**Why Hadoop Tools Are Required?**

Hadoop tools are required because Big Data cannot be efficiently handled by traditional databases.

Big Data is:

* Huge in volume (TBs, PBs of data).
* Generated fast (velocity – streaming, logs, IoT).
* Variety (structured, semi-structured, unstructured).
* Needs scalability & fault tolerance.

**Hadoop tools solve these by:**

1. HDFS (Hadoop Distributed File System) – stores large datasets reliably across multiple machines.

2. YARN – resource management & scheduling for distributed tasks.

3. MapReduce / Spark – parallel processing of data.

4. Ecosystem Tools – Hive (SQL-like queries), Pig, Sqoop, HBase, etc.

Without Hadoop tools, handling scalability, parallelism, and fault tolerance would be extremely complex.

**Hadoop Versions**

* **Hadoop 1.x** → Only **MapReduce** for computation, single NameNode (single point of failure), less scalability.
* **Hadoop 2.x** → Introduced **YARN** (resource manager, multi-framework support), multiple NameNodes (HA), better scalability.
* **Hadoop 3.x** → More efficient (erasure coding to save storage), support for containers (Docker), multiple standby NameNodes, improved fault tolerance, cloud integration.

**Ways to Download/Install Hadoop (Free)**

1. **Binary distributions** (from Apache).
2. **Pre-built vendor distributions** (Cloudera, Hortonworks/HDInsight, MapR).
3. **Cloud-based services** (AWS EMR, Google Dataproc, Azure HDInsight – pay for infra, software free).
4. **Sandbox/VM images** (Hortonworks Sandbox, Cloudera QuickStart VM).
5. **Docker containers** (community-built Hadoop Docker images).

**How HDFS Works in Hadoop?**

HDFS is the storage layer of Hadoop. It stores huge files across multiple machines.

Working:

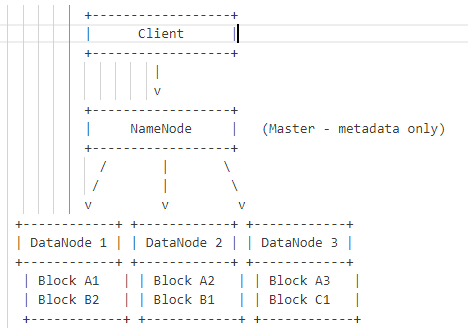
1. File Split – Large files are divided into blocks (default 128 MB or 256 MB).

2. Replication – Each block is replicated (default 3 copies) across different DataNodes to ensure fault tolerance.

3. NameNode – Stores metadata (which block is stored on which DataNode).

4. DataNode – Actually stores the file blocks.

5. Client – Requests file from HDFS → contacts NameNode → gets block locations → fetches blocks from DataNodes.



* File is split into blocks (A, B, C, …).
* Blocks are stored on different DataNodes with replication.
* NameNode knows only metadata, not the actual data.
* Client fetches data directly from DataNodes using block info from NameNode.

**YARN (Yet Another Resource Negotiator)**

HDFS stores data, but we need a system to manage resources (CPU, RAM) and schedule jobs across a Hadoop cluster. That’s YARN.

How YARN works:

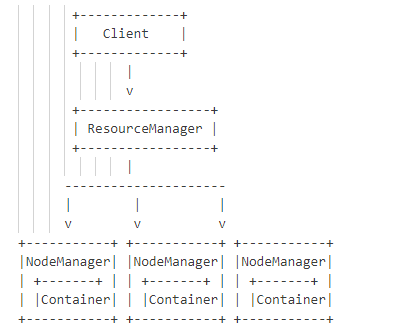
1. Client submits a job.

2. ResourceManager (Master) – decides which nodes will run the job (manages cluster resources).

3. NodeManager (Slave) – runs on each DataNode, manages resources (CPU, memory) on that node.

4. ApplicationMaster – launched per job, negotiates resources from ResourceManager and coordinates tasks on NodeManagers.

5. Containers – isolated environments where tasks actually run.



*YARN is the operating system of Hadoop, ensuring efficient resource allocation & scheduling.*

**MapReduce**

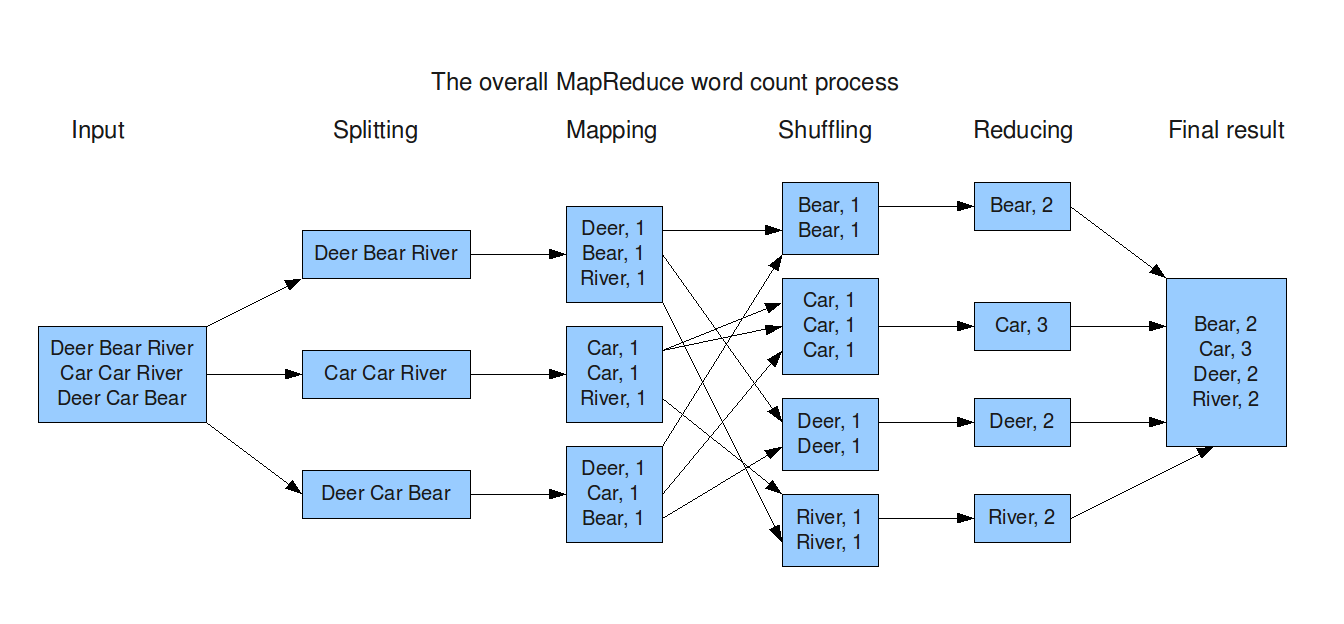
It is the processing engine of Hadoop. While HDFS stores data and YARN manages resources, MapReduce processes data in parallel.

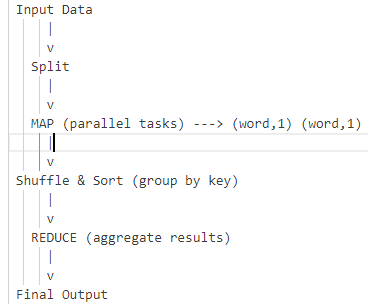
How MapReduce works:

1. Map Phase – Input data is split into key-value pairs and processed in parallel.

2. Shuffle & Sort – Intermediate data is grouped by keys.

3. Reduce Phase – Aggregates or combines results into final output.





*MapReduce = Batch processing framework for large-scale data.*

**Spark**

MapReduce is slow because it writes intermediate results to disk after every phase. Spark was introduced for in-memory, fast processing with support for real-time and batch jobs.

How Spark works:

1. Driver Program – main program that defines transformations & actions.

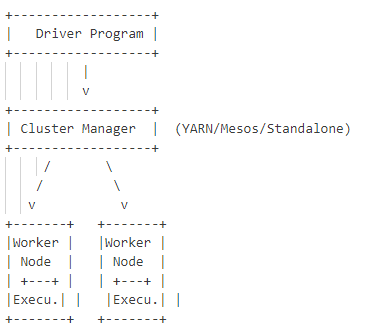
2. Cluster Manager (can be YARN, Mesos, or Spark’s standalone) – allocates resources.

3. Executors – run on worker nodes, perform tasks.

4. RDD (Resilient Distributed Dataset) – core data structure; data is kept in-memory whenever possible.

**Features vs MapReduce:**

* Much faster (100x in-memory, 10x on disk).
* Supports batch + streaming + ML + graph.
* APIs in Python, Scala, Java, R.



*Spark = Fast, general-purpose big data processing engine (better than MapReduce for most cases).*

**Spark Versions Difference**

* **Spark 1.x** → Basic RDD API, less optimized, slower execution.
* **Spark 2.x** → Introduced **DataFrames & Datasets**, Catalyst optimizer, Structured Streaming.
* **Spark 3.x** → Enhanced **pandas API on Spark**, better performance (Adaptive Query Execution), GPU/AI integration, Kubernetes support.

**MapReduce vs Spark/PySpark**

* **MapReduce** → Disk-based, slower (writes intermediate results to HDFS).
* **Spark** → In-memory computation, much faster, supports batch + streaming + ML + graph.
* **PySpark** → Python API for Spark (easier coding for Python users).

**MapReduce** → Written in **Java** (APIs also in Python, Ruby, C++).

**Spark** → Written in **Scala** (APIs in Python = PySpark, Java, R).

**ETL with Hadoop**

* **E (Extract):** Get raw data from multiple sources (databases, logs, CSVs, JSON, IoT streams). Put it into **HDFS** using tools like **Sqoop, Flume, Kafka, or direct upload**.
* **T (Transform):** Clean, filter, and process data using **MapReduce, Hive, Pig, or Spark** (e.g., converting formats, aggregating sales totals, removing duplicates).
* **L (Load):** Store the transformed data back into **HDFS**, **Hive tables**, or load into external systems (like MySQL, data warehouse, dashboards).

**HDFS stores → MapReduce/Spark transforms → Hive/DB loads for analytics.**

**Prerequisites for Hadoop**

* Linux basics (commands, file system).
* Java (JDK installed).
* SSH setup (for cluster nodes).
* Basic understanding of distributed systems.