BIG DATA AND BUSINESS ANALYTICS

B. Tech (CSE) - VII SEMESTER

Prepared by

Dr. M Madhu Bala

Professor & Head

Computer Science and Engineering

Institute of Aeronautical Engineering

Course Objectives

Students will try to learn:

I	The scope and essentiality of Big Data and Business Analytics.
II	The technologies used to store, manage, and analyze big data in a Hadoop ecosystem.
III	The techniques and principles in big data analytics with scalability and streaming capability.
IV	The hypothesis on the optimized business decisions in solving complex real-world problems.

Upon the successful completion of this course, students will be able to:

	Course Outcomes	Knowledge Level (Bloom's Taxonomy)
CO 1	Explain the evolution of big data with its characteristics and challenges with traditional business intelligence.	Understand
CO 2	Distinguish big data analysis and analytics in optimizing the business decisions.	Analyse
CO 3	Classify the key issues and applications in intelligent business and scientific computing.	Understand
CO 4	Explain the big data technologies used to process and querying the bigdata in Hadoop, MapReduce, Pig and Hive.	Understand
CO 5	Make use of appropriate components for processing, scheduling and knowledge extraction from large volumes in distributed Hadoop Ecosystem.	Apply
CO 6	Translate the data from traditional file system to HDFS for analyzing big data in Hadoop ecosystem.	Understand
CO 7	Develop a Map Reduce application for optimizing the jobs.	Apply
CO 8	Develop applications for handling huge volume of data using Pig Latin.	Apply
CO 9	Discuss the importance of bigdata framework HIVE and its built-in functions, data types and services like DDL.	Create
CO 10	Define business models and scientific computing paradigms, and tools for big data analytics.	Remember
CO 11	Develop real time big data analytics in various applications like recommender systems, social media applications etc.	Apply

UNIT-I	INTRODUCTION TO BIG DATA

Introduction to Big data: Characteristics of Data, Evolution of Big Data, Definition of Big Data, Challenges with Big Data, Traditional Business Intelligence (BI) versus Big Data.

Big data analytics: Classification of Analytics, Importance and challenges facing big data, Terminologies Used in Big Data Environments, The Big Data Technology Landscape.

UNIT-II	INTRODUCTION TO HADOOP

Introducing Hadoop, RDBMS versus Hadoop, Distributed Computing Challenges, History and overview of Hadoop, Use Case of Hadoop, Hadoop Distributors, Processing Data with Hadoop, Interacting with Hadoop Ecosystem

UNIT-III	THE HADOOP DISTRIBUTED FILE SYSTEM

Hadoop Distributed File System (HDFS):The Design of HDFS, HDFS Concepts, Basic Filesystem Operations, Hadoop Filesystems.

The Java Interface- Reading Data from a Hadoop URL, Reading Data Using the Filesystem API, Writing Data**.** Data Flow- Anatomy of a File Read, Anatomy of a File Write, Limitations.

UNIT -IV	UNDERSTANDING MAP REDUCE FUNDAMENTALS

Map Reduce Framework: Exploring the features of Map Reduce, Working of MapReduce, Exploring Map and Reduce Functions, Techniques to optimize MapReduce jobs, Uses of MapReduce.

Controlling MapReduce Execution with Input Format, Reading Data with custom Record Reader, -Reader, Writer, Combiner, Partitioners, MapReduce Phases, Developing simple MapReduce Application.

UNIT -V	INTRODUCTION TO PIG and HIVE

Introducing Pig: Pig architecture, Benefits, Installing Pig, Properties of Pig, Running Pig, Getting started with Pig Latin, Working with operators in Pig, Working with functions in Pig.

Introducing Hive: Getting started with Hive, Hive Services, Data types in Hive, Built-in functions in Hive, Hive DDL.

Textbooks

1. Seema Acharya, Subhashini Chellappan, —Big Data and Analytics, Wiley Publications, 2nd Edition,2014DT Editorial Services, —Big Data, Dream Tech Press, 2nd Edition, 2015.
2. Tom White, —Hadoop: The Definitive Guide, O'Reilly, 3rd Edition, 2012.
3. Black Book Big Data, dreamtech publications, 1st Edition, 2017

Reference Books

1. Michael Minelli, Michele Chambers, Ambiga Dhiraj, —Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Business, Wiley CIO Series, 1st Edition, 2013.
2. Rajiv Sabherwal, Irma Becerra- Fernandez, —Business Intelligence –Practice, Technologies and Management, John Wiley, 1st Edition, 2011.
3. Arvind Sathi, —Big Data Analytics: Disruptive Technologies for Changing the Game, IBM Corporation, 1st Edition, 2012.

UNIT 1 - INTRODUCTION TO BIG DATA

Topics

- **Introduction to Big data**
- **Characteristics of Data**
- **Evolution of Big Data**
- **Definition of Big Data**
- **Challenges with Big Data**
- **Traditional Business Intelligence (BI) versus Big Data**
- **Big data analytics**
- **Classification of Analytics**
- **Importance and challenges facing big data**
- **Terminologies Used in Big Data Environments**
- **The Big Data Technology Landscape.**

UNIT 1 - INTRODUCTION TO BIG DATA

- Characteristics of Data
  - Composition
  - Condition
  - Context
- Why Big Data
- Characteristics of Big Data
  - Volume
  - Velocity
  - Variety
    - Structured
    - Unstructured
    - Semi structured
    - Batch Data
    - Stream Data
- Challenges with Big Data

What is data?

Data is a raw and unorganized fact

Data are facts or details from which information is derived

Example

45	1
2	50

What is data?

- Data is a raw and unorganized fact
- Data are facts or details from which information is derived
- Example

45	1
2	50

What is information?

- **Information is a processed, organized data presented in a given context and useful.**
- **Example**

1	45
2	50

Characteristics of Data

- **Composition**
  - Deals with structure of data
  - The nature of the data as it is static or real time streaming
- **Condition**
  - Deals with state of the data
  - Can we use this data as input for analysis?
  - Does it require cleansing for further enhancement or enrichment?
- **Context**
  - Where this data has generated?
  - Why this has generated?
  - How sensitive is this data?
  - What are the events associated with this data

Why Big Data

More Data

More Accurate Analysis

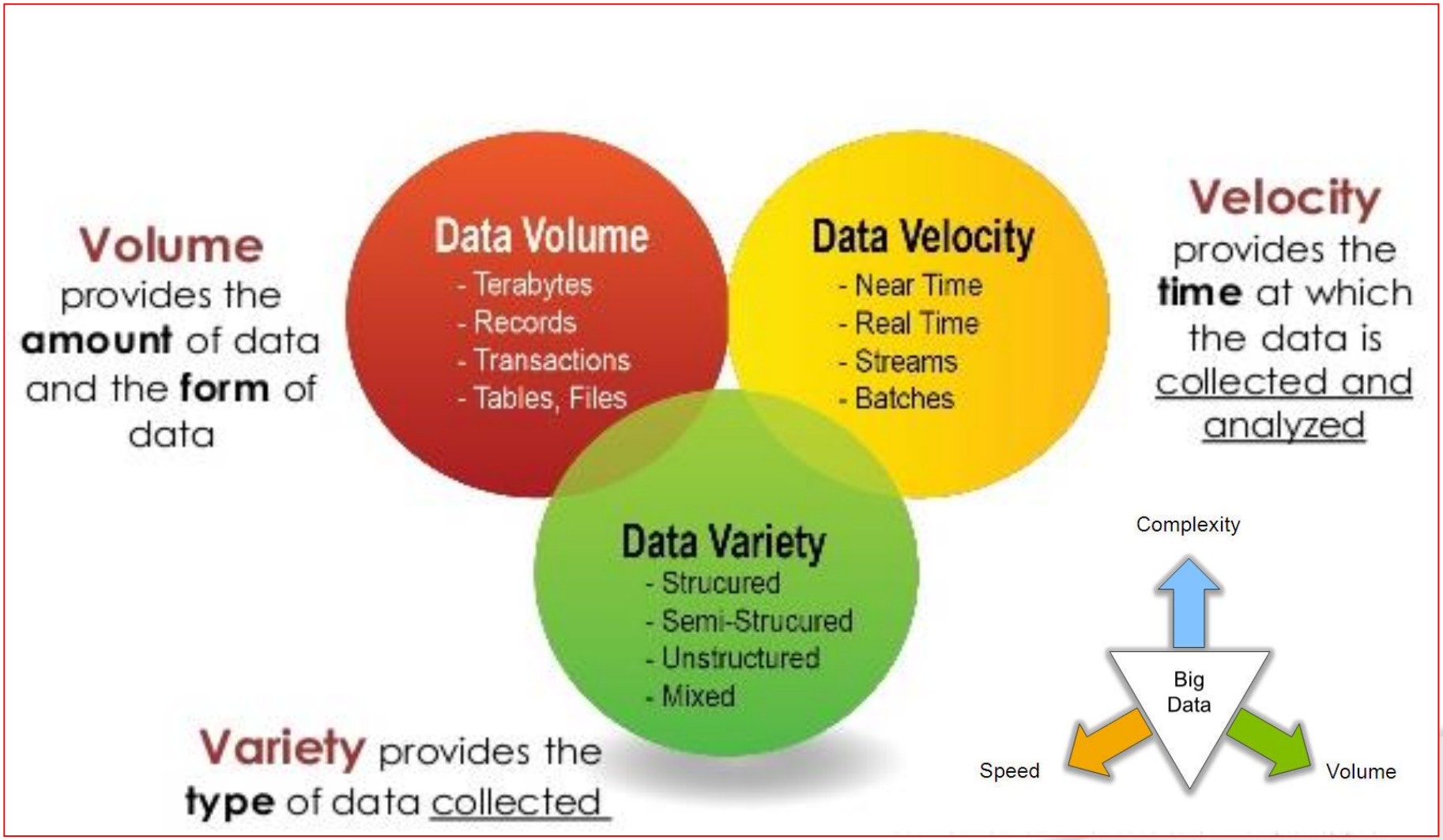
More confidence indecision making

Greater operational efficiencies

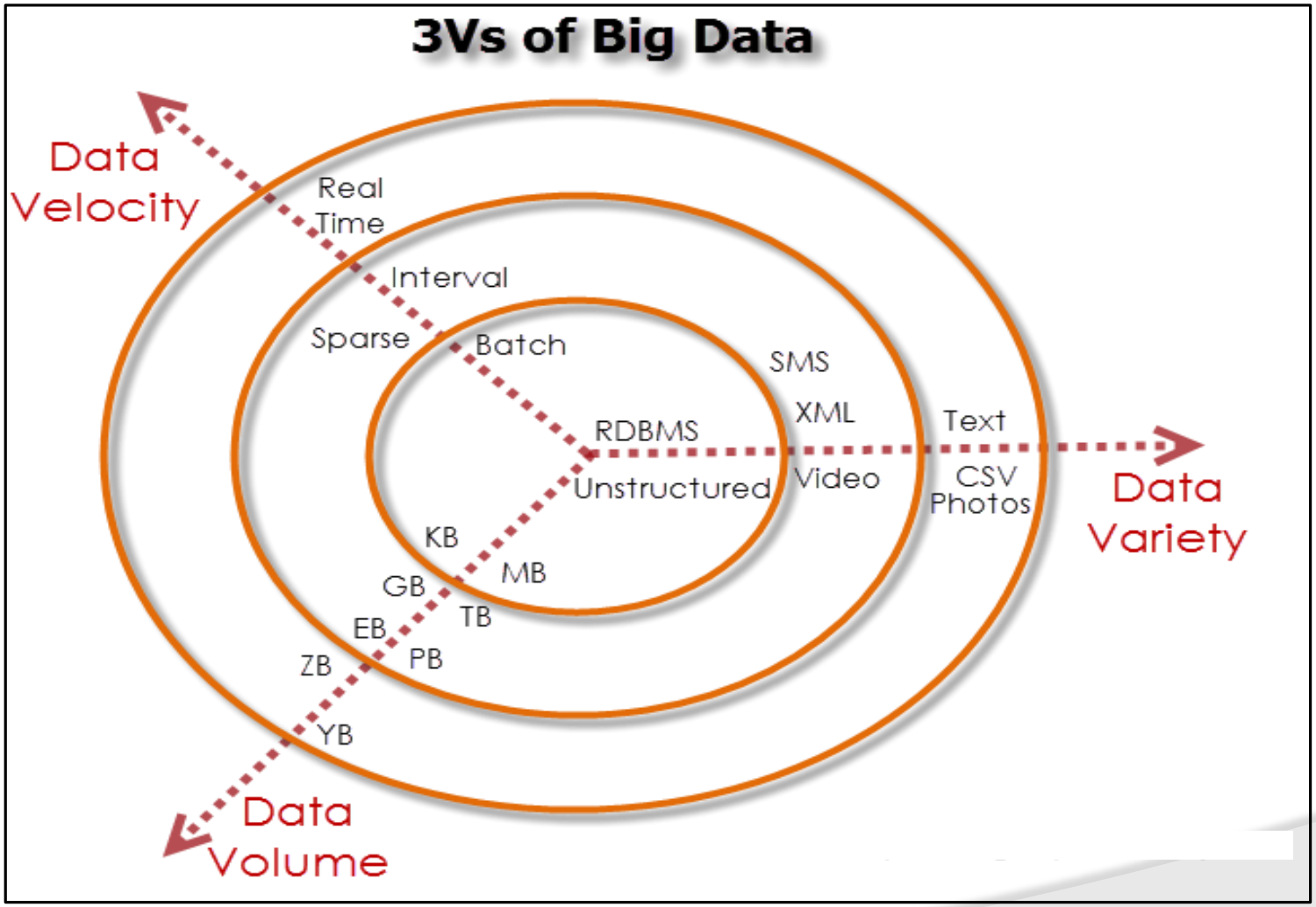
cost reduction, time reduction

new product development, Optimized offerings

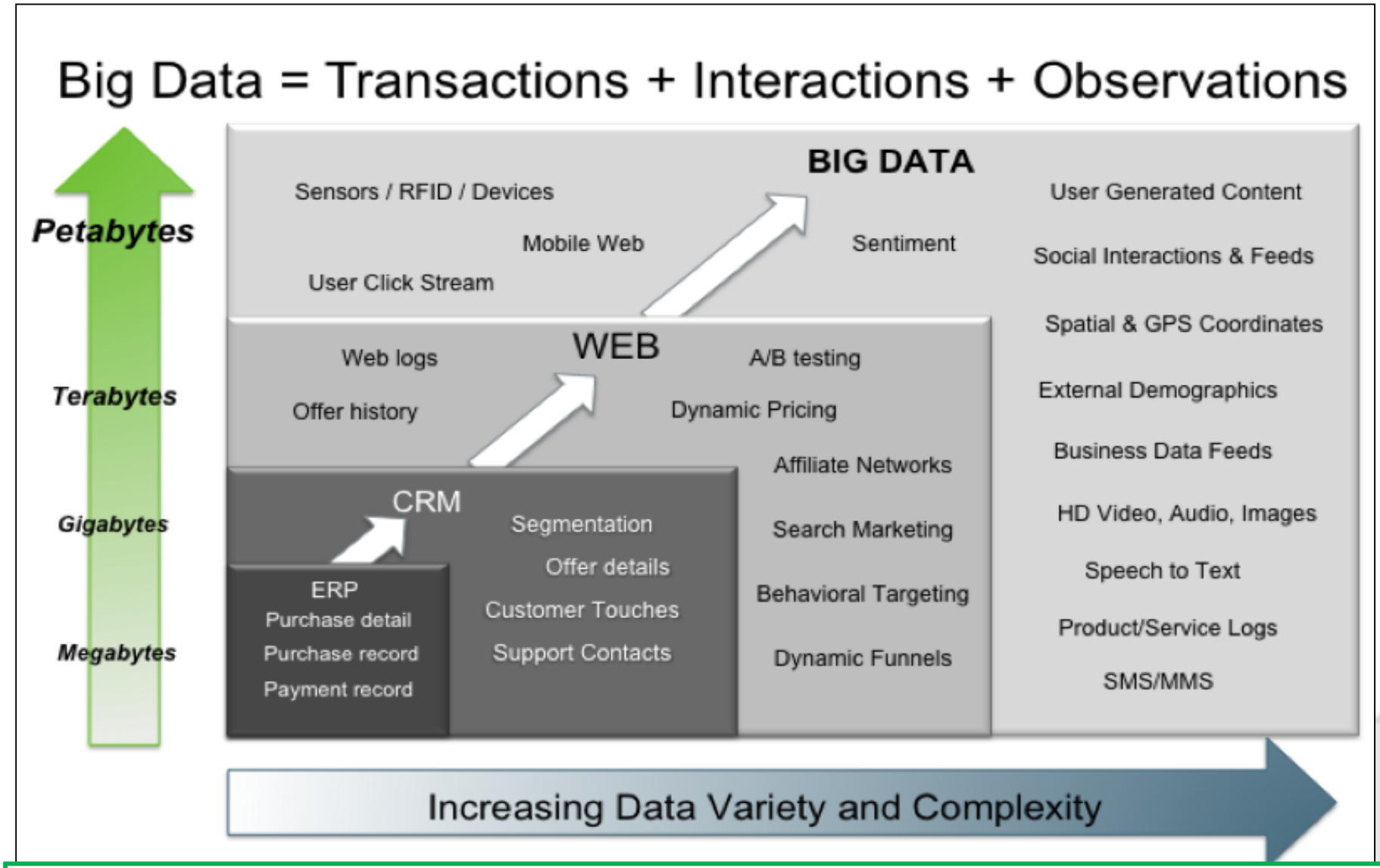
Characteristics :3 V's of Big Data



Characteristics :3 V's of Big Data



Characteristics :3 V's of big data



Velocity (Speed)

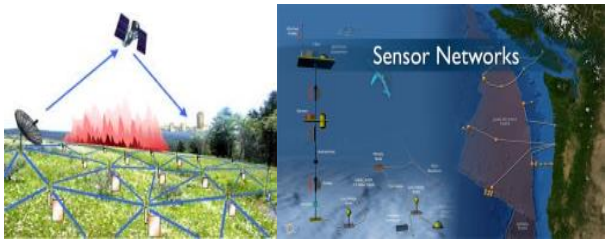
- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like ➔ send promotions right now for store next to you
  - **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction

Real-time/Fast Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)



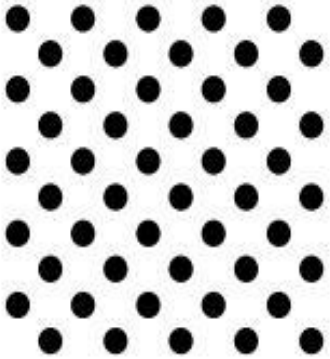
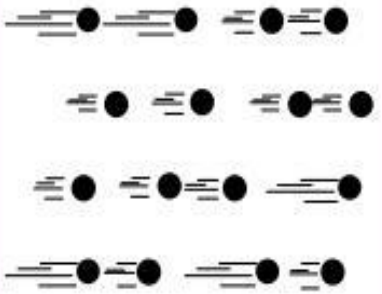
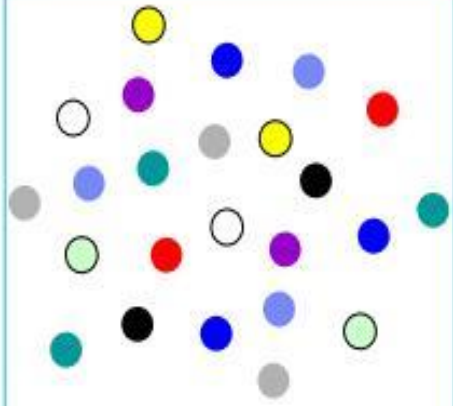
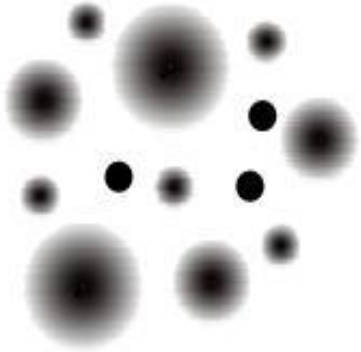
Sensor technology and networks
(measuring all kinds of data)

- **The progress and innovation is no longer hindered by the ability to collect data**
- **But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion**

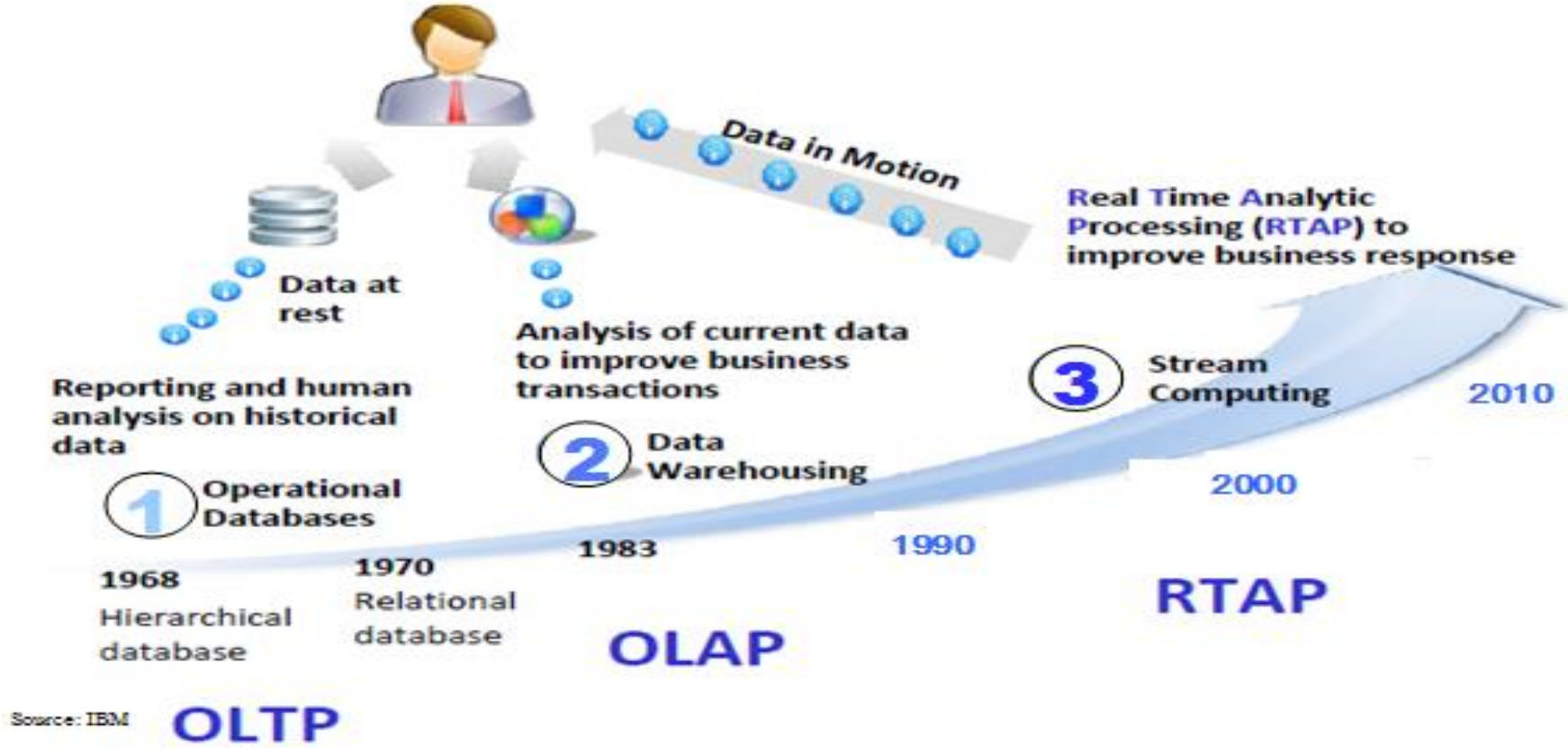
Facts of Big Data - Velocity

- For Every Minute
  - 48 hours of video on Youtube
  - 200 million emails
  - 300,000 tweets
  - 20 million photos view and 300,000 uploads in Flicker
  - 100 terabytes of data uploads in facebook
  - Walmart handles 1 million customer transactions every hour
- Hence increase of speed is required at create, process, store and analysis
- Increase of data in volume in terms of time is called velocity

Some Make it 4V's

Volume	Velocity	Variety	Veracity*
			
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

Harnessing Big Data



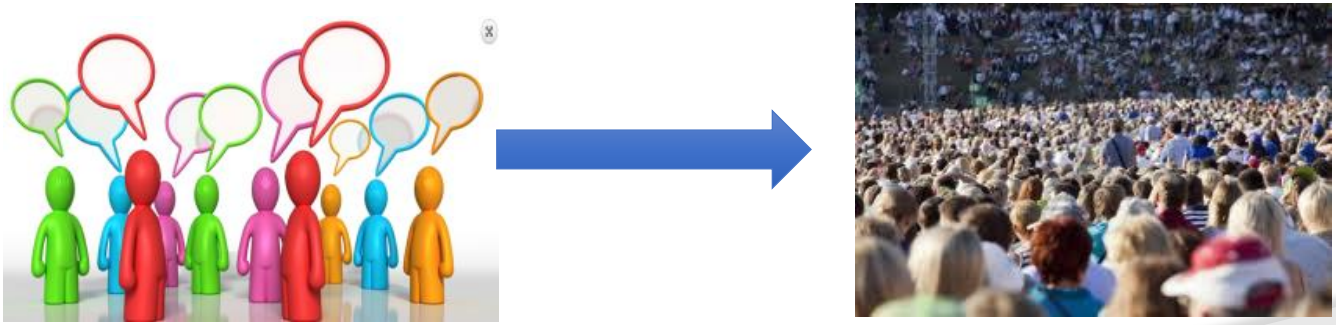
- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

The Model Has Changed…

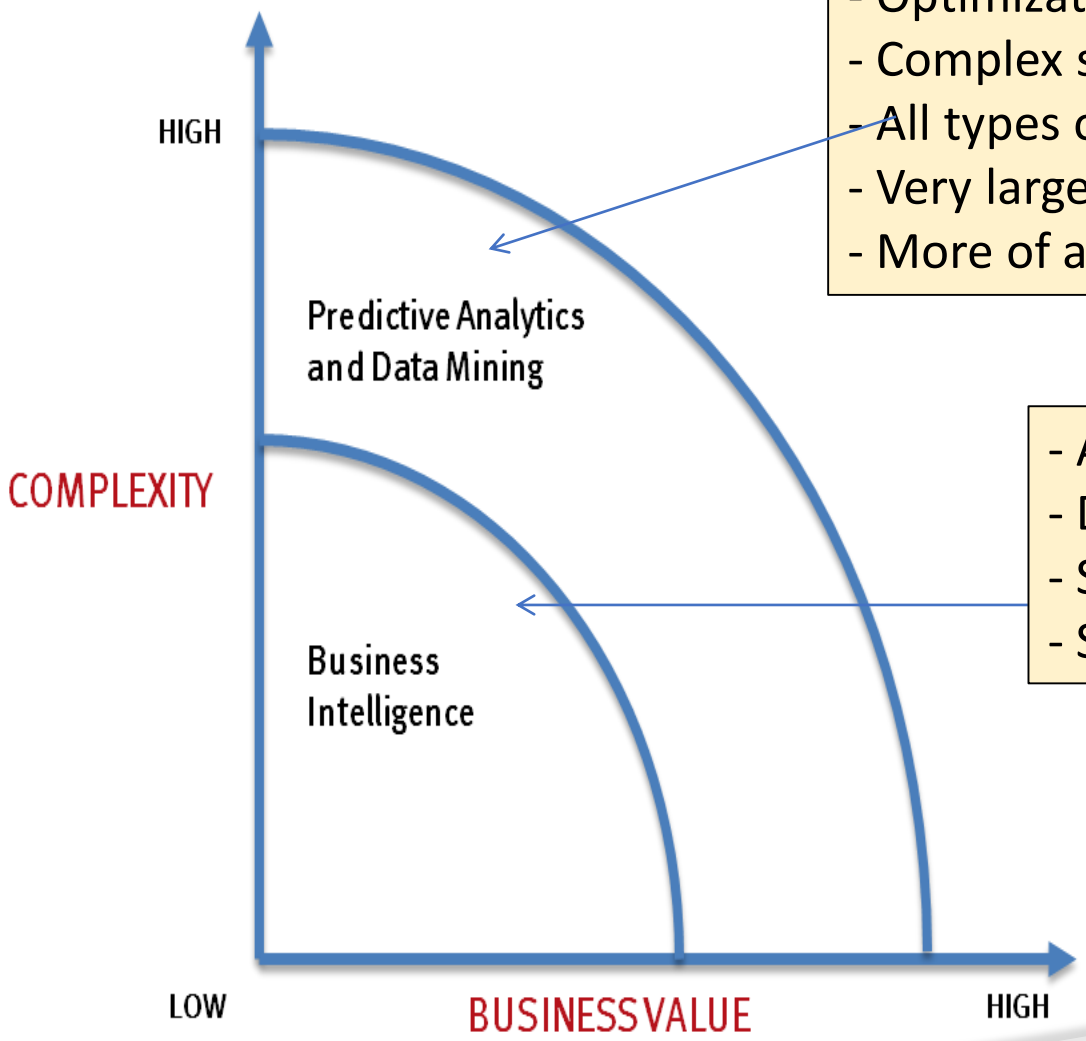
The Model of Generating/Consuming Data has Changed

Old Model: Few companies are **generating data**, all others are **consuming data**

New Model: all of us are generating data, and all of us are consuming data



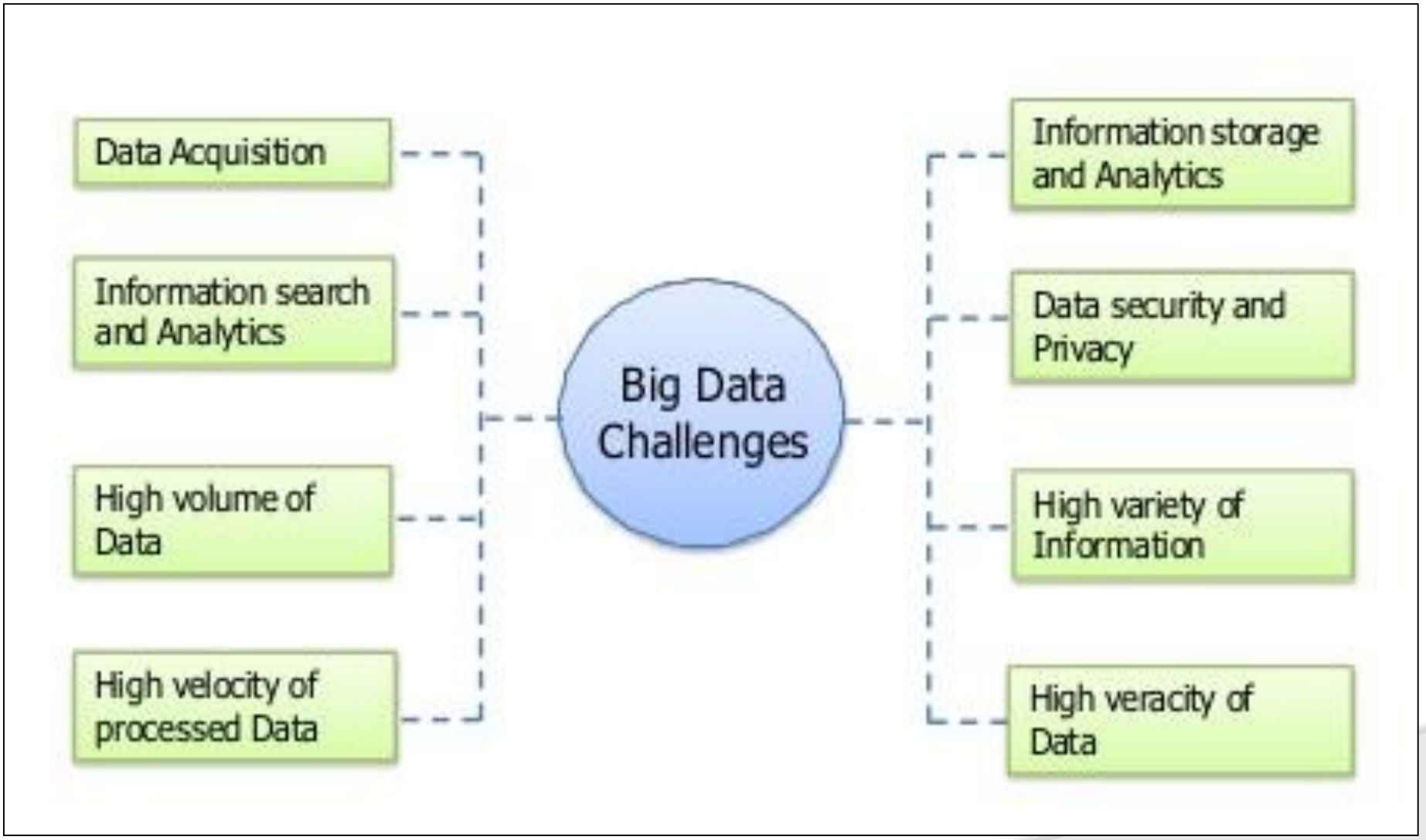
What's driving Big Data



- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

Challenges of big data



Challenges of big data

Problem 1: Data is too big to store on one machine.

HDFS: Store the data on multiple machines!

Problem 2: Very high end machines are too expensive

HDFS: Run on commodity hardware!

Problem 3: Commodity hardware will fail!

HDFS: Software is intelligent enough to handle hardware failure!

Problem 4: What happens to the data if the machine stores the data fails?

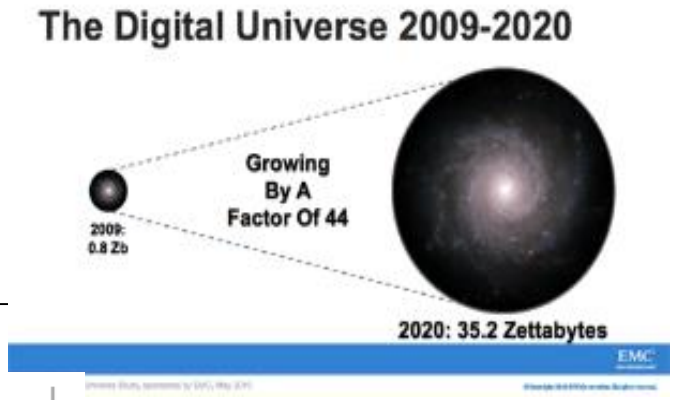
HDFS: Replicate the data!

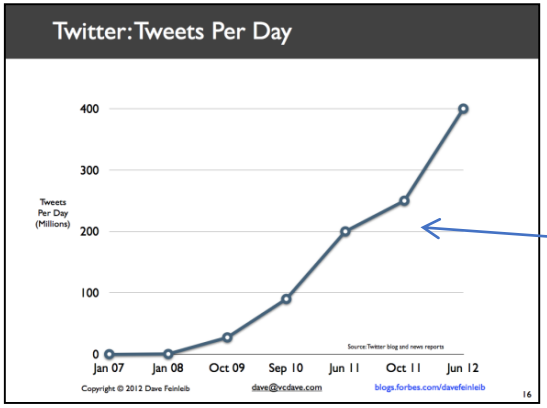
Problem 5: How can distributed machines organize the data in a coordinated way?

HDFS: Master-Slave Architecture!

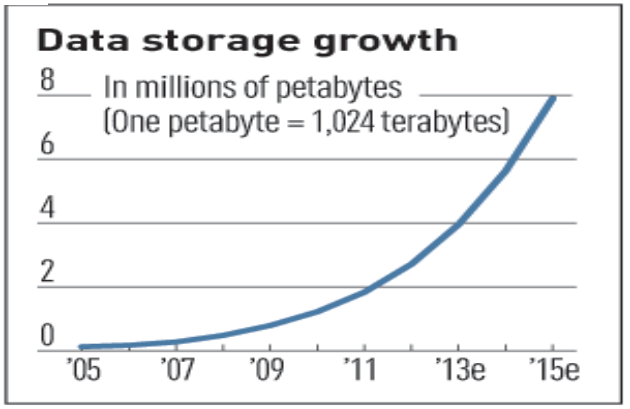
Volume (Scale)

- **Data Volume**
  - 44x increase from 2009-2020
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially



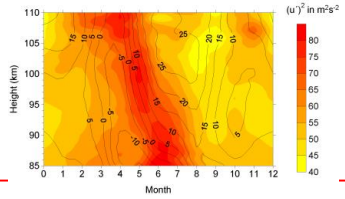


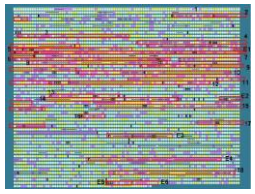
Exponential increase in collected/generated data

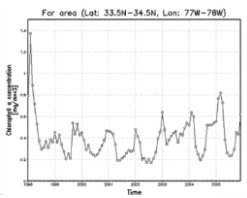


Variety (Complexity)

- Relational Data (Tables/Transaction/Legacy Data)
- Text Data (Web)
- Semi-structured Data (XML)
- Graph Data
  - Social Network, Semantic Web (RDF), …
- Streaming Data
  - You can only scan the data once
- A single application can be generating/collecting many types of data
- Big Public Data (online, weather, finance, etc)







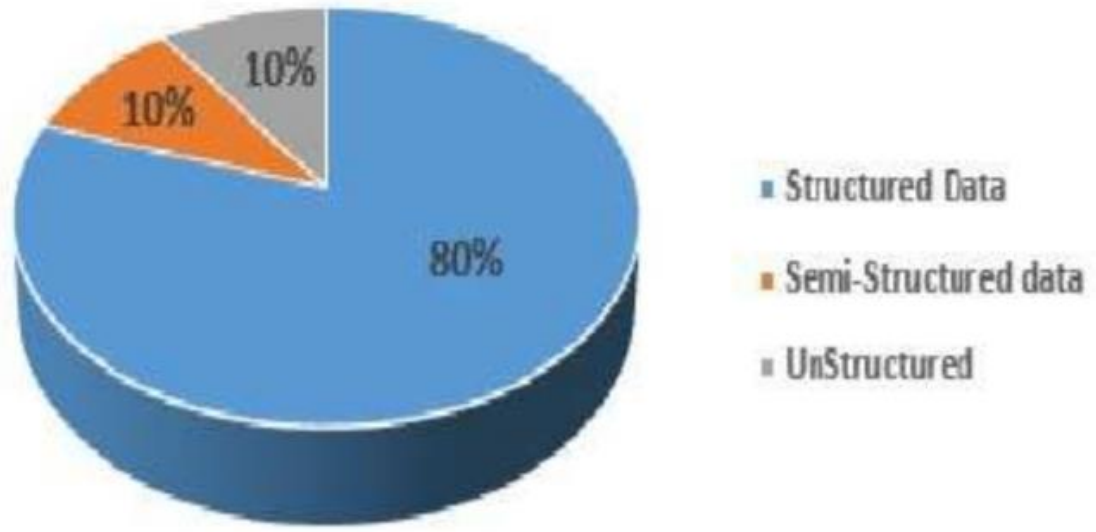
To extract knowledge➔ all these types of data need to linked together

Variety -Digital data

Digital data is classified into the following categories:

- Structured data
- Semi-structured data
- Unstructured data

Distribution of digital data



Variety – Structured data

Structured data is in an organized form (e.g., in rows and columns) and can be easily used by a computer program.

- Relationships exist between entities of data, such as classes and their objects.
- Data stored in databases

Sources of Data:

- Databases such as - Oracle, DB2, Teradata, MySql, PostgreSQL, etc
- Spreadsheets
- OLTP Systems

Ease with structured data: Input / Update / Delete

Security, Indexing , Searching, Scalability, Transaction Processing

Variety – Semi Structured Data

The data does not conform to a data model but has some structure.

- It is not in a form which can be used easily by a computer program.
- Example, emails, XML, markup languages like HTML, etc.
- Metadata for this data is available but is not sufficient.

Characteristics of Semi-structured Data

- Inconsistent Structure
- Self-describing (lable/value pairs)
- Often Schema information is blended with data values
- Data objects may have different attributes not known beforehand

Variety – Semi Structured Data

Sources of Semi-structured Data

- XML (eXtensible Markup Language) and Other Markup Languages
- JSON (Java Script Object Notation) is used to transmit data between a server and a web application.
- JSON is popularized by web services developed utilizing the Representational State Transfer (REST) - an architecture style for creating scalable web services.
- MongoDB (open-source, distributed, NoSQL, documented-oriented database) and
- Couchbase (originally known as Membase, open-source, distributed, NoSQL, document-oriented database) store data natively in JSON format.
-

Variety – Unstructured Data

- The data does not conform to a data model or is not in a form which can be used easily by a computer program.
- About 80–90% data of an organization is in this format.
- Example: memos, chat rooms, PowerPoint presentations, images, videos, letters, researches, white papers, body of an email, etc.

Sources of Unstructured Data

- Audios
- Videos
- Body of Email
- Text
- Messages
- Chats
- Social Media data

Variety – Unstructured Data

Issues with terminology – Unstructured Data

- Structure can be implied despite not being formerly defined.
- Data with some structure may still be labeled unstructured if the structure doesn't help with processing task at hand
- Data may have some structure or may even be highly structured in ways that are unanticipated or unannounced.

Dealing with Unstructured Data

- Data Mining
- Natural Language Processing (NLP)
- Text Analytics
- Noisy Text Analytics
- Image / Video Processing

Batch Data

Batch data is a group of transactions is collected over a period of time. This processing is an efficient way of processing high volumes of **data** .

Data is collected, entered, processed and then the **batch** results are produced

Examples: Real time **data** processing involves a continual input, process and output of **data**. ...

- Radar systems
- customer services and
- bank ATMs are **examples**.

(Hadoop is focused on **batch data** processing).

Streaming Data

- **Streaming data** is data that is continuously generated by different sources.
- Such **data** should be processed incrementally using **Stream** Processing techniques without having access to all of the **data**.
- Examples:

  play video **streaming** services include

- Netflix, iTunes, Hulu, YouTube, Vudu, Amazon Instant, LoveFilm, Baidu, NowTV and Vimeo. Free sources include the Internet Archive, Crackle, Engage Media, Retrovision, Uncle Earl's Classic TV Channel and Shocker Internet Drive In.

Questions ....

- ✓Which category (structured, semi-structured, or unstructured) will you place a **Web Page** in?
- ✓Which category (structured, semi-structured, or unstructured) will you place **Word Document** in?
- ✓State a few examples of **human generated** and **machine-generated** data.

Velocity (Speed)

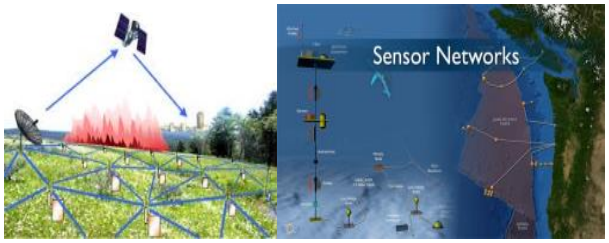
- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like ➔ send promotions right now for store next to you
  - **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction

Real-time/Fast Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)



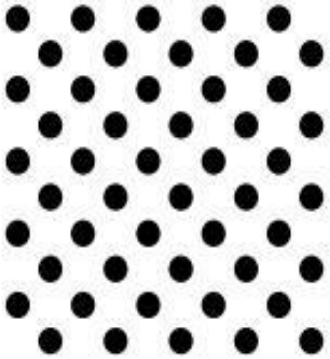
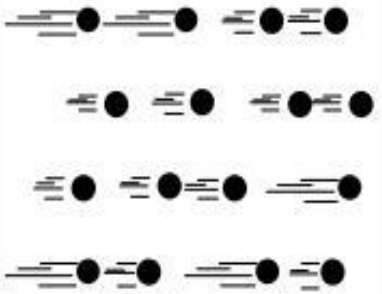
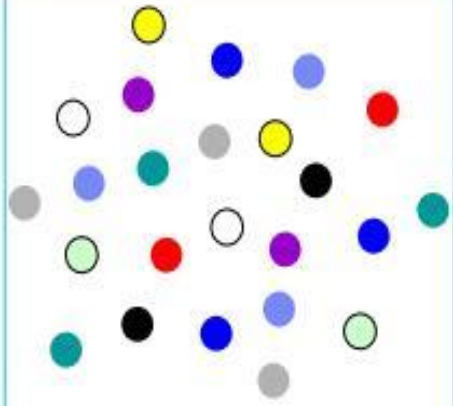
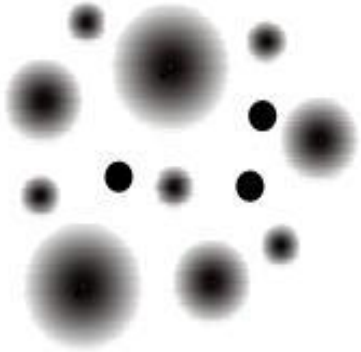
Sensor technology and networks
(measuring all kinds of data)

- **The progress and innovation is no longer hindered by the ability to collect data**
- **But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion**

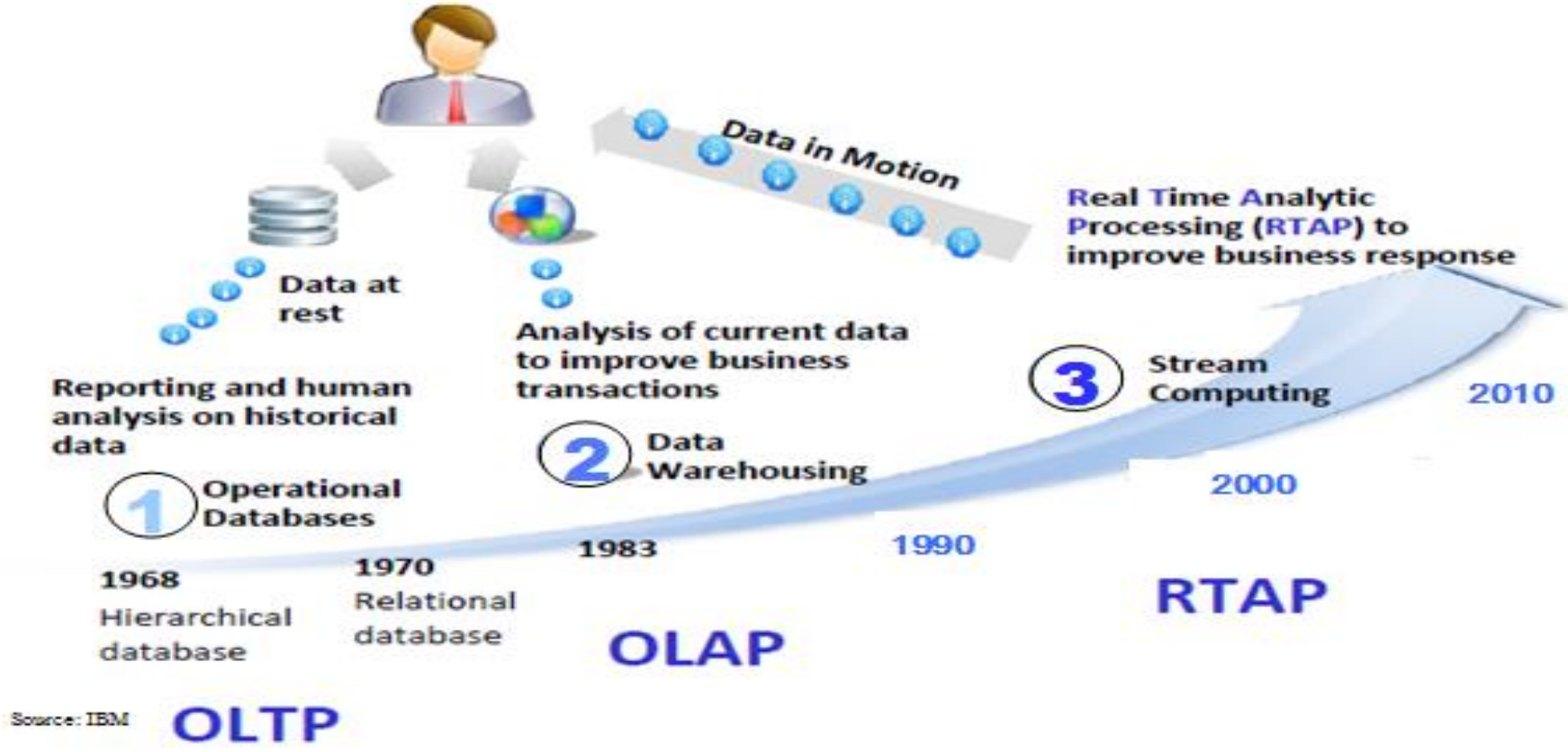
Facts of Big Data - Velocity

- For Every Minute
  - 48 hours of video on Youtube
  - 200 million emails
  - 300,000 tweets
  - 20 million photos view and 300,000 uploads in Flicker
  - 100 terabytes of data uploads in facebook
  - Walmart handles 1 million customer transactions every hour
- Hence increase of speed is required at create, process, store and analysis
- Increase of data in volume in terms of time is called velocity

Some Make it 4V's

Volume	Velocity	Variety	Veracity*
			
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

Harnessing Big Data



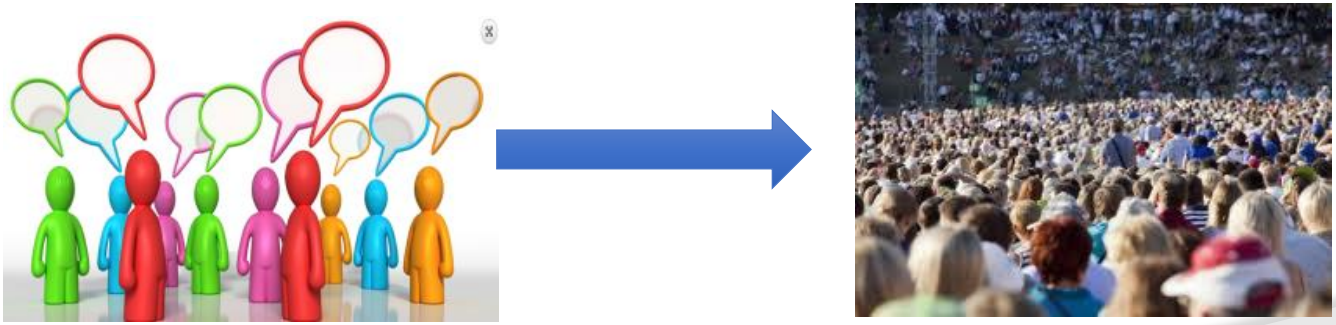
- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

The Model Has Changed…

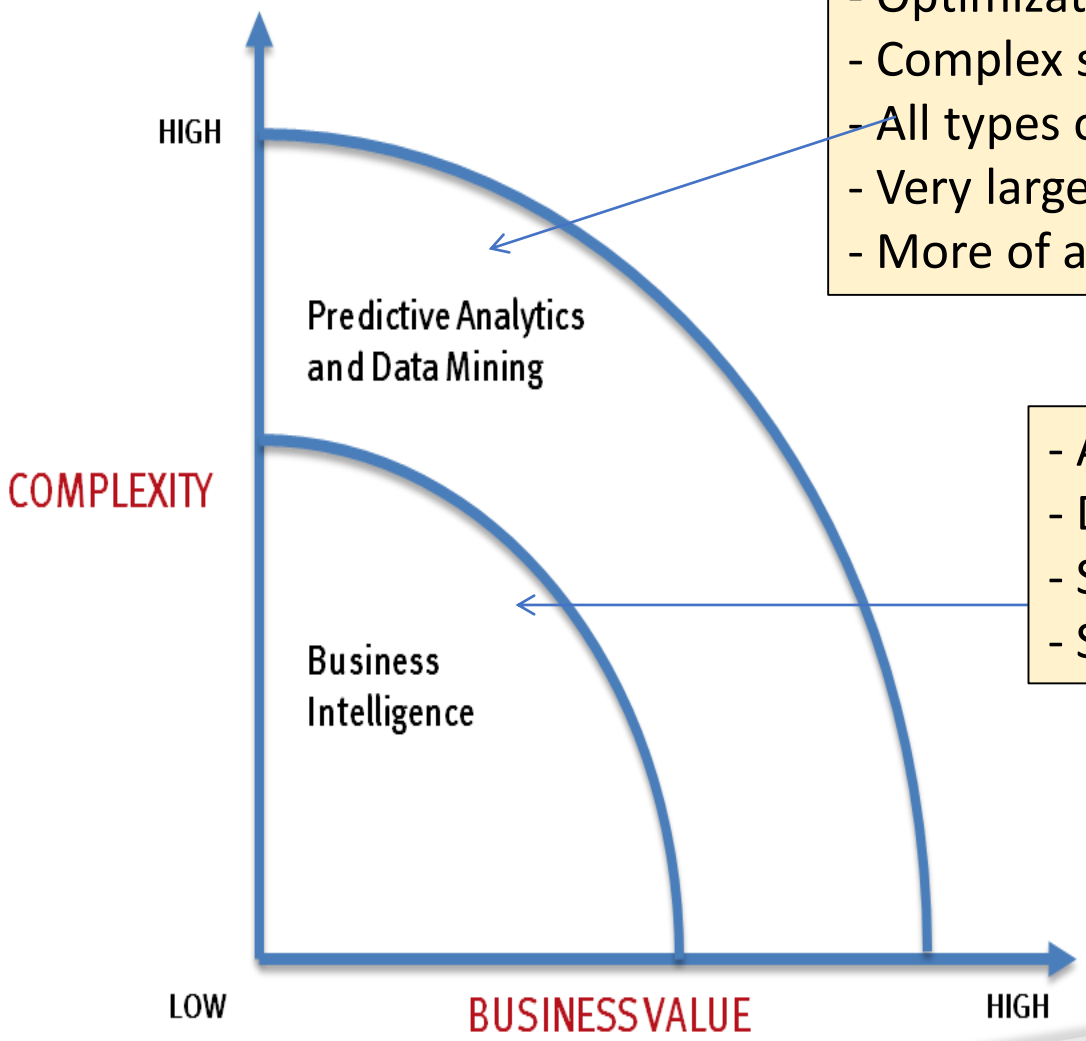
The Model of Generating/Consuming Data has Changed

Old Model: Few companies are **generating data**, all others are **consuming data**

New Model: all of us are generating data, and all of us are consuming data



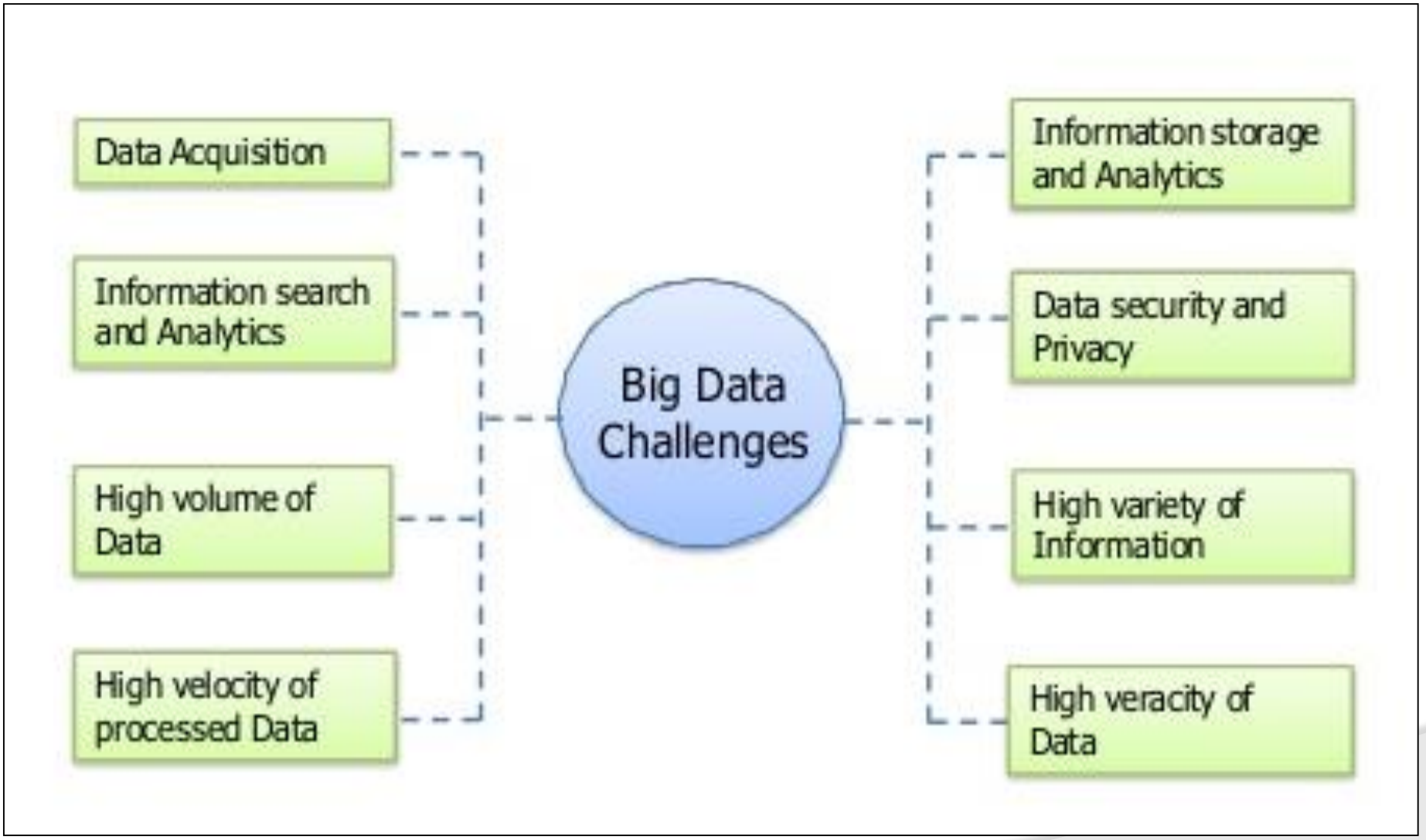
What's driving Big Data



- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

Challenges of big data



Challenges of big data

Problem 1: Data is too big to store on one machine.

HDFS: Store the data on multiple machines!

Problem 2: Very high end machines are too expensive

HDFS: Run on commodity hardware!

Problem 3: Commodity hardware will fail!

HDFS: Software is intelligent enough to handle hardware failure!

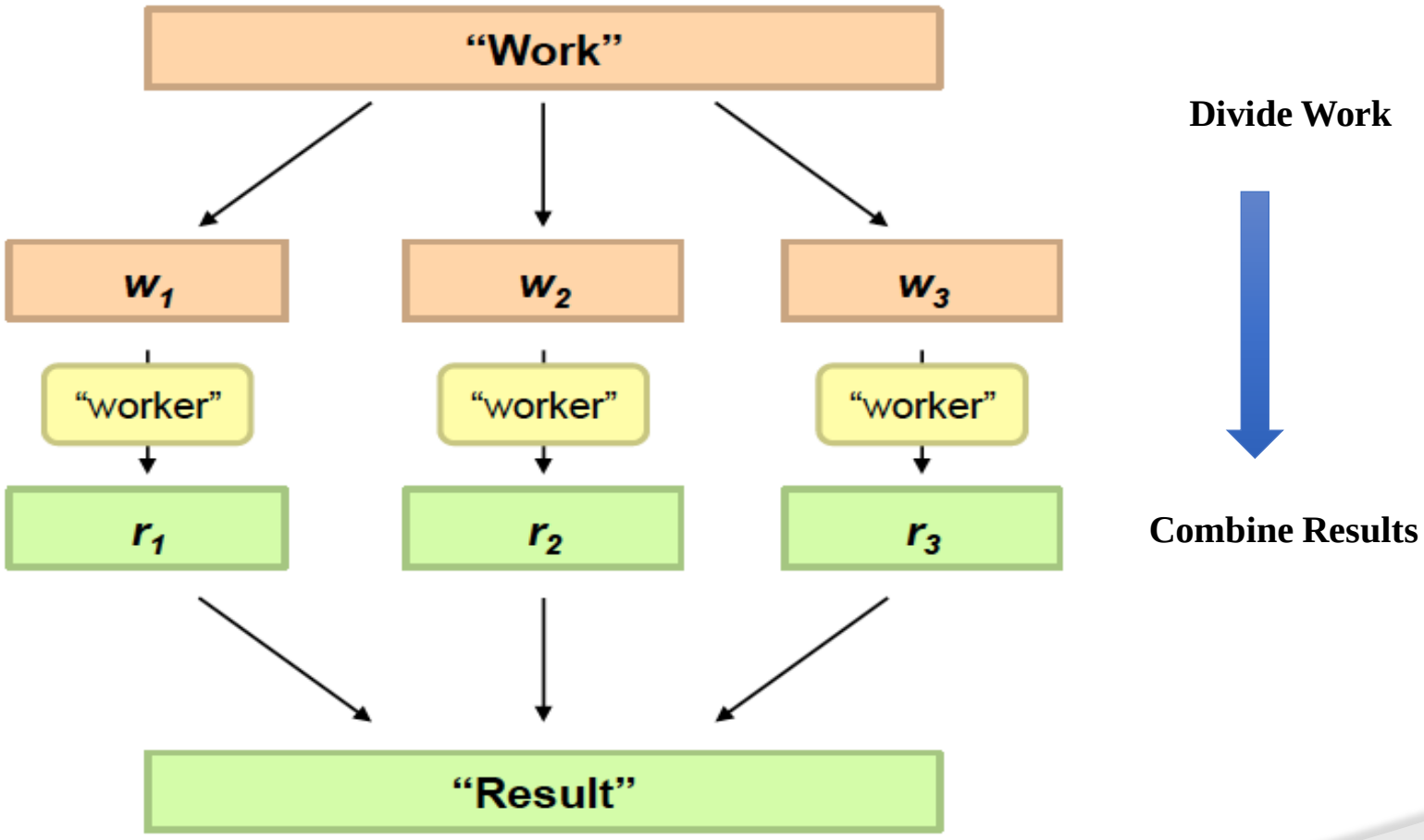
Problem 4: What happens to the data if the machine stores the data fails?

HDFS: Replicate the data!

Problem 5: How can distributed machines organize the data in a coordinated way?

HDFS: Master-Slave Architecture!

Divide and Conquer philosophy



Challenges of big data

Distributed processing is non-trivial

How to assign tasks to different workers in an efficient way?

What happens if tasks fail?

How do workers exchange results?

How to synchronize distributed tasks allocated to different workers?

Big data storage is challenging

✓Data Volumes are massive

✓Reliability of Storing PBs of data is challenging

✓All kinds of failures: Disk/Hardware/Network Failures

✓Probability of failures simply increase with the number of machines

…

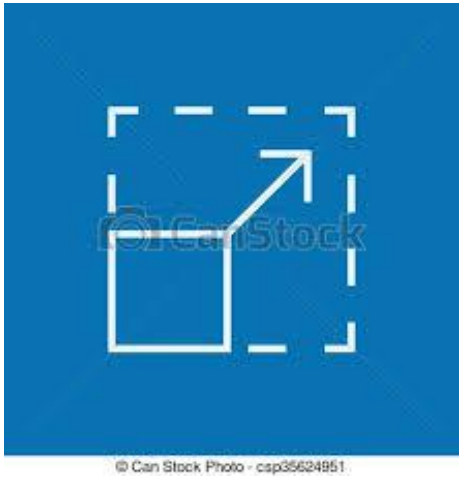
One popular solution: Hadoop

Hadoop Cluster at Yahoo! (Credit: Yahoo)

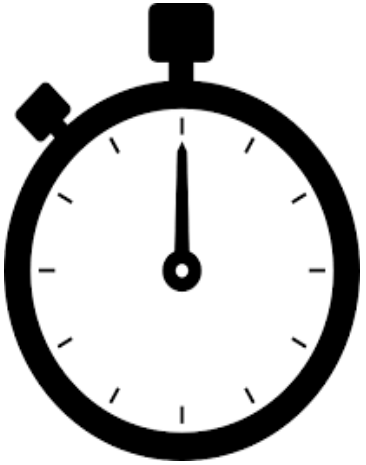
Draw backs of traditional system

Draw backs

Expensive

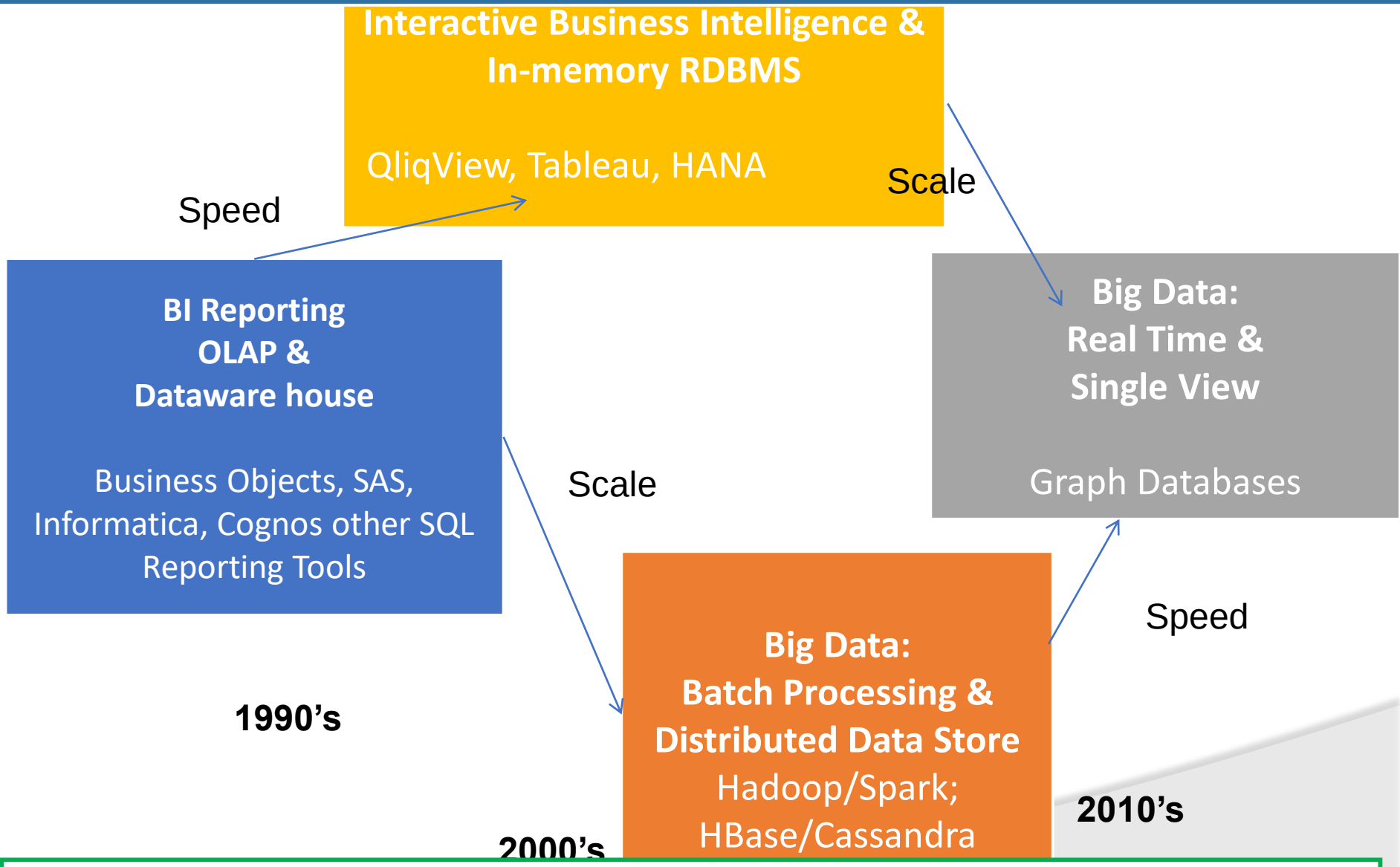


Scalability



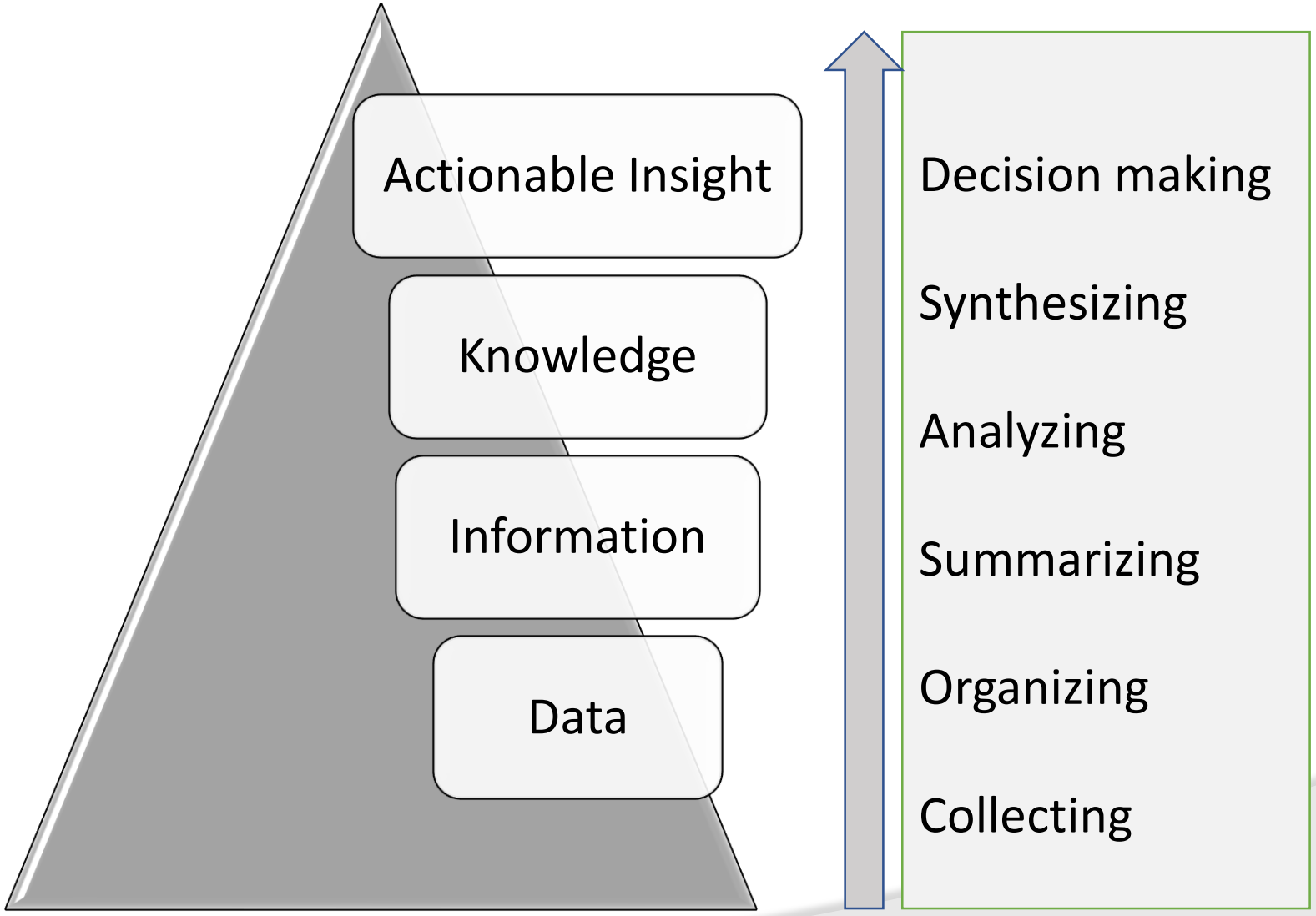
Time consuming

The Evolution of Business Intelligence



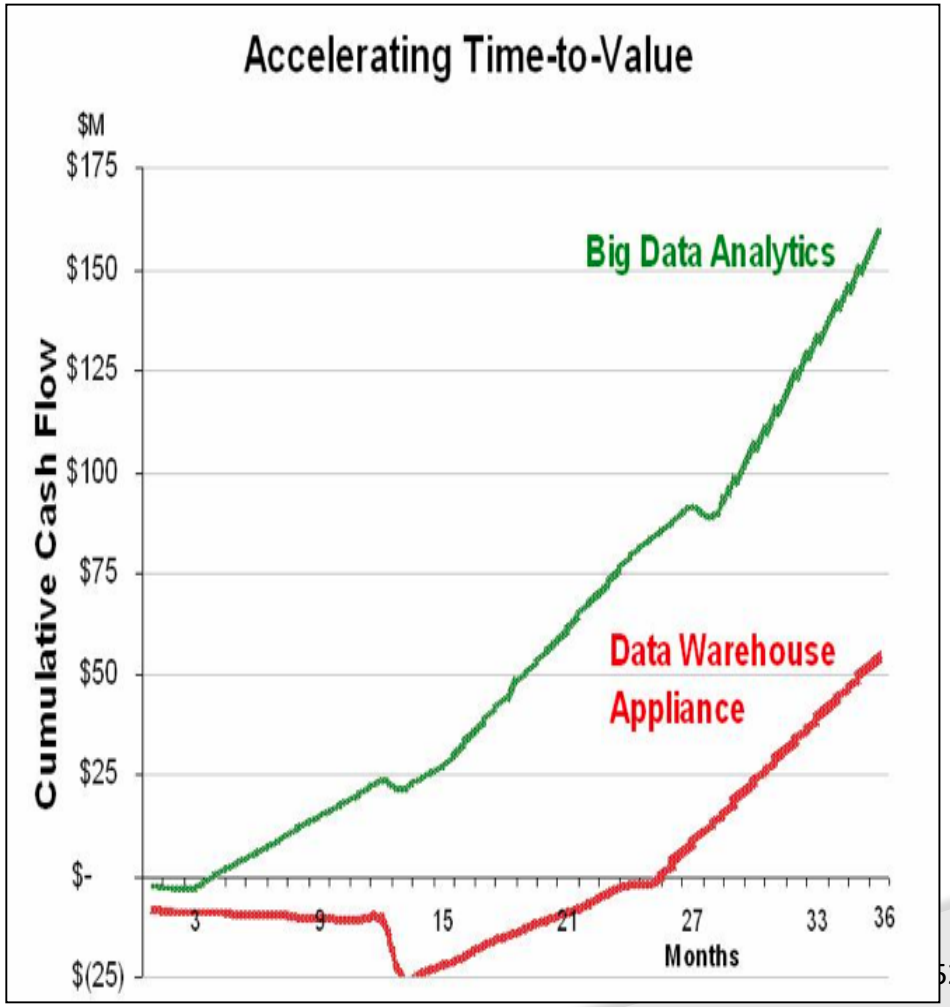
- Transformation of Data
- What is Big Data Analytics
- Classification of Analytics
  - Thought 1 classification
  - Thought 2 classification
    - Analytics 1.0
    - Analytics 2.0
    - Analytics 3.0
- Data Analysis Vs Data Analytics
- Why is Big Data Analytics Important?
- Technologies to meet big data challenges
- Terminologies Used in Big Data Environments
- The Big Data Technology Landscape

Transformation of Data to yield actionable insights

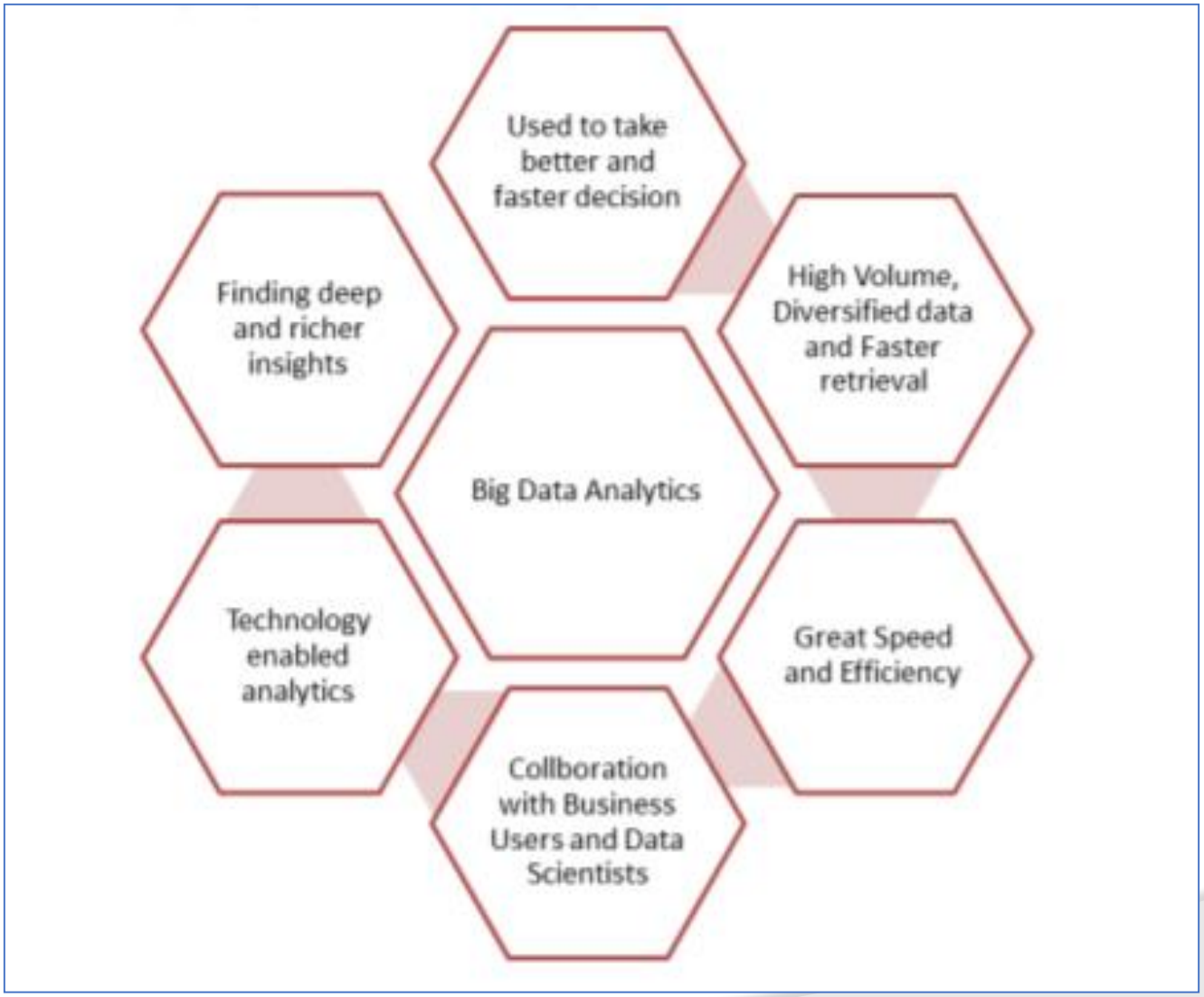


Traditional Business Intelligence (BI) versus Big Data

- Big data is more **real-time** in nature than **traditional DW** applications
- Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data applications
- Shared nothing, massively parallel processing, scale out architectures are well-suited for big data applications



What is Big Data Analytics



Classification of Analytics

Thought 1:

Basic Analytics

- Slicing and dicing of data to help basic insights of data
- Reporting on historical data and visualizations

Operational Analytics - Enterprise business process

Advanced Analytics

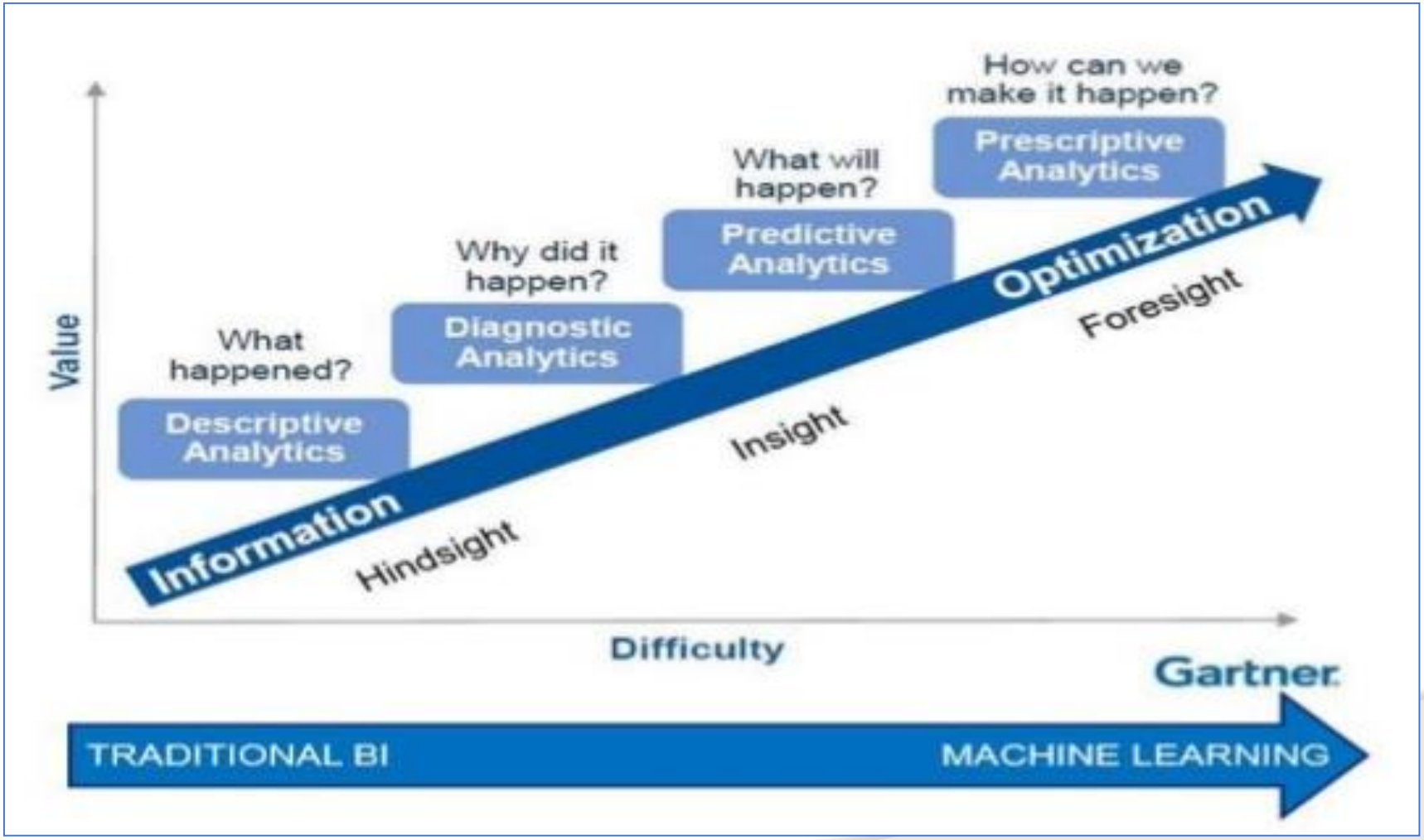
- Forecasting for future by predictive and prescriptive modelling

Monetarized Analytics

- used to derive direct business revenue.

Classification of Analytics

Thought 2: classified as Analytics 1.0, Analytics 2.0, Analytics 3.0



Analytics 1.0	Analytics 2.0	Analytics 3.0
1950-2009 Descriptive Analytics Report on events, occurrences etc. of the past Key questions: What happened? Why did happened? Data From Legacy systems, ERP, CRP and third party applications Small and structured data sources. Data stored in enterprise data warehouses or data marts Data Internally sourced Relational Databases	2005-2012 Descriptive + Predictive Analytics Use data from past to make predictions for future Key questions: What will happen? Why will it happen? Big data Big volume, variety and veracity of data being stored and processed. Usually on distributed servers on Hadoop. Externally sourced Database Applications, Hadoop clusters, SQL to Hadoop environment..	2012-present Descriptive + Predictive+ Prescriptive Analytics Use data from past to make predictions for future and make recommendations to leverage the situation to ones advantage Key questions: What will happen? When will it happen? What should be the action taken to take advantage of what will happen? Data From Big data, Legacy systems, ERP, CRP and third party applications Data is beiA blend of big data and traditional analytics offering with speed and impact Internally and externally sourced In memory analytics, Database processing, agile analytical

Data Analysis Vs. Analytics

- **Data analysis** refers to the process of examining, transforming and arranging a given data set in specific ways in order to study its individual parts and extract useful information.
- **Analysis** looks backwards over time, providing marketers with a historical view of what has happened.
- **Data analytics** is an overarching science or discipline that encompasses the complete management of data.
- **Analytics** look forward to model the future **or** predict a result.
- **It** involves statistical tools & techniques with business insight to bring out the hidden patterns, stories from the data.
- Data **analytics** is a broader term and includes data **analysis** as necessary subcomponent.

Why is Big Data Analytics Important?

REACTIVE	PROACTIVE
Business Intelligence It provides standard business reports, ad hoc reports, OLAP and even alerts and notifications based on analytics	**Big Analytics** Making forward-looking, proactive decisions requires proactive big analytics like optimization, predictive modeling, text mining, forecasting and statistical analysis.
Big Data BI Business intelligence software can give users greater insight into manufacturing costs and the ability to adjust production	**Big Data Analytics** Extract only the relevant information from terabytes, petabytes and exabytes, and analyze it to transform your business decisions for the future.

UNIT 1 - INTRODUCTION TO BIG DATA

- Terminologies Used in Big Data Environments
  - In-Memory Analytics
  - In-Database Analytics
  - Symmetric Multiprocessor System (SMP)
  - Massively Parallel Processing (MPP)
  - Difference between Parallel and Distributed Systems
  - Shared Nothing Architecture
  - CAP Theorem
- The Big Data Technology Landscape
  - NoSQL
  - Difference between SQL and NoSQL

Terminologies used in Big Data

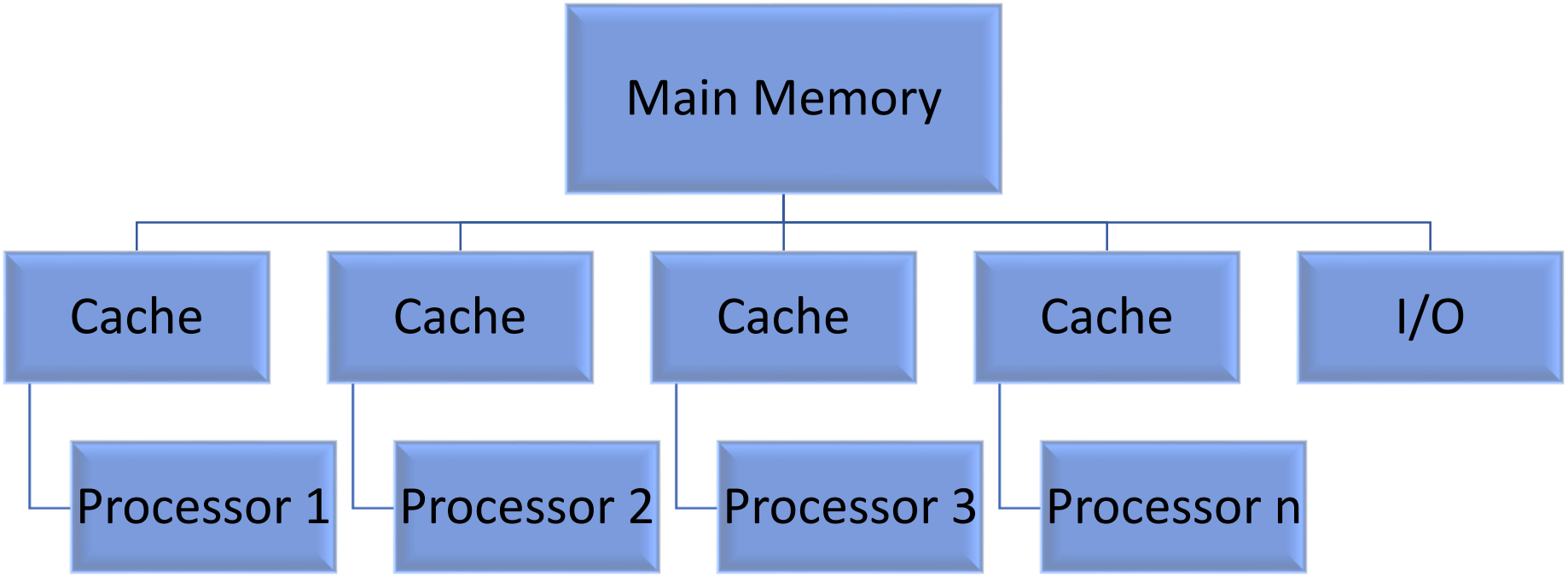
- **In memory Analytics**
  - Drawback with data access from non volatile data storage such as hard disk leads to slow process
  - This is addressed through in memory analytics.
  - In this all relevant data need to load in RAM or primary memory
- **In database Analytics**
  - Also called In database Processing
  - Works with fusing Datawarehouse with analytical system
  - All processed data stored in enterprise data warehouse or data mart

Terminologies used in Big Data

Symmetric multi processor System(SMP)

- Single common main memory that is shared by two or more identical processors
- The processors have full access to all I/O devices and are controlled by a single operating system instance.
- Each processor has own high memory as cache and connected using system bus
- Also called tightly coupled multi processor systems

Symmetric Multiprocessor System (SMP) Architecture



Terminologies used in Big Data

Massively parallel processing (MPP)

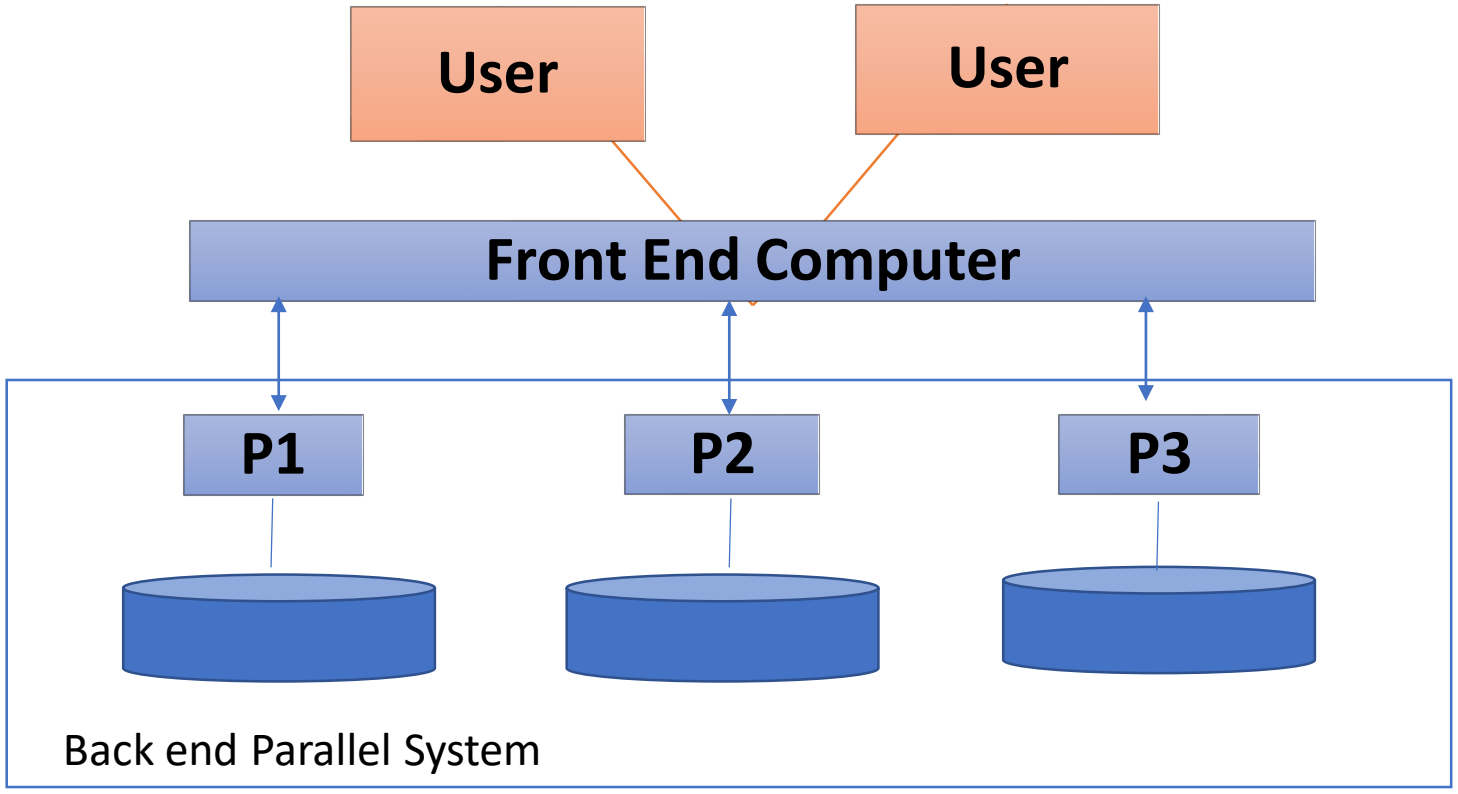
- Refers to coordinated processing of programs by no. of processors working parallel
- The processors, cache have their own operating systems and dedicated memory.
- They work on different parts of the same program.
- The MPP processors communicate using some sort of interface.
- The MPP systems are more difficult to program as the application must be divided in such a way that all the executing segments can communicate with each other.
- MPP is different from Symmetrically Multiprocessing (SMP) in that SMP works with the processors sharing the same operating system and same memory SMP is also referred to as tightly-coupled multiprocessing.

Differences between Parallel and Distributed Systems

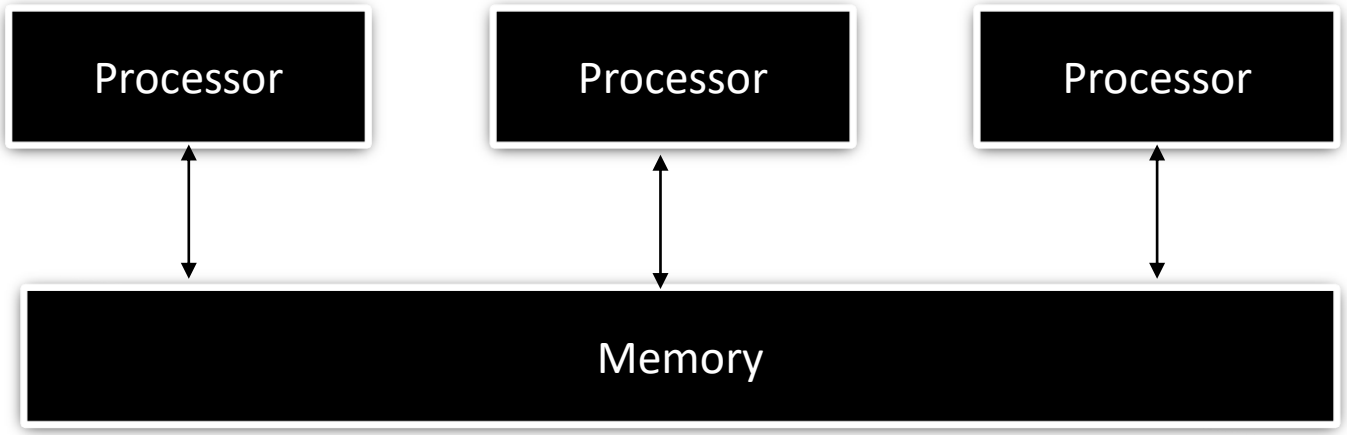
- **Parallel database system** is a tightly coupled system.
- The processors cooperate for query processing.
- The user is unaware of the parallelism since he/she has no access to a specific processor of the system.
- Either the processors have access to a common memory or make use of message passing for communication.

- **Distributed database systems** are loosely coupled and are composed by individual machines.
- Each of the machines can run their individual application and serve their own respective user.
- The data is usually distributed across several machines, thereby a number of machines to be accessed to answer a user query.

Parallel Systems Architecture



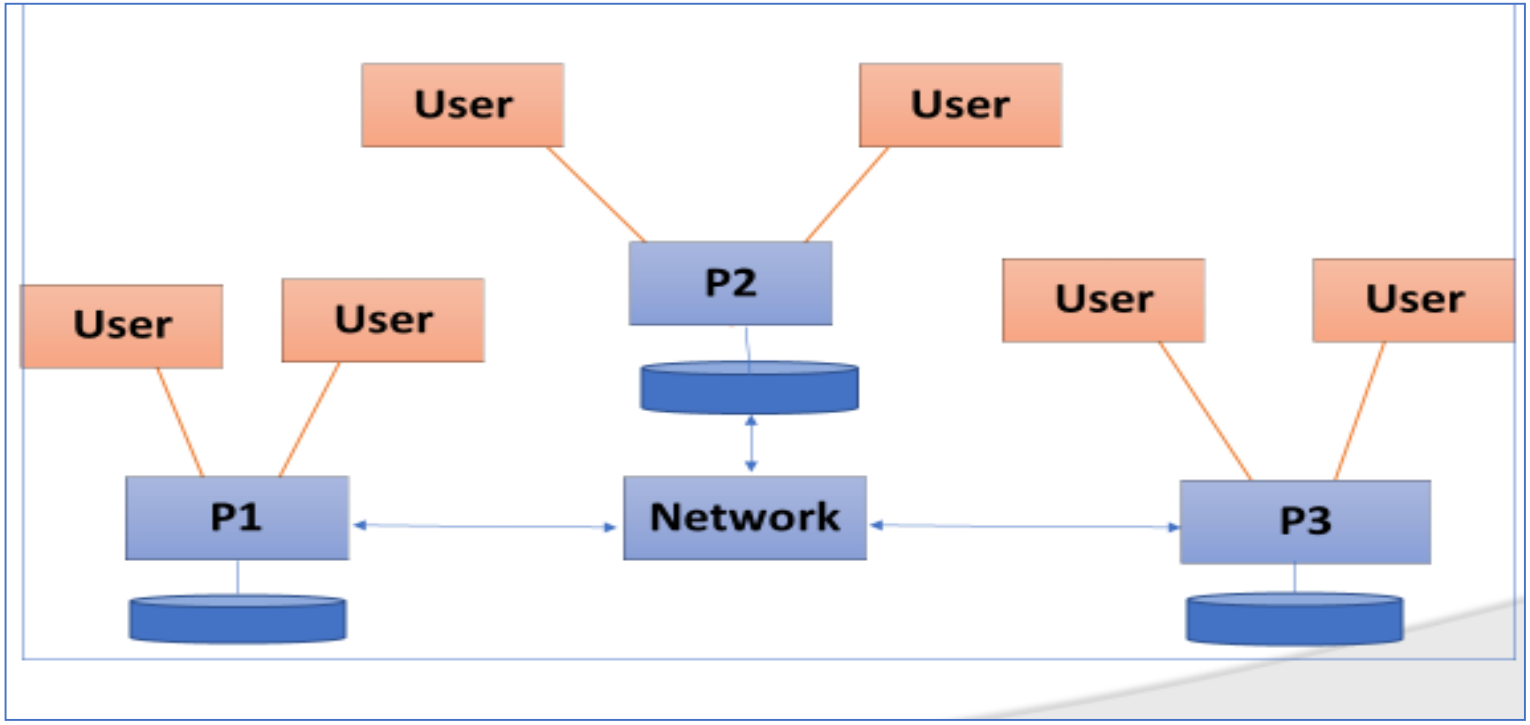
Parallel Systems Architecture



Terminologies used in Big Data

Distributed Systems

- Each machine can run their own application
- The data is distributed among several machines.
- To answer a query no. of machines to be accessed



Terminologies used in Big Data

Shared Nothing Architecture : Three most common types of architecture for multiprocessor high transaction rate systems.

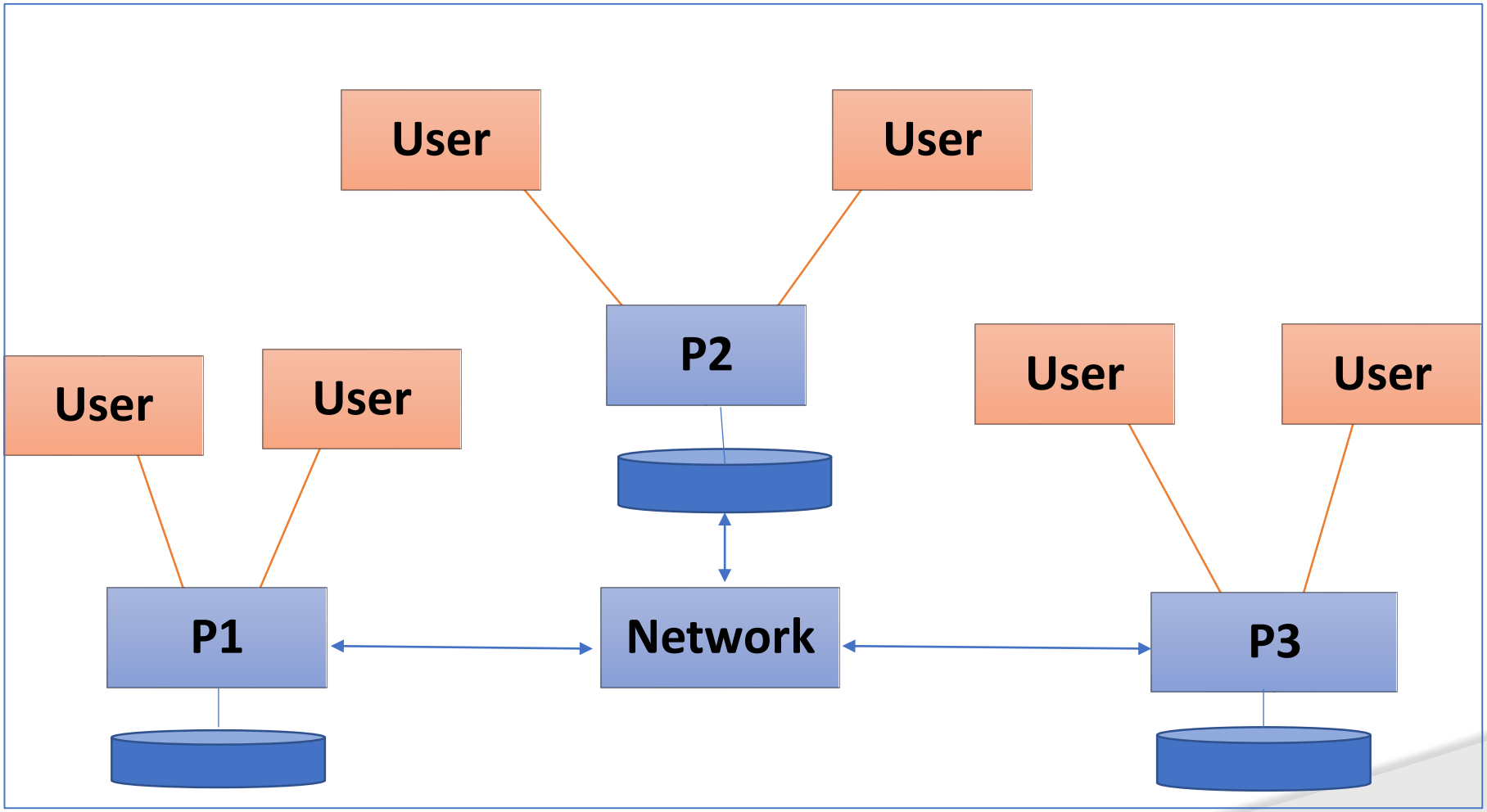
- Share memory (SM) – Central memory
- Shared Disk (SD) – common collection of disks
- Shared Nothing (SN) – neither disk nor memory

- **Shared memory architecture**, a common central memory is shared by multiple processors.
- **Shared disk architecture**, multiple processors share a common collection of disks while having their own private memory.
- **Shared nothing architecture**, neither memory nor disk is shared among multiple processors.

Advantages : Fault Isolation – **A fault in single node in confined to that node exclusively and exposed through messages**

Scalability - imposes a limit on how many nodes can be added to the distributed shared disk system, thus compromising on scalability

Distributed System



Terminologies used in Big Data

CAP Theorem

Also called Brewers's Theorem

It states that a distributed computing environment, it is impossible to provide following guarantee

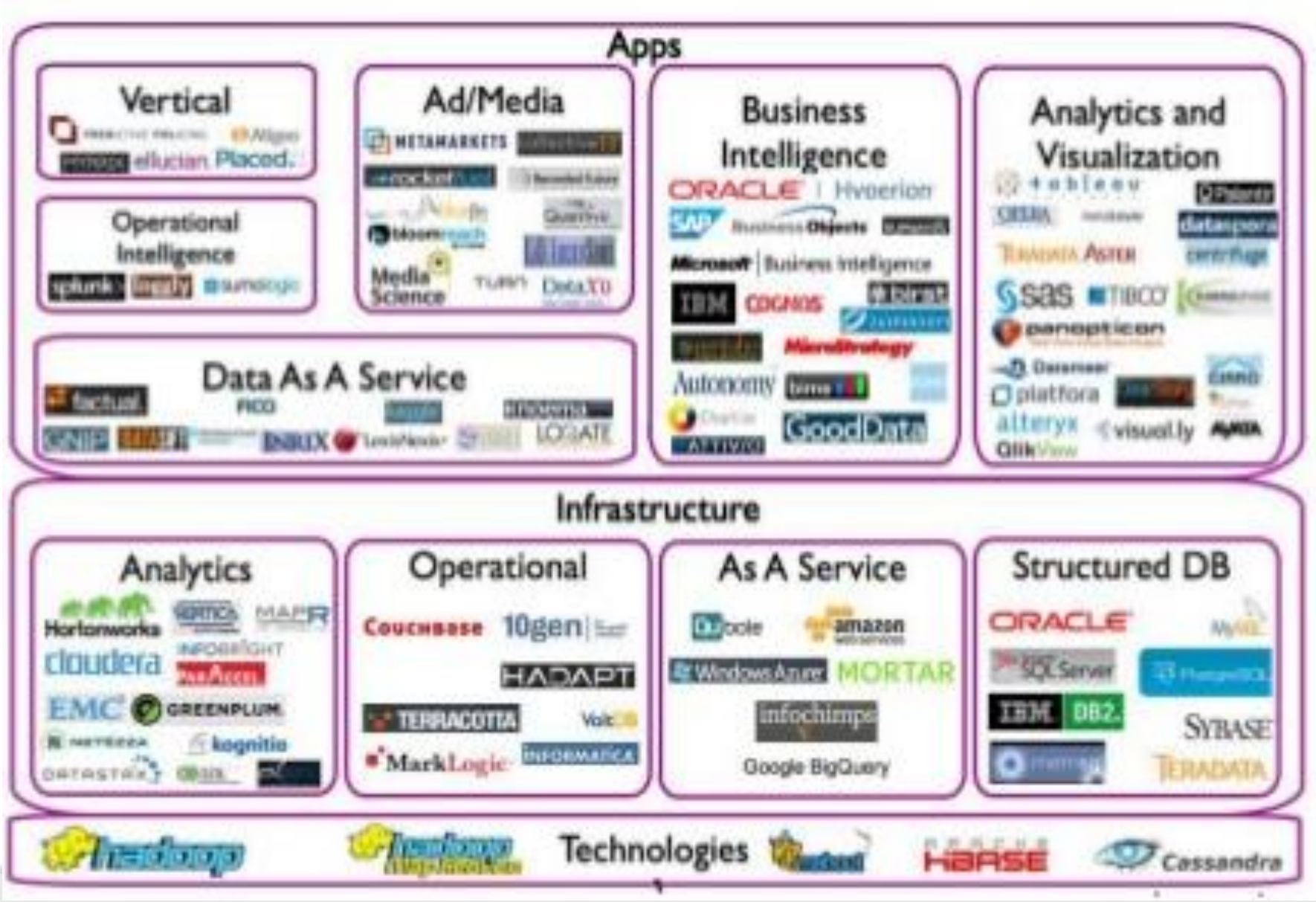
- Consistency – Every read fetches the last write
- Availability – Reads and Writes always succeed. Each non failing node will return a response in a reasonable amount of time.
- Partition Tolerance – the system will continue to function when network partition occurs

Top Data Analytics Tools

- MS Excel
- SAS
- IBM SPSS
- Statistica
- R Analytics
- Weka

- The Big Data Technology Landscape
  - NoSQL
  - Difference between SQL and NoSQL
- Hadoop Introduction

Big data technology Landscape



Big data technology Landscape

Majorly two important technologies

1. NoSQL
2. Hadoop

NoSQL (Not Only SQL):

- First coined by Carlo Strozzi in 1998 to name his lightweight, open-source, relational database that did not expose the standard SQL interface.
- Johan Oskarsson, who was then a developer at last. fm, in 2009 reintroduced the term NoSQL at an event called to discuss open-source distributed network.
- The #NoSQL was coined by Eric Evans and few other database people at the event found it suitable to describe these non-relational databases.

Big data technology Landscape

Features of NoSQL databases:

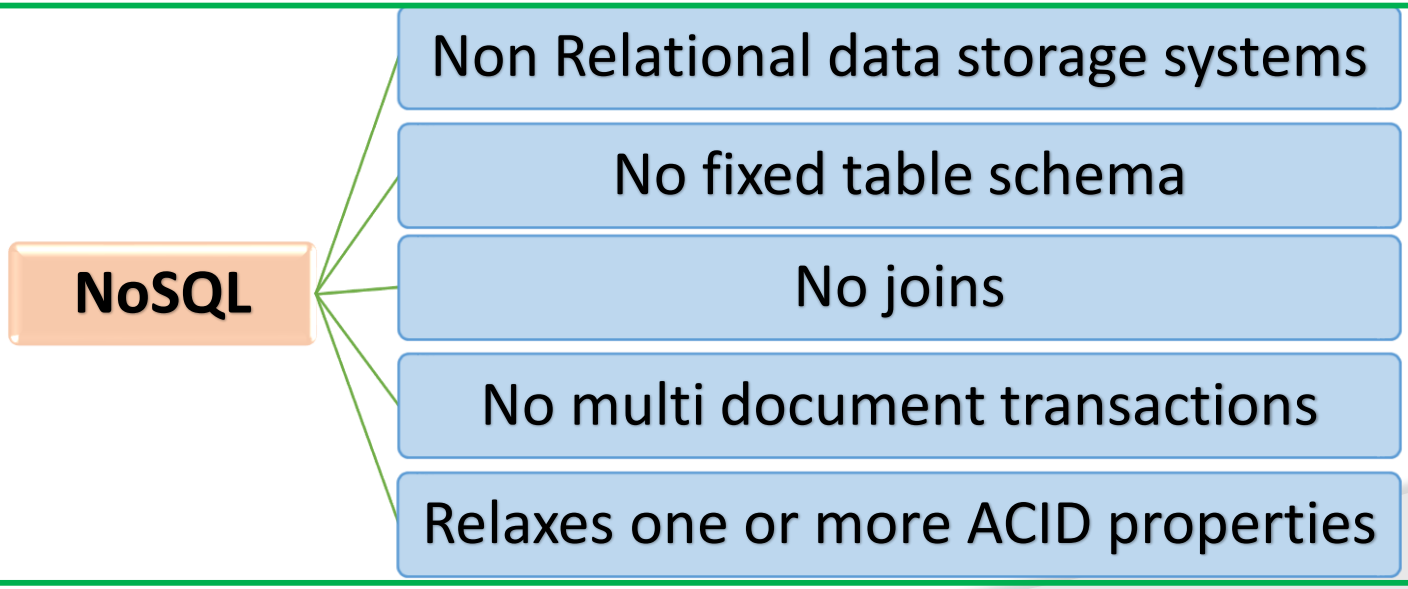
- open source.
- non-relational.
- distributed.
- schema-less.
- cluster friendly.
- born out of 21" century web applications.

Uses

- Real-time web applications.
- Stock log data which can then be pulled for analysis.
- Store social media data and all such data which cannot be stored and analyzed comfortably in RDBMS.

What is NoSQL

- NoSQL stands for Not Only SQL.
- These are **non-relational, open source, distributed** databases.
- They are hugely popular today owing to their **ability to scale out** or scale horizontally and the adoptness at dealing with a rich variety of data: **structured, semi-structured and unstructured data.**



NoSQL Databases

Non-relational: Do not adhere to relational data model. In fact, they are either **key-value pairs or document-oriented or column-oriented or graph-based databases**.

Distributed: The data is distributed across several nodes in a cluster constituted of low-cost commodity hardware.

No support for ACID properties (Atomicity, Consistency, Isolation, and Durability):

- They do not offer support for ACID properties of transactions.
- Brewer's CAP (Consistency, Availability, and Partition tolerance) theorem and Compromising on consistency in favor of availability and partition tolerance.

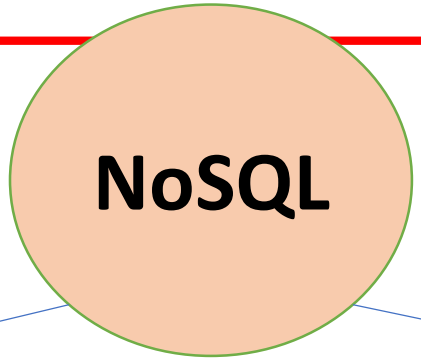
No fixed table schema: NoSQL databases are becoming increasing popular owing to their support for flexibility to the schema.

They do not mandate for the data to strictly adhere to any schema structure at the time of storage.

NoSQL Databases

NoSQL databases are non-relational. They can be broadly classified into the following:

1. Key-value or the big hash table.
2. Schema-less.



Key value or
the big hash table
Amazon S3 (Dynamo)
Scalaris

Schema less
Cassandra (Column based)
CouchDB (Document Based)
Neo4j (Graph based)
Hbase (Column based)

Types of NoSQL Databases

1. Key-value: It maintains a big hash table of keys and values.
Example, Dynamo, Redis, Riak, etc.
Sample Key valuePair in Key value database

Key	Value
FirstName	Sim
Last Name	Davis

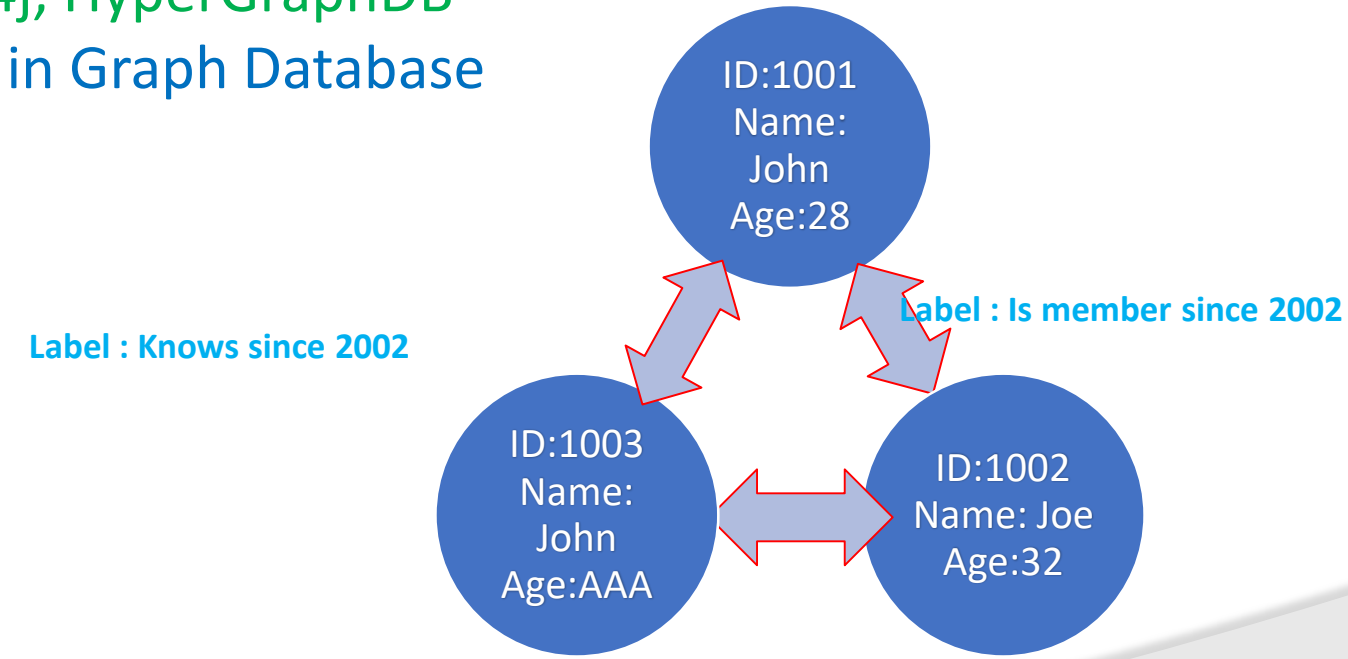
2. Document: It maintains data in collections constituted of documents.
Example, MongoDB, Apache CouchDB, Couchbase, MarkLogic, etc.
Sample Document in Document Database

```
{
"Book Name": Fundamentals of Big Data",
"Publisher" "Wiely India",
Year of Publication": "2011"
}
```

Types of NoSQL Databases

3. Column: Each storage block has data from only one column.
Example: Cassandra, HBase, etc.

4. Graph: They are called network database.
Example, Neo4j, HyperGraphDB
Sample Graph in Graph Database



Types of NoSQL Databases

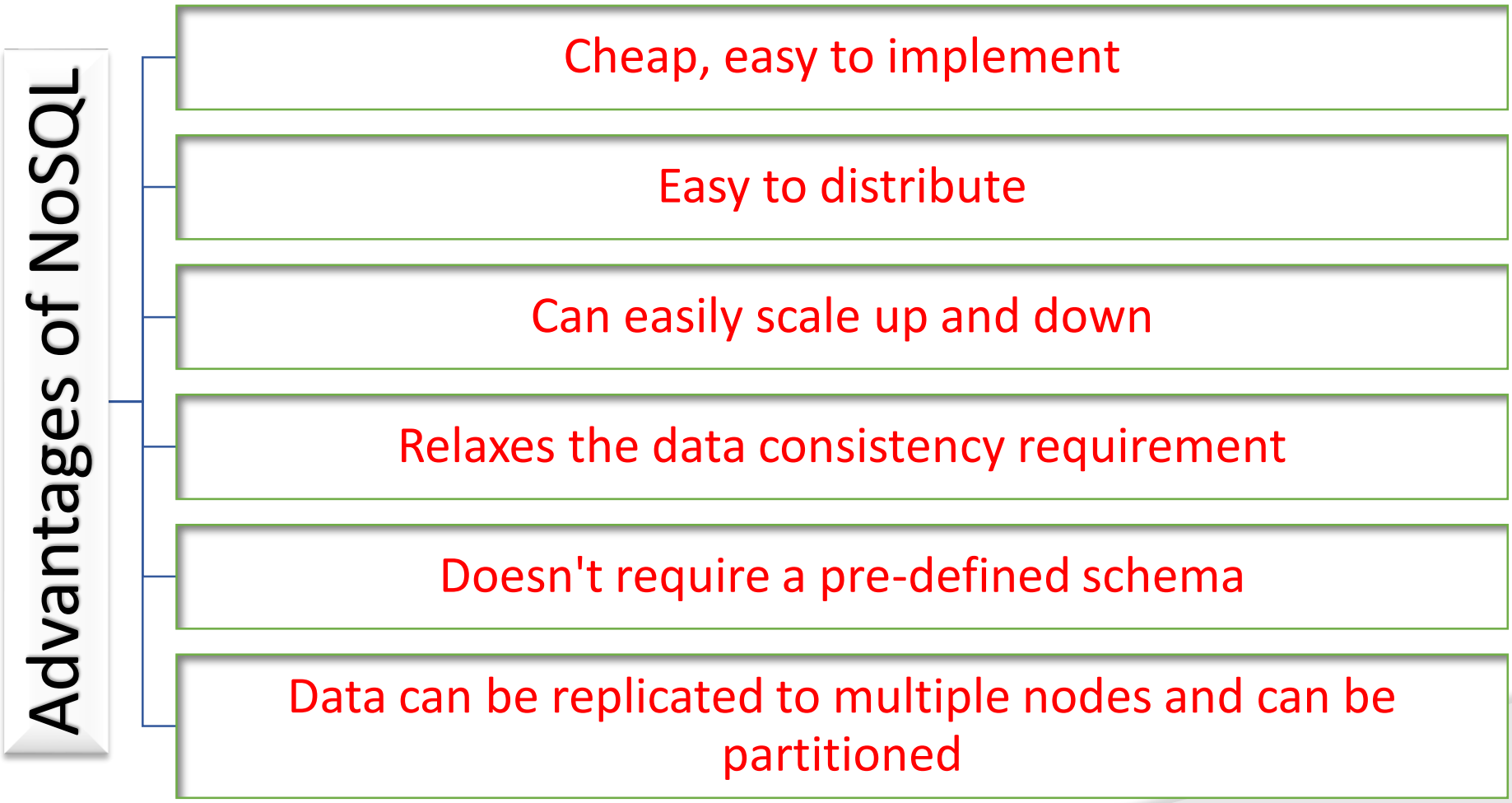
Popular Schema-less Databases

Key Value Data Store	Column Oriented Data Store	Document Data Store	Graph Data Store
Riak	Cassandra	MongoDB	InfiniteGraph
Rdis	HBase	CouchDB	Neo4j
Membase	HyperTable	RavenDB	AllegroGraph

Why NoSQL ?

- Scale out architecture instead of the monolithic architecture of relational databases.
- It can house large volumes of structured, semi-structured, and unstructured data.
- Dynamic schema: NoSQL database allows insertion of data without a predefined schema.
  - It facilitates application changes in real time.
  - Faster development, easy code integration, and requires less database administration.
- Auto-sharding : Automatically spreads data across an arbitrary number of servers.
  - It balances the load on servers
  - If any server goes down, it quickly replaces without major activity disruption
- Replication: Good support of replication.
- Guarantees high availability, fault tolerance and disaster recovery.

Advantages of NoSQL



Advantages of NoSQL

1. Can easily scale up and down: NoSQL database supports scaling rapidly and even allows to scale to the cloud.
   Cluster scale: It allows distribution of database across **100+ nodes** often in multiple data centers.
   Performance scale: It sustains over **100,000+ database** reads and writes per second.
   Data scale: It supports housing of **1 billion+ documents** in the database.
2. Doesn't require a pre-defined schema: NoSQL does not require any adherence to pre-defined schema. It is flexible.
   Example, In MongoDB, the documents (equivalent of records in RDBMS) in a collection (equivalent of table in RDBMS) can have different sets of key-value pairs.

Advantages of NoSQL

3. Cheap, easy to implement: Deploying NoSQL have benefits of scale, high availability, fault tolerance, etc. Lower operational costs
4. Relaxes the data consistency requirement : Adherence on CAP theorem. Most of the NoSQL databases compromises on consistency in favor of availability and partition tolerance.
5. Data can be replicated to multiple nodes and can be partitioned

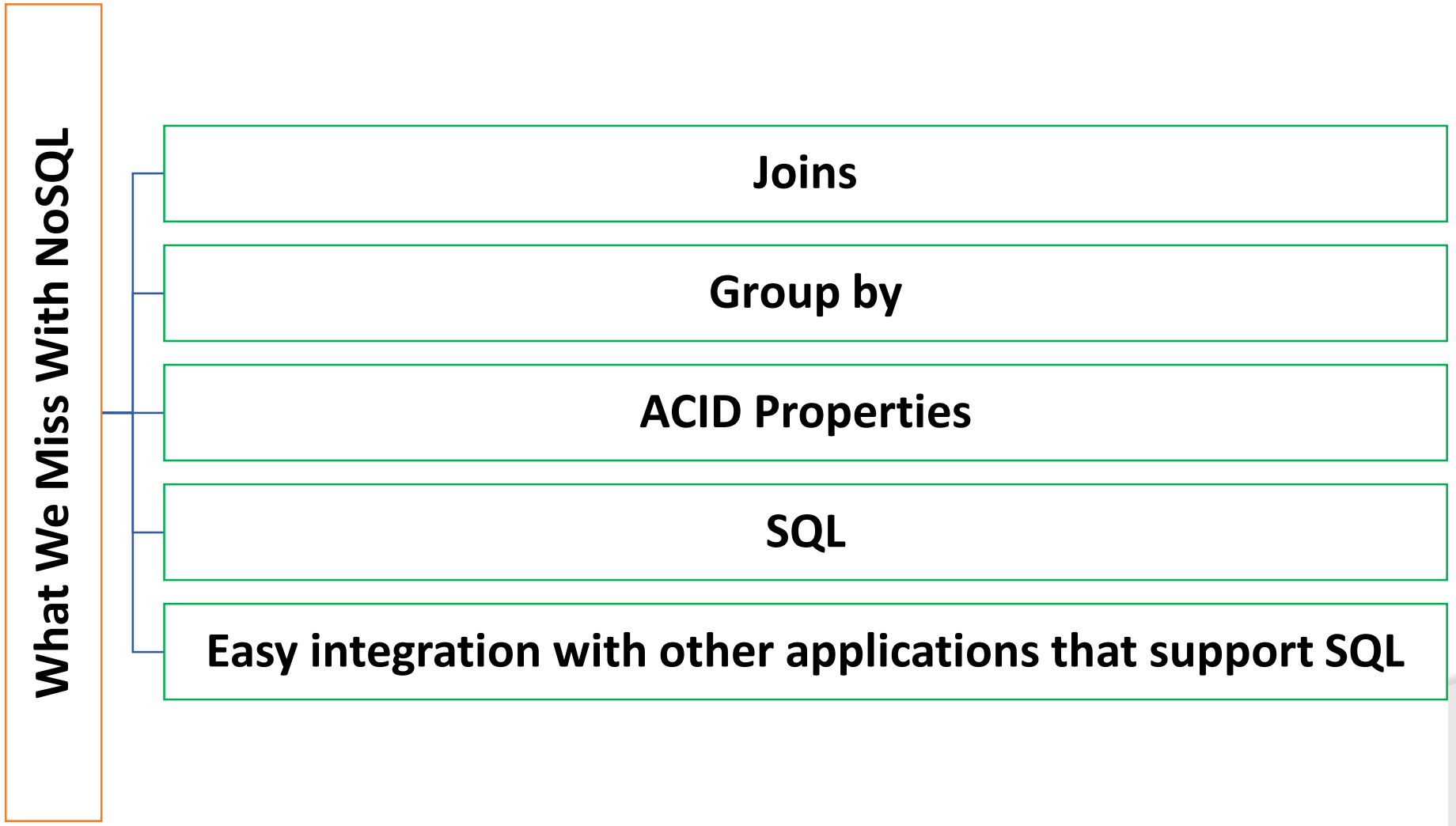
There are two terms to discuss

Sharding: Sharding is different pieces of data are distributed among multiple servers.

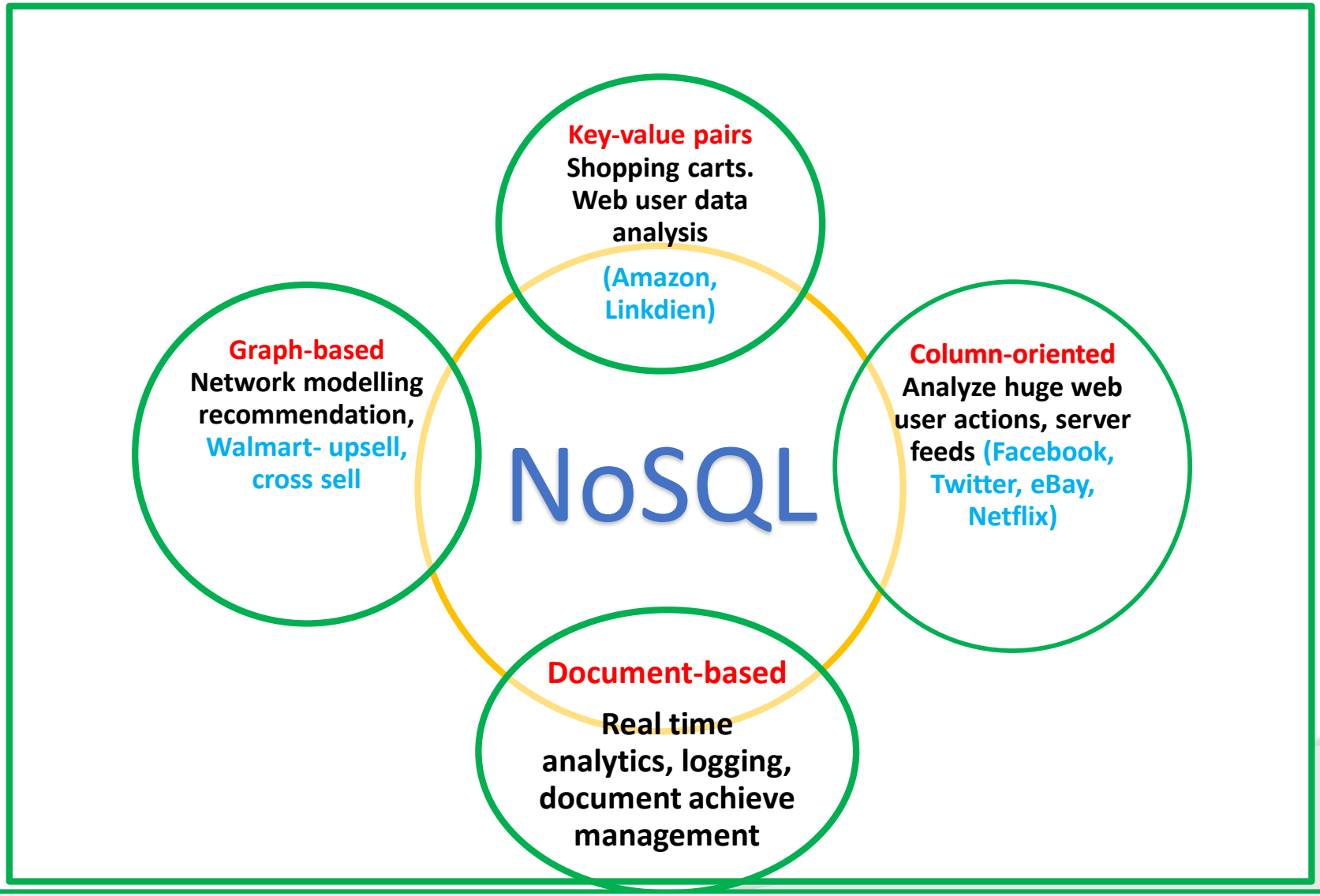
- NoSQL databases support auto-sharding.
- Servers can be added or removed from the data layer with out application down time.

Replication : Multiple copies of data are stored across the cluster and across data centers. This promises high availability and fault tolerance

What We Miss With NoSQL?



Use of NoSQL in Industry



NoSQL Vendors

Company	Product	Most Widely Used by
Amazon	Dynamo DB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, eBay
Google	BigTable	Adobe Photoshop

SQL vs NoSQL

SQL	NoSQL
Relational Database	Non Relational Database, Distributed Database
Relational Model	Model less Approach
Predefined Schema	Dynamic Schema for unstructured data
Table based databases	Document based, graph based or wide column store or key value pairs databases
Vertically scalable (by increasing system databases)	Horizontally scalable (by creating cluster of commodity machines)
Uses SQL (Structured Query Language)	Used UnQL (Unstructured Query Language)
Not preferred for large datasets	Preferred for Large datasets
Not a best fit for Hierarchical Data	Best fit for Hierarchical storage as it follows the key-value pair of storing data similar to JSON (Java Script Object Notation)
Emphasis on ACID properties	Follows Brewers CAP theorem
Excellent support from vendors	Relies on community support
Supports complex querying	Does not have good support for complex querying

MySQL

- MySQL is open source relational database management system that works on many platforms.
- Provides multiuser access to support many storage engines
- Commercial version from Oracle to get premium support

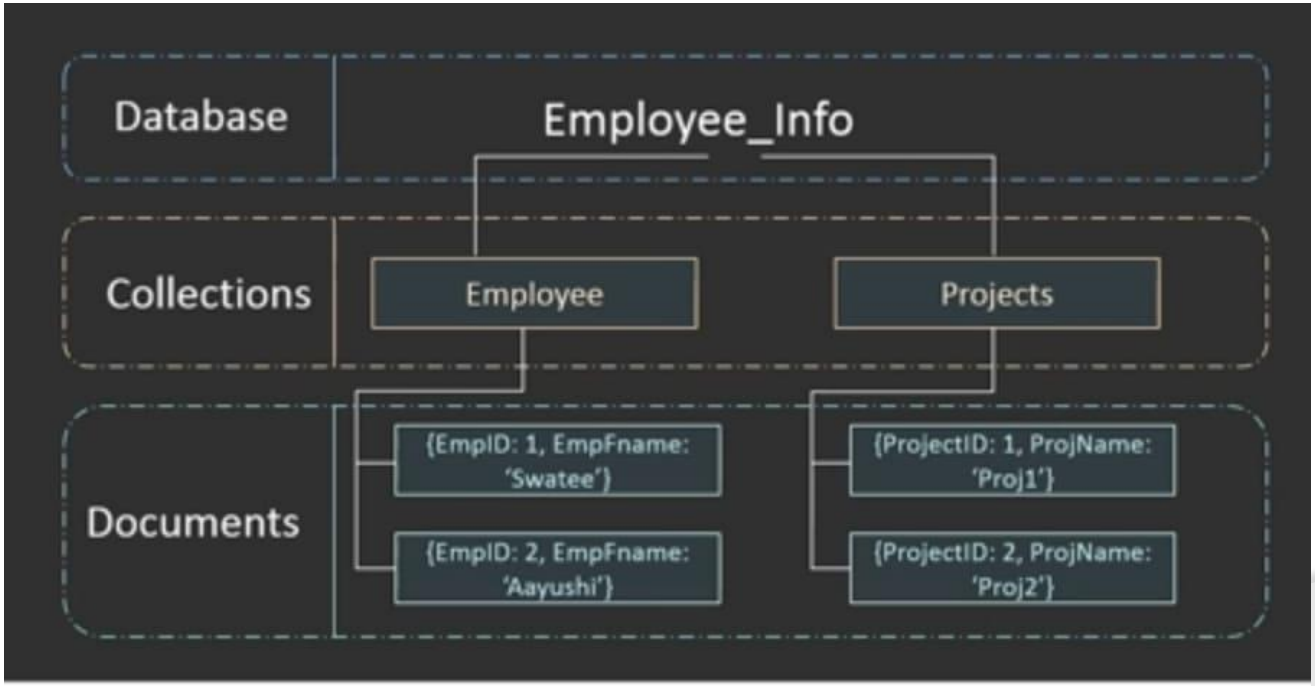
MongoDB

- **NoSQL database**
- **Non relational database, which stores data as documents in a binary representation called BSON(Binary JSON).**
- **Stores related information for fast query access through the MongoDB query language**

NoSQL Database structure

- NoSQL, Not only SQL
- Provides a mechanism for storage and retrieval of data
- Next generation database.
- No specific schema and can handle huge amount of data

Database Structure



NoSQL Database structure

Document-based data model

Looks like JSON. Example:

```
{
        "_id" : ObjectID("7b33e366ae32223aee34fd3"),
        "title" : "A blog post about MongoDB",
        "content" : "This is a blog post about MongoDB",
        "comments": [
                {
                        "name" : "Frank",
                        "email" : fkane@sundog-soft.com,
                        "content" : "This is the best article ever written!"
                        "rating" : 1
                }
        ]
}
```

MySQL vs mongoDB

Uses Structured Query Language

```
INSERT INTO employees (employee_id,
empage)
VALUES ('abc001', '23')
```

Uses Unstructured Query Language

```
db.employees.insert({
  employee_id: 'abc001',
  age: 23,
})
```

MySQL

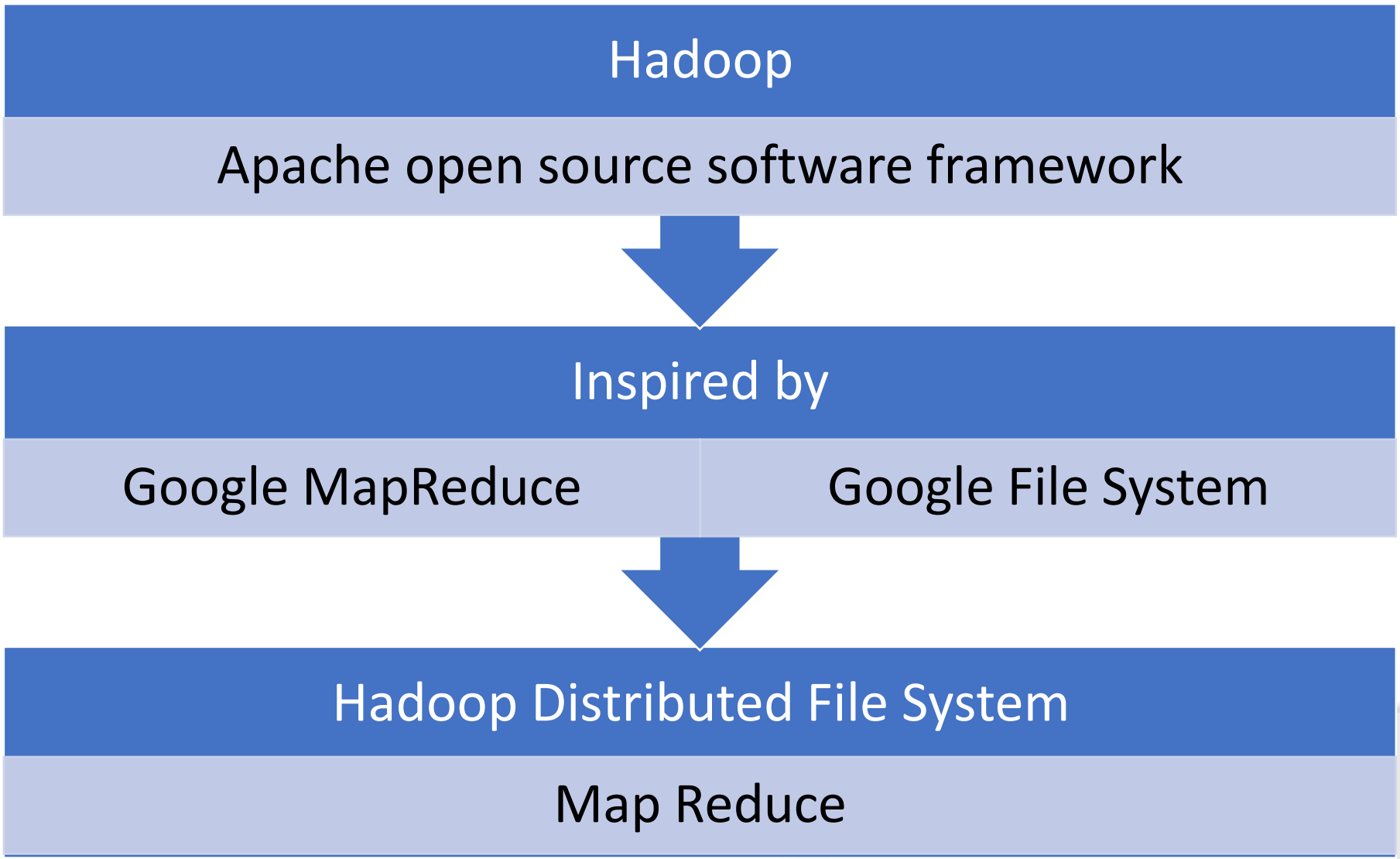
- ✓ Fixed Schema
- ✓ High Transaction
- ✓ Low Maintenance
- ✓ Data Security
- ✓ Limited Budget

mongoDB

- ✓ Unstable Schema
- ✓ No Database Administrator
- ✓ High Availability
- ✓ Cloud Computing
- ✓ In-Built Sharding

Hadoop

- Hadoop is an open-source project of the Apache foundation.
- It is a framework written in Java, originally developed by Doug Cutting in 2005 who named it after his son's toy elephant.
- He was working with Yahoo then.
- It was created to support distribution for "Nutch", the text search engine.
- Hadoop uses Google's MapReduce and Google File System technologies as its foundation.
- Hadoop is now a core part of the computing infrastructure for companies such as Yahoo, Facebook, LinkedIn, Twitter, etc.


Hadoop
Apache open source software framework
Inspired by
Google MapReduce
Google File System
Hadoop Distributed File System
Map Reduce

Features of Hadoop

1. It is optimized to **handle massive quantities of structured, semi-structured, and unstructured data,** using commodity hardware, that is, relatively inexpensive computers.
2. Hadoop has a **shared nothing architecture**.
3. It **replicates its data across multiple computers** so that if one goes down, the data can still be processed from another machine that stores its replica.
4. Hadoop is for high throughput rather than low latency. It is a batch operation handling massive quantities of data; therefore the response time is not immediate.
5. It complements **On-Line Transaction Processing (OLTP) and On-Line Analytical Processing (OLAP).** However, it is not a replacement for a relational database management system.
6. It is **NOT good when work cannot be parallelized** or when there are dependencies within the data.
7. It is NOT good for processing **small files**. It works best with huge data files and datasets.

Key Advantages of Hadoop

1. Stores data in its native format:

- Hadoop's data storage framework (**HDFS - Hadoop Distributed File System**) can store data in its native format.
- There is no structure that is imposed while keying in data or storing data.
- HDFS is pretty much **schema-less.**
- It is only later when the data needs to be processed that structure is imposed on the raw data.

2. Scalable:

Hadoop can store and **distribute very large datasets (involving thousands of terabytes of data) across hundreds of inexpensive servers** that operate in parallel.

3. Cost-effective:

Owing to its **scale-out architecture**, Hadoop has a much reduced cost/terabyte of storage and processing.

Key Advantages of Hadoop

4. Resilient to failure:

- Hadoop is **fault-tolerant**.
- It practices replication of data diligently which means whenever data is sent to any node, the same data also gets replicated to other nodes in the cluster, there by ensuring that in the event of a node failure, there will always be another copy of data available for use.

5. Flexibility:

- One of the key advantages of Hadoop is its ability to work with all kinds of data: structured, semi-structured, and unstructured data.
- It can help derive meaningful business insights from email conversations, social media data, click-stream data, etc.
- It can be put to several purposes such as log analysis, data mining, recommendation systems, market campaign analysis, etc.

6. Fast: Processing is extremely fast in Hadoop as compared to other conventional systems owing to the "move code to data" paradigm. Hadoop has a **shared-nothing architecture**

Key advantages offered by Hadoop for bigdata analytics

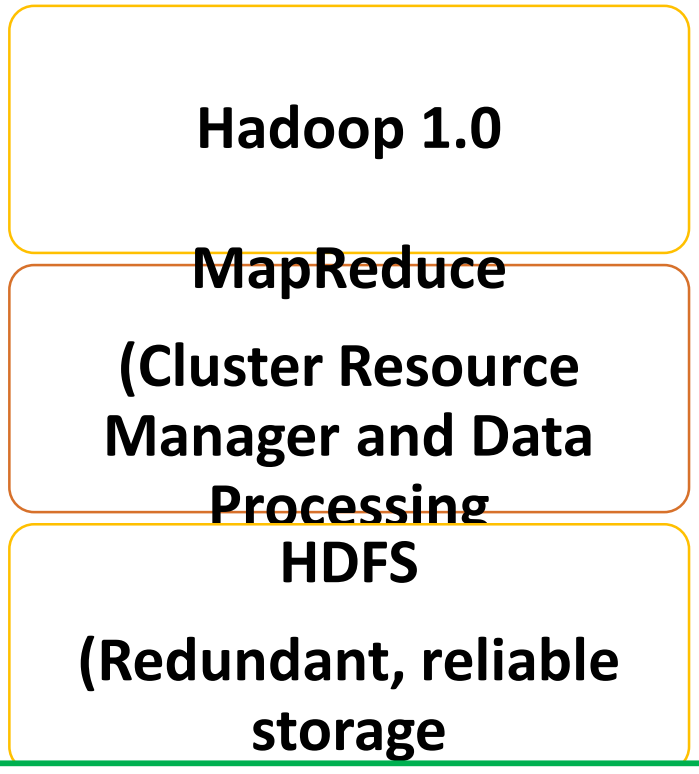
- Store data in its native format
- No loss of information as there is no translation / transformation to any specific schema
- Scalability – proven to scale by companies like Facebook & Yahoo
- Delivers new insights
- Higher Availability – Fault tolerance through replication of data / failover across computer nodes
- Reduced cost – lower cost / terabyte of storage and processing
- H/W can be added or swapped in or out of a cluster

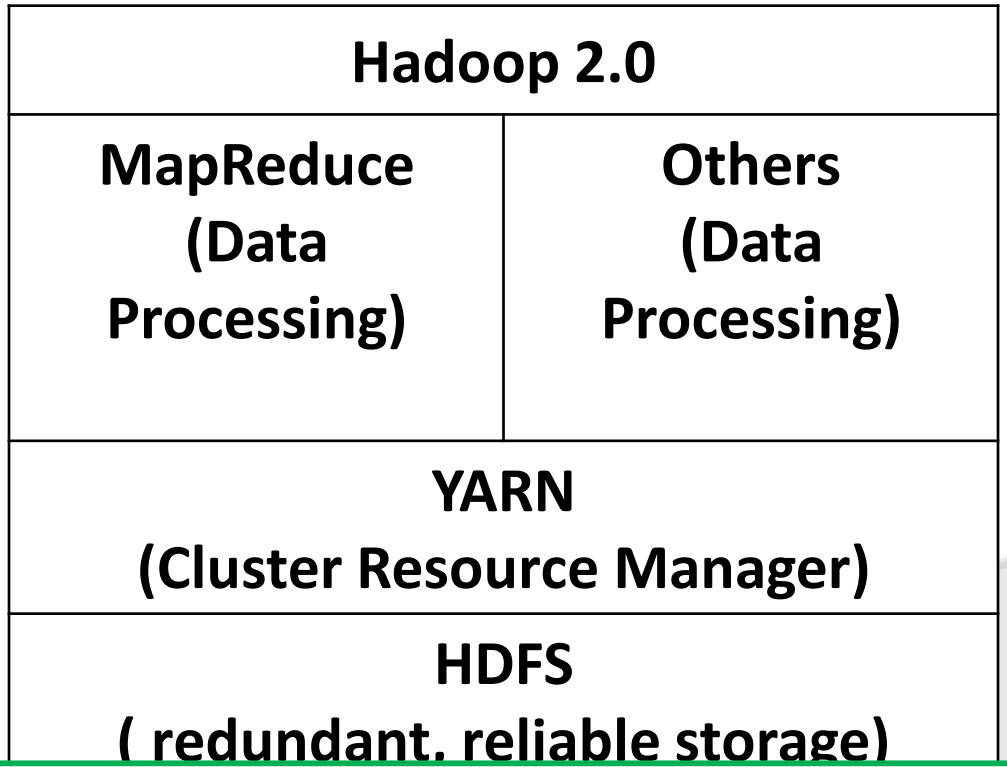
Hadoop

Versions of Hadoop

There are two versions of Hadoop available:

1. **Hadoop 1.0**
2. **Hadoop 2.0**





Hadoop 1.0

Tow parts:

1. Data storage framework 2. Data processing framework

Data storage framework

- General purpose file system called Hadoop Distributed file system (HDFS)
- HDFS is schema less.
- Stores data files in original form

Data processing framework

- A simple functional programming model
- Initially popularized by Google as MapReduce
- Uses two functions : MAP and REDUCE to process data
- Mapper – takes key-value pairs and generate intermediate data
- Reducer – takes input from Mapper and produce output
- The two functions work in isolation from one another
- Processing to be highly distributed in a highly parallel, fault tolarent and scalable way

Hadoop 1.0 Limitations

- The requirement for MapReduce programming expertise along with proficiency required in other programming languages, notably Java.
- It supported only batch processing suitable for tasks such as log analysis, large scale data mining
- Not suitable for other kinds of projects
- Tightly computationally coupled with MapReduce,

which meant that the established data management vendors were left with two options:

- Either rewrite their functionality in MapReduce to execute in Hadoop or extract the data from HDFS and process it outside of Hadoop.
- None of the options were viable as it led to process inefficiencies caused by the data being moved in and out of the Hadoop cluster.

UNIT 2 - INTRODUCTION TO HADOOP

- Introducing Hadoop
- Why Hadoop
- Why not RDBMS
- RDBMS vs Hadoop
- Distributed Computing Challenges
- History of Hadoop
- Clickstream data analysis with Hadoop –Pig
- Demonstration on usecase
- Hadoop Distributors
- Processing Data with Hadoop
- Interacting with Hadoop Ecosystem

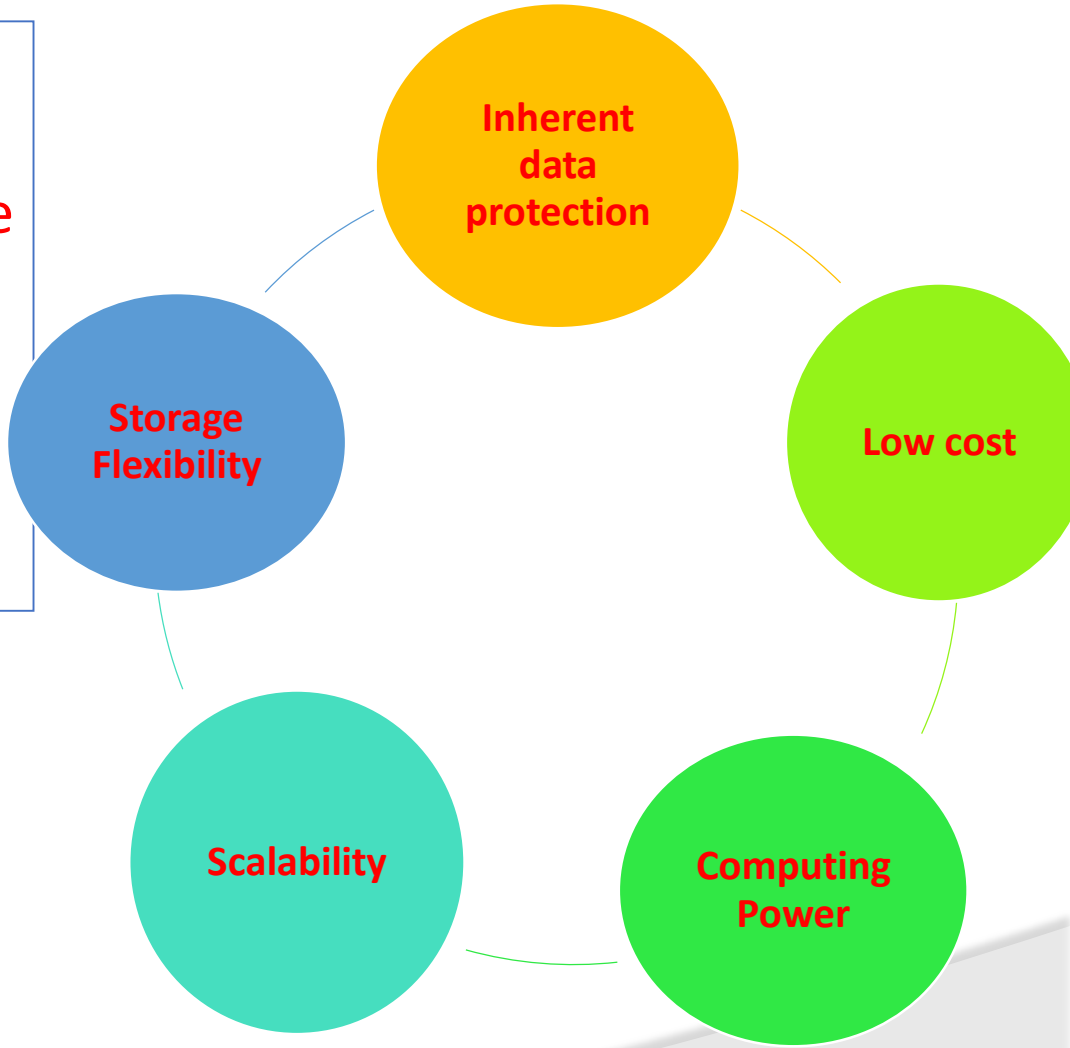
UNIT 2 - INTRODUCTION TO HADOOP

- Introducing Hadoop
- Why Hadoop
- Why not RDBMS
- RDBMS vs Hadoop
- Distributed Computing Challenges
- History of Hadoop
- Clickstream data analysis with Hadoop –Pig
- Demonstration on usecase

Why Hadoop?

The key considerations are

- Capability to handle massive amounts of data
- Different categories of data
- Quickly



Why Hadoop?

1. Low Cost:
- Open source framework
- Uses commodity hardware to store huge amount of data

2. Computing Power
- Based on distributing computing model – which process very large volume of data fairly quickly
- More number of nodes , more processing power

3. Scalability

Adding number of nodes as system grows

4. Storage Flexibility
- Like traditional systems, data need not be preprocessed before storing
- Flexibility of storing data as well as using data
- Can store all variety of data like images, videos, and text

5. Inherent data protection
- Protects data and executing applications against hardware failure.

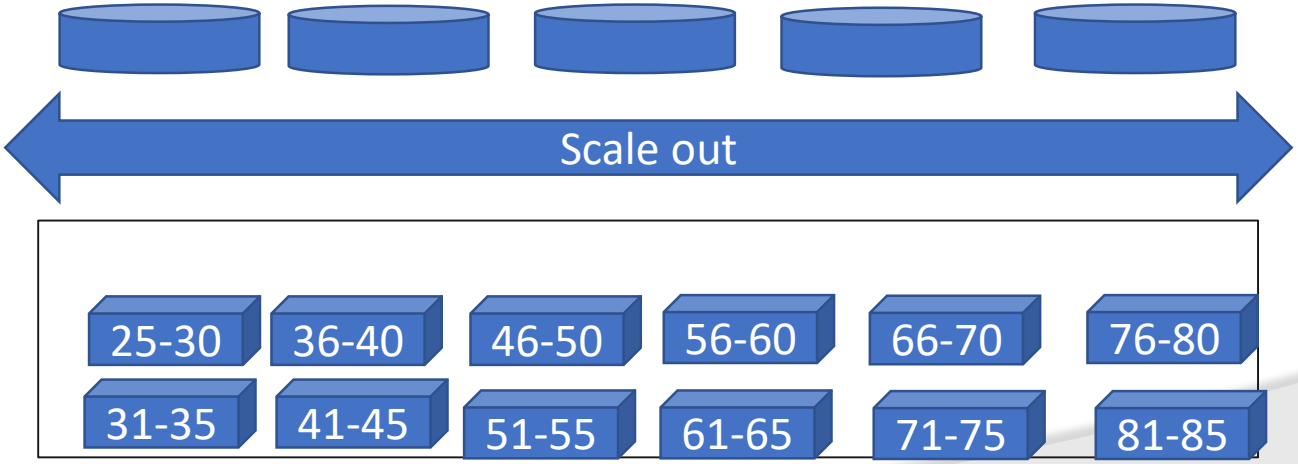
Hadoop Framework

New paradigm -Commodity hardware, Distributed file system and distributed computing

In this group of machines are gathered called a Cluster

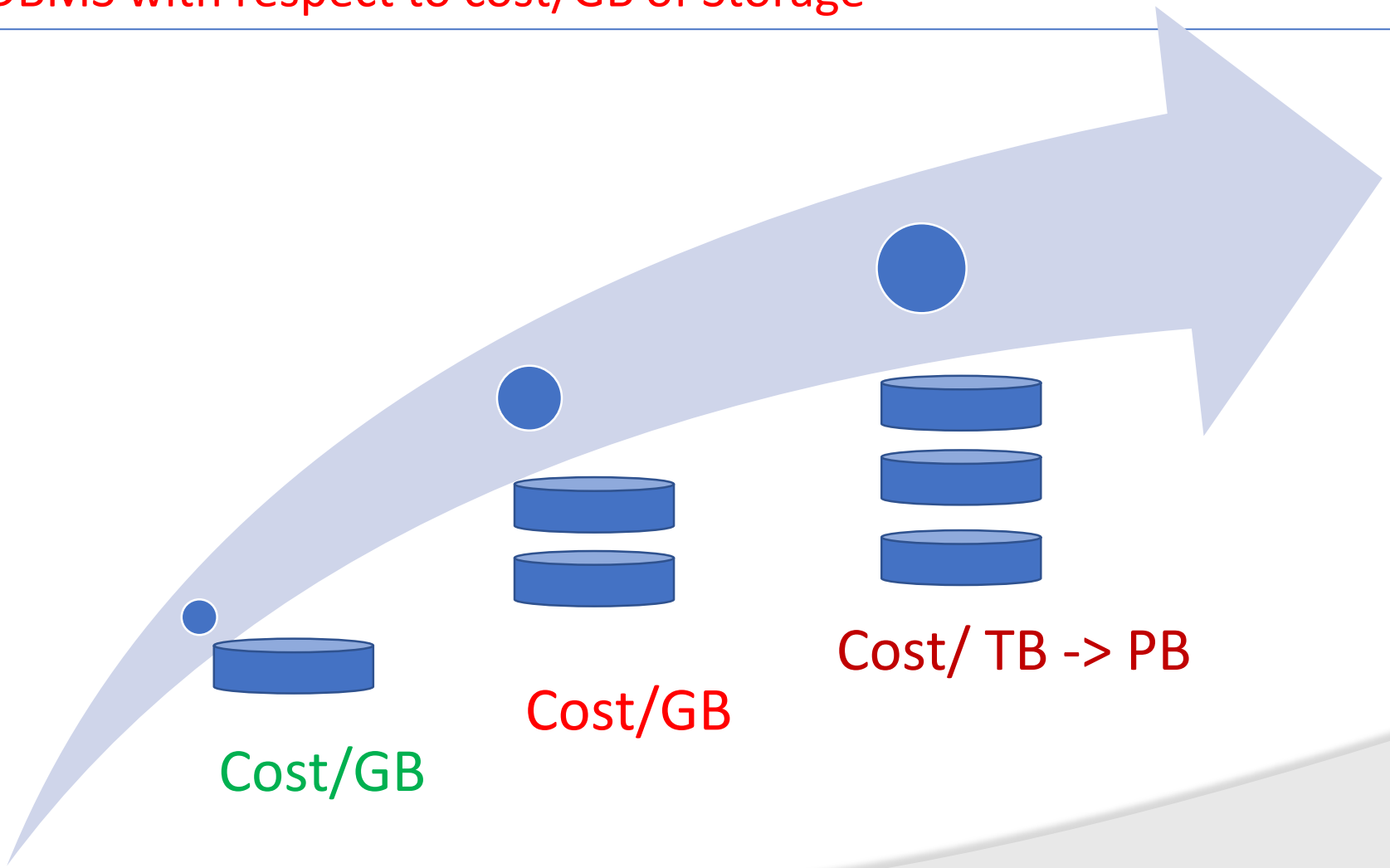
The data can be managed with Hadoop:

- Distributes the data and duplicates chunks of each data file across several nodes
- Ex. 25-30 nodes is one chunk of data
- Locally available compute resource is used to process each chunk of data in parallel
- Hadoop framework handles failover



Why not RDBMS?

RDBMS with respect to cost/GB of Storage



Differences between RDBMS and Hadoop

Parameter	RDBMS	HADOOP
System	Relational database management systems	Node based flat structure
Data	Structured data	Suitable for structured, unstructured data
Processing	OLTP	Analytical, Big Data Processing
Choice	Data needs consistent relationship	Does not require consistent relationships among data
Processor	Needs expensive hardware or high end processors to store huge volumes of data	In hadoop cluster a node requires only processor, a network card and few hard drives
Cost	Cost around 10000-14000Dolors per tera bytes of storage	Cost around 4000 Dolors per terabytes of storage

Distributed Computing Challenges

Two major challenges:

Hardware Failure

In distributed system, Several servers are networked together

- In general, a hard disk may fail in 3 years

Key challenge: A possibility of at least being down every day

Solution in Hadoop: Replication Factor (RF)

- No. of data copies of a given data item / block stored across the network

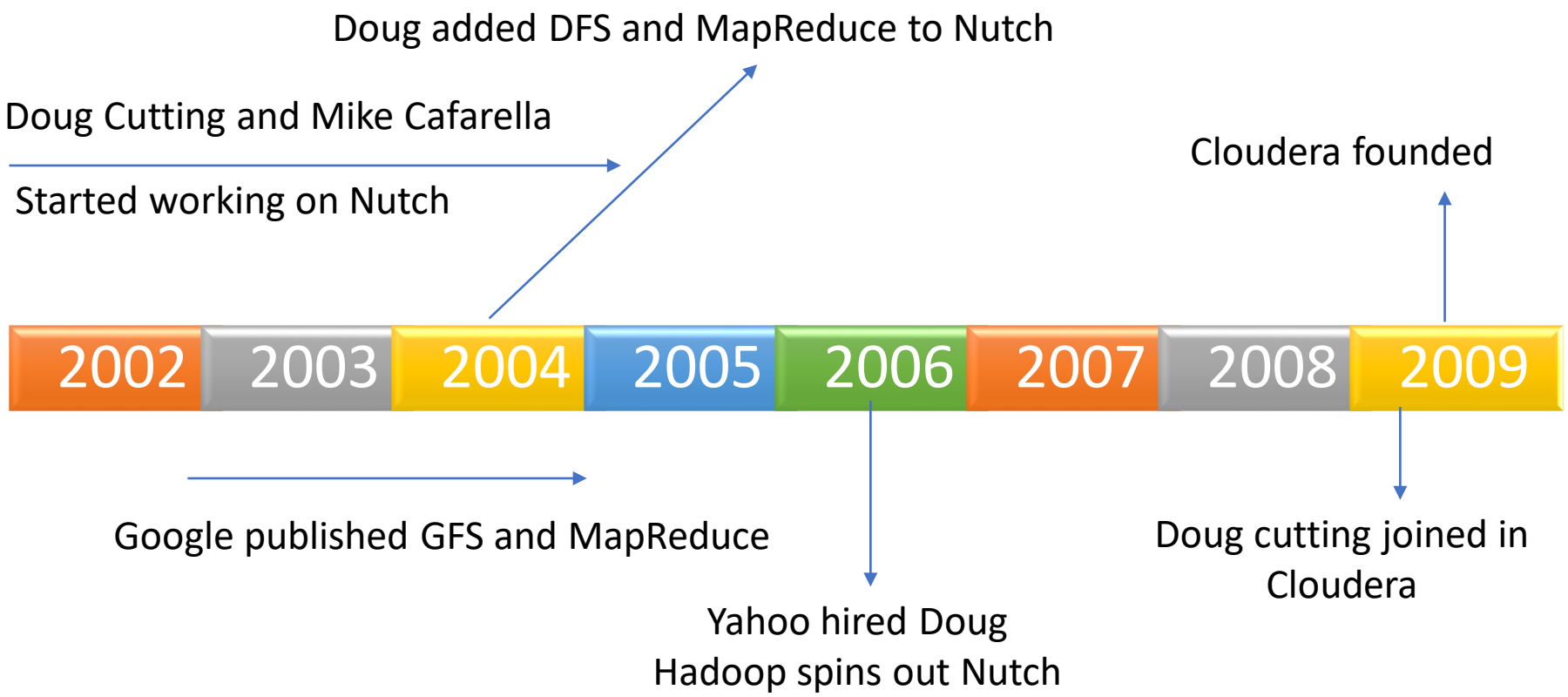
Process the distributed data

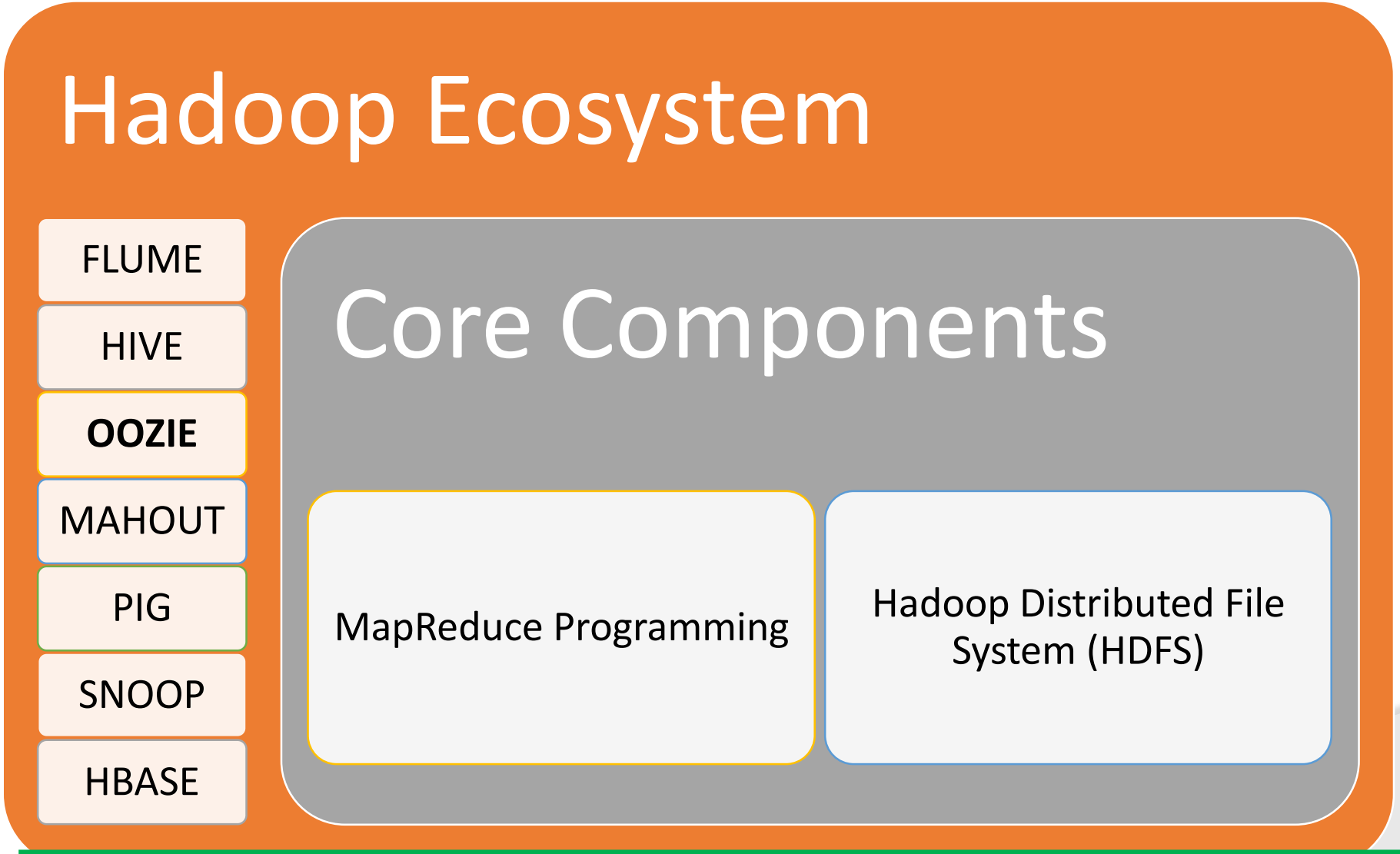
In distributed system, the data is spread across the network on several machines

Key challenge: Integrate the data available on several machines prior to processing.

Hadoop solution: MapReduce programming

History of Hadoop




Hadoop Ecosystem
FLUME
HIVE
OOZIE
MAHOUT
PIG
SNOOP
HBASE
Core Components
MapReduce Programming
Hadoop Distributed File System (HDFS)

Hadoop Components

Hadoop Core components:

HDFS:

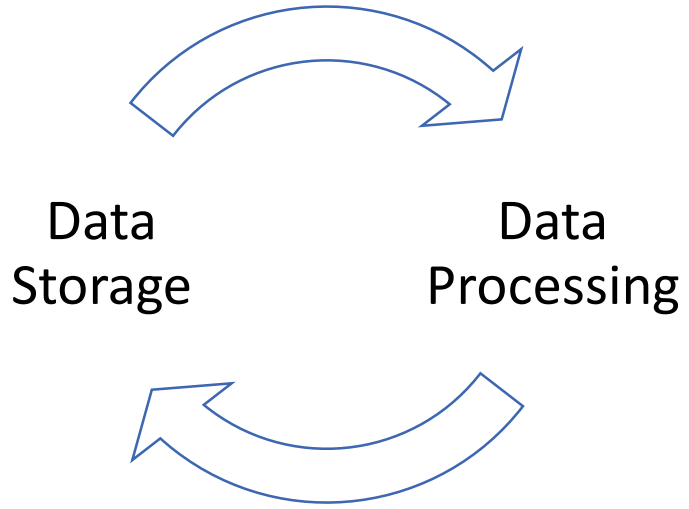
- Storage component
- Distributes data across several nodes
- Natively redundant

Hadoop Ecosystem

- HIVE
- PIG
- SQOOP
- HBASE
- FLUME
- OOZIE
- MAHOUT

Hadoop conceptual layer divided into two layers

- Data storage layer
- Data Processing layer



Hadoop High level Architecture

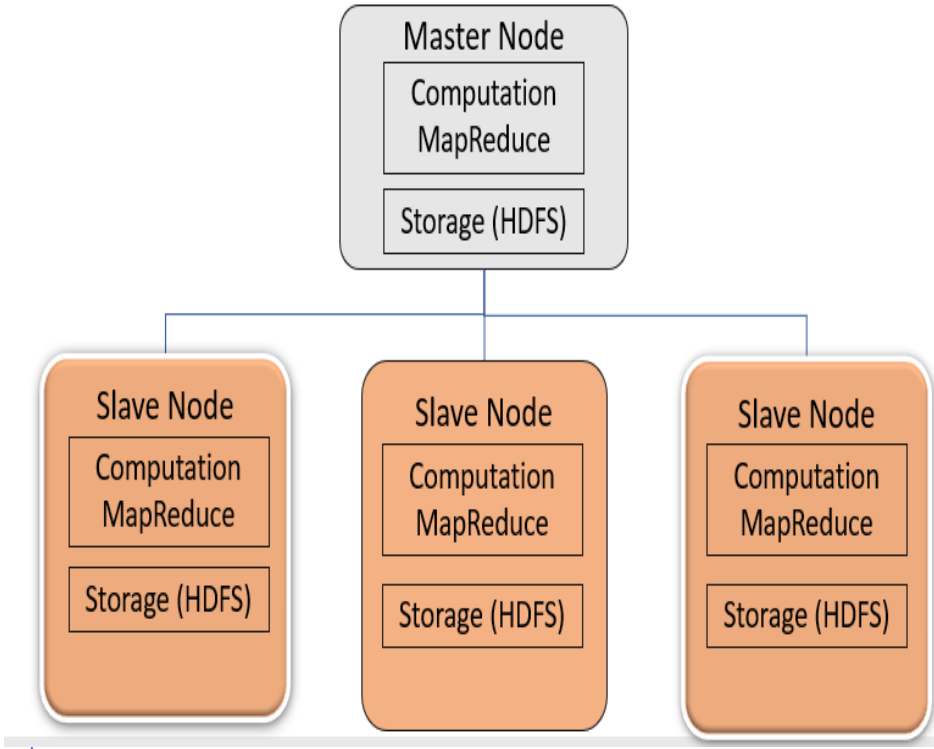
High level architecture of Hadoop -

Distributed Master-Slave Architecture

- **Master node is NameNode**
- **Slave node is DataNodes**

Master HDFS: The main Responsibility is partitioning the data storage across the slave nodes. Also keep track of data

Master MapReduce: It decides and schedules the computation cost on slave nodes on DataNodes



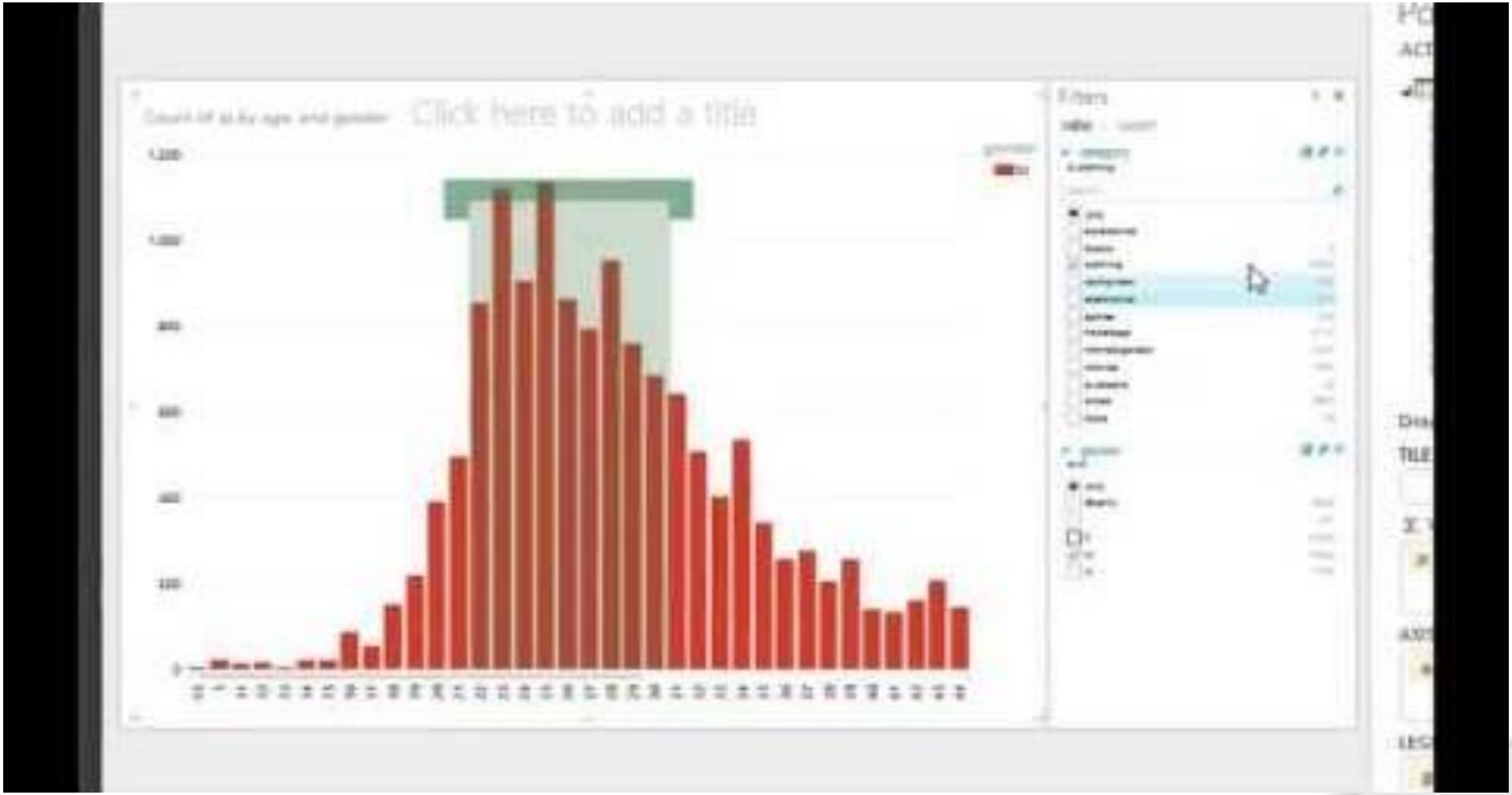
USE CASE OF HADOOP – ClickStream Data

- ClicStream Data – Mouse clicks
- To understand the purchasing behavior of customers.
- It helps online marketers to optimize their product web pages, promotional content etc.
- To improve their business

Click Stream Data Analysis using Hadoop – Key Benefits		
Joins ClickStream data with CRM and Sales data	Stores years of data without much incremental cost	Hive or Pig Script to analyze data

- With Hadoop benifits:
- Join ClickStream data with other data sources
- Scalability helps to store years of data
- Business analysis can use Apache Pig or Hive for website analysis

using Hadoop and the Hortonworks Data Platform to analyze clickstream data to increase online conversions and revenue.



https://www.youtube.com/watch?v=weJI6Lp9Vw0

Thank you & Discussions..

UNIT 2 - INTRODUCTION TO HADOOP

- Hadoop Distributers
- HDFS (HADOOP DISTRIBUTED FILE SYSTEM)
- Architecture
- DataNode
- Secondary NameNode
- Anatomy of File Read
- Anatomy of File Write
- Replica Placement Strategy
- Working with HDFS Commands

Common Hadoop distributors

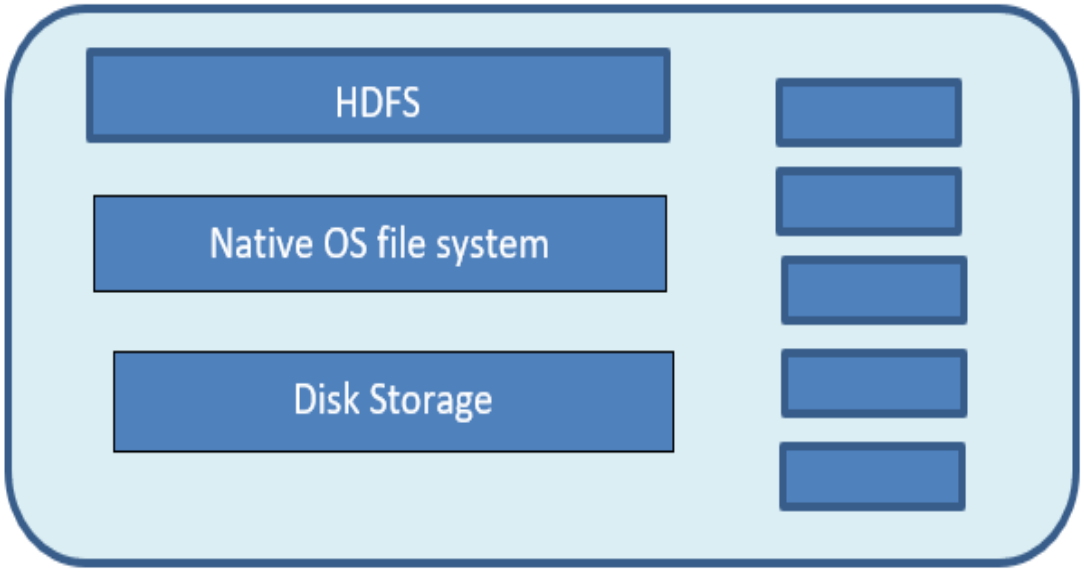
The products that include, Apache Hadoop and other commercial support or tools and utilities related to Hadoop

Cloudera	Hortonworks	MAPR	Apache Hadoop
CDH 4. 0 CDH 5.0	HDP 1.0 HDP 2.0	M3 M5 M8	Hadoop 1.0 Hadoop 2.0

HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

1. Storage component of Hadoop
2. Distributed File System
3. Modeled after Google File System
4. Optimized for high throughput (HDFS leverages large block size and moves computation where data is stored).
5. You can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.
6. Re-replicates data blocks automatically on nodes that have failed
7. You can realize the power of HDFS when you perform read or write on large files (gjgabyes and larger)
8. Sits on top of native file system such as ext3 and ext4.

Hadoop File system

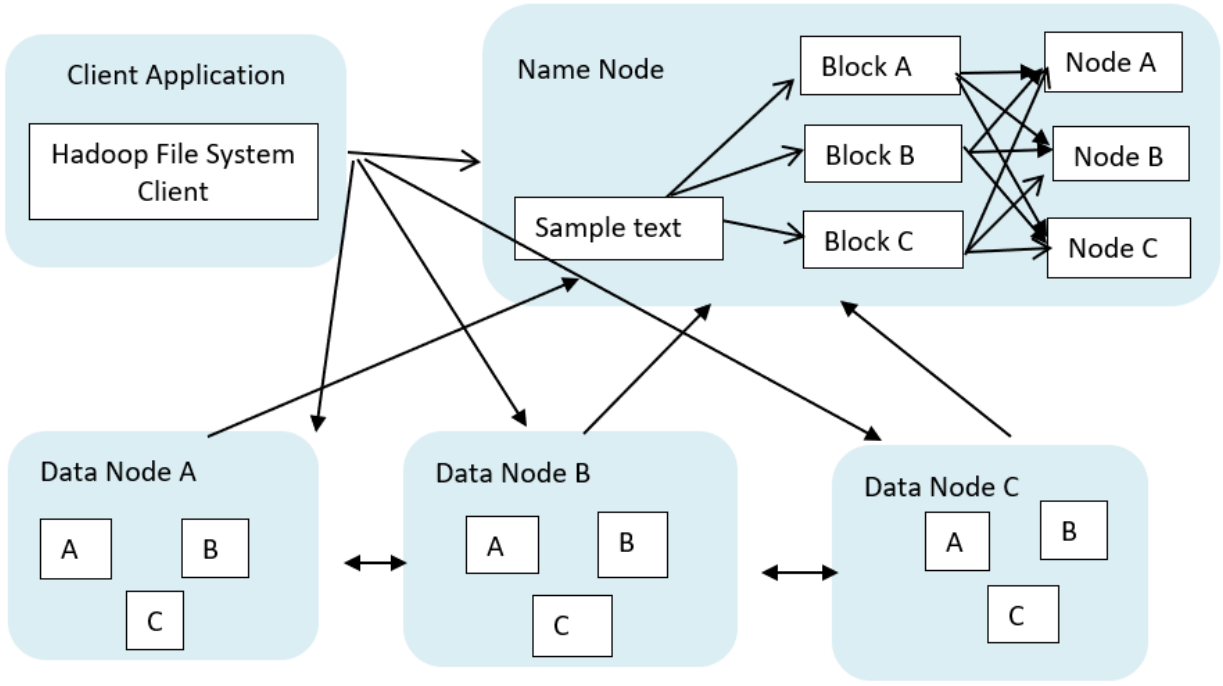


Hadoop Distributed File System

Hadoop Distributed file system Key points		
Block Structured File	Default Replication Factor: 3	Default Block size: 64MB

Hadoop File System Architecture

- Client Application interacts with NameNode for metadata related activities and communicates with DataNodes to read and write files
- DataNodes converse with each other for pipeline reads and writes.



Let us assume, file "Sample.txt is of size 192 MB. As per the default data block size (64 MB), it will be split into three blocks and replicated across the nodes on the cluster based on the default replication factor.

HDFS Daemons

- **NameNode**
- **DataNode**
- **Secondary NameNode**

NameNode – Manages file related operations	
FsImage – File Entire file system is stored	EdiLog – Records every transaction that occurs to file system meta data

HDFS Daemons - NameNode

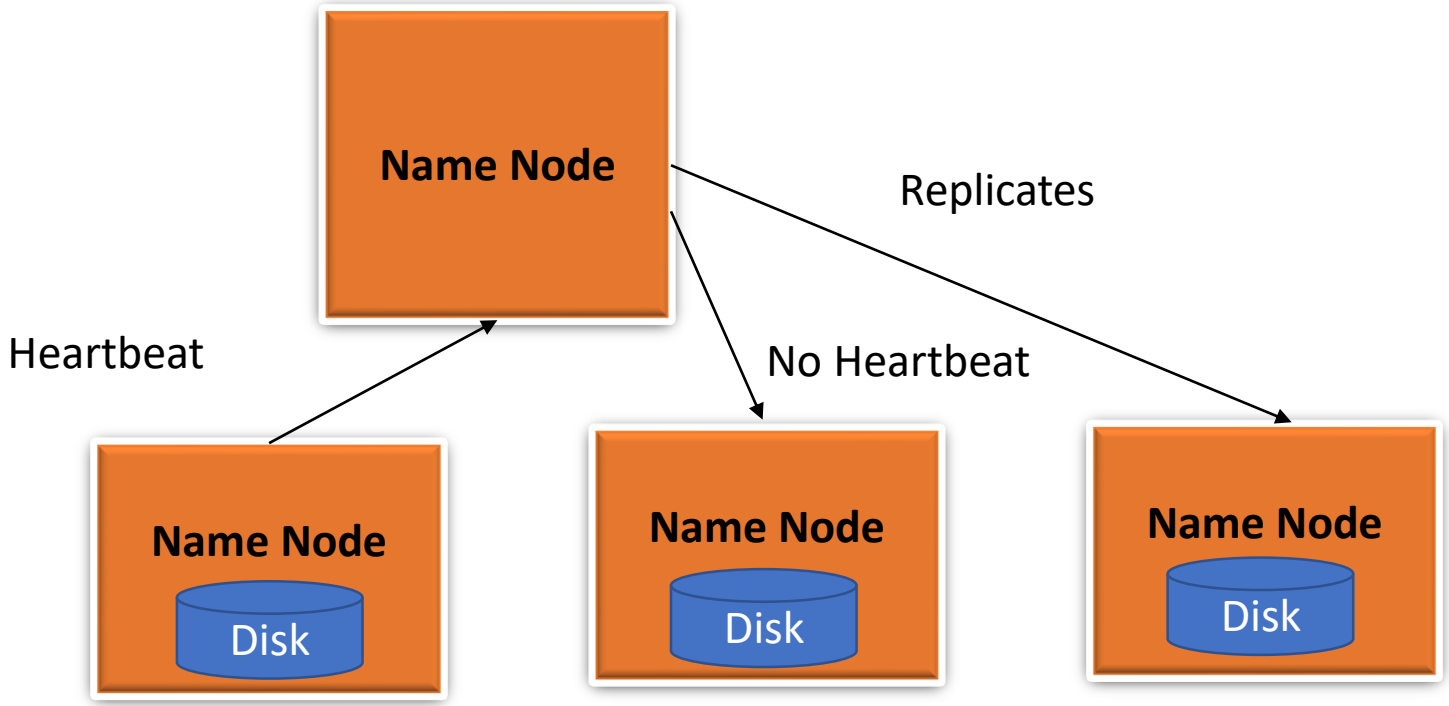
- HDFS breaks a large file into smaller pieces called blocks.
- NameNode uses a rackID to identify DataNodes in the rack.
- A rack is a collection of DataNodes within the cluster.
- NameNode keeps tracks of blocks of a file as it is placed on various DataNodes.
- NameNode manages file-related operations such as read, write, create, and delete.
- Its main job is managing the File System Namespace.
- A file system namespace is collection of files in the cluster.
- NameNode stores HDFS namespace.
- File system namespace includes mapping of blocks to file properties and is stored in a file called Fslmage.
- NameNode uses an Edilog (transaction log) to record every transaction that happens to the file system metadata.

HDFS Daemons - DataNode

- Multiple DataNodes per cluster.
- During Pipeline read and write DataNodes communicates each other.
- A DataNode also continuously sends "heartbeat message to NameNode to ensure the connectivity between the NameNode and DataNode.
- In case there is no heartbeat from a DataNode
- NameNode replicates that DataNode within the cluster and keeps on running as if nothing had happened.

HDFS Daemons - DataNode

- Let us explain the concept behind sending the heartbeat report by the DataNodes to the NameNode

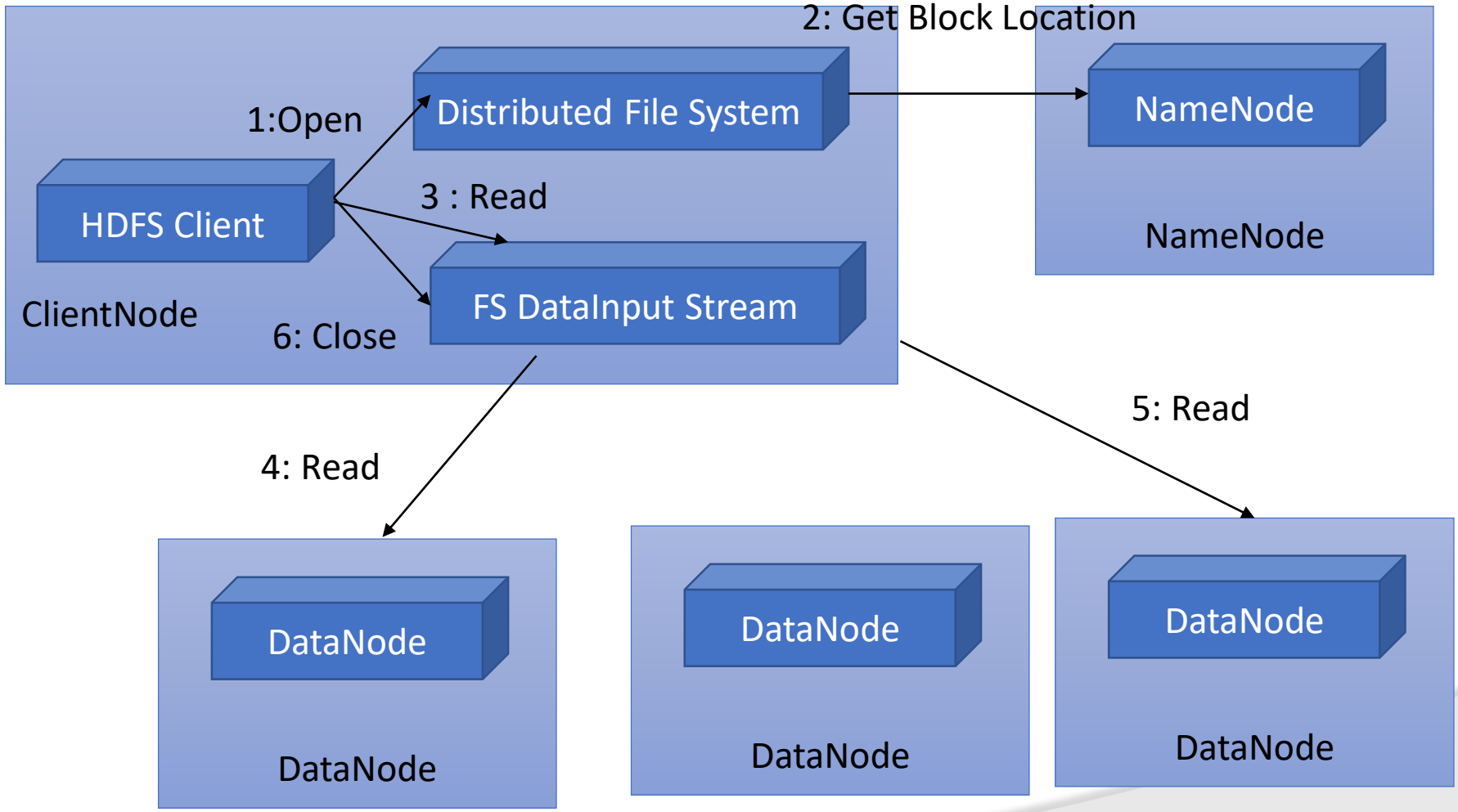


NameNode and DataNode Communication

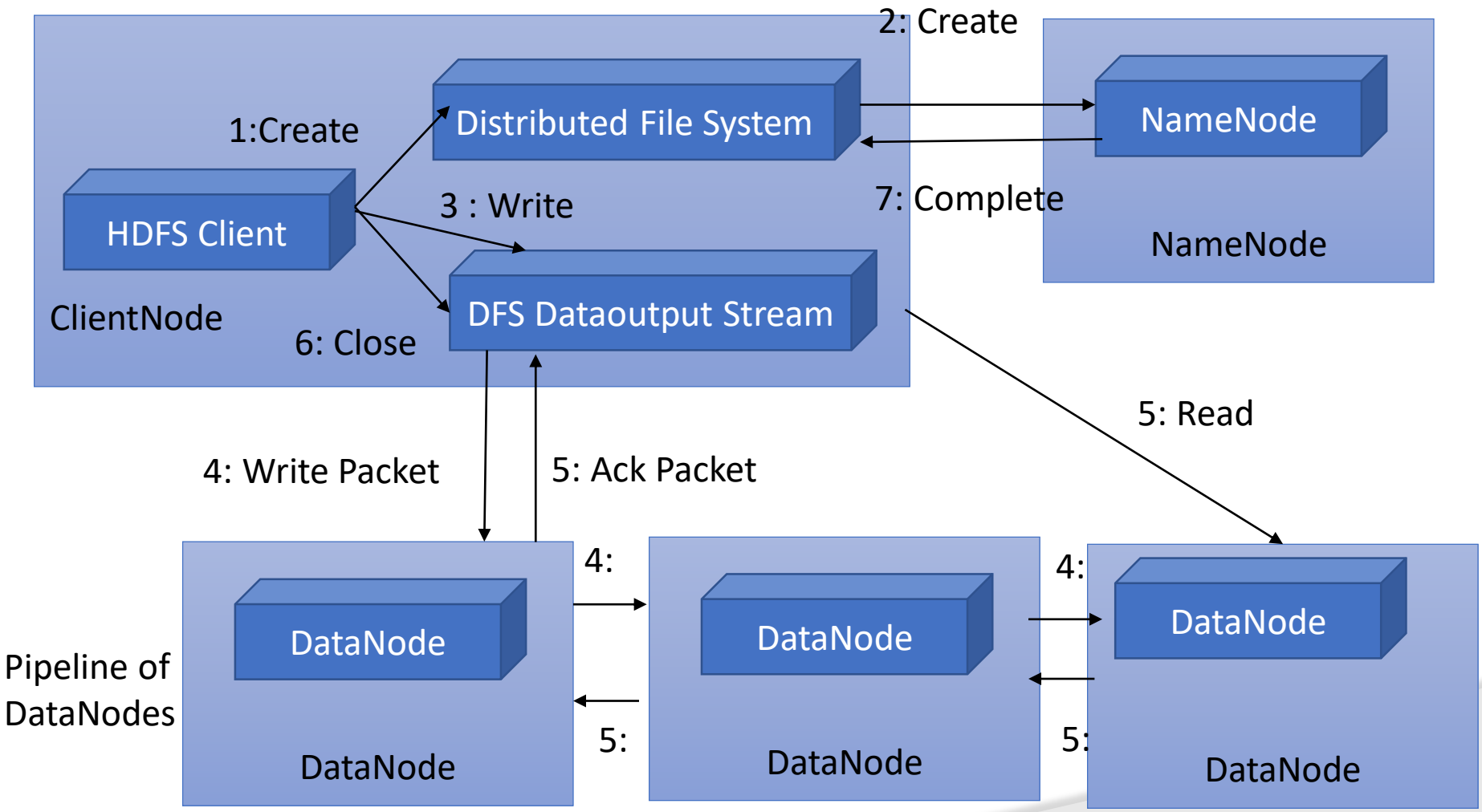
HDFS Daemons – Secondary NameNode

- The Secondary NameNode takes a snapshot of HDFS metadata at intervals specified in the Hadoop configuration.
- Since the memory requirements of Secondary NameNode are the same as NameNode, it is better to run NameNode and Secondary Name Node on different machines.
- In case of failure of the NameNode, the Secondary NameNode can be configured manually to bring up the cluster.
- However, the Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.

Anatomy of File Read



Anatomy of File Write



Hadoop Replica Placement Strategy

- Hadoop Replica Placement Strategy, first replica is placed on the same node as the client.
- Then it places second replica on a node that is present on different rack.
- It places the third replica on the same rack as second, but on a different node in the rack.
- Once replica locations have been set, a pipeline is built.
- This strategy provides good reliability.

UNIT 2 - INTRODUCTION TO HADOOP

- Working with HDFS Commands
- Interacting with Hadoop Ecosystem

Hadoop Commands

ls	*Lists all files with permissions*
mkdir	*Create a directory*
rm	*Remove file or directory*
put	*Store file/folder from local disk to HDFS*
Get	*Store file/folder from HDFS to local disk*
Cat	*Display the file content*
count	*No. of directory, no. of files and file size*
Cp	*From one directory to another on HDFS*
copyFromLocal	*copy a file from local file system to HDFS*
copyToLocal	*copy a file from HDFS to local disk*

Hadoop Commands

Objective: To get the list of directories and files at the root of HDFS
Act:

hadoop fs -1s /

Objective: To get the list of complete directories and files of HDFS.
Act

hadoop fs -ls -R /

Hadoop Commands(cont..)

Objective: To create a directory (say, sample) in HDFS.
Act

hadoop mkdir /sample

Objective: To copy a file from local file system to HDFS.
Act

hadoop fs -put /root/sample/test.txt /sample/test.txt

Hadoop Commands(cont..)

Objective: To copy a file from HDFS to local file system
Act

hadoop fs -get /sample/test.txt /root/sample/testsample.txt

Objective To copy a file from local file system to HDFS via copyFromLocal command
Act

hadoop fs -copyFromLocal /root/sample/test.txt /sample/testsample.txt

Hadoop Commands(cont..)

Objective To copy a file from Hadoop file system to local file system via copyToLocal command
Act

hadoop fs - copyToLocal /sample/test.txt /root/sample/testsample1.txt

Objective To display the content of an HDFS file on console
Act

hadoop fs -cat /sample/test.txt

Hadoop Commands(cont..)

Objective To copy a file from one directory to another on HDFS
Act

hadoop fs – cp /sample/test.txt /sample1

Objective To remove a directory from HDFS
Act

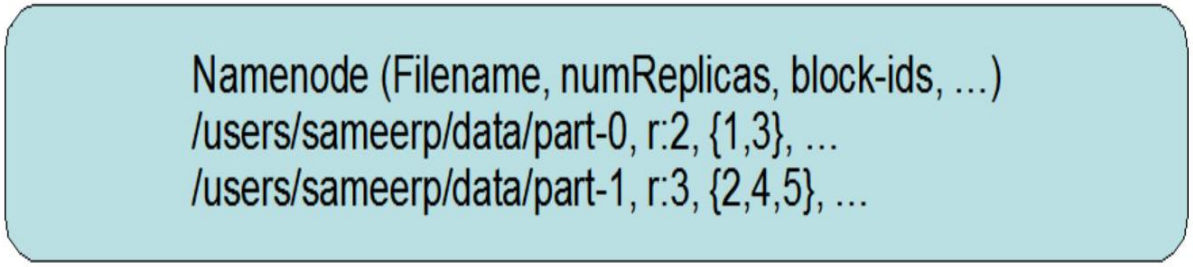
hadoop fs-rm-r /sample1

Special features of Hadoop

1. Data Replication:

- There is absolutely no need for a client application to track all blocks.
- It directs the client to the nearest replica to ensure high performance.

Block Replication



Special features of Hadoop

2. Data Pipeline:

- A client application writes a block to the first DataNode in the pipeline.
- DataNode takes over and forwards the data to the next node in the pipeline.
- This process continues for all the data blocks, and subsequently all the replicas are written to the disk.

Videos on Apache Hadoop HDFS and YARN setup

https://www.youtube.com/watch?v=6yXre...

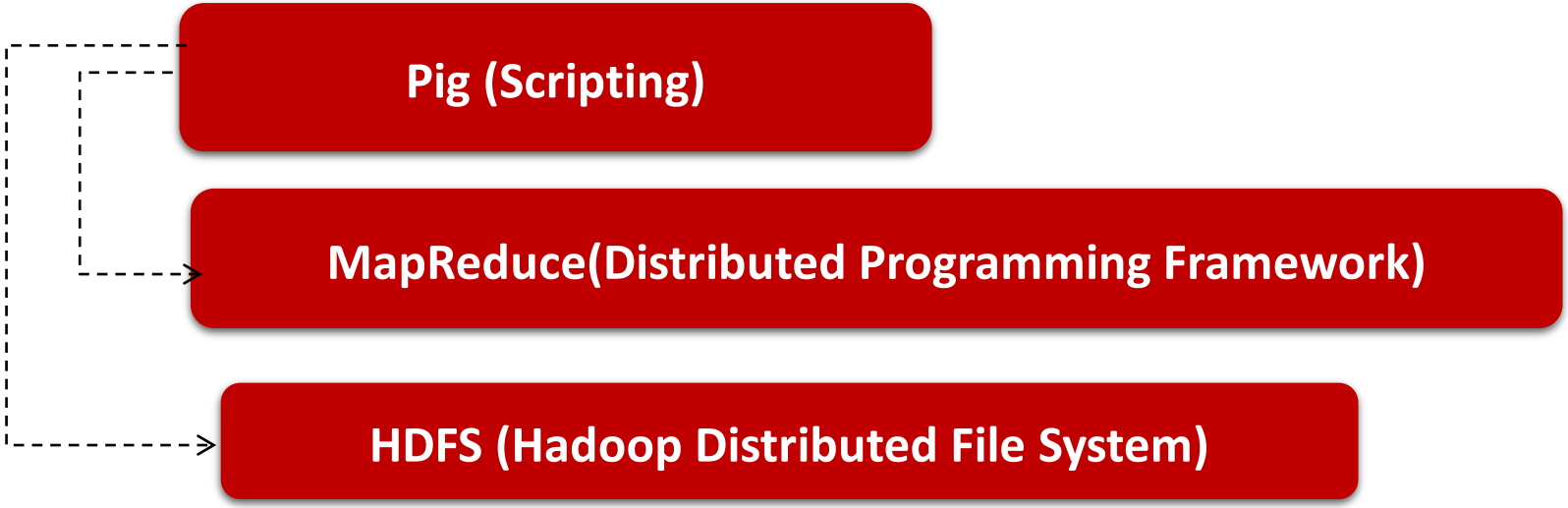
https://www.youtube.com/watch?v=EmD4r...

https://www.youtube.com/watch?v=tXMvn...

Apache Hadoop Familiarizing HDFS Commands

https://www.youtube.com/watch?v=qMcm23S4lpA

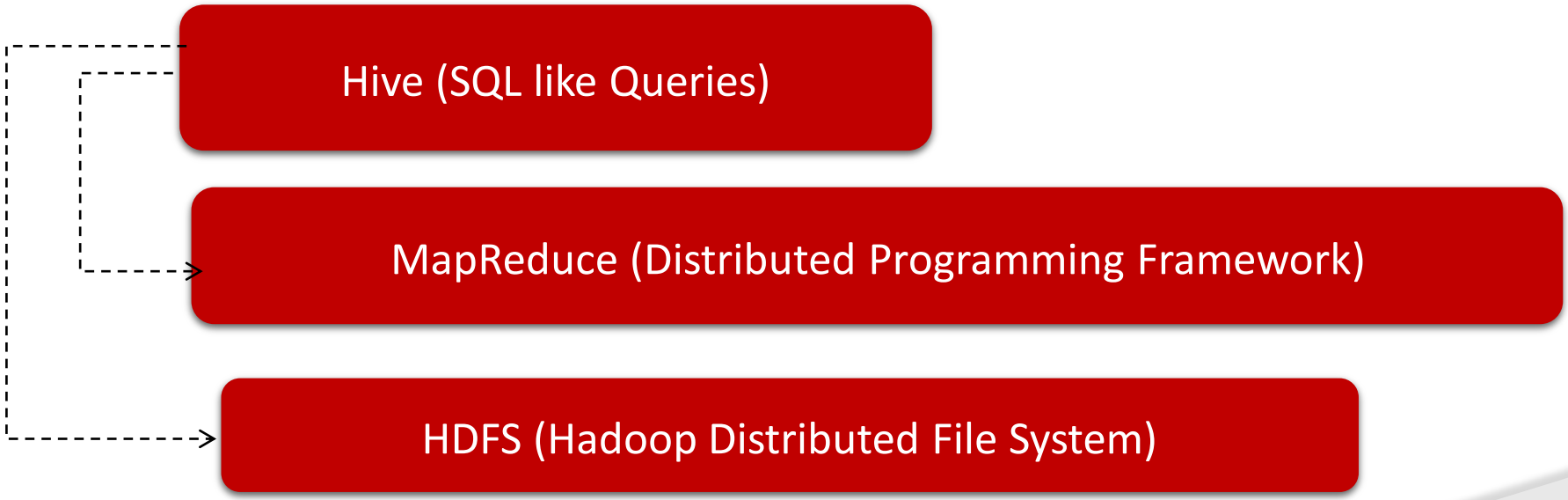
Interacting with Hadoop Ecosystem - PIG



PIG in the Hadoop Ecosystem

Interacting with Hadoop Ecosystem - HIVE

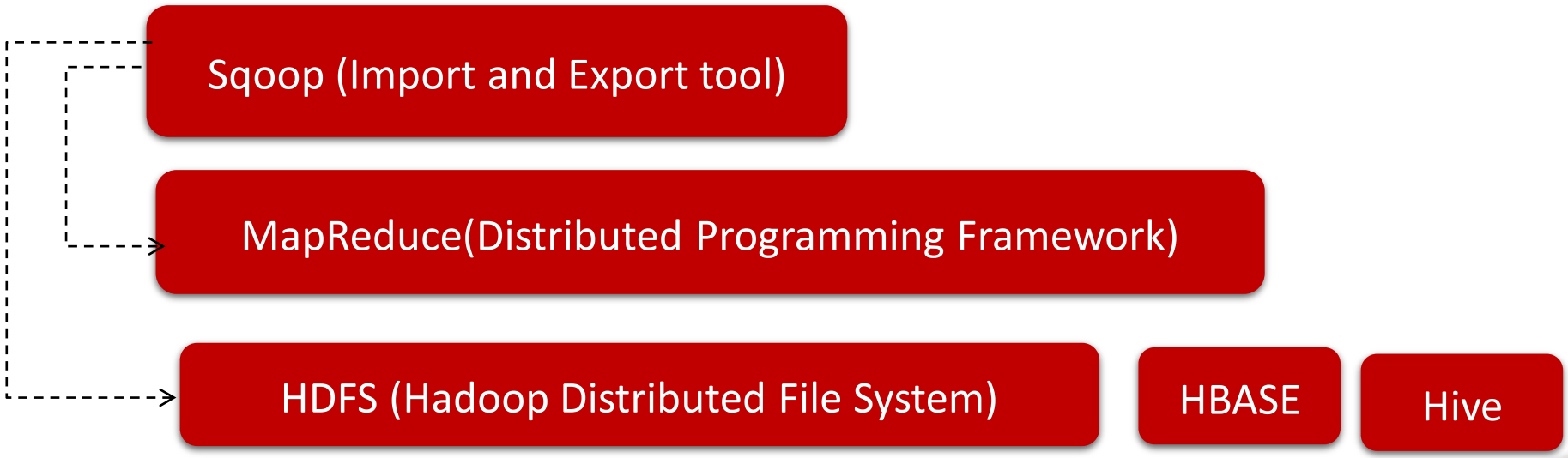
- Hive is a Data Warehousing Layer on top of Hadoop.
- Analysis and queries can be done using an SQL like language.
- Hive can be used to do ad-hoc queries, summarization, and data analysis.



Hive in the Hadoop Ecosystem

Interacting with Hadoop Ecosystem - Sqoop

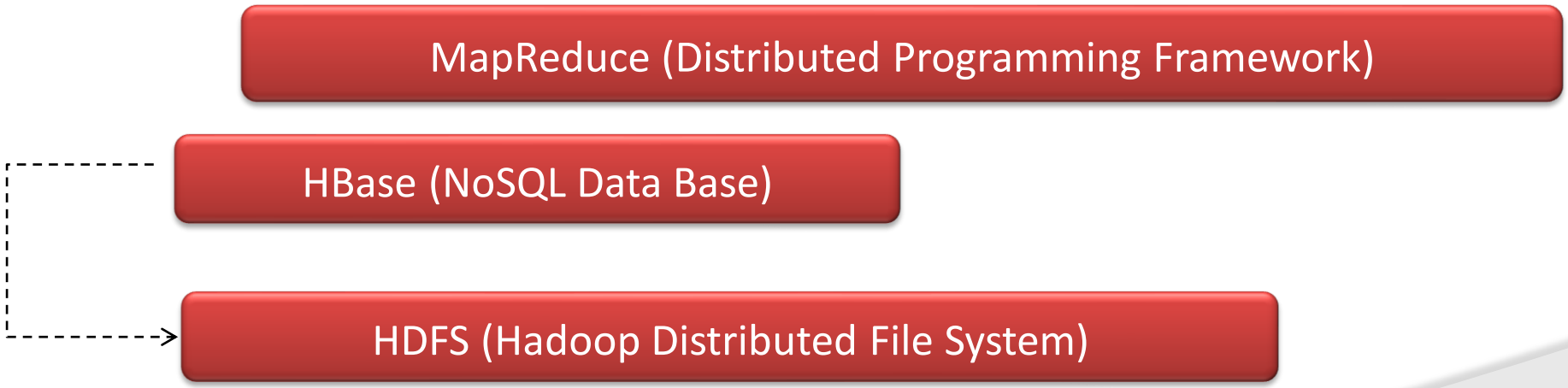
- Sqoop is a tool which helps to transfer data between Hadoop and Relational Databases.
- With the help of Sqoop, you can import data from RDBMS to HDFS and vice-versa.



Sqoop in the Hadoop Ecosystem

Interacting with Hadoop Ecosystem - HBase

- HBase is a column-oriented NoSQL database for hadoop
- Used to store billions of rows and millions of columns.
- Provides random read/write operation.
- Supports record level updates which is not possible using HDFS. HBase sits on top of HDFS.



HBase in the Hadoop Ecosystem

UNIT 2 - INTRODUCTION TO HADOOP

Processing Data with Hadoop

- MapReduce Programming
- Daemons
- Interaction
- Workflow
- Architecture
- Example

Data Processing - Hadoop

Hadoop Component – MapReduce

- MapReduce Programming is a software framework.
- Helps to process **massive amounts of data in parallel.**
- In MapReduce programming, the input dataset is split into independent chunks.
- **It has 2 major phases**
  - **Map Phase**
  - **Reduce Phase**

MapReduce Programming Phases and Daemons

Phases
Map: Converts input into Key Value pair
Reduce: Combines output of mappers and produces a reduced result set

Daemons:
JobTracker: Master. Schedules task
TaskTracker: Slave.executes task

- The **Map task** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The **Reduce task** takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

MapReduce Phases

Map tasks

- Process these independent chunks completely in a parallel manner.
- The output produced by the map tasks serves as intermediate data and is stored on the local disk of that server.
- The output of the mappers are automatically shuttled and sorted by the framework.
- MapReduce Framework **sorts the output based on keys**.
- This sorted output becomes the input to the reduce tasks.

MapReduce Phases

Reduce task

- Provides reduced output by combining the out put of the various mappers.
- Job inputs and outputs are stored in a file system.
- MapReduce framework also takes care of the other tasks such as scheduling, monitoring, re-executing failed tasks, etc.

MapReduce Programming Phases and Daemons

Phases
Map: Converts input into Key Value pair
Reduce: Combines output of mappers and produces a reduced result set

Daemons:
JobTracker: Master. Schedules task
TaskTracker: Slave.executes task

MapReduce Daemons

MapReduce Daemons

- **Job Tracker**
- **Task Tracker**

MapReduce Daemons

JobTrackers

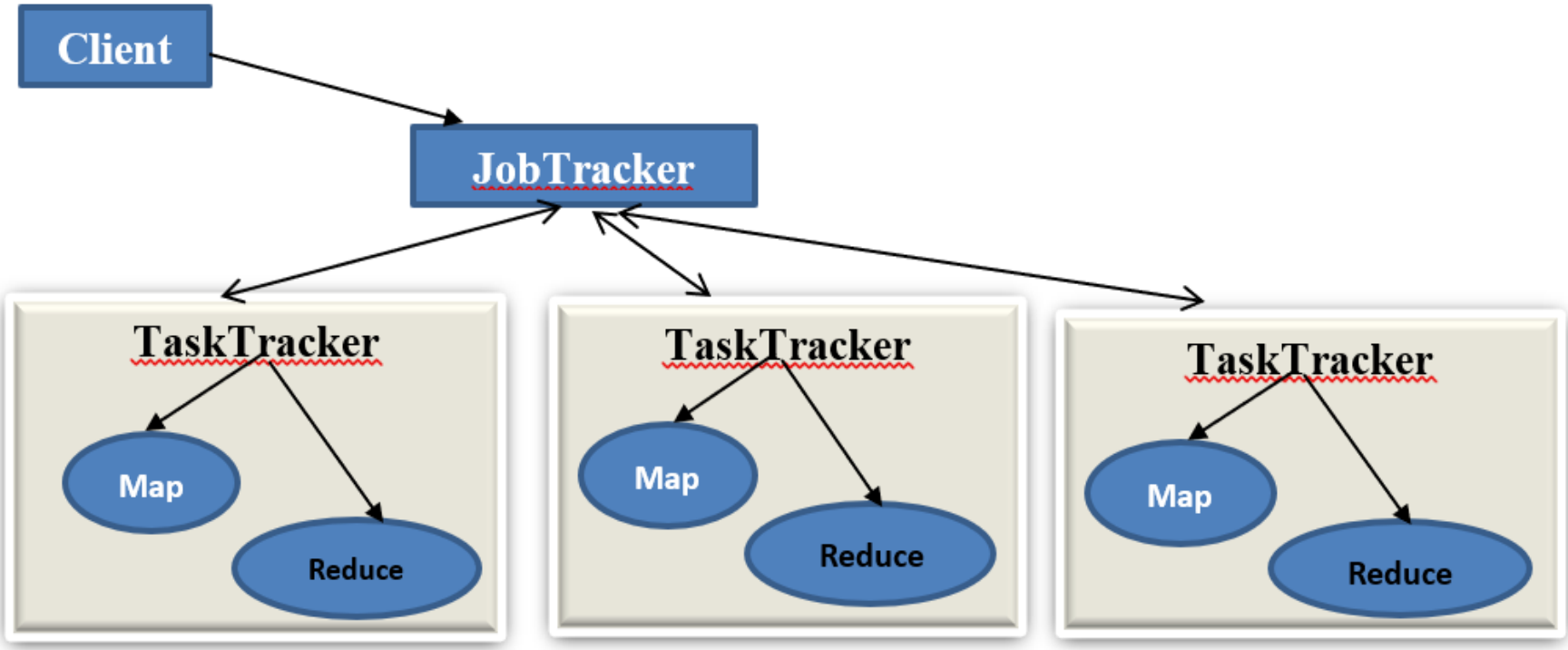
- Provides connectivity between Hadoop and client application.
- When code submit to cluster, JobTracker creates the execution plan by deciding which task to assign to which node.
- It also monitors all the running tasks.
- When a task fails, it automatically re-schedules the task to a different node after a predefined number of retries.
- JobTracker is a master daemon responsible for executing overall MapReduce job.
- **There is a single JobTracker per Hadoop cluster.**

MapReduce Daemons

TaskTrackers

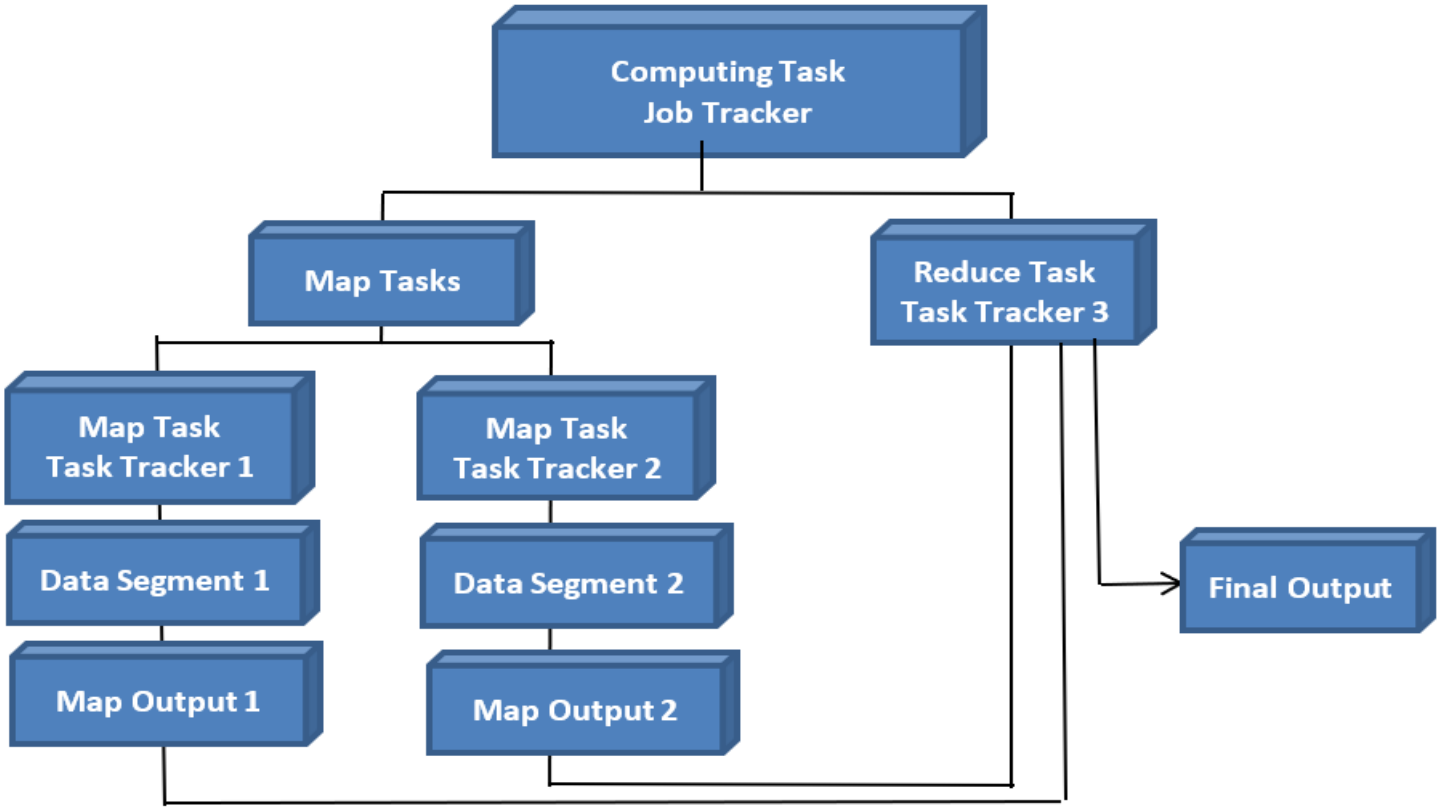
- A single TaskTracker per slave and spawns multiple Java Virtual Machines (VM) to handle multiple map of reduce tasks in parallel.
- TaskTracker continuously sends heartbeat measure to JobTracker.
- When the JobTracker fails to receive a heartbeat from a Task Tracker,
  - The JobTraker assumes that the TaskTracker has failed and
  - Resubmits the task to another available node in the cluster.
- Once the client submits a job to the Job Tracker, it partitions and assigns diverse MapReduce task for each TaskTracker in the cluster

JobTracker and TaskTracker Interactions



- MapReduce divides a data analysis task into two parts
- Map
- Reduce

MapReduce Programming Workflow



Example: Two mappers and one reducer

- Each mapper works on the partial dataset, that is stored in that node
- Reducer combines the output from the mappers to produce reduce result set

Steps to Perform Task in MapReduce

Step 1. The input dataset is split into multiple pieces of data (several small subsets)

Step 2. The framework creates a master and several workers processes and executes the worker processes remotely.

Step 3. Several map tasks work simultaneously and read pieces of data that were assigned to each map task.
The 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.

Step 4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of the specified mapper.

Steps to Perform Task in MapReduce

Step 5. When the map workers complete their work, the master instructs the reduce workers to begin their Work. The reduce workers in turn contact the map workers to get the key/value data tor their partition. The data thus received is shuflled and sorted as per keys.

Step 6. Then it calls reduce function for every unique key. This function writes the output to the file.

Step 7. When all the reduce workers complete their work, the master transfers the control to the user program.

SQL vs MapReduce

	SQL	MapReduce
Access	Interactive and Batch	Batch
Structure	Static	Dynamic
Updates	Read and Write many times	Write once and Read many times
Integrity	High	Low
Scalability	Nonlinear	Linear

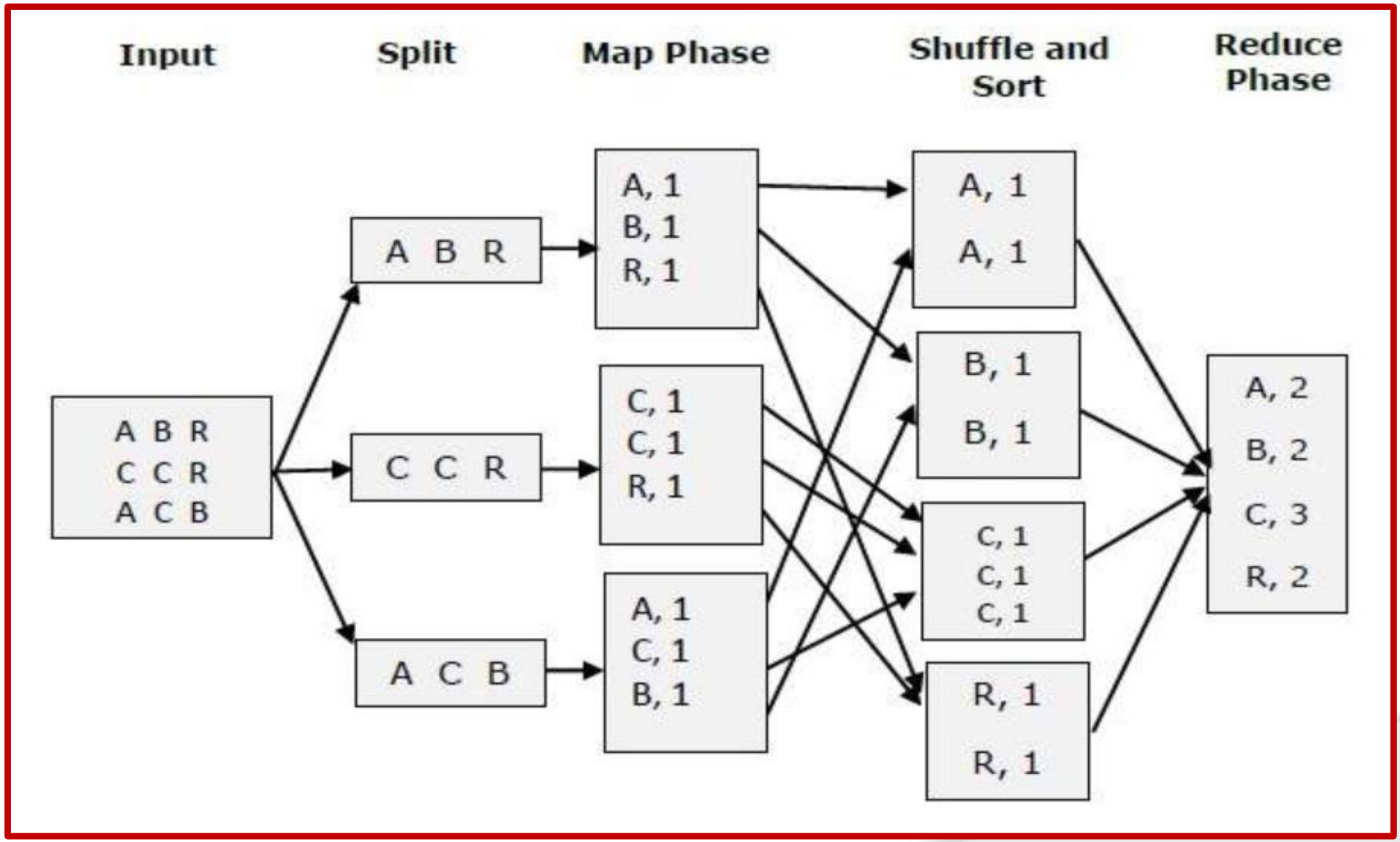
HDFS - MapReduce Phases

- **Input Phase** – A Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map** – Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** – They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** – A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.

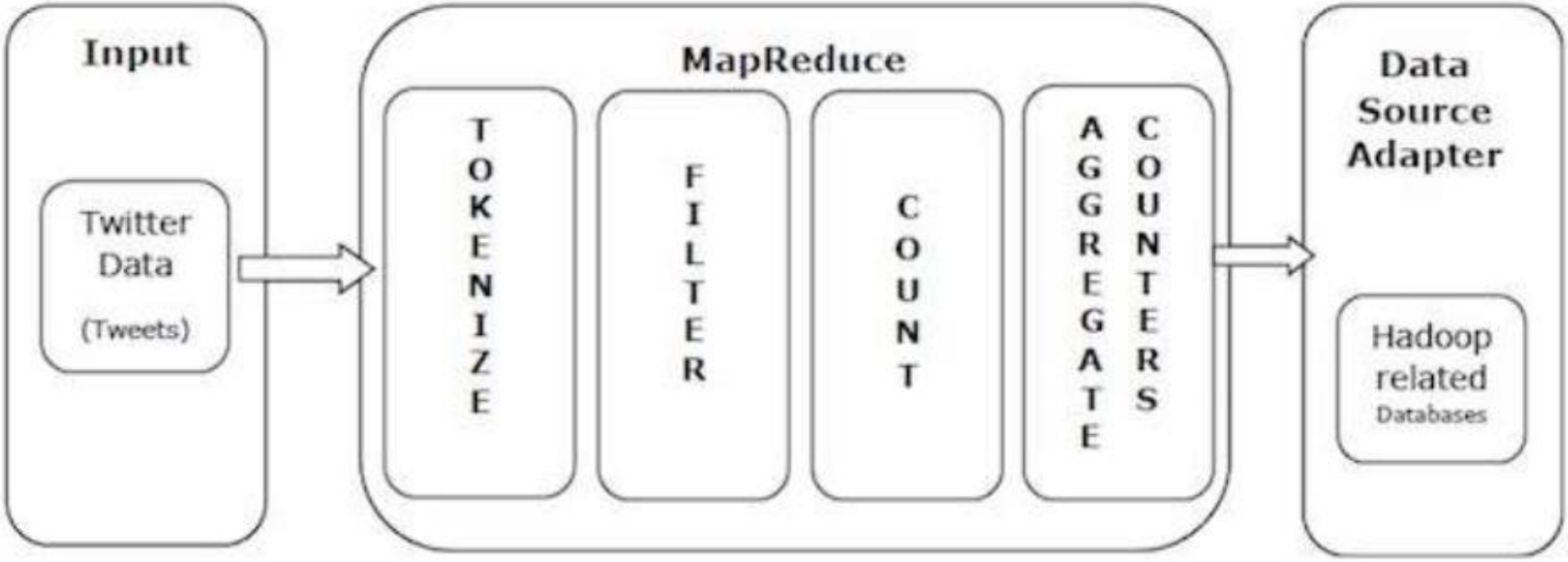
MapReduce Phases

- **Shuffle and Sort** – The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** – The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing.
- **Output Phase** – In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

MapReduce Example



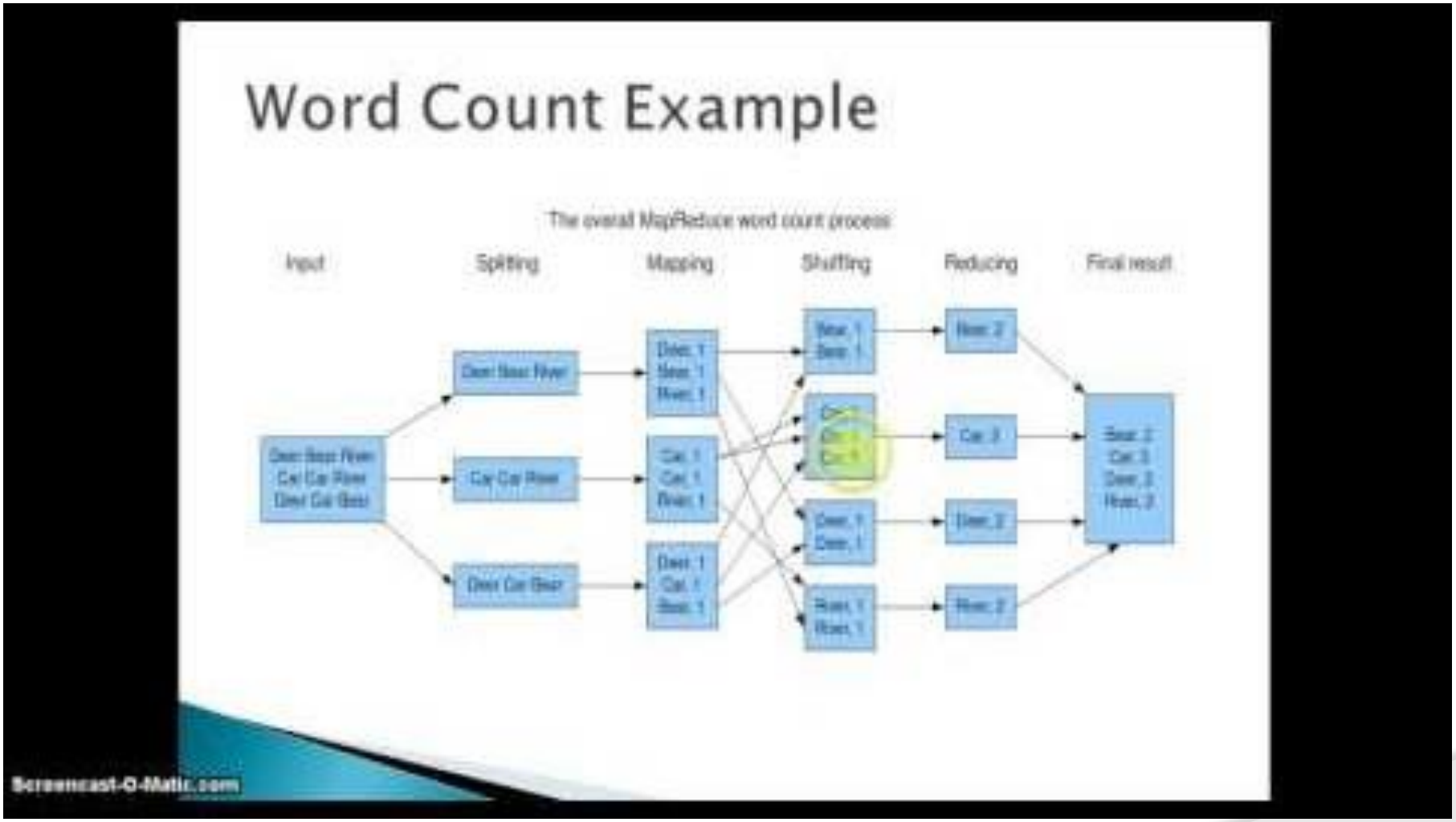
MapReduce – Real Time Example



A real-world example: Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second.

The above illustration shows how Tweeter manages its tweets with the help of MapReduce.

MapReduce - Word count Example



Mapreduce Example: https://www.youtube.com/watch?v=3LQAaAh4wM8

UNIT-III:THE HADOOP DISTRIBUTED FILE SYSTEM

Topics

- The Design of HDFS
- HDFS Concepts
- Basic Filesystem Operations
- Hadoop Filesystems
- The Java Interface
- Reading Data from a Hadoop URL
- Reading Data Using the Filesystem API
- Writing Data
- Data Flow- Anatomy of a File Read
- Anatomy of a File Write
- Limitations.

HDFS Concepts

Blocks

- A disk has a block size, which is the minimum amount of data that it can read or write.
- Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size.
- Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes.
- Generally transparent to the filesystem user who is simply reading or writing a file—of whatever length.
- Tools to perform filesystem maintenance, such as
- df and fsck, that operate on the filesystem block level.
- HDFS has a block size of —64 MB by default.
- Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units.

HDFS Concepts - Blocks

A block abstraction for a distributed filesystem brings several benefits.

- The most obvious: a file can be larger than any single disk in the network.
- Blocks fit well with replication for providing fault tolerance and availability

HDFS's fsck command understands blocks.
For example, running:
% hadoop fsck / -files -blocks
will list the blocks that make up each file in the filesystem.

HDFS Concepts - Namenodes and Datanodes

An HDFS cluster has two types of node operating in a master-worker pattern:

- a namenode (the master) and
- a number of datanodes (workers).

Namenode

- Manages the filesystem namespace.
- Maintains the filesystem tree and the metadata for all the files and directories in the tree.

Stored on the local disk in the form of two files:

- the namespace image and
- the edit log.

knows the datanodes on which all the blocks for a given file are located (reconstructed from datanodes when the system starts.)

HDFS Concepts - Namenodes and Datanodes

Datanodes

Datanodes are the workhorses of the filesystem.

- They store and retrieve blocks when they are told to (by clients or the namenode)
- Report back to the namenode periodically with lists of blocks that they are storing.

Failure of namenode leads to:

- Without the namenode, the filesystem cannot be used.
- In fact, if the machine running the namenode were distroyed, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes.

HDFS Concepts - Namenodes and Datanodes

Make the namenode resilient to failure,
Hadoop provides two mechanisms for this.

1. **Back up the files** that make up the persistent state of the filesystem metadata
   The namenode writes its persistent state to multiple filesystems.
   These writes are synchronous and atomic.
   To write to local disk as well as a remote NFS mount.
2. **Run a secondary namenode**
   which despite its name does not act as a namenode.
   Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large.
   The secondary namenode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge.

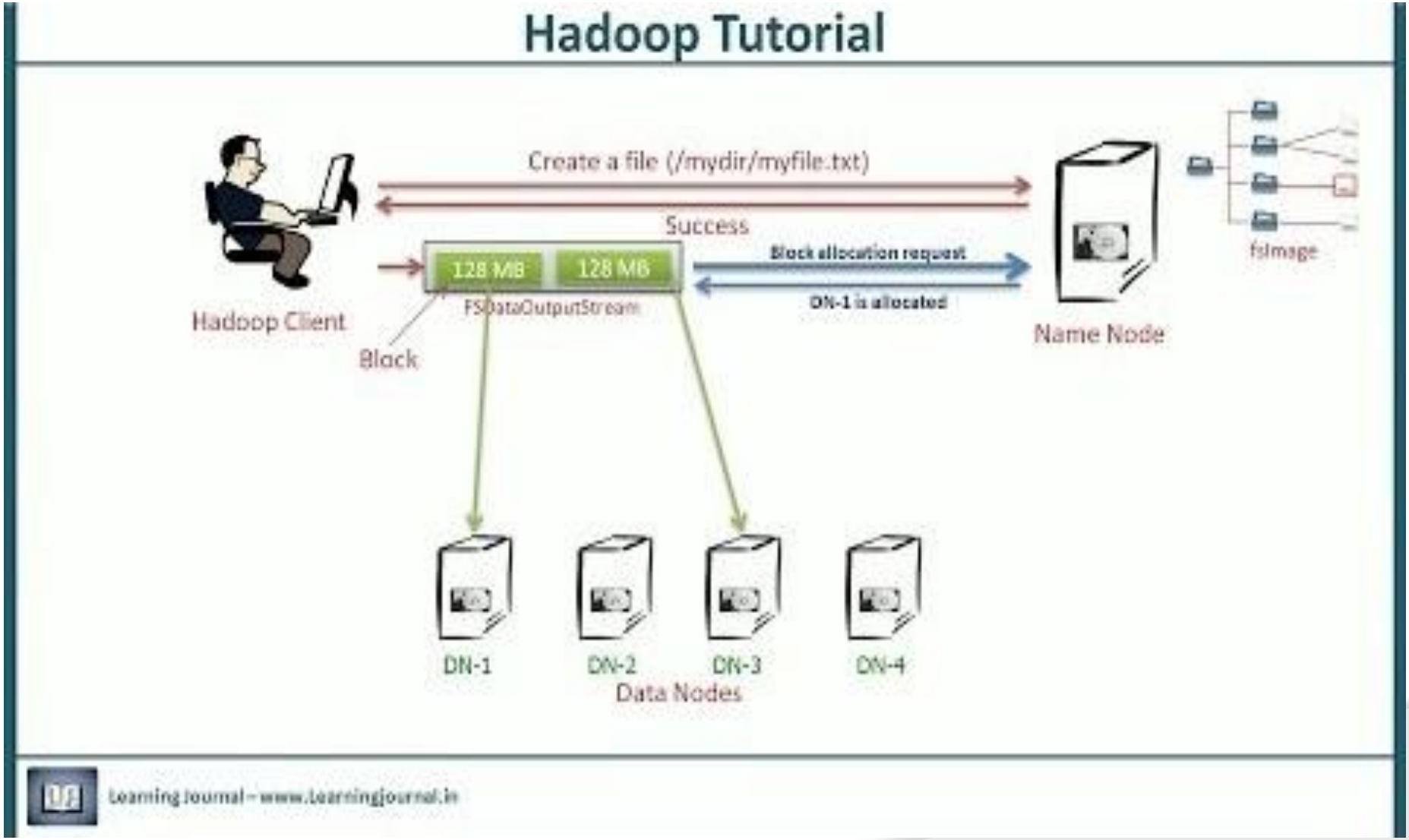
HDFS Concepts - HDFS Federation

- The namenode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling.
- HDFS Federation, introduced in the 0.23 release series, allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace.
- For example, one namenode might manage all the files rooted under /user, say, and a second namenode might handle files under /share.
- Under federation, each namenode manages a namespace volume, which is made up of the metadata for the namespace, and a block pool containing all the blocks for the files in the namespace.

HDFS Concepts - HDFS Federation

- Namespace volumes are independent of each other, which means namenodes do not communicate with one another,
- and furthermore the failure of one namenode does not affect the availability of the namespaces managed by other namenodes.
- Block pool storage is not partitioned, so datanodes register with each namenode in the cluster and store blocks from multiple block pools.
- To access a federated HDFS cluster, clients use client-side mount tables to map file paths to namenodes.
- This is managed in configuration using the ViewFileSystem, and viewfs:// URIs.

HDFS Architecture : https://www.youtube.com/watch?v=ECxlFPy7s-k



HDFS Concepts - HDFS High-Availability

- The combination of replicating namenode metadata on multiple filesystems and using the secondary namenode to create checkpoints protects against data loss but does not provide high-availability of the filesystem.
- The namenode is still a single point of failure (SPOF), since if it did fail, all clients - including MapReduce jobs—would be unable to read, write, or list files, because the namenode is the sole repository of the metadata and the file-to-block mapping.
- 0.23 release series of Hadoop remedies this situation by adding support for HDFS high-availability (HA).
- The implementation : A pair of namenodes in an active standby configuration.
- In the event of the failure of the active namenode, the standby takes over its duties to continue servicing client requests without a significant interruption.

HDFS Concepts - HDFS High-Availability

Architectural changes :

- The namenodes must use highly-available shared storage to share the edit log.

(In the initial implementation of HA this will require an NFS filer, but in future releases more options like ZooKeeper.)

When a standby namenode comes up it reads up to the end of the shared edit log to synchronize its state with the active namenode, and then continues to read new entries as they are written by the active namenode.

- Datanodes must send block reports to both namenodes since the block mappings are stored in a namenode's memory, and not on disk.
- Clients must be configured to handle namenode failover, which uses a mechanism that is transparent to users.

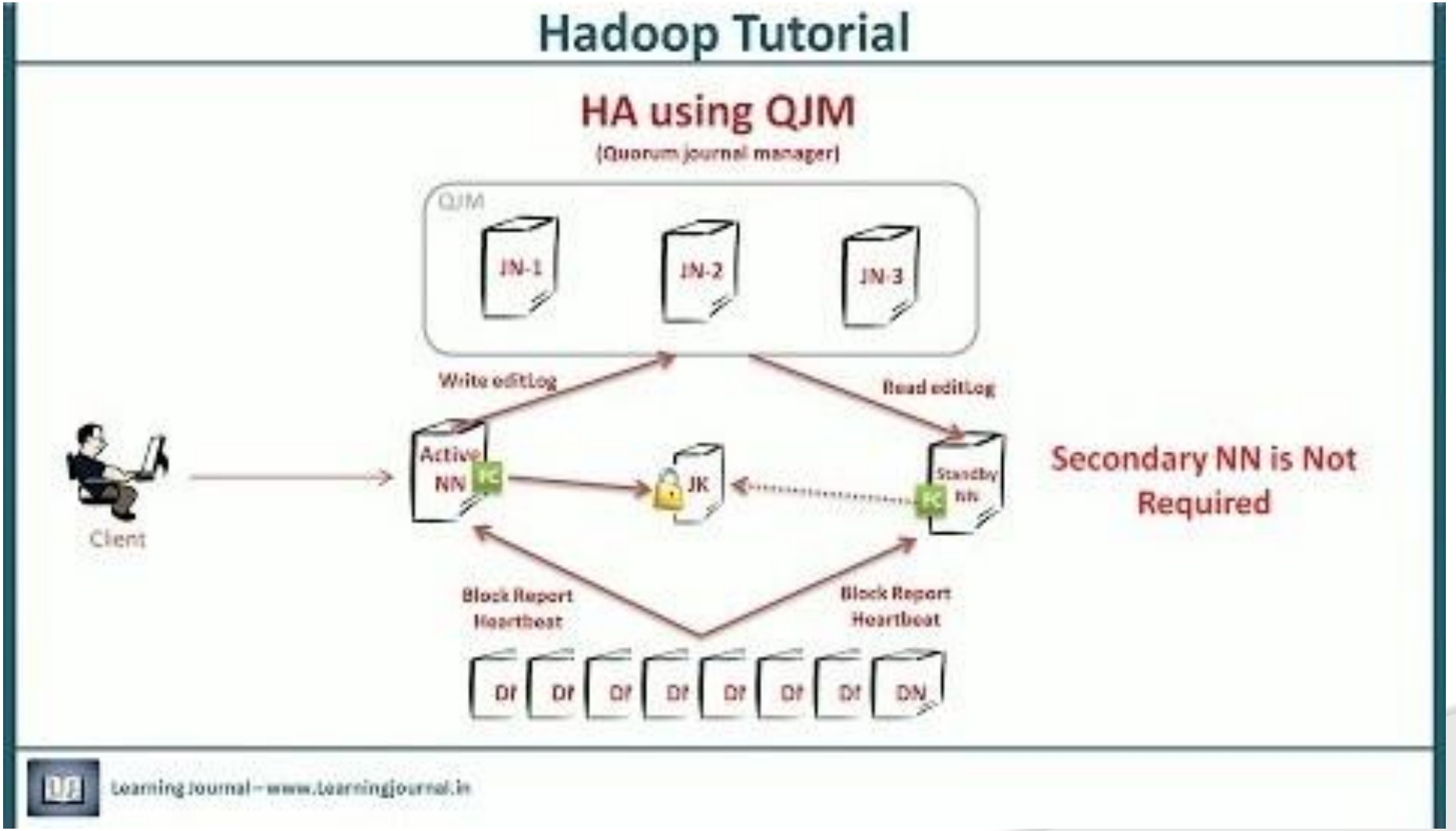
HDFS Concepts - Failover and fencing

- The transition from the active namenode to the standby is managed by failover controller.
- Failover controllers are pluggable, but the first implementation uses ZooKeeper to ensure that only one namenode is active.
- Each namenode runs a lightweight failover controller process whose job it is to monitor its namenode for failures (using a simple heartbeating mechanism) and trigger a failover should a namenode fail.
- Failover may also be initiated manually by an adminstrator, in the case of routine maintenance,
- This is known as a graceful failover, since the failover controller arranges an orderly transition for both namenodes to switch roles.

HDFS Concepts - Failover and fencing

- The HA implementation goes to great lengths to ensure that the previously active namenode is prevented from doing any damage and causing corruption—a method known as fencing.
- Fencing mechanisms,
  - **Killing the namenode's process**, revoking its access to the shared storage directory (typically by using a vendor-specific NFS command), and
  - **Disabling its network port** via a remote management command.

HDFS - https://www.youtube.com/watch?v=5U6EIv_w5Bg



HDFS Concepts - Hadoop Filesystems

- Hadoop has an abstract notion of filesystem, of which HDFS is just one implementation.
- The Java abstract class org.apache.hadoop.fs.FileSystem represents a filesystem in Hadoop, and there are several concrete implementations.
- Hadoop provides many interfaces to its filesystems, and it generally uses the URI scheme to pick the correct filesystem instance to communicate with.

HDFS Concepts - Hadoop Filesystems

File System	URI scheme	Java implementation org.apache.hadoop	Description
Local	file	fs.LocalFileSystem	A filesystem for a locally connected disk with clientside checksums. **Use RawLocalFileSystem for a local filesystem with no checksums.**
HDFS	hdfs	hdfs. DistributedFileSystem	Hadoop's distributed filesystem. HDFS is designed to **work efficiently in conjunction with MapReduce**
HFTP	hftp	hdfs.HftpFileSystem	A filesystem providing read-only access to HDFS over HTTP. (Despite its name, HFTP has no connection with FTP.) **Used with distcp to copy data between HDFS clusters running different versions.**

HDFS Concepts - Hadoop Filesystems

File System	URI scheme	Java implementation org.apache.hadoop	Description
HSFTP	hsftp	hdfs.HsftpFileSystem	A filesystem providing read-only access to HDFS over HTTPS. (Again, this has no connection with FTP.)
Web HDFS	webhdfs	hdfs.web.WebHdfsFile	A filesystem providing secure read-write access to HDFS over HTTP. **WebHDFS is intended as a replacement for HFTP and HSFTP.**
HAR	har	fs.HarFileSystem	A filesystem layered on another filesystem for archiving files. **Hadoop Archives are typically used for archiving files in HDFS to reduce the namenode's memory usage.**

HDFS Concepts - Hadoop Filesystems

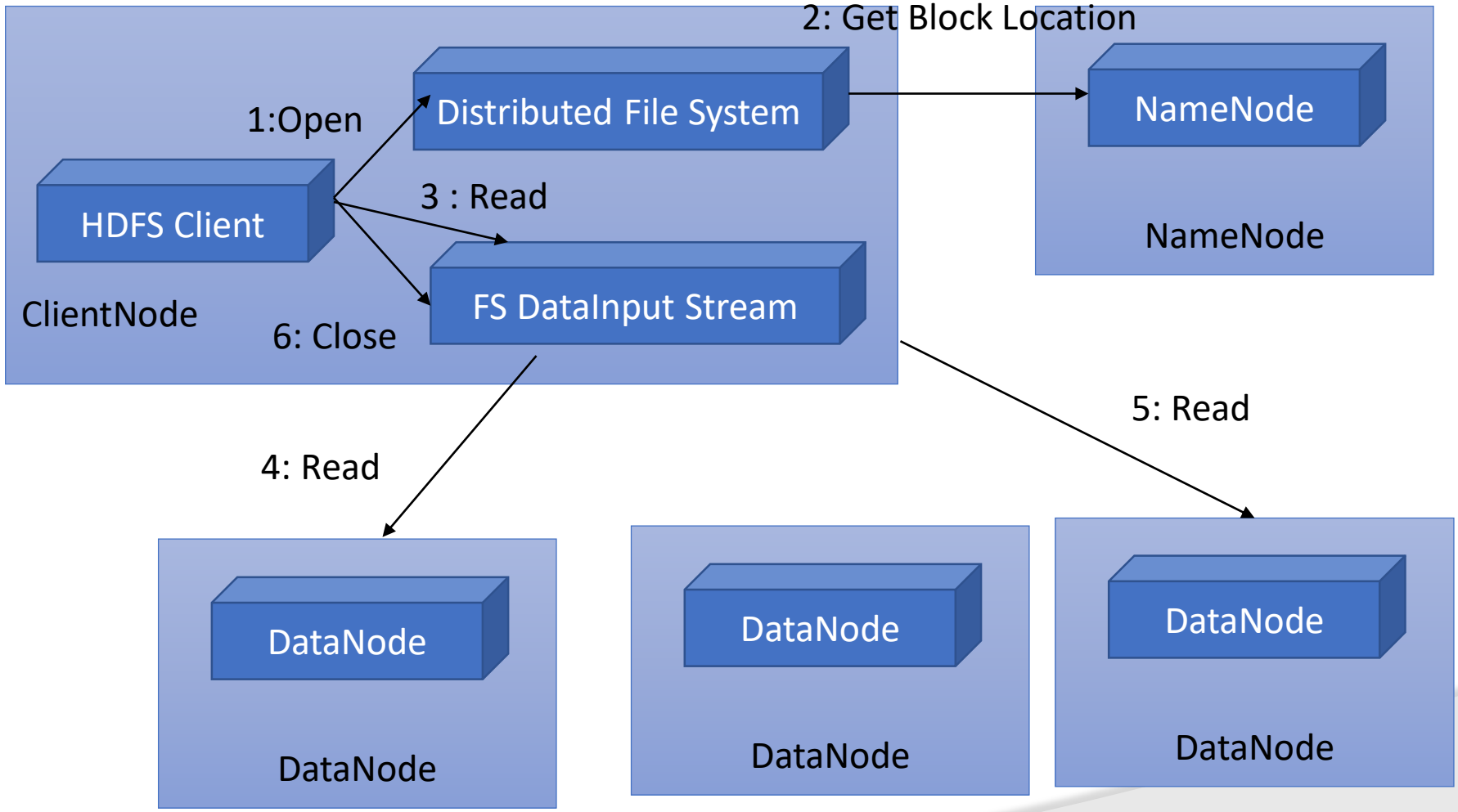
Filesystem	URI scheme	Java implementation org.apache.hadoop	Description
KFS (CloudStore)	kfs	fs.kfs. KosmosFileSystem	CloudStore (formerly Kosmos filesystem) is a distributed filesystem like HDFS or Google's GFS, written in C++.
FTP	ftp	fs.ftp.FTPFileSystem	Filesystem backed by an FTP server.
S3 (native)	s3n	fs.s3native.	NativeS3FileSystem A filesystem backed by Amazon S3. See http://wiki.apache.org/hadoop/AmazonS3
S3 (blockbased)	s3	fs.s3.S3FileSystem	A filesystem backed by Amazon S3, which stores files in blocks (much like HDFS) to overcome S3's 5 GB file size limit.

HDFS Concepts - Hadoop Filesystems

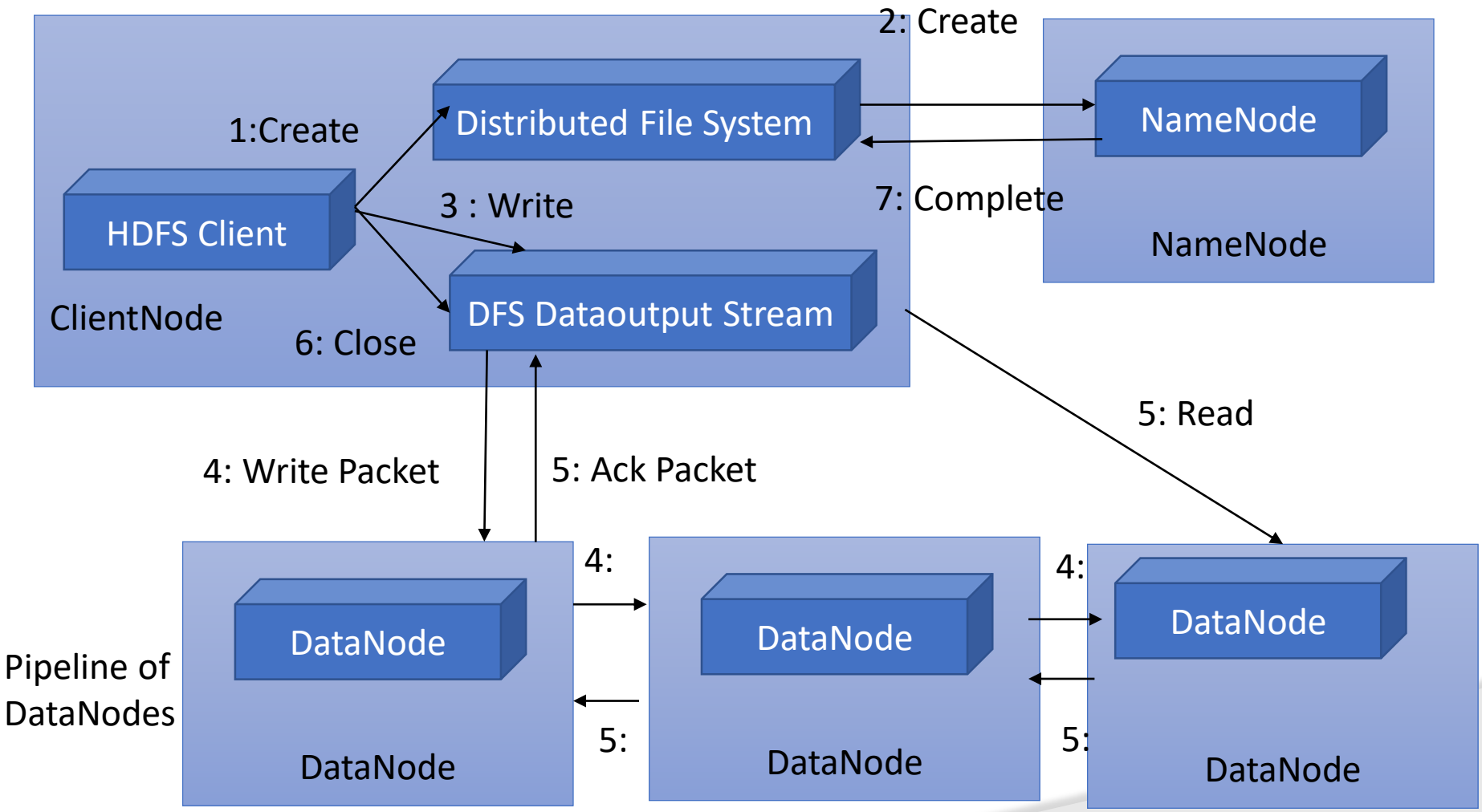
Filesystem	URI scheme	Java implementation org.apache.hadoop	Description
Distributed RAID	hdfs	hdfs.DistributedRaidFi	A "RAID" version of HDFS designed for archival storage. For each file in HDFS, a (smaller) parity file is created, which allows the HDFS replication to be reduced from three to two, which reduces disk usage by 25% to 30%, while keeping the probability of data loss the same. Distributed RAID requires that, run a RaidNode daemon on the cluster.
View	viewfs	viewfs.ViewFileSystem	A client-side mount table for other Hadoop filesystems. Commonly used to create mount points for federated Namenodes

- HDFS (HADOOP DISTRIBUTED FILE SYSTEM)
- Architecture
- DataNode
- Secondary NameNode
- Anatomy of File Read
- Anatomy of File Write

Anatomy of File Read



Anatomy of File Write



Hadoop Replica Placement Strategy

- Hadoop Replica Placement Strategy, first replica is placed on the same node as the client.
- Then it places second replica on a node that is present on different rack.
- It places the third replica on the same rack as second, but on a different node in the rack.
- Once replica locations have been set, a pipeline is built.
- This strategy provides good reliability.

Hadoop Commands

Objective: To get the list of directories and files at the root of HDES
Act:
hadoop fi-1s /

Objective: To get the list of complete directories and files of HDES.
Act
hadoop fs -ls -R /

Objective: To create a directory (say, sample) in HDES.
Act
hadoop mkdir /sample

Objective: To copy a file from local file system to HDES.
Act
Hadoop fs -gut /root/sample/test.vt /sample/test.Nt

Objective: To copy a file from HDFS to local file system
Act
hadoop fs -get /samplel/test.txt /root/sample/testsample.txt

UNIT -IV

UNDERSTANDING MAP REDUCE FUNDAMENTALS

- Map Reduce Framework: Exploring the features of Map Reduce
- Working of Map Reduce
- Exploring Map and Reduce Functions
- Techniques to optimize Map Reduce jobs
- Uses of Map Reduce
- Controlling MapReduce Execution with InputFormat
- Reading Data with custom RecordReader
- Reader, Writer, Combiner, Partitioners
- Map Reduce Phases
- Developing simple MapReduce Application

Data Processing - Hadoop

Hadoop Component – MapReduce

- MapReduce Programming is a software framework.
- Helps to process **massive amounts of data in parallel.**
- In MapReduce programming, the input dataset is split into independent chunks.
- **It has 2 major phases**
  - **Map Phase**
  - **Reduce Phase**

MapReduce Framework

Phases
Map: Converts input into Key Value pair
Reduce: Combines output of mappers and produces a reduced result set

Daemons:
JobTracker: Master. Schedules task
TaskTracker: Slave.executes task

- The **Map task** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The **Reduce task** takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

MapReduce Phases

Map tasks

- Process these independent chunks completely in a parallel manner.
- The output produced by the map tasks serves as intermediate data and is stored on the local disk of that server.
- The output of the mappers are automatically shuttled and sorted by the framework.
- MapReduce Framework **sorts the output based on keys**.
- This sorted output becomes the input to the reduce tasks.

MapReduce Phases

Reduce task

- Provides reduced output by combining the out put of the various mappers.
- Job inputs and outputs are stored in a file system.
- MapReduce framework also takes care of the other tasks such as scheduling, monitoring, re-executing failed tasks, etc.

MapReduce Programming Phases and Daemons

Phases
Map: Converts input into Key Value pair
Reduce: Combines output of mappers and produces a reduced result set

Daemons:
JobTracker: Master. Schedules task
TaskTracker: Slave.executes task

MapReduce Daemons

MapReduce Daemons

- **Job Tracker**
- **Task Tracker**

MapReduce Daemons

JobTrackers

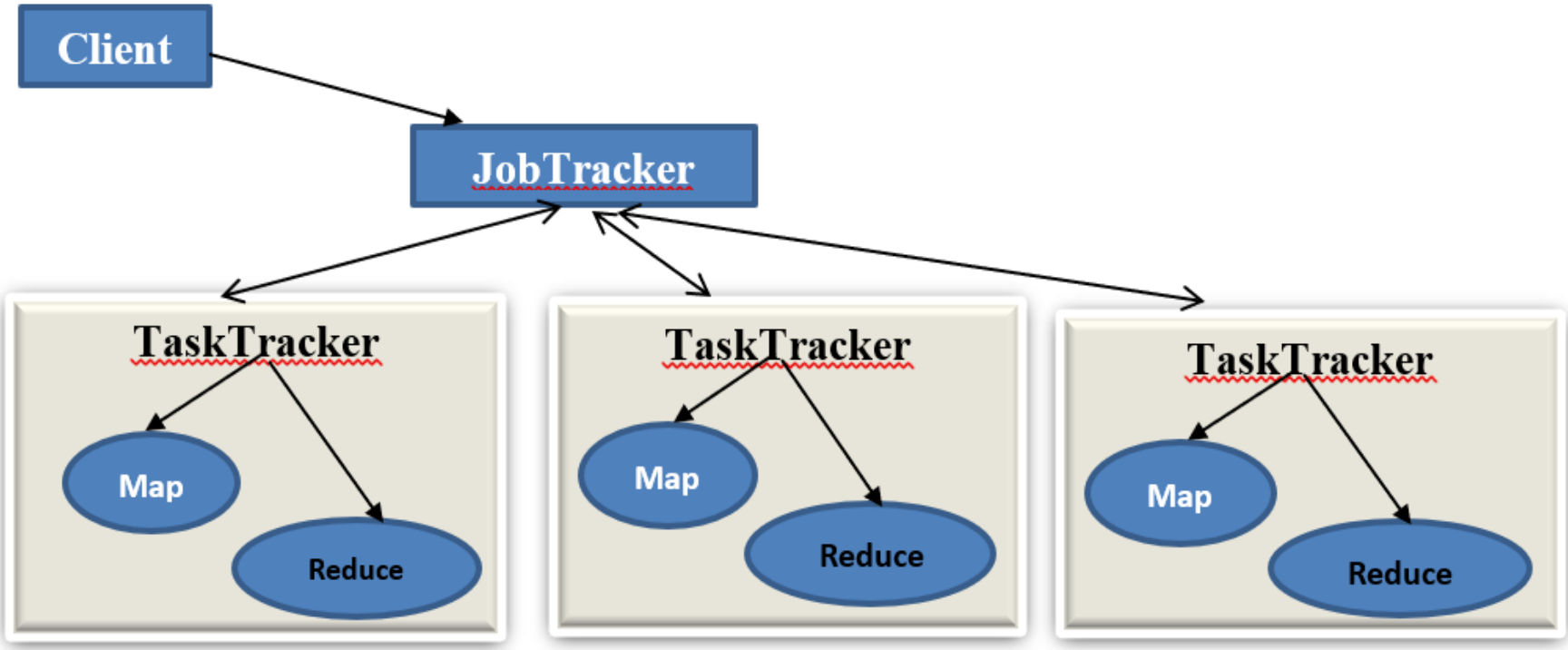
- Provides connectivity between Hadoop and client application.
- When code submit to cluster, JobTracker creates the execution plan by deciding which task to assign to which node.
- It also monitors all the running tasks.
- When a task fails, it automatically re-schedules the task to a different node after a predefined number of retries.
- JobTracker is a master daemon responsible for executing overall MapReduce job.
- **There is a single JobTracker per Hadoop cluster.**

MapReduce Daemons

TaskTrackers

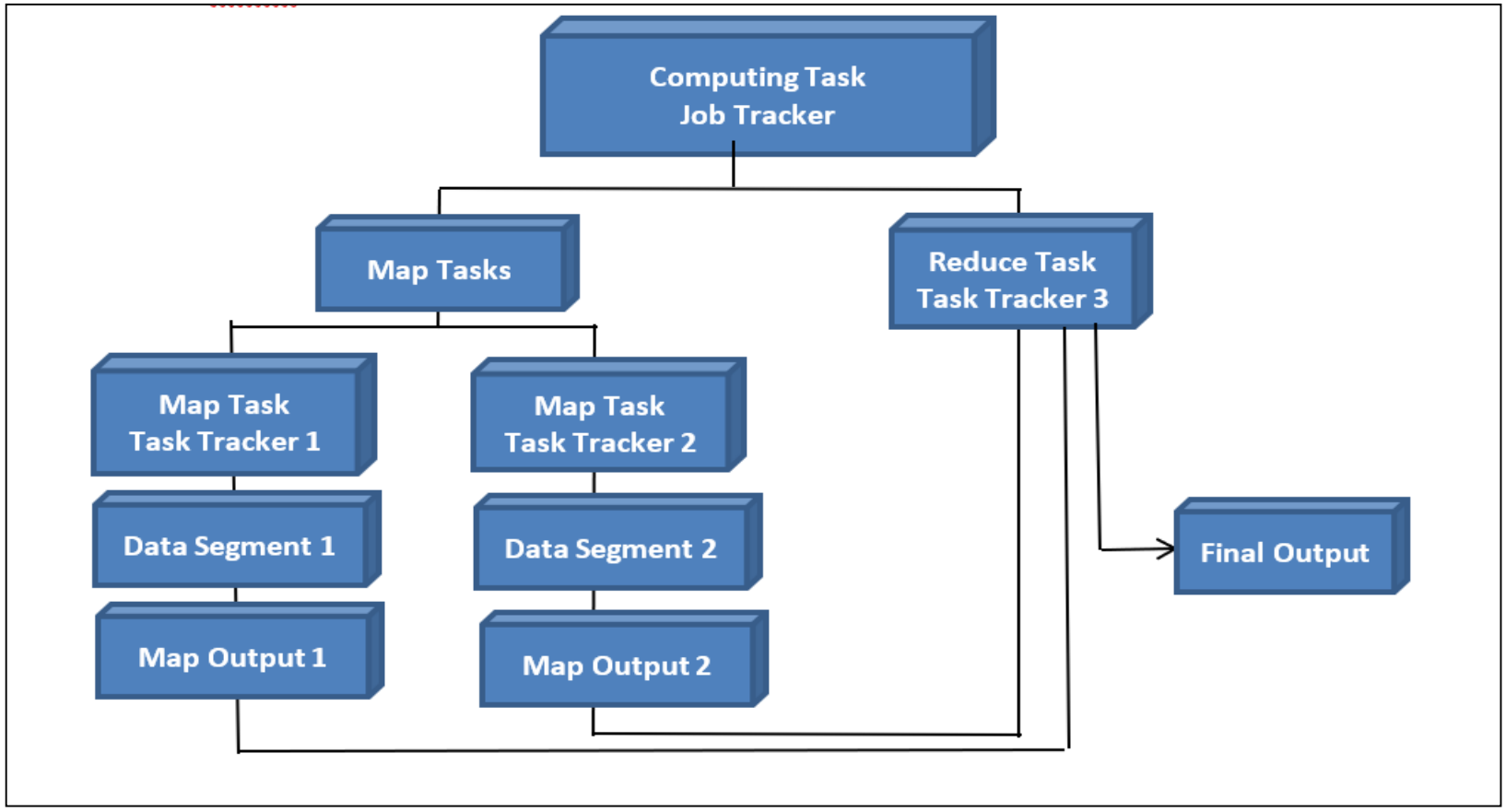
- A single TaskTracker per slave and spawns multiple Java Virtual Machines (VM) to handle multiple map of reduce tasks in parallel.
- TaskTracker continuously sends heartbeat measure to JobTracker.
- When the JobTracker fails to receive a heartbeat from a Task Tracker,
  - The JobTraker assumes that the TaskTracker has failed and
  - Resubmits the task to another available node in the cluster.
- Once the client submits a job to the Job Tracker, it partitions and assigns diverse MapReduce task for each TaskTracker in the cluster

JobTracker and TaskTracker Interactions



- MapReduce divides a data analysis task into two parts
- Map
- Reduce

MapReduce Programming Workflow



Example: Two mappers and one reducer

- Each mapper works on the partial dataset, that is stored in that node
- Reducer combines the output from the mappers to produce reduce result set

Steps to Perform Task in MapReduce

Step 1. The input dataset is split into multiple pieces of data (several small subsets)

Step 2. The framework creates a master and several workers processes and executes the worker processes remotely.

Step 3. Several map tasks work simultaneously and read pieces of data that were assigned to each map task.
The 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.

Step 4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of the specified mapper.

Steps to Perform Task in MapReduce

Step 5. When the map workers complete their work, the master instructs the reduce workers to begin their Work. The reduce workers in turn contact the map workers to get the key/value data tor their partition. The data thus received is shuflled and sorted as per keys.

Step 6. Then it calls reduce function for every unique key. This function writes the output to the file.

Step 7. When all the reduce workers complete their work, the master transfers the control to the user program.

HDFS - MapReduce Phases

- **Input Phase**
  - A Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map**
  - Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys**
  - They key-value pairs generated by the mapper are known as intermediate keys.

HDFS - MapReduce Phases

Combiner

- A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets.
- It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper.
- It is not a part of the main MapReduce algorithm;
- it is optional.

MapReduce Phases

Shuffle and Sort

- The Reducer task starts with the Shuffle and Sort step.
- It downloads the grouped key-value pairs onto the local machine, where the Reducer is running.
- The individual key-value pairs are sorted by key into a larger data list.
- The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

MapReduce Phases

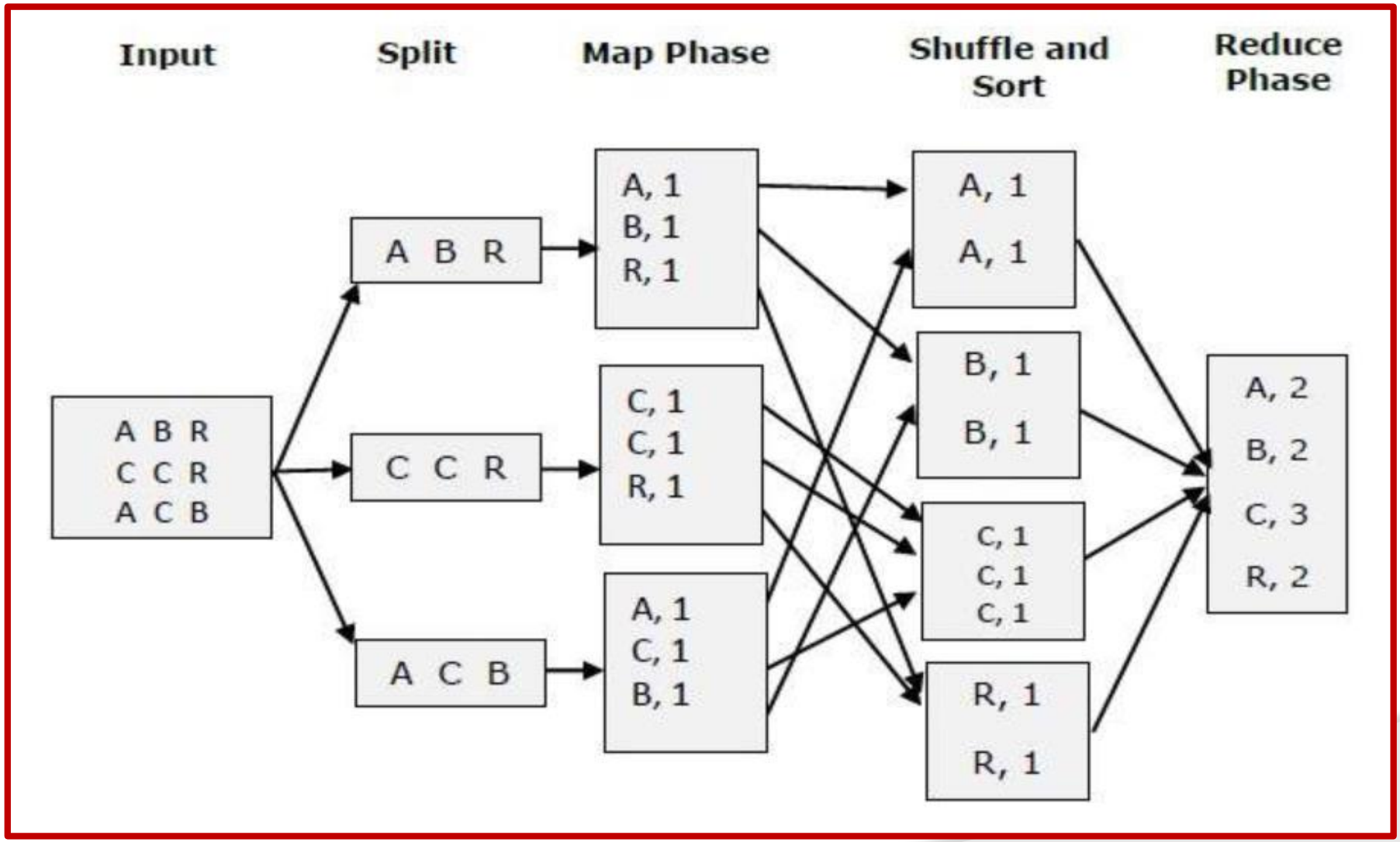
Reducer

- The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them.
- The data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing.
- Once the execution is over, it gives zero or more key-value pairs to the final step.

Output Phase

- In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

MapReduce Example



MapReduce – NCDC data Example

A Weather Dataset example, write a program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semistructured and record-oriented.

Data Format
National Climatic Data Center (NCDC, http://www .ncdc.noaa.gov/).
a sample line with some of the salient fields highlighted.
The line has been split into multiple lines to show each field: in the real file, fields are packed into one line with no delimiters.

MapReduce – NCDC data Example

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

MapReduce – NCDC data Example

Example. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
for year in all/*
do
 echo -ne `basename $year .gz`"\t"
 gunzip -c $year | \
 awk '{ temp = substr($0, 88, 5) + 0;
 q = substr($0, 93, 1);
 if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
 END { print max }'
done
```

a run:

```
% ./max_temperature.sh
1901 317
1902 244
1903 289
1904 256
```

MapReduce – NCDC data Example

Problems with parallel processing:

- First, dividing the work into equal-size pieces.
- Second, combining the results from independent processes may need further processing.
- Third, limited by the processing capacity of a single machine. If the best time you can achieve is 20 minutes with the number of processors

MapReduce Job– NCDC data Example

To run on a cluster of machines.

MAP

- The input to our **map phase** is the raw NCDC data.
- A text input format of each line in the dataset as a text value.
- The key is the offset of the beginning of the line from the beginning of the file.

Map function:

- The year and the air temperature are the interested fields
- The map function is just a data preparation phase
- The reducer function can do finding the maximum temperature for each year.
- The map function also drop bad records: here filter out temperatures that are missing, suspect, or erroneous.

UNIT -IV

UNDERSTANDING MAPREDUCE FUNDAMENTALS

- Map Reduce Framework:
- Exploring the features of Map Reduce
- Working of MapReduce, Exploring Map and Reduce Functions
- Techniques to optimize MapReduce jobs
- Uses of MapReduce.
- Controlling MapReduce Execution with Input Format
- Reading Data with custom Record Reader
- -Reader, Writer, Combiner, Partitioners
- MapReduce Phases
- Developing simple MapReduce Application.

UNIT -IV

UNDERSTANDING MAPREDUCE FUNDAMENTALS

- Map Reduce Framework: Exploring the features of Map Reduce
- Working of MapReduce

MapReduce Features

Advanced features of MapReduce, including counters, sorting and joining datasets.

Counters

- Counters are a useful channel for gathering statistics about the job: for quality control or for application-level statistics.
- Useful for problem diagnosis.
- To put a log message into your map or reduce task, it is often better to see whether you can use a counter instead to record that a particular condition occurred.
- In addition to counter values being much easier to retrieve than log output for large distributed jobs, you get a record of the number of times that condition occurred, which is more work to obtain from a set of logfiles.

MapReduce Features

Sorting

- The ability to sort data is at the heart of MapReduce.
- Even if your application isn't concerned with sorting, it may be able to
  - Use the sorting stage that MapReduce provides
  - Organize its data.

MapReduce Features

Joins

- MapReduce can perform joins between large datasets, but writing the code to do joins from scratch is fairly involved.
- Rather than writing MapReduce programs, you might consider using a higher-level framework such as Pig, Hive, or Cascading, in which join operations are a core part of the implementation.
- If the join is performed by the mapper, it is called a map-side join, whereas if it is performed by the reducer it is called a reduce-side join.
- Common example of this case is a user database and a log of some user activity (such as access logs).

MapReduce Working

MapReduce programming to perform wordcount.

Example,

consider the count of the occurances of similar words across 50 files.

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

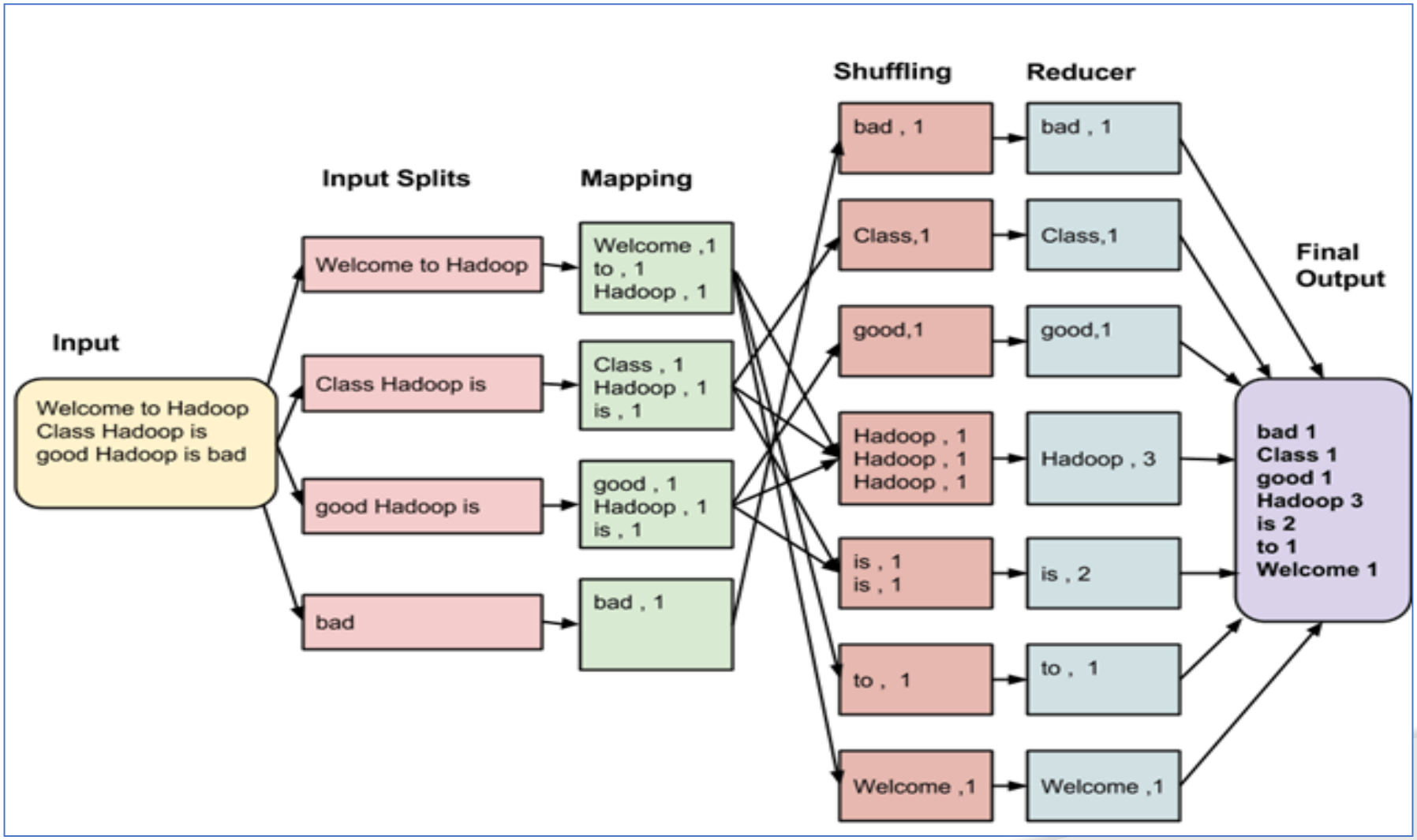
Consider following input data for your Map Reduce Program:

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad

MapReduce Working



MapReduce Working

The input of the MapReduce task is

Welcome to Hadoop Class
Hadoop is good
Hadoop is bad

The final output of the MapReduce task is

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1
Welcome	1

MapReduce Working

The data goes through the following phases

Input Splits

- An input to a MapReduce job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map

Mapping

- This is the very first phase in the execution of map-reduce program.
- In this phase data in each split is passed to a mapping function to produce output values.
- In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

MapReduce Working

Shuffling

- This phase consumes the output of Mapping phase.
- Its task is to consolidate the relevant records from Mapping phase output.
- In our example, the same words are clubbed together along with their respective frequency.

Reducing

- In this phase, output values from the Shuffling phase are aggregated.
- This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.
- In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

MapReduce Working

MapReduce program contains three components:

1. Driver Class: Specifies job configuration details

2. Mapper Class: Overrides the map function based on the problem statement

3. Reducer Class: Overrides the Reducer function based on the problem statement

MapReduce Working

Wordcounter.jave : Driver Program

```
Package com.app;
Import java.io.IOException;
Import org.apache.hadoop.fs.path;
Import org.apache.hadoop.io.text;
Import org.apache.hadoop.mapred.JobConf;
Import org.apache.hadoop.mapred.Mapper;
Import org.apache.hadoop.mapreduce.Job;
Import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
Import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
Import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
Import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class WordCounter {
public static void main (String [] args) throws IOExcption, InterruptedException, ClassNotFoundException{
Job job = new Job();
job.setJobName(“WodCounter”);
job. setJarByClass(“WodCounter.class”);
job.setMapperClass(WordcoonterMap.class)
job. setReducerdlass(WordcoonterRed.class)
job.setoutputKeyclass(Text.class)
job.setoutputValueclass(Text.class)
FileInputFormat.addinputPath(job, new Path(“/sample/word.txt”);
FileOutputFormat.addOutputPath(job, new Path(“/sample/wordCount”);
System.exit(job.waitForComplettion(true)?0:1);
}}
```

MapReduce Working

WordcounterMap.jave : Map Class

```
Package com.app;
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.mapreduce.Mapper;
public class WordCounterMap extends Mapper<LongWritable, Text, Text,
IntWritable> {
@Override
Protected void map (LongWriteable key, Text value, Context context) throws
IOExcption, InterruptedException{
String [] words=value.tostring().split(“,”);
for(String word: words){
context.write(new Text(word), new IntWritable(1));
}}}
```

MapReduce Working

WordcounterReduce.jave : Reduce Class

```
Package com.Infosys;
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.mapreduce.Mapper;
public class WordCounterMap extends Mapper<LongWritable, Text, Text, IntWritable> {
@Override
Protected void map(LongWriteable key, Text value, Contxt context) throws IOException,
InterruptedException {
        Integer count=0;
        for(IntWritable val: values){
        count += val.get();
        }
        count.write(word,new IntWritable(count));
}
```

MapReduce Application – NCDC data Example

A Weather Dataset example, write a program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semistructured and record-oriented.

Data Format
National Climatic Data Center (NCDC, http://www .ncdc.noaa.gov/).
a sample line with some of the salient fields highlighted.

The line has been split into multiple lines to show each field: in the real file, fields are packed into one line with no delimiters.

MapReduce – NCDC data Example

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

MapReduce – NCDC data Example

Example. **A program for finding the maximum recorded temperature by year from NCDC weather records**

```
#!/usr/bin/env bash
for year in all/*
do
 echo -ne `basename $year .gz`"\t"
 gunzip -c $year | \
 awk '{ temp = substr($0, 88, 5) + 0;
 q = substr($0, 93, 1);
 if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
 END { print max }'
done
```

a run:

```
% ./max_temperature.sh
1901 317
1902 244
1903 289
1904 256
```

MapReduce – NCDC data Example

Problems with parallel processing:

- First, dividing the work into equal-size pieces.
- Second, combining the results from independent processes may need further processing.
- Third, limited by the processing capacity of a single machine. If the best time you can achieve is 20 minutes with the number of processors

MapReduce Job– NCDC data Example

To run on a cluster of machines.

MAP

- The input to our **map phase** is the raw NCDC data.
- A text input format of each line in the dataset as a text value.
- The key is the offset of the beginning of the line from the beginning of the file.

Map function:

- The year and the air temperature are the interested fields
- The map function is just a data preparation phase
- The reducer function can do finding the maximum temperature for each year.
- The map function also drop bad records: here filter out temperatures that are missing, suspect, or erroneous.

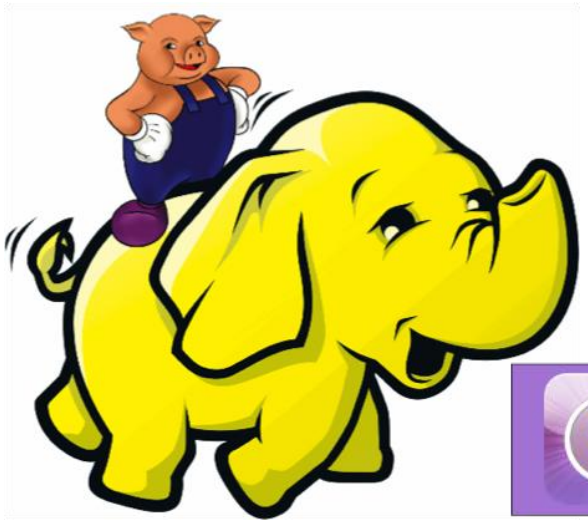
UNIT –V : INTRODUCTION TO PIG AND HIVE

- Introducing Pig: Pig architecture
- Benefits
- Installing Pig
- Properties of Pig
- Running Pig
- Getting started with Pig Latin
- Working with operators in Pig
- Working with functions in Pig.
- Introducing Hive: Getting started with Hive
- Hive Services
- Data types in Hive
- Built-in functions in Hive
- Hive DDL.

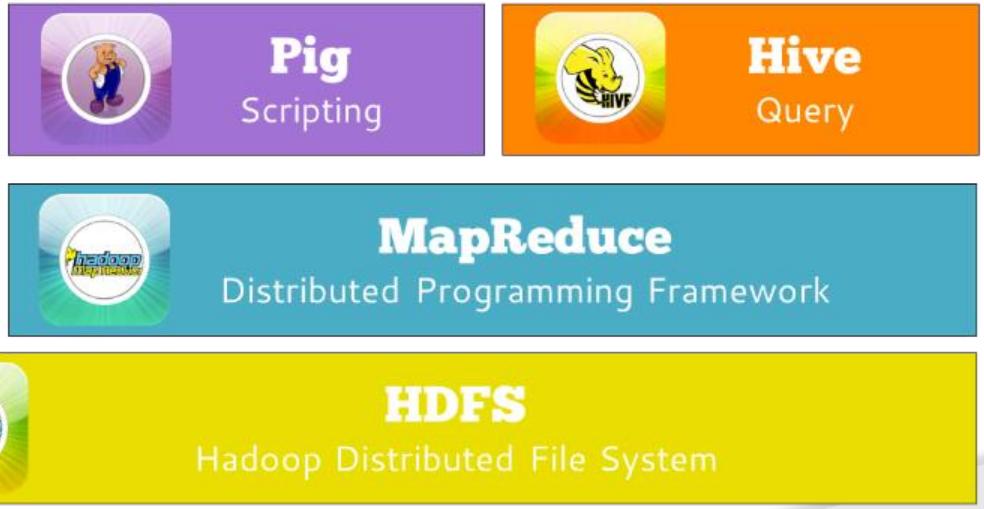
PIG

✓PIG's are omnivores animals which means they can consume both plants and animals.

✓The PIG consumes any type of data whether Structured or unStructured or any other machine data & helps processing the same.



PIG is on the top of hadoop.



Motivation:

- Map Reduce is very powerful, but:

– It requires a Java programmer.

– User has to re-invent common functionality (join, filter, etc.).

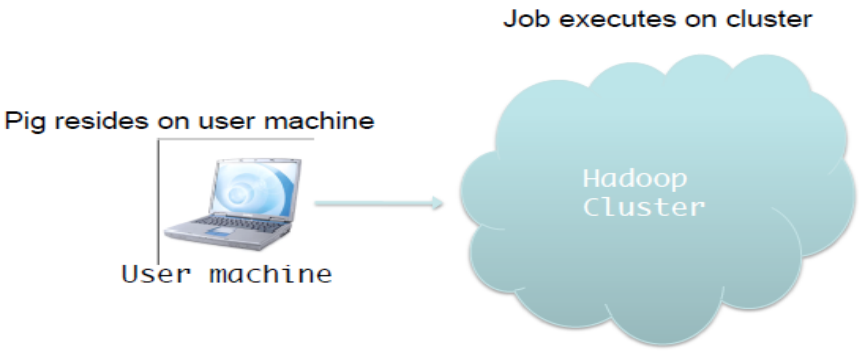
Word Count using Pig

```
Lines=LOAD 'input/access.log' AS (line: chararray);
Words = FOREACH Lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE
group, COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO /output/top5words;
```

What is PIG?

"is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. "

Components



No need to install anything extra on your Hadoop cluster.

- Sub-project of Apache Hadoop
- Platform for analyzing large data sets
- Includes a data-flow language Pig Latin
- Built for Hadoop
  - Translates script to MapReduce program under the hood
- Originally developed at Yahoo!
  - Huge contributions from Hortonworks, Twitter

- Framework for analyzing large un-structured and semi-structured data on top of Hadoop.
- Pig Engine Parses, compiles Pig Latin scripts into MapReduce jobs run on top of Hadoop.
- Pig Latin is simple but powerful data flow language similar to scripting languages.
  - SQL – like language
  - Provide common data operations (e.g. filters, joins, ordering)

How It Works

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

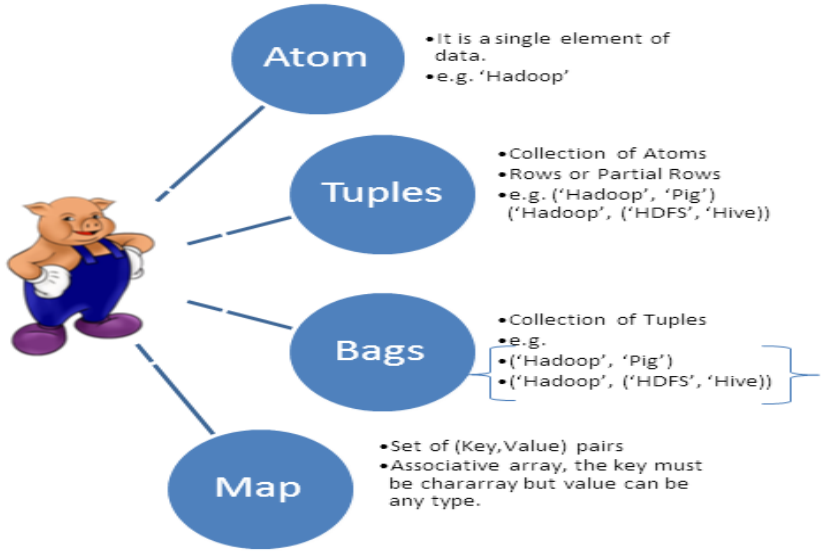
Execution Plan

Map:
Filter

Reduce:
Count

Why Pig?

- Makes writing hadoop jobs a lot simpler
  - 5% of the code, 5% of time
  - You don't have to be a programmer to write Pig scripts
- Provides major functionality required for DW and Analytics
  - Load, Filter, Join, Group By, Order, Transform, UDFs, Store
- User can write custom UDFs (User Defined Function)



Pig Tutorial

- Basic Pig knowledge: (Word Count)
  - Pig Data Types
  - Pig Operations
  - How to run Pig Scripts

PigLatin - the dataflow language

- PigLatin statements work with relations
  - A relation (analogous to database table) is a bag
  - A bag is a collection of tuples
  - A tuple (analogous to database row) is an ordered set of fields
  - A field is a piece of data
- Example, A = LOAD 'input.dat';
  - Here 'A' is a relation
  - All records in 'A' (from the file 'input.dat') collectively form a bag
  - Each record in 'A' is a tuple
  - A field is a single cell in each tuple

To remember : A Pig relation is a bag of tuples

Apache Pig Execution Modes

You can run Apache Pig in two modes, namely, **Local Mode** and **HDFS mode**.

Local Mode

In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode

MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

Apache Pig Execution Mechanisms

Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

- **Interactive Mode** (Grunt shell) – You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).
- **Batch Mode** (Script) – You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.
- **Embedded Mode** (UDF) – Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

Local mode

- All files are installed and run using your local host and file system
  - Does not involve a real hadoop cluster
- Great for starting off, debugging
- Specify local mode using the -x flag
  - $ pig -x local
  - $ grunt> a = load 'foo'; -- here the file 'foo' resides on local filesystem

Mapreduce mode

- Default mode
- Access to a Hadoop cluster and HDFS installation
- Point Pig to remote cluster by placing HADOOP_CONF_DIR on PIG_CLASSPATH
  - HADOOP_CONF_DIR is the directory containing your hadoop-site.xml, hdfs-site.xml, mapred-site.xml files
  - Example: $ export PIG_CLASSPATH=<path_to_hadoop_conf_dir>
  - $ pig
  - grunt> a = load 'foo'; -- here 'foo' refers to a file on HDFS

Data types

- int, long
- float, double
- chararray - Java String
- bytearray
  - default type of all fields if schema not specified
- Complex data types
  - tuple, eg (abc,def)
  - bag, eg {(19,2), (18,1)}
  - map, eg [sfdc#logs]

Pig Operations

- Loading data
  - LOAD loads input data
  - Lines=LOAD 'input/access.log' AS (line: chararray);
- Projection
  - FOREACH ... GENERTE (similar to SELECT)
  - takes a set of expressions and applies them to every record.
- De-duplication
  - DISTINCT removes duplicate records
- Grouping
  - GROUPS collects together records with the same key
- Aggregation
  - AVG, COUNT, COUNT_STAR, MAX, MIN, SUM

Loading data

- LOAD
  - Reads data from the file system
- Syntax
  - LOAD 'input' [USING function] [AS schema];
  - Eg, A = LOAD 'input' USING PigStorage('\t') AS (name:chararray, age:int, gpa:float);

Pig Commands

Pig Command	What it does
load	Read data from file system.
store	Write data to file system.
foreach	Apply expression to each record and output one or more records.
filter	Apply predicate and remove records that do not return true.
group/cogroup	Collect records with the same key from one or more inputs.
join	Join two or more inputs based on a key.
order	Sort records based on a key.
distinct	Remove duplicate records.
union	Merge two data sets.
split	Split data into 2 or more sets, based on filter conditions.
stream	Send all records through a user provided binary.
dump	Write output to stdout.
limit	Limit the number of records.

Schema

- Use schemas to assign types to fields
- A = LOAD 'data' AS (name, age, gpa);
  - name, age, gpa default to bytearrays
- A = LOAD 'data' AS (name:chararray, age:int, gpa:float);
  - name is now a String (chararray), age is integer and gpa is float

Describing Schema

- Describe
  - Provides the schema of a relation
- Syntax
  - DESCRIBE [alias];
  - If schema is not provided, describe will say "Schema for alias unknown"

```
grunt> A = load 'data' as (a:int, b: long, c: float);
grunt> describe A;
A: {a: int, b: long, c: float}

grunt> B = load 'somemoredata';
grunt> describe B;
Schema for B unknown.
```

Dump and Store

- Dump writes the output to console
  - grunt> A = load 'data';
  - grunt> DUMP A; //This will print contents of A on Console
- Store writes output to a HDFS location
  - grunt> A = load 'data';
  - grunt> STORE A INTO '/user/username/output'; //This will write contents of A to HDFS
- Pig starts a job only when a DUMP or STORE is encountered

Referencing Fields

- Fields are referred to by positional notation OR by name (alias)
  - Positional notation is generated by the system
  - Starts with $0
  - Names are assigned by you using schemas. Eg, A = load 'data' as (*name*:chararray, *age*:int);
- With positional notation, fields can be accessed as
  - A = load 'data';
  - B = foreach A generate $0, $1; //1st& 2nd column

Limit

- Limits the number of output tuples
- Syntax
  - alias = LIMIT alias n;

```
grunt> A = load 'data';

grunt> B = LIMIT A 10;

grunt> DUMP B; --Prints only 10 rows
```

Foreach.. Generate

- Used for data transformations and projections
- Syntax
  - alias = FOREACH { block | nested_block };
  - *nested_block usage later in the deck*

```
grunt>A = load 'data' as (a1,a2,a3);

grunt>B = FOREACH A GENERATE *,

grunt>DUMP B;
(1,2,3)
(4,2,1)

grunt>C = FOREACH A GENERATE a1, a3;

grunt> DUMP C;
(1,3)
(4,1)
```

Filter

- Selects tuples from a relation based on some condition
- Syntax
  - alias = FILTER alias BY expression;
  - Example, to filter for 'marcbenioff'
    - A = LOAD 'sfdcemployees' USING PigStorage(',') as (name:chararray,employeesince:int,age:int);
    - B = FILTER A BY name == 'marcbenioff';
  - You can use boolean operators (AND, OR, NOT)
    - B = FILTER A BY (employeesince< 2005) AND (NOT(name == 'marcbenioff'));

Group By

- Groups data in one or more relations (similar to SQL GROUP BY)
- Syntax:
  - alias = GROUP alias { ALL | BY expression} [, alias ALL | BY expression ...] [PARALLEL n];
  - Eg, to group by (employee start year at Salesforce)
    - A = LOAD 'sfdcemployees' USING PigStorage(',') as (name:chararray, employeesince:int, age:int);
    - B = GROUP A BY (employeesince);
  - You can also group by all fields together
    - B = GROUP B BY ALL;
  - Or Group by multiple fields
    - B = GROUP A BY (age, employeesince);

Using Grouped Results

- FOREACH works for grouped data
- Let's see an example to count the number of rows grouped by employee start year

```
grunt> A = load 'data' as (name, employeesince, age);
grunt> B = GROUP A by employeesince;
grunt> C = FOREACH B GENERATE group, COUNT(A);
```

- 'group' is an implicit field name given to group key
- Use the alias grouped, within an aggregation function - COUNT(A)

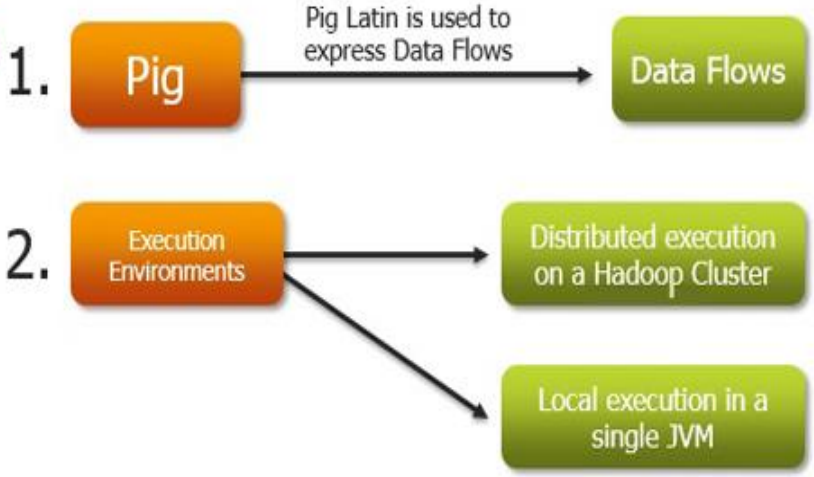
Aggregation

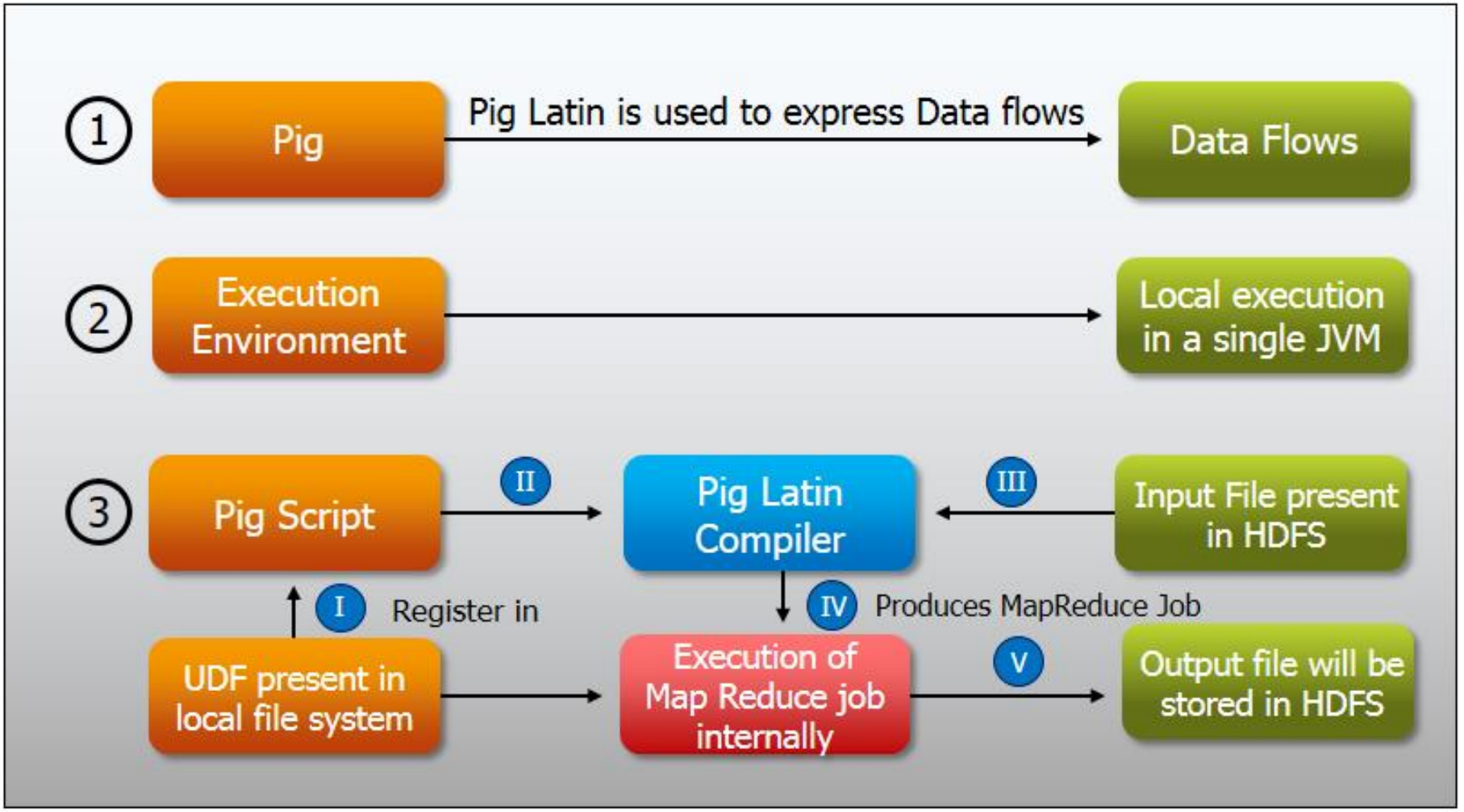
- Pig provides a bunch of aggregation functions
  - AVG
  - COUNT
  - COUNT_STAR
  - SUM
  - MAX
  - MIN

Case Sensitivity

- names (aliases) of relations and fields are case sensitive
  - A = load 'input'; B = foreacha generate $0; *--Won't work*
- UDF names are case sensitive
  - 'LENGTH' is not the same as 'length'
- PigLatin keywords are case insensitive
  - Load, dump, Group by, foreach..generate, join

Creating Your First Pig script




1
Pig
Pig Latin is used to express Data flows
Data Flows
2
Execution Environment
Local execution in a single JVM
3
Pig Script
II
Pig Latin Compiler
III
Input File present in HDFS
I
Register in
IV
Produces MapReduce Job
UDF present in local file system
Execution of Map Reduce job internally
V
Output file will be stored in HDFS

Thank you

&

Discussions..