program, and possibly break it, in order to make it run faster

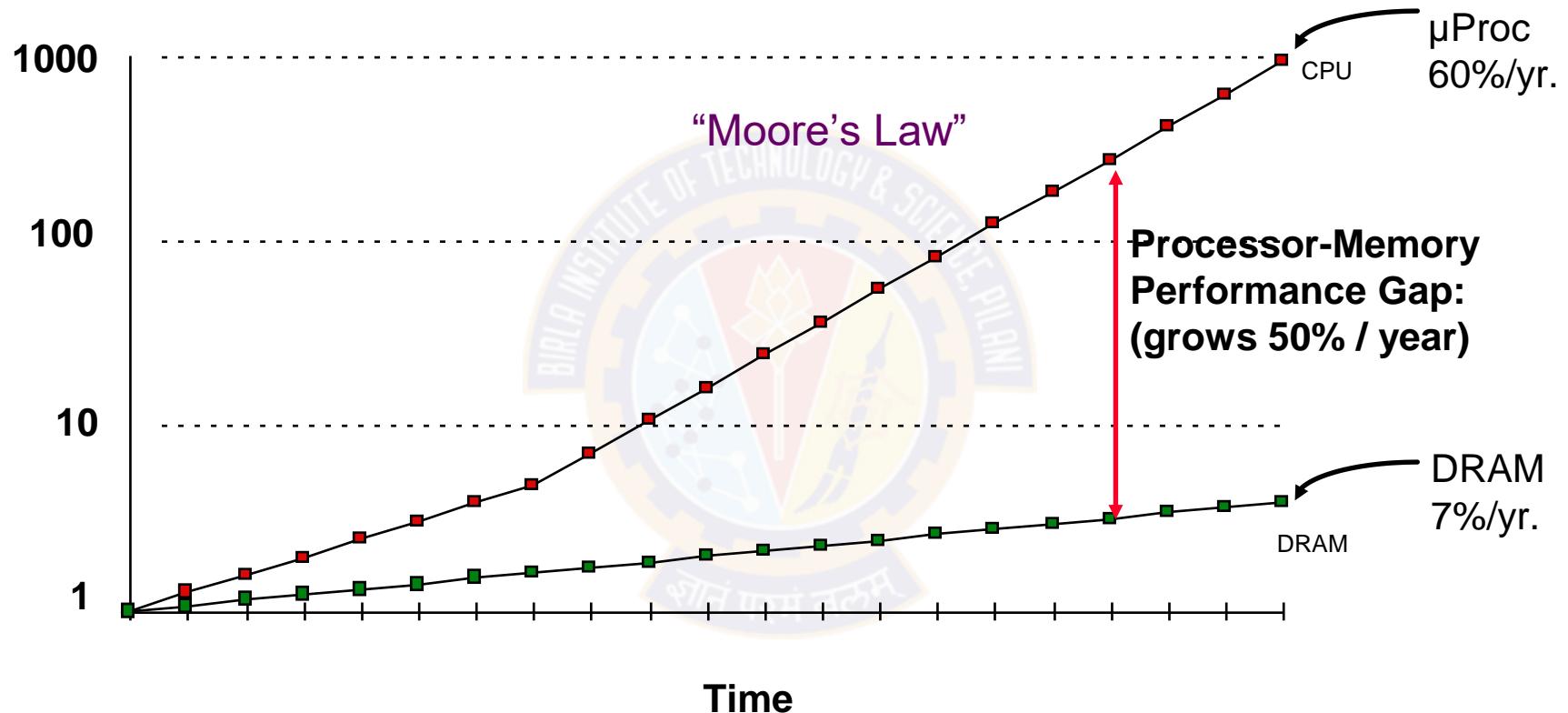
Revolutions in computing:

- 1990 - First Revolution in SW development – Object Oriented Programming
- Applications will increasingly need to be concurrent if they want to fully exploit continuing exponential multi-core CPU throughput gains
- Next Revolution in SW development – Parallel (Concurrent) Programming

# Thermal wall (CPU Power consumption)

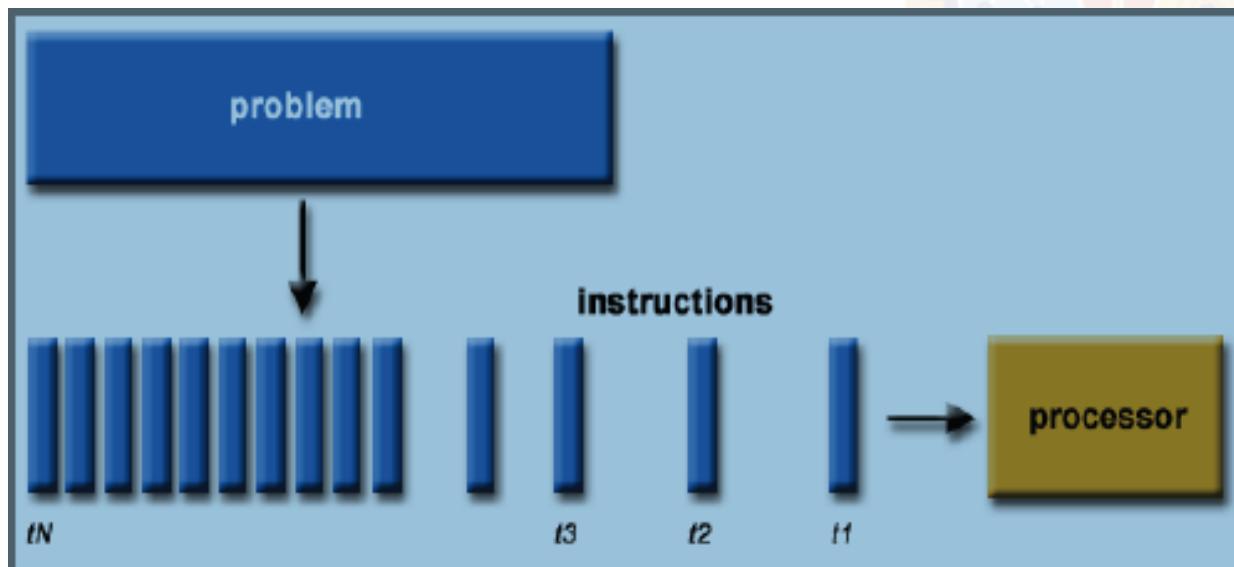


# Memory Wall



# Serial Computing

- Software written for serial computation:
  - ✓ A problem is broken into a discrete series of instructions
  - ✓ Instructions are executed sequentially one after another
  - ✓ Executed on a single processor
  - ✓ Only one instruction may execute at any moment in time
  - ✓ Single data stores - memory and disk

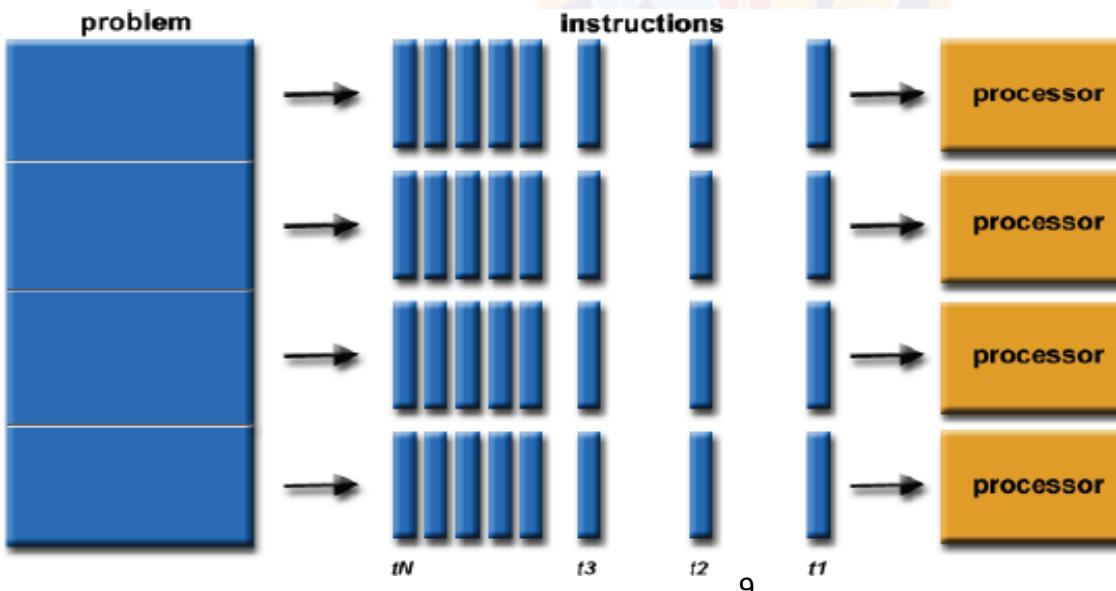


## Extra info:

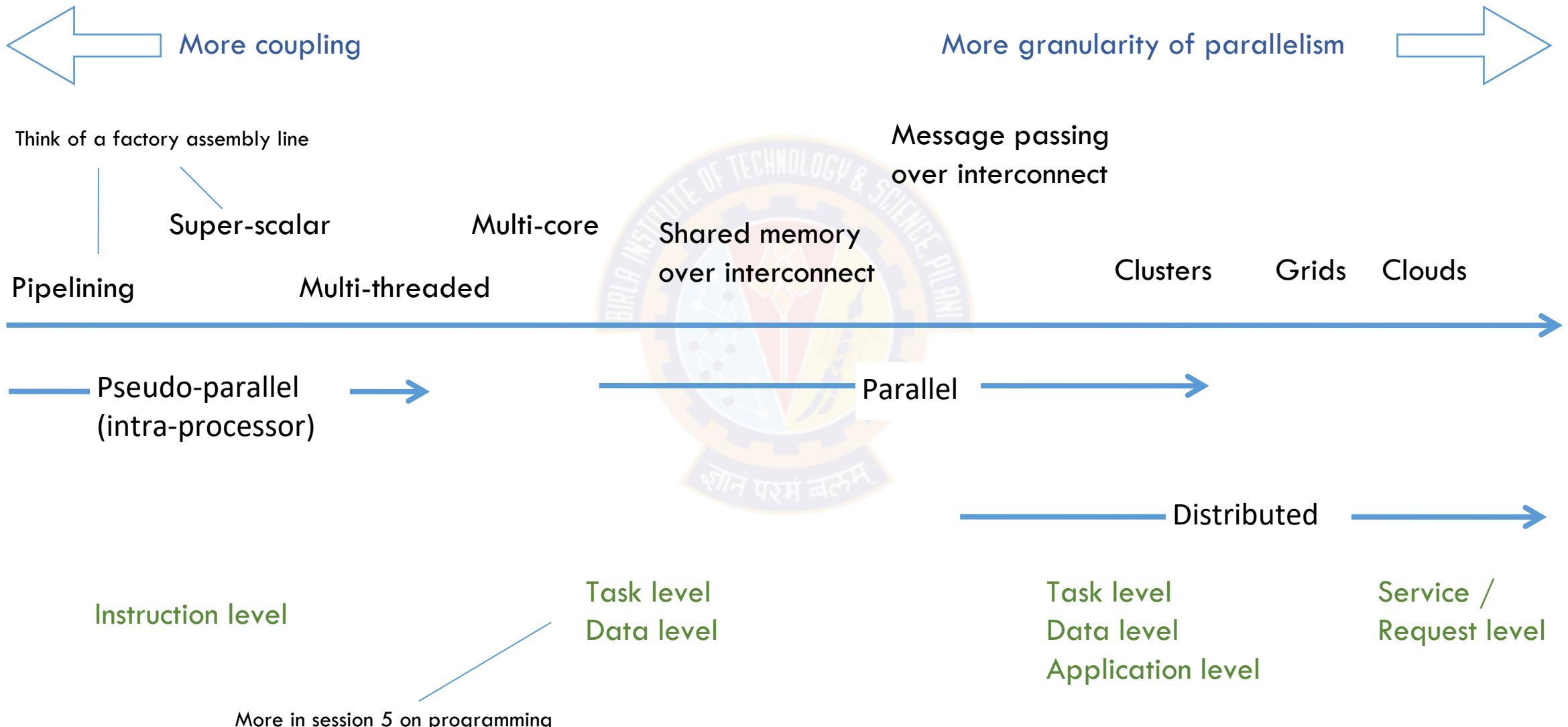
- Von Neumann architecture : common memory store and pathways between instructions and data - causes Von Neumann bottleneck
- Harvard architecture separates them to reduce bottleneck.
- Modern architectures use separate caches for instruction and data.

# Parallel Computing

- Simultaneous use of multiple compute resources to solve a computational problem
  - ✓ A problem is broken into discrete parts that can be solved concurrently
  - ✓ Each part is further broken down to a series of instructions
  - ✓ Instructions from each part execute simultaneously on different processors
  - ✓ Different processors can work with independent memory and storage
  - ✓ An overall control/coordination mechanism is employed

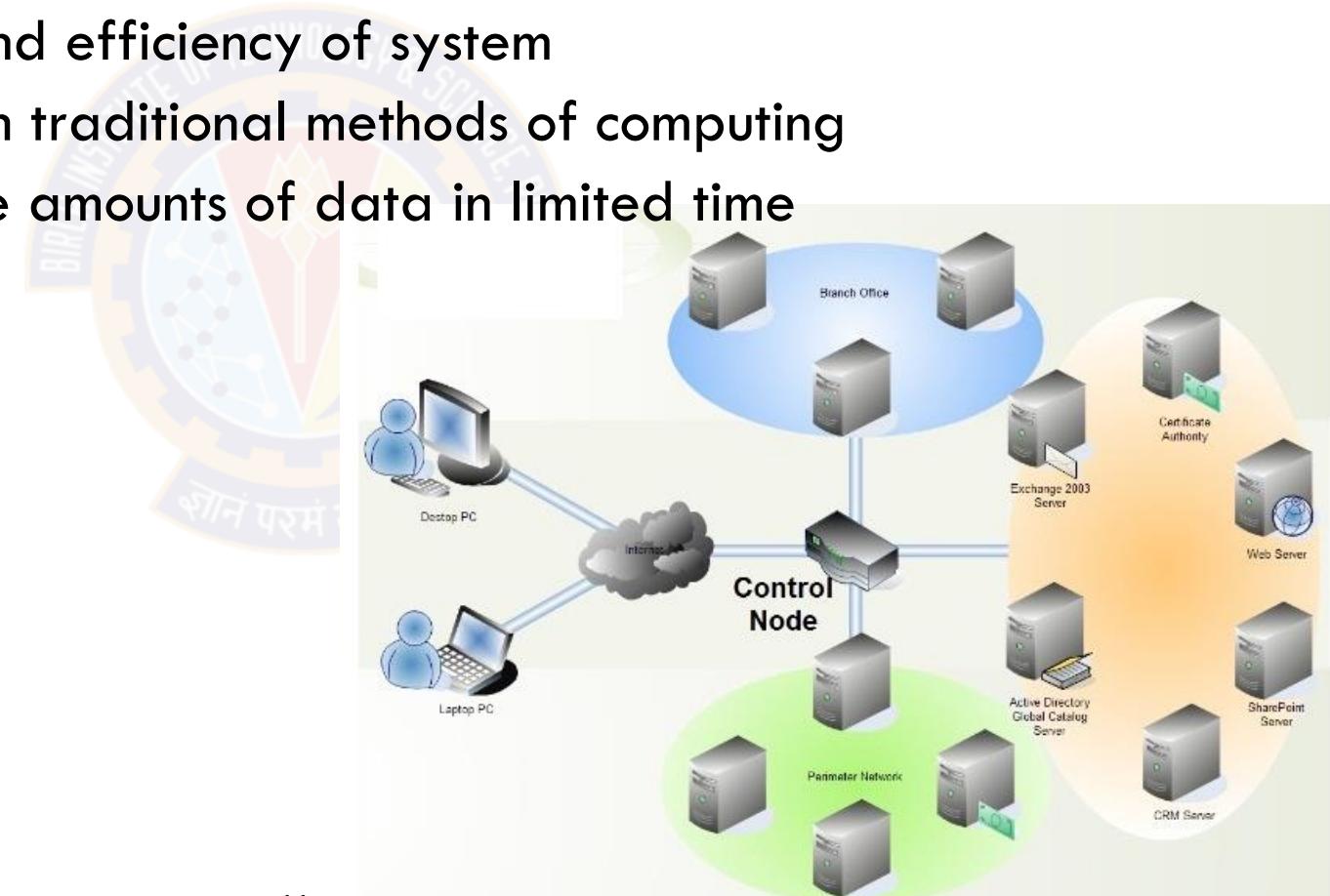


# Spectrum of Parallelism



# Distributed Computing

- In distributed computing,
  - ✓ Multiple computing resources are connected in network and computing tasks are distributed across these resources
  - ✓ Results in increase in speed and efficiency of system
  - ✓ Faster and more efficient than traditional methods of computing
  - ✓ More suitable to process huge amounts of data in limited time



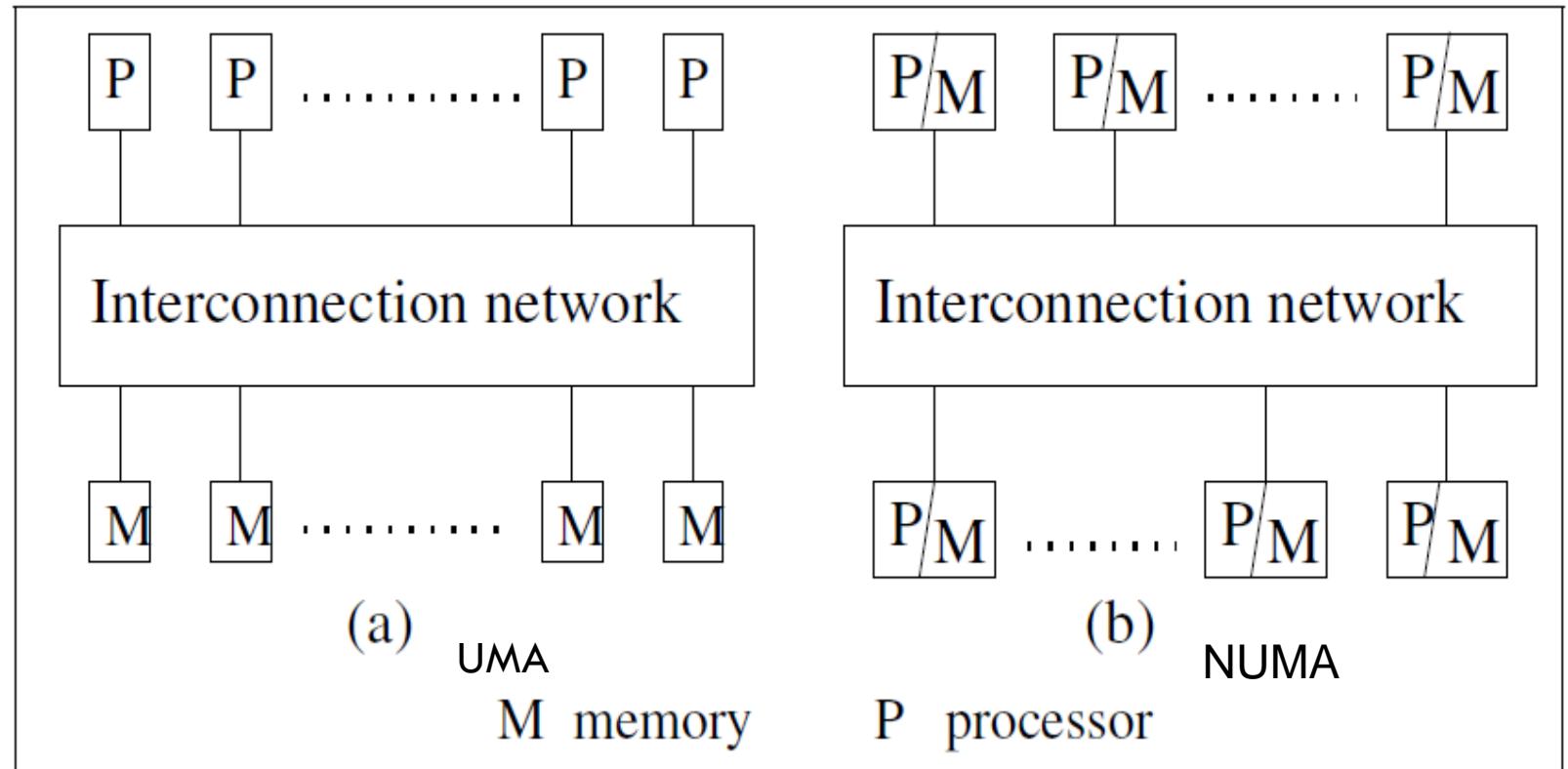
# Multi-processor Vs Multi-computer systems

## UMA

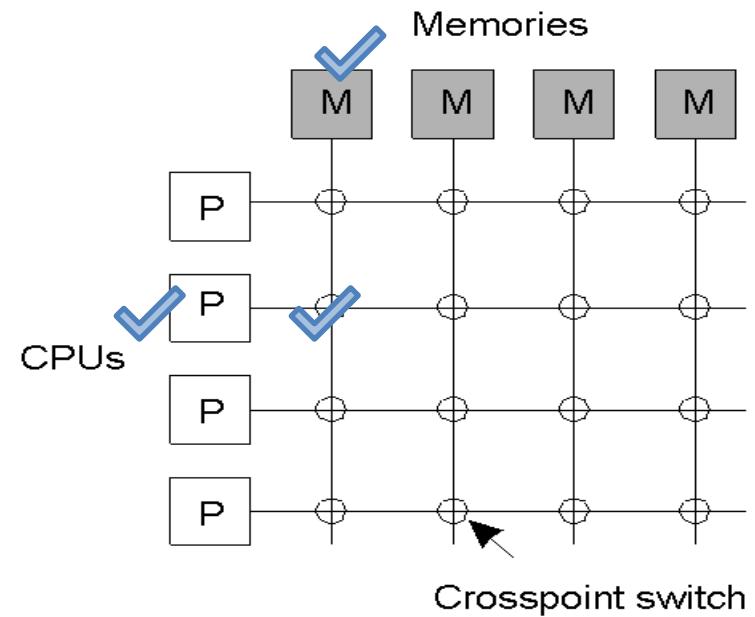
- » Uniform Memory Access Multiprocessor
- » Shared memory address space
- » No common clock
- » Fast interconnect

## NUMA

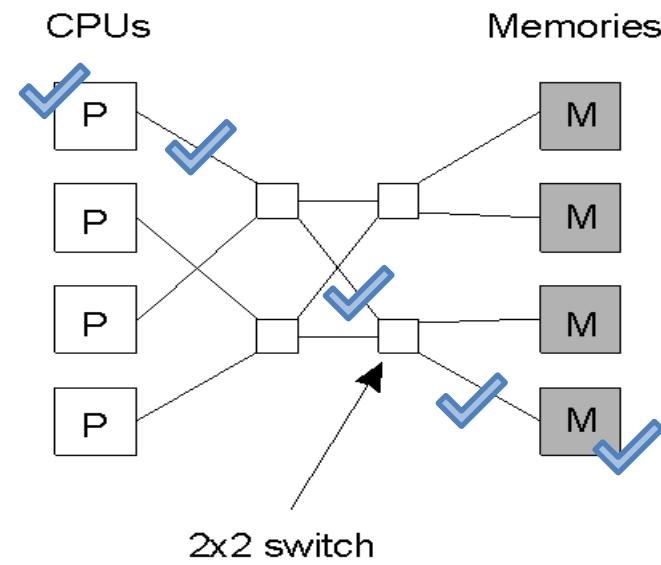
- » Non Uniform Memory Access Multicomputer
- » May have shared address spaces
- » Typically message passing
- » No common clock



# Interconnection Networks



(a)



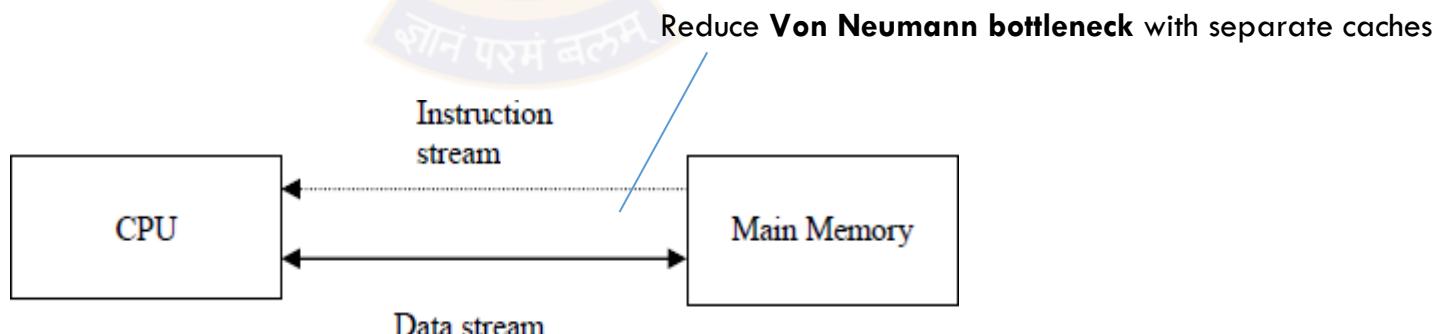
(b)

- a) A crossbar switch - faster
- b) An omega switching network - cheaper

# Classification based on Instruction and Data parallelism

## Instruction Stream and Data Stream

- The term ‘stream’ refers to a sequence or flow of either instructions or data operated on by the CPU.
- In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called instruction stream.
- Similarly, there is a flow of operands between processor and memory bi-directionally. This flow of operands is called data stream.



# Flynn's Taxonomy

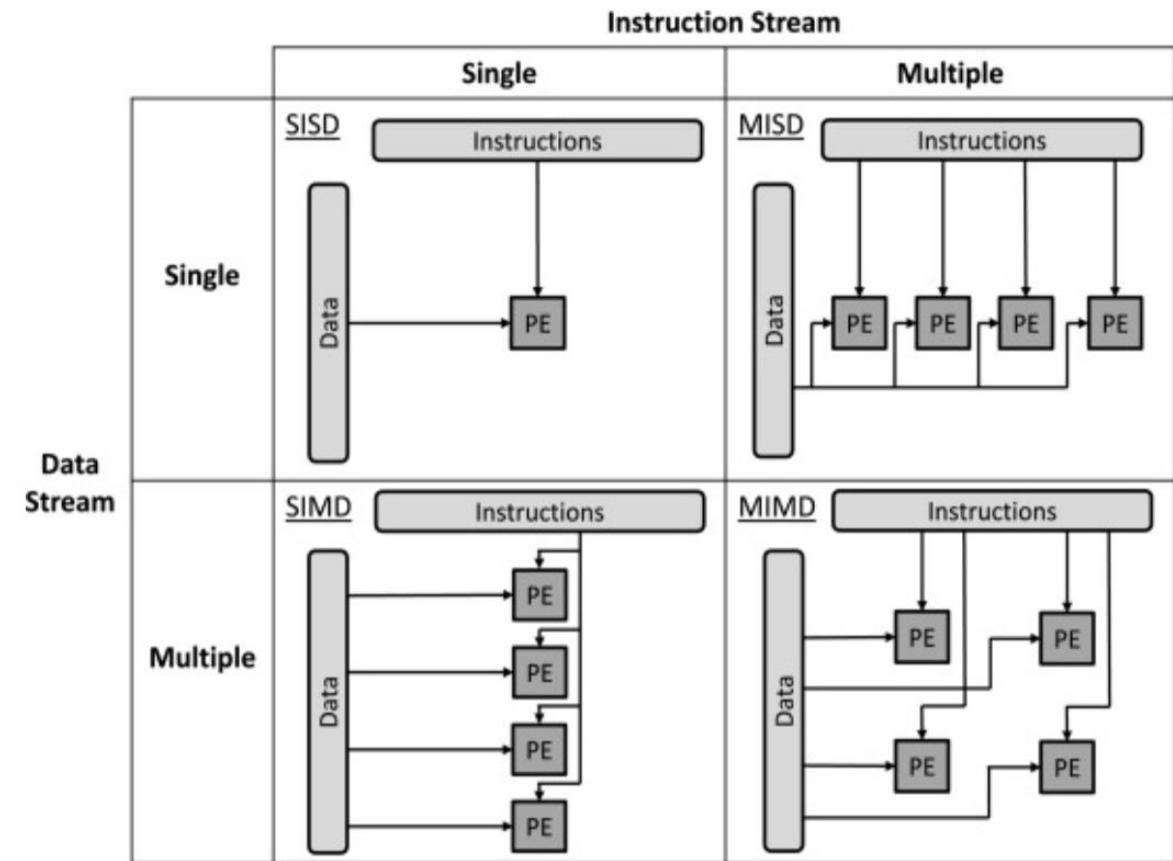
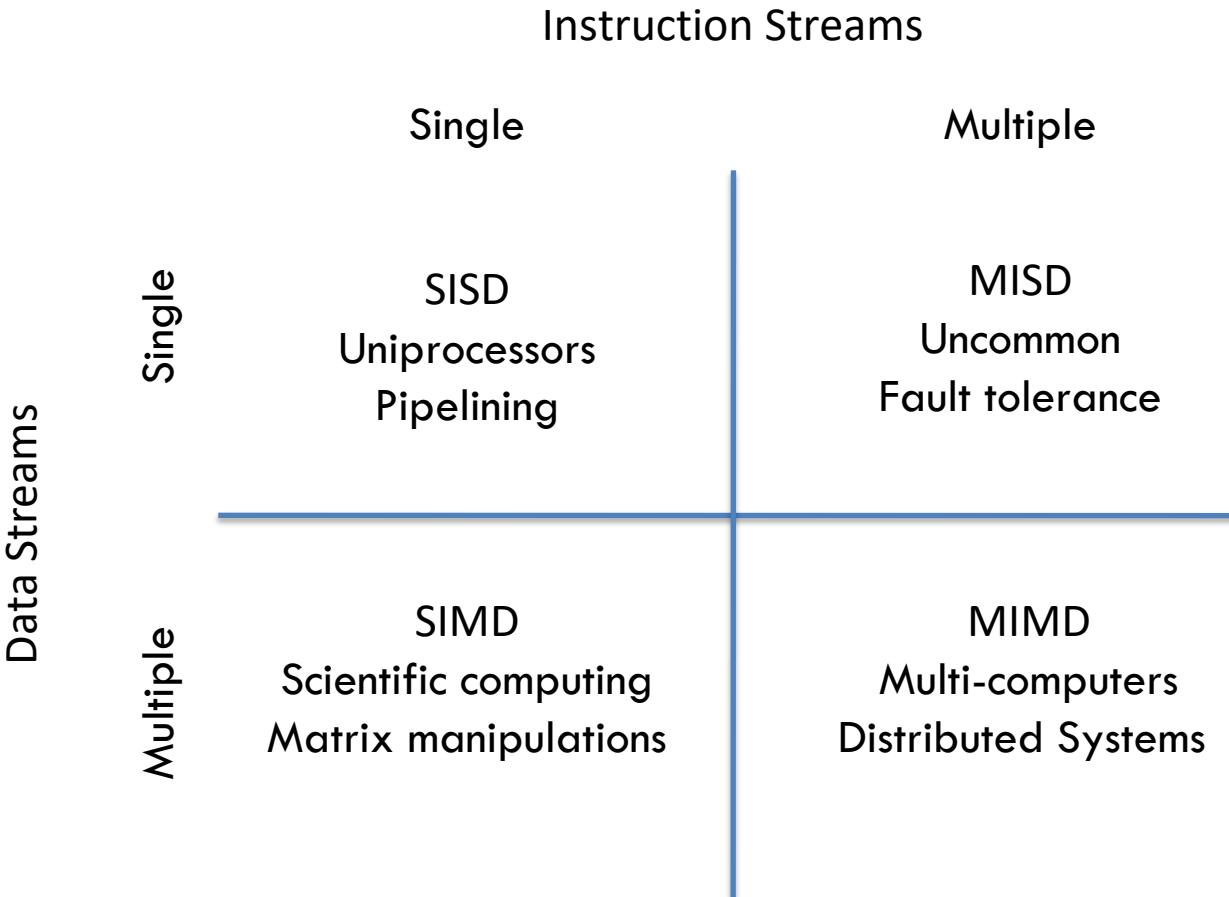


Image from [sciedirect.com](http://sciedirect.com)

# Some basic concepts

( esp. for programming in Big Data Systems )

- » Coupling
  - » Tight - SIMD, MISD shared memory systems
  - » Loose - NOW, distributed systems, no shared memory
- » Speedup
  - » how much faster can a program run when given N processors as opposed to 1 processor —  $T(1) / T(N)$
  - » We will study Amdahl's Law, Gustafson's Law
- » Parallelism of a program
  - » Compare time spent in computations to time spent for communication via shared memory or message passing
- » Granularity
  - » Average number of compute instructions before communication is needed across processors
- » Note:
  - » If coarse granularity, use distributed systems else use tightly coupled multi-processors/computers
  - » **Potentially high parallelism doesn't lead to high speedup if granularity is too small leading to high overheads**

# Comparing Parallel and Distributed Systems

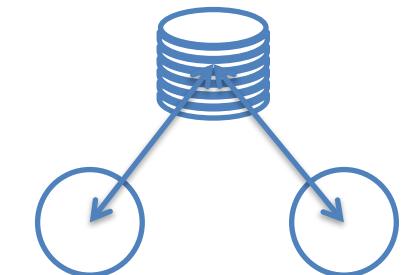
Parallel System	Distributed System
Computer system with several processing units attached to it	Independent, autonomous systems connected in a network accomplishing specific tasks
A common shared memory can be directly accessed by every processing unit in a network	Coordination is possible between connected computers with own memory and CPU
Tight coupling of processing resources that are used for solving single, complex problem	Loose coupling of computers connected in network, providing access to data and remotely located resources
Programs may demand fine grain parallelism	Programs have coarse grain parallelism

# Motivation for parallel / distributed systems (1)

- Inherently distributed applications
  - e.g. financial tx involving 2 or more parties
- Better scale in creating multiple smaller parallel tasks instead of a complex task
  - e.g. evaluate an aggregate over 6 months data
- Processors getting cheaper and networks faster
  - e.g. Processor speed 2x / 1.5 years, network traffic 2x/year, processors limited by energy consumption
- Better scale using replication or partitioning of storage
  - e.g. replicated media servers for faster access or shards in search engines
- Access to shared remote resources
  - e.g. remote central DB
- Increased performance/cost ratio compared to special parallel systems
  - e.g. search engine runs on a Network-of-Workstations



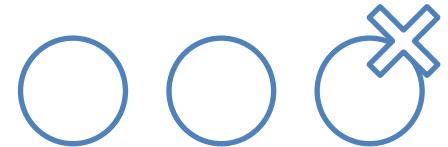
replicated / partitioned storage



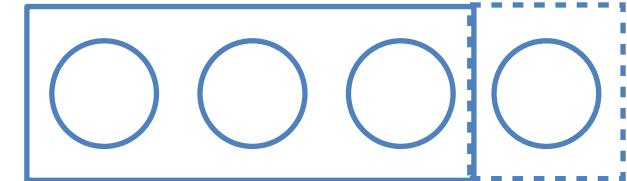
remote shared resource

# Motivation for parallel / distributed systems (2)

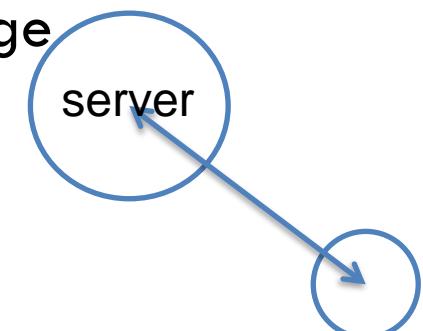
- Better reliability because less chance of multiple failures
  - Be careful about Integrity : Consistent state of a resource across concurrent access
- Incremental scalability
  - Add more nodes in a cluster to scale up
  - e.g. Clusters in Cloud services, autoscaling in AWS
- Offload computing closer to user for scalability and better resource usage
  - Edge computing



cluster nodes



resize cluster

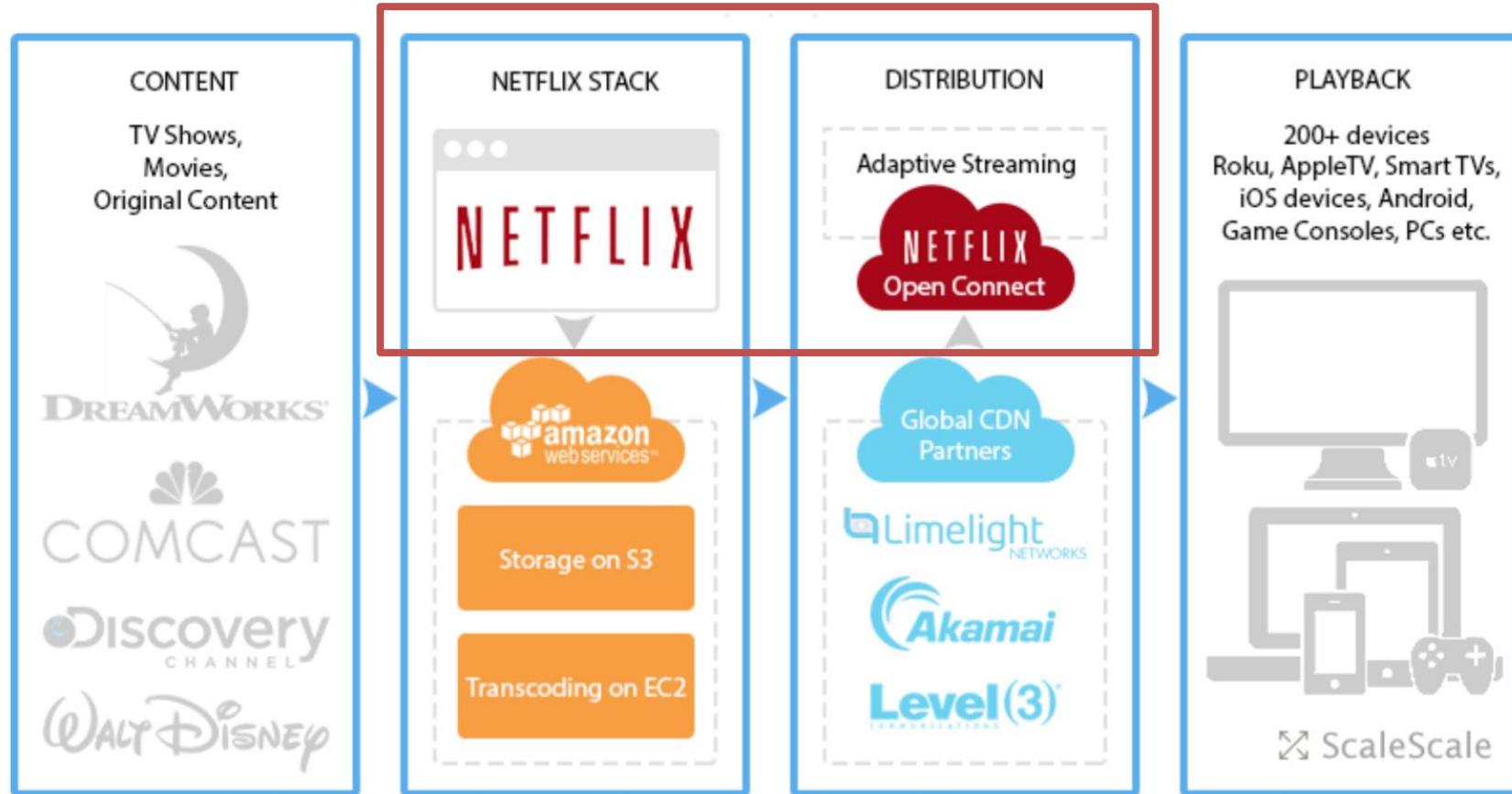


offload some error handling to edge

[Machine Learning at the Edge - DataScienceCentral.com](https://www.datasciencecentral.com/machine-learning-at-the-edge)

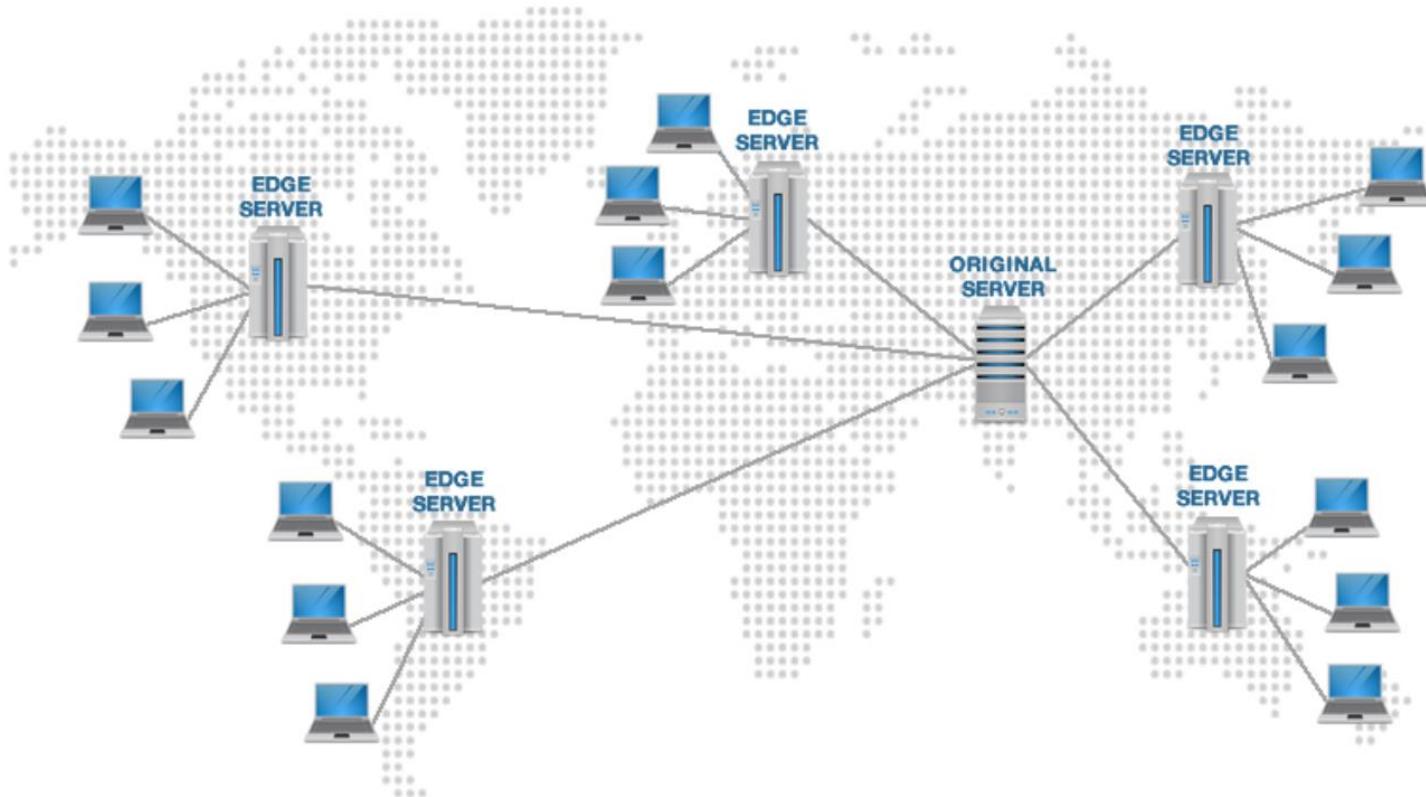
# Example: Netflix

~700+ distributed micro-services and hardware, integrated with other vendors



reference: <https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b>

# Distributed network of content caching servers



This would be a P2P network if you were using bit torrent for free

# Techniques for High Volume Data Processing

Method	Description	Usage
Cluster computing	A collection of computers, homogenous or heterogenous, using commodity components running open source or proprietary software, communicating via message passing	Commonly used in Big Data Systems, such as Hadoop
Massively Parallel Processing (MPP)	Typically, proprietary Distributed Shared Memory machines with integrated storage	May be used in traditional Data Warehouses, Data processing appliances, e.g. EMC Greenplum (postgreSQL on an MPP)
High-Performance Computing (HPC)	Known to offer high performance and scalability by using in-memory computing	Used to develop specialty and custom scientific applications for research where results is more valuable than cost

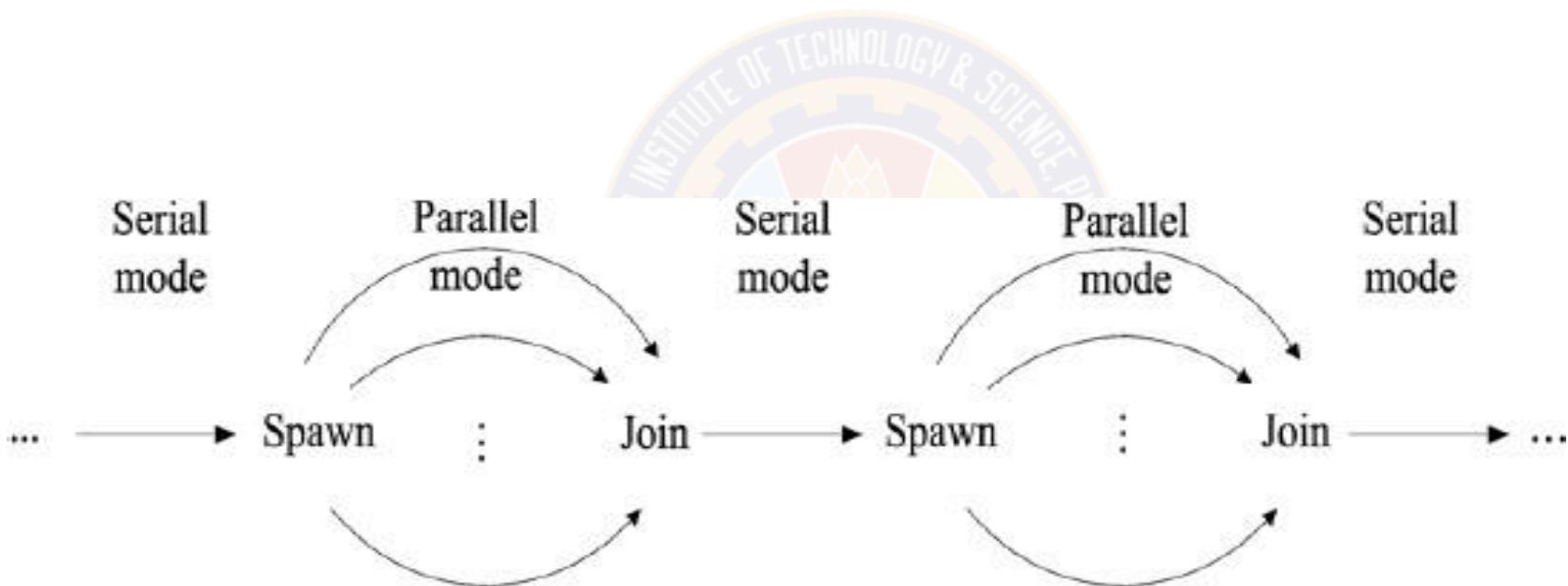
# Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- **Limits of parallelism**
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



# Limits of Parallelism

- A parallel program has some sequential / serial code and significant parallelized code



# Amdahl's Law – (1)

$$\text{Speed Up} = \frac{1}{(1-P) + P/N}$$

P = Parallel part (%) of the program

N = No. of Processors (Workers)

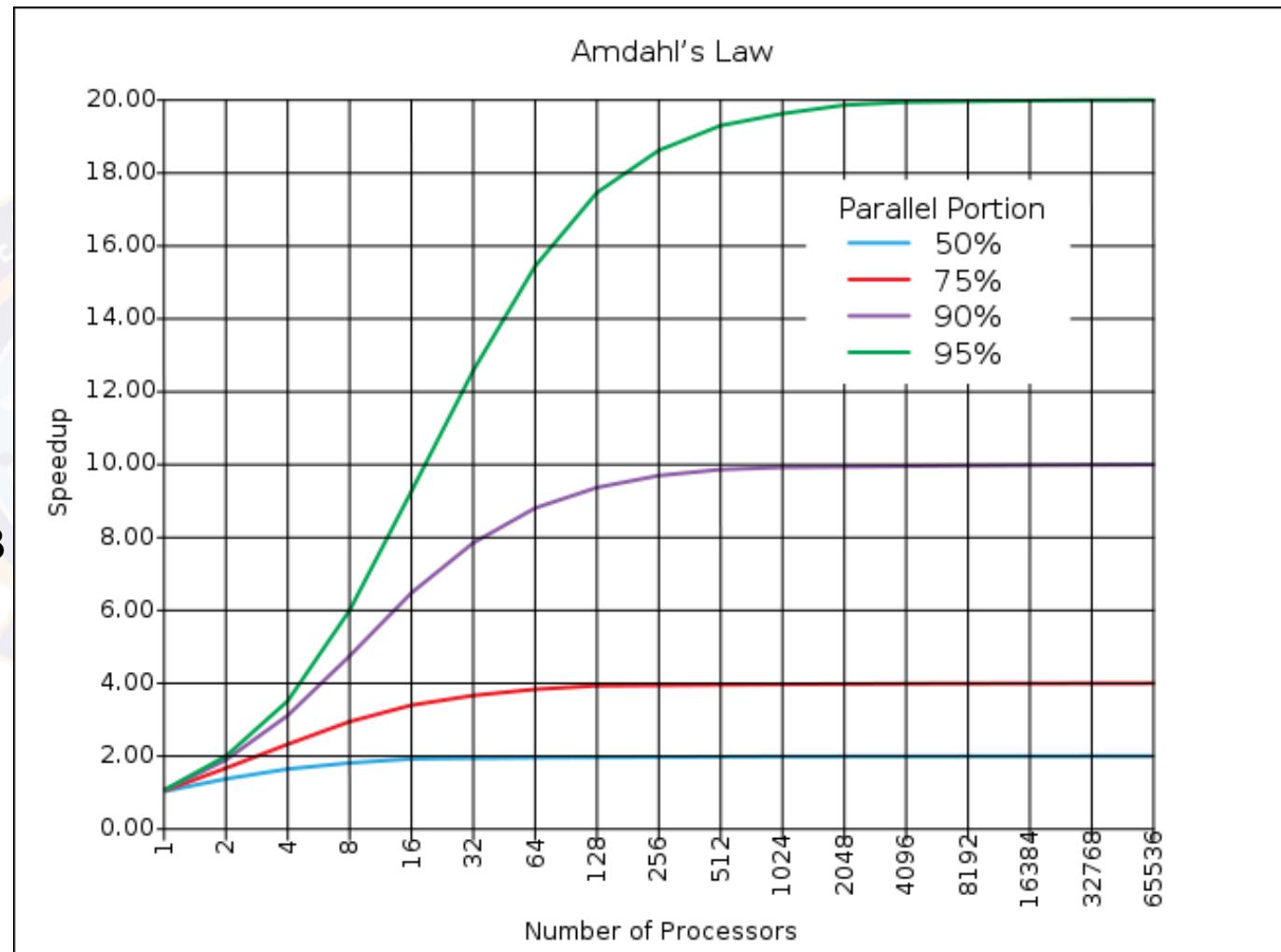
P = 0; Completely sequential – Speedup = 1

P = 1; Completely parallel – Speedup = N

P = 0.5; N=2; Partial Parallel – Speedup = 1.333

2007

**The 40<sup>th</sup> Anniversary  
of Amdahl's Law**



## Amdahl's Law – (2)

- $T(1)$  : Time taken for a job with 1 processor
- $T(N)$  : Time taken for same job with  $N$  processors
- Speedup  $S(N) = T(1) / T(N)$
- $S(N)$  is ideally  $N$  when it is a perfectly parallelizable program, i.e. data parallel with no sequential component
- Assume fraction of a program that cannot be parallelised (serial) is  $f$  and  $1-f$  is parallel
  - ✓  $T(N) \geq f * T(1) + (1-f) * T(1) / N$

- $S(N) = T(1) / ( f * T(1) + (1-f) * T(1) / N )$       **Only parallel portion is faster by  $N$**
- $S(N) = 1 / ( f + (1-f) / N )$
- Implication :
  - ✓ If  $N \Rightarrow \infty$ ,  $S(N) \Rightarrow 1/f$
  - ✓ The effective speedup is limited by the sequential fraction of the code

## Amdahl's Law - Example

10% of a program is sequential ( $f$ ) and there are 100 processors.

What is the effective speedup ?

$$S(N) = 1 / ( f + (1-f) / N )$$

$$S(100) = 1 / ( 0.1 + (1-0.1) / 100 )$$

$$= 1 / 0.109$$

$$= 9.17 \text{ (approx)}$$



# Limitations in speedup

Besides the sequential component of the program, communication delays also result in reduction of speedup

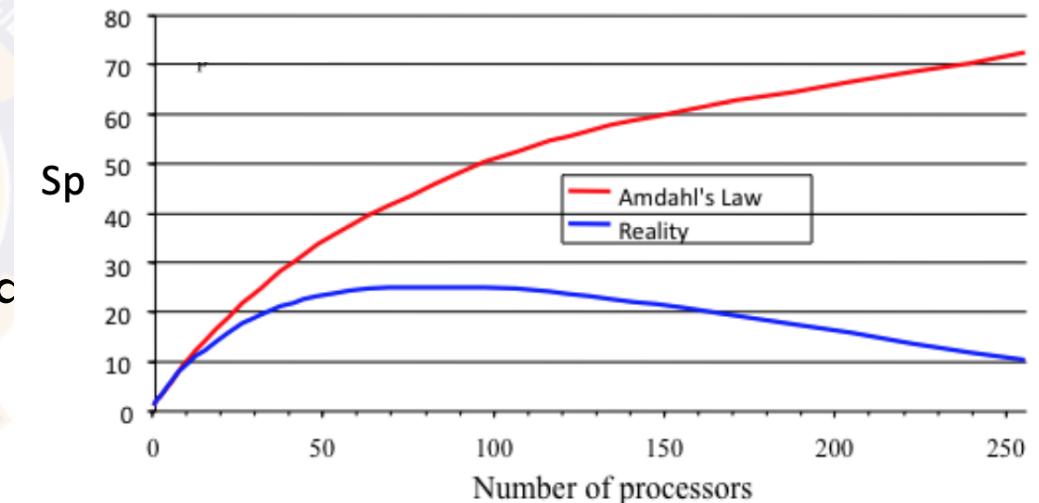
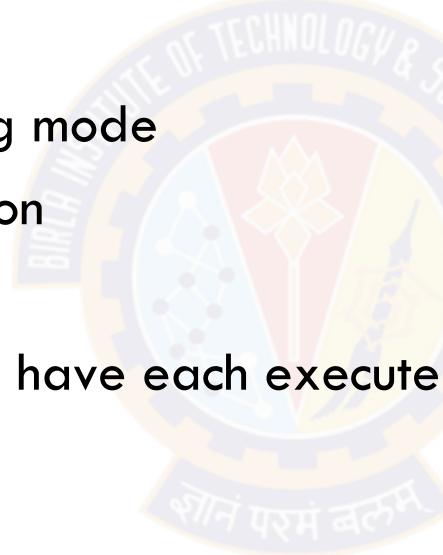
A and B exchange messages in blocking mode

Say processor speed is 0.5ns / instruction

Say network delay one way is 10 us

For one message delay, A and B would have each executed  
 $10\text{us}/0.5\text{ns} = 20000$  instructions

+ context switching, scheduling, load balancing, I/O ...

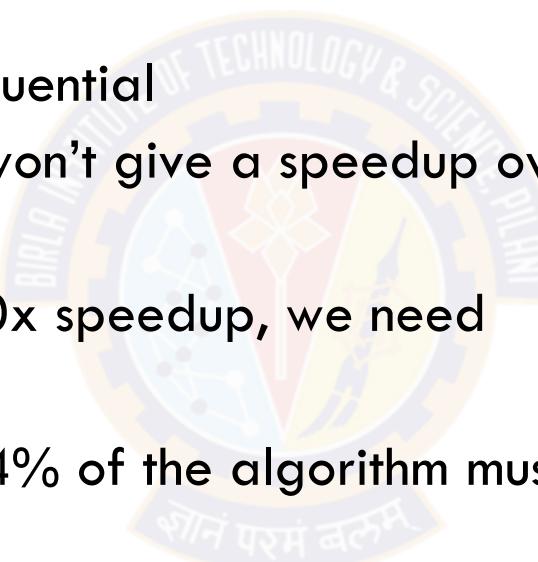


# Why Amdahl's Law is such bad news

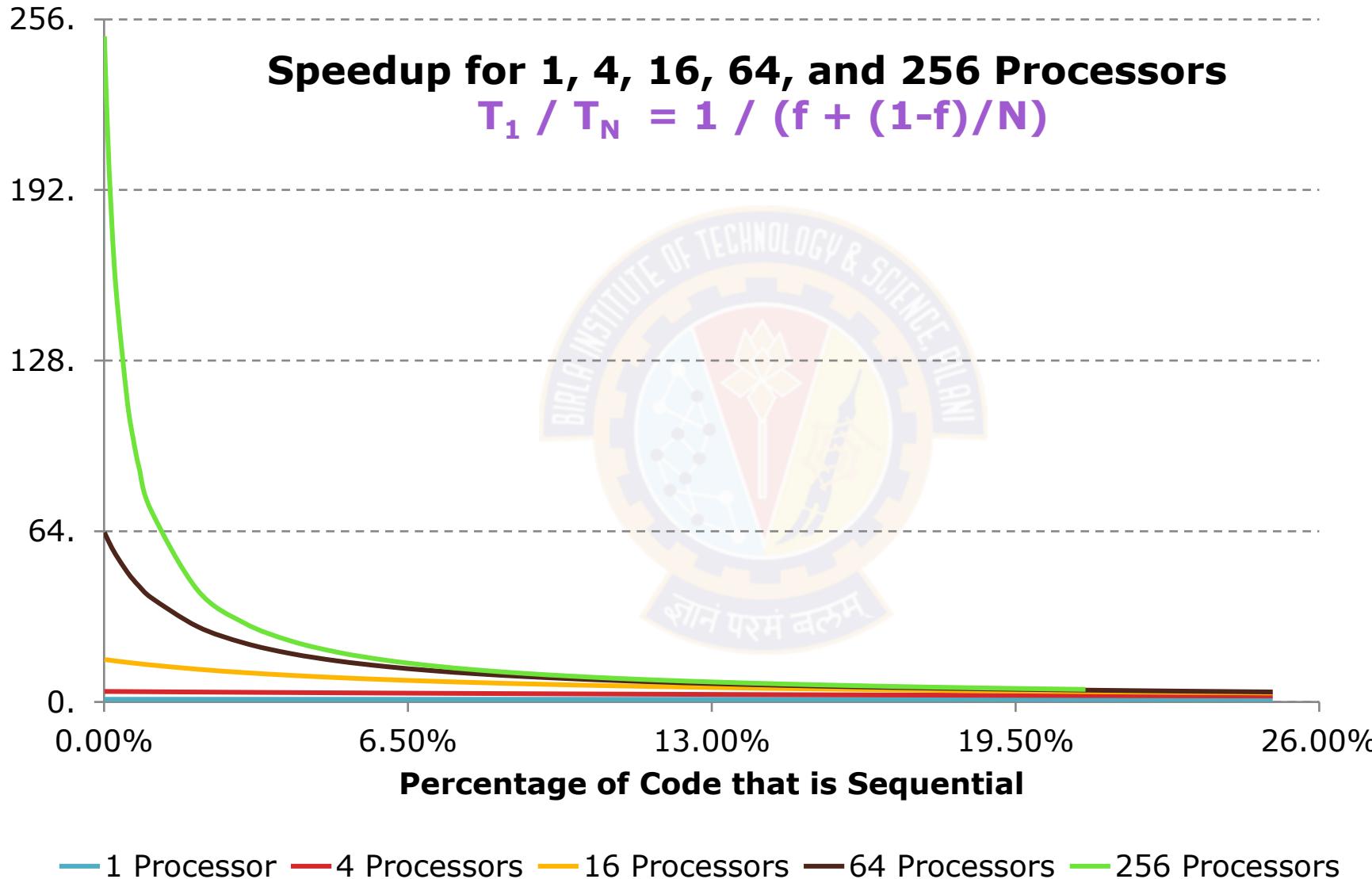
$$S(N) \sim 1/f, \text{ for large } N$$

Suppose 33% of a program is sequential

- Then even a billion processors won't give a speedup over 3
- For the 256 cores to gain  $\geq 100x$  speedup, we need  
$$100 \leq 1 / (f + (1-f)/256)$$
Which means  $f \leq .0061$  or 99.4% of the algorithm must be perfectly parallelizable !!

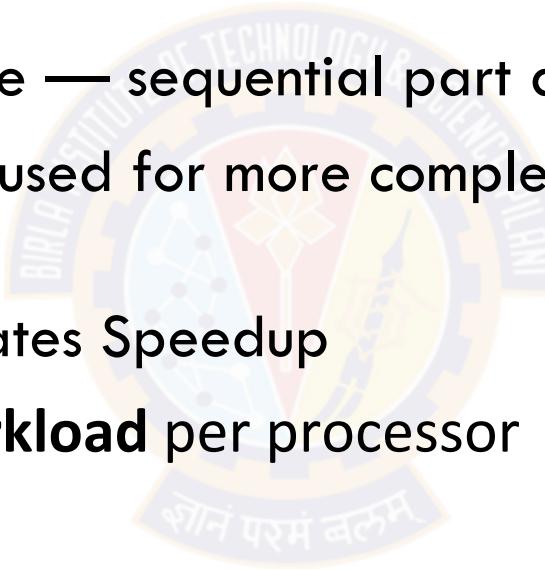


# Speedup plot



# But wait - may be we are missing something

- » The key assumption in Amdahl's Law is **total workload is fixed** as #processors is increased
- » This doesn't happen in practice — sequential part doesn't increase with resources
- » Additional processors can be used for more complex workload and new age larger parallel problems
- » So Amdahl's law under-estimates Speedup
- » What if we assume **fixed workload per processor**



# Gustafson-Barsis Law

Let  $W$  be the execution workload of the program before adding resources

$f$  is the sequential part of the workload

$$\text{So } W = f * W + (1-f) * W$$

Let  $W(N)$  be larger execution workload after adding  $N$  processors

$$\text{So } W(N) = f * W + N * (1-f) * W$$

Parallelizable work can increase  $N$  times

The theoretical speedup in latency of the whole task at a fixed interval time  $T$

$$\begin{aligned} S(N) &= T * W(N) / T * W \\ &= W(N) / W = (f * W + N * (1-f) * W) / W \end{aligned}$$

$$S(N) = f + (1-f) * N$$

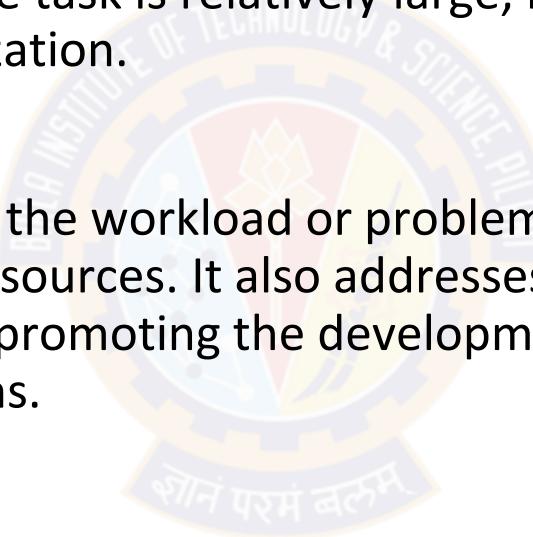
**S(N) is not limited by f as N scales**

Remember this when we discuss programming in Session 5

So solve larger problems when you have more processors

# Usage Scenarios

- **Amdahl's law** can be used when the workload is fixed, as it calculates the potential speedup with the assumption of a fixed workload. Moreover, it can be utilized when the non-parallelizable portion of the task is relatively large, highlighting the diminishing returns of parallelization.
- **Gustafson's law** is applicable when the workload or problem size can be scaled proportionally with the available resources. It also addresses problems requiring larger problem sizes or workloads, promoting the development of systems capable of handling such realistic computations.



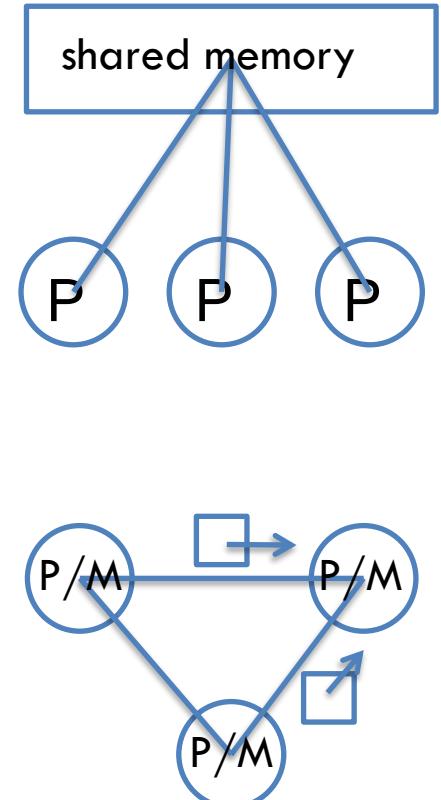
# Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- (Sec1)
- **Shared Memory vs Message Passing**
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



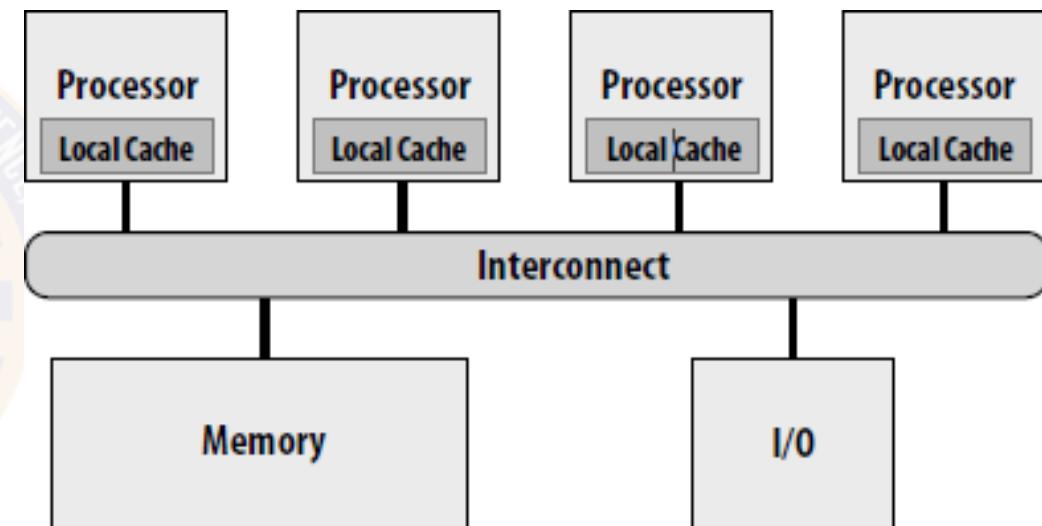
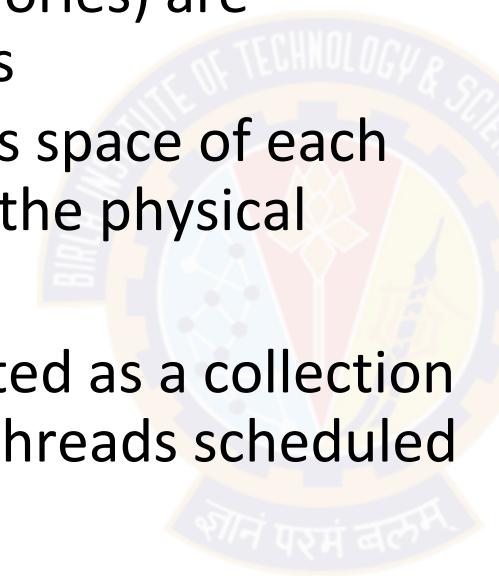
# Memory access models

- » Shared memory
  - » Multiple tasks on different processors access a common address space in UMA or NUMA architectures
  - » Conceptually easier for programmers
  - » Think of writing a voting algorithm - it is trivial because everyone is in the same room, i.e. writing same variable
- » Distributed memory
  - » Multiple tasks – executing a single program – access data from separate (and isolated) address spaces (i.e. separate virtual memories)
  - » How will this remote access happen ?



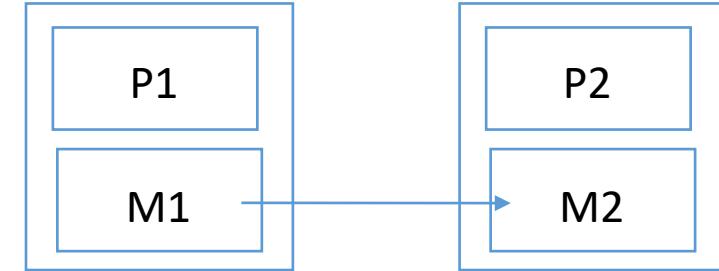
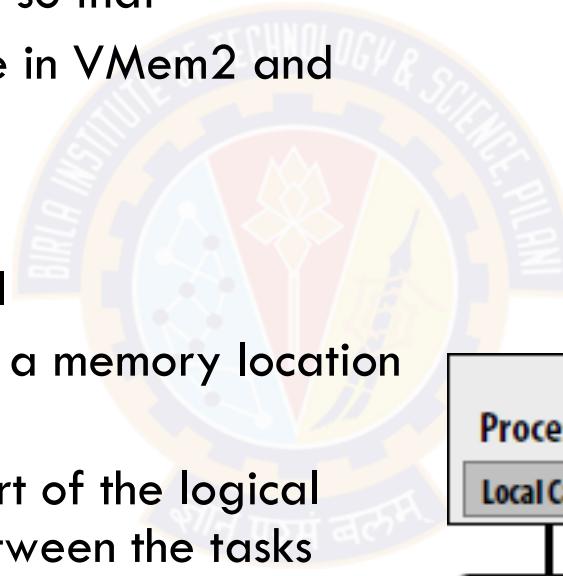
# Shared Memory Model: Implications for Architecture

- A shared memory system has
  - ✓ Physical memory (or memories) are accessible by all processors
  - ✓ The single (logical) address space of each processor is mapped onto the physical memory (or memories)
- A single program is implemented as a collection of threads, with one or more threads scheduled in a processor
- Conceptually easier to program

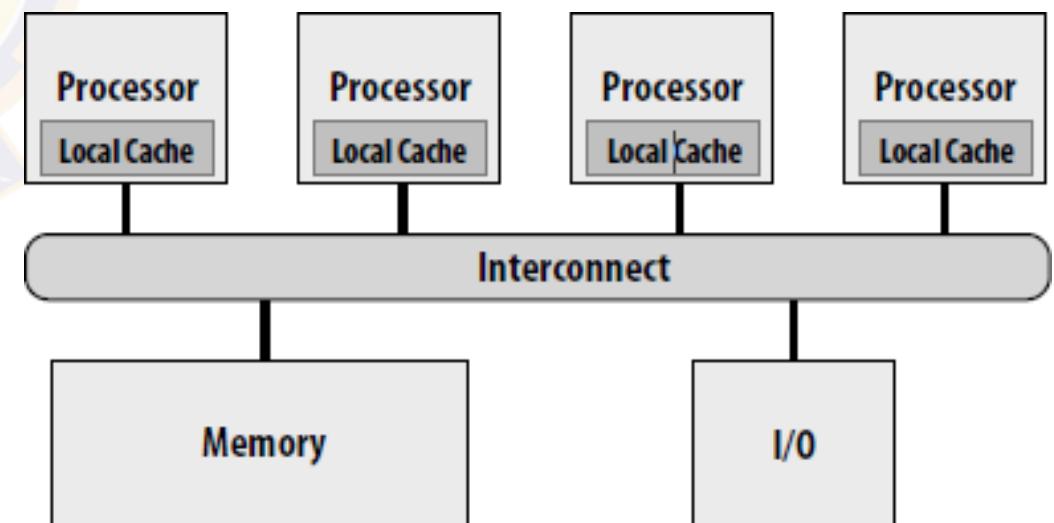


# Distributed memory with message passing Vs Shared memory

- In a Distributed Memory model, data has to be moved across Virtual Memories:
  - ✓ i.e. a data item in VMem1 produced by task T1 has to be “communicated” to task T2 so that
  - ✓ T2 can make a copy of the same in VMem2 and use it.

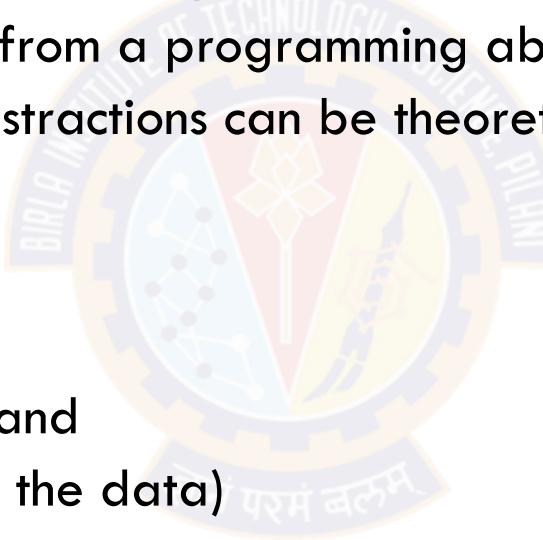


- Whereas in a Shared Memory model
  - ✓ task T1 writes the data item into a memory location and T2 can read the same
  - ✓ how ? the memory location is part of the logical address space that is shared between the tasks



# Computing model for message passing

- Each data item must be located in one of the address spaces
  - ✓ Data must be partitioned explicitly and placed (i.e. distributed)
  - ✓ All interactions between processes require explicit communication i.e. passing of messages
  - ✓ Harder than shared memory from a programming abstraction standpoint
    - ✓ Note: Shared memory abstractions can be theoretically created on message passing systems
- In the simplest form:
  - ✓ a sender (who has the data) and
  - ✓ a receiver (who has to access the data)
  - must co-operate for exchange of data



# Communication model for message passing

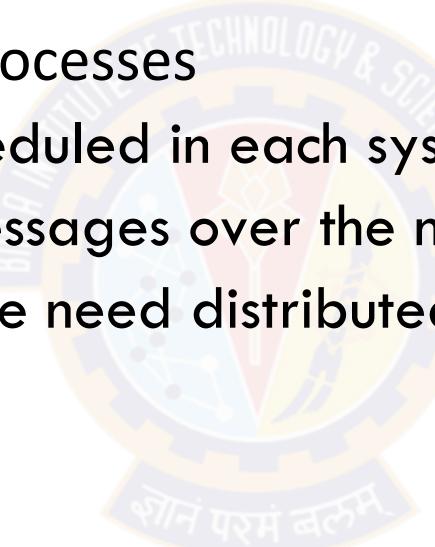
- Processes operate within their own private address spaces
- Processes communicate by sending/receiving messages
  - ✓ send: specifies recipient, buffer to be transmitted, and message identifier (“tag”)
  - ✓ receive: specifies buffer to store data, and optional message identifier
  - ✓ Sending messages is the only way to exchange data between processes 1 and 2



# Distributed Memory Model: Implications for Architecture

A distributed memory model is implemented in a distributed system, where

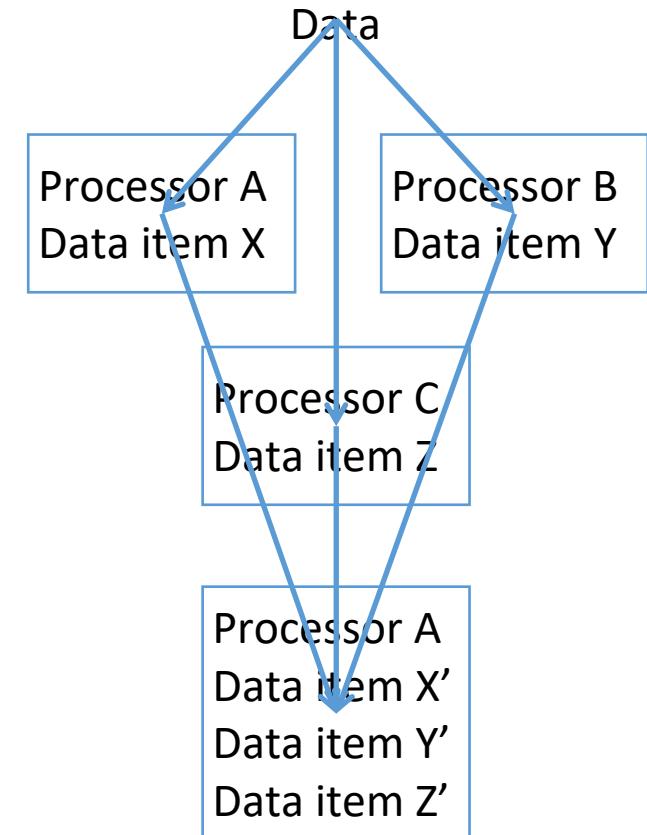
- ✓ A collection of stand-alone systems are connected in a network
- ✓ A task runs as a collection of processes
- ✓ One or more processes are scheduled in each system / node
- ✓ Data exchanges happen via messages over the network
- ✓ Harder for programmers - hence need distributed OS / middleware layer to hide some complexity \*



\* One can create a shared memory view using message passing and vice versa

# Message Passing Model – Separate Address Spaces

- Use of separate address spaces complicates programming
- But this complication is usually restricted to one or two phases:
  - ✓ Partitioning the input data
    - ✓ Improves locality - computation closer to data
    - ✓ Each process is enabled to access data from within its address space, which in turn is likely to be mapped to the memory hierarchy of the processor in which the process is running
  - ✓ Merging / Collecting the output data
    - ✓ This is required if each task is producing outputs that have to be combined

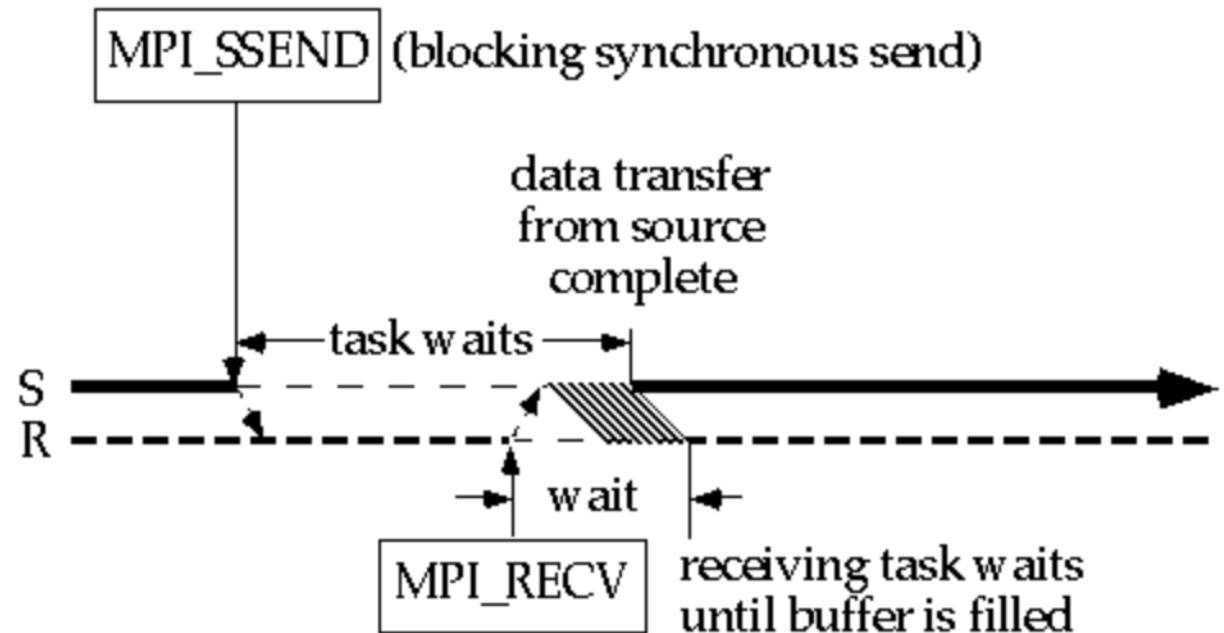


We will see example of Hadoop map-reduce where data is partitioned, outputs communicated over messages and merged to get final answer.

Remember granularity ?

# Message Passing Primitives

- » Operations: Send and Receive
- » Options:
  - » Blocking vs Non-Blocking
  - » Sync vs Async



- » Important : A message can be received in the OS buffer but may not have been delivered to application buffer. This is where a distributed message ordering logic can come in.

Image ref: <https://cvw.cac.cornell.edu/mpip2p/SyncSend>

# Send-Receive Options

1	Send	Sync	Blocking	Returns only after data is sent from kernel buffer. Easiest to program but longest wait.
2	Send	Async	Blocking	Returns after data is copied to kernel buffer but not sent. Handle returned to check send status.
3	Send	Sync	Non-blocking	Same as (2) but with no handle.
4	Send	Async	Non-blocking	Returns immediately with handle. Complex to program but minimum wait.
5	Receive	Sync	Blocking	Returns after application gets data.
6	Receive	Sync	Non-blocking	Returns immediately with data or handle to check status. More efficient.

# Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- **Data access strategies - Replication, Partitioning, Messaging**
- Cluster computing



# Data Access Strategies: Partition

- Strategy:
  - ✓ Partition data – typically, equally – to the nodes of the (distributed) system
- Cost:
  - ✓ Network access and merge cost when query needs to go across partitions
- Advantage(s):
  - ✓ Works well if task/algorithim is (mostly) data parallel
  - ✓ Works well when there is Locality of Reference within a partition
- Concerns
  - ✓ Merge across data fetched from multiple partitions
  - ✓ Partition balancing
  - ✓ Row vs Columnar layouts - what improves locality of reference ?
  - ✓ Will study shards and partition in Hadoop, MongoDB, and Cassandra

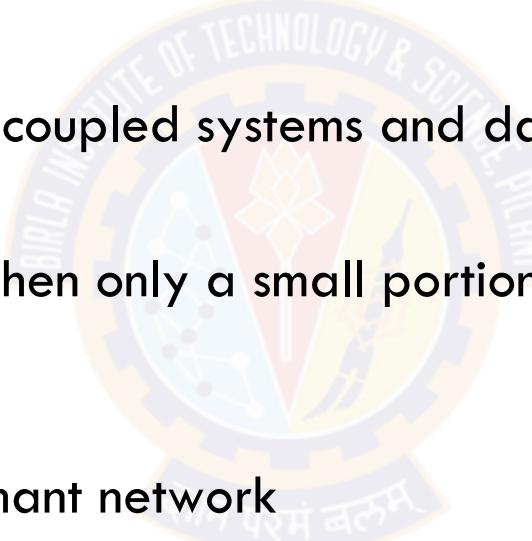
# Data Access Strategies: Replication

- **Strategy:**
  - ✓ Replicate all data across nodes of the (distributed) system
- **Cost:**
  - ✓ Higher storage cost
- **Advantage(s):**
  - ✓ All data accessed from local disk: no (runtime) communication on the network
  - ✓ High performance with parallel access
  - ✓ Fail over across replicas
- **Concerns**
  - ✓ Keep replicas in sync — various consistency models between readers and writers
  - ✓ Will study in depth for MongoDB, Cassandra



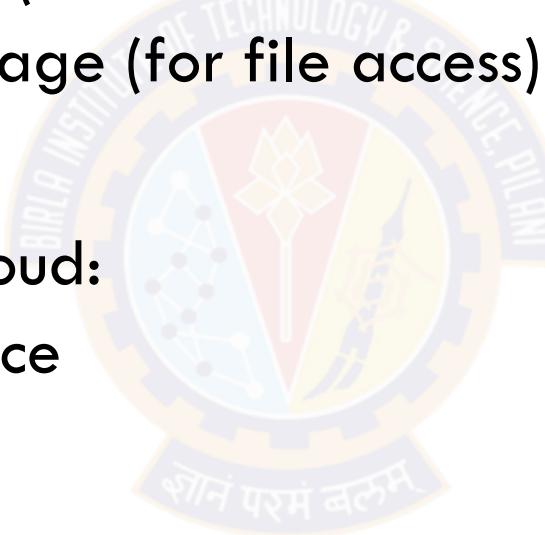
# Data Access Strategies: (Dynamic) Communication

- Strategy:
  - ✓ Communicate (at runtime) only the data that is required
- Cost:
  - ✓ High network cost for loosely coupled systems and data set to be exchanged is large
- Advantage(s):
  - ✓ Minimal communication cost when only a small portion of the data is actually required by each node
- Concerns
  - ✓ Highly available and performant network
  - ✓ Fairly independent parallel data processing



# Data Access Strategies – Networked Storage

- Common Storage on the Network:
  - ✓ Storage Area Network (for raw access – i.e. disk block access)
  - ✓ Network Attached Storage (for file access)
- Common Storage on the Cloud:
  - ✓ Use Storage as a Service
  - ✓ e.g. Amazon S3



More in-depth coverage when studying Amazon storage case study  
Webinar on Bigdata storage on cloud

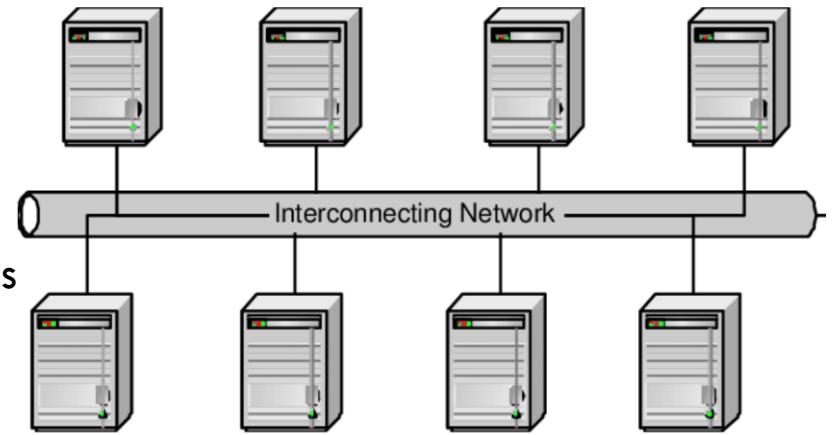
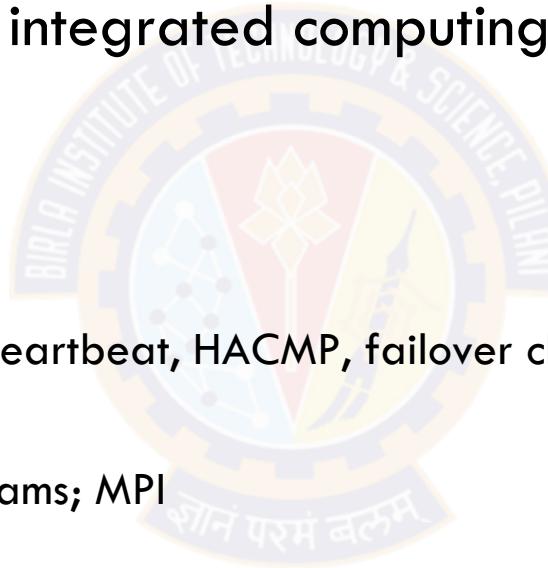
# Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- (Sec2)
- Cluster computing



# Computer Cluster - Definition

- A cluster is a type of distributed processing system
  - ✓ consisting of a collection of inter-connected stand-alone computers
  - ✓ working together as a single, integrated computing resource
  - ✓ Examples:
    - High Availability Clusters  
ServiceGuard, Lifekeeper, Failsafe, heartbeat, HACMP, failover clusters
    - High Performance Clusters  
Beowulf; 1000 nodes; parallel programs; MPI
    - Database Clusters  
Oracle Parallel Server (OPS)
    - Storage Clusters  
Cluster filesystems; same view of data from each node.  
HDFS



## Cluster - Goals

- Continuous availability
- Data integrity
- Linear scalability
- Open access
- Parallelism in processing
- Distributed systems management



# Cluster - Objectives

- A computer cluster is typically built for one of the following two reasons:
  - ✓ High Performance - referred to as compute-clusters
  - ✓ High Availability - achieved via redundancy

An off-the-shelf or custom load balancer, reverse proxy can be configured to serve the use case

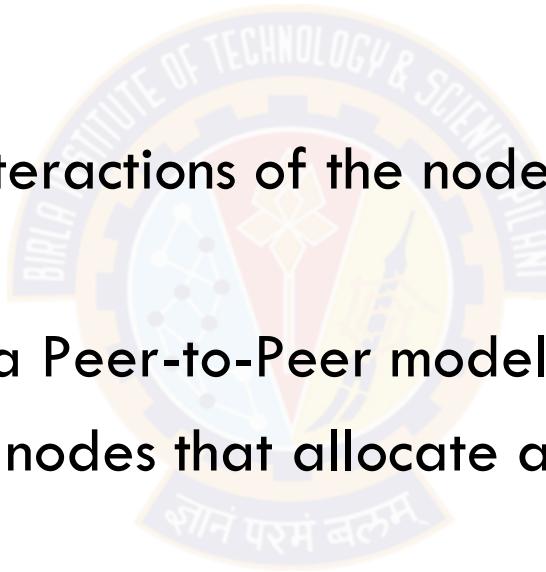
- Question: How is this relevant for Big Data?

Hadoop nodes are a cluster for performance (independent Map/Reduce jobs are started on multiple nodes) and availability (data is replicated on multiple nodes for fault tolerance)

**Most Big Data systems run on a cluster configuration for performance and availability**

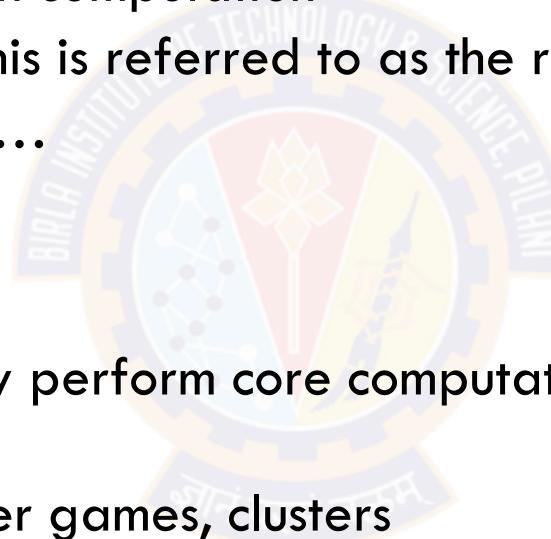
# Clusters – Peer to Peer computation

- Distributed Computing models can be classified as:
  - ✓ Client Server models
  - ✓ Peer-to-Peer models
- based on the structure and interactions of the nodes in a distributed system
- Clusters within the nodes use a Peer-to-Peer model of computation.
- There may be special control nodes that allocate and manage work thus having a master-slave relationship.



# Client-Server vs. Peer-to-Peer

- Client-Server Computation
  - ✓ A server node performs the core computation – business logic in case of applications
  - ✓ Client nodes request for such computation
  - ✓ At the programming level this is referred to as the request-response model
  - ✓ Email, network file servers, ...
- Peer-to-Peer Computation:
  - ✓ All nodes are peers i.e. they perform core computations and may act as client or server for each other.
  - ✓ bit torrent, some multi-player games, clusters



# Cloud and Clusters

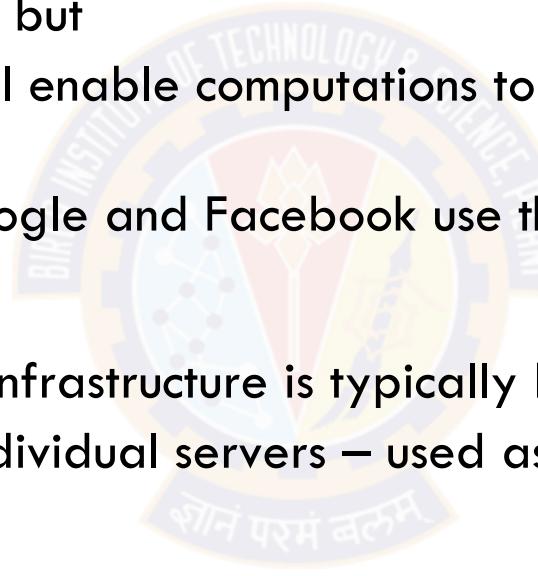
- A cloud uses a datacenter as the infrastructure on top of which services are provided
  - e.g. AWS would have a datacenter in many regions - Mumbai, US east, ... (you can pick where you want your services deployed)
- A cluster is the basic building block for a datacenter:
  - ✓ i.e. a datacenter is structured as a collection of clusters
- A cluster can host
  - ✓ a multi-tenant service across clients - cost effective
  - ✓ individual clients and their service(s) - dedicated instances

# Motivation for using Clusters (1)

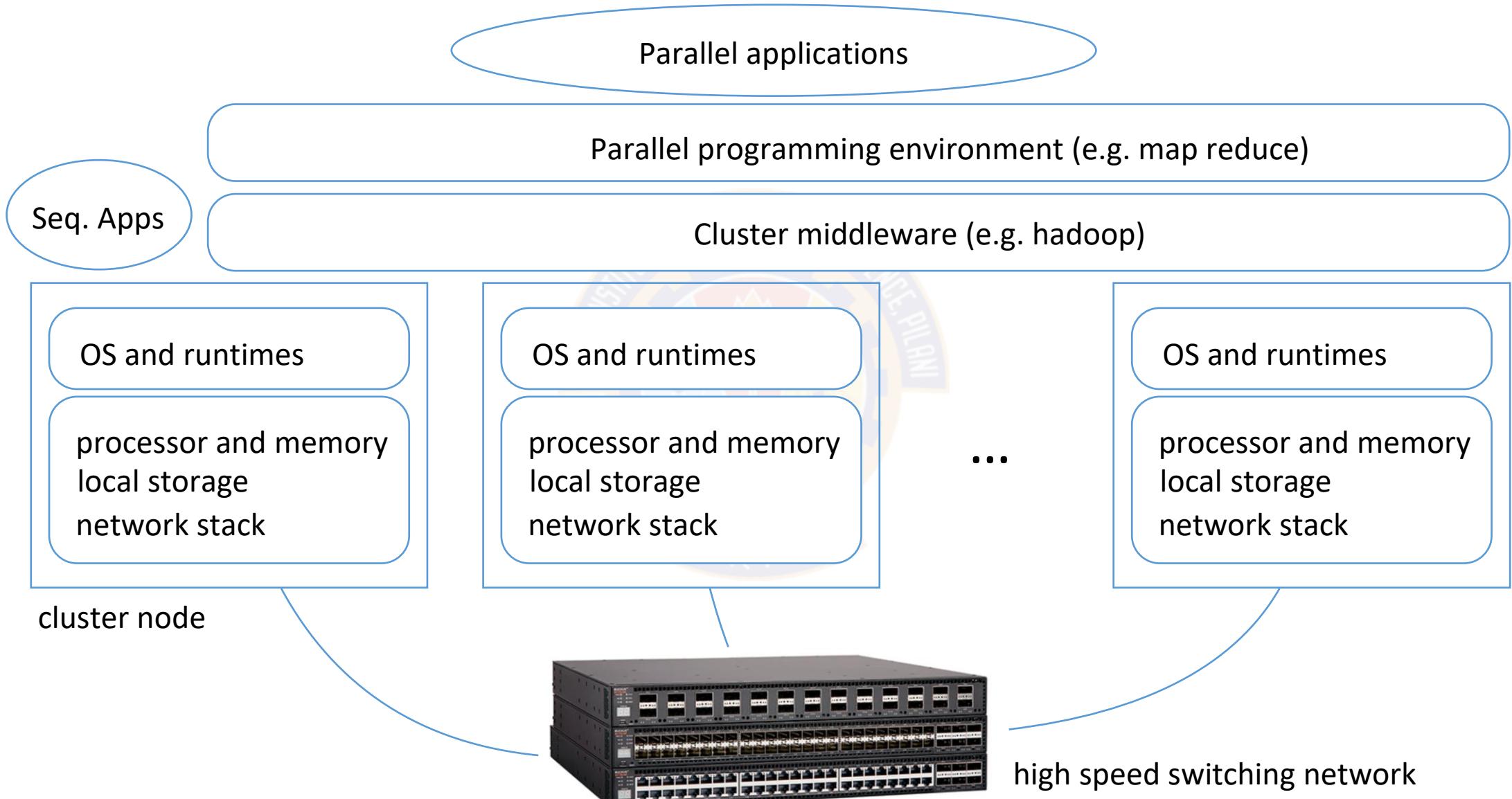
- Rate of obsolescence of computers is high
  - ✓ Even for mainframes and supercomputers
  - ✓ Servers (used for high performance computing) have to be replaced every 3 to 5 years.
- Solution: Build a cluster of commodity workstations
  - ✓ Incrementally add nodes to the cluster to meet increasing workload
  - ✓ Add nodes instead of replacing (i.e. let older nodes operate at a lower speed)
  - ✓ This model is referred to as a scale-out cluster

## Motivation for using Clusters (2)

- Scale-out clusters with commodity workstations as nodes are suitable for software environments that are resilient:
  - ✓ i.e. individual nodes may fail, but
  - ✓ middleware and software will enable computations to keep running (and keep services available) for end users
  - ✓ for instance, back-ends of Google and Facebook use this model.
- On the other hand, (public) cloud infrastructure is typically built as clusters of servers
  - ✓ due to higher reliability of individual servers – used as nodes – (compared to that of workstations as nodes).

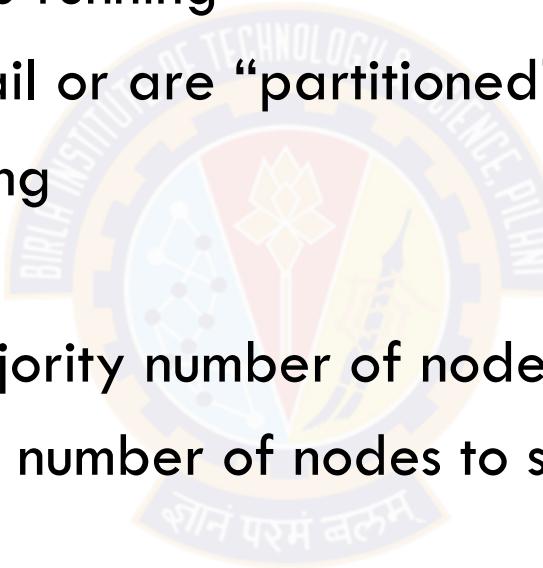


# Typical cluster components



# Split Brain – Evil of clustering

- Caused by failure of Heartbeat network connection(s)
- Two “halves” of a cluster keep running
  - ✓ All Heartbeat networks fail or are “partitioned”
- Each half wants to keep running
- Recovery options:
  - ✓ Allow cluster half with majority number of nodes to survive
  - ✓ Force cluster with minority number of nodes to shut down



# Cluster Middleware - Some Functions

## Single System Image (SSI) infrastructure

- ✓ Glues together OSs on all nodes to offer unified access to system resources
  - ✓ Single process space
  - ✓ Cluster IP or single entry point
  - ✓ Single auth
  - ✓ Single memory and IO space
  - ✓ Process checkpointing and migration
  - ✓ Single IPC space
  - ✓ Single fs root
  - ✓ Single virtual networking
  - ✓ Single management GUI

## High Availability (HA) Infrastructure

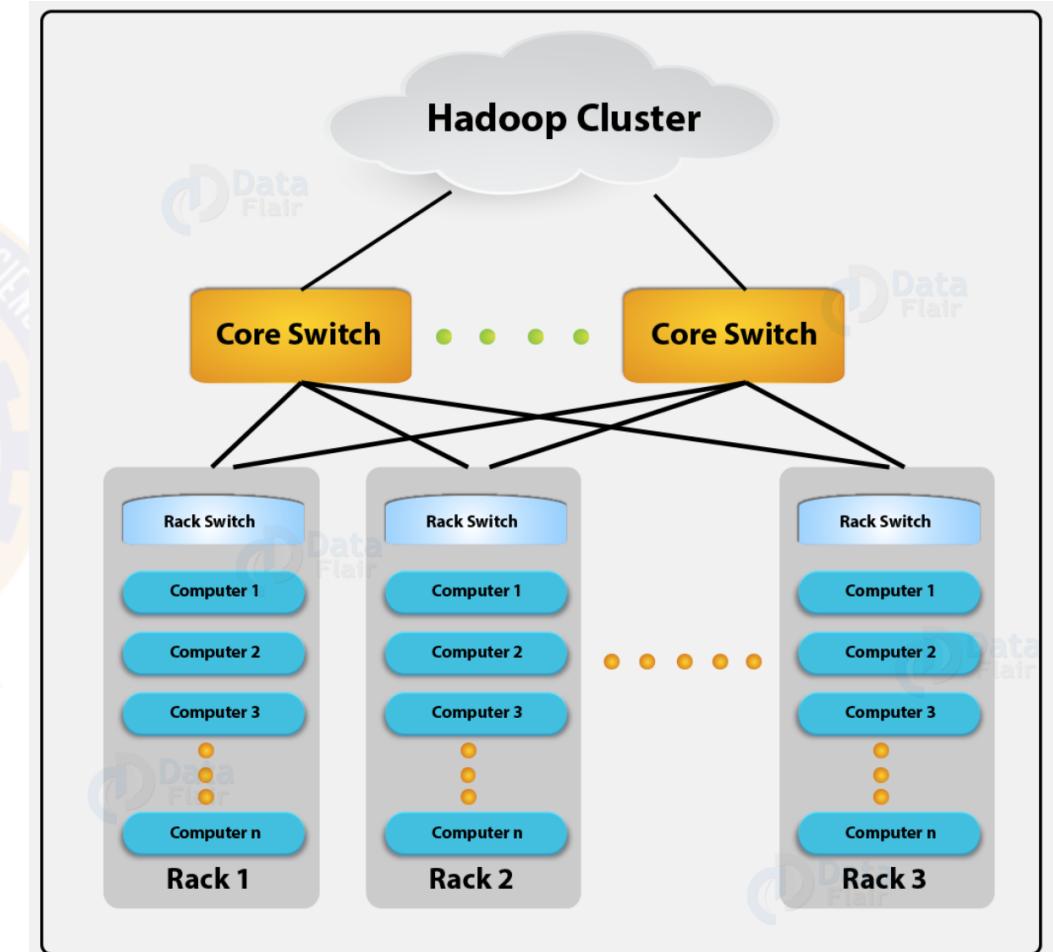
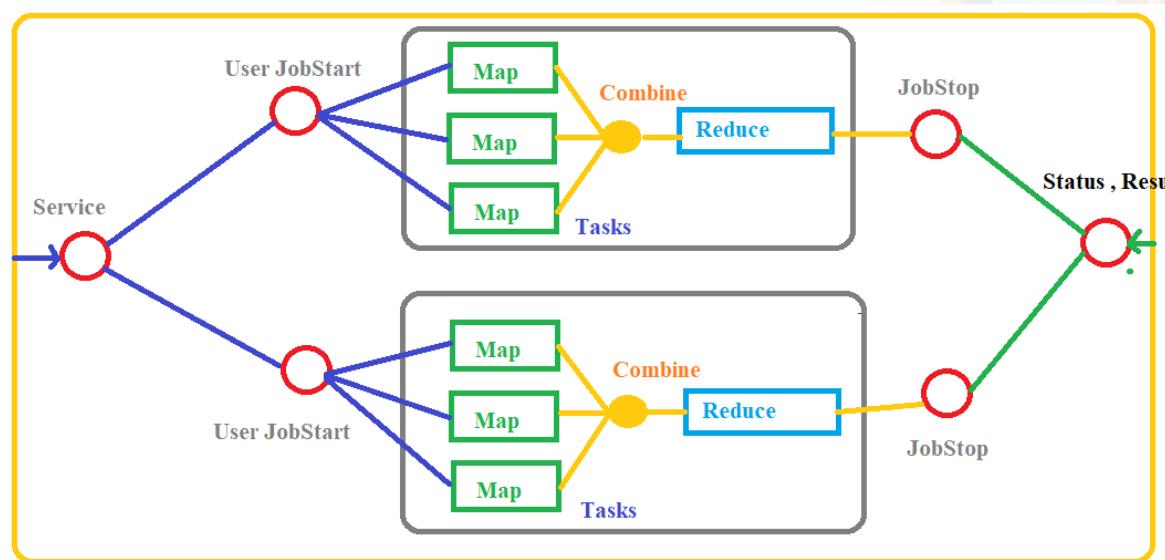
- ✓ Cluster services for
  - ✓ Availability
  - ✓ Redundancy
  - ✓ Fault-tolerance
  - ✓ Recovery from failures



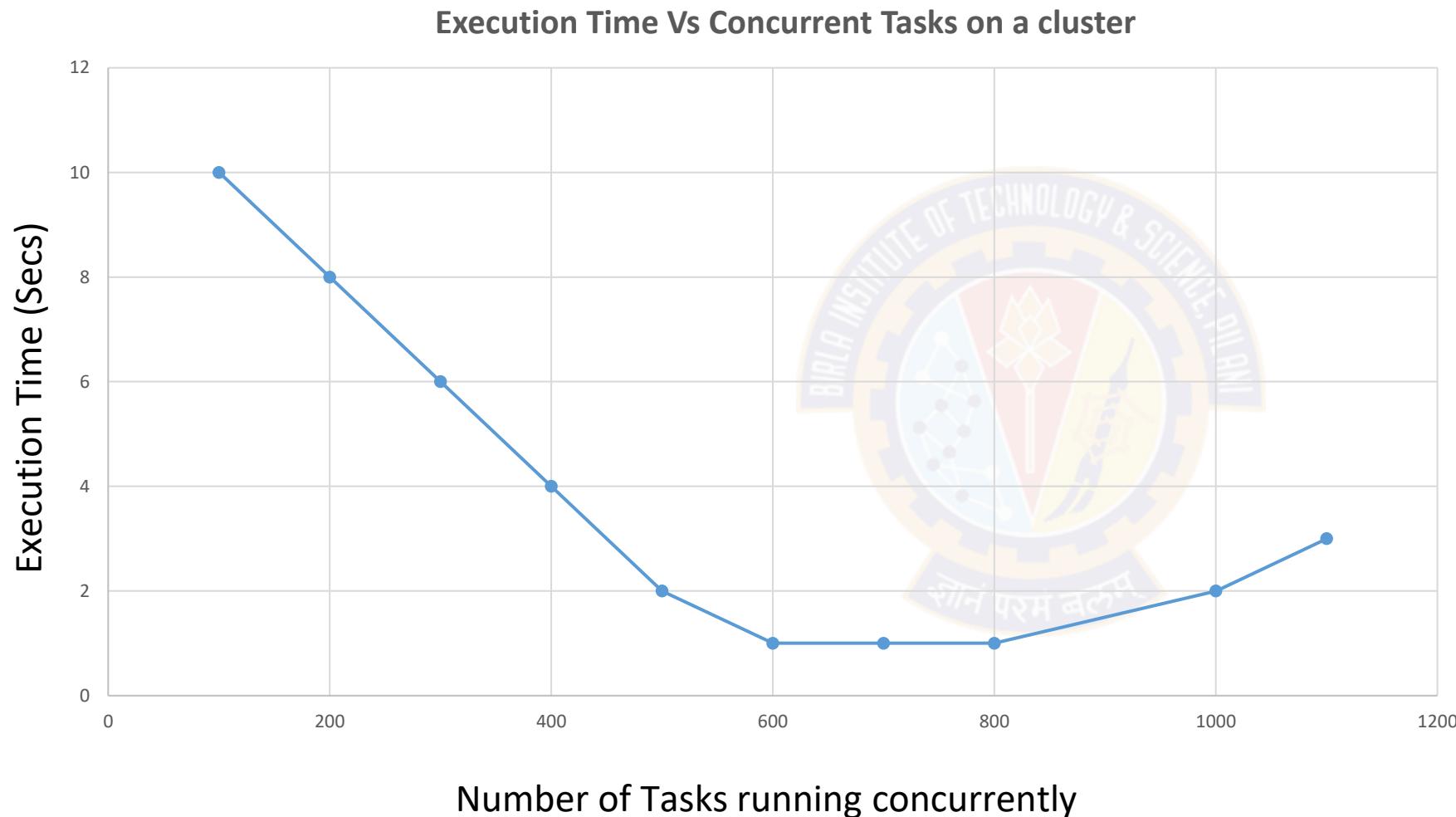
<http://www.cloudbus.org/papers/SSI-CCWhitePaper.pdf>

# Example cluster: Hadoop

- A job divided into tasks
- Considers every task either as a Map or a Reduce
- Tasks assigned to a set of nodes (cluster)
- Special control nodes manage the nodes for resource management, setup, monitoring, data transfer, failover etc.
- Hadoop clients work with these control nodes to get the job done



# Limits of Parallelism in distributed computing



## Overheads for

- ✓ Distributed Scheduling
- ✓ Local on node scheduling
- ✓ Communication
- ✓ Synchronization, etc.

# Summary

- Motivation and classification of parallel systems
- Computing limits of speedup
- Message passing
- How replication, partitioning helps in Big Data storage and access
- Cluster computing basics





Next Session:  
**Big Data Analytics and Systems**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 3 – FT, Big Data Analytics and Systems

---

Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **Distributed Computing – Reliability and Availability**
- Analytics
  - Definitions
  - Maturity model
  - Types
- Big Data Analytics
  - Characterization
  - Adoption challenges
  - Requirements
  - Technology challenges
  - Popular technologies - Hadoop, Spark



Will help you to pitch a Big Data project proposal

Will help you to build the system

# Distributed computing – living with failures

- Fault Tolerance or Graceful Degradation is the property that enables a system to continue operating properly in the event of failure of (one or more faults within) some of its component
- Failures of nodes and links is a common concern in Distributed Systems
  - Essential to have fault tolerance aspect in design
- Fault-Tolerant Systems - Ideally systems capable of executing their tasks correctly regardless of either HW failures or SW errors
- Fault tolerance is a measure of
  - How a distributed system functions in the presence of failures of system components
  - Tolerance of component faults is measured by 2 parameters
    - Reliability - An inverse indicator of failure rate
      - How soon a system will fail
    - Availability - An indicator of fraction of time a system is available for use
      - System is not available during failure

# The Nines of Availability



## The Nines of Availability

Availability percentages vs service downtime

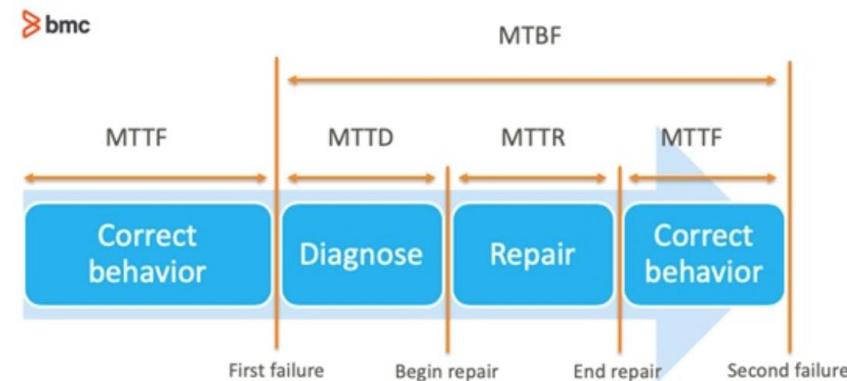
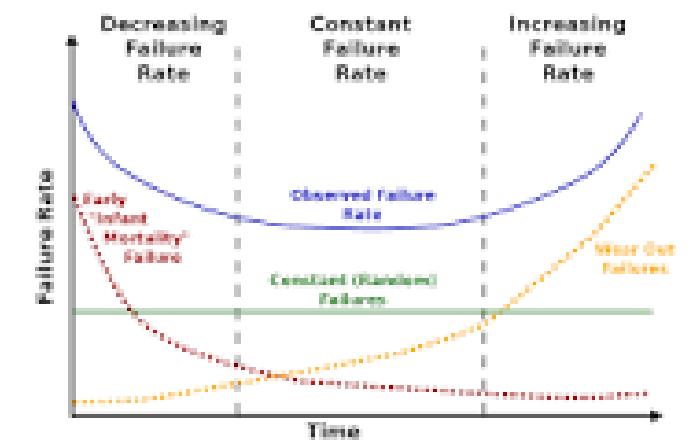
Availability %	Downtime per year	Downtime per month	Downtime per week
90% (one nine)	36.5 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (three nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.56 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% (seven nines)	3.15 seconds	0.259 seconds	0.0605 seconds

# The 9s Game – Different platforms

NonStop Integrity	99.9999
Mainframe	99.999
OpenVMS	99.998
AS400	99.998
HPUX	99.996
Tru64	99.996
Solaris	99.995
NT Cluster	99.992 - 99.995

# Metrics

- MTTF - Mean Time To Failure
  - $MTTF = 1 / \text{failure rate} = \text{Total \#hours of operation} / \text{Total \#units}$
  - MTTF is an averaged value. In reality, failure rate changes over time because it may depend on age of component.
- Failure rate =  $1 / MTTF$  (assuming average value over time)
- MTTR - Mean Time to Recovery / Repair
  - $MTTR = \text{Total \#hours for maintenance} / \text{Total \#repairs}$
- MTTD - Mean Time to Diagnose
- MTBF - Mean Time Between Failures
  - $MTBF = MTTD + MTTR + MTTF$



# Availability

- Availability = Time system is UP and accessible / Total time observed
- Availability = MTTF / (MTTD\* + MTTR + MTTF)
  - or
- Availability = MTTF / MTBF
- A system is highly available when
  - MTTF is high
  - MTTR is low

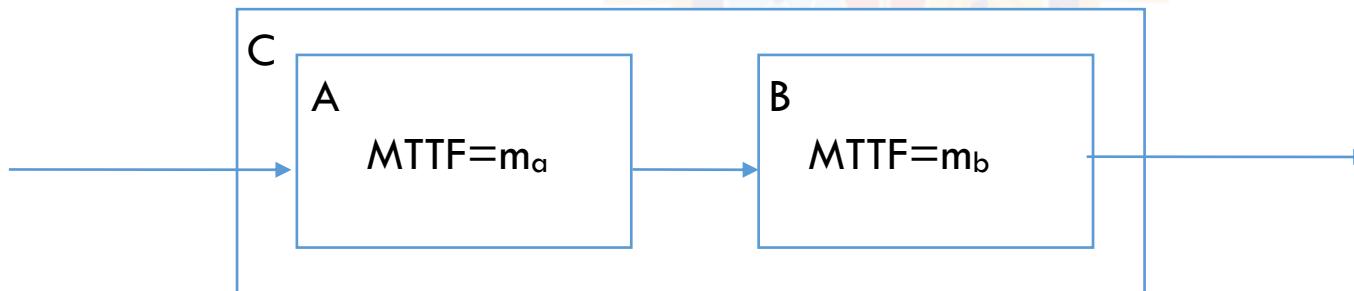


\* Unless specified one can assume MTTD = 0

[availabilitydigest.com](http://availabilitydigest.com)

# Reliability - serial assembly

- MTTF of a system is a function of MTTF of components
- Serial assembly of components
  - Failure of any component results in system failure
  - Failure rate of C = Failure rate of A + Failure rate of B =  $1/m_a + 1/m_b$
  - MTTF of system =  $1 / \text{SUM } (1/\text{MTTF}_i)$  for all components i
  - Failure rate of system =  $\text{SUM}(1/\text{MTTF}_i)$  for all components i

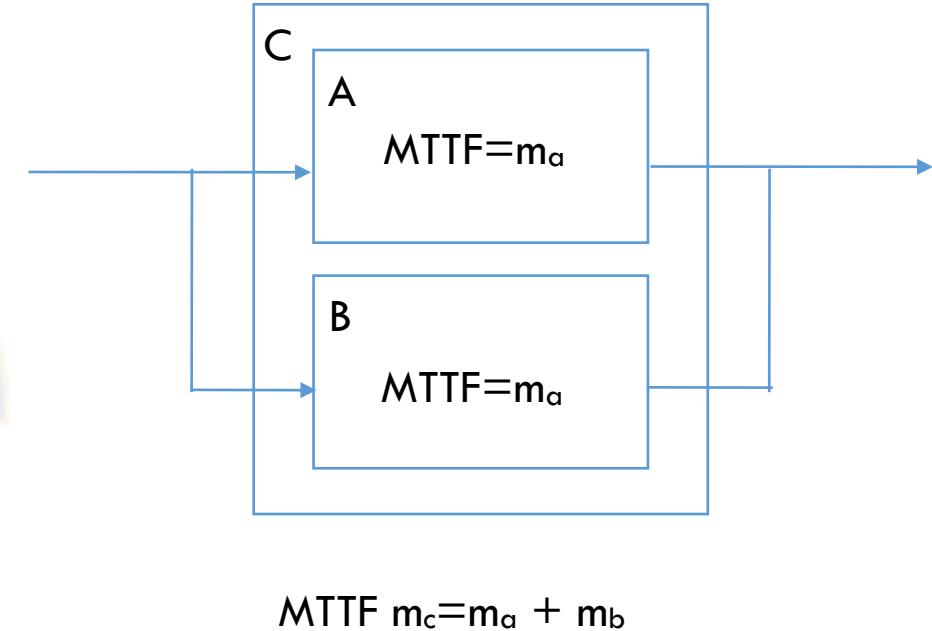
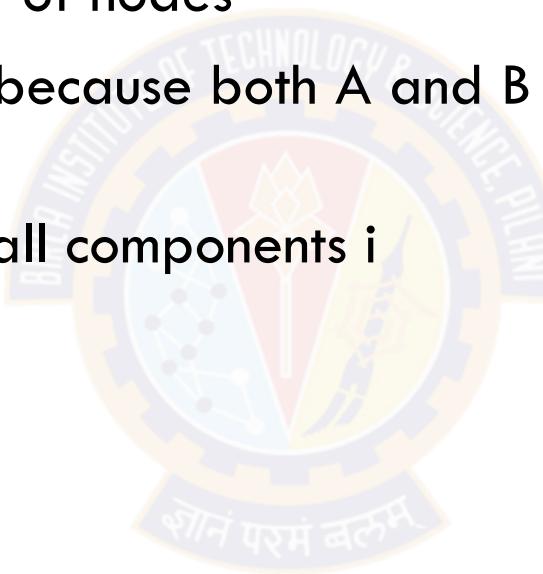


- The combined availability of system C is:

$A_c = A_a A_b$  , Where  $A_a$  is the availability of component A and  $A_b$  is the availability of component B

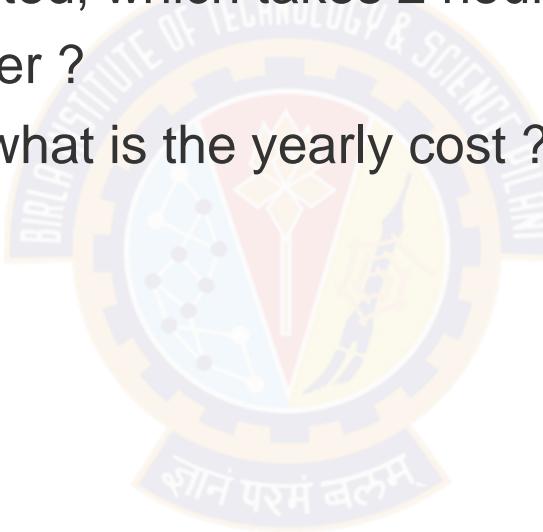
# Reliability - parallel assembly

- In a parallel assembly, e.g. a cluster of nodes
  - MTTF of C = MTTF A + MTTF B because both A and B have to fail for C to fail
  - MTTF of system =  $\text{SUM}(\text{MTTF}_i)$  for all components i



# Example

- A node in a cluster fails every 100 hours while other parts never fail.
  - On failure of the node the whole system needs to be shutdown, faulty node replaced and system. This takes 2 hours.
  - The application needs to be restarted, which takes 2 hours.
  - What is the availability of the cluster ?
  - If downtime is \$80k per hour, the what is the yearly cost ?
- 
- Solution
    - MTTF = 100 hours
    - MTTR =  $2 + 2 = 4$  hours
    - Availability =  $100/104 = 96.15\%$
    - Cost of downtime per year =  $80000 \times 3.85 \times 365 \times 24 / 100 = \text{USD } 27 \text{ million}$

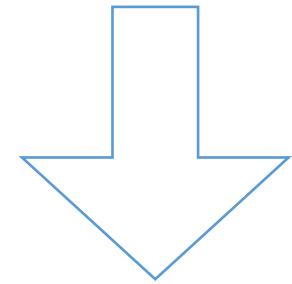


[https://www.brainkart.com/article/Fault-Tolerant-Cluster-Configurations\\_11320/](https://www.brainkart.com/article/Fault-Tolerant-Cluster-Configurations_11320/)

# Fault tolerance configurations - standby options

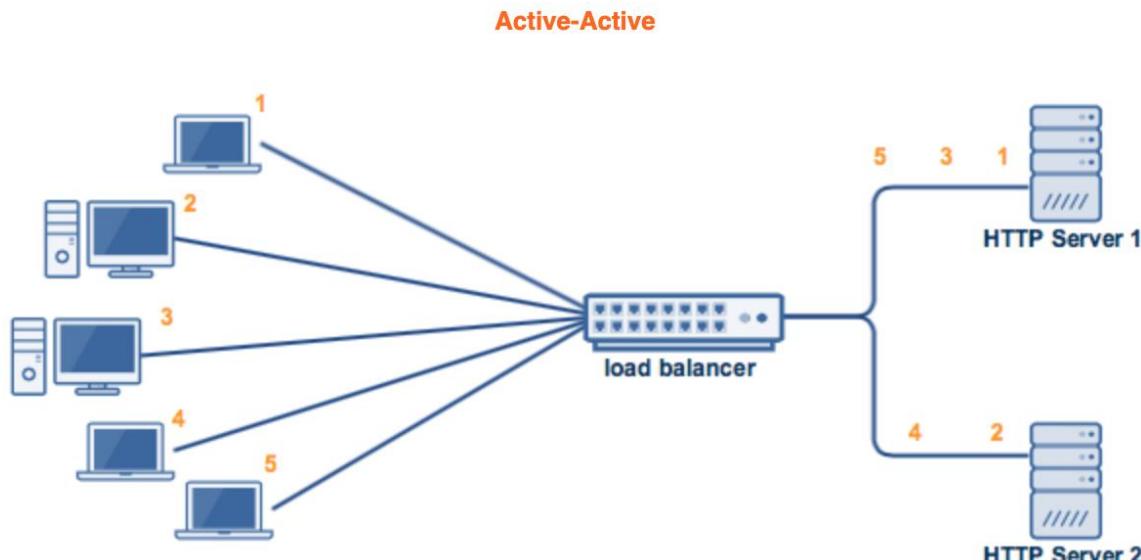
High availability model	Secondary node behavior	Data protection	Failover time
Load-balanced	Both the primary node and the secondary node are active and they process system requests in parallel.  <b>aka active-active</b>	Data replication is bidirectional and is performed based on the software capabilities.	Zero failover time
Hot standby	The software component is installed and available on both the primary node and the secondary node. The secondary system is up and running, but it does not process data until the primary node fails.  <b>aka active-passive</b>	Data is replicated and both systems contain identical data. Data replication is performed based on the software capabilities.	A few seconds
Warm standby	The software component is installed and available on the secondary server, which is up and running. If a failure occurs on the primary node, the software components are started on the secondary node. This process is automated by using a cluster manager.	Data is regularly replicated to the secondary system or stored on a shared disk.	A few minutes
Cold standby	A secondary node acts as the backup for an identical primary system. The secondary node is installed and configured only when the primary node breaks down for the first time. Later, in the event of a primary node failure, the secondary node is powered on and the data is restored while the failed component is restarted.	Data from a primary system can be backed up on a storage system and restored on a secondary system when it is required.	A few hours

Decreasing cost  
vs  
Increasing MTTR

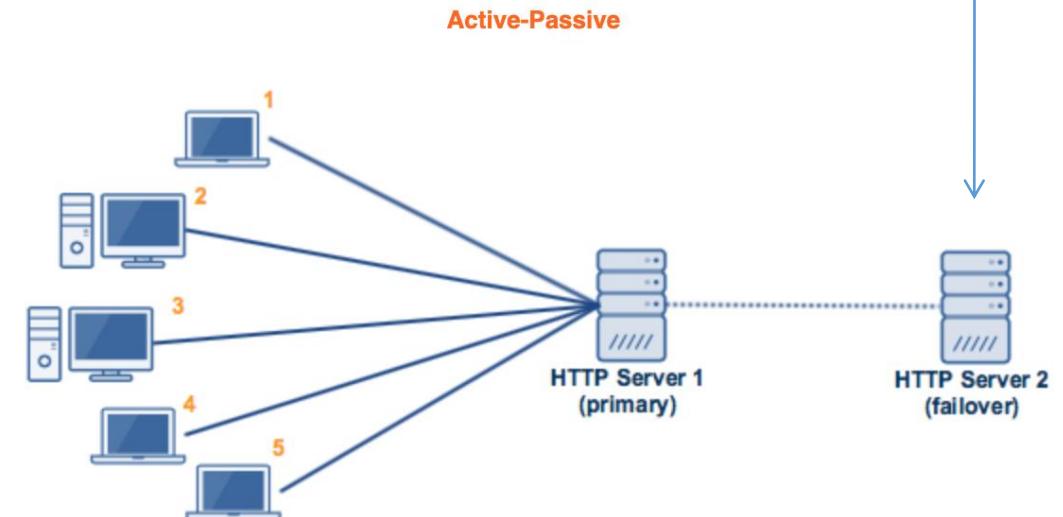


Assumption:  
Same software bug or runtime fault will not recur in the standby

# Fault tolerance configurations - standby options

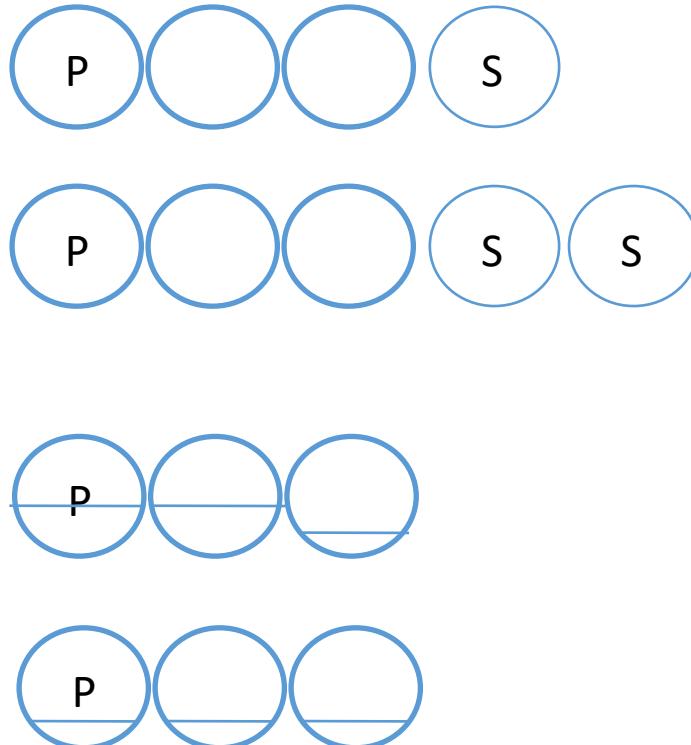


Warm and cold  
depends on how the  
passive / failover node  
is kept updated



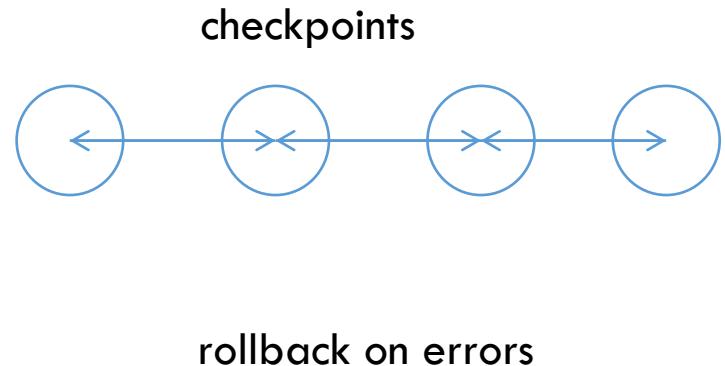
# Fault tolerance configurations - cluster topologies

- N+1
  - One node is configured to take the role of the primary
- N+M
  - M standby nodes if multiple failures are anticipated especially when running multiple services in the cluster
- N to 1
  - The secondary failover node is a temporary replacement and once primary is restored, services must be reactivated on it
- N to N
  - When any node fails, the workload is redistributed to the remaining active nodes. So there is no special standby node.



# Fault Tolerant Clusters – Recovery

- Diagnosis
  - Detection of failure and location of the failed component, e.g. using heartbeat messages between nodes
- Backward recovery
  - periodically do a checkpoint (save consistent state on stable storage)
  - on failure, isolate the failed component, rollback to last checkpoint and resume normal operation
  - Ease to implement, independent of application, but leads to wastage of execution time on rollback besides unused checkpointing work
- Forward recovery
  - In real-time systems or time-critical systems cannot rollback. So state is reconstructed on the fly from diagnosis data.
  - Application specific and may need additional hardware



# Topics for today

- Distributed Computing – Reliability and Availability
- **Analytics**
  - Definitions
  - Maturity model
  - Types
- Big Data Analytics
  - Characterization
  - Adoption challenges
  - Requirements
  - Technology challenges
  - Popular technologies - Hadoop, Spark



Will help you to pitch a Big Data project proposal

Will help you to build the system

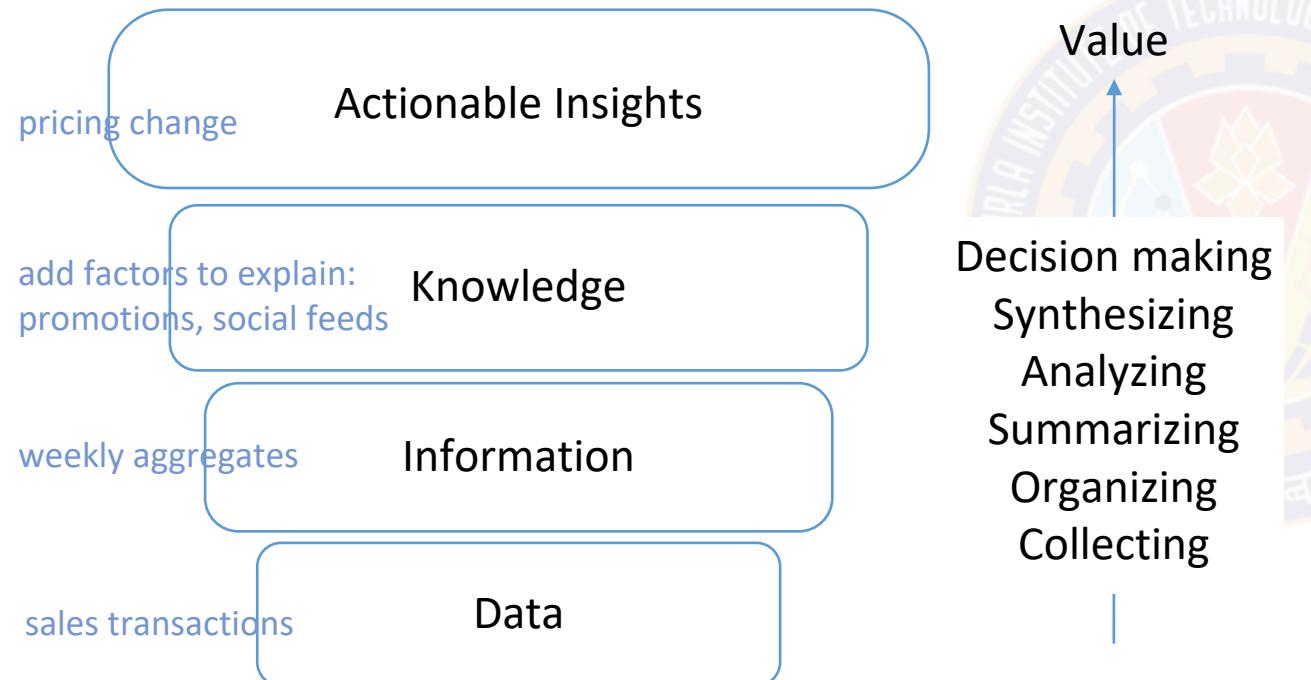
# Analytics - Definitions

- Extensive use of data, statistical and quantitative analysis, explanatory and predictive models, and fact based management to drive decisions and actions
- Purpose
  - ✓ To unearth hidden patterns
    - ✓ 2 / 100 stores had no sales for a promotion item because it was not in the right shelf
  - ✓ To decipher unknown correlations
    - ✓ The famous “Beer and diapers” story
  - ✓ Understand the rationale behind trends
    - ✓ What do users like about a popular product that has growing sales
  - ✓ Mine useful business information
    - ✓ Popular items during specific holiday sales
- Helps in
  - ✓ Effective marketing
  - ✓ Better customer service and satisfaction
  - ✓ Improved operational efficiency
  - ✓ Competitive advantage over rivals



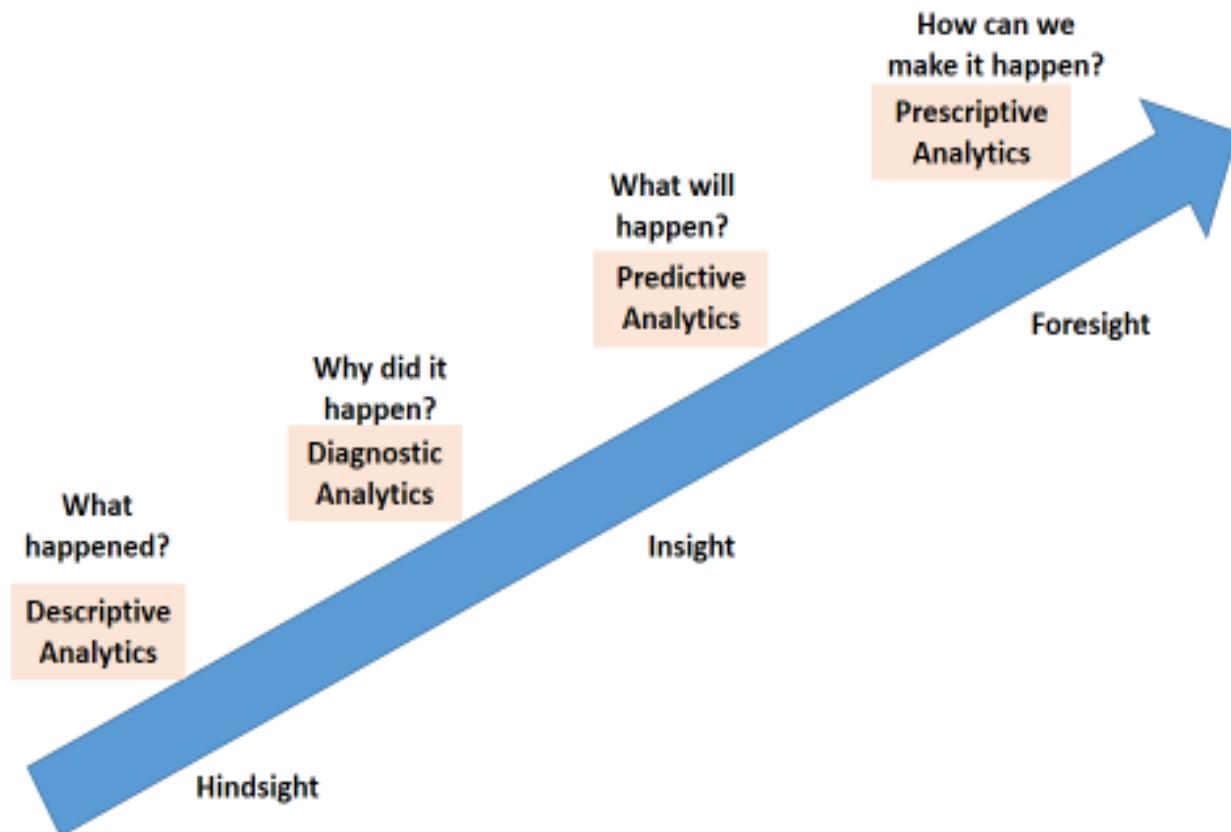
# Process of Analysis

## Transformation of Data



- Apply functions / transformations on data to get to the next level till it is actionable insight useful for the business
- Keep attaching more meta-data for context and make it meaningful

# Analytics 1.0, 2.0, 3.0



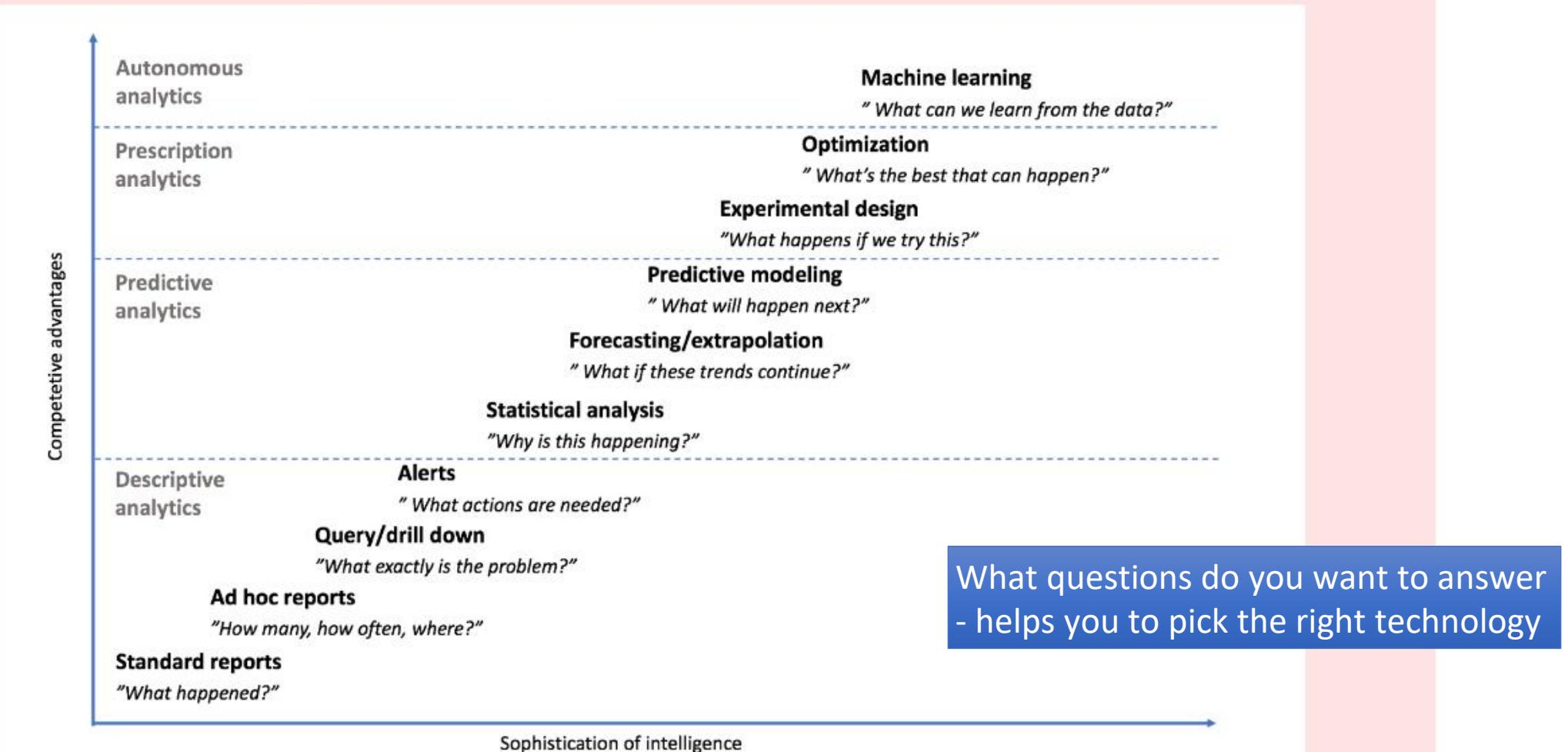
Analytics 1.0 → deals with historical data to report on events, occurrences of the past.

Analytics 2.0 → helps to predict what will happen in future

Analytics 3.0 → is about predicting what will happen and to best leverage the situation when it happens.

# Analytics Maturity and Competitive Advantage

## Analytics Maturity & Competitive Advantage



source: Competing on Analytics, Thomas Davenport

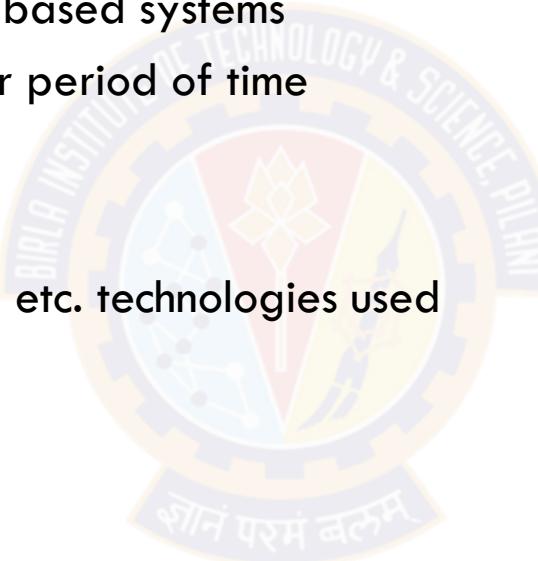
# Types of Analytics - Descriptive

- Provides ability to alert, explore and report using mostly internal and external data
- Business Intelligence (BI) or Performance reporting
- Provides access to historical and current data
- Reports on events, occurrences of the past
- Usually data from legacy systems, ERP, CRM used for analysis
- Based on relational databases and warehouse
- Structured and not very large data sets
- Sometimes also referred as Analytics 1.0
- Era : mid 1950s to 2009
- Questions asked
  - ✓ What happened?
  - ✓ Why did it happen? (Diagnostic analysis)

E.g. number of infections is significantly higher this month than last month and highly correlated with factor X

# Types of Analytics - Predictive

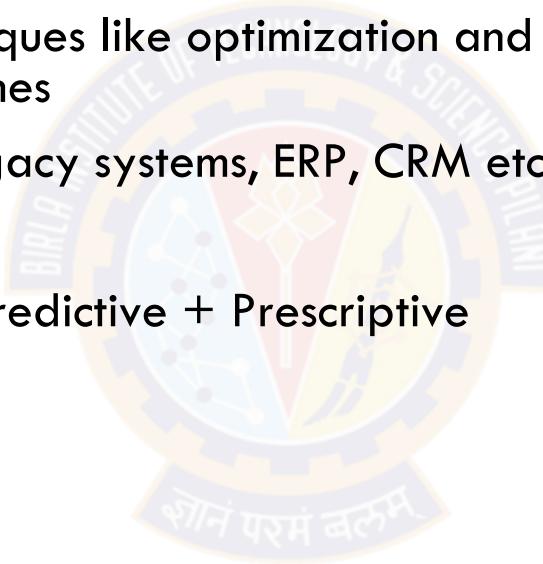
- Uses past data to predict the future
- Uses quantitative techniques like segmentation, forecasting etc. but also makes use of descriptive analytics for data exploration
- Uses technologies like models and rule based systems
- Based on large data set gathered over period of time
- Externally sourced data also used
- Unstructured data may be included
- Hadoop clusters, SQL on Hadoop data etc. technologies used
- aka Analytics 2.0
- Era : from 2005 to 2012
- Key questions
  - ✓ What will happen?
  - ✓ Why it will happen?
  - ✓ When will it happen?



E.g. number of infections will reach X in month Y and likely cause will be Z

# Types of Analytics - Prescriptive

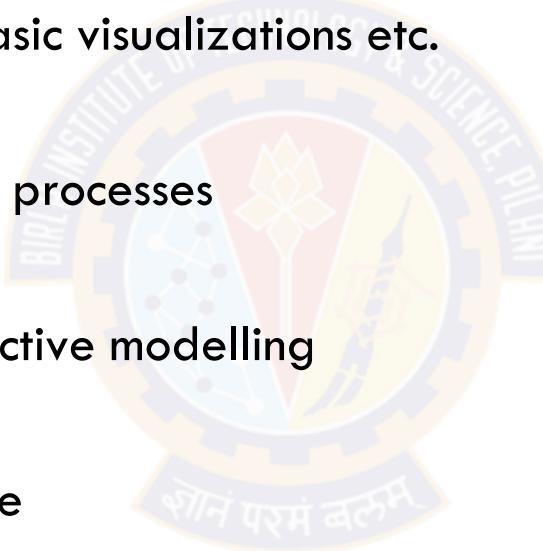
- Uses data from past to make prophecies of future and at the same time make recommendations to leverage the situation to one's advantage
- Suggests optimal behaviors and actions
- Uses a variety of quantitative techniques like optimization and technologies like models, machine learning and recommendations engines
- Data is blend from Big data and legacy systems, ERP, CRM etc.
- In-memory analysis etc.
- Aka Analytics 3.0 = Descriptive + Predictive + Prescriptive
- post 2012
- Questions:
  - ✓ What will happen
  - ✓ When will it happen
  - ✓ Why will it happen
  - ✓ What actions should be taken



E.g. number of infections will reach X in month Y and likely cause will be Z. W is the best recommended action to keep the number in month Y below X/2.

# Alternative categorisation

- Basic Analytics
  - ✓ Slicing and dicing of data to help with basic insights
  - ✓ Reporting on historical data, basic visualizations etc.
- Operationalized Analytics
  - ✓ Analytics integrated in business processes
- Advanced Analytics
  - ✓ Forecasting the future by predictive modelling
- Monetized Analytics
  - ✓ Used for direct business revenue



# Topics for today

- Distributed Computing – Reliability and Availability
- Analytics
  - Definitions
  - Maturity model
  - Types
- **Big Data Analytics**
  - **Characterization**
  - **Adoption challenges**
  - **Requirements**
  - **Technology challenges**
  - **Popular technologies - Hadoop, Spark**



Will help you to pitch a Big Data project proposal

Will help you to build the system

# Big Data Analytics

Working with datasets with huge volume, variety and velocity beyond storage and processing capability of RDBMS

Uses Principle Of Locality to move code near to Data

IT's collaboration with business users and Data Scientists

Better , Faster decision in real time

Richer, faster insights into customers, partners and business

Competitive Advantage

Technology enabled Analytics

Support for both batch and stream processing of data

What makes you think about this differently ?

# What Big Data Analytics is not

‘One size fit all!’ solution based on RDBMS with shared disk and memory

Only meant for big data companies

Only Volume Game

Big Data Analytics is not

Meant to replace Data warehouse\*

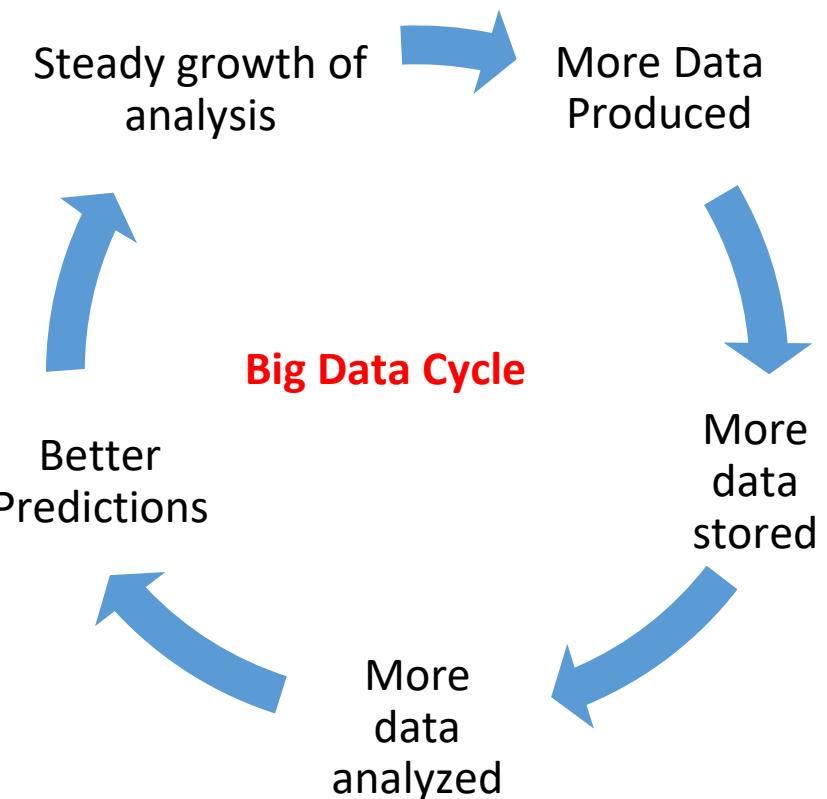
Just bothered about Technology

Meant to replace RDBMS

Things to know to avoid friction in Big Data projects

# Why the sudden hype ?

- Data is growing at 40% compound annual rate
  - ✓ 45 ZB in 2020
  - ✓ In 2010, 1.2 trillion Gigabytes data generated
  - ✓ In 2012, reached to 2.4 trillion Gigabytes
  - ✓ Volume of world wide data expected to double every 1.2 years
  - ✓ Every day 2.5 quintillion bytes of data is created
  - ✓ 90% of today's data is generated in last few years only!
  - ✓ Walmart processes one million customer transaction per hour
  - ✓ 500 million "tweets" are posted by users every day
  - ✓ 2.7 billion "likes" and comments by Facebooks users per day
- Cost of storage has hugely dropped
- Large number of user friendly analytics tools available for data processing



# Adoption Challenges in Organizations

- Obtaining executive sponsorship for investments in big data and its related activities
- Getting business units to share data / information across organizational silos
- Finding right skills (Business Analysts/Data Scientists and Data Engineers) that can manage large amount of variety of data and create insights from it
- Determining approach to scale rapidly and elastically , address storage and processing of large volume, velocity and variety of Big data
- Deciding whether to use structured or unstructured, internal or external data to make business decisions
- Choosing optimal way to report findings and analysis of big data
- Determining what to do with the insights created from big data

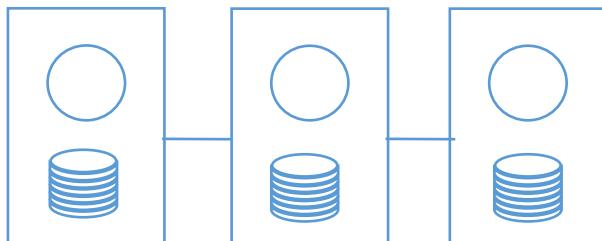
# Requirements of Big Data analytics

- Cheap abundant storage
- Processing options
  - batch / streaming,
  - disk based / memory based
- Open source platforms, e.g. Hadoop, Spark
- Parallel and distributed systems with high throughput rather than low latency
- Cloud or other flexible resource allocation arrangements
  - Flexibility to setup and tear down infrastructure for quick projects across various teams

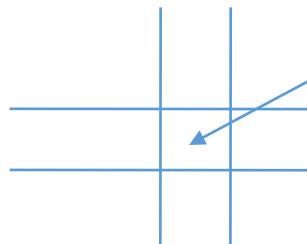


# Technology challenges (1)

- Scale
  - ✓ Need is to have storage that can best withstand large volume, velocity and variety of data
  - ✓ Scale vertically / horizontally ?
  - ✓ RDBMS / NoSQL ?
  - ✓ How does compute scale with storage - coupled or de-coupled, i.e. good idea to put common nodes for compute and storage
- Security \*
  - ✓ Most of recent NoSQL big data platforms have poor security mechanisms, e.g. challenges:
    - ✓ Fine grain control in semi-structured data, esp with columnar storage
    - ✓ Options for inconsistent data complicate matters
    - ✓ Larger attack surface across distributed nodes
    - ✓ Often encryption is turned off for performance
  - ✓ Lack of authorization techniques while safeguarding big data
  - ✓ May contain PII data (personally identifiable info)



*compute + data on same node: locality helps,  
but does compute scale with storage ?*



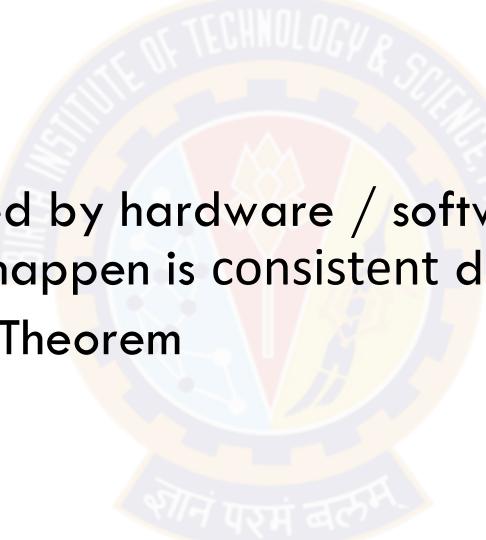
*Easier in RDBMS to control at row /  
column / cell level with always  
consistent values in a tightly coupled  
system*

# Technology challenges (2)

- Schema
  - ✓ Need is to have dynamic schema, static / fixed schemas don't fit
- Continuous availability
  - ✓ Needs 24 \* 7 \* 365 support as data is continuously getting generated and needs to be processed
  - ✓ Almost all RDBMS, NoSQL big data platforms has some sort of downtime
    - ✓ Memory cleanup, replica rebalancing, indexing, ...
    - ✓ Most of the large scale NoSQL systems also need weekly maintenance

# Technology challenges (3)

- Consistency
  - ✓ Should one go for strict consistency or eventual consistency? Is this like social media comments or application needs consistent reads ?
- Partition Tolerant
  - ✓ When a system gets partitioned by hardware / software failures. How to build partition tolerant systems ? When faults happen is consistent data available ?
  - ✓ We will discuss options in CAP Theorem
- Data quality
  - ✓ How to maintain data quality – data accuracy, completeness, timeliness etc.?
  - ✓ Do we have appropriate metadata in place esp with semi/un-structured data ?



# Popular technologies

- How to manage voluminous, varied, scattered and high velocity data ?
  - ✓ Think beyond an RDBMS depending on use case but not necessarily to replace it
- Some popular technologies
  - ✓ Distributed and parallel processing (covered in session 2)
  - ✓ Hadoop (more details in session 6-9)
    - ✓ File based large scale parallel data processing tasks
  - ✓ In-memory computing (more details in session 11-14)
    - ✓ Usage of main memory (RAM) helps to manage data processing tasks faster
  - ✓ Big Data Cloud (more details in session 15)
    - ✓ Helps to save cost and better management of resources using a services model



# Introduction to Hadoop

# What problems does Hadoop solve

Storage of huge amount of data

## ✓ Problems

- ✓ Multiple partitions of data for parallel access but more systems means more failures
- ✓ Multiple nodes can make the system expensive
- ✓ Arbitrary data - binary, structured ...

## ✓ Solution

- ✓ Replication Factor (RF) for failures : Number of data copies of a given data item / data block stored across the network
- ✓ Uses commodity heterogenous hardware
- ✓ Multiple file formats

Processing the huge amount of data

## ✓ Problems

- ✓ Data is spread across systems, how to process it in quick manner?
- ✓ Challenge is to integrate data from different machines before processing

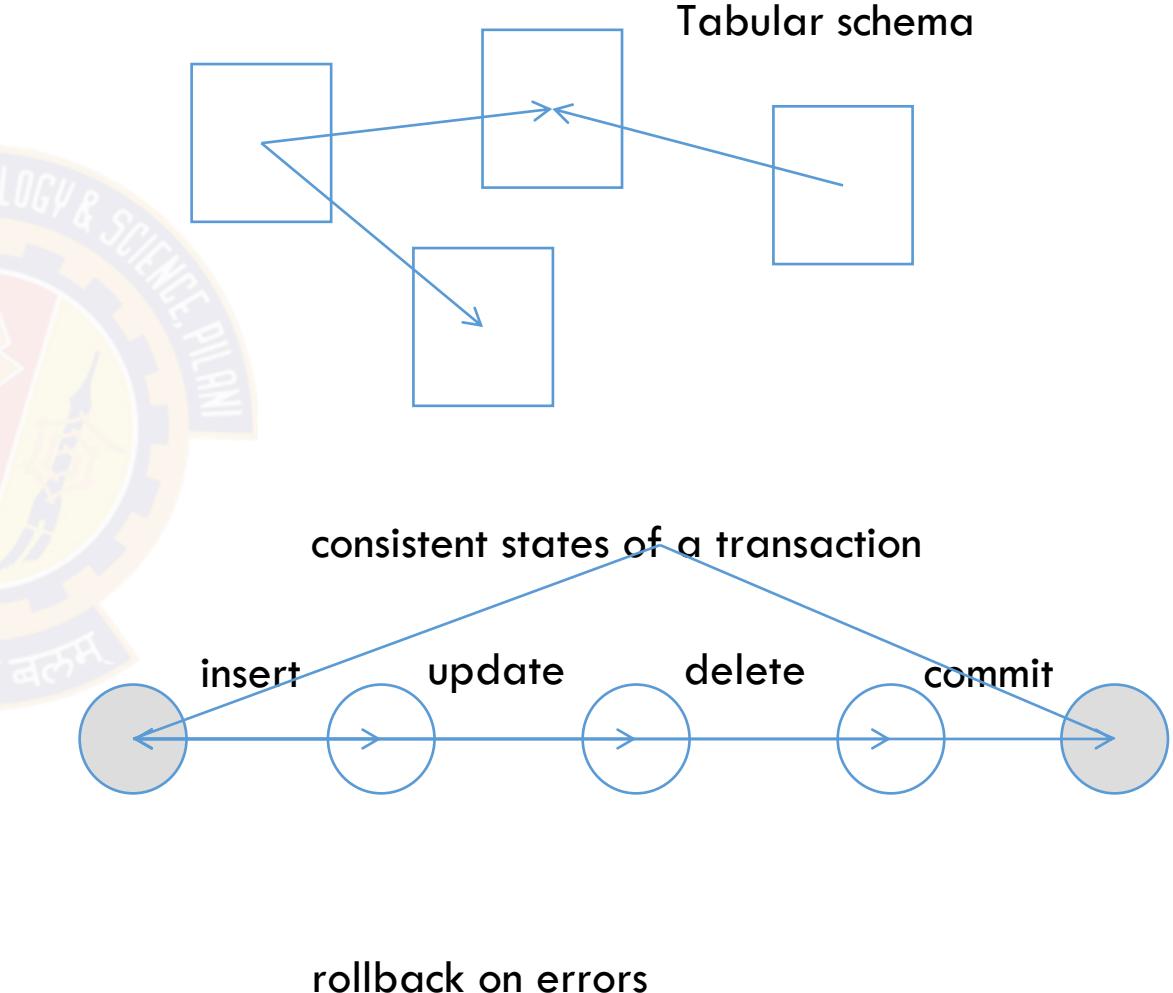
## ✓ Solution

- ✓ MapReduce programming model to process huge amount of data with high throughput
- ✓ Compute is close to storage for handling large data sets

# What's different from a Distributed Database

- **Distributed Databases**

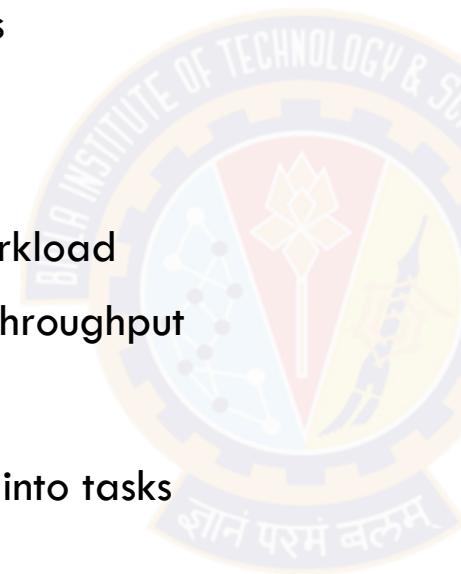
- Data model
  - Tables and relations
  - Schema is predefined (during write)
  - Supports partitioning
  - Fast indexed reads
  - Read and write many times
- Compute model
  - Generate notations of a transaction
  - ACID properties
  - Allow distributed transactions
  - OLTP workloads



# Hadoop in contrast ...

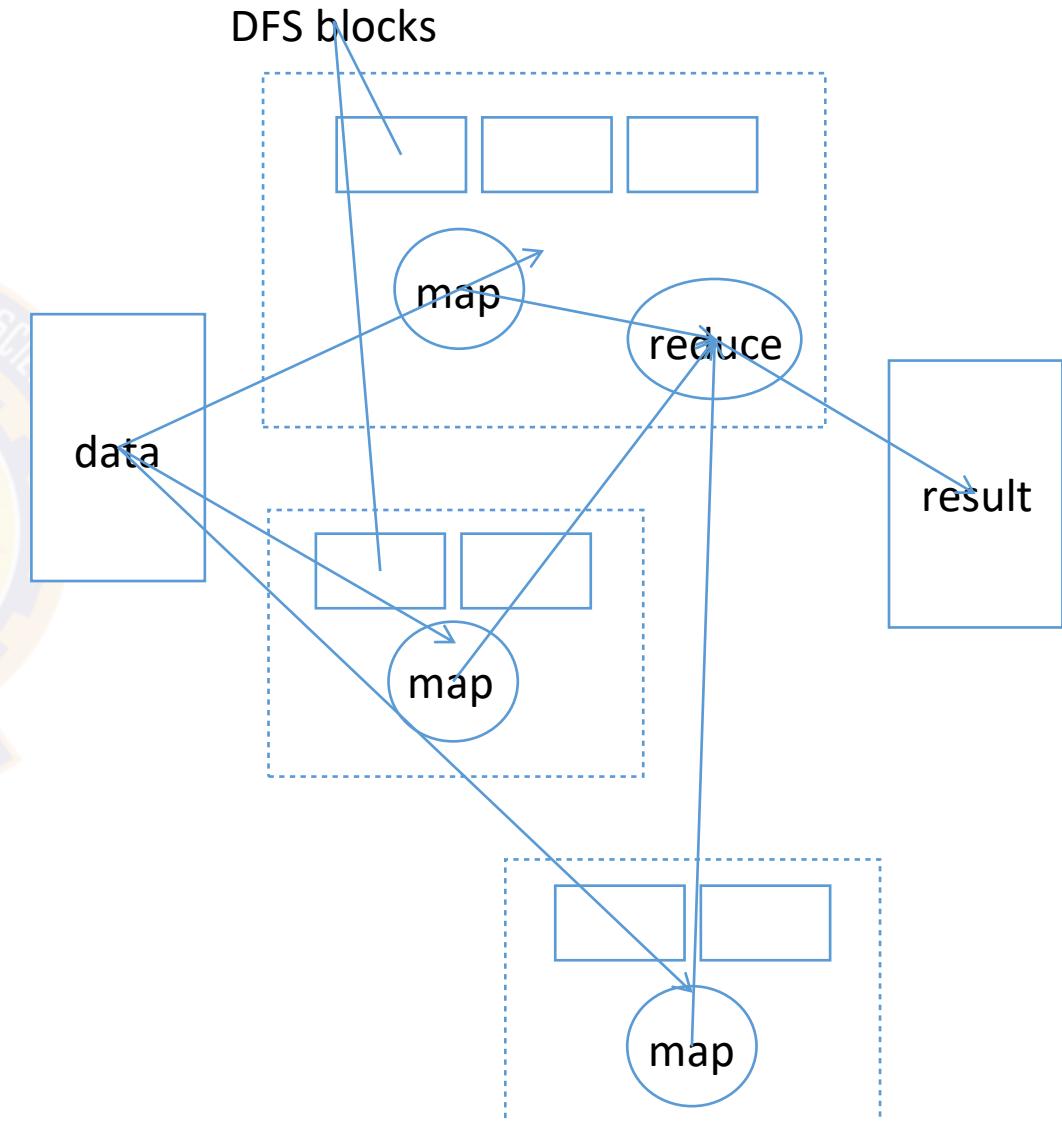
- Data model

- Flat files supporting multiple formats, including binary
- No pre-defined schema (i.e. during write)
- Divides files automatically into blocks
- Handles large files
- Optimized for write
- Write once and read many times workload
- Meant for scan workloads with high throughput



- Compute model

- Generate notations of a job divided into tasks
- MapReduce compute model
- Every task is a map or a reduce
- High latency analytics, data discovery workloads



# Versions of Hadoop

Hadoop 1.0
<b>MapReduce</b> (Cluster Resource Manager & Data Processing)

Hadoop 2.0	
<b>MapReduce</b> (Data Processing)	<b>Others</b> (Data Processing)
YARN (Cluster Resource Manager)	
<b>HDFS</b> (redundant, reliable storage)	



# Hadoop Distributions

Cloudera

CDH 4.0

CDH 5.0

Hortonworks

HDP 1.0

HDP 2.0

MAPR

M3

M5

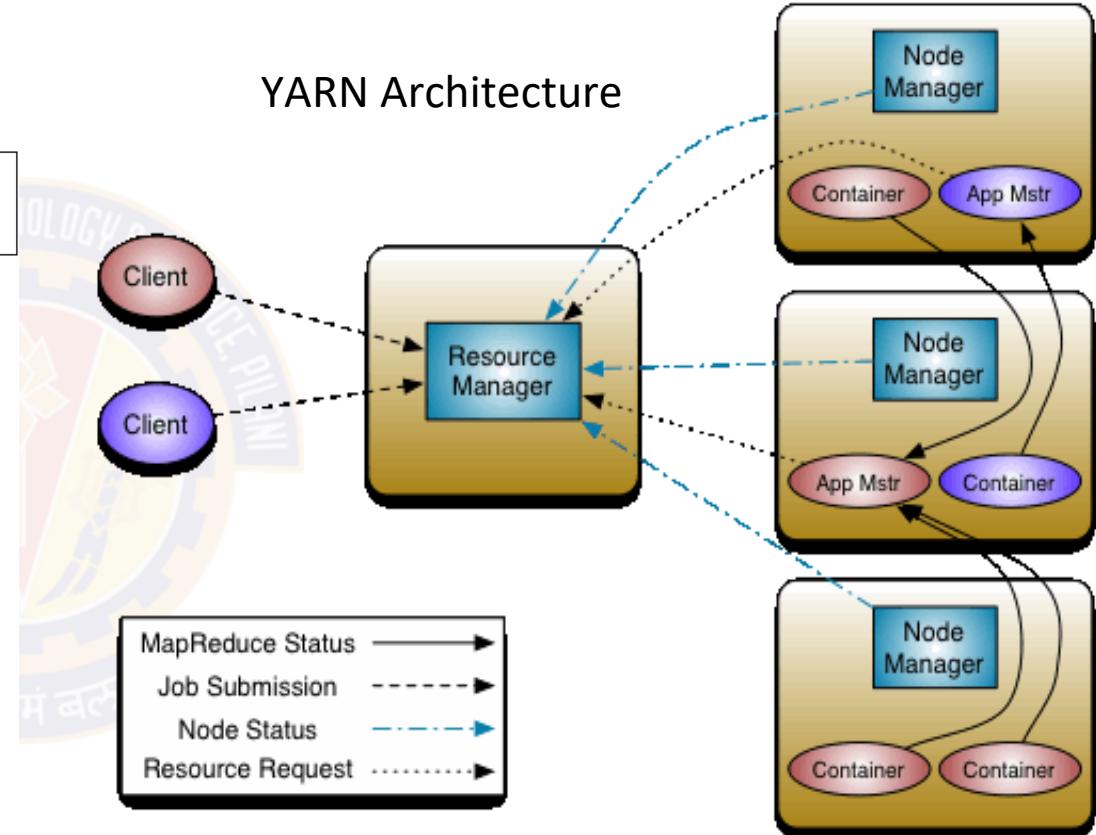
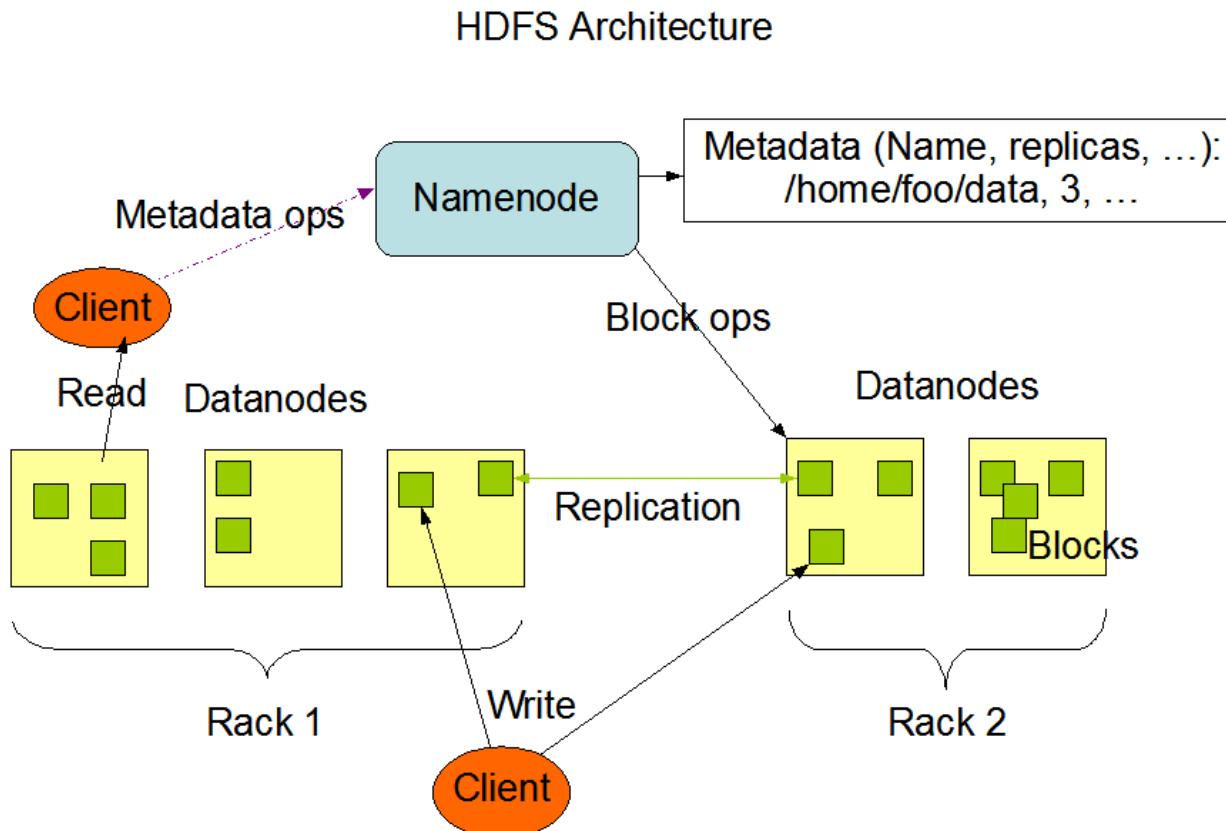
M8

Apache Hadoop

Hadoop 1.0

Hadoop 2.0

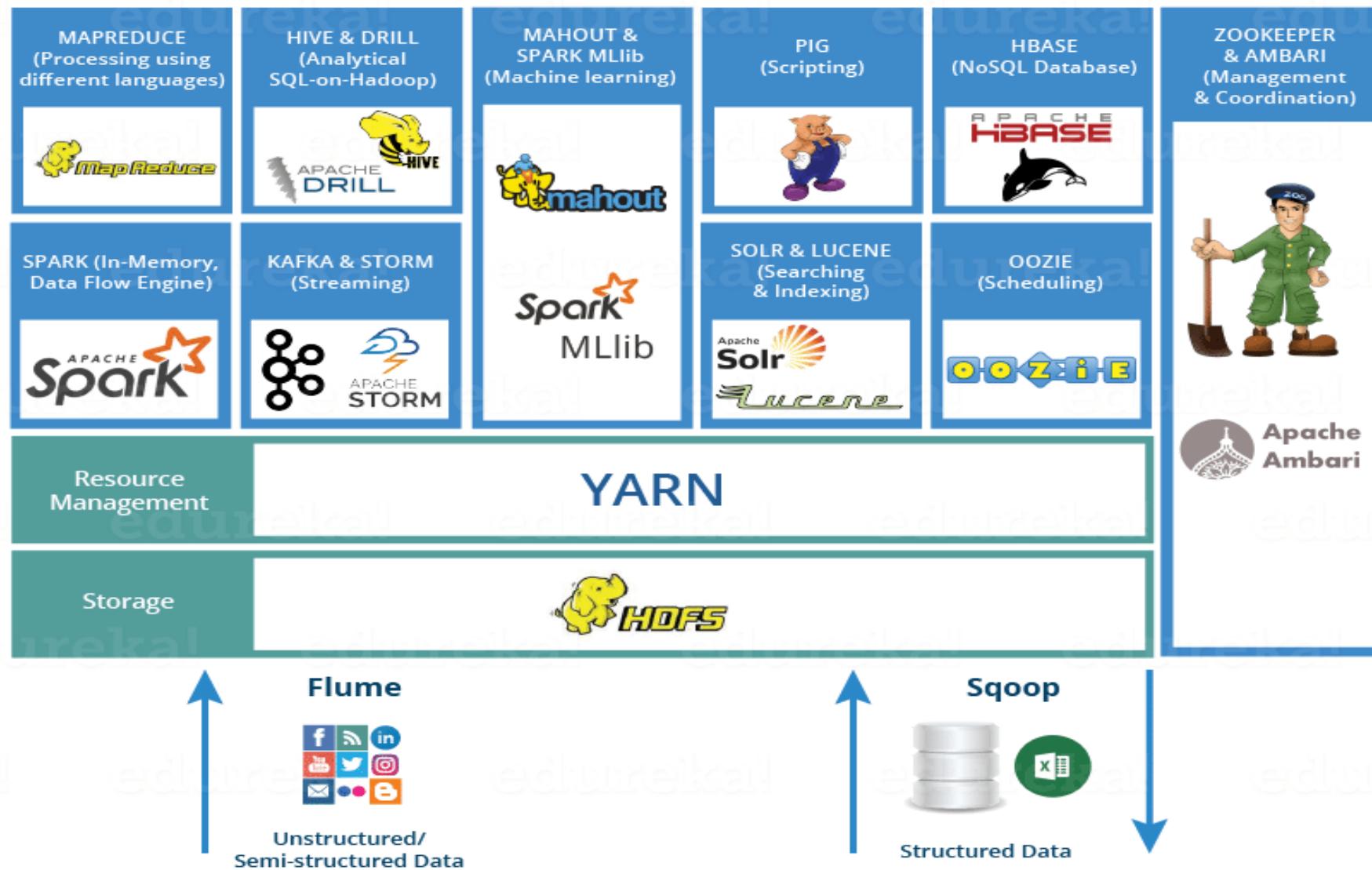
# Hadoop 2.0 high level architecture



# Advantages of using Hadoop

- Low cost – open source and low cost commodity storage
- Computing power – many nodes can be used for computation
- Scalability – simple to add nodes in system for parallel processing and storage
- Storage Flexibility – can store unstructured data easily
- Inherent data protection – protects against hardware failures

# Hadoop ecosystem



# Hadoop Ecosystem Components

**Components that help with Data Ingestion are:**

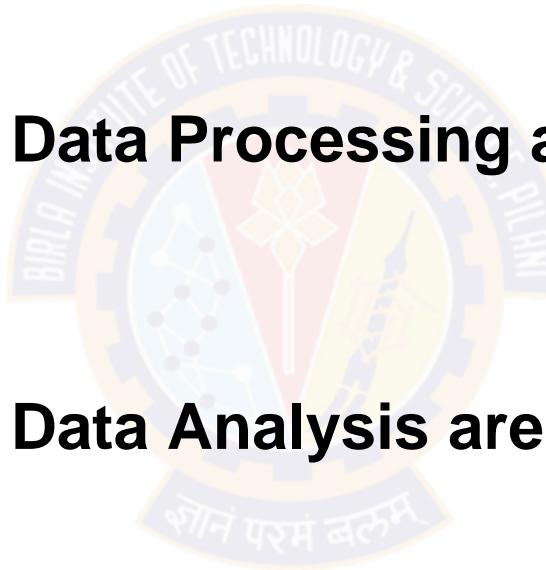
1. Sqoop
2. Flume

**Components that help with Data Processing are:**

1. MapReduce
2. Spark

**Components that help with Data Analysis are:**

1. Pig
2. Hive
3. Impala



# Hadoop Ecosystem Components for Data Ingestion

## Sqoop:

- Sqoop stands for SQL to Hadoop. It can provision the data from external system on to HDFS and populate tables in Hive and HBase.

## Flume:

- Flume is an important log aggregator (aggregates logs from different machines and places them in HDFS) component in the Hadoop Ecosystem.



# Hadoop Ecosystem Components for Data Processing

## MapReduce:

- It is a programming paradigm that allows distributed and parallel processing of huge datasets. It is based on Google MapReduce.

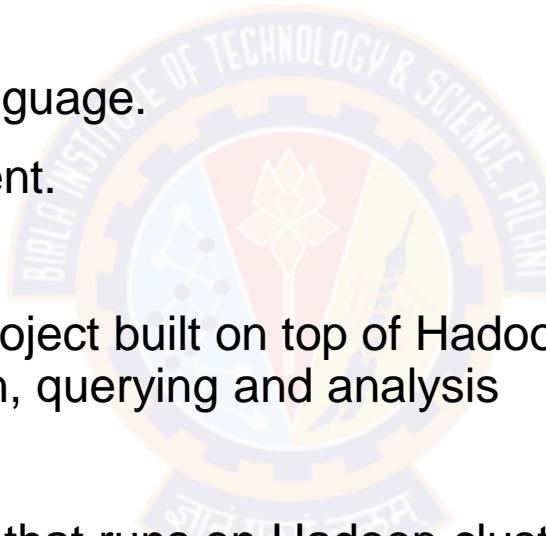
## Spark:

- It is both a programming model as well as a computing model. It is an open source big data processing framework.
- It is written in Scala. It provides in-memory computing for Hadoop.
- Spark can be used with Hadoop coexisting smoothly with MapReduce (sitting on top of Hadoop YARN) or used independently of Hadoop (standalone).

# Hadoop ecosystem components for Data Analysis

## Pig

- It is a high level scripting language used with Hadoop. It serves as an alternative to MapReduce. It has two parts:
- Pig Latin: It is a SQL like scripting language.
- Pig runtime: is the runtime environment.



## Hive:

- Hive is a data warehouse software project built on top of Hadoop. Three main tasks performed by Hive are summarization, querying and analysis

## Impala:

- It is a high performance SQL engine that runs on Hadoop cluster. It is ideal for interactive analysis. It has very low latency measured in milliseconds. It supports a dialect of SQL called Impala SQL.

# RDBMS Vs Hadoop

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs consistent relationship.	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around \$10,000 to \$14,000 per terabytes of storage.	Cost around \$4,000 per terabytes of storage.

# Hadoop – Different modes of operation

- **Standalone Mode**
  - Runs on single system on single JVM (Java Virtual Machine), called Local mode
  - None of the Daemon will run
  - Resource Manager and Node Manager is available for running jobs.
  - Mainly used Hadoop in this Mode for the Purpose of Learning, testing, and debugging.
- **Pseudo-distributed Mode**      (Demo cluster Setup)
  - Uses only single node
  - All the daemons: Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVMs
  - Daemons run on different java processes that is why it is called a Pseudo-distributed.
- **Fully-Distributed Mode**
  - Hadoop runs on clusters of Machine or nodes.
  - Few of the nodes run the Master Daemon's that are Namenode and Resource Manager
  - Rest of the nodes run the Slave Daemon's that are DataNode and Node Manager.
  - Data is distributed across different Data nodes.
  - *Production Mode* of Hadoop cluster providing high availability of data by replication

# Example - Word count

```
public void map(Object key, Text value, Context context  
                ) throws IOException, InterruptedException {  
    StringTokenizer itr = new StringTokenizer(value.toString());  
    while (itr.hasMoreTokens()) {  
        word.set(itr.nextToken());  
        context.write(word, one);  
    }  
}
```



```
public void reduce(Text key, Iterable<IntWritable> values,  
                  Context context  
                ) throws IOException, InterruptedException {  
    int sum = 0;  
    for (IntWritable val : values) {  
        sum += val.get();  
    }  
    result.set(sum);  
    context.write(key, result);  
}
```

input data

hello world bye world

hello hadoop goodbye hadoop

map

<hello, 1>  
<world, 1>  
<bye, 1>  
<world, 1>

map

file outputs of  
map jobs

<hello, 1>  
<hadoop, 1>  
<goodbye, 1>  
<hadoop, 1>

reduce

<bye, 1>  
<goodbye, 1>  
<hello, 2>  
<hadoop, 2>  
<world, 2>

shuffle and  
reduce file  
output

(Sec1 )

Hands on

Hadoop cluster setup in  
Pseudo-distributed Mode

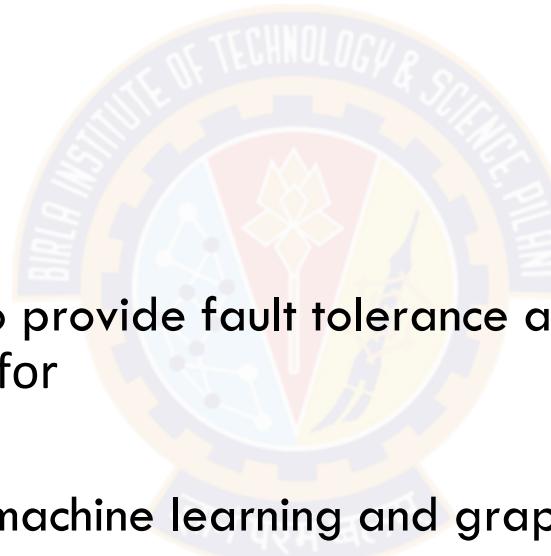




# Introduction to in-memory computing

# Issues with MapReduce on Hadoop

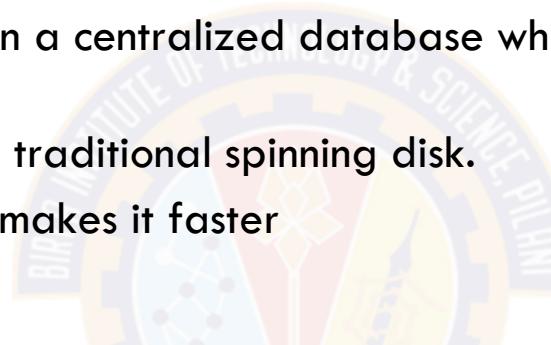
- ✓ Revolutionized big data processing, enabling users to store and process huge amounts of data at very low costs.
- ✓ An ideal platform to implement complex batch applications as diverse as
  - sifting through system logs
  - running ETL
  - computing web indexes
  - recommendation systems etc.
- ✓ Its reliance on persistent storage to provide fault tolerance and its one-pass computation model make MapReduce a poor fit for
  - low-latency applications
  - iterative computations, such as machine learning and graph algorithms
    - There are extensions for iterative MapReduce that we study later



Adapted from : <https://databricks.com/blog/2013/11/21/putting-spark-to-use.html>

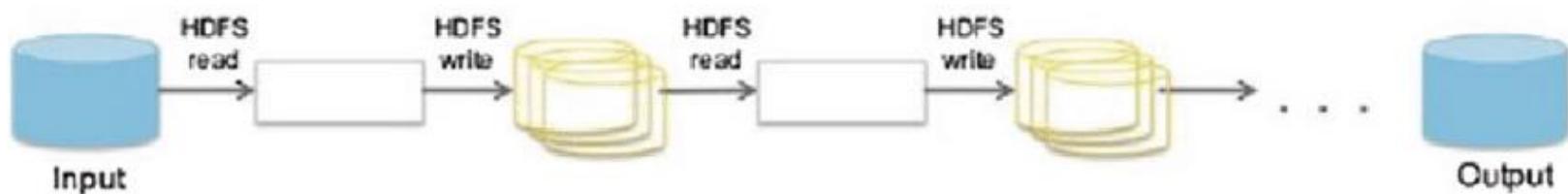
# In-Memory computing

- In-memory computing
  - ✓ means using a type of middleware software that allows one to store data in RAM, across a cluster of computers, and process it in parallel
- For example,
  - ✓ Operational datasets typically stored in a centralized database which you can now store in “connected” RAM across multiple computers.
  - ✓ RAM is roughly 5,000 times faster than traditional spinning disk.
  - ✓ Native support for parallel processing makes it faster

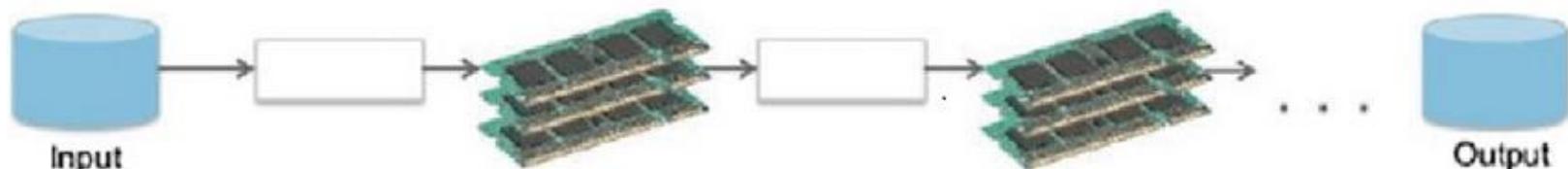


Note:  
Could be batch or streaming

**Hadoop MapReduce: Data Sharing on Disk**



**Spark: Speed up processing by using Memory instead of Disks**



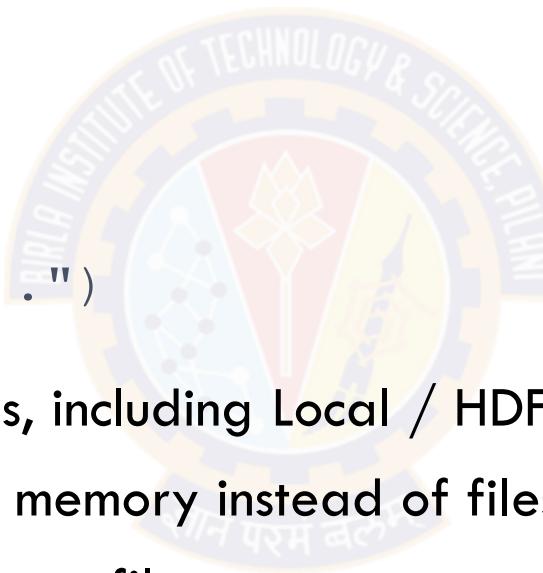
# Fast and easy big data processing with Spark

- At its core, Spark provides a general programming model that enables developers to write application by composing arbitrary operators, such as
  - ✓ mappers
  - ✓ reducers
  - ✓ joins
  - ✓ group-bys
  - ✓ filters
- This composition makes it easy to express a wide array of computations, including iterative machine learning, streaming, complex queries, and batch.
- Spark keeps track of the data that each of the operators produces, and enables applications to reliably store this data in memory using RDDs\*.
  - ✓ This is the key to Spark's performance, as it allows applications to avoid costly disk accesses.

\* RDD: Resilient Distributed Dataset

# Example - Word count

```
Val line = sparkContext.textFile("hdfs://...")  
  
.flatMap(line => line.split(" "))  
  
.map(word => (word, 1))  
  
.reduceByKey(_ + _)  
  
.saveAsTextFile("hdfs://...")
```



convert a file into lines

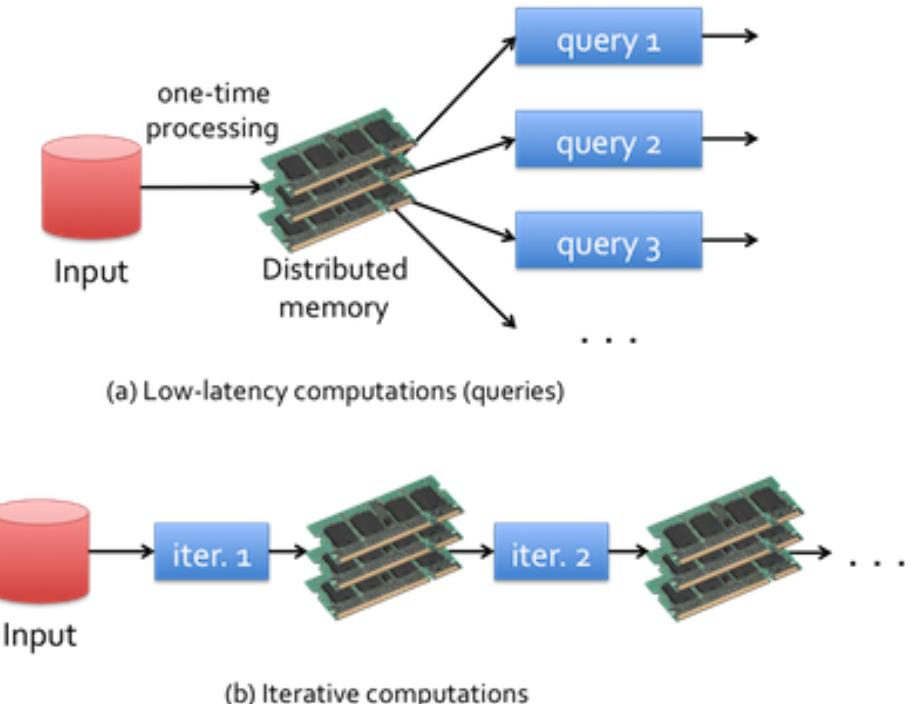
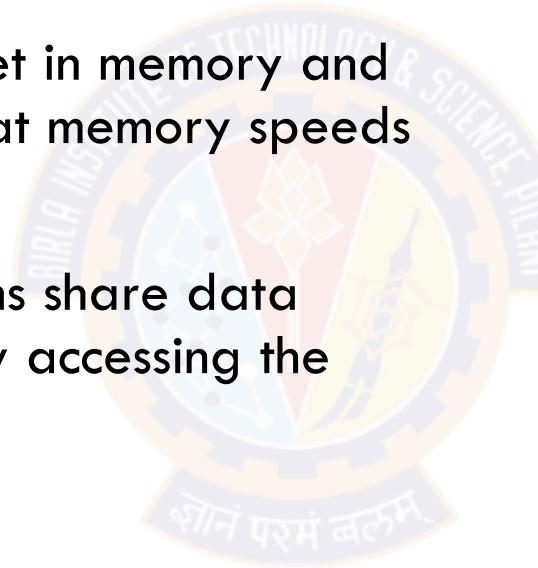
map: transform each word into  $\langle k, v \rangle$  pair

reduce: sum up values for each key

- Can read data from many sources, including Local / HDFS
- Map / reduce output is written to memory instead of files
- Memory content can be written out to files etc.
- A rich set of primitives on top of MapReduce model to make it easier to program

# Ideal Apache Spark applications

- Low-latency computations
  - ✓ by caching the working dataset in memory and then performing computations at memory speeds
- Efficient iterative algorithm
  - ✓ by having subsequent iterations share data through memory, or repeatedly accessing the same dataset



# Hands on with Apache Spark



# Fault tolerance comparison of Hadoop and Spark

- Hadoop and Spark are fault tolerant. They can handle node failures, data corruption, and network issues without affecting the overall execution.
- When any node fails, the workload is redistributed to the remaining active nodes. So, there is no special standby node. (N to N)
- However, Hadoop and Spark have different approaches to fault tolerance
  - Hadoop relies on data replication and checkpointing to ensure fault tolerance, which means that it duplicates the data blocks across multiple nodes and periodically saves the state of the computation to disk.
  - Hadoop provides high reliability and durability, but also consumes more disk space and network bandwidth.
  - Spark relies on data lineage and lazy evaluation to ensure fault tolerance, which means that it tracks the dependencies and transformations of the RDDs and only executes them when needed.
  - Spark provides high performance and flexibility, but also requires more memory and computation power.

# Summary

- Concepts of fault tolerance - availability and reliability
  - Calculating MTTR and availability of systems
  - HA cluster configurations
- Basic concepts and definitions of analytics and Big Data analytics
- Overview of some systems / technologies that support Big Data Analytics
  - Hadoop/MapReduce, Spark/In-memory computing ...
- Hands on with Hadoop Cluster, MapReduce, and Spark



Next Session:  
CAP Theorem and Big Data Lifecycle



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Big Data Systems

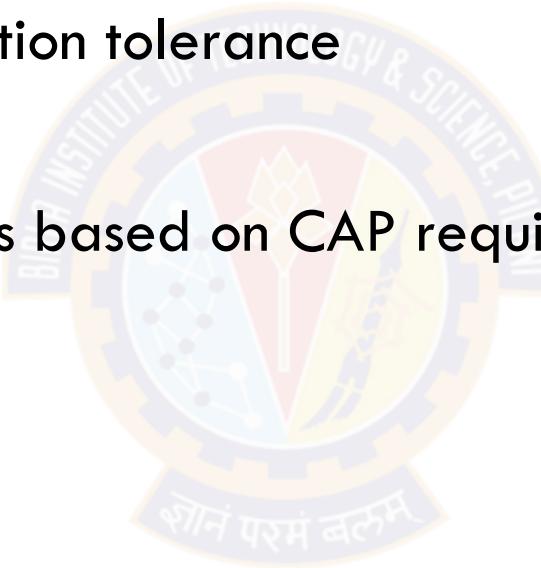
## Session 4 – BDA Lifecycle,CAP Theorem, NoSQL

---

Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

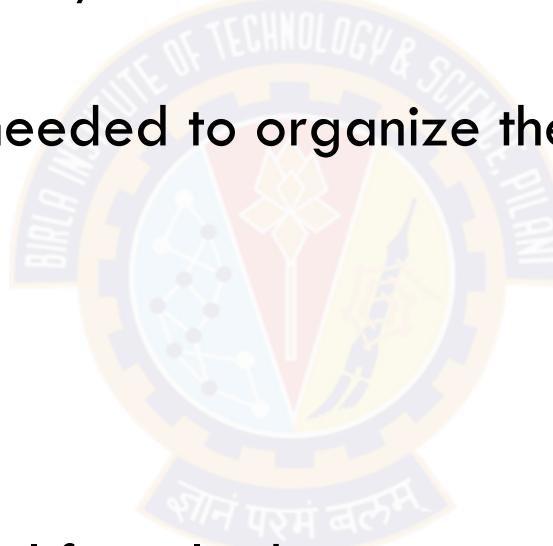
# Topics for today

- **Big Data Analytics lifecycle**
- Consistency, Availability, Partition tolerance
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra



# Big Data Analytics Lifecycle

- Big Data analysis differs from traditional data analysis primarily
  - ✓ due to the volume, velocity and variety characteristics of the data being processed
- A step-by-step methodology is needed to organize the activities / tasks involved with
  - ✓ Acquiring
  - ✓ Processing
  - ✓ Analyzing
  - ✓ Repurposing data
- Explore a specific data analytics lifecycle that organizes and manages the tasks and activities associated with the analysis of Big Data



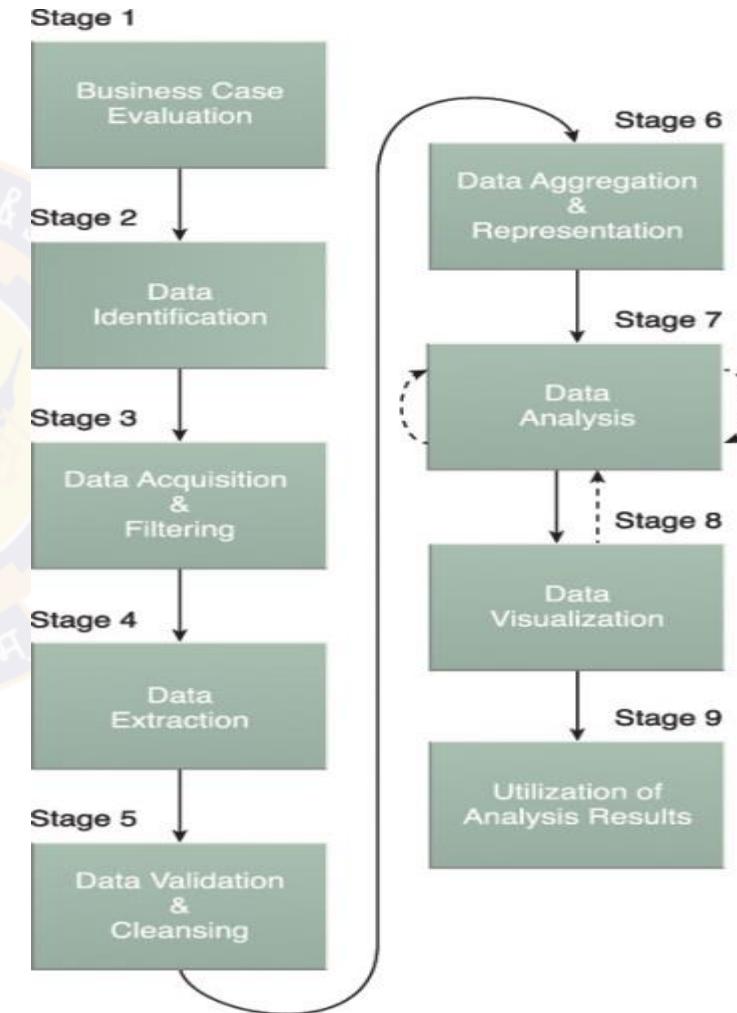
# Example

1. A market research firm creates and runs surveys to understand specific market segments for their client, e.g. consumer electronics - LED TVs
2. These surveys contain questions that have structured : numeric, boolean, categorical, grade as well as unstructured : free form text answers
3. A survey is rolled out to many users with various demographic attributes. The list of survey users could be provided by the client and/or MR firm
4. The results are collected and analyzed for business insights often using multiple tools and analysis techniques
5. The insights are curated and shared to create a presentation for the client of the MR firm
6. The client makes critical business decisions about their product based on the survey results

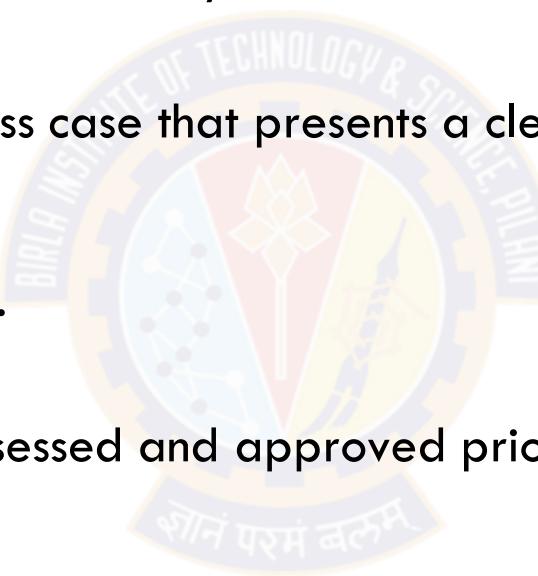
# Big Data Analytics Lifecycle

## Stages

1. Business Case Evaluation
2. Data Identification
3. Data Acquisition & Filtering
4. Data Extraction
5. Data Validation & Cleansing
6. Data Aggregation & Representation
7. Data Analysis
8. Data Visualization
9. Utilization of Analysis Results

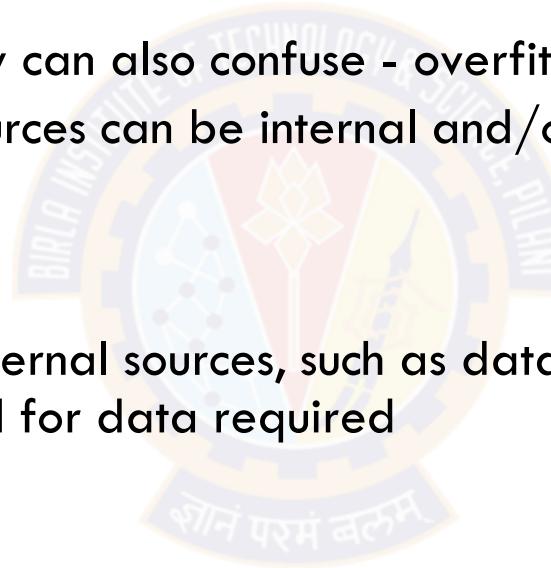


# 1. Business Case Evaluation

- Based on business requirements, determine whether the business problems being addressed is really Big Data problem
    - ✓ A business problem needs to be directly related to one or more of the Big Data characteristics of volume, velocity, or variety.
  - Must begin with a well-defined business case that presents a clear understanding of the
    - ✓ justification
    - ✓ motivation
    - ✓ goals of carrying out the analysis.
  - A business case should be created, assessed and approved prior to proceeding with the actual hands-on analysis tasks.
  - Helps decision-makers to
    - ✓ Understand the business resources that will need to be utilized
    - ✓ Identify which business challenges the analysis will tackle.
    - ✓ Identify KPIs can help determine assessment criteria and guidance for the evaluation of the analytic results
- 
- High volume  
Unstructured data
- Find market fit  
for new product
- What are the business questions ?  
Define thresholds  
on survey stats

## 2. Data Identification

- Main objective is to identify the datasets required for the analysis project and their sources
  - ✓ Wider variety of data sources may increase the probability of finding hidden patterns and correlations.
    - ✓ Caution: Too much data variety can also confuse - overfitting problem.
  - ✓ The required datasets and their sources can be internal and/or external to the enterprise.
- For internal datasets
  - ✓ A list of available datasets from internal sources, such as data marts and operational systems, are typically compiled and verified for data required
- For external datasets
  - ✓ A list of possible third-party data providers, such as data markets and publicly available datasets needs to be compiled
  - ✓ Data may be embedded within blogs or other types of content-based web sites, automated tools needs to be used to extract it

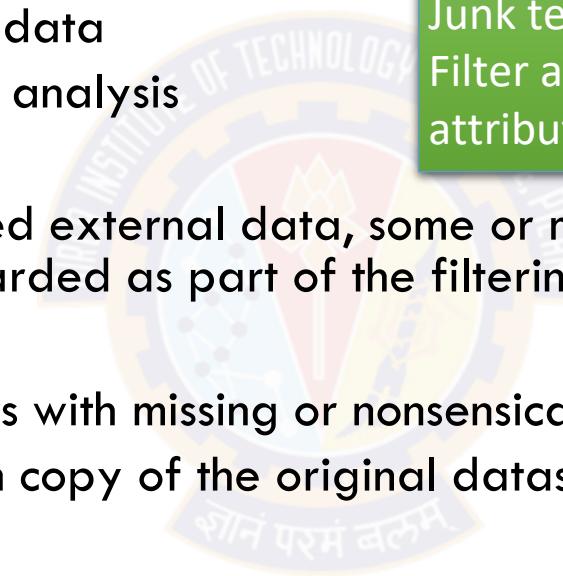


Identify respondents  
Demographics  
What questions to ask  
Do we need other surveys

### 3. Data Acquisition & Filtering

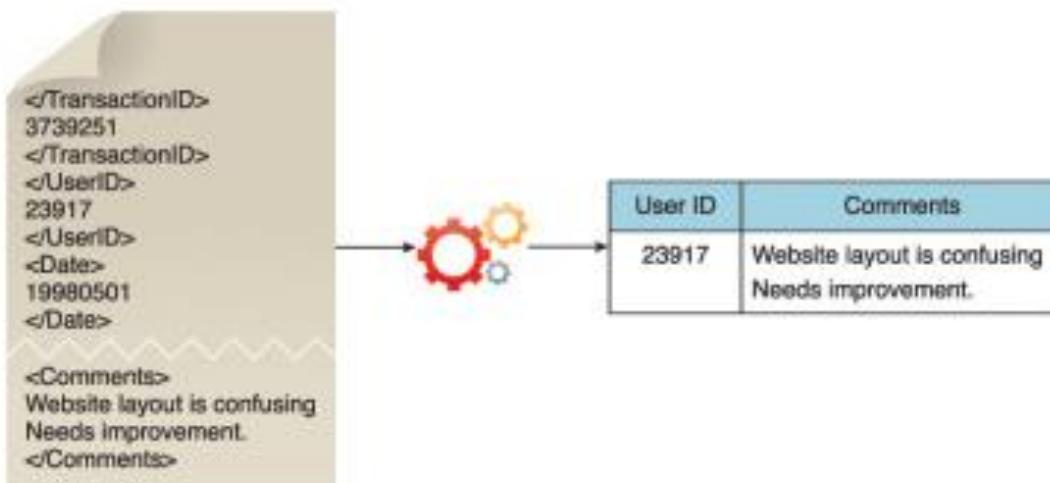
- The data is gathered from all of the data sources that were identified during the last stage
- The acquired data is then looked upon for
  - ✓ filtering / removal of corrupt data
  - ✓ removal of unusable data for analysis
- In many cases involving unstructured external data, some or most of the acquired data may be irrelevant (noise) and can be discarded as part of the filtering process.
- “Corrupt” data can include records with missing or nonsensical values or invalid data types
  - ✓ Advisable to store a verbatim copy of the original dataset before proceeding with the filtering
- Data needs to be persisted once it gets generated or enters the enterprise boundary
  - ✓ For batch analytics, this data is persisted to disk prior to analysis
  - ✓ For real-time analytics, the data is analyzed first and then persisted to disk

Clean bad data, e.g. empty responses  
Junk text inputs  
Filter a subset if we don't need to look at all attributes, all demographics



## 4. Data Extraction

- Dedicated to extracting data and transforming it into a format that the underlying Big Data solution can use for the purpose of the data analysis
- The extent of extraction and transformation required depends on the types of analytics and capabilities of the Big Data solution.



Structure the unstructured responses

Comments and user IDs are extracted from an XML document.

## 5. Data Validation & Cleansing

- Invalid data can skew and falsify analysis results
- Data input into Big Data analyses can be unstructured without any indication of validity
  - ✓ Complexity can further make it difficult to arrive at a set of suitable validation constraints
  - ✓ Dedicated stage is required to establish complex validation rules and removing any known invalid data.
- Big Data solutions often receive redundant data across different datasets.
  - ✓ This can be exploited to explore interconnected datasets in order to
    - assemble validation parameters
    - fill in missing valid data
- For batch analytics, data validation and cleansing can be achieved via an offline ETL operation
- For real-time analytics, a more complex in-memory system is required to validate and cleanse the data as it arrives from the source

Validate survey responses

Contradictory answers

Identify population skews, e.g. responses have inherent gender bias so no point in making a gender based analysis  
Codify certain columns for easier analysis

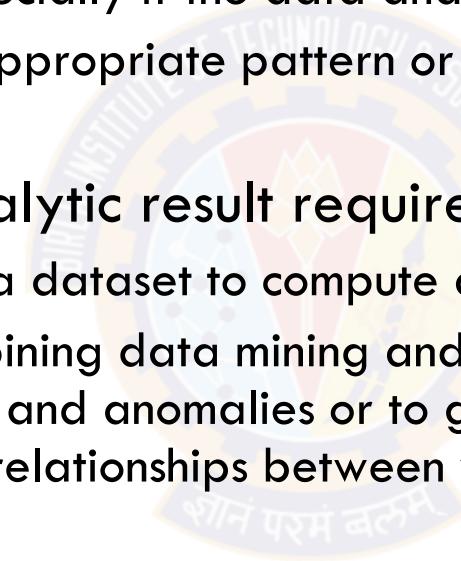
# 6. Data Aggregation & Representation

- Dedicated to integrating multiple datasets together to arrive at a unified view
  - ✓ Needs to merge together the data spread across multiple datasets through a common field
  - ✓ Needs reconciliation of data coming from different sources
  - ✓ Needs to identify the dataset representing the correct value needs to be determined.
- Can be complicated because of :
  - ✓ Data Structure – Although the data format may be the same, the data model may be different
  - ✓ Semantics – A value that is labeled differently in two different datasets may mean the same thing, for example “surname” and “last name.”
- The large volumes makes data aggregation a time and effort-intensive operation
  - ✓ Reconciling these differences can require complex logic that is executed automatically without the need for human intervention
- Future data analysis requirements need to be considered during this stage to help foster data reusability.

Final joined data set (e.g. current with old survey or 3rd party demographics data) with certain aggregations done for downstream analysis

## 7. Data Analysis

- Dedicated to carrying out the actual analysis task, which typically involves one or more types of analytics
  - ✓ Can be iterative in nature, especially if the data analysis is exploratory
  - ✓ Analysis is repeated until the appropriate pattern or correlation is uncovered
- Depending on the type of analytic result required
  - ✓ Can be as simple as querying a dataset to compute an aggregation for comparison
  - ✓ Can be as challenging as combining data mining and complex statistical analysis techniques to discover patterns and anomalies or to generate a statistical or mathematical model to depict relationships between variables.



Various types of descriptive / predictive analysis on survey data to understand market fit for new product. Writing SQL on data and create charts. Build models on the data for hypothesis testing, prediction.

# 7. Confirmatory / Exploratory data analysis

- Confirmatory data analysis
  - ✓ A deductive approach where the cause of the phenomenon being investigated is proposed beforehand - a hypothesis
  - ✓ Data is then analyzed to prove or disprove the hypothesis and provide definitive answers to specific questions
  - ✓ Data sampling techniques are typically used
  - ✓ Unexpected findings or anomalies are usually ignored since a predetermined cause was assumed
- Exploratory data analysis
  - ✓ Inductive approach that is closely associated with data mining
  - ✓ No hypothesis or predetermined assumptions are generated
  - ✓ Data is explored through analysis to develop an understanding of the cause of the phenomenon
  - ✓ May not provide definitive answers
  - ✓ Provides a general direction that can facilitate the discovery of patterns or anomalies

Confirm findings or exception cases in the survey data through adhoc exploration. E.g. in how many cases young males like feature X but don't like feature Y.

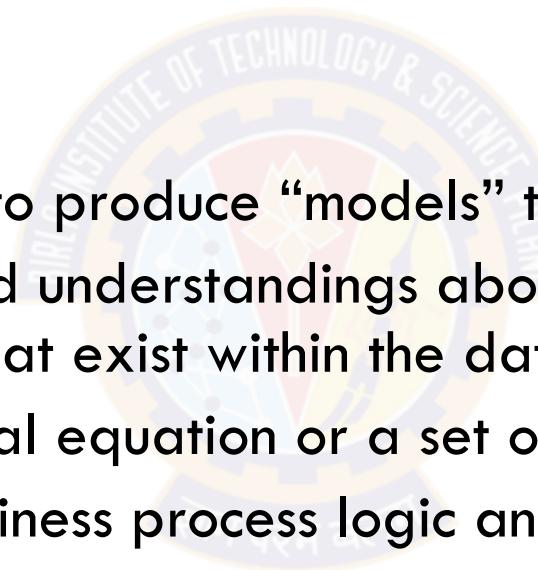
## 8. Data Visualization

Visual results will need to be shared with the stakeholders for the new product launch. e.g. show top features that appeal to each segment of target product user (gender, age group).

- Dedicated to using data visualization techniques and tools to graphically communicate the analysis results for effective interpretation by business users
  - ✓ The ability to analyze massive amounts of data and find useful insights carries little value if the only ones that can interpret the results are the analysts.
  - ✓ Business users need to be able to understand the results in order to obtain value from the analysis and subsequently have the ability to provide feedback
- Provide users with the ability to perform visual analysis, allowing for the discovery of answers to questions that users have not yet even formulated
  - ✓ A method of drilling down to comparatively simple statistics is crucial, in order for users to understand how the rolled up or aggregated results were generated
- Important to use the most suitable visualization technique by keeping the business domain in context
  - ✓ Interpretation of result can vary based on the visualization shown

## 9. Utilization of Analysis Results

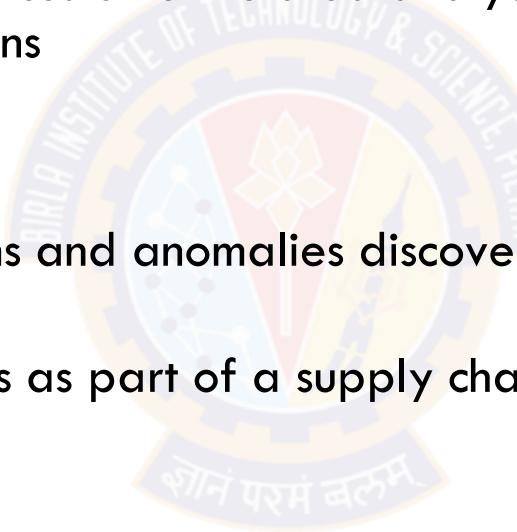
- Dedicated to determining how and where processed analysis data can be further leveraged
  - ✓ Apart from dashboards
- Possible for the analysis results to produce “models” that
  - ✓ encapsulate new insights and understandings about the nature of the patterns and relationships that exist within the data that was analyzed
  - ✓ May look like a mathematical equation or a set of rules
  - ✓ Can be used to improve business process logic and application system logic



Surveys and analysis output, models built are reusable assets  
Reports created to capture the insights.

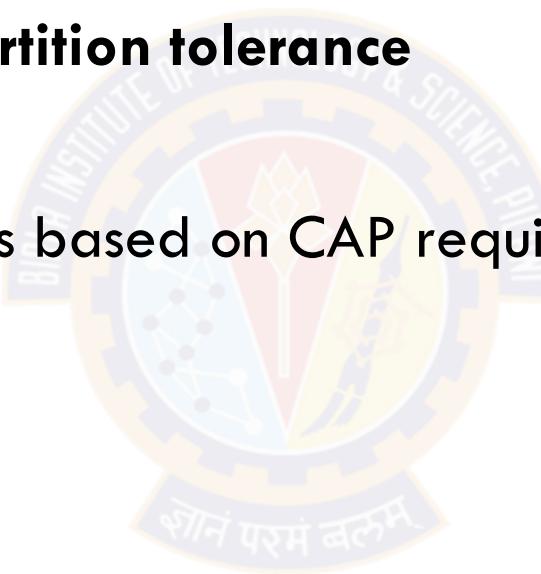
# 9. Utilization of Analysis Results

- Input for Enterprise Systems
  - ✓ Results may be automatically or manually fed directly into enterprise systems to enhance and optimize their behaviors and performance
  - ✓ Online store can be fed processed customer-related analysis results that may impact how it generates product recommendations
- Business Process Optimization
  - ✓ The identified patterns, correlations and anomalies discovered during the data analysis are used to refine business processes
  - ✓ Consolidating transportation routes as part of a supply chain process
- Alerts
  - ✓ Results can be used as input for existing alerts or may form the basis of new alerts
  - ✓ Alerts may be created to inform users via email or SMS text about an event that requires them to take corrective action



# Topics for today

- Big Data Analytics lifecycle
- **Consistency, Availability, Partition tolerance**
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra



# Consistency

- Causes of consistency problem
  - Big Data systems write replicas of a shard / partition
  - Any write needs to be updated on all replicas
  - Any read can happen in between from one or more replicas
- Consistency —
  - Do you allow a read of any replica in any thread to always read the latest value written in any thread ?
    - RDBMS / OLTP systems / Systems of Record
    - ACID (Atomicity, Consistency, Isolation, Durability)
  - Do you allow reads to return any value and eventually show the latest stable value
    - Some BigData systems / Systems of Engagement e.g. social network comments
    - BASE (Basic Availability, Soft state, Eventual consistency)

Ref: **Consistency Models of NoSQL Databases** - <https://www.mdpi.com/1999-5903/11/2/43>

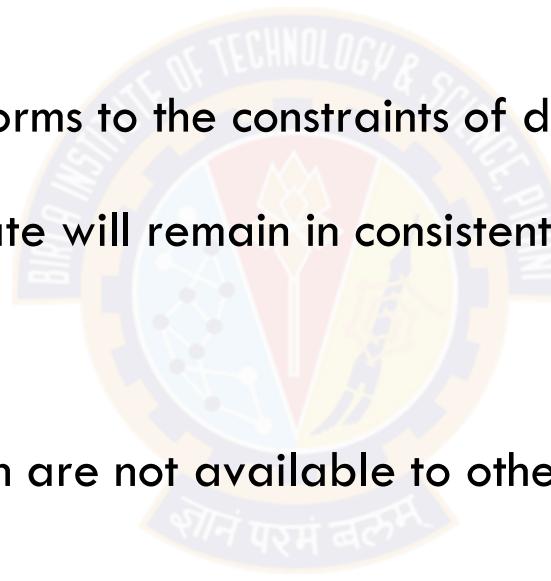
# ACID

- ACID is a database design principle related to transaction management
  - ✓ Atomicity
  - ✓ Consistency
  - ✓ Isolation
  - ✓ Durability
- ACID is the traditional approach to database transaction management as it is leveraged by relational database management systems



# ACID (2)

- Atomicity
  - ✓ Ensures that all operations will always succeed or fail completely
  - ✓ No partial transactions
- Consistency
  - ✓ Ensures that only data that conforms to the constraints of database schema can be written to the database
  - ✓ Database that is in inconsistent state will remain in consistent stage following a successful transaction.
- Isolation
  - ✓ Ensures that results of transaction are not available to other operations until it is complete.
  - ✓ Critical for concurrency control.
- Durability
  - ✓ Ensures that results of transaction are permanent
  - ✓ Once transaction is committed , it can not be rolled back.



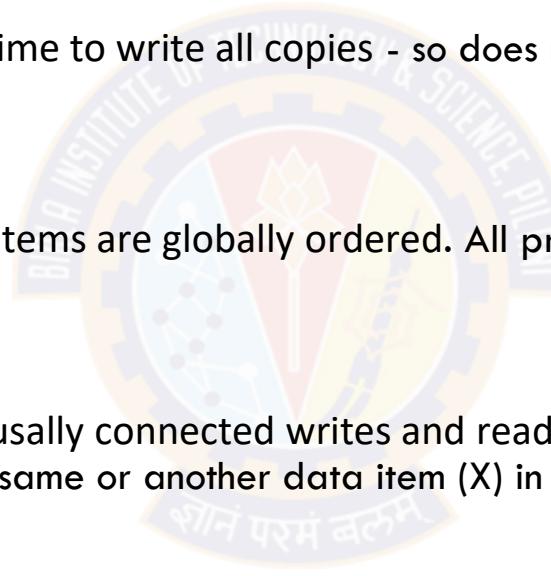
# Consistency clarification

- C in ACID is different from C as in CAP Theorem (discussed next) \*
- C in ACID of RDBMS is based OLTP systems
  - A broader concept at a data base level for a transaction involving multiple data items
  - Harder to achieve
- C in this context and in CAP Theorem for most NoSQL systems
  - Applies to ordering of operations for a single data item AND not a transaction involving multiple data items
  - So, it is a strict subset of C in ACID
  - Typically support of full ACID semantics for NoSQL systems defeats the purpose as it involves having a single transaction manager that becomes a scale bottleneck for large number of partitions and replicas

\* <https://hackingdistributed.com/2013/03/23/consistency-alphabet-soup/#:~:text=%22C%20as%20in%20ACID%22%20is,arbitrarily%20large%20groups%20of%20objects>

# Levels of Consistency

- Strict
  - Requires real time line ordering of all writes - assumes actual write time can be known. So “reads” read the latest data in real time across processors.
- Linearisable
  - Acknowledges that write requests take time to write all copies - so does not impose ordering within overlapping time periods of read / write
- Sequential
  - All writes across processors for all data items are globally ordered. All processors must see the same order. But does not need real-time ordering.
- Causal
  - Popular and useful model where only causally connected writes and reads need to be ordered. So if a write of a data item (Y) happened after a read of same or another data item (X) in a processor then all processors must observe write X before write to Y.
- Eventual (most relaxed as in BASE)
  - If there are no writes for “some time” then all Processors will eventually agree on a latest value of the data item



# Example: Strictly consistent

- Requires real time line ordering of all writes - assumes actual write time can be known.  
So “reads” read the latest data in real time across threads.

(Initial value of X and Y are 0)

P1:      W: x=5

---

P2:      W: y=10

---

P3:      R: x=5      R: y=10

---

P4:      R: y=10      R: x=5

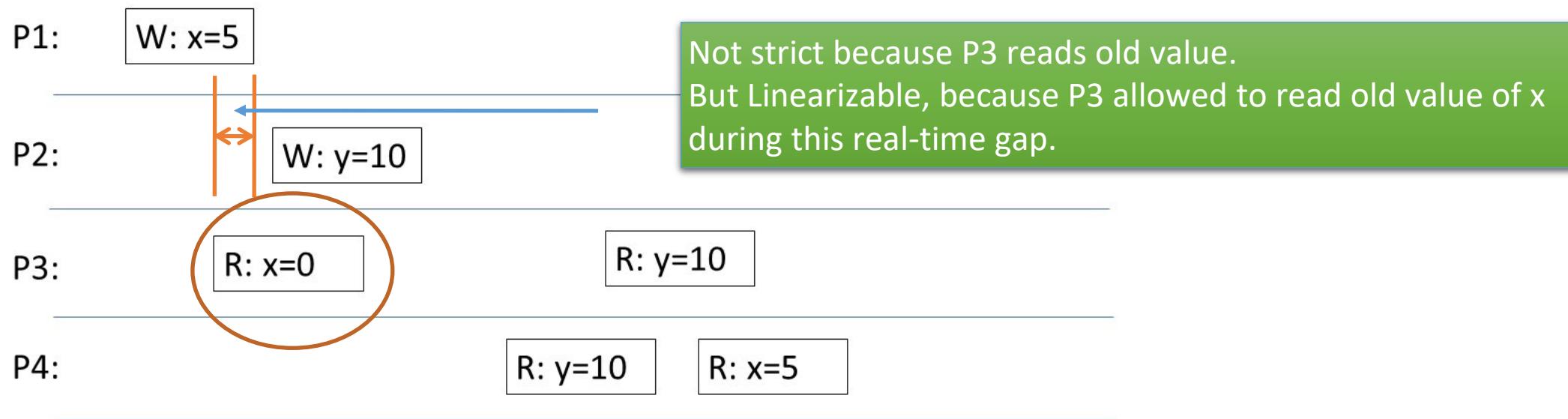
---

This schedule is sequentially consistent, causally consistent, linearizable and strictly consistent

# Example: Linearizable

- Acknowledges that write requests take time to write all copies - so does not impose ordering within overlapping time periods of read / write

(Initial value of X and Y are 0)

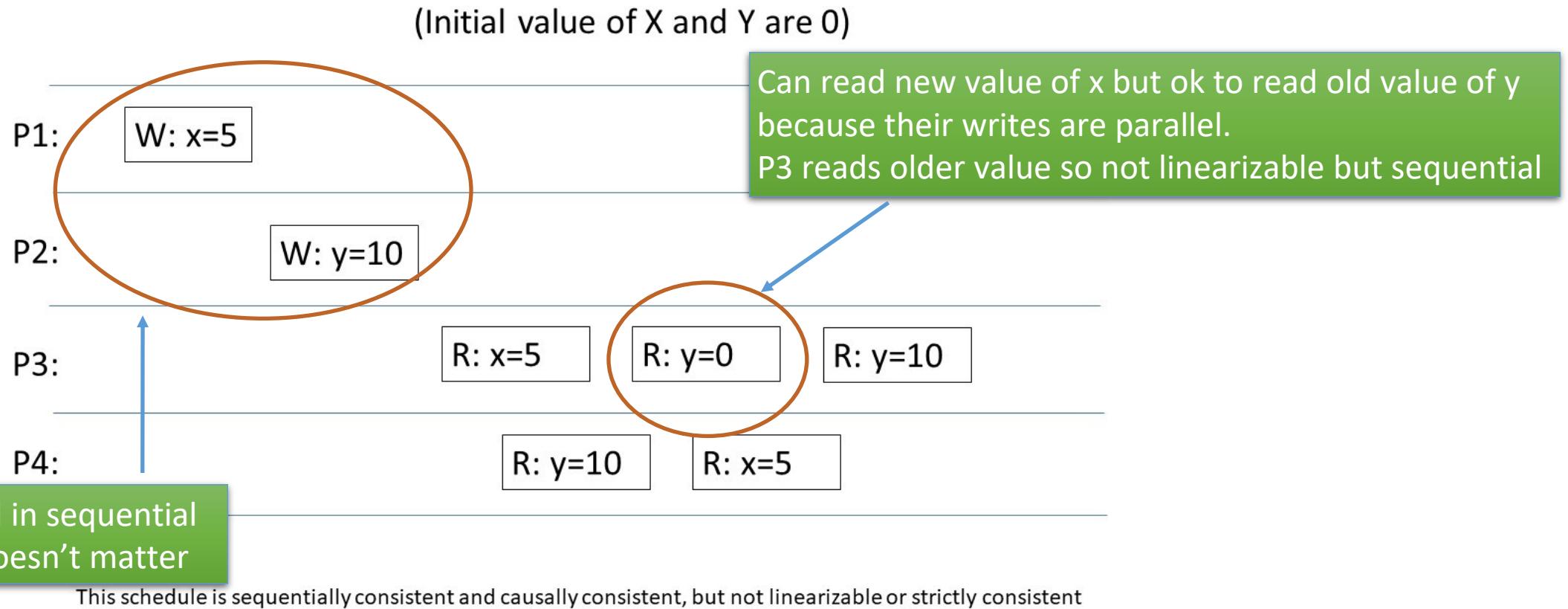


This schedule is sequentially consistent, causally consistent, and linearizable, but **not** strictly consistent

Linearizability takes into account the overlapping time when P3 could not read the latest write of x.

# Example: Sequentially consistent

- All writes across threads for all data items are globally ordered. All threads must see the same order. But does not need real-time ordering.

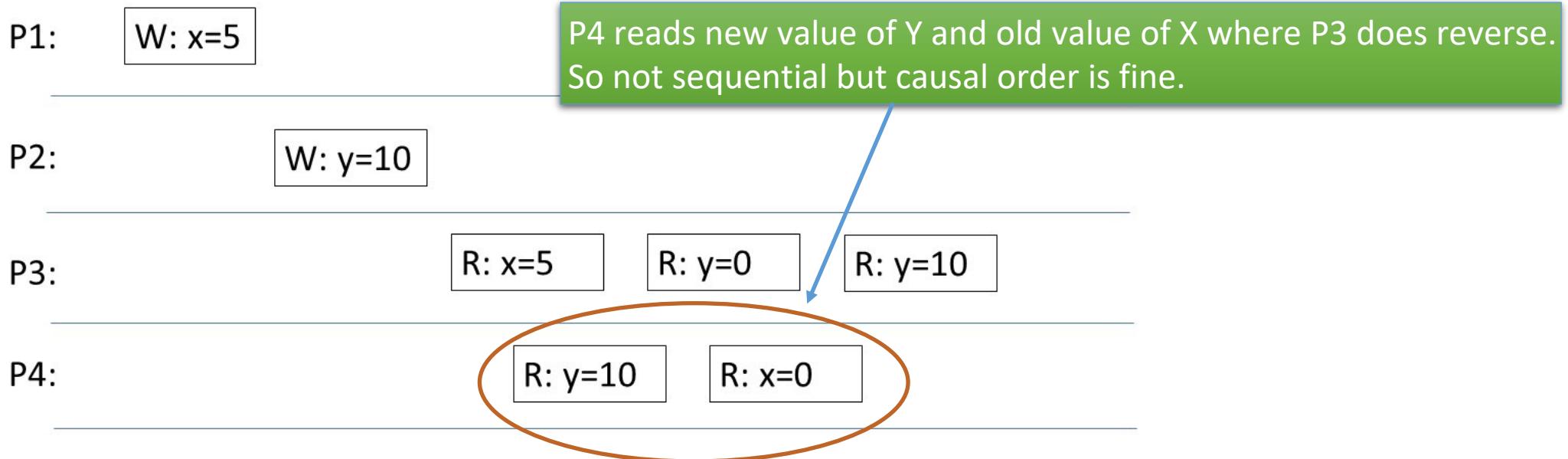


All writes across threads are globally ordered but not in real-time. Here P3 reads older value of Y in real-time but P1 and P2 writes are deemed parallel in sequential order.

# Example: Causally consistent

- Popular and useful model where only causally connected writes and reads need to be ordered. So if a write of a data item (Y) happened after a read of same or another data item (X) in a thread then all threads must observe write X before write to Y.

(Initial value of X and Y are 0)

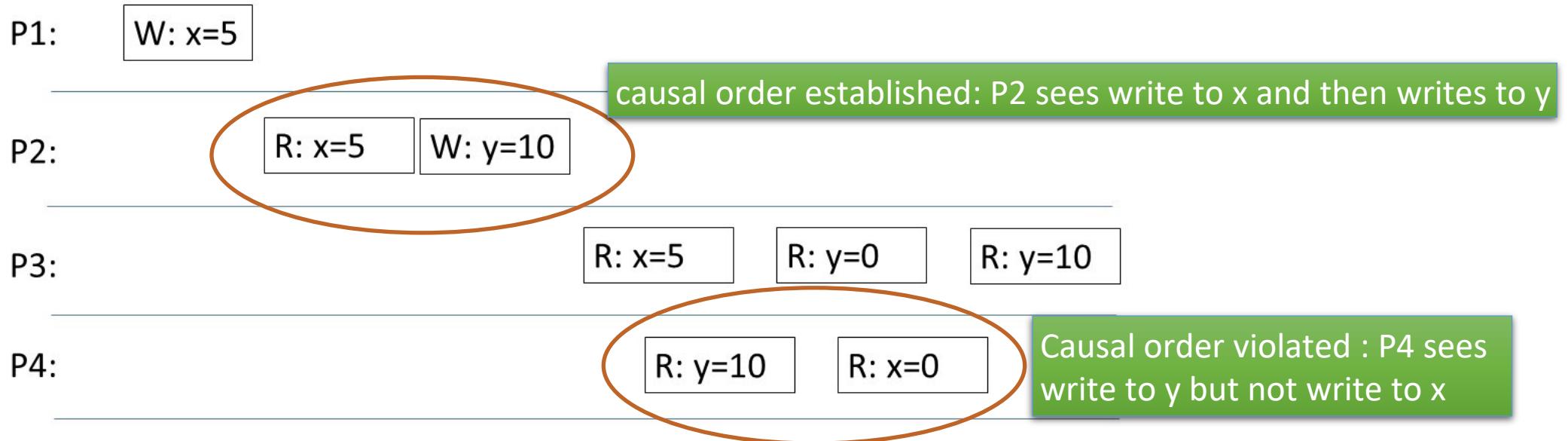


This schedule is causally consistent, but not linearizable or strictly consistent or even sequentially consistent

Cannot be sequentially consistent because P3 and P4 read X and Y values in different order. But no causal order violated because x and y are not causally related.

# Example: Eventually consistent

(Initial value of X and Y are 0)



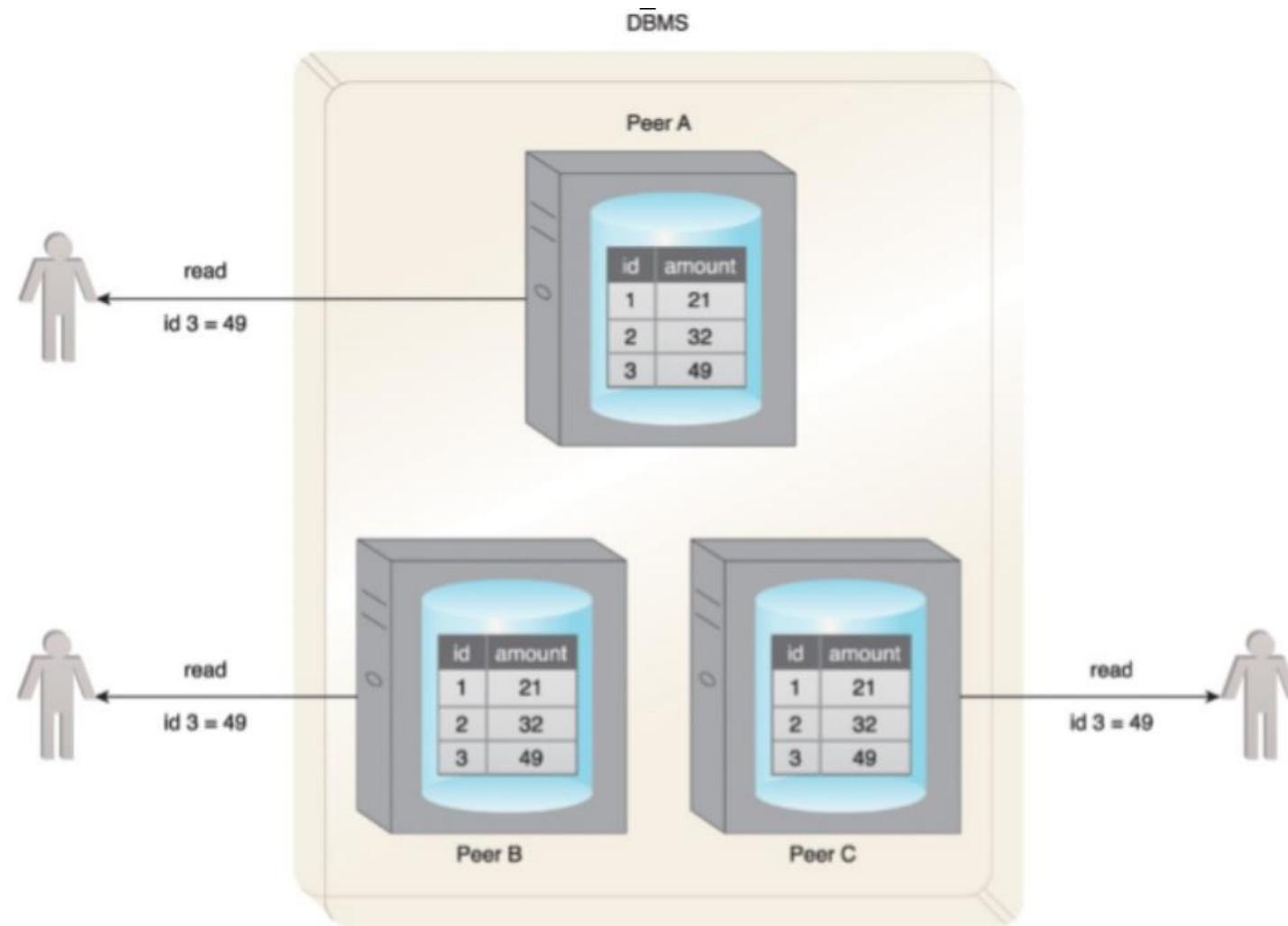
This schedule is **not** causally consistent, nor linearizable or strictly consistent or sequentially consistent

# CAP

- Stands for Consistency (C) , Availability (A) and Partition Tolerance (P)
- Triple constraint related to the distributed database systems
- Consistency
  - ✓ A read of a data item from any node results in same data across multiple nodes
- Availability
  - ✓ A read/write request will always be acknowledged in form of success or failure in reasonable time
- Partition tolerance
  - ✓ System can continue to function when communication outages split the cluster into multiple silos and can still service read/write requests

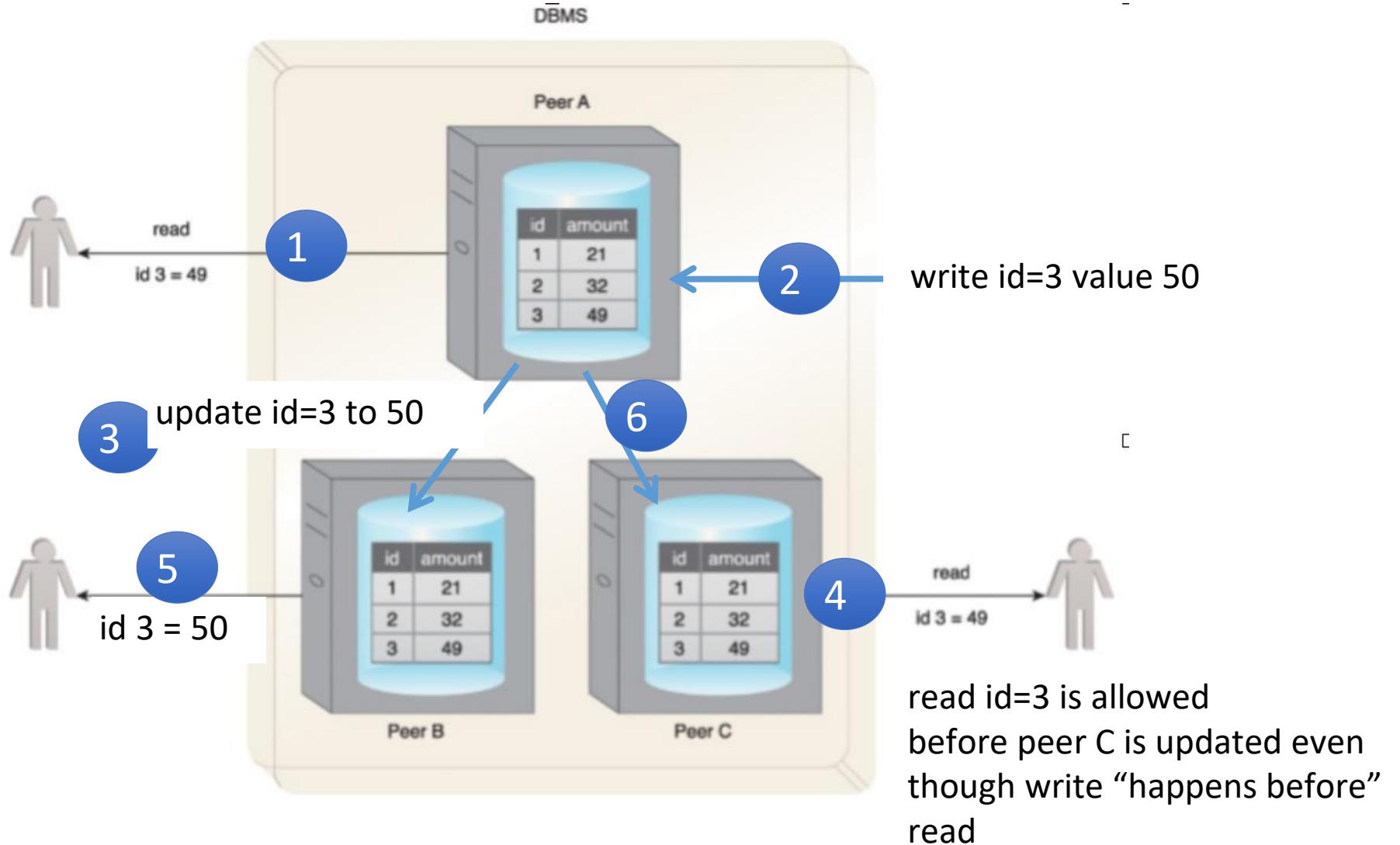
**split brain problem**

# C: Consistency

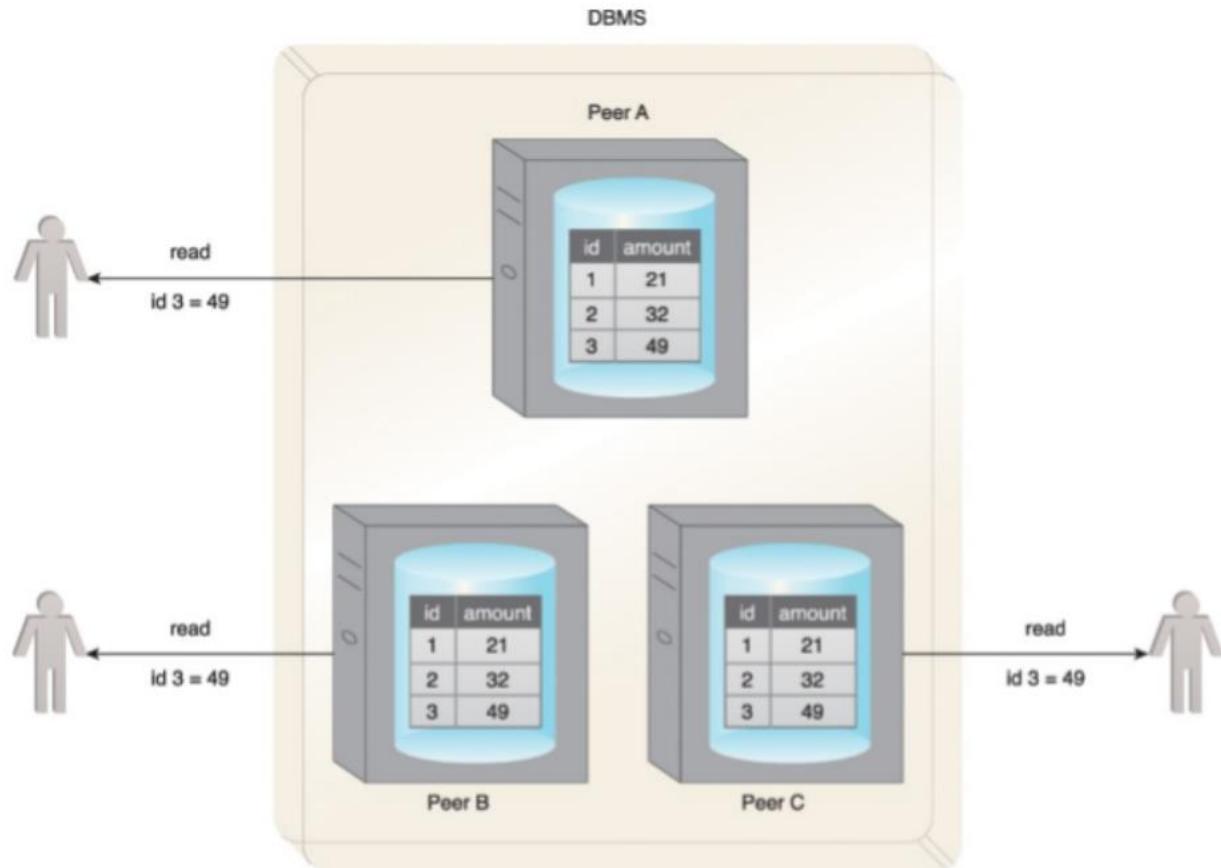


All users get the same value for the amount column even though different replicas are serving the record

# When can it be in-consistent

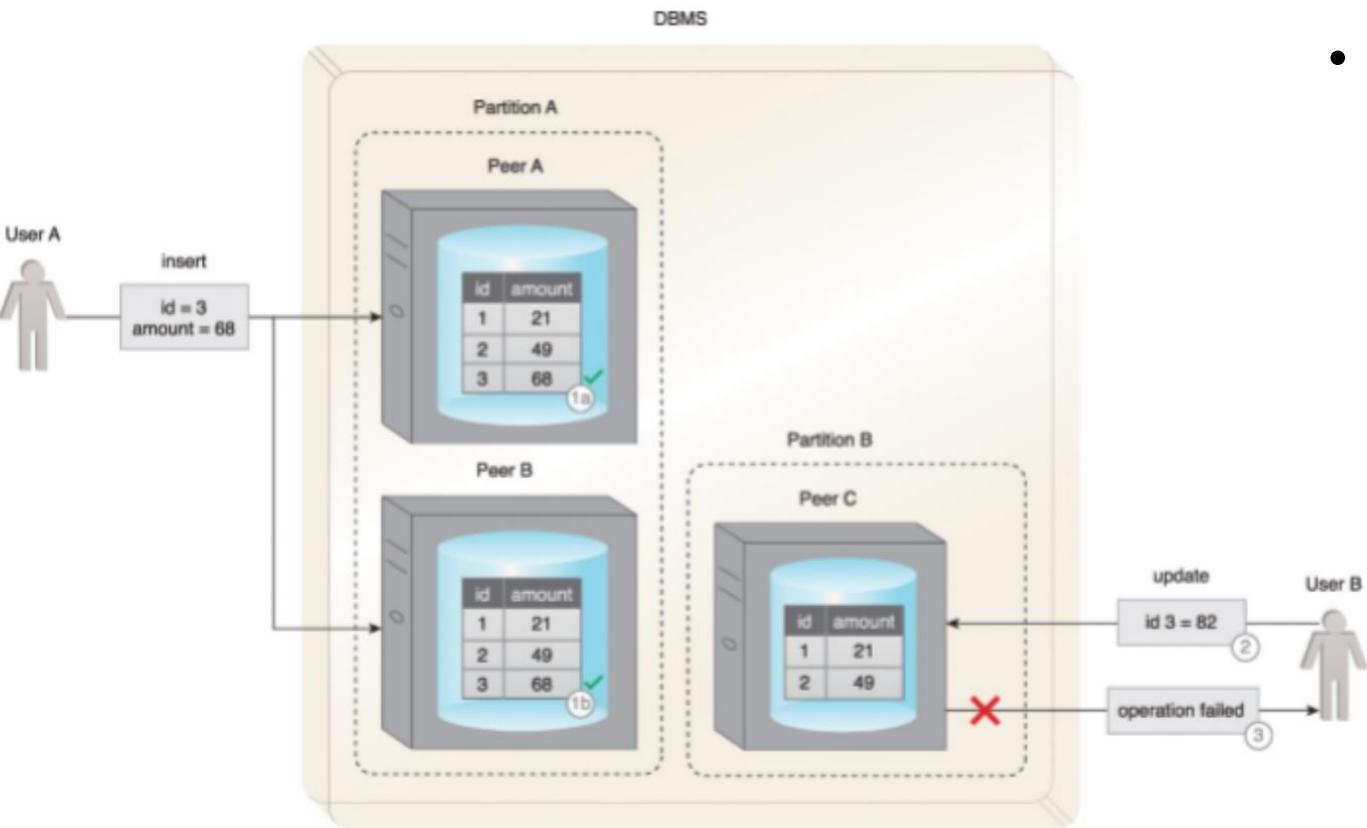


# A: Availability



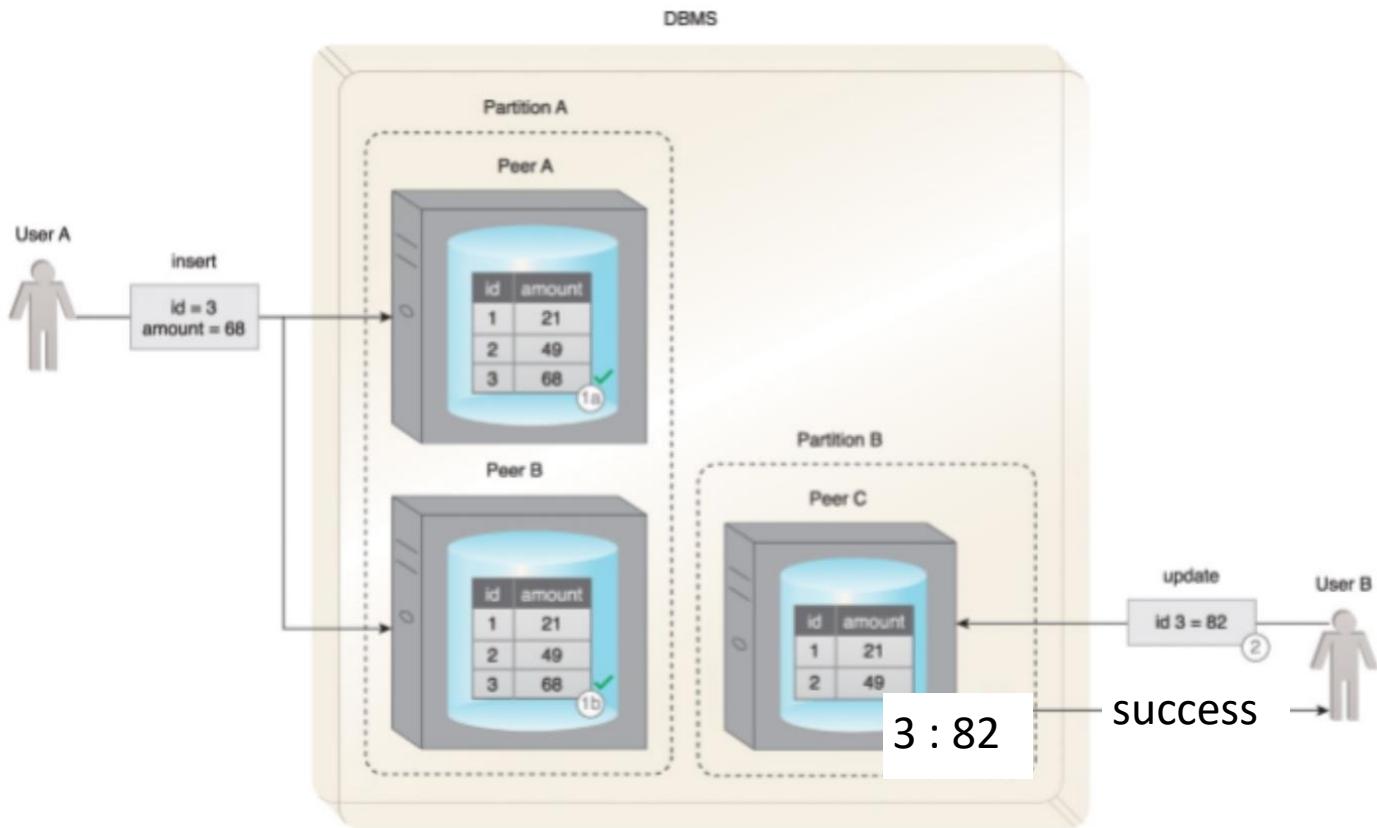
- A request from a user to a peer always responds with a success or failure within a reasonable time.
- Say Peer C is disconnected from the network, then it has 2 options
  - Available: Respond with a failure when any request is made, or
  - Unavailable: Wait for the problem to be fixed before responding, which could be unreasonably long and user request may time out

# P: Partition tolerance (1)



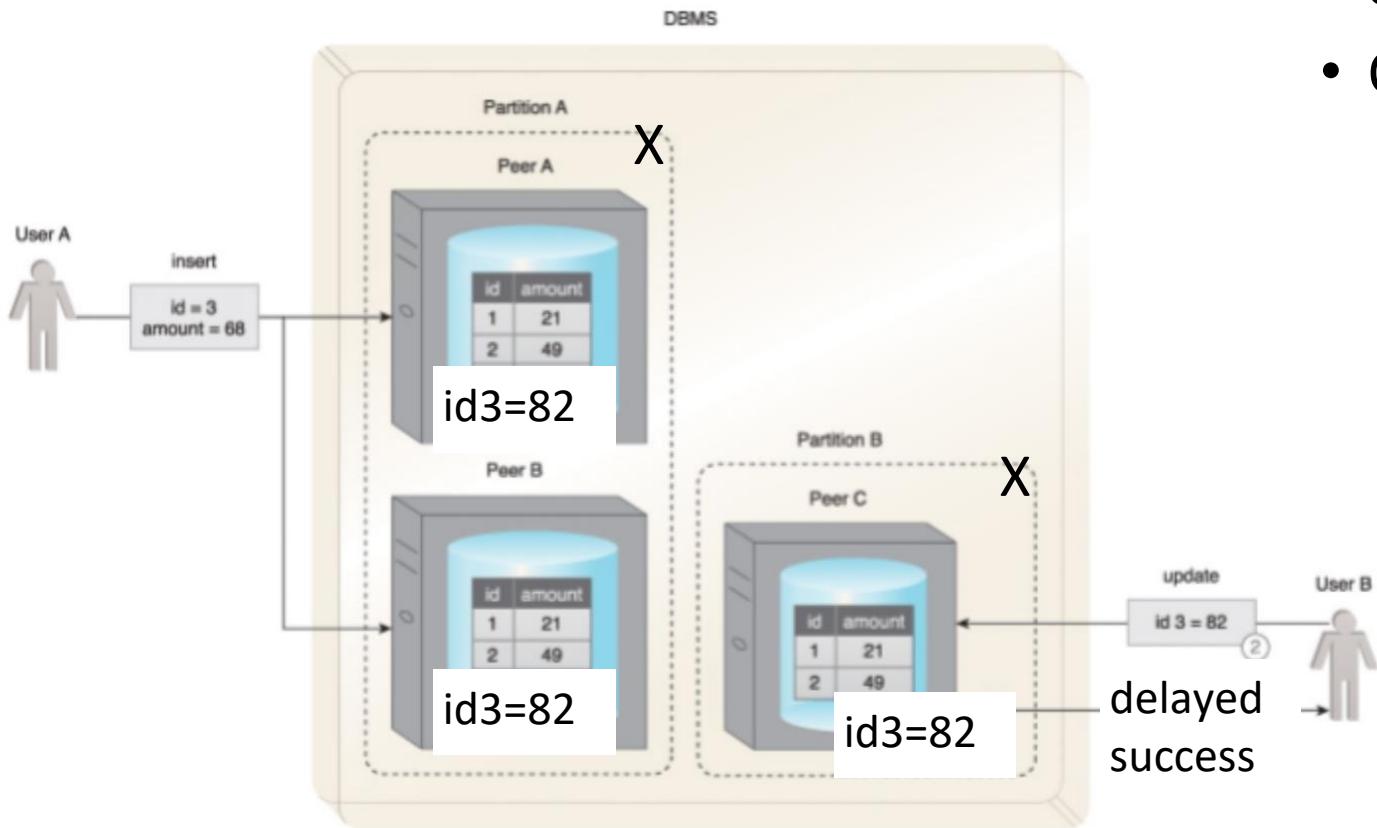
- A network partition happens and user wants to update peer C
- Option 1:
  - Any access by user on peer C leads to failure message response
  - System is available because it comes back with a response
  - There is consistency because user's update is not processed
  - But there is no partition tolerance
  - C and A no P

# P: Partition tolerance (2)



- A network partition happens and user wants to update peer C
- Option 2:
  - Peer C records the update by user with success
  - System is still available and now partition tolerant.
  - But system becomes inconsistent
  - P and A but no C

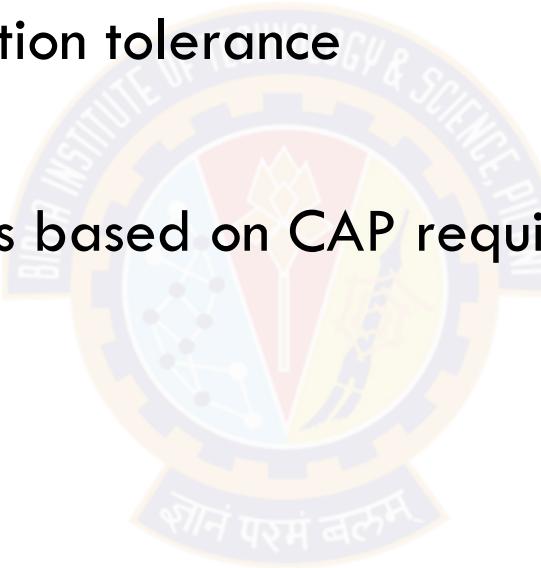
# P: Partition tolerance (3)



- A network partition happens and user wants to update peer C
- Option 3:
  - User is made to wait till partition is fixed (could be quite long) and data replicated on all peers before a success message is sent
  - System appears unavailable to user though it is partition tolerant and consistent
  - P and C but no A

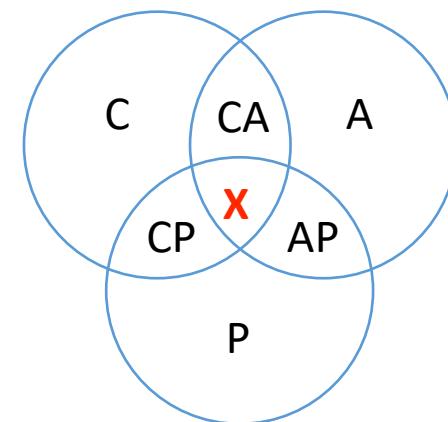
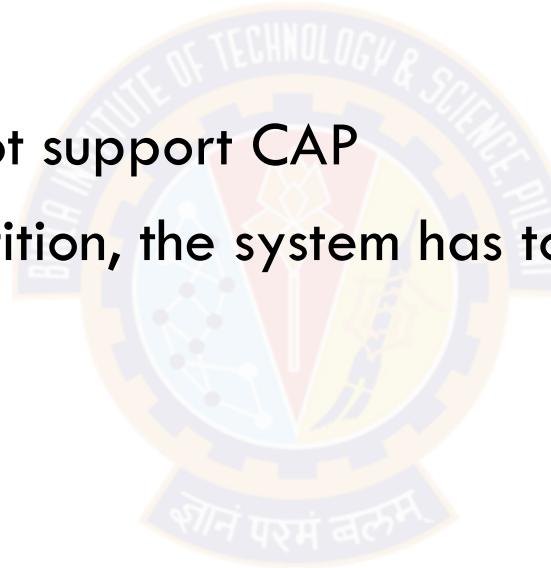
# Topics for today

- Big Data Analytics lifecycle
- Consistency, Availability, Partition tolerance
- **CAP theorem**
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra



# CAP Theorem (Brewer's Theorem)

- A distributed data system, running over a cluster, can only provide two of the three properties C, A, P but not all.
- So a system can be
  - CA or AP or CP but cannot support CAP
- In effect, when there is a partition, the system has to decide whether to pick consistency or availability.



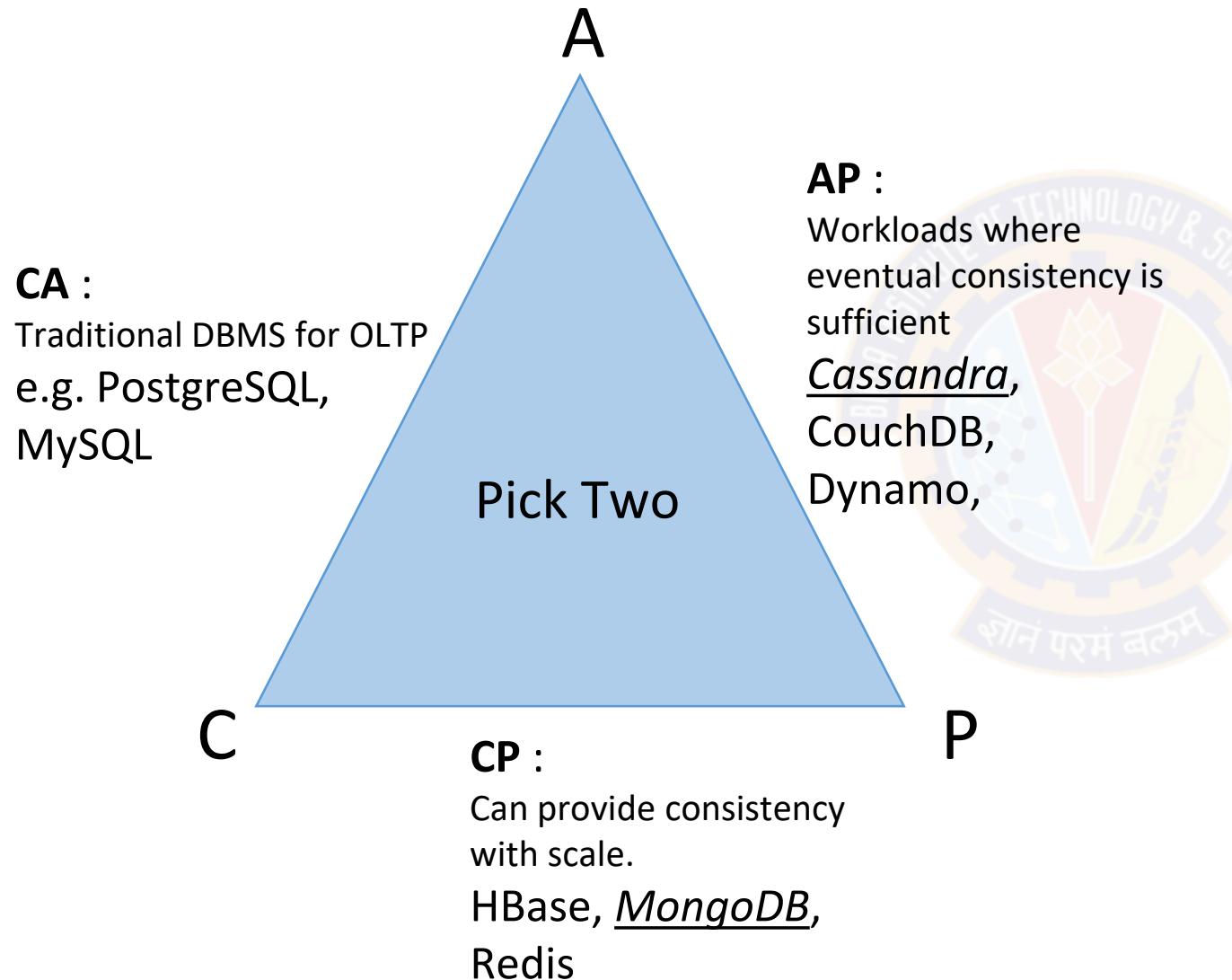
# CAP Theorem – Historical note

- . Prof. Eric Brewer (UC Berkeley) presented it as the CAP principle in a 1999 article
- Then as an informal conjecture in his keynote at the PODC 2000 conference
- . In 2002 a formal proof was given by Gilbert and Lynch, making CAP a theorem
- [Seth Gilbert, Nancy A. Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2): 51-59 (2002)]
  - [https://mwhittaker.github.io/blog/an\\_illustrated\\_proof\\_of\\_the\\_cap\\_theorem/](https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/)
- It is mainly about making the statement formal; the proof is straightforward

# Consistency or Availability choices

- In a distributed database,
  - Scalability and fault tolerance can be improved through additional nodes, although this puts challenges on maintaining consistency (C).
  - The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
    - May have to update all replicas before sending success to client . so longer takes time and system may not be available during this period to service reads on same data item.
- Large scale distributed systems cannot be 100% partition tolerant (P).
  - Although communication outages are rare and temporary, partition tolerance (P) must always be supported by distributed database
- Therefore, CAP is generally a choice between choosing either CP or AP
- Traditional RDBMS systems mainly provide CA for single data items and then on top of that provide ACID for transactions that touch multiple data items.

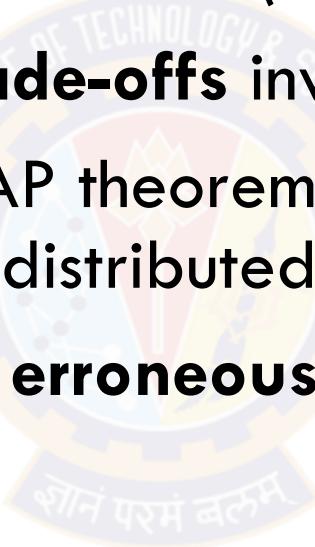
# Database options



- Different design choices are made by Big Data DBs
- Faults are likely to happen in large scale systems
- Provides flexibility depending on use case to choose C, A, P behavior mainly around consistency semantics when there are faults

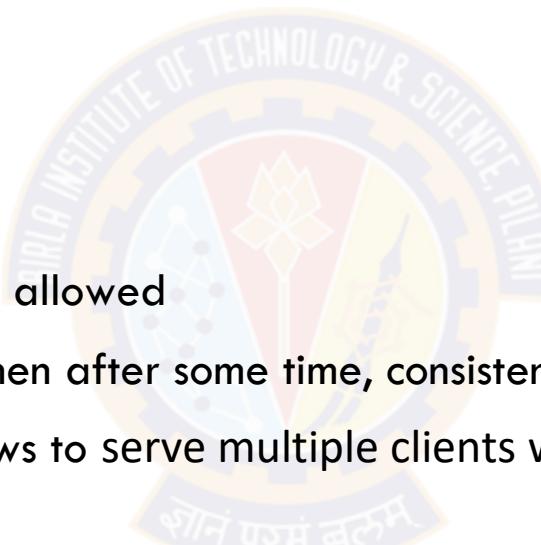
# Importance of CAP Theorem

- The future of databases is **distributed** (Big Data Trend, etc.)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **making decisions** about the future of distributed database **design**
- Misunderstanding can lead to **erroneous or inappropriate design choices**

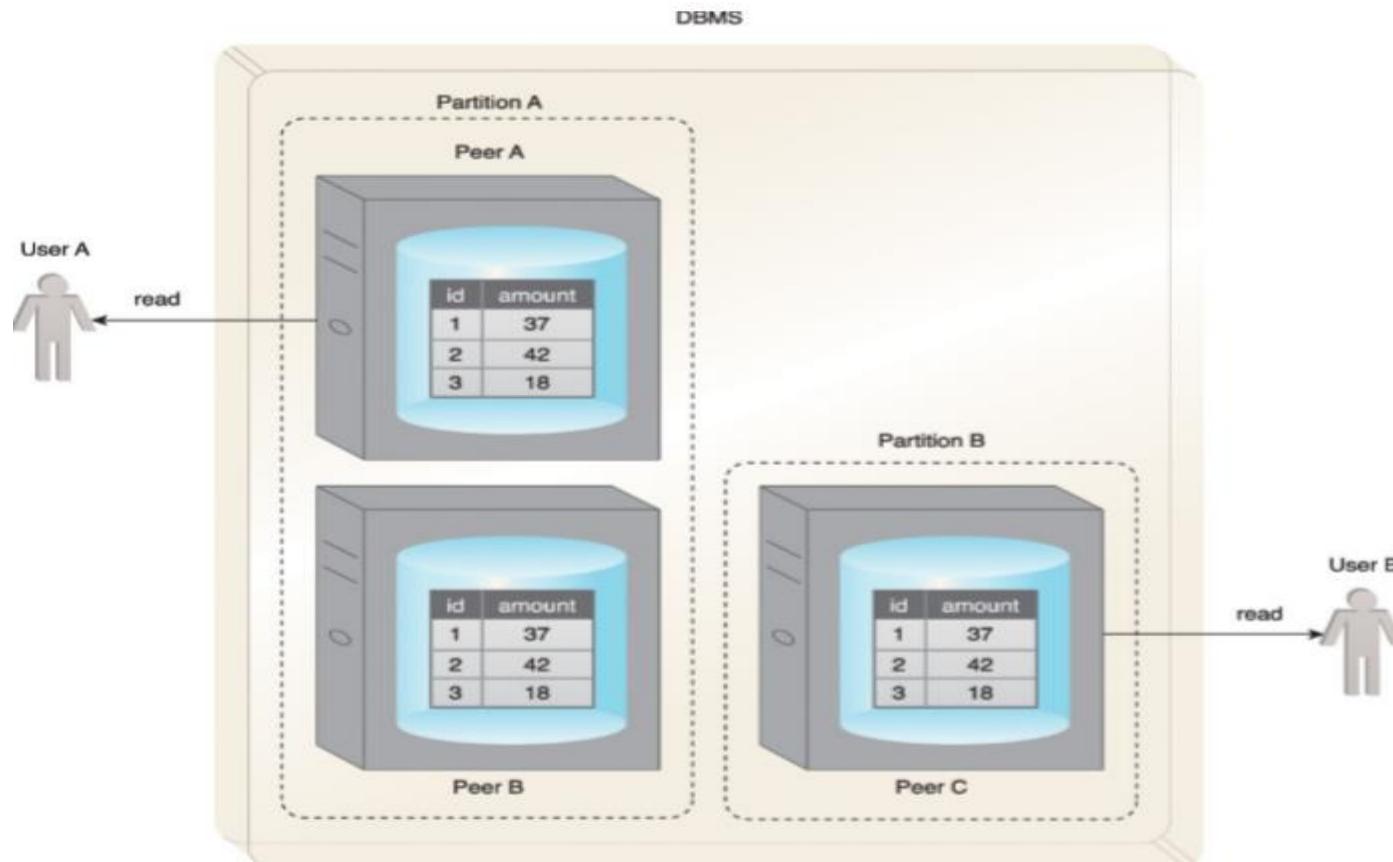


# BASE

- BASE is a database design principle based on CAP theorem
- Leveraged by AP database systems that use distributed technology
- Stands for
  - ✓ Basically Available (BA)
  - ✓ Soft state (S)
  - ✓ Eventual consistency (E)
- It favours availability over consistency
- Soft state - Inconsistency (stale answers) allowed
- Eventual consistency - If updates stop, then after some time, consistency will be achieved
- Soft approach towards consistency allows to serve multiple clients without any latency albeit serving inconsistent results
- Not useful for transactional systems where lack of consistency is concern
- Useful for write-heavy workloads where reads need not be consistent in real-time, e.g. social media applications, monitoring data for non-real-time analysis etc.
- BASE Philosophy: best effort, optimistic, staleness and approximation allowed



# BASE – Basically Available

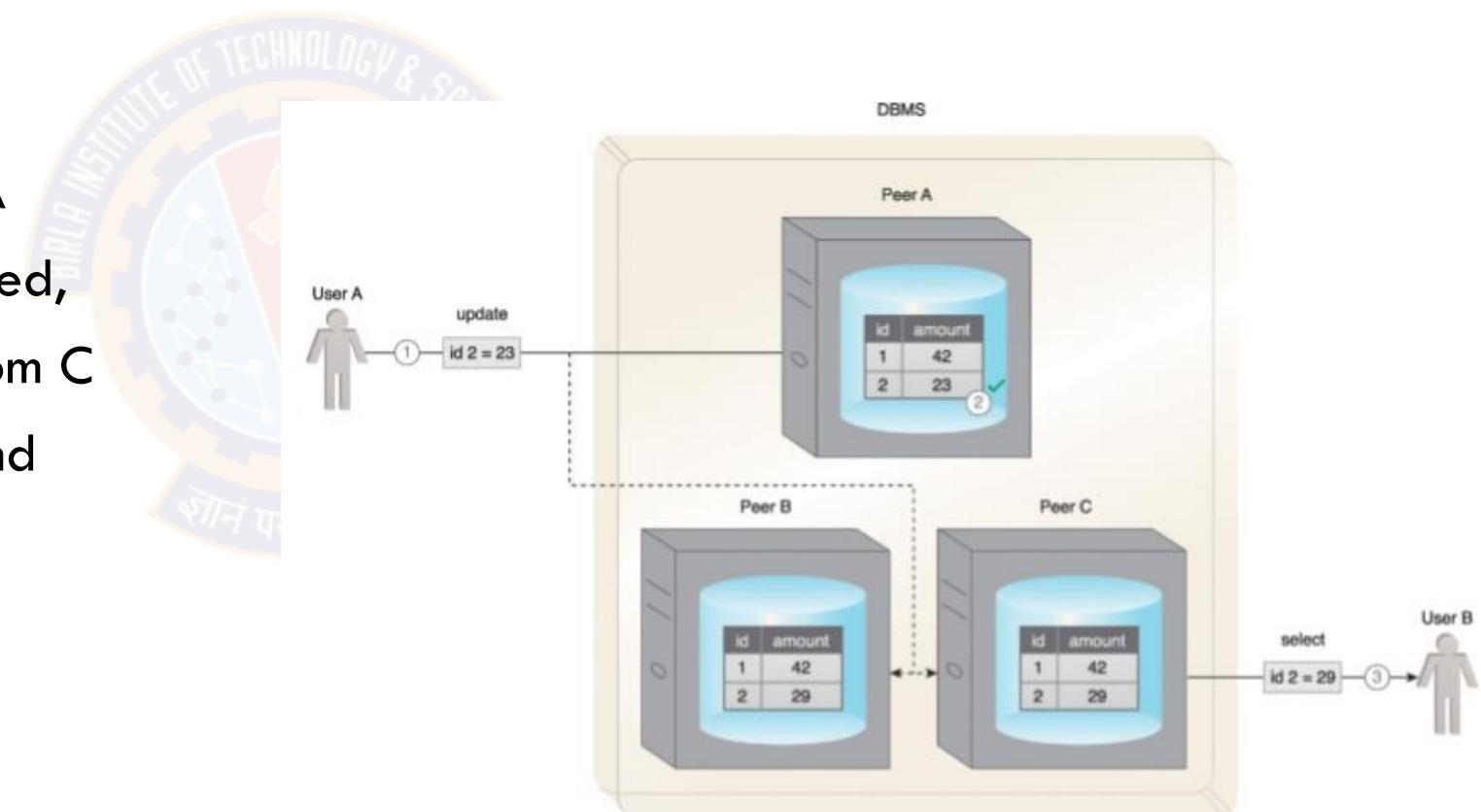


Database will always acknowledge a client's request, either in form of requested data or a failure notification

- User A and User B receive data despite the database being partitioned by a network failure.

# BASE – Soft State

- Database may be in inconsistent state when data is read, thus results may change if the same data is requested again
  - ✓ Because data could be uploaded for consistency, even though no user has written to the database between two reads



An example of the soft state property of BASE is shown here.

# BASE – Eventual Consistency

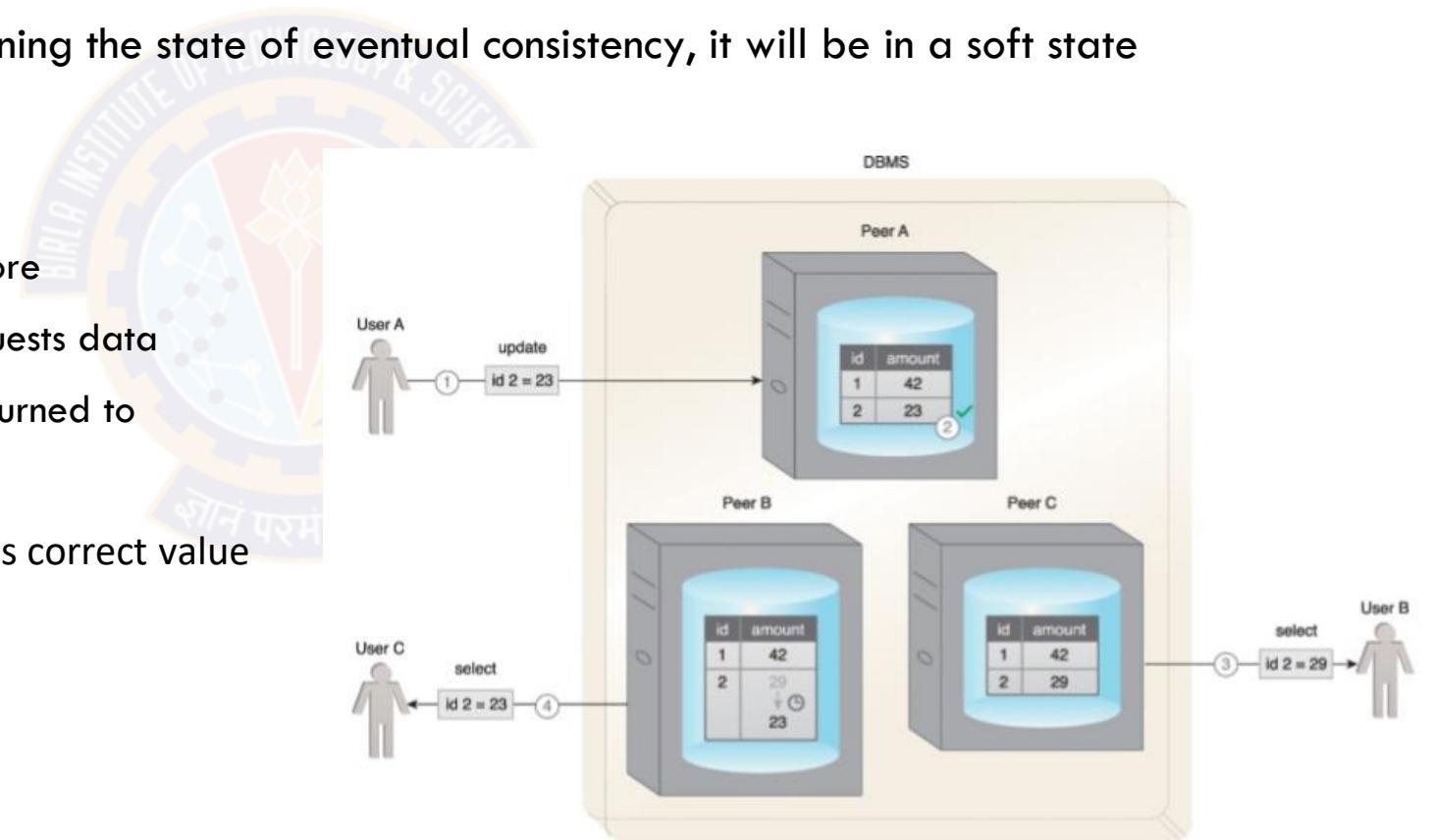
- State in which reads by different clients, immediately following a write to database, may not return consistent results
- Database only attains consistency once the changes have been propagated to all nodes
- While database is in the process of attaining the state of eventual consistency, it will be in a soft state

✓ User A updates a record

✓ Record only gets updated at node A, but before  
other peers can be updated, User B requests data

✓ Database is now in soft state, stale data is returned to  
User B from peer C

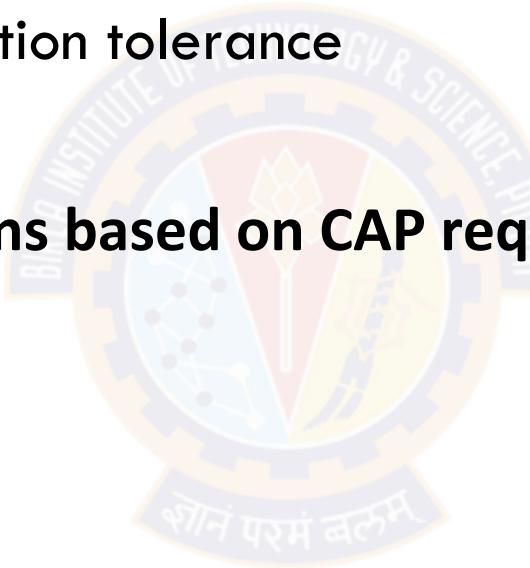
✓ Consistency is eventually attained, User C gets correct value



An example of the eventual consistency property of BASE.

# Topics for today

- Big Data Analytics lifecycle
- Consistency, availability, partition tolerance
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - Cassandra



# MongoDB

- Open source
- 1<sup>st</sup> release 2009
- Document store
  - An extended format called BSON (binary JSON)
- Supports replication (master/slave), sharding
  - Developer provides the “shard key” – collection is partitioned by ranges of values of this key
- Consistency guarantees, CP of CAP
- Used by Adobe (experience tracking), Craigslist, eBay, FIFA (video game), LinkedIn, McAfee
- Provides connector to Hadoop
  - Cloudera provides the MongoDB connector in distributions

# cloud.mongodb.com

## Clusters

Find a cluster... 

**SANDBOX**

● Cluster0  
Version 4.4.6

**CONNECT** **METRICS** **COLLECTIONS** **...**

**CLUSTER TIER**  
M0 Sandbox (General)

**REGION**  
AWS / Mumbai (ap-south-1)

**TYPE** Replica Set - 3 nodes

**LINKED REALM APP**  
None Linked

Find Indexes Schema Anti-Patterns 0 Aggreg

FILTER {"filter": "example"}

QUERY RESULTS 1-20 OF MANY

**sample\_airbnb**

- listingsAndReviews
- sample\_analytics
- sample\_geospatial
- sample\_mflix
- sample\_restaurants
- sample\_supplies
- sample\_training
- sample\_weatherdata

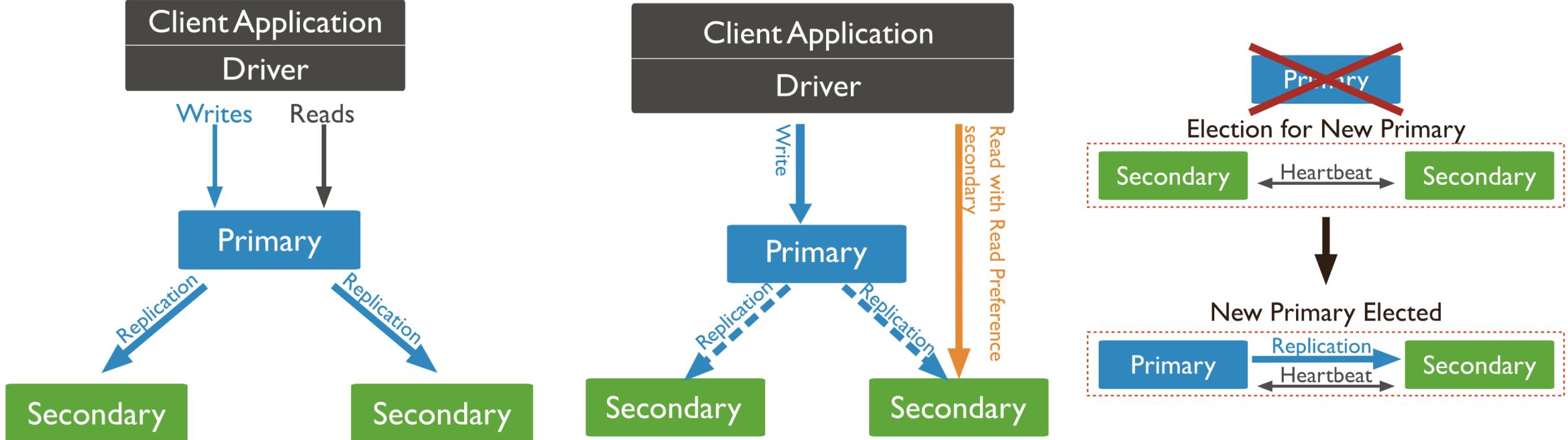
**\_id: "10006546"**  
**listing\_url: "https://www.airbnb.com/rooms/10006546"**  
**name: "Ribeira Charming Duplex"**  
**summary: "Fantastic duplex apartment with three bedrooms, lc**  
**space: "Privileged views of the Douro River and Ribeira squa**  
**description: "Fantastic duplex apartment with three bedrooms**  
**neighborhood\_overview: "In the neighborhood of the river, yc**  
**notes: "Lose yourself in the narrow streets and staircases z**  
**transit: "Transport: • Metro station and S. Bento railway 5m**  
**access: "We are always available to help guests. The house i**  
**interaction: "Cot - 10 € / night Dog - € 7,5 / night"**  
**house\_rules: "Make the house your home..."**  
**property\_type: "House"**  
**room\_type: "Entire home/apt"**  
**bed\_type: "Real Bed"**  
**minimum\_nights: "2"**  
**maximum\_nights: "30"**  
**cancellation\_policy: "moderate"**  
**last\_scraped: 2019-02-16T05:00:00.000+00:00**  
**calendar\_last\_scraped: 2019-02-16T05:00:00.000+00:00**  
**first\_review: 2016-01-03T05:00:00.000+00:00**  
**last\_review: 2019-01-20T05:00:00.000+00:00**  
**accommodates: 8**  
**bedrooms: 3**  
**beds: 5**

Get me top 10 beach front homes

```
# sort search results by score
s = collection.aggregate([
    { "$match": { "text": { "$search": "beach front" } } },
    { "$project": { "name": 1, "_id": 0, "score": { "$meta": "textScore" } } },
    { "$match": { "score": { "$gt": 1.0 } } },
    { "$sort": { "score": -1}},
    { "$limit": 10}
])
```

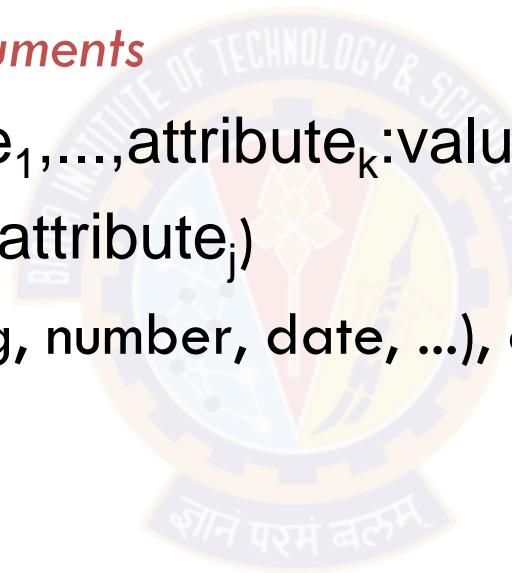
# MongoDB

- Document oriented DB
- Various read and write choices for flexible consistency tradeoff with scale / performance and durability
- Automatic primary re-election on primary failure and/or network partition



# MongoDB Data Model

- JavaScript Object Notation (JSON) model
- *Database* = set of named ***collections***
- *Collection* = sequence of ***documents***
- *Document* = {attribute<sub>1</sub>:value<sub>1</sub>,...,attribute<sub>k</sub>:value<sub>k</sub>}
- *Attribute* = string (attribute<sub>i</sub>≠attribute<sub>j</sub>)
- *Value* = **primitive** value (string, number, date, ...), or a **document**, or an **array**
- *Array* = [value<sub>1</sub>,...,value<sub>n</sub>]
- Key properties: **hierarchical** (like XML), **no schema**
  - Collection docs may have different attributes



# MongoDB Data Example

Collection inventory

```
{  
    item: "ABC2",  
    details: { model: "14Q3", manufacturer: "M1 Corporation" },  
    stock: [ { size: "M", qty: 50 } ],  
    category: "clothing"  
}  
  
{  
    item: "MNO2",  
    details: { model: "14Q3", manufacturer: "ABC Company" },  
    stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L", qty: 1 } ],  
    category: "clothing"  
}
```

```
db.inventory.insert(  
{  
    item: "ABC1",  
    details: {model: "14Q3",manufacturer: "XYZ Compa  
stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],  
category: "clothing"  
}  
)
```

Document insertion

# Example of Simple Query

Collection orders

```
{  
  _id: "a",  
  cust_id: "abc123",  
  status: "A",  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 3 },  
           { sku: "nnn", qty: 5, price: 2 } ]  
}  
  
{  
  _id: "b",  
  cust_id: "abc124",  
  status: "B",  
  price: 12,  
  items: [ { sku: "nnn", qty: 2, price: 2 },  
           { sku: "ppp", qty: 2, price: 4 } ]  
}
```

db.users.find(

  { status: "A" },

  { cust\_id: 1, price: 1, \_id: 0 }

)

*selection*

*projection*

In SQL it would look like this:

```
SELECT cust_id, price  
FROM orders  
WHERE status="A"
```

Results

{

  cust\_id: "abc123",  
  price: 25

}

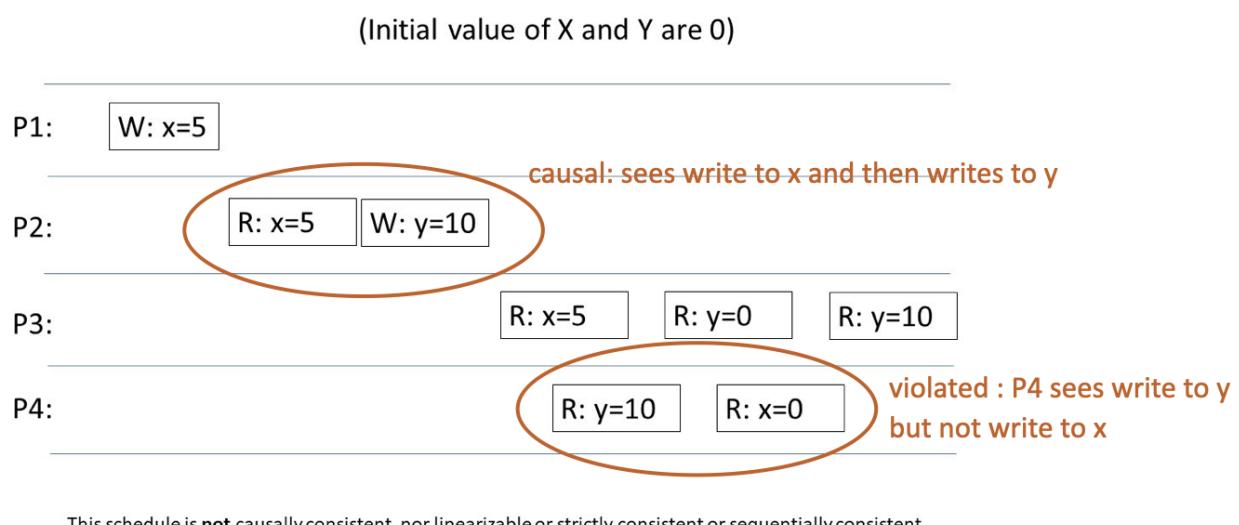
# What is Causal Consistency (recap)

**Read your writes** Read operations reflect the results of write operations that precede them.

**Monotonic reads** Read operations do not return results that correspond to an earlier state of the data than a preceding read operation.

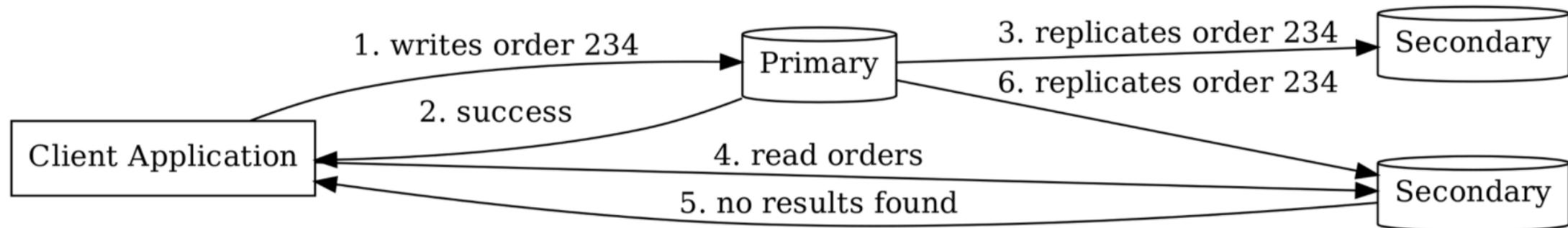
**Monotonic writes** Write operations that must precede other writes are executed before those other writes.

**Writes follow reads** Write operations that must occur after read operations are executed after those read operations.

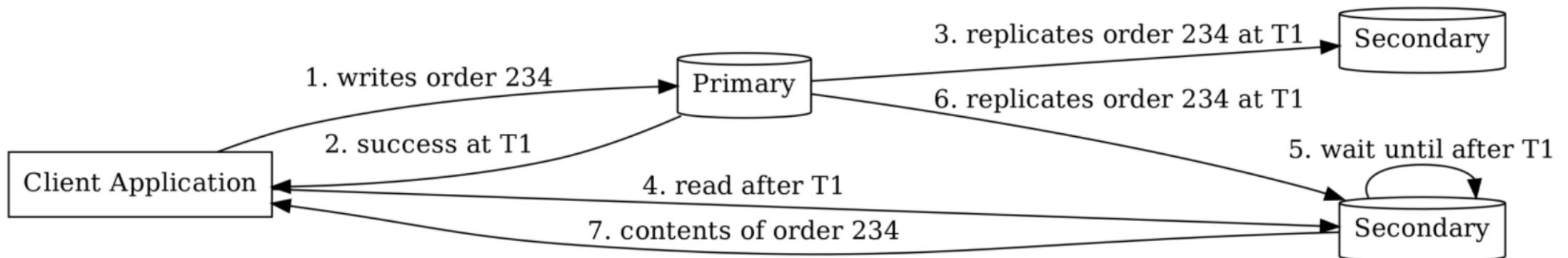


# Example in MongoDB

- Case 1 : No causal consistency



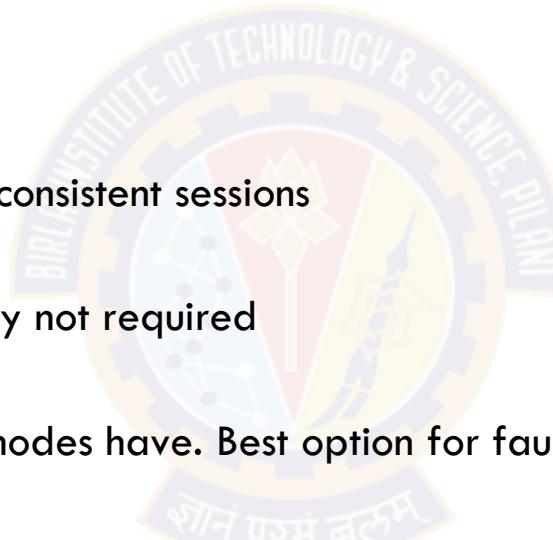
- Case 2: Causal consistency by making read to secondary wait



# MongoDB “read concerns”

Local, available, majority, linearizable

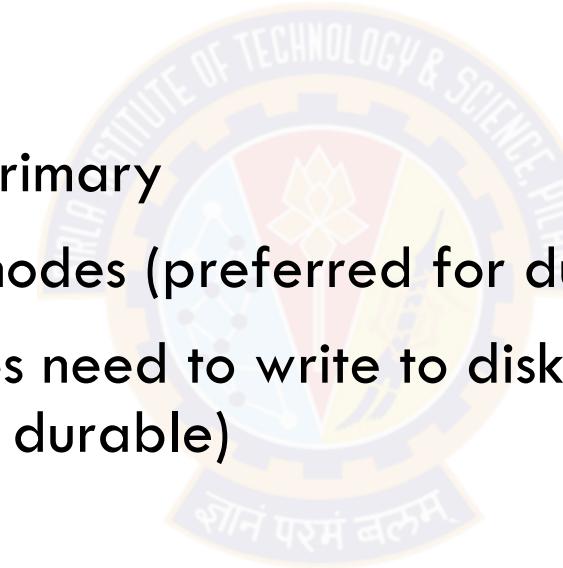
- local :
  - Client reads primary replica
  - Client reads from secondary in causally consistent sessions
- available:
  - Read on secondary but causal consistency not required
- majority :
  - If client wants to read what majority of nodes have. Best option for fault tolerance and durability.
- linearizable :
  - If client wants to read what has been written to majority of nodes before the read started.
  - Has to be read on primary
  - Only single document can be read



<https://docs.mongodb.com/v3.4/core/read-preference-mechanics/>

# MongoDB “write concerns”

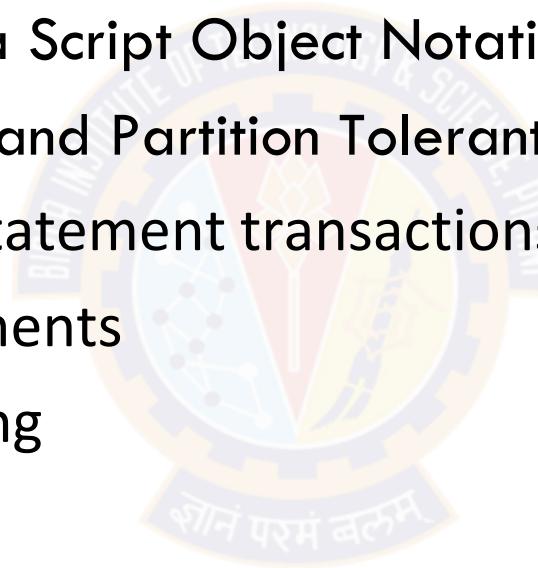
- how many replicas should ack
  - 1 - primary only
  - 0 - none
  - n - how many including primary
  - majority - a majority of nodes (preferred for durability)
- journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)
- timeout for write operation



<https://docs.mongodb.com/manual/reference/write-concern/>

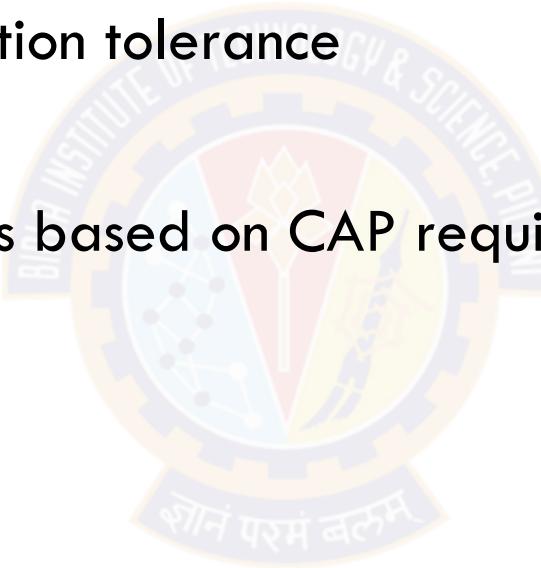
# MongoDB in a nutshell

- MongoDB is a non-relational. Open source, Distributed database
- It stores data into JSON (Java Script Object Notation) documents
- It adheres to CP (Consistency and Partition Tolerant) traits of Brewer's CAP Theorem
- It has NO support for multi-statement transactions
- It supports embedded documents
- It practices automatic sharding



# Topics for today

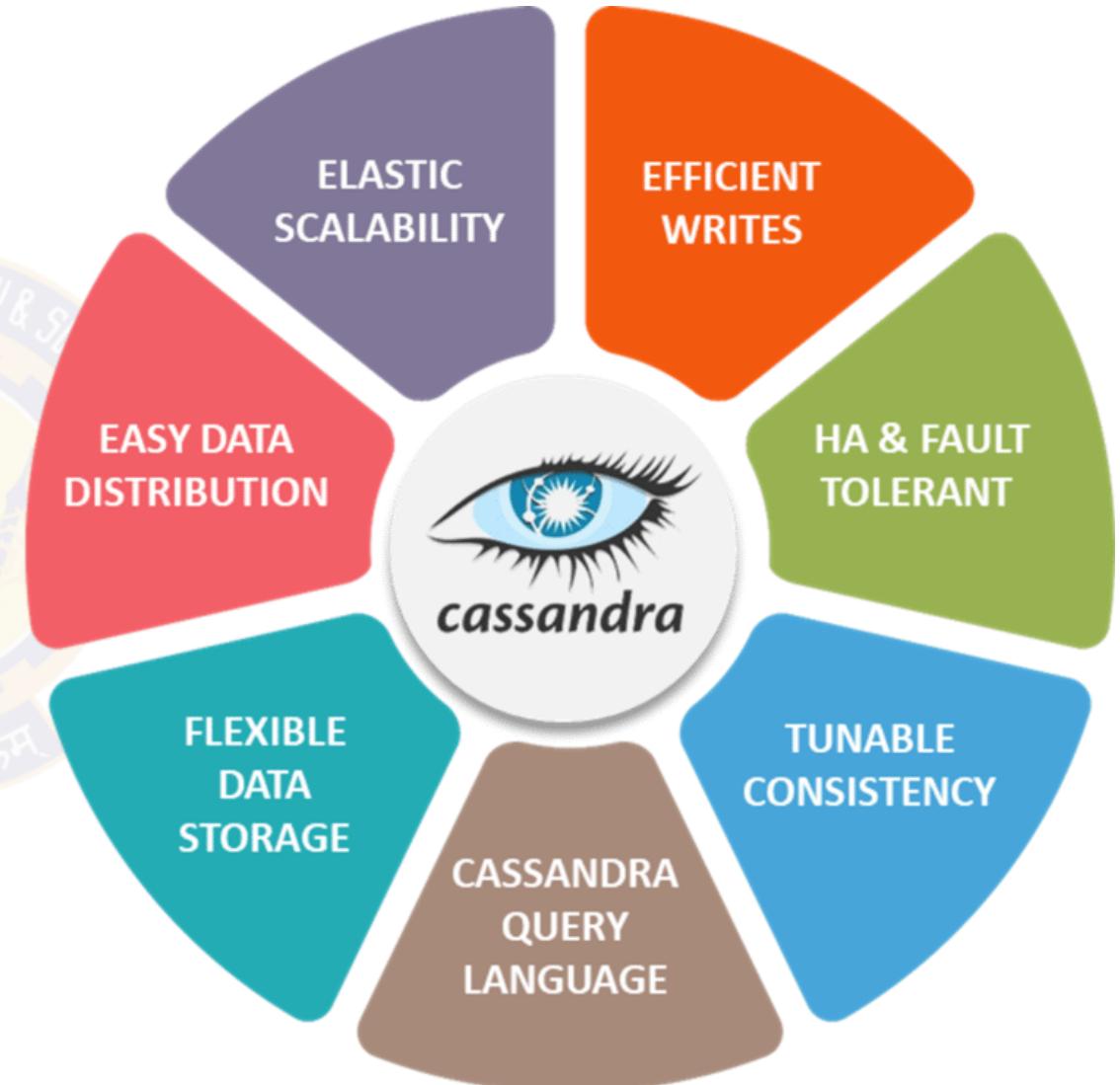
- Big Data analytics lifecycle
- Consistency, availability, partition tolerance
- CAP theorem
- Example BigData store options based on CAP requirement
  - MongoDB
  - **Cassandra**



- Initially developed by Facebook
  - Open-sourced in 2008
- Used by 1500+ businesses, e.g., Comcast, eBay, GitHub, Hulu, Instagram, Netflix, Best Buy, ...
- Column-family store
  - Supports key-value interface
  - Provides a SQL-like CRUD interface: CQL
- BASE consistency model is AP
  - Gossip protocol (constant communication) to establish consistency
  - Ring-based replication model

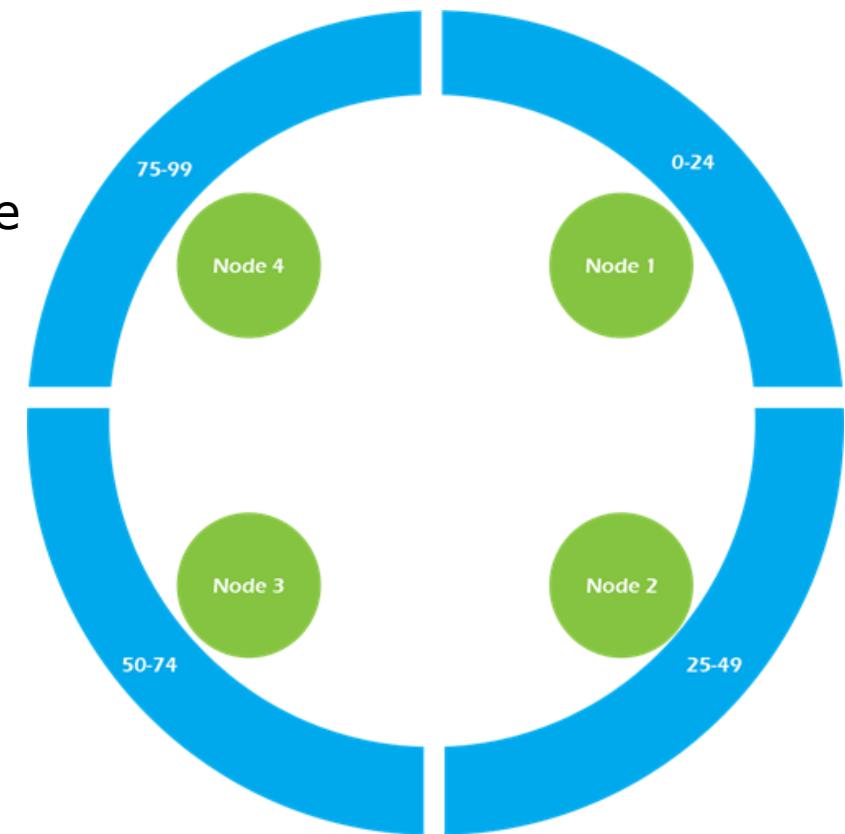
# Cassandra Features

- Massively scalable
- Master-less architecture
- Linear scale performance
- No single point of failure
- Automated Replication
- Peer-to-peer
- Written in Java
- Tunable consistency
- Clients - CQL and/or Thrift



# Cassandra Data Model: Partition Keys

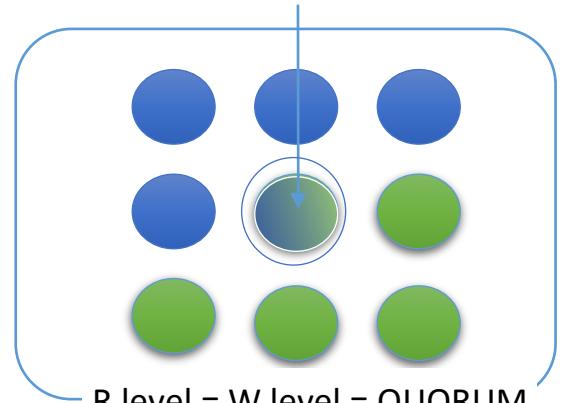
- Nodes in a Cassandra cluster owns a range of Keys (Tokens)
- Partition key is responsible for distributing data among nodes
- Partitions are groupings of data that will be co-located on storage of the node owning the key
- Partition keys are assigned to the nodes of the cluster
- Cassandra is organized into a cluster of nodes, with each node having an equal part of the partition key hashes.
- Imagine we have a four node Cassandra cluster.
- In the example cluster (ring):
- Node 1 is responsible for partition key hash values 0-24
- Node 2 is responsible for partition key hash values 25-49
- Node 3 is responsible for partition key hash values 50-74
- Node 4 is responsible for partition key hash values 75-99



# Cassandra

- Key-value store with columnar storage
- No primary replica - high partition tolerance and availability and levels of consistency
- Support for light transactions with “linearizable consistency”
- A Read or Write operation can pick a consistency level
  - ONE, TWO, THREE, ALL - 1,2,3 or all replicas respectively have to ack
  - QUORUM - majority have to ack
  - LOCAL\_QUORUM - majority within same datacenter have to ack
  - ...
- For “causal consistency” pick Read consistency level = Write consistency level = QUORUM
- Why ? At least one node will be common between write and read set so a read will get the last write of a data item
- What happens if read and write use LOCAL\_QUORUM ?
- If no overlap read and write sets then “Eventual consistency”

Read latest written value from common node



<https://cassandra.apache.org/doc/latest/architecture/dynamo.html>

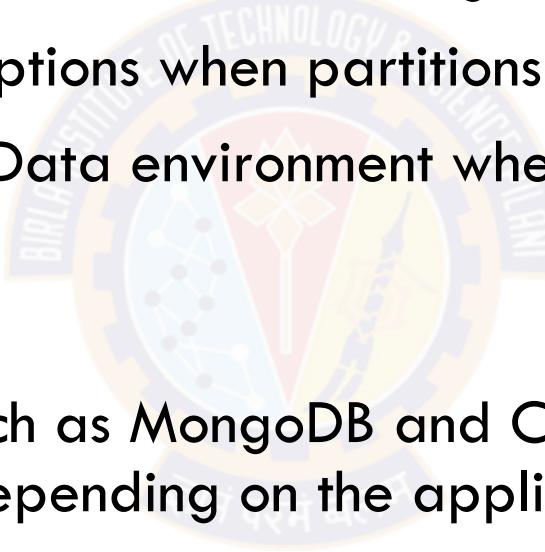
<https://cassandra.apache.org/doc/latest/architecture/guarantees.html#>

# Cassandra Data model: Comparison with RDBMS

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. <i>ROWxCOLUMN</i>	In Cassandra, a table is a list of "nested key-value pairs". <i>ROWxCOLUMNkeyxCOLUMNvalue</i>
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

# Summary

- What process steps must one should take in a Big Data Analytics project
- What is C, A, P and various options when partitions happen
- Why is this important for Big Data environment where faults may happen in large distributed systems
- CAP Theorem
- How example NoSQL DBs, such as MongoDB and Cassandra, can enable multiple consistency options depending on the application requirement





**Next Session:**  
Distributed programming

# Introduction to MongoDB

## Agenda

- What is MongoDB?
- Why MongoDB?
- Using JSON
- Creating or Generating a Unique Key
- Replication
- Sharding
- Terms used in RDBMS and MongoDB
- CRUD (Insert(), Update(), Save(), Remove(), Find())
- Aggregation
- Mongolimport
- MongoExport

# What is MongoDB?

MongoDB is:

1. Cross-platform.
2. Open source.
3. Non-relational.
4. Distributed.
5. NoSQL.
6. Document-oriented data store.

# Why MongoDB?

- Open Source
- Distributed
- Fast In-Place Updates
- Replication
- Full Index Support
- Rich Query Language
- Easy Scalability
- Auto sharding

# JSON (Java Script Object Notation)

## *Sample JSON Document*

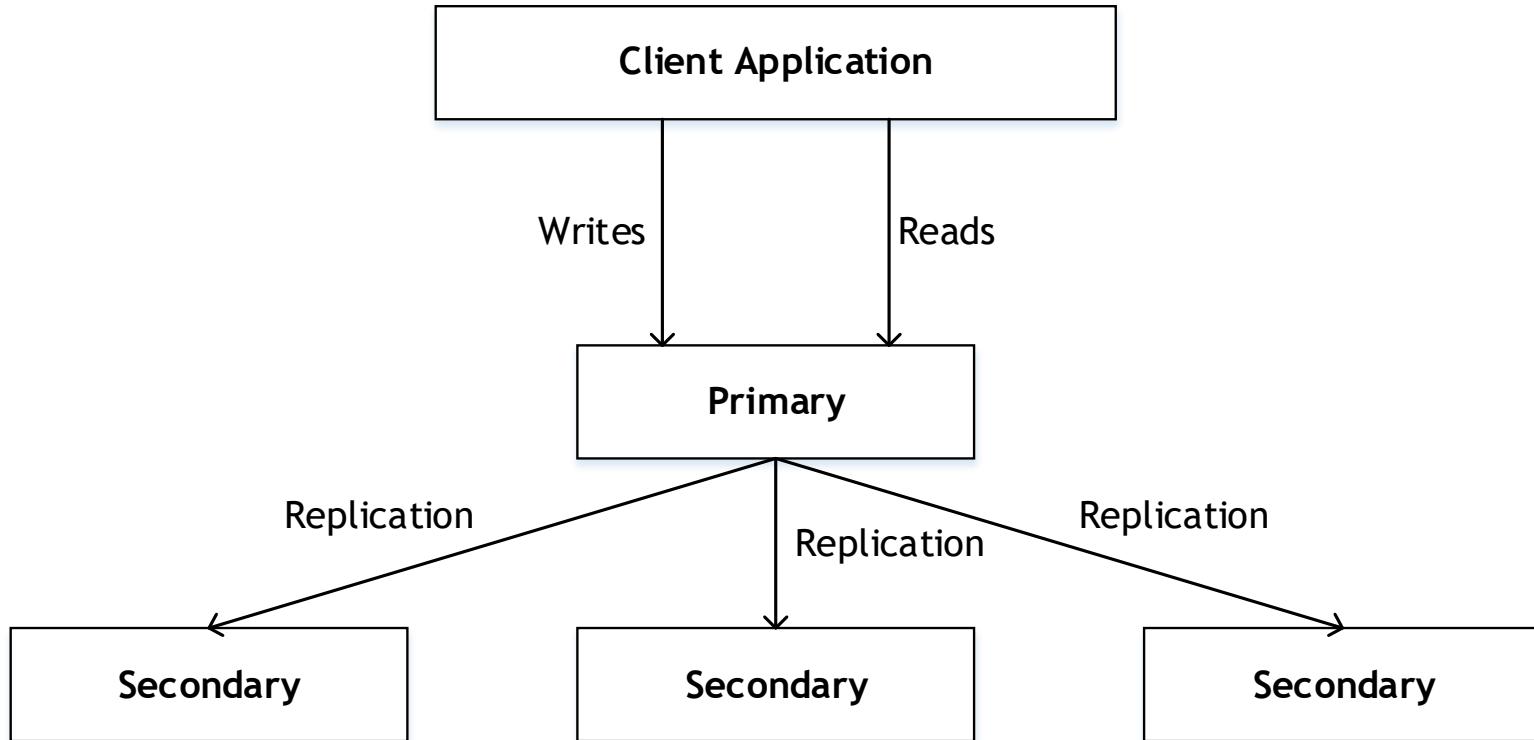
```
{  
  FirstName: John,  
  LastName: Mathews,  
  ContactNo: [+123 4567 8900, +123 4444 5555]  
}
```

## Unique Identifier

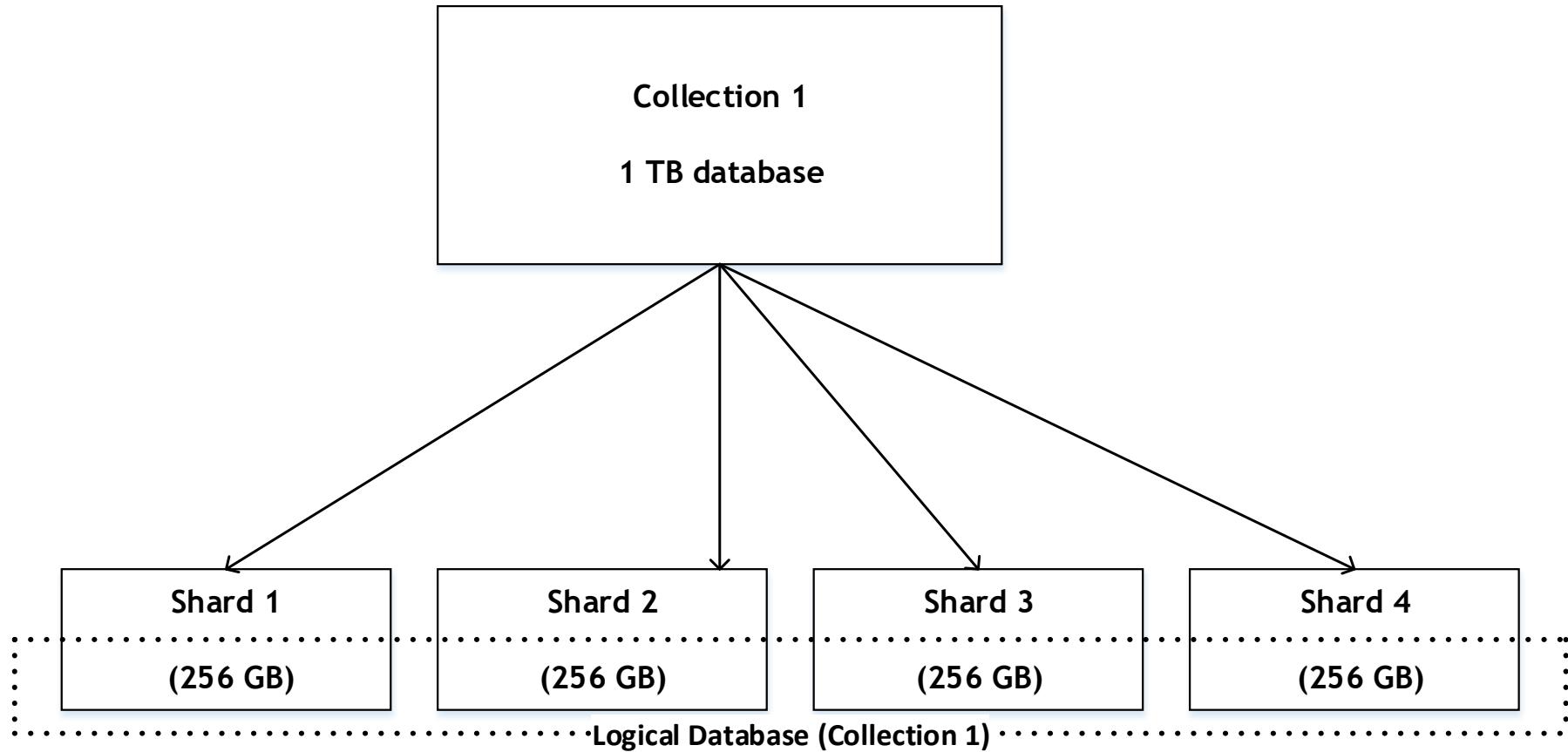
Each JSON document should have a unique identifier. It is the `_id` key.

0	1	2	3	4	5	6	7	8	9	10	11
Timestamp		Machine ID			Process ID		Counter				

# Replication in MongoDB



# Sharding in MongoDB



# Terms Used in RDBMS and MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields / Key Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is a identifier)

## Database creation / deletion

Create a database named myDB

- use myDB;

To confirm the existence of your database

- db;

To get a list of all databases

- show dbs;

To display list of collections in myDB

- show collections;

To display statistics that reflect the use state of a database

- db.stats();

To drop database myDB

- use myDB;
- db.dropDatabase();

## Insert Method

Create a collection by the name “Students” and store the following data in it.

```
db.createCollection("Students")
```

```
db.Students.insert({_id:1, StudName:"Michelle Jacintha", Grade: "VII", Hobbies: "Internet Surfing"})
```

List all documents in collection Students:

```
db.Students.find();
```

## Update Method

Insert the document for “Aryan David” into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from “Skating” to “Chess”.) Use “Update else insert” (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

```
db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});
```

## Find Method

To search for documents from the “Students” collection based on certain search criteria.

```
db.Students.find({StudName:"Aryan David"});
```

```
db.Students.find({Grade:"VII"});
```

## Find Method

To display only the StudName and Grade from all the documents of the Students collection. The identifier \_id should be suppressed and NOT displayed.

```
db.Students.find({}, {StudName:1,Grade:1,_id:0});
```

## Find Method

To find those documents where the Grade is set to ‘VII’

```
db.Students.find({Grade:{$eq:'VII'}}).pretty();
```

## Find Method

To find those documents from the Students collection where the Hobbies is set to either ‘Chess’ or is set to ‘Skating’.

```
db.Students.find ({Hobbies :{ $in: ['Chess','Skating']} }).pretty ();
```

## Find Method

To find documents from the Students collection where the StudName begins with “M”.

```
db.Students.find({StudName:/^M/}).pretty();
```

## Find Method

To find documents from the Students collection where the StudName has an “e” in any position.

```
db.Students.find({StudName:/e/}).pretty();
```

## Count Method

To find the number of documents in the Students collection.

```
db.Students.count();
```

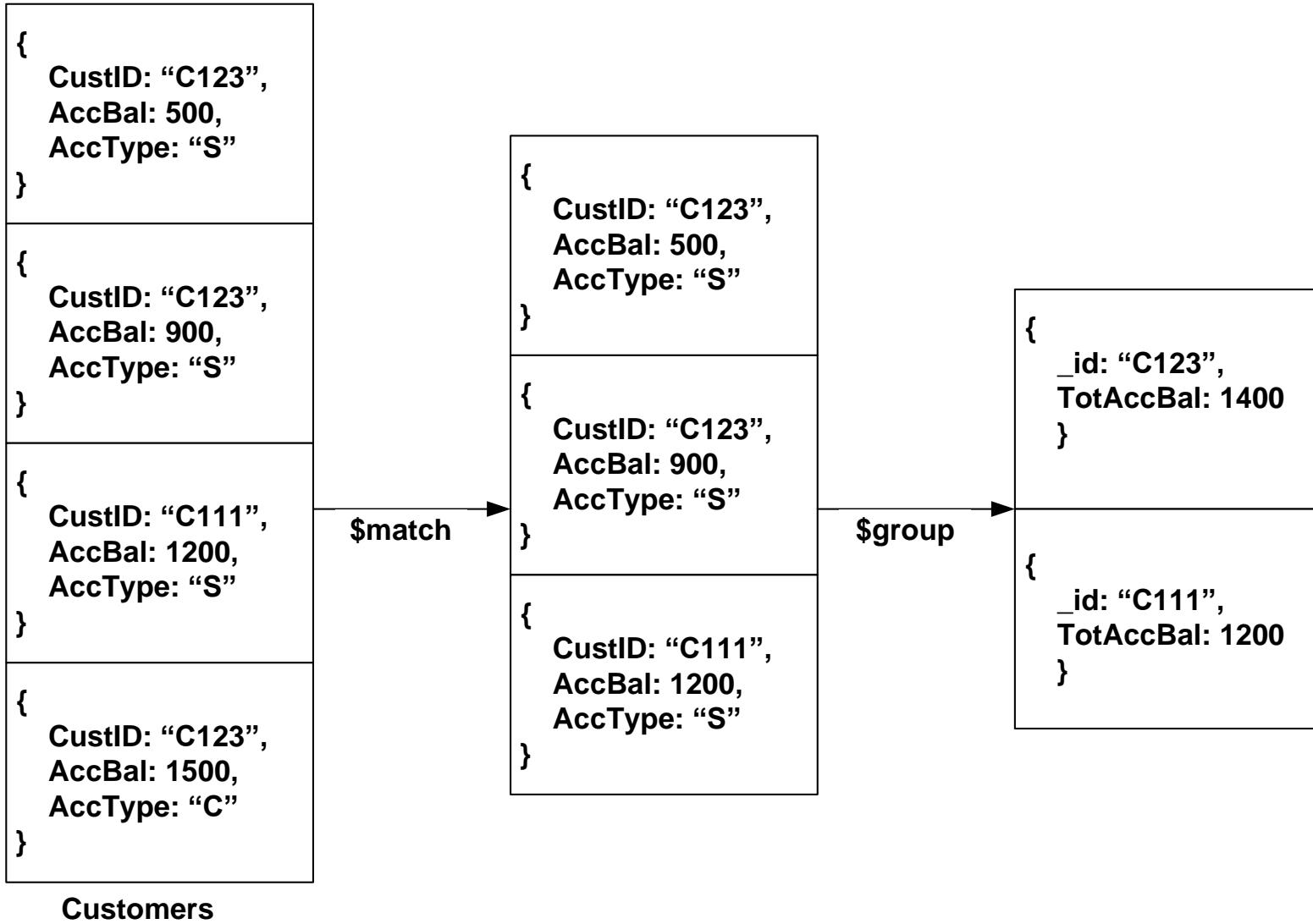
## Find Method

To sort the documents from the Students collection in the descending order of StudName.

```
db.Students.find().sort({StudName:-1}).pretty();
```

# Aggregate Function

# Aggregate Function



## Aggregate Function

First filter on “AccType:S” and then group it on “CustID” and then compute the sum of “AccBal” and then filter those documents wherein the “TotAccBal” is greater than 1200, use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },  
{ $group : { _id : "$CustID", TotAccBal : { $sum : "$AccBal" } } },  
{ $match : {TotAccBal : { $gt : 1200 } }});
```

## Import data from a CSV file

Given a CSV file “sample.txt” in the D: drive, import the file into the MongoDB collection, “SampleJSON”. The collection is in the database “test”.

```
mongoimport --db test --collection SampleJSON --type csv --headerline --file d:\sample.txt
```

## Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from “Customers” collection in the “test” database into a CSV file “Output.txt” in the D: drive.

```
mongoexport --db test --collection Customers --csv --fieldFile d:\fields.txt --out d:\output.txt
```

# MongoDB Data Example

Collection inventory

```
{  
    item: "ABC2",  
    details: { model: "14Q3", manufacturer: "M1 Corporation" },  
    stock: [ { size: "M", qty: 50 } ],  
    category: "clothing"  
}  
  
{  
    item: "MNO2",  
    details: { model: "14Q3", manufacturer: "ABC Co." },  
    stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L",  
        qty: 1 } ],  
    category: "clothing"  
}
```

Document insertion

```
db.inventory.insert(  
    {  
        item: "ABC1",  
        details: {model: "14Q3",manufacturer: "XYZ  
Company"},  
        stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 }  
        ],  
        category: "clothing"  
    }  
)
```

# Example of Simple Query

Collection orders

```
{  
  _id: "a",  
  cust_id: "abc123",  
  status: "A",  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price:  
    3 },  
           { sku: "nnn", qty: 5, price: 2 } ]  
}  
  
{  
  _id: "b",  
  cust_id: "abc124",  
  status: "B",  
  price: 12,  
  items: [ { sku: "nnn", qty: 2, price: 2  
    },  
           { sku: "ppp", qty: 2, price: 4 } ]  
}
```

db.users.find(

  { status: "A" },

  { cust\_id: 1, price: 1,  
    \_id: 0 }

)

*selection*

*projection*

In SQL it would look like this:

```
SELECT cust_id, price  
FROM orders  
WHERE status="A"
```

Results

```
{  
  cust_id: "abc123",  
  price: 25  
}
```

**Thank you**



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 5: Hadoop architecture and filesystem

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **Hadoop architecture overview**

- ✓ Components
- ✓ Hadoop 1 vs Hadoop 2

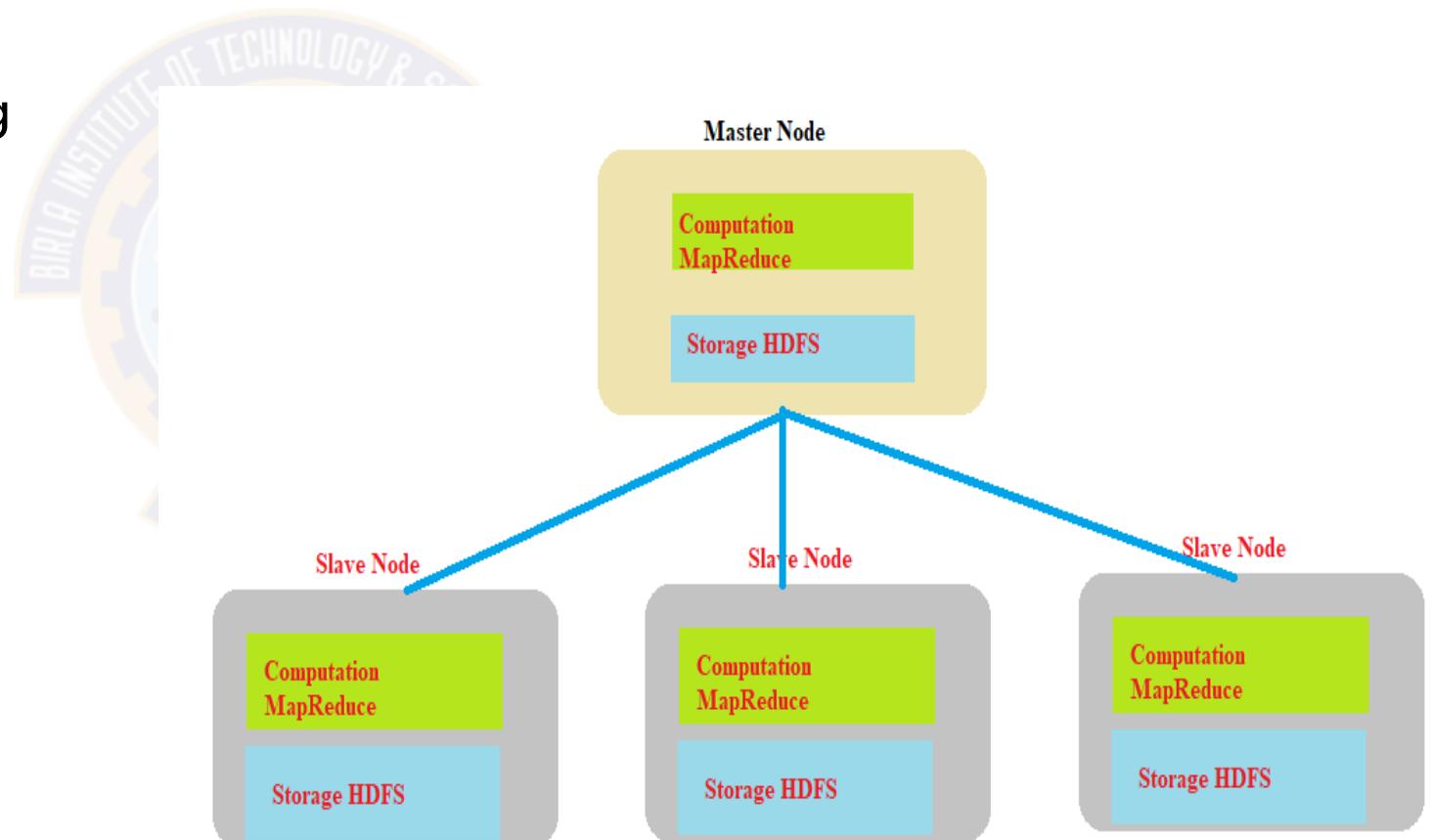
- **HDFS**

- ✓ Architecture
- ✓ Robustness
- ✓ Blocks and replication strategy
- ✓ Read and write operations
- ✓ File formats
- ✓ Commands



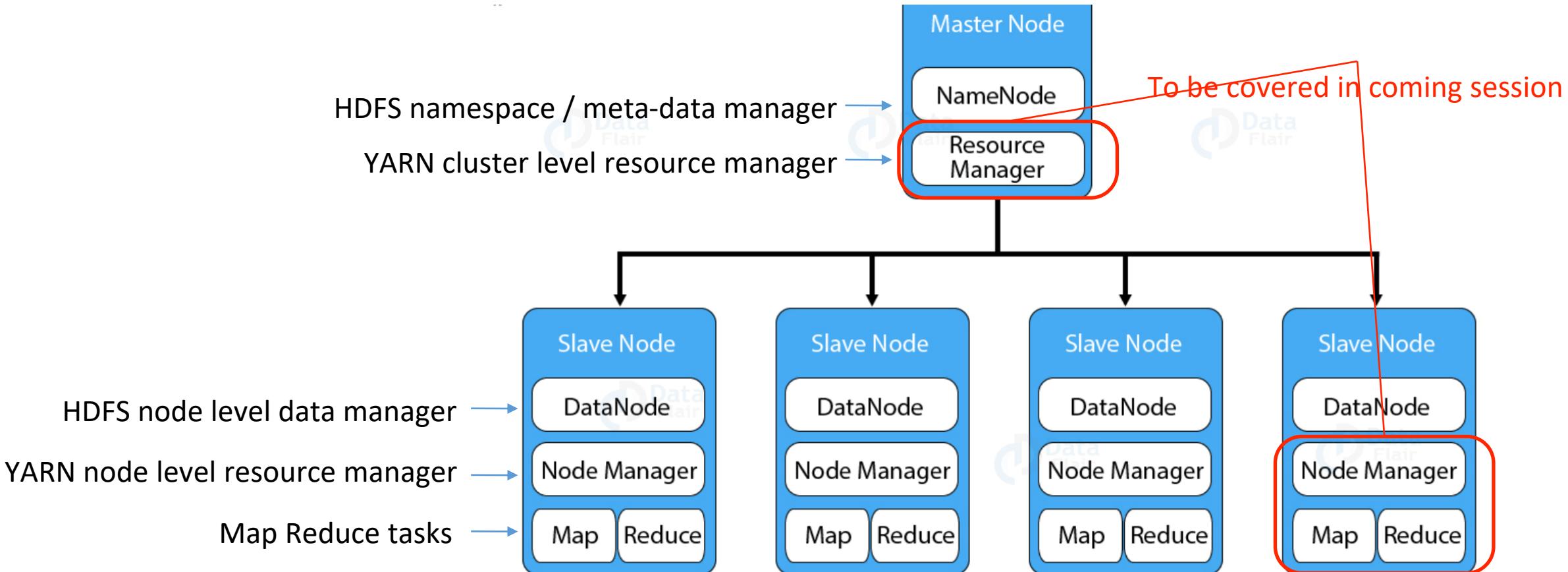
# Hadoop - Data and Compute layers

- A data storage layer
  - A Distributed File System - HDFS
- A data processing layer
  - MapReduce programming



# Hadoop 2 - Architecture

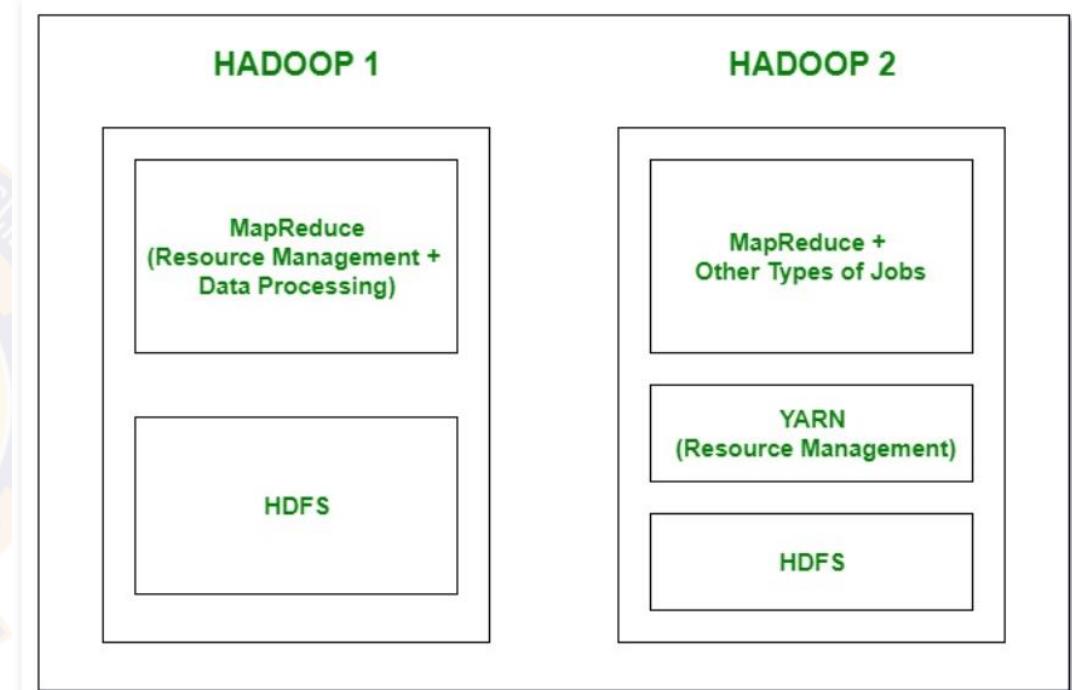
- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



**Note:** YARN Resource Manager also uses application level App Master processes on slave nodes for application specific resource management

# What changed from Hadoop 1 to Hadoop 2

- Hadoop 1:
  - MapReduce was coupled with resource management
    - Hadoop 2 brought in YARN as a resource management capability and MapReduce is only about data processing.
- Hadoop 1: Single Master node with NameNode is a SPOF
  - Hadoop 2 introduced active-passive and other HA configurations besides secondary NameNodes
- Hadoop 1: Only MapReduce programs
  - In Hadoop 2, non MR programs can be run by YARN on slave nodes (since decoupled from MapReduce) as well support for non-HDFS storage, e.g. Amazon S3 etc.



# Hadoop Distributions

- Open source Apache project
- Core components :

- ✓ Hadoop Common
- ✓ Hadoop Distributed File System
- ✓ Hadoop YARN
- ✓ Hadoop MapReduce



Hadoop Distribution

Amazon Web Services  
Elastic Map Reduce

Apache Hadoop

Intel Distribution

Cloudera CDH

MapR

EMC Greenplum HD

MS BigData Solution

IBM InfoSphere BigInsights

Hortonworks

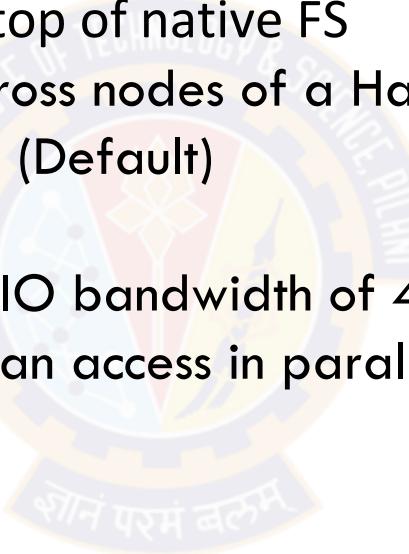
# Topics for today

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ **Architecture**
  - ✓ Robustness
  - ✓ Blocks and replication strategy
  - ✓ Read and write operations
  - ✓ File formats
  - ✓ Commands



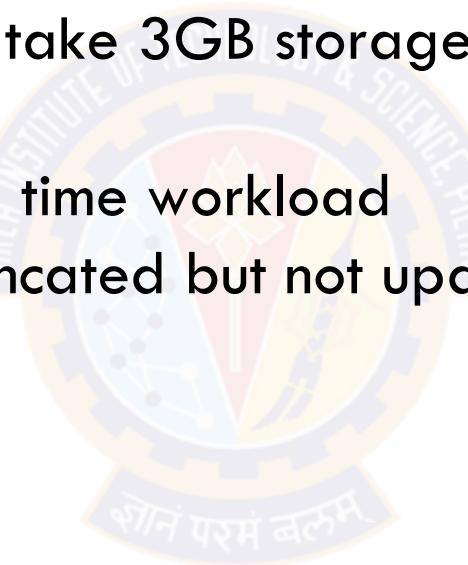
# HDFS Features (1)

- A DFS stores data over multiple nodes in a cluster and allows multi-user access
  - ✓ Gives a feeling to the user that the data is on single machine
  - ✓ HDFS is a Java based DFS that sits on top of native FS
  - ✓ Enables storage of very large files across nodes of a Hadoop cluster
  - ✓ Data is split into large blocks : 128MB (Default)
- Scale through parallel data processing
  - ✓ 1 node with 1TB storage can have an IO bandwidth of 400MBps across 4 IO channels = 43 min
  - ✓ 10 nodes with partitioned 1 TB data can access in parallel that data in 4.3 min



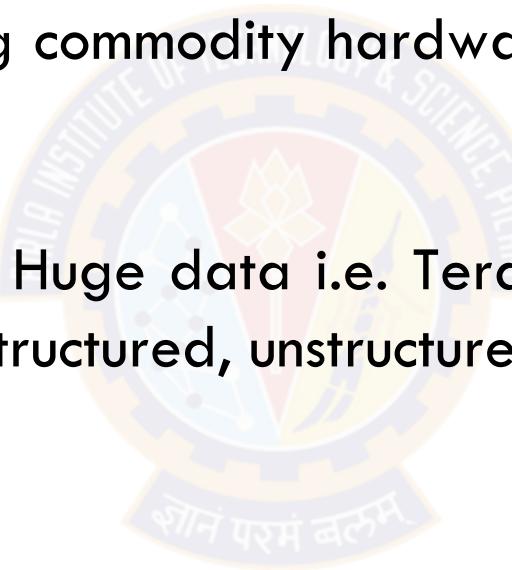
## HDFS Features (2)

- Fault tolerance through replication
  - ✓ Default replication factor = 3 for every block
  - ✓ So 1 GB data can actually take 3GB storage
- Consistency
  - ✓ Write once and read many time workload
  - ✓ Files can be appended, truncated but not updated at any arbitrary point



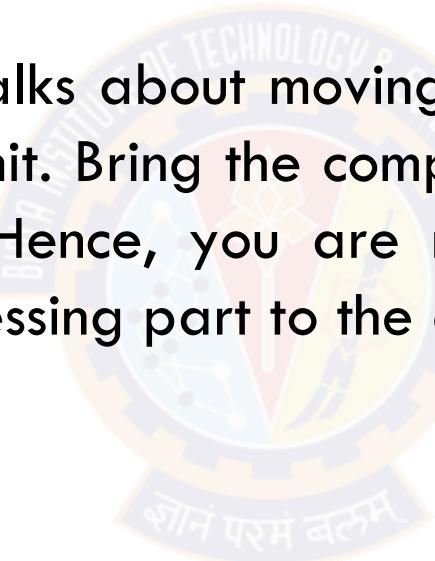
## HDFS Features (3)

- Cost: Typically deployed using commodity hardware for low TCO - so adding more nodes is cost-effective
- Variety and Volume of Data: Huge data i.e. Terabytes & petabytes of data and different kinds of data - structured, unstructured or semi structured.



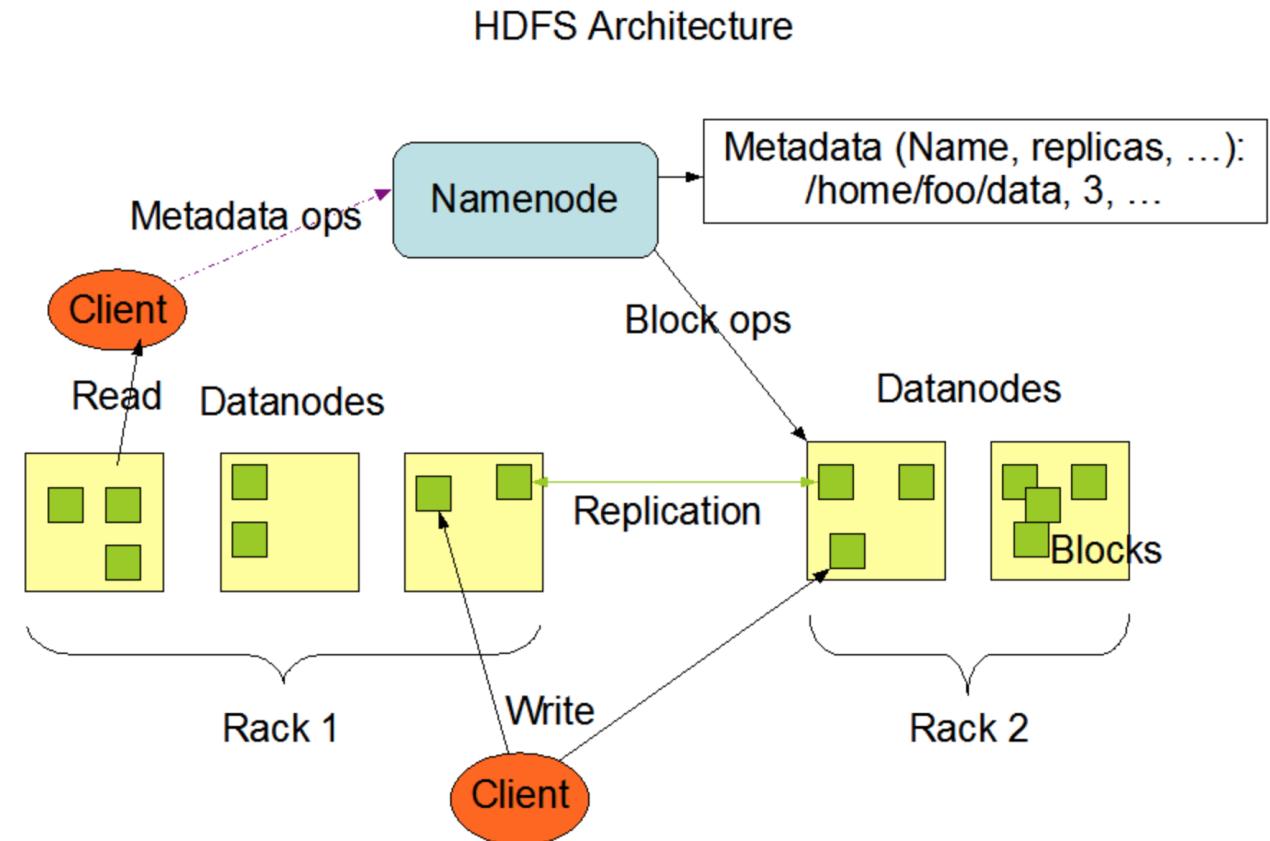
## HDFS Features (4)

- Data Integrity: HDFS nodes constantly verify checksums to preserve data integrity. On error, new copies are created and old copies are deleted.
- Data Locality: Data locality talks about moving processing unit to data rather than the data to processing unit. Bring the computation part to the data nodes where the data is residing. Hence, you are not moving the data, you are bringing the program or processing part to the data.



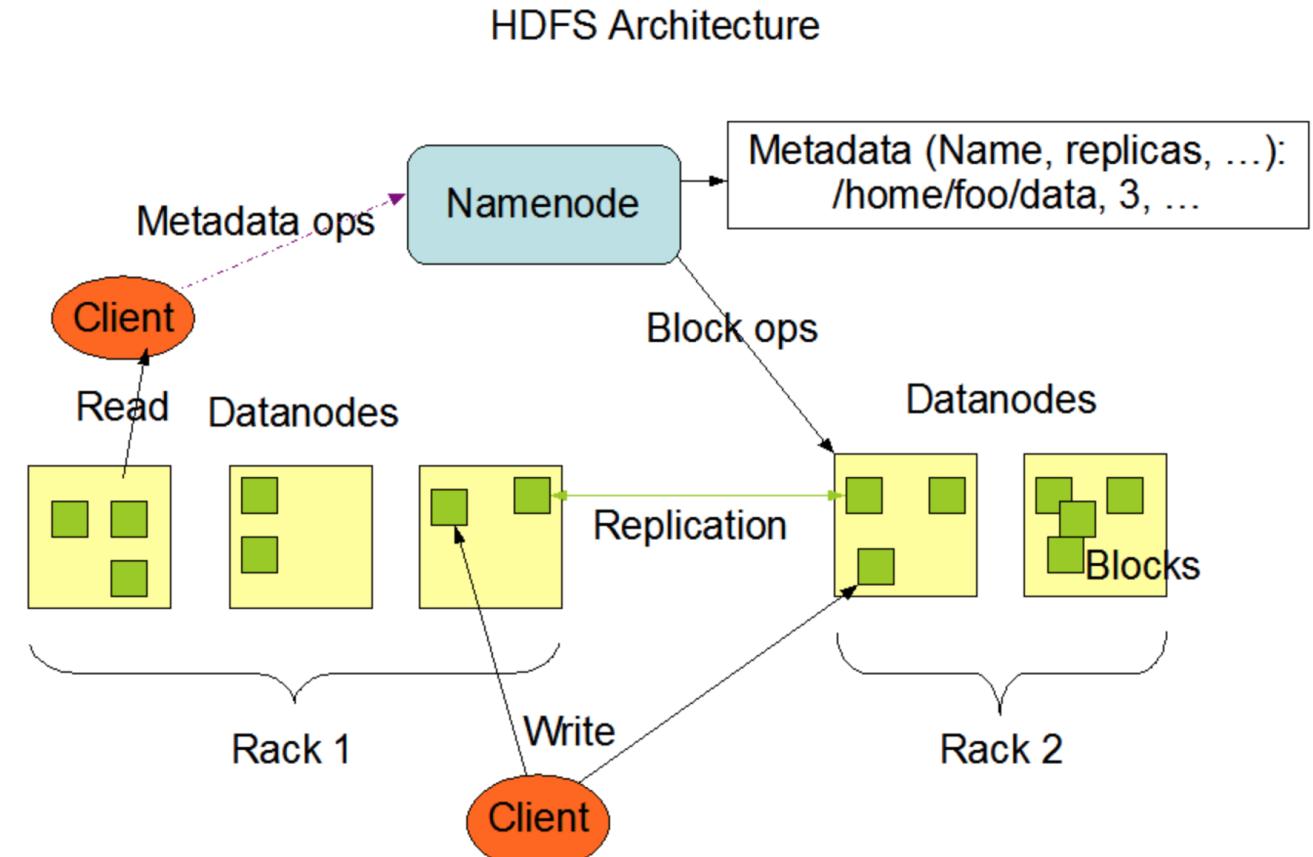
# HDFS Architecture - Master node

- Master slave architecture within a HDFS cluster
- One master node with NameNode
  - Maintains namespace - Filename to blocks and their replica mappings
  - Serves as arbitrator and doesn't handle actual data flow
  - HDFS client app interacts with NameNode for metadata



# HDFS Architecture - Slave nodes

- Multiple slave nodes with one DataNode per slave
  - Serves block R/W from Clients
  - Serves Create/Delete/Replicate requests from NameNode
  - DataNodes interact with each other for pipeline reads and writes.



# Functions of a NameNode

- Maintains namespace in HDFS with 2 files
  - FsImage: Contains mapping of blocks to file, hierarchy, file properties / permissions
  - EditLog: Transaction log of changes to metadata in FsImage
- Does not store any data - only meta-data about files
- Runs on Master node while DataNodes run on Slave nodes
- HA can be configured (discussed later)
- Records each change that takes place to the meta-data. e.g. if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- Receives periodic Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- Ensure replication factor is maintained across DataNode failures
  - In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes

# Where are fsimage and edit logs ?

```
[root@centos-s-4vcpu-8gb-blr1-01 current]# pwd  
/home/hadoop/hadoopdata/hdfs/namenode/current  
[root@centos-s-4vcpu-8gb-blr1-01 current]# ls  
VERSION  
edits_0000000000000001-0000000000000002  
edits_0000000000000003-0000000000000100  
edits_000000000000000101-0000000000000102  
edits_000000000000000103-0000000000000104  
edits_000000000000000105-0000000000000106  
edits_000000000000000107-0000000000000107  
edits_000000000000000108-0000000000000108  
edits_000000000000000109-0000000000000109  
edits_000000000000000110-0000000000000111  
edits_000000000000000112-0000000000000112  
edits_000000000000000113-0000000000000114  
edits_000000000000000115-0000000000000115  
edits_000000000000000116-0000000000000117  
edits_000000000000000118-0000000000000199  
edits_000000000000000200-0000000000000200  
edits_000000000000000201-0000000000000202  
edits_000000000000000203-0000000000000204  
You have mail in /var/spool/mail/root  
[root@centos-s-4vcpu-8gb-blr1-01 current]# █  
edits_000000000000000205-000000000000000206  
edits_000000000000000207-000000000000000216  
edits_000000000000000217-000000000000000298  
edits_000000000000000299-000000000000000381  
edits_000000000000000382-000000000000000383  
edits_000000000000000384-000000000000000384  
edits_000000000000000385-000000000000000386  
edits_000000000000000387-000000000000000558  
edits_000000000000000559-000000000000000560  
edits_000000000000000561-000000000000000683  
edits_000000000000000684-000000000000000685  
edits_000000000000000686-000000000000000776  
edits_000000000000000777-000000000000000893  
edits_000000000000000894-000000000000000971  
edits_000000000000000972-000000000000000973  
edits_000000000000000974-000000000000000978  
edits_000000000000000979-000000000000000980  
edits_000000000000000981-000000000000000982  
edits_000000000000000983-000000000000000984  
edits_000000000000000985-000000000000000986  
edits_000000000000000987-000000000000000988  
edits_000000000000000989-000000000000000990  
edits_000000000000000991-000000000000000992  
edits_000000000000000993-000000000000000994  
edits_000000000000000995-000000000000000996  
edits_000000000000000997-000000000000000998  
edits_000000000000000999-00000000000001000  
edits_0000000000000001001-00000000000001002  
edits_0000000000000001003-00000000000001004  
edits_0000000000000001005-00000000000001006  
edits_inprogress_00000000000001007  
fsimage_00000000000001004  
fsimage_00000000000001004.md5  
fsimage_00000000000001006  
fsimage_00000000000001006.md5  
seen_txid
```

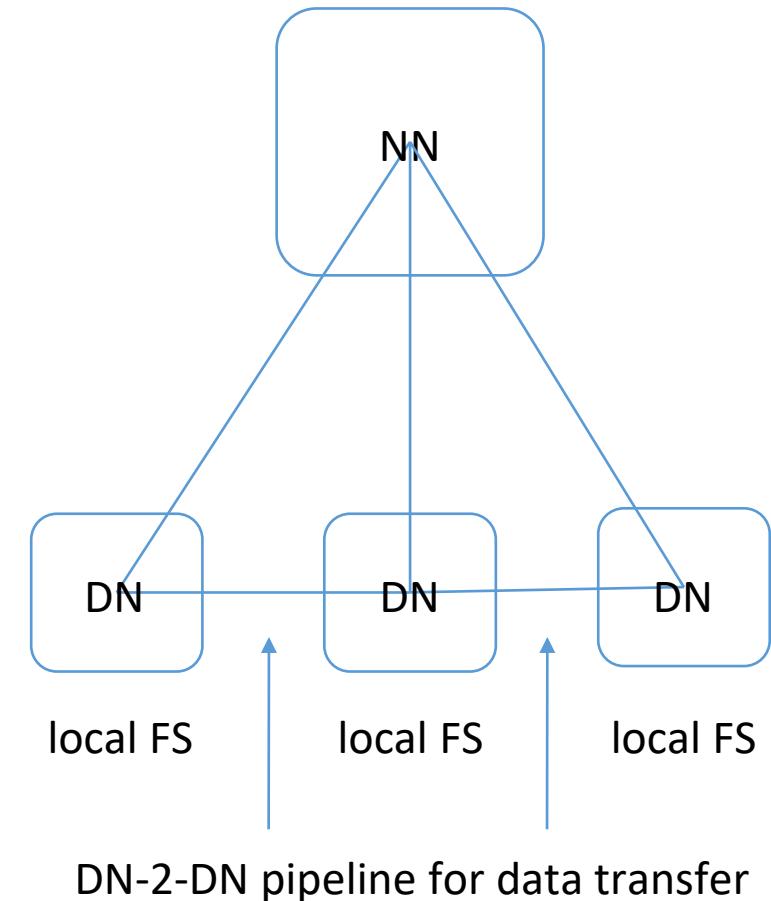
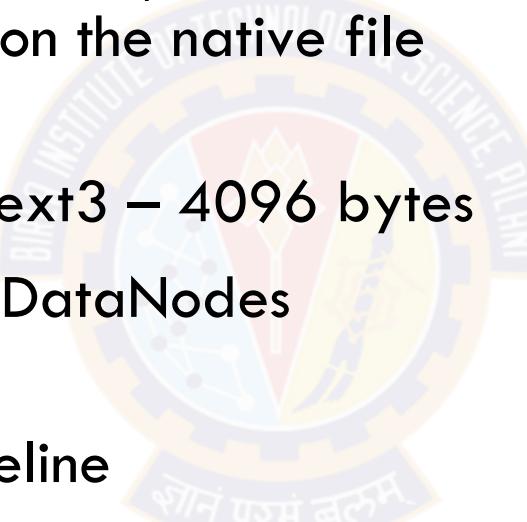
# Namenode - What happens on start-up

1. Enters into safe mode
  - ✓ Check for status of Data nodes on slaves
    - Does not allow any Datanode replications in this mode
    - Gets heartbeat and block report from Datanodes
    - Checks for minimum Replication Factor needed for configurable majority of blocks
  - ✓ Updates meta-data (this is also done at checkpoint time)
    - Reads FslImage and EditLog from disk into memory
    - Applies all transactions from the EditLog to the in-memory version of FslImage
    - Flushes out new version of FslImage on disk
    - Keeps latest FslImage in memory for client requests
    - Truncates the old EditLog as its changes are applied on the new FslImage
2. Exits safe mode
3. Continues with further replications needed and client requests

\* [ *hdfs dfsadmin -safemode enter | leave | get | wait | forceExit* ]

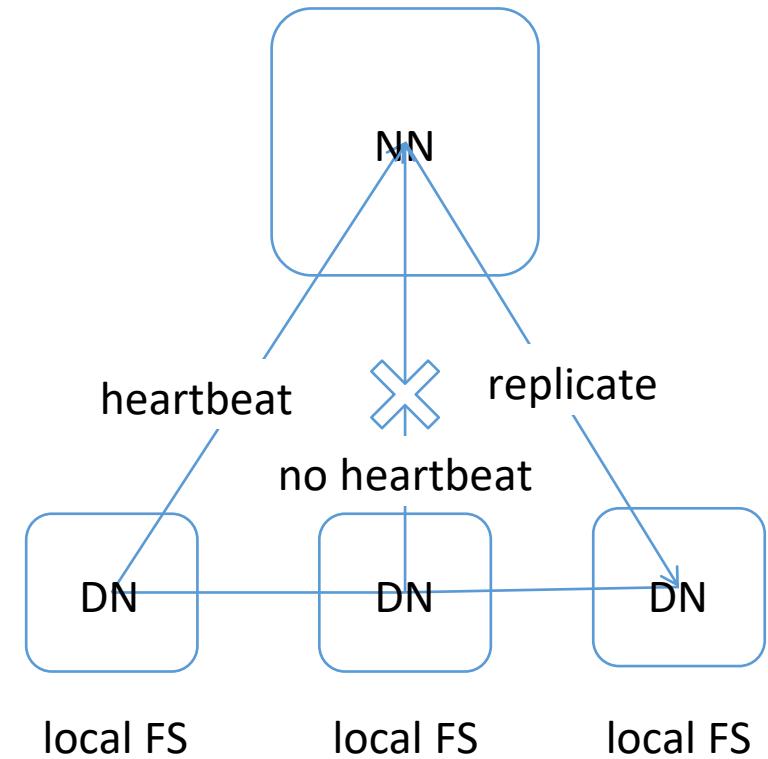
# Functions of a DataNode (1)

- Each slave in cluster runs a DataNode
- Nodes store actual data blocks and R/W data for the HDFS clients as regular files on the native file system, e.g. ext2 or ext3
- Default block size for ext2 and ext3 – 4096 bytes
- During pipeline read and write, DataNodes communicate with each other
  - We will discuss what's a pipeline
- No additional HA because blocks are anyway replicated



# Functions of a DataNode (2)

- DataNode continuously sends heartbeat to NameNode (default 3 sec)
  - ✓ To ensure the connectivity with NameNode
- If no heartbeat message from DataNode, NameNode replicates that DataNode within the cluster and removes the DN from the meta-data records
- DataNodes also send a BlockReport on start-up / periodically containing file list
- Applies some heuristic to subdivide files into directories based on limits of local FS but has no knowledge of HDFS level files



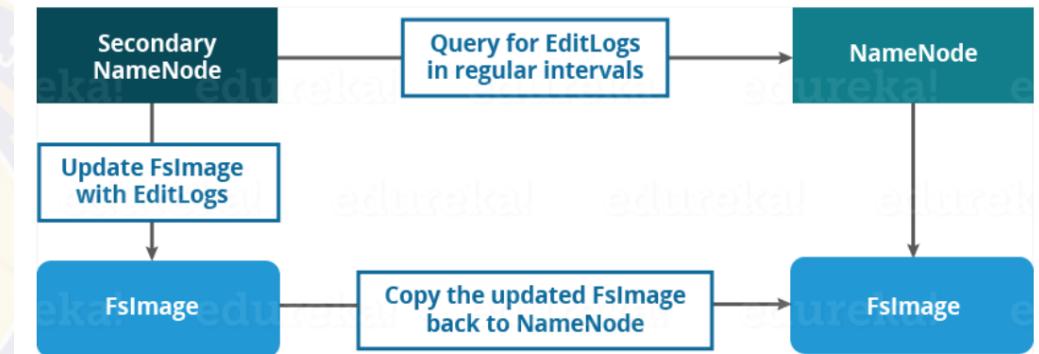
# Topics for today

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ **Robustness**
  - ✓ Blocks and replication strategy
  - ✓ Read and write operations
  - ✓ File formats
  - ✓ Commands



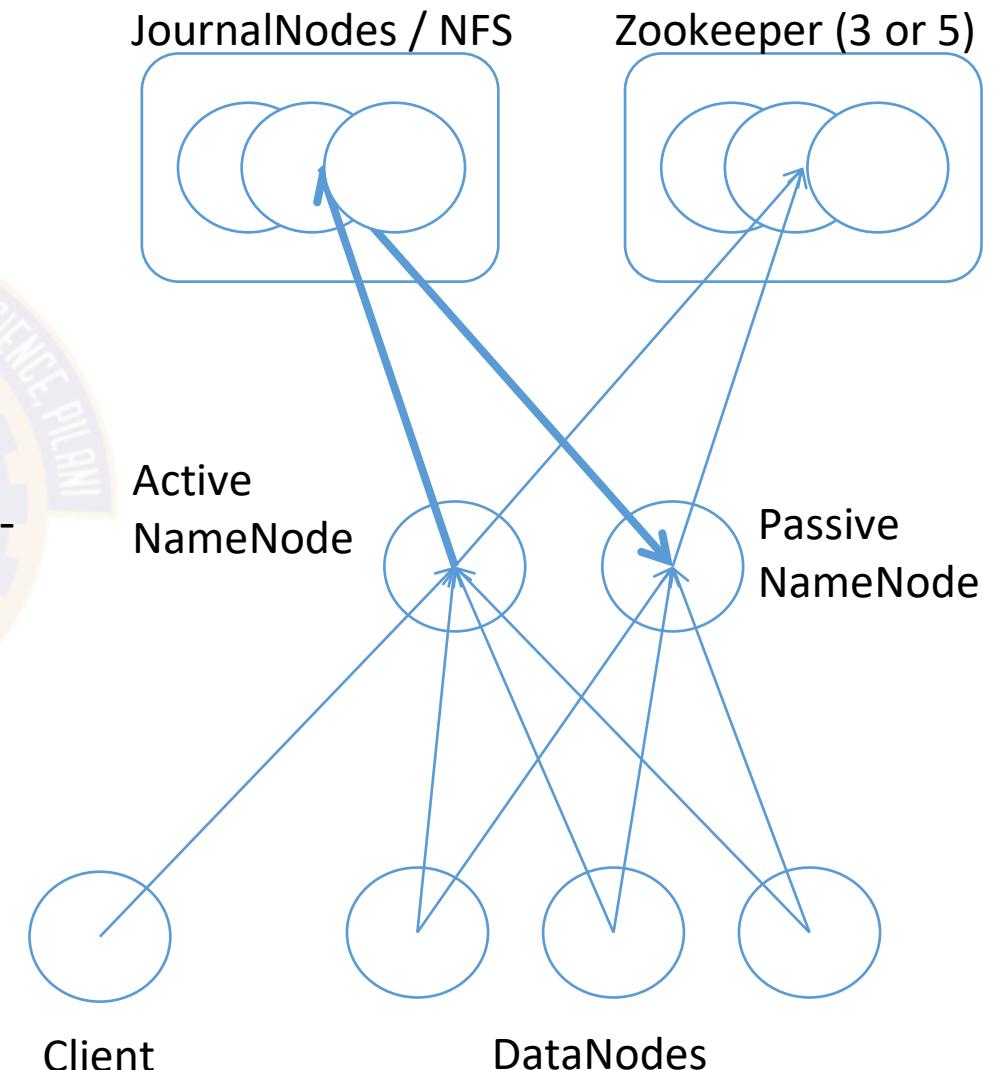
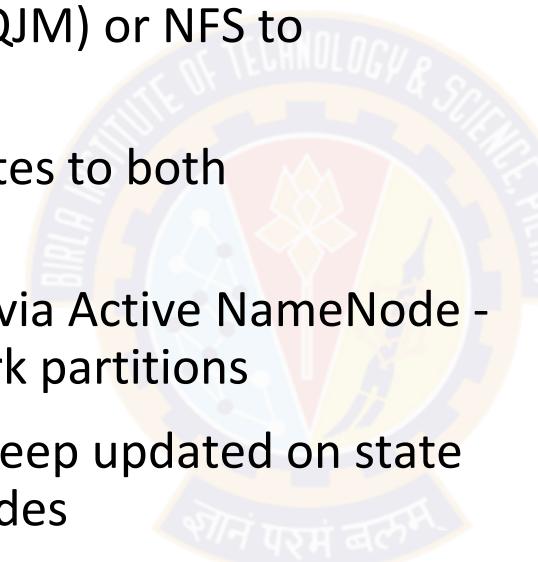
# Hadoop 2: Introduction of Secondary NameNode

- In the case of failure of NameNode,
  - ✓ The secondary NameNode can be configured manually to bring up the cluster
  - ✓ But it does not record any real-time changes that happen to the HDFS metadata
- The Secondary NameNode constantly reads all the file systems and metadata from the RAM of the NameNode (snapshot) and writes to its local file system.
- It is responsible for combining the EditLogs with Fslimage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to Fslimage.
- Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.
- After recover from a failure, the new Fslimage is copied back to the NameNode, which is used whenever the NameNode is started the next time.



# HA configuration of NameNode

- Active-Passive configuration can also be setup with a standby NameNode
- Can use a Quorum Journal Manager (QJM) or NFS to maintain shared state
- DataNodes send heartbeats and updates to both NameNodes.
- Writes to JournalNodes only happens via Active NameNode - avoids “split brain” scenario of network partitions
- Standby reads from JournalNodes to keep updated on state as well as latest updates from DataNodes
- Zookeeper session may be used for failure detection and election of new Active



# Other robustness mechanisms

- Types of failures - DataNode, NameNode failures and network partitions
- Heartbeat from DataNode to NameNode for handling DN failures
  - ✓ When data node heartbeat times out (10min) NameNode updates state and starts pointing clients to other replicas.
  - ✓ Timeout (10min) is high to avoid replication storms but can be set lower especially if clients want to read recent data and avoid stale replicas.
- Cluster rebalancing by keeping track of RF per block and node usage
- Checksums stored in NameNode for blocks written to DataNodes to check data integrity on corruption on node / link and software bugs

# Communication protocols

- TCP/IP at network level
- RPC abstraction on HDFS specific protocols
  - Clients talk to HDFS using Client protocol
  - DataNode and NameNode talk using DataNode protocol
- RPC is always initiated by DataNode to NameNode and not vice-versa
  - For better fault tolerance and NameNode state maintenance

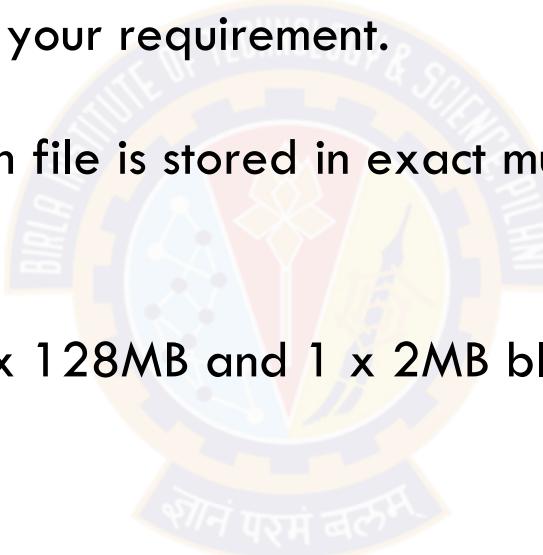
# Topics for today

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ Robustness
  - ✓ **Blocks and replication strategy**
  - ✓ Read and write operations
  - ✓ File formats
  - ✓ Commands



# Blocks in HDFS

- HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster.
- The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement.
- It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128 MB, 256 MB etc.).
  - A file of size 514 MB can have  $4 \times 128\text{MB}$  and  $1 \times 2\text{MB}$  blocks
  - Why large block of 128MB ?
    - HDFS is used for TB/PB size files and small block size will create too much meta-data
    - Larger blocks will further reduce the “indexing” at block level, impact load balancing across nodes etc.



# How to see blocks of a file in HDFS - fsck

```
[root@centos-s-4vcpu-8gb-blr1-01 bds]# hdfs fsck /SalesJan2009.csv -files -blocks
21/06/12 20:46:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
pllicable
Connecting to namenode via http://localhost:50070/fsck?ugi=root&files=1&blocks=1&path=%2FSalesJan2009.csv
FSCK started by root (auth:SIMPLE) from /127.0.0.1 for path /SalesJan2009.csv at Sat Jun 12 20:46:47 IST 2021
/SalesJan2009.csv 123637 bytes, 1 block(s):  OK
0. BP-235233240-127.0.0.1-1618685260802:blk_1073741825_1001 len=123637 Live_repl=1

Status: HEALTHY
Total size:    123637 B
Total dirs:    0
Total files:   1
Total symlinks:        0
Total blocks (validated): 1 (avg. block size 123637 B)
Minimally replicated blocks: 1 (100.0 %)
Over-replicated blocks:    0 (0.0 %)
Under-replicated blocks:   0 (0.0 %)
Mis-replicated blocks:    0 (0.0 %)
Default replication factor: 1
Average block replication: 1.0
Corrupt blocks:          0
Missing replicas:         0 (0.0 %)
Number of data-nodes:     1
Number of racks:          1
FSCK ended at Sat Jun 12 20:46:47 IST 2021 in 5 milliseconds

The filesystem under path '/SalesJan2009.csv' is HEALTHY
```

# HDFS Blocksize Vs Input Split

Example.txt

File split into  
2 blocks

130 MB

Block 1

Block  
2

128 MB

2 MB

Logical grouping  
of blocks

Block 1

Block  
2

InputSplit

\* Set mapreduce.input.fileinputformat.split.minsize parameter in mapred-site.xml

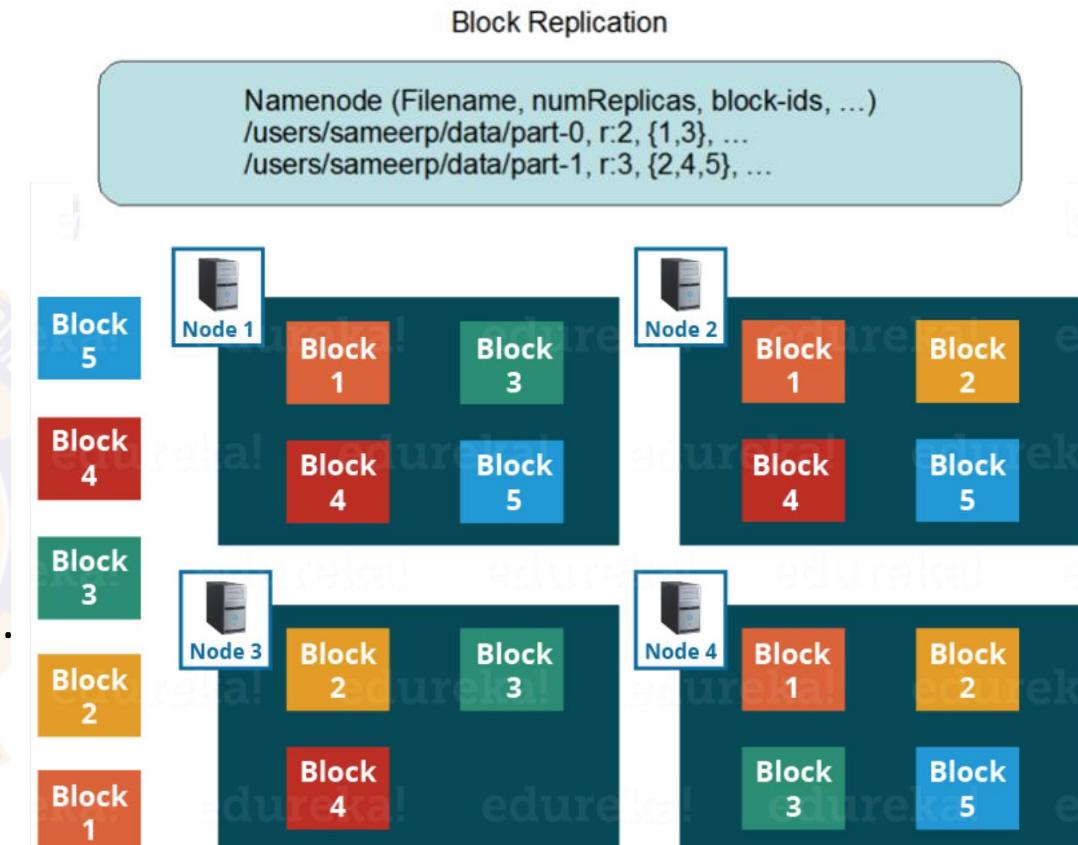
# HDFS on local FS

- Find / configure the root of HDFS in hdfs-site.xml - > dfs.data.dir property
  - e.g. \$HADOOP\_HOME/data/dfs/data/hadoop- \${user.name}/current
- If you want to see the files in local FS that store blocks of HDFS :
  - cd to the HDFS root dir specified in dfs.data.dir
  - go inside the sub-dir with name you got from fsck command
  - navigate into further sub-directories to find the block files
  - All this mapping is stored on the NameNode to map HDFS files to blocks (local FS files) on DataNodes

```
[root@centos-s-4vcpu-8gb-blr1-01 subdir0]# [root@centos-s-4vcpu-8gb-blr1-01 subdir0]# pwd /home/hadoop/hadoopdata/hdfs/datanode/current/BP-235233240-127.0.0.1-1618685260802/current/finalized/subdir0/subdir0 [root@centos-s-4vcpu-8gb-blr1-01 subdir0]# ls -l total 2712 -rw-r--r--. 1 root root 123637 Apr 18 01:18 blk_1073741825 -rw-r--r--. 1 root root 975 Apr 18 01:18 blk_1073741825_1001.meta -rw-r--r--. 1 root root 661 Apr 18 01:20 blk_1073741832 -rw-r--r--. 1 root root 15 Apr 18 01:20 blk_1073741832_1008.meta -rw-r--r--. 1 root root 352 Apr 18 01:20 blk_1073741833 -rw-r--r--. 1 root root 11 Apr 18 01:20 blk_1073741833_1009.meta -rw-r--r--. 1 root root 40183 Apr 18 01:20 blk_1073741834 -rw-r--r--. 1 root root 323 Apr 18 01:20 blk_1073741834_1010.meta -rw-r--r--. 1 root root 206080 Apr 18 01:20 blk_1073741835 -rw-r--r--. 1 root root 1619 Apr 18 01:20 blk_1073741835_1011.meta -rw-r--r--. 1 root root 661 Apr 18 12:22 blk_1073741842 -rw-r--r--. 1 root root 15 Apr 18 12:22 blk_1073741842_1018.meta -rw-r--r--. 1 root root 353 Apr 18 12:22 blk_1073741843 -rw-r--r--. 1 root root 11 Apr 18 12:22 blk_1073741843_1019.meta
```

# Replica Placement Strategy - with Rack awareness

- First replica is placed on the same node as the client
- Second replica is placed on a node that is present on different rack
- Third replica is placed on same rack as second but on a different node
- Putting each replica on a different rack is expensive write operation
- For replicas > 3, nodes are randomly picked for 4th replica without violating upper limit per rack as  $(\text{replicas}-1) / \text{racks} + 2$ .
- Total replicas  $\leq \# \text{DataNodes}$  with no 2 replicas on same DN
- Once the replica locations are set, pipeline is built
- Shows good reliability
- NameNode collects block report from DataNodes to balance the blocks across nodes and control over/under replication of blocks



# Why rack awareness ?

- **To improve the network performance:** The communication between nodes residing on different racks is directed via switch. In general, you will find *greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the **Rack Awareness helps you to have reduce write traffic in between different racks** and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.
- **To prevent loss of data:** We **don't have to worry about the data even if an entire rack fails** because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that ***never put all your eggs in the same basket.***

**Rack Awareness Algorithm**

Block A: Block B: Block C:



# Topics for today

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ Robustness
  - ✓ Blocks and replication strategy
  - ✓ **Read and write operations**
  - ✓ File formats
  - ✓ Commands

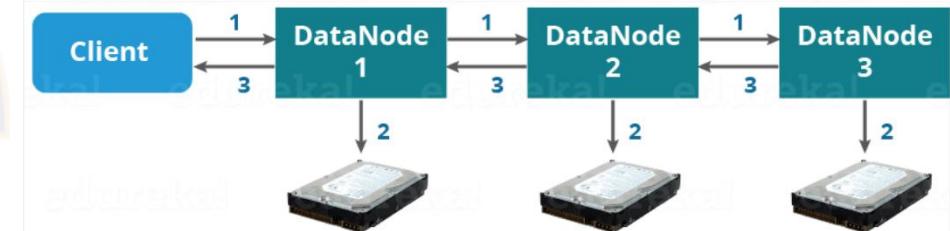


# HDFS data writes

Now, the following protocol will be followed whenever the data is written into

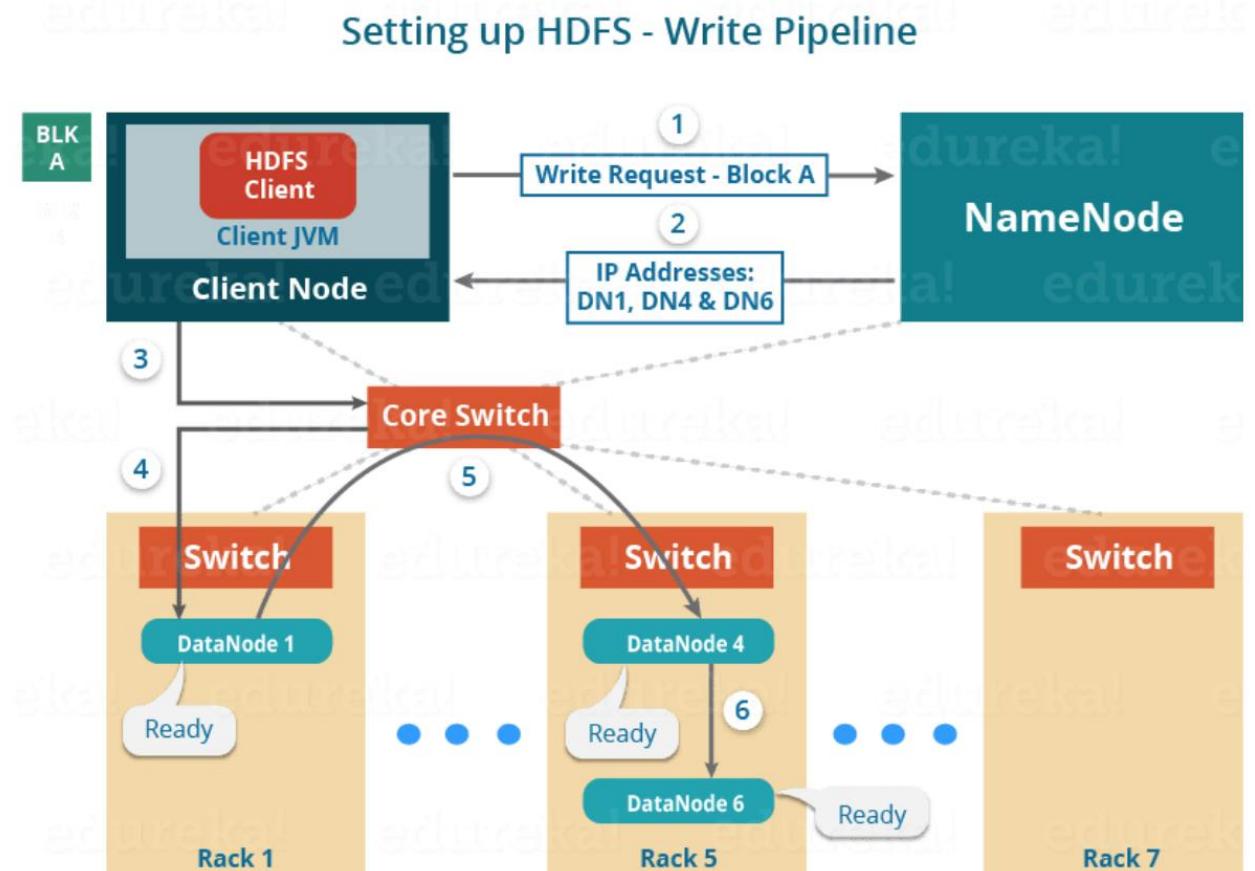
HDFS:

- HDFS client contacts NameNode for Write Request against the two blocks, say, Block A & Block B.
- NameNode grants permission to client with IP addresses of the DataNodes to copy blocks
- Selection of DataNodes is randomized but factoring in availability, RF, and rack awareness
- For 3 copies, 3 unique DNs needed, if possible, for each block.
  - For Block A, list A = {DN1, DN4, DN6}
  - For Block B, set B = {DN3, DN7, DN9}
- Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
- Now the whole data copy process will happen in three stages:
  1. Set up of Pipeline
  2. Data streaming and replication
  3. Shutdown of Pipeline (Acknowledgement stage)



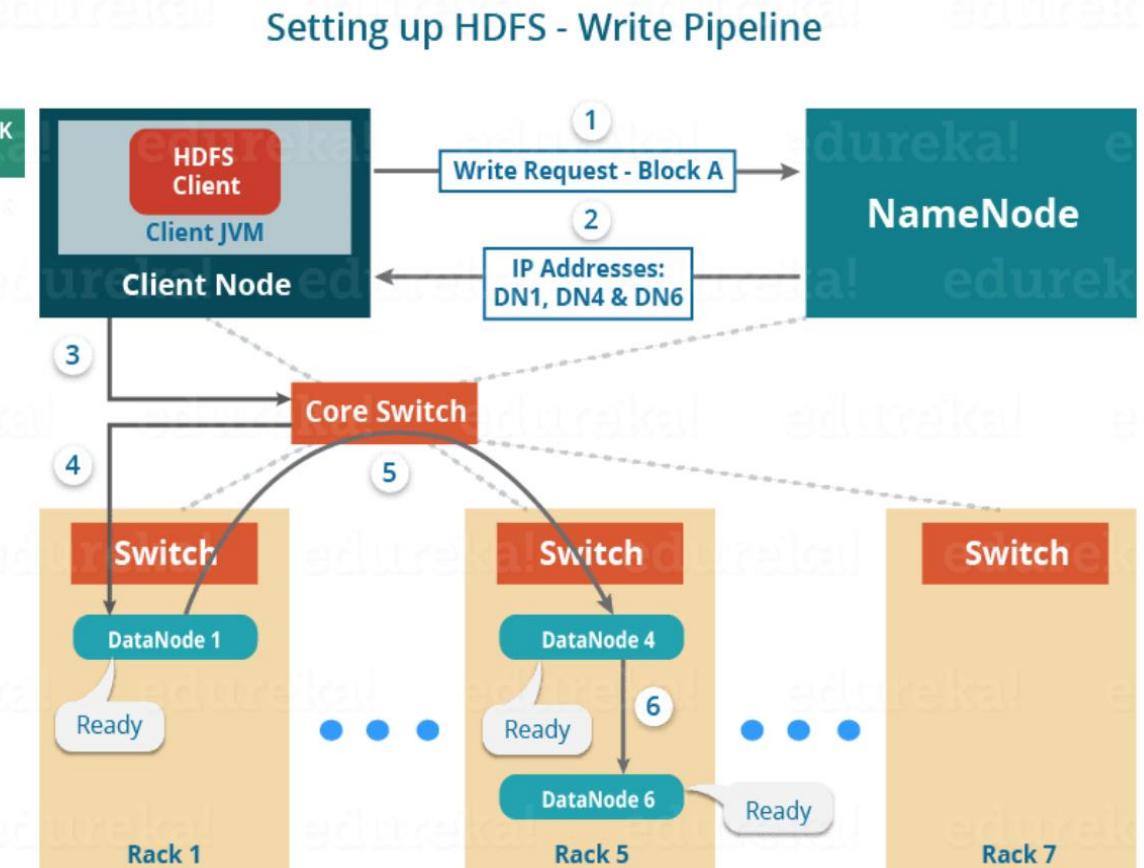
# HDFS Write: Step 1. Setup pipeline

Client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is DN1, DN4, DN6



# HDFS Write: Step 1. Setup pipeline for a block

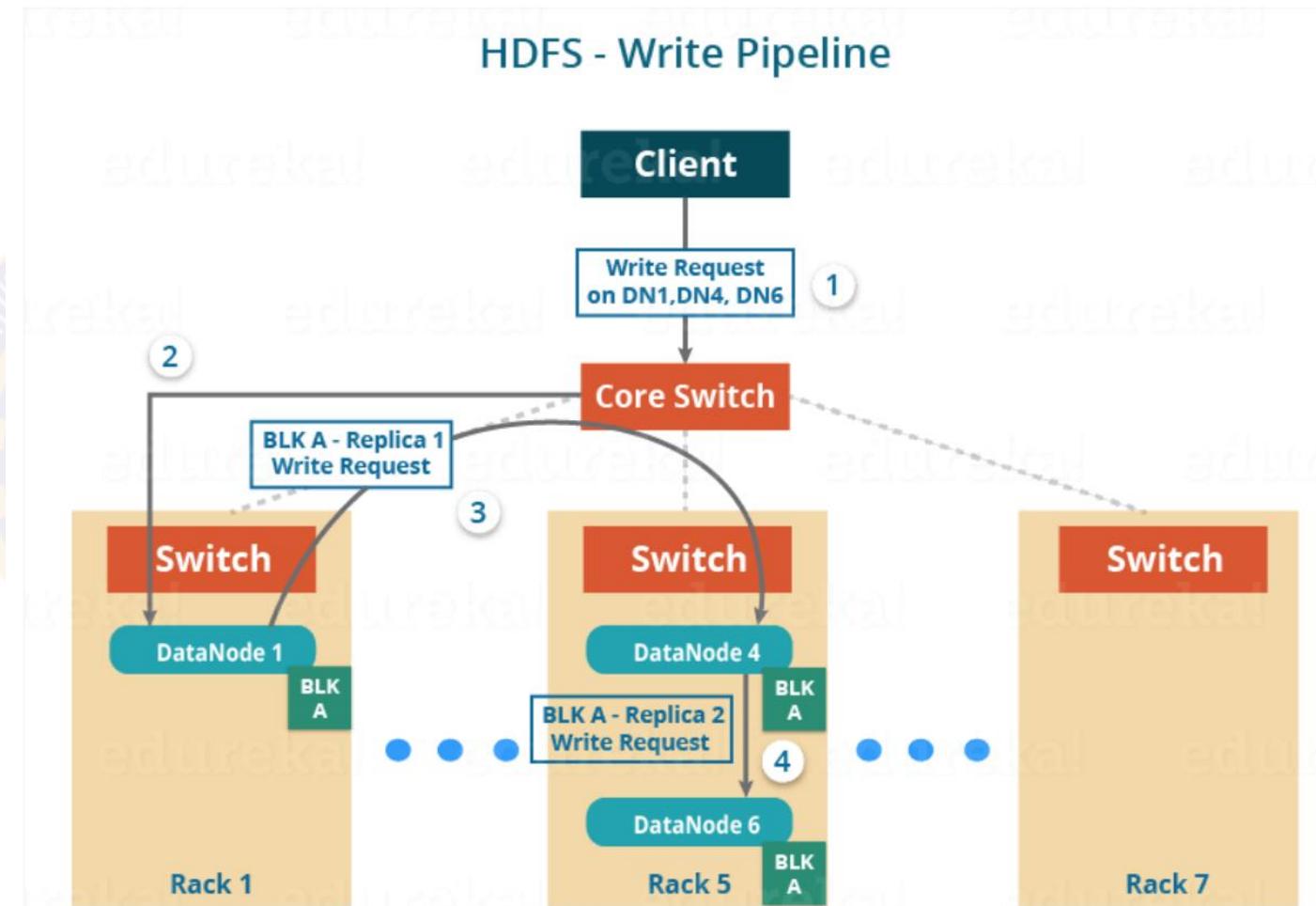
1. Client chooses the first DataNode (DN1) and will establish a TCP/IP connection.
2. Client informs DN1 to be ready to receive the block.
3. Provides IPs of next two DNs (4, 6) to DN1 for replication.
4. The DN1 connects to DN4 and informs it to be ready and gives IP of DN6. DN4 asks DN6 to be ready for data.
5. Ack of readiness follows the reverse sequence, i.e. from the DN6 to DN4 to DN1.
6. At last DN1 will inform the client that all the DNs are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
7. Now pipeline set up is complete and the client will finally begin the data copy or streaming process.



# HDFS Write: Step 2. Data streaming

Client pushes the data into the pipeline.

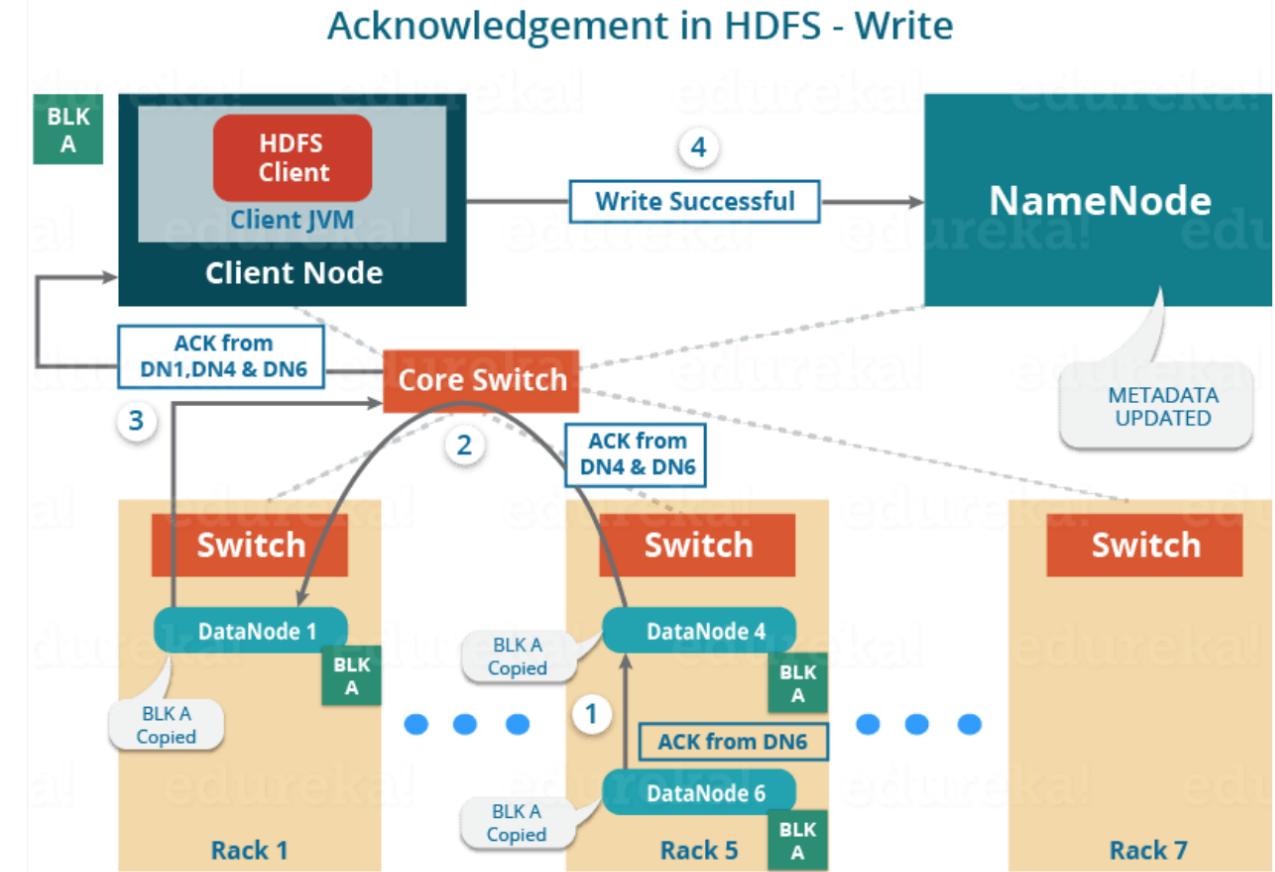
1. Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
2. Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
3. Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.



# HDFS Write: Step 3. Shutdown pipeline / ack

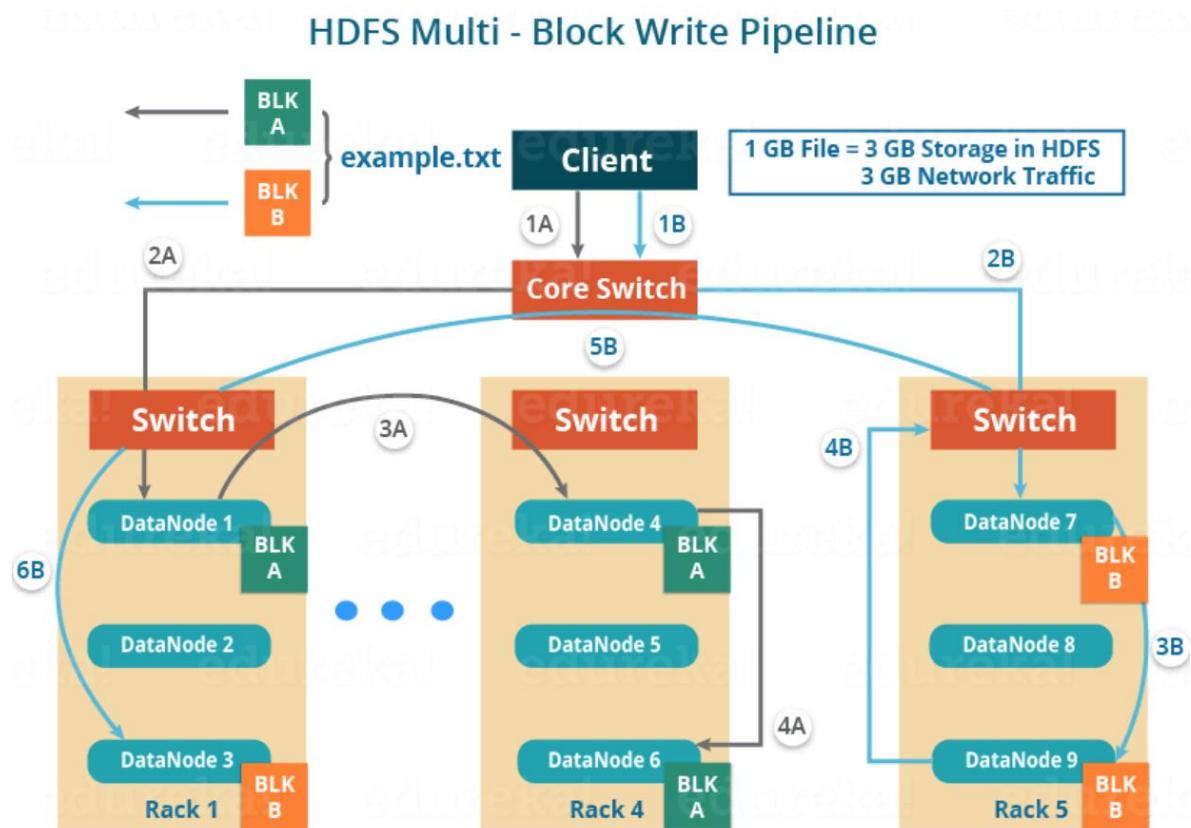
Block is now copied to all DNs. Client and NameNode need to be updated. Client needs to close pipeline and end TCP session.

1. Acknowledgement happens in the reverse sequence i.e. from DN 6 to 4 and then to 1.
2. DN1 pushes three acknowledgements (including its own) into pipeline and client.
3. Client informs NameNode that data has been written successfully.
4. NameNode updates metadata.
5. Client shuts down the pipeline.



# Multi-block writes

- The client will copy Block A and Block B to the first DataNode simultaneously.
- Parallel pipelines for each block
- Pipeline process for a block is same as discussed.
- E.g. 1A, 2A, 3A, ... and 1B, 2B, 3B, ... work in parallel

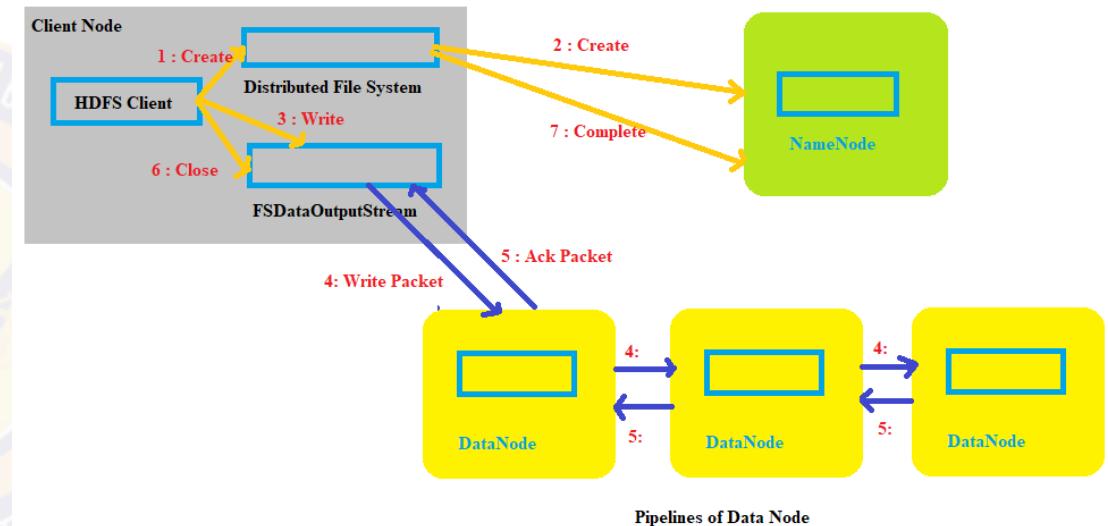


# Sample write code

```
public class WriteFileToHDFS{  
    public static void main(String[] args) throws IOException {  
        WriteFileToHDFS.writeFileToHDFS();  
    }  
  
    public static void writeFileToHDFS() throws IOException {  
        Configuration configuration = new Configuration();  
        configuration.set("fs.defaultFS", "hdfs://localhost:9000");  
        FileSystem fileSystem = FileSystem.get(configuration);  
        String fileName = "read_write_hdfs_example.txt";  
        Path hdfsWritePath = new Path("/javareadwriteexample/" + fileName);  
        FSDataOutputStream fsDataOutputStream = fileSystem.create(hdfsWritePath, true);  
        BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(fs  
        DataOutputStream, StandardCharsets.UTF_8));  
        bufferedWriter.write("Java API to write data in HDFS");  
        bufferedWriter.newLine();  
        bufferedWriter.close();  
        fileSystem.close();  
    }  
}
```

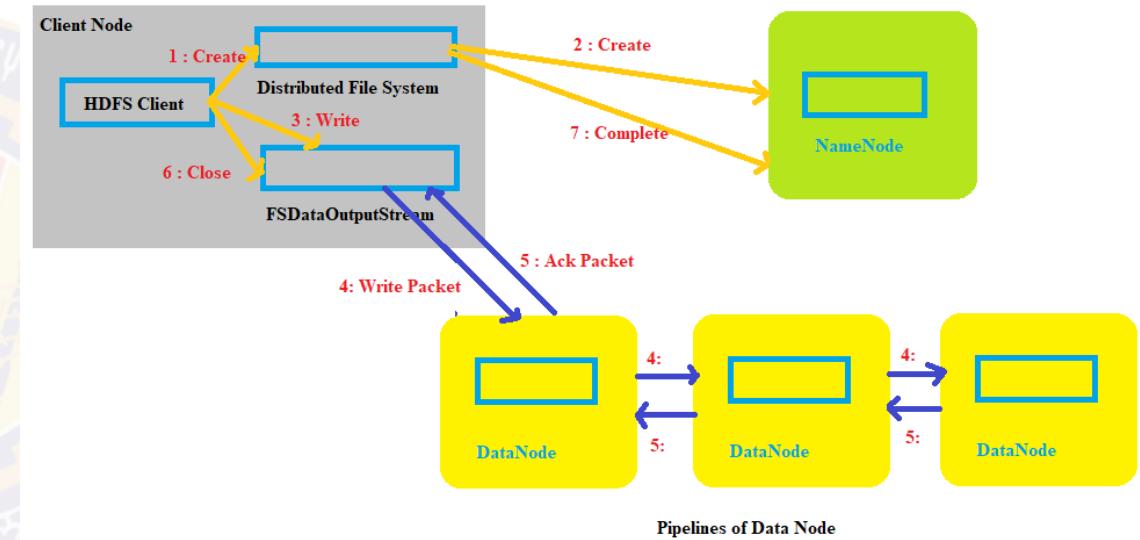
# HDFS Create / Write - Call sequence in code

- 1) Client calls create() on FileSystem to create a file
  - 1) RPC call to NameNode happens through FileSystem to create new file.
  - 2) NameNode performs checks to create a new file. Initially, NameNode creates a file without associating any data blocks to the file.
  - 3) The FileSystem.create() returns an FSDataOutputStream to client to perform write.
- 2) Client creates a BufferedWriter using FSDataOutputStream to write data to a pipeline
  - 1) Data is split into packets by FSDataOutputStream, which is then written to the internal queue.
  - 2) DataStreamer consumes the data queue
  - 3) DataStreamer requests NameNode to allocate new blocks by selecting a list of suitable DataNodes to store replicas. This is pipeline.
  - 4) DataStreamer streams packets to first DataNode in the pipeline.



# HDFS Create / Write - Call sequence in code

- 3) The first DataNode stores packet and forwards it to Second DataNode and then Second node transfer it to Third DataNode.
- 4) FSDataOutputStream also manages a “Ack queue” of packets that are waiting for the acknowledgement by DataNodes.
- 5) A packet is removed from the queue only if it is acknowledged by all the DataNodes in the pipeline
- 6) When the client finishes writing to the file, it calls close() on the stream
- 7) This flushes all the remaining packets to DataNode pipeline and waits for relevant acknowledgements before communicating the NameNode to inform the client that the writing of the file is complete.



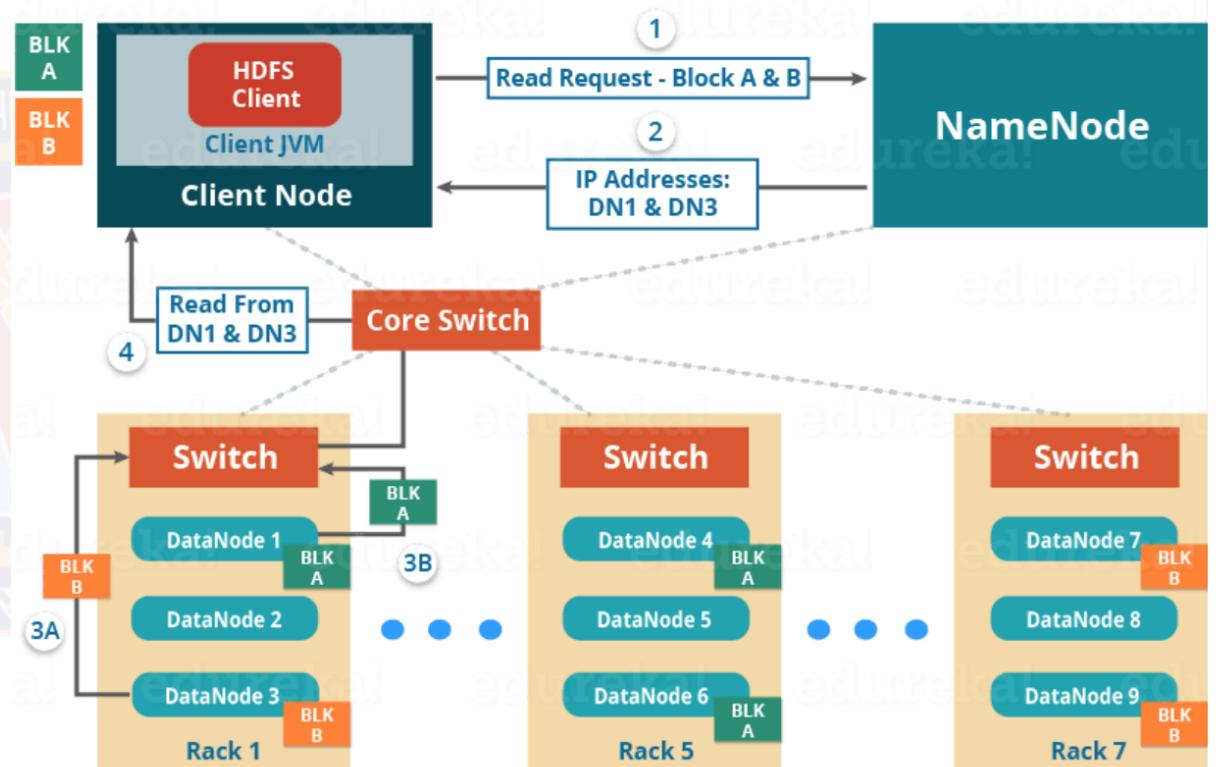
# HDFS Read (Sec1)

1. Client contacts NameNode asking for the block metadata for a file
2. NameNode returns list of DNs where each block is stored
3. Client connects to the DNs where blocks are stored
4. The client starts reading data parallel from the DNs (e.g. Block A from DN1, Block B from DN3)
5. Once the client gets all the required file blocks, it will combine these blocks to form a file.

How are blocks chosen by NameNode ?

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.

HDFS - Read Architecture

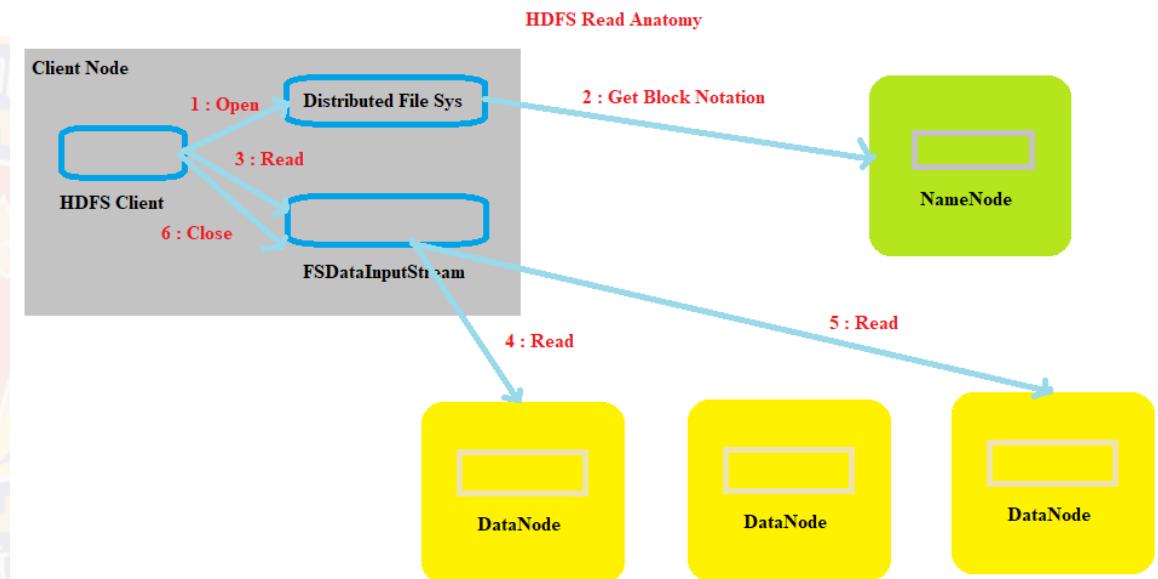


# Sample read code

```
public class ReadFileFromHDFS{  
    public static void main(String[] args) throws IOException {  
        ReadFileFromHDFS.readFileFromHDFS();  
    }  
    public static void readFileFromHDFS() throws IOException {  
        Configuration configuration = new Configuration();  
        configuration.set("fs.defaultFS", "hdfs://localhost:9000");  
        FileSystem fileSystem = FileSystem.get(configuration);  
        String fileName = "read_write_hdfs_example.txt";  
        Path hdfsReadPath = new Path("/javareadwriteexample/" + fileName);  
        FSDataInputStream inputStream = fileSystem.open(hdfsReadPath);  
        //Classical input stream usage  
        String out= IOUtils.toString(inputStream, "UTF-8");  
        System.out.println(out);  
        inputStream.close();  
        fileSystem.close();  
    }  
}
```

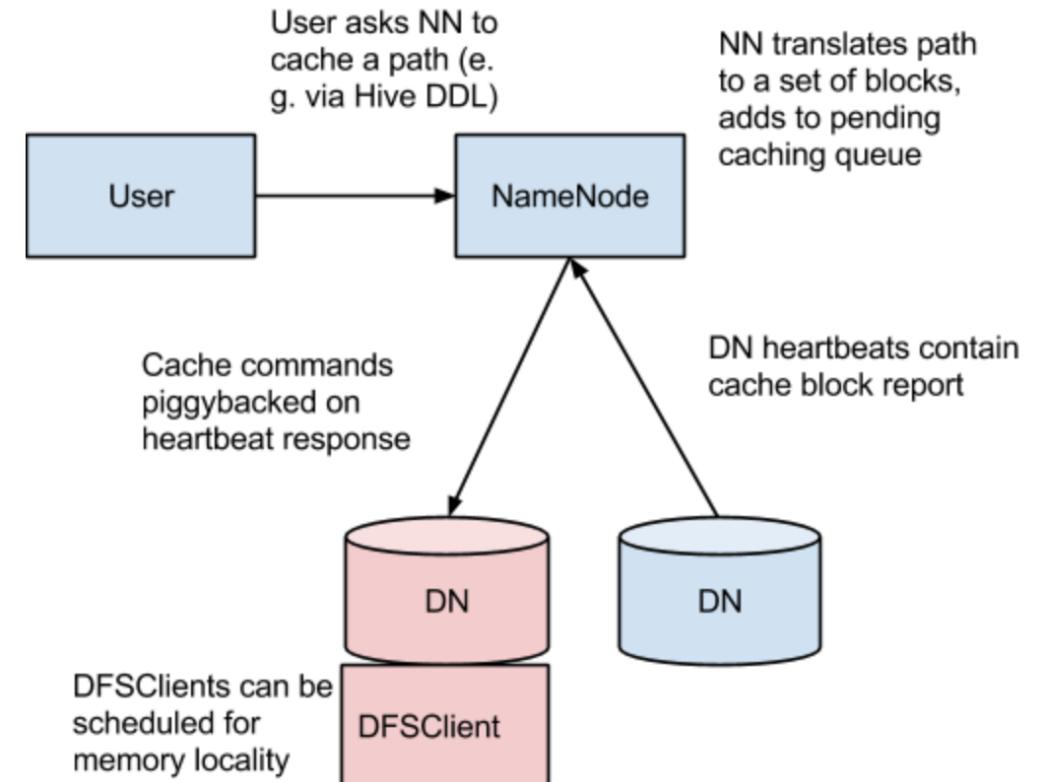
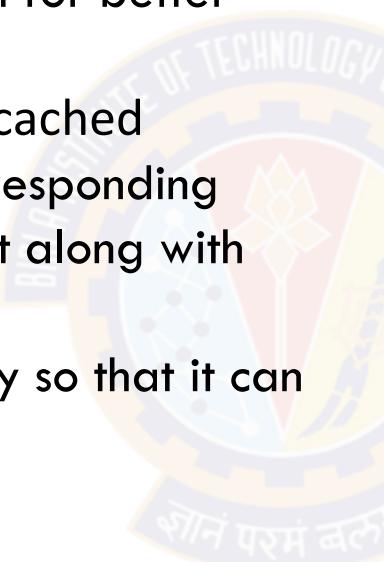
# HDFS Read - Call sequence in code

- 1) Client opens the file that it wishes to read from by calling open() on FileSystem
  - 1) FileSystem communicates with NameNode to get location of data blocks.
  - 2) NameNode returns the addresses of DataNodes on which blocks are stored.
  - 3) FileSystem returns FSDataInputStream to client to read from file.
- 2) Client then calls read() on the stream, which has addresses of DataNodes for first few blocks of file, connects to the closest DataNode for the first block in file
  - 1) Client calls read() repeatedly to stream the data from DataNode
  - 2) When end of block is reached, stream closes the connection with DataNode.
  - 3) The stream repeats the steps to find the best DataNode for the next blocks.
- 3) When the client completes the reading of file, it calls close() on the stream to close the connection.



# Read optimizations

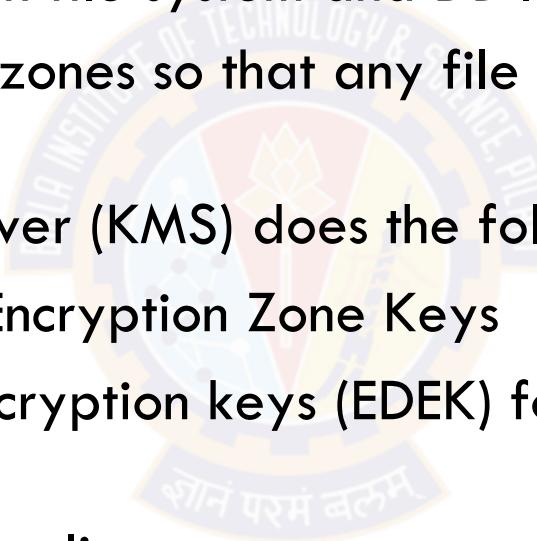
- Short-circuit reads can also be made by client to local file bypassing DataNode using /dev/shm for better performance.
- Client can ask certain HDFS paths to be cached
  - ✓ NameNode asks DNs to cache corresponding blocks and send cache block report along with heartbeat
- Clients can be scheduled for data locality so that it can use local reads or caches better



# Security

- POSIX style file permissions
- Choice of transparent encryption/decryption
  - ✓ HDFS encryption is between file system and DB level encryption
- Create hierarchical encryption zones so that any file within the zone (a path) has the same key
- Hadoop Key Management Server (KMS) does the following
  - ✓ Provides access to stored Encryption Zone Keys
  - ✓ Create encrypted data encryption keys (EDEK) for storage by NameNode
  - ✓ Decrypting EDEK for use by clients

Database  
HDFS  
Native FS



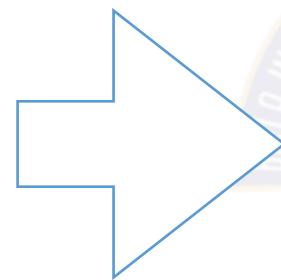
# Topics for Session 5

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ Robustness
  - ✓ Blocks and replication strategy
  - ✓ Read and write operations
  - ✓ **File formats**
  - ✓ Commands

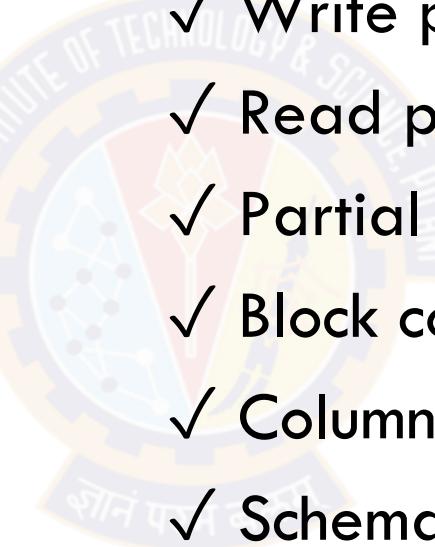


# File formats

- Text (JSON, CSV, ..)
- Sequence
- AVRO
- Parquet
- RC
- ORC

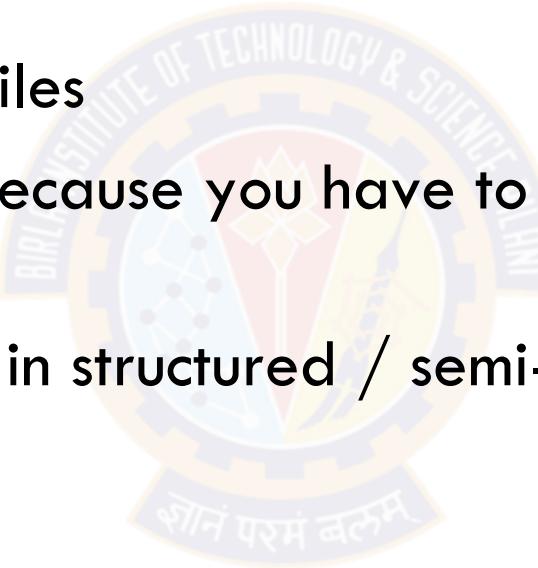


- Many ways to evaluate formats
  - ✓ Write performance
  - ✓ Read performance
  - ✓ Partial read performance
  - ✓ Block compression
  - ✓ Columnar support
  - ✓ Schema change



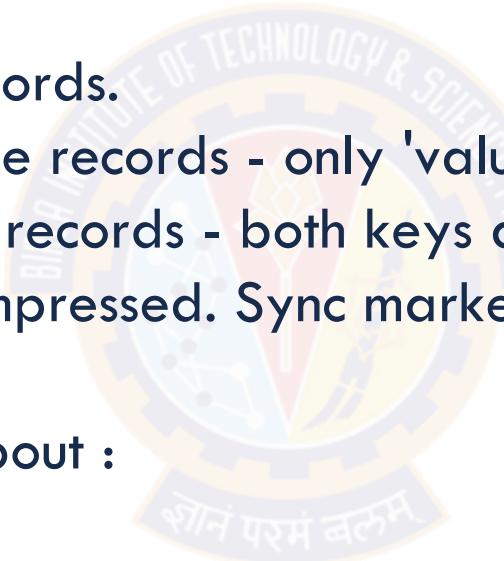
# File formats - text based

- Text-based (JSON, CSV ...)
  - ✓ Easily splittable
  - ✓ Can't split compressed files
    - so large Map tasks because you have to give the entire file to a Map task
  - ✓ Simplest to start putting in structured / semi-structured data



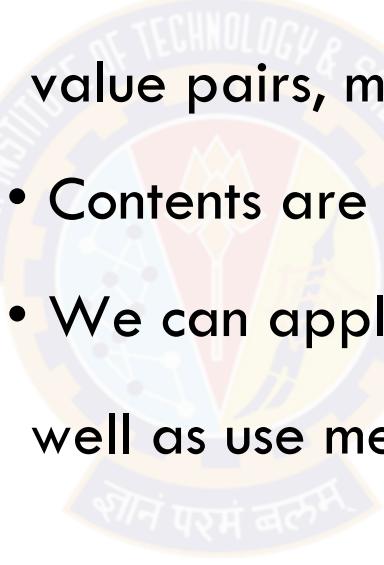
# File formats - sequence files

- A flat file consisting of binary key/value pairs
- Extensively used in MapReduce as input/output formats as well as internal temporary outputs of maps
- 3 types
  1. Uncompressed key/value records.
  2. Record compressed key/value records - only 'values' are compressed here.
  3. Block compressed key/value records - both keys and values are collected in configurable 'blocks' and compressed. Sync markers added for random access and splitting.
- Header includes information about :
  - key, value class
  - whether compression is enabled and whether at block level
  - compressor codec used



# Sequence File Writer in HDFS

```
public class SequenceFileWriter {  
  
    private static final String[] text = { "aa", "ccc", "eee", "fff" };  
  
    public static void main(String[] args) {  
  
        String uri = "hdfs://localhost:9000/user/seqdemo";  
  
        Configuration conf = new Configuration();  
  
        SequenceFile.Writer writer = null;  
  
        try {  
  
            FileSystem fs = FileSystem.get(URI.create(uri), conf);  
  
            Path path = new Path(uri);  
  
            IntWritable key = new IntWritable();  
  
            Text value = new Text();  
  
            writer = SequenceFile.createWriter(fs, conf, path,  
                key.getClass(), value.getClass());  
  
            for (int i = 0; i < 100; i++) {  
  
                key.set(100 - i);  
  
                value.set(text[i % text.length]);  
  
                writer.append(key, value);  
  
            }  
  
        } catch (IOException e) {  
  
            e.printStackTrace();  
  
        } finally {  
  
            IOUtils.closeStream(writer);  
  
        }  
    }  
}
```



- When we want to store binary(images)/text as key-value pairs, maybe attach some meta-data as well
- Contents are stored as binary format, e.g. sequence file
- We can apply some image processing on the files, as well as use meta-data to retrieve specific images

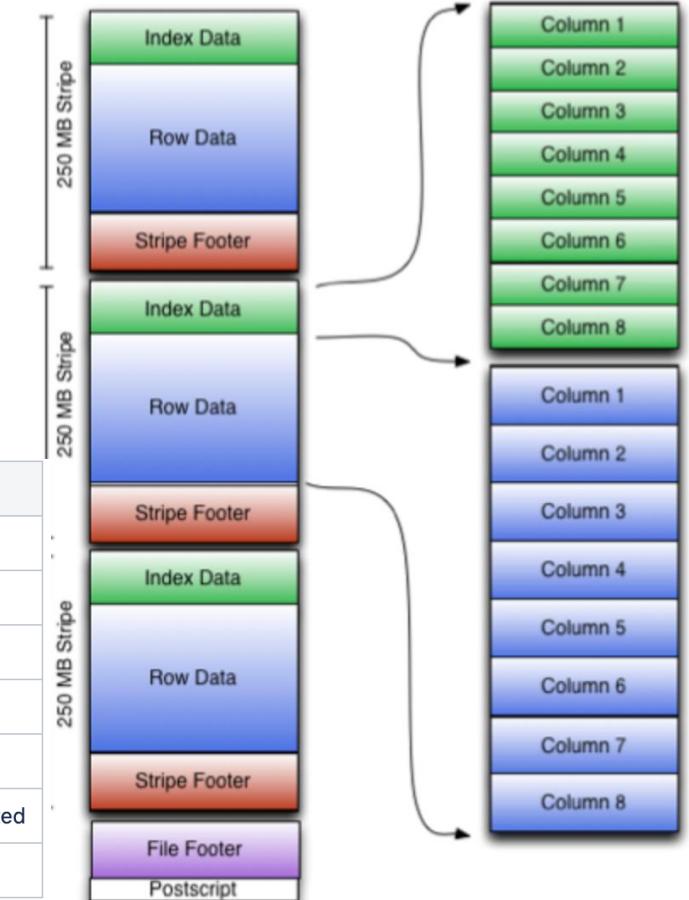
# File formats - Optimized Row Columnar (ORC) \*

Improves performance when Hive is reading, writing, and processing data

```
create table Addresses (
    name string,
    street string,
    city string,
    state string,
    zip int
) stored as orc tblprop
```

Key	Default	Notes
orc.compress	ZLIB	high level compression (one of NONE, ZLIB, SNAPPY)
orc.compress.size	262,144	number of bytes in each compression chunk
orc.stripe.size	67,108,864	number of bytes in each stripe
orc.row.index.stride	10,000	number of rows between index entries (must be >= 1000)
orc.create.index	true	whether to create row indexes
orc.bloom.filter.columns	""	comma separated list of column names for which bloom filter should be created
orc.bloom.filter.fpp	0.05	false positive probability for bloom filter (must >0.0 and <1.0)

Index data used to skip rows  
Includes min/max for each column and their row positions  
Row data used for table scans



\* more in Hive session

ref: <https://cwiki.apache.org/confluence/display/hive/languagemanual+orc>

# File formats - Parquet

- Columnar format
- Pros
  - ✓ Good for compression because data in a column tends to be similar
    - Support block level compression as well as file level
  - ✓ Good query performance when query is for specific columns
  - ✓ Compared to ORC - more flexible to add columns
  - ✓ Good for Hive and Spark workloads if working on specific columns at a time
- Cons
  - ✓ Expensive write due to columnar
    - So if use case is more about reading entire rows, then not a good choice
    - Anyway - Hadoop systems are more about write once and read many times - so read performance is paramount
  - Note: Avro is another option for workloads with full row scans because it is row major storage

# Topics for today

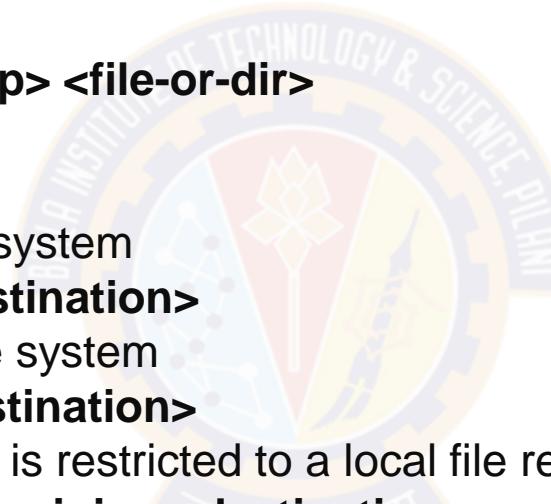
- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ Robustness
  - ✓ Blocks and replication strategy
  - ✓ Read and write operations
  - ✓ File formats
  - ✓ Commands



<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html>

# Basic command reference

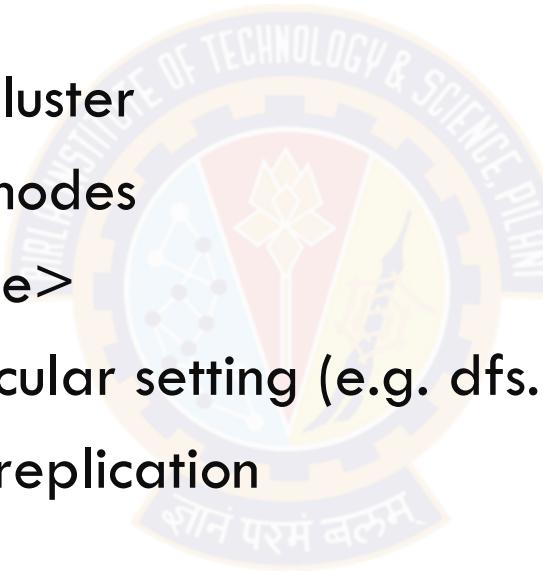
- list files in the path of the file system
  - ✓ **hadoop fs -ls <path>**
- alters the permissions of a file where <arg> is the binary argument e.g. 777
  - ✓ **hadoop fs -chmod <arg> <file-or-dir>**
- change the owner of a file
  - ✓ **hadoop fs -chown <owner>:<group> <file-or-dir>**
- make a directory on the file system
  - ✓ **hadoop fs -mkdir <path>**
- copy a file from the local storage onto file system
  - ✓ **hadoop fs -put <local-origin> <destination>**
- copy a file to the local storage from the file system
  - ✓ **hadoop fs -get <origin> <local-destination>**
- similar to the put command but the source is restricted to a local file reference
  - ✓ **hadoop fs -copyFromLocal <local-origin> <destination>**
- similar to the get command but the destination is restricted to a local file reference
  - ✓ **hadoop fs -copyToLocal <origin> <local-destination>**
- create an empty file on the file system
  - ✓ **hadoop fs -touchz**
- copy files to stdout
  - ✓ **hadoop fs -cat <file>**



# More HDFS commands - config and usage

✓ Get configuration data in general, about name nodes, about any specific attribute

- `hdfs getconf` return various configuration settings in effect
- `hdfs getconf -namenodes`
  - returns namenodes in the cluster
- `hdfs getconf -secondarynamenodes`
- `hdfs getconf -confkey <a.value>`
  - return the value of a particular setting (e.g. `dfs.replication`)
  - `hdfs getconf -confkey dfs.replication`
- `hdfs dfsadmin -report`
  - find out how much disk space us used, free, under-replicated, etc.
- `hadoop fs -setrep 2 /jps/wc/sample01.txt`
  - Set replication factor of 2 to sample01.txt file in HDFS



# Summary

- High level architecture of Hadoop 2 and differences with earlier Hadoop 1
- HDFS
  - ✓ Architecture
  - ✓ Read and write flows
  - ✓ File formats
  - ✓ Commands commonly used
  - ✓ Additional reading about HDFS:  
<https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>





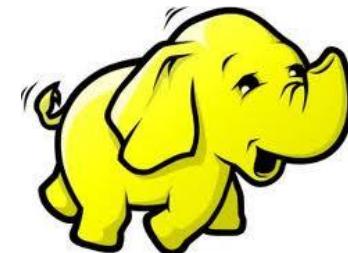
# Next Session: Distributed Programming

# Introduction to Hadoop

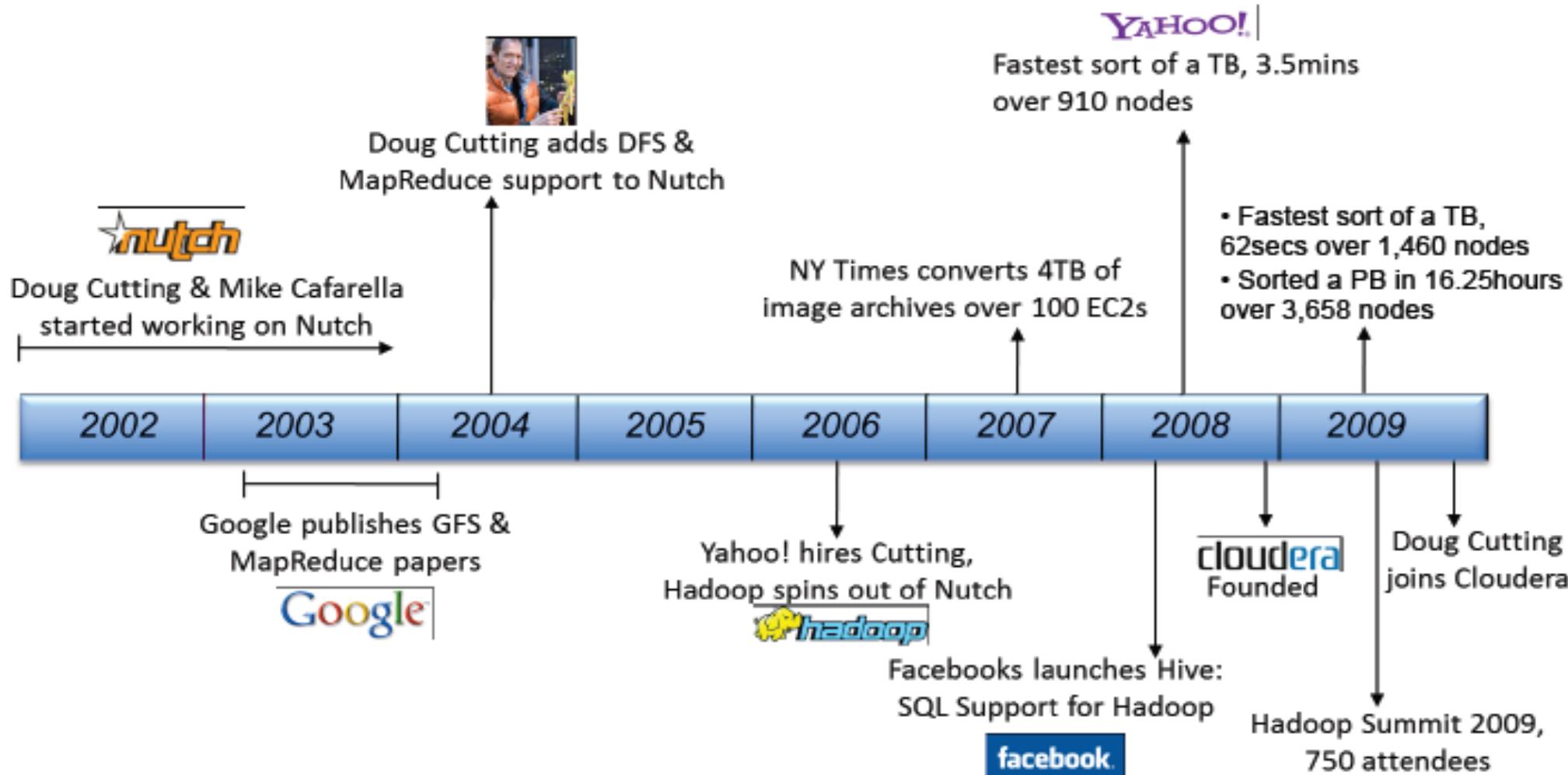
Janardhanan PS

# The Hadoop Project

- Originally based on papers published by Google in 2003 and 2004
  1. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. Appeared in 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.
  2. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Appeared in OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.
- Hadoop started in 2006 at Yahoo!
- Top level Apache Foundation project
- Large, active user base, user groups
- Very active development, strong development team



# History of Hadoop



# What is Hadoop?

- Hadoop is a complete, open-source ecosystem for capturing, organizing, storing, searching, sharing, analyzing and otherwise processing disparate data sources :
  - Structured
  - Semi-structured
  - Unstructured
- High Scalability
- High Fault-tolerance
- Impressive price/performance ratio :
  - Hadoop uses commodity hardware which results in a remarkably low cost.

# Principles of Hadoop

1. *All roads lead to scale-out*, scale-up architectures are rarely used and scale-out is the standard in big data processing.
2. *Share nothing*: communication and dependencies are bottlenecks, individual components should be as independent as possible to allow to proceed regardless of whether others fail.
3. *Expect failure*: components will fail at inconvenient times.
4. *Smart software, dumb hardware*: push smarts in the software, responsible for allocating generic hardware.
5. *Move processing, not data*: perform processing locally on data. What gets moved through the network are program binaries and status reports, which are dwarfed in size by the actual data set.
6. *Build applications, not infrastructure*. Instead of placing focus on data movement and processing, work on job scheduling, error handling, and coordination.

# Hadoop for Data Lakes

- Hadoop is changing the data warehousing paradigm.
  - Data warehouse is expensive but gives quick access to specific data records (DBMS based)
  - Data lakes are low-cost implementations which gives slow access to specific data records
  - Data lakes are ideal for applications in which the entire data set is to be accessed in every cycle of processing - (Hadoop based implementation is easy)
- [Read the Blog - What are Data Lakes ? - DataScienceCentral.com](#)

# Hadoop Components

Storage

HDFS

Self-healing  
high-bandwidth  
clustered storage

Processing

YARN

Framework for  
Fault-tolerant  
Distributed processing

# HDFS – Hadoop Distributed File System



# HDFS Daemons

## NameNode

- → The namenode stores the HDFS filesystem information in a file named fsimage.
  - updates to the filesystem are not updating the fsimage but instead are logging in to a file, so the I/O is fast
  - Client applications talk to the NameNode whenever they want to add/copy/move/delete a file.
- → When restarting, reads the fsimage and then applies log file to bring the filesystem state up to date in memory.

## Secondary Namenode

→ periodically read the filesystem changes log and apply them into the fsimage file.

# HDFS Daemons

## Datanode

- A DataNode stores data in the HDFS
- The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.
  - Client talks directly to datanode once namenode provides location of the data
  - Tasktracker installed in the same node where datanode is installed, so that operations are performed close to the data.
  - DataNode instances can talk to each other, which is what they do when they are replicating data

### Adding Datanode as online

- Add new node's IP inside 'worker' file of namenode
- start datanode of new slave node
  - `hdfs --daemon start datanode`
- Refresh the nodes
  - `hdfs dfsadmin -refreshNodes`

# Features of HDFS

- HDFS is derived from GFS (Google File System).
- HDFS forms the ‘Infrastructural’ part of Hadoop
- HDFS is a distributed file system with single root (/)
- It is designed to run on commodity hardware.
- It is a low-cost scalable data platform ideal for machine learning projects
- HDFS is fault tolerant and scalable
- It can magically scale from one-node cluster to thousand-nodes cluster .  
*(Yahoo! has 4,500-node cluster managing 40 petabytes of enterprise data).*

# Thank You



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Big Data Systems

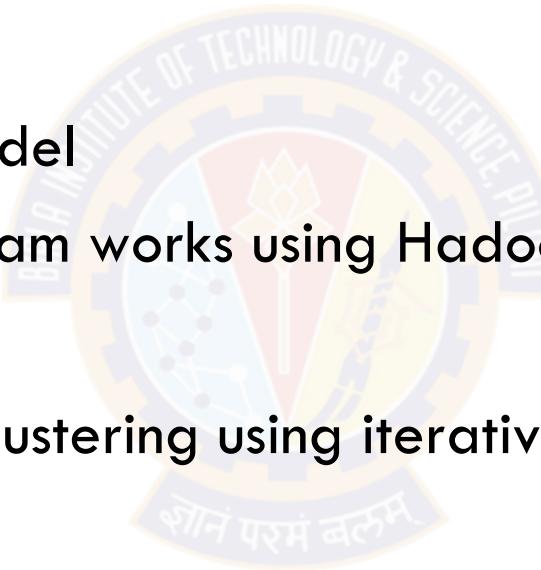
## Session 6 - Distributed Programming

---

Janardhanan PS  
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **Top down design**
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce
- Hands on demo of K-Means clustering using iterative MapReduce



# Top down design - sequential context

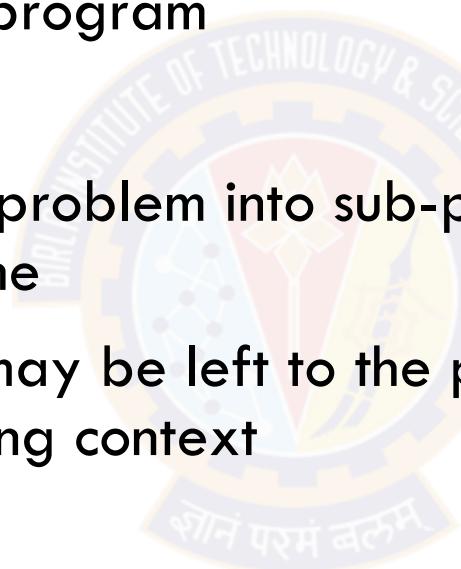
- In the context of a sequential program
  - Divide and conquer
    - It is easier to divide a problem into sub-problems and execute one by one
    - A sub-problem definition may be left to the programmer in a sequential programming context

main()

f1() → f2() → f5()

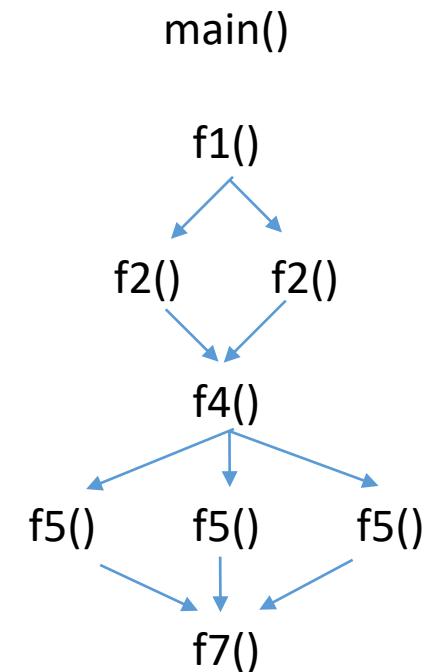
f3()

f4()



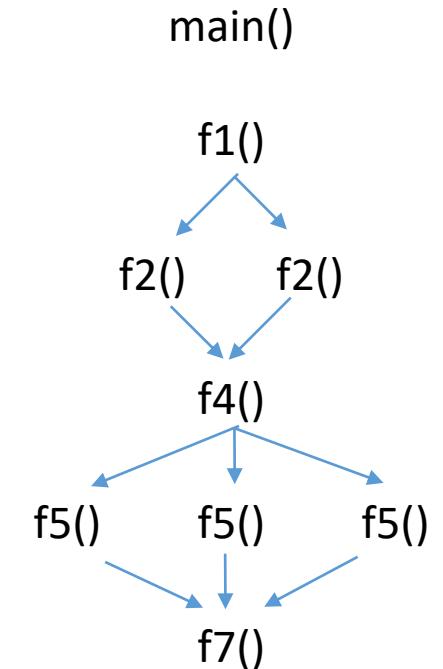
# Top down design - parallel context

- In the context of a parallel program, we cannot decompose the problem into sub-problems in anyway the programmer chooses to
- Need to think about
  - Each sub-problem needs to be assigned to a processor
    - Goal is to get the program work faster
  - Divide the problem only when we can combine at the end into the final answer
    - Need to decide where to do the combination
    - Is there any parallelism in combination or is it sequential or trivial



# Deciding on number of sub-problems

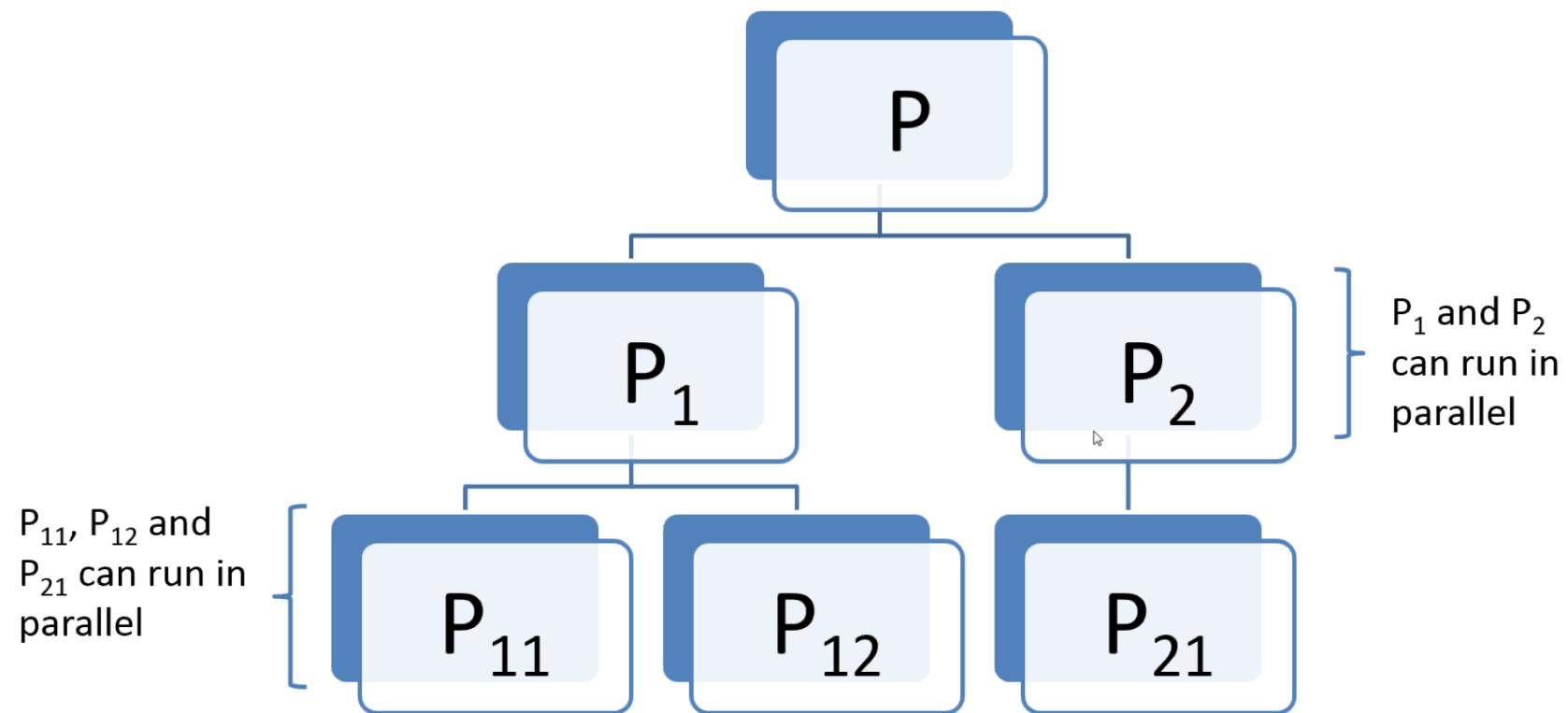
- In conventional top down design for sequential systems
  - Keep number of sub-problems manageable
  - Because need to keep track of them as computation progresses
- In parallel system it is dictated by number of processors
  - Processor utilisation is the key
  - If there are  $N$  processors, we can potentially have  $N$  sub-problems



How many processors ?

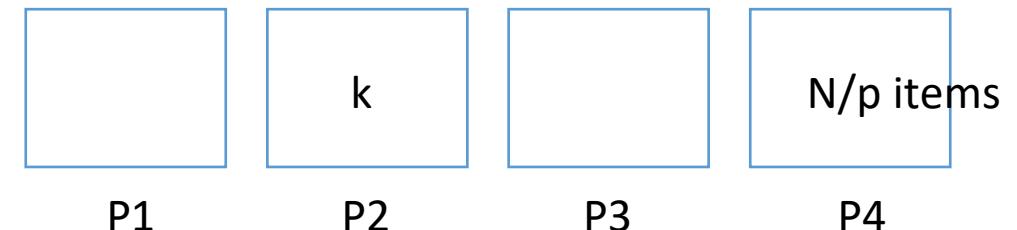
# Top-down design

- At each level problems need to run in parallel



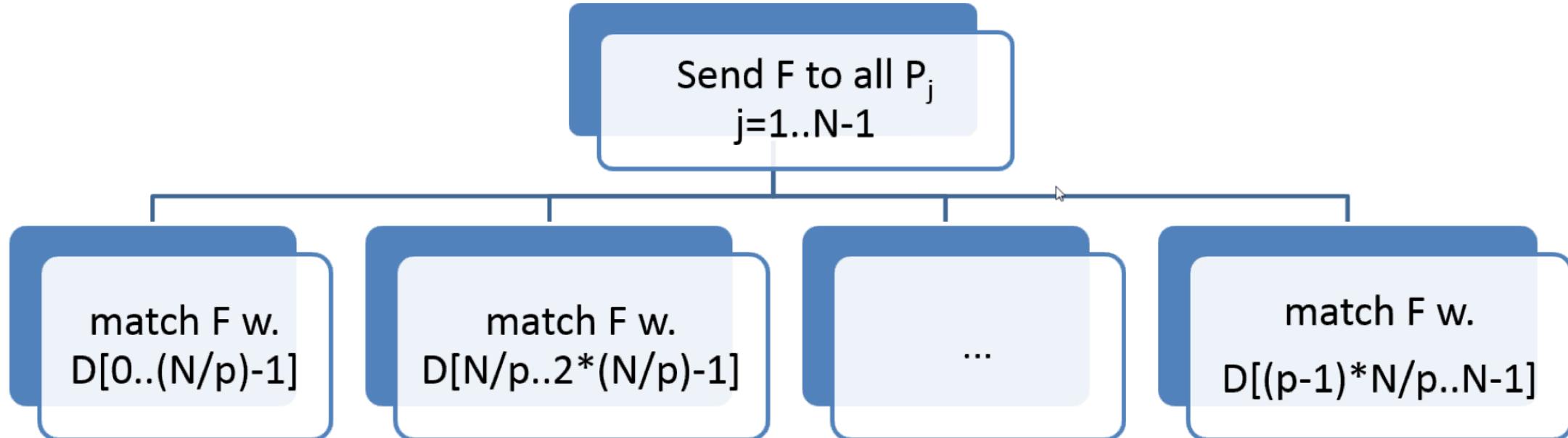
# Example 1 - Keyword search in list

- Problem:
  - Search for a key  $k$  in a sorted list  $L_s$  of size  $N$
- Data:
  - $L_s$  is stored in a distributed system with  $p$  processors each storing  $N/p$  items
- Solution:
  - Run binary search in each of the  $p$  processors in parallel
  - Whichever processor finds  $k$  return  $(i, j)$  where  $i$ th processor has found key in  $j$ th position
  - Combination: One or more positions are collected at processor 0
- Speedup:  $p$
- Time complexity:  $O(\log(N/p))$



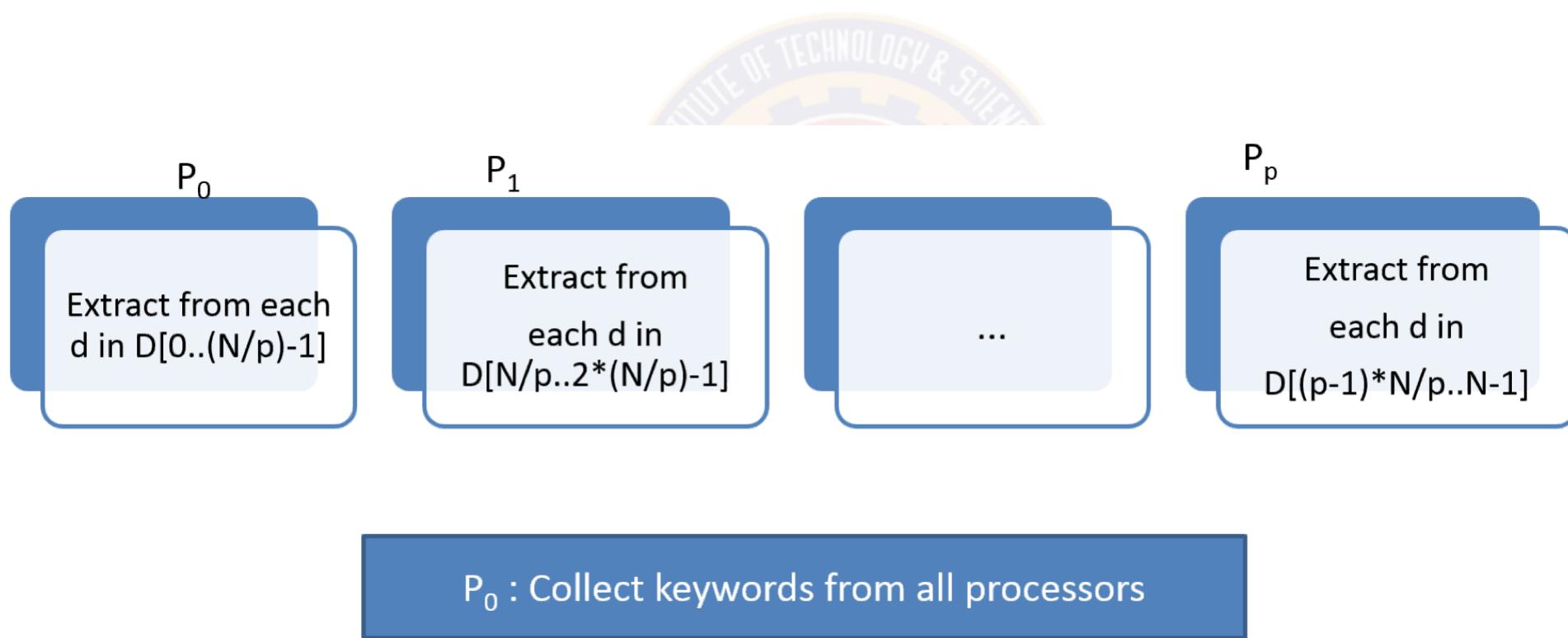
## Example 2 - Fingerprint matching

- Find matches for a fingerprint  $F$  in a database of  $D$  prints
- Set of  $D$  prints is partitioned and evenly stored in a distributed database
- Partitioning is an infrequent activity - only when many new entries in database
- Search is the frequent activity
- Speed up p
- Time complexity  $O(N/p)$  given sequential search in every partition



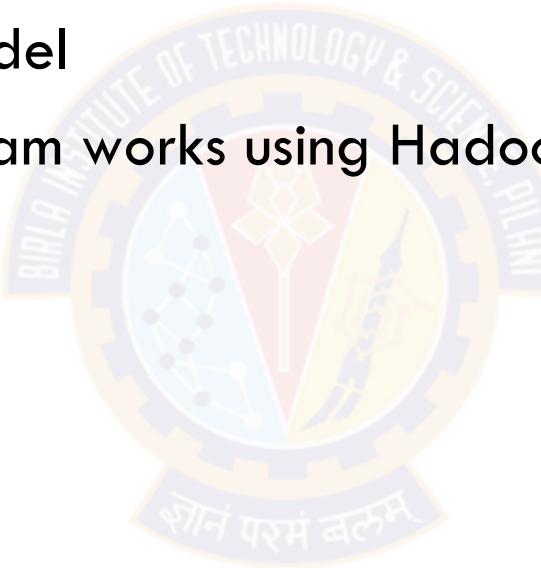
## Example 3: Document search

- Find keywords from each document  $d$  in a distributed document collection  $D$



# Topics for today

- Top down design
- **Types of parallelism**
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce



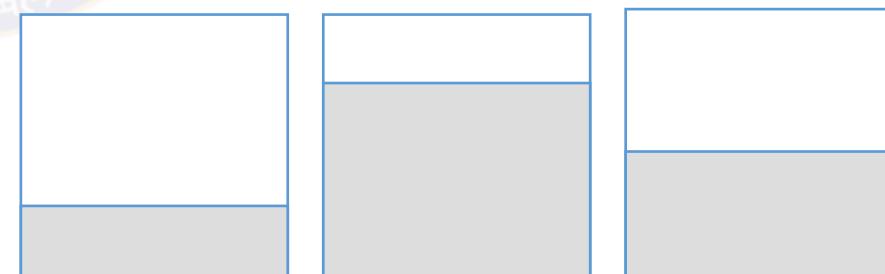
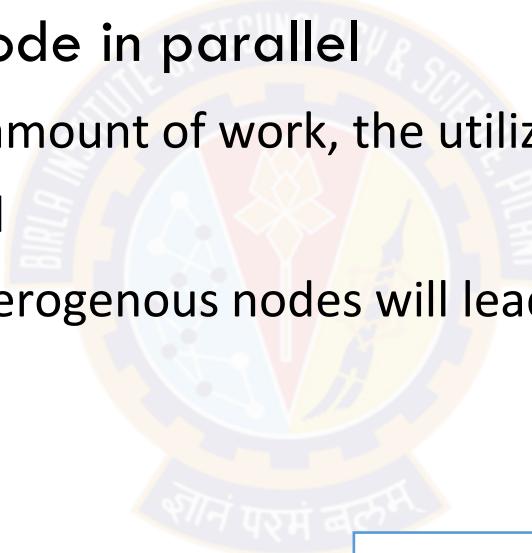
# Types of Parallelism

- Data Parallelism
- Tree Parallelism
- Task Parallelism
- Request Parallelism



# Data parallel execution model

- Data is partitioned to multiple nodes / processors
  - Try to make partitions equal or balanced
- All processors execute the same code in parallel
  - For homogenous nodes and equal amount of work, the utilization will be close to 100%
  - Execution time overhead is minimal
  - Unbalanced data size / work or heterogenous nodes will lead to higher execution time



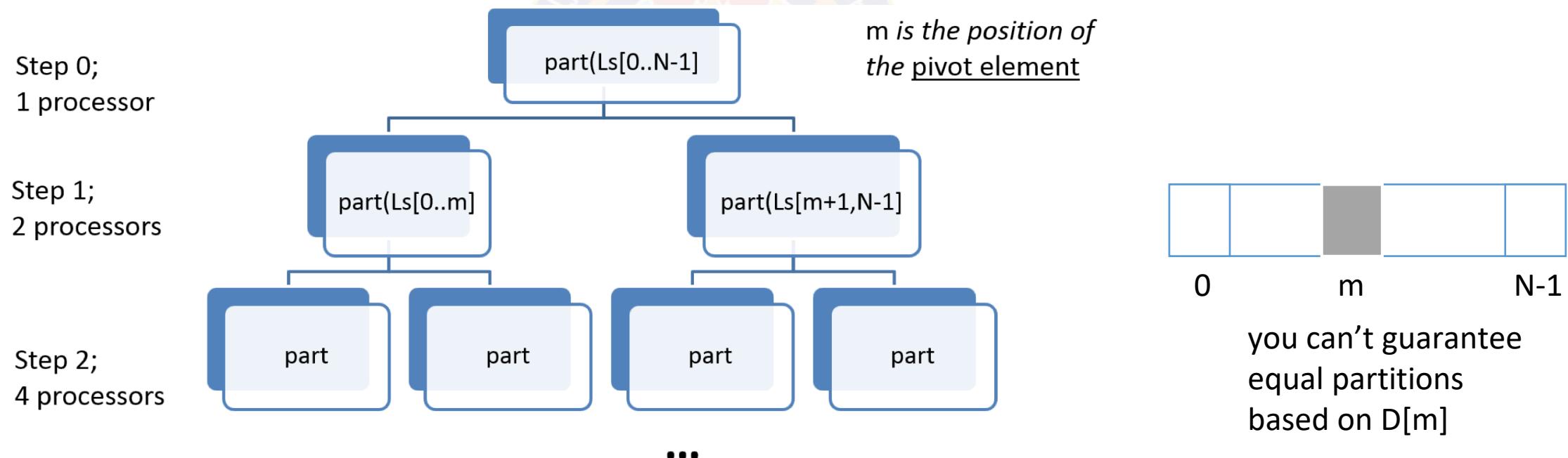
# Where data parallelism is not possible

- There are problems where you cannot divide the work
  1. equally
  2. independently to proceed in parallel
- QuickSort( $L_s, N$ )
  - All  $N$  items in  $L_s$  have to be in memory of processor 0
  - Time Complexity
    - Best case -  $O(n \log(n))$
    - Worst case –  $O(n^2)$



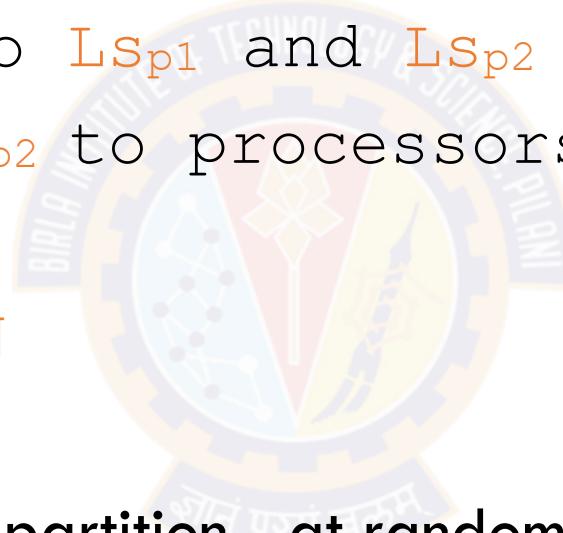
# Example : QuickSort

- Pick a pivot element at position  $m$  to partition  $L_s$  and partition into sub-problems
- Do that in each level
- There is dependency on parent level to partition - not a single level problem ( $\log N$  or  $\log p$  levels which ever is lower)
- Choice of  $m$  cannot guarantee equal partition
  - At a level one set of processors can get large partitions and another set small partitions
  - There could be techniques to maintain balanced partitions and improve processor utilization uniformly



# Tree parallel execution model for Quicksort parallel logic

- In step  $j$ 
  - foreach processor  $p$  from 1 to  $2^{j-1}$  do
    - partition  $L_{Sp}$  into  $L_{Sp1}$  and  $L_{Sp2}$
    - assign  $L_{Sp1}$  and  $L_{Sp2}$  to processors  $2*p-1$  and  $2*p$
    - $j = j + 1$
  - repeat until  $2^j == N$
- Depends on how good is the partition - at random ?
  - May be over long term for large lists and many processors
  - Time taken may be as bad as sequential with bad partitioning

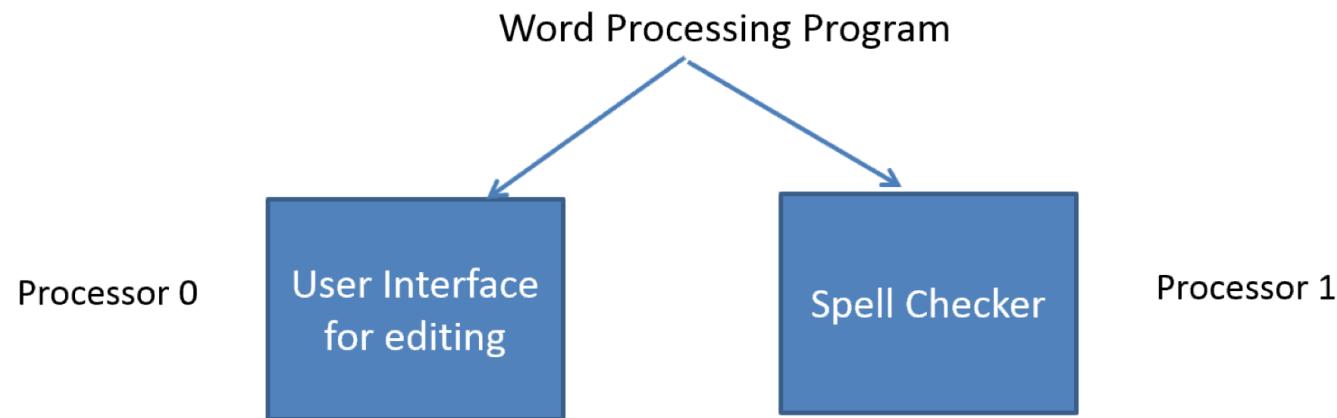


## Tree parallelism summary

- Dynamic version of divide and conquer - partitions are done dynamically
- Division of problem into sub-problems happens execution time
  - Sub-problem is identical in structure to the larger problem
  - What is the division step ?
    - In quick sort it was picking m to split into 2 sub-problems
    - Division / partitioning logic is important to find almost equal sub-problems
- If problem is divided into k sub-problems
  - then in  $\log_k N$  steps needed if N processors execute in parallel
  - If  $p = N$  then work gets done in  $\log(N)$  time with each list item assign to one processor finally
- What if we assign p processors with
  - $p < N$  : so that all processors are utilised
  - $p > N$  : under-utilised processors

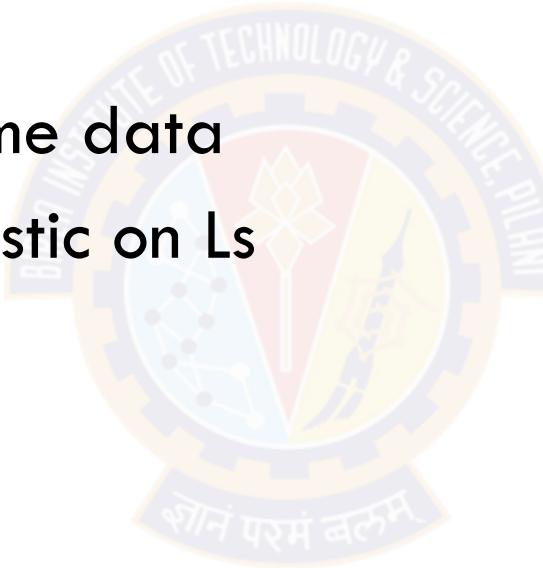
# Task parallelism - Example 1 : Word processor

- Parallel tasks that work on the same data
  - Unlike data and tree parallel Data doesn't need to be divided, the Task gets divided into sub-tasks
  - May work on same data instance, else need to make data copies and keep them in sync
- If on multiple core, different threads can execute tasks in parallel accessing same data instance in memory



## Task parallelism - Example 2 : Independent statistics

- Given a list  $L_s$  of numeric values find its mean, median and mode
- Solution
  - Independent tasks on same data
  - Each task can find a statistic on  $L_s$
  - Run tasks in parallel

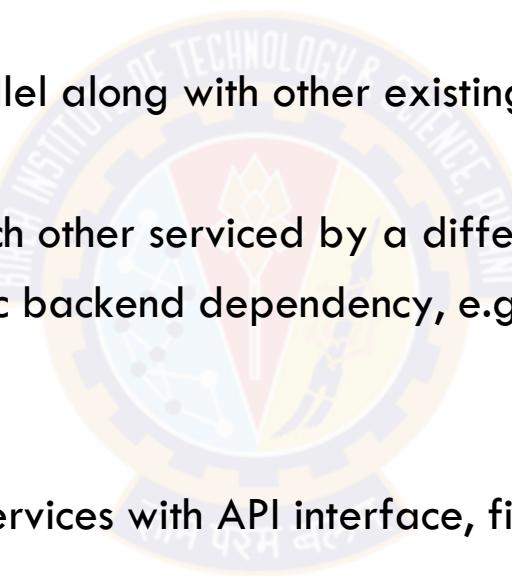


# Task parallelism summary

- Identify sub-tasks based on functionality with no common function
  - In Tree and Data parallel the tasks are identical function
- Sub-tasks are not identified based on data
- Independent sub-tasks are executed in parallel
- Sub-tasks are often limited and known statically in advance
  - We know in a word processor what are the sub-tasks
  - We know in statistical analysis what functions we will run in advance
  - So limited parallelism scope - not scalable with more resources
  - In data or tree parallelism we can potentially get more parallelism with more data
    - more scalable with more resources at same time interval

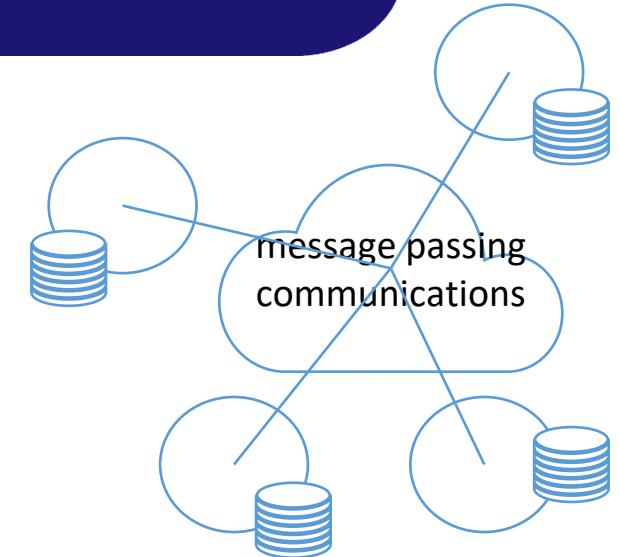
# Request parallelism

- Problem
  - Scalable execution of independent tasks in parallel
  - Execute same code but in many parallel instances
- Solution
  - On arrival, each request is serviced in parallel along with other existing tasks servicing prior requests
  - Could be processing same or fixed data
  - Request-reply pairs are independent of each other serviced by a different thread or process in the backend
    - There could be some application specific backend dependency, e.g. GET and POST on same data item
- Systems Fit
  - Servers in client-server models
  - e.g. email server, HTTP web-server, cloud services with API interface, file / storage servers
  - Microservices
- Scalability metrics : Requests / time (throughput)



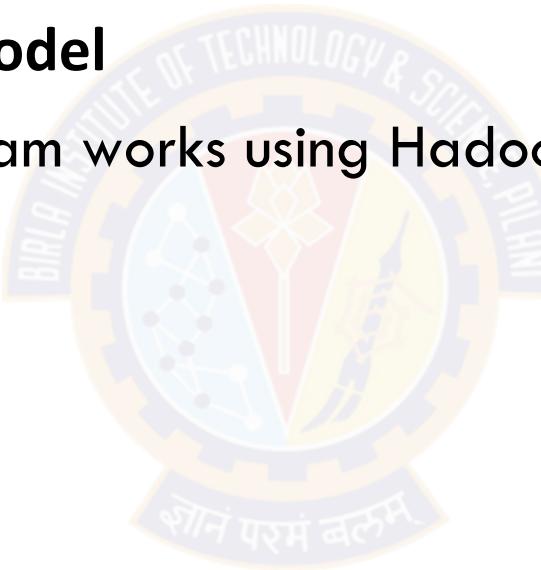
# What happens in a loosely coupled distributed system

- Divide
  - No shared memory
  - Memory / Storage is on separate nodes
  - So any exchange of data or coordination between tasks is via message passing
  - Divide the problem in a way that computation task can run on local data
- Conquer / Merge
  - In shared memory merge it is simpler with each process writing into a memory location
  - In distributed
    - Need to collect data from the different nodes
    - In search example, it is a simpler merge to just collect result - so low cost
    - In quick sort, it is simple append whether writing in place for shared memory or sending a message
  - Sometimes merges may become sequential
    - e.g. k-means - in each iteration (a) guess clusters in parallel to improve the clusters but (2) checking if we have found right clusters is sequential



# Topics for today

- Top down design
- Types of parallelism
- **MapReduce programming model**
- See how a map reduce program works using Hadoop
- Iterative MapReduce



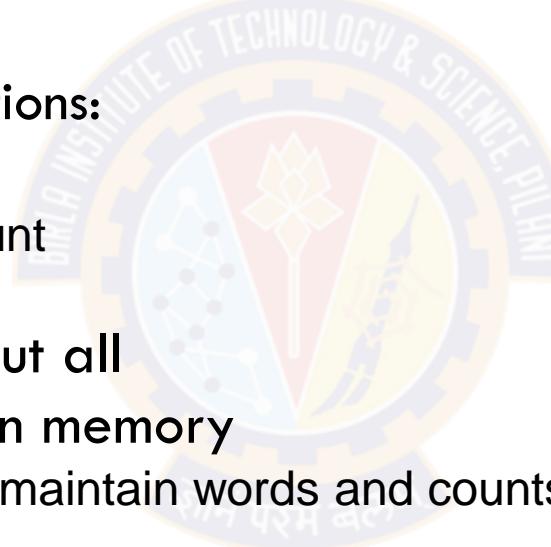
# MapReduce origins

- Created in Google on GFS
- Open source version created as Apache Hadoop
- Perform maps/reduces on data using many machines
  - The system takes care of distributing the data and managing fault tolerance
  - You just write code to map one element and reduce elements to a combined result
- Separates how to do recursive divide-and-conquer from what computation to perform
  - Old idea in higher-order functional programming transferred to large-scale distributed computing
  - Complementary approach to database declarative queries
    - In SQL you don't actually write the low level query execution code
  - Programmer needs to focus just on map and reduce logic and rest of the work is done by the map-reduce framework.
    - So **restricted programming interface** to the system to let the system do the distribution of work, job tracking, fault tolerance etc.

# MapReduce Evolution

- We have a large text file of words, one word in a line
- We need to count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs
- Word Count Solution Design Options:
  - 1: Entire file fits in memory
    - Read file into memory and count
  - 2: File too large for memory, but all  $\langle \text{word}, \text{count} \rangle$  pairs fit in memory
    - Read file line by line and maintain words and counts in memory
  - 3: File on disk, too many distinct words to fit in memory

```
sort words.txt | uniq -c
```



# Solution for multiple large files on disk

To make it slightly harder, suppose we have a large corpus of documents

Count the number of times each distinct word occurs in the corpus

- `words (docs/*) | sort | uniq -c`

where `words` takes a file and outputs the words in it, one to a line

- The above captures the essence of MapReduce
- Great thing is it is naturally parallelizable

MapReduce is conceptually like a UNIX pipeline

- One function (Map) processes data
- Output is input to another function (Reduce)

```
cat words.txt | sort | uniq -c | cat > file
```

```
input | map | shuffle | reduce | output
```



Developer specifies two functions:

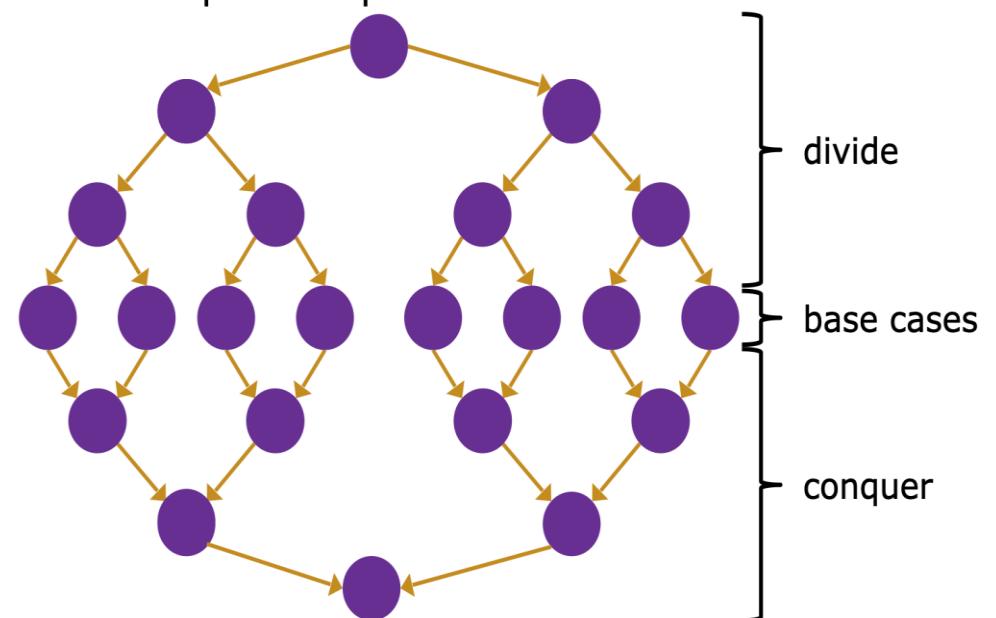
- `map()` - User code
- `reduce()` - User code

→ Rest of the job is done by the MapReduce framework

→ Tune the configuration parameters of the MapReduce framework for performance

# MapReduce in terms of Data and Tree parallelism

- Map
  - Data parallelism
  - Divide a problem into sub-problems based on data
- Reduce
  - Inverse tree parallelism
  - With every merge / reduce the parallelism reduces until we get one result
  - Depending on the problem “reduce” step may be simple or sequential



# Example: Word Count using MapReduce

```
map(key, value):
```

```
// key: document name; value: text of document
```

```
for each word w in value:
```

```
    emit(w, 1)
```

```
reduce(key, values):
```

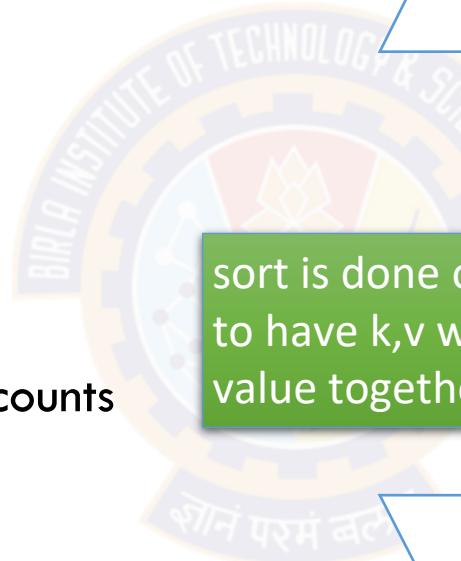
```
// key: a word; value: an iterator over counts
```

```
    result = 0
```

```
    for each count v in values:
```

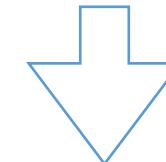
```
        result += v
```

```
    emit(result)
```



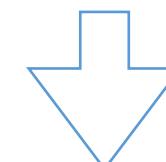
D1 : the blue ship on blue sea

the, 1 | blue, 1 | ship, 1 | on, 1  
blue, 1 | sea, 1



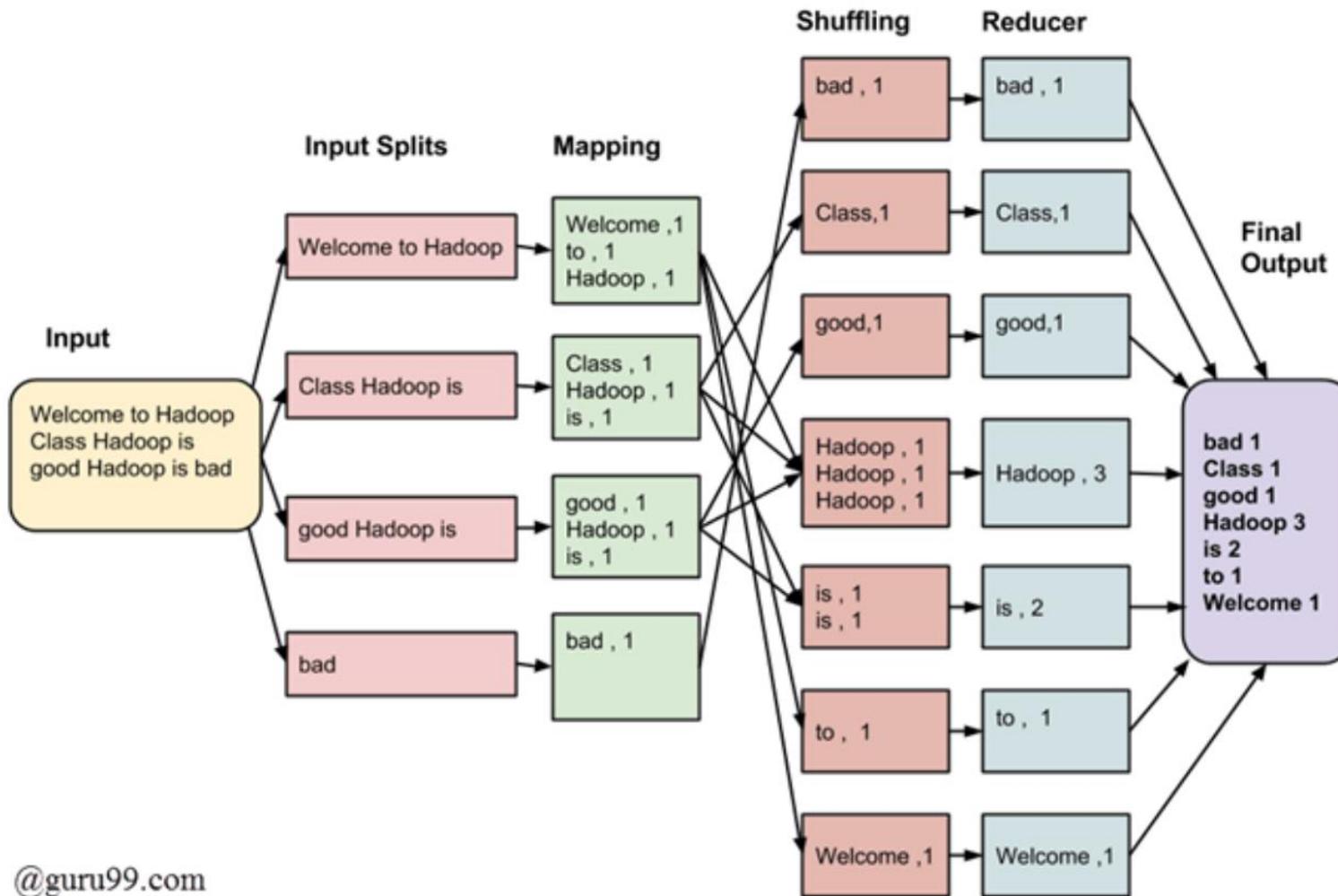
blue, [1,1] | on, 1 | sea, 1 | ship, 1 | the, 1

sort is done on keys  
to have k,v with same k  
value together

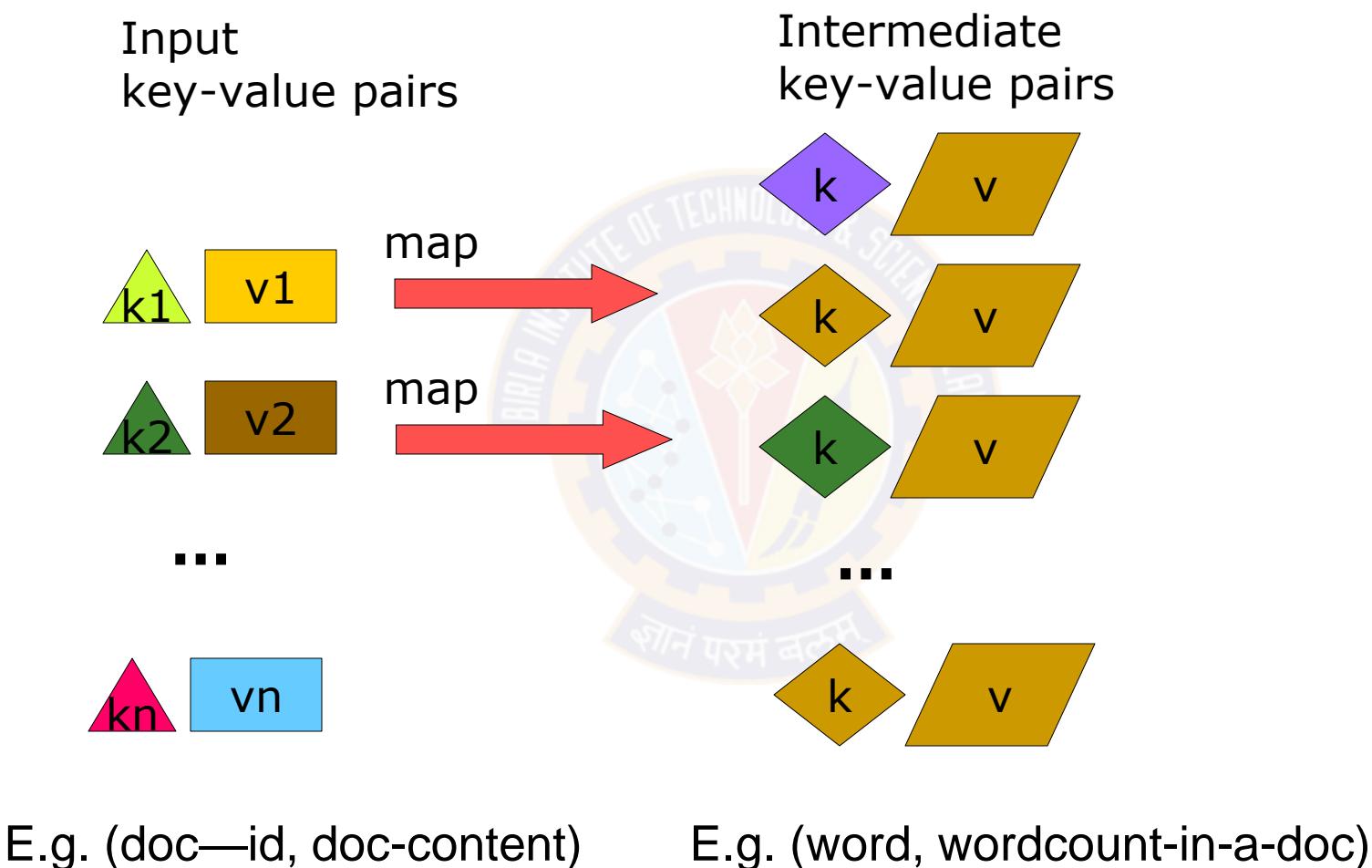


blue, 2 | on, 1 | sea, 1 | ship, 1 | the, 1

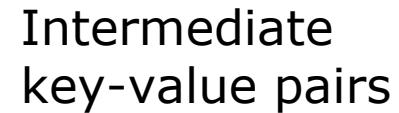
# Word count (2)



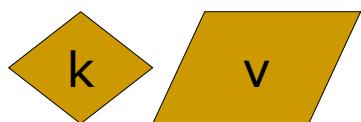
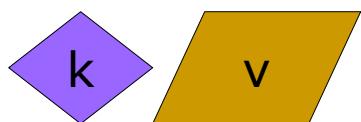
# MapReduce: The Map Step



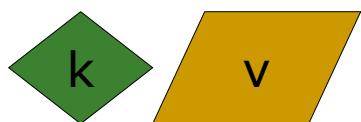
# MapReduce: The Reduce Step



group



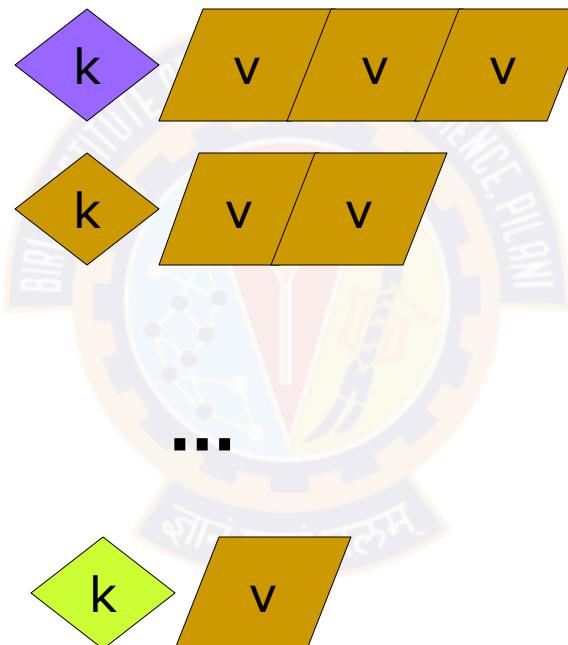
三



E.g.

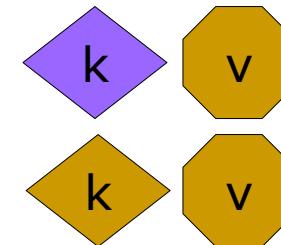
(word, wordcount-in-a-doc)

## Key-value groups

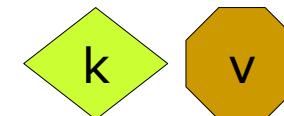


(word, list-of-wordcount)  
~ SQL Group by

## Output key-value pairs



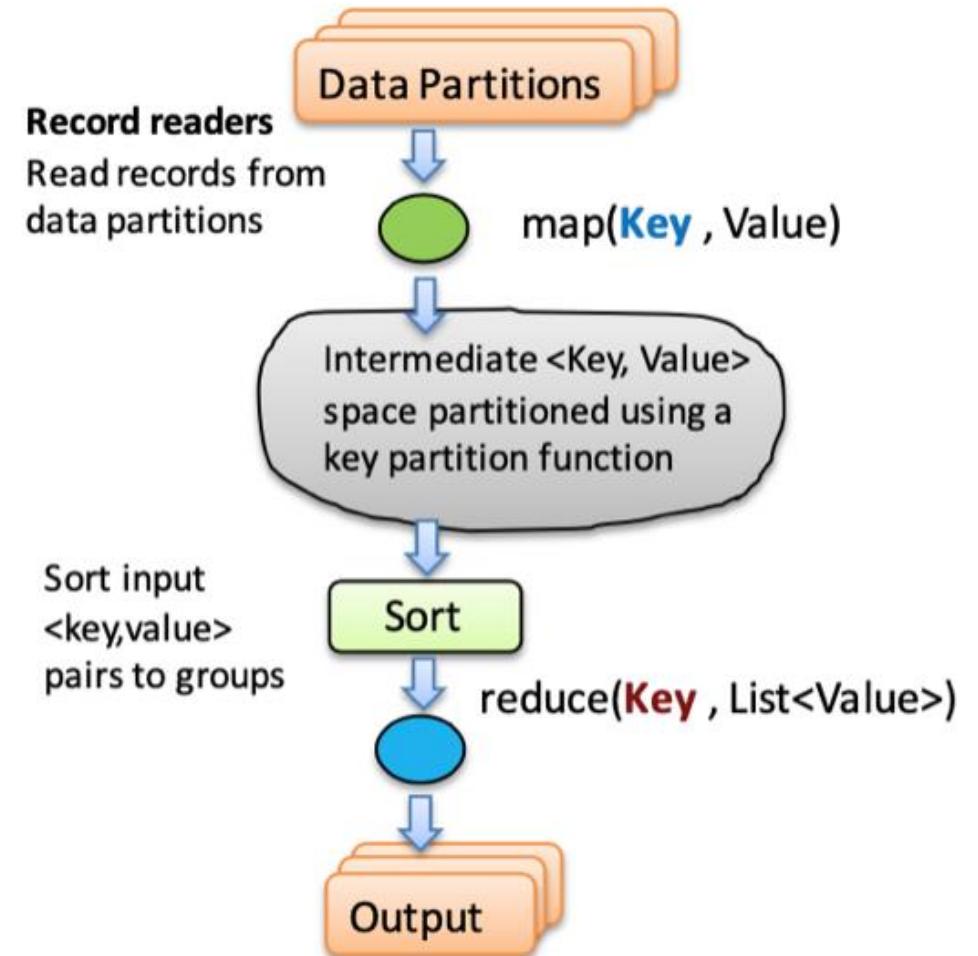
1



(word, final-count)  
~ SQL aggregation

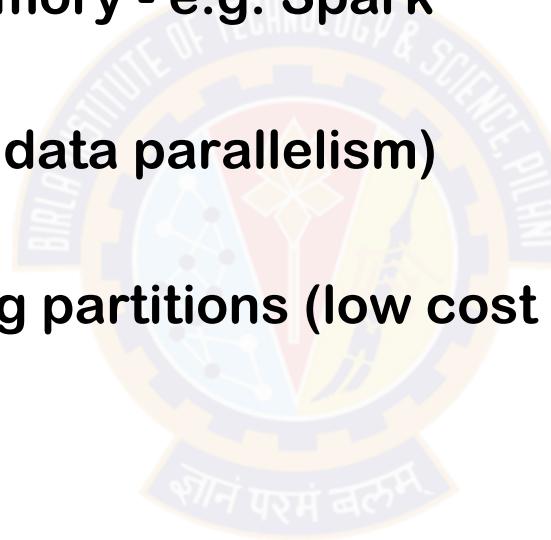
# Formal definition of a MapReduce program

- Input: a set of key/value pairs
- User supplies two functions:
  - $\text{map}(k,v) \rightarrow \text{list}(k_1, v_1)$
  - $\text{reduce}(k_1, \text{list}(v_1)) \rightarrow v_2$
- $(k_1, v_1)$  is an intermediate key/value pair
- Output is the set of  $(k_1, v_2)$  pairs



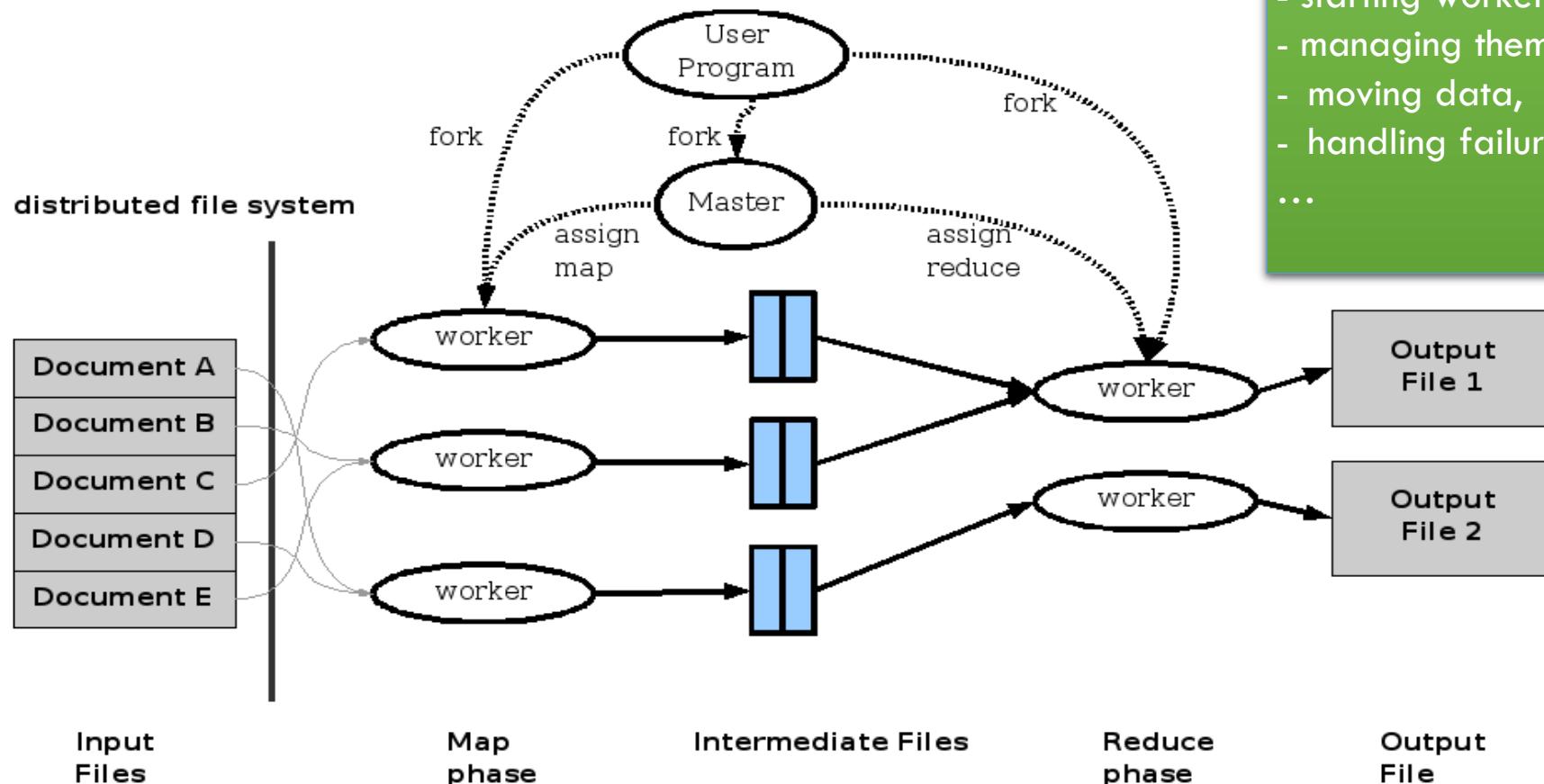
# When will you use this ?

- Huge set of documents that don't fit into memory
  - So need file based processing in stages, e.g. Hadoop
  - But can also do this in memory - e.g. Spark
- Lot of data partitioning (high data parallelism)
- Possibly simple merge among partitions (low cost inverse tree parallelism)



# MapReduce: Execution overview

- Data centric design
- Move computation closer to data
- Intermediate results on disk
- Dynamic task scheduling



A MapReduce library and runtime does all the work for

- allocating resources,
- starting workers,
- managing them,
- moving data,
- handling failures

...

# Topics for today

- Top down design
- Types of parallelism
- MapReduce programming model
- **See how a map reduce program works using Hadoop**
- Iterative MapReduce



# MapReduce example - sales data processing

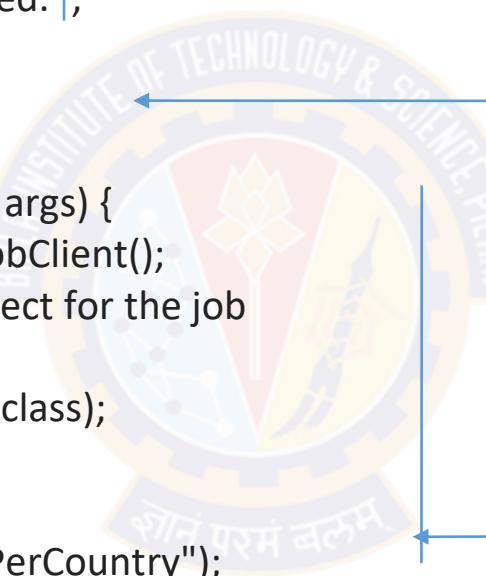
	A	B	C	D	E	F	G	H	I	J	K	L	
1	Transaction_date	Product	Price	Payment_Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude		
2	01-02-2009 06:17	Product1	1200	Mastercard	Carolina	Basildon	England	United Kingdom	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667	
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United States	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819	
4	01-02-2009 13:08	Product1	1200	Mastercard	Federica	Astoria	OR	United States	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83	
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75	
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W.	Cahaba	AL	United States	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025	
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United States	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381	
8	01-04-2009 20:11	Product1	1200	Mastercard	Fleur	Peoria	IL	United States	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889	
9	01-02-2009 20:09	Product1	1200	Mastercard	adam	Martin	TN	United States	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503	
10	01-04-2009 13:17	Product1	1200	Mastercard	Renee	Eli	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-France	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15	
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United States	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064	
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhoven	Noord-Brabant	Netherlands	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667	
14	01-02-2009 09:16	Product1	1200	Mastercard	Sean	Shavano	PTX	United States	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933	
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United States	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353	
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United States	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578	
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstown	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917	
18	01-05-2009 11:27	Product1	1200	Visa	Isaac	Ottawa	Ontario	Canada	01-05-2009 00:00	01-05-2009 10:24	45.41667	76.7	



<https://www.guru99.com/create-your-first-hadoop-program.html>

# Unravelling a MapReduce program - Driver (1)

```
package SalesCountry;  
  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapred.*;  
  
public class SalesCountryDriver {  
  
    public static void main(String[] args) {  
        JobClient my_client = new JobClient();  
        // Create a configuration object for the job  
        JobConf job_conf = new  
        JobConf(SalesCountryDriver.class);  
  
        // Set a name of the Job  
        job_conf.setJobName("SalePerCountry");  
  
        // Specify data type of output key and value  
        job_conf.setOutputKeyClass(Text.class);  
        job_conf.setOutputValueClass(IntWritable.class);  
        ...  
    }  
}
```



package name of jar

hadoop libs

driver class that contains main()

Hadoop job with driver class with SalesPerCountry as the application name

Output data types defined based on existing hadoop classes

# Unravelling a MapReduce program - Driver (2)

...

```
// Specify names of Mapper and Reducer Class  
job_conf.setMapperClass(SalesCountry.SalesMapper.class);  
job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);
```

Mapper and Reducer  
classes in the jar pkg

```
// Set input and output directories  
// arg[0] = name of input directory on HDFS, and  
// arg[1] = name of output directory to be created  
// to store the output file.  
FileInputFormat.setInputPaths(job_conf, new Path(args[0]));  
FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));
```

Paths to read input and  
send output

```
my_client.setConf(job_conf);  
try {  
    // Run the job  
    JobClient.runJob(job_conf);  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

Send job for execution

# Unravelling a MapReduce program - Mapper

```
package SalesCountry;  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.*;  
  
public class SalesMapper extends MapReduceBase implements Mapper <LongWritable, Text,  
Text, IntWritable> {  
    private final static IntWritable one = new IntWritable(1);  
    // can add some data structure here for across map() instances  
    public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable> output,  
    Reporter reporter) throws IOException {  
  
        String valueString = value.toString();  
        String[] SingleCountryData = valueString.split(",");  
        output.collect(new Text(SingleCountryData[7]), one);  
    }  
}
```

← app jar package

← hadoop libs

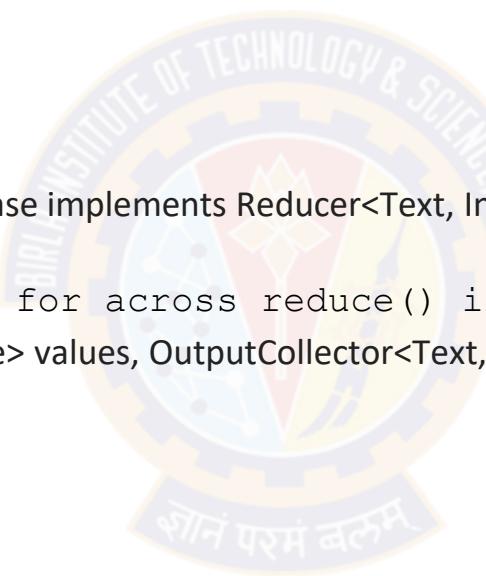
map function w inputs  
- one or more <key, value> pairs  
- output collector  
- ...

Map logic

<India, 1> <USA, 1> <India, 1> ....

# Unravelling a MapReduce program - Reducer

```
package SalesCountry;  
import java.io.IOException;  
import java.util.*;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapred.*;  
  
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,  
IntWritable> {  
    // can add some data structure here for across reduce() instances  
    public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable> output,  
    Reporter reporter) throws IOException {  
        Text key = t_key;  
        int frequencyForCountry = 0;  
        while (values.hasNext()) {  
            // replace type of value with the actual type of our value  
            IntWritable value = (IntWritable) values.next();  
            frequencyForCountry += value.get();  
        }  
        output.collect(key, new IntWritable(frequencyForCountry));  
    }  
}
```



Pkg and includes

reduce function w inputs  
- key and value list, e.g.  
  <India, {1,1,1,1}>  
- output collector  
- ...

Reduce logic  
- calculate frequency

For each reduce task, 1 output file created  
Can control #reducer in driver

# Running and checking status

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# ls -l
total 148
-rw-r--r--. 1 root root 44 Apr 18 01:11 Manifest.txt
-rw-r--r--. 1 root root 2966 Apr 18 01:12 ProductSalePerCountry.jar
drwxr-xr-x. 2 root root 96 Apr 19 00:57 SalesCountry
-rw-r--r--. 1 root root 1529 Apr 18 00:57 SalesCountryDriver.java
-rw-r--r--. 1 root root 746 Apr 18 00:56 SalesCountryReducer.java
-rw-r--r--. 1 root root 123637 Jun 6 16:56 SalesJan2009.csv
-rw-r--r--. 1 root root 661 Apr 18 00:56 SalesMapper.java
drwxr-xr-x. 3 root root 157 Jun 6 16:53 units
-rw-r--r--. 1 root root 219 Apr 18 21:10 units.csv
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# hadoop jar ProductSalePerCountry.jar /SalesJan2009.csv /output
21/06/06 17:27:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/06/06 17:27:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute remedy this.
21/06/06 17:27:40 INFO mapred.FileInputFormat: Total input files to process : 1
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: number of splits:2
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1622978356181_0002
21/06/06 17:27:41 INFO conf.Configuration: resource-types.xml not found
21/06/06 17:27:41 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/06/06 17:27:41 INFO impl.YarnClientImpl: Submitted application application_1622978356181_0002
21/06/06 17:27:41 INFO mapreduce.Job: The url to track the job: http://centos-s-4vcpu-8gb-blr1-01:8088/proxy/application_1622978356181_0002/
21/06/06 17:27:41 INFO mapreduce.Job: Running job: job_1622978356181_0002
21/06/06 17:27:49 INFO mapreduce.Job: Job job_1622978356181_0002 running in uber mode : false
21/06/06 17:27:49 INFO mapreduce.Job: map 0% reduce 0%
21/06/06 17:28:03 INFO mapreduce.Job: map 100% reduce 0%
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# jps
7744 SecondaryNameNode
8357 ResourceManager
8581 NodeManager
13541 Jps
7238 DataNode
10428 MRAppMaster
6910 NameNode
```



## MapReduce Application application\_1622978356181\_0002

Logged in as: dr.who

Active Jobs										
Show 20 entries Search:										
Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed		
job_1622978356181_0002	SalePerCountry	RUNNING	2	2		1		0		

Showing 1 to 1 of 1 entries First Previous 1 Next Last

# MapReduce stats

## File System Counters

```
FILE: Number of bytes read=17747
FILE: Number of bytes written=660936
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=127535
HDFS: Number of bytes written=661
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
```

## Job Counters

```
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=14658
Total time spent by all reduces in occupied slots (ms)=7011
Total time spent by all map tasks (ms)=14658
Total time spent by all reduce tasks (ms)=7011
Total vcore-milliseconds taken by all map tasks=14658
Total vcore-milliseconds taken by all reduce tasks=7011
Total megabyte-milliseconds taken by all map tasks=15009792
Total megabyte-milliseconds taken by all reduce tasks=7179264
```

## Map-Reduce Framework

```
Map input records=999
Map output records=999
Map output bytes=15743
Map output materialized bytes=17753
Input split bytes=180
Combine input records=0
Combine output records=0
Reduce input groups=58
Reduce shuffle bytes=17753
Reduce input records=999
Reduce output records=58
Spilled Records=1998
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=585
CPU time spent (ms)=2510
Physical memory (bytes) snapshot=772923392
Virtual memory (bytes) snapshot=6393163776
Total committed heap usage (bytes)=527433728
```

## Shuffle Errors

```
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
```

## File Input Format Counters

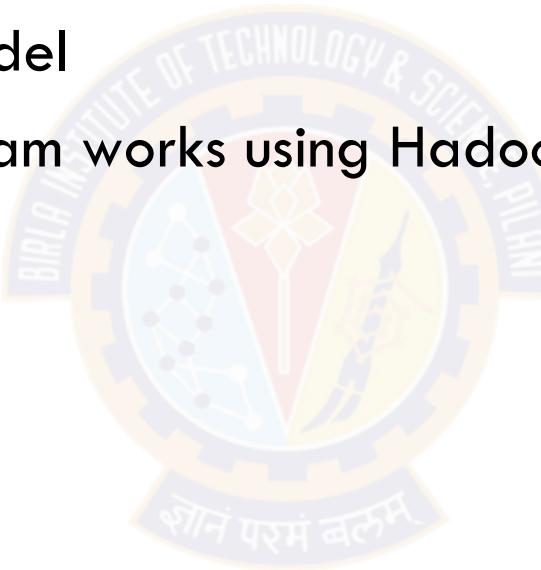
```
Bytes Read=127355
```

## File Output Format Counters

```
Bytes Written=661
```

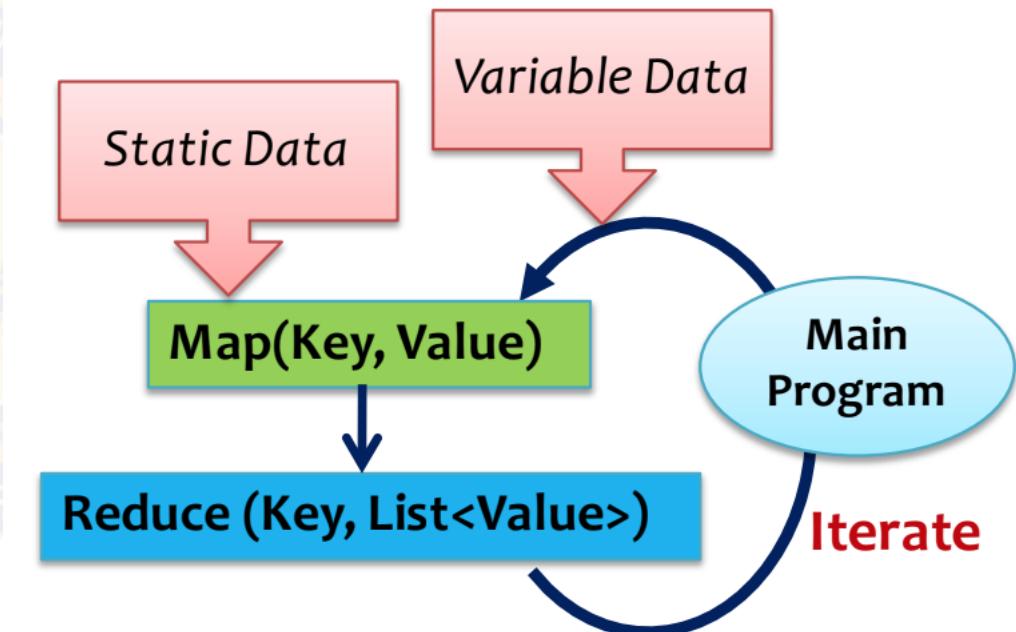
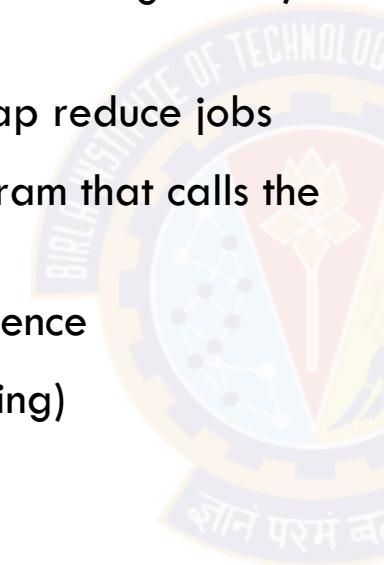
# Topics for today

- Top down design
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works using Hadoop
- K-Means clustering (Sec1)
- **Iterative MapReduce**

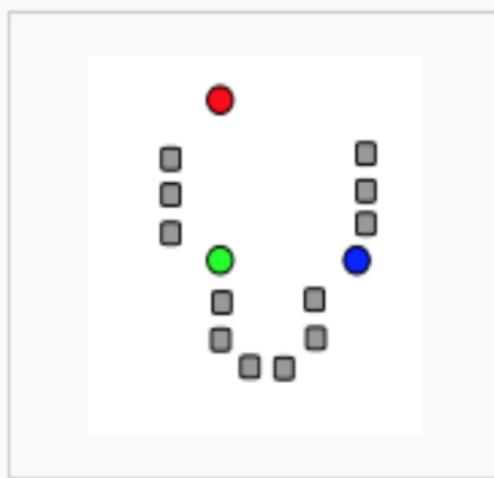


# Iterative map reduce

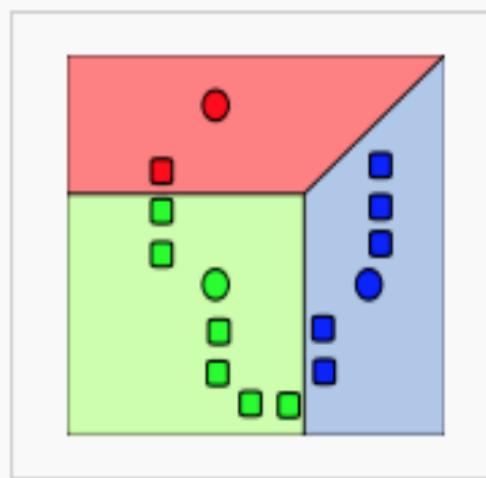
- MapReduce is a one-pass computation
- Many applications, esp in ML and Data Mining areas, need to iteratively process data
- So they need iterative execution of map reduce jobs
- An approach is to create a main program that calls the core map reduce with variable data
- Core program also checks for convergence
  - error bound (e.g. k-means clustering)
  - fixed iterations



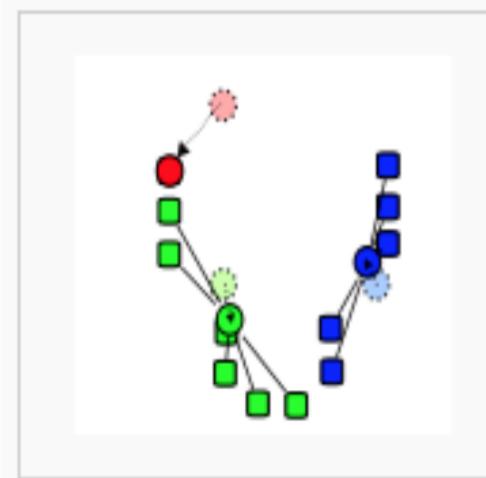
# Example 1: K-means clustering



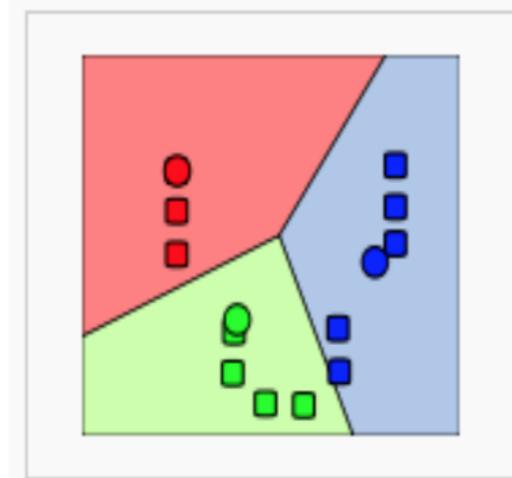
1)  $k$  initial "means" (in this case  $k=3$ ) are randomly selected from the data set (shown in color).



2)  $k$  clusters are created by associating every observation with the nearest mean. The partitions here represent the [Voronoi diagram](#) generated by the means.



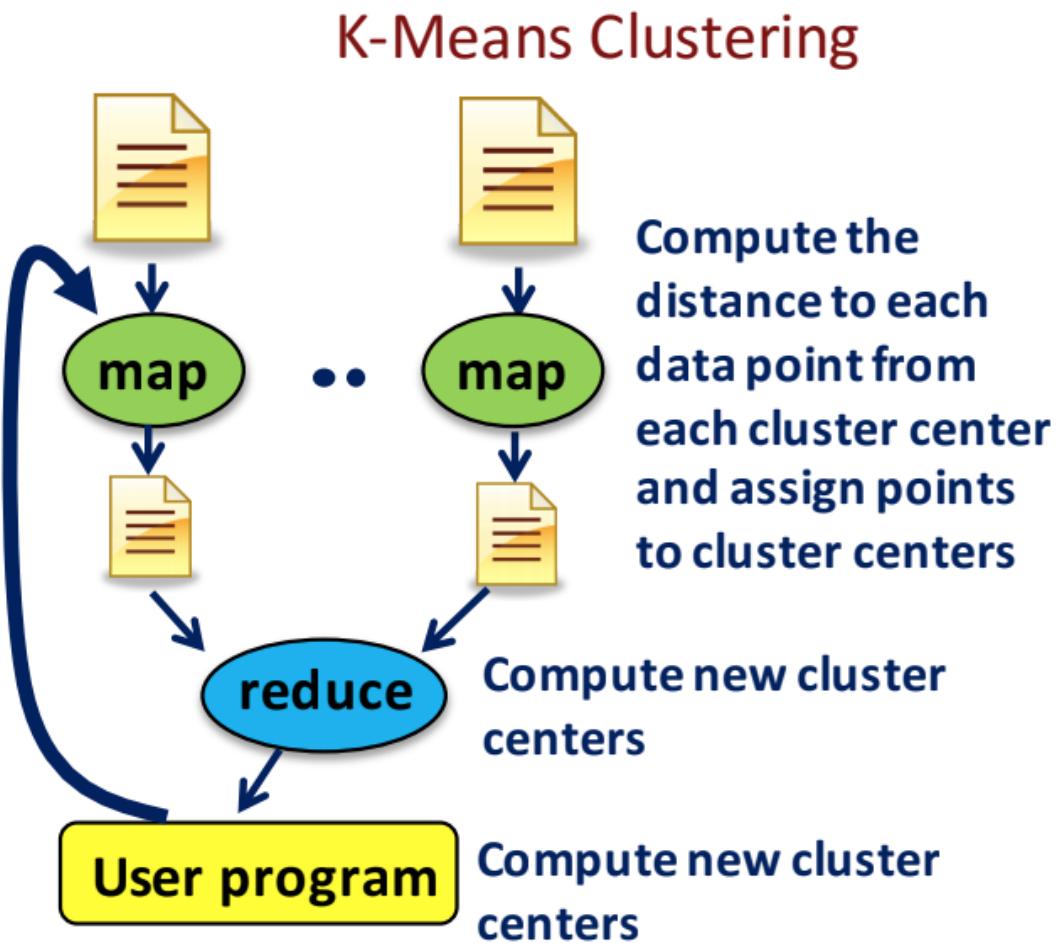
3) The [centroid](#) of each of the  $k$  clusters becomes the new means.



4) Steps 2 and 3 are repeated until convergence has been reached.

<http://shabal.in/visuals/kmeans/2.html>

# K-means as iterative map reduce



- The MapReduce program driver is responsible for repeating the steps via an iterative construct.
- Within each iteration map and reduce steps are called.
- Each map step reuses the result produced in previous reduce step.
  - e.g. k centers computed

<https://github.com/thomasjungblut/mapreduce-kmeans/tree/master/src/de/jungblut/clustering/mapreduce>

# K-Means Clustering by Iterative MapReduce

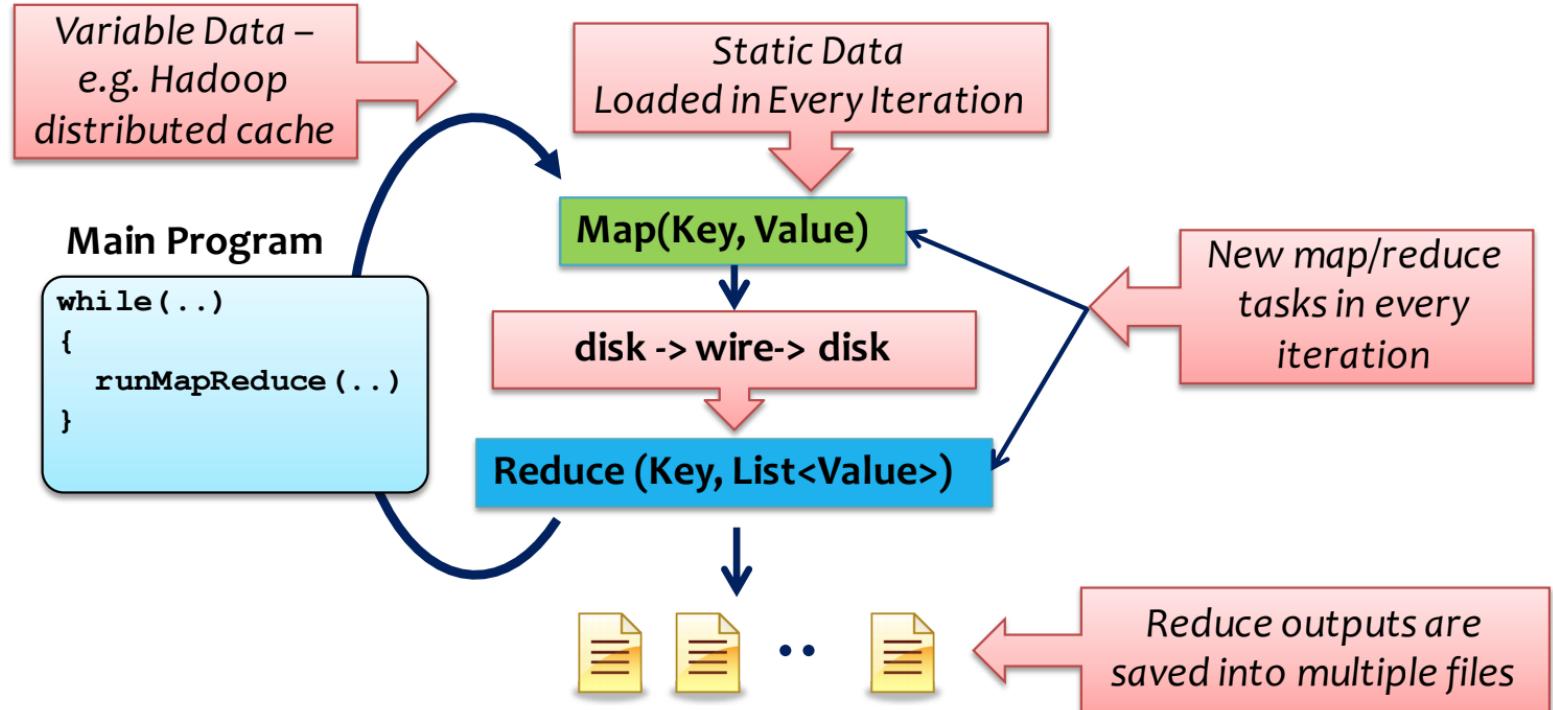
## Hands on demo

- 20Newsgroups folder - a set of around 20,000 postings to 20 different newsgroups with 1000 postings
- Convert all the newsgroup postings and turn them into bag-of-words vectors. – /data folder in HDFS
- Run a program that will choose an initial set of cluster centroids from the data – /clusters folder in HDFS (randomly sampled from the vectors)
- Run KMeans on Hadoop - hadoop jar MapRedKMeans.jar KMeans /data /clusters 3  
This will run 3 iterations of the KMeans algorithm on top of all documents in the 20\_newsgroups data set.  
This means that three separate MapReduce jobs will be run in sequence.
- The centroids produced at the end of:
  1. Iteration 1 will be put into the HDFS directory "/clusters1",
  2. Iteration 2 will be put into the HDFS directory "/clusters2",
  3. Iteration 3 will be put into the HDFS directory "/clusters3",

Reference: <https://cmj4.web.rice.edu/MapRedKMeans.html>

# Iterations using existing runtimes

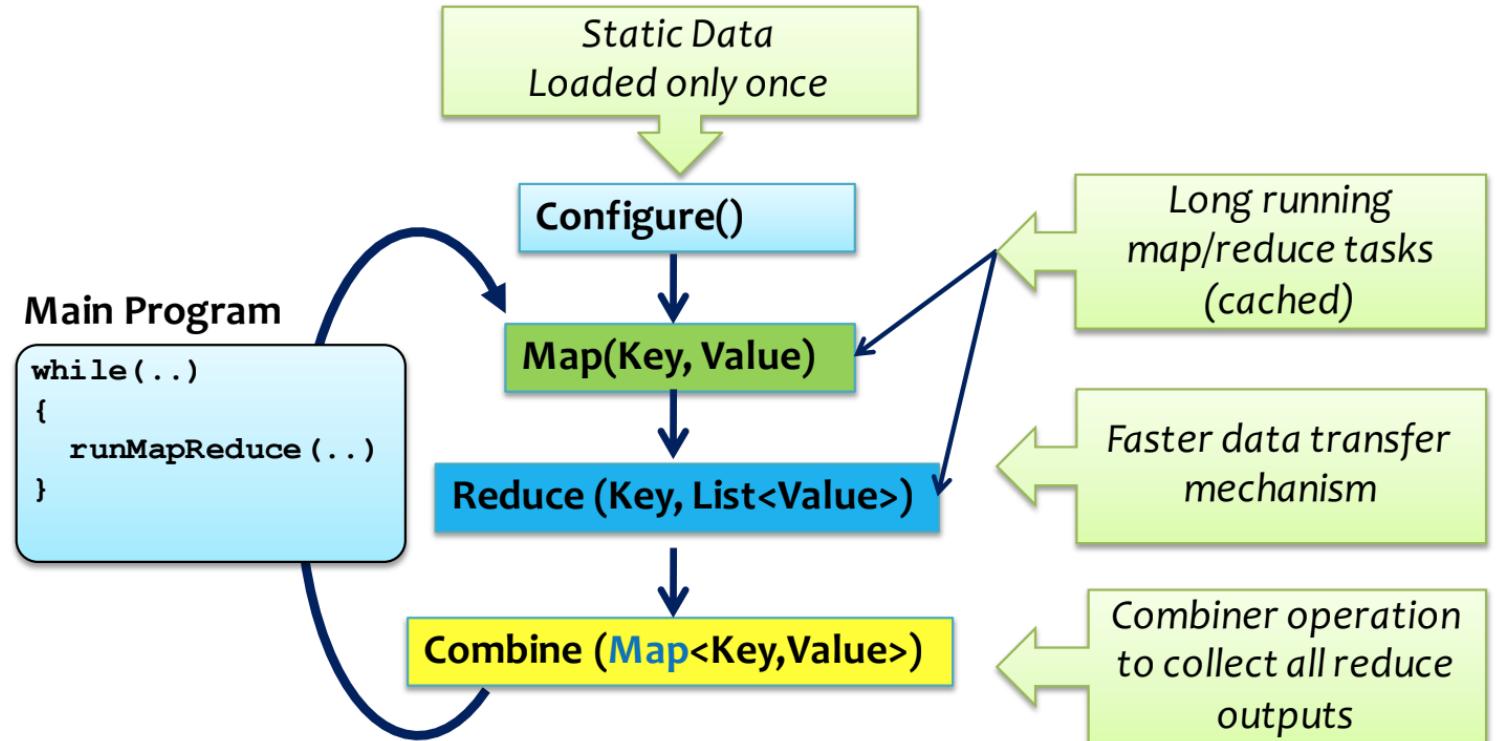
- Loop implemented on top of existing file-based single step map-reduce core
- Large overheads from
  - re-initialization of tasks
  - reloading of static data
  - communication and data transfers



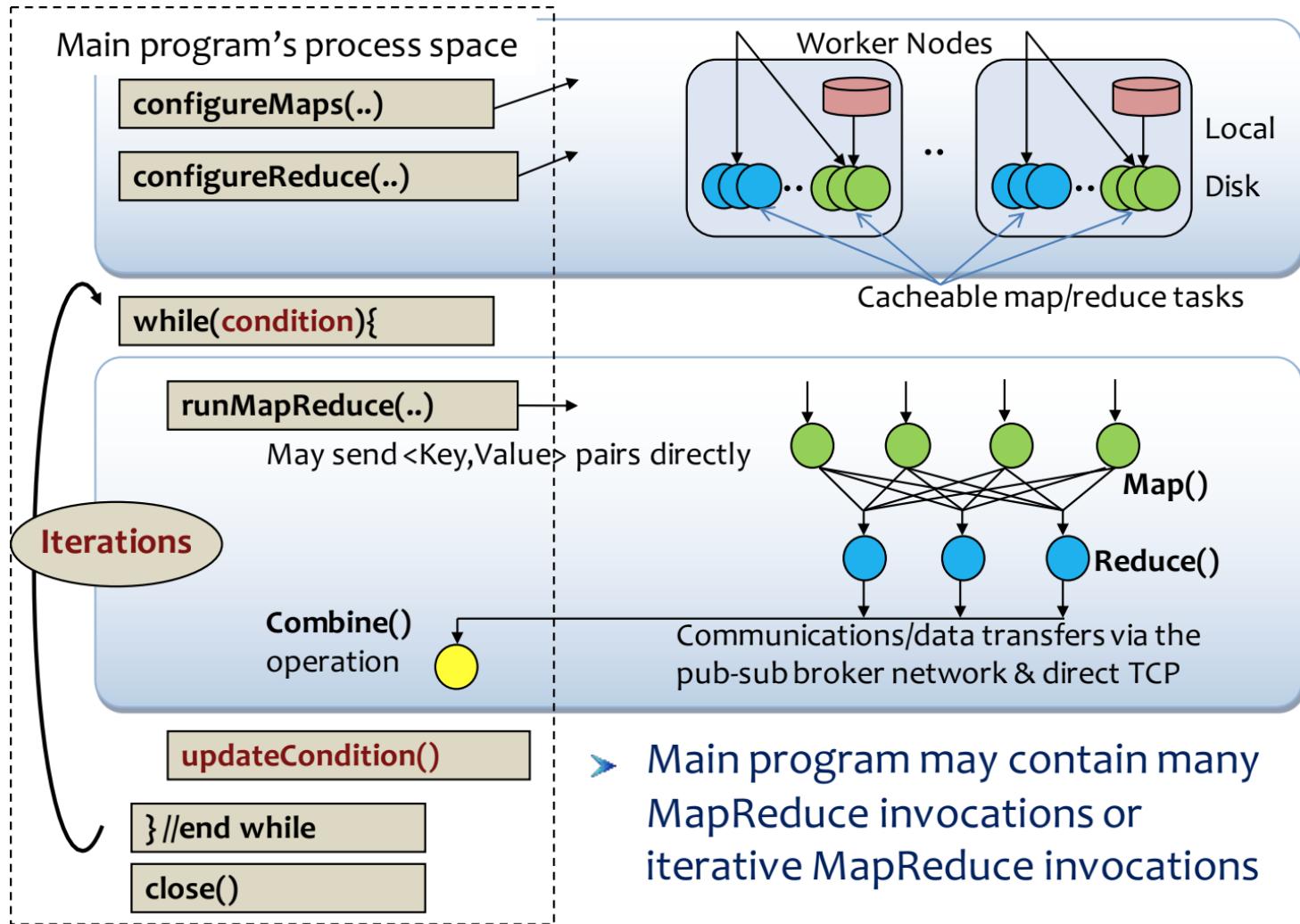
DistributedCache: <https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html>

# MapReduce++ : Iterative MapReduce

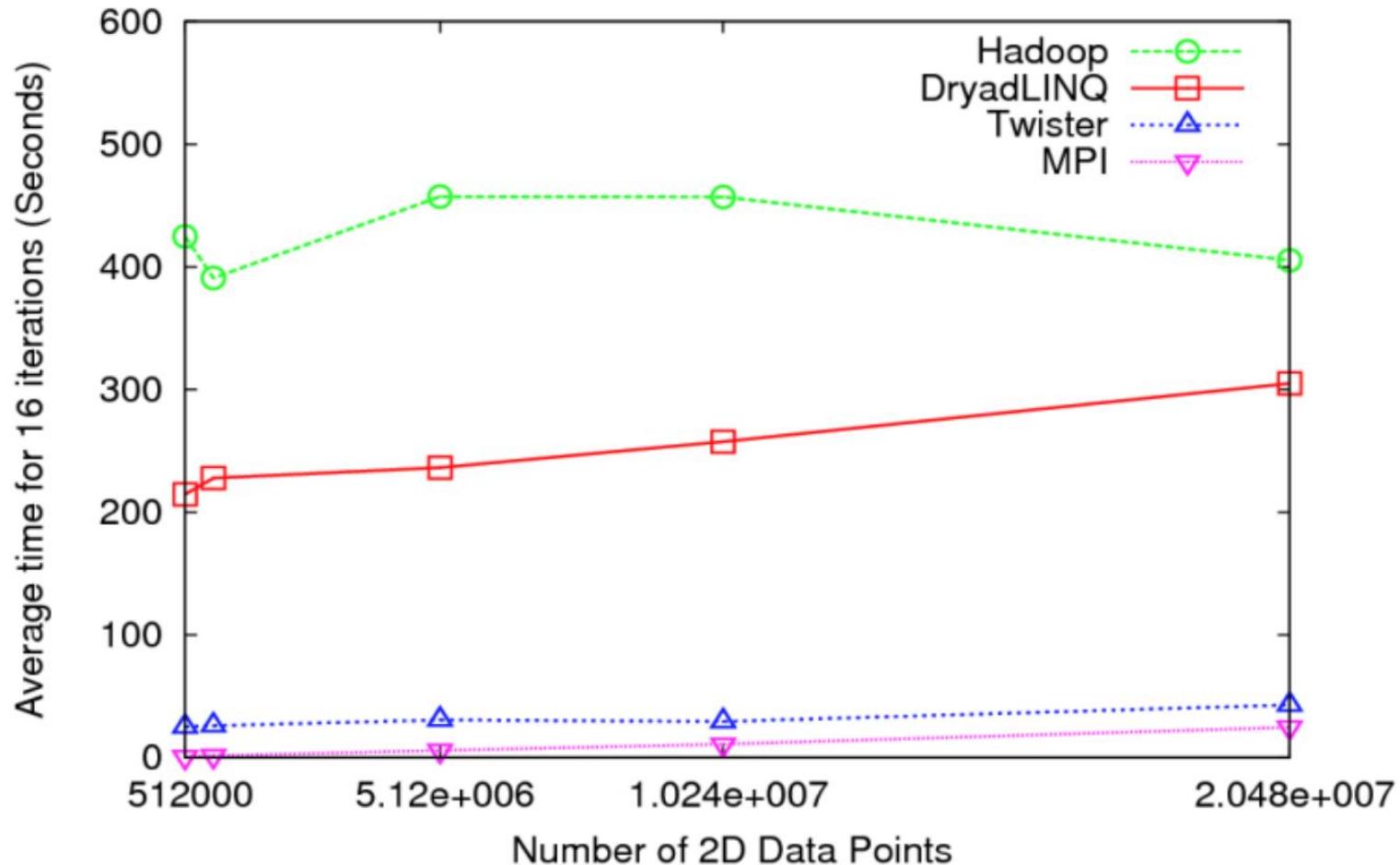
- Some optimizations are done on top of existing model
  - Static data loaded once
  - Cached tasks across invocations
  - Combine operations



# Example in Twister: Enables more APIs



# The optimisations indeed help



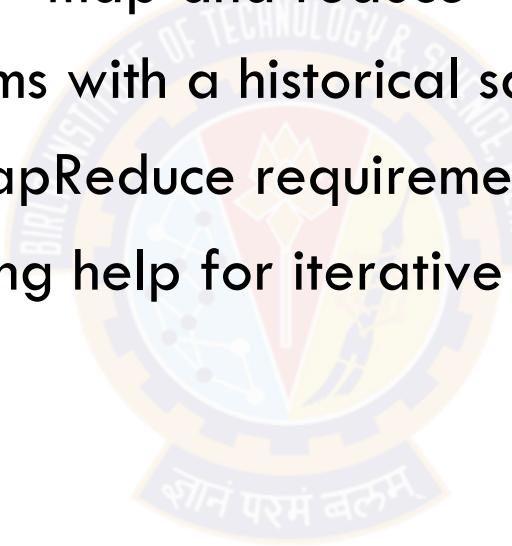
K-means clustering using various programming models

# Iterative MapReduce: Other options

- HaLoop
  - Modifies Hadoop scheduling to make it loop aware
  - Implements caches to avoid going to disk between iterations
  - Optional reading: Paper in [Proceedings of the VLDB Endowment](#) 3(1):285-296, Sep 2010
- Spark
  - Uses in-memory computing to speed up iterations
  - An in-memory structure called RDD : Resilient Distributed Dataset replaces files on disk
  - Ideal for iterative computations that reuse lot of data in each iteration

# Summary

- Different types of parallelism
- Data and tree parallelism —> map and reduce
- Basics of MapReduce programs with a historical sales data processing example
- Optimizations for iterative MapReduce requirements
- How does in-memory computing help for iterative MapReduce programming





**Next Session:**  
Hadoop MapReduce and YARN



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 7: Hadoop MapReduce and YARN

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **Hadoop MapReduce**
  - ✓ **MapReduce runtime**
  - ✓ More examples
  - ✓ Hadoop Streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands



# MapReduce Programming Architecture

1. Input dataset is split into multiple pieces of data (several small sets)
2. Framework creates a master (application master) and several worker processes and executes the worker processes remotely
3. Several Map tasks work simultaneously and read pieces of data that were assigned to each map. Map worker uses the map function to extract only those data that are present on their server and generates key/value pair for the extracted data.
4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of specified mapper.
5. When the map workers complete their work, the master instructs the reduce workers to begin their work.
6. The reduce workers in turn contact the map workers to get the key/value data for their partition (shuffle). The data thus received from various mappers is merge sorted as per keys.
7. Then it calls reduce function on every unique key. This function writes output to the file.
8. When all the reduce workers complete their work, the master transfers the control to the user program.

# Map

## 1. Record Reader

- ✓ Reads each record created by input splitter to pass key-value pair into Map
- ✓ Could be text, binary etc.
- ✓ Examples: LineRecordReader, SequenceFileRecordReader

## 2. Map

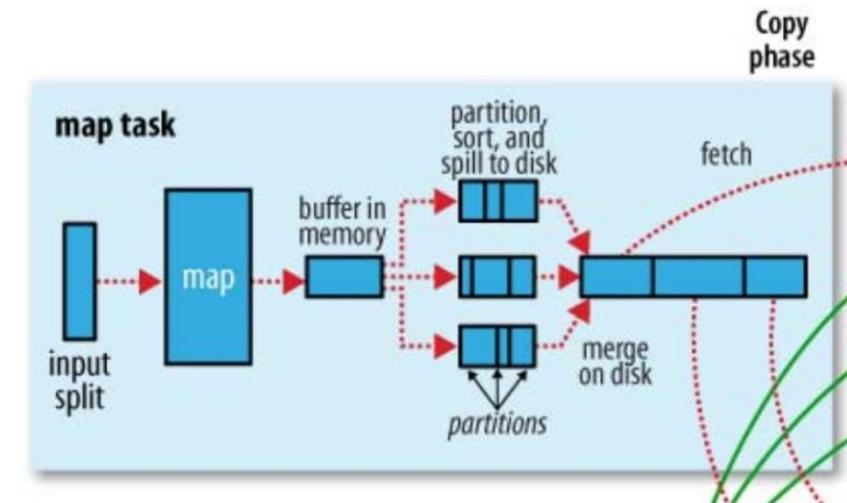
- ✓ User defined function to create key, value pair from input key, value pair

## 3. Combiner

- ✓ Localized optional reducer for better performance - can be same as reducer code (discussed later)
- ✓ e.g. <hello,1> x 3 pairs on same node can be <hello, 3>

## 4. Partitioner

- ✓ Takes intermediate output from map and shards it to send to different reducers, i.e. determines which reducer should get a shard (some sorting happens based on partition logic)
- ✓ Default partitioner: `key.hashCode()%num_reducers` spreads keyspace evenly among reducers
- ✓ Ensures same key is sent to same reducer
- ✓ Shards are written to disk waiting for reducer to pull
- ✓ Custom partitioner class can be implemented (see T1, Pg 226 example, also discussed later)



# Reduce

## 1. Shuffle and Sort

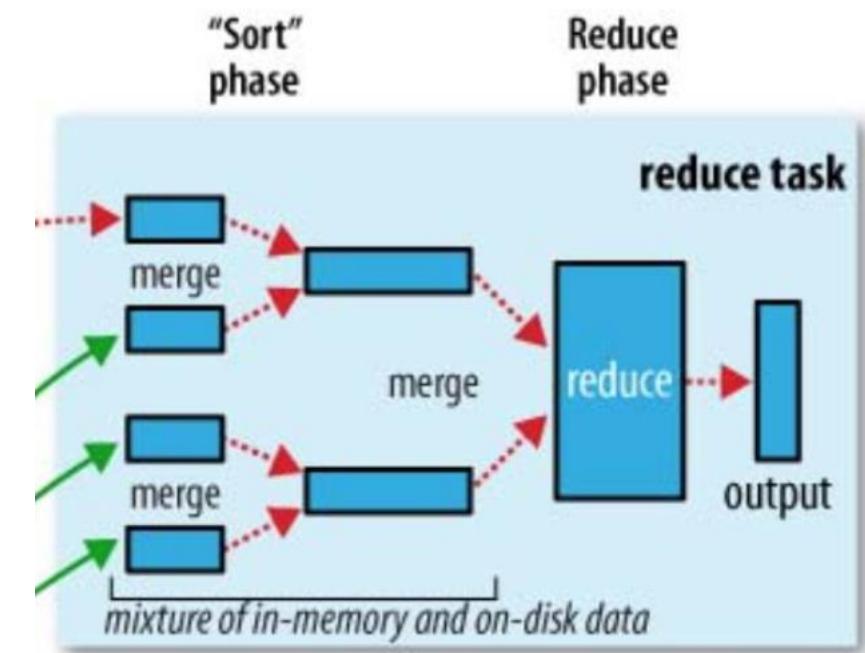
- ✓ Shuffle gets data from map node to reduce node or reduce task to happen where the relevant post-map data partition is
- ✓ Shuffling can start before Map is entirely complete
- ✓ Data is merge-sorted by key coming in from multiple maps

## 2. Reduce

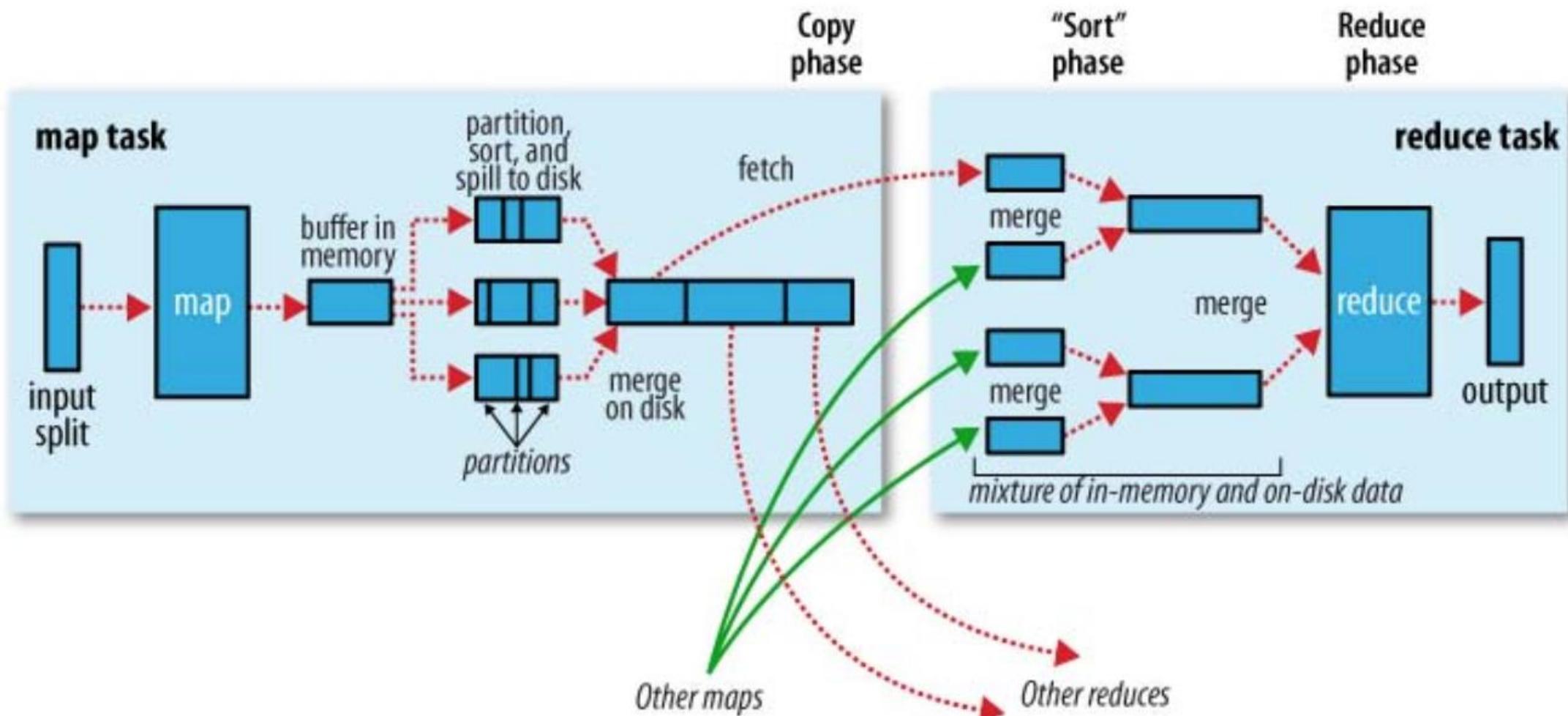
- ✓ Call user defined function per key grouping, e.g. <India,{1,1,1,1}>
- ✓ A reduce call looks at a unique key but global state can be maintained in Reduce task in class variable
- ✓ Can control number of reducers, e.g. one reducer if a sequential computation has to calculate one final value

## 3. Output Format

- ✓ Writes final output of reducer with separator of key, value and separator between pairs



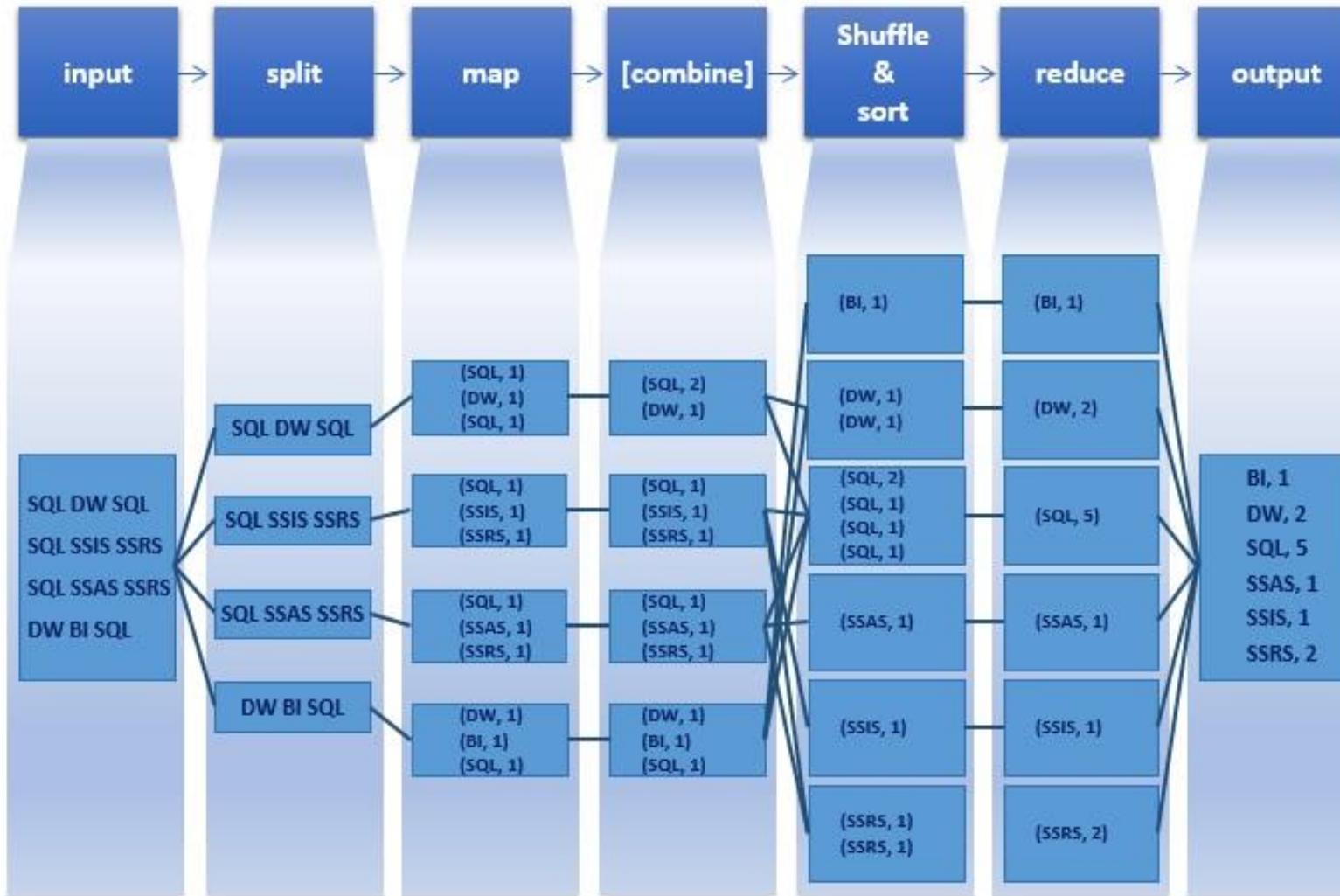
# MapReduce flow



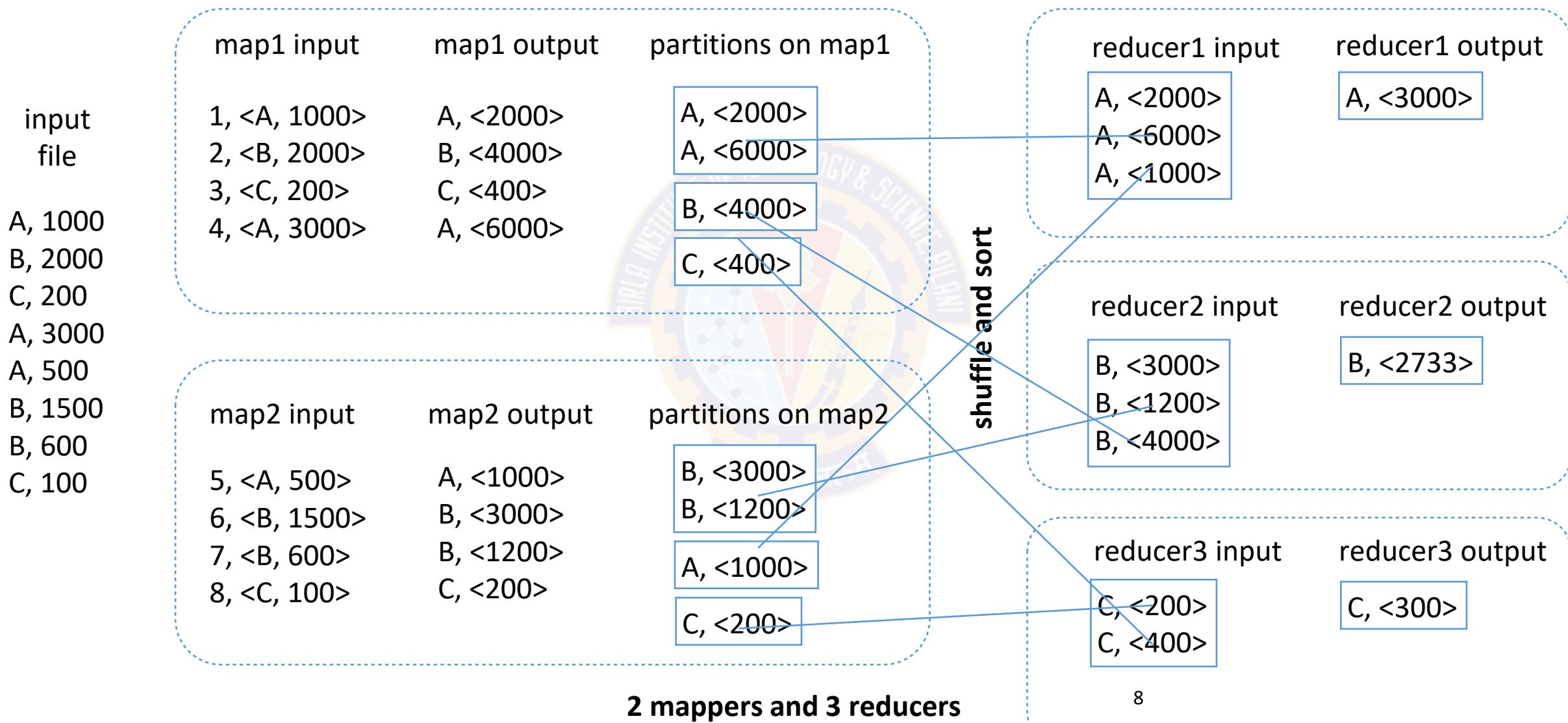
<https://stackoverflow.com/questions/22141631/what-is-the-purpose-of-shuffling-and-sorting-phase-in-the-reducer-in-map-reduce>

# MapReduce Data Flow

## MapReduce – Word Count Example Flow



# Map Reduce Example (input $\langle k, v \rangle$ find $\text{avg}(2v)$ for each $k$ )



# A word about Combiner

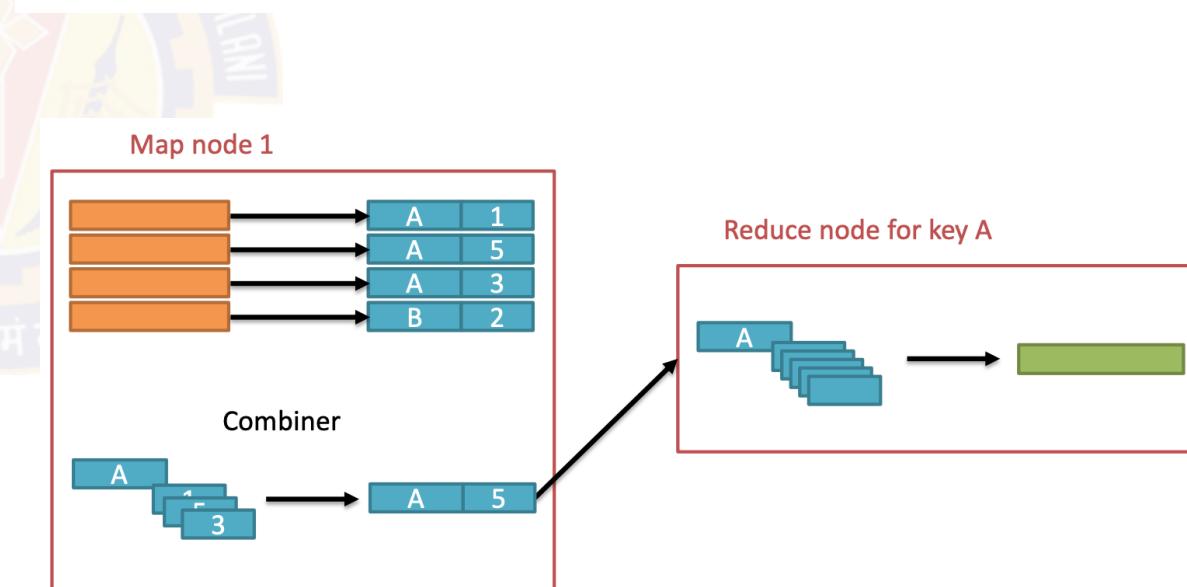
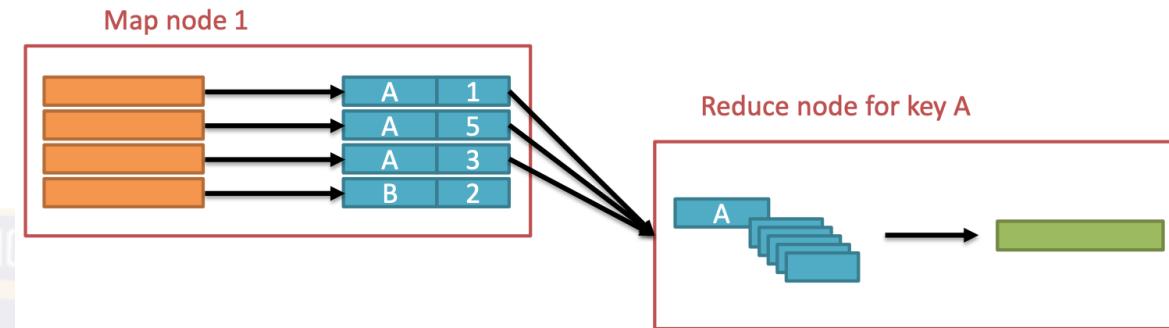
- Combiners can optimise the reduce by pre-processing on each node to compress data but output has to be same type as a map

- The reducer can also be set as the combiner class only if reduce is an associative and commutative operation

✓  $\max(1,2, \max(3,4,5)) = \max(\max(2,4), \max(1,5,3))$  or any other order

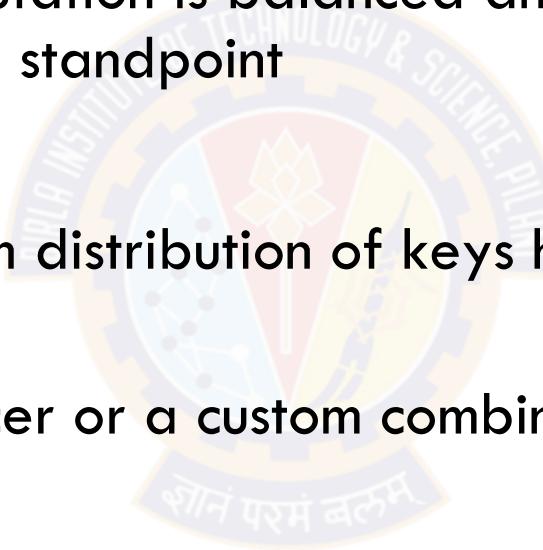
✓  $\text{avg}(1,2,\text{avg}(3,4,5)) = 2.33\dots$  but  
 $\text{avg}(\text{avg}(2,4),\text{avg}(1,5,3)) = 3$

- If not C&A then optionally write a new combiner logic



# Performance optimisations

- We studied types of parallelism earlier
- The goal for a parallel computation is balanced and maximum utilisation of the cluster from CPU and memory standpoint
- So carefully look at
  - ✓ partitioner - will a custom distribution of keys help and not use the key hash code default
  - ✓ combiner - will the reducer or a custom combiner help for compression of data to reducer



# MR job execution

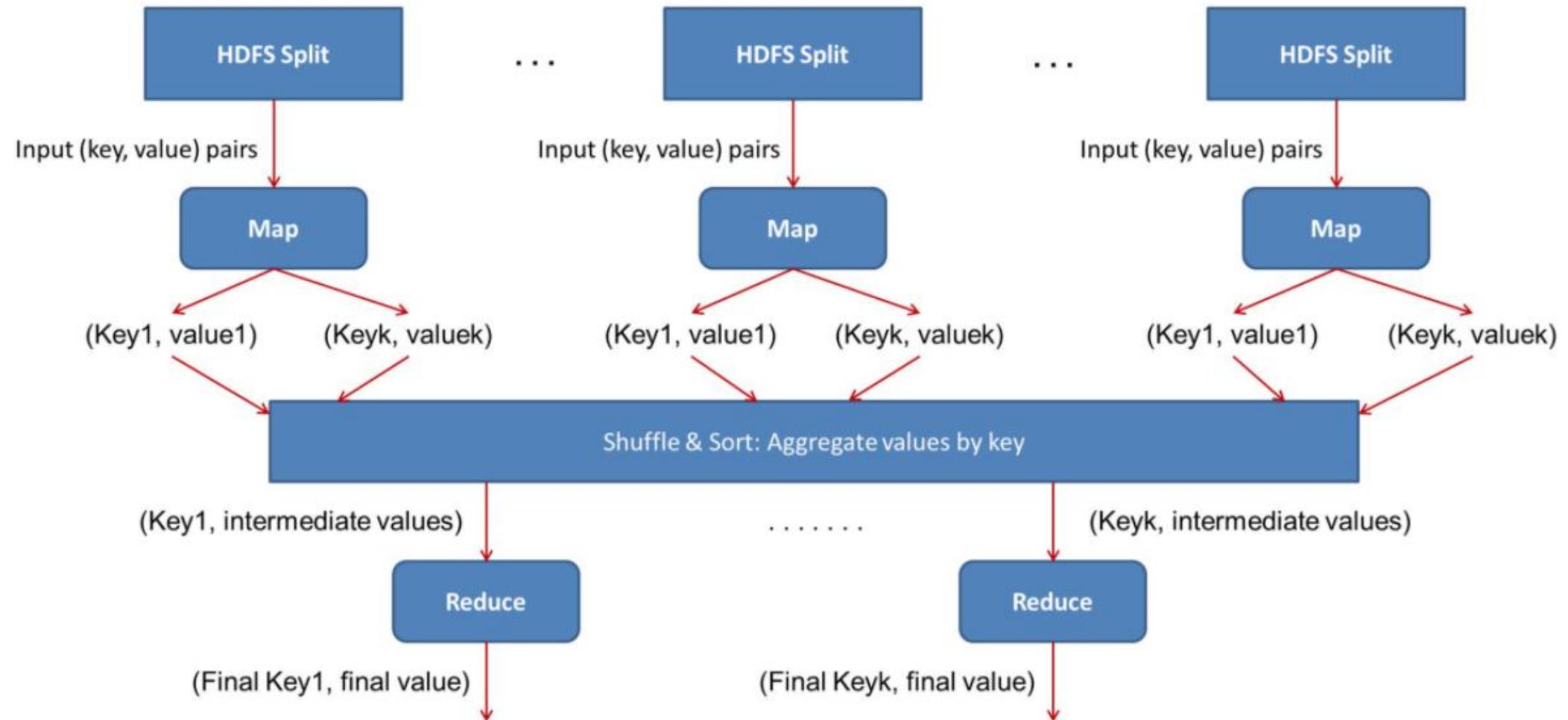


Image ref: <https://data-flair.training/blogs/hadoop-architecture/>

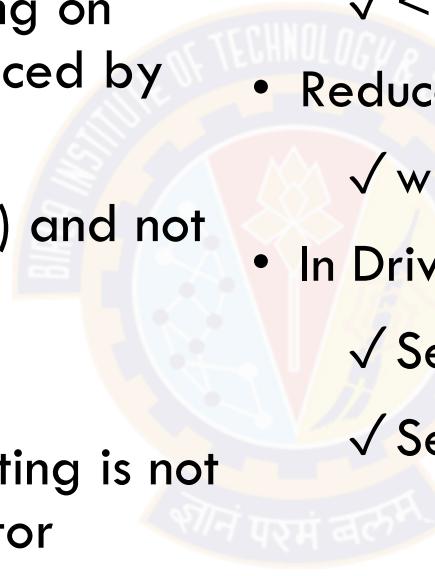
# Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop Streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands



# Example 1: Sorting by value

- Sort all employees by their salary given input file with records <name, salary>
- MapReduce automatically does sorting on keys given <key, value> pairs produced by Map.
- But we need to sort on values (salary) and not keys (name).
- So swap the key and values in map()
- Can set comparator class if value sorting is not done properly with default comparator
- Map logic
  - ✓ <key=k, value=v> -> <key=v, value=k>
- Reduce logic
  - ✓ write <k,v> - no extra logic
- In Driver
  - ✓ Set job.NumReduceTasks(1)
  - ✓ Set job.setComparatorClass(intComparator.class)



# Example 2: Find max / min in each group

Find max / min FX rate every year for each country

Date,Country,Value

1971-01-04,Australia,0.8987

1971-01-05,Australia,0.8983

1971-01-06,Australia,0.8977

1971-01-07,Australia,0.8978

1971-01-08,Australia,0.899

1971-01-11,Australia,0.8967

1971-01-12,Australia,0.8964

1971-01-13,Australia,0.8957

1971-01-14,Australia,0.8937

Key is year and country combination

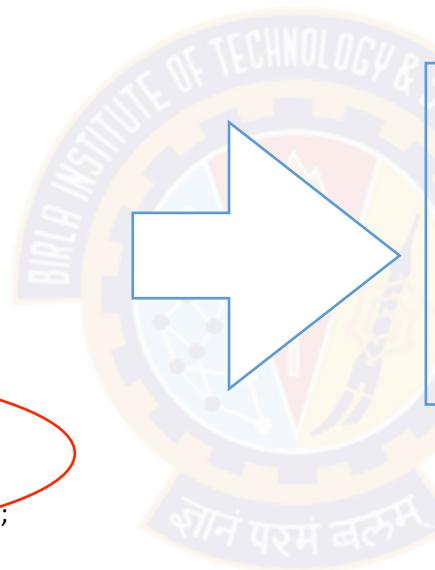


1971 Australia MIN	0.8412
1971 Australia MAX	0.899
1971 Austria MIN	23.638
1971 Austria MAX	25.873
1971 Belgium MIN	45.49
1971 Belgium MAX	49.73
1971 Canada MIN	0.9933
1971 Canada MAX	1.0248
1971 Denmark MIN	7.0665
1971 Denmark MAX	7.5067

# Example 2 ...

**Creates composite key of year + country in map for default hash-based partitioning**

```
public void map(Object key, Text value, Context context  
 ) throws IOException, InterruptedException {  
  
try {  
    //Split columns  
    String[] columns = value.toString().split(comma);  
  
    if ( columns.length<3 || columns[2] == null  
        || columns[2].equals("Value")){  
        return;  
    }  
    //Set FX rate  
    rate.set(Double.parseDouble(columns[2]));  
  
    //Construct key: e.g. 1971 Australia  
    YearCountry.set(columns[0].substring(0, 4) +" "+ columns[1]);  
  
    //Submit value into the Context  
    context.write(YearCountry, rate);  
}  
catch (NumberFormatException ex) {  
    context.write(new Text("ERROR"), new DoubleWritable(0.0d));  
}  
}
```



<key=year country, value=rate>  
<key=year country, value=rate>  
...  
<key=year country, value=rate>

Each map produces a mix of keys  
from input data

can you use a combiner here ? same logic as reducer ?

# Example 2 ...

Creates composite key of year + country in map for default hash-based partitioning

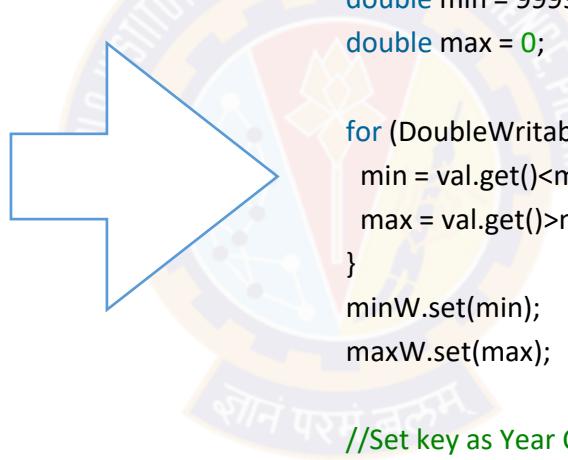
after shuffle-sort using key

```
<key=K1, value=rate>  
<key=K1, value=rate>  
...  
<key=K1, value=rate>
```

```
<key=K2, value=rate>  
<key=K2, value=rate>  
...  
<key=K2, value=rate>
```

#partitions = #keys = #reduce calls

example input to reduce() for a key:  
<key=K1, value=rate1, rate2, ...>



each reduce() call is for a key and value list in partition

```
public void reduce(Text key, Iterable<DoubleWritable>  
values, Context context) throws IOException,  
InterruptedException {  
  
    double min = 999999999999999d;  
    double max = 0;  
  
    for (DoubleWritable val : values) {  
        min = val.get() < min ? val.get() : min;  
        max = val.get() > max ? val.get() : max;  
    }  
    minW.set(min);  
    maxW.set(max);  
  
    //Set key as Year Country Min/Max
```

```
Text minKey = new Text(key.toString() + "+" + "MIN");  
Text maxKey = new Text(key.toString() + "+" + "MAX");
```

```
context.write(minKey, minW);
```

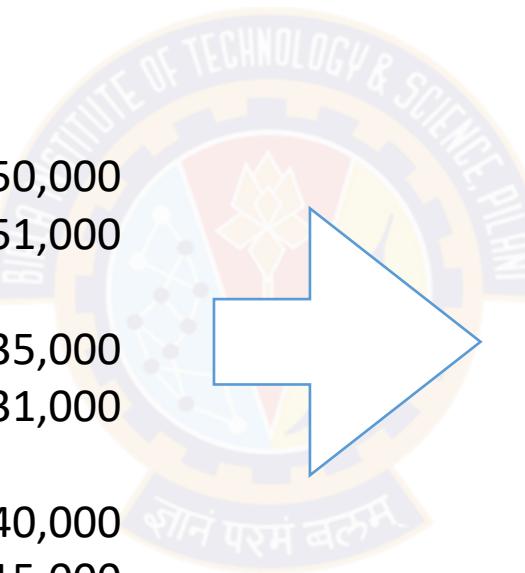
can you use a combiner here ? same logic as reducer ?

set #reducers = 1  
if all results needed in single node else collect later and let partitions run in parallel

# Example 3: Custom partitioning

Find max salary by gender and by age group

1201	gopal	45	Male	50,000
1202	manisha	40	Female	51,000
1203	Priya	34	Female	35,000
1204	prasanth	30	Male	31,000
1205	kiran	20	Male	40,000
1206	laxmi	25	Female	15,000



**Output in Part-00000**

Female	15000
Male	40000

age group 1

**Output in Part-00001**

Female	35000
Male	31000

age group 2

**Output in Part-00002**

Female	51000
Male	50000

age group 3

## Example 3: Custom partitioner instead of composite-key

- Map:
    - ✓ create list of <gender, {age, salary}>  
 $\langle \text{male}, \{45, 97000\} \rangle, \langle \text{female}, \{29, 80000\} \rangle, \dots$
  - Partitioner:
    - ✓ Return partition number as a function of age - create 3 age groups
    - ✓ So partition is not a hash of gender but age group - i.e. not 2 partitions only but 3 partitions created
  - Reduce:
    - ✓ 3 age groups means 3 reducers
    - ✓ A specific age partition across all genders goes to a specific reducer
      - So partition i goes to reducer i
      - In Driver set #reducers = 3
    - ✓ For each reducer, a call to reduce() is made for each gender because records are still <gender, {age, salary}>
      - So 6 reduce() calls across 3 reducers and 2 values for gender key
    - ✓ Each reducer reduce() call finds max() of values in <gender, {salary list}>
    - ✓ So each reducer finds max for each gender within the age group allocated to the reducer

# Example 4: Top N keys given key-value pairs

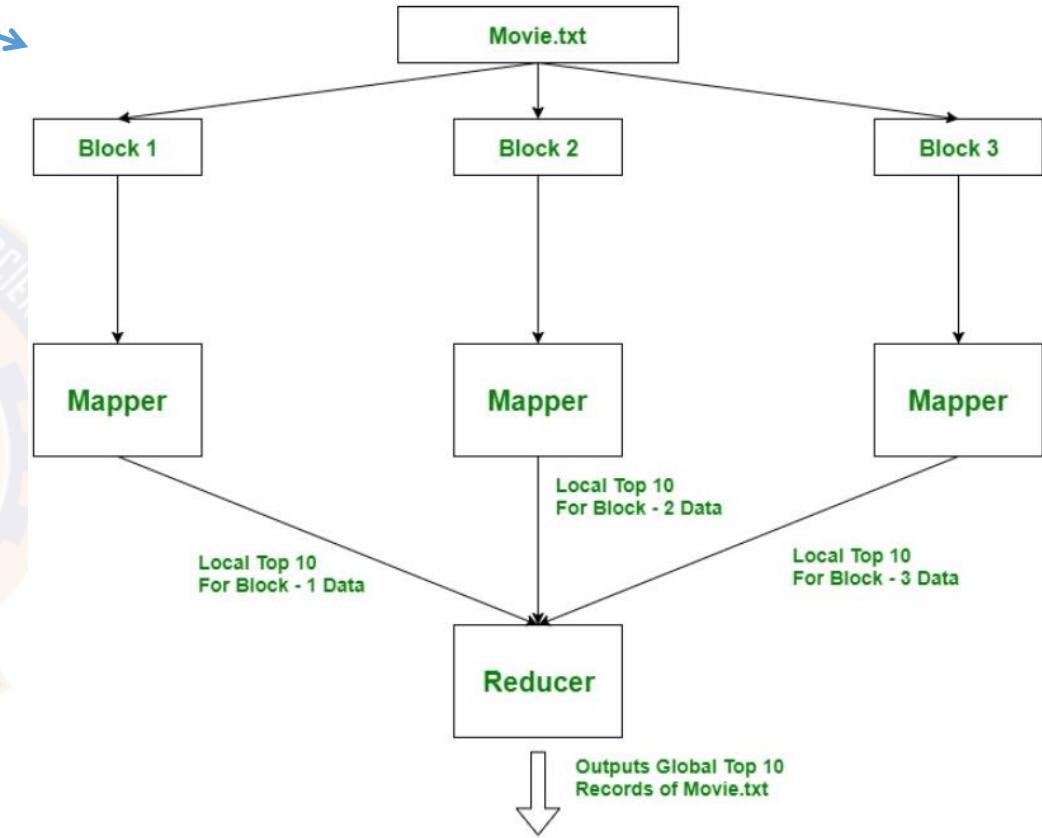
Find the Top 10 movies given <movie\_name,#views>

movie\_name and no\_of\_views

Jumanji (1995)	701
Grumpier Old Men (1995)	478
Waiting to Exhale (1995)	170
Father of the Bride Part II (1995)	296
Heat (1995)	940
Sabrina (1995)	458
Tom and Huck (1995)	68
Sudden Death (1995)	102
GoldenEye (1995)	888
American President, The (1995)	1033
Dracula: Dead and Loving It (1995)	160
Balto (1995)	99
Nixon (1995)	153
Cutthroat Island (1995)	146
Casino (1995)	682
Sense and Sensibility (1995)	835



3428	American Beauty (1999)
2991	Star Wars: Episode IV - A New Hope (1977)
2990	Star Wars: Episode V - The Empire Strikes Back (1980)
2883	Star Wars: Episode VI - Return of the Jedi (1983)
2672	Jurassic Park (1993)
2653	Saving Private Ryan (1998)
2649	Terminator 2: Judgment Day (1991)
2590	Matrix, The (1999)
2583	Back to the Future (1985)
2578	Silence of the Lambs, The (1991)



[Reference link](#)

# Example 4: Top N keys given key-value pairs

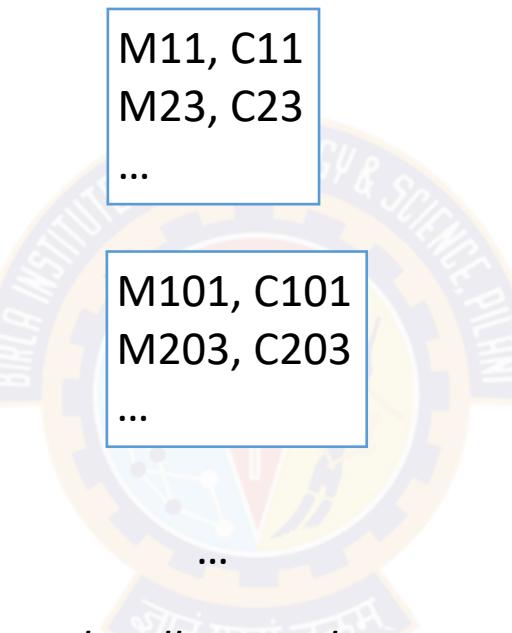
Find the Top N movies given <movie\_name,#views>

```
Mapper class {
    TreeMap tmap

    map( key, value ) {
        get movie and views from value
        tmap.put(value, movie);
        if (tmap.size() > N)
            tmap.remove(tmap.firstKey())

    for each Map :
        get all tmap entries <key, value>
        context.write(key, value)
    }
}
```

Keep a local sorting data structure  
like TreeMap or similar to locally  
sort top N in each map



```
Reducer class {
    TreeMap tmap

    reduce( key, value ) {
        tmap.put(value, key);
        if (tmap.size() > N)
            tmap.remove(tmap.firstKey());

    for each Reduce :
        get all tmap entries <key, value>
        context.write(key, value)
    }
}
```

Set reducer count = 1 in Driver  
Maintain a TreeMap or any other  
sort data structure to keep a global Top N

More efficient than passing all keys to reducer as in Example 1 on sorting.  
We only want top N, so filter top N at map level.

# Example 5

Given a document with several sentences. Each word has a weight calculated based on letters in the word. A letter's weight is based on ASCII code. Find document weight.

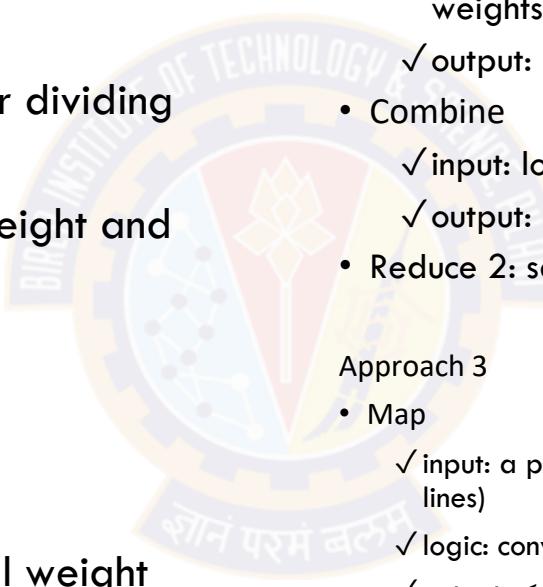
## Approach 1

- Map

- ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)
- ✓ logic: convert into words, calculate word weight and segment weight
- ✓ output: <segment id, weight>

- Reduce (set reducer count to 1)

- ✓ input: <segment id>, <weight>
- ✓ logic: keep a class variable to add up total weight across segments
- ✓ output: <\_, total weight>



## Approach 2

- Map

- ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)
- ✓ logic: convert into words, calculate word weight and emit word weights
- ✓ output: <word, weight>

- Combine

- ✓ input: local pairs <word, weight>
- ✓ output: <segment id, weight>

- Reduce 2: same as reduce 1

## Approach 3

- Map

- ✓ input: a part of a document (write a splitter dividing document in chunks every N lines)

- ✓ logic: convert into words, calculate word weight and emit word weights
- ✓ output: <word, weight>

- Combine (Same as reducer)

- ✓ input: local pairs <word, weight>
- ✓ output: <word, total weight across instances>

- Reduce: (set reducer count to 1)

- ✓ calculate total weight across all words using a variable in reducer class

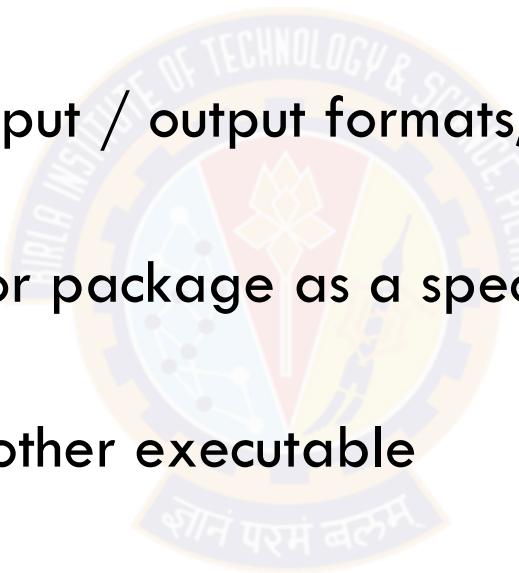
# Advanced examples

- Secondary sorting
  - ✓ <http://www.javamakeuse.com/2016/04/secondary-sort-example-in-hadoop-mapreduce.html>
- Iterative MR - k-means
  - ✓ <https://github.com/thomasjungblut/mapreduce-kmeans/tree/master/src/de/jungblut/clustering/mapreduce>



# Hadoop streaming

- Enables to run any executable as map reduce tasks
- Creates map / reduce tasks from the submitted code and monitor progress till completion
- One can override defaults of input / output formats, partitioners, combiners etc. with own implementations
- One can also use an aggregator package as a special reducer that supports max, min, count etc.
- Use to run python code or any other executable



# Hadoop streaming - Example

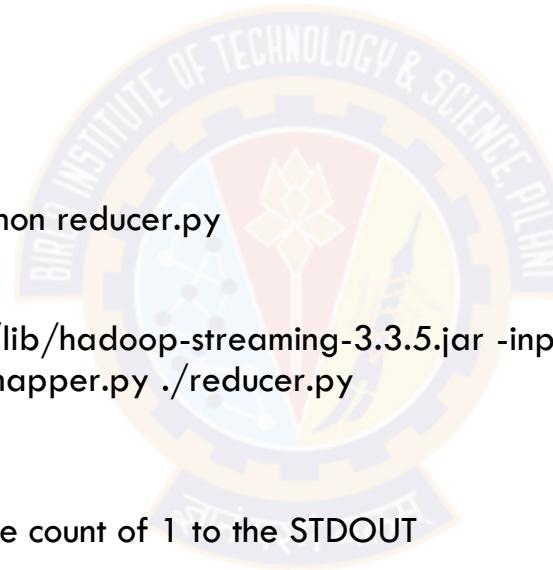
- Hadoop Streaming allows developers to use various languages for writing MapReduce programs
- It supports all the languages that can read from standard input and write to standard output
  - Python
  - C++
  - Ruby, etc

## 1. Manual testing of streams

```
cat sample.txt | python mapper.py | sort -k1,1 | python reducer.py
```

## 2. Running the program with hadoop streaming

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar -input /sample.txt -output /myoutput -mapper "python ./mapper.py" -reducer "python ./reducer.py" -file ./mapper.py ./reducer.py
```



### Mapper.py

Loop over the words array and print the word with the count of 1 to the STDOUT

```
words = line.split()  
for word in words:  
    print('%s\t%s' % (word, 1))
```

### Reducer.py

Reads from STDIN and outputs unique words with count to STDOUT

<https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/>

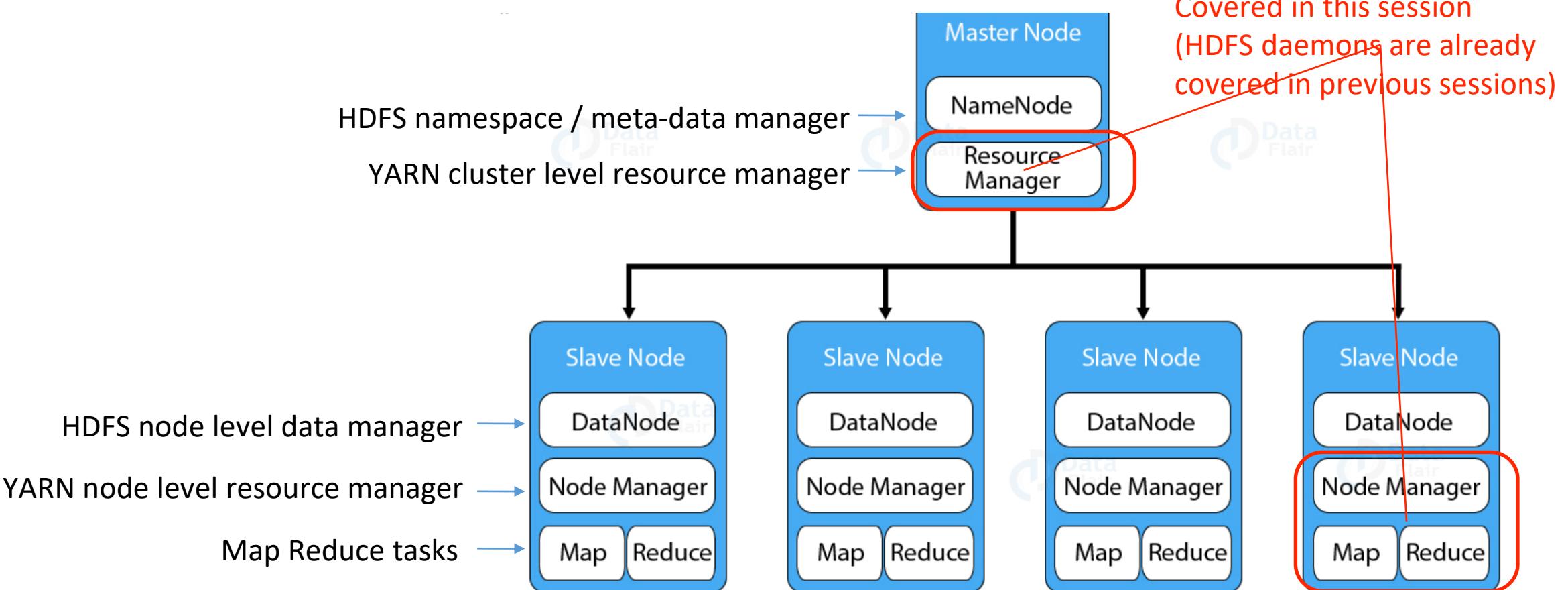
# Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands



# Hadoop 2 - Architecture

- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



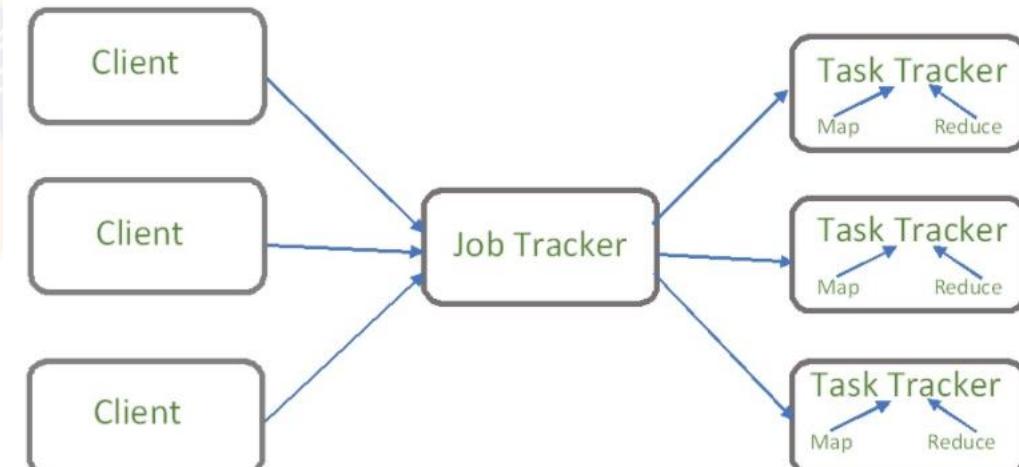
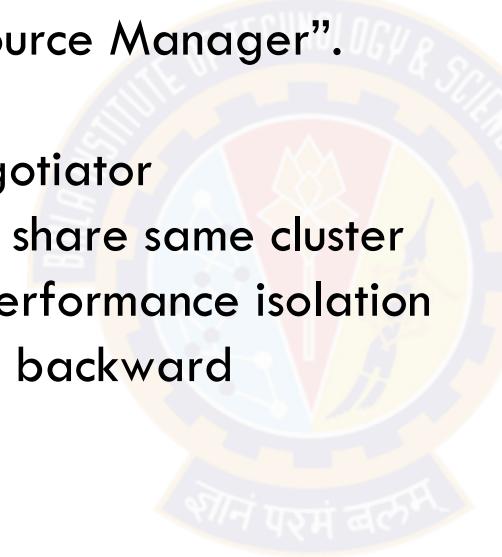
**Note:** YARN Resource Manager also uses application level App Master processes on slave nodes for application specific resource management

Yet Another resource Negotiator (YARN) was introduced in Hadoop 2.0 to make Hadoop scalable

- YARN is being used as a general purpose distributed computing framework
- It partitions system resources into containers and launches tasks in them
- Number of concurrently running tasks depends on the fraction of system resources allocated to the containers
- Large volume data is partitioned into chunks of equal sizes and tasks are assigned to each chunk
- YARN allows multi-tenant applications to run on the same cluster

# Changes from Hadoop 1 - Why YARN ?

- In Hadoop 1: MapReduce included resource management with JobTracker on Master and TaskTrackers on slaves
- Hadoop 2: The resource management was decoupled from MapReduce data processing and the daemons were refactored to have a “redesigned Resource Manager”.
- Result
  - ✓ YARN - Yet Another Resource Negotiator
- YARN enables multiple applications to share same cluster and not require separate clusters for performance isolation
- Can run Hadoop 1 jobs on Hadoop2 - backward compatibility

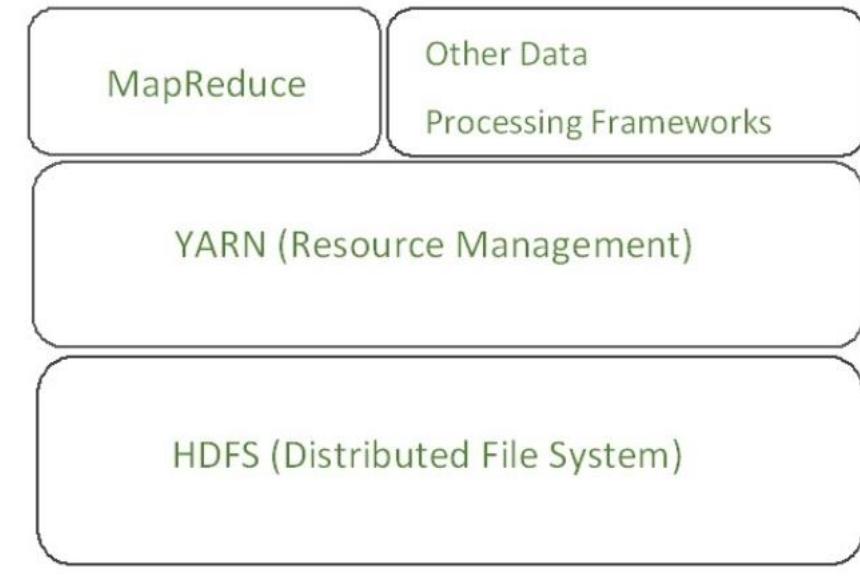
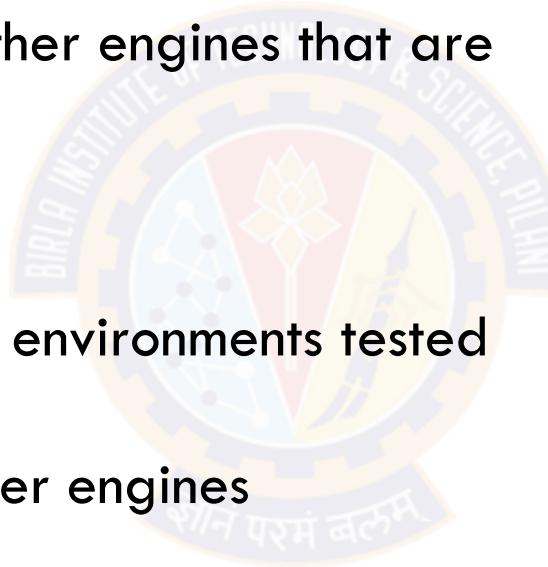


<https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

<https://blog.cloudera.com/apache-hadoop-yarn-concepts-and-applications/>

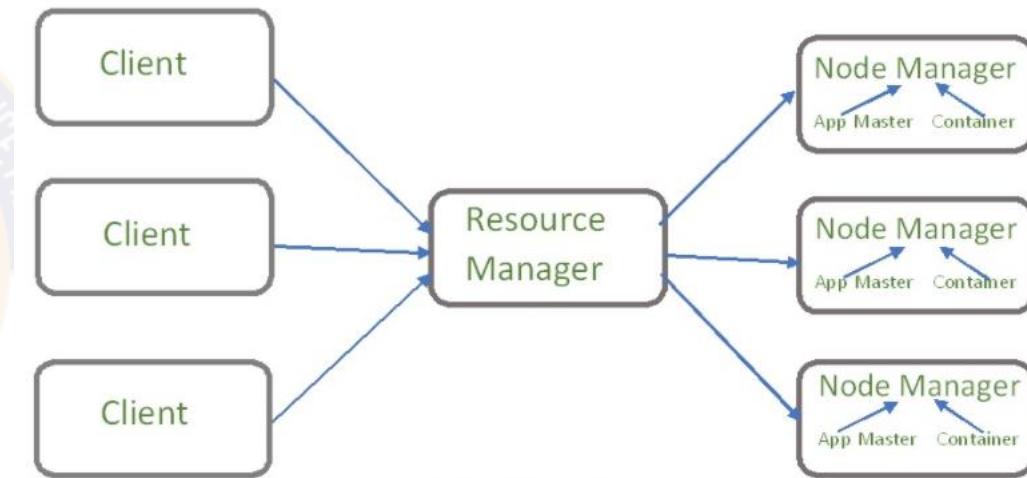
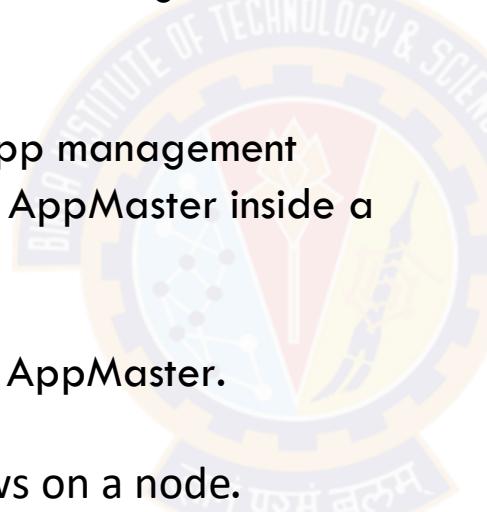
# Where does YARN fit in Hadoop ?

- YARN enables MapReduce and other data processing logic to run on the same Hadoop cluster using HDFS or other file systems.
- So now Hadoop can be used by other engines that are not batch processing
  - ✓ Graph, stream, interactive ...
- YARN provides
  - ✓ Scale across 1000s of nodes - environments tested with 10K+ nodes
  - ✓ Compatibility with MR and other engines
  - ✓ Dynamic cluster utilisation
  - ✓ Multi-tenancy across various processing engines



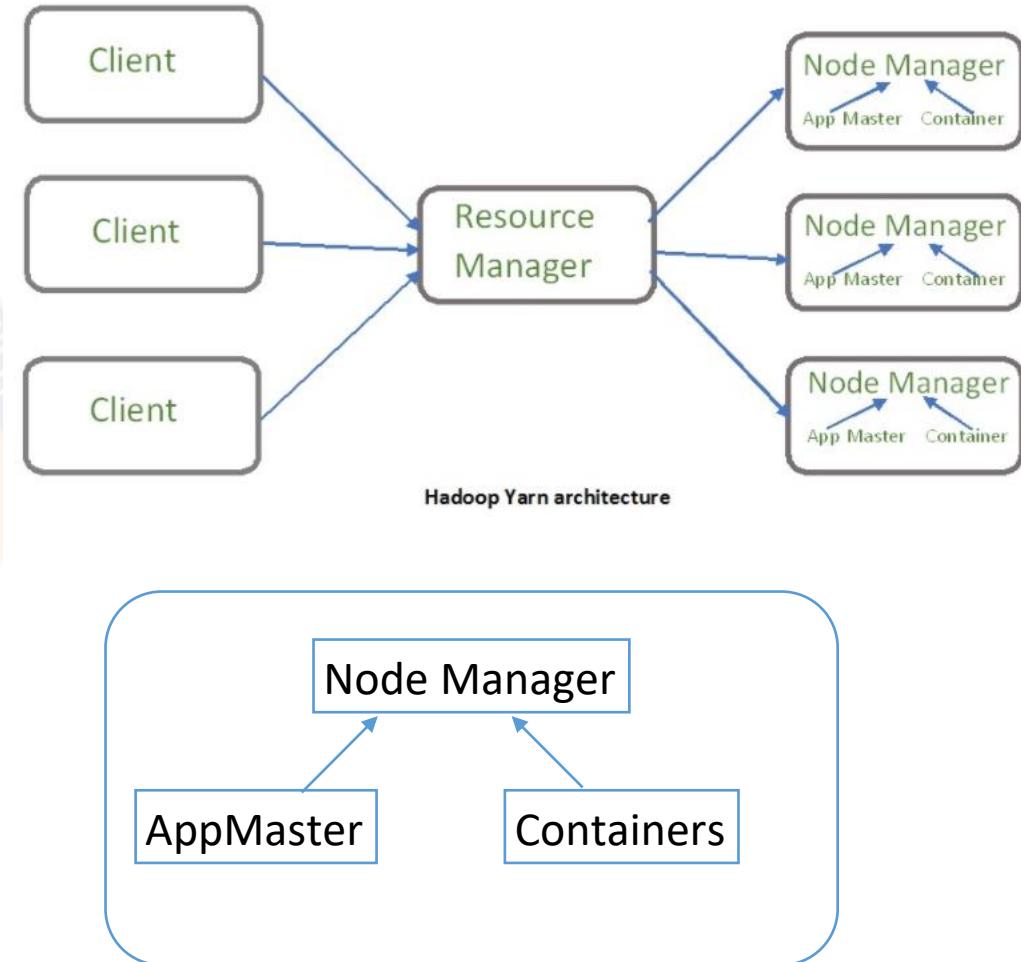
# YARN components (1)

- Client
  - ✓ Submits MR jobs
- Resource manager
  - ✓ Key role is to schedule resources in the cluster
  - ✓ Takes request from client and talks to Node Managers for allocation
  - ✓ 2 components:
    - Scheduler: key function
    - App Manager: Master component of app management
      - Accepts job request and sets up an AppMaster inside a container for each job on a slave.
      - Restarts the AppMaster on failure.
      - Main App specific work is done by AppMaster.
- Node Manager
  - ✓ Takes care of management and workflows on a node.
  - ✓ Creating, killing containers, monitoring usage, log management



# YARN components (2)

- AppMaster
  - ✓ Negotiates resources from Resource Manager per application for starting containers on nodes
  - ✓ Sends periodic health status of application containers and tracks progress
  - ✓ Talks via Node Manager for updates and usage reports to Resource Manager
  - ✓ Clients can directly talk to AppMaster
- Container
  - ✓ CPU, Memory, Storage resources on a node
  - ✓ Container Launch Context (CLC) - data structure that contains resource, security tokens, dependencies, environment vars



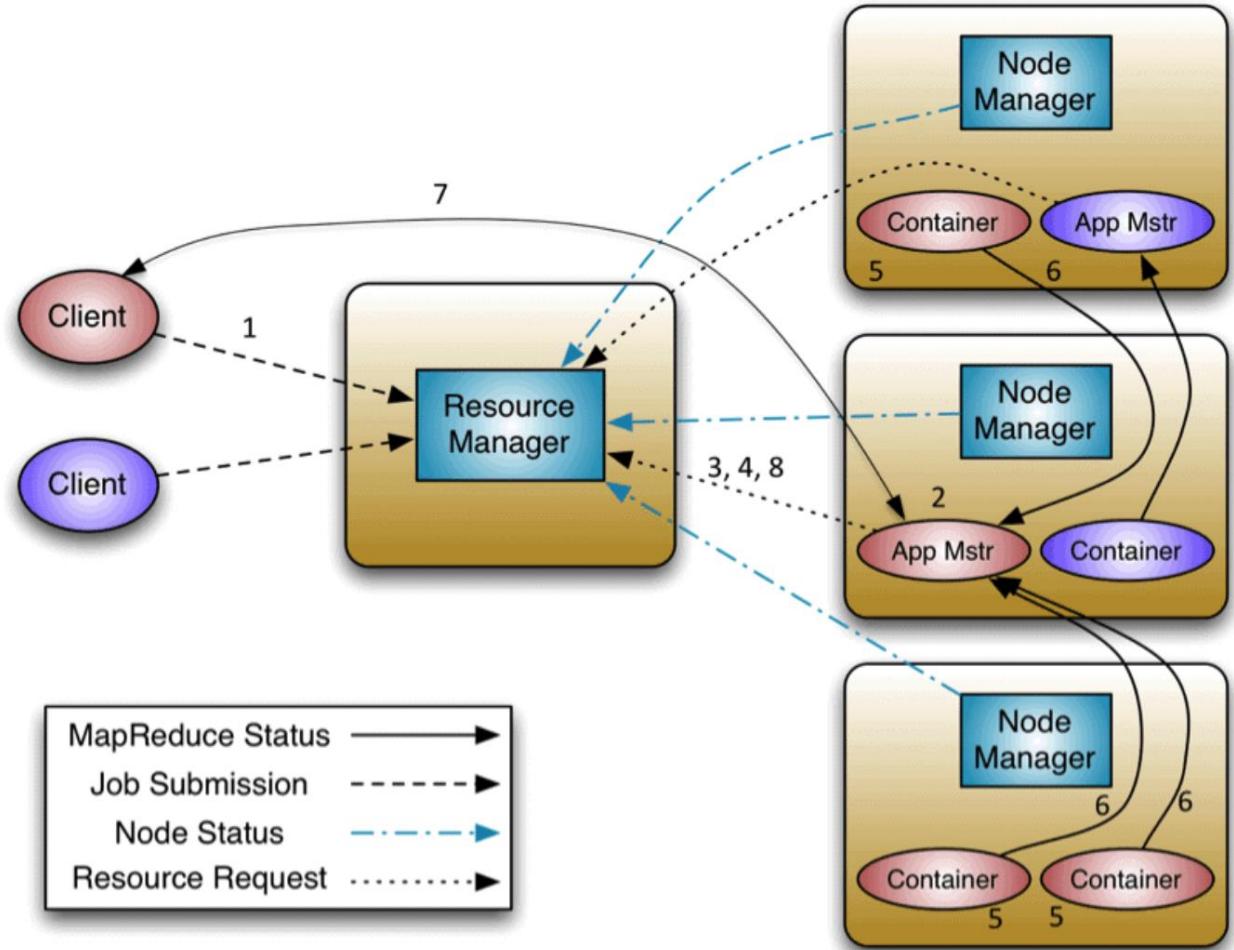
# Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ **Workflow (Sec2)**
  - ✓ Resource model and implementation
  - ✓ Resource scheduling
  - ✓ YARN sample commands



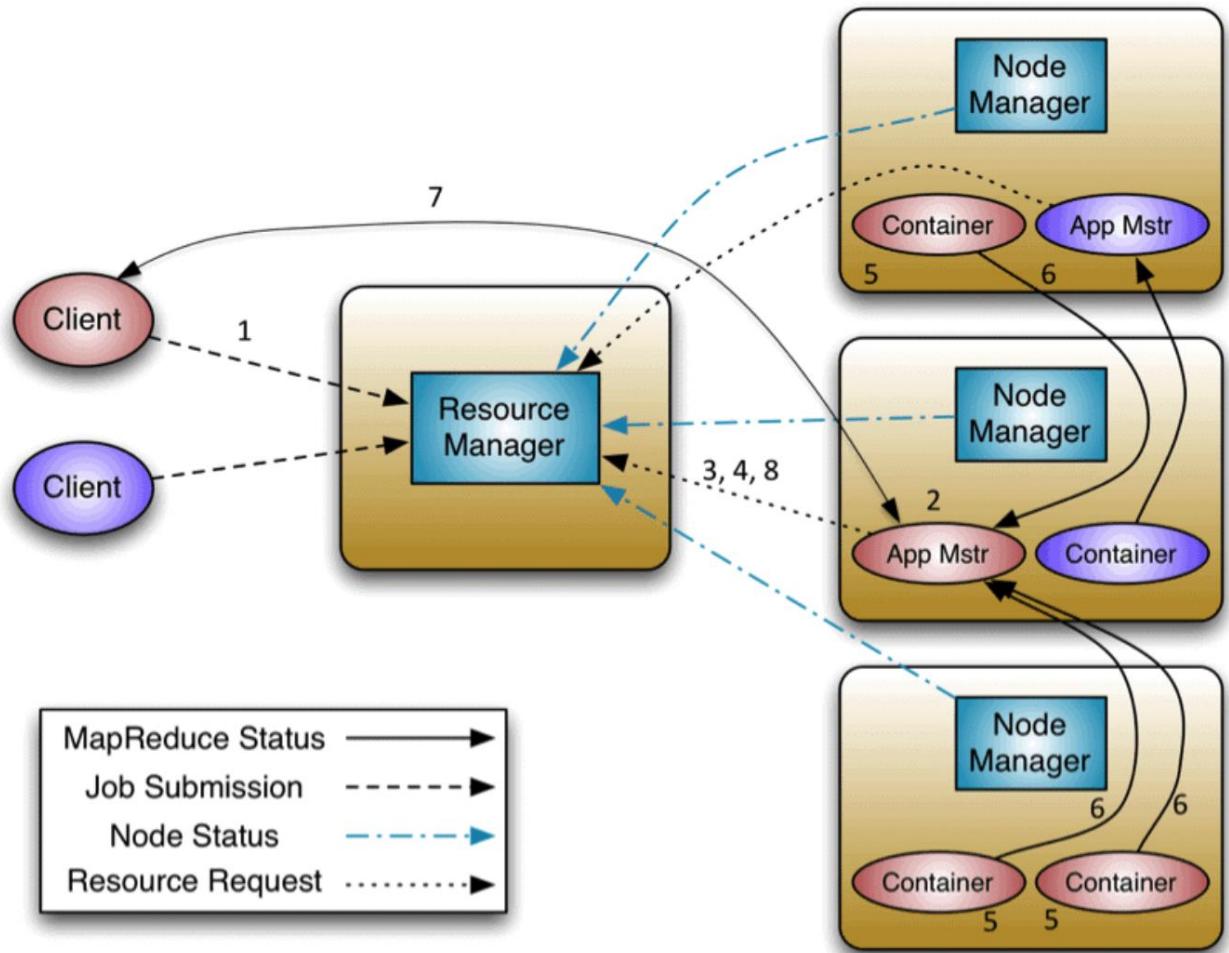
# YARN workflow (1)

1. A client program *submits* the application / job with specs to start AppMaster
2. The ResourceManager asks a NodeManager to start a container which can host the ApplicationMaster and then launches ApplicationMaster.
3. The ApplicationMaster on start-up registers with ResourceManager. So now the client can contact the ApplicationMaster directly also for application specific details.



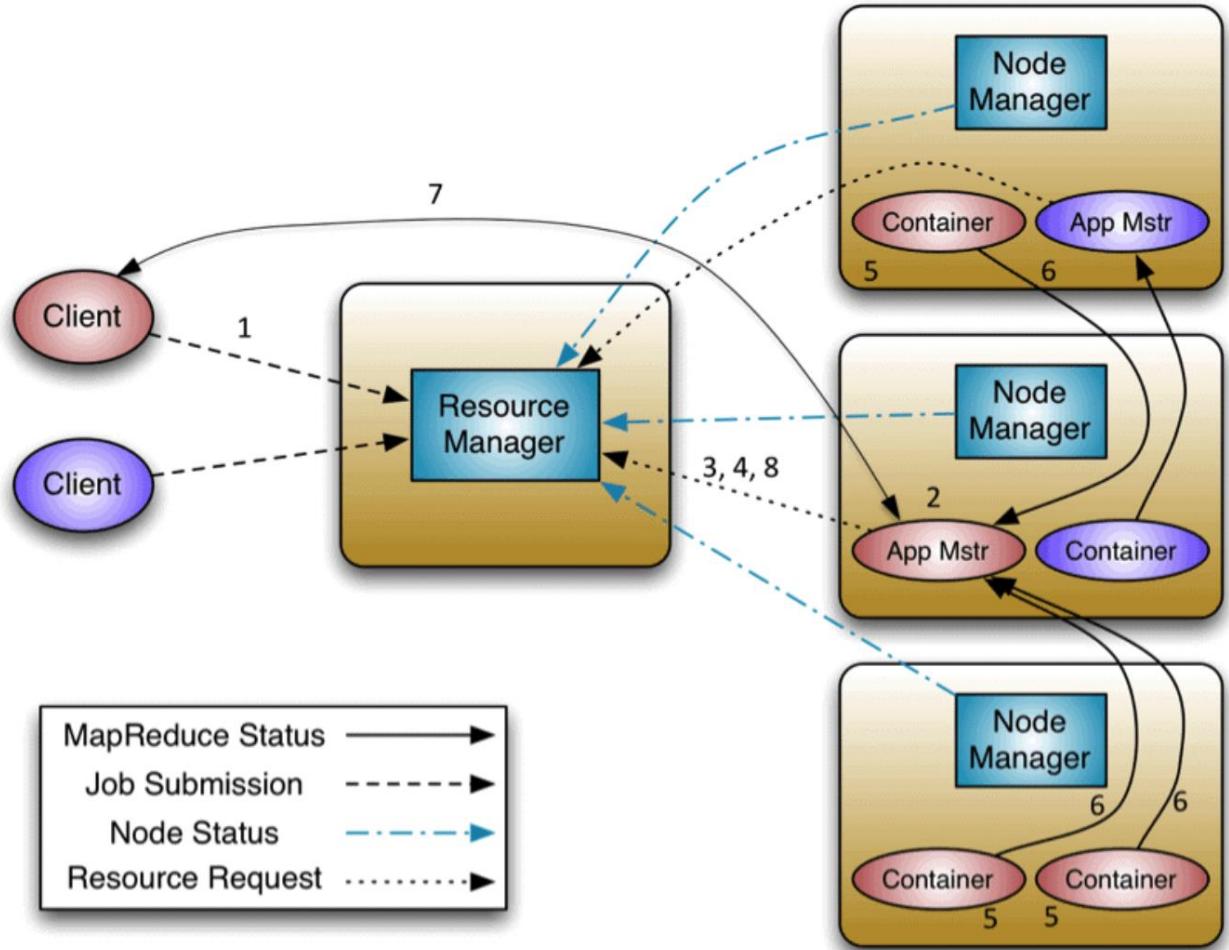
# YARN workflow (2)

4. As the application executes, the AppMaster negotiates resources in the form of containers via the resource request protocol involving the ResourceManager.
5. As a container is allocated successfully for an application, the AppMaster works with the NodeManager on same or diff node to launch the container as per the container spec. The spec involves how the AppMaster can communicate with the container.
6. The app specific code inside container provides runtime information to the AppMaster for progress, status etc. via application-specific protocol.

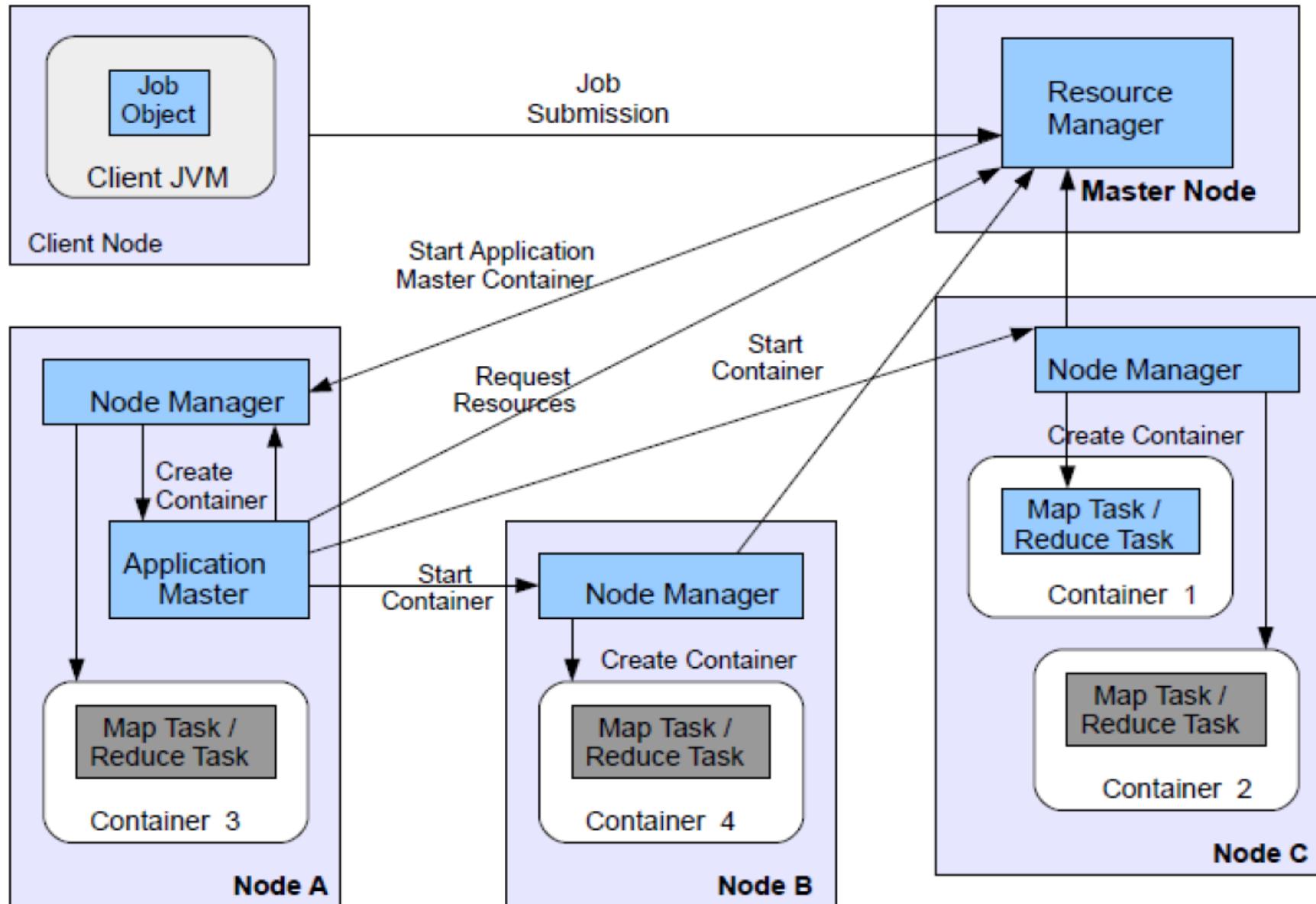


# YARN workflow (3)

7. The client that submitted the app / job can directly communicate with the AppMaster for progress, status updates. via the application specific protocol.
8. On completion of the app / job, the AppMaster de-registers from ResourceManager and shuts down. So the containers allocated can be repurposed.



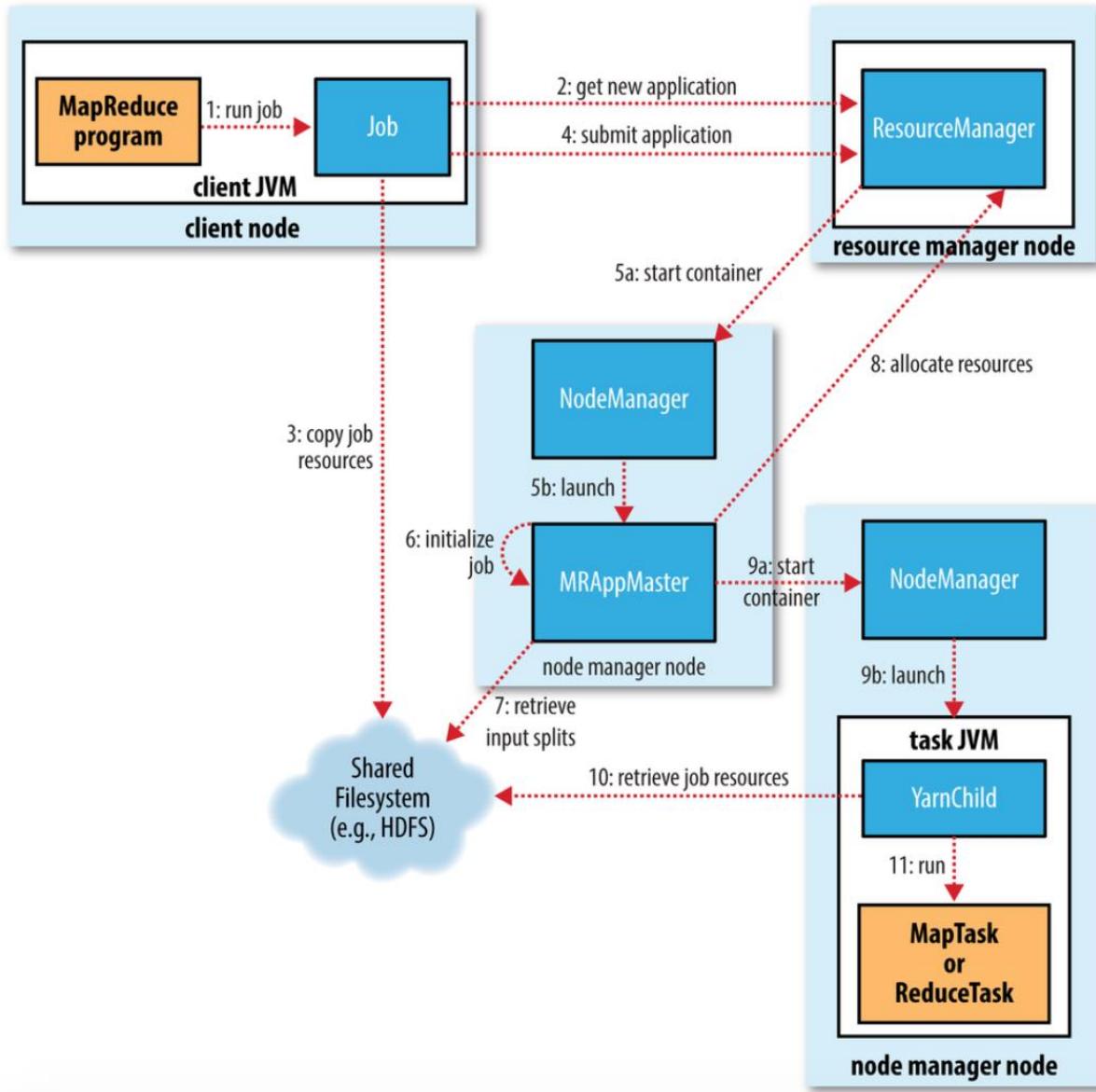
# Running a Job on YARN – Container management



# More about AppMaster

- Consider it as a framework specific library that is installed on a special / first container of the application by the Resource Manager
- Responsible for all resource requests from an application perspective
- Is user specific and Resource Manager needs to protect itself / cluster from malicious resource use
  - ✓ This enables RM to be just a scheduler while application needs, tracking / monitoring etc. is left to AppMaster (as application rep) and NodeManager (as Hadoop rep)
  - ✓ With all app specific code moved out of Resource Manager, cluster can be used for multiple data processing engines
- So, resource management can scale to 10K+ nodes while AppMaster doesn't become a cluster-wide bottleneck because it only manages a job / application
- It is possible to have an AppMaster instance manage multiple applications - it is user specific code

# Job submission flow



<https://stackoverflow.com/questions/34709213/hadoop-how-job-is-send-to-master-and-to-nodes-on-mapreduce>

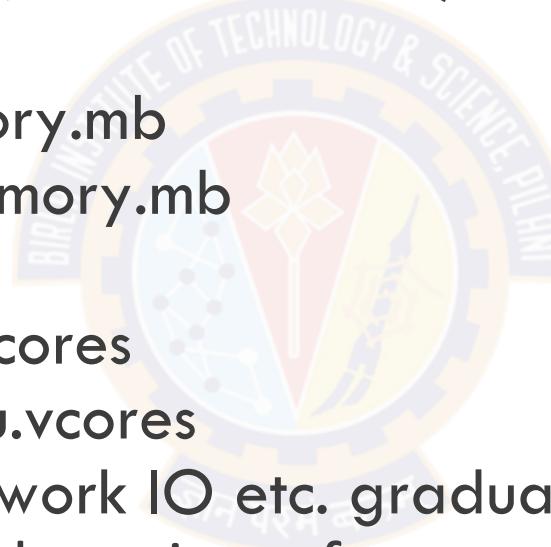
# Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ **Resource model and implementation (Sec1)**
  - ✓ Resource scheduling
  - ✓ YARN sample commands



# YARN Resource model

- Resource for an application is a set of containers.
- Each container is defined by :
  - Resource-name (hostname, rack name etc.)
  - Memory (in MB)
    - mapreduce.map.memory.mb
    - mapreduce.reduce.memory.mb
  - CPU (cores)
    - mapreduce.map.cpu.vcores
    - mapreduce.reduce.cpu.vcores
  - Possibly adding - Disk, network IO etc. gradually
- Resource Manager has complete view of resources across nodes to schedule a new request
- <http://192.168.1.36:8088>



# What are vCores

- VCore is the abbreviation for Virtual Cores and is a type of resource
- Vcores help sharing usage of host CPU in a Hadoop cluster
- Vcore indicates ‘usage share of a CPU core’
- Vcores are configured by YARN in such a way that all resources available in the cluster are utilized in the best possible way.
- Vcore to Physical core ratio is decided by the nature of the workload
  - ✓ For CPU intensive applications, this ratio is 1, ie 1 Vcore per physical core
  - ✓ For IO intensive application, this ratio  $\geq 4$

# Configuring vcore allocation

Name	Description
mapreduce.map.cpu.vcores	The number of virtual cores required for each map task.
mapreduce.reduce.cpu.vcores	The number of virtual cores required for each reduce task.
yarn.app.mapreduce.am.resource.cpu-vcores	The number of virtual CPU cores the MR AppMaster needs.
yarn.scheduler.maximum-allocation-vcores	The maximum allocation for every container request at the RM, in terms of virtual CPU cores. Requests higher than this won't take effect, and will get capped to this value.
yarn.scheduler.minimum-allocation-vcores	The minimum allocation for every container request at the RM, in terms of virtual CPU cores. Requests lower than this won't take effect, and the specified value will get allocated the minimum.
yarn.nodemanager.resource.cpu-vcores	Number of CPU cores that can be allocated for containers.

# YARN Tuning and Configuration

## Tuning

- A YARN cluster is composed of host machines.
- Hosts provide memory and CPU resources.
- A vcore (virtual core) is a usage share of a host CPU.
- Yarn containers are execution engines constituted by vcores and memory
- Tuning YARN consists of optimally defining containers on worker hosts
- Tune YARN hosts to optimize the use of vcores and memory

## Configuration

- YARN and MapReduce have many configurable properties.
- The YARN tuning spreadsheet ([yarn-tuning-guide.xlsx](#)) lists the essential subset of these properties that are most likely to improve performance for common MapReduce applications
- (Watch this video before using the spreadsheet - <https://youtu.be/lykWFhrGvJ4>)
- YARN tuning has three phases. (The phases correspond to the tabs in the YARN tuning spreadsheet)
  1. Cluster configuration, where you configure your hosts.
  2. YARN configuration, where you quantify memory and vcores.
  3. MapReduce configuration, allocates minimum and maximum resources for map and reduce tasks.

[https://docs.cloudera.com/documentation/enterprise/latest/topics/cdh\\_ig\\_yarn\\_tuning.html](https://docs.cloudera.com/documentation/enterprise/latest/topics/cdh_ig_yarn_tuning.html)

# Resource requests

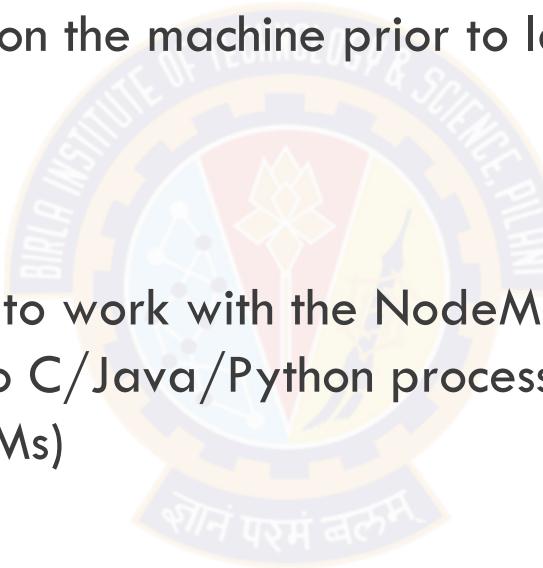
- A resource request :  
`<resource-name, priority, resource-requirement, number-of-containers>`  
resource-name is a host, rack or \* for a container  
priority is within application for this request  
resource-requirement is CPU, memory etc. needed by a container  
number-of-containers is count of above spec containers needed by application
- One or more containers is the result of a successful allocation request. It is a “right” given to an application to use certain amount of resources on a specific host / node.
- AppMaster presents that “right” to the Node Manager on a host to use resources. Security and verification is done by NodeManager.

# What can be launched in a Container ?

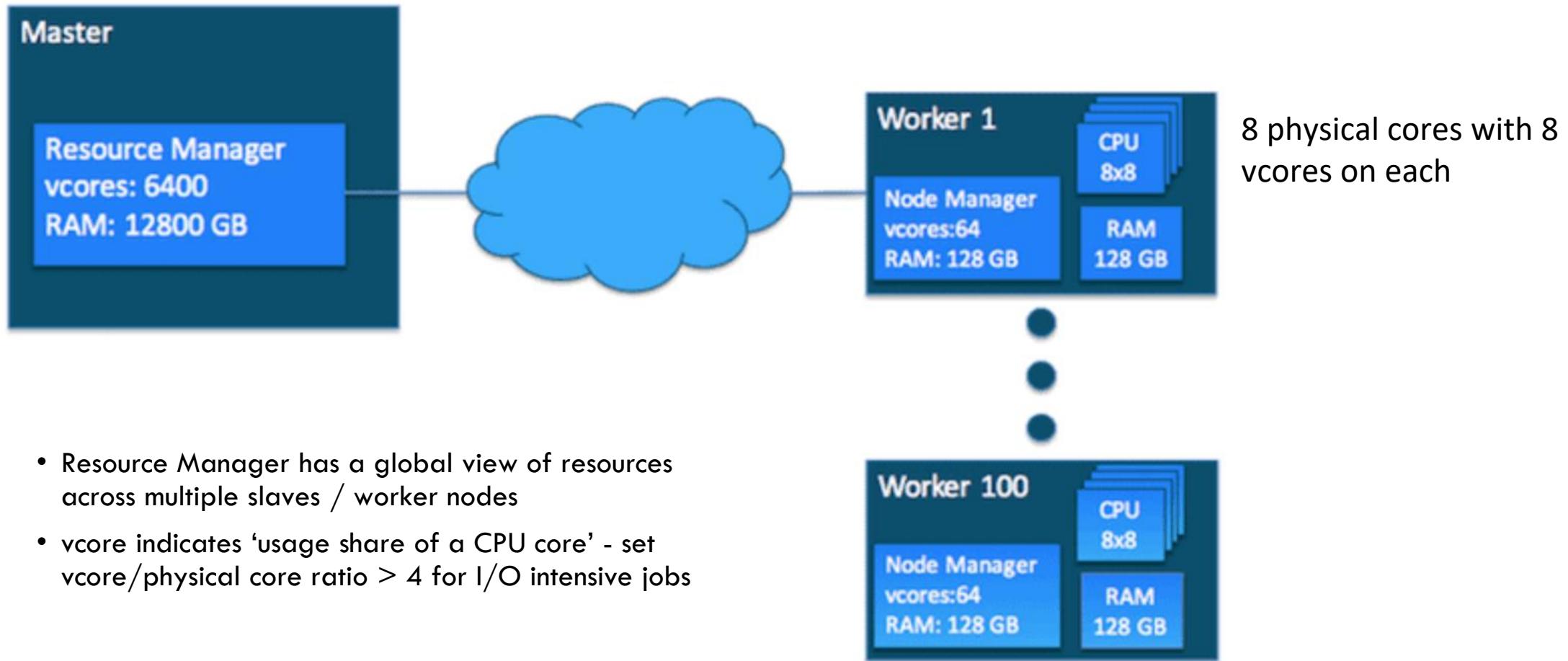
The YARN Container launch specification API is platform agnostic and contains:

- Command line to launch the process within the container
- Environment variables
- Local resources necessary on the machine prior to launch, such as jars, shared-objects, auxiliary data files etc.
- Security-related tokens.

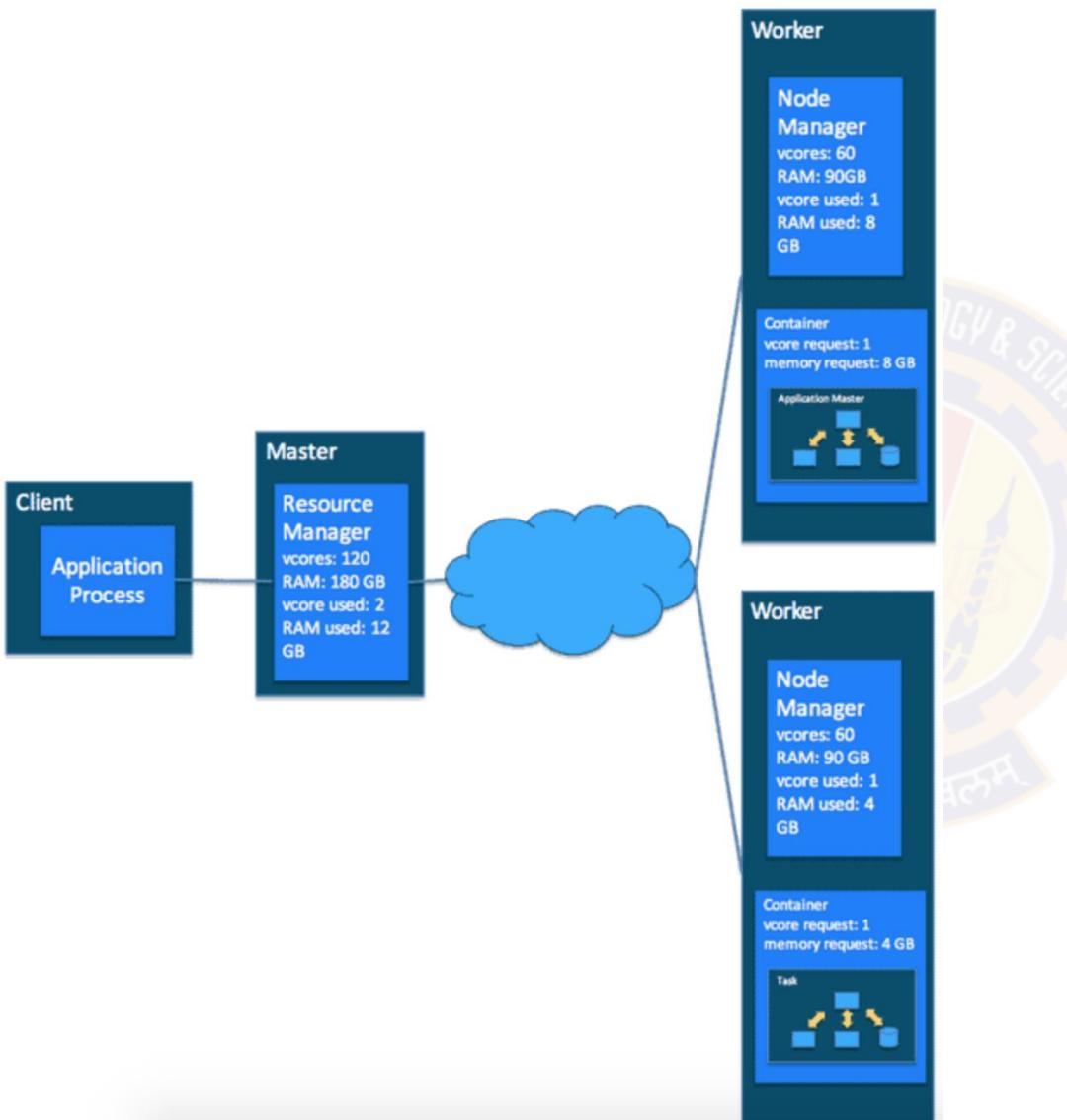
This allows the ApplicationMaster to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged virtual machines (e.g. KVMs)



# Example: Global resource view

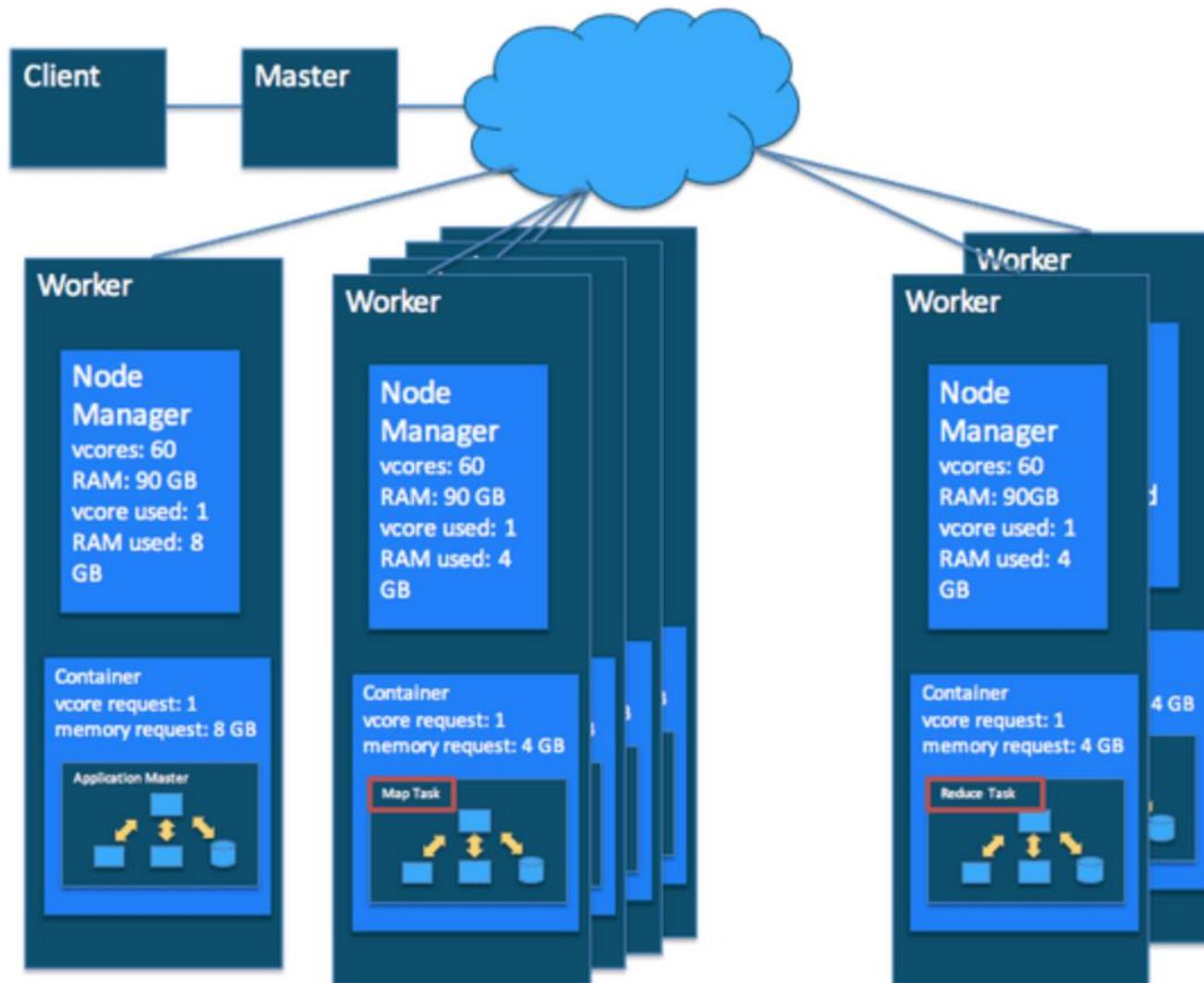


# Example: Containers on Workers



- Containers are allocated specific to application / job
- First container for an application has the Application Master
- Other containers to run tasks are negotiated by Application Master

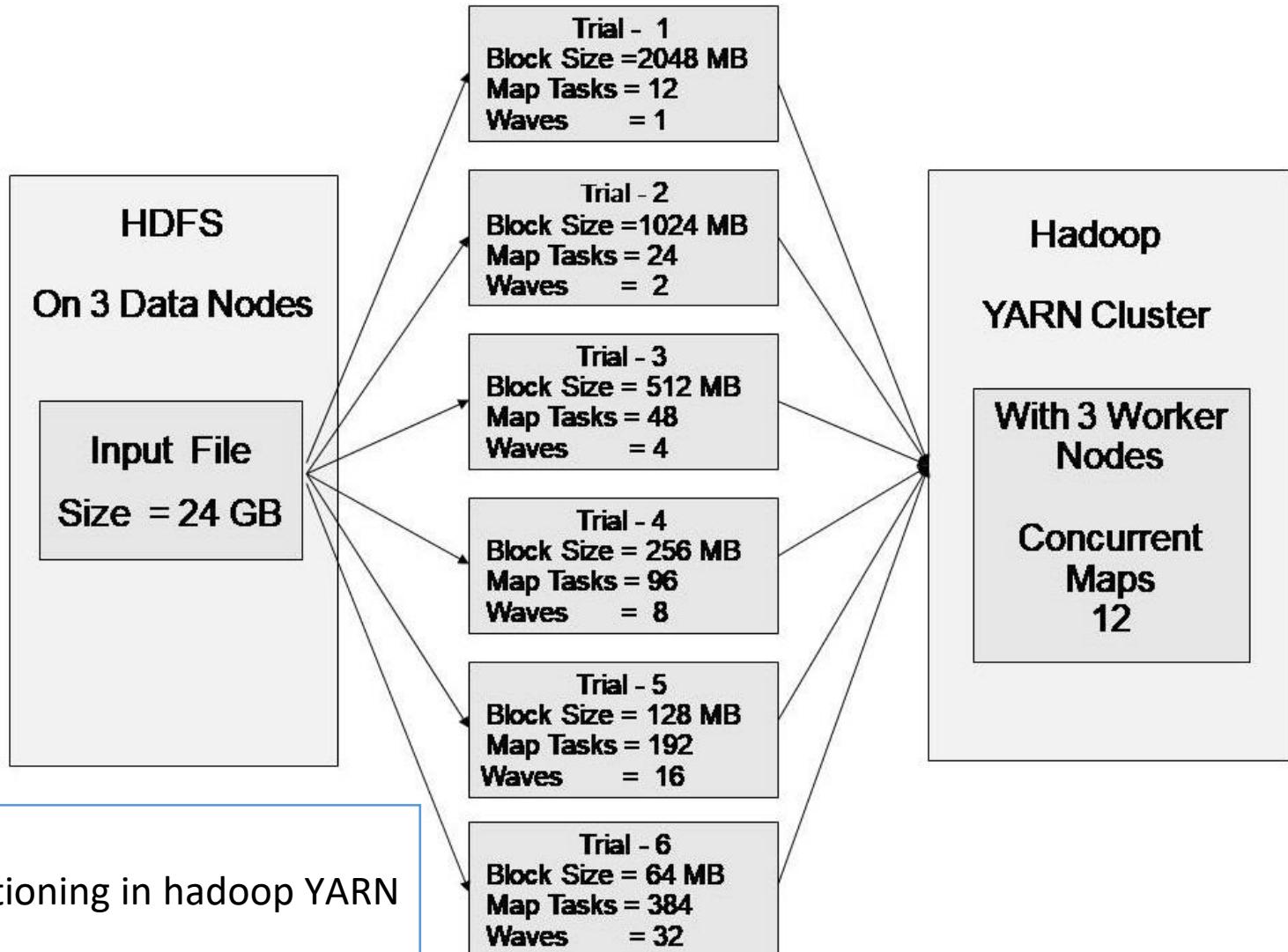
# Example: Map and Reduce tasks in containers



- Application specific tasks are started within the containers

# Block Size, Map tasks, Waves of execution

- No of concurrent containers depends on portion of the total resources allocated to it
- When no of map tasks > no of concurrent containers, map tasks executes in stages
- With containers having higher resource allocation, no of concurrent containers will be less
- With containers having less resource allocation, no of concurrent containers will be more..
- To guarantee maximum performance with minimum resources, we need to configure the YARN framework for optimum concurrency level in execution of containers.



Reference Paper:

Study of execution parallelism by resource partitioning in hadoop YARN

<http://ieeexplore.ieee.org/document/8126000/>

## Concurrent maps on 3 nodes– File Size 9GB, Block size 256Mb

No.	Concurrent maps	Tasks on nodes	Waves of execution	No. of vcores for maps	Memory in MB for map tasks	Cluster Capacity used
1	3	1	12	44	12288	94.90%
2	6	2	6	22	6144	94.90%
3	9	3	4	15	4096	94.90%
4	12	4	3	11	3072	94.90%
5	18	6	2	6	1792	94.90%
6	36	12	1	3	1024	97.90%

# Topics for today

- Hadoop MapReduce
  - ✓ MapReduce runtime
  - ✓ More examples
  - ✓ Hadoop streaming
- Yet Another Resource Negotiator (YARN)
  - ✓ Architectural components
  - ✓ Workflow
  - ✓ Resource model and implementation
  - ✓ **Resource scheduling**
  - ✓ YARN sample commands

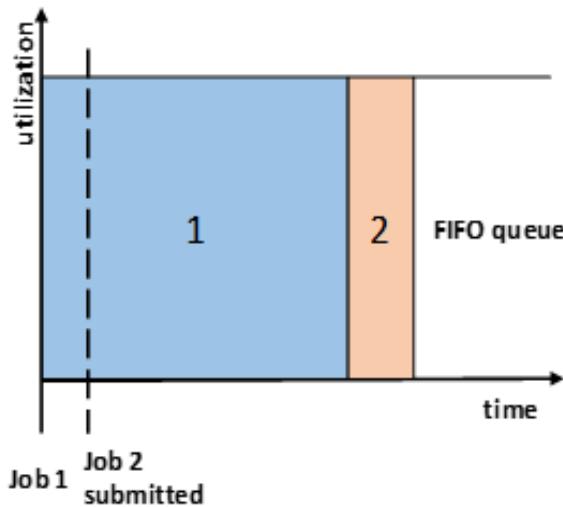


# Schedulers in YARN

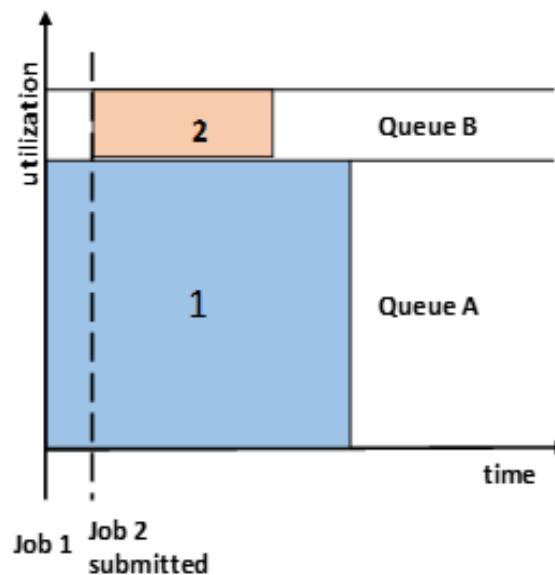
- FIFO – First in, First Out
- Capacity - Maintains separate queue for different types of jobs (Default on Hadoop-3.3.5)
- Fair share scheduler (Fair scheduler) - Dynamically balances resources into all accepted jobs

**These are pluggable to YARN, Any one at a time**

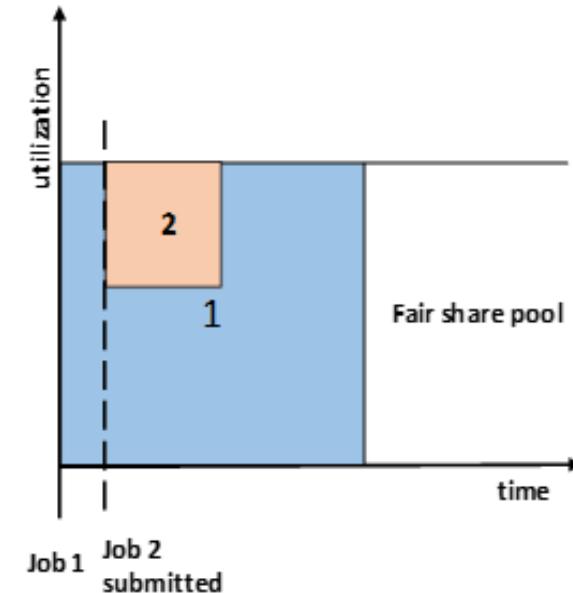
Source: [Hadoop: The Definitive Guide](#)



(a) FIFO scheduler



(b) Capacity Scheduler

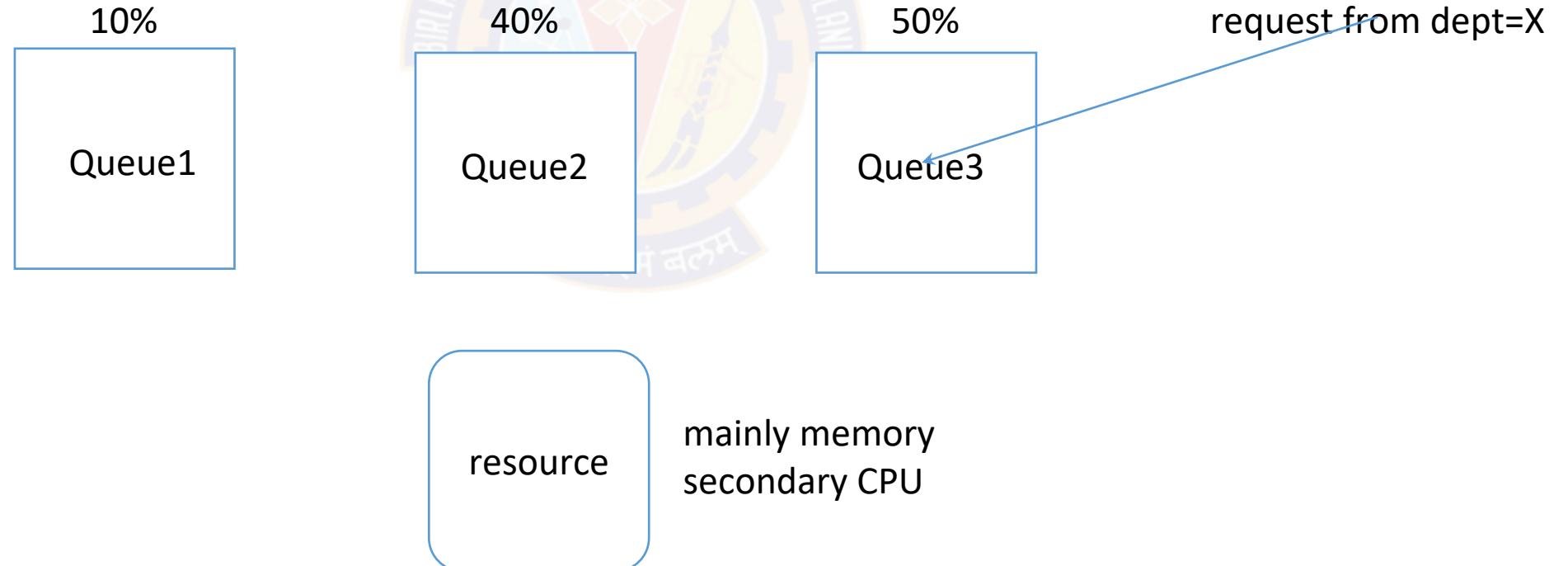


(c) Fair Scheduler

Figure 1: YARN Schedulers' cluster utilization vs. time

# Resource Manager scheduling - Capacity scheduler

- When container requests are made, which request do you satisfy ?
  - ✓ So put a queue where a request is entered and a scheduler will pick requests from the queue
- Typically, multiple queues with different shares of resources (by capacity) because you want to partition system resources by some criteria, e.g. organisation / app type etc.
- `hadoop queue [-list] | [-info <job-queue-name> [-showJobs]]`
- `Mapred queue -list`



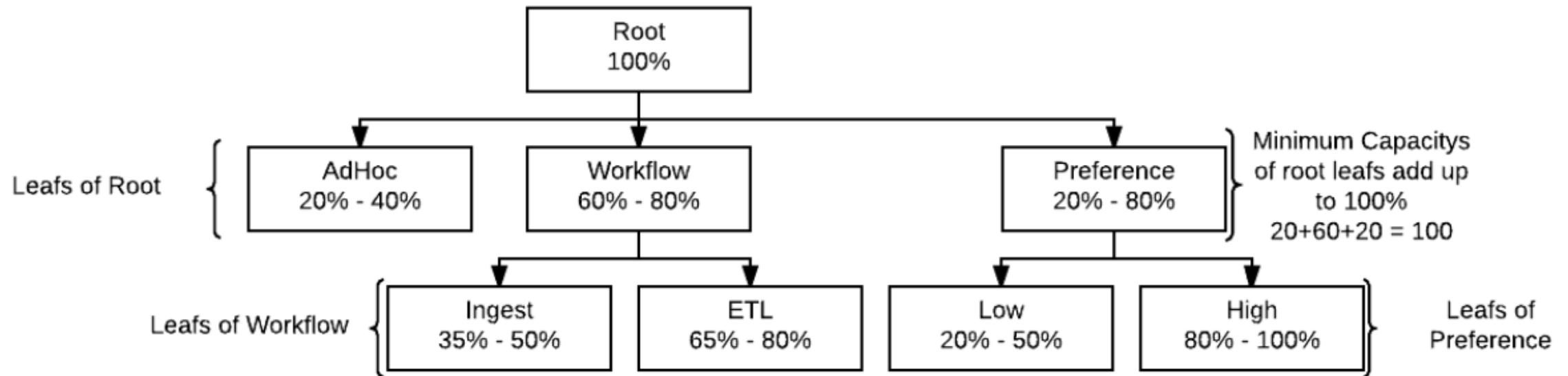
# Resource Manager scheduling - Capacity scheduler

- Multiple organisations can share a cluster
  - ✓ Individual org level capacity reservations are strictly met with isolation to have multi-tenancy
- Implements hierarchical queues where first level is between orgs and second level queue is within an org
  - ✓ Enables capacity to be first shared by users within org before any spare capacity (from set limits) is used by other orgs
- Security done by ACLs for job submission in relevant queues
- Elasticity to use spare capacity with pre-emption capability to stop jobs when higher priority jobs come in or the spare capacity is no longer available
- Configurable scheduling
  - ✓ Resource based scheduling for applications that use some resource more, e.g. memory intensive
  - ✓ Users can map jobs to specific queues and define placement rules, e.g. user/group/app can determine where the resources are allocated
  - ✓ Priority scheduling - higher value means high priority
  - ✓ Applications can ask for absolute value of resources

```
yarn.resourcemanager.scheduler.class = org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler
```

# Capacity scheduler: Hierarchical queue allocations

- Min capacity is what needs to be given to a queue even when cluster is at max usage
- Max capacity is an elastic like capacity that allows queues to make use of resources which are not being used to fill minimum capacity demand in other queues.
- e.g. Root->Preference->Low will get min 20% of the 20% min share of Root->Preference



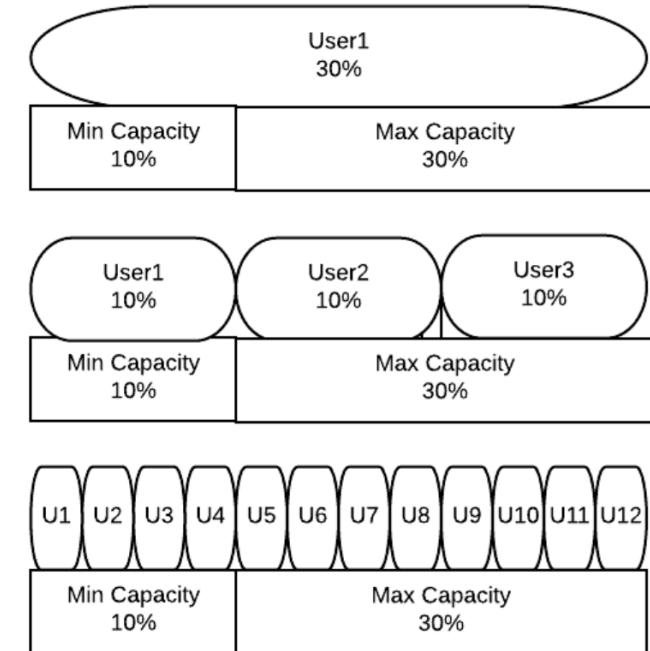
# Capacity scheduler : User level allocations

- Min user percentage : Smallest amount of resources a single user can get access to - varies between min and max of queue capacity.
- User limit factor: Used to control max resources given to a user as a factor of min user percentage.
- Note: Be aware of containers that are long lived vs short lived. Latter (high container churn) helps to balance queues better. The former use limit factors, pre-emption, or even dedicated queues without elastic capacity.

Queue A  
User Limit Factor = 3

Queue B  
User Limit Factor = 1

Queue C  
User Limit Factor = 0.25

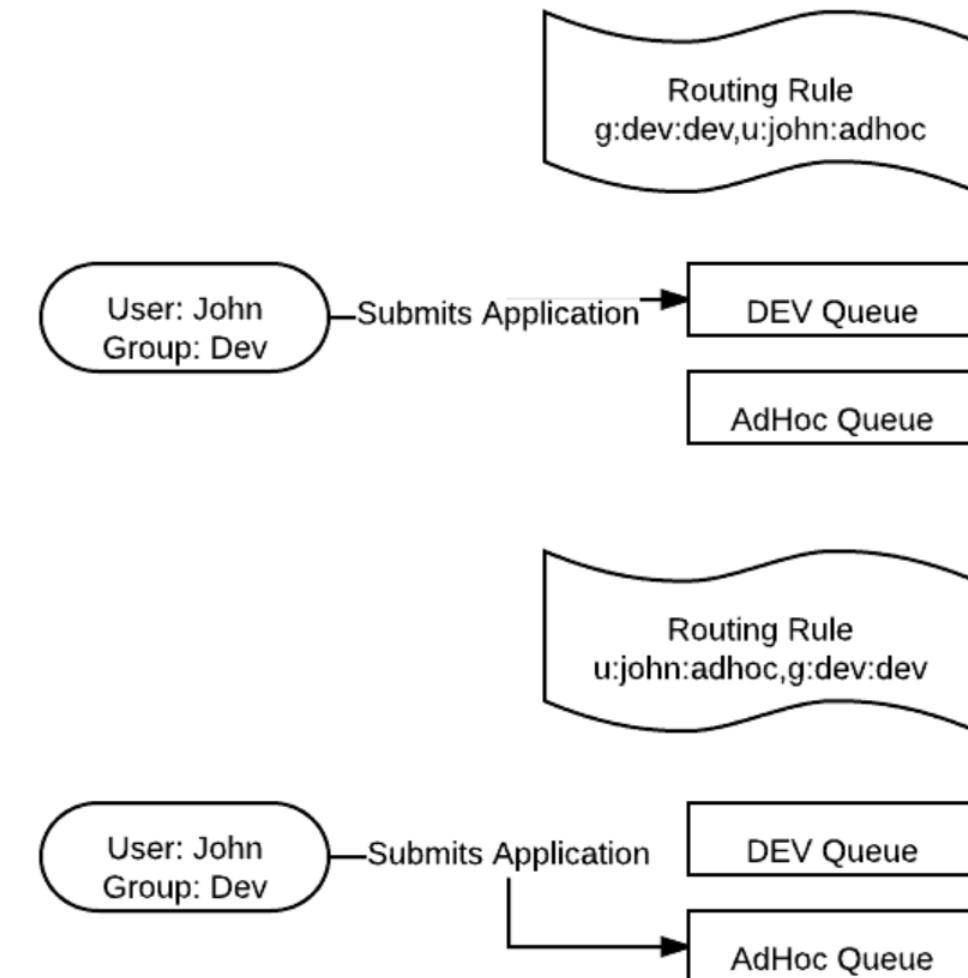


# Capacity scheduler : Ordering policies within queue

- FIFO ordering
  - ✓ Ordering based on arrival time
  - ✓ Large applications can block each other and cause user discontent
  - ✓ There is no preemption within a queue anyway
- FAIR ordering
  - ✓ Give resources to smallest requests first and new applications with least resources to get started (like shortest job first)
  - ✓ Works well when there is good container churn within a queue

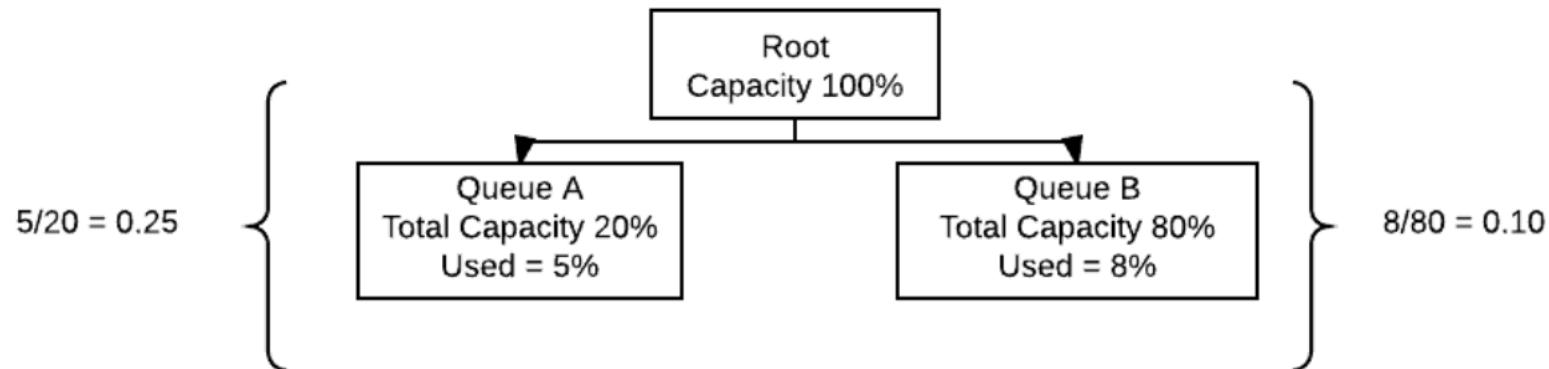
# Capacity scheduler : Mapping to queue

- User and group mapping to queue
- Depends on what order is specified for mapping in routing rules



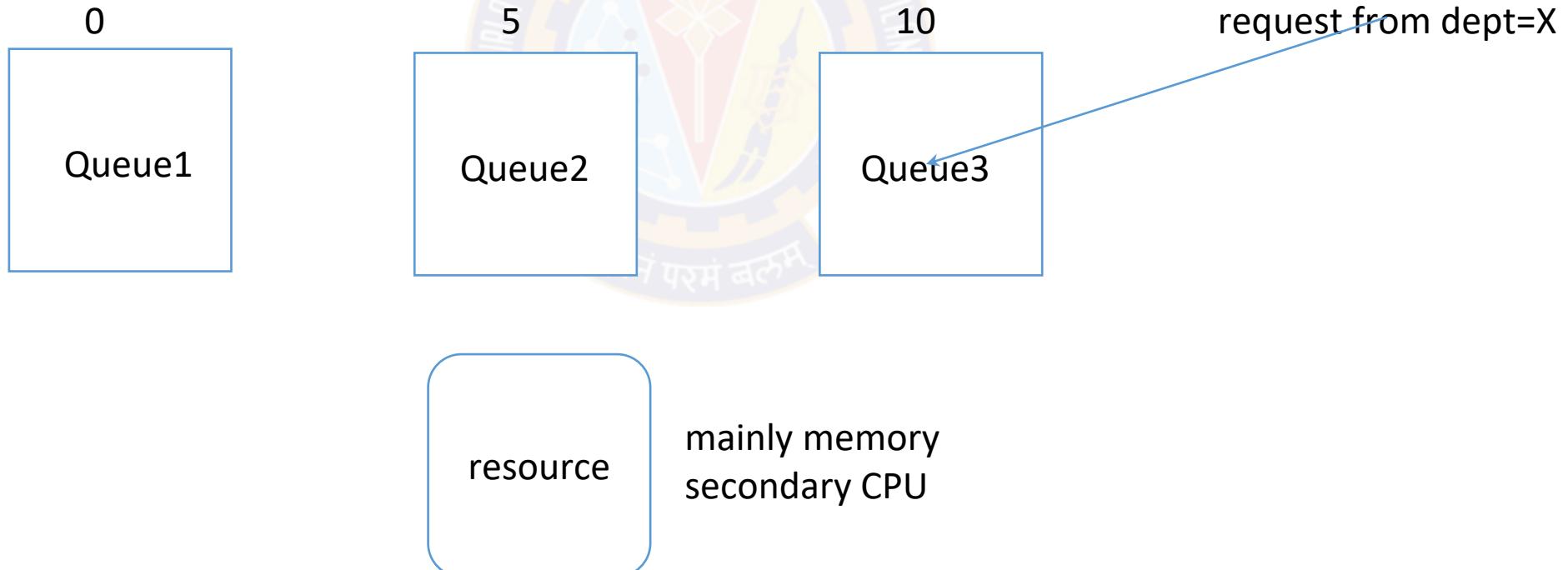
# Capacity scheduler : Priority

- Influence which queue gets new requests beyond resource utilisation driven
- Queue A relative capacity utilisation is  $5 / 20 = 0.25$
- Queue B relative capacity utilisation is  $8 / 80 = 0.10$
- So new requests will go to Queue B
- Increasing priority on Queue A will avoid that



# Resource Manager scheduling - Fair Share scheduler

- Weight replaces min%-max% capacity share
- So, sum of weights don't need to add up to 1 or 100% unlike capacity
- Weight is a measure of what a queue considers as a fair share it needs and weights make sense as ratios of each other
  - ✓ They can be used to calculate approx capacity allocations
- 0 means best effort basis allocation is good enough - it doesn't imply 0% capacity
  - ✓ If no requests in other queues, it may occupy all the capacity as best effort



# Resource Manager scheduling - Fair scheduler

- Ensures that all apps get a “fair” share on the average of the cluster - more popular approach
- Primary resource is memory but CPU + memory can also be configured with one as the dominant resource for deciding fairness
- Works well where shorter jobs are not starved and longer jobs also get to finish without getting unduly delayed when there are many short jobs
- Apps are assigned to queues and unless specified, all apps will go to default queue
- Fair share scheduler works within a queue and also across the queues
- Fair share of a queue is determined by weight (0 means no fair share - just best effort)
  - ✓ No capacity min-max specified - just a single weight
  - ✓ Queue level: A queue can be assigned a minimum share (determined by weight), e.g. a queue for production apps can be assigned a queue with a certain minimum share of the cluster. Excess can be shared with other queues.
  - ✓ App level: Works with app priorities where a weight can be assigned to an app to compute fair share (enable `yarn.scheduler.fair.sizebasedweight=TRUE`)
    - Weight is a function of natural log of memory demanded by the app
- A limit can also be put per queue / per user on how many apps will be picked up for scheduling
- Queues can be organized hierarchically

In `yarn-site.xml`

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
```

# Fair scheduler: Example



```
<?xml version="1.0"?>
<allocations>
  <queue name="marketing">
    <weight>3.0</weight>
    <queue name="reports">
      <weight>40.0</weight>
    </queue>
    <queue name="website">
      <weight>20.0</weight>
    </queue>
  </queue>
  <queue name="sales">
    <weight>4.0</weight>
    <queue name="northamerica">
      <weight>30.0</weight>
    </queue>
    <queue name="europe">
      <weight>30.0</weight>
    </queue>
  </queue>
  <queue name="datascience">
    <weight>13.0</weight>
    <queue name="short_jobs">
      <weight>100.0</weight>
    </queue>
    <queue name="best_effort_jobs">
      <weight>0.0</weight>
    </queue>
  </queue>
</allocations>
```

- Weights determine ratio of usage
  - ✓ Marketing : 3 (15%)
  - ✓ Sales : 4 (20%)
  - ✓ Datascience : 13 (65%)
- Weights need not add up to 100
- Change weights to change the fair share
- Best effort can be weight 0 - which means no fair share, but will get what is remaining if there is spare capacity available in the cluster
- So, min and max as in Capacity scheduler need not be set
- Weights under a hierarchy further indicate fair share within the fair share of the parent

# Preemption across queues in Fair scheduler

- Scenario
  - ✓ Application A is demanding more resources and using elastic capacity of a queue Q1
  - ✓ So Q1 gets unused min allocation from queue Q2
  - ✓ Suddenly Q2 needs its min capacity back because of new applications
- Preemption enables a queue to reclaim elastic resources to bring back its min allocation without making new applications wait till the older application tasks using elastic capacity finishes
- Preemption does not save the context of the task from where it be restarted
- Preemption is done by killing the task (There will of wastage of computing power)
- Preemption tries not to kill tasks of entire application to reclaim resources
  - ✓ It tries to kill younger tasks – tasks in initial stages of execution
  - ✓ Or kill reducers (not mappers that have done a lot of work already) because minimum work is lost
- Preemption will not do anything unless it can fulfil entire allocation request
- Preemption is about only reclaiming min and not max allocation
- Within a queue one has to work with User limit factor or FIFO/FAIR ordering policies.

From below example, If queue1 is at 80% of its 50% share and need more resources and waiting for more than 60 seconds, queue1 is allowed to preempt containers from queue2. On a similar situation for queue2, it cannot preempt containers from queue1 because it does not have these parameters set.

```
<allocations>
  <queue name="queue1">
    <fairSharePreemptionTimeout>60</fairSharePreemptio
nTimeout>

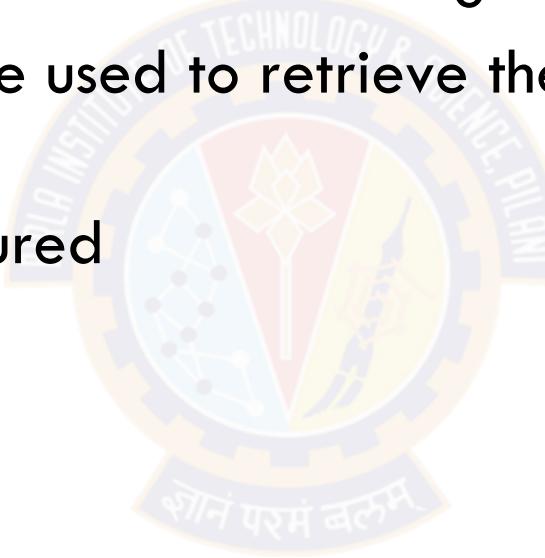
    <fairSharePreemptionThreshold>0.80</fairSharePreem
ptionThreshold>
      <weight>1.0</weight>
    </queue>
    <queue name="queue2">
      <weight>1.0</weight>
    </queue>
  .
  .
</allocations>
```

# Difference between Capacity scheduler and Fair scheduler

	Capacity	Fair
	<ul style="list-style-type: none"><li>Allows sharing a large cluster while giving each department a minimum capacity guarantee.</li><li>Available resources in the <a href="#">Hadoop cluster</a> are partitioned among multiple departments who collectively fund the cluster based on computing needs.</li><li>Set up by queues for each dept with specified amount of capacity.</li><li>Jobs within the same queue get scheduled in FIFO order.</li><li>Added benefit is that an organization can access any excess capacity not being used by others.</li><li>Provides elasticity for the organizations in a cost-effective manner</li></ul>	<ul style="list-style-type: none"><li>A method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time.</li><li>When there is a single job running, that job uses the entire cluster.</li><li>When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time.</li></ul>
	Default in Hadoop 3.3.5	

# Log management in YARN

- Unlike Hadoop 1, YARN makes sure that through Node Manager, local logs are moved to a file system like HDFS in configurable directories
- YARN commands / UI can be used to retrieve these logs that are at node level or application level
- Log retention can be configured



# Some YARN commands

Display yarn jobs:

`yarn top`

Display status of a specific queue:

`yarn queue -status BDS_Q1`

Start an application:

`yarn app start <app name>`

Start a JAR file:

`yarn jar <jar> <main class> <args>`

Stop an application:

`yarn app stop <appid>`

Change priority:

`yarn app -updatePriority <priority> -appId <appid>`

Know which applications are running:

`yarn app -list -appStates RUNNING`

Get logs from an application:

`yarn logs -appId <appid>`

Changing Queue Configuration:

`yarn rmadmin -refreshQueues`

# Summary

- Real world use cases with MapReduce model of computing
- Hadoop streaming
- YARN – Yet another resource negotiator
- Pluggable schedulers in YARN
- Yarn command interface





Next Session:  
**Hadoop ecosystem tech: Pig, Hive, HBase**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 8: Hadoop Ecosystem: PIG, HIVE, HBASE

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- Hadoop ecosystem technologies
  - ✓ Pig - Scripting on top of MapReduce
  - ✓ Hive - Data warehouse
  - ✓ HBase - Key-value store

Learn basics about

- A. Applicability
- B. Architecture
- C. Data manipulation usage

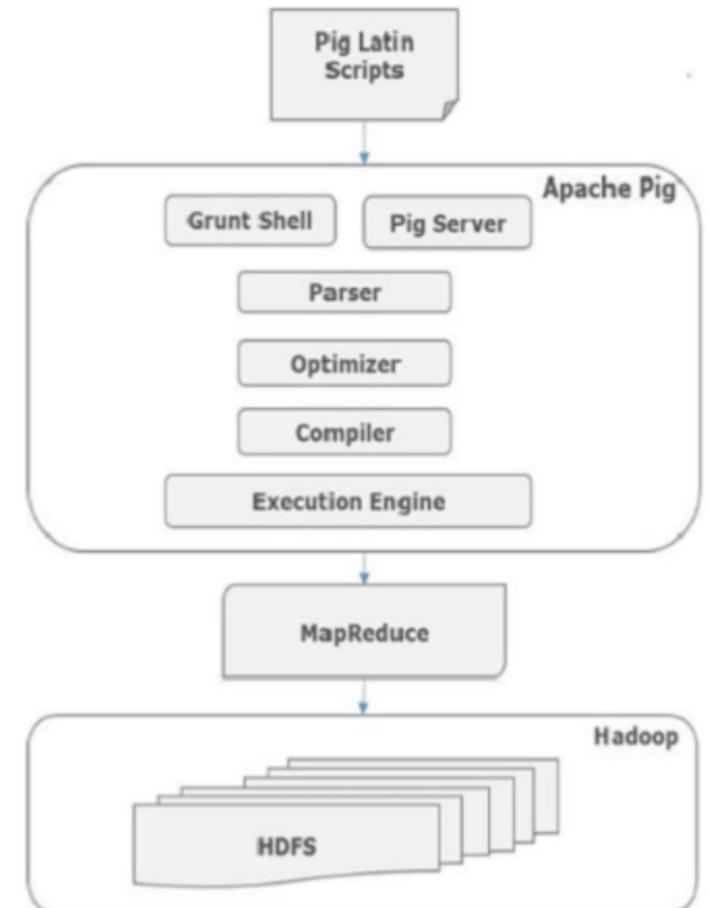


# Why PIG ?

- Hadoop MapReduce needs a developer to write UDFs for Map/Reduce using Java, Python etc.
- Pig gives developers a simpler scripting interface to manipulate data using a higher level programming interface
  - ✓ Similar to SQL - so becomes simpler for data engineers
- Pig script is translated to map reduce tasks by the Pig runtime and executed on the Hadoop cluster
  - ✓ However automated code may be less efficient than a native MR program
- Not an acronym - Pig can consume and process any data
- Developed in Yahoo

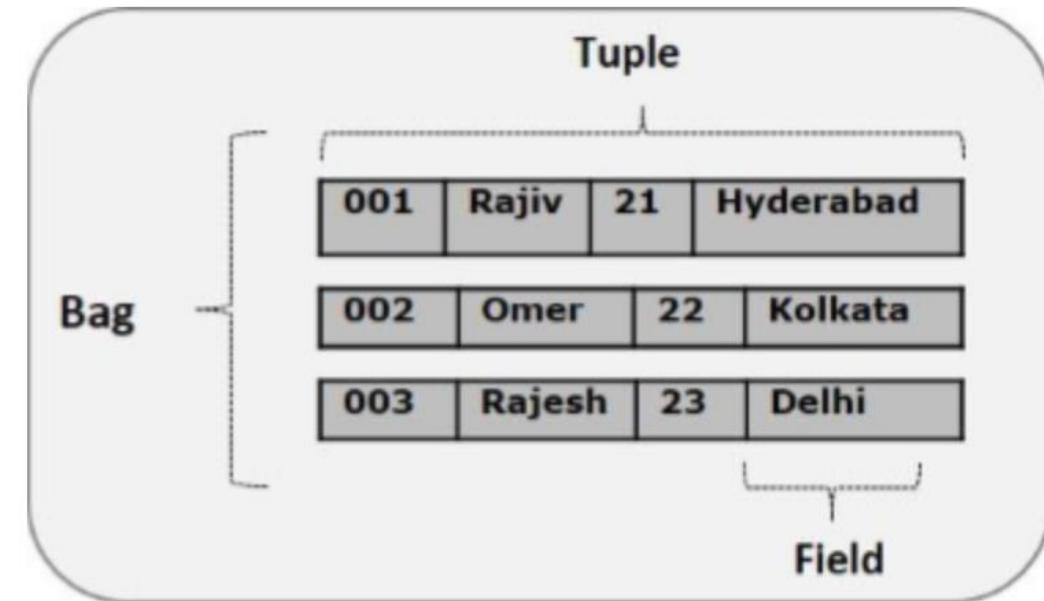
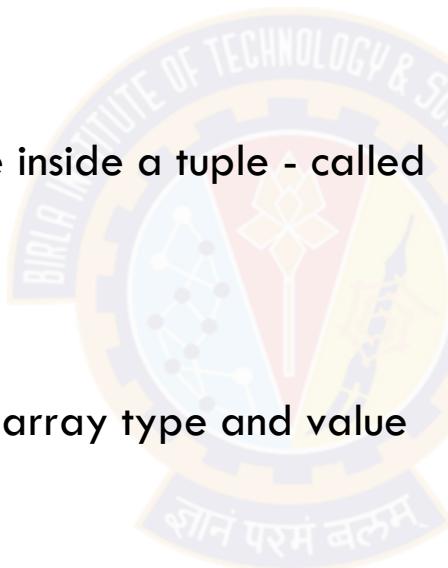
# Architecture

- Programs are written in Pig Latin - a scripting language
- Pig Latin scripts are processed by the following components
  - ✓ Parser: Creates a DAG with syntactic and type checks. Nodes are operator and edges are data flows. This is a logical plan.
  - ✓ Optimizer: Removes unnecessary data or columns.
  - ✓ Compiler: Generates MR jobs and optimises the execution order.
  - ✓ Execution engine: Run the MR jobs on the Hadoop platform
  - ✓ Execution mode: Can work in local JVM for dev/test or by default on Hadoop cluster



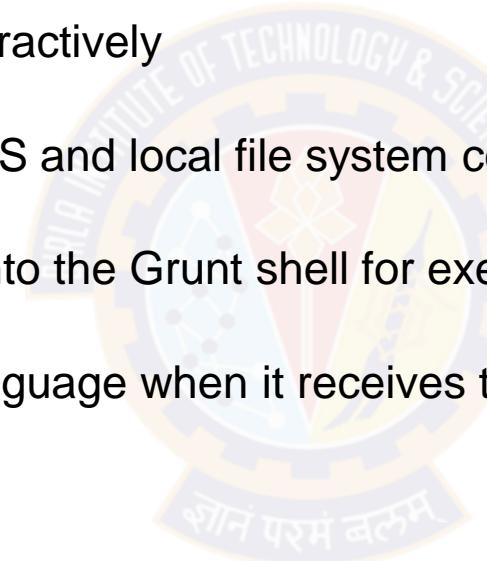
# Data Model

- Atom: Basic data types like byte, int, float, byte array.
  - ✓ e.g. andy, 67, ...
- Tuple: Ordered set of fields of various types, like a row in RDBMS table.
  - ✓ e.g. (andy, 25)
- Bag: An unordered set of tuples. A bag can be inside a tuple - called an inner bag.
  - ✓ e.g.  $\{(andy, 65), (ram, 23, 6000)\}$
  - ✓ e.g. inner bag:  $(1,\{(1,2,3)\})$
- Map: A key-value pair, where a key is a char array type and value can be any type.
  - ✓ e.g. [name#andy, age#25]
- Relation: A bag but not an inner bag which can be inside a tuple.  
Like a table in RDBMS.



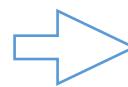
# PIG Grunt shell

- Pig Grunt is an interactive shell
- Enables users to enter Pig Latin interactively
- Provides a shell to interact with HDFS and local file system commands.
- Enter Pig Latin commands directly into the Grunt shell for execution.
- Pig starts executing the Pig Latin language when it receives the STORE or DUMP command.
- Invoke in Local mode
  - ✓ `pig -x local` (Specify sample.pig file to execute the pig script in local mode)
- Invoke in Mapreduce mode
  - ✓ `pig -x mapreduce` (Specify sample.pig file to execute pig script as Mapreduce job on Hadoop cluster)



# PIG Operators

- ✓ FOREACH – performs iterations over every record to perform a transformation
- ✓ ASSERT – asserts a condition on the data
- ✓ FILTER – enables to use a predicate for selecting the records that need to be retained in the pipeline
- ✓ GROUP – to group and aggregate
- ✓ ORDER BY – sorting a given relation, depending on one or more fields
- ✓ DISTINCT – used for removing duplicate fields from a given set of records
- ✓ JOIN – used for joining two or more relations
- ✓ LIMIT – to limit the number of results
- ✓ SAMPLE – used for selecting random data sample by providing a sample size
- ✓ SPLIT - partitions a given relation into two or more relations
- ✓ LOAD
- ✓ DUMP
- ✓ STORE



# PIG Analysis scenario 1: Count in groups

Count the number of transactions by item type for sales in India

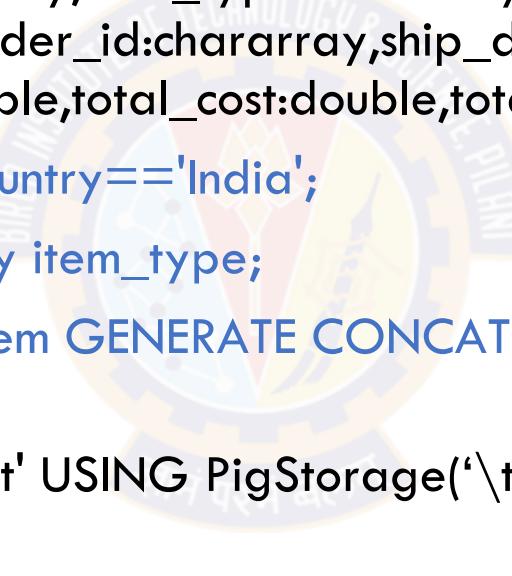
```
grunt> geosales = LOAD '/geosales.csv' using PigStorage(',') as  
(index:long,region:chararray,country:chararray,item_type:chararray,sales_channel:chararray,order  
_priority:chararray,order_date:Datetime,order_id:chararray,ship_date:Datetime,units_sold:int,unit  
_price:float,unit_cost:float,total_revenue:double,total_cost:double,total_profit:double);
```

```
grunt> india_data = FILTER geosales BY country=='India';
```

```
grunt> groupbyitem = group india_data by item_type;
```

```
grunt> countbyitem = FOREACH groupbyitem GENERATE CONCAT((chararray)$0,  
CONCAT(':',(chararray)COUNT($1)));
```

```
grunt> STORE countbyitem INTO 'pig-output' USING PigStorage('\t');
```

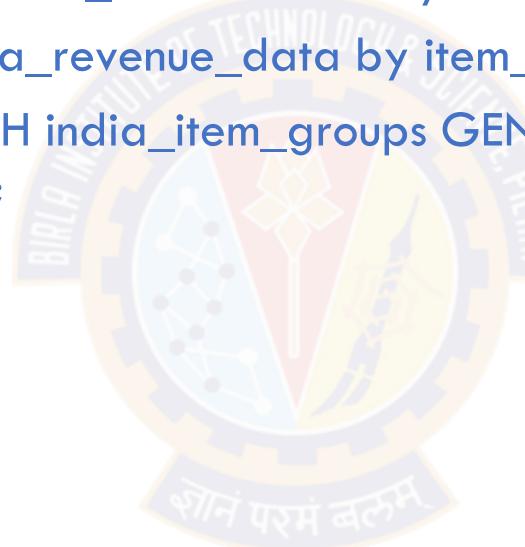


Meat:39
Cereal:41
Fruits:53
Snacks:34
Clothes:48
Baby Food:59
Beverages:42
Cosmetics:52
Household:33
Vegetables:43
Personal Care:48
Office Supplies:48

# PIG Analysis scenario 2: Basic stats

Avg/Max/Min/Sum revenue by item type for sales in India

```
grunt> revenue_data = FOREACH geosales GENERATE country, item_type, total_revenue ;  
grunt> india_revenue_data = FILTER revenue_data BY country=='India';  
grunt> india_item_groups = GROUP india_revenue_data by item_type;  
grunt> avg_revenue_by_item = FOREACH india_item_groups GENERATE $0,  
AVG(india_revenue_data.total_revenue);  
grunt> dump avg_revenue_by_item;
```



(Meat	2275631.3892307696)
(Cereal	1116549.6341463416)
(Fruits	46948.560000000005)
(Snacks	926205.4764705882)
(Clothes	528935.69)
(Baby Food	1401703.538983051)
(Beverages	240074.4047619048)
(Cosmetics	2583179.3846153845)
(Household	3683585.242424242)
(Vegetables	745693.3934883721)
(Personal Care	394076.519375)
(Office Supplies	3242835.86375)

# PIG Analysis scenario 3: Sort

Top revenue item in India / Sort items by revenue

```
grunt> revenue_data = FOREACH geosales GENERATE country, item_type,  
total_revenue ;  
  
grunt> india_revenue_data = FILTER revenue_data BY country=='India';  
  
grunt> india_item_groups = GROUP india_revenue_data by item_type;  
  
grunt> item_grand_totals = FOREACH india_item_groups GENERATE $0,  
SUM(india_revenue_data.total_revenue) as grand_total;  
  
grunt> sorted_items = ORDER item_grand_totals BY grand_total DESC;  
  
grunt> dump sorted_items;
```

# Topics for today

- Hadoop ecosystem technologies
  - ✓ Pig
  - ✓ **Hive**
  - ✓ HBase

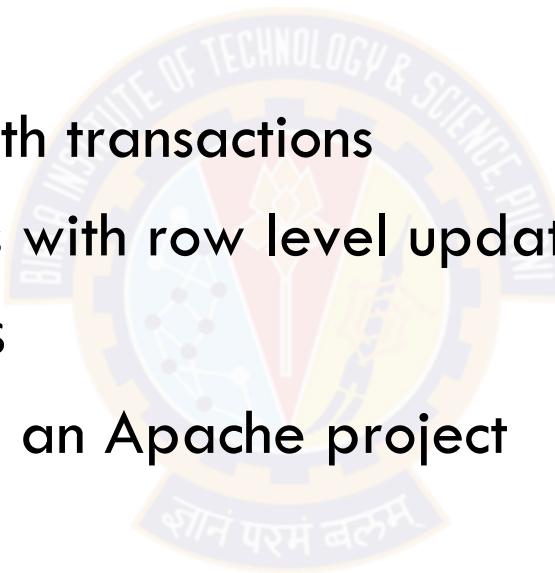
Learn basics about

- A. Applicability
- B. Architecture
- C. Data manipulation usage



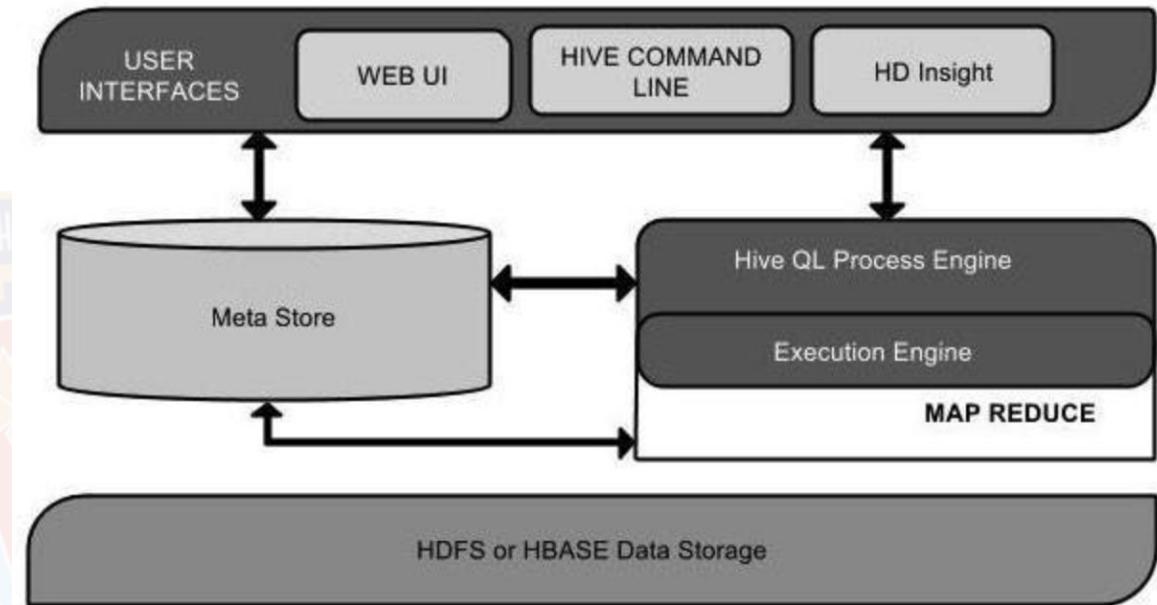
# Why Hive

- Provides a way to process large structured data in Hadoop
- Data Warehouse on Hadoop / HDFS
- SQL like query interface
  - ✓ But it is not an RDBMS with transactions
  - ✓ Not for real-time queries with row level updates
- Meant for OLAP type queries
- Initially in Facebook and now an Apache project



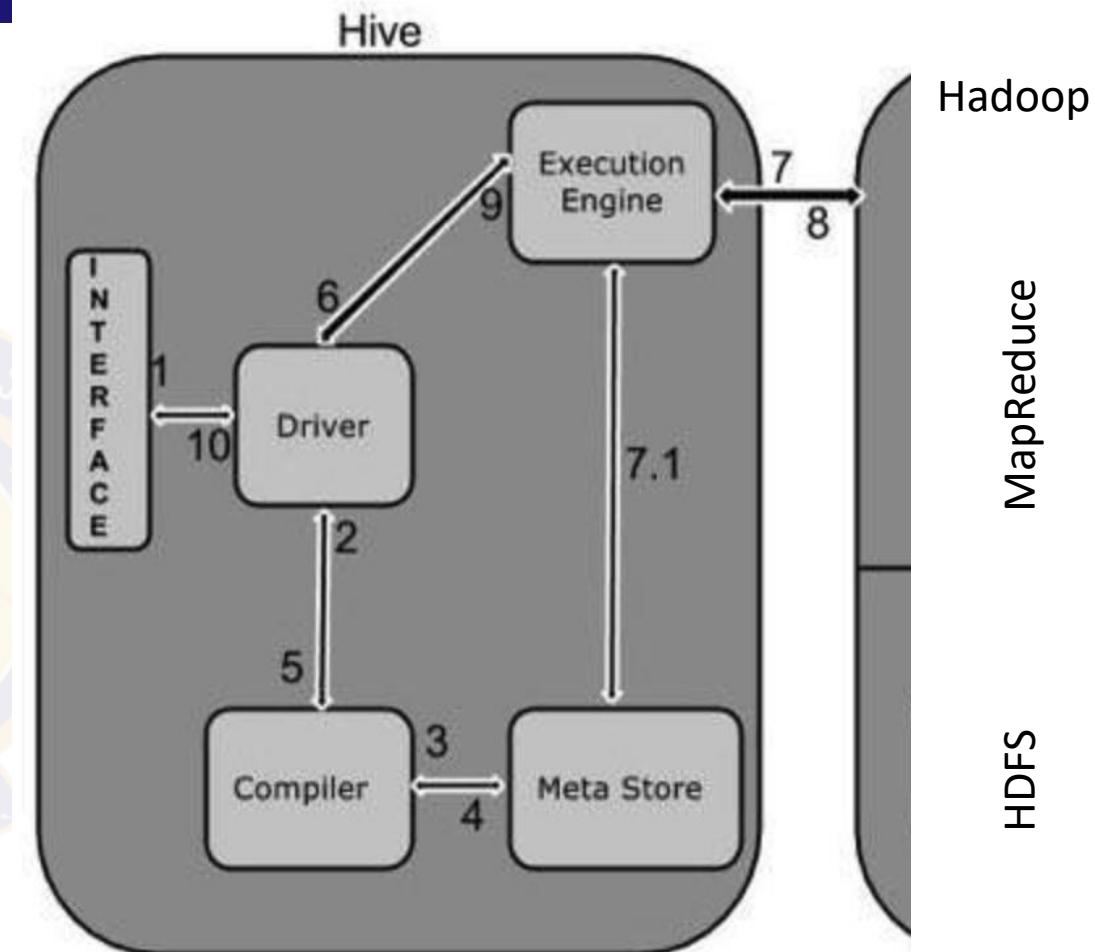
# Hive Architecture

- Meta Store: Meta-data or schema is stored in a relational DB (default Derby, MySQL, ..) containing structure of tables, partitions, columns, data types, [de]-serializers to read/write data, HDFS files or HBase meta-data where data is stored.
- Provides SQL-like interface called HiveQL
- Execution engine with a Compiler (in HiveQL process engine) that consults meta-data to translate a query into MapReduce jobs
- Data in HDFS or in HBase



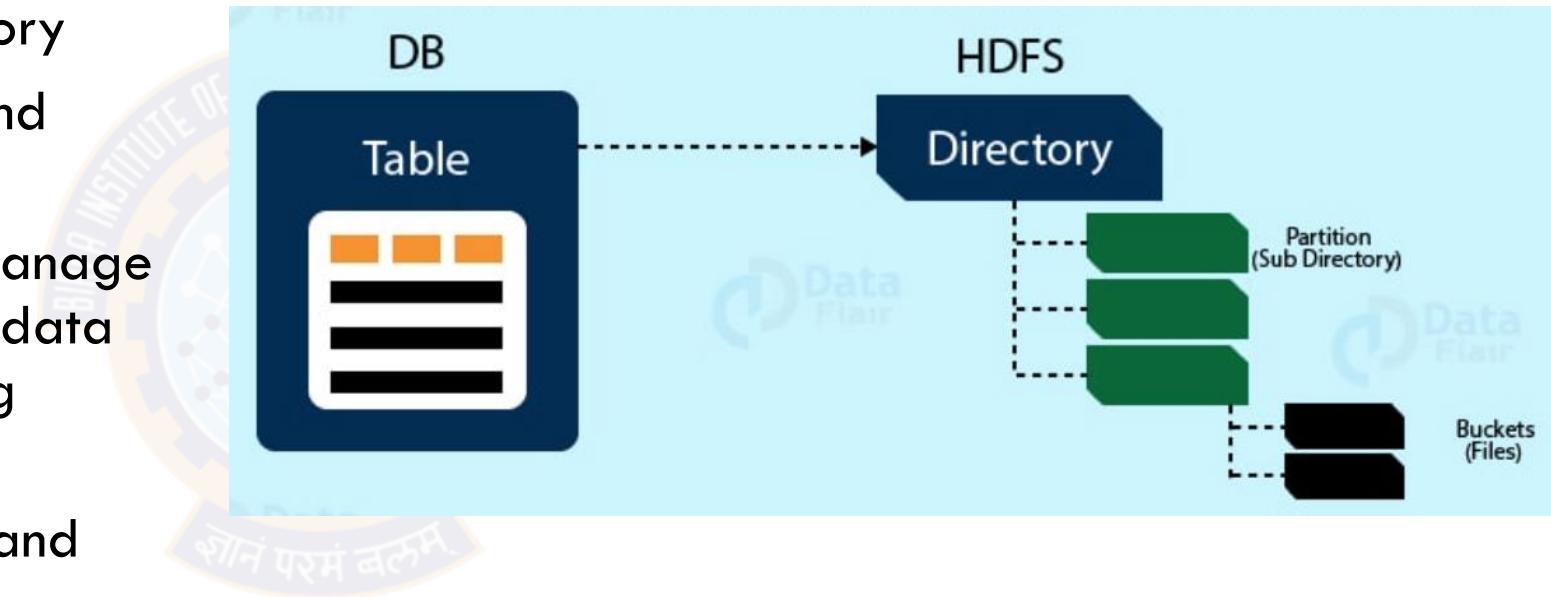
# Hive Query execution flow

1. HiveQL query sent to driver from Interface.
2. Driver sends to Compiler to create a query plan
3. Compiler builds an abstract syntax tree (AST) and then semantic analysis of the query to create a query plan graph (DAG). For this it consults the Meta Store.
4. Meta Store responds to Compiler on requests.
5. Compiler (post optimization of the DAG) responds to Driver with the final plan.
  - ✓ Possible optimizations are combining consecutive joins, splitting tasks - e.g. combiners before reducer etc.
6. Driver sends to Execution Engine that essentially executes the query plan via Hadoop MapReduce + data in HDFS or HBase and finally result is sent to Hive interface. (steps 7-10)



# Data model

- Table
  - ✓ Same as RDBMS tables
  - ✓ Data is stored in a HDFS directory
  - ✓ **Managed tables:** Hive stores and manages in warehouse dir
  - ✓ **External tables:** Hive doesn't manage but records the path. So actual data can be brought in after creating external table in Hive.
  - ✓ Tables are split into Partitions and then into Buckets



# Managed Table (Internal Table)

- When a user creates a table in Hive it is by default an internal table created in the **/user/hive/warehouse** directory in HDFS which is its default storage location.
  - The data present in the internal table will be stored in this directory and is fully managed by Hive and thus an internal table is also referred to as a managed table.
  - By default, the table created in the hive is an internal table, so we need not specify an **internal** keyword while table creation
- 
- Eg:
  - **CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';**

# External table

- When a user creates a table in Hive specifying the external keyword, then an external table is created.
- The data present in the external table will be fully managed by HDFS contrary to an internal table
- For external tables, metadata entries will be made in metastore\_db
- When an external table is dropped, entries in the meta\_store get removed and the data remains on HDFS
- Typical use case of Schema on Read

Eg: To create external table named 'EXT\_STUDENT'.

```
CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name STRING,gpa FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/STUDENT_INFO';
```

# Hands on Queries

Country	Population	GDP	GINI
Albania	3062	3658	0.642
Algeria	30463	6107	0.67
AntiguaandBarbuda	77	18007	0.747
Argentina	36896	11729	0.74
Armenia	3082	3068	0.684
Australia	19071	27193	0.622
Austria	8096	24836	0.646
Azerbaijan	8143	3555	0.678
Bangladesh	128916	1772	0.66

Write HiveQL queries to find the following:

1. List of countries in which the population is more than 100000
2. The country with the highest GDP
3. The country with the lowest value for the GINI coefficient
4. Name of 5 countries with the highest population.

# Queries on internal table

Create an internal table named world1 and load data into it:

```
CREATE TABLE world1 (country string, population int, GDP float, GINI float) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '/home/jps/Handson/Hive/WorldPopulation.csv' OVERWRITE INTO TABLE world1;
```



Queries:

```
select * from world1;
```

```
Select * from world1 where country='India';
```

```
select country from world1 where population>100000;
```

```
select country from world1 where GDP in (select Max(GDP) from world1);
```

```
select country from world1 where GINI in (select Min(GINI) from world1);
```

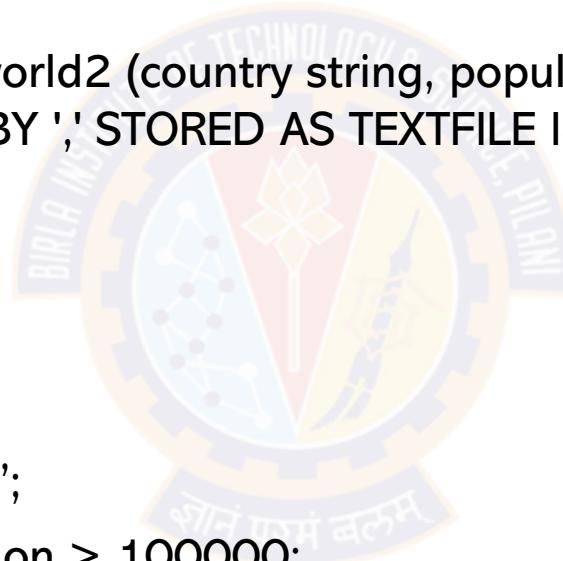
```
select country from world1 order by population DESC LIMIT 5;
```

# Queries on external table

Create an external table named world2:

Copy the .csv file to an HDFS folder - hadoop fs -put WorldPopulation.csv /hive

CREATE EXTERNAL TABLE IF NOT EXISTS world2 (country string, population int, GDP float, GINI float) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE location '/hive';



Queries:

select \* from world2;

Select \* from world2 where country='India';

select country from world2 where population > 100000;

select country from world2 where GDP in (select Max(GDP) from world2);

select country from world2 where GINI in (select Min(GINI) from world2);

select country from world2 order by population DESC LIMIT 5;

# Hands on with Hive queries

Hive configuration

Walk through of Hive installation

Create internal table

- Run queries

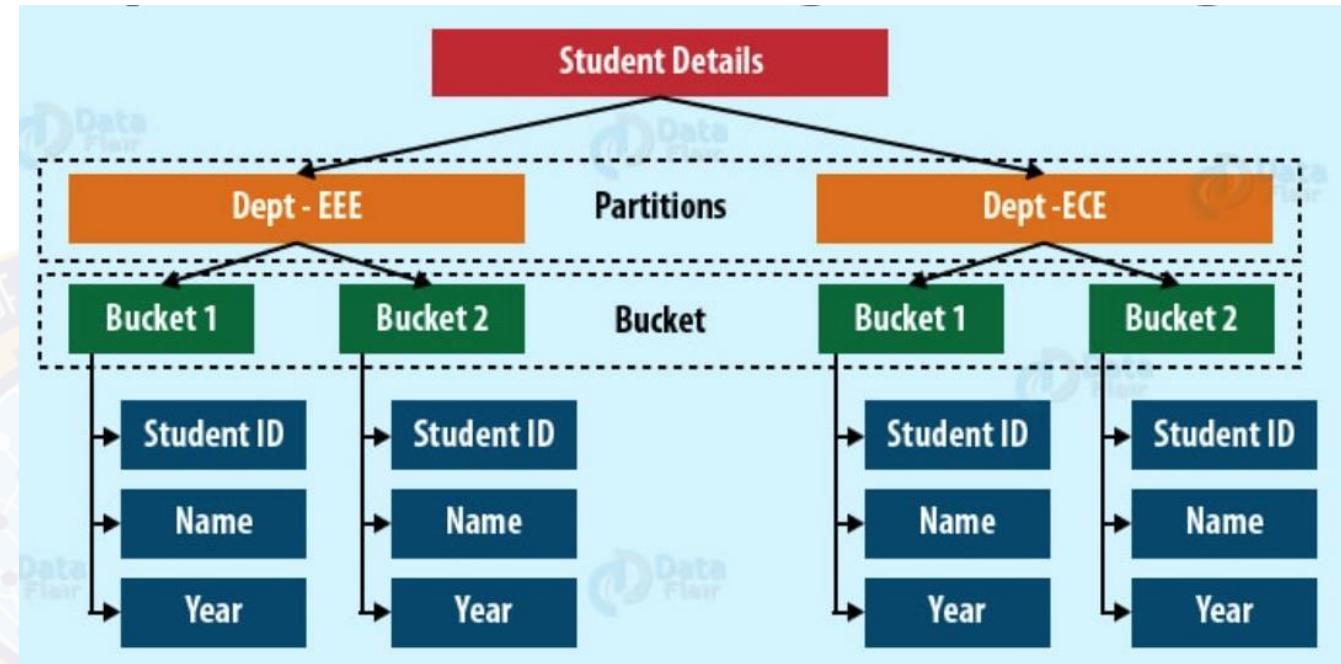
Create external table

- Run queries on multiple files in an HDFS folder



# Data model

- Partition
  - ✓ When creating a table a key can be used to split data into partitions - implemented as separate sub-dirs with table dir on HDFS
  - ✓ e.g. student data partitioned by dept id
- Bucket
  - ✓ Additional level of sub-division within a partition based on hash of some column to make some queries efficient
  - ✓ These map to actual files on HDFS / HBase



```
CREATE TABLE student_details (id int, name varchar(50), year int) PARTITIONED  
BY (dept varchar(3)) CLUSTERED BY (year) SORTED BY (id ASC) INTO N BUCKETS
```

*put year in same bucket but control total bucket count to keep their sizes balanced*

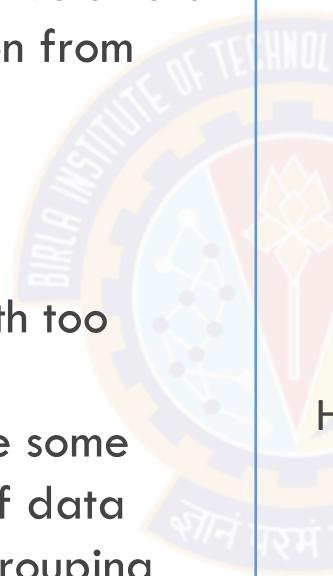
# Data partitioning vs bucketing

## Partitioning

- Distributes execution load horizontally.
- Map tasks can be given partitions
- Faster execution of queries with the low volume of data takes place. E.g. search population from Vatican City returns faster than China

However,

- May have too many small partitions with too many directories.
- Effective for low volume data. But there some queries like group by on high volume of data take a long time to execute. For e.g., grouping population of China will take longer than Vatican City.



## Bucketing

- Done when partition sizes vary a lot or there are too many partitions
- Provides faster query response within smaller segments of data. Like further indexing beyond partition keys.
- Almost equal volumes of data in each bucket — so joins at Map side will be quicker.
- Enables pre-sorting on smaller data sets. Makes Map side merge sorts even more efficient.

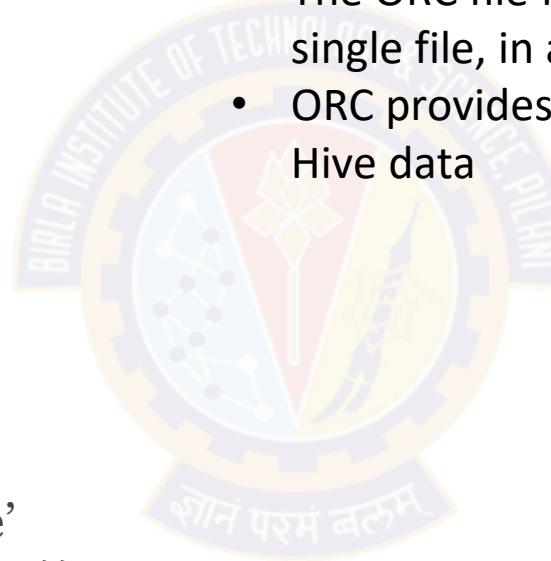
However,

- Can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually\* by programmers.

\* Like ETL - this is a Data Warehouse !

# Example of Partitions and Buckets

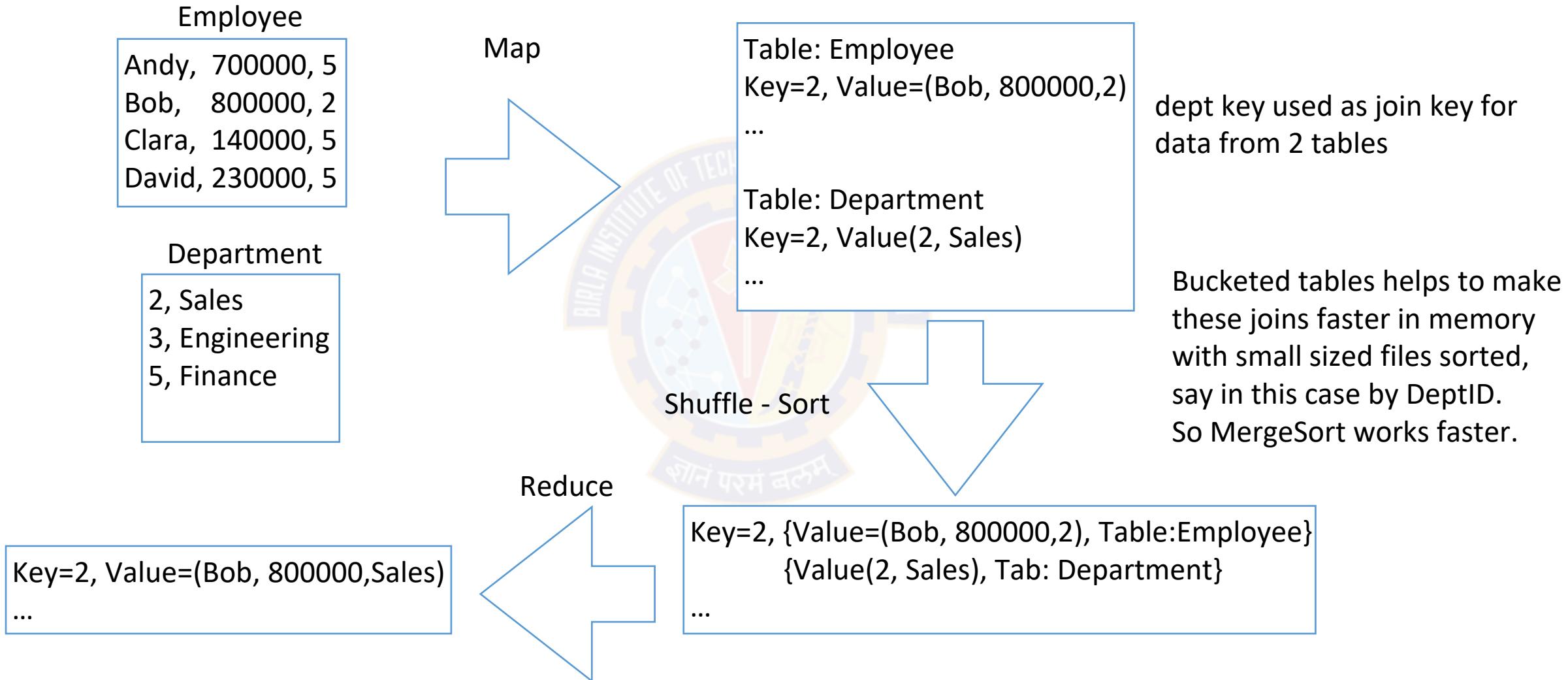
```
CREATE TABLE bucketed_user (
    firstname VARCHAR(64),
    lastname  VARCHAR(64),
    address   STRING,
    city      VARCHAR(64),
    state     VARCHAR(64),
    post      STRING,
    phone1    VARCHAR(64),
    phone2    STRING,
    email     STRING,
    web       STRING
)
COMMENT 'A bucketed sorted user table'
PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS
STORED AS SEQUENCEFILE;
```



- Optimized Row Columnar (ORC) is an open-source columnar storage file format
- The ORC file format stores collections of rows in a single file, in a columnar format within the file.
- ORC provides a highly-efficient way to store Apache Hive data

Create partitions based on country  
Group state records into same bucket  
Control total number of buckets  
Store as binary file to save space

# A point about Map-side joins and buckets



## Storage format - Record Columnar file (RC file or ORC file)

c1	c2	c3	c4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54



Row Group 1				Row Group 2			
c1	c2	c3	c4	c1	c2	c3	c4
11	12	13	14	41	42	43	44
21	22	23	24	51	52	53	54
31	32	33	34				

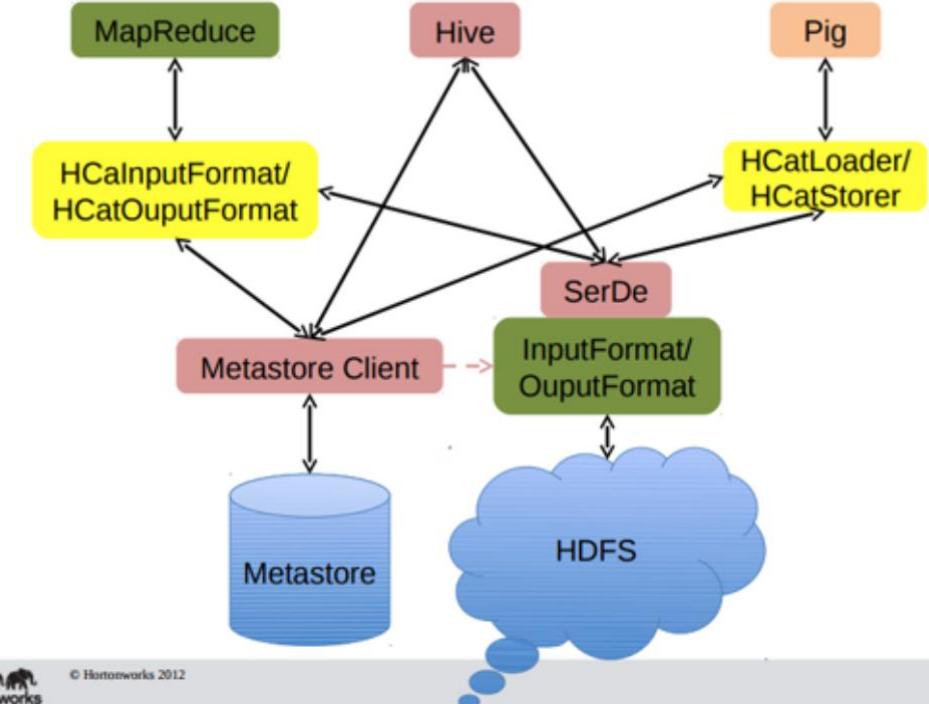


better compression opportunity  
for values from same column

Row Group 1	Row Group 2
11, 21, 31;	41, 51;
12, 22, 32;	42, 52;
13, 23, 33;	43, 53;
14, 24, 34;	44, 54;

# Broader use of Hive meta-store: HCatalog service

- Hive uses a Meta Store for structured data in tables with data on HDFS
- MapReduce programs, Pig scripts can independently access HDFS
- HCatalog: Hive MetaStore exposed as a REST service for other Hadoop ecosystem tech to use
  - ✓ E.g. Hadoop MapReduce Java code can use HCatalog HCatLoader and HCatStore implementations of Hadoop InputFormat and OutputFormat.
  - ✓ So read data from Hive and save results back to Hive
- Can be used to coordinate tasks working on same data across MR and Pig



a = LOAD 'hdfs://localhost:9000/user/hadoop/sales.csv'  
USING PigStorage(',') AS  
(shop:chararray,employee:chararray,sales:int);



a = LOAD 'sales.csv' using HCatLoader();

# Pig Vs Hive

## Pig

Procedural Data Flow Language

For Programming

Mainly used by Researchers and Programmers

Operates on the client side of a cluster.

Does not have a dedicated metadata database.

Pig is SQL like but varies to a great extent.

Pig supports Avro file format.

## Hive

Declarative SQLish Language

For creating reports

Mainly used by Data Analysts

Operates on the server side of a cluster.

Makes use of exact variation of dedicated SQL DDL language by defining tables beforehand.

Directly leverages SQL and is easy to learn for database experts.

Hive does not support it.

# Topics for today

- Hadoop ecosystem technologies
  - ✓ Pig
  - ✓ Hive
  - ✓ **HBase**

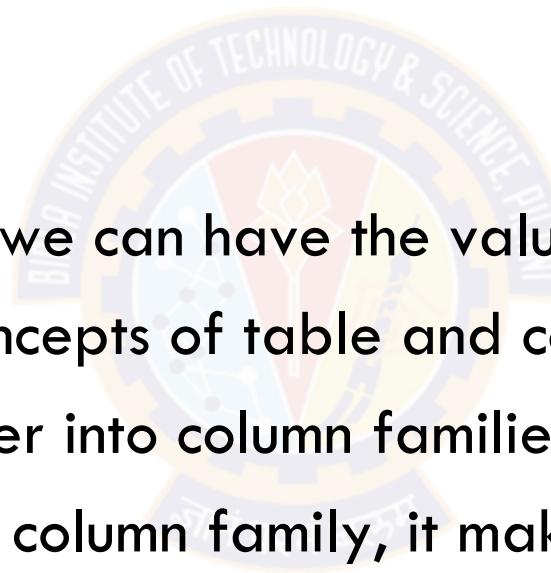
Learn basics about

- A. Applicability
- B. Architecture
- C. Data manipulation usage



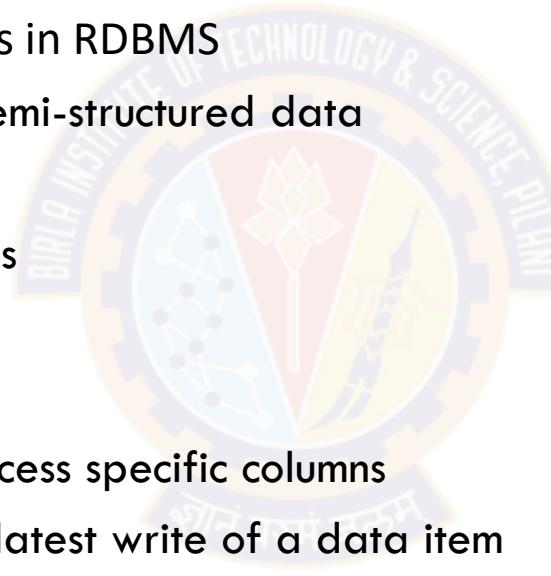
# HBase

- HBase is the Hadoop database
- HBase is a NoSQL database which is:
  - ✓ Consistent (C)
  - ✓ Partition Tolerant (P)
- It is a key value store where we can have the value and a key
- In HBase we have got the concepts of table and columns
- Columns are grouped together into column families
- To access columns of a single column family, it makes it faster as it does not scan other column families
- Use HBase when you need random, real time **read / write access to Bigdata**



# Why HBase ?

- HDFS provides sequential access to data
- HBASE provides random access capability of large files in HDFS
  - ✓ Hash tables used for indexing of HDFS files
- Key-value store with no fixed schema as in RDBMS
  - ✓ so can be used for structured and semi-structured data
  - ✓ type of NoSQL database
- Built for wide tables with many attributes
  - ✓ de-normalized data
- Column oriented storage
  - ✓ Tuned for analytical queries that access specific columns
- Strongly consistent because read is on latest write of a data item
- Origins of idea from Google BigTable (on GFS) leading to a Hadoop project (on HDFS)

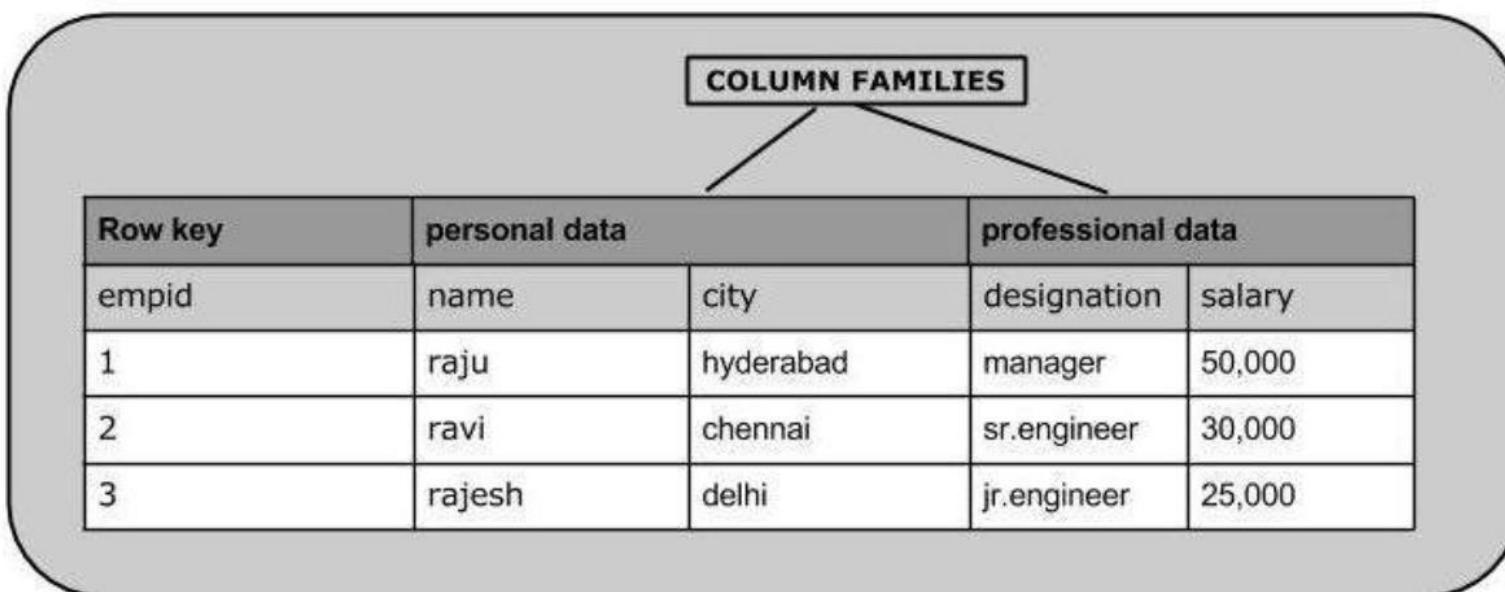


# When to use HBase

- Large data volume where many distributed nodes are needed, e.g. 100s of millions of rows
- Need to lookup data in a large data store
  - ✓ that's the key aspect of HBase on HDFS
- Need linear scalability with data volume
- None of these are required : transactional guarantees, secondary indices, DB triggers, complex queries
- Can add enough commodity hardware to scale with favourable cost-performance ratio

# Columnar storage

- Data in a row is a collection of column families with each column being a key-value pair
- A column values are stored together on disk
- Each cell value also has a timestamp

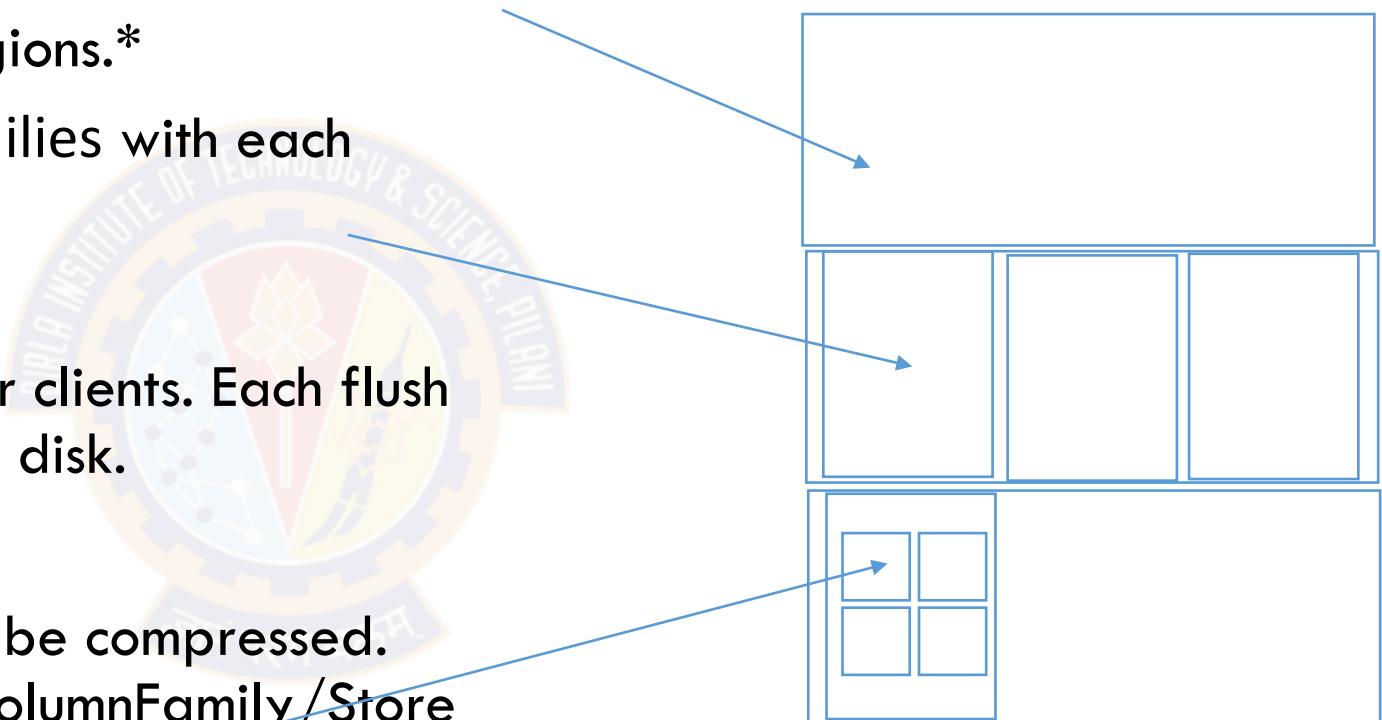


The diagram illustrates the structure of columnar storage. At the top, a box labeled "COLUMN FAMILIES" has two arrows pointing down to a table. The table has a header row with three columns: "Row key", "personal data", and "professional data". Below the header are four rows of data, each containing five cells. The "Row key" column contains values 1, 2, and 3. The "personal data" column contains values raju, ravi, and rajesh. The "professional data" column contains values hyderabad, chennai, and delhi. The next two columns, "designation" and "salary", are part of the "professional data" family and are grouped together in the table.

Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

# HBase storage structure

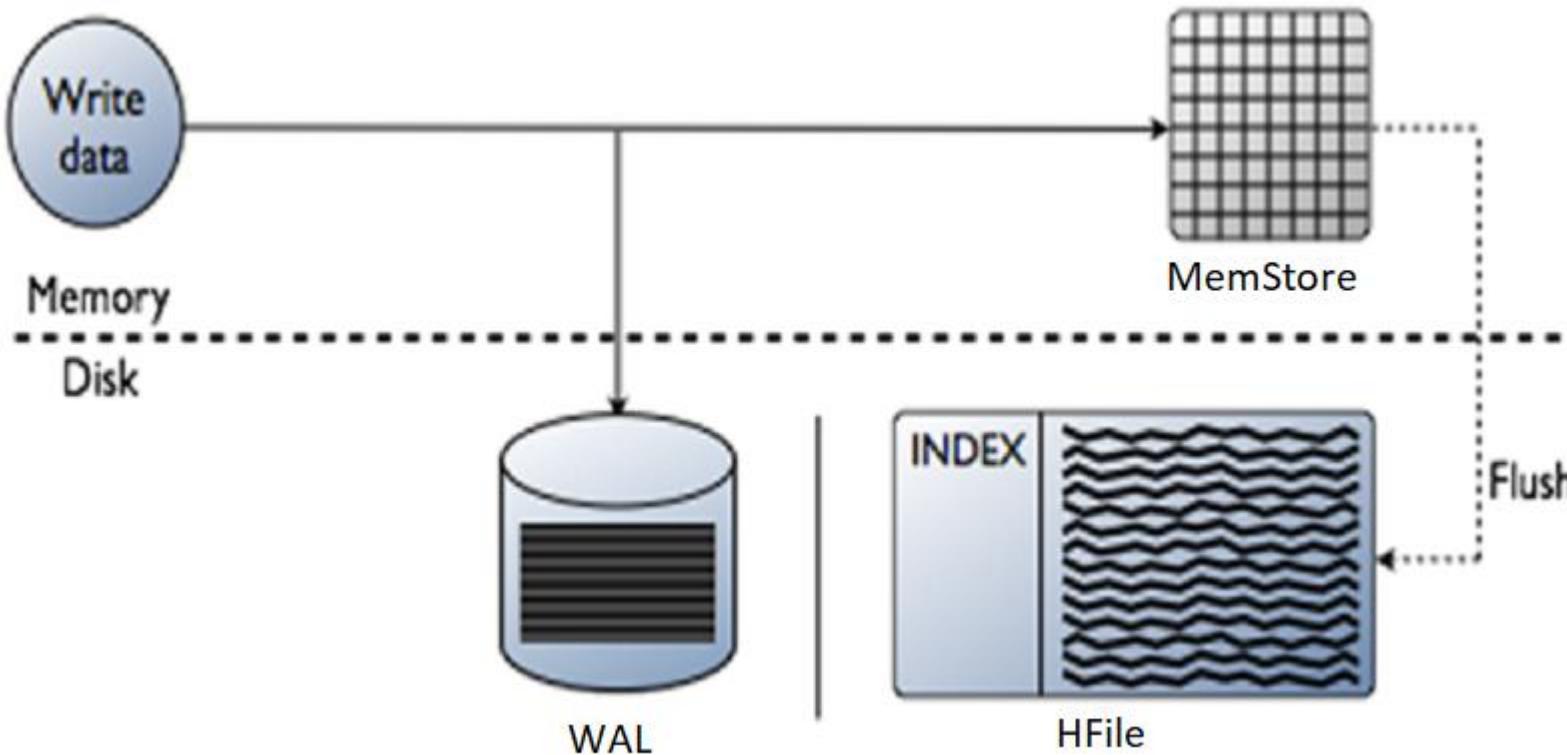
- Region - Continuous sorted set of rows stored together (using row-key to sort).
  - ✓ HBase tables are split into regions.\*
- A region has multiple Column Families with each column family having a Store
- Each Store has
  - ✓ MemStore\*\*: Write buffer for clients. Each flush creates new StoreFile/HFile on disk.
  - ✓ StoreFiles can be compacted
- Each StoreFile has Blocks that can be compressed. Blocksize can be configured per ColumnFamily/Store level.



\*\* configured in : hbase-site.xml/hbase.hregion.memstore.flush.size

\* configured in: hbase.hregion.max.filesize

# HBase write sequence



1. Write to Write Ahead Log (WAL) in the order of arrival
2. Store data in MemStore in the right order sorted by RowKey
3. When MemStore becomes full, transfer to HFile and delete WAL
4. Data in HFile is in sorted order, hence easy to merge

# Architectural components - Master

- HMaster: Single controller on master node.

✓ Assigns regions to RegionServers and checks health.

✓ Failover control

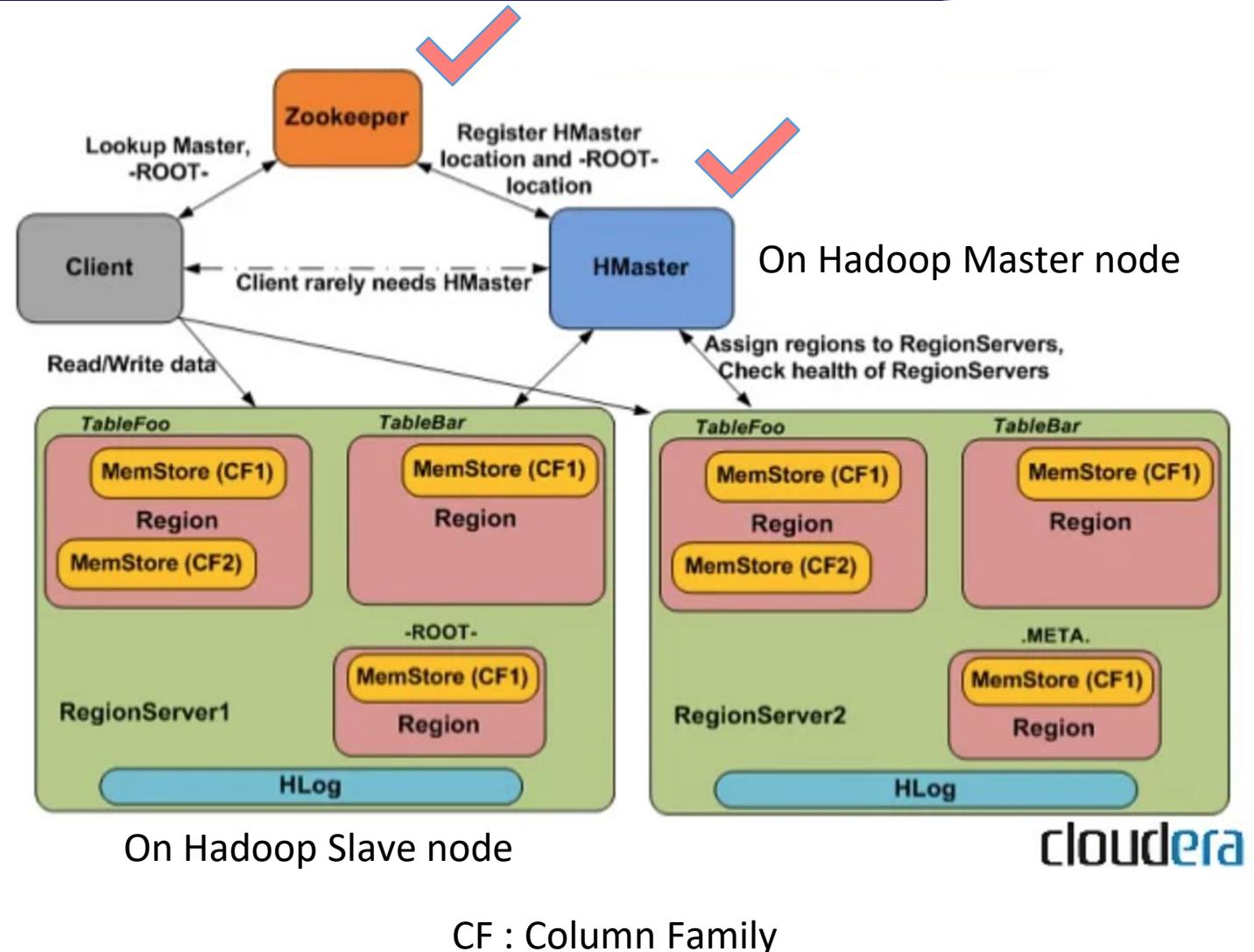
✓ DDL / meta-data operations

- Zookeeper cluster (separate nodes):

✓ Client communication

✓ Track failures with heartbeat

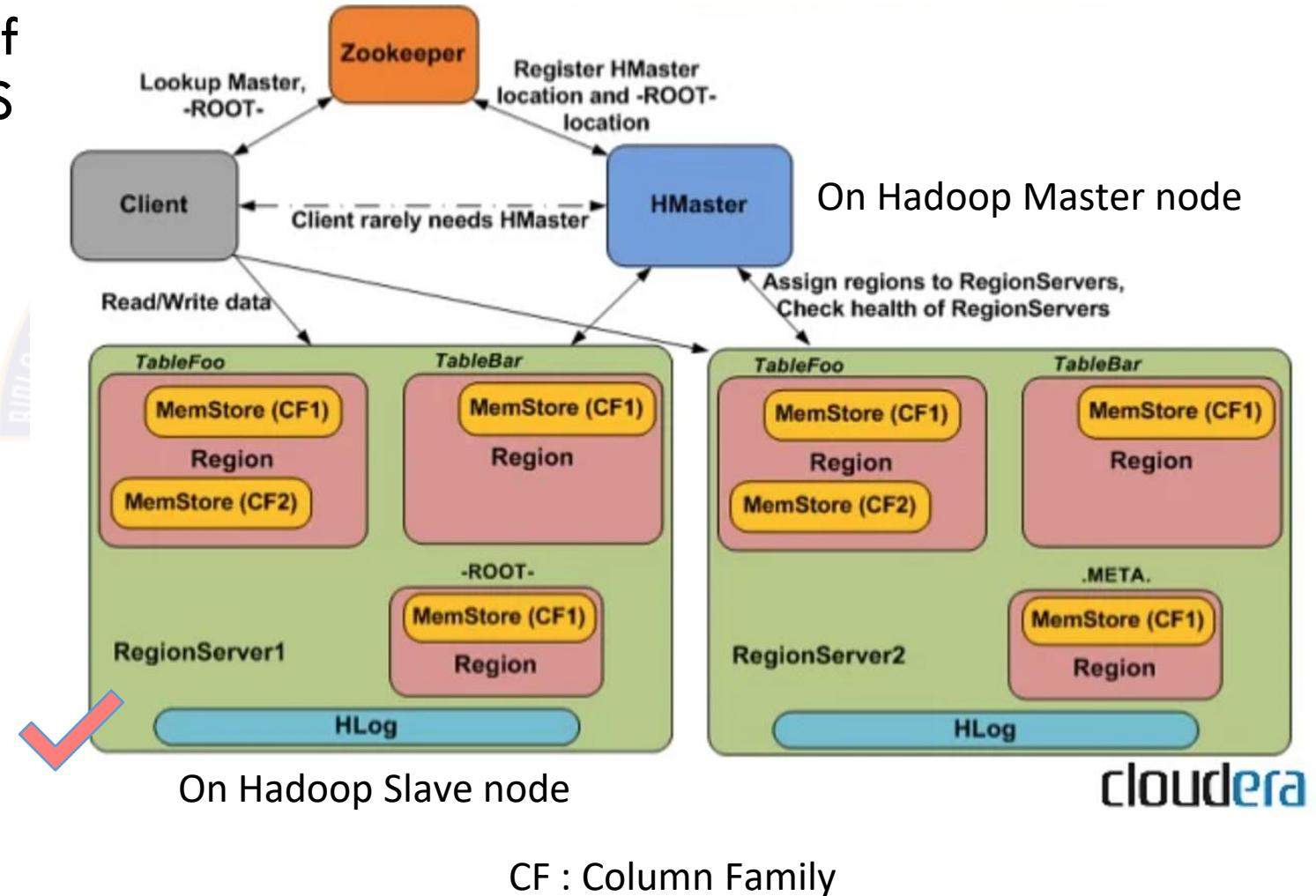
✓ Maintain cluster config data



# Architectural components - Slaves

- RegionServer: In-charge of set of regions on a worker node / HDFS DataNode

- ✓ Worker nodes handling read/write/delete requests from clients
- ✓ HLog or Write Ahead Logs (WAL) for MemStore (look in hbase/WALs/ on HDFS)



# Find HBase objects on HDFS

For data:

/hbase

```
/<Table> (Tables in the cluster)
  /<Region> (Regions for the table)
    /<ColumnFamily> (ColumnFamilies for the Region)
      /<StoreFile> (StoreFiles for the ColumnFamily)
```

For WAL logs:

/hbase

```
/.logs
  /<RegionServer> (RegionServers)
    /<HLog> (WAL HLog files for the RegionServer)
```

To see list of regions and utilisation per region of a table:

```
hadoop fs -du /hbase/<table name>
```

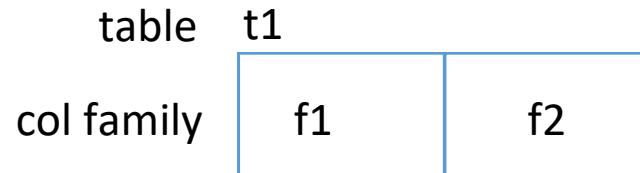
# Find HBase objects on HDFS - Example

```
> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase
drwxr-xr-x 6 anindya staff 192 Jun 27 21:44 MasterData
drwxr-xr-x 3 anindya staff 96 Jun 28 10:30 WALs
drwxr-xr-x 2 anindya staff 64 Jun 27 21:44 archive
drwxr-xr-x 2 anindya staff 64 Jun 27 21:44 corrupt
drwxr-xr-x 4 anindya staff 128 Jun 27 21:44 data
-rw-r--r-- 1 anindya staff 42 Jun 27 21:44 hbase.id
-rw-r--r-- 1 anindya staff 7 Jun 27 21:44 hbase.version
drwxr-xr-x 2 anindya staff 64 Jun 27 21:44 mobdir
drwxr-xr-x 62 anindya staff 1984 Jun 29 01:14 oldWALs
drwxr-xr-x 2 anindya staff 64 Jun 27 21:44 staging
```

Configured path for HBase files

Write Ahead Logs

Data files



```
> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase/data/default
```

```
drwxr-xr-x 5 anindya staff 160 Jun 27 21:44 t1
```

Table

```
> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb65f5fd84bb4201353d1764365
```

```
drwxr-xr-x 3 anindya staff 96 Jun 29 08:10 f1
```

Column families

```
drwxr-xr-x 3 anindya staff 96 Jun 27 23:05 f2
```

```
drwxr-xr-x 3 anindya staff 96 Jun 28 10:30 recovered.edits
```

```
> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb65f5fd84bb4201353d1764365/f1
```

```
-rw-rw-rw- 1 anindya staff 4891 Jun 27 22:53 01d6868c5ed4426393ab08815971b021
```

HDFS file storing a block

# HBase DML using shell - create

```
> create 't1', 'f1', 'f2'  
> describe 't1'  
Table t1 is ENABLED  
t1  
COLUMN FAMILIES DESCRIPTION  
{NAME => 'f1', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1',  
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION  
=> 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true',  
BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}  
  
{NAME => 'f2', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1',  
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION  
=> 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true',  
BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}  
  
2 row(s)  
Quota is disabled  
Took 0.2327 seconds
```

# HBase DML using shell - updates / inserts

```
> put 't1', 1, 'f1:name', 'andy'  
> put 't1', 1, 'f1:name', 'ram'  
> put 't1', 1, 'f2:salary', 10000  
> scan 't1'  
  
ROW    COLUMN+CELL  
1      column=f1:name, timestamp=2021-06-27T21:54:17.434, value=ram  
1      column=f2:salary, timestamp=2021-06-27T22:02:22.730, value=10000  
1 row(s)  
  
> put 't1', 2, 'f1:name', 'andy'  
> put 't1', 2, 'f2:salary', 20000  
> scan 't1'  
  
ROW    COLUMN+CELL  
1      column=f1:name, timestamp=2021-06-27T21:54:17.434, value=ram  
1      column=f2:salary, timestamp=2021-06-27T22:02:22.730, value=10000  
2      column=f1:name, timestamp=2021-06-27T22:04:26.820, value=andy  
2      column=f2:salary, timestamp=2021-06-27T22:05:29.310, value=20000  
2 row(s)
```

# HBase DML using shell - filtering

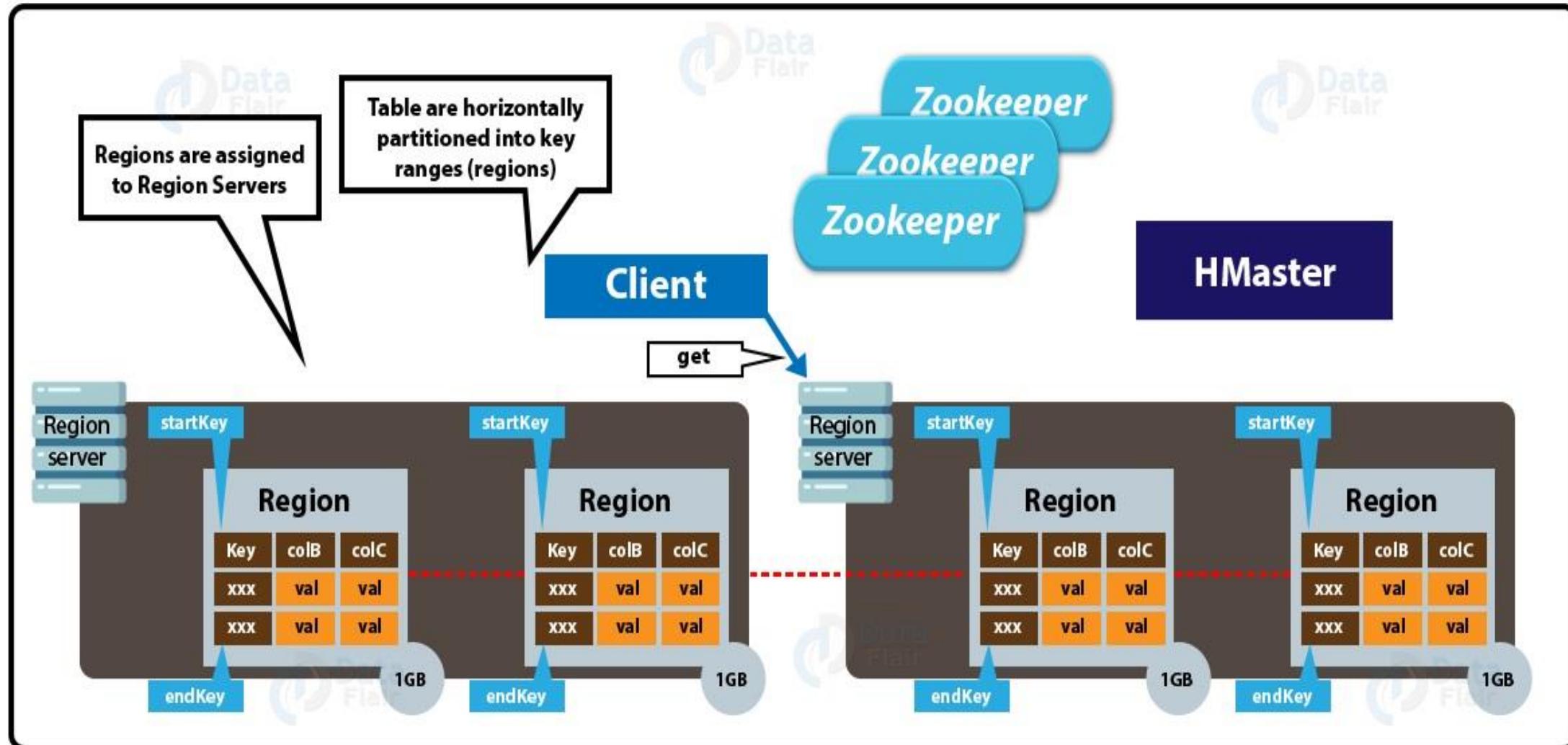
```
> import org.apache.hadoop.hbase.filter.SingleColumnValueFilter  
> import org.apache.hadoop.hbase.filter.CompareFilter  
> import org.apache.hadoop.hbase.filter.BinaryComparator  
  
> scan 't1', {COLUMNS=>['f1:name','f2:salary'],  
FILTER=>SingleColumnValueFilter.new(Bytes.toBytes('f2'),Bytes.toBytes('salary'),Co  
mpareFilter::CompareOp.valueOf('GREATER'),BinaryComparator.new(Bytes.toBytes('1200  
0')))}  
ROW    COLUMN+CELL  
2      column=f1:name, timestamp=2021-06-27T22:04:26.820, value=andy  
2      column=f2:salary, timestamp=2021-06-27T22:05:29.310, value=20000  
1 row(s)  
Took 0.0170 seconds
```

# Hands on with HBase queries

- Hbase configuration
- Walk through of Hbase installation
- Creation of databases
- Creation of tables
- Queries on tables

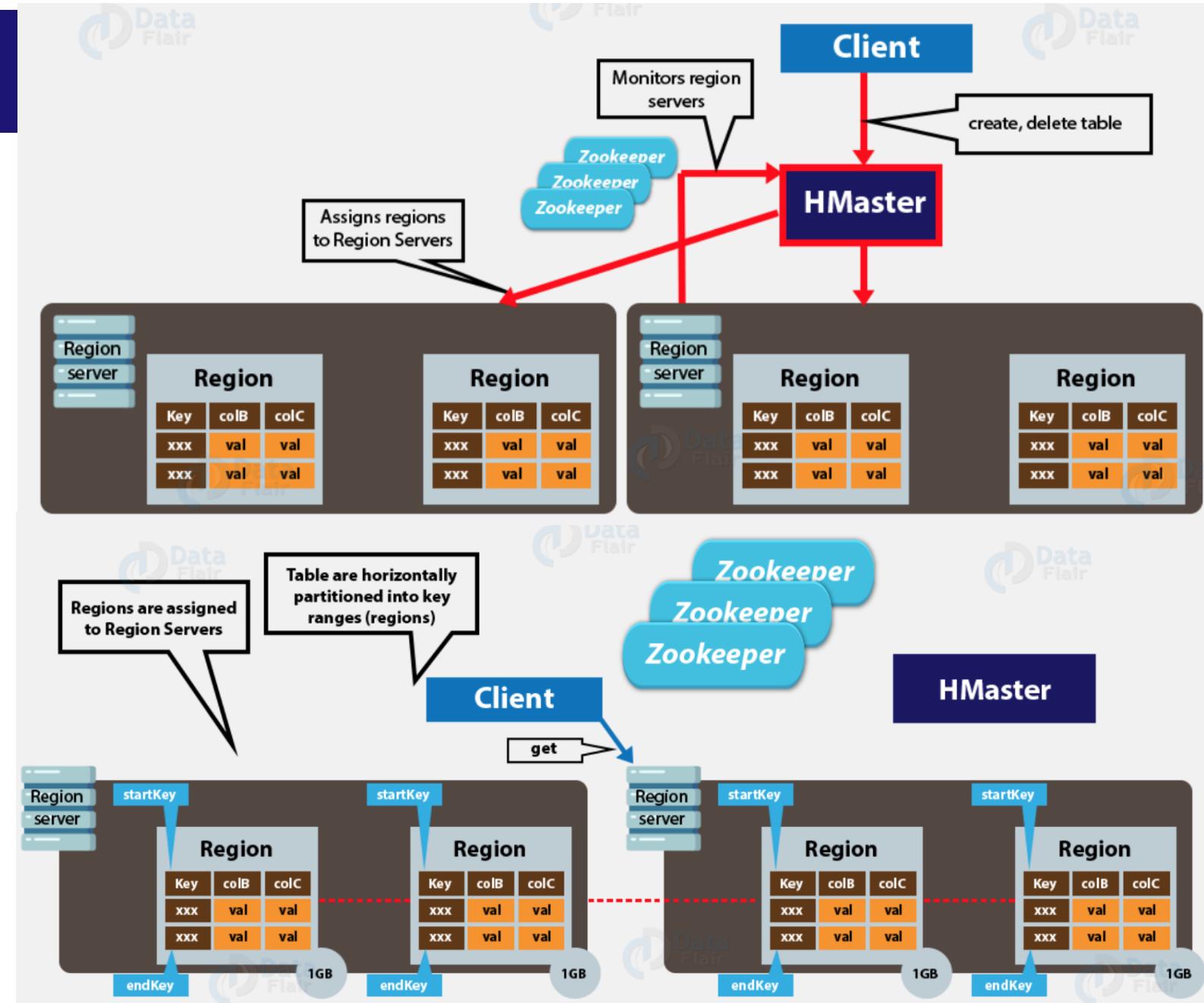


# Hbase Regions



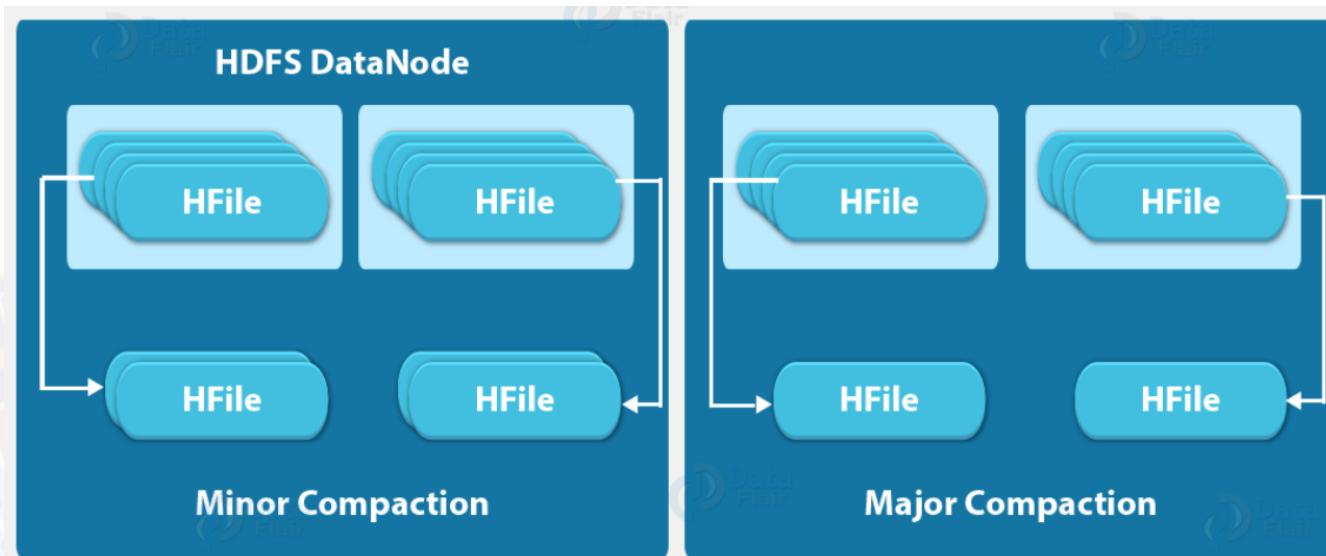
# Client operations

- Create / Delete via HMaster
- Meta-data from HMaster
- Actual Read / Write via RegionServer



# File compaction

- Regions get split on the same node when they grow large.
- Typically RegionServers manage all regions on same node.
- Load balancing, HDFS replication, or failure may have RegionServers managing regions with non-local StoreFiles.
- Compaction is about
  - ✓ merging StoreFiles into larger files
  - ✓ merging Regions
  - ✓ making sure RegionServers and corresponding region files are on same node.



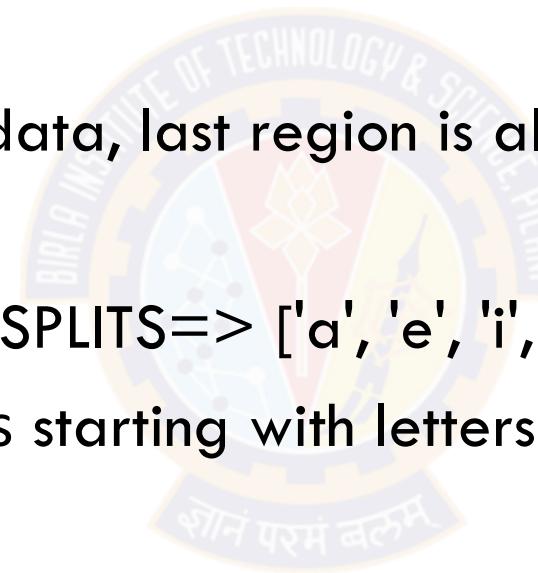
Merge smaller files into  
larger files on same  
ColumnFamily on same node

Merge all files into one for  
same Column Family possibly  
across nodes

Drops deleted cells also

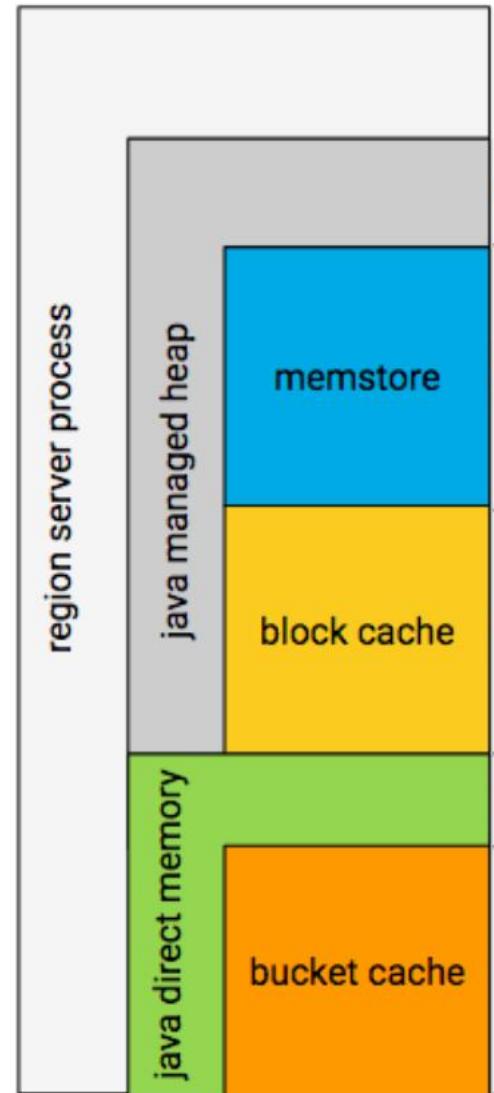
# Splitting tables into Regions

- Automatic splitting happens based on size
- However in some cases, user may want to control splitting
  - ✓ Hot spots for data access
  - ✓ For real-time / time-series data, last region is always active
  - ✓ Load balancing
- `hbase> create 'test_table', 'f1', SPLITS=> ['a', 'e', 'i', 'o', 'u']`
  - ✓ Splitting by sorted row keys starting with letters region1:a-d, region 2:e-h,  
...
- Can also merge regions:
  - ✓ `hbase> merge_region region1 region2`



# Caching on RegionServer

- MemStore is for writes
- Recent edits can also be read from MemStore
- Otherwise read caching is done in BlockCache
- HFile is a collection of Blocks - so Block is the smallest unit of IO at HBase level
- HBase blocks are 4 types
  - ✓ DATA : Contain user data
  - ✓ META : Contain meta-data about the HFile
  - ✓ INDEX, BLOOM: Index DATA blocks to access cell level
- Types of caches in recent HBase
  - ✓ LRUBlockCache : Within JVM Heap - so will be impacted when GC runs
  - ✓ SlabCache, BucketCache : Can use memory outside Heap to avoid GC pressure



# Approximate Sizing guidance

- 10 GB per region
- RegionServer : 100 regions x 10GB = 1 TB
  - ✓ Typical for PB data storage use cases
- Cache size: 10GB : 1% - which may barely cover block indices



# Advantages of HBase

## 1. Strong consistency model

- All readers will see same value, while a write returns.

## 2. Scales automatically

- While data grows too large, Regions splits automatically.
- To spread and replicate data, it uses HDFS.

## 3. Built-in recovery

- It uses Write Ahead Log for recovery.

## 3. Integrated with Hadoop

- On HBase MapReduce is straightforward.

# Summary

Learned basics about

- A. Applicability
- B. Architecture
- C. Data manipulation usage with hands on

of

- ✓ **Pig** - Scripting on top of MapReduce
- ✓ **Hive** - Data warehouse
- ✓ **HBase** - Key-value store , NoSQL database





Next Session:  
More Hadoop ecosystem technologies

# **Introduction to Hive**

## What is Hive?

Hive is a Data Warehousing tool. Hive is used to query structured data built on top of Hadoop. Facebook created Hive component to manage their ever-growing volumes of data. Hive makes use of the following:

1. HDFS for Storage
2. MapReduce for execution
3. Stores metadata in an RDBMS.

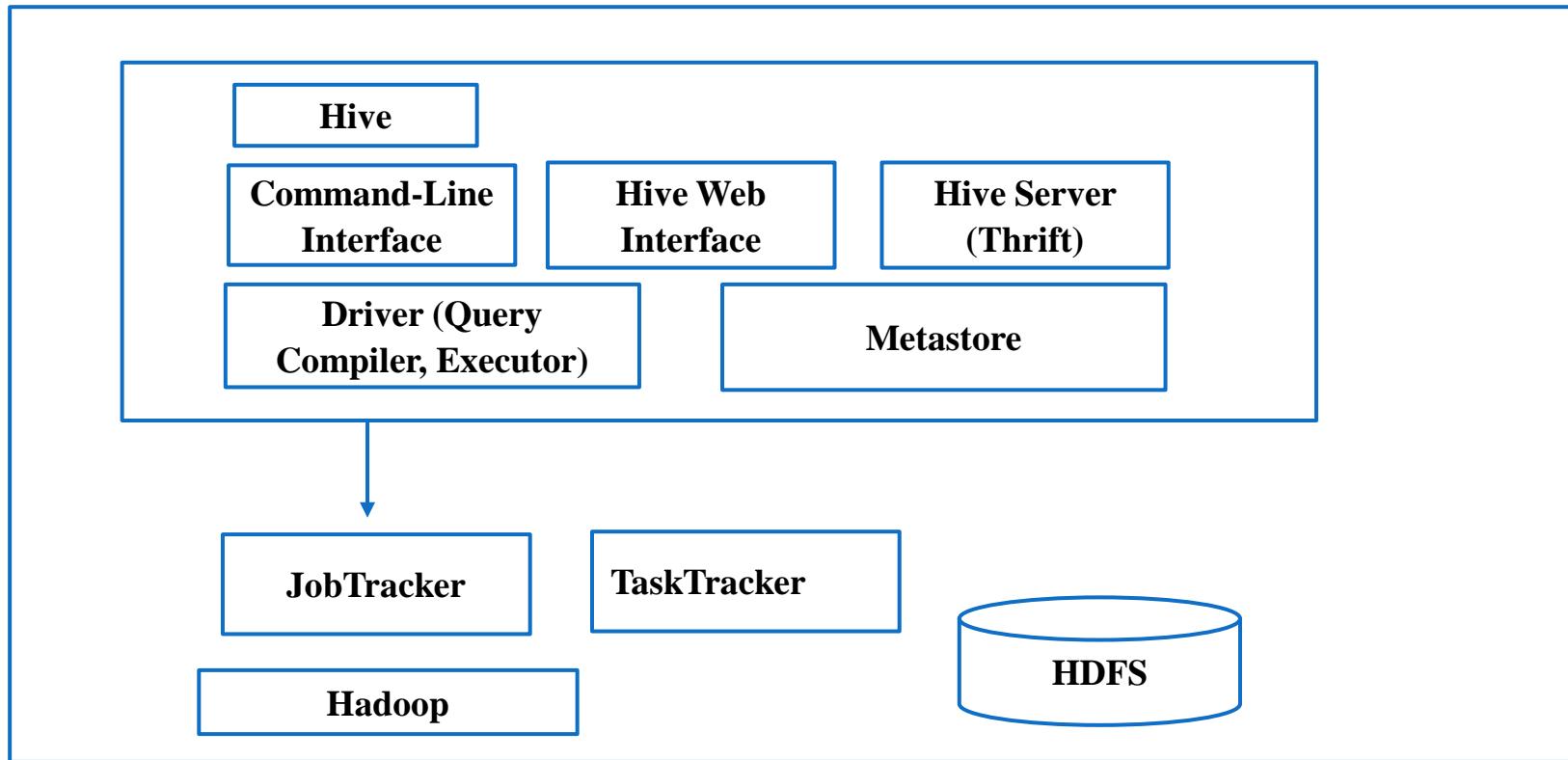
## Features of Hive

1. It is similar to SQL.
2. HQL is easy to code.
3. Hive supports rich data types such as structs, lists, and maps.
4. Hive supports SQL filters, group-by and order-by clauses.

# Hive Data Types

- **Databases**
- **Tables**
- **Partitions**
- **Buckets (Clusters)**

# Hive Architecture



# Hive Data Types

Numeric Data Type	
TINYINT	1 - byte signed integer
SMALLINT	2 -byte signed integer
INT	4 - byte signed integer
BIGINT	8 - byte signed integer
FLOAT	4 - byte single-precision floating-point
DOUBLE	8 - byte double-precision floating-point number

String Types	
STRING	
VARCHAR	Only available starting with Hive 0.12.0
CHAR	Only available starting with Hive 0.13.0
Strings can be expressed in either single quotes (' ) or double quotes ( " )	

Miscellaneous Types	
BOOLEAN	
BINARY	Only available starting with Hive

# Hive Data Types

## Collection Data Types

<b>STRUCT</b>	Similar to 'C' struct. Fields are accessed using dot notation. E.g.: <code>struct('John', 'Doe')</code>
<b>MAP</b>	A collection of key - value pairs. Fields are accessed using [] notation. E.g.: <code>map('first', 'John', 'last', 'Doe')</code>
<b>ARRAY</b>	Ordered sequence of same types. Fields are accessed using array index. E.g.: <code>array('John', 'Doe')</code>

# Hive Query Language (HQL)

- 1. Create and manage tables and partitions.**
- 2. Support various Relational, Arithmetic, and Logical Operators.**
- 3. Evaluate functions.**
- 4. Download the contents of a table to a local directory or result of queries to HDFS directory.**

## Database

To create a database named “STUDENTS” with comments and database properties.

```
CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details'  
WITH DBPROPERTIES ('creator' = 'JOHN');
```

# Database

To describe a database.

**DESCRIBE DATABASE STUDENTS;**

## Database

To drop database.

```
DROP DATABASE STUDENTS;
```

# Tables

Hive provides two kinds of tables:

- ▶ Managed Table
- ▶ External Table

## Tables

To create managed table named ‘STUDENT’.

```
CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

## Tables

To create external table named ‘EXT\_STUDENT’.

```
CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name  
STRING,gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
LOCATION '/STUDENT_INFO';
```

## Tables

To load data into the table from file named student.tsv.

```
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO  
TABLE EXT_STUDENT;
```

## Tables

To retrieve the student details from “EXT\_STUDENT” table.

```
SELECT * from EXT_STUDENT;
```

## Aggregations

Hive supports aggregation functions like avg, count, etc.

To write the average and count aggregation function.

```
SELECT avg(gpa) FROM STUDENT;
```

```
SELECT count(*) FROM STUDENT;
```

## Group by and Having

To write group by and having function.

```
SELECT rollno, name,gpa  
      FROM STUDENT  
      GROUP BY rollno,name,gpa  
      HAVING gpa > 4.0;
```

**Thank you**

# **Introduction to Pig**

# What is Pig?

Apache Pig is a platform for data analysis.

It is an alternative to Map Reduce Programming.

## Pig on Hadoop

- Pig runs on Hadoop.
- Pig uses both Hadoop Distributed File System and MapReduce Programming.
- By default, Pig reads input files from HDFS. Pig stores the intermediate data (data produced by MapReduce jobs) and the output in HDFS.
- However, Pig can also read input from and place output to other sources.

## Features of Pig

- It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
- It provides a language called “**Pig Latin**” to express data flows.
- Pig Latin contains operators for many of the traditional data operations such as join, filter, sort, etc.
- It allows users to develop their own functions (User Defined Functions) for reading, processing, and writing data.

# The Anatomy of Pig

The main components of Pig are as follows:

- Data flow language (**Pig Latin**).
- Interactive shell where you can type Pig Latin statements (**Grunt**).
- Pig interpreter and execution engine.

# Pig Latin Overview

# Pig Latin Statements

Pig Latin Statements are generally ordered as follows:

1. **LOAD** statement that reads data from the file system.
2. Series of statements to perform transformations.
3. **DUMP** or **STORE** to display/store result.

```
A = load 'student' (rollno, name, gpa);
```

```
A = filter A by gpa > 4.0;
```

```
A = foreach A generate UPPER (name);
```

```
STORE A INTO 'myreport'
```

## Pig Latin Identifiers

Valid Identifier	Y	A1	A1_2014	Sample
Invalid Identifier	5	Sales\$	Sales%	_Sales

## Pig Latin Comments

In Pig Latin two types of comments are supported:

1. Single line comments that begin with “—”.
2. Multiline comments that begin with “/\* and end with \*/”.

# Operators in Pig Latin

Arithmetic	Comparison	Null	Boolean
+	= =	IS NULL	AND
-	! =	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

# Data Types in Pig Latin

## Simple Data Types

Name	Description
int	Whole numbers
long	Large whole numbers
float	Decimals
double	Very precise decimals
chararray	Text strings
bytearray	Raw bytes
datetime	Datetime
boolean	true or false

## Complex Data Types

Name	Description
Tuple	An ordered set of fields. Example: (2,3)
Bag	A collection of tuples. Example: {(2,3),(7,5)}
map	key, value pair (open # Apache)

## Running Pig

Pig can run in two ways:

1. Interactive Mode.
2. Batch Mode.

## Execution Modes of Pig

You can execute pig in two modes:

1. Local Mode.
2. Map Reduce Mode.

## Filter

Find the tuples of those student where the GPA is greater than 4.0.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);  
B = filter A by gpa > 4.0;  
DUMP B;
```

## FOREACH

Display the name of all students in uppercase.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = foreach A generate UPPER (name);
```

```
DUMP B;
```

## Group

Group tuples of students based on their GPA.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = GROUP A BY gpa;
```

```
DUMP B;
```

## Distinct

To remove duplicate tuples of students.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
B = DISTINCT A;
```

```
DUMP B;
```

## Join

To join two relations namely, “student” and “department” based on the values contained in the “rollno” column.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);  
  
B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);  
  
C = JOIN A BY rollno, B BY rollno;  
  
DUMP C;  
  
DUMP B;
```

## Split

To partition a relation based on the GPAs acquired by the students.

- GPA = 4.0, place it into relation X.
- GPA is < 4.0, place it into relation Y.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
```

```
SPLIT A INTO X IF gpa==4.0, Y IF gpa<=4.0;
```

```
DUMP X;
```

## **Eval Function**

## Avg

To calculate the average marks for each student.

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as  
(studname:chararray,marks:int);
```

```
B = GROUP A BY studname;
```

```
C = FOREACH B GENERATE A.studname, AVG(A.marks);
```

```
DUMP C;
```

## Max

To calculate the maximum marks for each student.

```
A = load '/pigdemo/student.csv' USING PigStorage(',') as (studname:chararray, marks:int);
```

```
B = GROUP A BY studname;
```

```
C = FOREACH B GENERATE A.studname, MAX(A.marks);
```

```
DUMP C;
```

Map

# Map

**MAP** represents a key/value pair.

To depict the complex data type “map”.

```
John [city#Bangalore]  
Jack[city#Pune]  
James [city#Chennai]
```

```
A = load '/root/pigdemos/studentcity.tsv' Using PigStorage as  
(studname:chararray,m:map[chararray]);
```

```
B = foreach A generate m#'city' as CityName:chararray;
```

```
DUMP B
```

## When to use Pig?

Pig can be used in the following situations:

1. When data loads are time sensitive.
2. When processing various data sources.
3. When analytical insights are required through sampling.

## When NOT to use Pig?

Pig should not be used in the following situations:

1. When data is completely unstructured such as video, text, and audio.
2. When there is a time constraint because Pig is slower than MapReduce jobs.

# Pig Vs. Hive

Features	Pig	Hive
Used By	Programmers and Researchers	Analyst
Used For	Programming	Reporting
Language	Procedural data flow language	SQL Like
Suitable For	Semi - Structured	Structured
Schema / Types	Explicit	Implicit
UDF Support	YES	YES
Join / Order / Sort	YES	YES
DFS Direct Access	YES (Implicit)	YES (Explicit)
Web Interface	YES	NO
Partitions	YES	No
Shell	YES	YES

## Further Readings

- ▶ <http://pig.apache.org/docs/r0.12.0/index.html>
- ▶ <http://www.edureka.co/blog/introduction-to-pig/>

**Thank you**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 9: Hadoop Ecosystem – 2

Flume, Sqoop, Oozie, Zookeeper

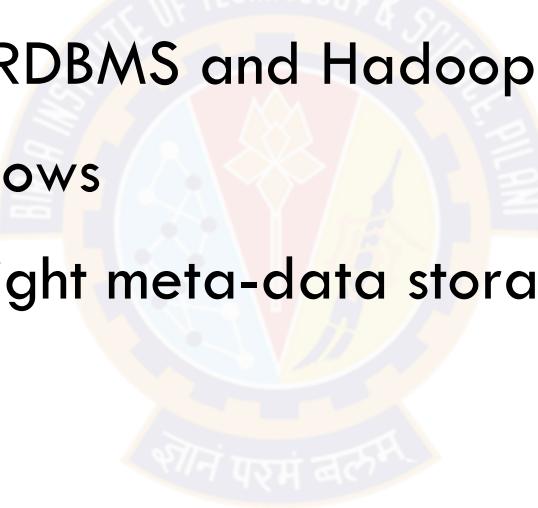
---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- Flume - stream semi-structured data into Hadoop
- Sqoop - move data between RDBMS and Hadoop
- Oozie - scheduling and workflows
- Zookeeper - coordination & light meta-data storage in distributed environment



# Topics for today

- **Flume**
- Sqoop
- Oozie
- Zookeeper



# Data – Lake , Warehouse, Pipeline

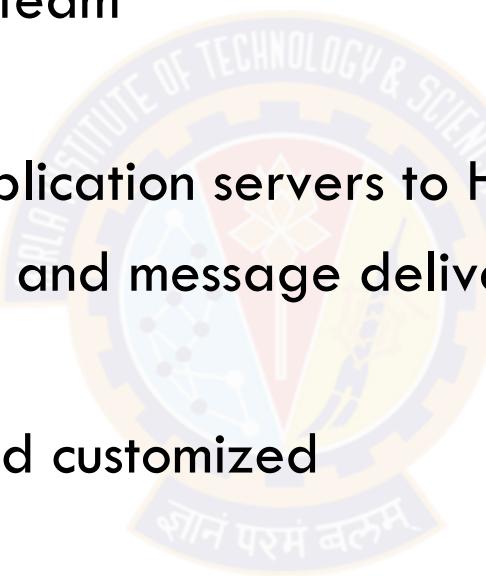
Data Lake		VS		Data Warehouse	
<b>Data</b> 	<b>Users</b> 	<b>Use Cases</b> 		<b>Data</b> 	<b>Users</b> 
Unstructured and Semi-structured	Data Scientists, Data Analysts, Data Engineers	Stream Processing, Machine Learning, Real-time Analysis		Structured	Business Analysts, DBA
<b>Raw</b>			<b>Refined</b>		
Data Lakes contain unstructured, semi-structured and structured data with minimal processing, it can be used to contain unconventional data such as long and sensor data.			Data warehouses contain highly structured data that is cleaned, pre-processed, and refined. This data is stored for very specific use cases, including advanced analytics and reporting.		
<b>Large</b>			<b>Smaller</b>		
Data Lakes contain vast amounts of data in the order of petabytes. Since the data can be in any form or size, large amounts of unstructured data can be stored indefinitely and can be transformed when in use only.			Data warehouses usually store less data than Data Lakes, often in the terabyte range. To ensure data cleanliness and warehouse health, preprocessing and periodic data purging may be required.		
<b>Undefined</b>			<b>Relational</b>		
Data in data lakes can be used for a wide variety of applications, such as Machine Learning, Streaming Analytics, and AI.			Data Warehouses typically contain historical and relational data, such as transaction systems and operational data.		

**Data pipeline** is a method in which raw data is **ingested** from various data sources and then ported to data store, like a data lake or data warehouse

## What are Data Pipelines

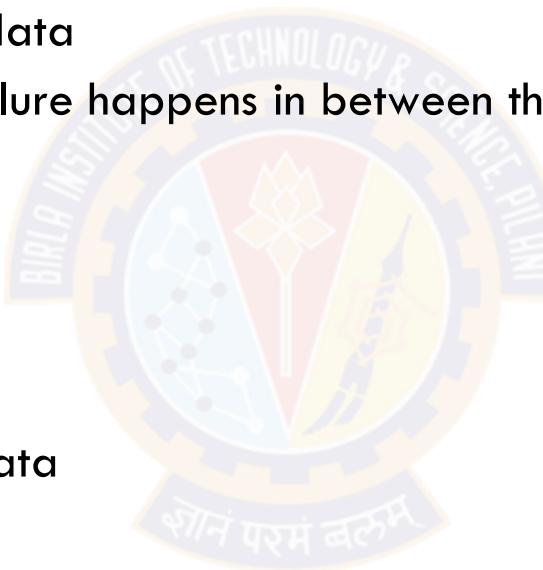
# Flume Introduction (1)

- An e-retailer generates customer browse and buy logs on its website
- These logs need to be streamed into a Hadoop environment for analysis of customer behavior by the data analytics team
- Flume can be used:
  - ✓ Moves logs from source application servers to HDFS files
  - ✓ Performs reliable buffering and message delivery for occasional data spikes
  - ✓ Scales with workload
  - ✓ Can be easily managed and customized
  - ✓ Parallel transfer support
  - ✓ Contextual routing



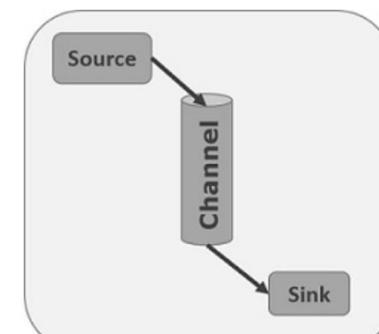
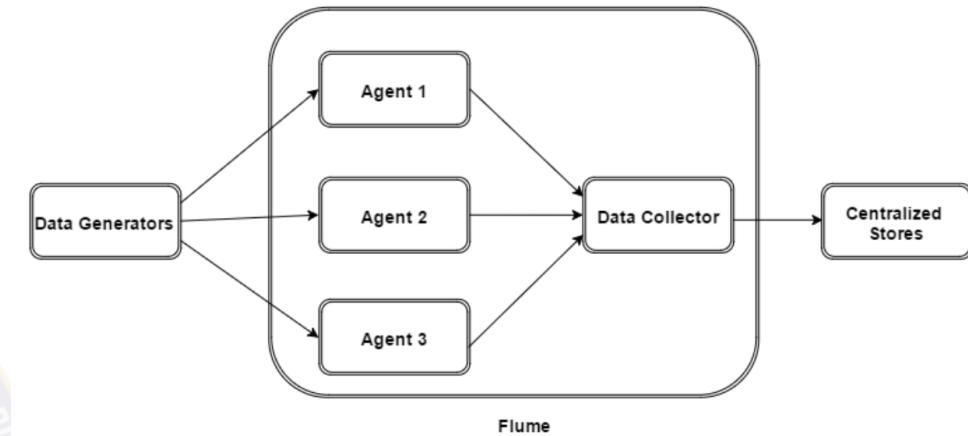
# Flume Introduction (2)

- Why can't we simply do an HDFS 'put' ?
  - ✓ It is a one-time command - we need to reliably stream continuously generated data
  - ✓ Batch mode processing of data in large chunks, may be at specific intervals
  - ✓ Can be used for non-time-sensitive data
  - ✓ If data is partially copied or if a failure happens in between the HDFS file is not closed and size is 0
- Solution: Apache Flume
  - ✓ Streaming data pipeline
  - ✓ Processes data in Real-time
  - ✓ Commonly used for Time sensitive data
    - Financial Transactions
    - Social media feeds
    - Web logs
    - Etc.



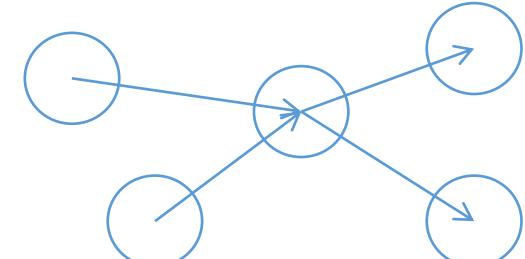
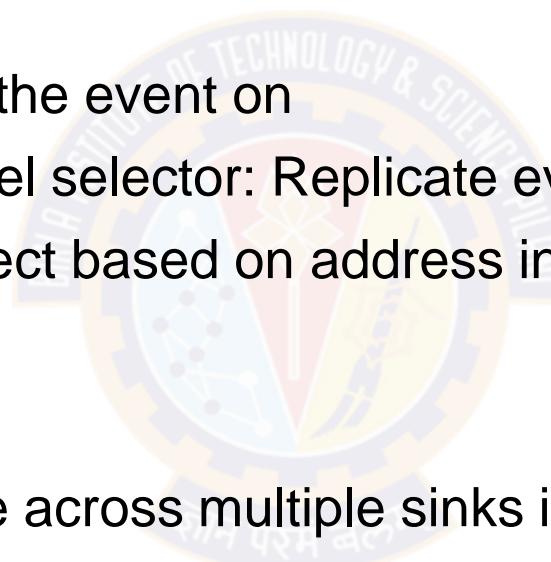
# Architecture – Flume agents

- Agents move data from input to output – (possible to other agents also)
  - ✓ Contain source, channel, sink
  - ✓ JVM process
  - ✓ Source: Receives data from generator or another agent, specific types (e.g. Avro source)
  - ✓ Sink: Consumes from channel and stores in Central store (e.g. HDFS, Hive, HBase sinks) or sends to another agent
  - ✓ Channel: “Transactional” transient data store to act as a bridge between “1 or more” sources and sinks (e.g. JDBC, memory, filesystem channels)
- Flume is event driven
  - ✓ Event: Basic unit of data transfer
- Reliable messaging: For each event, 2 transactions happen. One at sender and second at receiver. Sender commits after receiver commits.

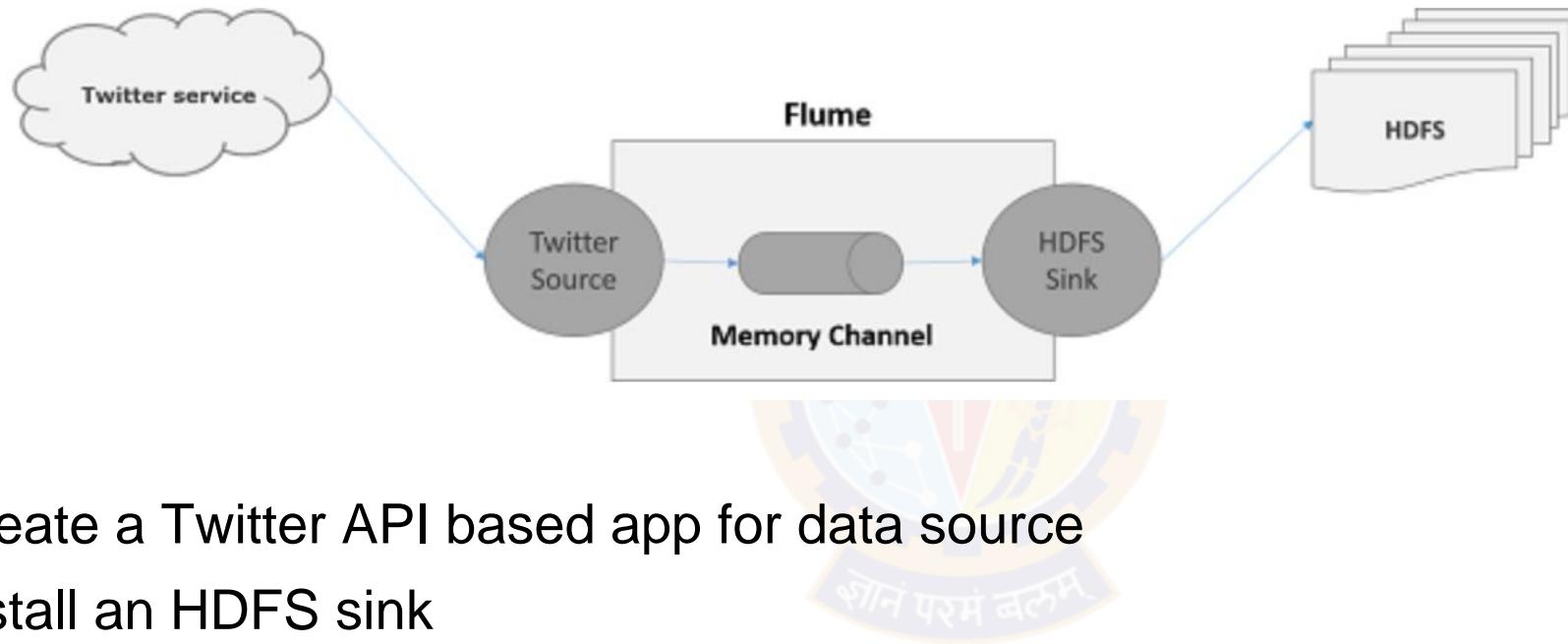


# Other components

- Interceptors
  - ✓ Inspect / Alter events between source and channel
- Channel selectors
  - ✓ Select which channel to send the event on
    - Replicating default channel selector: Replicate event to all channels
    - Multiplexing selector : Select based on address in header
- Sink Processors
  - ✓ Select a sink in a group
  - ✓ Used for failover, load balance across multiple sinks in a channel
- Using these components Flume data flows can have various topologies
  - ✓ Multi-hop, Fan-in, Fan-out



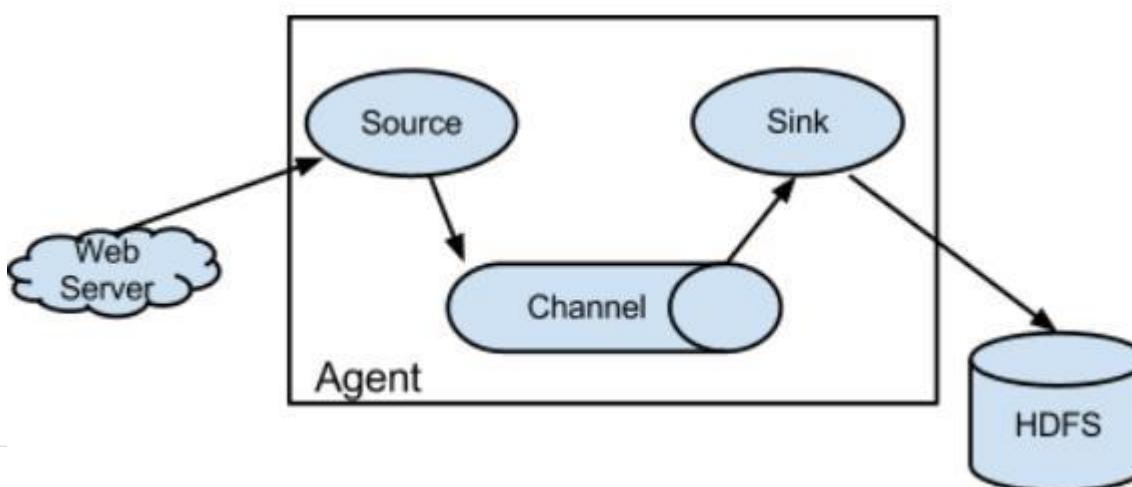
# Example



- Create a Twitter API based app for data source
- Install an HDFS sink
- Setup Flume and configure it with the app as a source and HDFS as the sink

# Sources , Channels and Sinks

SOURCES	CHANNELS	SINKS
Avro Source	Memory Channel	HDFS Sink
Thrift Source	JDBC Channel	Hive Sink
Exec Source	Kafka Channel	Logger Sink
JMS Source	File Channel	Avro Sink
Spooling Directory Source	Spillable Memory Channel	Thrift Sink
Twitter 1% firehose Source	Pseudo Transaction Channel	IRC Sink
Kafka Source		File Roll Sink
NetCat Source		Null Sink
Sequence Generator Source		HBaseSink
Syslog Sources		AsyncHBaseSink
Syslog TCP Source		MorphlineSolrSink
Multiport Syslog TCP Source		ElasticSearchSink
Syslog UDP Source		
HTTP Source		
Stress Source		
Legacy Sources		
Thrift Legacy Source		
Custom Source		
Scribe Source		



# Example

To create a Twitter API based app for data source.

To Setup Flume and configure an agent to stream data from Twitter to HDFS

Steps:

1. Naming the components on the current agent.

- ✓ TwitterAgent.sources = Twitter
- ✓ TwitterAgent.channels = MemChannel
- ✓ TwitterAgent.sinks = HDFS

2. Describing/Configuring the source

3. Describing/Configuring the sink

4. Describing/Configuring the channel

5. Binding the source and sink to the channel

- ✓ TwitterAgent.sources.Twitter.channels = MemChannel
- ✓ TwitterAgent.sinks.HDFS.channel = MemChannel



# Channel Configuration Parameters

- Events have a predefined maximum size
- Events get transferred from source to sink in transactions
- Transactions consists of a predefined number of events
- Sources and sinks can have a batch size parameter that determines the maximum number of events they process in one batch.
- This happens within a channel transaction that has an upper limit called transaction capacity.
- Batch size must be smaller than the channel's transaction capacity.

## Sample settings

--Sink:

eventSize 1024 Maximum size of a single event line in bytes

batchSize 20 Maximum number of lines to read and send to the channel at a time

batchTimeout 3000 Amount of time (in milliseconds) to wait, if the buffer size was not reached,  
before data is pushed downstream

– Memory channel:

Capacity 1000 - Maximum number of events stored in-memory queue.

Set this to zero if you want to disable the use of an in-memory queue.

transactionCapacity 100 - Maximum number of events that the channel will take from a flume source  
or give it to a flume sink per transaction.

# Simple - Test

- Install Flume
- Create a config for an agent
- Create Spool directory - `mkdir -p /home/jps/spool`
- Run flume
  - ✓ `sudo flume-ng agent -c . -f flume.conf -n a1 -Dflume.root.logger=INFO,console`
- Create logs at source
  - ✓ `sudo sh -c 'echo "hello flume" > /home/jps/spool/test.log'`
- View output at the logger sink

```
# Specify the component name of the Agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Specify Flume source (path to be monitored)
a1.sources.r1.type = spooldir
a1.sources.r1.spoolDir = /home/jps/spool

# Specify Flume sink
a1.sinks.k1.type = logger

# Specify Flume channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind source and sink to channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

# Kafka vs Flume

	<b>Kafka</b>	<b>Flume</b>
Base function	Ingestion and processing of streaming data in real-time	Ingestion of streaming log data into a data store
Scale	More scalable	Less scalable
Data movement	Pull based (pub-sub system)	Push based
Recovery	Resilient to failures, can auto-recover	Events may be lost on agent failure
Flexibility	General purpose streaming systems	Tuned for Hadoop ecosystem

\* Flume can have a 'Kafka Channel' which stores events in a Kafka cluster

# Hands on with Flume – Streaming files to HDFS

To stream files added to a local folder to an HDFS folder

- Configure a flume agent with sink type hdfs
- Launch the agent
- Copy files to the local folder
- Check whether they are getting streamed to the HDFS folder



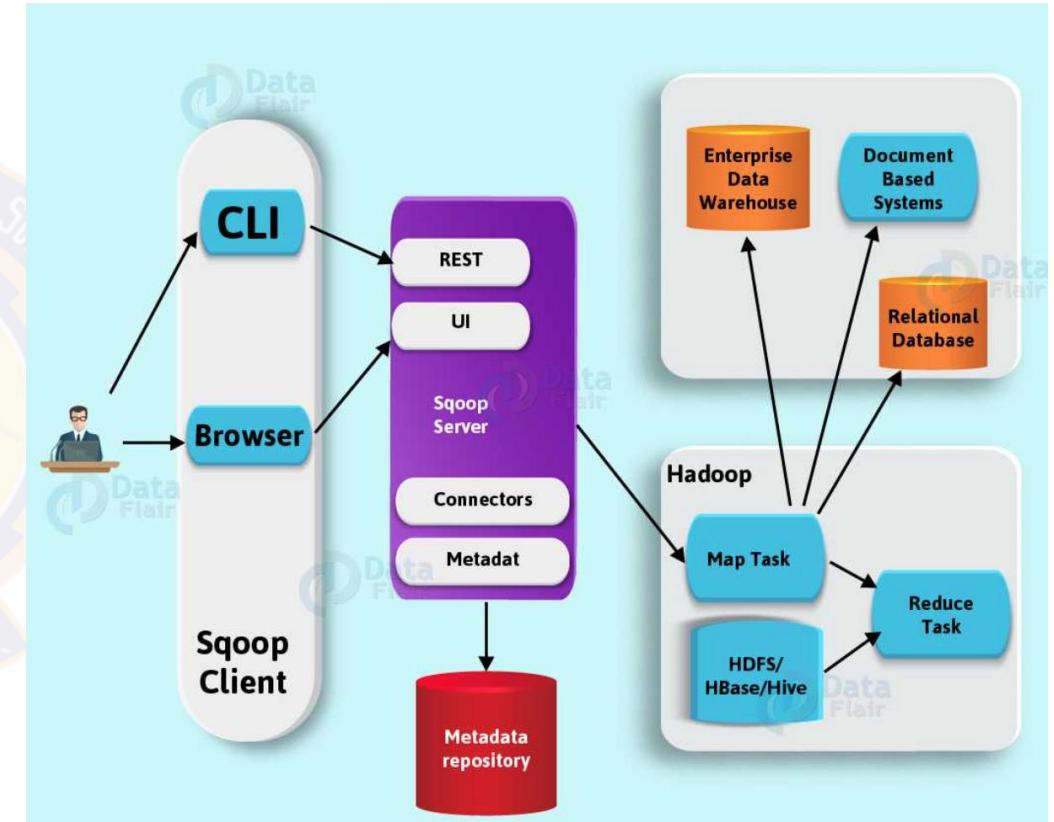
# Topics for today

- Flume
- Sqoop
- Oozie
- Zookeeper



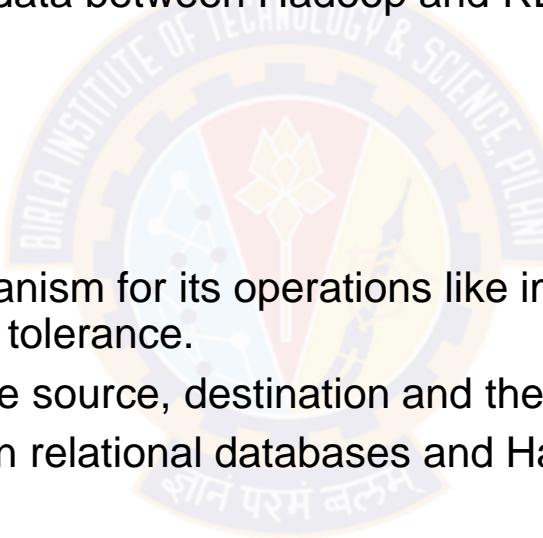
# Sqoop Introduction

- “Sql to Hadoop & Hadoop to SQL”
- A tool in Hadoop ecosystem which is designed to transfer data between HDFS (Hadoop storage) and RDBMS
  - ✓ like MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc.
- Efficiently transfers bulk data between Hadoop and external data stores such as enterprise data warehouses, relational databases, etc.
- Supports import and export operations
  - ✓ import data from relational DB such as MySQL, Oracle.
  - ✓ export data from HDFS to relational DB.



# Why Sqoop?

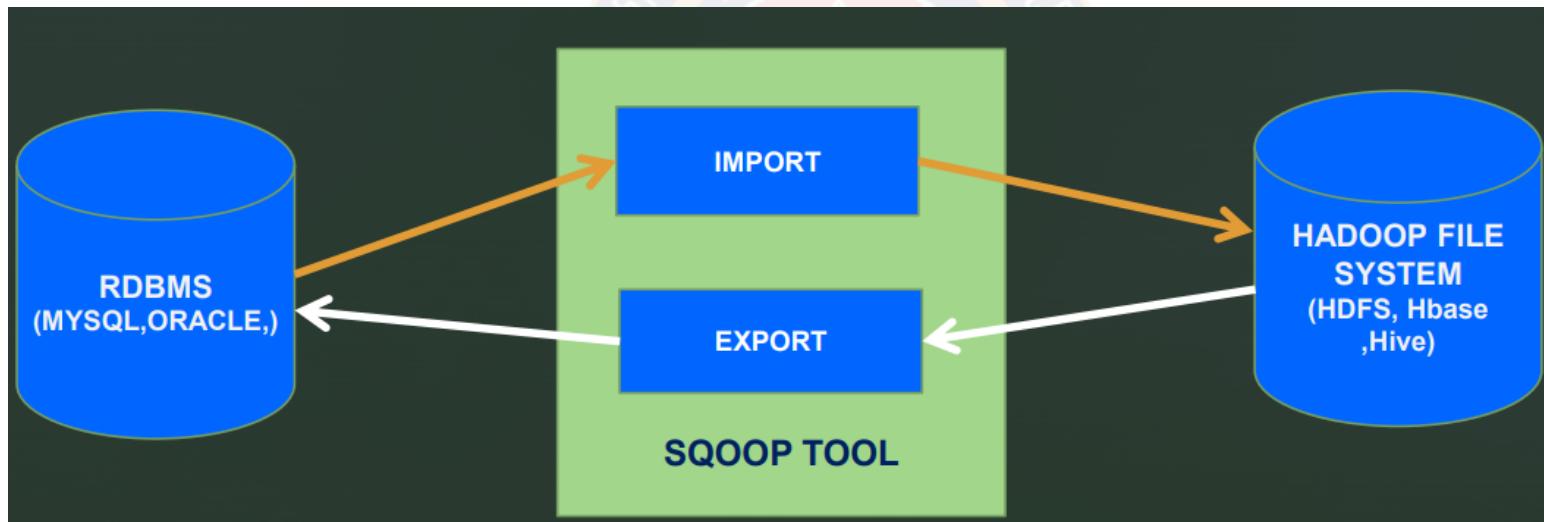
- Big data developer's
  - ✓ work starts once the data is in Hadoop system like in HDFS, Hive or Hbase
  - ✓ Performs magical stuff to find all the golden information hidden on such a huge amount of data
  - ✓ Used to write to import and export data between Hadoop and RDBMS
- Hence a tool was needed !
- Solution came in form of Sqoop
  - ✓ Sqoop uses the MapReduce mechanism for its operations like import and export work and work on a parallel mechanism as well as fault tolerance.
  - ✓ Developers just need to mention the source, destination and the rest of the work will be done by Sqoop
  - ✓ Filled the data transfer gap between relational databases and Hadoop system



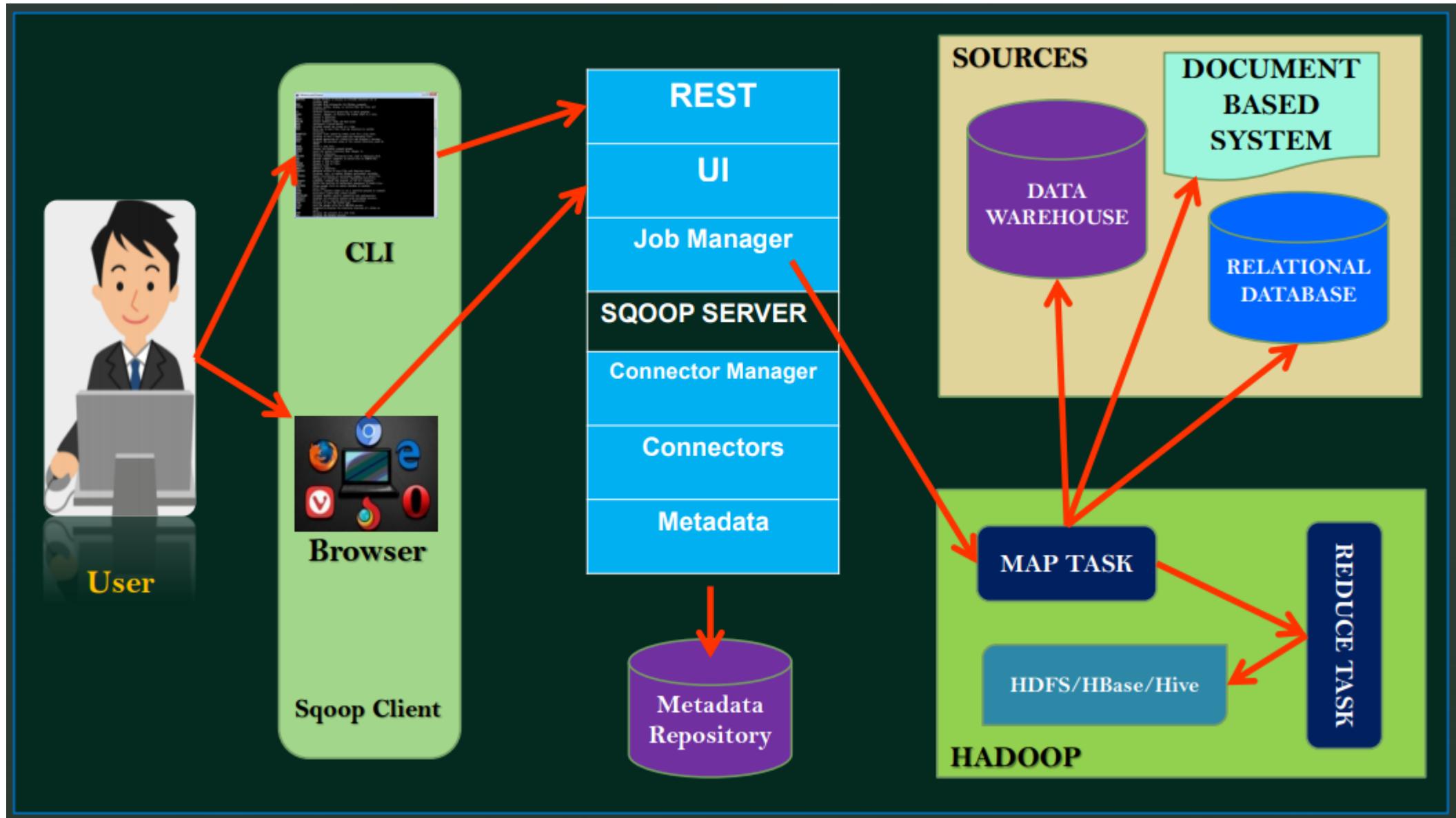
**APACHE SQUIRREL**

# Sqoop Architecture

- Source which is RDBMS like MySQL
- Destination like Hbase or HDFS or Hive
- Sqoop performs the operation for import and export



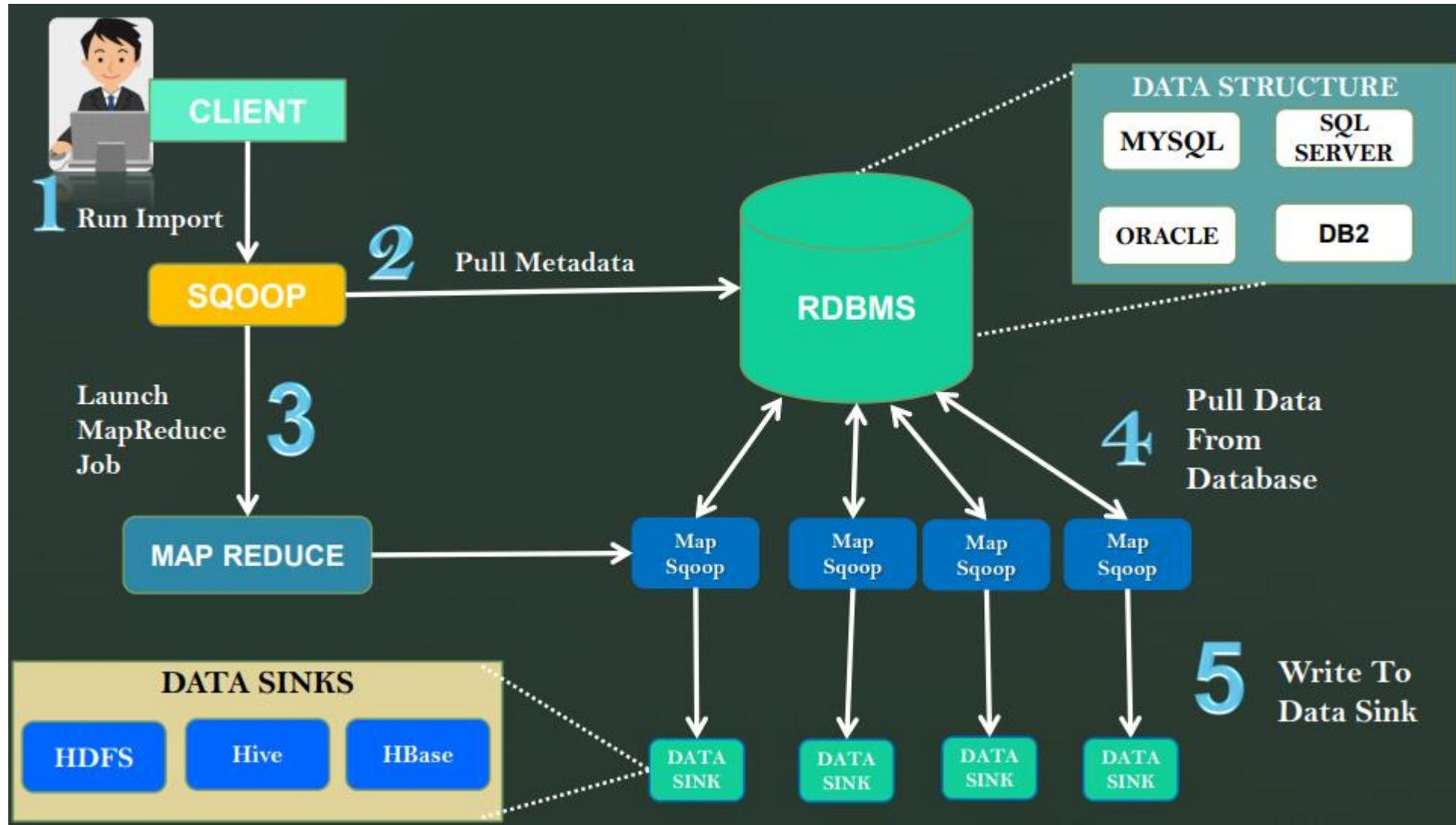
# Sqoop 2 Architecture



# Features of Sqoop

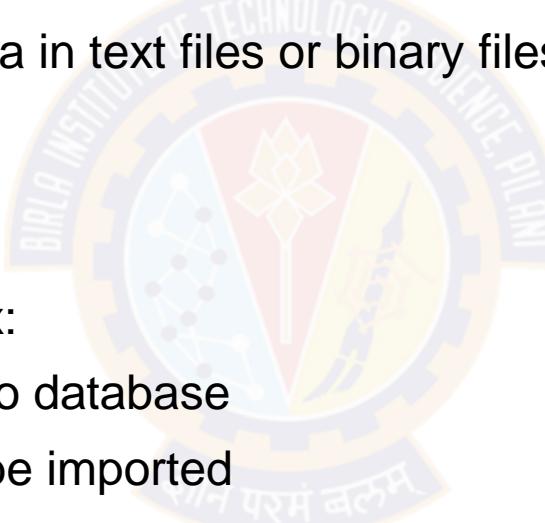
- Full Load
  - ✓ Can load the whole table by a single command
  - ✓ Can also load all the tables from a database using a single command
- Incremental Load
  - ✓ Also provides the facility of incremental load where you can load parts of table whenever it is updated
  - ✓ Supports two types of incremental imports: 1. Append 2. Last modified
- Parallel import/export
  - ✓ Uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism
- Import results of SQL query
  - ✓ Can also import the result returned from an SQL query in HDFS
- Compression
  - ✓ Can compress data by using deflate(gzip) algorithm with –compress argument, or by specifying –compression-codec argument
  - ✓ Can also load compressed table in Apache Hive
- Connectors for all major RDBMS Databases
  - ✓ Provides connectors for multiple RDBMS databases, covering almost the entire circumference
- Load data directly into HIVE/HBase
  - ✓ Can load data directly into Apache Hive for analysis and also dump data in HBase, which is a NoSQL database

# Five stage Sqoop Import Overview

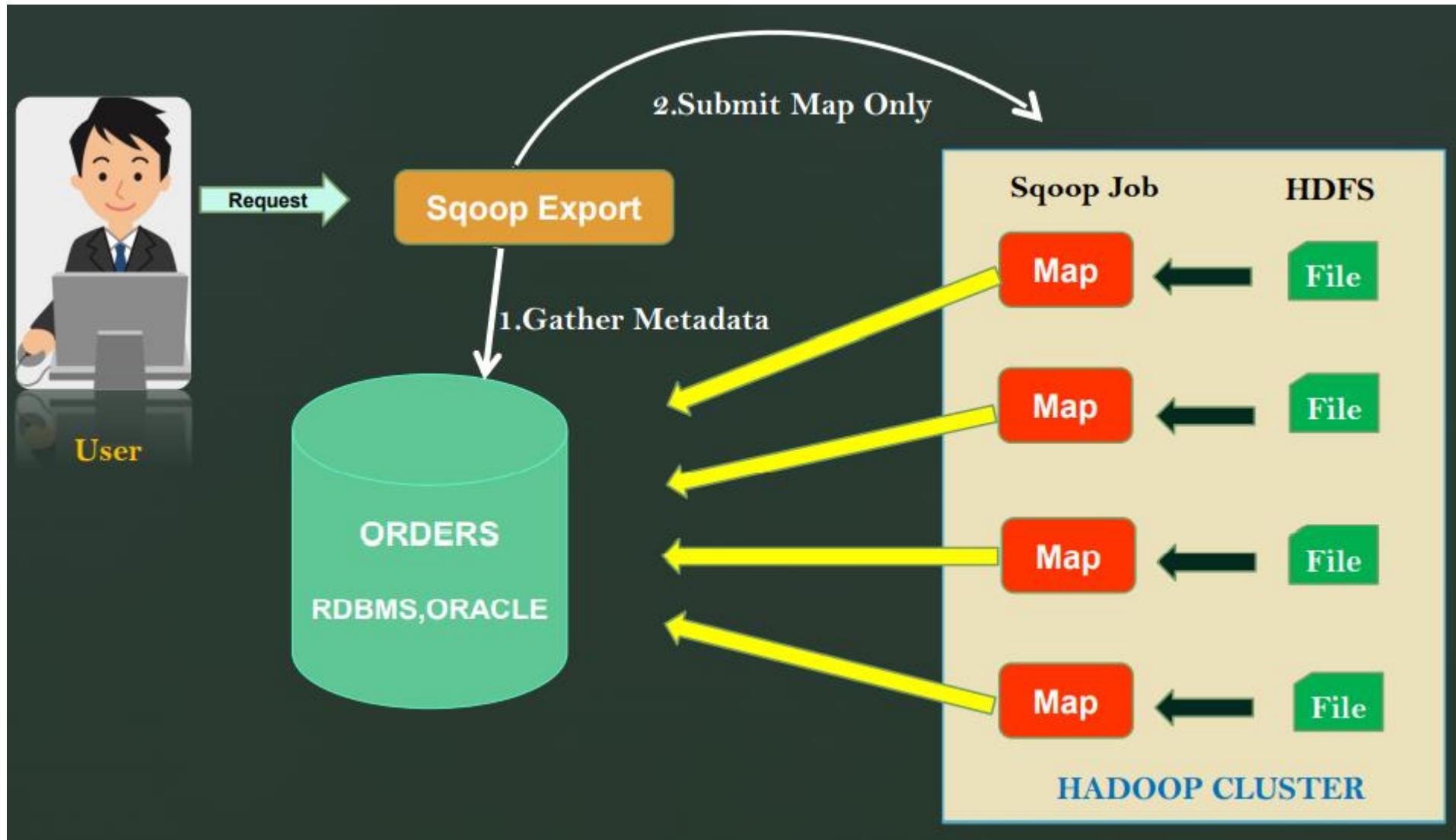


# Importing Data using Sqoop

- SQOOP Import Command
  - ✓ Import individual tables from RDBMS to HDFS
  - ✓ Each row in a table is treated as records in HDFS
  - ✓ All record are stored as text data in text files or binary files
- Generic Syntax:
- Importing a Table into HDFS Syntax:
  - ✓ Takes JDBC url and connects to database
  - ✓ --table - Source table name to be imported
  - ✓ --username - Username to connect to database
  - ✓ --password - Password of the connecting user
  - ✓ --target-dir - Imports data to the specified directory

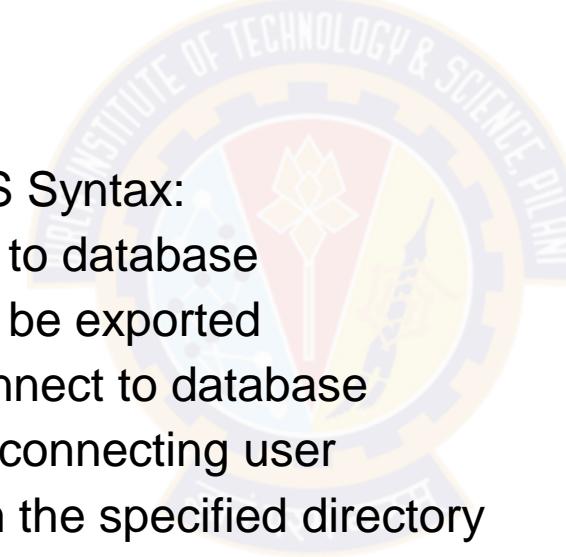


# Exporting Data using Sqoop



# Sqoop Export Data

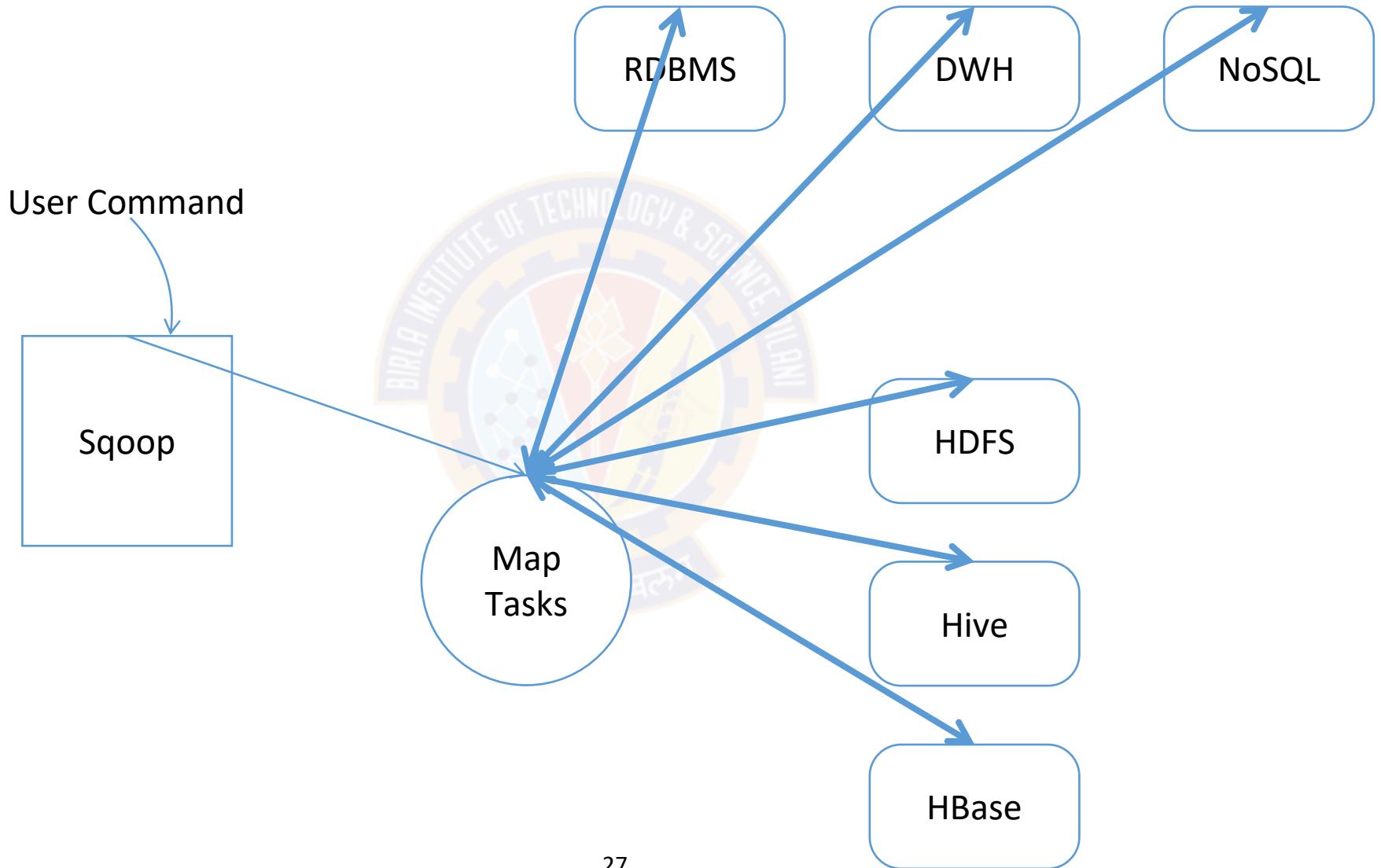
- SQUIOP Export Command
  - ✓ Export a set of files from HDFS back to RDBMS
  - ✓ Files given an input to SQUIOP contains records called as rows in table
- Generic Syntax:
  - ✓ Exporting a Table into RDBMS Syntax:
  - ✓ Takes JDBC url and connects to database
  - ✓ --table - Source table name to be exported
  - ✓ --username - Username to connect to database
  - ✓ --password - Password of the connecting user
  - ✓ --target-dir - Exports data from the specified directory



# Limitations of Sqoop

- **Sqoop cannot be paused and resumed**
  - ✓ an atomic step
  - ✓ If failed, need to clear things up and start again
- Failures need special handling in case of partial import or export
- Sqoop Export performance also depends upon the hardware configuration (Memory, Hard disk) of RDBMS server
- For few databases Sqoop provides bulk connector which has faster performance
  - ✓ Uses a JDBC connection to connect with RDBMS based on data stores, and this can be inefficient and less performance

# Sqoop - Import / Export



# Sqoop commands

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \
--columns "employee_id,dept_id,first_name,last_name,job_title" --split-by dept_id -m 2

$ hadoop fs -ls EMPLOYEES
Found 2 items
-rw-r-r- 1 someuser somegrp 2913511 2021-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00000
-rw-r-r- 1 someuser somegrp 1683938 2021-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00001

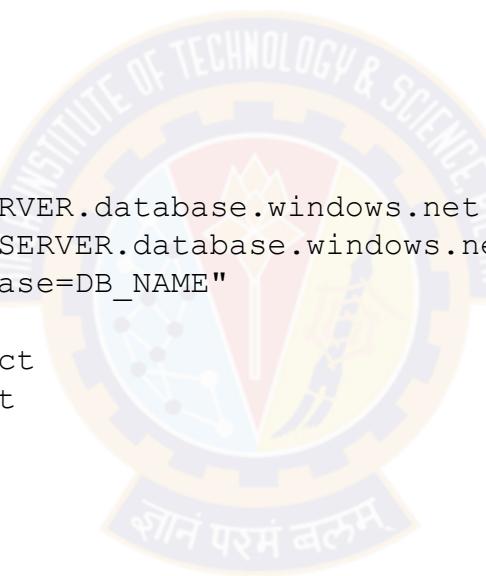
$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar --export-dir /results/bar_data
```

- Other commands : merge data, eval to run SQL etc.

# Commands - export

<https://docs.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-use-sqoop-mac-linux>

```
create table mobiledata (clientid varchar(100), querytime varchar(100), market varchar(100), deviceplatform  
varchar(100),  
devicemake varchar(100), devicemodel varchar(100), state varchar(100),  
country varchar(100),  
querydwelltime varchar(100),  
sessionid bigint,  
sessionpagevieworder bigint)  
  
export serverConnect="jdbc:sqlserver://DBSERVER.database.windows.net:1433;user=USRNAME;password=PASSWORD"  
Export serverDbConnect="jdbc:sqlserver://DBSERVER.database.windows.net:1433;  
user=USRNAME;password=PASSWORD;database=DB_NAME"  
  
sqoop list-databases --connect $serverConnect  
sqoop list-tables --connect $serverDbConnect  
  
sqoop export --connect $serverDbConnect \  
-table mobiledata \  
--hcatalog-table hivesampletable  
  
sqoop eval --connect $serverDbConnect \  
--query "SELECT COUNT(*) from dbo.mobiledata WITH (NOLOCK)"  
  
sqoop eval --connect $serverDbConnect \  
--query "SELECT TOP(10) * from dbo.mobiledata WITH (NOLOCK)"
```



# Commands - import

<https://docs.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-use-sqoop-mac-linux>

```
sqoop import --connect $serverDbConnect \
--table mobiledata \
--target-dir 'wasb:///tutorials/usesqoop/importeddata' \
--fields-terminated-by '\t' \
--lines-terminated-by '\n' -m 1
```

```
hadoop fs -tail /tutorials/usesqoop/importeddata/part-m-00000
```

```
sqoop import --connect $serverDbConnect \
--table mobiledata \
--target-dir 'wasb:///tutorials/usesqoop/importeddata2' \
--fields-terminated-by '\t' \
--lines-terminated-by '\n' \
--create-hive-table \
--hive-table mobiledata_imported2 \
--hive-import -m 1
```

```
beeline -u 'jdbc:hive2://headnodehost:10001/;transportMode=http'
show tables;
describe mobiledata_imported2;
SELECT COUNT(*) FROM mobiledata_imported2;
SELECT * FROM mobiledata_imported2 LIMIT 10;
```

# Sqoop vs Flume

	<b>Sqoop</b>	<b>Flume</b>
Data Flow	Various RDBMS, NoSQL, can map to Hive/HBase from RDBMS	Streaming data, e.g. logs
Loading type	Not event driven	Event driven
Source integration	Connector driven	Agent driven
Usage	Structured sources	Streaming systems with semi-structured data, e.g. twitter feed, web server logs
Performance and reliability	Parallel transfer with high utilisation	Reliable transfer with aggregations at low latency

# Topics for today

- Flume
- Sqoop
- **Oozie**
- Zookeeper



# What is Oozie

Oozie is an extensible, scalable and reliable system to define, manage, schedule, and execute **complex Hadoop workloads** via web services.

More specifically, this includes:

- XML-based declarative framework to specify a job or a complex workflow of dependent jobs
- Support different types of job such as Hadoop Map-Reduce, Streaming, Pig, Hive etc.
- Workflow scheduling based on frequency and/or data availability.
- Monitoring capability, automatic retry and failure handing of jobs.
- Authentication, authorization, and capacity-aware load throttling to allow multi-tenant SAS

# More on Oozie

- Oozie is a server-based Workflow Engine specialized in running workflow jobs with actions that run Hadoop Map/Reduce, Hive and Pig jobs.
- It sequences multiple jobs using a specification input
- Built in Java as a web-application
- Uses Hadoop execution engine to execute tasks
- So, load-balancing, failover is done by Hadoop
- Tasks have a callback URL to tell Oozie when done or can poll the task status

# Understanding Oozie Workflows

- **Actions** – specification of an individual workflow task
- **Transition** – After the completion of a given job or action, the action to be executed next is specified by a transition. Transition also describes the dependency among the actions
- **Workflow** – A collection of actions and transitions arranged in a dependency sequence graph is known as a workflow
- **Workflow application** – An Oozie workflow defined in Oozie workflow language (hPDL) is known as a workflow application.
- **Workflow job** – The Oozie server runs the Oozie workflow definitions, and the Oozie workflow job controls the sequence of these workflow definitions

# Oozie Actions

- Hadoop file system action
- Hadoop shell action
- Hadoop MapReduce action
- Java action
- Pig action
- Sqoop action
- Hive Action
- Oozie sub-workflow action

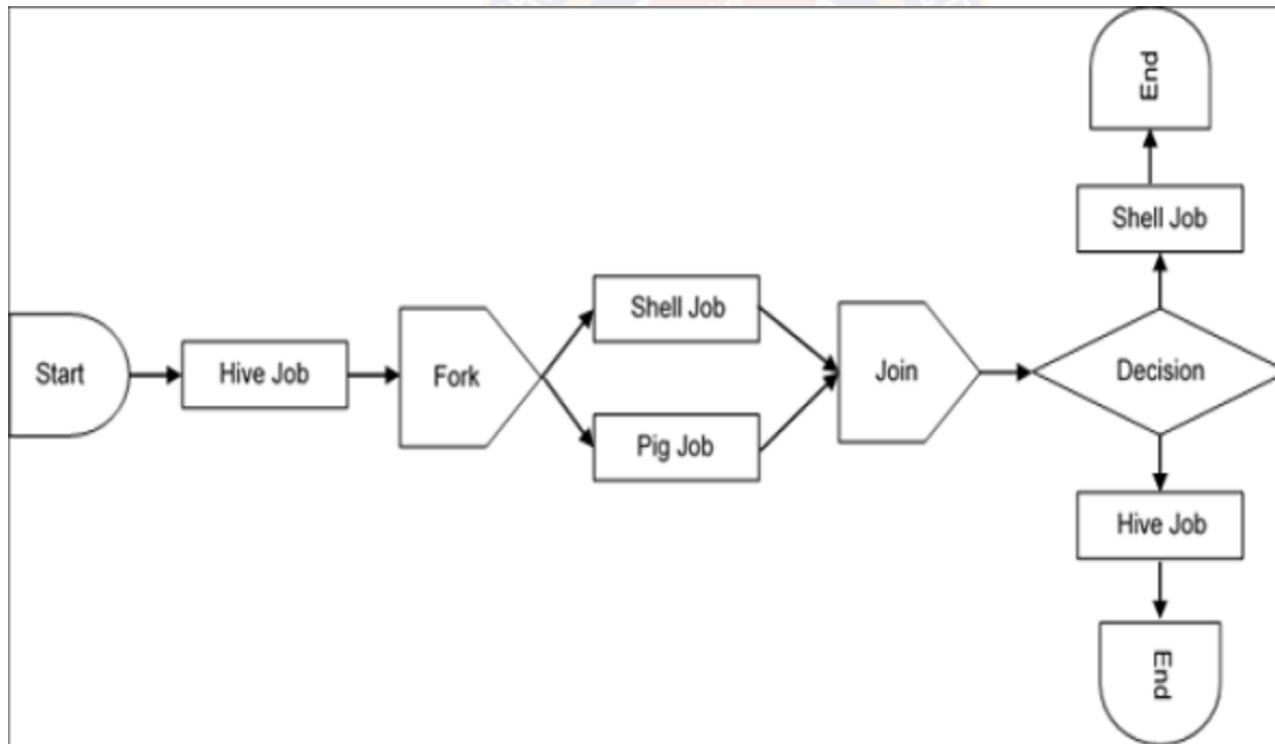


# Oozie workflows

- For the purposes of Oozie, a workflow is a collection of actions (i.e. Hadoop Map/Reduce jobs, Pig jobs) arranged in a control dependency DAG (Directed Acyclic Graph).
- Control dependency from one action to another means that the second action can't run until the first action has completed.
- Oozie workflows definitions are written in hPDL (a XML Process Definition Language)
- Oozie workflow actions start jobs in remote systems (i.e. Hadoop, Pig).
- Upon action completion, the remote systems callback Oozie to notify the action completion, at this point Oozie proceeds to the next action in the workflow.
- Oozie workflows contain control flow nodes and action nodes.
- Control flow nodes define the beginning and the end of a workflow ( start , end and fail nodes) and provide a mechanism to control the workflow execution path ( decision , fork andjoin nodes).
- Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task.
- Oozie provides support for different types of actions: Hadoop map-reduce, Hadoop file system, Pig, SSH, HTTP, eMail and Oozie sub-workflow.
- Oozie can be extended to support additional type of actions.
- Oozie workflows can be parameterized (using variables like \${inputDir} within the workflow definition). When submitting a workflow job values for the parameters must be provided. If properly parameterized (i.e. using different output directories) several identical workflow jobs can concurrently running

# Types of jobs

- Workflow jobs: In a DAG form to specify sequence of actions
- Coordinator jobs: Triggered by time and data availability
- Bundle: Package of workflow and coordinator jobs



# Workflow (1)

```
<workflow-app name = "simple-Workflow">
  <start to = "fork_node" />

  <fork name = "fork_node">
    <path start = "Job1"/>
    <path start = "Job2"/>
  </fork>

  <action name = "Job1">
    <parameters for running Job1 with arguments, settings, executable path>
    <ok to = "join_node" />
    <error to = "kill_job" />
  </action>

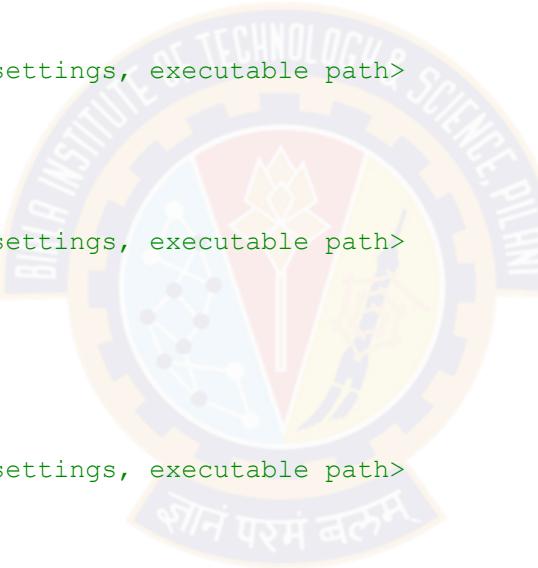
  <action name = "Job2">
    <parameters for running Job1 with arguments, settings, executable path>
    <ok to = "join_node" />
    <error to = "kill_job" />
  </action>

  <join name = "join_node" to = "Job3"/>

  <action name = "Job3">
    <parameters for running Job1 with arguments, settings, executable path>
    <ok to = "end" />
    <error to = "kill_job" />
  </action>

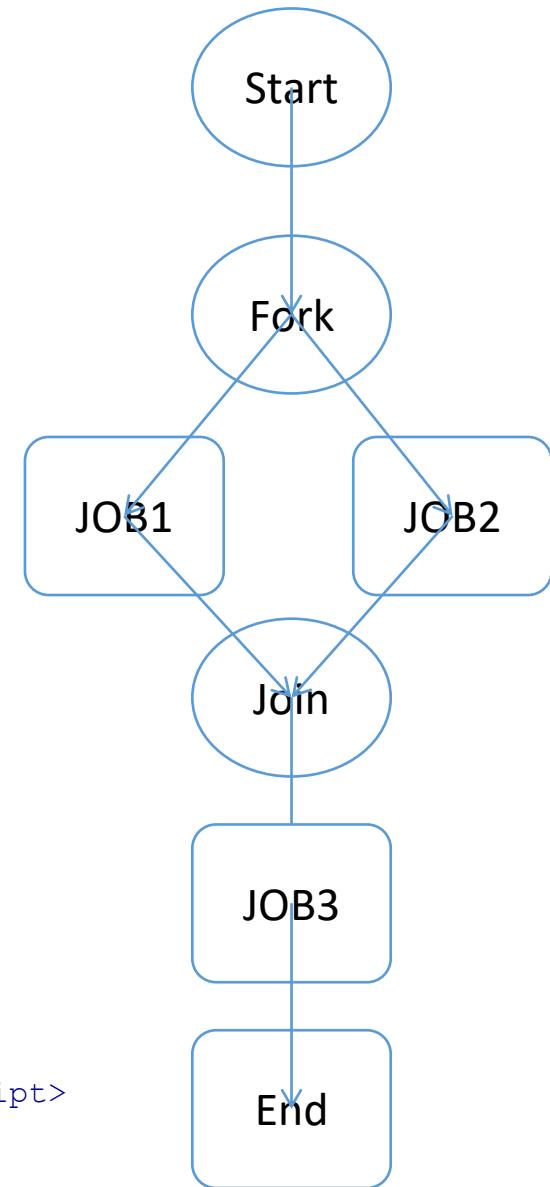
  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>

  <end name = "end" />
</workflow-app>
```



Example job spec :

```
<hive xmlns = "uri:oozie:hive-action:0.4">
  <job-tracker>xyz.com:8088</job-tracker>
  <name-node>hdfs://rootname</name-node>
  <script>hdfs_path_of_script/Copydata.hive</script>
  <param>database_name</param>
</hive>
```



# Workflow (2)

```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">
  <start to = "Decision1" />

  <decision name = "Decision1">
    <switch>
      <case to = "Job1">${fs:exists('/test/abc') eq 'false'}
        </case>
      <default to = "Job2" />
    </switch>
  </decision>

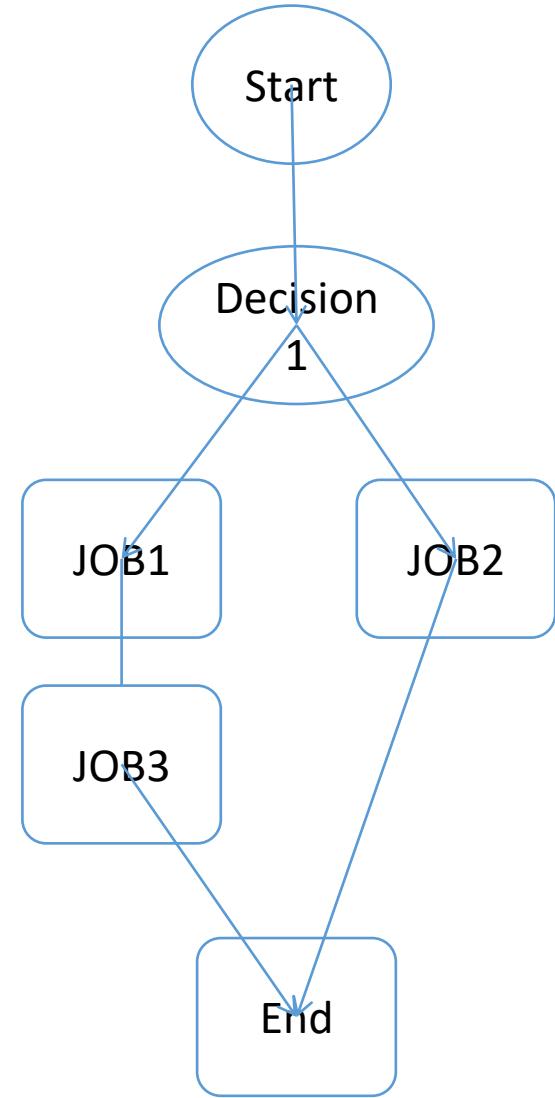
  <action name = "Job1">
    <action details>
      <ok to = "Job3" />
      <error to = "kill_job" />
    </action details>
  </action>

  <action name = "Job2">
    <action details>
      <ok to = "end" />
      <error to = "kill_job" />
    </action details>
  </action>

  <action name = "Job3">
    <action details>
      <ok to = "end" />
      <error to = "kill_job" />
    </action details>
  </action>

  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>

  <end name = "end" />
</workflow-app>
```



# Coordinator

## Base workflow

```
<workflow-app name = "simple-Workflow">
  <start to = "Job1" />

  <action name = "Job1">
    <ok to = "end" />
    <error to = "kill_job" />
  </action>

  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>
  <end name = "end" />

</workflow-app>
```

## Coordinator to run the workflow

```
<coordinator-app name =
  "simple coordinator" frequency = "5 * * * *" start =
  "2021-00-18T01:00Z" end = "2025-12-31T00:00Z" timezone
= "India">

  <controls>
    <timeout>1</timeout>
    <concurrency>1</concurrency>
    <execution>FIFO</execution>
    <throttle>1</throttle>
  </controls>

  <action>
    <workflow>
      <app-path>path to base workflow xml</app-path>
    </workflow>
  </action>

</coordinator-app>
```

# Exercise

- Create 2 PIG jobs using geosales.csv data and run through Oozie. Try workflow and use a coordinator to run the workflow periodically.



# Oozie bundle – Top level abstraction

- Oozie bundle enables users to start, stop, suspend, resume, or rerun a job at the bundle level
- Oozie bundle uses the following concepts:
  - ✓ Kick-off time – This refers to the time a bundle application starts
  - ✓ Bundle action – Refers to the start of a coordinator job of a coordinator application by the Oozie server
  - ✓ Bundle application – Includes some definitions to specify a set of coordinator applications of the bundle.
  - ✓ Bundle job – Bundle job describes all the applications in the Oozie bundle. The content of a bundle job helps the Oozie server to decide which bundle applications are required to be deployed.

# Benefits of Oozie

- **Complex workflow action dependencies** – The Oozie workflow contains actions and dependencies. Users create DAG to model their workflow. At runtime, Oozie manages dependencies and executes actions when the dependencies identified in the DAG are satisfied
- **Reduces Time-To-Market** : Users save on the time to build and maintain custom solutions for dependency and workflow management
- **Frequency execution** – Oozie workflow specification supports both data and time triggers. Users can specify execution frequency and wait for data arrival to trigger an action in the workflow.
- **Native Hadoop stack integration** – Oozie supports all types of Hadoop jobs and is integrated with the Hadoop stack
- **Mechanism to manage a variety of complex data analysis**

# Topics for today

- Flume
- Sqoop
- Oozie
- **Zookeeper**



# Zookeeper

ZooKeeper is essentially a service for distributed systems offering a hierarchical key-value store, which is used to provide a

- Distributed configuration service
- Synchronization service and
- Naming registry for large distributed systems

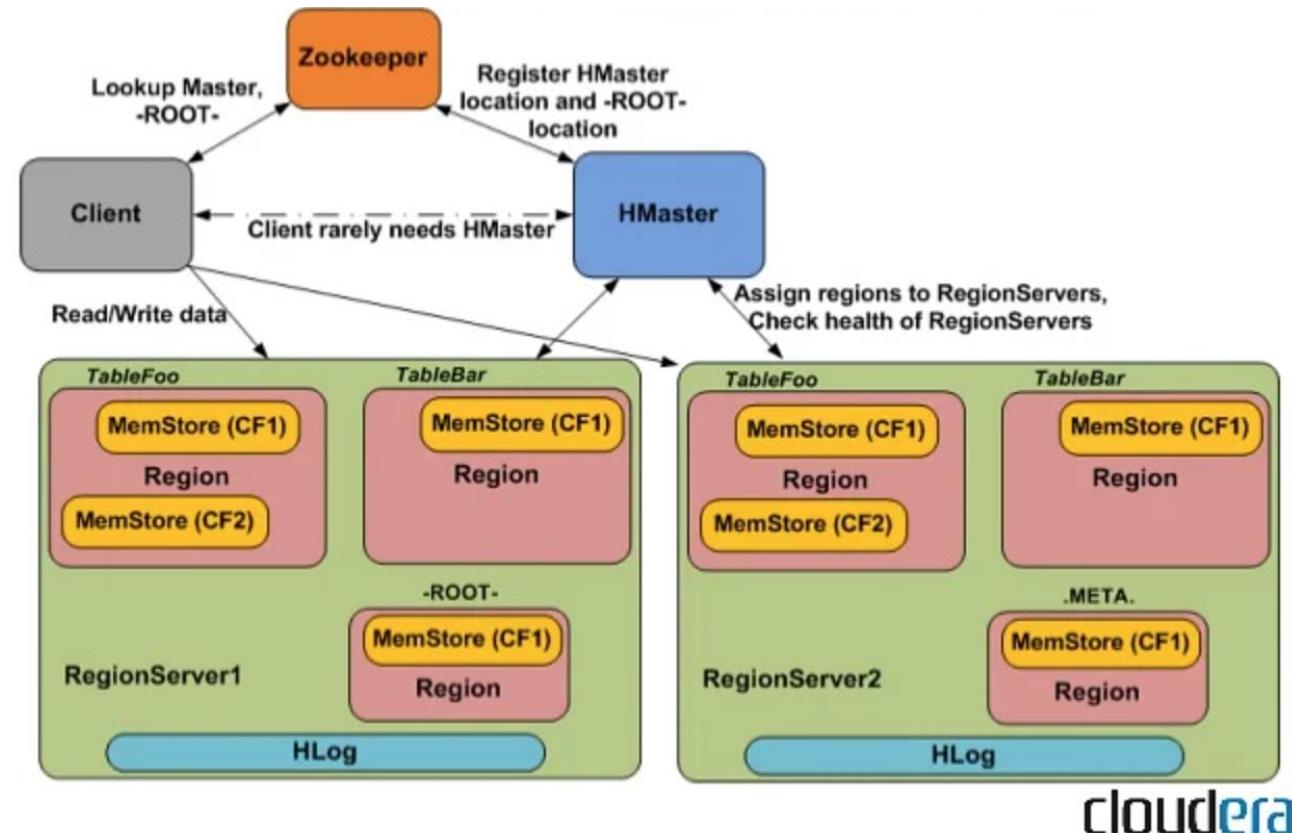


## Apache projects using ZooKeeper

- [Apache Hadoop](#)
- [Apache Accumulo](#)
- [Apache HBase](#)
- [Apache Hive](#)
- [Apache Kafka](#)
- [Apache Drill](#)
- [Apache Solr](#)
- [Apache Spark](#)
- [Apache NiFi](#)
- [Apache Druid](#)
- [Apache Helix](#)
- [Apache Pinot](#)
- [Apache Bookkeeper](#)
- [Apache Pulsar](#)

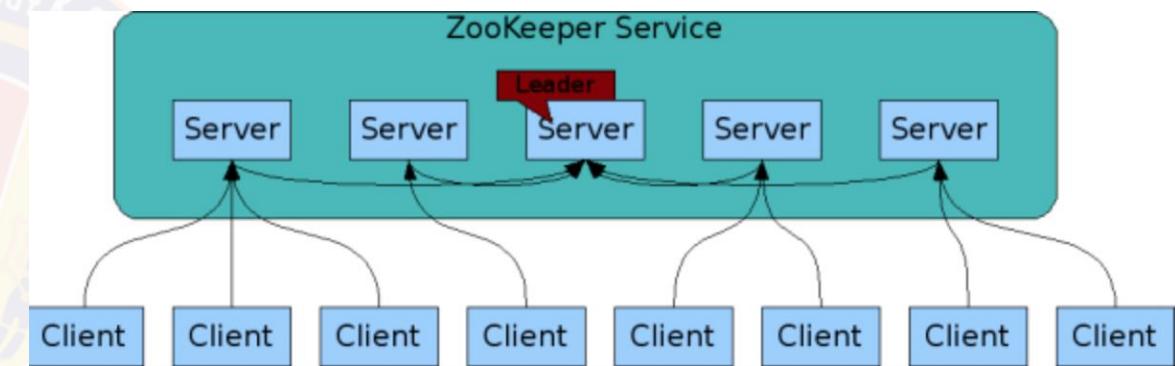
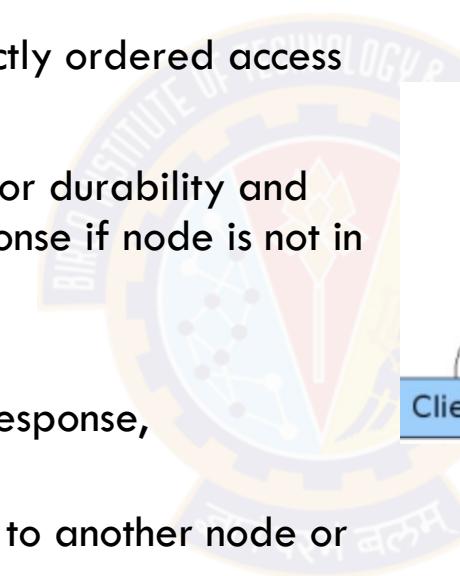
# Zookeeper usage in HBase

- Generic coordination service for distributed systems
- Lightweight store for meta-data, not a replacement of a distributed DB
- Data item updates seen across clients (connected to different nodes) are in same order - Total Ordering
  - ✓ Sequential consistency
- Data updates can be notified to clients
- Used for leader election - using unique node ID (refer to ring based leader election algorithms)
- How Zookeeper is used in HBase cluster
  - ✓ Client needs to lookup current HMaster node
  - ✓ Track failures in cluster with heartbeat and pick a new HMaster
  - ✓ Maintain light weight cluster config data



# Zookeeper Architecture

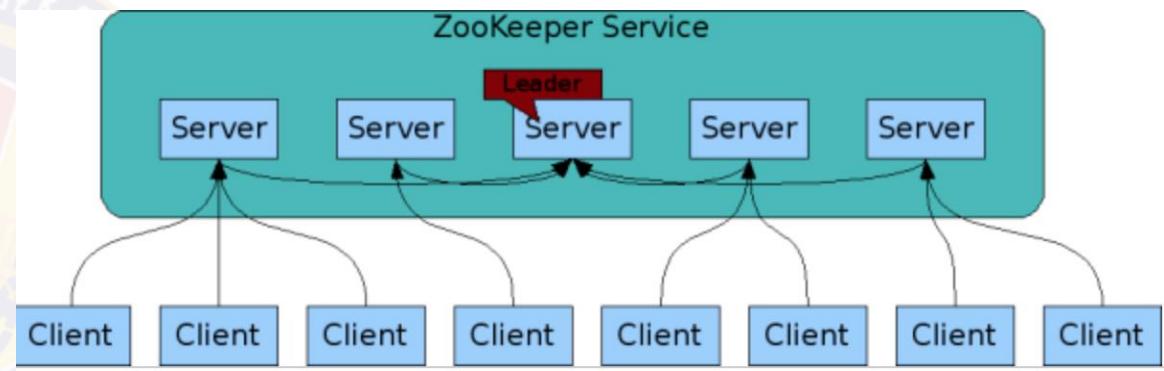
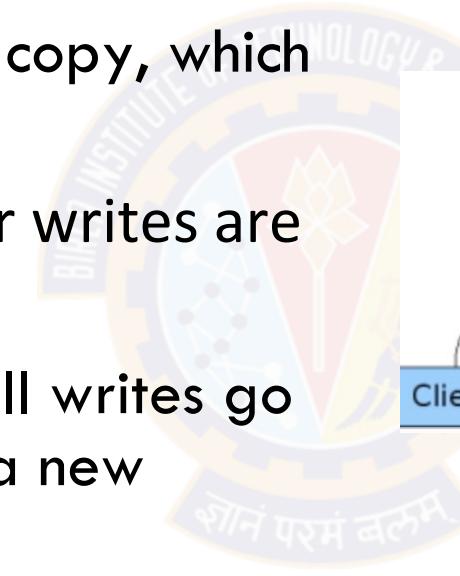
- Built in-memory and as a Java application
  - ✓ Organized like an in-memory file system with dirs/files (called znodes)
- Zookeeper itself is clustered - Ensemble
- High performance, highly fault tolerant, strictly ordered access
  - ✓ It is a CP system as per CAP Theorem
  - ✓ Uses majority quorum within Ensemble for durability and avoid split-brain - so client gets no response if node is not in majority quorum
  - ✓ ZAB\* protocol for consistency
- Clients connect on TCP session for request, response, heartbeats, events etc.
  - ✓ When a connection dies, client connects to another node or node detects client failure
- All client transactions are time stamped and ordered
- Typically meant for read-heavy workloads (10:1)



\* Zookeeper Atomic Broadcast

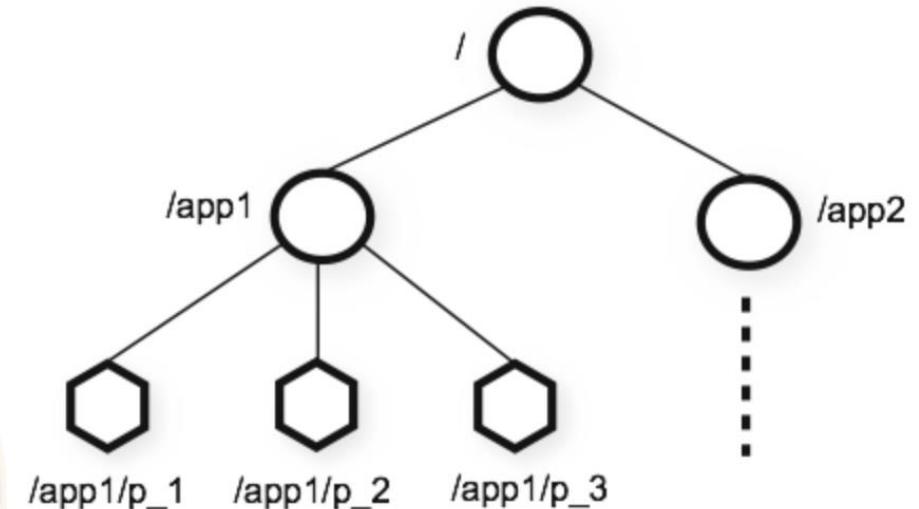
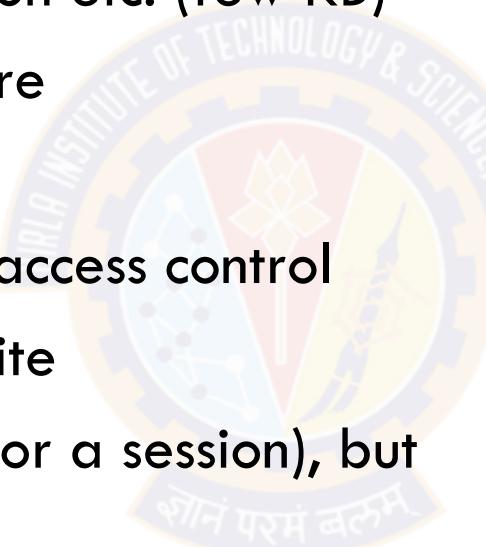
# Reads / Writes

- Replicated in-memory DB
- Updates are logged on disk for recovery
- Reads are serviced from local copy, which ever server client connects to
- Memory copy is updated after writes are on disk
- Messaging layer makes sure all writes go to Leader first and on failure a new Leader is assigned
- All local replicas are kept in sync
- Server process - QuorumPeerMain



# Data model

- Data is organised in a hierarchy with each node called a znode to hold some data
- Data can be status, config, location etc. (few KB)
- znode also contains a stat structure
  - ✓ version, ACL, timestamp
  - ✓ helps to order data and do access control
  - ✓ version updates on every write
- znodes can be ephemeral (only for a session), but these don't have children
- Clients can setup a watch / event for znode changes



# Guarantees

- Sequential Consistency - Updates on a data item from a client will be applied in the order that they were sent and total order across all operations.
- Atomicity - Updates on a data item either succeed or fail. No partial results.
- Single System Image - A client will see the same view of the service regardless of the server that it connects to.
- Reliability - Once an update has been applied on a data item, it will persist from that time forward until a client overwrites the update.
- Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

# Case study: Twitter

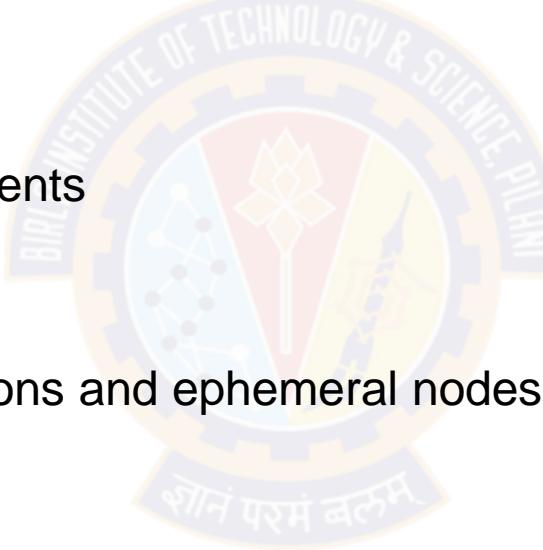
- Stores critical meta-data for coordination services, such as distributed locking, leader election etc.
- Stores a service registry for Twitter's service discovery
- Databases in Twitter (e.g. key-value store, image/video store), store their cluster topology
- Pub-sub messaging and compute platform uses it for leader election



<https://zookeeper.apache.org/doc/current/zookeeperUseCases.html>

# Zookeeper command line interface

- echo stat | nc bdslab1 2181 | grep Mode – find out whether a node is Leader or Follower
- Sample 4lw commands
  - stat - list brief details for the server and connected clients
  - conf - serving configuration
  - cons - connection details for all clients
  - crst - reset connections
  - dump - lists the outstanding sessions and ephemeral nodes (Works on Leader Node)
  - envi - serving environment
  - ruok - I am OK or No response
  - srst - reset server statistics
  - srvr - lists full details of the server



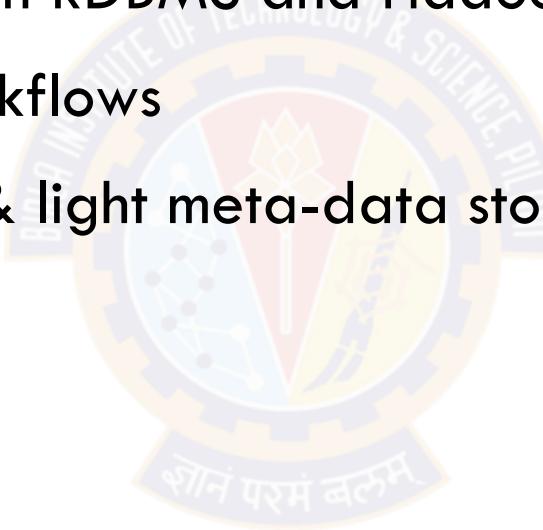
# Zookeeper Hands on

- Walk through of Zookeeper installation
- Simple 4lw commands
- Hands on with zkCli.sh
  - Creating znodes
  - Writing to znodes
  - Setting watch on znode updates
  - Reading from znodes



# Summary

- Flume - stream semi-structured data into Hadoop
- Sqoop - move data between RDBMS and Hadoop
- Oozie - scheduling and workflows
- Zookeeper - coordination & light meta-data storage in distributed environment





Next Session:  
NoSQL Databases



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 10: NoSQL Databases

---

Janardhanan PS

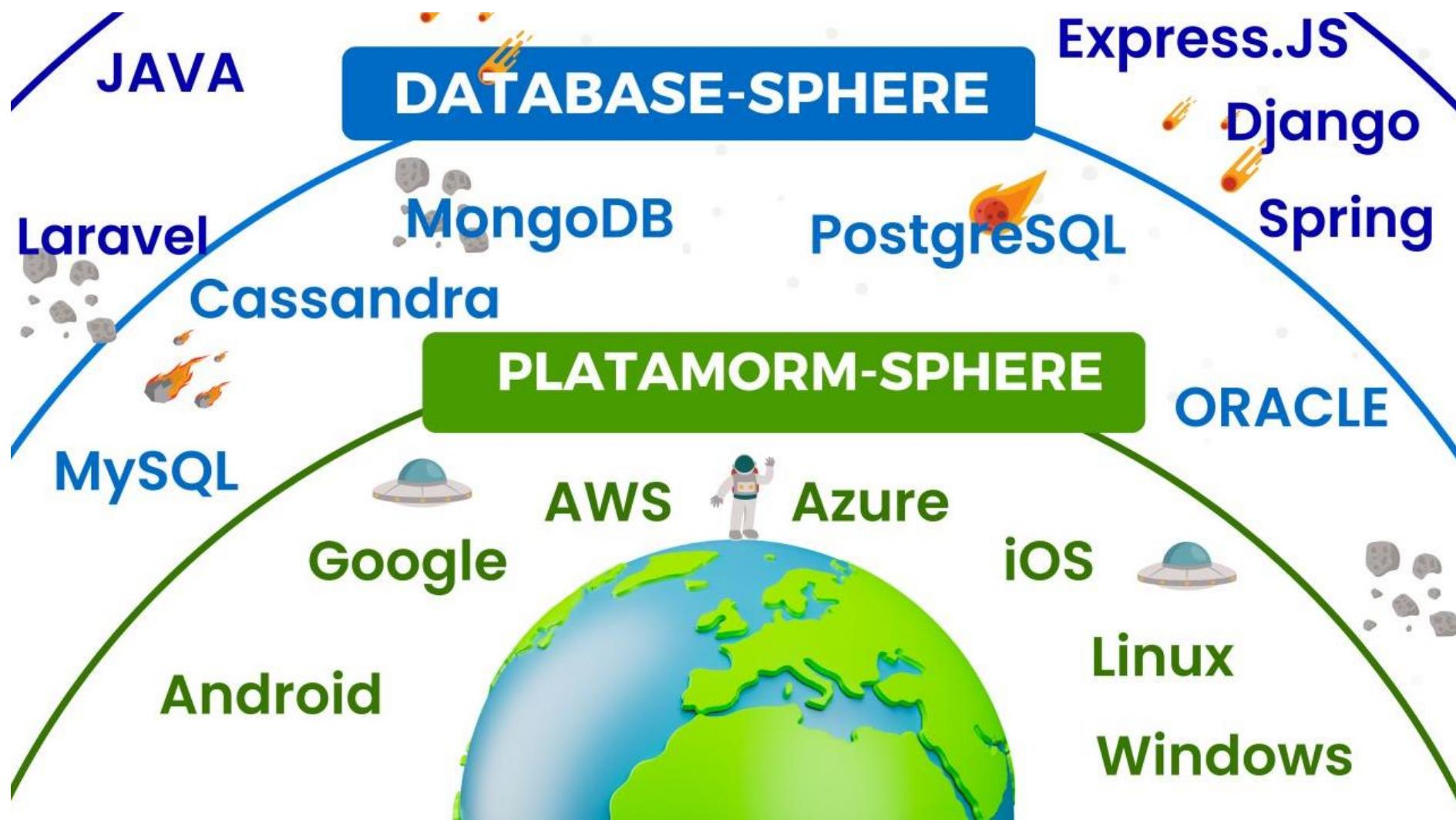
[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- NoSQL Introduction
- Classification
- Examples
  - MongoDB
  - Cassandra
  - GraphDBs: Neo4J

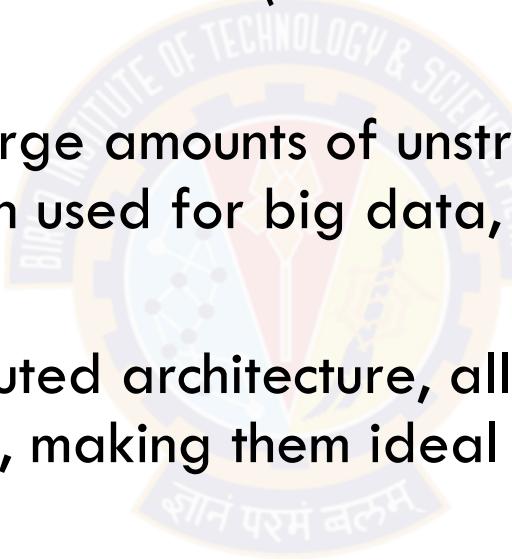


# Database Sphere



# What is NoSQL Database ?

- NoSQL databases, also known as "Not Only SQL" databases, are a type of database that do not use traditional SQL (Structured Query Language) for storing and manipulating data
- They are designed to handle large amounts of unstructured, semi-structured, or polymorphic data and are often used for big data, real-time data processing, and cloud-based applications
- NoSQL databases use a distributed architecture, allowing them to scale horizontally across multiple servers or nodes, making them ideal for handling high levels of concurrency and data volume



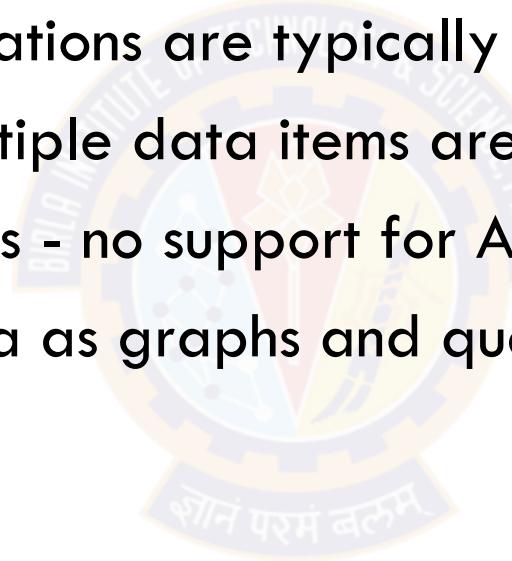
# What is NoSQL ?

- Coined by Carlo Strozzi in 1998
  - ✓ Lightweight, open source database without standard SQL interface
- Reintroduced by Johan Oskarsson in 2009
  - ✓ Non-relational databases
- Characteristics
  - ✓ Not Only SQL
  - ✓ Non-relational
  - ✓ Schema-less
  - ✓ Loosen consistency to address scalability and availability requirements in large scale applications
  - ✓ Open source movement born out of web-scale applications
  - ✓ Distributed for scale
  - ✓ Cluster Friendly



# Data model

- Supports rich variety of data : structured, semi-structured and unstructured
- No fixed schema, i.e. each record could have different attributes
- Non-relational - no join operations are typically supported
- Transaction semantics for multiple data items are typically not supported
- Relaxed consistency semantics - no support for ACID as in RDBMS
- In some cases can model data as graphs and queries as graph traversals

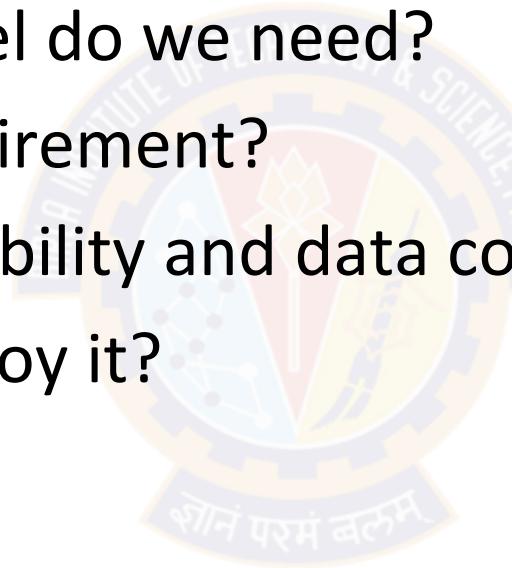


# How to choose the right NoSQL database?

## 5 questions to ask before choosing a NoSQL database

- Is NoSQL the right choice?
- Which NoSQL data model do we need?
- What is the latency requirement?
- How important are scalability and data consistency?
- How do we want to deploy it?

[Link to infoworld article](#)

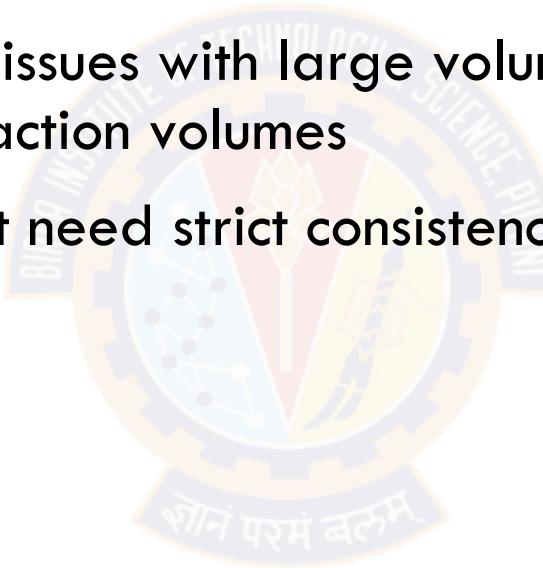


# NoSQL Use Cases

- **Big data:** NoSQL databases are perfect for handling large amounts of data since they can scale horizontally across multiple servers or nodes and handle high levels of concurrency
- **Real-time data processing:** They are often used for real-time data processing since they can handle high levels of concurrency and support low latency
- **Cloud-based applications:** NoSQL databases are perfect for cloud-based applications since they can easily scale and handle large amounts of data in a distributed environment
- **Content management:** NoSQL databases are often used for content management systems since they can handle large amounts of data and support flexible data models
- **Social media:** NoSQL databases are often used for social media applications since they can handle high levels of concurrency and support flexible data models
- **Internet of Things (IoT):** These databases are often used for IoT applications since they can handle large amounts of data from a large number of devices and handle high levels of concurrency
- **E-commerce:** They are often used for e-commerce applications since they can handle high levels of concurrency and support flexible data models

# Why NoSQL (1)

- RDBMS meant for OLTP systems / Systems of Record
  - Strict consistency and durability guarantees (ACID) over multiple data items involved in a transaction
  - But they have scale and cost issues with large volumes of data, distributed geo-scale applications, very high transaction volumes
- Typical web scale systems do not need strict consistency and durability for every use case
  - Social networking
  - Real-time applications
  - Log analysis
  - Browsing retail catalogs
  - Reviews and blogs
  - ...



## Why NoSQL (2)

- RDBMS ensure uniform structure and modelling of relationships between entities
- A class of emerging applications need granular and extreme connectivity information modelled between individual semi-structured data items. This information needs to be also queried at scale without large expensive joins.
  - Connectivity between users in a social media application: How many friends do you have between 2 hops ?
  - Connectivity between companies in terms of domain, technology, people skills, hiring : Useful for skills acquisition, M&A etc.
  - Connectivity between IT network devices: Useful for troubleshooting incidents

# Choice between consistency and availability

- In a distributed database
  - ✓ Scalability and fault tolerance can be improved through additional nodes, although this puts challenges on maintaining consistency (C).
  - ✓ The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
    - May have to update all replicas before sending success to client . so longer takes time and system may not be available during this period to service reads on same data item.
- Large scale distributed systems cannot be 100% partition tolerant (P).
  - ✓ Although communication outages are rare and temporary, partition tolerance (P) must always be supported by distributed database
- In NoSQL, generally a choice between choosing either CP or AP of CAP
- RDBMS systems mainly provide CA for single data items and then on top of that provide ACID for transactions that touch multiple data items.

# CAP Theorem – Implications of CA, CP, AP

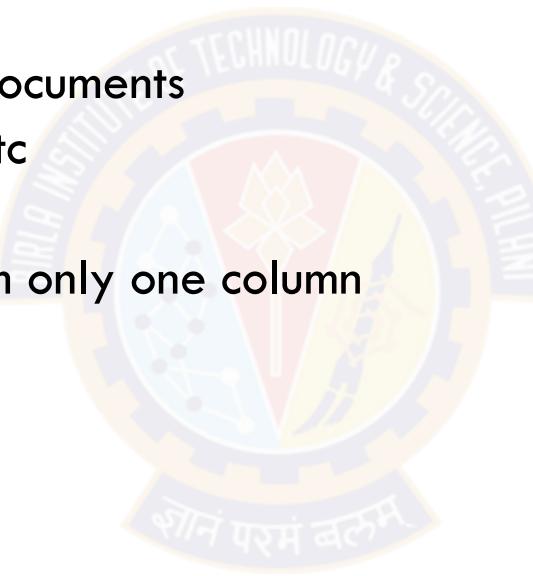
- CA- Implies single site cluster in which all nodes communicate with each other.
- CP – Implies all the available data consistent or accurate, but some data may not be available
- AP – Implies all data available, but some data returned may be inconsistent

[Consistency Models of NoSQL Databases](#)

**When you select a NoSQL database in the above categories, check whether it will match with the requirements of the application.**

# Classification of NoSQL DBs

- Key – value
  - ✓ Maintains a big hash table of keys and values
  - ✓ Example : DynamoDB, Redis, Riak etc
- Document
  - ✓ Maintains data in collections of documents
  - ✓ Example : MongoDB, CouchDB etc
- Column
  - ✓ Each storage block has data from only one column
  - ✓ Example : Cassandra, HBase
- Graph
  - ✓ Network databases
  - ✓ Graph stores data in nodes
  - ✓ Example : Neo4j, HyperGraphDB, Apache Tinkerpop



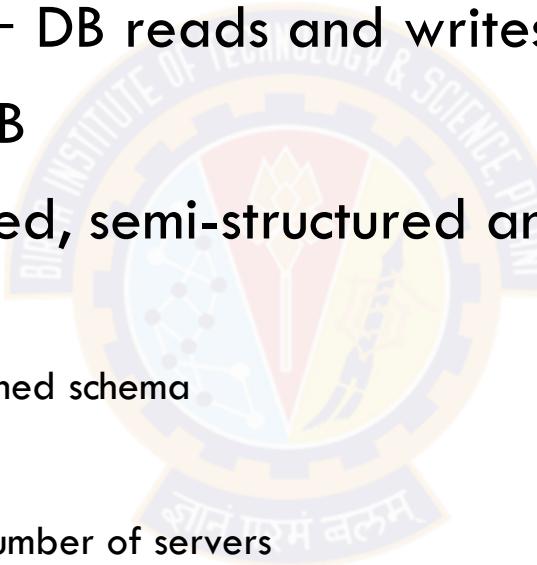
NoSQL Databases - <http://nosql-database.org/>

# NoSQL Offerings in Cloud

NOSQL TYPE	AWS	MICROSOFT AZURE	GOOGLE CLOUD	THIRD-PARTY OPTIONS*
Key-value store	■ Amazon DynamoDB	■ Azure Table Storage ■ Azure Cosmos DB Table API and SQL API	■ Google Cloud Bigtable	■ Redis ■ Riak
Document database	■ Amazon DocumentDB	■ Azure Cosmos DB SQL API	■ Firestore ■ Firebase Realtime Database	■ MongoDB ■ Couchbase Server
Graph database	■ Amazon Neptune	■ Azure Cosmos DB Gremlin API	■ Integrated third-party databases offered by other vendors	■ Neo4j ■ JanusGraph
Wide-column store	■ Amazon Keyspaces	■ Azure Cosmos DB Cassandra API	■ Google Cloud Bigtable	■ HBase ■ Cassandra
In-memory cache	■ Amazon ElastiCache ■ Amazon MemoryDB for Redis	■ Azure Cache for Redis	■ Memorystore	■ Memcached ■ Redis
Search database	■ Amazon OpenSearch Service	■ Azure Cognitive Search	■ Google Cloud Search	■ Elasticsearch ■ Splunk
Time series database	■ Amazon Timestream	■ Azure Data Explorer	■ Google Cloud Bigtable	■ InfluxDB ■ kdb+

# Characteristics

- Scale out architecture instead of monolithic architecture of relational databases
  - Cluster scale - distribution across 100+ nodes across DCs
  - Performance scale - 100K+ DB reads and writes per sec
  - Data scale - 1B+ docs in DB
- House large amount of structured, semi-structured and unstructured data
- Dynamic schemas
  - ✓ allows insertion of data without pre-defined schema
- Auto sharding
  - ✓ automatically spreads data across the number of servers
  - ✓ applications are not aware about it
  - ✓ helps in data balancing and failure from recovery
- Replication
  - ✓ Good support for replication of data which offers high availability, fault tolerance



# Pros and Cons

## Pros

- Cost effective for large data sets
- Easy to implement
- Easy to distribute esp across DCs
- Easier to scale up/down
- Relaxes data consistency when required
- No pre-defined schema
- Easier to model semi-structured data or connectivity data
- Easy to support data replication

## Cons

- Joins between data sets / tables
- Group by operations
- ACID properties for transactions
- SQL interface
- Lack of standardisation in this space
  - Makes it difficult to port from SQL and across NoSQL stores
- Less skills compared to SQL
- Lesser BI tools compared to mature SQL BI space



# SQL vs NoSQL

SQL	NoSQL
Relational database	Non relational, distributed databases
Pre-defined schema	Schema less
Table based databases	Multiple options: Key-Value, Document, Column, Graph
Vertically scalable	Horizontally scalable
Supports ACID properties	Supports CAP theorem
Supports complex querying	Relatively simpler querying
Excellent support from vendors	Relies heavily on community support

# Vendors

- Amazon
- Facebook
- Google
- Oracle



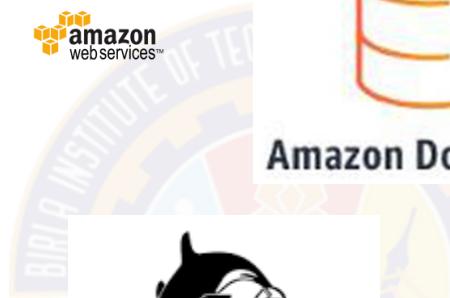
DynamoDB



cassandra



redis



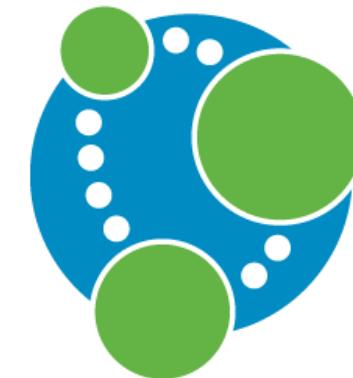
Amazon DocumentDB



**ORACLE®**  
NOSQL DATABASE



Amazon Neptune



# Topics for today

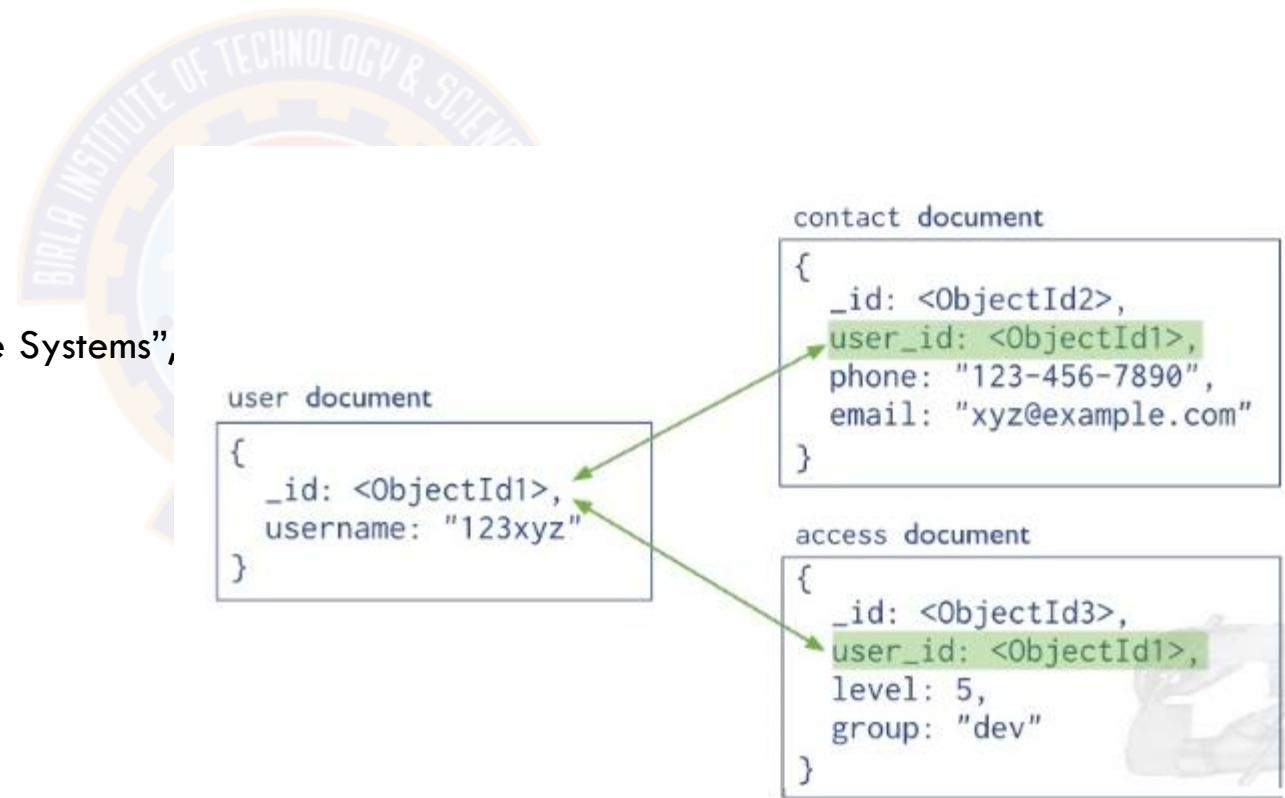
- NoSQL Introduction
- **Classification**
- Examples
  - Cassandra
  - Mongo
  - GraphDBs: Neo4J



# Classification: Document-based

- Store data in form of documents using well known formats like JSON
- Documents accessible via their id, but can be accessed through other index as well
- Maintains data in collections of documents
- Example,
  - MongoDB, CouchDB, CouchBase
- Book document :

```
{  
    "Book Title" : "Fundamentals of Database Systems",  
    "Publisher" : "Addison-Wesley",  
    "Authors" : "Elmasri & Navathe"  
    "Year of Publication" : "2011"  
}
```

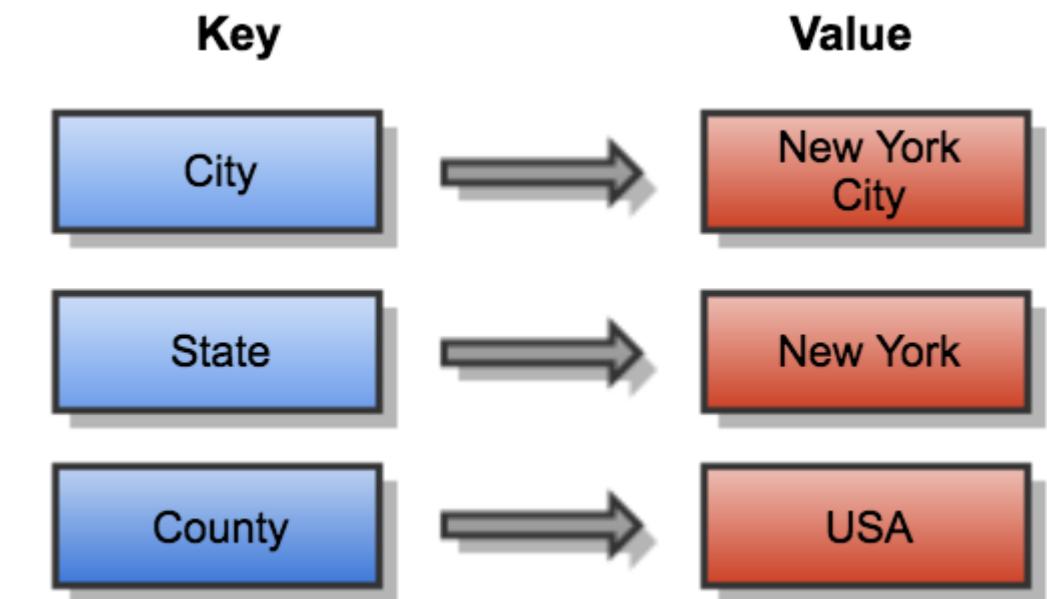
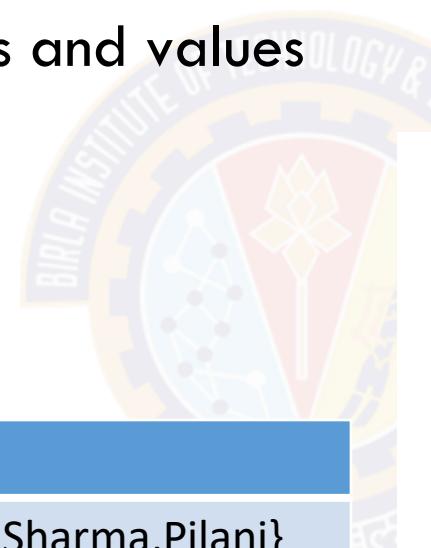


# Classification: Key-Value store

- Simple data model based on fast access by the key to the value associated with the key
- Value can be a record or object or document or even complex data structure
- Maintains a big hash table of keys and values
- For example,

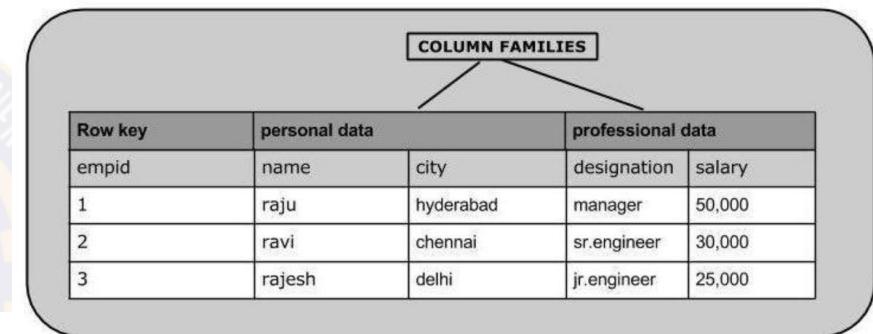
✓ DynamoDB, Redis, Riak

Key	Value
2014HW112220	{ Santosh,Sharma,Pilani}
2018HW123123	{Eshwar,Pillai,Hyd}

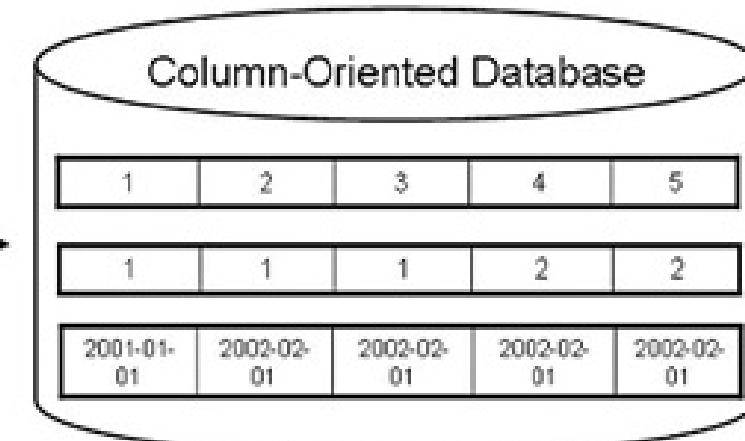


# Classification: Column-based

- Partition a table by column into column families
- A part of vertical partitioning where each column family is stored in its own files
- Allows versioning of data values
- Each storage block has data from only one column
- Example,
  - ✓ Cassandra, Hbase

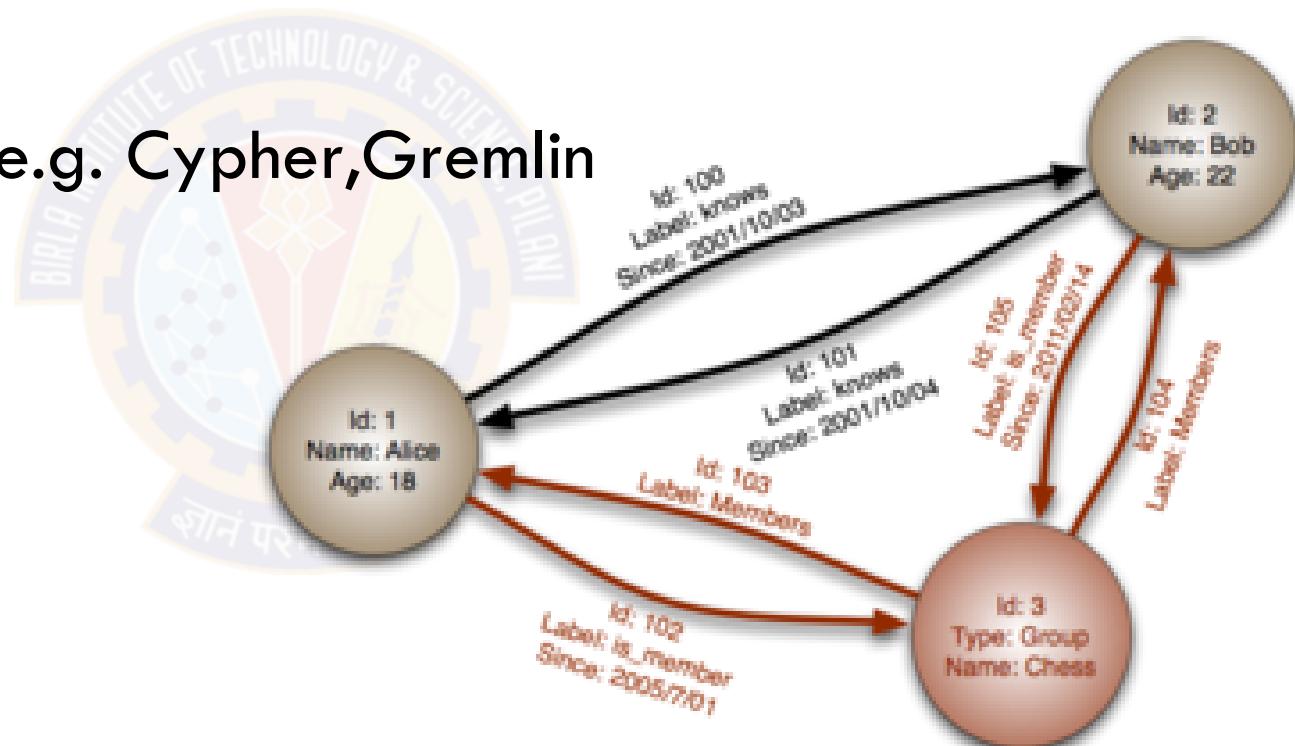


Emp_no	Dept_Id	Hire_date	Emp_In	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura



# Classification: Graph based

- Data is represented as graphs and related nodes can be found by traversing the edges using the path expression
- aka network database
- Graph query languages, e.g. Cypher, Gremlin
- Example
  - ✓ Neo4J
  - ✓ HyperGraphDB
  - ✓ GraphX
  - ✓ Apache TinkerPop



# Topics for today

- NoSQL Introduction
- Classification
- Examples
  - ✓ MongoDB
  - ✓ Cassandra
  - ✓ GraphDBs: Neo4J



# MongoDB

- Database is a set of collections
- A collection is like a table in RDBMS
- A collection stores documents
  - ✓ BSON or Binary JSON with hierarchical key-value pairs
  - ✓ Similar to rows in a table
  - ✓ Max 16MB documents stored in **WiredTiger** storage engine
- For larger than 16MB documents uses GridFS
  - ✓ Support for binary data
  - ✓ Large objects can be stored in ‘chunks’ of 255KB
  - ✓ Stores Meta-data in a separate collection
  - ✓ Does not support multi-document transactions
  - ✓ WiredTiger storage engine\*

# MongoDB

- Data is partitioned in shards
  - ✓ For horizontal scaling
  - ✓ Reduces amount of data each shard handles as the cluster grows
  - ✓ Reduces number of operations on each shard
- Data is replicated
  - ✓ Writes to primary in oplog. “write-concern” setting used to tweak write consistency.
  - ✓ Secondaries use oplog to get local copies updated
  - ✓ Clients usually read from primary but “read-preference” setting can tweak read consistency
- Data updates happen in place and not versioned / timestamped

# MongoDB Data Example

Collection inventory

```
{  
  item: "ABC2",  
  details: { model: "14Q3", manufacturer: "M1 Corporation" },  
  stock: [ { size: "M", qty: 50 } ],  
  category: "clothing"  
}  
  
{  
  item: "MNO2",  
  details: { model: "14Q3", manufacturer: "ABC Company" },  
  stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L", qty: 5 } ],  
  category: "clothing"  
}
```

Document insertion

```
db.inventory.insert(  
  {  
    item: "ABC1",  
    details: {model: "14Q3", manufacturer: "XYZ Company"},  
    stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],  
    category: "clothing"  
  }  
)
```

# Example of Simple Query

Collection orders

```
{  
  _id: "a",  
  cust_id: "abc123",  
  status: "A",  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 3 },  
           { sku: "nnn", qty: 5, price: 2 } ]  
}  
  
{  
  _id: "b",  
  cust_id: "abc124",  
  status: "B",  
  price: 12,  
  items: [ { sku: "nnn", qty: 2, price: 2 },  
           { sku: "ppp", qty: 2, price: 4 } ]  
}
```

db.users.find(

  { status: "A" },

  { cust\_id: 1, price: 1, \_id: 0 }

)

*selection*

*projection*

In SQL it would look like this:

```
SELECT cust_id, price  
FROM orders  
WHERE status="A"
```

Results

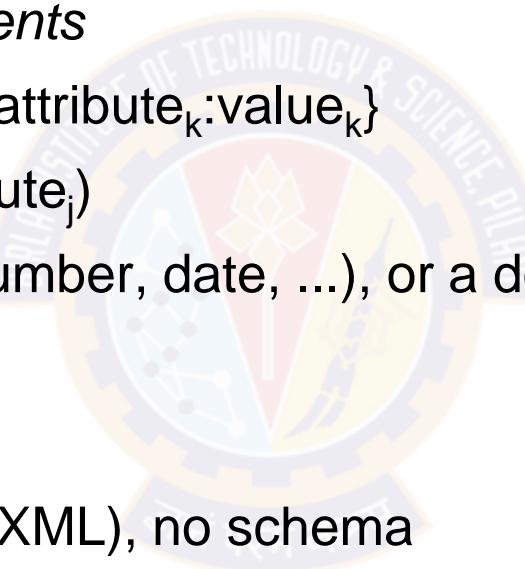
{

  cust\_id: "abc123",  
  price: 25

}

# MongoDB Data Model

- JavaScript Object Notation (JSON) model
- *Database* = set of named *collections*
- *Collection* = sequence of *documents*
- *Document* = {attribute<sub>1</sub>:value<sub>1</sub>,...,attribute<sub>k</sub>:value<sub>k</sub>}
- *Attribute* = string (attribute<sub>i</sub>≠attribute<sub>j</sub>)
- *Value* = primitive value (string, number, date, ...), or a document, or an *array*
- *Array* = [value<sub>1</sub>,...,value<sub>n</sub>]
  
- Key properties: hierarchical (like XML), no schema
  - ✓ Collection docs may have different attributes



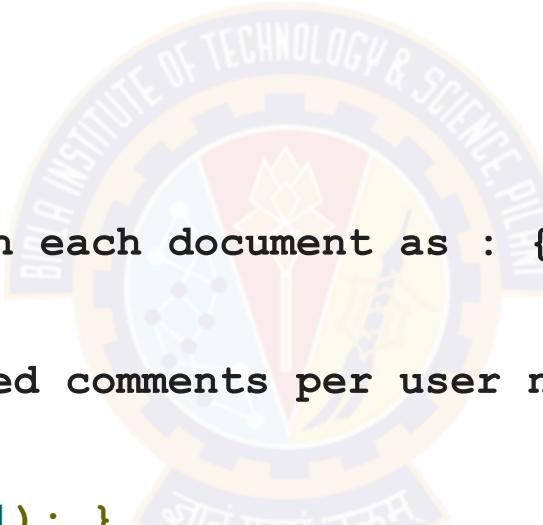
# MongoDB: MapReduce

```
> db.collection.mapReduce(  
  function() {emit(key,value);}, //map function  
  function(key,values) {return reduceFunction}, { //reduce function  
    out: collection,  
    query: document,  
    sort: document,  
    limit: number  
}
```

create a collection reviews with each document as : { "name": "abc", "review": "...", "publish": "true" }

now count the number of published comments per user name

```
>db.posts.mapReduce(  
  function() { emit(this.name,1); },  
  function(key, values) {return Array.sum(values)}, {  
    query:{publish:"true"},  
    out:"total_reviews"  
}  
) .find()
```



# MongoDB: Indexing

- Can create index on any field of a collection or a sub-document fields
- e.g. document in a collection

```
{  
    "address": {  
        "city": "New Delhi",  
        "state": "Delhi",  
        "pincode": "110001"  
    },  
    "tags": [  
        "football",  
        "cricket",  
        "badminton"  
    ],  
    "name": "Ravi"  
}
```



- indexing a field in ascending order and find

```
> db.users.createIndex({tags:1})  
> db.users.find({tags:"cricket"}).pretty()
```

- indexing a sub-document field in ascending order and find

```
> db.users.createIndex({"address.city":1,"address.state":1,"address.pincode":1})  
> db.users.find({"address.city":"New Delhi"}).pretty()
```

# MongoDB: Joins

- Mongo 3.2+ it is possible to join data from 2 collections using aggregate
- Collection books (isbn, title, author) and books\_selling\_data(isbn, copies\_sold)

```
db.books.aggregate([{ $lookup: {  
    from: "books_selling_data",  
    localField: "isbn",  
    foreignField: "isbn",  
    as: "copies_sold"  
}}])
```

- Sample joined document:

```
{  
  "isbn": "978-3-16-148410-0",  
  "title": "Some cool book",  
  "author": "John Doe",  
  "copies_sold": [  
    {  
      "isbn": "978-3-16-148410-0",  
      "copies_sold": 12500  
    }  
  ]  
}
```

If you have two collections (users , comments) and want to pull all the comments with pid=444 along with the user info for each comments

```
{ uid:12345, pid:444, comment:"blah" }  
{ uid:12345, pid:888, comment:"asdf" }  
{ uid:99999, pid:444, comment:"qwer" }
```

users

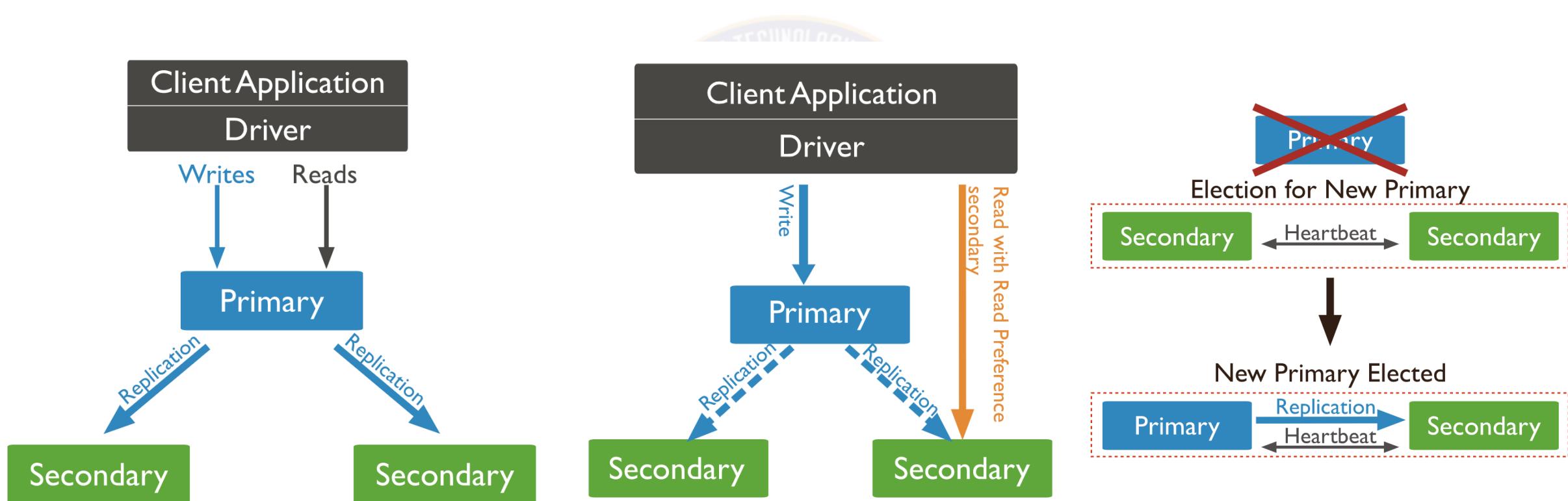
```
{ uid:12345, name:"john" }  
{ uid:99999, name:"mia" }
```

Join command - Join using \$lookup

```
db.users.aggregate({  
  $lookup:{  
    from:"comments",  
    localField:"uid",  
    foreignField:"pid",  
    as:"users_comments"  
  }  
})
```

# MongoDB – Writes and Reads

- Document oriented DB
- Various read and write choices for flexible consistency tradeoff with scale / performance and durability
- Automatic primary re-election on primary failure and/or network partition



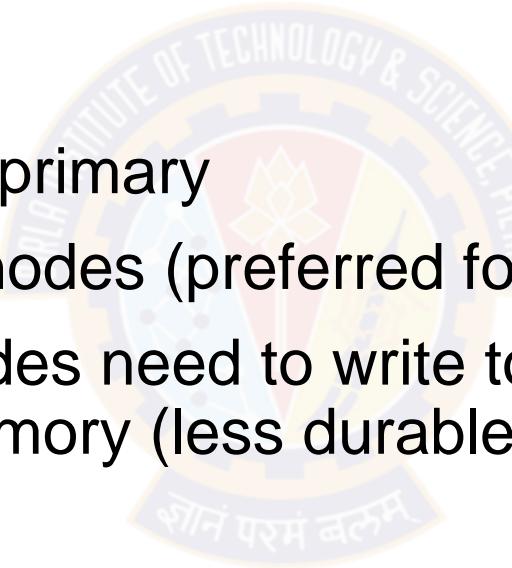
# MongoDB “read concerns”

- local :
  - ✓ Client reads primary replica
  - ✓ Client reads from secondary in causally consistent sessions
- available:
  - ✓ Read on secondary but causal consistency not required
- majority :
  - ✓ If client wants to read what majority of nodes have. Best option for fault tolerance and durability.
- linearizable :
  - ✓ If client wants to read what has been written to majority of nodes before the read started.
  - ✓ Has to be read on primary
  - ✓ Only single document can be read

<https://docs.mongodb.com/v3.4/core/read-preference-mechanics/>

# MongoDB “write concerns”

- how many replicas should ack
  - ✓ 1 - primary only
  - ✓ 0 - none
  - ✓ n - how many including primary
  - ✓ majority - a majority of nodes (preferred for durability)
- journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)
- timeout for write operation

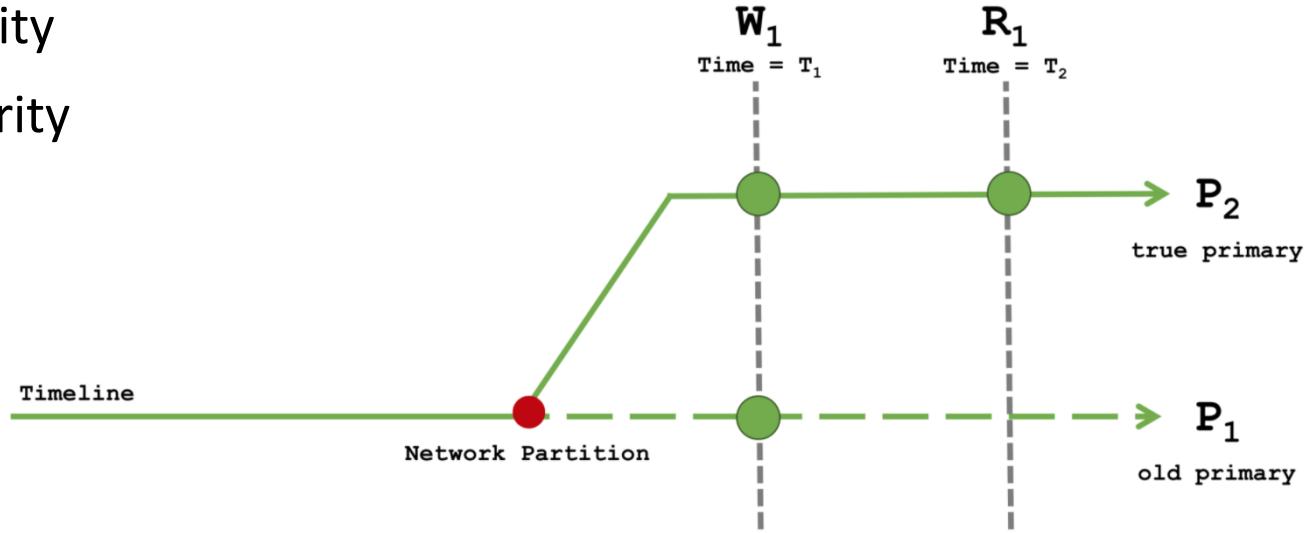


<https://docs.mongodb.com/manual/reference/write-concern/>

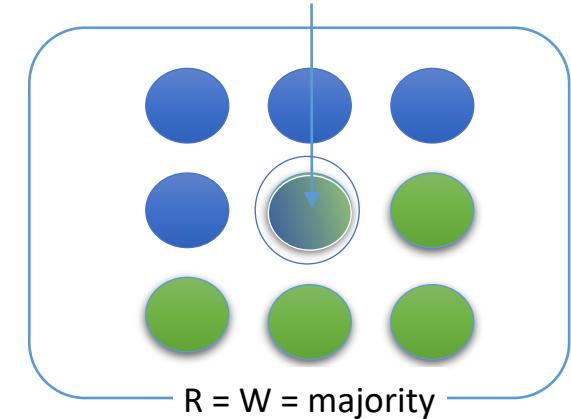
# Consistency scenarios - causally consistent and durable

read=majority

write=majority



Read latest written value  
from common node



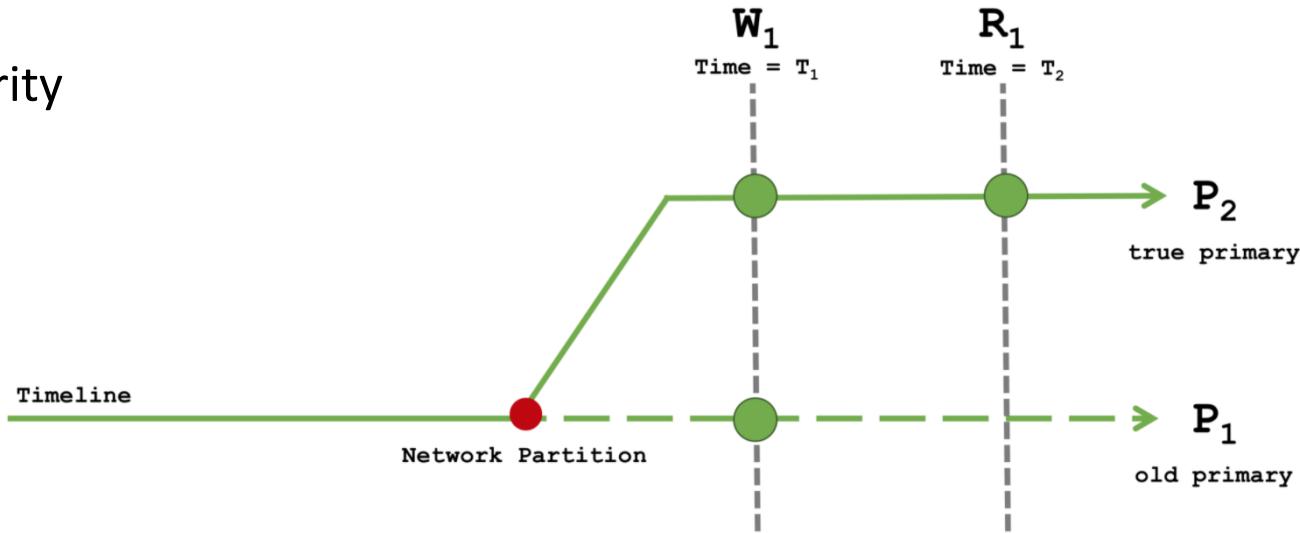
- $W_1$  and  $R_1$  for  $P_1$  will fail and will succeed in  $P_2$
- So causally consistent, durable even with network partition sacrificing performance
- *Example:* Used in critical transaction oriented applications, e.g. stock trading

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

# Consistency scenarios - causally consistent but not durable

read=majority

write=1



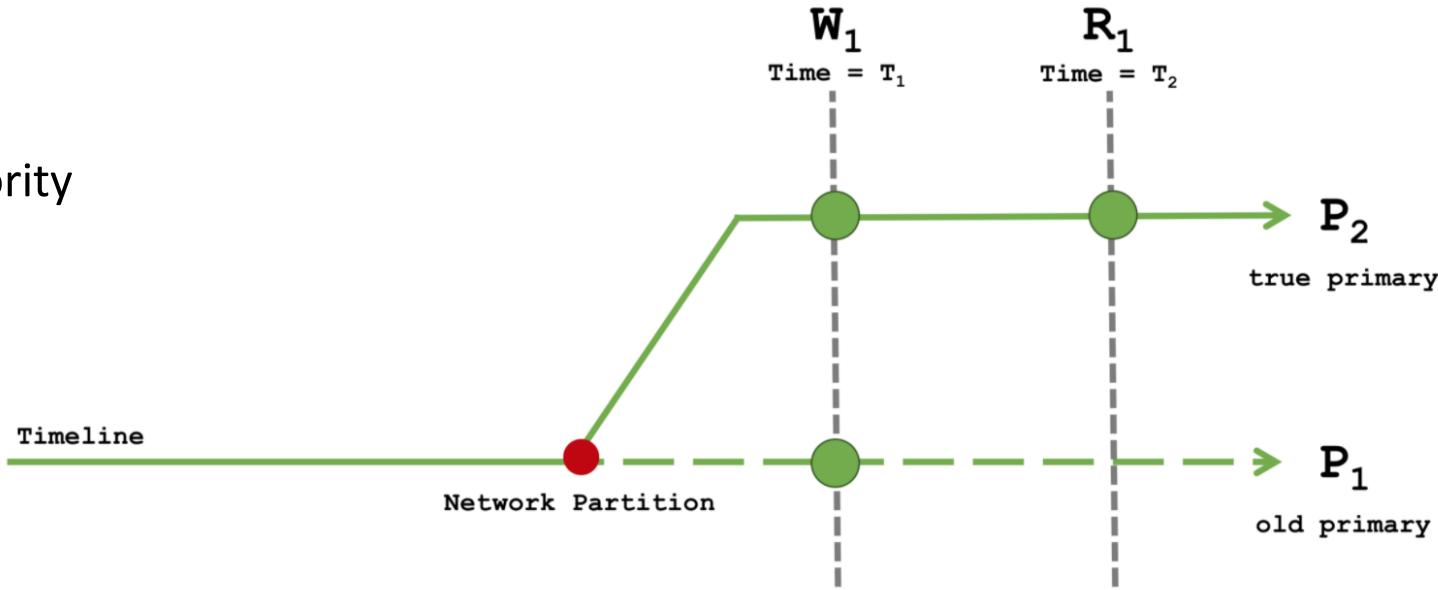
- $W_1$  may succeed on  $P_1$  and  $P_2$ .  $R_1$  will succeed only on  $P_2$ .  $W_1$  on  $P_1$  may roll back.
- So causally consistent but not durable with network partition. Fast writes, slower reads.
- *Example:* Twitter - a post may disappear but if on refresh you see it then it should be durable, else repost.

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

# Consistency scenarios - eventual consistency with durable writes

read=local

write=majority



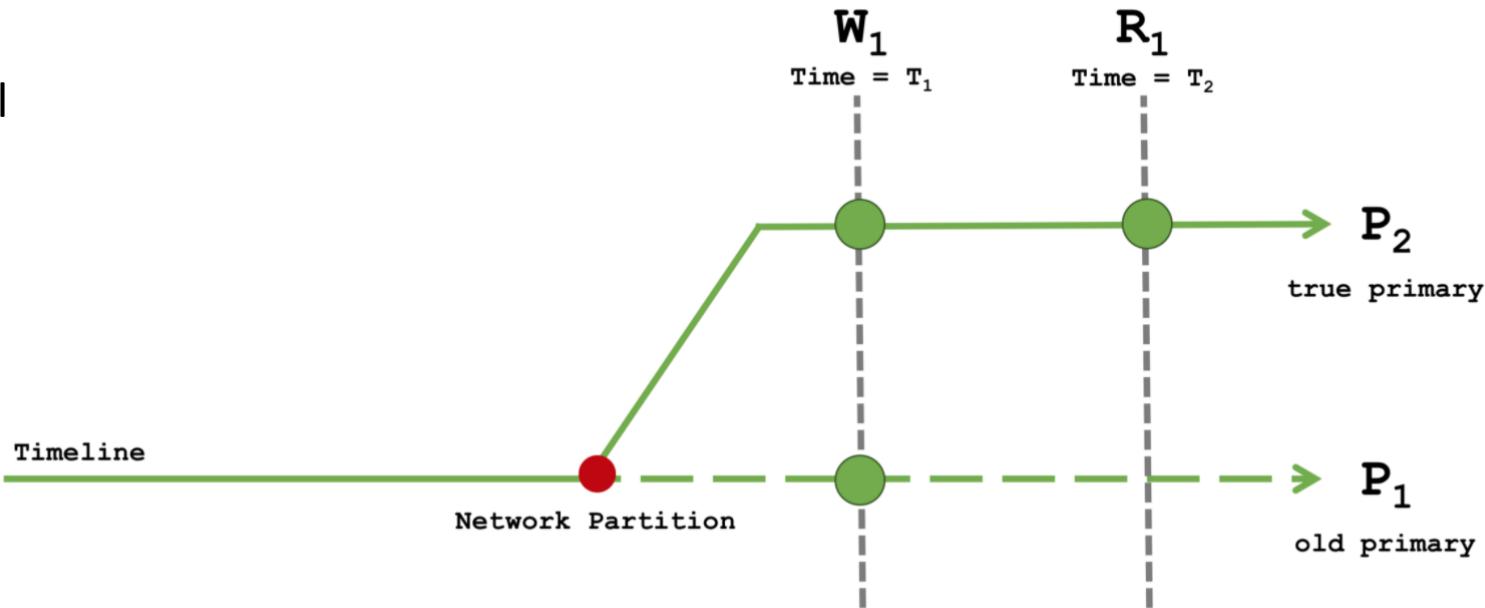
- $W_1$  will succeed only for  $P_2$  and will not be accepted on  $P_1$  after failure. Reads may not succeed to see the last write on  $P_1$ . Slow durable writes and fast non-causal reads.
- *Example:* Review site where write should be durable if committed but reads don't need causal guarantee as long as it appears some time (eventual consistency).

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

# Consistency scenarios - eventual consistency but no durability

read=local

write=1

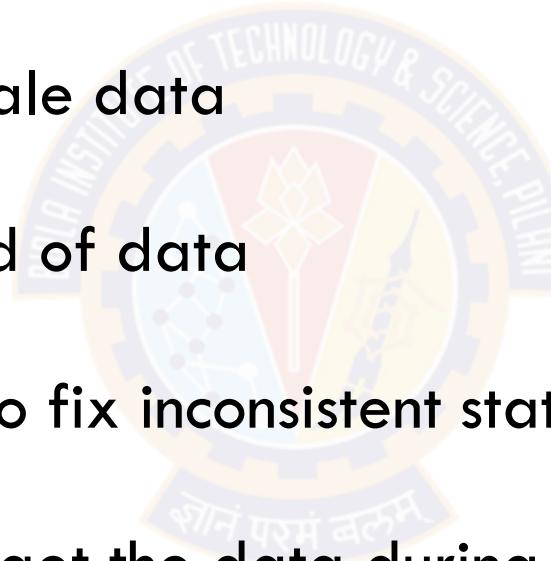


- Same as previous scenario and not writes are also not durable and may be rolled back.
- *Example:* Real-time sensor data feed that needs fast writes to keep up with the rate and reads should get as much recent real-time data as possible. Data may be dropped on failures.

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

# How applications deal with eventual consistency of AP

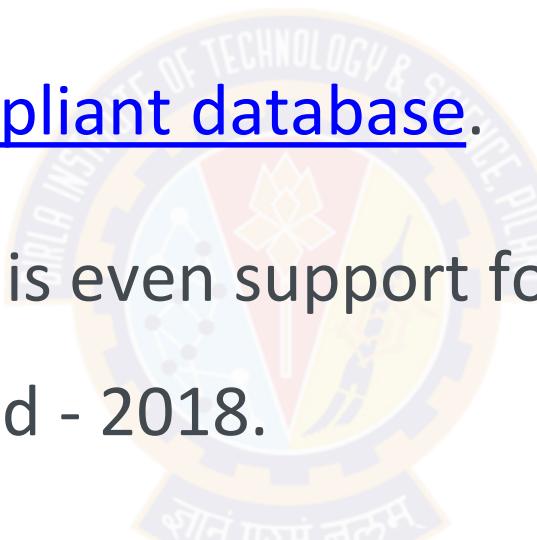
- Application must be ready to deal with multiple versions of data
- Application must handle stale data
- Expect NULL values instead of data
- Application must be able to fix inconsistent state
- Read again, if you do not get the data during the first read



# MongoDB – ACID Transactions

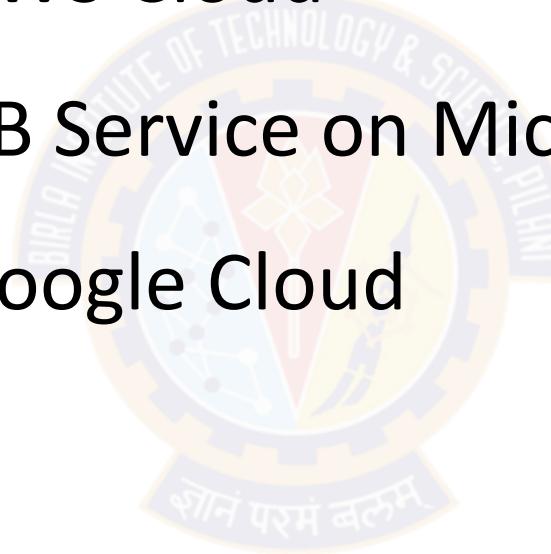
## Can NoSQL databases be ACID-compliant?

- MongoDB is an ACID-compliant database.
- As of MongoDB 4.0, there is even support for multi-document ACID transactions when required - 2018.
- Version 4.2 even brought distributed multi-document ACID transactions for even more flexibility - 2019.



# MongoDB on Cloud

- MongoDB Atlas on AWS Cloud
- Automated MongoDB Service on Microsoft Azure
- MongoDB Atlas on Google Cloud



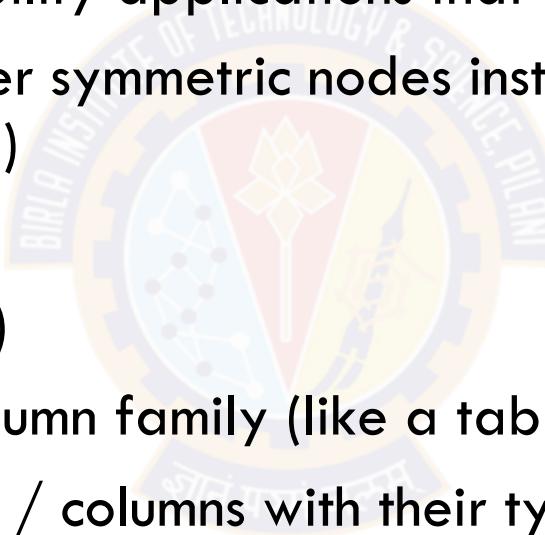
# Topics for today

- NoSQL Introduction
- Classification
- Examples
  - MongoDB
  - Cassandra
  - GraphDBs: Neo4J and Tinkerpop

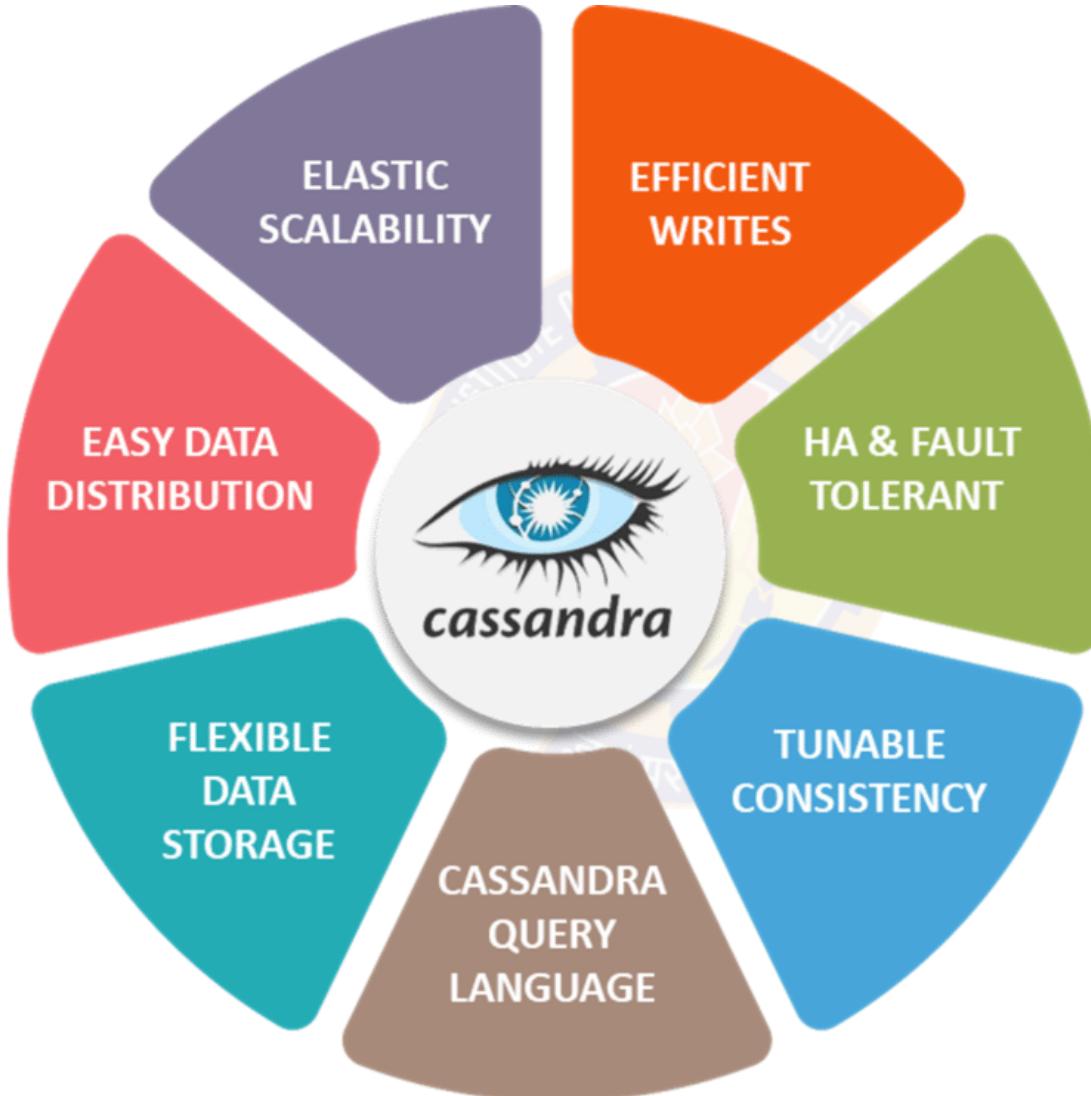


# Cassandra

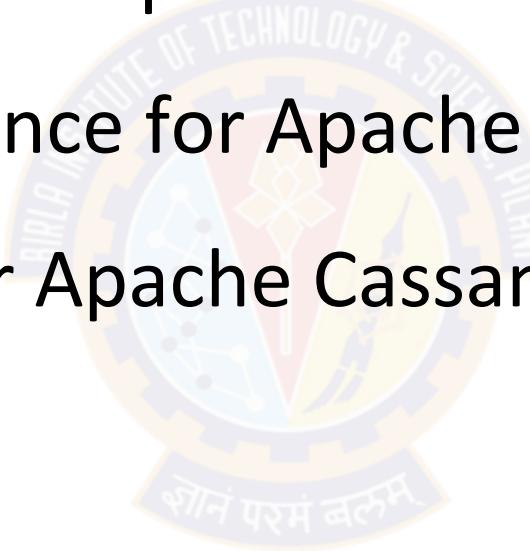
- Born in Facebook and built on Amazon Dynamo and Google Big Table concepts
- AP design in CAP context
- High performance, high availability applications that can sacrifice consistency
  - ✓ Hence built for peer-to-peer symmetric nodes instead of primary-secondary architecture (as in MongoDB)
- Column oriented DB
  - ✓ Create keyspace (like a DB)
  - ✓ Within keyspace create column family (like a table)
  - ✓ Within CF create attributes / columns with their types



# Cassandra features

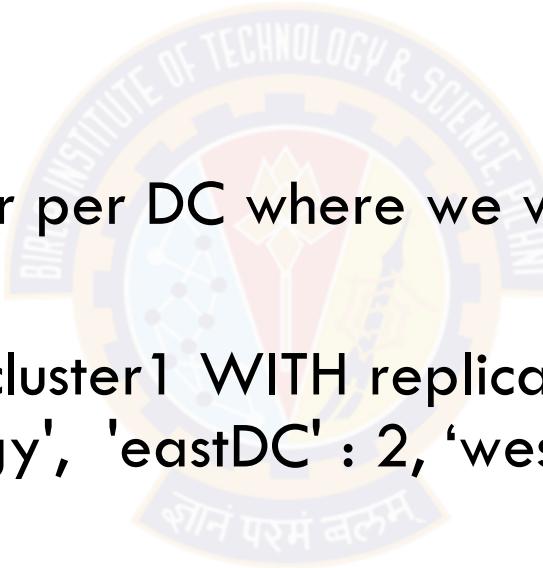


- Amazon Keyspaces (for Apache Cassandra)
- Azure Managed Instance for Apache Cassandra
- DataStax Astra DB for Apache Cassandra (Google Cloud)



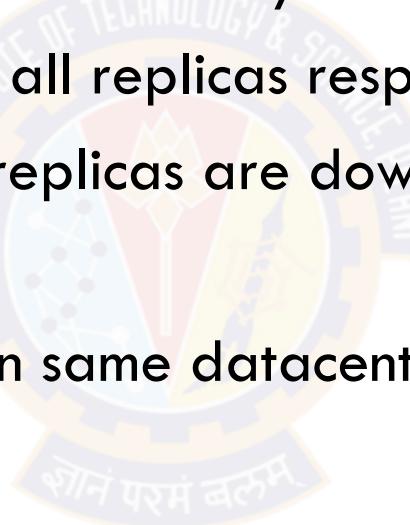
# Replication strategy for user data

- Simple
  - ✓ Specify replication factor = N and data is stored in N nodes of cluster
- NetworkTopology
  - ✓ Specify replication factor per DC where we want reliability from DC failures
  - ✓ e.g. CREATE KEYSPACE cluster1 WITH replication = {'class': 'NetworkTopologyStrategy', 'eastDC' : 2, 'westDC' : 3};



# Consistency semantics (1)

- No primary replica - high partition tolerance and availability and levels of consistency
- Support for light transactions with “linearizable consistency”
- A Read or Write operation can pick a consistency level
  - ONE, TWO, THREE, ALL - 1,2,3 or all replicas respectively have to ack
  - ANY - Write to any node even if replicas are down (ref Hinted Handoff)
  - QUORUM - majority have to ack
  - LOCAL\_QUORUM - majority within same datacenter have to ack
  - ...



<https://cassandra.apache.org/doc/latest/architecture/dynamo.html>

<https://cassandra.apache.org/doc/latest/architecture/guarantees.html#>

# Tunable Consistency

Cassandra guarantees strong consistency if

$(\text{nodes\_Written} + \text{nodes\_Read}) > \text{replication\_factor } N$

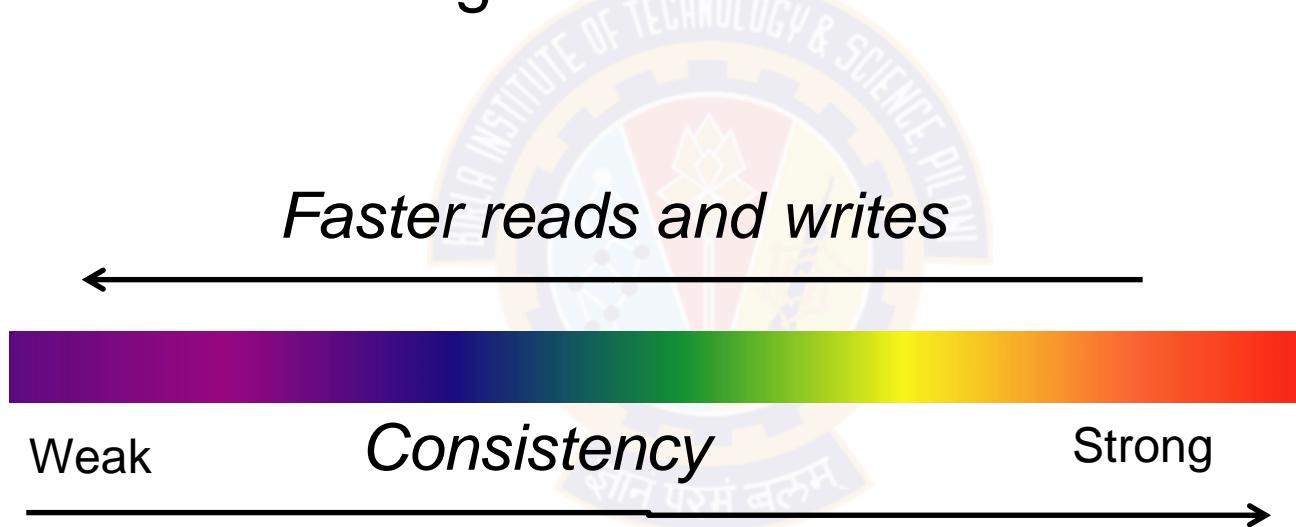
$$R + W > N$$

Tuning done by controlling the number of nodes (replicas)

- Selected for Write
- Selected for reads

# Cassandra Consistency Spectrum

- Cassandra has C A P.
- But Consistency is tunable
- Give up a little A and P to get more C



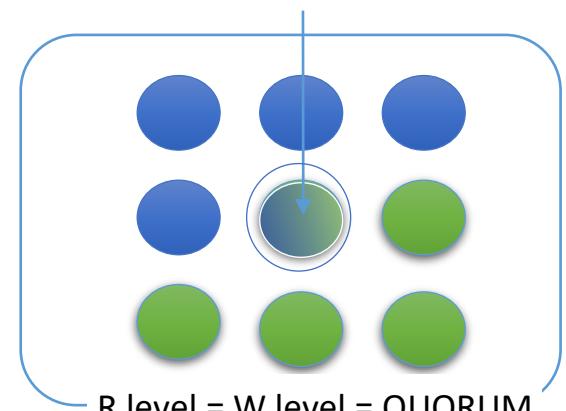
The higher the consistency, the less chance you get stale data during read

- Pay for this with latency
- Depends on your situational needs

## Consistency semantics (2)

- For “causal consistency” pick Read consistency level = Write consistency level = QUORUM
- Why ? At least one node will be common between write and read set so a read will get the last write of a data item
- What happens if read and write use LOCAL\_QUORUM ?
- If no overlap read and write sets then “Eventual consistency”

Read latest written value from common node

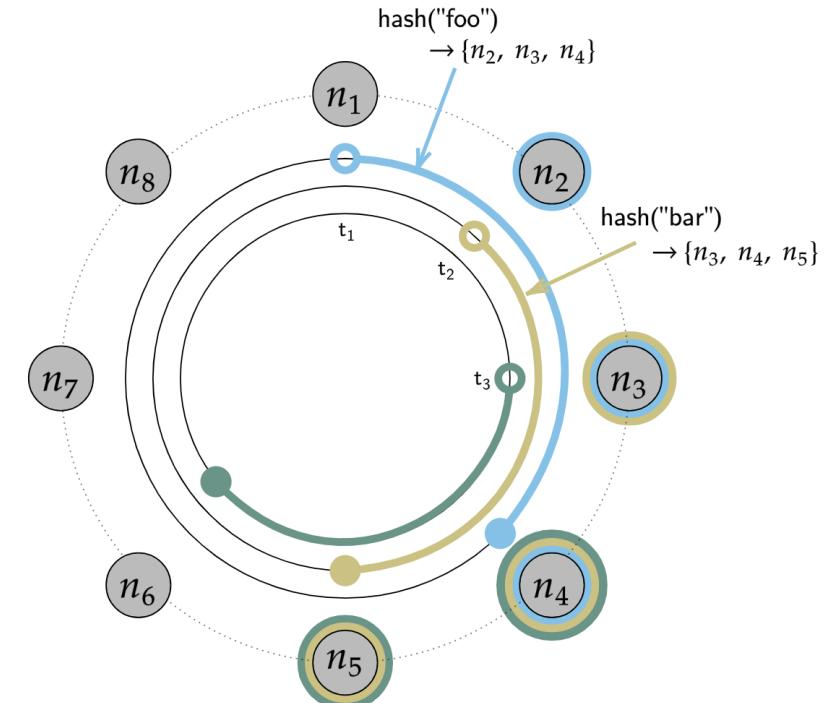
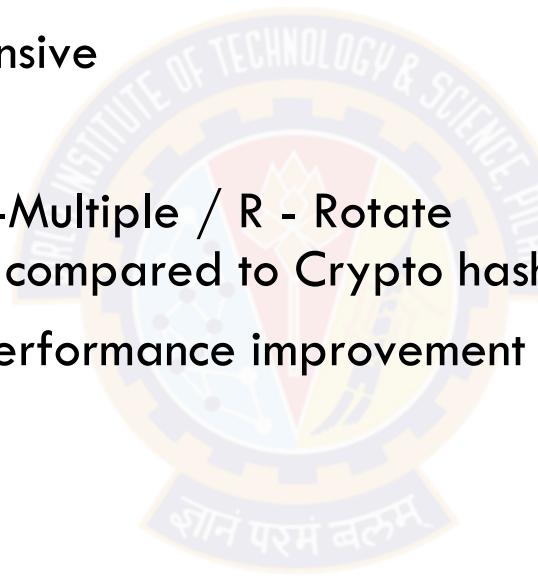


<https://cassandra.apache.org/doc/latest/architecture/dynamo.html>

<https://cassandra.apache.org/doc/latest/architecture/guarantees.html#>

# Partitioners

- Partitions data based on hashing to distribute data blocks from a column among nodes\*
- Random
  - ✓ Crypto hash (MD5)- more expensive
- Murmur
  - ✓ Non-crypto consistent hash (MU-Multiple / R - Rotate operations but easier to reverse compared to Crypto hash)
  - ✓ 3-5x faster and overall 10% performance improvement
- Byteorder
  - ✓ Lexical order

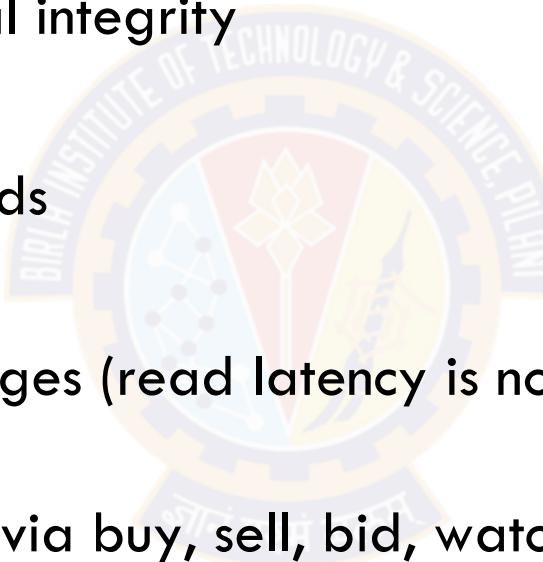


# Sample queries

```
> create keyspace demo with replication={'class':'SimpleStrategy',  
'replication_factor':1};  
> describe keyspaces;  
> use demo; or columnfamily  
> create table student_info (rollno int primary key, name text, DOJ  
timestamp, lastexampercent double);  
> describe table student_info ;  
> consistency quorum  
> insert into student_info (rollno, name, DOJ, lastexampercent) values  
(4, 'Roxanne', dateof(now()), 90) using ttl 30;  
> select rollno from student_info where name='Roxanne' ALLOW  
FILTERING;  
> update student_info set lastexampercent=98 where rollno=2 IF  
name='Sam';
```

# Case study - eBay

- Marketplace has 100 million active buyers with 200+ million items
- 2B page views, 80B DB calls, multi-PB storage capacity
- No transactions, joins, referential integrity
- Multi-DC deployment
- 400M+ writes and 200M+ reads
- 3 Use cases
  - ✓ Social signal on product pages (read latency is not important but write performance is key)
  - ✓ Connecting users and items via buy, sell, bid, watch events
  - ✓ Many time series analysis cases, e.g. fraud detection



<https://www.slideshare.net/jaykumarpatel/cassandra-at-ebay-13920376> / eBay Marketplaces 97 million active

# Case study - AdStage (from AWS use cases)

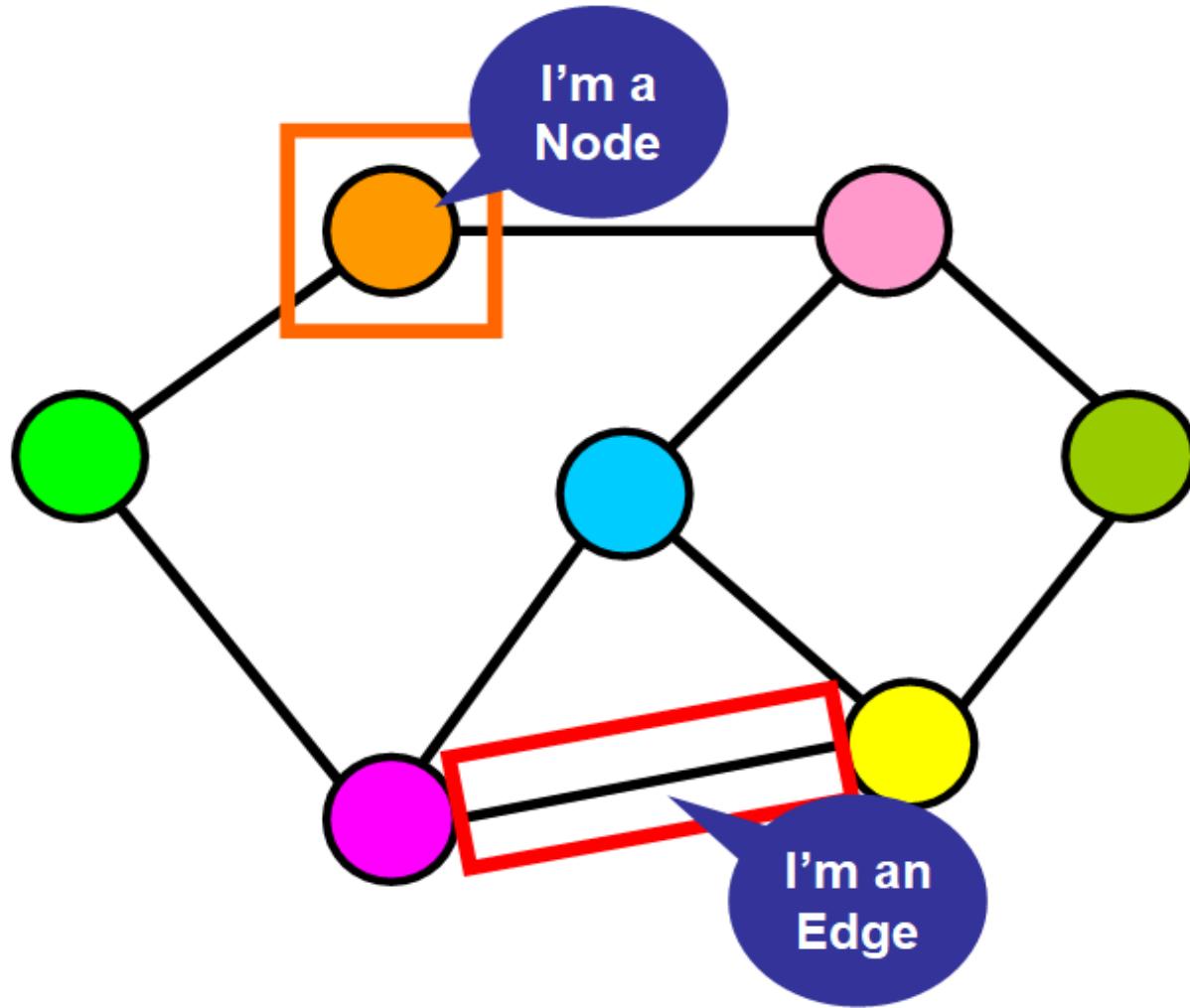
- Sector AdTech
- Online advertising platform to manage multi-channel ad campaigns on Google, FB, Twitter, Bing, LinkedIn
- 3 clusters with 80+ nodes on AWS
- Vast amount of real-time data from 5 channels
- Constantly monitor trends and optimise campaigns for advertisers
- High performance and availability - consistency is not critical as it is read mainly
- Cassandra cluster can scale as more clients are added with no SPOF

# Topics for today

- NoSQL Introduction
- Pros-Cons
- Classification
- Examples
  - MongoDB
  - Cassandra
  - **GraphDBs: Neo4J**

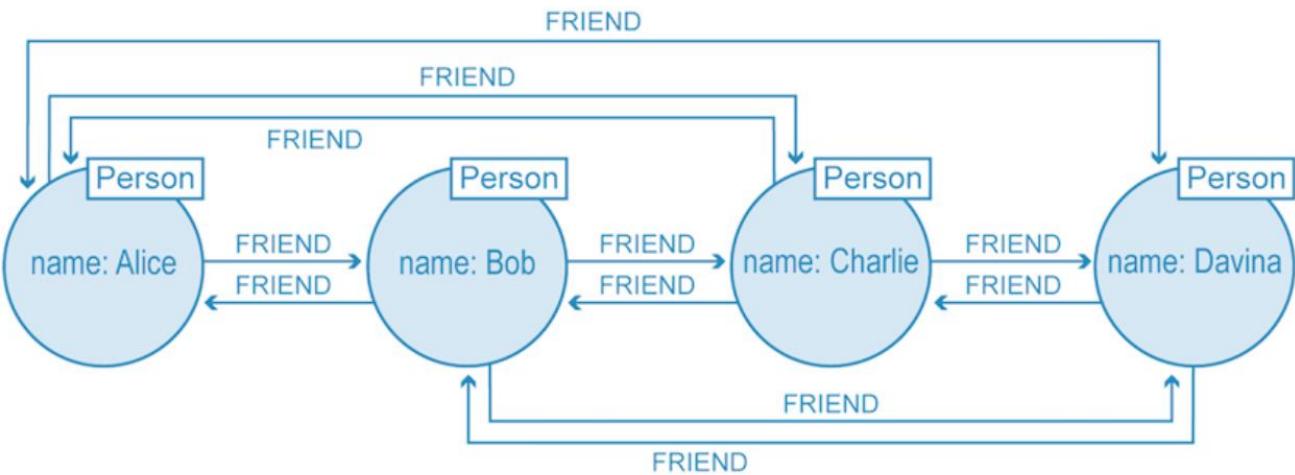
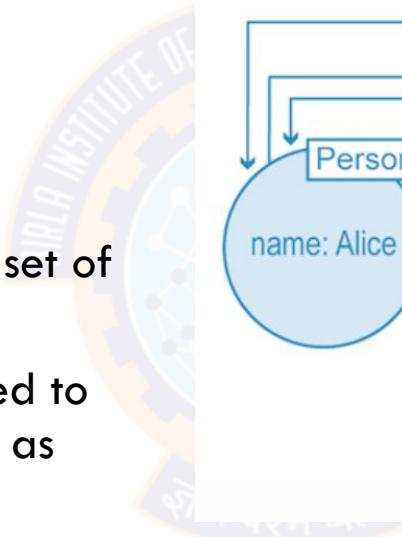


# Graphs



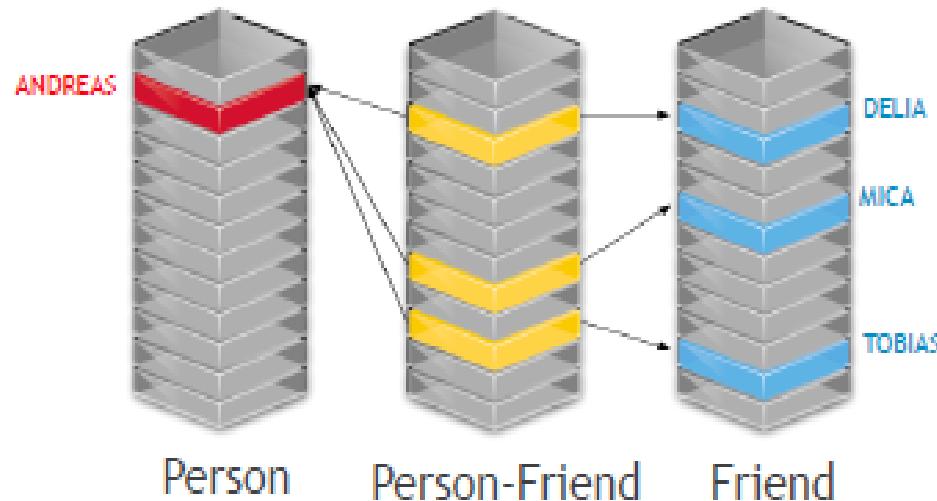
# Graph computing

- Property graphs
  - Data is represented as vertices and edges with properties
  - Properties are key value pairs
  - Edges are relationships between vertices
- When to use a graph DB ?
  - A relationship-heavy data set with large set of data items
  - Queries are like graph traversals but need to keep query performance almost constant as database grows
  - A variety of queries may be asked from the data and static indices on data will not work

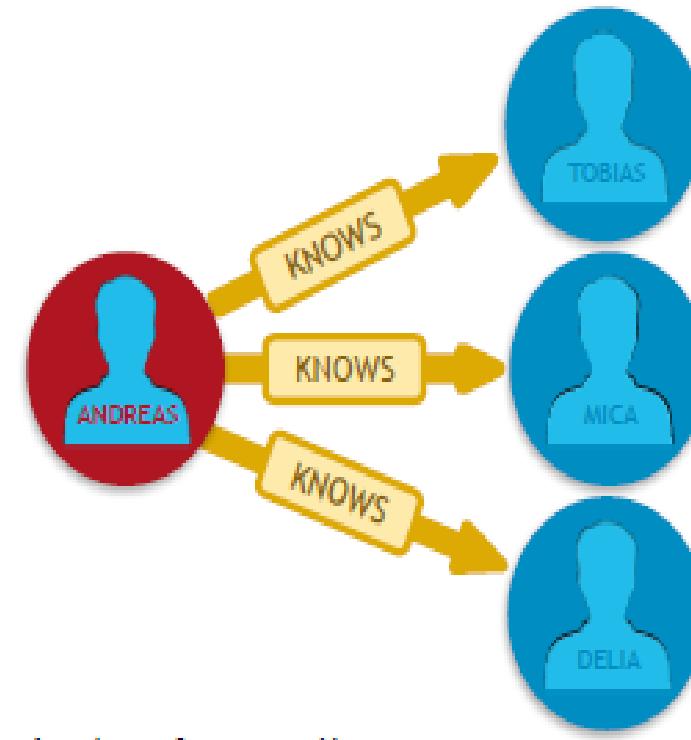


# Relational Vs Graph Models

Relational Model



Graph Model



Index free adjacency

# 5 Signs You Need a Graph Database

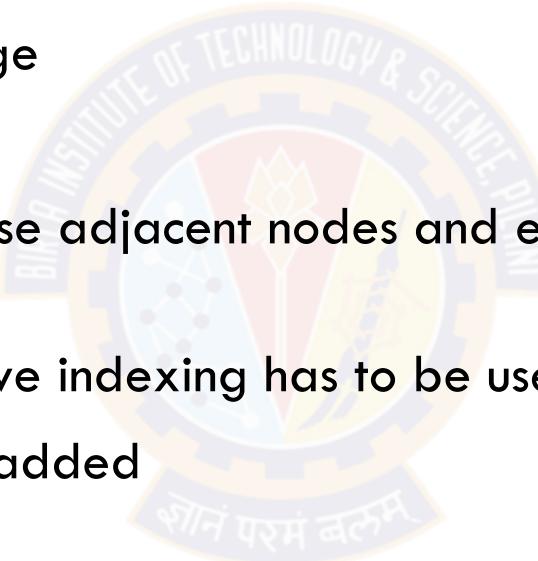
Is your traditional database not up to the task of deciphering your complex data? Here are five of the telltale signs:

1. Hard-to-navigate interconnected data
2. Lack of real-time insights
3. Inflexible data structures
4. Inefficient querying
5. Inability to decode a web of highly connected data

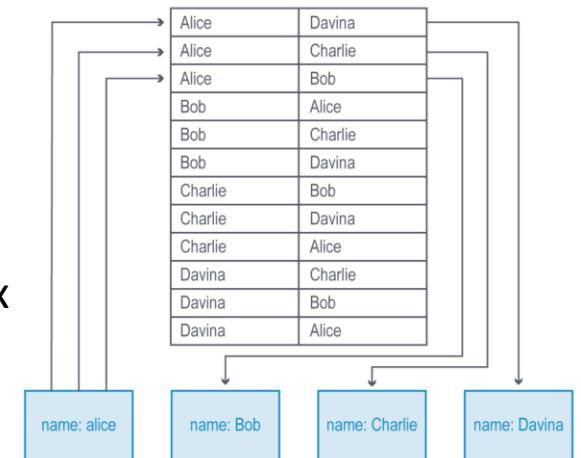


# Native vs Non-Native Graph storage

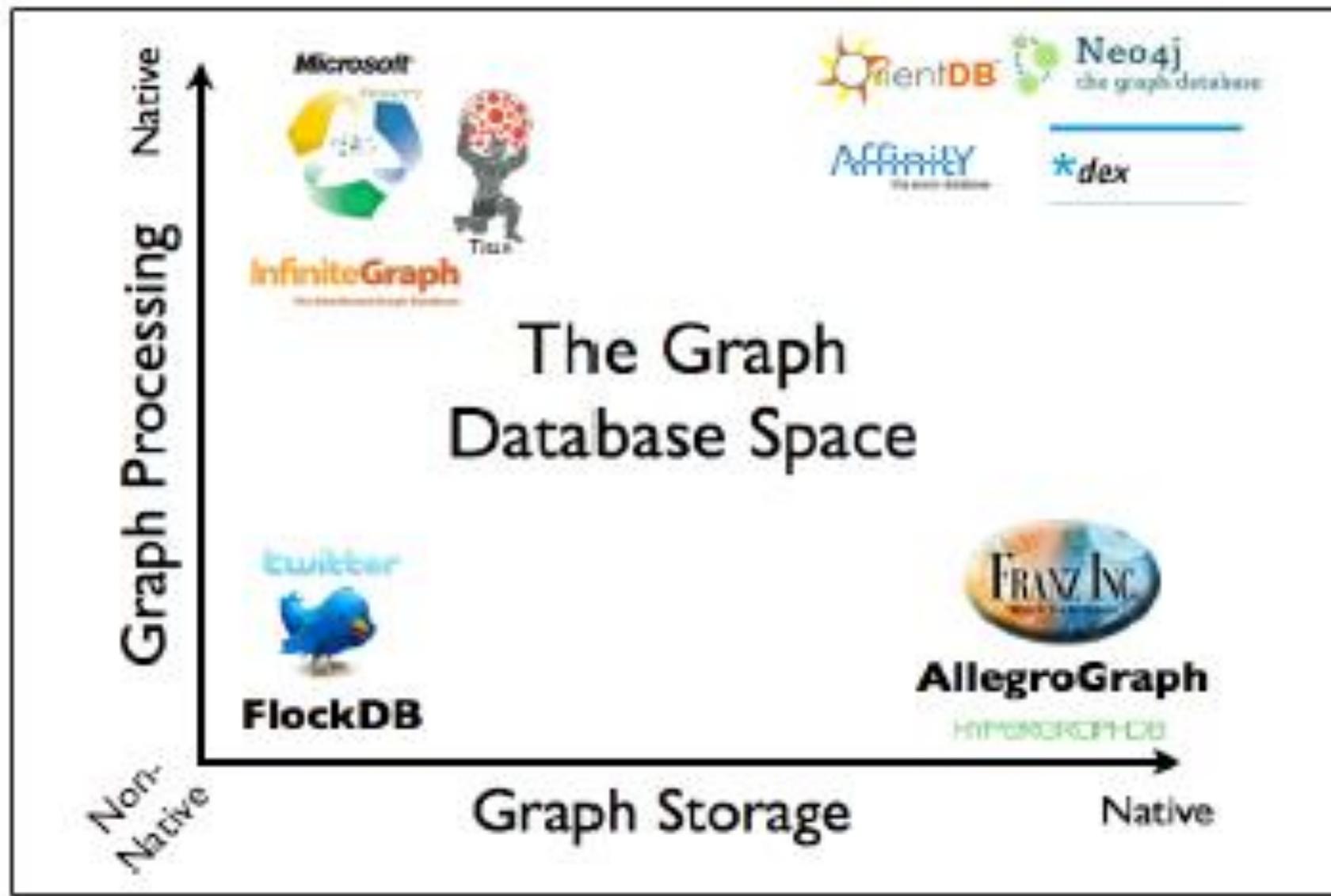
- Non-native graph computing platforms can use external DBs for data storage
  - e.g. TinkerPop is an in-memory DB + computing framework that can store in ElasticSearch, Cassandra etc.
- Native platform support built-in storage
  - e.g. Neo4j
- Native approach is much faster because adjacent nodes and edges are stored closer for faster traversal
  - In a non-native approach, extensive indexing has to be used
- Native approach scales as nodes get added



One-hop index

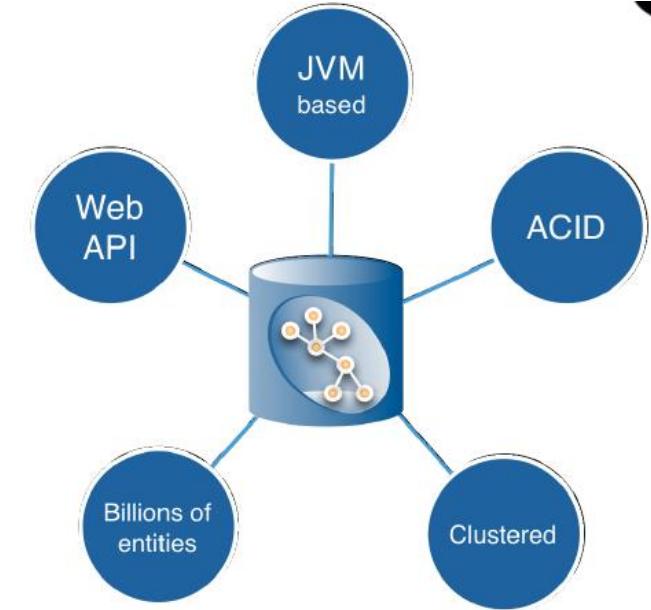


# Graphs for Storage and Processing



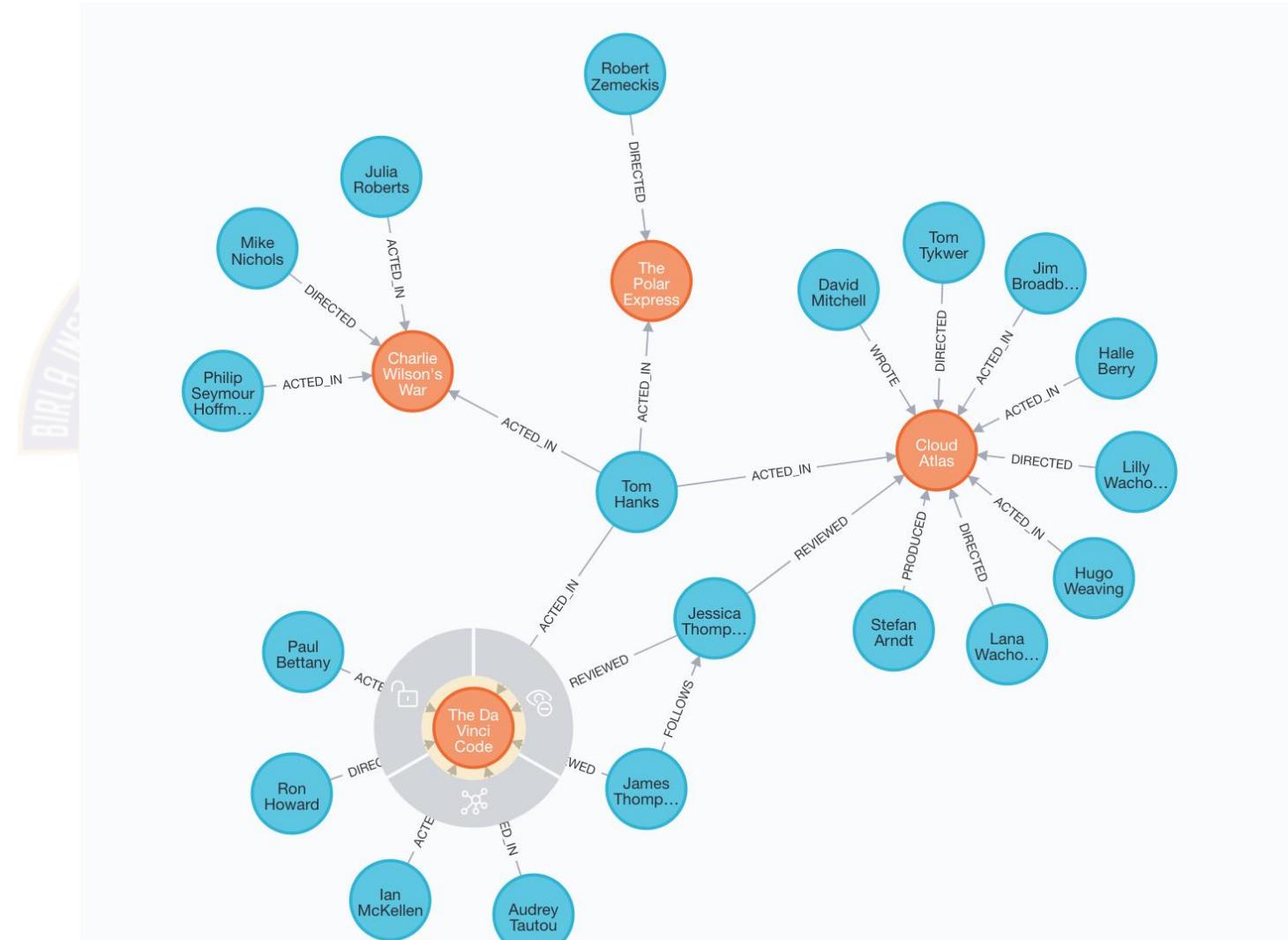
# What is Neo4J

- It's is a Graph Database supporting full ACID Transactions
- Embeddable in applications and server deployable
- Java based, Open sourced
- Schema free, bottom-up data model design
- Neo4j is stable
- In 24/7 operation since 2003
- Neo4j is under active development
- High performance graph operations
- Supports the **Cypher** query language
- Traverses 1,000,000+ relationships/sec on commodity hardware
- No. of nodes and relationships decide Volume of data



# Neo4j / Cypher

- Cypher is a Declarative language for graph query
- Example: `match (:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie) where m.released > 2000 RETURN m limit 5`



Launch a free sandbox with dataset on [neo4j website](#)

# Neo4j / Cypher: More queries

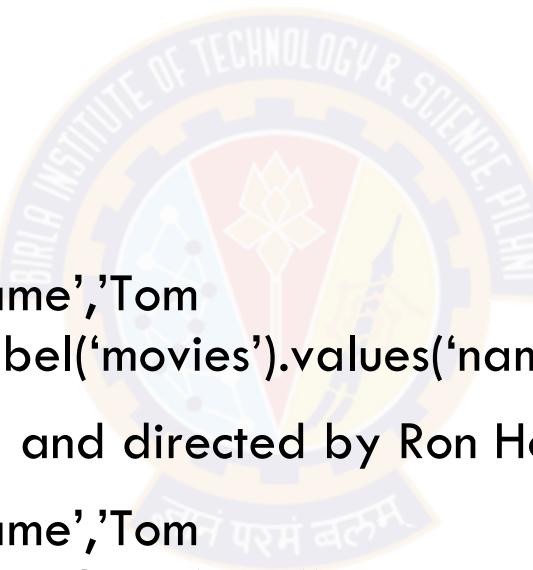
- Find movies that Tom Hanks acted in and directed by Ron Howard released after 2000
  - Match (:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m) where m.released > 2000 RETURN m limit 5
- Who were the other actors in the movie where Tom Hanks acted in and directed by Ron Howard released after 2000
  - Match (:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m), (p:Person)-[:ACTED\_IN]->(m) where m.released > 2000 RETURN p limit 5

- Neo4j Enterprise Edition on AWS
- Neo4j - The Easiest Way to Graph on MS Azure
- NEO4J AURA on Google Cloud



# Apache Tinkerpop / Gremlin

- TinkerPop is a computing platform that connects to GraphDBs that actually store the nodes and edges. Built-in TinkerGraph stores in-memory data only.
- Gremlin is the query language (with traversal machine) that supports Declarative and Imperative flavours
- Sample queries
  - movies where Tom Hanks has acted
    - `g.V().hasLabel('person').has('name','Tom Hanks').outE('ACTED_IN').hasLabel('movies').values('name')`
  - movies where Tom Hanks has acted and directed by Ron Howard
    - `g.V().hasLabel('person').has('name','Tom Hanks').outE('ACTED_IN').inE('DIRECTED').has('name','Ron Howard').outE('DIRECTED').values('name')`



ACTED\_IN

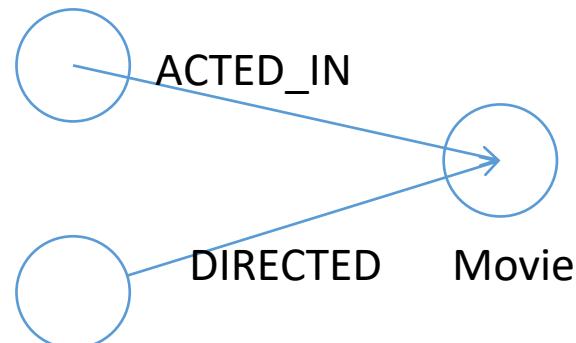


Person

Movie

DIRECTED

Person: Tom Hanks



DIRECTED

Movie

Person: Ron Howard

# NoSQL Certification Trainings

## 1. MongoDB

Free MongoDB courses - MongoDB University - <https://learn.mongodb.com/>

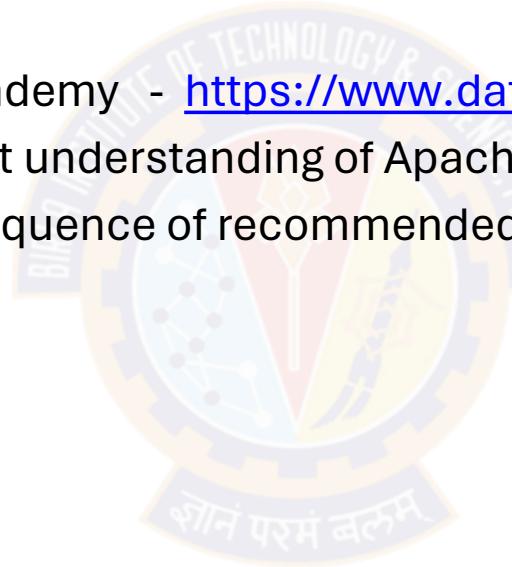
## 2. Cassandra

Free Cassandra courses – Datastax Academy - <https://www.datastax.com/dev/academy>

Multiple Learning Paths - Gain an expert understanding of Apache Cassandra™

Each Learning Path is composed of a sequence of recommended courses for your role

- Administrator Certification
  - ✓ DS201
  - ✓ DS210
- Developer Certification
  - ✓ DS201
  - ✓ DS220



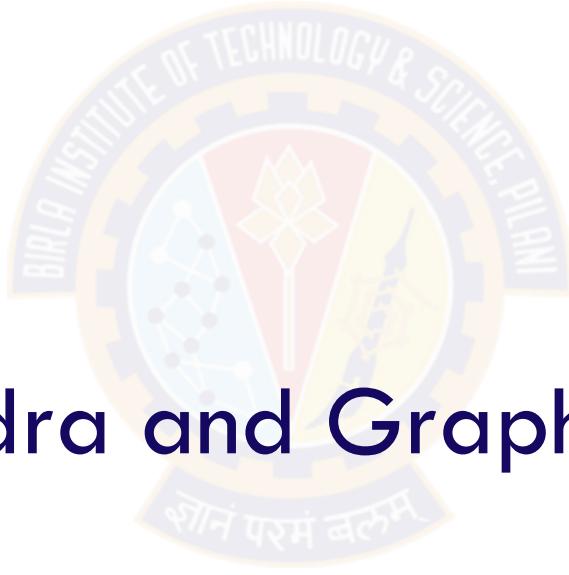
## 3. Neo4j

neo4j Graphacademy - <https://graphacademy.neo4j.com/>

[Free, Self-Paced, Hands-on Online Training | Free Neo4j Courses from GraphAcademy](#)

# Summary

- NoSQL databases are useful when
  - ✓ you have to deal with large data sets
  - ✓ may need geographical distribution
  - ✓ No need for ACID transactions and need flexible consistency
- Choices between key-value, column based, document based, graph based data stores
- Graph DBs and computing models are very suitable when data sets are relationship heavy - can be modelled as large number of nodes and edges and queries are similar to graph traversal
  - ✓ Complex relation centric queries are possible
  - ✓ Graph traversal costs can be kept stable with data growth



## Next Session: Cassandra and Graph Database in detail



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 11-1: Apache Cassandra

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Introduction

- Initially developed by Facebook
  - Open-sourced in 2008
- Used by 1500+ businesses, e.g., Comcast, eBay, GitHub, Hulu, Instagram, Netflix, Best Buy, ...
- Column-family store
  - Supports key-value interface
  - Provides a SQL-like CRUD interface: CQL
- BASE consistency model is AP
  - Gossip protocol (constant communication) to maintain cluster health
  - Ring-based replication model

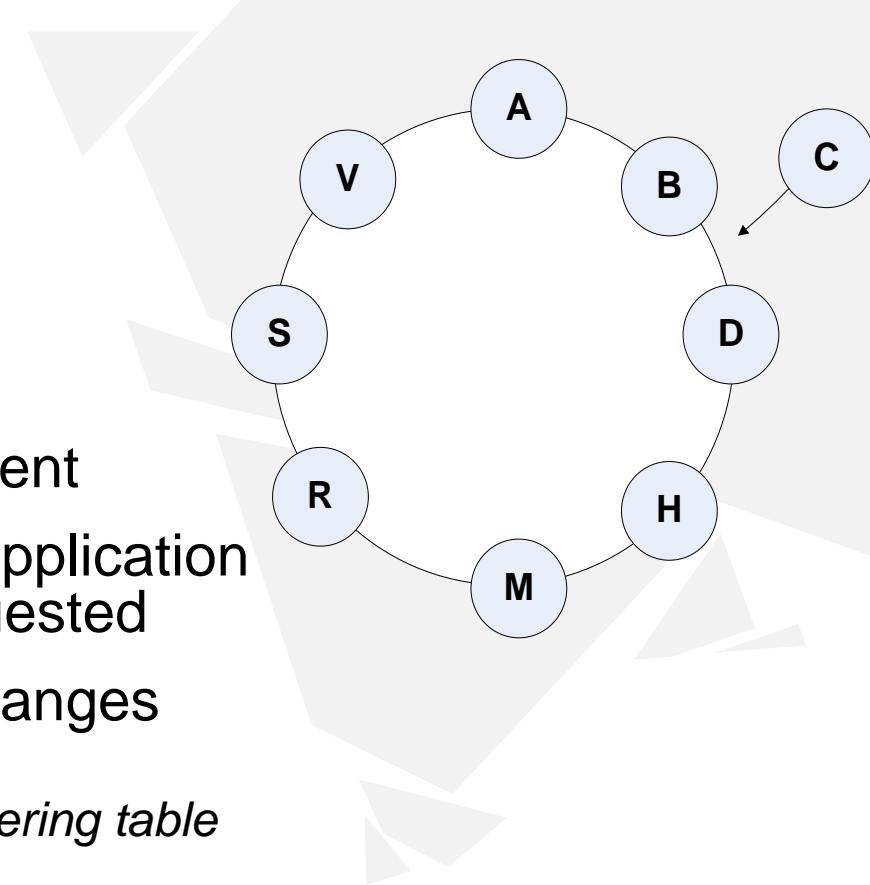
# Cassandra Installation

Download ( Datastax or Open source apache version)

- Stable Apache Cassandra from [https://cassandra.apache.org/\\_download.html](https://cassandra.apache.org/_download.html)
- You can get apache-cassandra-4.0.9-bin.tar.gz (Latest GA version)
- Untar the binary .gz file
  - tar -zxf apache-cassandra-4.0.9-bin.tar.gz
- Cassandra Configuration file - /conf/cassandra.yaml
  - You may change cluster name and leave all other values to default for a single node Cassandra
- Set JAVA\_HOME and PATH in ENV
  - export JAVA\_HOME=/usr/local/java/jdk1.8.0\_111
  - export PATH=\$JAVA\_HOME/bin:\$PATH
- Set CASSANDRA home and PATH in .bash\_profile
  - export CASSANDRA\_HOME=/home/user/apache-cassandra-4.0.9
  - export PATH=\$CASSANDRA\_HOME/bin:\$PATH
- Start cassandra
  - ./bin/cassandra -f  
(Watch for Startup complete)

# Cluster: Ring

- Cassandra Ring - Clustering for scaling
- Clients connect to any node of the cluster
- The connected node becomes the coordinator for the client
- The coordinator is to act as a proxy between the client application and the nodes (or replicas) that own the data being requested
- Each node is assigned different ranges of data - Token ranges
  - (1) `nodetool describecluster` (2) `nodetool ring` (3) `nodetool describering table`
- The coordinator routes the data to the proper node based on the token
- The partitioner does the job of creating the tokens
- Nodes can be added to the cluster online (redistributes token ranges)
- Nodes can be removed from a live cluster - decommissioning



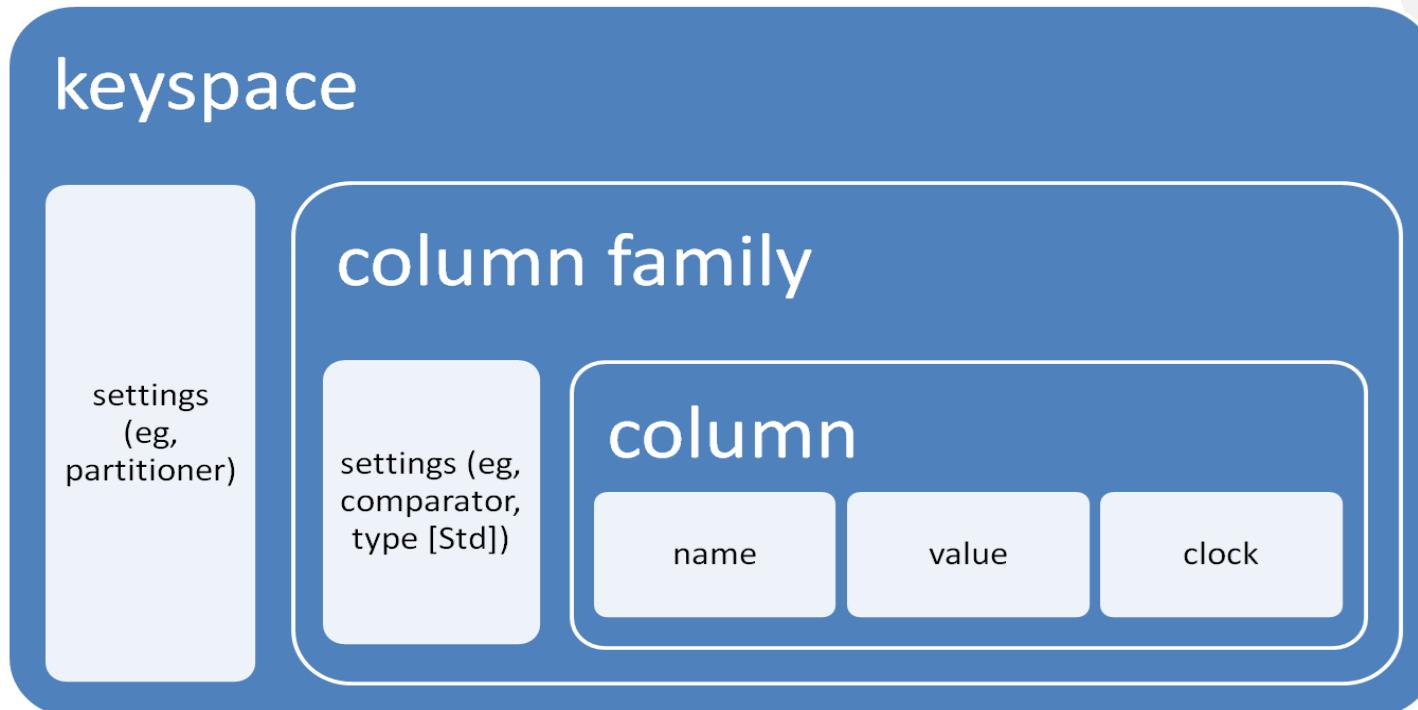
# CQL

- CQL - Cassandra Query Language
  - DDL and Query language for Cassandra
  - Very similar to SQL, but not SQL
- SELECT \* FROM users;
- SELECT does not allow JOINS (involving more than 1 table)
- Copy - imports/exports data from .csv files
- The native types supported by CQL are:
- ASCII | BIGINT | BLOB | BOOLEAN | COUNTER | DATE |
- DECIMAL | DOUBLE | DURATION | FLOAT |
- INET | INT | SMALLINT |
- TEXT | TIME | TIMESTAMP | TIMEUUID | TINYINT |
- UUID | VARCHAR | VARINT

<https://cassandra.apache.org/doc/latest/cassandra/cql/types.html>

# Cassandra Data Model

- Keyspace ~= database , typically one per application
- Some settings are configurable only per keyspace
- Row oriented - Each row is uniquely identified by a key



# Keyspaces

## Key spaces are

- Outermost container for **data** in Cassandra
- Top-level Namespace / Containers
- A bunch of attributes which define keyspace-wide behavior
- Stores the replication information
- Within the Keyspaces you have Table / Columnfamily
- CQL command use can be used to switch between different keyspaces
- Closest match is database in Relational Databases

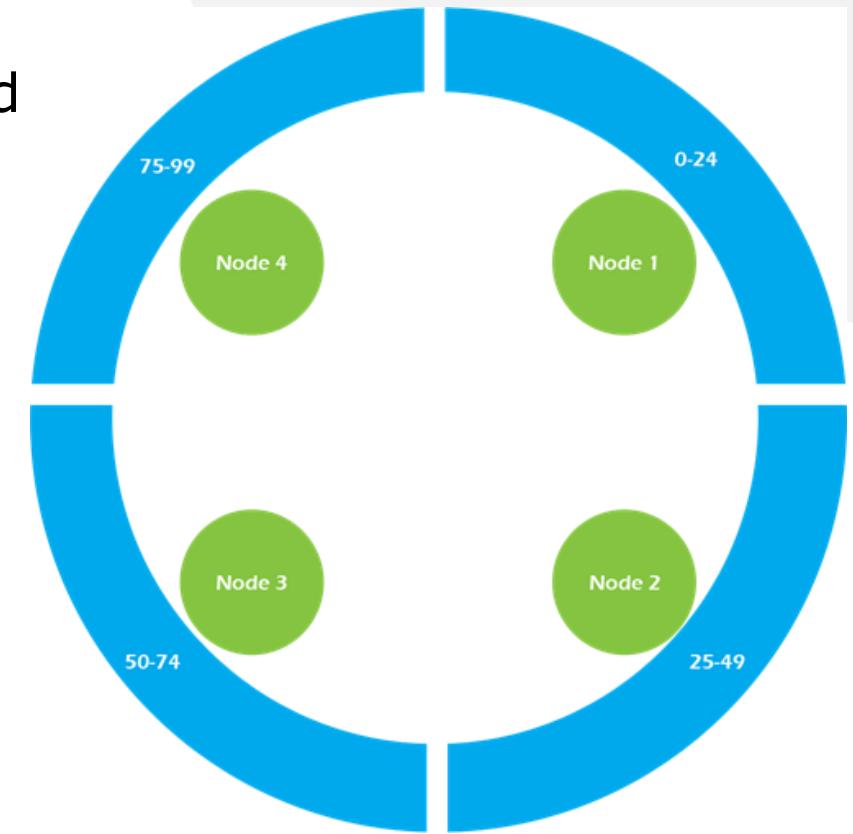
# Creating Keyspaces

```
CREATE KEYSPACE Keyspace name  
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

- Replication factor – It is the number of machines in the cluster that will receive copies of the same data.
- Replica placement strategy – Strategy to place replicas in the ring.
  - Simplestrategy - *rack – awarestrategy*
  - Network topology strategy - *datacenter – sharedstrategy*.

# Cassandra Data Model: Partitions

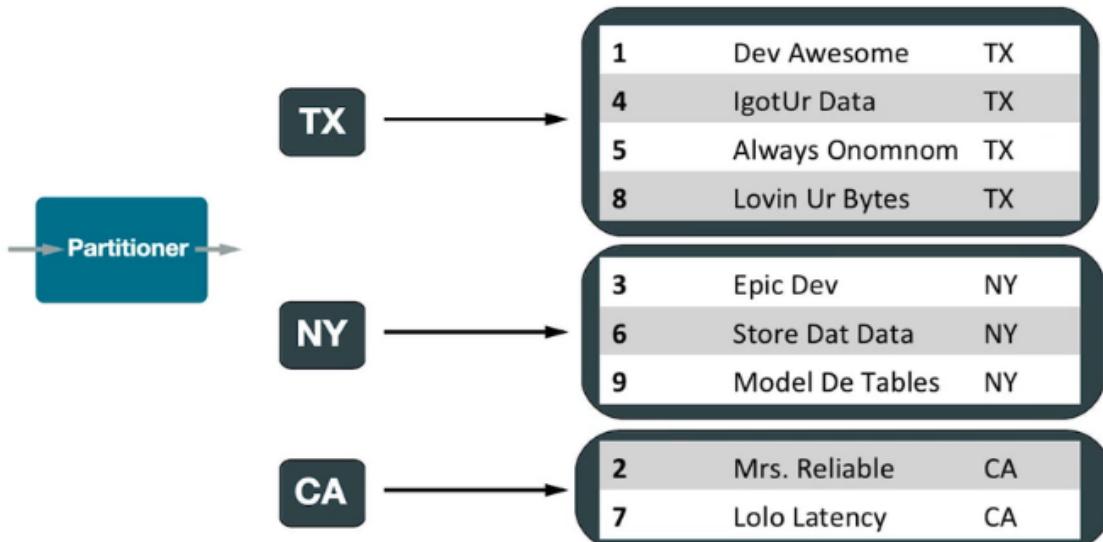
- Partitions are groupings of data that will be co-located on storage
- The partition key is responsible for distributing data among nodes.
- A partition key is the same as the primary key when the primary key consists of a single column.
- Partition keys belong to a node. Cassandra is organized into a cluster of nodes, with each node having an equal part of the partition key hashes.
- Imagine we have a four node Cassandra cluster. In the example cluster (ring):
  - Node 1 is responsible for partition key hash values 0-24
  - Node 2 is responsible for partition key hash values 25-49



# Cassandra Data Model: Partitions

- If only one primary key, it becomes the partition key
- Partitioner finds out a numeric Token value from the key

## Partitions



## Partitioners

- Murmur3Partitioner (default):** Uniformly distributes data across the cluster based on MurmurHash hash values.
- RandomPartitioner:** Uniformly distributes data across the cluster based on MD5 hash values.
- ByteOrderedPartitioner:** Keeps an ordered distribution of data lexically by key bytes

# High level goals of Cassandra Data Model

- **Spread data evenly around the cluster** – For Cassandra to work optimally, data should be spread as evenly as possible across cluster nodes. Distributing data evenly depends on selecting a good partition key and Partitioner.
- **Minimize the number of partitions to read** – When Cassandra reads data, it's best to read from as few partitions as possible since each partition potentially resides on a different cluster node. If a query involves multiple partitions, the coordinator node responsible for the query needs to interact with many nodes, thereby reducing performance.
- **Anticipate how data will grow, and think about potential bottlenecks in advance** – A particular data model might make sense when you have a few hundred transactions per user, but what would happen to performance if there were millions of transactions per user? Always “think big” when building data models for Cassandra and avoid knowingly introducing bottlenecks.

# Access Keys for Cassandra Tables

## Partition Key – Single Column

- These types of tables have primary keys that are also partition keys.
- Eg: **PRIMARY KEY** (videoid)

## Partition Key – Multiple Columns

- These types of tables have primary keys that are composed of partition and clustering keys
- Consists of one or more partition keys and zero or more clustering key components
- Always puts the partition key first and then the clustering key
- Eg: **PRIMARY KEY** (yyyymmdd, added\_date, videoid)

## Composite Partition keys

- A partition key in Cassandra can be made up of multiple columns.
- Eg: **PRIMARY KEY** ((tag, location), videoid)

# Cassandra Data Model: Clustering Columns

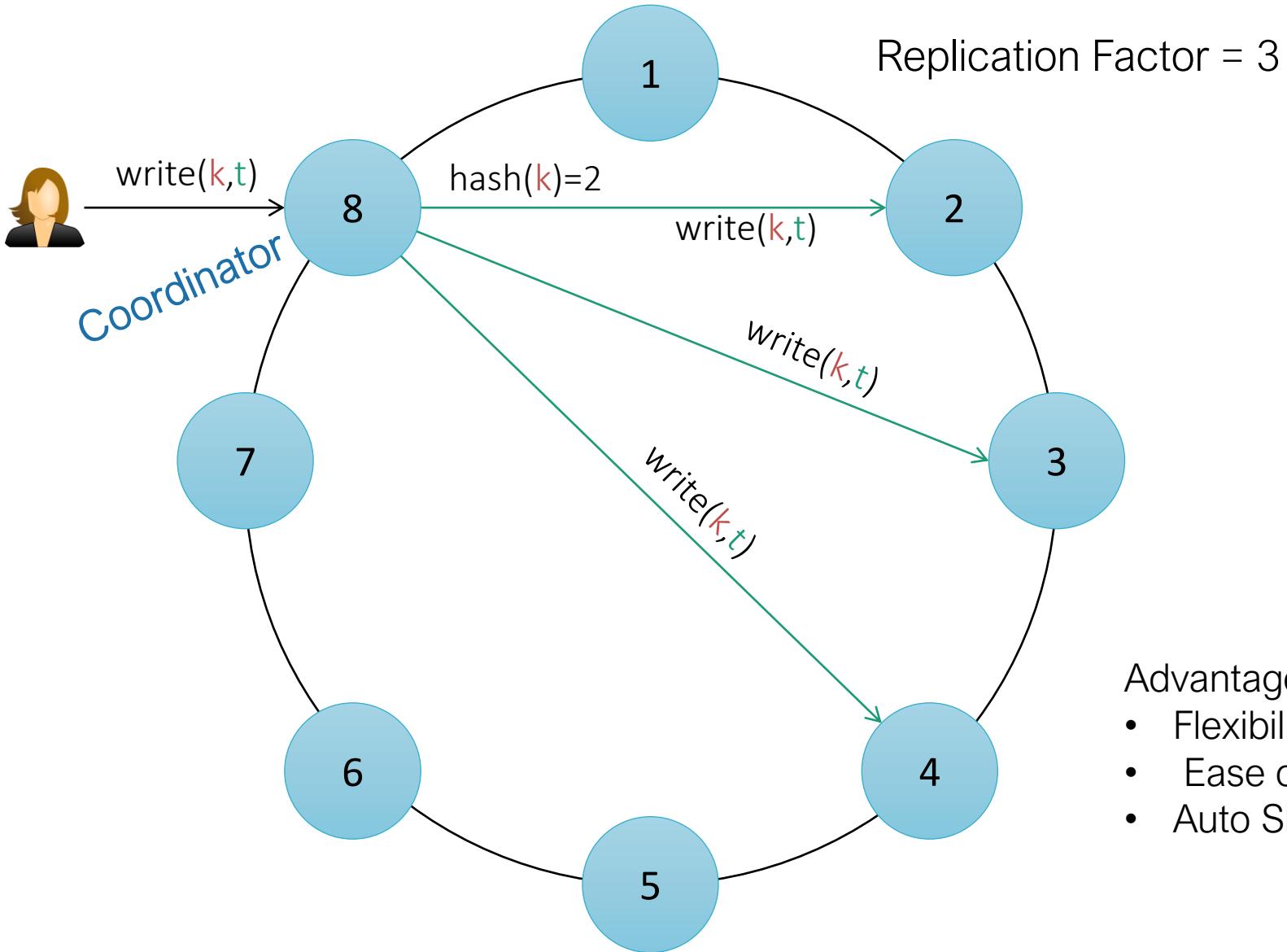
- Clustering is the process of sorting the data within a partition and is based on the columns defined as the clustering keys.
- The selection of clustering key columns depends on how we want to use the data in our application.
- All data within a partition is stored in continuous storage, sorted by clustering key columns
- Clustering columns follow ascending order by default
- Clustering allows us to change default ordering using ORDER BY
- Change ordering direction via WITH CLUSTERING ORDER BY
- Ordering is done when data is stored

## Allow Filtering

- Relaxes the querying on partition key constraint
- Causes Cassandra to scan all partitions
- Do not use it
  - Unless you really have to
  - Best on small data sets

```
CREATE TABLE users (
    state text,
    city text,
    name text,
    id uuid,
    PRIMARY KEY((state), city, name, id))
    WITH CLUSTERING ORDER BY(city DESC, name ASC);
```

# Cassandra's Ring Model

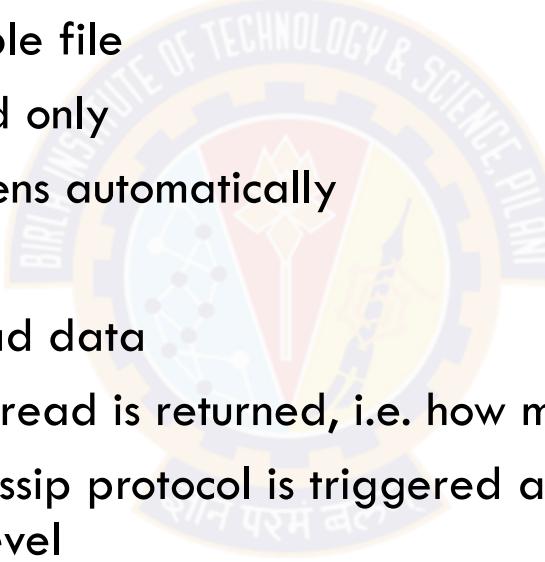


# Memtables, SSTables, and Commit Logs

- Commit Log for Durability
  - Once written never lost
  - Commit logs :all writes go in for recovery
- Memtable
  - memory-resident data structure
  - When contents become too big. Flushed into SSTable
- SSTable : File in Harddisk

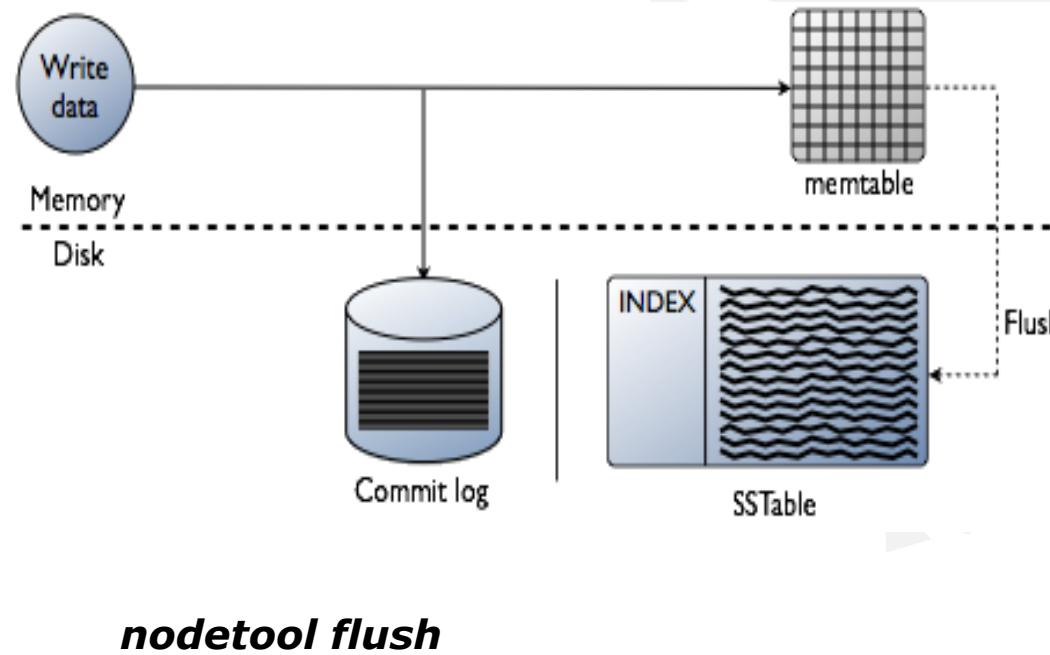
# Read / Write

- Writes
  - Written to commit log sequentially and deemed successful
  - Data is indexed and put into in-memory Memtable (one or more per Column Family)
  - Memtable is flushed to disk SSTable file
  - SSTable is immutable and append only
  - Partitioning and replication happens automatically
- Reads
  - Client connects to any node to read data
  - Consistency level decides when a read is returned, i.e. how many replicas should contain the same copy
  - Read repair: replication via a Gossip protocol is triggered as a client issues a read and Cassandra has to meet the required consistency level



# Internal Architecture: Write-Path

- A client issues a write request to a random node in the Cassandra cluster.
- The “Partitioner” determines the nodes responsible for the data
- On receiving a Write request, log it in disk commit log
- Make changes to appropriate memtables – In-memory representation of multiple key-value pairs



- Later, when memtable is full or old, flush to disk , remove commitLog
- Data File: An SSTable (Sorted String Table) – list of key value pairs, sorted by key
- Index file: An SSTable – (key, position in data sstable) pairs

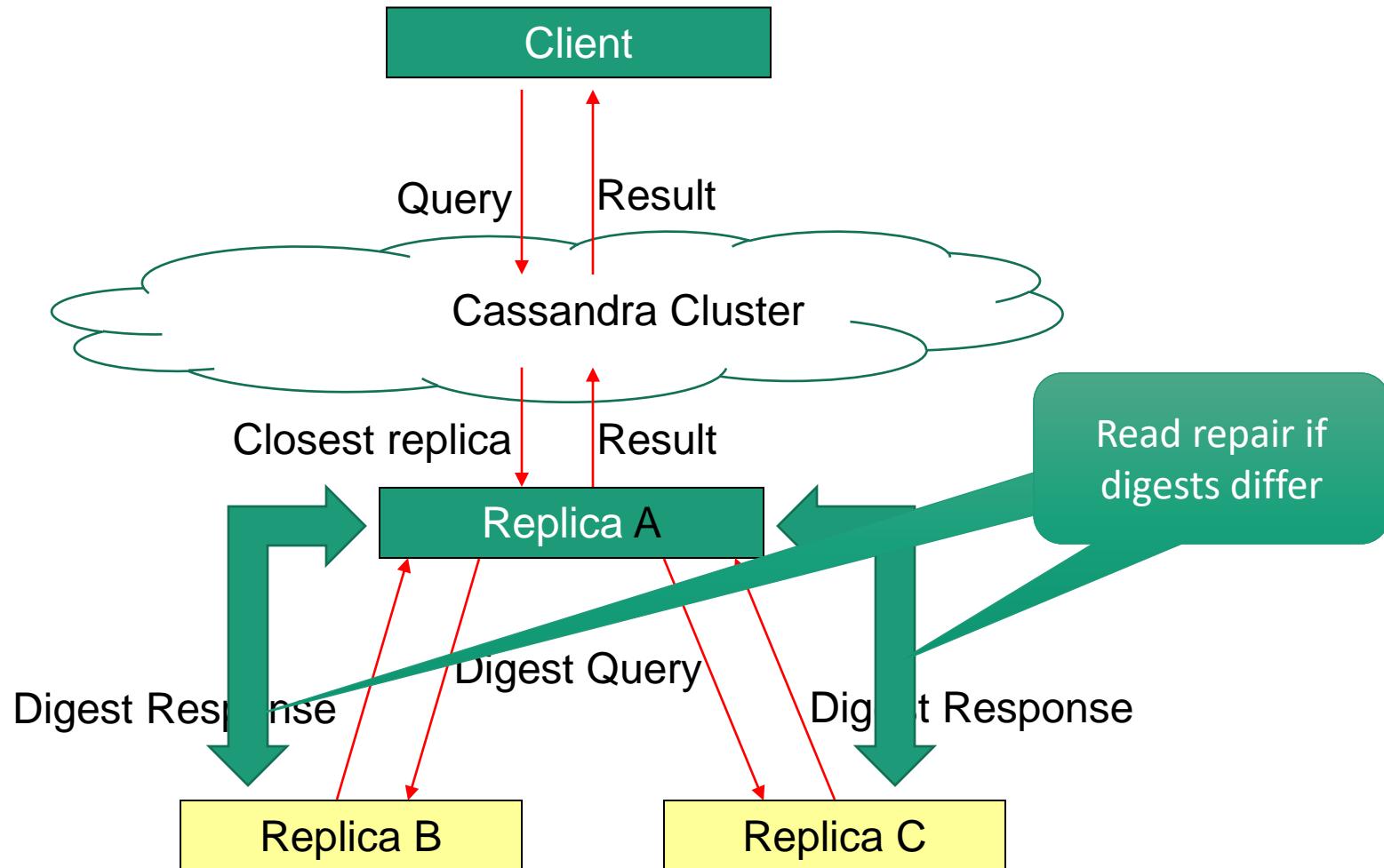
# Internal Architecture: Read-Path

- Read: Similar to writes, except
  - Need to touch commit log and multiple SSTables
  - Coordinator can contact closest replica (e.g., in same rack)
  - Coordinator also fetches from multiple replicas
    - Check consistency in the background, initiating a read-repair if any two values are different
    - Makes read slower than writes (but still fast)
    - Read repair: uses gossip (remember this?)
  - Slower than writes
- Delete: don't delete item right away
  - Add a tombstone to the log
  - Compaction will remove tombstone and delete item

# Replication/Consistency: Read-repair

- Network partitions can cause nodes to get out of sync
- Hinted-handoff works only for 3 hours
- Read repair is the process of making sure your data is consistent across all replicas
- It finds an answer to the question "*Oh, is this data consistent with the rest of the replicas?*" at the time of read
- Read repair is done by comparing the timestamps
- Read repair happens for reads with consistency level < ALL
- Choose data with latest timestamp, return that data to the client and update it on all other nodes
- Repair manually at least once in 10 days using ***nodetool repair*** command - (repairs one or more tables)

# Internal Architecture: Read Operation



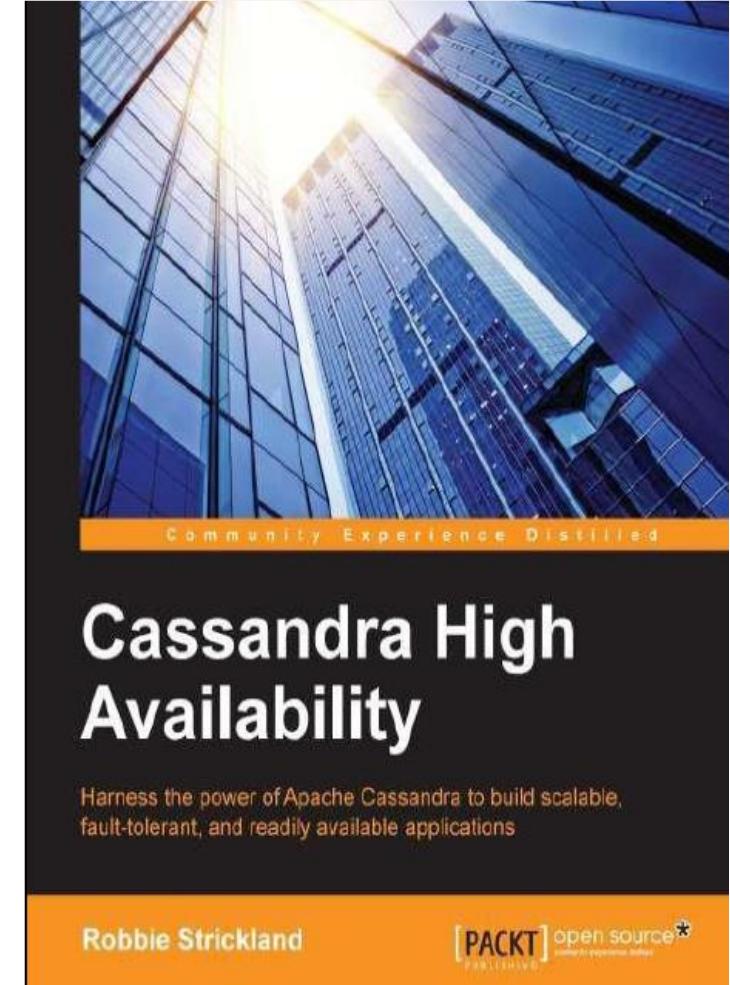
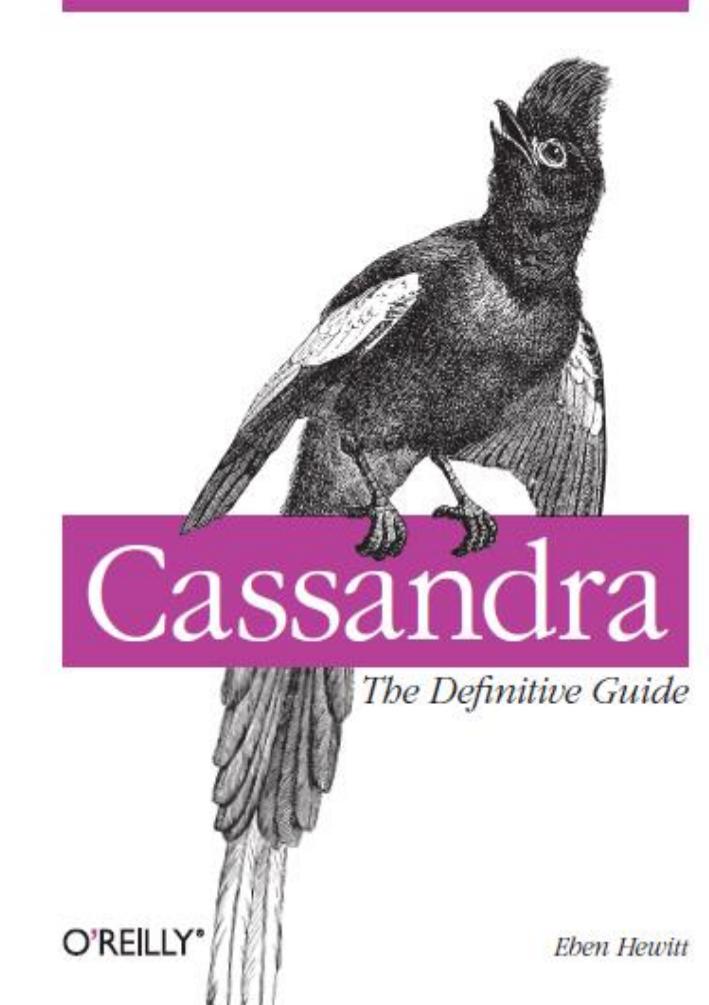
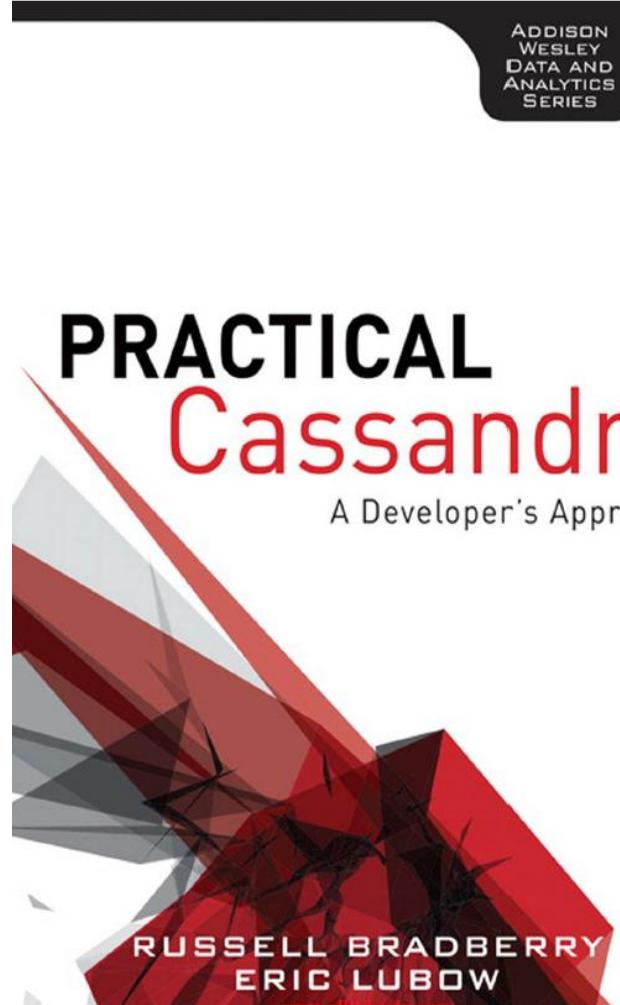
# Handson with CQLSH

- Creating Keyspaces and Column families
- Demonstration of TTL
- Flexible schema
- Update, Insert, Delete on tables
- Load data from files
- Aggregation
- Indexing
- Dropping tables and Keyspaces



Next Session:  
Introduction to Neo4j Graph database

# Cassandra Books





**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 11-2: Spark - Part 1

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **Introduction**
- **Getting started**
  - ✓ Setup
- RDDs
  - ✓ Transformations
  - ✓ Actions
  - ✓ Programming
- Sample programs



# BigData computing engines

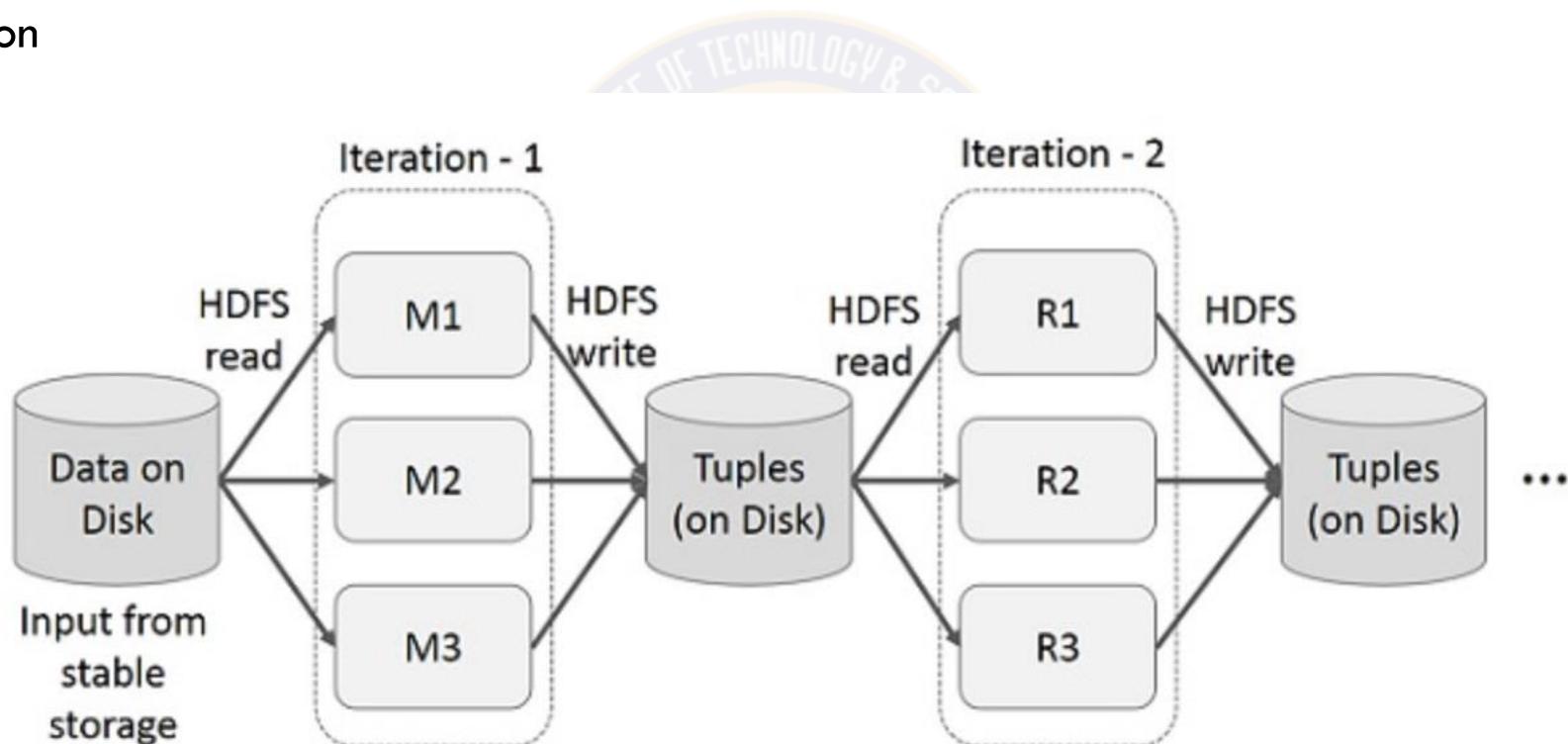
- Batch processing
  - Hadoop
- Stream processing
  - Storm
- Interactive processing
  - Tez, Impala
- Graph processing
  - Neo4j
- Machine learning
  - Mahout



Is it possible to create a general purpose computing engine for Big Data analysis ?  
Answer: Spark

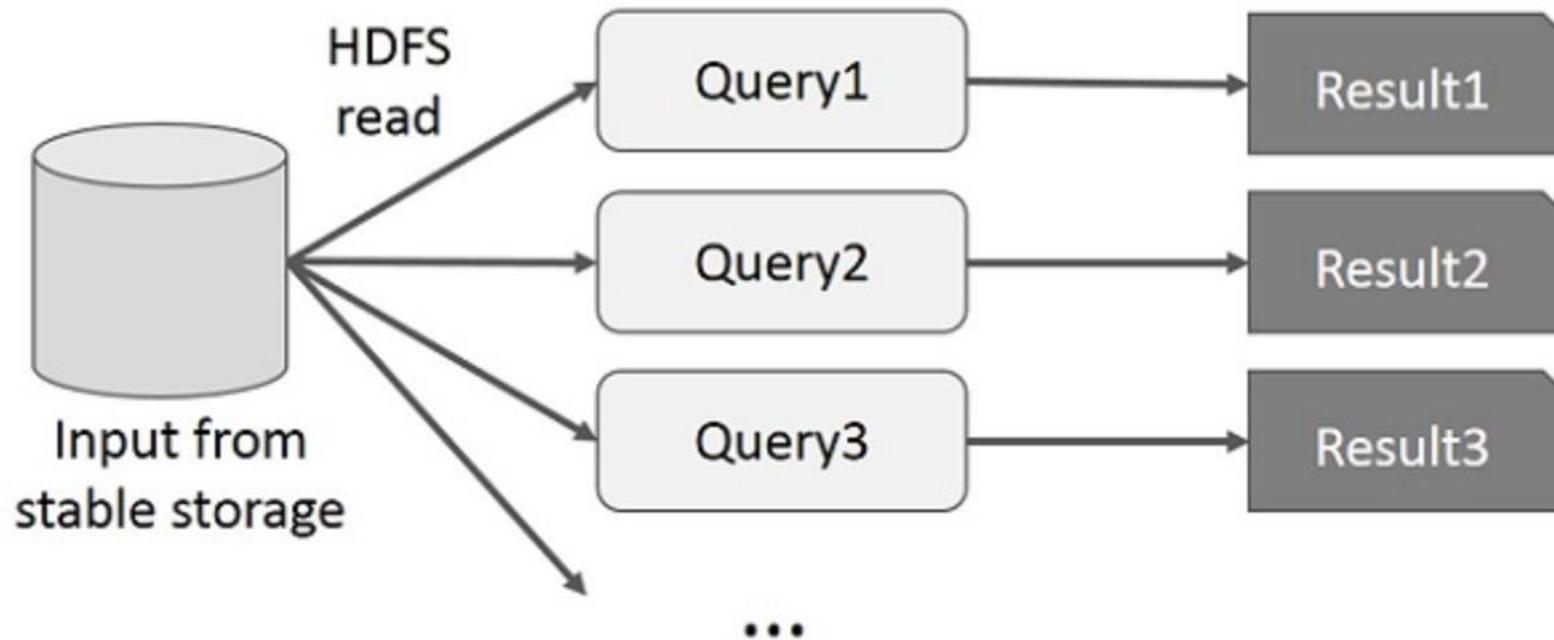
# Hadoop MapReduce inefficiencies – Iterative applications

- Iterative applications incur significant overheads
  - ✓ Serialization
  - ✓ Disk I/O
  - ✓ Replication



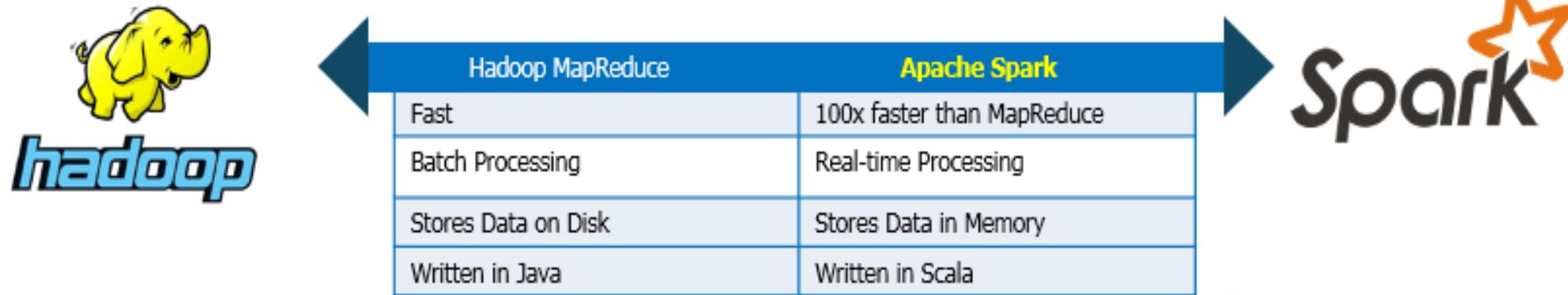
# Hadoop MapReduce inefficiencies – Interactive applications

- Interactive operations incur significant overheads with each query reading from stable storage and disk I/O will dominate application execution time



# Spark

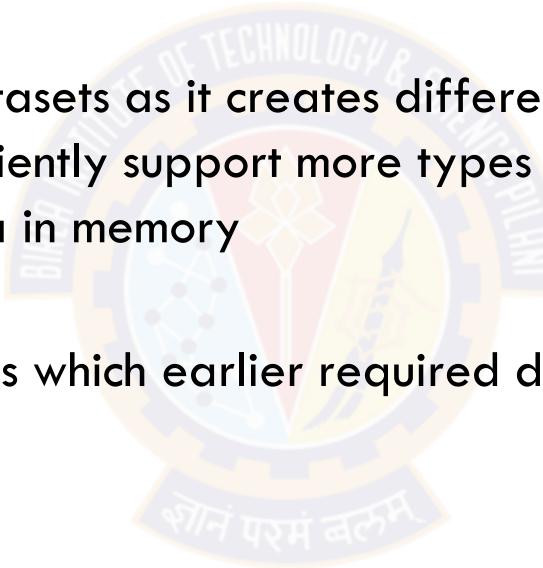
- In-Memory Cluster Computing for **Iterative** and **Interactive** Applications
- Provides programming abstraction and parallel runtime to hide complexities of fault-tolerance
- “Here’s an operation, run it on all of the data”, I don’t care where it runs (you schedule that)
- Originally developed in 2009 in UC Berkeley’s AMP Lab
- Fully open sourced in 2010 – now a Top Level Project at the Apache Software Foundation



# Spark

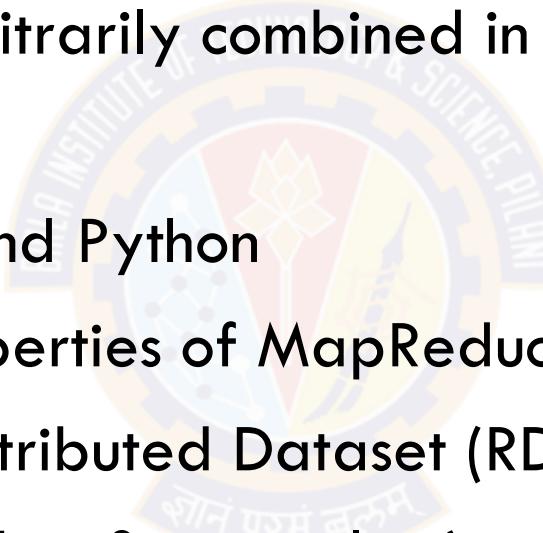


- A unified analytics engine for large-scale data processing
- Cluster computing platform
  - ✓ Designed to be fast and general purpose
- Speed
  - ✓ Is important in processing large datasets as it creates difference when data is being explored
  - ✓ Extends MapReduce model to efficiently support more types of computations
  - ✓ Runs computations by keeping data in memory
- Generality
  - ✓ Covers a wide variety of workloads which earlier required different distributed systems
  - ✓ Including
    - ❖ Batch applications
    - ❖ Iterative algorithms
    - ❖ Interactive queries
    - ❖ Streaming
  - ✓ Easy and inexpensive to combine different processing types in data pipelines



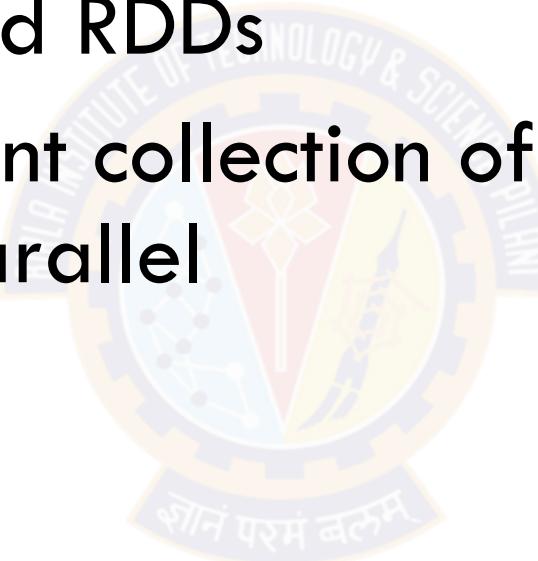
# Spark

- Processing engine; instead of just “map” and “reduce”, defines a large set of *operations known as transformations & actions*
  - Operations can be arbitrarily combined in any order
- Open source software
- Supports Java, Scala, R and Python
- Retains the attractive properties of MapReduce
- Key construct: Resilient Distributed Dataset (RDD)
  - Fault tolerant (for crashes & stragglers)
  - Data locality by partitioning
  - Scalability by partitioning

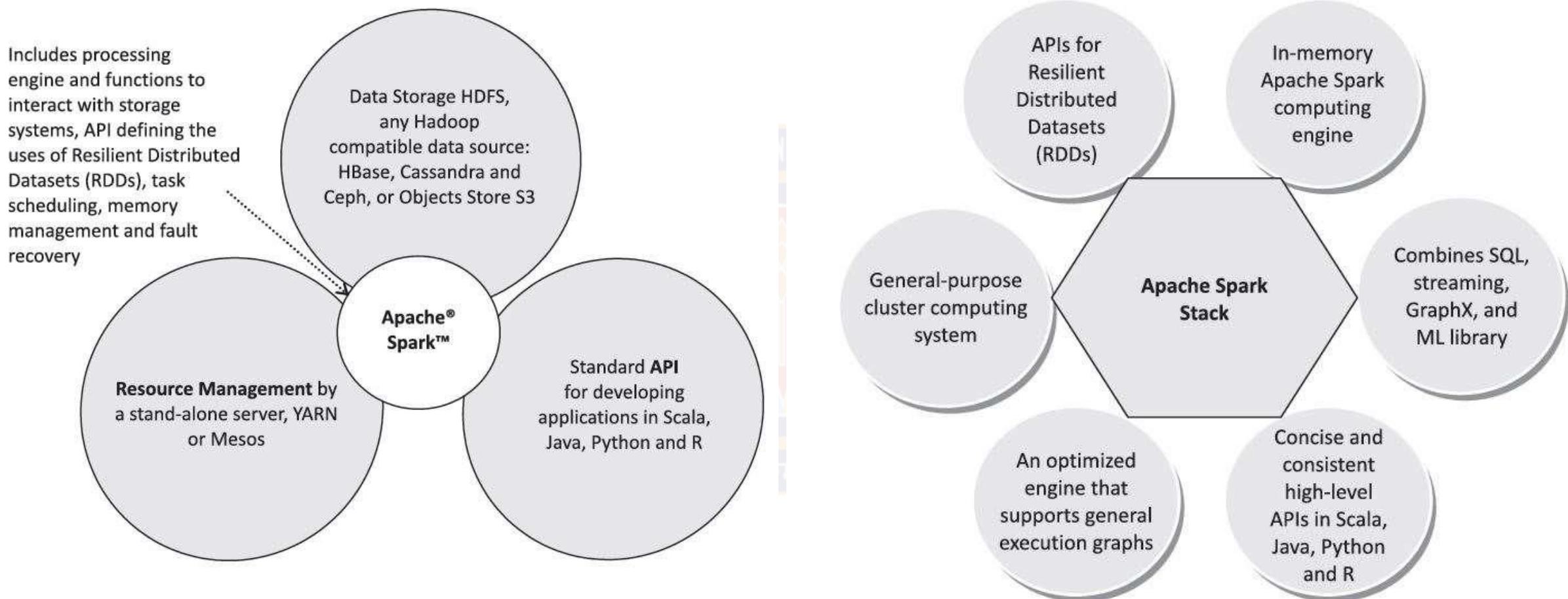


# RDDs

- Spark revolves around RDDs
- RDDs are fault-tolerant collection of elements that can be operated on in parallel



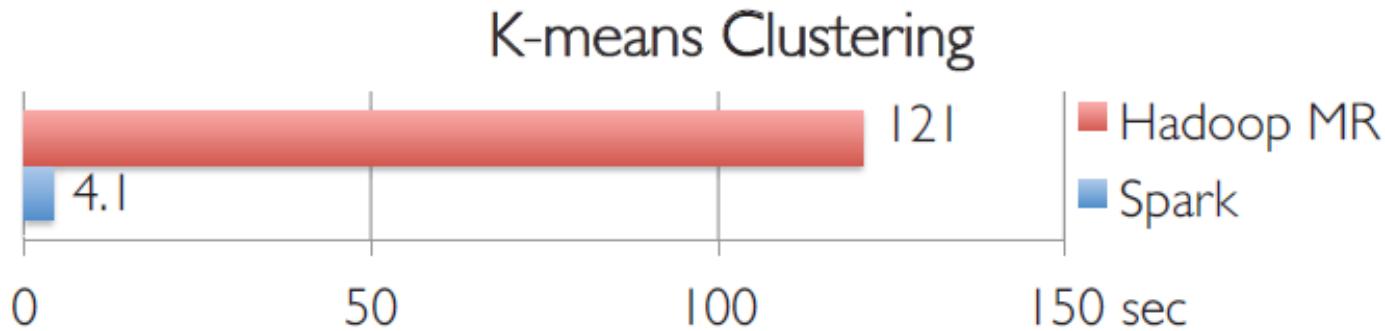
# Main Components of Spark



# Spark and MapReduce – Execution Time Comparison

## In-Memory Can Make a Big Difference

- Two iterative Machine Learning algorithms:



# Topics for today

- Introduction
- **Getting started**
  - ✓ Setup
- RDDs
  - ✓ Transformations
  - ✓ Actions
  - ✓ Programming
- Sample programs



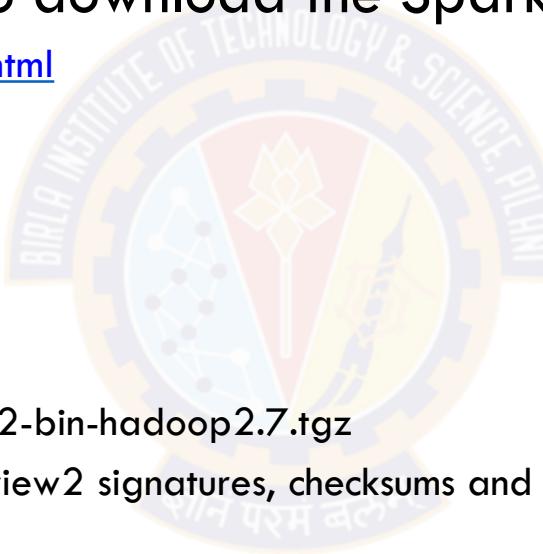
# Getting Started

- Spark can be used from Python, Java or Scala
  - Spark is written in Scala and runs on JVM
  - Spark can be run in local mode (standalone) or cluster mode
  - Spark can be run on both Windows and Unix like systems
- 
- Requirements
    - ✓ It's easy to run locally on one machine — all you need is to have java installed on your system PATH, or the JAVA\_HOME environment variable pointing to a Java installation
    - ✓ Spark runs on Java 8, Python 2.7+/3.4+ and R 3.1+. For the Scala API, Spark 2.4.5 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

# Getting Started (2)

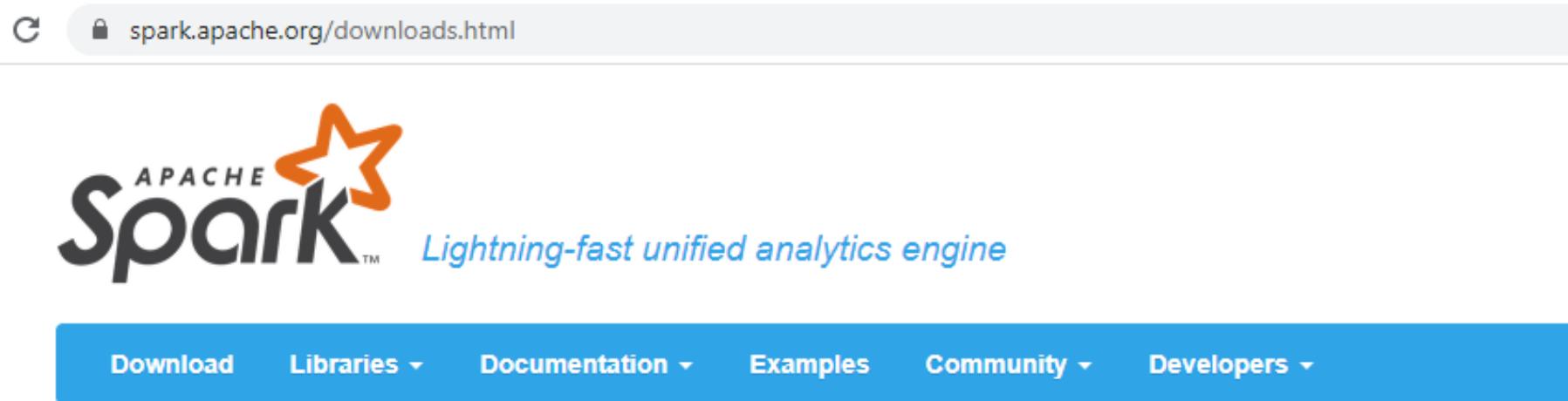
## Download

- Needs to download and unpack the tar ball
- Go to the Spark homepage to download the Spark release
  - ✓ <https://spark.apache.org/downloads.html>
- Steps -
  - ✓ Choose a Spark release:
  - ✓ Choose a package type:
  - ✓ Download Spark: spark-3.0.0-preview2-bin-hadoop2.7.tgz
  - ✓ Verify this release using the 3.0.0-preview2 signatures, checksums and project release KEYS.



# Getting Started (3)

## Download

A screenshot of a web browser showing the Apache Spark download page at spark.apache.org/downloads.html. The page features the Apache Spark logo and tagline "Lightning-fast unified analytics engine". A blue navigation bar at the top includes links for Download, Libraries, Documentation, Examples, Community, and Developers. The main content area is titled "Download Apache Spark™" and provides step-by-step instructions for downloading the software.

spark.apache.org/downloads.html

APACHE   
**Spark**™ *Lightning-fast unified analytics engine*

Download   Libraries ▾   Documentation ▾   Examples   Community ▾   Developers ▾

### Download Apache Spark™

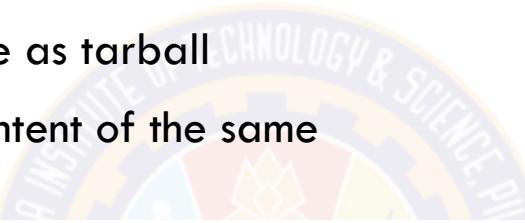
1. Choose a Spark release: [3.0.0-preview2 \(Dec 23 2019\) ▾](#)
2. Choose a package type: [Pre-built for Apache Hadoop 2.7 ▾](#)
3. Download Spark: [spark-3.0.0-preview2-bin-hadoop2.7.tgz](#)
4. Verify this release using the 3.0.0-preview2 [signatures](#), [checksums](#) and [project release KEYS](#).

Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12.

# Getting Started (4)

## Unpack

- The tarball .tgz will get downloaded
- Open the terminal and unzip the tar ball
- Result into new directory with same name as tarball
- Change into the directory and list the content of the same



```
Applications Places Terminal Wed 12:35
[csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7]
File Edit View Search Terminal Help
[csishydlab@apache-spark ~]$ ls Downloads/
CentOS-7-x86_64-DVD-1908.torrent      Spark-DataFrame(1).html
CentOS-7-x86_64-Everything-1908.iso   Spark-DataFrame.html
kafka_2.12-2.4.0.tgz                 Spark-DataFrame.ipynb
kafka_2.12-2.4.1.tgz                 spark-streaming-kafka-0-8-assembly_2.11-2.4.4(1).jar
[csishydlab@apache-spark ~]$ cd ~
[csishydlab@apache-spark ~]$ ls
data      kafka_2.12-2.4.0      output      scala-2.10.1.tgz
Desktop   kafka_2.12-2.4.0.tgz  Pictures    spark-2.4.4-bin-hadoop2.7
Documents  logs                  Public     spark-2.4.4-bin-hadoop2.7.tgz
Downloads Music                 python_code  spark_kafka_demo.logs
[csishydlab@apache-spark ~]$ cd spark-2.4.4-bin-hadoop2.7/
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ ls
bin      data      jars      LICENSE  NOTICE  R      regression_metrics_example.py  sample_linear_regression_data.txt  yarn
conf    examples  kubernetes  licenses  python  README.md  RELEASE
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$
```

## Getting Started (5)

## Spark Shells

- Interactive shell available for interactive data analysis
    - ✓ Similar to other shells like R , Bash or Windows command prompt
  - Allows to interact with data that is distributed on disk or in memory across many machines
  - Provides both Python and Scala shells
    - ✓ Python – execute bin/pyspark
    - ✓ Scala – execute bin/spark-shell



# Distributed on disk or in memory across many machines

Applications Places Terminal

csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7

File Edit View Search Terminal Help

```
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ bin/pyspark
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
20/04/15 12:40:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

   / \
  / \ \
 /_ \_ \_ \_ \_ \_ \_ \_ \
version 2.4.4

Using Python version 2.7.5 (default, Aug 7 2019 00:51:29)
SparkSession available as 'spark'.
>>> 
```

# Topics for today

- Introduction
- Getting started
  - ✓ Setup
- RDDs
  - ✓ Transformations
  - ✓ Actions
  - ✓ Programming
- Sample programs



# RDDs

- An RDD is an immutable, partitioned, logical collection of records
  - Need not be materialized at creation, but rather contains information to rebuild a dataset from stable storage
  - Partitioning can be based on a key in each record (using hash or range partitioning)
  - Built from data from files for the first time
  - Or using transformations on other RDDs subsequently
  - Can be cached for future reuse

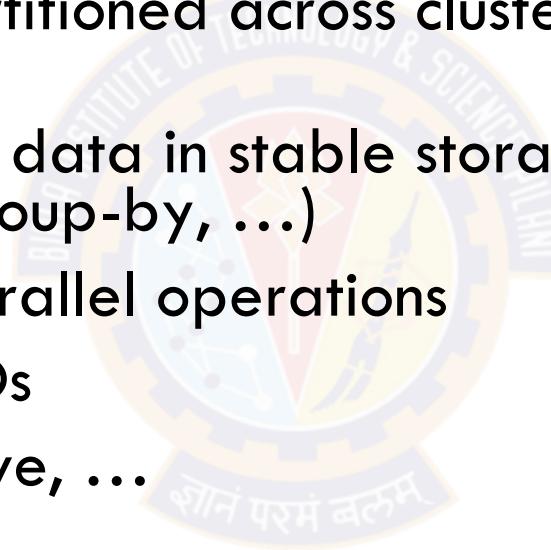
# RDDs Comparison with memory mapped files

A memory-mapped file contains the contents of a file in virtual memory. This mapping between a file and memory space enables an application, to modify the file by reading and writing directly to the memory.

Memory Mapped Files	RDDs
The file is confined to the memory of 1 system	The RDD spans the memory of multiple systems
Gets check pointed to disk by the Virtual memory system	Remain in memory when they are used in actions
Mutable - The contents can be modified	Immutable - Contents cannot be modified
No fault tolerance	Fault tolerant - Gets reconstructed from lineage when there is a failure
Remain mapped to the files from which it is created	Built from data files for the first time and after that works independent of the file

# Programming the RDDs

- Resilient distributed datasets (RDDs) are
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  - Can be cached across parallel operations
- Parallel operations on RDDs
  - Reduce, collect, count, save, ...
- Restricted shared variables
  - Accumulators, broadcast variables

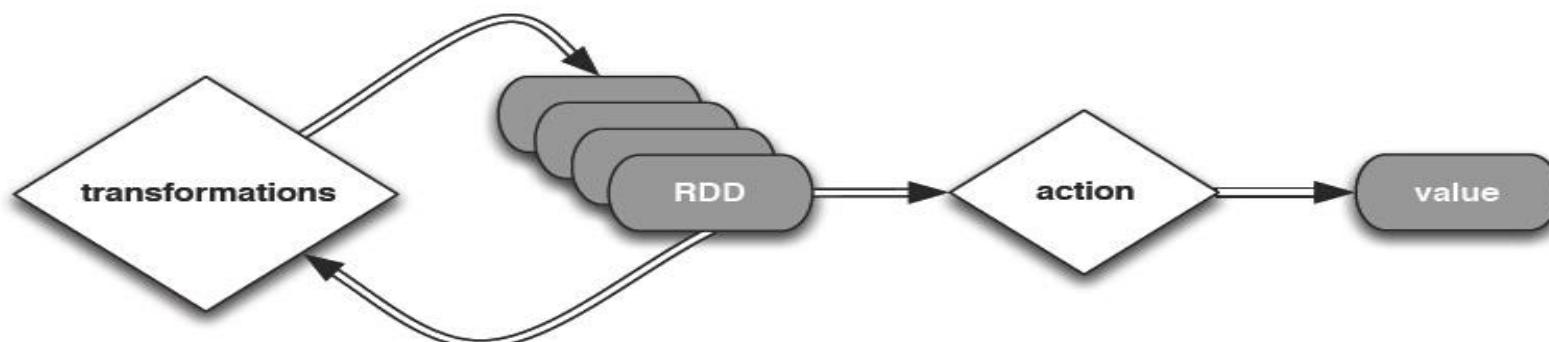


# How RDDs are created

## Spark Essentials: RDD

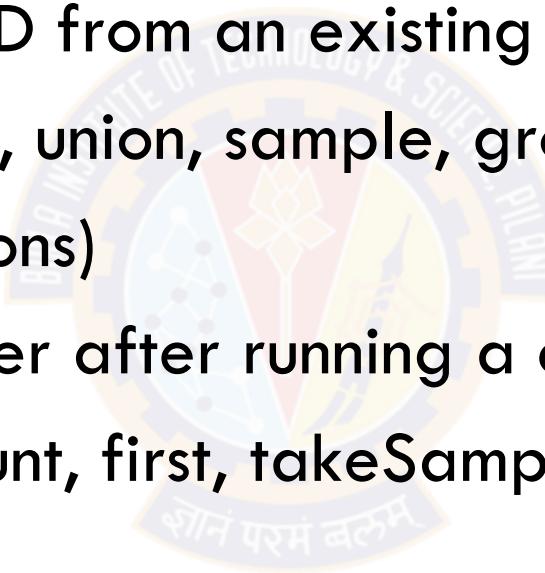
Spark can create RDDs from any file stored in HDFS or other storage systems supported by Hadoop, e.g., local file system, Amazon S3, Hypertable, HBase, etc.

Spark supports text files, SequenceFiles, and any other Hadoop InputFormat, and can also take a directory or a glob (e.g. /data/201404\*)



# RDD Operations

- Transformations
  - Creation of a new RDD from an existing
    - map, filter, distinct, union, sample, groupByKey, join, etc...
- Actions (Parallel Operations)
  - Return a value to driver after running a computation
    - collect, reduce, count, first, takeSample, foreach, etc



# RDD Transformations

## **Spark Essentials:** *Transformations*

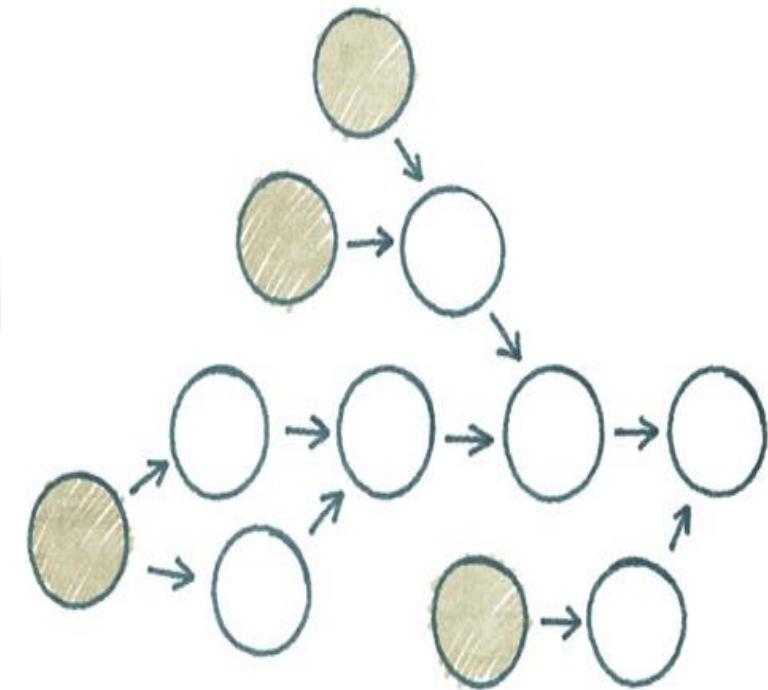
Transformations create a new dataset from an existing one

All transformations in Spark are *lazy*: they do not compute their results right away – instead they remember the transformations applied to some base dataset

- optimize the required calculations
- recover from lost data partitions

# Spark programs

- Spark programs consists of a set of DAGs (Directed Acyclic Graphs)
- Nodes are RDDs
- Edges are transformations
- Directed edges
  - Only in a single direction
- Acyclic
  - No looping



# RDD Transformations

## Spark Essentials: *Transformations*

<i>transformation</i>	<i>description</i>
<b>map( <i>func</i> )</b>	return a new distributed dataset formed by passing each element of the source through a function <i>func</i>
<b>filter( <i>func</i> )</b>	return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true
<b>flatMap( <i>func</i> )</b>	similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item)
<b>sample( <i>withReplacement</i>, <i>fraction</i>, <i>seed</i> )</b>	sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator <i>seed</i>
<b>union( <i>otherDataset</i> )</b>	return a new dataset that contains the union of the elements in the source dataset and the argument
<b>distinct( [ <i>numTasks</i> ] )</b>	return a new dataset that contains the distinct elements of the source dataset

# RDD Transformations

## Spark Essentials: Transformations

<i>transformation</i>	<i>description</i>
<b>groupByKey( [ numTasks ] )</b>	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs
<b>reduceByKey( func, [ numTasks ] )</b>	when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function
<b>sortByKey( [ ascending ], [ numTasks ] )</b>	when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
<b>join( otherDataset, [ numTasks ] )</b>	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
<b>cogroup( otherDataset, [ numTasks ] )</b>	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith
<b>cartesian( otherDataset )</b>	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

# RDD Actions

## Spark Essentials: Actions

action	description
<b>reduce(func)</b>	aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
<b>collect()</b>	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
<b>count()</b>	return the number of elements in the dataset
<b>first()</b>	return the first element of the dataset – similar to <i>take(1)</i>
<b>take(n)</b>	return an array with the first <i>n</i> elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
<b>takeSample(withReplacement, fraction, seed)</b>	return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, using the given random number generator seed

# RDD Actions

## Spark Essentials: Actions

action	description
<b>saveAsTextFile(path)</b>	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file
<b>saveAsSequenceFile(path)</b>	write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's Writable interface or are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
<b>countByKey()</b>	only available on RDDs of type (K, V). Returns a 'Map' of (K, Int) pairs with the count of each key
<b>foreach(func)</b>	run a function <code>func</code> on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

# Spark language support

- Spark handles highly accessible, simple APIs in

- ✓ Java
- ✓ Python
- ✓ Scala
- ✓ SQL



- Integrates easily with other big data tools / platforms like
  - ✓ Hadoop (HDFS) and other ecosystem tools
  - ✓ Kafka
  - ✓ AWS S3
  - ✓ Cassandra



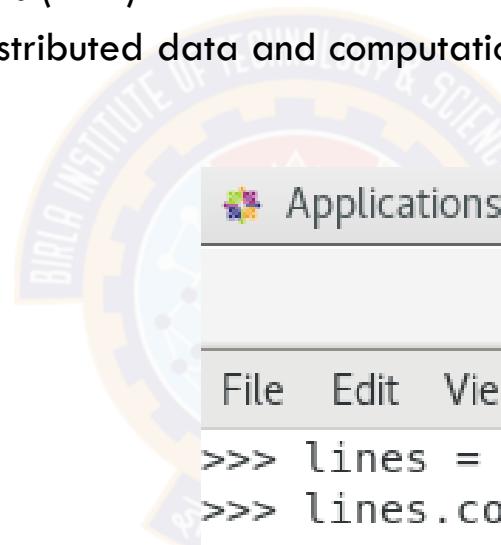
# Getting Started with Spark programs

## Executing in shells

- Computations are expressed on collections distributed across the cluster
  - ✓ Termed as Resilient Distributed Datasets (RDD)
  - ✓ Sparks fundamental abstraction for distributed data and computation
- Example

- ✓ Create RDD from local text file
- ✓ Do some very simple analysis on it
- ✓ Output the result
- ✓ Exit the shell , Ctrl – D

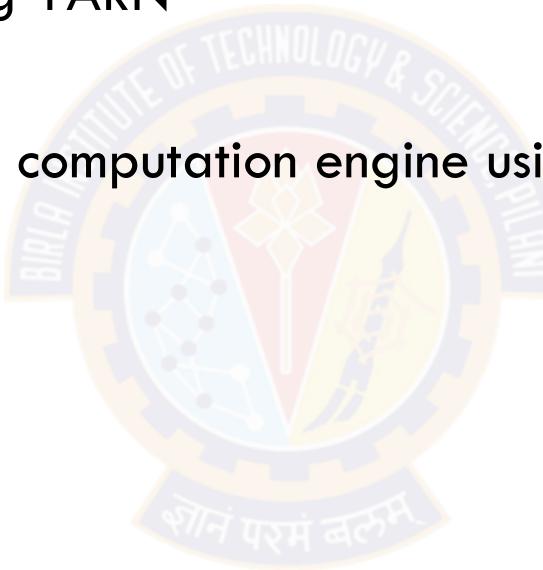
(2. in 01-Spark-RDD-Samples.txt)



```
Applications Places Terminal
File Edit View Search Terminal Help
>>> lines = sc.textFile("README.md")
>>> lines.count()
105
>>> lines.first()
u'# Apache Spark'
>>>
```

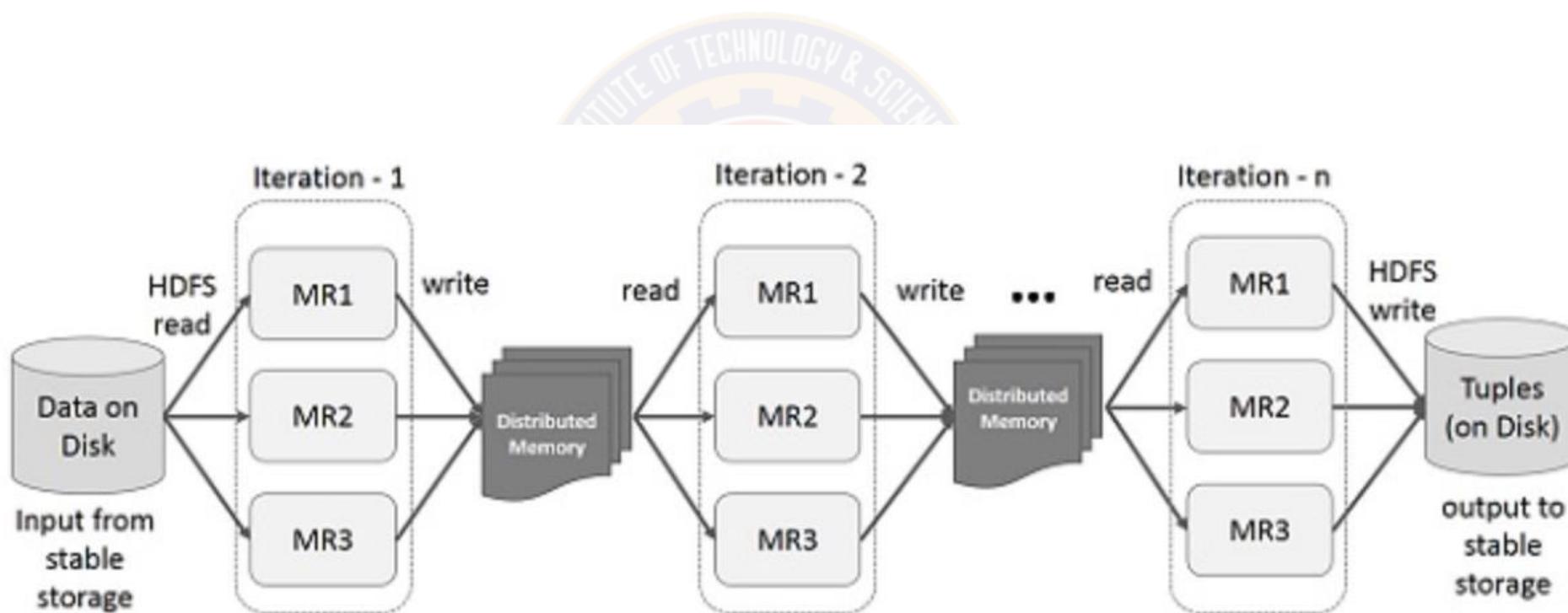
# Hadoop and Spark (Sec1, Sec2)

- Spark introduced to address speed issue of Hadoop computation
- Not a modified version of Hadoop but can use Hadoop as an option
  - For cluster management using YARN
  - For HDFS storage
- Spark has own in-memory cluster computation engine using RDD
- Supports more than MapReduce
  - SQL, Streaming, ML, Graph



# In-memory computing using RDD (1)

- Store intermediate results in distributed memory without complicating user programming
- Spill-over can be transparently stored on disk



# What is an RDD: Resilient Distributed Dataset

- Fundamental data structure in Spark
- Immutable distributed collection of objects
- Can be created from other RDDs, or
  - ‘parallelise’ an existing collection in driver program

```
val data = Array(1,2,3,4,5,6,7,8,9,10)
val rdd = sc.parallelize(data)
println("Number of Partitions: "+rdd.getNumPartitions)
println("Action: First element: "+rdd.first())
val rdd2 = rdd.max()
```

(Handson – 1. 01-Spark-RDD-Samples.txt)

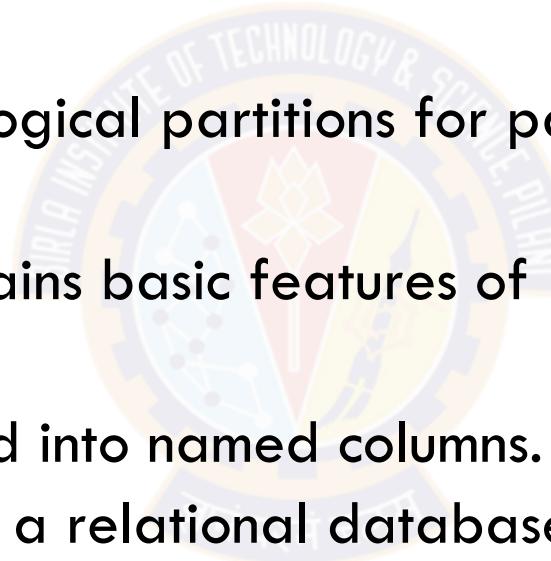
- reference a data set in an external storage: HDFS, HBase etc.

```
val dstfile = sc.textFile("/sample.txt")
dstfile.count()
```

(Handson – 3. 01-Spark-RDD-Samples.txt)

# RDDs as Resilient Distributed Datasets

- Resilient
  - A lineage graph of operations helps to reconstruct when a node fails and part of an RDD is lost
- Distributed
  - Each RDD is divided into logical partitions for parallel computation on cluster
- Immutable
- Improvisations of RDD (all retains basic features of RDD like immutability)
  - Dataframe
    - Set of data organized into named columns.
    - Similar to the table in a relational database
    - Allows processing of structured data
  - Dataset
    - Can contain any type of objects depending on language used



# Options in Spark APIs

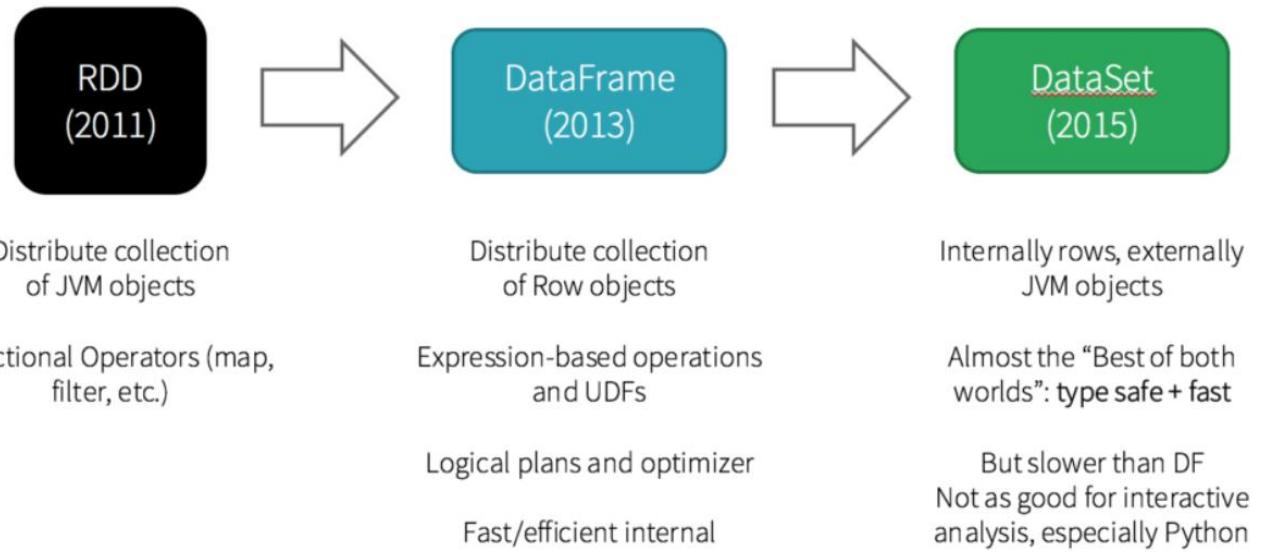
- **RDD useful when**

- Dealing with unstructured data
- Don't want to impose schema
- Low level operations on data
- Not interested in optimizations done for structured data



databricks

[RDD(Spark 1.0)] -> [Dataframe(Spark1.3)] -> [Dataset(Spark1.6)]



02-BasicRDDOperations.txt  
04-Wordcount1.txt

Named columns  
Fastest option

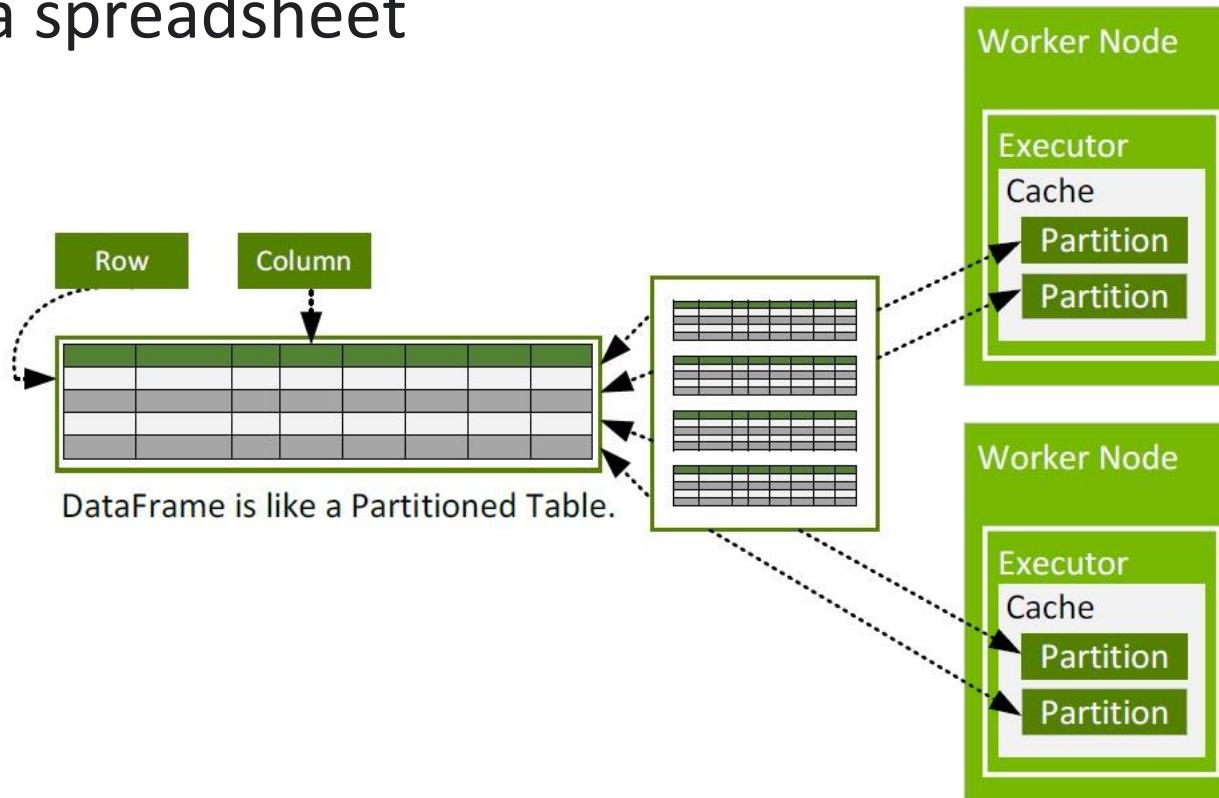
01-Spark-dataframes.txt

Added type-safety

01-Spark-datasets.txt

# DataFrames

A DataFrame is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet



# Datasets

Dataset clubs the features of RDD and DataFrame. It provides:

- The convenience of RDD.
- Performance optimization of DataFrame.
- Static type-safety of Scala
- Available only with Scala and Java

Datasets provides a more functional programming interface to work with structured data

```
# Create and display a dataset
```

```
case class Book(name: String, cost: Int)
```

```
val BookDS = Seq((Book("Scala",400)), (Book("Spark",500)), (Book("Kafka",300))).toDS()
```

```
BookDS.show()
```

```
# Word Count example
```

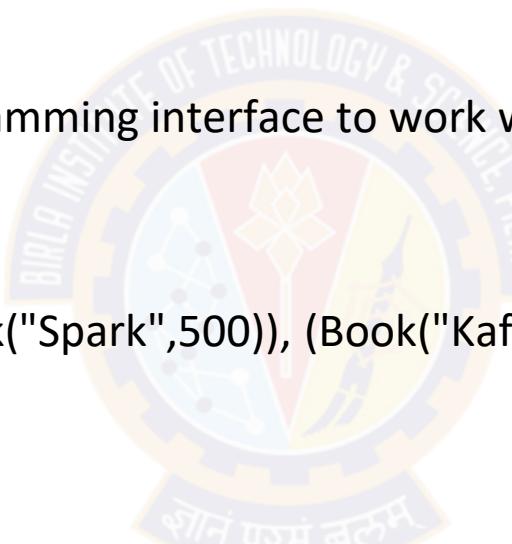
```
val linesDS = sc.parallelize(Seq("Spark is fast","Spark has Dataset","Spark Dataset is typesafe")).toDS()
```

```
val wordsDS=linesDS.flatMap(_.toLowerCase.split(" ")).filter(_ != "")
```

```
val groupedDS=wordsDS.groupBy("value")
```

```
val countsDS=groupedDS.count()
```

```
countsDS.show()
```



# Joins with Spark Datasets

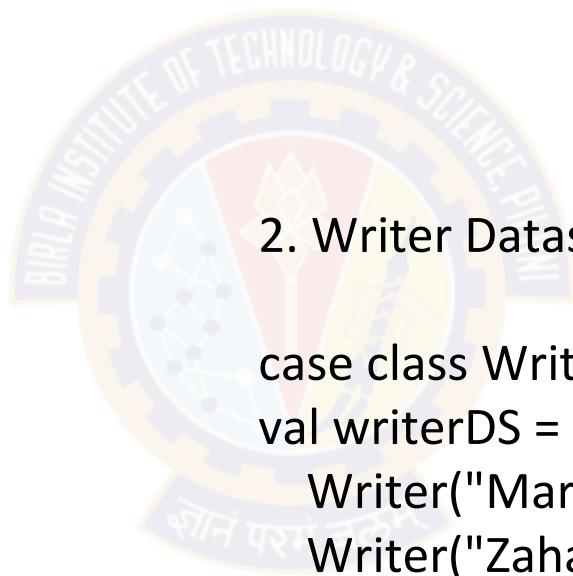
- Joining data is one of the most common and important operations for fulfilling business use cases.
- Spark Datasets supports all the fundamental types of joins.
  - ✓ INNER JOIN
  - ✓ LEFT JOIN
  - ✓ RIGHT JOIN
  - ✓ FULL OUTER JOIN
- While joining, we need to also consider performance as they may require large network transfers or even create datasets beyond the capability to handle



# Tables for Joining

## 1. Book Dataset

```
case class Book(book_name: String, cost: Int, writer_id:Int)
val bookDS = Seq(
    Book("Scala", 400, 1),
    Book("Spark", 500, 2),
    Book("Kafka", 300, 3),
    Book("Java", 350, 5)
).toDS()
bookDS.show()
```



## 2. Writer Dataset

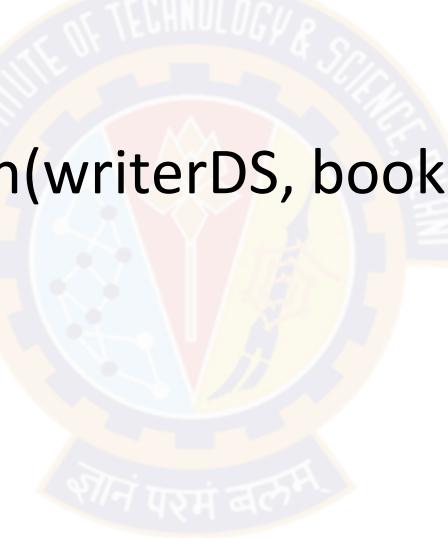
```
case class Writer(writer_name: String, writer_id:Int)
val writerDS = Seq(
    Writer("Martin",1),
    Writer("Zaharia ",2),
    Writer("Neha", 3),
    Writer("James", 4)
).toDS()
writerDS.show()
```

# Inner Join

- The INNER JOIN returns the dataset which has the rows that have matching values in both the datasets
- i.e. value of the common field will be the same.

```
val BookWriterInner = bookDS.join(writerDS, bookDS("writer_id") ===  
writerDS("writer_id"), "inner")
```

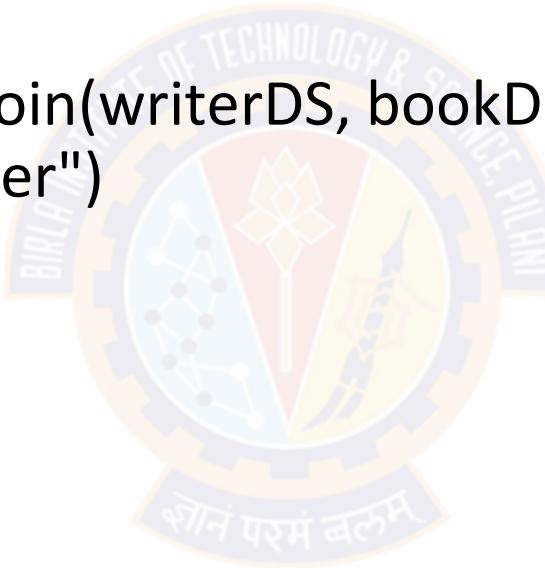
```
BookWriterInner.show()
```



# Left Join

- The LEFT JOIN returns the dataset that has all rows from the left dataset, and the matched rows from the right dataset.

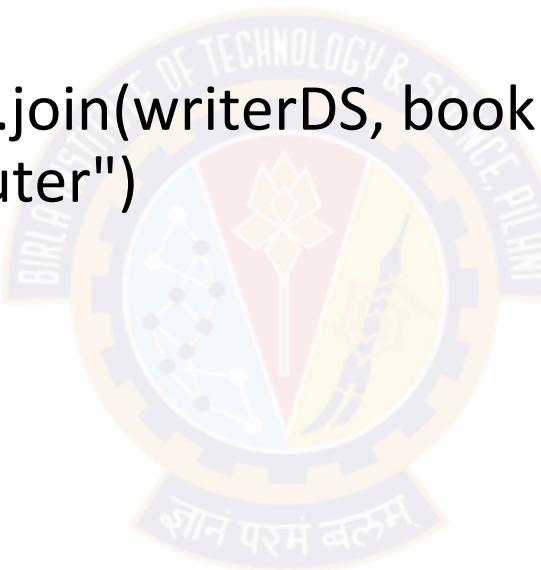
```
val BookWriterLeft = bookDS.join(writerDS, bookDS("writer_id") ===  
writerDS("writer_id"), "leftouter")  
BookWriterLeft.show()
```



# Right Join

- The RIGHT JOIN returns the dataset that has all rows from the right dataset, and the matched rows from the left dataset.

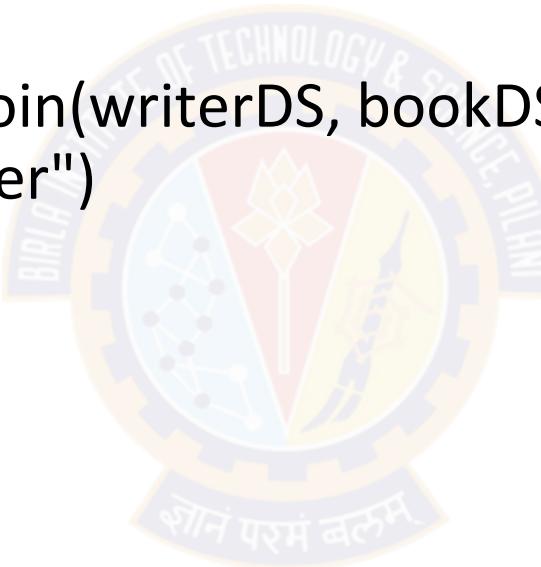
```
val BookWriterRight = bookDS.join(writerDS, bookDS("writer_id") ===  
writerDS("writer_id"), "rightouter")  
BookWriterRight.show()
```



# Full Outer Join

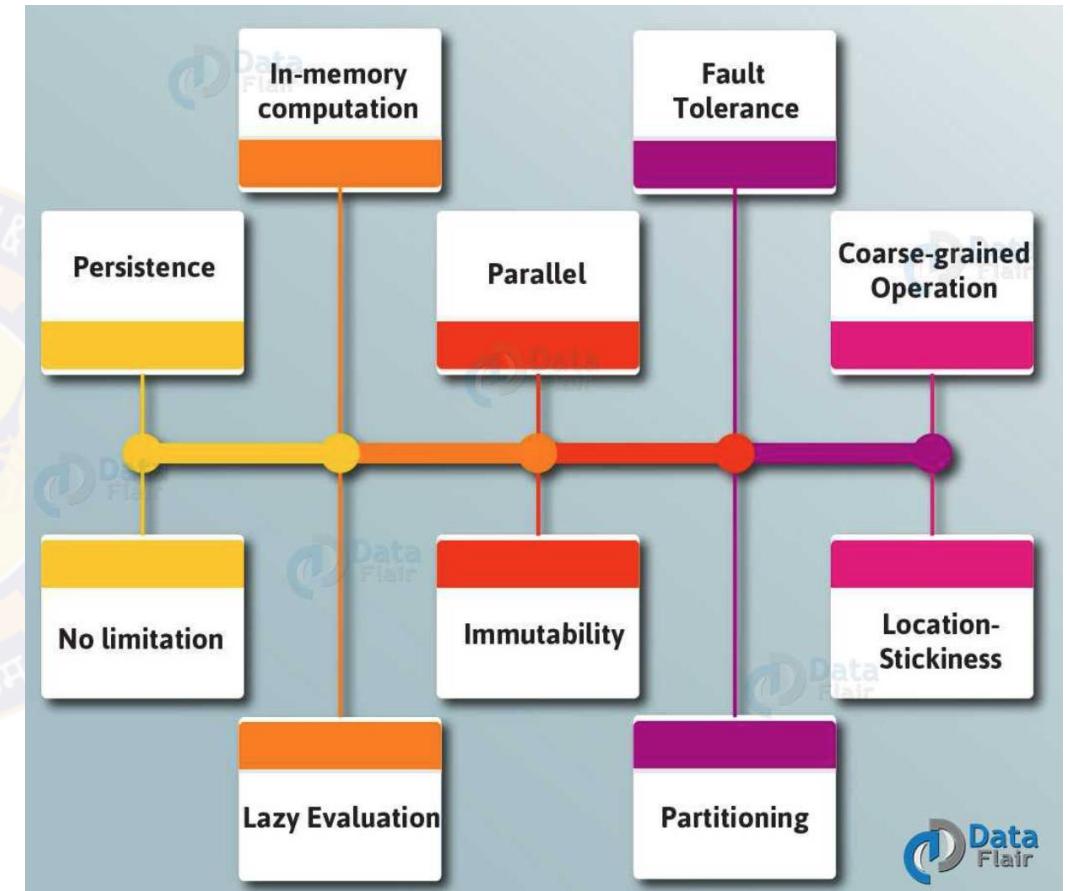
- The FULL OUTER JOIN returns the dataset that has all rows when there is a match in either the left or right dataset.

```
val BookWriterFull = bookDS.join(writerDS, bookDS("writer_id") ===  
writerDS("writer_id"), "fullouter")  
BookWriterFull.show()
```



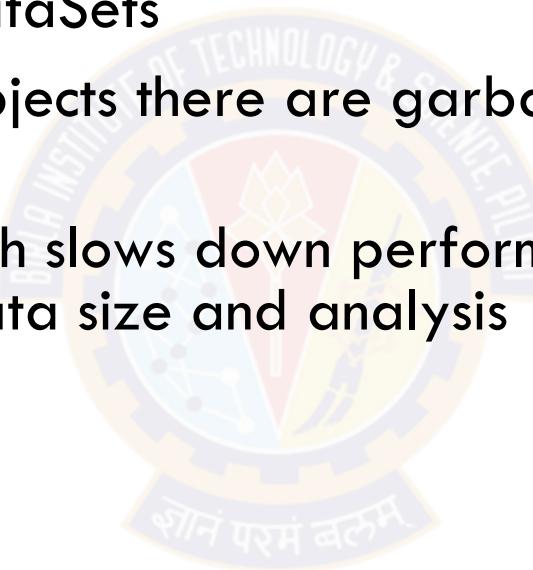
# RDD features

- Keep data in memory as much as possible
- Evaluate only when an action triggers
- A failed lost RDD partition on a worker can be recovered from lineage of operations
- Cannot change once created - Immutable
- Persist in memory or storage for reuse
- Parallelism through partitioning
- Put tasks closer to data location
- Apply operations to entire set of data at coarse grain and not one data item within RDD



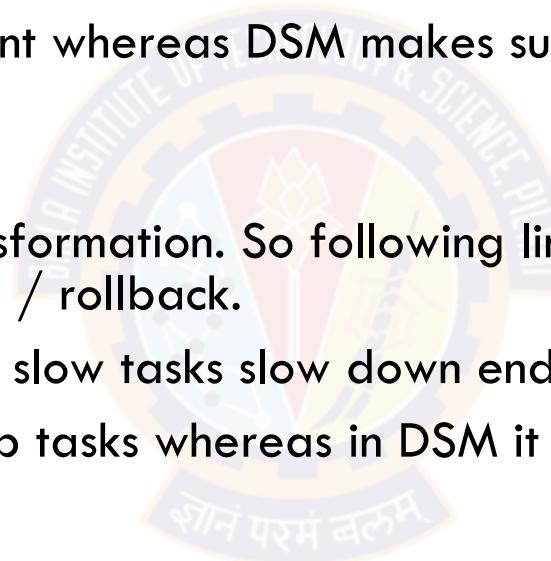
# RDD limitations

- For structured data
  - RDDs do not exploit any optimizers
  - Better to use DataFrame or DataSets
- Since RDDs are in-memory JVM objects there are garbage collection and Java serialization overheads
- Spill over data is put on disks which slows down performance, hence machines need to have enough memory given the data size and analysis

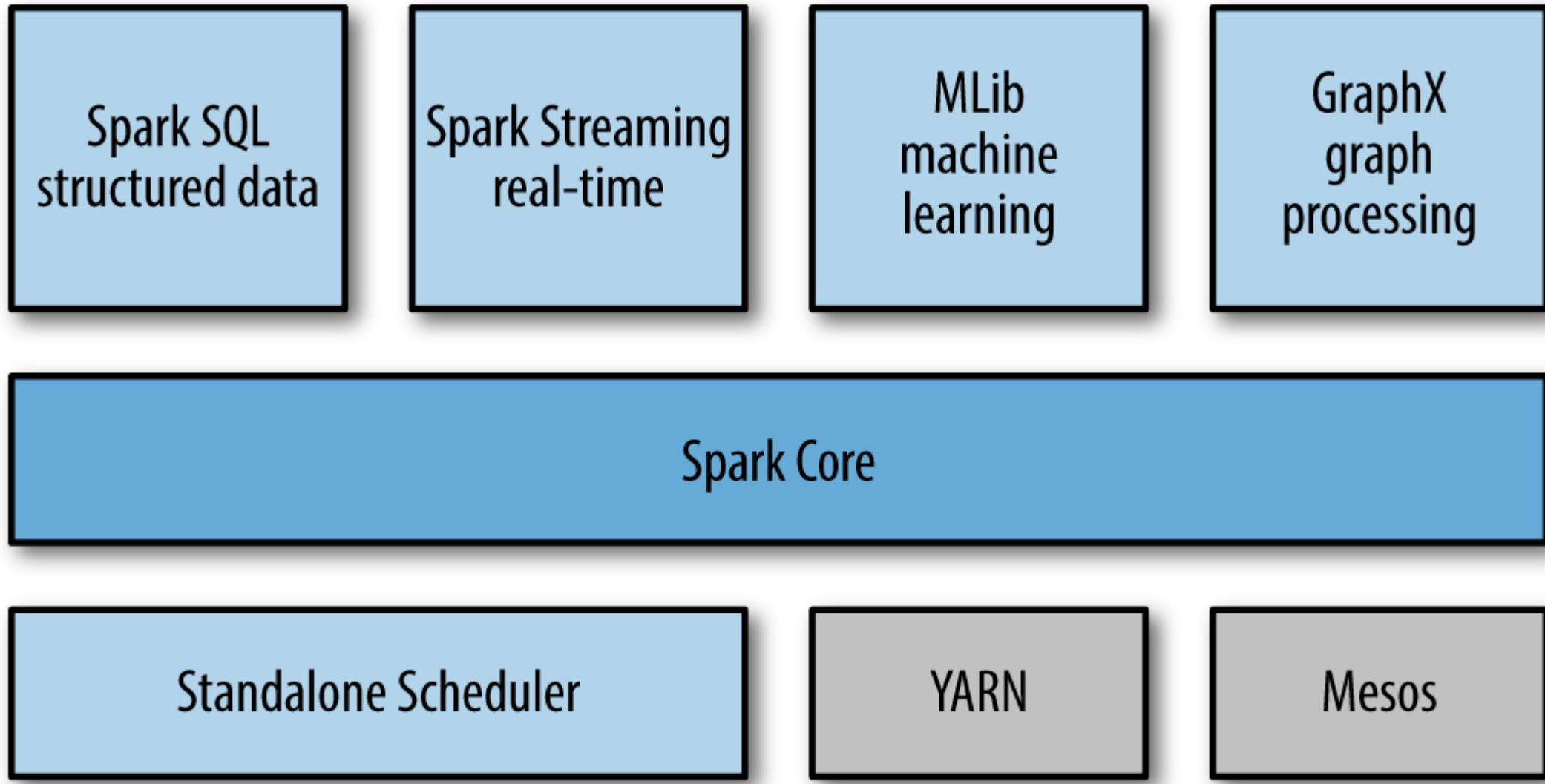


# RDD and Distributed Shared Memory (DSM)

- Grain of R/W operation
  - RDD is coarse grained as it works at dataset level whereas DSM is at specific data item level
- Consistency
  - Immutable RDDs are trivially consistent whereas DSM makes sure of consistency if programmer follows a set of rules
- Fault recovery
  - New RDDs are created on each transformation. So following lineage of operations RDDs can be recovered. DSMs need checkpointing / rollback.
- Straggler mitigation: Problem of having slow tasks slow down end to end performance
  - RDDs make it little easier with backup tasks whereas in DSM it is difficult
- Out-of-memory behaviour
  - Spill over data goes to on-disk RDDs gradually degrading performance whereas in DSM system swaps may lead to instability

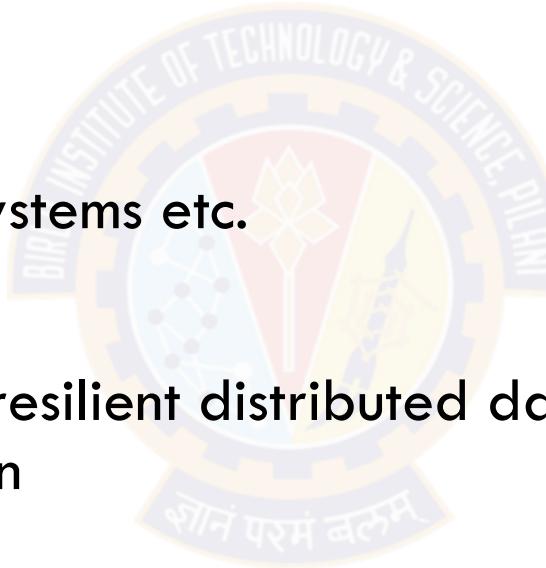


# Spark Unified Stack



# Spark Core

- Contains the basic functionality of Spark, including components for
  - ✓ task scheduling
  - ✓ memory management
  - ✓ fault recovery
  - ✓ interacting with storage systems etc.
- Home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction
- Provides many APIs for building and manipulating these collections

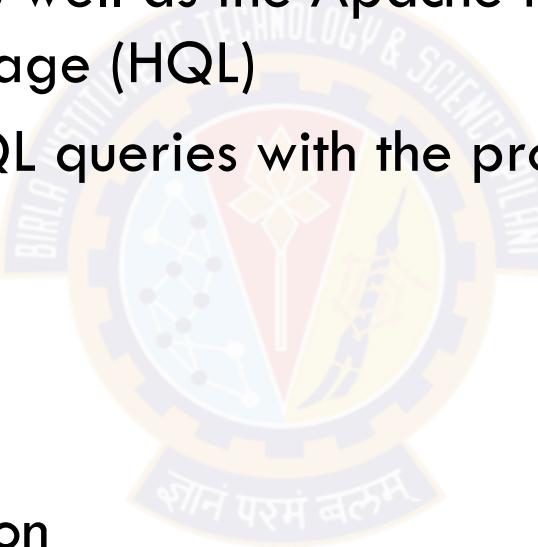


# Spark SQL



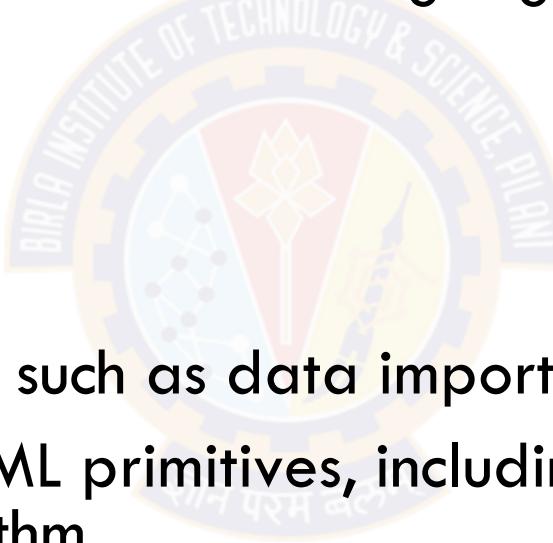
- Spark's package for working with structured data
- Allows querying data via SQL as well as the Apache Hive variant of SQL
  - ✓ called the Hive Query Language (HQL)
- Allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in
  - ✓ Python
  - ✓ Java
  - ✓ Scala

all within a single application
- Tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool



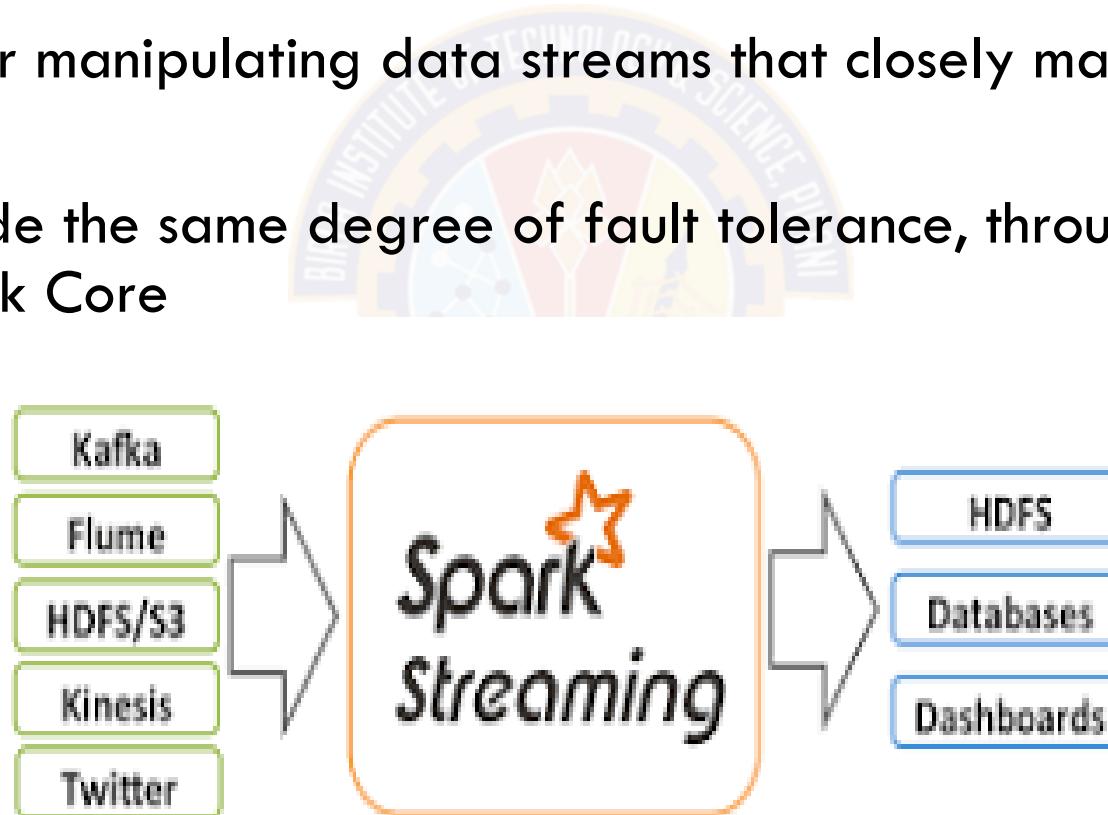
# MLlib

- Built in library containing common machine learning (ML) functionality
- Provides multiple types of machine learning algorithms, including
  - ✓ Classification
  - ✓ Regression
  - ✓ Clustering
  - ✓ Collaborative filtering
  - ✓ Supporting functionality such as data import and model evaluation
- Provides some lower-level ML primitives, including a generic gradient descent optimization algorithm
- All of these methods are designed to scale out across a cluster



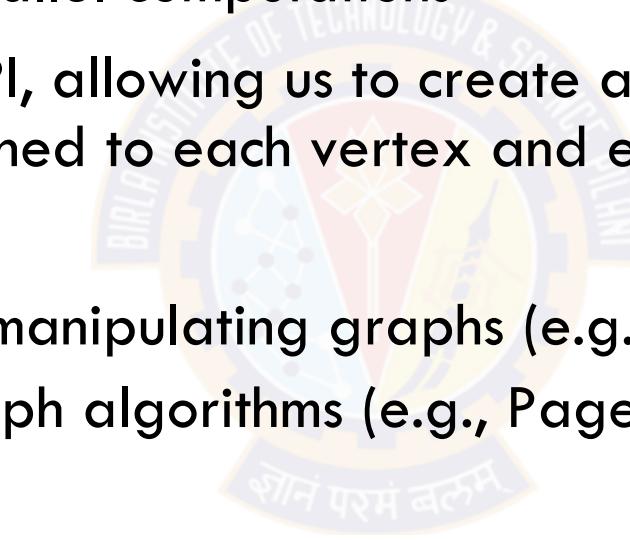
# Spark Streaming

- Spark component that enables processing of live streams of data
- Examples of data streams include web servers log files
- Provides an API for manipulating data streams that closely matches the Spark Core's RDD API
- Designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core

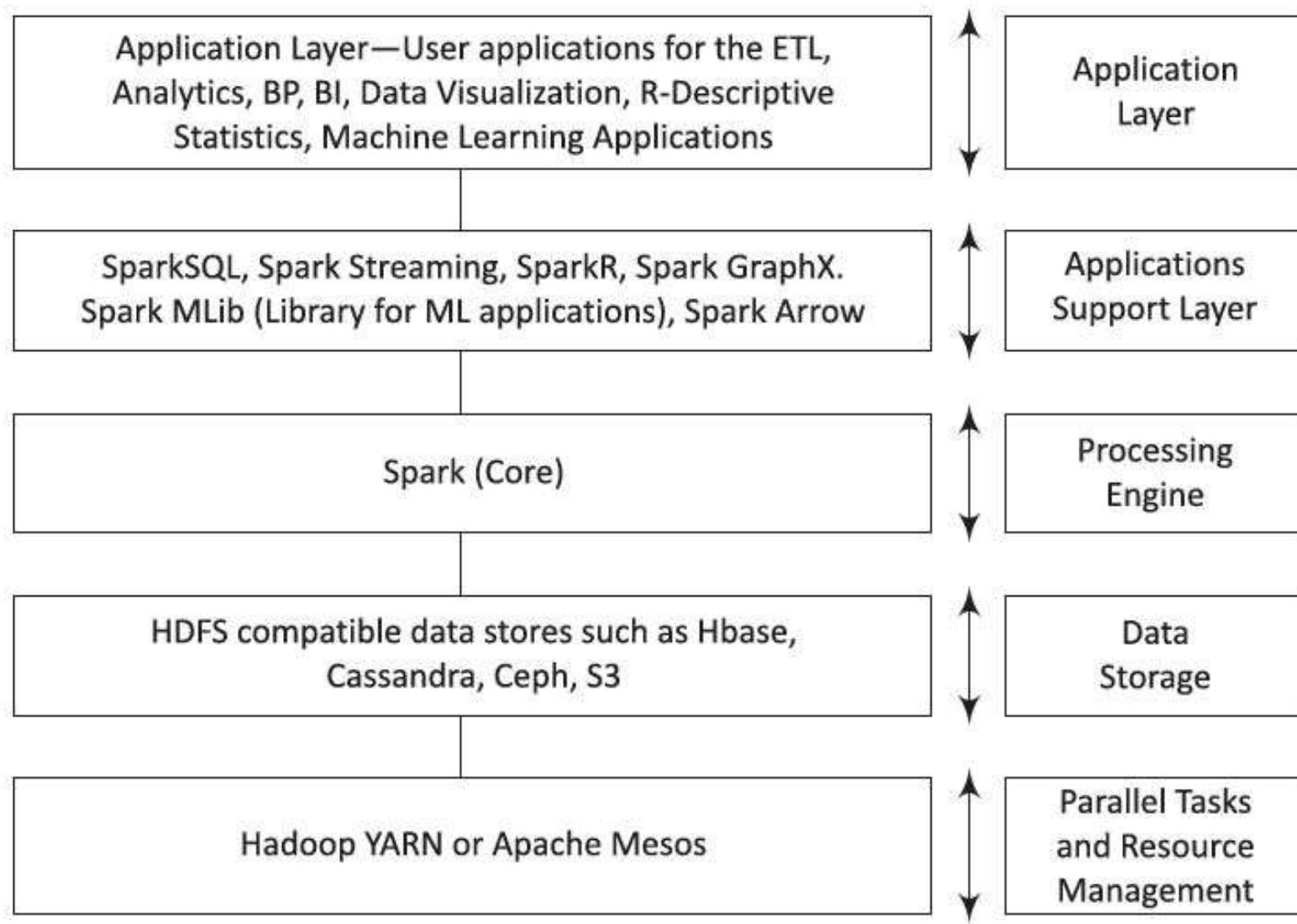


# GraphX

- Library for
  - ✓ manipulating graphs (e.g. a social network's relations graph)
  - ✓ performing graph-parallel computations
- Extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge
- Provides
  - ✓ various operators for manipulating graphs (e.g., subgraph and mapVertices)
  - ✓ library of common graph algorithms (e.g., PageRank and triangle counting)



# 5 Layer Architecture for running Spark Applications



# GPU Acceleration for CPU intensive Spark jobs

## NVIDIA RAPIDS Accelerator

- Operates as a software plugin to popular Apache Spark platforms
- Automatically accelerates supported operations
- Requires no code changes
- Operations currently accelerated
  - ✓ Spark SQL
  - ✓ DataFrame
- Works with Spark standalone, YARN clusters, Kubernetes clusters
- Gives 5x Fast execution



# Cluster Managers

- Spark is designed to efficiently scale up from one to many thousands of compute nodes
- Can run over a variety of cluster managers, including
  - ✓ Hadoop YARN
  - ✓ Apache Mesos
  - ✓ Simple cluster manager included in Spark itself called the Standalone Scheduler
  - ✓ Kubernetes
- If you are just installing Spark on an empty set of machines
  - ✓ the Standalone Scheduler provides an easy way to get started
- If you already have a Hadoop YARN or Mesos cluster,
  - ✓ Spark's support for these cluster managers allows your applications to also run on them

- Launch Overheads of Spark Applications on Standalone and Hadoop YARN Clusters  
[https://link.springer.com/chapter/10.1007/978-981-15-5558-9\\_5](https://link.springer.com/chapter/10.1007/978-981-15-5558-9_5)

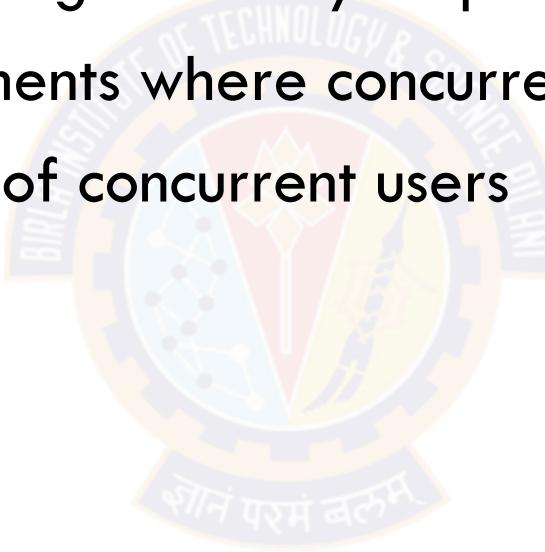


# Use cases for Spark

- Banking
  - Customer segmentation, credit risk assessment, targeted advertisement on products
- e-commerce
  - Clustering on streaming data for identifying trends and providing recommendations
- Travel
  - Provide personalised hotel recommendations, restaurant reservations, e.g. TripAdvisor
- Media and entertainment
  - Automatic game complexity level setting by mining patterns in real-time, advertisement using real-time analysis often combined with data from MongoDB, e.g. Pinterest, Yahoo
- Healthcare
  - Patient record analysis with past clinical data to proactively avoid hospitalisation, genomic sequencing
- Fog computing / IoT
  - Sensor data analysis on the edge

## When not to use Spark

- Large batch processes with high memory requirements
- Multi-user analysis environments where concurrent demand for memory is high
- May not scale with number of concurrent users



# Topics for today

- Introduction
- Getting started
  - ✓ Setup
- RDDs
  - ✓ Transformations
  - ✓ Actions
  - ✓ Programming
- Sample programs



# Sample programs

- 03-Spark-Scala-Reverse.txt
- 04-Wordcount2.txt
- 04-WordCount3.py



# Summary

- Introduction to basic concepts
  - In-memory for speed
  - RDD basics, limitations
  - General purpose big data computing covering multiple use cases
- Sample programs to get started
- Additional Reading
  - Learning Spark - Lightning-Fast Data Analysis - O'Reilly
  - Apache Spark documentation



**Next Session:  
Spark - Part 2**



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 11-1: Graph Databases

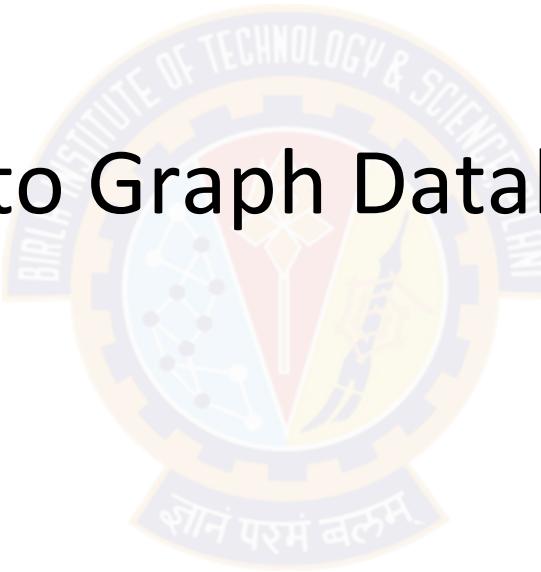
---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

## Introduction to Graph Databases



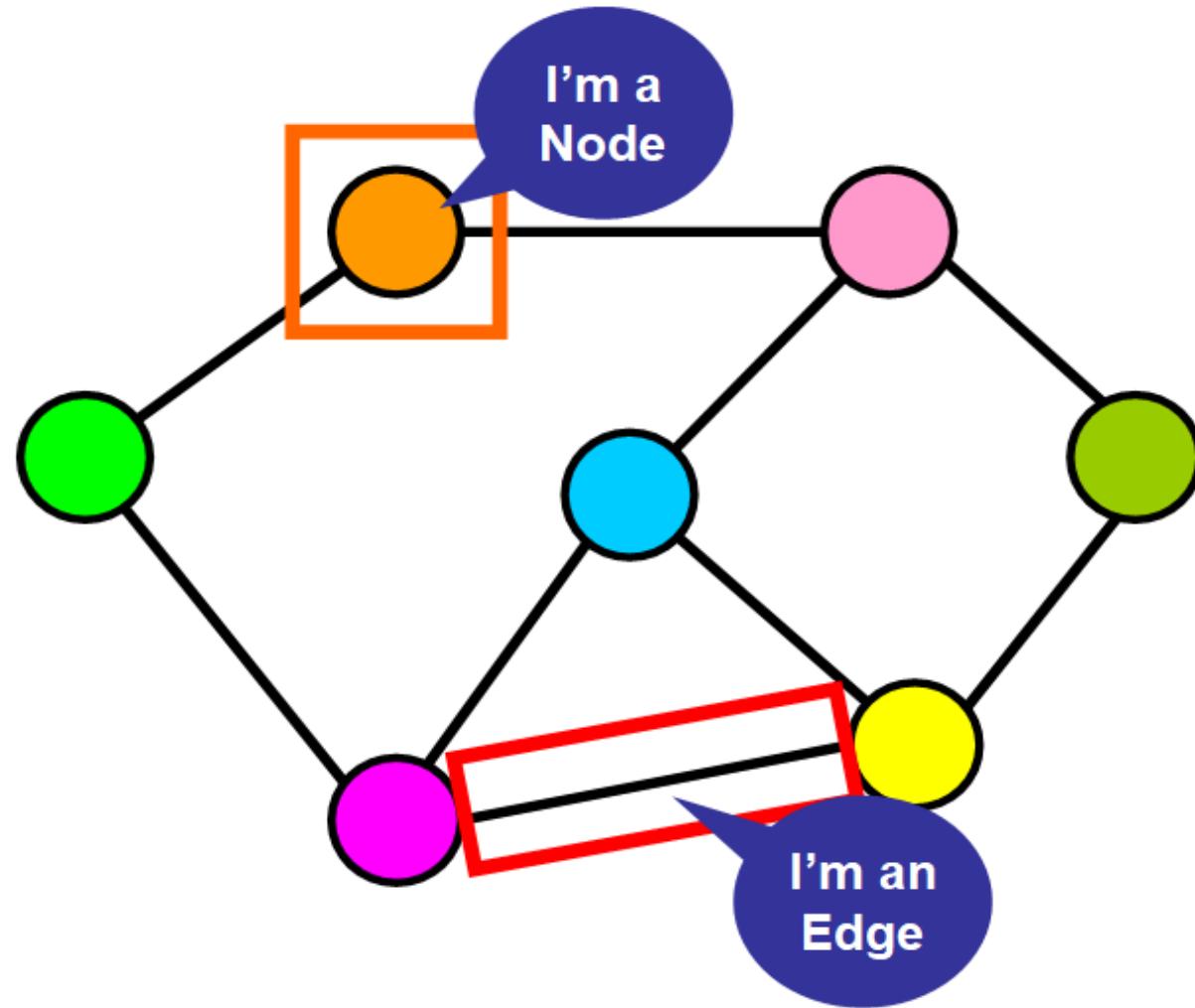
# 5 Signs You Need a Graph Database

Is your traditional database not up to the task of deciphering your complex data? Here are five of the telltale signs:

1. Hard-to-navigate interconnected data
2. Lack of real-time insights
3. Inflexible data structures
4. Inefficient querying
5. Inability to decode a web of highly connected data



# Graphs



# Graph Theory



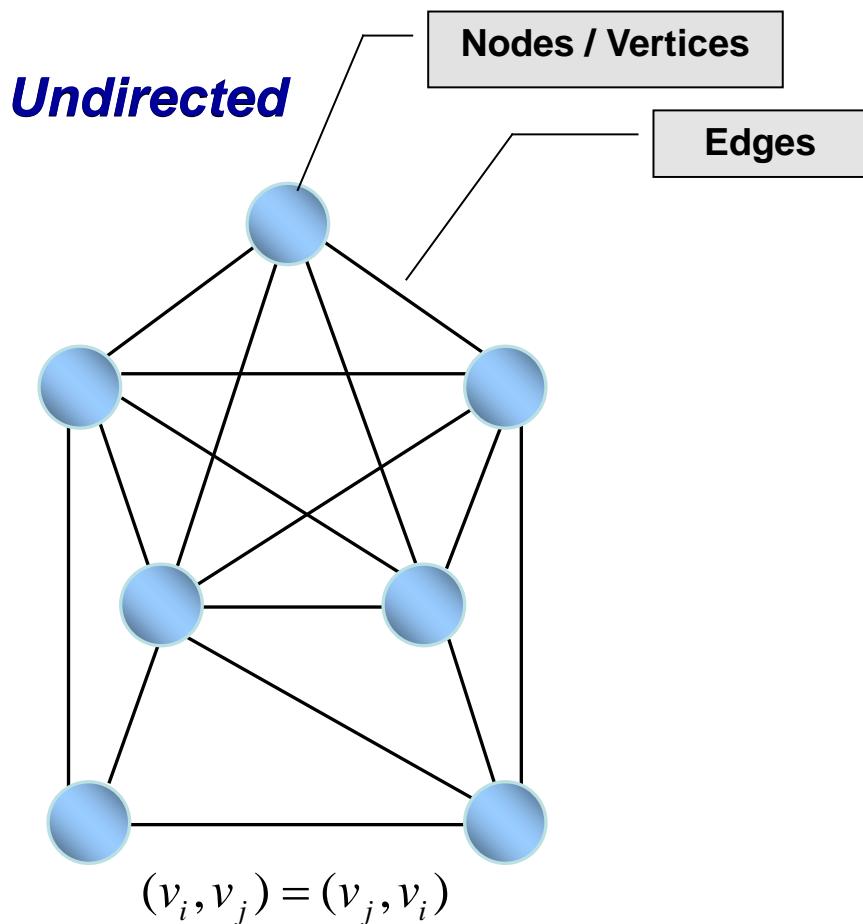
## Leonhard Euler

invented Graph Theory  
in **1736**,

**275 years** before  
Edgar Codd formulated  
the relational model

# Graphs - Mathematically

Graph with 7 nodes and 16 edges

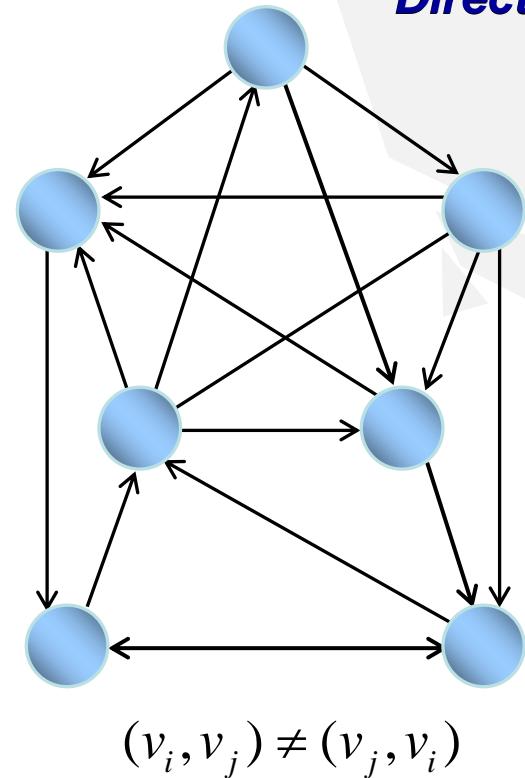


$$G = (V, E)$$

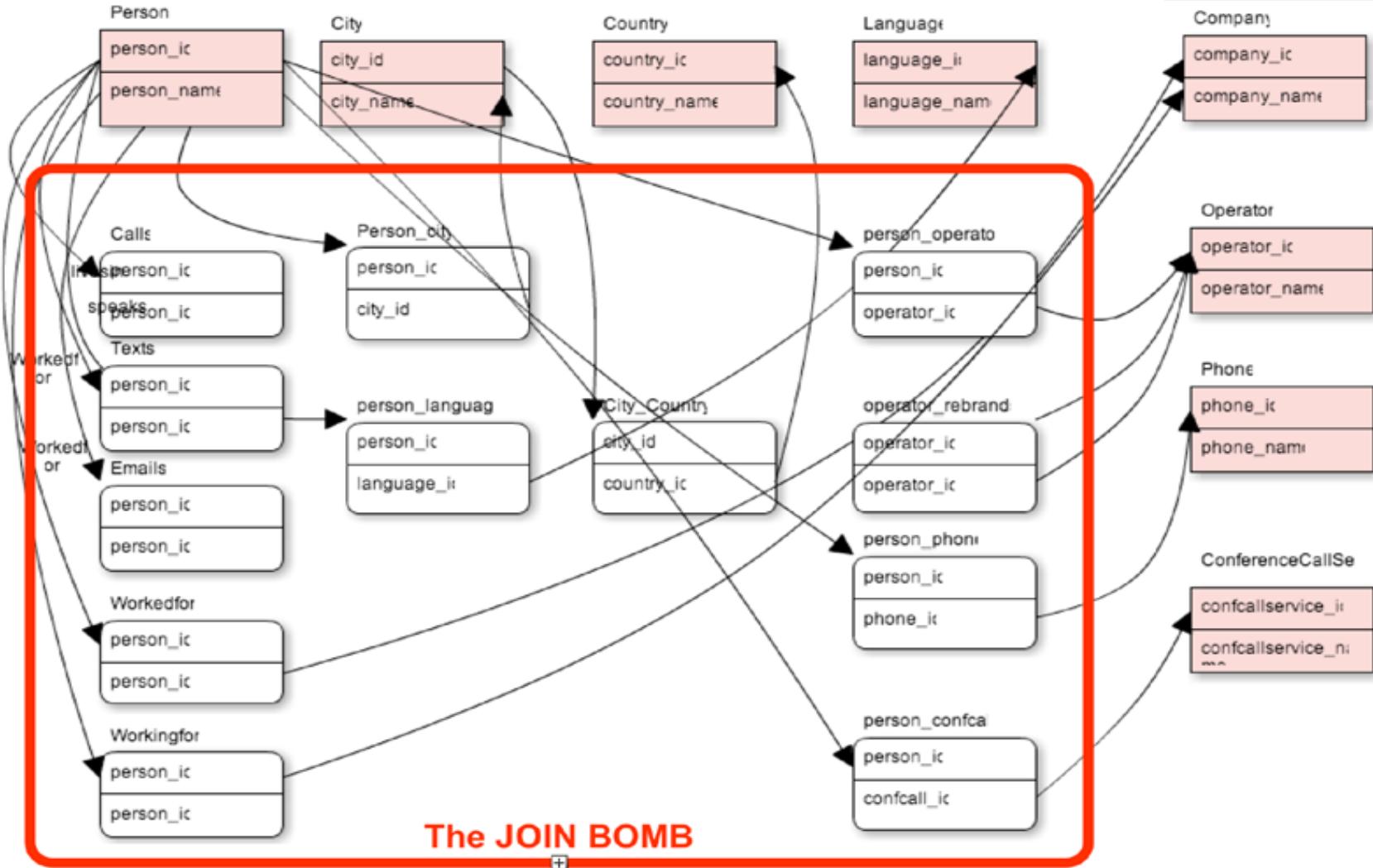
$$V = \{v_1, v_2, \dots, v_n\}$$

$$E = \{e_k = (v_i, v_j) \mid v_i, v_j \in V, k = 1, \dots, m\}$$

**Directed**



# Relational Database is Anti-Relational

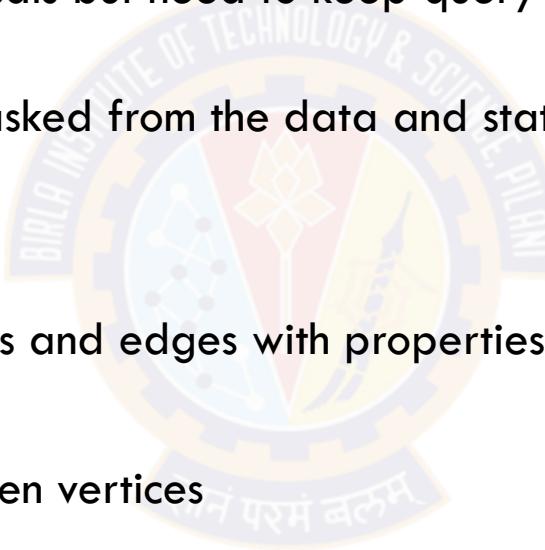


# What is Graph Database

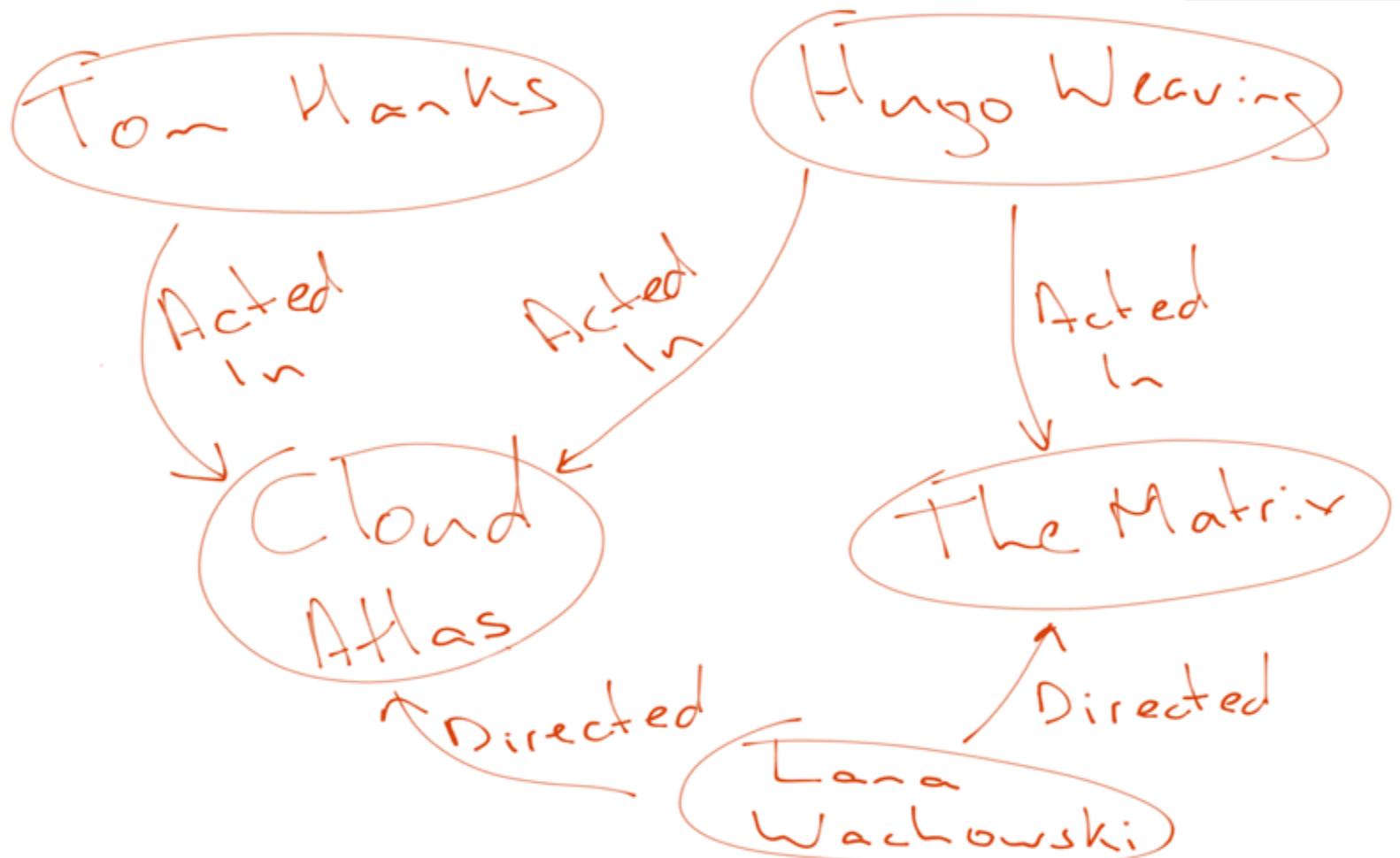
- A database built around graph structure – Nodes & Edges
- Each Node has a set of incoming and outgoing edges
- Relationships / Edges connect nodes
- Each Node / Edge is uniquely identified
- Each Node/Edge has a collection of properties
- Properties store named data values
- Each edge has a label that defines the relationship between its ending nodes
- Can store any kind of data
- No fixed number of schema or properties
- Nodes are indexed for fast initial lookup
- **Strong mathematics (graph theory) behind**

# Graph computing

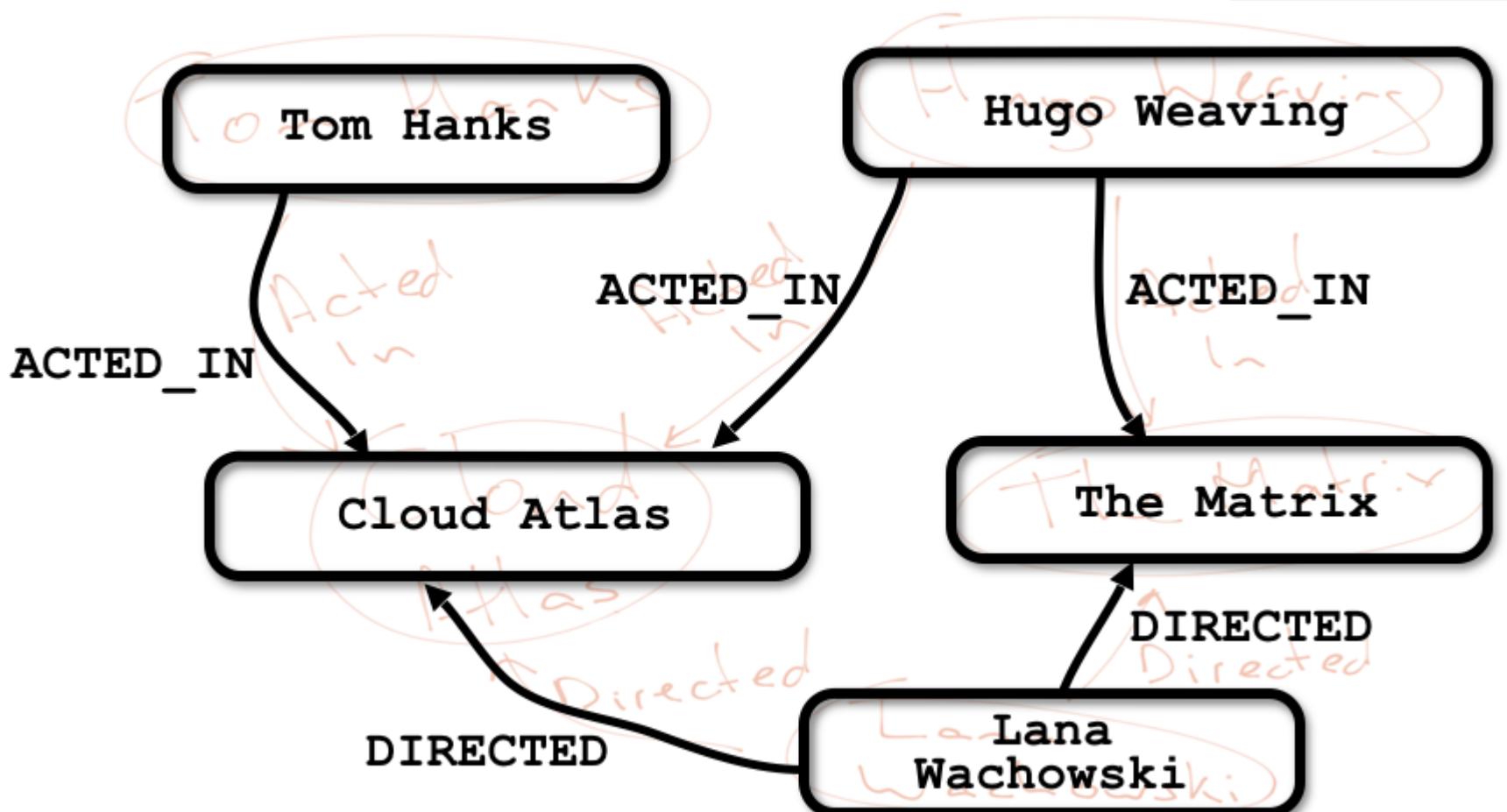
- When to use a graph DB ?
  - A relationship-heavy data set with large set of data items
  - Queries are like graph traversals but need to keep query performance almost constant as database grows
  - A variety of queries may be asked from the data and static indices on data will not work
- Property graphs
  - Data is represented as vertices and edges with properties
  - Properties are key value pairs
  - Edges are relationships between vertices



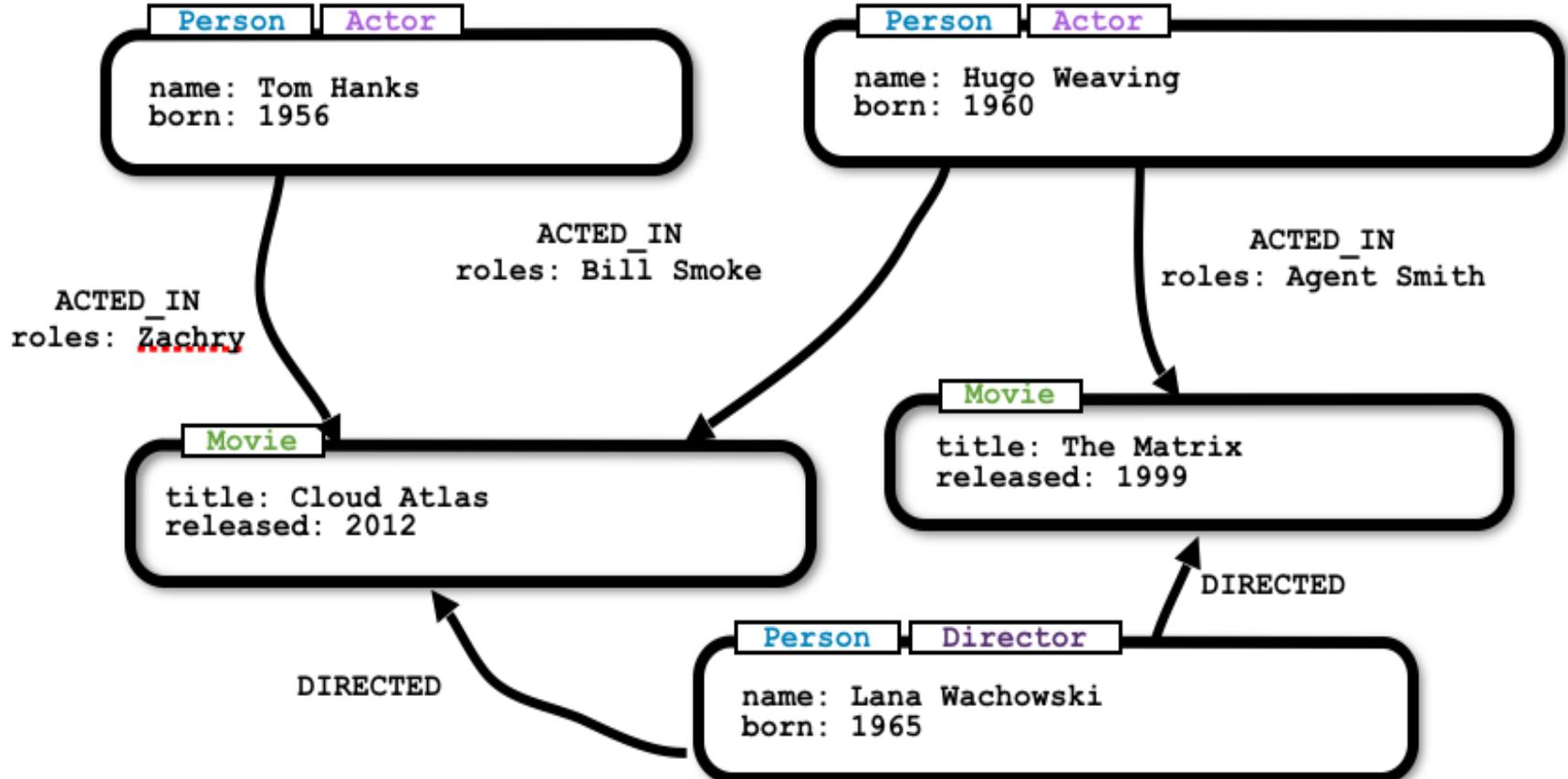
# Graph Data Modelling – Whiteboard model



# Nodes & Relationships

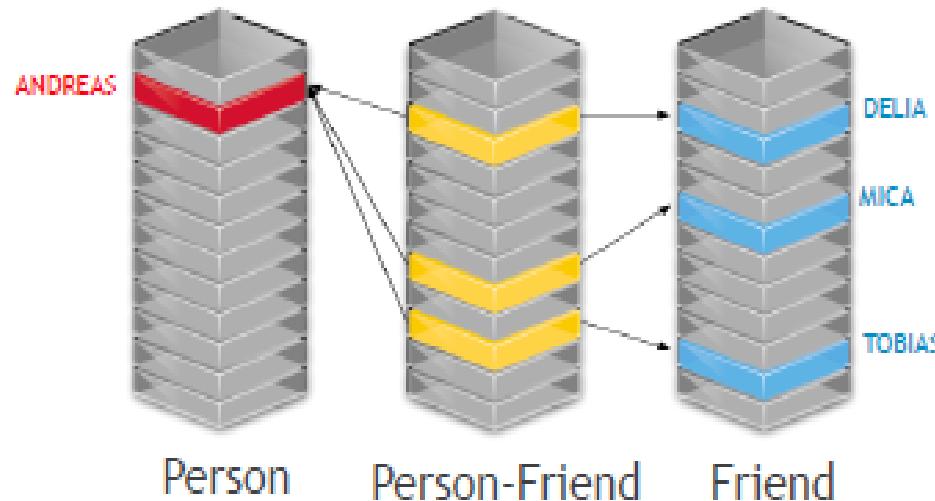


# Add labels and properties

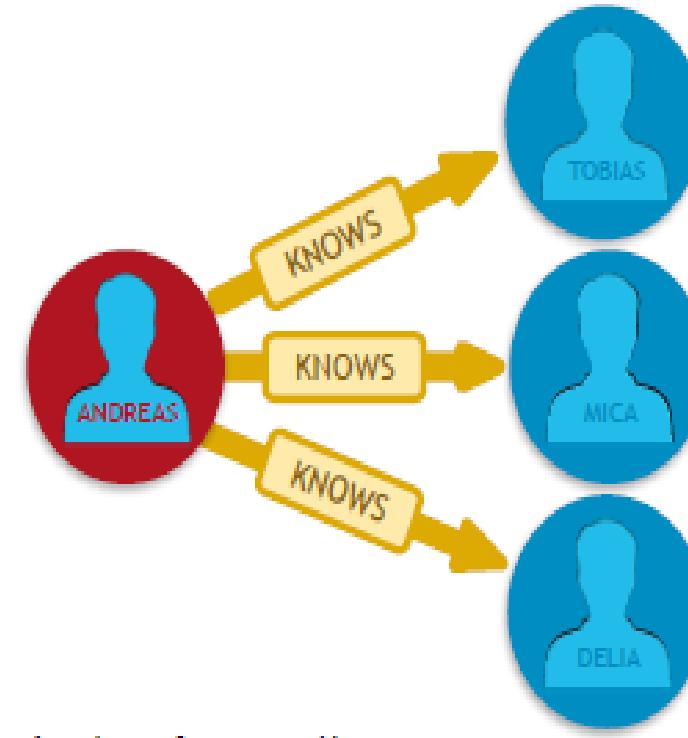


# Relational Vs Graph Models

Relational Model



Graph Model



Index free adjacency

# Nodes

- Stores data in a Graph, with records called Nodes
- The simplest graph has just a single node with some named values called Properties
- Data is stored as Properties
- Properties are simple name/value pairs
- Schema-free, nodes can have a mix of common and unique properties
- Nodes are usually indicated by Nouns
- **Nodes are the name for data records in a graph (row in RDBMS)**



# Labels

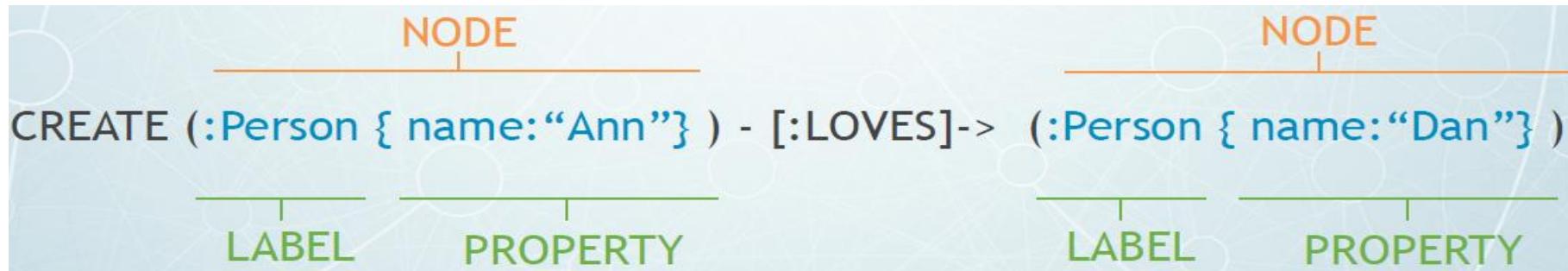
- Labels are tags that associate a set of nodes.
- Nodes can be grouped together by applying a Label to each member. In social graph, we'll label each node that represents a Person.
- Apply the label "Person" to the node we created for Emil
- A node can have zero or more labels
- Labels do not have any properties
- **Label name maps to Table Name in RDBMS**

# More data , More nodes

- Like any database, storing data in graph database can be as simple as adding more records
- Graph databases (Neo4j) can store billions of nodes
- Similar nodes can have different properties
- Node Properties can be
  - Strings
  - Arrays of Strings
  - Numbers
  - Boolean values
  - Byte[]
  - Date
- Node properties are usually adjectives

# Relationships

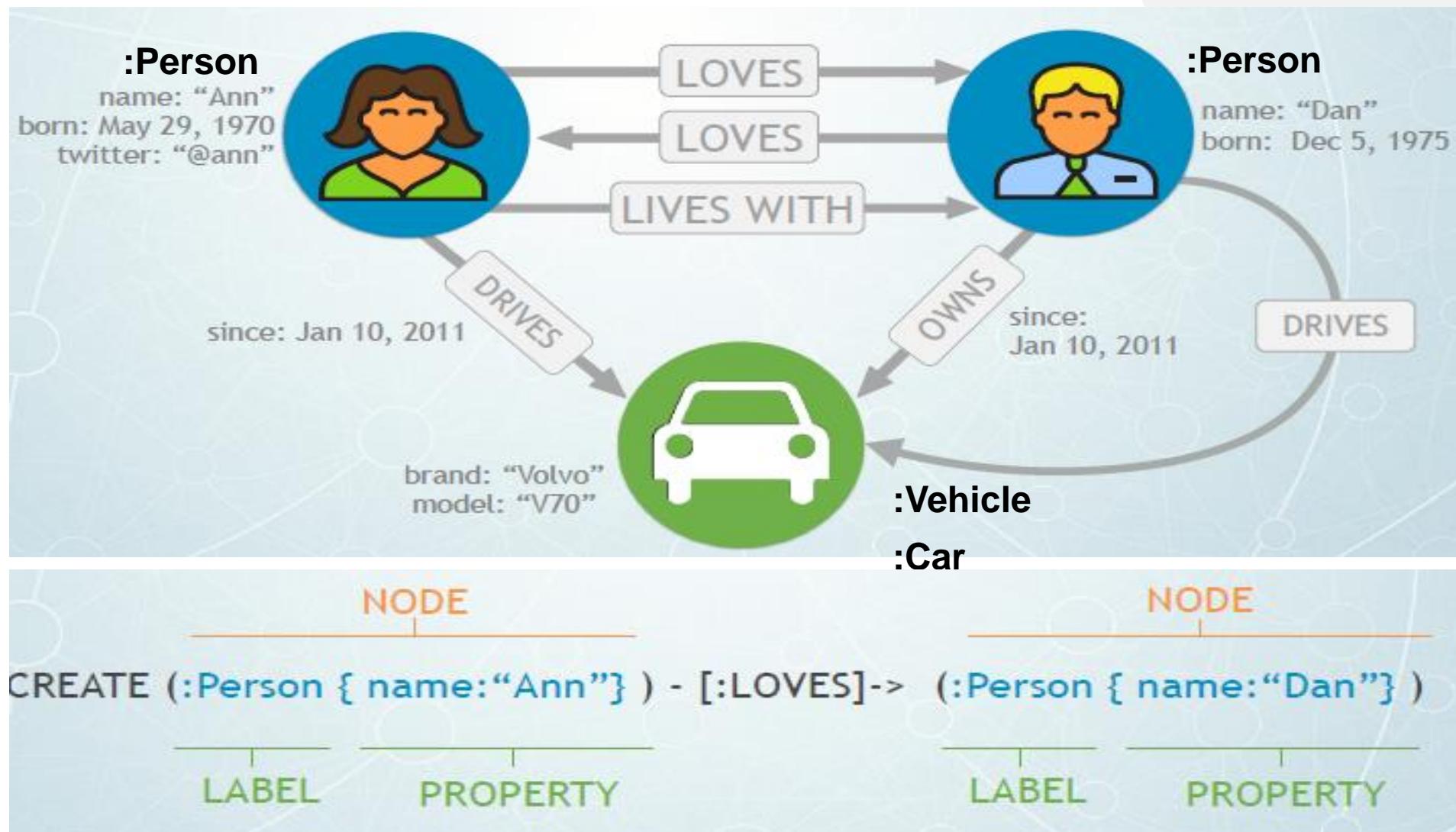
- The real power of Graph database is in connected data
- To associate any two nodes, add a Relationship which describes how the records are related.
- Relationships always have direction
- Relationships always have an identifier
- Relationships form patterns of data
- Relationships are usually represented by Verbs



# Relationship properties

- Relationships have properties , they are also data records
- Relationship properties:
- Store information shared by two nodes
- They are usually adverbs
- Examples:
  - Emil has known Johan since 2001
  - Emil rates Ian 5 (out of 5)

# Property Graph



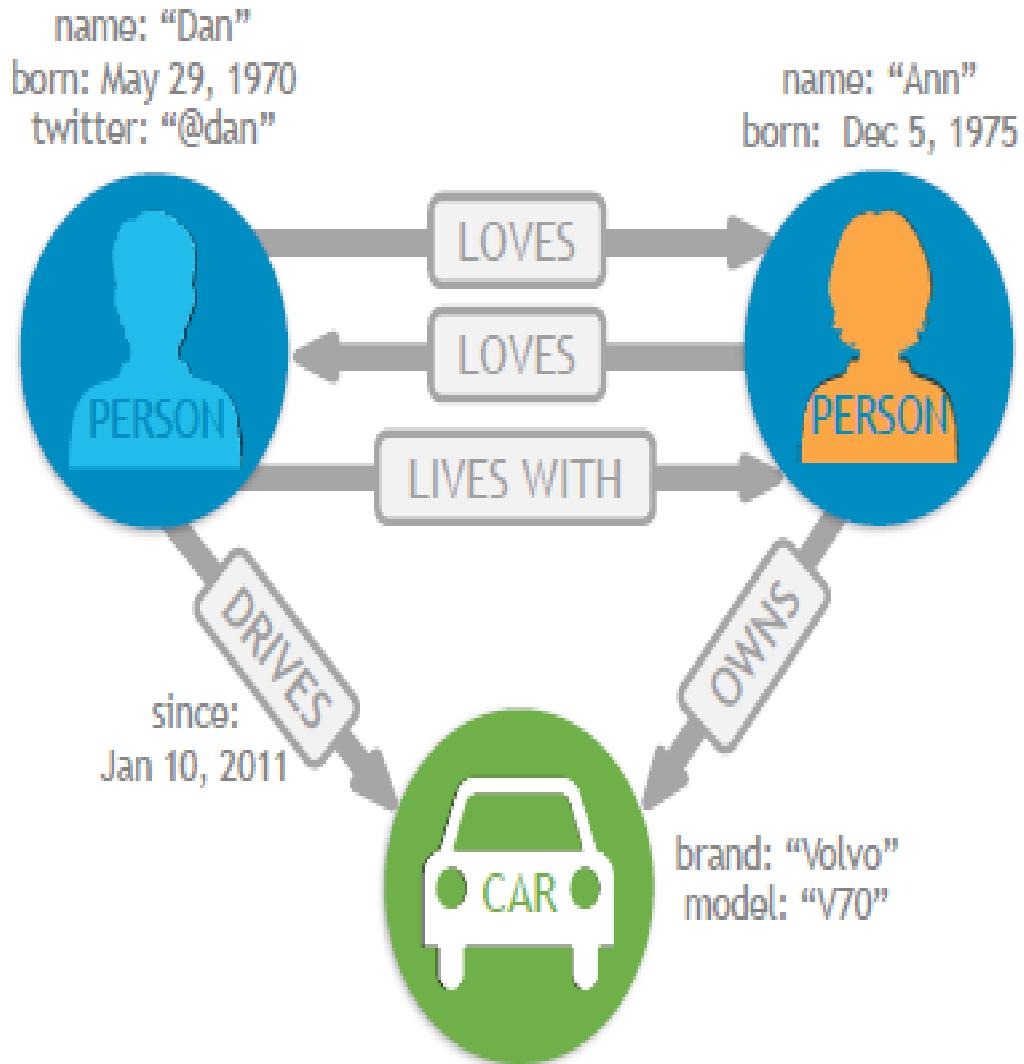
# Property Graph Model Components

## Nodes

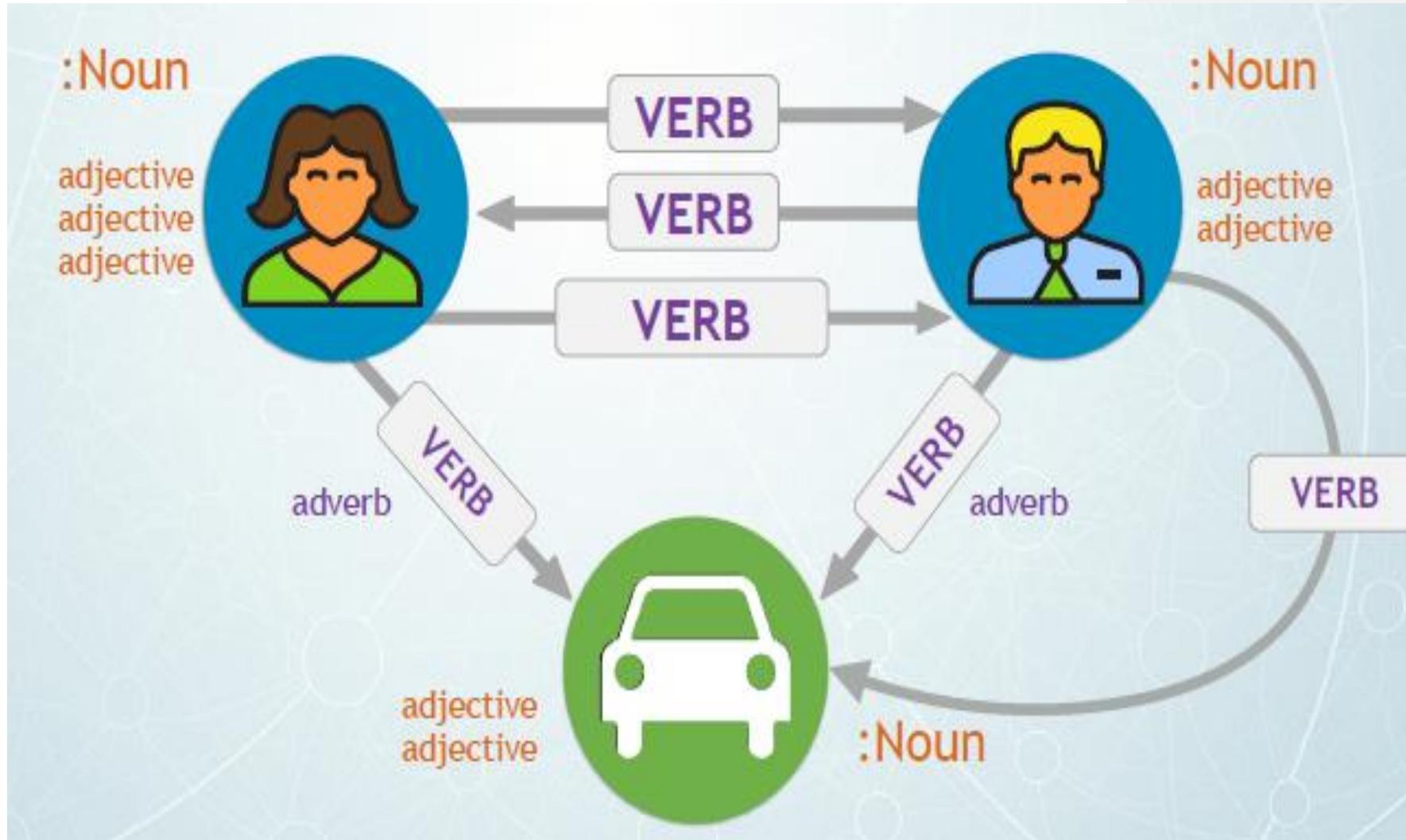
- The objects in the graph
- Can have name-value *properties*
- Can be *labeled*

## Relationships

- Relate nodes by type and direction
- Can have name-value *properties*



# Mapping Properties To Languages context



# Graph Data Base Space

## Amazon Neptune

Fast, reliable graph database built for the cloud



ArchiGraph



FlockDB



Titan



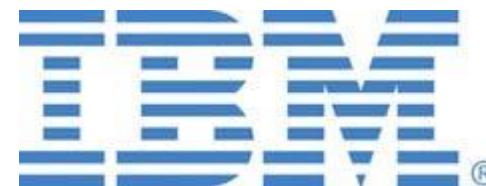
GraphBase



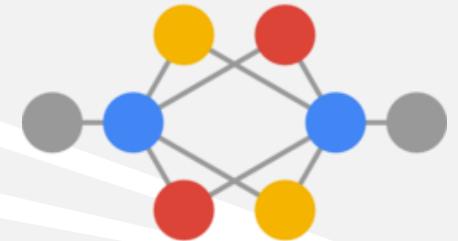
neo4j



HyperGraphDB

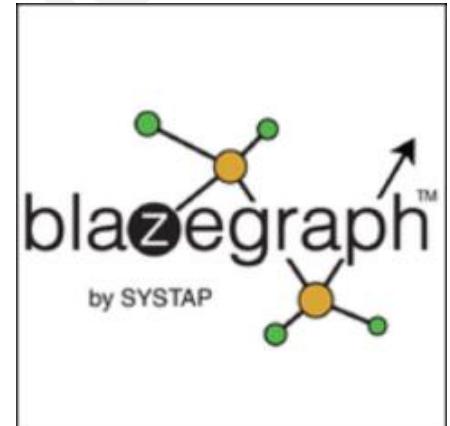


IBM Graph



Cayley

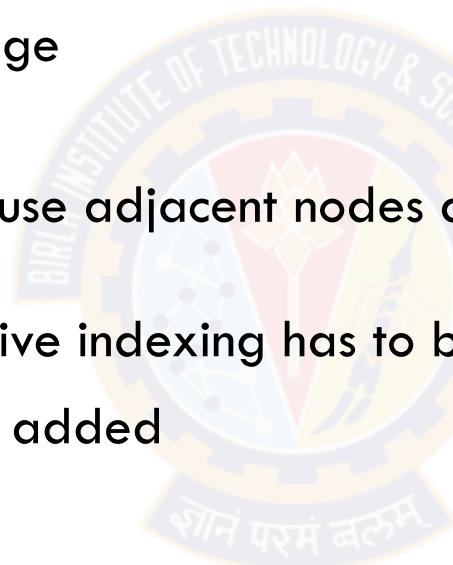
ANZOGRAPH® DB



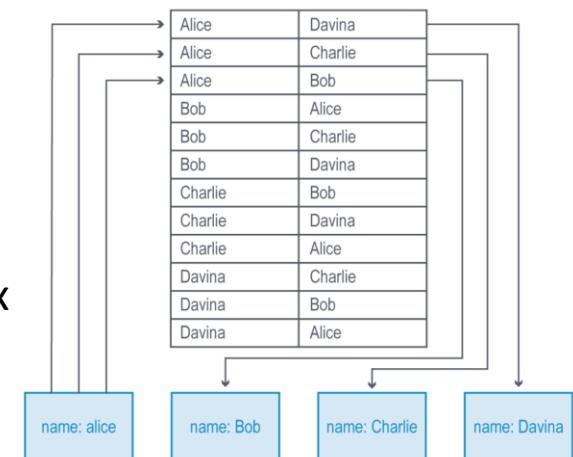
by SYSTAP

# Native vs Non-Native Graph storage

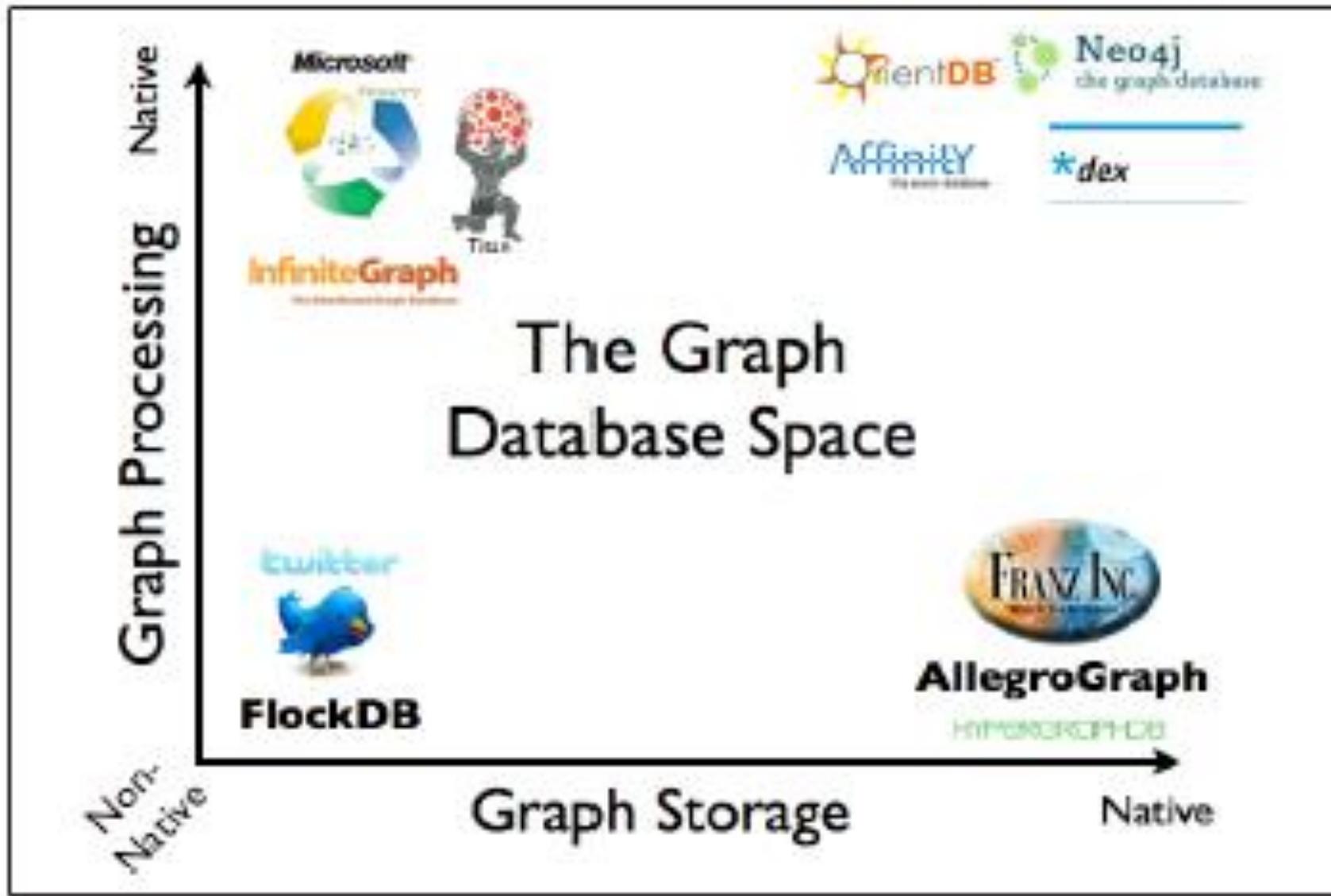
- Non-native graph computing platforms can use external DBs for data storage
  - e.g. TinkerPop is an in-memory DB + computing framework that can store in ElasticSearch, Cassandra etc.
- Native platform support built-in storage
  - e.g. Neo4j
- Native approach is much faster because adjacent nodes and edges are stored closer for faster traversal
  - In a non-native approach, extensive indexing has to be used
- Native approach scales as nodes get added



One-hop index

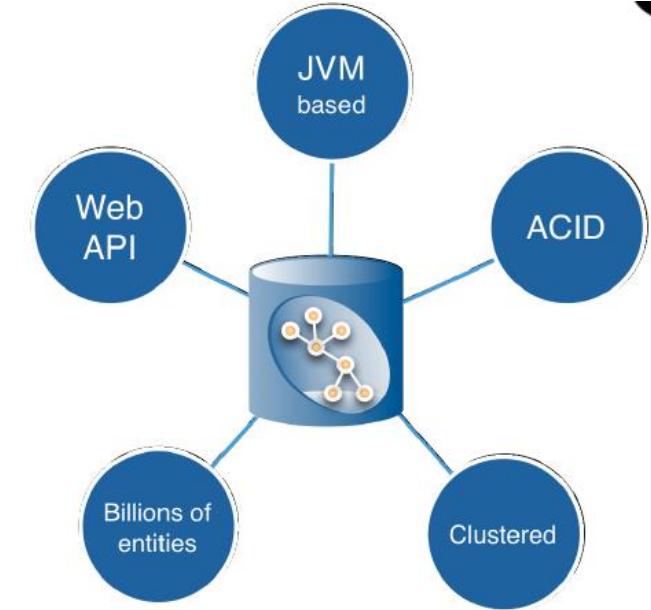


# Graphs for Storage and Processing



# What is Neo4J

- It's is a Graph Database supporting full ACID Transactions
- Embeddable in applications and server deployable
- Java based, Open sourced
- Schema free, bottom-up data model design
- Neo4j is stable
- In 24/7 operation since 2003
- Neo4j is under active development
- High performance graph operations
- Supports the **Cypher** query language
- Traverses 1,000,000+ relationships/sec on commodity hardware
- No. of nodes and relationships decide Volume of data

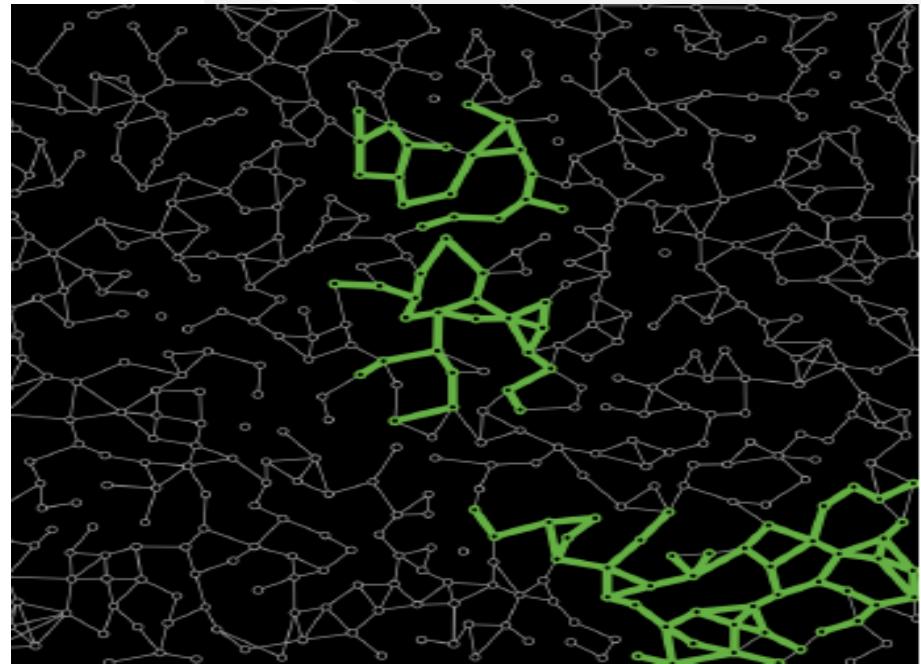


# neo4j – Key product features

- Native Graph Storage - Ensures data consistency and performance
- Native Graph Processing - Millions of hops per second, in real time
- Whiteboard Friendly Data Modelling - Model data as it naturally occurs
- High Data Integrity - Fully ACID transactions
- Powerful Expressive Query Language - Requires 10x to 100x less code than SQL
- Scalability and High Availability -Vertical & Horizontal scaling optimized for graphs
- Built-in ETL - Seamless import from other databases
- Integration - Drivers and APIs for all popular languages

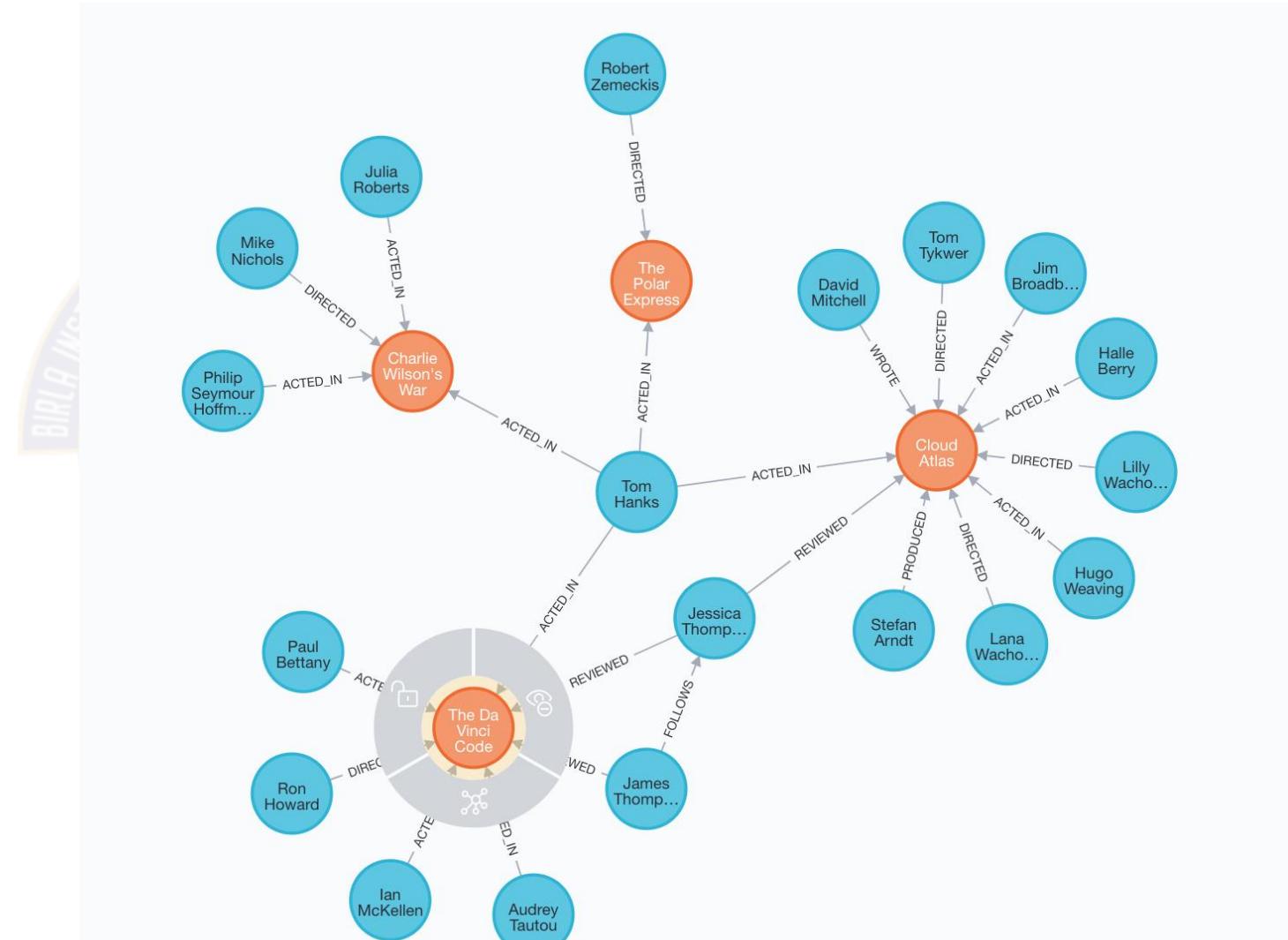
# Querying a Graph

- Nodes are indexed using Apache Lucene - Scalable high-performance indexing
- Nodes can be indexed by properties also
- Find one or more start nodes - Anchor nodes
- Explore surrounding graph starting from the anchor
- Millions of hops per second using index free adjacency



# Neo4j / Cypher

- Cypher is a Declarative language for graph query
- Example: `match (:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie) where m.released > 2000 RETURN m limit 5`



Launch a free sandbox with dataset on [neo4j website](#)

# Neo4j / Cypher: More queries

- Find movies that Tom Hanks acted in and directed by Ron Howard released after 2000
  - Match (:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m) where m.released > 2000 RETURN m limit 5
- Who were the other actors in the movie where Tom Hanks acted in and directed by Ron Howard released after 2000
  - Match (:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m), (p:Person)-[:ACTED\_IN]->(m) where m.released > 2000 RETURN p limit 5

# Neo4j editions

**Community Edition** - is a full functioning version of Neo4j suitable for single instance deployments. It has full support for key Neo4j features, such as ACID compliance, Cypher and access via the binary protocol and HTTP APIs. It is ideal for smaller internal or do-it-yourself projects that do not require high levels of scaling or professional services and support.

**Enterprise Edition** - extends the functionality of the Community Edition to include key features for performance and scalability such as a clustering architecture for High Availability and online backup functionality. It is the right choice for production systems with availability requirements or needs for scaling up or out.

Subscription based licensing:

- USD 10,000 per core - license cost per year
- 3 Machines / 8 Cores each - 20,000 per year ( Total 24 cores)

# Neo4j on Cloud

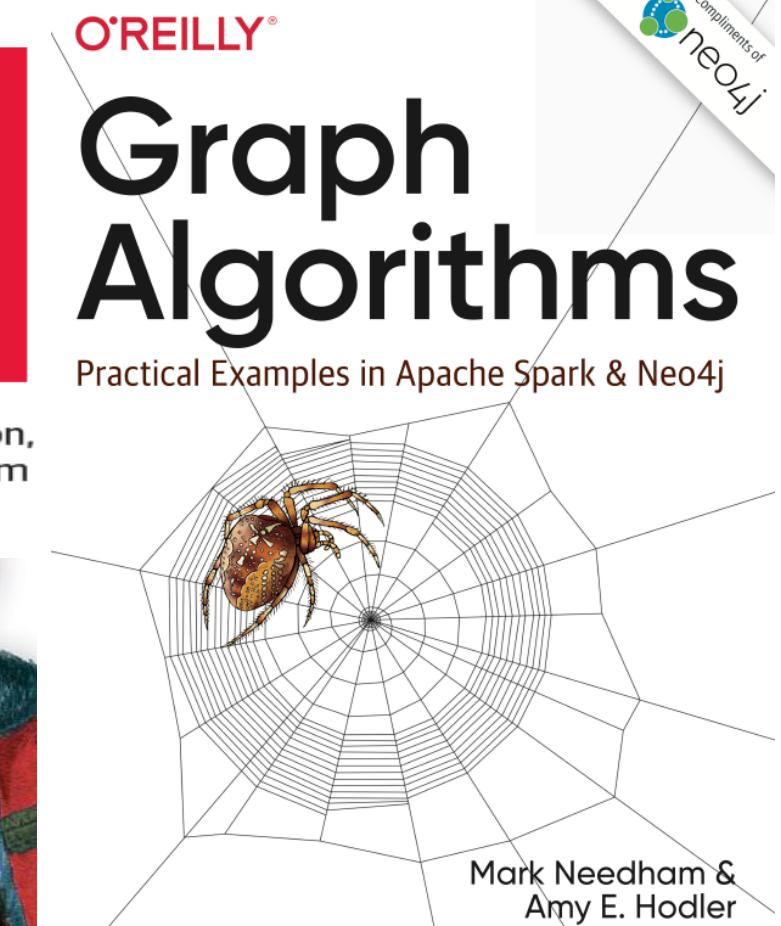
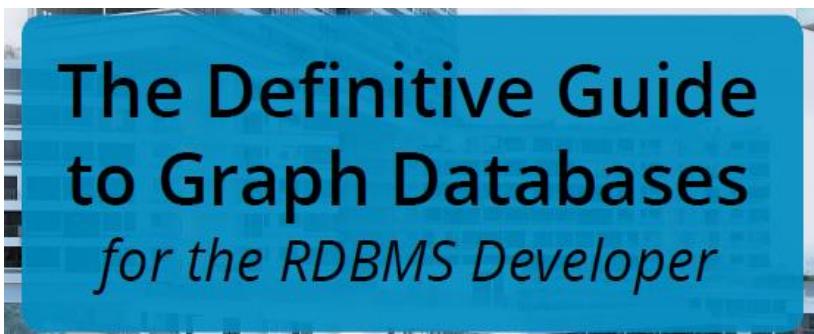
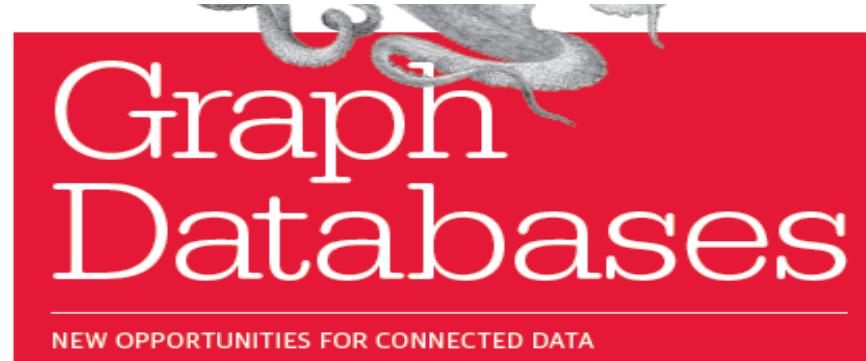
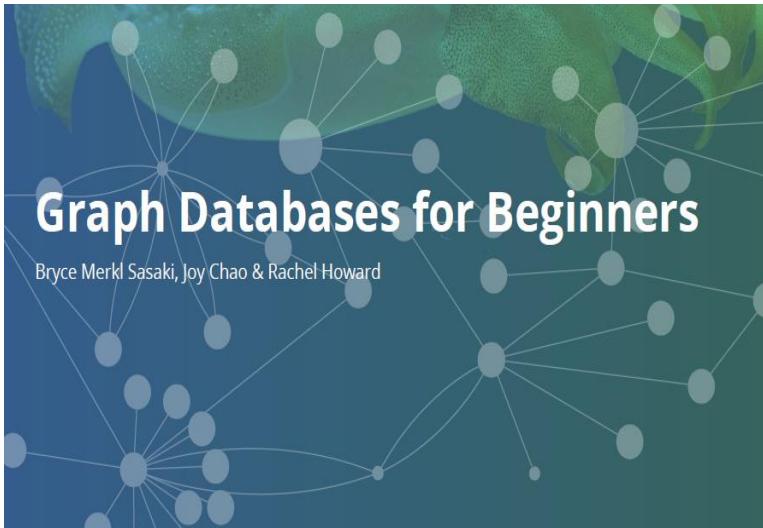
- Neo4j Enterprise Edition on AWS
- Neo4j - The Easiest Way to Graph on MS Azure
- NEO4J AURA on Google Cloud



# Webcasts /Books on Graph Database

<https://www.infoq.com/presentations/neo4j-graph-theory>

<https://www.infoq.com/presentations/graph-db-tools>



# Neo4j Training / Certification

Neo4j Training

neo4j Graphacademy - <https://graphacademy.neo4j.com/>

[Free, Self-Paced, Hands-on Online Training | Free Neo4j Courses from GraphAcademy](#)



# Spark GraphX

GraphX is Apache Spark's API for graphs and graph-parallel computation.

<https://spark.apache.org/graphx/>

## Features:

- Flexibility
  - ✓ GraphX unifies ETL, exploratory analysis, and iterative graph computation within a single system. You can view the same data as both graphs and collections, transform and join graphs with RDDs efficiently, and write custom iterative graph algorithms
- Speed
  - ✓ Comparable performance to the fastest specialized graph processing systems.
  - ✓ GraphX competes on performance with the fastest graph systems while retaining Spark's flexibility, fault tolerance, and ease of use.
- Algorithms
  - ✓ Choose from a growing library of graph algorithms.
  - ✓ In addition to a highly flexible API, GraphX comes with a variety of graph algorithms, many of which were contributed by our users

# Apache Tinkerpop / Gremlin

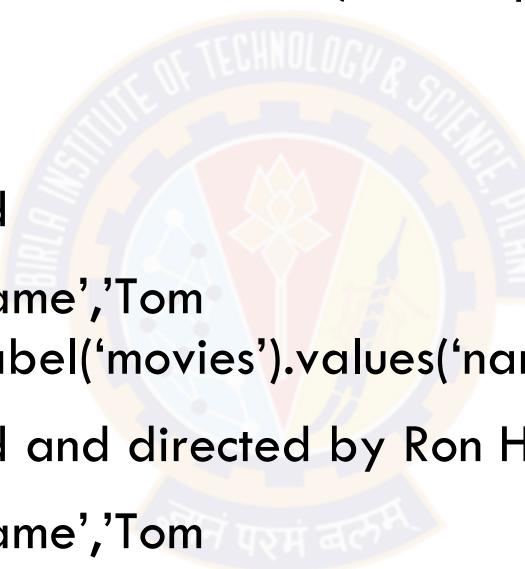
- TinkerPop is a computing platform that connects to GraphDBs that actually store the nodes and edges. Built-in TinkerGraph stores in-memory data only.
- Gremlin is the query language (with traversal machine) that supports Declarative and Imperative flavours
- Sample queries

✓ movies where Tom Hanks has acted

- `g.V().hasLabel('person').has('name','Tom Hanks').outE('ACTED_IN').hasLabel('movies').values('name')`

✓ movies where Tom Hanks has acted and directed by Ron Howard

- `g.V().hasLabel('person').has('name','Tom Hanks').outE('ACTED_IN').inE('DIRECTED').has('name','Ron Howard').outE('DIRECTED').values('name')`



ACTED\_IN

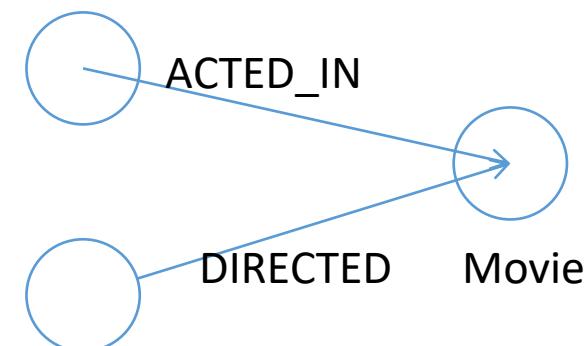


Person

Movie

DIRECTED

Person: Tom Hanks



DIRECTED

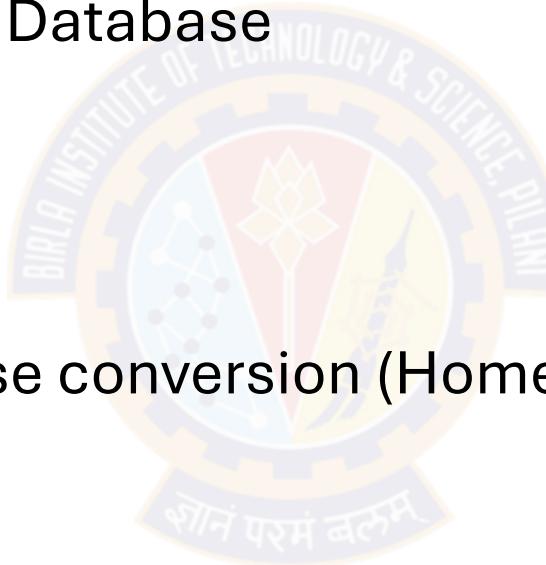
Movie

Person: Ron Howard

# Handson with GraphDB

Download and install [Neo4j Desktop](#)

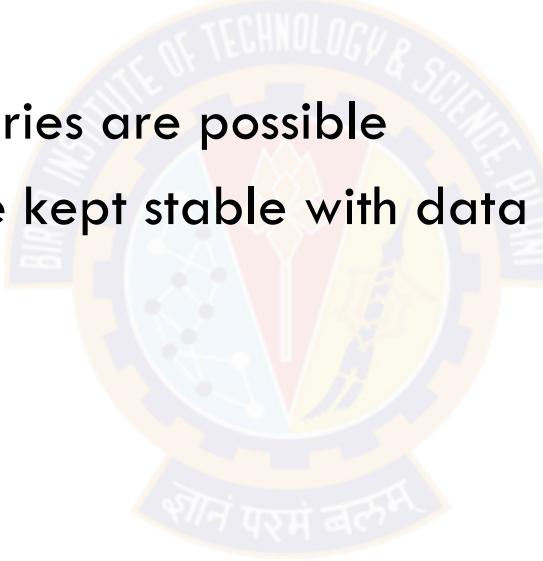
- ✓ Basic Cypher commands
- ✓ NoSQL features of Graph Database
- ✓ Setting Relationships
- ✓ Cypher queries
- ✓ Path finding
- ✓ RDBMS to Graph Database conversion (Home work)



# Summary

Graph DBs and computing models are very suitable when data sets are relationship heavy - can be modelled as large number of nodes and edges and queries are similar to graph traversal

- ✓ Complex relation centric queries are possible
- ✓ Graph traversal costs can be kept stable with data growth





## Next Session: Introduction to Apache Spark



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 12: Spark -Part 2

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

- **RDD operations**
- SparkSQL
- Standalone application
- Cluster architecture



# Types of RDD operations

- Transformations: Produce new RDD from existing RDD.

- ✓ Create a lineage of transformations - DAG.

- ✓ Lazy i.e. not done till an action is performed.

- ✓ e.g. map(), filter()

- Actions: Actually work on the data but new RDD is not formed.

- ✓ Non-RDD values returned to driver program or put on storage

- ✓ Trigger transformations to create lineage of RDDs required

- ✓ e.g. count()

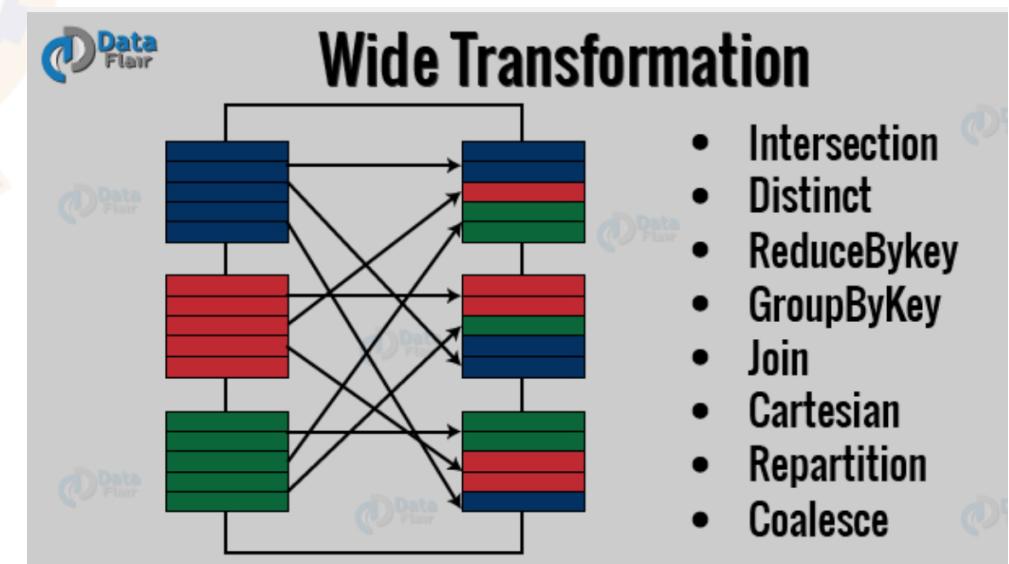
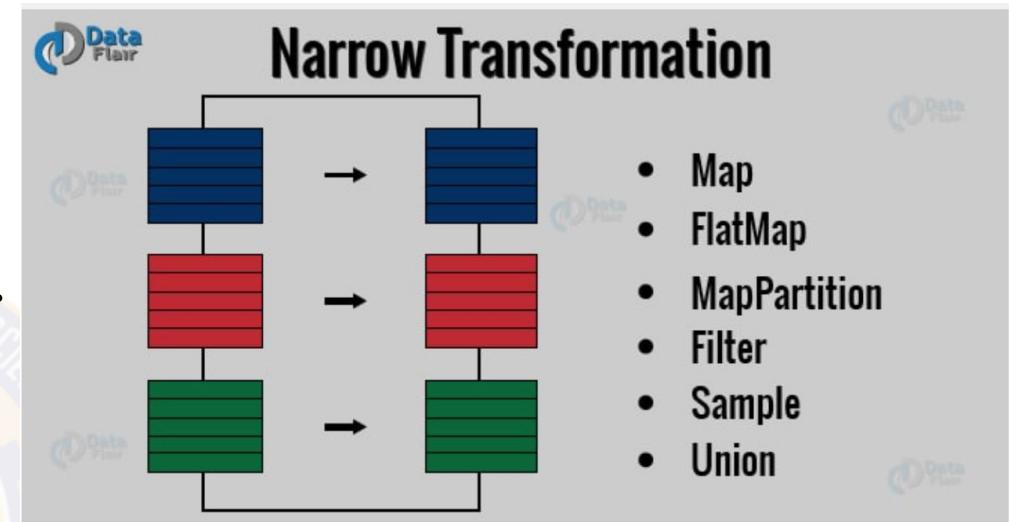
# Transformations

- Narrow

- ✓ All elements required to compute a partition of result RDD are in the corresponding partition in parent RDD.
- ✓ No shuffles across nodes.
- ✓ Examples:- `map()`, `mapPartition()`, `flatMap()`, `filter()`, `union()`

- Wide :

- ✓ Elements required to compute a partition of result RDD are in multiple partitions in parent RDD.
- ✓ Need shuffle across nodes.
- ✓ Examples:- `groupByKey()`, `aggregateByKey()`, `aggregate()`, `join()`, `repartition()`



# Transformations

- `map(f)`: Apply a function  $f$  over every element of an RDD and returns only one element - so  $N$  elements can create  $N$  output RDDs
- `flatMap(f)`: Same as `map` but can create an RDD with a list of  $N$  elements
- `filter(p)`: Return elements that only satisfy a predicate
- `mapPartition(f) / mapPartitionWithIndex(f)`: Same as `map` but  $f$  is applied on each partition of the input RDD in parallel instead of  $f$  being called on each element. Done to avoid overheads of moving data across cluster nodes, e.g. during initialisation.
- `union() / intersection()`: Create new RDD with union/intersection of elements of input RDDs
- `distinct()`: Returns distinct elements
- `groupByKey()`: Shuffles data by key in  $(k, v)$  lists. Leads to data movement across nodes / partitions.
- `reduceByKey(f)`: Similar to a combiner in Hadoop MapReduce
- `sortByKey()`: Create RDD with sort on key where we input  $(k, v)$  pairs in an RDD
- `join()`: Create RDD with join on key of input RDDs which have  $(k, v)$  lists
- `coalesce(n)`: Reduce the number of partitions of the RDD to  $n$

# Actions

- `count()` : Return number of elements in RDD
- `collect()`: Return RDD to driver and check whether it fits in memory
- `take(n)`: Return any n elements from RDD and minimise number of partitions it touches.
- `top(n)`: Return first n elements considering default order
- `countByValue()`: Return number of occurrences of each value
- `reduce()`: Take the set of elements of an RDD and apply a supplied commutative and associative operation on it, e.g. add
- `fold()`: Like reduce but takes an initial value, e.g. 0 for + or 1 for \*
- `aggregate()`: Different from reduce/fold because datatype of output can be different from input. One function to combine elements from RDD and then another to combine next level.
- `foreach()`: Work with each element in RDD but not return to driver. E.g. print each element.

# Pair RDD

- RDD with (key, value) pairs
  - ✓ Common format for processing data sets
- Enables a set of transformation operations
  - ✓ groupByKey
  - ✓ reduceByKey
  - ✓ combineByKey
  - ✓ mapValues(f) : just apply f to value
  - ✓ keys()
  - ✓ values()
  - ✓ sortByKey()
- and action operations
  - ✓ countByKey()
  - ✓ collectAsMap()
  - ✓ lookup(key)

**(05-PairRDD.txt)**

**(06-SumByMapReduce.txt)**



**06-Spark-RDDExamples-bank.txt**

# Examples (1)

```
> spark-shell
scala> val bankdata = sc.textFile("bank.csv")
scala> bankdata.take(1)
  Array[String] =
  Array(age,job,marital,education,default,balance,housing,loan,contact,day,month,duration,campaign,pdays,previous,poutcome,deposit)
scala> val bankdata2 = bankdata.flatMap(lines => lines.split("\n")).filter(value =>
value.contains("admin"))
scala> bankdata2.count()
  Long = 11334
convert each line to RDD and filter #records
scala> val bankdata3 = bankdata2.map(x => (x.split(",") (3), x.split(",") (0).toInt))
scala> bankdata3.foreach(println)
  (tertiary,37)
  (secondary,59)
  (secondary,56)
  (secondary,29)
  (tertiary,54)
  (secondary,28)
  (tertiary,33)
  (secondary,38)
extract education and age
...
...
```

# Examples (2)

```
scala> val bankdata4 = bankdata3.reduceByKey((x,y)=>x+y)
scala> bankdata4.foreach(println)
(secondary,42848)
(tertiary,5817)
(primary,2003)
(unknown,1857)
scala> val bankdata5 = bankdata4.map(_.swap)
scala> bankdata5.foreach(println)
(42848,secondary)
(5817,tertiary)
(2003,primary)
(1857,unknown)
scala> val sorted = bankdata5.sortByKey(numPartitions=1)
scala> sorted.foreach(println)
(1857,unknown)
(2003,primary)
(5817,tertiary)
(42848,secondary)
scala> bankdata4.lookup("primary")
res39: Seq[Int] = ArrayBuffer(2003)
```

add up all ages by education  
- just want to show how numbers are added up

swap key and value

sort : limit to one partition to avoid  
different sort orders since sorting is per partition

find value for a key

# Examples (3)

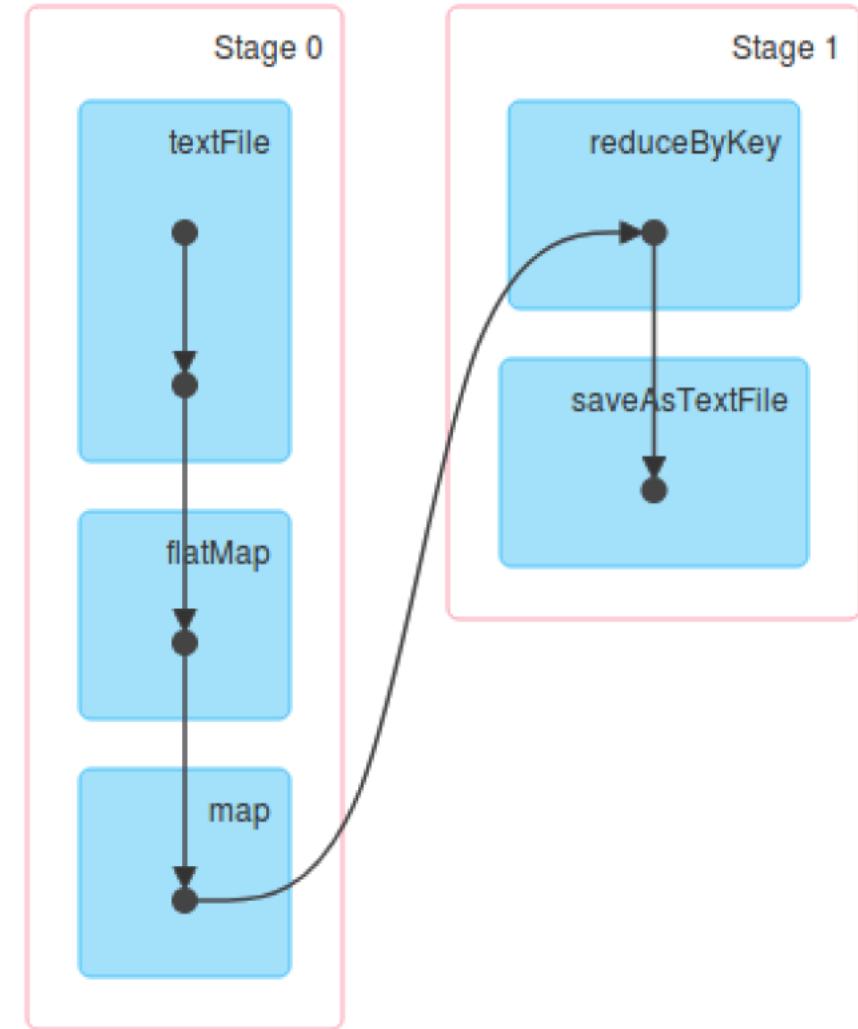
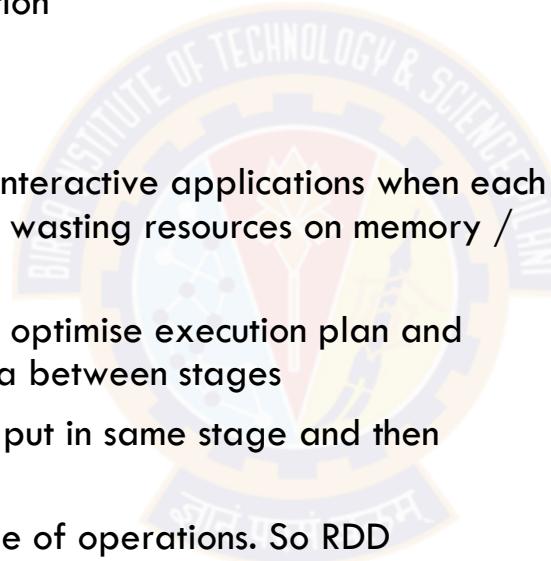
```
scala> val countsbykey = bankdata3.countByKey() # this is not an RDD because it is action operation
scala> val counts = sc.parallelize(Seq(countsbykey))
scala> counts.foreach(println)
Map(secondary -> 1084, tertiary -> 168, primary -> 42, unknown -> 40)
scala> val counts2 = counts.flatMap(x=>x)
scala> counts2.foreach(println)
(secondary,1084)
(tertiary,168)
(primary,42)
(unknown,40)
scala> val joinrdd = bankdata4.join(counts2)
scala> joinrdd.foreach(println)
(secondary,(42848,1084))
(tertiary,(5817,168))
(primary,(2003,42))
(unknown,(1857,40))
scala> val avg = joinrdd.mapValues{case (x,y) => (1.0 * x/y) }
scala> avg.foreach(println)
(secondary,39.52767527675277)
(tertiary,34.625)
(primary,47.69047619047619)
(unknown,46.425)
```



showing some manipulations on creating RDD from a list, joining 2 RDDs, and doing an average calculation on 2 columns

# Directed Acyclic Graph of RDDs

- Vertices are RDDs
- Edges are operations
- Operations are divided into stages
- Each stage has a set of tasks working on a partition
- Tasks may execute in parallel
- Why DAG ?
  - ✓ Avoid Hadoop MR limitation for iterative / interactive applications when each MR operation is considered independent - so wasting resources on memory / IO
  - ✓ Breaking upto into DAG and stages helps to optimise execution plan and avoid shuffling data around and sharing data between stages
  - ✓ Operations that don't need shuffling can be put in same stage and then shuffling happens across stages.
  - ✓ Achieves fault tolerance by capturing lineage of operations. So RDD partitions can be reconstructed on node failures.



# Topics for today

- RDD operations
- **SparkSQL**
- Standalone application
- Cluster architecture



# Spark SQL

- Integrate SQL querying into Spark programs (with other analytical processing) to query structured data
  - ✓ Distributed SQL query engine with in-memory processing
- Work with a variety of data in Hive tables, JSON files, Cassandra etc. via SQL interface
- Write your code in Python, Java, Scala or HiveQL
  - ✓ Can significantly speedup HiveQL queries with in-memory processing
- Can use JDBC/ODBC connectors with Spark SQL (ref Spark Thrift Server that is a port of HiveServer2)
- SparkSQL eliminated the need for firing MapReduce jobs in the background as done in Hive

# Method 1 - SparkSQL Tables – Create Table and load Data

```
val res = spark.sql("create database temp")
val res = spark.sql("show databases").show()
```

```
spark.sql("CREATE TABLE employee(id INT, name STRING, age INT) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n'")
```

```
val res = spark.sql("show tables")
```

```
res.show()
```

```
val res = spark.sql("desc employee")
```

```
res.show()
```

```
spark.sql("LOAD DATA INPATH '/employee.csv' INTO TABLE employee")
```

```
val res = spark.sql("FROM employee SELECT id, name, age where age > 40")
```

```
res.show()
```

**Handson Script: - /home/jps/Handson/SparkSQL/01-sparkSQL.txt**

# Method 2 – Create Table view from DataFrame

- Spark SQL works with DataFrames
- DataFrame is a distributed collection of data similar to RDD but organized into columns
- DataFrames are similar to relational tables
- Read data into dataframe and create a Table view
- val DF = spark.read.option("header",true).csv("bank.csv") // read CSV file into a DF
- DF.show() // Examine the DF
- **DF.createOrReplaceTempView("BANK")** // Create a Table named BANK from DF
- spark.sql("desc BANK").show() // Display schema
- spark.sql("SELECT age, job, balance FROM BANK").show(5) // SQL select query
- spark.sql("SELECT age, job, balance FROM BANK").where("job == 'admin.'").show(10)
- // SORT by age  
spark.sql(""" SELECT age, job, balance FROM BANK WHERE job in ('admin.','services') order by age""").show(10)
- // SQL GROUP BY clause  
spark.sql(""" SELECT job, count(\*) as count FROM BANK GROUP BY job""").show()
- //SQL JOIN  
val joinDF = spark.sql("select \* from EMP e, DEPT d where e.emp\_dept\_id == d.dept\_id")

**Handson Script:** - /home/jps/Handson/SparkSQL/01-sparkSQL.txt

# Joins with Spark SQL Tables

Table1 – Employee

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1	Smith	-1	2018	10	M	3000
2	Rose	1	2010	20	M	4000
3	Williams	1	2010	10	M	1000
4	Jones	2	2005	10	F	2000
5	Brown	2	2010	40		-1
6	Brown	2	2010	50		-1

Table 2 – Departments

dept_name	dept_id
Finance	10
Marketing	20
Sales	30
IT	40

# Joining the tables

Create Table views from Data frames

```
empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")
```

SQL JOIN

```
val joinDF = spark.sql("select * from EMP e, DEPT d where e.emp_dept_id == d.dept_id")
```

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary	dept_name	dept_id
1	Smith	-1	2018	10	M	3000	Finance	10
2	Rose	1	2010	20	M	4000	Marketing	20
3	Williams	1	2010	10	M	1000	Finance	10
4	Jones	2	2005	10	F	2000	Finance	10
5	Brown	2	2010	40		-1	IT	40

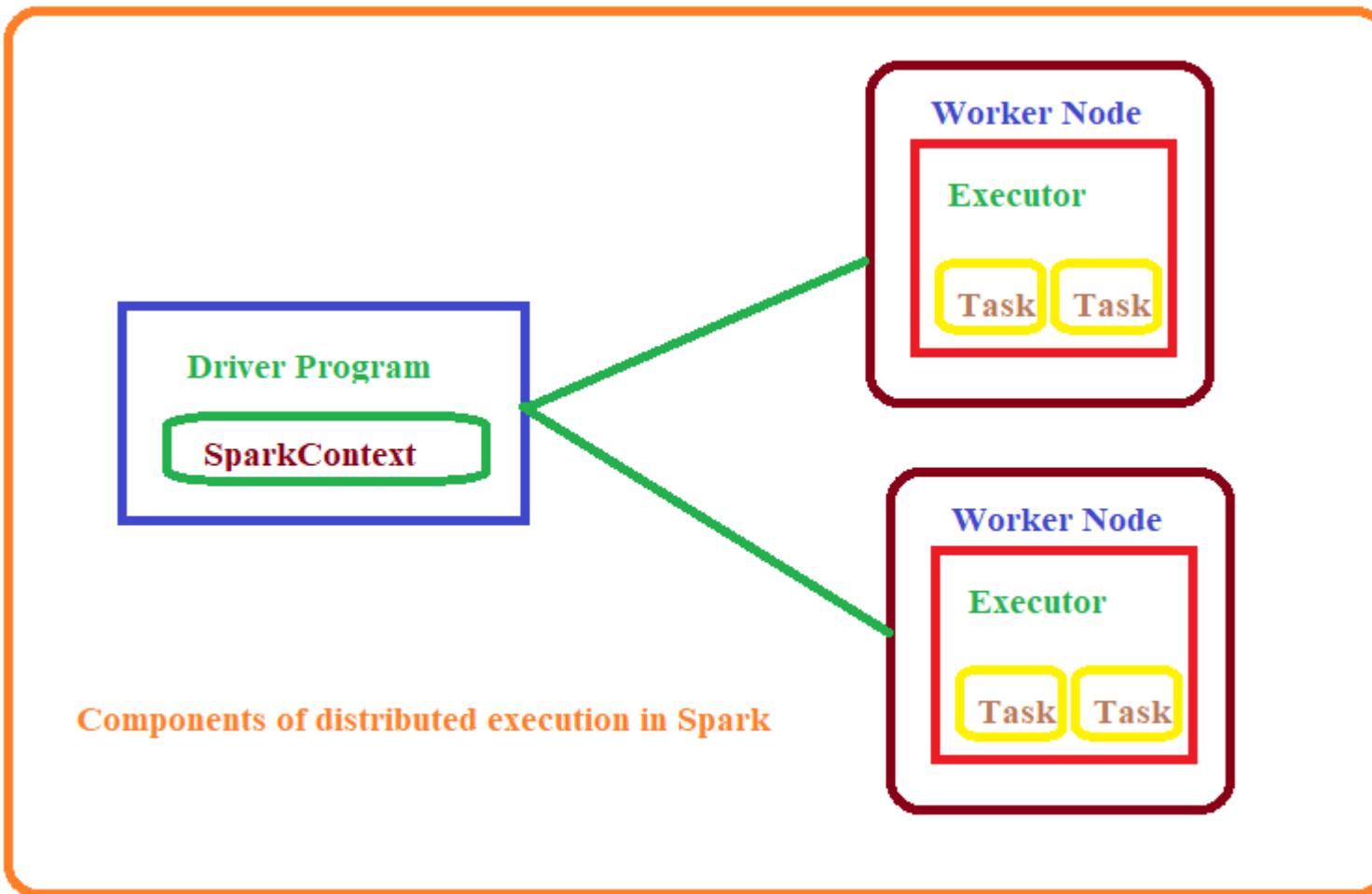
# Topics for today

- RDD operations
- SparkSQL
- **Standalone application**
- Cluster architecture



# Core Spark Concepts (Sec1)

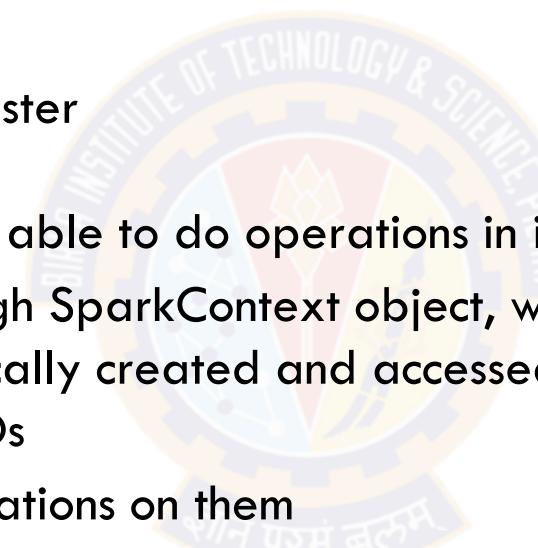
Components of distributed execution



# Core Spark Concepts (2)

## Driver Program

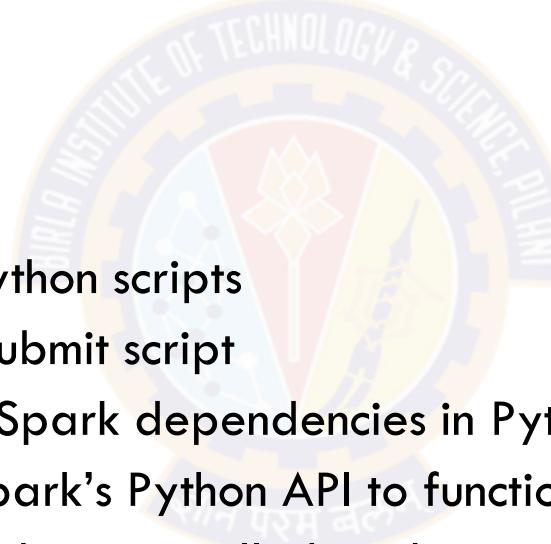
- Every Spark application consists of driver program that launches various parallel operations on a cluster
- Driver program
  - ✓ contains main function
  - ✓ defines distributed datasets on cluster
  - ✓ then applies actions to them
    - Shell is the driver program, hence able to do operations in it
- Driver program accesses Spark through SparkContext object, which is connection to the cluster
  - ✓ In shell, SparkContext is automatically created and accessed as “sc”
  - ✓ SparkContext is used to build RDDs
- RDDs can be used to run various operations on them
- To run these operations, driver program manage number of nodes running one more executors
  - ✓ Executors will be present on the different nodes



# Standalone Applications

## Using Python

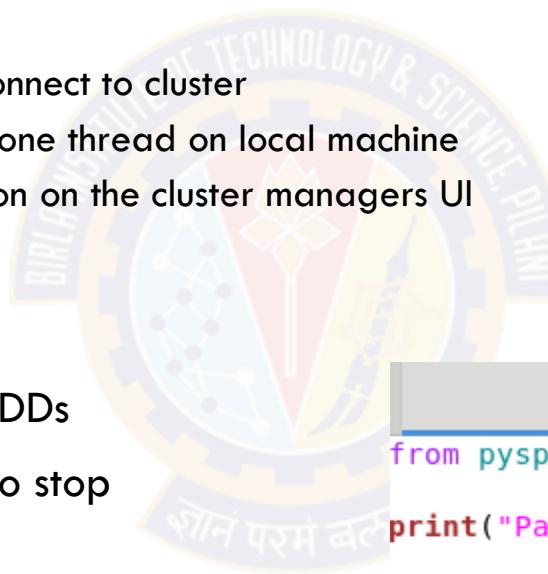
- Spark can be linked into standalone applications in Java, Scala or Python
  - ✓ Need to initialize the SparkContext in program
  - ✓ After that API is same!
- In Python,
  - ✓ Simply write applications as Python scripts
  - ✓ Run them using the bin/spark-submit script
    - ❖ Spark-submit includes the Spark dependencies in Python
    - ❖ Sets up environment for Spark's Python API to function
    - ❖ Need to have pyspark package installed on the system



# Standalone Applications(2)

## Initializing and Using SparkContext

- Import the packages in program
- Create SparkConf – to configure the applications
- Create SparkContext
  - ✓ A cluster URL, which tells Spark how to connect to cluster
  - ✓ Local is special value that runs Spark on one thread on local machine
  - ✓ Application name – to identify application on the cluster managers UI
- Use methods to create RDDs
- Use APIs to carry out operations on the RDDs
- Call stop() method SparkContext object to stop execution of application



```
my_script.py
from pyspark import SparkConf, SparkContext
print("Packages imported!")

conf = SparkConf().setMaster("local").setAppName("MyScript")
sc = SparkContext(conf = conf)

print('Able to contact Spark!')

lines = sc.textFile("README.md")
print('*****')
print(lines.count())
print('*****')
```

# Standalone Applications(3)

## Executing program

- Using spark-submit , execute the program



```
File Edit View Search Terminal Help
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ spark-submit my_script.py

20/04/15 13:31:51 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (PythonRDD[2] at count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12) (first 15 tasks are for partitions Vector(0))
20/04/15 13:31:51 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks
20/04/15 13:31:51 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, executor driver, partition 0, PROCESS_LOCAL, 7917 bytes)
20/04/15 13:31:51 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
20/04/15 13:31:51 INFO HadoopRDD: Input split: file:/home/csishydlab/spark-2.4.4-bin-hadoop2.7/README.md:0+3952
20/04/15 13:31:52 INFO PythonRunner: Times: total = 315, boot = 266, init = 49, finish = 0
20/04/15 13:31:52 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1547 bytes result sent to driver
20/04/15 13:31:52 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 428 ms on localhost (executor driver) (1/1)
20/04/15 13:31:52 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/04/15 13:31:52 INFO PythonAccumulatorV2: Connected to AccumulatorServer at host: 127.0.0.1 port: 58581
20/04/15 13:31:52 INFO DAGScheduler: ResultStage 0 (count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12) finished in 0.486 s
20/04/15 13:31:52 INFO DAGScheduler: Job 0 finished: count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12, took 0.539315 s
105
*****
```

# Topics for today

- SparkSQL
- RDD operations
- Standalone application
- **Cluster architecture**

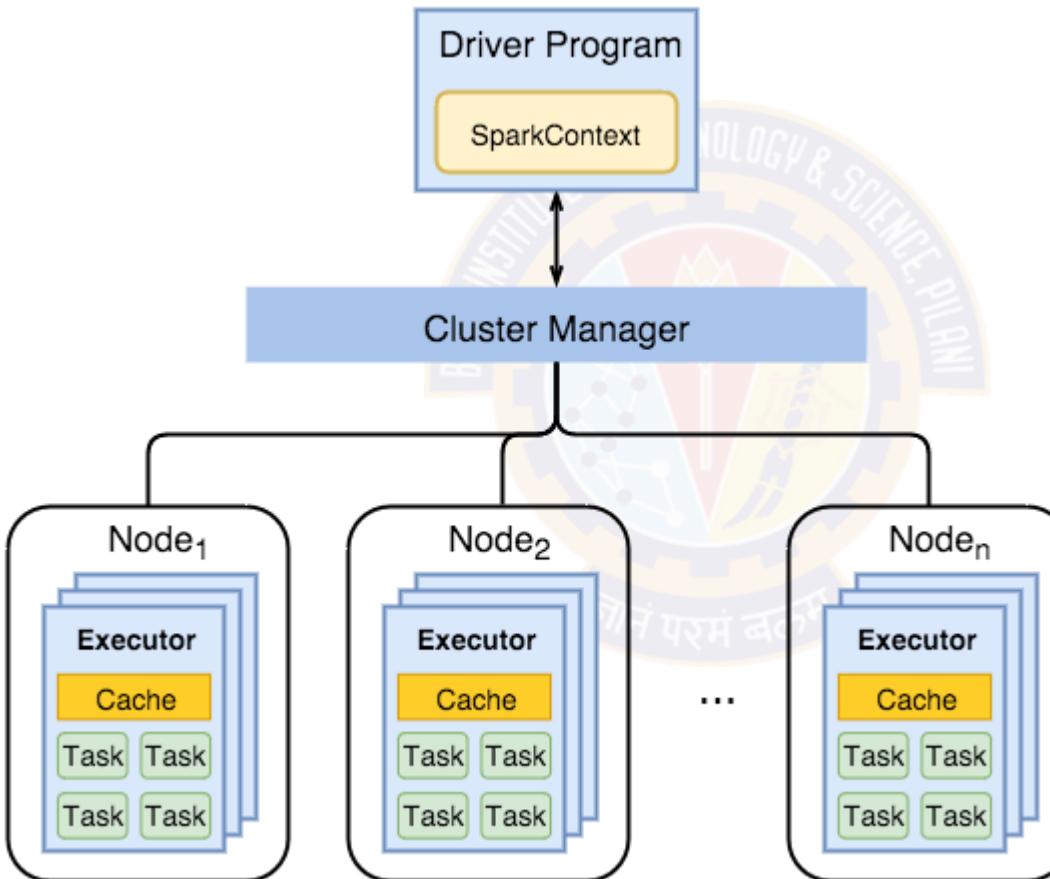


# Motivation

- Till now executed applications on the local (standalone) mode
- How to move to the cluster?
  - ✓ If same application needs to be run on the cluster, nothing needs to be changed!
  - ✓ Just need to add more machines and run it in the cluster mode!
  - ✓ All this possible because of high level API of Spark!
- Can rapidly prototype applications on smaller dataset locally
- Run unmodified code on very large clusters !

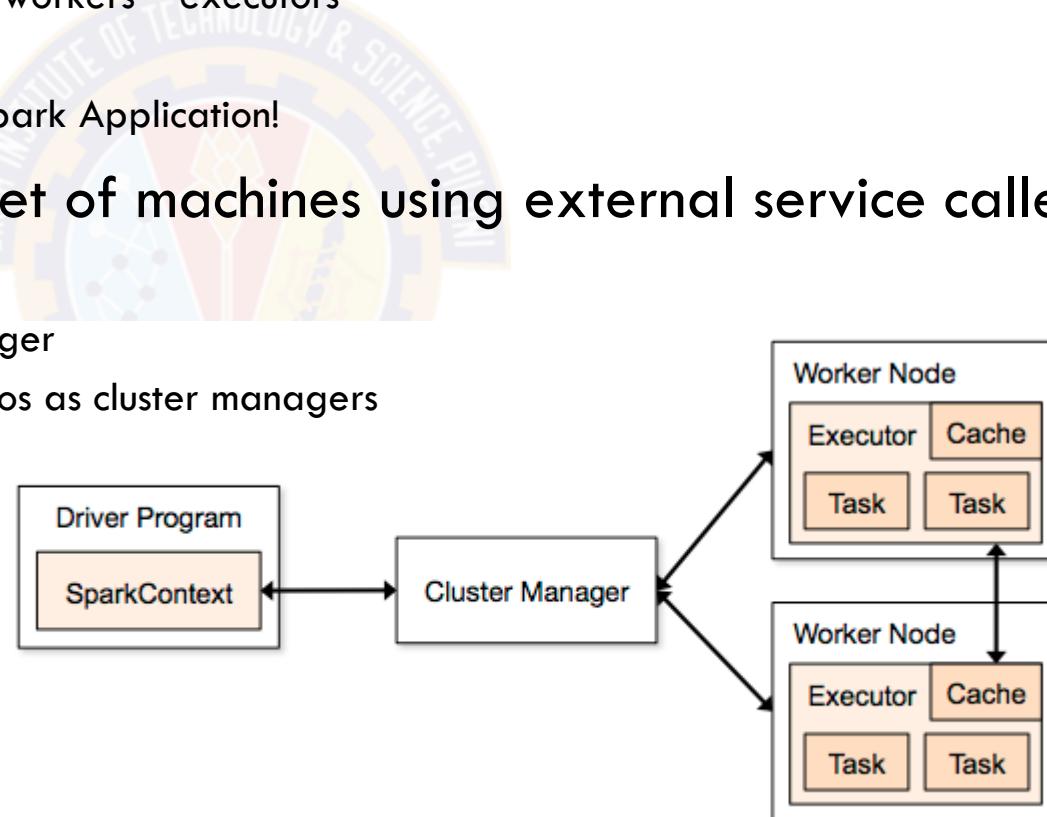
# Spark Runtime Architecture

## Components of Distributed Spark Application



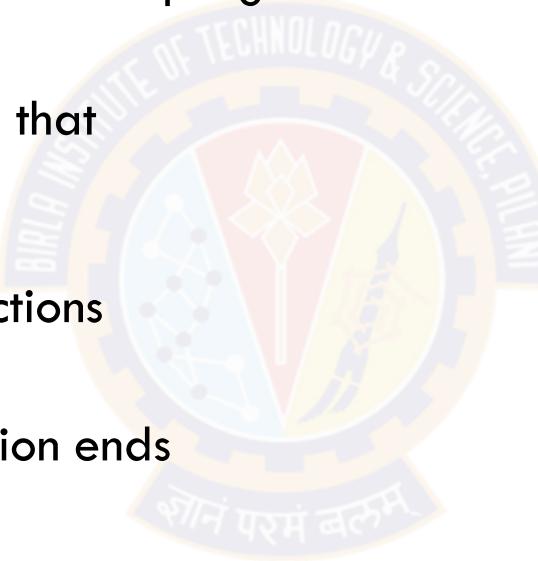
# Spark Runtime Architecture (2)

- Uses Master/Slave architecture with one central coordinator and many distributed workers
  - ✓ Central coordinator – driver
  - ✓ Driver communicates with large number of workers – executors
  - ✓ Both run in their separate Java processes
  - ✓ Driver and Executor together termed as Spark Application!
- Spark application launched on set of machines using external service called cluster manager
  - ✓ Spark has built-in standalone cluster manager
  - ✓ Also supports Hadoop YARN, Apache Mesos as cluster managers



# Driver

- Driver is the central coordinator for Spark application
  - ✓ Runs in separate Java process
  - ✓ It's the process where main() method of program runs
- It's the process running the user code that
  - ✓ Creates SparkContext
  - ✓ Creates RDDs
  - ✓ Performs transformations and actions
- Once the driver terminates, application ends
- Two responsibilities
  - ✓ Converting user program into tasks
  - ✓ Scheduling tasks on executors



# Driver (2)

Two Duties - Converting user program into tasks

- Converting user program into tasks

- ✓ Physical execution unit is called as task
- ✓ Smallest unit of work – task!
- ✓ Programs creates SparkContext, Create RDDs and perform transformations, actions on RDDs
- ✓ Implicitly a logical directed acyclic graph (DAG) of operations is created
- ✓ When driver runs, it converts the DAG into physical execution plan
- ✓ Converts execution plan into stages which in turn consists of tasks
- ✓ Tasks are bundled up and sent to the cluster

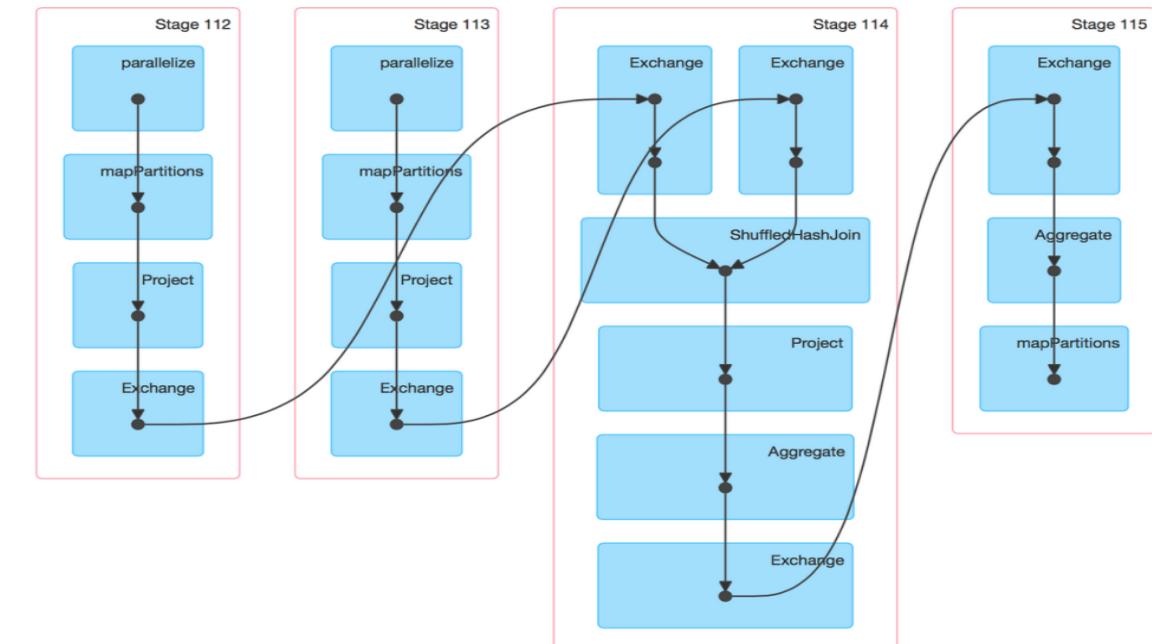


## Details for Job 8

Status: SUCCEEDED

Completed Stages: 4

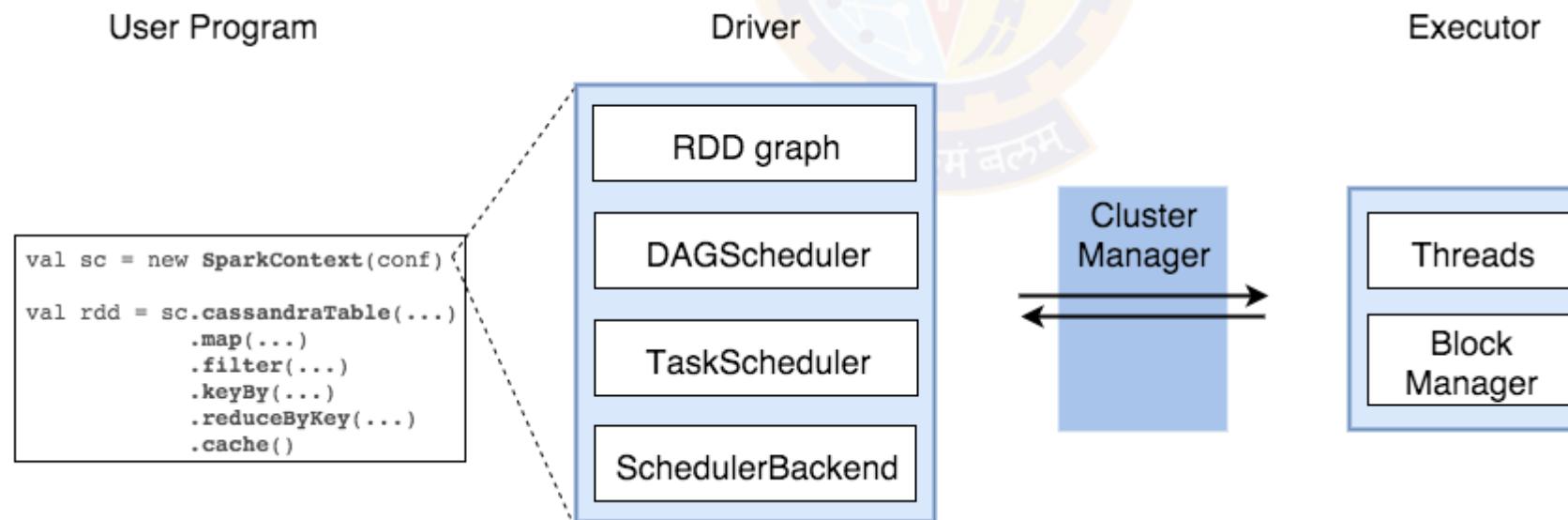
- ▶ Event Timeline
- ▼ DAG Visualization



# Driver (3)

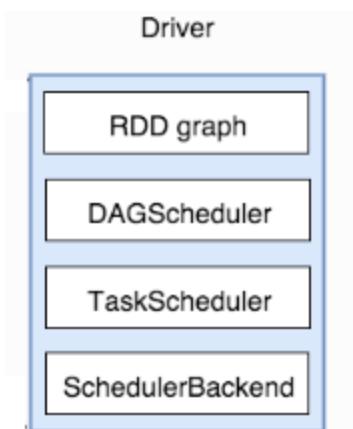
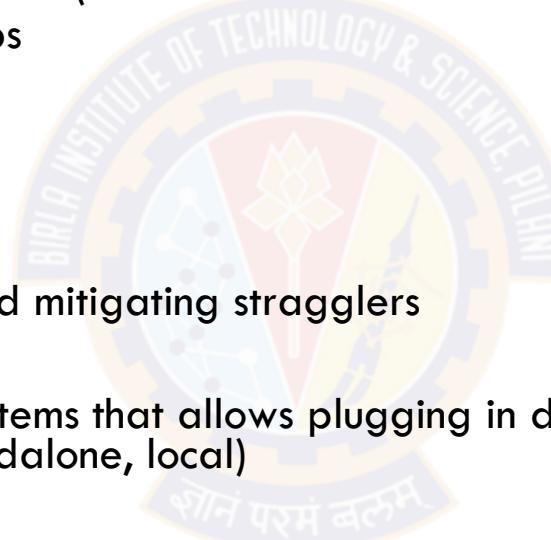
## Two Duties – Tasks Scheduling

- Scheduling the tasks on executors
  - ✓ With execution plan, Driver needs to coordinate scheduling of tasks on executors
  - ✓ When executor starts, it gets registered with driver, so driver has complete info about them
  - ✓ Executor is process capable of running tasks and storing RDD data
- Based on data placement, Spark will identify the executors to schedule the tasks
  - ✓ When tasks completed on executor, they might have cached data
  - ✓ Driver uses that cached data to schedule future tasks based on that data



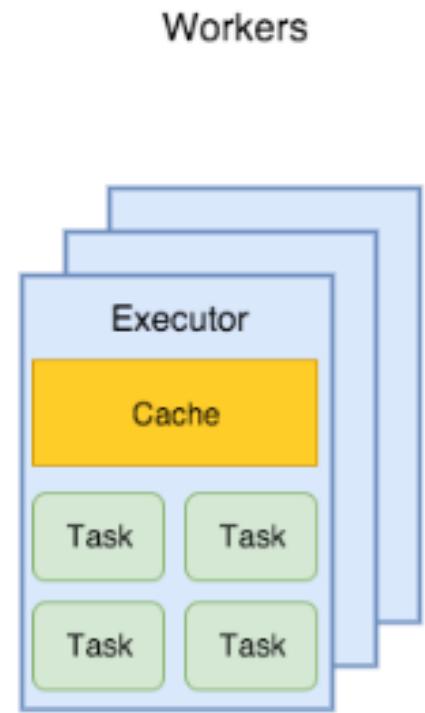
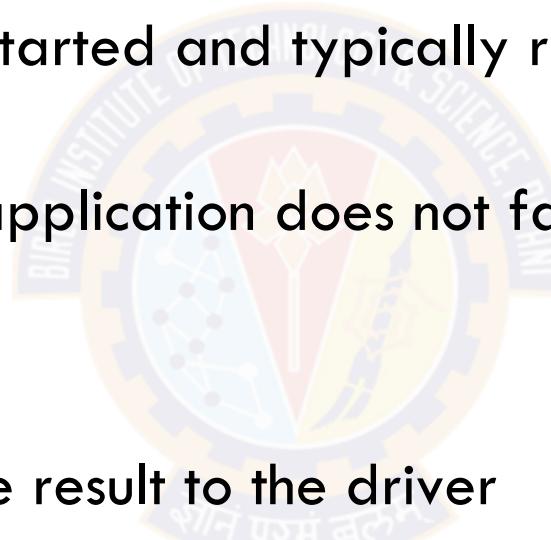
# Driver: Program translation into Tasks

- DAGScheduler
  - ✓ computes a DAG of stages for each job and submits them to TaskScheduler
  - ✓ determines preferred locations for tasks (based on cache status or shuffle files locations)
  - ✓ finds minimum schedule to run the jobs
- TaskScheduler
  - ✓ responsible for
    - ✓ sending tasks to the cluster
    - ✓ running them,
    - ✓ retrying if there are failures, and mitigating stragglers
- SchedulerBackend
  - ✓ backend interface for scheduling systems that allows plugging in different implementations (Mesos, YARN, Standalone, local)
- BlockManager
  - ✓ provides interfaces for putting and retrieving blocks both locally and remotely into various stores (memory, disk)



# Executors

- Worker processes responsible for running individual tasks in given Spark job
- Launched when application is started and typically run till lifetime of application
- Even if executors fails, Spark application does not fail
- Roles of executors
  - ✓ Run the tasks and return the result to the driver
  - ✓ Provide in-memory storage for RDDs that are cached by user programs

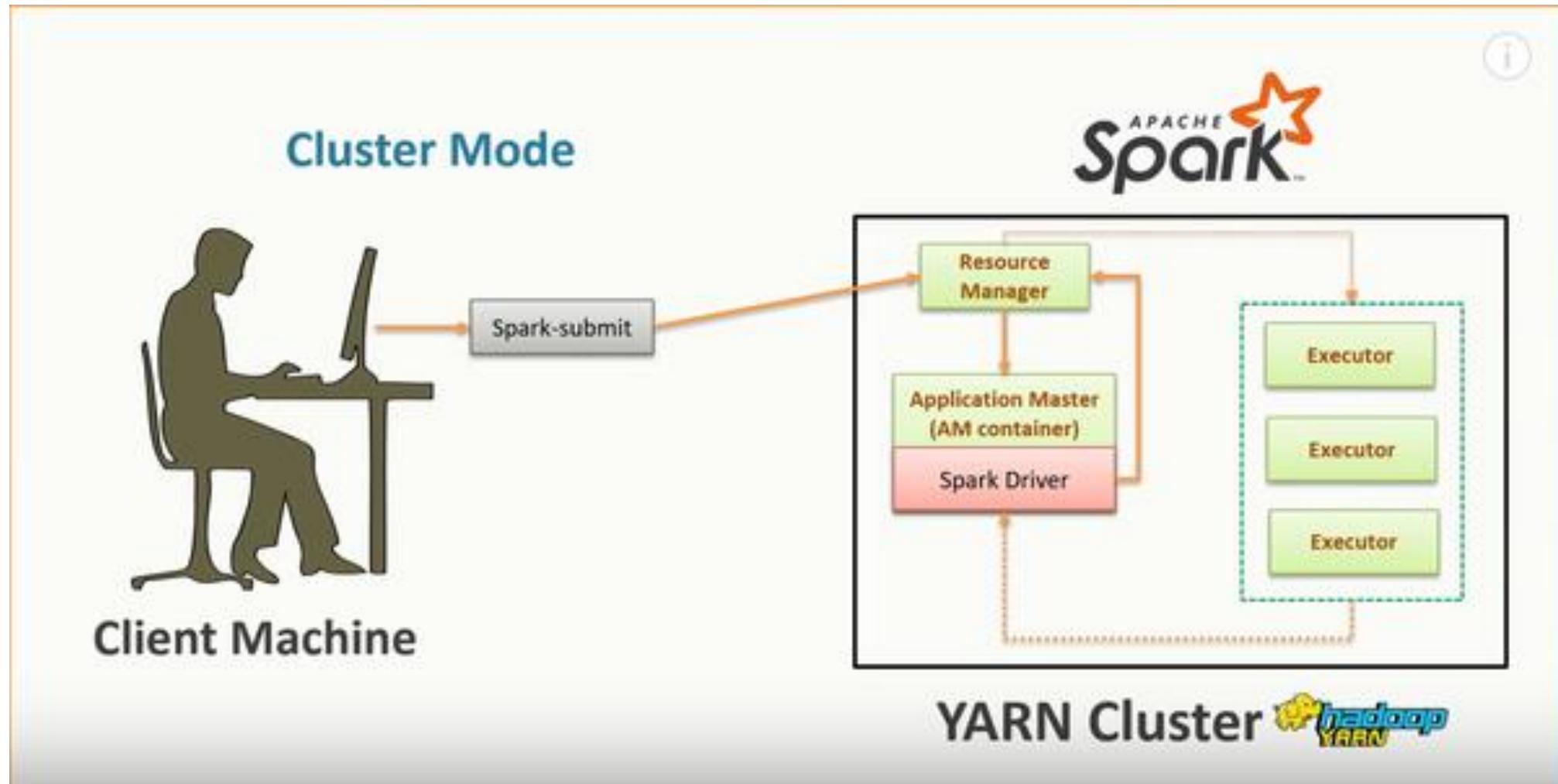


# Cluster Manager

- Cluster manager is used to launch the Spark application
- Pluggable component
- Allows Spark to run on top of different external managers like YARN, Mesos as well as standalone cluster manager
- Single script to submit your application to cluster i.e. spark-submit
  - ✓ With different options, it can be used to connect to different clusters
  - ✓ Used to manage the resources allotted to the application



# Application execution



Source : [Quora answer](#)

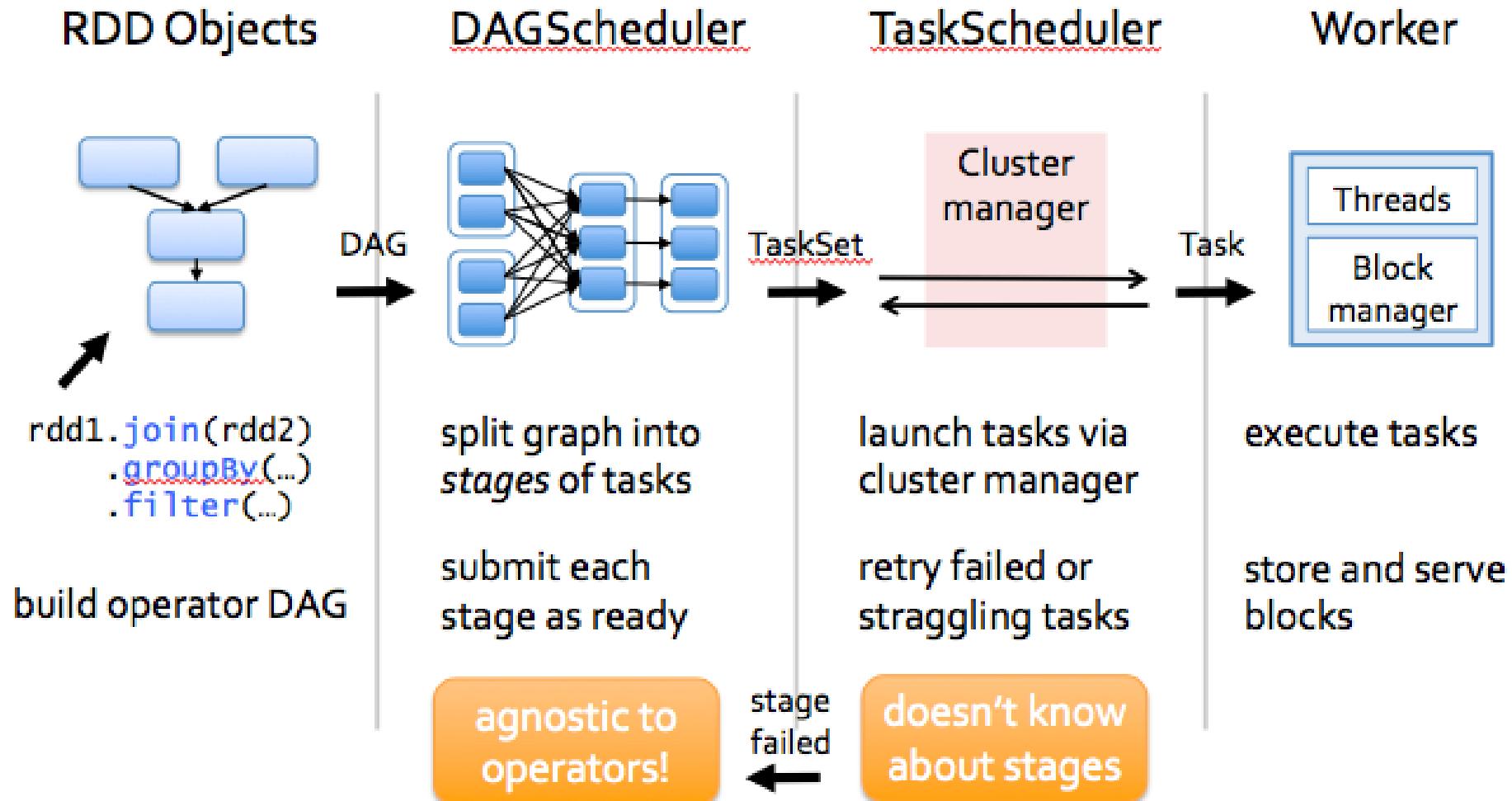
# Application execution (2)

## Detailed

1. User submits the application using spark-submit
2. Spark-submit launches the driver program and invokes the main() method of program
3. Driver program contacts the cluster manager to ask for resources to launch executors
4. Cluster manager launches the executors on the behalf of driver program
5. Driver process runs through user application
  - Based on RDD transformations and actions in the program, driver sends work to executors in form of tasks
6. Tasks are run on executor processes to compute and save results
7. If the driver's main method exits or calls SparkContext.stop(), it will terminate all executors and release resources from the cluster manager

# Application execution (3)

## Application flow



# Cluster concepts

Application	User program built on Spark. Consists of a <i>driver program</i> and <i>executors</i> on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called <i>stages</i> that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

# Submitting Applications

## Spark-submit

- Used to launch applications on a cluster
- Can use all of Spark's supported cluster managers through a uniform interface
  - ✓ don't have to configure application especially for each one
- Bundling Application's Dependencies
  - If code depends on other projects, need to package them alongside application in order to distribute the code to a Spark cluster
    - ✓ Need to create an assembly jar (or "uber" jar) containing code and its dependencies
    - ✓ Both sbt and Maven have assembly plugins
    - ✓ Spark and Hadoop as provided dependencies – provided at runtime
    - ✓ Call the bin/spark-submit script while passing jar
  - For Python,
    - ✓ Can use the --py-files argument of spark-submit to add .py, .zip files to be distributed with application
    - ✓ If depend on multiple Python files recommend packaging them into a .zip or .egg.

# Launching Applications with spark-submit

- Once a user application is bundled, it can be launched using the bin/spark-submit script
- This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports



```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
--deploy-mode <deploy-mode> \
--conf <key>=<value> \
... # other options
<application-jar> \
[application-arguments]
```

# Launching Applications with spark-submit (2)

## Common options

- **--class:**
  - ✓ The entry point for application (e.g. org.apache.spark.examples.SparkPi)
- **--master:**
  - ✓ The master URL for the cluster (e.g. spark://23.195.26.187:7077)
- **--deploy-mode:**
  - ✓ Whether to deploy your driver on the worker nodes (cluster) or locally as an external client (client) (default: client)
- **--conf:**
  - ✓ Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap “key=value” in quotes
- **application-jar:**
  - ✓ Path to a bundled jar including application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an hdfs:// path or a file:// path that is present on all nodes.
- **application-arguments:**
  - ✓ Arguments passed to the main method of your main class, if any

# Launching Applications with spark-submit (3)

## On Standalone cluster

```
# Run application locally on 8 cores
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local[8] \
/path/to/examples.jar \
100

# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://207.184.161.138:7077 \
--executor-memory 20G \
--total-executor-cores 100 \
/path/to/examples.jar \
1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://207.184.161.138:7077 \
--deploy-mode cluster \
--supervise \
--executor-memory 20G \
--total-executor-cores 100 \
/path/to/examples.jar \
1000
```

# Launching Applications with spark-submit (4)

On YARN cluster

```
# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \ # can be client for client mode
--executor-memory 20G \
--num-executors 50 \
/path/to/examples.jar \
1000
```

# Launching Applications with spark-submit (4)

On Mesos and Kubernetes cluster

```
# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000

# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

# Launching Applications with spark-submit (5)

## Submitting Python Code

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
--master spark://207.184.161.138:7077 \
examples/src/main/python/pi.py \
1000
```



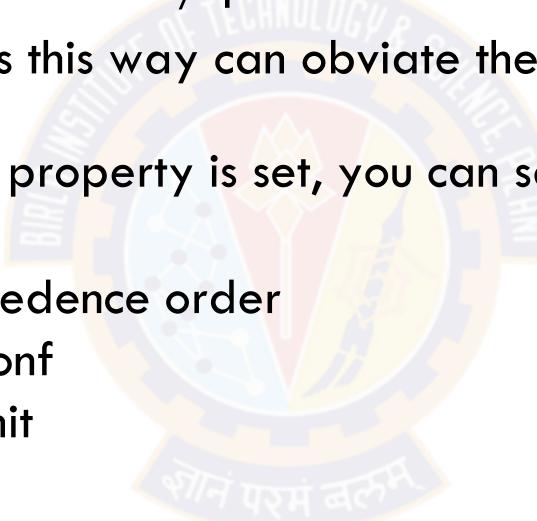
# Master URLs

Master URL	Meaning
local	Run Spark locally with one worker thread (i.e. no parallelism at all).
local[K]	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
local[*]	Run Spark locally with as many worker threads as logical cores on your machine.
spark://HOST:PORT	Connect to the given <a href="#">Spark standalone cluster</a> master. The port must be whichever one your master is configured to use, which is 7077 by default.
spark://HOST1:PORT1,HOST2:PORT2	Connect to the given <a href="#">Spark standalone cluster with standby masters with Zookeeper</a> . The list must have all the master hosts in the high availability cluster set up with Zookeeper. The port must be whichever each master is configured to use, which is 7077 by default.
mesos://HOST:PORT	Connect to the given <a href="#">Mesos</a> cluster. The port must be whichever one your is configured to use, which is 5050 by default. Or, for a Mesos cluster using ZooKeeper, use <code>mesos://zk://...</code> . To submit with <code>--deploy-mode cluster</code> , the HOST:PORT should be configured to connect to the <a href="#">MesosClusterDispatcher</a> .
yarn	Connect to a <a href="#">YARN</a> cluster in <code>client</code> or <code>cluster</code> mode depending on the value of <code>--deploy-mode</code> . The cluster location will be found based on the <code>HADOOP_CONF_DIR</code> or <code>YARN_CONF_DIR</code> variable.

# Loading Configuration from a File

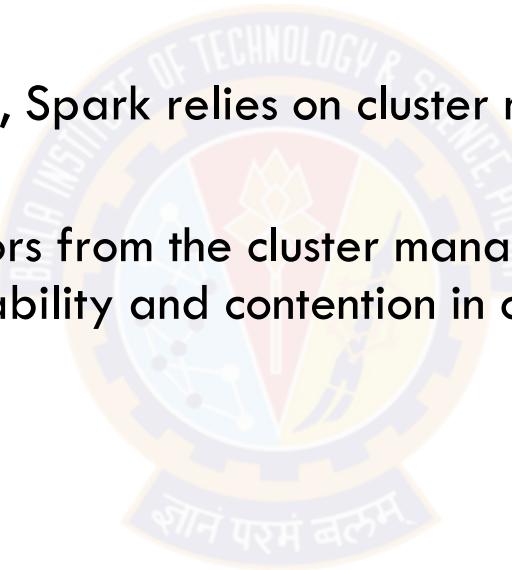
## Configuration file

- The spark-submit script can load default Spark configuration values from a properties file and pass them on to application
  - ✓ By default, it will read options from conf/spark-defaults.conf in the Spark directory
- Loading default Spark configurations this way can obviate the need for certain flags to spark-submit.
  - ✓ For instance, if the spark.master property is set, you can safely omit the --master flag from spark-submit.
- In general, configuration values precedence order
  - ✓ Directly set values on a SparkConf
  - ✓ then flags passed to spark-submit
  - ✓ then values in the defaults file.
- If you are ever unclear where configuration options are coming from, you can print out fine-grained debugging information by running spark-submit with the --verbose option.



# Scheduling Spark Applications

- Many clusters are shared between multiple users/programs
  - ✓ Challenge is how to schedule the jobs?
  - ✓ What happens if two users launch applications that each want to use entire clusters resources?
- For scheduling in multi-tenant clusters, Spark relies on cluster managers to share resources between spark applications
- When a application asks for executors from the cluster manager, it may receive more or fewer resources depending upon the availability and contention in clusters
- Variety of cluster managers
  - ✓ Standalone
  - ✓ Hadoop YARN
  - ✓ Apache Mesos



# Standalone Cluster Manager

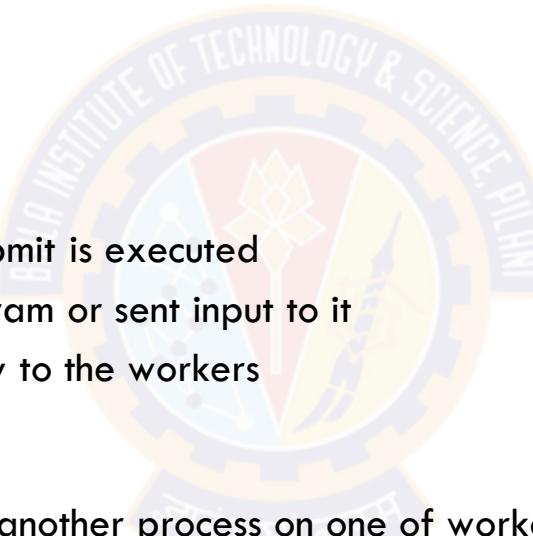
## Submitting applications

- Simple way to run applications on cluster
- Consists of master and multiple workers , each with configured amount of memory and CPU cores
- When app is submitted, user specifies
  - ✓ how much memory its executors will use
  - ✓ Total number of cores across all executors
- Submitting applications
  - ✓ Need to set master argument as sparkL//masternode:7077
  - ✓ Web UI can be accessed at same cluster URL

# Standalone Cluster Manager (2)

## Deployment modes

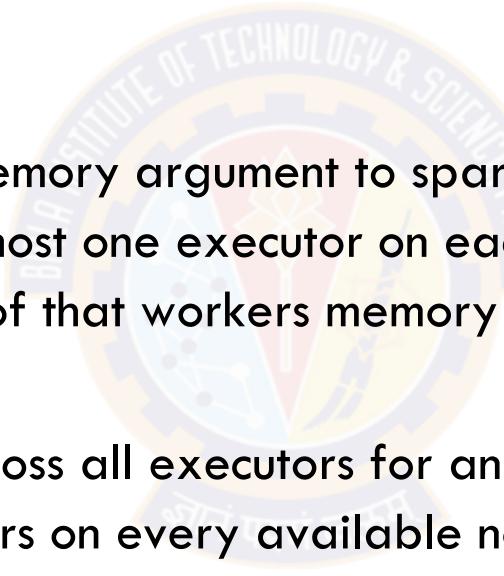
- Two deploy modes – where driver program of application runs
  - ✓ Client mode (default)
  - ✓ Cluster mode
- Client mode
  - ✓ Driver runs on machine where spark-submit is executed
  - ✓ Directly can see the output of the program or sent input to it
  - ✓ Machine needs to have fast connectivity to the workers
- Cluster mode
  - ✓ Driver is launched within the cluster, as another process on one of worker nodes
  - ✓ Spark-submit is “fire and forget” – close the laptop still applications run!
  - ✓ Will be able to access logs through cluster managers Web UI



# Standalone Cluster Manager (3)

## Resource configurations

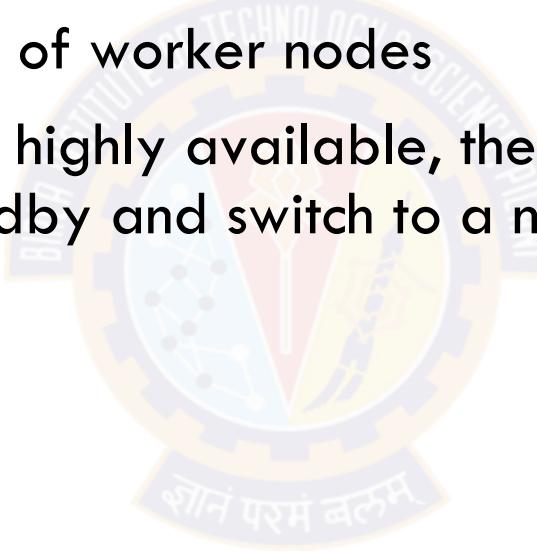
- Needs to decide how to allocate resources between the executors
- Managed through two settings
- Executor memory
  - ✓ Configured using `--executor-memory` argument to `spark-submit` (default : 1GB)
  - ✓ Each application will have at most one executor on each worker node
  - ✓ This setting controls how much of that workers memory the application will claim
- Maximum number of total cores
  - ✓ Total number of cores used across all executors for an application (default : unlimited)
  - ✓ Application will launch executors on every available node of cluster
  - ✓ Set through `--total-executor-cores` argument to `spark-submit`



# Standalone Cluster Manager (4)

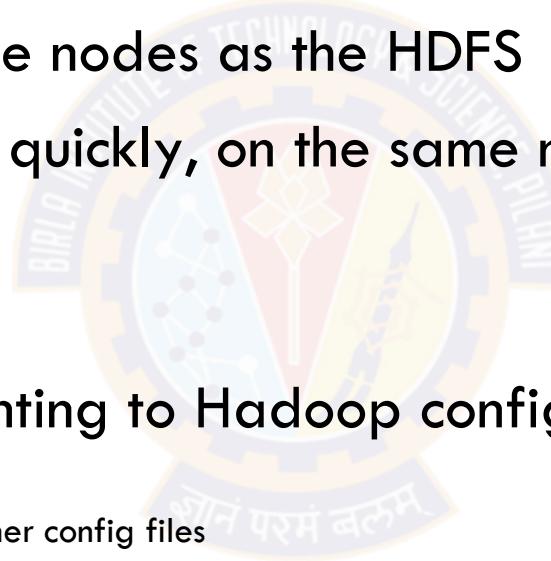
## High Availability

- Cluster needs to be available even if the worker nodes fails
- Gracefully support the failure of worker nodes
- If the master also needs to be highly available, then zookeeper can be used to keep multiple masters on standby and switch to a new one when any of them fails



## Submitting applications

- Introduced in Hadoop 2.0
- Typically installed on the same nodes as the HDFS
- Lets Spark access HDFS data quickly, on the same nodes where the data is stored
- Using it straightforward:
- Set environment variable pointing to Hadoop configuration directory
  - ✓ HADOOP\_CONF\_DIR
  - ✓ Directory that contains yarn-site.xml and other config files
  - ✓ Usually HADOOP\_HOME/conf
- Submit jobs to a special master URL with spark-submit
  - `spark-submit --master yarn my_app`



# Hadoop YARN (2)

## Deployment modes

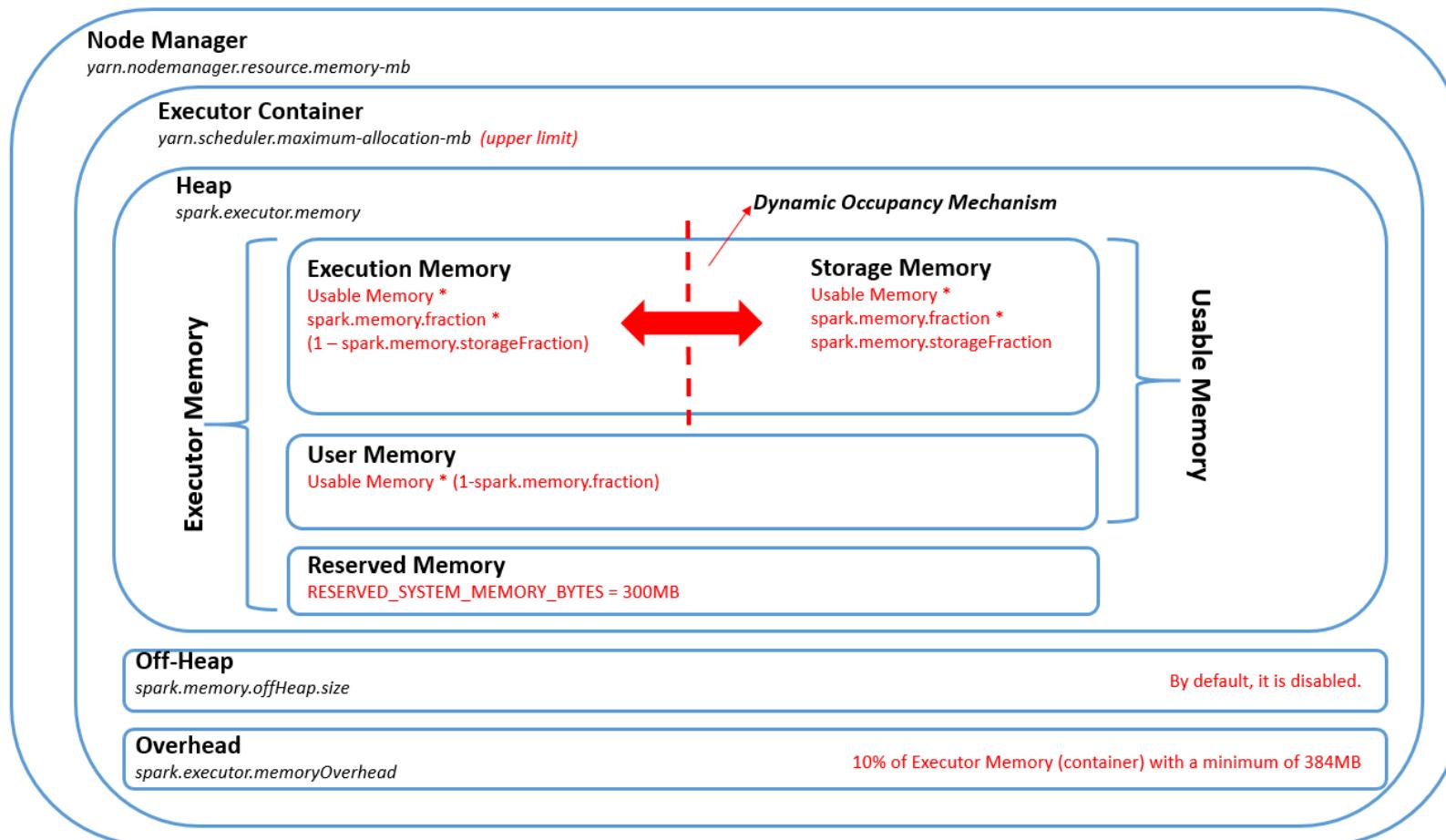
- Set the mode through `--deploy-mode` argument to `spark-submit`
- Two deployment modes
  - ❖ Client mode
    - ✓ Driver program for application runs on machine from which application is submitted
  - ❖ Cluster mode
    - ✓ Driver runs inside YARN container



# Spark Configuration Settings

Deciding about Pyspark configuration parameters with the usage of YARN as a cluster management framework

[Basics of Apache Spark Configuration Settings | by Halil Ertan | Towards Data Science](#)



# Hadoop YARN (3)

## Resource configurations

- **--num-executors**
  - ✓ Spark applications use a fixed number of executors (default:2)
  - ✓ Set through spark-submit command
- **--executor-memory**
  - ✓ Memory used by each executor (default : 8GB)
- **--executor-cores**
  - ✓ Number of cores it claims from YARN
- Usually, Spark will run better with a smaller number of large executors (with more memory and cores) since it can optimize the communication within each executor



Optimum Parallelism in Spark Framework on Hadoop YARN for Maximum Cluster Resource Utilization  
[https://link.springer.com/chapter/10.1007/978-981-15-0029-9\\_28](https://link.springer.com/chapter/10.1007/978-981-15-0029-9_28)

# Apache Mesos

## Submitting applications

- General purpose cluster manager – can run analytics workloads and long running services
- Need to pass mesos URI with spark-submit
  - ✓ spark-submit --master mesos://masternode:5050 my\_app
- Zookeeper can be used to elect a master when running on multimaster node
- Connecting Spark to Mesos
  - ✓ To use Mesos from Spark, you need a Spark binary package available in a place accessible by Mesos, and a Spark driver program configured to connect to Mesos.
  - ✓ Alternatively, you can also install Spark in the same location in all the Mesos slaves, and configure spark.mesos.executor.home (defaults to SPARK\_HOME) to point to that location.

# Apache Mesos (2)

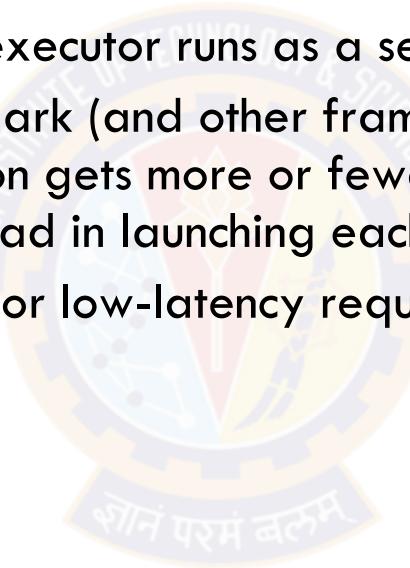
## Scheduling modes

- Spark can run over Mesos in two modes: “coarse-grained” and “fine-grained” mode
- In “coarse-grained” mode
  - ✓ Each Spark executor runs as a single Mesos task.
  - ✓ Spark executors are sized according to the following configuration variables:
    - Executor memory: `spark.executor.memory`
    - Executor cores: `spark.executor.cores`
    - Number of executors: `spark.cores.max/spark.executor.cores`
- Executors are brought up eagerly when the application starts, until `spark.cores.max` is reached
  - ✓ If you don't set `spark.cores.max`, the Spark application will consume all resources offered to it by Mesos
- The benefit of coarse-grained mode is much lower startup overhead, but at the cost of reserving Mesos resources for the complete duration of the application.

# Apache Mesos (3)

## Scheduling modes

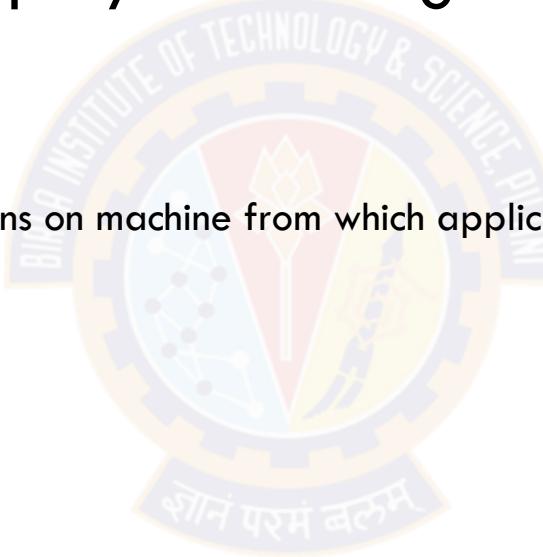
- In “fine-grained” mode,
  - ✓ Each Spark task inside the Spark executor runs as a separate Mesos task
  - ✓ This allows multiple instances of Spark (and other frameworks) to share cores at a very fine granularity, where each application gets more or fewer cores as it ramps up and down, but it comes with an additional overhead in launching each task
  - ✓ This mode may be inappropriate for low-latency requirements like interactive queries or serving web requests
  - ✓ It is deprecated now!



# Apache Mesos (4)

## Deployment modes

- Set the mode through –deploy-mode argument to spark-submit
- Two deployment modes
  - ❖ Client mode
    - ✓ Driver program for application runs on machine from which application is submitted
  - ❖ Cluster mode
    - ✓ Driver runs inside Mesos Cluster



# Which Cluster Manager to use?

## Guidelines to choose a cluster manager

- **If this is new deployment**
  - ✓ Start with standalone cluster as its easy to setup
  - ✓ Provides almost all the features as the other cluster managers
- **If needs to run Spark alongside with other applications or to use richer resource scheduling capabilities**
  - ✓ Both YARN and Mesos provide these features
  - ✓ YARN will be preinstalled in Hadoop distributions
  - ✓ Mesos offers more fine grained resource management
- **In all cases,**
  - ✓ Best to run Spark on the same nodes as HDFS for fast access to storage
  - ✓ Hadoop distribution installs YARN and HDFS together
  - ✓ Mesos and Standalone cluster managers can be installed manually on the same nodes

# Summary

- RDD operations to help create data analysis programs
- SparkSQL provides SQL like queries on structured data
- Spark applications can be run standalone or on clusters with no code changes
- Easier to move from small development environments to larger staging / production environments by just scaling number of nodes
- Can run on a variety of clusters, including Hadoop YARN
- Developers focus more on logic than deployment scenarios
- Extensive visibility into application execution and cluster operations



**Next Session:  
Spark - Part 3**



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 13: Spark -Part 3

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Topics for today

## Spark Performance

### Spark MLlib

- ✓ Regression
- ✓ Classification
- ✓ Clustering
- ✓ Collaborative filtering



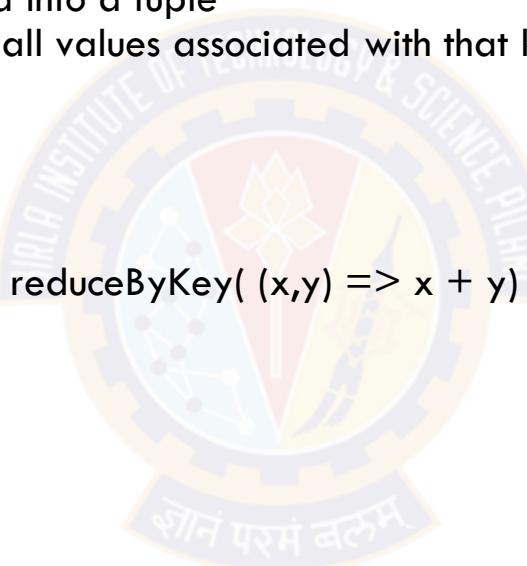
# Shuffle Operations

- Certain operations (wide transformations) within Spark trigger an event known as the shuffle
- The shuffle is Spark's mechanism for re-distributing data so that it's grouped differently across partitions
- This typically involves copying data across executors and machines, making the shuffle a complex and costly operation
- In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation
- During computations, a single task will operate on a single partition - thus, to organize all the data for a single `reduceByKey` reduce task to execute, Spark needs to perform an all-to-all operation
- It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key - this is called the **shuffle**.
- Additional operations which can cause a shuffle include
  - ✓ repartition operations like `repartition` and `coalesce`
  - ✓ 'ByKey operations (except for counting) like `groupByKey` and `reduceByKey`
  - ✓ join operations like `co-group` and `join`.

# Shuffle

## reduceByKey() Example

- The reduceByKey operation generates a new RDD
- For this:
  - ✓ all values for a single key are combined into a tuple
  - ✓ by executing a reduce function against all values associated with that key
- For example,
  - ✓ Let pair RDD is { (1,2), (3,4), (3, 6) }
  - ✓ Apply reduceByKey() on rdd ---→ rdd. reduceByKey(  $(x,y) \Rightarrow x + y$  )
  - ✓ Result will be
    - {
    - (1, 2)
    - (3, 10)
    - }
- The challenge is that not all values for a single key necessarily reside
  - ✓ on the same partition
  - ✓ or even the same machine
  - ✓ all keys don't have same number of values
- but they must be co-located to compute the result



# Broadcast variables

- Broadcast variables are read-only shared variables that are cached and available on all nodes in a cluster
- Instead of sending data along with every task, spark distributes broadcast variables to the nodes.
- Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.
- Broadcast variables are used in the same way for RDD, DataFrame, and Dataset
- Typical use of broadcast variable is in sharing Lookup tables across nodes of the cluster
- Broadcast variables are created from a variable `v` by calling `SparkContext.broadcast(T, scala.reflect.ClassTag<T>)`.
- Eg:

```
val broadcastVar = sc.broadcast(Array(1, 2, 3))
```

```
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(0)
```

# RDD Persistence

- One of the most important capabilities in Spark is persisting (or caching) a dataset in memory across operations
- When you persist an RDD
  - ✓ every node stores any partitions of it that it computes in memory
  - ✓ reuses them in other actions on that dataset (or datasets derived from it)
  - ✓ allows future actions to be much faster (often by more than 10x)
- **Caching is a key tool for iterative algorithms and fast interactive use.**
- Can mark an RDD to be persisted using the `persist()` or `cache()` methods on it
  - ✓ The first time it is computed in an action, it will be kept in memory on the nodes
  - ✓ Spark's cache is fault-tolerant – if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it

# RDD storage level (1)

- Each persisted RDD can be stored using a different storage level, allowing you,
  - ✓ to persist the dataset on disk
  - ✓ to persist it in memory but as serialized Java objects (to save space)
  - ✓ to replicate it across nodes
- These levels are set by passing a StorageLevel object (Scala, Java, Python) to persist()
  - ✓ The cache() method is a shorthand for using the default storage level:
    - ✓ StorageLevel.MEMORY\_ONLY for RDD
    - ✓ And StorageLevel.MEMORY\_AND\_DISK for Dataset
  - ✓ Usage Example:

```
import org.apache.spark.storage.StorageLevel
df=spark.read.format("csv").option("header","true").load("/xyz.csv").persist(StorageLevel.MEMORY_AND_DISK)
```

# RDD storage level (2)

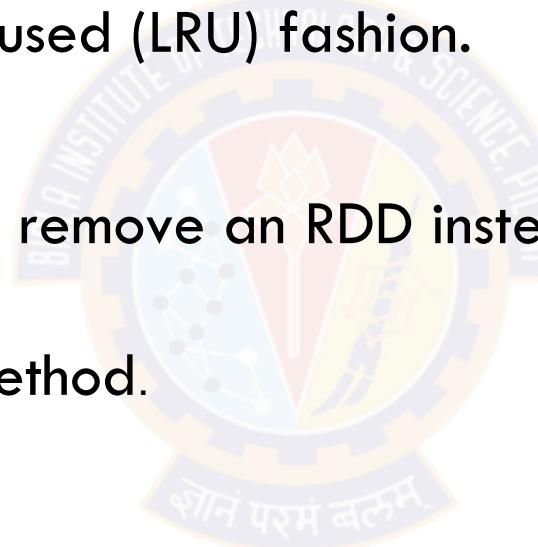
Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.

# Which Storage Level to Choose?

- Spark's storage levels are meant to provide different trade-offs between memory usage and CPU efficiency.
- Recommended process
  - ✓ If your RDDs fit comfortably with the default storage level (MEMORY\_ONLY), leave them that way.
    - Most CPU-efficient option, allowing operations on the RDDs to run as fast as possible
  - ✓ If not, try using MEMORY\_ONLY\_SER and selecting a fast serialization library to make the objects much more space-efficient, but still reasonably fast to access. (Java and Scala)
  - ✓ Don't spill to disk unless the functions that computed your datasets are expensive, or they filter a large amount of the data
  - ✓ Use the replicated storage levels if you want fast fault recovery (e.g. if using Spark to serve requests from a web application)
    - All the storage levels provide full fault tolerance by recomputing lost data, but the replicated ones let you continue running tasks on the RDD without waiting to recompute a lost partition.

# Removing Data from RDD

- Spark automatically monitors cache usage on each node and drops out old data partitions in a least-recently-used (LRU) fashion.
- If you would like to manually remove an RDD instead of waiting for it to fall out of the cache
  - ✓ use the `RDD.unpersist()` method.



# Topics for today

Spark Performance

## Spark MLlib

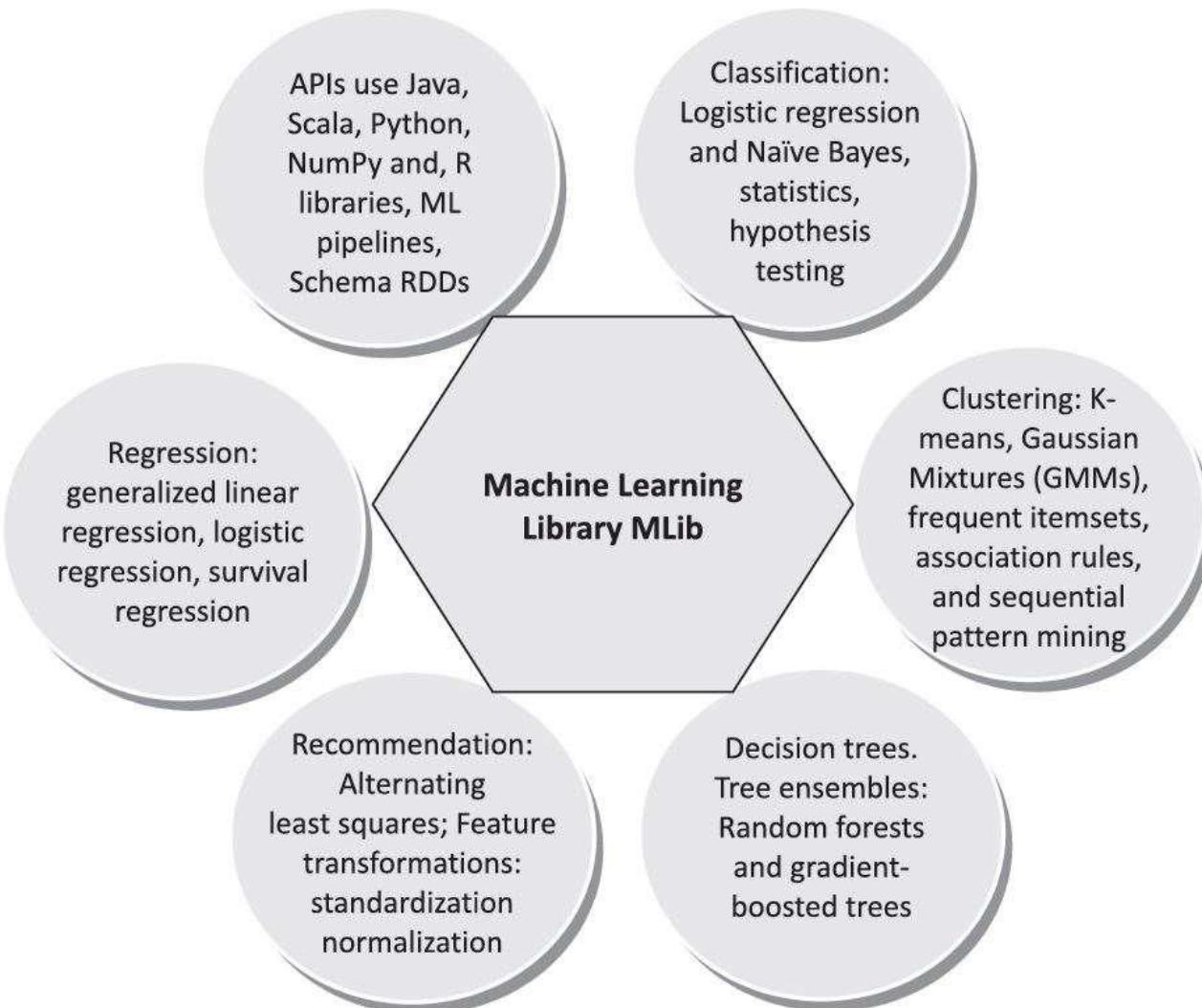
- ✓ Regression
- ✓ Classification
- ✓ Clustering
- ✓ Collaborative filtering



# What is MLLib ?

- MLLib is for **large scale machine learning** with a built-in library of machine learning algorithms
- MLLib allows for preprocessing, munging, training of models, and making predictions at scale on data.
- MLLib is a package, built on and included in Spark, that provides interfaces for
  - ✓ gathering and cleaning data,
  - ✓ feature engineering and feature selection,
  - ✓ training and tuning large scale supervised and unsupervised machine learning models,
  - ✓ and using those models in production.

# Main Usages of Mlib



# Spark MLlib

- MLlib is Spark's machine learning (ML) library.
- Makes practical machine learning scalable and easy.
- At a high level, it provides tools such as:
  - ✓ ML Algorithms: regression, classification, clustering, and collaborative filtering
  - ✓ Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - ✓ Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
  - ✓ Persistence: saving and load algorithms, models, and Pipelines
  - ✓ Utilities: linear algebra, statistics, data handling, etc.
- Primary Machine Learning API for Spark is now DataFrame-based API in `spark.ml` package.
- DataFrame-based API for MLlib provides a uniform API across ML algorithms for multiple languages.
- MLlib RDD-based API (`spark.mllib`) is now in maintenance mode.
- **Spark ML** is occasionally used to refer to the MLlib DataFrame-based API
- Refer: <https://spark.apache.org/docs/latest/ml-guide.html>

# MLlib - Data Types

- ✓ MLlib supports local vectors and matrices stored on a single machine, as well as distributed matrices backed by one or more RDDs.
- ✓ Local vectors and local matrices are simple data models that serve as public interfaces.
  - [Local vector](#)
  - [Labeled point](#)
  - [Local matrix](#)
  - [Distributed matrix](#)
    - [RowMatrix](#)
    - [IndexedRowMatrix](#)
    - [CoordinateMatrix](#)



[Data Types - MLlib - Spark 1.2.1 Documentation \(apache.org\)](#)

# Sample data for Machine Learning

Folder in spark installation - /spark-3.3.1-bin-hadoop3/data/mllib

- ✓ sample\_kmeans\_data.txt
- ✓ sample\_lda\_data.txt
- ✓ sample\_lda\_libsvm\_data.txt
- ✓ kmeans\_data.txt
- ✓ sample\_libsvm\_data.txt
- ✓ pagerank\_data.txt
- ✓ sample\_linear\_regression\_data.txt
- ✓ sample\_movielens\_data.txt
- ✓ sample\_multiclass\_classification\_data.txt
- ✓ sample\_binary\_classification\_data.txt
- ✓ sample\_svm\_data.txt
- ✓ streaming\_kmeans\_data\_test.txt
- ✓ sample\_isotonic\_regression\_libsvm\_data.txt



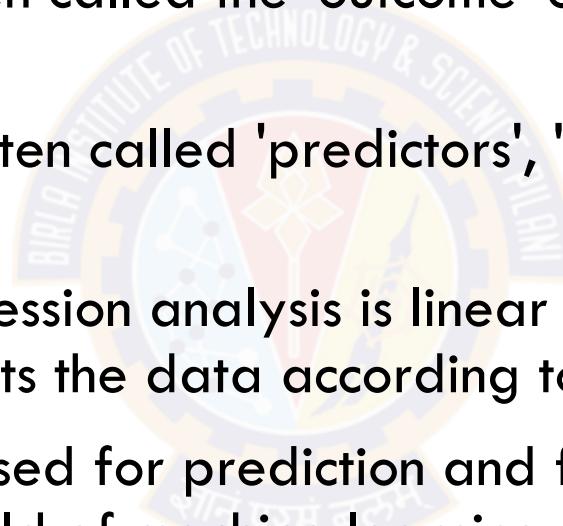
# Topics for today

- Spark Performance
- Spark MLlib
  - ✓ Regression
  - ✓ Classification
  - ✓ Clustering
  - ✓ Collaborative filtering



# Regression Analysis

- Regression analysis is a set of statistical processes for estimating the relationships between a dependent variable and one or more independent variables
- The dependent variable is often called the 'outcome' or 'response' variable, or a 'label' in machine learning parlance
- The independent variable is often called 'predictors', 'covariates', 'explanatory variables' or 'features'.
- The most common form of regression analysis is linear regression, in which one finds the straight line that most closely fits the data according to a specific mathematical criterion.
- Regression analysis is widely used for prediction and forecasting, where its usage has substantial overlap with the field of machine learning.
- Requires many techniques for modeling and performing the analysis using multiple variables
- Facilitates prediction of future values of dependent variables



# Linear and Non-linear Regression

## Linear Regression

Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a “least squares” method to discover the best-fit line for a set of paired data. You then estimate the value of Y (dependent variable) from X (independent variable).

$$Y = a_0 + a_1.X$$

Linear Regression is used to predict the relationship between two variables. The variable that needs to be predicted is known as the dependent variable and the variables that are used to predict the value of the dependent variable are known as independent variables.

## Non Linear Regression

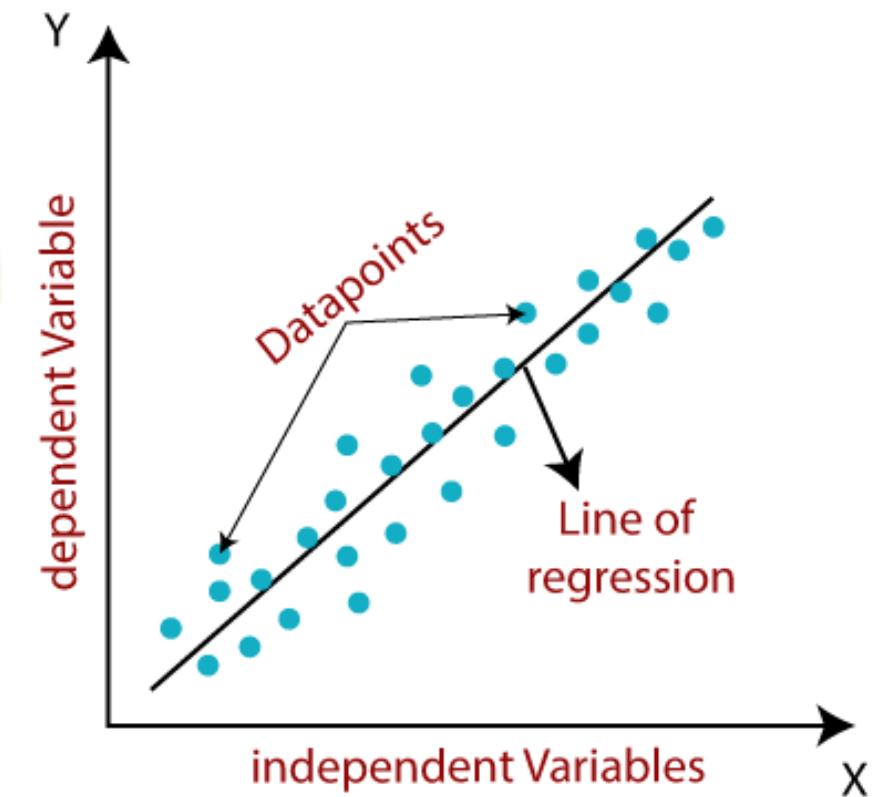
Non-Linear regression is a type of polynomial regression.

It is a method to model a non-linear relationship between the dependent (y) and independent (x) variables.

$$y = a_0 + a_1.x^1 + a_2.x^2 + a_3.x^3 \dots\dots,$$

# Linear regression by Machine Learning

- Supervised learning
- The purpose of linear regression is to come up with an equation of a line that fits through a cluster of points with minimal amount of deviation from the line
- The best fitting line is called the regression line
- Deviation of the points from the line is called an 'error'
- Once this regression equation (line) is obtained, the dependent variable can be predicted from independent predictor variable
- Training and Prediction
  - ✓ Use a training data set to learn the function
  - ✓ Use new data to predict outcome from features
  - ✓ Can extrapolate once it learns a function



# Multiple Linear Regression

- Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.
- The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables.

✓  $y = a + c_1 \cdot x_1 + c_2 \cdot x_2 + \dots + c_n \cdot x_n$

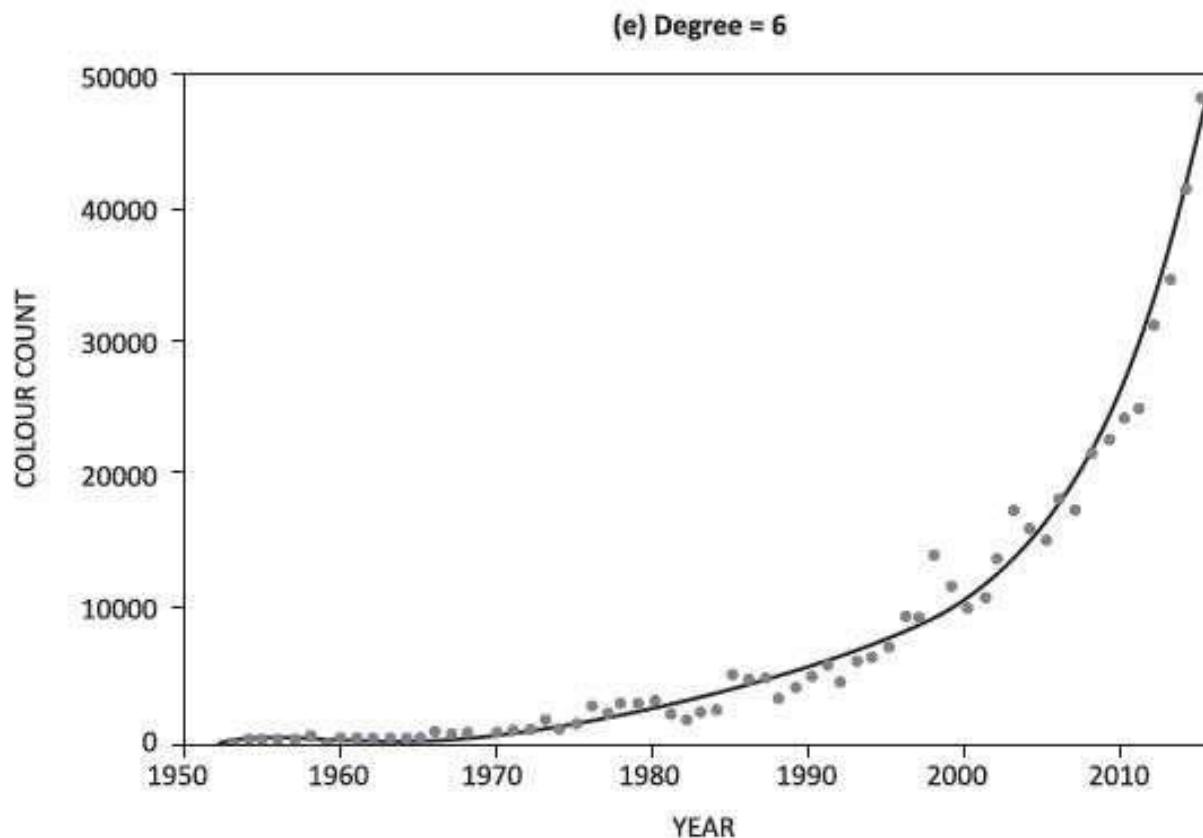
✓ Independent variable  $x_1, x_2, \dots, x_n$ , on which value of  $y$  depends

✓ Coefficients  $c_1, c_2, \dots, c_n$  are also called weights of  $x_1, x_2, \dots, x_n$

Assumptions:

1. There is a **linear relationship** between the dependent variable and the independent variables
2. The independent variables are not too highly **correlated** with each other

# Polynomial Regression



```
regPoly.py:54: RankWarning: Polyfit may be poorly conditioned  
coefs = np.polyfit (x, y, 6)  
Colour Count predicted in 2020 = 76463
```

```
regPoly.py:54: RankWarning: Polyfit may be poorly conditioned  
coefs = np.polyfit (x, y, 6)  
Colour Count predicted in 2020 = 76463
```

# Linear Regression by calculator

Input data:

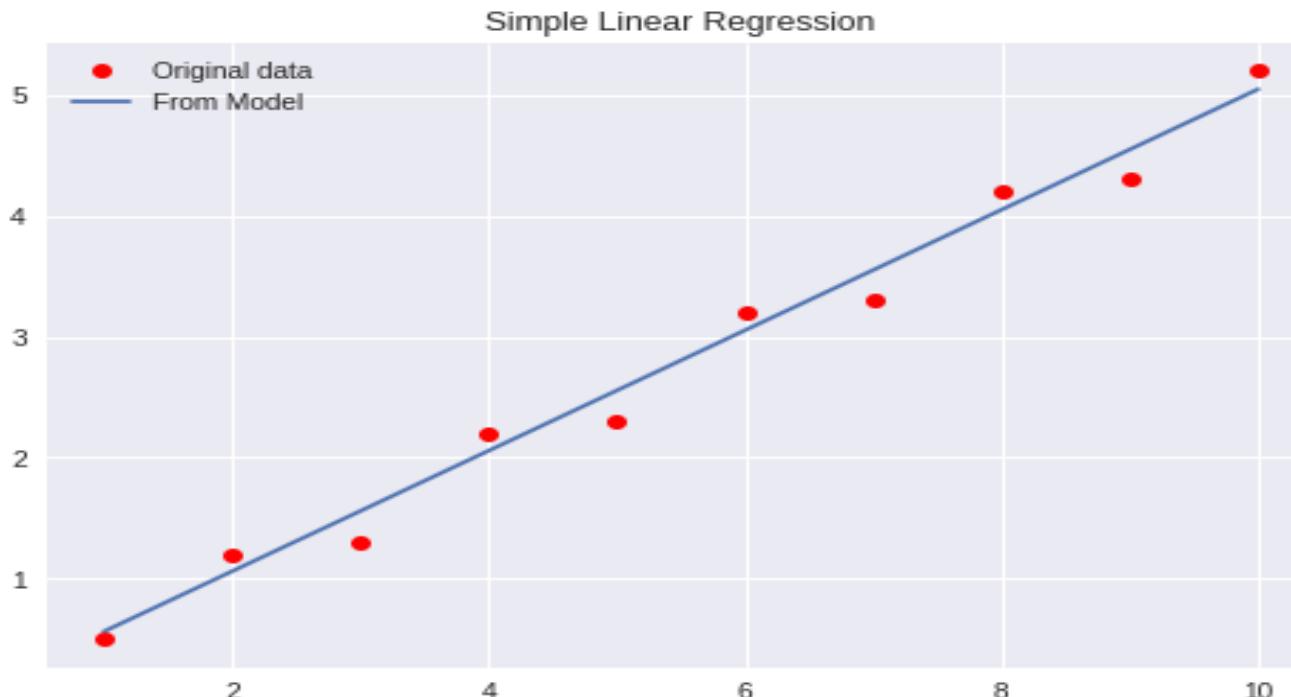
$x = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]$

$y = [0.5, 1.2, 1.3, 2.2, 2.3, 3.2, 3.3, 4.2, 4.3, 5.2]$

Calculated results:

$a$  (bias,intercept) = 0.0133

$b$  (weight,slope) = 0.5012



[Linear Regression Calculator](#)

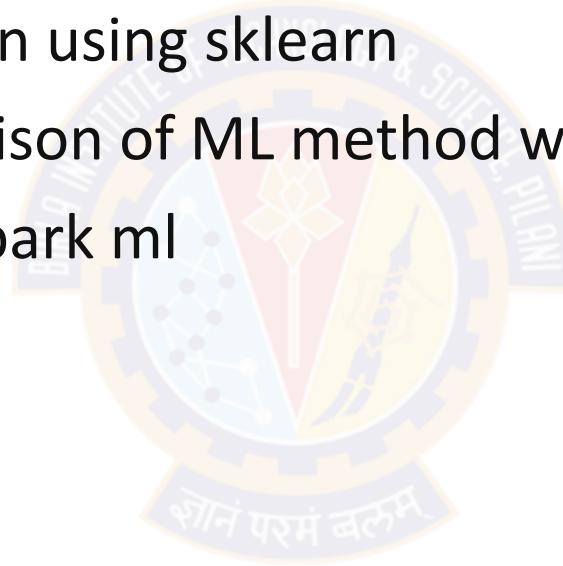
# Regression algorithms in Spark

- Linear Regression
- Generalized linear regression
- Decision tree algorithm
- Random forest regression
- Gradient-boosted tree regression
- XGBoost regression
- Survival regression
- Isotonic regression



# Linear Regression Handson

- Linear regression in python using sklearn
- Linear Regression comparison of ML method with manual method
- Linear Regression using Spark ml



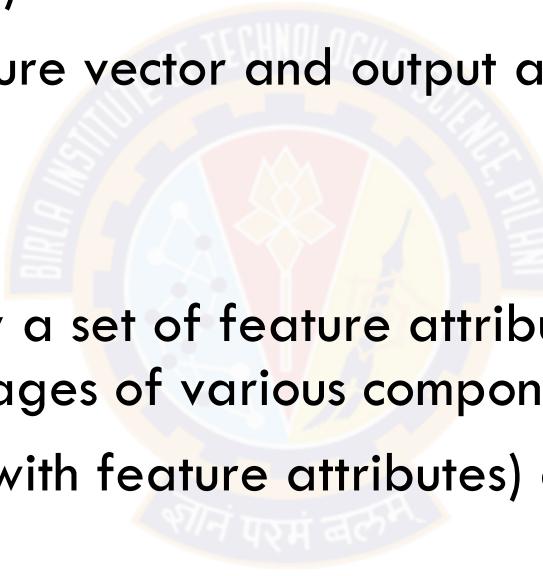
# Topics for today

- Spark Performance
- Spark MLlib
  - ✓ Regression
  - ✓ **Classification**
  - ✓ Clustering
  - ✓ Collaborative filtering



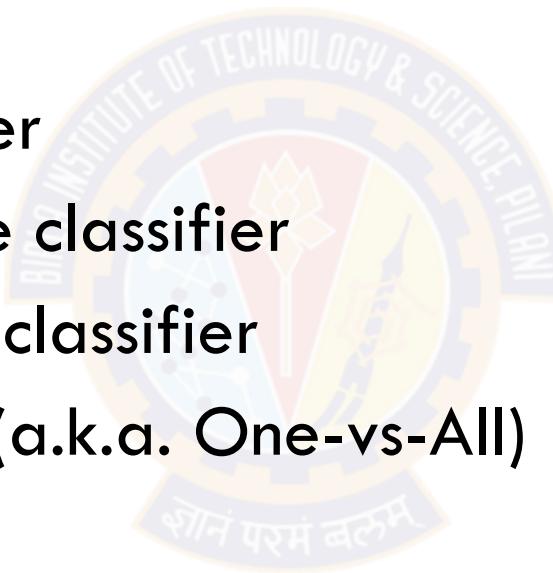
# Classification

- Supervised learning approach
- Train a classifier with labelled data, i.e. with input as a set of features (independent variables) and a label (dependent variable)
- Test the model with an input feature vector and output a label / class
- Similar algorithms as regression
- E.g.
  - ✓ Wine quality is measured by a set of feature attributes, e.g. alcohol content, acidity level, fermentation period, percentages of various components etc.
  - ✓ Train with a set of samples (with feature attributes) and add quality label for each sample
  - ✓ Given a new sample with a set of features predict the quality



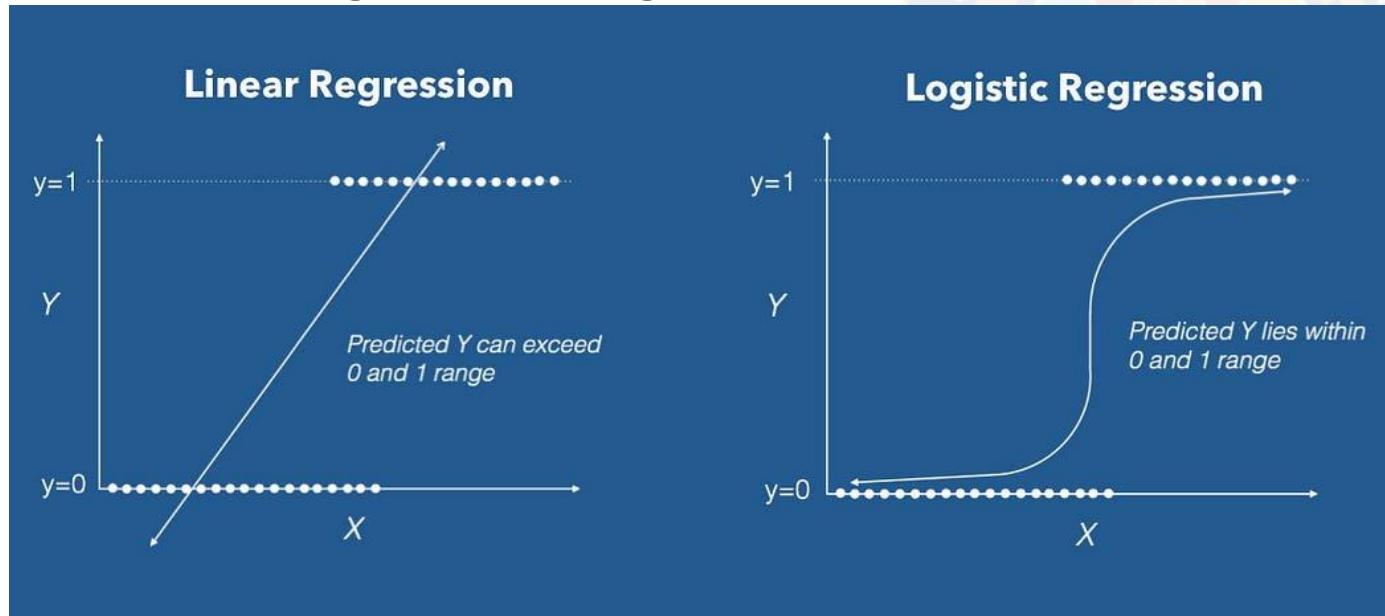
# Common Classifiers

- Logistic regression
- Decision tree classifier
- Random forest classifier
- Gradient-boosted tree classifier
- Multilayer perceptron classifier
- One-vs-Rest classifier (a.k.a. One-vs-All)



# Logistic Regression (Classification)

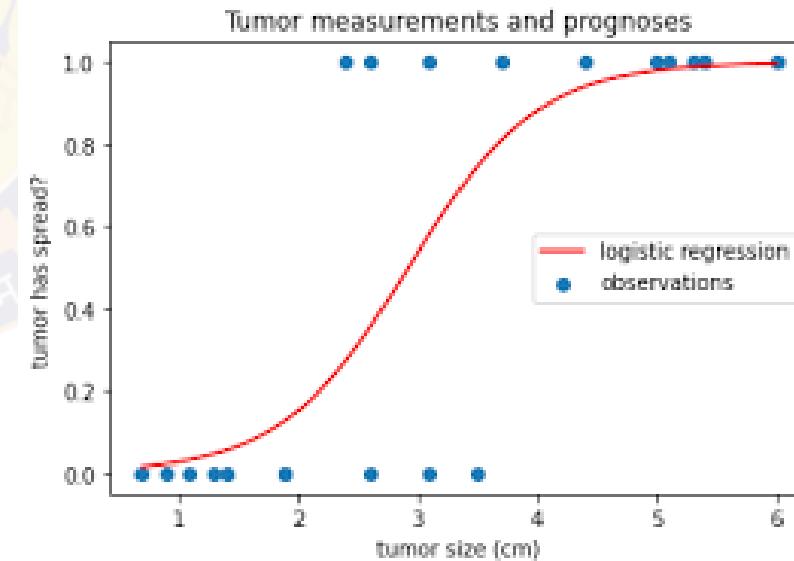
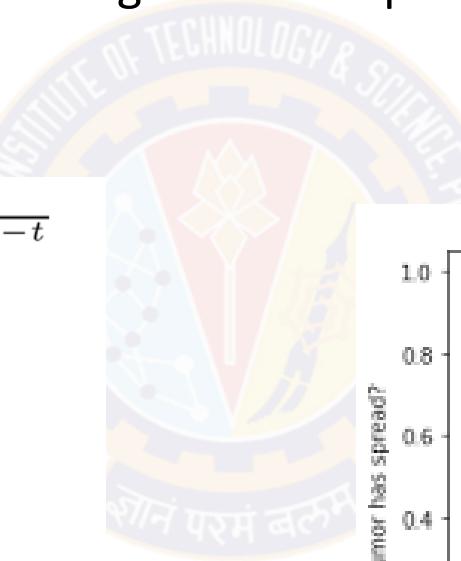
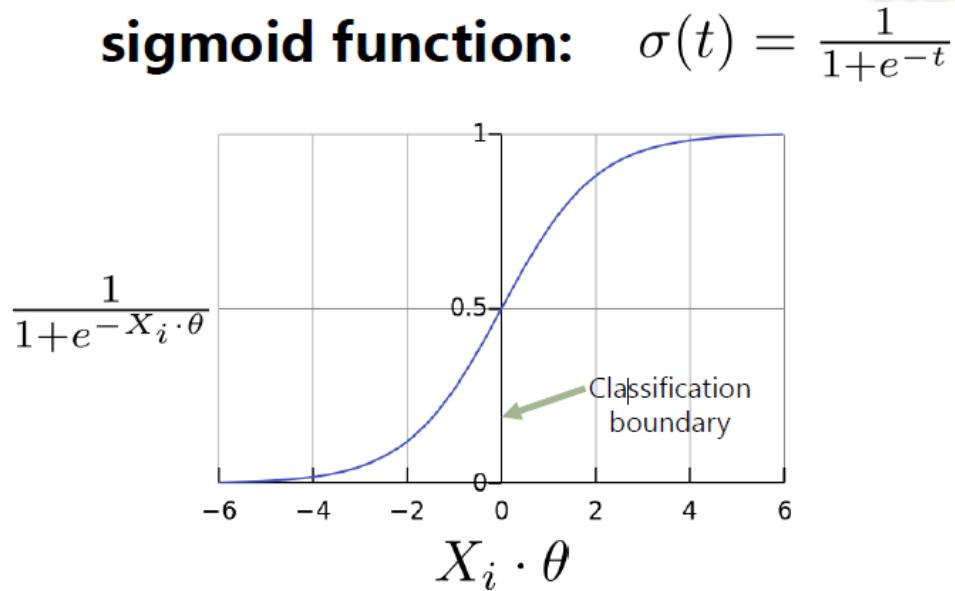
- ❑ Logistic Regression is a widely used statistical method for modeling the relationship between a binary outcome and one or more explanatory variables
- ❑ Logistic regression is often used in binary classification problems
- ❑ Examples:
  - ❑ Email spam or not spam
  - ❑ Online transactions Fraud or not Fraud
  - ❑ Tumor Malignant or Benign.



We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.

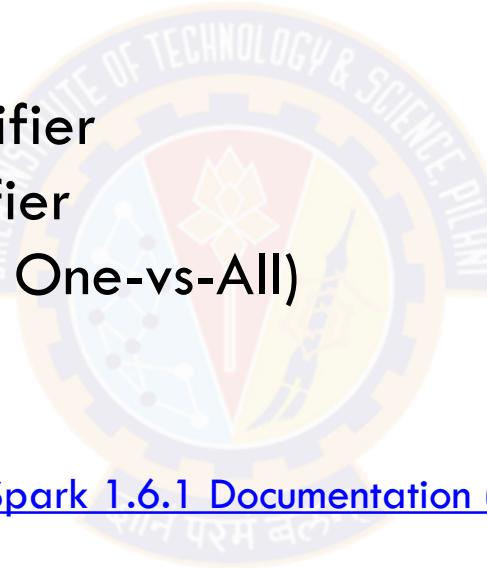
# Logistic Regression – Sigmoid function

- A model that generates a probability for each possible discrete label value in classification problems by applying a sigmoid function to a linear prediction.
- Sigmoid function maps logistic or multinomial regression output (log odds) to probabilities, returning a value between 0 and 1.



# Handson with Classifiers

- Logistic regression
- Decision tree classifier
- Random forest classifier
- Gradient-boosted tree classifier
- Multilayer perceptron classifier
- One-vs-Rest classifier (a.k.a. One-vs-All)



[Classification and regression - spark.ml - Spark 1.6.1 Documentation \(apache.org\)](#)

# Topics for today

- Spark Performance
- Spark MLlib
  - ✓ Regression
  - ✓ Classification
  - ✓ Clustering
  - ✓ Collaborative filtering



# What is Clustering

- Clustering is the process of grouping similar objects together.
- One can use clustering algorithms to segment data as in classification algorithms.
- Classification models are used to segment data based on previously defined classes that are mentioned in the target, whereas clustering models do not use any target.
- Clustering is the most popular version of unsupervised learning.
- In unsupervised learning, the goal is to identify patterns or structures in the data without any prior knowledge of what to expect.

# Clustering

- An unsupervised machine learning method
  - ✓ Items are not labelled
- Group set of items into clusters, i.e. learn the labels automatically
- Why isn't data labelled ?
  - ✓ Too expensive, e.g. grouping search results
  - ✓ Not known in advance, e.g. market segmentation, where an algorithm needs to find it
- Can include clustering to label data as part of a classification process
- E.g.: Spam filter, Fake news detection
  - ✓ Use email header, sender, specific content to cluster messages, articles etc.
  - ✓ Users are sometimes asked to label potential SPAM

# Use Case of Clustering

- Suppose you are the head of a retail store and wish to understand the preferences of your customers.
- Can you look at the details of each customer and devise a unique business strategy for each one of them?
- What you can do is cluster all of your customers into, say 5 groups based on their purchasing habits and use a separate strategy for each group.

# Clustering categories

You can use clustering when you want to explore data. Clustering algorithms are mainly used for natural groupings. There are different categories of clustering.

- **Hierarchical:** Hierarchical cluster identifies the cluster within the cluster. A news article group can further have other groups such as business, politics, and sports in which each group can still have subgroups. For example, inside sports news there could be news on baseball sport, news on basketball sport, and so on.
- **Partitional:** Partitional creates a fixed number of clusters. The K-means clustering algorithm belongs to this category. Let us study the K-mean clustering algorithm in detail.

# K-Means Clustering

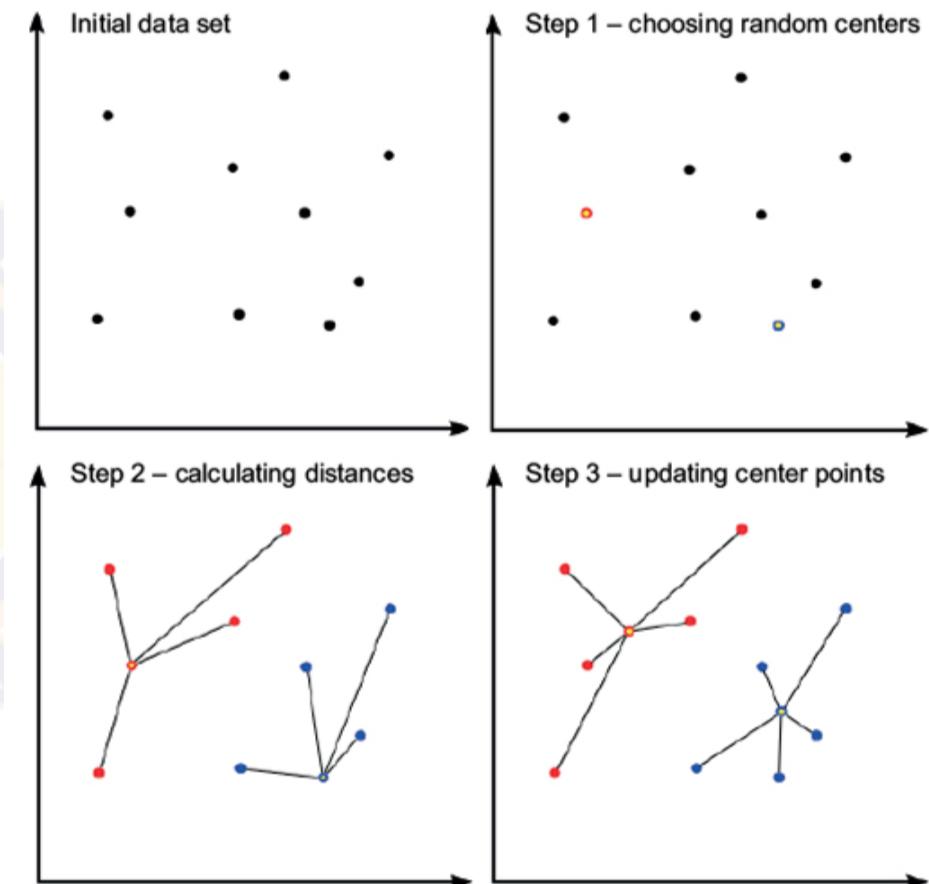
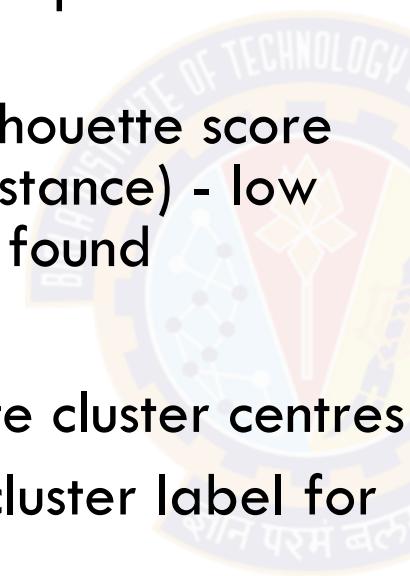
The steps involved in K-means algorithm are as follows:

1. Choose the final required number of clusters.
2. Examine each element in the population and assign it to one of the clusters depending on the minimum distance.
3. Each time a new element is added to the cluster, the centroid's position is recalculated. This process is performed until all the elements are grouped into the required number of clusters.

**Centroid:** It is a point whose parameter values are the mean of the parameter values of all the points in the cluster.

# K-Means clustering method

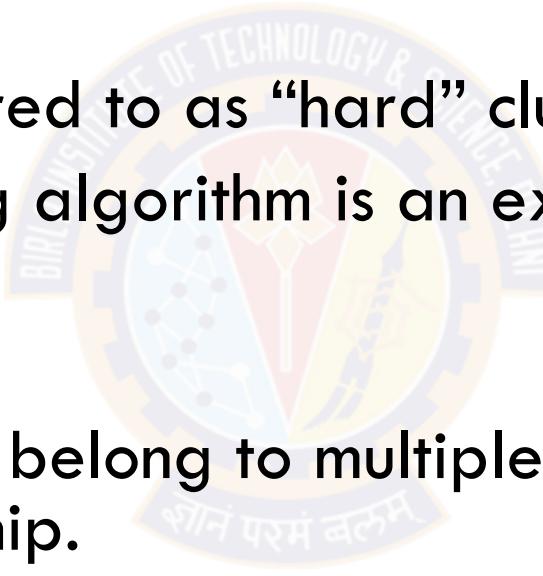
- Iteratively choose new centroids till convergence
- Distance computation can run in parallel in each iteration
- How good is the clustering: Silhouette score (mean squared intra cluster distance) - low means tightly coupled clusters found
- Prediction on new data:
  - ✓ Use training data to create cluster centres
  - ✓ Use new data to predict cluster label for each data item
- Other methods in Spark: Gaussian Mixture Model, Power Iteration Clustering etc.



[K-means clustering, starting with 4 left-most points \(shabal.in\)](#)

# Types of partitional clusters

- Exclusive clustering
  - ✓ A form of grouping that requires a data point to exist only in one cluster.
  - ✓ This can also be referred to as “hard” clustering.
  - ✓ The K-means clustering algorithm is an example of exclusive clustering
- Overlapping clustering
  - ✓ Allows data points to belong to multiple clusters with separate degrees of membership.
  - ✓ “Soft” or fuzzy k-means clustering is an example of overlapping clustering.



# Handson with K-Means clustering

<https://spark.apache.org/docs/latest/ml-clustering.html>



# Topics for today

- Spark Performance
- Spark MLlib
  - ✓ Regression
  - ✓ Classification
  - ✓ Clustering
  - ✓ **Collaborative filtering**



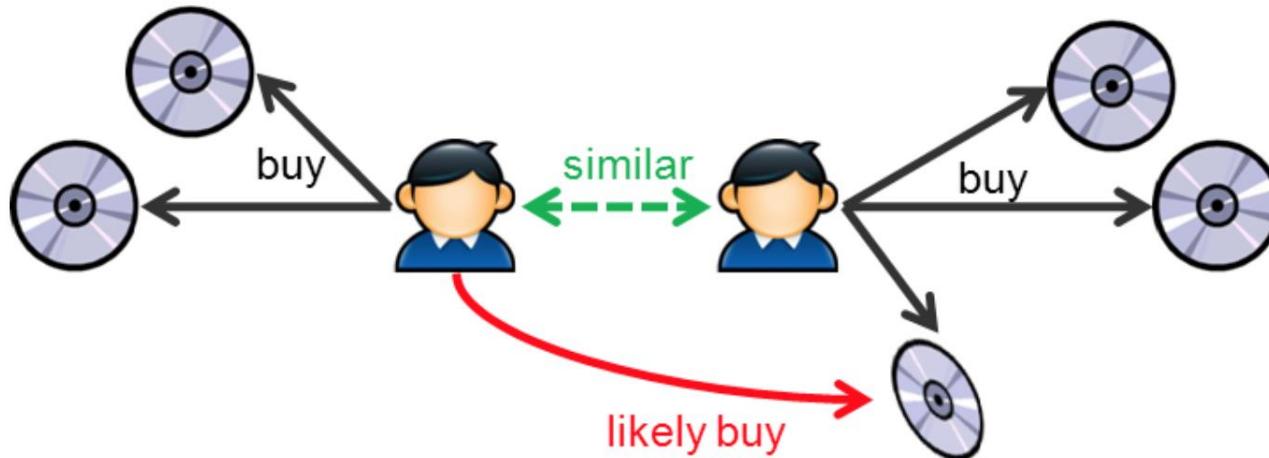
# Recommendation systems

- Try to predict the preference of a user based on past behaviour
  - ✓ Set of items (X)
  - ✓ Set of users (Y)
  - ✓ Learn a function based on past interactions that predicts the likeliness of X to Y
  - ✓ e.g. movie, songs, shopping recommendations
- Broadly 2 types
  - ✓ Content based filtering: uses only attributes of items
    - e.g. Anyone has heard songs a, b, c then likely may want to hear u and v next <- same recommendation for all users
    - Typically items must have multiple attributes to create relationships, e.g. movie genre, actors, director, ..
  - ✓ Collaborative filtering: utilizes user interaction behaviour in addition to item attributes
    - e.g. Specific user's interactions with songs matters to understand who are similar users what else do they listen to so that next song recommendation can be generated
    - Item attributes are less important
- A hybrid approach is better because Collaborative Filtering has a cold start issue - enough user interactions have to be there for recommendation

# Collaborative Filtering

- Collaborative filtering can be defined as **Social Navigation**. We say that human beings are social animals owing to their tendency to follow other people's advice or judgment when looking for information or buying products. This is in a way similar to an ant looking for food trudging behind other ants.
- Similar to asking friends or people with “similar” preferences
- Most collaborative filtering systems use a “similarity index” wrt active user
  - ✓ Set of “similar user” preferences are aggregated
- What matters is the relationship of users to items rather than only among items
  - ✓ Content based filtering is the latter approach
- So similarity in items is determined by similarity of preferences of those items by the users who have rated both items
- Core technique is to measure similarity or correlation
- Unsupervised technique

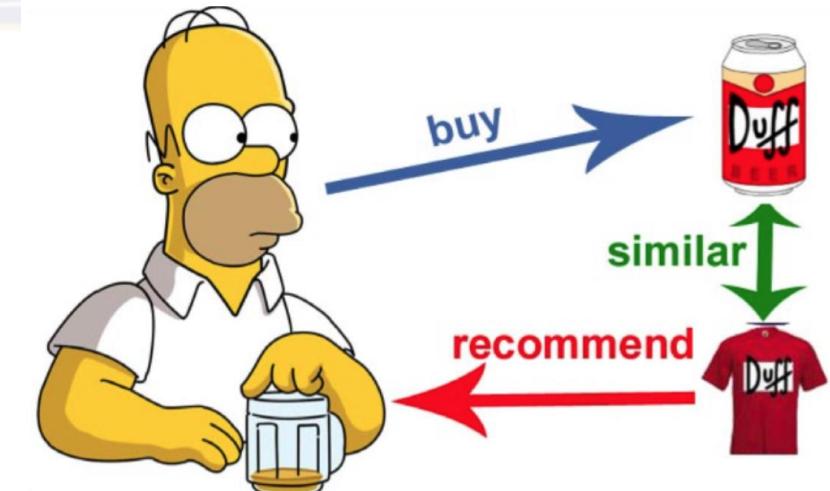
# Nearest-neighborhood approaches



Source: <https://dzone.com/articles/recommendation-engine-models>

Item-based: 2 items are similar because same user has given similar ratings

User-based: 2 users are similar because they have similar ratings on same items



Source: <https://medium.com/tiket-com-dev-team/build-recommendation-engine-using-graph-cbd6d8732e46>

# Handson with Collaborative filtering

- In this example, we load rating data.
- Each row of rating data consists of a user, a product and a rating.
- We use the default [ALS.train\(\)](#) method which assumes ratings are explicit.
- We evaluate the recommendation model by measuring the Mean Squared Error of rating prediction

1, 1, 5.0

1, 2, 1.0

1, 3, 5.0

1, 4, 1.0

2, 1, 5.0

2, 2, 1.0

.....



## Collaborative Filtering - RDD-based API

<https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>

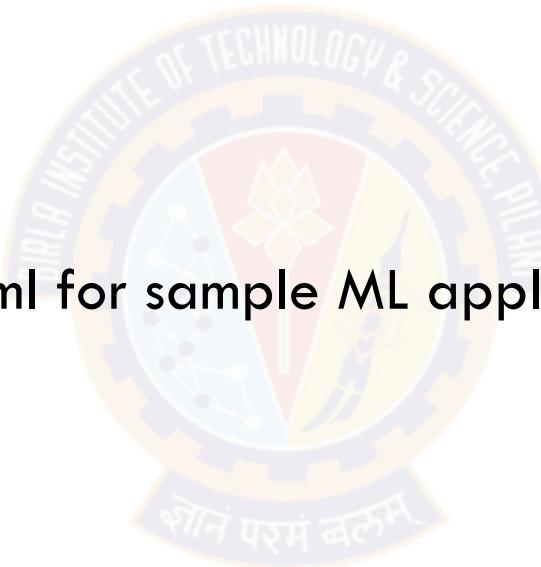
# Additional Reading

## The Data Scientist's Guide to Apache Spark - Databricks

- <https://www.analyticsvidhya.com/blog/2021/01/a-quick-overview-of-regression-algorithms-in-machine-learning/>
- <https://databricks.com/blog/2014/07/23/scalable-collaborative-filtering-with-spark-mllib.html>
- <https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26>

# Summary

- Basic capabilities of Spark ML
- Regression
- Classification
- Clustering
- Collaborative filtering
- Handson demo in using Spark ml for sample ML applications





**Next Session:  
Spark - Part 4**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 14: Spark -Part 4

---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

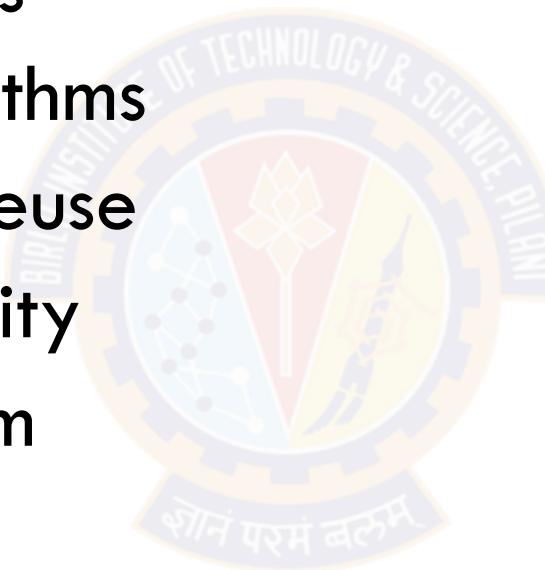
# Topics for today

- Spark ML performance
- Stream processing
- Spark Streaming
- Windowing



# MLlib: Tips and Performance Considerations

- Preparing Features
- Configuring Algorithms
- Caching RDDs to reuse
- Recognizing Sparsity
- Level of Parallelism



# Preparing Features

- Usually too much emphasis given on the algorithms but features are more critical
- **Each algorithm is only as good as the features put into it!**
- Feature preparation is the most important step to large-scale machine learning!
- Improvements into results can be obtained by
  - ✓ Adding new more informative features
  - ✓ Converting available features to suitable vector representations
- Common tips:
  - ✓ Scale your features
  - ✓ Featurize text correctly (can think of use of external libraries like NLTK)
  - ✓ Label classes correctly



# Configuring Algorithms

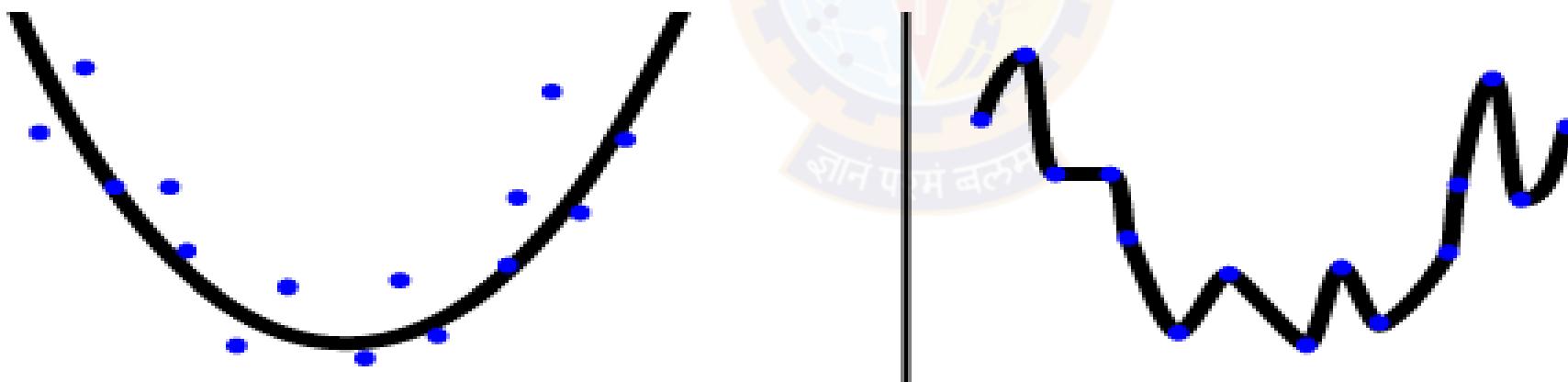
- Most algorithms perform better in terms of accuracy with a suitable regularization when its applicable and available
  - ✓ Models often learn random irrelevant behaviour from training data
  - ✓ Regularization avoid overfitting by making models simpler, penalizing large coefficients etc.
- Though default values works well but
  - ✓ Try changing the values from default to see if improves the accuracy
  - ✓ Make sure evaluating these changes on test data as well

# Use the Right Regularization technique

**Regularization: Make your Machine Learning Algorithms “Learn”, not “Memorize”**

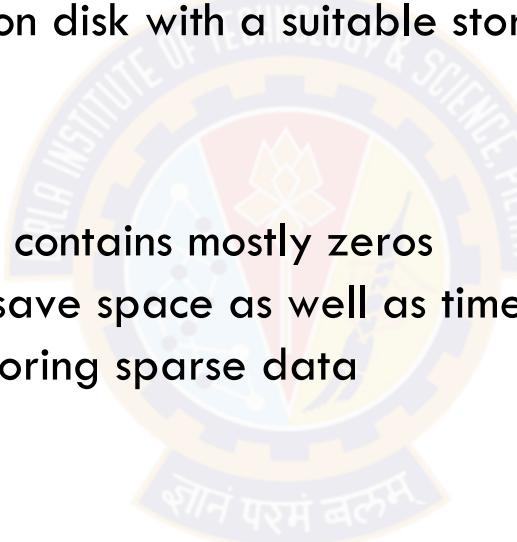
There are three main regularization techniques, namely:

- 1.Ridge Regression (L2 Norm)
- 2.Lasso (L1 Norm)
- 3.Dropout



# Other Tips

- Caching RDDs to reuse
  - ✓ Most algorithms are iterative in nature, needs to pass over data sets multiple times
  - ✓ Important to cache the dataset before passing to MLLib
  - ✓ If does not fit in memory, persist on disk with a suitable storage level
- Recognizing Sparsity
  - ✓ Check whether the feature vector contains mostly zeros
  - ✓ Storing in the sparse format can save space as well as time for processing
  - ✓ LIBSVM data format is good in storing sparse data
- Level of Parallelism
  - ✓ Have at least as many partitions in input RDD as the number of cores on cluster
  - ✓ Can specify the number of partitions while reading data
  - ✓ Can use repartition method on RDD to partition it into more pieces
  - ✓ Be careful – adding more partitions increases communication overhead!



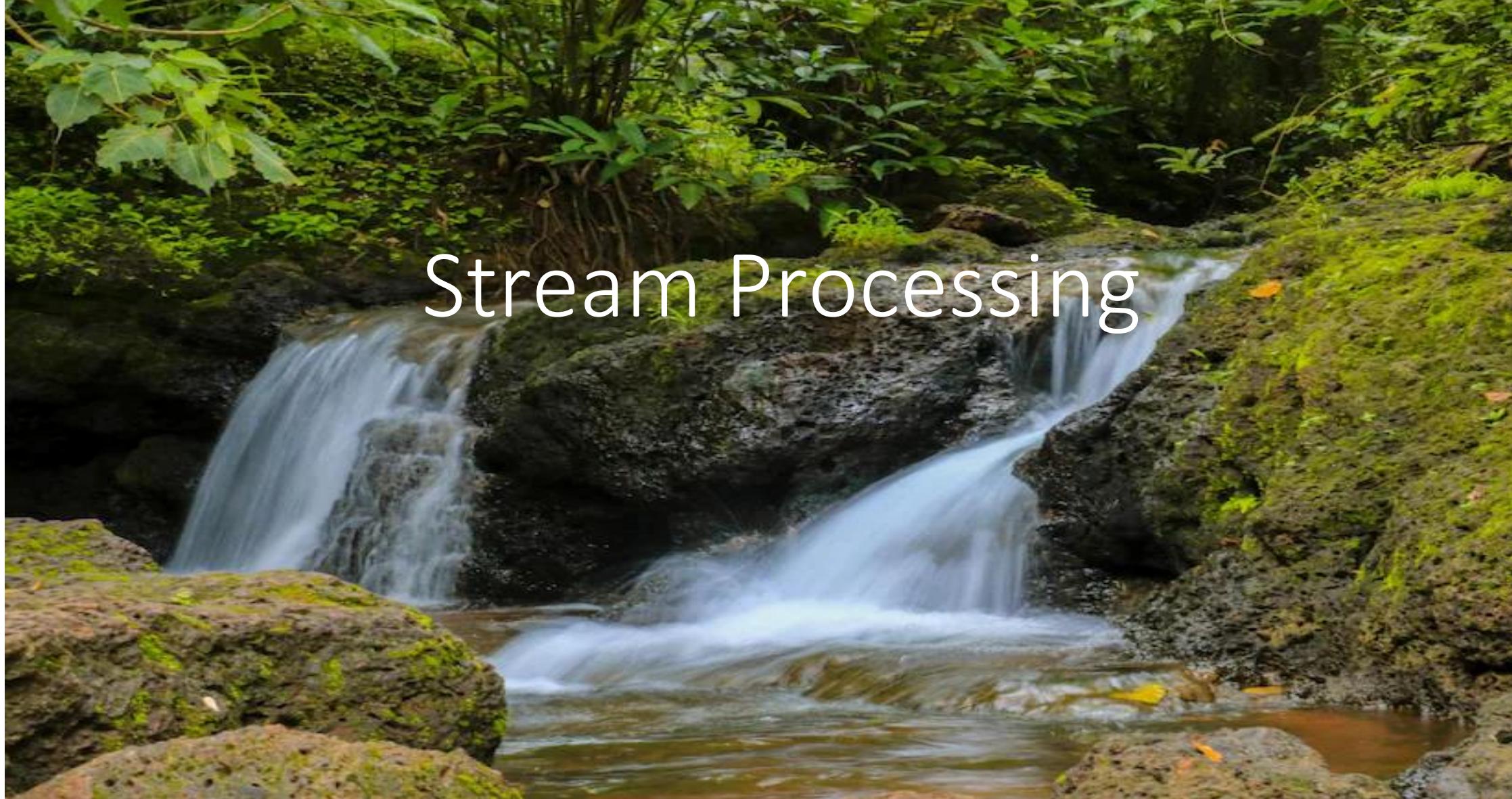
# Hardware provisioning tips

- Run closer to data: If you are using HDFS data then make it is on the same network or run Spark on same nodes, preferably using YARN.
- Storage: Use 4-8 disks per node with separate mount points with no RAID. configure spark.local.dir with these mount points. If using HDFS then use the same disks.
- Memory: Configure 8GB+ on nodes and 25% is typically taken up by OS and buffer cache.
  - ✓ How much memory is needed ? Load typical datasets you use into RDD and check memory taken up using spark UI (on `http:<driver machine>: 4040`)
  - ✓ Typically JVM memory should be < 200GB. If machine has more then run multiple executors.
- Network: With shuffle / distributed reduce operations need 10GB+ network.
- CPU cores: At least 8-16 cores per machine because typically threads will run in parallel CPU-bound once data is in memory. So CPU and network are heavily used. Observe utilization and can scale cores as well as add more nodes.

# Topics for today

- Spark ML Performance
- **Stream processing**
- Spark Streaming
- Windowing

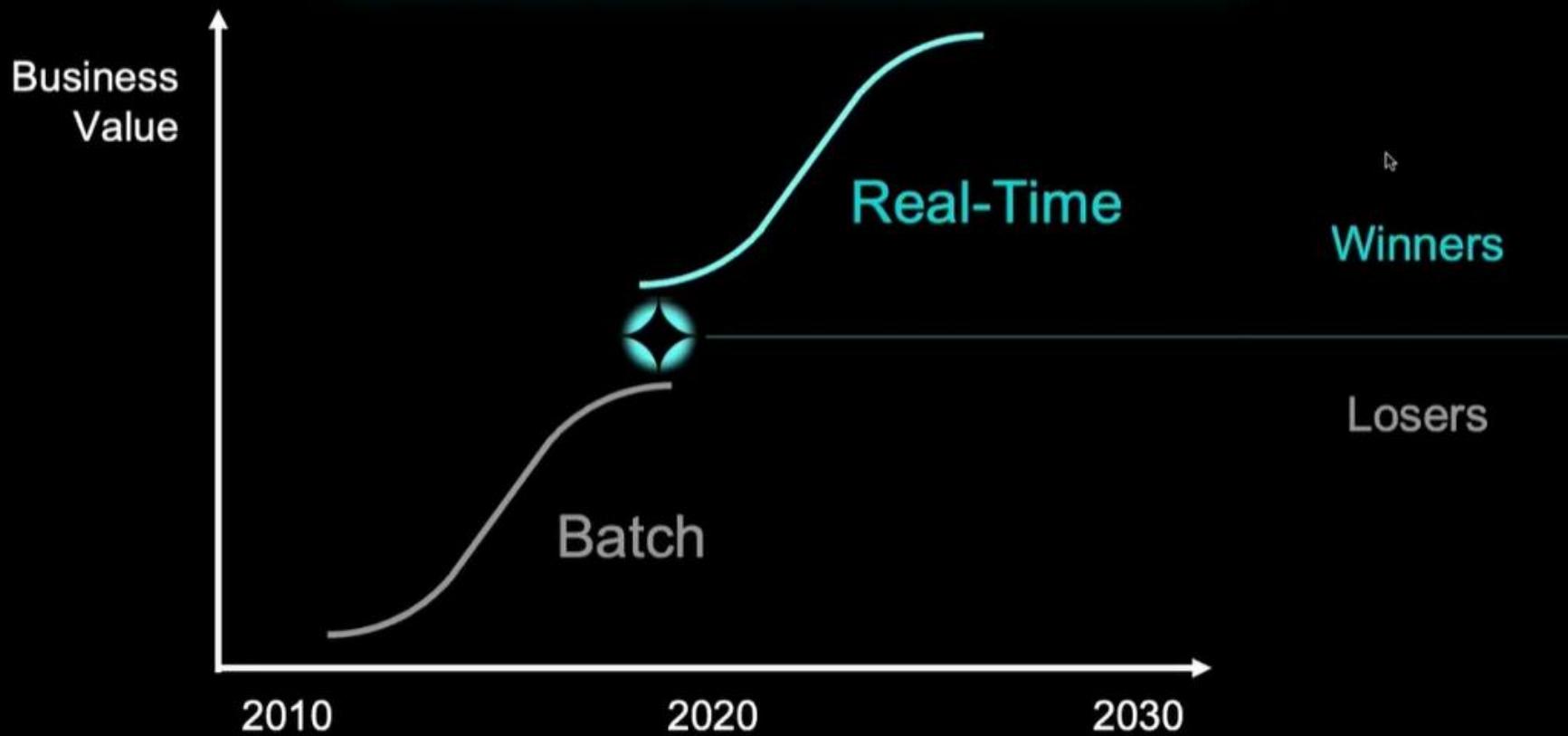




# Stream Processing

# The Real-Time Economy Is Emerging Quickly

Leveraging the Changing Data Pattern



# Stream Processing (1)

- ✓ Stream processing is a powerful technique for performing work on data that is being moved, i.e., “data in motion.”
- ✓ This type of data is often newly created and processed immediately to take advantage of real-time responsiveness.
- ✓ But it can also be any data that is continuously transmitted from one location to another.
- ✓ The unbounded (unending) characteristic of data in motion requires the use of stream processing technologies for efficient management, processing, and analysis.

# Stream Processing (2)

- ✓ Insights from data are more valuable immediately after events happen
  - (Near) real-time: from milliseconds to seconds, or minutes
- ✓ Stream processing allows faster reaction to events
  - Detecting patterns, setting alerts
- ✓ Some data is naturally unbounded (e.g. sensor data)
- ✓ Resource constraints (storage and compute)
  - Process large volumes of data arriving at high velocities
  - Retain only what is useful
- ✓ Continuous processing

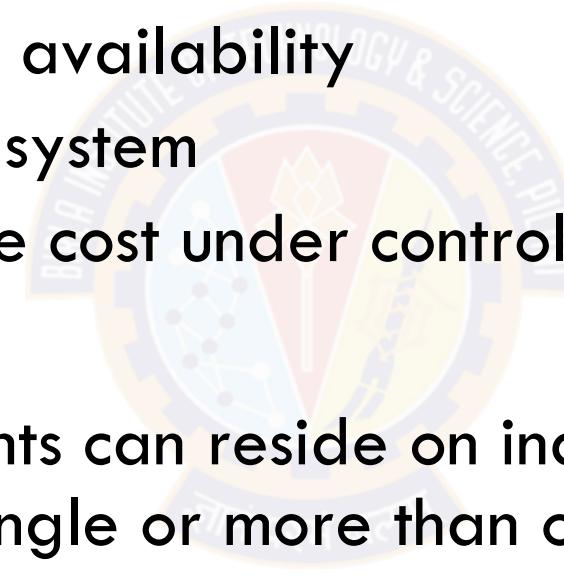
# Applications of Stream processing

- Computing
  - ✓ Log analysis,
  - ✓ Detection of DoS attacks,
  - ✓ Scaling service capacities
- Real-time monitoring
  - ✓ Fraud detection (credit cards)
  - ✓ Anomaly and Pattern detection
  - ✓ Intrusion detection (surveillance)
- Sensor data processing
  - ✓ Weather,
  - ✓ Transportation
  - ✓ Traffic
  - ✓ Patient health
- Social media
  - ✓ Trend analysis
- Industry
  - ✓ Process optimization
  - ✓ Predictive maintenance
  - ✓ Logistics
- Advertising and promotions
  - ✓ Contextualized to user behavior or geolocation
- Financial trading
  - ✓ Algorithmic trading
  - ✓ Risk analysis



# Streaming Data Systems

- Layered systems that rely on several loosely coupled systems to process continuously collected data
  - Helps in achieving high availability
  - Helps in managing the system
  - Helps in maintaining the cost under control
- All subsystems / components can reside on individual physical servers or can be co hosted on the single or more than one servers
- Not all components to be present in every system



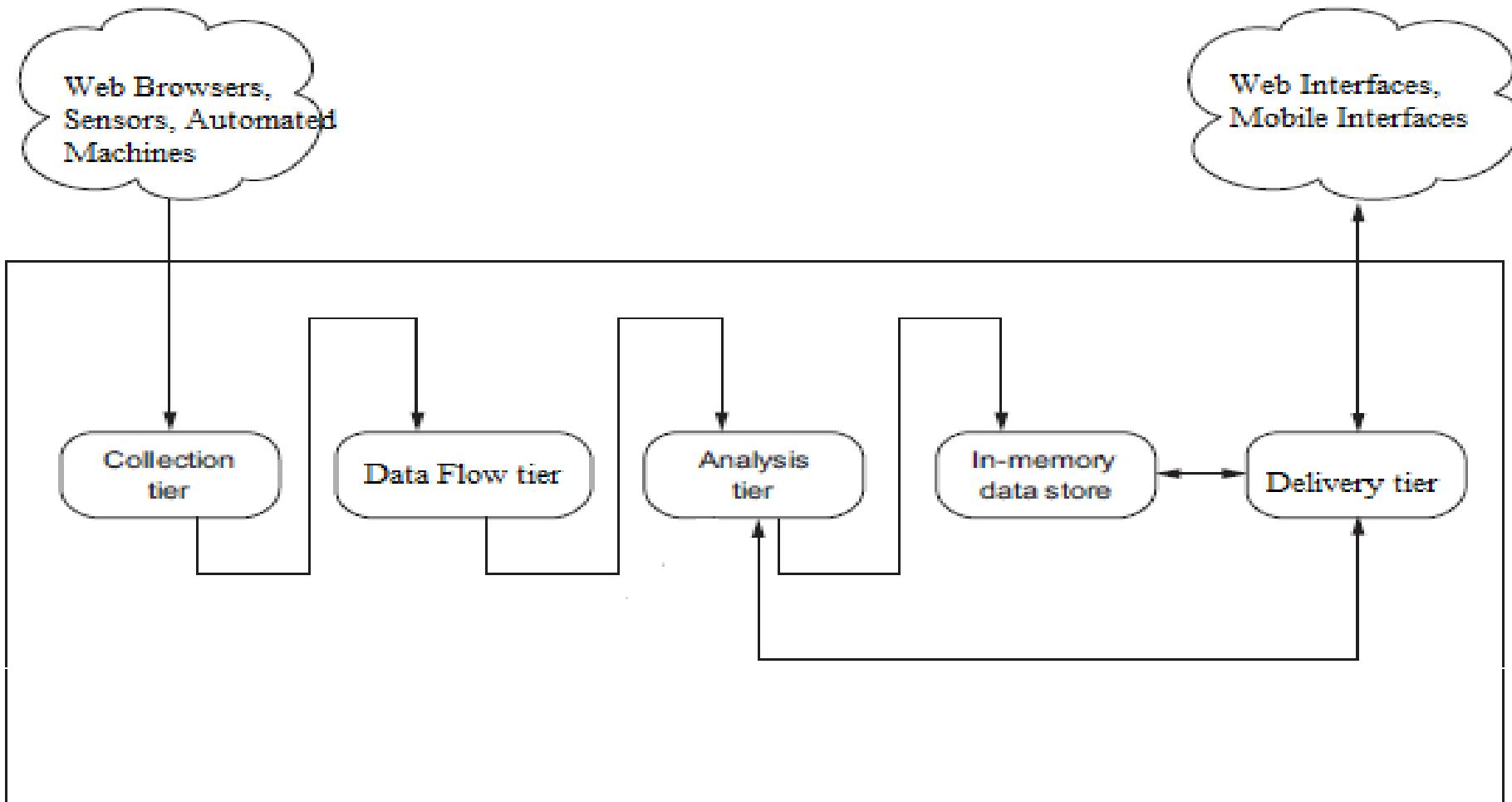
# Streaming Data System Components

## Streaming Data System Architecture Components

1. Collection
2. Data Flow
3. Processing
4. Storage
5. Delivery



# Generalized Architecture



Altered version, original concept : Andrew G. Psaltis

# Architecture Components (1)

## Collection System

- Mostly communication over TCP/IP network using HTTP
  - Websites log data was the initial days use case
  - W3C standard log data format was used
  - Newer formats like JSON, AVRO, Thrift are available now
- 
- Collection happens at specialized servers called edge servers
  - Collection process is usually application specific
  - New servers integrates directly with data flow systems
  - Old servers may or may not integrate directly with data flow systems

# Architecture Components (2)

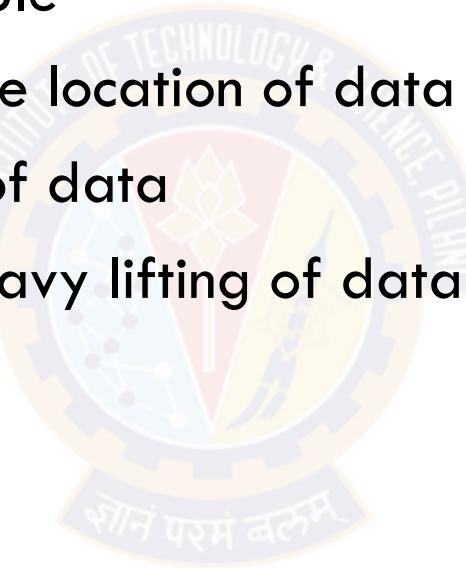
## Data Flow Tier

- Separation between collection tier and processing layer is required
  - Rates at which these systems works are different
  - What if one of system is not able to cope with another system?
- Required intermediate layer that takes responsibility of
  - accepting messages / events from collection layer
  - providing those messages / events to processing layer
- Real time interface to data layer for both producers and consumers of data
- Helps in guaranteeing the “at least once” semantics

# Architecture Components (3)

## Processing / Analytics Tier

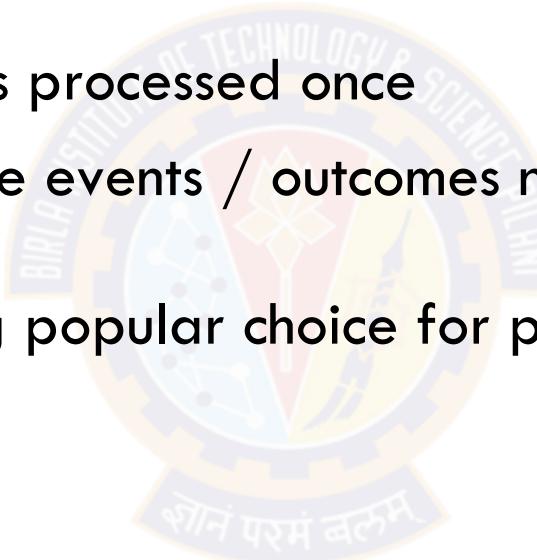
- Based on “data locality” principle
- Move the software / code to the location of data
- Rely on distributed processing of data
- Framework does most of the heavy lifting of data partitioning, job scheduling, job managing
- Available Frameworks
  - Apache Storm
  - Apache Spark (Streaming)
  - Apache Kafka Streaming etc.



# Architecture Components (4)

## Storage Tier

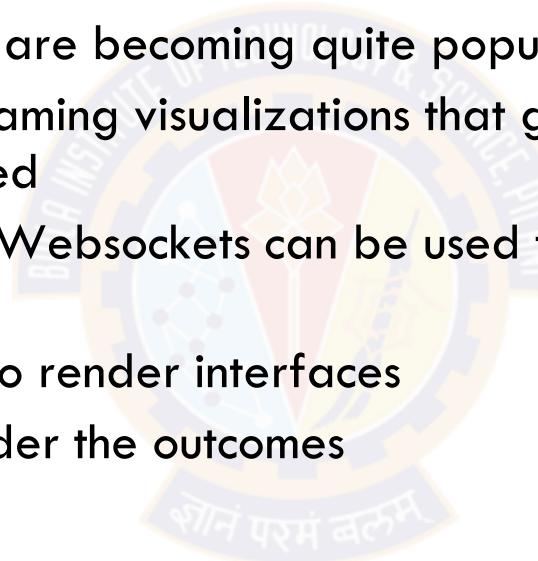
- In memory or permanent
- Usually in memory as data is processed once
- But can have use cases where events / outcomes needs to be persisted as well
- NoSQL databases becoming popular choice for permanent storage
  - MongoDB
  - Cassandra
- But usage varies as per the use case, still no database that fits all use cases



# Architecture Components (5)

## Delivery Layer

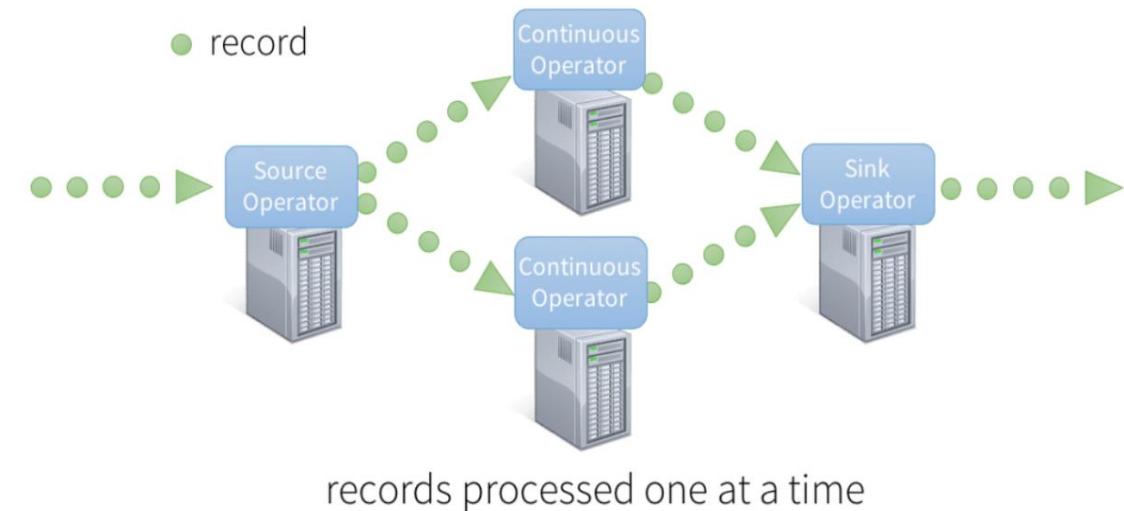
- Usually web based interface
  - Now a days mobile interfaces are becoming quite popular
  - Dashboards are built with streaming visualizations that gets continuously updated as underlying events are processed
  - HTML + CSS + Java script + Websockets can be used to create interfaces and update them
  - HTML5 elements can be used to render interfaces
  - SVG, PDF formats used to render the outcomes
- Monitoring / Alerting Use cases
- Feeding data to downstream applications



# Continuous operator model

- A simple and natural model
  - ✓ Streaming data is received from data sources (e.g. live logs, system telemetry data, IoT device data, etc.) into some data ingestion system like Apache Kafka, Amazon Kinesis, etc.
  - ✓ The data is then processed in parallel on a cluster.
  - ✓ Results are given to downstream systems like HBase, Cassandra, Kafka, etc.
  - ✓ Data is received from ingestion systems via Source operators and given as output to downstream systems via sink operators
  - ✓ Cluster has set of worker nodes, each of which runs one or more continuous operators
  - ✓ Each continuous operator processes the streaming data one record at a time and forwards the records to other operators in the pipeline

Traditional stream processing systems  
*continuous operator model*



# Continuous operator model - Challenges

- Fast Failure and Straggler Recovery
  - ✓ System must be able to quickly and automatically recover from failures and stragglers to provide results
  - ✓ Difficult to achieve due to the static allocation of continuous operators to worker nodes
- Load Balancing
  - ✓ The system needs to be able to dynamically adapt the resource allocation based on the workload
  - ✓ Uneven allocation of the processing load between the workers can cause bottlenecks
- Unification of Streaming, Batch and Interactive Workloads
  - ✓ System might have requirements to query the streaming data interactively, or to combine it with static datasets
  - ✓ Hard in continuous operator systems which are not designed for ad-hoc queries
  - ✓ This requires a single engine that can combine batch, streaming and interactive queries
- Advanced Analytics with Machine learning and SQL Queries
  - ✓ Complex workloads require continuously learning and updating data models, or even querying the streaming data with SQL queries.
  - ✓ Having a common abstraction across these analytic tasks makes the developer's job much easier.

# Main Distributed Stream processing platforms

- **Apache Flink** - open-source, unified platform for batch and stream processing.
- **Apache Storm** - one of the first streaming platforms (originated in 2010).
- **Google Cloud DataFlow** - a managed service in a Google cloud.
- **Hazelcast Platform** - open source, lightweight, and embeddable platform for batch and stream processing using a pipeline API or SQL, contains distributed in memory data structures to store operational or reference data and publish results.
- **Kafka Streams** - open-source Stream Processing Engine (SPE) integrated into Apache Kafka ecosystem.
- **ksqldb** - open source SPE that uses SQL for processing data.
- **Apache Spark Streaming** - open-source, utilizes Spark batch processing platform by dividing continuous stream to sequence of discrete micro-batches

# Topics for today

- Spark ML performance
- Stream processing
- **Spark Streaming**
- Windowing

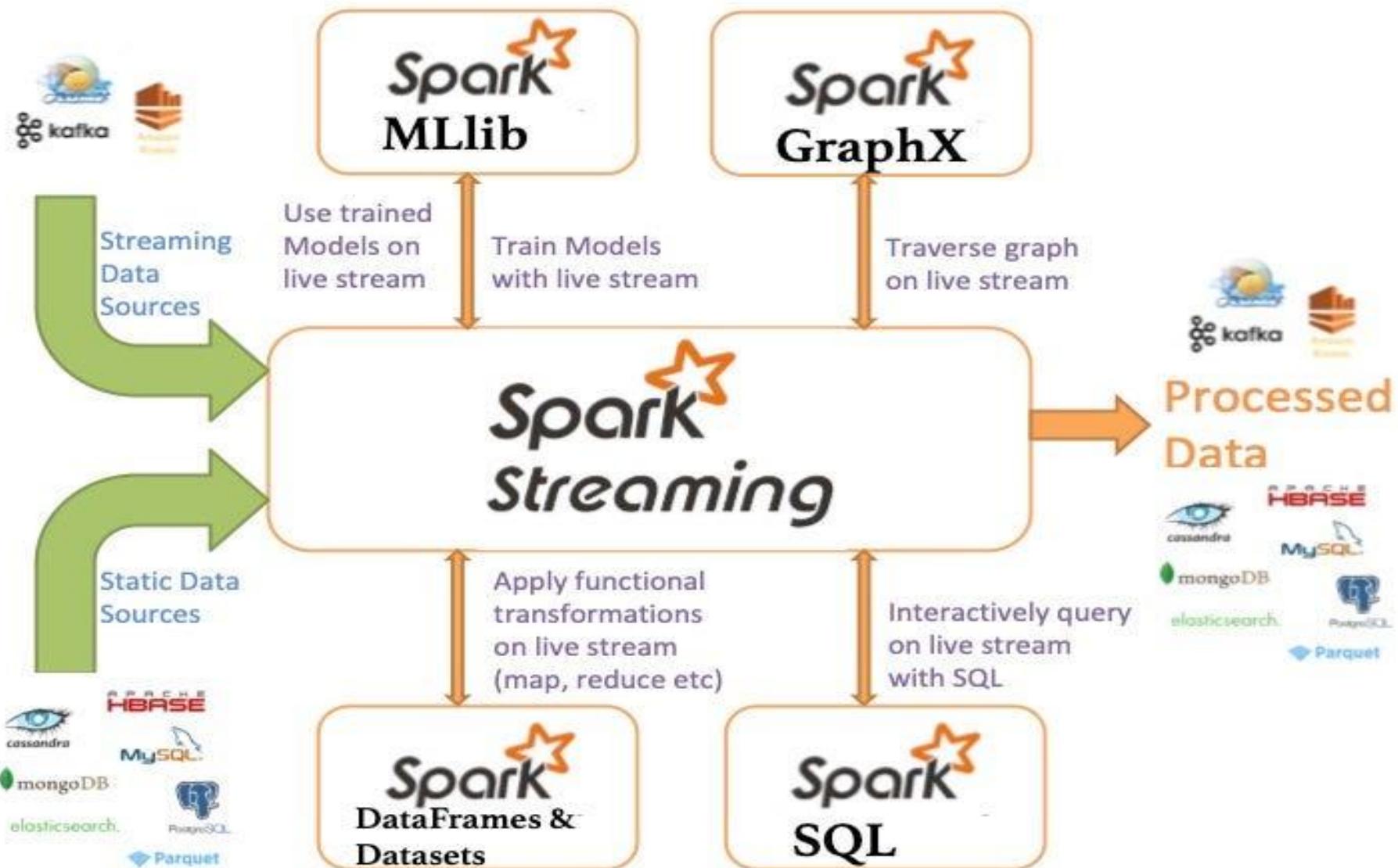


# Why Spark Streaming?

## Spark provides

- Fast recovery from failures and stragglers
- Better load balancing and resource usage
- Data from streaming sources can combine with a very large range of static data sources available through Apache Spark SQL
- Provides the unification of disparate data processing capabilities, e.g. streaming with interactive, batch etc.
  - Has a unified engine that natively supports both batch and streaming workloads
  - Makes it very easy for developers to use a single framework to satisfy all the processing needs
- Native integration with advanced libraries, such as ML, graph, SQL

# Spark Streaming



# Overview

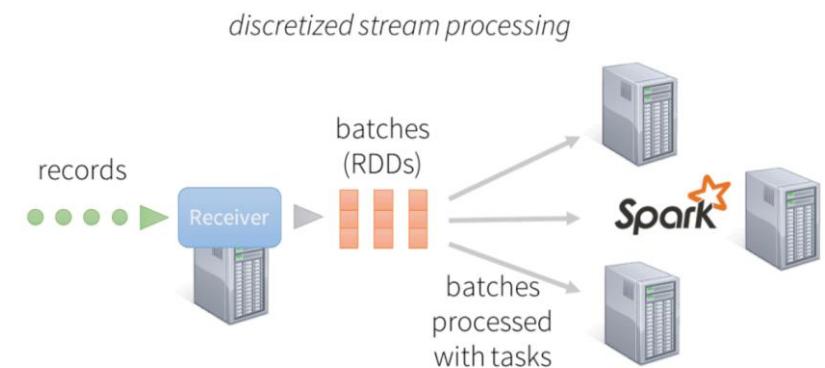
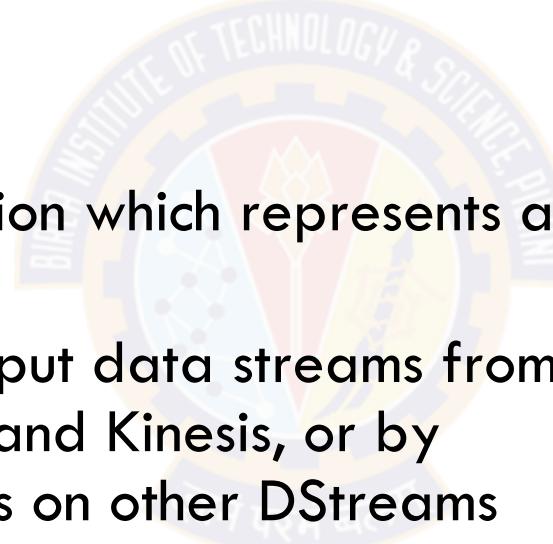
- An extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams
- Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets
- Data can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window
- Processed data can be pushed out to filesystems, databases, and live dashboards
- Can apply Spark's machine learning and graph processing algorithms on data streams



<https://spark.apache.org/streaming/>

# Discretized Streams

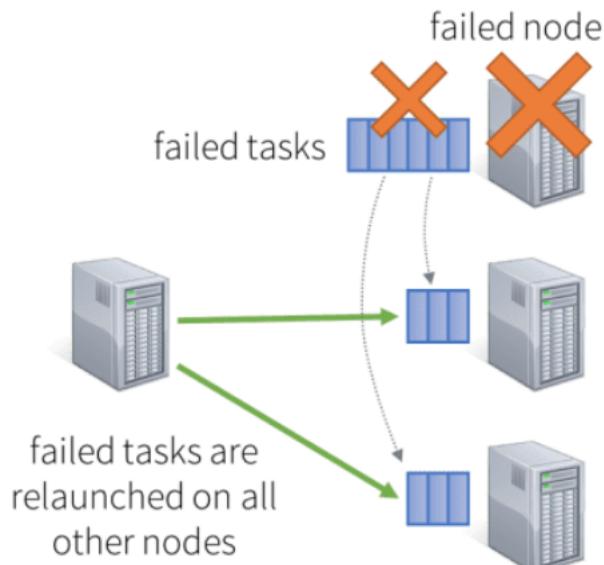
- Live input data streams are divided into batches
- These batches are then processed to generate the final stream of results in batches.
- DStreams
  - Provides a high-level abstraction which represents a continuous stream of data
  - Can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams
  - Internally, a DStream is represented as a sequence of small RDDs



records processed in batches with short tasks  
each batch is a RDD (partitioned dataset)

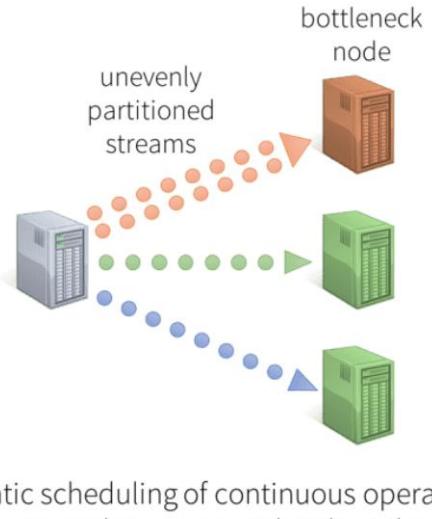
- <https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html>

# Benefits of batching (1)

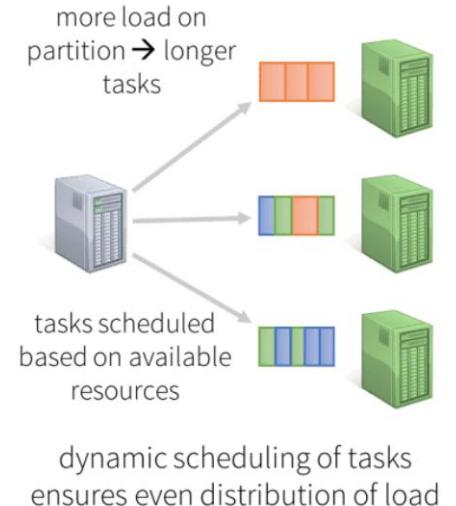


faster recovery by using multiple nodes for recomputations

## Traditional systems



## Spark Streaming



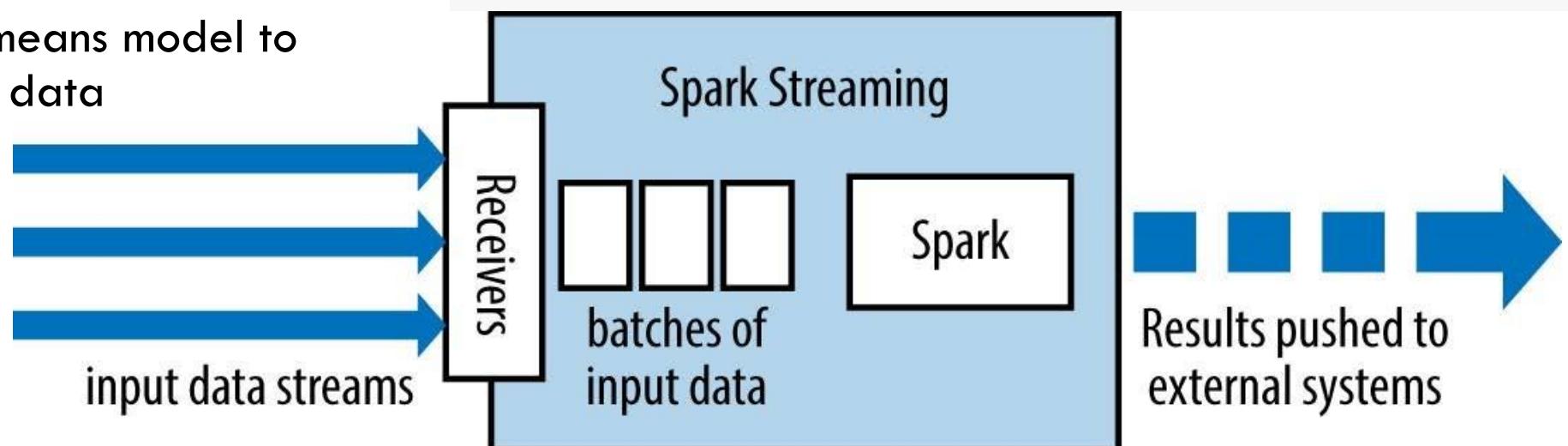
- Faster recovery from failures
- Dynamic scheduling of batches / tasks leads to better load balancing

# Benefits of batching (2)

- Easier interoperability of batch, stream and interactive analysis
  - DStream is just a series of RDDs
  - E.g. join a DStream with static RDD or convert into DataFrame and interactively query using SQL
- Can apply MLlib functions on DStream RDDs.
  - E.g. apply a k-means model to label streaming data

```
// Create data set from Hadoop file  
val dataset = sparkContext.hadoopFile("file")  
// Join each batch in stream with the dataset  
kafkaDStream.transform { batchRDD =>  
    batchRDD.join(dataset).filter(...)  
}
```

```
// Learn model offline  
val model = KMeans.train(dataset, ...)  
// Apply model online on stream  
val kafkaStream = KafkaUtils.createDStream(...)  
kafkaStream.map { event => model.predict(featurize(event)) }
```



# Spark Streaming Sources & Receivers

- Every input DStream (except file stream) is associated with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.
- Basic sources
  - ✓ These are the sources directly available in the StreamingContext API.
  - ✓ Examples: file systems, and socket connections
- Advanced sources
  - ✓ Sources like Kafka, Flume, Kinesis, etc. are available through extra utility classes
  - ✓ These require linking against extra dependencies
- Reliable Receiver
  - ✓ Is the one that correctly sends an acknowledgment to a source when the data receives and stores in Spark with replication
- Unreliable Receiver
  - ✓ Is the one that does not send an acknowledgment to a source

# Basic Streaming Transformations

Transformations are used to express the business logic of a streaming application

## Stateless transformations:

- ✓ Map transforms one record to one record, for example: change format of the record, enrich record with some data.
- ✓ Filter filters out the records that doesn't satisfy the predicate.
- ✓ FlatMap is the most general type of stateless transformation, outputting zero or more records for each input record, for example: tokenize a record containing a sentence into individual words.

## Stateful transformations include:

- ✓ Aggregation combines all the records to produce a single value. Examples: min, max, sum, count, avg.
- ✓ Group-and-aggregate extracts a grouping key from the record and computes a separate aggregated value for each key.
- ✓ Join joins same-keyed records from several streams.
- ✓ Sort sorts the records observed in the stream.

# Spark Streaming Transformations

## Transformation Operations

- Spark transformations allow modification of the data from the input Dstream

Transformation	Meaning
<code>map(func)</code>	Return a new DStream by passing each element of the source DStream through a function <i>func</i> .
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items.
<code>filter(func)</code>	Return a new DStream by selecting only the records of the source DStream on which <i>func</i> returns true.
<code>repartition(numPartitions)</code>	Changes the level of parallelism in this DStream by creating more or fewer partitions.
<code>union(otherStream)</code>	Return a new DStream that contains the union of the elements in the source DStream and <i>otherDStream</i> .
<code>count()</code>	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream.
<code>reduce(func)</code>	Return a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function <i>func</i> (which takes two arguments and returns one). The function should be associative and commutative so that it can be computed in parallel.
<code>countByValue()</code>	When called on a DStream of elements of type K, return a new DStream of (K, Long) pairs where the value of each key is its frequency in each RDD of the source DStream.

# Spark Streaming Operations

## Output Operations

- Output operations allow DStream's data to be pushed out to external systems like a database or a file systems

Output Operation	Meaning
<code>print()</code>	Prints the first ten elements of every batch of data in a DStream on the driver node running the streaming application. This is useful for development and debugging. <small>Python API</small> This is called <code>pprint()</code> in the Python API.
<code>saveAsTextFiles(prefix, [suffix])</code>	Save this DStream's contents as text files. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ".
<code>saveAsObjectFiles(prefix, [suffix])</code>	Save this DStream's contents as SequenceFiles of serialized Java objects. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ". <small>Python API</small> This is not available in the Python API.
<code>saveAsHadoopFiles(prefix, [suffix])</code>	Save this DStream's contents as Hadoop files. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ". <small>Python API</small> This is not available in the Python API.
<code>foreachRDD(func)</code>	The most generic output operator that applies a function, <code>func</code> , to each RDD generated from the stream. This function should push the data in each RDD to an external system, such as saving the RDD to files, or writing it over the network to a database. Note that the function <code>func</code> is executed in the driver process running the streaming application, and will usually have RDD actions in it that will force the computation of the streaming RDDs.

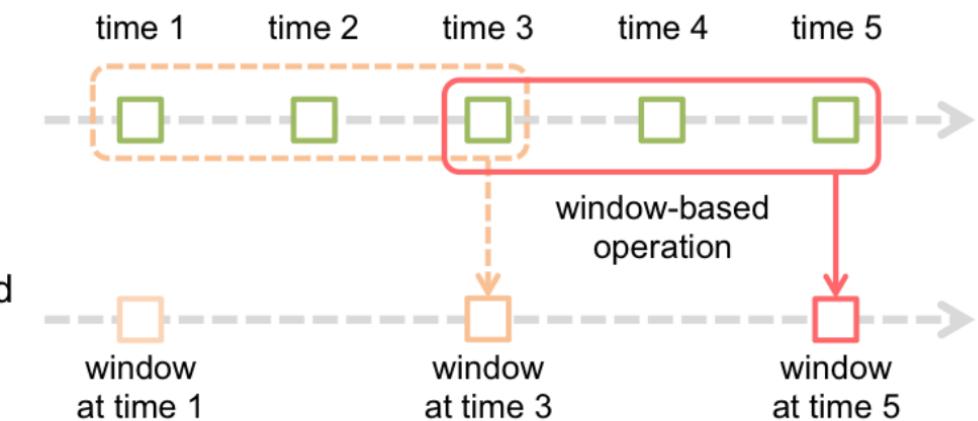
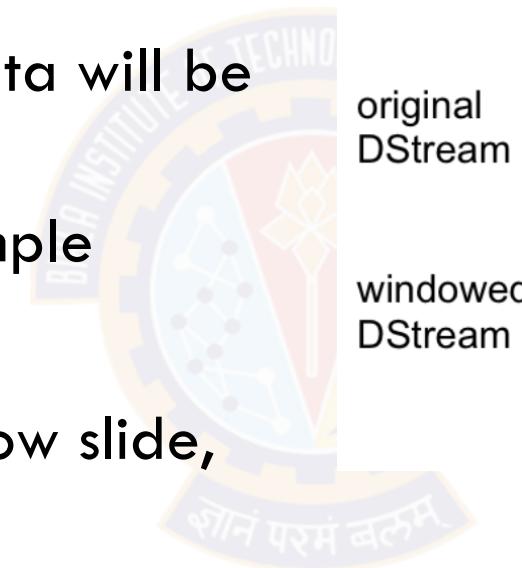
# Topics for today

- Spark ML performance
- Stream processing
- Spark Streaming
- Windowing



# Window operations (1)

- Batch interval : Time period taken to batch data into an RDD
- Window size : How much data will be in RDD before processing
  - ✓ e.g. 3 RDDs in this example
- Sliding interval
  - ✓ How much will the window slide, e.g. 2 in this example



# Window operations (2)

Transformation	Meaning
<code>window(windowLength, slideInterval)</code>	Return a new DStream which is computed based on windowed batches of the source DStream.
<code>countByWindow(windowLength, slideInterval)</code>	Return a sliding window count of elements in the stream.
<code>reduceByWindow(func, windowLength, slideInterval)</code>	Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using <code>func</code> . The function should be associative and commutative so that it can be computed correctly in parallel.
<code>reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])</code>	When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function <code>func</code> over batches in a sliding window. <b>Note:</b> By default, this uses Spark's default number of parallel tasks (2 for local mode, and in cluster mode the number is determined by the config property <code>spark.default.parallelism</code> ) to do the grouping. You can pass an optional <code>numTasks</code> argument to set a different number of tasks.
<code>reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])</code>	A more efficient version of the above <code>reduceByKeyAndWindow()</code> where the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides. However, it is applicable only to "invertible reduce functions", that is, those reduce functions which have a corresponding "inverse reduce" function (taken as parameter <code>invFunc</code> ). Like in <code>reduceByKeyAndWindow</code> , the number of reduce tasks is configurable through an optional argument. Note that <code>checkpointing</code> must be enabled for using this operation.
<code>countByValueAndWindow(windowLength, slideInterval, [numTasks])</code>	When called on a DStream of (K, V) pairs, returns a new DStream of (K, Long) pairs where the value of each key is its frequency within a sliding window. Like in <code>reduceByKeyAndWindow</code> , the number of reduce tasks is configurable through an optional argument.

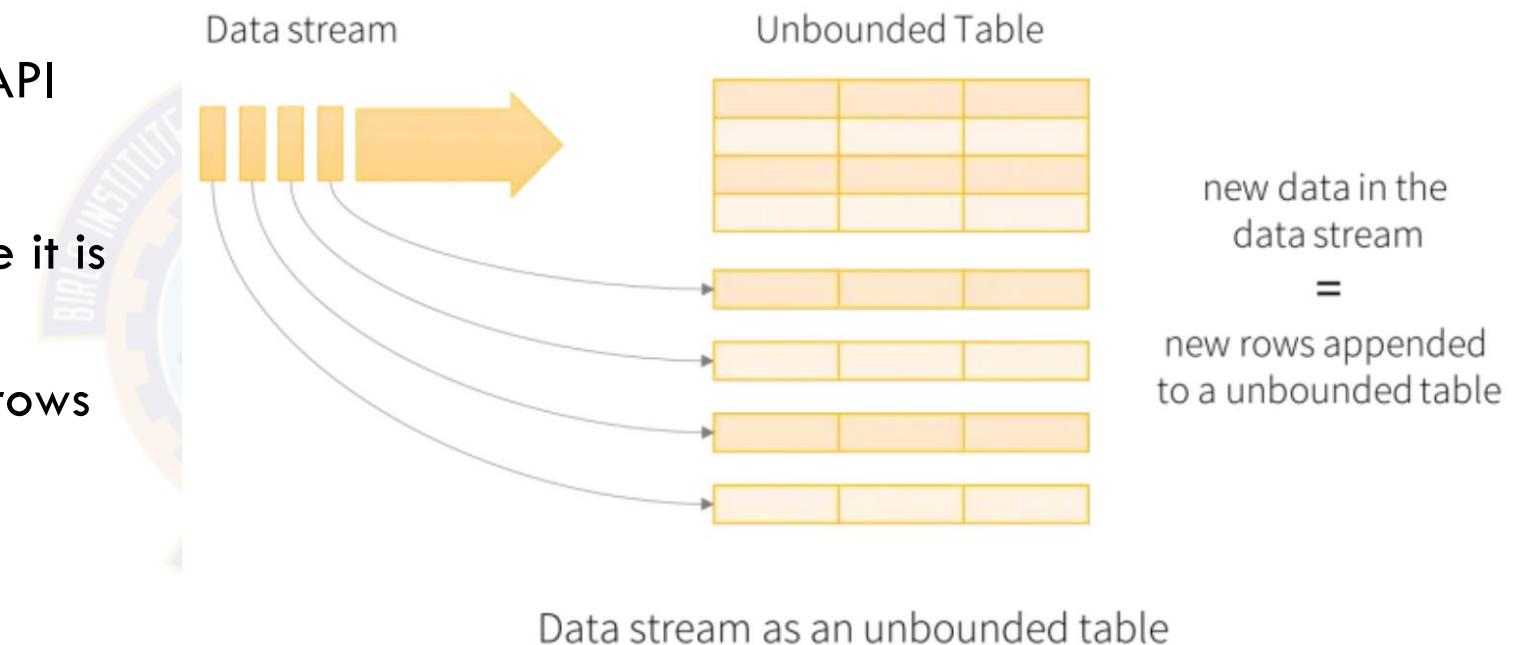
# Structured Streaming

- Structured Streaming treats a live data stream as a table that is being continuously appended.
- This leads to a new stream processing model that is very similar to a batch processing model
- Express streaming computation as standard batch-like query as on a static table
- Spark runs it as an incremental query on the unbounded input table
- Structured Streaming does not materialize the entire table.
- It reads the latest available data from the streaming data source
- Processes data incrementally to update the result
- After processing, it discards the source data.
- Spark is responsible for updating the Result Table when there is new data

<https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>

# Structured Streaming

- Works on DataSets/DataFrames API
- No batching
- Data is processed as rows because it is structured
- Closer to real streaming because rows are smaller than an RDD



# pyspark.sql.streaming – Output Modes

Specifies how data of a streaming DataFrame/Dataset is written to a streaming sink.

Options include:

- ***append***: Only the new rows in the streaming DataFrame/Dataset will be written to the sink
- ***complete***: All the rows in the streaming DataFrame/Dataset will be written to the sink every time there are some updates
- ***update***: only the rows that were updated in the streaming DataFrame/Dataset will be written to the sink every time there are some updates. If the query doesn't contain aggregations, it will be equivalent to *append* mode.

Example:

```
writer = sdf.writeStream.outputMode('append')
```

***Note:- append and update options are under implementation in apache spark***

# Comparison of Spark with Kafka

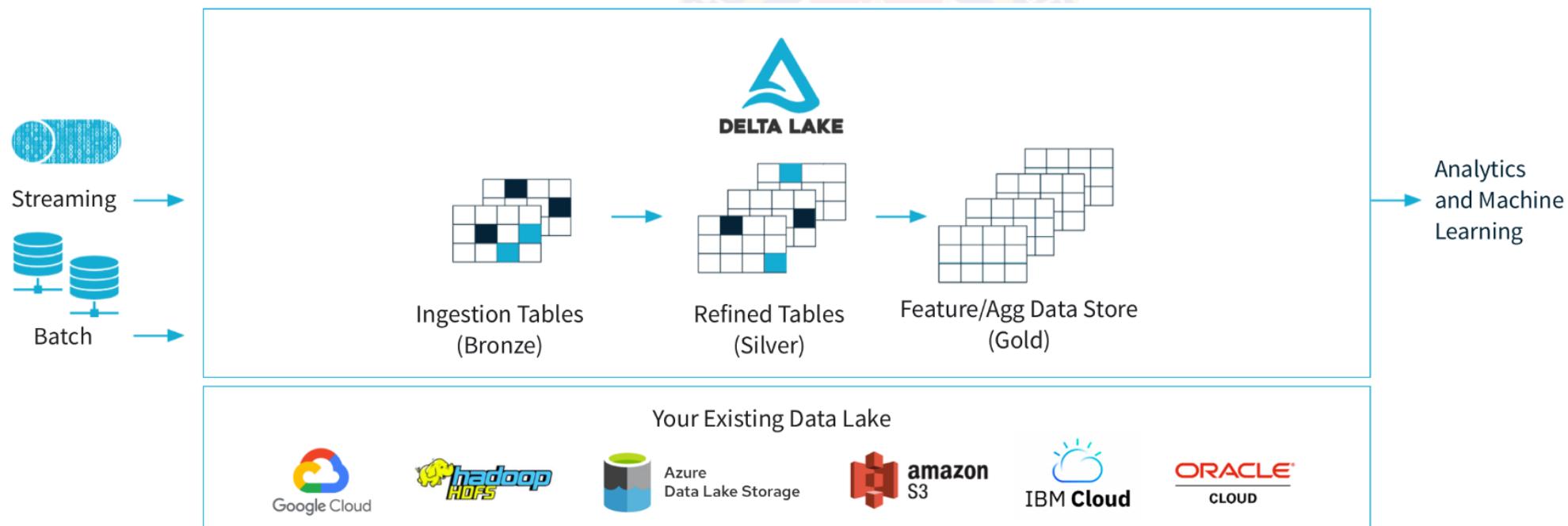
	Apache Spark	Apache Kafka
Description	General purpose distributed processing system used for big data workloads. It uses in-memory caching and optimized query execution to provide fast analytic queries against data of any size.	Distributed Streaming platform that allows developers to create applications that continuously produce and consume data streams.
Processing Model	Batch processing and streaming	Event Streaming
Throughput & Latency	High throughput, low latency	High throughput, low latency
Scalability	Horizontally scalable as a distributed system, though scaling is expensive due to RAM requirement	Horizontally scalable to support a growing number of users and use cases, meaning that it can handle an increasing amount of data by adding more nodes & partitions to the system
Fault Tolerance	Fault tolerant by nature (RDDs). In case of node failure, RDDs are automatically recreated.	Fault-tolerant as it replicates data across multiple servers and can automatically recover from node failures.

# Delta Lake

## Build Lakehouses with Delta Lake

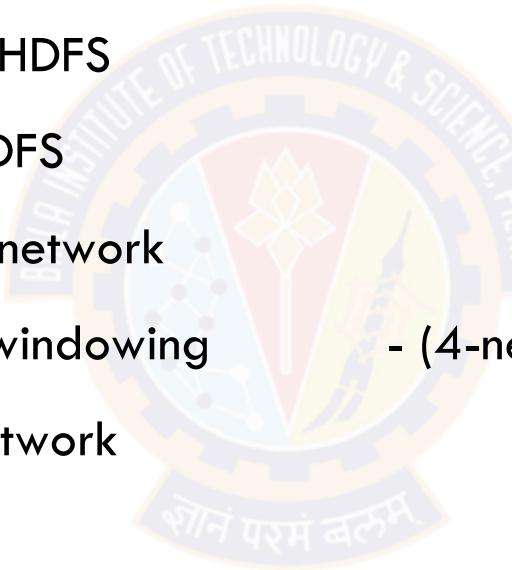
Delta Lake is an open-source storage framework that enables building a Lakehouse architecture with compute engines including Spark, PrestoDB, Flink, Trino, and Hive and APIs for Scala, Java, Rust, and Python.

<https://docs.databricks.com/en/delta/index.html>



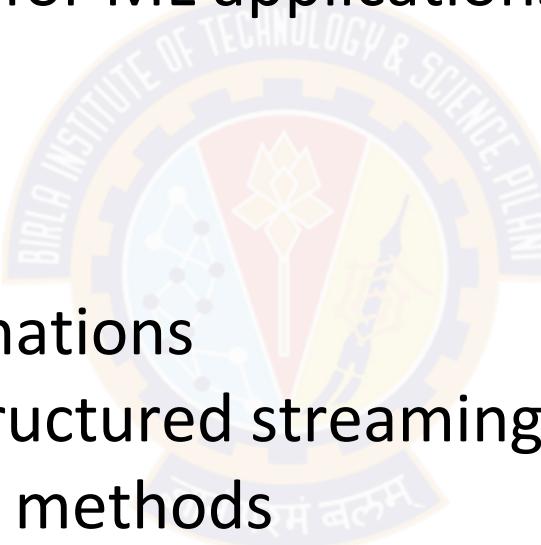
# Spark Streaming – Hands on demos

- Streaming of unstructured data from HDFS - (1-hdfs\_wordcount.py)
- Streaming of structured data from HDFS - (2-hdfsStructuredStreaming.py)
- Streaming of unstructured data from network - (3-network\_wordcount.py)
- Streaming of unstructured data with windowing - (4-network\_wordcount\_windowed.py)
- Streaming of structured data from network - (5-sql\_network\_wordcount.py)



# Summary

- Spark RDD Performance tuning
- Spark RDD Persistence levels
- Spark Performance tuning for ML applications
- Hardware provisioning
- Stream processing
- Spark Streaming
  - ✓ - Streaming Transformations
  - ✓ - Unstructured and structured streaming
  - ✓ - Different windowing methods
  - ✓ - Handson demos of Spark streaming
- Comparison with Kafka streams





Next Session:  
Cloud Computing

# Real Time Windows



*Janardhanan PS*

# Windowing

- Windows provide a finite, bounded view on top of an infinite stream
- A window defines how records are selected from the stream and grouped together to a meaningful frame
- The transformations (business logic) are executed just on the records contained in the window

# Windowing

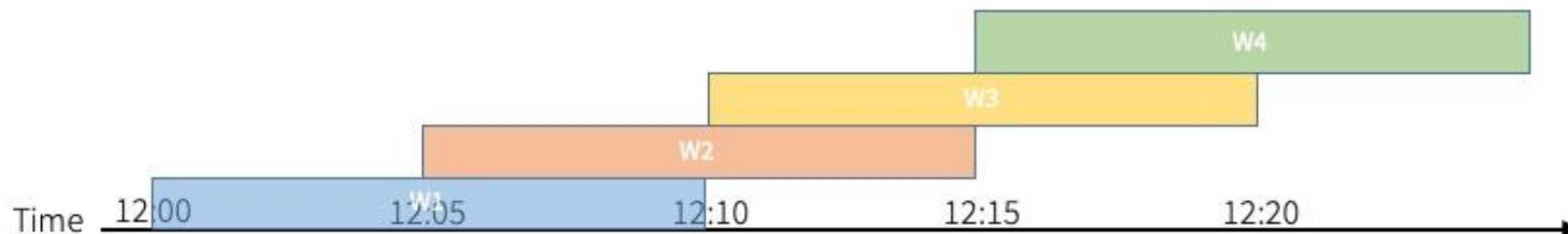
- Aggregations continue to build up over time
- Windowing gives snapshot of an aggregate within a given timeframe
- Four types of windows we'll discuss
  - Tumbling
  - Sliding
  - Session
  - Hopping

# Types of Time Windows

Tumbling Windows (5 mins)



Sliding Windows (10 mins, slide 5 mins)



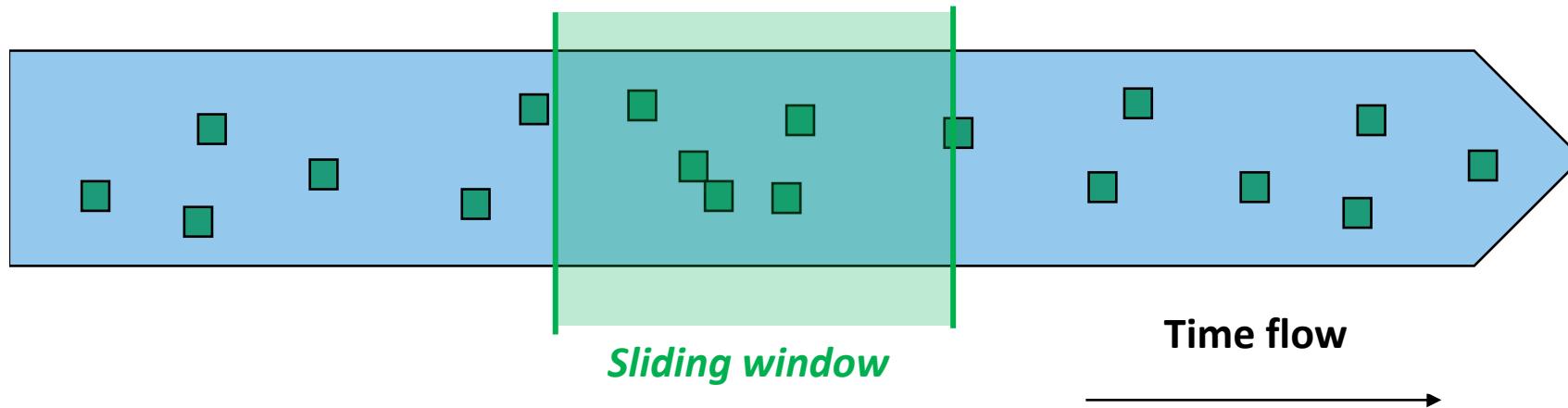
Session Windows (gap duration 5 mins)



# Tumbling Window



# Sliding Window

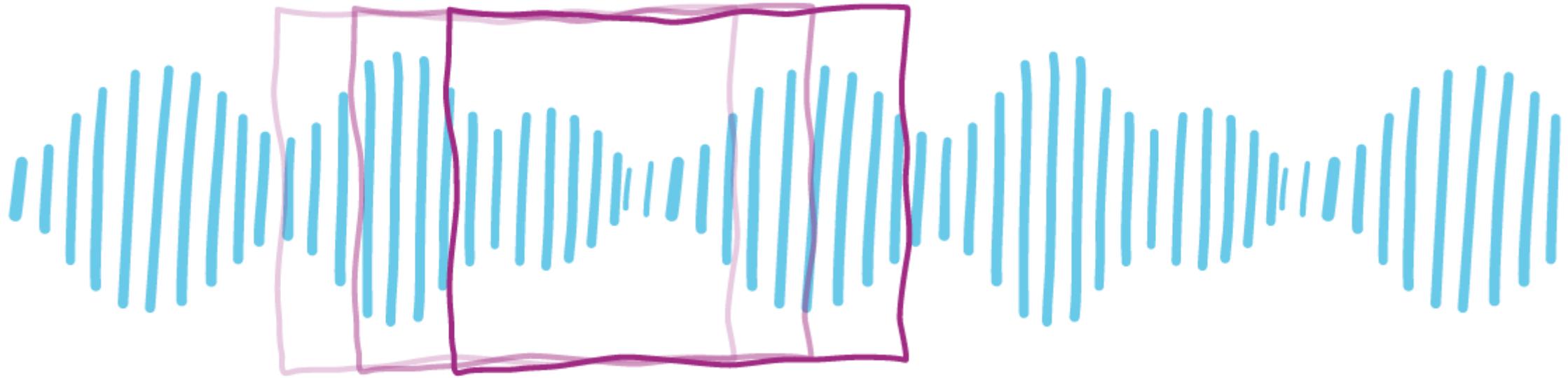


```
// Group the data by window and and compute word count in each group  
val windowedCounts = words.groupBy( window($"timestamp", "10 minutes", "5 minutes"), $"word" ).count()
```

# Session Window



# Hopping Window



# Difference between Sliding & Hopping Windows

## Hopping windows:

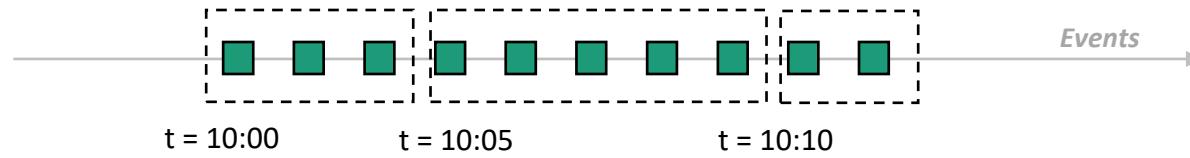
- The window is re-evaluated on fixed time intervals, independent of the actual content of the data stream (ie, independent of the records).
- You could use a hopping window if you need to get periodic results.
- For example: a daily business report over the last seven days; or an hourly update over the last 24h. Even if no new record are processed, you want to get a result in fixed time intervals sent downstream.

## Sliding window:

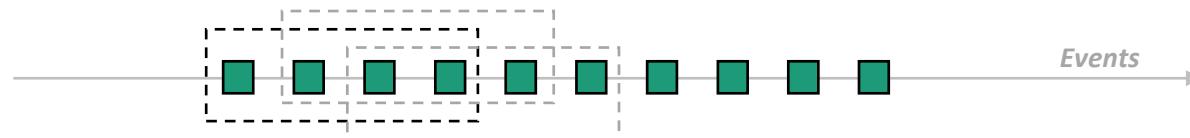
- Is re-evaluated only if the content of the window changes, ie, each time a new record enters or leaves the window.
- This type of window is good for a “moving average” computation as an example. As long as no new records arrive, the result (current average) does not change and thus you don’t want get the same result sent downstream over and over again.
- Only if a record enters or leave the window, and the average changes you want to get an update. Alerting on a threshold may be a good use-case: it’s only useful to re-evaluate the threshold if it did change; there is no advantage to evaluate the same result in fixed time intervals

# Related Concepts

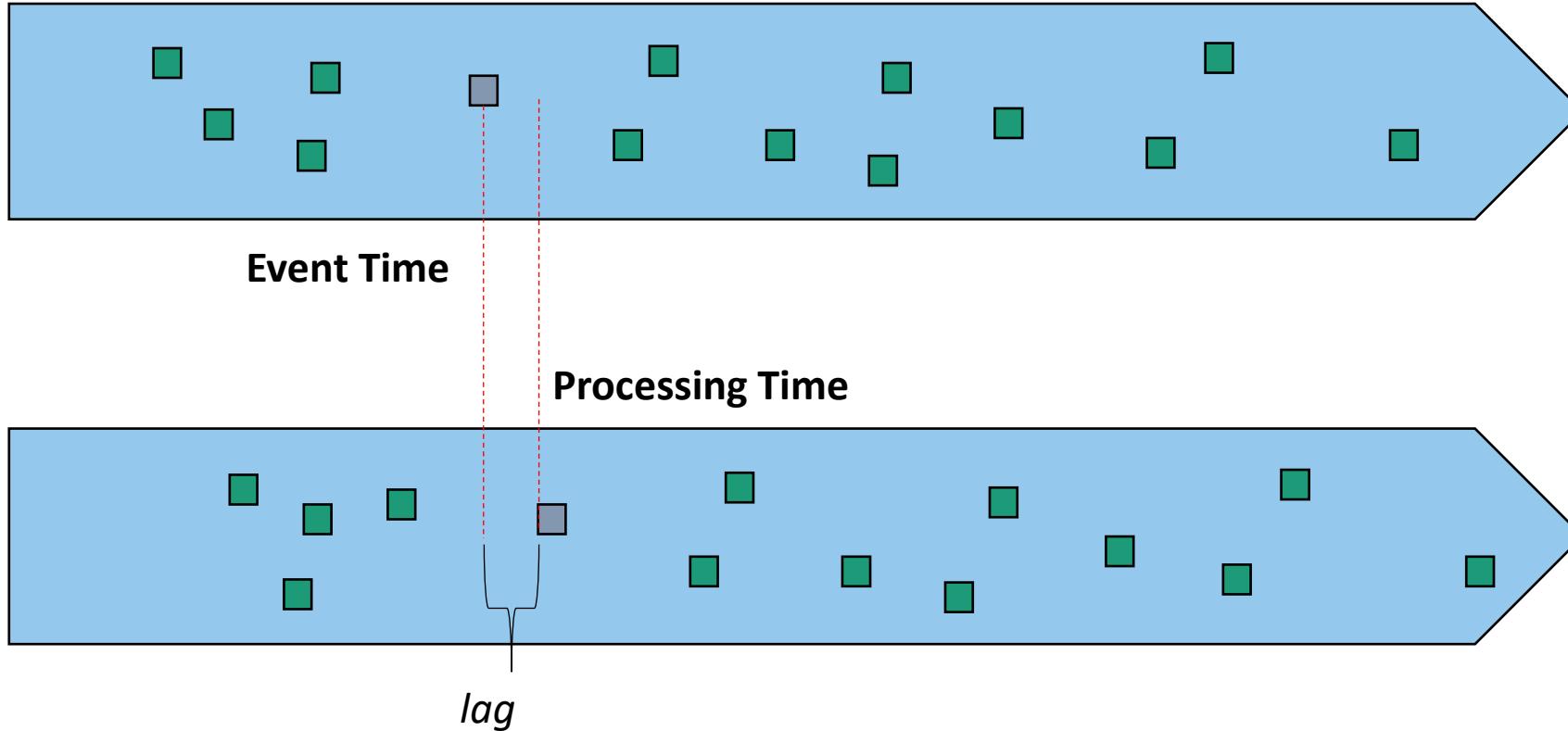
## Time-based windows



## Count-based windows



# Related Concepts - Lag



Thank you



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DSECL ZG 522: Big Data Systems

## Session 15: Cloud computing

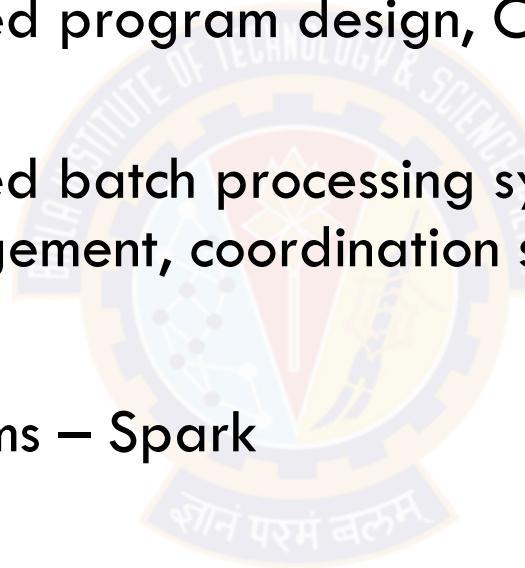
---

Janardhanan PS

[janardhanan.ps@wilp.bits-pilani.ac.in](mailto:janardhanan.ps@wilp.bits-pilani.ac.in)

# Where are we in the course

- Basics: LoR, caching, parallel / distributed systems, understand performance, availability etc.
- Advanced concepts: Distributed program design, Consistency models, CAP theorem
- Hadoop ecosystem - file based batch processing systems — compute, storage/FS, DBs, workflows, cluster management, coordination systems (ZK) etc.
- NoSQL DBs
- In-memory / Streaming systems – Spark
- What next ?  
Cloud and big data processing



# Topics for today

- Cloud concepts
- Compute instances - AWS
  - EC2
  - EMR
  - Lambda
- AWS demos
- Scalability and Elasticity
- Dynamic provisioning
- Multi-Tenant Data Architecture



# What really is Cloud Computing?

**Cloud computing** is a new computing paradigm, involving data and/or computation **outsourcing**, with

- Infinite and elastic resource scalability
- On demand “just-in-time” provisioning
- No upfront cost ... pay-as-you-go

That is, use **as much or as less you need**, use only **when you want**, and **pay for only what you use**

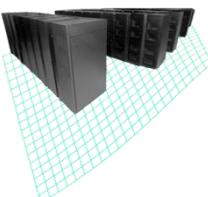
# Cloud Computing – A New Paradigm

- An IT service delivered to users that provides:
  - A simple user interface that automatically provisions IT resources
  - Capacity on demand with massive scalability
  - Innovative service delivery models for applications

1990

## Grid Computing

- Solving large problems with parallel computing



## Utility Computing

- Offering computing resources as a metered service



## Software as a Service

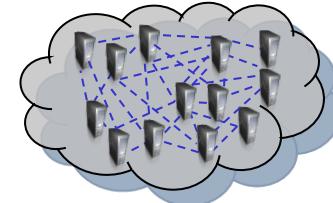
- Network-based subscriptions to applications



2009

## Cloud Computing

- Anytime, anywhere access to resources delivered dynamically as a service



# Definition

The US National Institute of Standards (NIST) defines cloud computing as follows:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.



## NIST's 3-4-5 rule of Cloud Computing

- **5** essential characteristics of cloud computing infrastructure
- **4** deployment models
- **3** cloud service models or service types for any cloud platform

# 5 Essential Characteristics of Cloud Computing

1. On demand self-service
2. Broad network access
3. Resource pooling – Location transparent
4. Rapid elasticity
5. Measured service – pay per use



On-demand  
self-service

Ubiquitous  
network  
access

Location  
transparent  
resource  
pooling

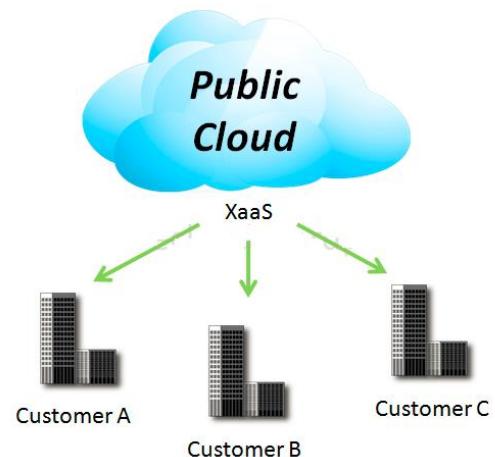
Rapid  
elasticity

Measured  
service with  
pay per use

# 4 Deployment models

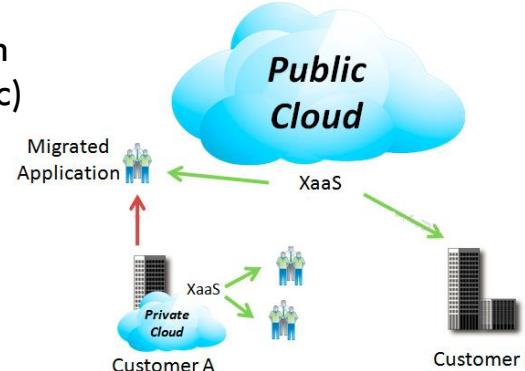
## Public Cloud

Mega-scale cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services. E.g. AWS, Azure, Google, ...



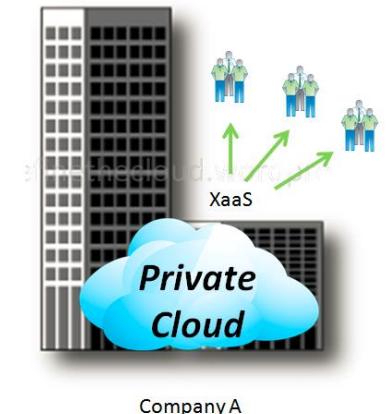
## Hybrid Cloud

The cloud infrastructure is a composition of two or more clouds (private or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability, cross domain security etc.



## Private Cloud

The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

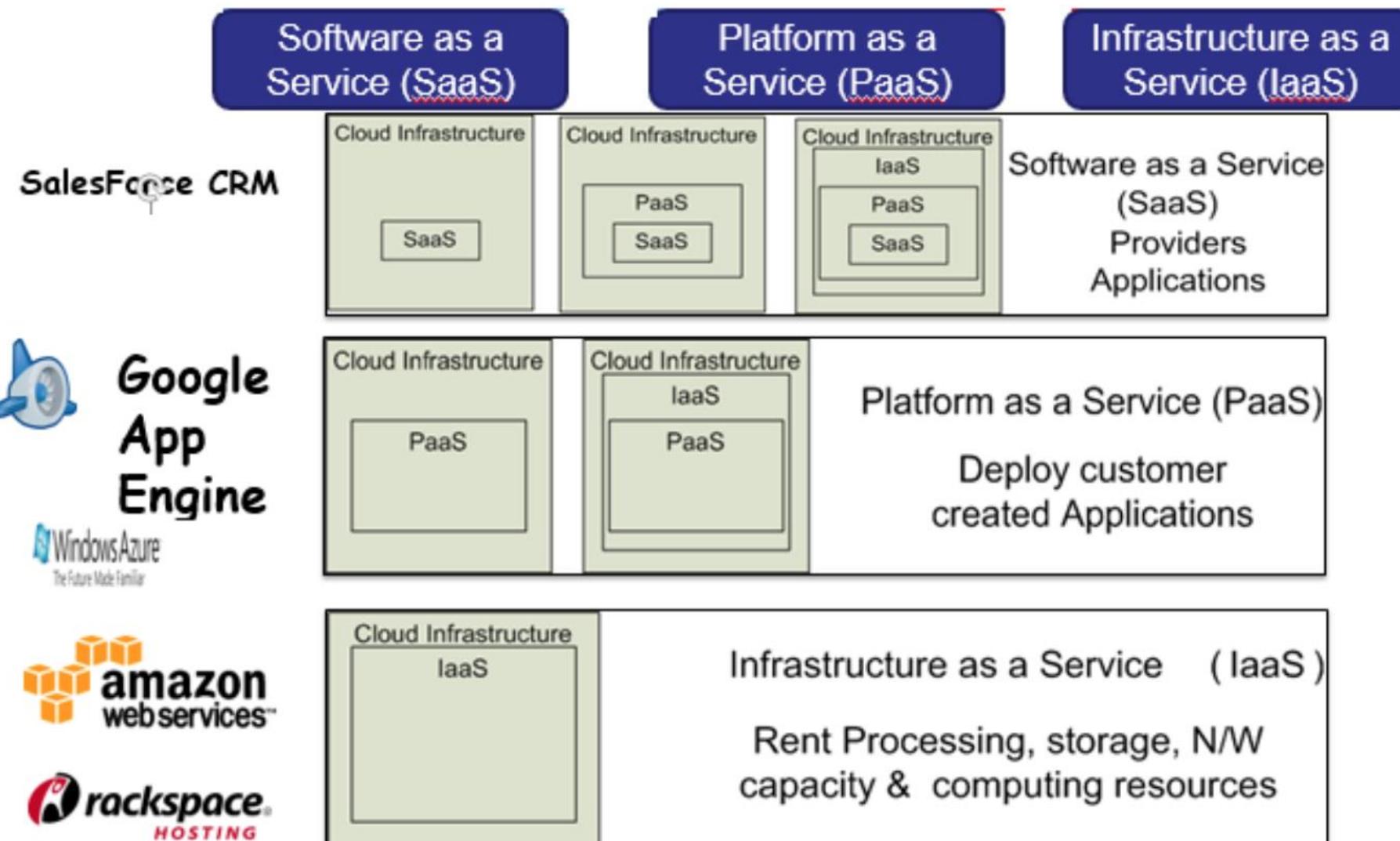


## Community Cloud

An ‘infrastructure shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise’ according to NIST.

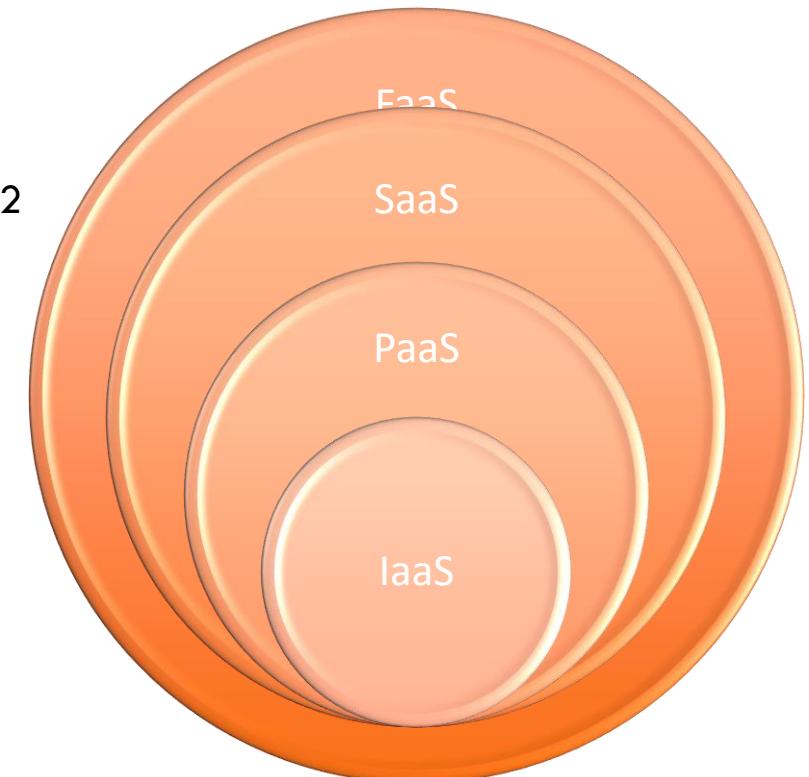


# 3 Cloud Service Models



# Cloud services for Big Data

- Cloud follows same model as Big Data, both requiring distributed clusters of computing devices.
  - ✓ Cloud Computing considered as ideal platform for handling big data
- IaaS:
  - ✓ Can provide huge storage and computational power requirements for Big Data through limitless storage and computing ability of cloud computing, e.g. AWS S3, EC2
- PaaS:
  - ✓ Vendors offer platforms ready with Hadoop and MapReduce (AWS EMR).
  - ✓ Saves hassles of installations and managements of these environments
- SaaS:
  - ✓ Great help to organizations which requires specialized software's for big data like for social media analytics, feedback monitoring etc.
  - ✓ SaaS vendors provides out of the box solution for such common use cases
- FaaS:
  - ✓ A new way to host code at function level where user is abstracted away from the underlying infrastructure needs to run the function



# Cloud services –Comparison with Generations of Programming languages

There are five generations of Programming languages:

✓ First-Generation Languages :

- Low-level languages like machine language.

✓ Second-Generation Languages :

- Low-level assembly languages used in kernels and hardware drives.

✓ Third-Generation Languages :

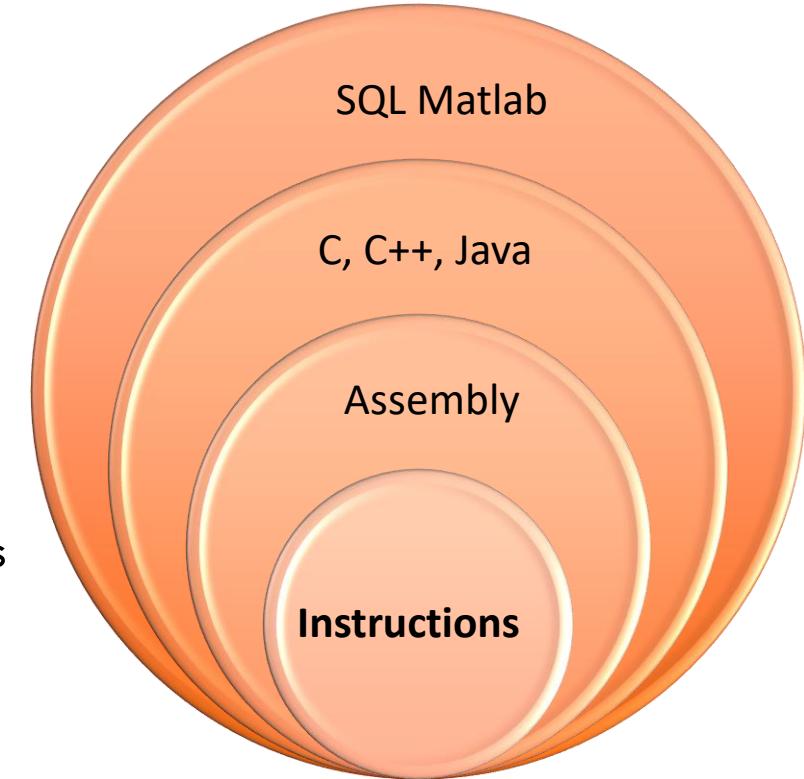
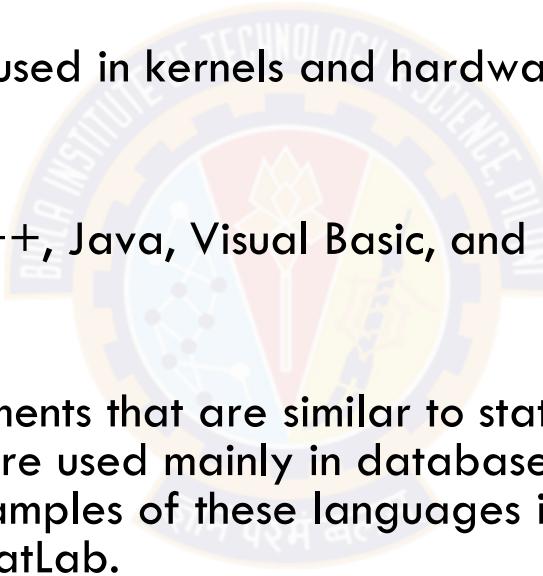
- High-level languages like C, C++, Java, Visual Basic, and JavaScript.

✓ Fourth Generation Languages :

- Languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting. Examples of these languages include Perl, Python, Ruby, SQL, and MatLab.

✓ Fifth Generation Languages :

- Domain Specific Languages (DSLs) Languages that have visual tools to develop a program. Examples include Mercury, OPS5, and Prolog.



# Cloud providers in Big Data market

- **Amazon (Amazon Web Services AWS)**

- ✓ EC2
- ✓ EMR - Elastic MapReduce
- ✓ Kinesis
- ✓ DynamoDB
- ✓ Amazon S3
- ✓ Redshift

- **Google (Google Cloud Platform GCP)**

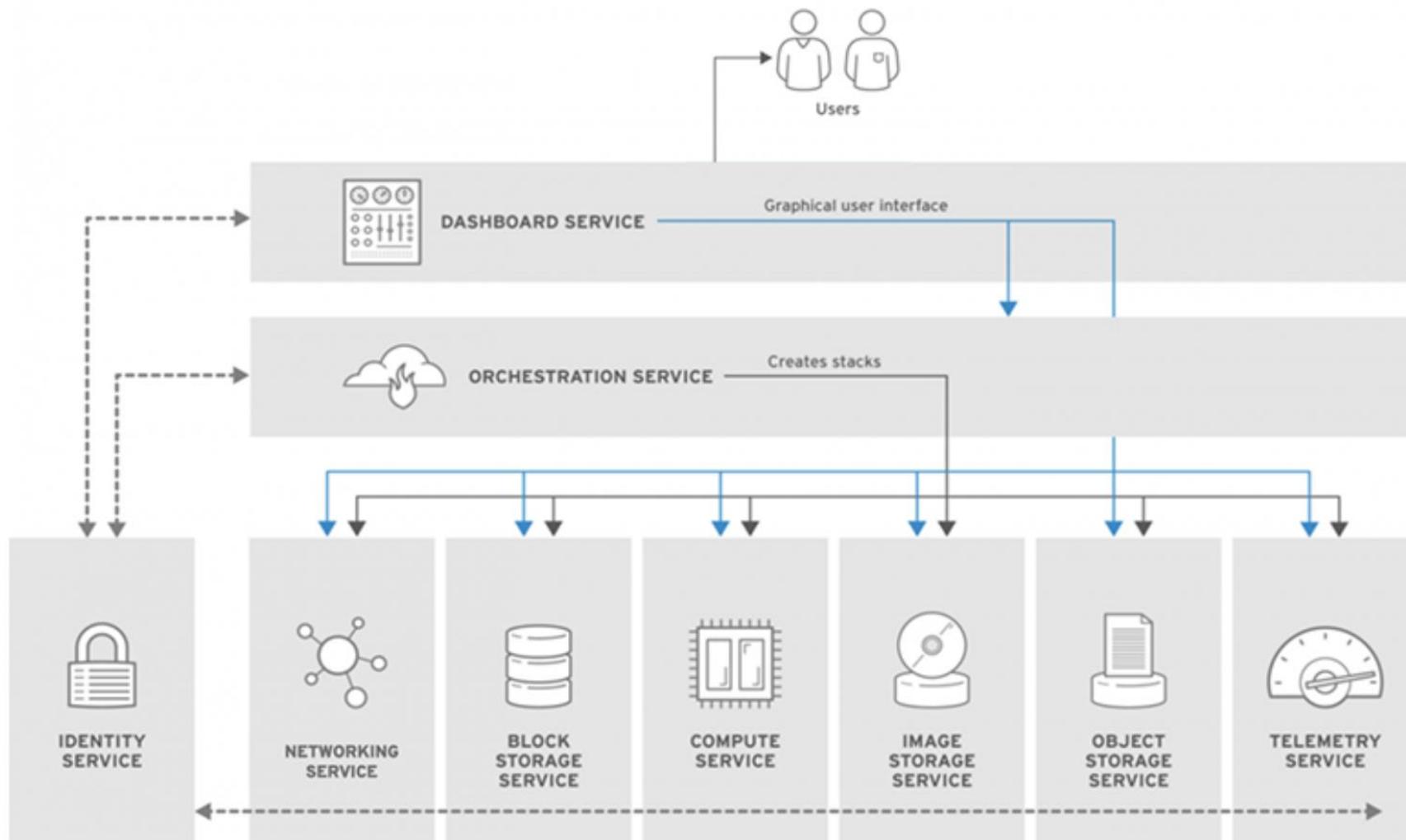
- ✓ Google Compute Engine
- ✓ Google BigQuery
- ✓ Google Prediction API

- **Microsoft Azure**

- ✓ Azure PaaS cloud based on Windows and SQL
- ✓ Windows Azure HD Insight

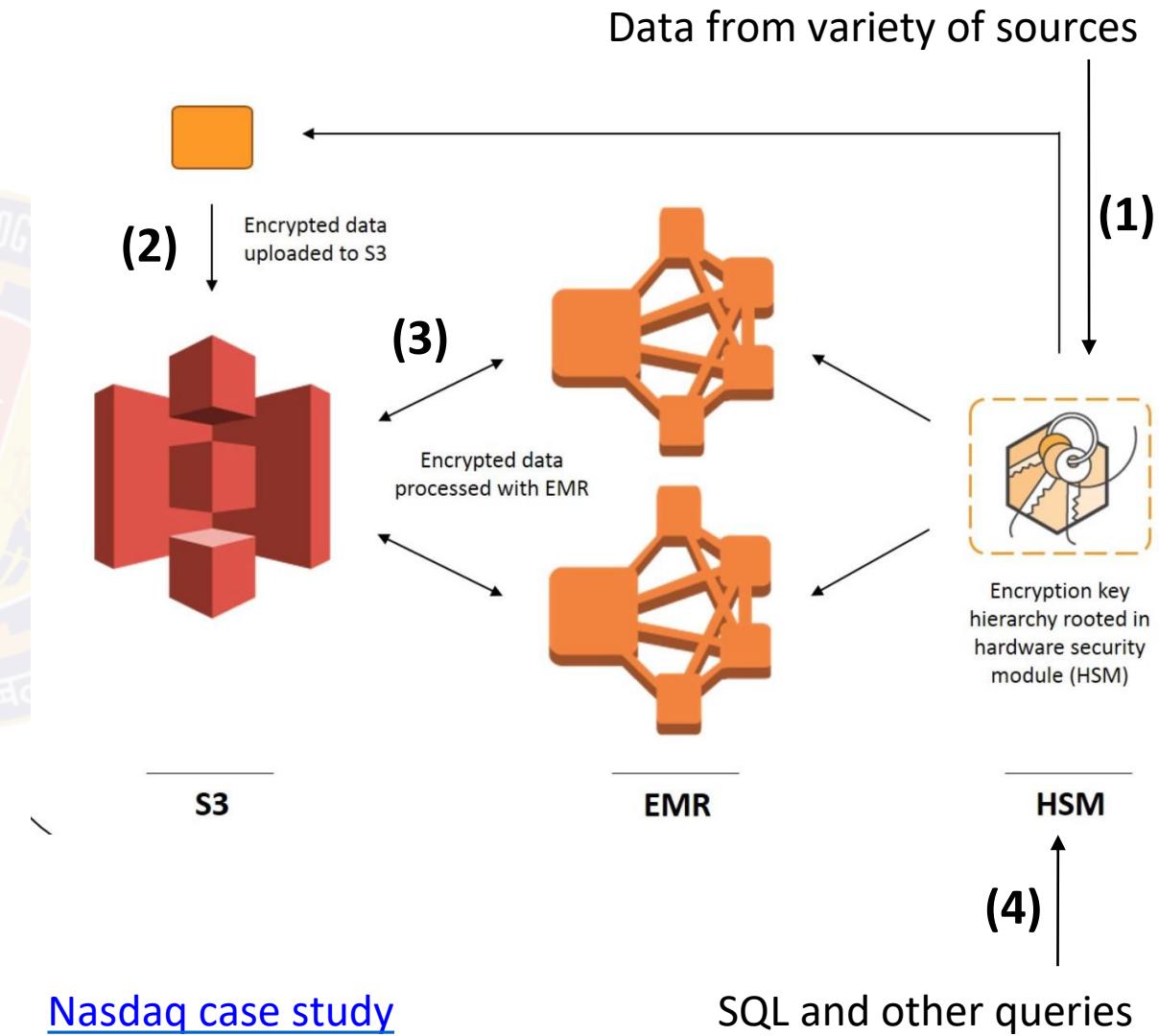
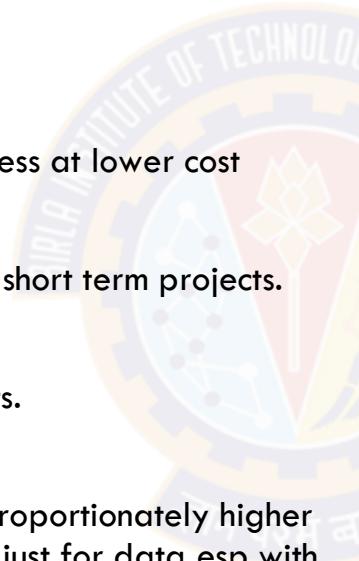


# A basic IaaS architecture



# Case study: Nasdaq

- Operates financial exchanges
- AWS Redshift is the data warehouse with 5.5B rows/day with peak 14B/day in Oct 2014.
- A new DWH environment is setup using EMR and S3
  - give more teams access to gigantic data sets
  - cost efficiency
- Hadoop enables large and more varied historical data access at lower cost
- Why EMR and S3 ?
  - On demand set up of Hadoop clusters on Cloud to run short term projects.
  - Existing tech to move around TBs of data.
  - Storage on Cloud in S3 to scale to very large data sets.
- Why not HDFS and use S3 ?
  - Decouple data and compute because data size is disproportionately higher than compute frequency - so can't add Hadoop nodes just for data esp with replication factor
  - Similar to Netflix. EMRFS layer can enable compute nodes access data in S3



# Case study: Social media analytics



Ref: <https://aws.amazon.com/kinesis/>

# Cloud: Pros and Cons

- Cost effective in specific cases, esp for short term projects, experiments, without getting into capital expense.
  - ✓ Be clear about usage
  - ✓ No in-house capability
- Reliability: Shift the onus of uptime on expert providers who can scale manpower, tools, systems to meet the demands
- Backup-recovery managed by experts
- Storage and compute scaling as required for short time periods. Stop guessing capacity.
- Use global DCs that are accessible from anywhere but without having to own one.

- 
- Need to re-think security for data on Cloud or hybrid mode of operation where on-prem systems need to be connected
  - Need to depend on external providers for infra / applications
  - Vendor lock-in
  - Not cost effective if not well planned and tracked for usage

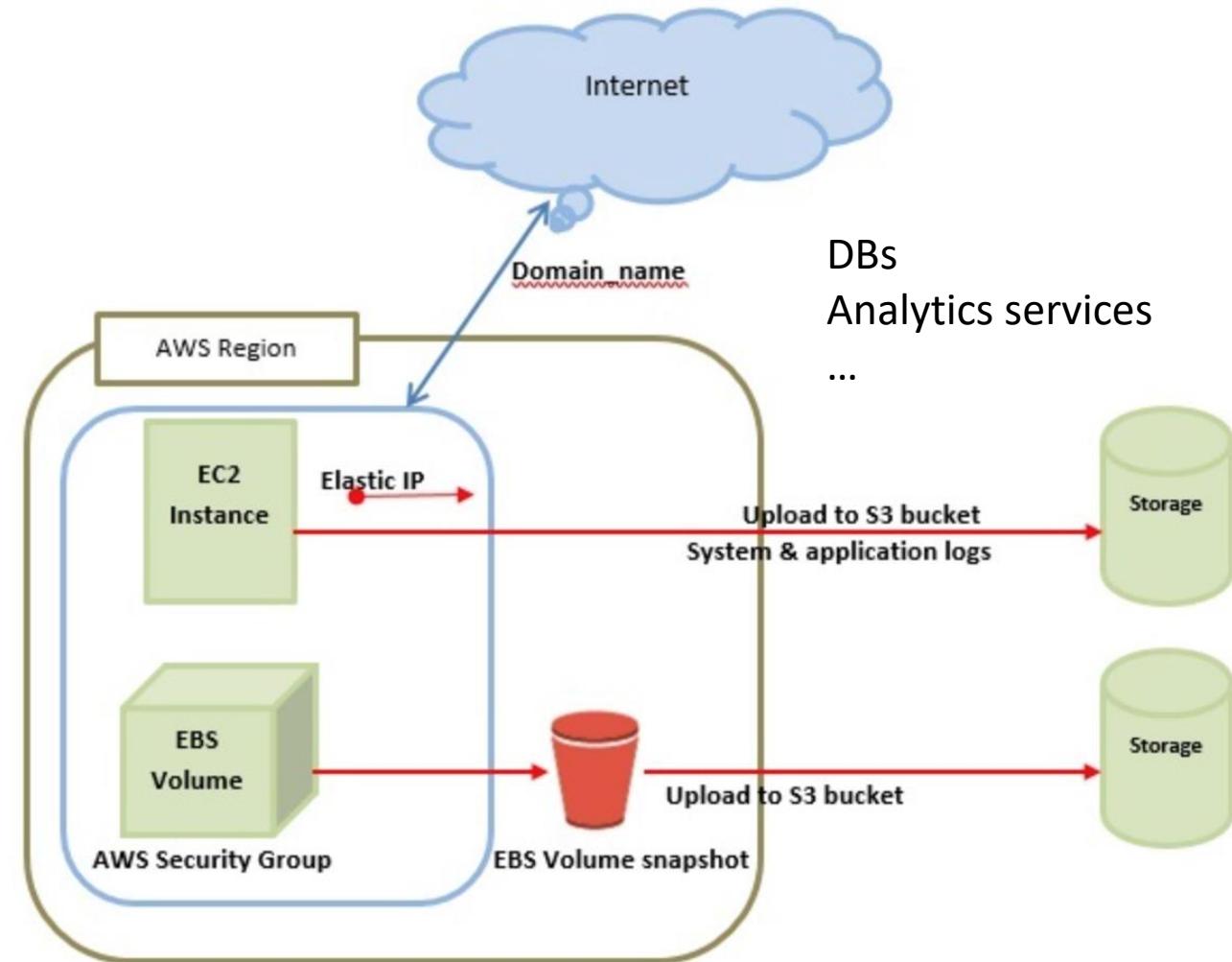
# Topics for today

- Cloud concepts
- **Compute instances - AWS**
  - ✓ EC2
  - ✓ EMR
  - ✓ Lambda
- AWS demos
- Scalability and Elasticity
- Dynamic provisioning
- Multi-Tenant Data Architecture



# Example: AWS Architecture

- Region: DCs are located in specific regions
- Availability zone: A region is split into AZs to isolate failures. AZs within region are connected with low-latency network and may be within or across DCs but with redundant power and networking.
- Compute instance: A machine, VM, container or function (FaaS). With DNS support, Elastic static IPs. Can be started from images.
- Attached volume: A compute instance is connected to block storage volumes for local FS.
- External storage: To serve as large remote raw data storage. E.g. An object storage like S3
- Security groups: Rules created to control access to the compute and storage instances.
- Load balancing: Service to spray load across multiple compute instances across or within AZ.
- Elastic scaling: Manual or automatically creation / deletion of compute instances.
- Backup/recovery: Snapshots can be taken to preserve state.



# Amazon Simple Storage Service (S3)

- Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web.
- Write, read, and delete objects containing from 1 byte to 5 terabytes of data each. The number of objects you can store is unlimited.
- Each object is stored in a bucket and retrieved via a unique, developer-assigned key.
- A bucket can be stored in one of several Regions.
- You can choose a Region to optimize for latency, minimize costs, or address regulatory requirements.
- Objects stored in a Region never leave the Region unless you transfer them out.
- Authentication mechanisms are provided to ensure that data is kept secure from unauthorized access.
- Objects can be made private or public, and rights can be granted to specific users.
- S3 charges based on per GB-month AND per I/O requests AND per data modification requests.

# Centrally managed via consoles

The screenshot shows the AWS Management Console interface for the Amazon EC2 service. A red arrow points from the text "Navigation bar" to the top navigation bar which includes links for Elastic Beanstalk, S3, EC2, VPC, CloudWatch, Elastic MapReduce, Lambda, CloudFront, CloudFormation, RDS, ElastiCache, SQS, IAM, SNS, and SES. Another red arrow points from the text "Navigation Pane" to the left sidebar titled "Navigation". The sidebar contains several sections: EC2 Dashboard, INSTANCES (with "Instances" highlighted in orange), IMAGES (AMIs), ELASTIC BLOCK STORE (Volumes, Snapshots), and NETWORK & SECURITY (Security Groups, Elastic IPs, Placement Groups, Load Balancers, Key Pairs). A third red arrow points from the text "Current page" to the main content area, specifically the "My Instances" table where one instance is listed. The instance details are shown in a modal dialog at the bottom right.

**Navigation bar**

**Navigation Pane**

**Current page**

**AWS Management Console > Amazon EC2**

**Christophe Coenraets**

**Navigation**

**Region:** US East (Virginia)

**INSTANCES**

**Instances**

Spot Requests  
Reserved Instances

**IMAGES**

AMIs  
Bundle Tasks

**ELASTIC BLOCK STORE**

Volumes  
Snapshots

**NETWORK & SECURITY**

Security Groups  
Elastic IPs  
Placement Groups  
Load Balancers  
Key Pairs

**My Instances**

**Viewing:** All Instances

**Name** **Instance** **AMI ID** **Root Device** **Type** **Status** **Security Groups**

Name	Instance	AMI ID	Root Device	Type	Status	Security Groups
empty	i-ab059cc8	ami-1b614f72	ebs	t1.micro	running	quick-start-1

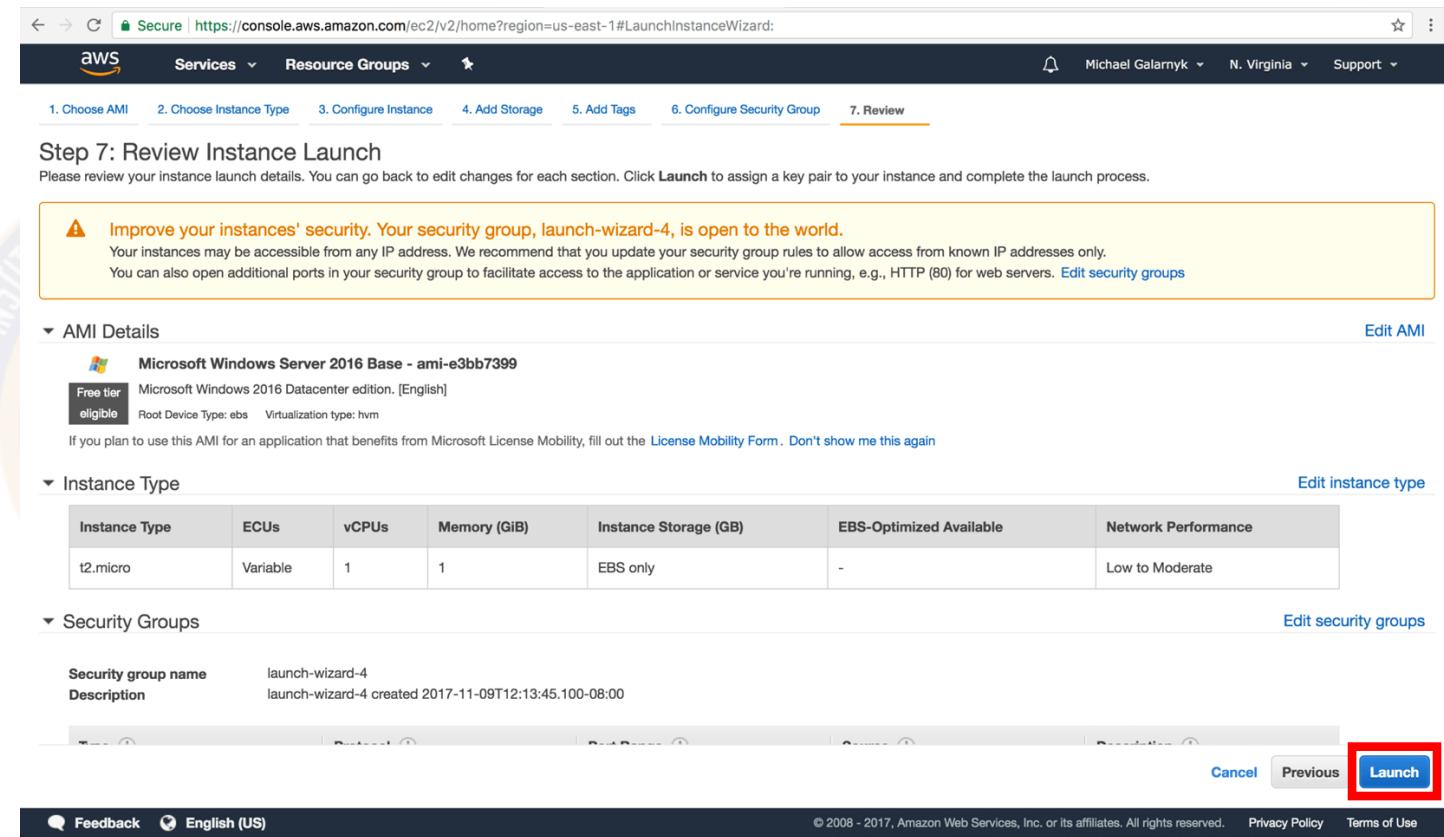
**1 EC2 Instance selected.**

**EC2 Instance: i-ab059cc8 ec2-50-17-14-16.compute-1.amazonaws.com**

Description	Monitoring	Tags
AMI: amzn-ami-2011.09.2.x86_64-ebs (ami-1b614f72)	Zone: us-east-1d	
Security Groups: quick-start-1	Type: t1.micro	
Status: running	Owner: 873349114418	
VPC ID: -	Subnet ID: -	
Source/Dest. Check: -	Virtualization: paravirtual	
Placement Group: -	Reservation: r-36edc558	
RAM Disk ID: -	Platform: -	
Key Pair Name: christophe	Kernel ID: aki-825ea7eb	
Monitoring: basic	AMI Launch Index: 0	
Elastic IP: -	Root Device: sda1	
Root Device Type: ebs	Tenancy: default	
Lifecycle: normal		
Block Devices: sda1		
<b>Public DNS:</b> ec2-50-17-14-16.compute-1.amazonaws.com		
<b>Private DNS:</b> ip-10-98-187-15.ec2.internal		

# Compute (1): ec2

- Pick from a set of machine specs + OS / container / container cluster and launch within a region / AZ
- Configure security profile to allow/deny specific traffic - Firewall rules
- Can choose multiple instances across regions / AZ for fault tolerance
- Use elastic IP to quickly reassigned instances in other AZ and stay connected
- Create new instances on the fly using Auto-scaling or manually add new instances to a Load Balancer

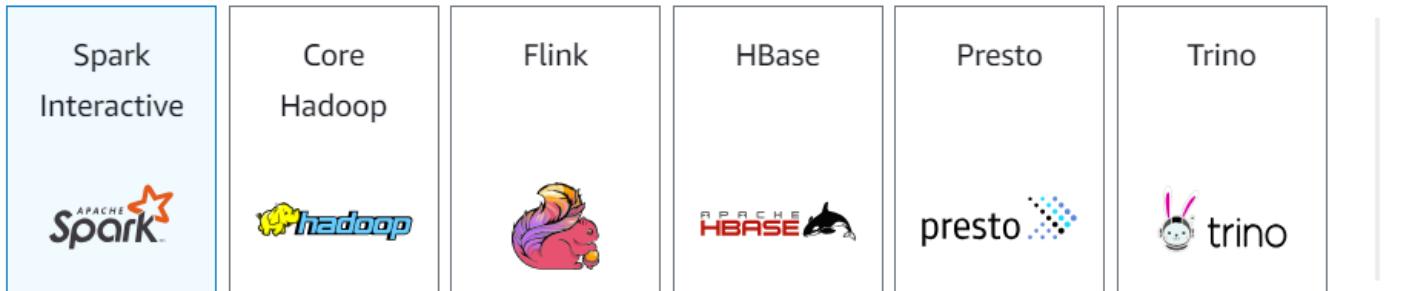


# Compute (2) AWS – Elastic Map Reduce (EMR)

- Amazon EMR is a web service that makes it easy to quickly and cost-effectively process vast amounts of data using Hadoop.
- Amazon EMR distribute the data and processing across a resizable cluster of Amazon EC2 instances.
- With Amazon EMR you can launch a persistent cluster that stays up indefinitely or a temporary cluster that terminates after the analysis is complete.
- Amazon EMR supports a variety of Amazon EC2 instance types and Amazon EC2 pricing options (On-Demand, Reserved, and Spot).
- When launching an Amazon EMR cluster (also called a "job flow"), you choose how many and what type of Amazon EC2 Instances to provision.
- The Amazon EMR price is in addition to the Amazon EC2 price.
- Amazon EMR is used in a variety of applications, including log analysis, web indexing, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics.

# EMR – Bundle options

## Application bundle



- AmazonCloudWatchAgent  
1.300031.1
- HCatalog 3.1.3
- Hue 4.11.0
- Livy 0.7.1
- Phoenix 5.1.3
- Spark 3.5.0
- Tez 0.10.2
- ZooKeeper 3.5.10

- Flink 1.18.0
- Hadoop 3.3.6
- JupyterEnterpriseGateway 2.6.0
- MXNet 1.9.1
- Pig 0.17.0
- Sqoop 1.4.7
- Trino 426
- HBase 2.4.17
- Hive 3.1.3
- JupyterHub 1.5.0
- Oozie 5.2.1
- Presto 0.283
- TensorFlow 2.11.0
- Zeppelin 0.10.1

## AWS Glue Data Catalog settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

- Use for Hive table metadata
- Use for Spark table metadata

AWS demo video

<https://youtu.be/OLY-Dh2vE3w>

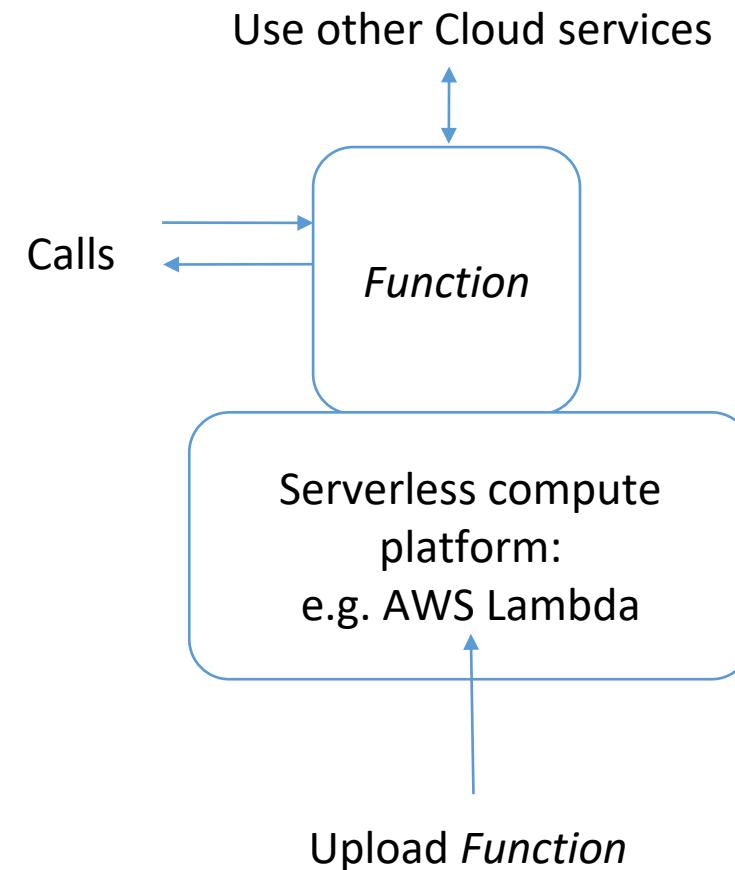
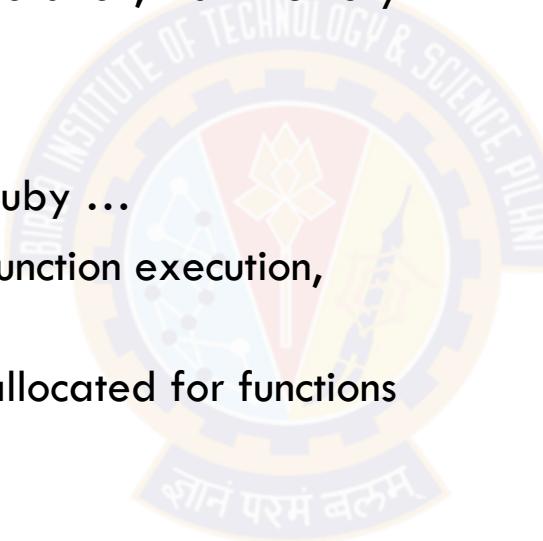
# Compute (3): Serverless and FaaS

- Creating and running a Cloud application needs infrastructure - core code, libraries, integrations, event triggers, messaging, databases, app servers, web front ends, containers, networking etc.
  - ✓ Users still have to manage / configure some of this even in a Cloud platform
- How can we simplify this —
  - ✓ Simplify Infrastructure management and servers, esp in large scale deployments
    - Adopt serverless platforms
    - E.g. we can use a server less DB or a Kube cluster and leave management, scaling, config to service provider
  - ✓ Just focus on core code and micro-services that can run on a stateless compute platform
    - Adopt function-as-a-service (FaaS) and event-driven programming model

<https://martinfowler.com/articles/serverless.html>

# FaaS (1)

- FaaS computing - a new way to host code at function level where user is abstracted away from the underlying infrastructure needs to run the function
  - ✓ Pricing could be in terms of #requests, duration, max memory
  - ✓ e.g. AWS Lambda
- Stateless functions
- Pick from a set of runtimes - Node, Python, Ruby ...
- Specify memory requirements, timeouts for function execution, where to get data from etc.
- FaaS platform has some limits on resources allocated for functions
- State has to be managed externally
- Not meant for complex functionalities
- Need to manage multiple functions



# FaaS (2)

- Functions execute when triggered by events, e.g.
  - ✓ A REST API call to an API gateway
  - ✓ A file upload, e.g. an image upload to AWS S3 bucket
  - ✓ A timer expired, e.g. in AWS CloudWatch
  - ✓ A message arrival event in a queuing service
  - ✓ An alert or notification, e.g. in AWS SNS
  - ✓ ...
- Functions can use storage and DB services, SNS etc. for input or output

# Topics for today

- Cloud concepts
- Compute instances - AWS
  - ✓ EC2
  - ✓ EMR
  - ✓ Lambda
- **AWS demos**
- Scalability and Elasticity
- Dynamic provisioning
- Multi-Tenant Data Architecture



# AWS Demos

- S3
- EC2
- EMR
- LAMBDA



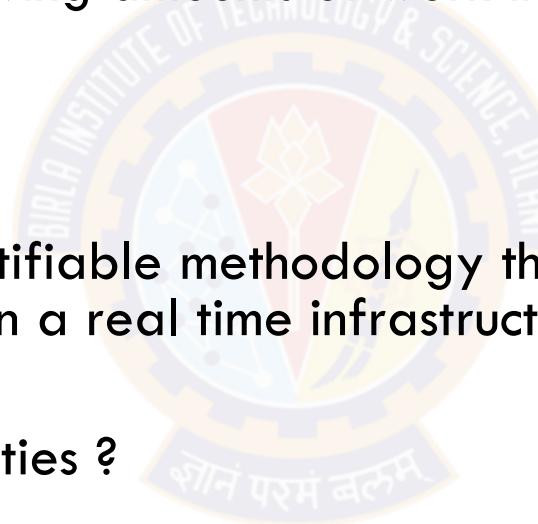
# Topics for today

- Cloud concepts
- Compute instances - AWS
  - ✓ EC2
  - ✓ EMR
  - ✓ Lambda
- AWS demos
- **Scalability and Elasticity**
- Dynamic provisioning
- Multi-Tenant Data Architecture

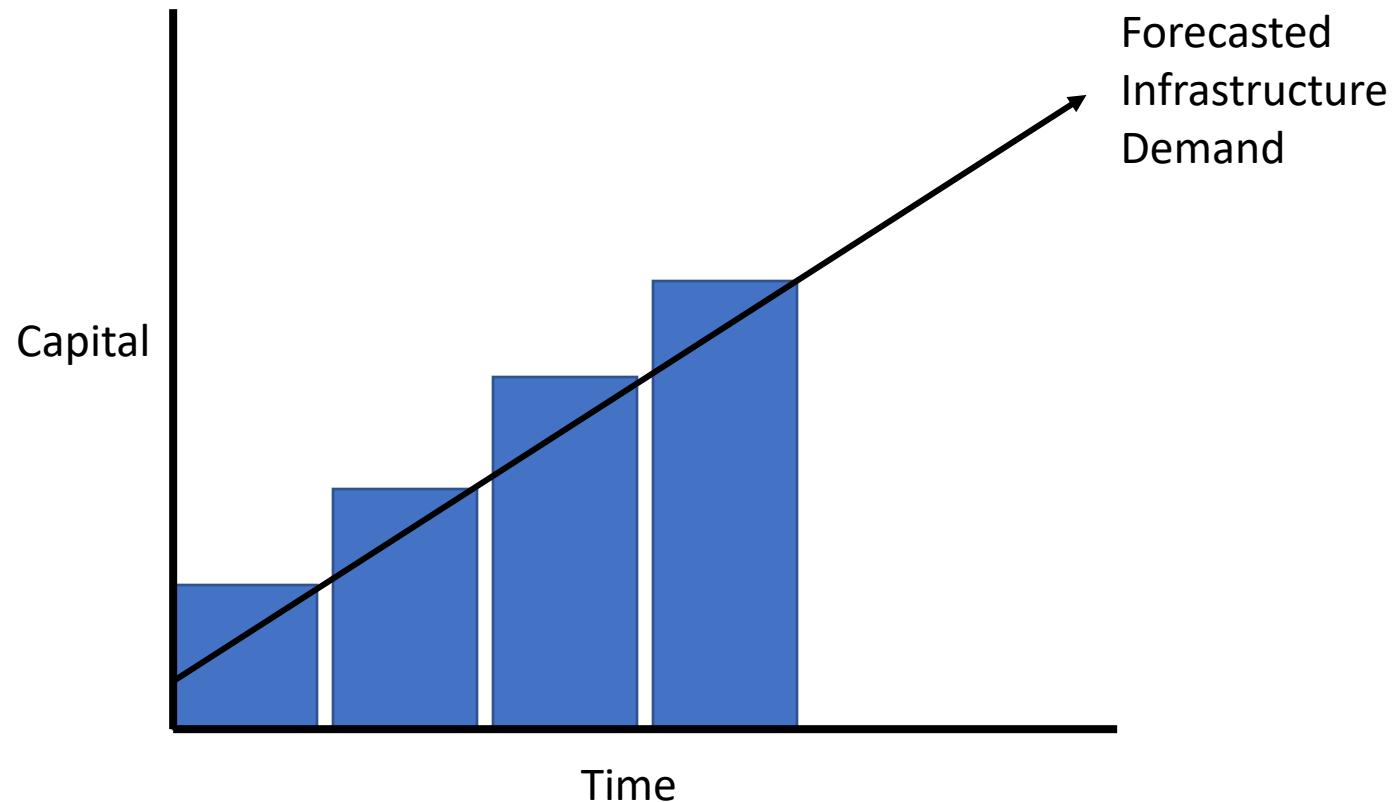


# Scalability & Elasticity

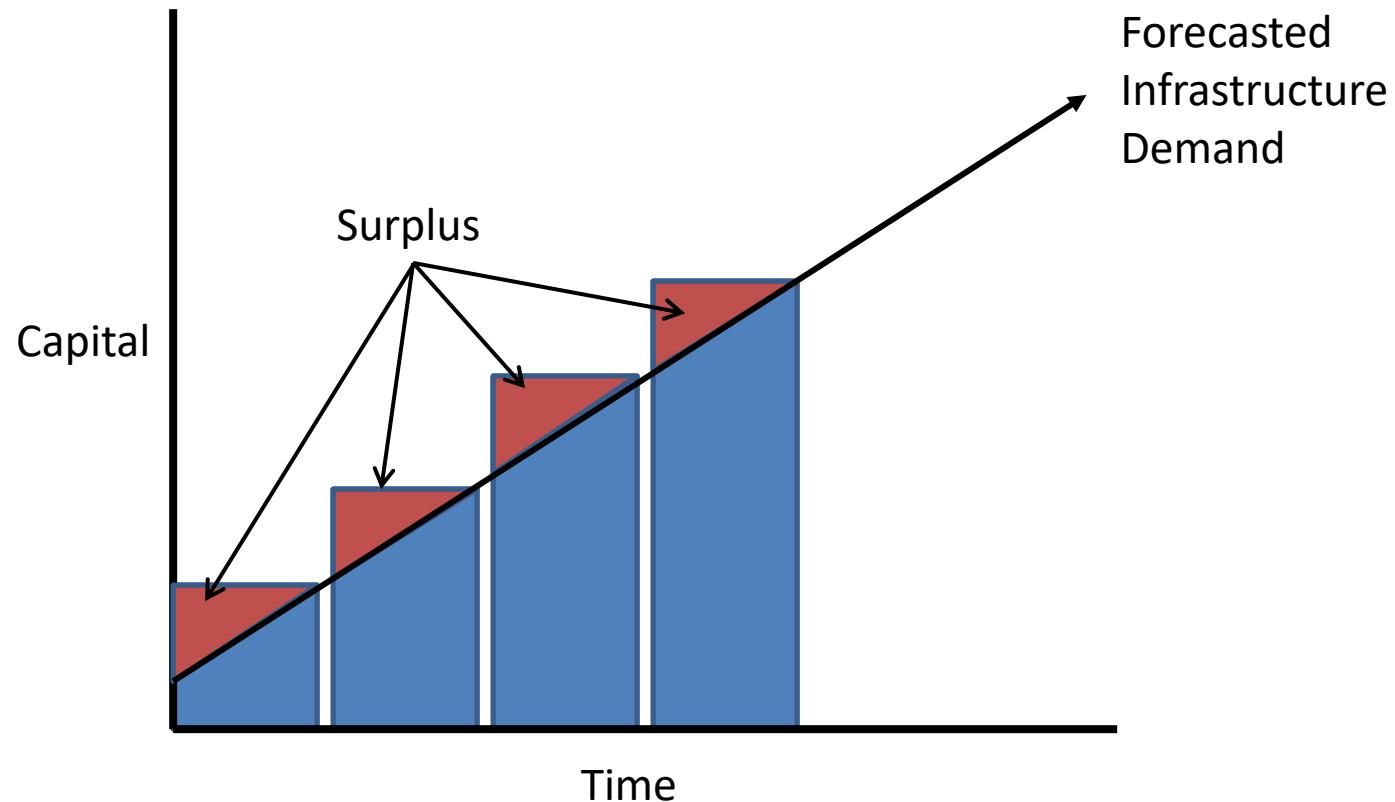
- What is scalability ?
  - ✓ A desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged.
- What is elasticity ?
  - ✓ The ability to apply a quantifiable methodology that allows for the basis of an adaptive introspection within a real time infrastructure.
- But how to achieve these properties ?
  - ✓ Dynamic provisioning
  - ✓ Multi-tenant design



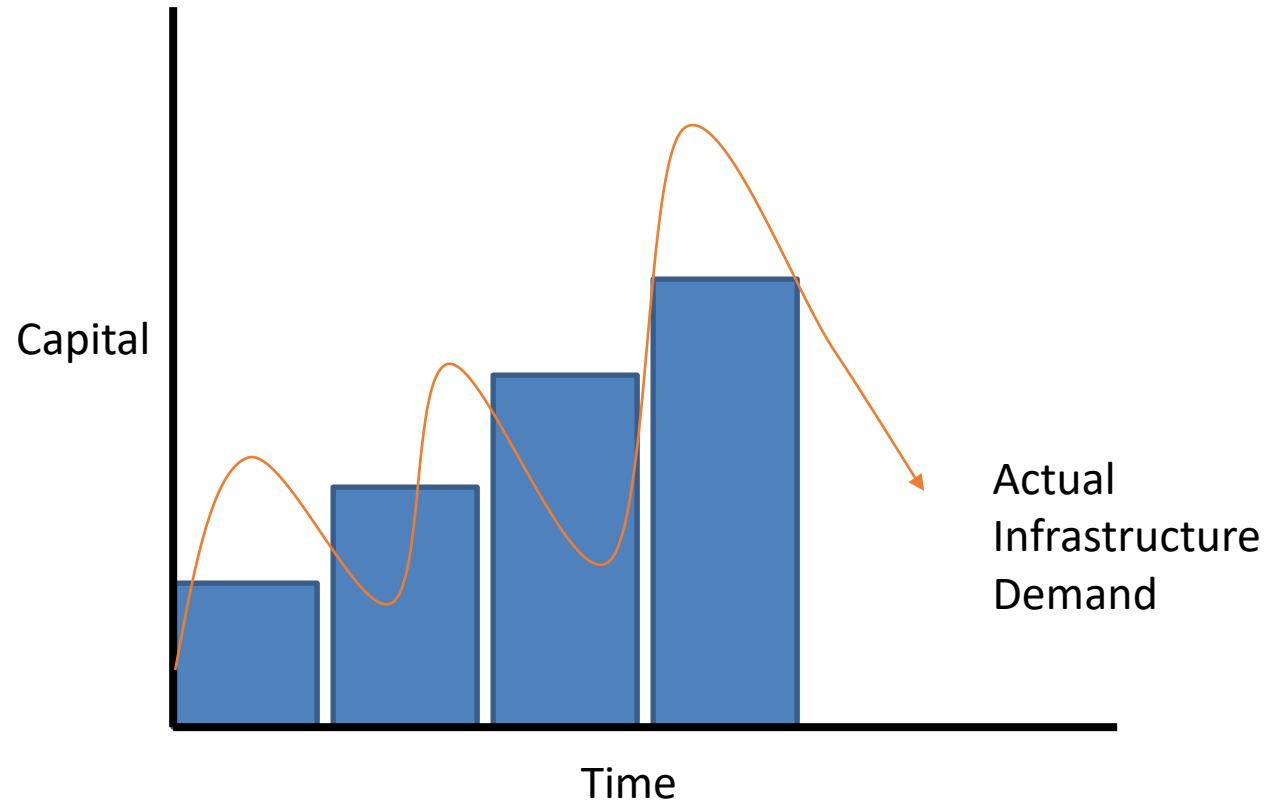
# Traditional Infrastructure Model



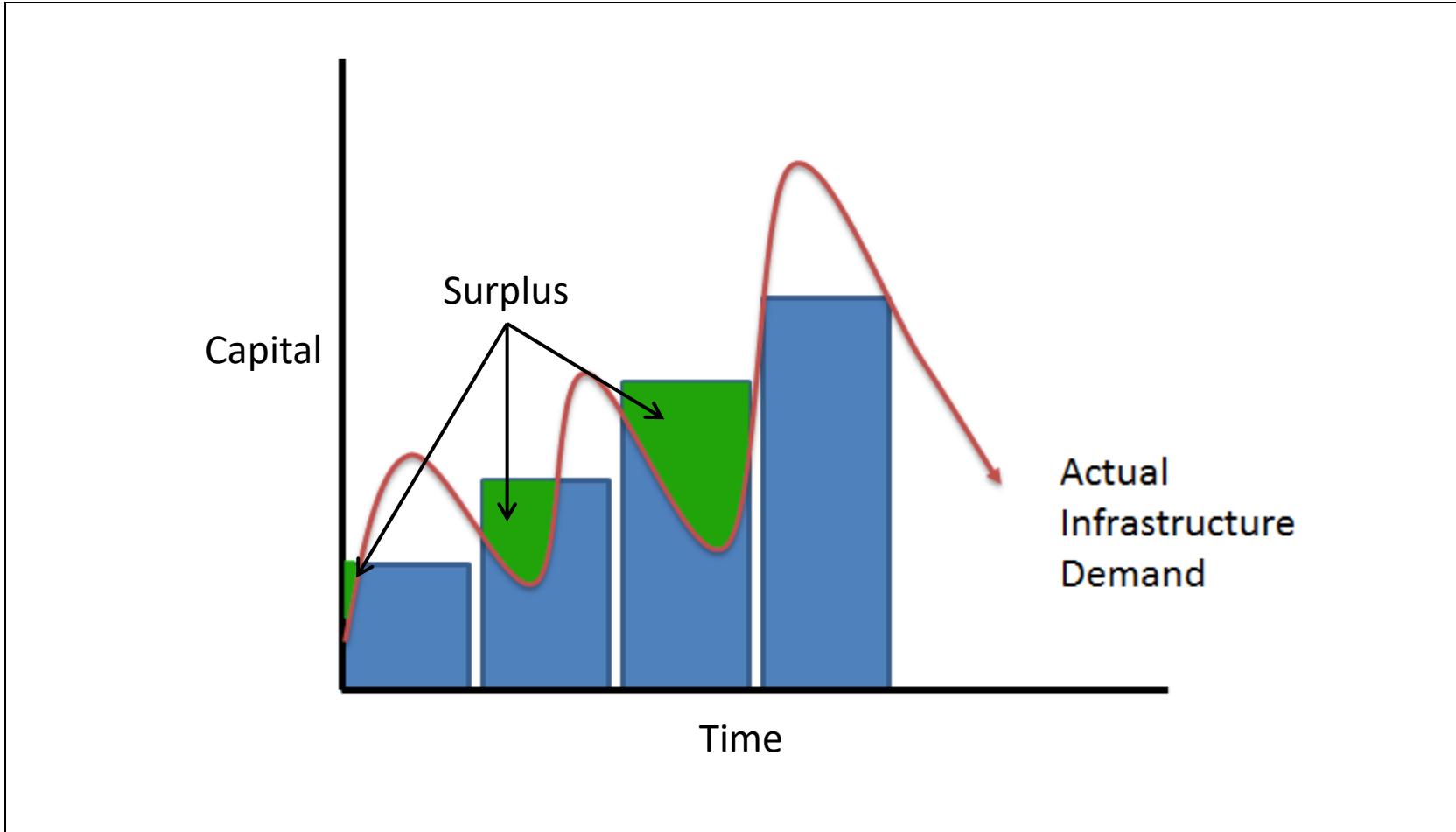
# Acceptable Surplus



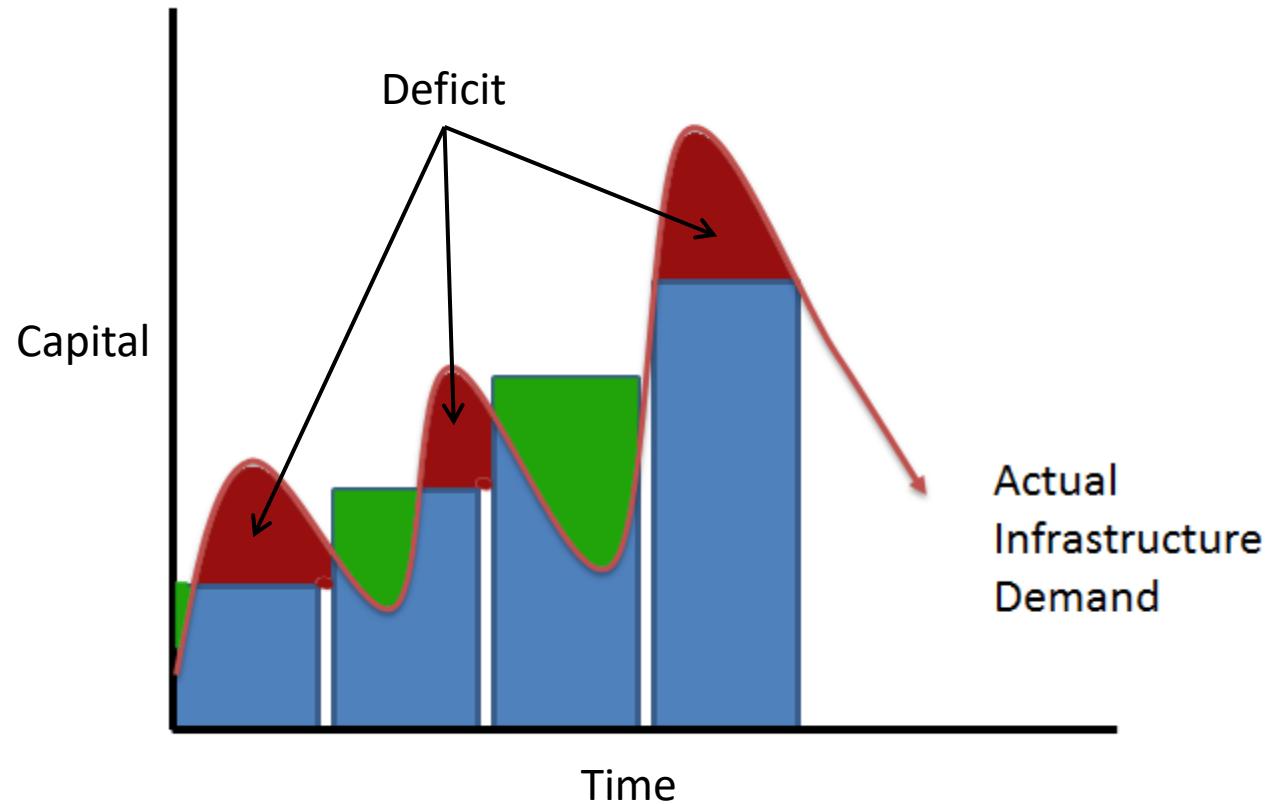
# Actual demand



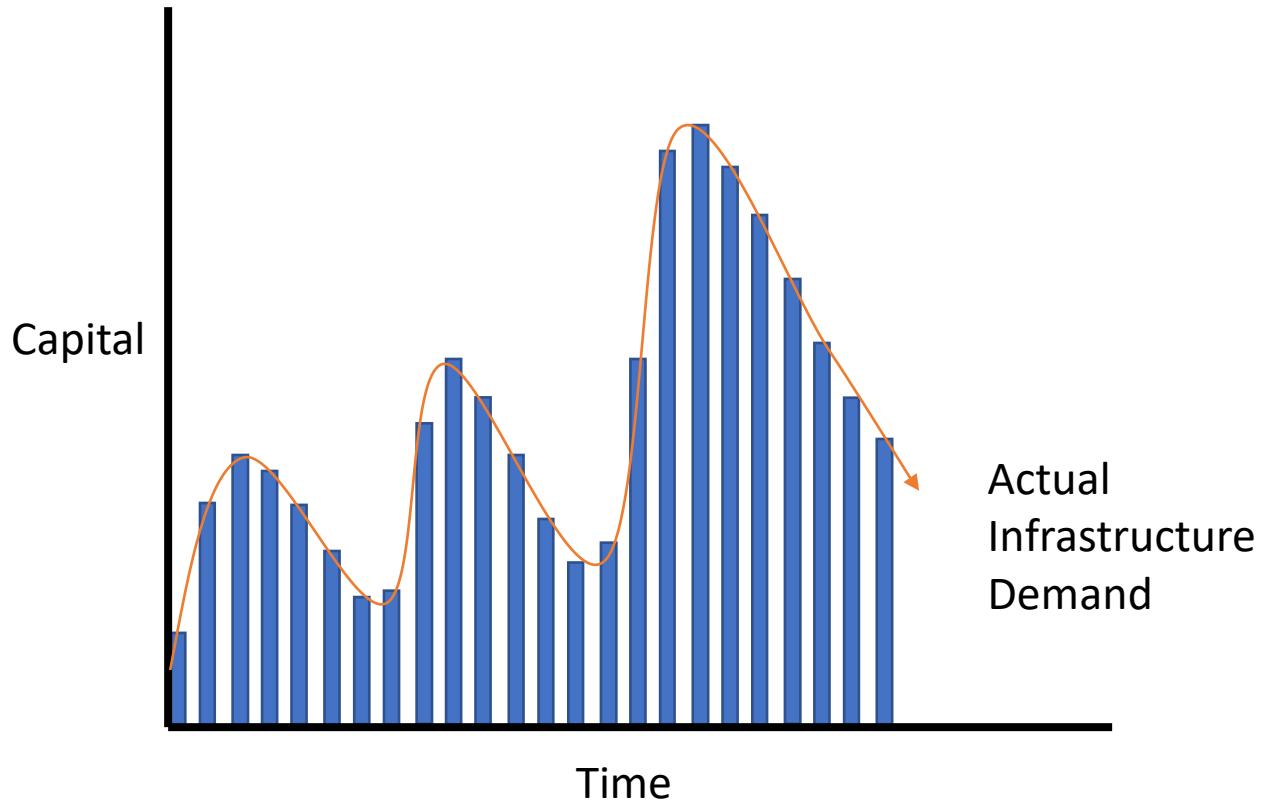
# Unacceptable Surplus



# Unacceptable deficit



# Utility Infrastructure Model



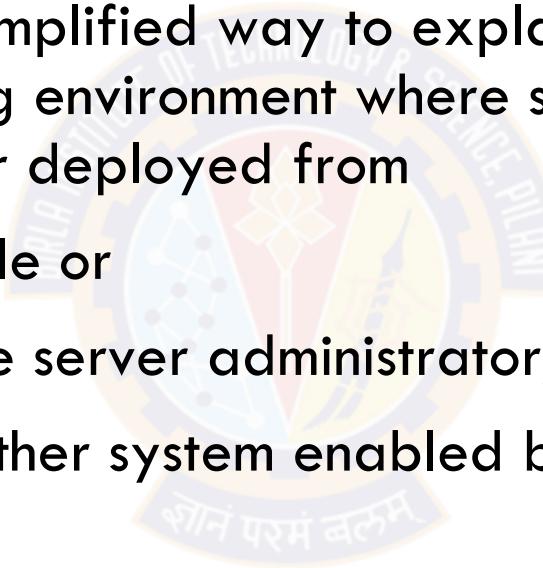
# Topics for today

- Cloud concepts
- Compute instances - AWS
  - EC2
  - EMR
  - Lambda
- AWS demos
- Scalability and Elasticity
- **Dynamic provisioning**
- Multi-Tenant Data Architecture



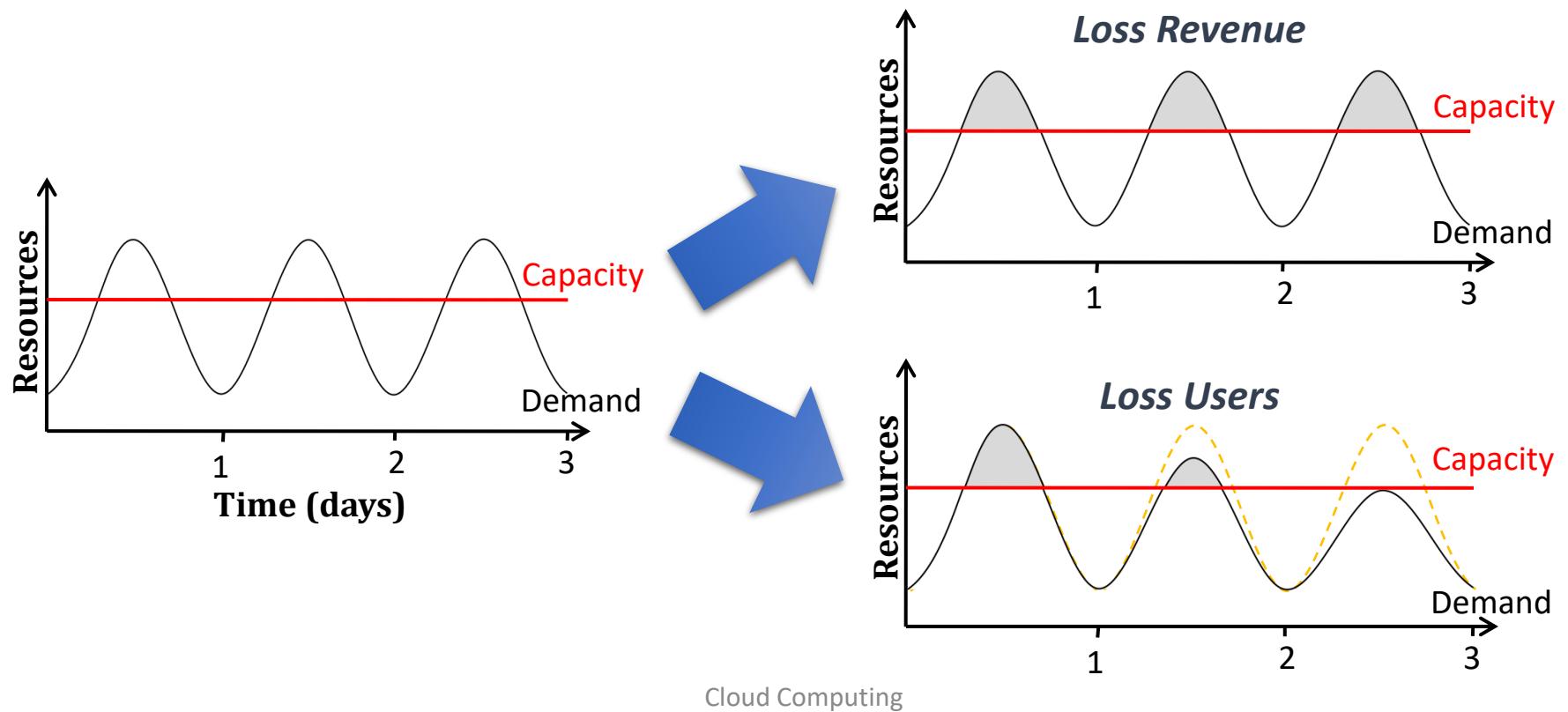
# Dynamic Provisioning (1)

- What is dynamic provisioning ?
  - ✓ Dynamic Provisioning is a simplified way to explain a complex networked server computing environment where server computing instances are provisioned or deployed from
    - an administrative console or
    - client application by the server administrator, network administrator
    - automatically by any other system enabled by user.



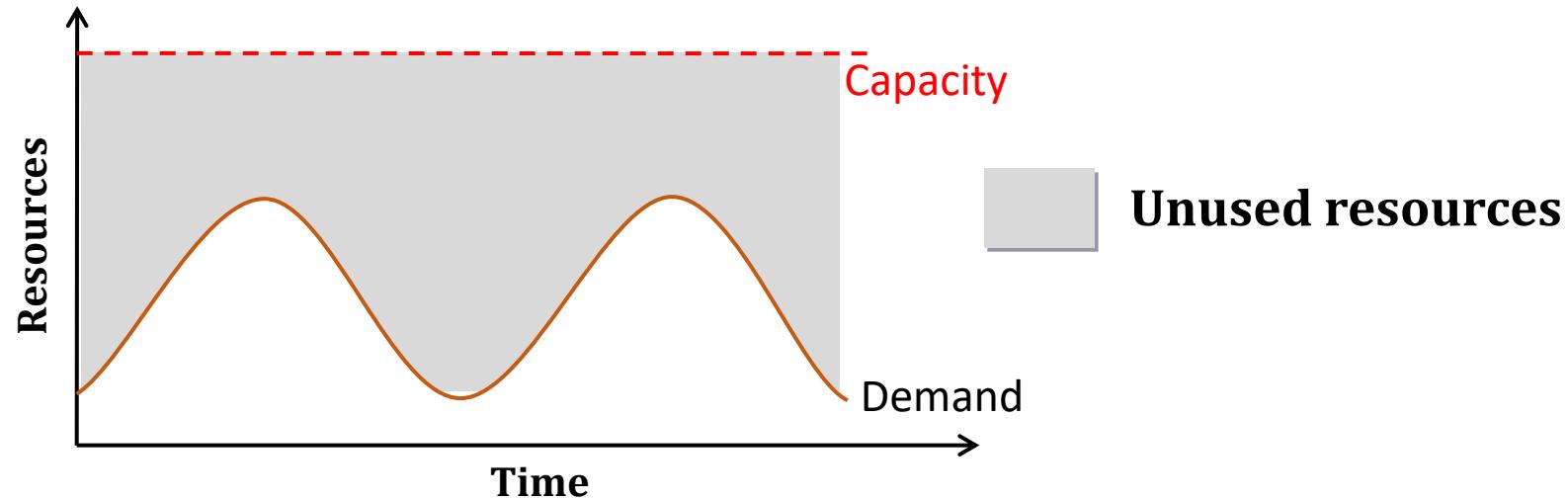
# Dynamic Provisioning (2)

- In traditional computing model, two common problems :
  - Underestimate system utilization which result in under provision



# Dynamic Provisioning (3)

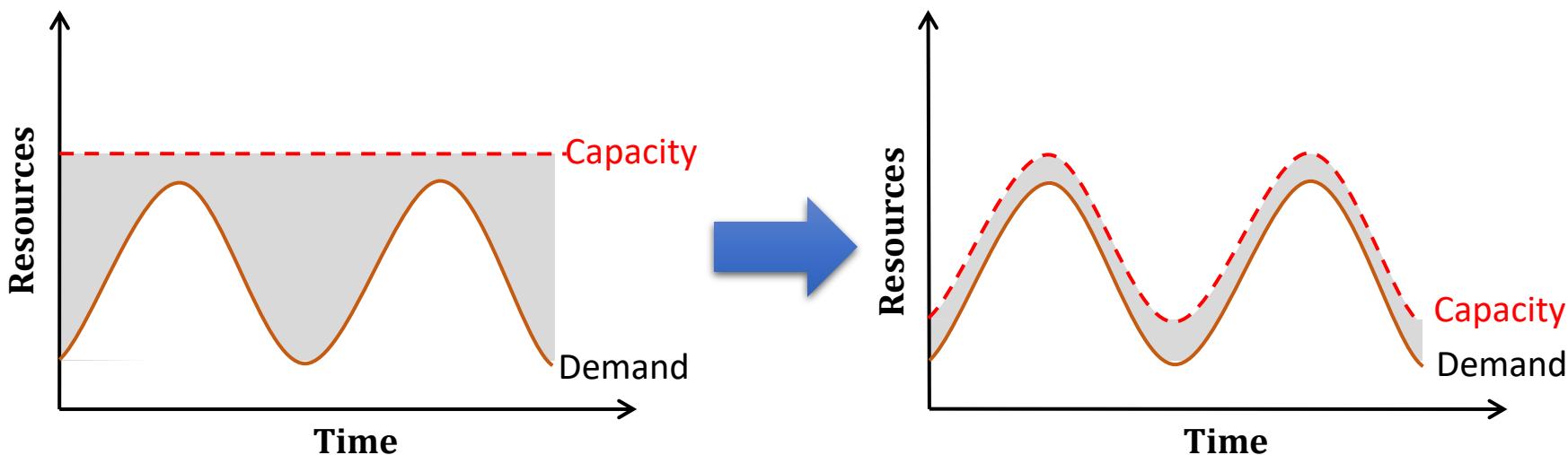
- Overestimate system utilization which result in low utilization



- How to solve this problem ??
- Dynamically provision resources

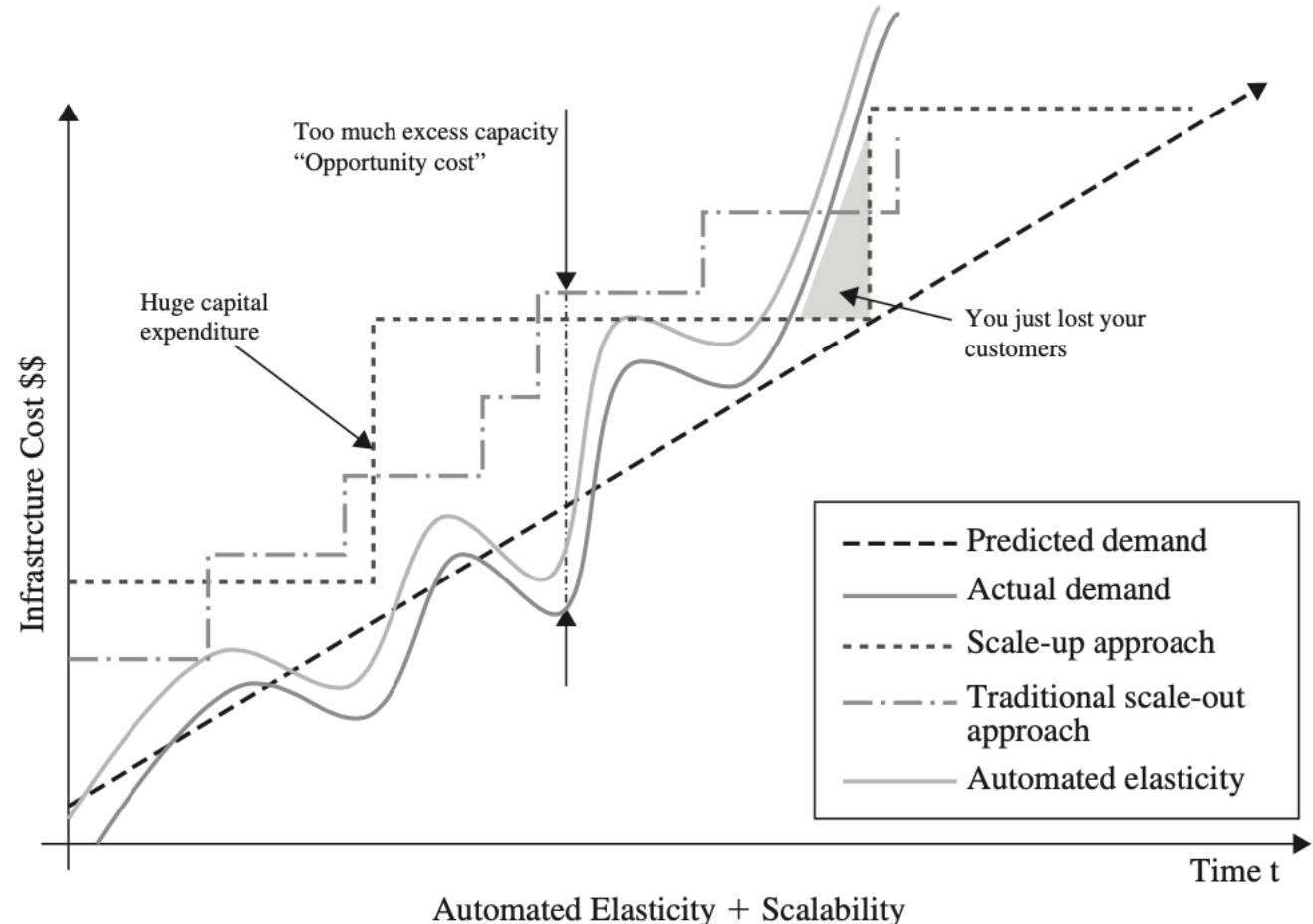
# Dynamic Provisioning (4)

- Cloud resources should be provisioned dynamically
  - Meet seasonal demand variations
  - Meet demand variations between different industries
  - Meet burst demand for some extraordinary events



# Automated elasticity

- Applications can be provisioned with resources on-demand when they need it
- Re-think application design
  - ✓ Understand which application components can be elastic
  - ✓ What is the impact on overall application being elastic



# Design for failure (e.g. in AWS)

- Use Elastic IP, a static IP that can be mapped to different servers
- Use multiple AZ (logical DCs) during deployment - create replicas across AZ
- Maintain images (Amazon Machine Images) for replication / cloning, quick restoration
- Use monitoring services, e.g. AWS CloudWatch
- Create Auto Scaling groups to replace failed instances quickly
- Take snapshots and backup on external storage, e.g. S3



# Automation of infrastructure

- Key to auto-scaling .. create auto-scaling groups for different clusters
- Monitor for resources (CPU, mem, I/O) and take actions like using machine images to launch new instances or sending notifications
- Store config data used for automation in a DB
- Build process : latest binaries go on a global external storage (e.g. AWS S3) - so any instance can spin up using latest build
- Open source config management tools: Chef, Puppet, Ansible etc.
- Build machine images with minimum OS for quick deployment. Configs and user details can be passed on and after launch.
- Bootup from 1 or more attached block storage volumes, e.g. EBS volumes attached to EC2 instance
- App components should not depend on location, hardware, IP addresses because the image/binary can be moved anywhere on failure

# Topics for today

- Cloud concepts
- Compute instances - AWS
  - ✓ EC2
  - ✓ EMR
  - ✓ Lambda
- AWS demos
- Scalability and Elasticity
- Dynamic provisioning
- **Multi-Tenant Data Architecture**



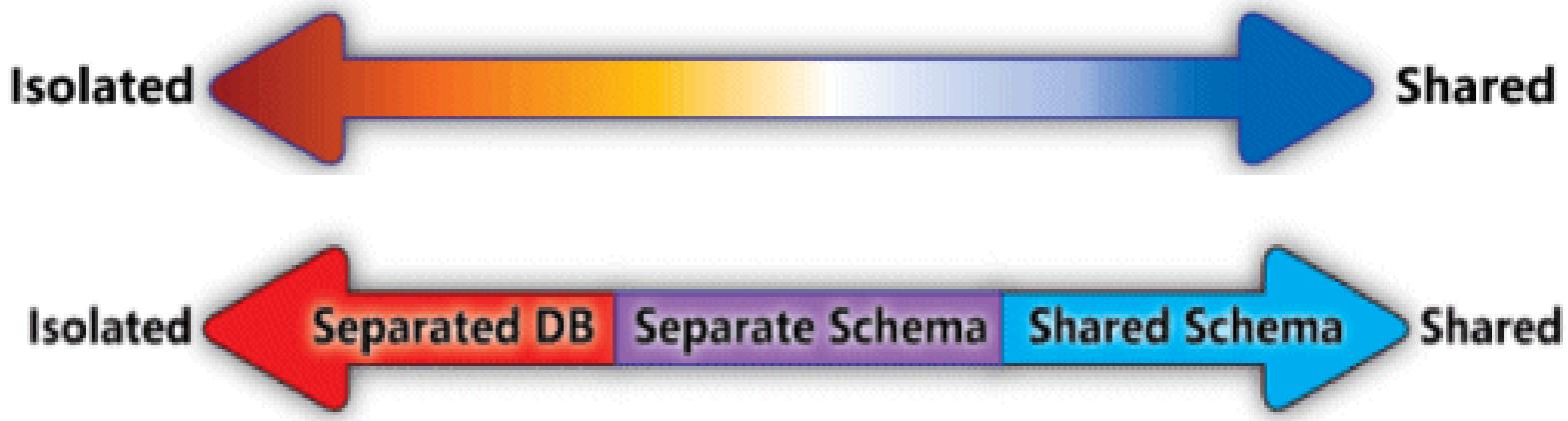
# Multi Tenancy

A Multi-tenant application lets customers share the same hardware resources, by offering them one shared application and **database instance**, while allowing them to configure the application to fit their needs as if it runs on dedicated environment.

These definitions focus on what we believe to be the key aspects of multi-tenancy:

1. The ability of the application to share hardware resources.
2. The offering of a high degree of configurability of the software.
3. The architectural approach in which the tenants make use of a single application and database instance.

# Multi-Tenant Data (MTD) Architecture



## Separate Database

Traditional – Isolated Database Instance Per Customer (Tenant)

## Shared Database Separate Schema

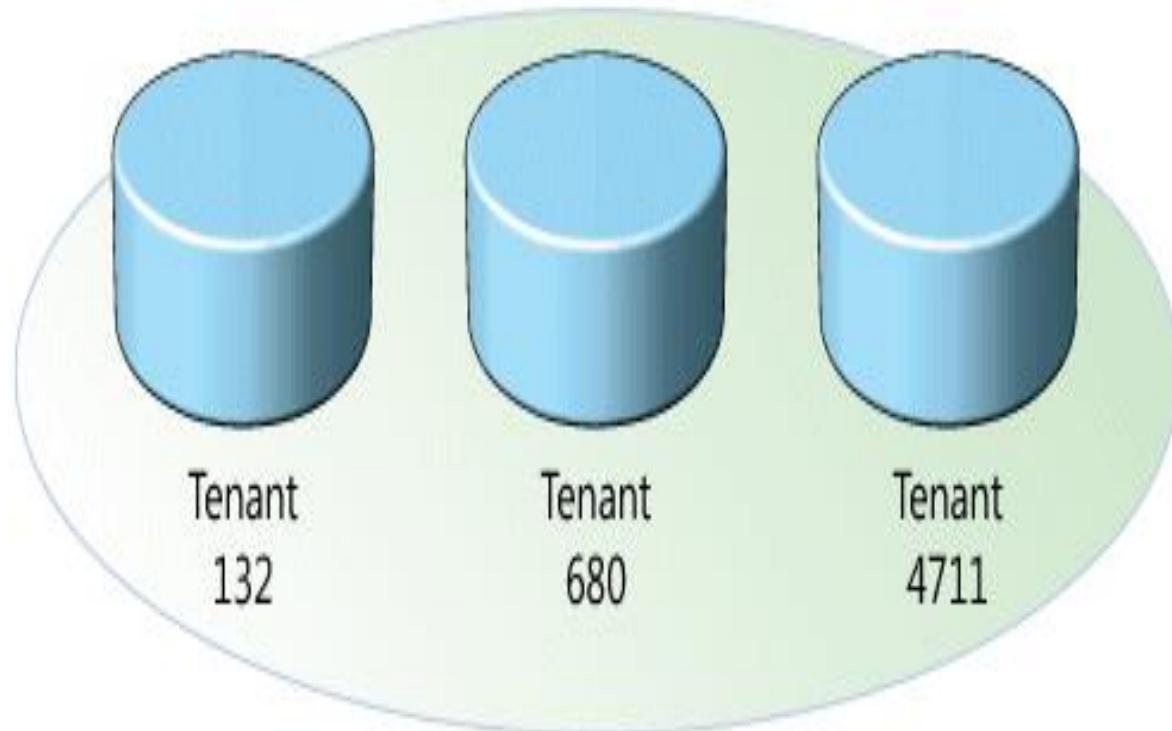
Customers get their own schema but are co-hosted in the same database

## Shared Database – Shared Schema

Drives the highest efficiency. All Customers data is stored in the same database and schema with a **tenant id** qualifier

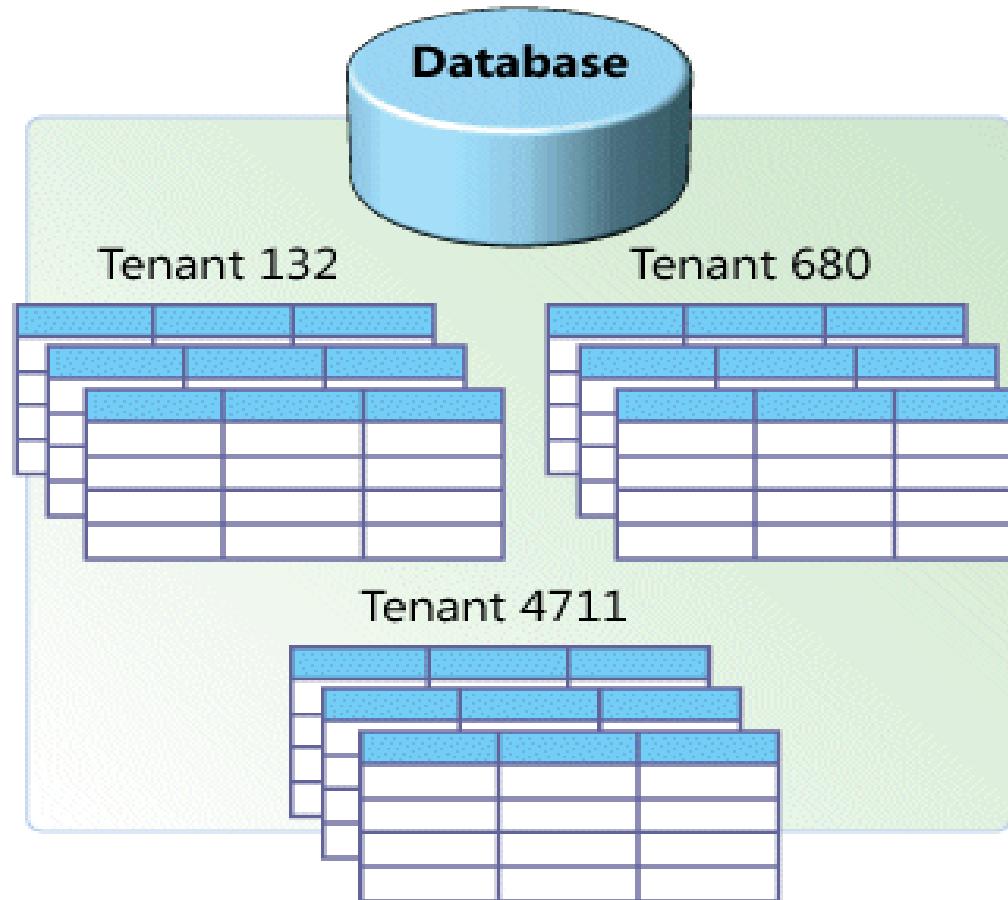
Source: [Multi Tenant Data Architecture](#), Frederick Chong, Gianpaolo Carraro, and Roger Wolter Microsoft Corporation

# Separate Databases



Different Database for each Tenant

# Shared Database, Separate Schema



- Each tenant has its own separate set of tables in a common database

# Shared Database, Shared Schema

TenantID	CustName	Address		
4	TenantID	ProductID	ProductName	
1	4	TenantID	Shipment	Date
6	1	4711	324965	2006-02-21
4	6	132	115468	2006-04-08
	4	680	654109	2006-03-27
		4711	324956	2006-02-23

- All tenants share the same set of tables
- Tenant ID associates each tenant with the rows that it owns

# MTD for Classic Web Application

- ❑ Pack multiple tenants into the same tables by adding a tenant id column
- ❑ Great consolidation but no extensibility

Account			
TenId	AcctId	Name	...
17	1	Acme	
17	2	Gump	
35	1	Ball	
42	1	Big	

# Database Multi Tenancy Implementation

## Isolated Database and Shared Database – Separate Schema

Standard Data Access simply returns the appropriate connection based on tenant context

## Shared Database Shared Schema

### Approach 1

- Business Logic and Data Access is aware of multi tenant context and therefore query appropriately
  - Pros – Easy to build
  - Cons – High Probability of bugs leading to data leakage

### Approach 2

- Abstract Multi Tenancy concern to the Data Access Layer and write business logic without tenant context.
- Data Access Layer automatically adds tenant context to all data calls

- From a JDBC Perspective this implies different connection strings based on the customer.
- Typical Tenant Context is set by an intercepting filter and obtained at the DAO layer possibly via a ThreadLocal variable
- For Hibernate implement a Tenant aware ConnectionProvider and switch off the second level cache.

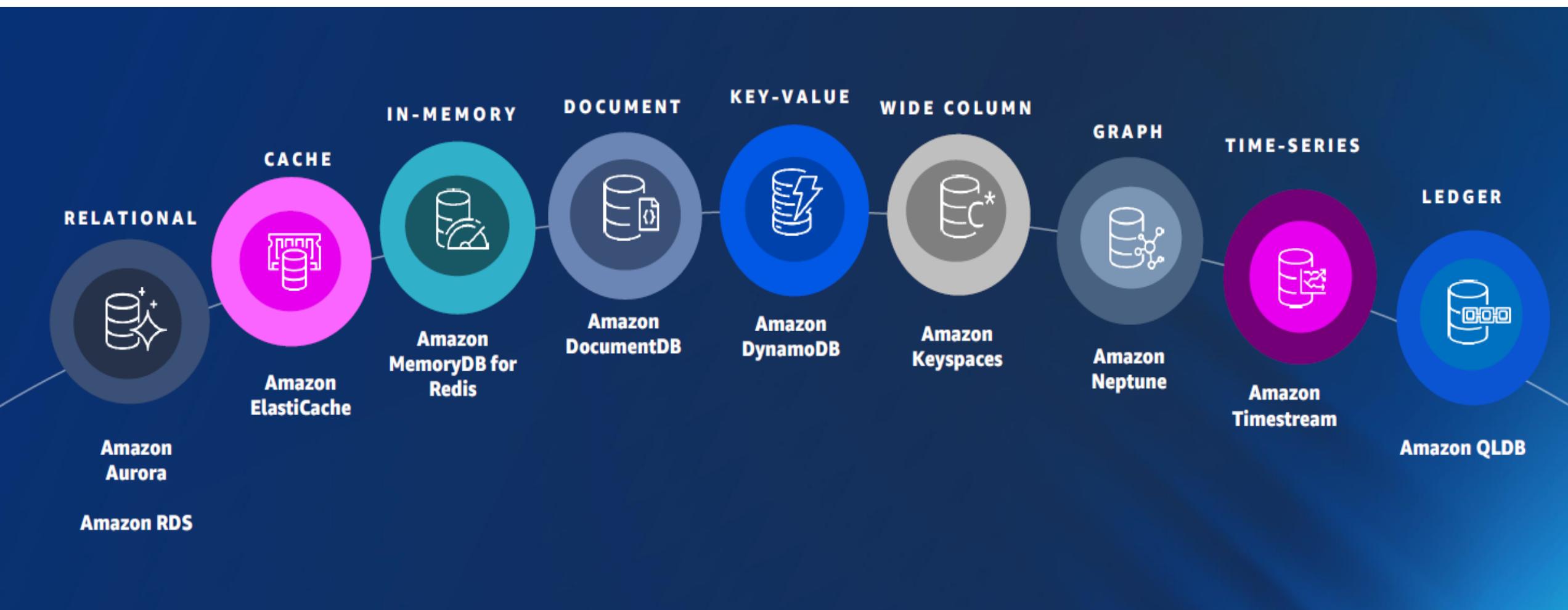
### For Hibernate

- Use Filters
- Use Hibernate Shards

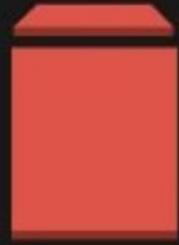
# Multi-Tenancy using NoSQL databases

- **Auto-Sharding**
  - Use Tenant ID as Partition Key and distribute data on separate nodes.
- **Flexible Schema**
  - Addition / removal of columns on the fly to tailor for the needs of each tenant.
- **Fault Tolerance**
  - Configurable replication factor based on tenant requirements
- **Denormalized data**
  - Need to manage only one column family / table for 1 tenant
- **Volume independent query performance**
  - Achieved by dynamic provisioning
- **Elastic scalability**
- **Facility to store Documents, Key-Values, Column families, Graphs etc**

# AWS Purpose built databases



# AWS Storage types



Amazon EBS  
(persistent)



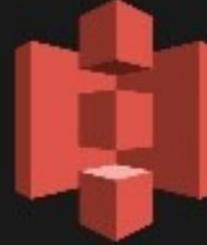
Amazon EC2  
Instance Store  
(ephemeral)

Block

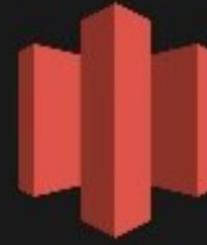


Amazon EFS

File



Amazon S3



Amazon Glacier

Object



AWS  
Snow Family



AWS Storage  
Gateway



EFS  
File Sync



3<sup>rd</sup> Party  
Connectors



AWS Direct  
Connect



S3 Transfer  
Acceleration

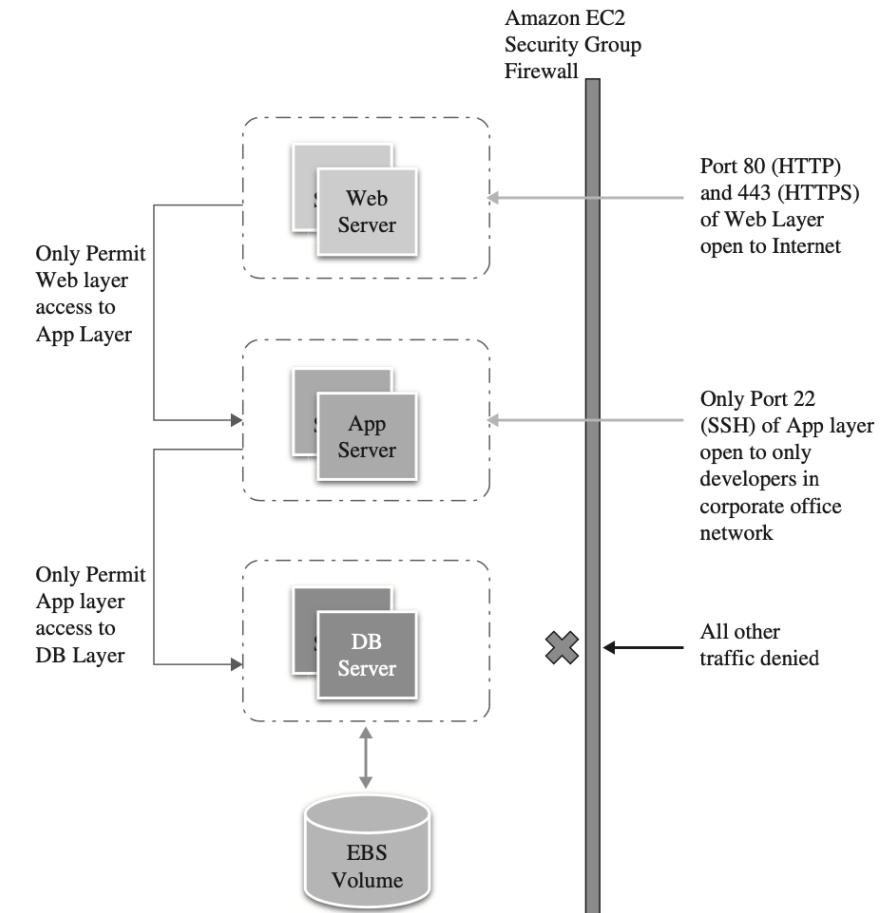
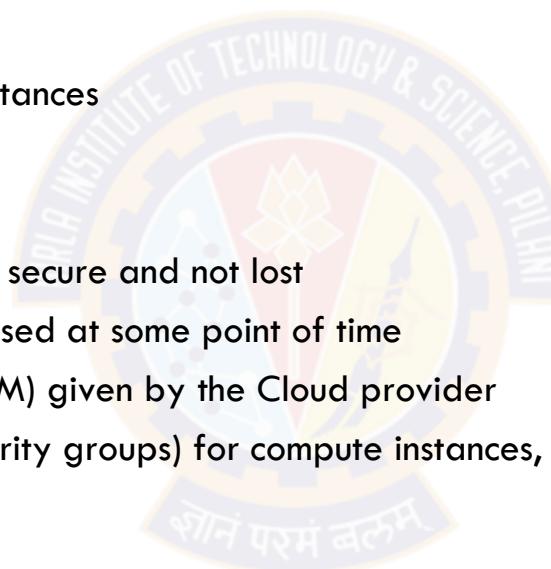


Amazon  
Kinesis

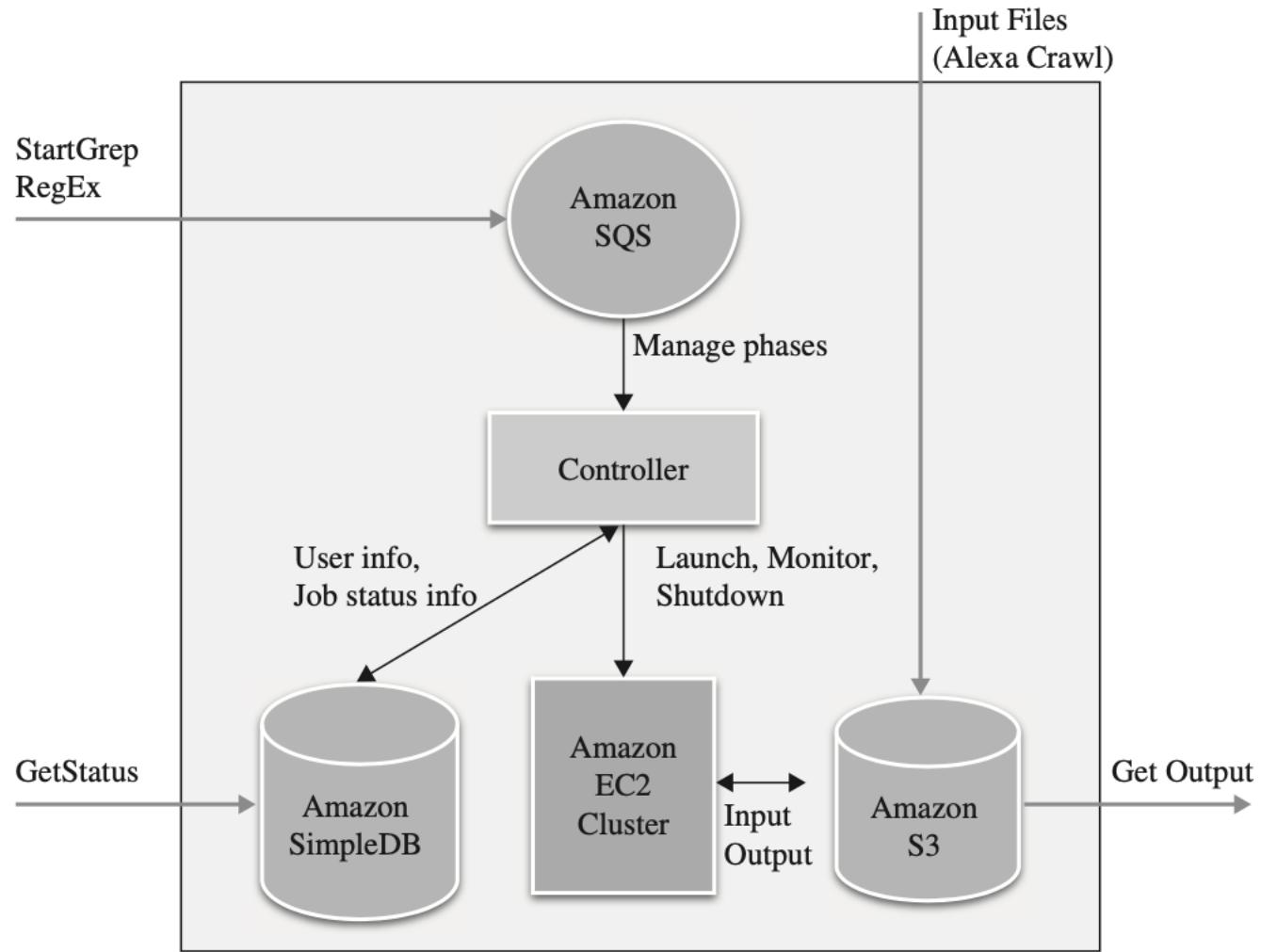
Data Transfer

# Security

- Traditional Enterprise perimeter security doesn't work on Cloud
- Security needed at every layer of application architecture
- Protect data in motion
  - ✓ Use SSL certificates and encryption in data movement
  - ✓ VPC to isolate within public cloud
  - ✓ Secure VPN for connecting on-prem to Cloud instances
- Protect data at rest
  - ✓ Can encrypt files or entire volumes
  - ✓ Key management is critical: Make sure keys are secure and not lost
  - ✓ Snapshot volumes for recovery, e.g. if compromised at some point of time
- Use the Identity and Access Management service (IAM) given by the Cloud provider
- Use instance specific virtual firewalls (e.g. AWS security groups) for compute instances, load balancers, DB etc.
- Use Network ACLs for subnet level access control
- Use Web App Firewall (WAF) along with application level load balancers to block application specific attacks

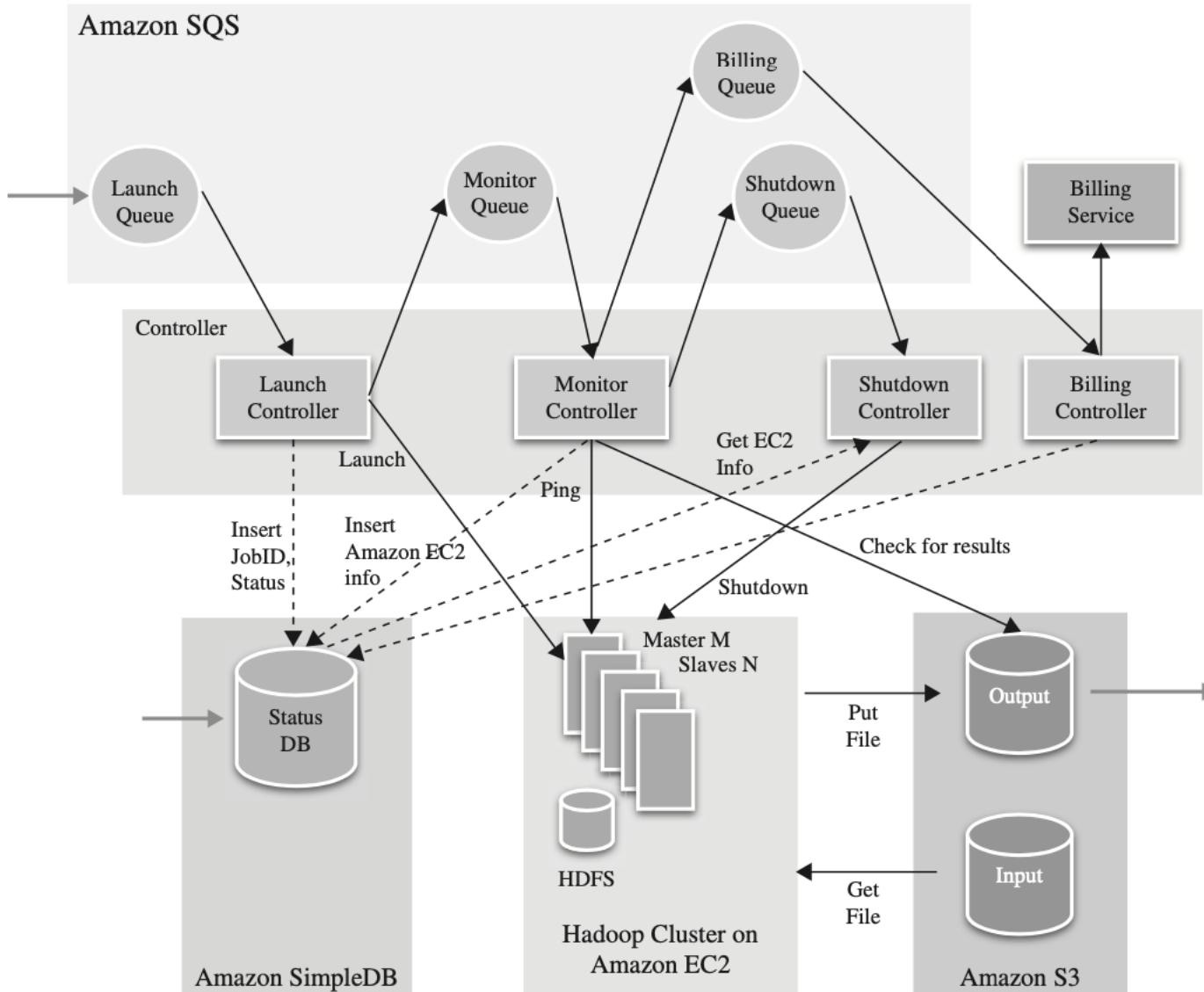


# Example application: GrepTheWeb



- Store crawled URLs on global storage
- Run a parallel search application for multiple users using Cloud best practices architecture
  - ✓ Queueing
  - ✓ Clustering
  - ✓ DB to manage meta-data
  - ✓ Monitor

# Example application: GrepTheWeb



# System design exercise

Design Instagram as a Cloud native Big Data Application.

How would you go about creating a design ?

Identify the Big Data requirements - what kind of DBs/Storage would you need ?

May not be just one.

What is the data consistency desired ?

What compute instances would you have ?

Any load balancers, messaging systems ?

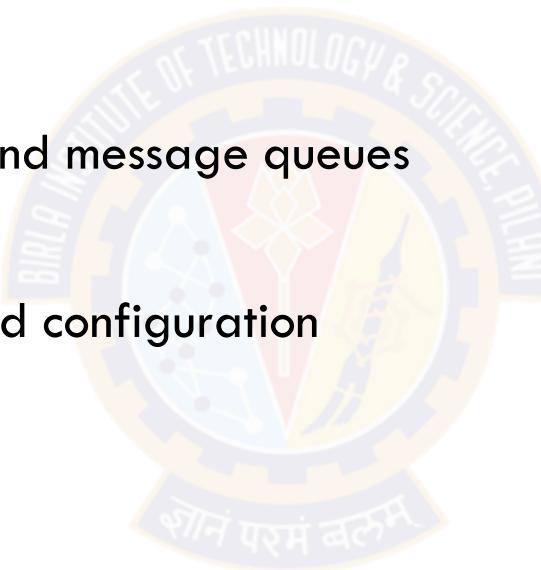
What about caching ?

How would you interact with users - Query? Notifications / Updates ?

What APIs would you have for users or internally ?

# Summary

- Different Cloud models and types of services
- Key best practices
  - ✓ Horizontal scale out
  - ✓ Layered security, esp for data
  - ✓ Application componentization and message queues
  - ✓ Design for failures
  - ✓ Automate build, deployment and configuration
  - ✓ Dynamic provisioning
  - ✓ Multi-Tenant Databases
- Additional Reading: AWS Case study from - “Cloud Computing Principles and Paradigms” (Wiley Series on Parallel and Distributed Computing), R. Buyya et al, Chapter 18



# Post mid-sem topics review

1. Data ingestion tools for moving data into and out of HDFS, Coordination and job orchestration tool
2. Storage layouts for different NoSQL stores - how would we model a data set in a NoSQL store - what about normalization, joins etc.
3. Spark internal architecture - resiliency and distribution, driver and cluster
4. Different types of operations in Spark - you should be aware as you write a Spark program on where shuffles happen and why a DAG is used, memory/storage optimizations discussed in class
5. Write basic Spark ML programs - know the basic flow and which functions solve which use cases
6. Cloud storage solutions - object store vs block storage vs file storage



Thank you

# Big Data Systems

## Webinar 1 -MongoDB

---

# Agenda

---

Introduction

Architecture

Important Features

NoSQL Database

MongoDB Vs RDBMS

CRUD Operations

MongoDB- Replication & Aggregation

# Introduction

---

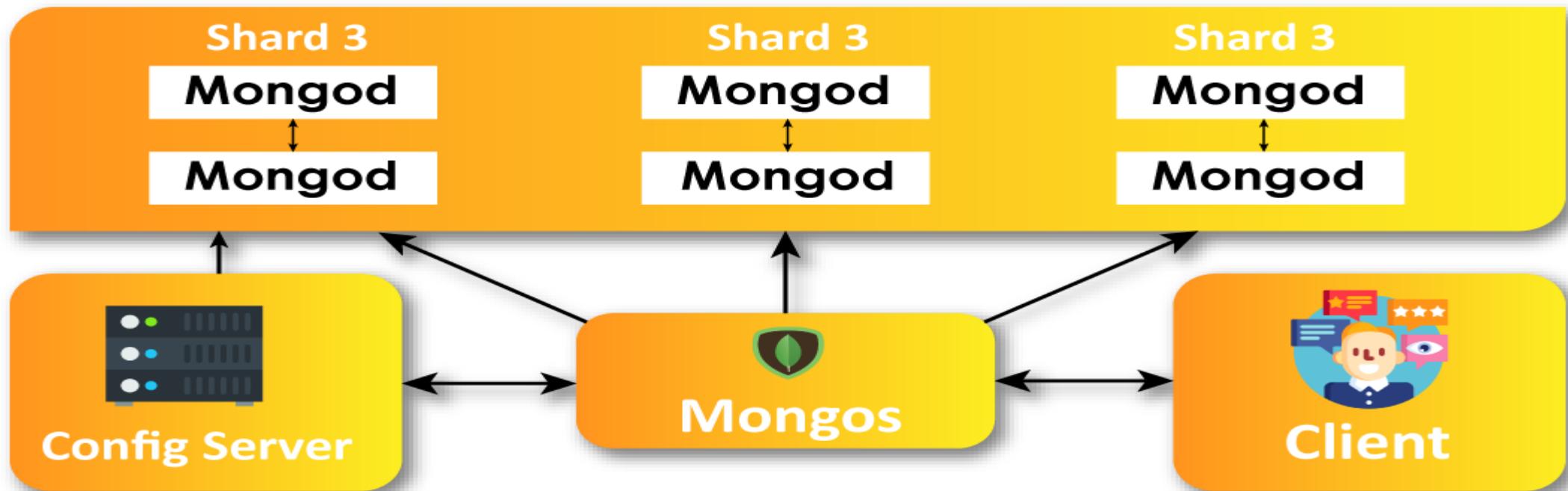
- MongoDB is a cross-platform, document oriented database
- It is a NoSQL database
- It is open source
- It provides high performance and scalability
- It stores data in the form of key/value pairs (document)
- It eliminates the need for Object Relational Mapping (ORM) in database development

---

MOngoDB stores data in form of BSON (binary JavaScript Object Notation) documents

```
{  
  name: "travis",  
  salary: 30000,  
  designation: "Computer Scientist",  
  teams: [ "front-end", "database" ]  
}
```

# MongoDB Architecture [1]



# Important Features of MongoDB?



- 
- **GridFS:** This feature will allow the files to divide into smaller parts and store them in different documents without complicating the stack.
  - **Schema-less Database:** MongoDB is a schema-less database programmed in C++ language.
  - **Document-oriented Storage:** It uses BSON format which is similar to JSON
  - **Procedures:** MongoDB JavaScript works better than procedures as databases use the language more than procedures.

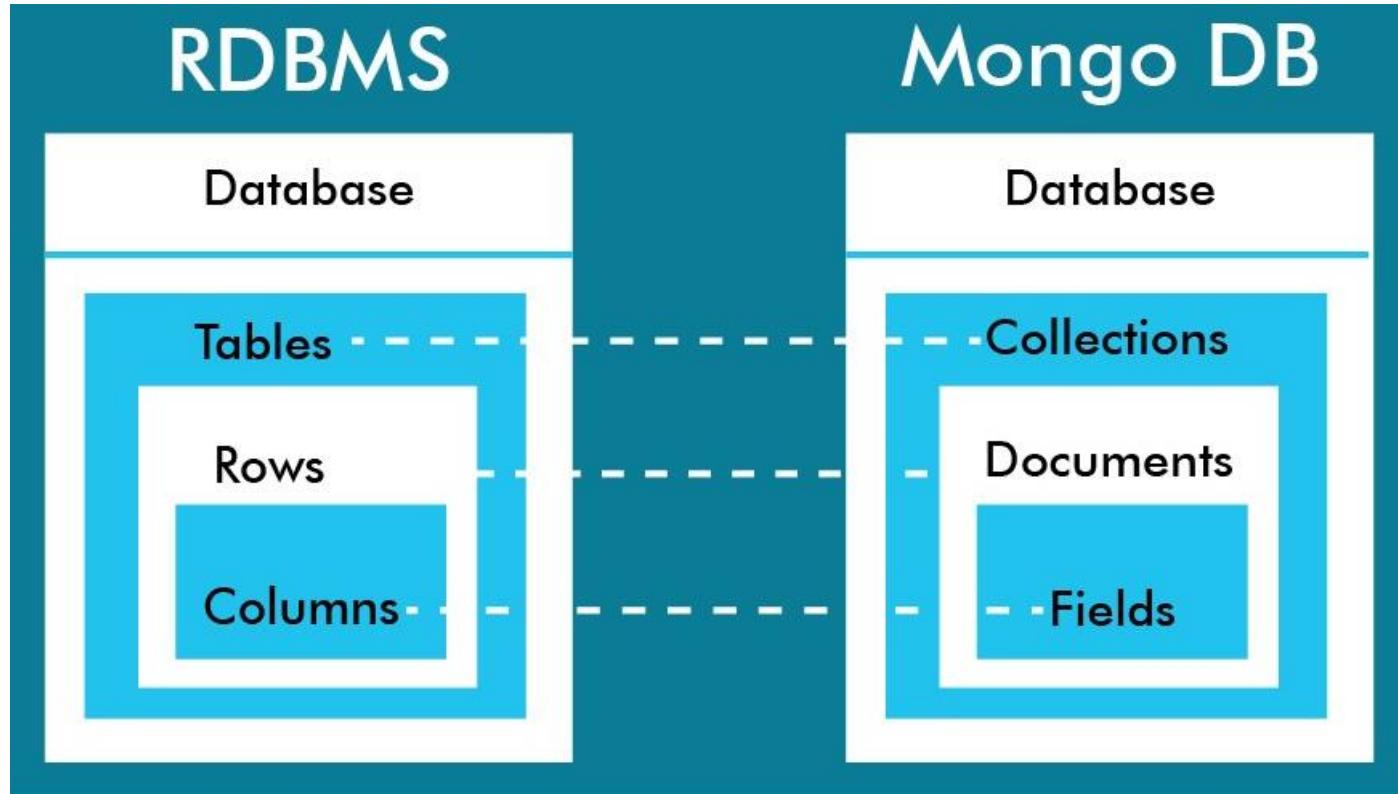
# NoSQL Database

---

- It is a non-relational database.
- No need to create tables, relations for storing data .
- This type of database is used to store web applications or large databases.



# RDBMS vs MongoDB



# Key Terms

---

## Collection

- It is a group of MongoDB documents.
- It is the equivalent of an RDBMS table.
- A collection exists within a single database.
- Collections do not enforce a schema.
- Documents within a collection can have different fields.

## Document

- A document is a set of key-value pairs.
- It has dynamic schema.
- Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

## MongoDB does not have schema: We can insert the data in any order.

- We can see in the screenshot that the first collection (tuple) in the document (table) newdb the first tuple has fields name and age while the second tuple had gender and name it is not mandatory to maintain a structure in MongoDB.

```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
> db.newdb.insert([{"name": "madhukar", "age": "21"}, {"gender": "male", "name": "ram"}])
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 2,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})
> db.newdb.find()
{ "_id" : ObjectId("58f0bb1b4bd378877d987c45"), "name" : "madhukar", "age" : "21" }
{ "_id" : ObjectId("58f0bb1b4bd378877d987c46"), "gender" : "male", "name" : "ram" }
> -
```

# Sample MongoDB Query

Query all employee names with salary greater than 18000 sorted in ascending order

```
db.users.find({salary:{$gt:18000}, {name:1}}).sort({salary:1})
```

Collection

Condition

Projection

Modifier

{salary:25000, ...}
{salary:10000, ...}
{salary:20000, ...}
{salary:2000, ...}
{salary:30000, ...}
{salary:21000, ...}
{salary:5000, ...}
{salary:50000, ...}

{salary:25000, ...}
{salary:20000, ...}
{salary:30000, ...}
{salary:21000, ...}
{salary:50000, ...}

{salary:20000, ...}
{salary:21000, ...}
{salary:25000, ...}
{salary:30000, ...}
{salary:50000, ...}



# Installation Process

---

- Cloud : MongoDB Atlas
  
- Stand-alone system – [link](#)
  - Install shell
  - Install compass
  - Install data tool (if needed)



# Execution Process

- Run MongoDB’s database server from command prompt
    - If we run the command “`mongod`”, it will activate the mongoDB database server which is running at port 27017.

```
C:\Program Files\MongoDB>cd Server  
  
C:\Program Files\MongoDB\Server>cd 3.4  
  
C:\Program Files\MongoDB\Server\3.4>cd bin  
  
C:\Program Files\MongoDB\Server\3.4\bin>mongod  
    2017-04-01T10:35:53.309+0530 I CONTROL [initandlisten] MongoDB starting : pid=10520 port=27017 dbpath=C:\data\db\ 64-bit host=Vignesh  
2017-04-01T10:35:53.311+0530 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2  
2017-04-01T10:35:53.311+0530 I CONTROL [initandlisten] db version v3.4.2  
2017-04-01T10:35:53.311+0530 I CONTROL [initandlisten] git version: 3f76e40c105fc223b3e5aac3e20dc026b83b38b  
2017-04-01T10:35:53.311+0530 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten] allocator: tcmalloc  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten] modules: none  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten] build environment:  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten]   distmod: 2008plus-ssl  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten]   distarch: x86_64  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten]   target arch: x86_64  
2017-04-01T10:35:53.312+0530 I CONTROL [initandlisten] options: {}  
2017-04-01T10:35:53.314+0530 I - [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.  
2017-04-01T10:35:53.315+0530 I STORAGE [initandlisten] wiredtiger.open config: create,cache_size=3530M/session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager_time=100000,checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),  
2017-04-01T10:35:54.229+0530 I CONTROL [initandlisten]  
2017-04-01T10:35:54.229+0530 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.  
2017-04-01T10:35:54.229+0530 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.  
2017-04-01T10:35:54.229+0530 I CONTROL [initandlisten]  
2017-04-01T10:35:55.714+0530 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/data/db/diagnostic.data'  
2017-04-01T10:35:55.715+0530 I NETWORK [thread1] waiting for connections on port 27017
```



- Run MongoDB's shell

- We run the “**mongo**” file, which is the MongoDB Shell, from a new command prompt window.
  - This is where we feed the database server with commands such as creating a database or a collection and dropping collections.

```
C:\Program Files\MongoDB>cd Server  
C:\Program Files\MongoDB\Server>cd 3.4  
C:\Program Files\MongoDB\Server\3.4>cd bin  
C:\Program Files\MongoDB\Server\3.4\bin>mongo  
MongoDB shell version v3.4.2  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 3.4.2  
Server has startup warnings:  
2017-04-01T10:35:54.229+0530 I CONTROL  [initandlisten]  
2017-04-01T10:35:54.229+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.  
2017-04-01T10:35:54.229+0530 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.  
2017-04-01T10:35:54.229+0530 I CONTROL  [initandlisten]
```

# Create Database

---

MongoDB uses the ‘use’ command to create a database.

If the database exists already, then it will be returned. Otherwise, a new database with the given database name will be created.

Syntax: ***use <DATABASE\_NAME>***

Example : **To create a database ‘movie’.**

- ***use movie***      //This will create a new database called ‘movie’.

```
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-30T20:25:05.952+0530 I CONTROL  [initandlisten]
2017-03-30T20:25:05.952+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-30T20:25:05.953+0530 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-03-30T20:25:05.954+0530 I CONTROL  [initandlisten]
> use movie
switched to db movie
>
```

# Drop Database

MongoDB uses the ‘`db.dropDatabase()`’ command to drop a database.

If the database exists already, then it will be dropped and true will be returned. Otherwise, nothing will be dropped and 0 will be returned.

Syntax: `db.dropDatabase()`

### *Example:*

- o `db.dropDatabase()`

```
PS C:\Users\Vignesh> mongo
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten]
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-31T18:45:20.800+0530 I CONTROL  [initandlisten] **                 Read and write access to data and configuration is unrestricted.
2017-03-31T18:45:20.801+0530 I CONTROL  [initandlisten]
> use movie
switched to db movie
> db.dropDatabase()
{ "ok" : 1 }
>
```



# Create Collection

A collection is what is referred to as a table in normal RDBMS.

It stores documents which may not be in the same structure.

A collection has various options such as setting maximum size and validating rules.

---

## Syntax:

***db.createCollection(name,options)***

- **Parameter :Name**

This field is used to specify the name of the collection which you are creating.

- **Parameter :Options**

This field is used to specify particular configurations for the collection which you have created.

Field	Type	Description
<b>capped</b>	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.
<b>autoIndexId</b>	Boolean	(Optional) If true, automatically create index on _id fields Default value is false.
<b>size</b>	number	(Optional) Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.
<b>max</b>	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

# Example

---

To create a database ‘movie’.

*use movie* //This will create a new database called ‘movie’.

To create collection ‘KMovies’

*db.createCollection("KMovies", { capped : true, autoIndexId : true, size :6142800, max : 10000 } )*

This will now create a collection KMovies which is capped, auto indexed with specified size and specified maximum number of documents.

This will now create a collection KMovies with capping, auto indexing, we will set the size to 6124800 and we will set the maximum number of documents to 10000.

```
Windows PowerShell - Command Prompt - mongo
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Vignesh>mongo
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten]
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-31T18:45:20.800+0530 I CONTROL  [initandlisten] **             Read and write access to data and configuration is unrestricted.
2017-03-31T18:45:20.801+0530 I CONTROL  [initandlisten]
> use movie
switched to db movie
> db.dropDatabase()
{ "ok" : 1 }
> use movie
switched to db movie
> db.createCollection("mycol", { capped : true, autoIndexId : true, size :6142800, max : 10000 } )
{
    "note" : "the autoIndexId option is deprecated and will be removed in a future release",
    "ok" : 1
}
> -
```



# Drop Collection

Since there are several collections in a particular database, we need to specify to MongoDB which collection we are aiming to drop.

We use “show collections” to find out which all collections we have in our database.

Syntax: ***db.COLLECTION\_NAME.drop()***

**Example : db.KMovies.drop()**

```
ch\ Command Prompt - mongo
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Vignesh>mongo
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten]
2017-03-31T18:45:20.797+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-31T18:45:20.800+0530 I CONTROL  [initandlisten] **             Read and write access to data and configuration is unrestricted.
2017-03-31T18:45:20.801+0530 I CONTROL  [initandlisten]

> show collections
> use movie
switched to db movie
> show collections
<Movies
mycol
> db.KMovies.drop()
true
>
```

# MongoDB Datatypes

This is highly or most often used data type.

---

- **Integer**
- **Boolean**
- **Double**
- **Arrays**
- **Date**
- **Null**
- **Binary etc...**
  
- Strings in MongoDB should be “ ”.

# Insert Document

---

This is used to insert several documents(tuples) into a collection(table)

If the collection is not available a new collection is created with the specified collection name and documents are inserted into it.

## Syntax

- db.collection\_name.insert(document)
  - // document parameter includes the column name and their values respectively, the rows are separated using {} and comma operators.

```
db.users.insertOne(           ← collection
{
    name: "sue",             ← field: value
    age: 26,                 ← field: value
    status: "pending"        ← field: value
}
)
```

# Code

---

```
db.list.insert ([{'movie':'kathi', 'actor':'vijay','director':'armurgadass'},  
{'movie':'vedhalam','actor':'ajith','director':'siva'},  
{'movie':'mass','actor':'surya','director':'venkatprabhu'},  
{'movie':'theri','actor':'vijay','director':'atlee'}])
```

# o/p

---

```
↳ C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.llist.insert([{'movie':'kathi','actor':'vijay','director':'ar murgadass'},{'movie':'vedhalam','actor':'ajith','director':'siva'}
y','director':'atlee']])
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 4,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})
> -
```

# Query Document

---

Find() Command

**Syntax**

```
db.collection_name.find()
```

//where collection\_name(table name) is the name of the collection.

# Code & O/P

db.list.find()

```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe

MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] **                 Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.list.find()
[ "_id" : ObjectId("58cf83da8604bb1938f94fcff"), "movie" : "Kathi", "actor" : "vijay", "director" : "ar murgadass" }
[ "_id" : ObjectId("58cf84b38604bb1938f94fd0"), "movie" : "vedhalam", "actor" : "ajith", "director" : "shiva" }
[ "_id" : ObjectId("58cf84b38604bb1938f94fd1"), "movie" : "mass", "actor" : "surya", "director" : "venkat prabhu" }
[ "_id" : ObjectId("58cf84b38604bb1938f94fd2"), "movie" : "theri", "actor" : "vijay", "director" : "atlee" }
>
```

---

## Pretty() Command

This is used to display the contents of collection in a formatted way.

### Syntax

- `db.collection_name.find().pretty()`
  - //where collection\_name(table name) is the name of the collection.

# Code and O/P

---

db.list.find.pretty()

```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.list.find().pretty()
{
    "_id" : ObjectId("58cf83da8604bb1938f94fcf"),
    "movie" : "Kathi",
    "actor" : "vijay",
    "director" : "ar murgadass"
}
{
    "_id" : ObjectId("58cf84b38604bb1938f94fd0"),
    "movie" : "vedhalam",
    "actor" : "ajith",
    "director" : "shiva"
}
{
    "_id" : ObjectId("58cf84b38604bb1938f94fd1"),
    "movie" : "mass",
    "actor" : "surya",
    "director" : "venkat prabhu"
}
{
    "_id" : ObjectId("58cf84b38604bb1938f94fd2"),
    "movie" : "theri",
    "actor" : "vijay",
    "director" : "atlee"
}
> -
```

# Update Document





# Code and O/P

```
db.list.update({‘movie’:’theri’},{$set:{‘movie’:’vijay61’}})
```

//using which the column movie with theri is replaced by vijay61.

C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe

```
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.list.update({ 'movie' : 'theri' }, { $set: { 'movie' : 'vijay61' } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.list.find()
{ "_id" : ObjectId("58cf83da8684bb1938f94fcf"), "movie" : "Kathi", "actor" : "vijay", "director" : "ar murgadass" }
{ "_id" : ObjectId("58cf84b38684bb1938f94fd0"), "movie" : "vedhalam", "actor" : "ajith", "director" : "shiva" }
{ "_id" : ObjectId("58cf84b38684bb1938f94fd1"), "movie" : "mass", "actor" : "surya", "director" : "venkat prabhu" }
{ "_id" : ObjectId("58cf84b38684bb1938f94fd2"), "movie" : "vijay61", "actor" : "vijay", "director" : "atlee" }
> -
```

# Delete Document

---

This is used to delete the value of a certain column in a collection

## Syntax

```
db.collection_name.remove(selection_criteria)
```

//where collection\_name(table name) is the name of the collection, selection\_criteria is which value should be deleted.

# Code and O/P

```
db.list.remove({'movie':'vijay61'})
```

↳ C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe

```
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] **             Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.list.remove({'movie':'vijay61'})
WriteResult({ "nRemoved" : 1 })
> db.list.find()
{ "_id" : ObjectId("58cf83da8604bb1938f94fcf"), "movie" : "Kathi", "actor" : "vijay", "director" : "ar murgadass" }
{ "_id" : ObjectId("58cf84b38604bb1938f94fd0"), "movie" : "vedhalam", "actor" : "ajith", "director" : "shiva" }
{ "_id" : ObjectId("58cf84b38604bb1938f94fd1"), "movie" : "mass", "actor" : "surya", "director" : "venkat prabhu" }
> -
```

# Projection

---

## **find()** Command

This is used to display only selected columns, if there are five columns in a collection and we want to display only three columns then this is possible using projection.

### Syntax

```
db.collection_name.find({}, {key:1})
```

- where `collection_name` is the name of the collection, `key` is the column name and `1` is specified , because that column should be displayed.

# Code and O/P

---

```
db.list.find({},{'movie':1,'actor':1})
```

- //using which only the movie columns and actor columns are displayed and director column is neglected from the collection.

```
◆ C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
MongoDB shell version v3.4.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.2
Server has startup warnings:
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten]
2017-03-25T00:00:59.731+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten] **                 Read and write access to data and configuration is unrestricted.
2017-03-25T00:00:59.732+0530 I CONTROL  [initandlisten]
> db.list.find({},{'movie':1,'actor':1})
{ "_id" : ObjectId("58cf83da8604bb1938f94fcf"), "movie" : "Kathi", "actor" : "vijay" }
{ "_id" : ObjectId("58cf84b38604bb1938f94fd0"), "movie" : "vedhalam", "actor" : "ajith" }
{ "_id" : ObjectId("58cf84b38604bb1938f94fd1"), "movie" : "mass", "actor" : "surya" }
> -
```



# Atomic operations

## FindAndModify() Command

This is used to modify certain fields of a document (tuple) in a collection (table).

## Syntax

```
db.collection_name.FindAndModify(query,update)
```

- where collection\_name is the name of the collection, query is the condition for which values should be modified , update is updated values.

# Code and O/P

---

```
db.list.findAndModify({ query:{movie:"kathi"},  
update:{$push:{Staring:[{actor:"illayathalapathy",actress:"beautifullady"}]}}})
```

```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe  
> db.list.findAndModify({query:{movie:"kathi"},update:{$push:{staring:[{actor:"ilayathalapathy",actress:"beautifullady"}]}}})  
{  
    "_id" : ObjectId("58f046024983a741aa10740f"),  
    "staring" : [  
        {  
            "actor" : "vijay",  
            "actress" : "samantha"  
        }  
    ],  
    "movie" : "kathi",  
    "director" : "ar murgadass",  
    "staring" : [  
        [  
            {  
                "actor" : "ilayathalapathy",  
                "actress" : "beautifullady"  
            }  
        ]  
    ]  
}
```

---

## **Limit() Method**

This method is used to limit the number of records that we want to display.

### **Syntax**

`limit(number)`

**`db.COLLECTION_NAME.find().limit(NUMBER)`**

### **Code**

`db.movie.find({}, {"director":1, _id:0}).limit(2) //this is used to display first two records;`

---

## Sort() Method

This method is used to sort the document in ascending/descending order. 1 is used for ascending order while -1 is used for descending order.

### Syntax

The basic syntax of sort() method is as follows –

**db.COLLECTION\_NAME.find().sort({KEY:1})**

### Code

```
db.movie.find({}, {"director":1,_id:0}).sort({"ratting":-1}) //this is used to sort the records in  
descending order ;
```

To query the document on the basis of some condition, you can use following operations.<sup>[3]</sup>

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>: {\$eq:<value>}}	db.mycol.find({"by":"tut p"}).pretty()	where by = 'tut p'
Less Than	{<key>: {\$lt:<value>}}	db.mycol.find({"likes": {\$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{\$lte: <value>}}	db.mycol.find({"likes": {\$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>: {\$gt:<value>}}	db.mycol.find({"likes": {\$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{\$gte: <value>}}	db.mycol.find({"likes": {\$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>: {\$ne:<value>}}	db.mycol.find({"likes": {\$ne:50}}).pretty()	where likes != 50

# MongoDB- Replication

---

Replication in mongoDB refers to creating and storing multiple copies of same data across different servers.

This helps in data safety due to system failure or data loss at single server.

It also ensures redundancy and data availability at different locations

# How replication works in MongoDB



# MongoDB Aggregation

---

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

It is similar to count(\*) and group by in sql.

## Syntax

Basic syntax of aggregate() method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION_TO_BE_DONE)
```

Expression	Description	Example
\$sum	Sums up the defined value from all the documents in the collection.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$sum : "\$likes"}}}])
\$avg	Calculates the average of all the given values in the documents of a collection.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$avg : "\$likes"}}}])
\$min	Gets the minimum of the corresponding values from all documents in the collection.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$min : "\$likes"}}}])

Expression	Description	Example
\$max	Gets the maximum of the corresponding values from all documents in the collection.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$max : "\$likes"}}}])
\$push	Inserts the value to an array in the resulting document.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$push : "\$likes"}}}])
\$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$addToSet : "\$likes"}}}])
\$First	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "\$sort"-stage.	db.movie.aggregate([{\$group : {_id : "\$by_actor", movie_name : {\$First : "\$likes"}}}])

# MongoDB – Help Command

---

To see the list of help for methods you can use on the db object, call the db.help() method:

- `db.help()`

## Example - Collection Help

- `db.collection.help()`

# Import csv / excel data

---

```
mongoimport --db <file name> --collection <collectionName> --type csv --<fileLocation.csv> --  
headerline
```

mongo

Show dbs

Use <databaseNAme>

Show collection

Db.<collectionname>.findone()

---

```
mongoimport --db irctc -c irctcR --type csv --file  
C:\Users\manis\OneDrive\Desktop\MongoDB\irctc.csv --headerline
```

Command Prompt

Microsoft Windows [Version 10.0.22000.1219]  
(c) Microsoft Corporation. All rights reserved.

C:\Users\manis>cd C:\Program Files\MongoDB\Server\5.0\bin

C:\Program Files\MongoDB\Server\5.0\bin>mongoimport --db irctc -c irctcC --type csv --file irctc.csv --headerline  
'mongoimport' is not recognized as an internal or external command,  
operable program or batch file.

C:\Program Files\MongoDB\Server\5.0\bin>mongoimport --db irctc -c irctcC --type csv --file irctc.csv --headerline  
2022-12-06T18:45:08.316+0530 connected to: mongodb://localhost/  
2022-12-06T18:45:08.377+0530 3 document(s) imported successfully. 0 document(s) failed to import.

C:\Program Files\MongoDB\Server\5.0\bin>

```
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> mongoimport --db irctc --collection irctcR --type csv --file C:\Users\manis\OneDrive\Desktop\MongoDB\irctc.csv --headerline
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:14
> mongoimport --db irctc --collection irctcR --type csv --file C:\Users\manis\OneDrive\Desktop\MongoDB\irctc.csv --headerline
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:14
> mongoimport --db irctc -c irctcC --type csv --file irctc.csv --headerline
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:14
> show dbs
admin 0.000GB
config 0.000GB
irctc 0.000GB
local 0.000GB
movie 0.000GB
>
```

```
@(shell):1:1
> show collections
> show dbs
admin 0.000GB
config 0.000GB
irctc 0.000GB
local 0.000GB
movie 0.000GB
> db irctc
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:3
> db
test
> use irctc
switched to db irctc
> show collections
irctcC
> db.irctcC.find()
{ "_id" : ObjectId("638f405cbbf21e961eb5347e"), "ticketNo" : 1, "Trainname" : "xyz", "TrainNo" : 121 }
{ "_id" : ObjectId("638f405cbbf21e961eb5347f"), "ticketNo" : 2, "Trainname" : "rt", "TrainNo" : 122 }
{ "_id" : ObjectId("638f405cbbf21e961eb53480"), "ticketNo" : 3, "Trainname" : "tyz", "TrainNo" : 123 }
>
```

# Use full link for assignment

---

<https://www.mongodb.com/docs/manual/crud/>

<https://www.mongodb.com/docs/manual/aggregation/>

# References

---

- [1] <https://mindmajix.com/what-is-mongodb>
- [2] <https://www.mongodb.com/docs/manual/crud/>



# BITS Pilani presentation

**BITS** Pilani  
Pilani Campus

Manu Saxena

# Content

---

Spark: Introduction, Architecture and Features

Programming on Spark: Resilient Distributed Datasets, Transformation, Examples

# What is Hadoop?

---

Apache Hadoop is a platform that handles large datasets in a distributed fashion. The framework uses MapReduce to split the data into blocks and assign the chunks to nodes across a cluster. MapReduce then processes the data in parallel on each node to produce a unique output.

The Apache Hadoop Project consists of four main modules:

HDFS – Hadoop Distributed File System. This is the file system that manages the storage of large sets of data across a Hadoop cluster. HDFS can handle both structured and unstructured data. The storage hardware can range from any consumer-grade HDDs to enterprise drives.

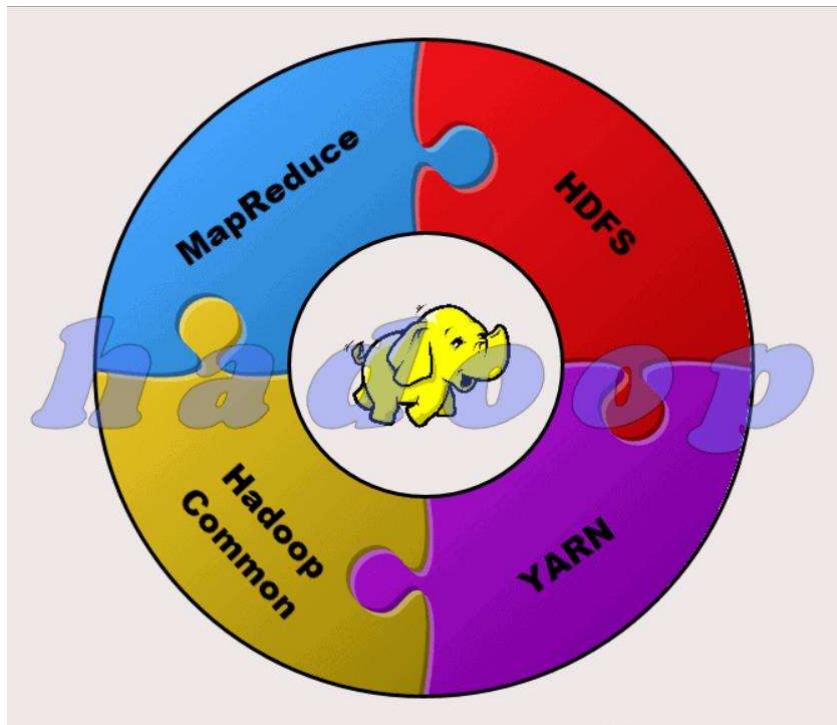
MapReduce. The processing component of the Hadoop ecosystem. It assigns the data fragments from the HDFS to separate map tasks in the cluster. MapReduce processes the chunks in parallel to combine the pieces into the desired result.

YARN. Yet Another Resource Negotiator. Responsible for managing computing resources and job scheduling.

Hadoop Common. The set of common libraries and utilities that other modules depend on. Another name for this module is Hadoop core, as it provides support for all other Hadoop components.

---

# What is Hadoop?



# What is Spark?

---

Apache Spark is an open-source tool. This framework can run in a standalone mode or on a cloud or cluster manager such as Apache Mesos, and other platforms. It is designed for fast performance and uses RAM for caching and processing data.

Spark performs different types of big data workloads. This includes MapReduce-like batch processing, as well as real-time stream processing, machine learning, graph computation, and interactive queries.

The Spark engine was created to improve the efficiency of MapReduce and keep its benefits. Even though Spark does not have its file system, it can access data on many different storage solutions. The data structure that Spark uses is called Resilient Distributed Dataset, or RDD.

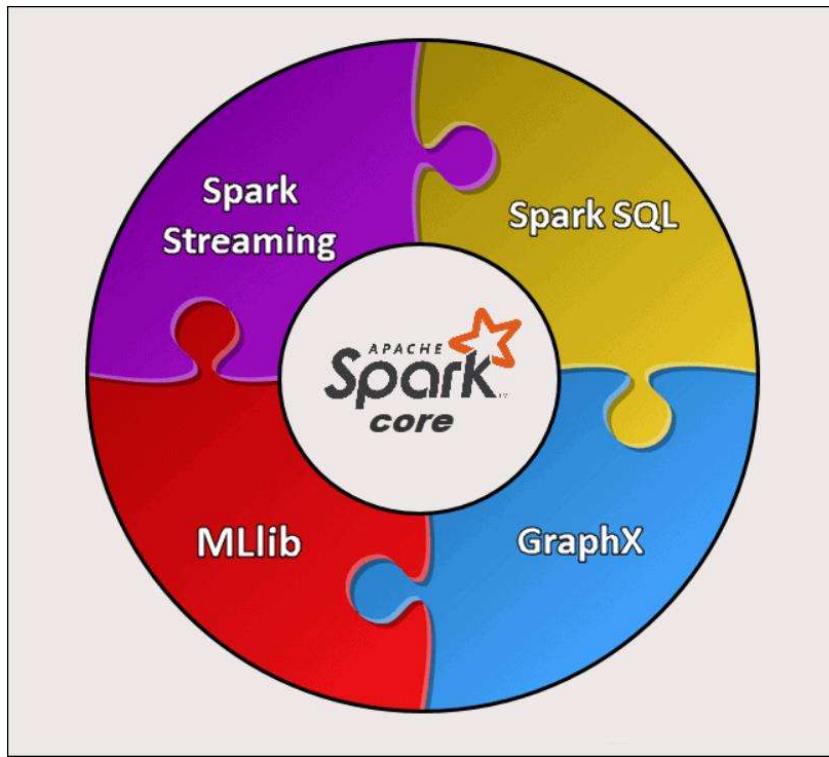
# What is Spark?

---

There are five main components of Apache Spark:

- 1. Apache Spark Core.** The basis of the whole project. Spark Core is responsible for necessary functions such as scheduling, task dispatching, input and output operations, fault recovery, etc. Other functionalities are built on top of it.
- 2. Spark Streaming.** This component enables the processing of live data streams. Data can originate from many different sources, including Kafka, Kinesis, Flume, etc.
- 3. Spark SQL.** Spark uses this component to gather information about the structured data and how the data is processed.
- 4. Machine Learning Library (MLlib).** This library consists of many machine learning algorithms. MLlib's goal is scalability and making machine learning more accessible.
- 5. GraphX.** A set of APIs used for facilitating graph analytics tasks.

# What is Spark?



# Key Differences Between Hadoop and Spark

<b>Category for Comparison</b>	<b>Hadoop</b>	<b>Spark</b>
<b>Performance</b>	Slower performance, uses disks for storage and depends on disk read and write speed.	Fast in-memory performance with reduced disk reading and writing operations.
<b>Cost</b>	An open-source platform, less expensive to run. Uses affordable consumer hardware. Easier to find trained Hadoop professionals.	An open-source platform, but relies on memory for computation, which considerably increases running costs.
<b>Data Processing</b>	Best for batch processing. Uses MapReduce to split a large dataset across a cluster for parallel analysis.	Suitable for iterative and live-stream data analysis. Works with RDDs and DAGs to run operations.

# Key Differences Between Hadoop and Spark

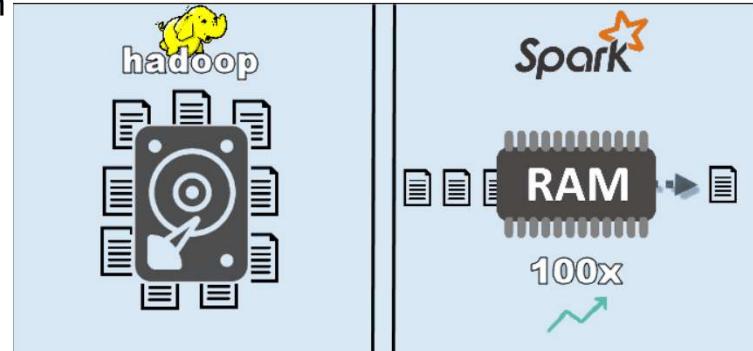
Category for Comparison	Hadoop	Spark
Fault Tolerance	A highly fault-tolerant system. Replicates the data across the nodes and uses them in case of an issue.	Tracks RDD block creation process, and then it can rebuild a dataset when a partition fails. Spark can also use a DAG to rebuild data across nodes.
Scalability	Easily scalable by adding nodes and disks for storage. Supports tens of thousands of nodes without a known limit.	A bit more challenging to scale because it relies on RAM for computations. Supports thousands of nodes in a cluster.
Security	Extremely secure. Supports LDAP, ACLs, Kerberos, SLAs, etc.	Not secure. By default, the security is turned off. Relies on integration with Hadoop to achieve the necessary security level.

# Key Differences Between Hadoop and Spark

Category for Comparison	Hadoop	Spark
<b>Ease of Use and Language Support</b>	More difficult to use with less supported languages. Uses Java or Python for MapReduce apps.	More user friendly. Allows interactive shell mode. APIs can be written in Java, Scala, R, Python, Spark SQL.
<b>Machine Learning</b>	Slower than Spark. Data fragments can be too large and create bottlenecks. Mahout is the main library.	Much faster with in-memory processing. Uses MLlib for computations.
<b>Scheduling and Resource Management</b>	Uses external solutions. YARN is the most common option for resource management. Oozie is available for workflow scheduling.	Has built-in tools for resource allocation, scheduling, and monitoring.

# Performance

By accessing the data stored locally on HDFS, Hadoop boosts the overall performance. However, it is not a match for Spark's in-memory processing. According to Apache's claims, Spark appears to be 100x faster when using RAM for computing than Hadoop with MapReduce.



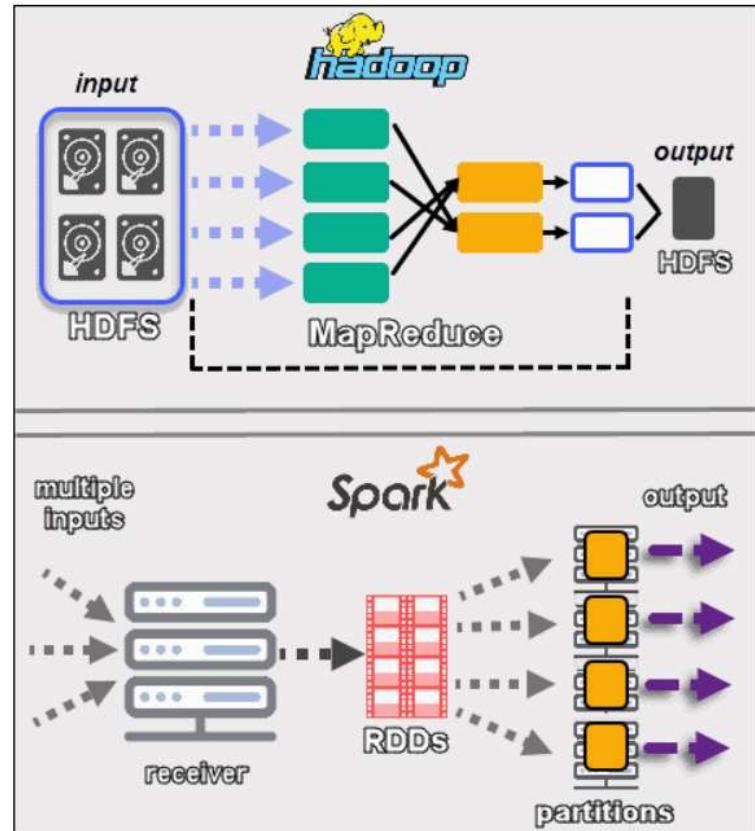
The dominance remained with sorting the data on disks. Spark was 3x faster and needed 10x fewer nodes to process 100TB of data on HDFS. This benchmark was enough to set the world record in 2014.

The main reason for this supremacy of Spark is that it does not read and write intermediate data to disks but uses RAM. Hadoop stores data on many different sources and then process the data in batches using MapReduce.

# Data Processing

The two frameworks handle data in quite different ways. Although both Hadoop with MapReduce and Spark with RDDs process data in a distributed environment, Hadoop is more suitable for batch processing. In contrast, Spark shines with real-time processing.

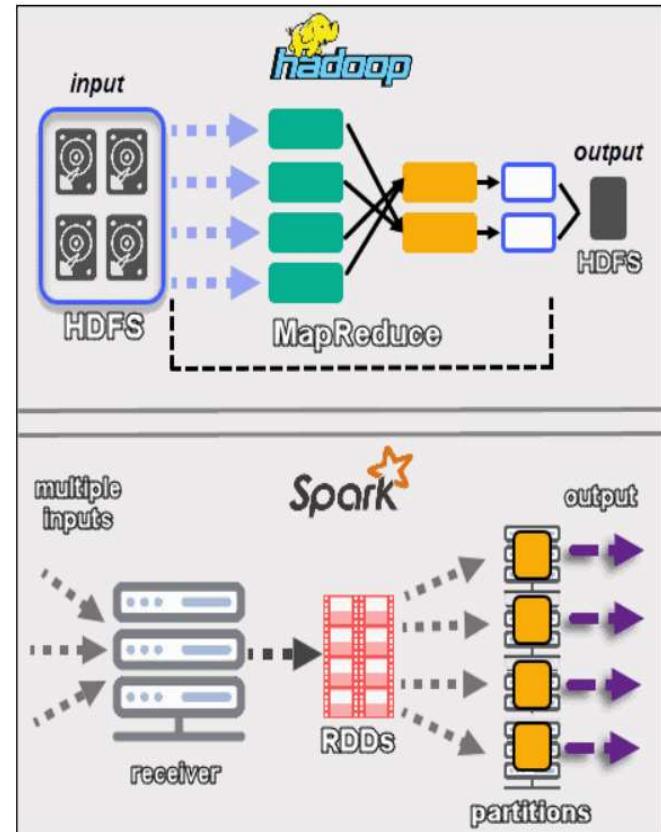
Hadoop's goal is to store data on disks and then analyze it in parallel in batches across a distributed environment. MapReduce does not require a large amount of RAM to handle vast volumes of data. Hadoop relies on everyday hardware for storage, and it is best suited for linear data processing.



# Data Processing

Apache Spark works with resilient distributed datasets (RDDs). An RDD is a distributed set of elements stored in partitions on nodes across the cluster. The size of an RDD is usually too large for one node to handle. Therefore, Spark partitions the RDDs to the closest nodes and performs the operations in parallel. The system tracks all actions performed on an RDD by the use of a Directed Acyclic Graph (DAG).

With the in-memory computations and high-level APIs, Spark effectively handles live streams of unstructured data. Furthermore, the data is stored in a predefined number of partitions. One node can have as many partitions as needed, but one partition cannot expand to another node.



# Fault Tolerance

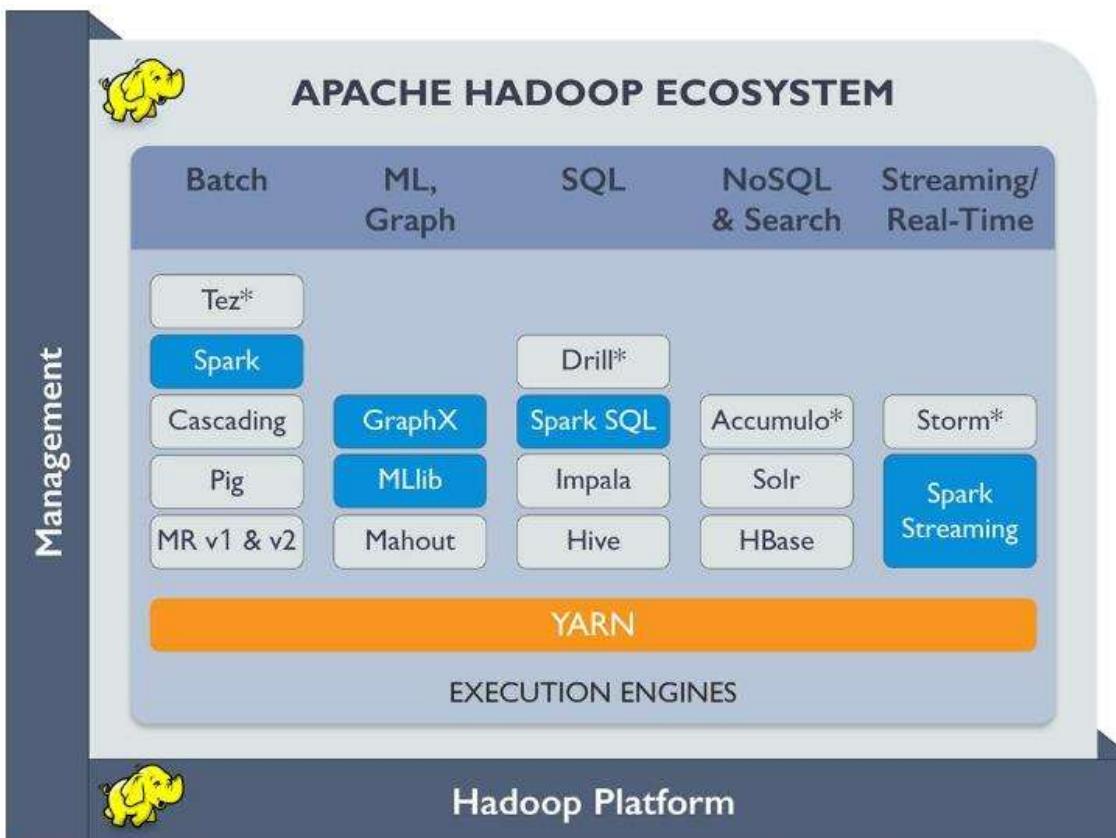
---

Both provide a respectable level of handling failures but the way they approach fault tolerance is different.

Hadoop has fault tolerance as the basis of its operation. It replicates data many times across the nodes. In case an issue occurs, the system resumes the work by creating the missing blocks from other locations. The master nodes track the status of all slave nodes. Finally, if a slave node does not respond to pings from a master, the master assigns the pending jobs to another slave node.

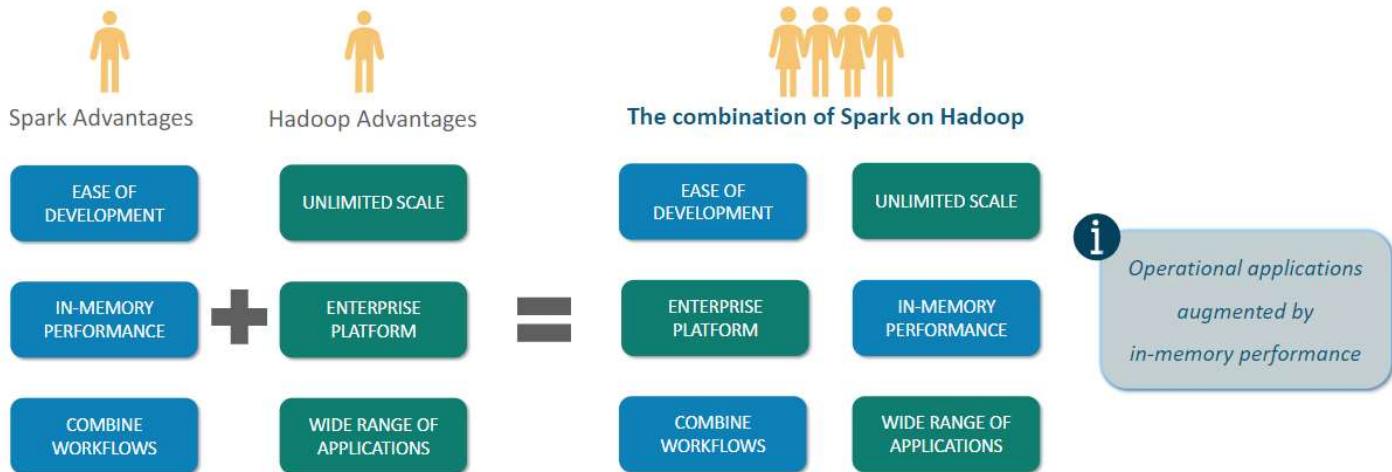
Spark uses RDD blocks to achieve fault tolerance. The system tracks how the immutable dataset is created. Then, it can restart the process when there is a problem. Spark can rebuild data in a cluster by using DAG tracking of the workflows. This data structure enables Spark to handle failures in a distributed data processing ecosystem.

# Spark in Hadoop ecosystem

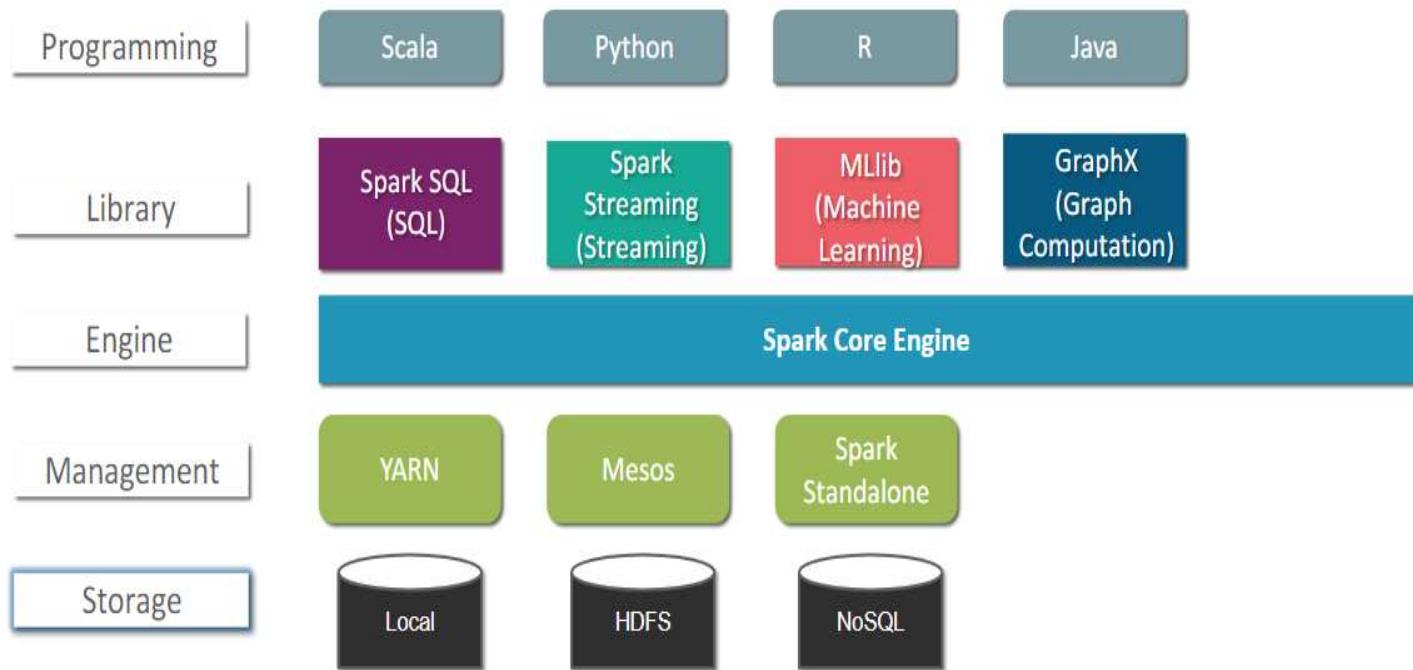


# Spark and Hadoop

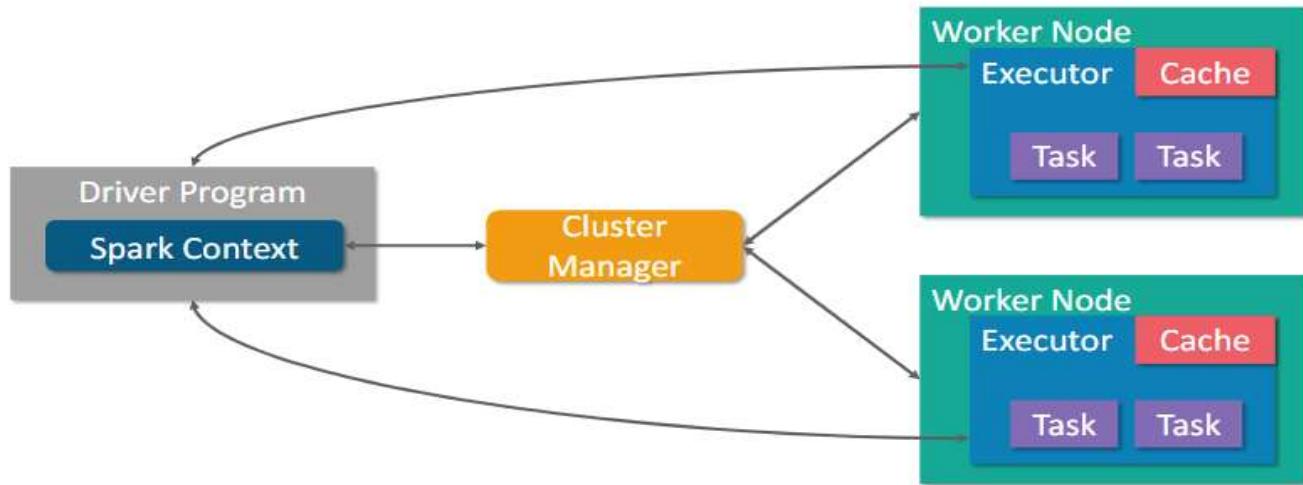
## Spark And Hadoop Together



# Spark Components



# Spark Architecture



1. ***SparkContext*** object in *driver program* coordinates all the distributed process and allows resource allocation. Driver program acts like a *Name Node*
2. Cluster Managers provide Executors with JVM process with logic
3. *SparkContext* object sends the application to executors
4. *SparkContext* executes tasks in each executor

# Spark Deployment Modes

Spark can be deployed in any of the following ways:

## Amazon EC2

- Scripts that let you launch a cluster on EC2 in about 5 minutes

## Standalone Deploy Mode

- Launch a standalone cluster quickly without a third-party cluster manager

## Mesos

- Deploy a private cluster using Apache Mesos

## YARN

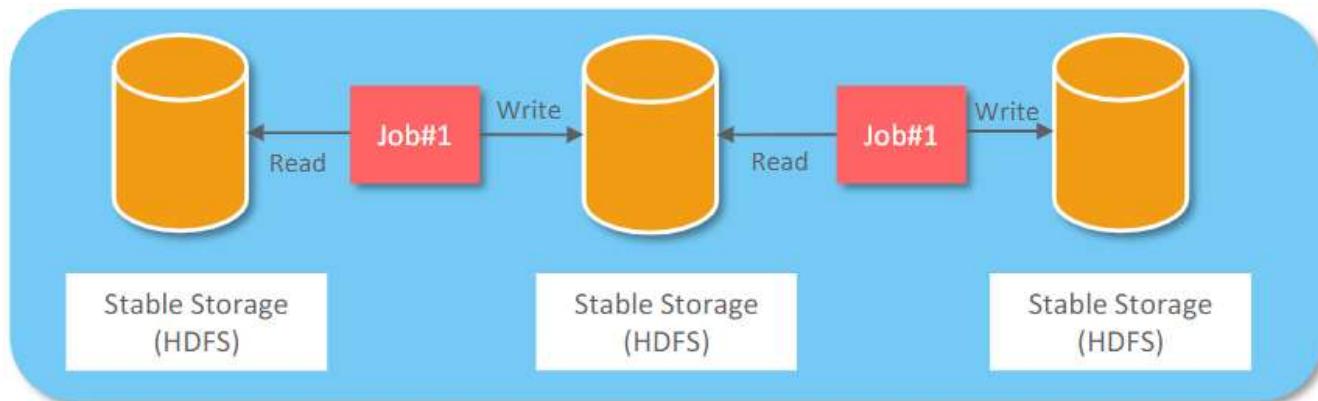
- Deploy Spark on top of Hadoop YARN

# Challenge in Existing Computing Methods

When it comes to *iterative distributed computing*, i.e. processing data over multiple jobs in computations, we need to reuse or share data among multiple jobs

There are problems with the current computing method (*MapReduce*)

- We need to store data in some intermediate stable distributed storage such as HDFS
- Multiple I/O operations makes the overall computation of ***jobs slower***
- Replications and serializations which in turn makes the ***process slower***



# Probable Solution

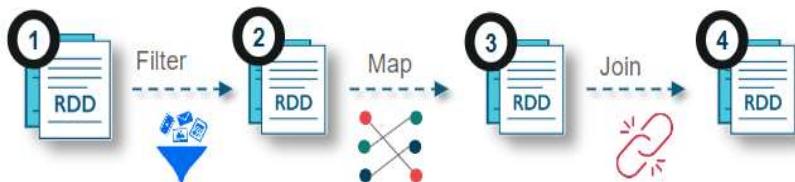
- Our Goal here is to reduce the number of I/O operations through HDFS
- This can only be achieved through "*In-Memory Data Sharing*"
- The *In-Memory Data Sharing* is 10-100x faster than *Network/Disk* sharing



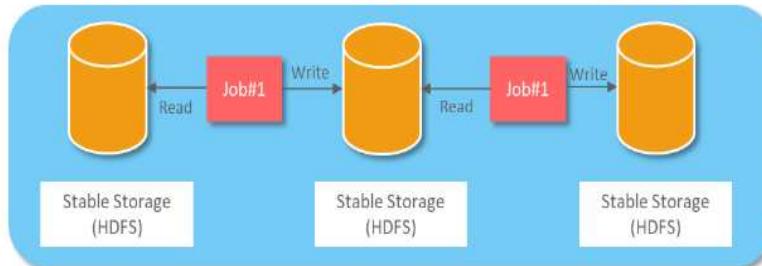
# RDD – Perfect Solution

- RDDs try to solve all the problems by enabling *fault tolerant, distributed In-memory computations*
- Since RDDs are created over a set of transformations, it logs those transformations, rather than actual data

*For example:*



- In case if we lose some partition of RDD, we can replay the transformation on that partition in *lineage* to achieve the same computation, rather than doing data replication across multiple nodes
- This characteristic is biggest benefit of RDD, because it *saves* a lot of *efforts in data management and replication* and thus achieves *faster computations*



# What is RDD

- *RDD = Resilient Distributed Datasets*
- RDD is a *distributed memory abstraction* which lets programmers perform *in-memory computations* on large clusters in a *fault-tolerant manner*
- They are *read-only collection* of objects *partitioned across* a set of machines that *can be rebuilt* if a partition is lost
- RDDs can be *created from multiple data sources* e.g. Scala collection, local file system, Hadoop, Amazon S3, HBase table etc
- There are several operations performed on RDDs:
  - *Functions*
  - *Transformations*
  - *Actions*



# Features of RDD

## In-Memory Computation

RDDs have a provision of in-memory computation

## Lazy Evaluations

All transformations are lazy, it does not compute their results right away

## Fault Tolerance

RDDs track data lineage information to rebuild lost data automatically

## Coarse Grained Operations

Applies to all elements in datasets through maps or filter or group by operation

## Persistence

Users can reuse RDDs and choose a storage strategy for them

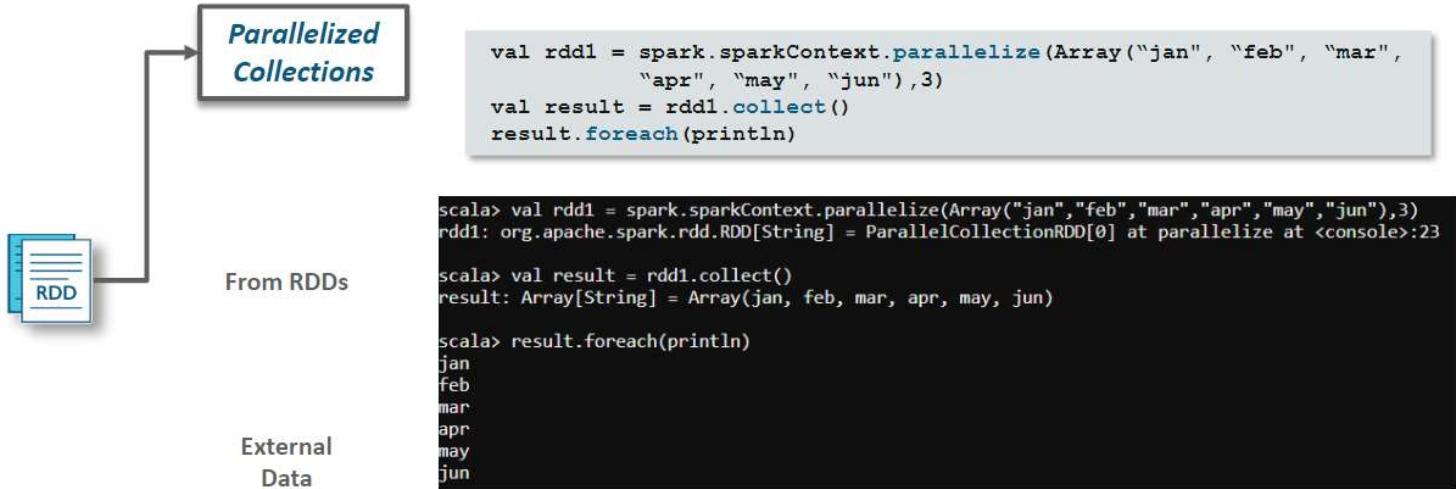
## Partitioning

Partitioning is the fundamental unit of parallelism in Spark RDD

## Immutability

Data can be created or retrieved anytime

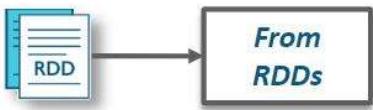
# How to create RDD



# How to create RDD

Parallelized  
Collections

```
val words = spark.sparkContext.parallelize(Seq("the", "quick",
                                              "brown", "fox", "jumps", "over", "the", "lazy", "dog"))
val wordPair = words.map(w => (w.charAt(0), w))
Val wordPair1 = wordPair.collect()
wordPair1.foreach(println)
```



External  
Data

```
scala> val words=spark.sparkContext.parallelize(Seq("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"))
words: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[10] at parallelize at <console>:23

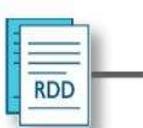
scala> val wordPair = words.map(w => (w.charAt(0), w))
wordPair: org.apache.spark.rdd.RDD[(Char, String)] = MapPartitionsRDD[11] at map at <console>:25

scala> val wordPair1 = wordPair.collect()
wordPair1: Array[(Char, String)] = Array((t,the), (q,quick), (b,brown), (f,fox), (j,jumps), (o,over), (t,the), (l,lazy), (d,dog))

scala> wordPair1.foreach(println)
(t,the)
(q,quick)
(b,brown)
(f,fox)
(j,jumps)
(o,over)
(t,the)
(l,lazy)
(d,dog)
```

# How to create RDD

Parallelized  
Collections



From RDDs

```
val dataRDD =  
spark.read.textFile("file:///mnt/home/edureka_292003/input.txt").rdd
```

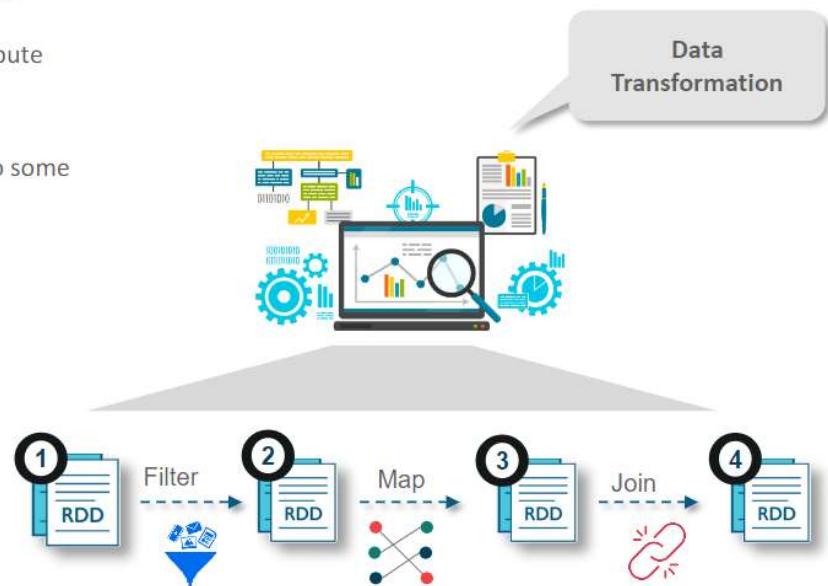
*External  
Data*

```
scala> val dataRDD = spark.read.textFile("file:///mnt/home/edureka_292003/input.txt").rdd  
dataRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at rdd at <console>:23
```

# RDD Operations: Transformations

## Transformations

- Transformations *create a new RDD from an existing one*
- All *transformations* in Spark *are lazy*: they do not compute their results right away
- Instead they remember the transformations applied to some base dataset
- This helps in:
  - *Optimizing the required calculations*
  - *Recovery of lost data partitions*



# RDD Operations: Transformations

Transformation	Meaning
map()	Returns a new distributed dataset formed by passing each element of the source through the function
filter(func)	Returns a new dataset formed by selecting those elements of the source on which func returns true
intersection(otherDataset)	Returns a new RDD that contains the intersection of elements in the source dataset and the argument
groupByKey([numTasks])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs
reduceByKey(func, [numTasks])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V
join(otherDataset, [numTasks])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
cartesian(otherDataset)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

# RDD Operations: Transformations

Transformation	Meaning
distinct()	Returns a new RDD that contains each unique value only once
flatMap()	Similar to map, but allows emitting more than one item in the map function
union(Dataset)	Returns a new RDD that contains all the elements of the source dataset and the argument
coalesce(numPartitions)	Decreases the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset
cogroup(otherDataset, [numTasks])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, Iterable<V>, Iterable<W>) tuples
subtract()	Removes the content of one RDD

# RDD Operations: Transformations

map

filter

groupByKey

join

Example

```
val a = sc . parallelize ( List (" dog " , " salmon " , " salmon " , " rat " ,  
" elephant") , 3)  
val b = a . map ( _ . length )  
val c = a . zip ( b )  
c . collect
```

```
scala> val a = sc . parallelize ( List (" dog " , " salmon " , " salmon " , " rat " , " elephant") , 3)  
a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:24  
  
scala> val b = a . map ( _ . length )  
b: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at map at <console>:26  
  
scala> val c = a . zip ( b )  
c: org.apache.spark.rdd.RDD[(String, Int)] = ZippedPartitionsRDD[2] at zip at <console>:28  
  
scala> c . collect  
res0: Array[(String, Int)] = Array((" dog ",5), (" salmon ",8), (" salmon ",8), (" rat ",5), (" elephant",9))
```

# RDD Operations: Transformations

map

filter

groupByKey

join

Example

```
val a = sc . parallelize (1 to 10 , 3)
a . filter ( _ % 2 == 0)
b . collect
```

```
scala> val a = sc . parallelize (1 to 10 , 3)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[3] at parallelize at <console>:24

scala> a . filter ( _ % 2 == 0)
res1: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at filter at <console>:27

scala> b . collect
res2: Array[Int] = Array(5, 8, 8, 5, 9)
```

# RDD Operations: Transformations

map

filter

groupByKey

join

Example

```
val data =  
spark.sparkContext.parallelize(Array(('k',5), ('s',3), ('s',4), ('p',7), ('p',5), ('t',8),  
('k',6)), 3)  
val group = data.groupByKey().collect()  
group.foreach(println)
```

```
scala> val data = spark.sparkContext.parallelize(Array(('k',5), ('s',3), ('s',4), ('p',7), ('p',5), ('t',8), ('k',6)), 3)  
data: org.apache.spark.rdd.RDD[(Char, Int)] = ParallelCollectionRDD[5] at parallelize at <console>:23  
  
scala> val group = data.groupByKey().collect()  
group: Array[(Char, Iterable[Int])] = Array((s,CompactBuffer(3, 4)), (p,CompactBuffer(7, 5)), (t,CompactBuffer(8)), (k,CompactBuffer(5, 6)))  
  
scala> group.foreach(println)  
(s,CompactBuffer(3, 4))  
(p,CompactBuffer(7, 5))  
(t,CompactBuffer(8))  
(k,CompactBuffer(5, 6))
```

# RDD Operations: Transformations

map

filter

groupByKey

join

Example

```
val a = sc . parallelize ( List (" dog " , " salmon " , " salmon " , " rat " ,  
" elephant") , 3)  
val b = a . keyBy ( _ . length )  
val c = sc . parallelize ( List (" dog " , " cat " , " gnu " , " salmon " , "  
rabbit " , " turkey" , " wolf " , " bear " , " bee " ) , 3)  
val d = c . keyBy ( _ . length )  
b . join ( d ) . collect
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val rdd1 =  
spark.sparkContext.parallelize(Seq((1,"jan",2016), (3,"nov",2014)  
, (16,"feb",2014)))  
val rdd2 =  
spark.sparkContext.parallelize(Seq((5,"dec",2014), (1,"jan",2016)  
val comman = rdd1.intersection(rdd2)  
comman.collect.foreach(println)
```

```
scala> val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014), (16,"feb",2014)))  
rdd1: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[34] at parallelize at <console>:23  
  
scala> val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(1,"jan",2016)))  
rdd2: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[35] at parallelize at <console>:23  
  
scala> val comman = rdd1.intersection(rdd2)  
comman: org.apache.spark.rdd.RDD[(Int, String, Int)] = MapPartitionsRDD[41] at intersection at <console>:27  
  
scala> comman.collect.foreach(println)  
(1,jan,2016)
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val words =  
  Array("one", "two", "two", "four", "five", "six", "six", "eight", "nine"  
  , "ten")  
  val data = spark.sparkContext.parallelize(words).map(w =>  
    (w, 1)).reduceByKey(_+_)  
  data.collect.foreach(println)
```

```
scala> val words = Array("one", "two", "two", "four", "five", "six", "six", "eight", "nine", "ten")  
words: Array[String] = Array(one, two, two, four, five, six, six, eight, nine, ten)  
  
scala> val data = spark.sparkContext.parallelize(words).map(w => (w,1)).reduceByKey(_+_)  
data: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[19] at reduceByKey at <console>:25  
  
scala> data.collect.foreach(println)  
(two,2)  
(one,1)  
(nine,1)  
(six,2)  
(five,1)  
(four,1)  
(eight,1)  
(ten,1)
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val rdd1 =  
spark.sparkContext.parallelize(Seq((1,"jan",2016), (3,"nov",2014)  
, (16,"feb",2014), (3,"nov",2014)))  
val result = rdd1.distinct()  
println(result.collect().mkString(", "))
```

```
scala> val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014),(16,"feb",2014),(3,"nov",2014)))  
rdd1: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[45] at parallelize at <console>:23  
  
scala> val result = rdd1.distinct()  
result: org.apache.spark.rdd.RDD[(Int, String, Int)] = MapPartitionsRDD[48] at distinct at <console>:25  
  
scala> println(result.collect().mkString(", "))  
(3,nov,2014), (16,feb,2014), (1,jan,2016)
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val a = sc . parallelize (1 to 3 , 1)
val b = sc . parallelize (5 to 7 , 1)
( a ++ b ) . collect
```

```
scala> val a = sc . parallelize (1 to 3 , 1)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[49] at parallelize at <console>:24

scala> val b = sc . parallelize (5 to 7 , 1)
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[50] at parallelize at <console>:24

scala> ( a ++ b ) . collect
res13: Array[Int] = Array(1, 2, 3, 5, 6, 7)
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val a = sc . parallelize (1 to 10 , 5)
a . flatMap (1 to _ ) . collect
```

```
scala> val a = sc . parallelize (1 to 10 , 5)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[32] at parallelize at <console>:24
scala> a . flatMap (1 to _ ) . collect
res23: Array[Int] = Array(1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val rdd1 =  
spark.sparkContext.parallelize(Array("jan","feb","mar","april",  
"may","jun"),3)  
val result = rdd1.coalesce(2)  
result.collect.foreach(println)
```

```
scala> val rdd1 = spark.sparkContext.parallelize(Array("jan","feb","mar","april","may","jun"),3)  
rdd1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[54] at parallelize at <console>:23
```

```
scala> val result = rdd1.coalesce(2)  
result: org.apache.spark.rdd.RDD[String] = CoalescedRDD[55] at coalesce at <console>:25
```

```
scala> result.collect.foreach(println)  
jan  
feb  
mar  
april  
may  
jun
```

# RDD Operations: Transformations

intersection

reduceByKey

distinct

union

flatmap

coalesce

subtract

```
val a = sc . parallelize (1 to 9 , 3)
val b = sc . parallelize (1 to 3 , 3)
val c = a . subtract ( b )
c . collect
```

```
scala> val a = sc . parallelize (1 to 9 , 3)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[56] at parallelize at <console>:24

scala> val b = sc . parallelize (1 to 3 , 3)
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[57] at parallelize at <console>:24

scala> val c = a . subtract ( b )
c: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[61] at subtract at <console>:28

scala> c . collect
res17: Array[Int] = Array(6, 9, 4, 7, 5, 8)
```

# RDD Operations: Actions

*Spark forces the calculations for execution only when actions are invoked on the RDDs*

Action	Meaning
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one)
first()	Returns the first element of the dataset (similar to take(1))
takeOrdered(n, [ordering])	Returns the first n elements of the RDD using either their natural order or a custom comparator
count()	Returns the number of elements in the dataset
collect()	Returns all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data
saveAsSequenceFile(path)	Writes the elements of the dataset as a Hadoop SequenceFile in a given path in the local file system, HDFS or any other Hadoop-supported file system
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an accumulator variable
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key

# RDD Operations: Actions

first

reduce

```
val c = sc . parallelize ( List (" Gnu " , " Cat " , " Rat " ,  
" Dog ") , 2)  
c . first
```

takeOrdered

countByKey

```
scala> val c = sc . parallelize ( List (" Gnu " , " Cat " , " Rat " , " Dog ") , 2)  
c: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[62] at parallelize at <console>:24  
  
scala> c . first  
res18: String = " Gnu "
```

aggregate

# RDD Operations: Actions

first

reduce

```
val a = sc.parallelize(1 to 100, 3)
a.reduce(_+_)
```

takeOrdered

countByKey

```
scala> val a = sc.parallelize(1 to 100,3)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[63] at parallelize at <console>:24
```

```
scala> a.reduce(_+_)
res19: Int = 5050
```

aggregate

# RDD Operations: Actions

first

reduce

takeOrdered

countByKey

aggregate

```
val b = sc . parallelize ( List (" dog " , " cat " , " ape " ,  
" salmon " , " gnu ") , 2)  
b . takeOrdered (2)
```

```
scala> val b = sc . parallelize ( List (" dog " , " cat " , " ape " , " salmon " , " gnu ") , 2)  
b: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[64] at parallelize at <console>:24
```

```
scala> b . takeOrdered (2)  
res20: Array[String] = Array(" ape ", " cat ")
```

# RDD Operations: Actions

first

reduce

takeOrdered

countByKey

aggregate

```
val c = sc . parallelize ( List (3," Gnu " ) , (3," Yak " ) ,  
(5," Mouse " ) , (3," Dog " ) ) , 2)  
c.countByKey
```

```
scala> val c = sc . parallelize ( List ((3 , " Gnu " ) , (3 , " Yak " ) , (5 , " Mouse " ) , (3 , "Dog " ) ) , 2)  
c: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[34] at parallelize at <console>:24
```

```
scala> c . countByKey  
res25: scala.collection.Map[Int,Long] = Map(3 -> 3, 5 -> 1)
```

# RDD Operations: Actions

first

reduce

takeOrdered

countByKey

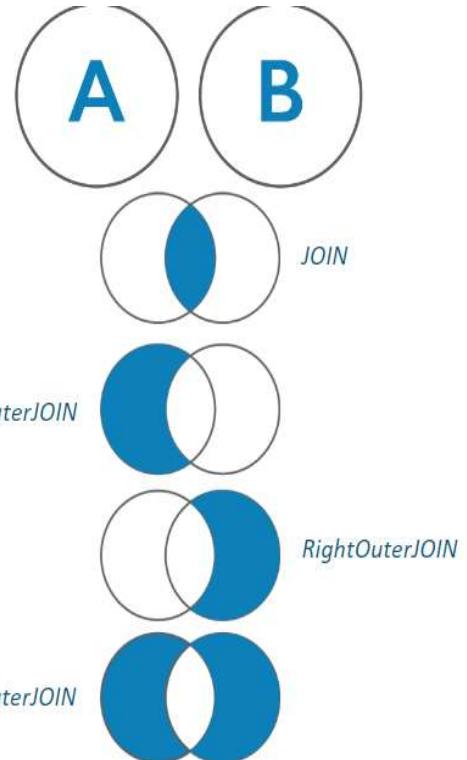
aggregate

```
val z = sc . parallelize ( List (" a " , " b " , " c " , " d " ,  
," e " , " f " ) ,2)  
z . aggregate ( "" ) ( _ + _ , _ + _ )
```

```
scala> val z = sc . parallelize ( List (" a " , " b " , " c " , " d " , " e " , " f " ) ,2)  
z: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[66] at parallelize at <console>:24  
  
scala> z . aggregate ( "" ) ( _ + _ , _ + _ )  
res22: String = " a b c d e f "
```

# RDD Joins

- Spark allows *easy manipulation* of *multiple data sets* with out-of-the-box join operators
- Spark Pair RDDs provide *join*, *leftOuterJoin*, *rightOuterJoin*, *fullOuterJoin* methods to perform respective joins
- Programmers need to create the RDDs carefully, as the *keys* are *automatically chosen by the join operation*
- Pair RDDs also provide the *cartesian* and *cogroup* methods to perform the *respective operations*



# DAG Representation

```

1 val r00 = sc.parallelize(0 to 9)
2 val r01 = sc.parallelize(0 to 90 by 10)
3 val r10 = r00 cartesian r01
4 val r11 = r00.map(n => (n, n))
5 val r12 = r00 zip r01
6 val r13 = r01.keyBy(_ / 20)
7 val r20 = Seq(r11, r12, r13).foldLeft(r10)(_ union _)
    
```

