Chapter 1

Types of Digital Data

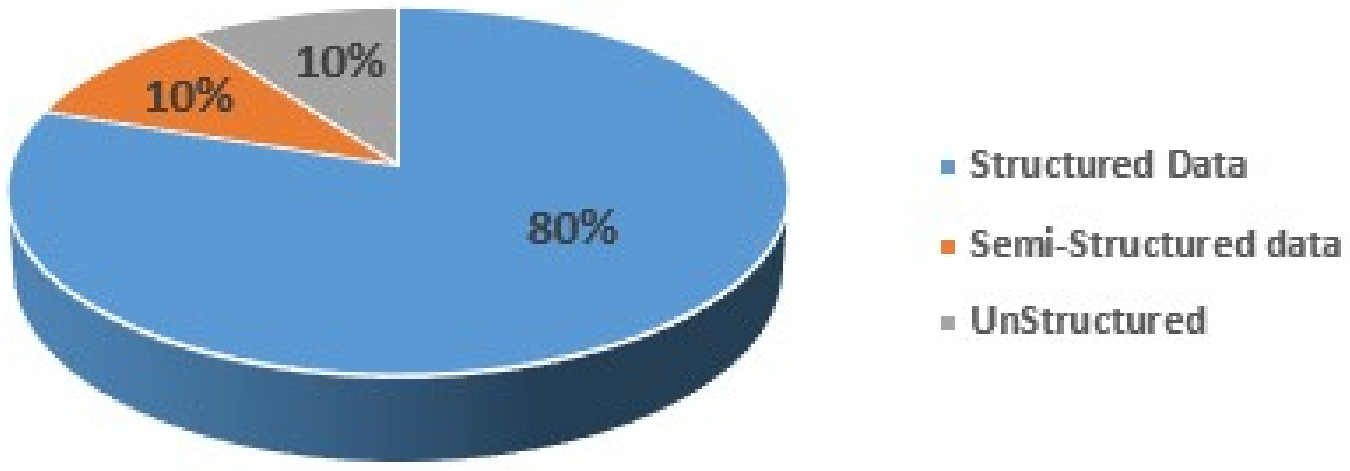
Classification of Digital Data

Digital data is classified into the following categories:

- Structured data
- Semi-structured data
- Unstructured data

Approximate Percentage Distribution of Digital Data

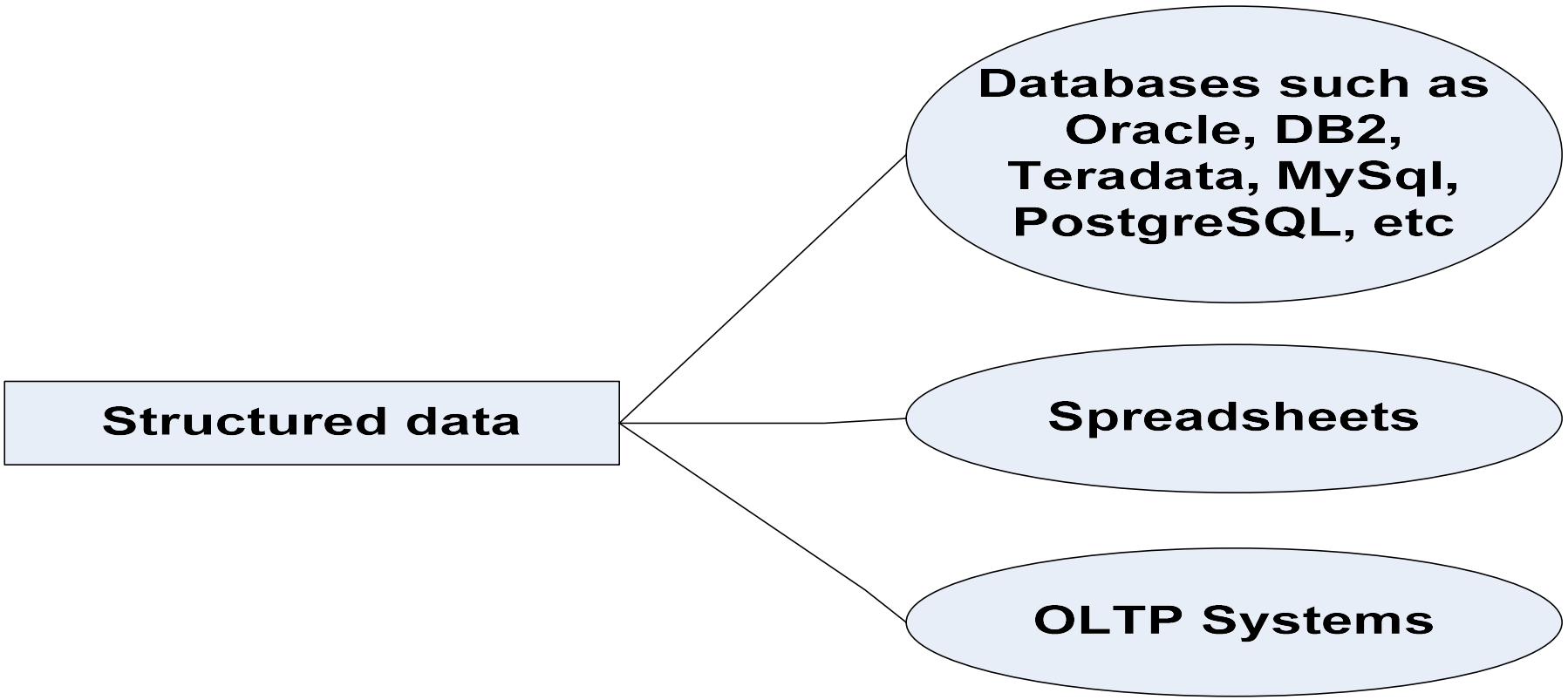
Approximate percentage distribution of digital data



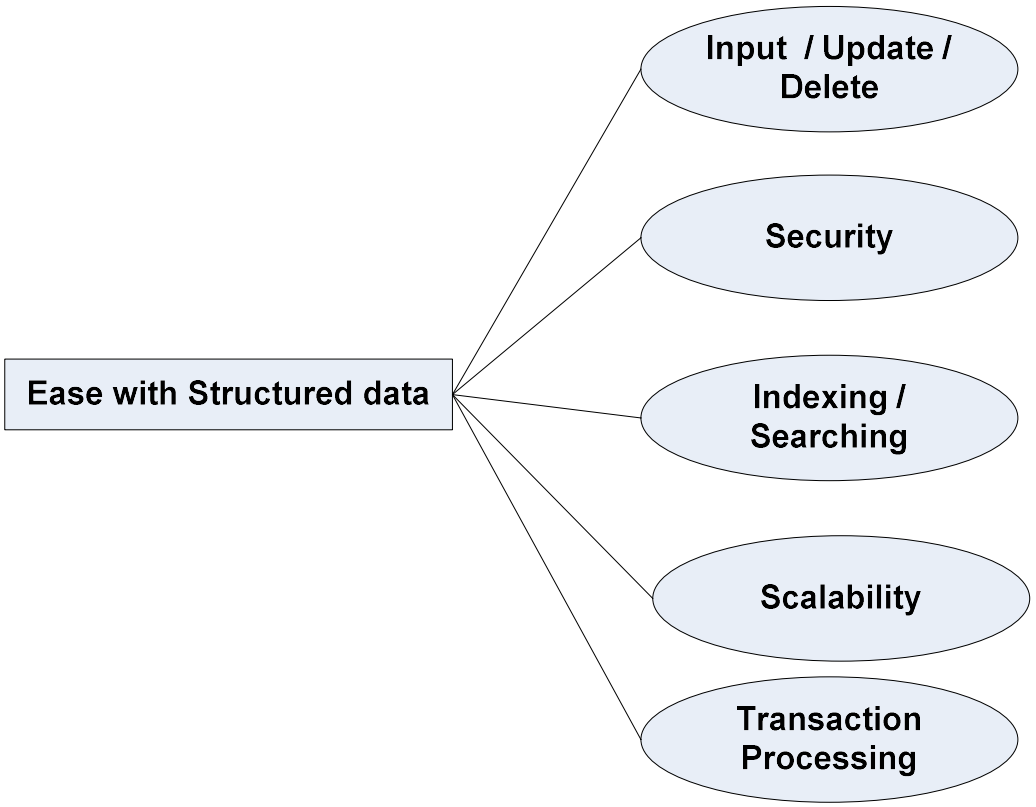
Structured Data

- This is the data which 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 is an example of structured data.

Sources of Structured Data



Ease with Structured Data

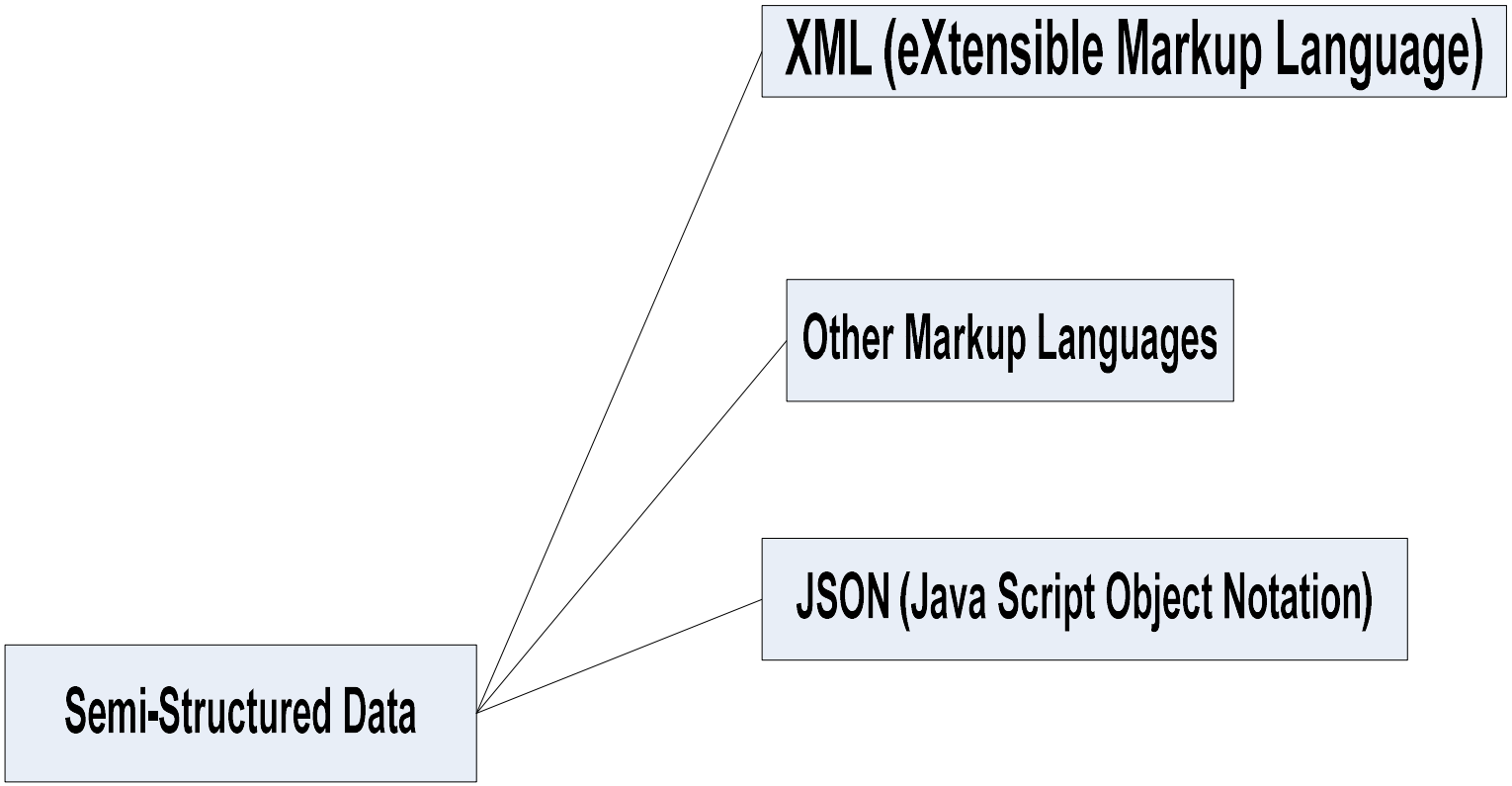


Semi-structured Data

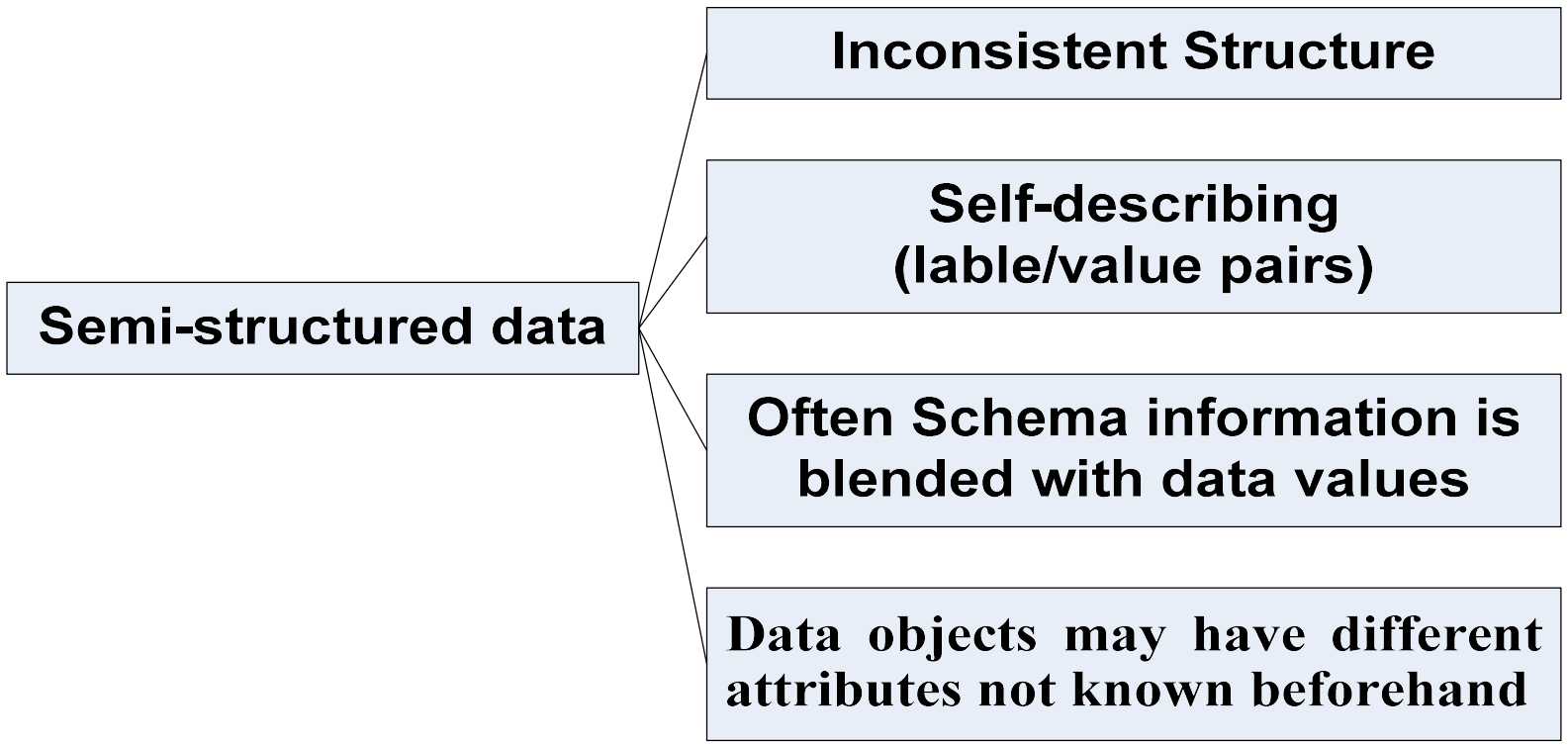
- This is the data which does not conform to a data model but has some structure. However, it is not in a form which can be used easily by a computer program.

- Example, emails, XML, markup languages like HTML, etc. Metadata for this data is available but is not sufficient.

Sources of Semi-structured Data



Characteristics of Semi-structured Data



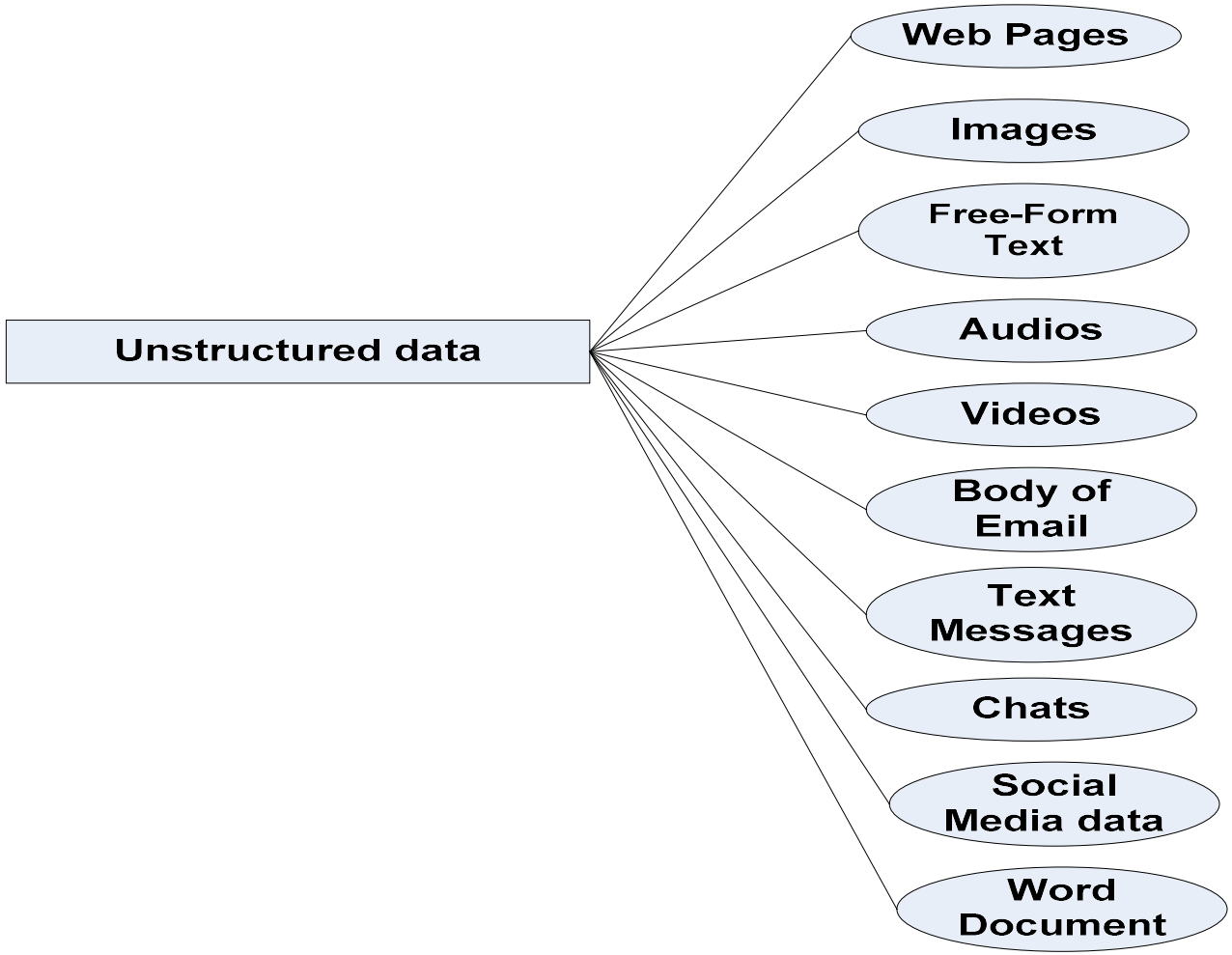
Unstructured Data

- This is the data which does not conform to a data model or is not in a form which can be used easily by a computer program.

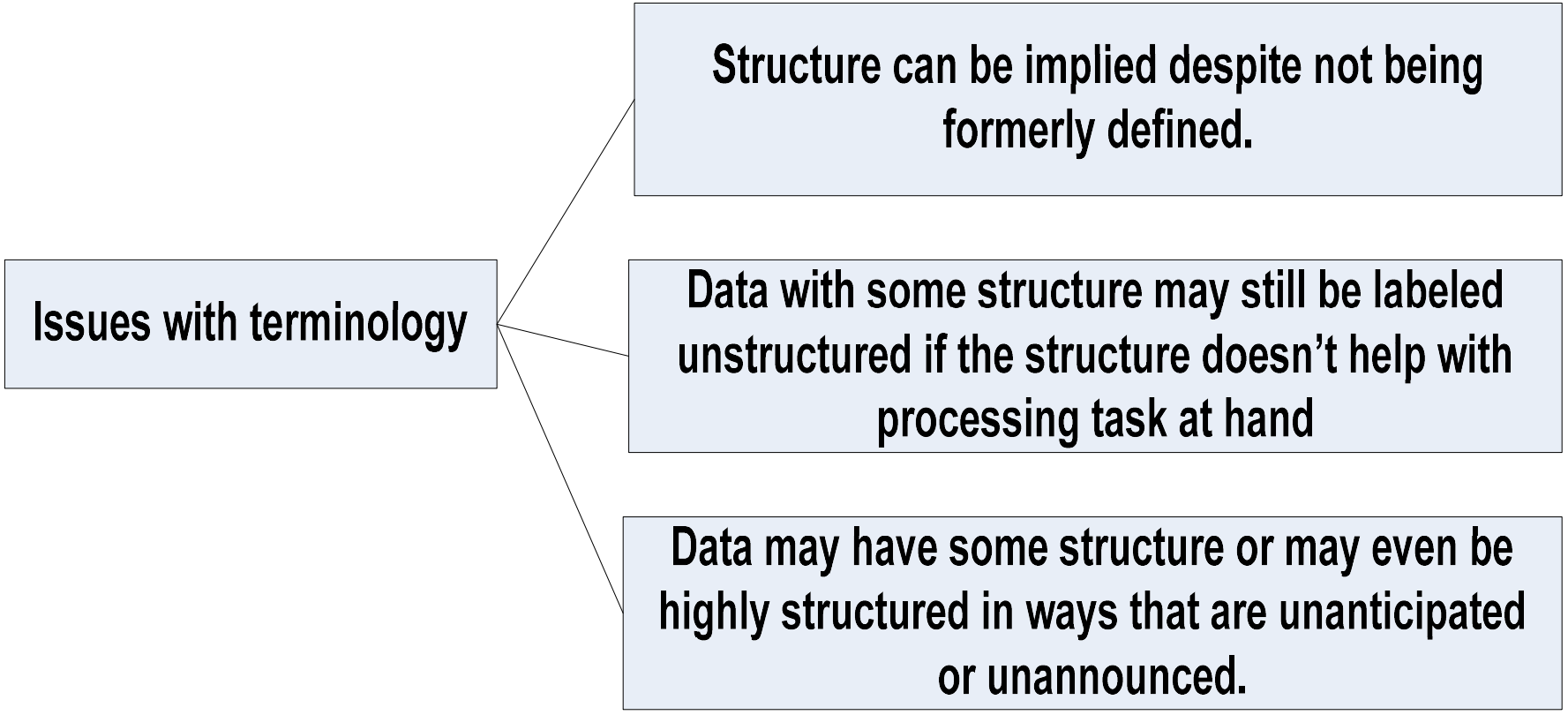
- About 80–90% data of an organization is in this format.

- Example: memos, chat rooms, PowerPoint presentations, images, videos, letters, researches, white papers, body of an email, etc.

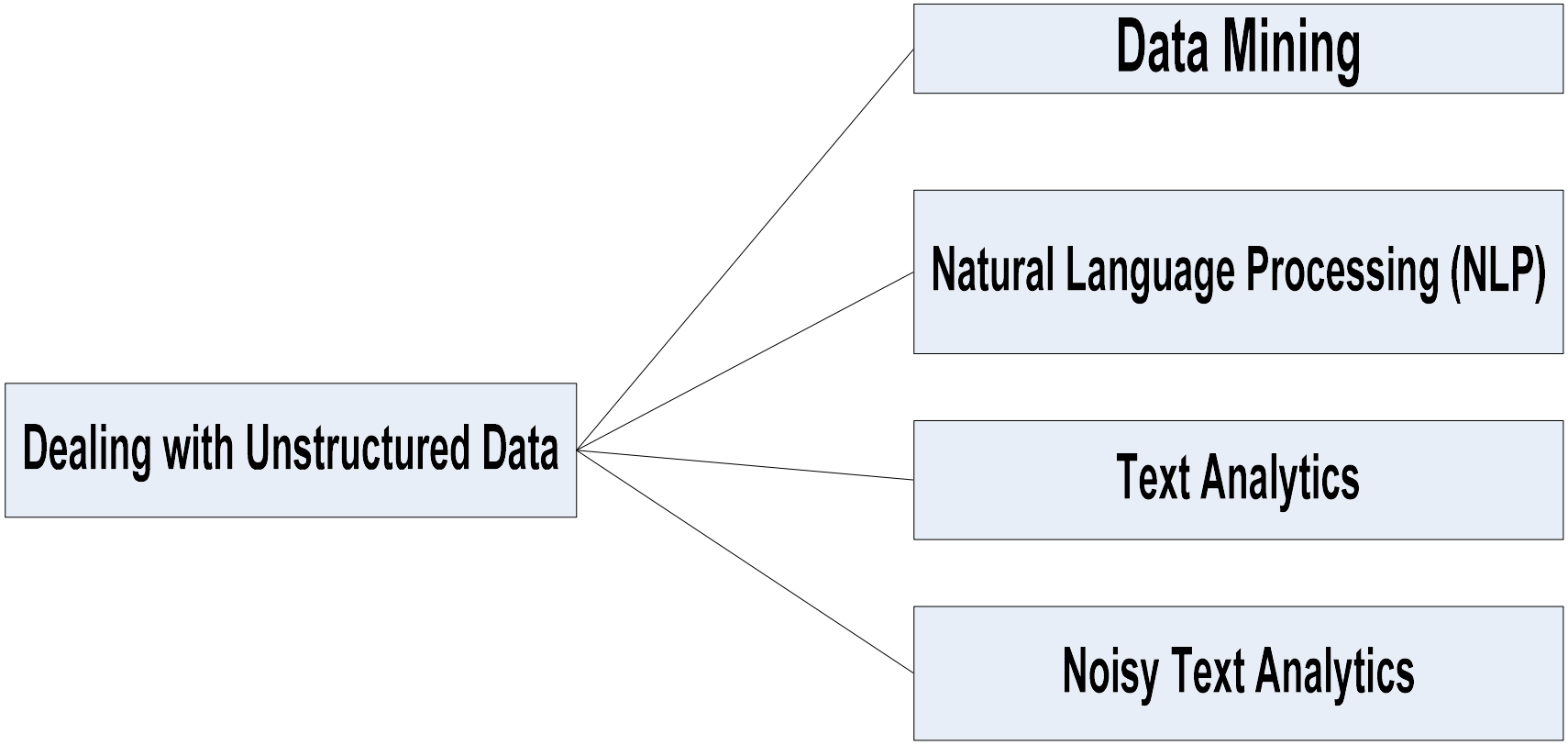
Sources of Unstructured Data



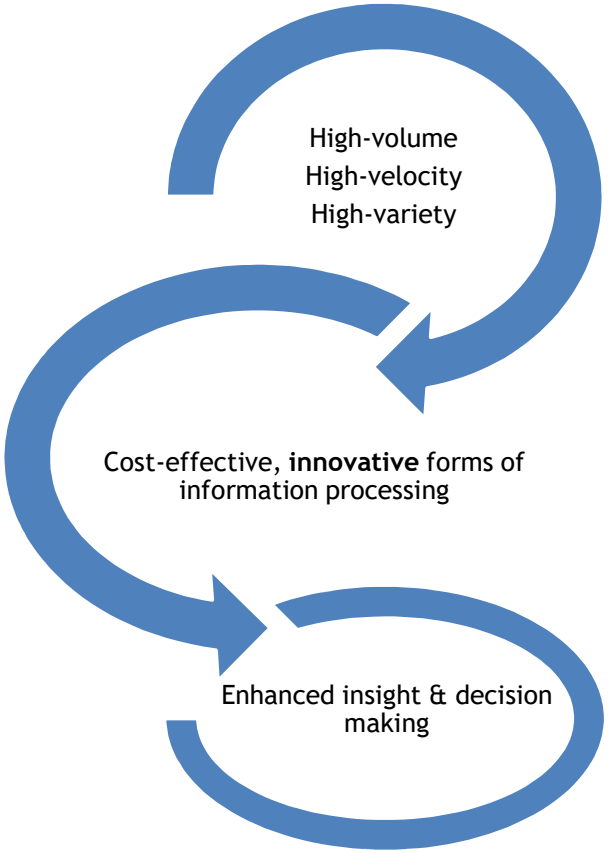
Issues with terminology - Unstructured Data



Dealing with Unstructured Data

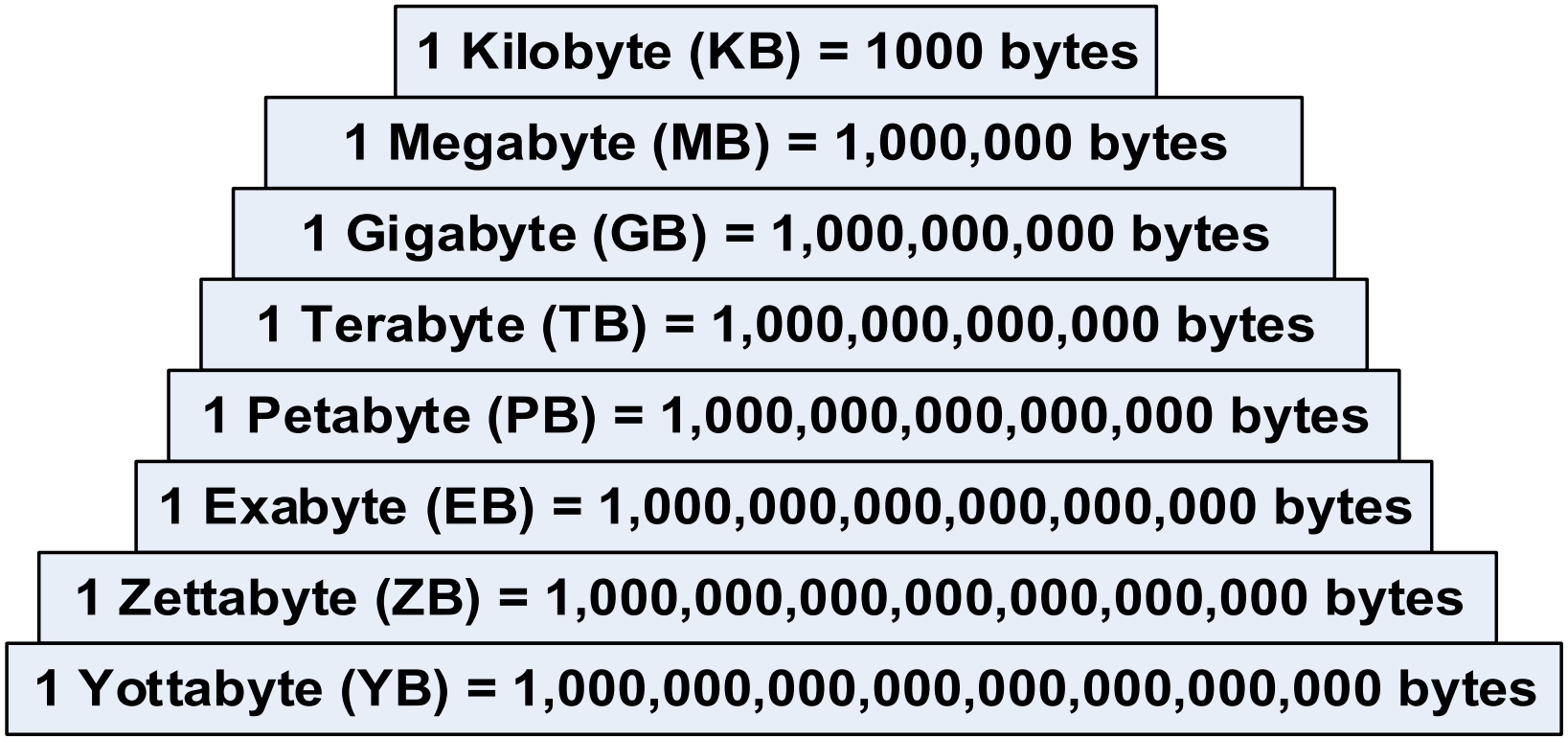


Definition of Big Data



Big Data is high-volume, high-velocity, and high-variety information assets that demand cost effective, innovative forms of information processing for enhanced insight and decision making.

Volume - A Mountain of Data



Velocity

Batch → Periodic → Near real time → Real-time processing

Variety

- **Structured data:** example: traditional transaction processing systems and RDBMS, etc.

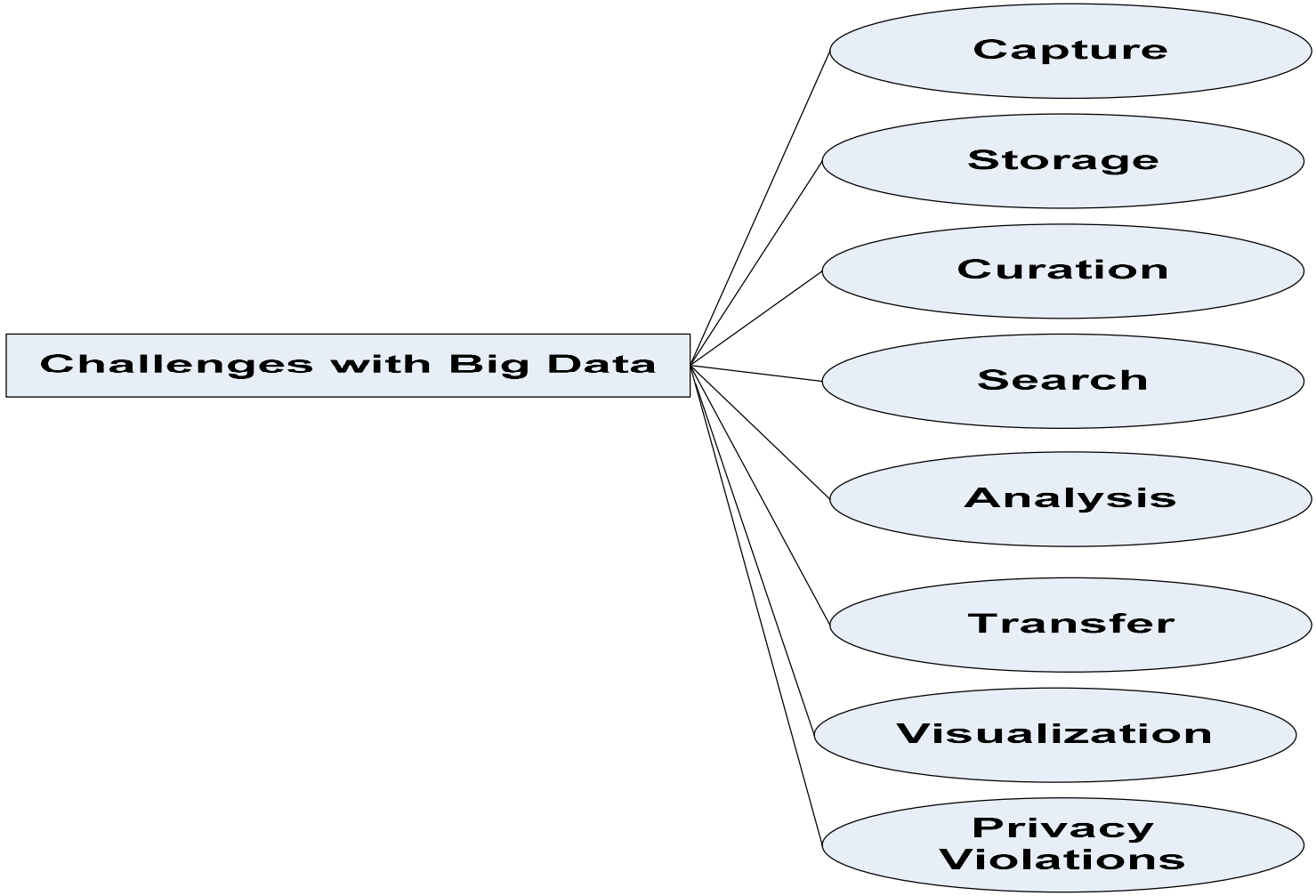
- **Semi-structured data:** example: Hyper Text Markup Language (HTML), eXtensible Markup Language (XML).

- **Unstructured data:** example: unstructured text documents, audio, video, email, photos, PDFs, social media, etc.

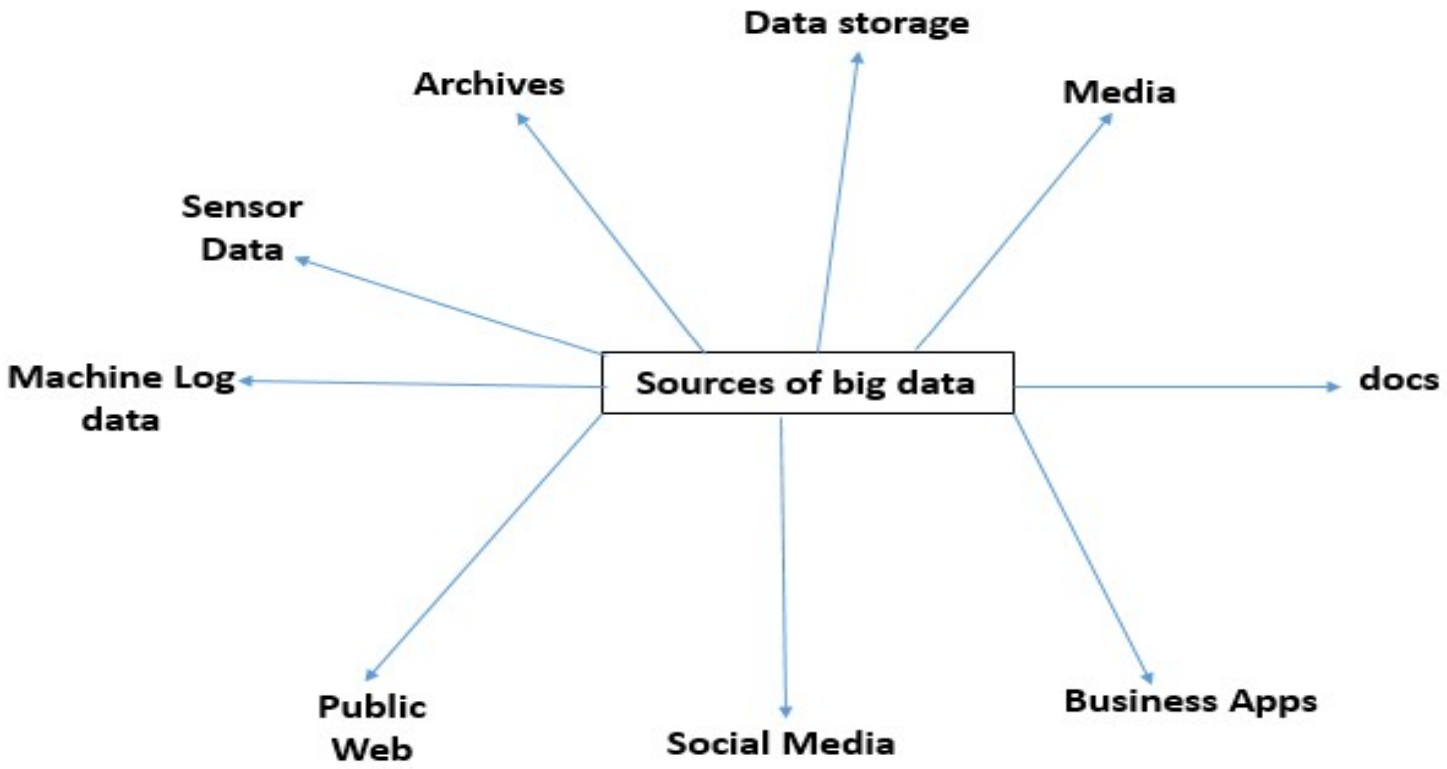
Other Characteristics of Data – Which are not Definitional Traits of Big Data

- Veracity and Validity- refers to biases, noise, and abnormality in data
- Volatility of data deals with, how long is the data valid?
- Variability

Challenges with Big Data



Sources of Big Data



Traditional BI vs Big Data Environment

Traditional Business Intelligence (BI)

1. Data Sources:

Primarily structured data from internal databases.

Relational databases, spreadsheets, ERP systems.

2. Data Volume:

Handles moderate data volumes.

Data is usually aggregated and sampled.

3. Data Processing:

Batch processing.

Periodic updates (daily, weekly, monthly).

Big Data Environment

1. Data Sources:

Structured, semi-structured, and unstructured data.

Includes web logs, social media, sensor data, etc.

2. Data Volume:

Designed to handle very large data volumes (terabytes to petabytes).

Capable of dealing with real-time streaming data.

3. Data Processing:

Both batch and real-time processing.

Distributed computing and parallel processing.

4. Technology Stack:
Relational Database Management Systems (RDBMS).
SQL for querying.
ETL (Extract, Transform, Load) tools.
Data warehouses.
5. Analysis and Reporting:
Pre-defined reports and dashboards.
Static and historical analysis.
Limited real-time capabilities.
6. User Base:
Mainly business analysts and managers.
Focus on descriptive analytics (what happened).
7. Performance:
Optimized for complex queries on structured data.
Performance can degrade with very large datasets.

4. Technology Stack:
NoSQL databases, Hadoop ecosystem, Spark, etc.
Advanced data integration tools.
Data lakes for storing raw data.
5. Analysis and Reporting:
Advanced analytics including predictive and prescriptive analytics.
Real-time data analysis.
Machine learning and AI capabilities.
6. User Base:
Data scientists, data engineers, and advanced business users.
Focus on deeper insights, predictions, and automated decisions.
7. Performance:
Scalable and can handle high-velocity data.
Designed for high availability and fault tolerance.

Difference between Traditional BI and Big Data Environment

- **Data Variety:**
- Traditional BI deals primarily with structured data.
- Big Data environments handle a wide variety of data types.
- **Data Velocity:**
- Traditional BI processes data in batches.
- Big Data environments support real-time data processing.

Scalability:
Traditional BI systems can struggle with scaling.
Big Data environments are built to scale out horizontally
Cost:
Traditional BI solutions can be costly due to licensing and infrastructure.
Big Data environments often use open-source technologies, potentially reducing costs.
Flexibility:
Traditional BI systems are less flexible with data schemas.
Big Data environments are more flexible, accommodating changes in data structure.

Big Data (Descriptive, Predictive and Prescriptive)

- **Descriptive Analytics:** This type of analytics summarizes or extracts insights based on the **incoming** We came up with a description based on the data. For example, insights drawn for your YouTube channel are based on the data such as likes, shares, and views on your videos.
- **Predictive Analytics:** This type of analytics **predicts** what might happen. Questions such as 'how' and 'why' reveal particular patterns that help predict future trends. Machine Learning concepts are also used for such types of analysis. For example, prediction of weather, prediction of malfunctioning in the parts of an airplane, etc.
- **Prescriptive Analytics:** These types of analytics are based on rules and recommendations and thus prescribe an analytical path. The analysis is generally based on the question, **'what actions should be taken?' Google's** self-driving car is an example of prescriptive analysis.

1. Descriptive Analytics

Purpose:

To describe and summarize historical data.

To identify patterns and trends from past data.

Key Features:

Provides insights into what happened in the past.

Uses techniques such as data aggregation, data mining, and data visualization.

Often involves creating reports, dashboards, and summaries.

Examples:

Monthly sales reports showing total revenue.

Analysis of customer demographics.

Trends in website traffic over time.

Tools and Technologies:

Business Intelligence (BI) tools like Tableau, Power BI.

SQL databases.

Data warehouses.

Benefits:

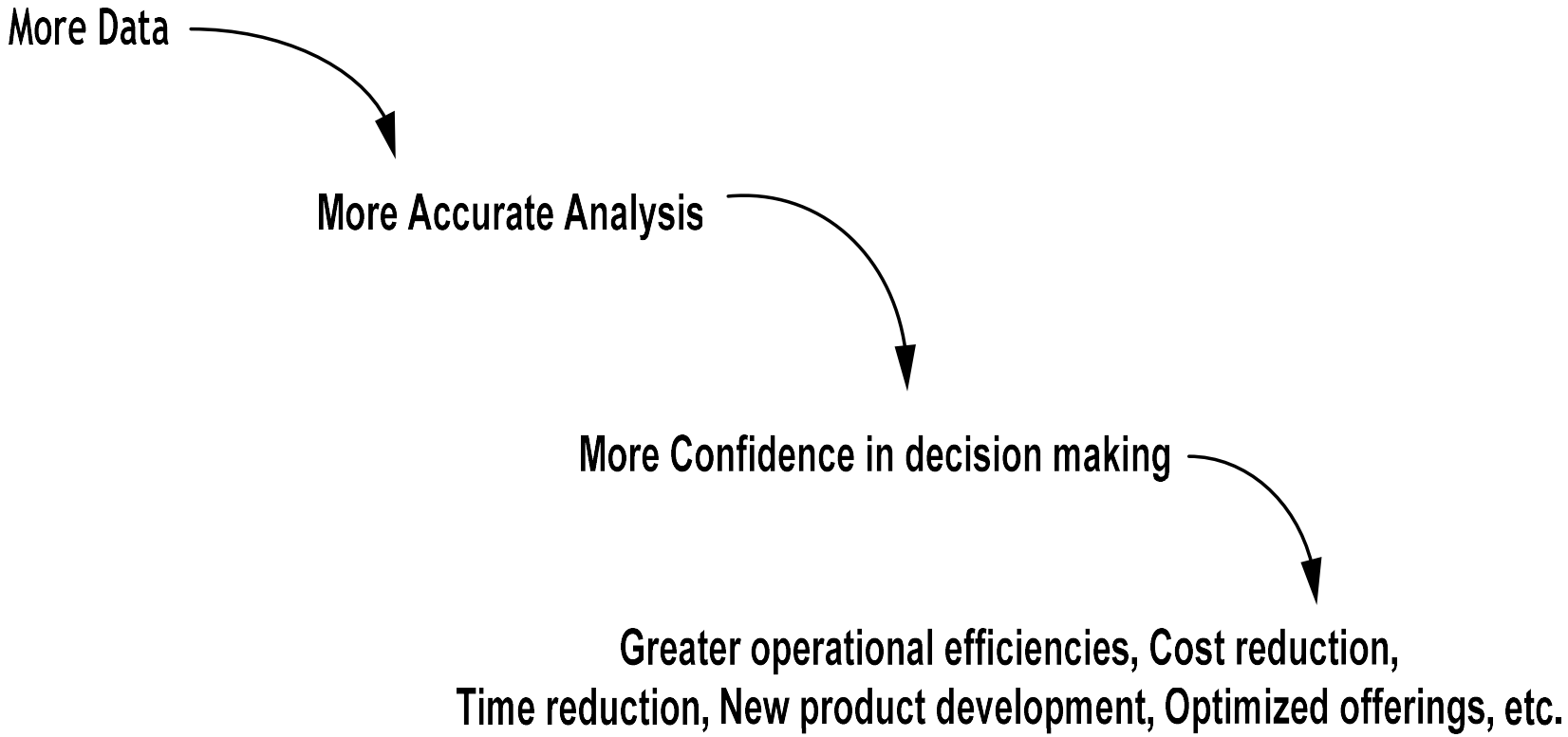
Helps in understanding historical performance.

Assists in identifying patterns and anomalies.

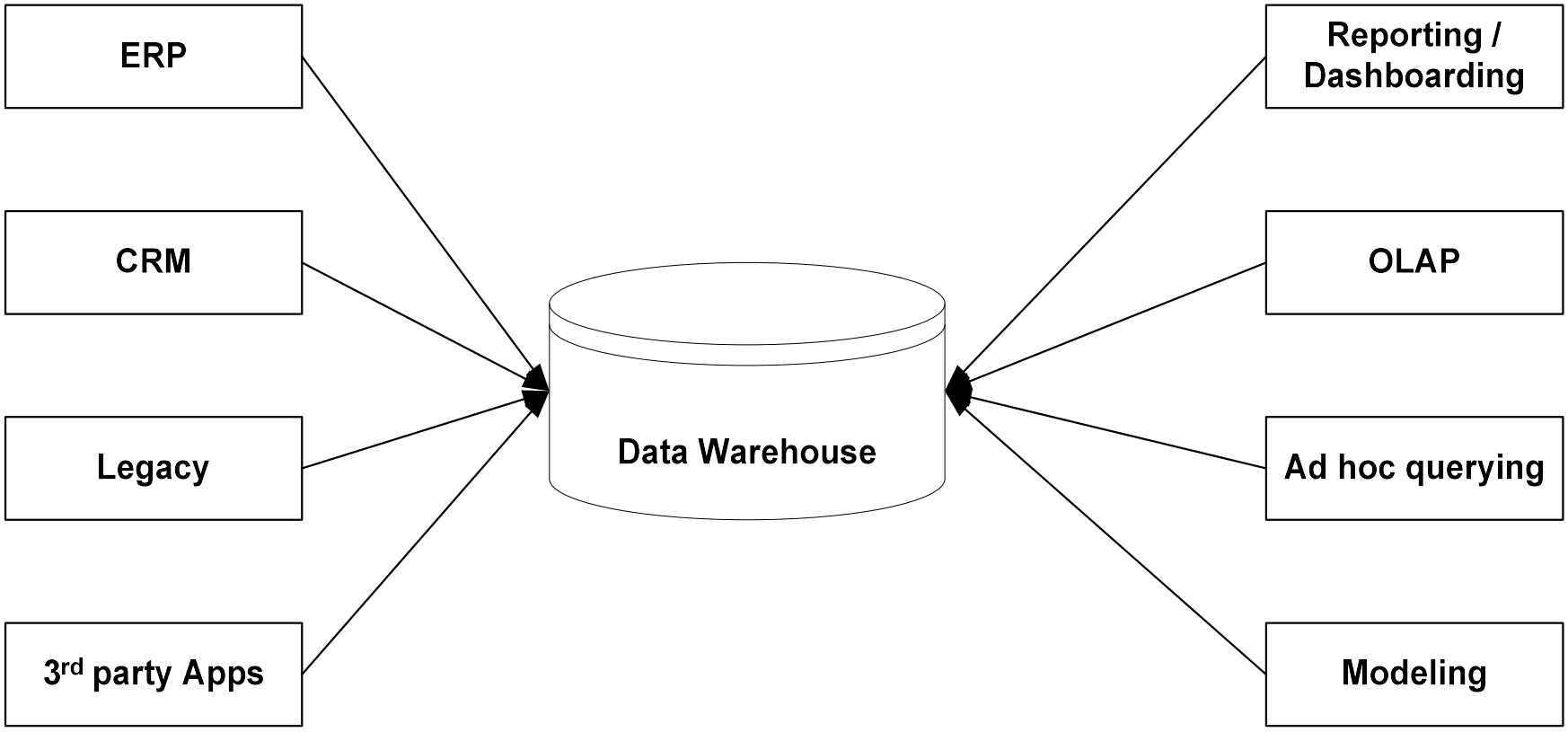
- **2. Predictive Analytics**
- **Purpose:**
- To predict future outcomes based on historical data.
- To identify potential opportunities and risks.
- **Key Features:**
- Uses statistical models, machine learning algorithms, and data mining.
- Provides insights into what is likely to happen.
- Involves techniques like regression analysis, time series analysis, and classification.
- **Examples:**
- Predicting customer churn rates.
- Forecasting stock prices or sales.
- Identifying potential fraud in transactions.
- **Tools and Technologies:**
- Machine learning frameworks like Scikit-Learn, TensorFlow, and PyTorch.
- Predictive modeling tools like IBM SPSS, SAS.
- Big Data platforms like Hadoop and Spark.
- **Benefits:**
- Enhances decision-making by anticipating future trends.
- Helps in risk management and strategic planning.

- **3. Prescriptive Analytics**
- **Purpose:**
- To recommend actions based on predictive insights.
- To optimize decision-making by suggesting the best course of action.
- **Key Features:**
- Combines predictive analytics with optimization techniques.
- Uses advanced algorithms, machine learning, and simulation.
- Provides insights into what should be done.
- **Examples:**
- Recommending inventory levels to minimize stockouts and overstock situations.
- Suggesting personalized marketing strategies for individual customers.
- Optimizing supply chain logistics.
- **Tools and Technologies:**
- Advanced analytics platforms like IBM Watson, SAS Analytics.
- Optimization tools like CPLEX, Gurobi.
- Machine learning libraries and frameworks.
- **Benefits:**
- Improves operational efficiency and effectiveness.
- Helps in achieving desired outcomes with data-driven recommendations.

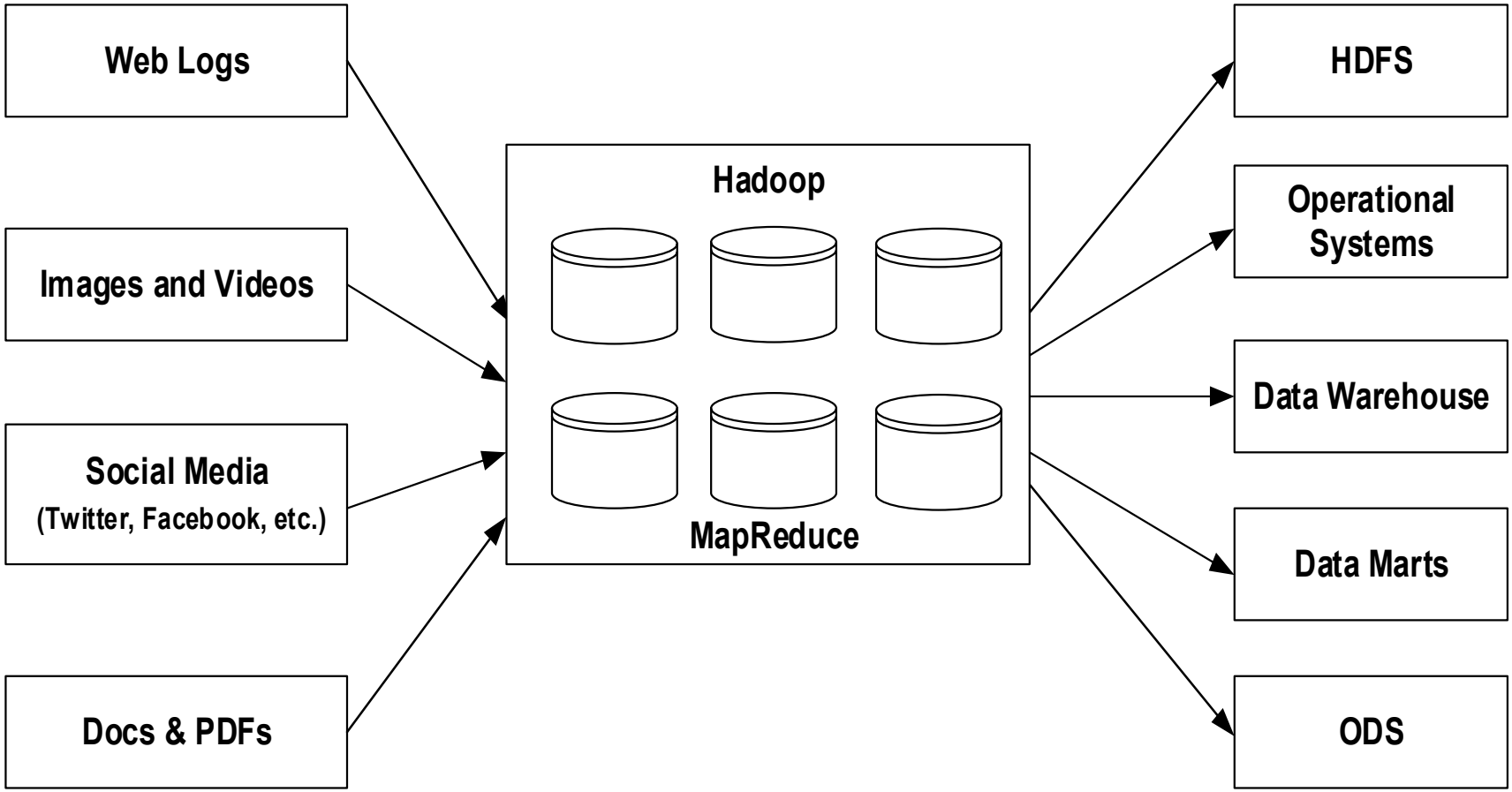
Why Big Data?


More Data
More Accurate Analysis
More Confidence in decision making
Greater operational efficiencies, Cost reduction,
Time reduction, New product development, Optimized offerings, etc.

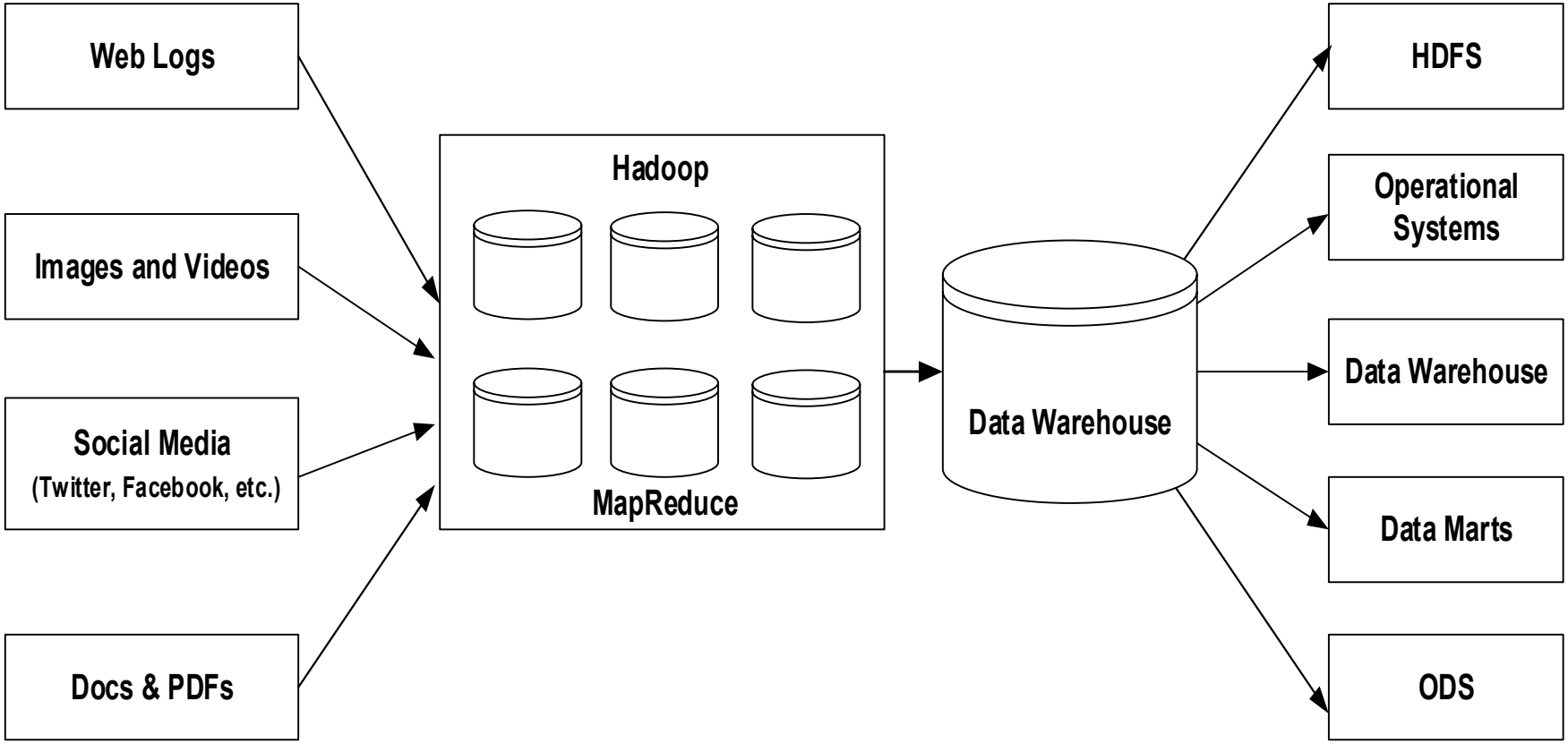
A Typical Data Warehouse Environment



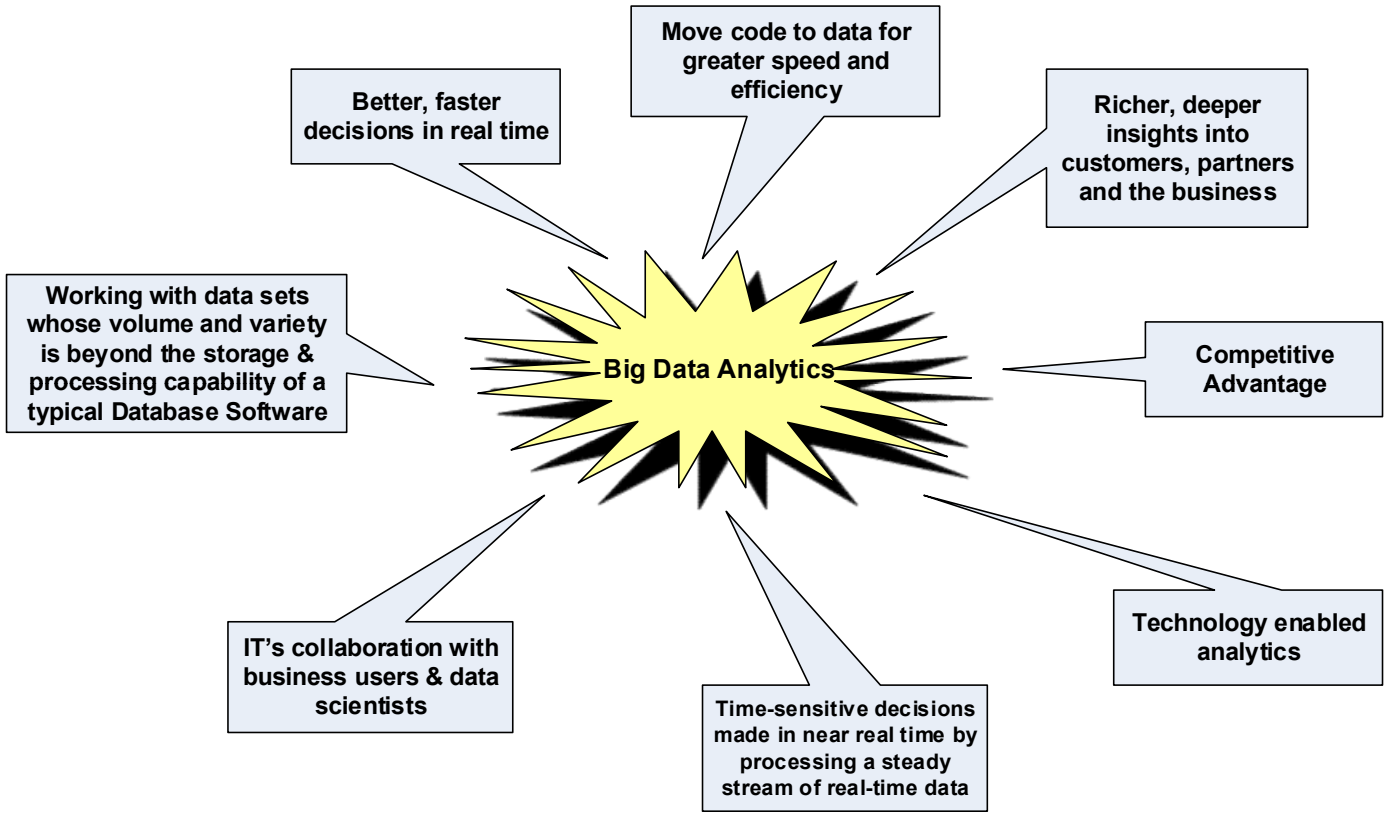
A Typical Hadoop Environment



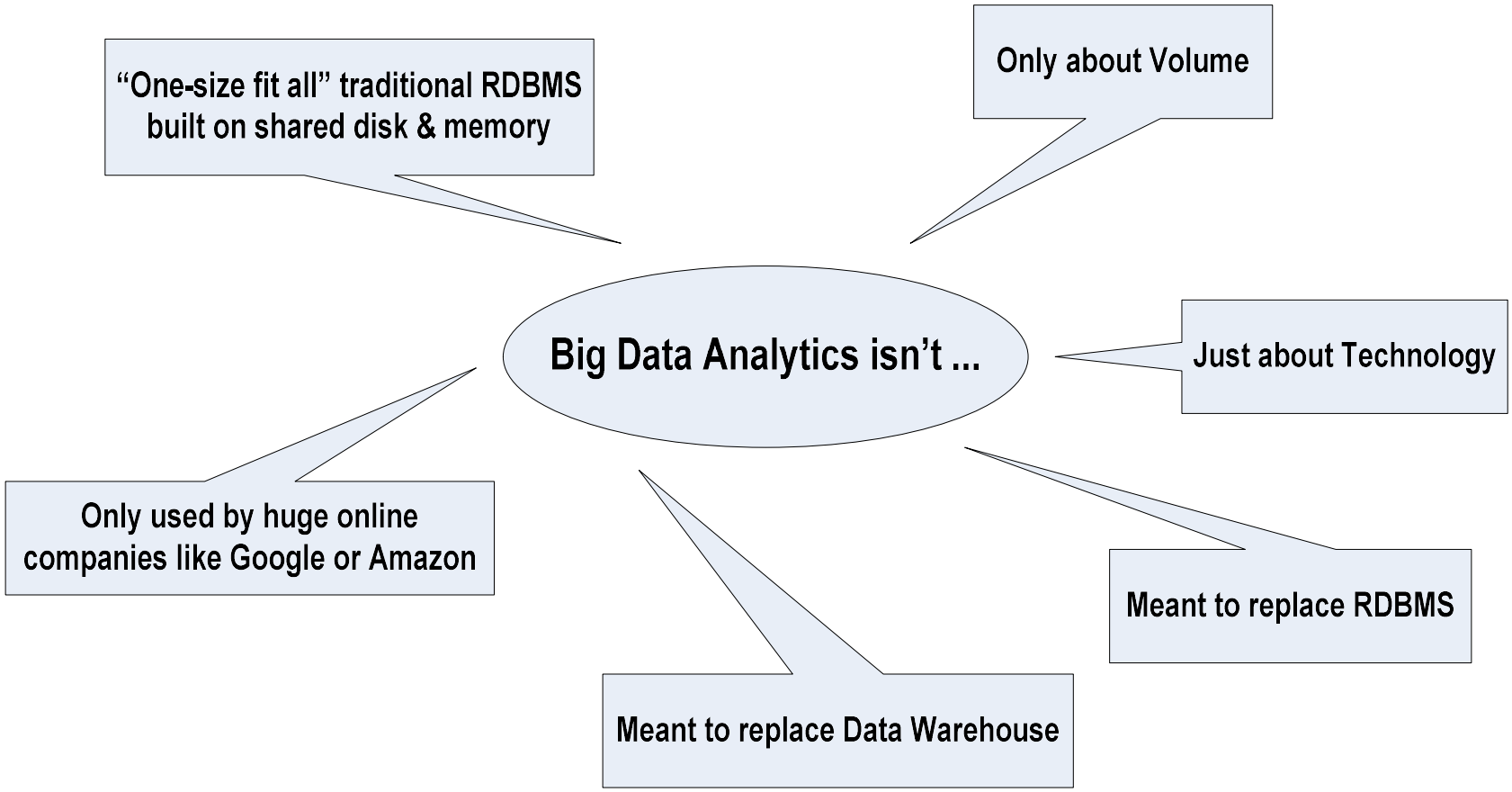
Co-existence of Big Data and Data Warehouse



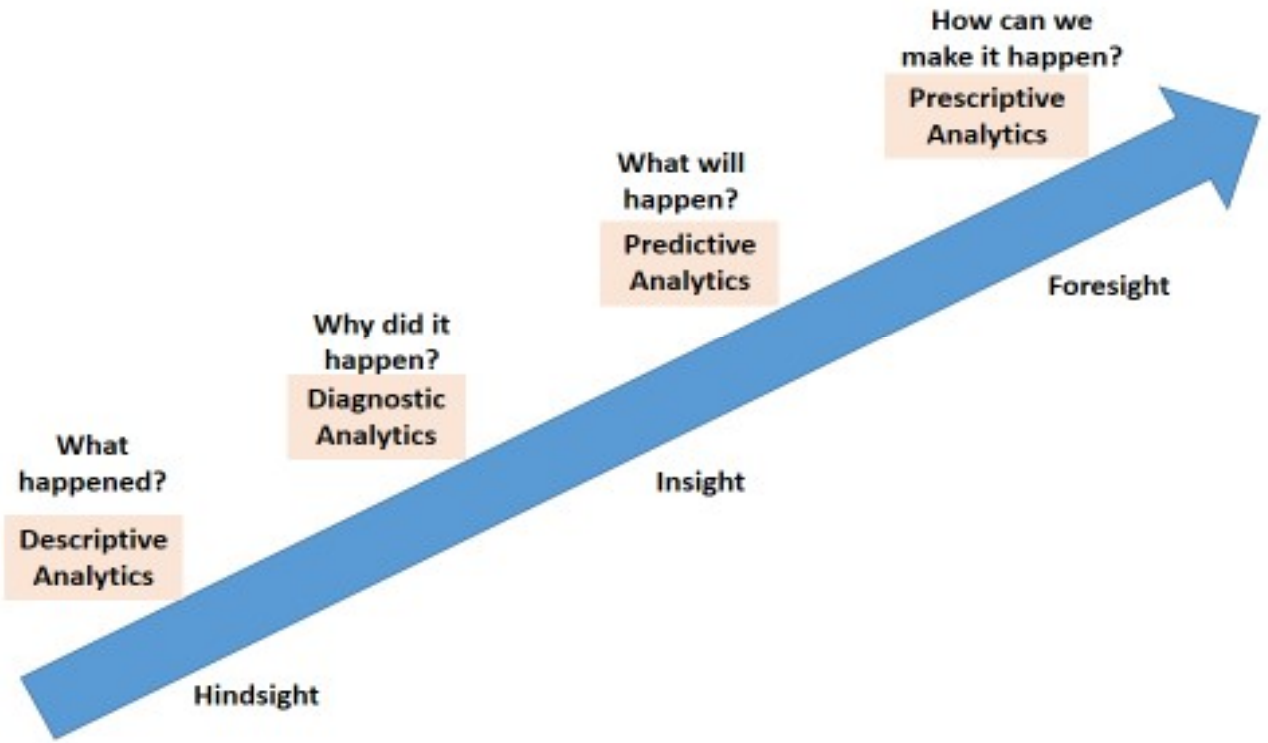
What is Big Data Analytics?



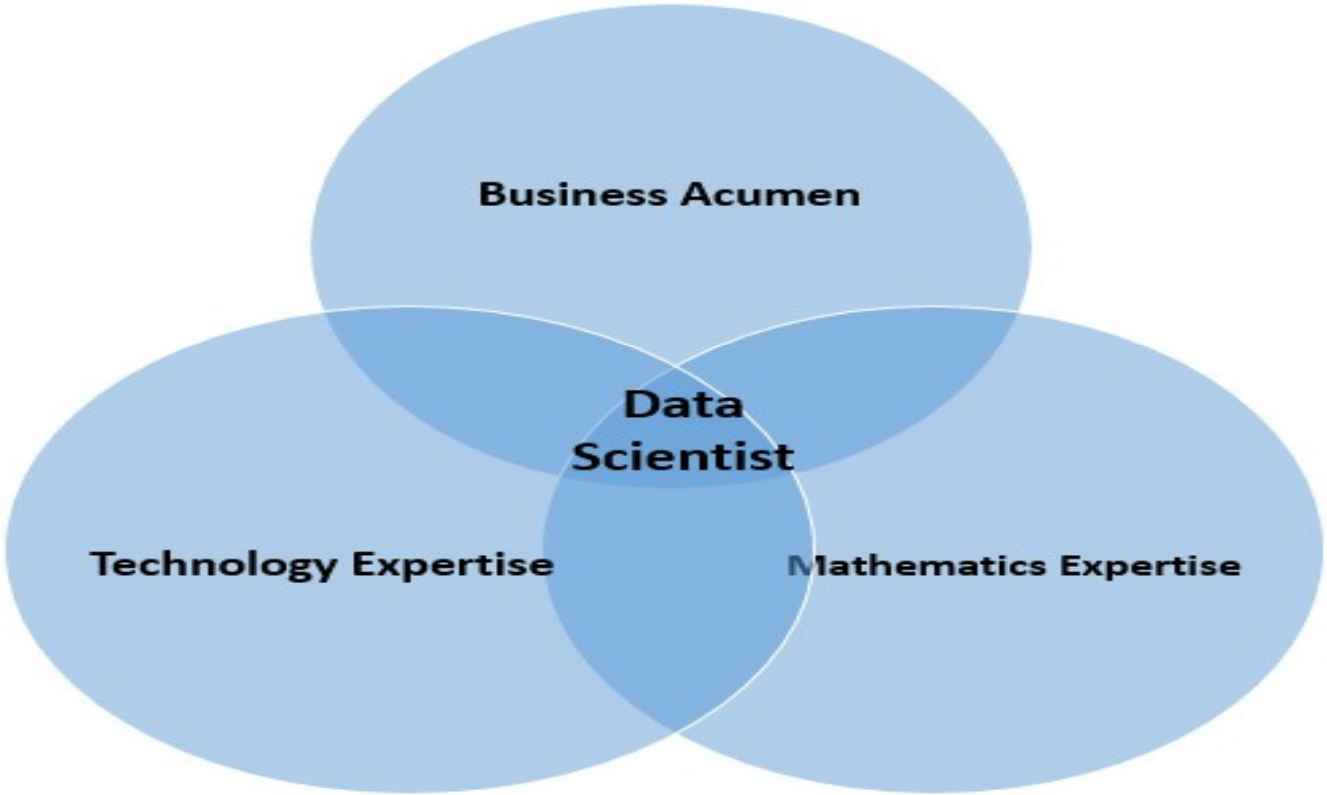
What Big Data Analytics isn't?



Analytics 1.0, 2.0 and 3.0



Data Scientist

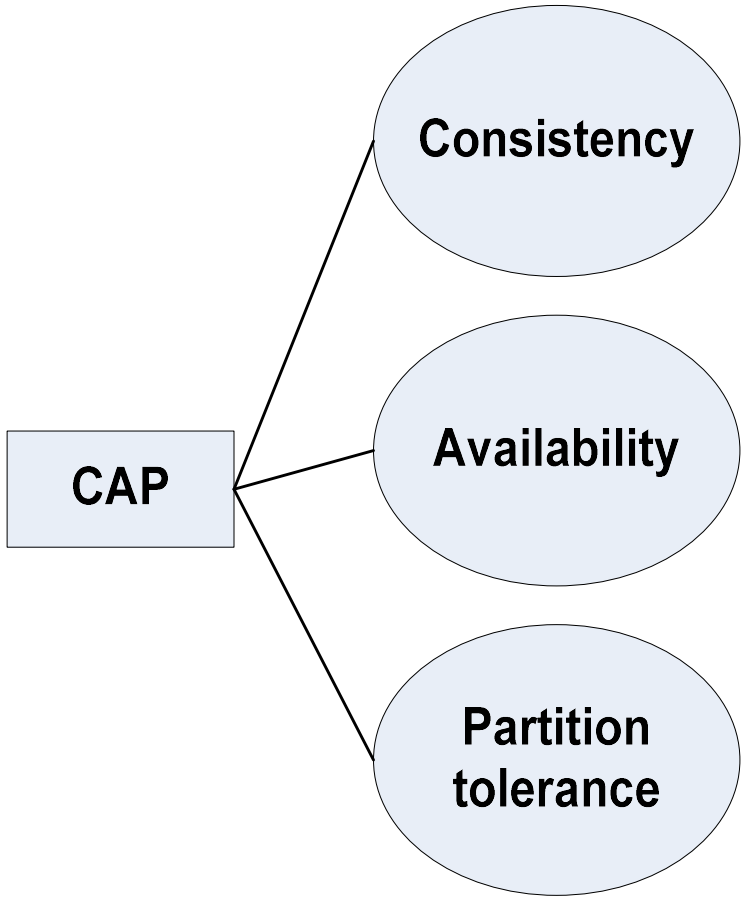


Terminologies Used in Big data Environments

- In-Memory Analytics
- In-Database Processing
- Massively Parallel Processing
- Parallel System
- Distributed System
- Shared Nothing Architecture

Brewer's CAP

The three letters in CAP refer to three desirable properties of distributed systems with replicated data: **consistency** (among replicated copies), **availability** (of the system for read and write operations) and **partition tolerance** (in the face of the nodes in the system being partitioned by a network fault).



- **Consistency**
  Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions.
- A guarantee that every node in a distributed cluster returns the same, most recent and a successful write. Consistency refers to every client having the same view of the data.
- There are various types of consistency models. Consistency in CAP refers to sequential consistency, a very strong form of consistency.

•Availability

•Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed.

•Every non-failing node returns a response for all the read and write requests in a reasonable amount of time.

•The key word here is "every". In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.

•Partition Tolerance

• Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other.

•That means, the system continues to function and upholds its consistency guarantees in spite of network partitions.

•Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

CA(Consistency and Availability)-

The system prioritizes availability over consistency and can respond with possibly stale data.

Example databases: Cassandra, CouchDB, Riak, Voldemort.

AP(Availability and Partition Tolerance)-

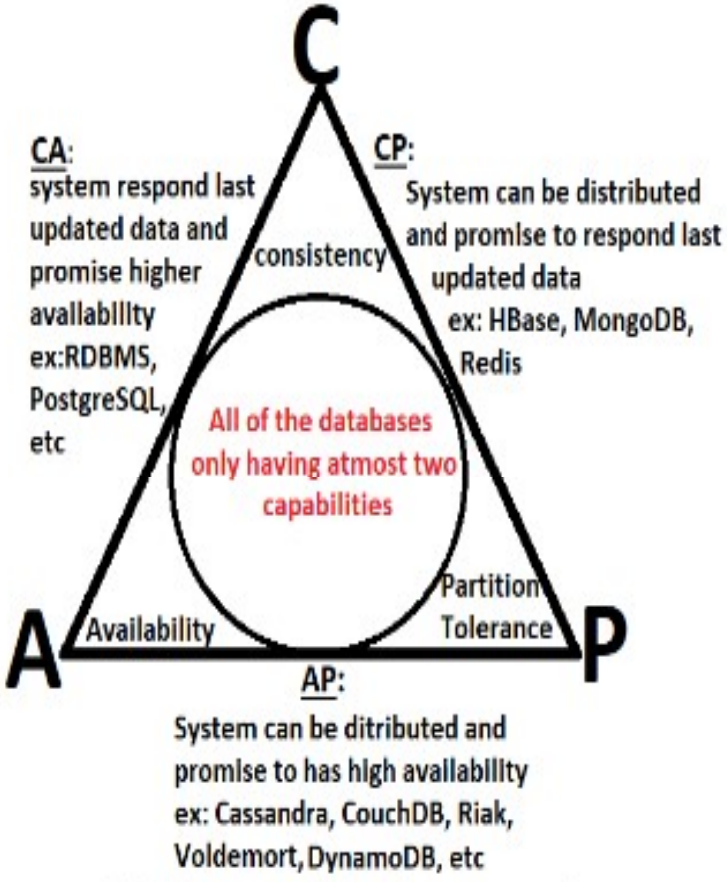
The system prioritizes availability over consistency and can respond with possibly stale data. The system can be distributed across multiple nodes and is designed to operate reliably even in the face of network partitions.

Example databases: Amazon DynamoDB, Google Cloud Spanner.

AP(Availability and Partition Tolerance)-

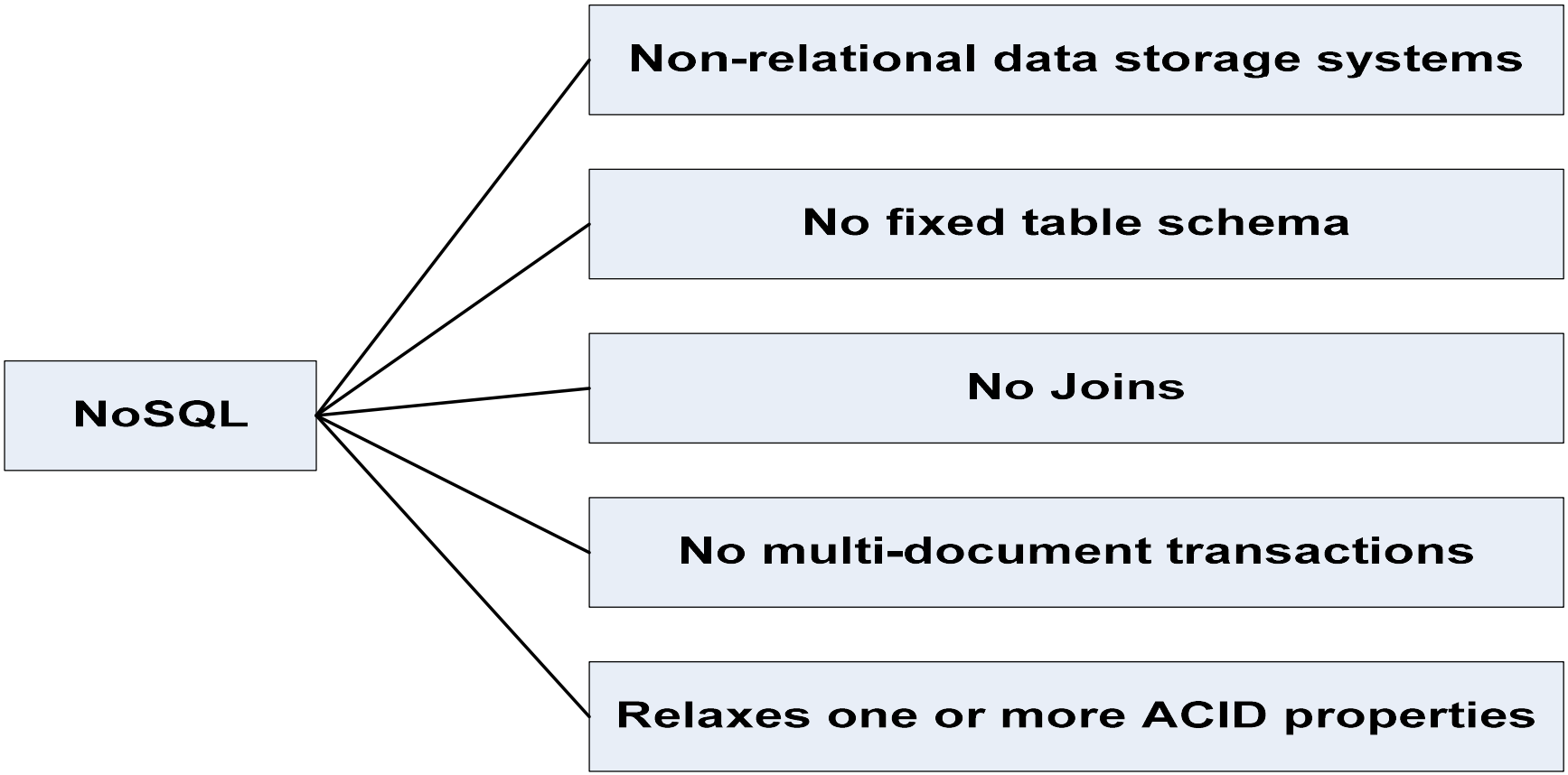
The system prioritizes availability over consistency and can respond with possibly stale data. The system can be distributed across multiple nodes and is designed to operate reliably even in the face of network partitions.

Example databases: Amazon DynamoDB, Google Cloud Spanner.



CAP theorem with databases examples

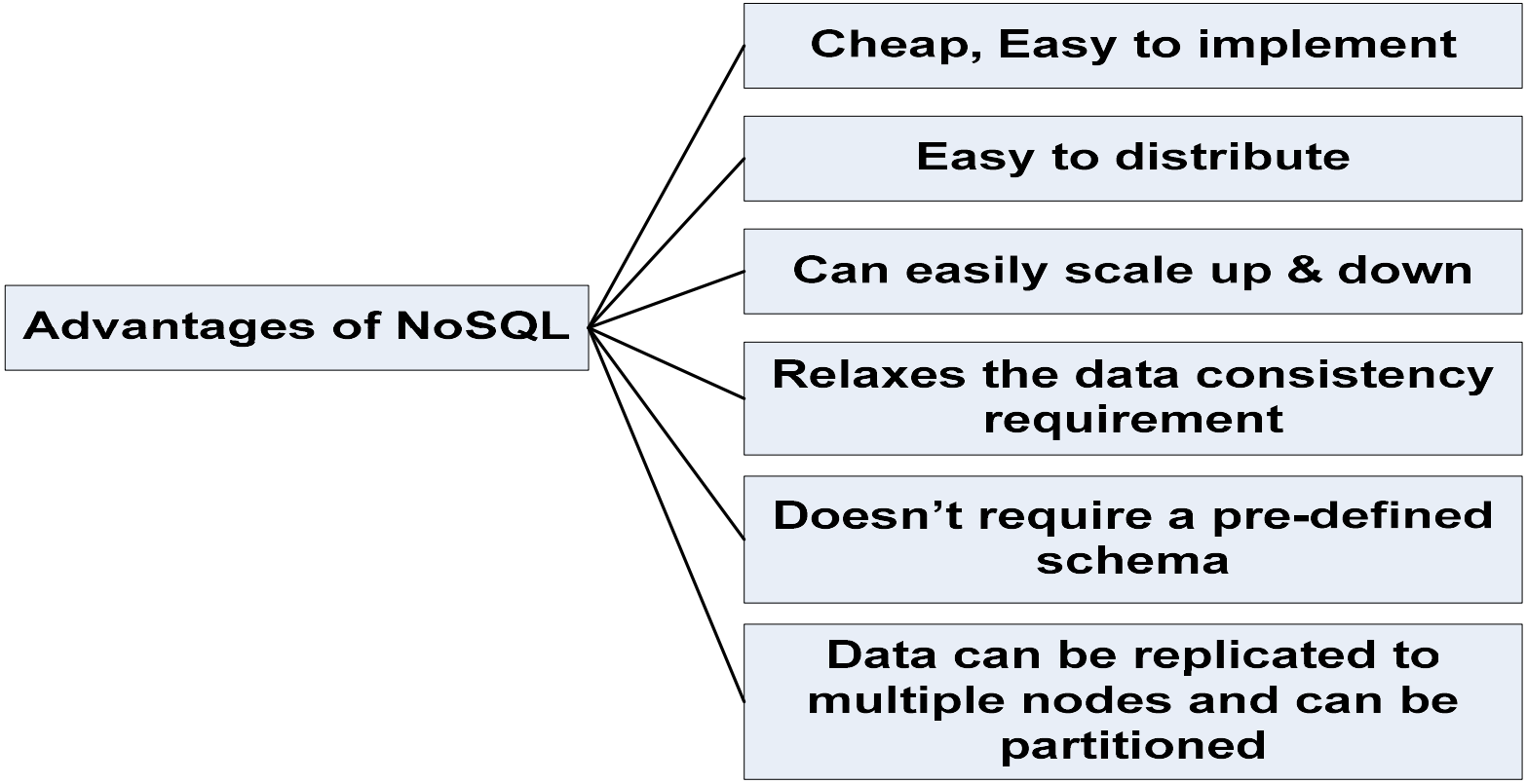
What is NoSQL?



Types of NoSQL

Key value data store	Column-oriented data store	Document data store	Graph data store
• Riak • Redis • Membase	• Cassandra • HBase • HyperTable	• MongoDB • CouchDB • RavenDB	• InfiniteGraph • Neo4 • Allegro Graph

Advantages of NoSQL



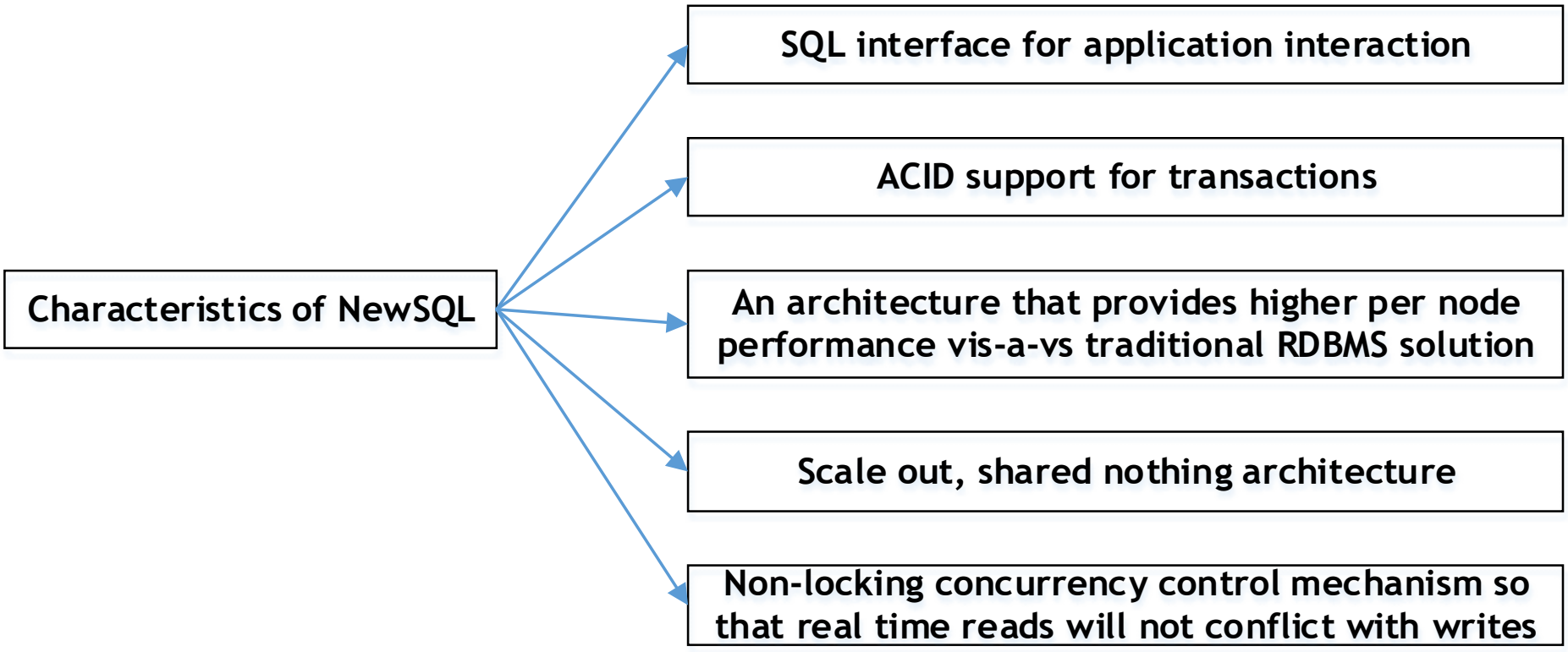
NoSQL Vendors

Company	Product	Most widely used by
Amazon	DynamoDB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, eBay
Google	BigTable	Adobe Photoshop

SQL Vs. NoSQL

SQL	NoSQL
Relational database	Non-relational, distributed database
Relational model	Model-less approach
Pre-defined schema	Dynamic schema for unstructured data
Table based databases	Document-based or graph-based or wide column store or key-value pairs databases
Vertically scalable (by increasing system resources)	Horizontally scalable (by creating a cluster of commodity machines)
Uses SQL	Uses UnQL (Unstructured Query Language)
Not preferred for large datasets	Largely 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 Brewer's CAP theorem
Excellent support from vendors	Relies heavily on community support
Supports complex querying and data keeping needs	Does not have good support for complex querying
Can be configured for strong consistency	Few support strong consistency (e.g., MongoDB), few others can be configured for eventual consistency (e.g., Cassandra)
Examples: Oracle, DB2, MySQL, MS SQL, PostgreSQL, etc.	MongoDB, HBase, Cassandra, Redis, Neo4j, CouchDB, Couchbase, Riak, etc.

NewSQL


Characteristics of NewSQL
SQL interface for application interaction
ACID support for transactions
An architecture that provides higher per node performance vis-a-vs traditional RDBMS solution
Scale out, shared nothing architecture
Non-locking concurrency control mechanism so that real time reads will not conflict with writes

SQL Vs. NoSQL Vs. NewSQL

	SQL	NoSQL	NewSQL
Adherence to ACID properties	Yes	No	Yes
OLTP/OLAP	Yes	No	Yes
Schema rigidity **Adherence to data model**	Yes Adherence to relational model	No	Maybe
Data Format Flexibility	No	Yes	Maybe
Scalability	Scale up Vertical Scaling	Scale out Horizontal Scaling	Scale out
Distributed Computing	Yes	Yes	Yes
Community Support	Huge	Growing	Slowly growing

What is Hadoop

Hadoop is:

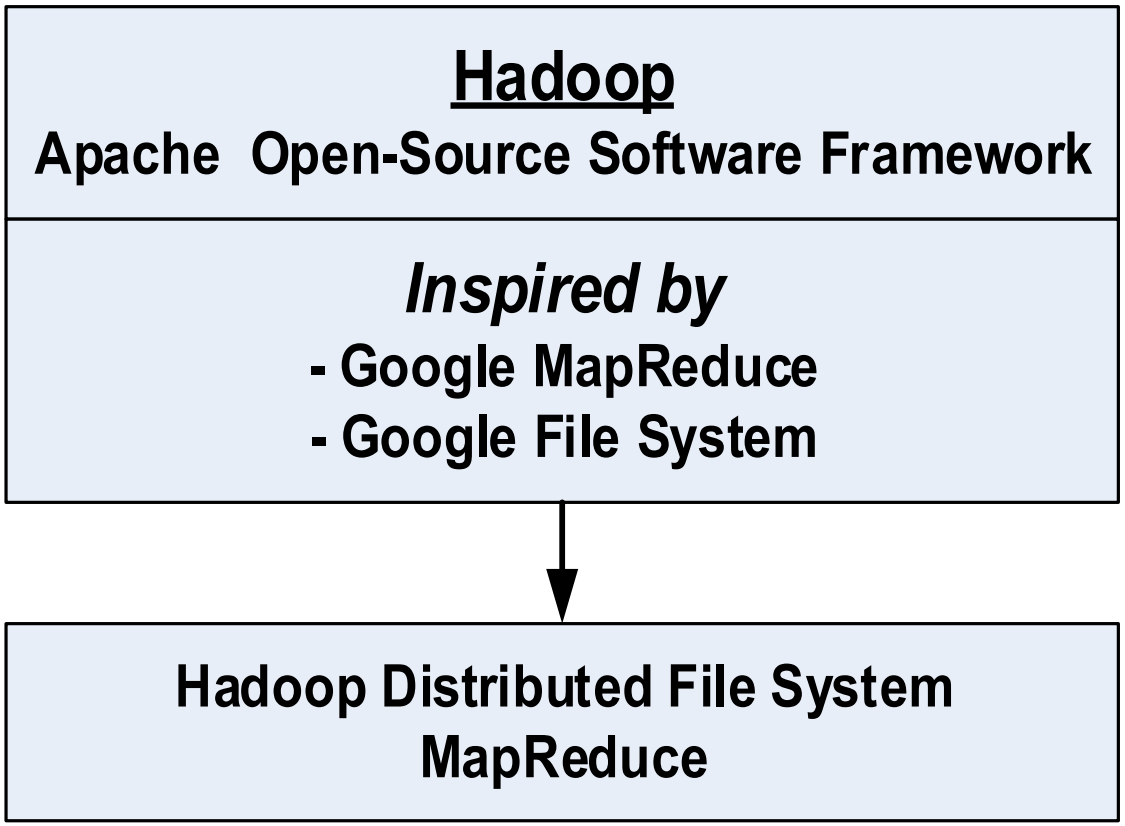
Ever wondered why Hadoop has been and is one of the most wanted technologies!!

The key consideration (the rationale behind its huge popularity) is:

Its capability to handle massive amounts of data, different categories of data – fairly quickly.

The other considerations are :

Hadoop


Hadoop
Apache Open-Source Software Framework
Inspired by
- Google MapReduce
- Google File System
Hadoop Distributed File System
MapReduce

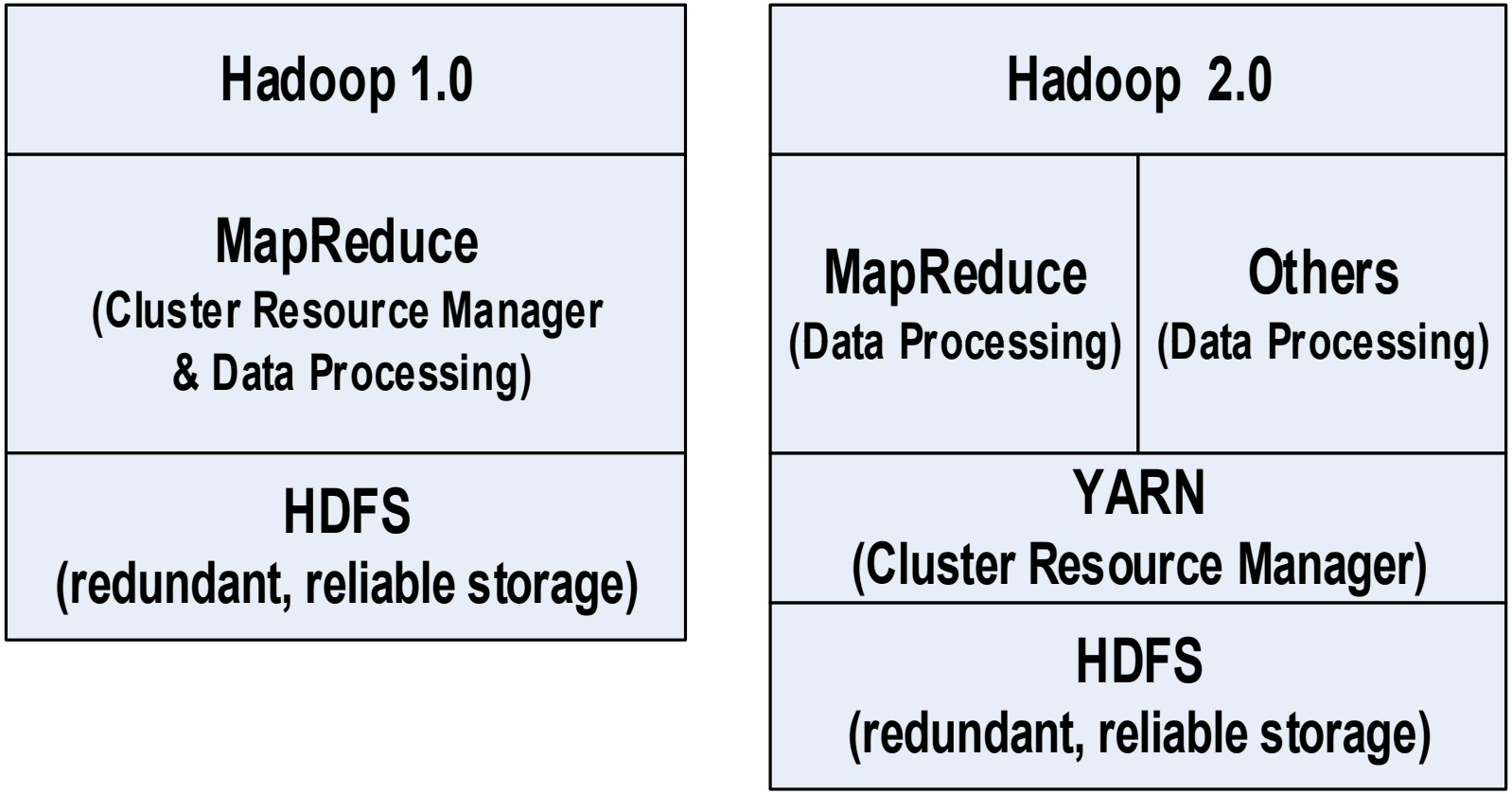
Key Advantages of Hadoop

- Stores data in its native format
- Scalable
- Cost-effective
- Resilient to failure
- Flexibility
- Fast

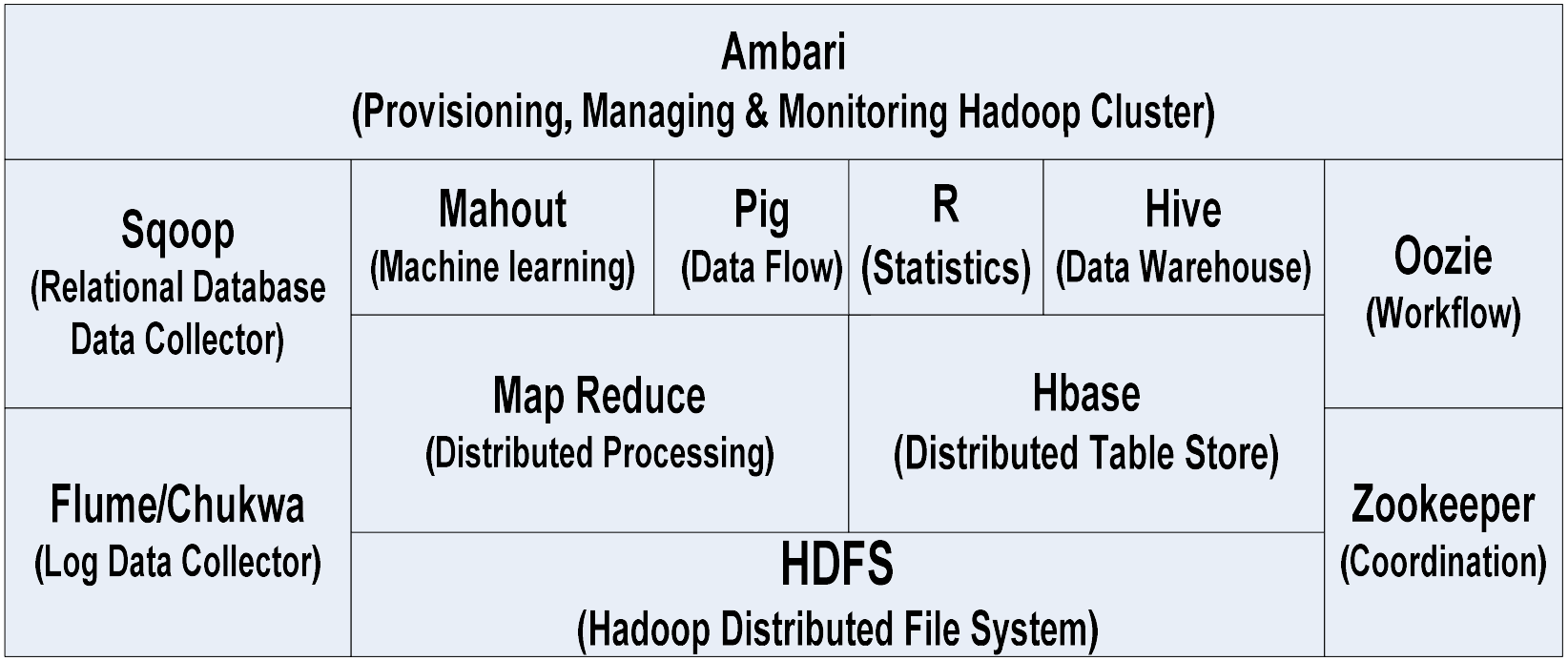
Features of Hadoop

- Fault Tolerance
- Highly Scalable
- Easy Programing
- Huge and Flexible
- Low Cost

Versions of Hadoop



Hadoop Ecosystem



Hadoop Ecosystem

Components that help with Data Integration are:

1. Sqoop
2. Flume

Components that help with Data Processing are:

1. MapReduce
2. Spark

Components that help with Data Analysis are:

1. Pig
2. Hive
3. Impala

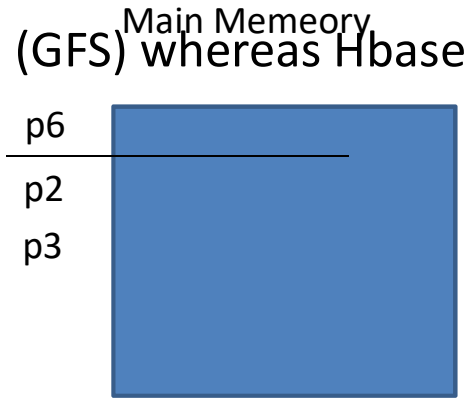
Three Difference between HBase and Hadoop/ HDFS

- HDFS is the file system where as HBase is a Hadoop database. It is like NTFS and MySQL.

- HDFS is WORM (Write once and read multiple times or many times). Latest versions supports appending of data but this feature is rarely used. However HBase supports real time random read and write.
- HDFS is based on Google File System (GFS) whereas Hbase is based on Google Big Table.

C:/

D:/



Hadoop Ecosystem Components for Data Ingestion

Sqoop:

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

Flume:

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

Hadoop Ecosystem Components for Data Processing

MapReduce:

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

Spark:

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

Hadoop ecosystem components for Data Analysis

Pig

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

Hive:

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

Impala:

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

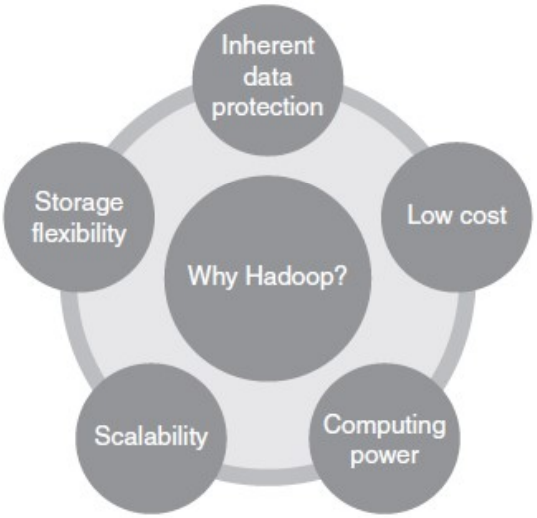
Hadoop Distributions

- Cloudera
- Greenpulm
- Ibm infosphere

Hadoop vs SQL

- Distributed/central
- Scale out/scale up
- Functional programming/queries
- Key value pair/relational data

Why Hadoop


Inherent data protection
Storage flexibility
Low cost
Why Hadoop?
Scalability
Computing power

RDBMS versus HADOOP

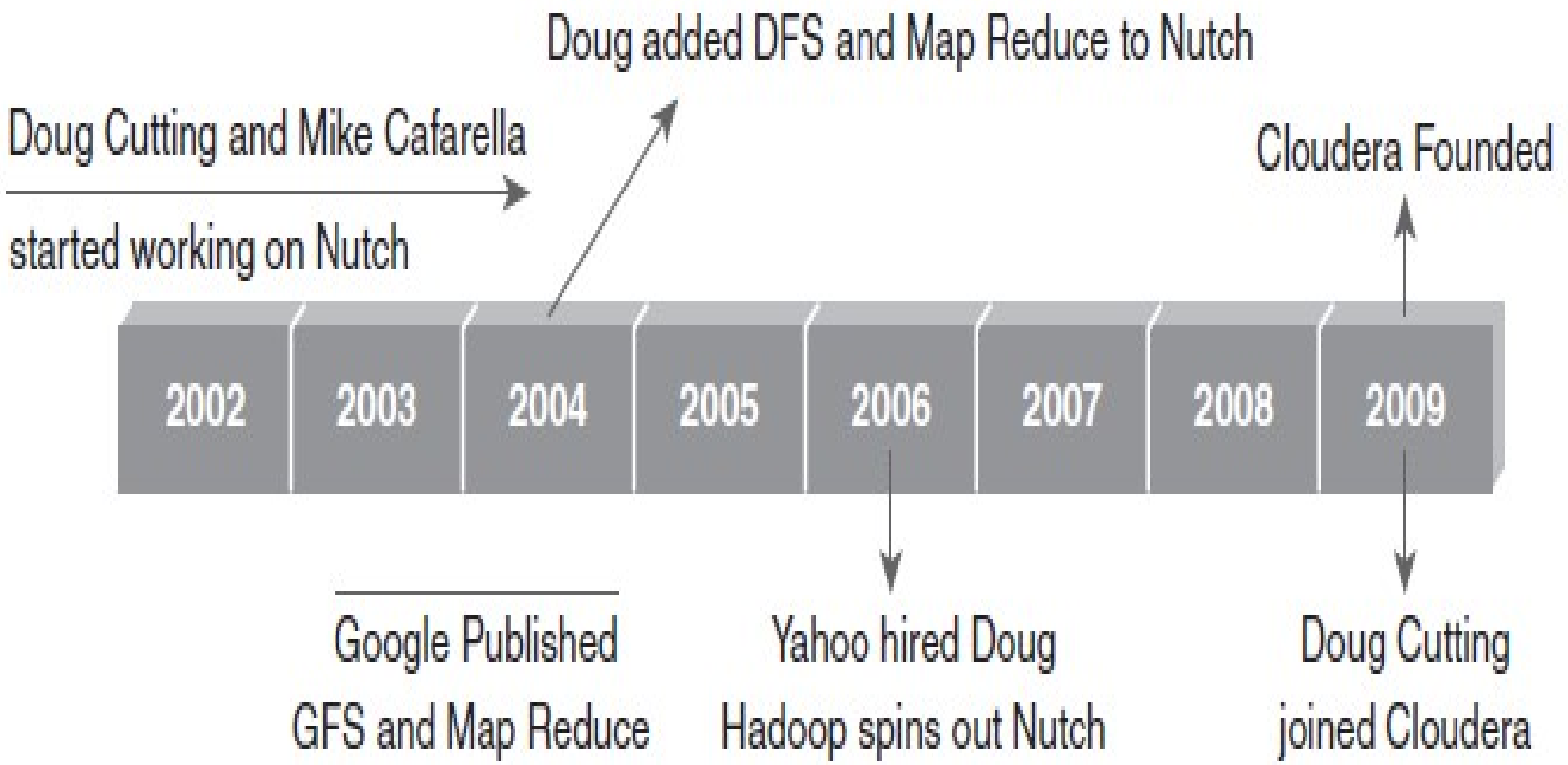
PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical,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

Distributed Computing Challenges

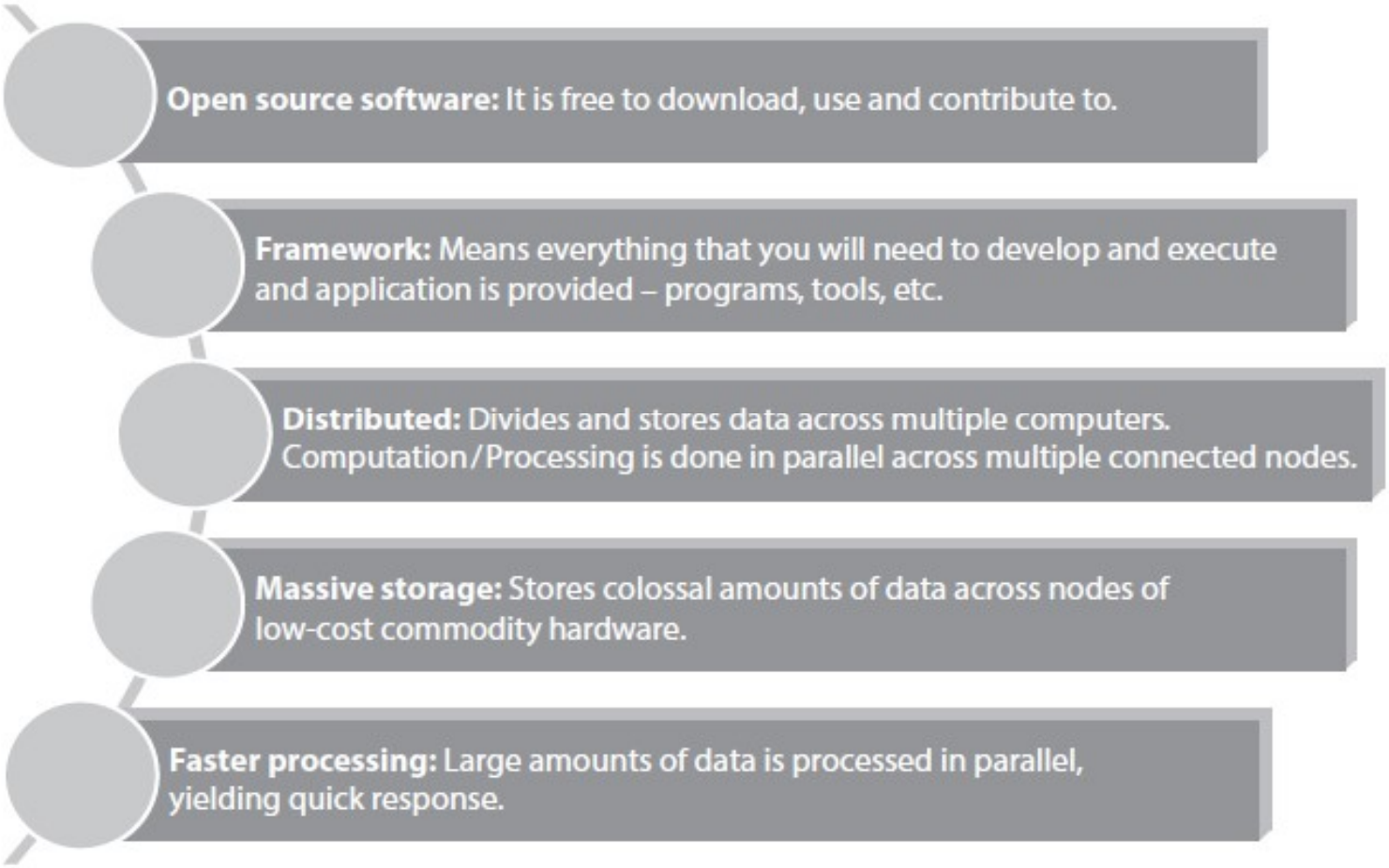
- Hardware Failure

- How to Process This Gigantic Store of Data?

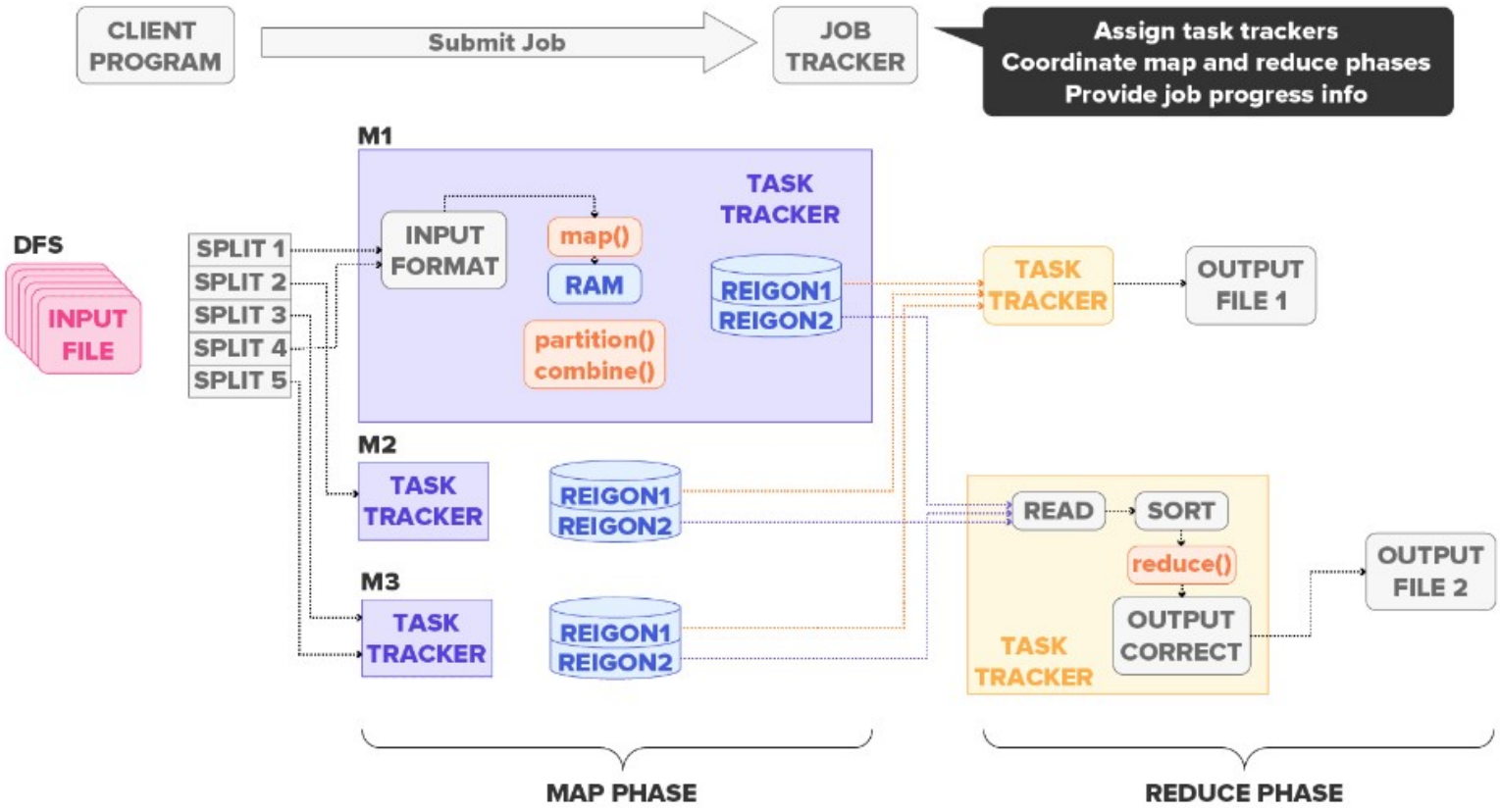
History of Hadoop



Key Aspects of Hadoop



Hadoop Architecture

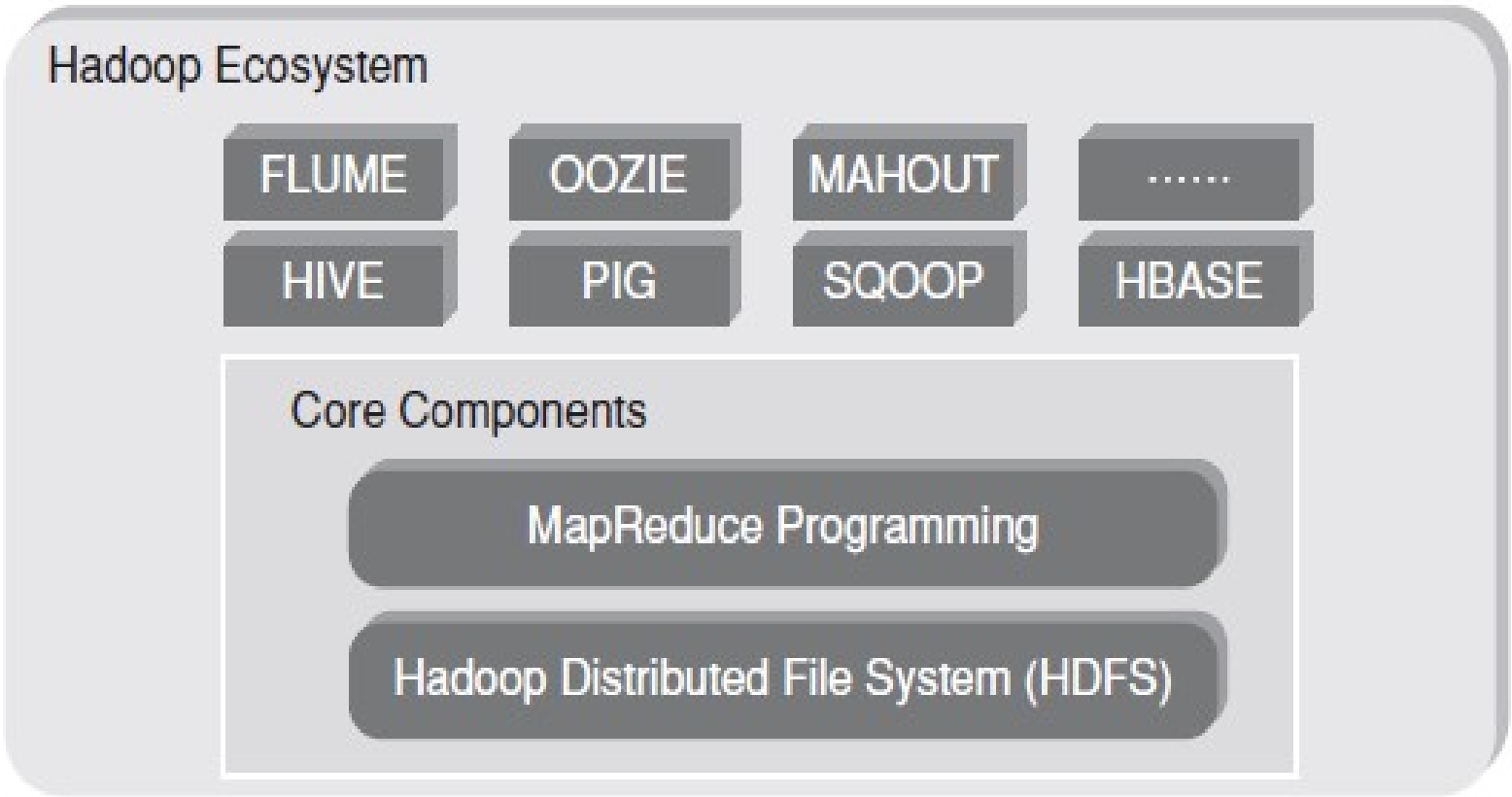


- The provided diagram illustrates the architecture of Hadoop, specifically focusing on the MapReduce framework.
- **Components:**
- **Client Program**:
  - Submits the job to the Job Tracker.
- **Job Tracker**:
  - Coordinates the map and reduce phases.
  - Assigns tasks to Task Trackers.
  - Provides job progress information.
- **Distributed File System (DFS)**:
  - Stores the input file which is split into multiple parts for processing.

- **Phases:**
- **Map Phase**:
  - **Input File**: The input file is divided into multiple splits (e.g., Split 1, Split 2, etc.).
  - **Task Trackers (M1, M2, M3)**: Each split is processed by different Task Trackers located on different machines (M1, M2, M3).
  - **Input Format**: Reads the input splits and prepares them for the mapping process.
  - **Map Function**: Processes the input data and generates intermediate key-value pairs.
  - **Partition & Combine**: The data is partitioned and combined for optimization before sending it to the reducers.
  - **Intermediate Data Storage**: The intermediate data is stored in regions (e.g., Region1, Region2).

- **Reduce Phase**:
  - **Task Trackers**: Task Trackers fetch the intermediate data from the map phase.
  - **Read & Sort**: The data is read and sorted.
  - **Reduce Function**: Processes the sorted data and generates the final output.
  - **Output File**: The final output is stored in output files (e.g., Output File 1, Output File 2).
- **Data Flow:**
- The client program submits the job to the Job Tracker.
- The Job Tracker assigns the task to different Task Trackers located on different machines.
- Each Task Tracker processes a portion of the input file in the Map Phase.
- The output of the Map Phase is stored in intermediate storage regions.
- The Reduce Phase fetches the intermediate data, processes it, and stores the final output in output files.

Hadoop Components



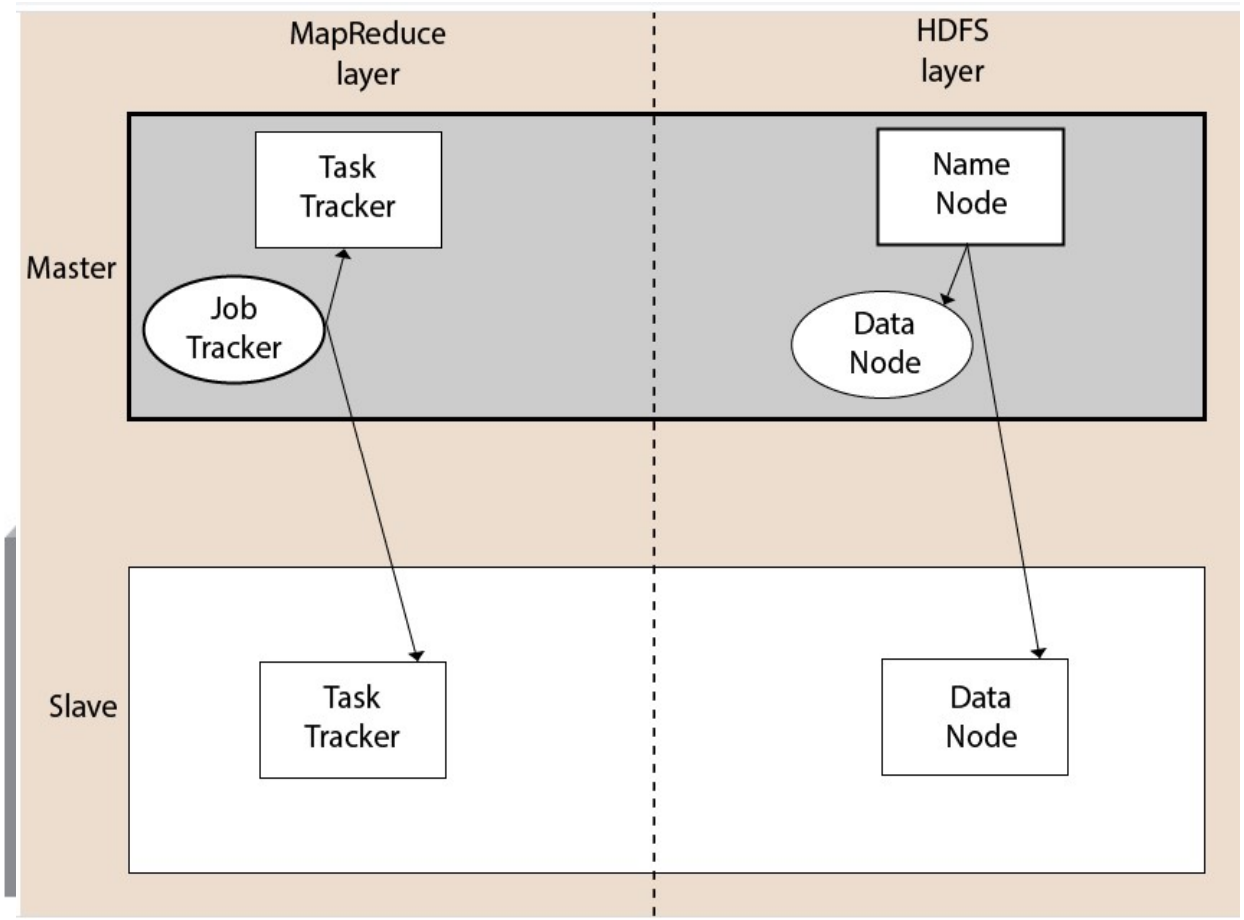
Hadoop Components

Hadoop Core Components:

HDFS:
(a) Storage component.
(b) Distributes data across several nodes.
(c) Natively redundant.

MapReduce:
(a) Computational framework.
(b) Splits a task across multiple nodes.
(c) Processes data in parallel.

Hadoop High Level Architecture



A
b

B
c

C
a

NameNode

- It is a single master server exist in the HDFS cluster.
- As it is a single node, it may become the reason of single point failure.
- It manages the file system namespace by executing an operation like the opening, renaming and closing the files.
- It simplifies the architecture of the system.

DataNode

- The HDFS cluster contains multiple DataNodes.
- Each DataNode contains multiple data blocks.
- These data blocks are used to store data.
- It is the responsibility of DataNode to read and write requests from the file system's clients.
- It performs block creation, deletion, and replication upon instruction from the NameNode.

Job Tracker

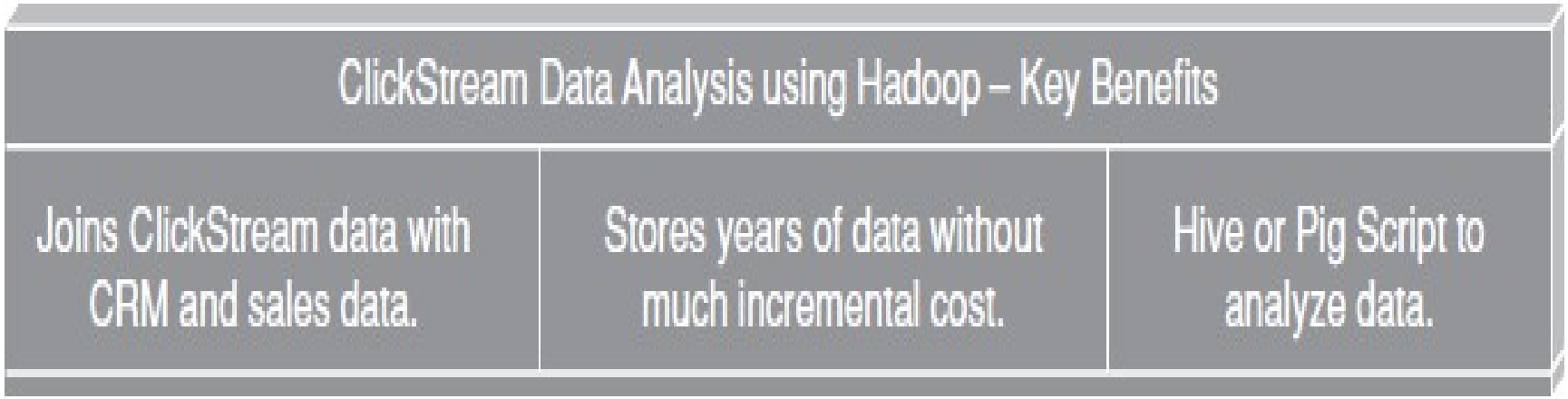
- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.
- In response, NameNode provides metadata to Job Tracker.

Task Tracker

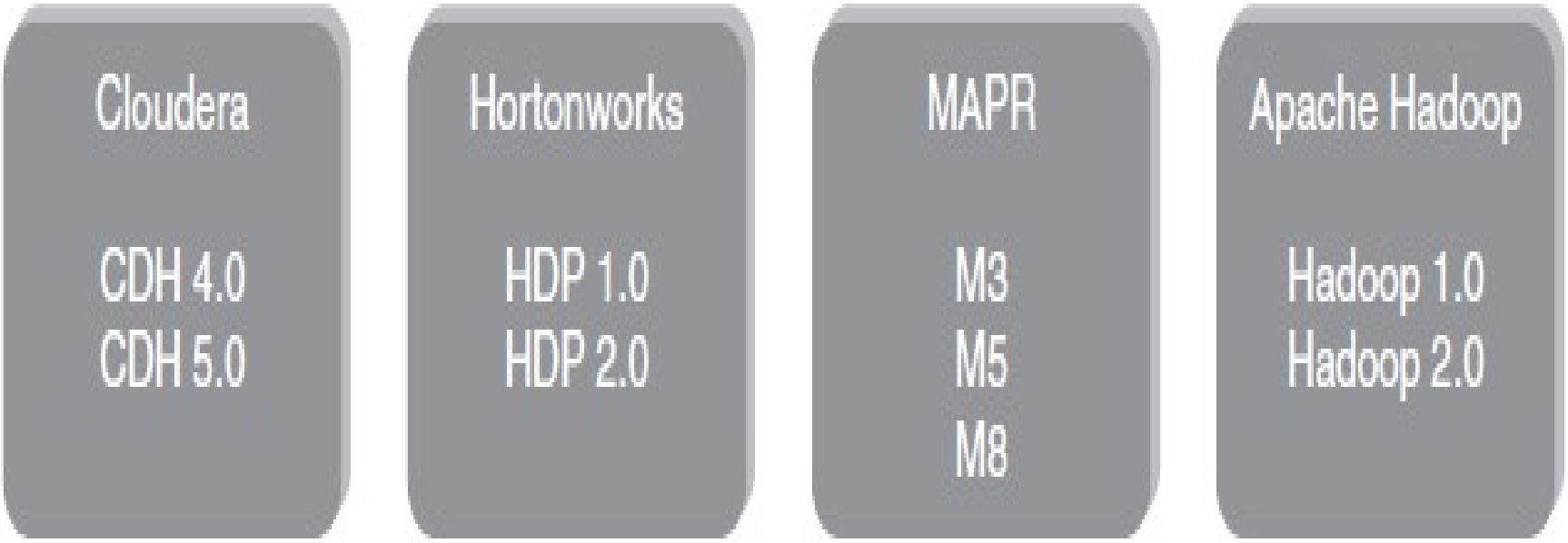
- It works as a slave node for Job Tracker.
- It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper

ClickStream Data Analysis

ClickStream data (mouse clicks) helps you to understand the purchasing behavior of customers. ClickStream analysis helps online marketers to optimize their product web pages, promotional content, etc. to improve their business.



Hadoop Distributors



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 (gigabytes and larger).

8. Sits on top of native file system such as ext3 and ext4, which is described

HDFS Daemons

NameNode:

- Single NameNode per cluster.
- Keeps the metadata details

DataNode:

- Multiple DataNode per cluster
- Read/Write operations

SecondaryNameNode:

- Housekeeping Daemon

HDFS: Design and Goals

- **Architecture:**
- **Master-Slave Architecture:** HDFS employs a master-slave architecture with two main components: the NameNode (master) and DataNodes (slaves). The NameNode manages the file system namespace and metadata, while DataNodes store the actual data blocks.
- **NameNode:** This server maintains the metadata for the entire file system, including file-to-block mapping, permissions, and directory structure. It does not store the actual data but keeps track of where data blocks are stored on the DataNodes.
- **DataNodes:** These servers store the data blocks and are responsible for serving read and write requests from the client. They periodically send heartbeat signals and block reports to the NameNode to confirm their status and data integrity.

- **Data Storage:**
- **Block-based Storage:** Files in HDFS are split into fixed-size blocks (typically 128 MB or 256 MB), which are distributed across DataNodes. This block-based storage allows HDFS to efficiently manage large files and provide high-throughput access.
- **Replication:** Each data block is replicated across multiple DataNodes (typically three replicas) to ensure fault tolerance and data reliability. The replication factor can be adjusted based on the desired level of fault tolerance and data redundancy.

- **Data Access:**
- **Write Once, Read Many:** HDFS is optimized for write-once and read-many operations, making it well-suited for large-scale data processing tasks such as batch processing and data analytics.
- **Sequential Access:** HDFS is designed for high-throughput data access, supporting sequential reads and writes rather than random access. This design choice improves performance for large-scale data processing jobs.

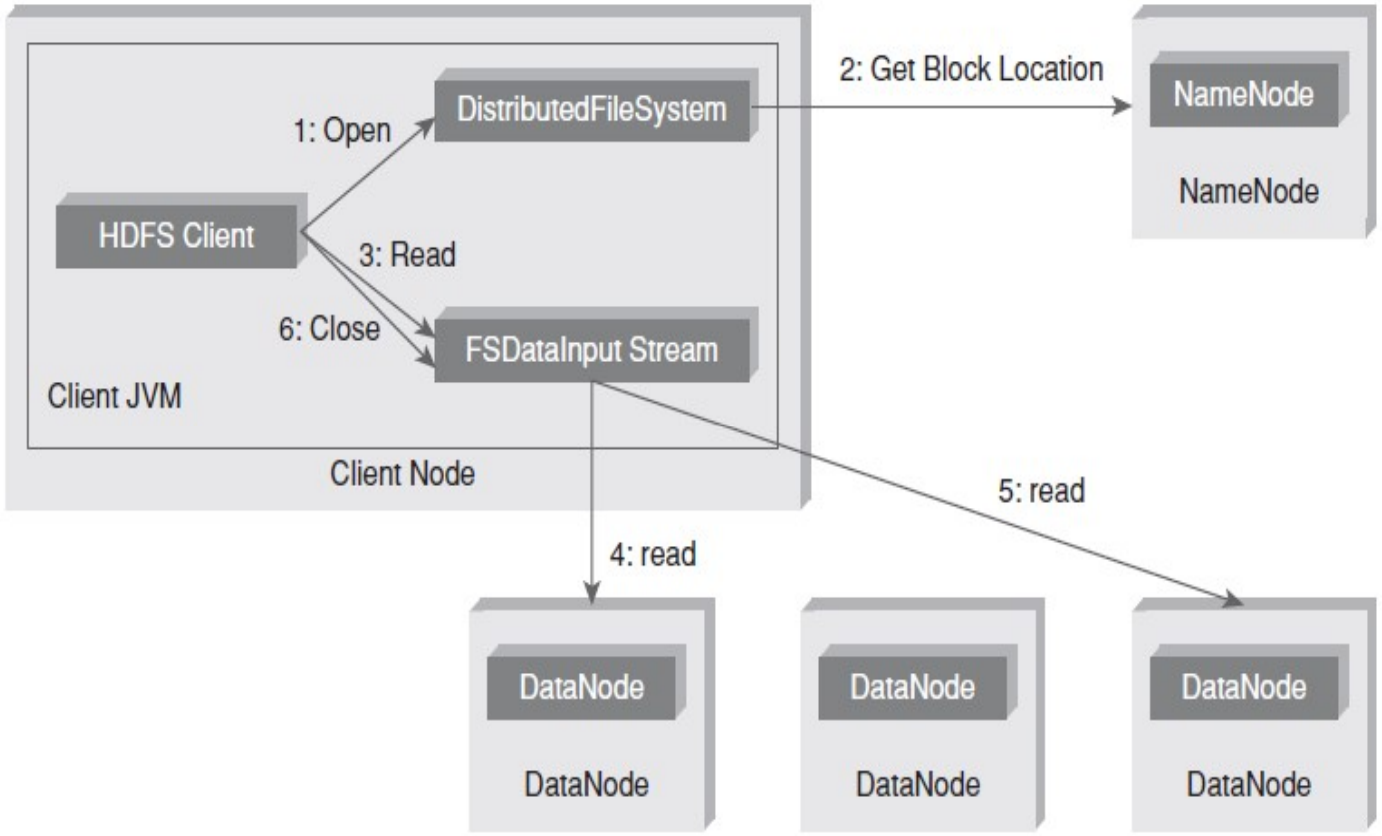
- **Fault Tolerance:**
- **Automatic Recovery:** In the event of a DataNode failure, HDFS automatically re-replicates the lost data blocks to other available DataNodes to maintain the desired replication factor and ensure data availability.
- **Heartbeat and Block Reports:** DataNodes regularly send heartbeat signals and block reports to the NameNode, enabling the system to detect failures and perform necessary recovery actions.

Goals

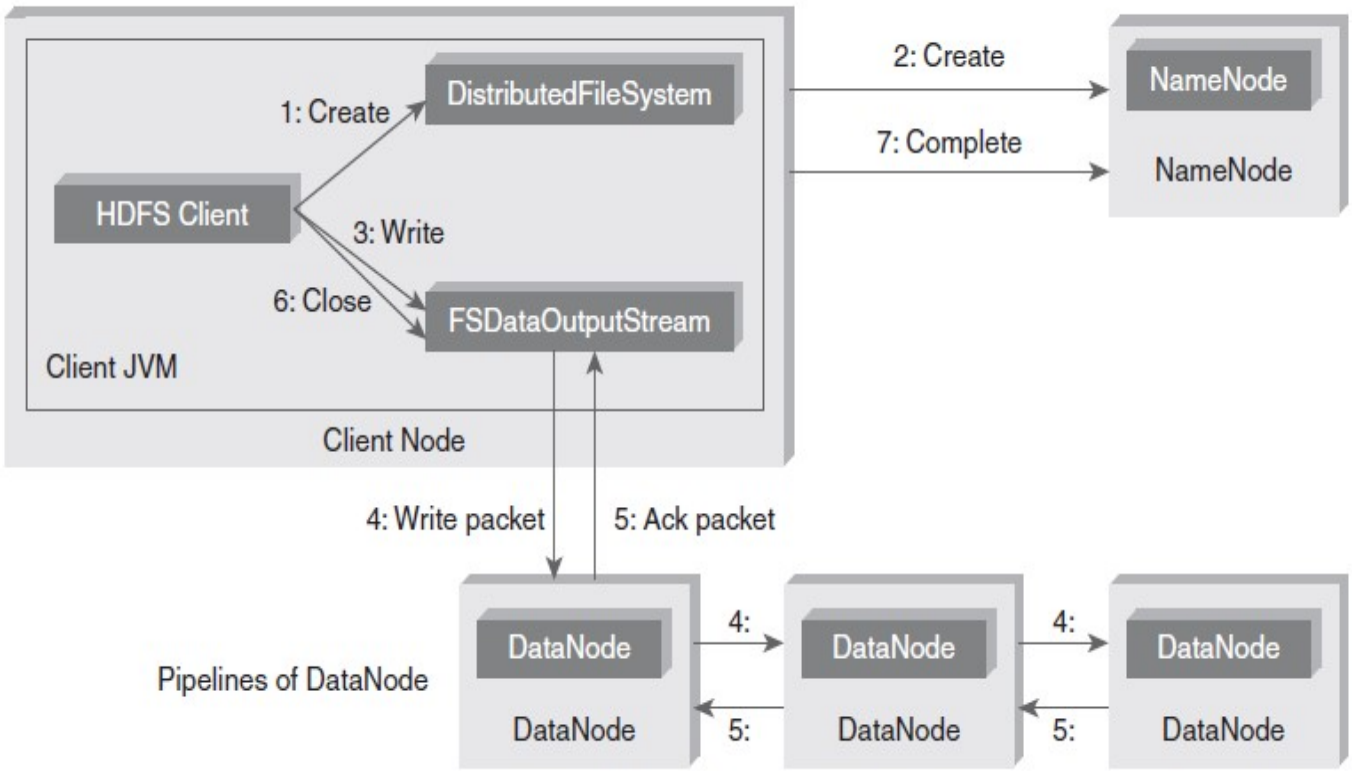
- The primary goals of HDFS are to provide:
- **Scalability:** HDFS is designed to scale horizontally, allowing the addition of more nodes to the cluster as data volume and processing requirements increase. This scalability ensures that the system can handle petabytes of data efficiently.
- **Fault Tolerance:** HDFS aims to ensure data reliability and availability through replication and automatic recovery mechanisms. By replicating data blocks across multiple DataNodes, HDFS can tolerate hardware failures and provide uninterrupted access to data.

- **High Throughput:** The system is optimized for high-throughput data access, making it suitable for applications that require the processing of large datasets. HDFS supports parallel data processing by distributing data blocks across multiple nodes, enabling efficient computation.
- **Data Locality:** HDFS promotes data locality by scheduling computations on the nodes where the data resides. This approach minimizes data transfer across the network and improves overall performance by reducing latency and network congestion.
- **Simplicity:** The design of HDFS emphasizes simplicity in its architecture and implementation. By focusing on a limited set of features and a straightforward architecture, HDFS provides a robust and easy-to-manage file system for distributed data processing.

Anatomy of File Read

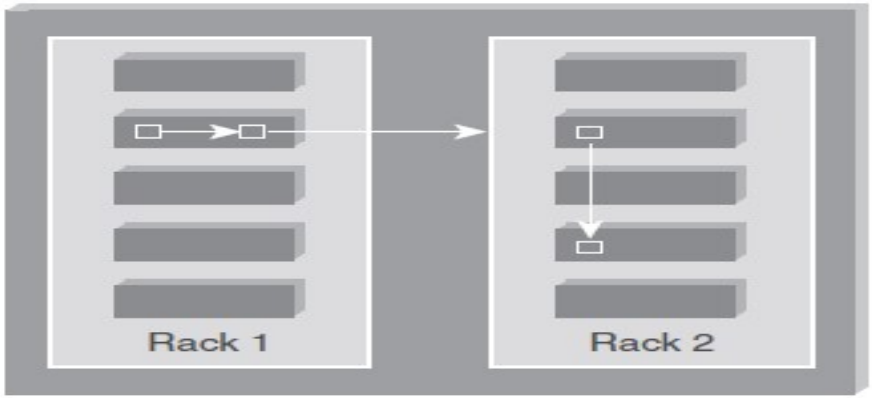


Anatomy of File Write

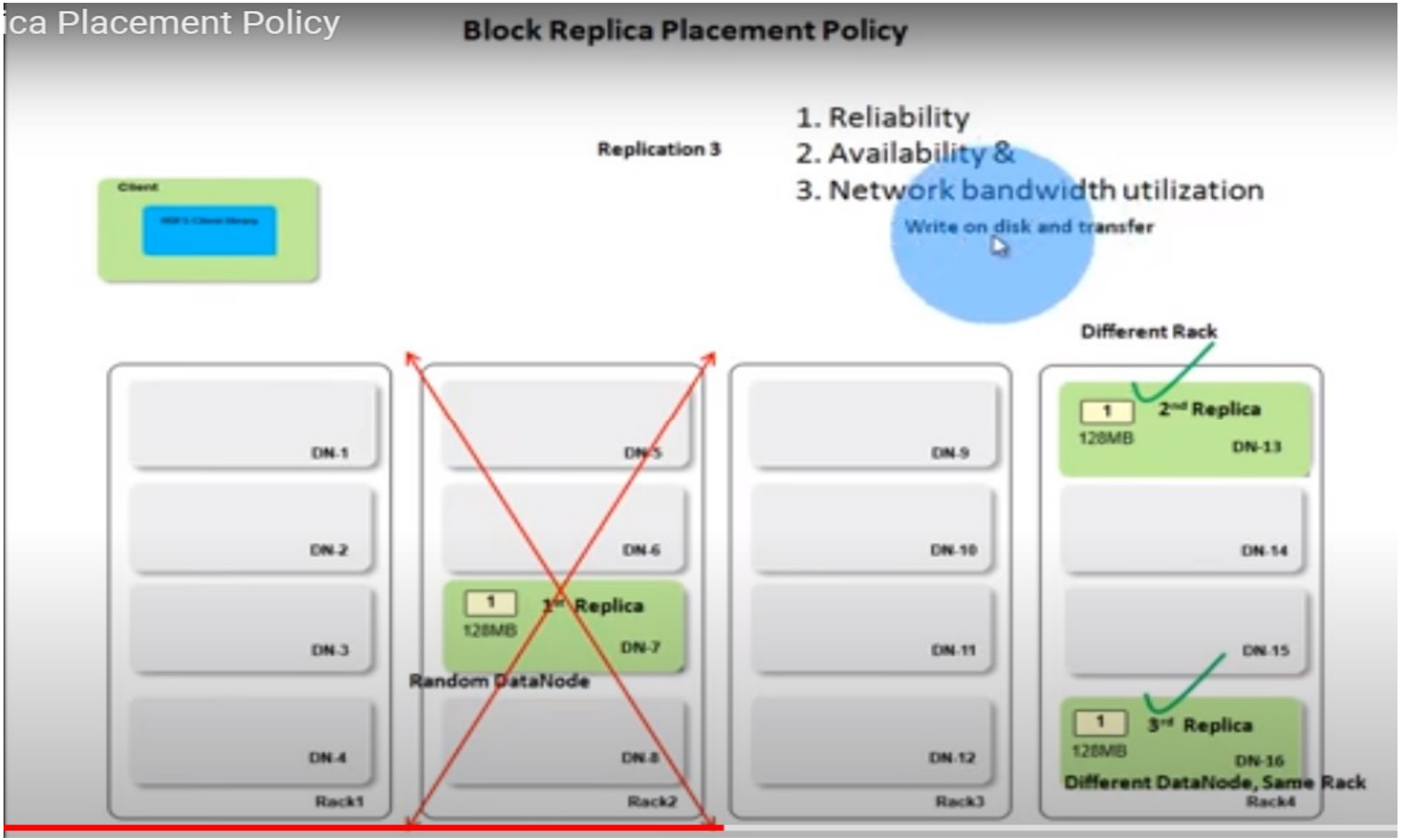


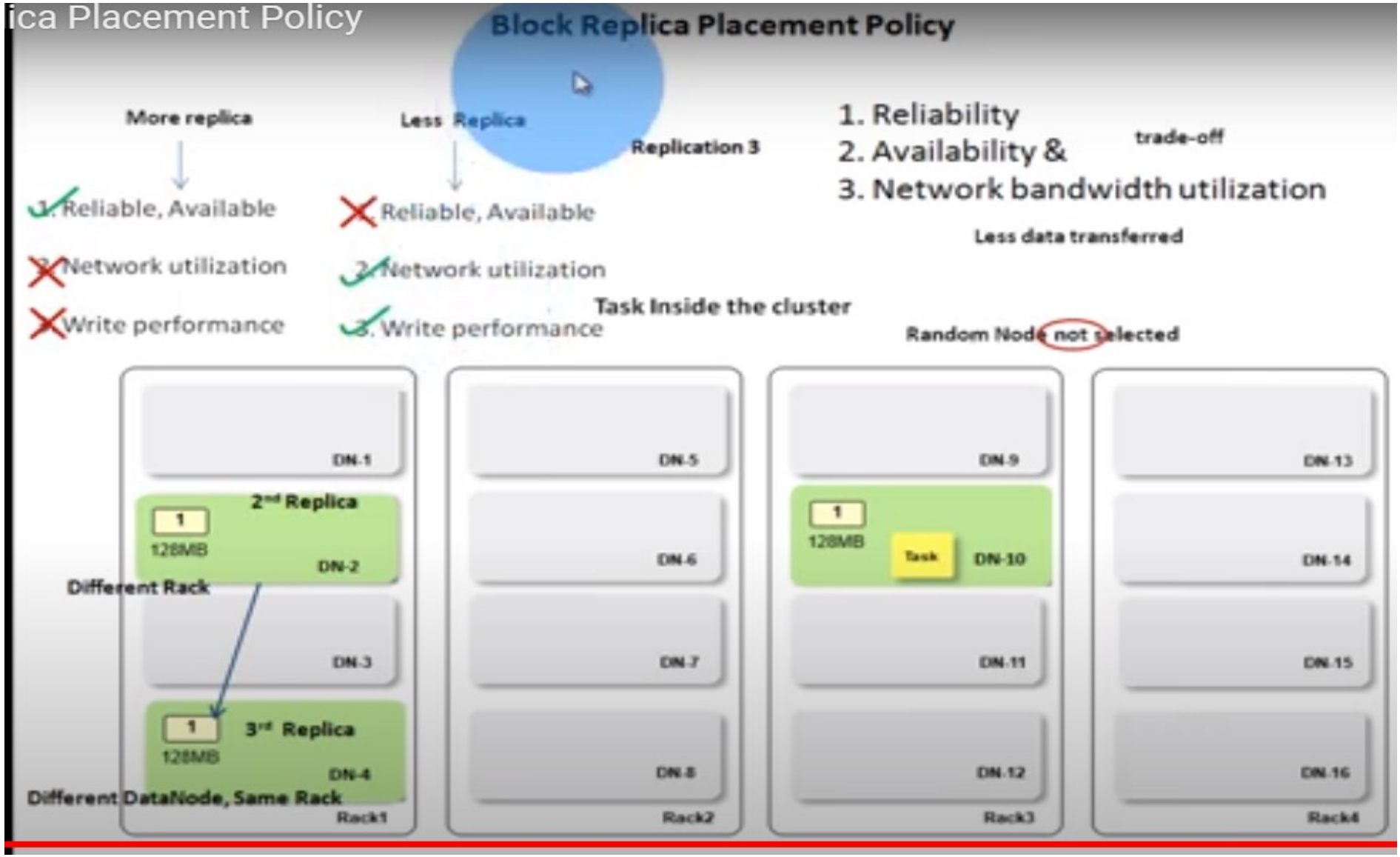
Replica Placement Strategy

As per the 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.



ica Placement Policy




ica Placement Policy
Block Replica Placement Policy
More replica
Less Replica
Replication 3
1. Reliability
2. Availability &
3. Network bandwidth utilization
trade-off
1. Reliable, Available
2. Network utilization
3. Write performance
Reliable, Available
Network utilization
Write performance
Less data transferred
Task Inside the cluster
Random Node not selected
DN-1
2nd Replica
1
128MB
DN-2
Different Rack
DN-3
1
3rd Replica
128MB
DN-4
Different DataNode, Same Rack
Rack1
DN-5
DN-6
DN-7
DN-8
Rack2
DN-9
1
128MB
Task
DN-10
DN-11
DN-12
Rack3
DN-13
DN-14
DN-15
DN-16
Rack4

Working with HDFS Commands

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

Act:

hadoop fs -mkdir /sample

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

Act:

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

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

Act:

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