



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

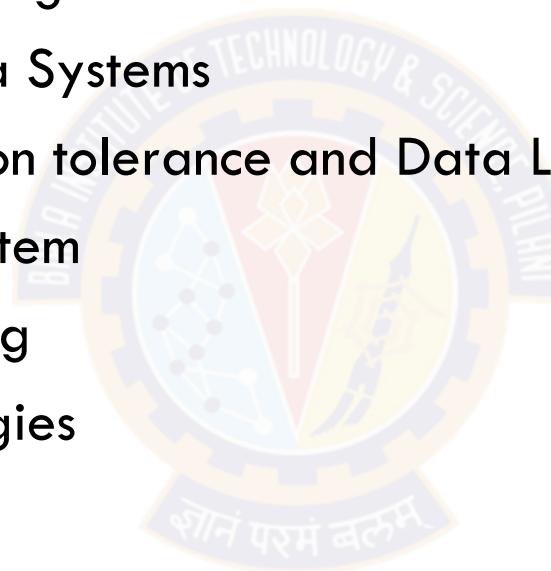
DSECL ZG 522: Big Data Systems

Session 1.1: Introduction to Big Data

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

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)

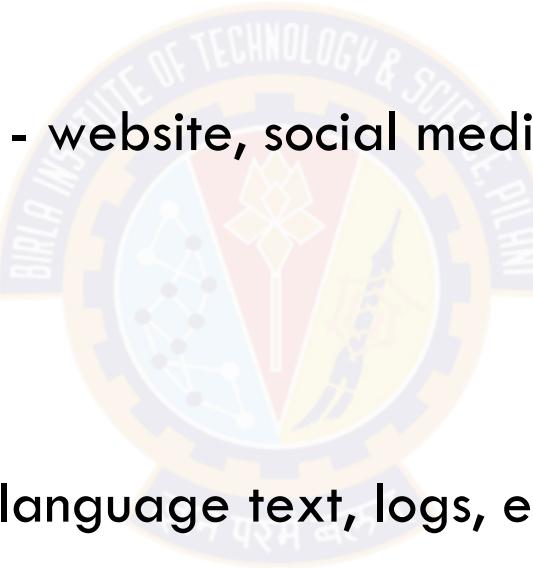
Topics for today

- 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



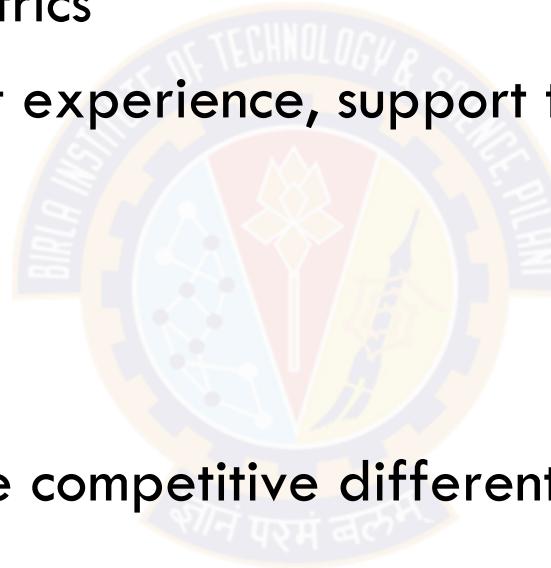
Example of a data-driven Enterprise: A large online retailer (1)

- 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.

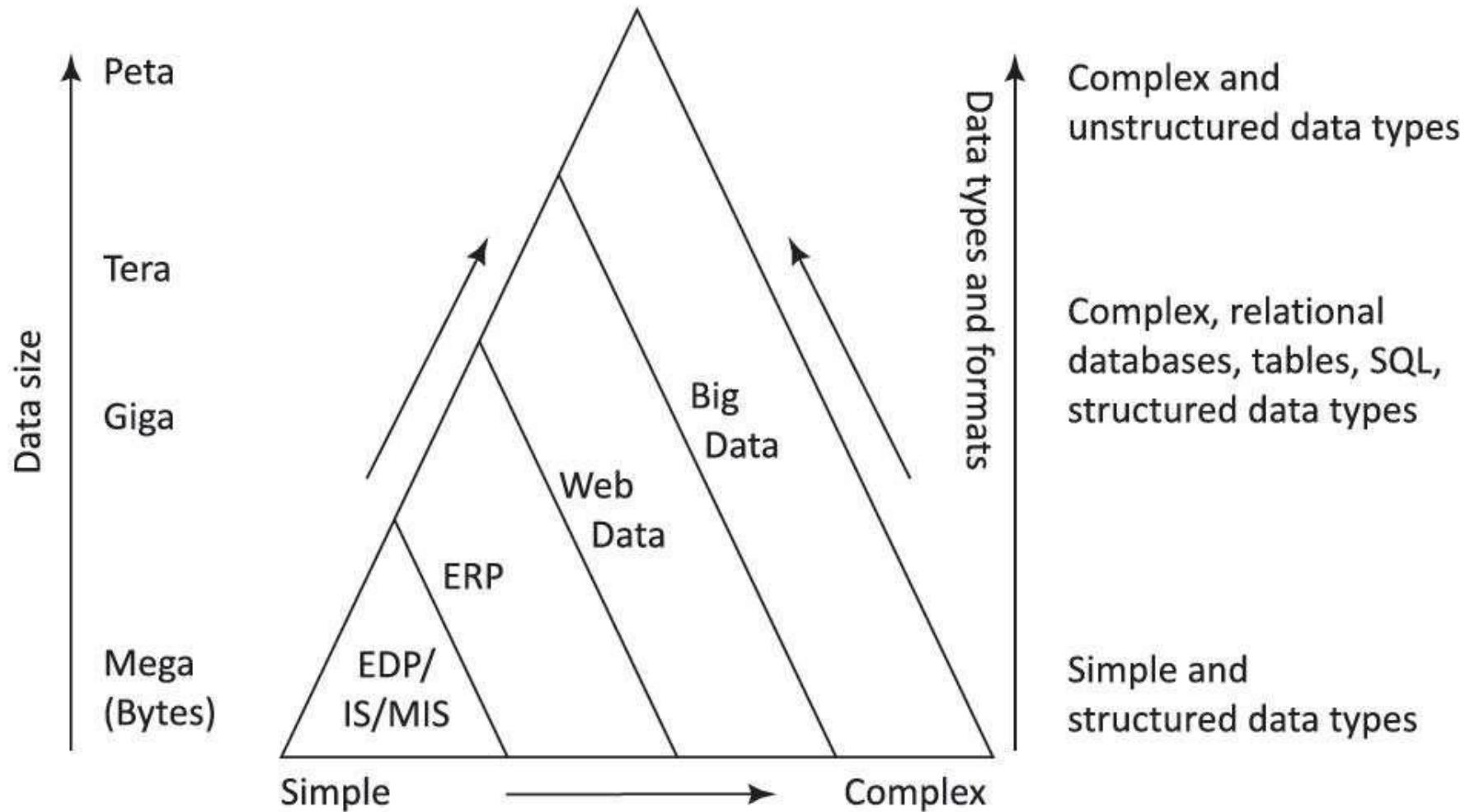


Example of a data-driven Enterprise: A large online retailer (2)

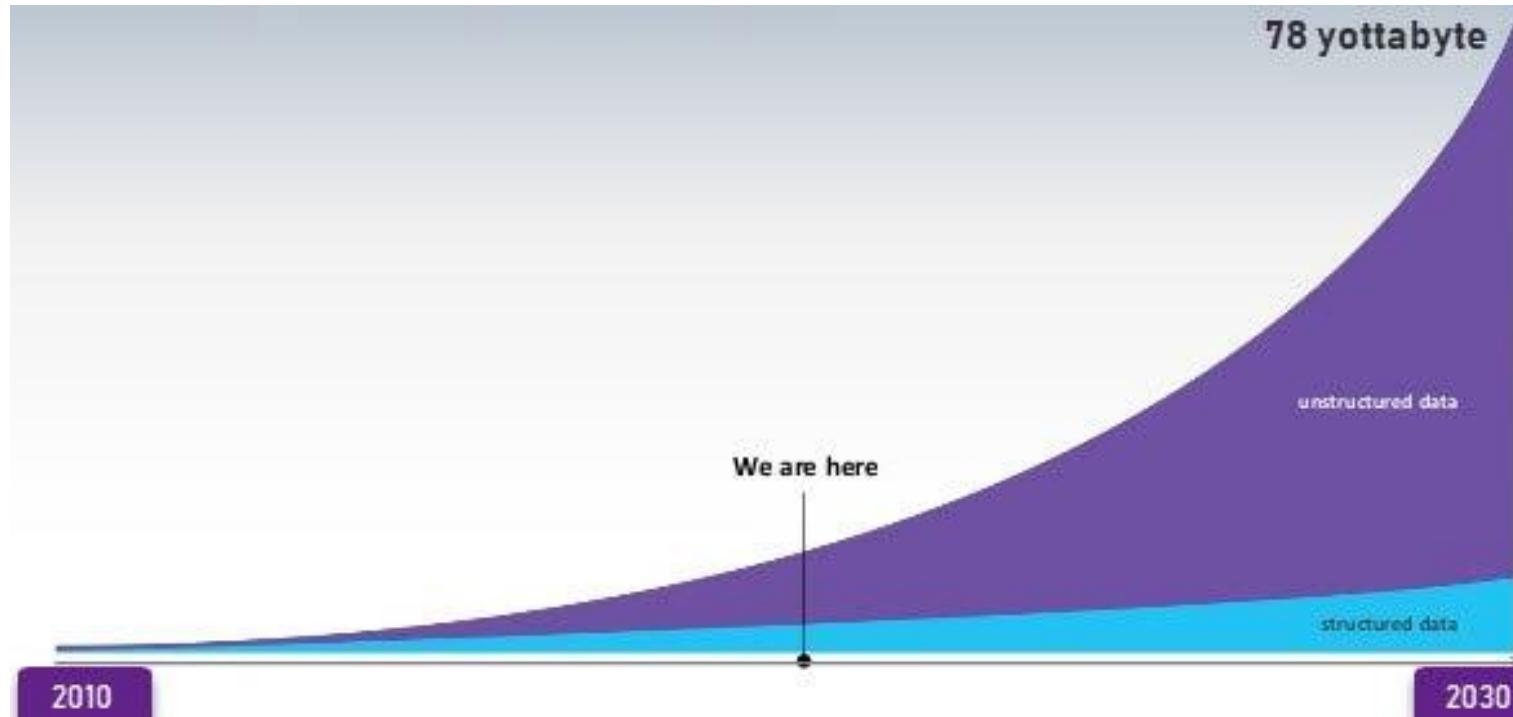
- What is this data used for
 - ✓ User profiling for better shopping experience
 - ✓ Operations efficiency metrics
 - ✓ Improve customer support experience, support training
 - ✓ Demand forecasting
 - ✓ Product marketing
 - ✓ ...
- Data is the only way to create competitive differentiators, retain customers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



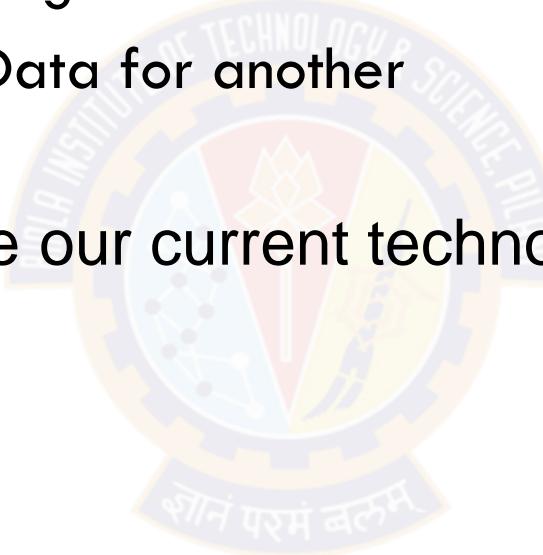
[Source : What is Big Data?](#)

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

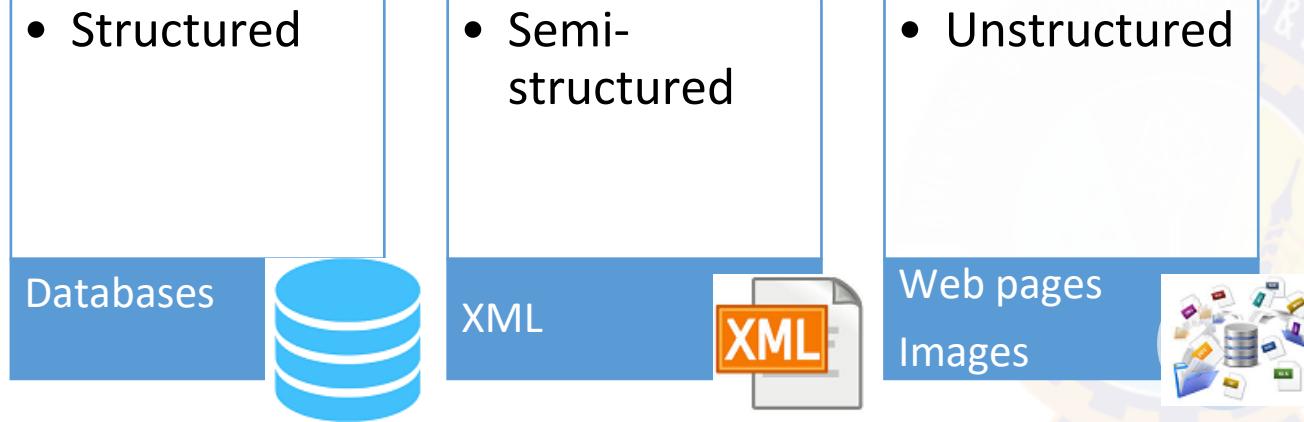
- Volume
- Communication
- Speed of Generating
- Meaningful Analysis



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**

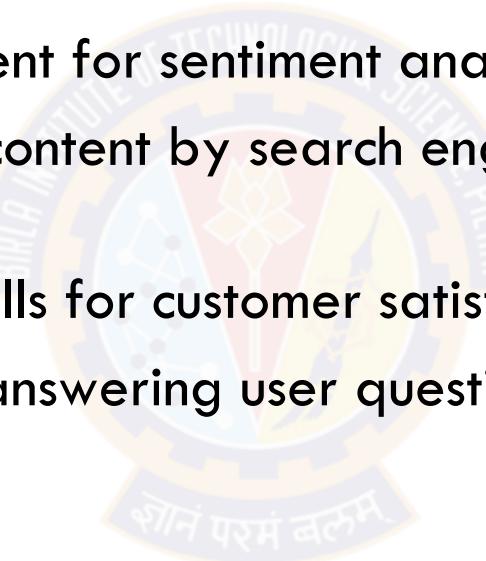
Data classification



- 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

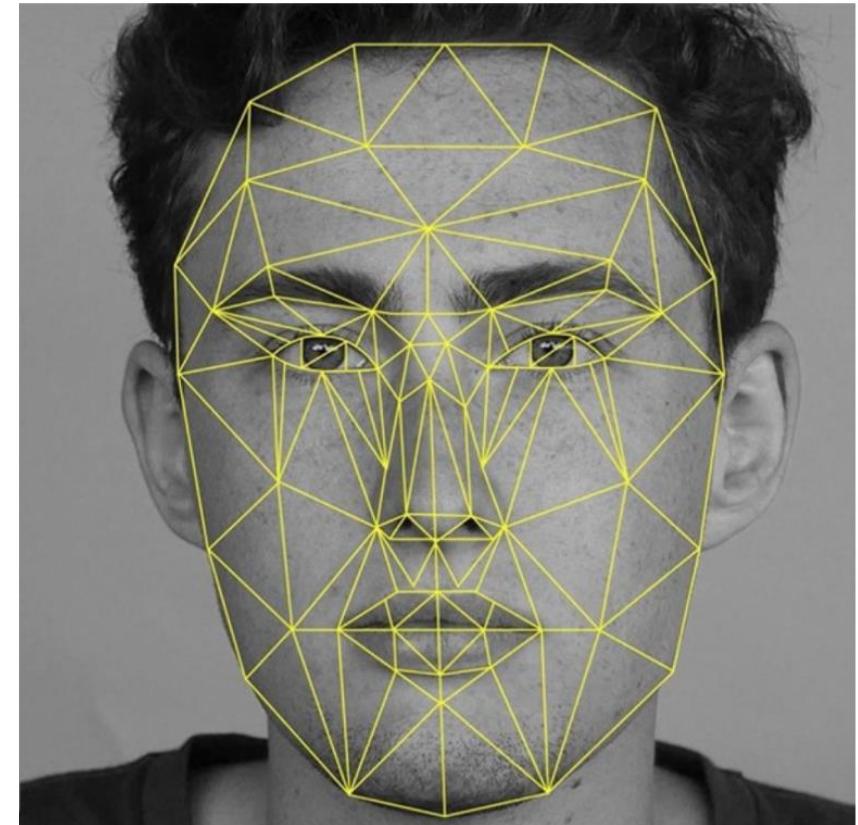
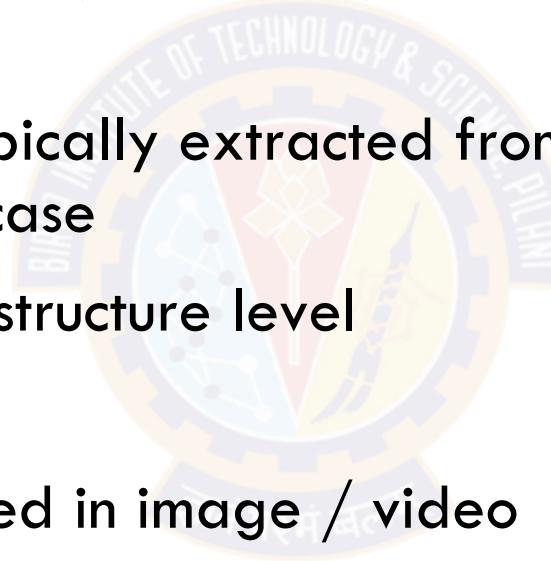
- No explicit data and schema separation
- Models real life situations better because attributes for every record could be different
- Easy to add new attributes
- XML, JSON structures
- Databases typically support flexible ACID properties, esp consistency of replicas
- Typically used by Systems of Engagement,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

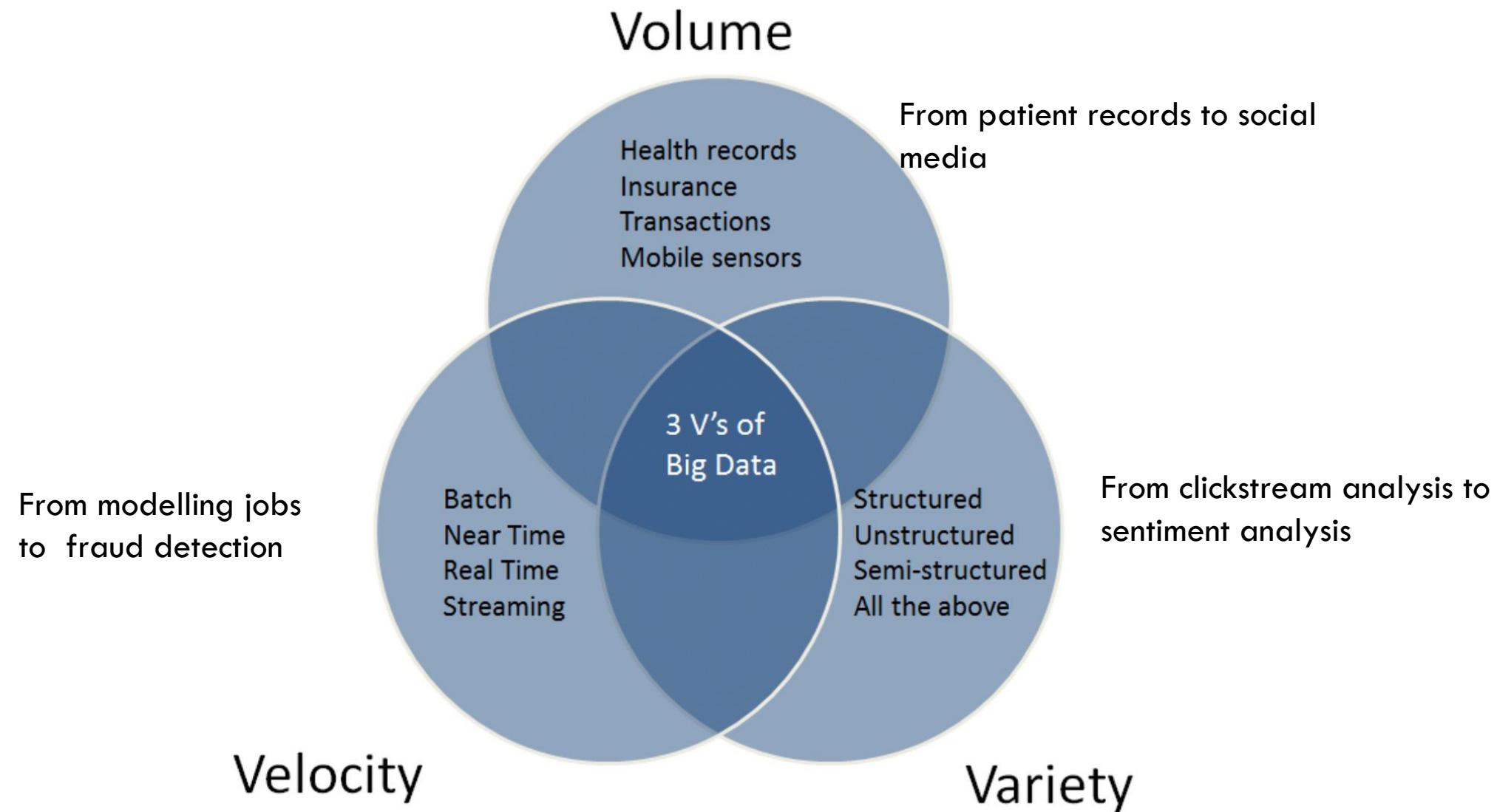
- More real-life data
 - ✓ video, voice, text, emails, chats, comments, reviews, blogs ...
- There is some structure that is typically extracted from the data depending on the use case
 - ✓ image adjustments at pixel structure level
 - ✓ face recognition from video
 - ✓ tagging of features extracted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



Unstructured Data (2)

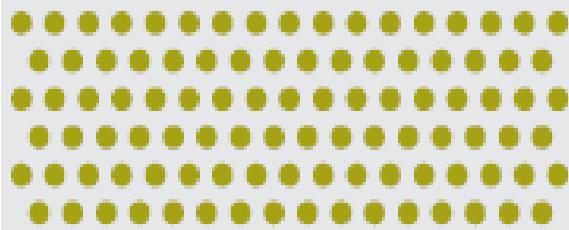
- What can we do with it ?
 - ✓ Data mining
 - Association rule mining, e.g. market basket or affinity analysis
 - Regression, e.g. predict dependent variable from independent variables
 - Collaborative filtering, e.g. predict a user preference from group preferences
 - ✓ NLP - e.g. Human to Machine 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

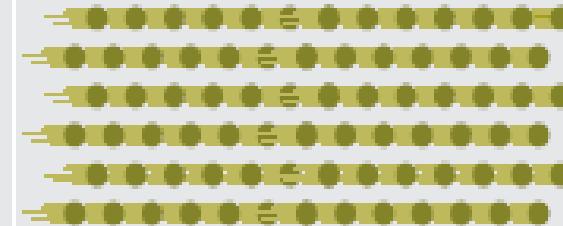
Variety



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

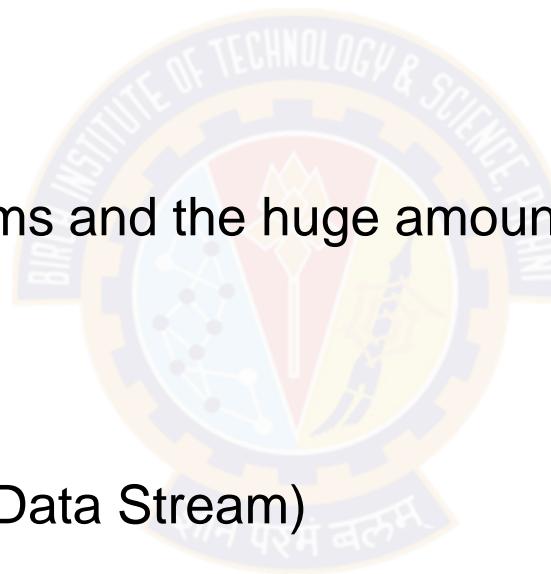


Data uncertainty

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

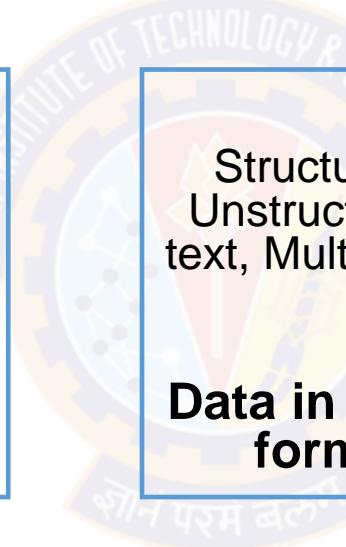
Structured, Unstructured, text, Multimedia

Data in many forms

Veracity

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

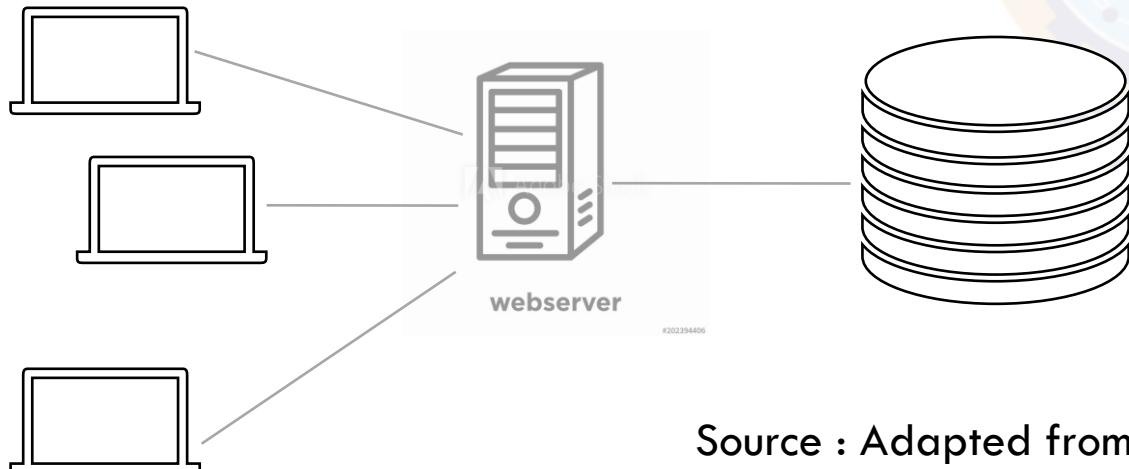
Data in Doubt



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



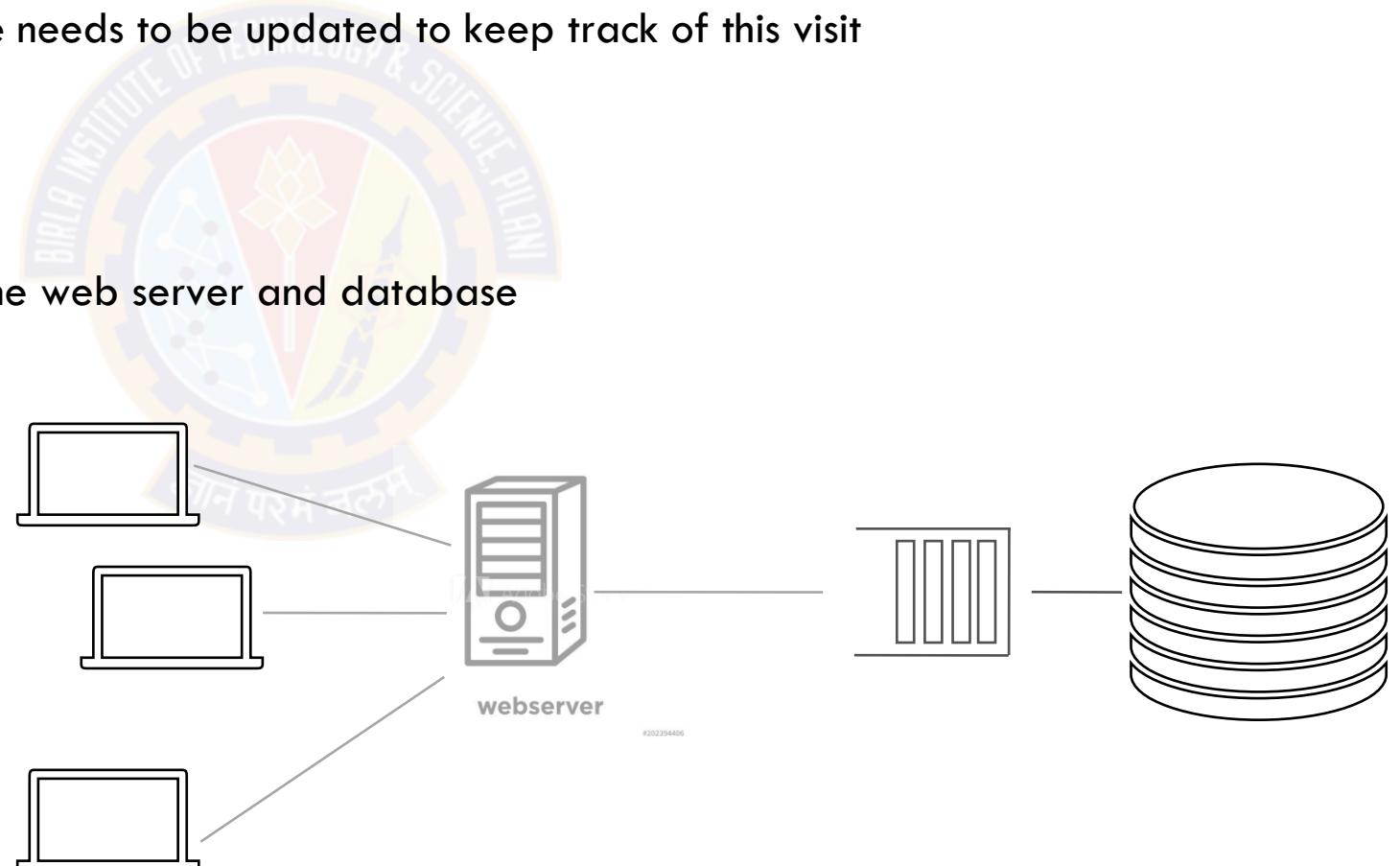
Column	Data Type
Id	Integer
User_ID	Integer
Page_URL	Varchar
Page_count	Long

Source : Adapted from Big Data by Nathan Marz

Scaling with intermediate layer

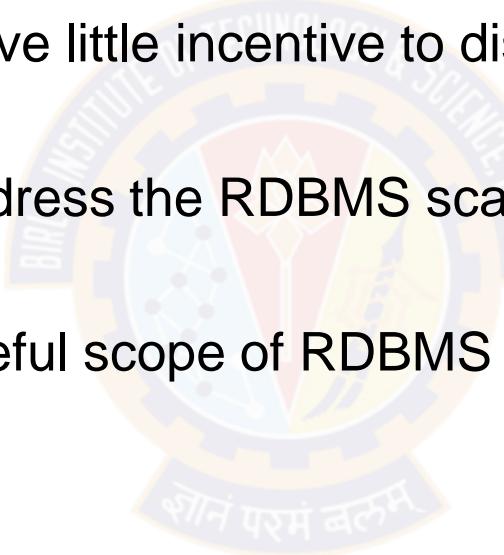
Using a queue

- 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

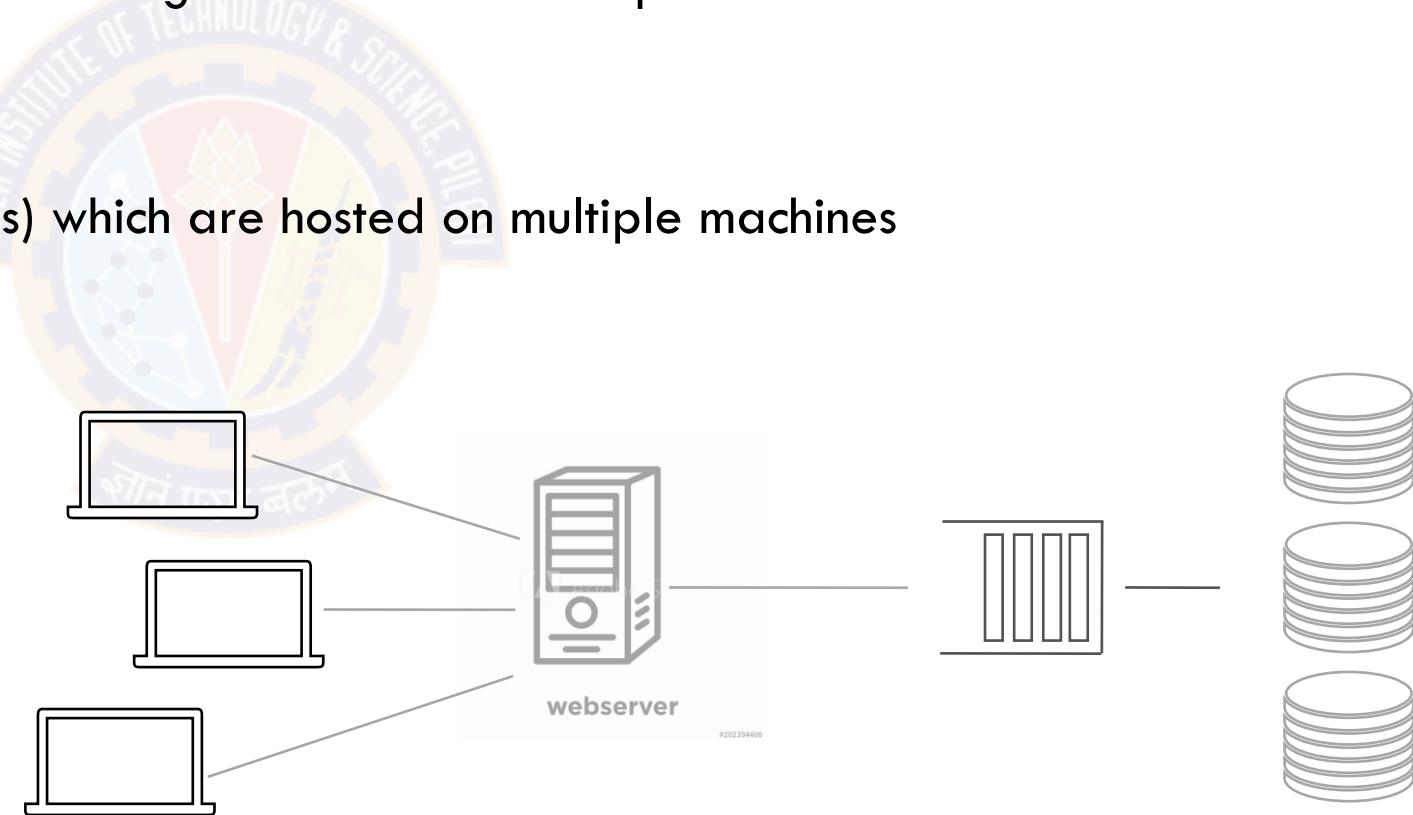


Provocative statement:

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

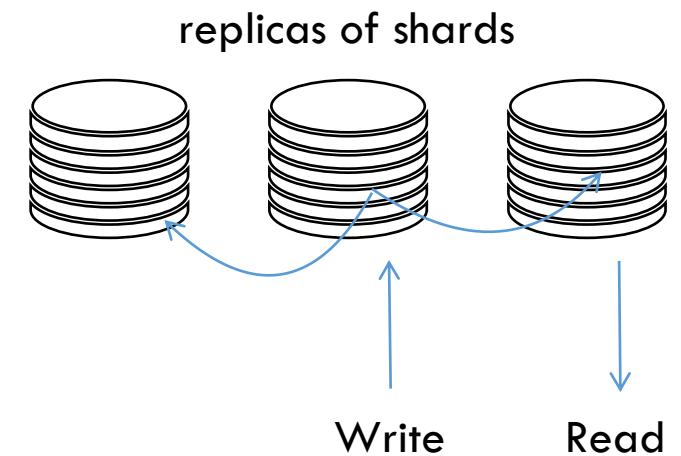
Scaling with Database Partitions (Sharding)

- 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!



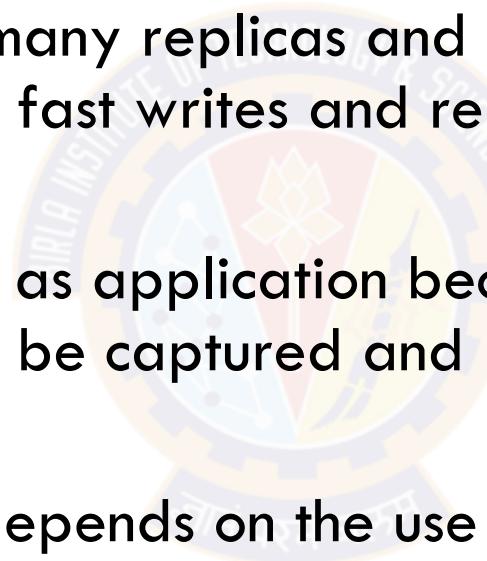
Issues with RDBMS sharding

- 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 ?

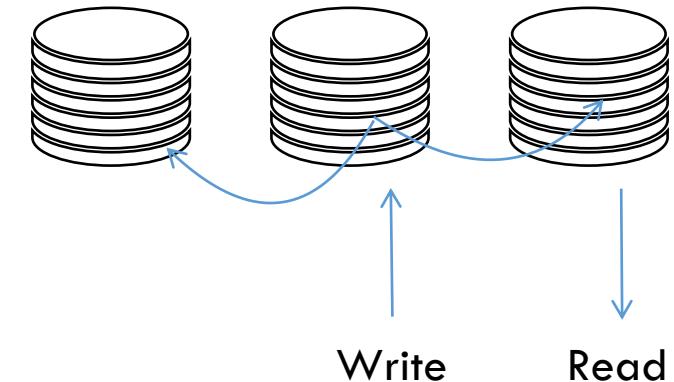


Issues with RDBMS (1)

- 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

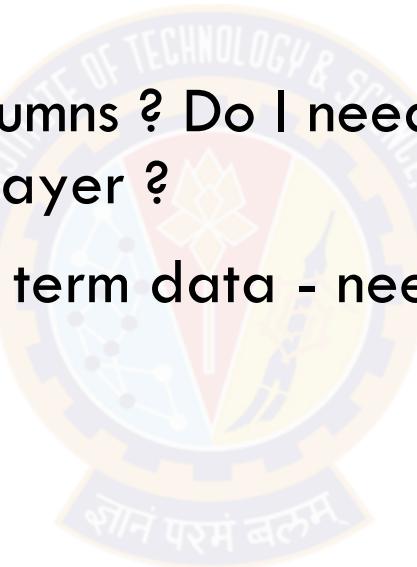
ACID to BASE Transformation in NoSQL era

[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



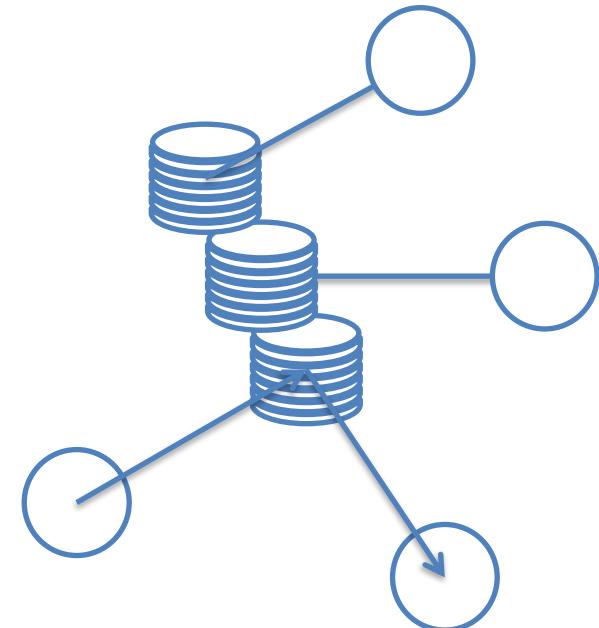
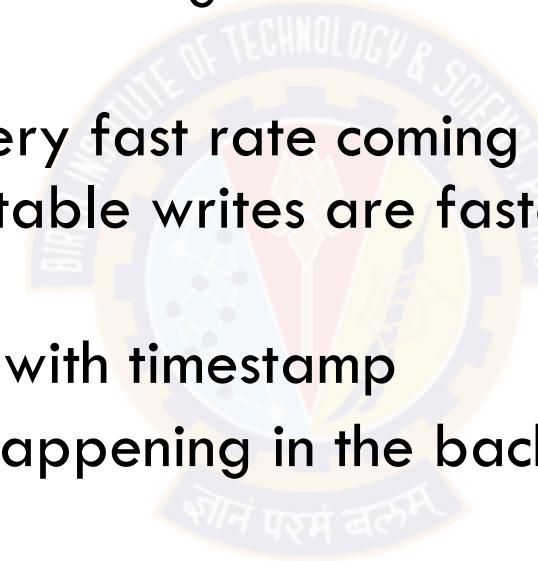
key-value document

graph

t1, k t2, k t3, k ...

Characteristics of Big Data Systems (2)

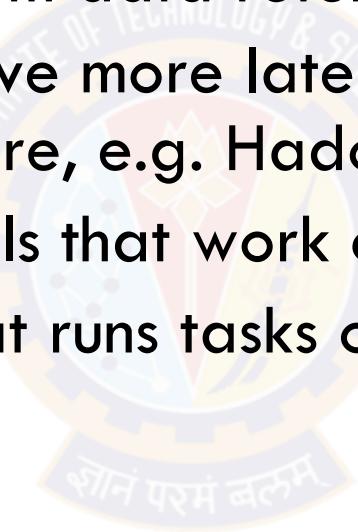
- Application specific consistency models
 - ✓ a reader may read a replica that's has not been updated yet as in “read preference” options in MongoDB
 - ✓ e.g. comments on social media
- Handles high data volume, at very fast rate coming from variety of sources because immutable writes are faster with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



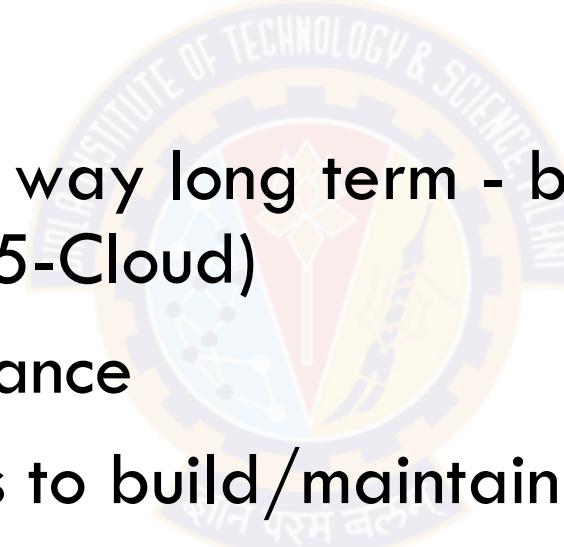
Challenges in Big Data Systems (1)

- 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



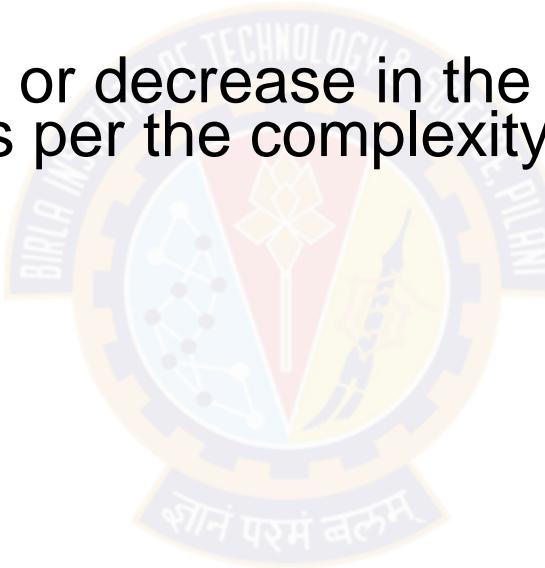
Topics for today

- 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



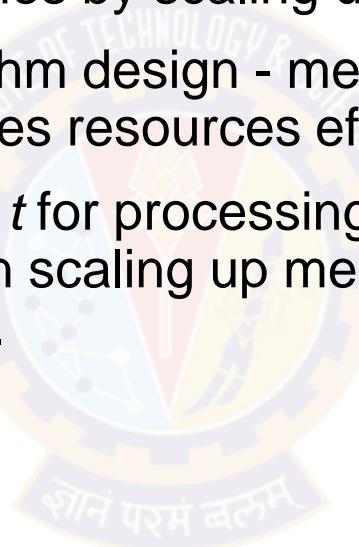
Needs of Big Data Systems

- 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



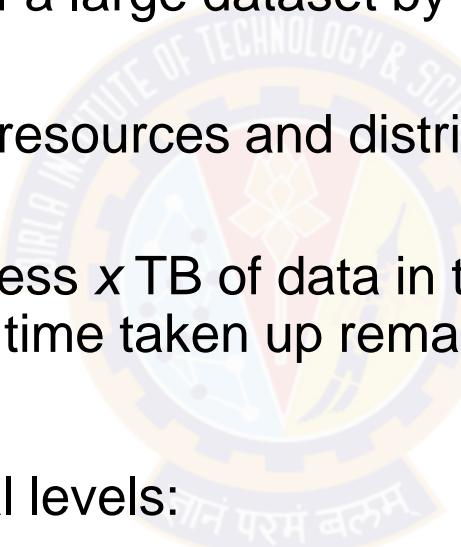
Vertical Scalability (Scaling Up)

- 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

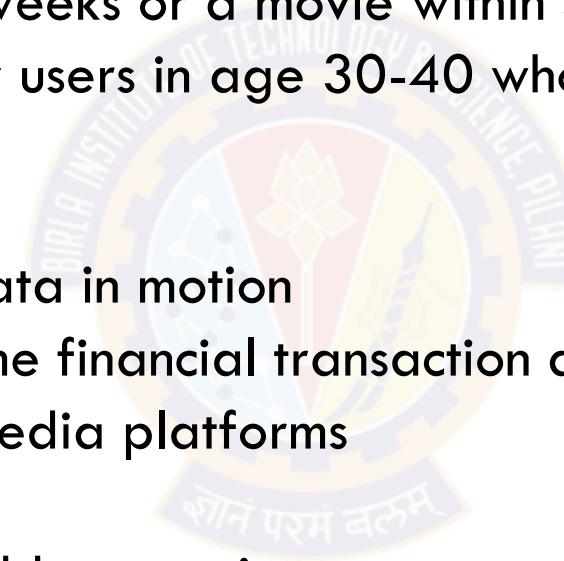
- (i) on-demand service
- (ii) resource pooling,
- (iii) scalability,
- (iv) accountability, and
- (v) broad network access.



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

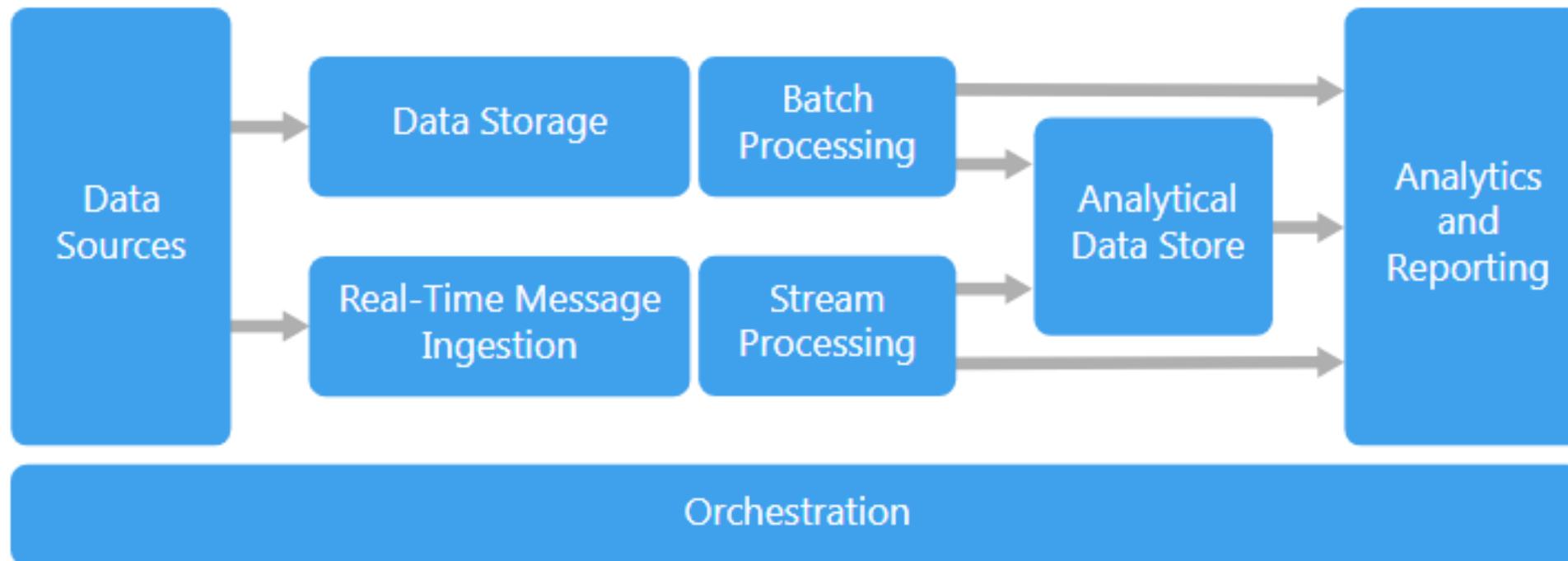
Types of Big Data solutions

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

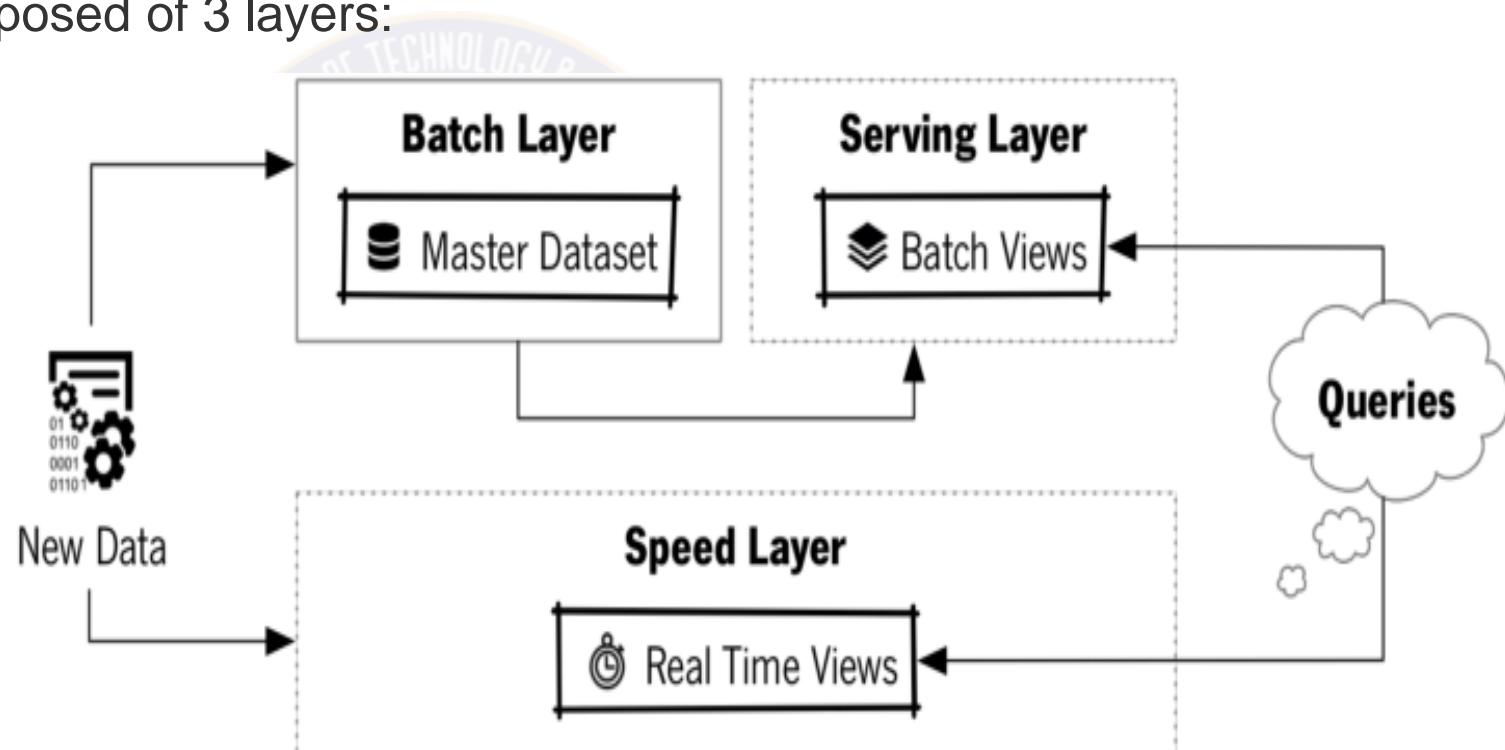
- Designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



[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.

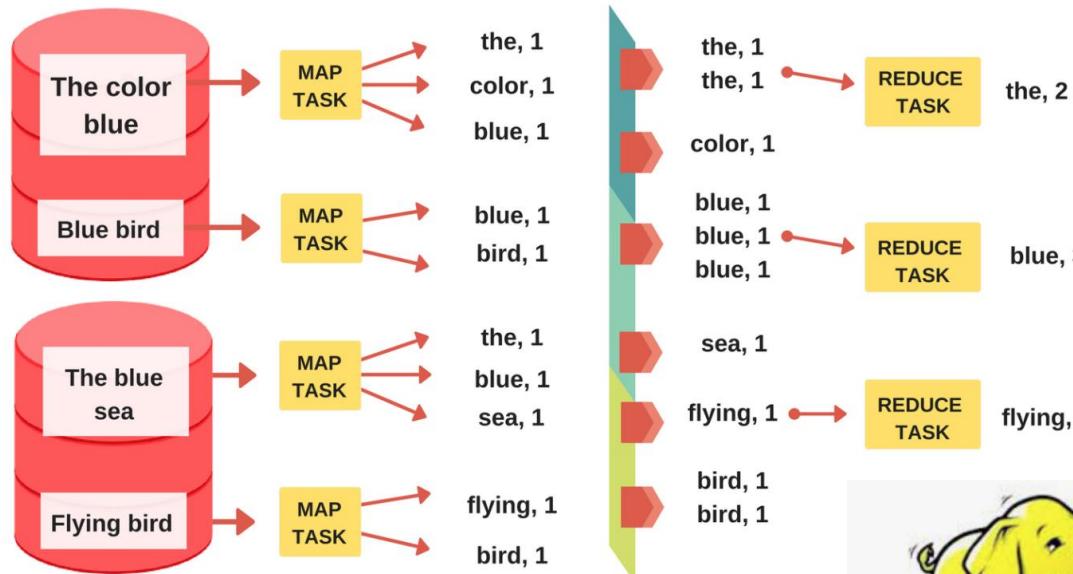


e.g. social data



e.g. medical images

Big Data Systems Components (2)



e.g. search scans on unindexed docs



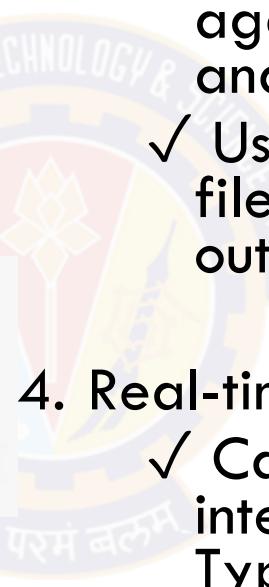
e.g. credit card transactions for fraud detection

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

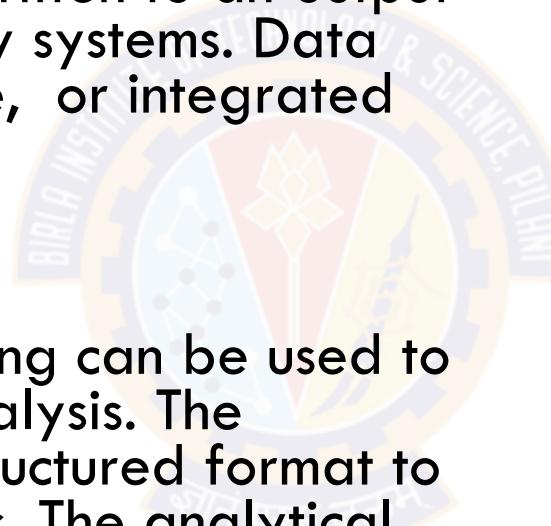
- ✓ Capture data from real-time sources and integrate with stream processing. Typically these are in-memory systems with optional storage backup for resiliency.



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



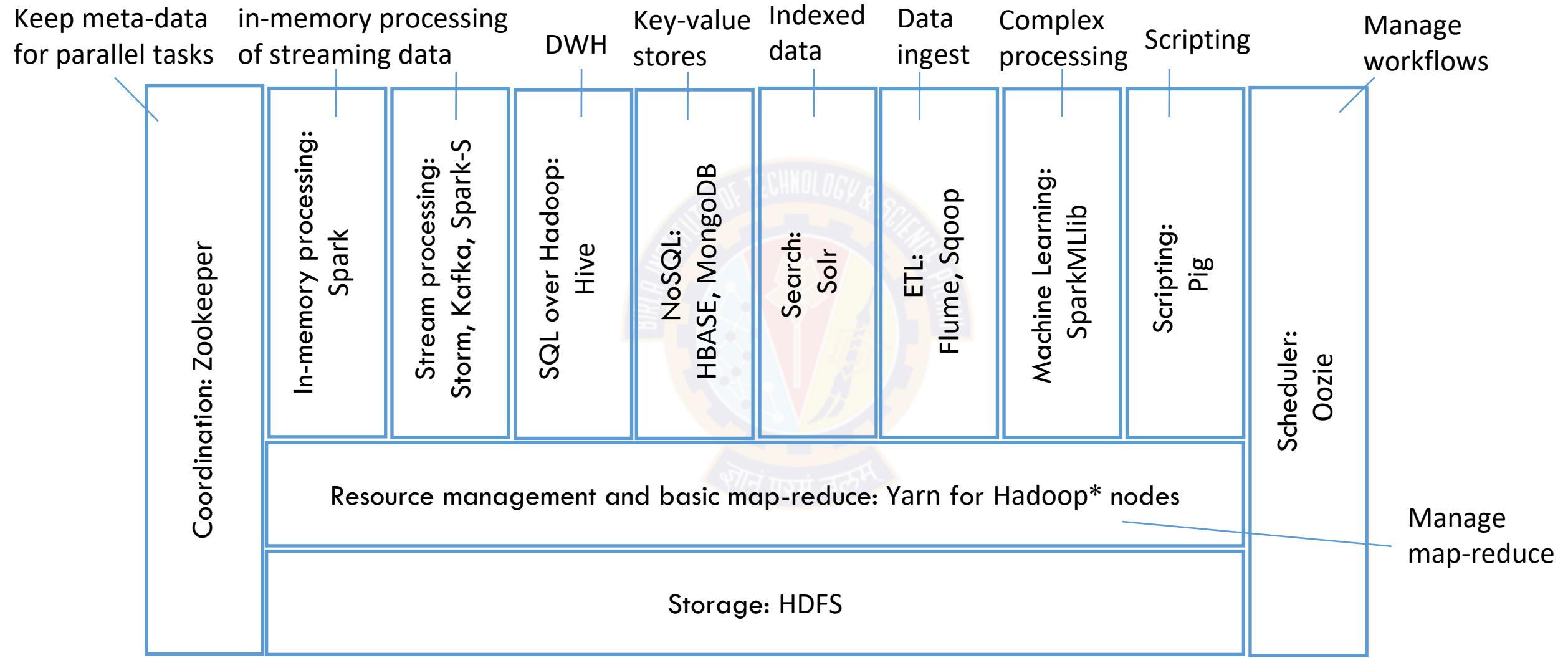
7. Analysis and reporting

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)





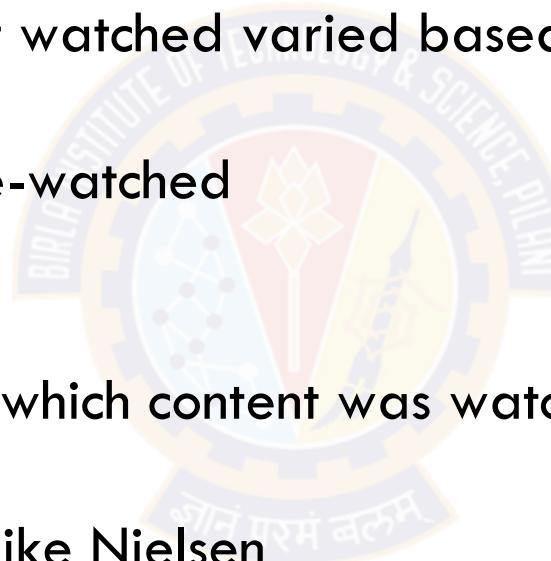
Case Study: Netflix

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

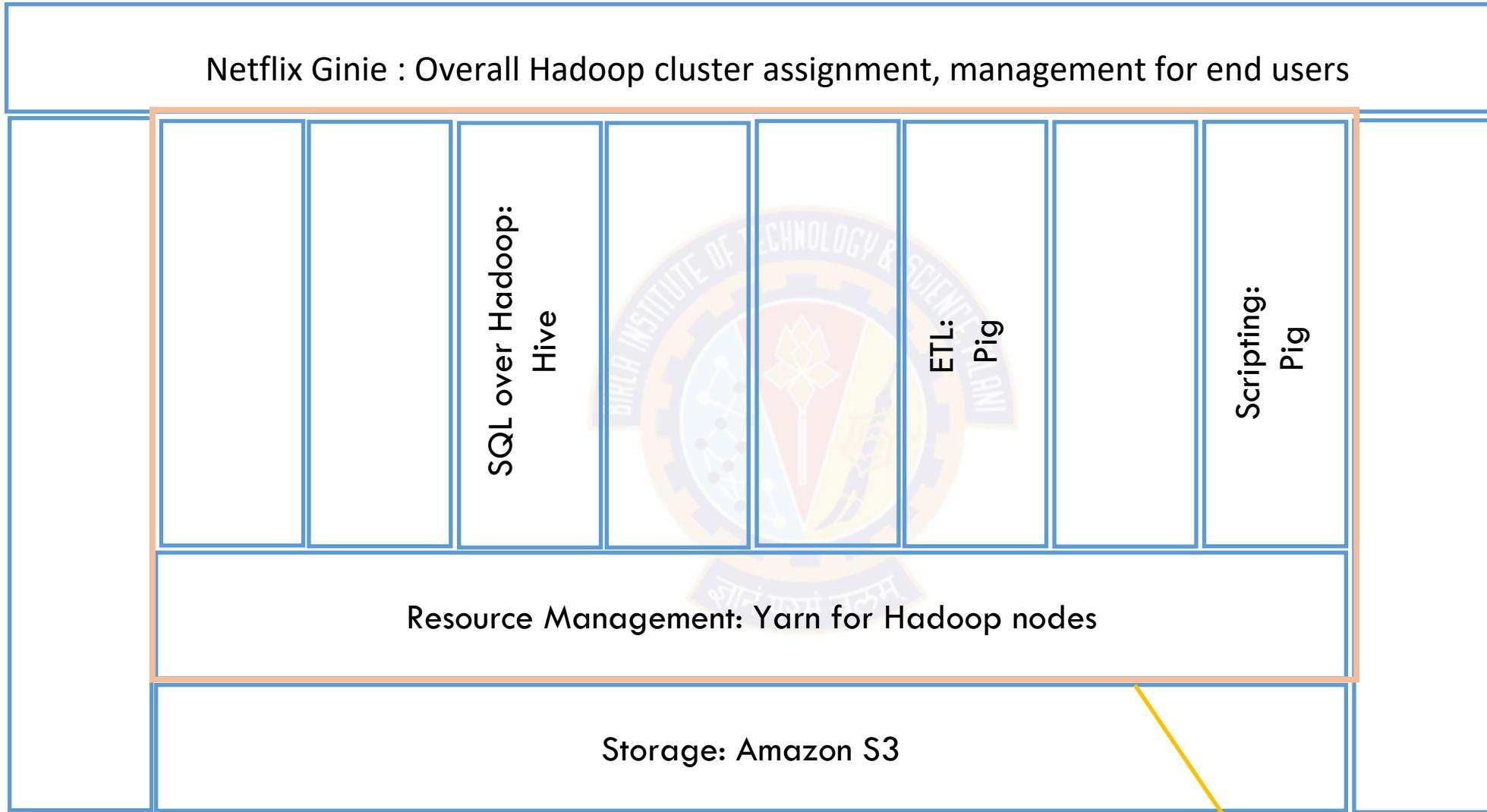
- Date content watched
- The device on which the content was watched
- How the nature of the content watched varied based on the device
- Searches on its platform
- Portions of content that got re-watched
- Whether content was paused
- User location data
- Time of the day and week in which content was 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

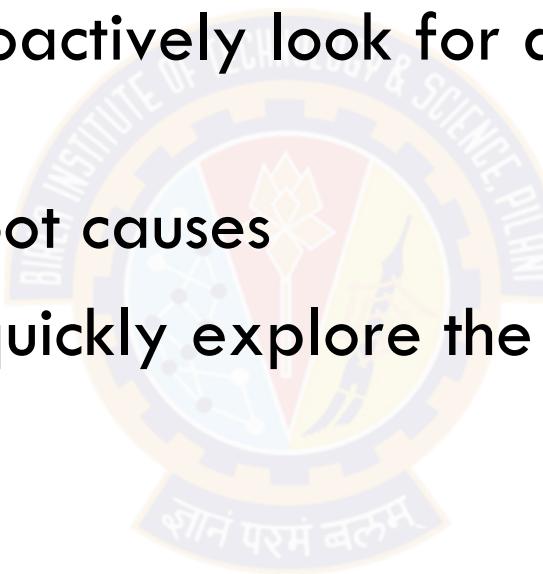


Case Study: IT Ops

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



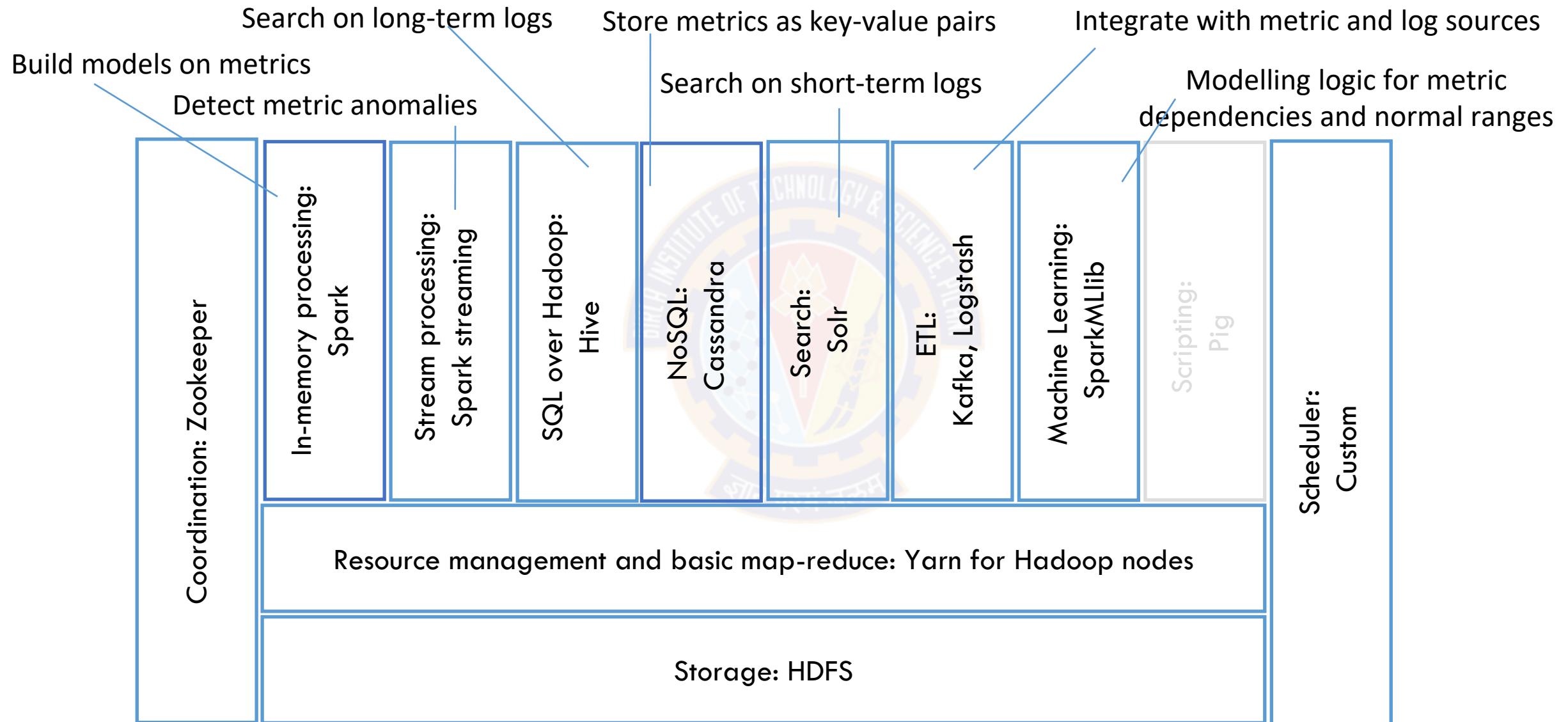
Real-time streaming analysis of metrics

- 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)

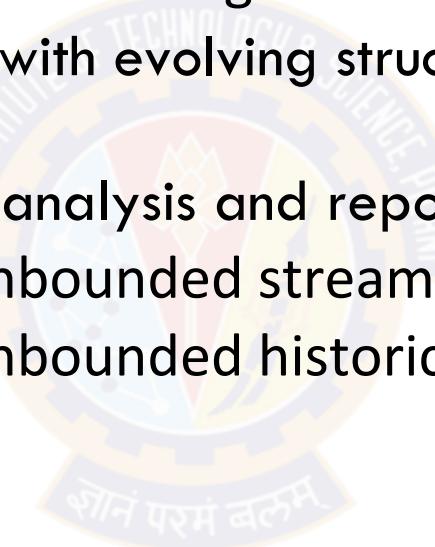
Big Data Platform



Where will you apply this architecture style

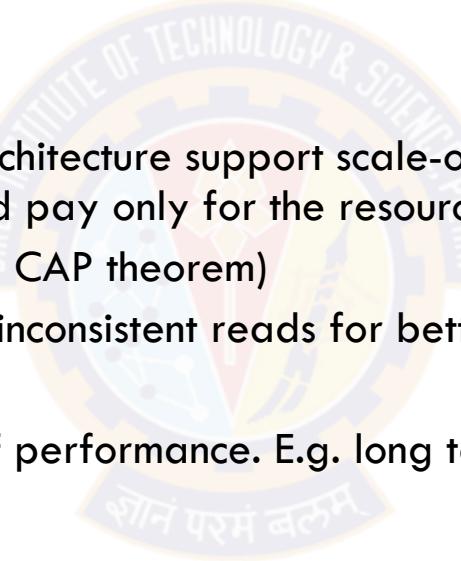
- Consider this architecture style when you need to:

- ✓ 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.



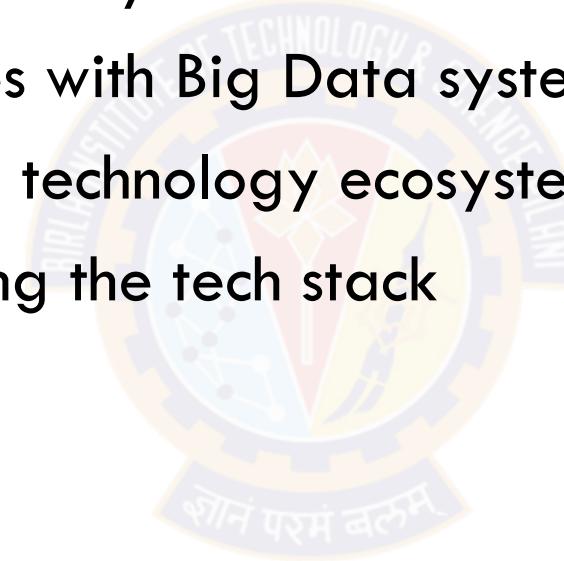
Topics for today

- 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



Locality of Reference (LoR)

Spatial and temporal locality principles for processing and storage

Context

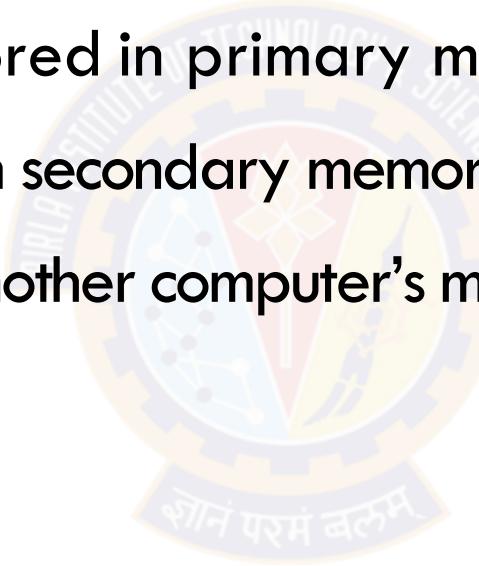
- Big Data Systems need to transport large volumes of data
 - ✓ e.g. browse and stream content on Netflix or answer analytical queries on e-commerce transaction data in Amazon
- Is there a way to reduce latency of data requests using locality of reference in the data and request origin



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->Netherlands->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.

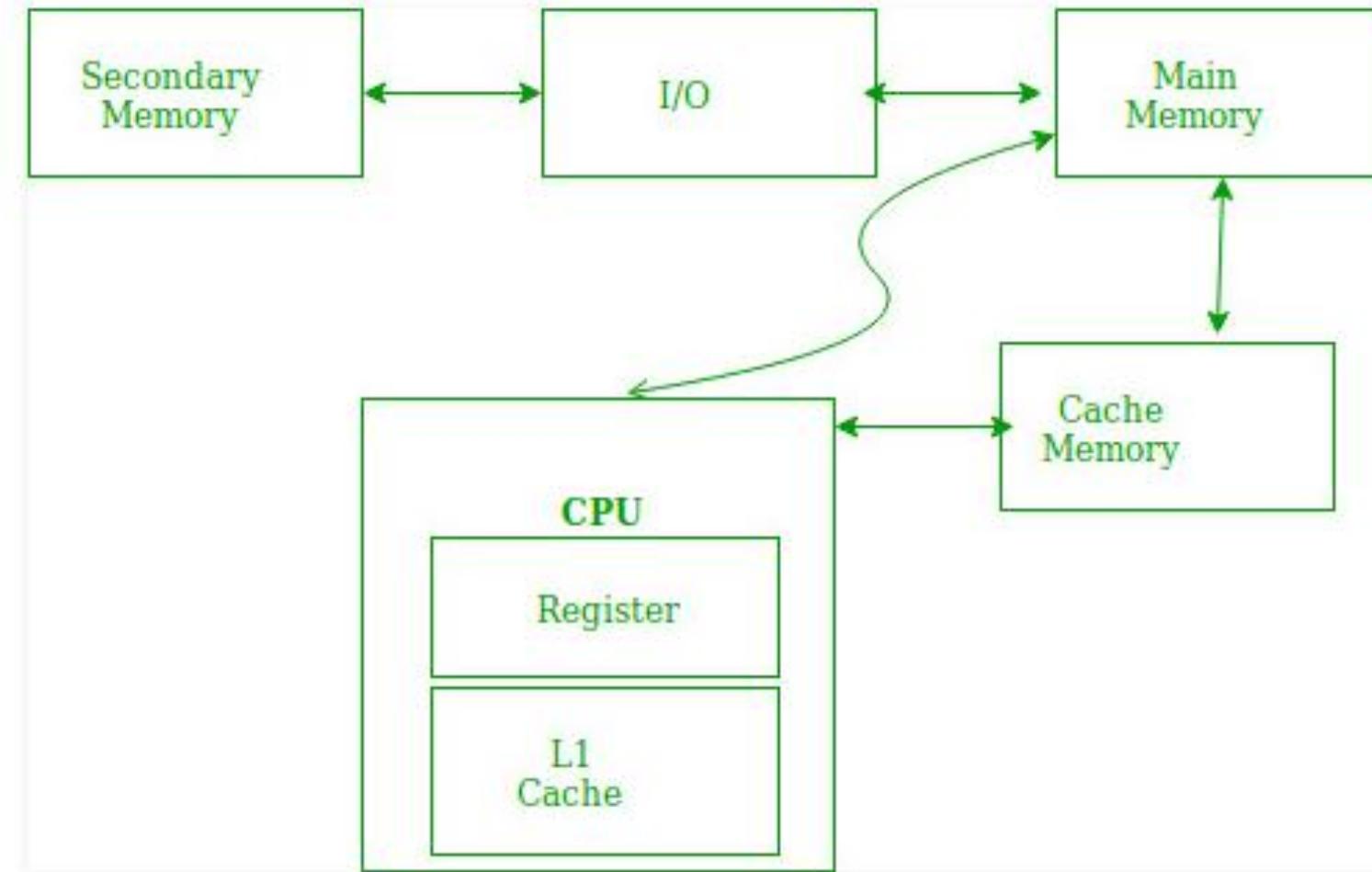
- **Classic:**



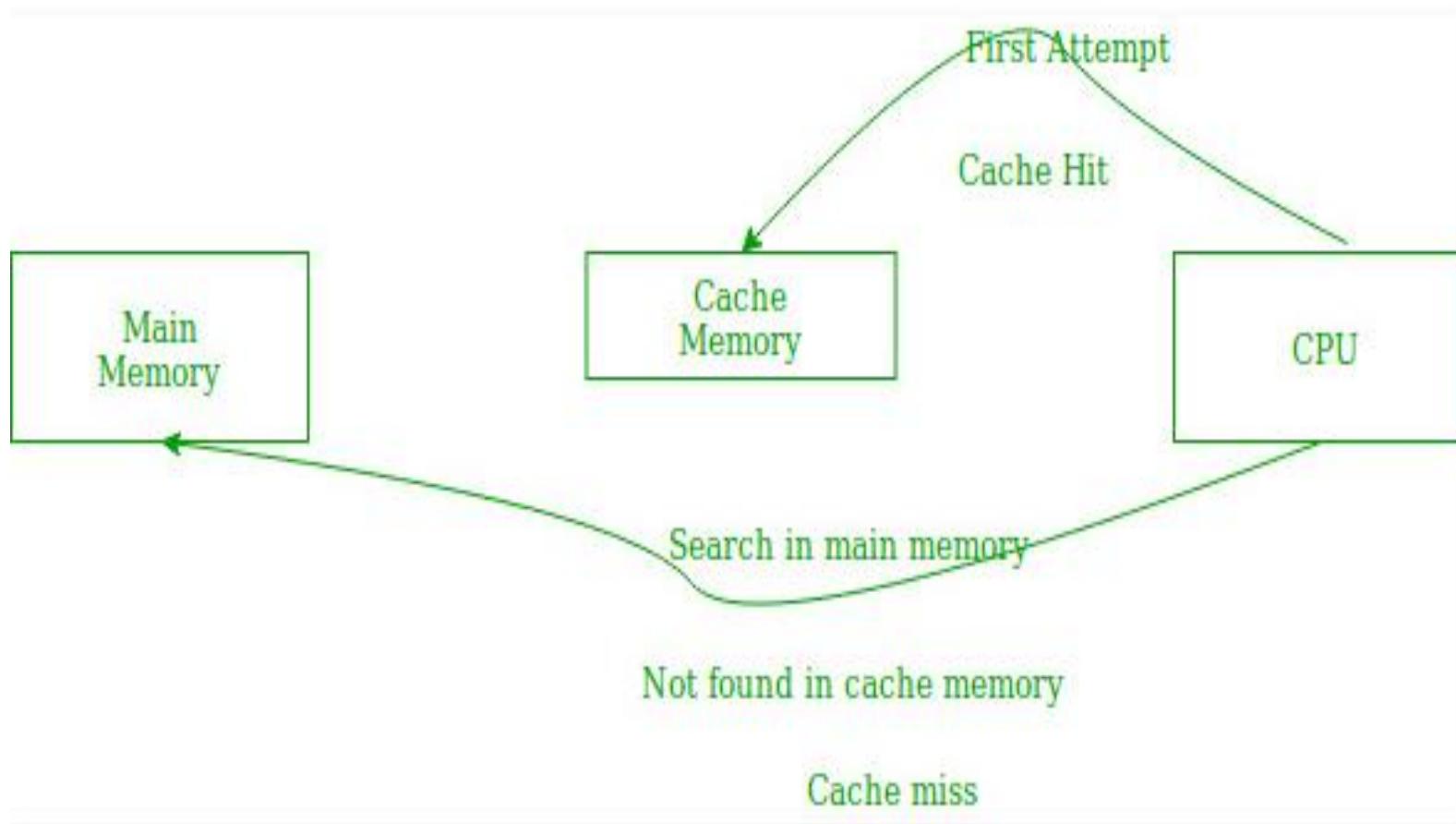
- **Modern:**



Memory hierarchy - access



Memory hierarchy - cache hit/miss



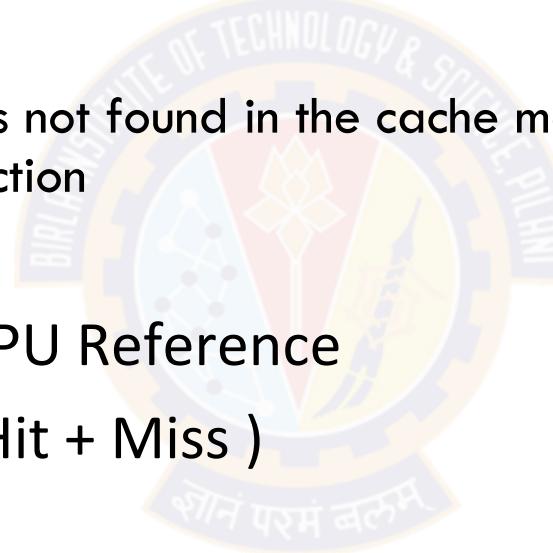
Cache performance

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



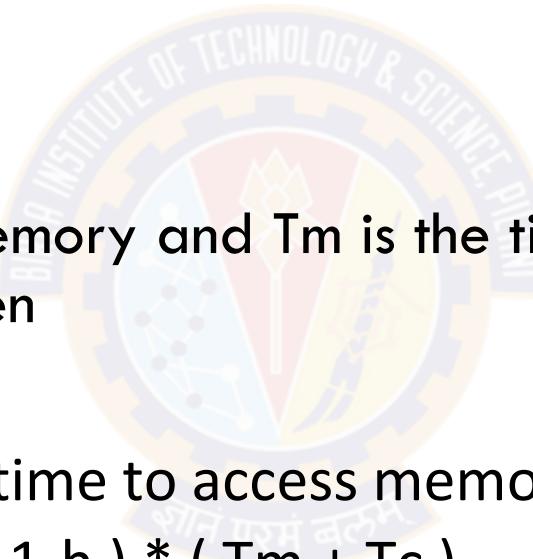
- One can generalise this to any form of cache e.g. movie request from user to nearest Netflix content cache

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

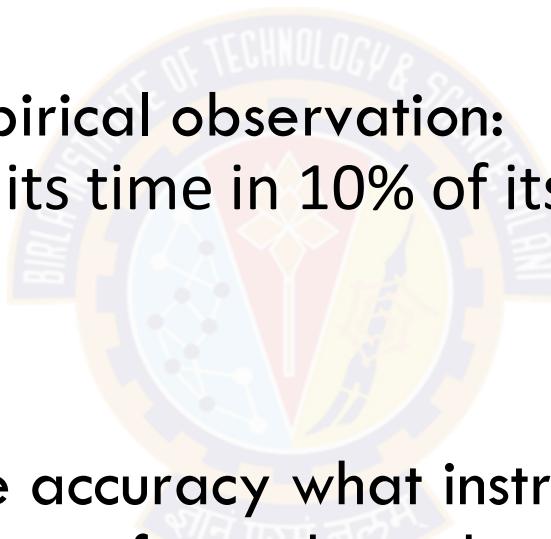
T_{avg} = Average time to access memory

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



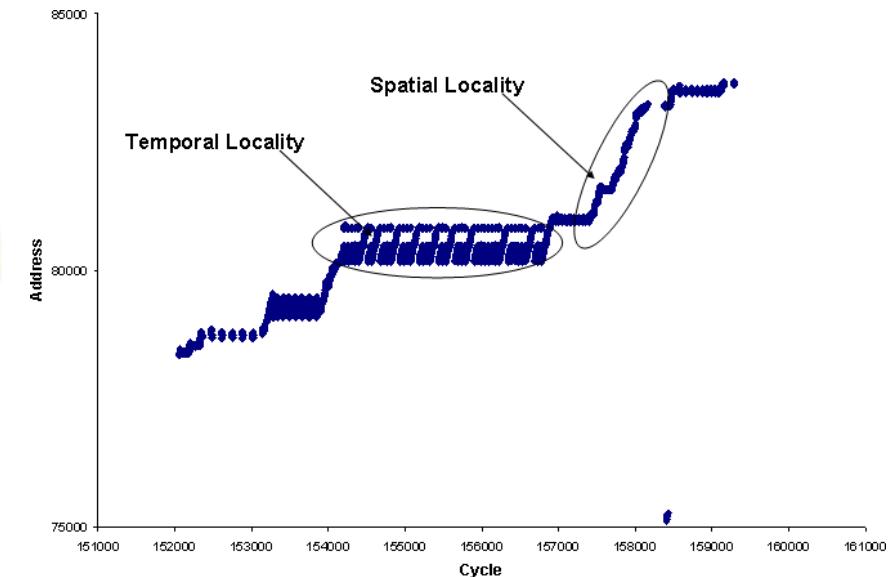
Data reference empirical evidence

- 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



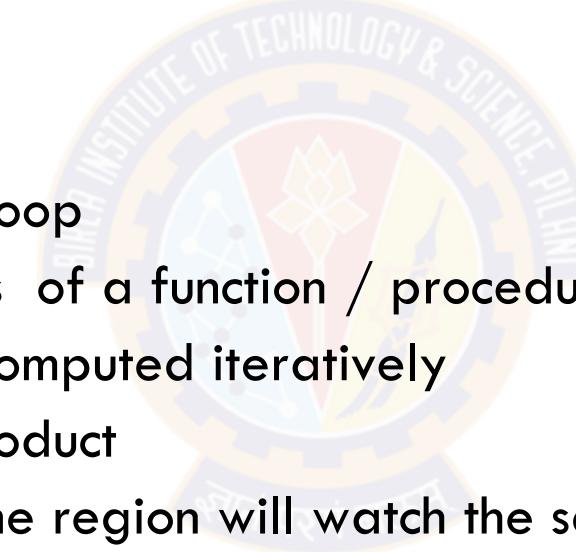
Principle of Locality of Reference (LoR)

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



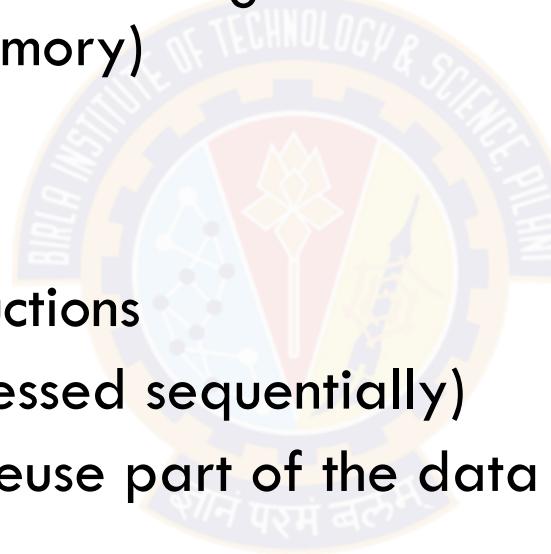
```
void swap(int x,  
         int y)  
{  
    t = x;  
    x = y;  
    y = t;  
}
```

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

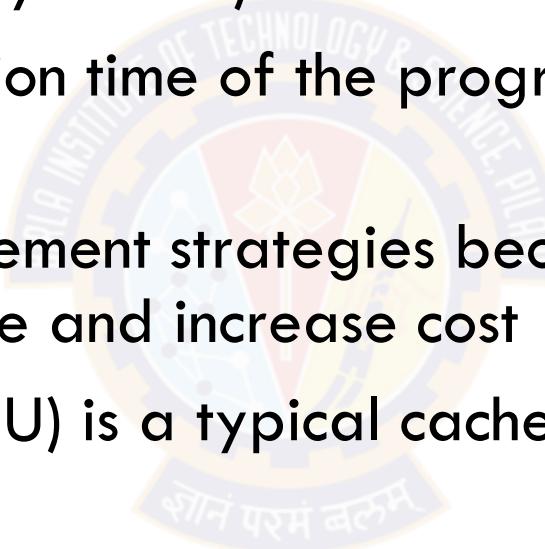
1. A linear sequence of instructions
2. Elements of an Array (accessed sequentially)
3. Another user's query will reuse part of the data file brought in for current user's query



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

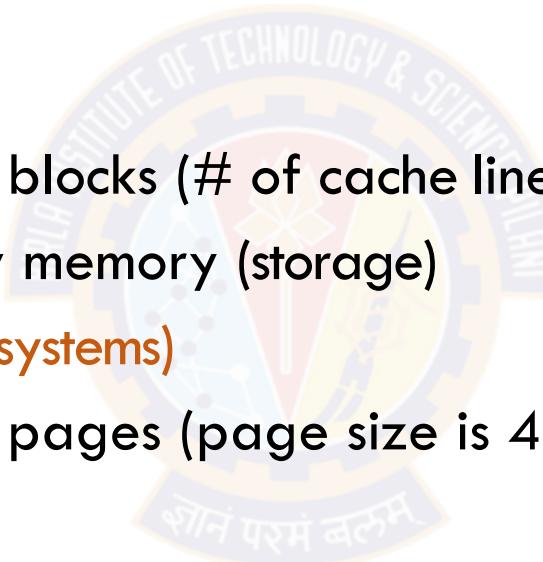
Memory hierarchy and locality of reference (1)

- 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



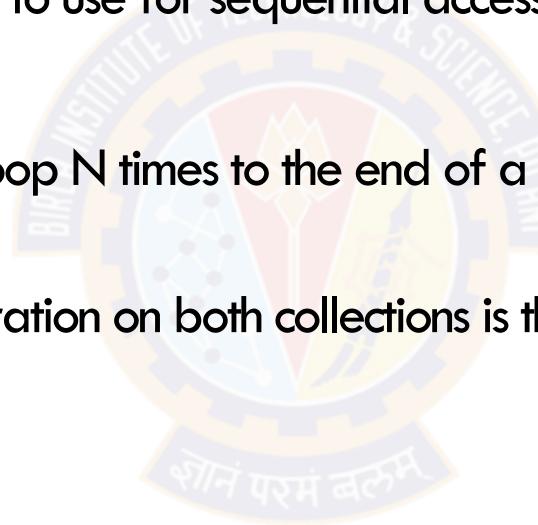
Memory hierarchy and locality of reference (2)

- 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



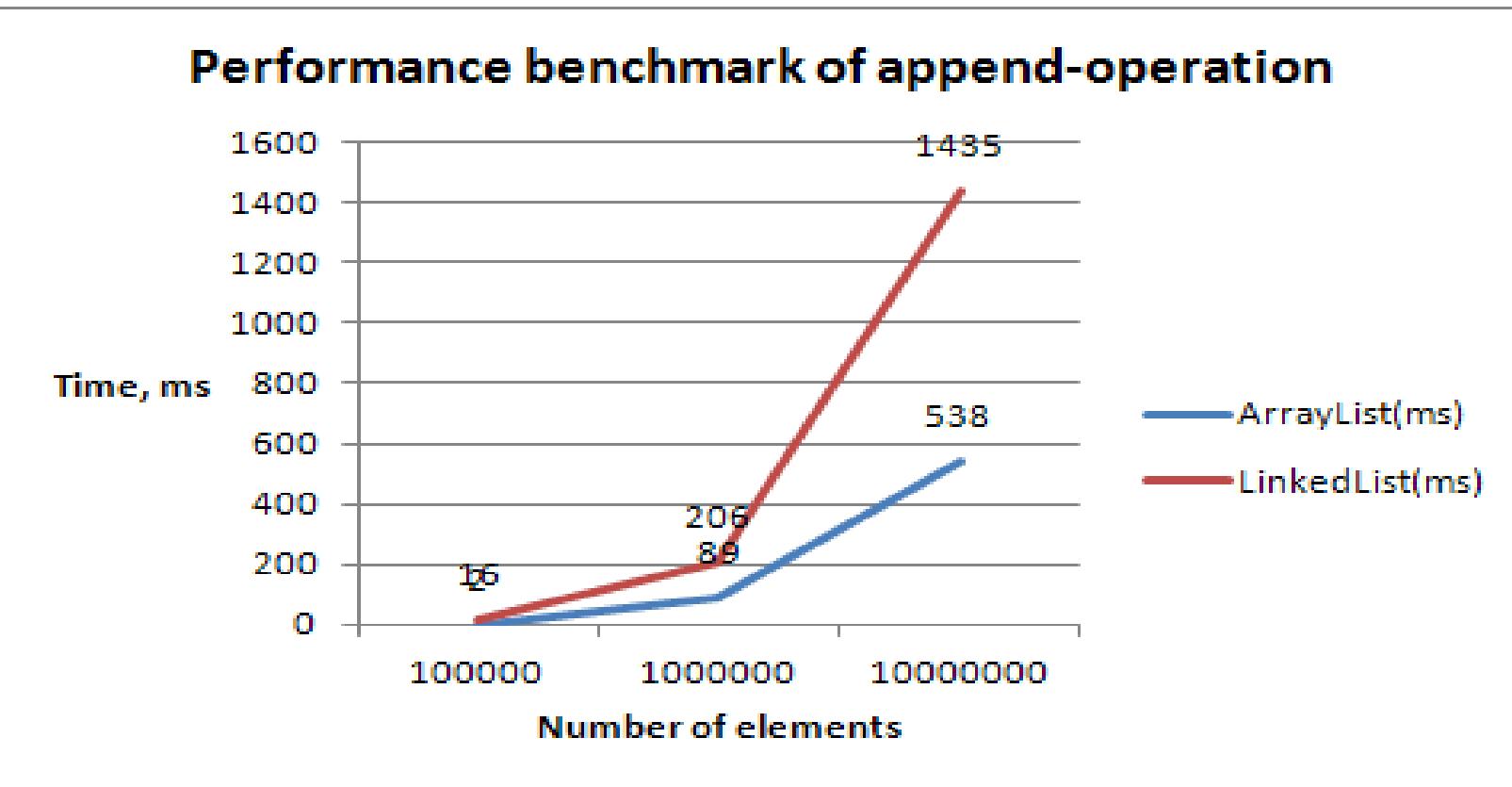
LoR and data structure choices

- 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)

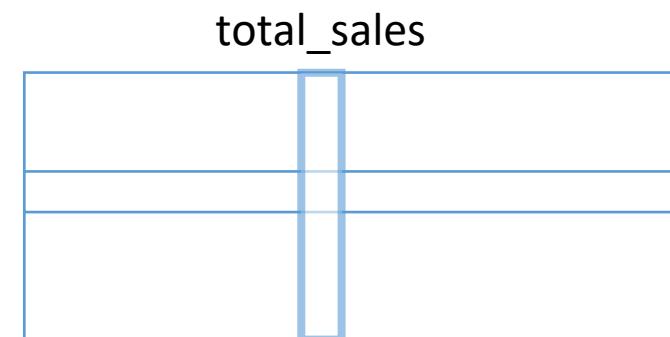


- Analysis - sequential access in arrays is faster than on linked lists on many machines, because they have a very good spacial locality of reference and thus make good use of data caching.

Big Data: Storage organisation matters

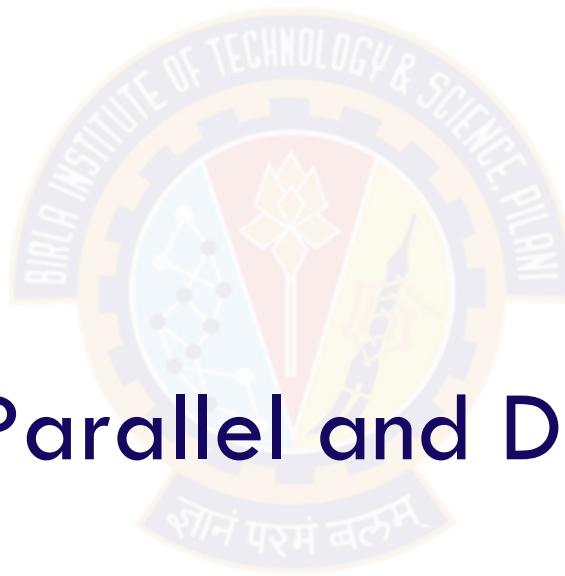
Example

- 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

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

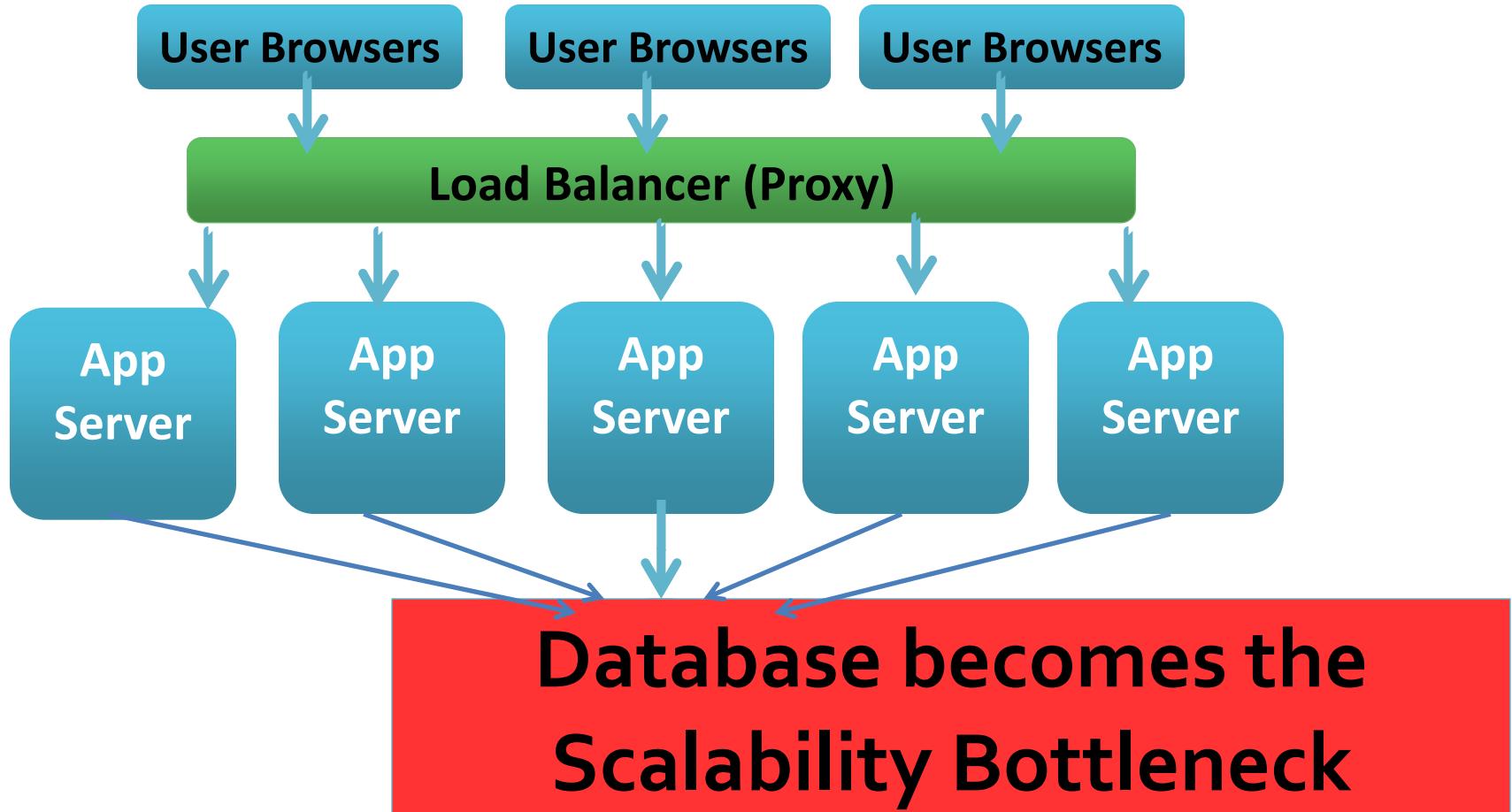
- Most enterprise solutions have RDBMS back-end
- Very little change to RDBMS between 1980 and 2000
- Many application servers, one database
 - Response slows down when DB is concurrently accessed by applications and Analytics tools
 - Response of SQL queries depend on size of data base
 - Easy to parallelize 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



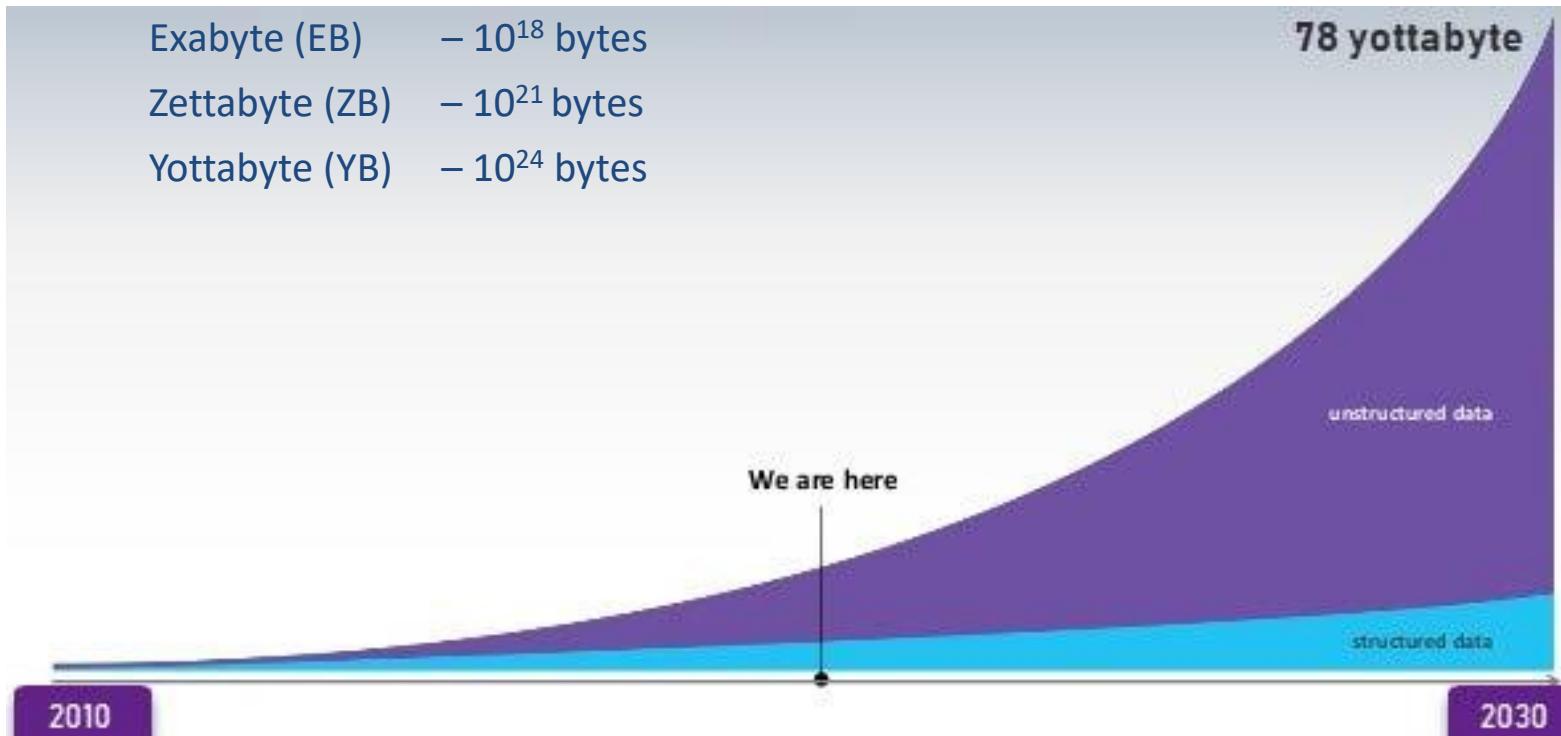
Changes in Data used in Web applications

.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

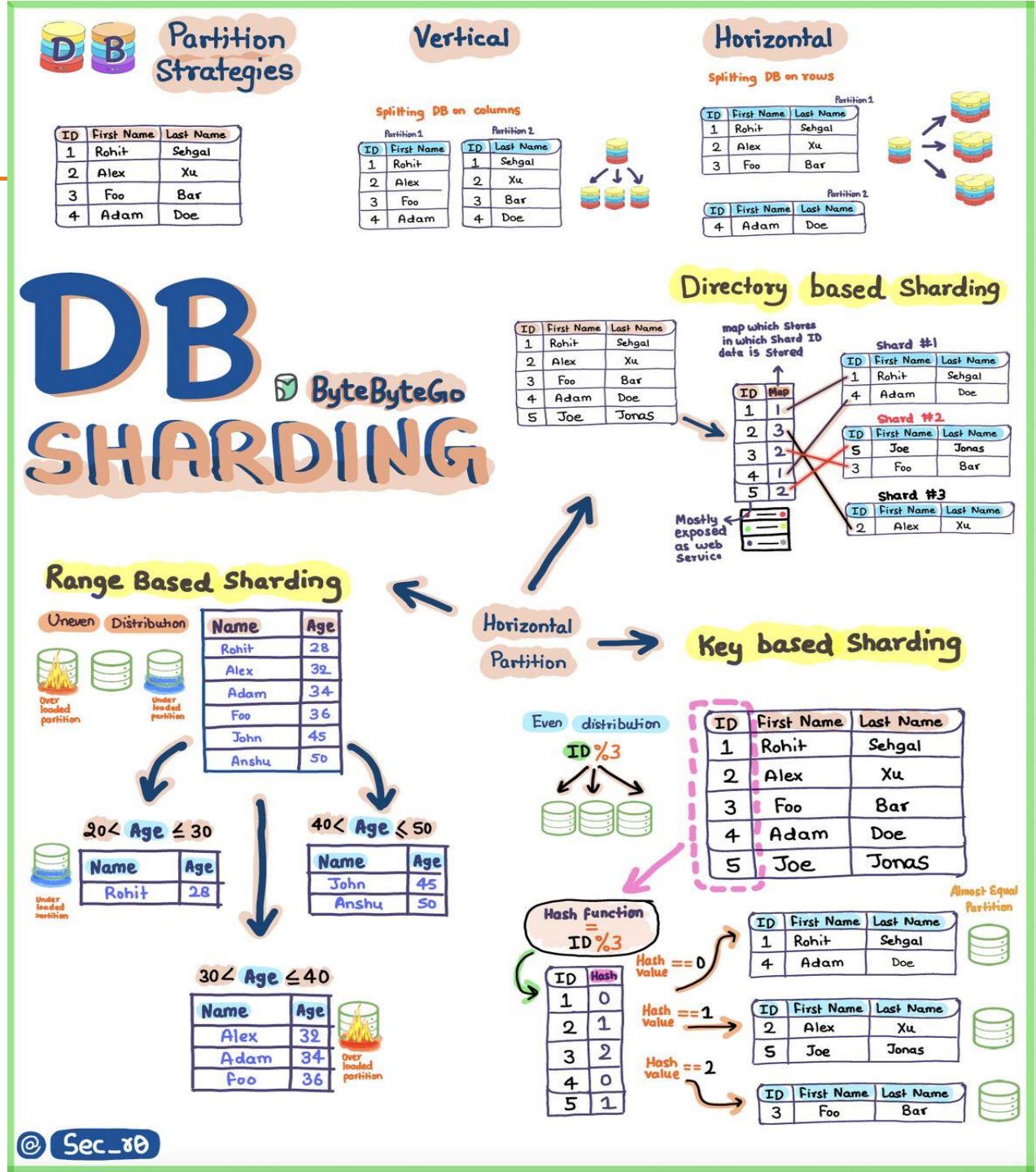
Scaling out RDBMS – Master/Slave

- 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



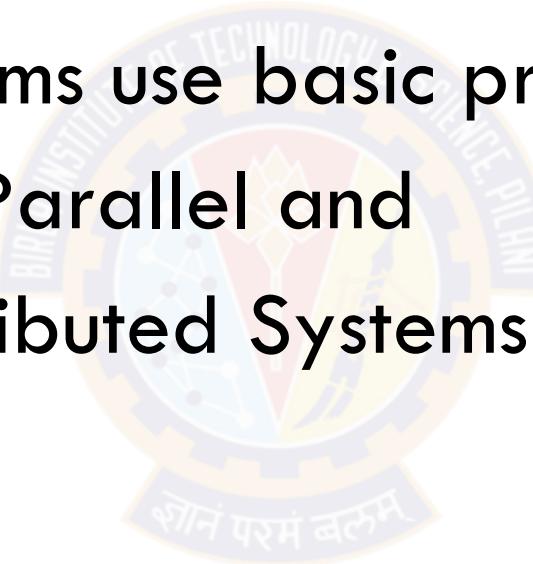
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 2: Parallel and Distributed Systems

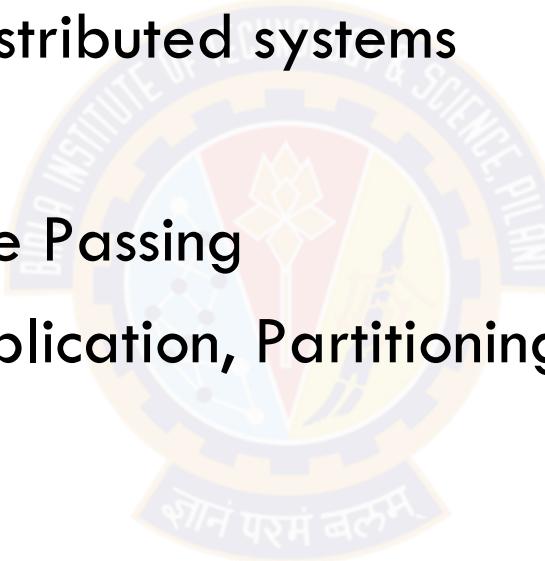
Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

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)

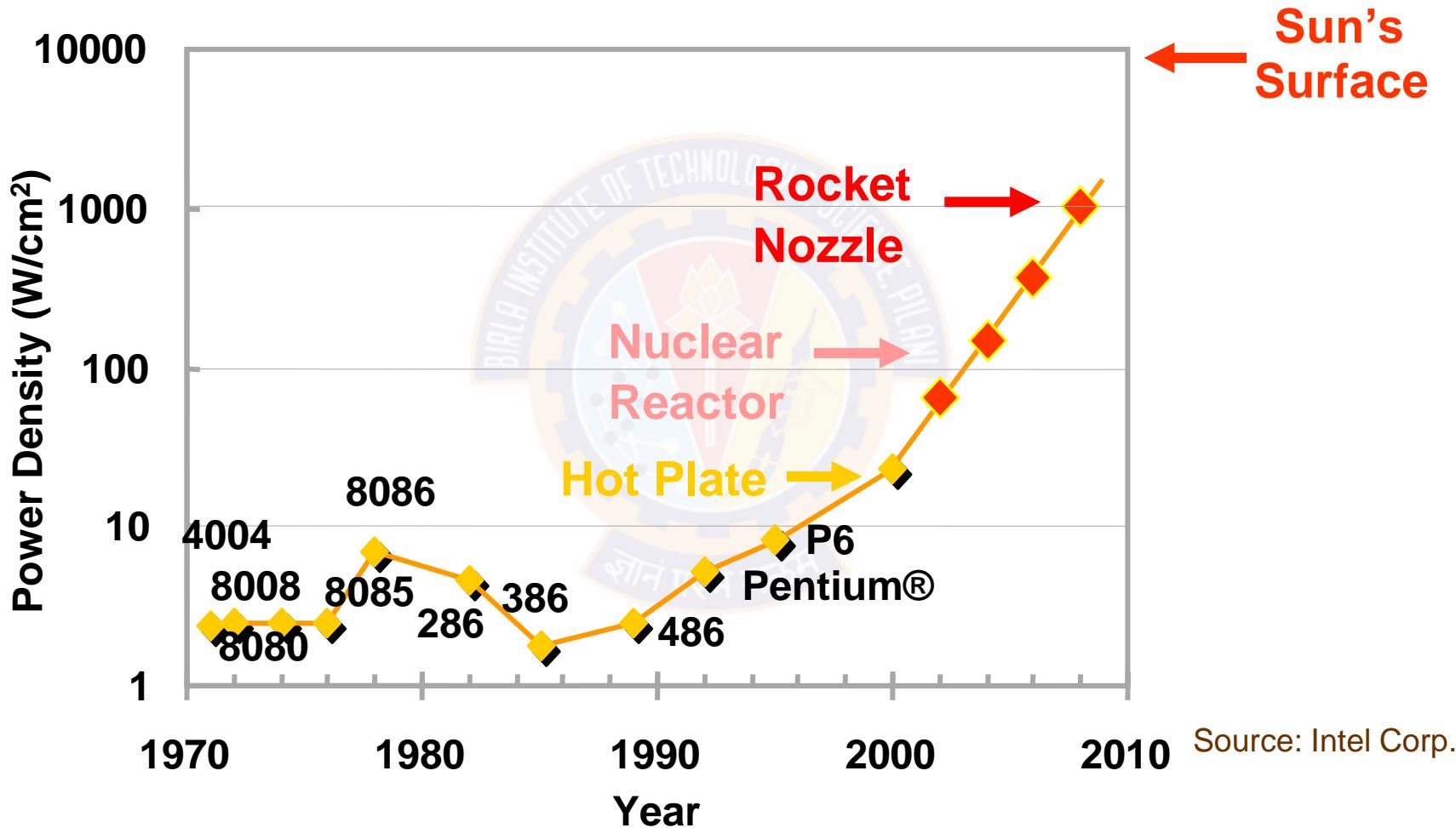
Why we do not have a 10GHz processor now?

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

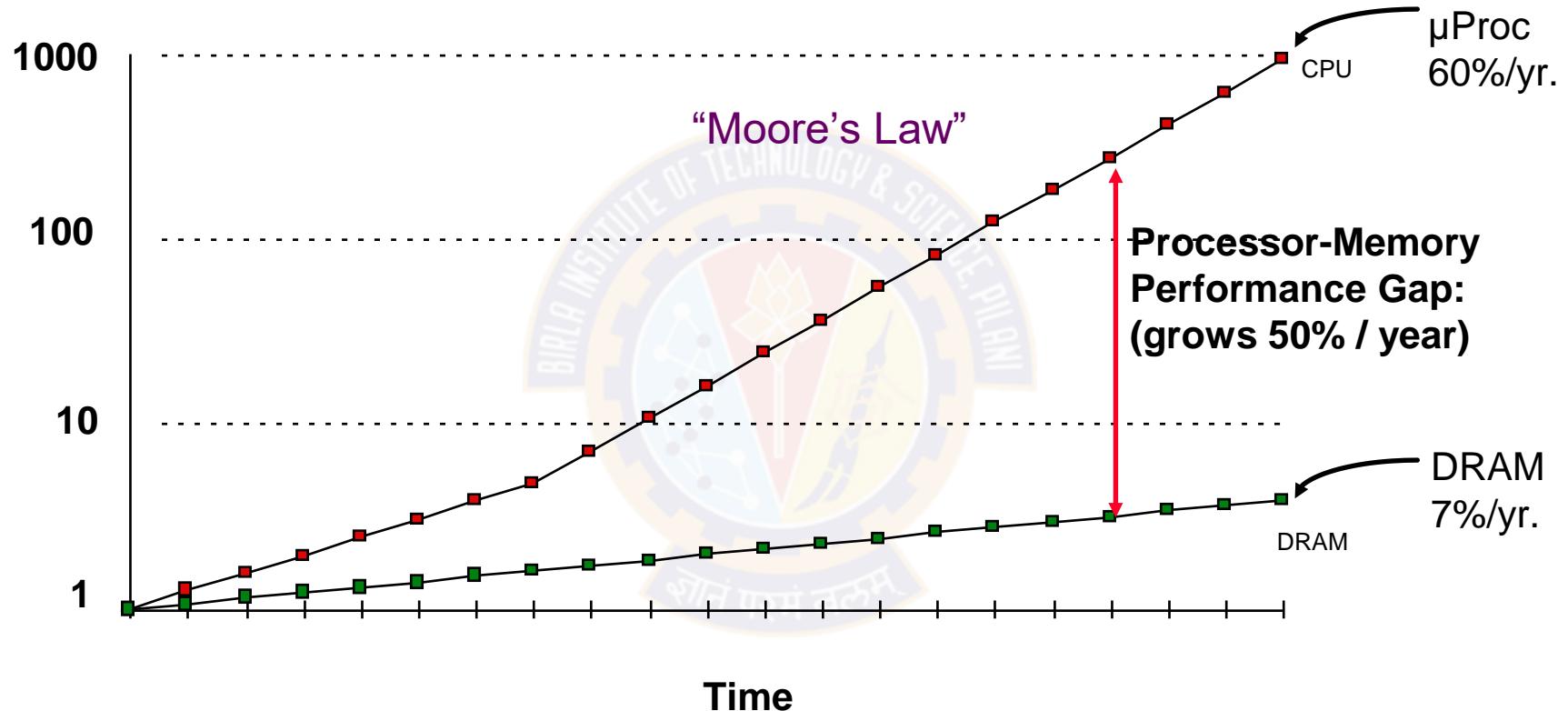
Revolutions in computing:

- 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)

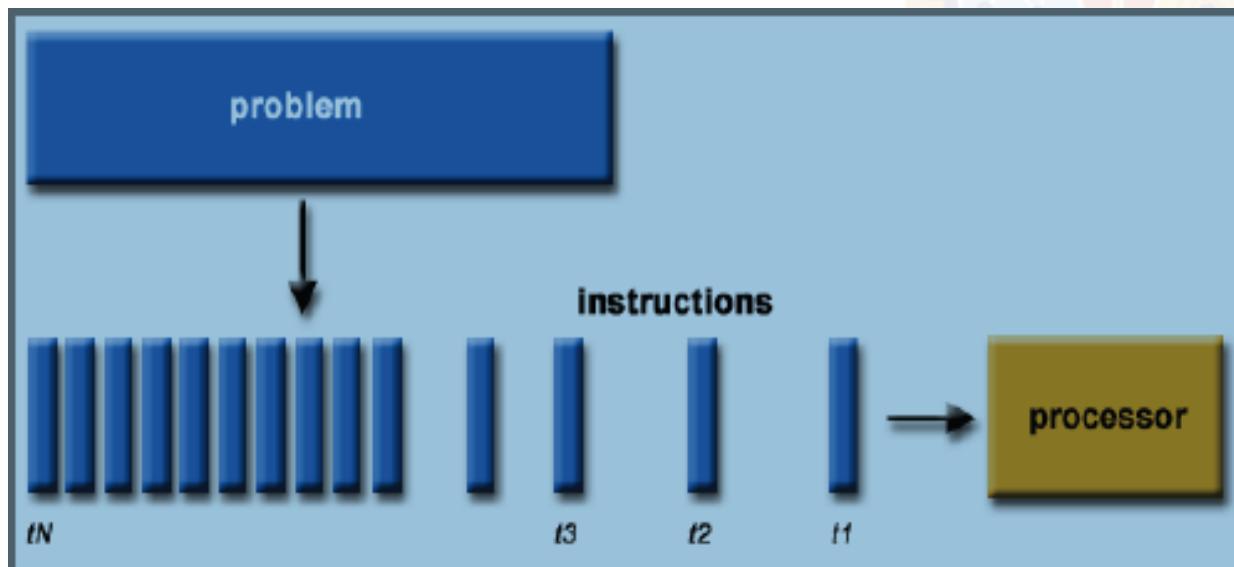


Memory Wall



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

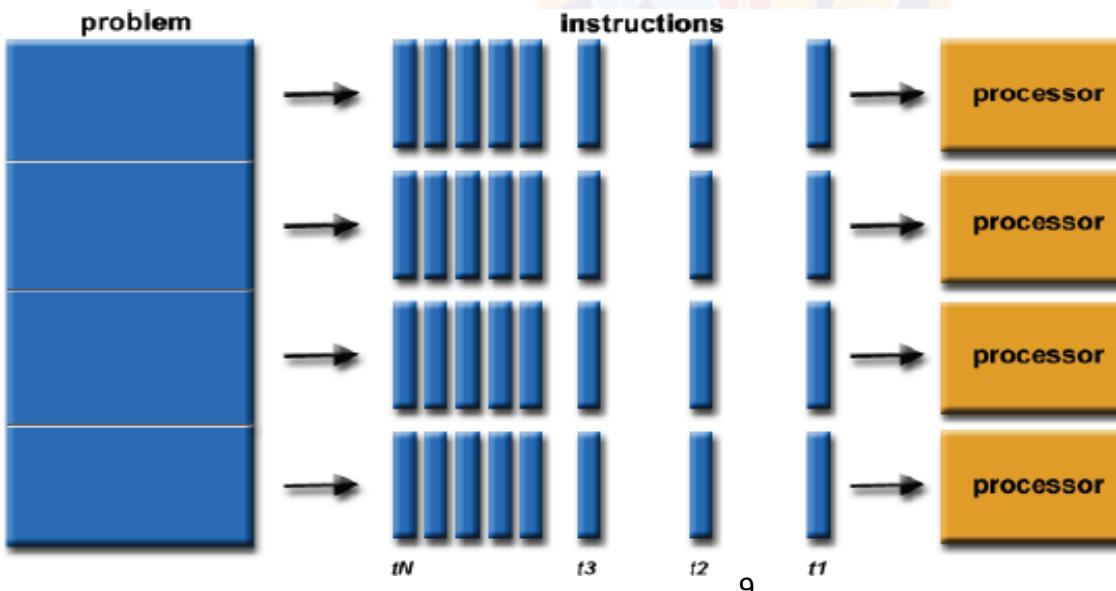


Extra info:

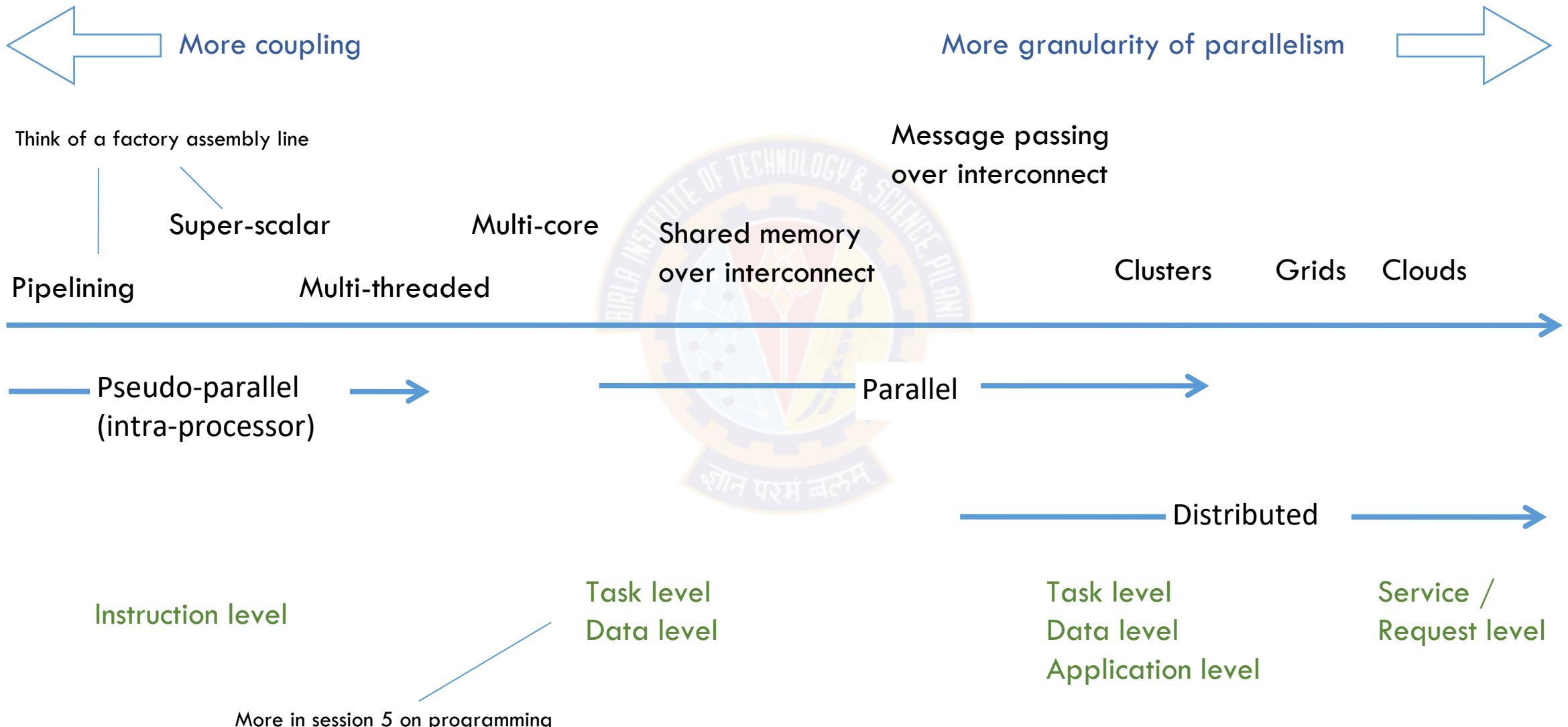
- Von Neumann architecture : common memory store and pathways between instructions and data - causes Von Neumann bottleneck
- Harvard architecture separates them to reduce bottleneck.
- Modern architectures use separate caches for instruction and data.

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

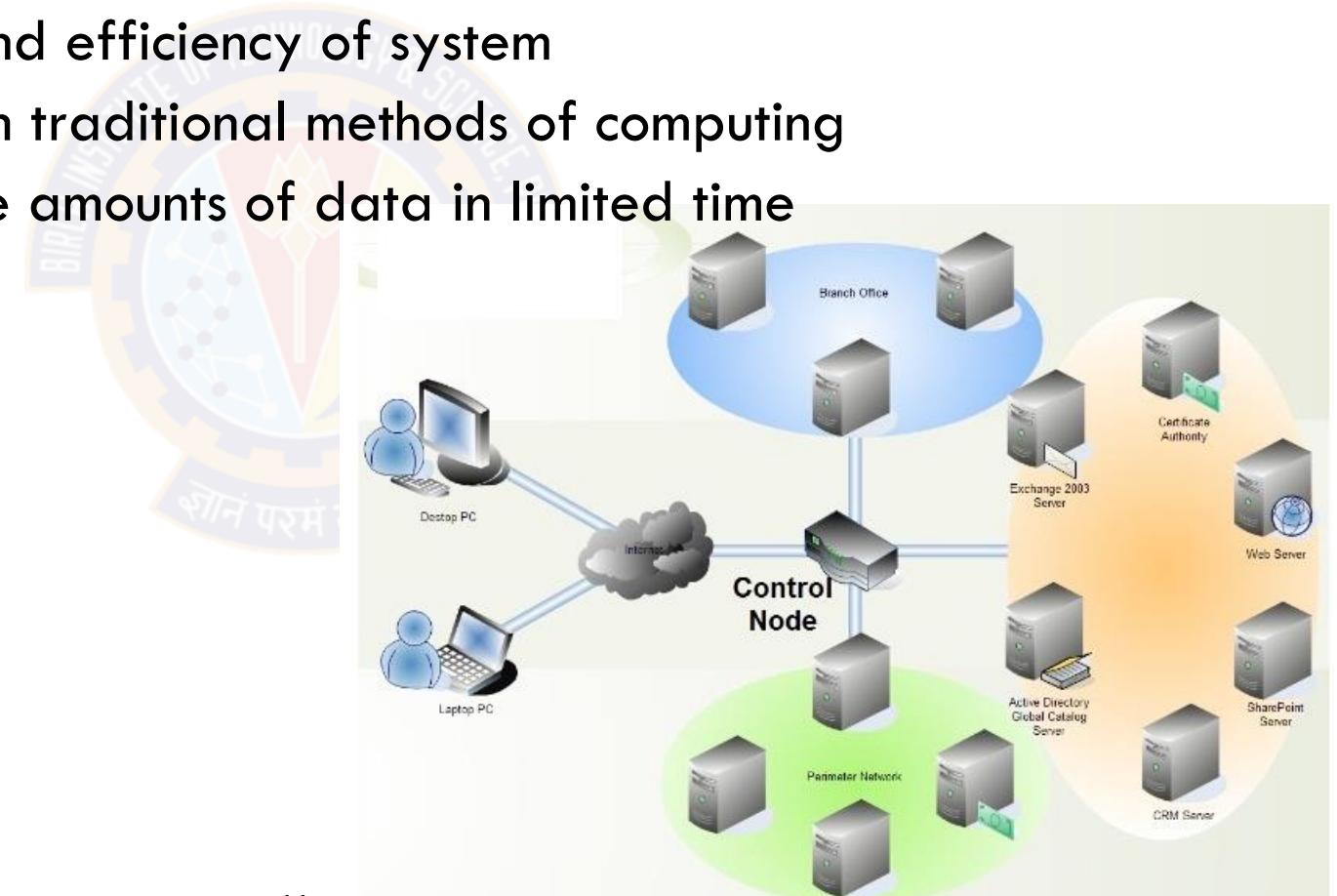


Spectrum of Parallelism



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



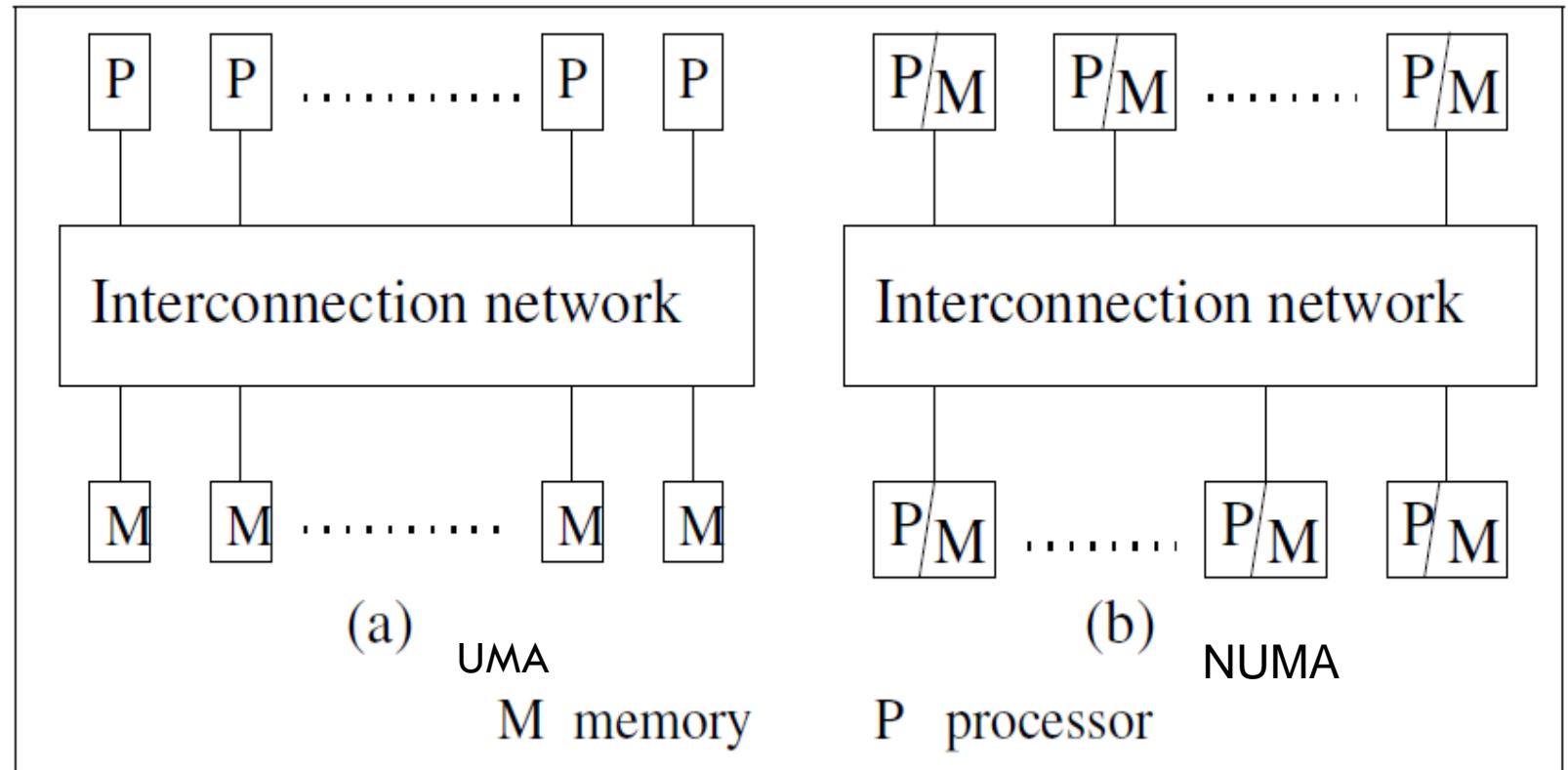
Multi-processor Vs Multi-computer systems

UMA

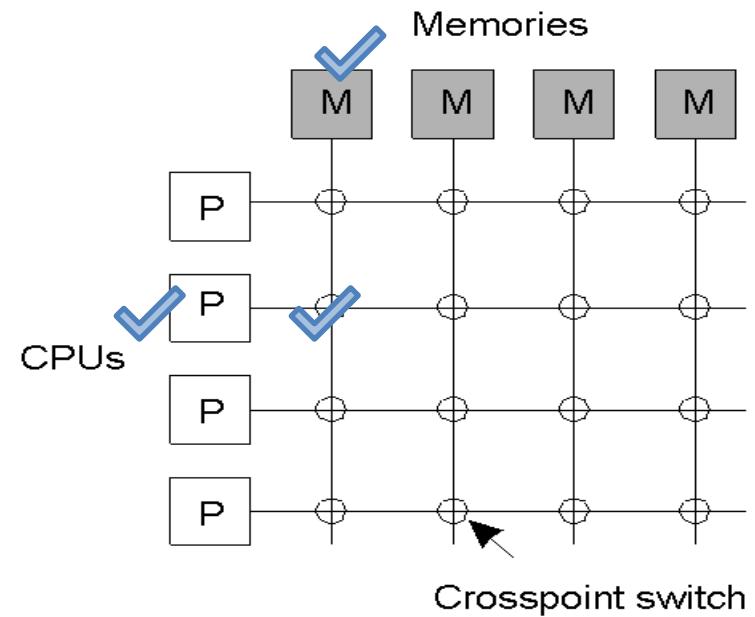
- » Uniform Memory Access Multiprocessor
- » Shared memory address space
- » No common clock
- » Fast interconnect

NUMA

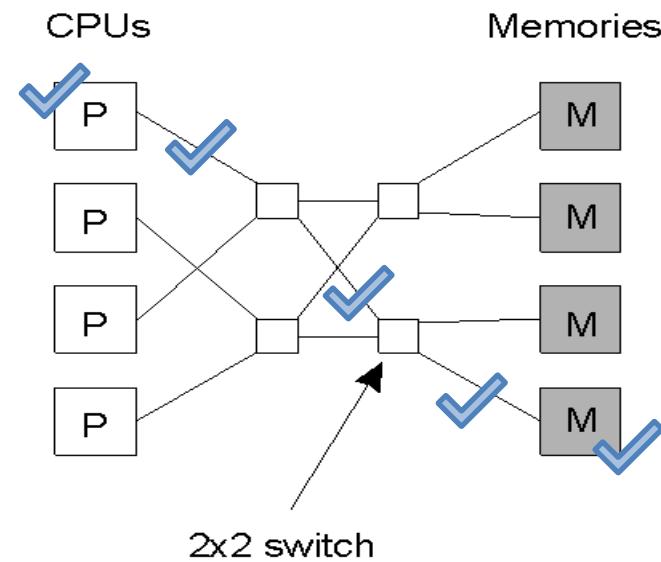
- » Non Uniform Memory Access Multicomputer
- » May have shared address spaces
- » Typically message passing
- » No common clock



Interconnection Networks



(a)



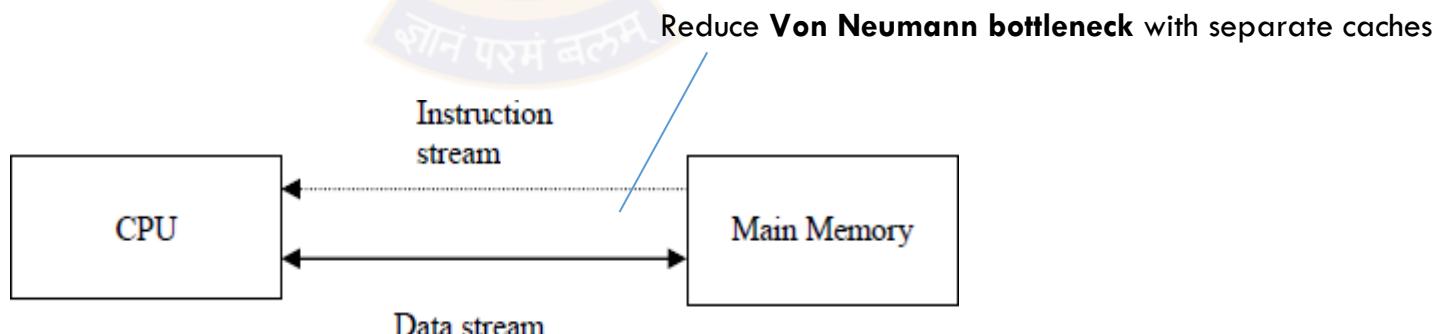
(b)

- 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

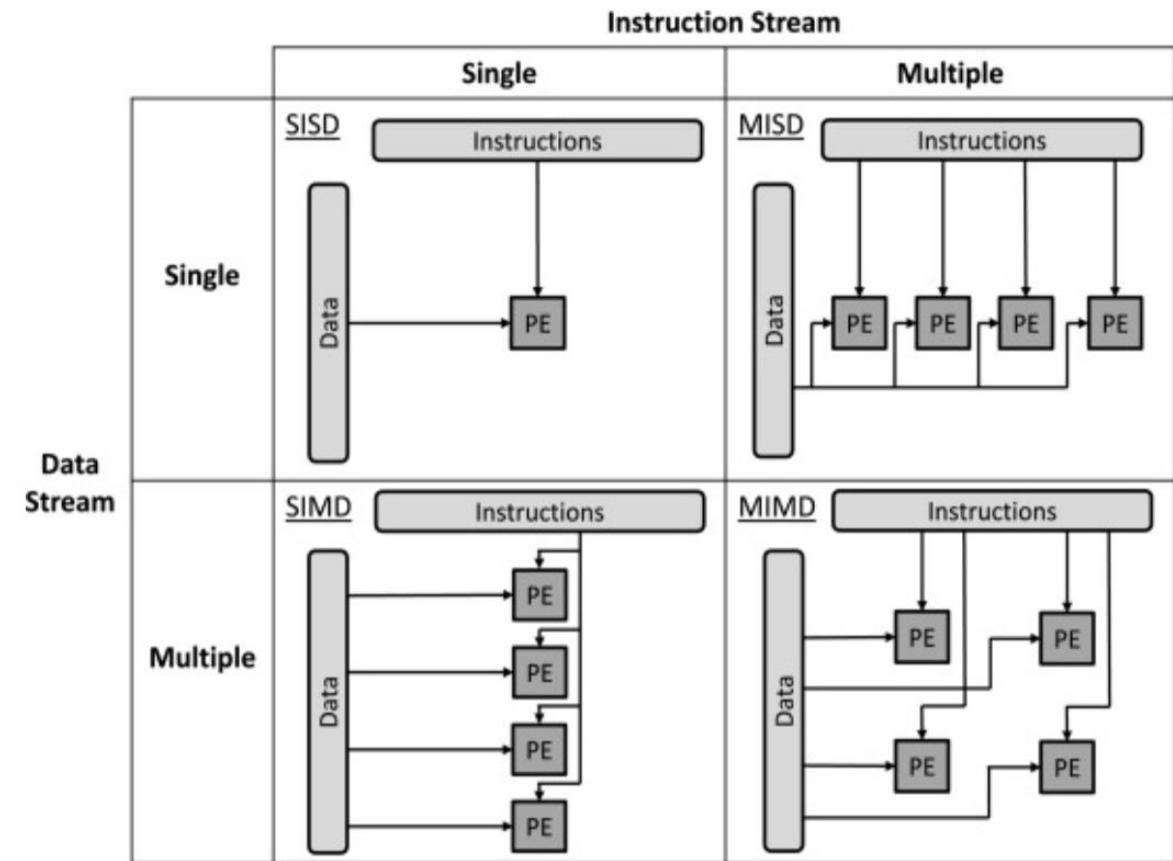
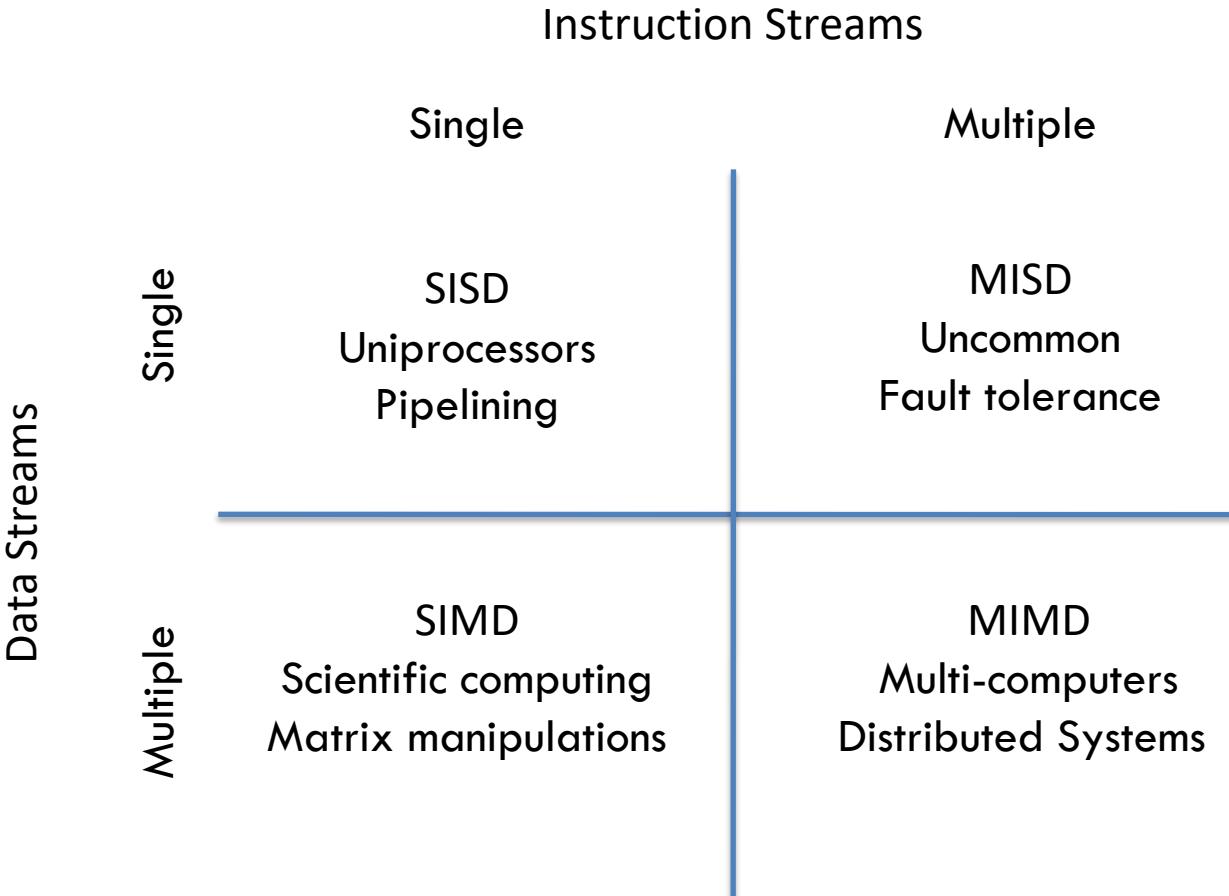


Image from [sciencedirect.com](https://www.sciencedirect.com)

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

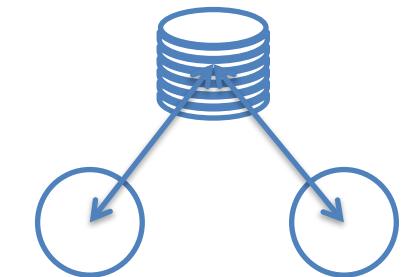
Parallel System	Distributed System
Computer system with several processing units attached to it	Independent, autonomous systems connected in a network accomplishing specific tasks
A common shared memory can be directly accessed by every processing unit in a network	Coordination is possible between connected 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



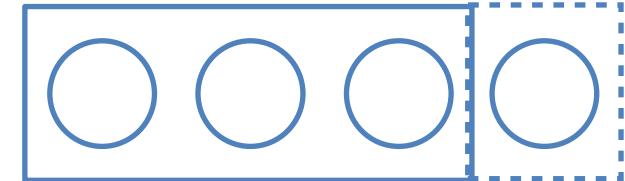
remote shared resource

Motivation for parallel / distributed systems (2)

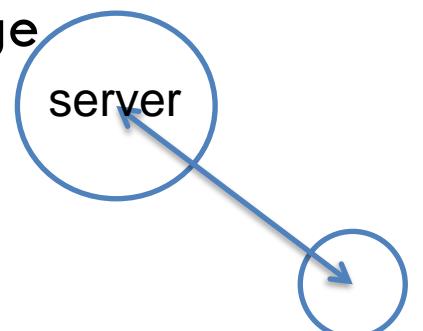
- Better reliability because less chance of multiple failures
 - Be careful about Integrity : Consistent state of a resource across concurrent access
- Incremental scalability
 - Add more nodes in a cluster to scale up
 - e.g. Clusters in Cloud services, autoscaling in AWS
- Offload computing closer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



cluster nodes



resize cluster

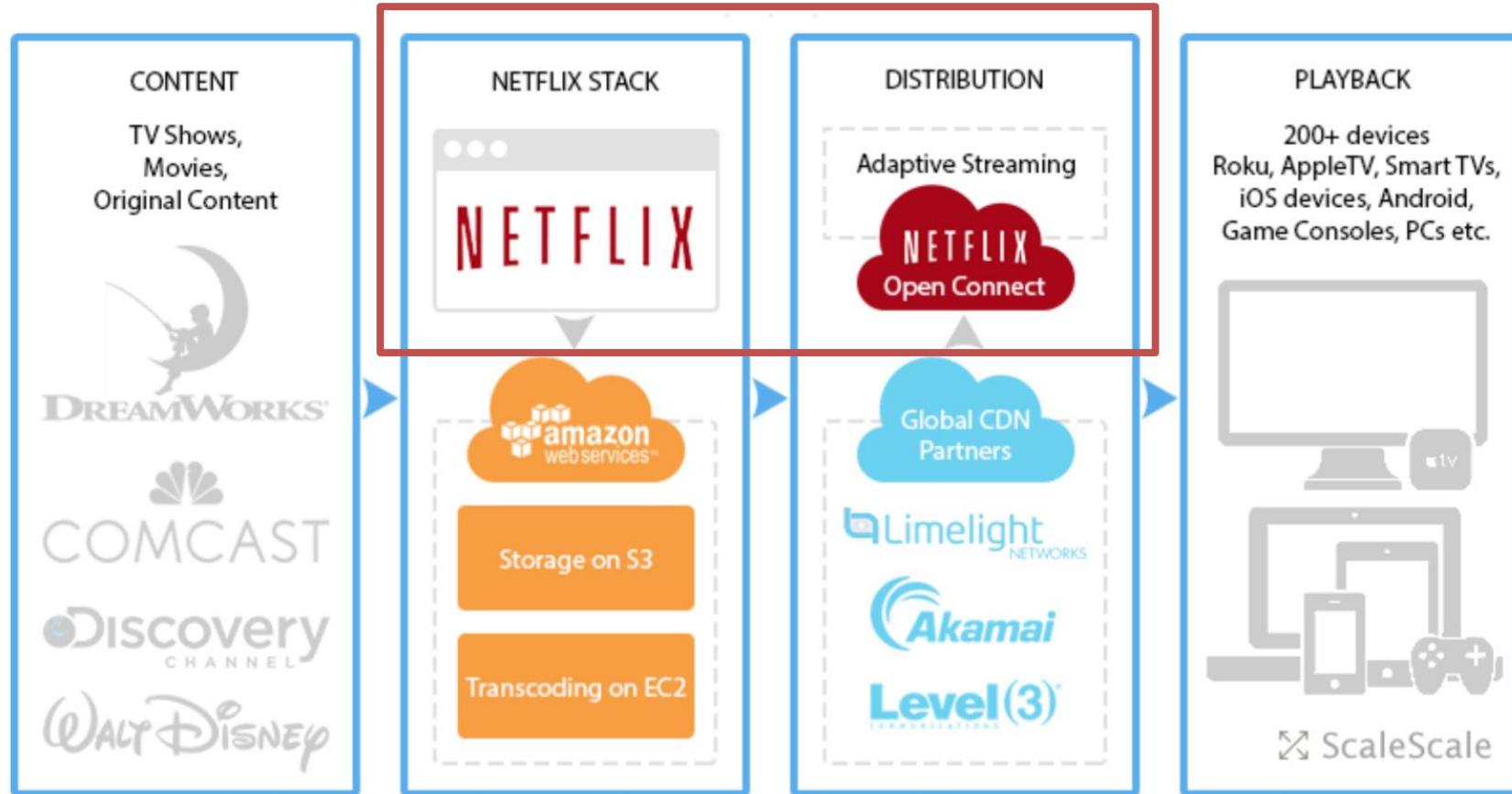


offload some error handling to edge

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

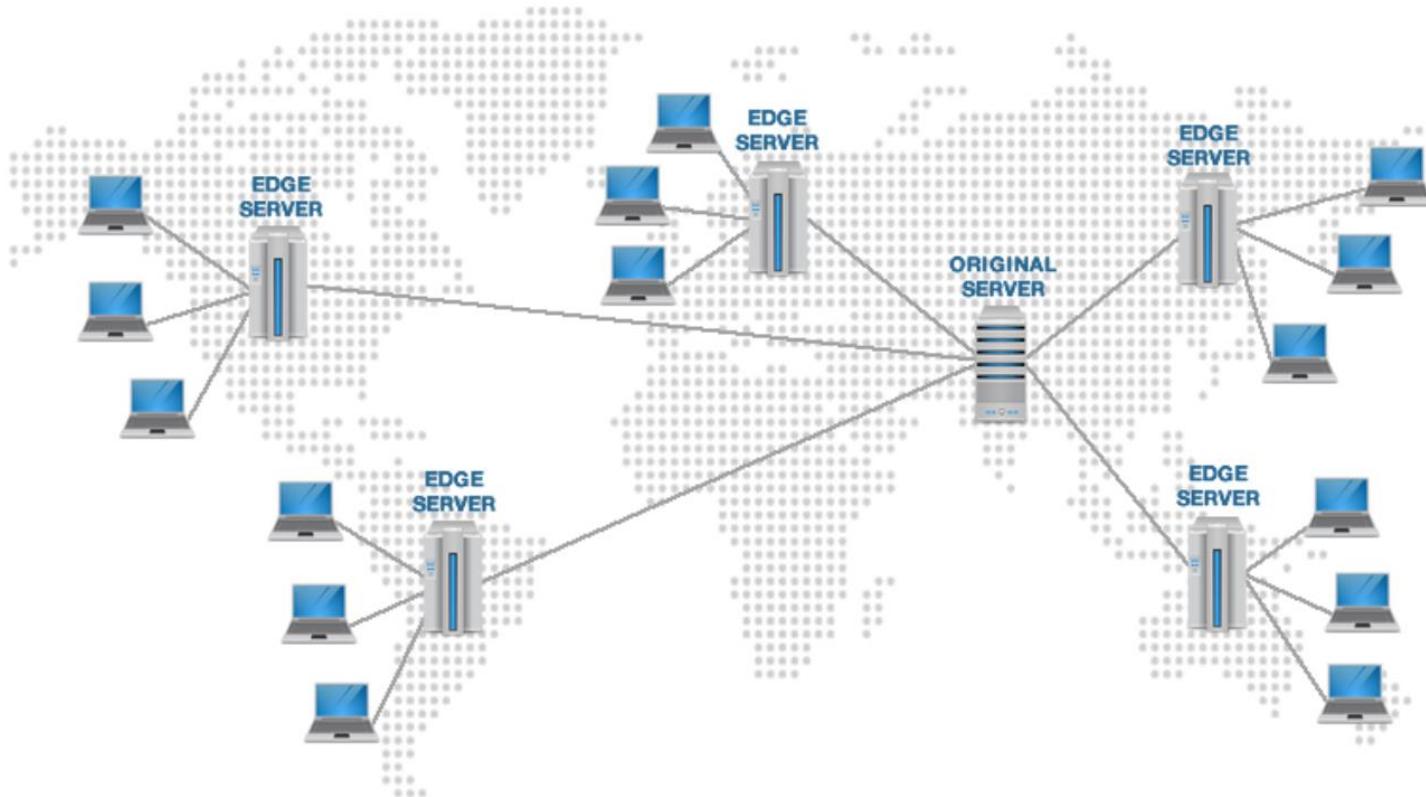
Example: Netflix

~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

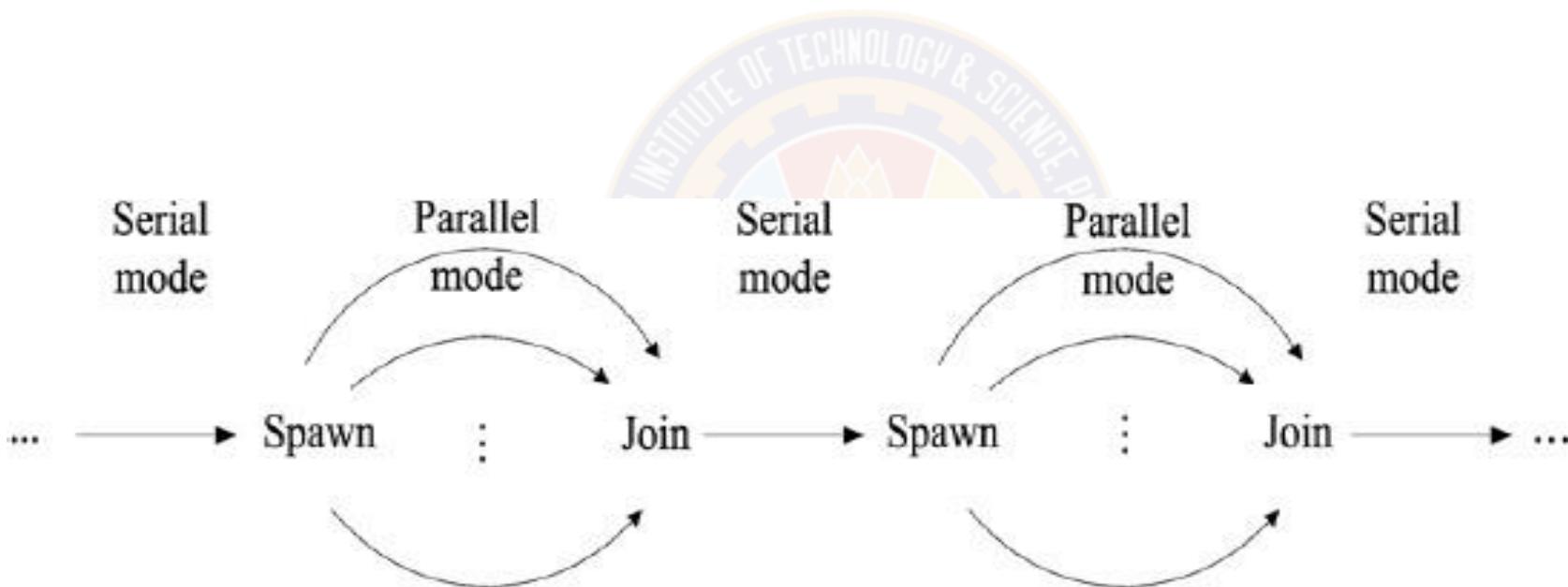
Topics for today

- 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)

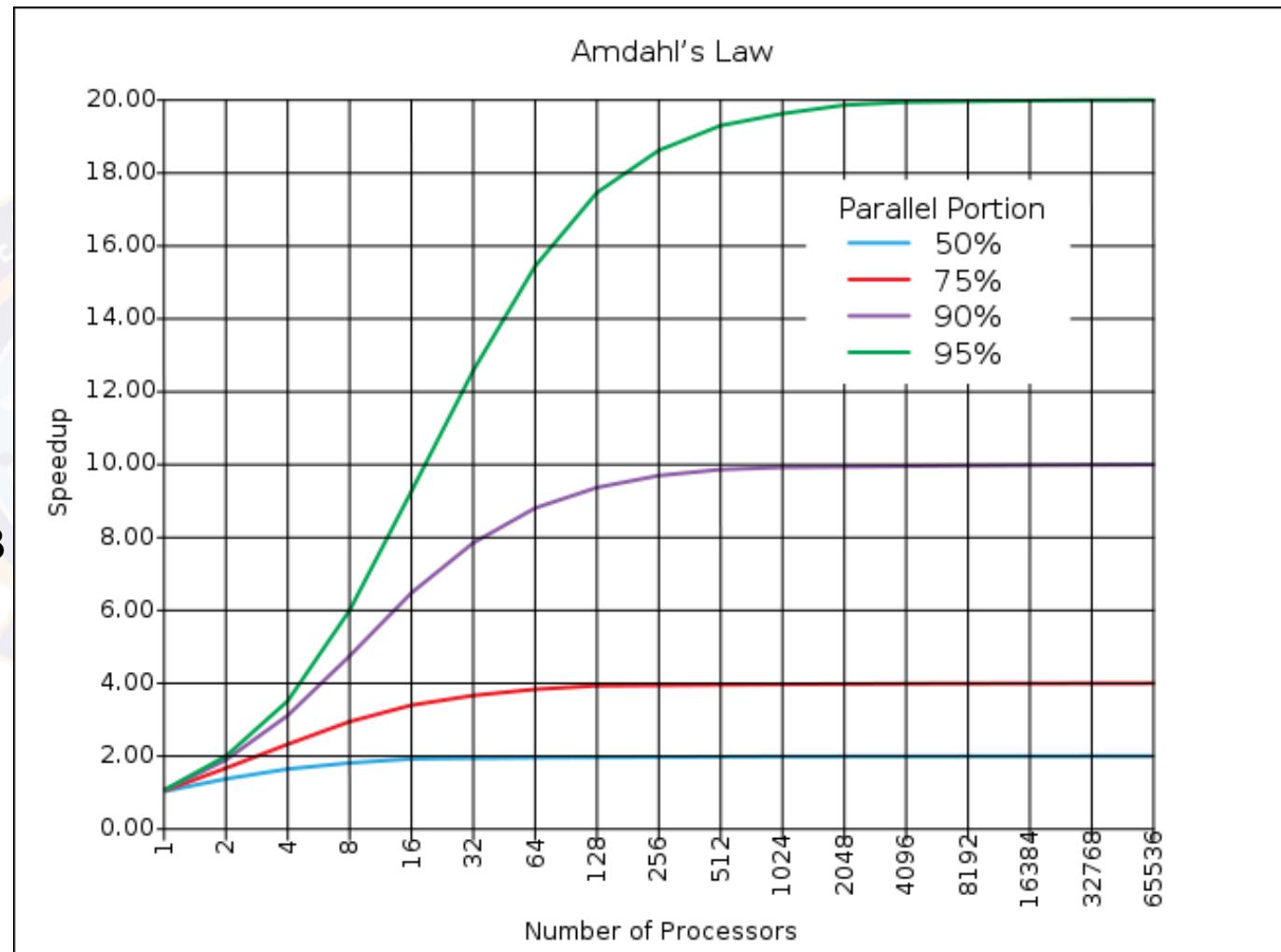
P = 0; Completely sequential – Speedup = 1

P = 1; Completely parallel – Speedup = N

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

2007

**The 40th 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

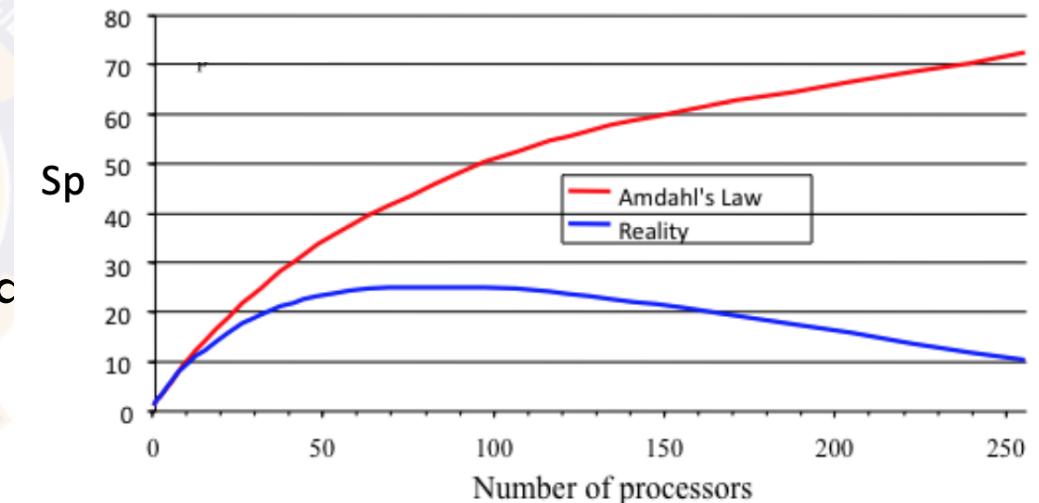
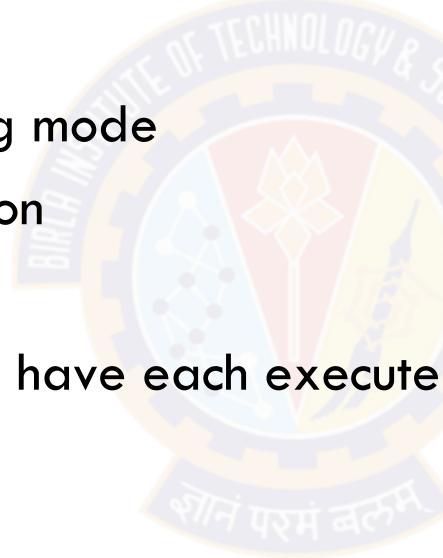
A and B exchange messages in blocking mode

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 ...

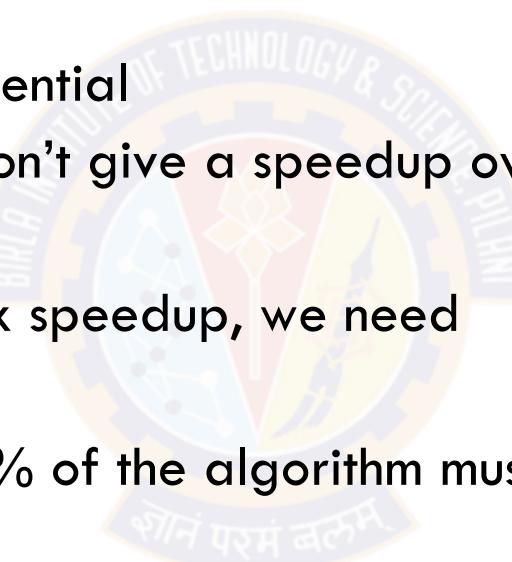


Why Amdahl's Law is such bad news

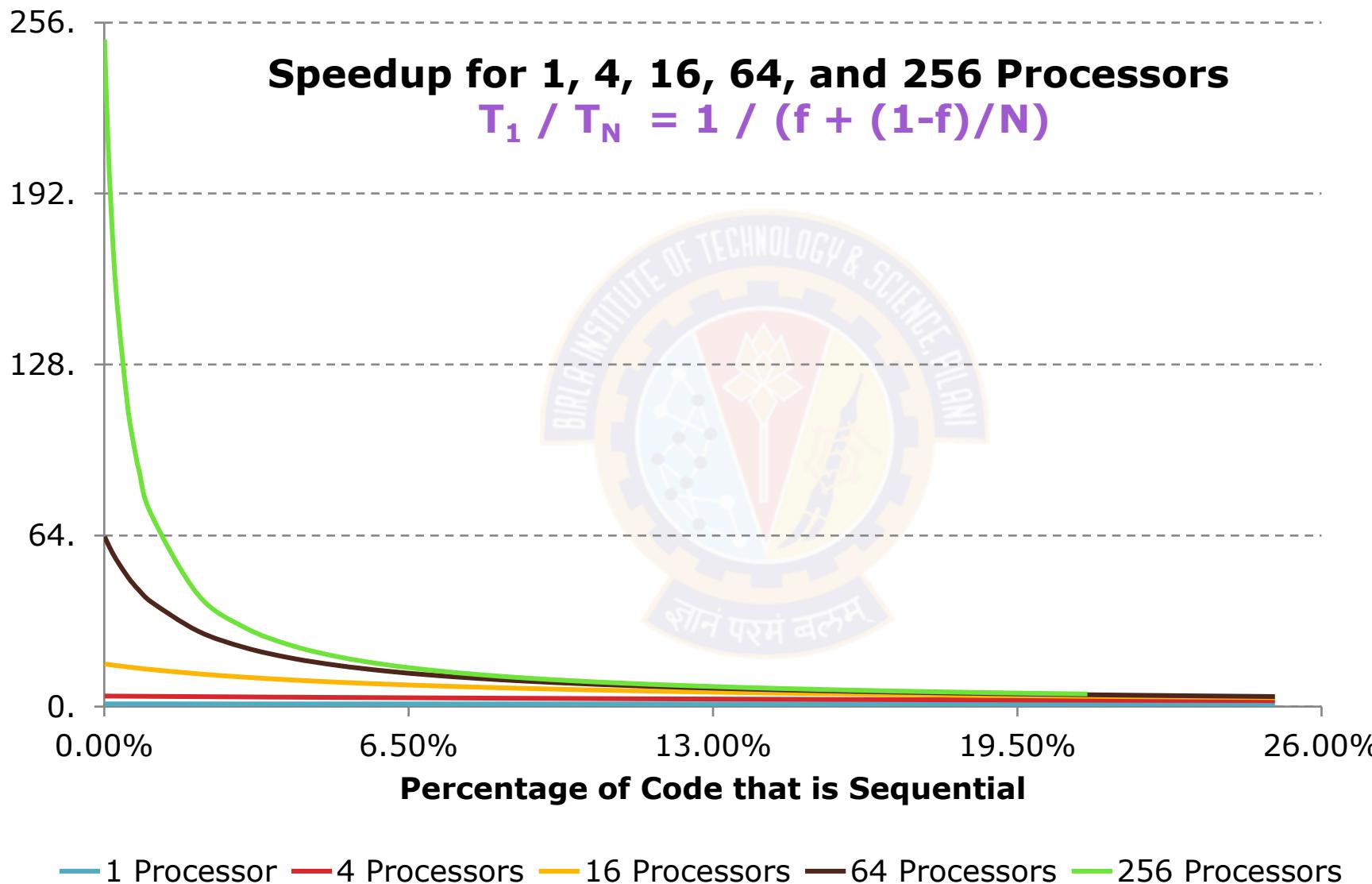
$$S(N) \sim 1/f, \text{ for large } N$$

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 !!

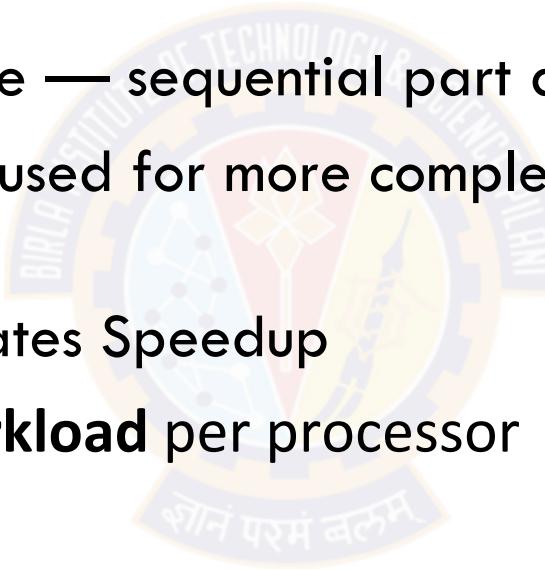


Speedup plot



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

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

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

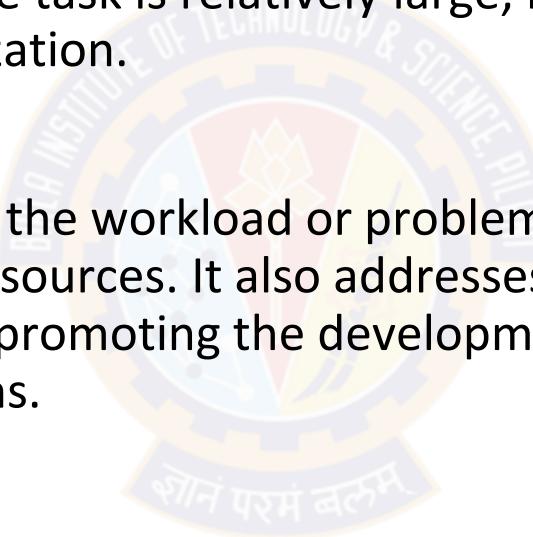
S(N) is not limited by f as N scales

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.



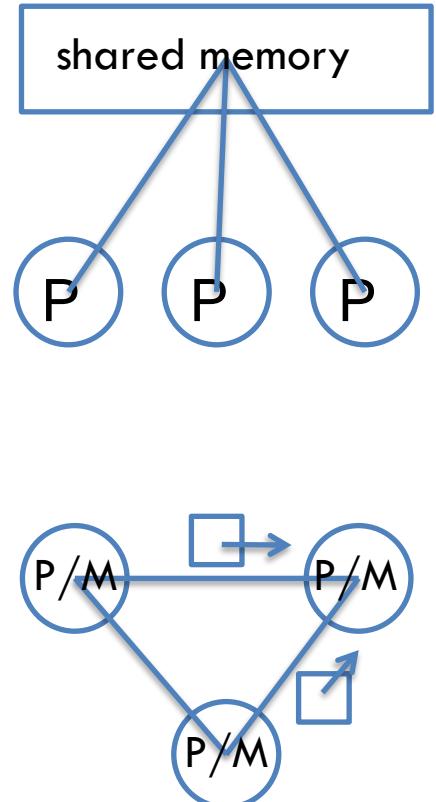
Topics for today

- 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



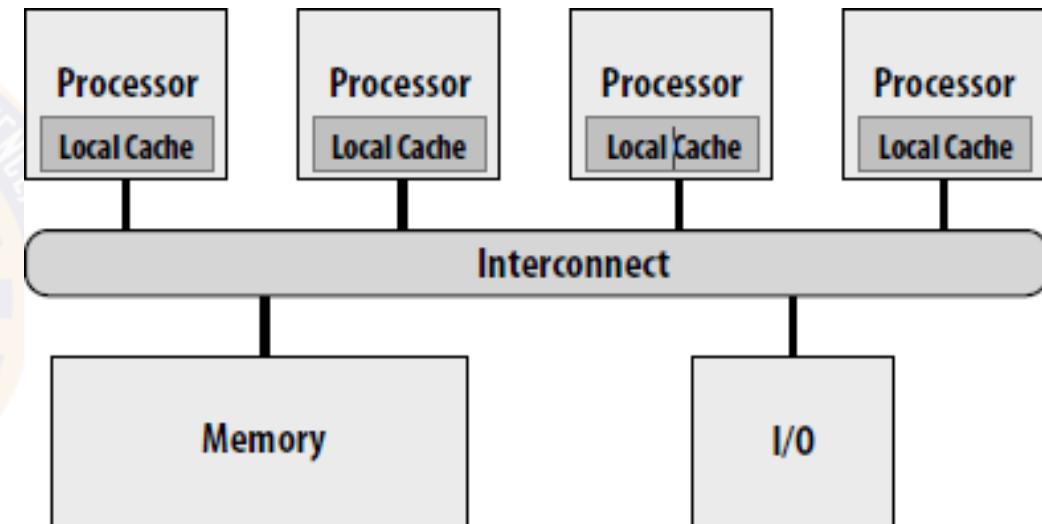
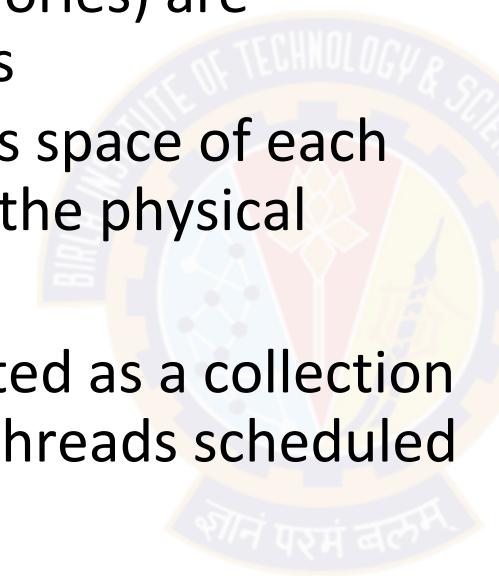
Memory access models

- » 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 ?



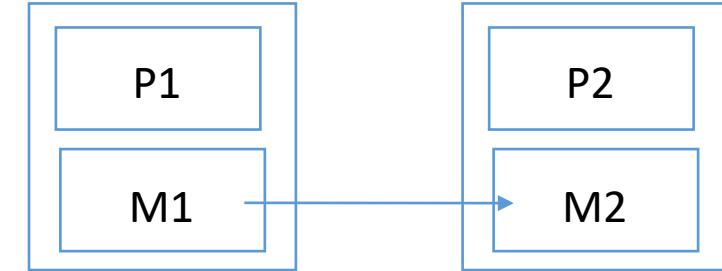
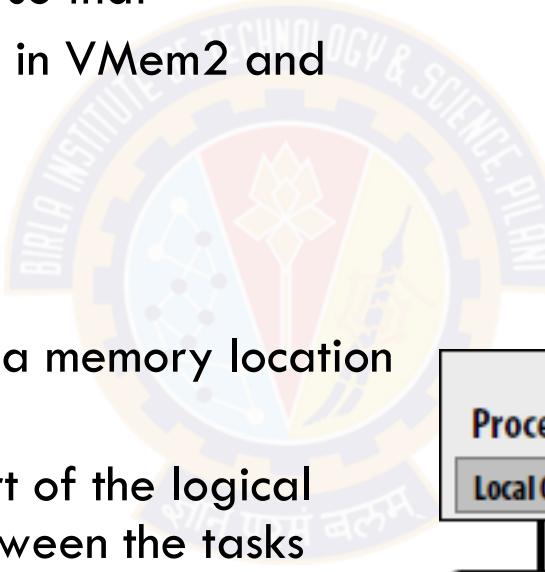
Shared Memory Model: Implications for Architecture

- 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

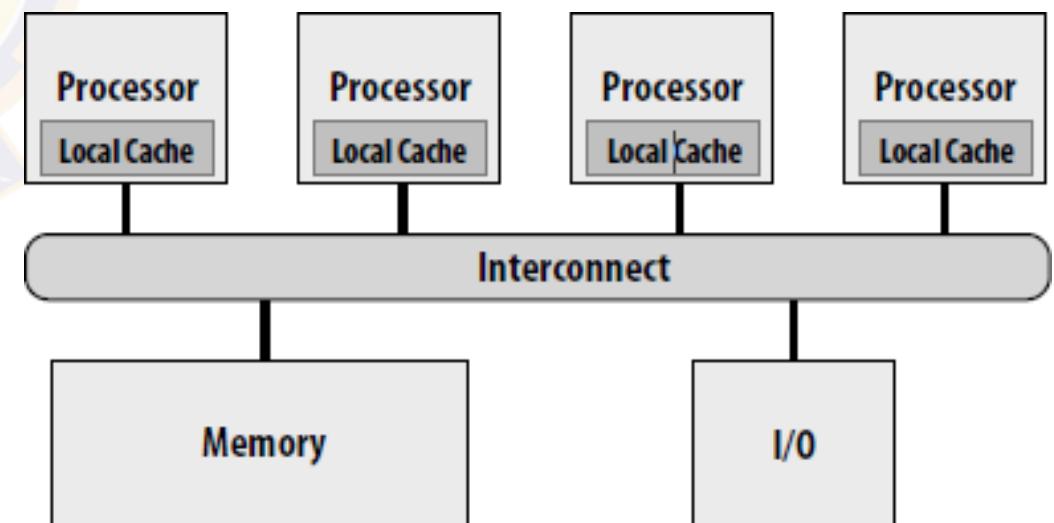


Distributed memory with message passing Vs Shared memory

- 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.

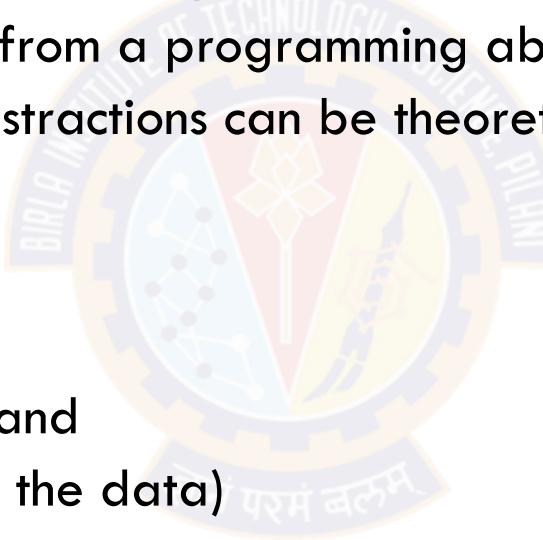


- Whereas in a Shared Memory model
 - ✓ task T1 writes the data item into a memory location and T2 can read the same
 - ✓ how ? the memory location is part of the logical address space that is shared between the tasks



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

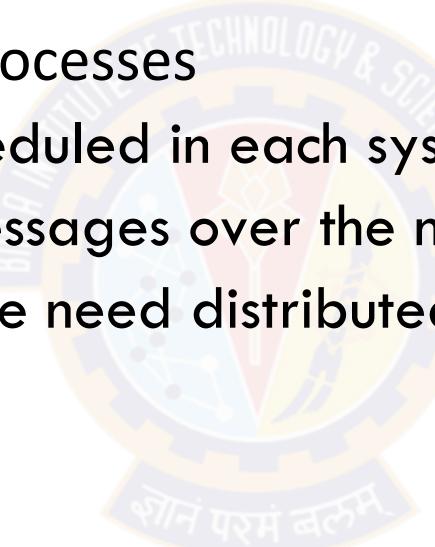
- Processes operate within their own private address spaces
- Processes communicate by sending/receiving messages
 - ✓ send: specifies recipient, buffer to be transmitted, and message identifier (“tag”)
 - ✓ receive: specifies buffer to store data, and optional message identifier
 - ✓ Sending messages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

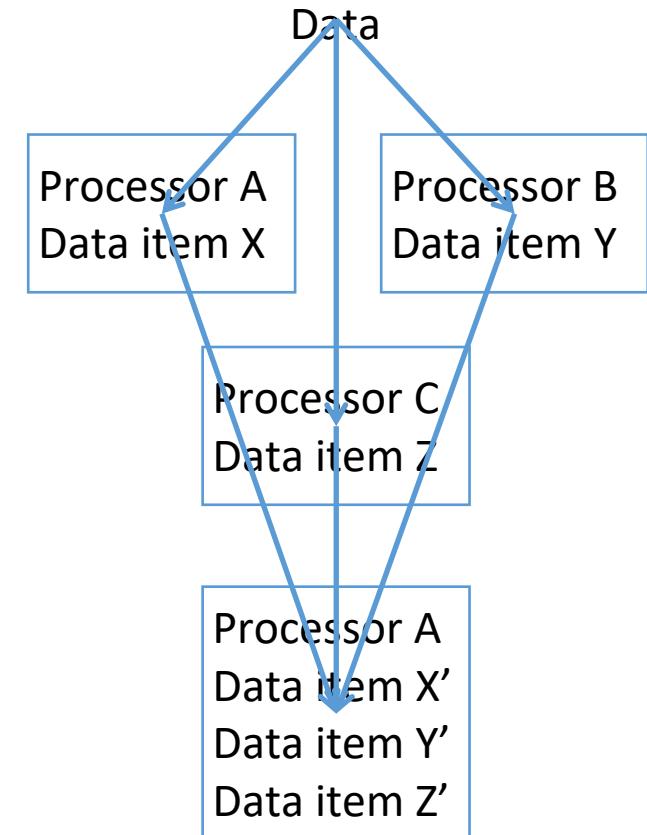
- ✓ A collection of stand-alone systems are connected in a network
- ✓ A task runs as a collection of processes
- ✓ One or more processes are scheduled in each system / node
- ✓ Data exchanges happen via messages over the network
- ✓ Harder for programmers - hence need distributed OS / middleware layer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

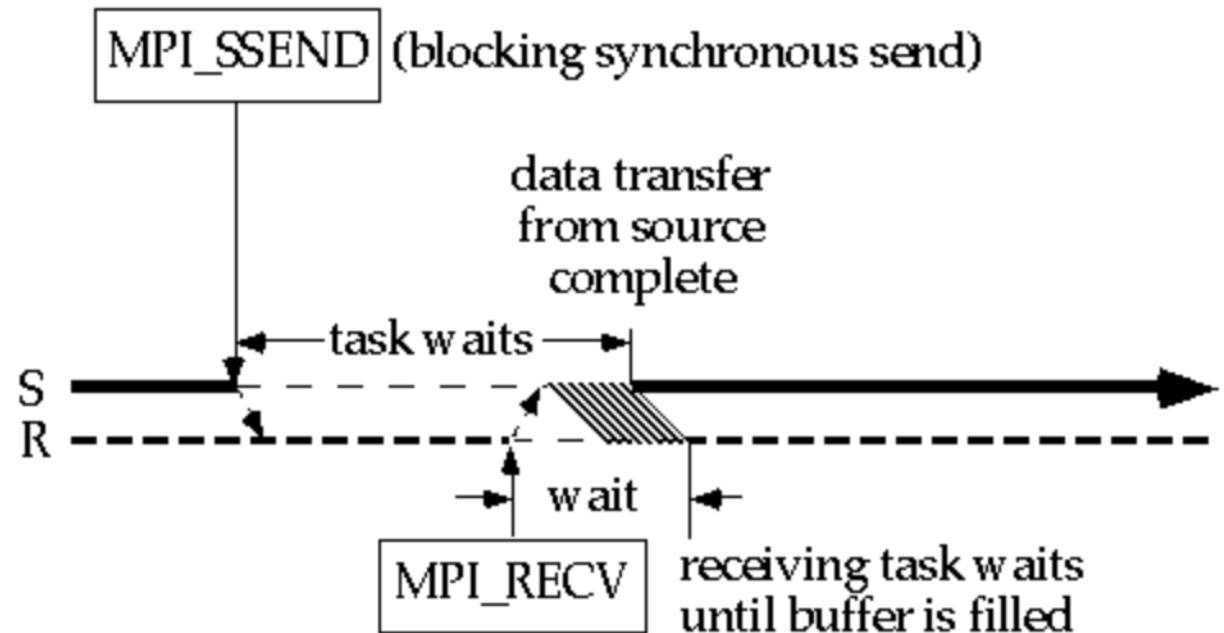


We will see example of Hadoop map-reduce where data is partitioned, outputs communicated over messages and merged to get final answer.

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

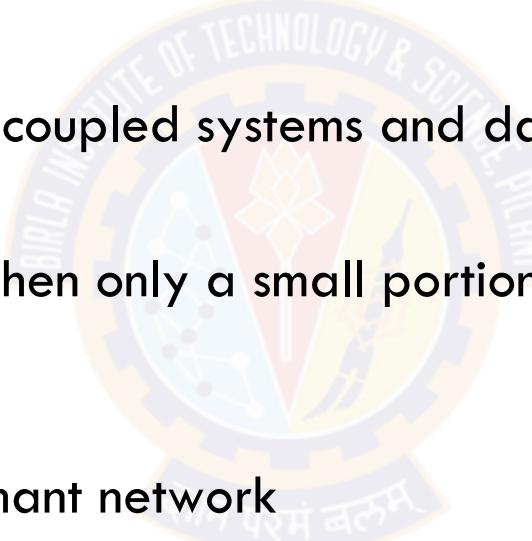
Data Access Strategies: Replication

- **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



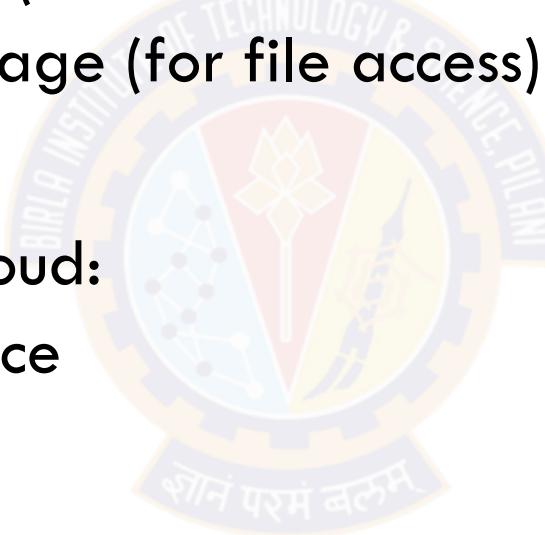
Data Access Strategies: (Dynamic) Communication

- 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

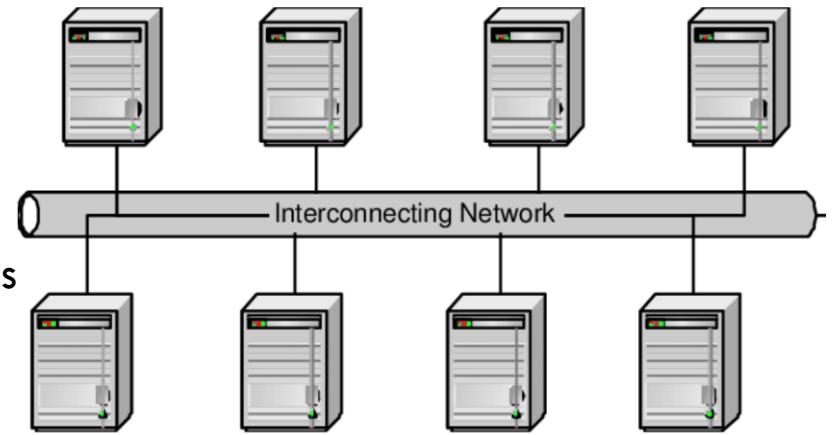
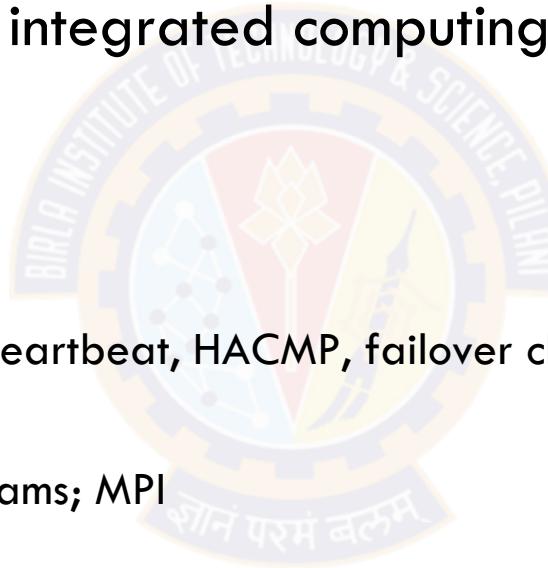
Topics for today

- 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

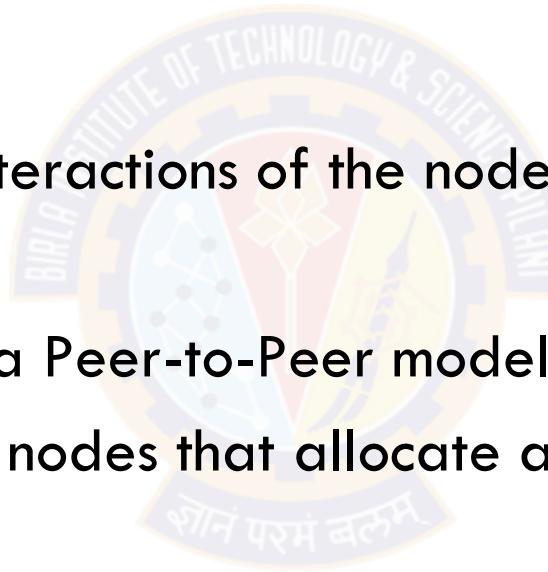
- Question: How is this relevant for Big Data?

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

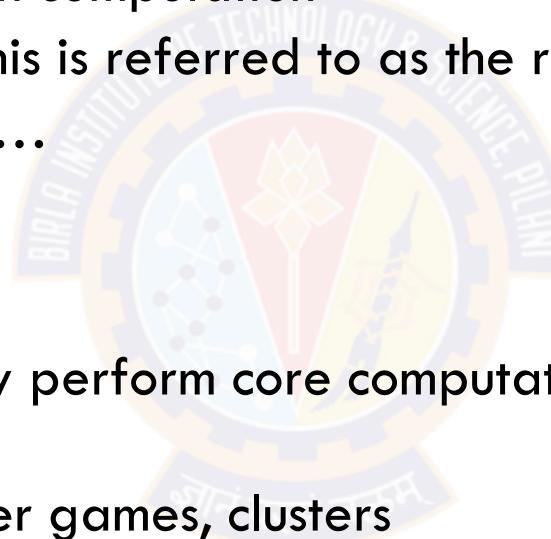
Clusters – Peer to Peer computation

- 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

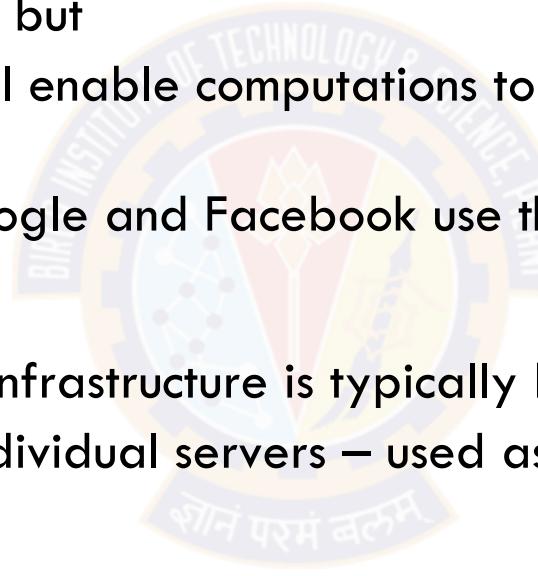
- A cloud uses a datacenter as the infrastructure on top of which services are provided
 - e.g. AWS would have a datacenter in many regions - Mumbai, US east, ... (you can pick where you want your services deployed)
- A cluster is the basic building block for a datacenter:
 - ✓ i.e. a datacenter is structured 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)

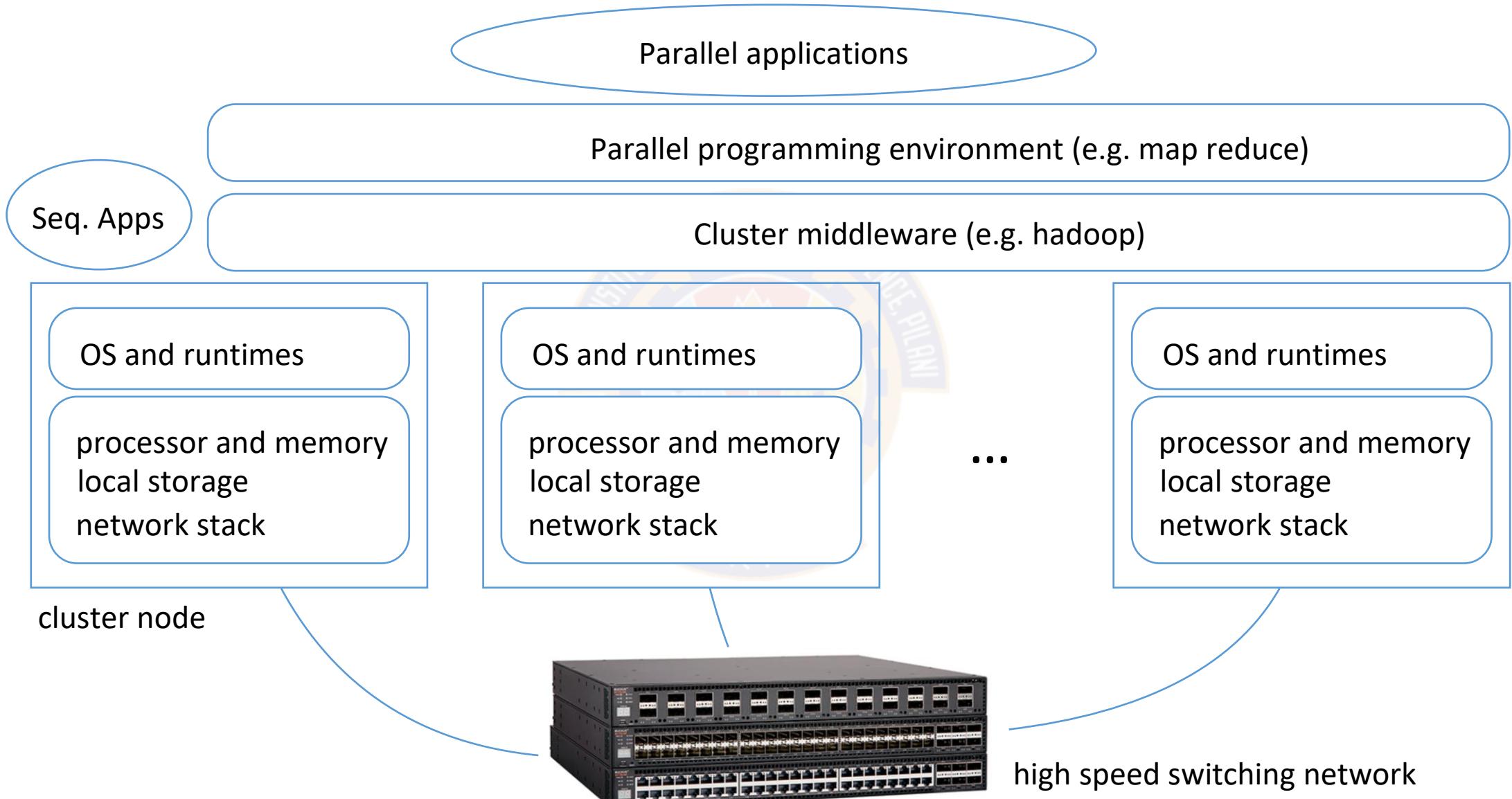
- Rate of obsolescence of computers is high
 - ✓ Even for mainframes and supercomputers
 - ✓ Servers (used for high performance computing) have to be replaced every 3 to 5 years.
- Solution: Build a cluster of commodity workstations
 - ✓ Incrementally add nodes to the cluster to meet increasing workload
 - ✓ 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).

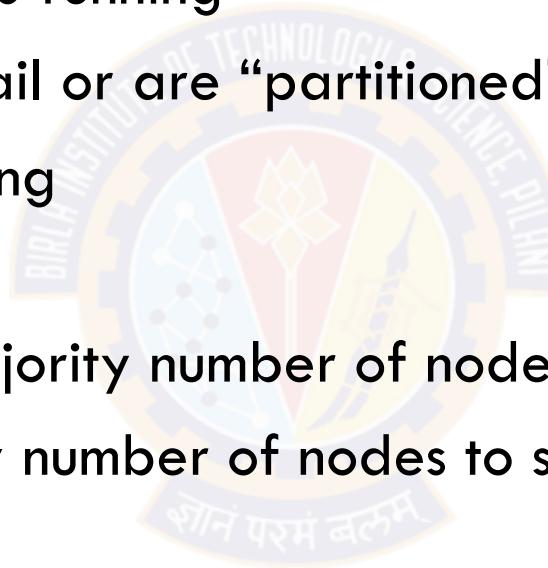


Typical cluster components



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

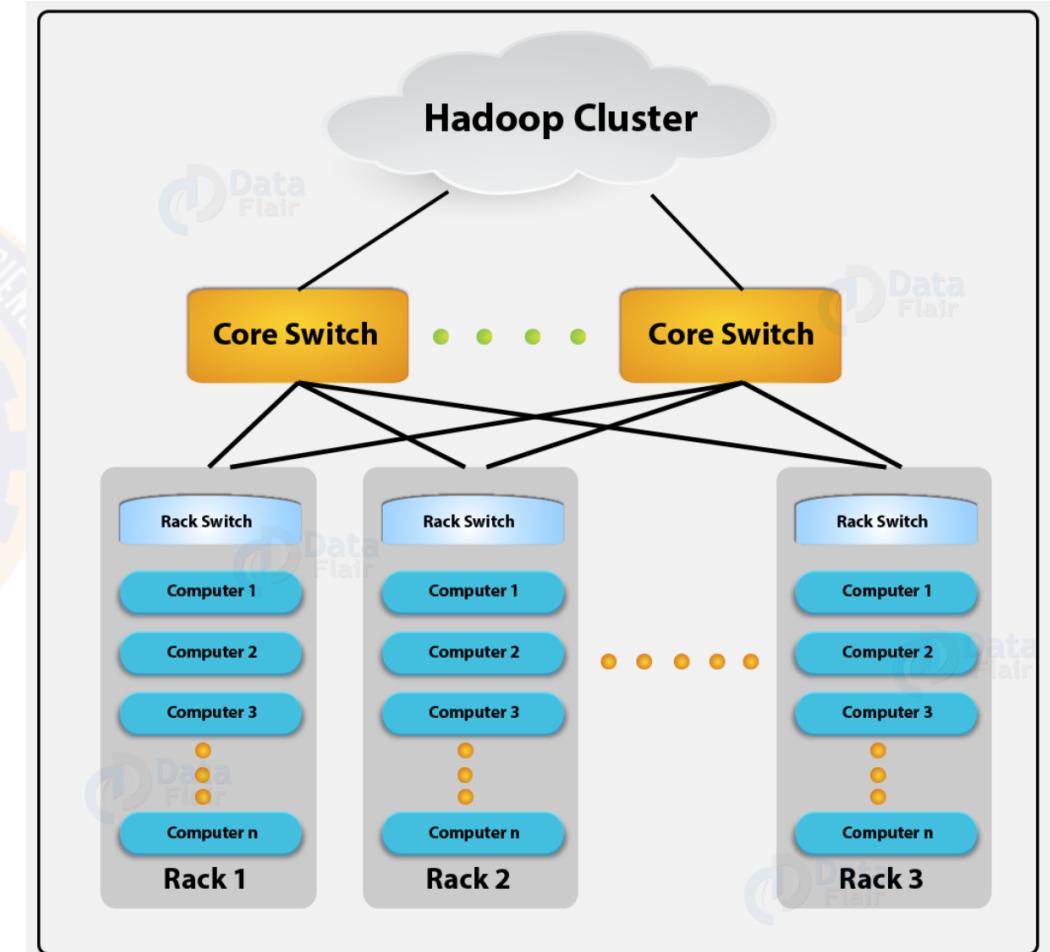
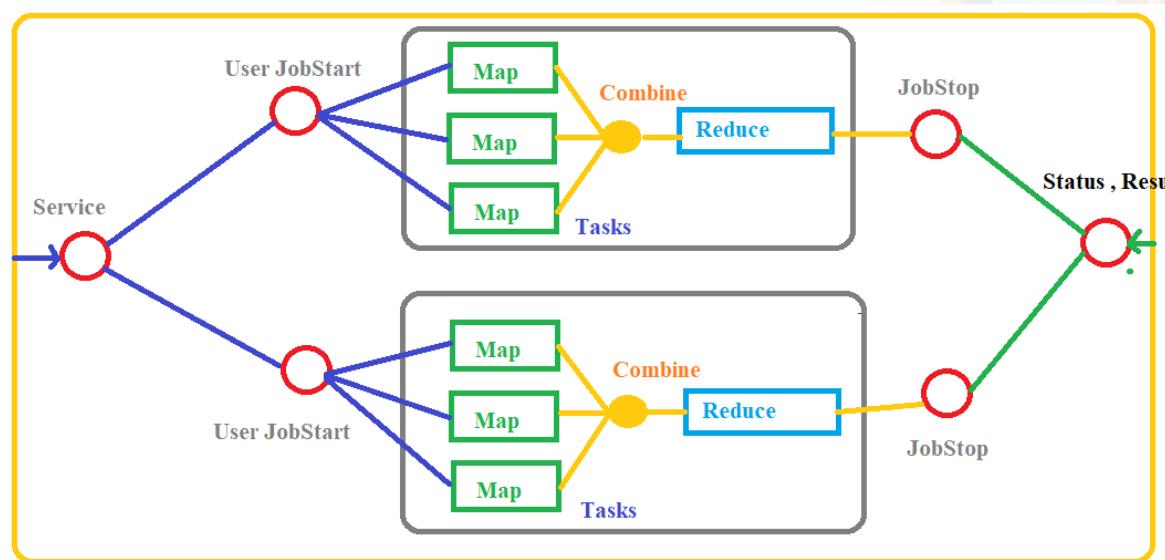
- ✓ Cluster services for
 - ✓ Availability
 - ✓ Redundancy
 - ✓ Fault-tolerance
 - ✓ Recovery from failures



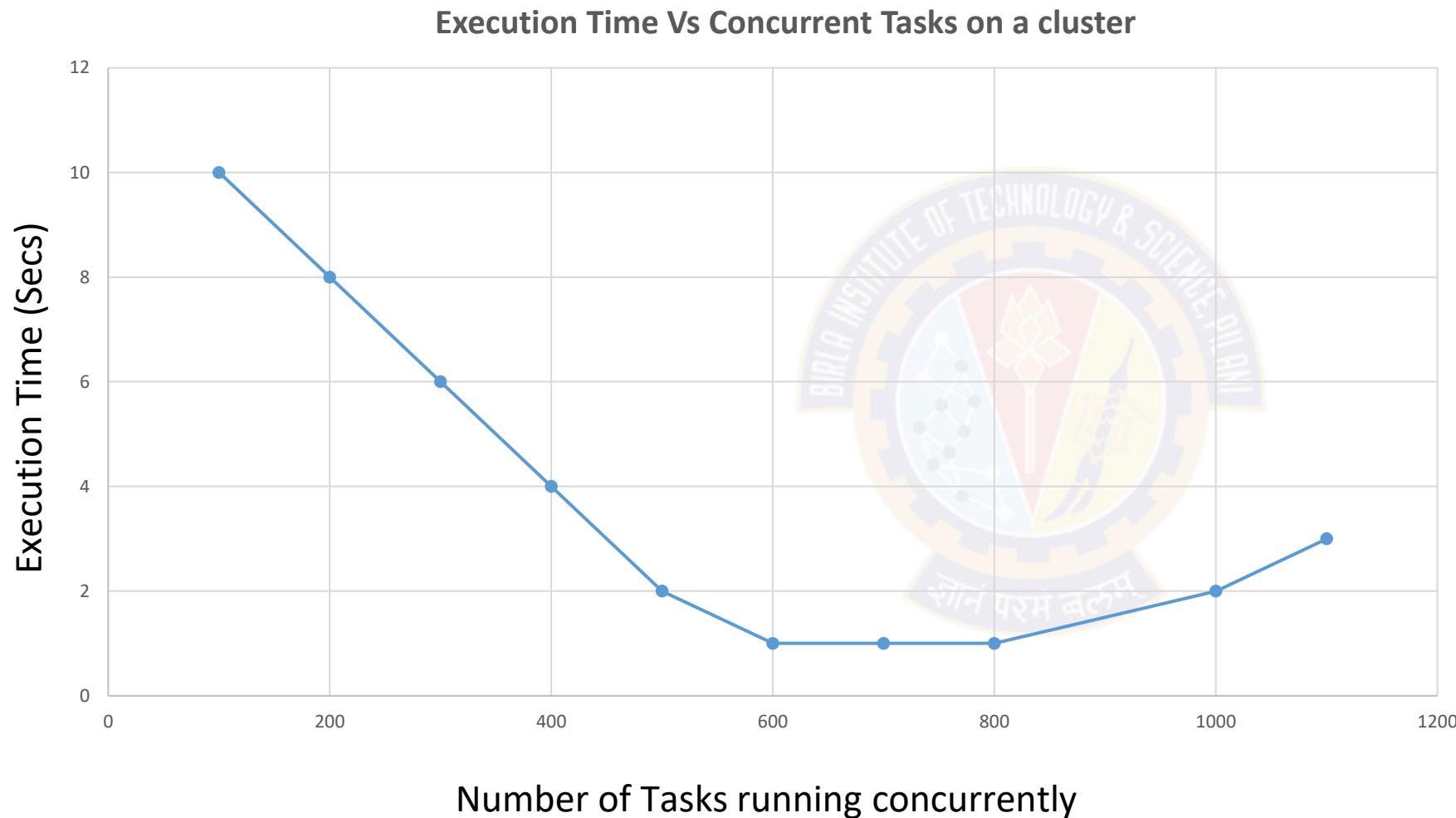
<http://www.cloudbus.org/papers/SSI-CCWhitePaper.pdf>

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

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

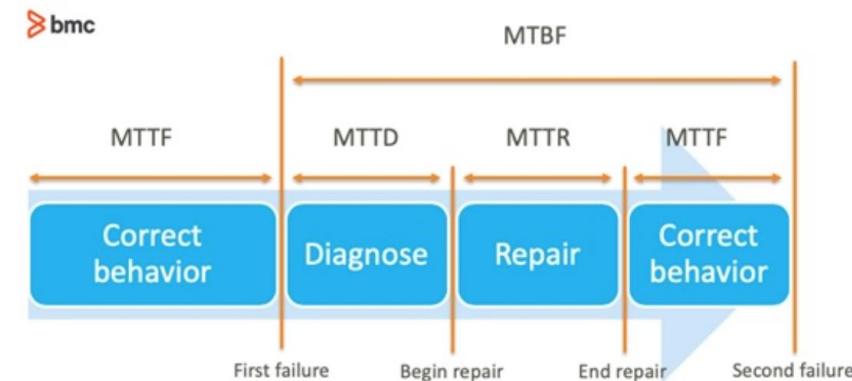
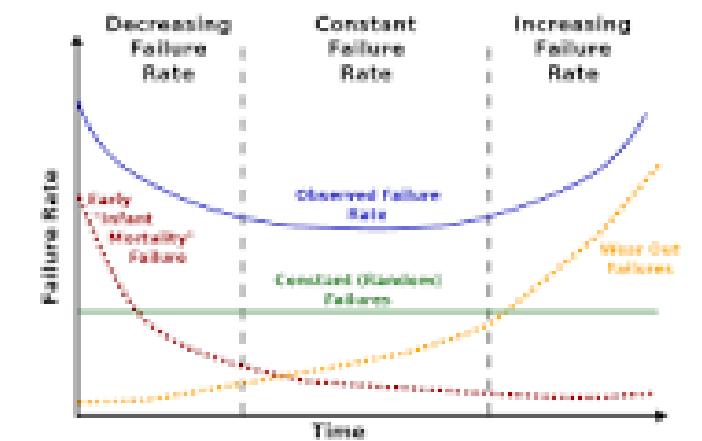
Availability %	Downtime per year	Downtime per month	Downtime per week
90% (one nine)	36.5 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (three nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
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

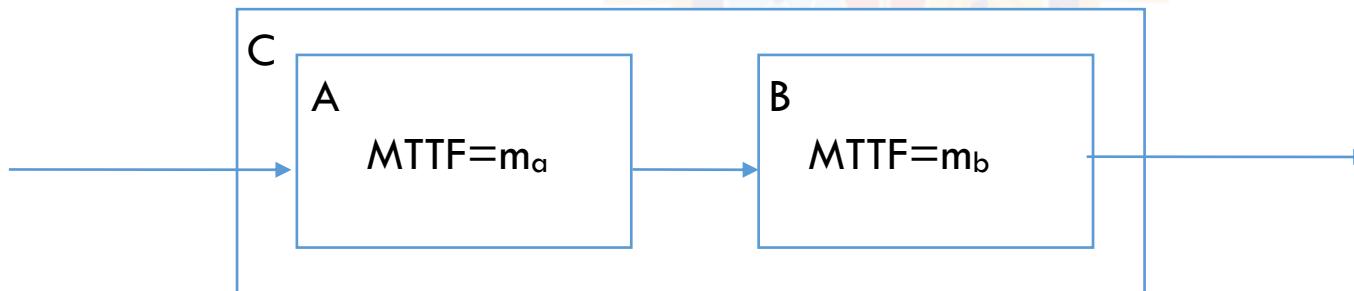


* Unless specified one can assume MTTD = 0

availabilitydigest.com

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

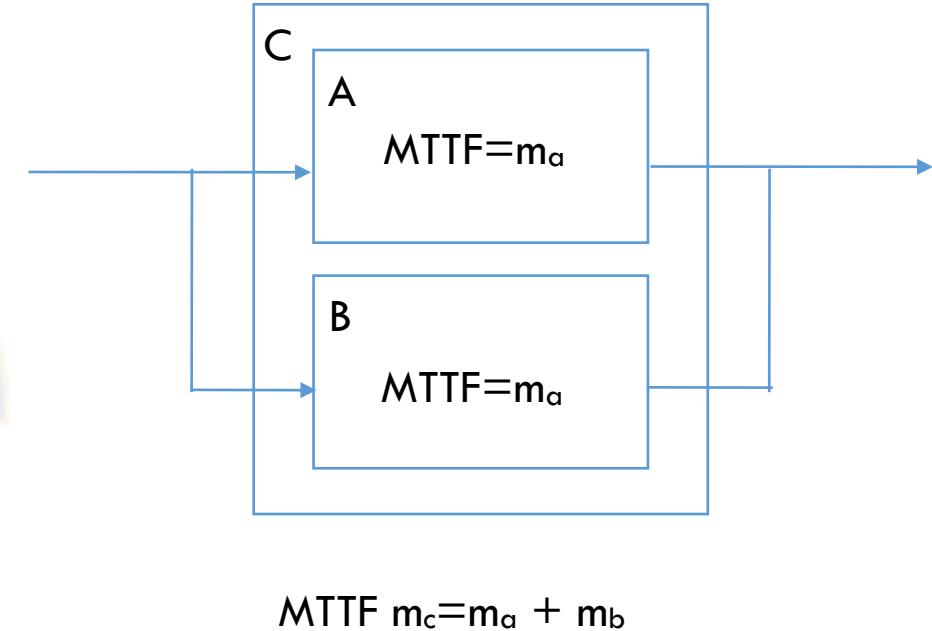
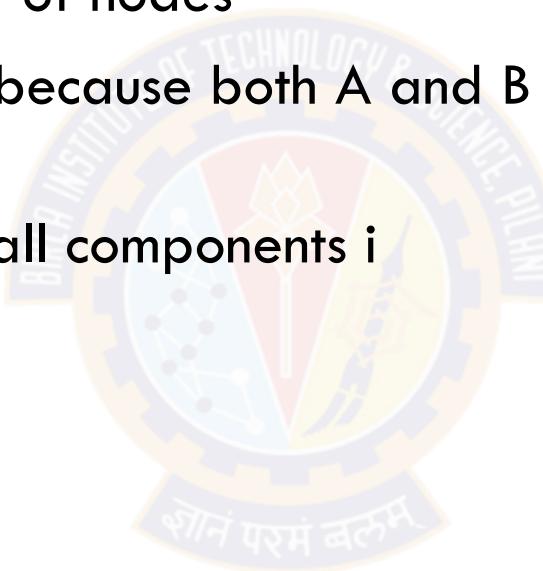


- The combined availability of system C is:

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

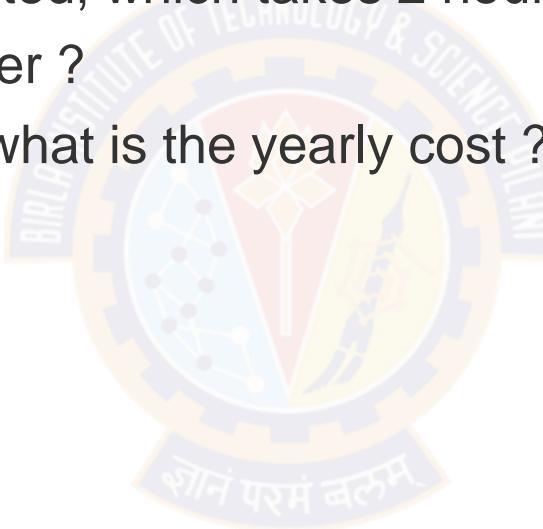
Reliability - parallel assembly

- 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 = $80000 \times 3.85 \times 365 \times 24 / 100 = \text{USD } 27 \text{ million}$

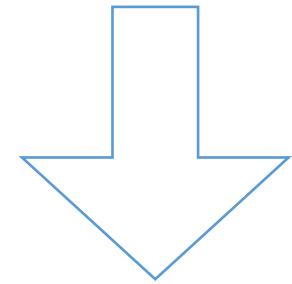


https://www.brainkart.com/article/Fault-Tolerant-Cluster-Configurations_11320/

Fault tolerance configurations - standby options

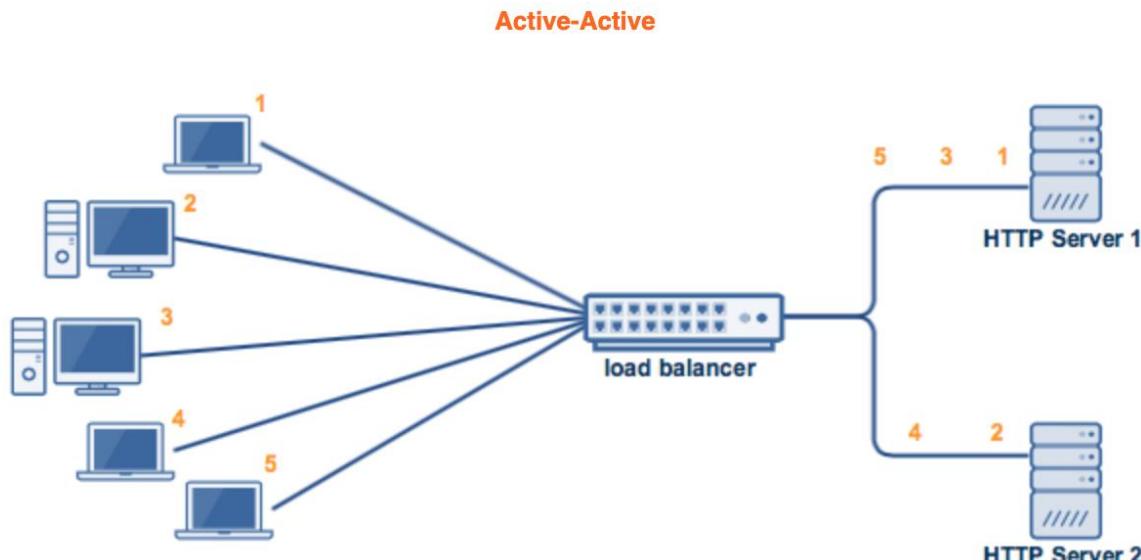
High availability model	Secondary node behavior	Data protection	Failover time
Load-balanced	Both the primary node and the secondary node are active and they process system requests in parallel. aka active-active	Data replication is bidirectional and is performed based on the software capabilities.	Zero 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. aka active-passive	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

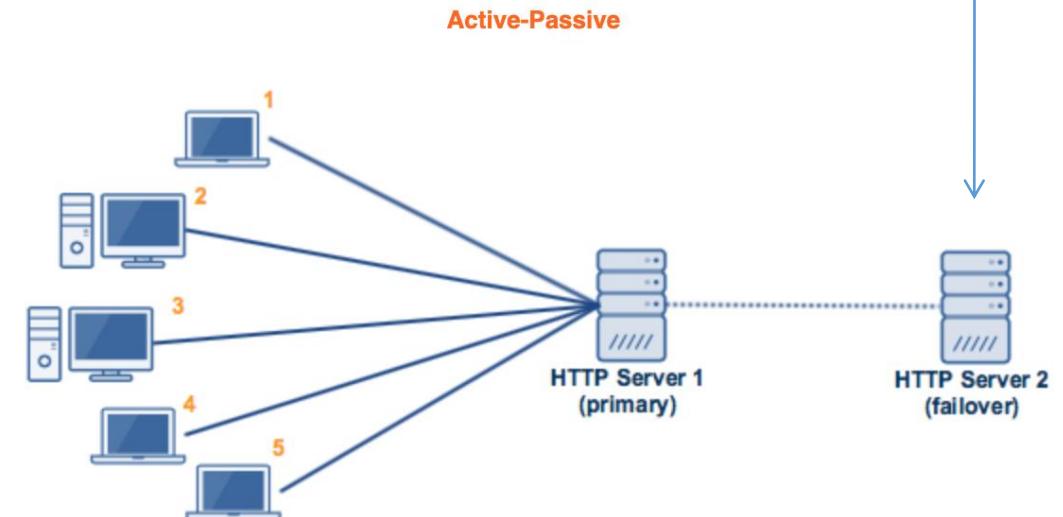


Assumption:
Same software bug or runtime fault will not recur in the standby

Fault tolerance configurations - standby options

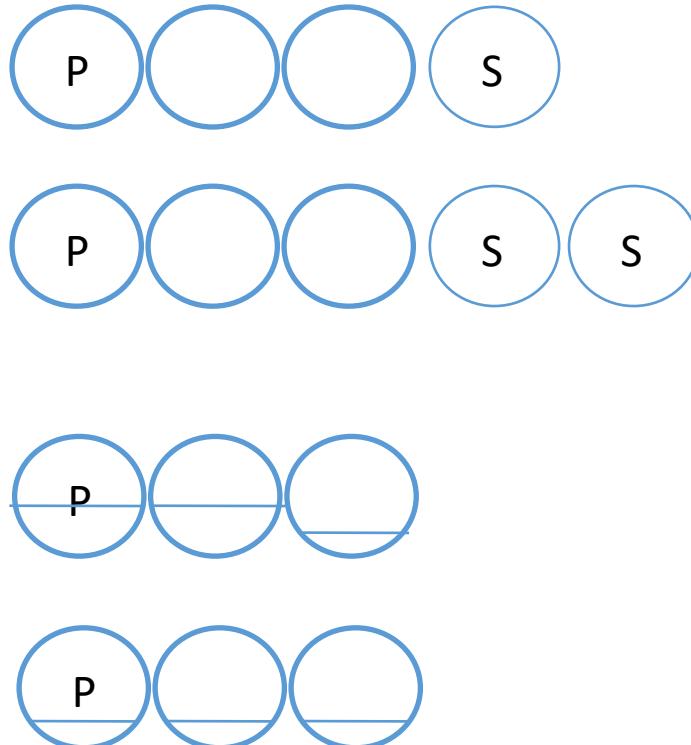


Warm and cold
depends on how the
passive / failover node
is kept updated



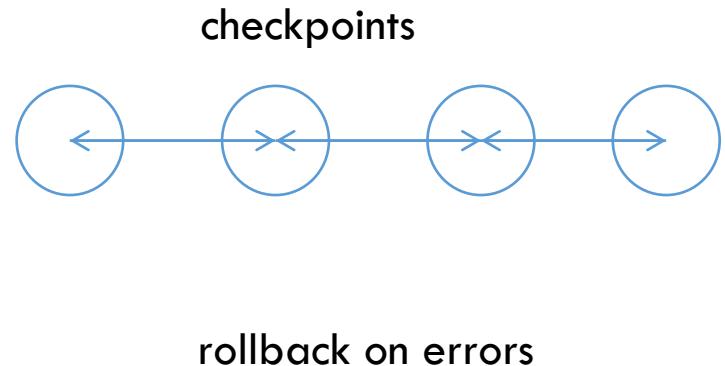
Fault tolerance configurations - cluster topologies

- 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

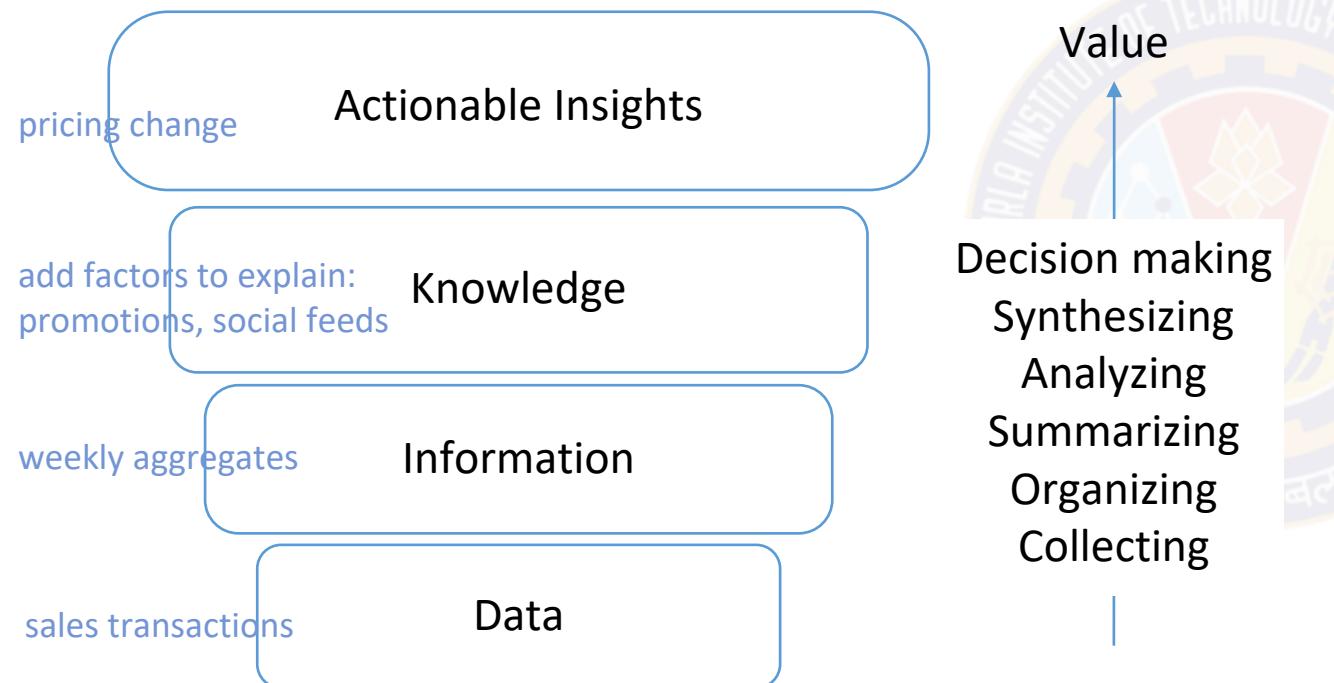
Analytics - Definitions

- 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



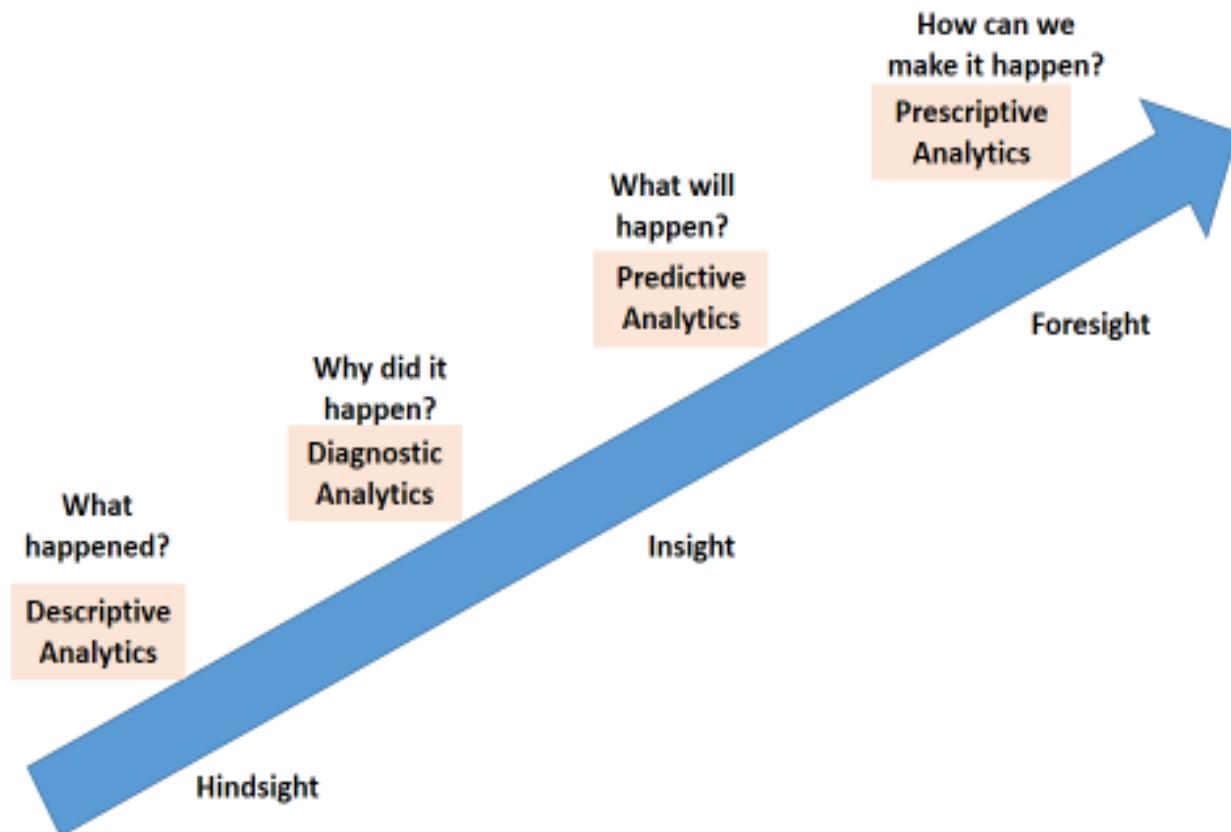
Process of Analysis

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



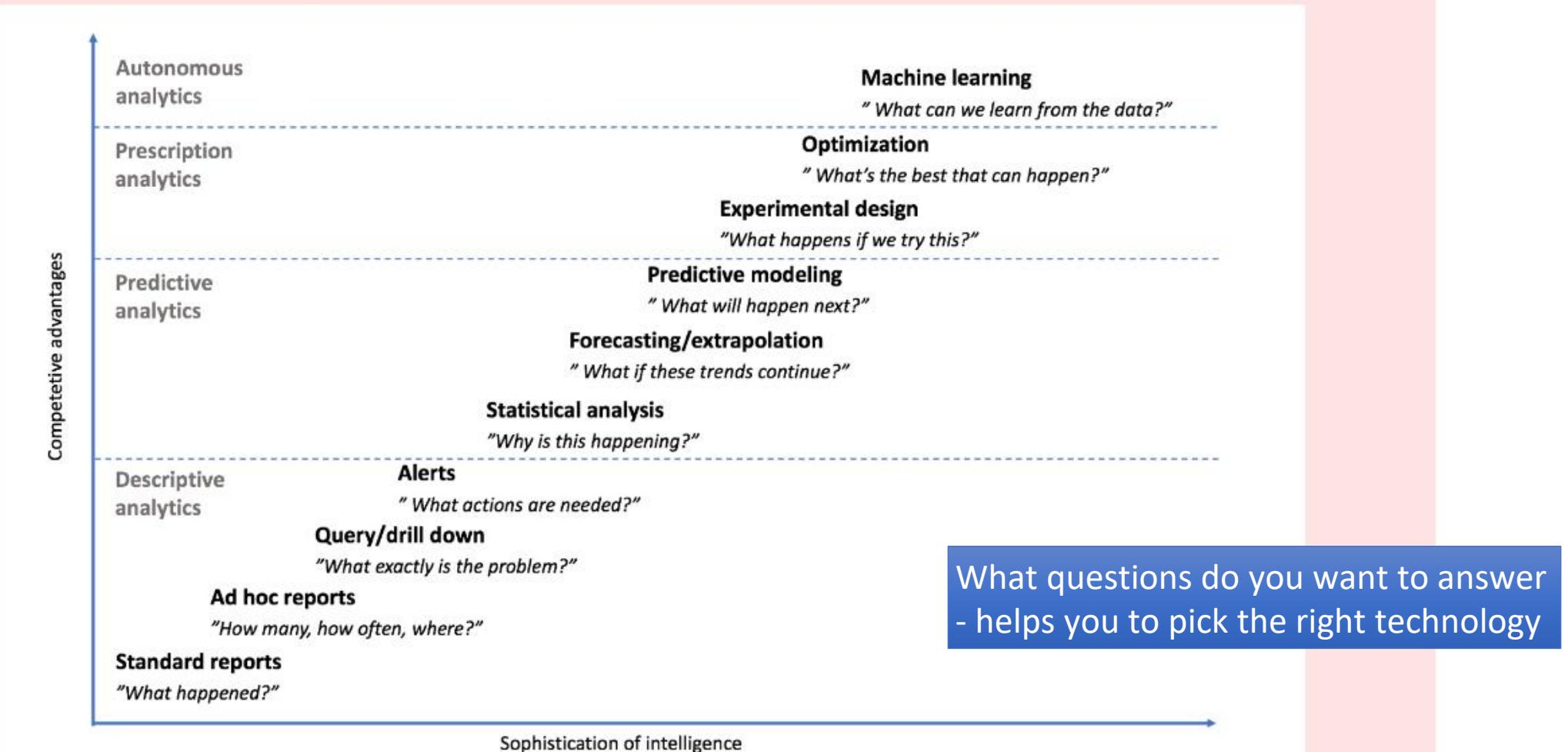
Analytics 1.0 → deals with historical data to report on events, occurrences of the past.

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

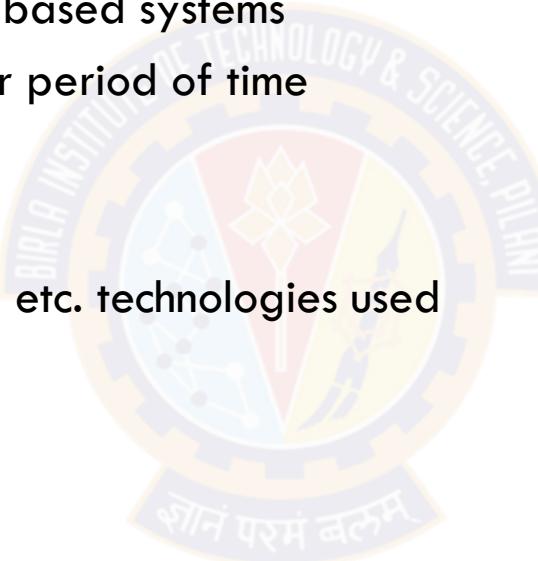
Types of Analytics - Descriptive

- 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

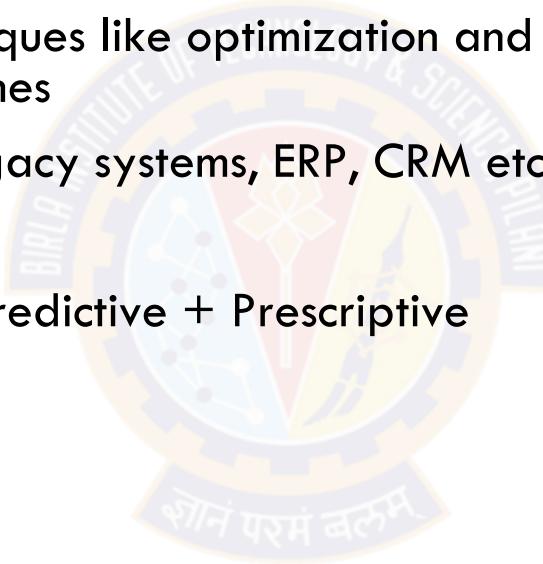
- Uses past data to predict the future
- Uses quantitative techniques like segmentation, forecasting etc. but also makes use of descriptive analytics for data exploration
- Uses technologies like models and rule based systems
- Based on large data set gathered over period of time
- Externally sourced 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

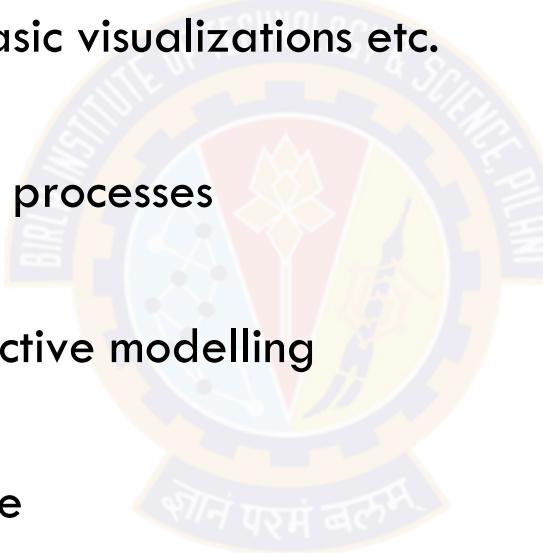
- Uses data from past to make prophecies of future and at the same time make recommendations to leverage the situation to one's advantage
- Suggests optimal behaviors and actions
- Uses a variety of quantitative techniques like optimization and technologies like models, machine learning and recommendations 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 ?

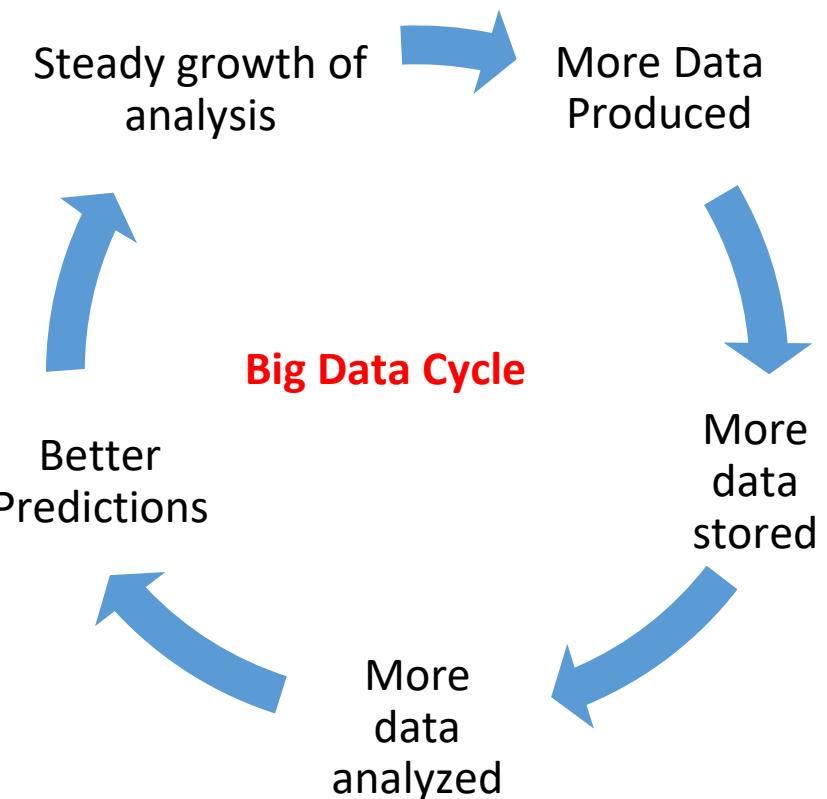
What Big Data Analytics is not



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

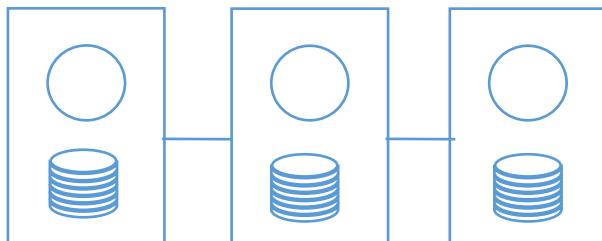
- Cheap abundant storage
- Processing options
 - batch / streaming,
 - disk based / memory based
- Open source platforms, e.g. Hadoop, Spark
- Parallel and distributed systems with high throughput rather than low latency
- Cloud or other flexible resource allocation arrangements
 - Flexibility to setup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



Technology challenges (1)

- Scale

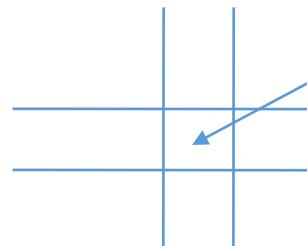
- ✓ Need is to have storage that can best withstand large volume, velocity and variety of data
- ✓ Scale vertically / horizontally ?
- ✓ RDBMS / NoSQL ?
- ✓ How does compute scale with storage - coupled or de-coupled, i.e. good idea to put common nodes for compute and storage



compute + data on same node: locality helps, but does compute scale with 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)



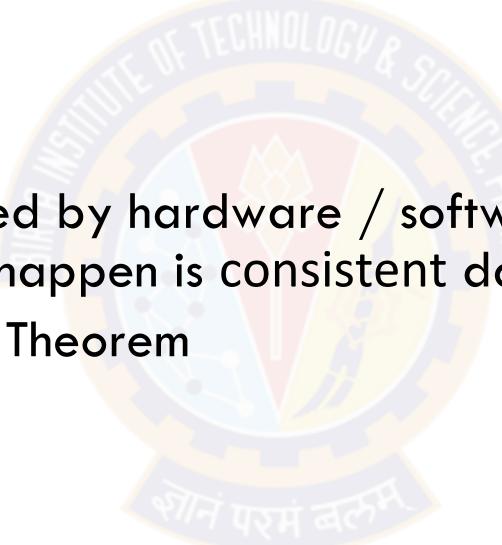
Easier in RDBMS to control at row / column / cell level with always consistent values in a tightly coupled system

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

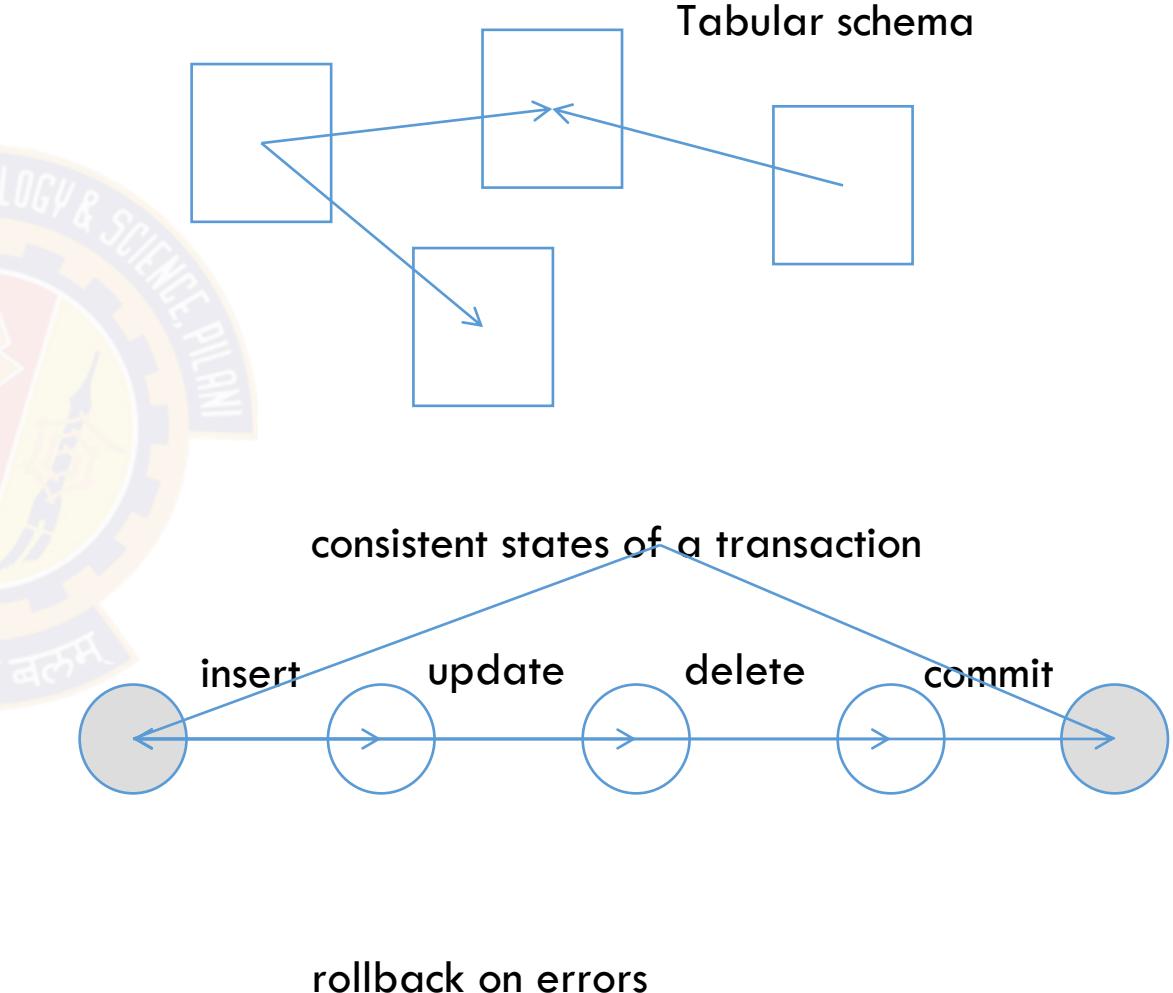
✓ Solution

- ✓ 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**

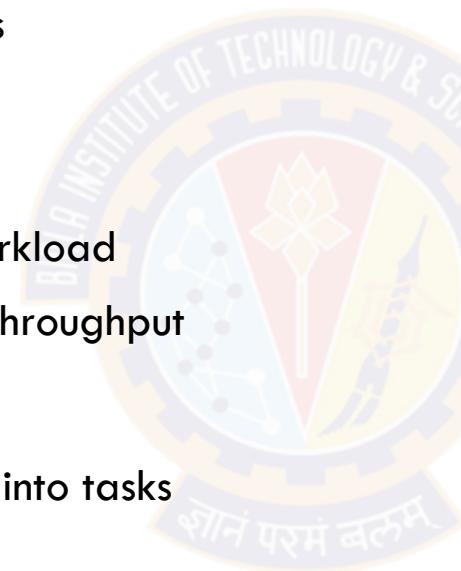
- Data model
 - Tables and relations
 - Schema is predefined (during write)
 - Supports partitioning
 - Fast indexed reads
 - Read and write many times
- Compute model
 - Generate notations of a transaction
 - ACID properties
 - Allow distributed transactions
 - OLTP workloads



Hadoop in contrast ...

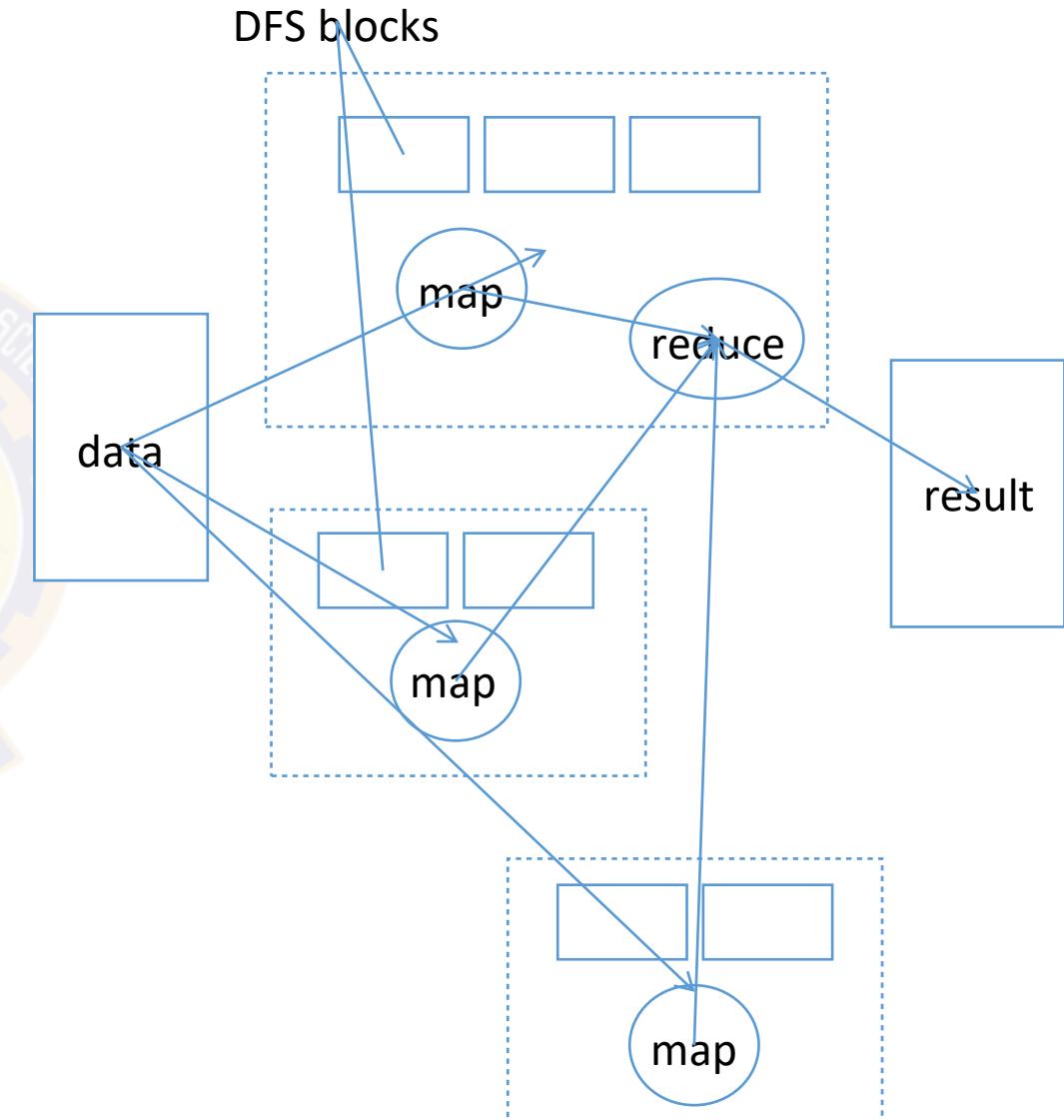
- Data model

- 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
MapReduce (Cluster Resource Manager & Data Processing)

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



Hadoop Distributions

Cloudera

CDH 4.0

CDH 5.0

Hortonworks

HDP 1.0

HDP 2.0

MAPR

M3

M5

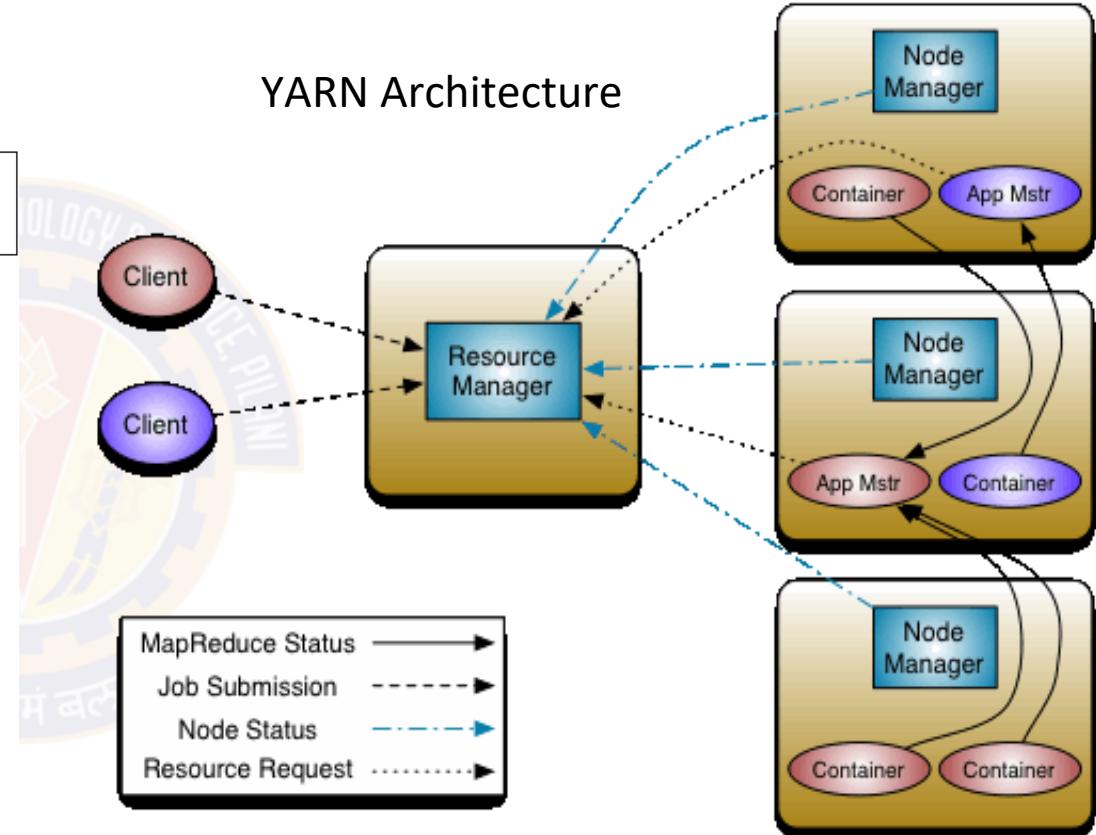
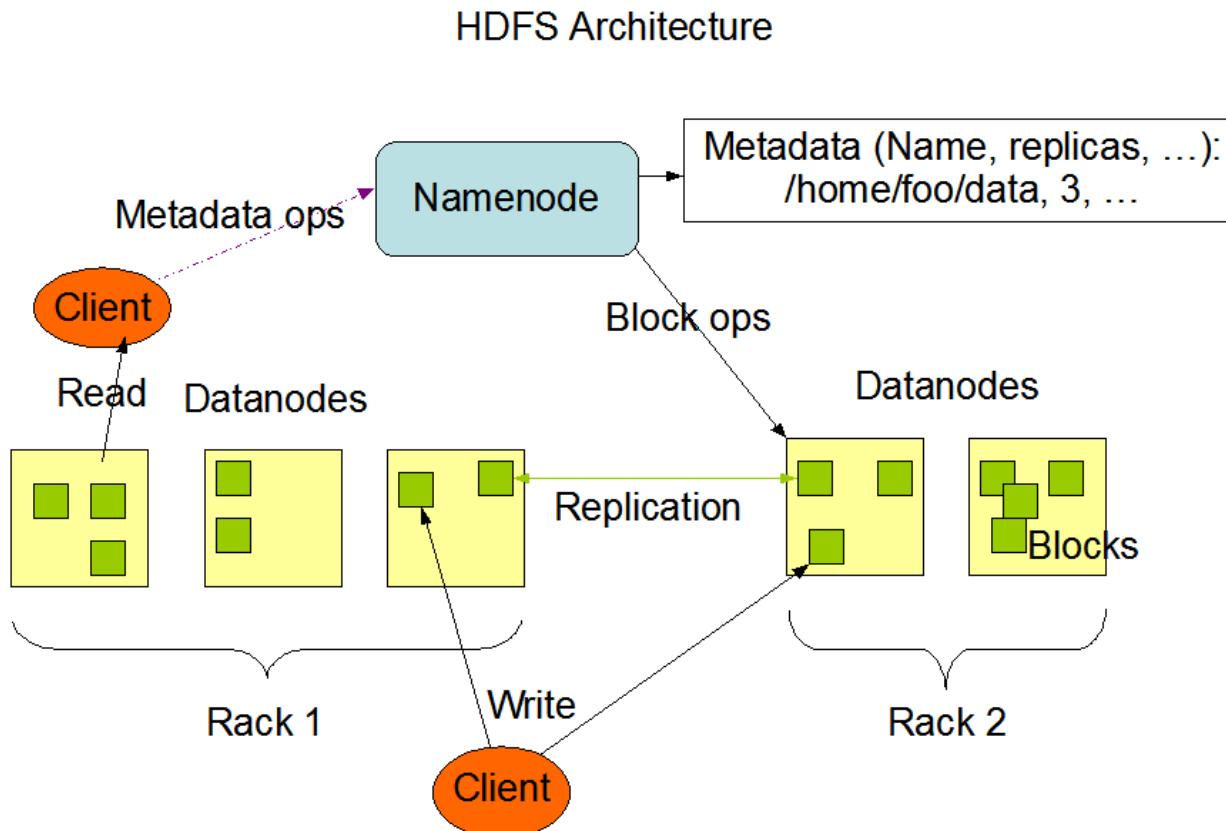
M8

Apache Hadoop

Hadoop 1.0

Hadoop 2.0

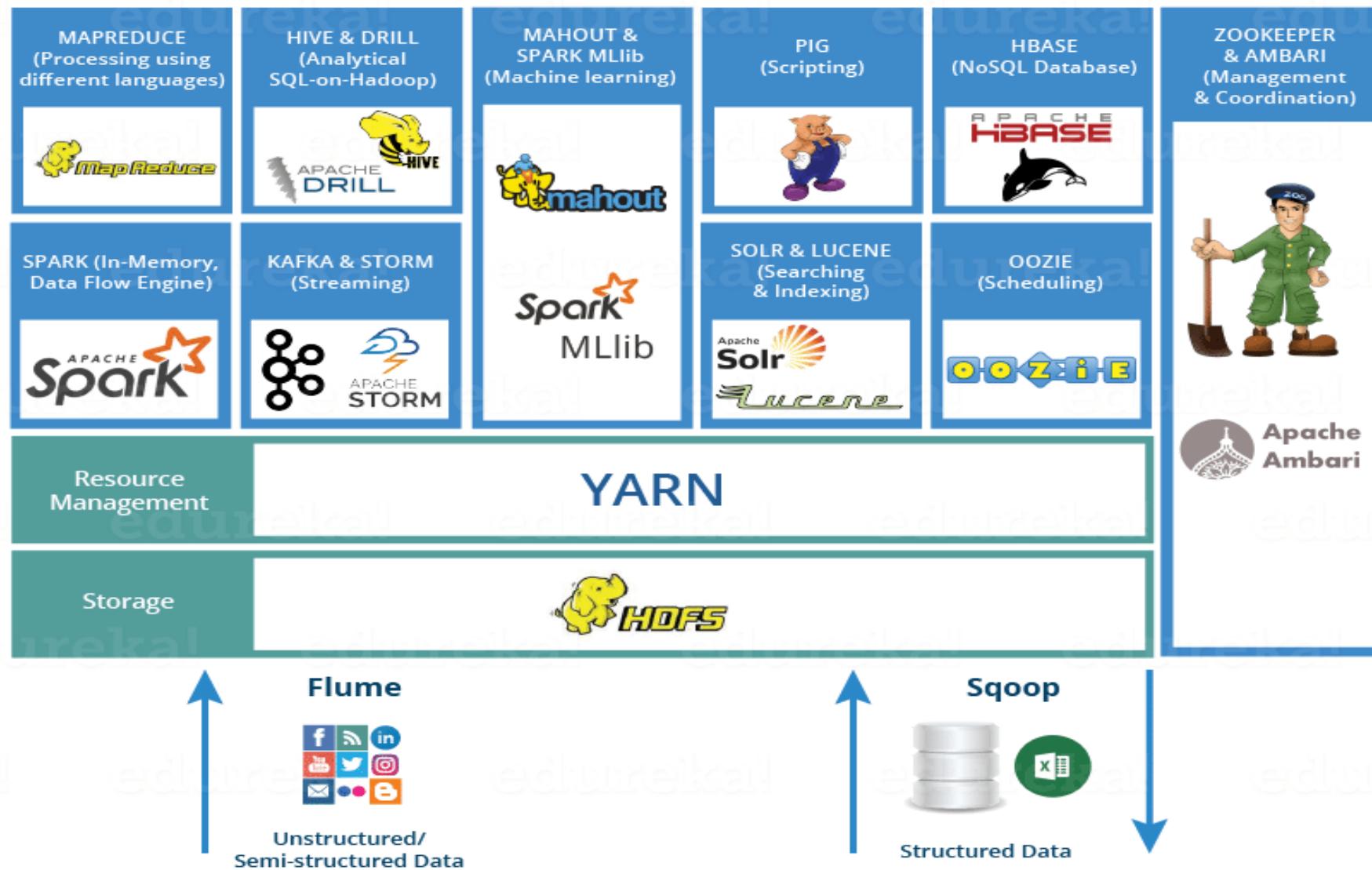
Hadoop 2.0 high level architecture



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:

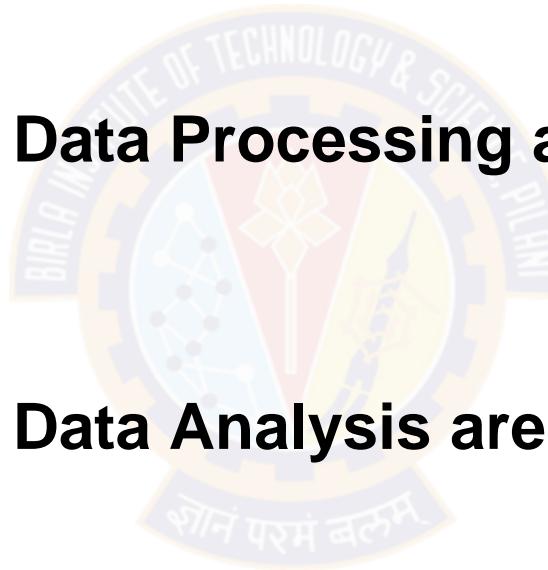
1. Sqoop
2. Flume

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.

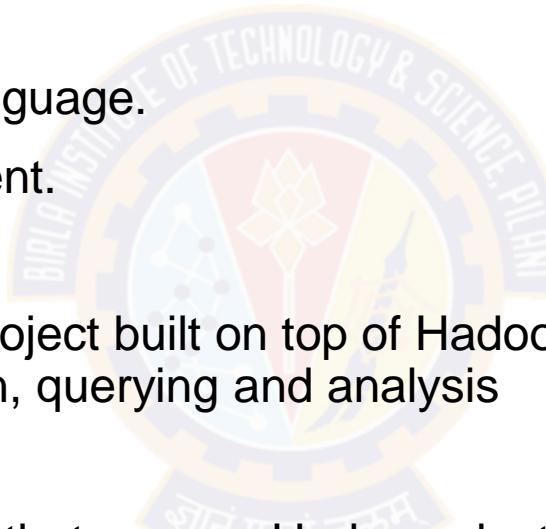
Spark:

- 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

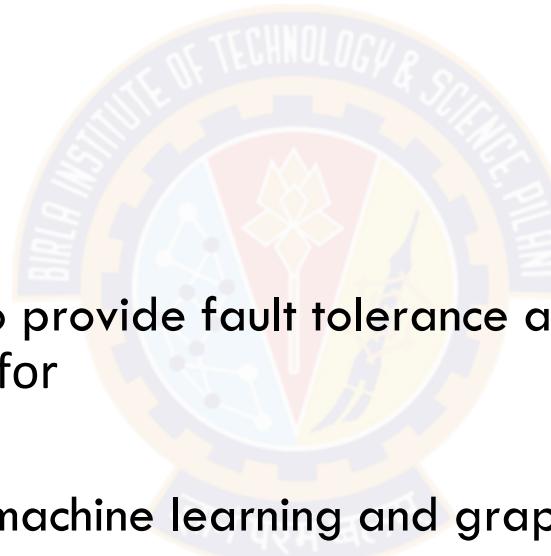




Introduction to in-memory computing

Issues with MapReduce on Hadoop

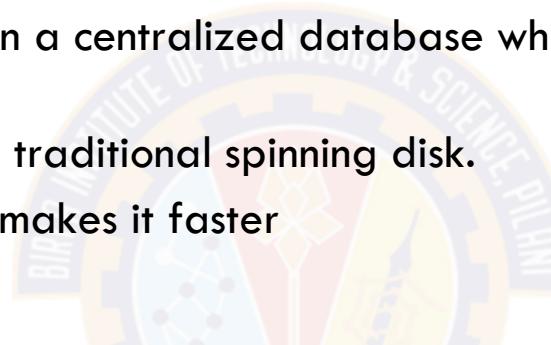
- ✓ Revolutionized big data processing, enabling users to store and process huge amounts of data at very low costs.
- ✓ An ideal platform to implement complex batch applications as diverse as
 - sifting through system logs
 - running ETL
 - computing web indexes
 - recommendation systems etc.
- ✓ Its 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>

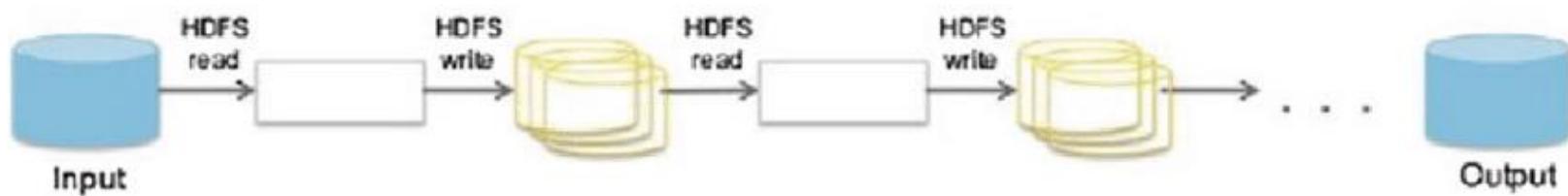
In-Memory computing

- 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

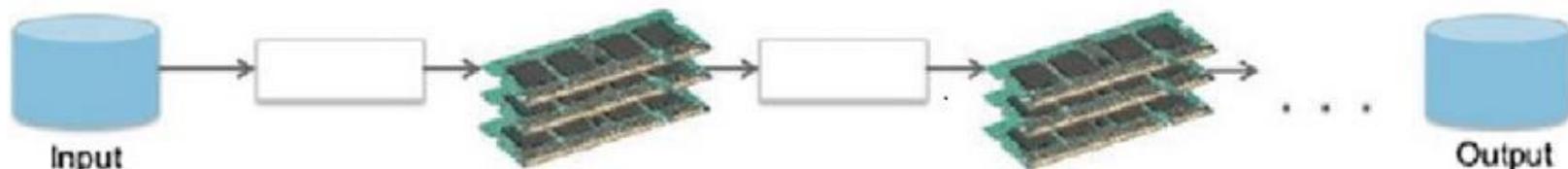


Note:
Could be batch or streaming

Hadoop MapReduce: Data Sharing on Disk



Spark: Speed up processing by using Memory instead of Disks



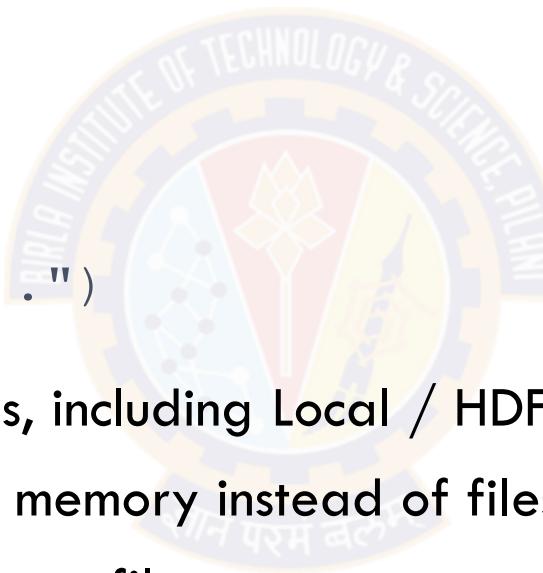
Fast and easy big data processing with Spark

- 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

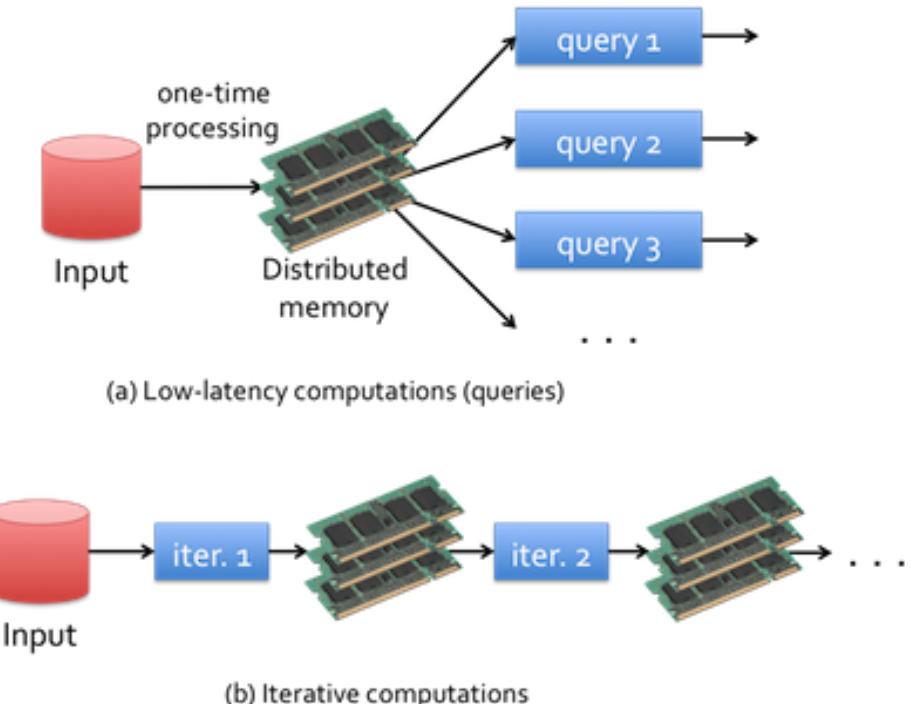
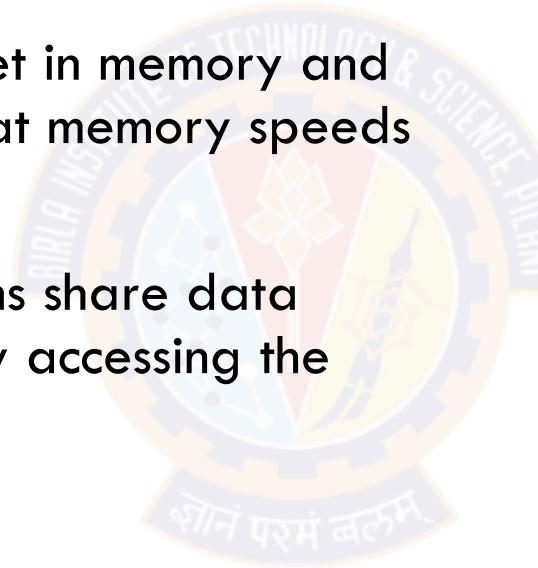
map: transform each word into $\langle k, v \rangle$ pair

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

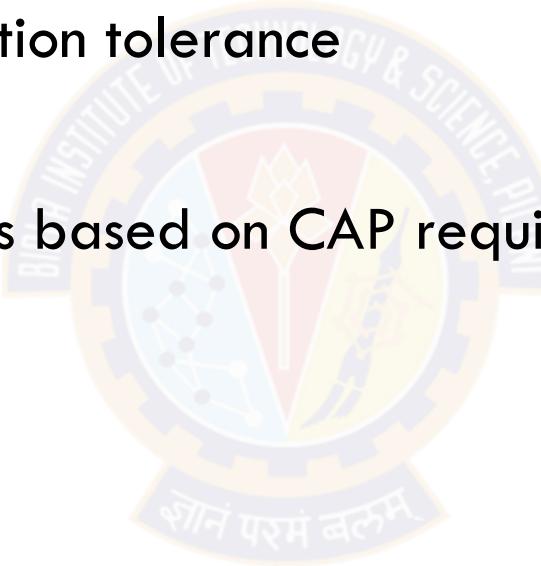
Big Data Systems

Session 4 – BDA Lifecycle,CAP Theorem, NoSQL

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

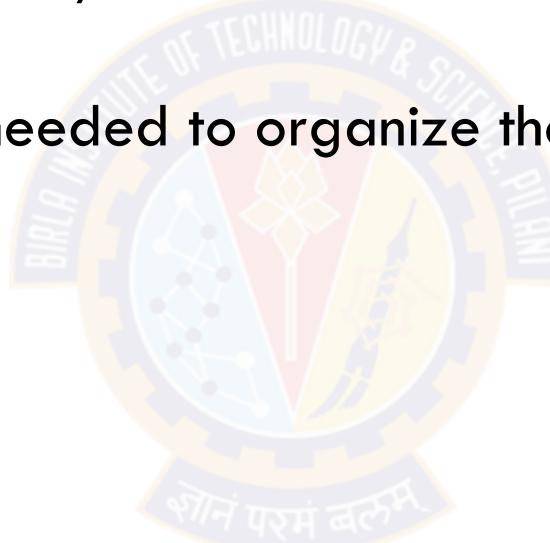
Topics for today

- **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



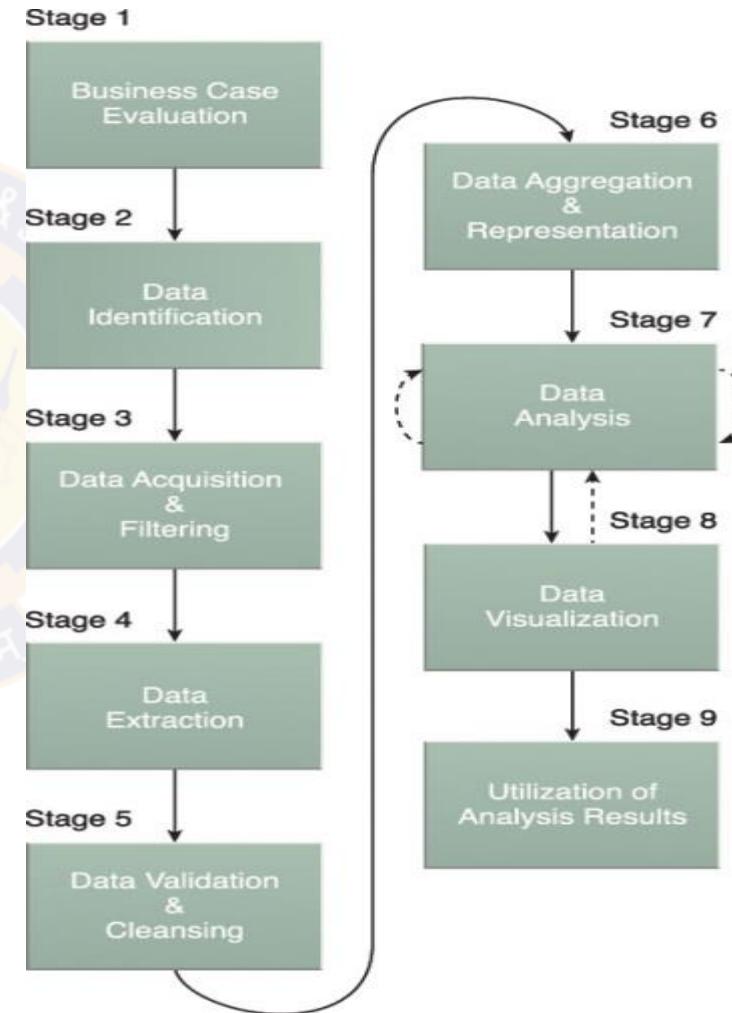
Example

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

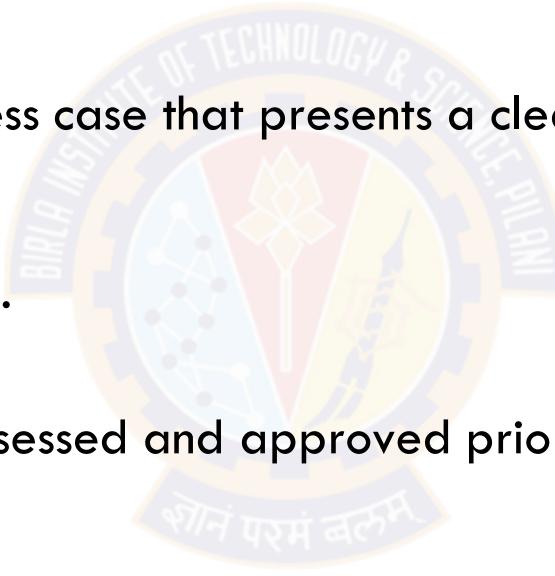
Big Data Analytics Lifecycle

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

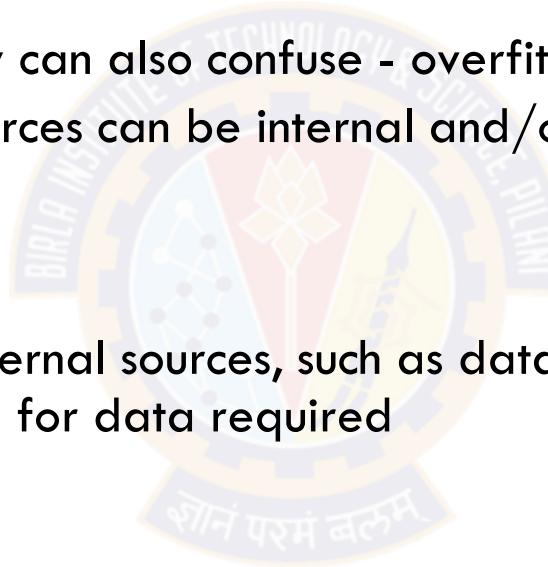


1. Business Case Evaluation

- 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

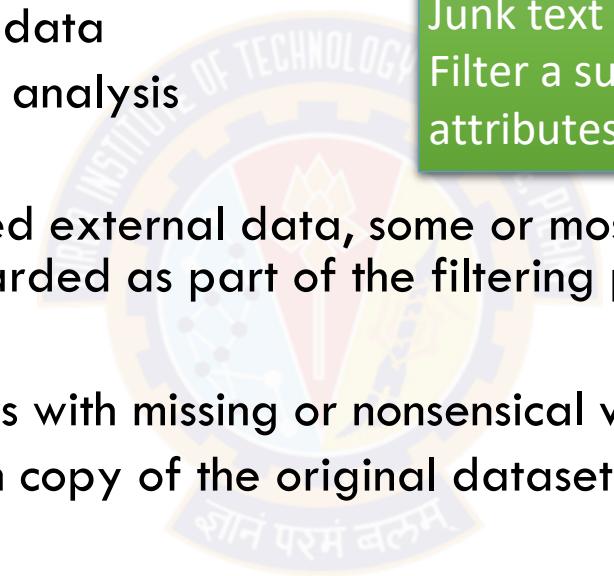


Identify respondents
Demographics
What questions to ask
Do we need other surveys

3. Data Acquisition & Filtering

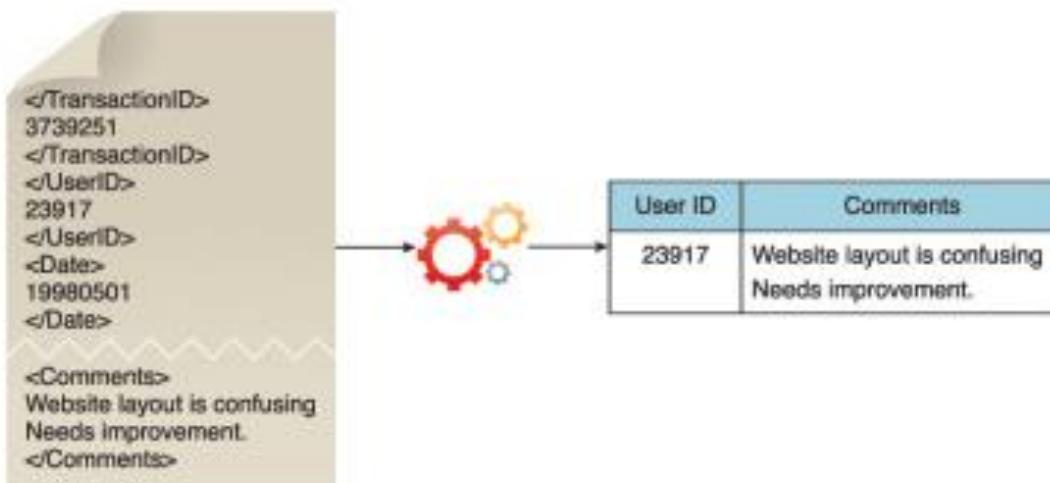
- The data is gathered from all of the data sources that were identified during the last stage
- The acquired data is then looked upon for
 - ✓ filtering / removal of corrupt data
 - ✓ removal of unusable data for analysis
- In many cases involving unstructured external data, some or most of the acquired 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

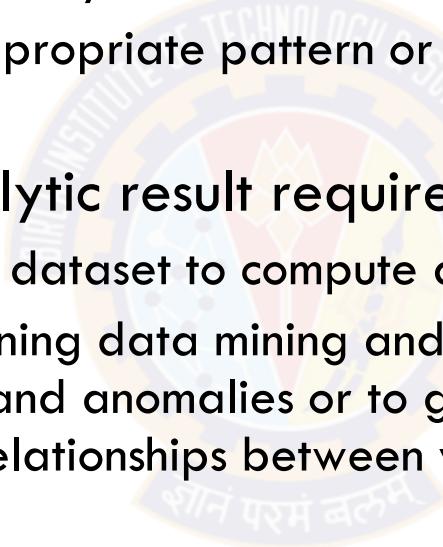
6. Data Aggregation & Representation

- 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.

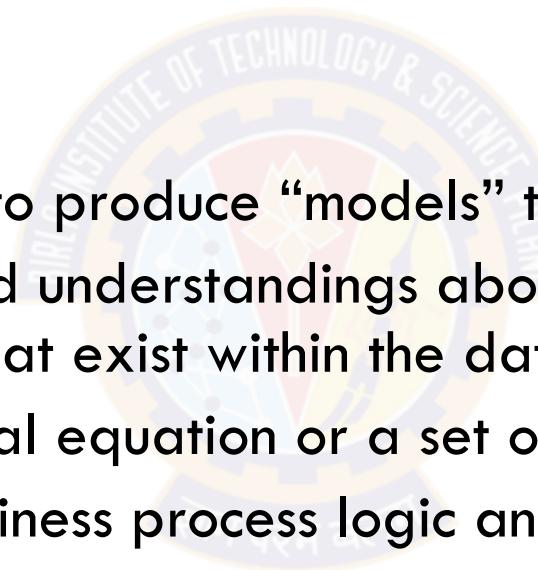
8. Data Visualization

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

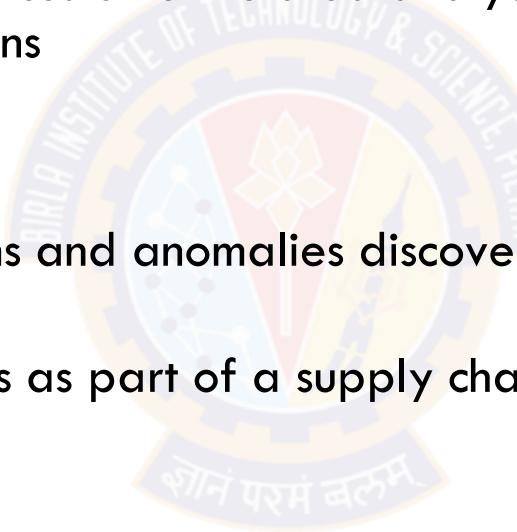
- Dedicated to determining how and where processed analysis data can be further leveraged
 - ✓ Apart from dashboards
- Possible for the analysis results to produce “models” that
 - ✓ encapsulate new insights and understandings about the nature of the patterns and relationships that exist within the data 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.

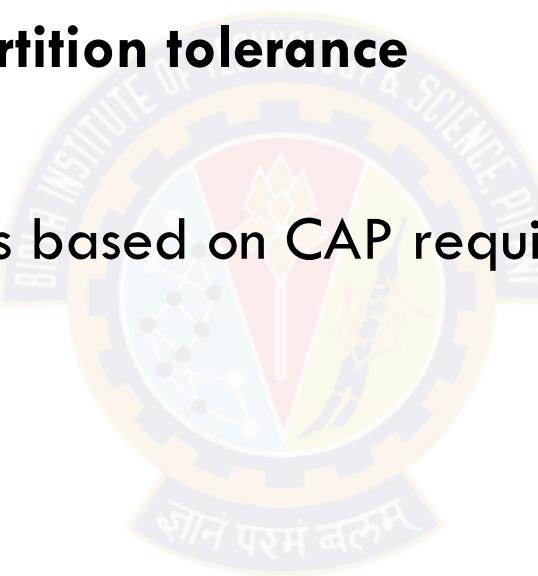
9. Utilization of Analysis Results

- 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>

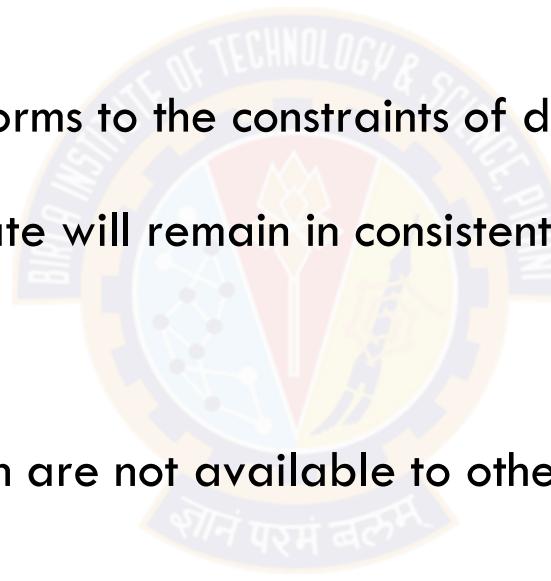
ACID

- 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.



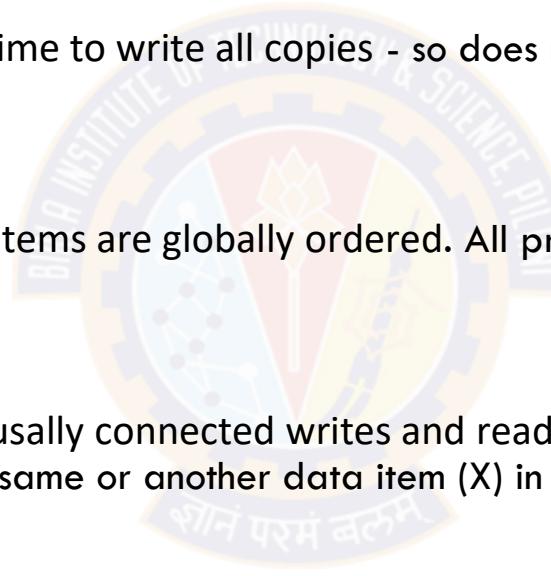
Consistency clarification

- 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

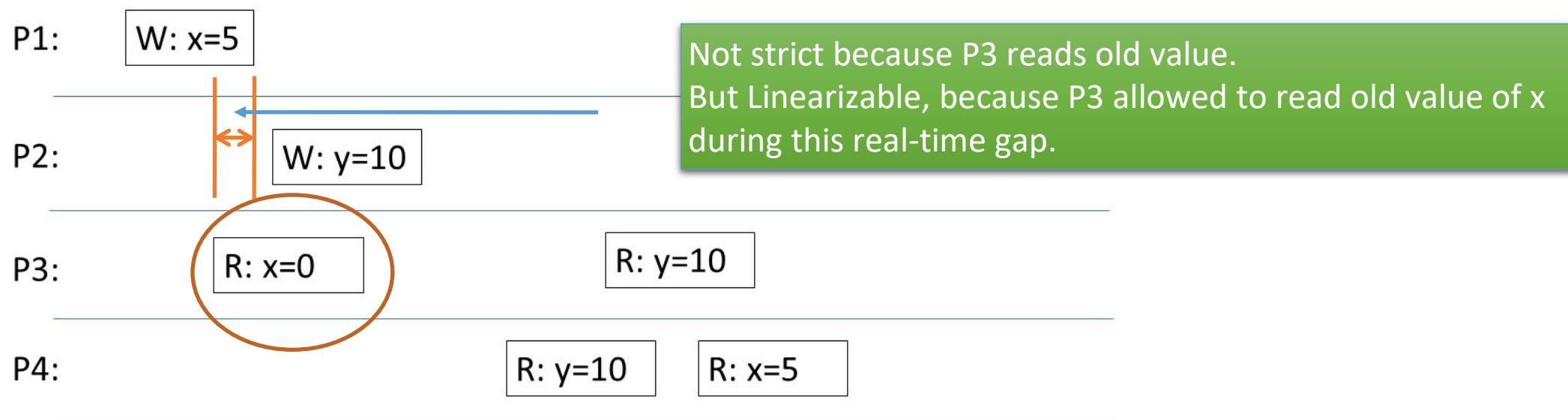
P4: R: y=10 R: x=5

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)

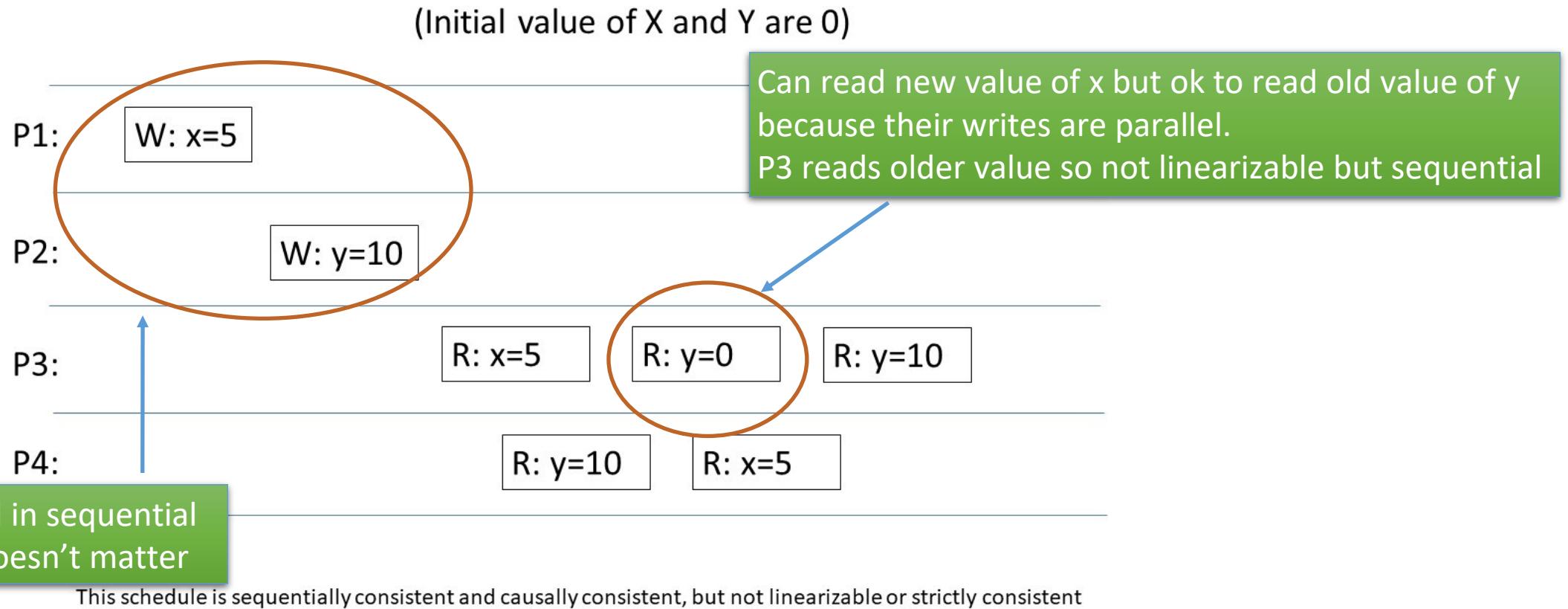


This schedule is sequentially consistent, causally consistent, and linearizable, but **not** strictly consistent

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.

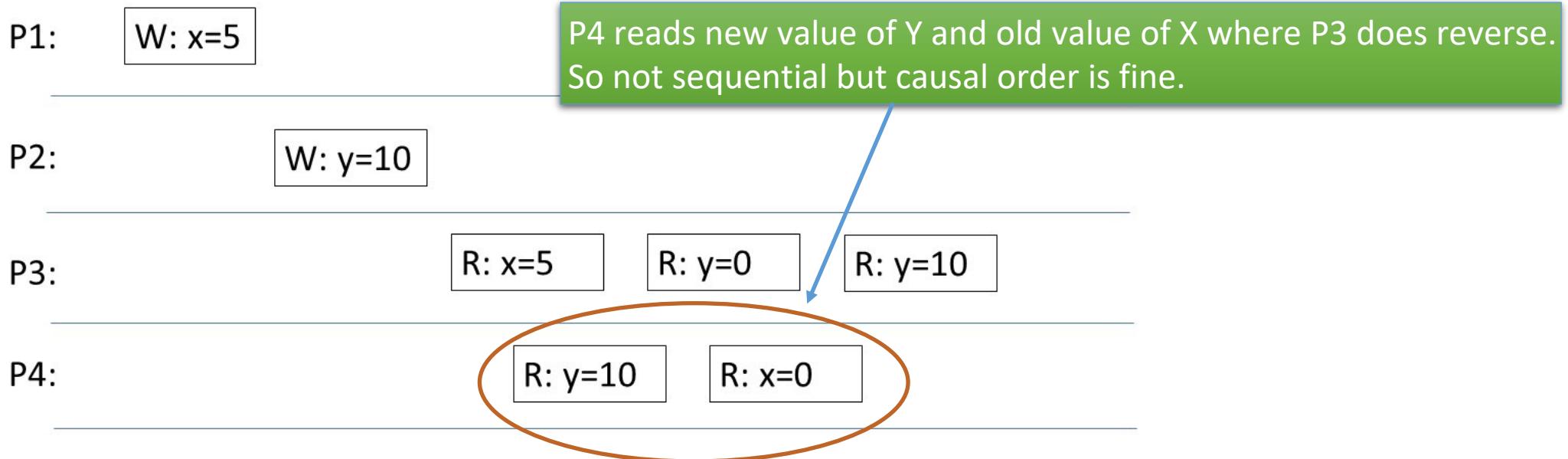


All writes across threads are globally ordered but not in real-time. Here P3 reads older value of Y in real-time but P1 and P2 writes are deemed parallel in sequential order.

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)

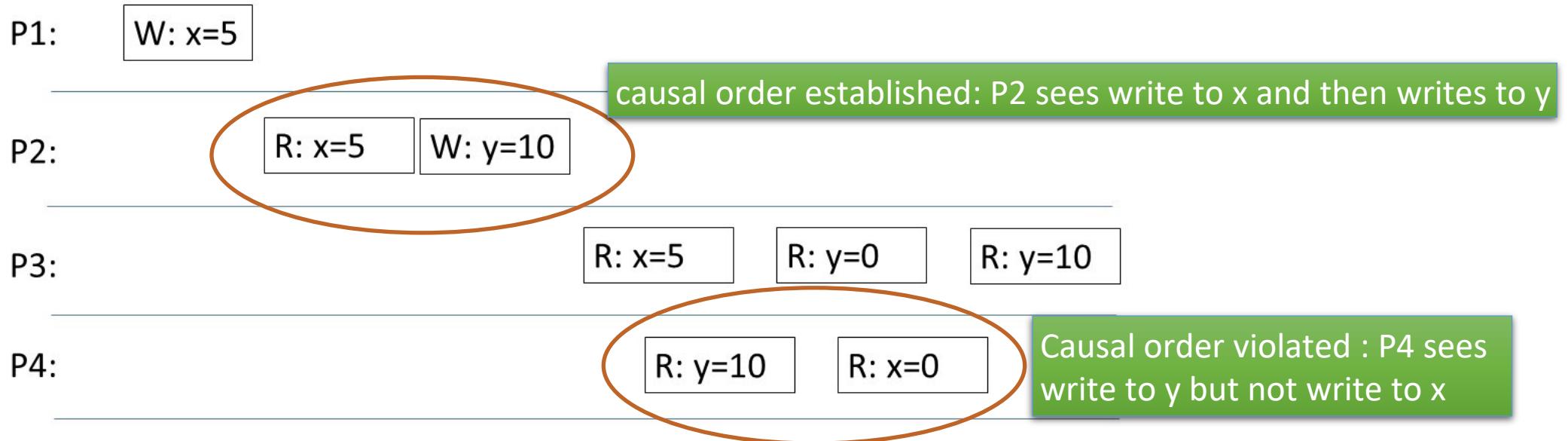


This schedule is causally consistent, but not linearizable or strictly consistent or even sequentially consistent

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)



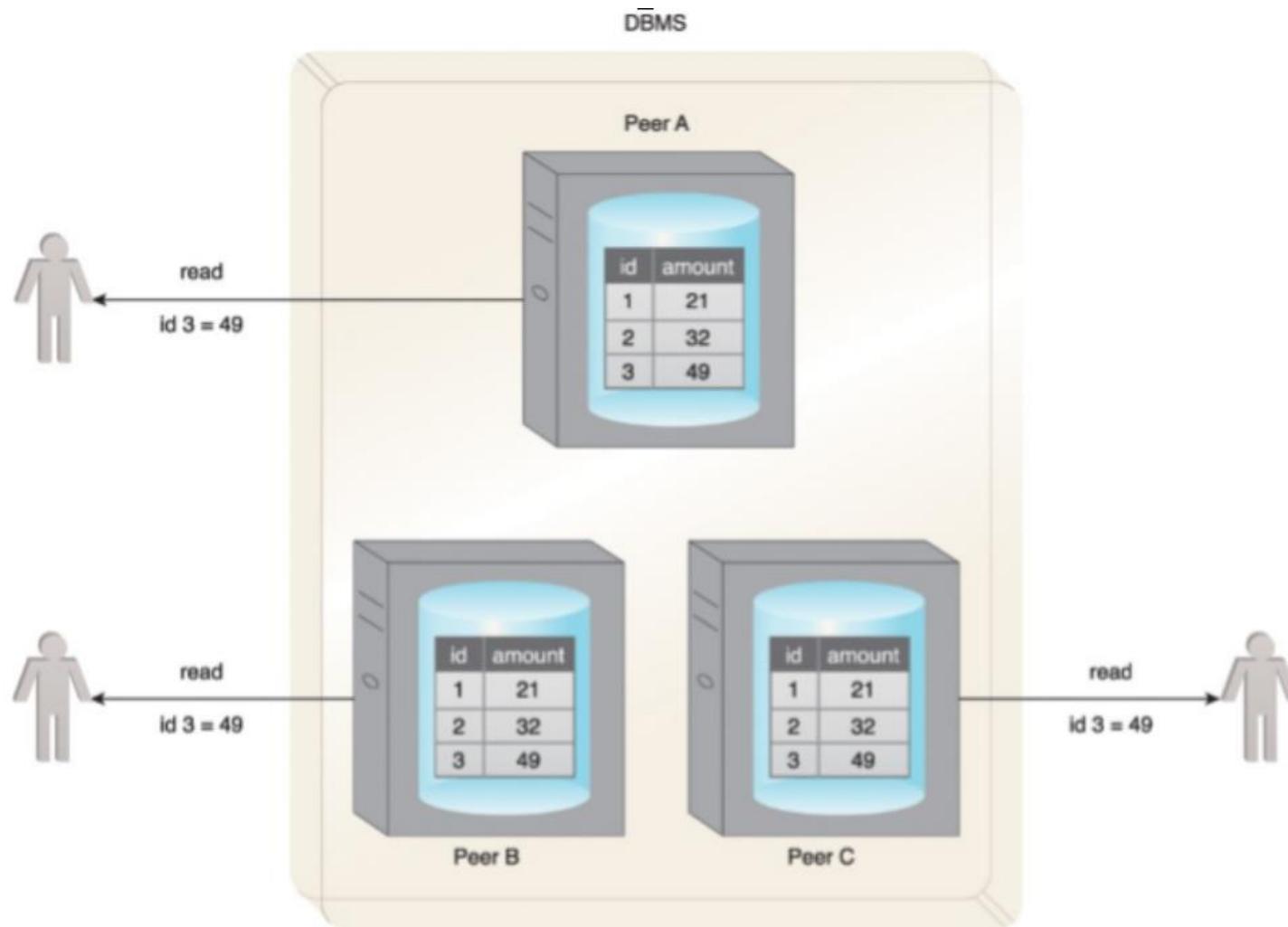
This schedule is **not** causally consistent, nor linearizable or strictly consistent or sequentially consistent

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

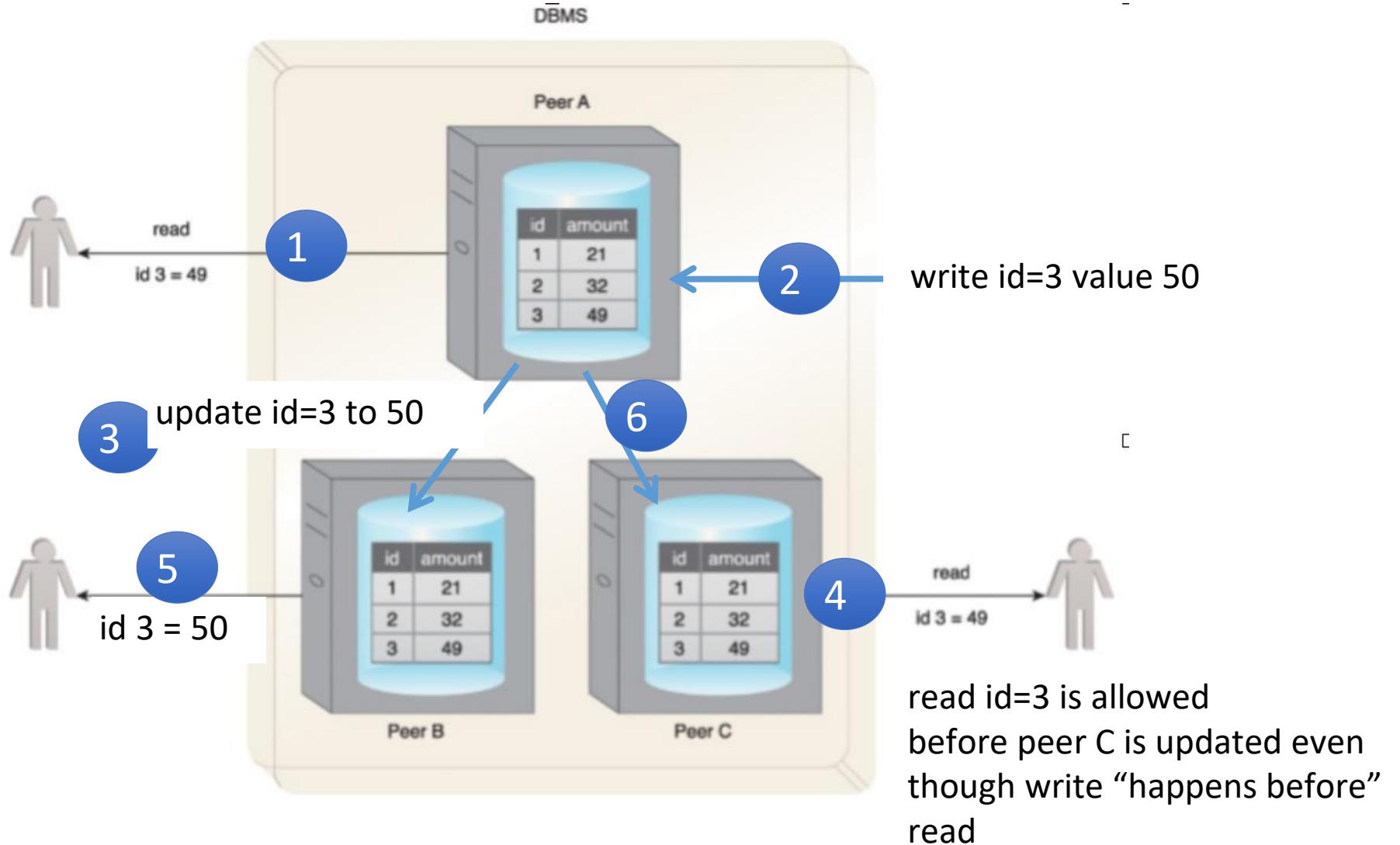
split brain problem

C: Consistency

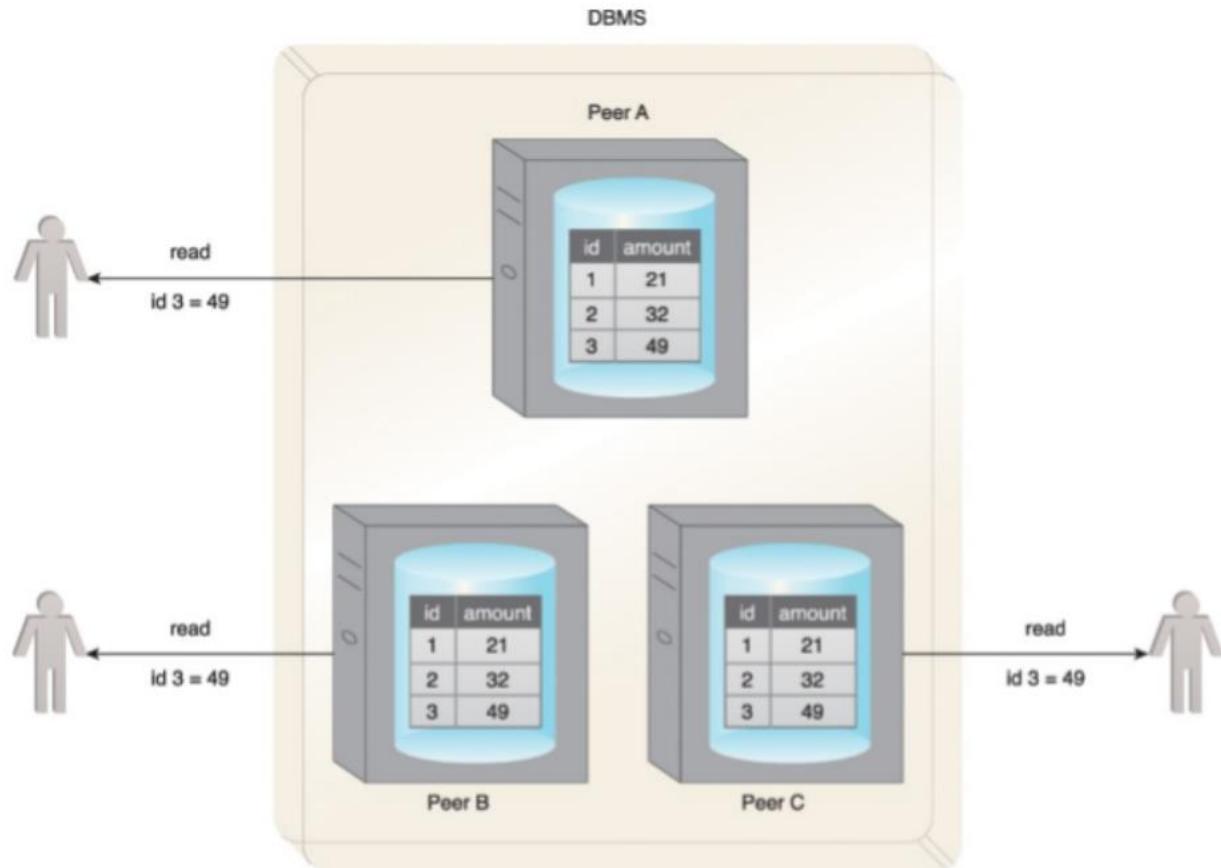


All users get the same value for the amount column even though different replicas are serving the record

When can it be in-consistent

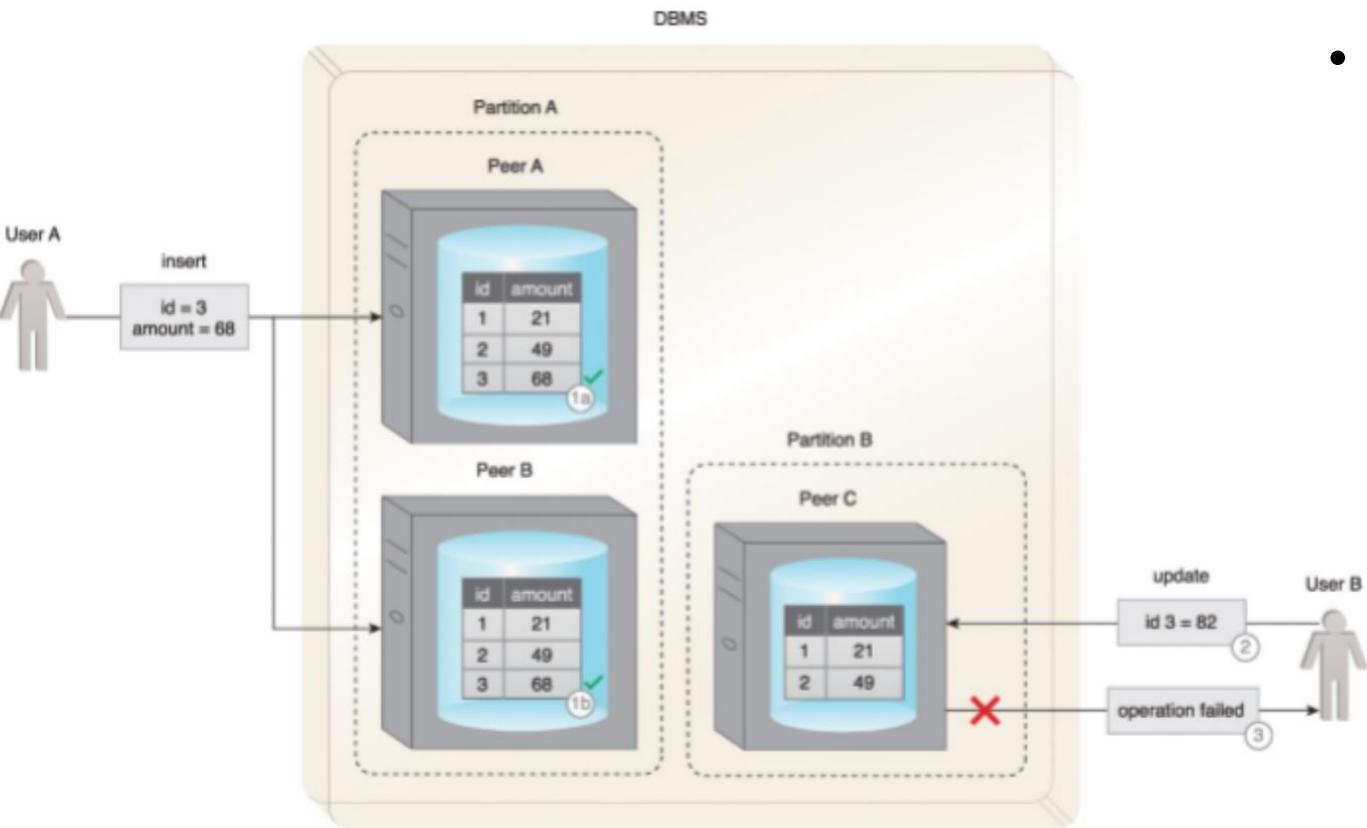


A: Availability



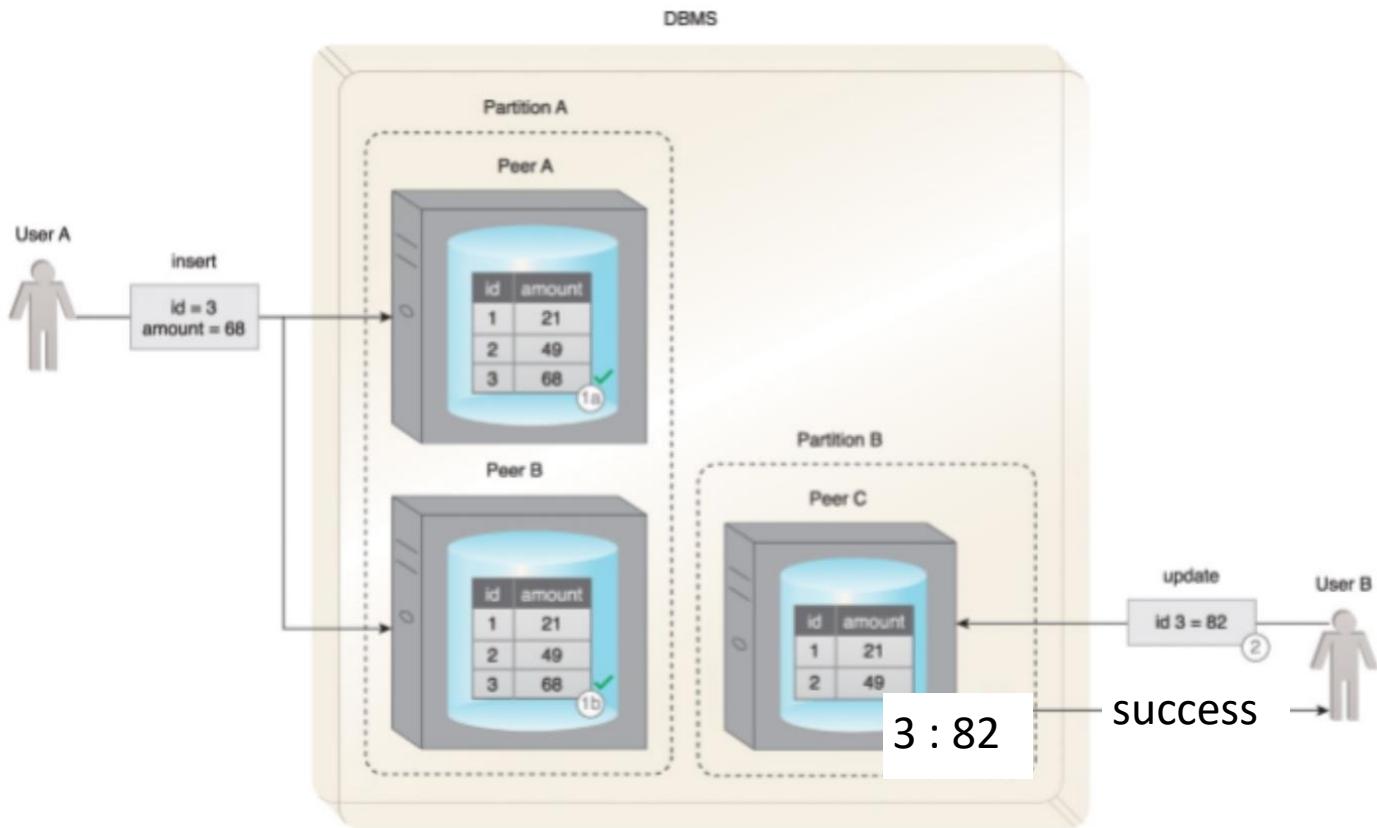
- A request from a user to a peer always responds with a success or failure within a reasonable time.
- Say Peer C is disconnected from the network, then it has 2 options
 - Available: Respond with a failure when any request is made, or
 - Unavailable: Wait for the problem to be fixed before responding,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

P: Partition tolerance (1)



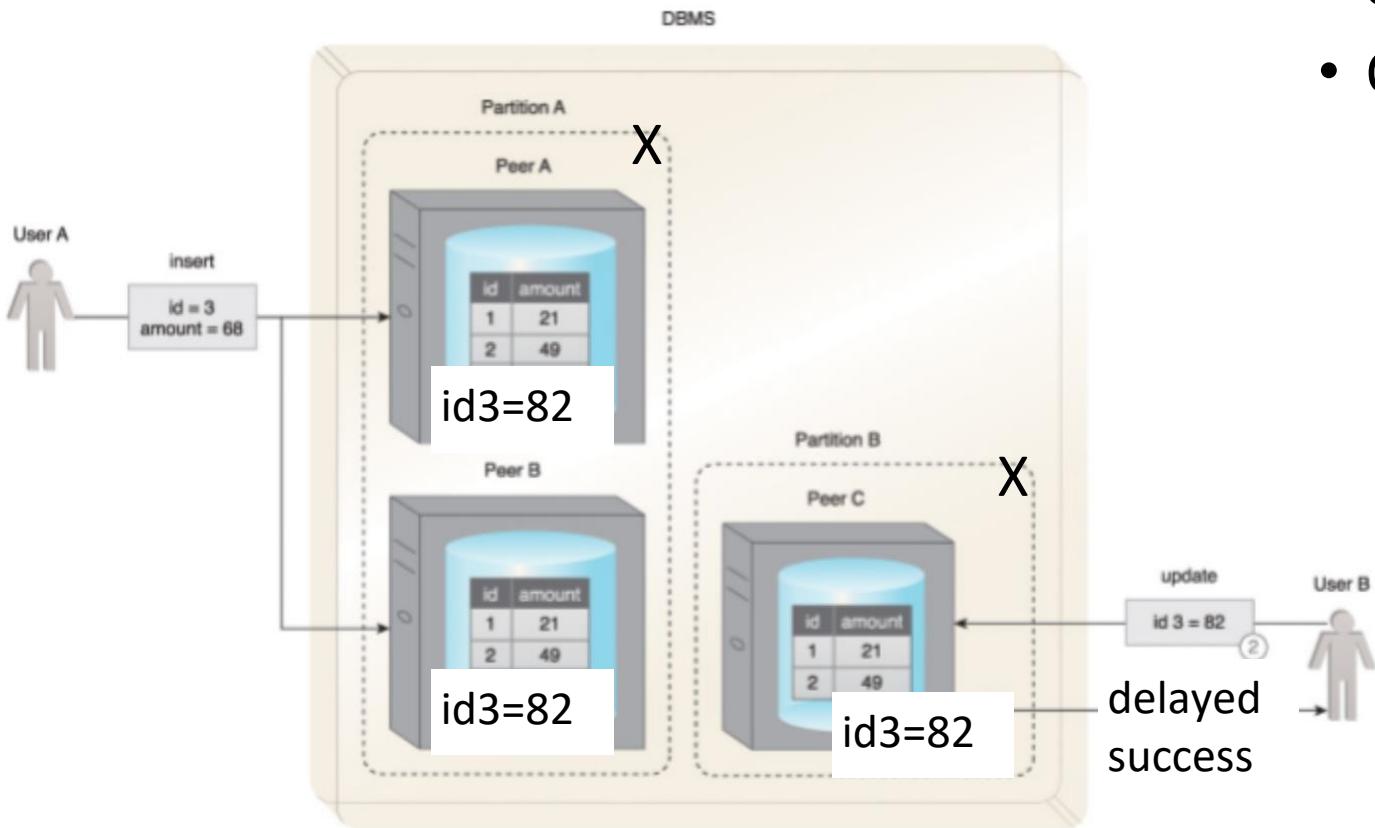
- A network partition happens and user wants to update peer C
- Option 1:
 - Any access by user on peer C leads to failure message response
 - System is available because it comes back with a response
 - There is consistency because user's update is not processed
 - But there is no partition tolerance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

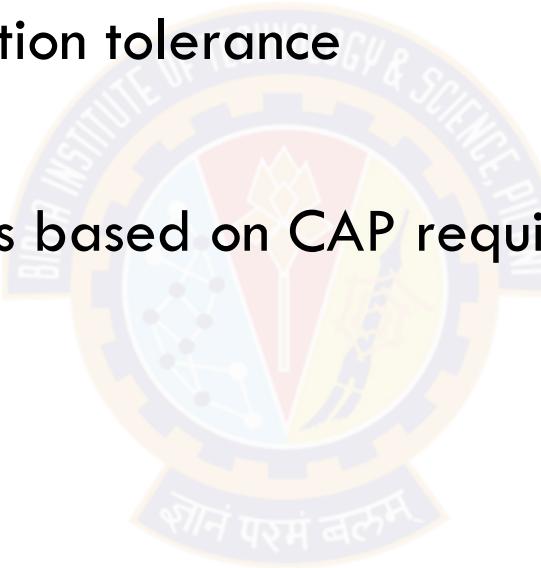
P: Partition tolerance (3)



- 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

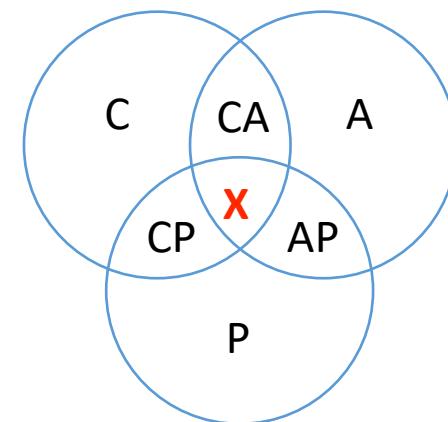
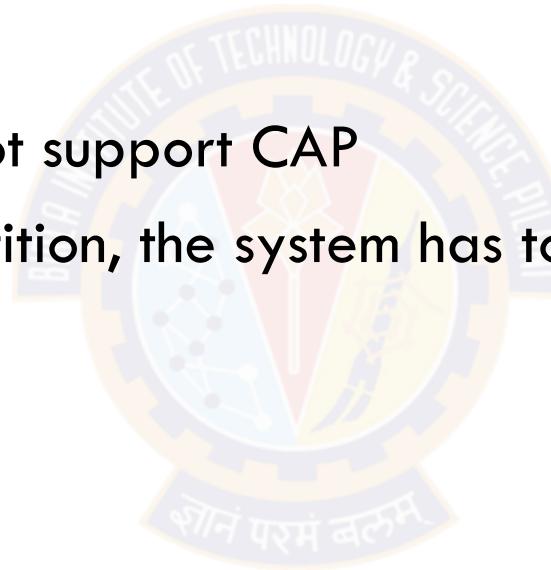
Topics for today

- 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.



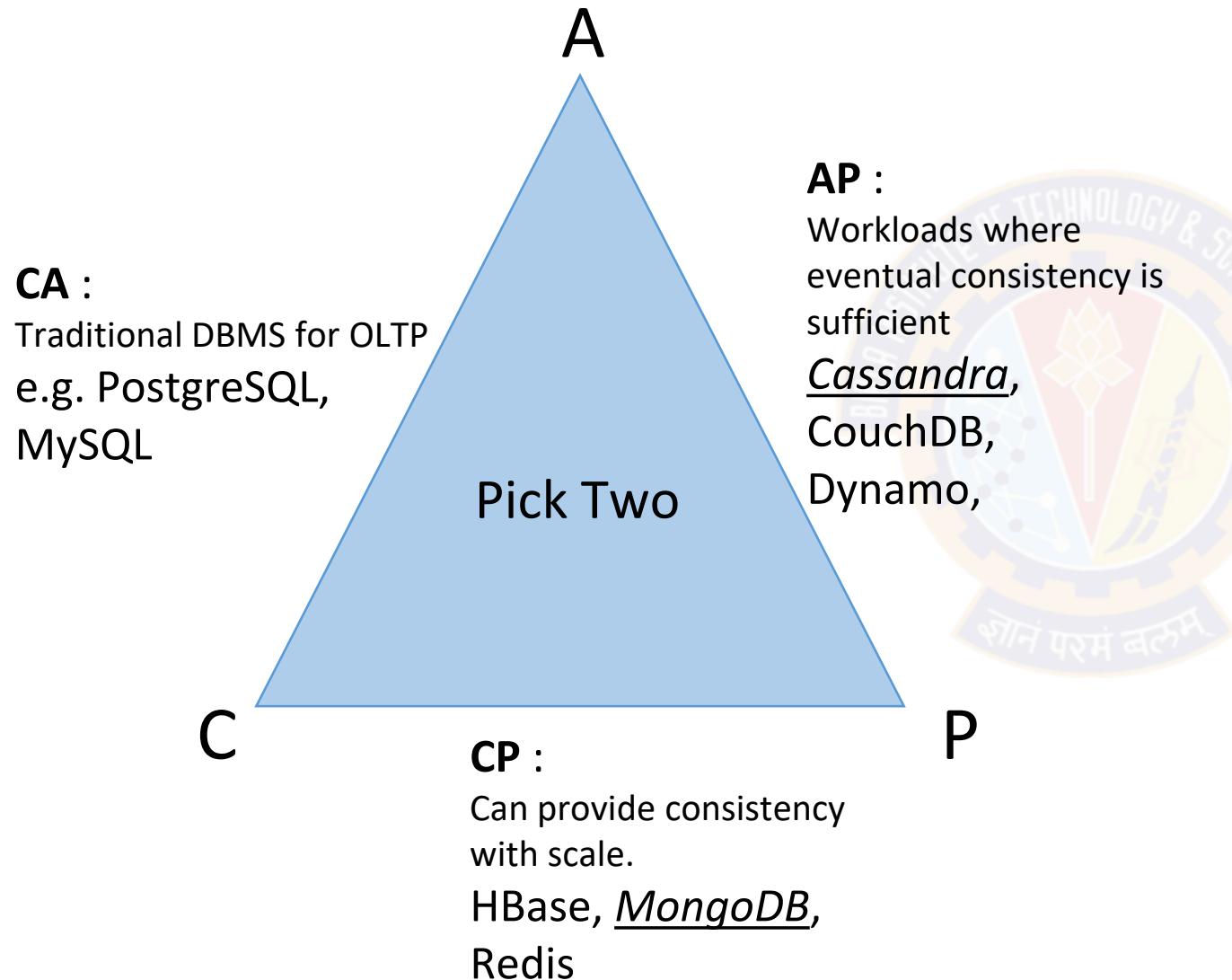
CAP Theorem – Historical note

- . 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/
- 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.

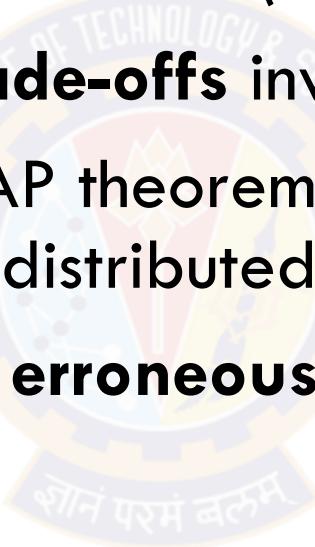
Database options



- 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

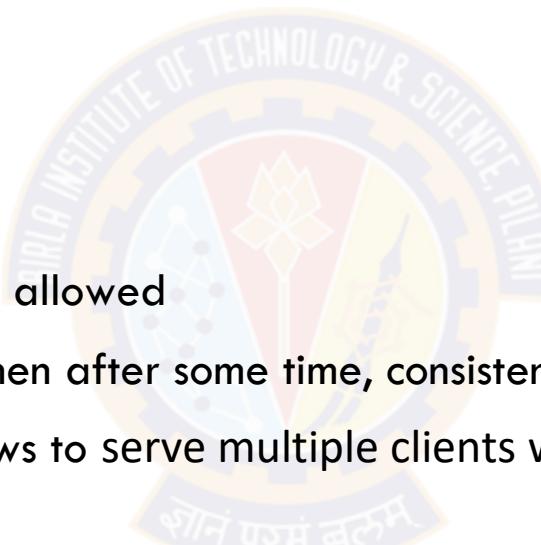
Importance of CAP Theorem

- 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**

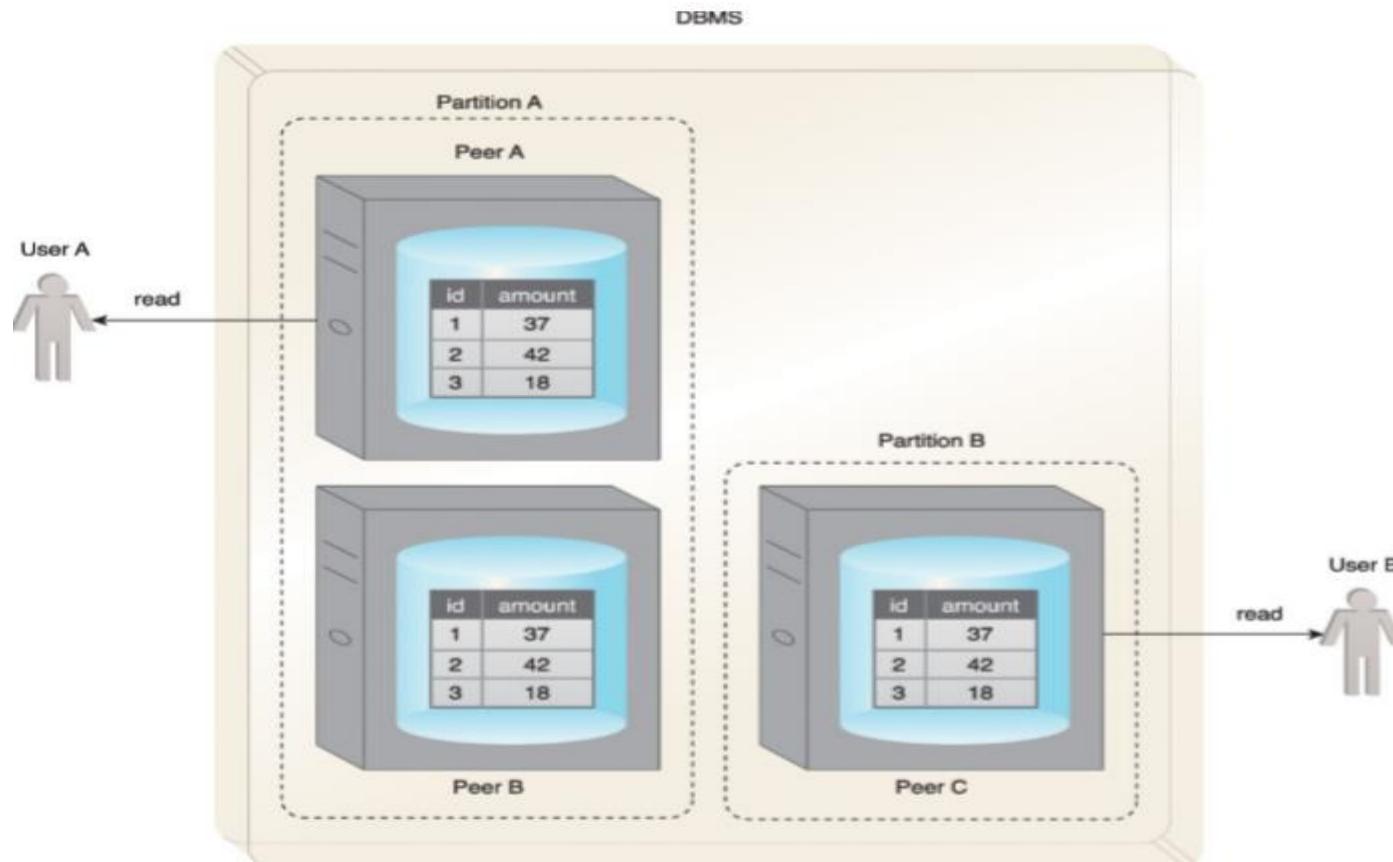


BASE

- 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



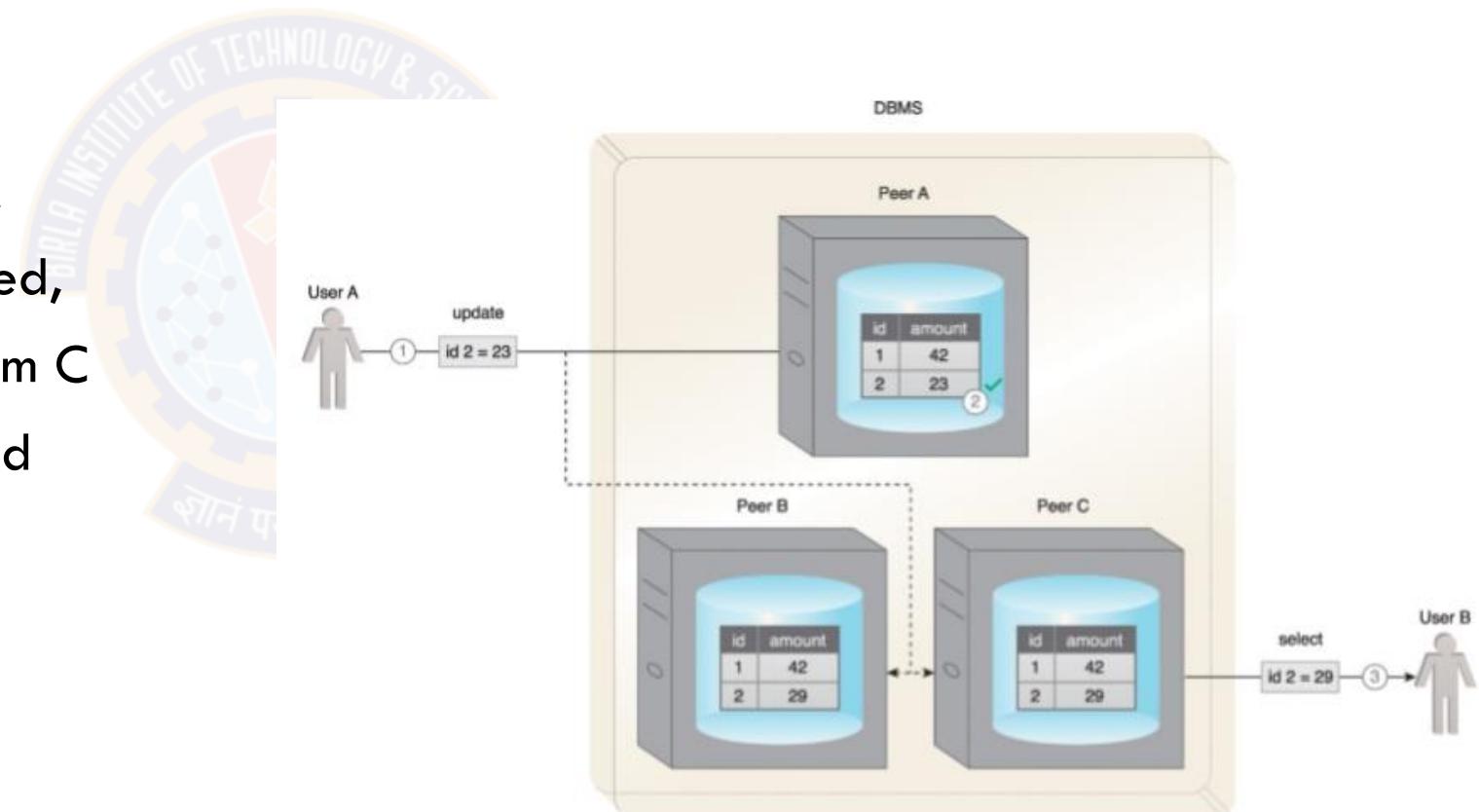
Database will always acknowledge a client's request, either in form of requested data or a failure notification

- 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

- User A updated record on node A
- Before the other nodes are updated,
User B requests same record from C
- Database is now in a soft state and
- Stale data is returned to User B



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

BASE – Eventual Consistency

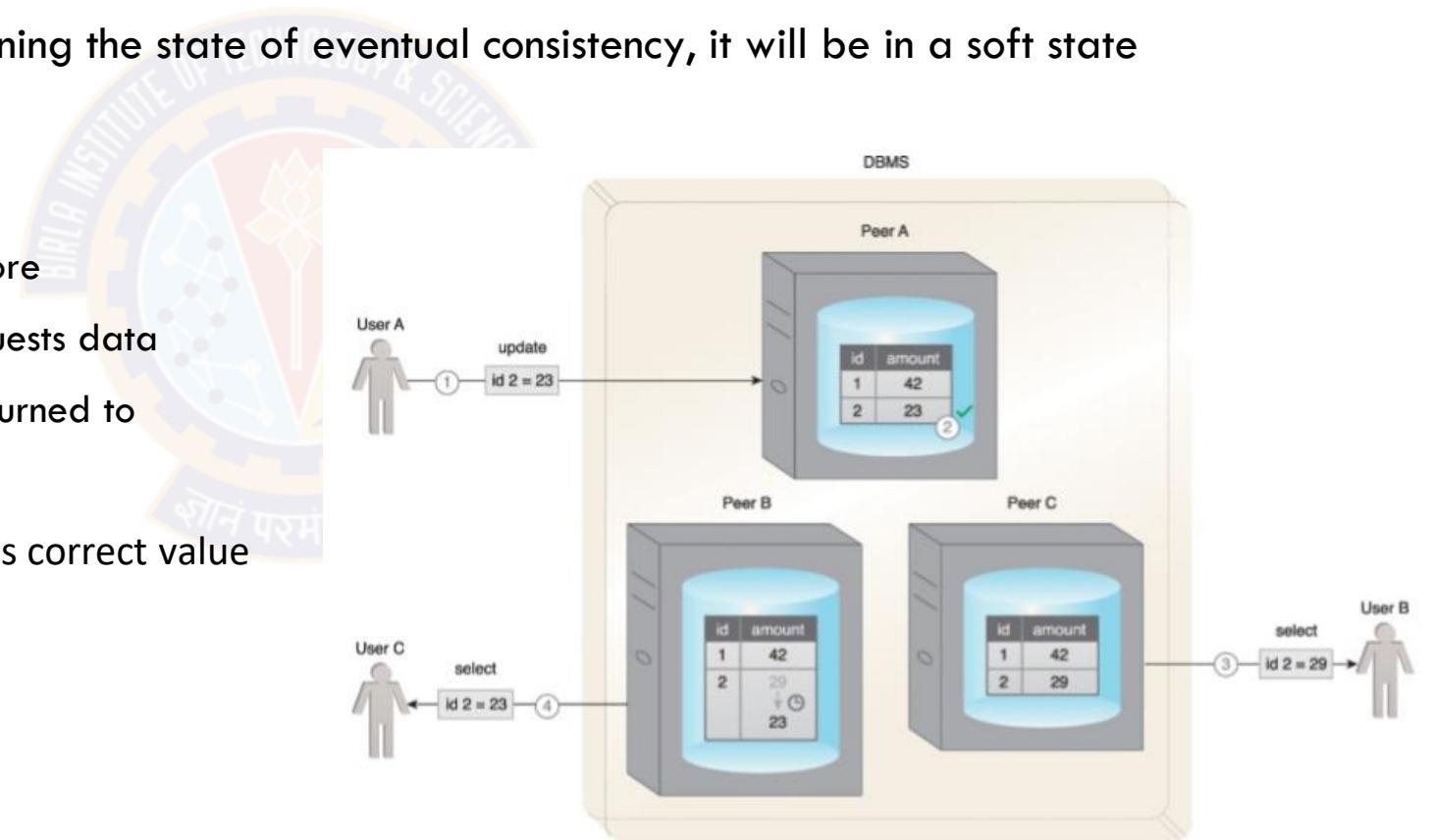
- State in which reads by different clients, immediately following a write to database, may not return consistent results
- Database only attains consistency once the changes have been propagated to all nodes
- While database is in the process of attaining the state of eventual consistency, it will be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

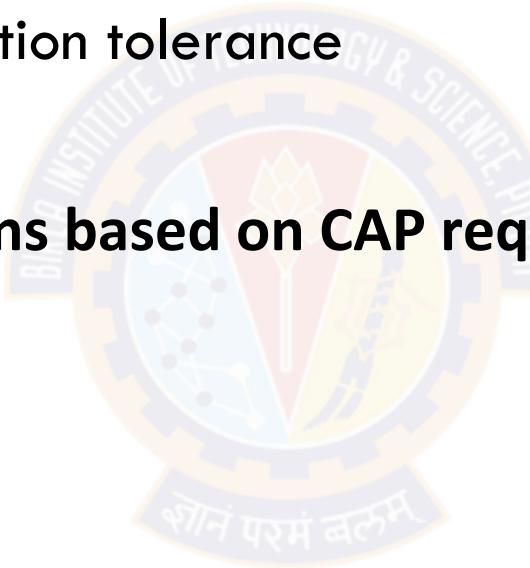
✓ Consistency is eventually attained, User C gets correct value



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
- 1st 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

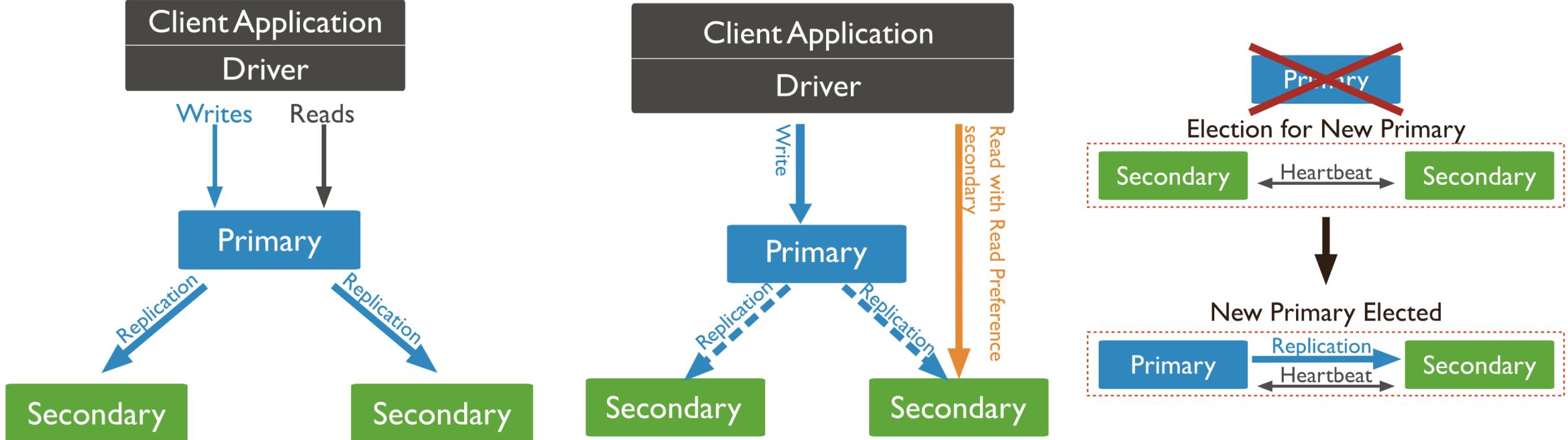
_id: "10006546"
listing_url: "https://www.airbnb.com/rooms/10006546"
name: "Ribeira Charming Duplex"
summary: "Fantastic duplex apartment with three bedrooms, lc
space: "Privileged views of the Douro River and Ribeira squa
description: "Fantastic duplex apartment with three bedrooms
neighborhood_overview: "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}
])
```

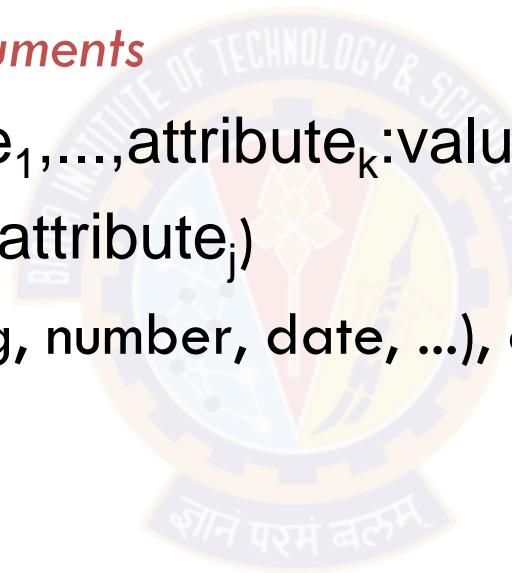
MongoDB

- 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* = {attribute₁:value₁,...,attribute_k:value_k}
- *Attribute* = string (attribute_i≠attribute_j)
- *Value* = **primitive** value (string, number, date, ...), or a **document**, or an **array**
- *Array* = [value₁,...,value_n]
- 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

}

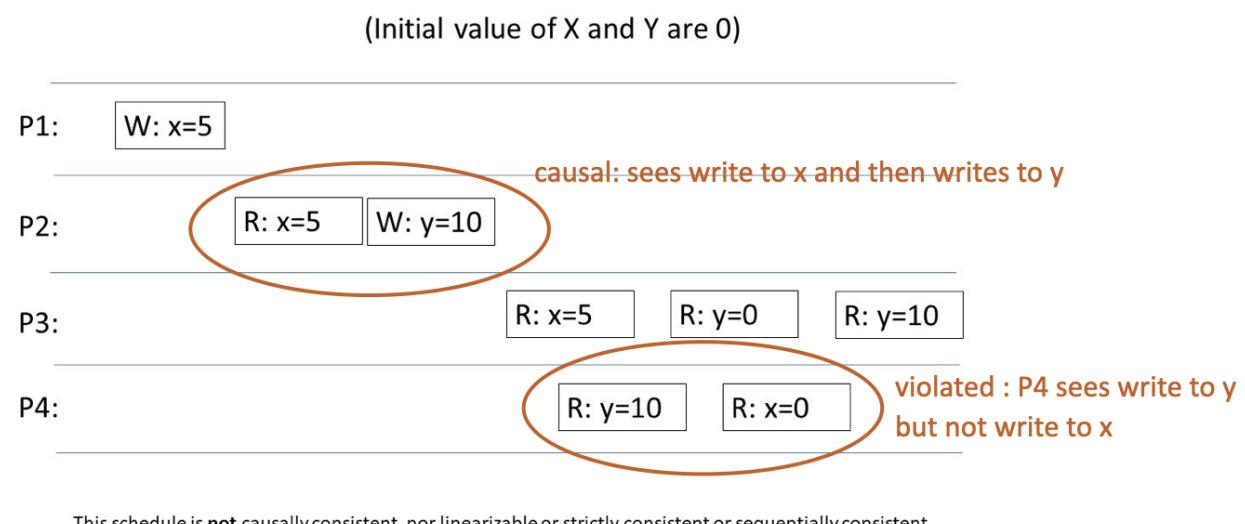
What is Causal Consistency (recap)

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.

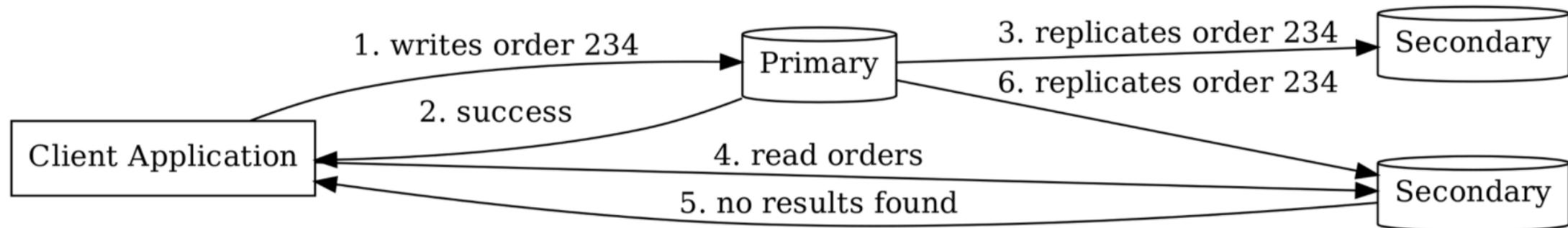
Monotonic writes Write operations that must precede other writes are executed before those other writes.

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

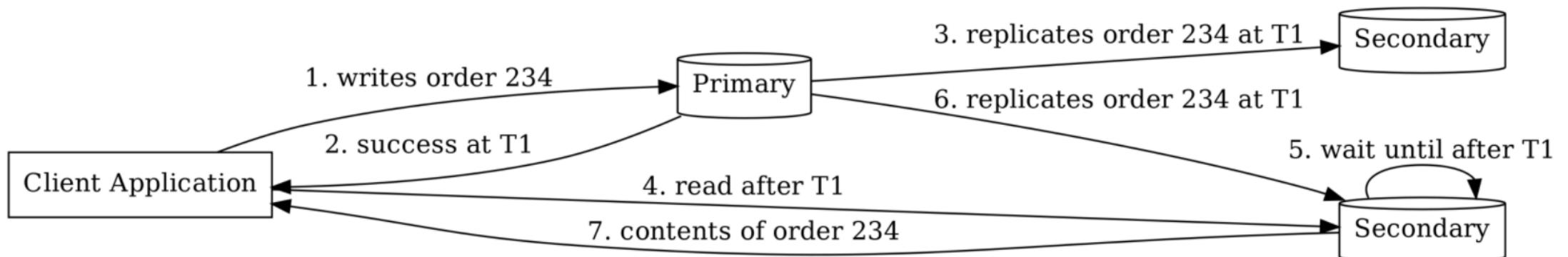


Example in MongoDB

- Case 1 : No causal consistency



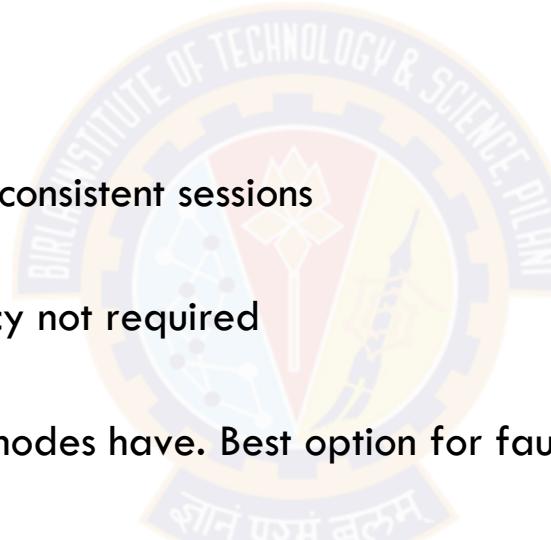
- Case 2: Causal consistency by making read to secondary wait



MongoDB “read concerns”

Local, available, majority, linearizable

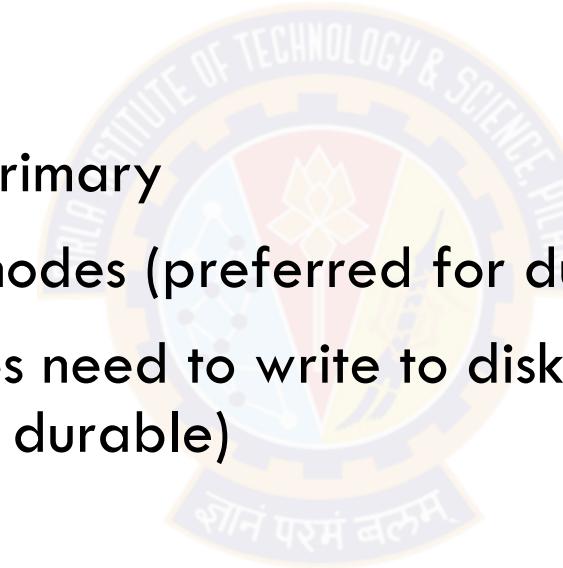
- local :
 - Client reads primary replica
 - Client reads from secondary in causally consistent sessions
- available:
 - Read on secondary but causal consistency not required
- majority :
 - If client wants to read what majority of nodes have. Best option for fault tolerance and durability.
- linearizable :
 - 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”

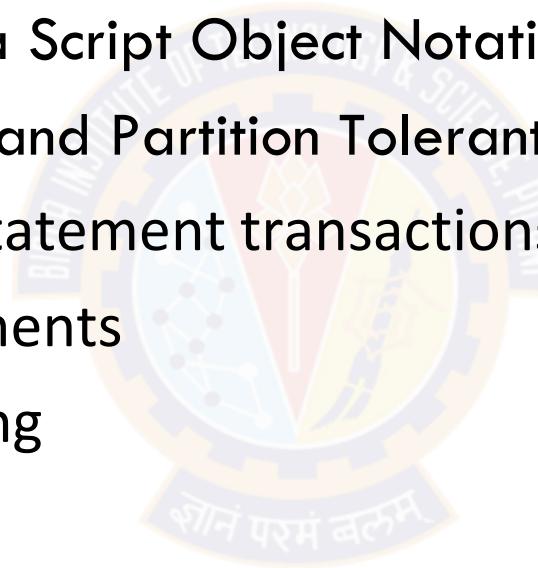
- how many replicas should ack
 - 1 - primary only
 - 0 - none
 - n - how many including primary
 - majority - a majority of nodes (preferred for durability)
- journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)
- timeout for write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



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

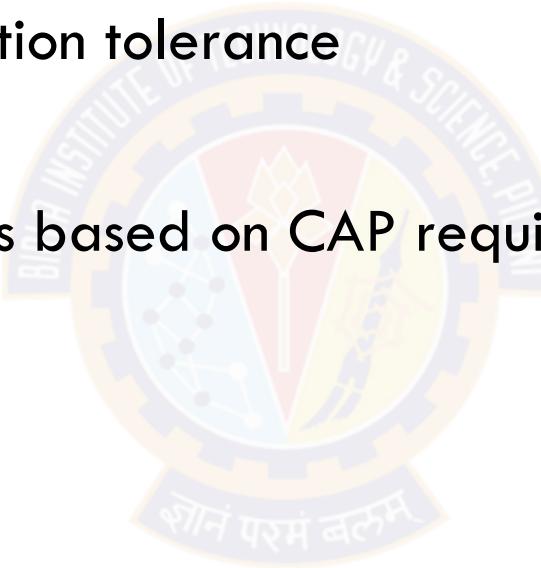
MongoDB in a nutshell

- 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

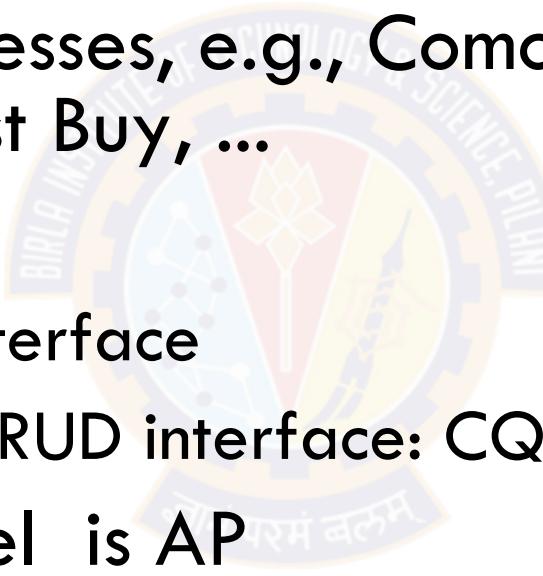


Topics for today

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

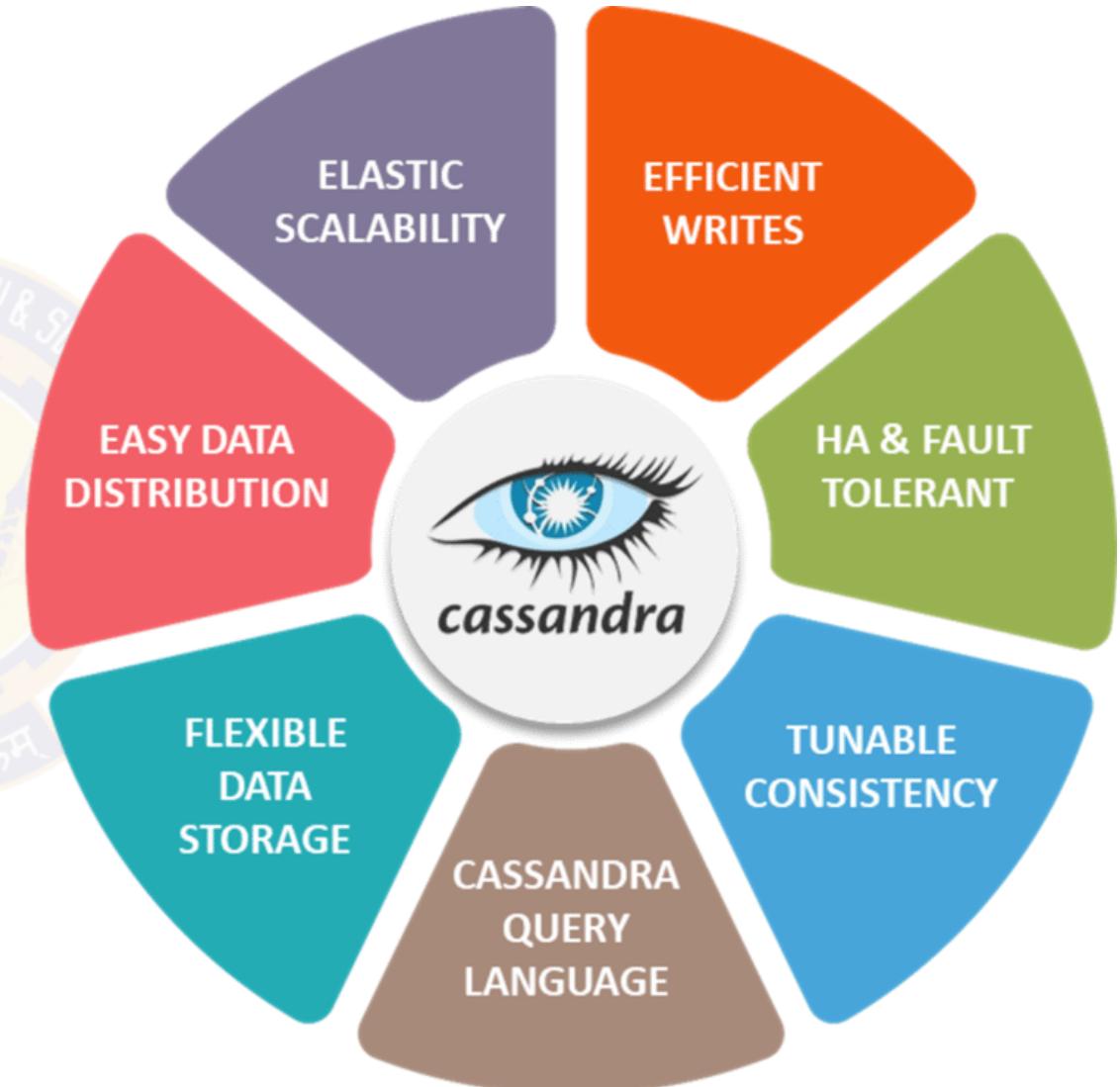


- Initially developed by Facebook
 - Open-sourced in 2008
- Used by 1500+ businesses, e.g., Comcast, eBay, GitHub, Hulu, Instagram, Netflix, Best Buy, ...
- Column-family store
 - Supports key-value interface
 - Provides a SQL-like CRUD interface: CQL
- BASE consistency model is AP
 - Gossip protocol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



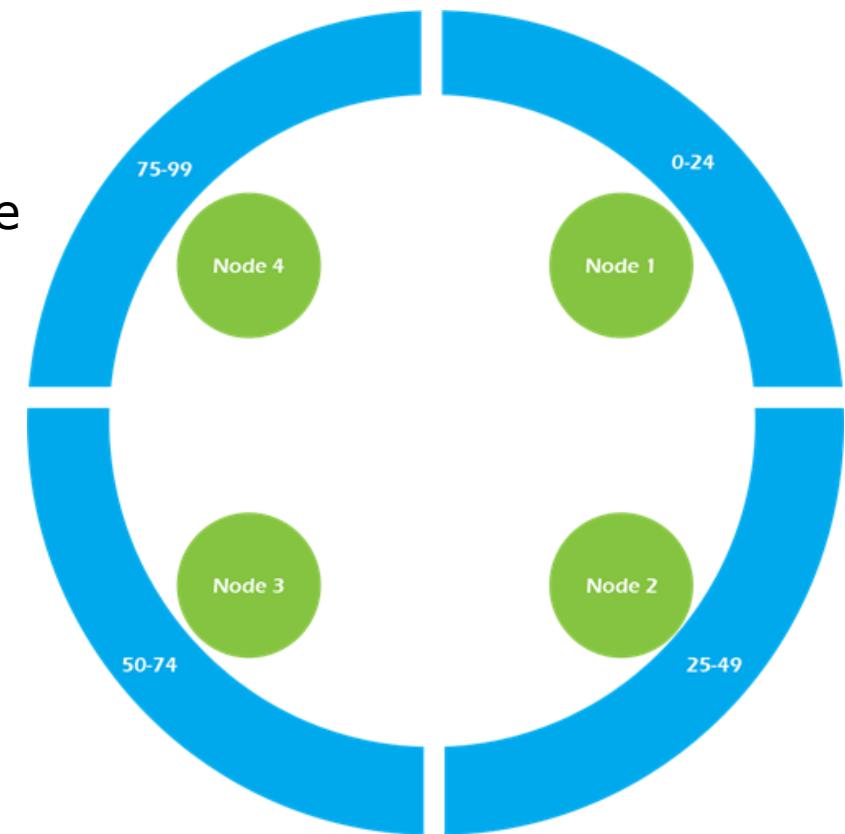
Cassandra Features

- 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

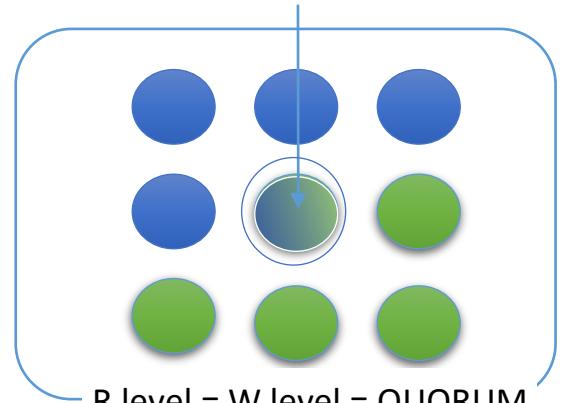
- Nodes in a Cassandra cluster owns a range of Keys (Tokens)
- Partition key is responsible for distributing data among nodes
- Partitions are groupings of data that will be co-located on storage of the node owning the key
- Partition keys are assigned to the nodes of the cluster
- Cassandra is organized 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>

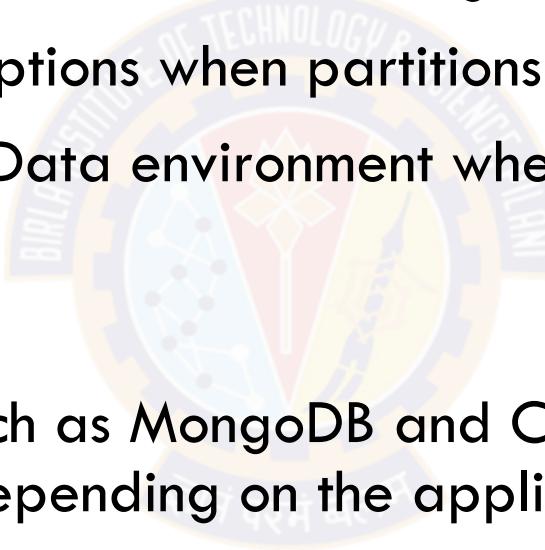
<https://cassandra.apache.org/doc/latest/architecture/guarantees.html#>

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

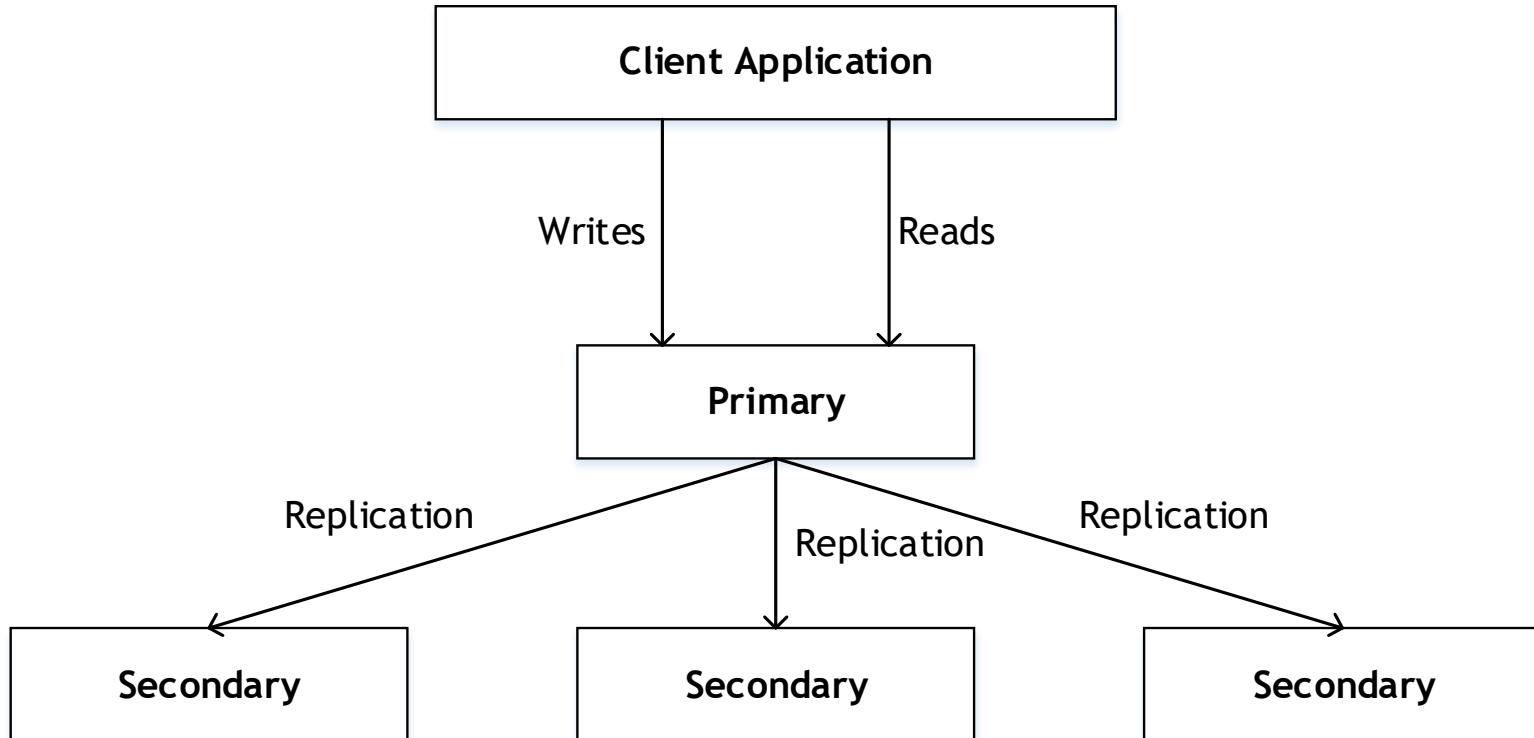
```
{  
  FirstName: John,  
  LastName: Mathews,  
  ContactNo: [+123 4567 8900, +123 4444 5555]  
}
```

Unique Identifier

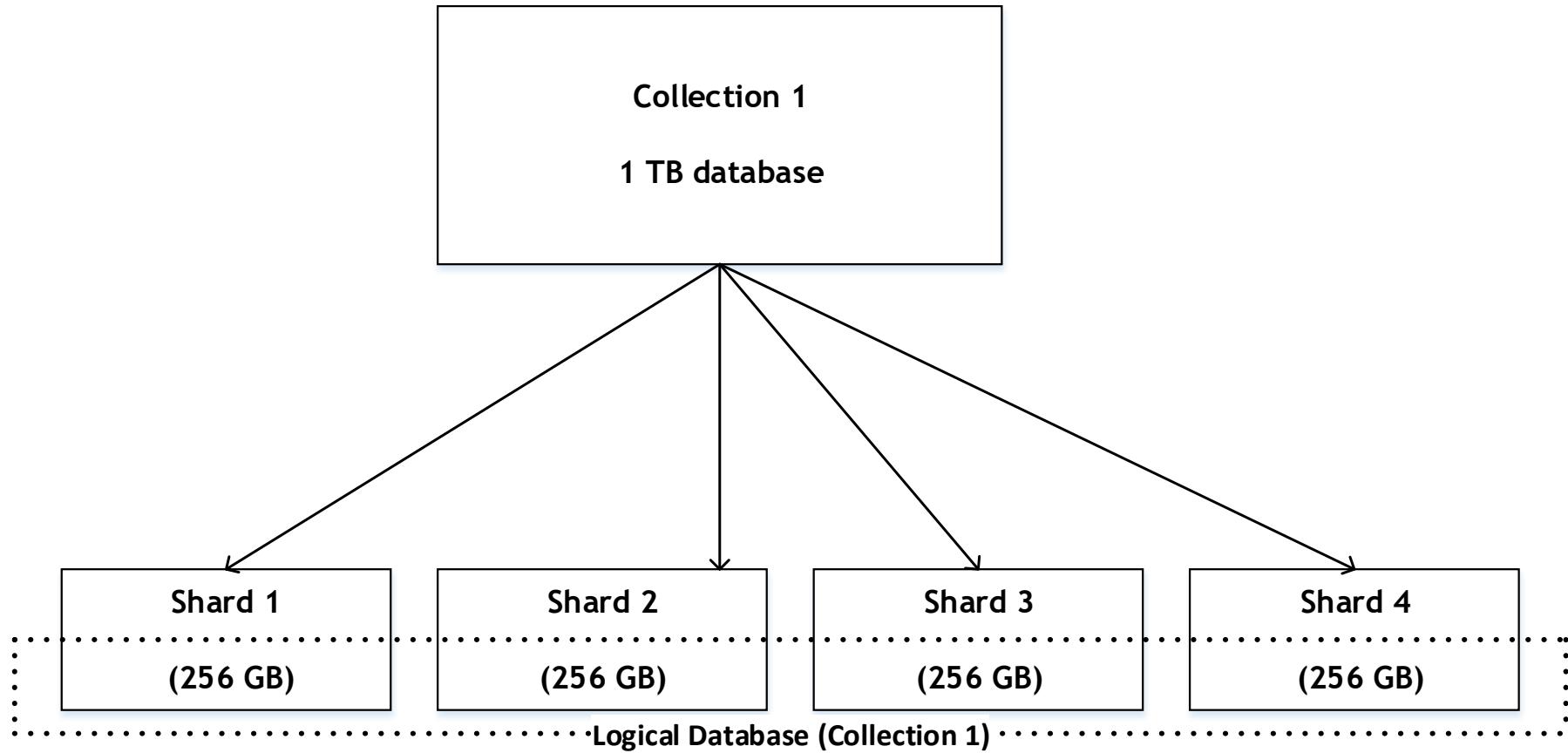
Each JSON document should have a unique identifier. It is the `_id` key.

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();
```

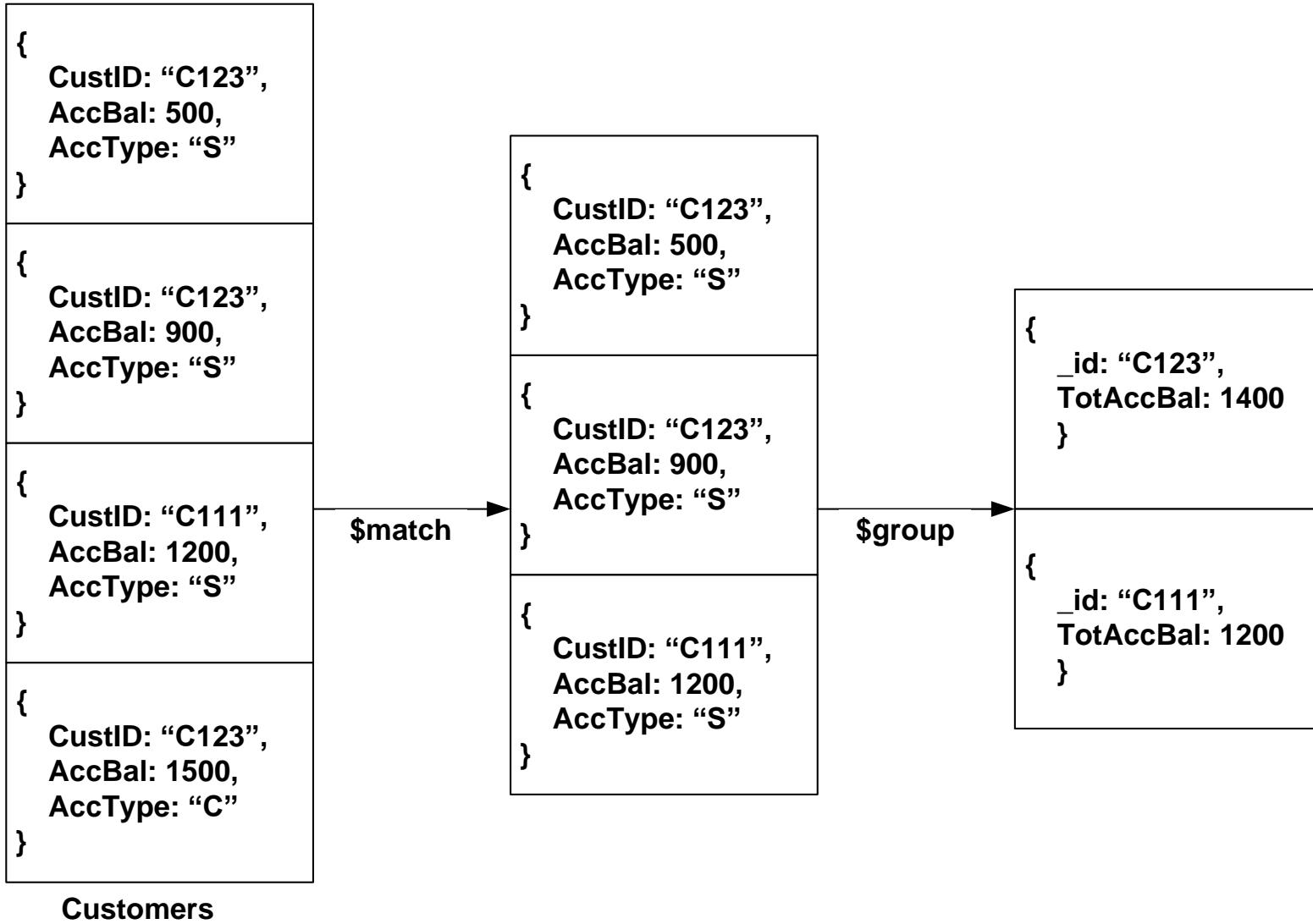
Find Method

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

Topics for today

- **Hadoop architecture overview**

- ✓ Components
- ✓ Hadoop 1 vs Hadoop 2

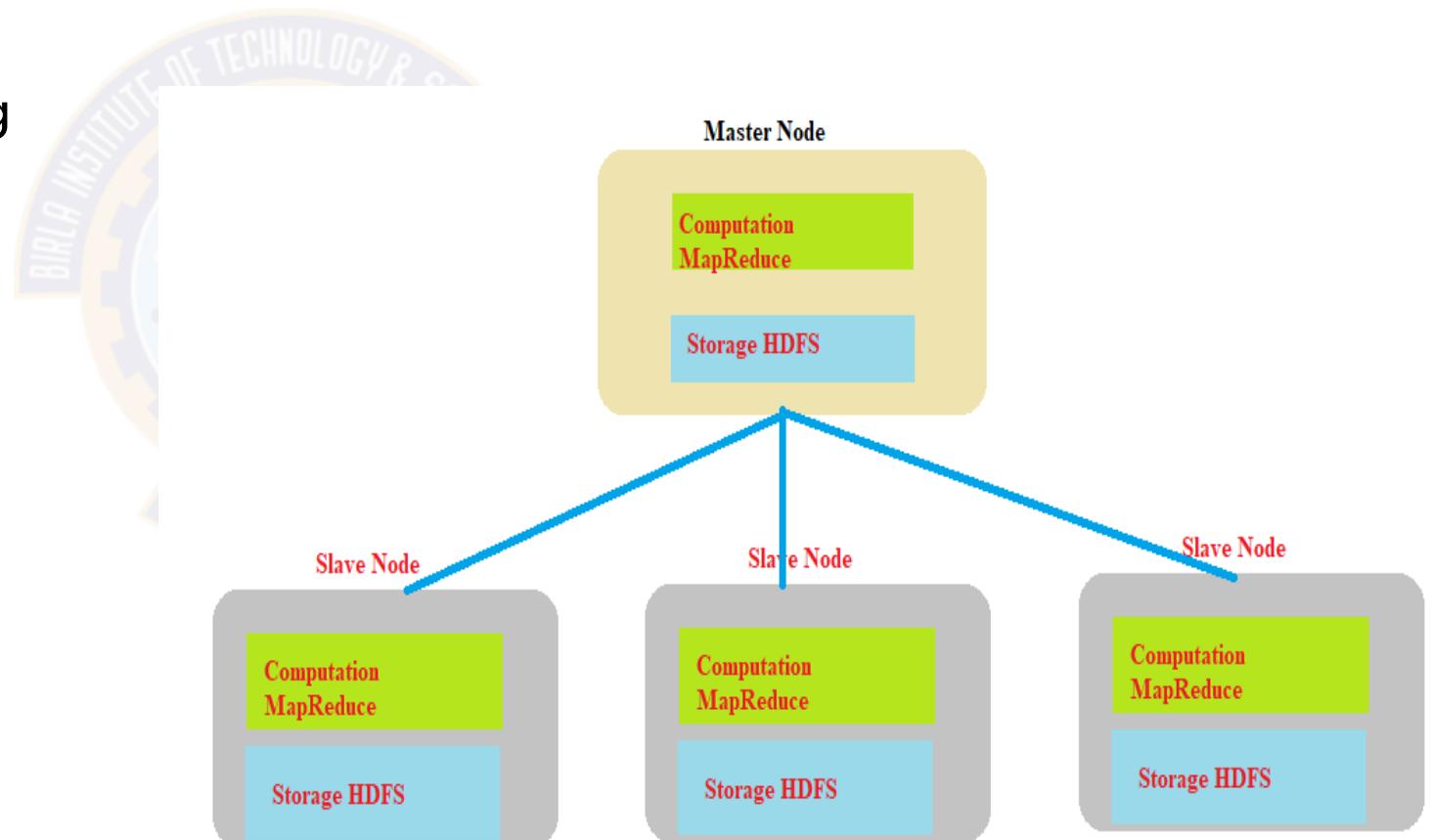
- **HDFS**

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



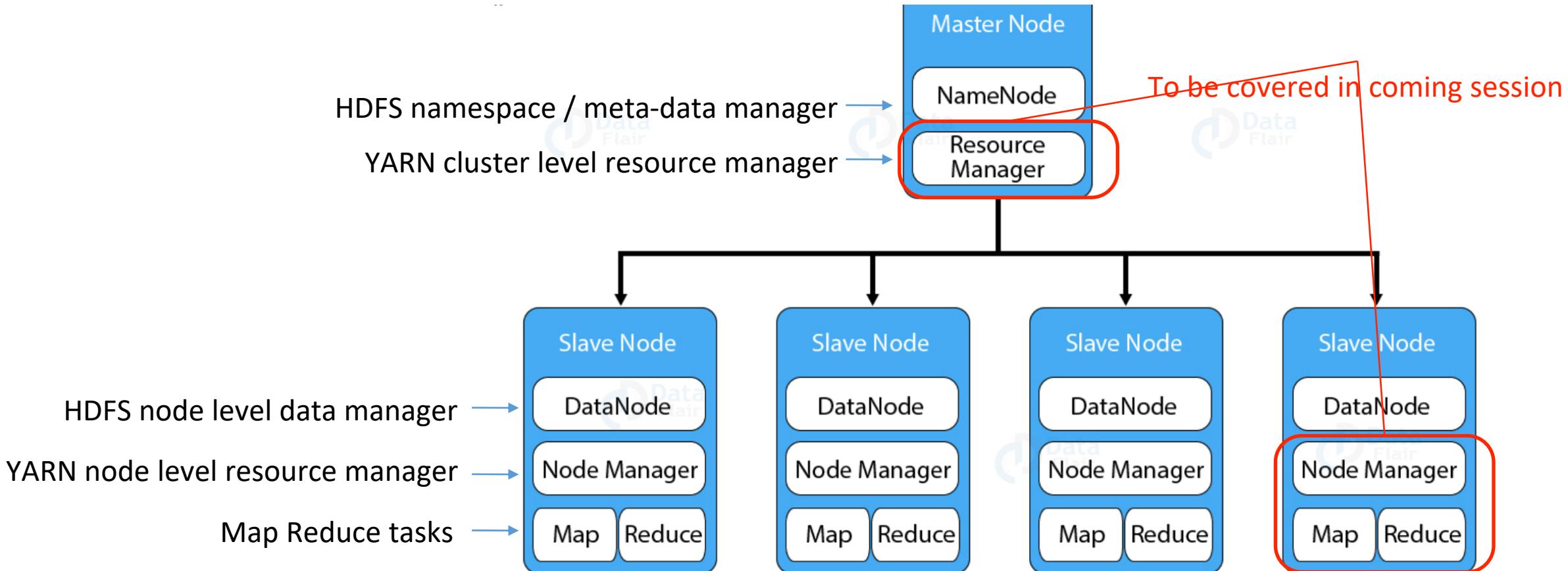
Hadoop - Data and Compute layers

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



Hadoop 2 - Architecture

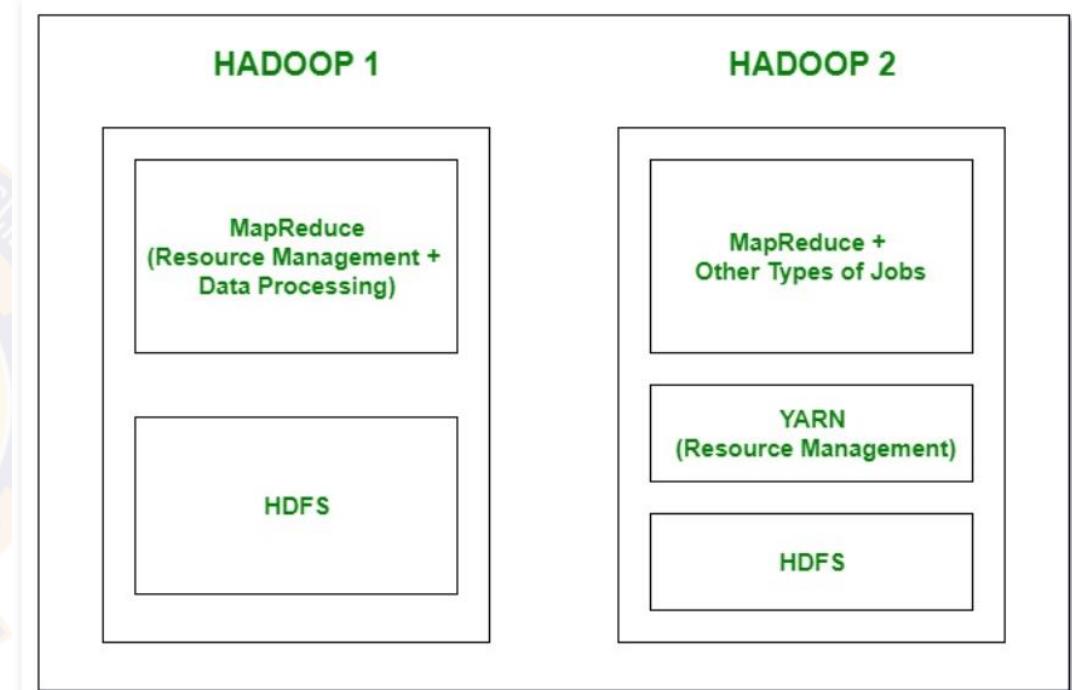
- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



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

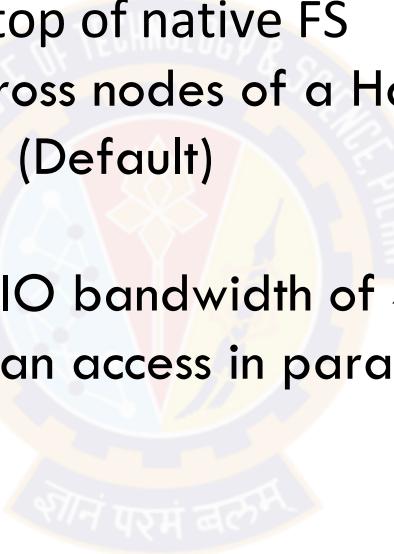
Topics for today

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



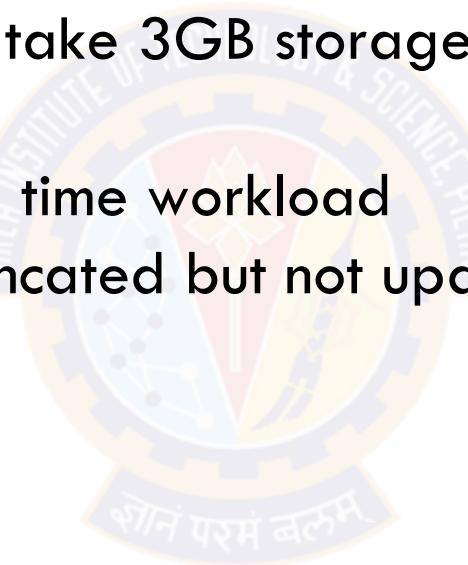
HDFS Features (1)

- 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



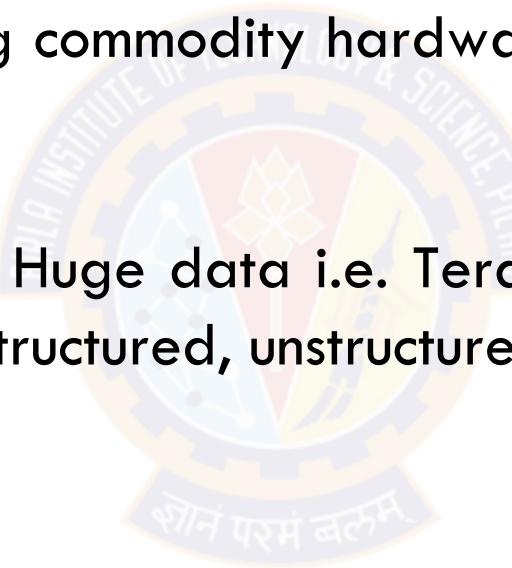
HDFS Features (2)

- 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



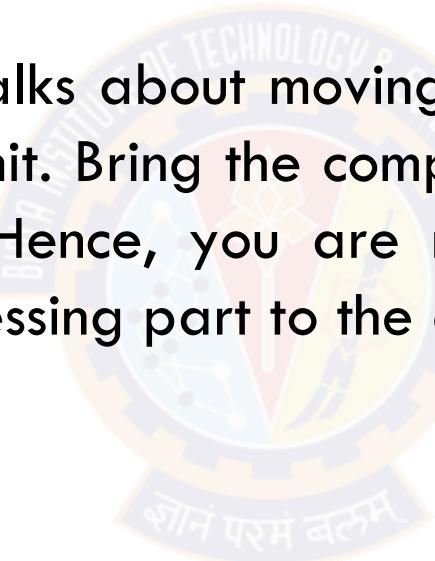
HDFS Features (3)

- 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.



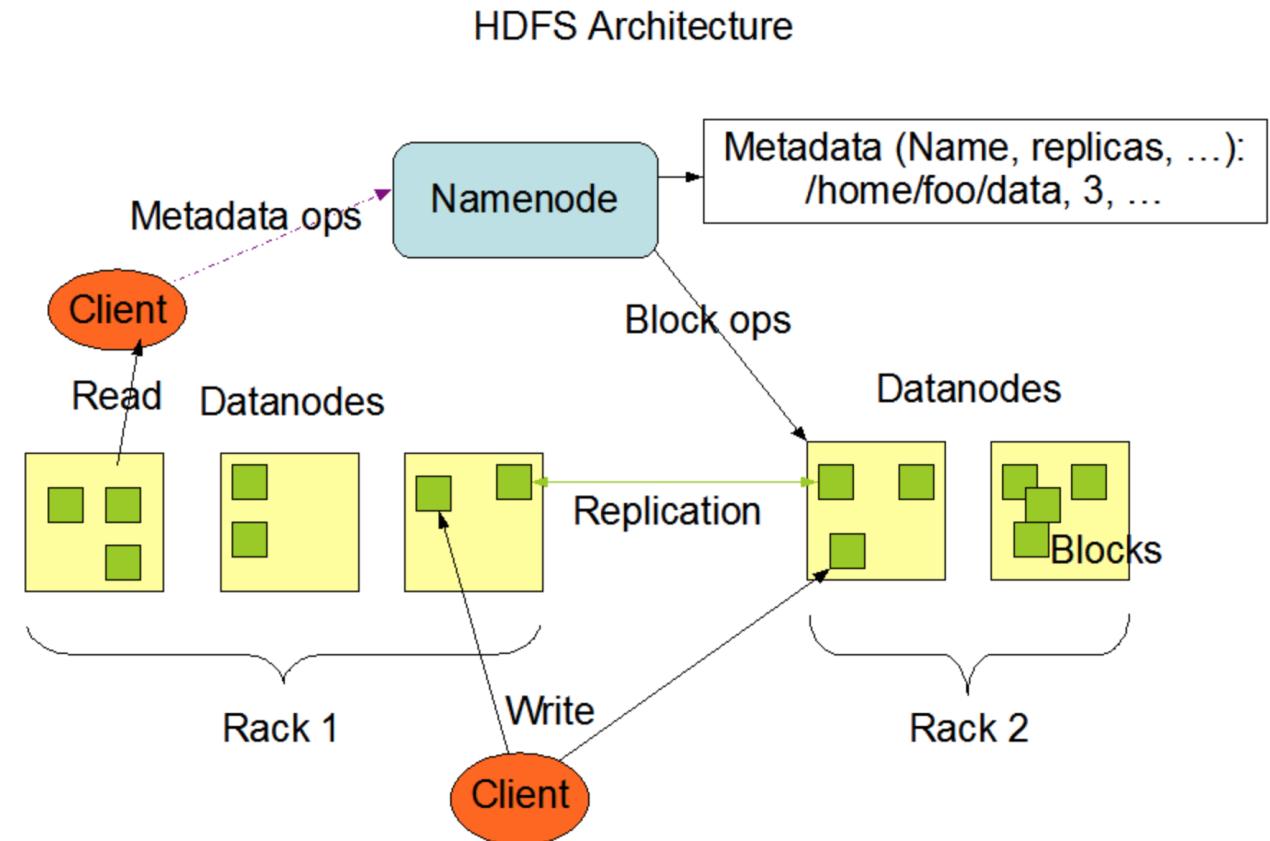
HDFS Features (4)

- 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.



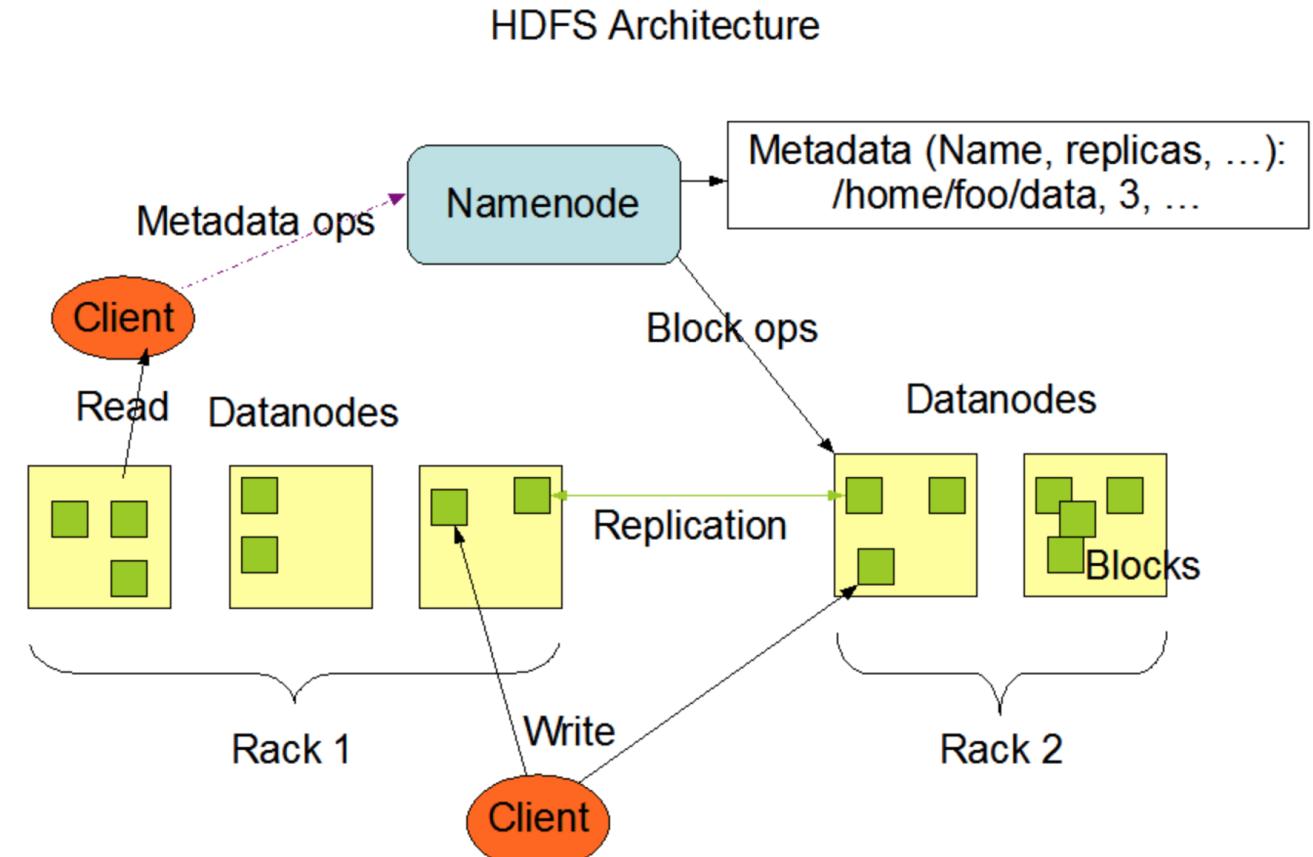
HDFS Architecture - Master node

- 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
```

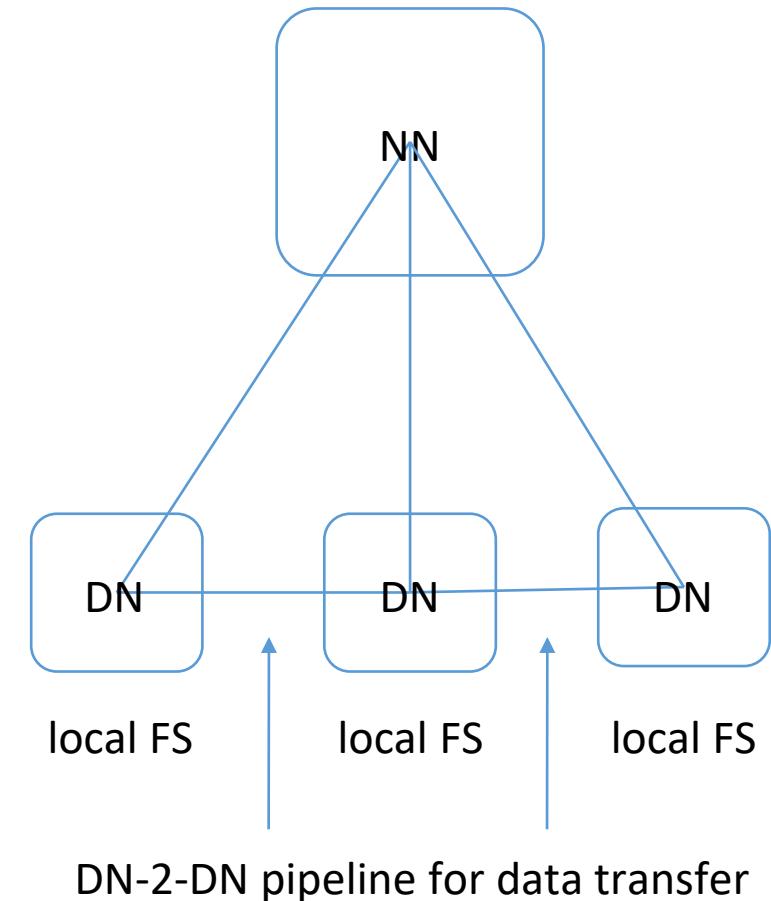
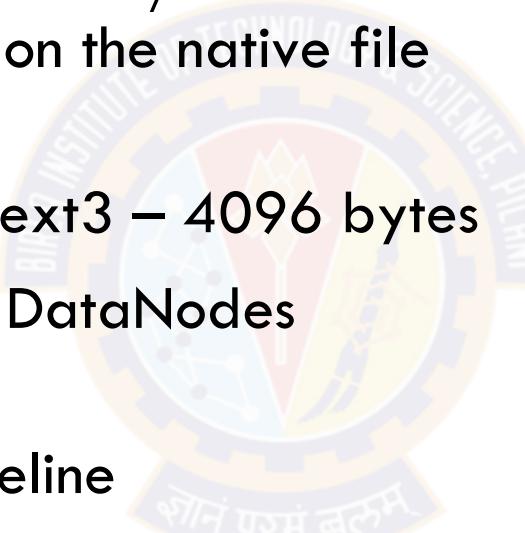
Namenode - What happens on start-up

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*]

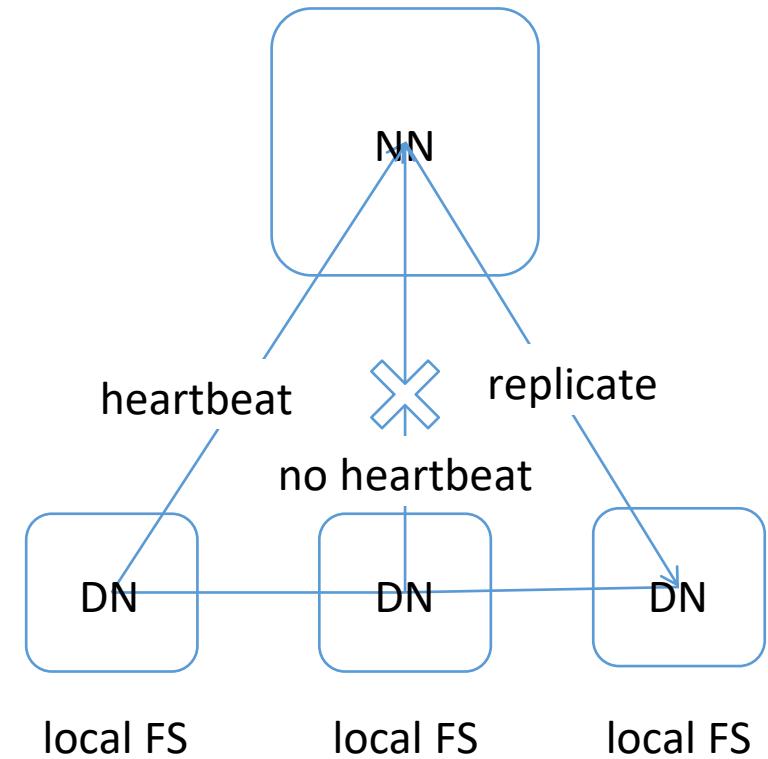
Functions of a DataNode (1)

- 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



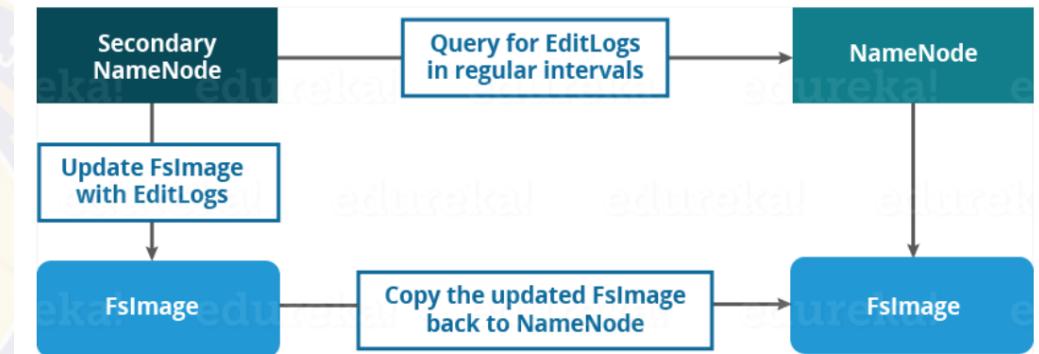
Topics for today

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



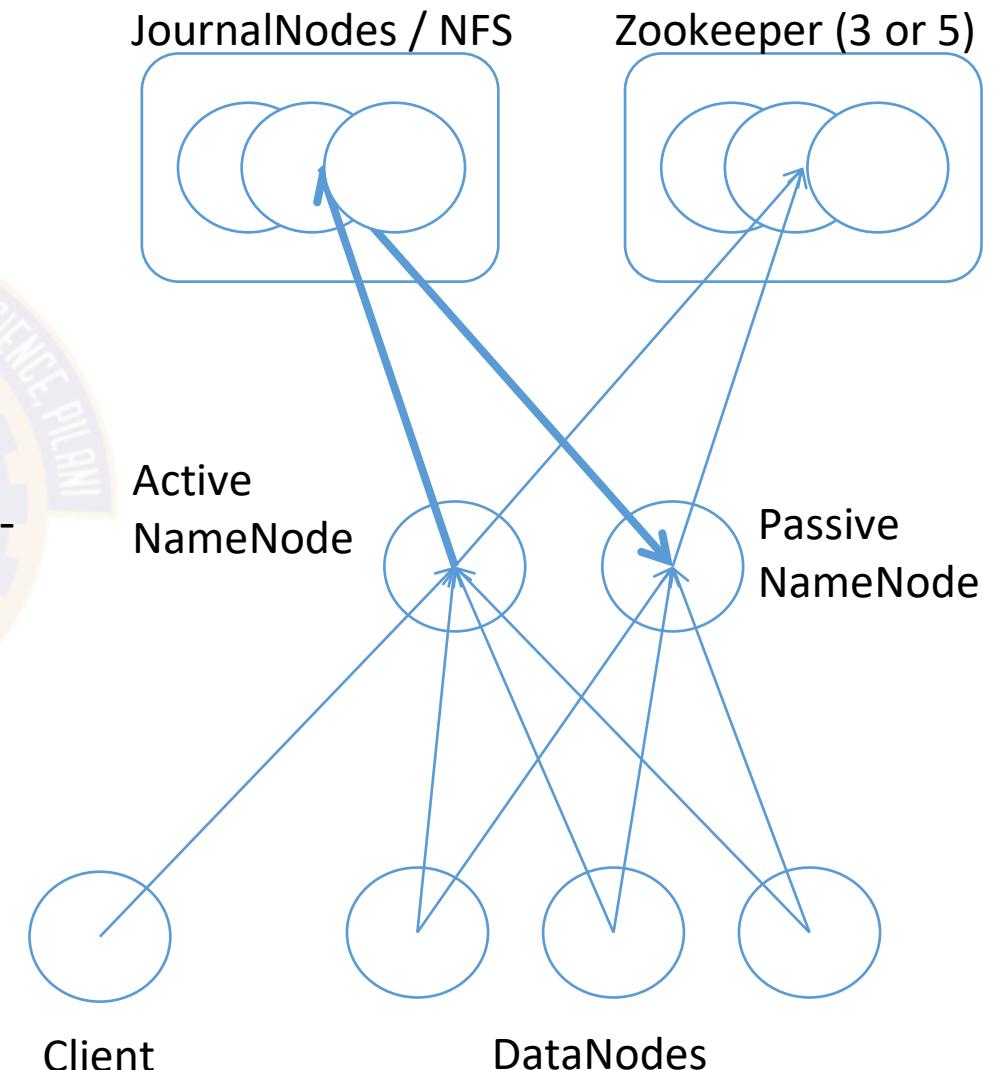
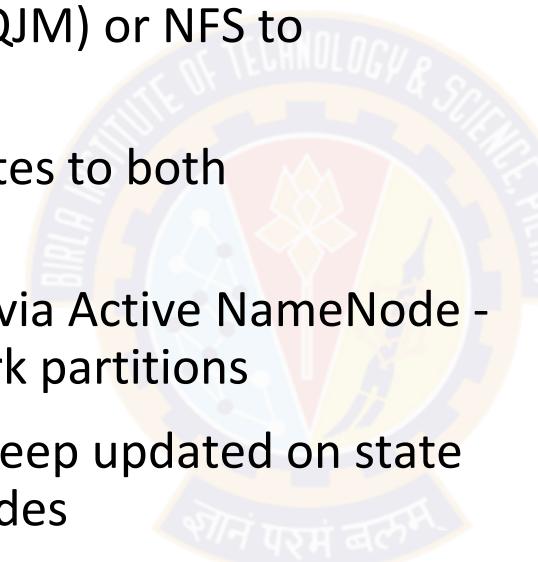
Hadoop 2: Introduction of Secondary NameNode

- 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

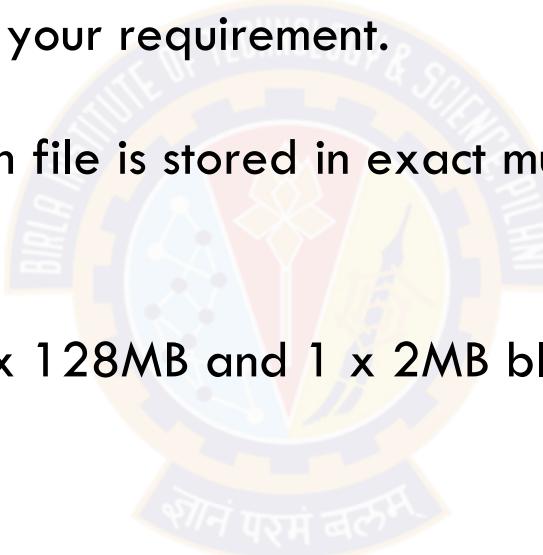
Topics for today

- 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

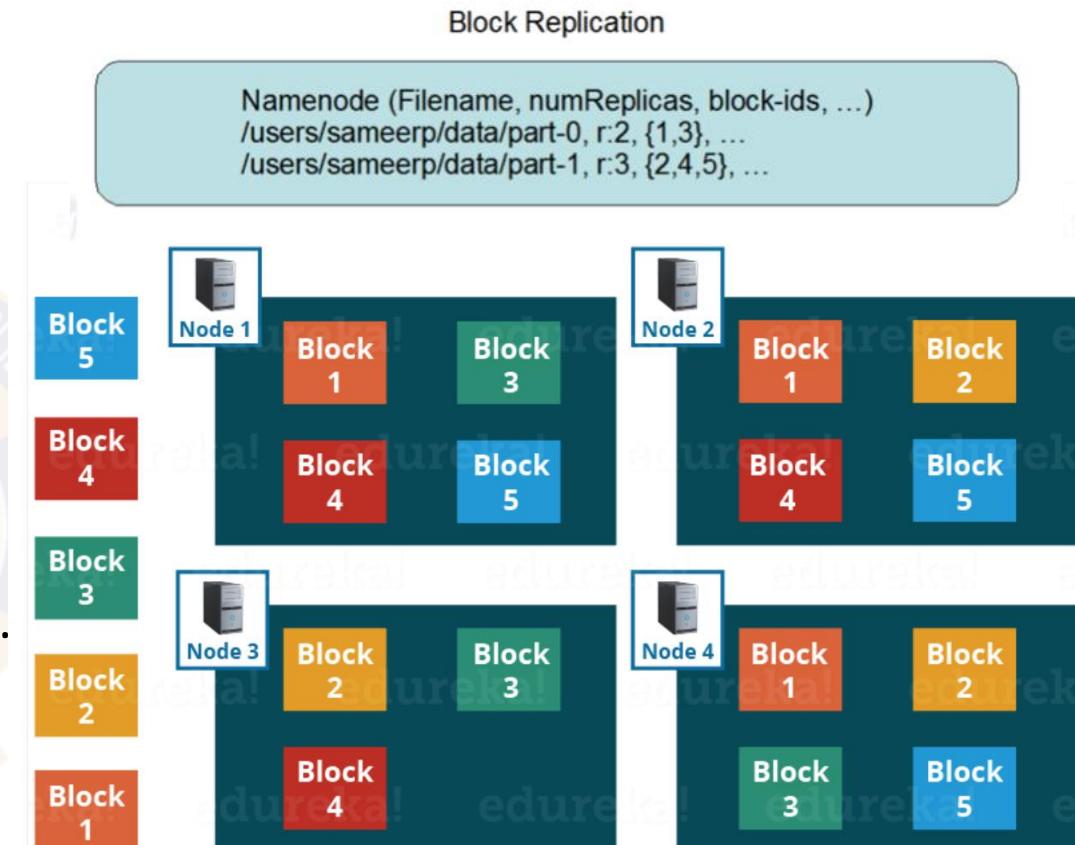
HDFS on local FS

- 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

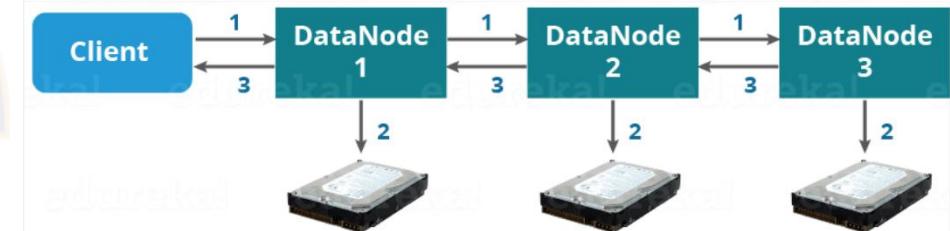


HDFS data writes

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

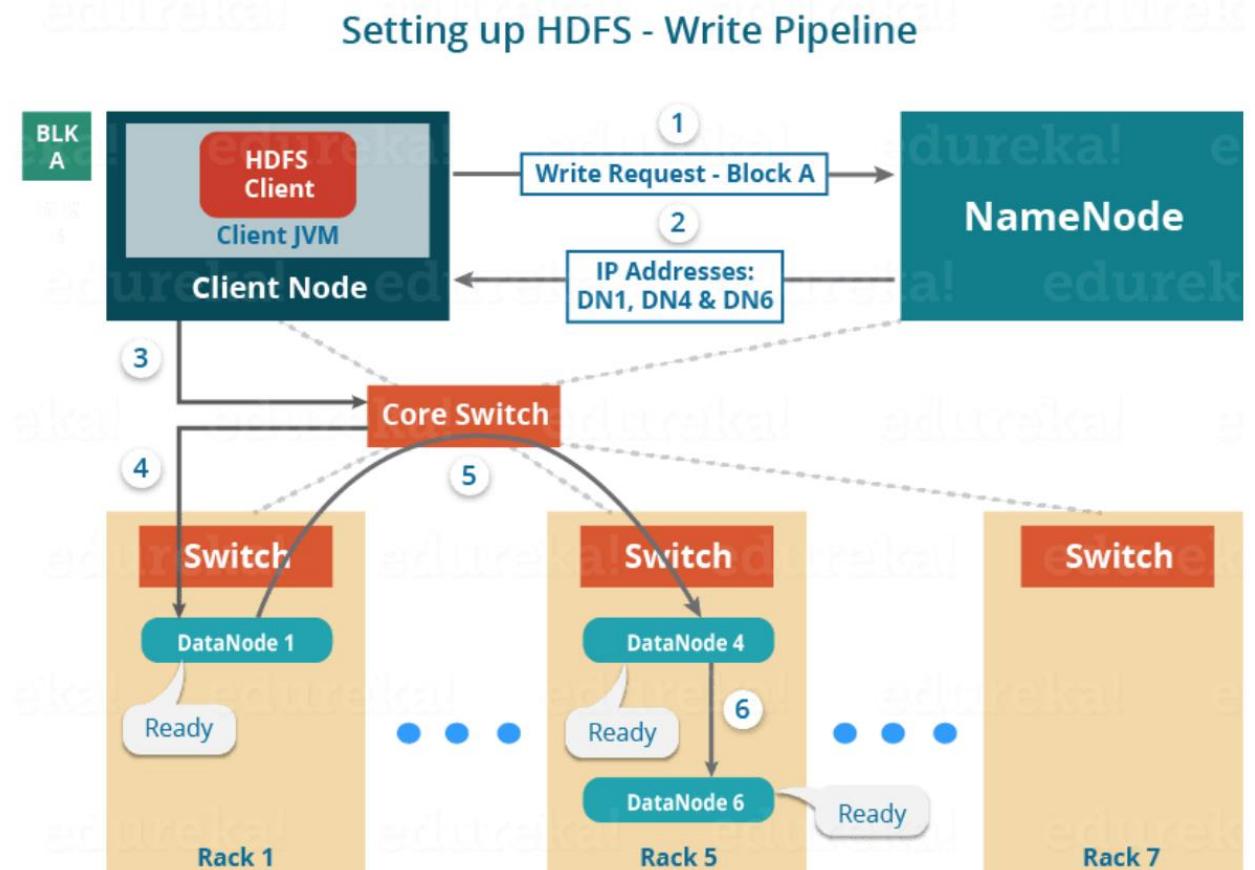
HDFS:

- 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)



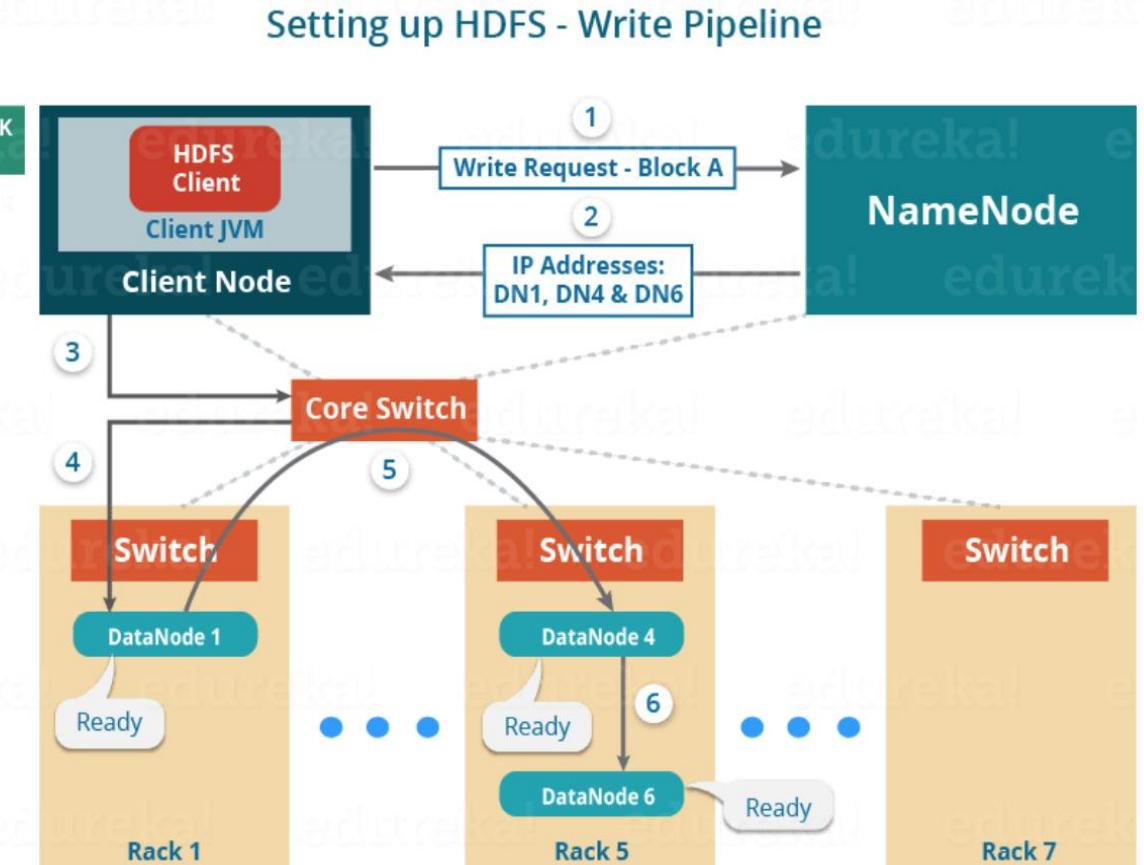
HDFS Write: Step 1. Setup pipeline

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

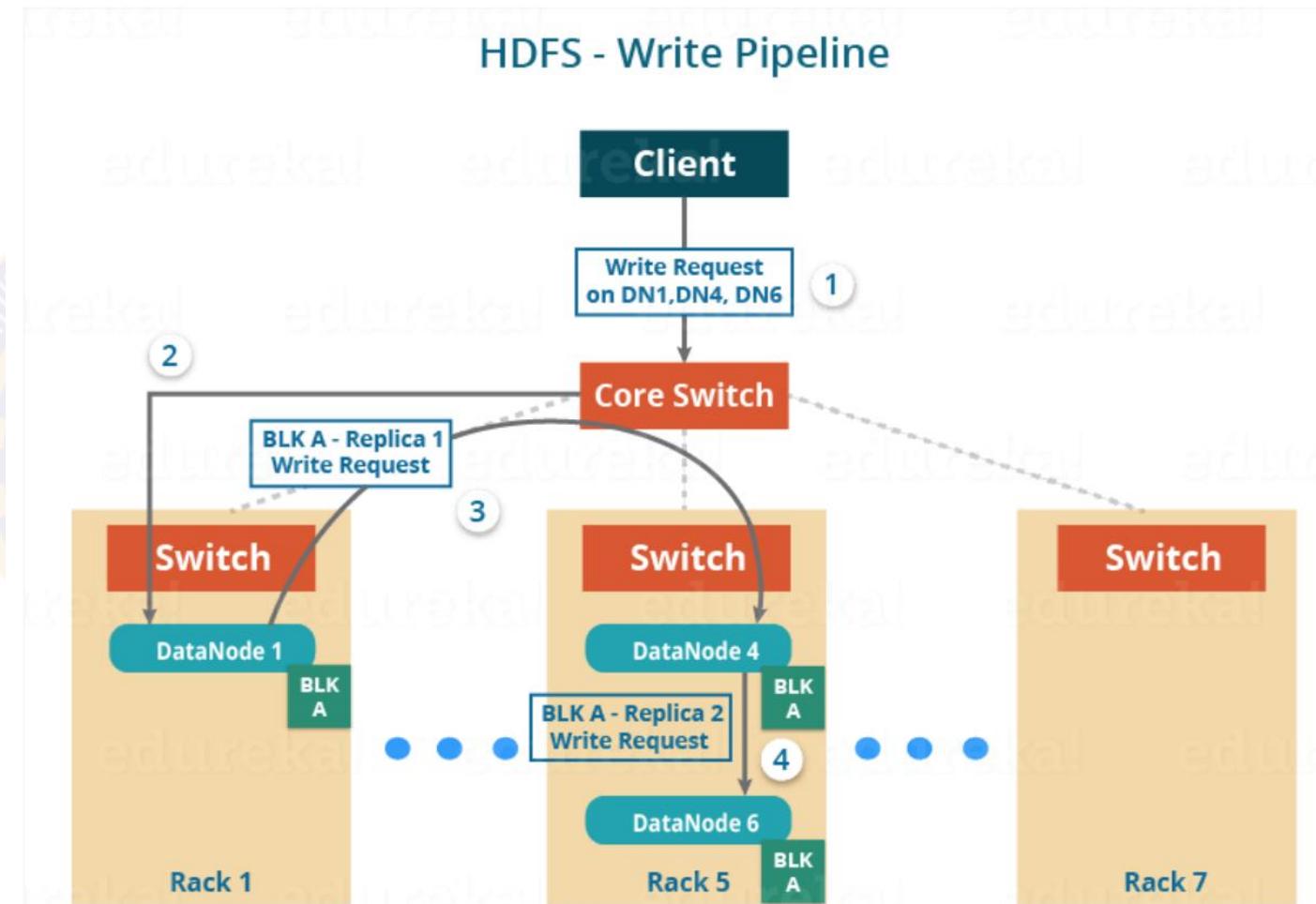
1. Client chooses the first DataNode (DN1) and will establish a TCP/IP connection.
2. Client informs DN1 to be ready to receive the block.
3. Provides IPs of next two DNs (4, 6) to DN1 for replication.
4. The DN1 connects to DN4 and informs it to be ready and gives IP of DN6. DN4 asks DN6 to be ready 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.

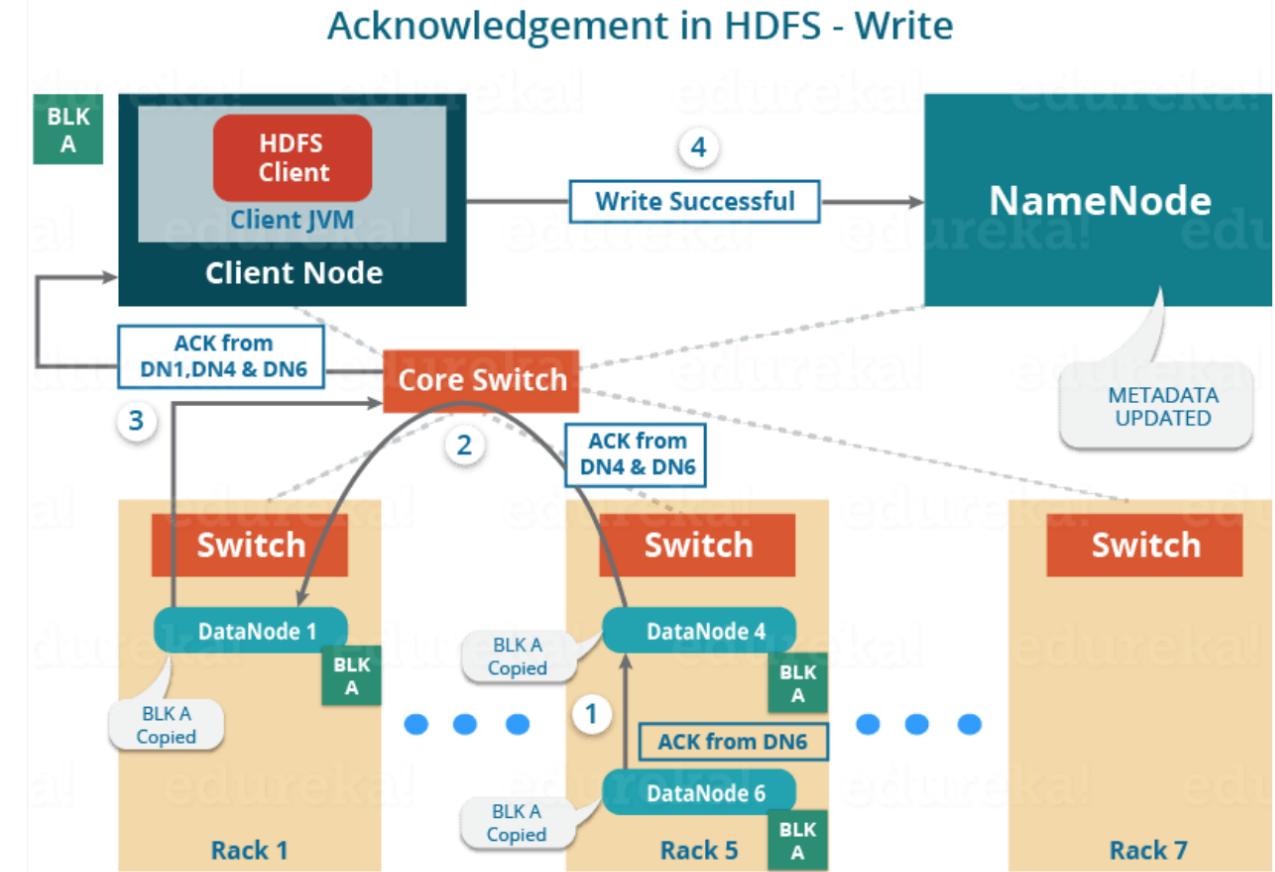
1. Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
2. Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
3. Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.



HDFS Write: Step 3. Shutdown pipeline / ack

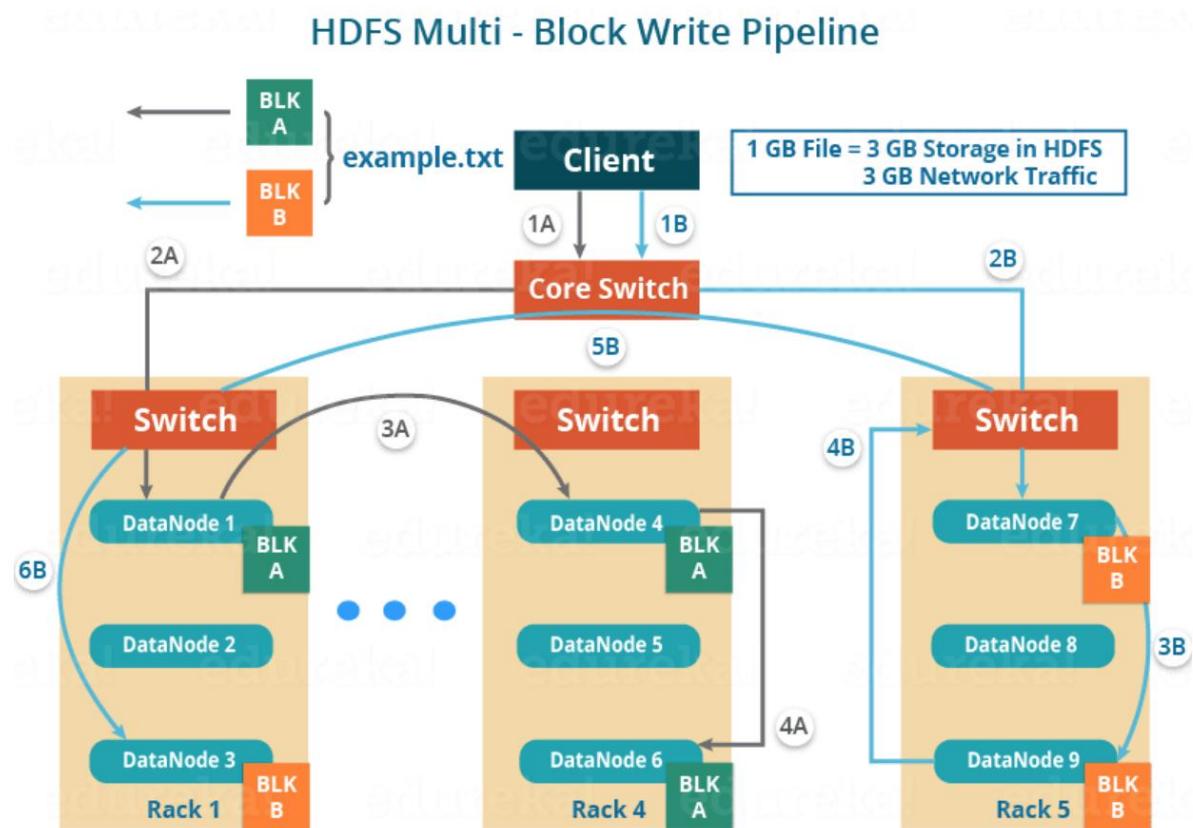
Block is now copied to all DNs. Client and NameNode need to be updated. Client needs to close pipeline and end TCP session.

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

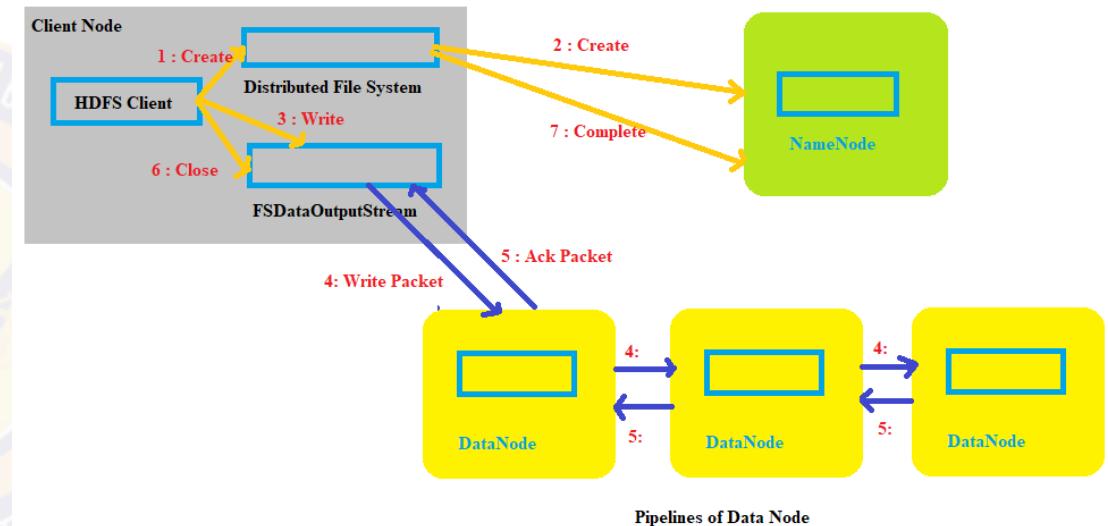


Sample write code

```
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();  
    }  
}
```

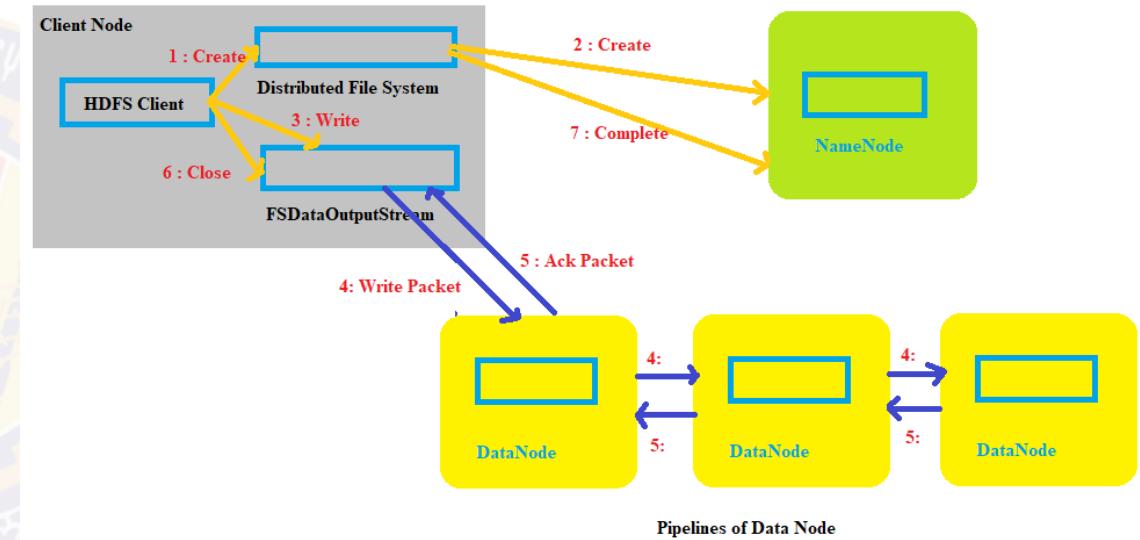
HDFS Create / Write - Call sequence in code

- 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.



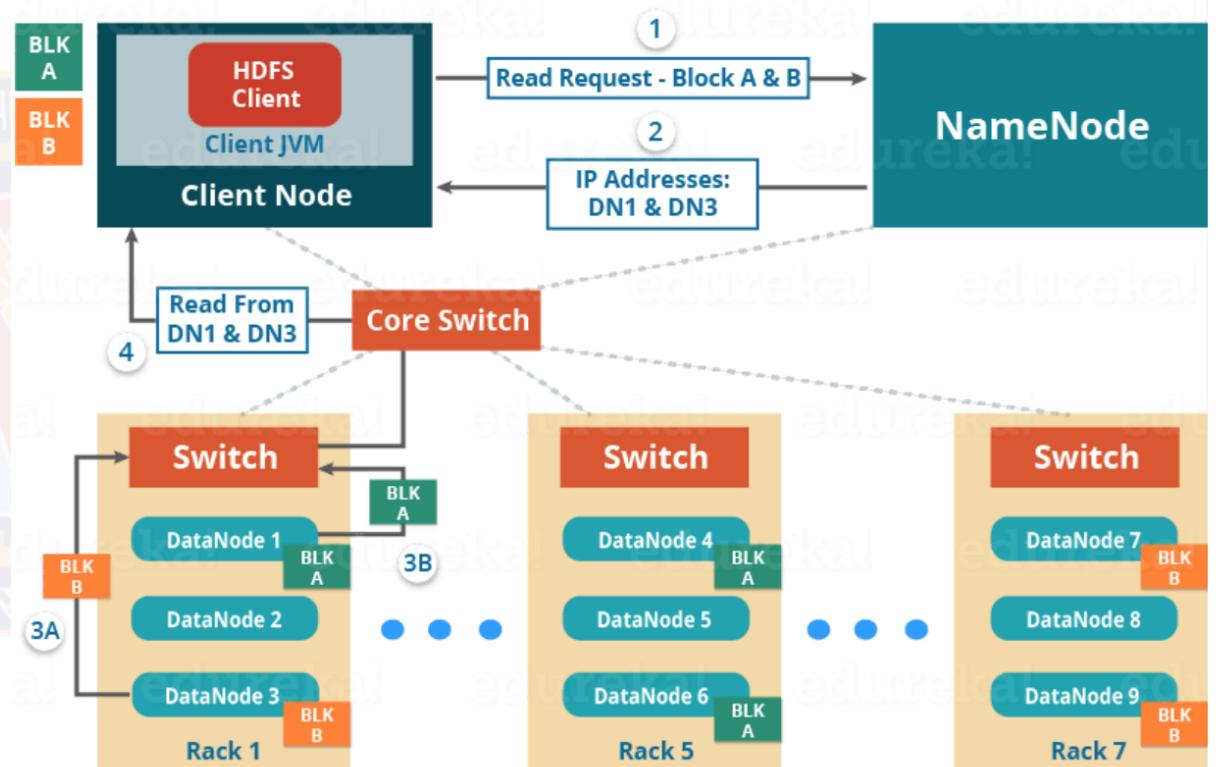
HDFS Read (Sec1)

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

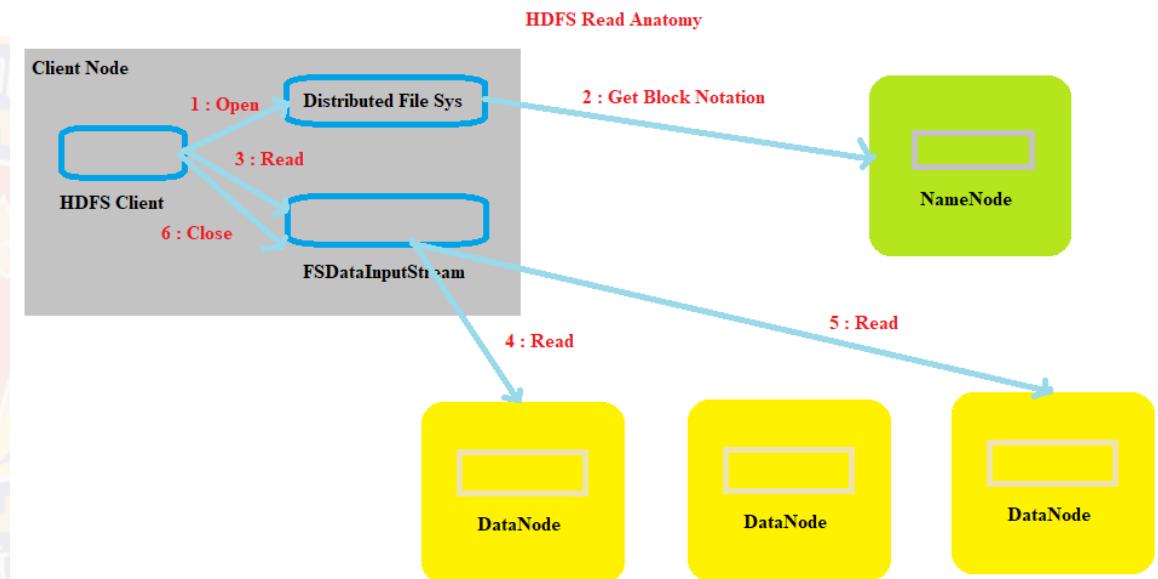


Sample read code

```
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();  
    }  
}
```

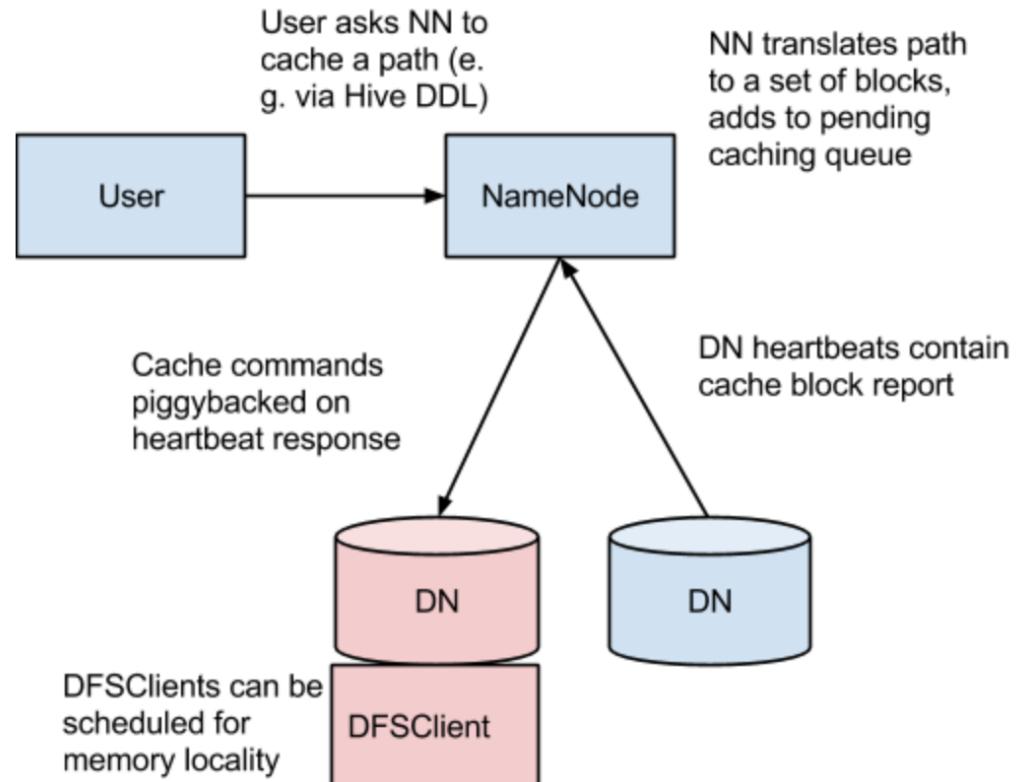
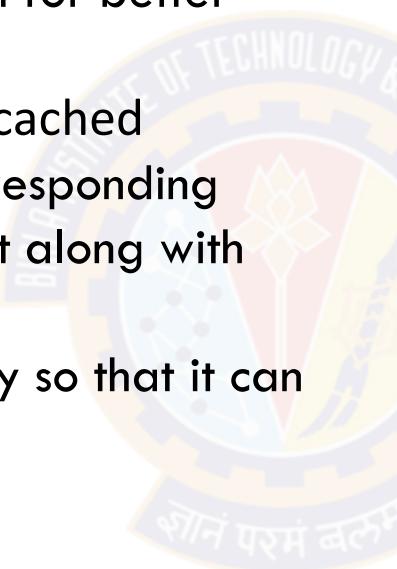
HDFS Read - Call sequence in code

- 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

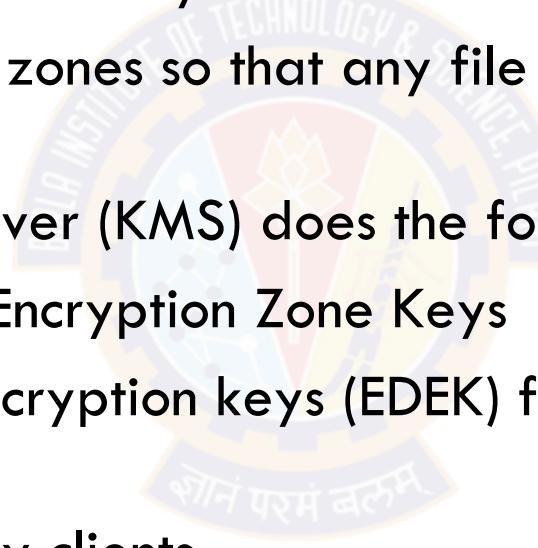
- Short-circuit reads can also be made by client to local file bypassing DataNode using /dev/shm for better performance.
- Client can ask certain HDFS paths to be cached
 - ✓ NameNode asks DNs to cache corresponding blocks and send cache block report along with heartbeat
- Clients can be scheduled for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



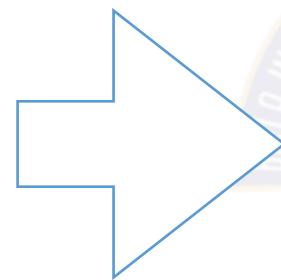
Topics for Session 5

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

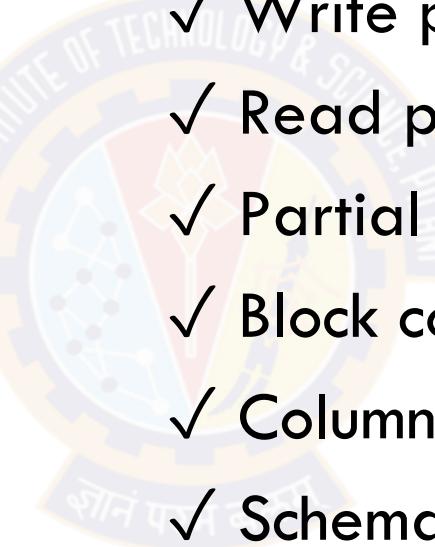


File formats

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

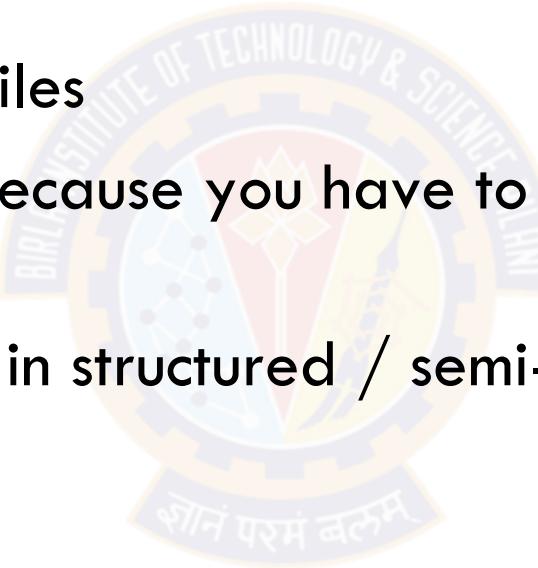


- Many ways to evaluate formats
 - ✓ Write performance
 - ✓ Read performance
 - ✓ Partial read performance
 - ✓ Block compression
 - ✓ Columnar support
 - ✓ Schema change



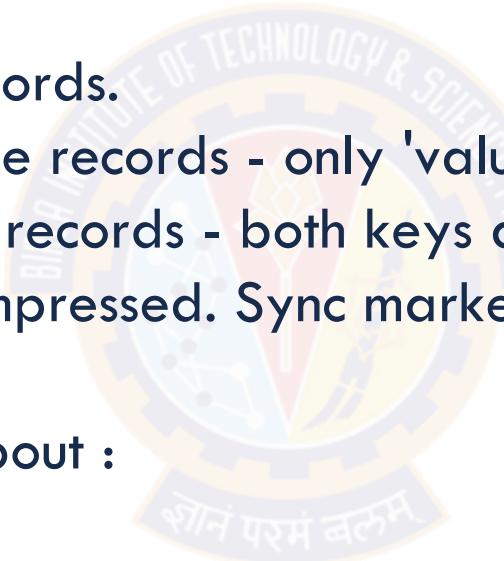
File formats - text based

- 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



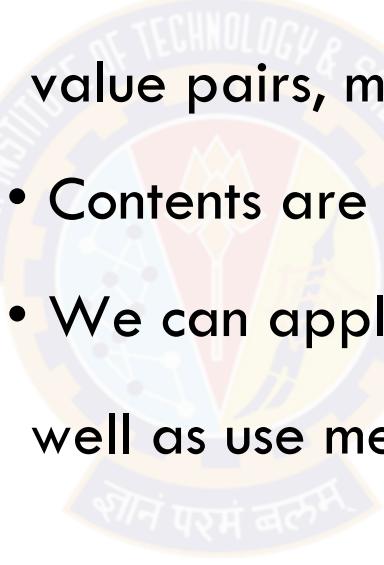
File formats - sequence files

- 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

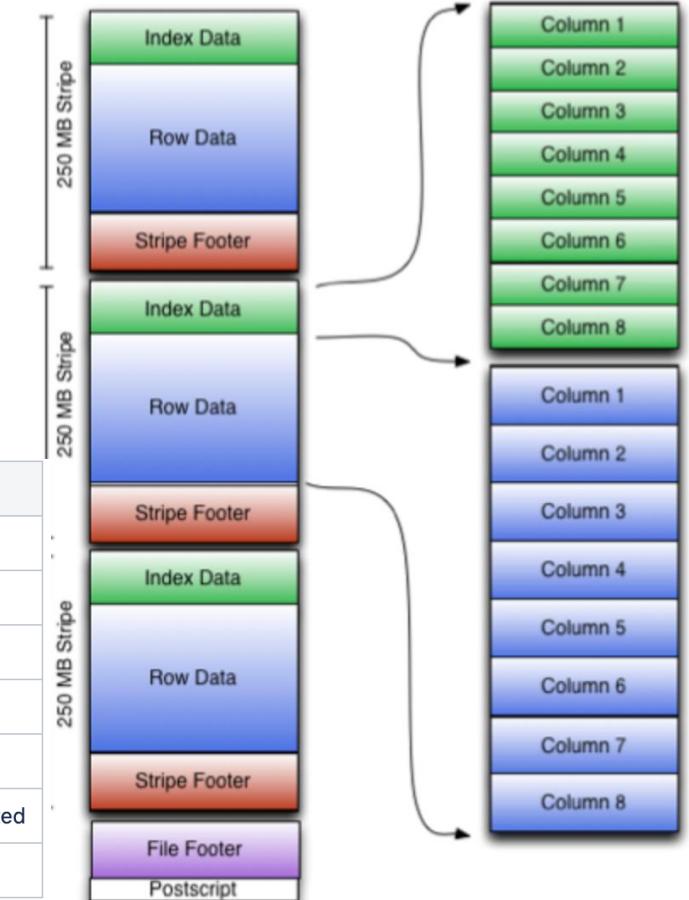
File formats - Optimized Row Columnar (ORC) *

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

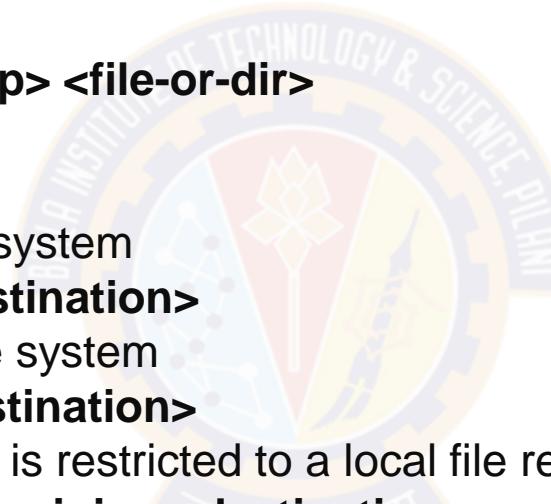
- Hadoop architecture overview
 - ✓ Components
 - ✓ Hadoop 1 vs Hadoop 2
- HDFS
 - ✓ Architecture
 - ✓ Robustness
 - ✓ Blocks and replication strategy
 - ✓ Read and write operations
 - ✓ File formats
 - ✓ Commands



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

Basic command reference

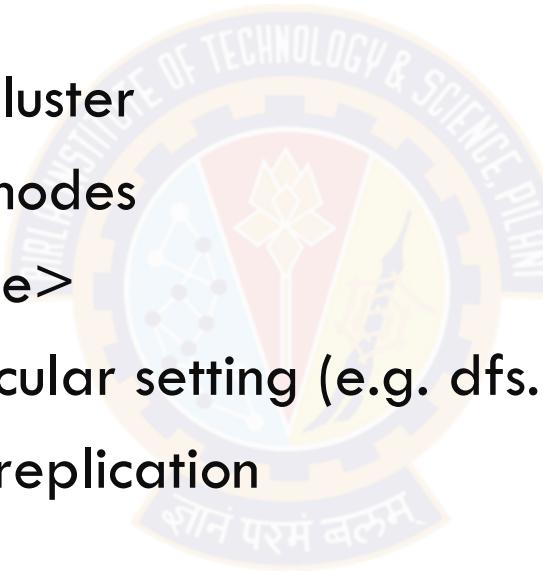
- list files in the path of the file system
 - ✓ **hadoop fs -ls <path>**
- alters the permissions of a file where <arg> is the binary argument e.g. 777
 - ✓ **hadoop fs -chmod <arg> <file-or-dir>**
- change the owner of a file
 - ✓ **hadoop fs -chown <owner>:<group> <file-or-dir>**
- make a directory 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>





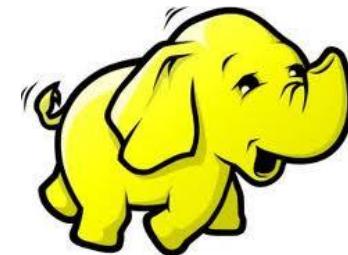
Next Session: Distributed Programming

Introduction to Hadoop

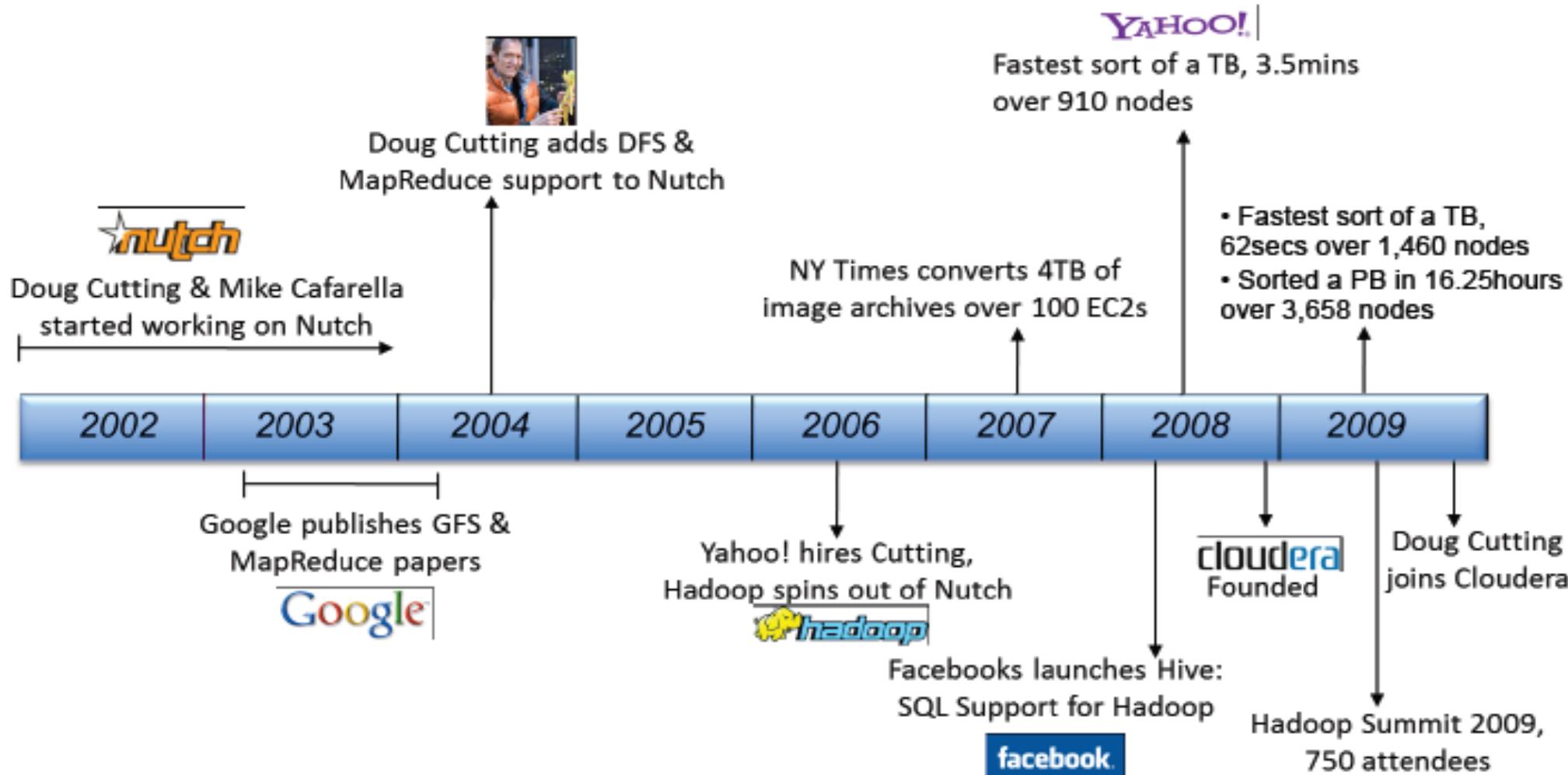
Janardhanan PS

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

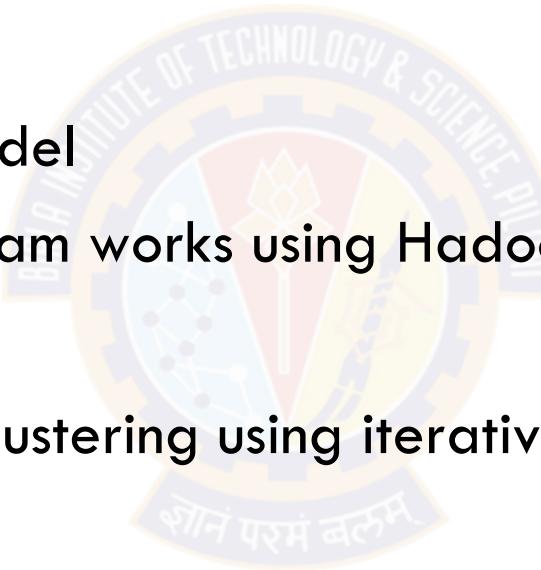
Big Data Systems

Session 6 - Distributed Programming

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

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

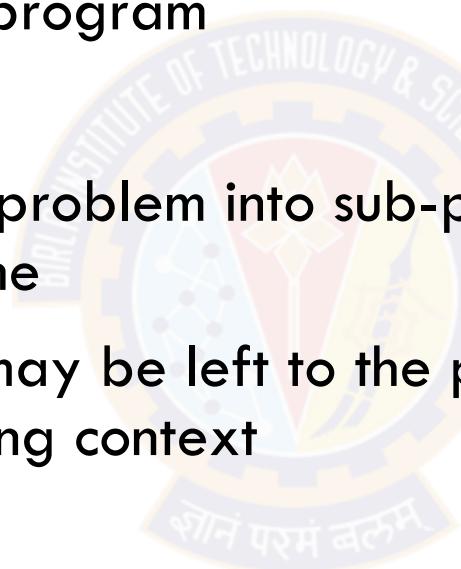
- In the context of a sequential program
 - Divide and conquer
 - It is easier to divide a problem into sub-problems and execute one by one
 - A sub-problem definition may be left to the programmer in a sequential programming context

main()

f1() → f2() → f5()

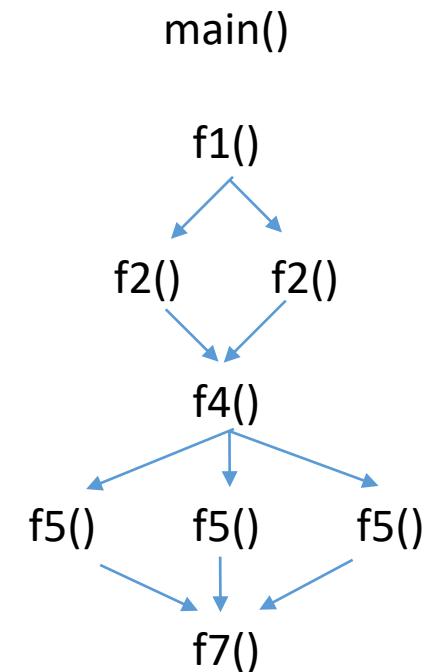
f3()

f4()



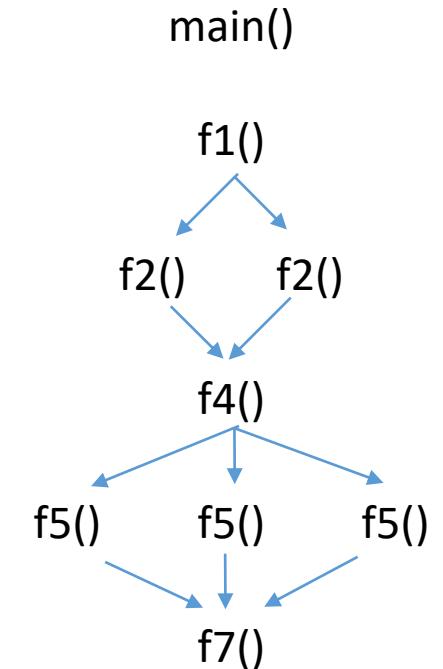
Top down design - parallel context

- 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

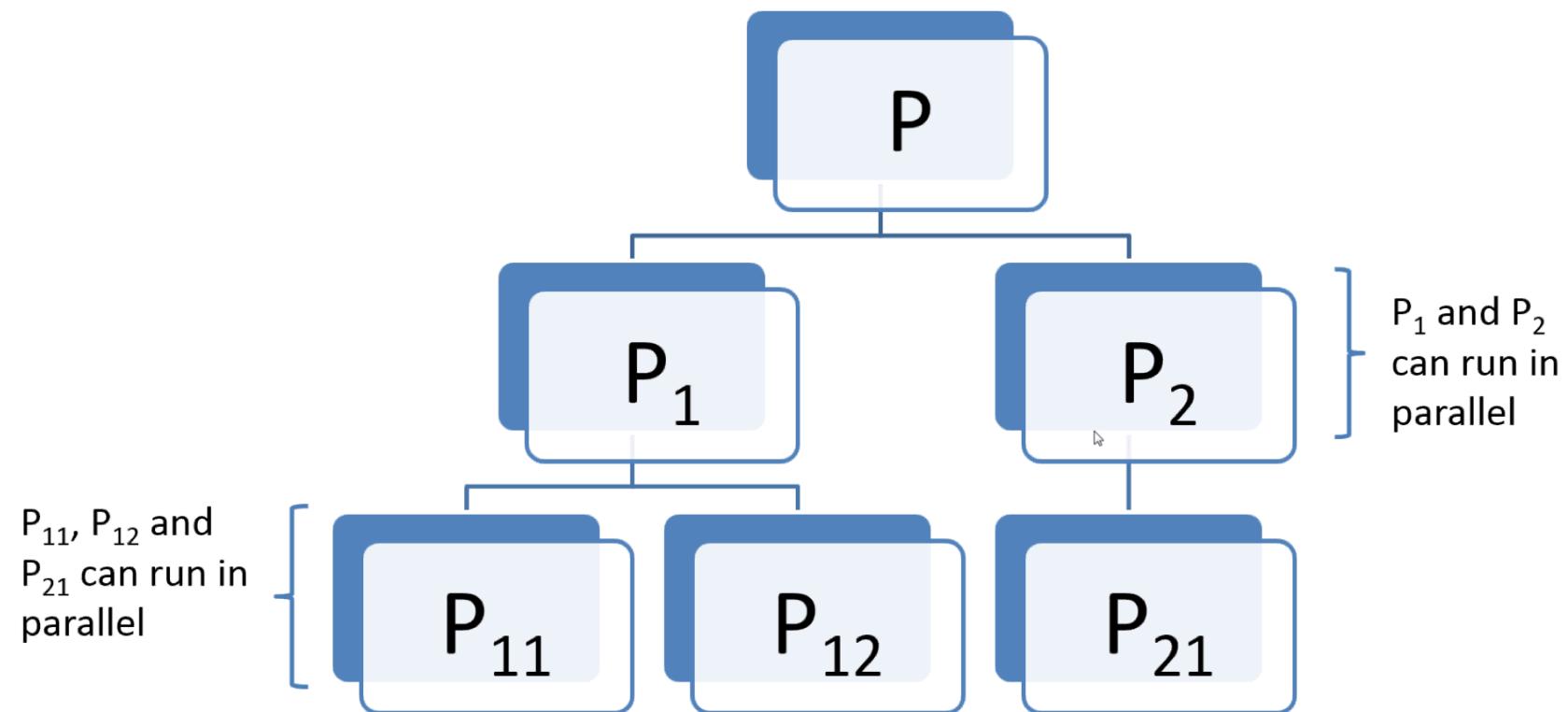
- In conventional top down design for sequential systems
 - Keep number of sub-problems manageable
 - Because need to keep track of them as computation progresses
- In parallel system it is dictated by number of processors
 - Processor utilisation is the key
 - If there are N processors, we can potentially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



How many processors ?

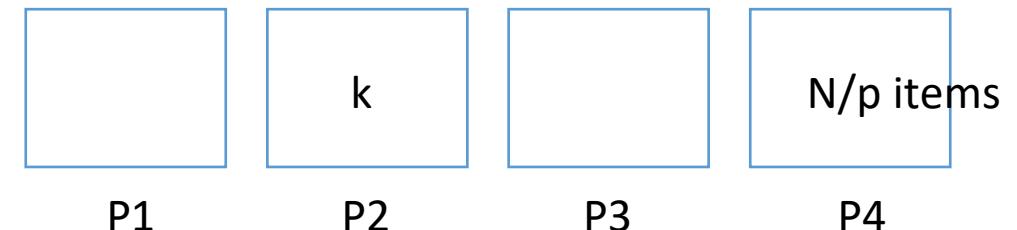
Top-down design

- At each level problems need to run in parallel



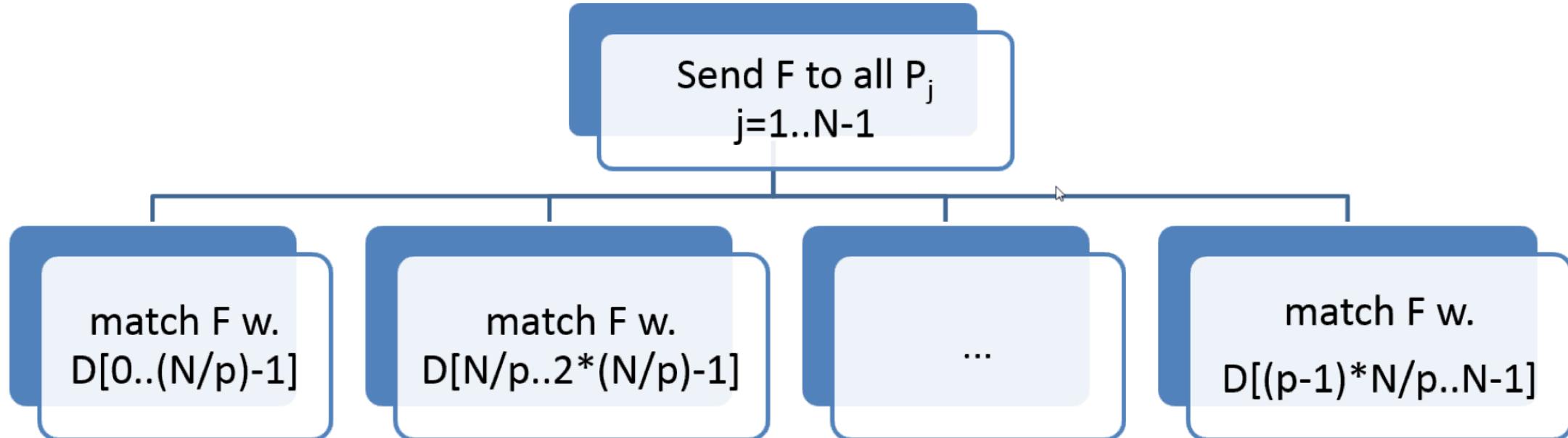
Example 1 - Keyword search in list

- 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: $O(\log(N/p))$



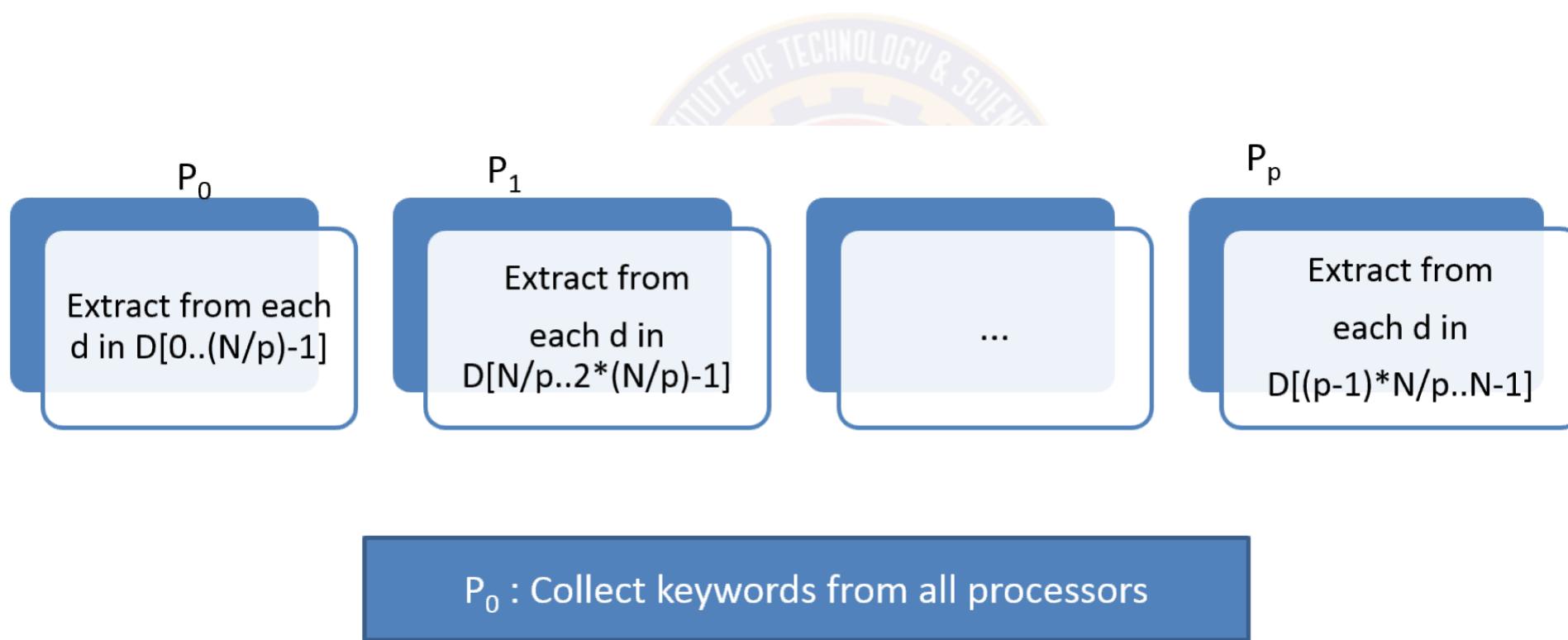
Example 2 - Fingerprint matching

- 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



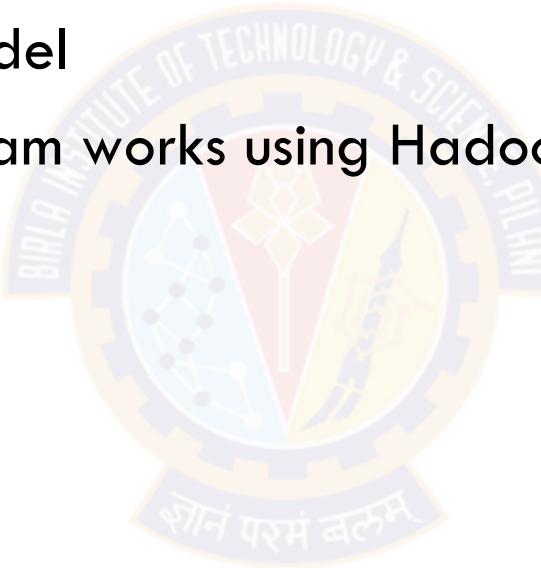
Example 3: Document search

- 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



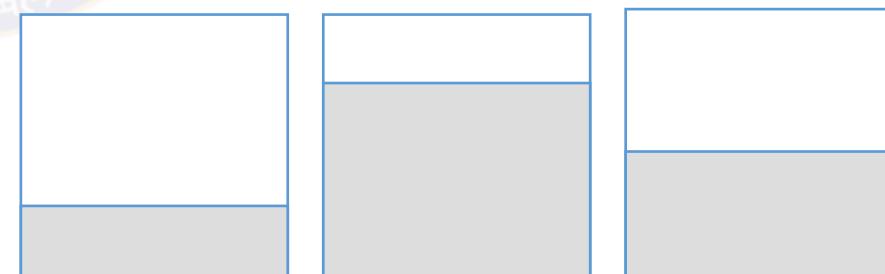
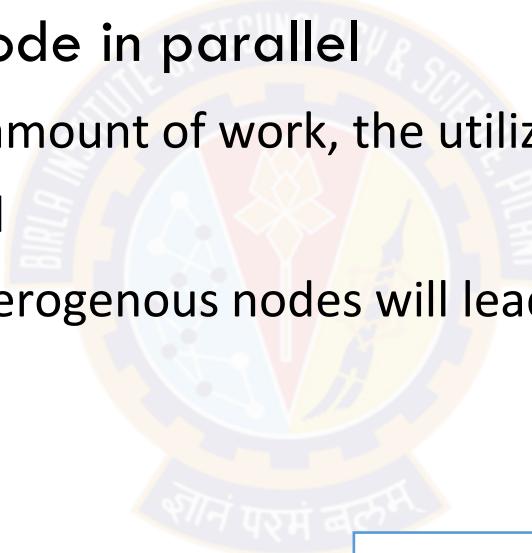
Types of Parallelism

- 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



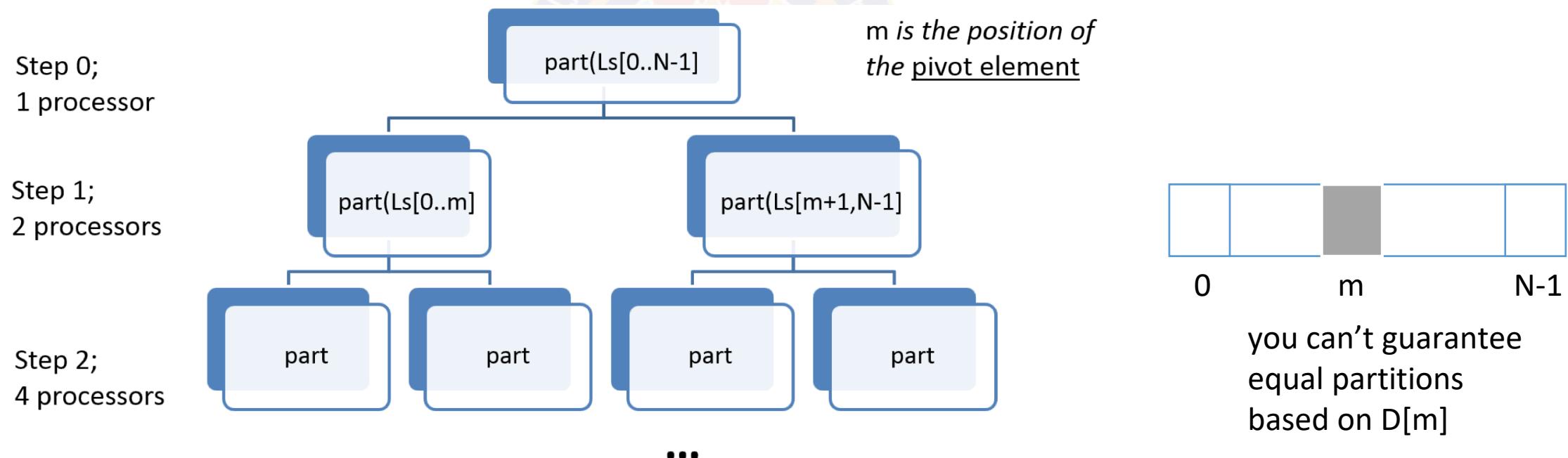
Where data parallelism is not possible

- 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

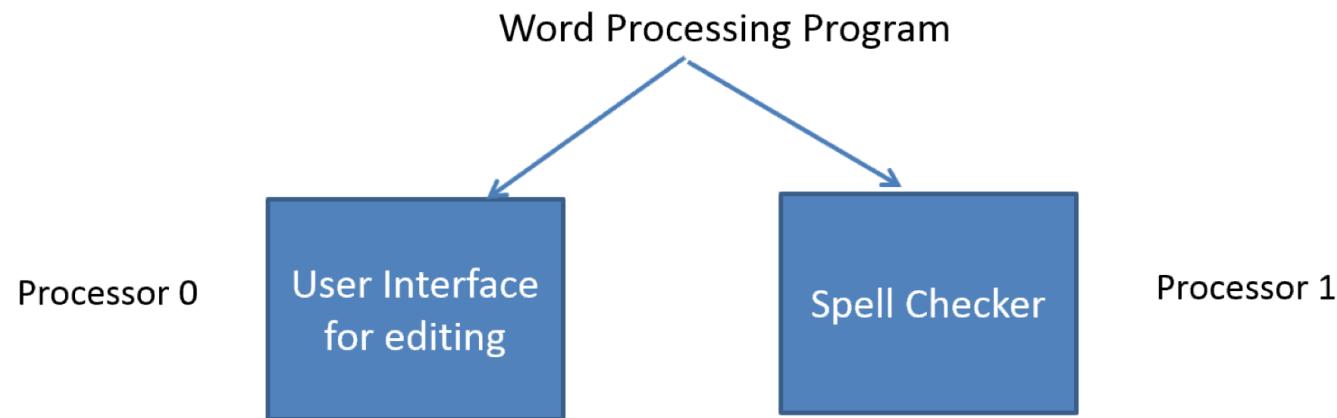


Tree parallelism summary

- 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

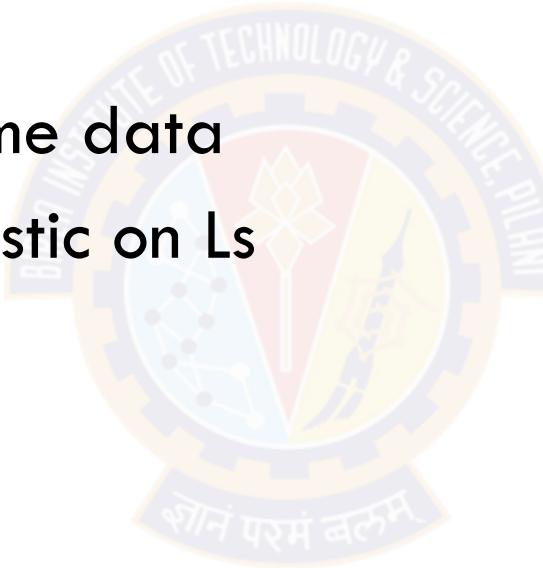
Task parallelism - Example 1 : Word processor

- 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

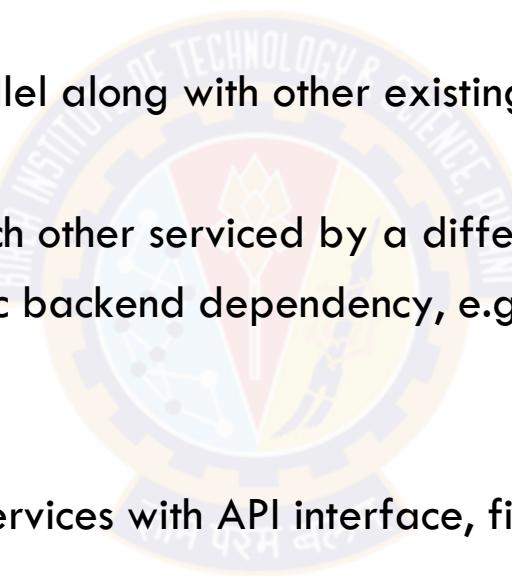


Task parallelism summary

- 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

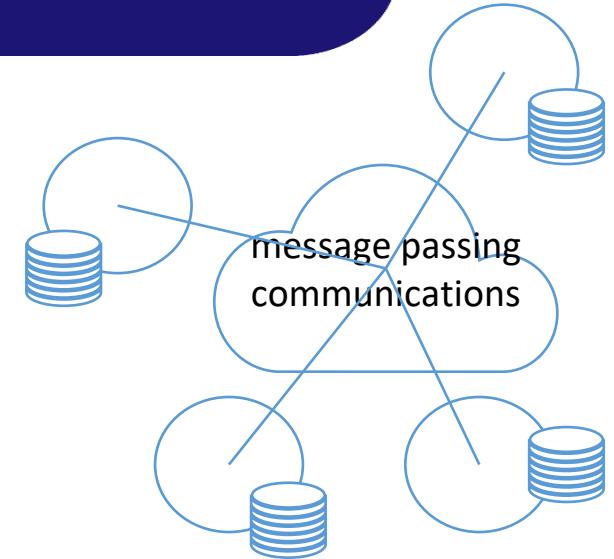
Request parallelism

- 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)



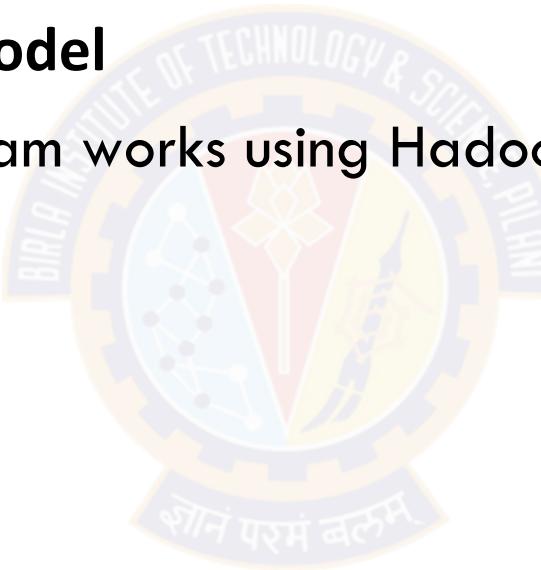
What happens in a loosely coupled distributed system

- 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



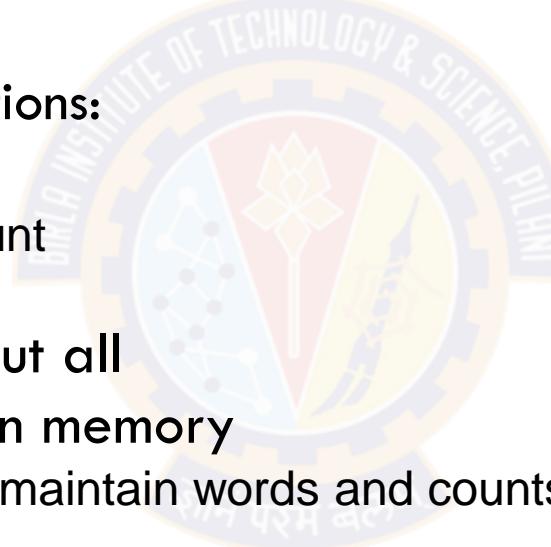
MapReduce origins

- 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:

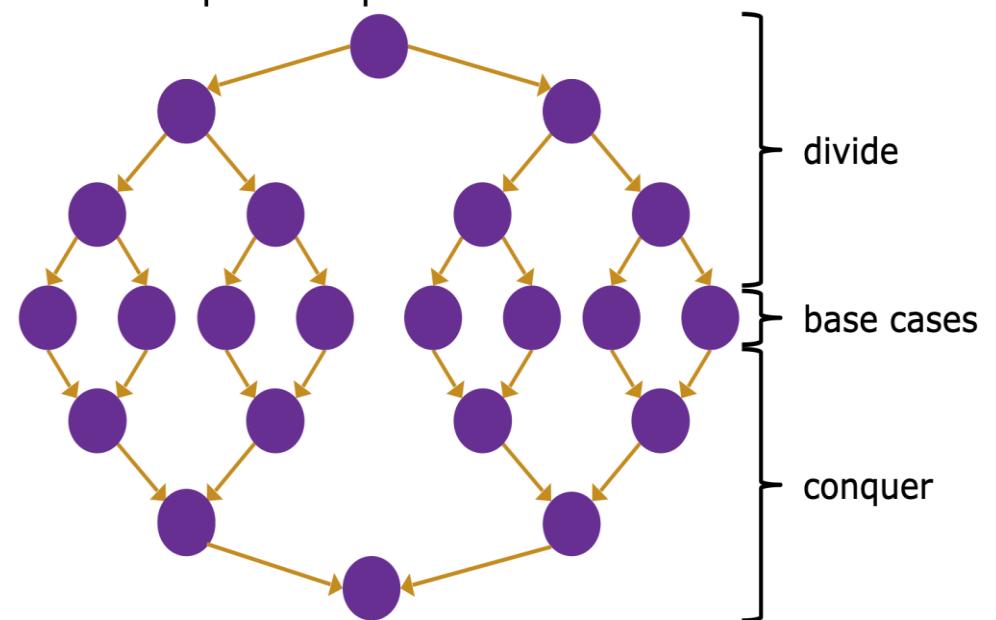
- `map()` - User code
- `reduce()` - User code

→ 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):
```

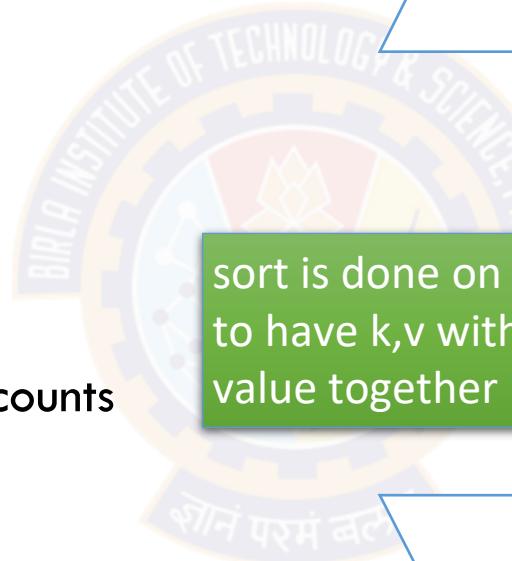
```
// key: a word; value: an iterator over counts
```

```
    result = 0
```

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

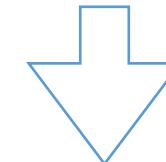
```
        result += v
```

```
    emit(result)
```



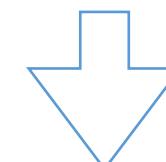
D1 : the blue ship on blue sea

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



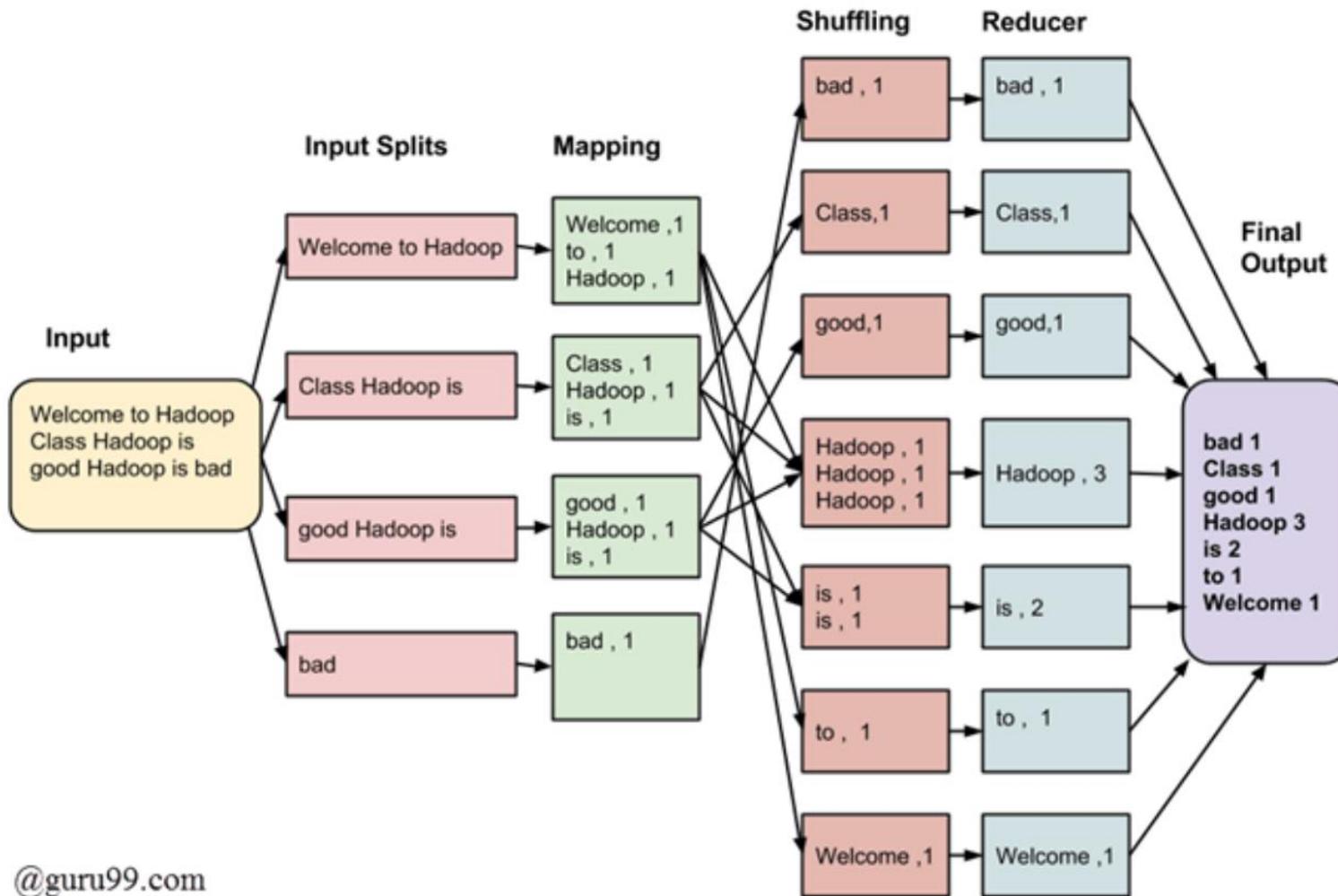
blue, [1,1] | on, 1 | sea, 1 | ship, 1 | the, 1

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

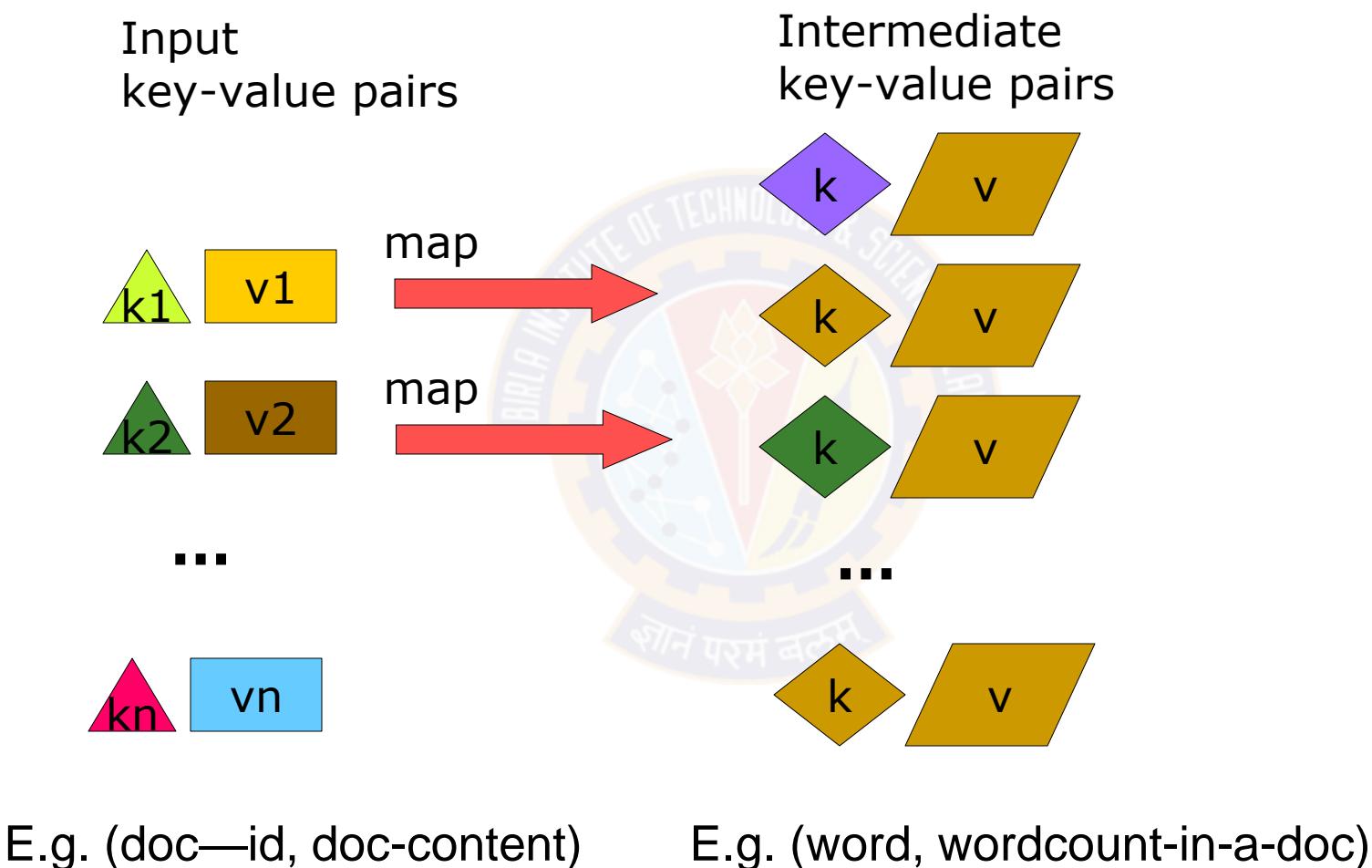


blue, 2 | on, 1 | sea, 1 | ship, 1 | the, 1

Word count (2)

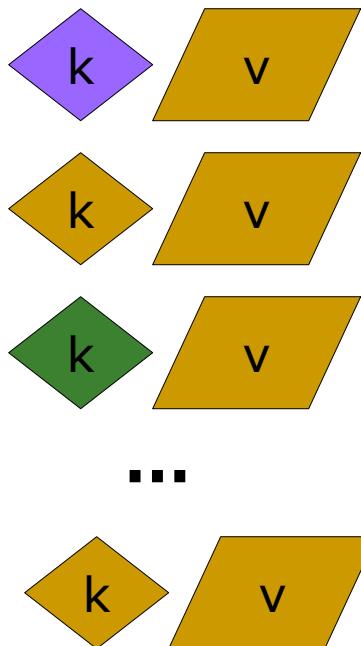


MapReduce: The Map Step



MapReduce: The Reduce Step

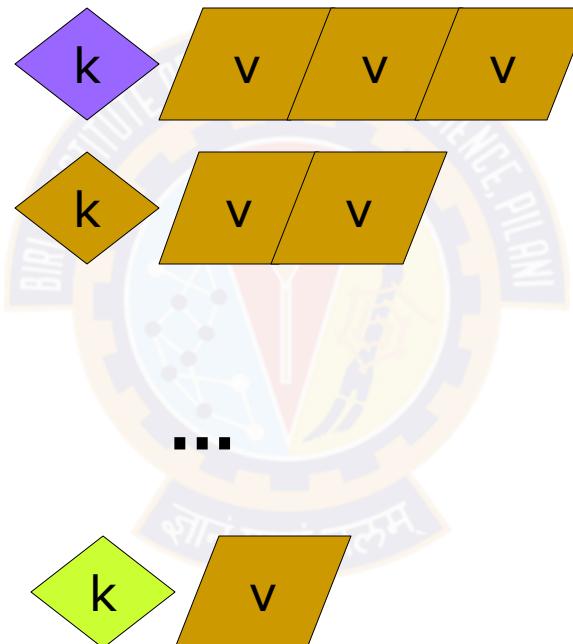
Intermediate key-value pairs



E.g.
(word, wordcount-in-a-doc)

group
→

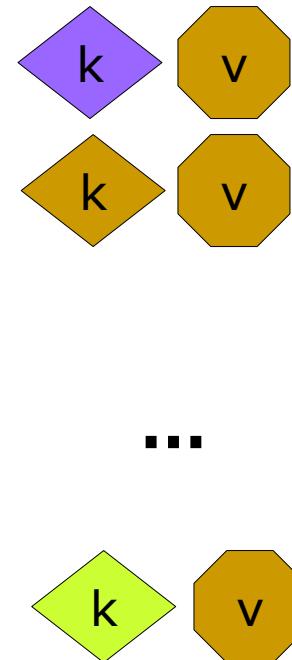
Key-value groups



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

reduce
→

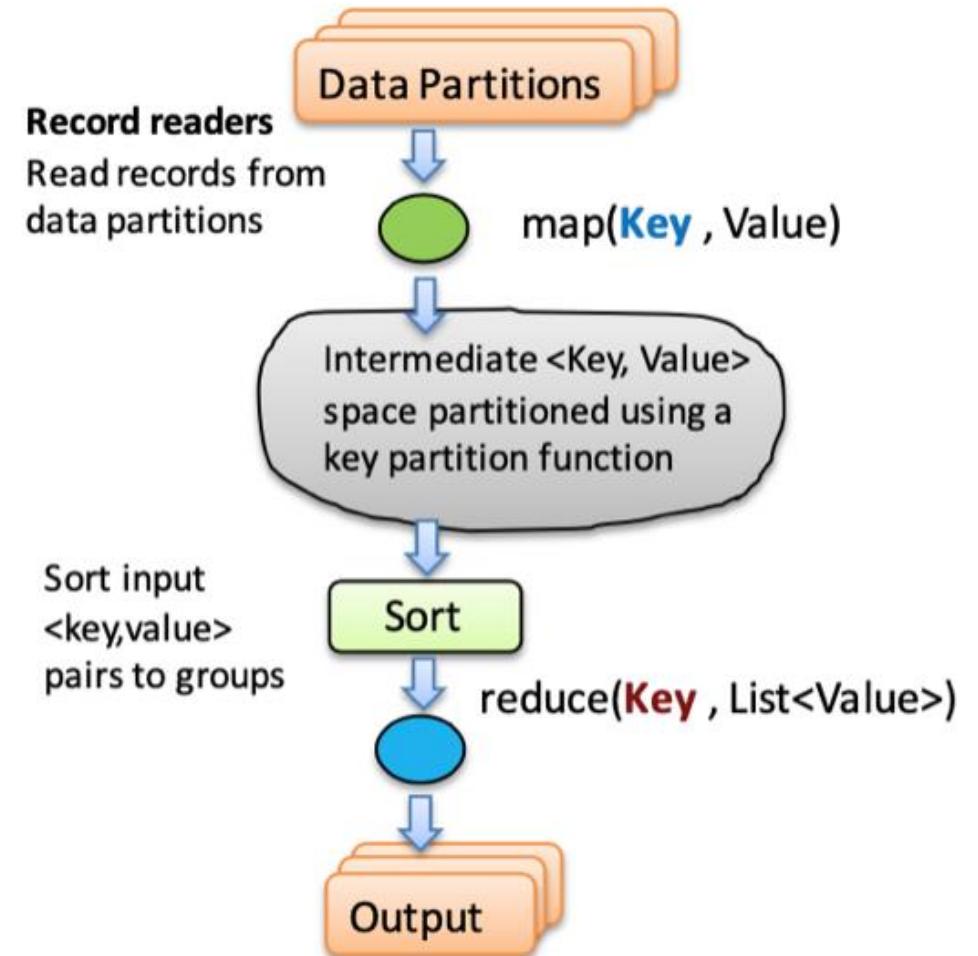
Output key-value pairs



(word, final-count)
~ SQL aggregation

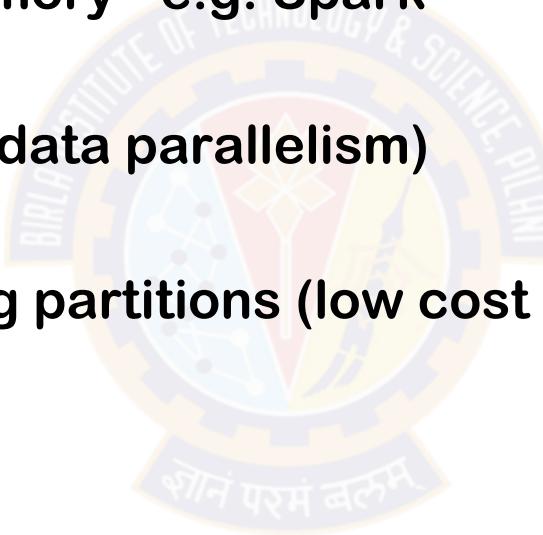
Formal definition of a MapReduce program

- 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



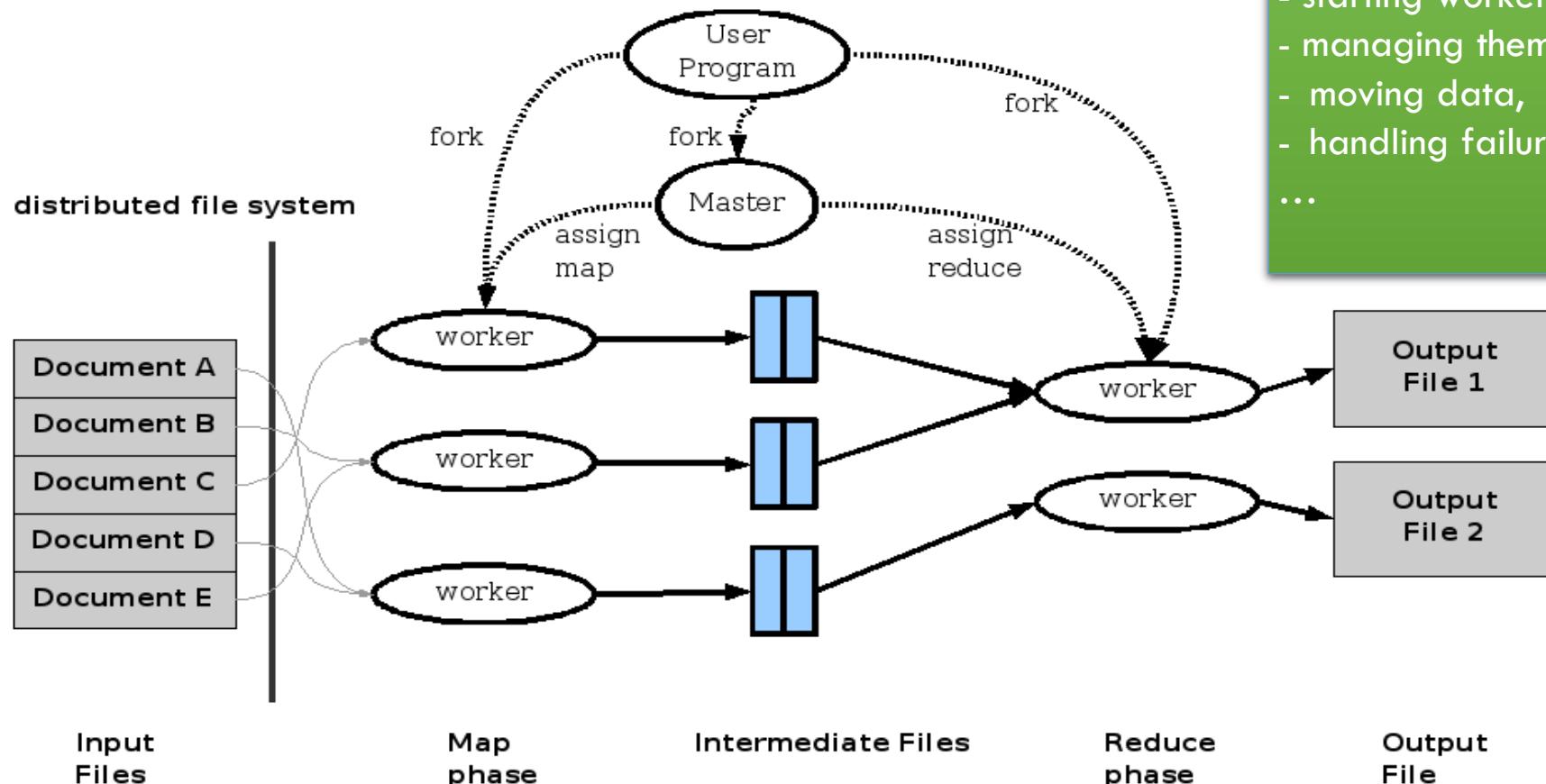
When will you use this ?

- 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

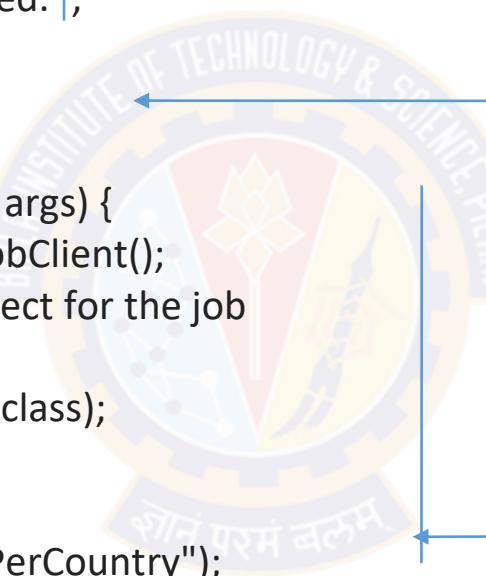
	A	B	C	D	E	F	G	H	I	J	K	L	
1	Transaction_date	Product	Price	Payment_Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude		
2	01-02-2009 06:17	Product1	1200	Mastercard	Carolina	Basildon	England	United Kingdom	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667	
3	01-02-2009 04:53	Product1	1200	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 00:02	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

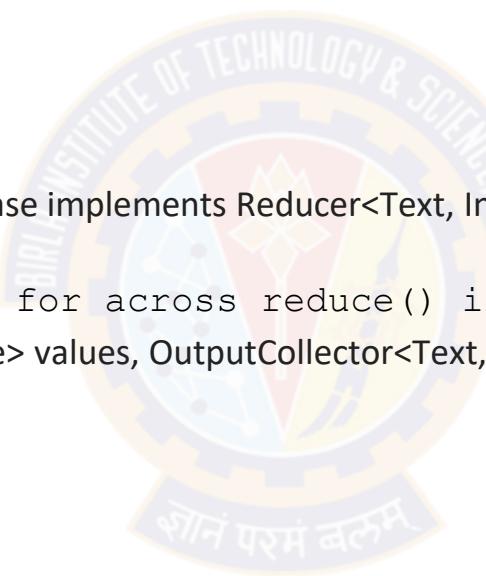
map function w inputs
- one or more <key, value> pairs
- output collector
- ...

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

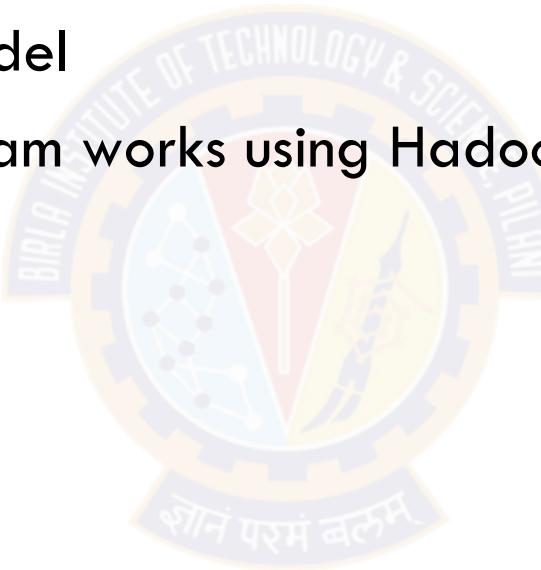
```
Bytes Read=127355
```

File Output Format Counters

```
Bytes Written=661
```

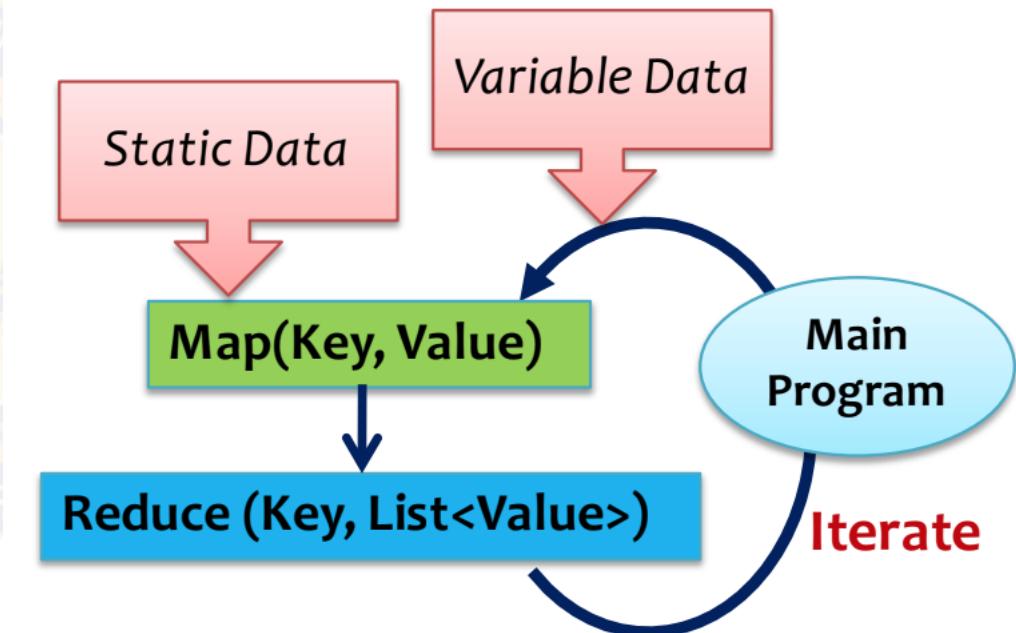
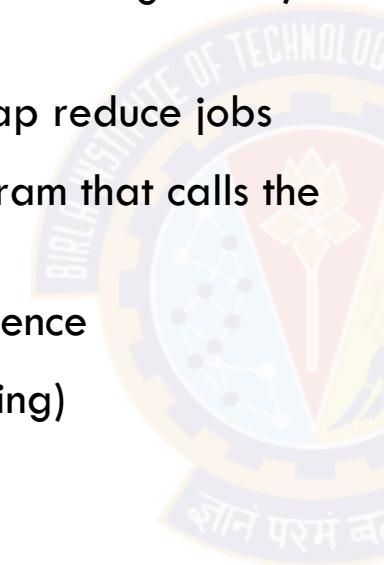
Topics for today

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

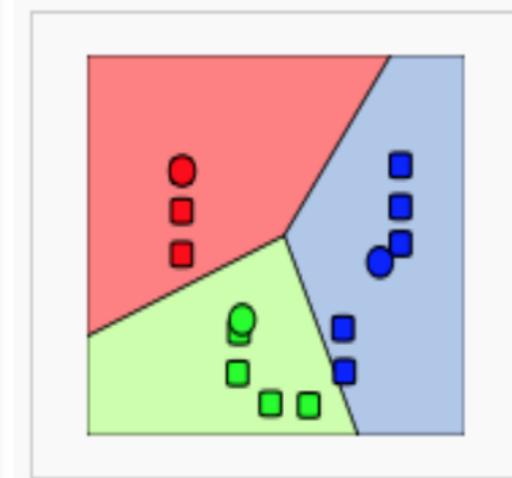
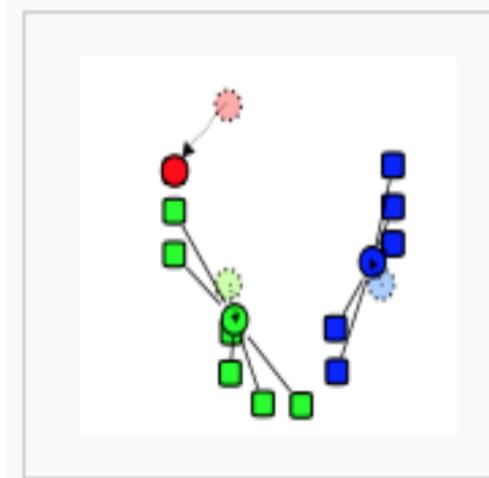
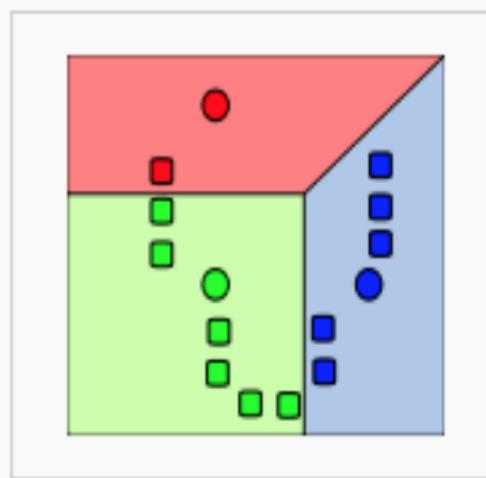
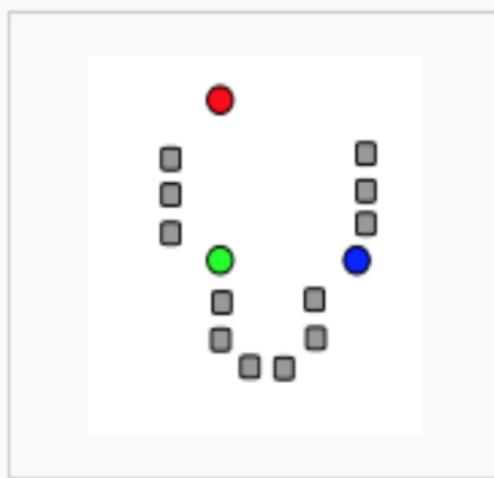


Iterative map reduce

- 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



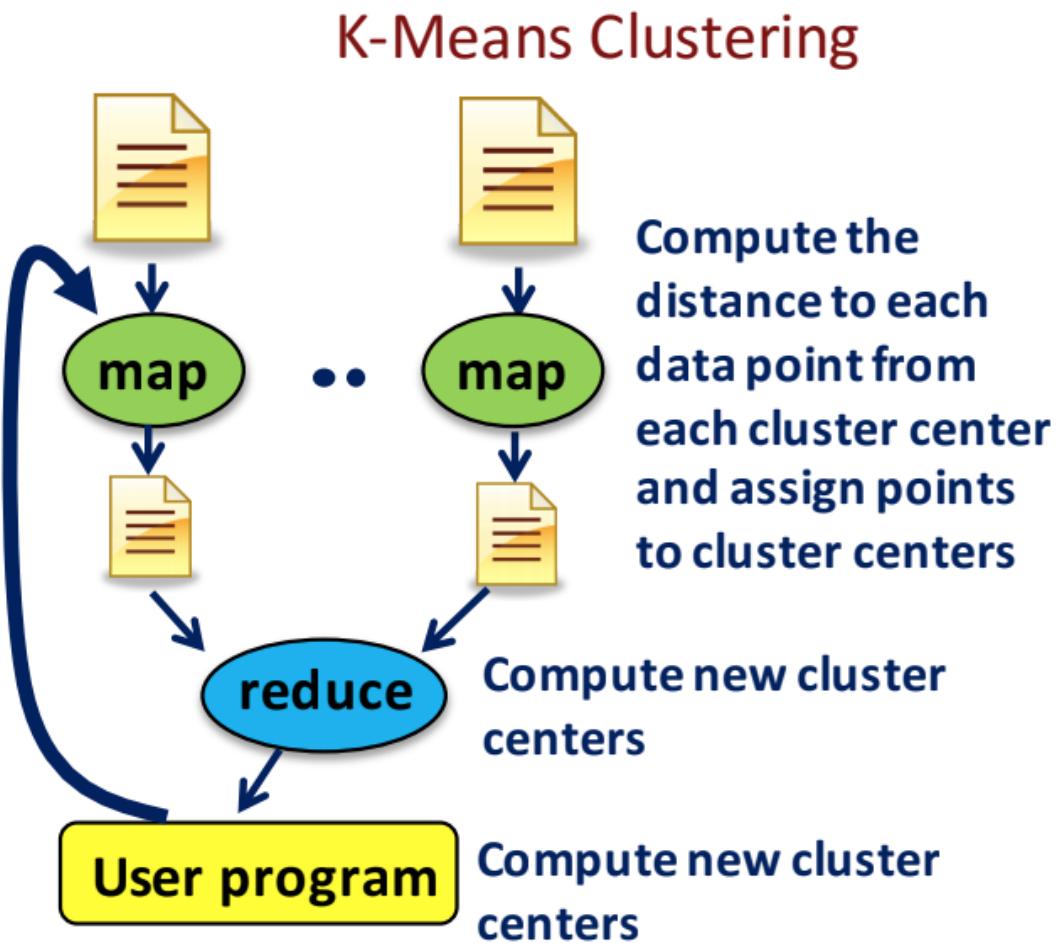
1) k initial "means" (in this case $k=3$) are randomly selected from the data set (shown in color).

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.

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

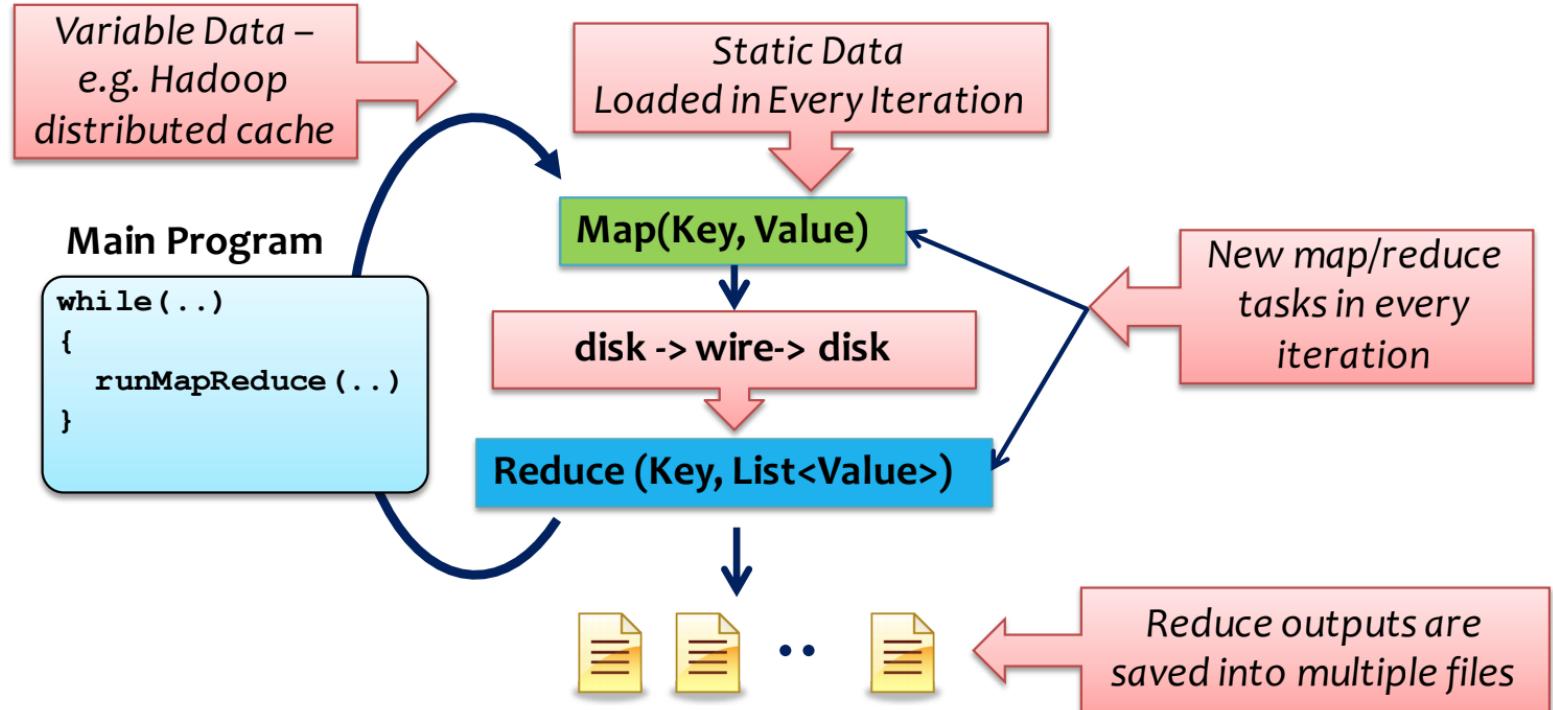
Hands on demo

- 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

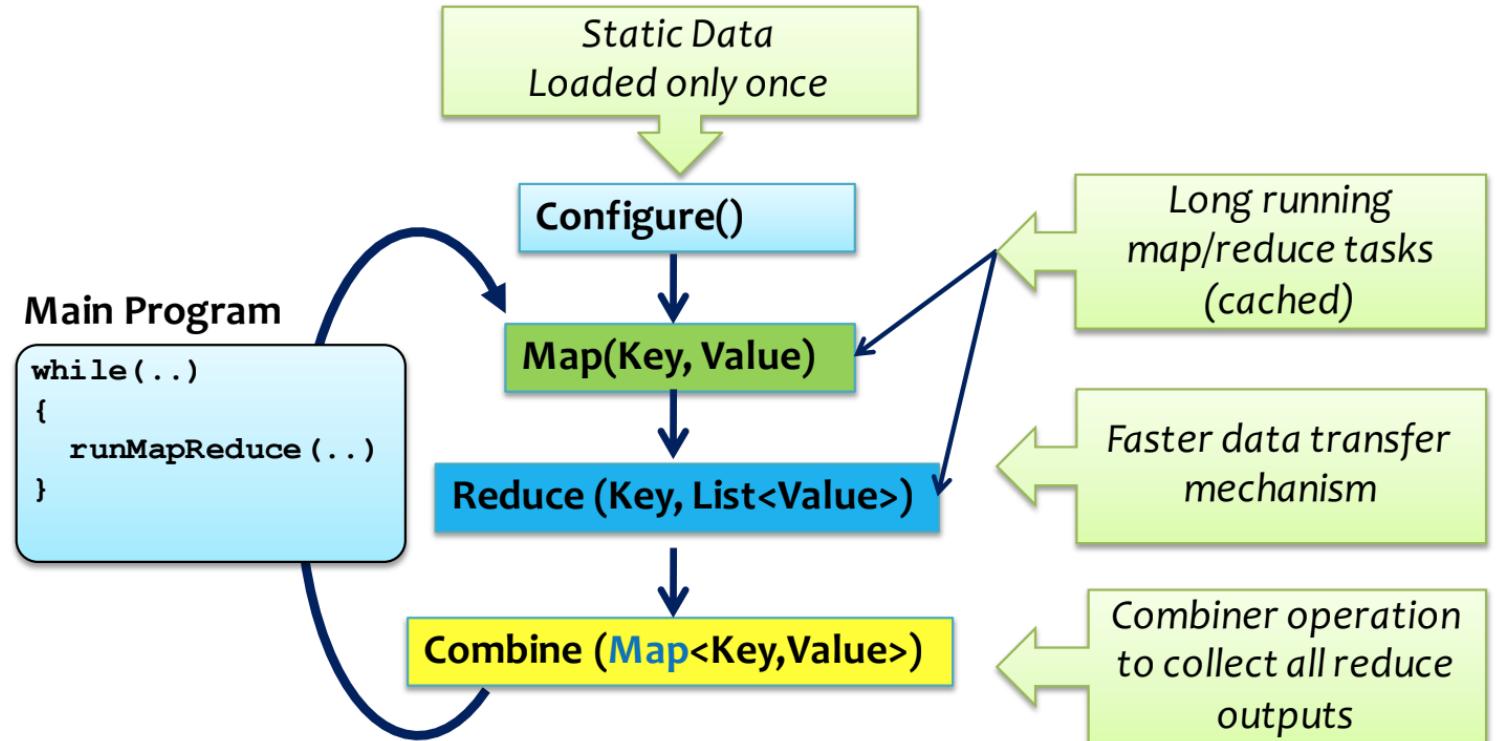
- Loop implemented on top of existing file-based single step map-reduce core
- Large overheads from
 - re-initialization of tasks
 - reloading of static data
 - communication and data transfers



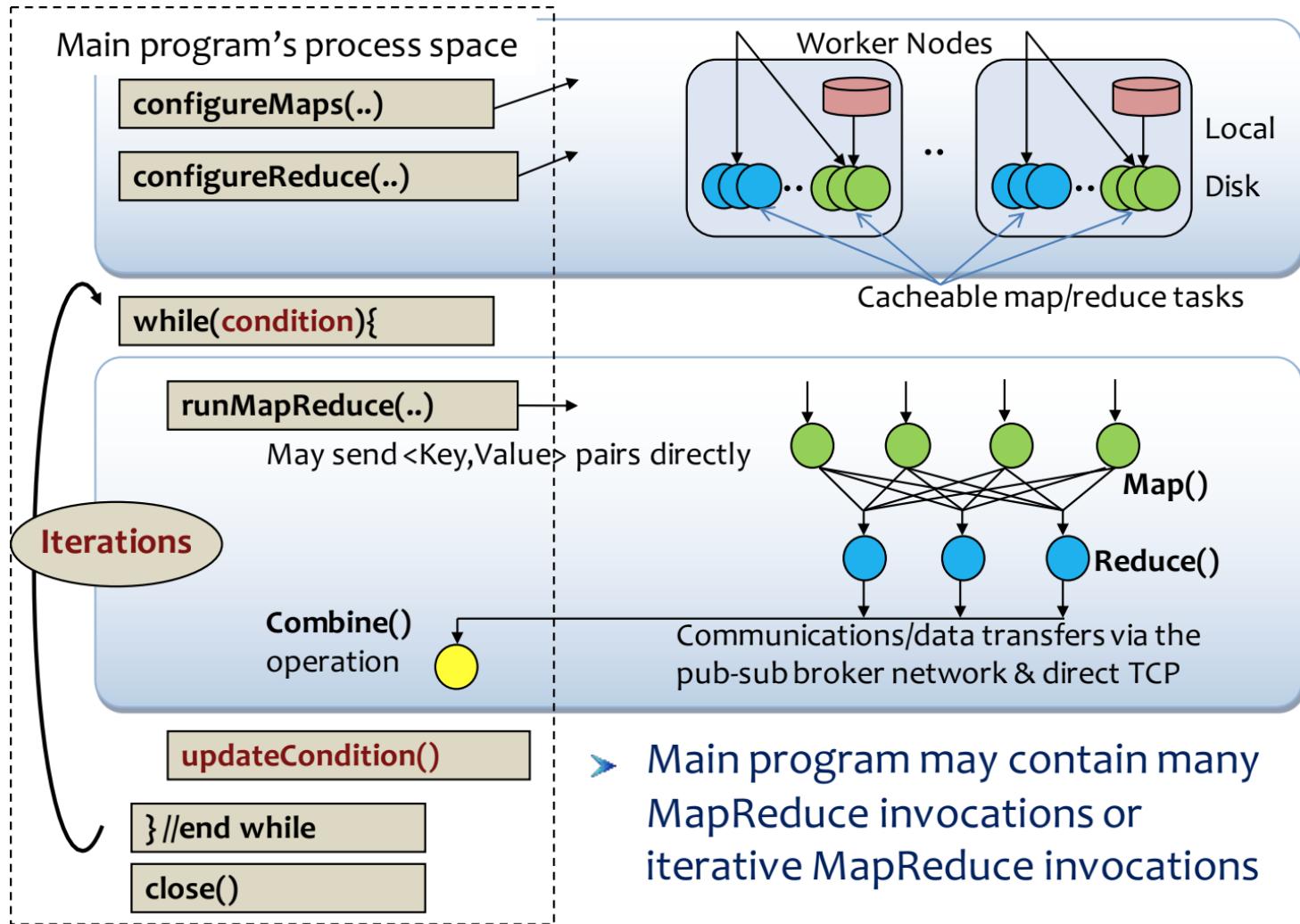
DistributedCache: <https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html>

MapReduce++ : Iterative MapReduce

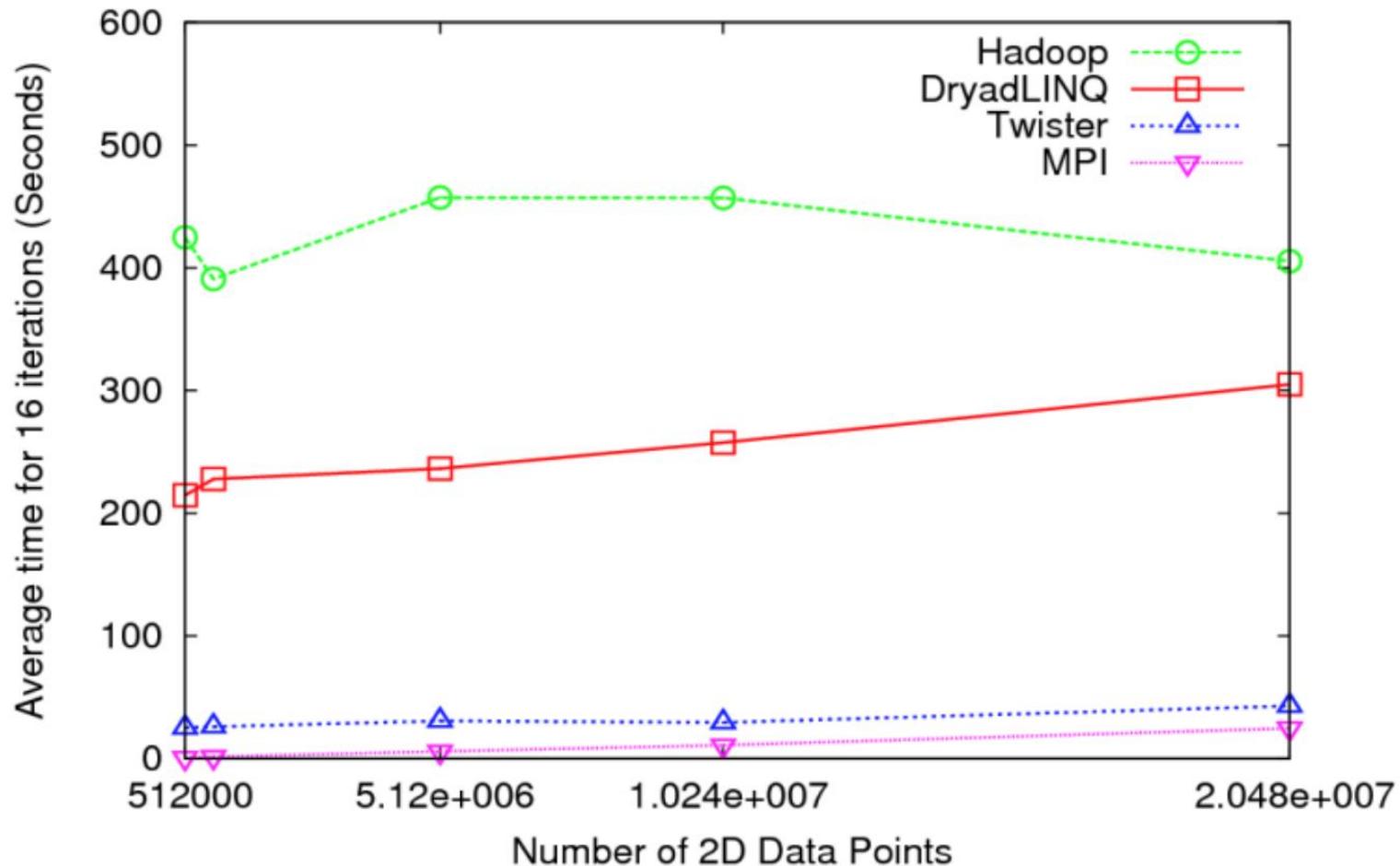
- Some optimizations are done on top of existing model
 - Static data loaded once
 - Cached tasks across invocations
 - Combine operations



Example in Twister: Enables more APIs



The optimisations indeed help



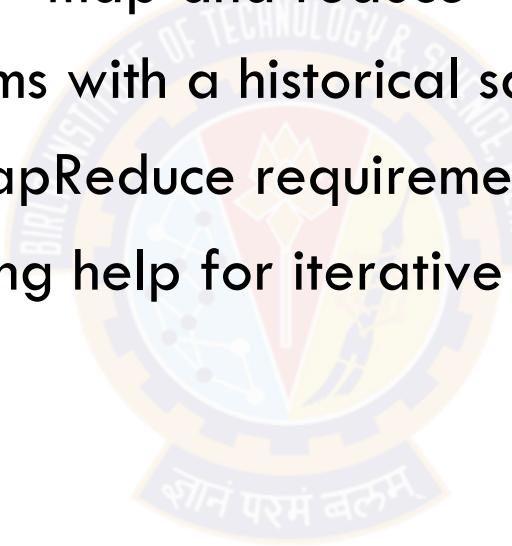
K-means clustering using various programming models

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

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

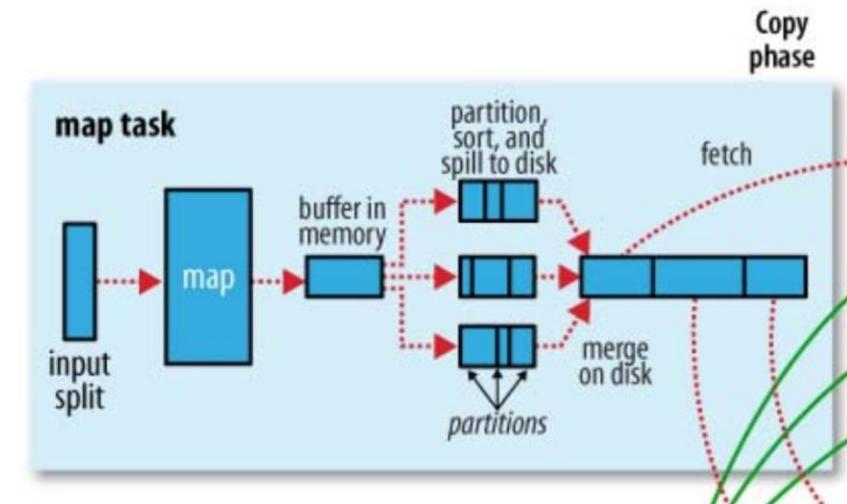
- ✓ User defined function to create key, value pair from input key, value pair

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

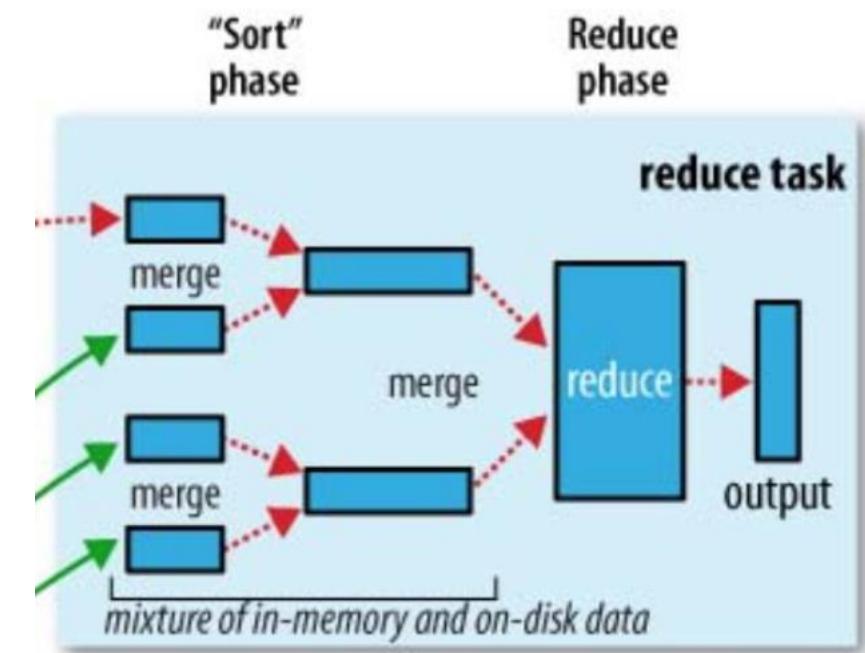
- ✓ Shuffle gets data from map node to reduce node or reduce task to happen where the relevant post-map data partition is
- ✓ Shuffling can start before Map is entirely complete
- ✓ Data is merge-sorted by key coming in from multiple maps

2. Reduce

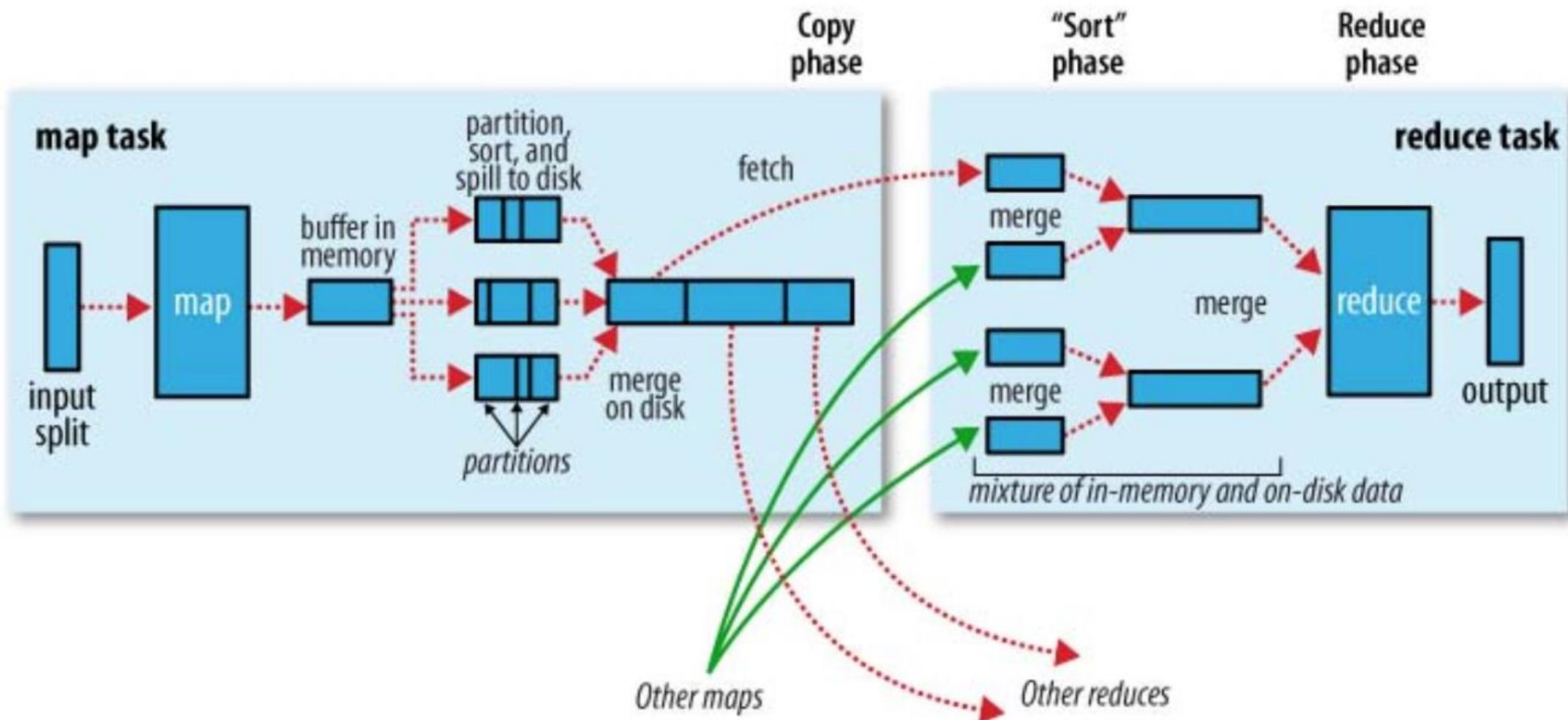
- ✓ Call user defined function per key grouping, e.g. <India,{1,1,1,1}>
- ✓ A reduce call looks at a unique key but global state can be maintained in Reduce task in class variable
- ✓ Can control number of reducers, e.g. one reducer if a sequential computation has to calculate one final value

3. Output Format

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



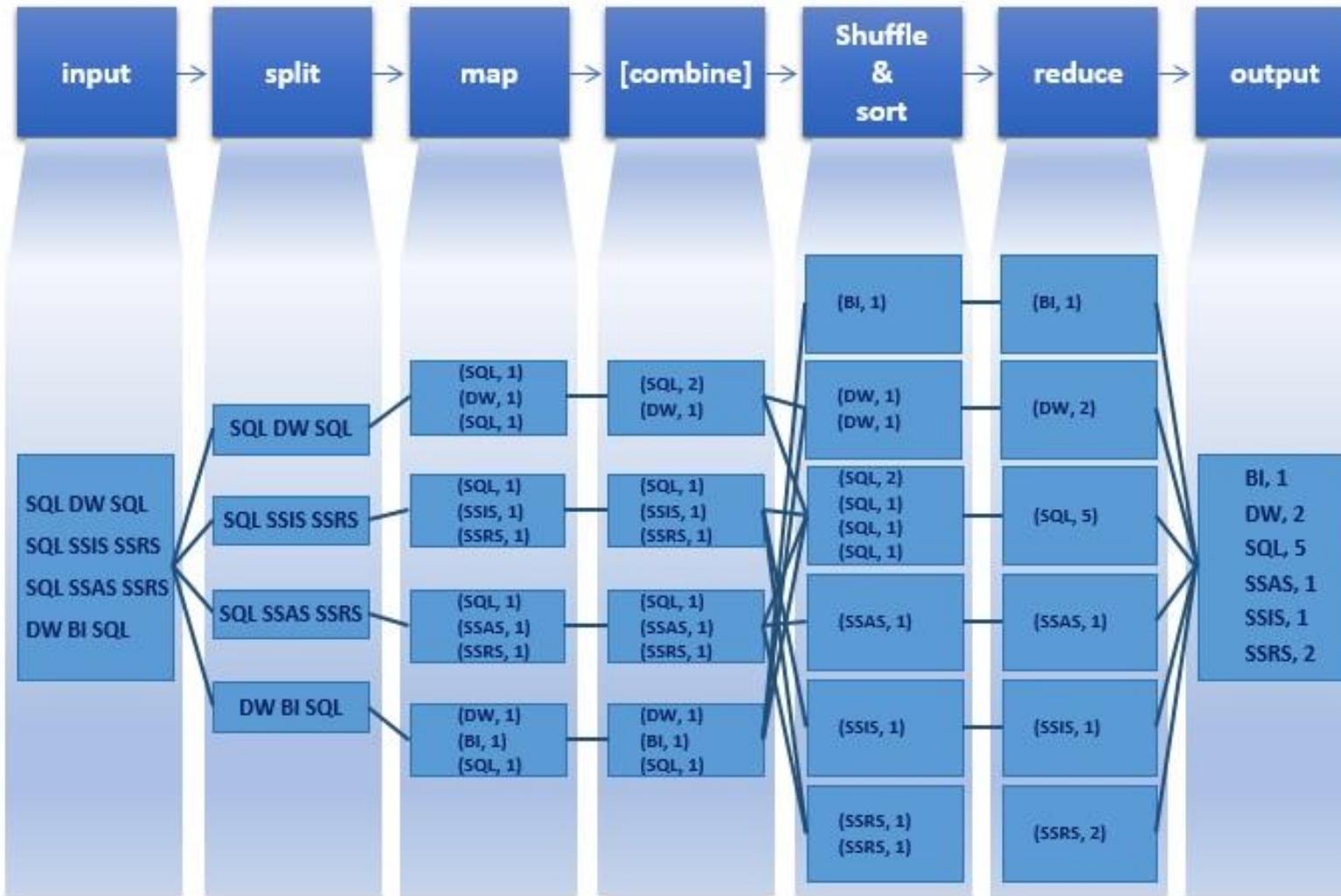
MapReduce flow



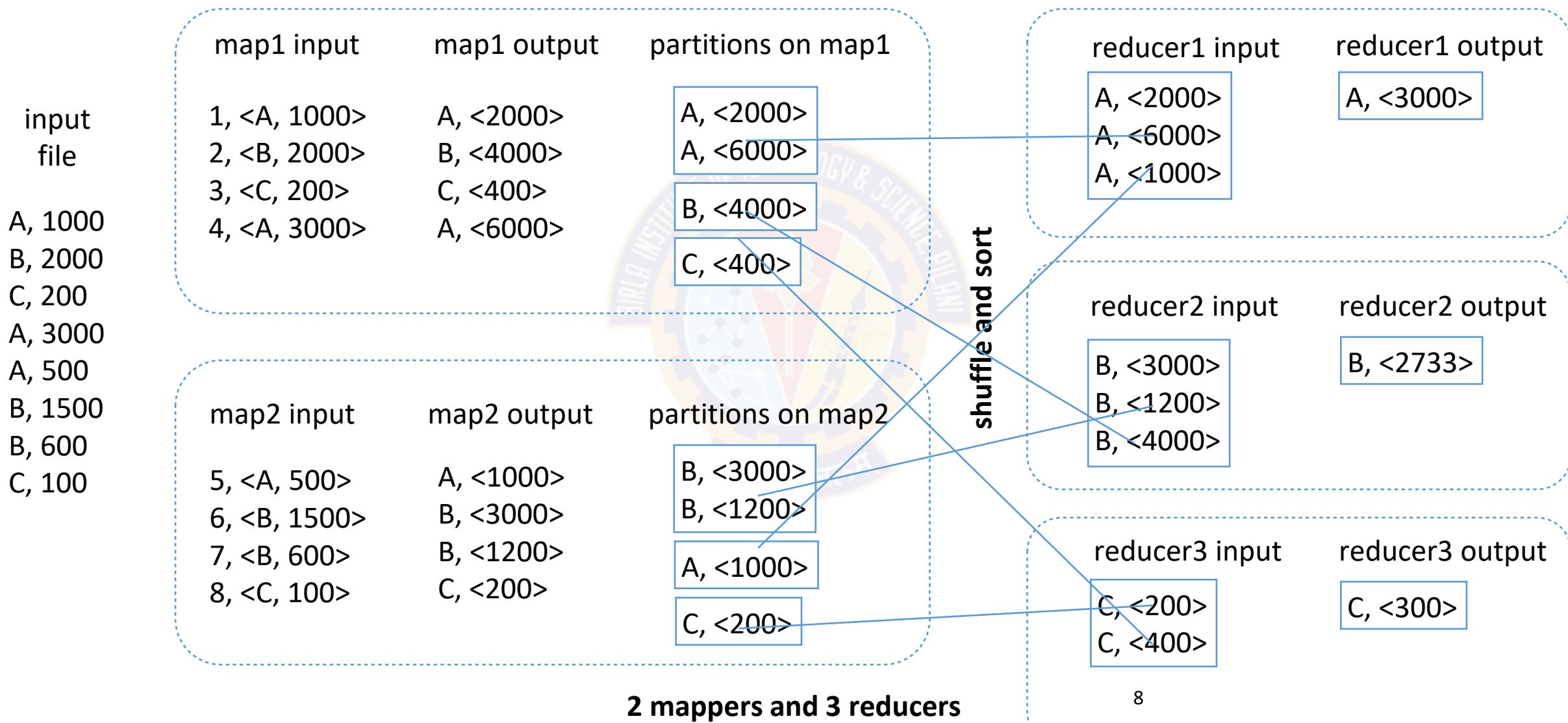
<https://stackoverflow.com/questions/22141631/what-is-the-purpose-of-shuffling-and-sorting-phase-in-the-reducer-in-map-reduce>

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

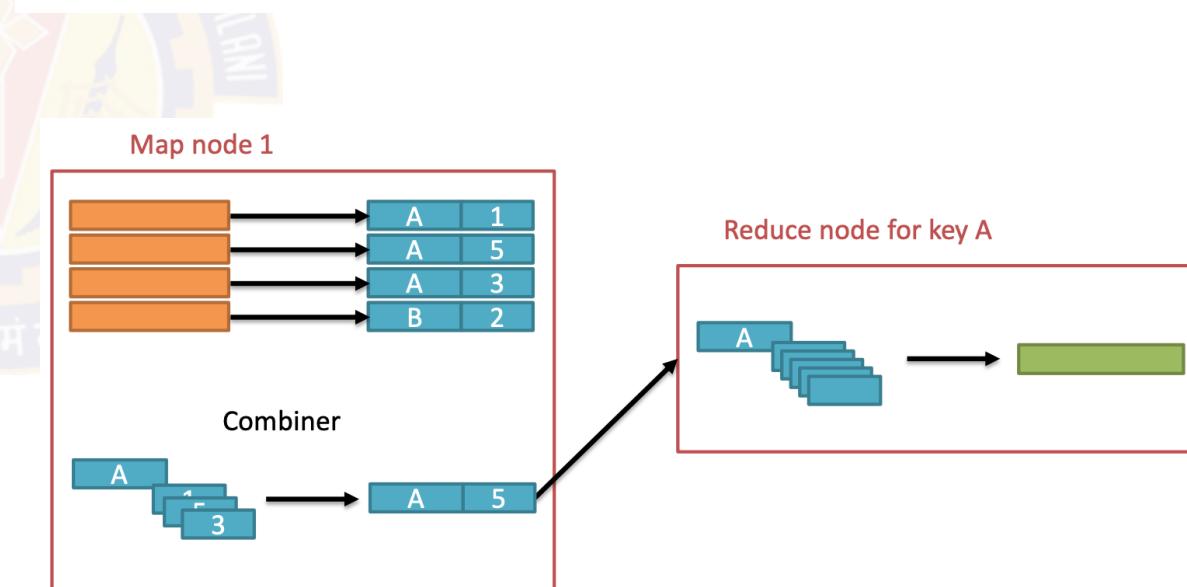
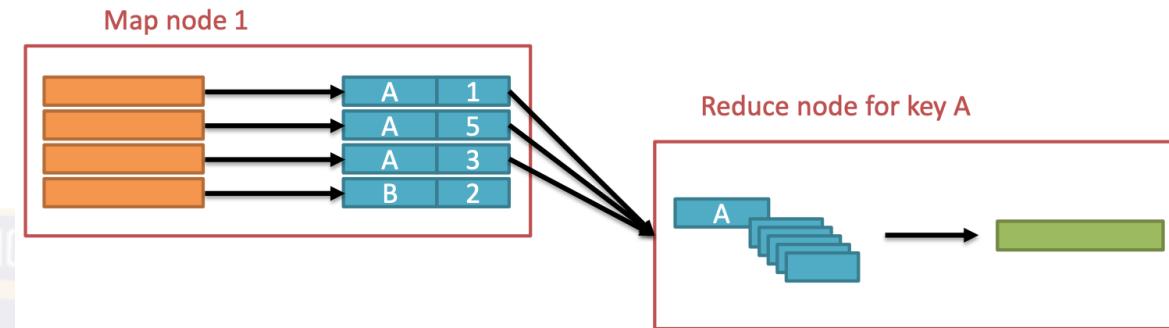
- Combiners can optimise the reduce by pre-processing on each node to compress data but output has to be same type as a map

- 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

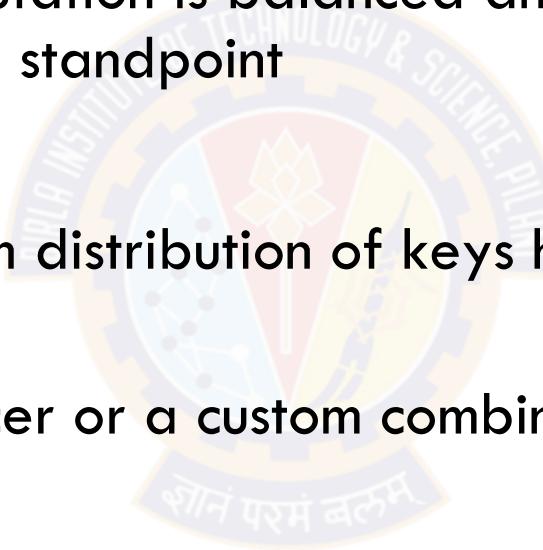
✓ $\text{avg}(1,2,\text{avg}(3,4,5)) = 2.33\dots$ but
 $\text{avg}(\text{avg}(2,4),\text{avg}(1,5,3)) = 3$

- 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

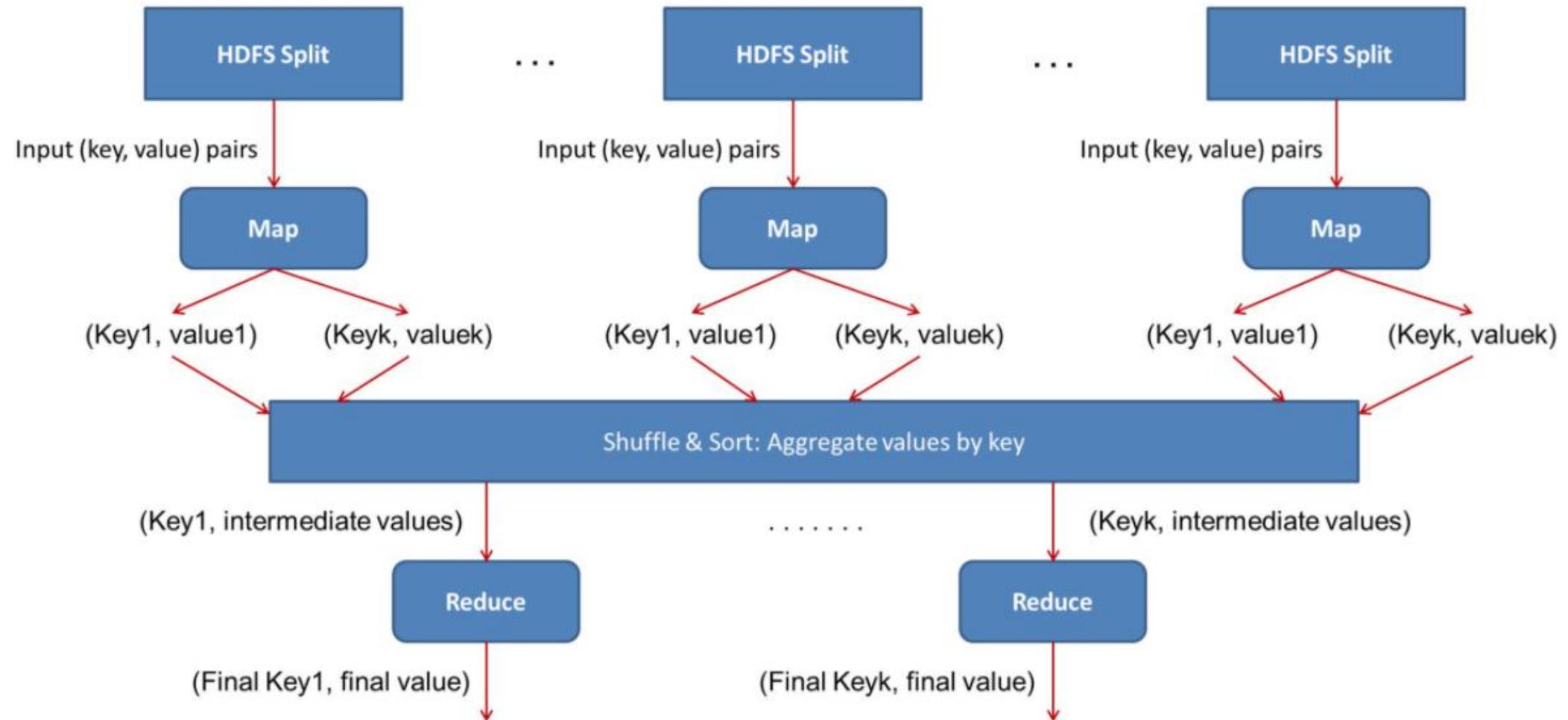


Image ref: <https://data-flair.training/blogs/hadoop-architecture/>

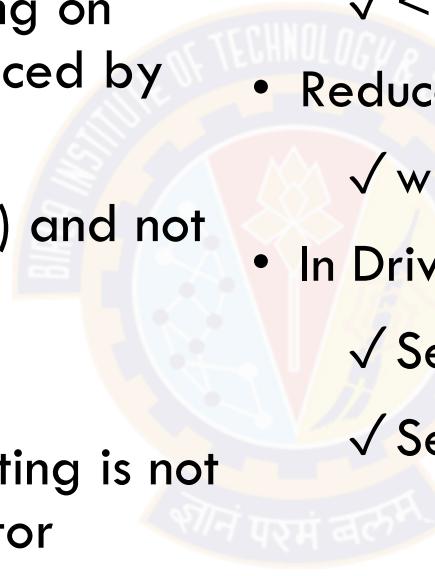
Topics for today

- 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

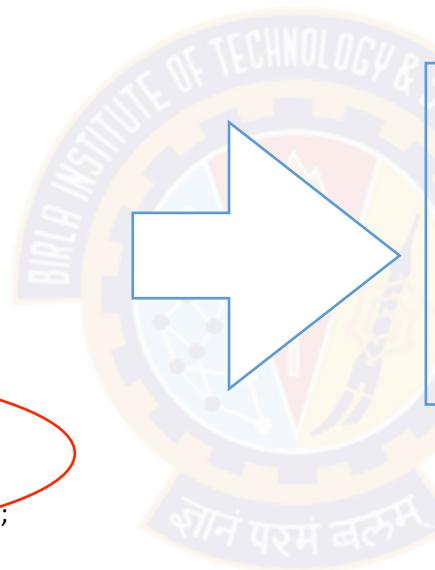


1971 Australia MIN	0.8412
1971 Australia MAX	0.899
1971 Austria MIN	23.638
1971 Austria MAX	25.873
1971 Belgium MIN	45.49
1971 Belgium MAX	49.73
1971 Canada MIN	0.9933
1971 Canada MAX	1.0248
1971 Denmark MIN	7.0665
1971 Denmark MAX	7.5067

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

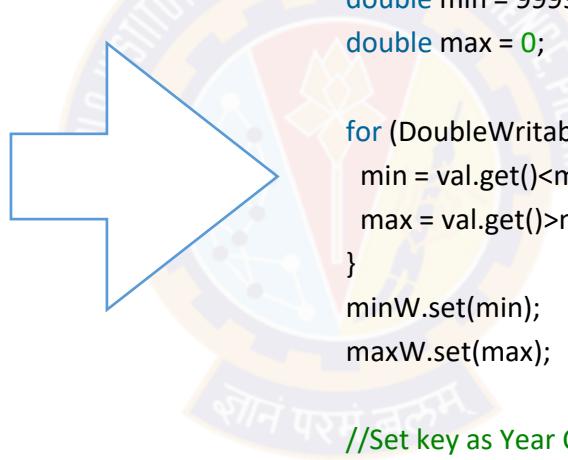
after shuffle-sort using key

```
<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);
```

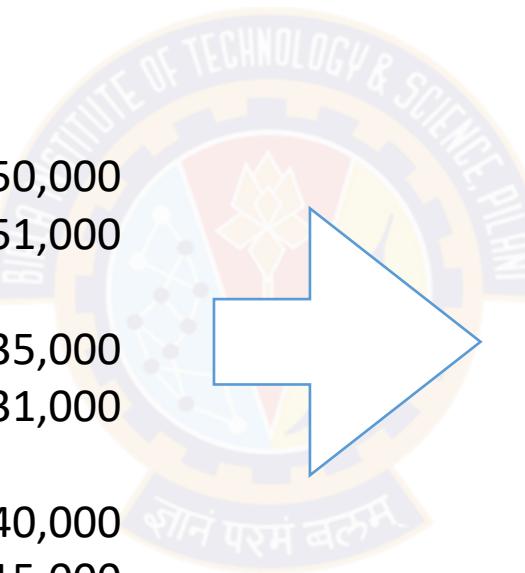
can you use a combiner here ? same logic as reducer ?

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

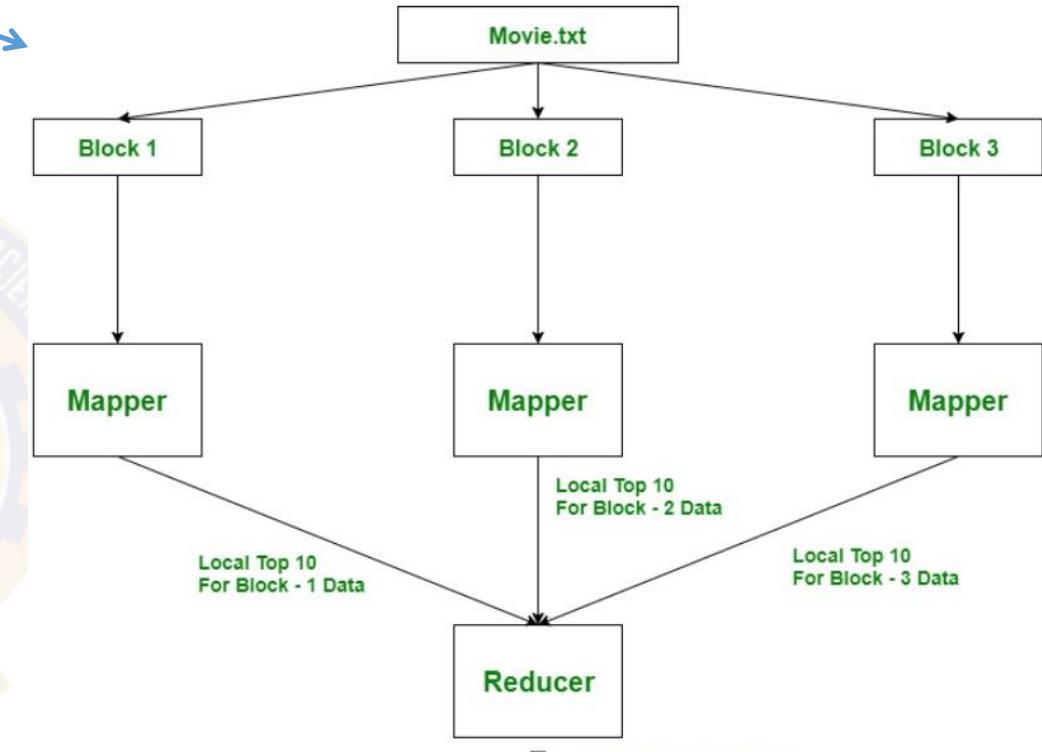
Find the Top 10 movies given <movie_name,#views>

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

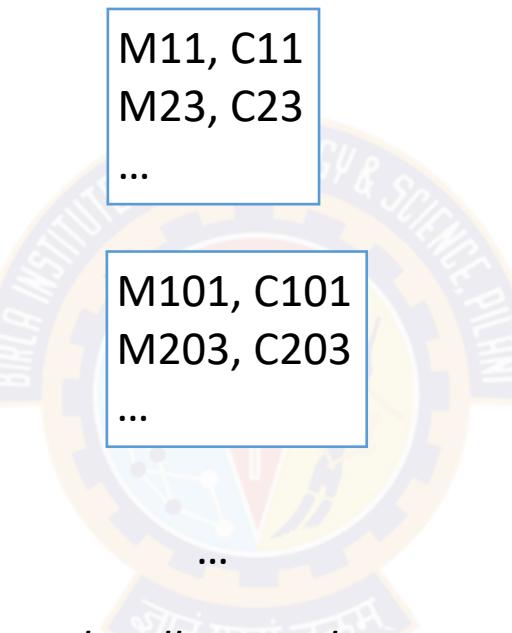
Find the Top N movies given <movie_name,#views>

```
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.

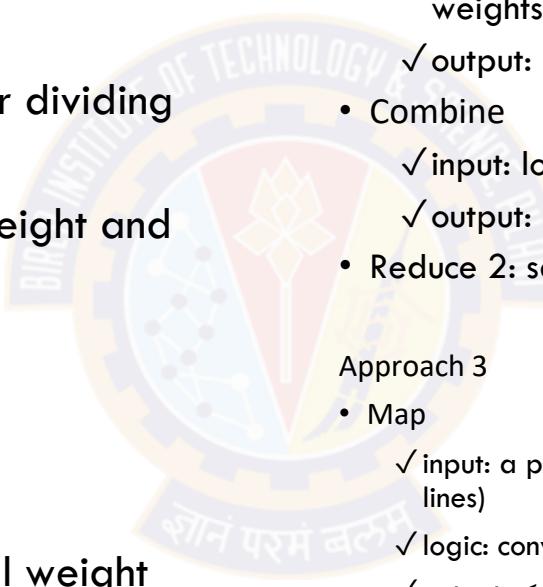
Approach 1

- 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

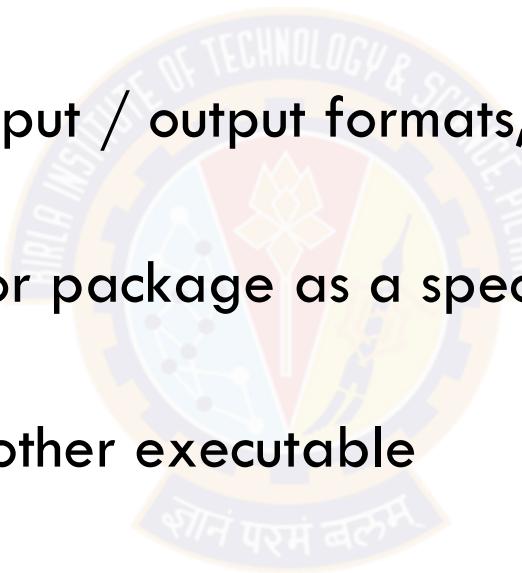
Advanced examples

- 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

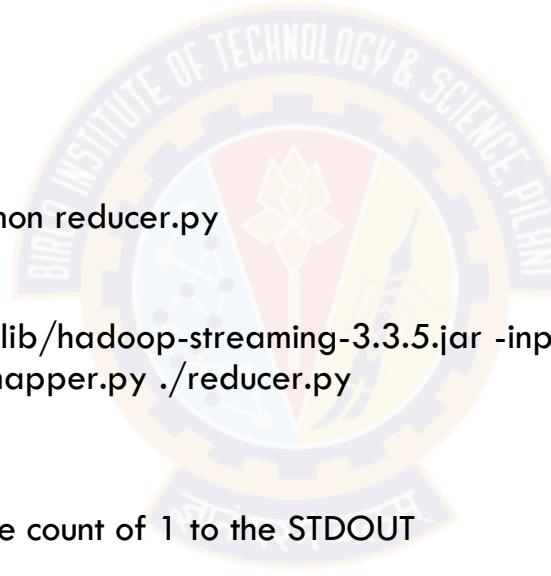
- Hadoop Streaming allows developers to use various languages for writing MapReduce programs
- It supports all the languages that can read from standard input and write to standard output
 - Python
 - C++
 - Ruby, etc

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/>

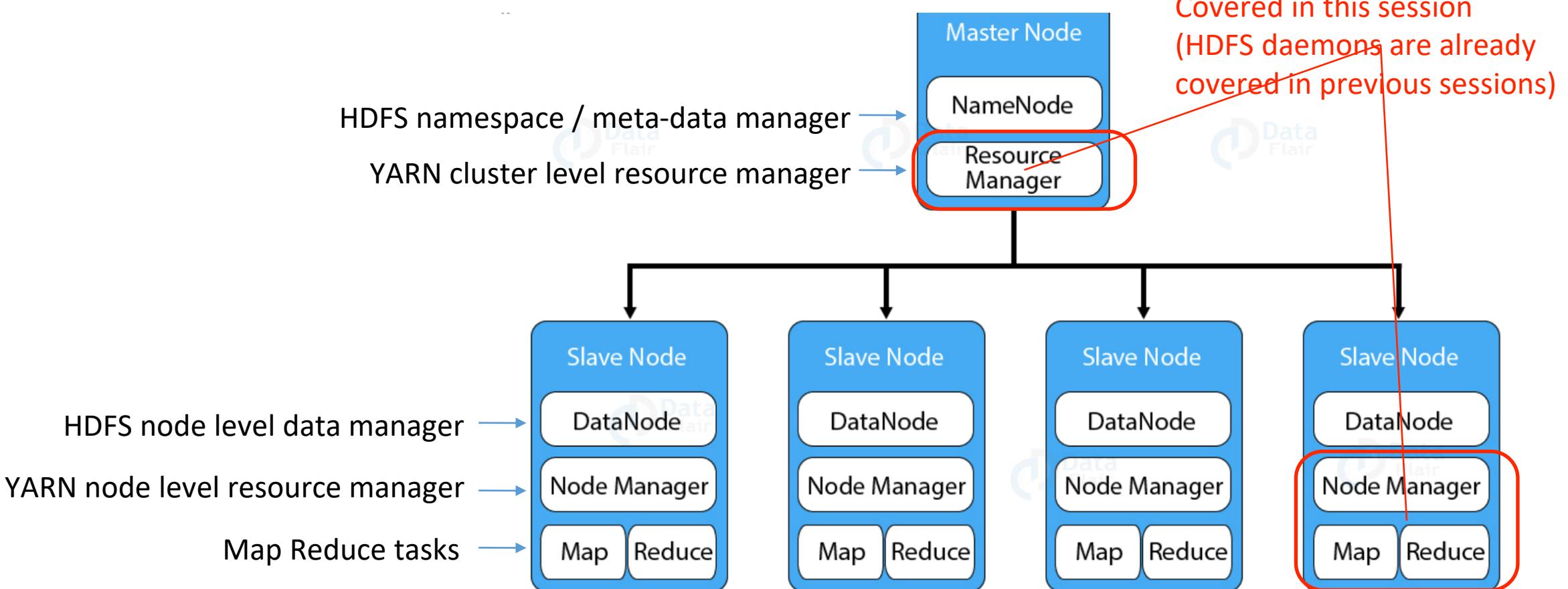
Topics for today

- 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



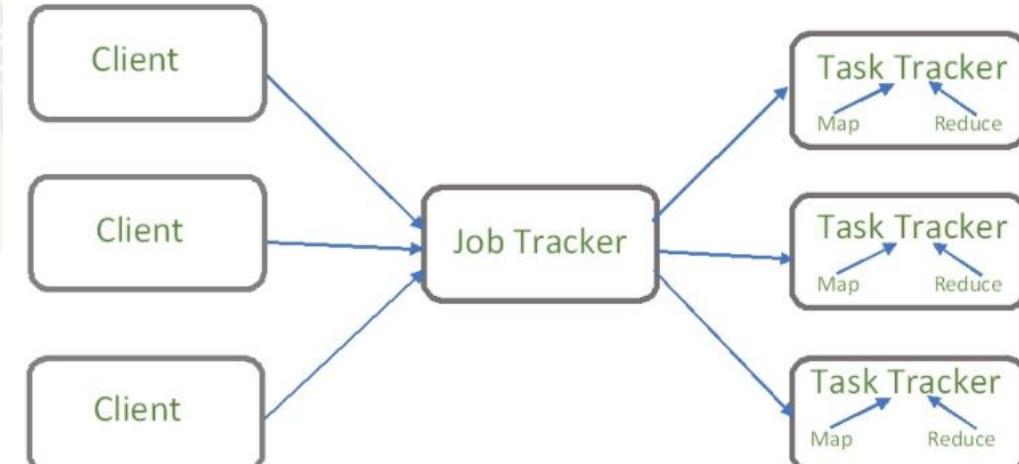
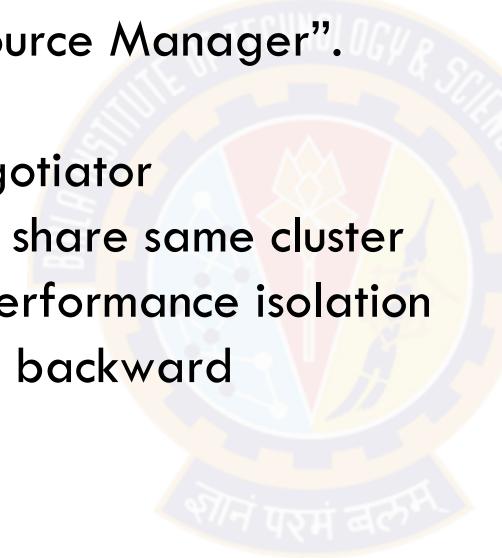
Note: YARN Resource Manager also uses application level App Master processes on slave nodes for application specific resource management

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

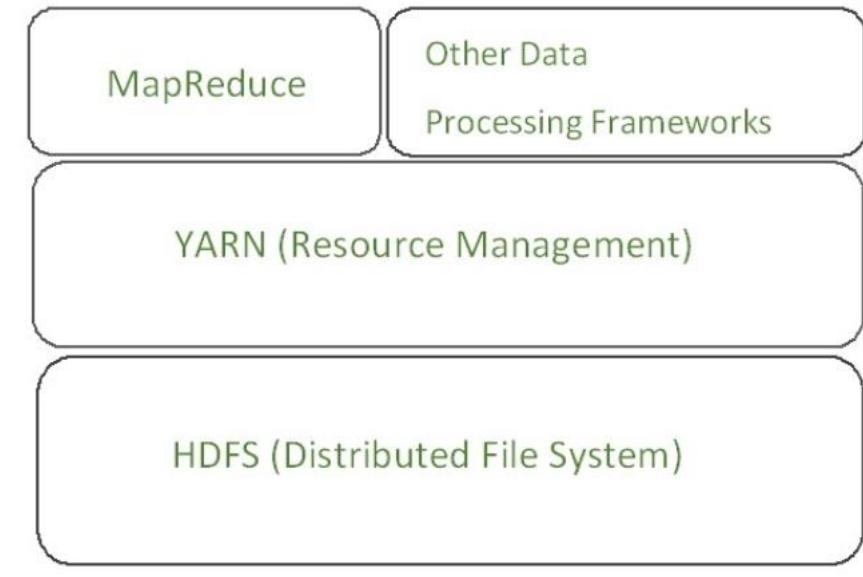
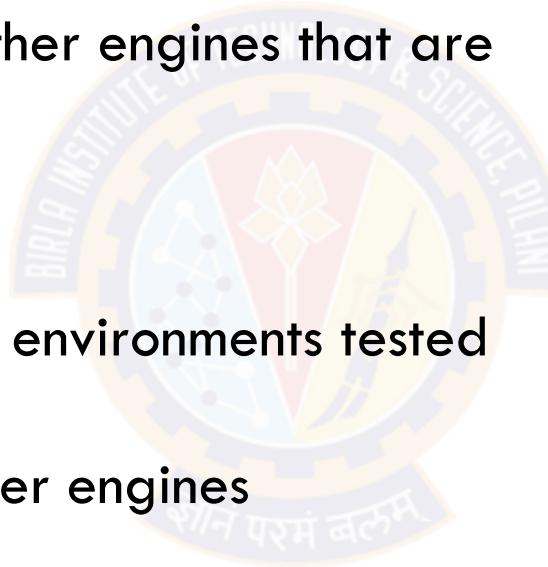


<https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

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

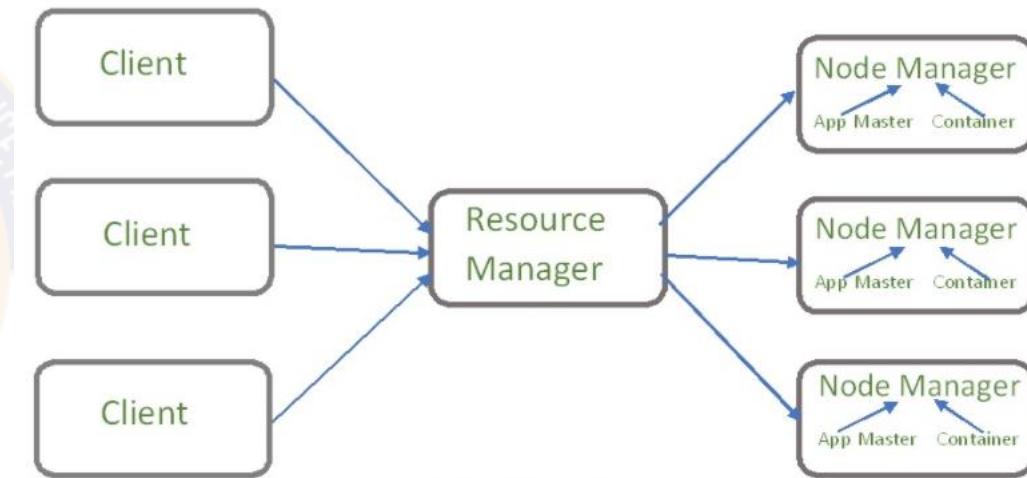
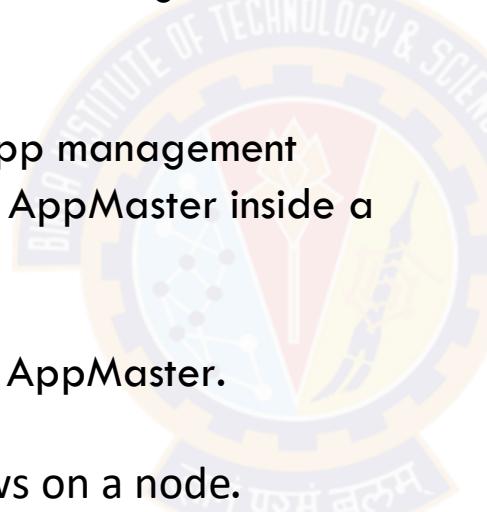
Where does YARN fit in Hadoop ?

- 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



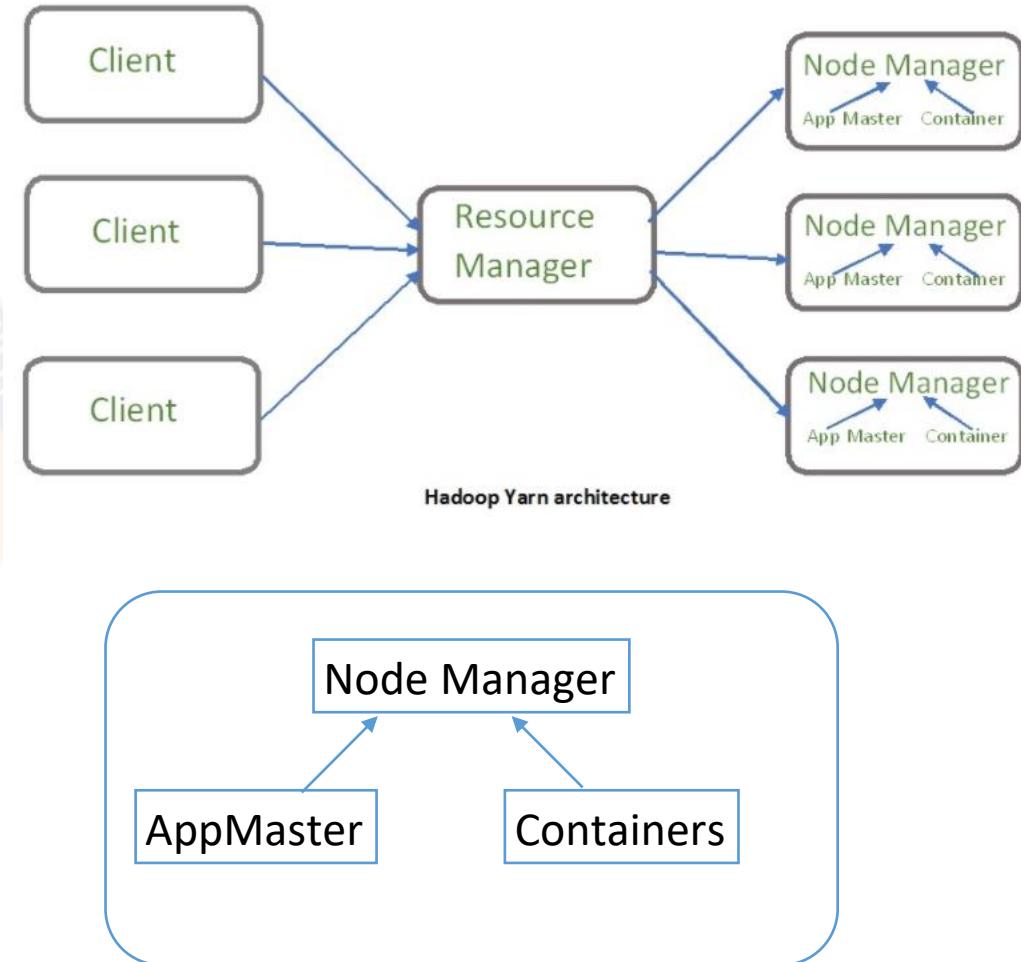
YARN components (1)

- 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



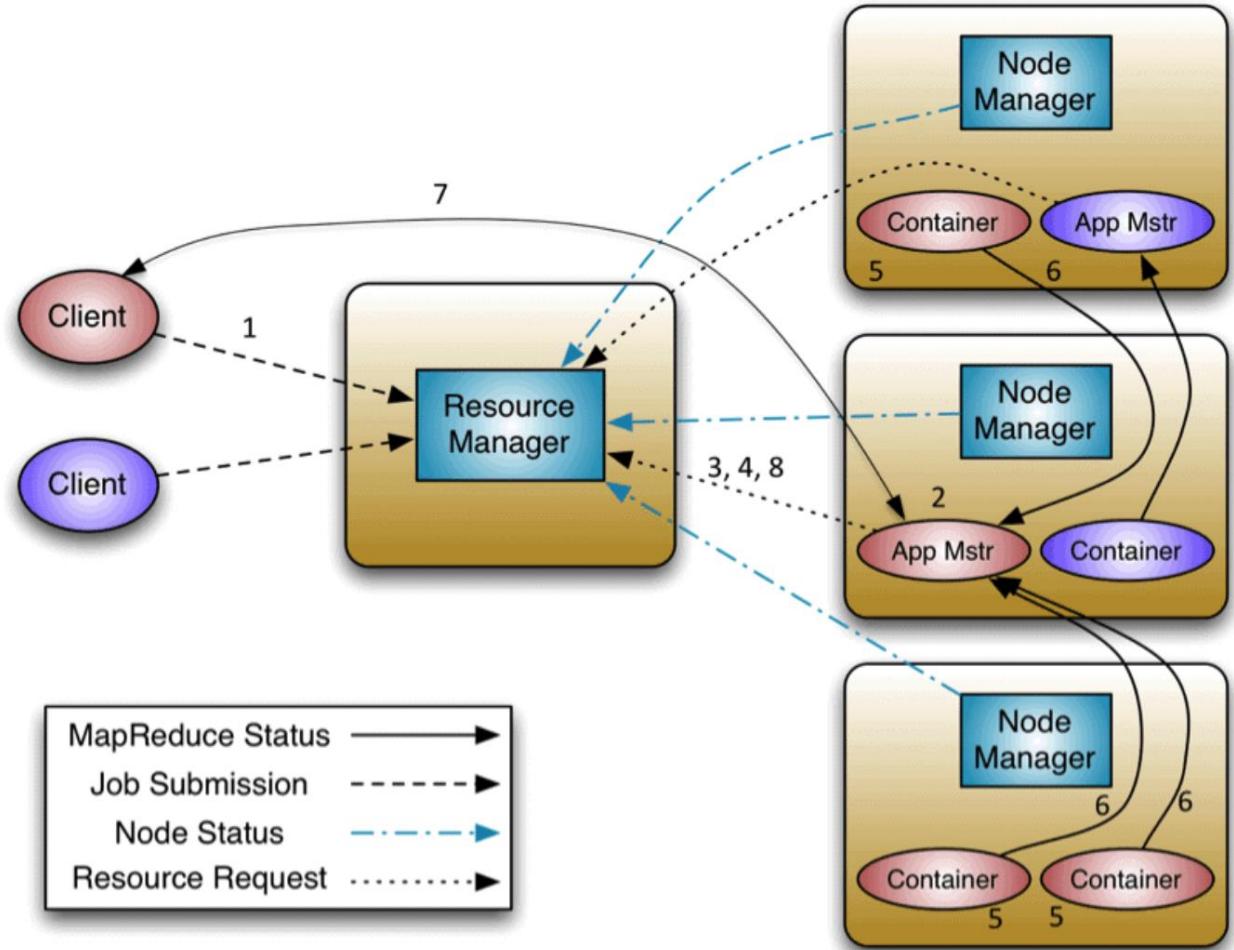
Topics for today

- 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



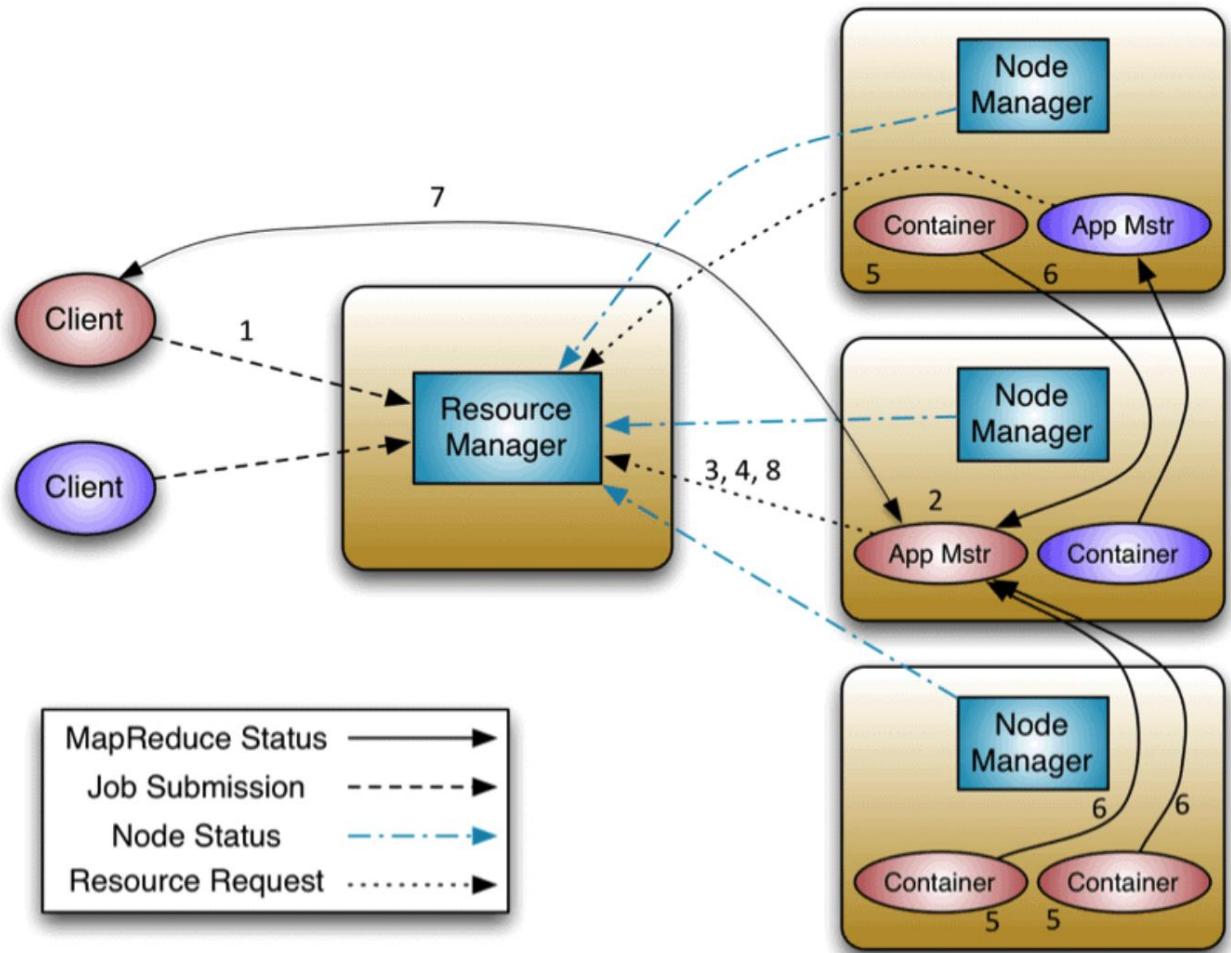
YARN workflow (1)

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.



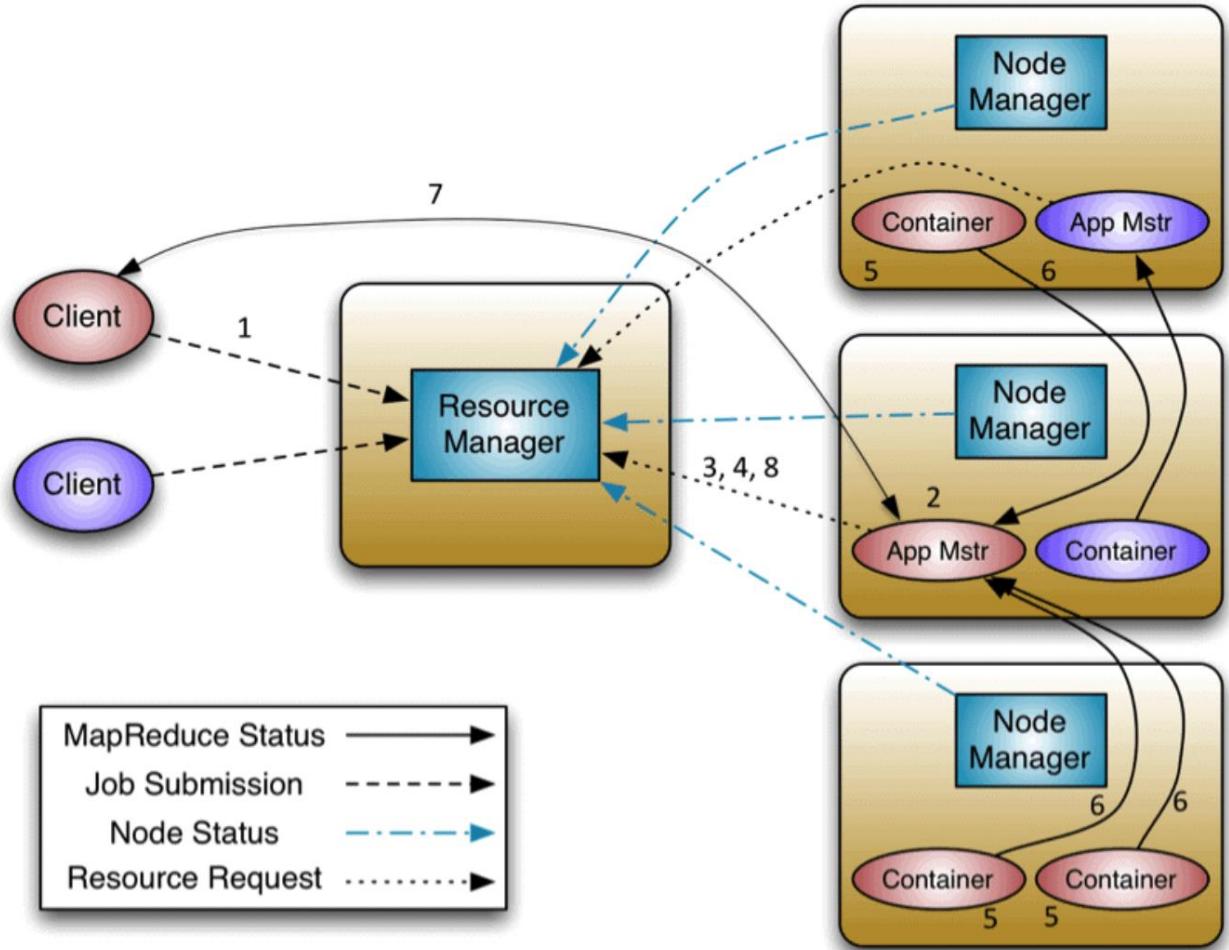
YARN workflow (2)

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.

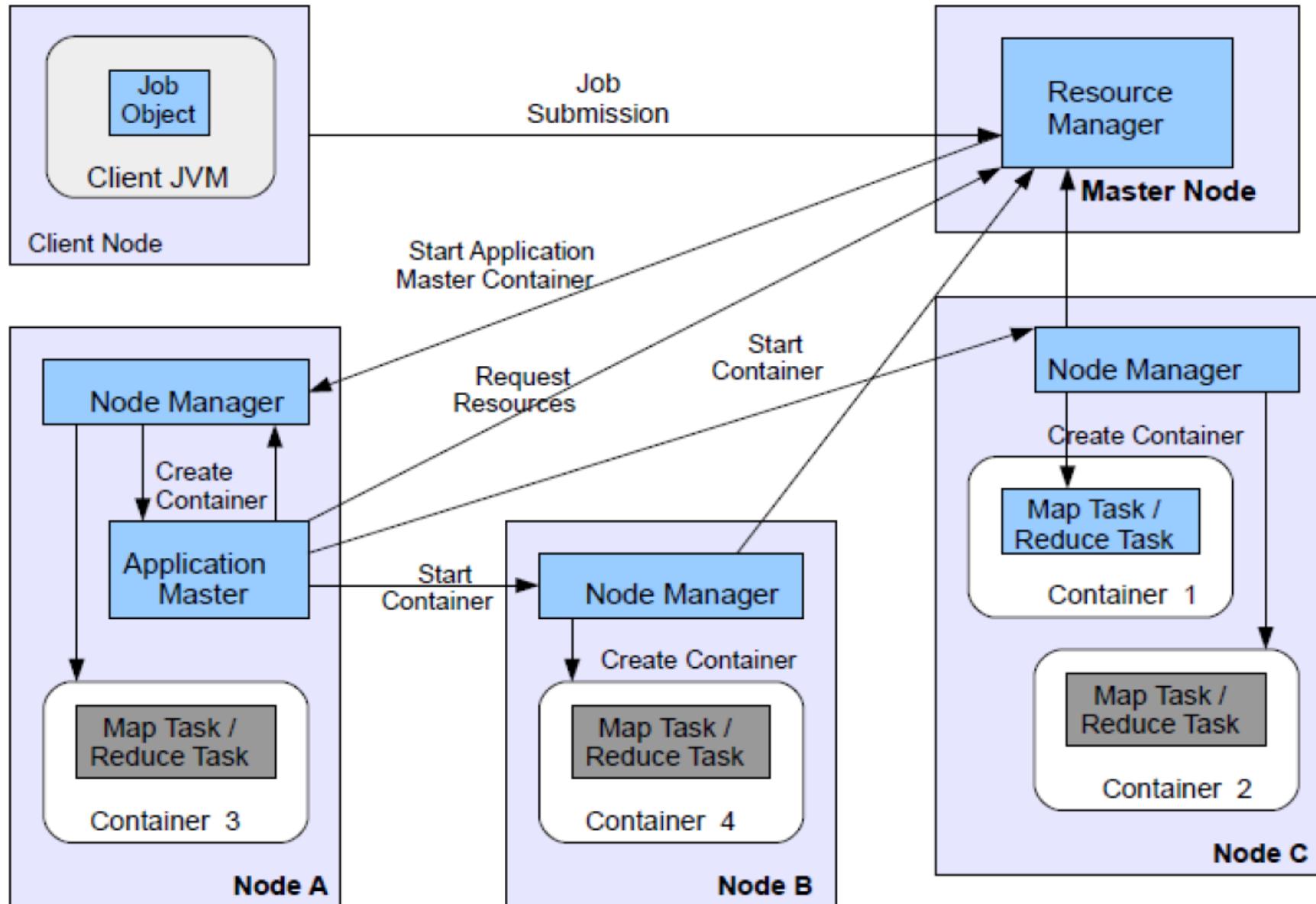


YARN workflow (3)

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.



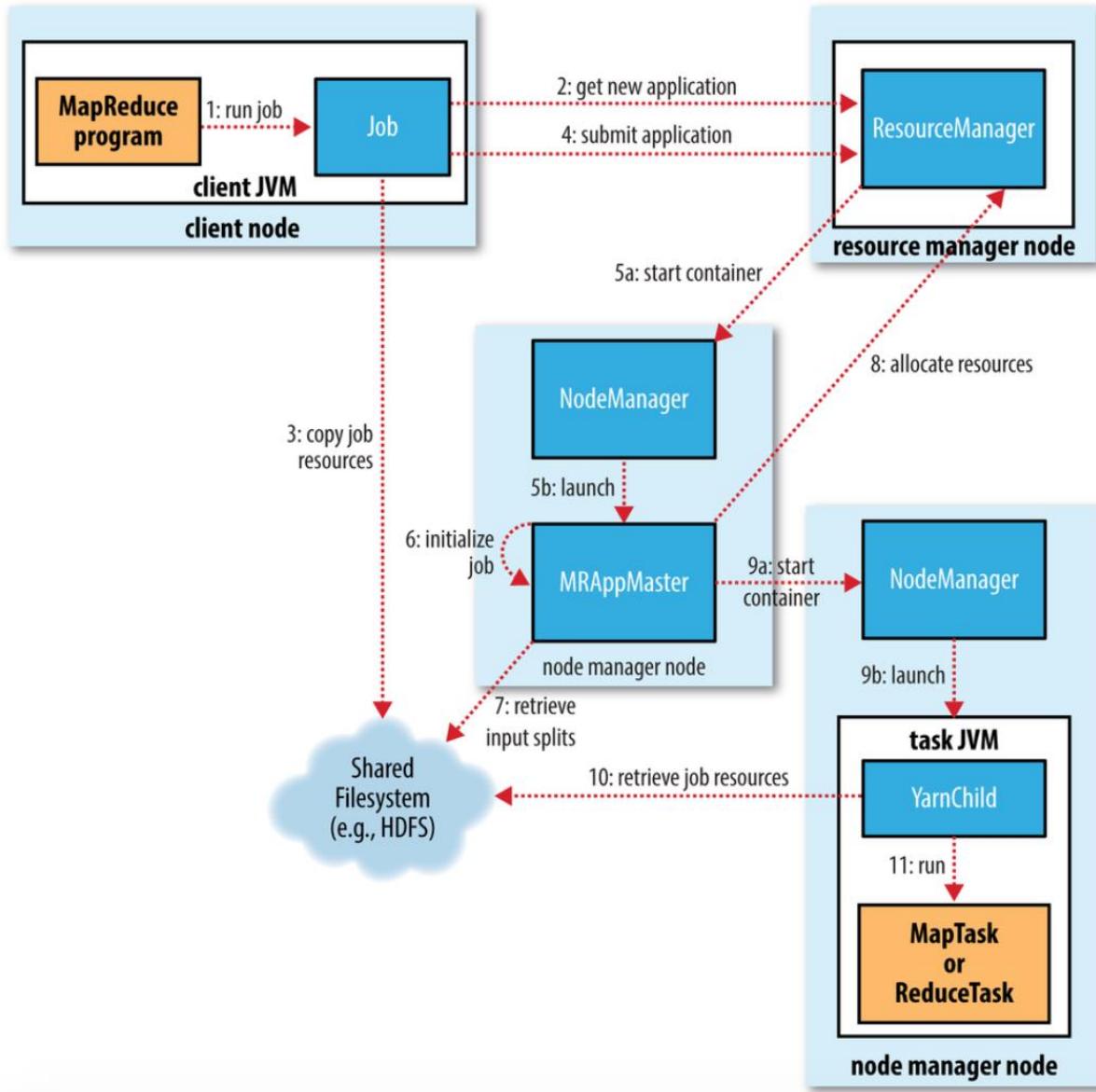
Running a Job on YARN – Container management



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>

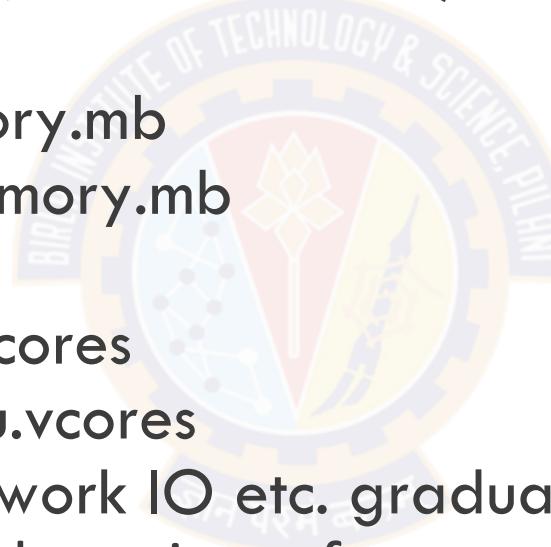
Topics for today

- 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 ≥ 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

Resource requests

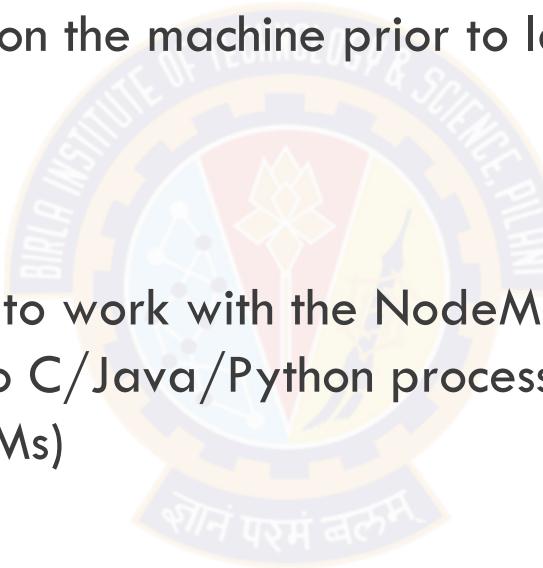
- A resource request :
`<resource-name, priority, resource-requirement, number-of-containers>`
resource-name is a host, rack or * for a container
priority is within application for this request
resource-requirement is CPU, memory etc. needed by a container
number-of-containers is count of above spec 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 ?

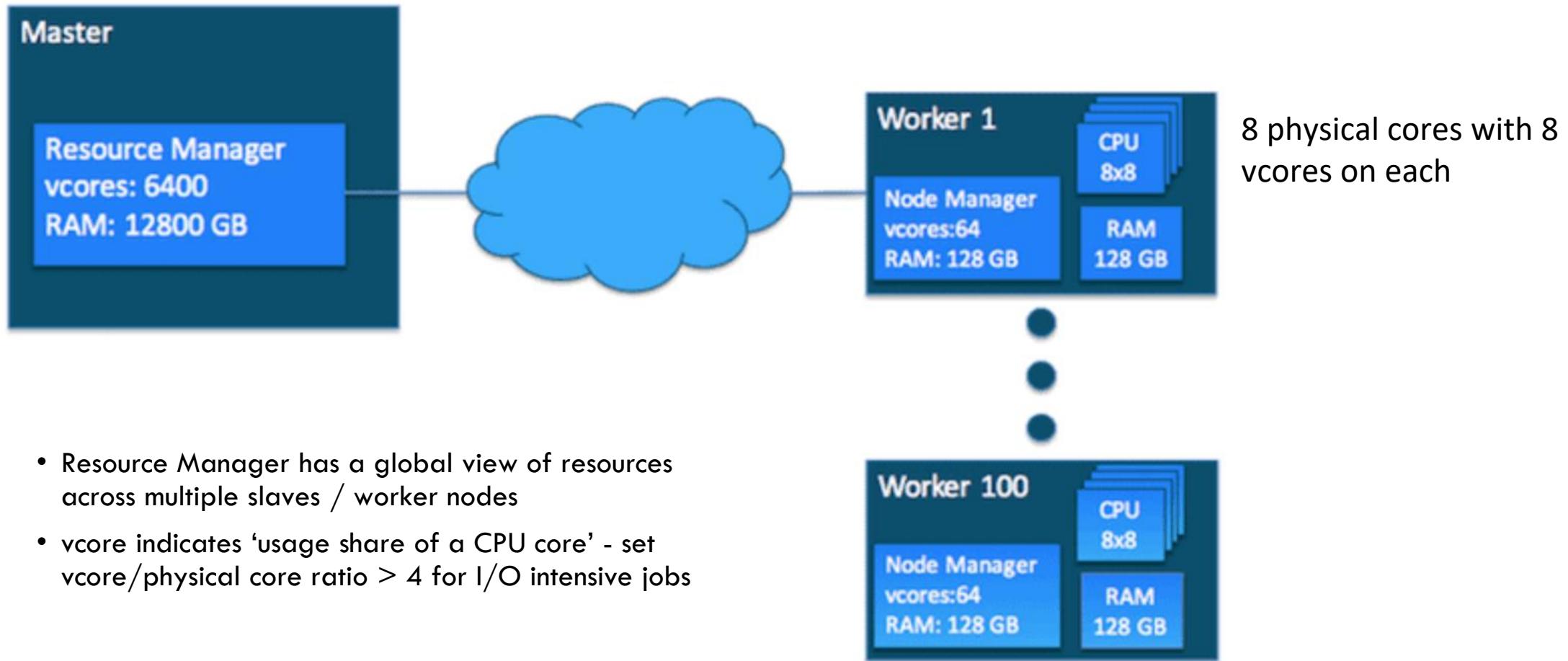
The YARN Container launch specification API is platform agnostic and contains:

- 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.

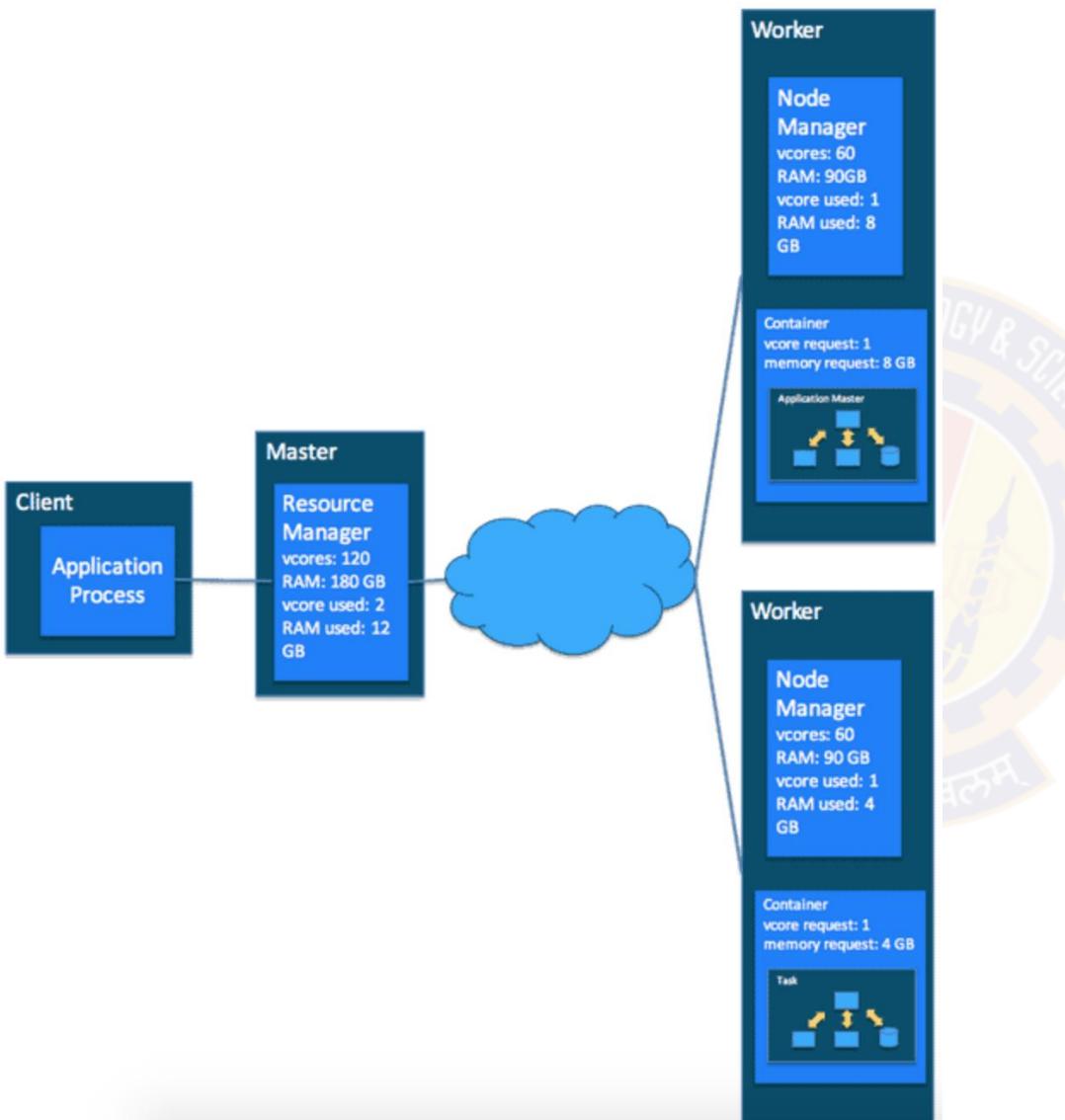
This allows the ApplicationMaster to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged virtual machines (e.g. KVMs)



Example: Global resource view

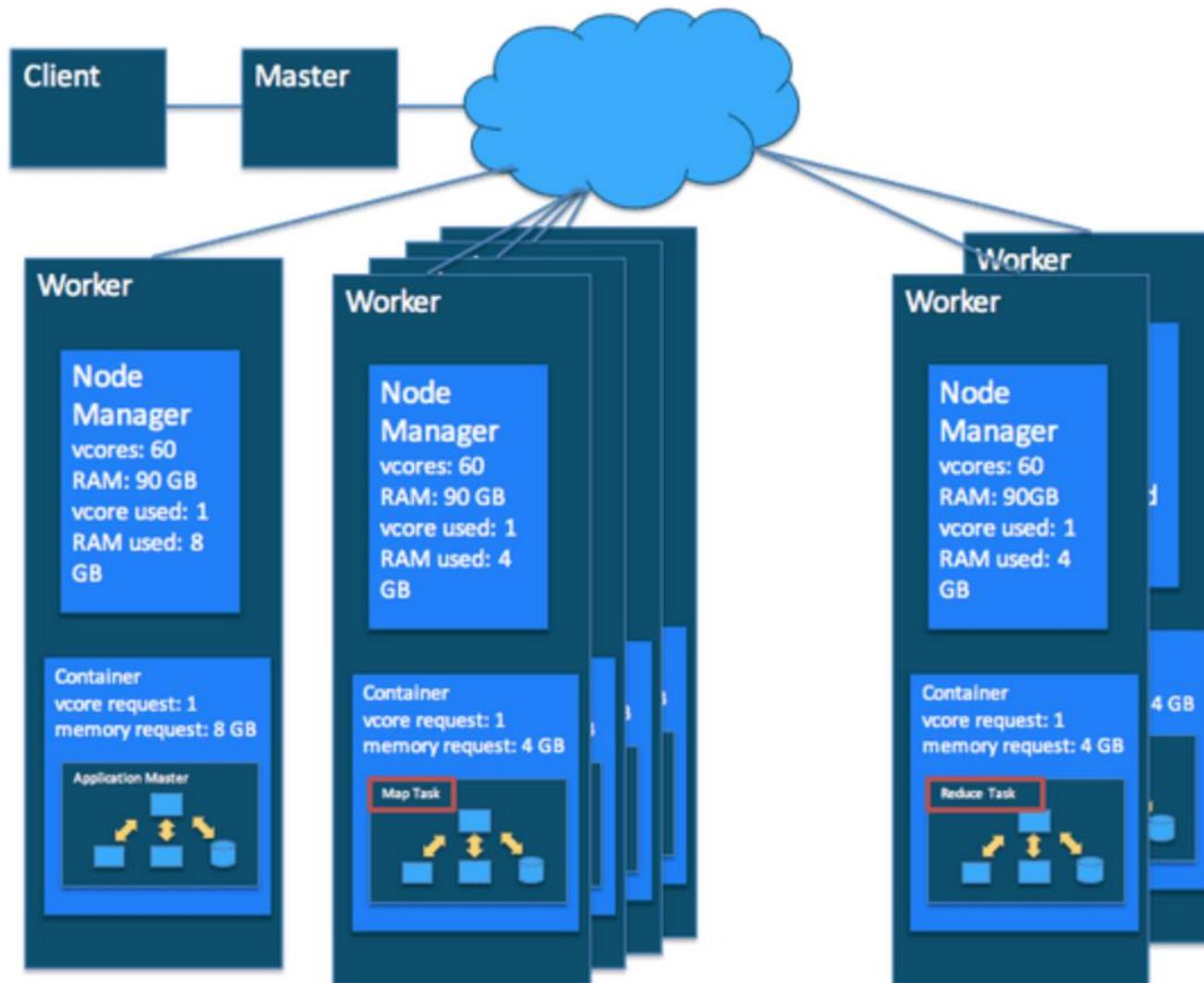


Example: Containers on Workers



- 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

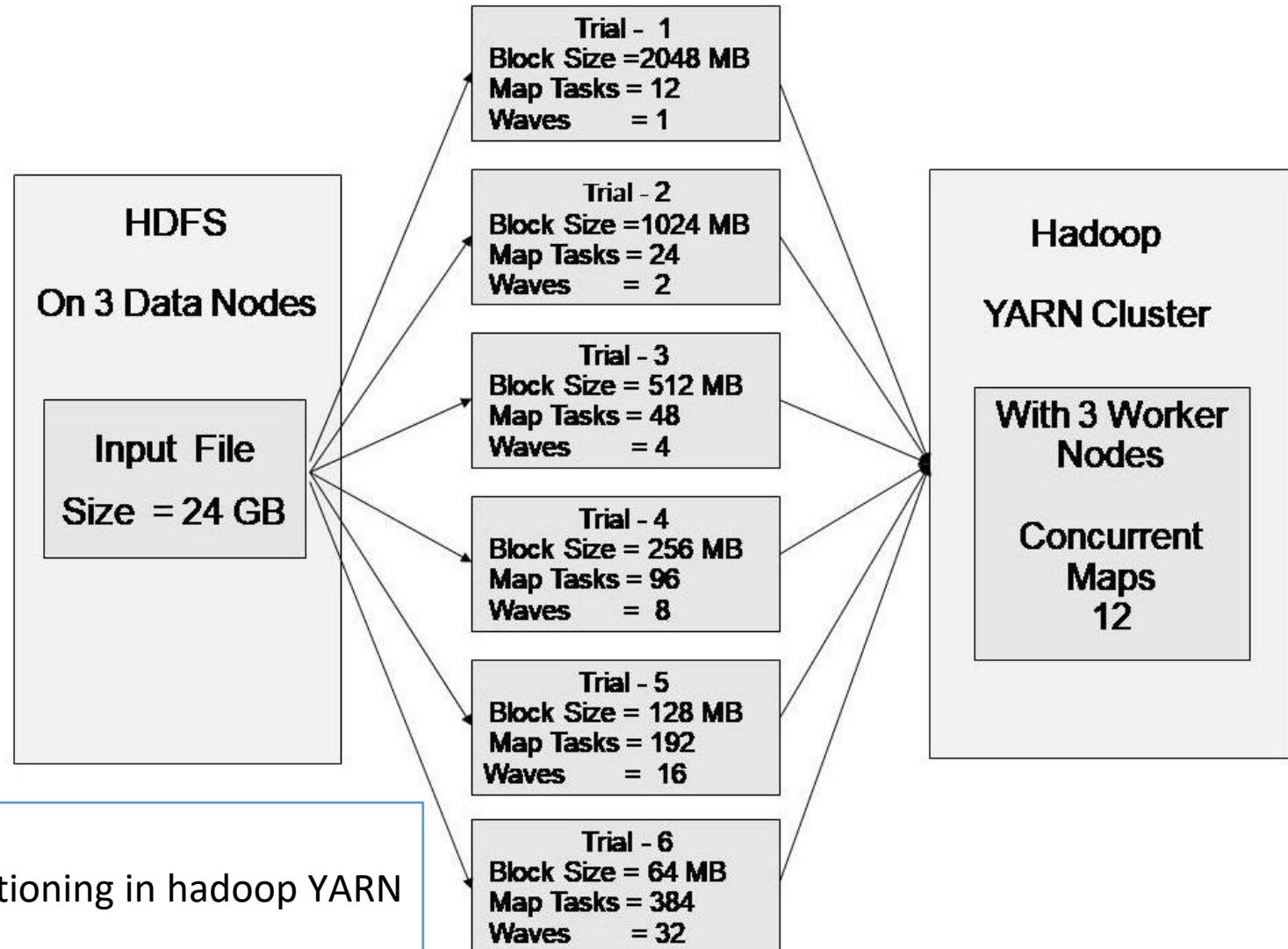
Example: Map and Reduce tasks in containers



- 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

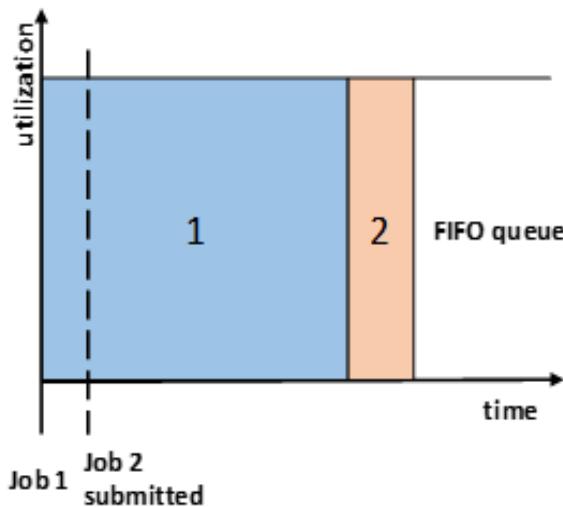


Schedulers in YARN

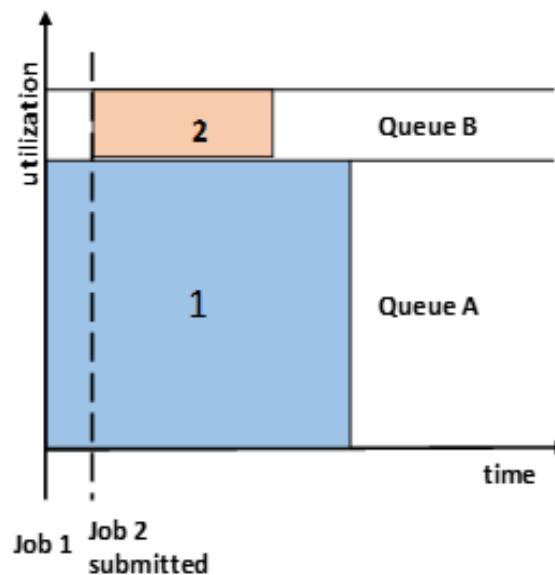
- FIFO – First in, First Out
- Capacity - Maintains separate queue for different types of jobs (Default on Hadoop-3.3.5)
- Fair share scheduler (Fair scheduler) - Dynamically balances resources into all accepted jobs

These are pluggable to YARN, Any one at a time

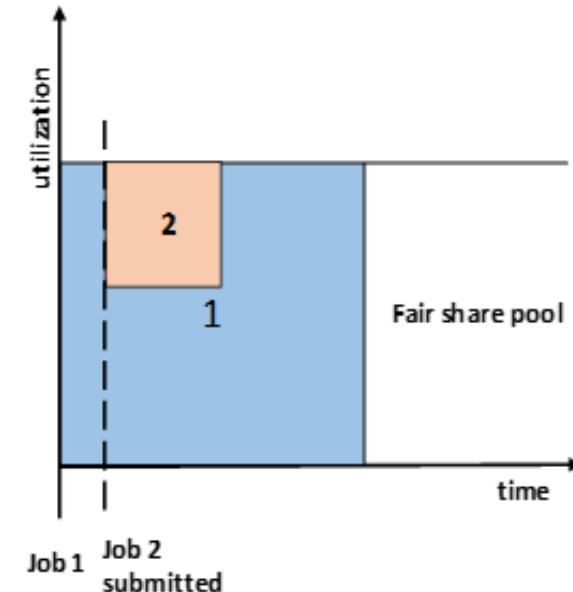
Source: [Hadoop: The Definitive Guide](#)



(a) FIFO scheduler



(b) Capacity Scheduler

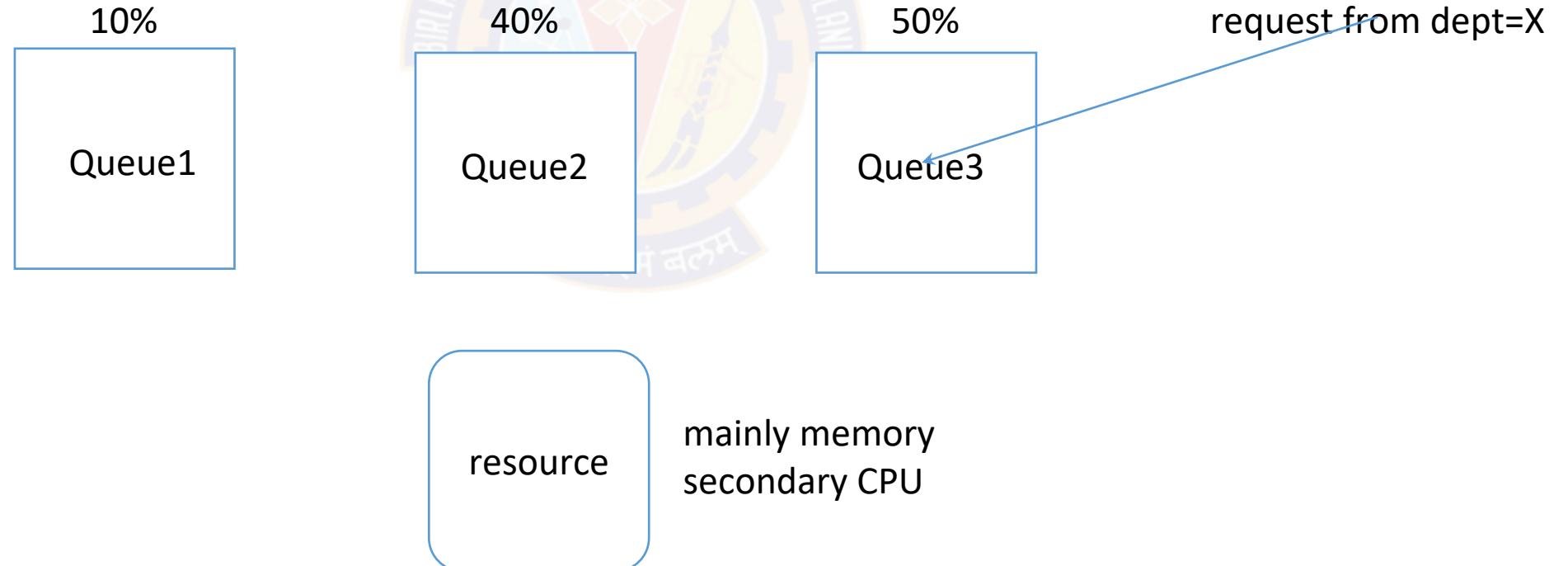


(c) Fair Scheduler

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`



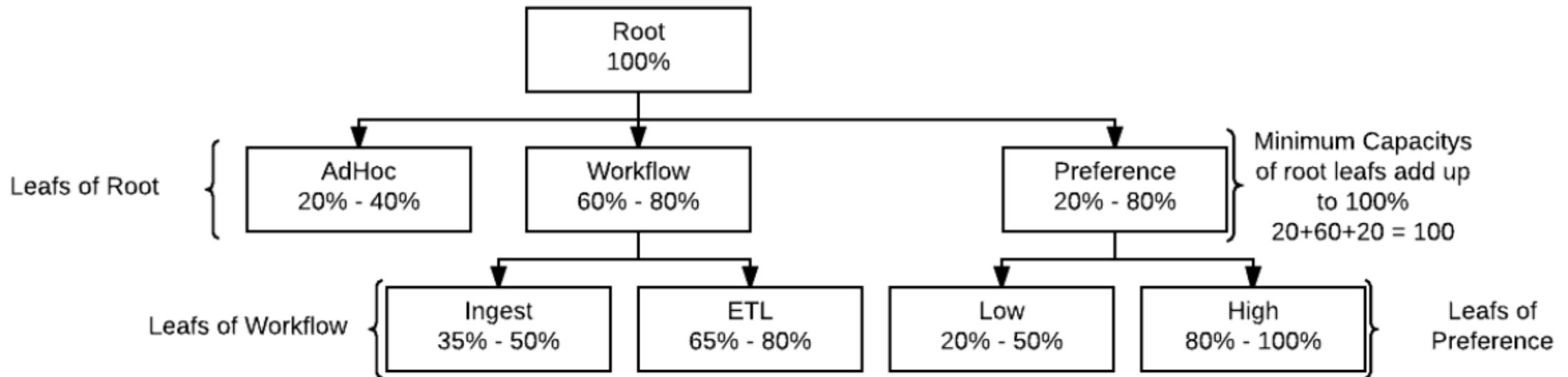
Resource Manager scheduling - Capacity scheduler

- 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



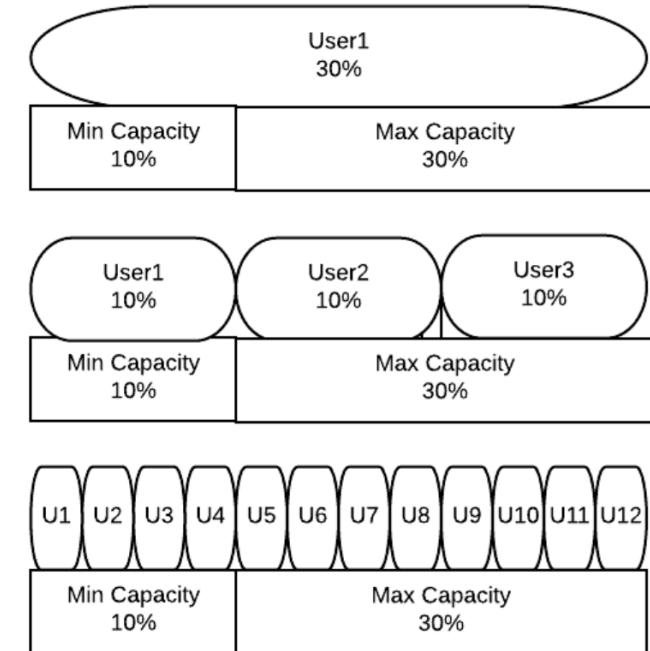
Capacity scheduler : User level allocations

- 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

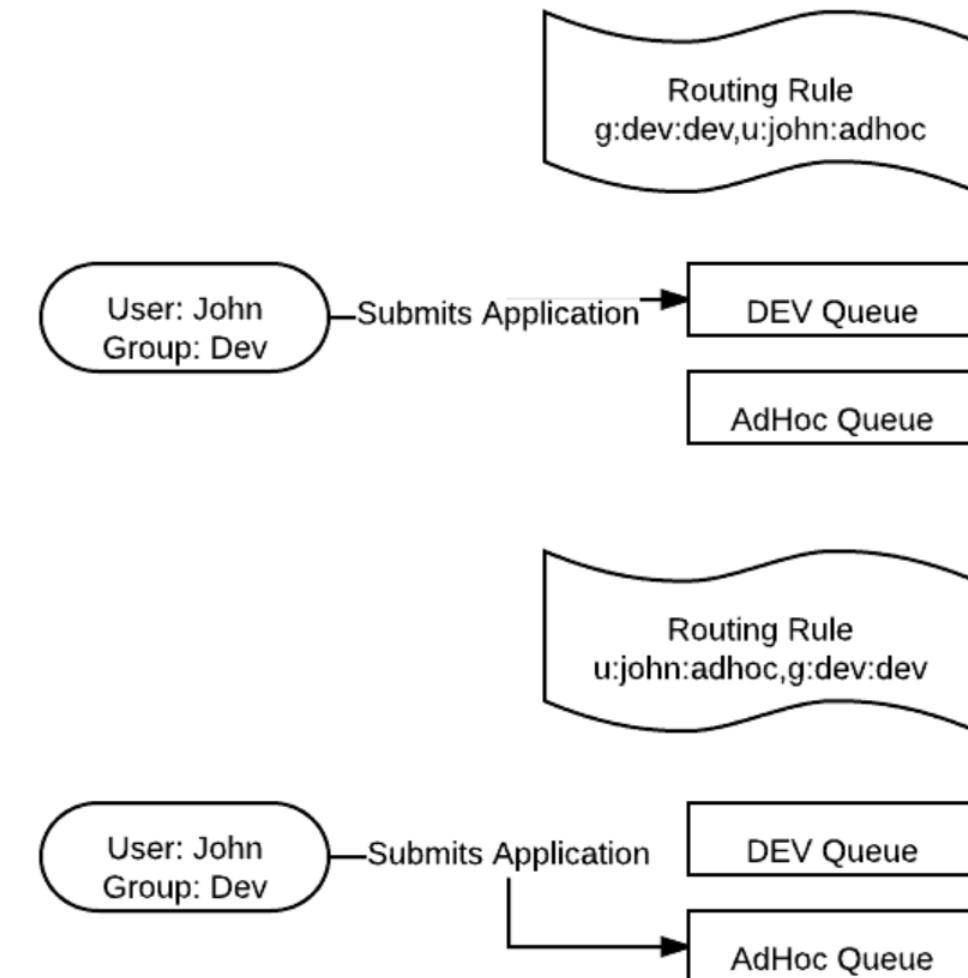


Capacity scheduler : Ordering policies within queue

- 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

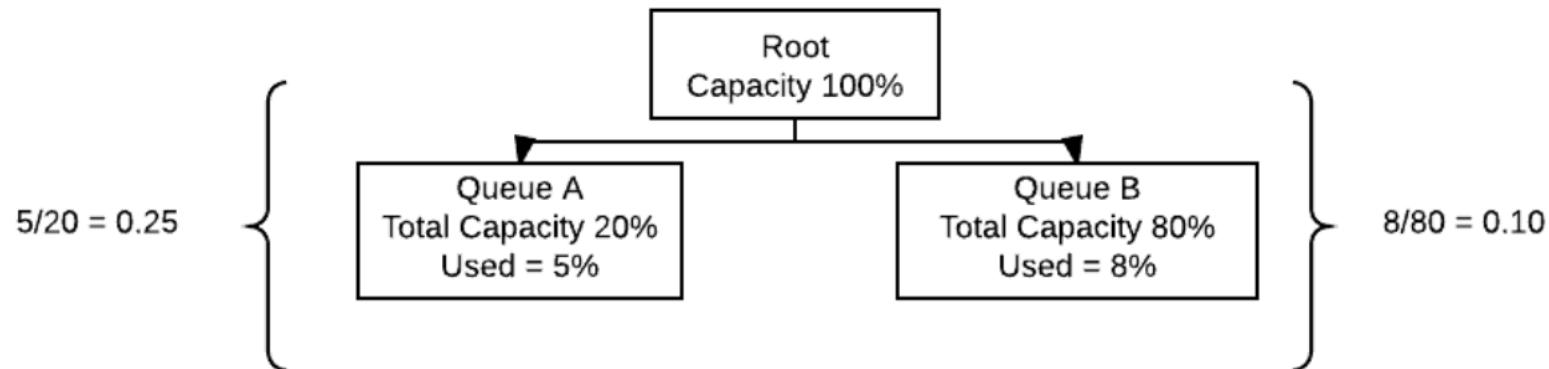
Capacity scheduler : Mapping to queue

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



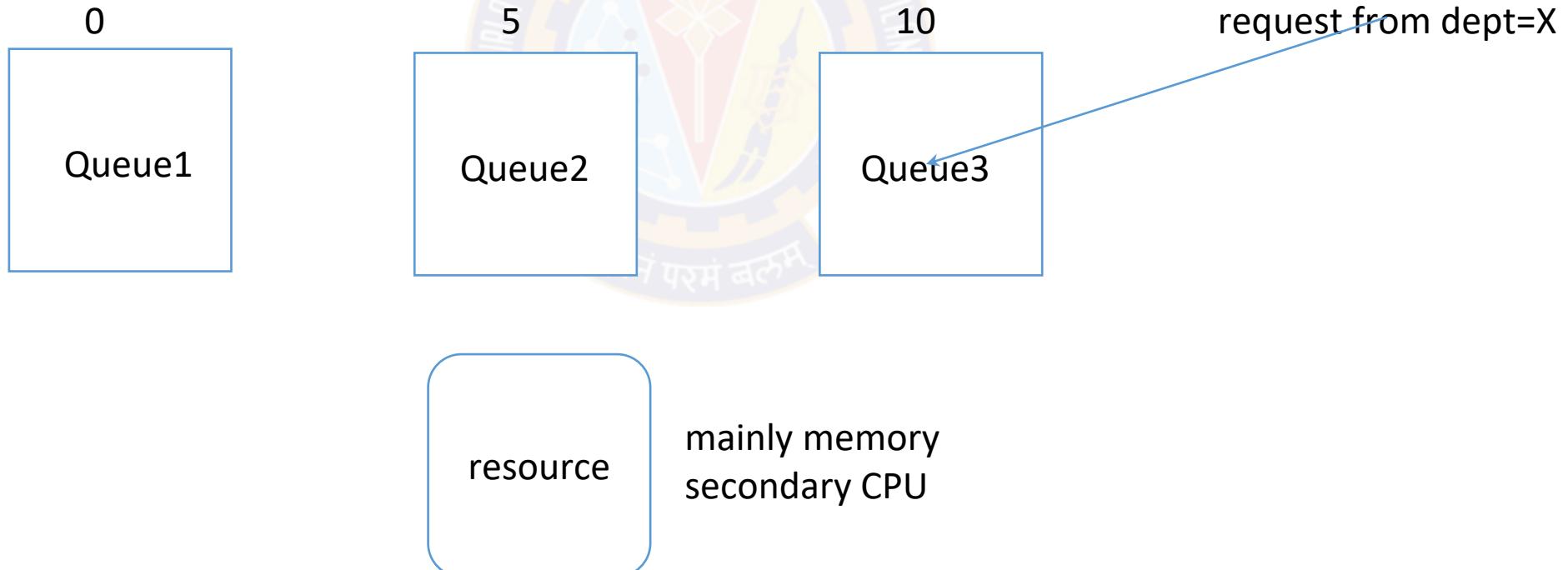
Capacity scheduler : Priority

- 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.

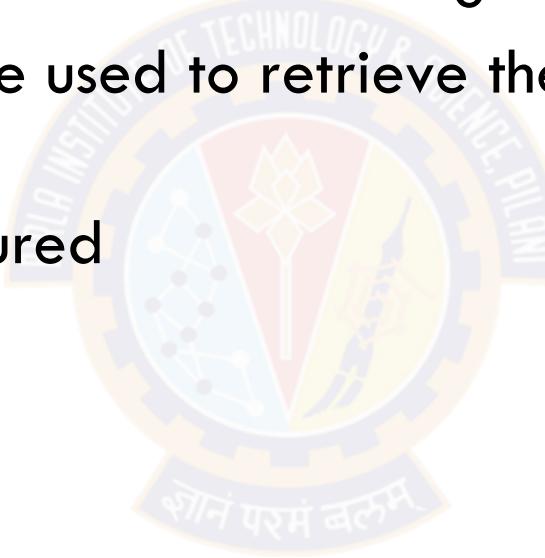
```
<allocations>
  <queue name="queue1">
    <fairSharePreemptionTimeout>60</fairSharePreemptio
nTimeout>
    <fairSharePreemptionThreshold>0.80</fairSharePreem
ptionThreshold>
      <weight>1.0</weight>
    </queue>
    <queue name="queue2">
      <weight>1.0</weight>
    </queue>
  .
  .
</allocations>
```

Difference between Capacity scheduler and Fair scheduler

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



Study of Execution Parallelism by Resource Partitioning in Hadoop YARN

Janardhanan PS

*SunTec Business Solutions Pvt.Ltd
Tejaswini, TechnoPark
Trivandrum-695581, India
janardhananps@suntecgroup.com*

Philip Samuel

*Head of Information Technology, SOE
Cochin University of Science and Technology
Kochi-682022, India
philips@cusat.ac.in*

Abstract— Hadoop YARN is a widely used distributed computing framework mainly used for big data processing. Yet Another Resource Negotiator (YARN) was introduced in Hadoop 2.0 to solve the scalability issues of the earlier realization. Hadoop YARN in combination with Hadoop Distributed File System (HDFS) can be considered as a distributed operating system with capabilities similar to that of the proprietary Tandem Nonstop kernel. Now it is possible to use YARN as a general purpose framework for distributed computing. When a job is divided into tasks and distributed among the nodes of a cluster and further distributed on different CPU cores of the nodes, parallelism in the execution of tasks plays a major role in the completion time of jobs. In distributed computing frameworks, a global scheduler distributes the tasks of the workloads among the nodes and the local schedulers manage the tasks submitted to the nodes. Hadoop YARN partitions the system resources into containers and launches the tasks in them. YARN does not provide a single configuration parameter to define the number of containers that will be deployed concurrently on the nodes of the cluster. The number of concurrent containers depends on the fraction of the resources allocated to them. If the resources are divided with coarse granularity, the number of containers that can run in parallel will be limited. If the resources are divided with fine granularity, a larger number of containers can be run in parallel. When a Hadoop job is deployed on cloud platforms, it is required to select adequate resources for the platform to meet the deadlines in execution time. To guarantee maximum performance with minimum resources, we need to configure the YARN framework for optimum concurrency level in execution. This paper studies how to control the parallelism in execution of tasks by controlling the number of concurrent containers and how the execution time of jobs depends on the concurrency level of tasks.

Keywords—Distributed Computing, Parallel processing, Resource Partitioning, Hadoop YARN, MapReduce

I. INTRODUCTION

In distributed computing, jobs are split into tasks and distributed across multiple nodes of a cluster for performance. The performance is decided by the job granularity and optimum level of concurrency in execution. It is always a challenge to determine whether to go for small number of large chunks of computation or large number of small chunks of computation. Hadoop 2.0 with YARN is a commonly used distributed computing framework for parallel processing of

huge volumes of data. In the work reported in this paper, we evaluated the execution parallelism of tasks belonging to a job on a Hadoop YARN cluster and its dependency on the job completion time. YARN launches tasks on resource partitions called containers and the performance of the application depends on ideal resource partitioning mechanism [2]. When jobs are deployed on Hadoop clusters on cloud in IaaS model, we need to procure adequate resources in cloud infrastructure to meet the SLAs in execution time. To effectively utilize the resources in the cloud platform, we need to configure the Hadoop distributed computing framework for maximum performance by choosing optimum level of concurrency.

This paper starts with the problem statement. To split a job into tasks and run at different concurrency levels, we used a job developed using MapReduce programming paradigm and a small description of MapReduce is given in section III. It is followed by a section on description of the test environment. The resource management mechanism implemented in YARN is fairly complicated and it is very essential to understand the YARN internals. In this section, the details of system configuration, Hadoop YARN architecture and Hadoop cluster configuration are given. Section V describes the issues in container sizing and it is followed by a section on configuration details for resource partitioning of Hadoop YARN. Section VII is on partitioning resources into containers for desired level of concurrency in execution. The next section describes the testing method used to determine the dependency of performance on execution parallelism. Section IX analyzes the impact of execution concurrency level on job completion time. Finally, the paper is concluded with a section on recommendations on cluster configuration for optimum performance of applications running on Hadoop YARN cluster.

II. PROBLEM STATEMENT

High performance computing world has moved from supercomputers to distributed computing and all modern computing frameworks depend on distributed parallel processing based on clustering of multiple machines. With the increase in the computing resources like CPU and memory on the nodes of the cluster, it is possible to partition these resources into small units and execute multiple tasks concurrently on the same node itself. In these frameworks,

data is divided into small chunks and stored on a distributed file system and computation is moved close to the data. These frameworks manage the total computing resources available in the cluster and allocate resources to the chunks of computation in the form of containers. Hadoop YARN is a distributed computing framework commonly used for processing huge volumes of data [5]. Hadoop centrally manages the computing resources available in the cluster and allocates resources to the computing chunks based on availability. The performance of a distributed application depends on the effective utilization of the computing resources and the number of the computing chunks into which the application gets divided. Given a set of computing resources and a computational job, it is a problem to determine the right number of chunks into which the job is to be divided so that the total computational resources are effectively used to deliver maximum performance. When jobs are deployed on cloud platforms, we need to procure clusters with the right quantity of resources to meet the SLAs in job completion time. It is always a challenge to select IaaS platforms with adequate resources to meet SLAs.

III. MAPREDUCE FOR JOB PARALLELIZATION

To evaluate the dependency of concurrency level on performance, we need a mechanism to split a job into tasks of equal size. We have used MapReduce for this purpose, since it provides easy method to control the granularity by controlling the number of map tasks. MapReduce is a technique introduced by Google in 2004 to distribute the processing of very large data files across a large cluster of machines [1]. High performance is achieved by breaking the processing into small units of work that can be run in parallel across hundreds of nodes in the cluster. MapReduce is a software framework using a divide-and-conquer algorithm for parallel processing. Hadoop Distributed File System (HDFS) is an integral component of Hadoop that stores and replicates large files across multiple nodes of the cluster.

In MapReduce computational model, jobs consists of a set of automatically parallelized tasks called Map and Reduce tasks. Multiple instances of these tasks are distributed and scheduled on containers created on the nodes of the cluster. Hadoop splits the input data in smaller chunks and one Map task will be assigned the responsibility to process each chunk. The extent of parallelism in execution is controlled by the number of containers configured for the Map and Reduce tasks. The number of containers running on a single node can have a significant impact on performance due to their aggregate effects on CPU and memory utilization. To achieve maximum utilization of computing resources available on servers for guaranteed job completion times, it is crucial to select ideal number of concurrent map/reduce tasks. Too less number of map/reduce tasks increases job completion time due to under utilization of resources. Launching a large number of concurrent Map and Reduce tasks results in I/O contention due to simultaneous disk I/O. Thus, the number of concurrent Map and Reduce tasks must be chosen such that the resources are maximally utilized, resulting in optimum performance.

If the total number of tasks belonging to a MapReduce job is greater than the total number of concurrent containers available for processing the job, the task assignment will take

place in multiple rounds, which are called waves. Ideally, the total number of tasks must be an exact multiple of the number of available containers. This is to ensure full utilization of the computing capacity during all waves of execution. The goal of YARN resource manager is to find an optimal assignment policy for the tasks belonging to individual jobs.

IV. TEST ENVIRONMENT

The test environment consists of a Hadoop cluster with 4 nodes in which 3 nodes are configured as worker nodes / data nodes of HDFS. Workloads from Intel HiBench benchmarking tool is used for evaluating the performance at different levels of concurrency.

A. System Configuration

The hardware details of the nodes of the cluster are:

Processor:	Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz
Cache size:	6144 KB
Number of physical cores:	4
Memory available:	16GB
Physical storage:	512GB with 7200RPM
Network Card	Full duplex 1000 Mbps

Each node will have 16 logical cores. Hence the total resource available on the 3 worker nodes will be 48 cores and 48GB memory. The total storage space available on the distributed file system is around 1.5 TB.

Each of the nodes runs on Oracle Linux release 6.7. The cluster is configured using Hadoop version 2.8.0. One node is master node dedicated exclusively for the resource manager and the Namenode of HDFS.

B. Intel HiBench Benchmarking tool

The job performance at different levels of concurrency is determined using one of the benchmarking workloads provided in Intel HiBench Suite 6. This tool allows us to create input data file of desired size using the RandomTextWriter, and run standard applications to process the data. The job performance was evaluated using the Wordcount workload available in the Micro-bench category of HiBench workloads. Wordcount is CPU bound job with moderate disk I/O which calculates the number of occurrence of each word in the input text file. It is one of the basic MapReduce operations which tests the split, reduce and map operations of a normal MapReduce job. The input text file is stored on HDFS with a block size of 256 MB. The output file also gets created on HDFS.

C. Hadoop YARN Architecture

Distributed scheduling is a decision making process about assigning large number of tasks to a relatively less number of worker nodes available across the cluster. YARN introduced in Hadoop 2.0 is responsible for cluster resource management and scheduling.[5] Hadoop YARN cluster allows us to run various applications that need to use parallel processing for

performance. MapReduce application on YARN is just one choice. YARN is a dynamic scheduler which uses static configuration information to make decisions on the fly. YARN achieves load sharing by allocating tasks to lightly loaded nodes at the beginning and once a task is allocated to a node, it does not do dynamic load balancing by relocating running tasks to other nodes of the cluster. YARN is a centralized dynamic scheduler in which the responsibility of scheduling physically resides on a single node, the master node. It can be considered as a cooperative scheduling algorithm in which the distributed entities like Node Manager (NM) and Application Master(AM) cooperate with each other to make scheduling decisions.[3]

The YARN Resource Manager (RM) arbitrates system resources in units of containers. Containers are customizable collection of resources like CPU and memory. When a job is accepted by YARN, it creates *Container 0* on one of the nodes and launches the AM on it. There has to be one AM per application and they run in *Container 0* also known as AM container. AM is responsible for launching subsequent containers as required by the job. Containers are basic schedulable units of execution for running tasks and they are allocated by the RM upon request from the AM. The application tasks run on one or more containers either concurrently or sequentially depending on the number of containers that can be run concurrently and the total number of tasks.

The Node Manager runs on each node of the cluster; creates execution containers and monitors them based on the requests from the RM. It exchanges heartbeats with RM. AM coordinates and manages jobs; negotiates with RM to schedule tasks; the tasks are started by Node Managers. The AM containers will be running on one of the slave nodes and they will be destroyed after the job is completed. The container management scheme involved in running a job on YARN cluster is shown in Fig.1. The interaction with data blocks in HDFS is not shown in this figure.

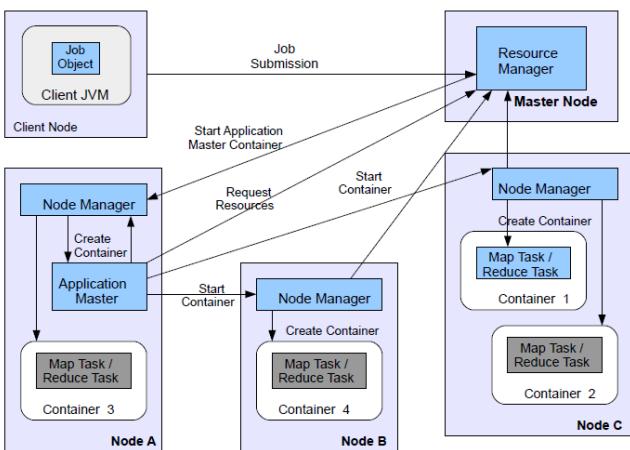


Fig. 1. Container management in YARN

One or more containers can get deployed on one node of a cluster. The number of containers on a node depends on the fraction of the total resources allocated to the containers. When

a job is run, for each task, containers get created and they get destroyed at the end of execution. There is fixed overhead in the creation and destruction of containers and the task execution time must be much higher than this overhead to take full advantage of the benefits of parallel processing.

D. Hadoop cluster configuration

The Hadoop cluster used for the experimental study consisted of 4 nodes with identical computing resources. One node is designated as the master node and the remaining 3 nodes are used as the worker nodes. The resource manager runs on the master node and the chunks of jobs run on the worker nodes. YARN scheduler makes use of containers to run the chunks of jobs known as tasks. Containers consist of partitions of computing resources with dedicated allocation of CPU and memory. The container size determines the maximum number of concurrent tasks that can run each node of the cluster.[4] The resource allocation of containers in the YARN cluster and MapReduce tasks are configured using the configuration parameters given in section VI - Configuration for resource partitioning.

V. CONTAINER SIZING

The configuration parameters for container resource allocation are defined on each node. This provides an option to deploy containers having different capacities on the nodes of the cluster. For running MapReduce jobs, it is required to have identical computing capabilities for the containers. This is to ensure that all the Map and Reduce tasks belonging to a job get completed at the same time. Container resource allocation on a cluster with nodes having identical hardware is fairly easy since we need to consider only the allocation of memory and CPU cores. If the nodes have heterogeneous hardware capabilities and CPU clock speeds, then it becomes very difficult to define containers with equal processing power.

VI. CONFIGURATION FOR RESOURCE PARTITIONING

YARN configuration allows us to partition the system into containers having dedicated resources allocated to them. At present, the resources considered are memory and CPU. The need for partitioning CPU as a resource existed for a long time. Hadoop YARN introduced a new concept called “vcores”, short for virtual cores. The decision of how much it should be set is driven by the type of workloads running in the cluster and the type of hardware available. The general recommendation is to set it to the number of physical cores on the node for CPU intensive tasks.

But, for I/O bound tasks generally seen in Hadoop environments, the vcores can be set to a value 4 times that of the physical cores. This helps in achieving higher levels of parallelism and improved CPU utilization. Support for CPU as a resource has been implemented in *CapacityScheduler* by the introduction of *DominantResourceCalculator*.

The xml file hadoop-2.8.0/etc/hadoop/yarn-site.xml residing on each node of the cluster is used for specifying the YARN configuration parameters. The configuration attributes used to define the vcores and memory allocation for the containers are given in TABLE.I. The resource allocation for

containers is controlled by specifying the minimum and maximum allocation of memory and vcores.

TABLE I. YARN CONFIGURATION PARAMETERS

Attribute	Typical value
yarn.nodemanager.resource.cpu-vcores	44
yarn.nodemanager.resource.memory-mb	12288 MB
yarn.scheduler.minimum-allocation-mb	1024 MB
yarn.scheduler.maximum-allocation-mb	12288 MB
yarn.scheduler.increment-allocation-mb	256 MB
yarn.scheduler.minimum-allocation-vcores	1
yarn.scheduler.maximum-allocation-vcores	44
yarn.scheduler.increment-allocation-vcores	1
yarn.nodemanager.vmem-pmem-ratio	2.1

YARN can be configured to trigger failure when container creation fails due to the non availability of configured quantity of resources. Virtual Memory allocation is limited by Node Manager. A Task is killed if it exceeds its allowed Virtual Memory limit. Virtual memory allocation is configured in multiples of physical memory. By default, it is 2.1 of container's physical memory. For example, if a container is configured with 1 GB of physical memory then it will not be able to exceed 2.1 GB of Virtual Memory. This can be adjusted via *yarn.nodemanager.vmem-pmem-ratio* property in yarn configuration file. YARN can be configured to allow multiple container assignments in one heartbeat for better performance

TABLE II. MAPREDUCE CONFIGURATION PARAMETERS

Attribute	Task	Typical Value
yarn.app.mapreduce.am.resource.cpu-vcores	App Master	1
yarn.app.mapreduce.am.resource.mb	App Master	1536 MB
mapreduce.map.cpu.vcores	Map	4
mapreduce.map.memory.mb	Map	1024 MB
mapreduce.map.java.opts.max.heap	Java VM	1024 MB
mapreduce.reduce.cpu.vcores	Reduce	8
mapreduce.reduce.memory.mb	Reduce	2048 MB
mapreduce.reduce.java.opts	Java VM	2048 MB

YARN provides an xml configuration file in hadoop-2.8.0/etc/hadoop/mapred-site.xml to specify the resource allocation for containers for MapReduce jobs. Using this configuration file, we can do resource allocation specifically for the application master, map and reduce tasks of the MapReduce jobs. The MapReduce specific configuration parameters are shown in TABLE.II

VII. RESOURCE PARTITION AND CONCURRENCY IN EXECUTION

The application master containers can get launched on any one of the nodes of the cluster. Hence we need to reserve resources for the application master container on all nodes of the cluster. In Hadoop distributed computing framework, when a MapReduce job is run, the total number of map tasks of a job depends on the size of the input file and the block size. For each block of file, one map task will be assigned. Hadoop Uber mode is disabled to create a run-time environment similar to that of a huge workload in which the containers get launched on separate JVMs.

In the test environment, the input file is of size 9 GB with a block size of 256 MB. So, there will be 36 map tasks in total. In a Hadoop cluster , the number of concurrent maps depends on the resource partitioning in the form of containers. Leaving memory and CPU resources required for the operating system, the system has 12 GB of memory and 44 vcores that can be allocated to the Hadoop YARN cluster. When the resources allocated to the containers are less, it is possible to run more number of concurrent maps and the utilization of the cluster capacity also increases. TABLE.III shows the relation between parallelism, number of map tasks, number of waves and container resource allocation for the map tasks. This table is based on the total resources available on the 3 worker nodes of the cluster.

TABLE III. CONFIGURATION FOR CONCURRENT MAPS

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%

The number of map tasks in the state of concurrent execution will be controlled by the MapReduce configuration file by setting suitable values for the attributes *mapreduce.map.cpu.vcores* and *mapreduce.map.memory.mb*. The reduce tasks are configured to run after completion of all

the map tasks so that the containers get reused and there will not be any resource contention between the map and reduce tasks.

VIII. TESTING EXECUTION PARALLELISM

The MapReduce configuration to define the container size is done in 6 different ways as shown in Table.III. A text file of size 9 GB is created on HDFS. The WordCount workload from the HiBench suite is run with each of these configurations. The total execution time of the job in each case is observed along with the cluster capacity utilization shown in the last column of TABLE.III. The number of Reduce tasks is configured as 8 and their execution time is marginal when compared with that of the map tasks. The major contribution of the execution time is attributed to the map phase of the job.

IX. TASK CONCURRENCY AND JOB COMPLETION TIME

The job completion time at various levels of concurrency is shown in Fig.2. Node level concurrency is seen at concurrency level of 3, meaning one map task each on the nodes and in this case the job completion time is maximum. At the highest level of concurrency of 36, there will be 12 map tasks each running on each node.

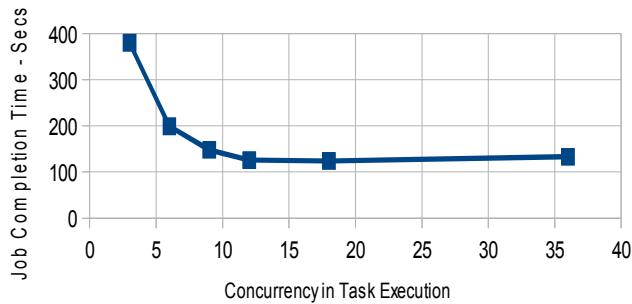


Fig. 2. Concurrency level Vs Performance

A model representing the dependency of execution time on parallelism is created by nonlinear regression analysis using the SPSS package. The model is represented by a nonlinear function in the form of $T=F(p)$ where T is the execution time and p is the level of execution parallelism. The model generated from the data represented in Fig.2 is given in (1).

$$T = A + B \exp(-C p) + D p^2 \quad (1)$$

Where:

$$A = 110.0, B = 720.0, C = 0.33, D = 0.02$$

In (1) the parameter A is dependent on the hardware platform and it decides the base performance level. The parameters B , C and D are independent of the hardware platform. These parameters depend on the YARN scheduler and the task granularity in execution. It can be seen that initially there is an exponential decay in the execution time and settles down at the base performance level. The execution time increases nonlinearly beyond the base level with the

increasing level of parallelism. Hence there is no advantage in going for parallelism beyond this base threshold level. The global and local schedulers influence the performance up to the threshold level of parallelism. The overheads in local scheduler and Disk I/O congestion are responsible for the increase in the execution time beyond the threshold level of parallelism.

The threshold level of performance is dependent on the system resources. In the test machines with 16 logical cores and 16 GB memory, containers configured with 4 logical cores and 4 GB memory gives minimum execution time. This gives a node level concurrency of 4. On servers with single hard disk, it is recommended to define containers with $\frac{1}{4}$ of the available logical cores and $\frac{1}{4}$ of the available physical memory on each node of the cluster. This ensures that there will be 4 task containers running concurrently on each node of the cluster.

X. RECOMMENDATIONS

If small number of large containers is defined, it results in lower utilization of computing resources. This results in increased execution time. If large number of small containers is defined, it will result in better utilization of the cluster resources. But, the execution time may not reduce due to the resource contention on memory and I/O subsystem. The reason for this is thrashing resulting in large variation in completion time of the tasks. When a MapReduce job is run, it is very common to re-use the containers for running map and reduce tasks. Hence until all the map tasks are completed, the reduce tasks cannot be started and the late completion of a single map task can delay the start of the reduce tasks. This study shows that the ideal number of concurrent containers that can be run on a node depends on the number of physical cores and the system memory. It is recommended to define containers with $\frac{1}{4}$ of the available logical cores and $\frac{1}{4}$ of the available physical memory on each node of the cluster. This ensures that there will be 4 task containers running concurrently on each node of the cluster. In this experimental study, this will result in 12 cluster wide concurrent containers. And this is the level of concurrency at which the minimum execution time is seen.

XI. CONCLUSION

YARN introduced in Hadoop 2.0 provides a flexible resource manager. This resource manager helps in allocating dedicated portion of system resources for the Hadoop batch processing applications leaving rest of the resources for development and for running general purpose applications. Now it is possible to run wide variety of applications developed in languages like Scala, Python and R on a Hadoop cluster making it possible to use it as a powerful distributed computing framework. Performance of distributed applications depends on effective utilization of the computing resources available in the cluster by judicious selection of concurrency level of execution of the component tasks of the jobs. The performance on single disk systems depends on overheads in local schedulers and Disk I/O congestion. The CPU and memory resources on cloud servers are to be selected based on

the constraints in execution parallelism that can be supported on the nodes of the cluster.

REFERENCES

- [1] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters", Symposium on Operating System Design and Implementation (OSDI), San Francisco, California, USA, Dec 2004
- [2] Bressoud,Thomas C and Tang Qiuyi, "Analysis, modeling, and simulation of Hadoop YARN MapReduce", IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016
- [3] Vellaipandian, Solaimurugan and Vignesh Raja P, "Performance evaluation of distributed framework over YARN cluster manager", IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2016
- [4] Genkin Mikhail, Dehne Franket, et al, "Automatic, on-line tuning of YARN container memory and CPU parameters" , IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016.
- [5] Diaz Adam, "How YARN changed job scheduling in hadoop" , Linux Journal: Volume 2014 Issue 240, April 2014.

Big Data Systems

Webinar 1 -MongoDB

Agenda

Introduction

Architecture

Important Features

NoSQL Database

MongoDB Vs RDBMS

CRUD Operations

MongoDB- Replication & Aggregation

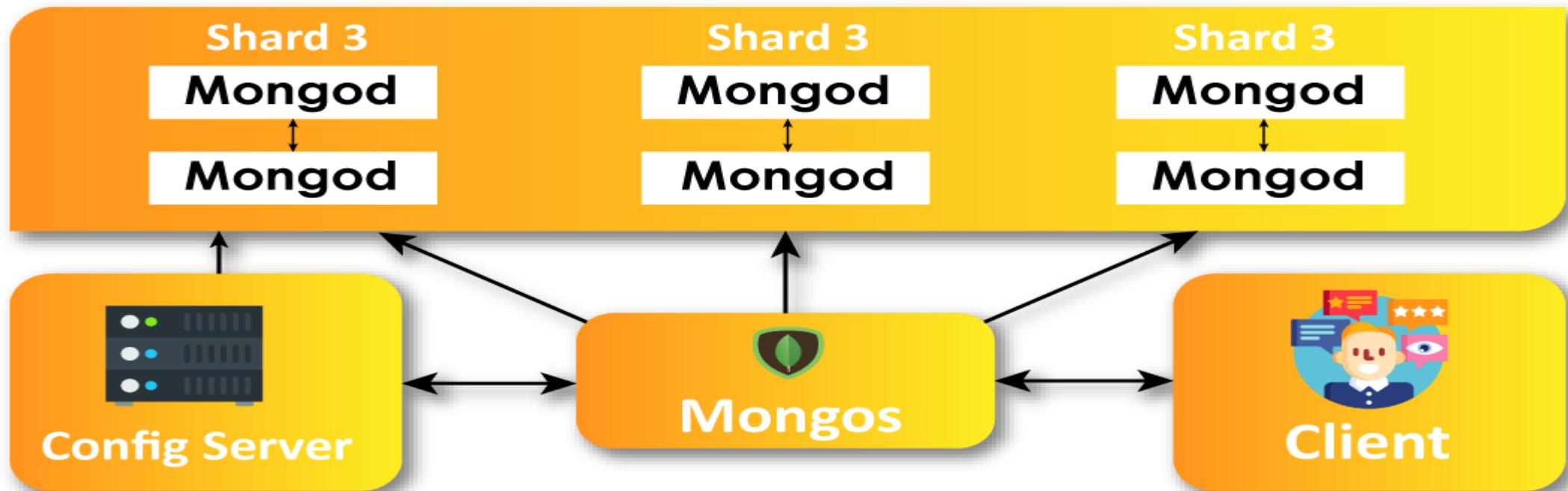
Introduction

- 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?



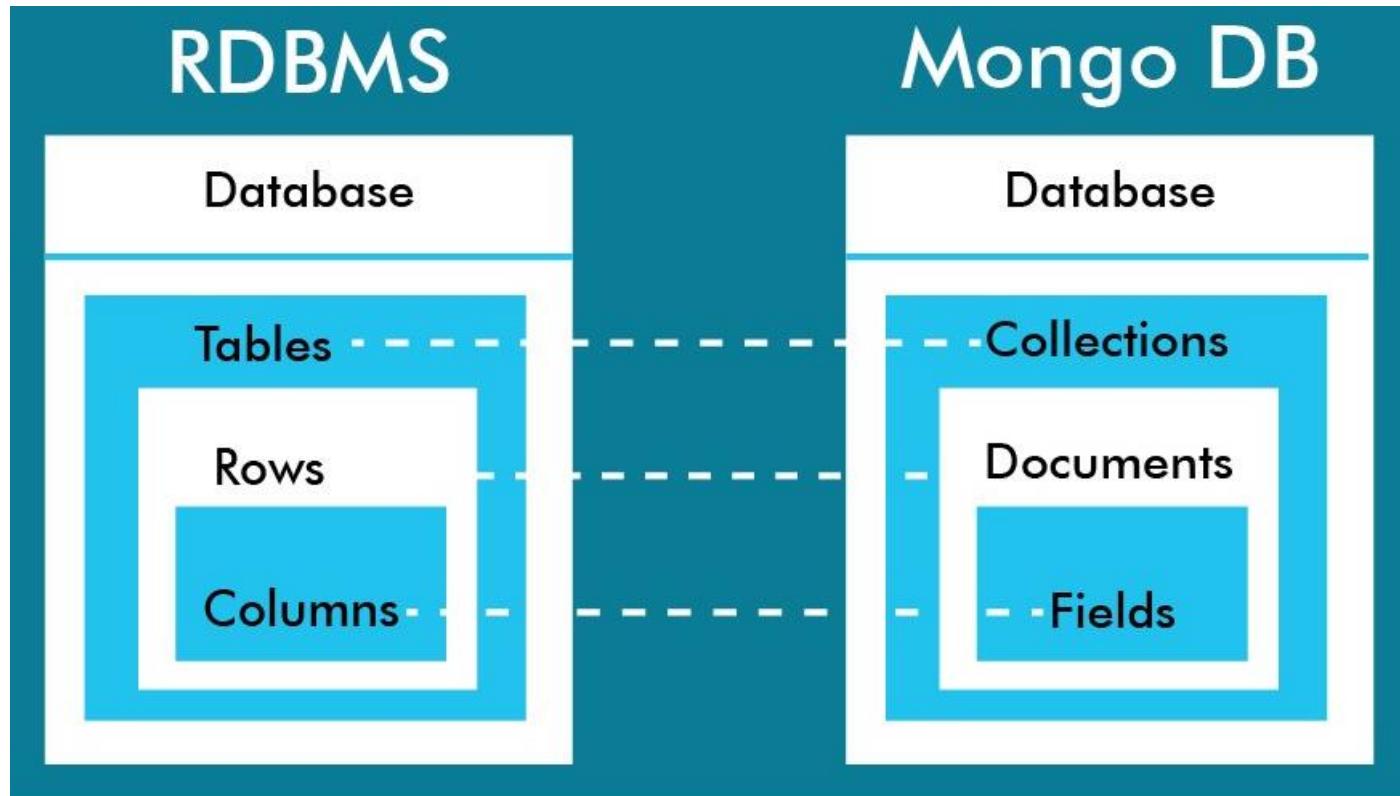
-
- **GridFS:** This feature will allow the files to divide into smaller parts and store them in different documents without complicating the stack.
 - **Schema-less Database:** MongoDB is a schema-less database programmed in C++ language.
 - **Document-oriented Storage:** It uses BSON format which is 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" }
> -
```

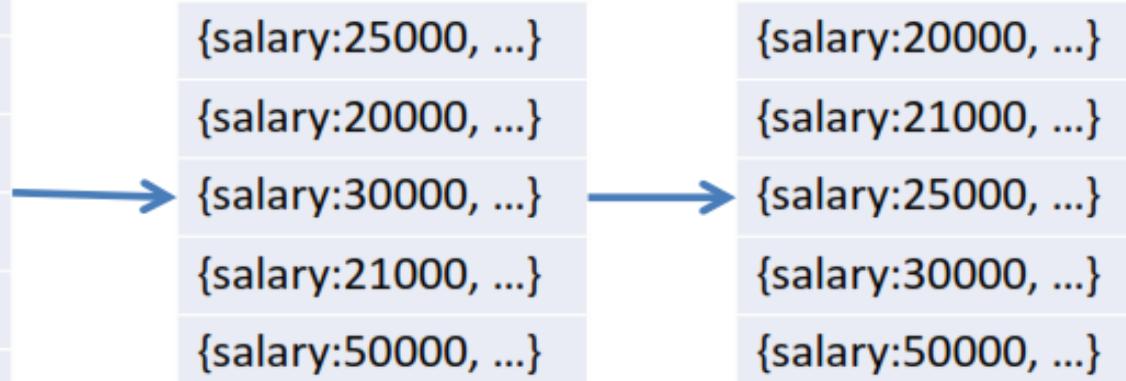
Sample MongoDB Query

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, ...}



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
capped	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.
autoIndexId	Boolean	(Optional) If true, automatically create index on _id fields Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.
max	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.

```
c:\ 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
KMovies
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
}
)
} } document
```

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.^[3]

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:

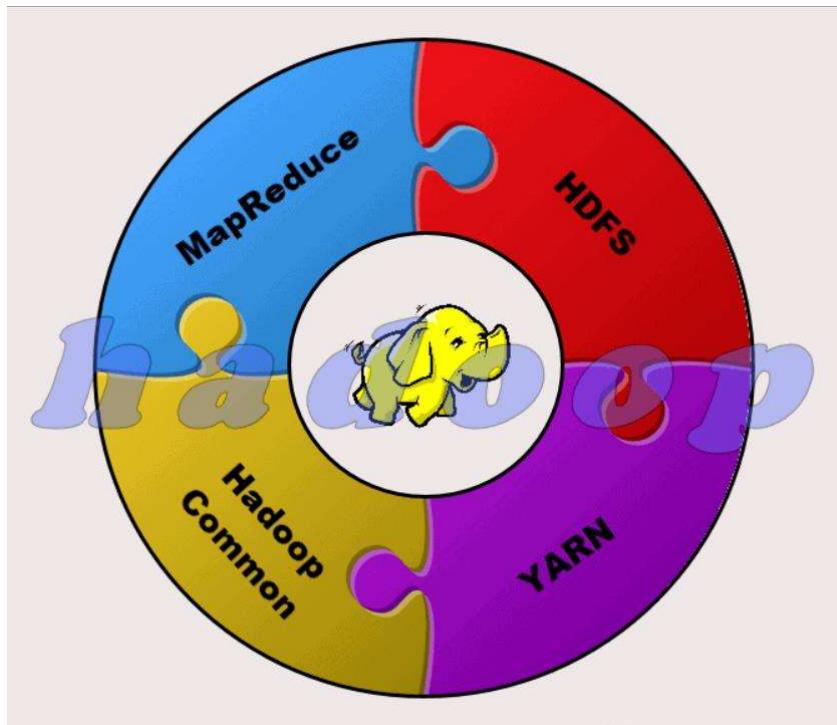
HDFS – Hadoop Distributed File System. This is the file system that manages the storage of large sets of data across a Hadoop cluster. HDFS can handle both structured and unstructured data. The storage hardware can range from any consumer-grade HDDs to enterprise drives.

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.

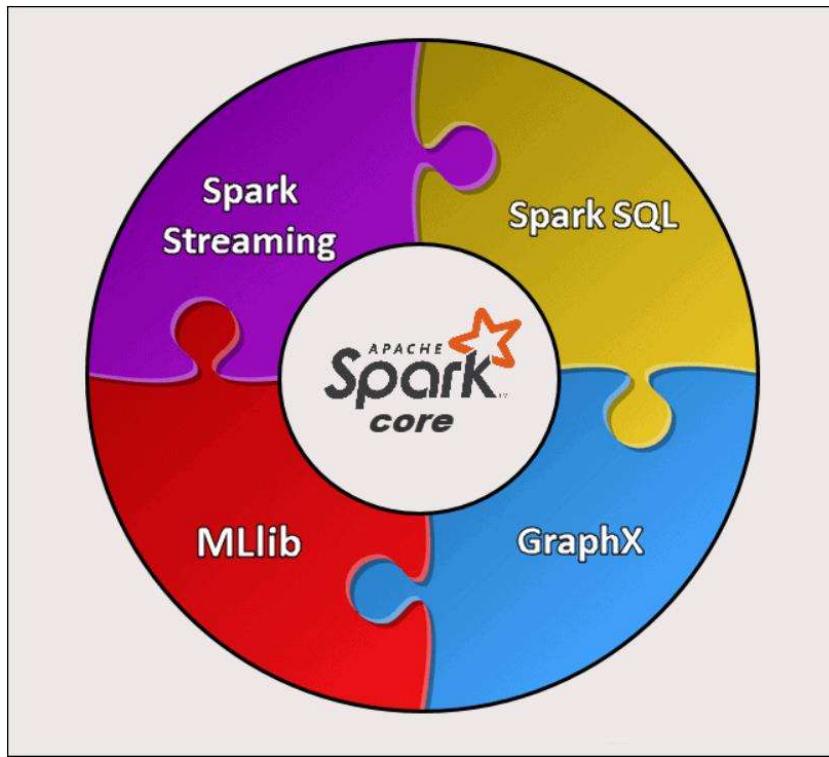
The Spark engine was created to improve the efficiency of MapReduce and keep its benefits. Even though Spark does not have its file system, it can access data on many different storage solutions. The data structure that Spark uses is called Resilient Distributed Dataset, or RDD.

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

Category for Comparison	Hadoop	Spark
Performance	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.
Cost	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.
Data Processing	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

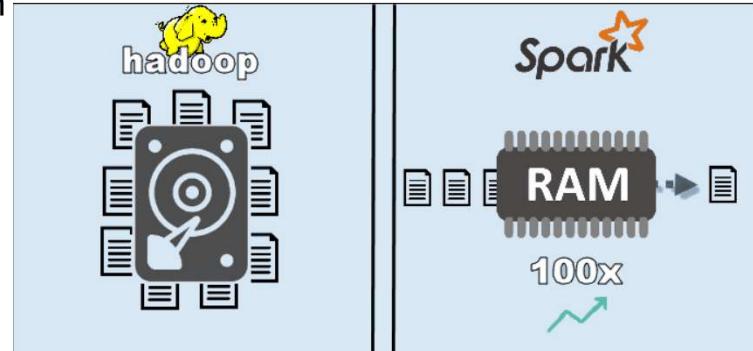
Category for Comparison	Hadoop	Spark
Fault Tolerance	A highly fault-tolerant system. Replicates the data across the nodes and uses them in case of an issue.	Tracks RDD block creation process, and then it can rebuild a dataset when a partition fails. Spark can also use a DAG to rebuild data across nodes.
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
Ease of Use and Language Support	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.
Machine Learning	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.
Scheduling and Resource Management	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.



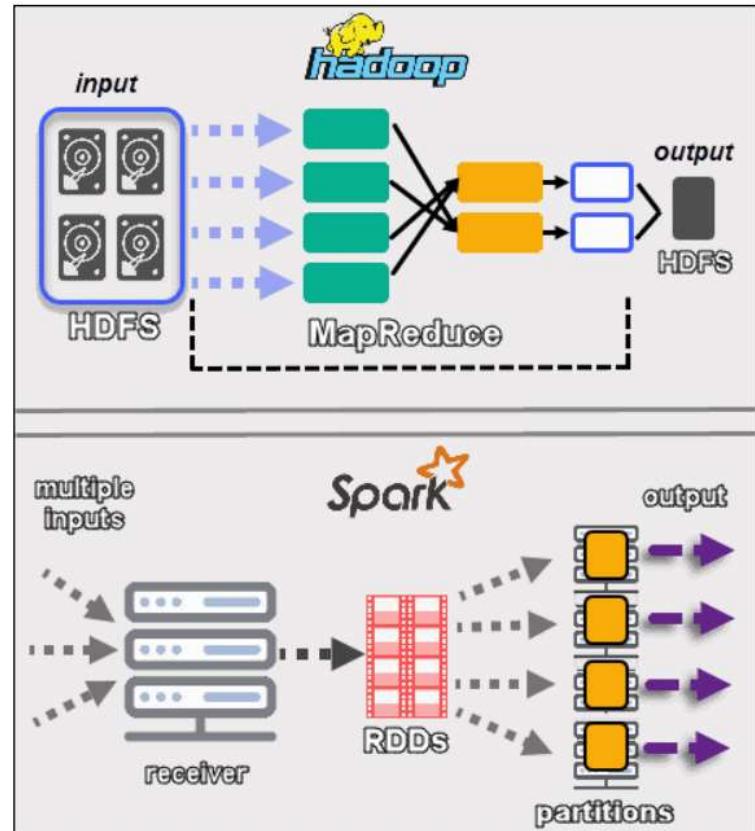
The dominance remained with sorting the data on disks. Spark was 3x faster and needed 10x fewer nodes to process 100TB of data on HDFS. This benchmark was enough to set the world record in 2014.

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.

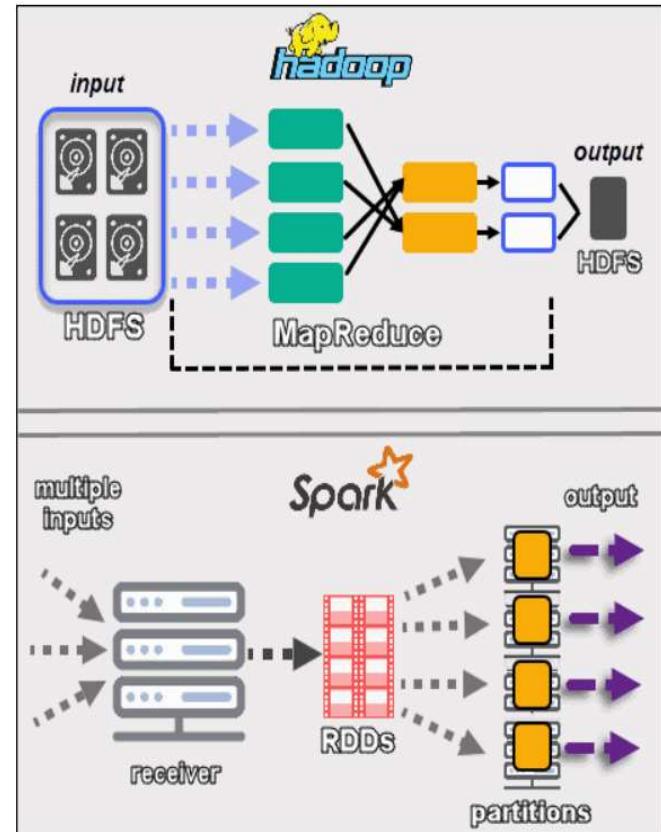
Hadoop's goal is to store data on disks and then analyze it in parallel in batches across a distributed environment. MapReduce does not require a large amount of RAM to handle vast volumes of data. Hadoop relies on everyday hardware for storage, and it is best suited for linear data processing.



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.



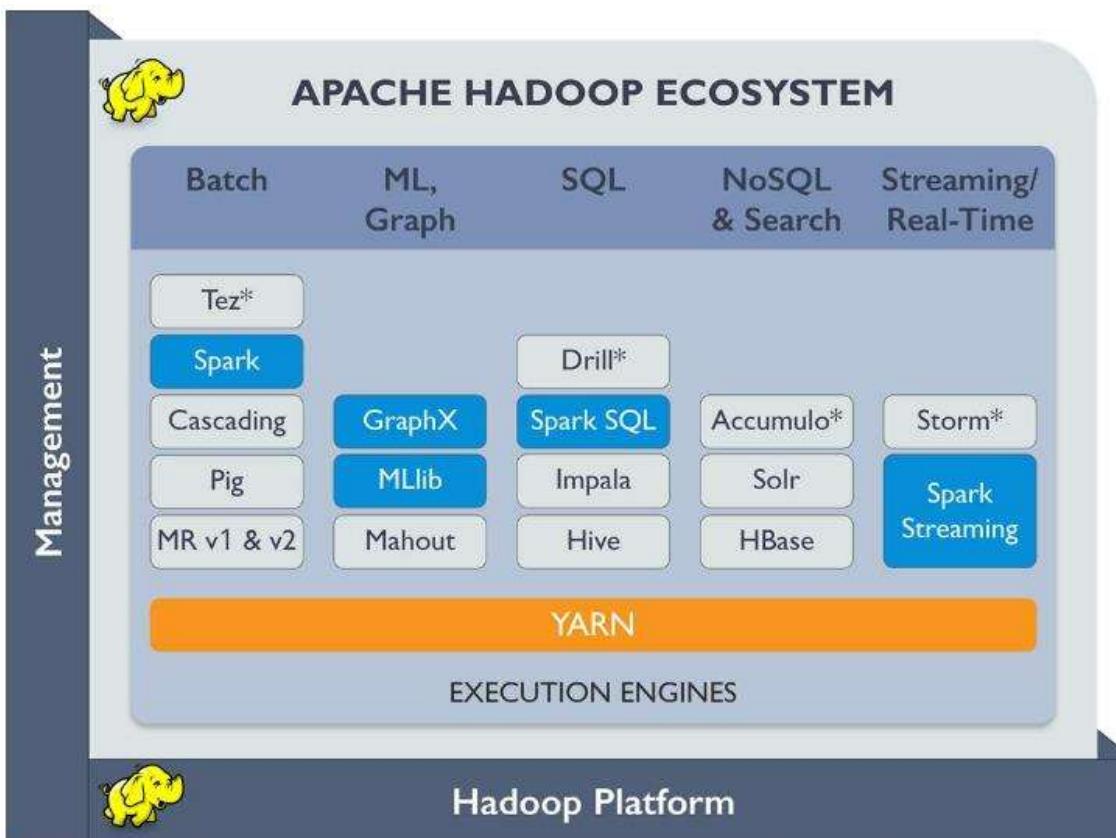
Fault Tolerance

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.

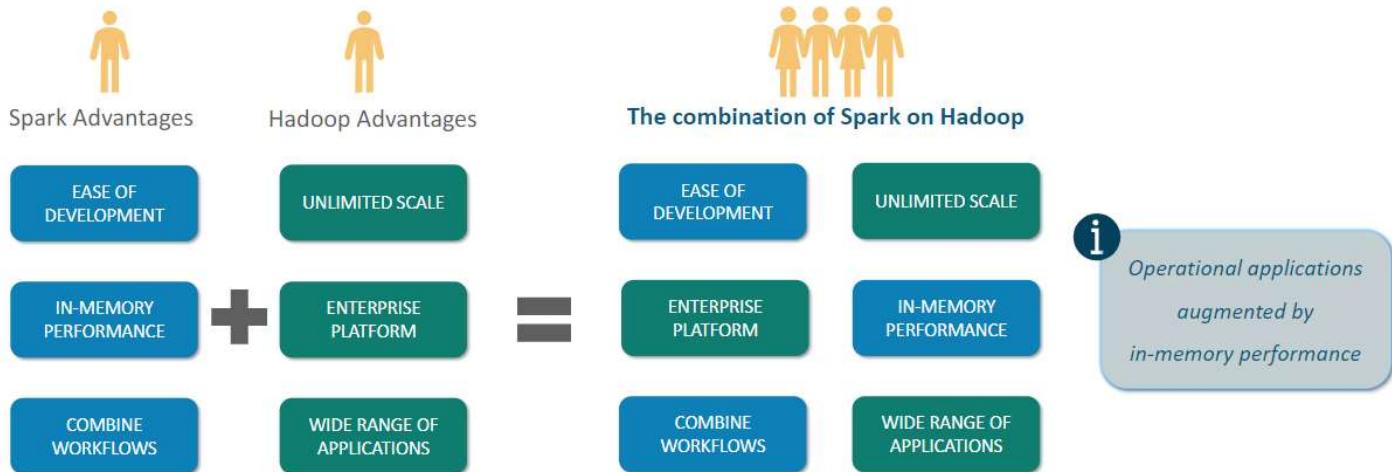
Spark uses RDD blocks to achieve fault tolerance. The system tracks how the immutable dataset is created. Then, it can restart the process when there is a problem. Spark can rebuild data in a cluster by using DAG tracking of the workflows. This data structure enables Spark to handle failures in a distributed data processing ecosystem.

Spark in Hadoop ecosystem

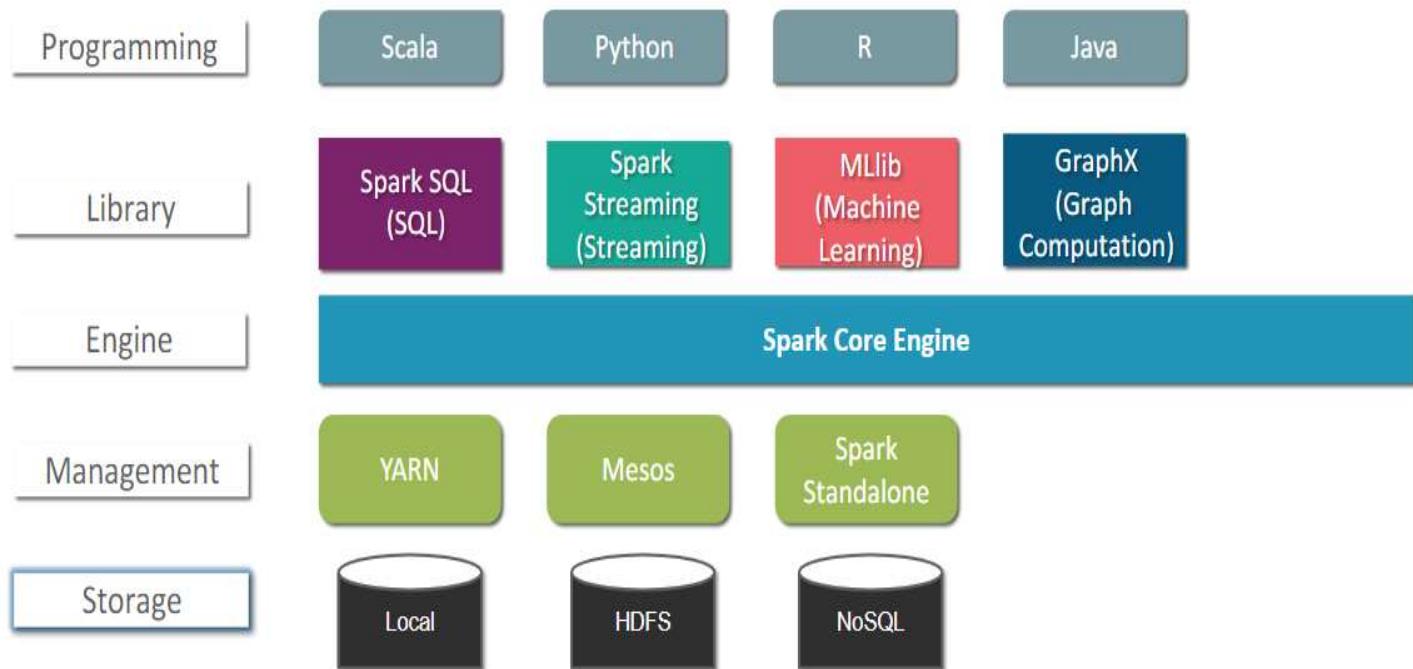


Spark and Hadoop

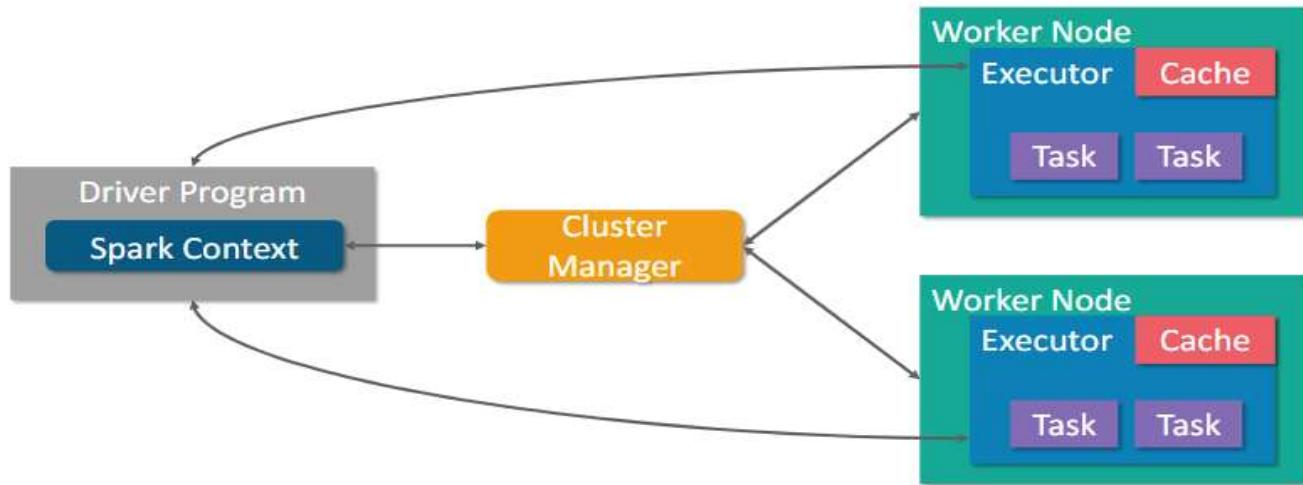
Spark And Hadoop Together



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

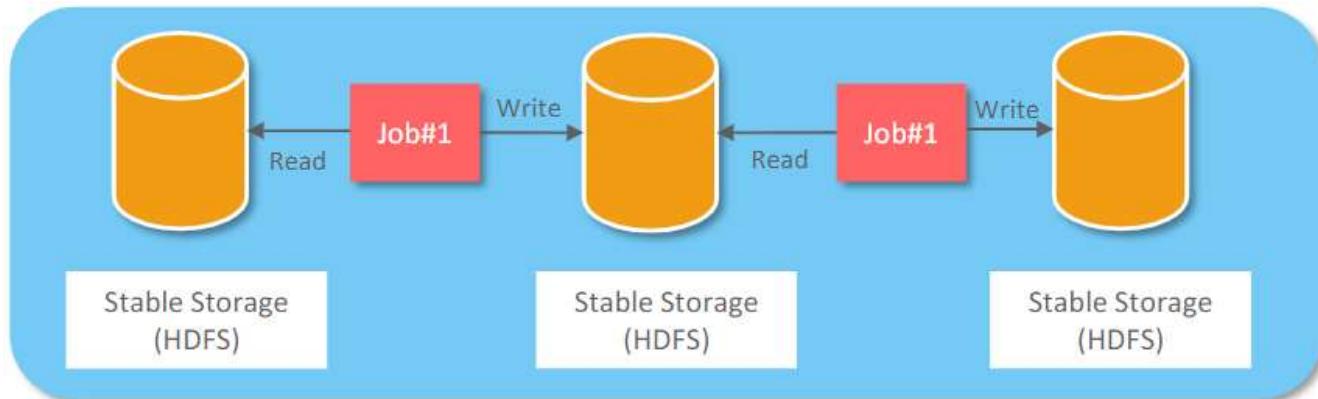
- Deploy Spark on top of Hadoop YARN

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

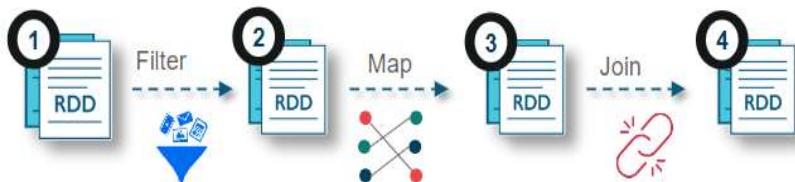
- Our Goal here is to reduce the number of I/O operations through HDFS
- This can only be achieved through "***In-Memory Data Sharing***"
- The ***In-Memory Data Sharing*** is 10-100x faster than ***Network/Disk*** sharing



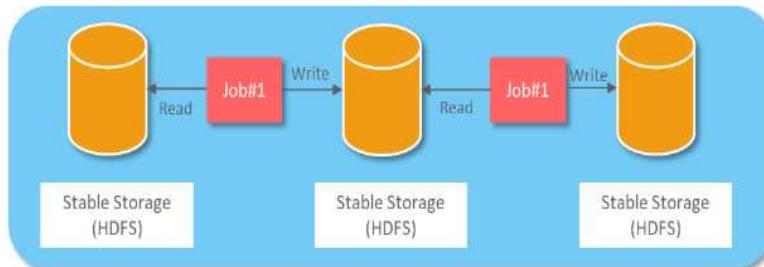
RDD – Perfect Solution

- 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

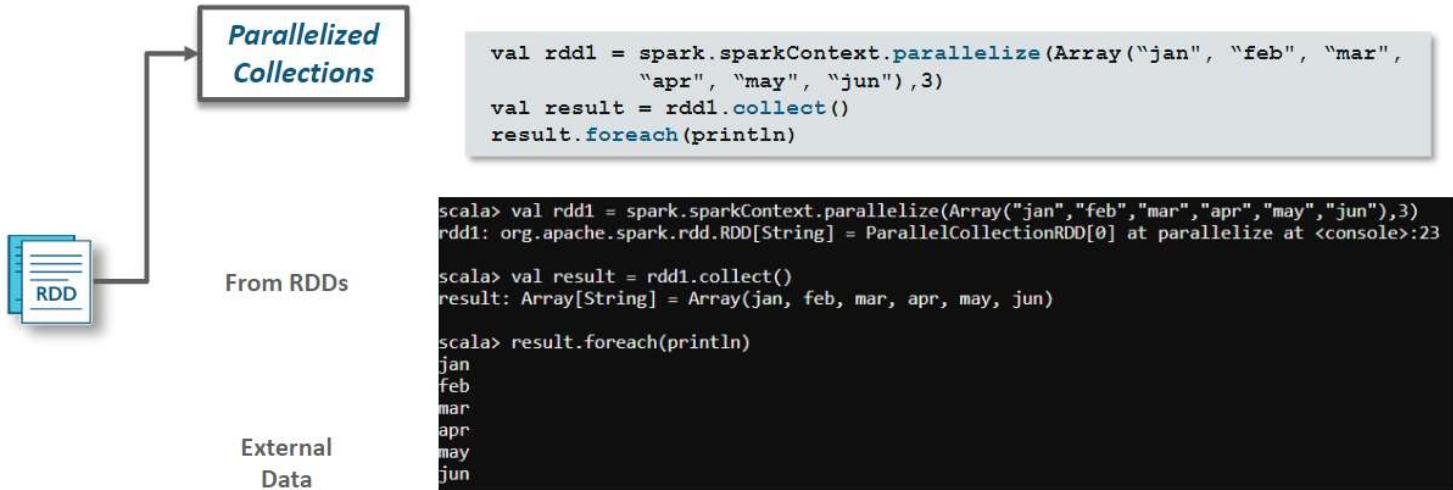
Partitioning

Partitioning is the fundamental unit of parallelism in Spark RDD

Immutability

Data can be created or retrieved anytime

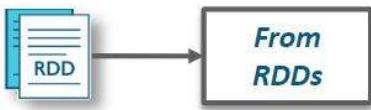
How to create RDD



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

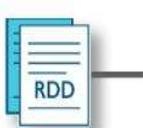
scala> val wordPair = words.map(w => (w.charAt(0), w))
wordPair: org.apache.spark.rdd.RDD[(Char, String)] = MapPartitionsRDD[11] at map at <console>:25

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
```

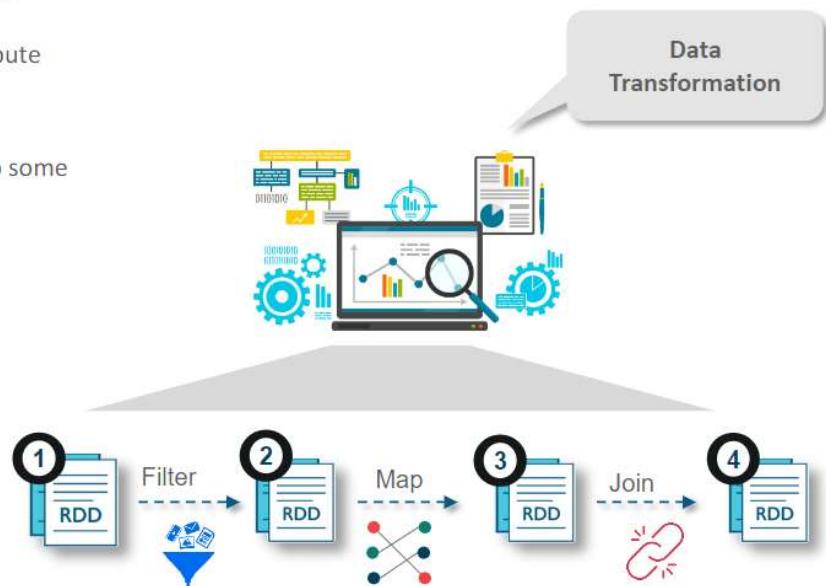
*External
Data*

```
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

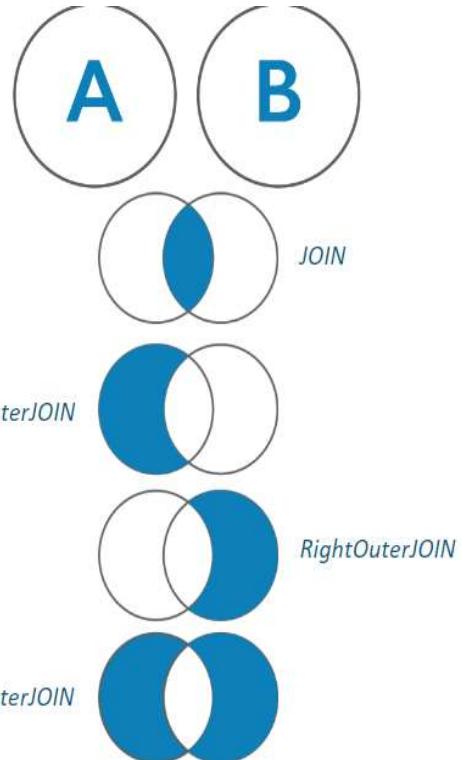
aggregate

```
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 _)
    
```

