**DAY 7 (13-06-25)**

**BIG DATA**

**What is Big Data?**

**Big Data** refers to extremely large and complex datasets that cannot be managed, processed, or analyzed using traditional data processing tools and techniques.

A close-up of a white board

AI-generated content may be incorrect.

**The Evolution of Big Data: Past, Present, and Future**

Although the concept of big data is relatively new, the need to manage large data sets dates back to the 1960s and ’70s, with the first data centers and the development of the relational database.

**Past.** Around 2005, people began to realize just how much data users generated through Facebook, YouTube, and other online services. Apache Hadoop, an open source framework created specifically to store and analyze big data sets, was developed that same year. NoSQL also began to gain popularity during this time.

**Present.** The development of open source frameworks, such as Apache Hadoop and more recently, Apache Spark, was essential for the growth of big data because they make big data easier to work with and cheaper to store. In the years since then, the volume of big data has skyrocketed. Users are still generating huge amounts of data—but it’s not just humans who are doing it.

With the advent of the Internet of Things (IoT), more objects and devices are connected to the internet, gathering data on customer usage patterns and product performance. The emergence of [machine learning](https://www.oracle.com/in/artificial-intelligence/machine-learning/what-is-machine-learning/) has produced still more data.

**Future.** While big data has come far, its value is only growing as [generative AI](https://www.oracle.com/in/artificial-intelligence/generative-ai/what-is-generative-ai/) and cloud computing use expand in enterprises. The cloud offers truly elastic scalability, where developers can simply spin up ad hoc clusters to test a subset of data. And [graph databases](https://www.oracle.com/in/autonomous-database/what-is-graph-database/) are becoming increasingly important as well, with their ability to display massive amounts of data in a way that makes analytics fast and comprehensive.

Key Characteristics: The **5 V's** of Big Data

| **V** | **Description** |
| --- | --- |
| **Volume** | Huge amount of data (TBs to ZBs) |
| **Velocity** | Data is generated at high speed (e.g., social media, sensors) |
| **Variety** | Different formats – structured (tables), semi-structured (JSON), unstructured (videos, images) |
| **Veracity** | Trustworthiness and quality of data |
| **Value** | Extracting useful insights and business value from data |

A poster of a data presentation

AI-generated content may be incorrect.

A whiteboard with red writing

AI-generated content may be incorrect.

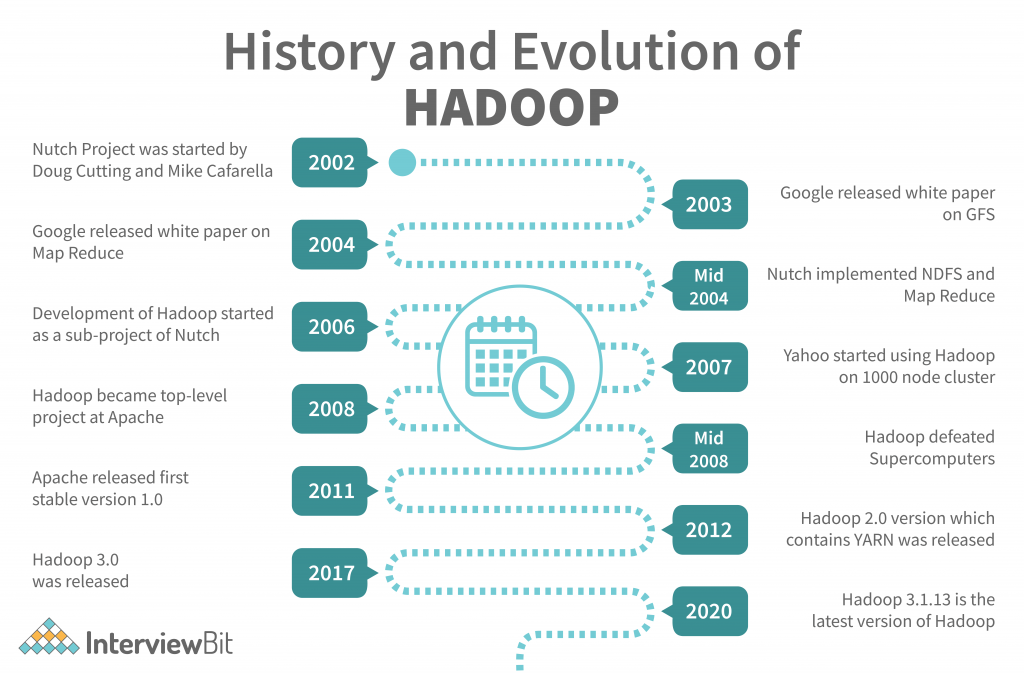
How does big data work?

The central concept of big data is that the more visibility you have into anything, the more effectively you can gain insights to make better decisions, uncover growth opportunities, and improve your business model.

Making big data work requires three main actions:

* **Integration:**Big data collects terabytes, and sometimes even petabytes, of raw data from many sources that must be received, processed, and transformed into the format that business users and analysts need to start analyzing it.
* **Management:**Big data needs big storage, whether in the cloud, on-premises, or both. Data must also be stored in whatever form required. It also needs to be processed and made available in real time. Increasingly, companies are turning to cloud solutions to take advantage of the unlimited compute and scalability.
* **Analysis:**The final step is analyzing and acting on big data—otherwise, the investment won’t be worth it. Beyond exploring the data itself, it’s also critical to communicate and share insights across the business in a way that everyone can understand. This includes using tools to create data visualizations like charts, graphs, and dashboards.

What is **Hadoop**?



**Hadoop** is an **open-source** framework developed by **Apache** that allows for the **storage** and **processing** of **large datasets** across clusters of computers using simple programming models.

Think of Hadoop as a system that helps you **store and analyze big data** using **multiple machines** working together.

**Core Components of Hadoop:**

**1. HDFS (Hadoop Distributed File System)**

* Stores data by **splitting it into blocks** and **distributing** it across multiple machines.
* Provides **fault-tolerance** and **high throughput**.

**2. MapReduce**

* The **processing engine** of Hadoop.
* Works in two steps:
  + **Map**: Splits task into smaller sub-tasks
  + **Reduce**: Combines sub-task results into final output

**3. YARN (Yet Another Resource Negotiator)**

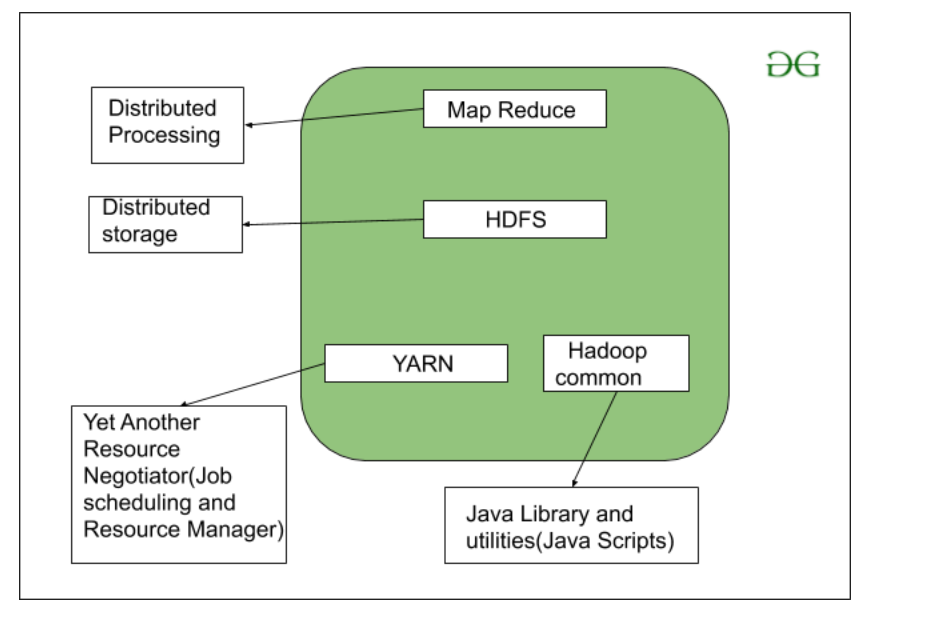
* Manages **resources** and **scheduling** tasks across the cluster.
* Makes sure all machines are used efficiently.

**4. Common Utilities**

* Provides libraries and Java files needed by the other components.

Key Features of Hadoop:

| **Feature** | **Description** |
| --- | --- |
| **Open Source** | Free to use and modify |
| **Scalable** | Easily add more machines |
| **Fault Tolerant** | Automatically handles failures |
| **Distributed** | Stores and processes data across multiple nodes |
| **High Throughput** | Ideal for processing large volumes of data |



Hadoop Architecture

Components of Hadoop:

**1. HDFS – Hadoop Distributed File System**

**Purpose:** Storage

* Breaks large files into **blocks** (usually 128 MB or 256 MB)
* Stores them across **multiple nodes**
* Ensures **fault tolerance** by making **copies (replicas)** of each block (default = 3)
* Has two parts:
  + **NameNode** – Manages metadata (file names, block locations)
  + **DataNodes** – Store the actual data blocks

**Think of HDFS as a smart, reliable, distributed file storage system**

**2. MapReduce – Data Processing Engine**

**Purpose:** Processing

* A programming model to process data in **two phases**:
  1. **Map**: Splits large data into smaller chunks and processes them
  2. **Reduce**: Aggregates and combines results

Example: Counting word frequency from a book

* **Map** phase: Split book into paragraphs and count words in each
* **Reduce** phase: Add counts from all paragraphs to get the final result

**3. YARN – Yet Another Resource Negotiator**

**Purpose:** Resource Management

* Allocates system resources (CPU, memory, etc.) to various tasks
* Schedules jobs across the cluster
* Helps multiple applications run simultaneously

**YARN is the "manager" or "coordinator" of all tasks running in the Hadoop system**

**4. Hadoop Common – Shared Utilities**

**Purpose:** Support

* Provides **Java libraries**, **APIs**, and other utilities needed by HDFS, MapReduce, and YARN
* Includes basic tools, I/O functions, and configuration files

**It’s like the “toolkit” used by all other components**

**Summary:**

| **Component** | **Role** | **Description** |
| --- | --- | --- |
| **HDFS** | Storage system | Stores data in distributed blocks |
| **MapReduce** | Processing engine | Processes data using map and reduce |
| **YARN** | Resource manager | Allocates and schedules tasks |
| **Common** | Shared tools/libraries | Provides utilities and configs |

Hadoop Distributed File System (HDFS):

**What is HDFS?**

**HDFS** stands for **Hadoop Distributed File System**.  
It is the **storage system** used by **Hadoop** to store large files across multiple machines (called a **cluster**) in a **fault-tolerant** and **scalable** way.

**Why HDFS?**

Imagine you have a 500 GB file. A single machine might be too slow or may not have enough space.  
**HDFS splits the file into smaller blocks and stores them across many machines** so they can be processed faster and stored easily.

**Basic Architecture**

**1. NameNode (Master)**

* Stores **metadata** – file names, locations, permissions.
* Knows **which block is stored on which machine**.

**2. DataNodes (Workers)**

* Actually store the **data blocks**.
* Send heartbeat to NameNode to show they are active.

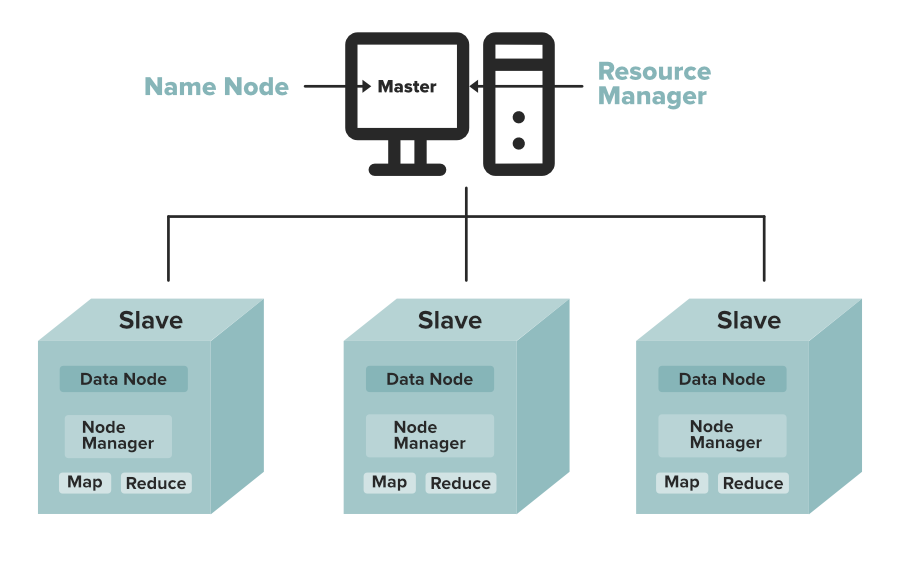
**3. Blocks**

* Files are split into fixed-size blocks (default is **128 MB**).
* Blocks are **replicated** (usually 3 copies) for **fault tolerance**.

Architecture of HDFS is below,

A diagram of a data flow

AI-generated content may be incorrect.



HDFS vs Traditional File System:

| **Feature** | **HDFS** | **Traditional FS** |
| --- | --- | --- |
| Storage | Distributed (across nodes) | Centralized (single disk) |
| Fault Tolerance | Yes (replication) | Limited |
| Suitable For | Big files, batch processing | Small files, random access |

**Data Replication**

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

A screenshot of a computer

AI-generated content may be incorrect.

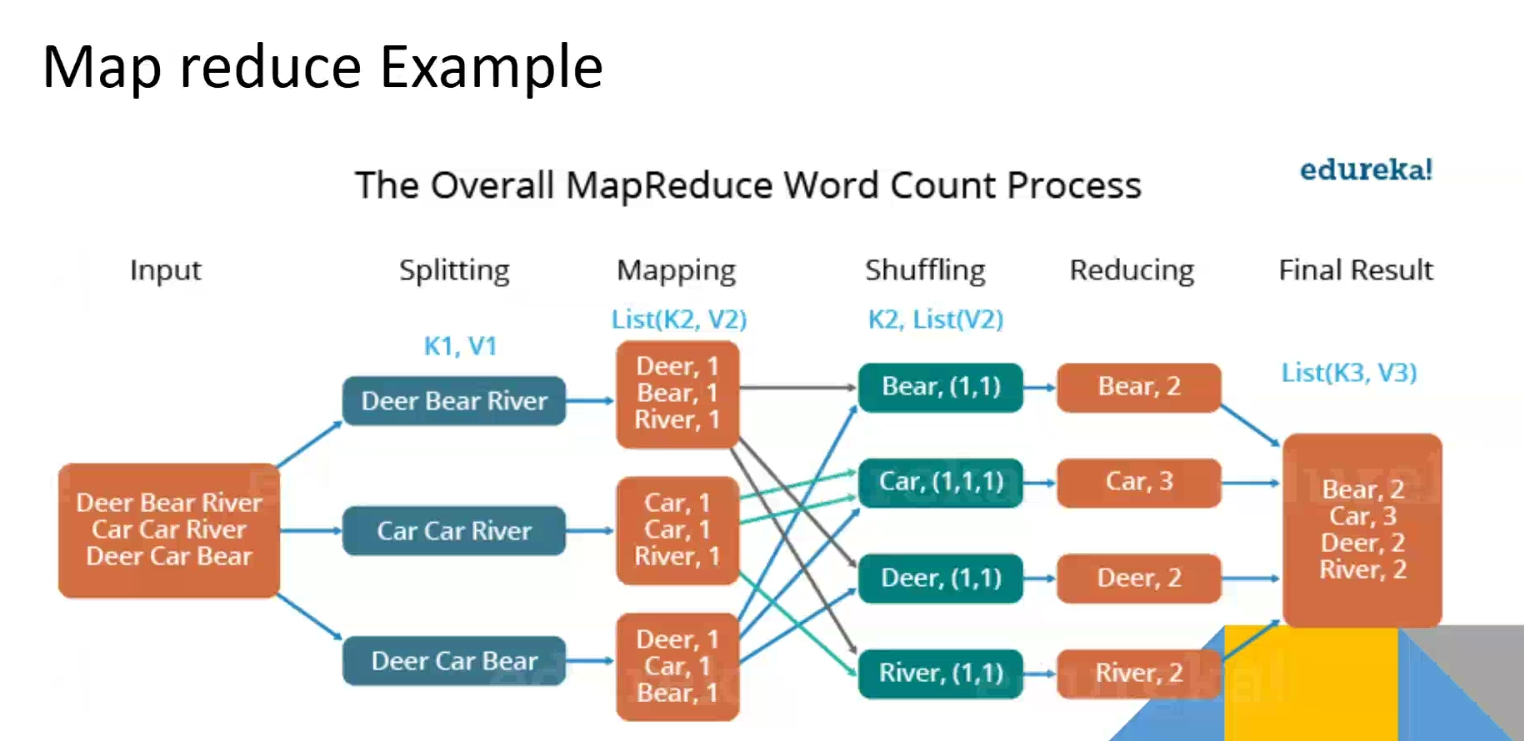
Big Data & Hadoop Ecosystem – Explained Simply and Technically:

| **Term** | **Simple Explanation** | **Technical Definition** |
| --- | --- | --- |
| **Big Data** | Huge amount of data that is hard to manage using normal tools | Large, complex datasets that traditional data processing software cannot handle efficiently; defined by 5 Vs – Volume, Velocity, Variety, Veracity, and Value |
| **Hadoop** | A tool that helps store and process Big Data using many computers | An open-source framework by Apache that allows for distributed storage (via HDFS) and processing (via MapReduce) of large datasets |
| **HDFS** | A system that stores big files by splitting and saving them across multiple computers | Hadoop Distributed File System – a scalable and fault-tolerant file storage system that divides data into blocks and stores them across nodes in a cluster |
| **MapReduce** | A way to process large data in two steps: breaking and combining | A programming model in Hadoop that processes data in two phases: Map (filter/sort) and Reduce (aggregate/summarize) |
| **YARN** | A manager that assigns work to computers in the Hadoop system | Yet Another Resource Negotiator – Hadoop’s resource management layer that handles scheduling and cluster resource allocation |
| **Hadoop Common** | Basic tools used by all parts of Hadoop | A collection of shared utilities, libraries, and APIs needed by other Hadoop modules to function |

Simple Summary Table:

| **Term** | **Simple Meaning** |
| --- | --- |
| **Big Data** | Huge and complex data |
| **Hadoop** | Tool to store and process Big Data |
| **HDFS** | Hadoop’s way to store data across machines |
| **MapReduce** | Method to process data in steps |
| **YARN** | Manages resources and tasks |
| **Common** | Tools that support everything in Hadoop |

Map Reduce:



**MapReduce** is a **programming model** and **processing technique** introduced by **Google** and later implemented in **Apache Hadoop**.  
It is designed to **process large volumes of data in parallel** by dividing the work across a cluster of computers.

MapReduce works in two main steps:

* **Map Phase** → Break the data into small key-value pairs and perform filtering or transformation.
* **Reduce Phase** → Aggregate or summarize those results based on keys.

It is ideal for batch processing of massive data (like log files, social media data, sensor data, etc.).

**How It Works – Step-by-Step**

Let’s walk through the full MapReduce workflow:

**1. Splitting**

Hadoop divides the input data into **splits** (usually based on block size – e.g., 128 MB). Each split is sent to a different **Mapper**.

**2. Mapping (Map Phase)**

Each **Mapper** processes its split.  
It takes a line or record and transforms it into **key-value pairs**.

* Example:  
  Input: "Big data is big"
* Output:
  + (Big, 1)
  + (data, 1)
  + (is, 1)
  + (big, 1)

Note: The words are case-insensitive, so "Big" and "big" might be normalized before processing.

**3. Shuffling and Sorting**

After mapping, Hadoop groups and sorts the intermediate data **by key** (e.g., all "big" values together).  
It then sends the grouped key and list of values to **Reducers**.

Example:

* (big, [1, 1])
* (data, [1])
* (is, [1])

**4. Reducing (Reduce Phase)**

The **Reducer** takes each key and applies an **aggregation function** (like sum, max, min, etc.).

* Output:
  + (big, 2)
  + (data, 1)
  + (is, 1)

**5. Output**

The result is written to HDFS in output files (e.g., part-0000).

**Advantages of MapReduce**

* **Scalable**: Works well on huge datasets across hundreds or thousands of machines.
* **Fault-tolerant**: Failed tasks are automatically retried.
* **Cost-effective**: Uses commodity hardware.
* **Parallelism**: Breaks work into independent tasks.

What is YARN?

* **YARN (Yet Another Resource Negotiator)** is the **resource management and job scheduling layer** of Hadoop.  
  It allows **multiple applications** (like MapReduce, Spark, Hive, etc.) to run on a Hadoop cluster by **managing resources** (CPU, memory) and **allocating tasks** dynamically.
* YARN separates **job processing** and **resource management**, which makes Hadoop more **scalable** and **efficient**.
* **YARN** is like the **manager or boss** in the Hadoop system.
* It decides **which computer** will run **which task** and **when**.

**Why YARN?**

Before YARN (in Hadoop v1), MapReduce was responsible for both:

* Processing jobs
* Managing cluster resources

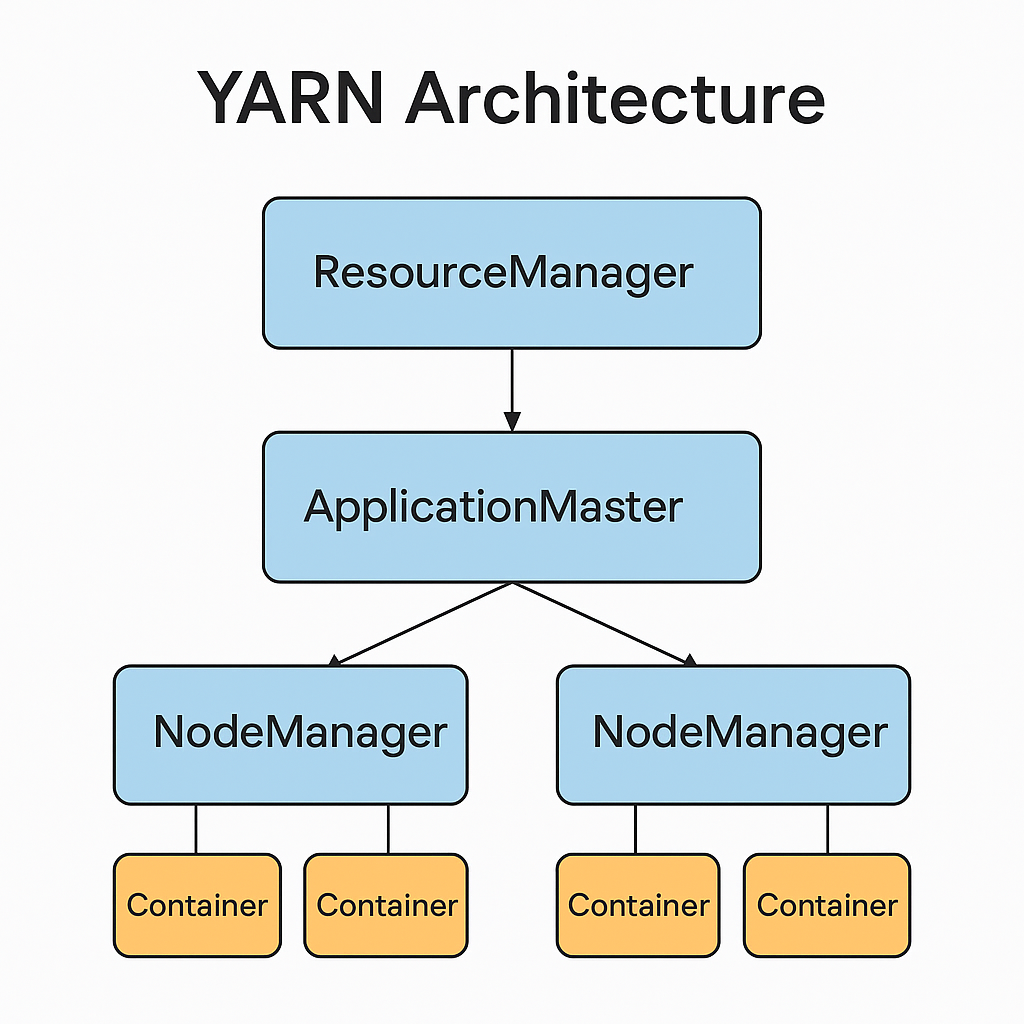
This was limiting, because only MapReduce jobs could run.

With YARN (Hadoop v2+):

* **Resource Management** is done by YARN
* **Job Processing** can be done by **any engine** (MapReduce, Spark, etc.)

**YARN Architecture Components**

| **Component** | **Description** |
| --- | --- |
| **ResourceManager (RM)** | The **main master**. It manages resources across the whole cluster. |
| **NodeManager (NM)** | A **worker** on each machine. Reports to the RM and manages containers on that machine. |
| **ApplicationMaster (AM)** | Controls **one running job**. It talks to RM to get resources and to NM to run tasks. |
| **Containers** | Units where actual tasks (like Map or Reduce) run. Each has its own memory and CPU. |



**How YARN Works:**

1. **User submits a job** (e.g., a MapReduce task)
2. **ResourceManager** creates a **Container** to start an **ApplicationMaster**
3. **ApplicationMaster** requests more **Containers** to run Map or Reduce tasks
4. **NodeManagers** launch those tasks in their local **Containers**
5. Once done, the **ApplicationMaster shuts down**

**Key Features of YARN:**

| **Feature** | **Explanation** |
| --- | --- |
| **Multi-engine support** | Runs MapReduce, Spark, Tez, etc. |
| **Efficient resource use** | Avoids idle nodes, improves cluster usage |
| **Fault-tolerant** | Restarts failed components |
| **Scalable** | Supports thousands of nodes |
| **Flexible** | Works for batch, real-time, streaming apps |

**YARN Summary:**

| **Term** | **Simple Meaning** | **Technical Definition** |
| --- | --- | --- |
| **YARN** | Boss of the Hadoop system | A resource manager and job scheduler in Hadoop |
| **ResourceManager** | Master that tracks all work | Allocates resources and monitors node status |
| **NodeManager** | Worker that does the job | Manages tasks on a single machine |
| **ApplicationMaster** | Controls one running job | Manages execution of a single application |
| **Container** | Space where tasks run | A resource slot (CPU + memory) to run a task |

YARN Application Types:

YARN supports many kinds of applications, not just MapReduce:

| **Engine** | **Purpose** |
| --- | --- |
| **MapReduce** | Batch processing |
| **Apache Spark** | In-memory fast computation |
| **Apache Tez** | DAG-based jobs (used in Hive) |
| **Apache Flink / Storm** | Real-time streaming |

**YARN Resource Scheduling Policies:**

| **Policy** | **Description** | **Use Case** |
| --- | --- | --- |
| **FIFO** | First-In-First-Out | Simple, but not resource-balanced |
| **Capacity** | Resources divided among queues (departments, teams) | Large orgs needing quotas |
| **Fair** | Distributes resources so all jobs get fair share | Multi-user clusters |