Birla Vishvakarma Mahavidyalaya
Engineering College, Vallabh Vidyanagar

4CP02: BIG DATA

Unit 1
Introduction to Big Data

What is Data?

Information

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals

Classification of Digital Data

- The structure of the data decides how to work with it and also what insights it can produce.
- All data goes through a process called ETL before it can be analyzed.
- Data is harvested, formatted to be readable by an application, and then stored for use.
- The ETL process varies for each structure of data.

1. **Structured data**
2. **Unstructured Data**
3. **Semi-structured**

Structured data

- easiest to work with
- highly organized
- quantitative
- follows schemas
- tabular form
- stored in spread sheets or RDBMS

Unstructured data

- lacks any specific form or structure
- does not have a consistent format
- format varying all time
- very difficult and time-consuming to process and analyze
- Audio, Video, Images

Semi-structured data

- inherits a few properties of Structured Data
- schema is not appropriately defined
- does not obey the formal structure of data models such as RDBMS
- works with OLTP (Online Transaction Processing)
- JSON, XML, CSV, TSV

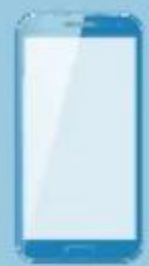

```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

1. **Evolution of Technology**
2. IOT
3. Social Media
4. Other Factors

Telephone

Desktop

Car



Mobile

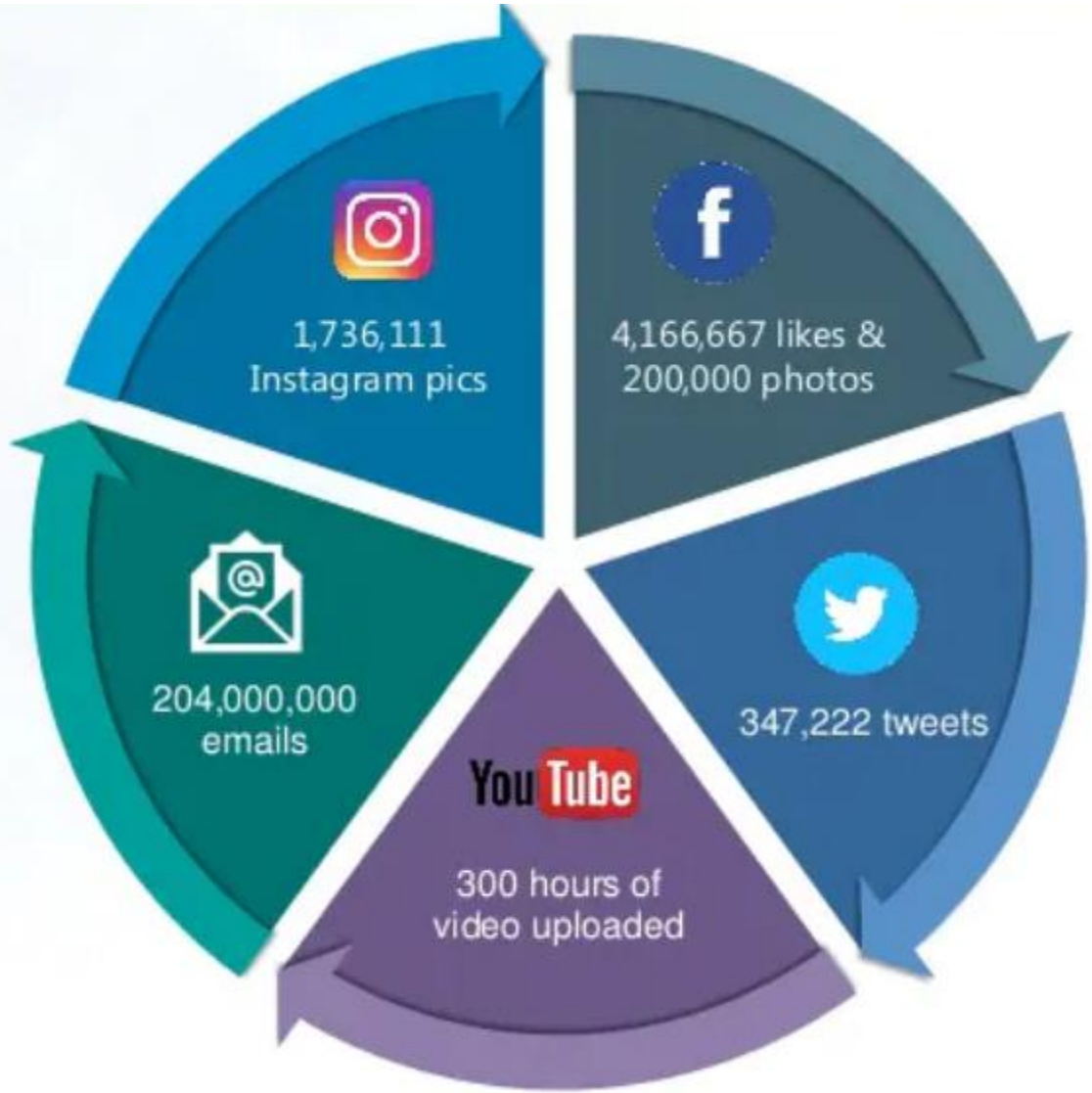
Cloud

Smart Car

1. Evolution of Technology
2. **IOT**
3. Social Media
4. Other Factors

IOT: 50 Billion devices by 2020

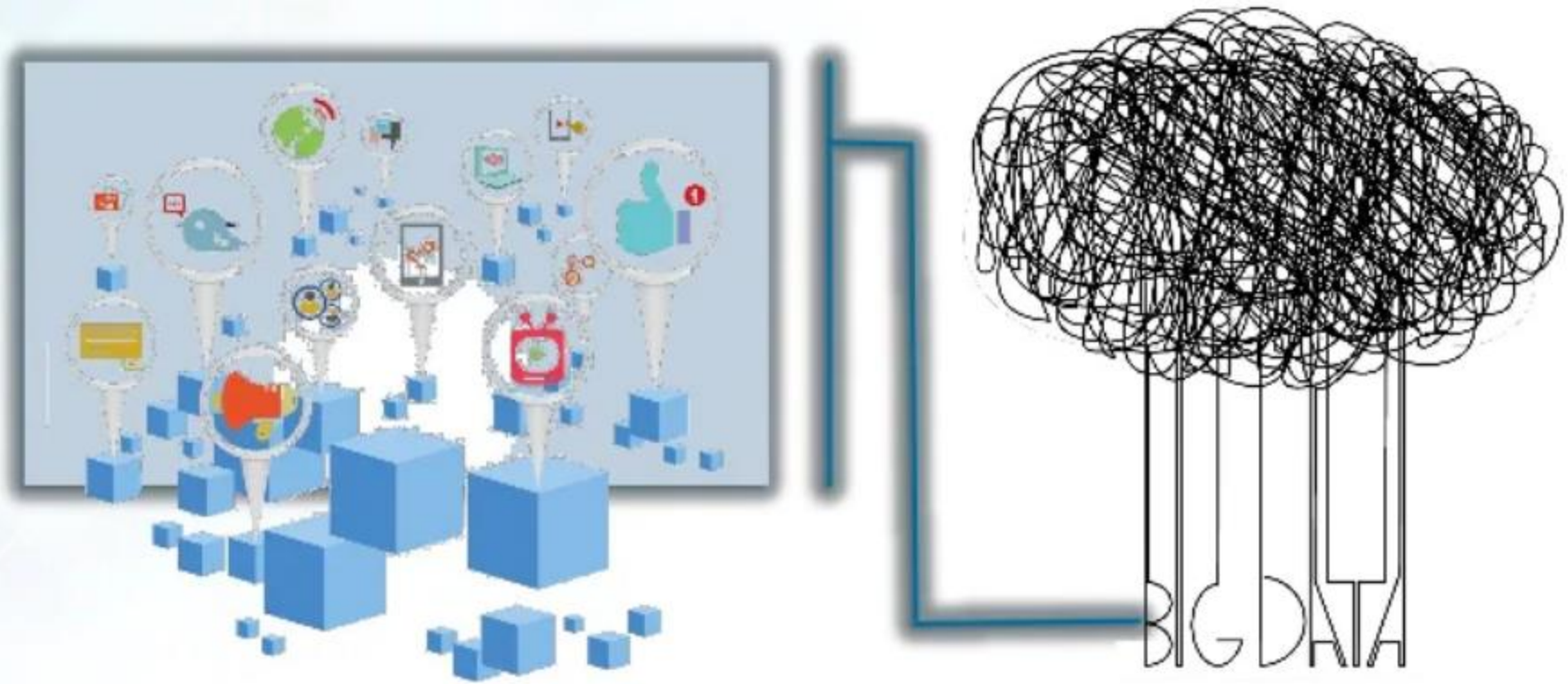
1. Evolution of Technology
2. IOT
3. **Social Media**
4. Other Factors



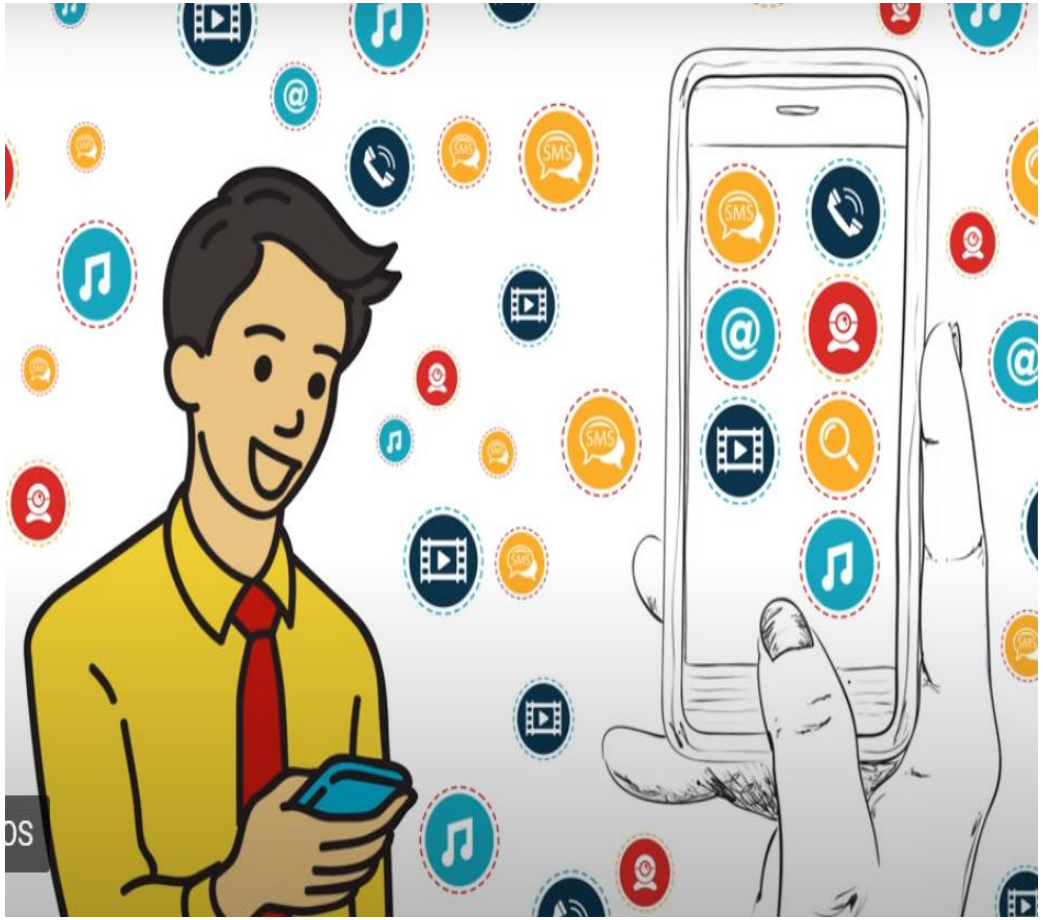
Social Media

1. Evolution of Technology
2. IOT
3. Social Media
4. Other Factors

Big data is the term for collection of data sets so large and complex that it becomes difficult to process using on-hand database system tools or traditional data processing applications



Why Big Data?



- 40 Exabytes data generated by single user per month
- 40 Exabytes X 5000000000 ?????
- Not possible to compute and accessing through traditional computer.
- This massive amount of data term as “BIG DATA”.

UNITS OF DATA

Unit	Bytes
Kilobyte (KB)	10^3 (1,000)
Megabyte (MB)	10^6 (1,000,000)
Gigabyte (GB)	10^9 (1,000,000,000)
Terabyte (TB)	10^{12} (1,000,000,000,000)
Petabyte (PB)	10^{15} (1,000,000,000,000,000)
Exabyte (EB)	10^{18} (1,000,000,000,000,000,000)
Zettabyte (ZB)	10^{21} (1,000,000,000,000,000,000,000)
Yottabyte (YB)	10^{24} (1,000,000,000,000,000,000,000,000)

How much is a zettabyte?
1,000,000,000,000,000,00
0,000 bytes
A stack of 1TB hard disks
that is 25,400 km high

Examples Of Big Data

- The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day.

Data generated per minute

“2.1Million“

“3.8Million“

“1.0Million“

“4.5Million“

- That’s a lot of data

- A single **Jet engine** can generate ***10+terabytes*** of data in ***30 minutes*** of flight time. With many thousand flights per day, generation of data reaches up to many ***Petabytes.***

Introduction to Big Data

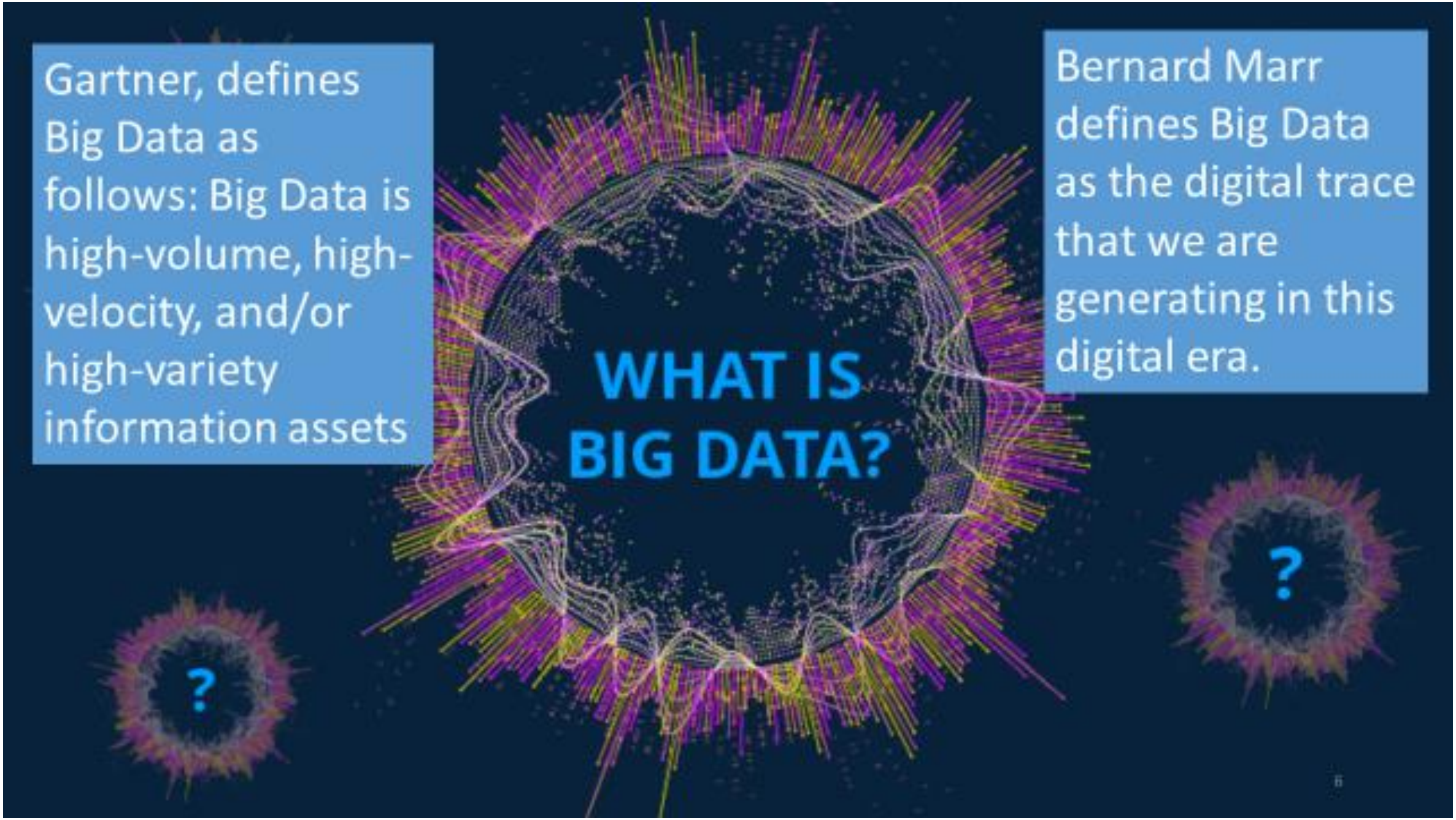
No single standard definition…

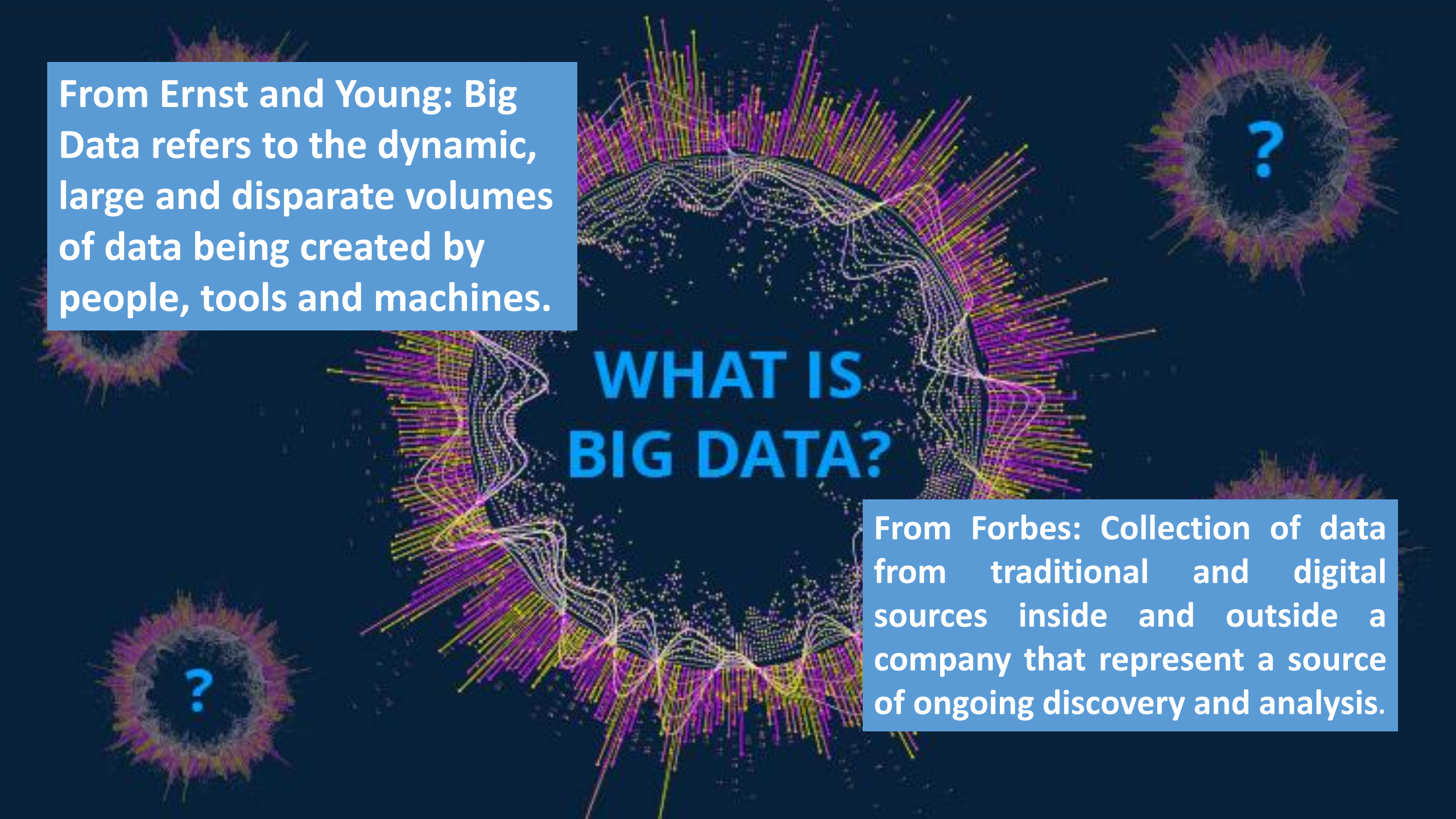
Big data is a term that describes the large volume of data – both structured and unstructured.

❖ "***Big Data***" is data

➢ Whose scale, diversity, and complexity require new architecture, techniques, algorithms, & analytics to manage it and extract value & hidden knowledge from it…

➢ Big data refers to data sets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze


Gartner, defines Big Data as follows: Big Data is high-volume, high-velocity, and/or high-variety information assets
WHAT IS BIG DATA?
Bernard Marr defines Big Data as the digital trace that we are generating in this digital era.
?
?


From Ernst and Young: Big Data refers to the dynamic, large and disparate volumes of data being created by people, tools and machines.
WHAT IS BIG DATA?
From Forbes: Collection of data from traditional and digital sources inside and outside a company that represent a source of ongoing discovery and analysis.

Big Data Is Making Fast Food Faster

- McDonald’s or Burger King
- Some fast food chains are now monitoring their drive-through lanes and changing their menu features (you know, the ones on the LCD screen as opposed to the numbers on the board) in response.
- if the line is really backed up, the features will change to reflect items that can be quickly prepared and served so as to move through the queue faster.
- If the line is relatively short, then the features will display higher margin menu items that take a bit more time to prepare.

We Missed You!

- Imagine this: you're relaxing at home, trying to decide which restaurant to eat at with your spouse.
- The decision is taking a bit longer than it should;
- Suddenly, an email arrives in your inbox. you see an email from Fig & Olive, you were a regular at but haven't been able to visit in more than a month. The subject line says "We Miss You!"
- when you open it, you're greeted with a message that communicates two points:
  - Fig & Olive is wondering why you haven't been in for a while.
  - They want to give you a free order of crostini because they just miss you so much!
- "Honey", you exclaim, "I know where we're going!"

Behind the scenes

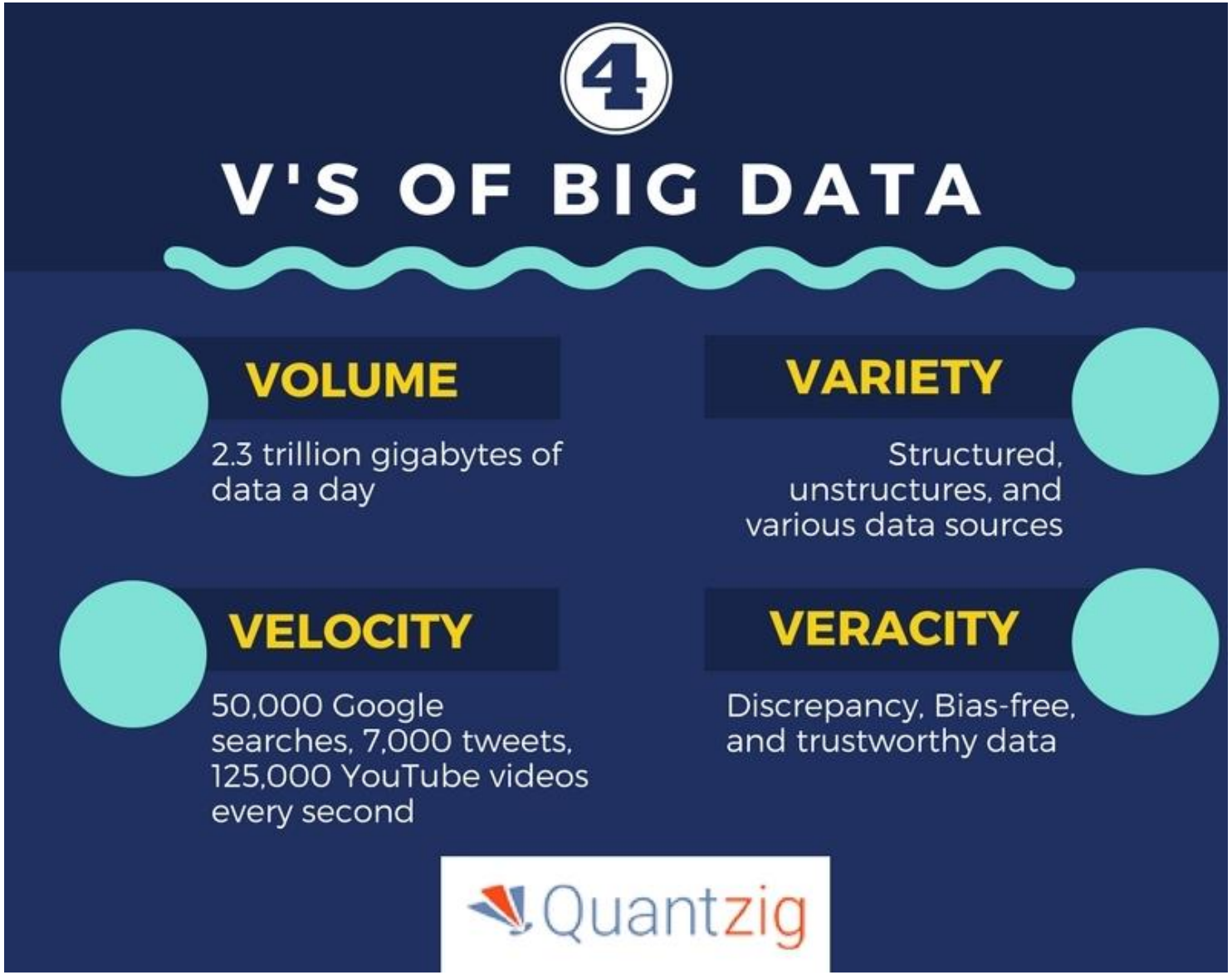
- The 7-unit NY-based Fig & Olive has been using guest management software to track their guests ordering habits and to deliver targeted email campaigns.
- For example, the “We Miss You!” campaign generated almost 300 visits and $36,000 in sales

A simple example of a big data process

- Analysis:
  - lollipops bought by people older than 25
  - lollipops preferred by people younger than 10
- Interpretation:
  - Moms believe: lollipops = bad teeth
  - Boys and girls believe that lollipops are for babies
- Decision:
  - We make lollipops without sugar
  - We ask dentists to advertise our lollipops
  - We make commercials targeted to boys and girls

Characteristics Of Big Data

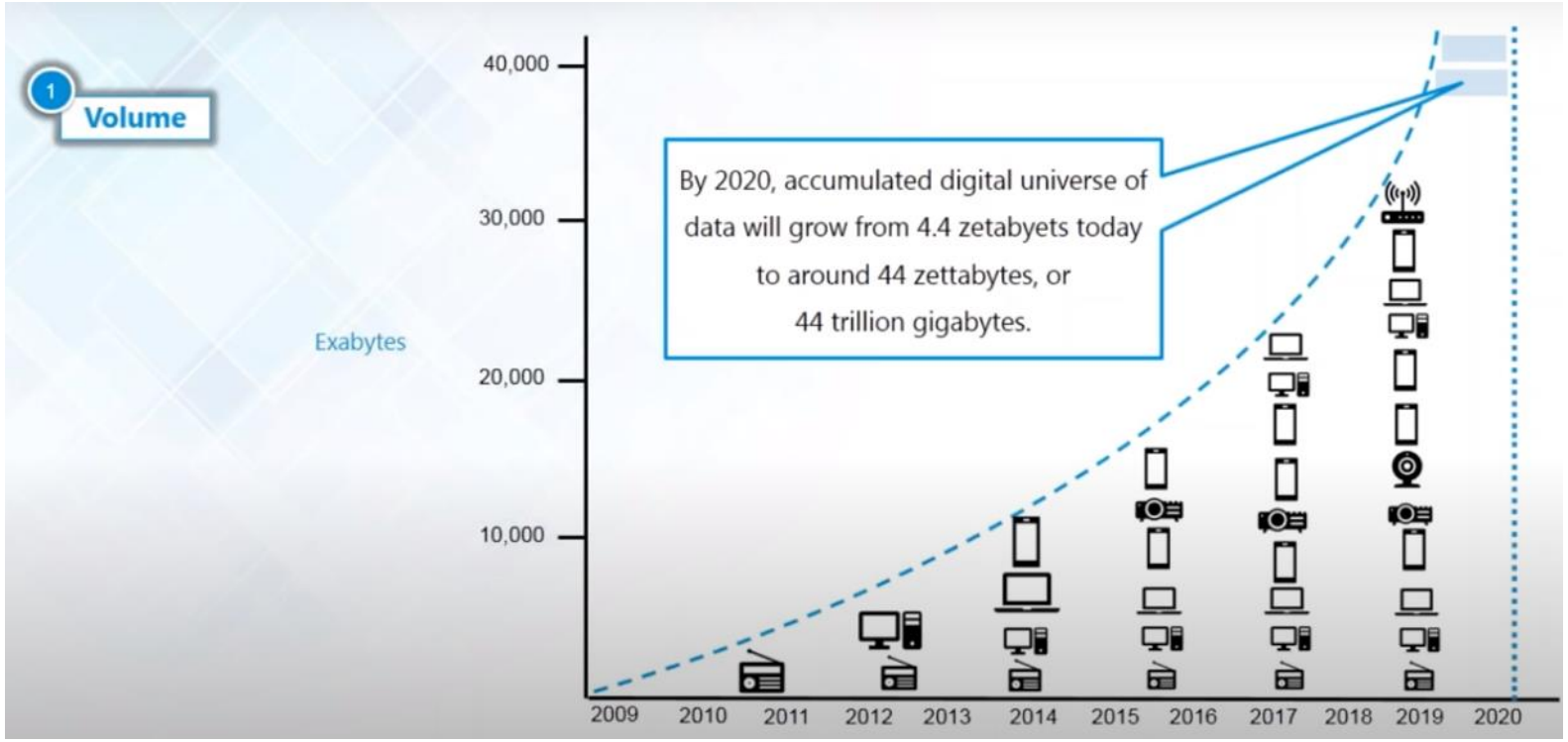
- Volume
- Variety
- Velocity
- Veracity



Volume

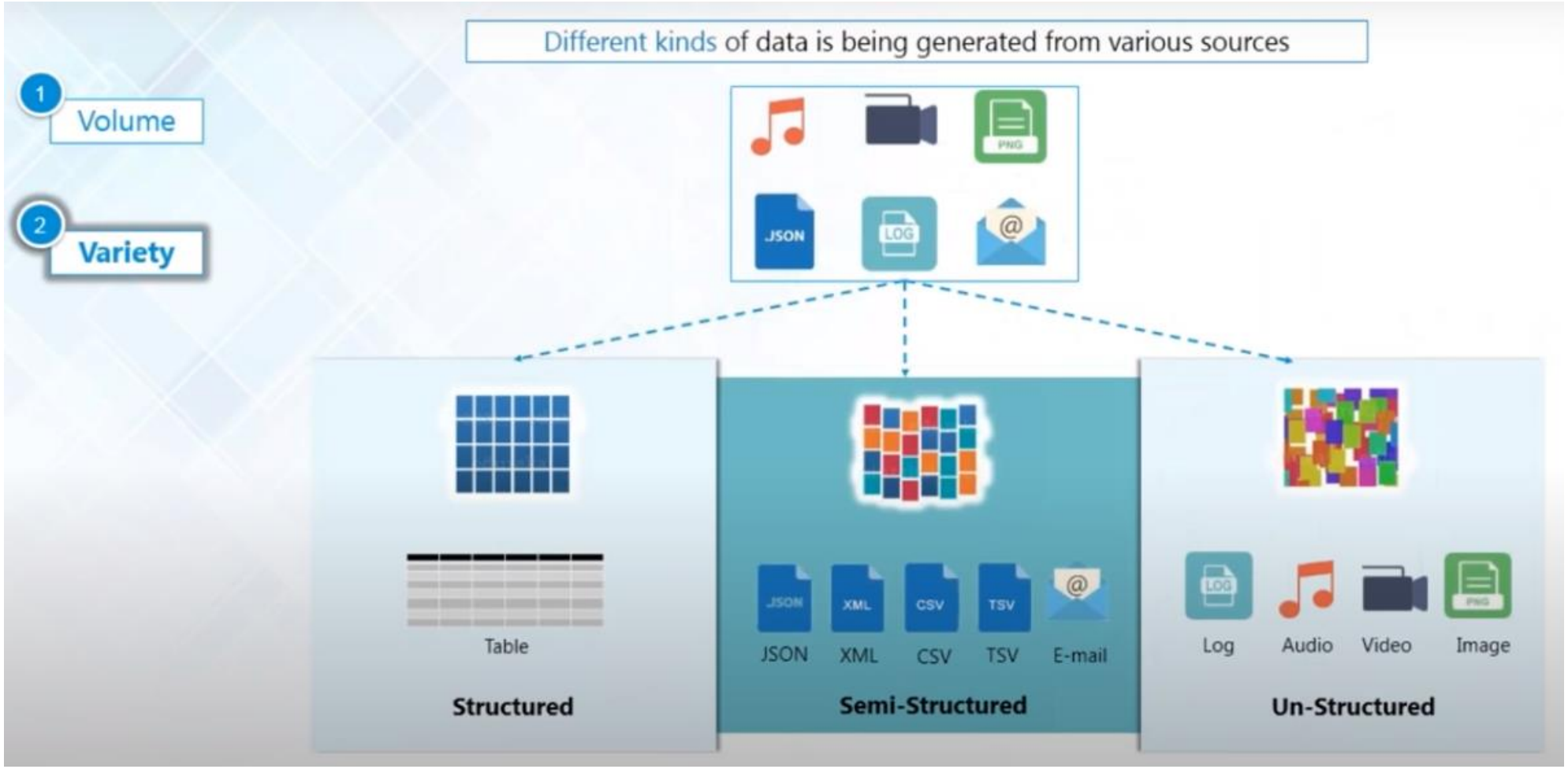
- Big data is enormous. While traditional data is measured in familiar sizes like megabytes, gigabytes and terabytes, big data is stored in petabytes and zettabytes.
- Volume refers to the amount of data generated through websites, portals and online applications.

Facebook has 2 billion users, Youtube 1 billion users,
Twitter 350 million users and Instagram 700 million users.
Every day, these users contribute to billions of images,
posts, videos, tweets etc.



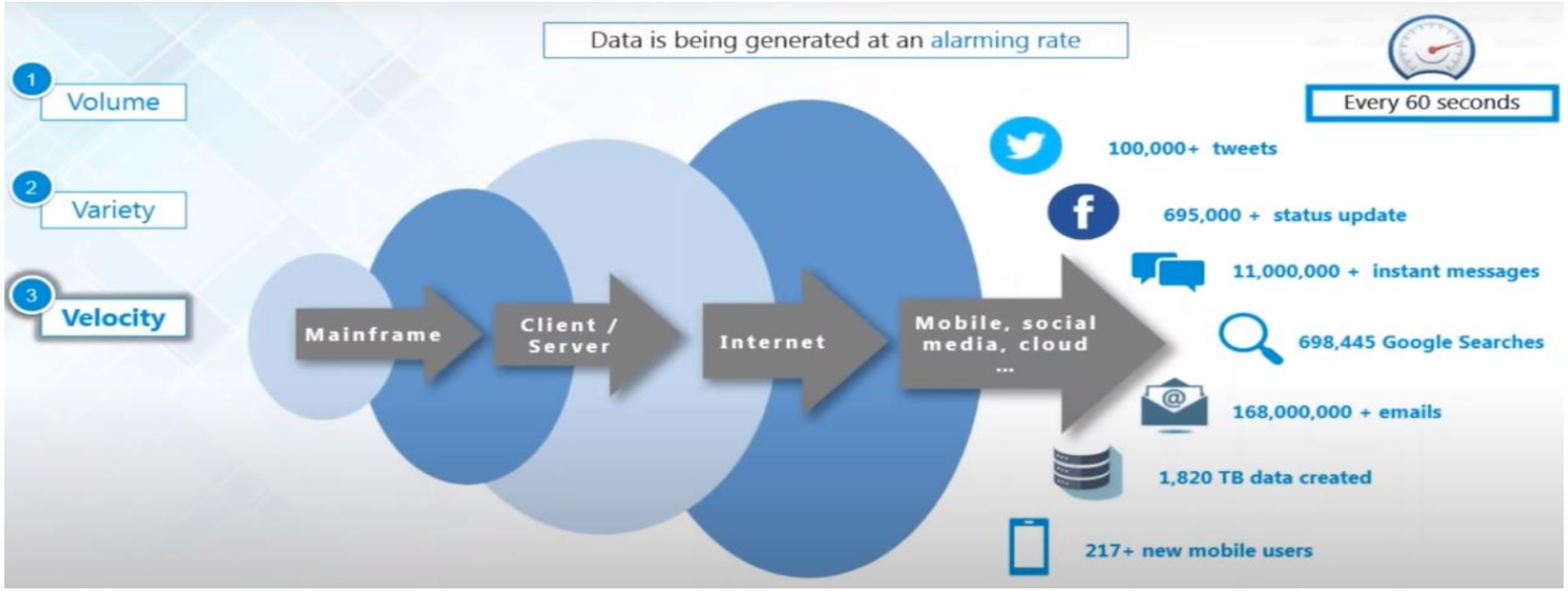
VARIETY

Variety in Big Data refers to all the structured and unstructured data that has the possibility of getting generated either by humans or by machines.



VELOCITY

- With Velocity we refer to the speed with which data are being generated.
- Staying with our social media example, every day 900 million photos are uploaded on Facebook, 500 million tweets are posted on Twitter, 0.4 million hours of video are uploaded on Youtube and 3.5 billion searches are performed in Google.
- This is like a nuclear data explosion.
- Big Data helps the company to hold this explosion, accept the incoming flow of data and at the same time process it fast so that it does not create bottlenecks.



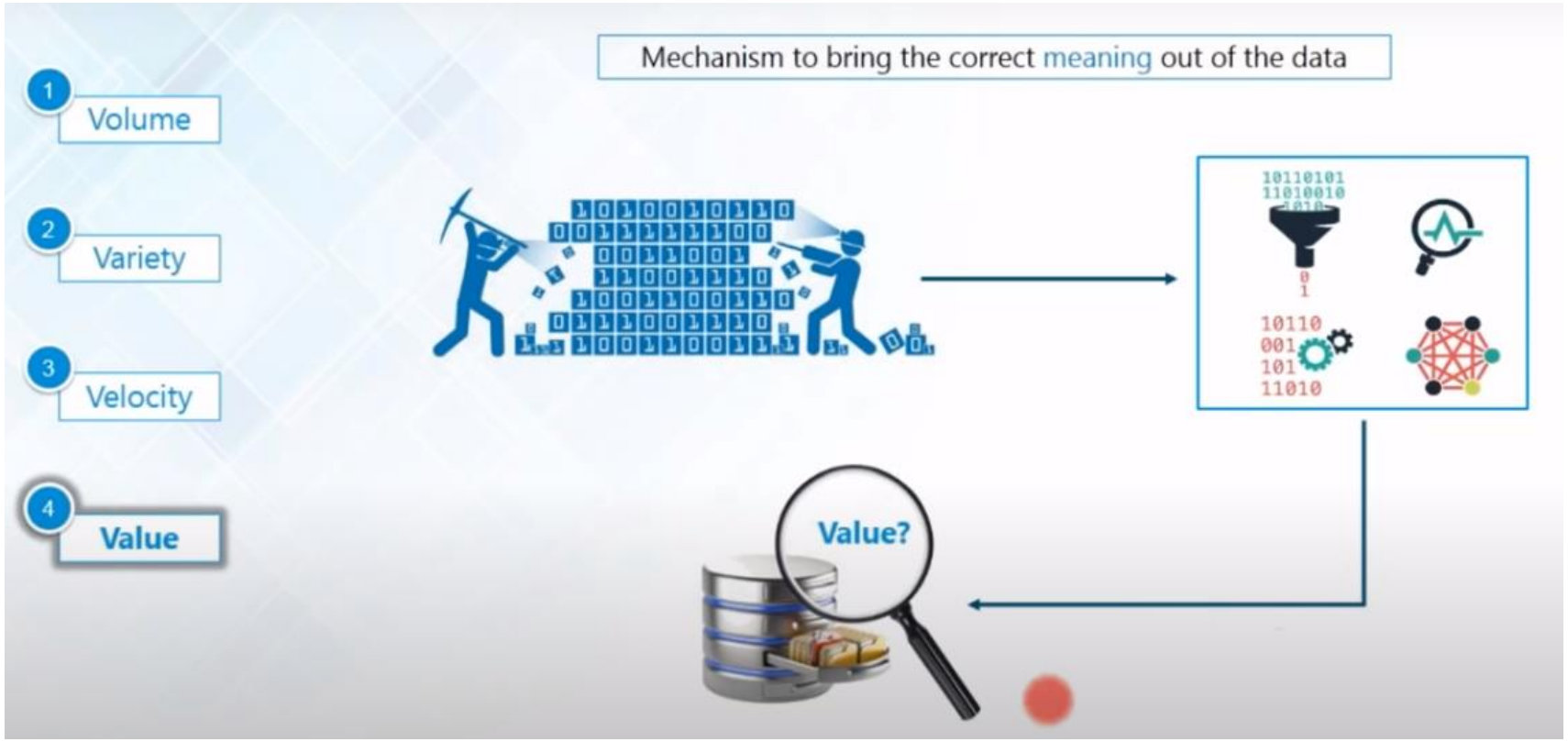
Veracity

- inconsistencies and uncertainty in data, that is data which is available can sometimes get messy and quality and accuracy are difficult to control.
- how accurate or truthful a data set may be
- The veracity of big data denotes the trustworthiness of the data.

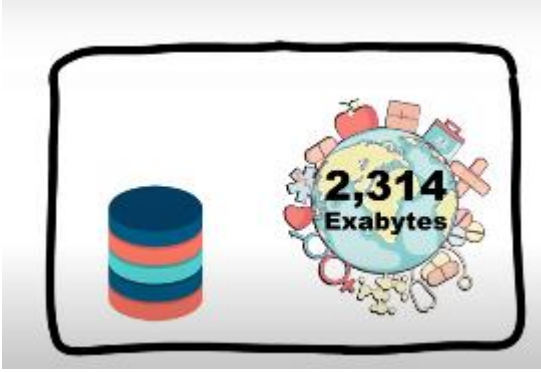
- when it comes to the accuracy of big data, it’s not just the quality of the data itself but how trustworthy the data source, type, and processing of it is.

Value

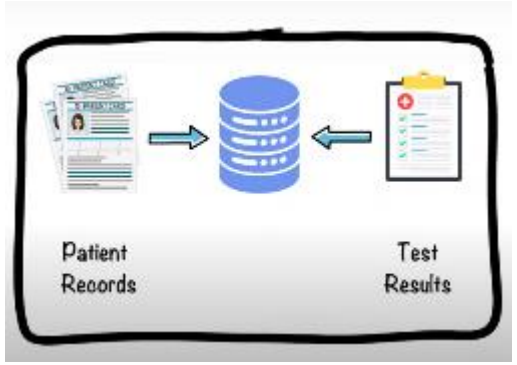
The bulk of Data having no Value is of no good to the company, unless you turn it into something useful.



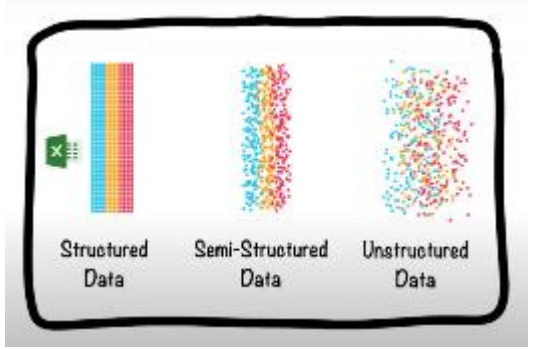
Volume



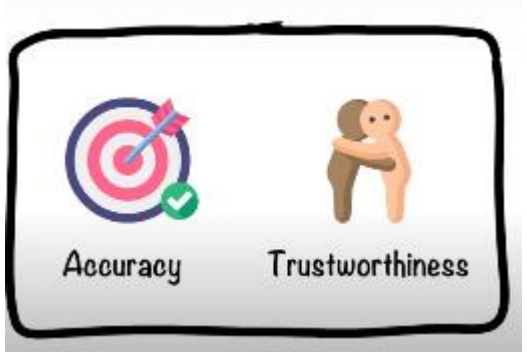
Velocity



Variety



Veracity



Value

The Sources of Big Data

- **Black Box Data :** This is the data generated by airplanes, including jets and helicopters. Black box data includes flight crew voices, microphone recordings, and aircraft performance information.
- **Social Media Data:** This is data developed by such social media sites as Twitter, Facebook, Instagram, Pinterest, and Google+.
- **Stock Exchange Data**: This is data from stock exchanges about the share selling and buying decisions made by customers.
- **Power Grid Data**: This is data from power grids. It holds information on particular nodes, such as usage information.
- **Transport Data** : This includes possible capacity, vehicle model, availability, and distance covered by a vehicle.
- **Search Engine Data**: This is one of the most significant sources of big data. Search engines have vast databases where they get their data.

Data Analysis Vs Data Analytics

Data Analysis	Data Analytics
the separation of a whole into its component parts	the method of logical analysis.
to think in terms of past	to think in terms of future.
Analysis looks backwards over time, providing marketers with a historical view of what has happened	analytics look forward to model the future or predict a result.

Data Analysis Vs Data 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.

- **Data analytics**, on the other hand, is a broader term referring to a *discipline* that encompasses the complete management of data – including collecting, cleaning, organizing, storing, governing, and analyzing data – as well as the tools and techniques used to do so.
- **Analytics is the process of obtaining an optimal and realistic decision based on existing data.**

Uses and Examples of Big Data Analytics

- Using analytics to understand customer behavior in order to optimize the customer experience
- Predicting future trends in order to make better business decisions
- Improving marketing campaigns by understanding what works and what doesn't
- Increasing operational efficiency by understanding where bottlenecks are and how to fix them
- Detecting fraud and other forms of misuse sooner

Benefits and Advantages of Big Data Analytics

1. Risk Management

- Banco de Oro, a Phillippine banking company, uses Big Data analytics to identify fraudulent activities and discrepancies. The organization leverages it to narrow down a list of suspects or root causes of problems.

2. Product Development and Innovations

Rolls-Royce, one of the largest manufacturers of jet engines for airlines and armed forces across the globe, uses Big Data analytics to analyze how efficient the engine designs are and if there is any need for improvements.

3. Quicker and Better Decision Making Within Organizations

- Starbucks uses Big Data analytics to make strategic decisions. For example, the company leverages it to decide if a particular location would be suitable for a new outlet or not. They will analyze several different factors, such as population, demographics, accessibility of the location, and more.

4. Improve Customer Experience

- Delta Air Lines uses Big Data analysis to improve customer experiences. They monitor tweets to find out their customers' experience regarding their journeys, delays, and so on. The airline identifies negative tweets and does what's necessary to remedy the situation. By publicly addressing these issues and offering solutions, it helps the airline build good customer relations.

Qualitative analysis

- When we do qualitative analysis, we are exploring how we describe something.
- When we do qualitative work, we work with descriptions.
- We work with feelings, thoughts, perceptions. We attempt to understand motivations and behaviors.
- gathering information :
  - Asking customers for information on why they chose your product, which provides a new perspective on your company’s competitive advantage.
  - Gathering opinions from customers on the value they see in the product or service offered.
  - Using written customer feedback on websites to guide future marketing campaigns,

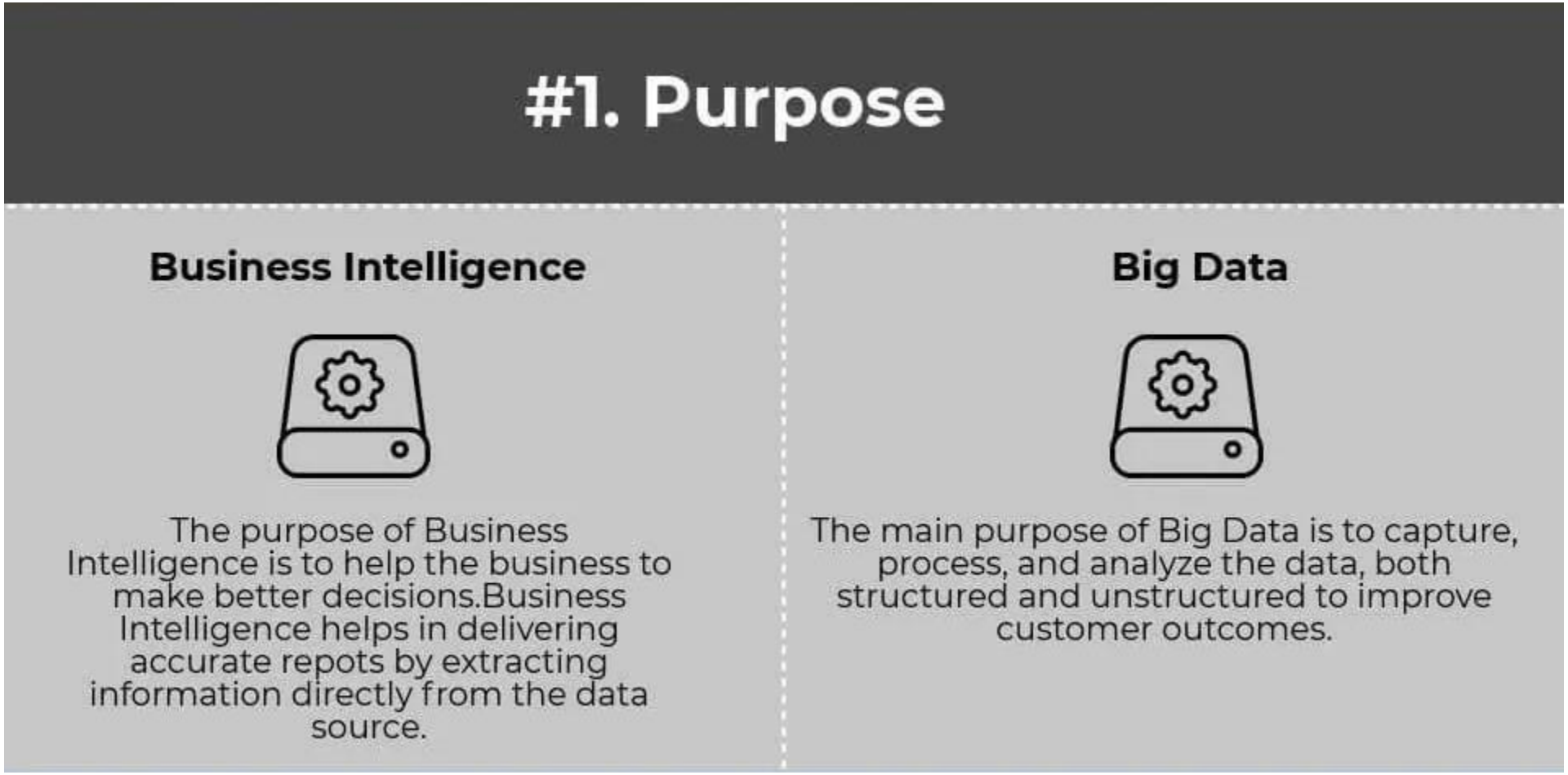
Quantitative analysis

- When we do quantitative analysis, we are exploring facts, measures, numbers and percentages.
- When we do quantitative work, we work with numbers, statistics, formulae and data.

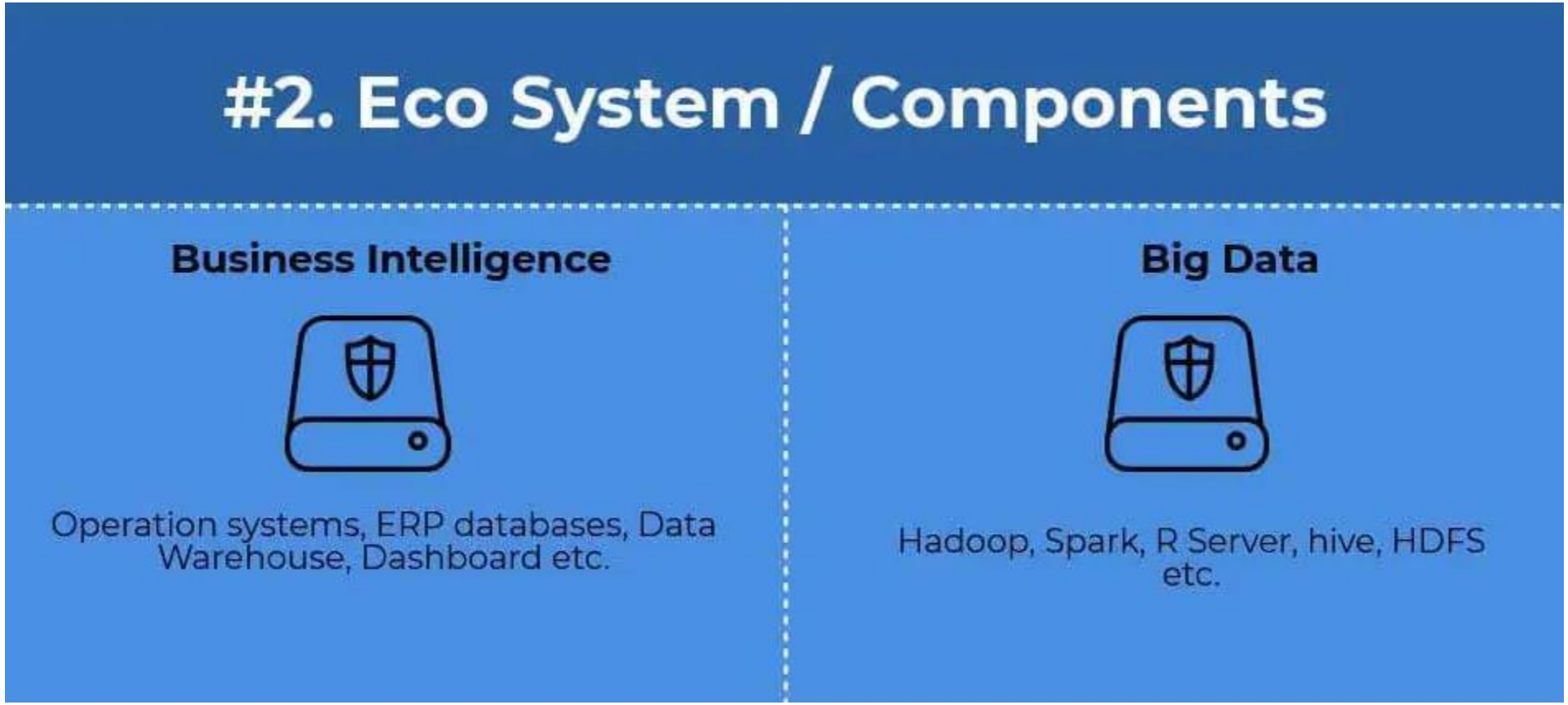
Big Data Analysis Techniques:

- Association rule learning
- Classification tree analysis
- Genetic algorithms
- Machine learning
- Regression analysis
- Sentiment analysis
- Social network analysis

Traditional Business intelligence vs Big Data



Traditional Business intelligence vs Big Data



#3. Tools

Business Intelligence

Below is the list of tools used for business intelligence. These tools enable business to collate, analyze and visualize data, which can be used in making better business decisions and to come up with good strategic plans.

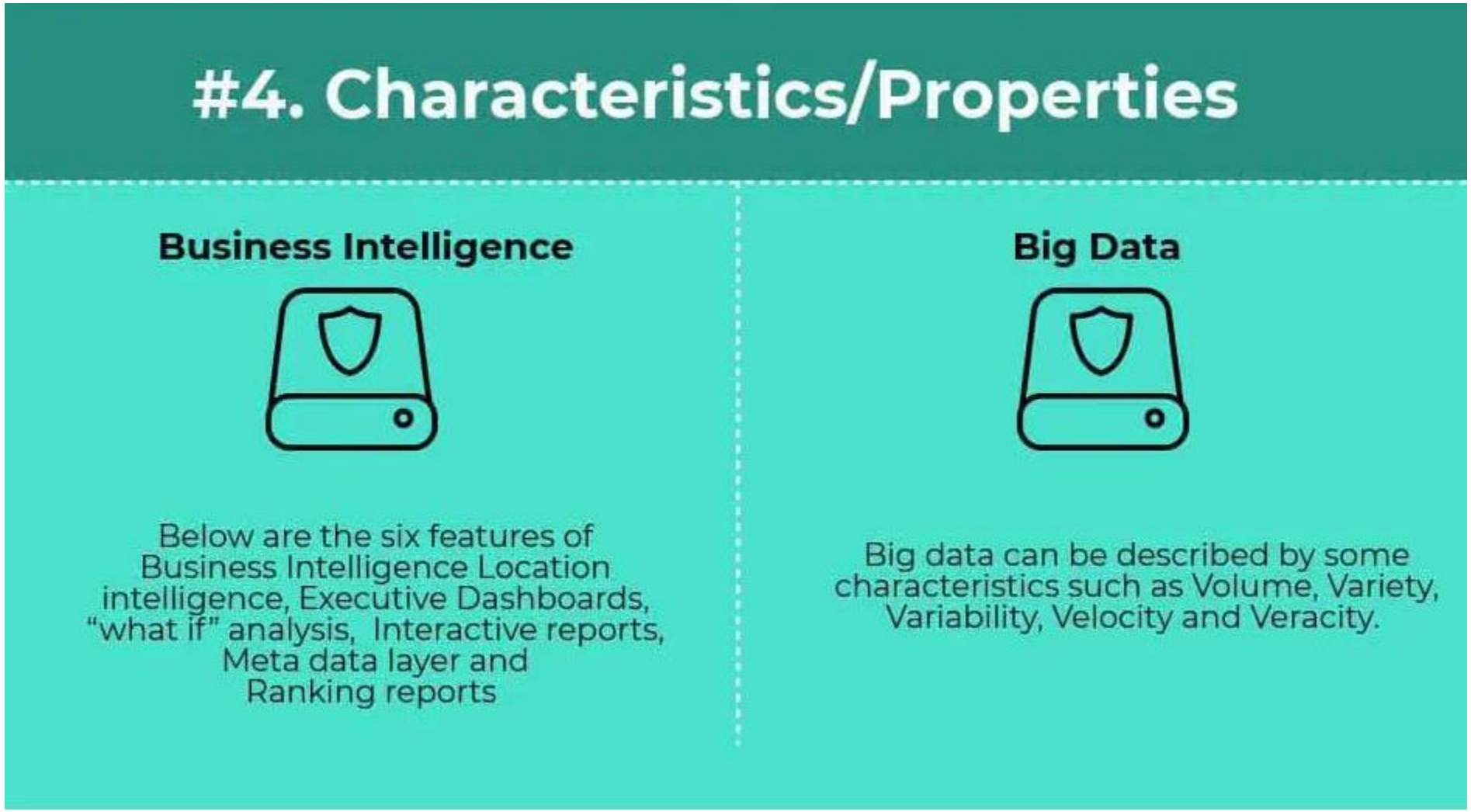
- Tableau
- Qlik Sense
- Online analytical processing (OLAP)
- Sisense
- Data Warehousing
- Digital Dashboards and Data mining
- Microsoft Power BI
- Google Analytics etc

Big Data

Below is the list of tools used in Big Data. These tools or frameworks store large amount of data and process them to get the insights from data to make good decisions for business.

- Hadoop
- Spark
- Hive
- Polybase
- Presto
- Cassandra
- Plotly
- Cloudera
- Storm etc

Traditional Business intelligence vs Big Data



#5. Benefits

Business Intelligence

Below is the list of benefits of Business Intelligence.

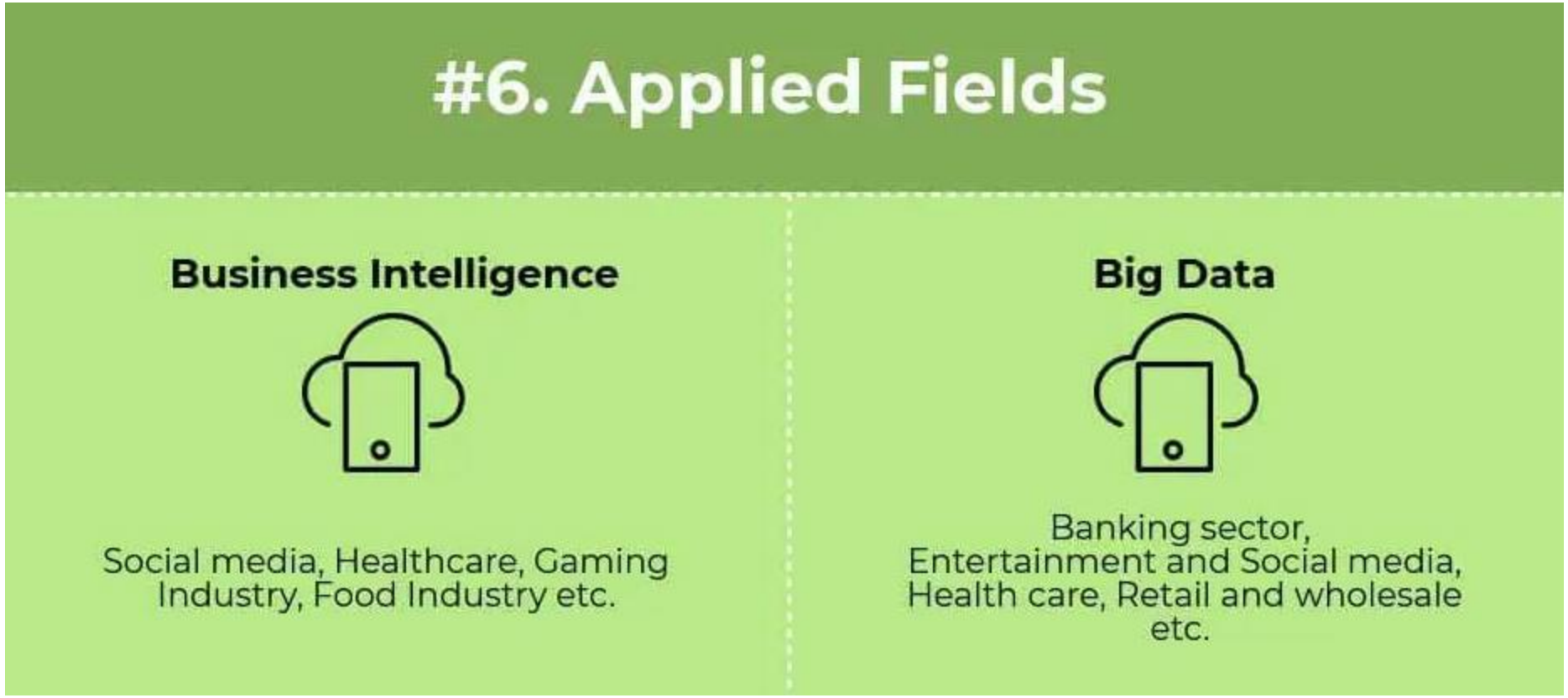
- Helps in making better business decisions
- Faster and more accurate reporting and analysis
- Improved data quality
- Reduced costs
- Increase revenues
- Improved operational efficiency etc.

Big Data

Below is the list of benefits of Big Data.

- Better Decision making
- Fraud detection
- Storage, mining and analysis of data
- Market prediction &and forecasting
- Improves the service
- Helps in implementing the new strategies
- Keep up with customer trends
- Cost saving
- Better sales insights, which helps in increasing revenues etc.

Traditional Business intelligence vs Big Data



Big Data Analytics Tool

Hadoop - helps in storing and analyzing data

MongoDB - used on datasets that change frequently

Talend - used for data integration and management

Cassandra - a distributed database used to handle chunks of data

Spark - used for real-time processing and analyzing large amounts of data

STORM - an open-source real-time computational system

Kafka - a distributed streaming platform that is used for fault-tolerant storage

Birla Vishvakarma Mahavidyalaya
Engineering College, Vallabh Vidyanagar

4CP02: DATA ANALYTICS AND VISUALIZATION

Unit 2
The Big Data Technologies

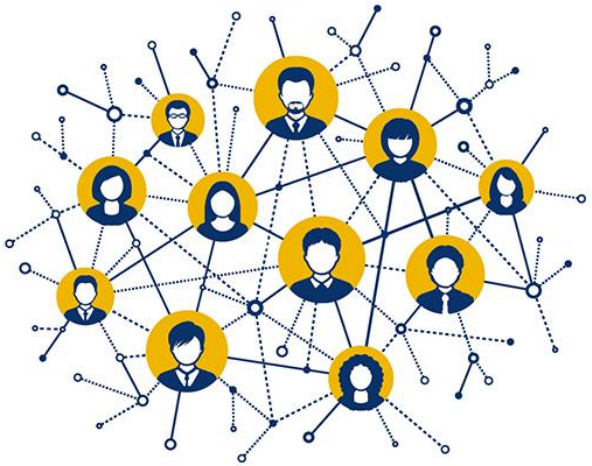
NoSQL

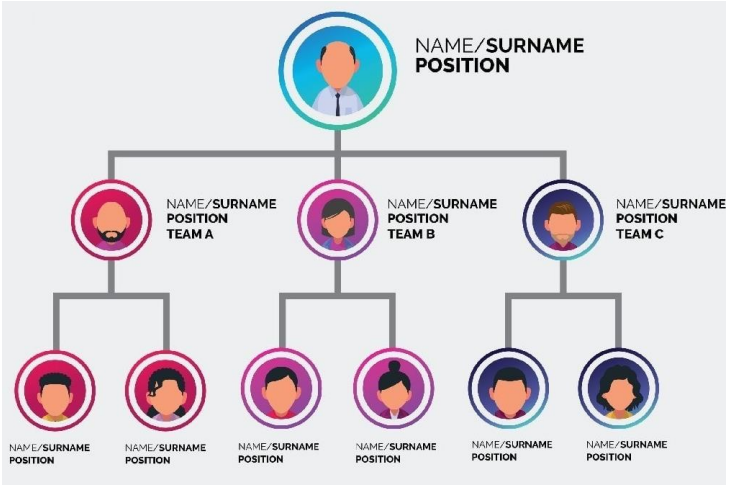
- NoSQL stands for “not only SQL” rather than “no SQL” at all.

Ancient DBMS

Early Database Management Systems

- Flat file
- Hierarchical
- Network





Flat File Database Management Systems

- In a flat file, records follow a uniform format, and there are no structures for indexing or recognizing relationships between records.
- The file is simple.
- A flat file can be a plain text file, or a binary file.

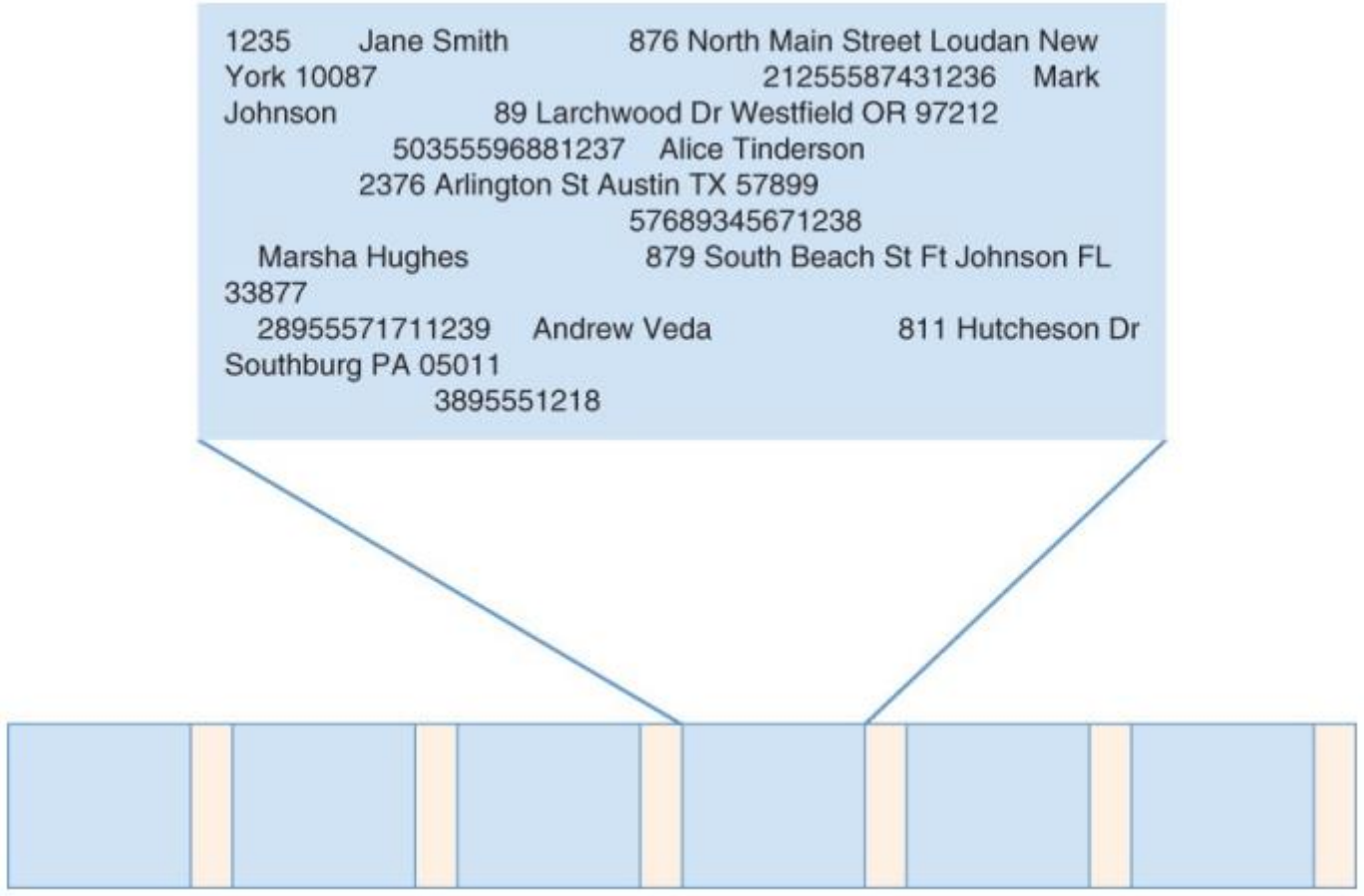
Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Flat File Database Management Systems

- At the time flat files were commonly used data management, but magnetic tape was also in widespread use.
- For this reason, early data management files had to accommodate the physical constraints of physical systems.

Flat File Database Management Systems



A block is a chunk of data read by tape or disk drive in a single read operation.

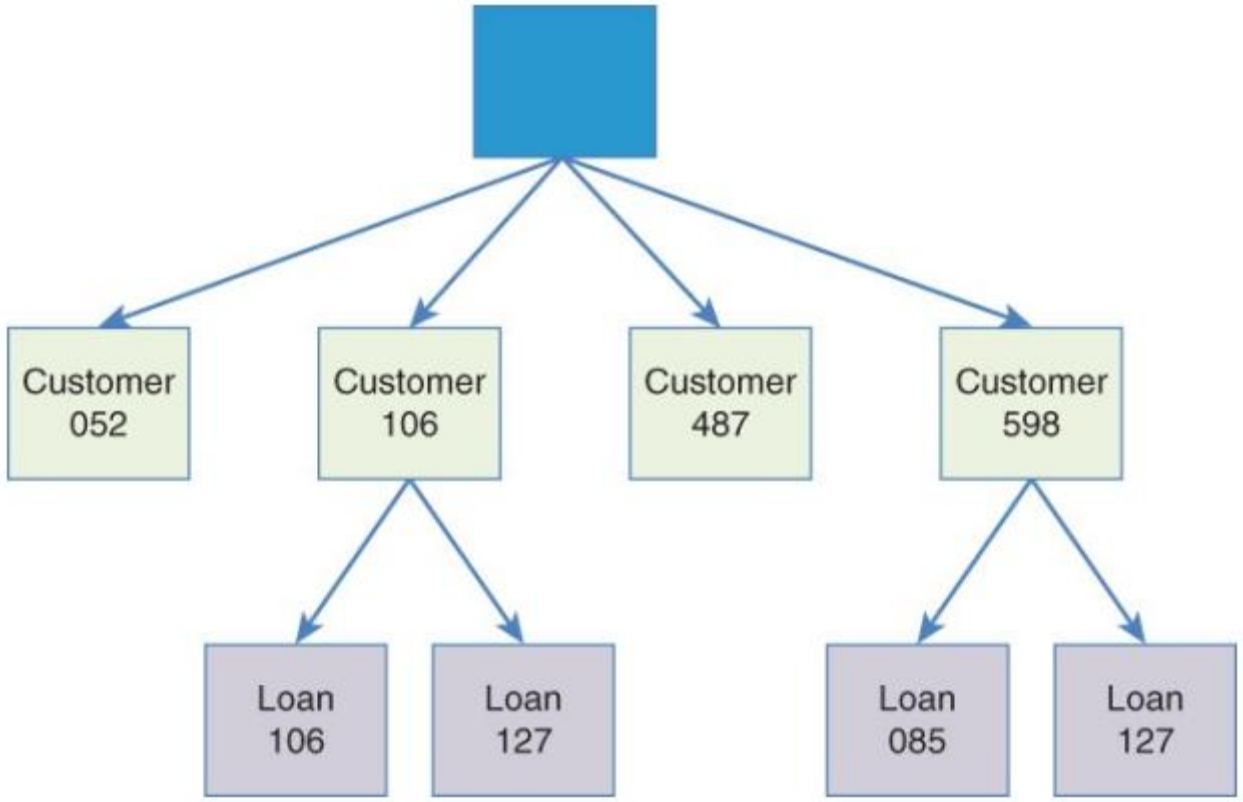
Flat File Database Management Systems

Random access to blocks on tape can take more time than sequential access

Hierarchical DBMS

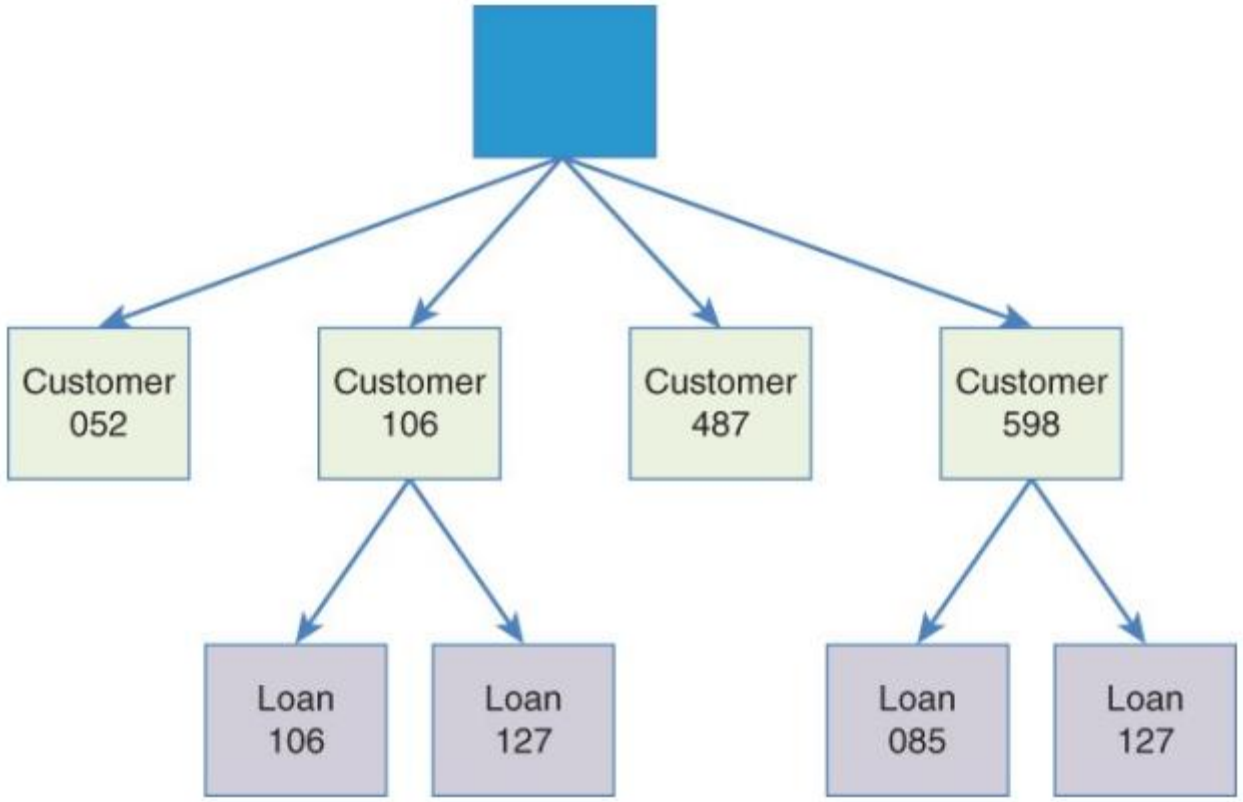
- One of the limitations of flat file–based data management systems is that they can be inefficient to search.
- Hierarchical data models address this problem by organizing data in a hierarchy of parent-child relationships.
- A hierarchy starts with a root node that links to the top layer of data nodes or records.
- These top-layer records can have child records that contain additional data about the parent record.

Hierarchical DBMS



A hierarchical data model for a loan management database.

Hierarchical DBMS

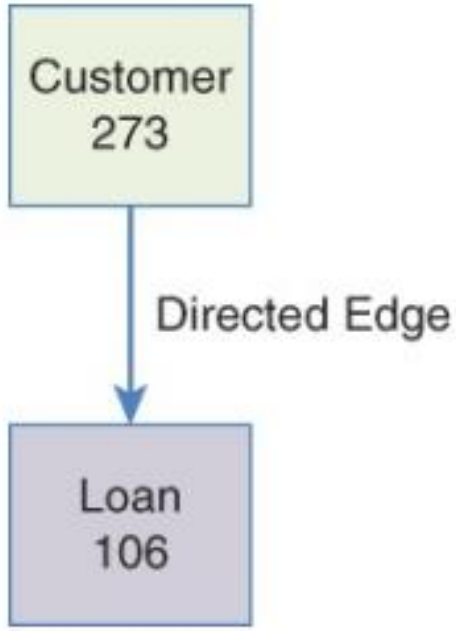


A hierarchical data model for a loan management database.

Network DBMS

- A network data model is like a hierarchical data model in that it uses links between records.
- However, unlike hierarchical data models, you are not restricted to having one parent record.
- Also, unlike flat file data management systems and hierarchical data management systems, network data models have two essential components: a schema and the database itself.

Network DBMS

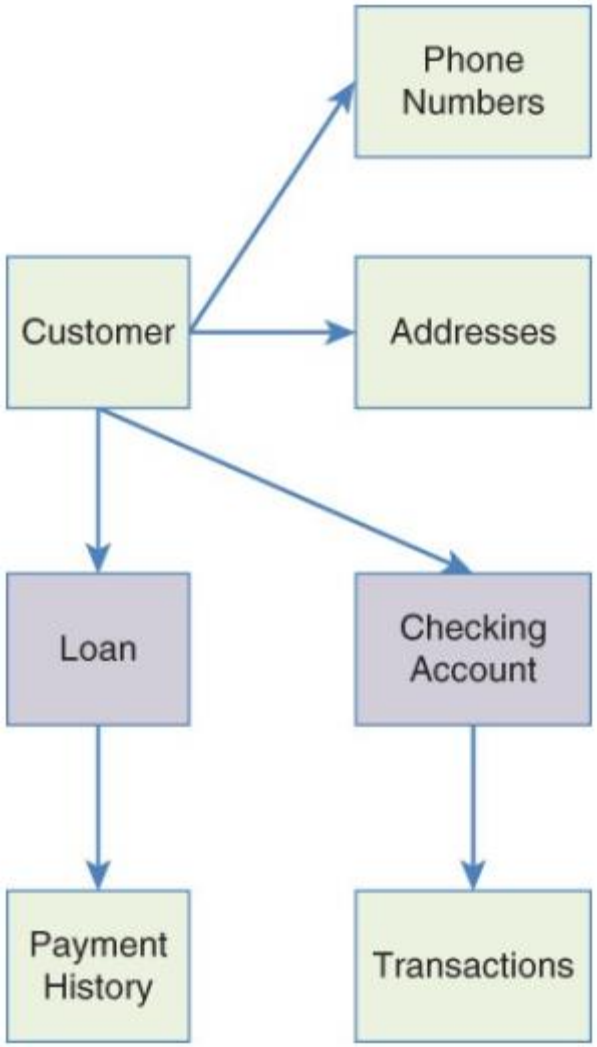


A parent-child relationship is represented by a directed edge.

Network DBMS

- A network is made up of data records linked together.
- The data records are known as nodes and the links are known as edges.
- The collection of nodes and edges is known as a graph.
- Network data models allow for multiple parents, such as two customers on a loan.
- But you cannot have cycles in the graph.

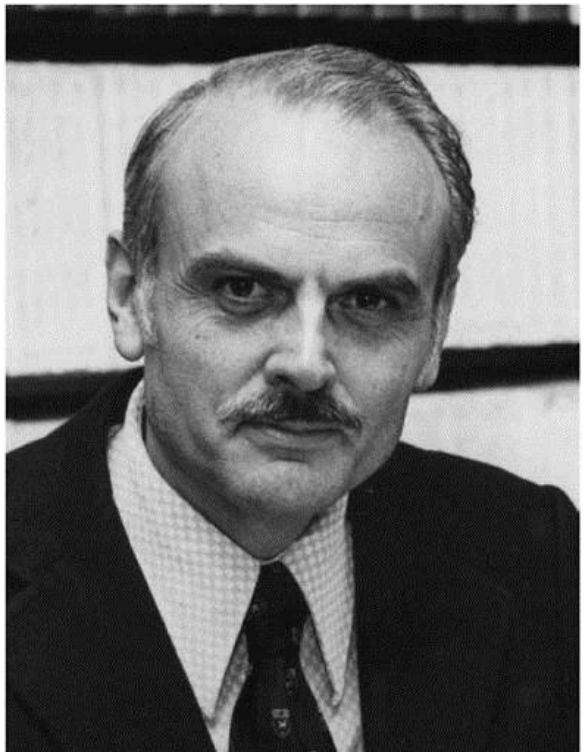
Network DBMS



A simple network schema shows which entities can link to other entities.

The Relational Database Revolution

- Network and hierarchical data management systems improved on flat file data management systems.
- But data management technology radically changed in 1970 when E. F. Codd published a paper on the design of a new type of database.



NoSQL

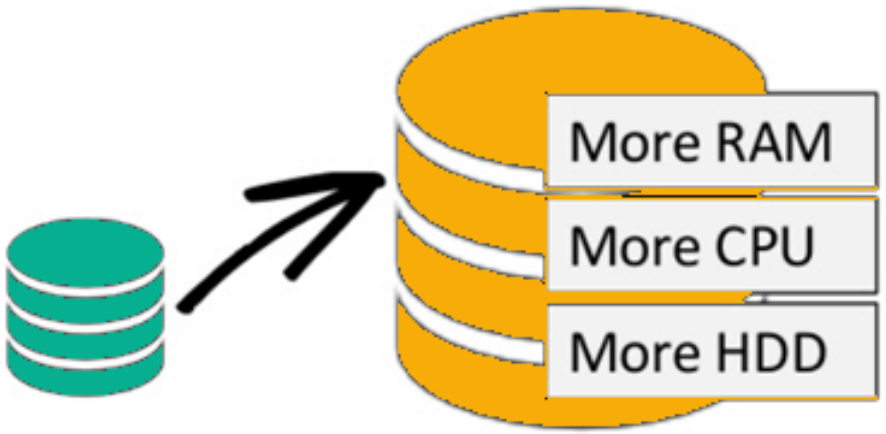
- NoSQL stands for “not only SQL” rather than “no SQL” at all.
- NoSQL databases solve a wide variety of data management problems by offering several types of solutions.
- NoSQL databases are commonly designed to use multiple servers, but this is not a strict requirement.
- A common way to satisfy need for scalability, flexibility, cost control, and availability is by designing data management systems to work across multiple servers, that is, as a distributed system.

Why NoSQL?

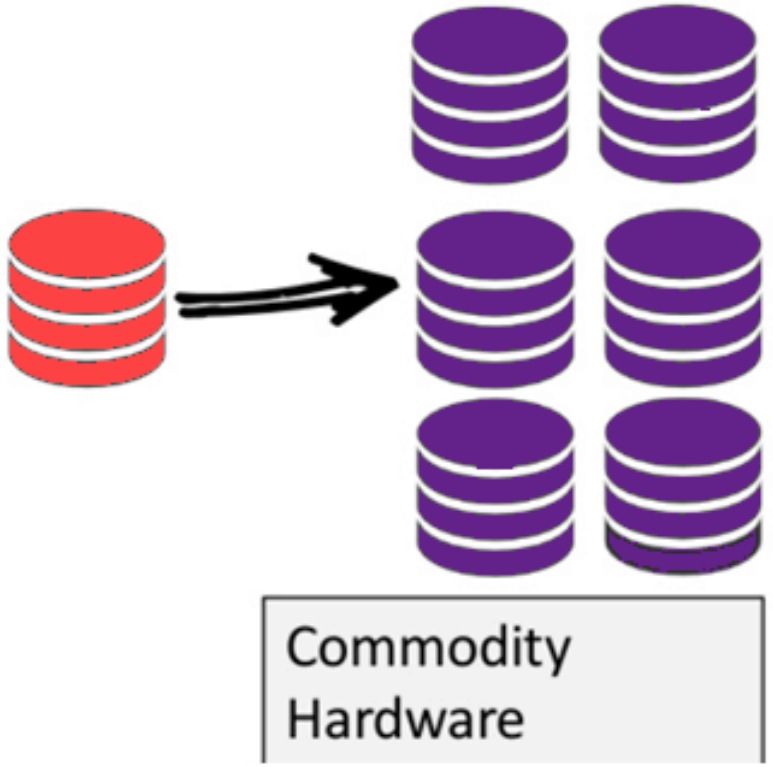
1) **Scaling Out:-**

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could “scale up” our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as “scaling out.”

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):



2) Dynamic Schema:-

- It allows insertion of data without a pre-defined schema.
- So, it facilitates application changes in real time, which supports

➢Faster development

➢Easy code integration

➢Require less database administration.

3) Auto Sharding :-

- It automatically spreads data across arbitrary number of servers.
- It balances a load of data and query on available servers.
- And if any server goes down, it is quickly replace without any major activity disruptions.

4) Replication:-

- It offers good support of replication which in turn guarantees

➢high availability

➢Fault tolerance

➢Disaster recovery

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

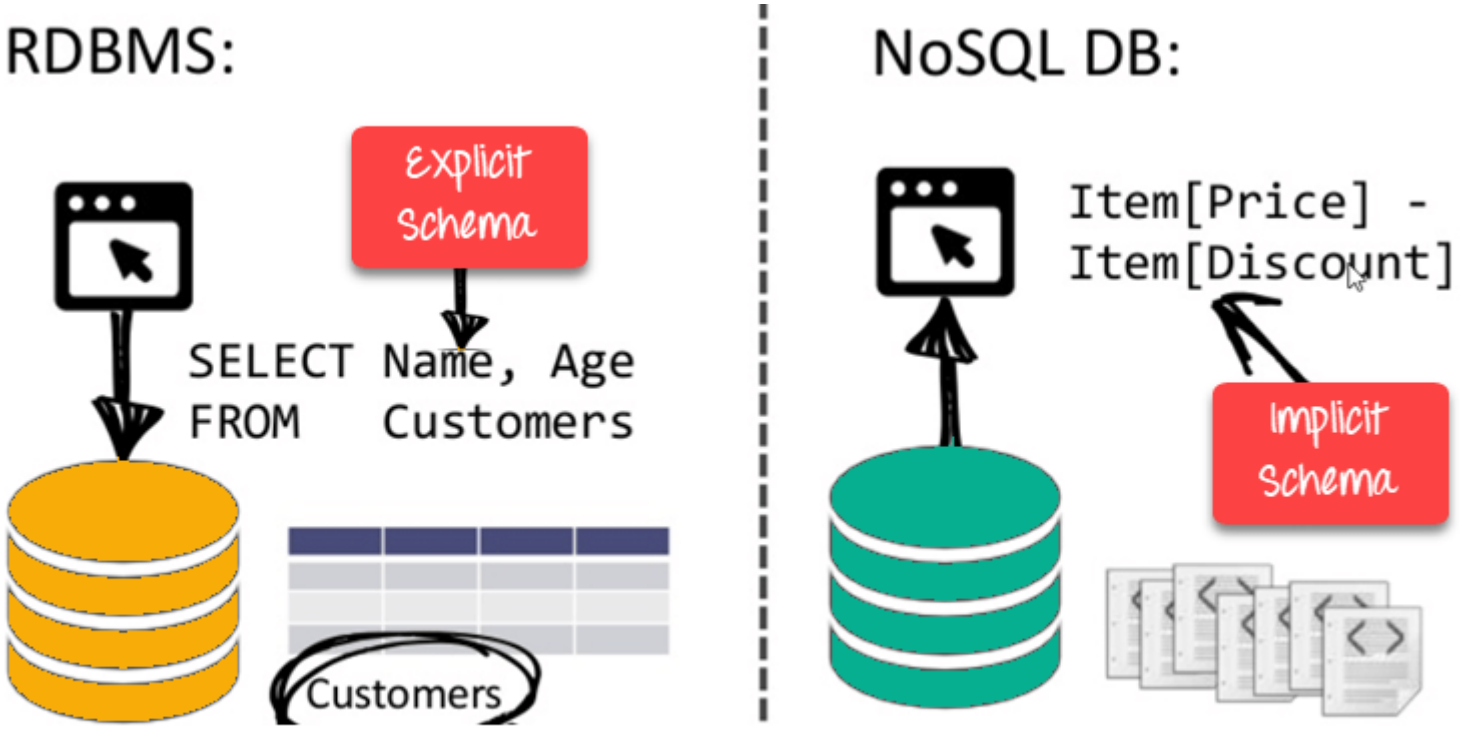
Features of NoSQL

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn’t require object-relational mapping and data normalization
- No complex features like query languages, query planners,referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



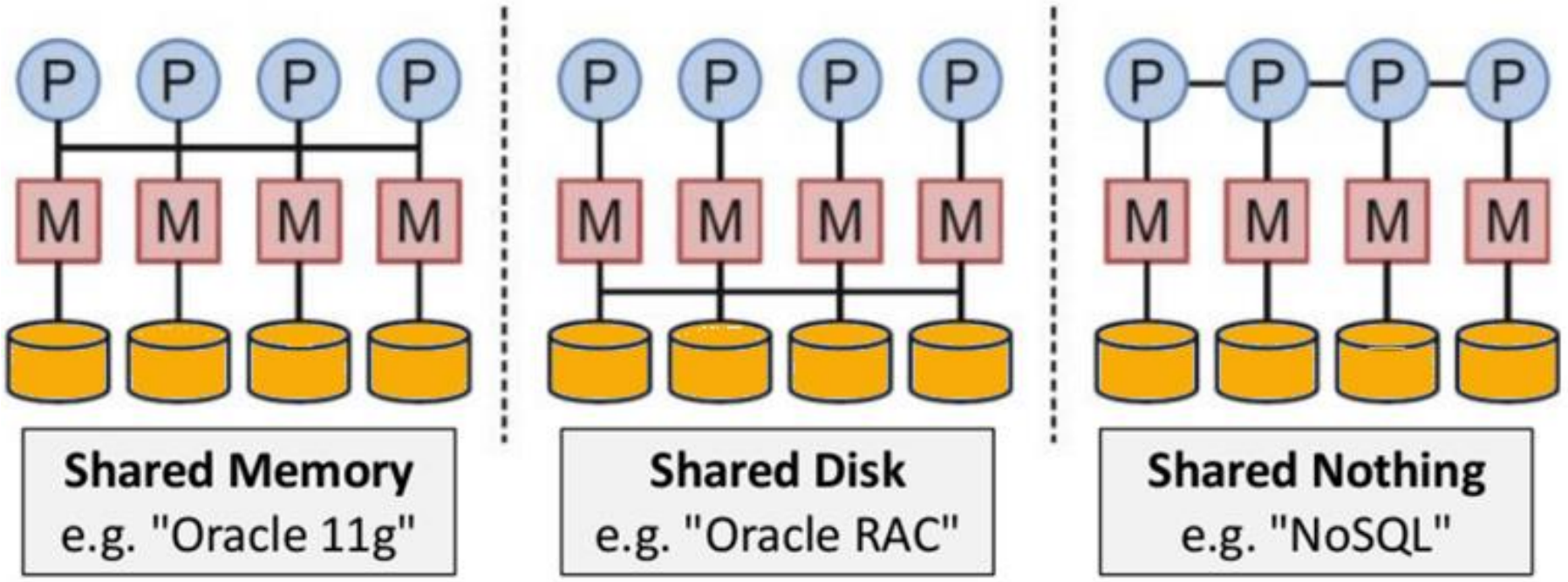
NoSQL is Schema-Free

Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based NoSQL query language
- Web-enabled databases running as internet-facing services

Distributed

- Data is distributed across several nodes in a cluster.
- Shared Nothing Architecture.
- This enables less coordination and higher distribution.



NoSQL

- In addition to the other benefits of NoSQL databases, distributed systems offer some level of operational simplicity.
- You can add and remove servers as needed rather than adding or removing memory, CPUs, and so on from a single server.
- Also, some NoSQL databases include features that automatically detect when a server is added or removed from a cluster.
- Many NoSQL databases take advantage of distributed systems, but they may employ different data management strategies.

Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented

1. Key-value databases: works with a simple model based on keys, which are identifiers for looking up data, and values, the data that is associated with keys.

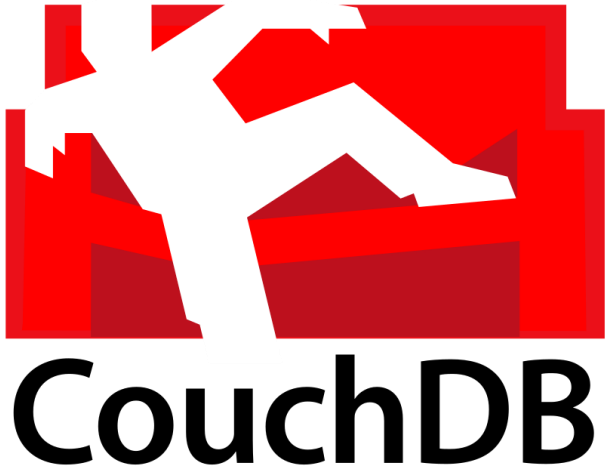
- Data is stored in key/value pairs.
- It is designed in such a way to handle lots of data and heavy load.
- Store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

- used as a collection, dictionaries, associative arrays, etc.
- Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

2. Document databases: It also use identifiers to look up values, but the values are typically more complex than those typically stored in key-value databases.

- Documents are collections of data items stored together in a flexible structure.
- The document is stored in JSON or XML formats



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Relational Vs. Document

3. Column family databases:-

have some of the characteristics of relational databases, such as organizing data into collections of columns.

Every column is treated separately. Values of single column databases are stored contiguously.

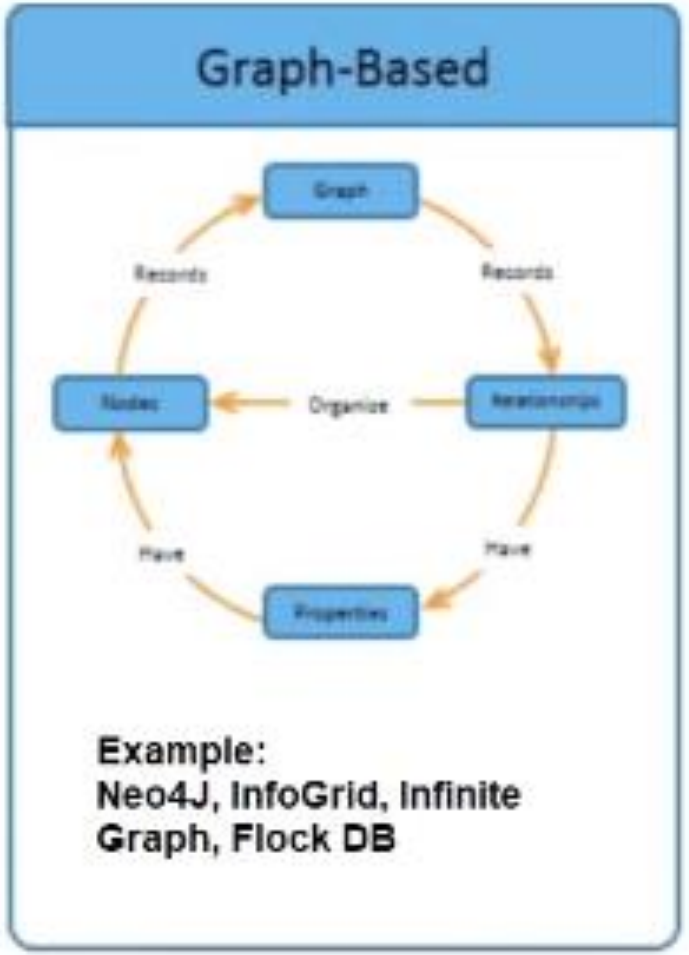
ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column based NoSQL database

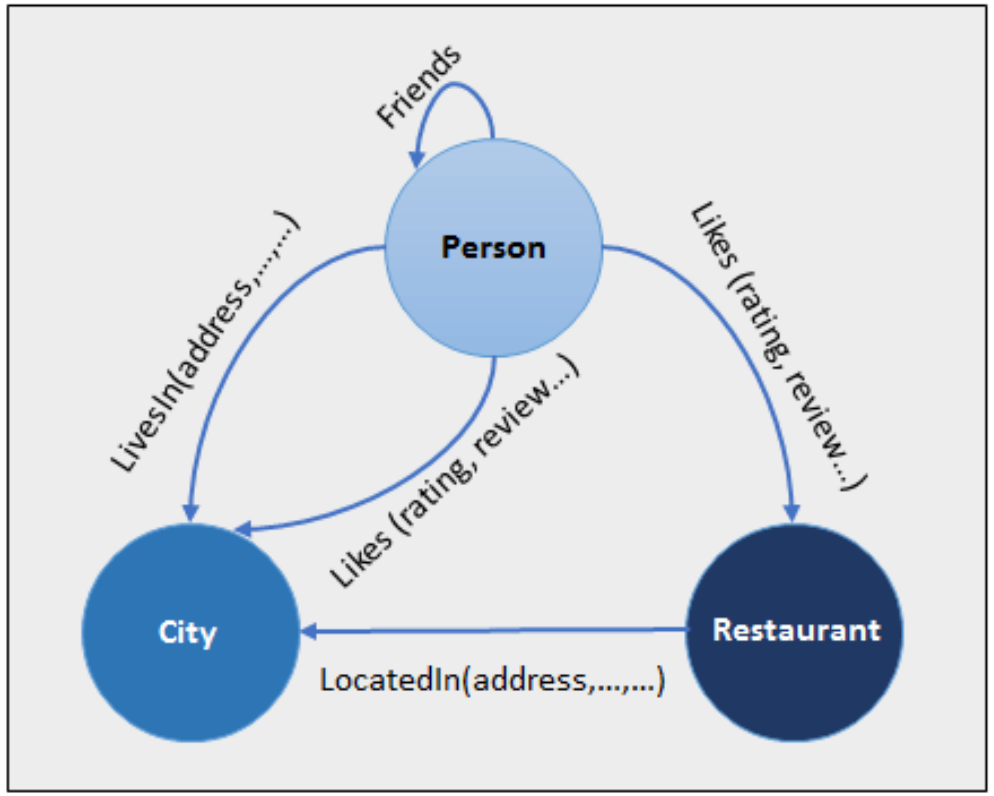
- They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
- widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

4. Graph databases:-

are well suited to model objects and relationships between objects.



- Stores entities and relations amongst those entities.
- The entity is stored as a node with the relationship as edges.
- An edge gives a relationship between nodes.
- Every node and edge has a unique identifier.



- Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature.
- Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.
- Graph base database mostly used for social networks, logistics, spatial data.

Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible

- NoSQL databases don’t need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption.

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn’t work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server etc	**Examples:** MongoDB, GraphQL, HBase, Neo4j, Cassandra etc

Data Management with Distributed Databases

- Databases are designed to do two things: store data and retrieve data.
- To meet these objectives, the database management systems must do three things:
- Store data persistently
- Maintain data consistency
- Ensure data availability

Data Management with Distributed Databases

- Store data persistently
- Data must be stored persistently; that is, it must be stored in a way that data is not lost when the database server is shut down.
- If data were only stored in memory—that is, RAM, then it would be lost when power to the memory is lost.
- Only data that is stored on disk, flash, tape, or other long-term storage is considered persistently stored.

Data Management with Distributed Databases

- Maintain data consistency
- It is important to ensure that the correct data is written to a persistent storage device.
- If the write or read operation does not accurately record or retrieve data, the database will not be of much use.
- This is rarely a problem unless there is a hardware failure.
- A more common issue with reading and writing occurs when two or more people are using the database and want to use the same data at the same time.

Data Management with Distributed Databases

- Ensure data availability
- Data should be available whenever it is needed.
- This is difficult to guarantee: Hardware may fail, An operating system on the database server may need patching, You might need to install a new version of the database management system.
- A database that runs on a single server can be unavailable for a large number of reasons.
- One way to avoid the problem of an unavailable database server is to have two database servers: One is used for updating data and responding to queries while the other is kept as a backup in case the first server fails.

Data Management with Distributed Databases

- Ensure data availability
- When the company used a single server, there was just one step in updating the number of a particular product in the warehouse.
- Now that the company is using a backup database, the number of chairs would have to be updated on the primary server and the backup server.
- The process for updating both databases is similar to other multistep transactions: Both databases must succeed for the operation to succeed.

- In pioneer days they used oxen for heavy pulling, and when one ox couldn’t budge a log, they didn’t try to grow a larger ox. We shouldn’t be trying for bigger computers, but for more systems of computers.

  - Grace Hopper

Grace Hopper

American computer scientist

Grace Brewster Murray Hopper was an American computer scientist and United States Navy rear admiral. One of the first programmers of the Harvard Mark I computer, she was a pioneer of computer programming who invented one of the first linkers.

Hadoop

- An open source framework for writing and running distributed applications that process large amounts of data.
- Key distinctions of Hadoop:
  - Accessible - Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's EC2.
  - Robust – It is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.
  - Scalable - It scales linearly to handle larger data by adding more nodes to the cluster.
  - Simple - It allows users to quickly write efficient parallel code.

Hadoop: Brief History

- Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library.

Hadoop: Brief History

- Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

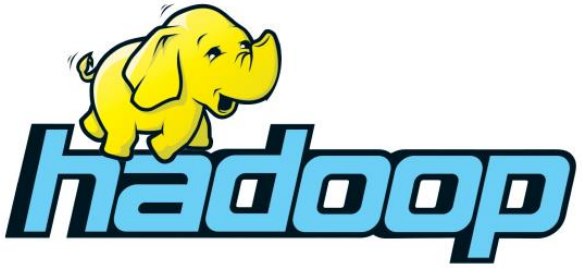
Hadoop: Brief History

- Nutch's creators realized that their architecture wouldn't scale to the billions of pages on the Web.

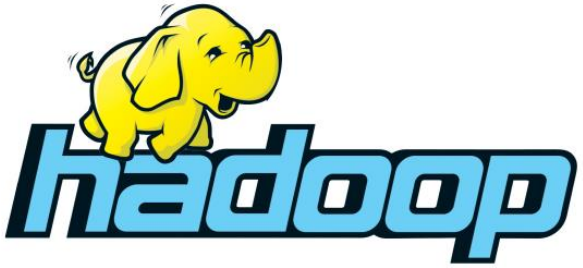
- Publication of a paper in 2003 that described the architecture of Google's distributed filesystem, called GFS, which was being used in Google.

Hadoop: Brief History

- In 2004, Google published the paper that introduced MapReduce to the world.
- Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS.
- in February 2006 they moved MapReduce and NDFS out of Nutch to form an independent subproject of Lucene called Hadoop.



Hadoop: Brief History



- At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale.
- This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000 core Hadoop cluster.

Hadoop: Brief History

- In January 2008, Hadoop was made its own top-level project under Apache, confirming its success and its diverse, active community.

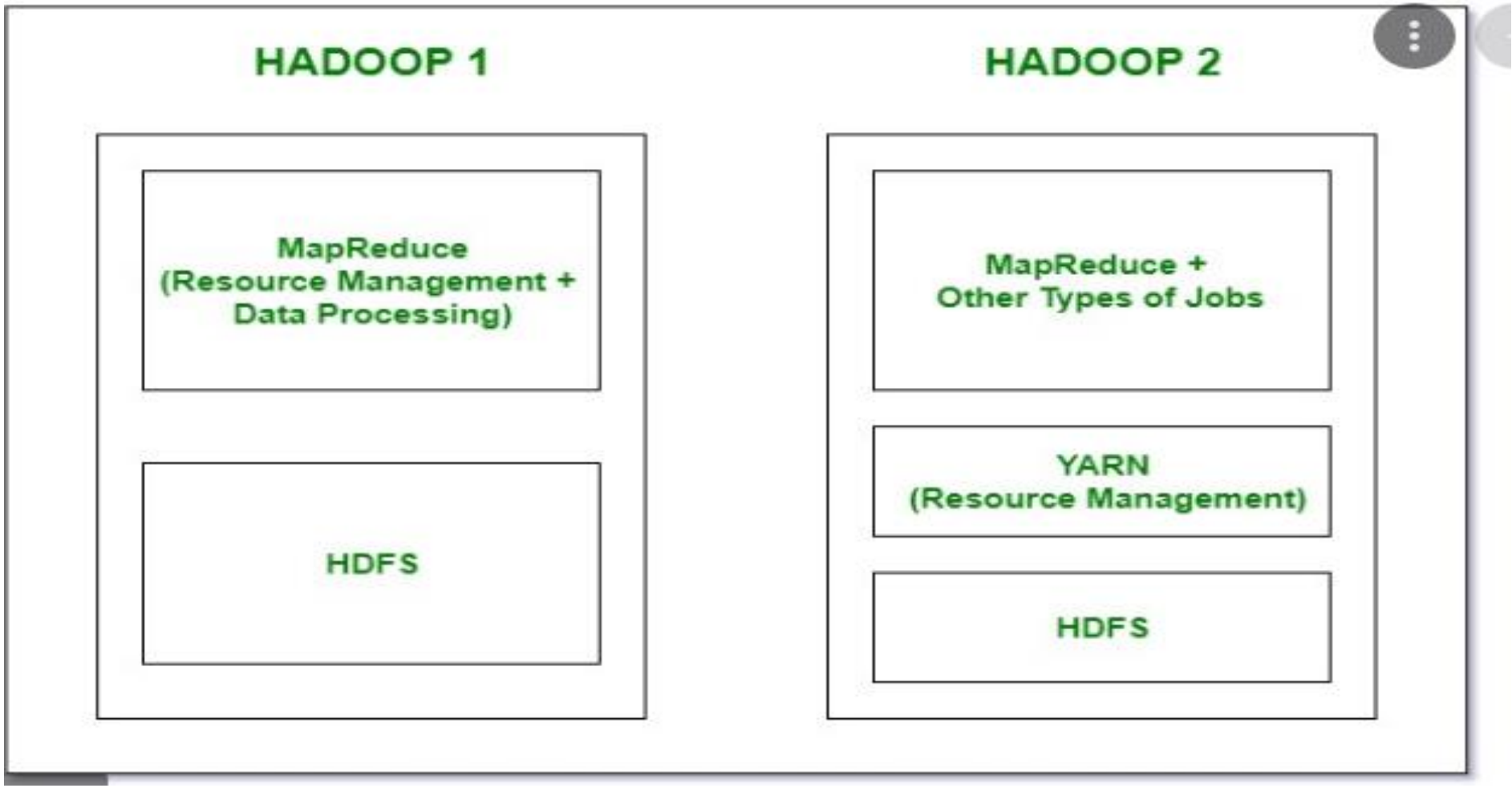
Hadoop: Brief History

- The name Hadoop is not an acronym; it's a made-up name.
- The project's creator, Doug Cutting, explains how the name came about: The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.

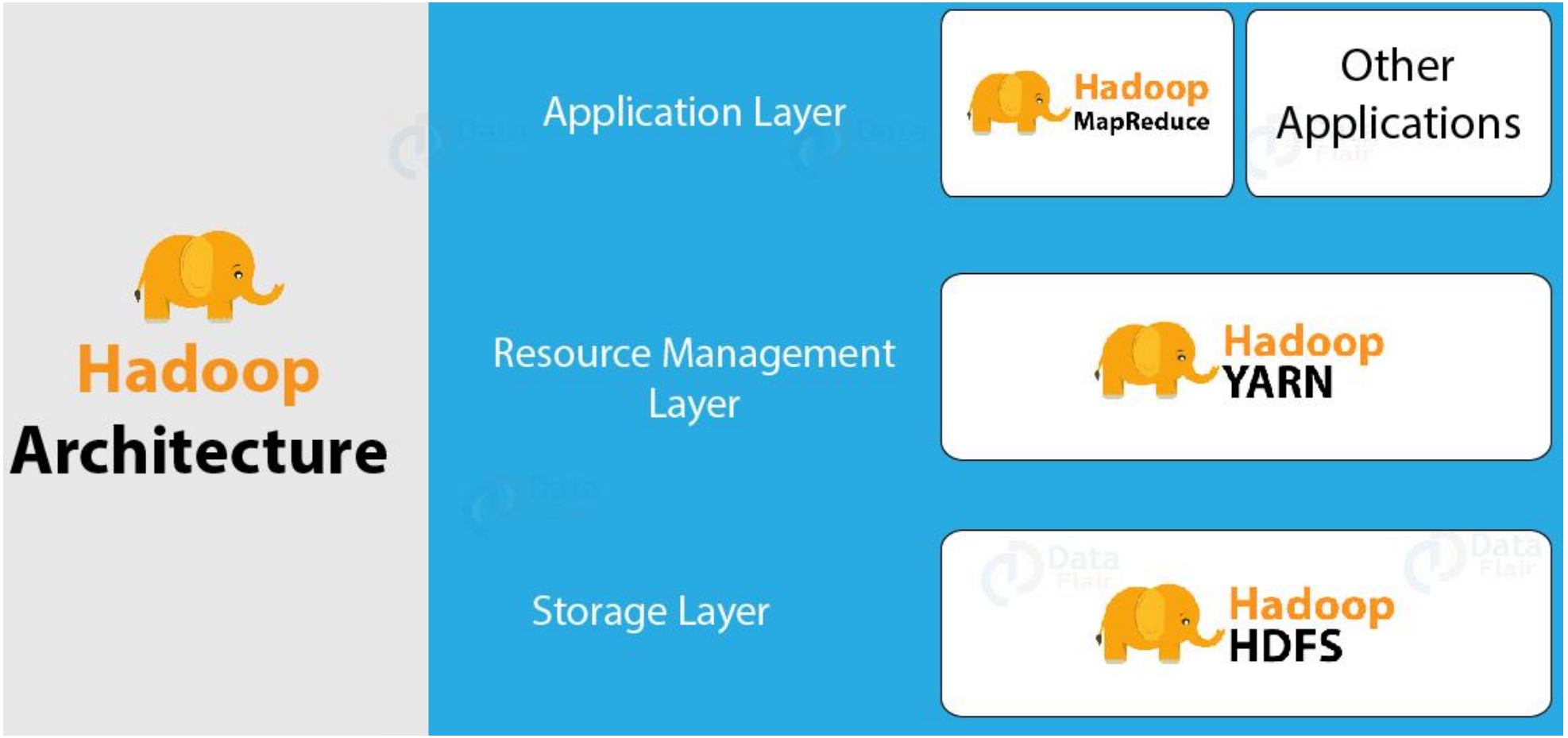
Features of Hadoop

1. Open Source
2. Highly Scalable Cluster
3. Fault Tolerance is Available
4. High Availability is Provided
5. Cost-Effective
6. Hadoop Provide Flexibility
7. Easy to Use
8. Hadoop uses Data Locality
9. Provides Faster Data Processing

Version of Hadoop

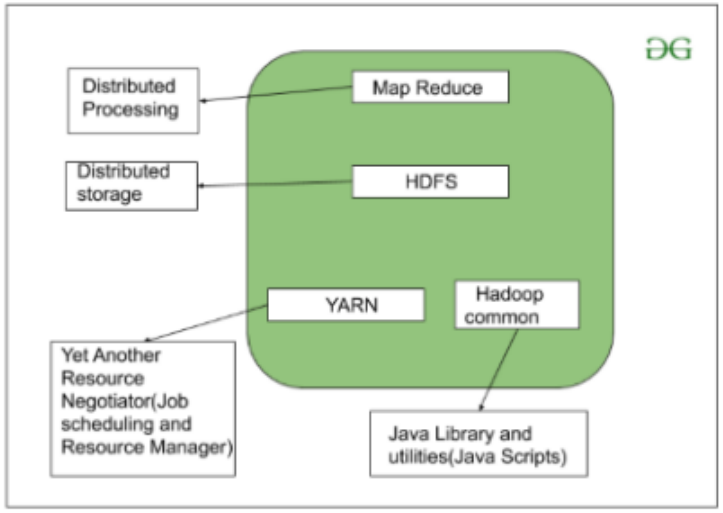


Hadoop: Architecture



Hadoop: Architecture

1. HDFS (Hadoop Distributed File System)
2. YARN(Yet Another Resource Nevigator)
3. MapReduce
4. Common Utilities or Hadoop Common



Hadoop: Architecture

1. HDFS

- HDFS stands for Hadoop Distributed File System.
- It provides for data storage of Hadoop.
- HDFS splits the data unit into smaller units called blocks and stores them in a distributed manner.

- It has got two daemons running.
- One for master node – NameNode and other for slave nodes – DataNode.

Hadoop: Architecture

1. HDFS

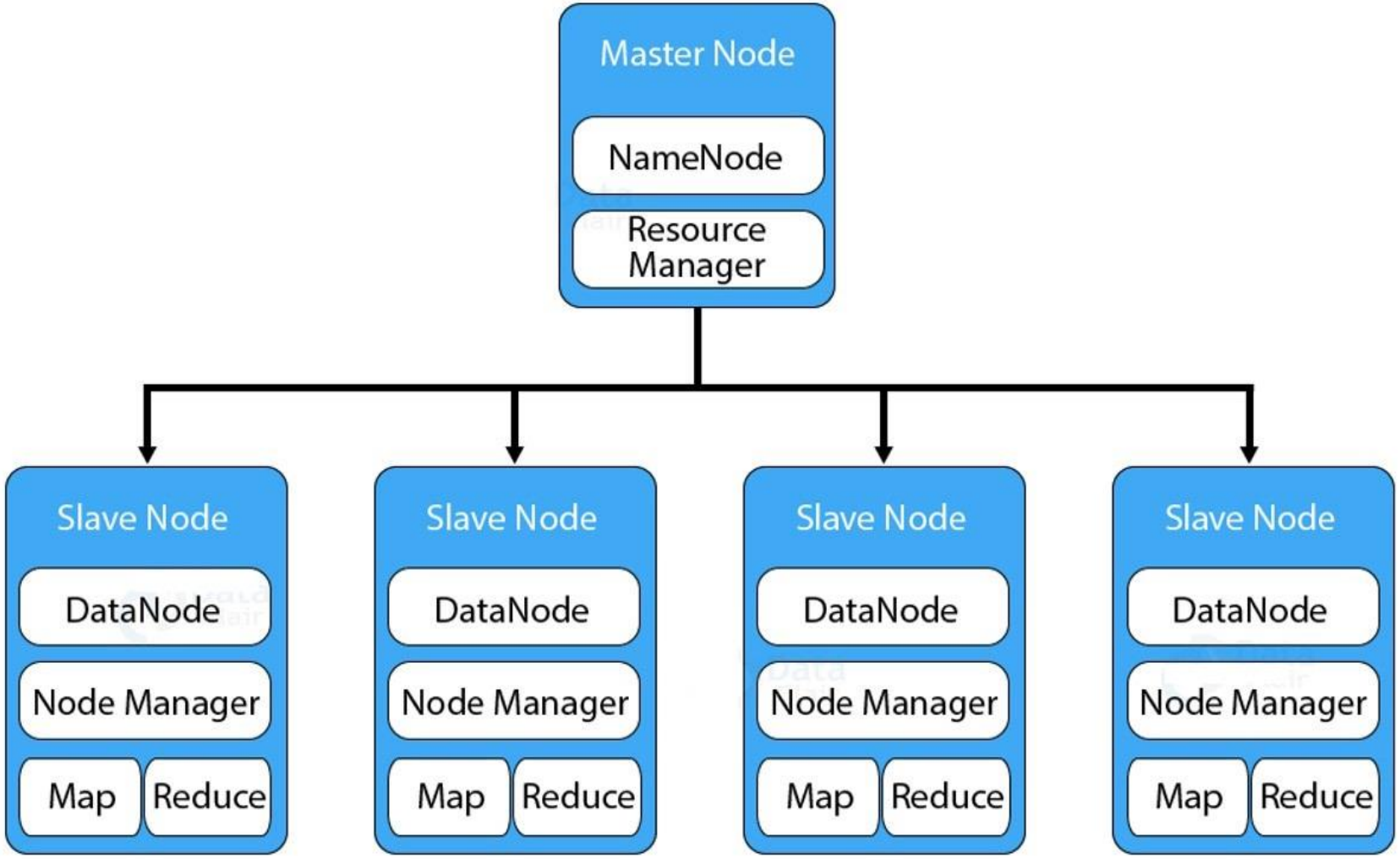
- HDFS has a Master-slave architecture.
- The daemon called NameNode runs on the master server.
- It is responsible for Namespace management and regulates file access by the client.
- DataNode daemon runs on slave nodes.
- It is responsible for storing actual business data.

Hadoop: Architecture

1. HDFS

- Internally, a file gets split into a number of data blocks and stored on a group of slave machines.
- Namenode manages modifications to file system namespace. These are actions like the opening, closing and renaming files or directories. NameNode also keeps track of mapping of blocks to DataNodes.
- DataNodes serves read/write request from the file system's client. They also creates, deletes and replicates blocks on demand from NameNode.

Hadoop: Architecture



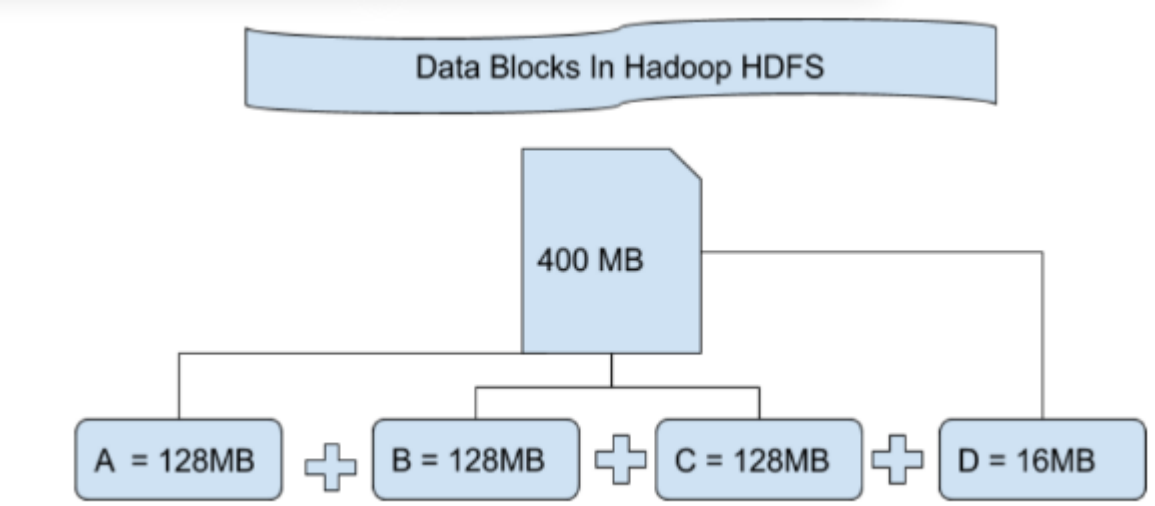
NameNode:

- NameNode works as a Master in a Hadoop cluster that guides the Datanode(Slaves).
- Namenode is mainly used for storing the Metadata i.e. the data about the data. Meta Data can be the transaction logs that keep track of the user's activity in a Hadoop cluster.
- Meta Data can also be the name of the file, size, and the information about the location(Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication.
- Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

DataNode:

- DataNodes works as a Slave DataNodes are mainly utilized for storing the data in a Hadoop cluster.
- The number of DataNodes can be from 1 to 500 or even more than that.
- The more number of DataNode, the Hadoop cluster will be able to store more data.
- So it is advised that the DataNode should have High storing capacity to store a large number of file blocks.

- **File Block In HDFS:** Data in HDFS is always stored in terms of blocks. So the single block of data is divided into multiple blocks of size 128MB which is default and you can also change it manually.



Replication In HDFS

- Replication ensures the availability of the data.
- Replication is making a copy of something and the number of times you make a copy of that particular thing can be expressed as it's Replication Factor.
- As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks.
- By default, the Replication Factor for Hadoop is set to 3 which can be configured means you can change it manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of 4×3 = 12 blocks are made for the backup purpose.

- This is because for running Hadoop we are using commodity hardware (inexpensive system hardware) which can be crashed at any time.
- We are not using the supercomputer for our Hadoop setup.
- That is why we need such a feature in HDFS which can make copies of that file blocks for backup purposes, this is known as fault tolerance.
- After making so many replica's of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage so nobody cares for this extra storage. You can configure the Replication factor in your *hdfs-site.xml* file.

Rack Awareness

- The rack is nothing but just the physical collection of nodes in our Hadoop cluster (maybe 30 to 40).
- A large Hadoop cluster is consists of so many Racks .
- with the help of this Racks information Namenode chooses the closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic.

basic HDFS File operations

1. Creating a directory:
Syntax: hdfs dfs –mkdir <path>
Eg. hdfs dfs –mkdir /chp

2. Remove a file in specified path:
Syntax: hdfs dfs –rm <src>
Eg. hdfs dfs –rm /chp/abc.txt

3. Copy file from local file system to hdfs:
Syntax: hdfs dfs –copyFromLocal <src> <dst>
Eg. hdfs dfs –copyFromLocal /home/hadoop/sample.txt /chp/abc1.txt

4. To display list of contents in a directory:
Syntax: hdfs dfs –ls <path>
Eg. hdfs dfs –ls /chp

5. To display contents in a file:
Syntax: hdfs dfs –cat <path>
Eg. hdfs dfs –cat /chp/abc1.txt

6. Copy file from hdfs to local file system:
Syntax: hdfs dfs –copyToLocal <src <dst>
Eg. hdfs dfs –copyToLocal /chp/abc1.txt /home/hadoop/Desktop/sample.txt

7. To display last few lines of a file:
Syntax: hdfs dfs –tail <path>
Eg. hdfs dfs –tail /chp/abc1.txt

8. Display aggregate length of file in bytes:
Syntax: hdfs dfs –du <path>
Eg. hdfs dfs –du /chp

9. To count no.of directories, files and bytes under given path:
Syntax: hdfs dfs –count <path>
Eg. hdfs dfs –count /chp
o/p: 1 1 60

10. Remove a directory from hdfs
Syntax: hdfs dfs –rmr <path>
Eg. hdfs dfs rmr /chp

Hadoop: Architecture

2. MapReduce

- MapReduce is the data processing layer of Hadoop.
- It is a software framework that allows you to write applications for processing a large amount of data. MapReduce runs these applications in parallel on a cluster of low-end machines.
- It does so in a reliable and fault-tolerant manner.

Map Task:

RecordReader

- The purpose of *recordreader* is to break the records.
- It is responsible for providing key-value pairs in a Map() function.
- The key is actually is its locational information and value is the data associated with it.

Map:

- A map is nothing but a user-defined function whose work is to process the Tuples obtained from record reader.
- The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.

Combiner:

- Combiner is used for grouping the data in the Map workflow.
- It is similar to a Local reducer.
- The intermediate key-value that are generated in the Map is combined with the help of this combiner.
- Using a combiner is not necessary as it is optional.

Partitionar:

- Partitional is responsible for fetching key-value pairs generated in the Mapper Phases.
- The partitioner generates the shards corresponding to each reducer.
- Hashcode of each key is also fetched by this partition.
- Then partitioner performs it's(Hashcode) modulus with the number of reducers (*key.hashcode()%(number of reducers)).*

Reduce Task

Shuffle and Sort:

- The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as *Shuffling*.
- Using the Shuffling process the system can sort the data using its key value.
- Once some of the Mapping tasks are done, Shuffling begins so it is a faster process and does not wait for the completion of the task performed by Mapper.

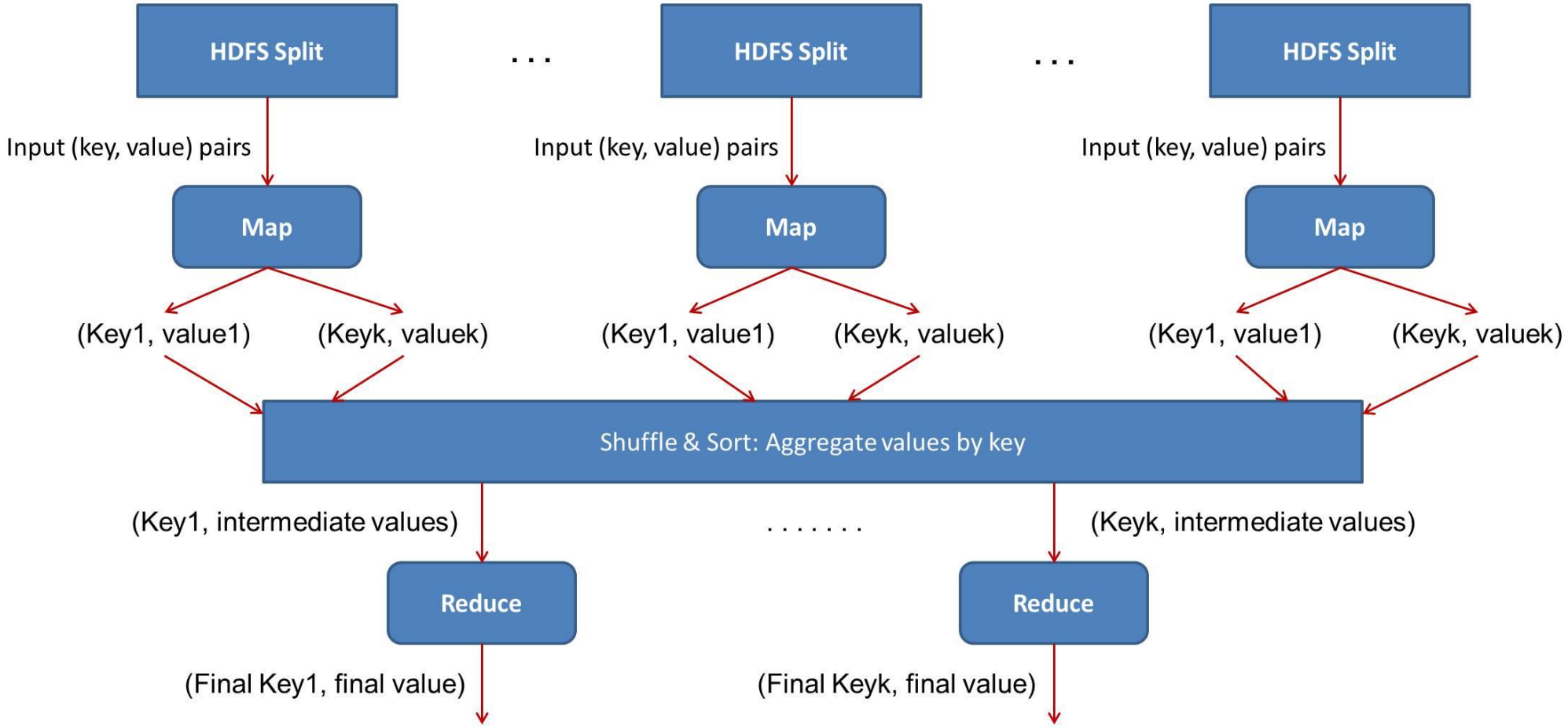
Reduce:

- The main function or task of the Reduce is to gather the Tuple generated from Map and then perform some sorting and aggregation sort of process on those key-value depending on its key element.

OutputFormat:

- Once all the operations are performed, the key-value pairs are written into the file with the help of record writer, each record in a new line, and the key and value in a space-separated manner.

Hadoop: Architecture



Hadoop: Architecture

3. YARN

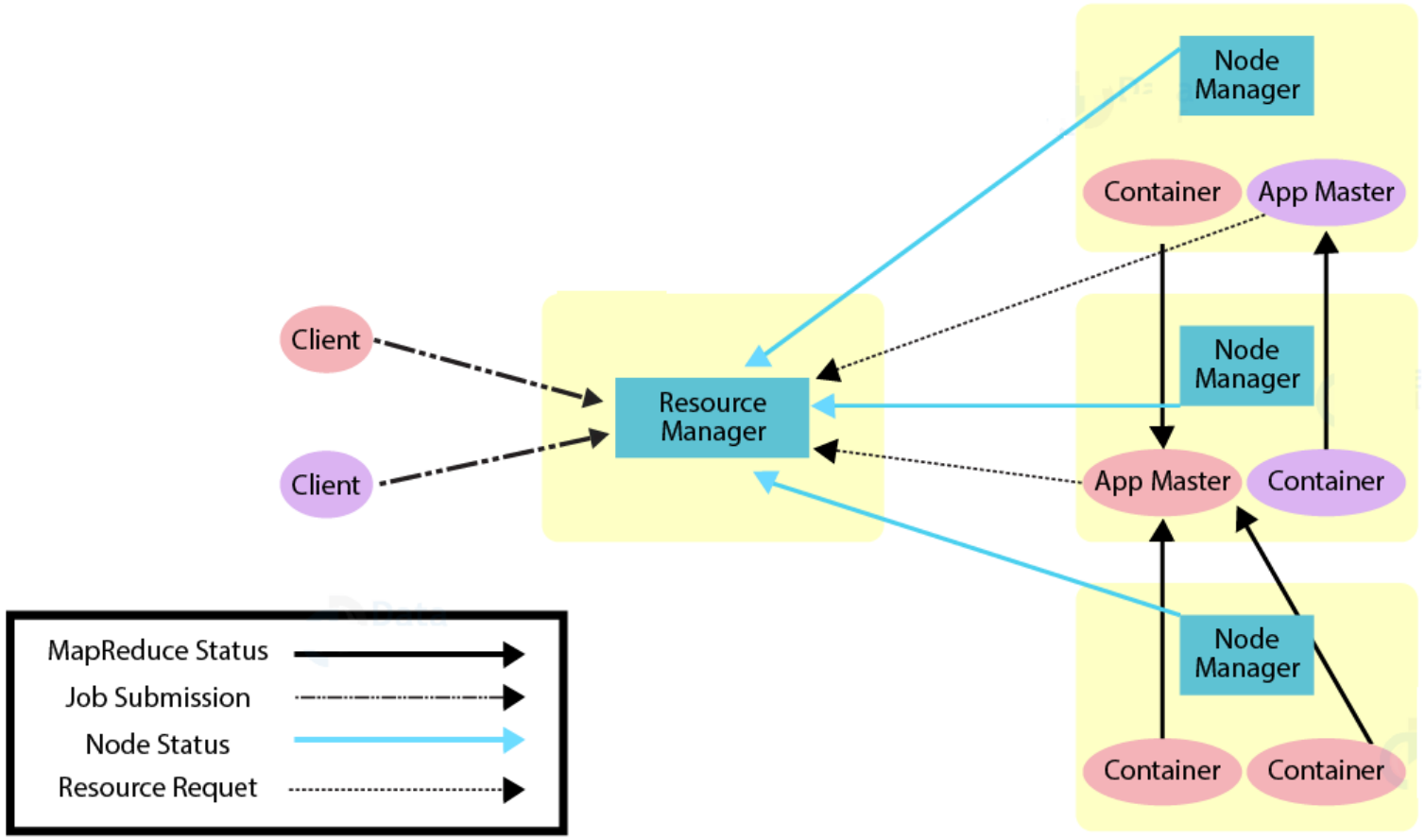
- YARN or Yet Another Resource Negotiator is the resource management layer of Hadoop.
- The basic principle behind YARN is to separate resource management and job scheduling/monitoring function into separate daemons.
- In YARN there is one global ResourceManager and per-application ApplicationMaster.

Hadoop: Architecture

3. YARN

- Inside the YARN framework, we have two daemons ResourceManager and NodeManager.
- The ResourceManager mediates resources among all the competing applications in the system.
- The job of NodeManger is to monitor the resource usage by the container and report the same to ResourceManger.
- The resources are like CPU, memory, disk, network and so on.
- The ApplcationMaster negotiates resources with ResourceManager and works with NodeManger to execute and monitor the job.

Hadoop: Architecture



Features of YARN

- Multi-Tenancy
- Scalability
- Cluster-Utilization
- Compatibility

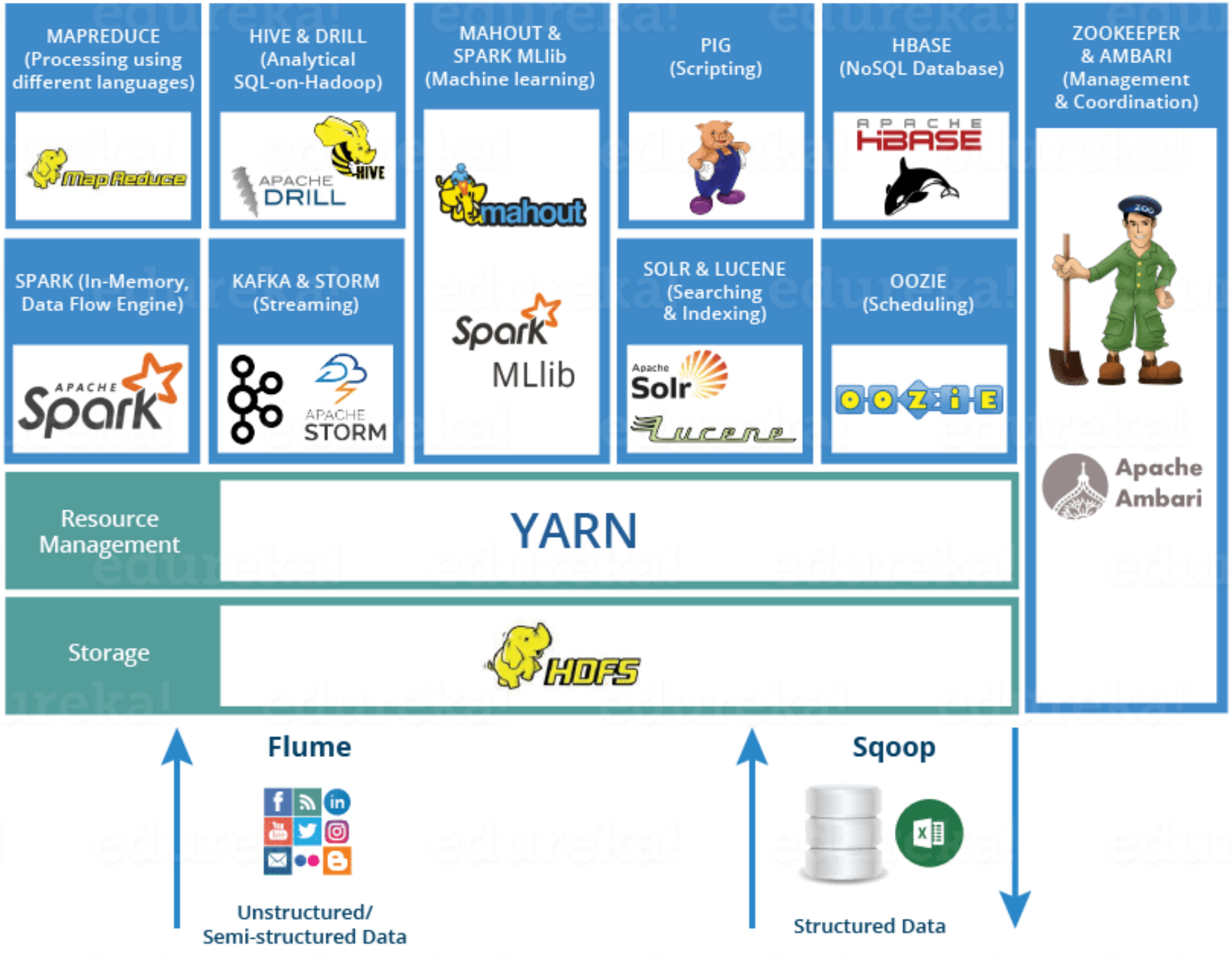
4 . Hadoop common or Common Utilities

- Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster.
- These utilities are used by HDFS, YARN, and MapReduce for running the cluster.
- Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

Feature	Hadoop	SQL
Technology	Modern	Traditional
Volume	Usually in PetaBytes	Usually in GigaBytes
Operations	Storage, processing, retrieval and pattern extraction from data	Storage, processing, retrieval and pattern mining of data
Fault Tolerance	Hadoop is highly fault tolerant	SQL has good fault tolerance
Storage	Stores data in the form of key-value pairs, tables, hash map etc in distributed systems.	Stores structured data in tabular format with fixed schema in cloud
Scaling	Linear	Non linear
Providers	Cloudera, Horton work, AWS etc. provides Hadoop systems.	Well-known industry leaders in SQL systems are Microsoft, SAP, Oracle etc.
Data Access	Batch oriented data access	Interactive and batch oriented data access
Cost	It is open source and systems can be cost effectively scaled	It is licensed and costs a fortune to buy a SQL server, moreover if system runs out of storage additional charges also emerge

Time	Statements are executed very quickly	SQL syntax is slow when executed in millions of rows
Optimization	It stores data in HDFS and process though Map Reduce with huge optimization techniques.	It does not have any advanced optimization techniques
Structure	Dynamic schema, capable of storing and processing log data, real-time data, images, videos, sensor data etc.(both structured and unstructured)	Static Schema, capable of storing data(fixed schema) in tabular format only(structured)
Data Update	Write data once, read data multiple times	Read and Write data multiple times
Integrity	Low	High
Interaction	Hadoop uses JDBC(Java Database Connectivity) to communicate with SQL systems to send and receive data	SQL systems can read and write data to Hadoop systems
Hardware	Uses commodity hardware	Uses propriety hardware
Training	Learning Hadoop for entry-level as well as seasoned profession is moderately hard	Learning SQL is easy for even entry-level professionals

Hadoop Eco System



PIG:

- Pig was basically developed by Yahoo which works on a pig Latin language, which is Query based language similar to SQL.
- It is a platform for structuring the data flow, processing and analyzing huge data sets.
- Pig does the work of executing commands and in the background, all the activities of MapReduce are taken care of. After the processing, pig stores the result in HDFS.
- Pig Latin language is specially designed for this framework which runs on Pig Runtime. Just the way Java runs on the JVM.
- Pig helps to achieve ease of programming and optimization and hence is a major segment of the Hadoop Ecosystem.

HIVE:

- With the help of SQL methodology and interface, HIVE performs reading and writing of large data sets. However, its query language is called as HQL (Hive Query Language).
- It is highly scalable as it allows real-time processing and batch processing both. Also, all the SQL datatypes are supported by Hive thus, making the query processing easier.
- Similar to the Query Processing frameworks, HIVE too comes with two components: *JDBC Drivers* and *HIVE Command Line*.
- JDBC, along with ODBC drivers work on establishing the data storage permissions and connection whereas HIVE Command line helps in the processing of queries.

Mahout:

- Mahout, allows Machine Learnability to a system or application.
- Machine Learning, as the name suggests helps the system to develop itself based on some patterns, user/environmental interaction or on the basis of algorithms.
- It provides various libraries or functionalities such as collaborative filtering, clustering, and classification which are nothing but concepts of Machine learning.
- It allows invoking algorithms as per our need with the help of its own libraries.

Apache Spark:

- It’s a platform that handles all the process consumptive tasks like batch processing, interactive or iterative real-time processing, graph conversions, and visualization, etc.
- It consumes in memory resources hence, thus being faster than the prior in terms of optimization.
- Spark is best suited for real-time data whereas Hadoop is best suited for structured data or batch processing, hence both are used in most of the companies interchangeably.

Apache HBase:

- It’s a NoSQL database which supports all kinds of data and thus capable of handling anything of Hadoop Database.
- It provides capabilities of Google’s BigTable, thus able to work on Big Data sets effectively.
- At times where we need to search or retrieve the occurrences of something small in a huge database, the request must be processed within a short quick span of time.
- At such times, HBase comes handy as it gives us a tolerant way of storing limited data.

Other Components:

Apart from all of these, there are some other components too that carry out a huge task in order to make Hadoop capable of processing large datasets. They are as follows:

- **Solr, Lucene:**
- These are the two services that perform the task of searching and indexing with the help of some java libraries.
- Especially Lucene is based on Java which allows spell check mechanism, as well. However, Lucene is driven by Solr.

Zookeeper:

- There was a huge issue of management of coordination and synchronization among the resources or the components of Hadoop which resulted in inconsistency, often.
- Zookeeper overcame all the problems by performing synchronization, inter-component based communication, grouping, and maintenance.

Oozie:

- Oozie simply performs the task of a scheduler, thus scheduling jobs and binding them together as a single unit.
- There is two kinds of jobs .i.e Oozie workflow and Oozie coordinator jobs.
- Oozie workflow is the jobs that need to be executed in a sequentially ordered manner
- Oozie Coordinator jobs are those that are triggered when some data or external stimulus is given to it.

S.No.	RDBMS	Hadoop
1.	Traditional row-column based databases, basically used for data storage, manipulation and retrieval.	An open-source software used for storing data and running applications or processes concurrently.
2.	In this structured data is mostly processed.	In this both structured and unstructured data is processed.
3.	It is best suited for OLTP environment.	It is best suited for BIG data.
4.	It is less scalable than Hadoop.	It is highly scalable.
5.	Data normalization is required in RDBMS.	Data normalization is not required in Hadoop.
6.	It stores transformed and aggregated data.	It stores huge volume of data.
7.	It has no latency in response.	It has some latency in response.
8.	The data schema of RDBMS is static type.	The data schema of Hadoop is dynamic type.
9.	High data integrity available.	Low data integrity available than RDBMS.
10.	Cost is applicable for licensed software.	Free of cost, as it is an open source software.

Word Count program in hadoop.

MapReduce consists of 2 steps:

•Map Function –

It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

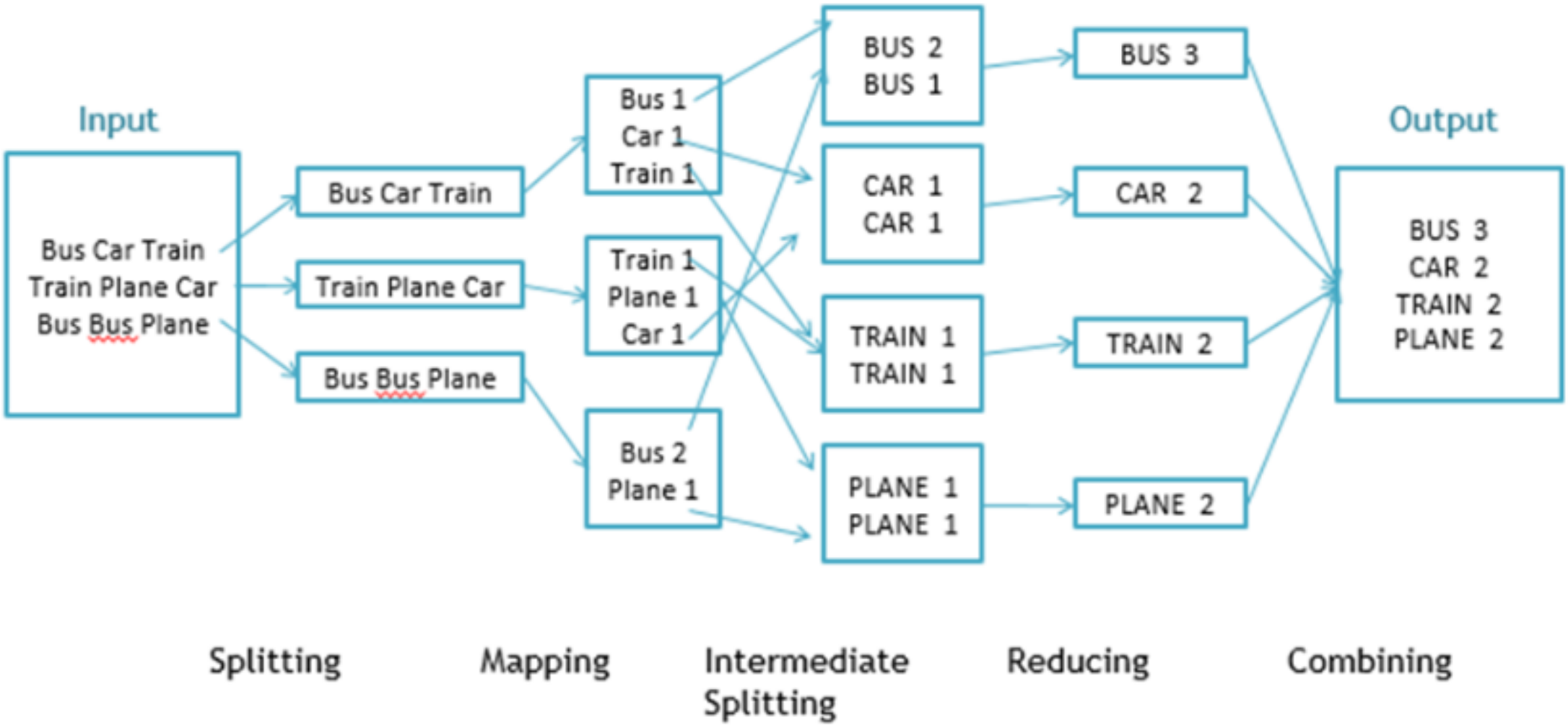
Example – (Map function in Word Count)

Input	Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function –

- Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.
- **Example –** (Reduce function in Word Count)

Input (output of Map function)	Set of Tuples	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)
Output	Converts into smaller set of tuples	(BUS,7), (CAR,7), (TRAIN,4)


Input
Bus Car Train
Train Plane Car
Bus Bus Plane
Bus Car Train
Train Plane Car
Bus Bus Plane
Bus 1
Car 1
Train 1
Train 1
Plane 1
Car 1
Bus 2
Plane 1
BUS 2
BUS 1
CAR 1
CAR 1
TRAIN 1
TRAIN 1
PLANE 1
PLANE 1
BUS 3
CAR 2
TRAIN 2
PLANE 2
Output
BUS 3
CAR 2
TRAIN 2
PLANE 2
Splitting
Mapping
Intermediate Splitting
Reducing
Combining

Content of Input text file:

- Hello World Bye World
- Hello Hadoop Goodbye Hadoop

- Output of Mapper - 1
- <Hello, 1>
- <World, 1>
- <Bye, 1>
- <World, 1>

Output of Mapper - 2

<Hello, 1>

<Hadoop, 1>

<Goodbye, 1>

<Hadoop, 1>

- Output Shuffle Step:

<Bye, 1>

<Goodbye, 1>

<Hadoop, 2>

<Hello, 1>

<Hello, 1>

<World, 2>

Output of Combiner - 1

<Bye, 1>

<Hello, 1>

<World, 2>

Output of Combiner - 2

<Goodbye, 1>

<Hadoop, 2>

<Hello, 1>

```
import java.io.IOException;import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

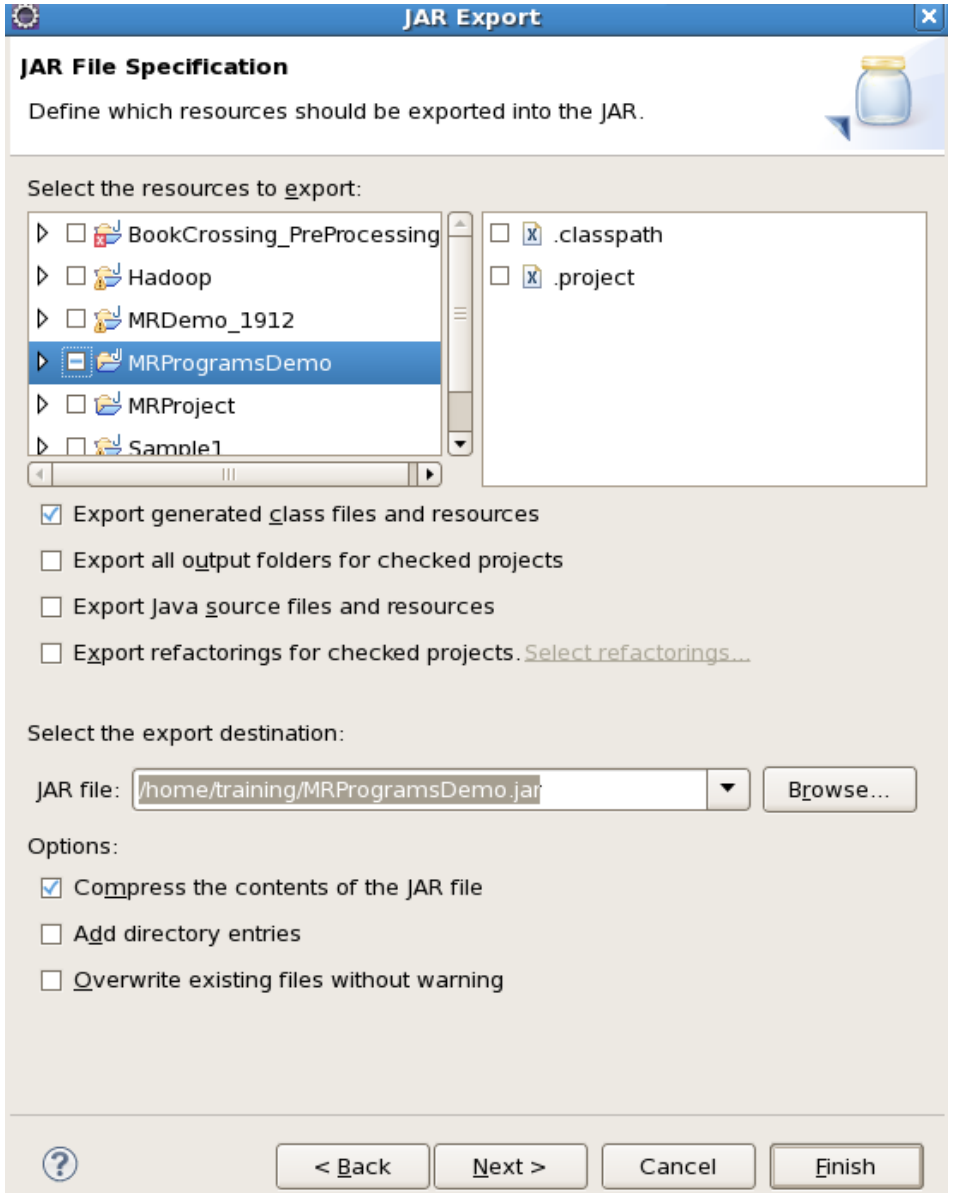
```
public class WordCount
{
        public static class TokenizerMapper      extends Mapper<Object, Text, Text, IntWritable>
        {
                private Text word = new Text();
                private final static IntWritable one = new IntWritable(1);
                public void map(Object key, Text value, Context context  ) throws IOException,
InterruptedException
                {
                        StringTokenizer itr = new StringTokenizer(value.toString());
                        while (itr.hasMoreTokens())
                        {
                                word.set(itr.nextToken());
                                context.write(word, one);
                        }
                }
}
```

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

```
public static void main(String[] args) throws Exception
{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Steps

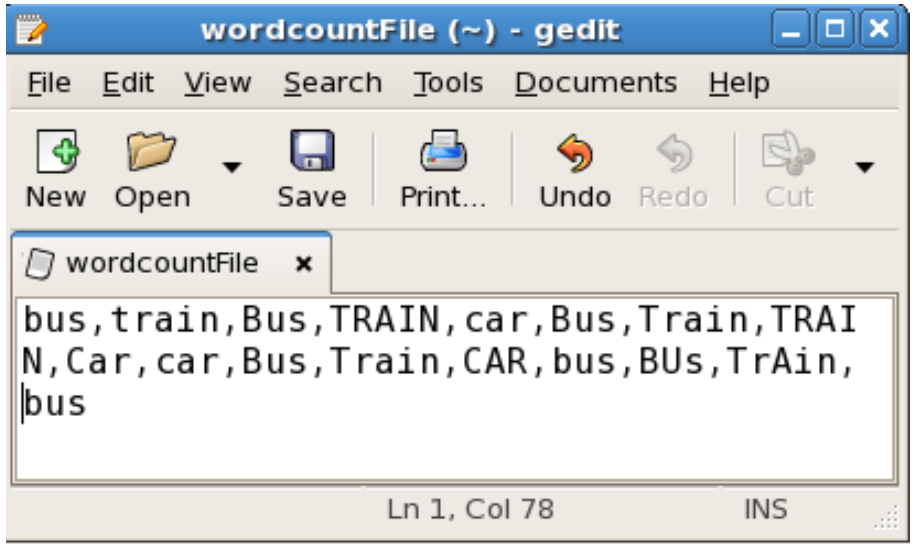
1. Open Eclipse> File > New > Java Project >(Name it – MRProgramsDemo) > Finish.
2. Right Click > New > Package (Name it - PackageDemo) > Finish.
3. Right Click on Package > New > Class (Name it - WordCount).
4. Add Following Reference Libraries:
    1. Right Click on Project > Build Path> Add External
        1. */usr/lib/hadoop-0.20/**hadoop-core.jar***
        2. *Usr/lib/hadoop-0.20/lib/**Commons-cli-1.2.jar***



Make a jar file

Right Click on Project> Export> Select export destination as **Jar File** > next> Finish.

7. Take a text file and move it into HDFS format



To move this into Hadoop directly, open the terminal and enter the following commands

```
[training@localhost ~]$ hadoop fs -put wordcountFile wordCountFile
```

8. Run the jar file:

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

```
[training@localhost ~]$ hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1
```

Birla Vishvakarma Mahavidyalaya
Engineering College, Vallabh Vidyanagar

4CP02: DATA ANALYTICS AND VISUALIZATION

Unit 3

Distributed Computing Challenges

Distributed Computing Challenges

- Heterogeneity: A system consisting of multiple distinct components.
- Hardware devices: computers, tablets, mobile phones, embedded devices, etc.
- Operating System: Ms Windows, Linux, Mac, Unix, etc.
- Network: Local network, the Internet, wireless network, satellite links, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles of software developers, designers, system managers

Distributed Computing Challenges

- Heterogeneity: A system consisting of multiple distinct components.
- In order to overcome heterogeneity, a software layer known as Middleware is often used to hide the differences amongst the components underlying layers.
- The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages.

Distributed Computing Challenges

- Heterogeneity: A system consisting of multiple distinct components.
- Most middleware is implemented over the Internet protocols, which themselves mask the differences of the underlying networks, but all middleware deals with the differences in operating systems and hardware.

Distributed Computing Challenges

- Transparency: Concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.
- Distributed systems designers must hide the complexity of the systems as much as they can.

Distributed Computing Challenges

- Transparency
- Access: Hide differences in data representation and how a resource is accessed
- Location: Hide where a resource is located
- Migration: Hide that a resource may move to another location
- Relocation: Hide that a resource may be moved to another location while in use

Distributed Computing Challenges

- Transparency
- Replication: Hide that a resource may be copied in several places
- Concurrency: Hide that a resource may be shared by several competitive users
- Failure: Hide the failure and recovery of a resource
- Persistence: Hide whether a (software) resource is in memory or a disk

Distributed Computing Challenges

- Openness: Property of each subsystem to be open for interaction with other systems.
- The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.
- The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

Distributed Computing Challenges

- Openness: Property of each subsystem to be open for interaction with other systems.
- If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future.
- Example: Twitter and Facebook have API that allows developers to develop theirs own software interactively.

Distributed Computing Challenges

- Concurrency
- Both services and applications provide resources that can be shared by clients in a distributed system.
- Concurrency issues arise when several clients attempt to request a shared resource at the same time.
- This is problematic as the outcome of any such data may depend on the execution order, and so synchronization is required.

Distributed Computing Challenges

- Concurrency
- For example, a data structure that records bids for an auction may be accessed very frequently when it gets close to the deadline time.
- For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent.
- This can be achieved by standard techniques such as semaphores, which are used in most operating systems.

Distributed Computing Challenges

- Security
- Security for information resources has three components:

1. Confidentiality: protection against disclosure to unauthorized individuals
2. Integrity: protection against alteration or corruption
3. Availability: for the authorized protection against interference with the means to access the resources.

Distributed Computing Challenges

- Scalability
- As the system, number of resources, or users increase the performance of the system is not lost and remains effective in accomplishing its goals.
- Defined by B. Clifford Neuman: A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.

Distributed Computing Challenges

- Scalability
- Scalability has 3 dimensions:

1. Size: Number of users and resources to be processed. Problem associated is overloading.
2. Geography: Distance between users and resources. Problem associated is communication reliability.
3. Administration: As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative mess.

Distributed Computing Challenges

- Failure Handling
- Failures are inevitable in any system; some components may stop functioning while others continue running normally.
- Way to handle faults is:

1. Detect Failures – Various mechanisms can be employed such as checksums.
2. Mask Failures – retransmit upon failure to receive acknowledgement.

Distributed Computing Challenges

- Failure Handling
- Way to handle faults is:

3. Recover from failures – if a server crashes roll back to previous state.
4. Build Redundancy – Redundancy is the best way to deal with failures. It is achieved by replicating data so that if one sub system crashes another may still be able to provide the required information.

Availability and Consistency in Distributed Databases

- When two database servers must keep consistent copies of data, they incur longer times to complete a transaction.
- This is acceptable in applications that require both consistency and high availability at all times.
- Ex. Financial systems
- But there are applications in which the fast database operations are more important than maintaining consistency at all times.
- Ex. E-commerce site

Availability and Consistency in Distributed Databases

- One is programming the user interface for an e-commerce site.
- How long should the customer wait after clicking on an “Add to My Cart” button?

- The interface would respond immediately so the customer could keep shopping.

Availability and Consistency in Distributed Databases

- One way to achieve fast response is to write the updates to one database and then let the program know the data has been saved.
- The interface can indicate to the customer that the product has been added to the cart. 
- While the customer receives the message that the cart has been updated, the database management system is making a copy of the newly updated data and writing it to another server.

Availability and Consistency in Distributed Databases

- There is a brief period of time when the customer's cart on the two servers is not consistent, but the customer is able to continue shopping anyway.
- In this case, we are willing to tolerate inconsistency for a brief period of time knowing that eventually the two carts will have the same products in it.
- This is especially true with online shopping carts because there is only a small chance someone else would read that customer's cart data.

CAP Theorem

- Also known as Brewer’s theorem
- States that distributed databases cannot have consistency (C), availability (A), and partition protection (P) all at the same time.
- Consistency, in this case, means consistent copies of data on different servers.
- Availability refers to providing a response to any query.
- Partition protection means if a network that connects two or more database servers fails, the servers will still be available with consistent data.

CAP Theorem

- We saw in a previous example of the e-commerce shopping cart that it is possible to have a backup copy of the cart data that is out of sync with the primary copy.
- The data would still be available if the primary server failed, but the data on the backup server would be inconsistent with data on the primary server if the primary server failed prior to updating the backup server.

CAP Theorem

- In the two-phase commit we can have consistency but at the risk of the most recent data not being available for a brief period of time.
- While the two-phase commit is executing, other queries to the data are blocked.
- The updated data is unavailable until the two-phase commit finishes.
- This favors consistency over availability.

CAP Theorem

- Partition protection deals with situations in which servers cannot communicate with each other.
- This would be the case in the event of a network failure.
- This splitting of the network into groups of devices that can communicate with each other from those that cannot is known as partitioning.
- Partitioning has multiple meanings in data management.
- It is important to remember that when talking about the CAP theorem, partitioning has to do with the inability to send messages between database servers.

CAP Theorem

- If database servers running the same distributed database are partitioned by a network failure, then you could continue to allow both to respond to queries and preserve availability but at the risk of them becoming inconsistent.
- Alternatively, you could disable one so that only one of the servers responds to queries.
- This would avoid returning inconsistent data to users querying different servers but at the cost of availability to some users.

CAP Theorem

- From a practical standpoint, network partitions are rare, at least in local area networks.
- This means that database application designers have to deal with the trade-offs between consistency and availability more than issues with partitioning.

BASE

- Basically Available, Soft State, Eventually Consistent
- BA is for basically available.
- This means that there can be a partial failure in some parts of the distributed system and the rest of the system continues to function.
- For example, if a NoSQL database is running on 10 servers without replicating data and one of the servers fails, then 10% of the users' queries would fail, but 90% would succeed.
- NoSQL databases often keep multiple copies of data on different servers.

BASE

- This allows the database to respond to queries even if one of the servers has failed.
- S is for soft state.
- Usually in computer science, the term soft state means data will expire if it is not refreshed.
- In NoSQL operations, it refers to the fact that data may eventually be overwritten with more recent data.
- E is for eventually consistent.
- This means that there may be times when the database is in an inconsistent state.

BASE

- For example, some NoSQL databases keep multiple copies of data on multiple servers.
- There is a possibility that the multiple copies may not be consistent for a short period of time.
- This can occur when a user or program updates one copy of the data and other copies continue to have the old version of the data.
- Eventually, the replication mechanism in the NoSQL database will update all copies, but in the meantime, the copies are inconsistent.
- The time it takes to update all copies depends on several factors, such as the load on the system and the speed of the network.

Eventual Consistency

- In ACID (atomicity, consistency, isolation, durability), consistency relates to the guarantee that when a transaction is finished the database is in a consistent state.
- For example, when transferring money from one account to another the total amount held in both accounts should not change.
- In ACID-based systems, this kind of consistency is often the responsibility of the developer writing the transaction but can be assisted by the database managing integrity constraints.

Client-side Consistency: Components

- A storage system: It is something of large scale and highly distributed, and that it is built to guarantee durability and availability.
- Process A: This is a process that writes to and reads from the storage system.
- Processes B and C: These two processes are independent of process A and write to and read from the storage system.
- Processes need to communicate to share information.

Client-side Consistency

- Client-side consistency has to do with how and when observers (in this case the processes A, B, or C) see updates made to a data object in the storage systems.
- In the following examples illustrating the different types of consistency, process A has made an update to a data object:
- Strong consistency: After the update completes, any subsequent access (by A, B, or C) will return the updated value.

Client-side Consistency

- Weak consistency: The system does not guarantee that subsequent accesses will return the updated value.
- A number of conditions need to be met before the value will be returned.
- The period between the update and the moment when it is guaranteed that any observer will always see the updated value is called the inconsistency window.

Client-side Consistency

- Eventual consistency: This is a specific form of weak consistency; the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value.
- If no failures occur, the maximum size of the inconsistency window can be determined based on factors such as communication delays, the load on the system, and the number of replicas involved in the replication scheme.
- The most popular system that implements eventual consistency is DNS (Domain Name System).

Types of Eventual Consistency

- Casual consistency
- Read-your-writes consistency
- Session consistency
- Monotonic read consistency
- Monotonic write consistency

Types of Eventual Consistency

- Casual consistency
- Casual consistency ensures that the database reflects the order in which operations were updated.
- If process A has communicated to process B that it has updated a data item, a subsequent access by process B will return the updated value, and a write is guaranteed to supersede the earlier write.
- Access by process C that has no causal relationship to process A is subject to the normal eventual consistency rules.

Types of Eventual Consistency

- Read-your-writes consistency
- Read-your-writes consistency means that once you have updated a record, all of your reads of that record will return the updated value.
- You would never retrieve a value inconsistent with the value you had written.
- This is an important model where process A, after it has updated a data item, always accesses the updated value and will never see an older value.
- This is a special case of the causal consistency model.

Types of Eventual Consistency

- Session consistency
- Session consistency ensures read-your-writes consistency during a session.
- You can think of a session as a conversation between a client and a server or a user and the database.
- As long as the conversation continues, the database “remembers” all writes you have done during the conversation.
- This is a practical version of the previous model, where a process accesses the storage system in the context of a session.

Types of Eventual Consistency

- Session consistency
- If the session ends and you start another session with the same server, there is no guarantee it will “remember” the writes you made in the previous session.
- A session may end if you log off an application using the database or if you do not issue commands to the database for so long that the database assumes you no longer need the session and abandons it.

Types of Eventual Consistency

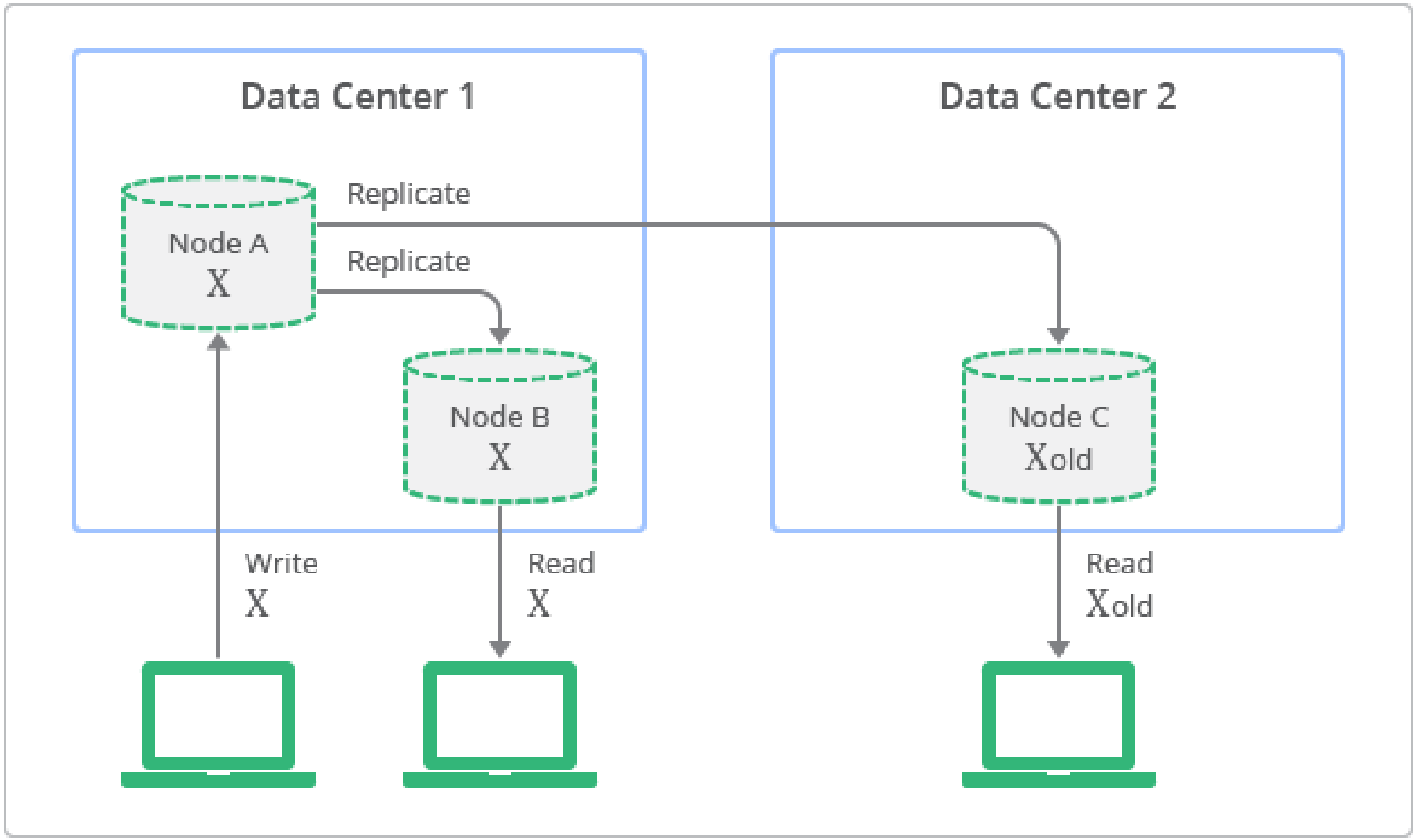
- Monotonic read consistency
- Monotonic read consistency ensures that if you issue a query and see a result, you will never see an earlier version of the value.
- If a process has seen a particular value for the object, any subsequent accesses will never return any previous values.

Types of Eventual Consistency

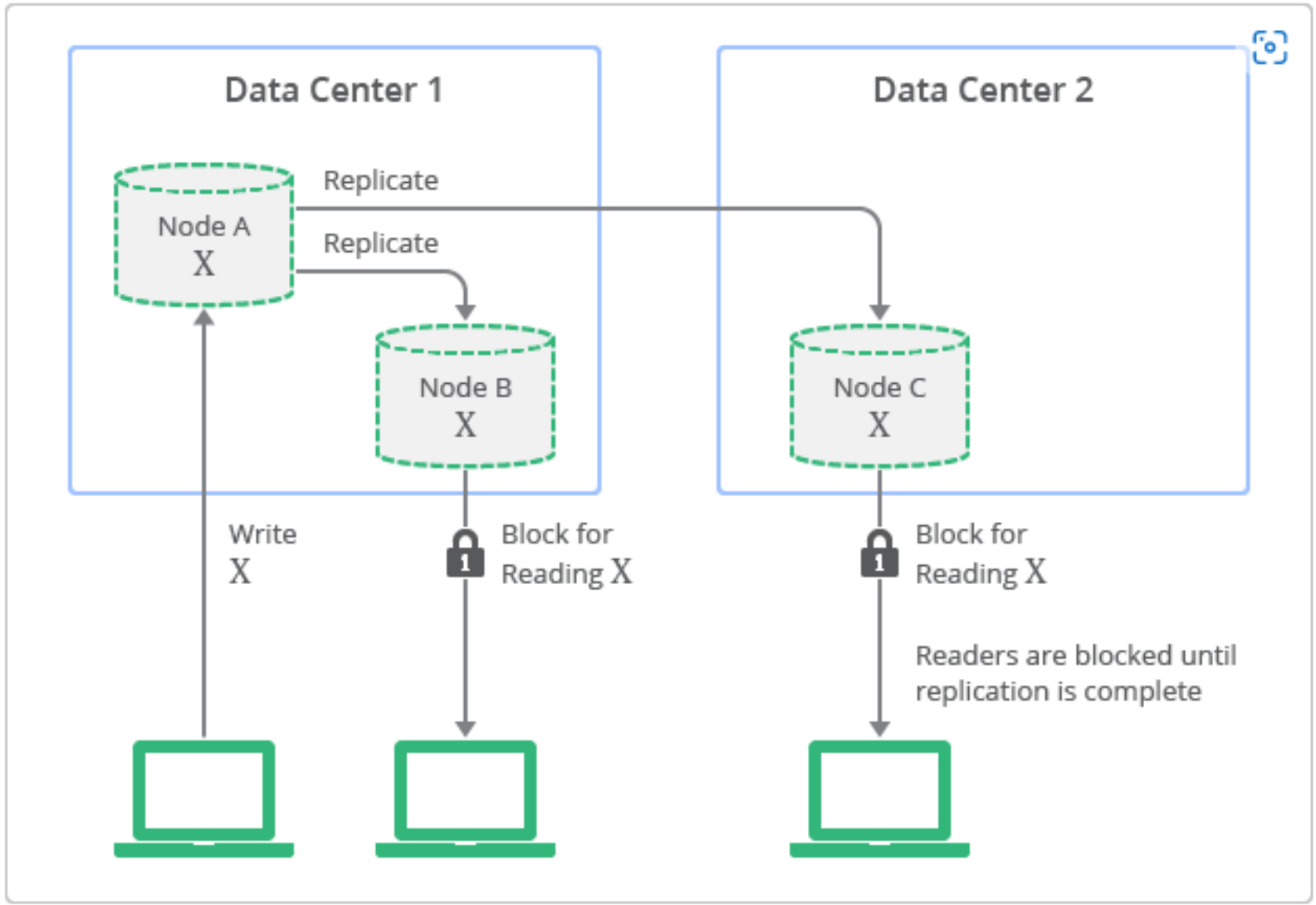
- Monotonic write consistency
- Monotonic write consistency ensures that if you were to issue several update commands, they would be executed in the order you issued them.
- In this case the system guarantees to serialize the writes by the same process.

Types of Eventual Consistency

- A number of these properties can be combined.
- For example, one can get monotonic reads combined with session-level consistency.
- From a practical point of view these two properties (monotonic reads and read-your-writes) are most desirable in an eventual consistency system, but not always required.
- These two properties make it simpler for developers to build applications, while allowing the storage system to relax consistency and provide high availability.



Eventual Consistency (Source - Google's blog)



Strong Consistency

Hadoop HDFS Data Write Operation

- To write a file in HDFS, a client needs to interact with master i.e. **namenode** (master).
- Now namenode provides the address of the **datanodes** (slaves) on which client will start writing the data.
- Client directly writes data on the datanodes, now datanode will create data write pipeline.
- The first datanode will copy the block to another datanode, which intern copy it to the third datanode.
- Once it creates the replicas of blocks, it sends back the acknowledgment.

HDFS Data Write Pipeline Workflow

i) The HDFS client sends a **create** request on *DistributedFileSystem* APIs.

ii) *DistributedFileSystem* makes an RPC call to the namenode to create a new file in the file system's namespace.

The namenode performs various checks to make sure that the file doesn't already exist and that the client has the permissions to create the file.

When these checks pass, then only the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an *IOException.*

iii) The *DistributedFileSystem* returns a *FSDataOutputStream* for the client to start writing data to.

As the client writes data, *DFSOutputStream* splits it into packets, which it writes to an internal queue, called the data queue.

The data queue is consumed by the *DataStreamer, which* is responsible for asking the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas.

iv) The list of datanodes form a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline.

The *DataStreamer* streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline.

Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline.

v) *DFSOutputStream* also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the *ack queue*.

A packet is removed from the ack queue only when it has been acknowledged by the datanodes in the pipeline.

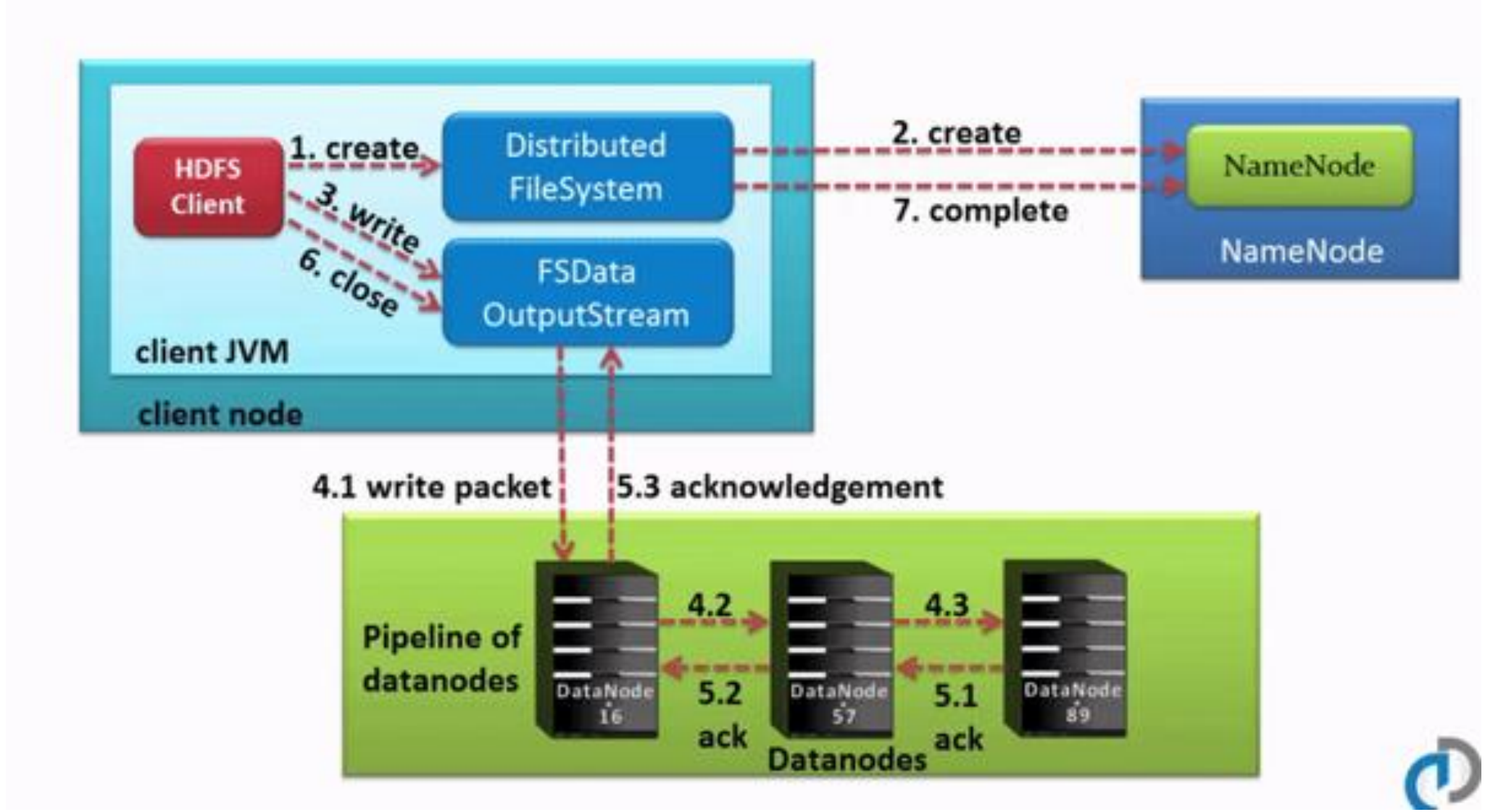
Datanode sends the acknowledgment once required replicas are created (3 by default).

Similarly, all the blocks are stored and replicated on the different datanodes, the data blocks are copied in parallel.

vi) When the client has finished writing data, it calls **close()** on the stream.

vii) This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete.

The namenode already knows which blocks the file is made up of, so it only has to wait for blocks to be minimally replicated before returning successfully.


HDFS Client
1. create
3. write
6. close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2. create
7. complete
NameNode
NameNode
4.1 write packet
5.3 acknowledgement
Pipeline of datanodes
4.2
5.2 ack
4.3
5.1 ack
DataNode 16
DataNode 57
DataNode 89
Datanodes

Hadoop HDFS Data Read Operation

- To read a file from HDFS, a client needs to interact with namenode (master) as namenode is the centerpiece of Hadoop cluster (it stores all the metadata i.e. data about the data).
- Now namenode checks for required privileges, if the client has sufficient privileges then namenode provides the address of the slaves where a file is stored.
- Now client will interact directly with the respective datanodes to read the data blocks.

HDFS File Read Workflow

i) Client opens the file it wishes to read by calling **open()** on the *FileSystem* object, which for HDFS is an instance of *DistributedFileSystem*.

ii) *DistributedFileSystem* calls the namenode using RPC to determine the locations of the blocks for the first few blocks in the file.

For each block, the namenode returns the addresses of the datanodes that have a copy of that block and datanode are sorted according to their proximity to the client.

iii) *DistributedFileSystem* returns a *FSDataInputStream* to the client for it to read data from.

FSDataInputStream, thus, wraps the *DFSInputStream* which manages the datanode and namenode I/O. Client calls **read()** on the stream.

DFSInputStream which has stored the datanode addresses then connects to the closest datanode for the first block in the file.

iv) Data is streamed from the datanode back to the client, as a result client can call **read()** repeatedly on the stream.

When the block ends, DFSInputStream will close the connection to the datanode and then finds the best datanode for the next block.

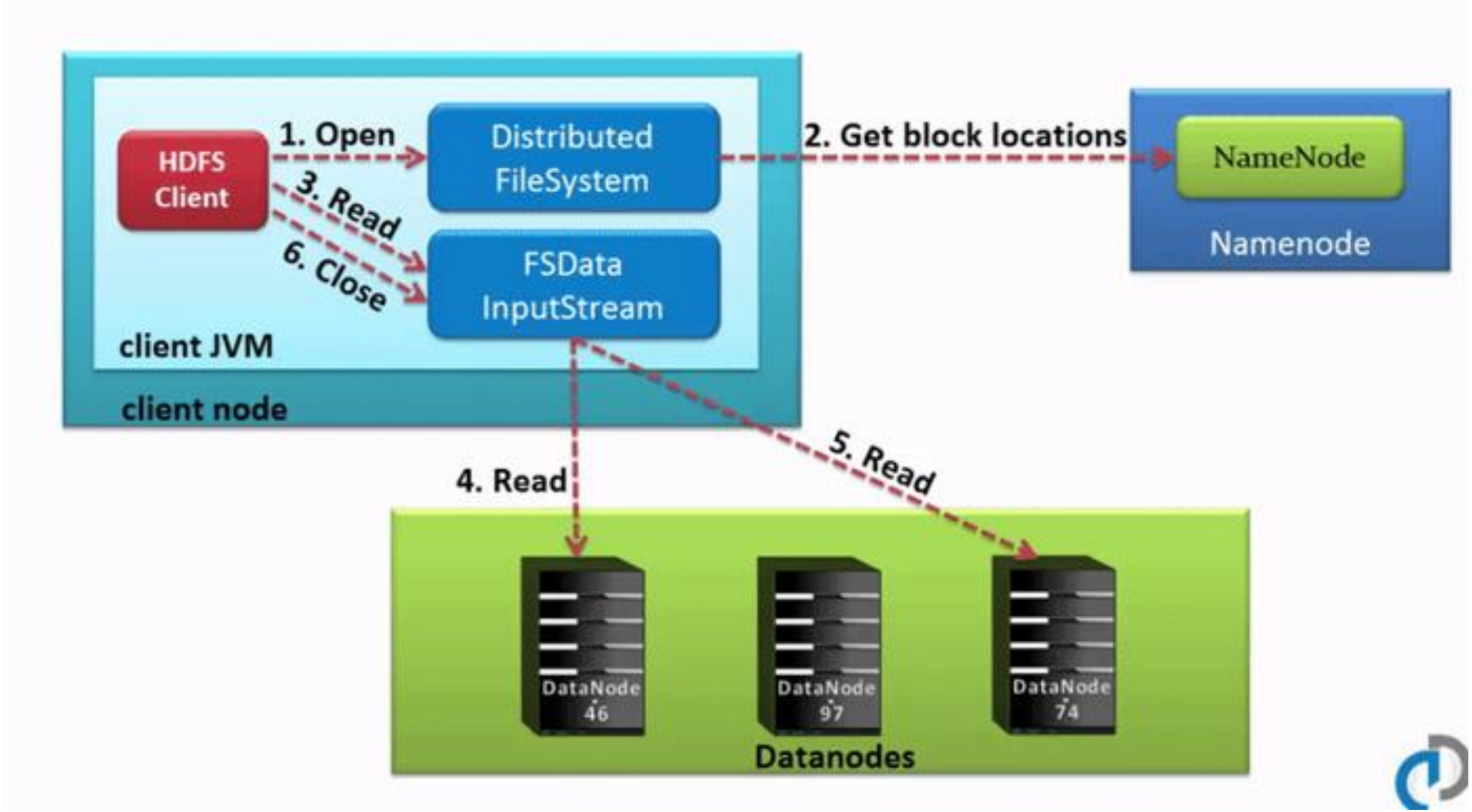
v) If the *DFSInputStream* encounters an error while communicating with a datanode, it will try the next closest one for that block.

It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks.

The *DFSInputStream* also verifies checksums for the data transferred to it from the datanode.

If it finds a corrupt block, it reports this to the namenode before the *DFSInputStream* attempts to read a replica of the block from another datanode.

vi) When the client has finished reading the data, it calls **close()** on the stream.


HDFS Client
1. Open
3. Read
6. Close
Distributed FileSystem
FSData InputStream
client JVM
client node
2. Get block locations
NameNode
Namenode
4. Read
5. Read
DataNode 46
DataNode 97
DataNode 74
Datanodes

CDRConstants.java

```
1  package org.example.hadoopcodes;
2
3  public class CDRConstants {
4
5      public static int fromPhoneNumber = 0;
6      public static int toPhoneNumber = 1;
7      public static int callStartTime = 2;
8      public static int callEndTime = 3;
9      public static int STDFlag = 4;
10
11 }
```

CDRConstants.java | STDSubscribers.java

```
1  package org.example.hadoopcodes;
2
3  import java.io.IOException;
4  import java.text.ParseException;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7
8  import org.apache.hadoop.conf.Configuration;
9  import org.apache.hadoop.fs.Path;
10 import org.apache.hadoop.io.LongWritable;
11 import org.apache.hadoop.io.Text;
12 import org.apache.hadoop.mapreduce.Job;
13 import org.apache.hadoop.mapreduce.Mapper;
14 import org.apache.hadoop.mapreduce.Reducer;
15 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
17
```

```
69    public static class TokenizerMapper extends
70            Mapper<Object, Text, Text, LongWritable> {
71
72        Text phoneNumber = new Text();
73        LongWritable durationInMinutes = new LongWritable();
74
75        public void map(Object key, Text value,
76                Mapper<Object, Text, Text, LongWritable>.Context context)
77                throws IOException, InterruptedException {
78            String[] parts = value.toString().split("[|]");
79            if (parts[CDRConstants.STDFlag].equalsIgnoreCase("1")) {
80
81                phoneNumber.set(parts[CDRConstants.fromPhoneNumber]);
82                String callEndTime = parts[CDRConstants.callEndTime];
83                String callStartTime = parts[CDRConstants.callStartTime];
84                long duration = toMillis(callEndTime) - toMillis(callStartTime);
85                durationInMinutes.set(duration / (1000 * 60));
86                context.write(phoneNumber, durationInMinutes);
87            }
88        }
```

```
50    public static class SumReducer extends
51            Reducer<Text, LongWritable, Text, LongWritable> {
52        private LongWritable result = new LongWritable();
53
54        public void reduce(Text key, Iterable<LongWritable> values,
55                Reducer<Text, LongWritable, Text, LongWritable>.Context context)
56                throws IOException, InterruptedException {
57            long sum = 0;
58            for (LongWritable val : values) {
59                sum += val.get();
60            }
61            this.result.set(sum);
62            if (sum >= 60) {
63                context.write(key, this.result);
64            }
65
66        }
67    }
```

```
31 public class STDSubscribers {
32     public static void main(String[] args) throws Exception {
33         Configuration conf = new Configuration();
34         if (args.length != 2) {
35             System.err.println("Usage: stdsubscriber <in> <out>");
36             System.exit(2);
37         }
38         Job job = new Job(conf, "STD Subscribers");
39         job.setJarByClass(STDSubscribers.class);
40         job.setMapperClass(TokenizerMapper.class);
41         job.setCombinerClass(SumReducer.class);
42         job.setReducerClass(SumReducer.class);
43         job.setOutputKeyClass(Text.class);
44         job.setOutputValueClass(LongWritable.class);
45         FileInputFormat.addInputPath(job, new Path(args[0]));
46         FileOutputFormat.setOutputPath(job, new Path(args[1]));
47         System.exit(job.waitForCompletion(true) ? 0 : 1);
48     }
```