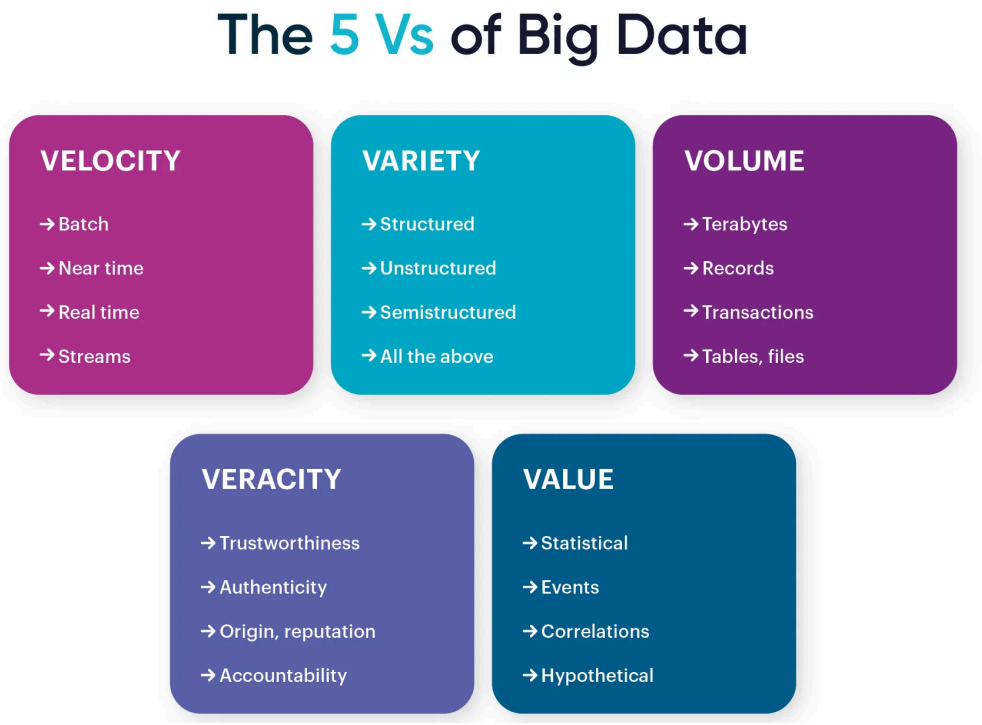
Module 1. Introduction to Big data and Hadoop

Q. 5 V's of Big Data
=>



5 V's of Big Data Infographic

1. **Volume – Scale of Data**
   The name 'Big Data' itself is related to a size which is enormous. Volume is a huge amount of data. To determine the value of data, size of data plays a very crucial role. If the volume of data is very large, then it is actually considered as a 'Big Data'. This means whether a particular data can actually be considered as a Big Data or not is dependent upon the volume of data. Hence while dealing with Big Data it is necessary to consider a characteristic 'Volume'.

   Example: In the year 2016, the estimated global mobile traffic was **6.2 Exabytes (6.2 billion GB) per month**. Also, by the year 2020 we will have almost **40000 Exabytes of data**.

2. **Velocity – Speed of Data**
   Velocity refers to the high speed of accumulation of data. In Big Data, velocity data flows in from sources like machines, networks, social media, mobile phones, etc. There is a massive and continuous flow of data. This determines the potential of data — how fast the data is generated and processed to meet the demands. Sampling data can help in dealing with issues like 'velocity'.

Example: There are more than **3.5 billion searches per day** made on Google. Also, Facebook users are increasing by **approximately 22%** year by year.

3. **Variety – Diversity of Data**
   It refers to the nature of data that is structured, semi-structured, and unstructured. It also refers to heterogeneous sources. Variety is basically the arrival of data from new sources that are both inside and outside of an enterprise.

   - **Structured data:** Organized data with a defined length and format.

   - **Semi-structured data:** Semi-organized data that does not conform to formal structures (e.g., log files).

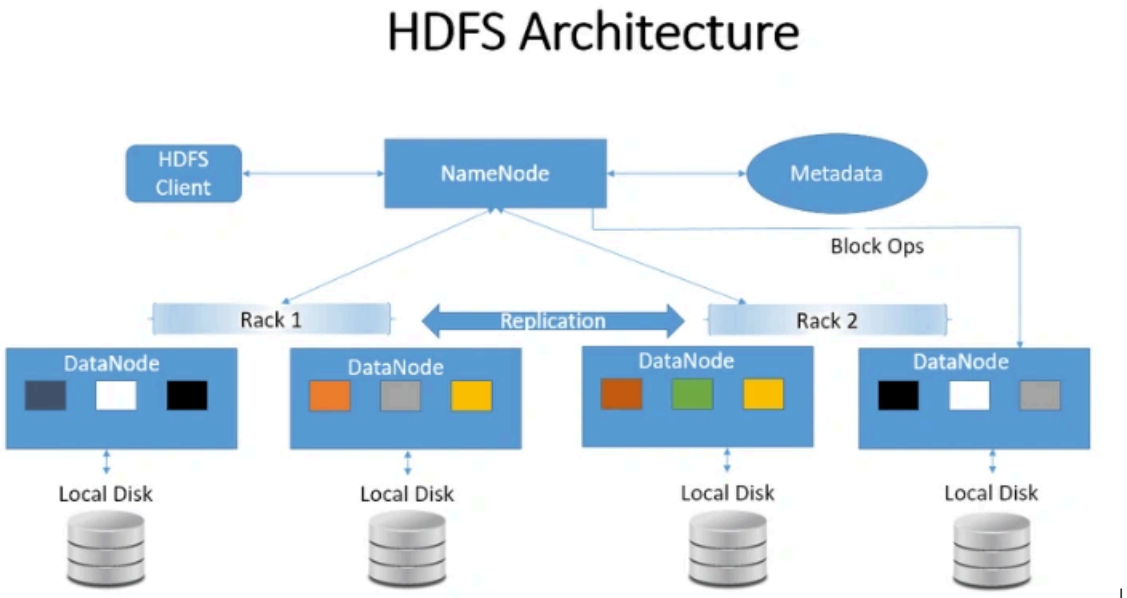
   - **Unstructured data:** Unorganized data that doesn't fit into rows and columns (e.g., texts, pictures, videos).

4. **Veracity – Trustworthiness of Data**
   It refers to inconsistencies and uncertainty in data — data can sometimes get messy and its quality and accuracy are difficult to control. Big Data is also variable because of the multitude of data dimensions resulting from multiple disparate data types and sources.

   Example: Data in bulk could create confusion, whereas less data could convey half or incomplete information.

5. **Value – Usefulness of Data**
   After considering the 4 V's, there comes one more V which stands for **Value**. A bulk of data having no value is of no good to the company unless it is turned into something useful. Data in itself is of no use or importance but needs to be converted into something valuable to extract information. Hence, **Value** is often considered the most important V of all.

Q. Data Node vs Name Node
=>



Feature	NameNode (Master)	DataNode (Slave)
Role	Manages metadata and directory structure of HDFS	Stores actual blocks of data
Type of Node	Master node in Hadoop Distributed File System (HDFS)	Worker (slave) nodes in HDFS
Data Stored	File system namespace, block locations, permissions	Actual file data in block format
Memory Usage	High (keeps metadata in memory for fast access)	Relatively low (stores data on disk)
Dependency	HDFS cannot function without NameNode	DataNodes depend on NameNode for instructions

Failure Impact	Single point of failure (if no backup or HA setup)	Loss of one DataNode does not stop HDFS, data replicated elsewhere
Communication	Communicates with clients and DataNodes	Communicates only with NameNode (for block reports, heartbeats)
Operations	Handles namespace operations like open, close, rename	Handles read/write requests for data blocks
Location Knowledge	Knows where every block of a file is stored	Does not know about overall file structure
Hardware	Requires reliable, high-performance hardware	Can use commodity hardware (inexpensive systems)

Q. Secondary Name node is a backup of Name node. Is this statement True or False? Justify your answer.
=>

The statement is **False.** The Secondary NameNode is *not* a backup or standby NameNode. Its function is often misunderstood because of its name.

Understanding the NameNode

- In **Hadoop Distributed File System (HDFS)**, the **NameNode** is the master node.
- It stores the **metadata** of the file system — for example:
  - Directory tree structure of HDFS
  - File names, permissions, and replication factors
  - Mapping of file blocks to DataNodes
- Since the NameNode holds all the namespace and block information in memory, it is a **single point of failure (SPOF)** in older Hadoop versions.

Role of the Secondary NameNode

- The **Secondary NameNode (SNN)** does **not** replace or back up the NameNode.
- Its main purpose is to perform **checkpointing**, i.e., periodically merging:
  - **FsImage** (a snapshot of HDFS metadata) and
  - **EditLogs** (a log of every namespace change)
- By combining these into a single, updated FsImage, the Secondary NameNode prevents EditLogs from growing too large and keeps the NameNode efficient.

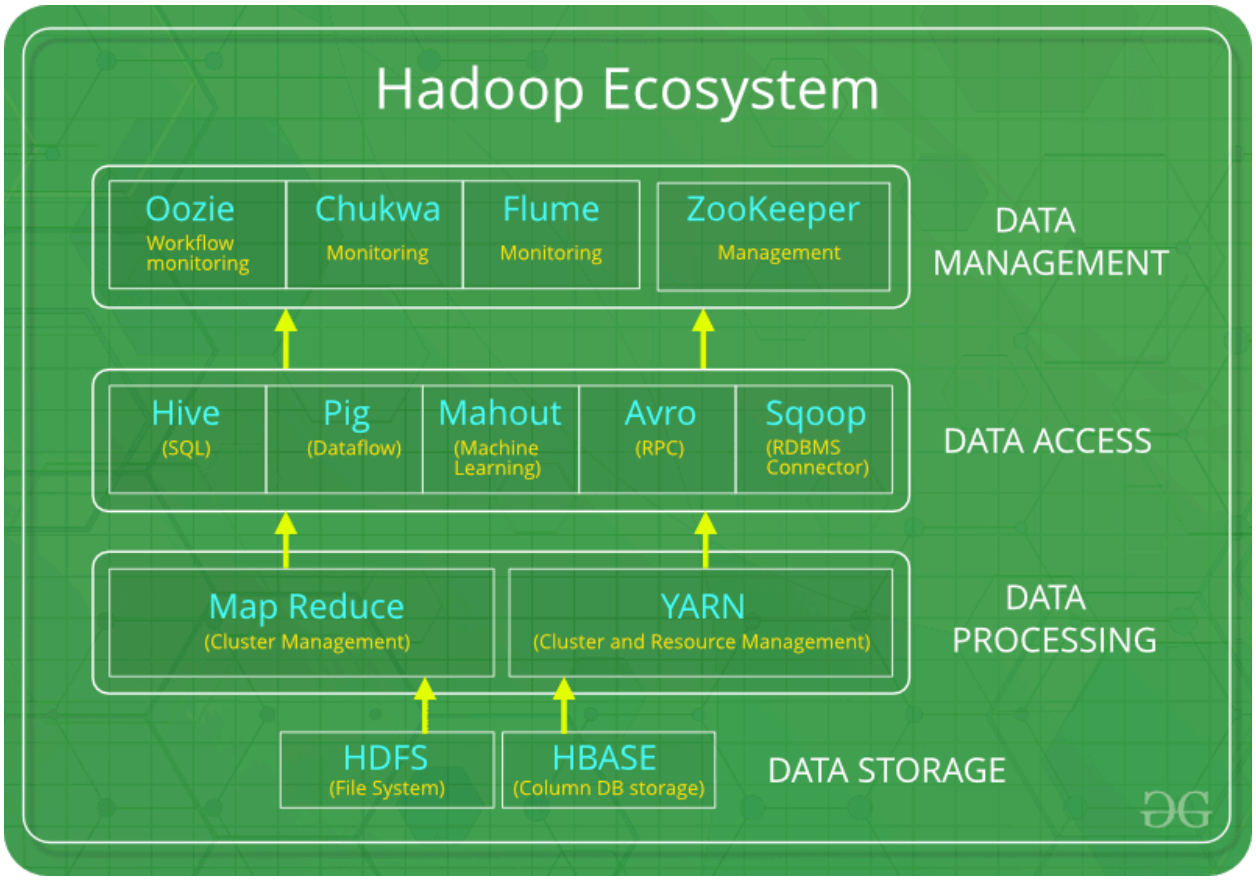
Why it is NOT a Backup

- **No Real-time Synchronization:** The SNN does not hold an up-to-date, in-memory copy of metadata. It only works at intervals, not continuously.
- **No Automatic Failover:** If the NameNode crashes, the SNN **cannot automatically take over**. The cluster will still stop functioning.
- **Manual Recovery Required:** Administrators can use the checkpointed FsImage and EditLogs stored by the SNN to reconstruct metadata and start a new NameNode instance, but this is a **manual process**, not seamless failover.

Illustrative Example

- Suppose the NameNode is running and the SNN checkpoints every hour.
- If the NameNode fails at 12:45 p.m., the last consistent checkpoint might be from 12:00 p.m.
- The cluster will stop because there is no active NameNode.
- An administrator must manually copy the SNN's latest FsImage and EditLogs to a fresh NameNode instance and restart the service.
- This proves that the SNN is **not** an active backup or standby NameNode.

Q. Hadoop core components and ecosystem diagram
=>



Hadoop is an **open-source framework** developed by the Apache Software Foundation for storing and processing **massive datasets in a distributed computing environment**. Inspired by Google's MapReduce and Google File System (GFS), Hadoop uses **commodity hardware**, is **highly fault-tolerant**, and is capable of **horizontal scaling** (adding more machines to handle more data). It allows organizations to process structured, semi-structured, and unstructured data efficiently.

1. Hadoop Distributed File System (HDFS)

HDFS is the **primary storage layer of Hadoop**, designed to store huge files reliably across multiple machines.

- **Architecture:**
  HDFS follows a **master-slave model**:
    - **NameNode (Master):** Manages the *metadata* of the file system, including filenames, block locations, and access permissions. It does not store actual data but keeps track of where each piece of data resides in the cluster.

    - **DataNodes (Slaves):** Store actual data blocks on local disks. They regularly send heartbeats and block reports to the NameNode to confirm their health and the status of data blocks.

- **Key Characteristics:**
    - **Distributed Storage:** Files are divided into fixed-size *blocks* (default 128 MB or 256 MB) and stored across different nodes in the cluster.
    - **Fault Tolerance via Replication:** Each block is replicated (default replication factor is 3). If one DataNode fails, another copy is always available.
    - **High Throughput:** Optimized for reading large files in streaming mode rather than random access.
    - **Scalability:** Additional nodes can be added seamlessly to handle growing data needs.
- **Example:** If a 512 MB log file is uploaded, HDFS splits it into four 128 MB blocks and distributes them across different nodes with three replicas each. Even if a node goes down, the file remains accessible.

2. Hadoop MapReduce

MapReduce is the **processing engine** of Hadoop that executes computations on data stored in HDFS using a *parallel programming model*.

- **Working Mechanism:**
    - **Map Phase:** Input data is divided into *splits*, and each split is processed independently by mapper tasks that transform the input into intermediate *key-value pairs*.
    - **Shuffle and Sort:** The framework groups all values by key and redistributes data to the reducers.
    - **Reduce Phase:** Reducer tasks aggregate or summarize the grouped data to produce the final result.
- **Advantages:**
    - **Parallel Processing:** Many nodes can work on different chunks of data simultaneously, significantly speeding up computation.
    - **Fault Tolerance:** If a task fails, it is automatically rescheduled on another node.

- **Simplicity for Developers:** Programmers only write map() and reduce() functions, while the framework handles scheduling, synchronization, and communication.

- **Example:** If analyzing website logs to count the number of visits per URL, mappers will output (URL, 1) pairs for every visit, and reducers will sum these to produce (URL, total_visits) results.

3. Hadoop YARN (Yet Another Resource Negotiator)

YARN is the **resource management and job scheduling layer** introduced in Hadoop 2.x, separating resource allocation from MapReduce logic.

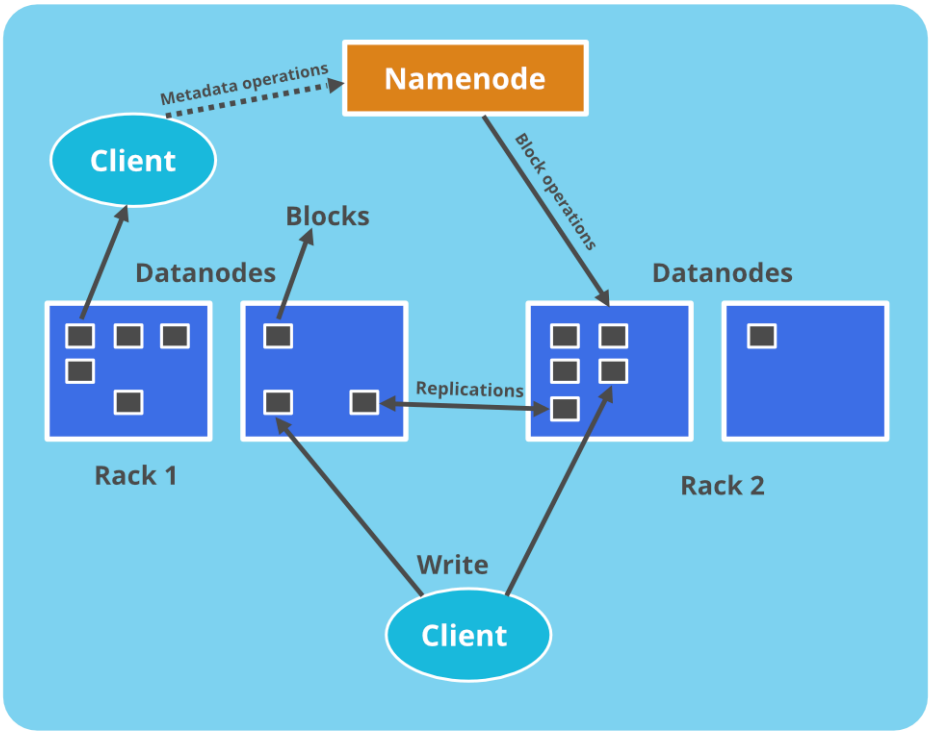
- **Core Components of YARN:**
  - **ResourceManager (Master):** Allocates cluster resources (CPU, memory) to applications.
  - **NodeManagers (Slaves):** Manage resources on individual nodes and report status to the ResourceManager.
  - **ApplicationMaster:** Negotiates resources from the ResourceManager and monitors application execution.
- **Key Features:**
  - **Efficient Resource Management:** Dynamically assigns resources based on application needs.
  - **Cluster Utilization:** Multiple frameworks (MapReduce, Spark, Hive, HBase) can run simultaneously on the same cluster.
  - **Scalability and Flexibility:** New nodes can be added without downtime, and diverse workloads (batch processing, real-time queries, streaming) can be handled effectively.
- **Example:** In a Hadoop cluster running both Spark jobs for analytics and MapReduce jobs for ETL, YARN ensures fair allocation of resources between both without conflict.

How Hadoop Works (Integration of Components)

1. **HDFS** stores large datasets reliably across multiple nodes with replication.
2. **MapReduce** or other engines process the stored data in parallel.
3. **YARN** manages CPU and memory allocation dynamically so that multiple applications can run on the same cluster without resource contention.

Q. Node Failure Handling in Hadoop

=>



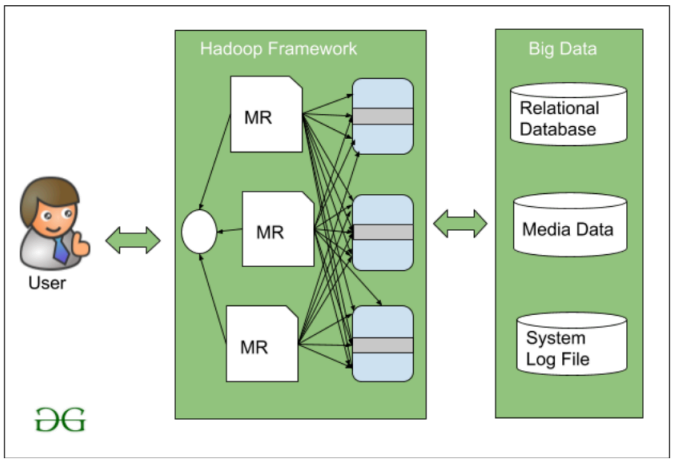
1. **HDFS uses a Master/Slave architecture:**
    - The **NameNode** is the master that stores metadata (file paths, block locations, replication details).
    - **DataNodes** are slaves that store the actual file data as blocks.
2. **Role of NameNode:**
    - It monitors DataNodes by receiving *heartbeat* signals every 3 seconds.
    - If it stops receiving heartbeats for **10 minutes (default)**, it marks the DataNode as **dead**.
3. **Role of DataNodes:**
    - They perform block creation, replication, and deletion as instructed by the NameNode.

- Even if one DataNode fails, the cluster keeps running because data is **replicated** across multiple nodes.

4. **Heartbeat and Block Reports:**
    - Heartbeats confirm the **liveness** of a DataNode.
    - Block reports provide a **complete list of blocks** stored on each DataNode to the NameNode.
5. **What happens on DataNode failure?**
    - If a DataNode is marked dead, its blocks are now **under-replicated**.
    - The NameNode immediately triggers replication of those blocks from healthy DataNodes to maintain the required replication factor (default is 3).
6. **Direct Data Transfer for Replication:**
    - When replication starts, **data moves directly between DataNodes**.
    - The NameNode only gives instructions, it does not carry the actual data, reducing network overhead.
7. **No Impact on Data Availability:**
    - Because of replication, **file data remains accessible** even if a node fails.
    - Users and applications experience no major interruption.
8. **NameNode failure is critical:**
    - If the single active NameNode fails, **the entire cluster becomes inaccessible**.
    - High Availability (HA) configurations or backup NameNodes (Standby NameNode) are used to handle this.
9. **Automatic Recovery:**
    - Hadoop automatically redistributes data blocks to ensure fault tolerance.
    - Administrators do not need to manually intervene in most cases of DataNode failure.

Q. How Big Data Problems Are Handled by the Hadoop System

=>



1. The Challenge of Big Data

Big data is characterized by **Volume, Velocity, Variety, Veracity, and Value**. Traditional systems using **RDBMS on single servers** were unable to cope because:

- They could only store **limited volumes** of data.
- Processing speed was restricted by the **single processor's capability**.
- They could not handle **unstructured or semi-structured data** efficiently.
- Scaling required **expensive, high-end hardware** rather than affordable commodity systems.

2. Google's MapReduce – A New Approach

- Google introduced **MapReduce**, a *master-slave processing model*, where:
    - **Master Node (JobTracker):** Breaks tasks into smaller chunks and assigns them.
    - **Slave Nodes (TaskTrackers):** Process these chunks in parallel.
- After processing, the results from all slave nodes are combined by the master to produce the **final dataset**.
- This distributed approach solved **scalability** and **parallel processing** problems.

3. Hadoop – Open Source Solution

- In 2005, *Doug Cutting* and *Mike Cafarella* created **Hadoop**, an **open-source framework** based on Google's MapReduce and Google File System (GFS) concepts.
- Hadoop **uses clusters of commodity hardware** instead of a single powerful server, making it cost-effective and fault-tolerant.

4. How Hadoop Handles Big Data

(a) Distributed Storage with HDFS:

- **Hadoop Distributed File System (HDFS)** splits huge files into fixed-size blocks (128 MB or 256 MB) and distributes them across multiple nodes.
- Each block is **replicated** (default 3 copies) to ensure fault tolerance. Even if one node fails, data remains accessible.

(b) Parallel Processing with MapReduce:

- Hadoop's MapReduce divides processing tasks across nodes.
- **Map Phase:** Processes input data in parallel, producing key-value pairs.
- **Reduce Phase:** Aggregates these intermediate results into a final output.
- This ensures **fast data computation** even on massive datasets.

(c) Scalability and Cost Efficiency:

- Adding more nodes increases storage and processing power **horizontally**, without needing costly upgrades.
- Uses **low-cost commodity hardware** rather than enterprise-grade machines.

(d) Fault Tolerance:

- If a DataNode fails, Hadoop automatically replicates its blocks on other nodes using **heartbeat and block reports**.
- The system self-heals without human intervention.

5. Example

Suppose a company needs to process **1 petabyte of website logs**:

- HDFS will **store logs across hundreds of nodes**, each holding small replicated blocks.
- MapReduce jobs will **analyze traffic patterns in parallel**, drastically reducing processing time from days to hours.

Q. Name the three ways that resources can be shared between computer systems. Name the architecture used in big data solutions and describe it in detail.

=>

1. Ways resources can be shared between computer systems

In distributed systems, resources can be shared in three main ways:

Way of Sharing	Explanation	Examples
Data Sharing	Multiple systems share and access common datasets, files, or databases. This improves collaboration and ensures data consistency across the organization.	Shared databases, distributed file systems (HDFS).
Computation Sharing	Processing tasks are divided among multiple systems to leverage parallel computing. This increases processing speed and scalability.	MapReduce jobs, distributed computation frameworks like Spark.
Peripheral/Hardware Sharing	Hardware devices or network resources are made accessible to multiple nodes. This avoids duplication and reduces cost.	Shared printers, scanners, or network-attached storage.

2. Architecture used in big data solutions: Hadoop Architecture

Big data solutions most commonly use **Hadoop's master-slave architecture**, which is specifically designed for distributed storage and parallel processing.

Components and Workflow:

1. **HDFS (Hadoop Distributed File System)** – *Storage Layer*
    - **NameNode (Master)**:

        - Manages metadata (file directory tree, block locations, permissions).
        - Does **not** store the actual data, only the information about where it resides.
    - **DataNodes (Slaves):**
        - Store the actual data blocks.
        - Perform read/write operations as directed by the NameNode.
    - **Features:** Fault tolerance through replication (default factor = 3), scalability by adding more nodes.
2. **MapReduce (Processing Layer)** – *Computation Engine*
    - **JobTracker (Master):**
        - Schedules tasks, assigns them to TaskTrackers, monitors progress.
    - **TaskTrackers (Slaves):**
        - Perform the actual computation on data stored locally.
    - **Map Phase:** Input data is split into chunks and processed in parallel.
    - **Reduce Phase:** Results are aggregated into final output.
3. **YARN (Yet Another Resource Negotiator)** – *Resource Management Layer*
    - Allocates cluster resources dynamically.
    - Separates resource management from job scheduling for improved scalability.

How this architecture supports big data:

- **Scalability:** Can grow by adding inexpensive commodity machines.
- **Fault Tolerance:** Automatic replication ensures data is not lost even if nodes fail.
- **High Throughput:** Parallel processing of data chunks reduces execution time.
- **Cost Efficiency:** Uses low-cost hardware rather than specialized servers.
- **Data Locality:** Computation is sent to where data resides, minimizing network overhead.