HADOOP2.x and WordCount

Praveen Kumar Chandaliya, DoAI, SVNIT

Hadoop Storage Scalability

- **KB (Kilobyte)** = 10^3 bytes -> **MB (Megabyte)** = 10^6 bytes
- **GB (Gigabyte)** = 10^9 bytes-> **TB (Terabyte)** = 10^{12} bytes
- **PB (Petabyte)** = 10^{15} bytes ->**EB (Exabyte)** = 10^{18} bytes
- **ZB (Zettabyte)** = 10^{21} bytes- >**YB (Yottabyte)** = 10^{24} bytes

At the lower end (KB–GB):

Hadoop is **not efficient** for very small files (like KB or MB scale) because each file creates metadata overhead in the **NameNode**. A few GB can be managed, but traditional databases or file systems are usually better at this scale.

Medium scale (TB–PB):

•Hadoop truly shines here.

•Large-scale enterprises, research labs, and social networks typically manage **terabytes to petabytes** of logs, images, videos, and structured/unstructured data in HDFS.

•The default block size (64MB, 128 MB, or 256 MB) makes it efficient for splitting huge files and distributing them across DataNodes.

Large scale (EB–YB):

•Modern Hadoop clusters (especially with **cloud + commodity hardware**) can scale up to **exabytes** of storage.

•Companies like Facebook, Yahoo, and LinkedIn maintain **multi-exabyte Hadoop clusters**.

•**Yottabyte scale** is more theoretical as of now (no single Hadoop cluster yet runs at yottabyte scale), but HDFS design can extend horizontally by just adding nodes

Data Fact

1 TERABYTE
A $200 HARD DRIVE THAT HOLDS 260,000 SONGS

20 TERABYTE
PHOTOS UPLOADED TO FACEBOOK EACH MONTH

120 TERABYTE
ALL THE DATA AND IMAGES COLLECTED BY THE HUBBLE SPACE TELESCOPE

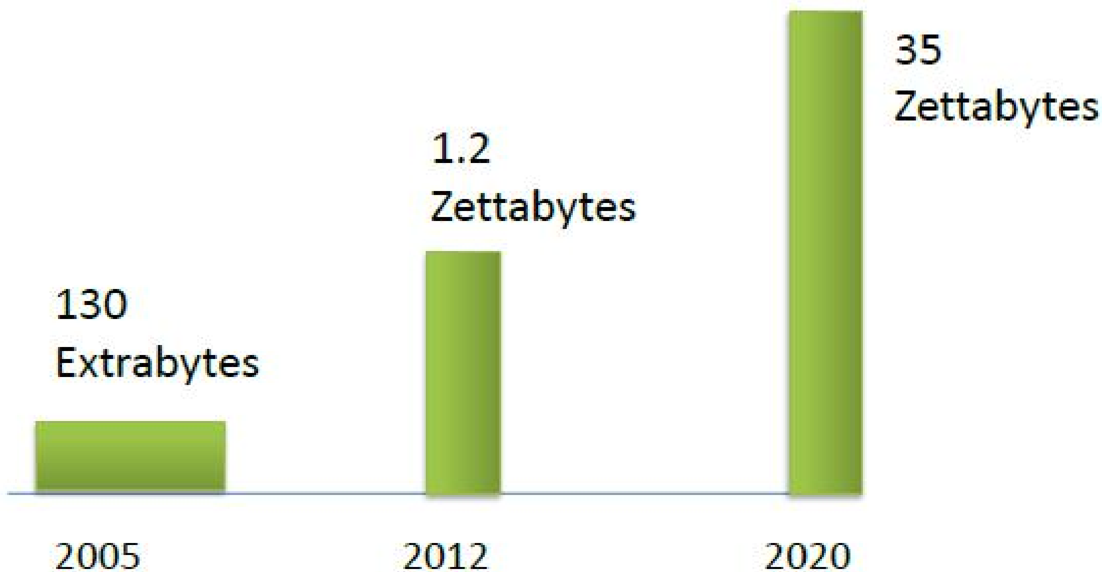
330 TERABYTE
Data that the large Hadron Collider will produce each week

460 TERABYTE
All the digital weather data compiled by the National climatic data center

530 TERABYTE
All the videos on YouTube

600 TERABYTE
Ancestry.coms genealogy data base (includes all U.S. Census records 1790-2000)

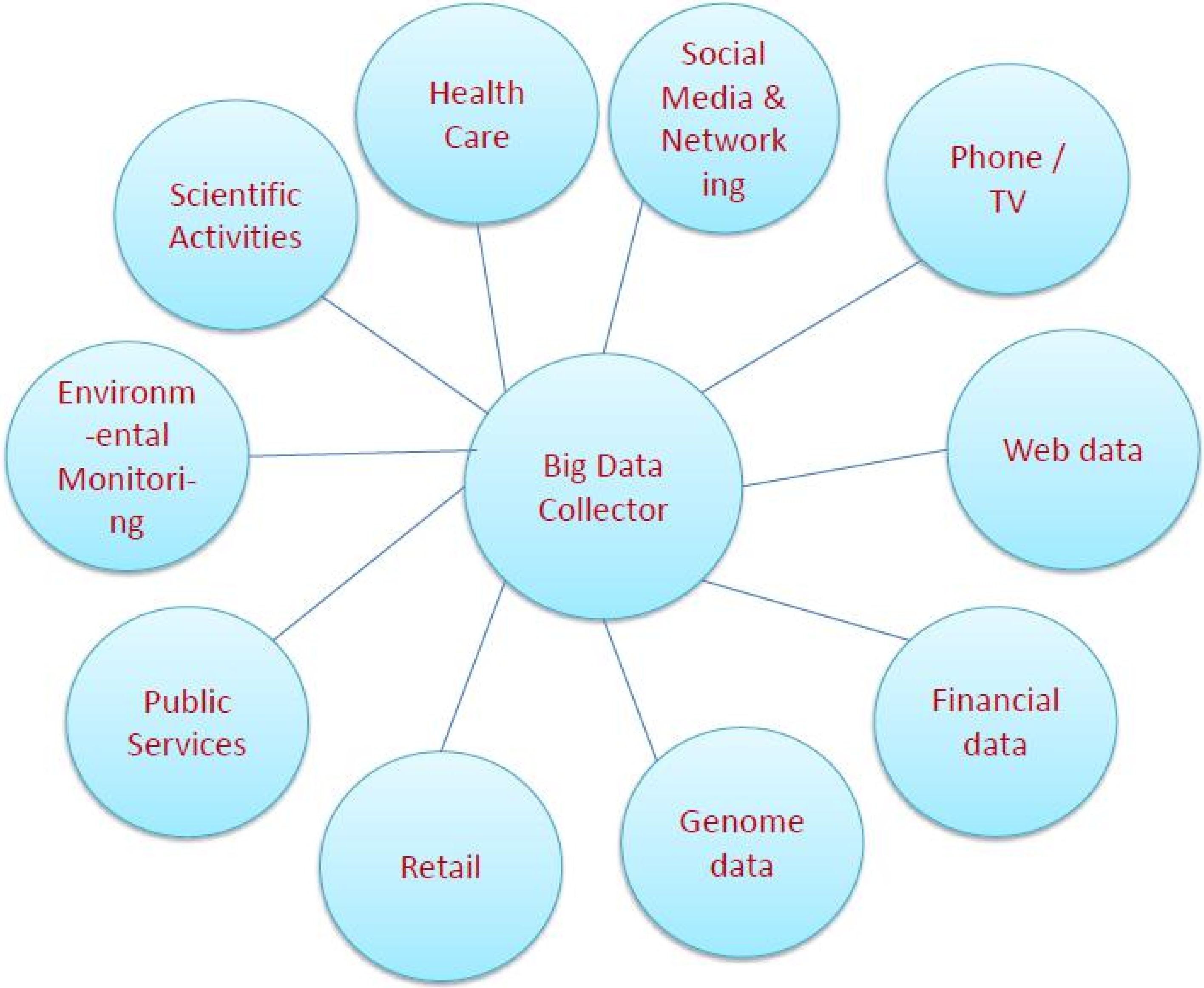
1 PETABYTE
Data processed by Google's servers every 72 minutes



Digital Universe Growth

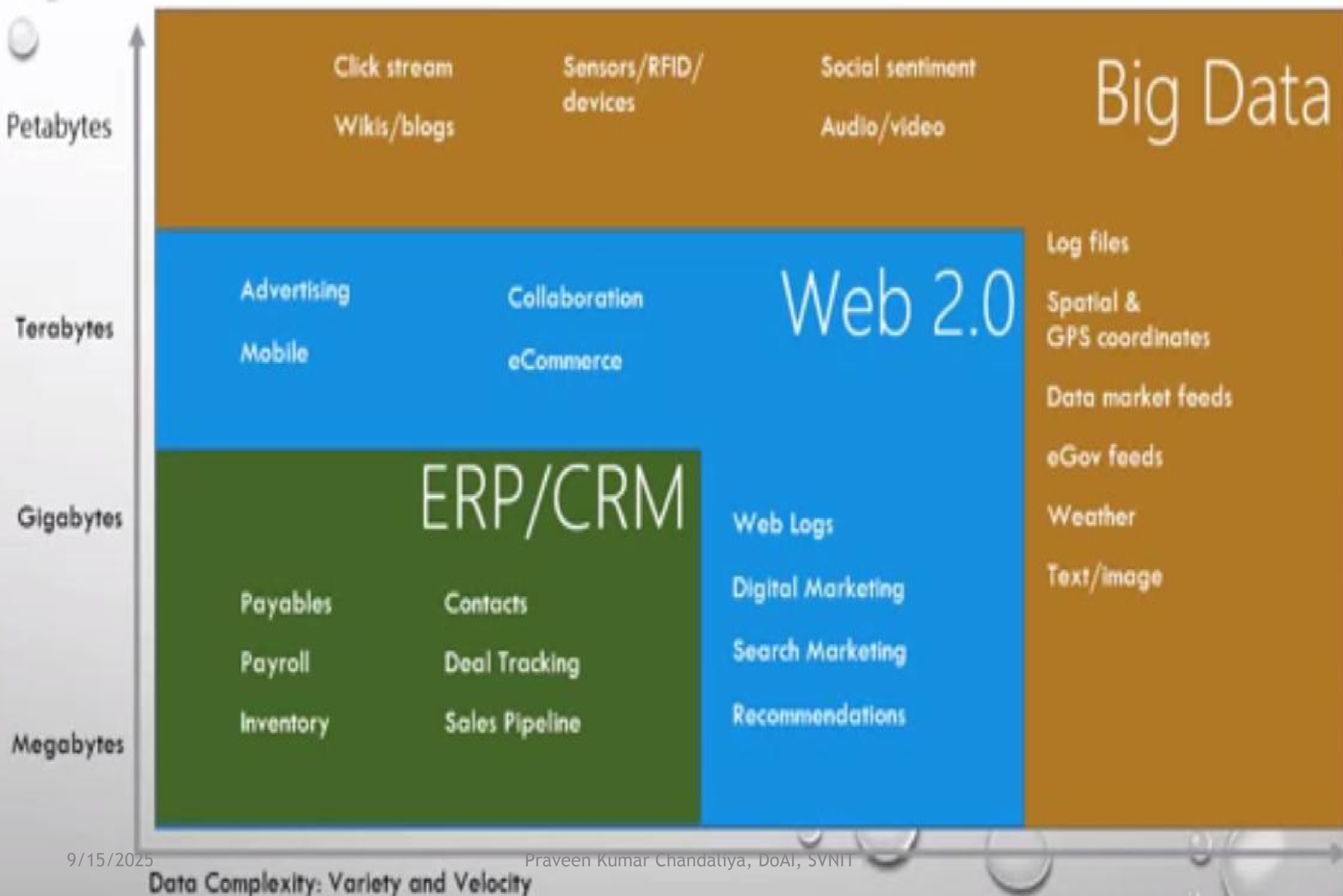
130 extrabyte of data were created and stored in 2005
1.2 Zettabytes of data were created and stored in 2012
35 Zettabytes of data were created and stored in 2020

Note:- Big data is difficult to work using RDMS



Sources of Big Data

WHAT IS BIG DATA?

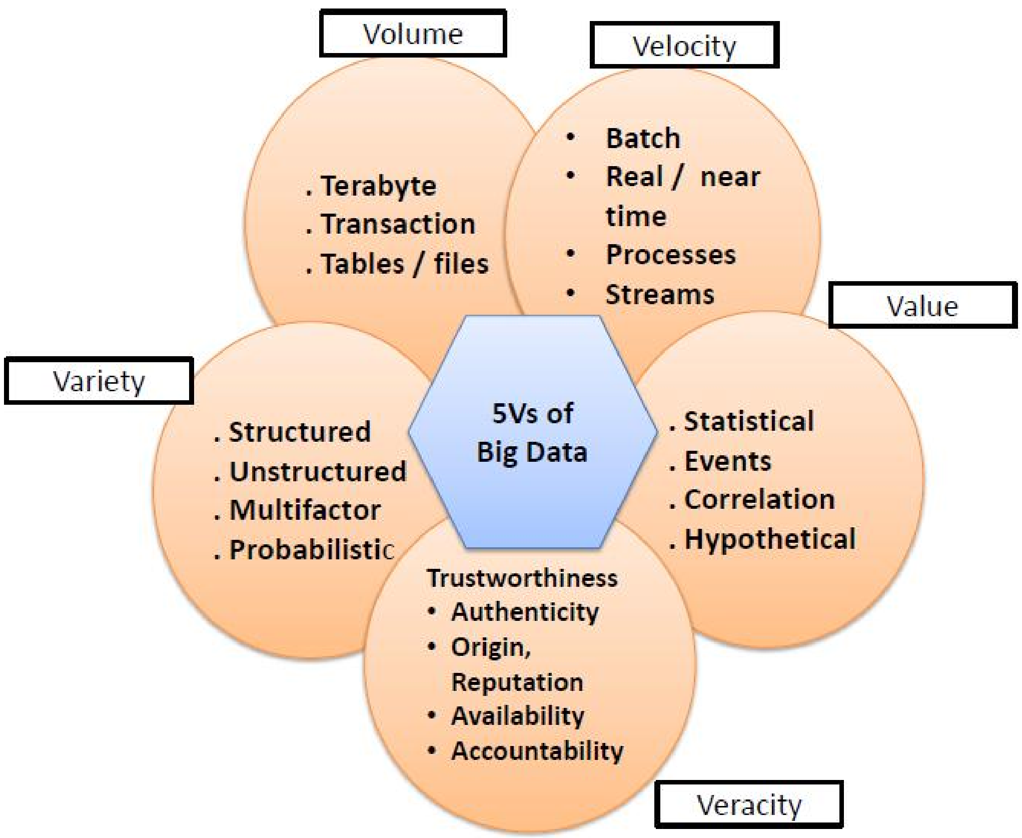


Big Data Definition

- Big data is high volume, high velocity and high-variety information assets that demand cost effective, innovative forms of information processing for enhanced insight and decision making-Gartner.

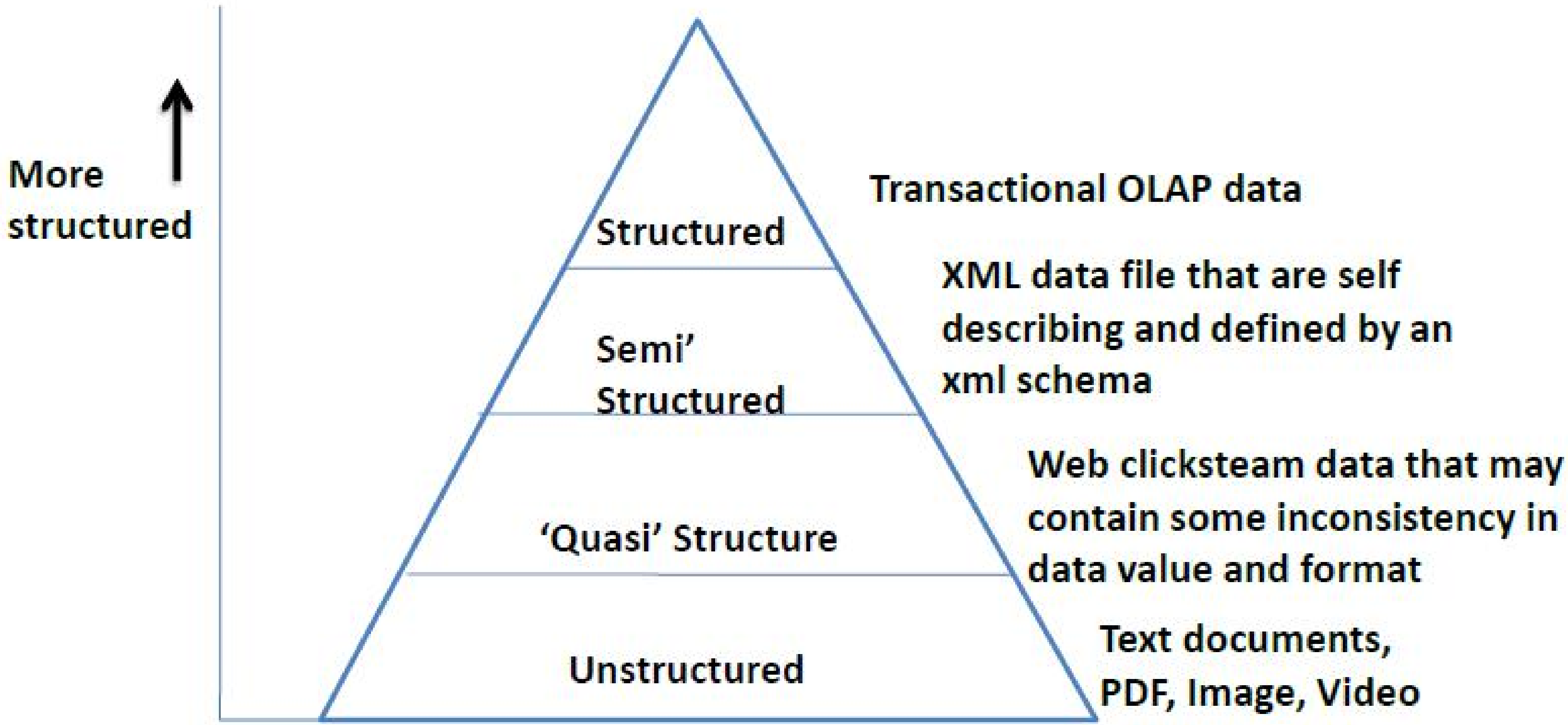
- Big data refers to large data sets that are challenging to store, search, share, visualize and analyze.

- With big data the value is discovered through a refining modeling process. Make hypothesis, create statistical, visual, semantic models validate then make a new hypothesis.



"5Vs of Big Data"

Big Data Characteristics

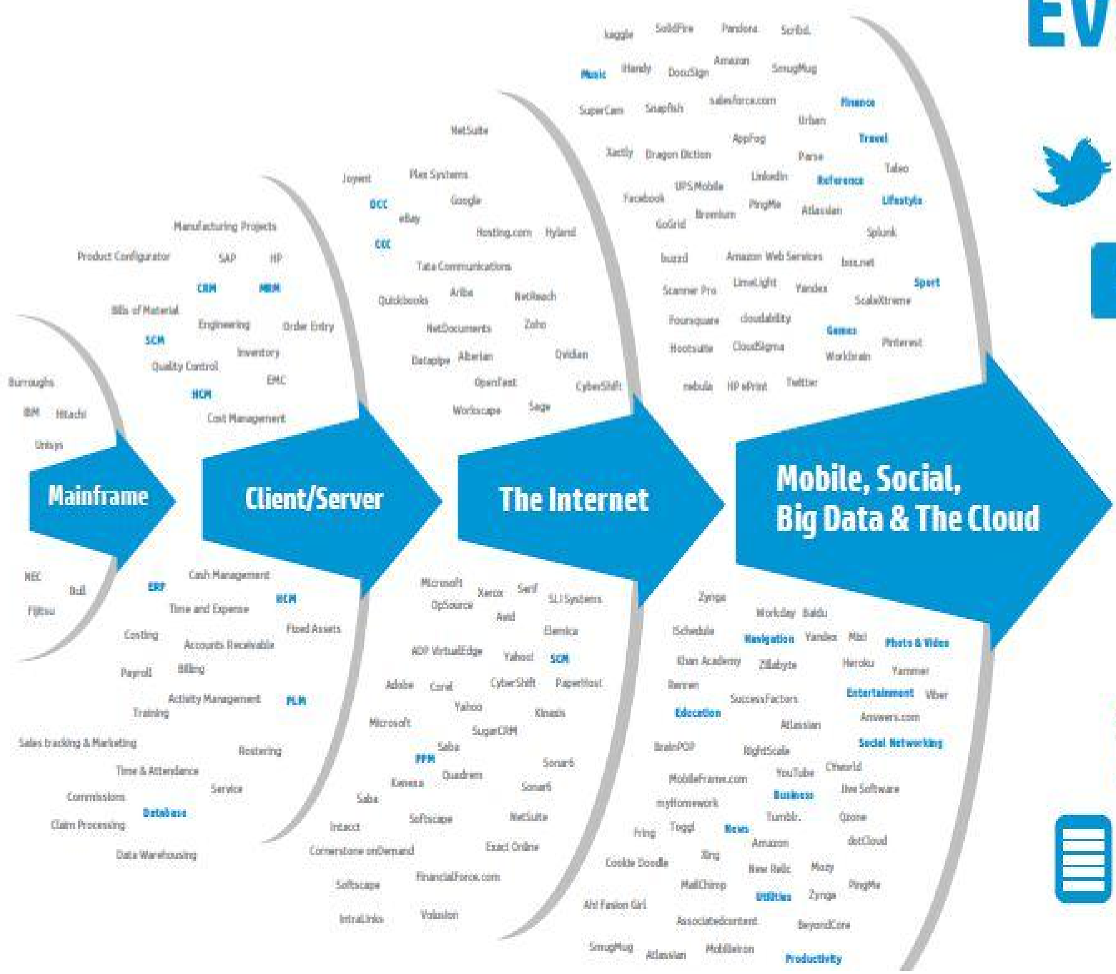


OLAP: Online transaction Processin

Characteristics of Big Data V5

Characteristics	Description
Volume	Scale of data and processing need. It calls for scalable storage and distributed approach to querying.
Velocity	How fast data is being produced & changed. The speed with which data is received, understood & processed.
Veracity	The quality and provenance of the information in the face of data un-certainty from many places.
Variety	Data in different format and from various sources. Difficult to integrate data may be structured, quesi/ semi structured.
Value	Identifying which data is valuable then transformed and analysed.

A new style of IT emerging



Every 60 seconds

- **98,000+** tweets
- **695,000** status updates
- **11million** instant messages
- **698,445** Google searches
- **168 million+** emails sent
- **1,820TB** of data created
- **217** new mobile web users

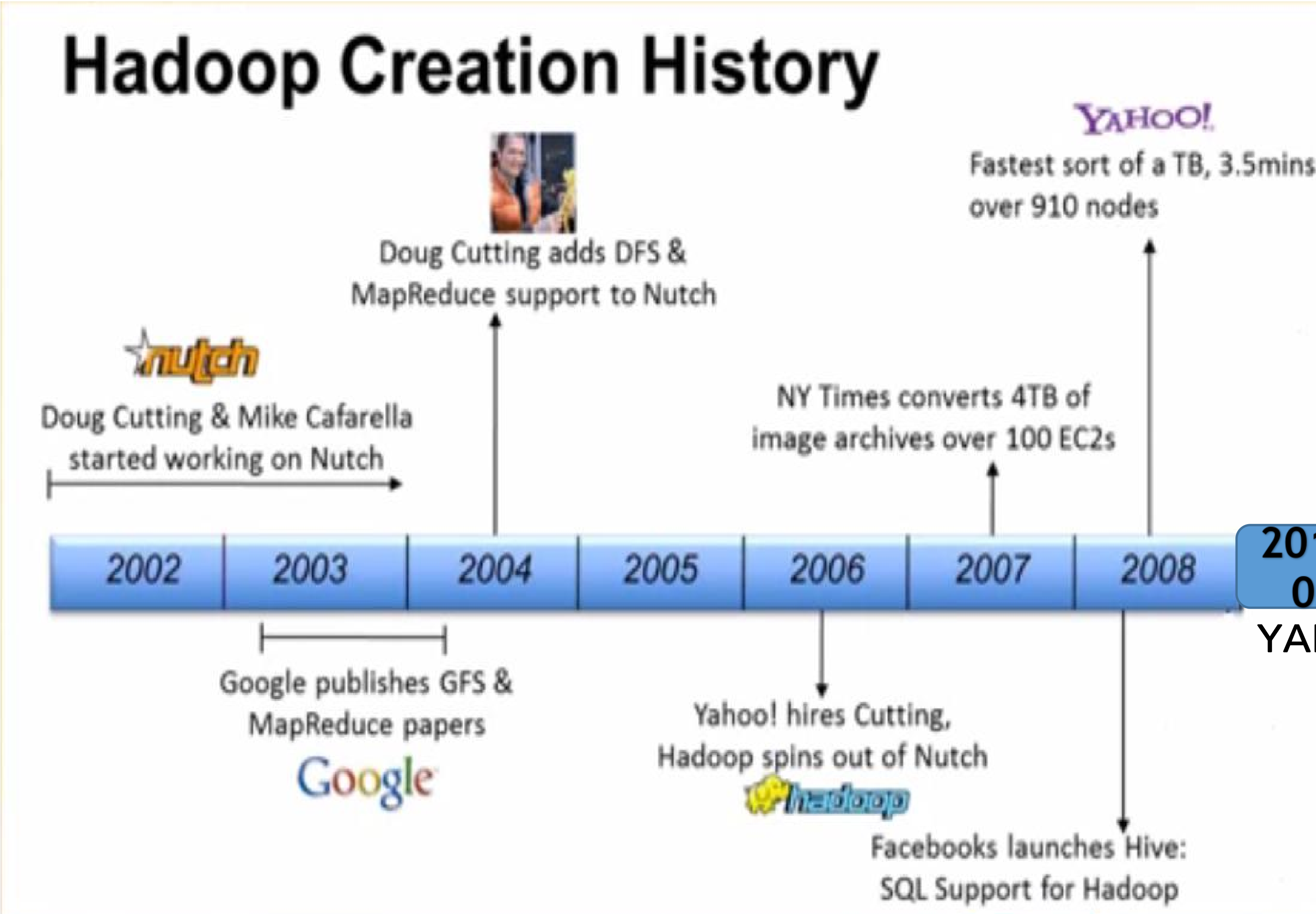

A NEW SET OF QUESTIONS
What's the social sentiment for my brand or products
SOCIAL & WEB ANALYTICS
LIVE DATA FEEDS
How do I optimize my fleet based on weather and traffic patterns?
How do I better predict future outcomes?
ADVANCED ANALYTICS

What is Hadoop?

- Hadoop is an open-source software framework for storing and processing massive amounts of data (Big Data) across clusters of commodity computers using simple programming models.

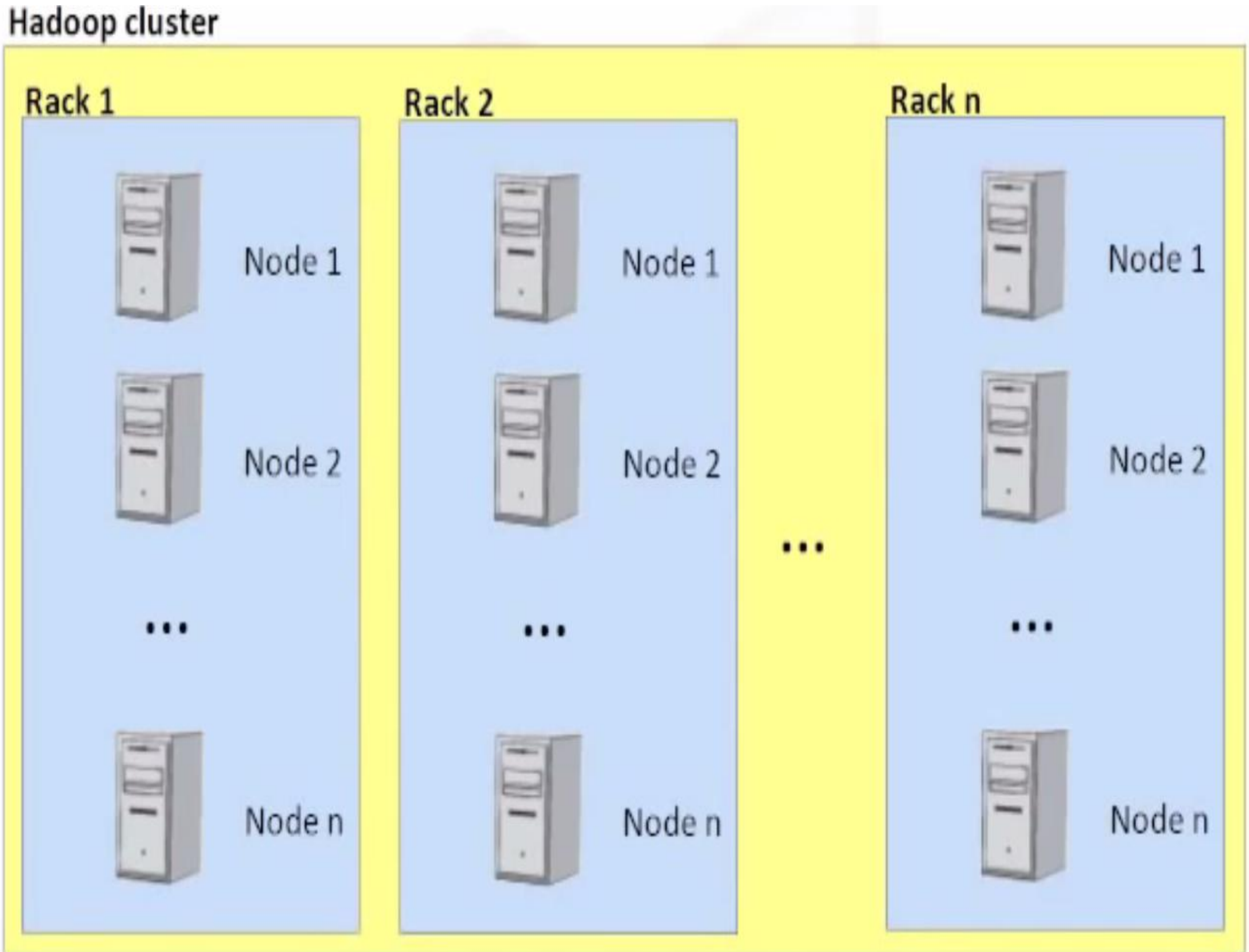
Characteristic

- **HDFS is designed to store a very large amount of information(TB & PB). This requires spreading the data across a large number of machines. It also supports much larger file sizes than NFS.**
- **HDFS should store data reliably. If individual machines in the cluster malfunction(fail), data should still be available.**
- **HDFS should provide fast, scalable access to this information. it should be possible to serve a large number of clients by simply adding more machines to the cluster.**
- **HDFS should integrate well with Hadoop MapReduce, allowing data to be read and computed upon locally when possible.**



2013-2017
YARN

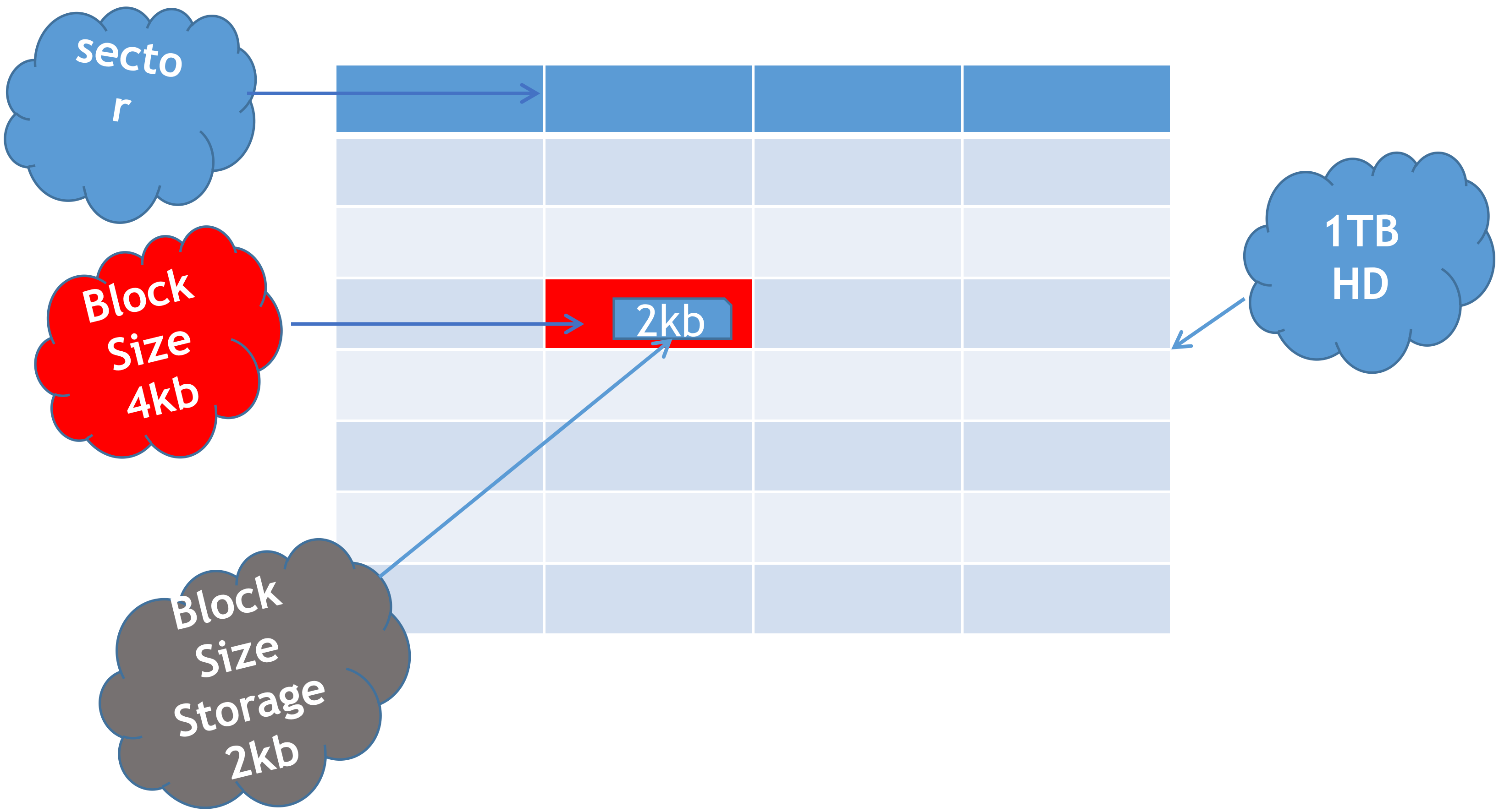
2017 Onward
Hadoop3.0


Hadoop cluster
Rack 1
Node 1
Node 2
...
Node n
Rack 2
Node 1
Node 2
...
Node n
...
Rack n
Node 1
Node 2
...
Node n

HDFS

- **It is a specially designed file system for storing huge data sets with a cluster of commodity hardware with a streaming access pattern(write one, read any number of times, but do not try to change the content of the file that was written in HDFS).**

NORMAL FILE SYSTEM



Four Main Modules of Hadoop

Hadoop is composed of four key modules that work together:

1. Hadoop Distributed File System (HDFS) - The Storage Layer

This is the storage system of Hadoop.
• **How it works:** It breaks down huge files into smaller blocks (typically 64MB, 128MB, or 256MB) and distributes and replicates these blocks across multiple machines in a cluster.

• **Key Features:**

- **Fault Tolerance:** If one machine (or disk) fails, the data is still available on other machines because it's replicated. The system automatically handles this.
- **Scalability:** You can easily add more machines to the cluster to increase storage capacity.
- **Designed for Hardware Failure:** It assumes that hardware failures are common and handles them gracefully.

HDFS Components

i. NameNode

It is also known as *the Master* node. NameNode does not store actual data or a dataset. NameNode stores Metadata, i.e., number of blocks, their location, on which Rack, which Datanode the data is stored, and other details. It consists of files and directories.

Tasks of HDFS NameNode

•Manage file system namespace., Regulates client's access to files, Executes file system execution such as naming, closing, opening files and directories.

ii. DataNode

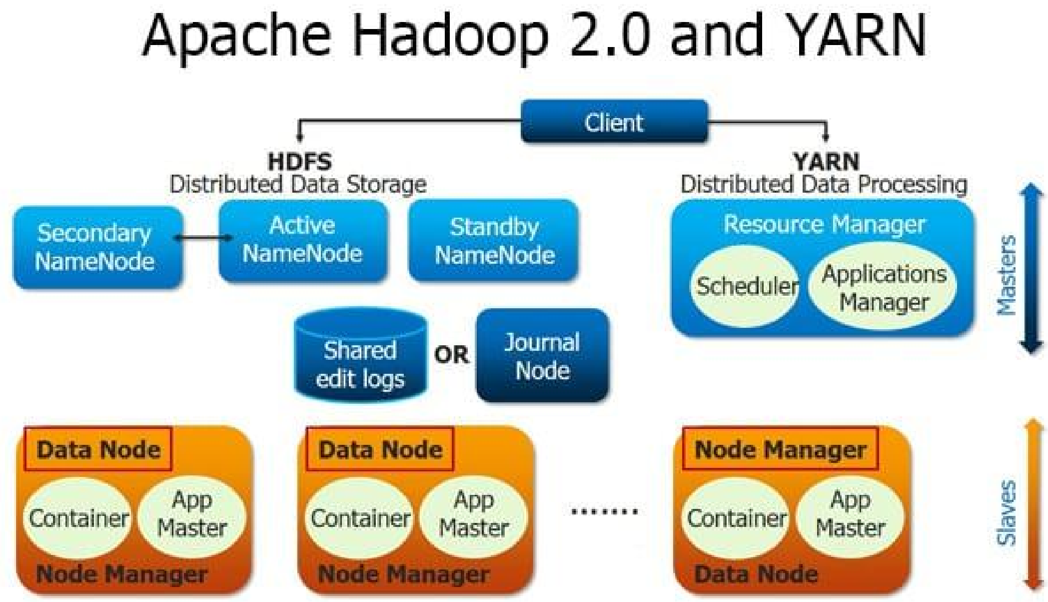
It is also known as *Slave*. HDFS Datanode is responsible for storing actual data in HDFS. Datanode performs **read and write operation** as per the request of the clients. Replica block of Datanode consists of 2 files on the file system.

Tasks of HDFS DataNode

- DataNode performs operations like block replica creation, deletion, and replication according to the instruction of NameNode.
- DataNode manages data storage of the system. This was all about HDFS as a Hadoop Ecosystem component.

2. Yet Another Resource Negotiator (YARN)

Yet Another Resource Negotiator (YARN) - The Resource Manager
YARN is the **brain of the Hadoop ecosystem**, managing the cluster's resources and job scheduling.
How it works: It acts as an operating system for Hadoop. When a user submits a data processing job, YARN allocates the necessary resources (CPU, memory) from the cluster to the application.
Key Function: It separates the responsibilities of resource management from the data processing logic, making Hadoop more efficient and allowing it to run a variety of processing frameworks (like Spark, Tez, etc.), not just the original MapReduce.



3. MapReduce - The Processing Model

This is a programming model for processing large datasets in parallel.

How it works: It works in two stages:

> **Map Stage:** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
> **Reduce Stage:** Takes the output from the Map stage, shuffles and sorts it, and then aggregates the results to produce the final output.

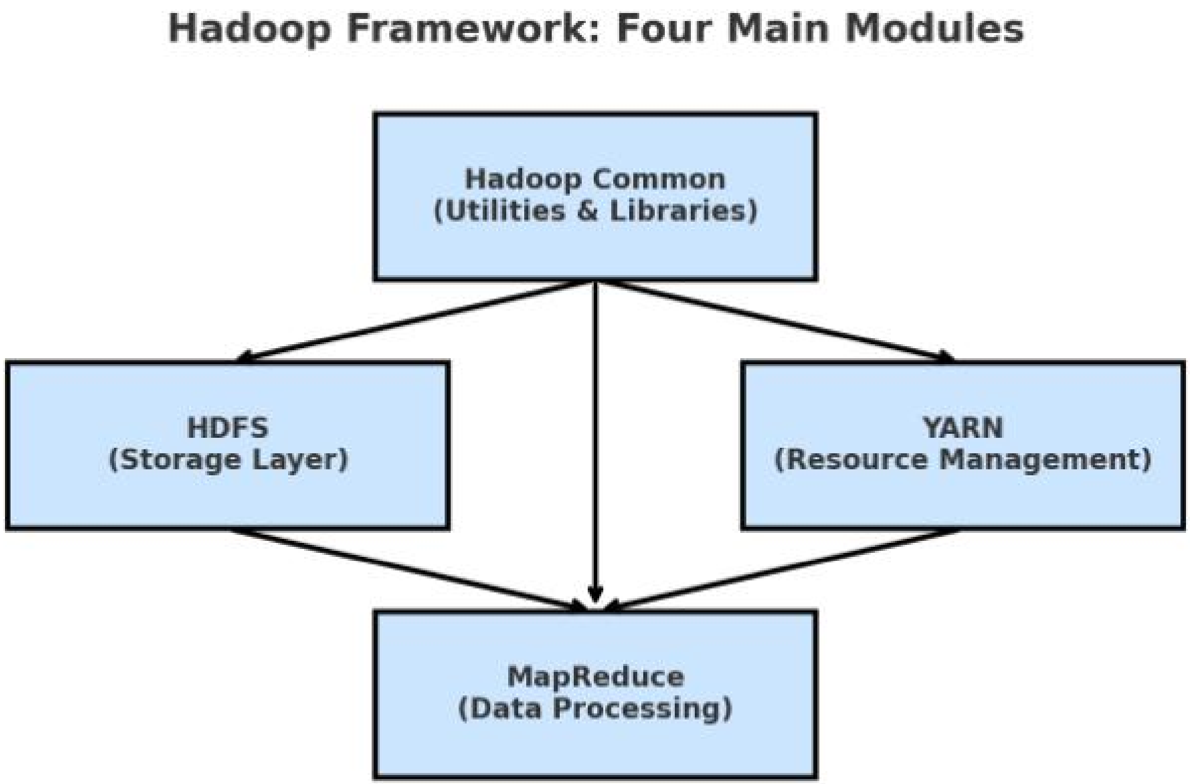
Features of MapReduce

Simplicity – MapReduce jobs are easy to run. Applications can be written in any language, such **as Java, C++, and Python**.
Scalability – MapReduce can process petabytes of data**.**
Speed – By means of parallel processing problems that take days to solve, it is solved in hours and minutes by MapReduce.
Fault Tolerance – MapReduce takes care of failures. If one copy of data is unavailable, another machine has a copy of the same key pair, which can be used for solving the same subtask.

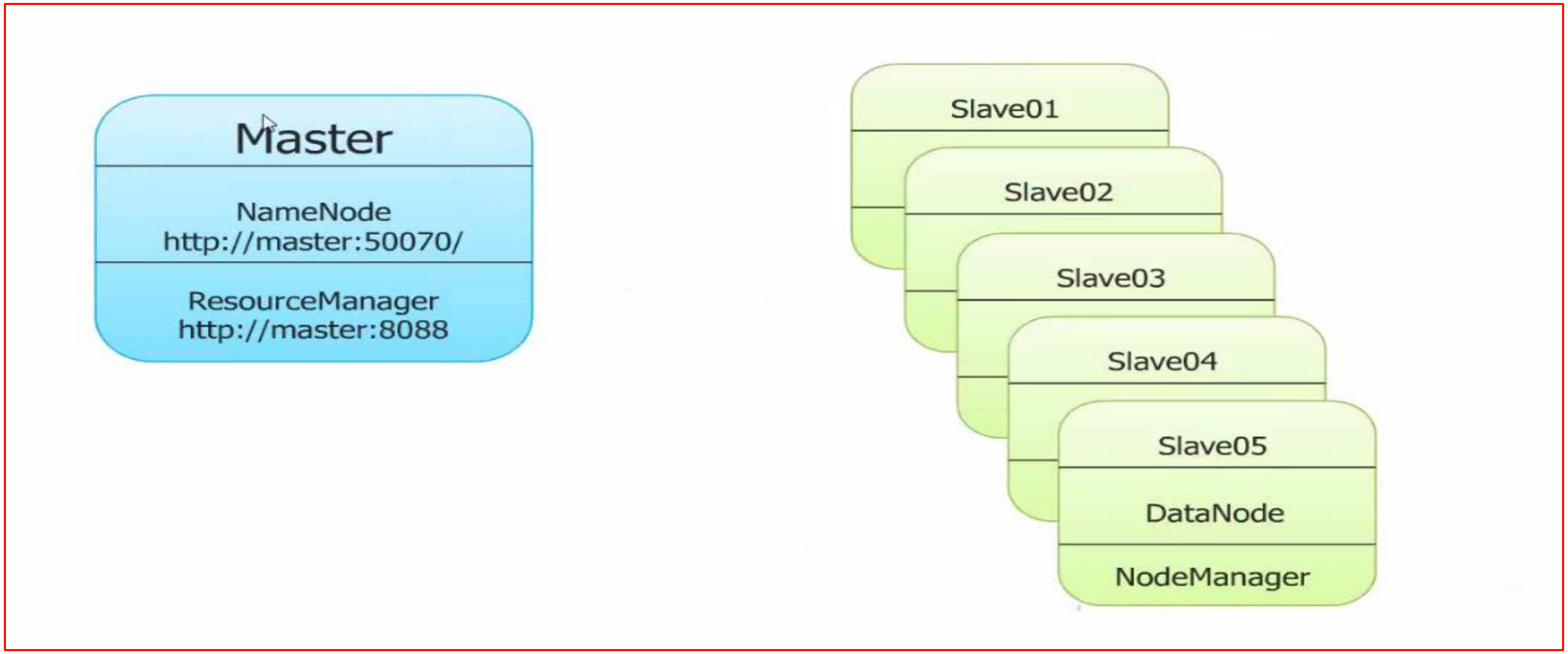
Note: While foundational, MapReduce is often replaced today by faster frameworks like **Apache Spark**, which can run on top of Hadoop's YARN and HDFS.

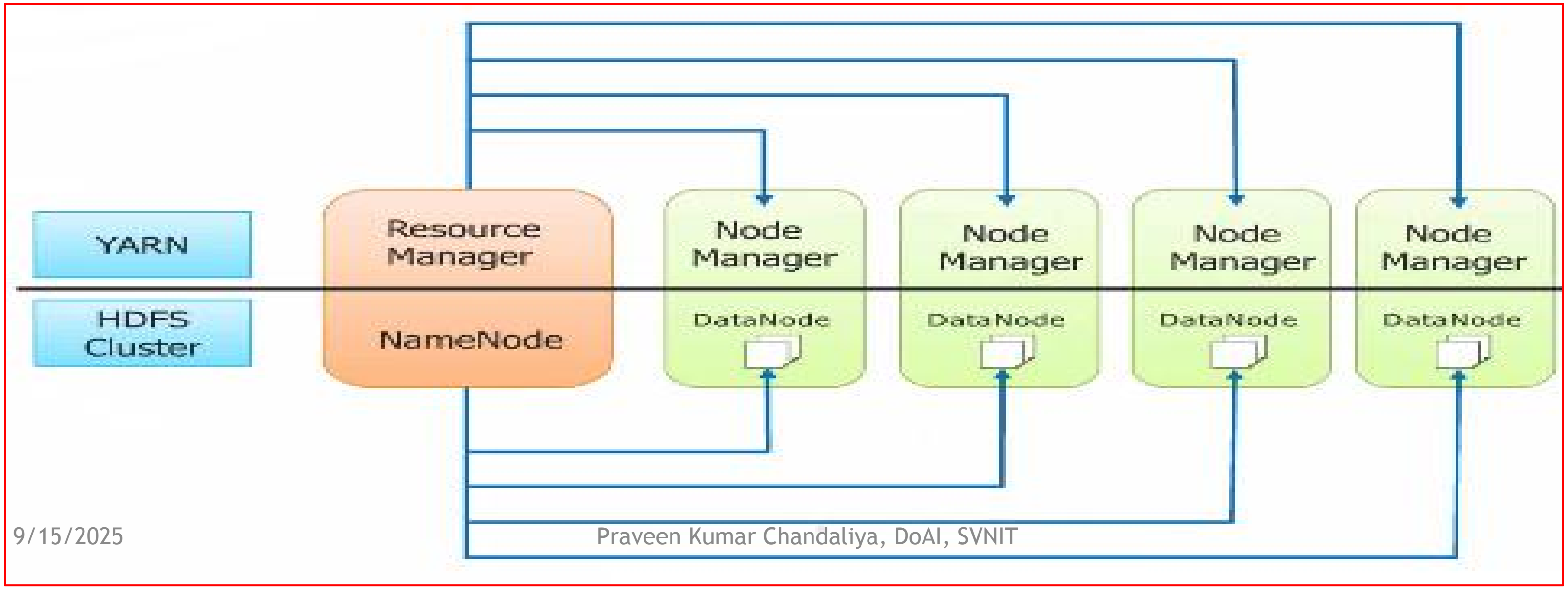
4. Hadoop Common

- The foundation of the entire Hadoop ecosystem.
- Provides **common utilities, libraries, and Java APIs** needed by other Hadoop modules.
- Includes components like configuration files, I/O utilities, RPC (Remote Procedure Call), and serialization libraries.
- Basically, it ensures that the other Hadoop modules can work together seamlessly.

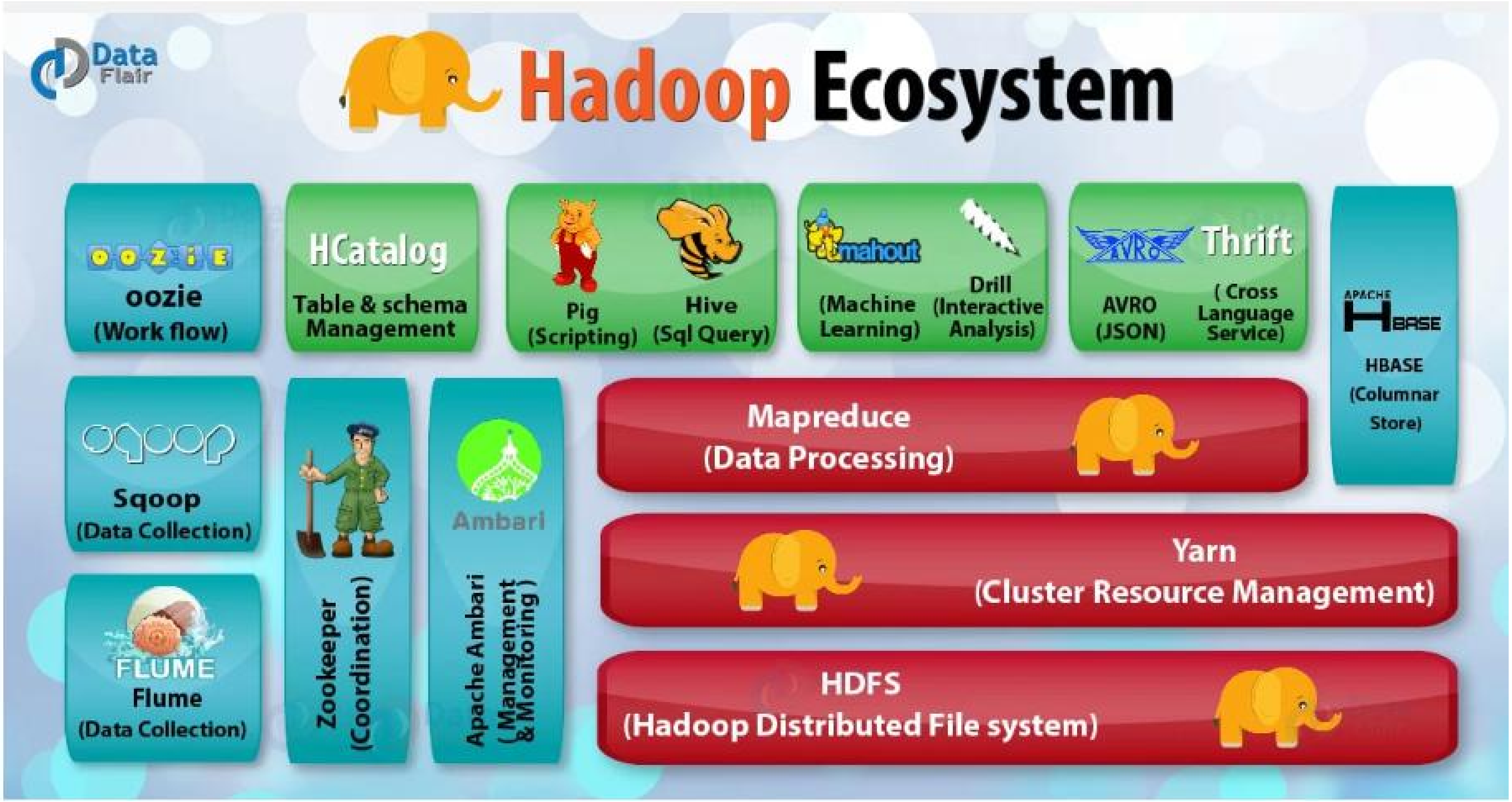


Hadoop2.x Architecture & Hadoop Component





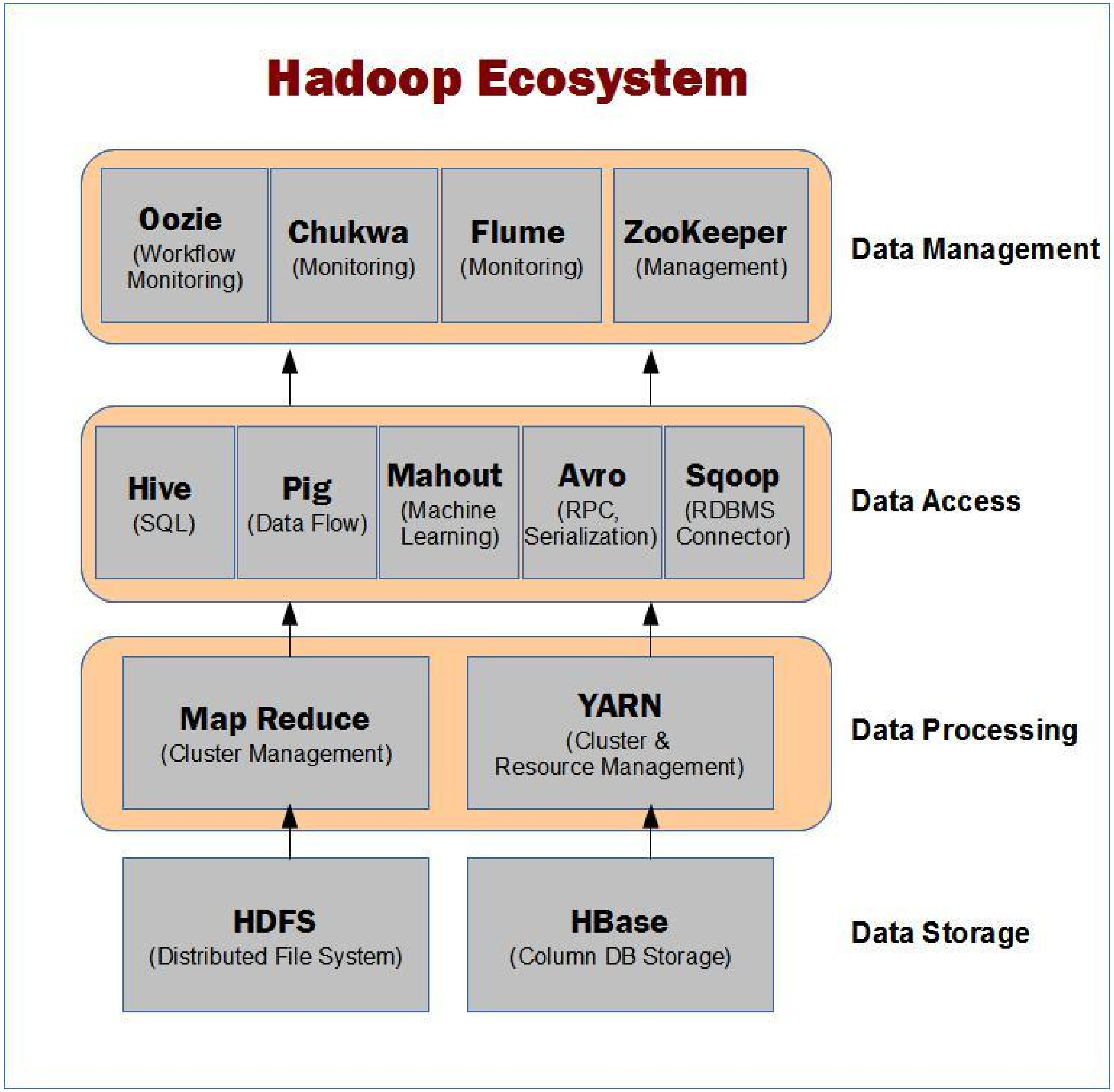
Hadoop Ecosystem



[Top 15 Hadoop Analytics Tools for 2025 – Take a Dive into Analytics - DataFlair]()

[Hadoop Ecosystem and Their Components - A Complete Tutorial - DataFlair]()

Hadoop Ecosystem



Map Reduce

WordCount Job/Program

input.txt (200MB)

hi how are you **how is your job**	**64MB**
how is your family **how is your brother**	**64MB**
how is your sister **what is the time now**	**64MB**
what is the strength of hadoop	**8MB**

Input File Formats :

1. TextInputFormat
2. KeyValueTextInputFormat
3. SequenceFileInputFormat

Objective types used for (key, value) pairs:

Java data types are mapped to Hadoop Writable types for use in MapReduce jobs

Wrapper classes (in Java)	Primitive types	Box classes (in Hadoop)
Integer	int	IntWritable
Long	long	LongWritable
Float	float	FloatWritable
Double	double	DoubleWritable
String	String	Text

```
int num = 42;
IntWritable writableNum = new IntWritable(num);   // primitive → Hadoop box
int backToPrimitive = writableNum.get();          // Hadoop box → primitive
```

```
long bigNum = 100000L;
LongWritable writableLong = new LongWritable(bigNum);
long backToPrimitive = writableLong.get();
```

```
boolean flag = true;
BooleanWritable writableBool = new BooleanWritable(flag);
boolean backToPrimitive = writableBool.get();
```

```
String msg = "Hello Hadoop";
Text writableText = new Text(msg);
String backToPrimitive = writableText.toString();
```

Primitive	Hadoop Box Class	To Box Class	Back to Primitive
int	IntWritable	new IntWritable(i)	writable.get()
long	LongWritable	new LongWritable(l)	writable.get()
float	FloatWritable	new FloatWritable(f)	writable.get()
double	DoubleWritable	new DoubleWritable(d)	writable.get()
boolean	BooleanWritable	new BooleanWritable(b)	writable.get()
byte	ByteWritable	new ByteWritable(b)	writable.get()
String	Text	new Text(str)	writable.toString()
null	NullWritable	NullWritable.get()	(no value)

1. TextInputFormat (Default)

•**Most commonly used** input format in Hadoop.
•**How it works:**

- Splits input files into lines.
- Each line is treated as a record.
- The **key** = line’s byte offset in the file.
- The **value** = contents of the line (text).

Input Line: "hi how are you"
Mapper Input: <0, "hi how are you">

WordCount jobs, log processing, CSV/TSV files

2. KeyValueTextInputFormat

•Used when data is **structured** as **key-value pairs** separated by a delimiter (default = tab \t).
•**How it works:**

- Splits each line into **key** and **value**.
- Left side of delimiter → key, right side → value.

name John
age 25
city Surat

<"name", "John">
<"age", "25">
<"city", "Surat">

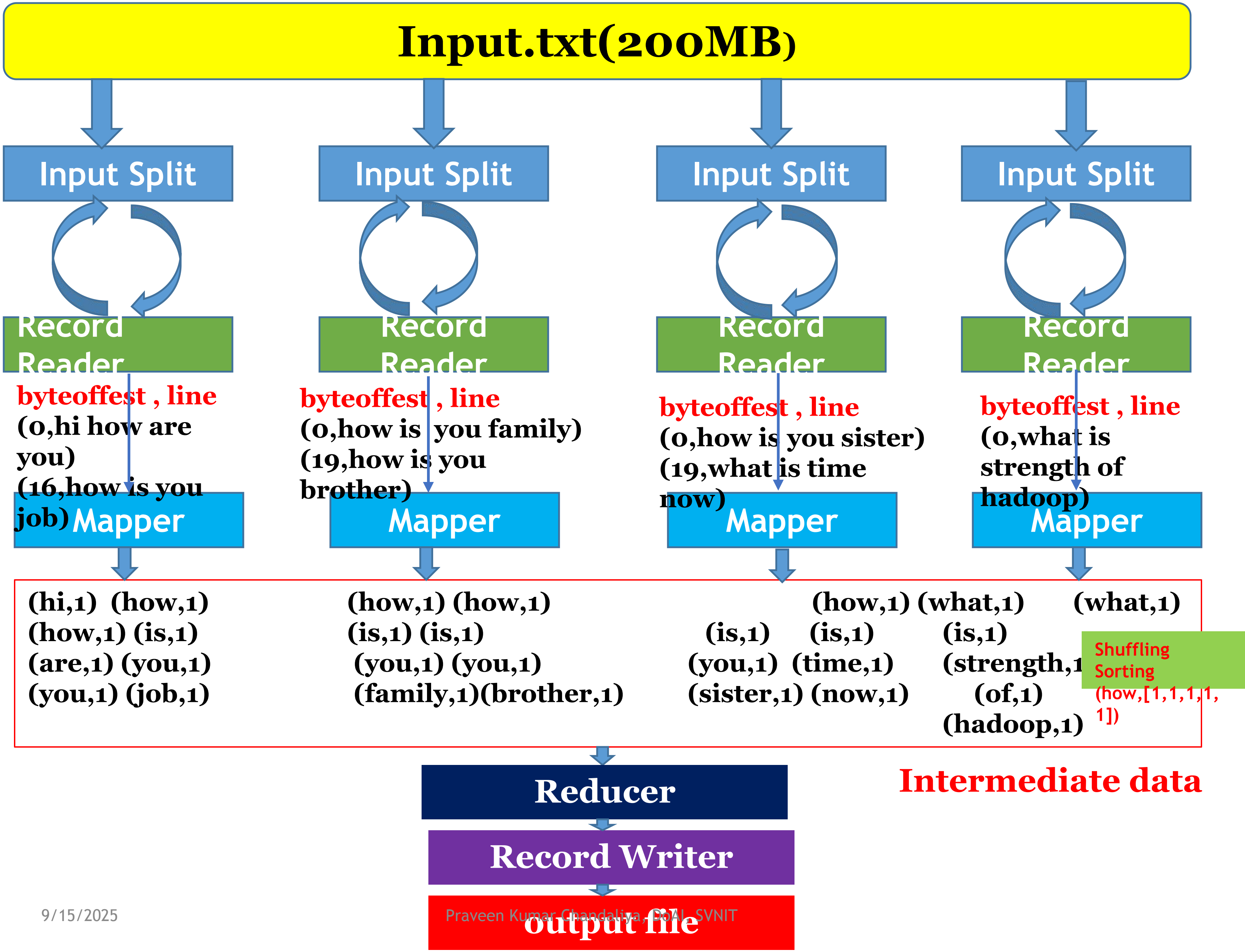
Use case: datasets already in key-value form

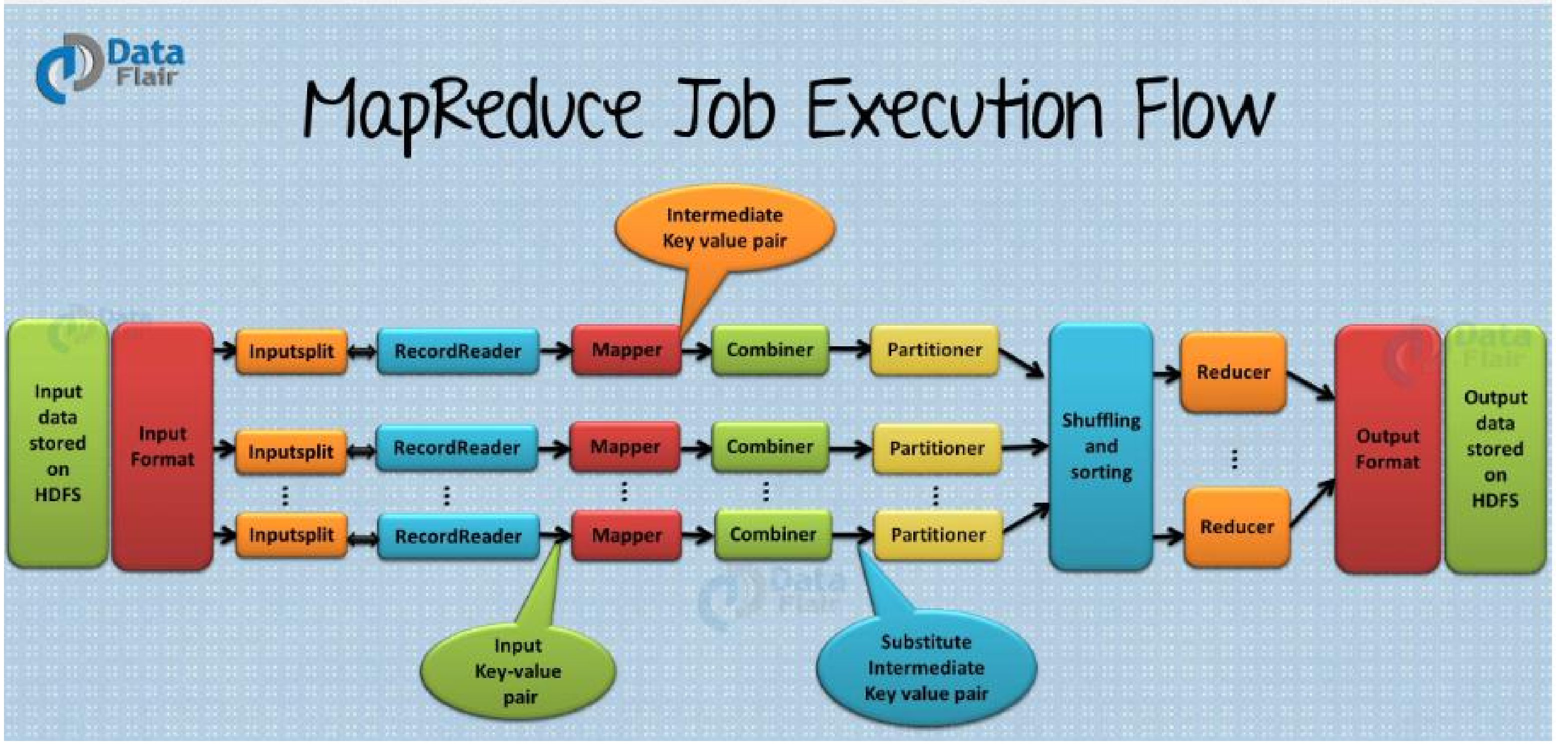
3. SequenceFileInputFormat

- Hadoop's **binary file format**.
- Stores **serialized key-value pairs** in a compressed, splittable way.
- **How it works:**
    - Reads key-value pairs directly from binary sequence files.
    - More efficient than plain text.

<1001, "hi how are you">
<1002, "how is your job">

Intermediate data, machine learning datasets, and large-scale binary storage
images, serialized objects, feature vectors
Used in computer vision pipelines on Hadoop


Input.txt(200MB)
Input Split
Input Split
Input Split
Input Split
Record Reader
Record Reader
Record Reader
Record Reader
byteoffest , line
(0,hi how are you)
(16,how is you job)
byteoffest , line
(0,how is you family)
(19,how is you brother)
byteoffest , line
(0,how is you sister)
(19,what is time now)
byteoffest , line
(0,what is strength of hadoop)
Mapper
Mapper
Mapper
Mapper
(hi,1) (how,1)
(how,1) (is,1)
(are,1) (you,1)
(you,1) (job,1)
(how,1) (how,1)
(is,1) (is,1)
(you,1) (you,1)
(family,1)(brother,1)
(how,1) (what,1)
(is,1) (is,1)
(you,1) (time,1)
(sister,1) (now,1)
(what,1)
(is,1)
(strength,1)
(of,1)
(hadoop,1)
Shuffling
Sorting
(how,[1,1,1,1,1])
Intermediate data
Reducer
Record Writer
output file



Combiner is also known as “Mini-Reducer”. Use of a combiner decreases the time taken for data transfer between the mapper and reducer. The combiner increases the overall performance of the reducer. It reduces the amount of data that the reducer has to process.

[Hadoop Ecosystem and Their Components - A Complete Tutorial - DataFlair]()

[Hadoop Tutorial For Beginners | Hadoop Crash Course | Learn Hadoop From Scratch | Simplilearn]()

Shuffling:
Shuffling is a phase on intermediate data to combine all values into a collection associated with the same key.

(how, [1,1,1,1,1])

(is, [1,1,1,1,1])

Sorting :
Sorting is another phase on intermediate data to sort all (key, value) pairs. Because of the shuffling phase, all unique keys will compare with each other and give output in some sorting order. Basically, this sorting will be done because of Box classes as we have seen….

- IntWritable
- LongWritable
- FloatWritable
- DoubleWritable
- Text

All the above classes implemented the Writable and WritableComparable interfaces so that they can compare with each other.

Step 3: Sort Phase : Keys are **sorted** (lexicographically by default).

(Hadoop, [1])
(are, [1,1])
(hi, [1,1])
(how, [1,1,1])
(is, [1])
(you, [1,1])

Step 4: Reduce Phase : Reducer aggregates values for each key.

(Hadoop,1)
(are,2)
(hi,2)
(how,3)
(is,1)
(you,2)

Step 5: Final WordCount Output

Hadoop 1
are 2
hi 2
how 3
is 1
you 2

Hadoop Distribute File System command

1) $hadoop fs -ls
2) $hadoop fs -mkdir <directory>
3) $hadoop fs -put <local src directory> <hdfs des directory>
4) $hadoop fs -copyFromLocal <local src directory> <hdfs des directory>
5) $hadoop fs -moveFormLocal <local src directory> <hdfs des directory> //cut and past
6) $hadoop fs -get <hdf src directory> <local des directory>
7) $hadoop fs -copyToLocal <hdf src directory> <local des directory>
8) $hadoop fs -moveToLocal <hdf src directory> <local des directory>
9) $hadoop fs -cat <file name>
10) $hadoop fs -rm <file name> //remove file
11) $hadoop fs -rmr <directory name> //remove directory
12) $hadoop fs -cp <src-location> <des-location>
13) $hadoop fs -mv <src-location> <des-location>
14) $hadoop fs -du <file name> //disk use

MAPPER CLASS

Name of the Mapper Class which inherits Super Class Mapper

```
public static class Map extends
        Mapper<LongWritable, Text, Text, IntWritable> {
```

Mapper Class takes 4 Arguments i.e.
Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>

```
//Defining a local variable one of type IntWritable
private final static IntWritable one = new IntWritable(1);
//Defining a local variable word of type Text
private Text word = new Text();
```

We override the map method which is defined in the Parent (Mapper) Class. It takes 3 arguments as Inputs
map (KEYIN key, VALUEIN value, Context context)

```
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
```

//Converting the record (single line) to String and storing it
in a String variable line

```
String line = value.toString();
//StringTokenizer is breaking the record (line) into words
StringTokenizer tokenizer = new StringTokenizer(line);
//Running while loop to get each token(word) one by one from StringTokenizer

while (tokenizer.hasMoreTokens()) {
//Saving the token(word) in a variable word
word.set(tokenizer.nextToken());
//Writing the output as (word, one), the value  Of word will be equal to token and value  of one is 1
context.write(word, one);
}
}
```

In the map method, we receive a record (single line). It is stored in a string variable line. Using StringTokenizer, we are breaking the line into individual words called tokens, on the basis of space as delimiter. If the line was Hello There, StringTokenizer will give two tokens Hello and There. Finally using the context object we are dumping the Mapper output. So as per our example the Output from the Mapper will be
Hello 1 & There 1 and so on.

The Output from the Mapper is taken as Input by the Reducer

Name of the Reducer Class that inherits the Super Class Reducer

public static class Reduce extends
Reducer<Text, IntWritable, Text, IntWritable> {

Reducer Class takes 4 Arguments i.e.
Reducer <KEYIN, VALUEIN, KEYOUT, VALUEOUT>

We override the reduce method which is defined in the Parent (Reduce) Class. It takes 3 arguments as Inputs

reduce (KEYIN key, VALUEIN value, Context context)

public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
//Defining a local variable sum of type int
int sum = 0;
//Running for loop to iterate over the values present in
Iterator
for (IntWritable val : values) {
//We are adding the value to the variable over every iteration
sum = sum + val.get();

//Finally writing the key and the value of sum(number of times

In the reduce method, we receive a key as word and a list of values as input.
For Example: Hello <1,1,1,1>
To find out the occurrence of the word 'Hello' in the input file then we simply have to sum all the values of the list. Hence, we run a for loop to iterate over the values one by one and adding it to variable sum.
Finally, we will write the output, i.e, key (word) & value (sum) using the context object.
So as per the above example the output will be:
Hello 4

The main method is known as the entry point of the application. This is the method that is called as soon as jar is executed

```
public static void main(String[] args) throws Exception {
//Creating an object of Configuration class, which loads the configuration parameters
Configuration conf = new Configuration();
//Creating the object of the Job class and passing the conf object and Job name as arguments.
The Job class allows the user to configure the job, submit it, and control its execution.
Job job = new Job(conf, "wordcount");
```

```
//Setting the jar by finding where a given class came from
job.setJarByClass(WordCountNew.class);
//Setting the key class for job output data
job.setOutputKeyClass(Text.class);
//Setting the value class for job output data
job.setOutputValueClass(IntWritable.class);

//Setting the mapper for the job
job.setMapperClass(Map.class);
//Setting the reducer for the job
job.setReducerClass(Reduce.class);

//Setting the Input Format for the job
job.setInputFormatClass(TextInputFormat.class);
//Setting the Output Format for the job
job.setOutputFormatClass(TextOutputFormat.class);
//Adding a path which will act as an input for the MR job. args[0]  means it will use the first
argument written on the terminal  as the input path
FileInputFormat.addInputPath(job, new Path(args[0]));
//Setting the path to a directory where the MR job will dump the output. args[1] means it will use
the second argument written on the terminal as the output path
FileOutputFormat.setOutputPath(job,new Path(args[1]));
//Submitting the job to the cluster and waiting for its completion
job.waitForCompletion(true);
```

Step to Develop Word Count Problem Using Eclipse IDE

Step 1: Create a file (inputfile.txt).
Step 2 : Make a hdfs directory
$ hadoop fs -mkdir -p /home/user/input
Step 3 :loading inputfile.txt from local file system to HDFS.
$hadoop fs –put /home/user/inputfile.txt /home/user/input
Step 4: Write Java programs for processing using Eclipse ID
i) **DriverCode.java ii) MapperCode.java iii) ReducerCode.java**
Step 5 : Set the Java Build path(class path) and Eclipse IDE Compile All java file.
Step 6 : Create jar file using Eclipse IDE.
Step 7: Run jar file
$hadoop jar WordCount.jar /home/user/input/inputfile.txt /home/user/output DriverCode args[0] args[1]
Note : if Main methods file is selected at the time of the jar file creation
$hadoop jar WordCount.jar /home/user/input/inputfile.txt /home/user/output
Step 8: View the result
$hadoop fs -cat /home/user/output/part-r-00000
or
$ hadoop fs -cat /home/user/output/part-00000
Step 9 : if want to remove the folder created using hdfs
$hadoop fs –rm -R /home/user/output