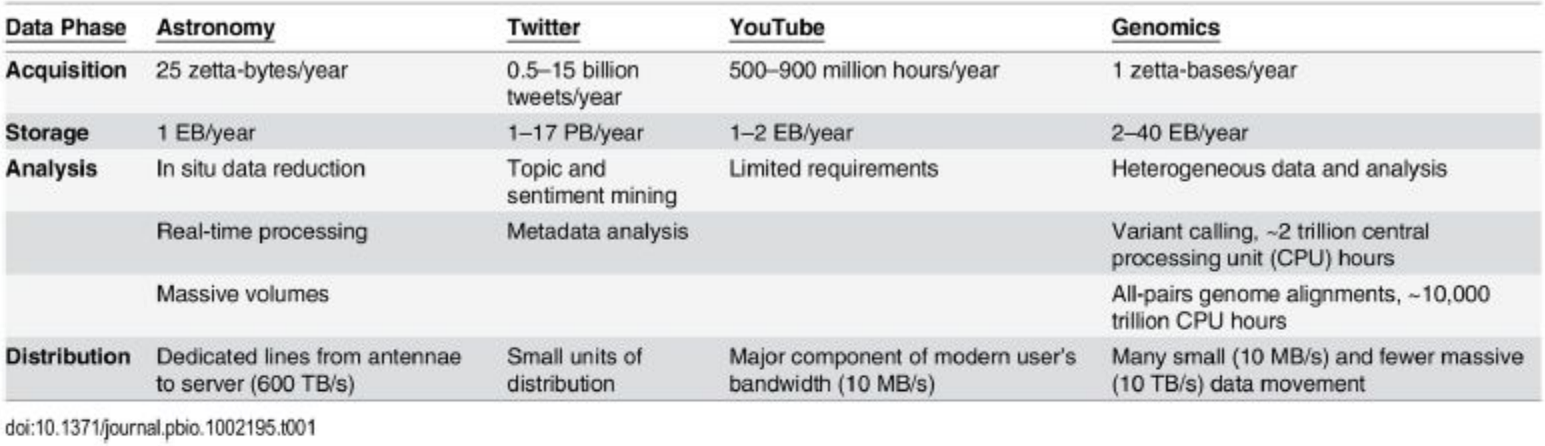
Hadoop : Introduction and Use Cases for Data Analysis

Big Data in 2025

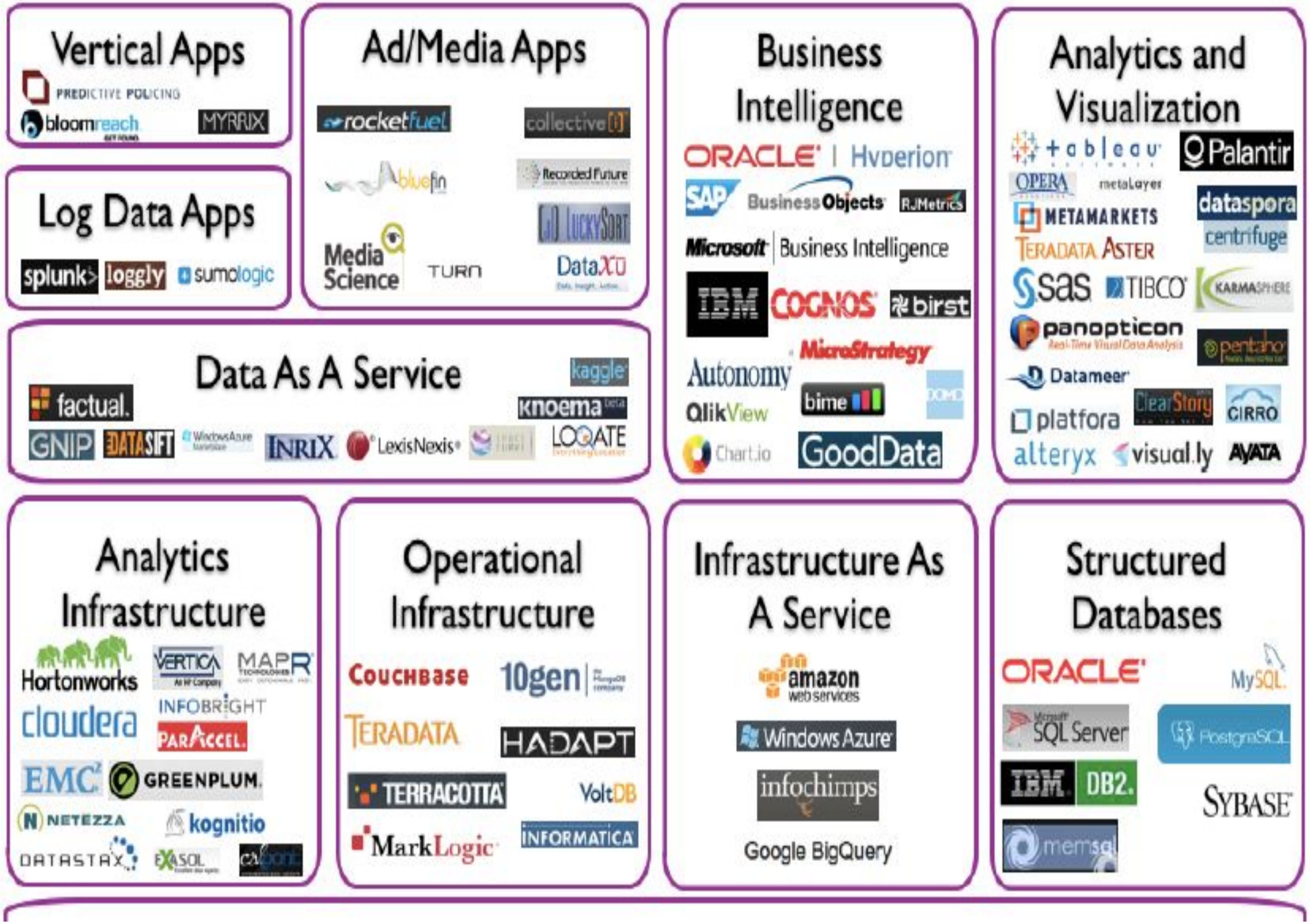


Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001

This table[1] shows the projected annual storage and computing needs in four domains (astronomy, social media, genomics)

Big Data Landscape



Vertical Apps

Log Data Apps

Ad/Media Apps

Data As A Service

Business Intelligence

Analytics and Visualization

Analytics Infrastructure

Operational Infrastructure

Infrastructure As A Service

Structured Databases

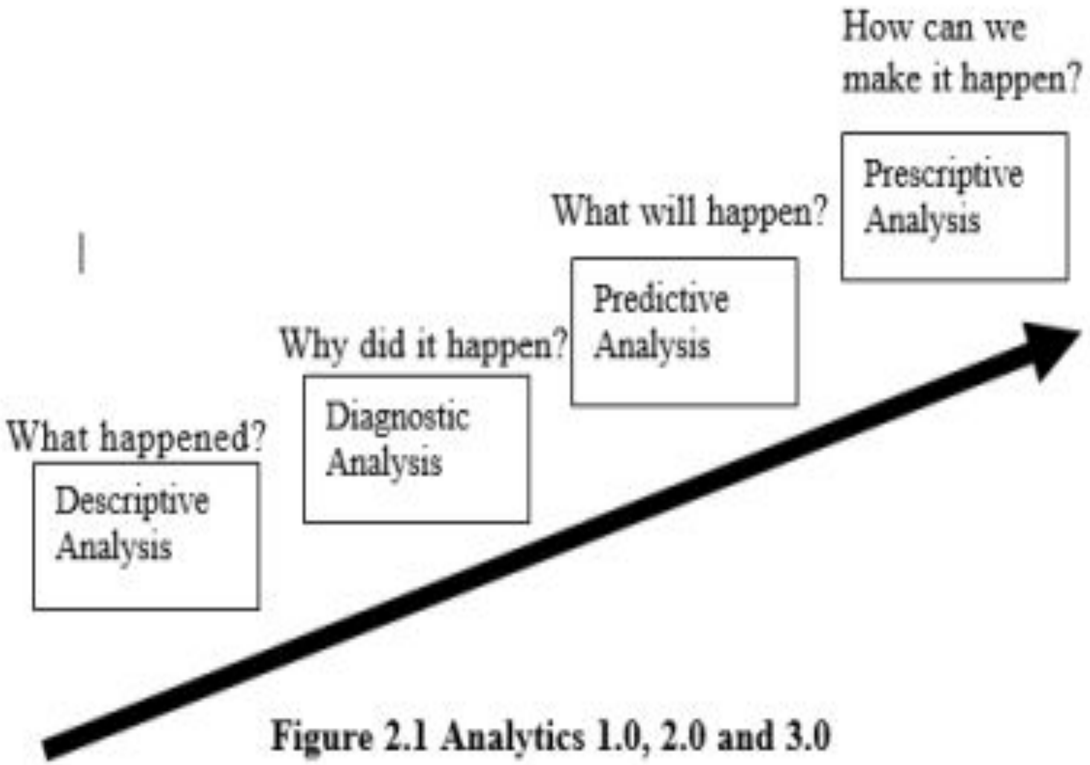


Figure 2.1 Analytics 1.0, 2.0 and 3.0

(Big Data and Analytics)

Growth of Big Datasets

- Internet/Online Data
  - Clicks
  - Searches
  - Server requests
  - Web logs
  - Cell phone logs

  - Mobile GPS locations
  - User generated content
  - Entertainment (YouTube, Netflix, Spotify, …)
- Healthcare and Scientific Computations
  - Genomics, medical images,
  - healthcare data, billing data
- Graph data
  - Telecommunications network
  - Social networks (Facebook, Twitter, LinkedIn, …)
  - Computer networks

Data

- The Large Hadron Collider produces about 30 petabytes of data per year
- Facebook’s data is growing at 8 petabytes per month
- The New York stock exchange generates about 4 terabyte of data per day
- YouTube had around 80 petabytes of storage in 2012
- Internet Archive stores around 19 petabytes of data

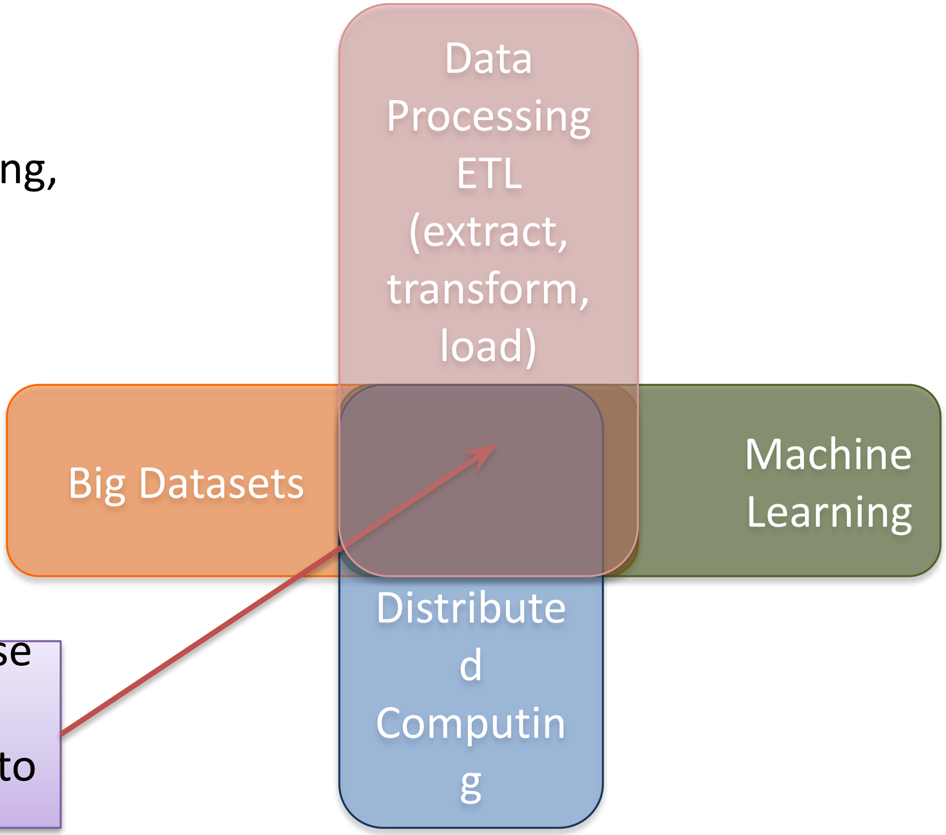
Cloud and Distributed Computing

- The second trend is pervasiveness of cloud based storage and computational resources
  - For processing of these big datasets
- Cloud characteristics
  - Provide a scalable standard environment
  - On-demand computing
  - Pay as you need
  - Dynamically scalable
  - Cheaper

Data Processing and Machine learning Methods

- Data processing (third trend)
  - Traditional ETL (extract, transform, load)
  - Data Stores (HBase, ……..)
  - Tools for processing of streaming, multimedia & batch data
- Machine Learning (fourth trend)
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering

Working at the Intersection of these four trends is very exciting and challenging and require new ways to store and process **Big Data**

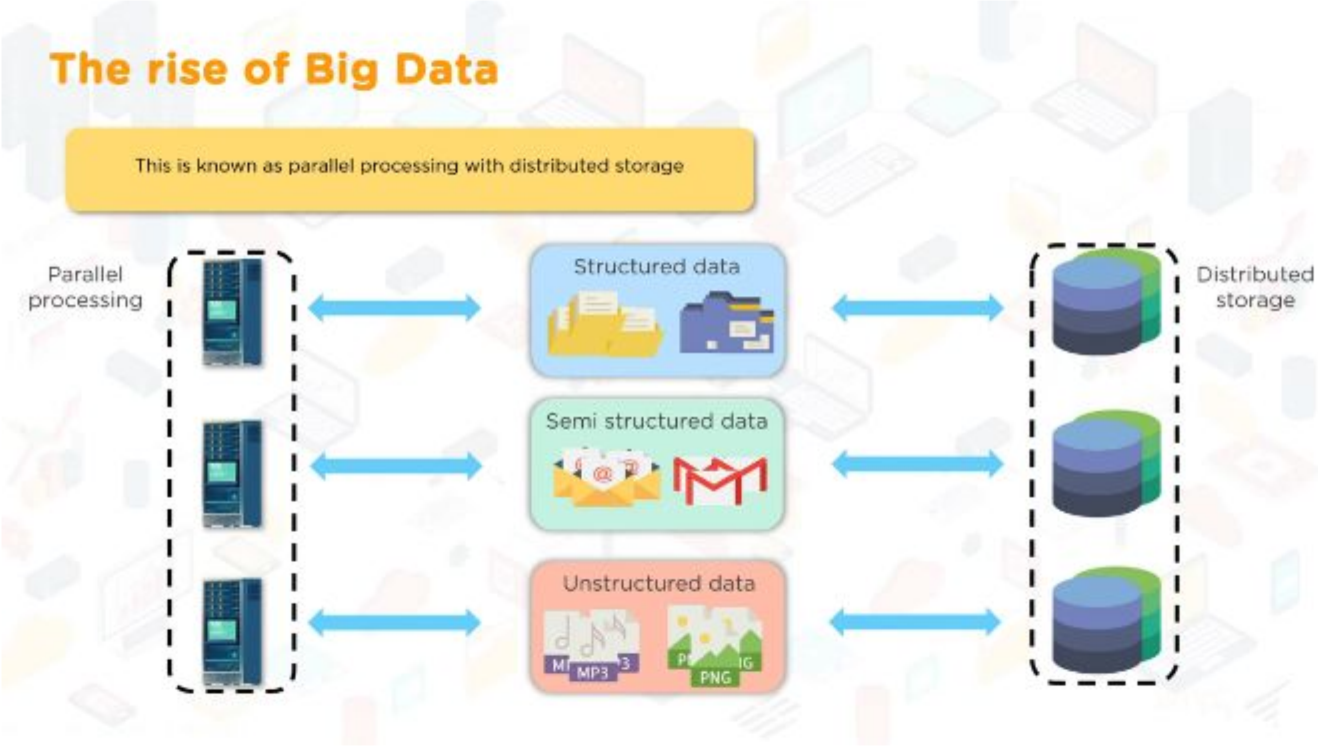


Data Processing ETL (extract, transform, load)

Big Datasets

Machine Learning

Distribute d Computin g



The rise of Big Data

This is known as parallel processing with distributed storage

Parallel processing

Structured data

Semi structured data

Unstructured data

Distributed storage

Hadoop

Hadoop is a

- Java-based,
- open-source framework
- designed for storing and processing large datasets
- in a distributed environment
- It uses a cluster of commodity hardware to handle big data
- and analytics jobs by breaking them into smaller,
- parallelizable tasks.

Hadoop Ecosystem

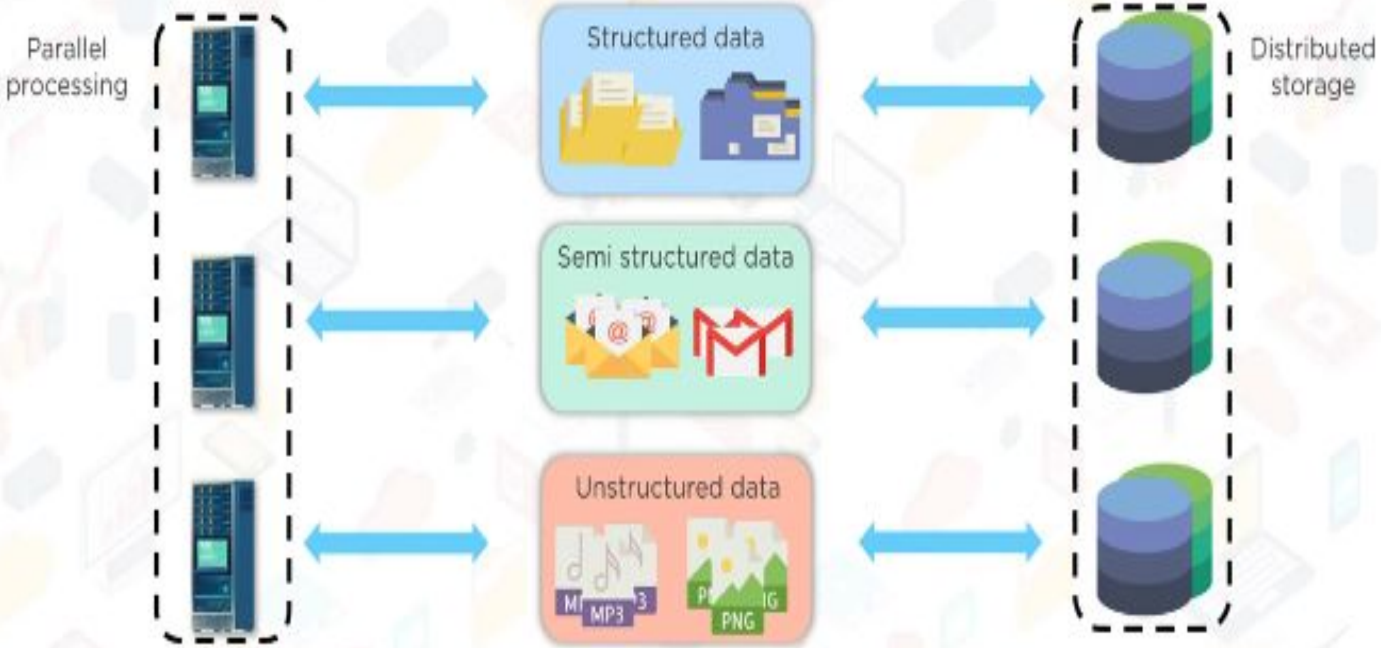
- Enable Scalability
  - on commodity hardware
- Handle Fault Tolerance
- Can Handle a Variety of Data type
  - Text, Graph, Streaming Data, Images,…
- Shared Environment
- Provides Value
  - Cost

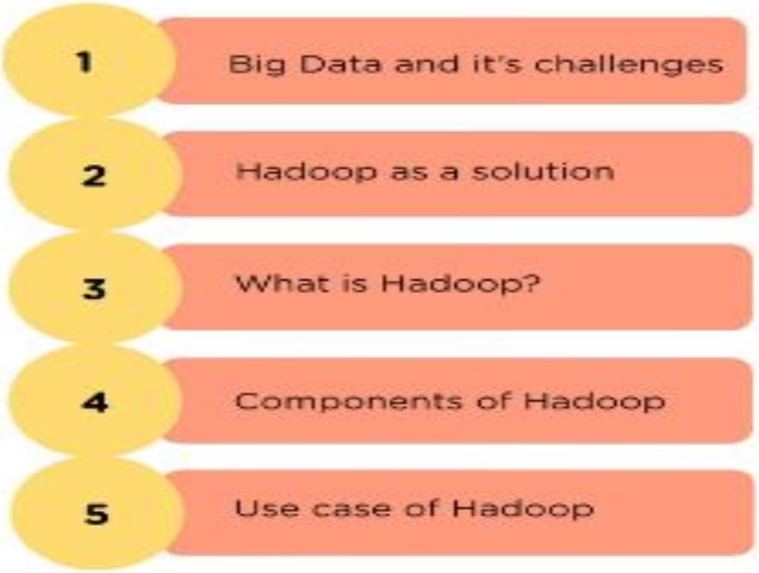
Storing file on HDFS

- To ensure that data is not lost, data can typically be replicated on:
  - local rack
  - remote rack (in case local rack fails)
  - remote node (in case local node fails)
  - randomly
- Default replication factor is 3

The rise of Big Data

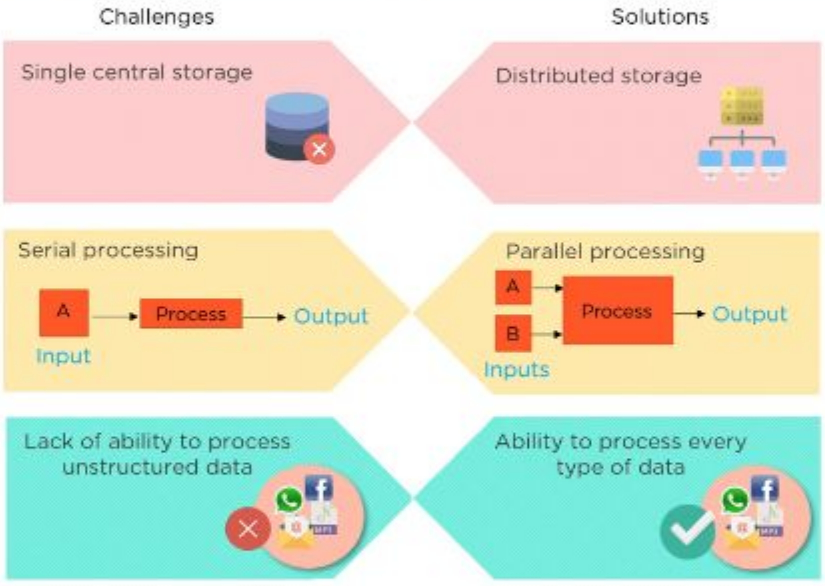
This is known as parallel processing with distributed storage





1. Big Data and it's challenges
2. Hadoop as a solution
3. What is Hadoop?
4. Components of Hadoop
5. Use case of Hadoop

Big Data challenges and solution



Challenges

Solutions

Single central storage

Distributed storage

Serial processing

A → Process → Output

Input

Parallel processing

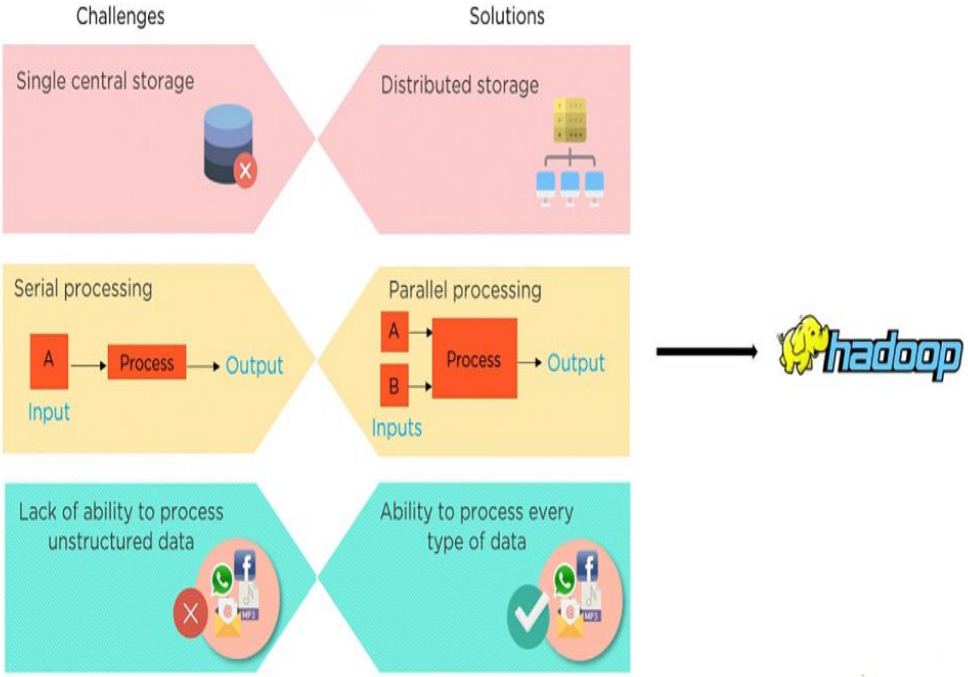
A, B → Process → Output

Inputs

Lack of ability to process unstructured data

Ability to process every type of data

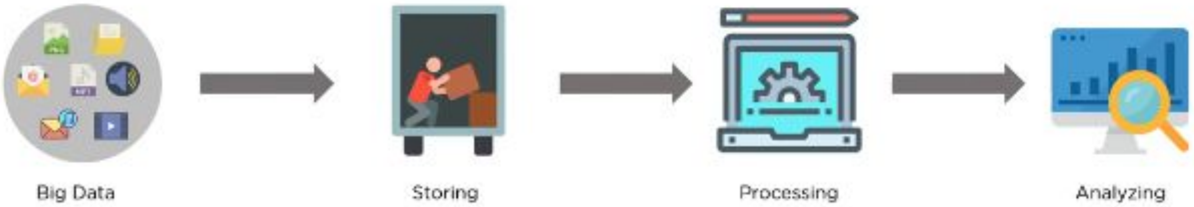
Hadoop as a solution



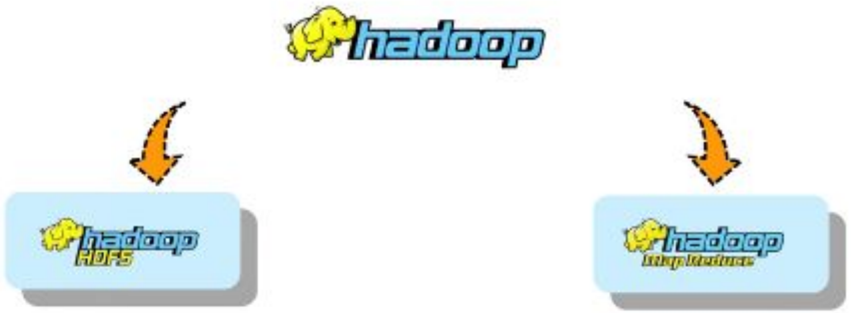
Challenges	Solutions
Single central storage	Distributed storage
Serial processing	Parallel processing
Lack of ability to process unstructured data	Ability to process every type of data

What is Hadoop?

Hadoop is a framework that manages big data storage in a distributed way and processes it parallelly



Components of Hadoop

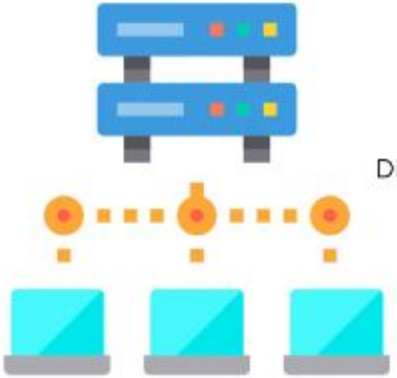


Storage unit of Hadoop

Processing unit of Hadoop

What is HDFS?

Hadoop Distributed File System (HDFS) is specially designed for storing huge datasets in commodity hardware



Distributed storage

What is HDFS?

Hadoop Distributed File System (HDFS) has two core components NameNode and DataNode

NameNode

DataNode

Terminology

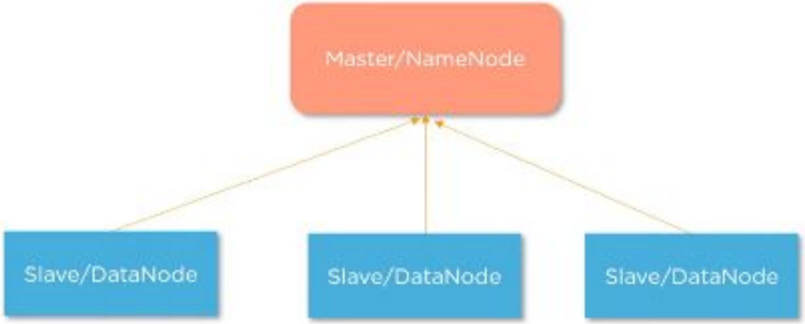
- **HDFS**: Hadoop Distributed File System
- **Datanode**: A DataNode stores data in HDFS.
- **Namenode**: The centerpiece of an HDFS file system.
    - ○ Keeps the directory tree of all files in the file system
    - ○ Tracks where across the cluster the file data is kept.
        - ■ Does not store the data of these files itself.
    - ○ Active : Actively serving request
    - ○ Standby: Becomes Active if the current Active node fails

Tasks of NameNode

- Manages File System
  - mapping files to blocks and blocks to data nodes
- Maintaining status of data nodes
  - Heartbeat
    - Datanode sends heartbeat at regular intervals
    - If heartbeat is not received, datanode is declared dead
  - Blockreport
    - DataNode sends list of blocks on it
    - Used to check health of HDFS

What is HDFS?

Master/slave nodes typically form the HDFS cluster



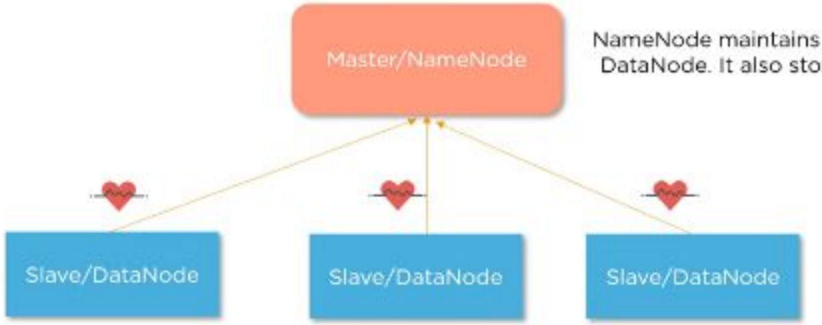
NameNode Functions

- Replication
  - On Datanode failure
  - On Disk failure
  - On Block corruption
- Data integrity
  - Checksum for each block
  - Stored in hidden file

- Rebalancing - balancer tool
  - Addition of new nodes
  - Decommissioning
  - Deletion of some files

What is HDFS?

Master/slave nodes typically form the HDFS cluster



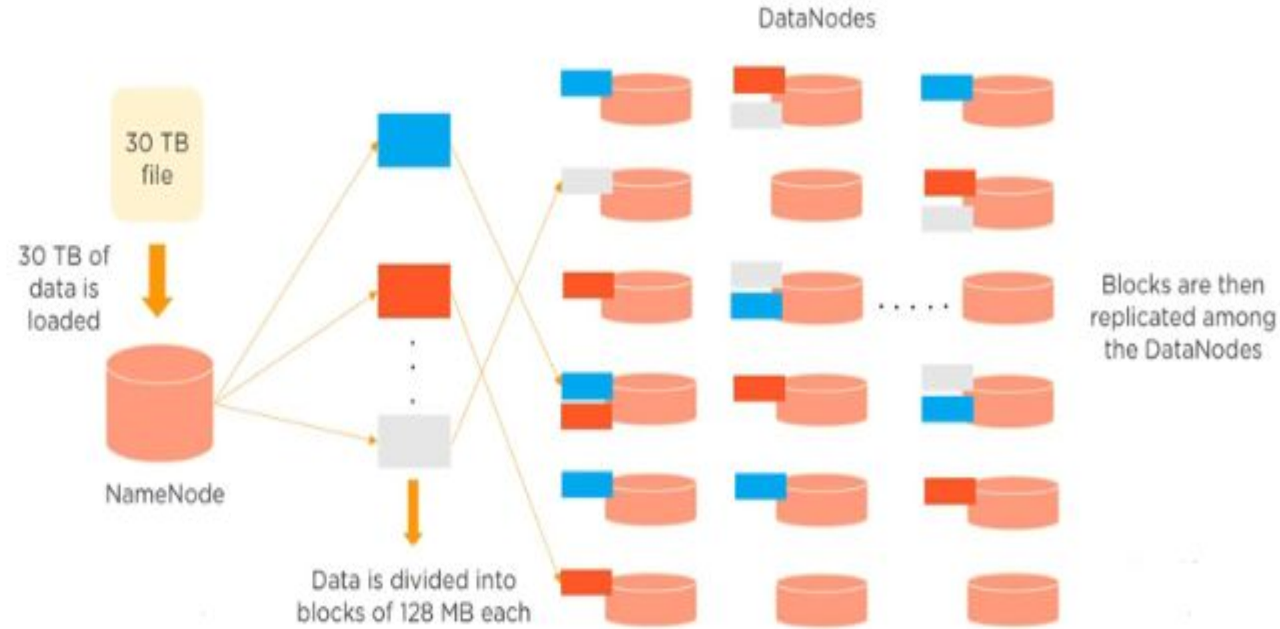
NameNode maintains and manages the DataNode. It also stores the metadata

DataNodes stores the actual data, does reading, writing and processing. Performs replication as well

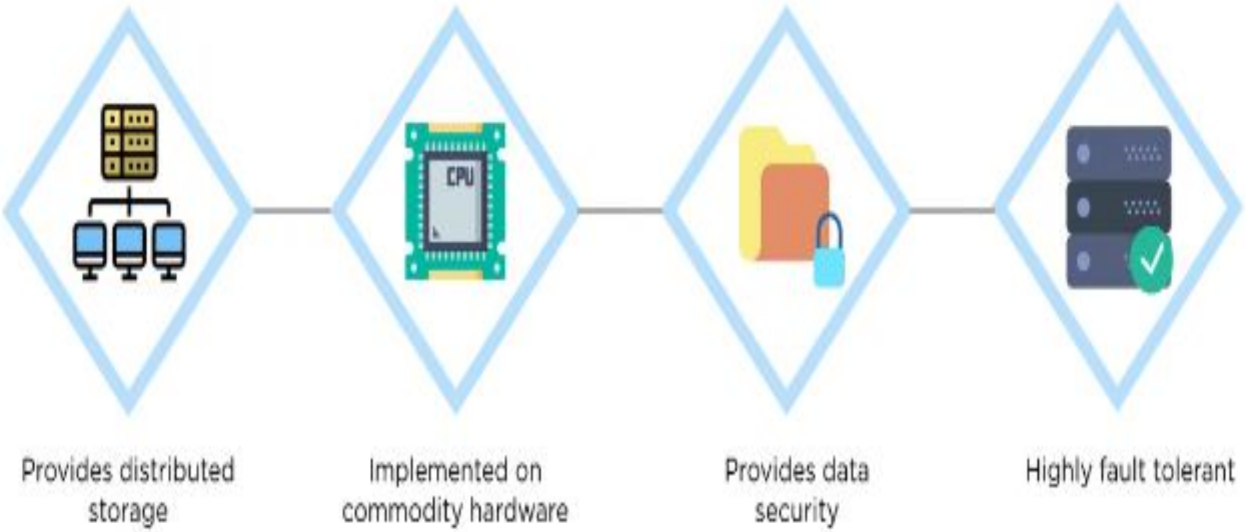
HeartBeat is the signal that DataNode continuously sends to the NameNode. This signal shows the status of the DataNode

What is HDFS?

In HDFS, data is stored in a distributed manner


DataNodes
30 TB file
30 TB of data is loaded
NameNode
Data is divided into blocks of 128 MB each
Blocks are then replicated among the DataNodes

Features of HDFS



Provides distributed storage

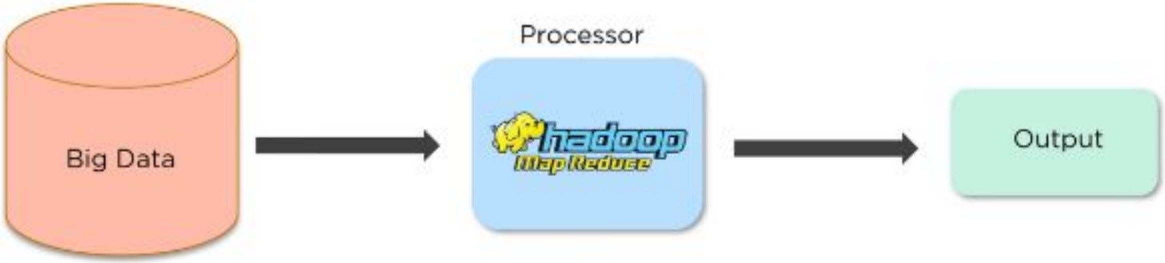
Implemented on commodity hardware

Provides data security

Highly fault tolerant

What is MapReduce?

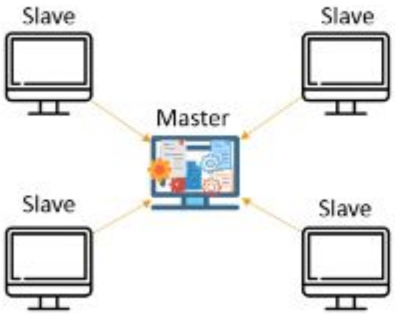
Hadoop MapReduce is a programming technique where huge data is processed in a parallel and distributed fashion



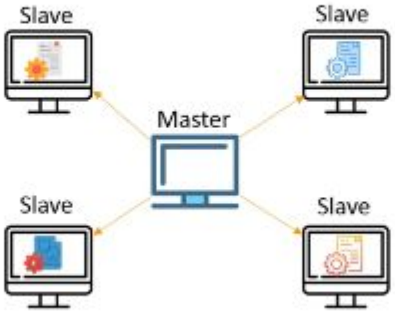
MapReduce is used for parallel processing of the Big Data, which is stored in HDFS

What is MapReduce?

In MapReduce approach, processing is done at the slave nodes and the final result is sent to the master node

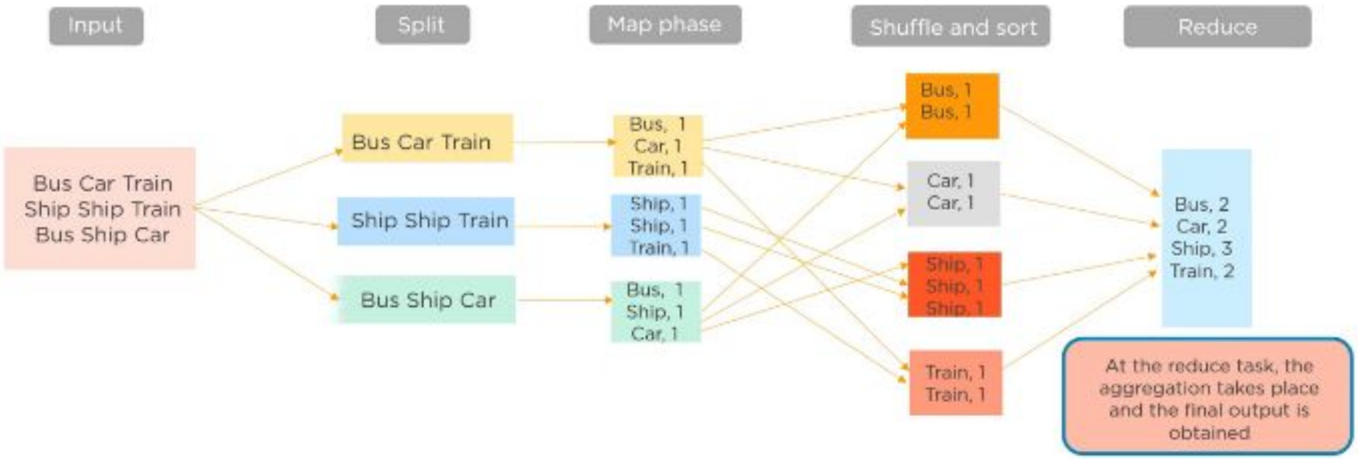


Traditional approach - Data is processed at the Master node



MapReduce approach - Data is processed at the Slave nodes

What is MapReduce?



Input

Split

Map phase

Shuffle and sort

Reduce

Bus Car Train
Ship Ship Train
Bus Ship Car

Bus Car Train

Ship Ship Train

Bus Ship Car

Bus, 1
Car, 1
Train, 1

Ship, 1
Ship, 1
Train, 1

Bus, 1
Ship, 1
Car, 1

Bus, 1
Bus, 1

Car, 1
Car, 1

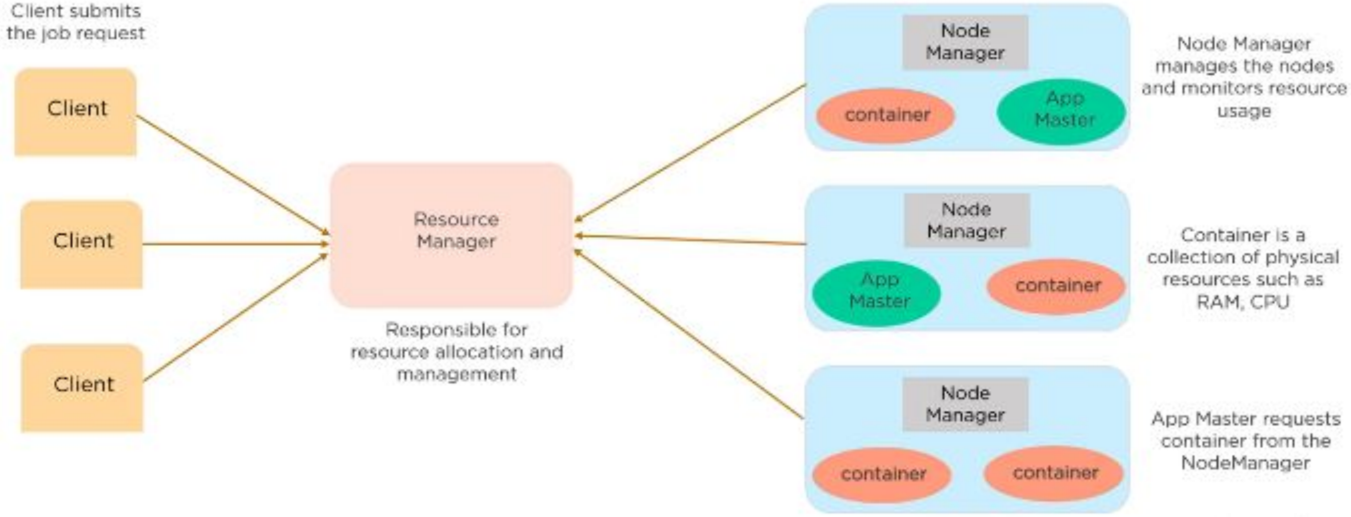
Ship, 1
Ship, 1
Ship, 1

Train, 1
Train, 1

Bus, 2
Car, 2
Ship, 3
Train, 2

At the reduce task, the aggregation takes place and the final output is obtained

What is YARN?



Client submits the job request

Responsible for resource allocation and management

Node Manager manages the nodes and monitors resource usage

Container is a collection of physical resources such as RAM, CPU

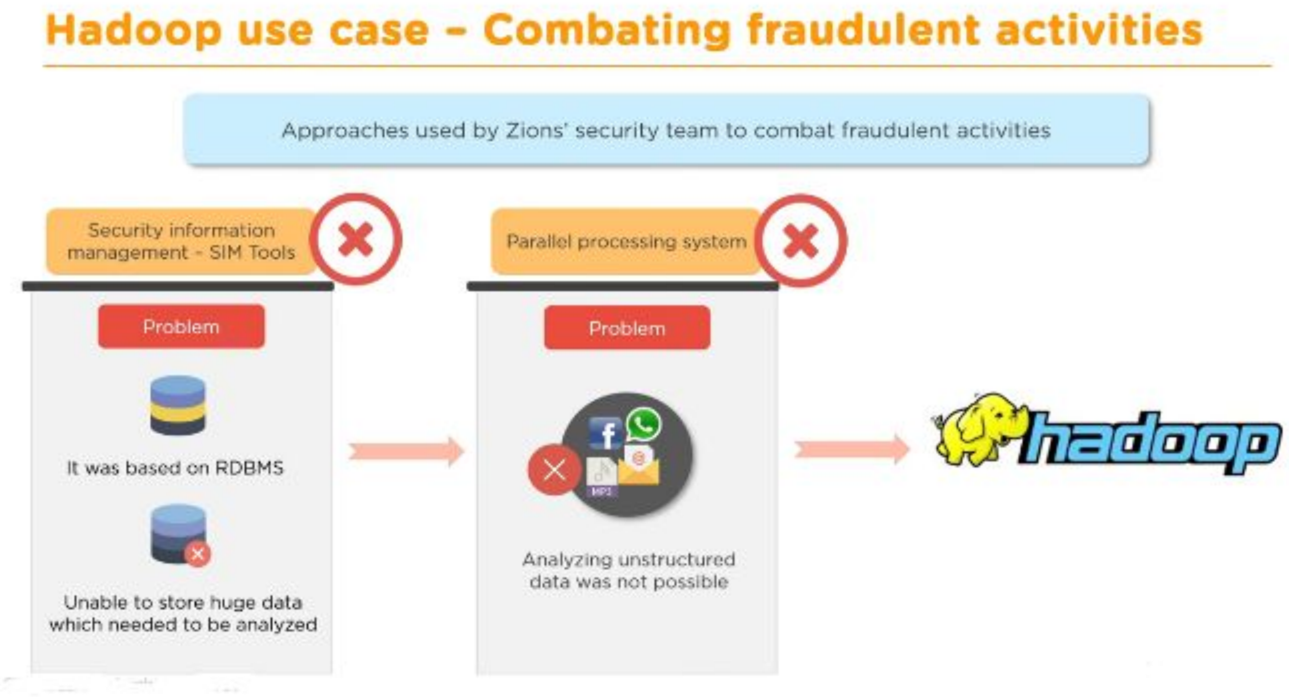
App Master requests container from the NodeManager

Hadoop Case Study

Hadoop use case – Combating fraudulent activities

Detecting fraudulent transactions is one among the various problems any bank faces

Fraud activities



Hadoop use case – Combating fraudulent activities

Approaches used by Zions' security team to combat fraudulent activities

Security information management - SIM Tools

Problem

It was based on RDBMS

Unable to store huge data which needed to be analyzed

Parallel processing system

Problem

Analyzing unstructured data was not possible

Hadoop use case – Combating fraudulent activities

How Hadoop solved the problems

Storing

Zions could now store massive amount of data using Hadoop

Processing

Processing of unstructured data (like server logs, customer data, customer transactions) was now possible

Analyzing

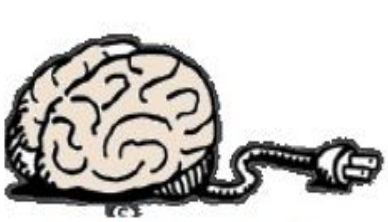
In-depth analysis of different data formats became easy and time efficient

Detecting

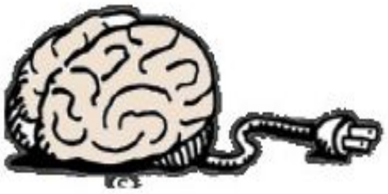
The team could now detec everything from malware spear phishing attempts t account takeovers

Components of HDFS

Secondary NameNode



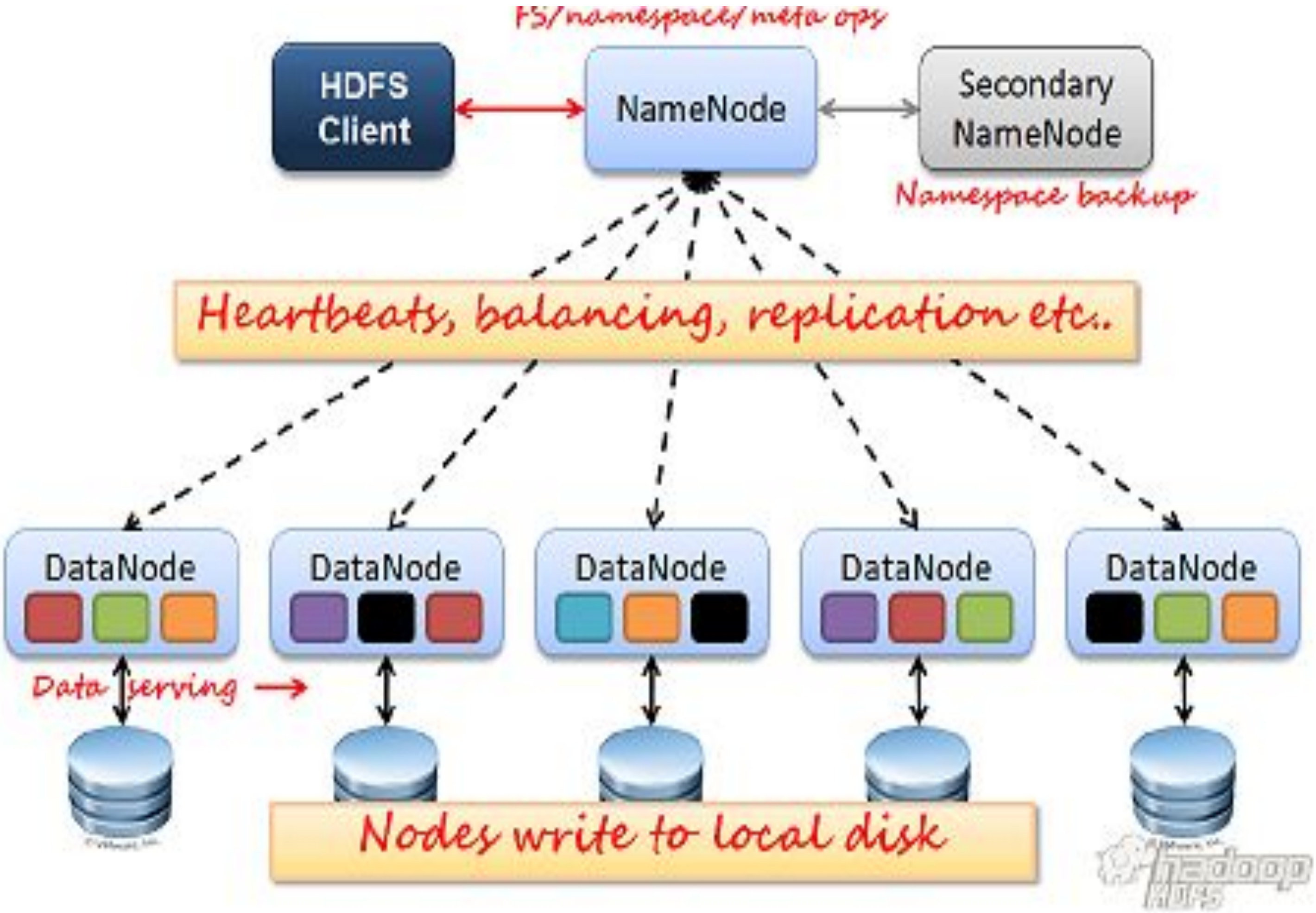
Active NameNode



Standby NameNode

DataNodes

Architecture



FS/namespace/meta ops

Namespace backup

Heartbeats, balancing, replication etc..

Data serving →

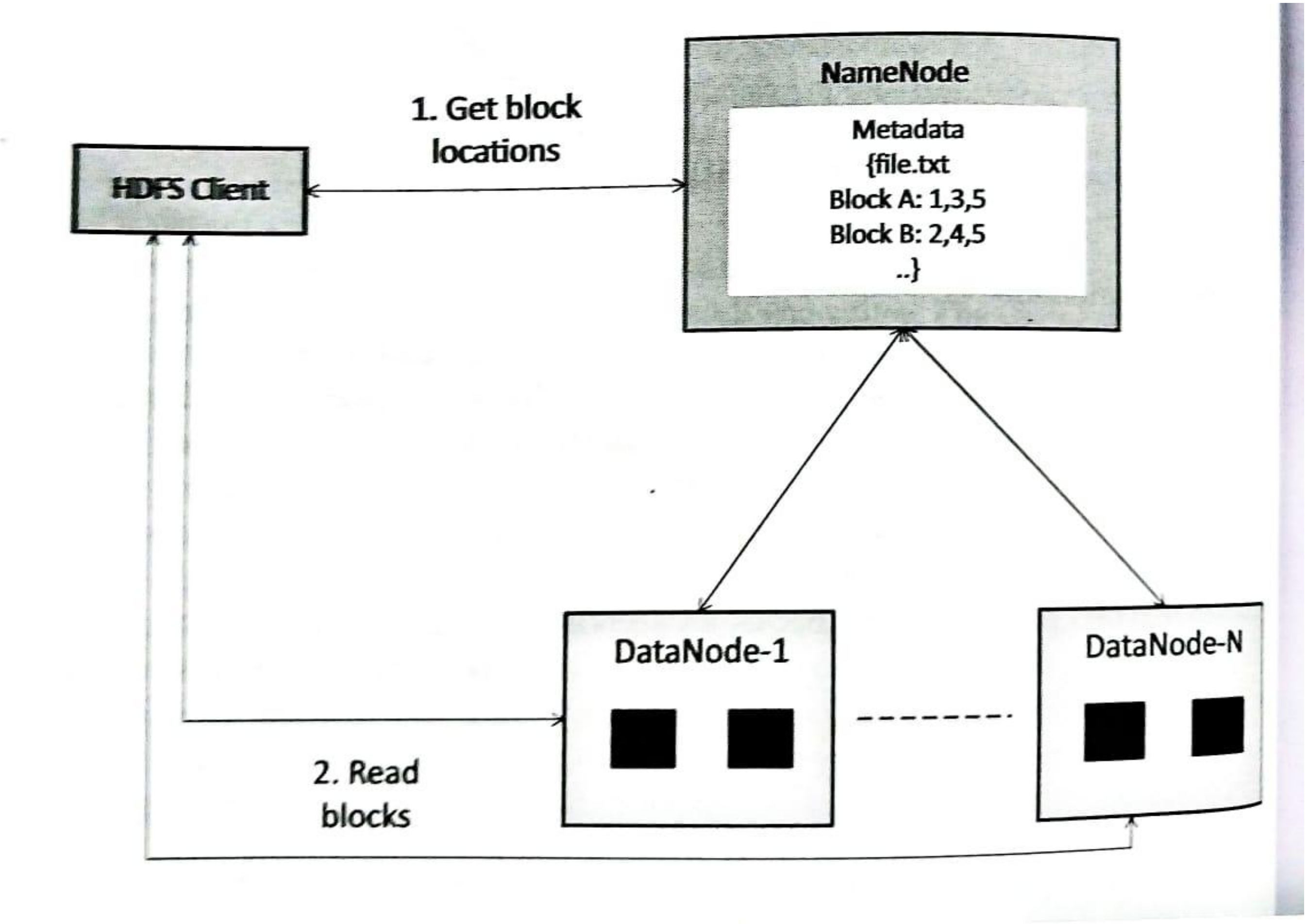
Nodes write to local disk

Storing file on HDFS

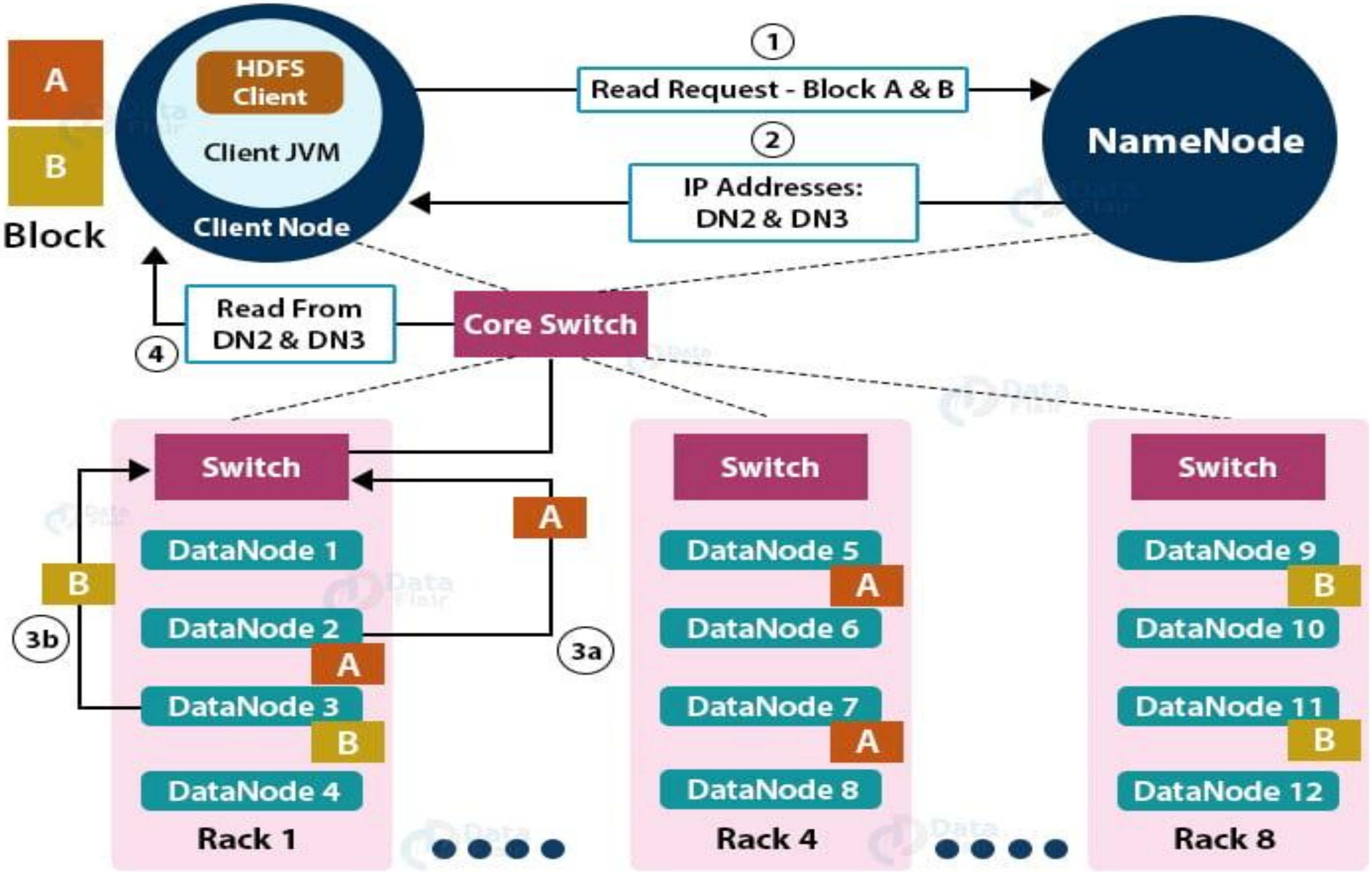
Motivation: Reliability, Availability , Network Bandwidth

- The input file (say 1 TB) is split into smaller chunks/blocks of 128 MB
- The chunks are stored on multiple nodes as independent files on data nodes

Data Read operation in HADOOP



HDFS- Read Operations



- As the NameNode stores the block’s metadata for the file “File.txt’, the client will reach out to NameNode asking locations of DataNodes containing data blocks.
- The NameNode first checks for required privileges, and if the client has sufficient privileges, the NameNode sends the locations of DataNodes containing blocks (A and B).

- NameNode also gives a **security token** to the client, which they need to show to the DataNodes for authentication
- Let the NameNode provide the following list of IPs for block A and B – for block A, location of DataNodes D2, D5, D7, and for block B, location of DataNodes D3, D9, D11.

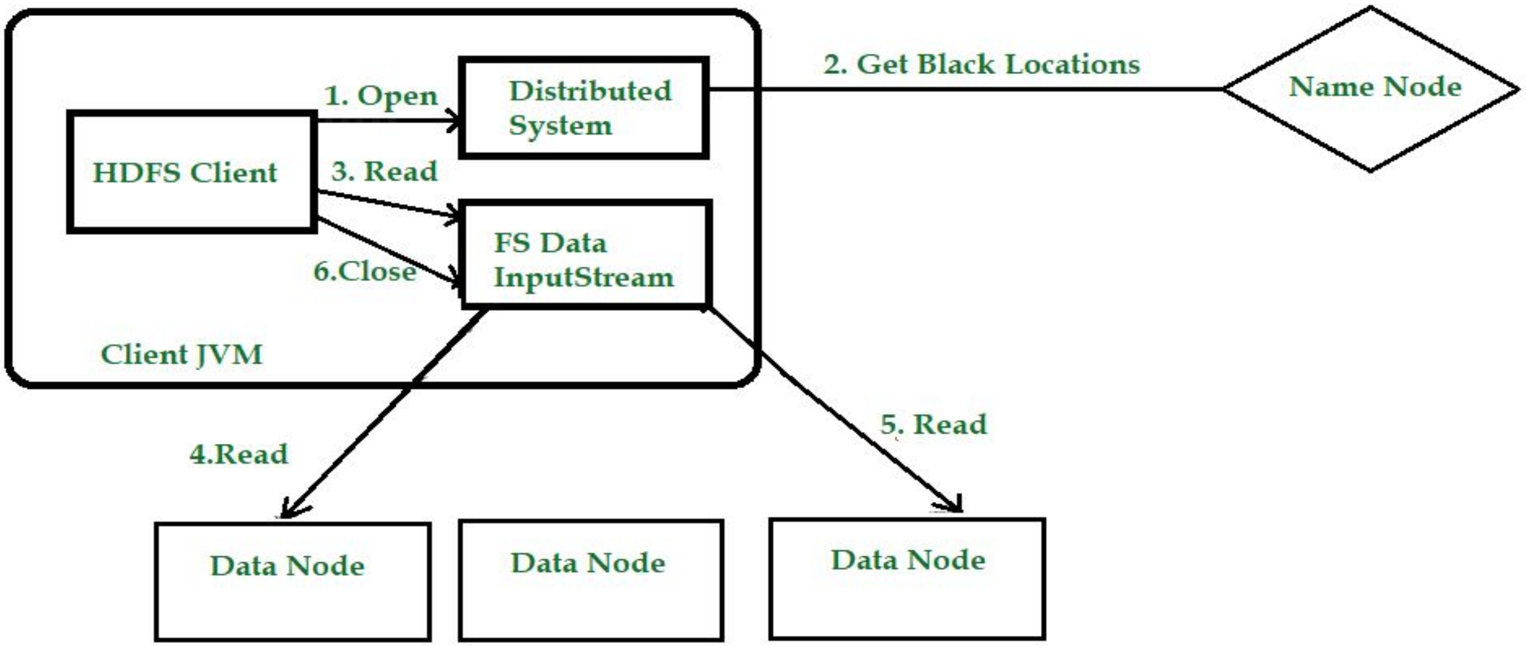
- After receiving the addresses of the DataNodes, the client directly interacts with the DataNodes. The client will send a request to the closest DataNodes (D2 for block A and D3 for block B) through the **FSDataInputstream** object. The **DFSInputstream** manages the interaction between client and DataNode.

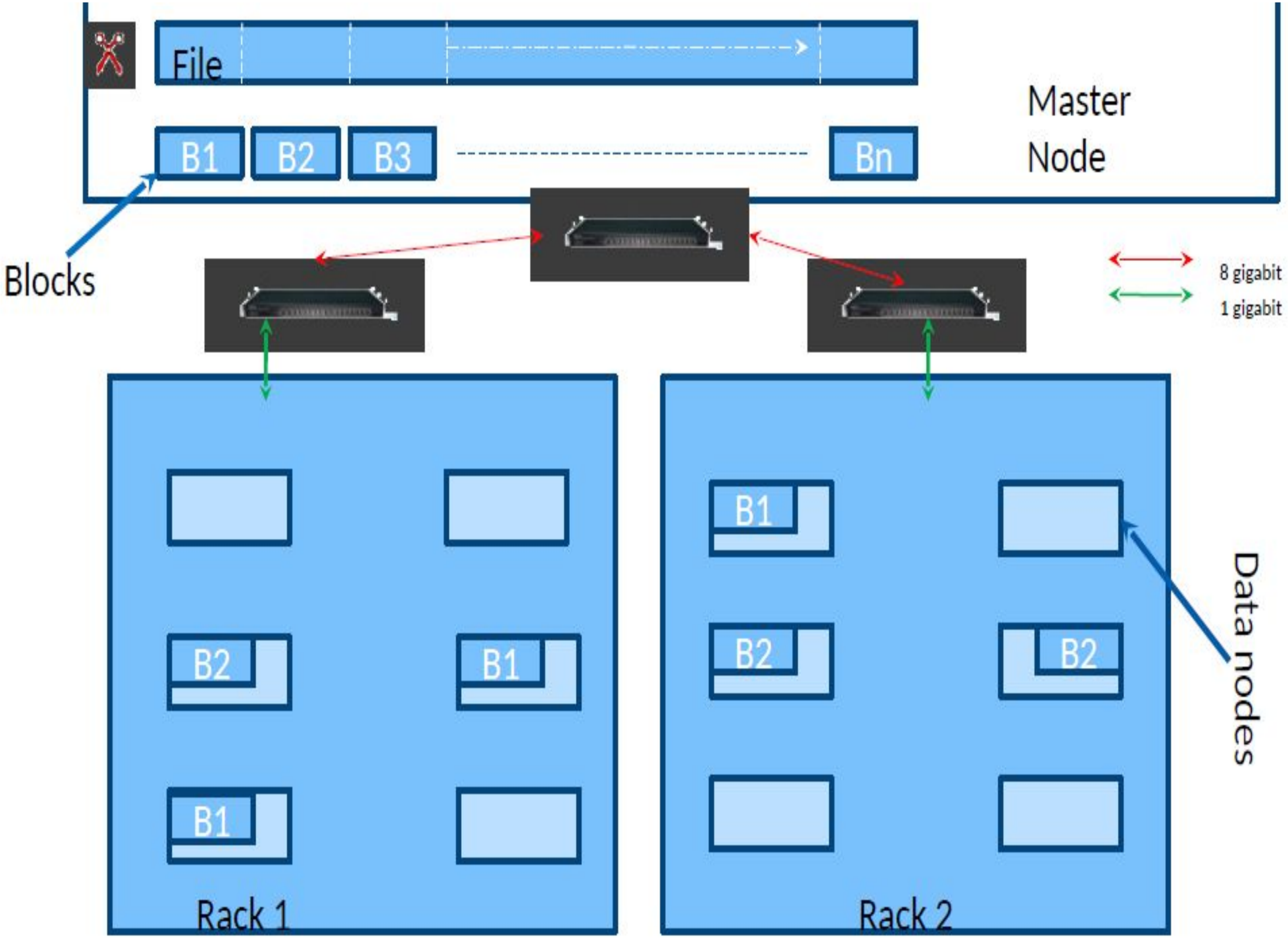
- The client will show the security tokens provided by NameNode to the DataNodes and start reading data from the DataNode. The data will flow directly from the DataNode to the client.
- After reading all the required file blocks, the client calls close() method on the FSDataInputStream object

How to Read a file from HDFS – Java Program

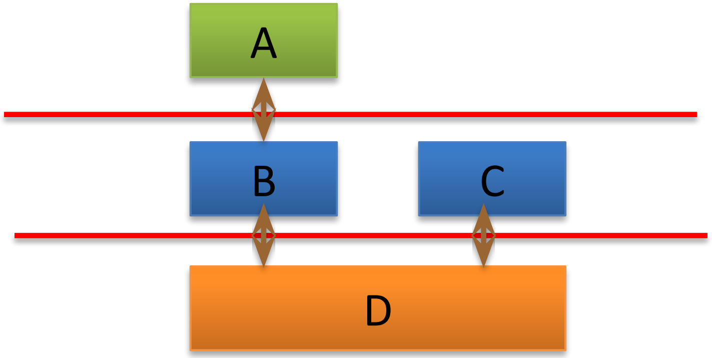
- FileSystem fileSystem = FileSystem.get(conf);
- Path path = **new** Path("/path/to/file.ext");
- **if** (!fileSystem.exists(path)) {
- System.out.println("File does not exists");
- **return**;
- }
- FSDataInputStream **in** = fileSystem.open(path);
- int numBytes = 0;
- **while** ((numBytes = **in**.read(b))> 0) {
- System.out.prinln((char)numBytes));// code to manipulate the data which is read
- }
- **in**.close();
- out.close();
- fileSystem.close();

File read

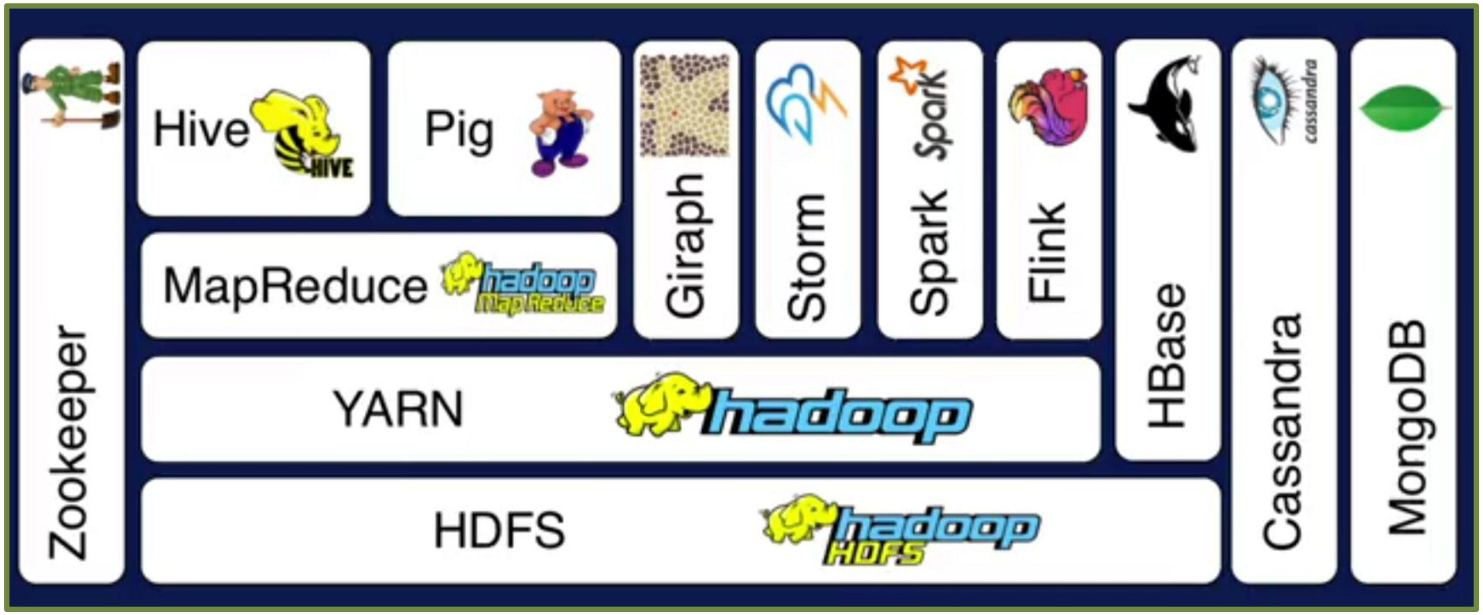

2. Get Black Locations
Name Node
1. Open
Distributed System
HDFS Client
3. Read
FS Data InputStream
6.Close
Client JVM
4.Read
5. Read
Data Node
Data Node
Data Node


File
B1
B2
B3
Bn
Master
Node
Blocks
8 gigabit
1 gigabit
B2
B1
B1
B1
B2
B2
B2
Data nodes
Rack 1
Rack 2

Hadoop Ecosystem

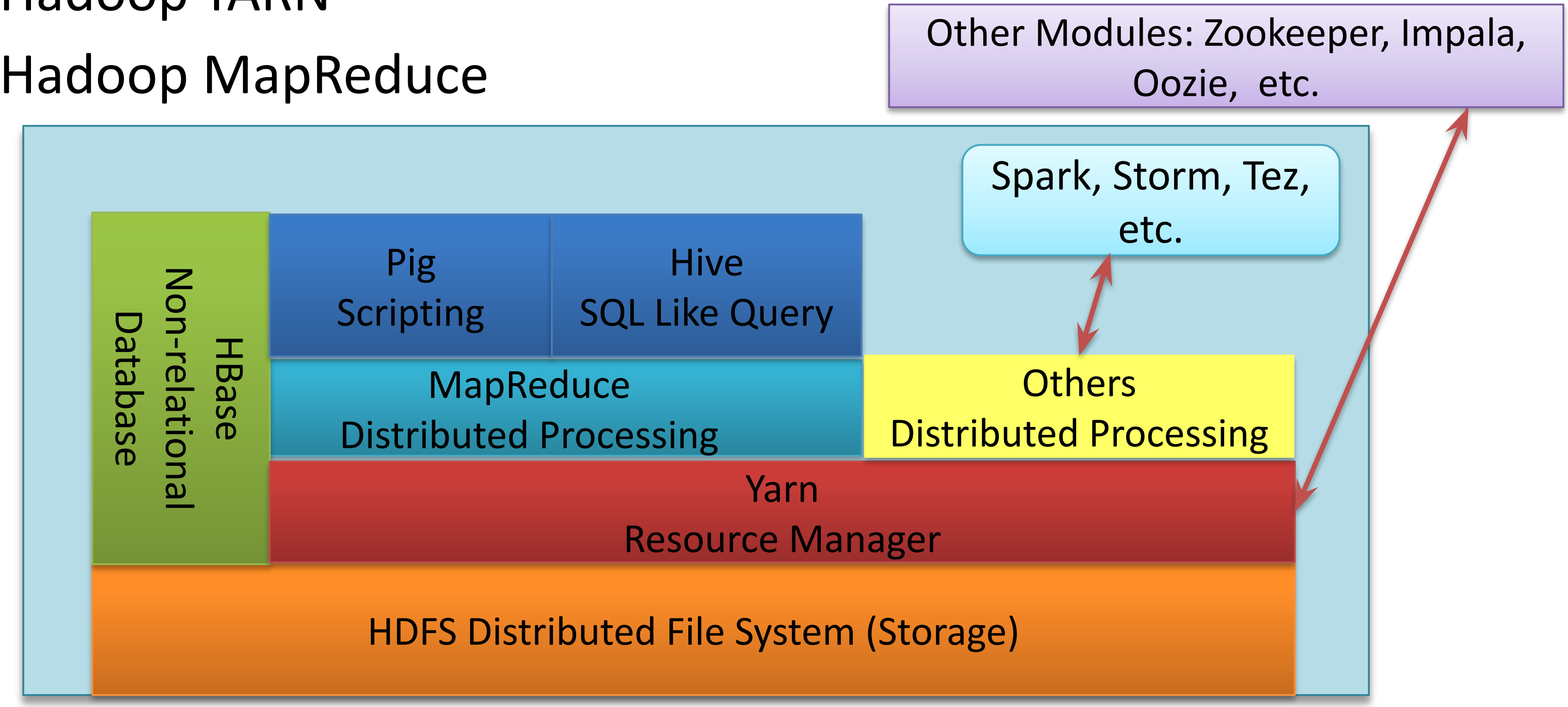


Layer Diagram

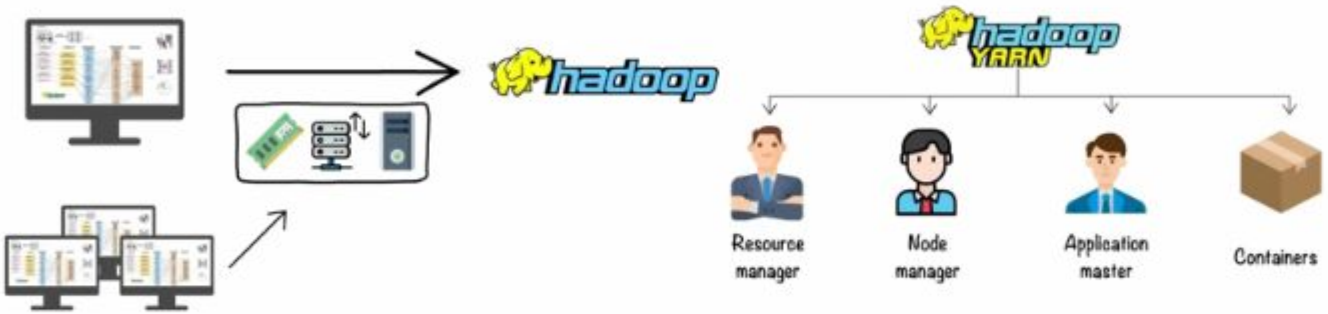


Apache Hadoop Basic Modules

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Hadoop YARN
- Hadoop MapReduce


Other Modules: Zookeeper, Impala, Oozie, etc.
Spark, Storm, Tez, etc.
HBase Non-relational Database
Pig Scripting
Hive SQL Like Query
MapReduce Distributed Processing
Others Distributed Processing
Yarn Resource Manager
HDFS Distributed File System (Storage)

3. YARN



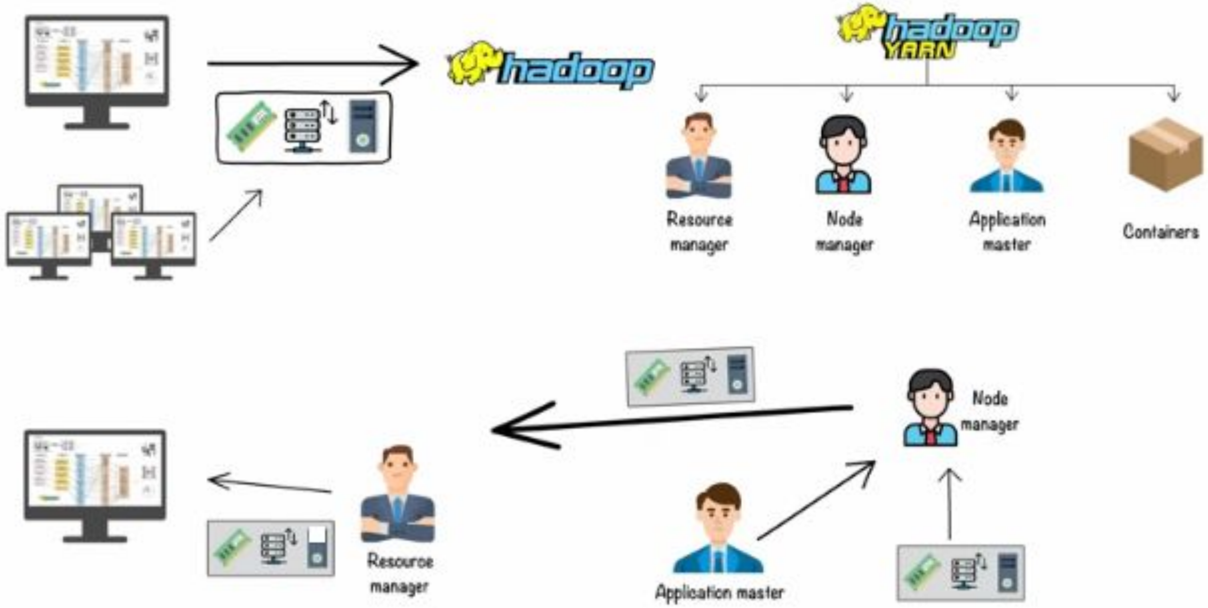
Resource manager

Node manager

Application master

Containers

3. YARN



YARN processes job requests and manages cluster resources

Hadoop HDFS

- Hadoop distributed File System (based on Google File System (GFS) paper, 2004)
  - Serves as the distributed file system for most tools in the Hadoop ecosystem
  - Scalability for large data sets
  - Reliability to cope with hardware failures
- HDFS good for:
  - Large files
  - Streaming data
- Not good for:
  - Lots of small files
  - Random access to files
  - Low latency access

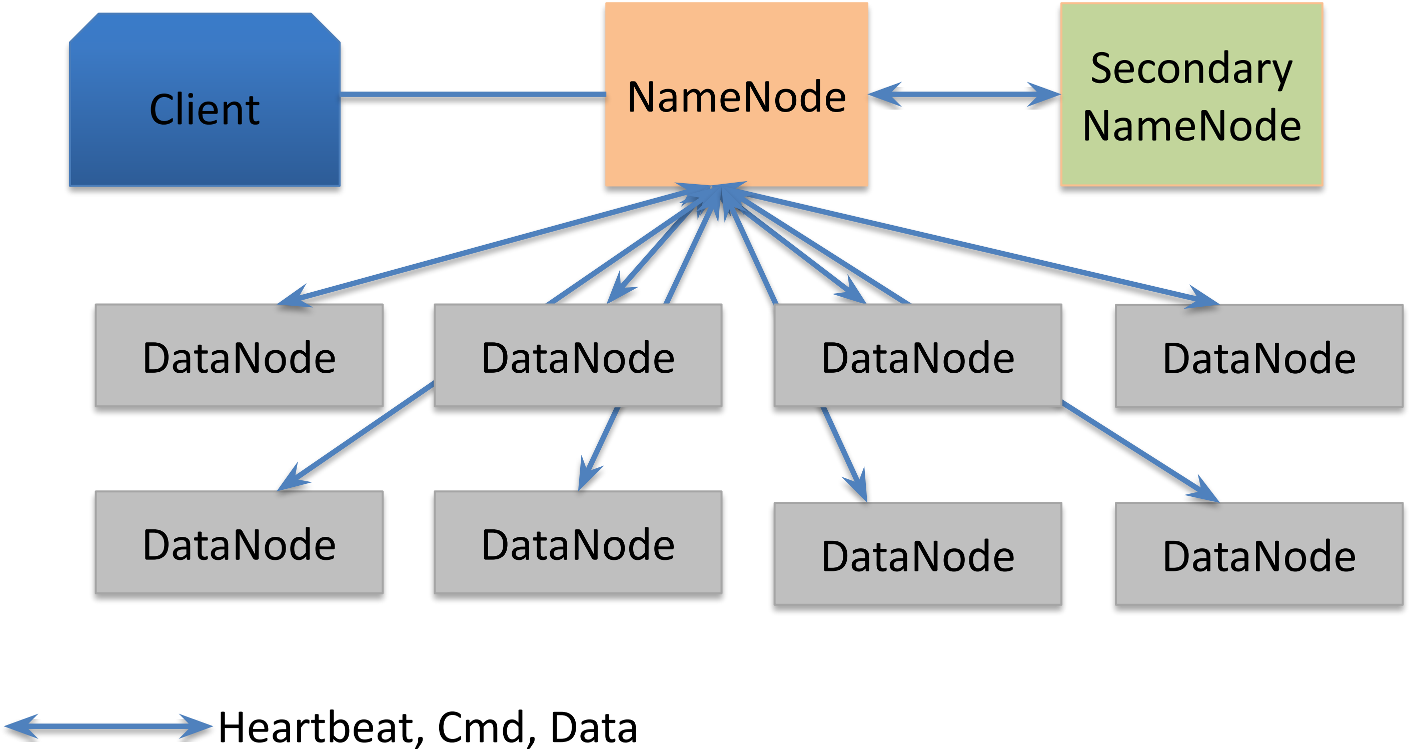
Single Hadoop cluster with 5000 servers and 250 petabytes of data

Design of Hadoop Distributed File System (HDFS)

- Master-Slave design
- Master Node
  - Single NameNode for managing metadata
- Slave Nodes
  - Multiple DataNodes for storing data
- Other
  - Secondary NameNode as a backup

HDFS Architecture

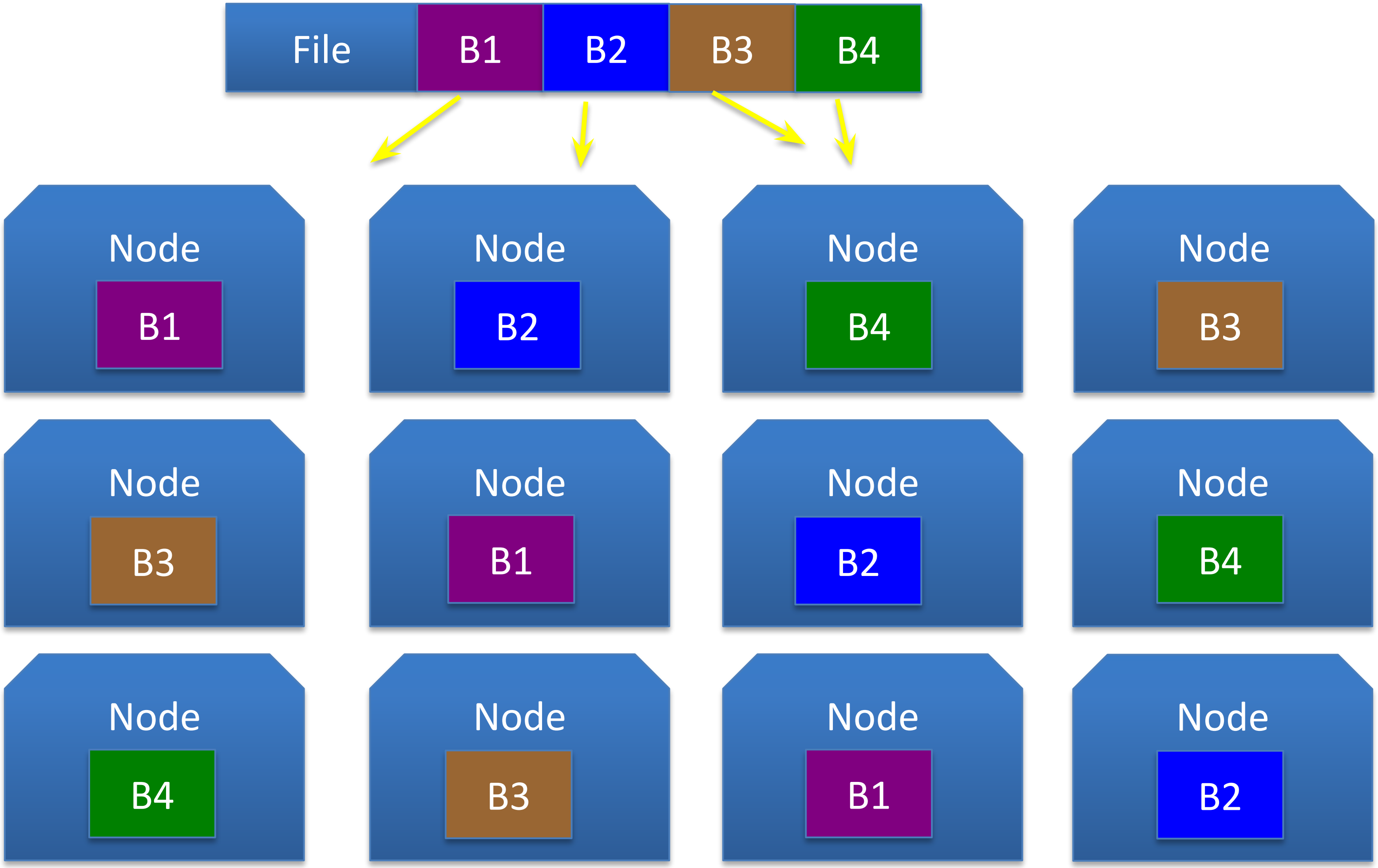
NameNode keeps the metadata, the name, location and directory
DataNode provide storage for blocks of data



HDFS

What happens; if node(s) fail?

Replication of Blocks for fault tolerance



HDFS

- HDFS files are divided into blocks
  - It’s the basic unit of read/write
  - Default size is 64MB, could be larger (128MB)
  - Hence makes HDFS good for storing larger files
- HDFS blocks are replicated multiple times
  - One block stored at multiple location, also at different racks (usually 3 times)
  - This makes HDFS storage fault tolerant and faster to read

Few HDFS Shell commands

Create a directory in HDFS

- hadoop fs -mkdir /user/godil/dir1

List the content of a directory

- hadoop fs -ls /user/godil

Upload and download a file in HDFS

- hadoop fs -put /home/godil/file.txt   /user/godil/datadir/
- hadoop fs -get /user/godil/datadir/file.txt   /home/

Look at the content of a file

- Hadoop fs -cat /user/godil/datadir/book.txt

Many more commands, similar to Unix

HBase

- NoSQL data store build on top of HDFS
- Based on the Google BigTable paper (2006)
- Can handle various types of data
- Stores large amount of data (TB,PB)
- Column-Oriented data store
- Big Data with random read and writes
- Horizontally scalable

HBase, not to use for

- Not good as a traditional RDBMs (Relational Database Model)
  - Transactional applications
  - Data Analytics

- Not efficient for text searching and processing

MapReduce: Simple Programming for Big Data

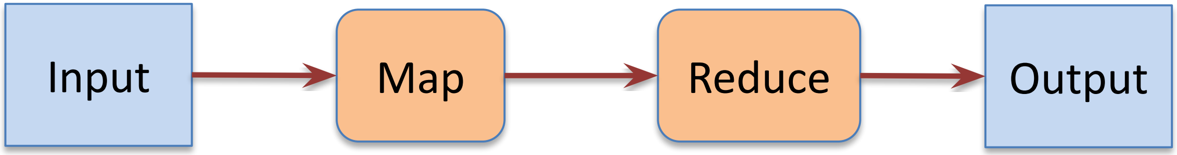
Based on Google's MR paper (2004)

- MapReduce is simple programming paradigm for the Hadoop ecosystem
- Traditional parallel programming requires expertise of different computing/systems concepts
  - examples: multithreads, synchronization mechanisms (locks, semaphores, and monitors)
  - incorrect use: can crash your program, get incorrect results, or severely impact performance
  - Usually not fault tolerant to hardware failure
- The MapReduce programming model greatly simplifies running code in parallel
  - you don't have to deal with any of above issues
  - only need to create, map and reduce functions

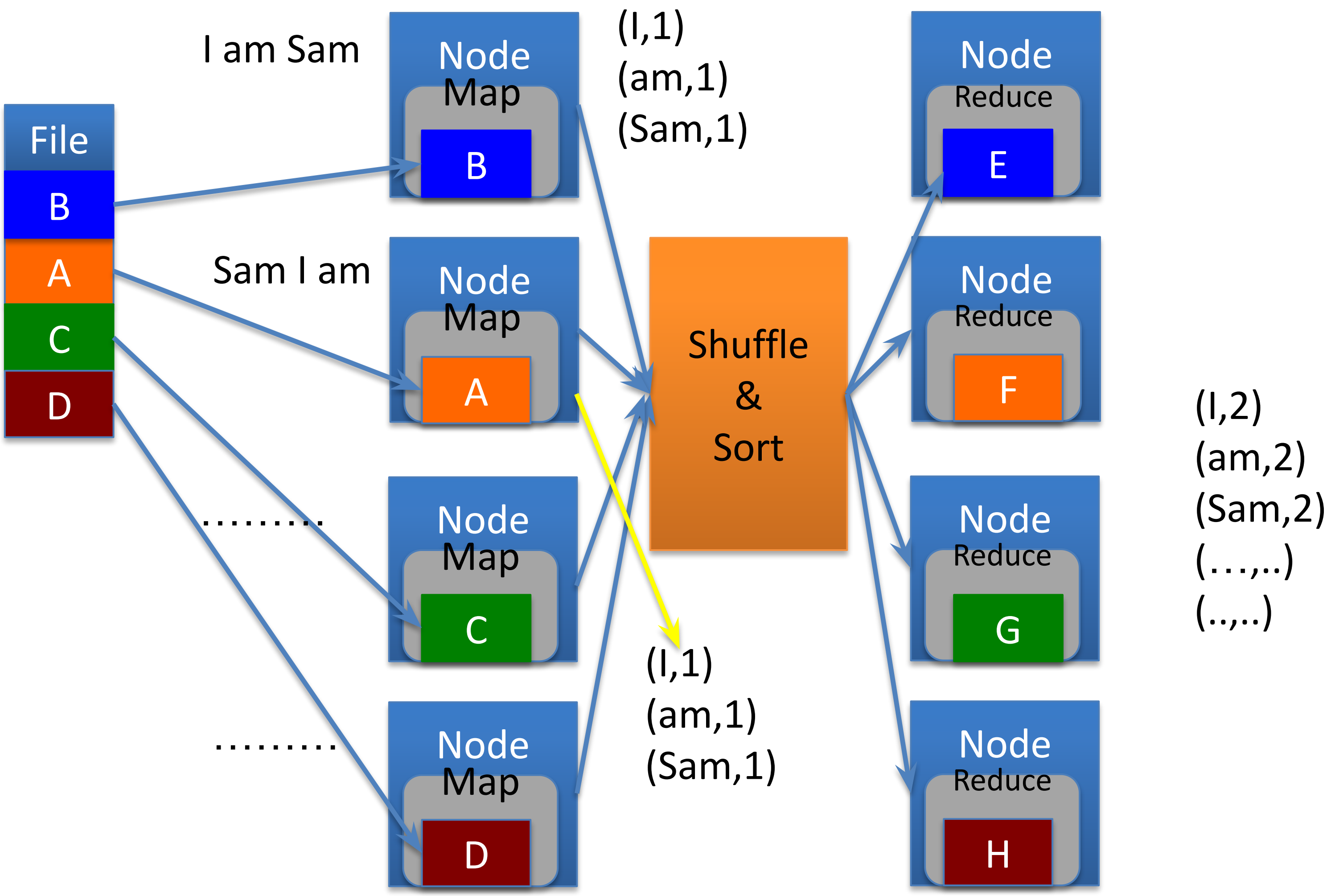
Map Reduce Paradigm

- Map and Reduce are based on functional programming

Map:	**Reduce:**
Apply a function to all the elements of List list1=[1,2,3,4,5]; square x = x * x list2=Map square(list1) print list2 -> [1,4,9,16,25]	Combine all the elements of list for a summary list1 = [1,2,3,4,5]; A = reduce (+) list1 Print A -> 15



MapReduce Word Count Example



Shortcoming of MapReduce

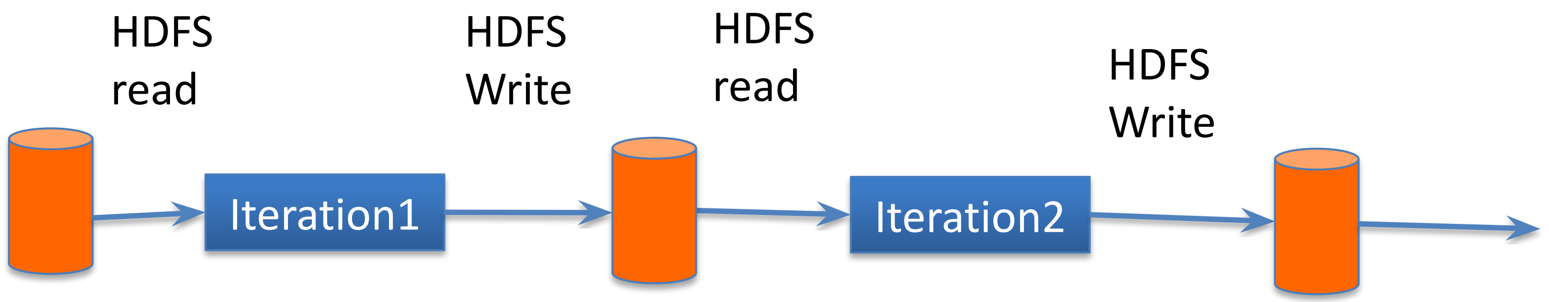
- Forces your data processing into Map and Reduce
  - Other workflows missing include join, filter, flatMap, groupByKey, union, intersection, …
- Based on “Acyclic Data Flow” from Disk to Disk (HDFS)
- Read and write to Disk before and after Map and Reduce (stateless machine)
  - Not efficient for iterative tasks, i.e. Machine Learning
- Only Java natively supported
  - Support for others languages needed
- Only for Batch processing
  - Interactivity, streaming data

One Solution is Apache Spark

- A new general framework, which solves many of the short comings of MapReduce
- It capable of leveraging the Hadoop ecosystem, e.g. HDFS, YARN, HBase, S3, …
- Has many other workflows, i.e. join, filter, flatMapdistinct, groupByKey, reduceByKey, sortByKey, collect, count, first…
  - (around 30 efficient distributed operations)
- In-memory caching of data (for iterative, graph, and machine learning algorithms, etc.)
- Native Scala, Java, Python, and R support
- Supports interactive shells for exploratory data analysis
- Spark API is extremely simple to use
- Developed at AMPLab UC Berkeley, now by Databricks.com

Spark Uses Memory instead of Disk

Hadoop: Use Disk for Data Sharing



HDFS read

Iteration1

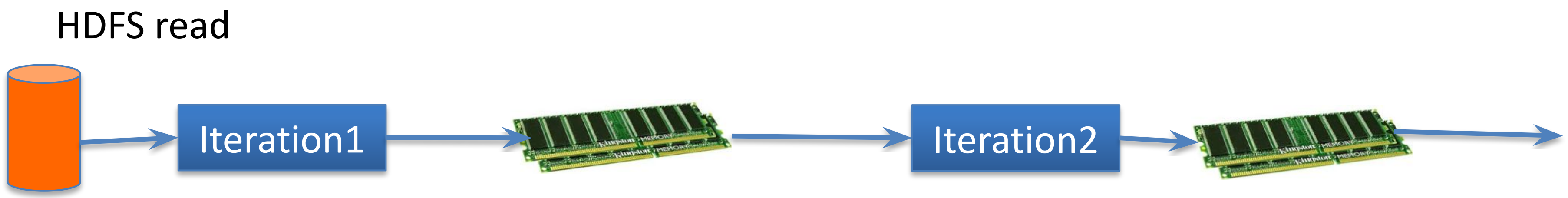
HDFS Write

HDFS read

Iteration2

HDFS Write

Spark: In-Memory Data Sharing



HDFS read

Iteration1

Iteration2

Sort competition

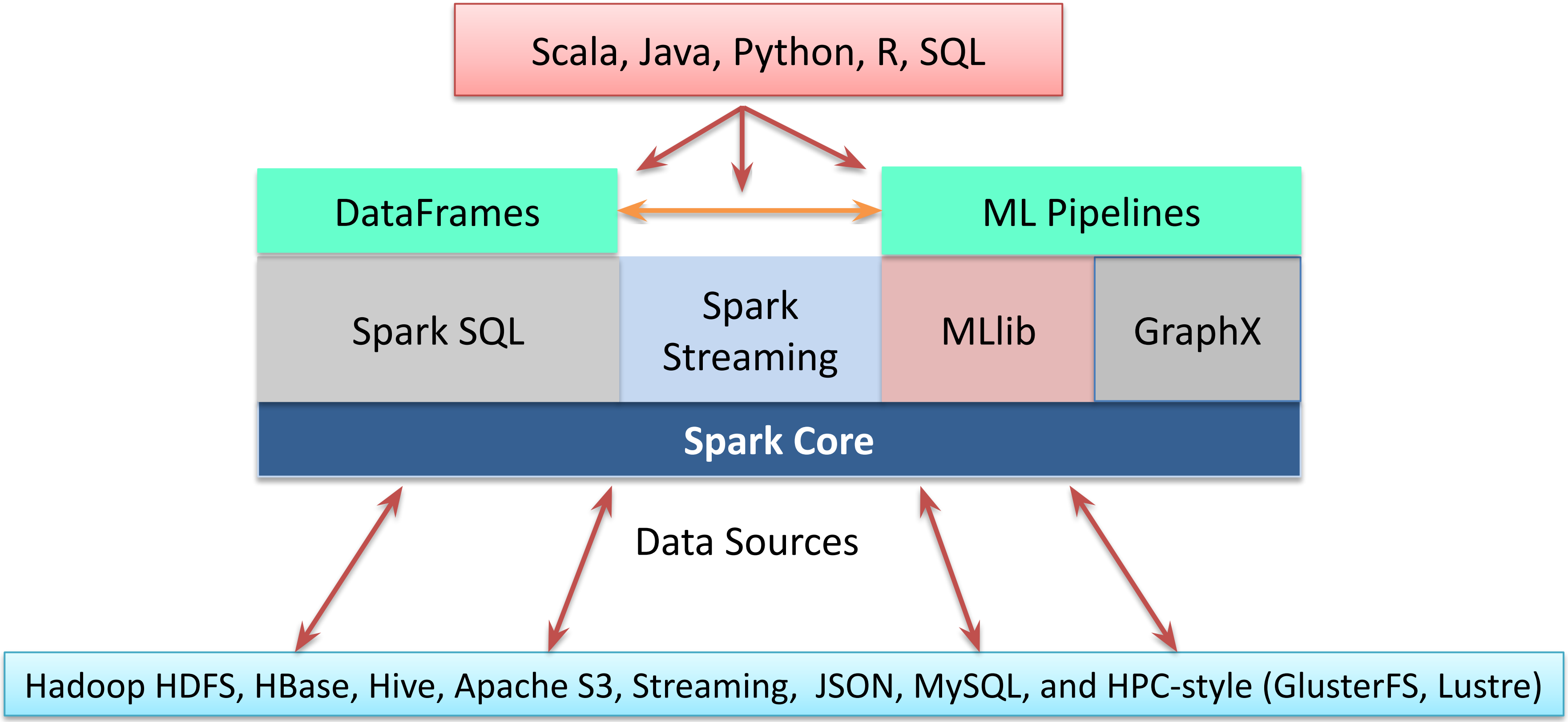
	Hadoop MR Record (2013)	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark, 3x faster with 1/10 the nodes

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)
http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Apache Spark

Apache Spark supports data analysis, machine learning, graphs, streaming data, etc. It can read/write from a range of data types and allows development in multiple languages.


Scala, Java, Python, R, SQL
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
Hadoop HDFS, HBase, Hive, Apache S3, Streaming, JSON, MySQL, and HPC-style (GlusterFS, Lustre)

Resilient Distributed Datasets (RDDs)

- RDDs (Resilient Distributed Datasets) is Data Containers
- All the different processing components in Spark share the same abstraction called RDD
- As applications share the RDD abstraction, you can mix different kind of transformations to create new RDDs
- Created by parallelizing a collection or reading a file
- Fault tolerant

DataFrames & SparkSQL

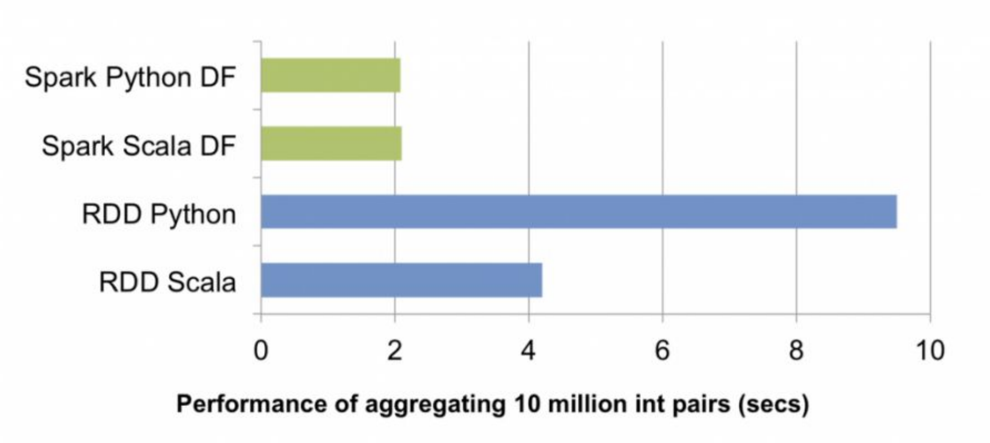
- DataFrames (DFs) is one of the other distributed datasets organized in named columns
- Similar to a relational database, Python Pandas Dataframe or R’s DataTables
  - Immutable once constructed
  - Track lineage
  - Enable distributed computations
- How to construct Dataframes
  - Read from file(s)
  - Transforming an existing DFs(Spark or Pandas)
  - Parallelizing a python collection list
  - Apply transformations and actions

DataFrame example

```
// Create a new DataFrame that contains “students”
students = users.filter(users.age < 21)

//Alternatively, using Pandas-like syntax
students = users[users.age < 21]

//Count the number of students users by gender
students.groupBy("gender").count()

// Join young students with another DataFrame called
logs
students.join(logs, logs.userId == users.userId,
“left_outer")
```

RDDs vs. DataFrames

- RDDs provide a low level interface into Spark
- DataFrames have a schema
- DataFrames are cached and optimized by Spark
- DataFrames are built on top of the RDDs and the core Spark API



Example: performance

Spark Operations

Transformations
(create a new RDD)

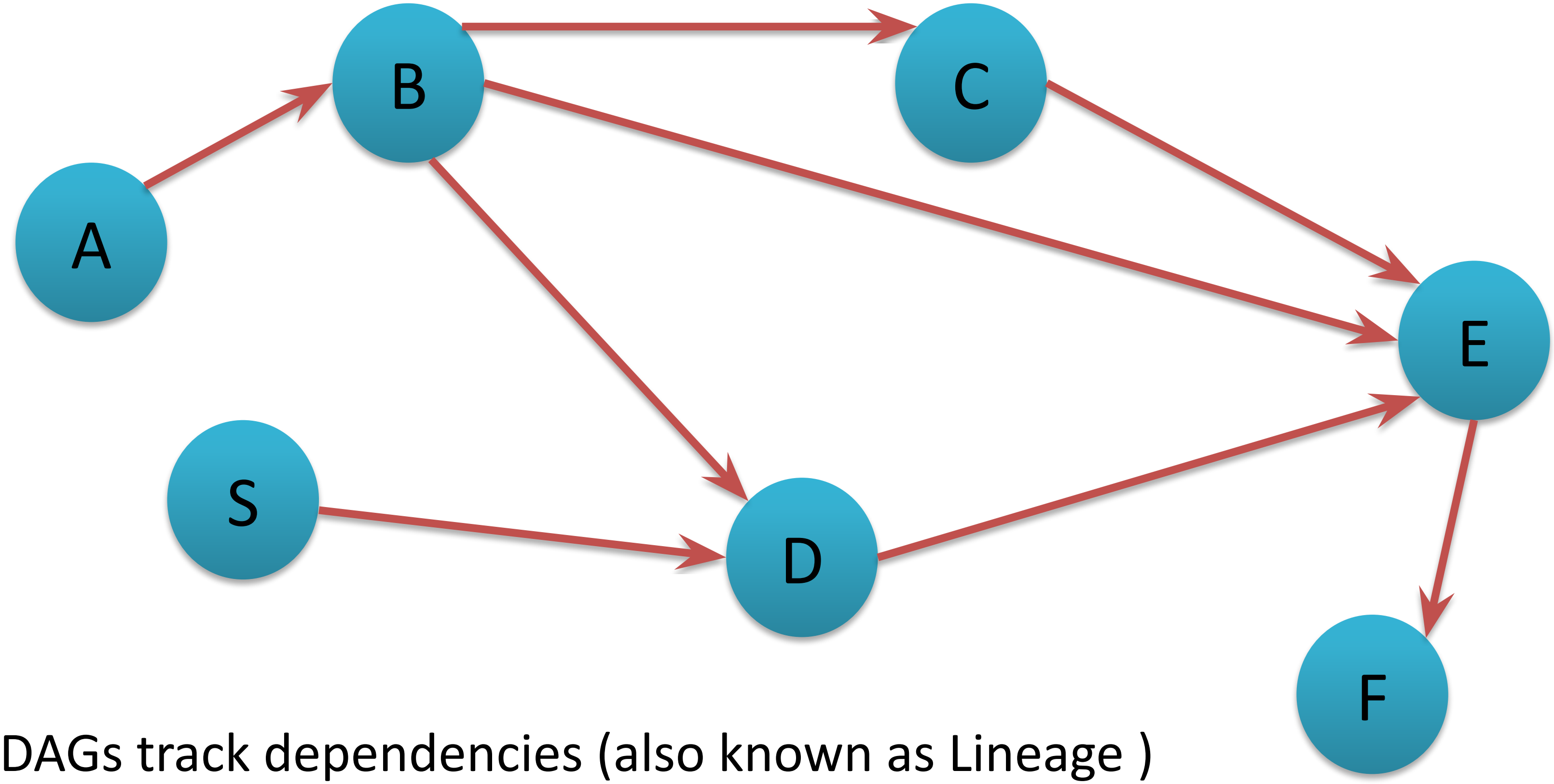
map
filter
sample
groupByKey
reduceByKey
sortByKey
intersection

flatMap
union
join
cogroup
cross
mapValues
reduceByKey

Actions
(return results to driver program)

collect
first
Reduce
take
Count
takeOrdered
takeSample

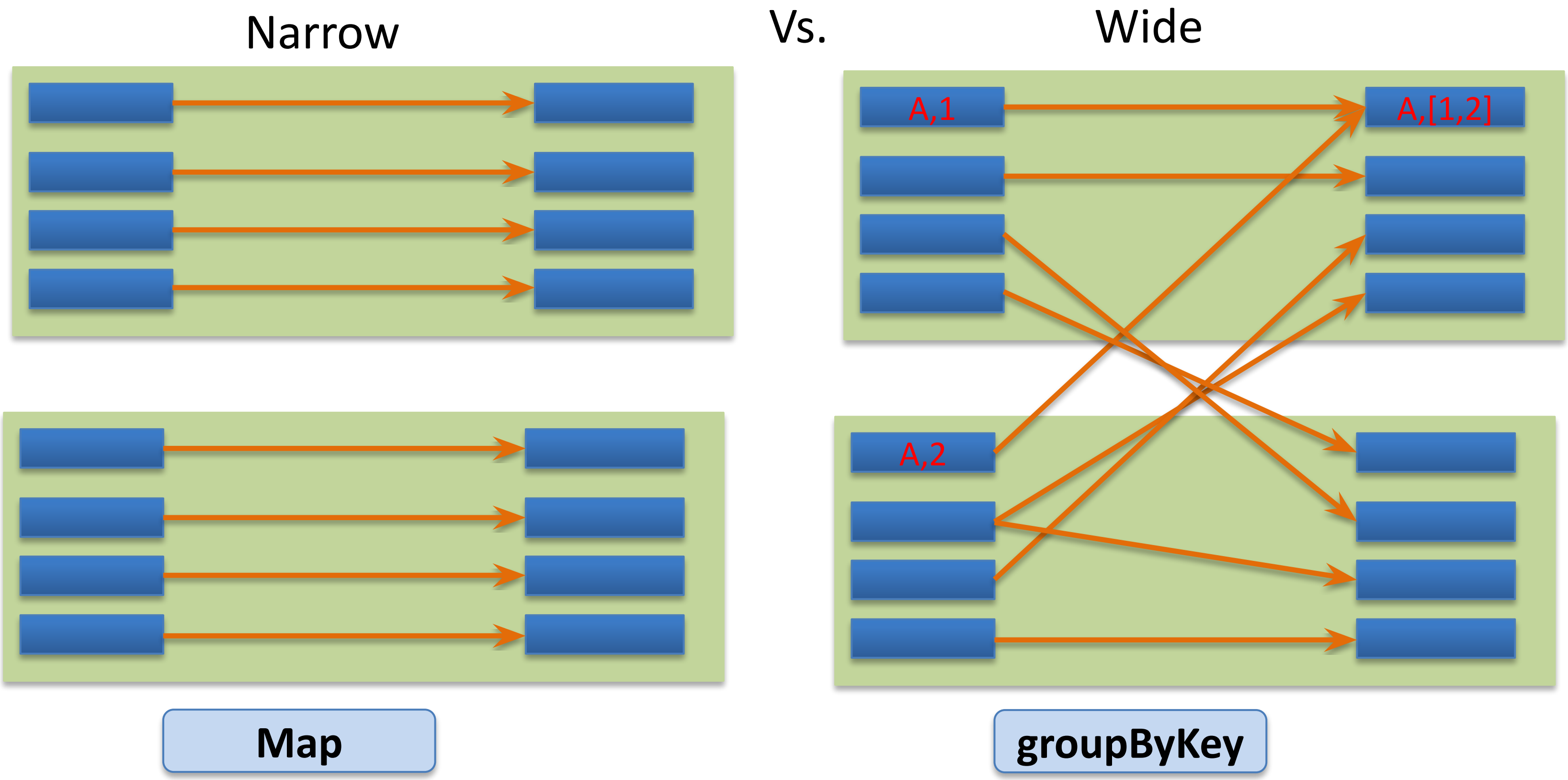
Directed Acyclic Graphs (DAG)



DAGs track dependencies (also known as Lineage)

- nodes are RDDs
- arrows are Transformations

Narrow Vs. Wide transformation



Narrow

Vs.

Wide

A,1

A,[1,2]

A,2

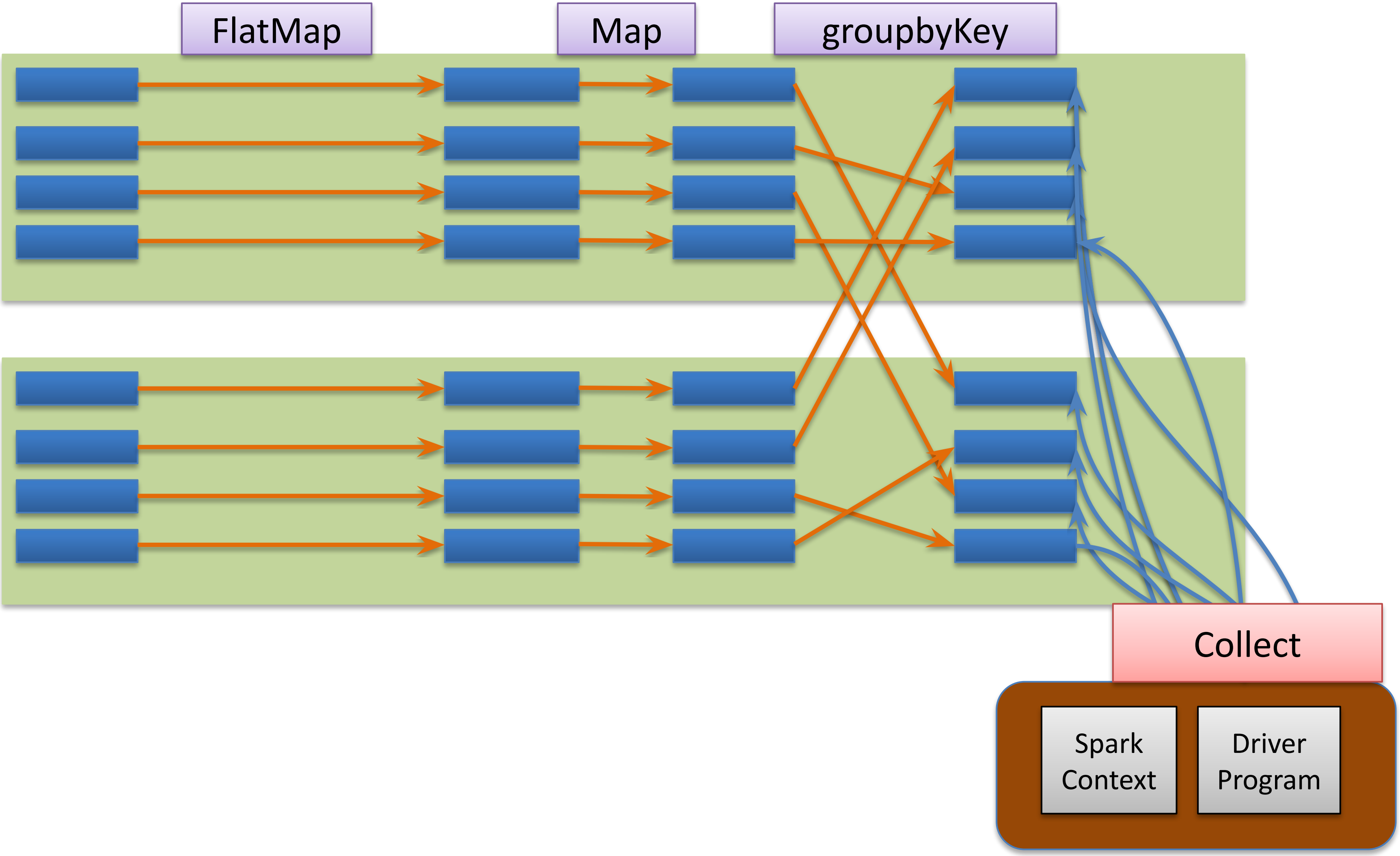
Map

groupByKey

Actions

- What is an action
  - The final stage of the workflow
  - Triggers the execution of the DAG
  - Returns the results to the driver
  - Or writes the data to HDFS or to a file

Spark Workflow



Python RDD API Examples

- Word count

```
text_file = sc.textFile("hdfs://usr/godil/text/book.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
           .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://usr/godil/output/wordCount.txt")
```

- Logistic Regression

```
# Every record of this DataFrame contains the label and
# features represented by a vector.
df = sqlContext.createDataFrame(data, ["label", "features"])
# Set parameters for the algorithm.
# Here, we limit the number of iterations to 10.
lr = LogisticRegression(maxIter=10)
# Fit the model to the data.
model = lr.fit(df)
# Given a dataset, predict each point's label, and show the results.
model.transform(df).show()
```

Examples from http://spark.apache.org/

RDD Persistence and Removal

- RDD Persistence
  - RDD.persist()
  - Storage level:
    - MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, DISK_ONLY,…….

- RDD Removal
  - RDD.unpersist()

Broadcast Variables and Accumulators (Shared Variables)

- Broadcast variables allow the programmer to keep a read-only variable cached on each node, rather than sending a copy of it with tasks

```
>broadcastV1 = sc.broadcast([1, 2, 3,4,5,6])
>broadcastV1.value
[1,2,3,4,5,6]
```

- Accumulators are variables that are only “added” to through an associative operation and can be efficiently supported in parallel

```
accum = sc.accumulator(0)
accum.add(x)
accum.value
```

Spark’s Main Use Cases

- Streaming Data
- Machine Learning
- Interactive Analysis
- Data Warehousing
- Batch Processing
- Exploratory Data Analysis
- Graph Data Analysis
- Spatial (GIS) Data Analysis
- And many more

My Spark Use Cases

- Fingerprint Matching
  - Developed a Spark based fingerprint minutia detection and fingerprint matching code
- Twitter Sentiment Analysis
  - Developed a Spark based Sentiment Analysis code for a Twitter dataset

Spark in the Real World (I)

- Uber – the online taxi company gathers terabytes of event data from its mobile users every day.
  - By using Kafka, Spark Streaming, and HDFS, to build a continuous ETL pipeline
  - Convert raw unstructured event data into structured data as it is collected
  - Uses it further for more complex analytics and optimization of operations

- Pinterest – Uses a Spark ETL pipeline
  - Leverages Spark Streaming to gain immediate insight into how users all over the world are engaging with Pins—in real time.
  - Can make more relevant recommendations as people navigate the site
  - Recommends related Pins
  - Determine which products to buy, or destinations to visit

Spark in the Real World (II)

Here are Few other Real World Use Cases:

- Conviva – 4 million video feeds per month
  - This streaming video company is second only to YouTube.
  - Uses Spark to reduce customer churn by optimizing video streams and managing live video traffic
  - Maintains a consistently smooth, high quality viewing experience.
- Capital One – is using Spark and data science algorithms to understand customers in a better way.
  - Developing next generation of financial products and services
  - Find attributes and patterns of increased probability for fraud
- Netflix – leveraging Spark for insights of user viewing habits and then recommends movies to them.
  - User data is also used for content creation

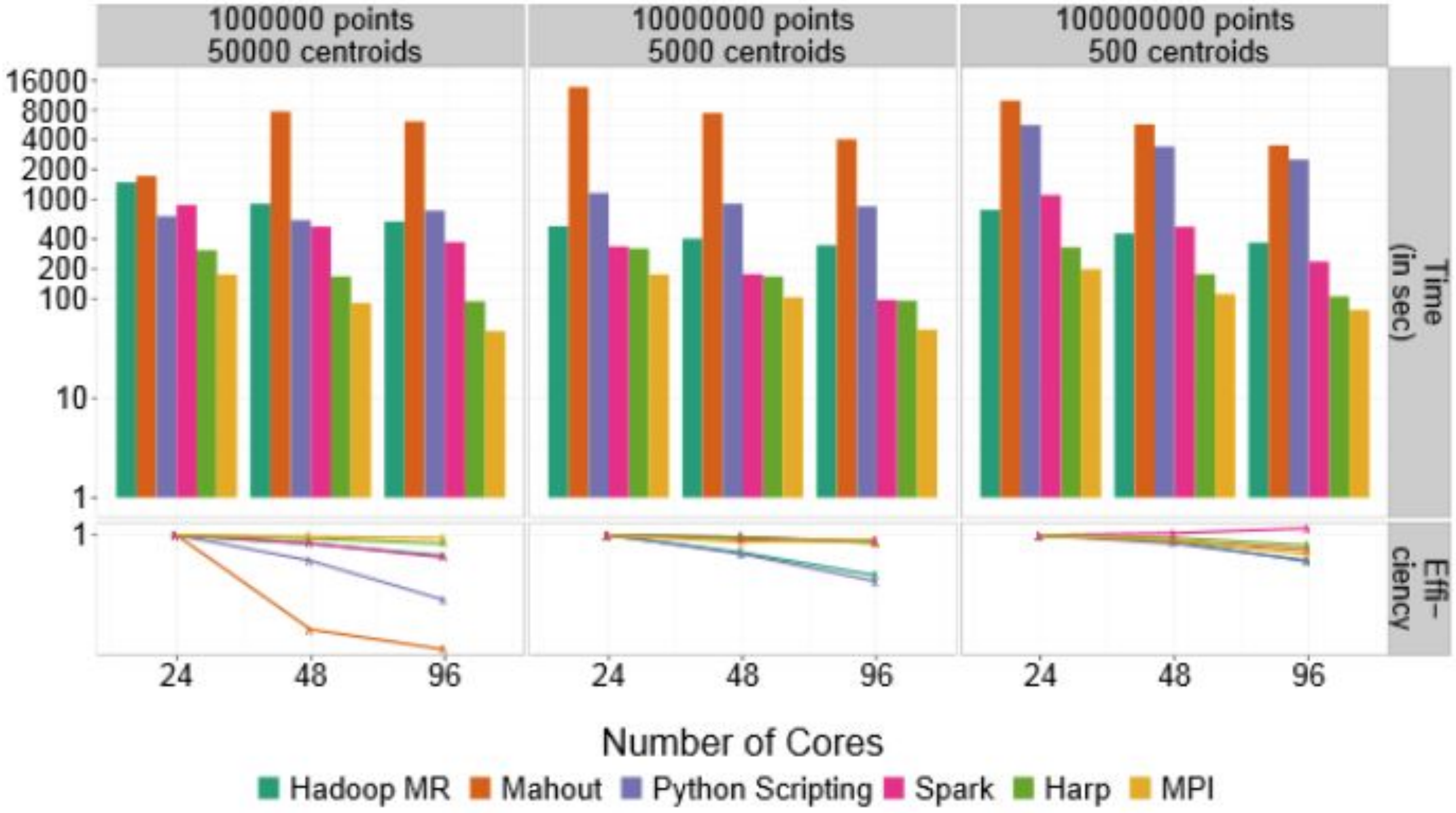
Spark: when not to use

- Even though Spark is versatile, that doesn’t mean Spark’s in-memory capabilities are the best fit for all use cases:
  - For many simple use cases Apache MapReduce and Hive might be a more appropriate choice
  - Spark was not designed as a multi-user environment
  - Spark users are required to know that memory they have is sufficient for a dataset
  - Adding more users adds complications, since the users will have to coordinate memory usage to run code

HPC and Big Data Convergence

- Clouds and supercomputers are collections of computers networked together in a datacenter
- Clouds have different networking, I/O, CPU and cost trade-offs than supercomputers
- Cloud workloads are data oriented vs. computation oriented and are less closely coupled than supercomputers
- Principles of parallel computing same on both
- Apache Hadoop and Spark vs. Open MPI

HPC and Big Data K-Means example



MPI definitely outpaces Hadoop, but can be boosted using a hybrid approach of other technologies that blend HPC and big data, including Spark and HARP. Dr. Geoffrey Fox, Indiana University. (http://arxiv.org/pdf/1403.1528.pdf)

Conclusion

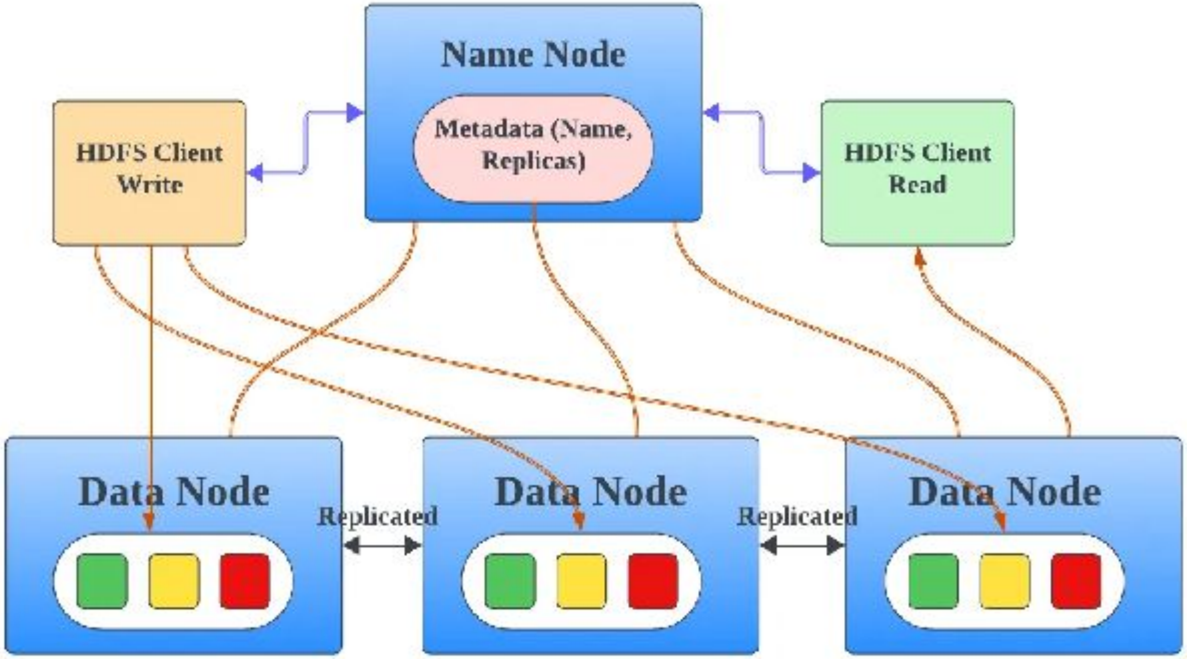
- Hadoop (HDFS, MapReduce)
  - Provides an easy solution for processing of Big Data
  - Brings a paradigm shift in programming distributed system
- Spark
  - Has extended MapReduce for in memory computations
  - for streaming, interactive, iterative and machine learning tasks
- Changing the World
  - Made data processing cheaper and more efficient and scalable
  - Is the foundation of many other tools and software

Backup

MapReduce vs. Spark for Large Scale Data Analysis

- MapReduce and Spark are two very popular open source cluster computing frameworks for large scale data analytics
- These frameworks hide the complexity of task parallelism and fault-tolerance, by exposing a simple programming API to users

Tasks	Word Count	sort	K-means	Page-Rank
MapReduce				
Spark				


Name Node
Metadata (Name, Replicas)
HDFS Client Write
HDFS Client Read
Data Node
Replicated
Data Node
Replicated
Data Node

How HDFS Stores a File