Unit 2
INTRODUCTION TO HADOOP AND HADOOP ARCHITECTURE

Dr. Nilesh M. Patil

Associate Professor

Dept. of Computer Engineering, DJSCE

Syllabus

- Big Data – Apache Hadoop & Hadoop EcoSystem
- Moving Data in and out of Hadoop – Understanding inputs and outputs of MapReduce Concept of Hadoop
- HDFS Commands
- MapReduce-The Map Tasks, Grouping by Key, The Reduce Tasks, Combiners, Details of MapReduce Execution
- **8 Hours**
- **Marks: 20 (approx.)**

Introduction to Hadoop

- Hadoop is an open-source framework from Apache and is used to store process and analyze data that are very huge in volume.
- Founders: Doug Cutting and Mike Cafarella
- Hadoop is written in Java and is not OLAP (online analytical processing).
- It is used for batch/offline processing.
- It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn, and many more.
- Moreover, it can be scaled up just by adding nodes in the cluster.

History of Hadoop

- Founders: Doug Cutting and Mike Cafarella
- 2002 – Apache Nutch (Open Source Web Crawler Software Project)
- 2003 – Google introduced GFS, proprietary distributed file system
- 2004 – Google released white paper on Map Reduce
- 2005 - Doug Cutting and Mike Cafarella introduced NDFS
- 2006 – Doug Cutting and Mike Cafarella quit Google and joined Yahoo, introduced HDFS. Hadoop 0.1.0 version was released.
- 2007 – Yahoo runs 2 clusters of 1000 machines
- 2008 – Hadoop became the fastest system to sort 1TB data on 900 node cluster within 209 seconds
- 2013 – Hadoop 2.2 was released
- 2017 – Hadoop 3.0 was released

Why Hadoop?

Overcomes the traditional limitations of storage and compute.

Leverage inexpensive, commodity hardware as the platform

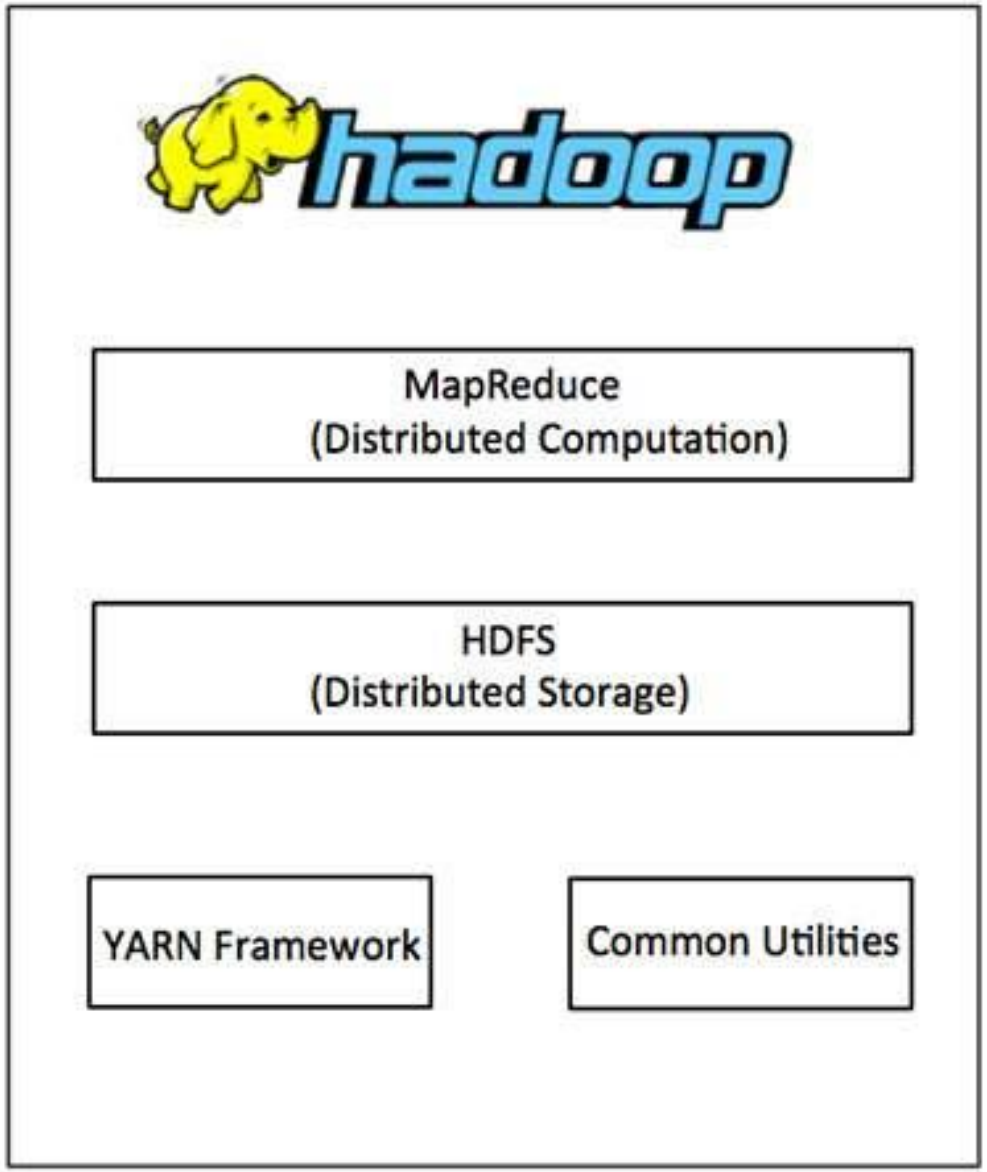
Provide linear scalability from 1 to 4000 servers.

Low cost open source software.

Hadoop Goals

1. **Scalable**: It can scale up from a single server to thousands of servers.
2. Fault tolerance: It is designed with a very high degree of fault tolerance.
3. Economical: It uses commodity hardware instead of high-end hardware.
4. Handle hardware failures: It has the ability to detect and handle failures at the application layer.

Core Hadoop Components



Hadoop Common Package

- Hadoop Common refers to the collection of common utilities and libraries that support other Hadoop modules.
- Hadoop Common is also known as Hadoop Core.
- Hadoop Common also contains the necessary Java Archive (JAR) files and scripts required to start Hadoop.
- The Hadoop Common package also provides source code and documentation, as well as a contribution section that includes different projects from the Hadoop Community.

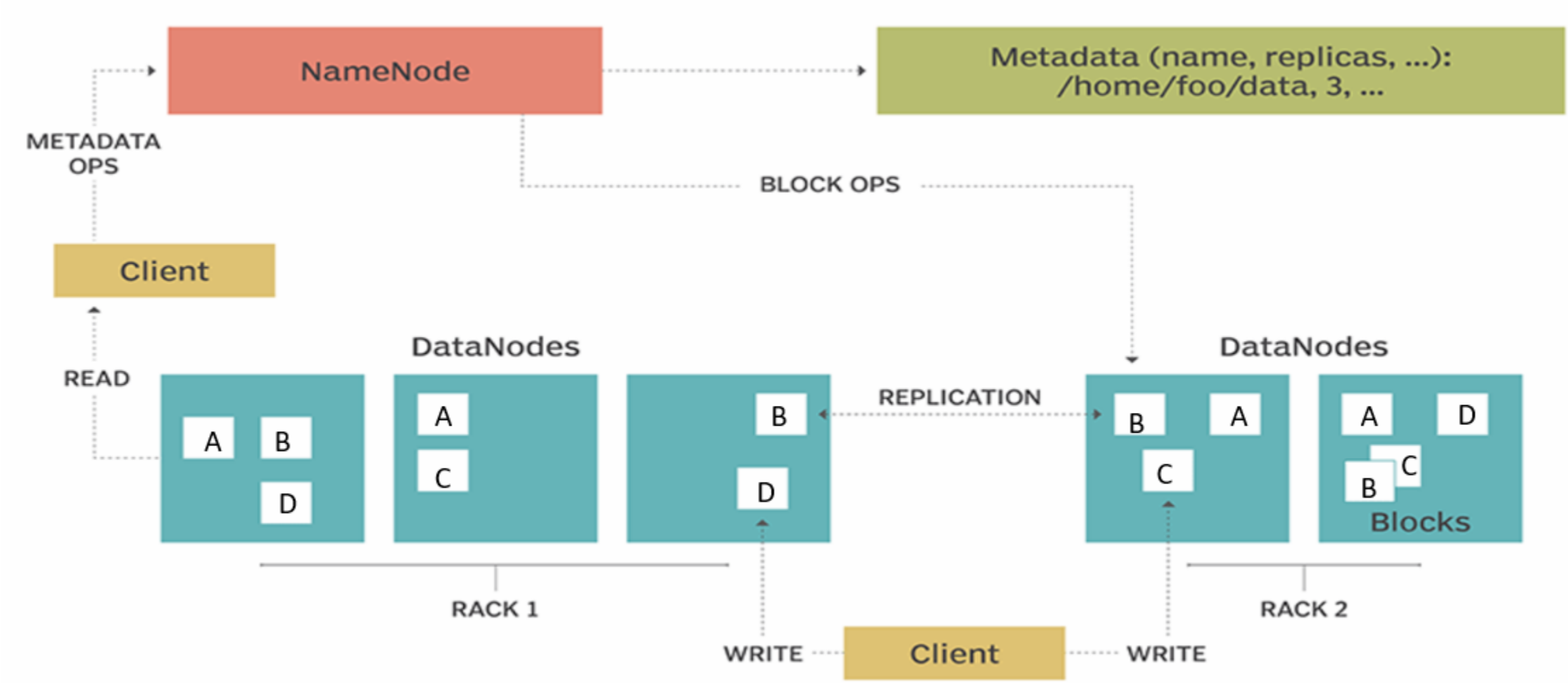
Hadoop Distributed File System (HDFS)

- Hadoop File System was developed using distributed file system design.
- It is run on commodity hardware.
- Unlike other distributed systems, HDFS is highly fault-tolerant and designed using low-cost hardware.
- HDFS holds a very large amount of data and provides easier access.
- To store such huge data, the files are stored across multiple machines.
- These files are stored in a redundant fashion to rescue the system from possible data losses in case of failure.
- HDFS also makes applications available for parallel processing.

HDFS Features

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of NameNode and DataNode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

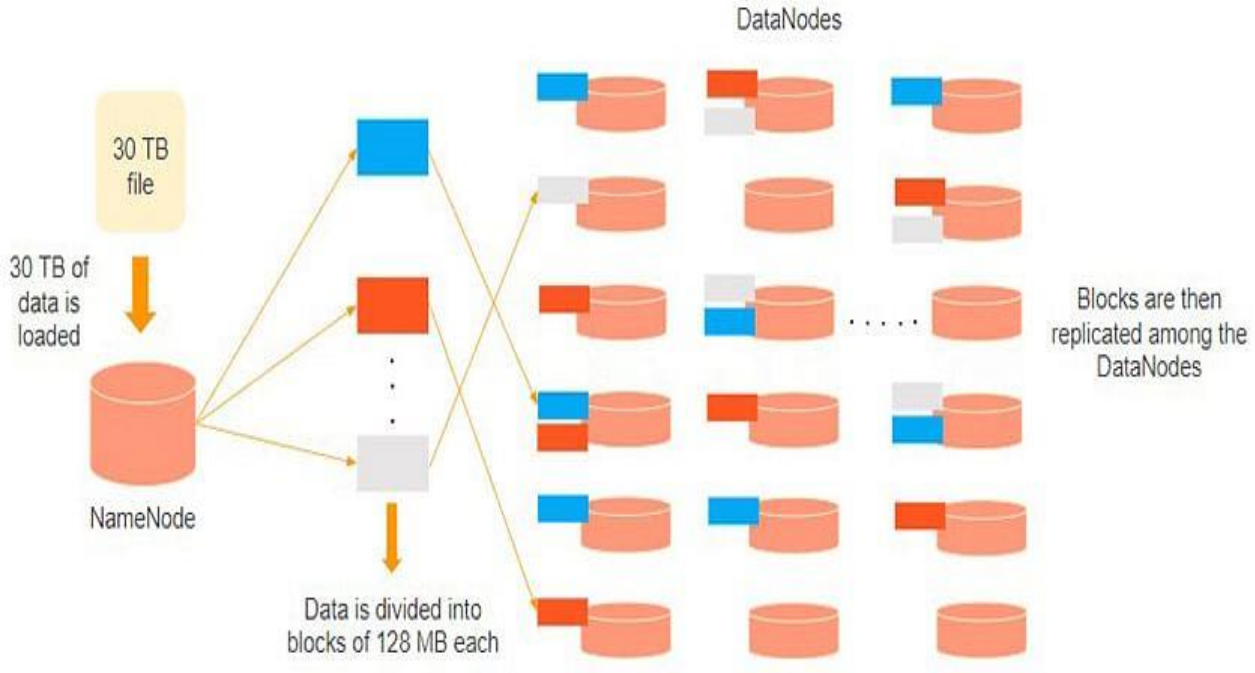


NameNode

- NameNode is the master node that contains the metadata.
- The NameNode is responsible for the workings of the data nodes.
- NameNode is the primary server that manages the file system namespace and controls client access to files.
- The NameNode performs file system namespace operations, including opening, closing and renaming files and directories.
- The NameNode also governs the mapping of blocks to the DataNodes.

DataNode

- The DataNodes are called the slaves.
- The DataNodes read, write, process, and replicate the data.
- They also send signals, known as heartbeats, to the NameNode. These heartbeats show the status of the DataNode.
- While there is only one NameNode, there can be multiple DataNodes.

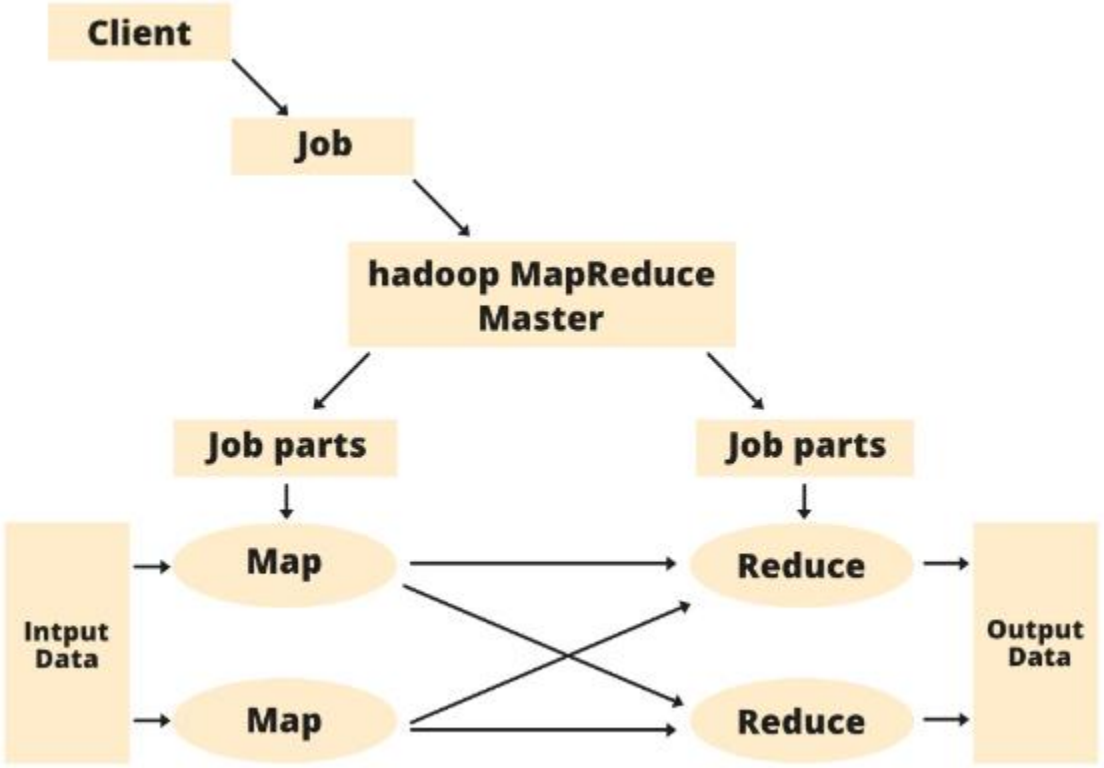


- Consider that 30TB of data is loaded into the NameNode.
- The NameNode distributes it across the DataNodes, and this data is replicated among the DataNodes.
- You can see in the image above that the blue, grey, and red data are replicated among the three DataNodes.
- Replication of the data is performed three times by default. It is done this way, so if a commodity machine fails, you can replace it with a new machine that has the same data.

Hadoop MapReduce

- Hadoop MapReduce is the processing unit of Hadoop.
- In the MapReduce approach, the processing is done at the slave nodes, and the final result is sent to the master node.
- A data containing code is used to process the entire data. This coded data is usually very small in comparison to the data itself.
- You only need to send a few kilobytes worth of code to perform a heavy-duty process on computers.
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** − The map or mapper's job is to process the input data. Generally, the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

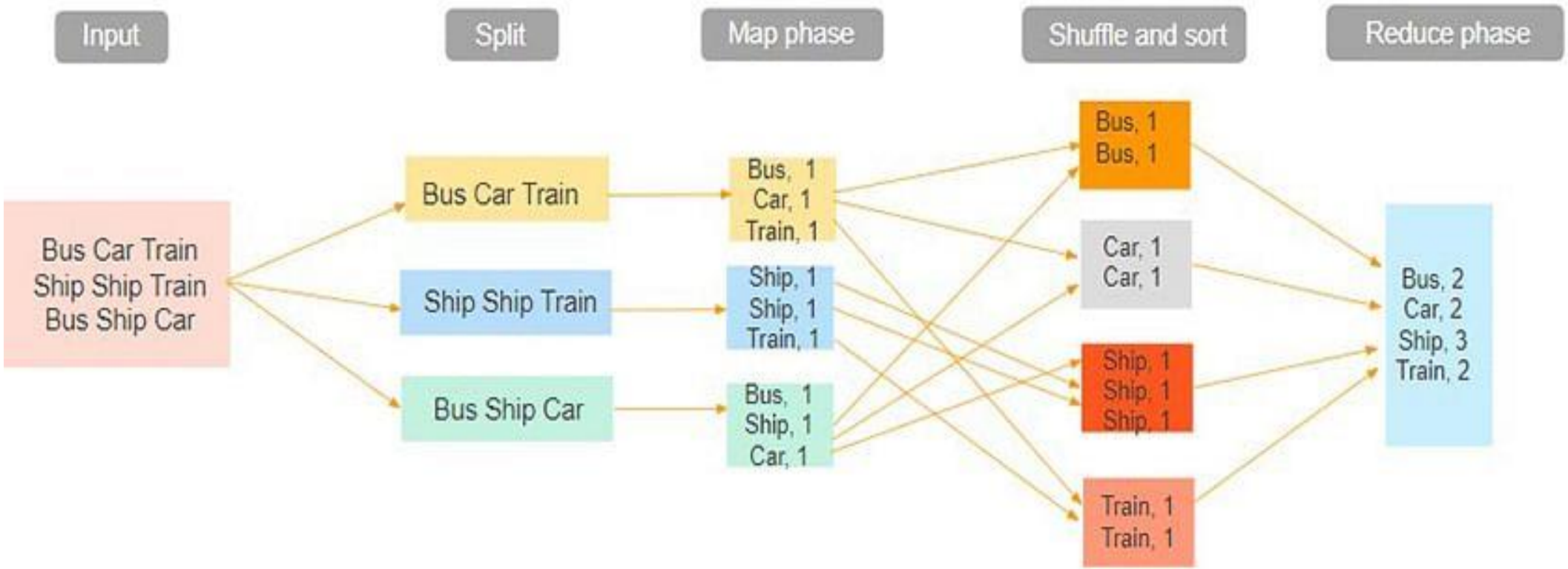
MapReduce Architecture



Components of MapReduce Architecture:

1. **Client:** The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.
2. **Job:** The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.
3. **Hadoop MapReduce Master:** It divides the particular job into subsequent job-parts.
4. **Job-Parts:** The task or sub-jobs that are obtained after dividing the main job. The result of all the job-parts are combined to produce the final output.
5. **Input Data:** The data set that is fed to the MapReduce for processing.
6. **Output Data:** The final result is obtained after the processing.

MapReduce Example



Hadoop YARN

- Hadoop YARN stands for Yet Another Resource Negotiator.
- It is the resource management unit of Hadoop and is available as a component of Hadoop version 2.
- Hadoop YARN acts like an OS to Hadoop. It is a file system that is built on top of HDFS.
- It is responsible for managing cluster resources to make sure you don't overload one machine.
- It performs job scheduling to make sure that the jobs are scheduled in the right place.

Hadoop Ecosystem

Data Flair

oozie
(Work flow)

HCatalog
Table & schema Management

Pig
(Scripting)

Hive
(Sql Query)

mahout
(Machine Learning)

Drill
(Interactive Analysis)

AVRO
(JSON)

Thrift
(Cross Language Service)

APACHE HBASE
HBASE
(Columnar Store)

Sqoop
(Data Collection)

Flume
(Data Collection)

Zookeeper
(Coordination)

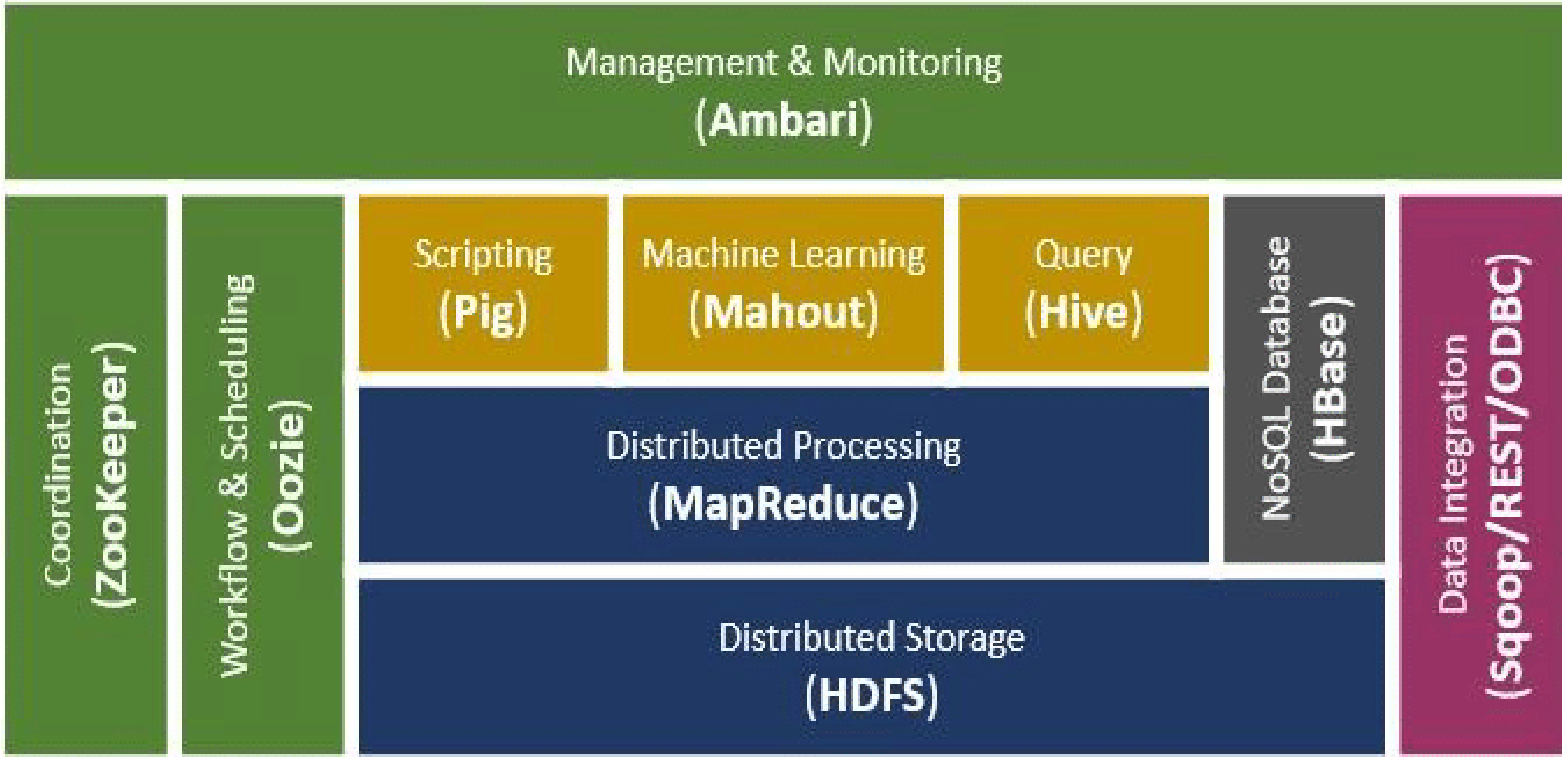
Ambari
Apache Ambari
(Management & Monitoring)

Mapreduce
(Data Processing)

Yarn
(Cluster Resource Management)

HDFS
(Hadoop Distributed File system)

Hadoop Ecosystem

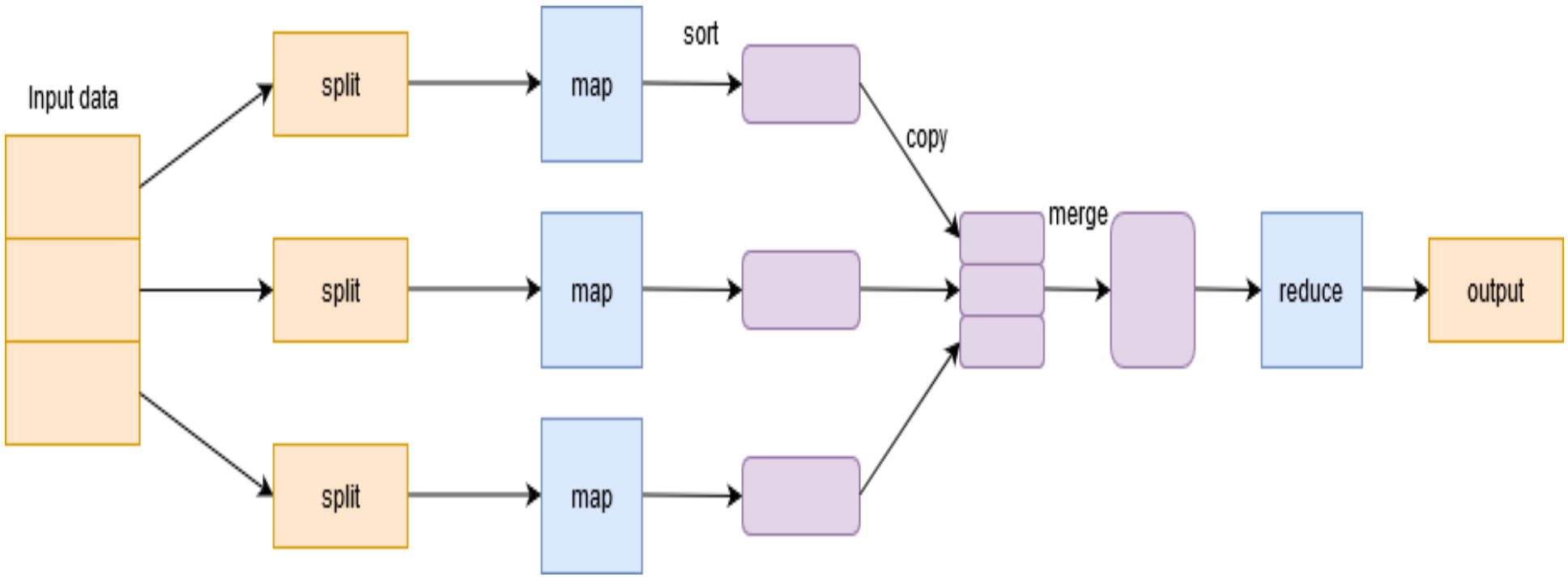


HDFS (Hadoop Distributed File System)

- It is the storage component of Hadoop that stores data in the form of files.
- Each file is divided into blocks of 128MB (configurable) and stores them on different machines in the cluster.
- It has a master-slave architecture with two main components: Name Node and Data Node.
- **Name node** is the master node and there is only one per cluster. Its task is to know where each block belonging to a file is lying in the cluster
- **Data node** is the slave node that stores the blocks of data and there are more than one per cluster. Its task is to retrieve the data as and when required. It keeps in constant touch with the name node through heartbeats.

MapReduce

- To handle Big Data, Hadoop relies on the MapReduce algorithm introduced by Google and makes it easy to distribute a job and run it in parallel in a cluster.
- It essentially divides a single task into multiple tasks and processes them on different machines.
- In layman's terms, it works in a divide-and-conquer manner and runs the processes on the machines to reduce traffic on the network.
- It has two important phases: Map and Reduce.
- **Map phase** filters, groups, and sorts the data. Input data is divided into multiple **splits**. Each map task works on a split of data in parallel on different machines and outputs a key-value pair. The output of this phase is acted upon by the **reduce task** and is known as the **Reduce phase**. It aggregates the data, summarizes the result, and stores it on HDFS.


Input data
split
split
split
map
map
map
sort
copy
merge
reduce
output

YARN

- YARN or Yet Another Resource Negotiator manages resources in the cluster and manages the applications over Hadoop.
- It allows data stored in HDFS to be processed and run by various data processing engines such as batch processing, stream processing, interactive processing, graph processing, and many more.
- This increases efficiency with the use of YARN.

HBase

- HBase is a Column-based NoSQL database.
- It runs on top of HDFS and can handle any type of data.
- It allows for real-time processing and random read/write operations to be performed in the data.

Pig

Apache Pig

- Pig was developed for analyzing large datasets and overcomes the difficulty to write map and reduce functions.
- It consists of two components: Pig Latin and Pig Engine.
- Pig Latin is the Scripting Language that is similar to SQL.
- Pig Engine is the execution engine on which Pig Latin runs.
- Internally, the code written in Pig is converted to MapReduce functions and makes it very easy for programmers who aren't proficient in Java.

Hive

- Hive is a distributed data warehouse system developed by Facebook.
- It allows for easy reading, writing, and managing files on HDFS.
- It has its own querying language for the purpose known as Hive Querying Language (HQL) which is very similar to SQL.
- This makes it very easy for programmers to write MapReduce functions using simple HQL queries.

Sqoop

- A lot of applications still store data in relational databases, thus making them a very important source of data.
- Therefore, Sqoop plays an important part in bringing data from Relational Databases into HDFS.
- The commands written in Sqoop internally converts into MapReduce tasks that are executed over HDFS.
- It works with almost all relational databases like MySQL, Postgres, SQLite, etc.
- It can also be used to export data from HDFS to RDBMS.

Flume

- Flume is an open-source, reliable, and available service used to efficiently collect, aggregate, and move large amounts of data from multiple data sources into HDFS.
- It can collect data in real-time as well as in batch mode.
- It has a flexible architecture and is fault-tolerant with multiple recovery mechanisms.

Kafka

- There are a lot of applications generating data and a commensurate number of applications consuming that data. But connecting them individually is a tough task. That's where Kafka comes in.
- It sits between the applications generating data (Producers) and the applications consuming data (Consumers).
- Kafka is distributed and has in-built partitioning, replication, and fault-tolerance.
- It can handle streaming data and also allows businesses to analyze data in real-time.

Oozie

- Oozie is a workflow scheduler system that allows users to link jobs written on various platforms like MapReduce, Hive, Pig, etc.
- Using Oozie you can schedule a job in advance and can create a pipeline of individual jobs to be executed sequentially or in parallel to achieve a bigger task.
- For example, you can use Oozie to perform ETL operations on data and then save the output in HDFS.

Zookeeper

APACHE ZooKeeper™

- In a Hadoop cluster, coordinating and synchronizing nodes can be a challenging task. Therefore, Zookeeper is the perfect tool for the problem.
- It is an open-source, distributed, and centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services across the cluster.

Mahout

- Mahout offers a platform to develop machine learning software that can be scaled.
- Machine learning algorithms enable the creation of self-learning systems that learn by themselves without having to be explicitly programmed.
- Based on the user's behavior patterns, data and previous experiences, it can make crucial choices.
- It can be described as an ancestor from Artificial Intelligence (AI).
- Mahout is a collaborative filtering system as well as clustering and classification.

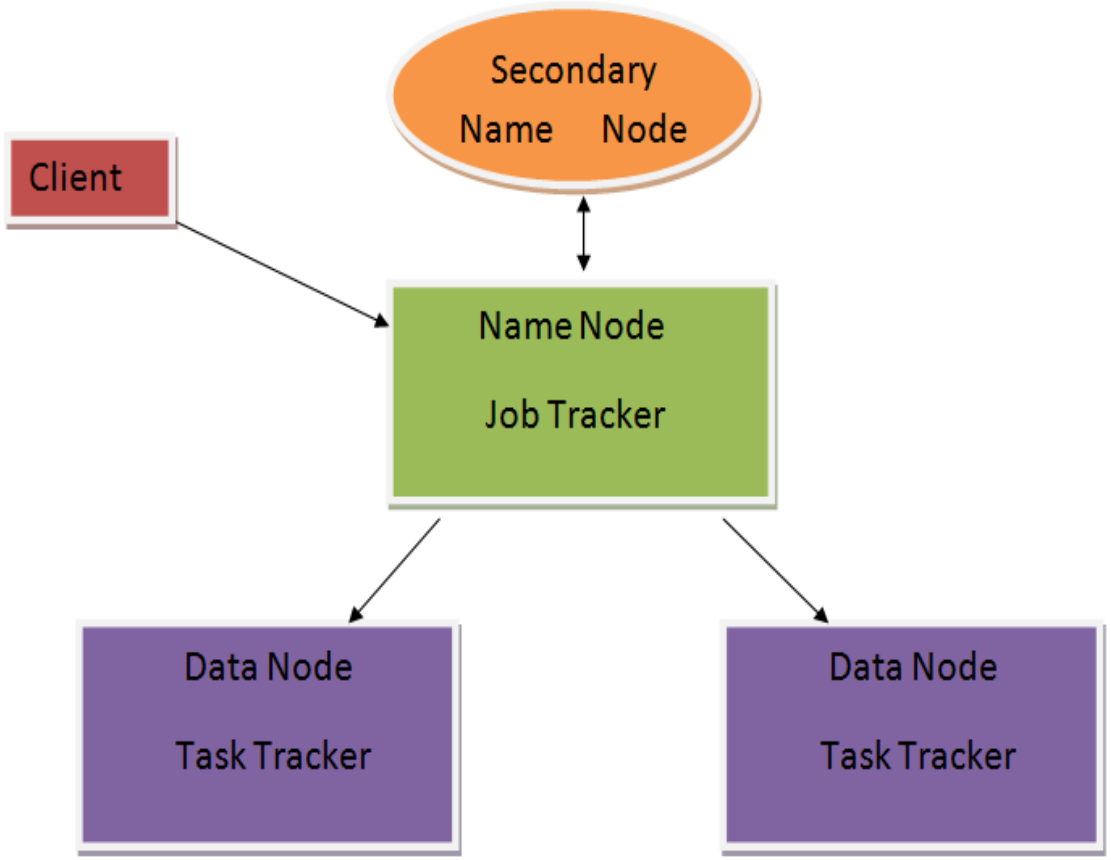
Spark

- Spark is an alternative framework to Hadoop built on Scala but supports varied applications written in Java, Python, etc.
- Compared to MapReduce it provides in-memory processing which accounts for faster processing.
- In addition to batch processing offered by Hadoop, it can also handle real-time processing.

Ambari

- Ambari is an Apache Software Foundation Project which seeks to make the ecosystem Hadoop easier to manage.
- It is a software solution for ***provisioning, and managing Apache Hadoop clusters***.

Physical Architecture of Hadoop



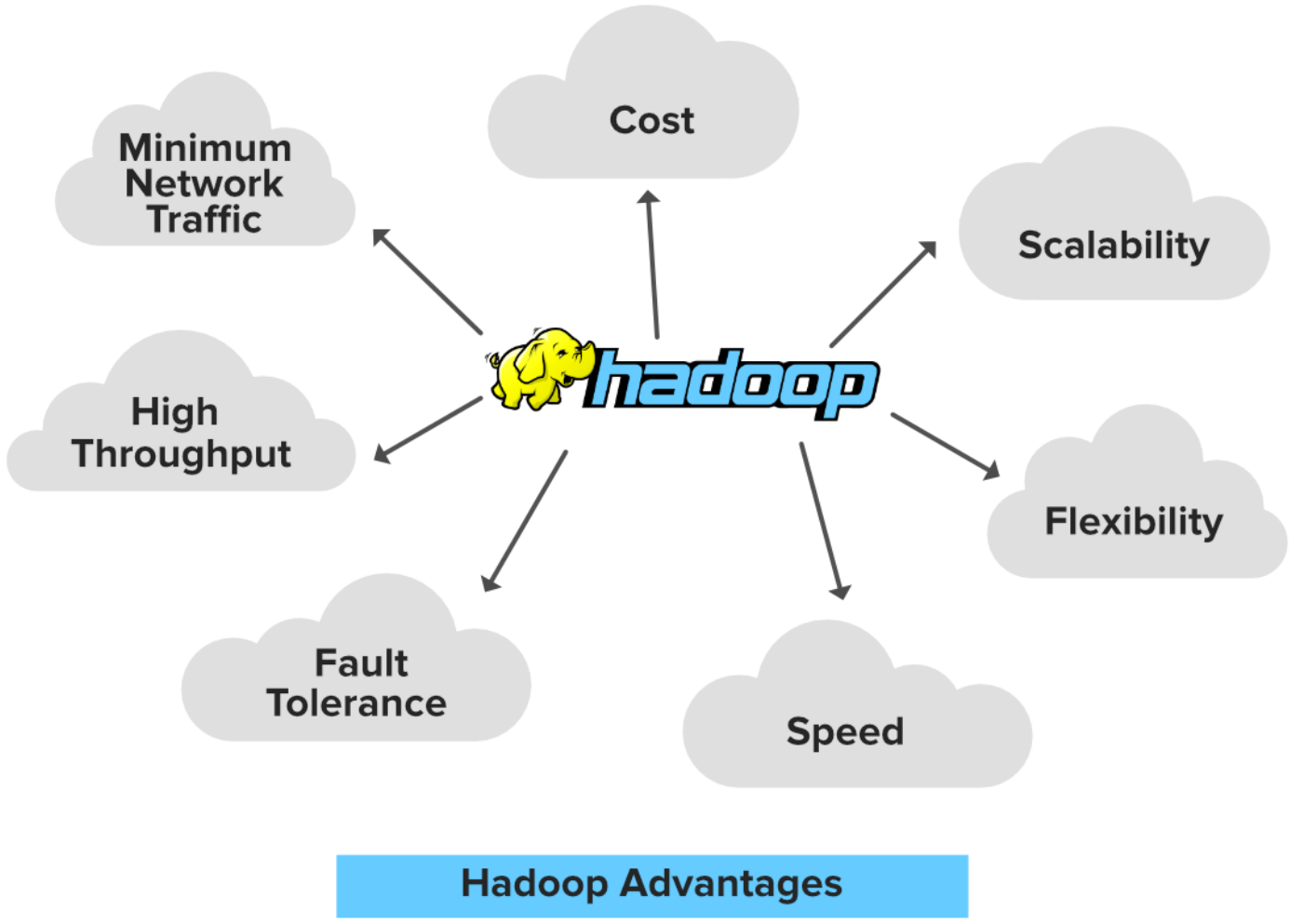
Description of Hadoop components

- **Name Node**
  - It is the master of HDFS (Hadoop file system).
  - Contains Job Tracker, which keeps tracks of a file distributed to different data nodes.
  - Failure of Name Node will lead to the failure of the full Hadoop system.
- **Data node**
  - Data node is the slave of HDFS.
  - A data node can communicate with each other through the name node to avoid replication in the provided task.
  - Data nodes update the change to the data node.
- **Job Tracker**
  - Determines which file to process.
  - There can be only one job tracker for per Hadoop cluster.
- **Task Tracker**
  - Only single task tracker is present per slave node.
  - Performs tasks given by job tracker and also continuously communicates with the job tracker.
- **SSN (Secondary Name Node)**
  - Its main purpose is to monitor.
  - One SSN is present per cluster.

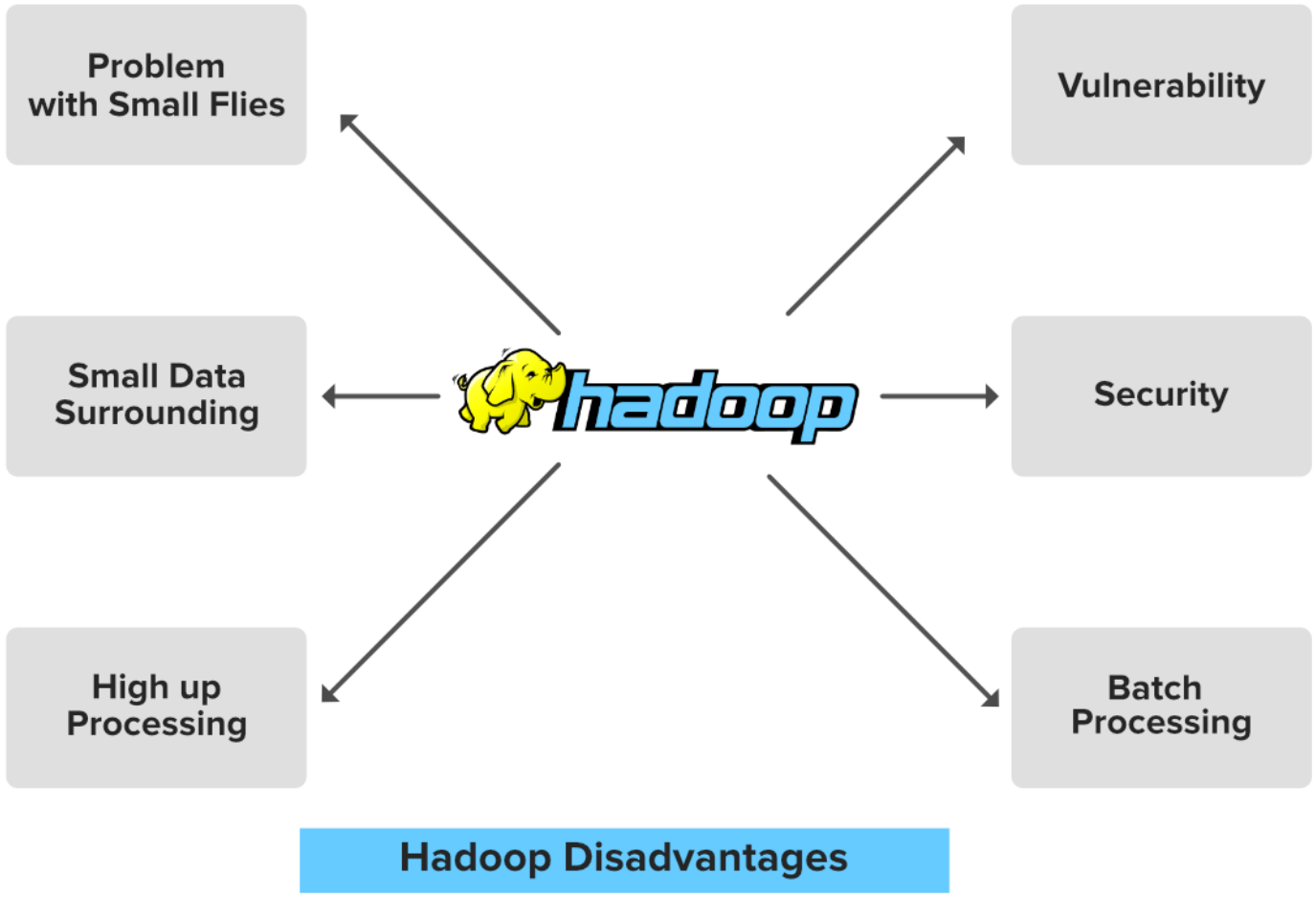
Working

1. When the client submit his job, it will go to the NameNode.
2. Now, NameNode will decide whether to accept the job or not.
3. After accepting the job, the NameNode will transfer the job to the job tracker.
4. Job tracker will divide the job into components and transfer them to DataNodes.
5. Now, DataNodes will further transfer the jobs to the task tracker.
6. The actual processing will be done here, means the execution of the job submitted is done here.
7. The job tracker continuously communicates with the task trackers. In the case in any moment job trackers do not get a reply from any of the task trackers, it considers that it failed and transfers its work to another one.
8. Then, after completing the part of the jobs assigned to them, the task tracker will submit the completed task to the job tracker via the DataNode.
9. The task of secondary NameNode is to just monitor the whole process ongoing.
10. There is no fixed number of data nodes, it can be as much as required or made.

Hadoop Advantages



Limitations of Hadoop



Hadoop Installation

Steps to Install Hadoop

•Install Java JDK 1.8
•Download Hadoop and extract and place under C drive
•Set Path in Environment Variables
•Config files under Hadoop directory
•Create folder datanode and namenode under data directory
•Edit HDFS and YARN files
•Set Java Home environment in Hadoop environment
•Setup Complete. Test by executing start-all.cmd

There are two ways to install Hadoop, i.e.

•Single node
•Multi node

Single node cluster means only one DataNode running and setting up all the NameNode, DataNode, ResourceManager and NodeManager on a single machine.

While in a Multi node cluster, there are more than one DataNode running and each DataNode is running on different machines. The multi node cluster is practically used in organizations for analyzing Big Data. In real time when we deal with petabytes of data, it needs to be distributed across hundreds of machines to be processed. Thus, here we use multi node cluster.

Setting up a single node Hadoop cluster

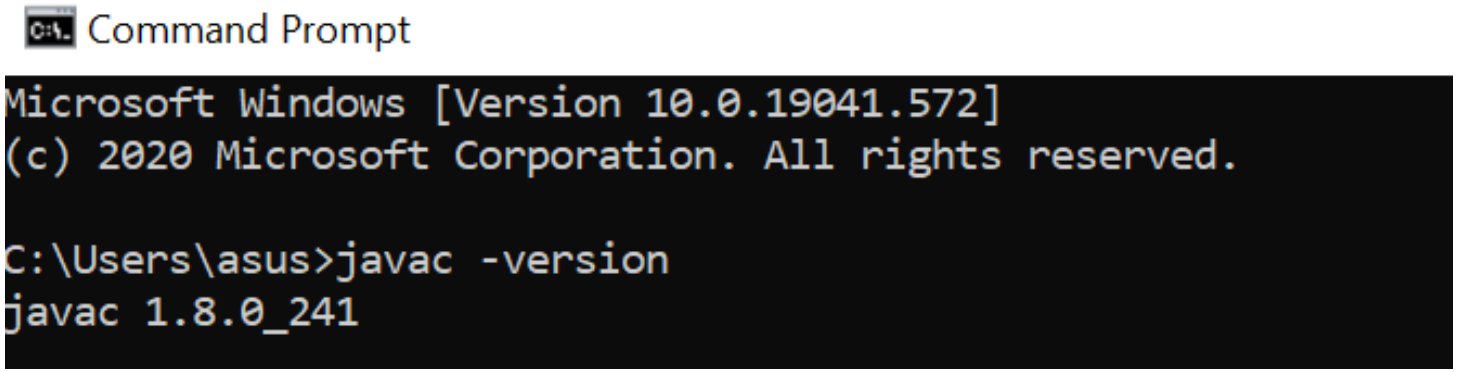
- Prerequisites to install Hadoop on windows
- VIRTUAL BOX (For Linux): it is used for installing the operating system on it.
- OPERATING SYSTEM: You can install Hadoop on Windows or Linux based operating systems. Ubuntu and CentOS are very commonly used.
- JAVA: You need to install the Java 8 package on your system.
- HADOOP: You require Hadoop’s latest version

1. Install Java

- Java JDK Link to download

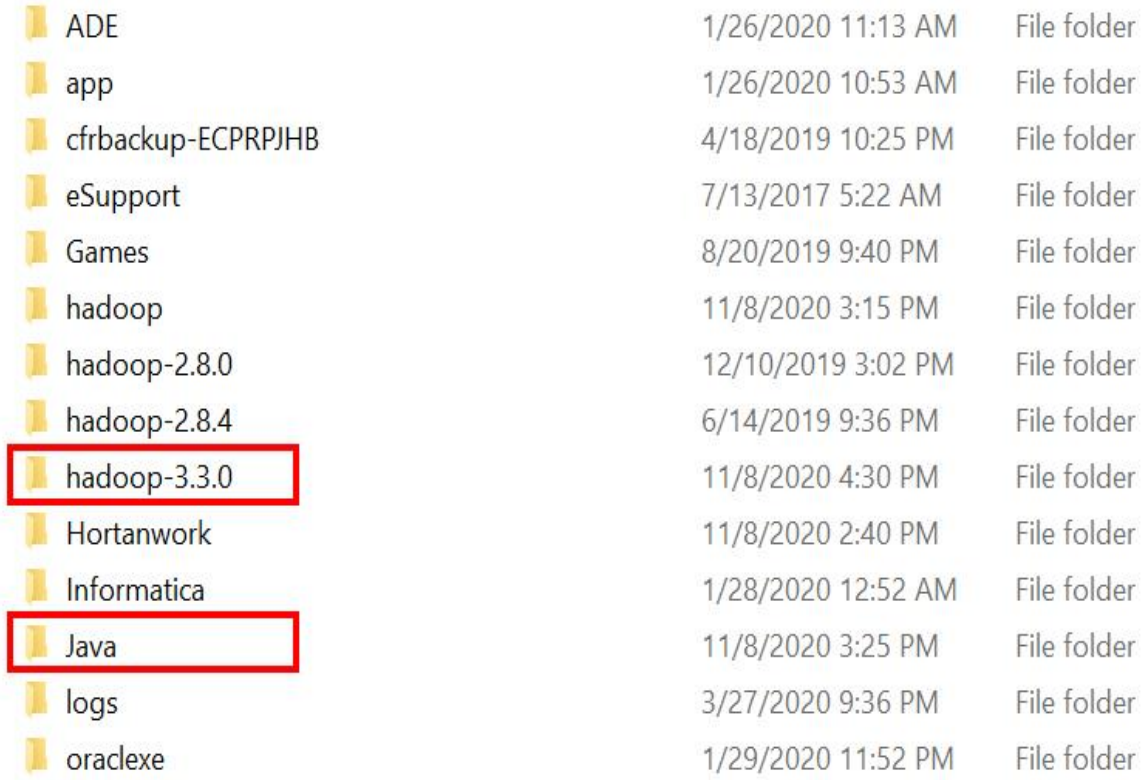
https://www.oracle.com/java/technologies/javase-jdk8-downloads.html

- extract and install Java in C:\Java
- open cmd and type -> javac -version



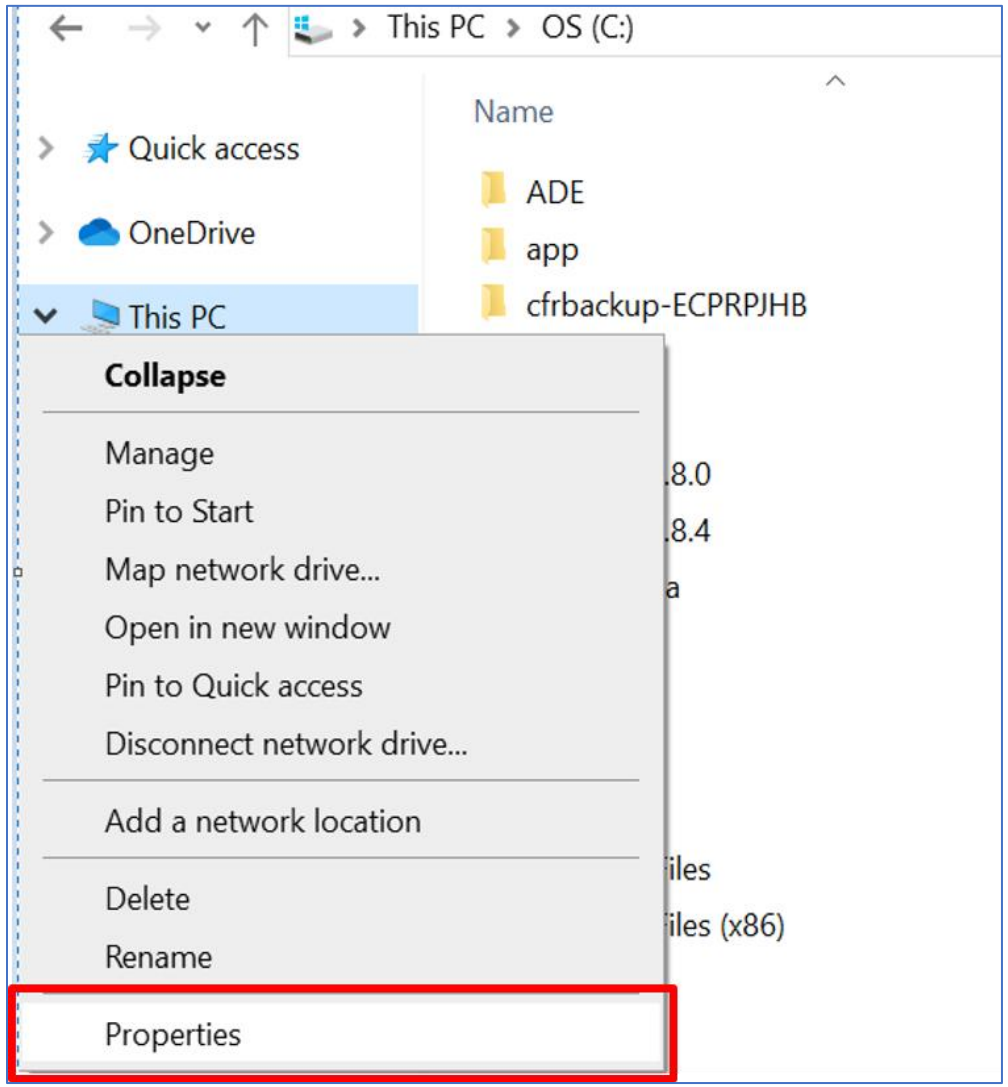
2. Download Hadoop

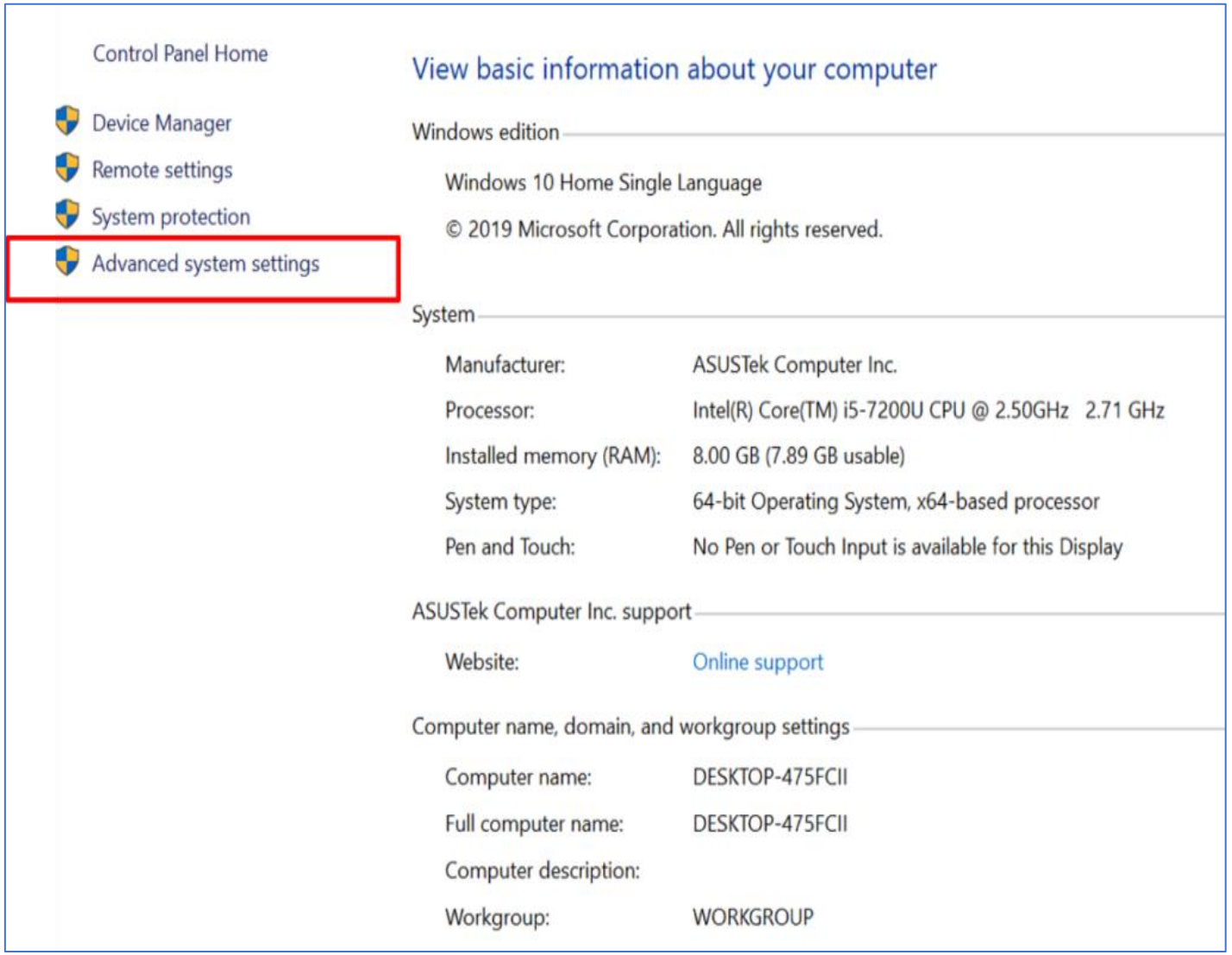
- https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
- **extract to C:\Hadoop**

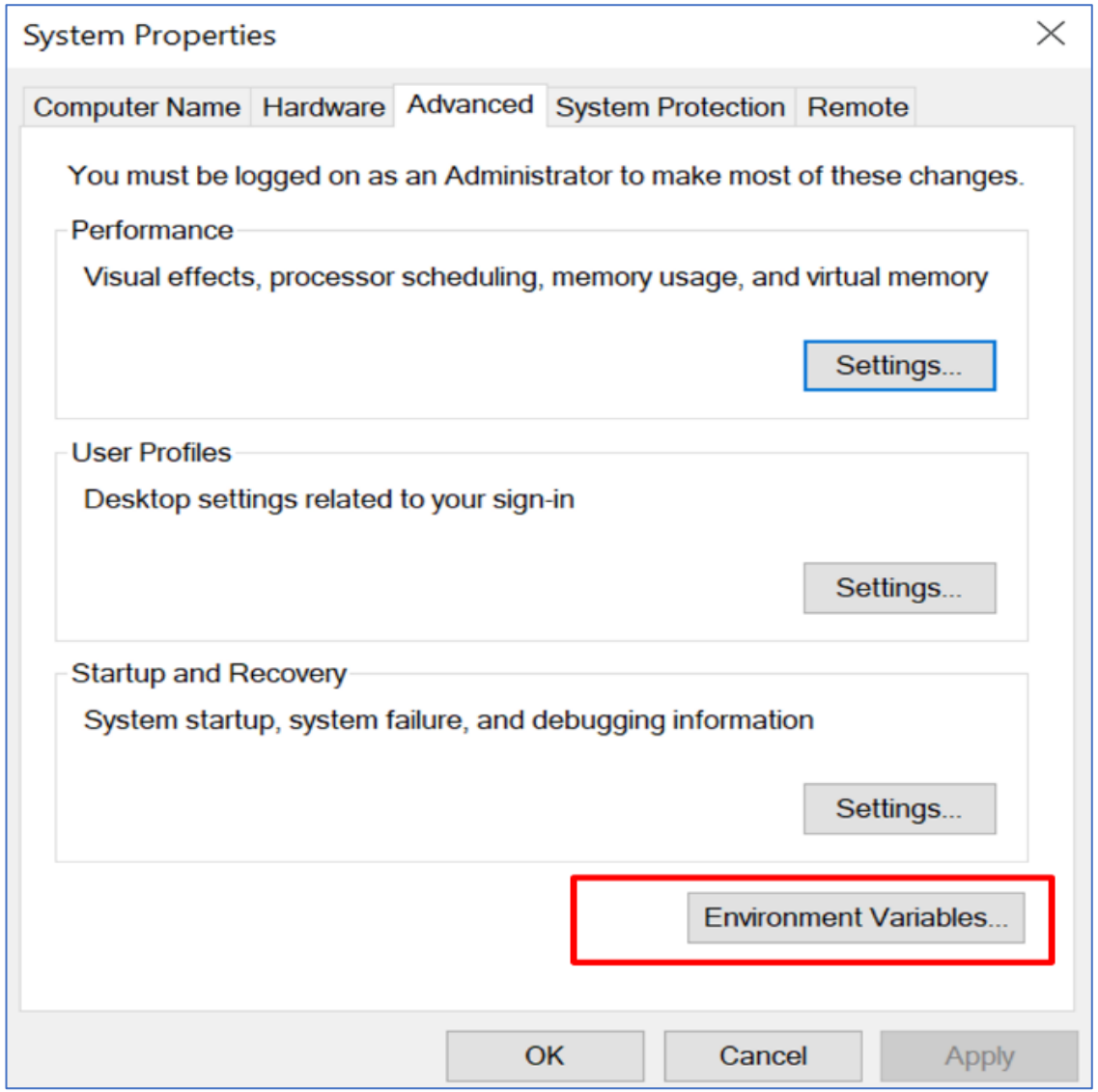


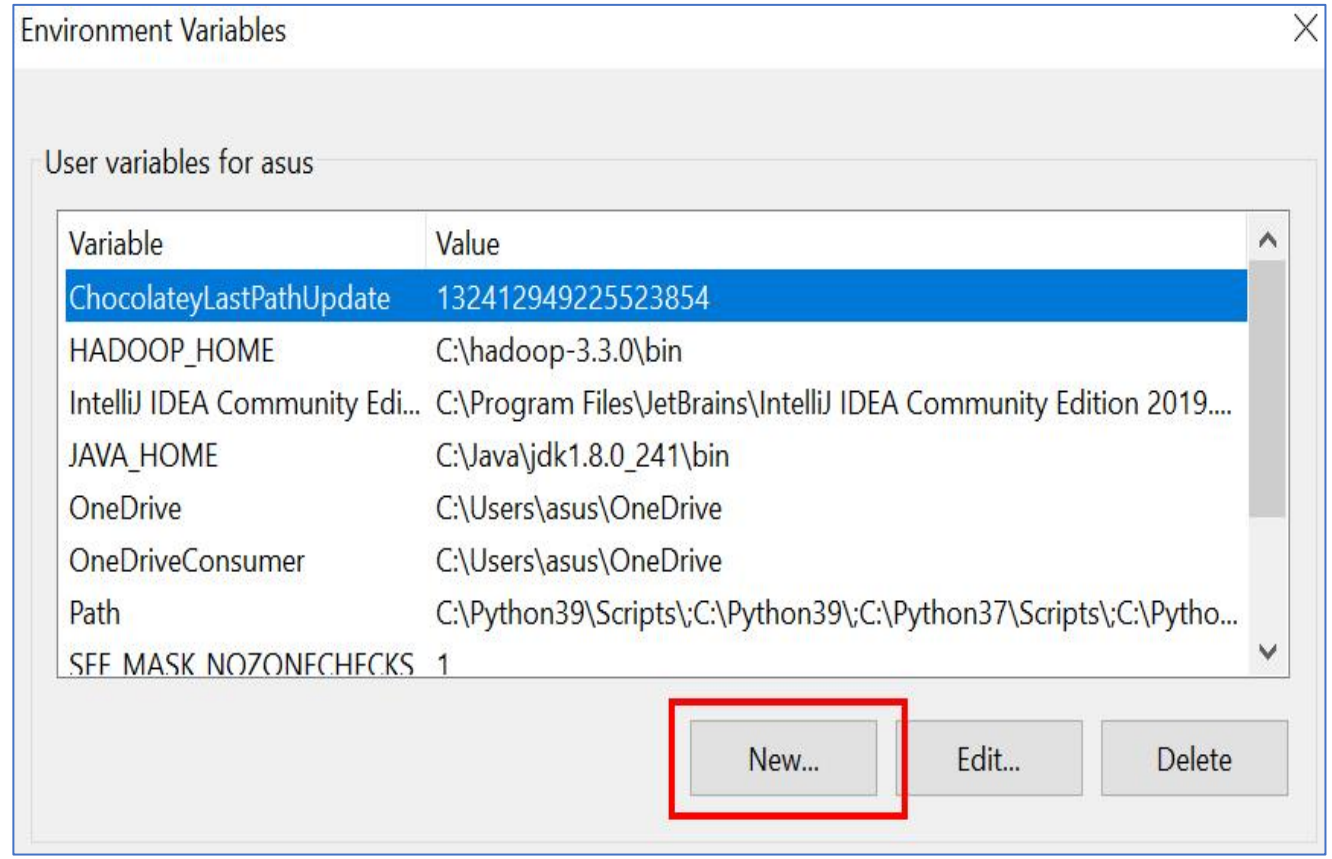
3.Set the path JAVA_HOME Environment variable

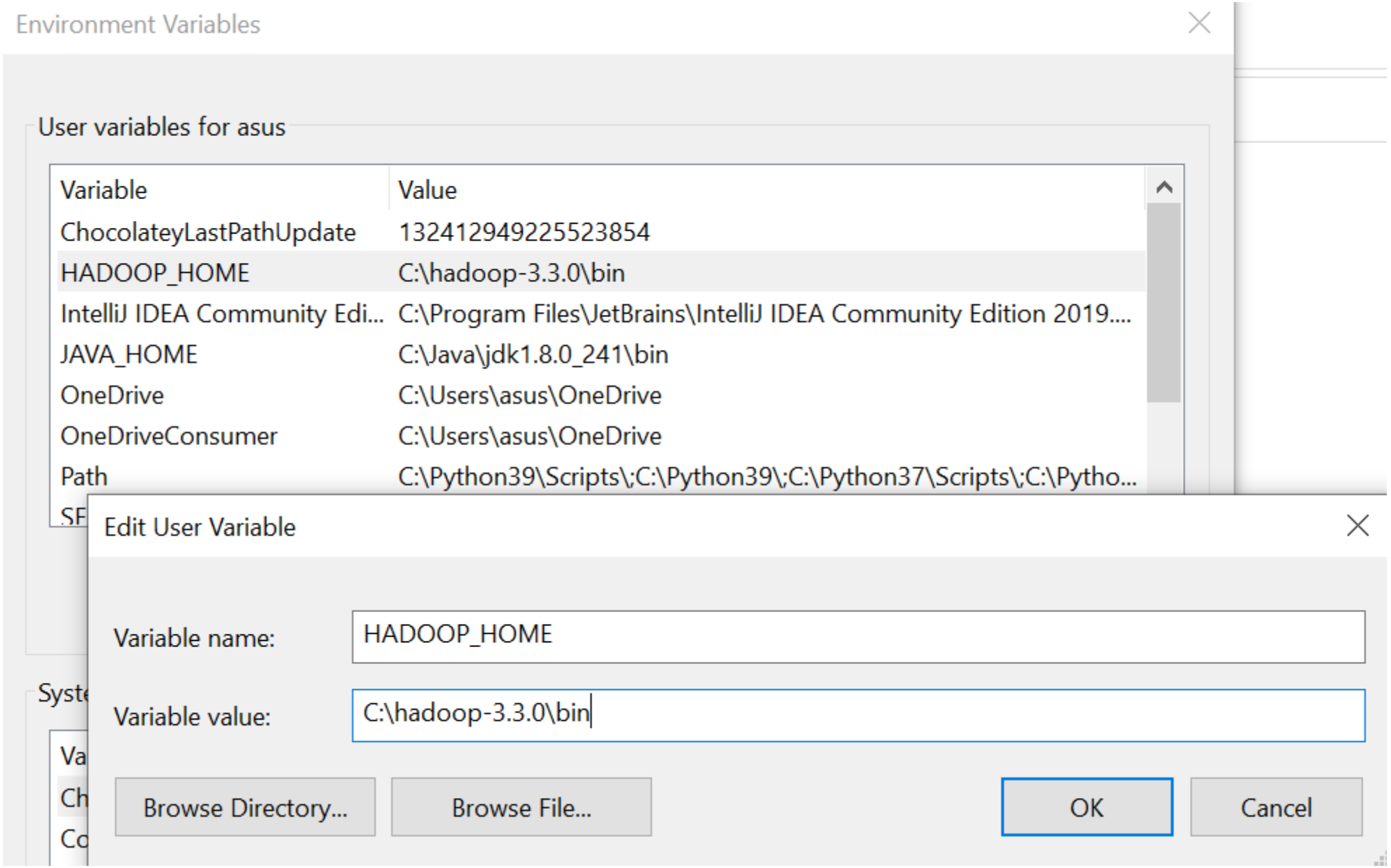
4.Set the path HADOOP_HOME Environment variable

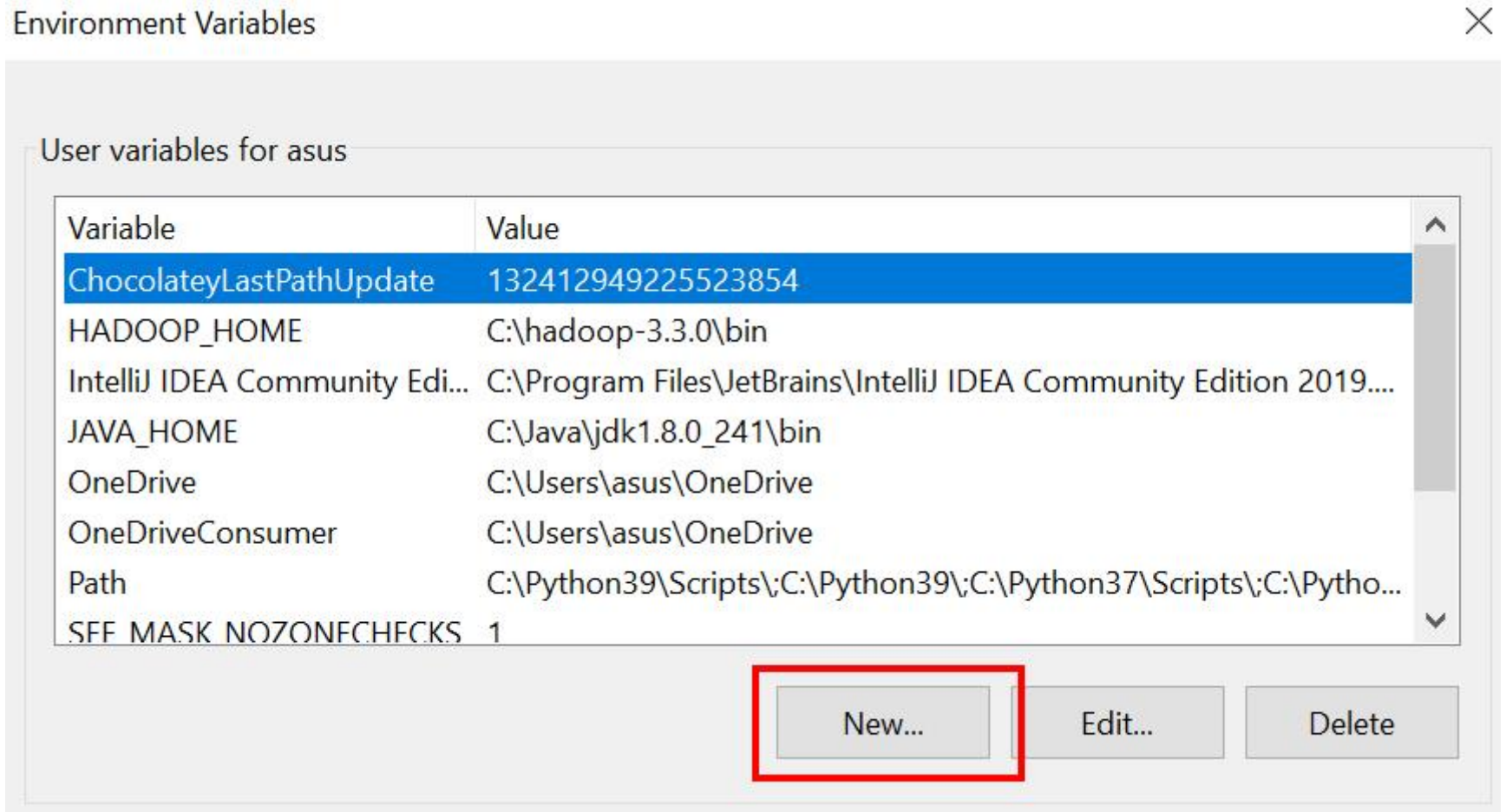


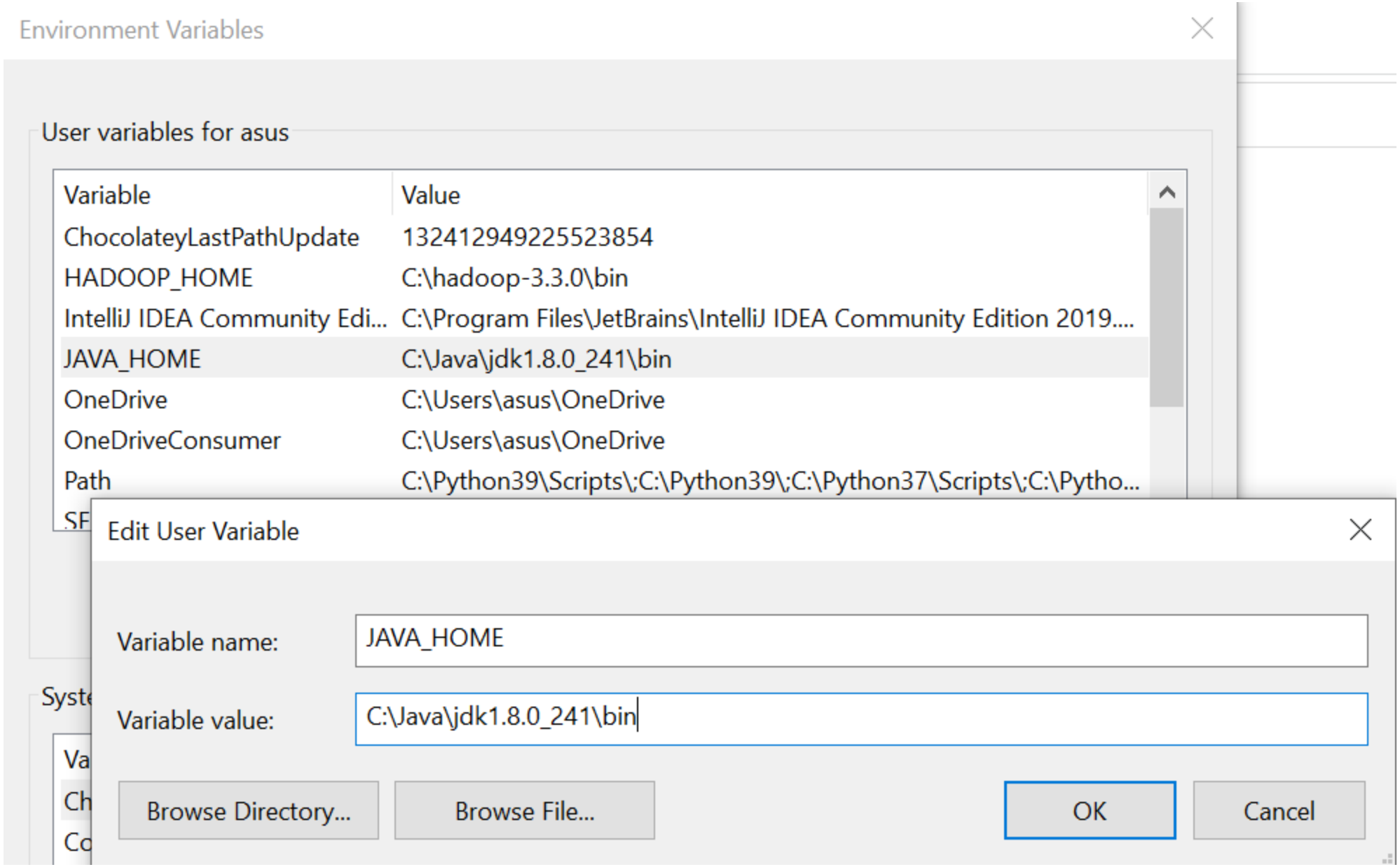



System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply


Environment Variables
User variables for asus
Variable
Value
ChocolateyLastPathUpdate
132412949225523854
HADOOP_HOME
C:\hadoop-3.3.0\bin
IntelliJ IDEA Community Edi...
C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019....
JAVA_HOME
C:\Java\jdk1.8.0_241\bin
OneDrive
C:\Users\asus\OneDrive
OneDriveConsumer
C:\Users\asus\OneDrive
Path
C:\Python39\Scripts\;C:\Python39\;C:\Python37\Scripts\;C:\Pytho...
New...
Edit...
Delete


Environment Variables
User variables for asus
Variable
Value
ChocolateyLastPathUpdate
132412949225523854
HADOOP_HOME
C:\hadoop-3.3.0\bin
IntelliJ IDEA Community Edi...
C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019....
JAVA_HOME
C:\Java\jdk1.8.0_241\bin
OneDrive
C:\Users\asus\OneDrive
OneDriveConsumer
C:\Users\asus\OneDrive
Path
C:\Python39\Scripts\;C:\Python39\;C:\Python37\Scripts\;C:\Pytho...
Edit User Variable
Variable name:
HADOOP_HOME
Variable value:
C:\hadoop-3.3.0\bin
Browse Directory...
Browse File...
OK
Cancel


Environment Variables
User variables for asus
Variable
Value
ChocolateyLastPathUpdate
132412949225523854
HADOOP_HOME
C:\hadoop-3.3.0\bin
IntelliJ IDEA Community Edi...
C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019....
JAVA_HOME
C:\Java\jdk1.8.0_241\bin
OneDrive
C:\Users\asus\OneDrive
OneDriveConsumer
C:\Users\asus\OneDrive
Path
C:\Python39\Scripts\;C:\Python39\;C:\Python37\Scripts\;C:\Pytho...
SFE_MASK_NOZONECHECKS
1
New...
Edit...
Delete


Environment Variables
User variables for asus
Variable
Value
ChocolateyLastPathUpdate
132412949225523854
HADOOP_HOME
C:\hadoop-3.3.0\bin
IntelliJ IDEA Community Edi...
C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019....
JAVA_HOME
C:\Java\jdk1.8.0_241\bin
OneDrive
C:\Users\asus\OneDrive
OneDriveConsumer
C:\Users\asus\OneDrive
Path
C:\Python39\Scripts\;C:\Python39\;C:\Python37\Scripts\;C:\Pytho...
Edit User Variable
Variable name:
JAVA_HOME
Variable value:
C:\Java\jdk1.8.0_241\bin
Browse Directory...
Browse File...
OK
Cancel

User variables for asus

Variable	Value
HADOOP_HOME	C:\hadoop-2.8.4\hadoop-2.8.4\bin
IntelliJ IDEA Community Edi...	C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019....
JAVA_HOME	C:\Java\jdk-12.0.1\bin
OneDrive	C:\Users\asus\OneDrive
OneDriveConsumer	C:\Users\asus\OneDrive
Path	C:\Python37\Scripts\;C:\Python37\;C:\Python27\Scripts;C:\Pytho...
SEE_MASK_NOZONECHECKS	1
TEMP	C:\Users\asus\AppData\Local\Temp

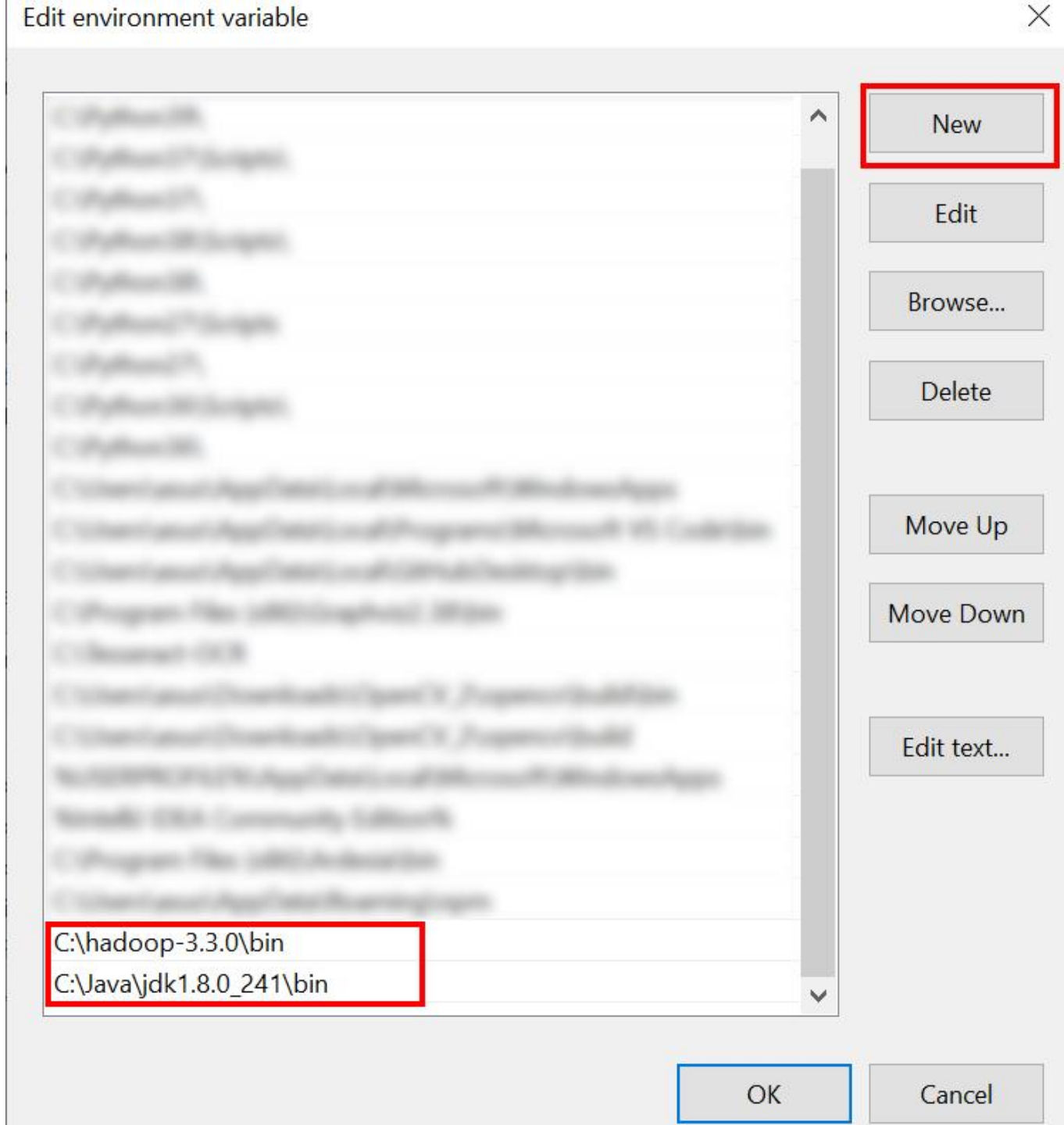
New... Edit... Delete

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DokanLibrary1	C:\Program Files\Dokan\Dokan Library-1.2.0\
DokanLibrary1_LibraryPath_...	C:\Program Files\Dokan\Dokan Library-1.2.0\lib\
DokanLibrary1_LibraryPath_...	C:\Program Files\Dokan\Dokan Library-1.2.0\x86\lib\
DriverData	C:\Windows\System32\Drivers\DriverData
INFA_TRUSTSTORE	C:\Informatica\9.6.1\clients\shared\security
NUMBER_OF_PROCESSORS	4
OS	Windows_NT

New... Edit... Delete

OK Cancel


Edit environment variable
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
C:\hadoop-3.3.0\bin
C:\Java\jdk1.8.0_241\bin
OK
Cancel

5. Configurations

Edit file C:/Hadoop-3.3.0/etc/hadoop/core-site.xml,
paste the xml code in folder and save

```
<configuration>
  <property>
     <name>fs.defaultFS</name>
     <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Rename “mapred-site.xml.template” to “mapred-site.xml” and edit this file C:/Hadoop-3.3.0/etc/hadoop/mapred-site.xml, paste xml code and save this file.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Create folder “data” under “C:\Hadoop-3.3.0”
Create folder “datanode” under “C:\Hadoop-3.3.0\data”
Create folder “namenode” under “C:\Hadoop-3.3.0\data”

Edit file C:\Hadoop-3.3.0/etc/hadoop/hdfs-site.xml, paste xml code and save this file.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/hadoop-3.3.0/data/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/hadoop-3.3.0/data/datanode</value>
  </property>
</configuration>
```

Edit file C:/Hadoop-3.3.0/etc/hadoop/yarn-site.xml, paste xml code and save this file.

```
<configuration>
  <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
  </property>
  <property>
        <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

Edit file C:/Hadoop-3.3.0/etc/hadoop/hadoop-env.cmd by closing the command line “JAVA_HOME=%JAVA_HOME%” instead of set “JAVA_HOME=C:\Java”

6. Hadoop Configurations

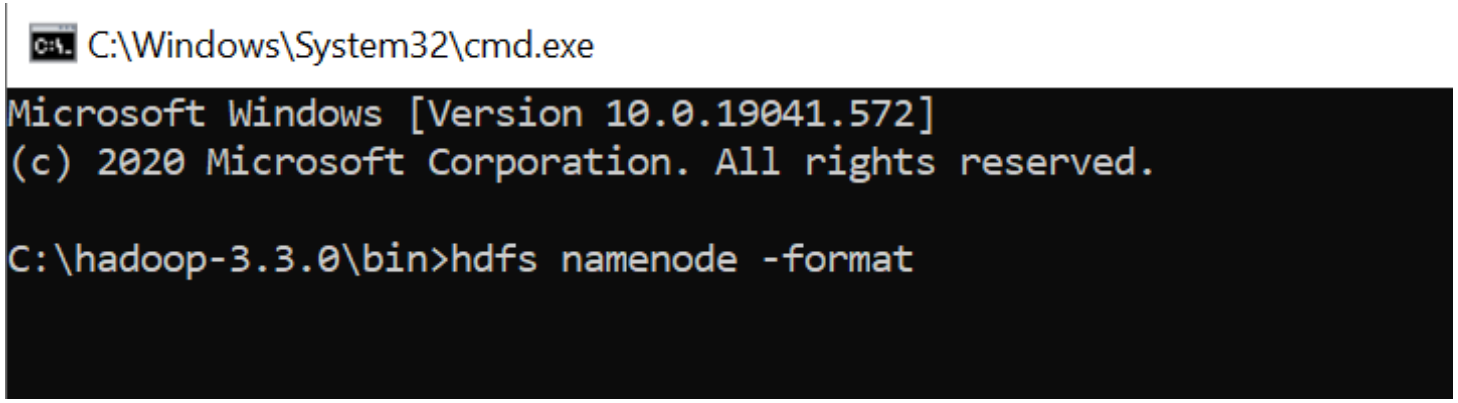
- **Download**

https://github.com/brainmentorspvtltd/BigData_RDE/blob/master/Hadoop%20Configuration.zip

- **or (for hadoop 3)**

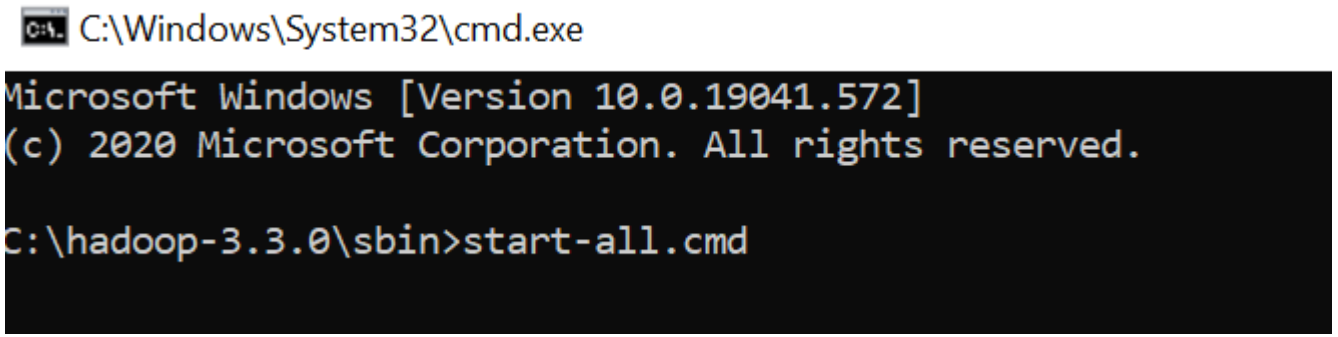
https://github.com/s911415/apache-hadoop-3.1.0-winutils

- **Copy folder bin and replace existing bin folder in C:\Hadoop-3.3.0\bin**
- **Format the NameNode**
- **Open cmd and type command “hdfs namenode –format”**



7. Testing

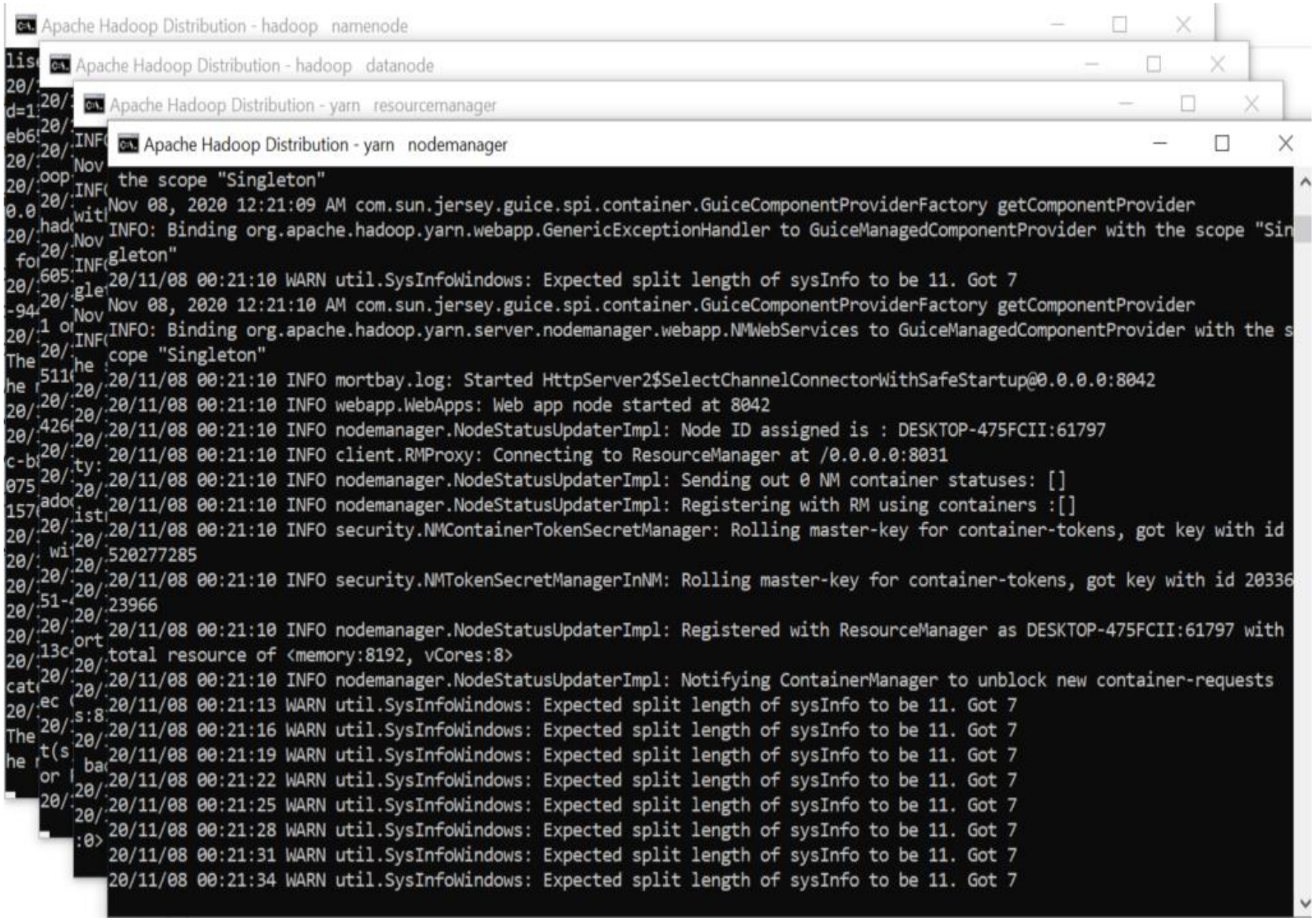
- **Open cmd and change directory to C:\Hadoop-3.3.0\sbin**
- **type start-all.cmd**



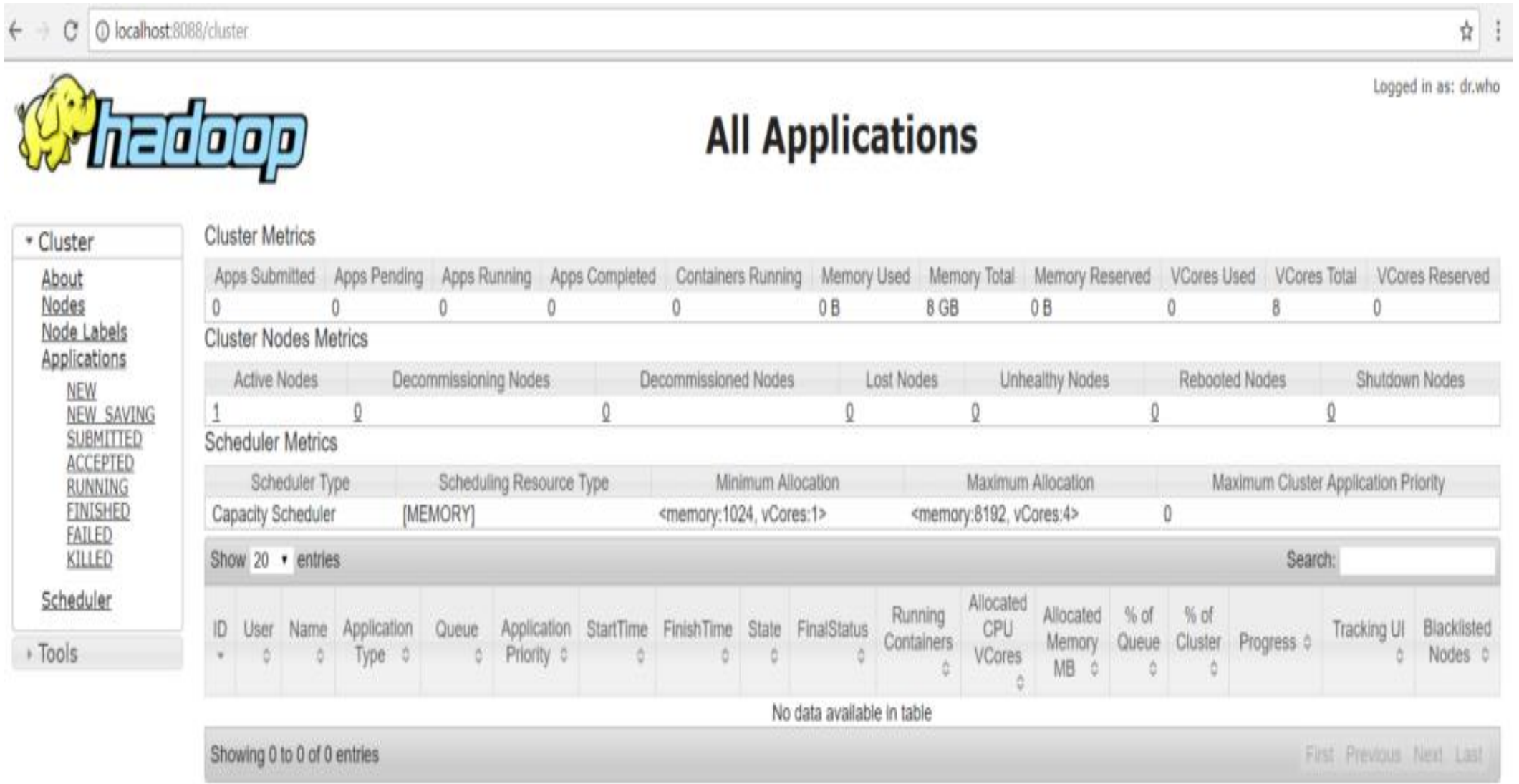
(Or you can start like this)
– Start namenode and datanode with this command
– type start-dfs.cmd
– Start yarn through this command
– type start-yarn.cmd

Make sure these apps are running

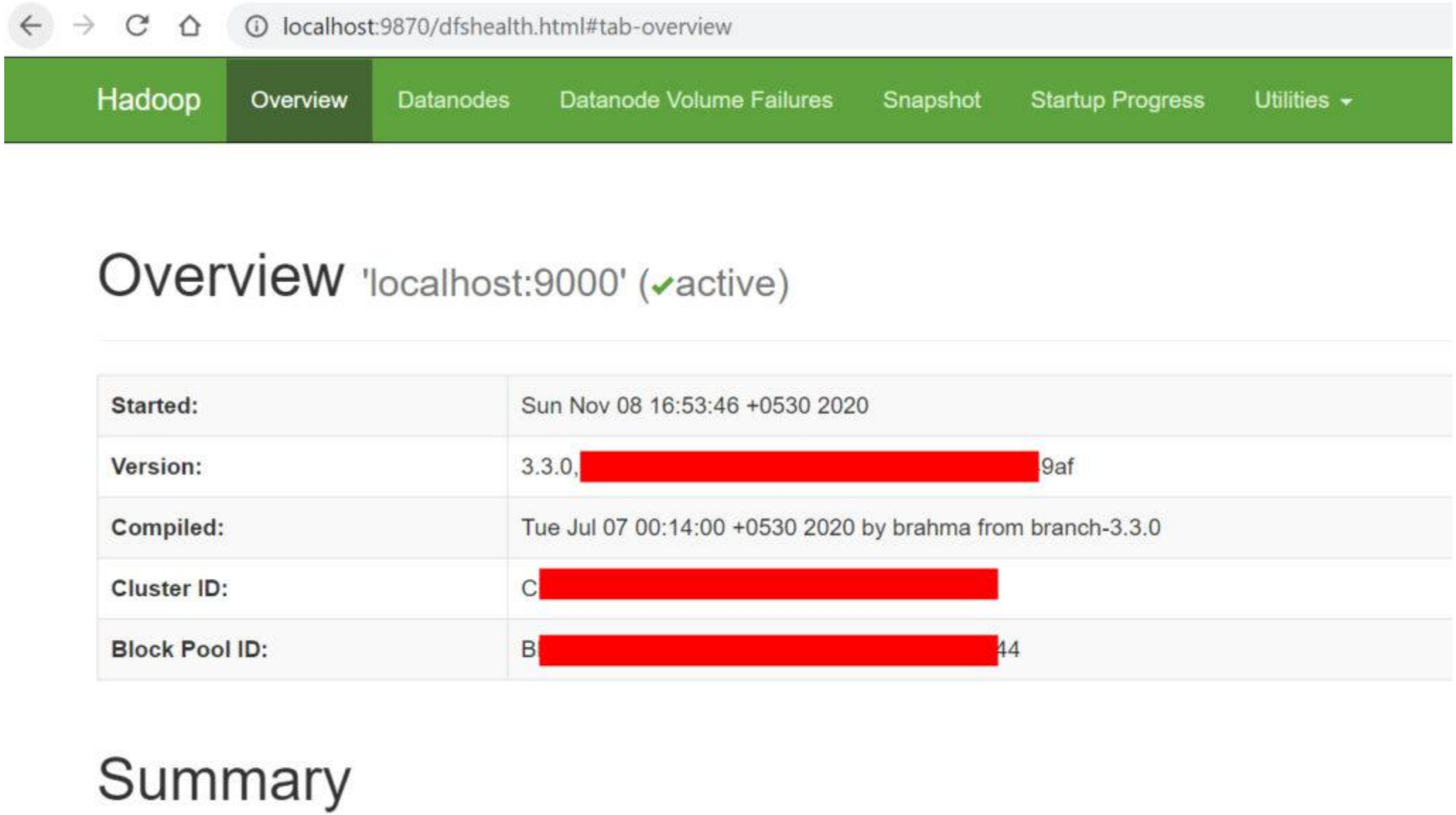
- Hadoop Namenode
- Hadoop datanode
- YARN Resource Manager
- YARN Node Manager



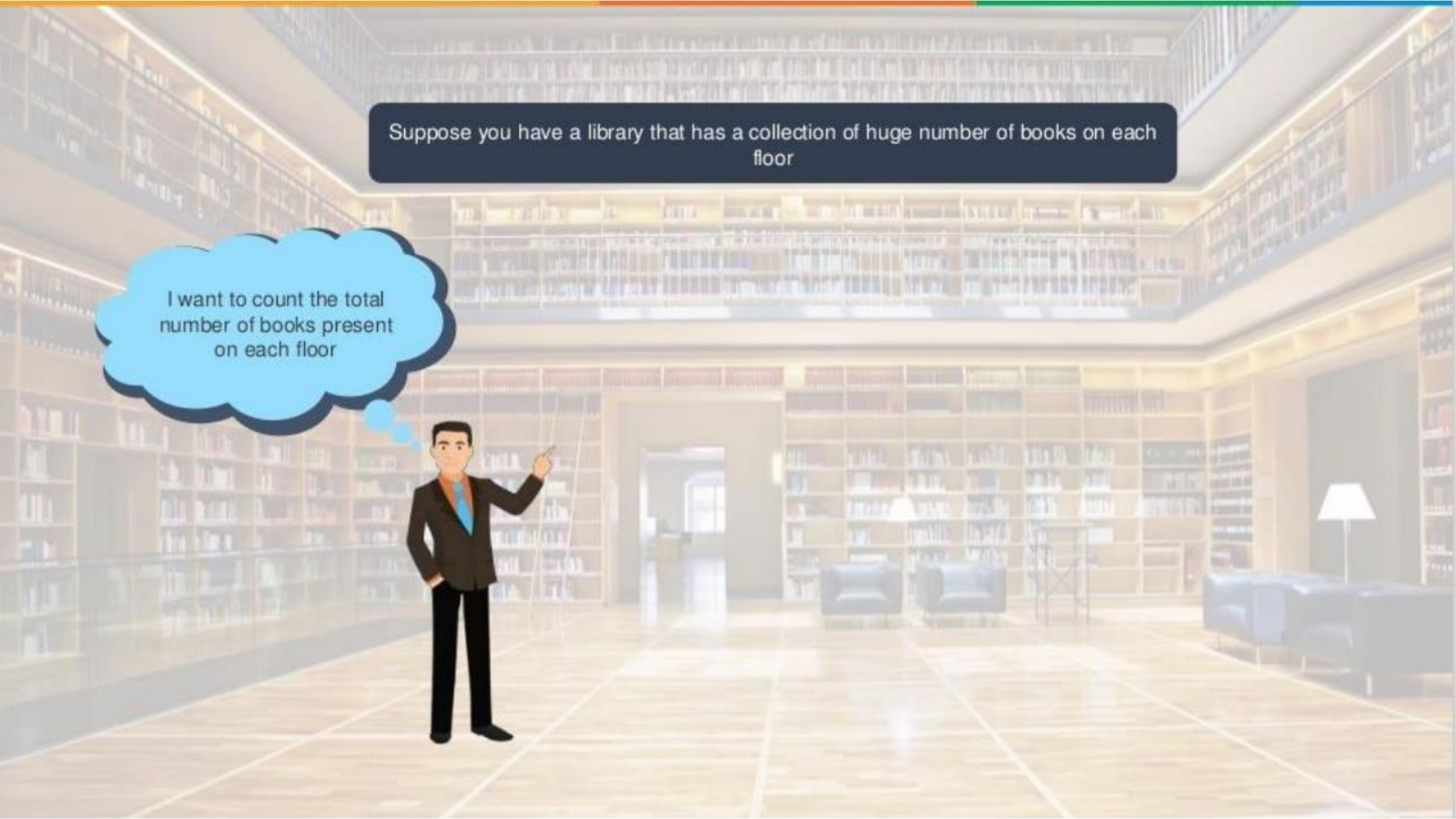
Open: http://localhost:8088

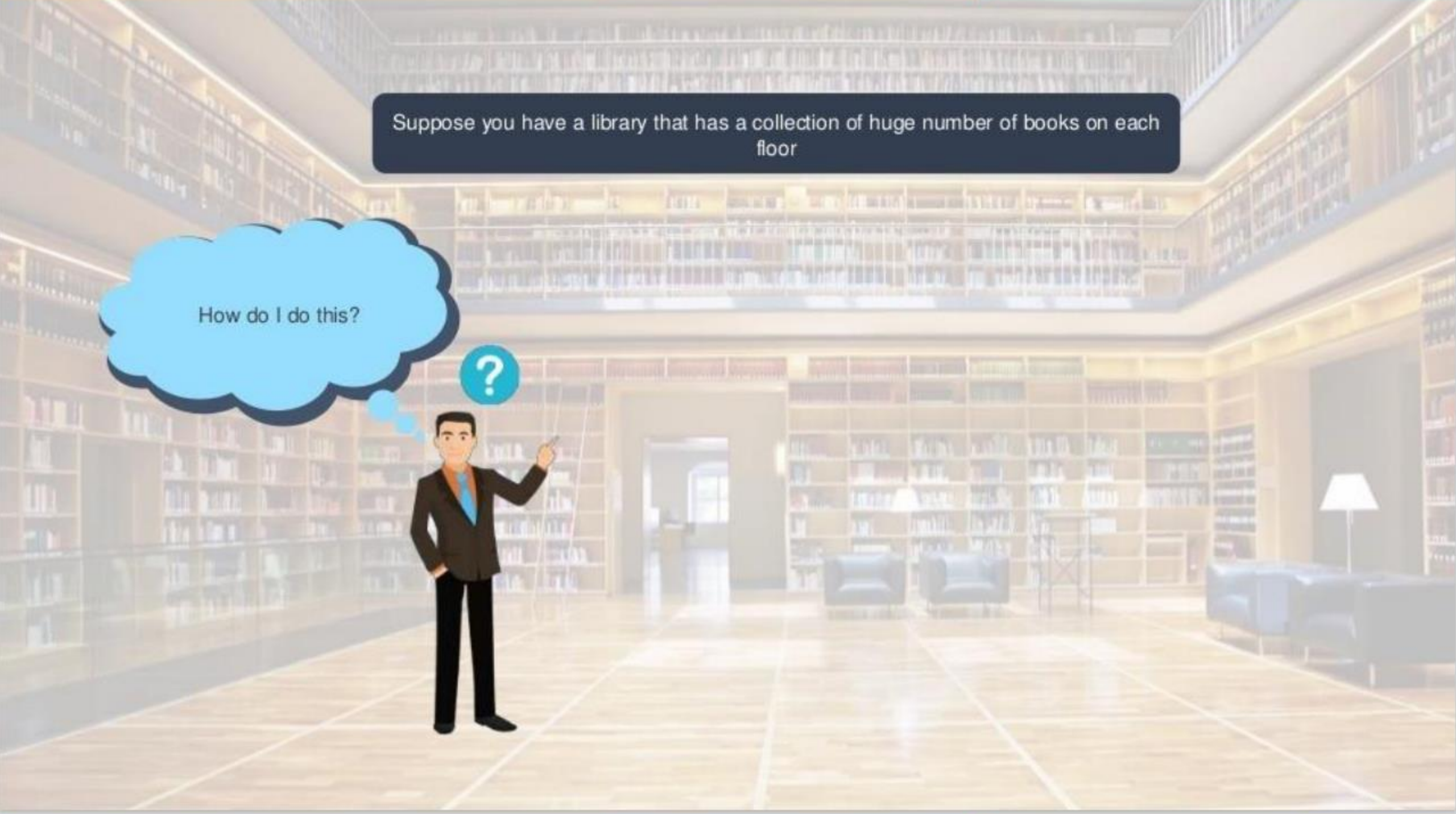


Open: http://localhost:9870



MapReduce


Suppose you have a library that has a collection of huge number of books on each floor
I want to count the total number of books present on each floor


Suppose you have a library that has a collection of huge number of books on each floor
How do I do this?
?

Suppose you have a library that has a collection of huge number of books on each floor

If I do it all myself, this would take a lot of time and effort

Not the most efficient way to count

Suppose you have a library that has a collection of huge number of books on each floor

Let me get 3 friends to do this task. Each friend will count the books for 1 particular floor

Faster and easy way to count

This type of process involves parallel processing by using multiple people (resources)

This type of process involves parallel processing by using multiple people (resources)

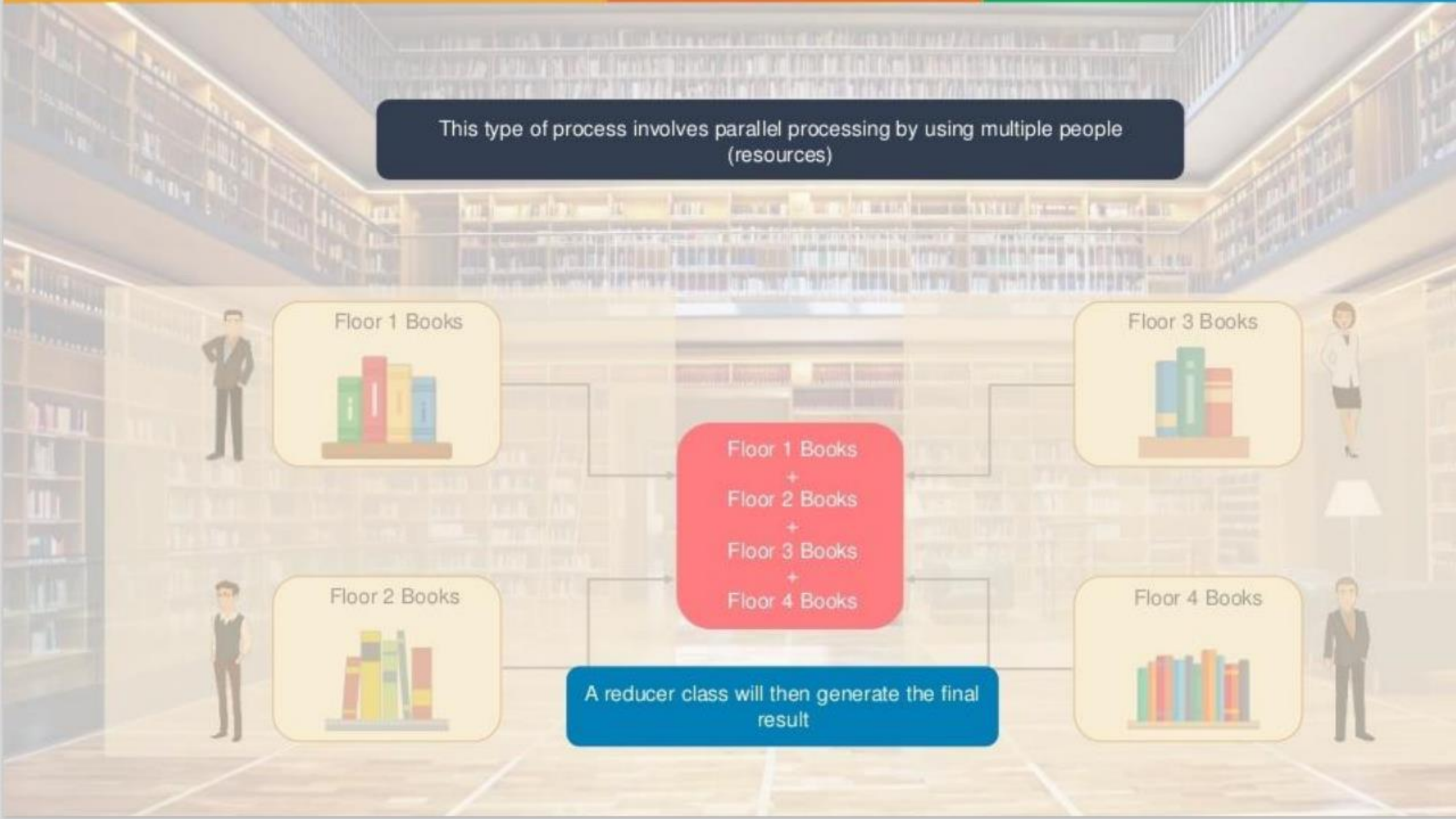
Floor 1 Books

Floor 3 Books

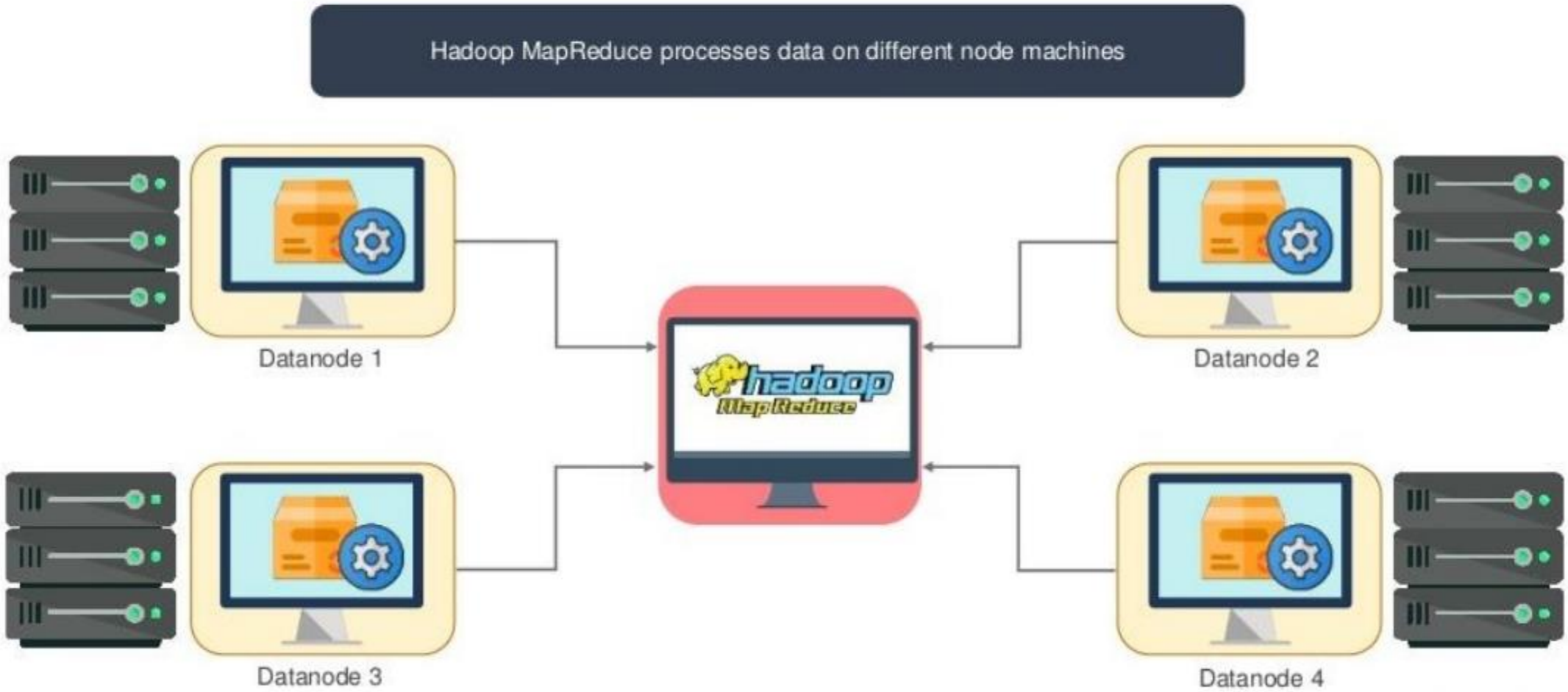
Each person will map the data of a particular floor

Floor 2 Books

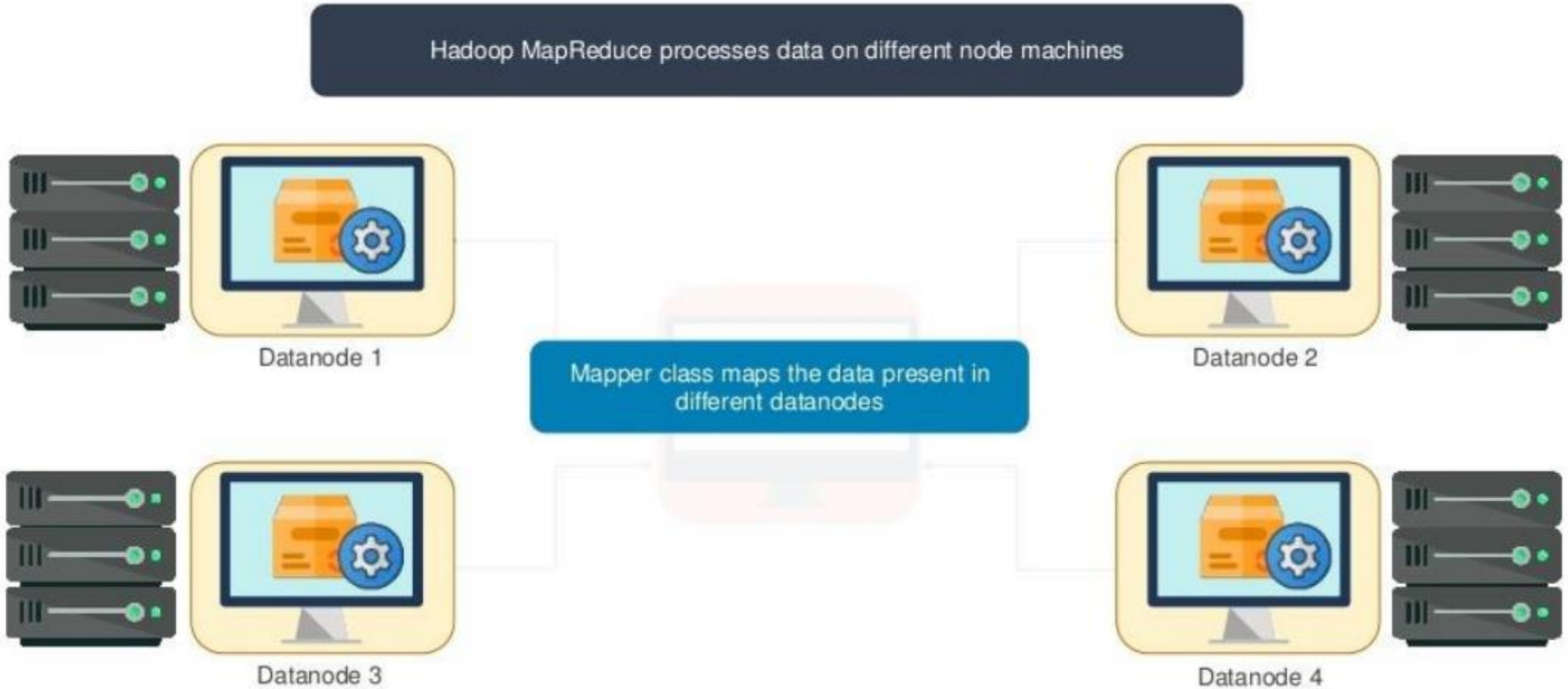
Floor 4 Books


This type of process involves parallel processing by using multiple people (resources)
Floor 1 Books
Floor 3 Books
Floor 1 Books
+
Floor 2 Books
+
Floor 3 Books
+
Floor 4 Books
Floor 2 Books
Floor 4 Books
A reducer class will then generate the final result

MapReduce Analogy



MapReduce Analogy

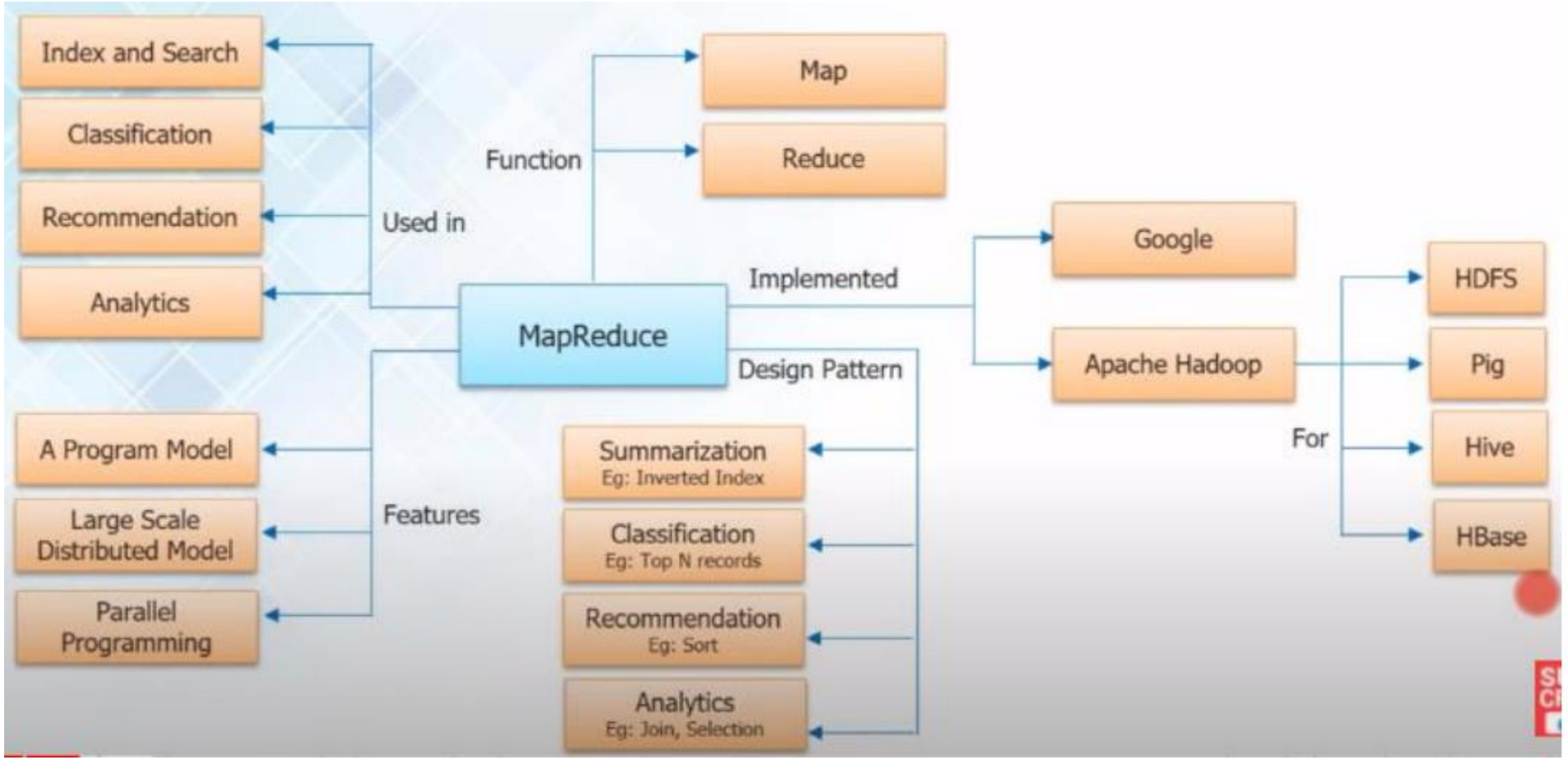


MapReduce Analogy

Hadoop MapReduce processes data on different node machines

A reducer class aggregates and reduces the output of different datanodes to generate the final output

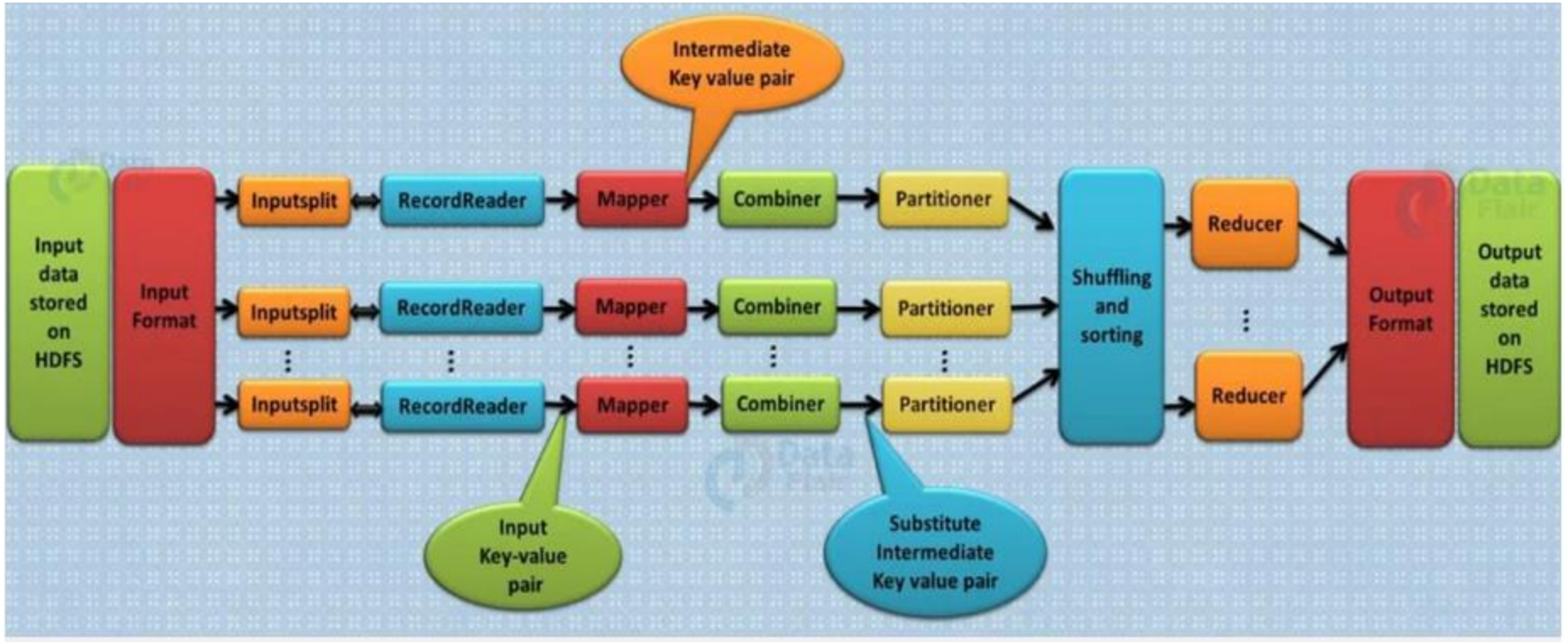
MapReduce in Nutshell



What is MapReduce?

MapReduce is the processing engine of Hadoop that processes and computes vast volumes of data

MapReduce Execution Pipeline



Input Files

The data for a MapReduce task is stored in **input files**, and input files typically lives in **HDFS**. The format of these files is arbitrary, while line-based log files and binary format can also be used.

InputFormat

Now, **InputFormat** defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.

InputSplits

It is created by InputFormat, logically represent the data which will be processed by an individual **Mapper**. One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper.

RecordReader

It communicates with the **InputSplit** in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

Mapper

It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process

Combiner

The combiner is also known as 'Mini-reducer'. Hadoop MapReduce Combiner performs local aggregation on the mappers' output, which helps to minimize the data transfer between mapper and **reducer** (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work.

Partitioner

Hadoop MapReduce, **Partitioner** comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).

Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition.

According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.

Shuffling and Sorting

Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.

Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.

RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

OutputFormat

The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus, the final output of reducer is written on HDFS by OutputFormat instances.

Types of InputFormat in MapReduce

- FileInputFormat
- TextInputFormat
- KeyValueTextInputFormat
- SequenceFileInputFormat
- SequenceFileAsTextInputFormat
- SequenceFileAsBinaryInputFormat
- NLineInputFormat
- DBInputFormat

FileInputFormat

- It is the base class for all file-based InputFormats.
- Hadoop FileInputFormat specifies input directory where data files are located.
- When we start a Hadoop job, FileInputFormat is provided with a path containing files to read.
- FileInputFormat will read all files and divides these files into one or more InputSplits.

TextInputFormat

- TextInputFormat is the default InputFormat.
- Each record is a line of input.
- The key, a LongWritable, is the byte offset of the beginning of the line within the file.
- The value is the contents of the line, excluding any line terminators.

example

input_data

A king should hunt regularly
A queen should shop daily,
Other people should just try.

The records are interpreted as the following key-value pairs.using TextInputFormat

Key	value
0	A king should hunt regularly
29	A queen should shop daily,
55	Other people should just try.

KeyValueTextInputFormat

- It is similar to TextInputFormat as it also treats each line of input as a separate record.
- While TextInputFormat treats entire line as the value, but the KeyValueTextInputFormat breaks the line itself into key and value by a tab character (‘/t’).
- Here Key is everything up to the tab character while the value is the remaining part of the line after tab character.

example

input-data ==> 2016,lakshmi,2000,10

2017,gowthami,3000,30

2018,suresh,4000,10

2019,danunjaya,6000,40

The records are interpreted as the following key-value pairs.using KeyvalueTextinputFormat

Key	value
2016	lakshmi,2000,10
2017	thalami,3000,30
2018	suresh,4000,10
2019	anunjaya,6000,40

SequenceFileInputFormat

- Hadoop **SequenceFileInputFormat** is an InputFormat which reads sequence files.
- Sequence files are binary files that stores sequences of binary **key-value pairs**.
- Sequence files block-compress and provide direct serialization and deserialization of several arbitrary data types (not just text).
- Here Key & Value both are user-defined.

SequenceFileAsTextInputFormat

- Hadoop **SequenceFileAsTextInputFormat** is another form of SequenceFileInputFormat which converts the sequence file key values to Text objects.
- By calling **'tostring()'** conversion is performed on the keys and values.
- This InputFormat makes sequence files suitable input for streaming.

SequenceFileAsBinaryInputFormat

- Hadoop **SequenceFileAsBinaryInputFormat** is a SequenceFileInputFormat using which we can extract the sequence file's keys and values as an opaque binary object.

NLineInputFormat

- Hadoop **NLineInputFormat** is another form of TextInputFormat where the keys are byte offset of the line and values are contents of the line.
- If we want our mapper to receive a fixed number of lines of input, then we use NLineInputFormat.
- N is the number of lines of input that each mapper receives.
- By default (N=1), each mapper receives exactly one line of input.
- If N=2, then each split contains two lines.
- One mapper will receive the first two Key-Value pairs and another mapper will receive the second two key-value pairs.

DBInputFormat

- Hadoop **DBInputFormat** is an InputFormat that reads data from a relational database, using JDBC.
- It is best for loading relatively small datasets.
- Here Key is LongWritables while Value is DBWritables.

OutputFormat in MapReduce

- TextOutputFormat
- SequenceFileOutputFormat
- SequenceFileAsBinaryOutputFormat
- MapFileOutputFormat
- MultipleOutputs
- LazyOutputFormat
- DBOutputFormat

TextOutputFormat

- The default OutputFormat is TextOutputFormat.
- It writes (key, value) pairs on individual lines of text files.
- Its keys and values can be of any type.
- The reason behind is that TextOutputFormat turns them to string by calling **toString()** on them.
- It separates key-value pair by a tab character.

SequenceFileOutputFormat

- This OutputFormat writes sequences files for its output.
- SequenceFileOutputFormat is also intermediate format used between MapReduce jobs.
- It serializes arbitrary data types to the file.
- It presents the data to the next **mapper** in the same manner as it was emitted by the previous reducer.

SequenceFileAsBinaryOutputFormat

- It is another variant of SequenceFileInputFormat.
- It also writes keys and values to sequence file in binary format.

MapFileOutputFormat

- It is another form of FileOutputFormat.
- It also writes output as map files.
- The framework adds a key in a MapFile in order.
- So, we need to ensure that reducer emits keys in sorted order.

MultipleOutputs

- This format allows writing data to files whose names are derived from the output keys and values.

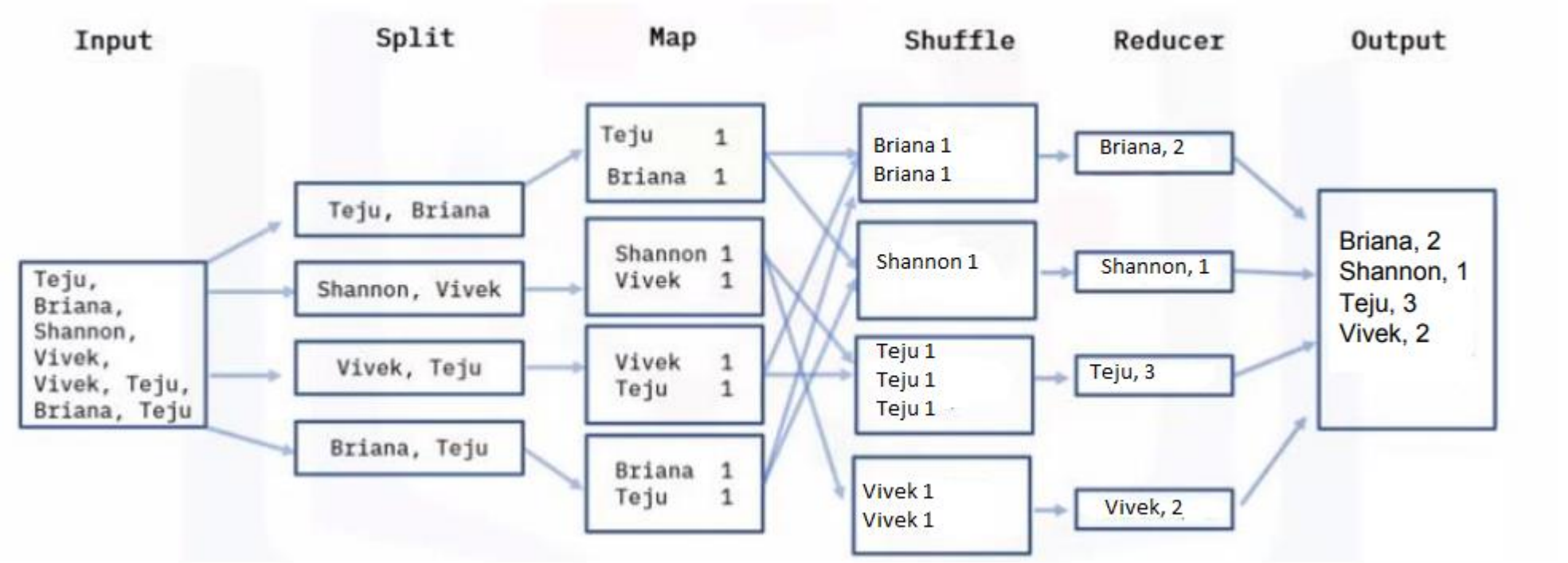
LazyOutputFormat

- In MapReduce job execution, FileOutputFormat sometimes create output files, even if they are empty.
- LazyOutputFormat is also a wrapper OutputFormat.

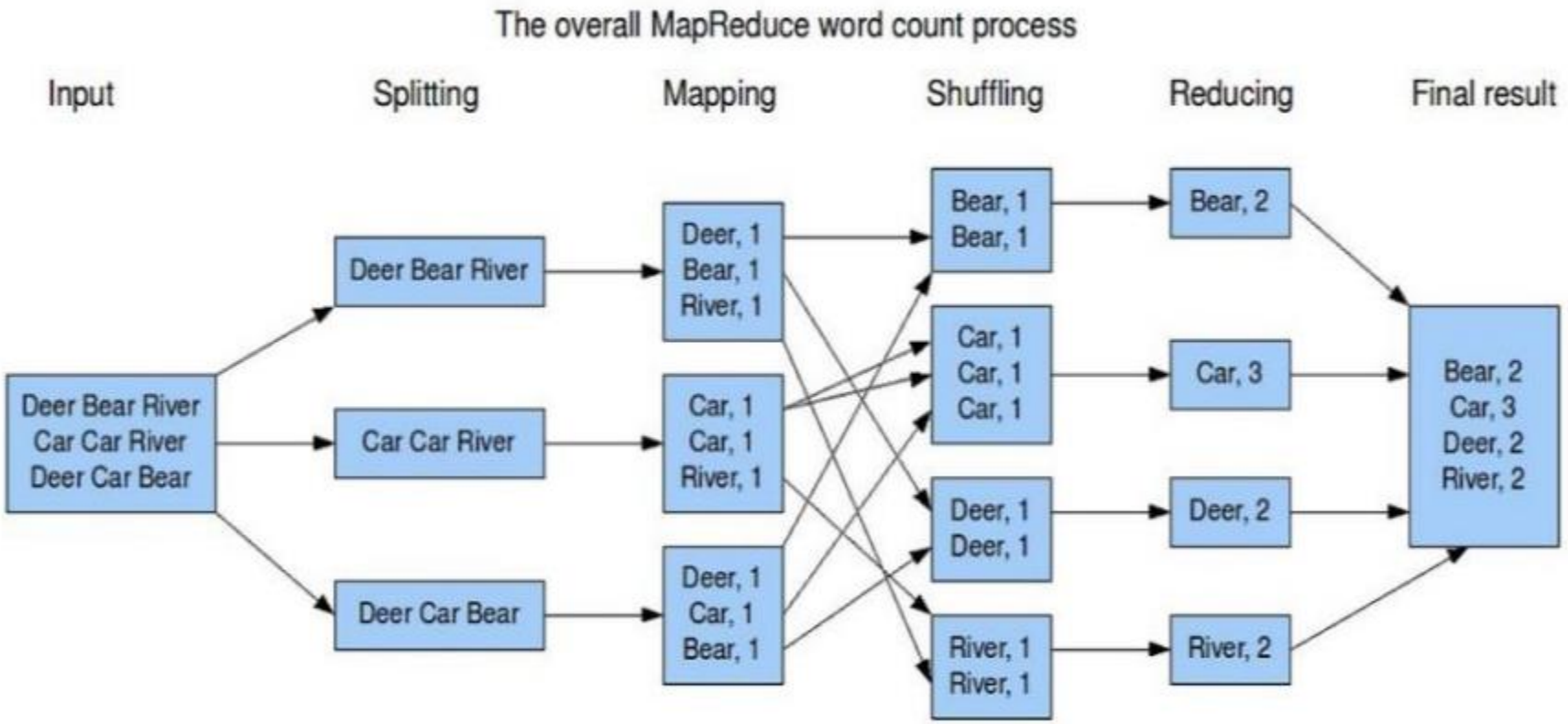
DBOutputFormat

- It is the OutputFormat for writing to relational databases and HBase.
- This format also sends the reduce output to a SQL table.
- It also accepts key-value pairs.
- In this, the key has a type extending DBwritable.

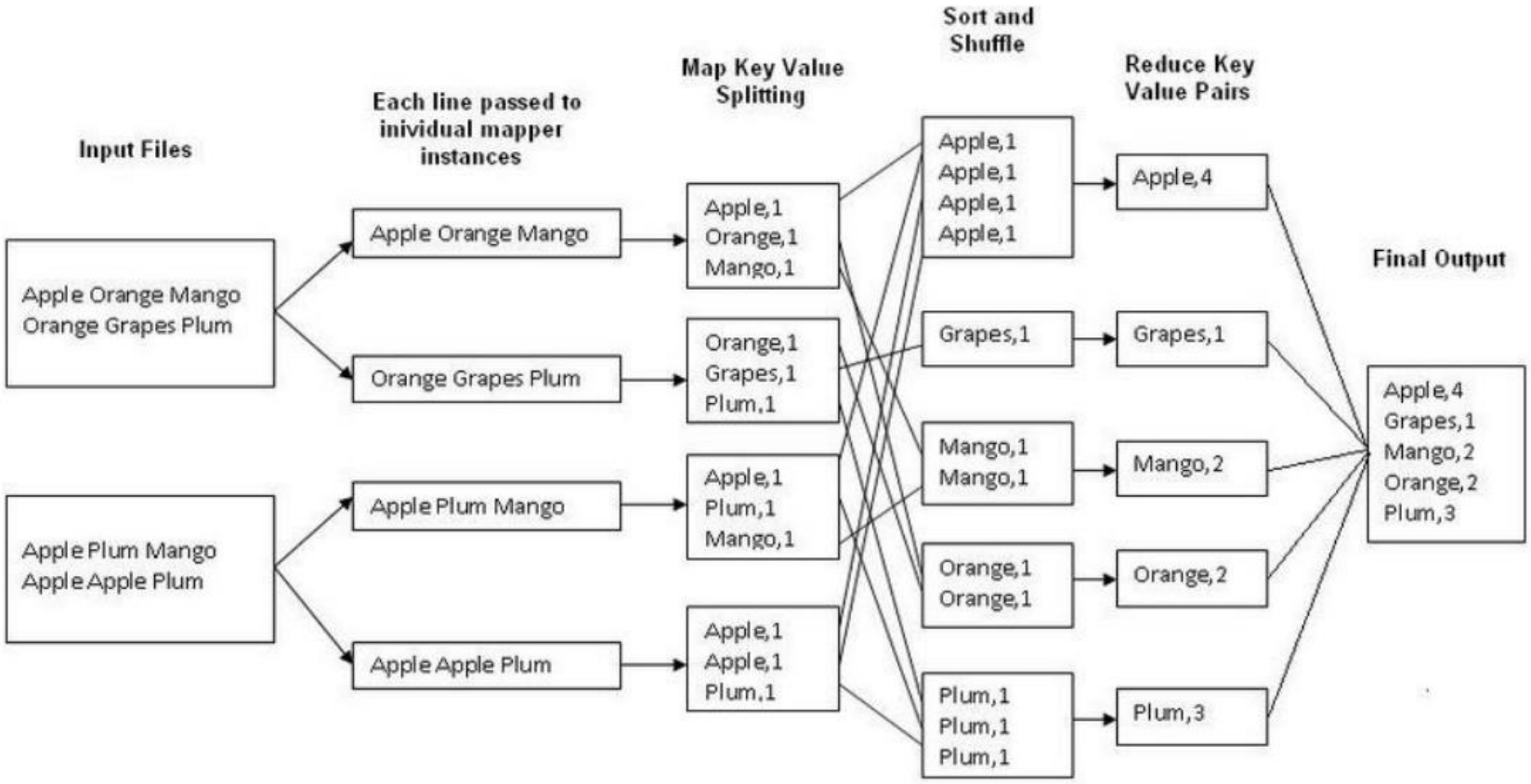
MapReduce WordCount Example 1



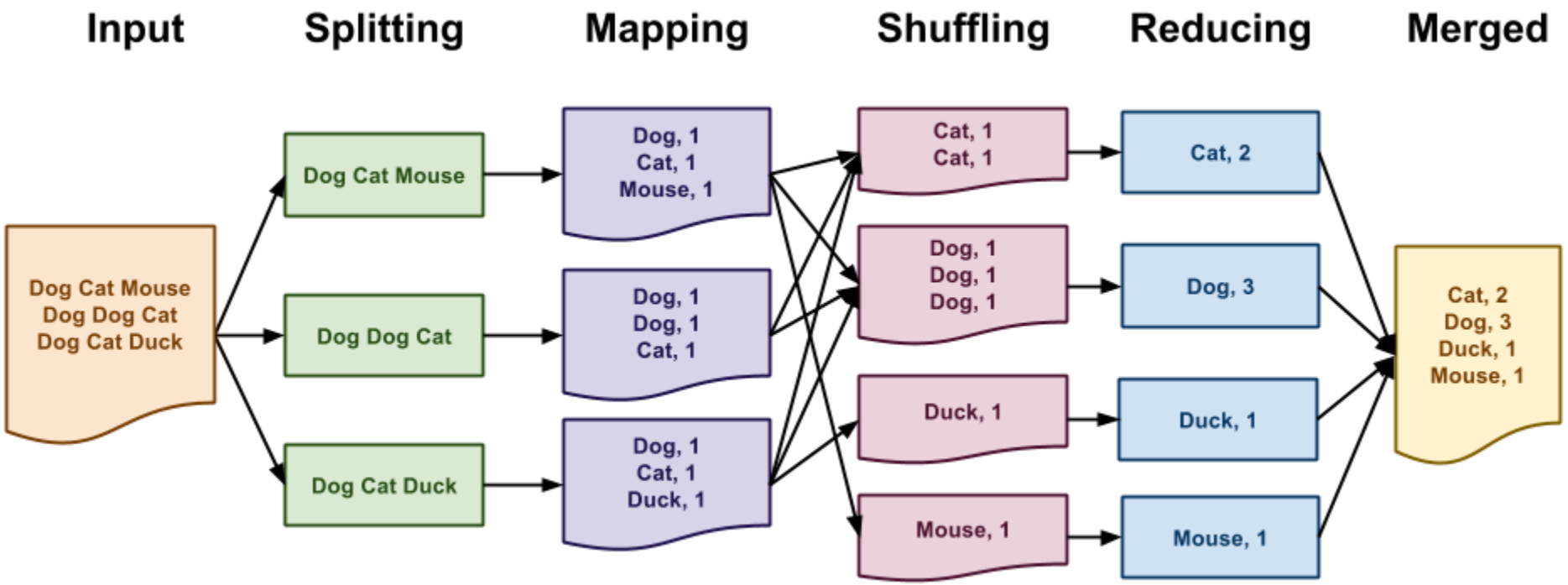
MapReduce WordCount Example 2



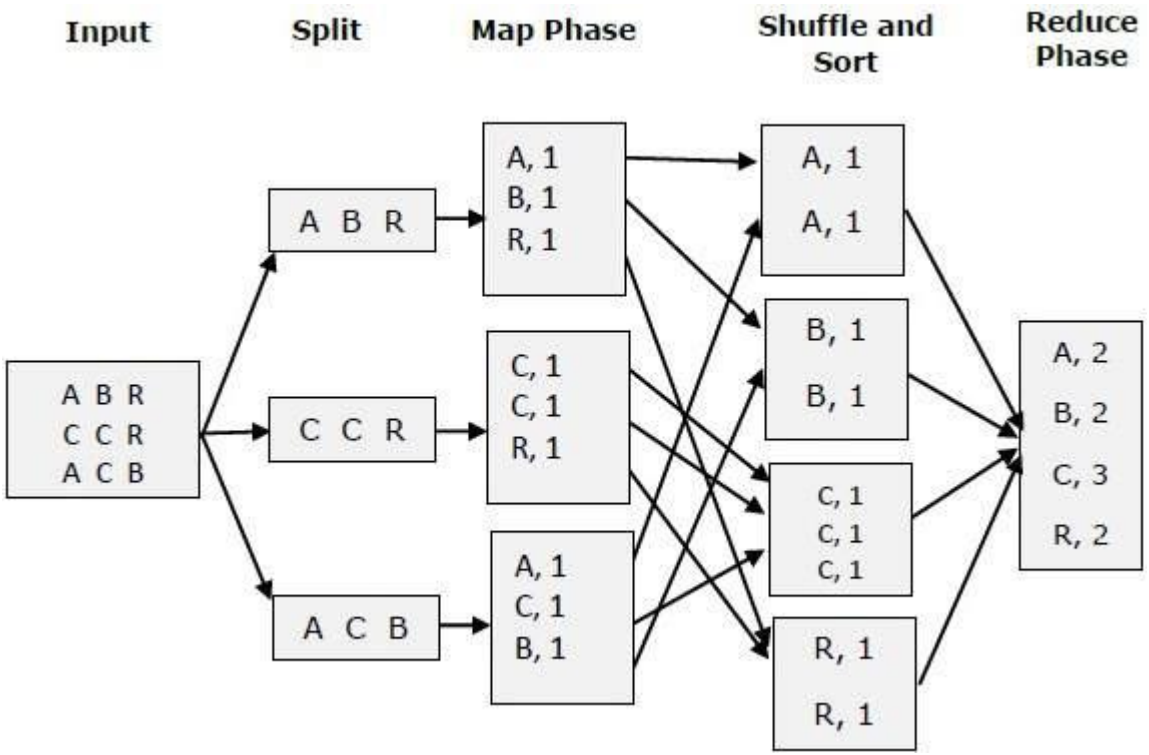
MapReduce WordCount Example 3



MapReduce Wordcount Example 4



MapReduce Wordcount Example 5



MapReduce Matrix Multiplication

Algorithm 1: The Map Function

1 **for** *each element* m_{ij} *of* M **do**
2 produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up to the number of columns of N
3 **for** *each element* n_{jk} *of* N **do**
4 produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up to the number of rows of M
5 **return** *Set of (key, value) pairs that each key,* (i,k)*, has a list with values* (M,j,m_{ij}) *and* (N,j,n_{jk}) *for all possible values of* j

Algorithm 2: The Reduce Function

1 **for** *each key (i,k)* **do**
2 sort values begin with M by j in $list_M$
3 sort values begin with N by j in $list_N$
4 multiply m_{ij} and n_{jk} for j_{th} value of each list
5 sum up $m_{ij} * n_{jk}$
6 **return** $(i,k), \sum_{j=1} m_{ij} * n_{jk}$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

A → M×N i*j o/p C → M×P i*k

B → N×P j*k

Map() → (key, value)

A ⇒ (i,k) key (A, j, a_{ij}) value

B ⇒ (i,k) key (B, j, b_{jk}) value

Input File

$a_{11}=1$ (ij)	(1,1) (A,1,1)
	(1,2) (A,1,1)
$a_{12}=2$ (ij)	(1,1) (A,2,2)
	(1,2) (A,2,2)
$a_{21}=3$ (ij)	(2,1) (A,1,3)
	(2,2) (A,1,3)
$a_{22}=4$ (ij)	(2,1) (A,2,4)
	(2,2) (A,2,4)

k value not known, so duplicate each entry.

$b_{11}=5$ (jk)	(1,1) (B,1,5)
	(2,1) (B,1,5)
$b_{12}=6$ (jk)	(1,2) (B,1,6)
	(2,2) (B,1,6)
$b_{21}=7$ (jk)	(1,1) (B,2,7)
	(2,1) (B,2,7)
$b_{22}=8$ (jk)	(1,2) (B,2,8)
	(2,2) (B,2,8)

i value not known, so duplicate each entry.

Mapper

(1,1)	(A,1,1) (A,2,2) (B,1,5) (B,2,7)
(1,2)	(A,1,1) (A,2,2) (B,1,6) (B,2,8)
(2,1)	(A,1,3) (A,2,4) (B,1,5) (B,2,7)
(2,2)	(A,1,3) (A,2,4) (B,1,6) (B,2,8)

Reducer

(1,1) (A,1,1) (A,2,2)
(B,1,5) (B,2,7)
5 + 14
= 19

(1,2) (A,1,1) (A 2,2)
(B,1,6) (B,2,8)
6 + 16
= 22

(2,1) (A,1,3) (A,2,4)
(B,1,5) (B,2,7)
15 + 28
= 43

(2,2) (A,1,3) (A,2,4)
(B,1,6) (B,2,8)
18 + 32
= 50

$$\text{O/P} = C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix}$

A → M×N (3*2, i*j) O/P :- C → M×P (3*2, i*k)

B → N×P (j*k, 2*2)

Map() → (key, value)

A ⟹ (i,k) (A, j, a_{ij}) key value

B ⟹ (i,k) (B, j, b_{jk}) key value

Input File

$a_{11}=1$	(1,1)	(A,1,1)
ij	(1,2)	(A,1,1)
$a_{12}=2$	(1,1)	(A,2,2)
ij	(1,2)	(A,2,2)
$a_{21}=2$	(2,1)	(A,1,2)
ij	(2,2)	(A,1,2)
$a_{22}=1$	(2,1)	(A,2,1)
ij	(2,2)	(A,2,1)
$a_{31}=3$	(3,1)	(A,1,3)
ij	(3,2)	(A,1,3)
$a_{32}=4$	(3,1)	(A,2,4)
ij	(3,2)	(A,2,4)

$b_{11}=1$	(1,1)	(B,1,1)
jk	(2,1)	(B,1,1)
	(3,1)	(B,1,1)
$b_{12}=2$	(1,2)	(B,1,2)
jk	(2,2)	(B,1,2)
	(3,2)	(B,1,2)
$b_{21}=1$	(1,1)	(B,2,1)
jk	(2,1)	(B,2,1)
	(3,1)	(B,2,1)
$b_{22}=3$	(1,2)	(B,2,3)
jk	(2,2)	(B,2,3)
	(3,2)	(B,2,3)

Mapper

(1,1)	(A,1,1)	(A,2,2)	(B,1,1)	(B,2,1)
(1,2)	(A,1,1)	(A,2,2)	(B,1,2)	(B,2,3)
(2,1)	(A,1,2)	(A,2,1)	(B,1,1)	(B,2,1)
(2,2)	(A,1,2)	(A,2,1)	(B,1,2)	(B,2,3)
(3,1)	(A,1,3)	(A,2,4)	(B,1,1)	(B,2,1)
(3,2)	(A,1,3)	(A,2,4)	(B,1,2)	(B,2,3)

Reducer

(1,1)	(A,1,1)	(A,2,2)	O/P:- 1×1 + 2×1 = 3
	(B,1,1)	(B,2,1)	
(1,2)	(A,1,1)	(A,2,2)	O/P:- 1×2 + 2×3 = 8
	(B,1,2)	(B,2,3)	
(2,1)	(A,1,2)	(A,2,1)	O/P:- 2×1 + 1×1 = 3
	(B,1,1)	(B,2,1)	
(2,2)	(A,1,2)	(A,2,1)	O/P:- 2×2 + 1×3 = 7
	(B,1,2)	(B,2,3)	
(3,1)	(A,1,3)	(A,2,4)	O/P:- 3×1 + 4×1 = 7
	(B,1,1)	(B,2,1)	
(3,2)	(A,1,3)	(A,2,4)	O/P:- 3×2 + 4×3 = 18
	(B,1,2)	(B,2,3)	

Output :- $C = \begin{bmatrix} 3 & 8 \\ 3 & 7 \\ 7 & 18 \end{bmatrix}$


The End