Here's a **complete, in-depth explanation of Hadoop** in the context of **Parallel and Distributed Computing**, with architecture, components, examples, and use cases.

Hadoop: Parallel & Distributed Computing

1. Definition of Hadoop

Hadoop is an **open-source framework** for **distributed storage and parallel processing of large datasets** across clusters of computers using simple programming models.

Exam-ready one-line definition:

Hadoop is an open-source framework that allows distributed storage and parallel processing of large datasets across clusters of computers in a fault-tolerant and scalable manner.

---

2. Why Hadoop is Needed

Traditional computing problems:

- Data too large for a single machine
- Processing is slow
- Risk of hardware failure

Hadoop solves this by:

- **Distributing data across multiple machines**
- **Parallel processing** using multiple nodes
- **Fault tolerance** (replication of data)
- **Scalability** (add more nodes easily)

---

3. Core Features of Hadoop

1. **Scalability:** Can scale from single server to thousands of machines.
2. **Fault Tolerance:** Automatically replicates data.
3. **Cost-Effective:** Runs on commodity hardware.
4. **Flexibility:** Handles structured, semi-structured, and unstructured data.
5. **High Throughput:** Processes large data in parallel.

---

4. Hadoop Architecture

Hadoop has **two main layers**:

4.1 HDFS (Hadoop Distributed File System)

- Distributed file storage system.
- Splits files into **blocks** and stores them across cluster nodes.
- Each block is **replicated** for fault tolerance (default 3 copies).

Components:

- **NameNode:** Master node, stores metadata.
- **DataNode:** Slave nodes, store actual data.

4.2 MapReduce

- Parallel data processing framework.
- Processes large datasets **in parallel** on multiple nodes.

Phases:

1. **Map Phase:** Data is processed and transformed into key-value pairs.
2. **Shuffle & Sort:** Keys are grouped together.
3. **Reduce Phase:** Aggregation or computation on grouped keys.

---

5. Hadoop Ecosystem Components

Component	Role
HDFS	Distributed storage
MapReduce	Distributed computation
YARN	Resource management & scheduling
Hive	SQL-like query on Hadoop
Pig	Script-based data processing
HBase	NoSQL database
Sqoop	Data transfer from RDBMS to Hadoop
Flume	Streaming data ingestion
Oozie	Workflow scheduler

---

6. How Hadoop Supports Parallel Computing

- **Data is split into blocks** (default 128 MB).
- **Map tasks** run on different nodes simultaneously.
- **Reduce tasks** aggregate results.
- Maximum **CPU utilization** across cluster.

Example: Word Count

1. Input: Text files on HDFS
2. Map Phase: Count words in each block
3. Shuffle: Group same words
4. Reduce: Sum counts per word
5. Output: Final word count

---

7. Hadoop Example (Simplified)

Data:

```
File1: "apple banana apple"
File2: "banana apple orange"
```

Map Output:

```
("apple", 1), ("banana", 1), ("apple", 1)
("banana", 1), ("apple", 1), ("orange", 1)
```

Shuffle & Sort:

```
("apple", [1,1,1])
("banana", [1,1])
("orange", [1])
```

Reduce Output:

```
("apple", 3)
("banana", 2)
("orange", 1)
```

---

8. Fault Tolerance in Hadoop

- HDFS replicates blocks (default 3 times).
- NameNode tracks block locations.

- DataNode failures automatically handled.
- MapReduce tasks re-run if a node fails.

---

9. Hadoop vs OpenStack / AWS / Globus / Condor

Feature	Hadoop	OpenStack	AWS	Globus	Condor
Type	Framework	Cloud platform	Cloud platform	Middleware	Job scheduler
Focus	Big Data storage & processing	Cloud infrastructure	Cloud services	Grid computing	High-throughput tasks
Parallelism	MapReduce tasks	VMs, containers	EC2, Lambda	Distributed tasks	Independent jobs
Distributed	Across cluster nodes	Across cloud nodes	Across regions/zones	Across grid nodes	Across idle machines
Fault Tolerance	Built-in replication & task re-run	Multi-zone	Multi-zone replication	Depends on middleware	Job migration & checkpointing

---

10. Real-World Use Cases

- Facebook: Hive and HBase on Hadoop
- Yahoo: Log processing and analytics
- Amazon: Big data analytics
- Weather forecasting: Parallel processing of climate data
- Financial data analysis: Large-scale computations

---

11. Advantages of Hadoop

- Scalable to thousands of nodes
- Cost-effective (commodity hardware)
- Fault tolerant
- Handles structured & unstructured data
- Enables parallel processing across cluster

---

12. Limitations of Hadoop

- Not ideal for low-latency or real-time processing
- Complex setup & management
- Programming with MapReduce requires learning
- High disk I/O overhead

---

13. Hadoop in Parallel & Distributed Computing

- Hadoop **splits data into blocks** → executes tasks on multiple nodes (parallelism)
- Cluster nodes work together → distributed computing
- Fault tolerance ensures reliability
- Scales dynamically with nodes

---

14. Exam-Oriented Summary

Keywords:

- HDFS, MapReduce, YARN, parallel processing, distributed storage, fault tolerance, cluster computing

One-Paragraph Answer:

Hadoop is an open-source framework for distributed storage and parallel processing of large datasets across clusters of computers. Using HDFS for data storage and MapReduce for computation, Hadoop splits data into blocks and executes tasks simultaneously on multiple nodes. It provides fault tolerance through replication, high scalability, and is widely used for big data analytics, scientific simulations, and large-scale data processing.

If you want, I can now create a **diagram showing Hadoop architecture with parallel processing flow**, which is **very useful for exams and understanding distributed computing visually**.

Do you want me to make that diagram? Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets. Its framework is based on Java programming with some native code in C and shell scripts.

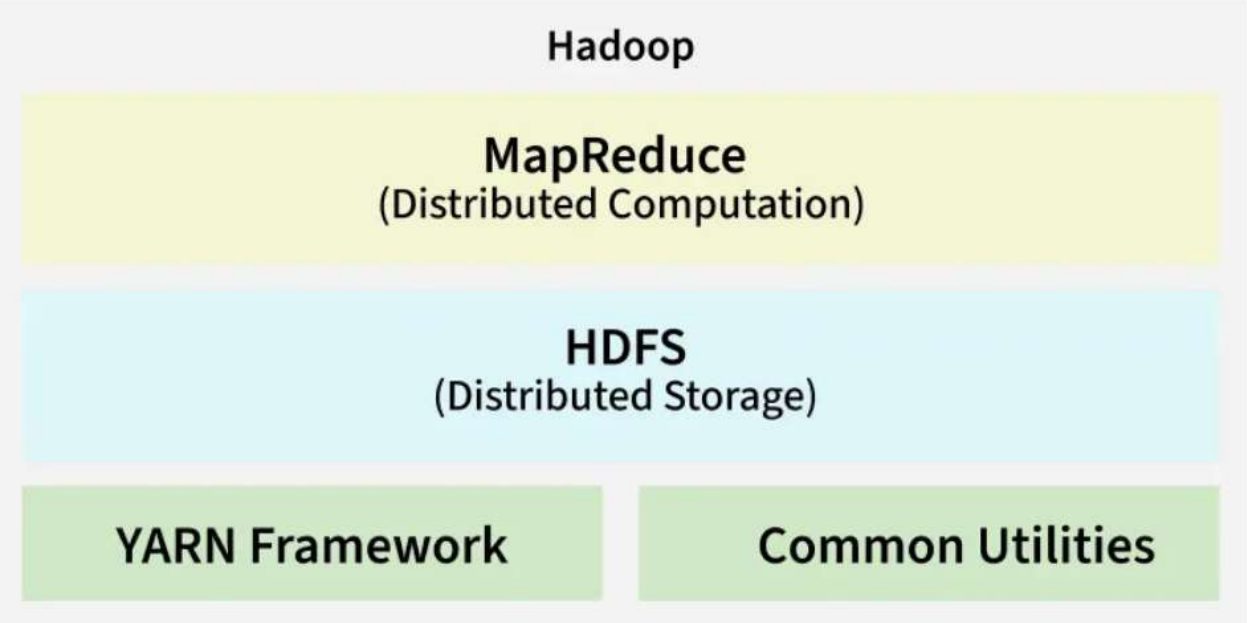
Hadoop is designed to process large volumes of data (Big Data) across many machines without relying on a single machine. It is built to be scalable, fault-tolerant and cost-effective. Instead of relying on expensive high-end hardware, Hadoop works by connecting many inexpensive computers (called nodes) in a cluster

Hadoop Architecture

Hadoop has two main components:

- **Hadoop Distributed File System (HDFS):** HDFS breaks big files into blocks and spreads them across a cluster of machines. This ensures data is replicated, fault-tolerant and easily accessible even if some machines fail.
- **MapReduce:** MapReduce is the computing engine that processes data in a distributed manner. It splits large tasks into smaller chunks (map) and then merges the results (reduce), allowing Hadoop to quickly process massive datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules **Hadoop Common** which are Java libraries and utilities required by other Hadoop modules and **Hadoop YARN** which is a framework for job scheduling and cluster resource management.



Hadoop Distributed File System (HDFS)

HDFS is the storage layer of Hadoop. It breaks large files into smaller blocks (usually 128 MB or 256 MB) and stores them across multiple DataNodes. Each block is replicated (usually 3 times) to ensure fault tolerance so even if a node fails, the data remains available.

Key features of HDFS:

- **Scalability:** Easily add more nodes as data grows.
- **Reliability:** Data is replicated to avoid loss.
- **High Throughput:** Designed for fast data access and transfer.

MapReduce

MapReduce is the computation layer in Hadoop. It works in two main phases:

1. **Map Phase:** Input data is divided into chunks and processed in parallel. Each mapper processes a chunk and produces key-value pairs.
2. **Reduce Phase:** These key-value pairs are then grouped and combined to generate final results.

This model is simple yet powerful, enabling massive parallelism and efficiency.

How Does Hadoop Work?

Here's a overview of how Hadoop operates:

1. *Data is loaded into HDFS, where it's split into blocks and distributed across DataNodes.*
2. *MapReduce jobs are submitted to the ResourceManager.*
3. *The job is divided into map tasks, each working on a block of data.*
4. *Map tasks produce intermediate results, which are shuffled and sorted.*
5. *Reduce tasks aggregate results and generate final output.*
6. *The results are stored back in HDFS or passed to other applications.*

Thanks to this distributed architecture, Hadoop can process petabytes of data efficiently.

Advantages and Disadvantages of Hadoop

Advantages:

- **Scalability:** Easily scale to thousands of machines.
- **Cost-effective:** Uses low-cost hardware to process big data.
- **Fault Tolerance:** Automatic recovery from node failures.
- **High Availability:** Data replication ensures no loss even if nodes fail.
- **Flexibility:** Can handle structured, semi-structured and unstructured data.
- **Open-source and Community-driven:** Constant updates and wide support.

Disadvantages:

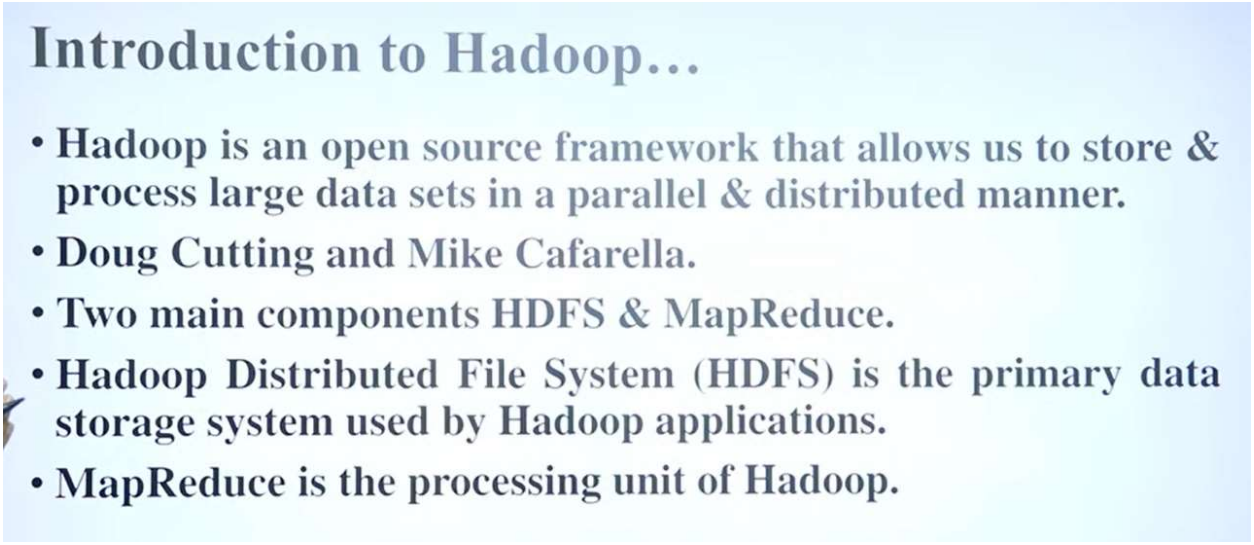
- Not ideal for real-time processing (better suited for batch processing).
- Complexity in programming with MapReduce.
- High latency for certain types of queries.
- Requires skilled professionals to manage and develop.

Applications

Hadoop is used across a variety of industries:

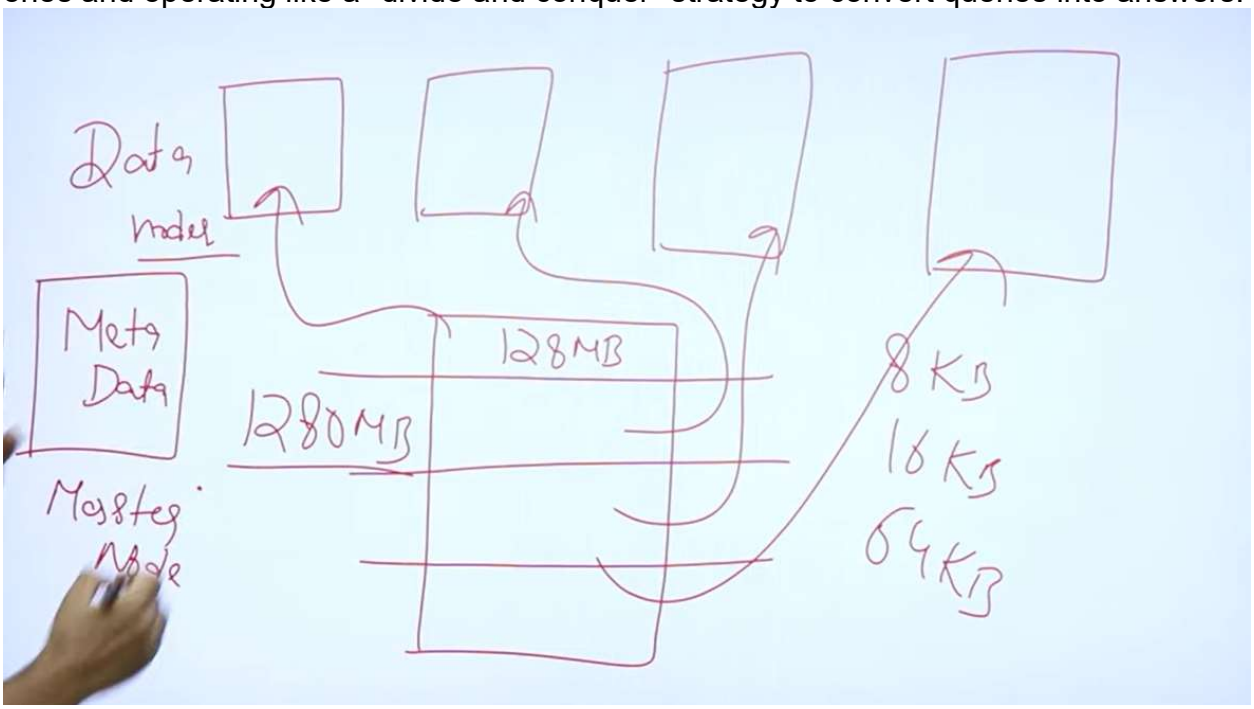
1. **Banking:** Fraud detection, risk modeling.
2. **Retail:** Customer behavior analysis, inventory management.
3. **Healthcare:** Disease prediction, patient record analysis.
4. **Telecom:** Network performance monitoring.
5. **Social Media:** Trend analysis, user recommendation engines.

Introduction to Hadoop🐘| What is Hadoop🐘| Hadoop Framework 📄



- **Origin of Hadoop and Big Data (0:07-2:30)**: The story of Hadoop begins in the early 21st century (around 2001) with the rise of the internet. Before the internet, data was typically structured and limited in volume, allowing for storage on single devices and processing by single elements. However, with the internet's popularity, the type of data rapidly changed to include text, images, and videos, leading to an explosion in data volume, commonly referred to as **Big Data**.
- **Development of Hadoop (2:31-3:17)**: Around 2002, **Doug Cutting** (often called the Father of Big Data or Father of Hadoop) and Mike started working on a project named Hadoop. Their goal was to find a way to easily store and process this massive amount of Big Data. In 2008, Yahoo declared Hadoop an open-source project, and by 2012, Apache made it publicly available, leading to its popularity as **Apache Hadoop**.
- **What is Hadoop? (3:18-4:20)**: Hadoop is an **open-source framework**, not just a software or a simple database. It's a complete framework designed to properly store and access data in a **distributed manner**. This means instead of storing all data in a single location, Hadoop distributes it across different areas or clusters, allowing for parallel access. Initially written in Java, it supports programming in C++, Python, and Java.

- **Core Components of Hadoop (4:23-9:11)**: The two most fundamental components of the Hadoop framework are:
- **HDFS (Hadoop Distributed File System)** (4:33-8:18): Similar to file systems in Windows or Linux, HDFS is specifically designed for Big Data. It avoids storing large files in a single location. Instead, it **divides** a file (e.g., a 1280 MB file) into smaller blocks, typically 128 MB by default, and stores these partitions in different locations called **Data Nodes**. This distributed storage uses existing, cost-effective commodity hardware. A **Master Node** stores metadata (6:31), which provides information about where each data block is located, its size, and other characteristics. To prevent data loss in case of a node failure, HDFS uses **replication (or duplication)** (7:19-8:18), creating multiple copies (typically 2-3) of each data block on different data nodes.
- **MapReduce** (8:27-9:11): This is the **processing element** of Hadoop, acting like its CPU. Its main function is to process user queries by accessing data in parallel. MapReduce works on a **key-value pair** concept, reducing large problems into smaller ones and operating like a "divide and conquer" strategy to convert queries into answers.

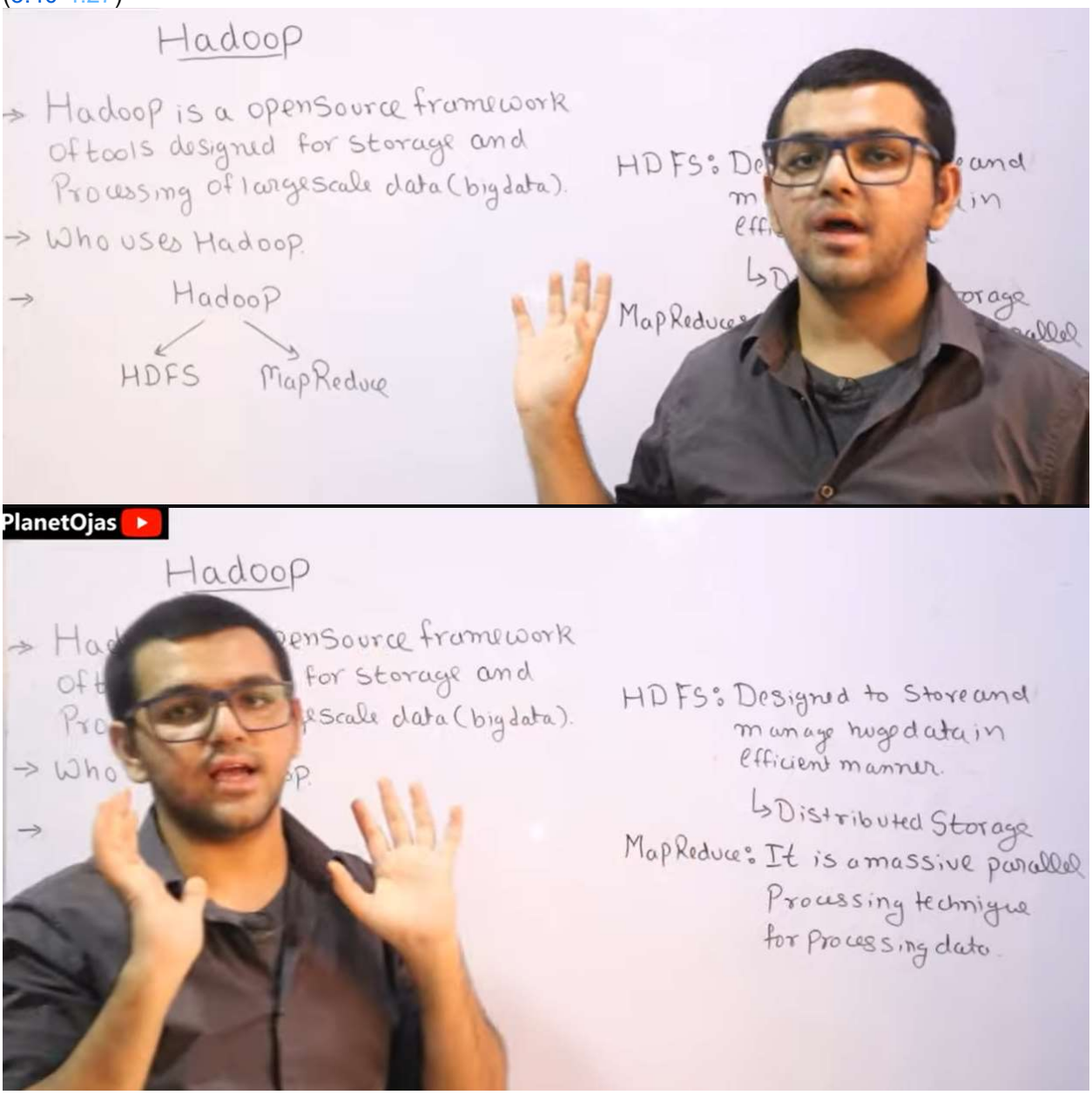


Introduction to Hadoop

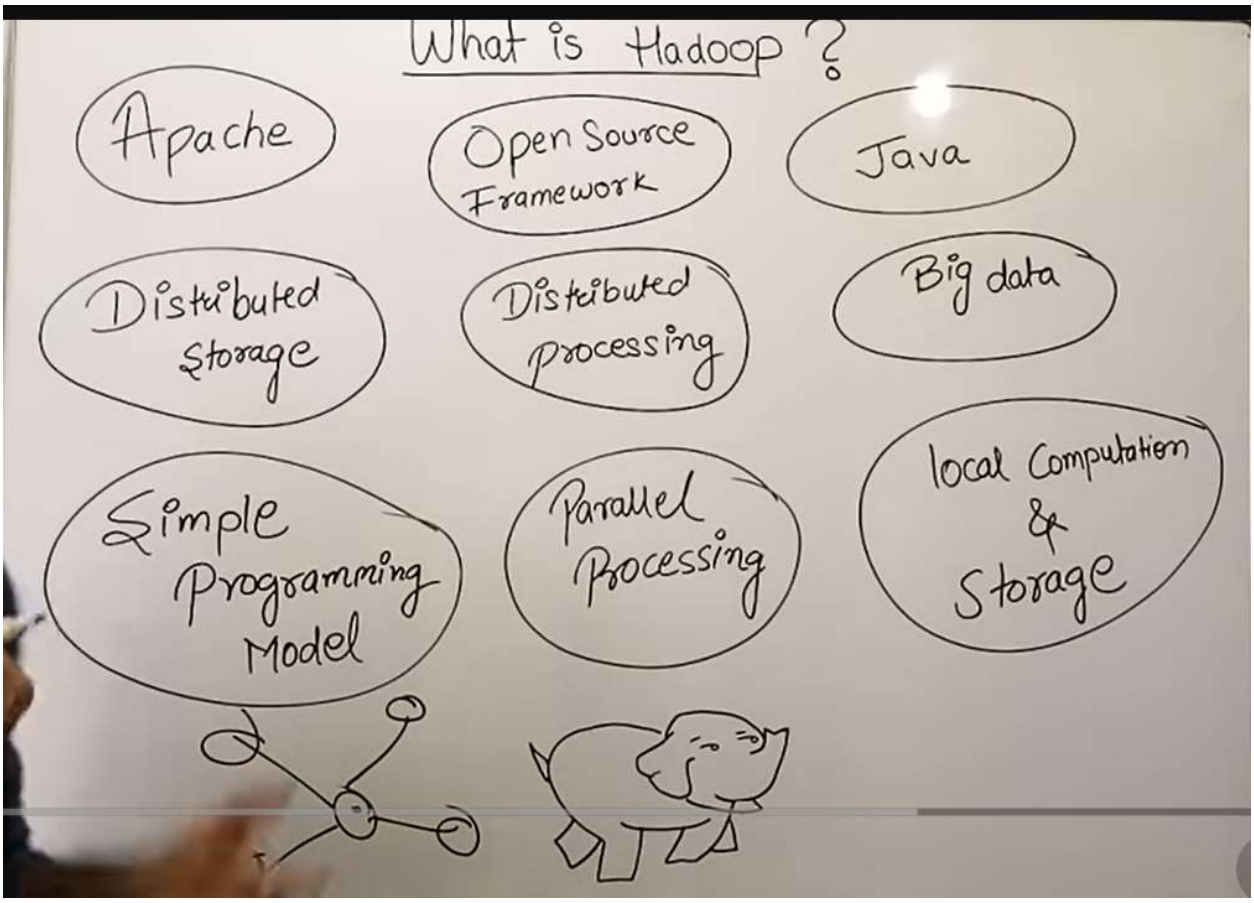
The video covers:

- **What is Hadoop:** Hadoop is an open-source framework designed for processing and storing big data. It's free to use and was developed by Doug Cutting and Mike Cafarella (0:00-0:18). It was named after Doug Cutting's son's toy elephant (0:45-0:50).
- **Hadoop Distributed File System (HDFS):** HDFS is designed to store and manage large amounts of data efficiently. It addresses the challenges of storing massive datasets on a single system by distributing data across multiple machines (1:25-2:45).
- **MapReduce:** MapReduce is a programming model used within Hadoop for processing large datasets in parallel. It breaks down large data processing tasks into smaller sub-tasks that can be executed concurrently across different systems (2:50-3:27).

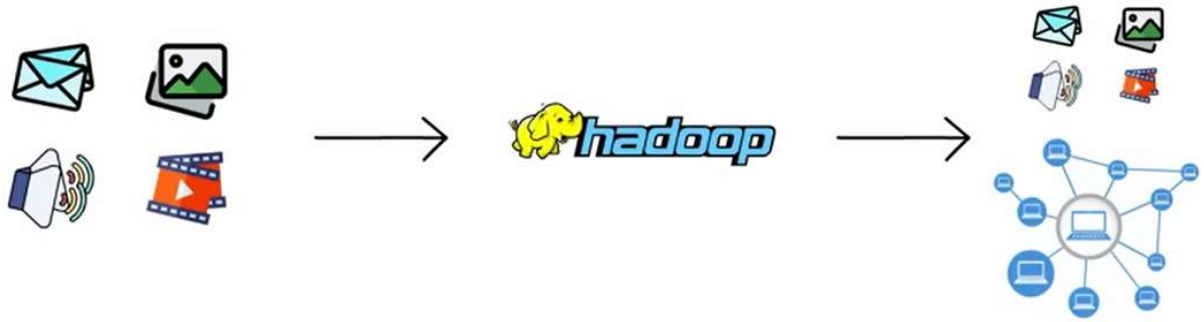
- **Why Use Hadoop:** Hadoop is essential because traditional database systems struggle to handle the immense volume and velocity of modern big data. It overcomes the limitations of older systems by providing a robust and scalable solution for storing and processing continuously generated data (3:40-4:27)



Hadoop II Introduction to Hadoop II Features Explained with Examples

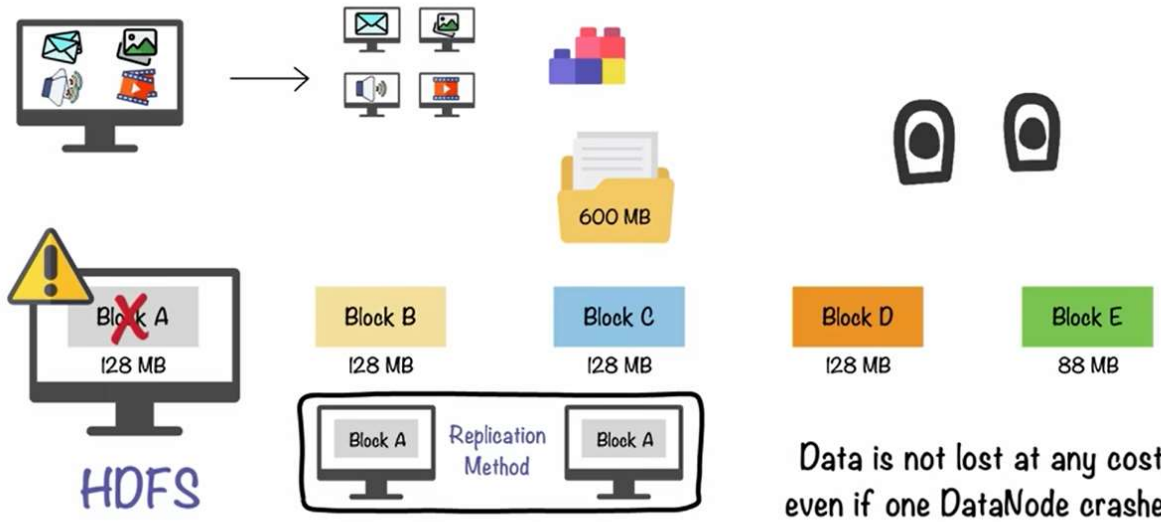


What Is Hadoop? | Introduction To Hadoop | Hadoop Explained |Simplilearn



Hadoop consisted of three components that were specifically designed to work on big data

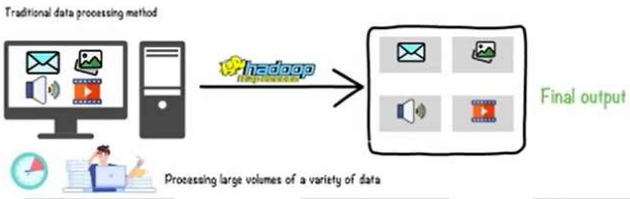
1. Storage unit → HDFS

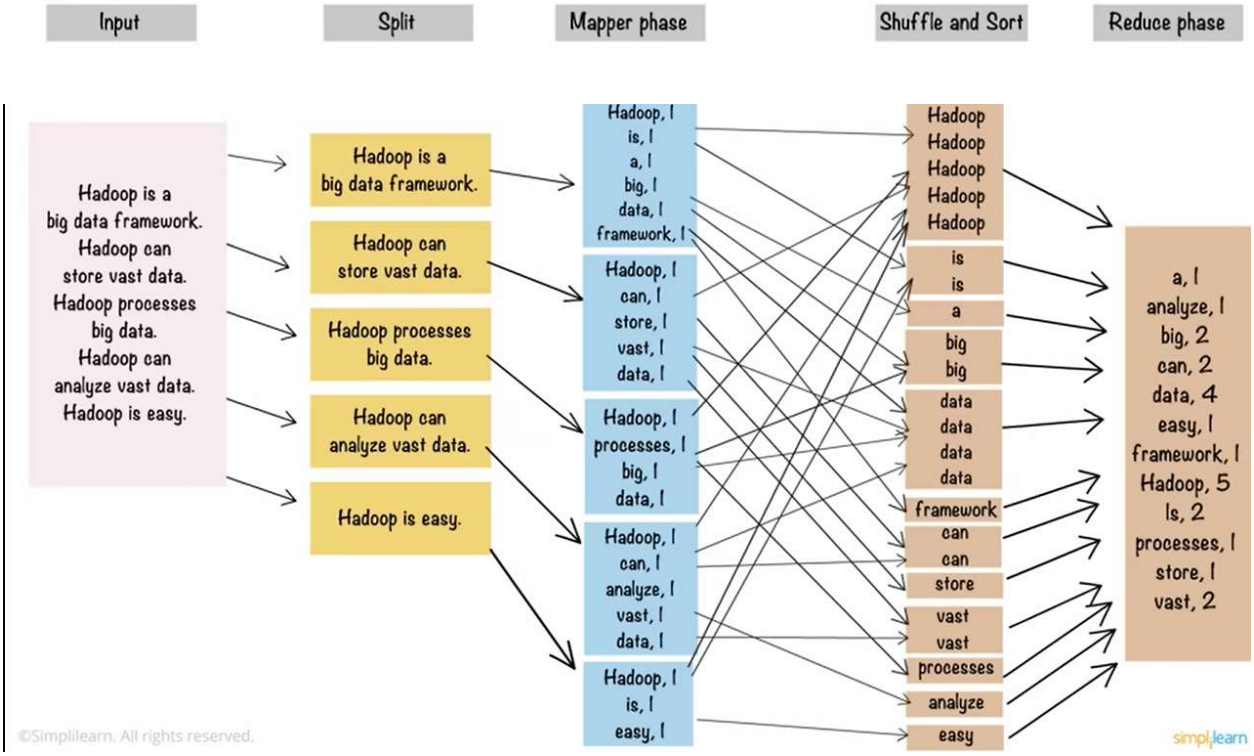


HDFS makes copies of the data and stores it across multiple systems

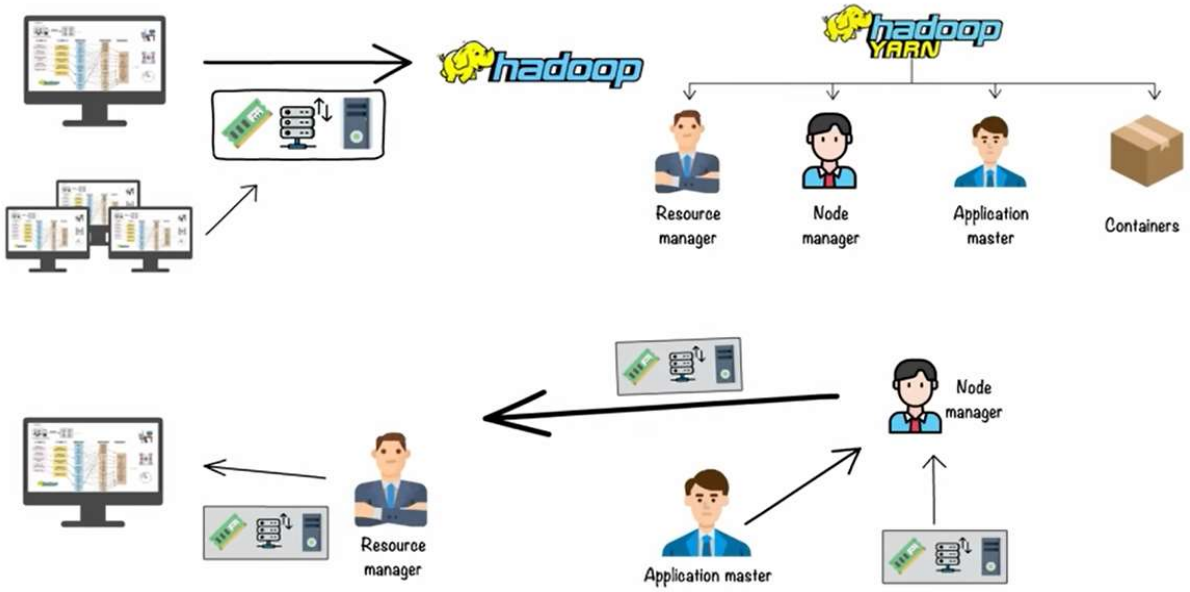
Data is not lost at any cost; even if one DataNode crashes, making HDFS fault-tolerant

2. MapReduce

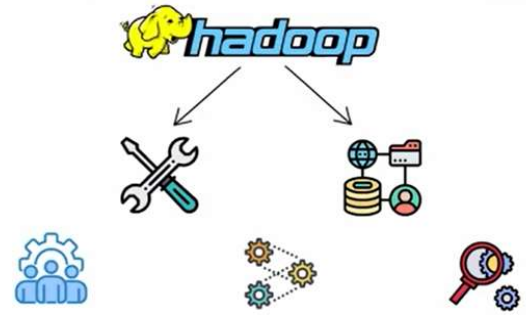




3. YARN



YARN processes job requests and manages cluster resources



The Hadoop ecosystem comprises several other components like

The Hadoop ecosystem works together on big data management

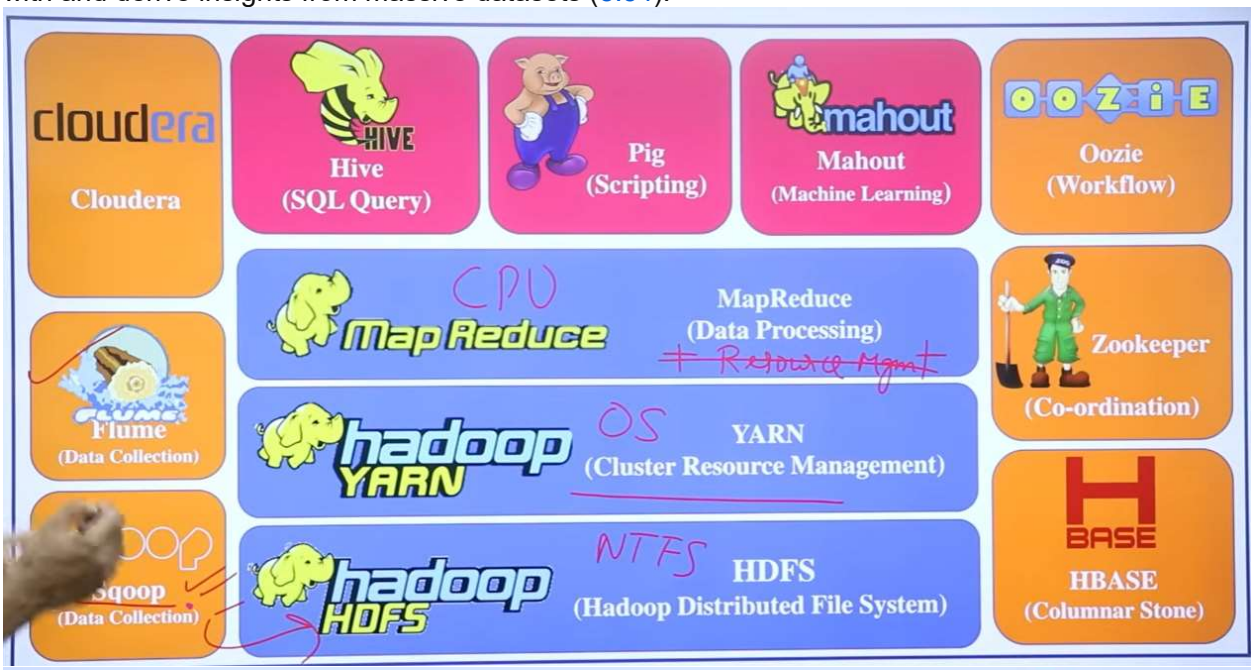
Hadoop🐘Ecosystem | All Components Hdfs🐘,Mapreduce,Hive🐝,Flume,Sqoop,Yarn,Hbase,Zookeeper,Pig🐻

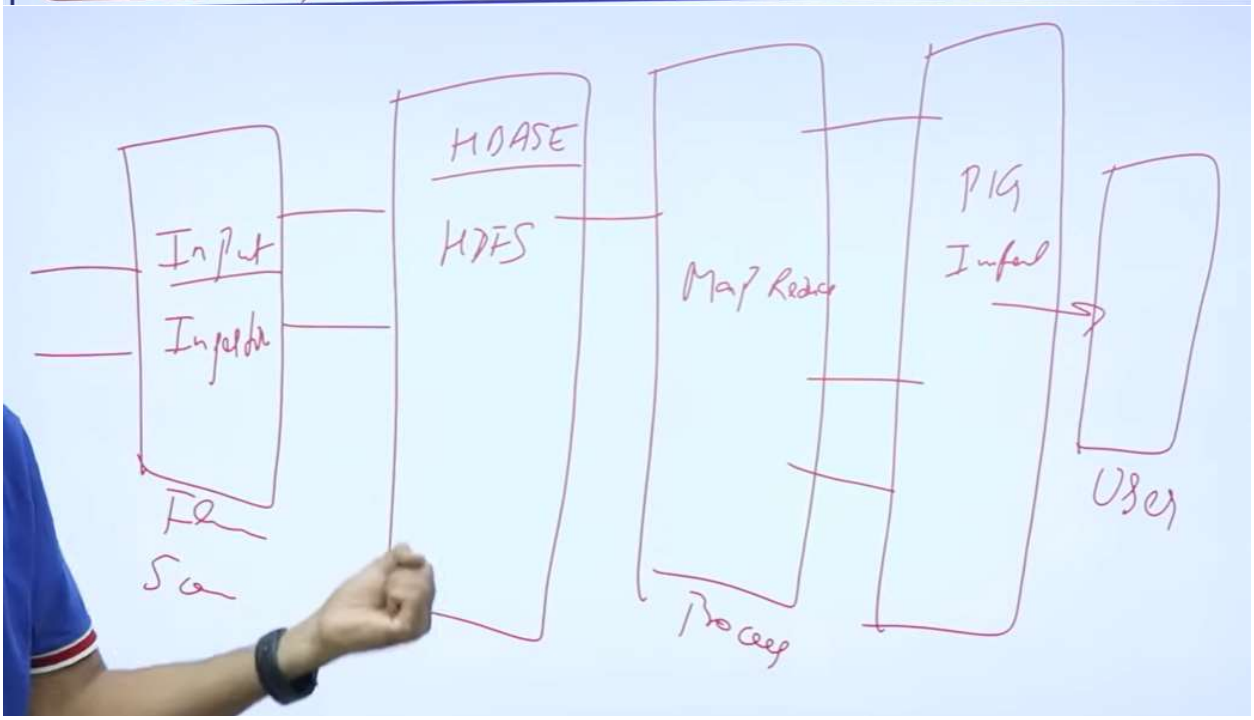
The video explains the various components of the **Hadoop ecosystem** (0:04) and how they work together to manage, process, and analyze large datasets in a distributed environment.

Here's a summary of the key components and their roles:

- **HDFS (Hadoop Distributed File System)** (0:47): This is a mandatory component responsible for storing and managing data in a distributed manner, allowing for easy access and processing. It acts as the file system for Hadoop, similar to NTFS in Windows or NFS in Linux (1:00-1:18).
- **YARN (Yet Another Resource Negotiator)** (1:26): YARN manages resources and schedules jobs within the Hadoop cluster. Initially, resource management was part of MapReduce, but due to increasing data and slowing performance, YARN was separated to handle resource allocation and job scheduling efficiently (1:55-2:40).
- **MapReduce** (1:37): This component is used for processing data in a distributed and parallel fashion. It processes data stored in HDFS and converts it into output (1:41-1:50).
- **Flume and Sqoop** (3:16): These tools are used for data ingestion or collection.
- **Sqoop** (3:52): Used for structured data, specifically for importing and exporting data between Hadoop and relational databases like MySQL, SQL Server, or Oracle (3:52-4:18).
- **Flume** (3:56): Used for unstructured or semi-structured data, especially for collecting streaming data, live streams, photos, videos, and log files in real-time (4:24-4:43).
- **HBase** (4:46): A NoSQL, column-oriented database that allows for storing large amounts of data without predefined schemas, unlike traditional relational databases (5:05-5:21).
- **Hive** (6:04): Used for SQL-like querying of structured data within Hadoop, allowing users familiar with SQL to interact with Hadoop data without extensive Java coding (6:04-6:33).
- **Pig** (6:35): A scripting platform that simplifies complex MapReduce programming. It significantly reduces the lines of code required for data processing tasks, making development faster (6:40-7:02).
- **Mahout** (7:05): A machine learning library that provides various algorithms for building machine learning applications on top of Hadoop (7:07-7:42).
- **Cloudera** (9:16): A platform that provides a user-friendly interface for managing, monitoring, and exploring data within the Hadoop ecosystem (9:16-9:24).

- **Oozie** (9:31): A workflow scheduler that helps in automating and orchestrating jobs within the Hadoop system, scheduling tasks, and managing their execution times (9:34-9:53).
- **ZooKeeper** (9:54): Provides centralized coordination services for large Hadoop clusters, managing and monitoring different components across a very large scale, essentially acting as a "zoo keeper" for all the components (9:56-10:55).
  The video emphasizes that the Hadoop ecosystem facilitates **data ingestion** (7:50), **data storage** (8:02), **data processing** (8:22), and **data analysis/visualization** (8:41) to enable users to interact with and derive insights from massive datasets (8:54).



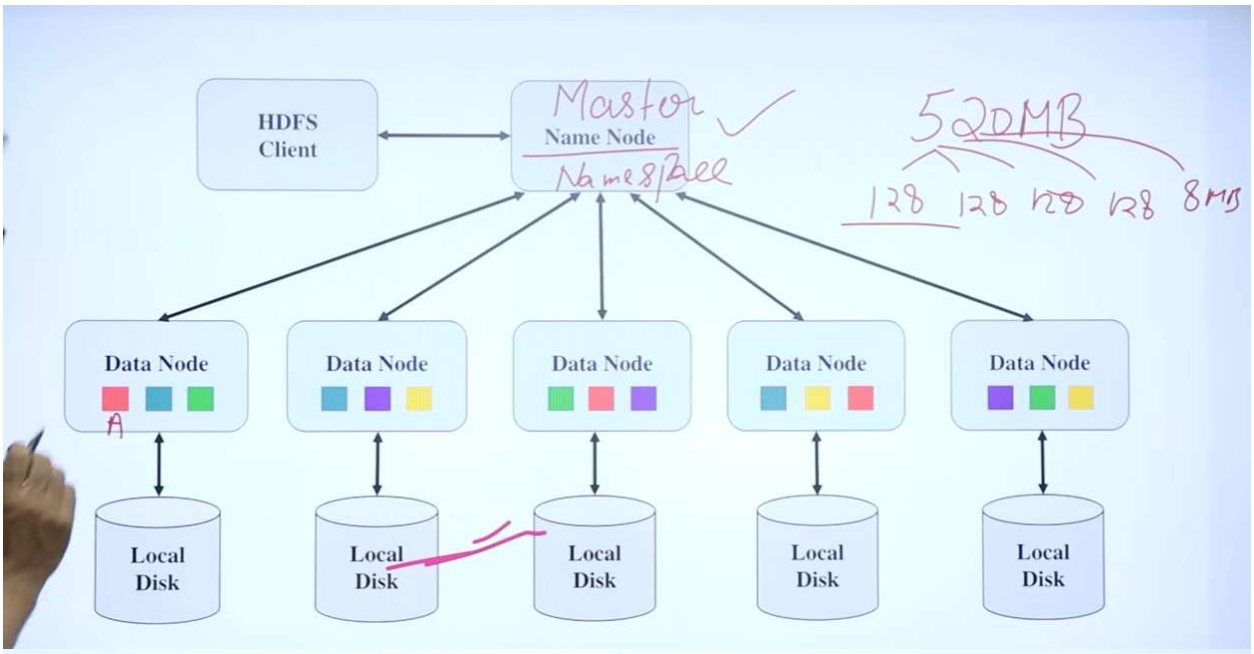


What is HDFS | Name Node vs Data Node | Replication factor | Rack Awareness | Hadoop Framework

Hadoop Distributed File System (HDFS) is a core component of Apache Hadoop, designed for storing and managing large datasets for big data analytics (0:30-1:13). It addresses the challenge of storing massive amounts of data efficiently, while MapReduce handles the processing (1:03-1:10). HDFS is a robust, distributed file system built upon traditional hierarchical file system architecture, but scaled for huge datasets (1:13-1:48).

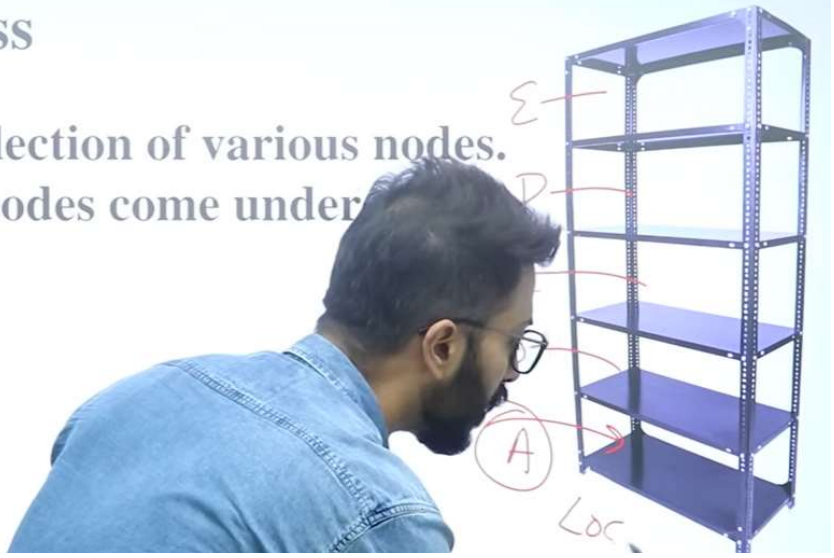
Key components and concepts of HDFS include:

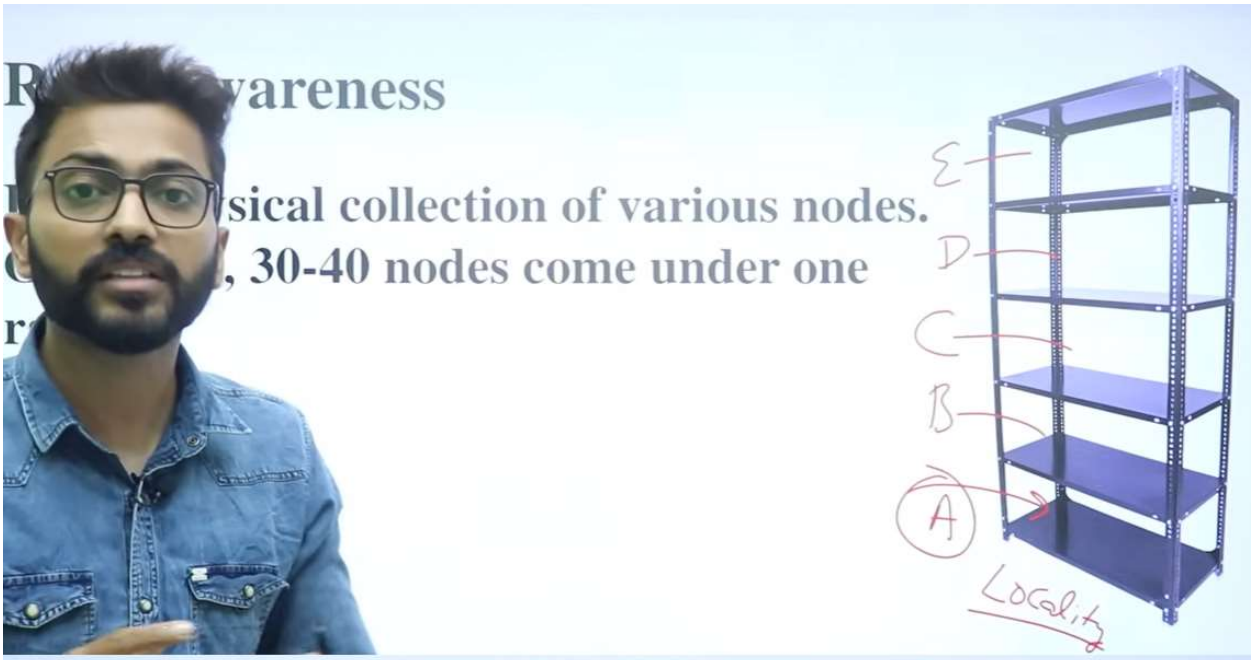
- **NameNode:** This acts as the "boss" or "master node," responsible for managing the file system namespace and storing metadata (e.g., file names, permissions, block locations, replication factors) (2:09-2:21, 5:07-5:56).
- **DataNodes:** These are the "employees" that store the actual data in physical blocks (2:24-2:47). HDFS works with commodity hardware (3:00-3:07).
- **Data Storage and Block Size:** HDFS divides large files into smaller blocks, with a default size of 128 MB (3:19-4:11). These blocks are distributed across various DataNodes for parallel access and processing (4:39-5:07).
- **Replication Factor:** To ensure fault tolerance and high availability, HDFS replicates each data block multiple times, with a default replication factor of three (5:58-6:46). This means if one DataNode goes down, the data can still be accessed from its replicas (6:49-7:23).
- **Rack Awareness:** HDFS intelligently places replicas across different racks to mitigate data loss if an entire rack fails (7:26-8:33). It also prioritizes placing one copy locally to enable faster access for read operations (8:35-9:50).
- **NameNode Functions:** The NameNode maintains an **FSImage**, which is a snapshot of the file system's metadata, and an **EditLog**, which records all changes made to the file system (9:53-10:33). A **Secondary NameNode** assists by periodically merging the EditLog into the FSImage to keep it updated, acting as a helper to the NameNode (10:43-11:52).
- **Heartbeat Messages:** DataNodes regularly send "heartbeat" messages to the NameNode (every 3 seconds) to indicate they are alive and functioning (11:54-12:17).
- **Read and Write Operations:** When reading data, the client interacts with the NameNode to get the block locations, then directly accesses the DataNodes (12:34-13:01). Write operations are more expensive as changes need to be synchronized across all replicas simultaneously (13:03-13:38



Rack Awareness

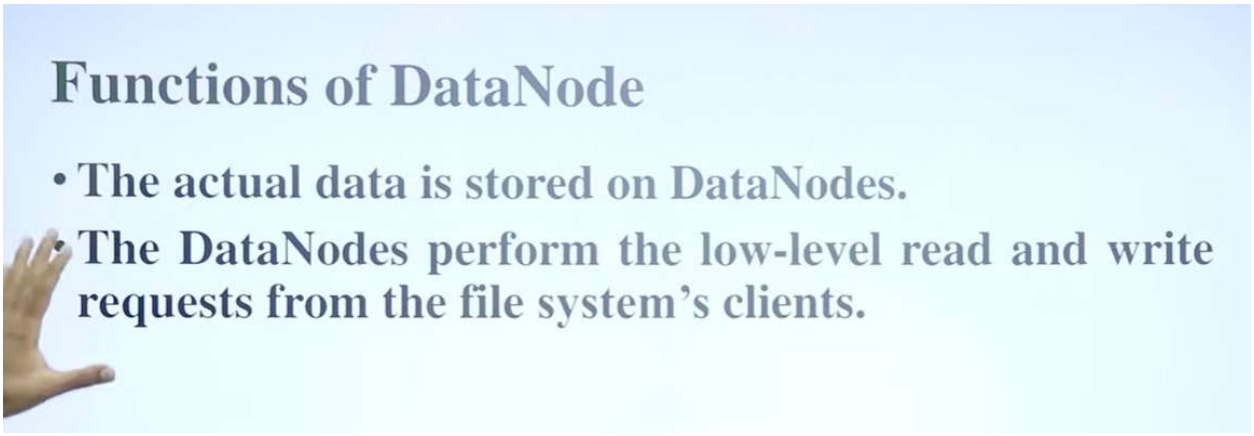
It is a physical collection of various nodes. Generally, 30-40 nodes come under one rack.



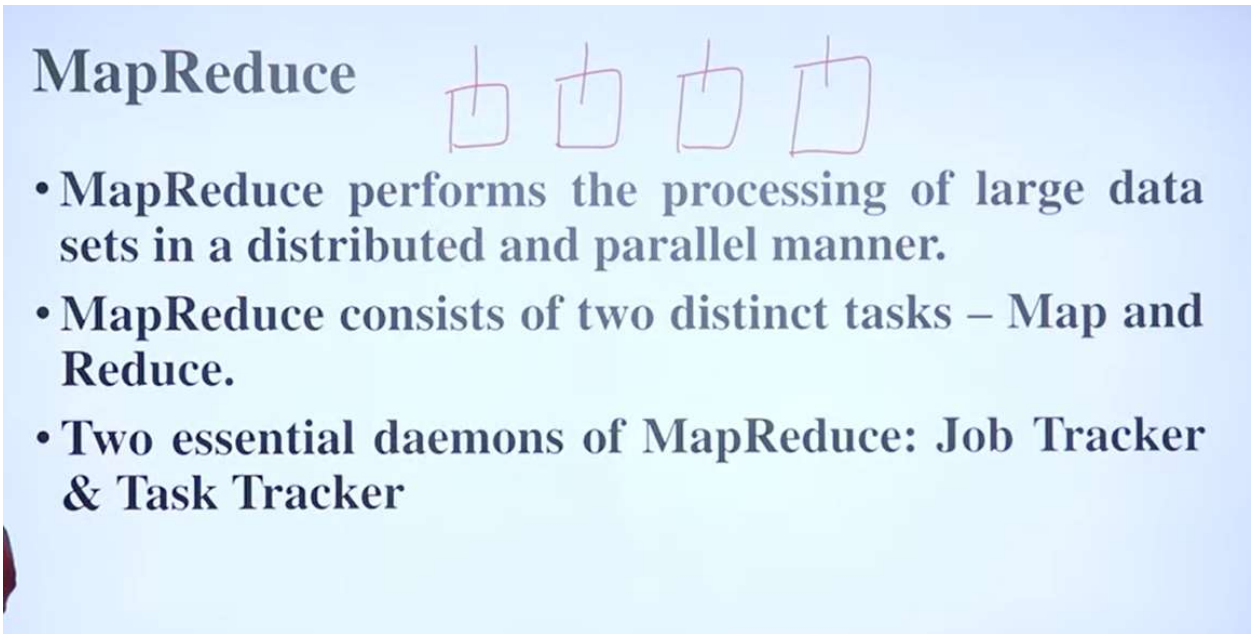


Functions of NameNode

- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
- FsImage: It contains the complete state of the file system namespace since the start of the NameNode.
- EditLogs: It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.



What is MapReduce♻️in Hadoop🐘| Apache Hadoop🐘

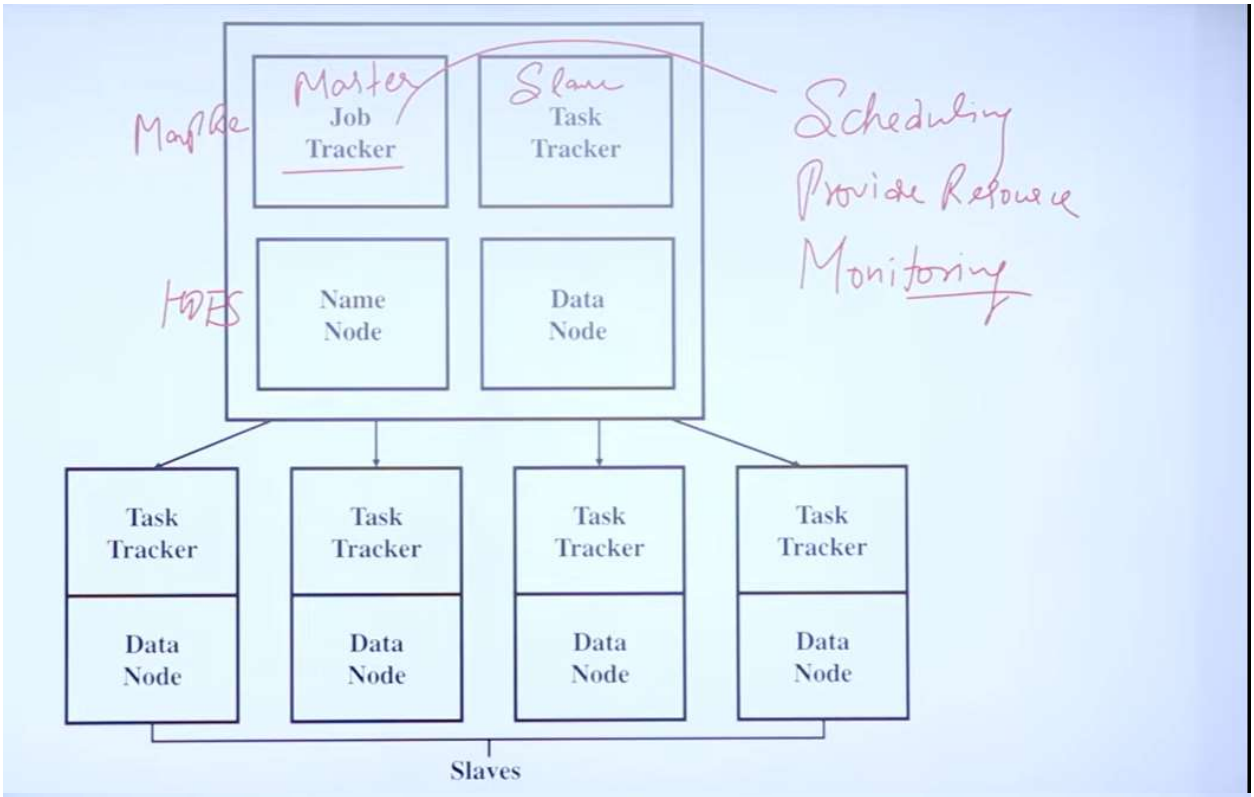


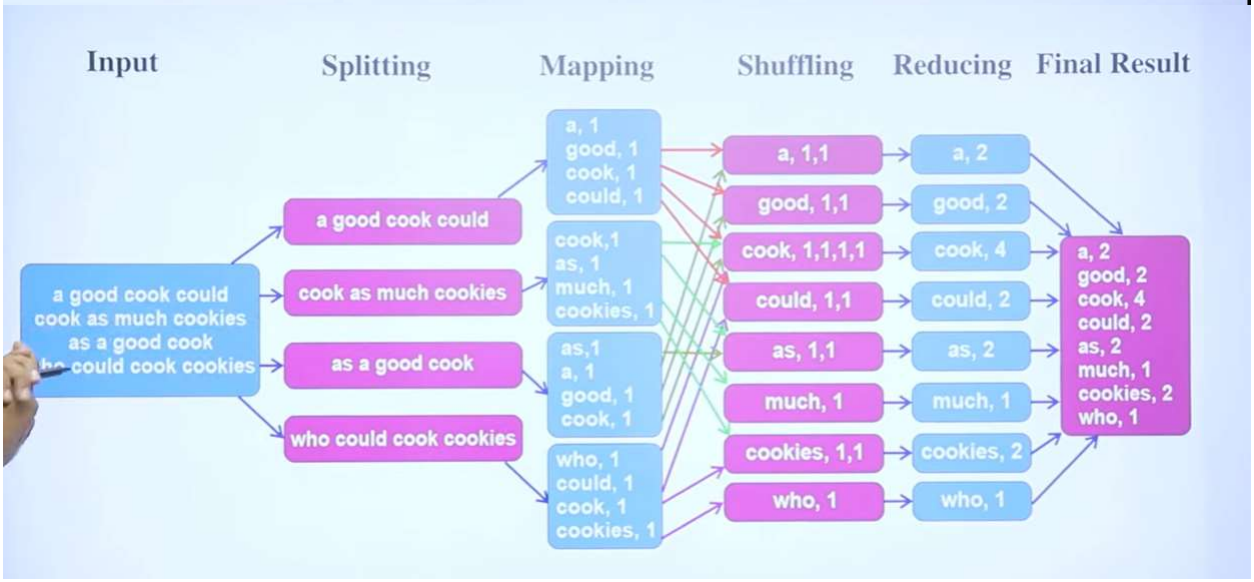
Key concepts discussed include:

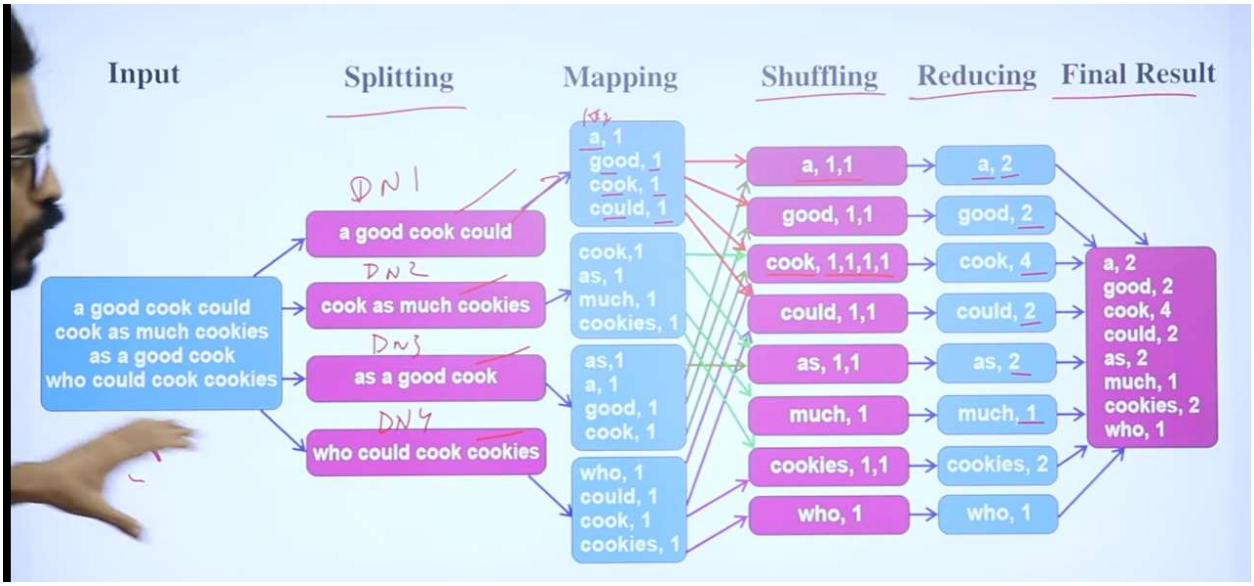
- **Purpose of MapReduce:** It acts as the processing unit for big data stored in HDFS (Hadoop Distributed File System), enabling parallel processing to improve efficiency and reduce processing time (1:00-1:19).
- **Map and Reduce Tasks:** MapReduce operates on a "divide and conquer" approach, splitting large problems into smaller ones.
- **Map Task (Mapper Class):** Breaks down the large input into smaller pieces, often converting data into key-value pairs (1:41-2:26).
- **Reduce Task (Reducer Class):** Combines the intermediate results generated by the Map tasks to produce a final output (2:51-2:58).
- **MapReduce Daemons:** Two main background processes manage the MapReduce operations.
- **Job Tracker:** Acts as the master node or "boss," responsible for scheduling processes, providing resources, and monitoring the overall job execution (3:11-3:46).
- **Task Tracker:** The "employee" node that performs the actual work, executing tasks on the data nodes where the data is stored (3:32-3:48).

- **Data Locality:** A crucial concept in MapReduce is that processing happens where the data resides (on local disks of data nodes), rather than moving data to the processing unit. This minimizes network delay and improves efficiency (5:00-6:52).
- **Word Count Example:** The video illustrates the MapReduce workflow with a word count example (6:57).

1. **Splitting:** The input paragraph is divided into smaller chunks and distributed across data nodes (7:26-7:46).
2. **Mapping:** Each chunk is processed in parallel to generate key-value pairs (word, count) (7:49-8:26).
3. **Shuffling/Grouping:** Similar keys (words) from different mappers are grouped together (8:30-9:14).
4. **Reducing:** The grouped key-value pairs are processed to get the final count for each word (9:16-9:27).
5. **Final Output:** The combined results provide the total count for each word (9:33-9:43).


Master
Job Tracker
Slave
Task Tracker
Scheduling
Provide Resource
Monitoring
HDFS
Name Node
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Slaves


Input
Splitting
Mapping
Shuffling
Reducing
Final Result
a good cook could cook as much cookies as a good cook who could cook cookies
a good cook could
cook as much cookies
as a good cook
who could cook cookies
a, 1
good, 1
cook, 1
could, 1
cook,1
as, 1
much, 1
cookies, 1
as,1
a, 1
good, 1
cook, 1
who, 1
could, 1
cook, 1
cookies, 1
a, 1,1
good, 1,1
cook, 1,1,1,1
could, 1,1
as, 1,1
much, 1
cookies, 1,1
who, 1
a, 2
good, 2
cook, 4
could, 2
as, 2
much, 1
cookies, 2
who, 1
a, 2
good, 2
cook, 4
could, 2
as, 2
much, 1
cookies, 2
who, 1

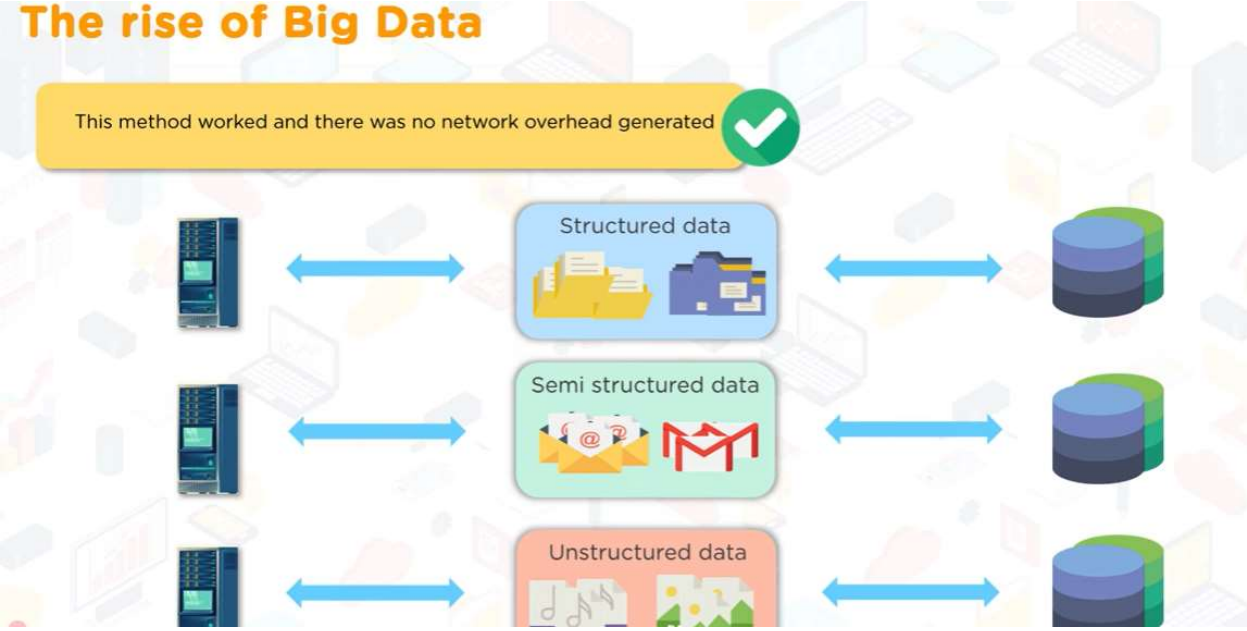


What Is Hadoop? | Introduction To Hadoop

Hadoop as a solution for managing and processing **Big Data**.
The video begins by illustrating the challenges of traditional data processing systems with an analogy of a farm (0:26) and then moves on to explain the rise of **Big Data** (2:30), defining it by its **five V's**: **Volume**, **Velocity**, **Variety**, **Value**, and **Veracity** (6:31). It highlights the challenges of processing large volumes of diverse data with single storage and serial processing methods (9:40).
Hadoop is presented as a framework designed to manage big data storage in a distributed way and process it in parallel (11:17). The core components of Hadoop include:

- **Hadoop Distributed File System (HDFS)** (12:20): This is Hadoop's storage unit, designed for storing huge datasets on commodity hardware. It uses a **NameNode** (master) to manage metadata and **DataNodes** (slaves) to store the actual data and perform replication (13:17). Data is distributed and replicated (default three times) across these nodes, ensuring fault tolerance and data security (16:05).
- **Hadoop MapReduce** (17:10): This is the processing unit of Hadoop, a programming technique for parallel processing of huge data stored in HDFS. It involves a **Map phase** where data chunks are processed independently, followed by a **Shuffle and Sort phase** to group data, and finally a **Reduce phase** where aggregated results are obtained (19:22).
- **Hadoop YARN** (Yet Another Resource Negotiator) (22:55): This acts as the resource management unit or "operating system" for Hadoop. It's responsible for managing cluster resources and job scheduling across the different nodes (23:30).
  A practical **use case of Hadoop** is demonstrated through a scenario of combating fraudulent activities in banking (25:16). Hadoop enabled a bank to store massive amounts of structured and unstructured data, process server logs, customer data, and transactions efficiently, leading to in-depth analysis and improved detection of fraudulent activities like malware and phishing attempts (27:23).

The video concludes by reiterating these key takeaways regarding Big Data, Hadoop's components (HDFS, MapReduce, YARN), and its real-world application in fraud detection (28:50).

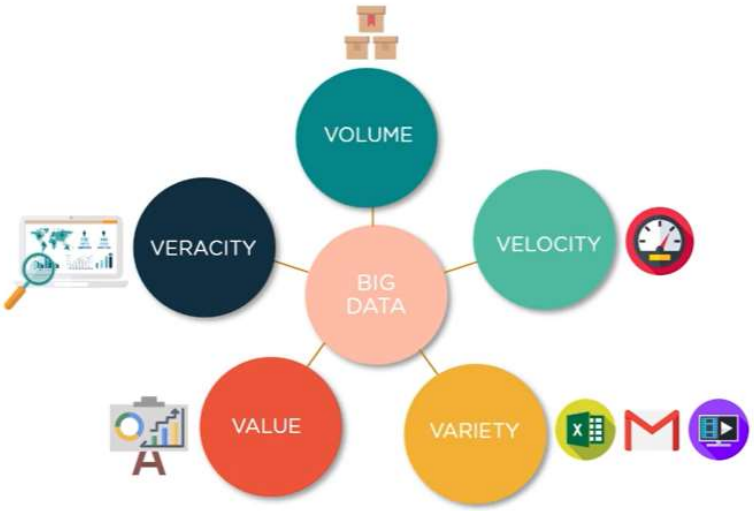


What's in it for you?

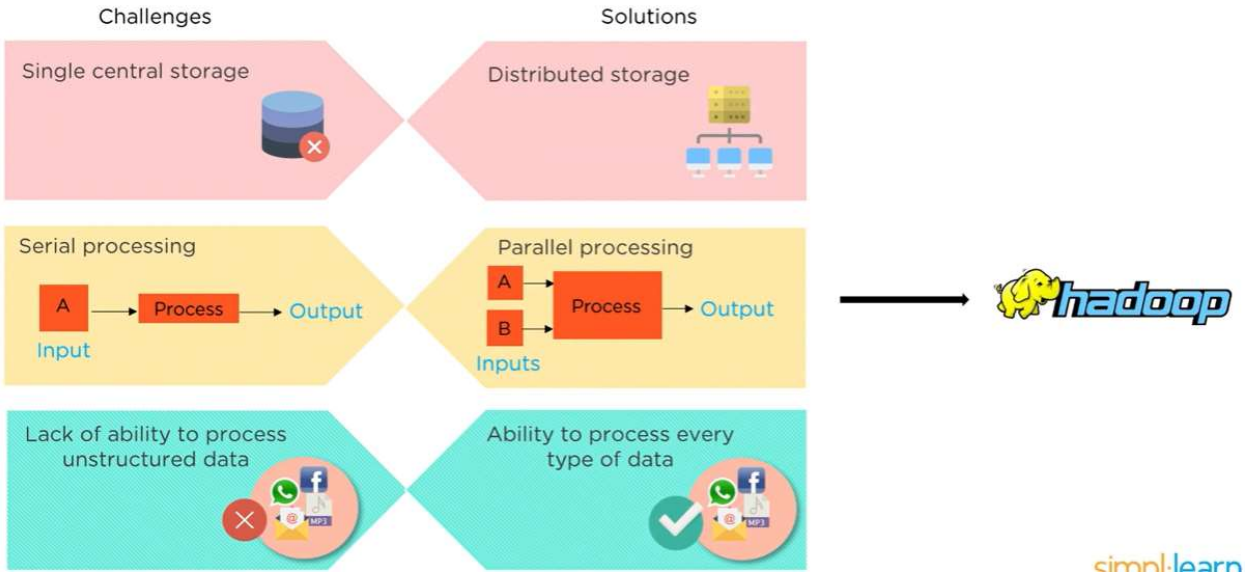
1. Big Data and it's challenges
2. Hadoop as a solution
3. What is Hadoop?
4. Components of Hadoop
5. Use case of Hadoop

What is Big Data?

Massive amount of data which cannot be stored, processed and analyzed using the traditional ways



Hadoop as a solution



simplilearn

What is Hadoop?

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

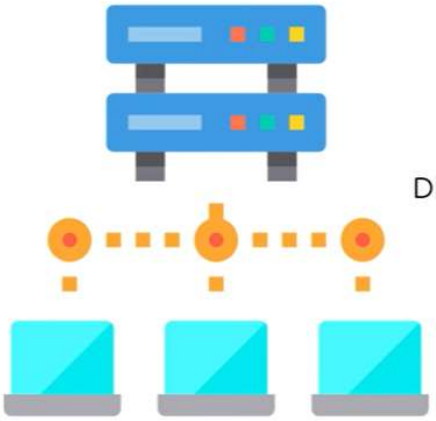
Components of Hadoop

Storage unit of Hadoop

Processing unit of Hadoop

What is HDFS?

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



Distributed storage

What is HDFS?

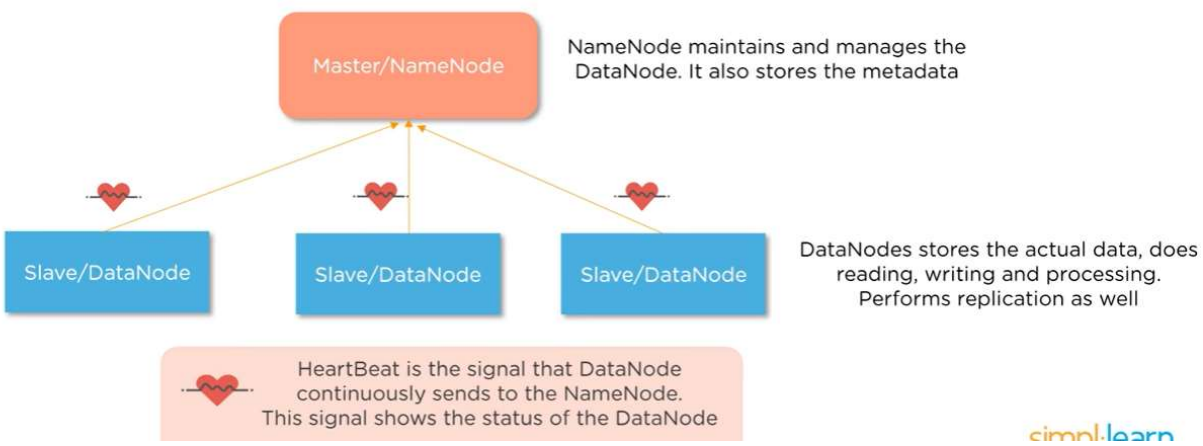
Hadoop Distributed File System (HDFS) has two core components NameNode and DataNode

NameNode

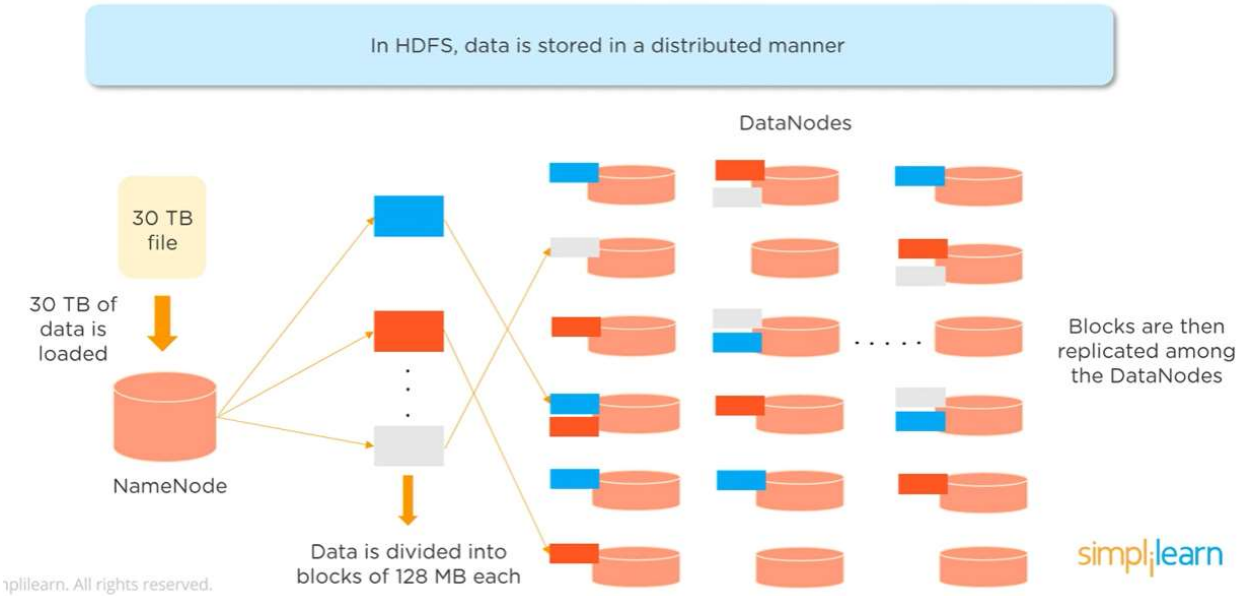
DataNode

What is HDFS?

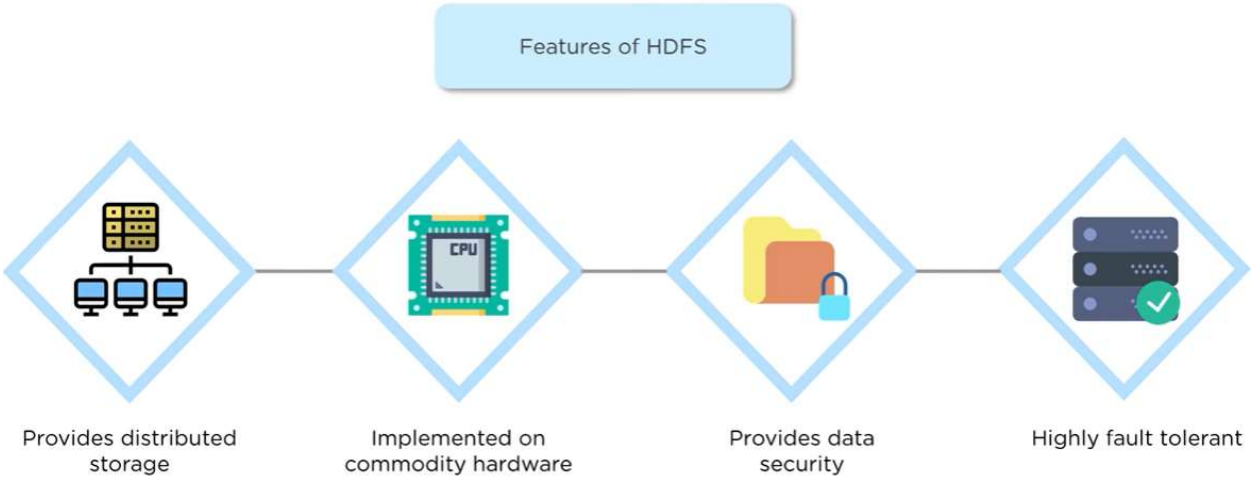
Master/slave nodes typically form the HDFS cluster



What is HDFS?



What is HDFS?



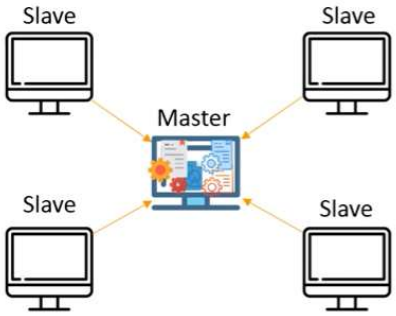
What is MapReduce?

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

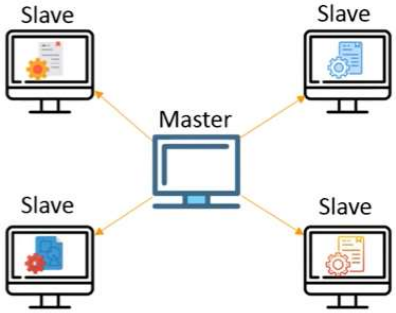
MapReduce is used for parallel processing of the Big Data, which is stored in HDFS

What is MapReduce?

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

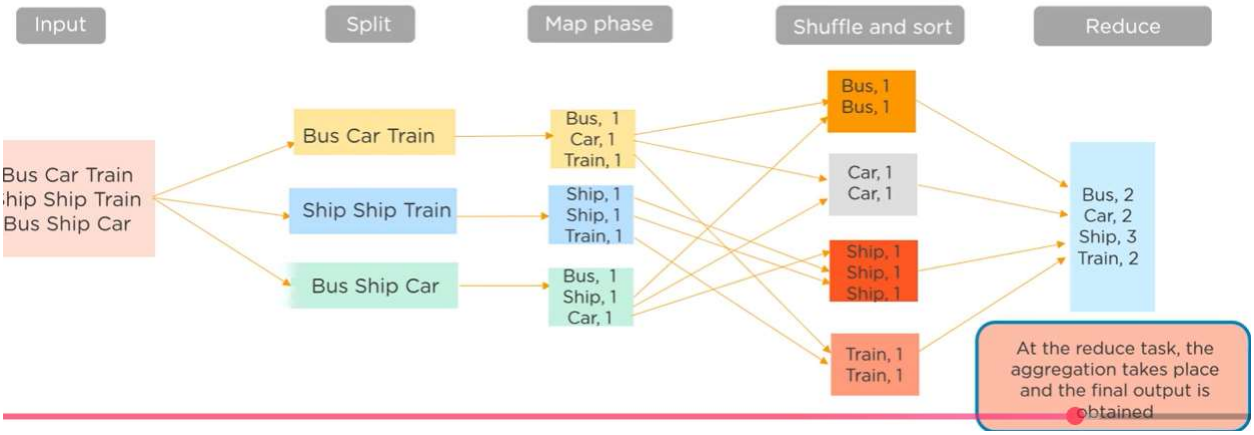


Traditional approach - Data is processed at the Master node

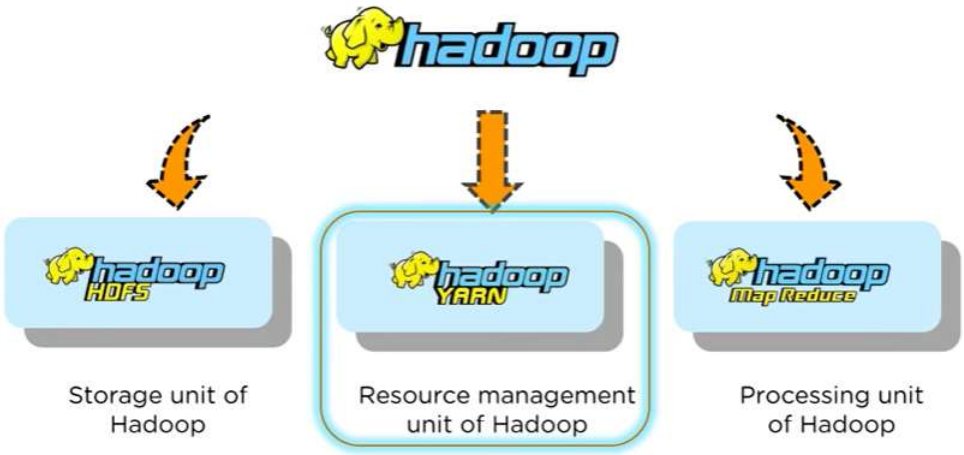


MapReduce approach - Data is processed at the Slave nodes

What is MapReduce?



Components of Hadoop version 2.0



What is YARN?

YARN – Yet Another Resource Negotiator

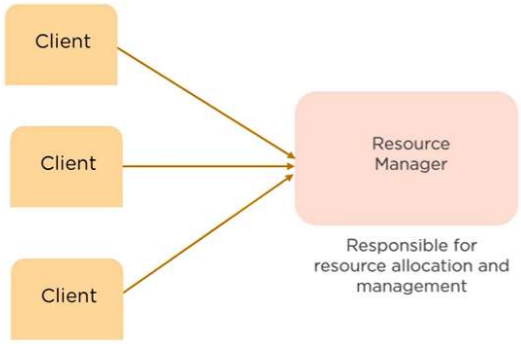
Acts like an OS to Hadoop 2

Responsible for managing cluster resources

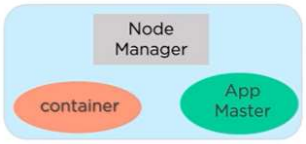
Does job scheduling

What is YARN?

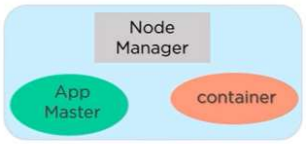
Client submits the job request



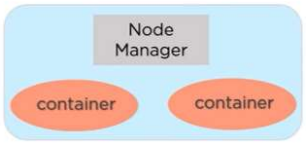
Responsible for resource allocation and management



Node Manager manages the nodes and monitors resource usage



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

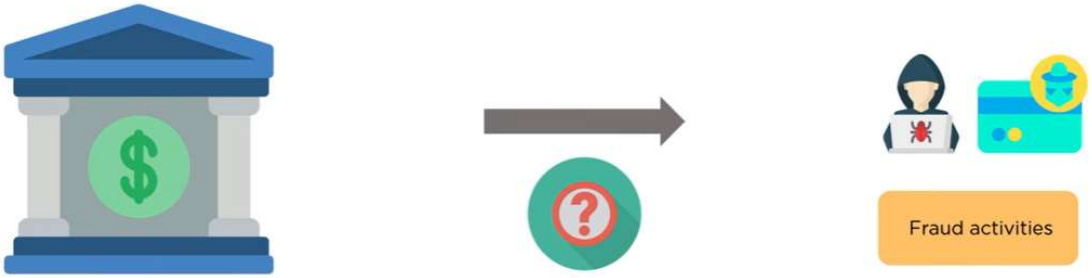


App Master requests container from the NodeManager

simplilearn

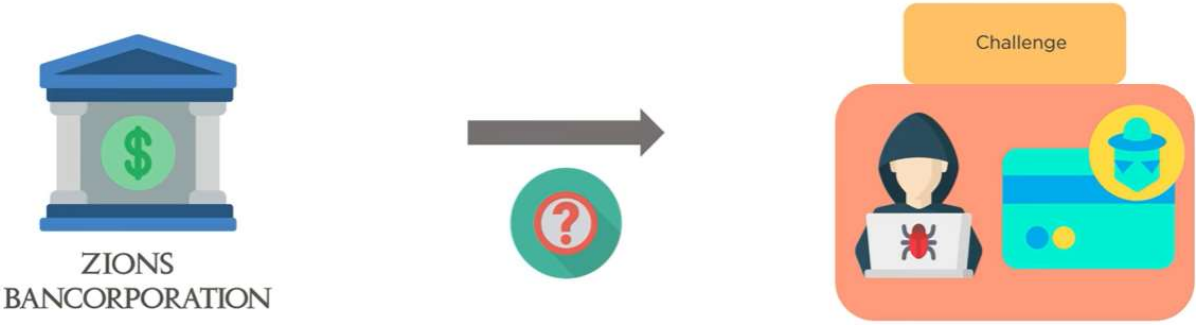
Hadoop use case – Combating fraudulent activities

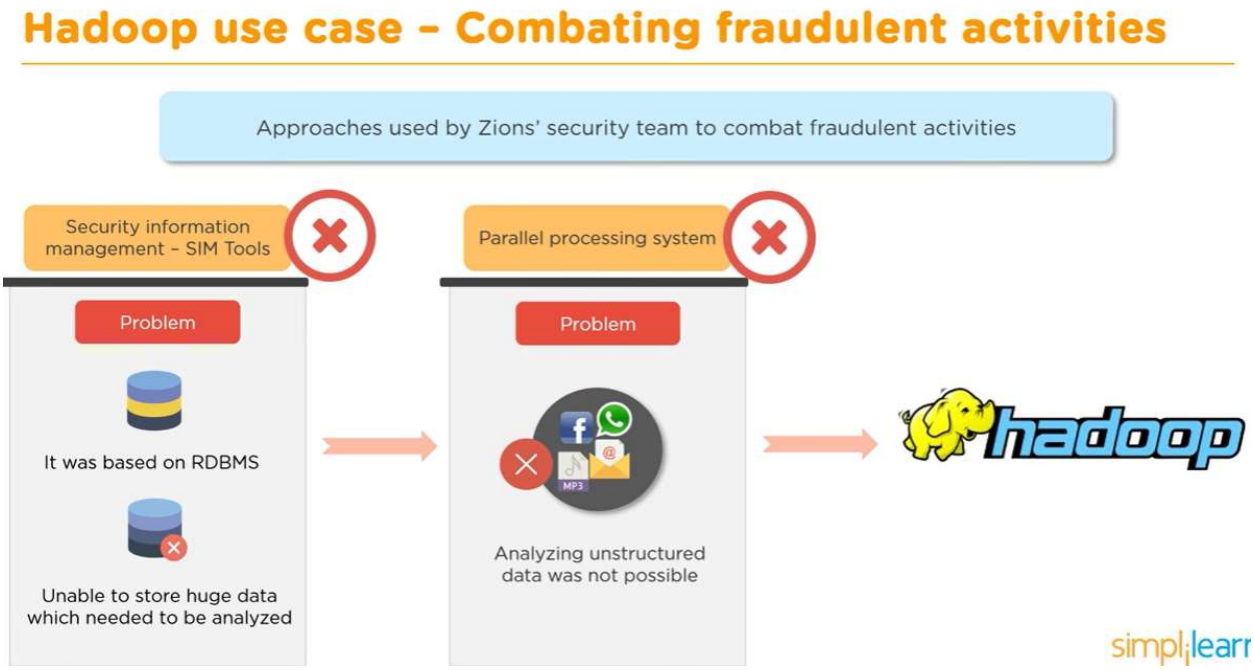
Detecting fraudulent transactions is one among the various problems any bank faces

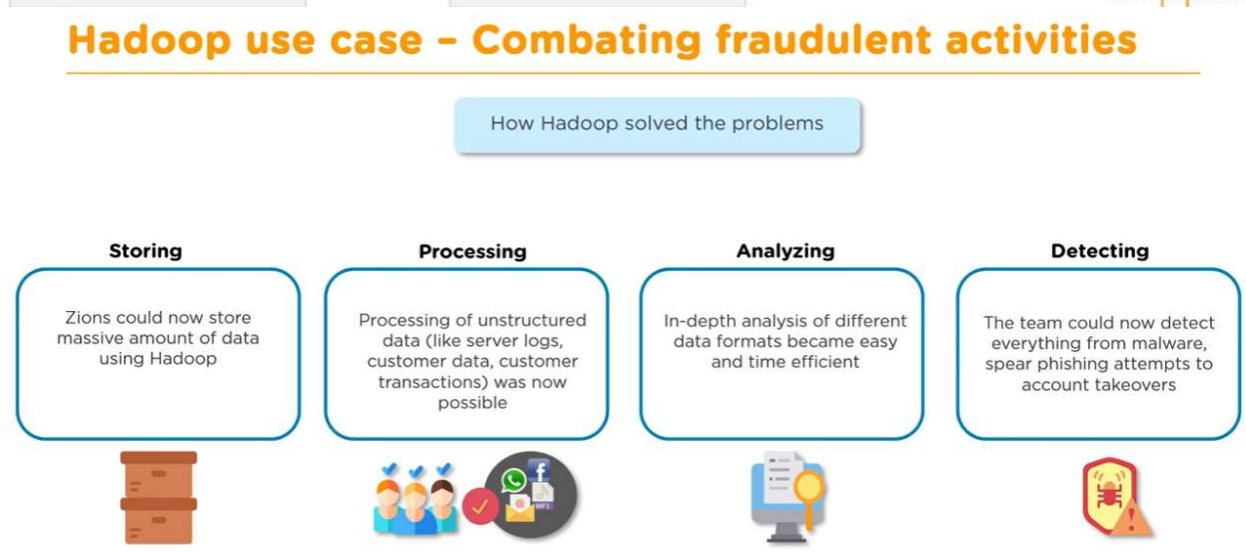


Hadoop use case – Combating fraudulent activities

Zions' main challenge was to combat the fraudulent activities which were taking place







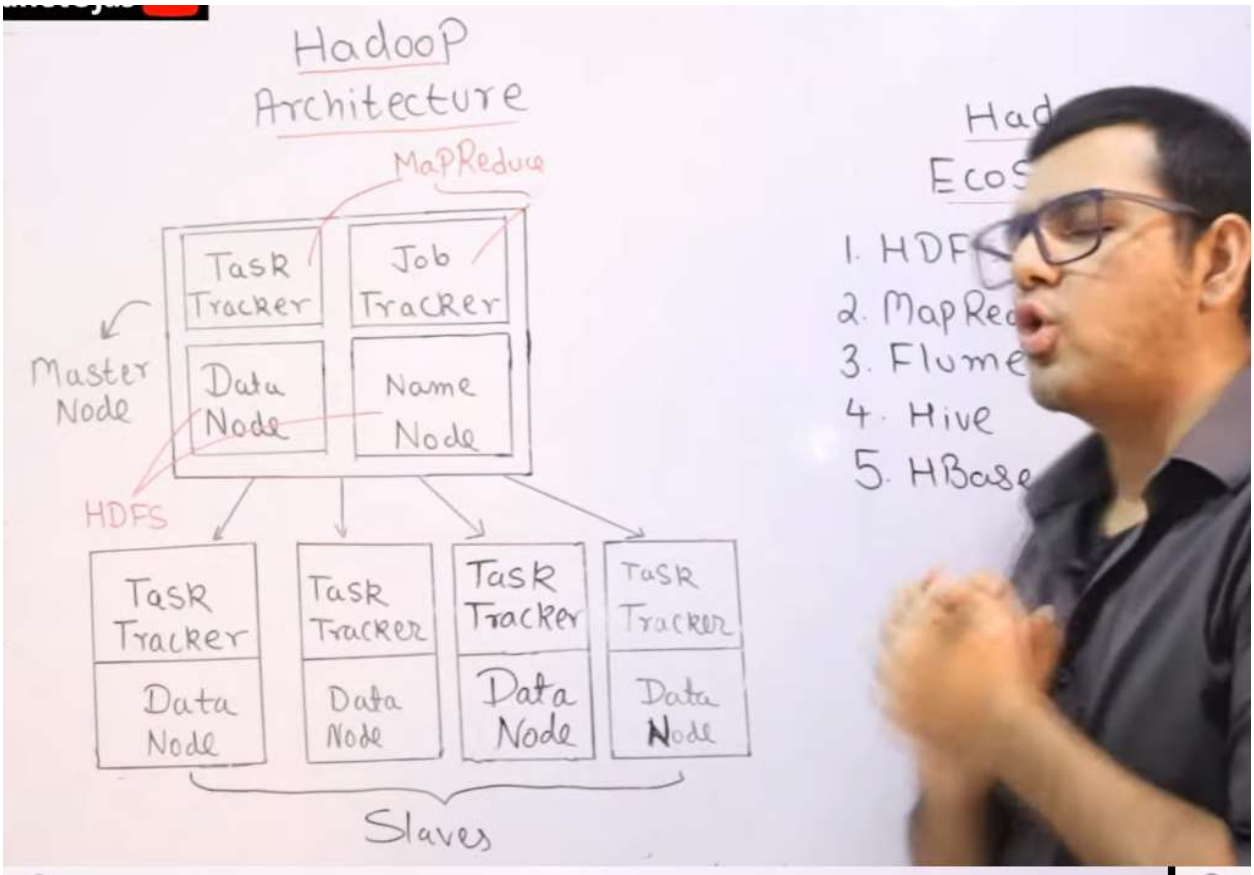
Hadoop Ecosystem

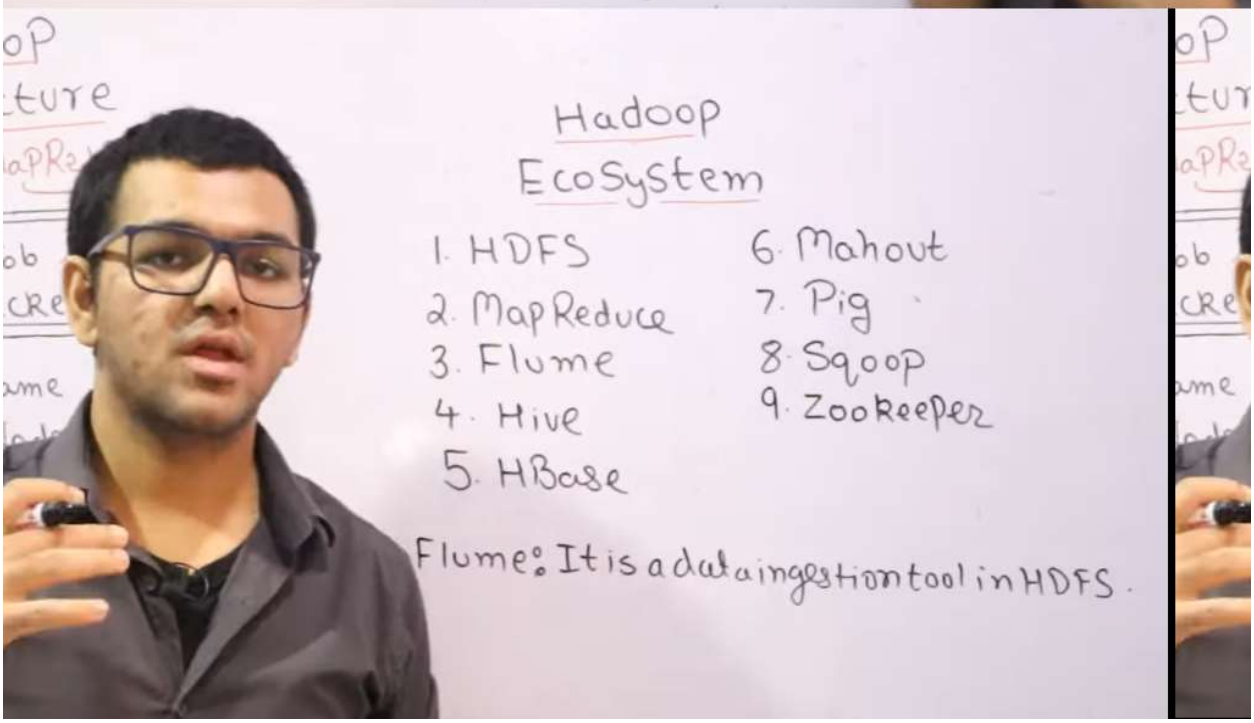
the **Hadoop Ecosystem** within the context of Big Data Analytics (0:00). The speaker explains that Hadoop is a platform designed to solve big data problems, often requiring various tools due to its complexity (0:13-0:28).

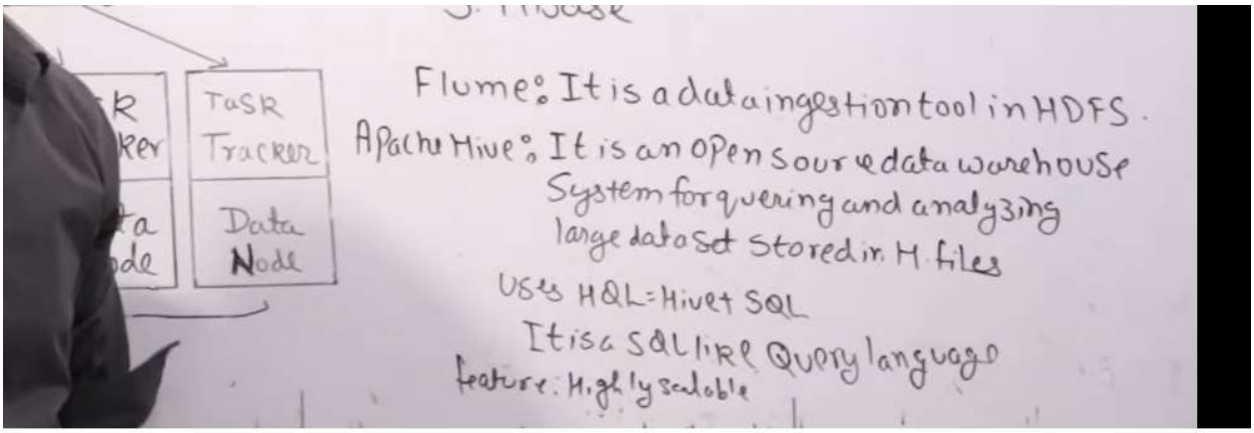
Key components and tools discussed include:

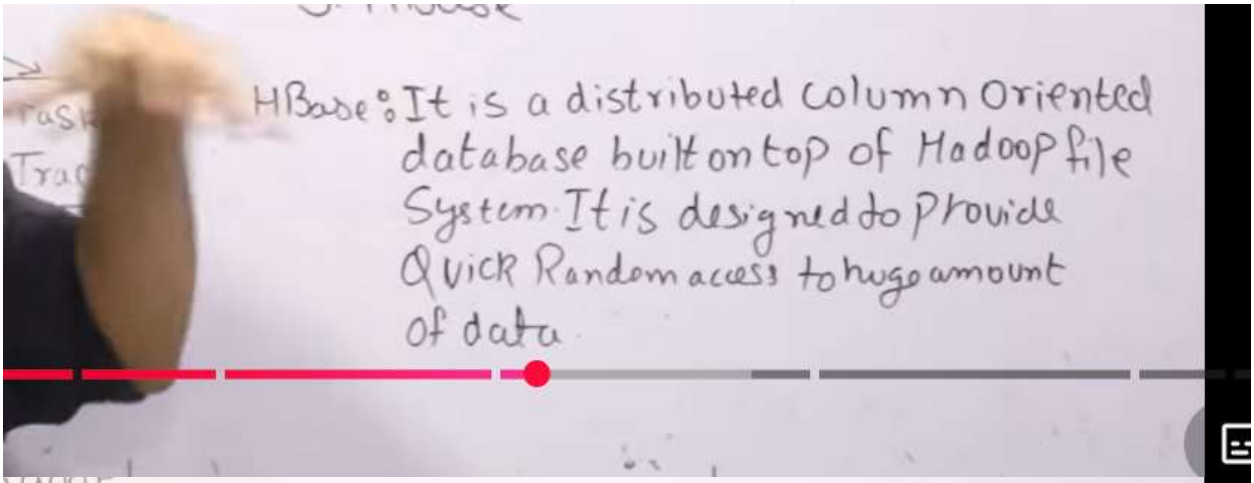
- **MapReduce & HDFS (Hadoop Distributed File System)** (0:30): These are fundamental components of the Hadoop architecture. HDFS is responsible for storing data (3:08-3:13) while MapReduce handles processing (3:04-3:08) and acts as the job tracker (2:45-2:50). The video illustrates their interaction with an example of finding the YouTube channel with the most videos (1:40-2:42).
- **Apache Flume** (3:40): Used to import streaming data into HDFS (3:45-3:53).

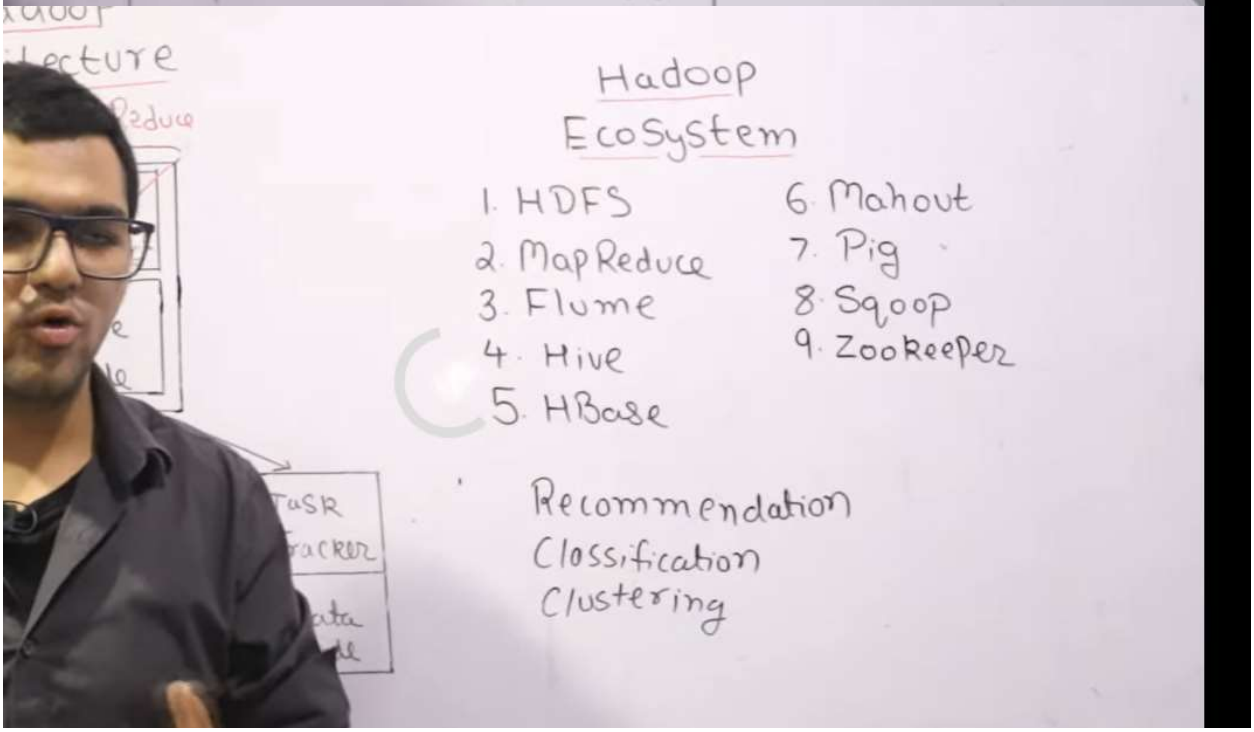
- **Apache Hive** (4:30): Provides a data warehousing solution on top of HDFS, allowing SQL-like queries for easy data retrieval and analysis (4:39-4:54).
- **HBase** (5:00): A NoSQL database that offers random, real-time read/write access to big data stored in HDFS, addressing the limitations of HDFS for quick data access (5:51-6:02).
- **Mahout** (6:00): A machine learning library built on Hadoop, enabling the creation of scalable machine learning algorithms (6:18-6:25).
- **Pig** (7:10): A high-level platform for analyzing large datasets. It simplifies the process of writing complex MapReduce jobs with its Pig Latin language (7:10-7:55).
- **Sqoop** (8:20): Facilitates data transfer between Hadoop and relational database management systems (RDBMS), allowing import and export of structured data (8:32-8:40).
- **ZooKeeper** (8:40): Manages and coordinates the various services within the Hadoop ecosystem, ensuring smooth communication and operation among them (9:10-9:23).


Hadoop
Architecture
MapReduce
Task Tracker
Job Tracker
Master Node
Data Node
Name Node
HDFS
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Slaves
Had
Eco
1. HDF
2. MapRe
3. Flume
4. Hive
5. HBase

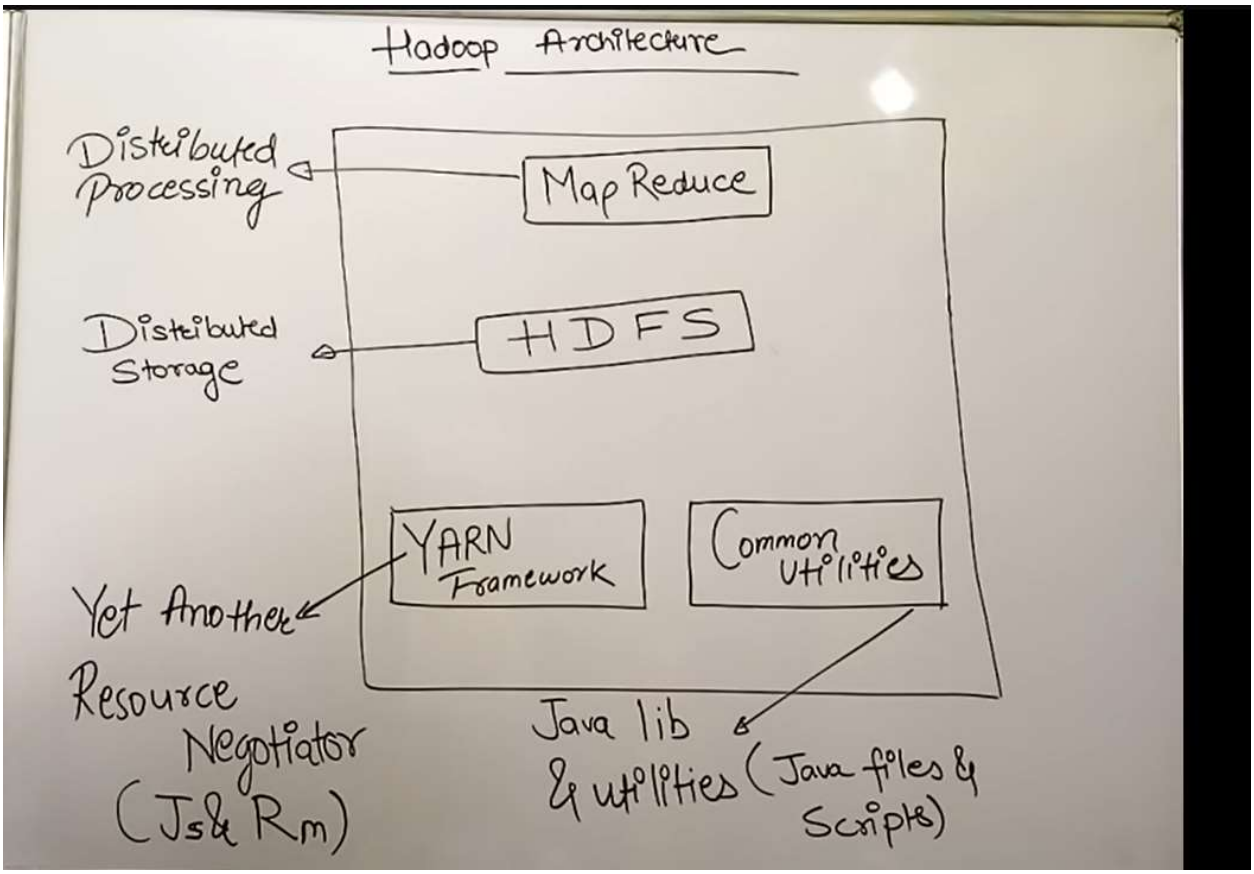

Hadoop
EcoSystem
1. HDFS
2. MapReduce
3. Flume
4. Hive
5. HBase
6. Mahout
7. Pig
8. Sqoop
9. Zookeeper
Flume: It is a data ingestion tool in HDFS.


TaSK Tracker
Data Node
Flume: It is a data ingestion tool in HDFS.
Apache Hive: It is an open source data warehouse System for querying and analyzing large datasets stored in H. files
Uses HQL = Hive+ SQL
It is a SQL like Query language
feature: Highly scalable


HBase: It is a distributed column Oriented database built on top of Hadoop file System. It is designed to provide Quick Random access to huge amount of data.


Hadoop
EcoSystem
1. HDFS
2. MapReduce
3. Flume
4. Hive
5. HBase
6. Mahout
7. Pig
8. Sqoop
9. ZooKeeper
Recommendation
Classification
Clustering

Hadoop Architecture II Map Reduce,HDFS,YARN Framework, Common Utilities



Big Data Analytics Lectures | Introduction to Hadoop

Here's a breakdown of the key topics:

- **Introduction to Hadoop** (0:02-0:58): The speaker introduces Hadoop as a compulsory subject for Big Data Analytics in Mumbai University, noting its syllabus matches that of Data Science. The video will specifically cover Hadoop's architecture.
- **What is Hadoop?** (1:25-3:06): Hadoop is defined as an **open-source software framework** that provides **huge data storage**. It's described as a cloud-like platform where data can be stored, emphasizing that it's a framework, not just a software, similar to how Google Cloud is a platform.
- **Basic Diagram of Hadoop** (3:06-3:56): The video presents a basic diagram showing clients accessing a **cluster** of networked machines. A cluster is explained as a group of machines networked together to process client requests, rather than a single machine.

- **Components of Hadoop** (3:58-5:02): Hadoop consists of five main components, referred to as "demons" because they are essential resident programs for Hadoop to function. These are:

1. Name Node
2. Job Tracker
3. Secondary Name Node
4. Data Node
5. Task Tracker

- **Physical Architecture and Working of Hadoop** (5:03-9:47): The video explains the working of these components in the physical architecture.
- **Client Job Submission**: A client submits a job to the **Name Node** (6:26).
- **Job Division**: The Name Node passes the job to the **Job Tracker** (6:54). The Job Tracker then divides the job into smaller tasks (7:06).
- **Task Distribution**: These tasks are distributed to **Data Nodes** (7:14), which act as managers within the system.
- **Task Execution**: Each Data Node assigns its tasks to **Task Trackers** (7:56), which are responsible for executing the actual work (8:02).
- **Task Completion and Reporting**: Once Task Trackers complete their assigned tasks, they report back to the Job Tracker (8:17). The Job Tracker then accumulates the results and sends the final output back to the client (8:43).
- **Secondary Name Node**: This component acts as a monitor, continuously observing the processes and reactivity of the system (9:00).
- **Communication and Failure Handling**: The Job Tracker continuously communicates with Task Trackers to monitor their status (9:26). If a Task Tracker fails to respond, the Job Tracker assumes it has failed and reassigns its tasks to another Task Tracker (9:32).

Hadoop

- Hadoop is an

  Open Source

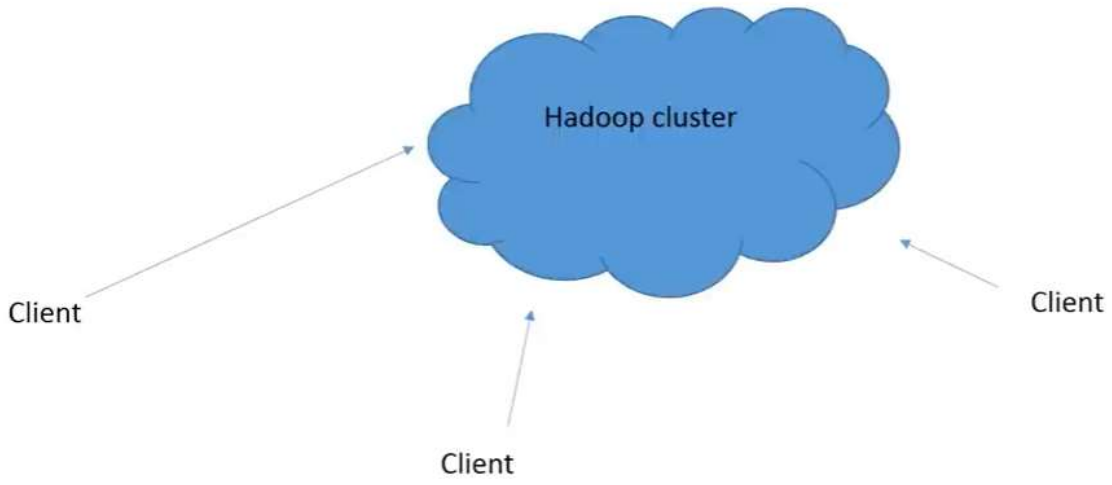
  Software framework

  which provides

  huge Data storage

"Hadoop is an open source software framework which huge data storage facility"

Hadoop diagram



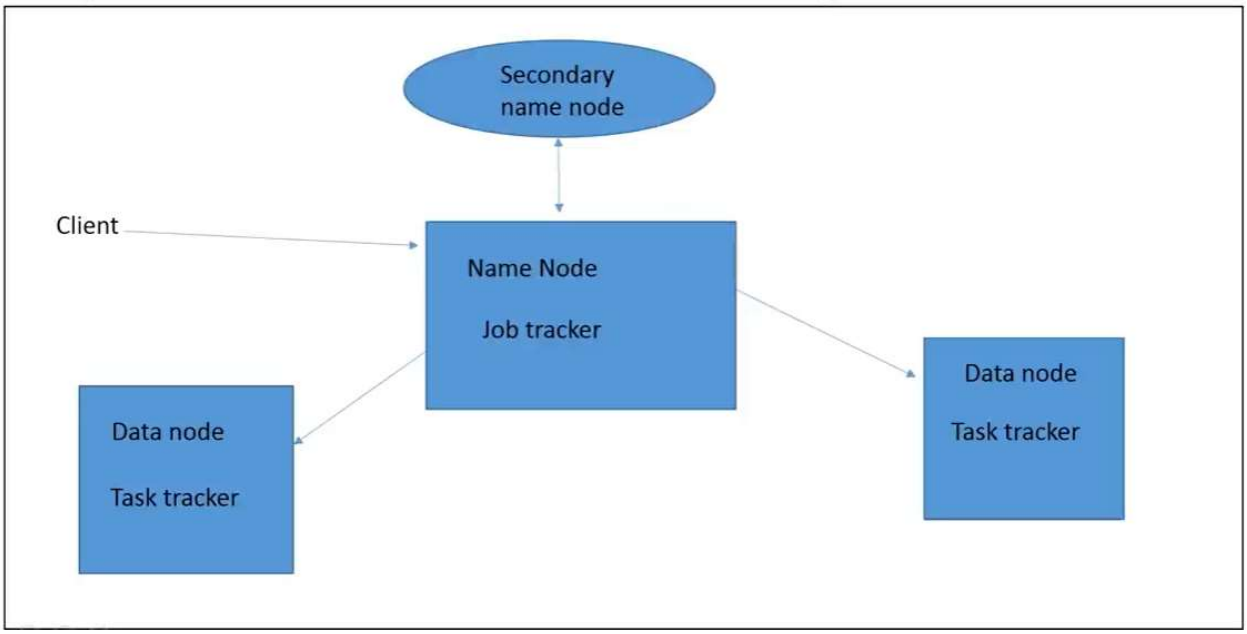
Components present in Hadoop

- Name node(NN)
- Job tracker(JT)
- Secondary name node(SNN)
- Data node (DN)
- Task Tracker(TT)

- Note:- ALL THIS ARE DAEMONS,WHICH ARE RESIDENT PROGRAMS

- KEYWORD:-J SNDT

Physical Architecture of hadoop



Process

- Client provides it's job request to Hadoop.
- This request is accepted by the name Node
- Name node is Master in Hadoop
- It also contains a Job tracker which is again a part of Master.
- This job is divided into task's and job tracker provides it to the Data node.
- Now the Data node is a slave and it posseses Task tracker which actually performs the task.
- And job tracker continuously communicates with task tracker and if anytime it fails to reply then it assumes that the task traker may have crashed

Name node

- It is the master of HDFS
- It has Job tracker which keeps track of files distributed to Data nodes
- Data node is the only single point of failure

Data node

- It is the slave of HDFS
- It takes client block address from Name Node
- For replication purpose it can communicate with other
- DataNode informs local changes /updates to NameNod

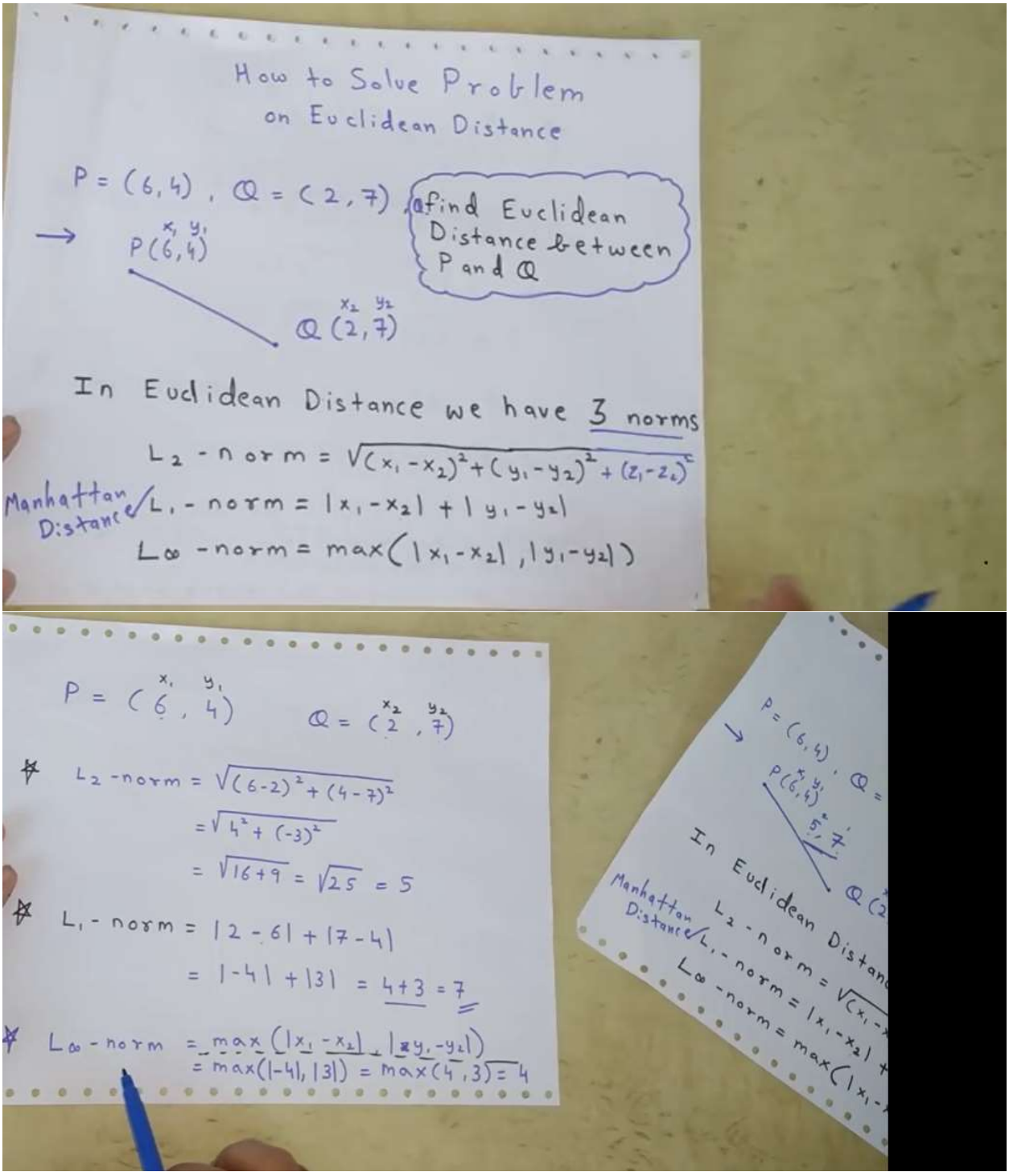
Job tracker

- It determines the files to process
- Only one Jobtracker per Hadoop cluster is allowed
- It runs on a server as a master node of cluster

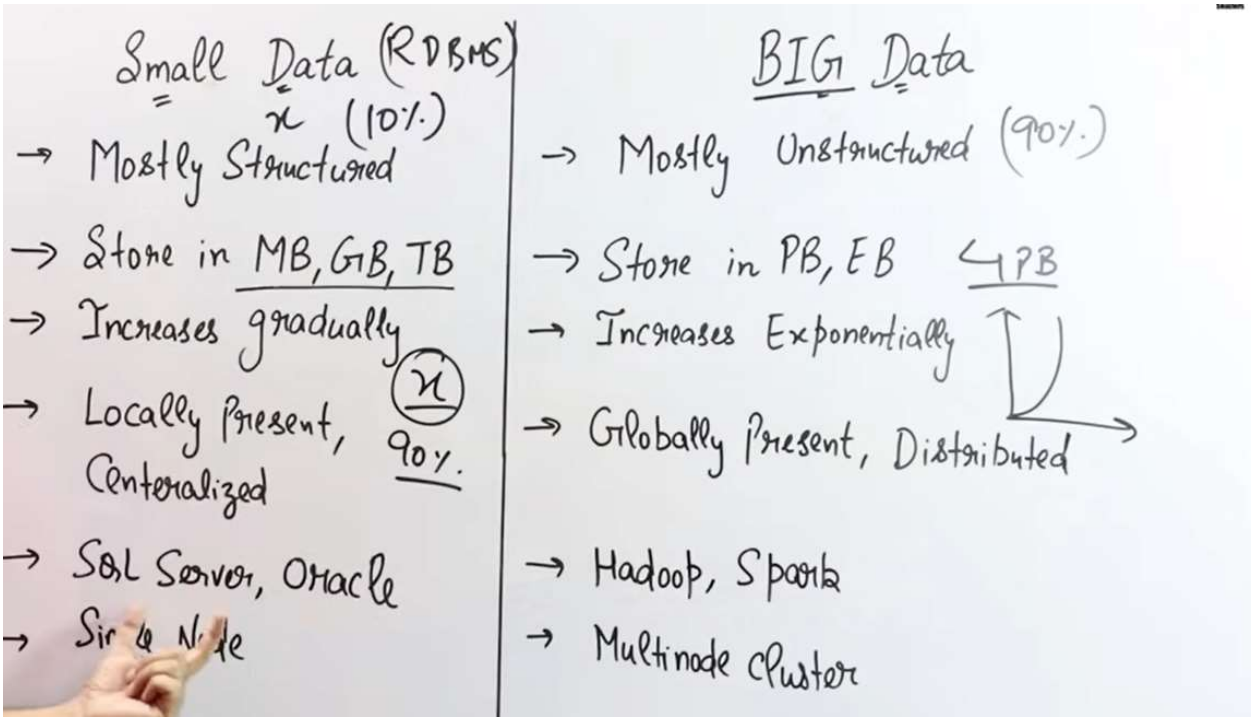
Task tracker

- There is a single task tracker per slave node.
- It may handle multiple task parallelly .
- Individual tasks are assigned by jobtracker to tasktracker.
- Job tracker continuously communicates with task tracker and if anytime it fails to reply then it assumes that the task tracker has crashed.

Big Data Analytics Lectures | Euclidean Distance with Solved Example



Introduction to BIG Data



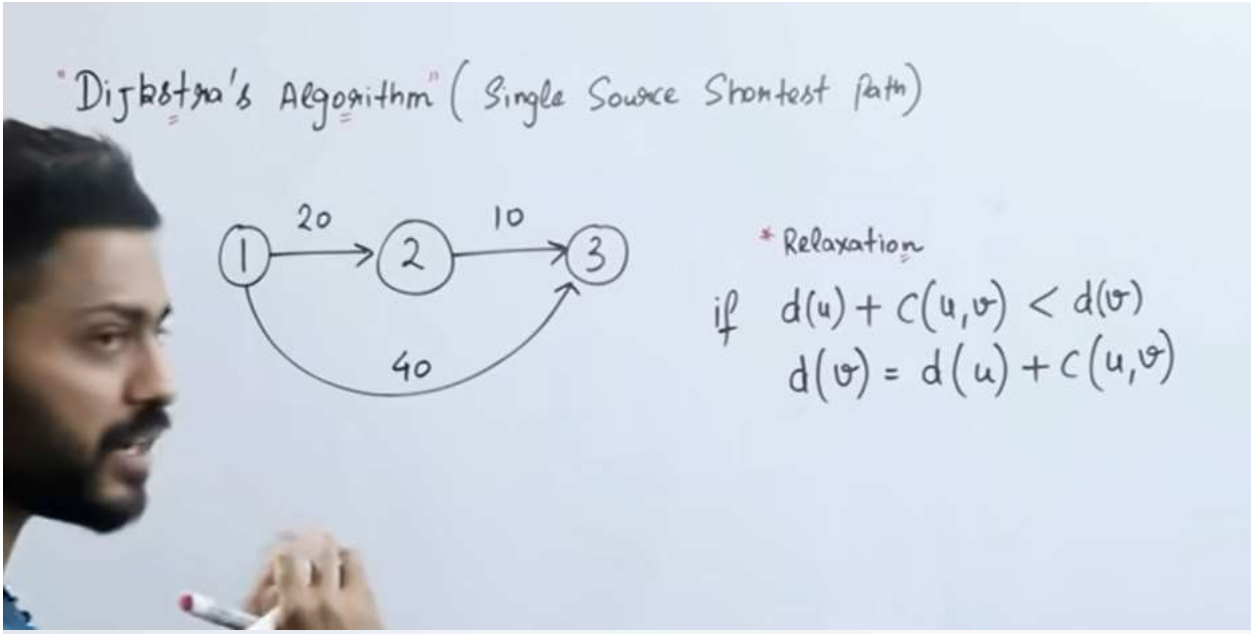
Dijkstra's Algorithm - Single Source Shortest Path - Greedy Method

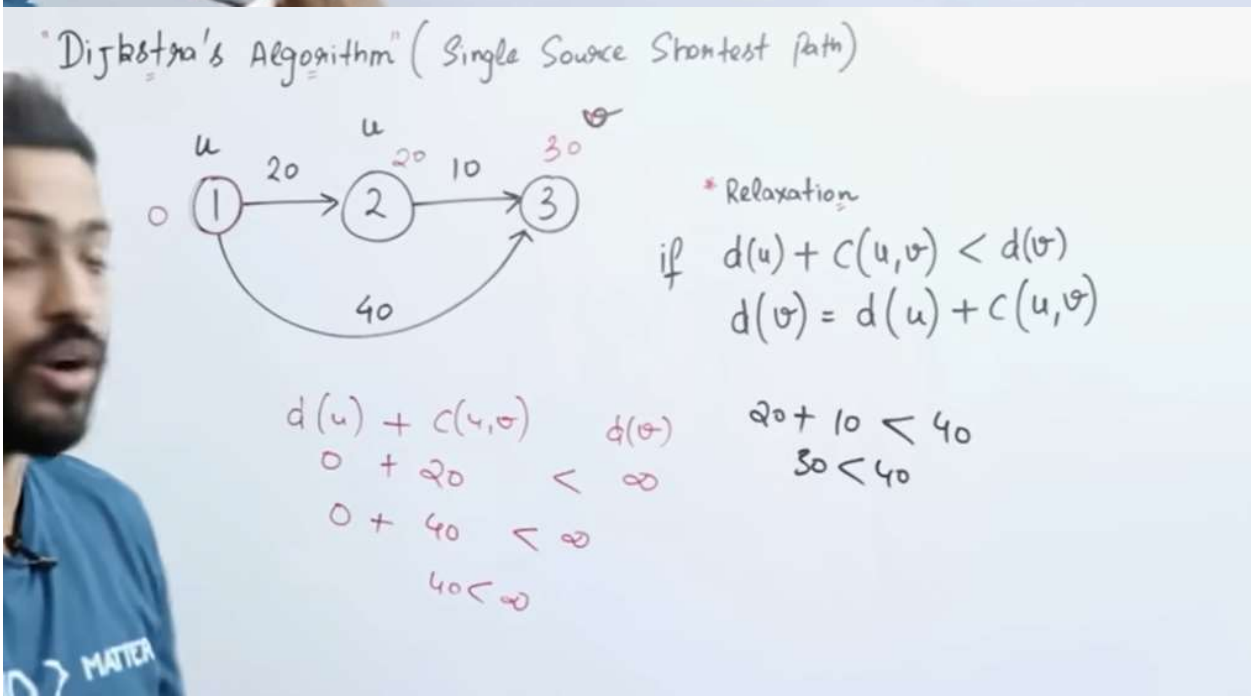
Here's a breakdown of the video's content:

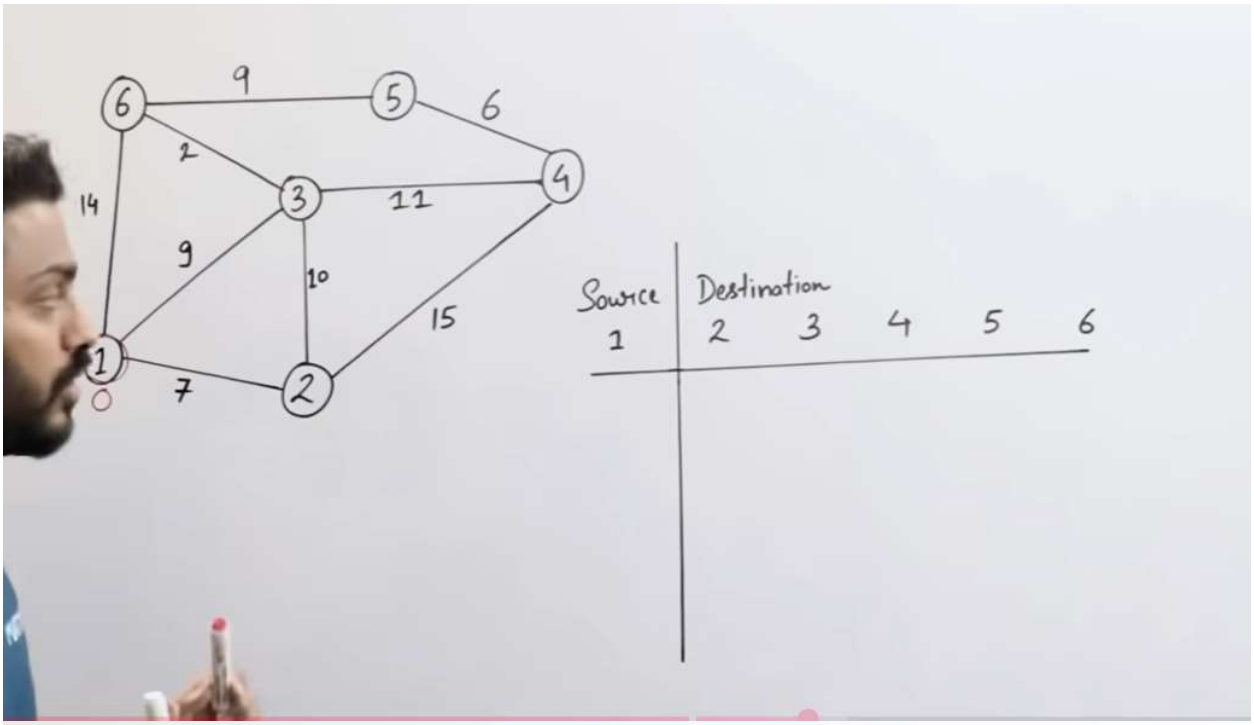
- **Introduction and Advantages** (0:39): Dijkstra's Algorithm is a **greedy approach** that finds the shortest path from a single source in a graph. It's often referred to as a **single-source shortest path algorithm** or a **minimization algorithm** because it aims to minimize the path cost. Real-world applications include **Google Maps** (0:45), **social networking**, **DNA mapping**, and **telecommunication networks** (1:13). The algorithm always provides an **optimal answer** (1:55).
- **Working Principle - Relaxation** (1:57): The core of Dijkstra's Algorithm lies in its **relaxation technique**. The video explains this concept using a small graph with three vertices (1, 2, 3). Initially, the distance to all nodes from the source (e.g., node 1) is considered infinite, except for the source itself (0:00). The algorithm iteratively updates the shortest distance to each node. For example, to find the shortest path from node 1 to node 2, it compares the current distance to node 2 with the sum of the distance to the current node (node 1, which is 0) and the cost of the edge from node 1 to node 2 (20). If the new path is shorter, the distance is updated (3:24). This process is repeated for all adjacent nodes (5:25).
- **Detailed Example** (7:48): The video then demonstrates Dijkstra's Algorithm on a larger, **undirected graph** with six vertices. It shows how to create a table to track the shortest distances from the source node (node 1) to all other nodes (2, 3, 4, 5, 6). The steps involve:

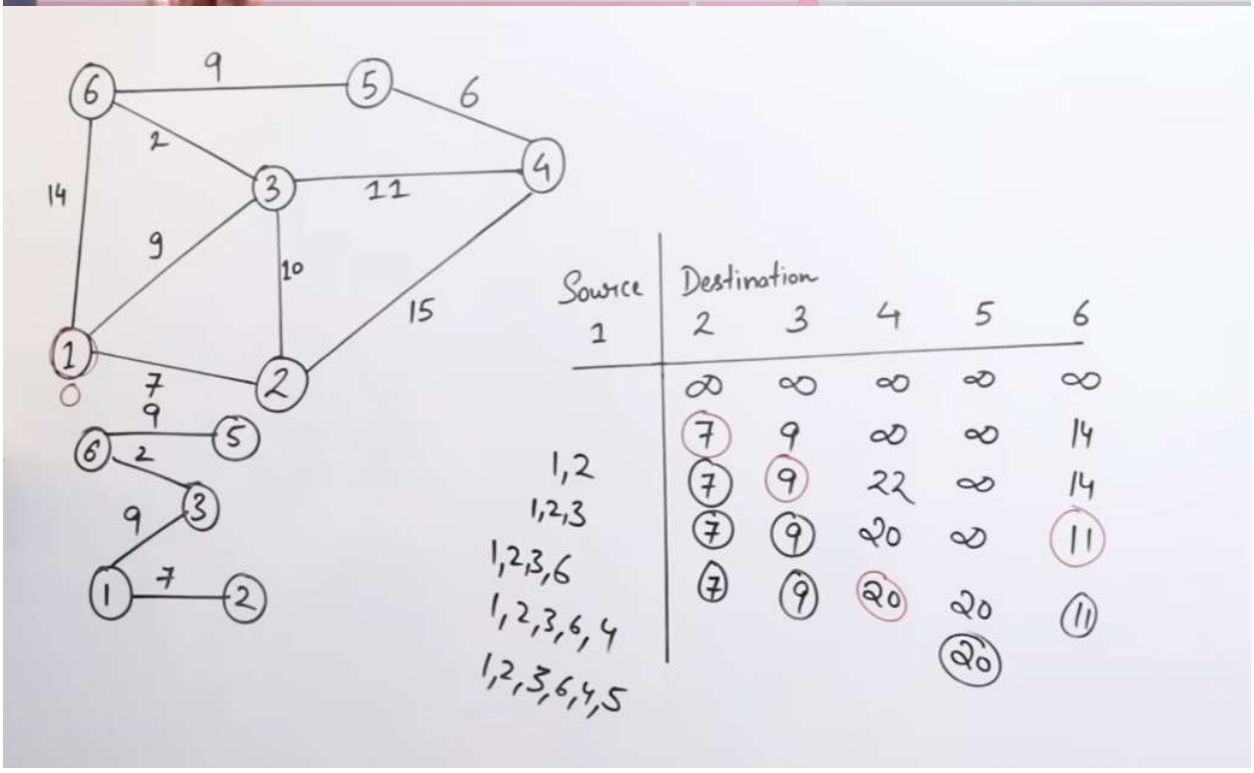
1. Initializing distances: Source to itself is 0; others are infinite (8:24).
2. Updating direct connections: For example, from node 1, direct paths to node 2 (cost 7), node 3 (cost 9), and node 6 (cost 14) are updated (9:06).

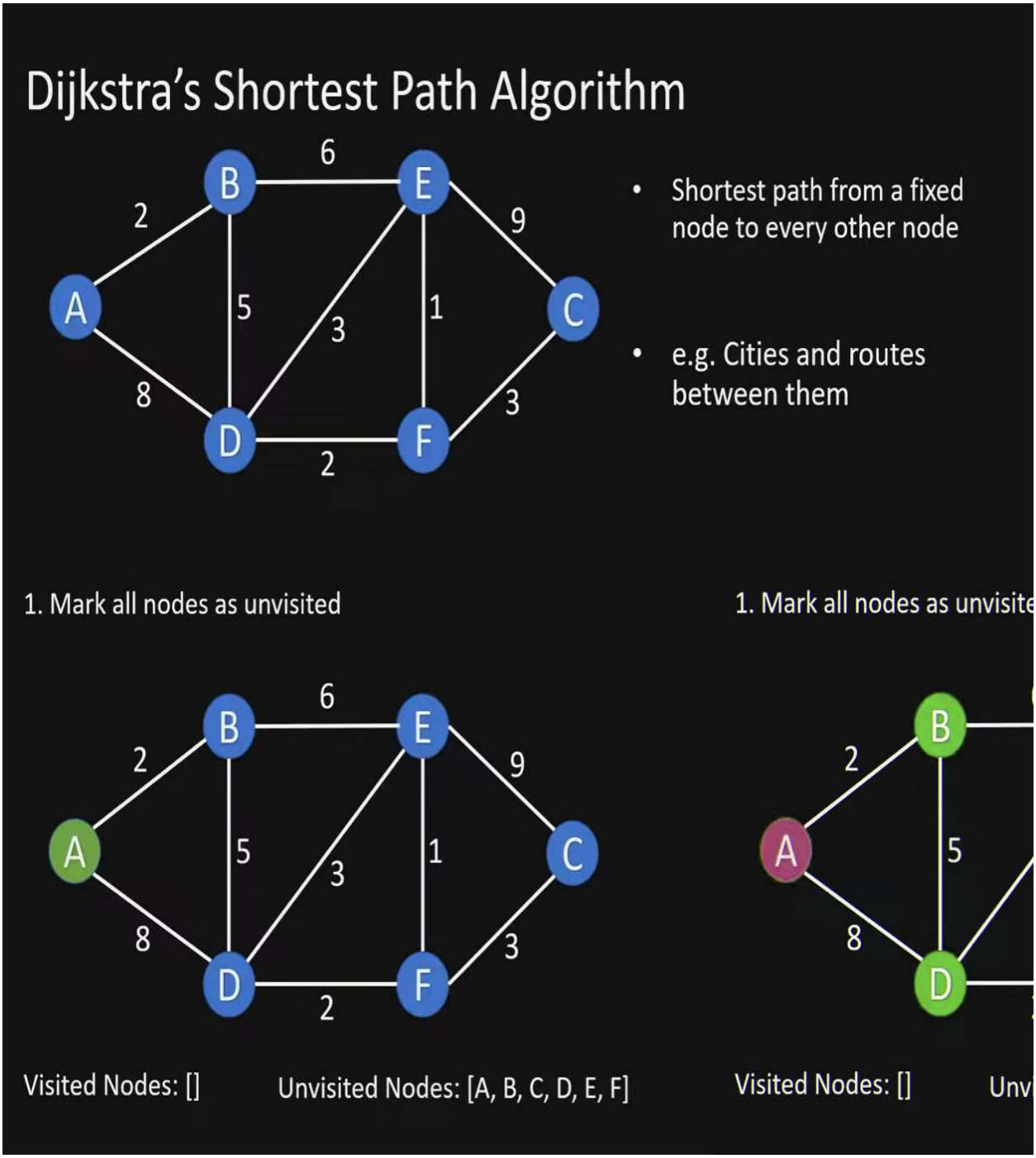
3. Selecting the minimum: The node with the minimum current shortest distance (e.g., node 2 with cost 7) is selected and fixed (9:47).
4. Relaxing neighbors: From the fixed node (node 2), its neighbors are checked. If a shorter path is found through node 2, the distances are updated (10:19).
5. Repeating the process: Steps 3 and 4 are repeated until all nodes have been visited and their shortest distances from the source are finalized (11:08-14:33). The final result is a tree-like structure showing the shortest paths from the source to all other nodes (14:47).






6
9
5
6
2
4
14
3
11
9
10
15
1
7
2
Source
Destination
1
2
3
4
5
6


6
9
5
6
2
4
14
3
11
9
10
15
1
7
2
7
9
6
5
2
9
3
1
7
2
Source
Destination
1
2
3
4
5
6
∞ ∞ ∞ ∞ ∞
7 9 ∞ ∞ 14
7 9 22 ∞ 14
7 9 20 ∞ 11
7 9 20 20 11
20
1,2
1,2,3
1,2,3,6
1,2,3,6,4
1,2,3,6,4,5

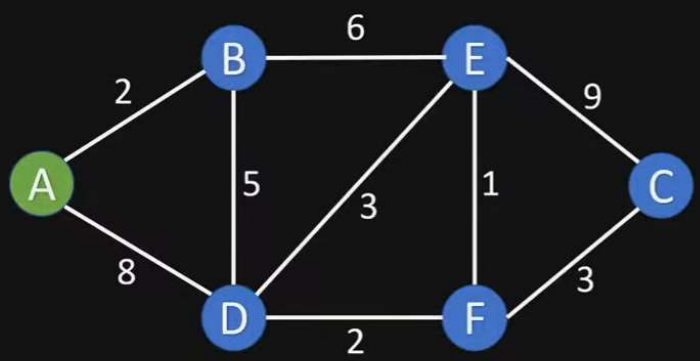


The video details the following steps to implement Dijkstra's algorithm:

1. **Initialization:** All nodes are initially marked as unvisited. Two lists are created: `unvisited nodes` (containing all graph nodes) and `visited nodes` (initially empty) (0:45-1:01).
2. **Tentative Distance Assignment:** Each node is assigned a tentative distance value. The starting node's distance is set to 0, and all other nodes are set to infinity (1:02-1:32). A table is used to track the current shortest distance from the start node to each node and the `previous node` that leads to this shortest distance (1:32-1:43).

3. **Iteration and Update:** The algorithm iteratively selects the unvisited node with the smallest tentative distance (2:49-3:11). For the current node, it calculates the distances to all its unvisited neighbors by adding the current node's distance to the weight of the edge connecting them (3:18-3:37). If a newly calculated distance is shorter than the current tentative distance for a neighbor, the table is updated with the new shorter distance and the current node as the `previous node` (2:05-2:22, 3:56-4:02, 4:13-4:27).
4. **Marking Visited:** Once all unvisited neighbors of the current node have been processed, the current node is marked as visited and moved from the `unvisited nodes` list to the `visited nodes` list (2:33-2:40, 4:32-4:37, 5:38-5:41, 7:02-7:05, 7:09-7:12).
5. **Reconstruction of Path:** After the algorithm completes, the table contains the shortest distance to all nodes and the `previous node` information, which can be used to reconstruct the actual shortest path from the starting node to any other node (7:16-8:13).

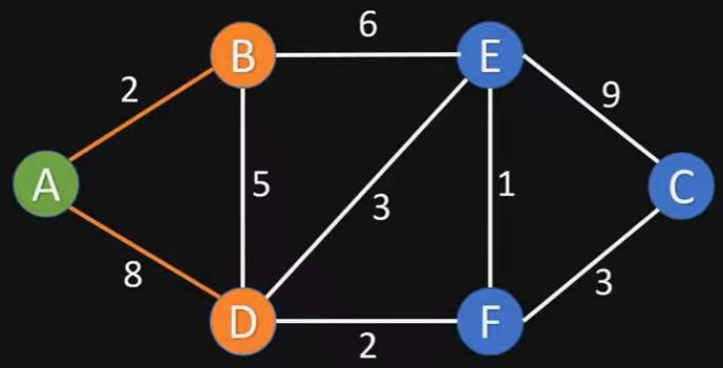
2. Assign to all nodes a tentative distance value



Node	Shortest Distance	Previous Node
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

Visited Nodes: [] Unvisited Nodes: [A, B, C, D, E, F]

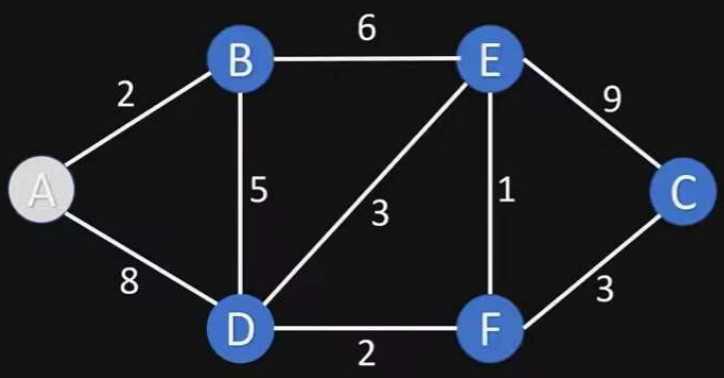
3. For the current node calculate the distance to all unvisited neighbours
3.1. Update shortest distance, if new distance is shorter than old distance



Node	Shortest Distance	Previous Node
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

Visited Nodes: [] Unvisited Nodes: [A, B, C, D, E, F]

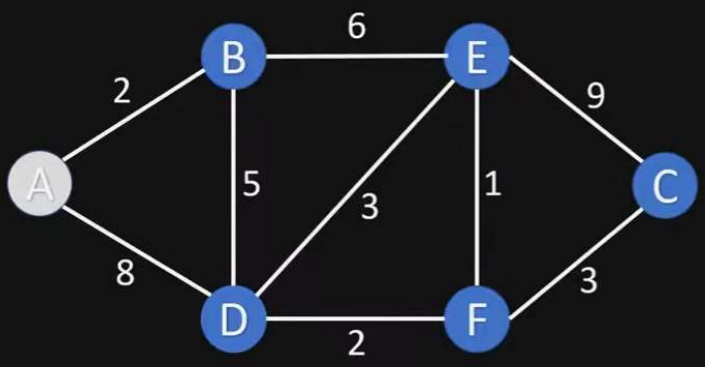
4. Mark current node as visited



Node	Shortest Distance	Previous Node
A	0	
B	**2**	A
C	∞	
D	**8**	A
E	∞	
F	∞	

Visited Nodes: [A] Unvisited Nodes: [B, C, D, E, F]

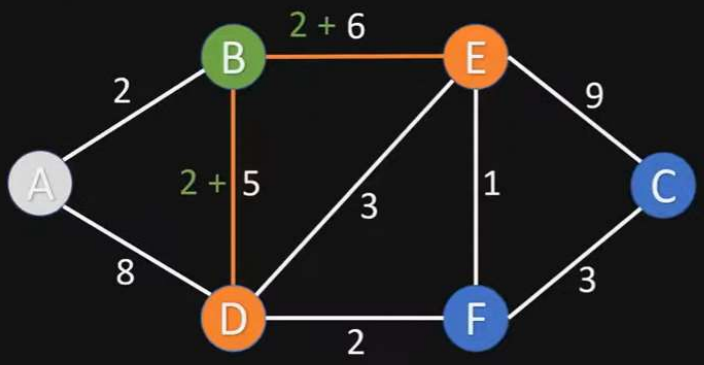
5. Choose new current node from unvisited nodes with minimal distance



Node	Shortest Distance	Previous Node
A	0	
B	**2**	A
C	∞	
D	**8**	A
E	∞	
F	∞	

Visited Nodes: [A] Unvisited Nodes: [B, C, D, E, F]

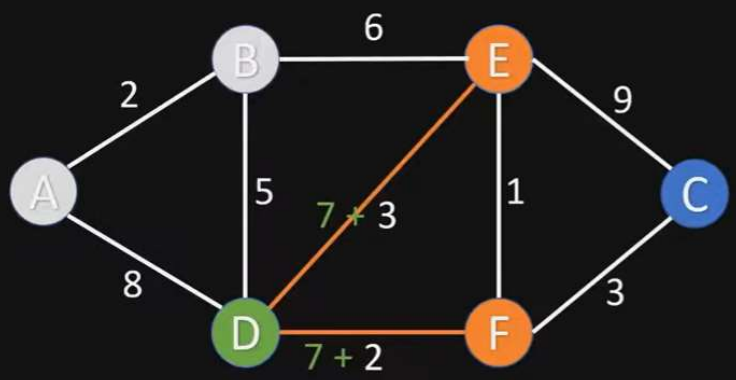
3. For the current node calculate the distance to all unvisited neighbours
3.1. Update shortest distance, if new distance is shorter than old distance



Node	Shortest Distance	Previous Node
A	0	
B	**2**	A
C	∞	
D	**8**	A
E	**8**	B
F	∞	

Visited Nodes: [A] Unvisited Nodes: [B, C, D, E, F]

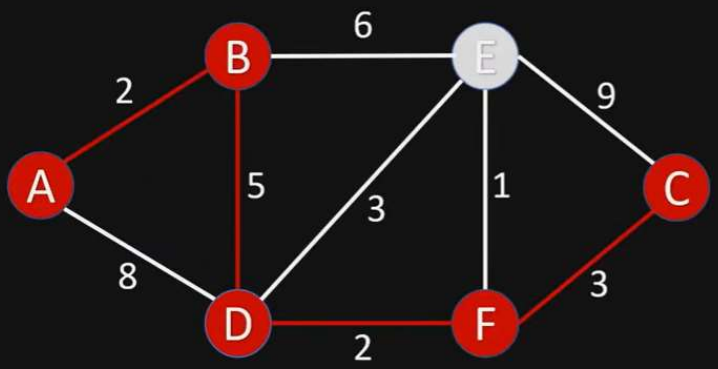
3. For the current node calculate the distance to all unvisited neighbours



Node	Shortest Distance	Previous Node
A	0	
B	**2**	A
C	∞	
D	**7**	B
E	**8**	B
F	∞	

Visited Nodes: [A, B] Unvisited Nodes: [C, D, E, F]

Get shortest path from A to C



Node	Shortest Distance	Previous Node
A	0	
B	**2**	A
C	**12**	F
D	**7**	B
E	**8**	B
F	**9**	D