Chapter 1 : Big Data

→ Data Types

1) Structured :

- highly organized (rows / columns)
- Database format

2) Unstructured

- No predified structure
- emails, images, videos, audio, text ...

3) Semi - Structured

- Some organization, flexible format
- XML, JSON

→ Big Data

- Def : Data sets that are so large and complex, that traditional data processing applications can't deal with them

Characteristics (4 Vs)

1) Volume (immense amount of data generated daily)

2) Variety (different types of data)

3) Velocity (rapid rate at which data is generated)

4) Veracity (uncertainty, incompleteness of data)

Challenges:

Traditional systems struggle with big data because of its size, complexity and speed

=> This bring us to Hadoop as a primary big data solution

→ Hadoop

. Def: Open-Source Software Framework for storing data and running applications on clusters of commodity hardware

. Why it's important?

1) Ability to store and process huge amounts of any kind of data, quickly

2). Uses a distributed computing model (multiple computers working togeth) to process big data fast
. (more computing nodes = more processing power)

3) Fault tolerance
. Protects data and processing against hardware failures
. If a node fails, jobs are redirected to other nodes, and multiple copies of all data are stored automatically

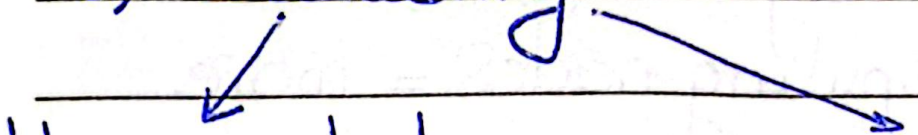
4) Flexibility

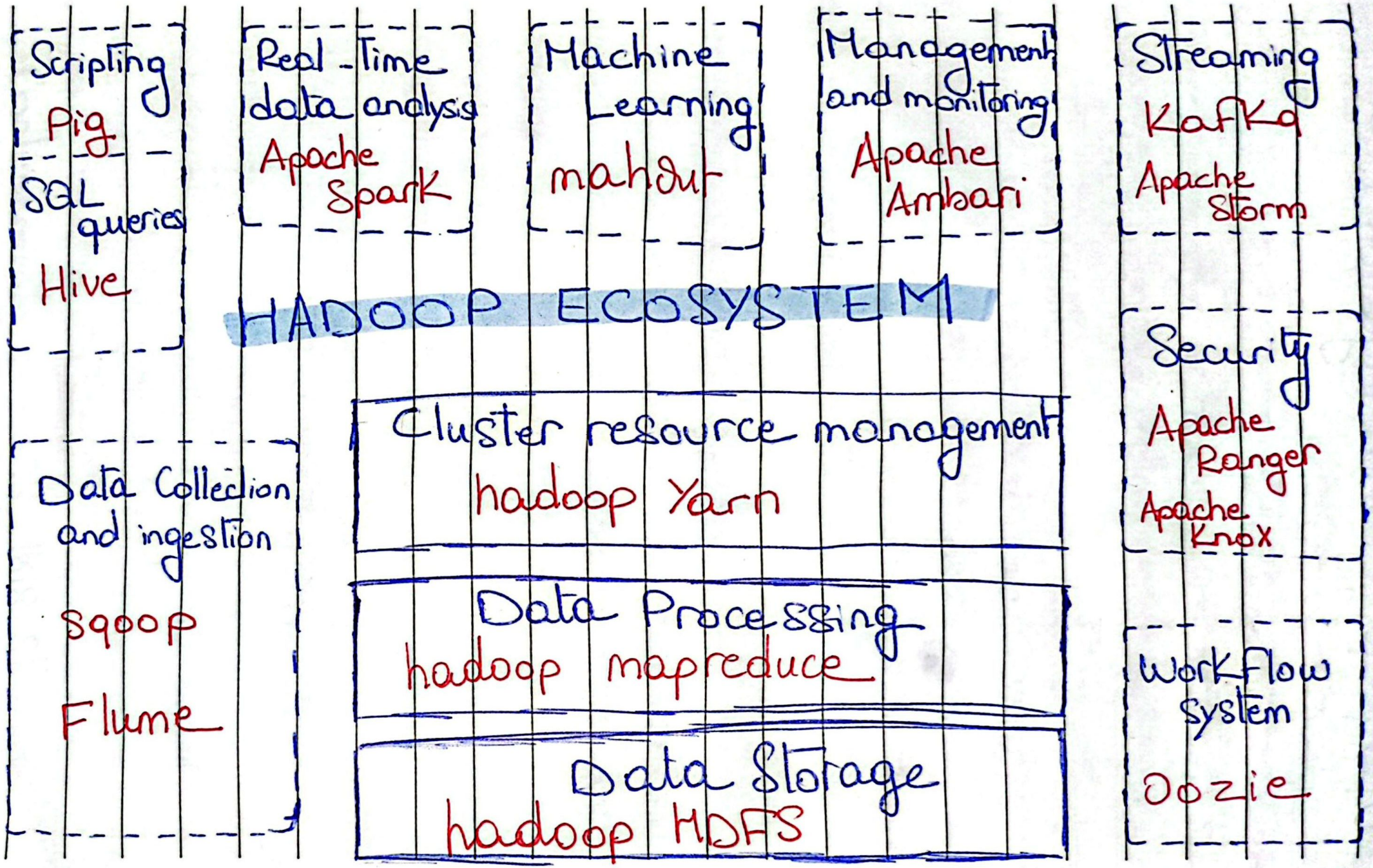
- Doesn't require data preprocessing before storage
- Can store any kind of data

5) Low Cost

- Open-Source (Free)
- uses inexpensive commodity hardware to store/process large amount of data

6) Scalability




Scripting
Pig
SQL queries
Hive
Real-Time data analysis
Apache Spark
Machine Learning
mahout
Management and monitoring
Apache Ambari
Streaming
Kafka
Apache Storm
HADOOP ECOSYSTEM
Security
Apache Ranger
Apache Knox
Cluster resource management
hadoop Yarn
Data Collection and ingestion
Sqoop
Flume
Data Processing
hadoop mapreduce
Data Storage
hadoop HDFS
Work Flow System
Oozie

→ Layers of the Hadoop Ecosystem

Layer	What it does	Examples
Storage	Save the data	HDFS
Processing	Analyze the data	MapReduce, Spark
Query	Ask questions to data	Hive, Impala
Ingestion	Bring data into Hadoop	Sqoop (From db) Storm (real-time)
Management	Monitor and Control	Ambari (dashboa

1) Sqoop (Data Ingestion)

- Helps us move data between traditional databases (MySQL, Oracle, etc) and Hadoop
- Import data from db → Hadoop
- Export results from Hadoop → db

2) Apache Storm (Data Ingestion)

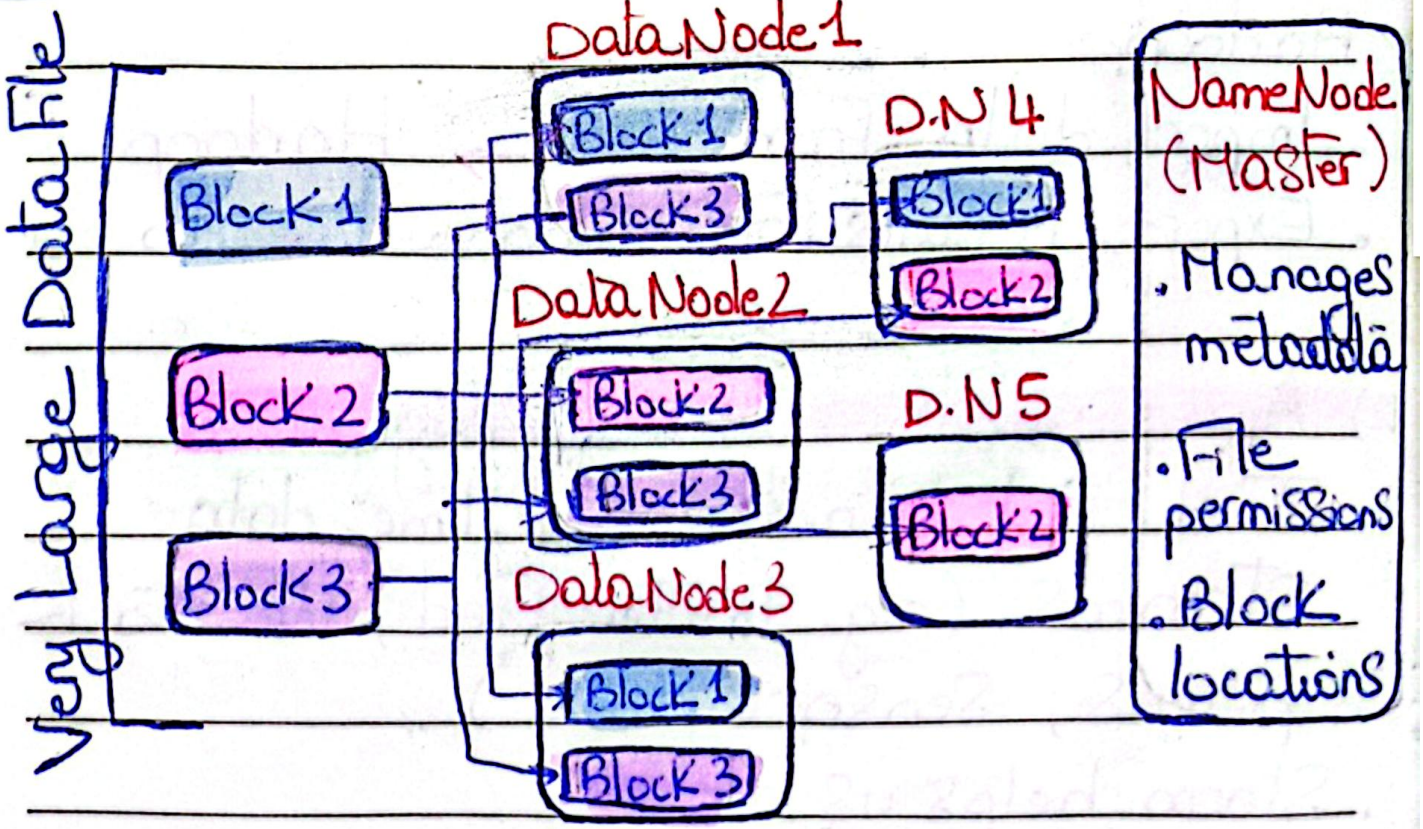
- Tool that handles real-time data streams (e.g. Twitter Feed, live stock prices, Sensors data)
- Storm helps us:
  - → Continuously process this stream
  - → Do analysis as the data arrives

3) Hadoop Distributed File System (HDFS) (Data Storage)

→ What is HDFS

- Based on Google File System (GFS)
- Store large files across multiple machines in a distributed manner
- Key characteristics
  - Stores big data (petabytes)
  - data streaming access (write once, read many)
  - Runs on commodity hardware
  - Highly Fault tolerant

→ Core Architecture Components



1) NameNode (Master)

↳ Centerpiece of HDFS

↳ Functions :

- Executes the file system namespace operations (open, close, rename)
- Manages and maintains the DataNodes
- Determines the mapping of blocks of a File to DataNodes
- Records each change made to the file system
- Keeps the locations of each block of a file
- Takes care of the replication Factor of all the blocks
- Receives hearbeat and blocks from DataNodes that ensure DataNode is alive
- If the DataNode fails, the NameNode chooses new DataNodes for new replicas

↳ Storage : Keeps metadata on local disk in Two Files :

- Edit Log : Recent changes to File system
- FSImage : File system state

2) Data Nodes (Slaves)

↳ Worker nodes that store actual data

↳ Functions:

- Serve client read/write requests
- Perform block creation, replication and deletion (based on NameNode instruction)
- Send heartbeat to NameNode
- Send block report to NameNode

3) Backup Node

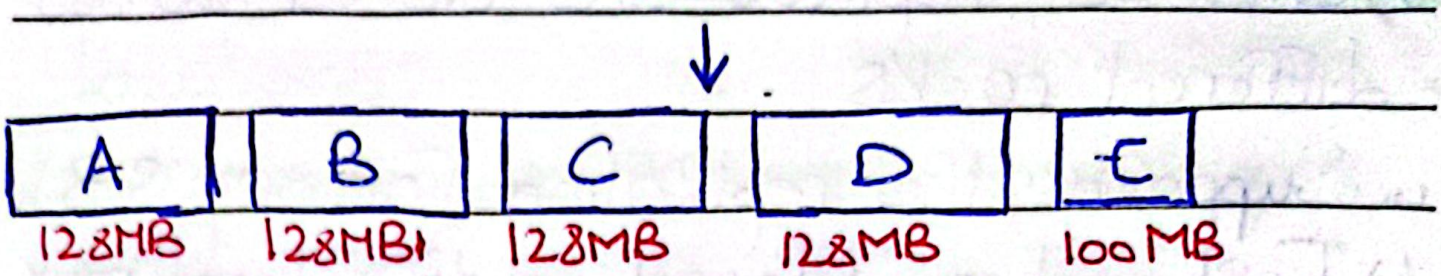
↳ Keeps an in-memory, up-to-date copy of the file system namespace.

→ How Data is Stored

1) Block Structure

- Files are split into 128MB blocks (default, configurable)
- Small files don't occupy full block space
- Users have no control over block location

Example.txt
612 MB



2) Replication Strategy

- Data must be redundant to multiple places so that if one machine fails, the data is accessible from other machines
- Default replication factor: 3 copies of each block
- Storage Calculation:

File size x replication factor = Total storage used

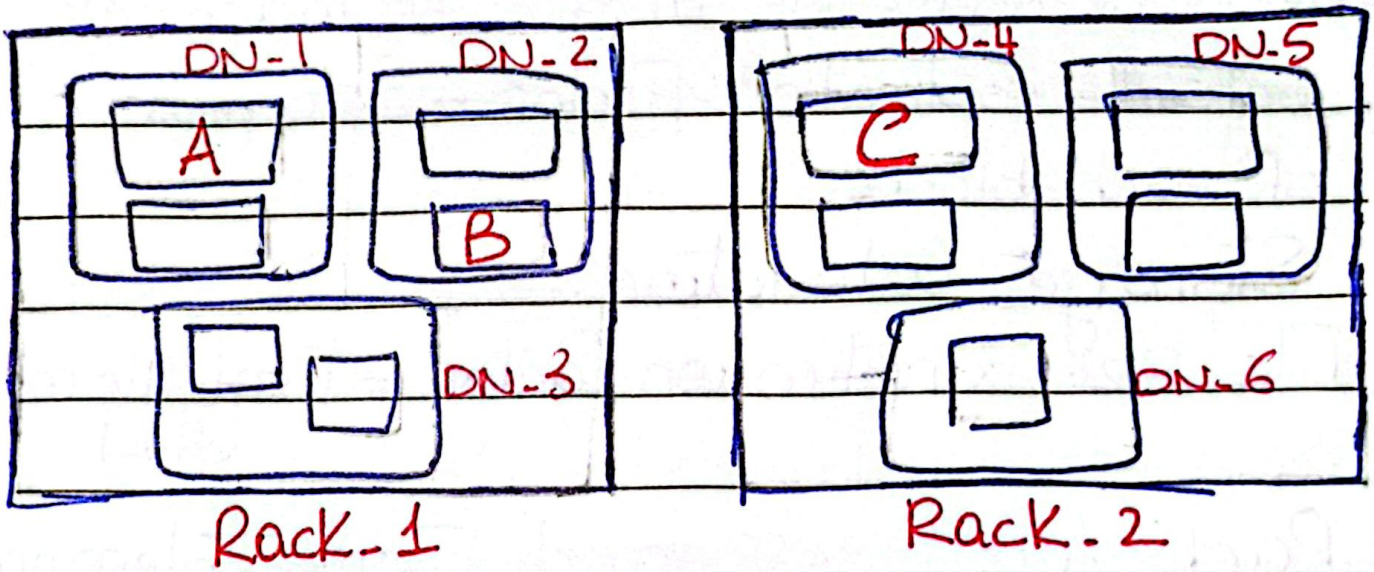
→ Rack Awareness and Fault Tolerance

- Rack: Collection of 40-50 machines connected via network switch
- Risk: If network switch fails, entire rack becomes unavailable

Rack Awareness Algorithm

- Ensures replicas are stored across different racks
- Suppose the replication factor is 3:

A) First replica: stored on the local rack
B) Second replica: on another DataNode in the same rack
C) Third replica: on a different rack



→ Client Interactive Flow

1) Writing Files

- Client contact NameNode for metadata
- NameNode responds with block locations and details
- Client directly interacts with DataNod to store data

2) Reading Files

- Client queries NameNode for block locations
- NameNode provides DataNode addresses
- Client directly reads from DataNodes

→ Access Methods

1) FS Shell Command Line: hadoop fs commands
2) Java API: Programmatic access
3) Ecosystem projects: tools like Hive, Pig, etc.

→ NameNode Availability Solutions

- Problem: If NameNode stops → entire cluster becomes inaccessible
- Solutions:
  1) Backup NameNode
     ↳ Issue: Data loss between backups
  2) Standby NameNode
     ↳ periodic metadata updates
     ↳ issue: Data loss between updates
  3) Secondary NameNode (Best)
     ↳ Continuous metadata update

b) MapReduce (Data Processing)

→ What is MapReduce?

- A method for distributing a task across multiple nodes in a Hadoop cluster
- Each node processes data stored locally on that node

→ Key Terminology

- Job: A complete execution of Mappers and Reducers over a dataset (application in MapReduce 2)
- Task: The execution of a single Mapper or Reducer over a slice of data

→ MapReduce Workflow

Input (raw data) → Split (data division) → Map (Process & Transform)

↓

Shuffle (Sort and group) → Reduce (Aggregate) → Output

→ Core Components

1) The Mapper

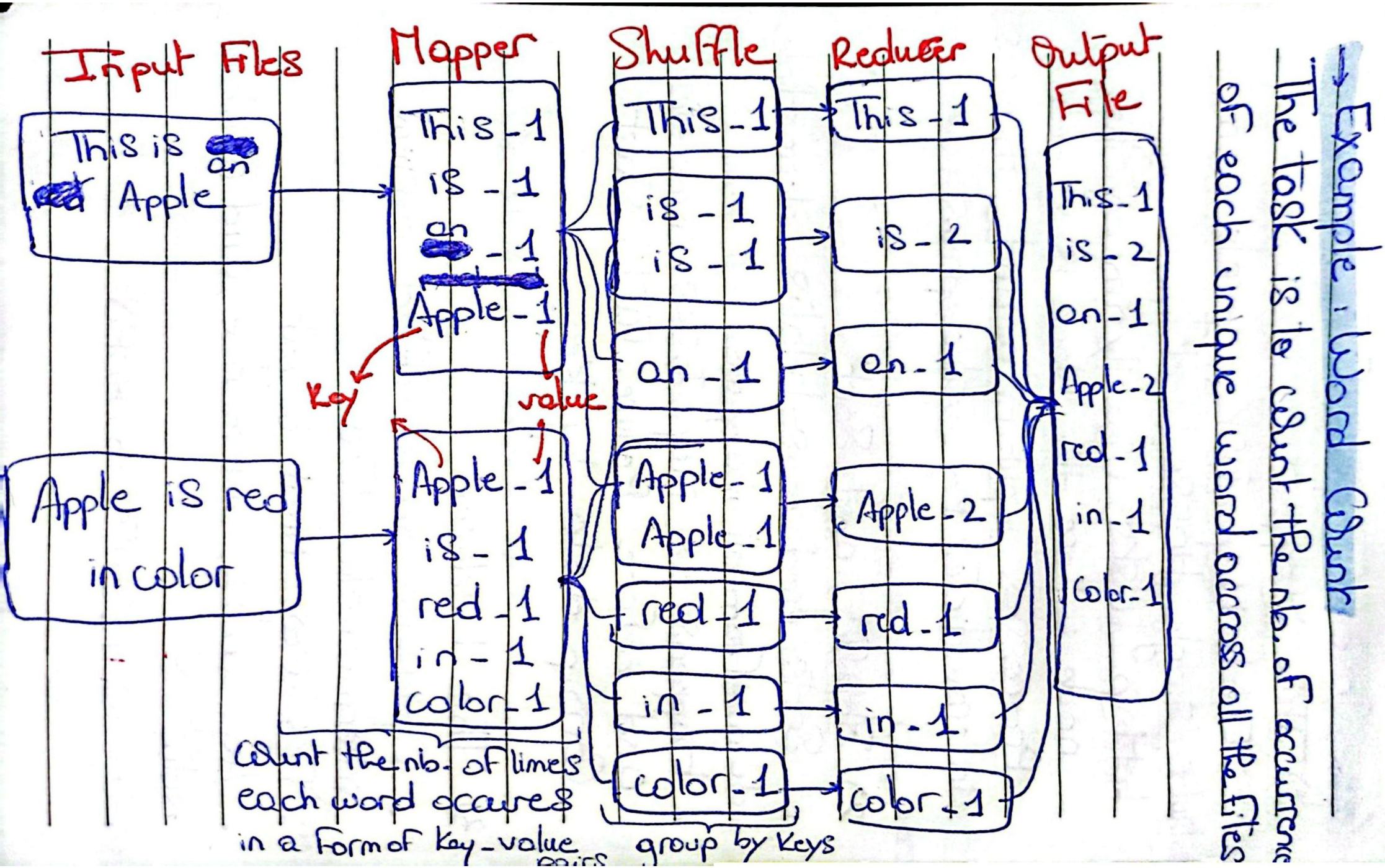
- Each Map task operates on a single HDFS block
- Map tasks usually run on the node where the block is stored
- Processes input data and produces intermediate key-value pairs

2) Shuffle and Sort

- Sorts and consolidates intermediate data from all mappers
- Happens as Map Tasks complete and before Reduce tasks start
- Group data by key

3) The Reducer

- Operates on shuffled / sorted intermediate data
- Produces final output
- Aggregates, summarize, and combines data
- Writes results to HDFS


Input Files
Mapper
Shuffle
Reducer
Output File
This is an Apple
Apple is red in color
This -1
is -1
an -1
Apple -1
Apple -1
is -1
red -1
in -1
color -1
Key
value
is -1
is -1
an -1
Apple -1
Apple -1
red -1
in -1
color -1
is -2
an -1
Apple -2
red -1
in -1
color -1
This -1
is -2
an -1
Apple -2
red -1
in -1
Color -1
Count the nb. of lines each word occures in a form of key-value pairs
group by Keys
Example : Word Count
The task is to count the nb. of occurrence of each unique word across all the files

5) Hive (Data Querying)

→ What is Hive?

- Distributed data warehouse built on top of HDFS to manage and organize large amounts of structured data
- Provide SQL-like interface for querying big data stored in Hadoop
- Translates HiveQL queries MapReduce jobs automatically

6) Apache Ambari (Management)

- Easy to use Hadoop management web ui
- Interface for Hadoop and other applications from the Hadoop ecosystem