HDFS ARCHITECTURE

HADOOP

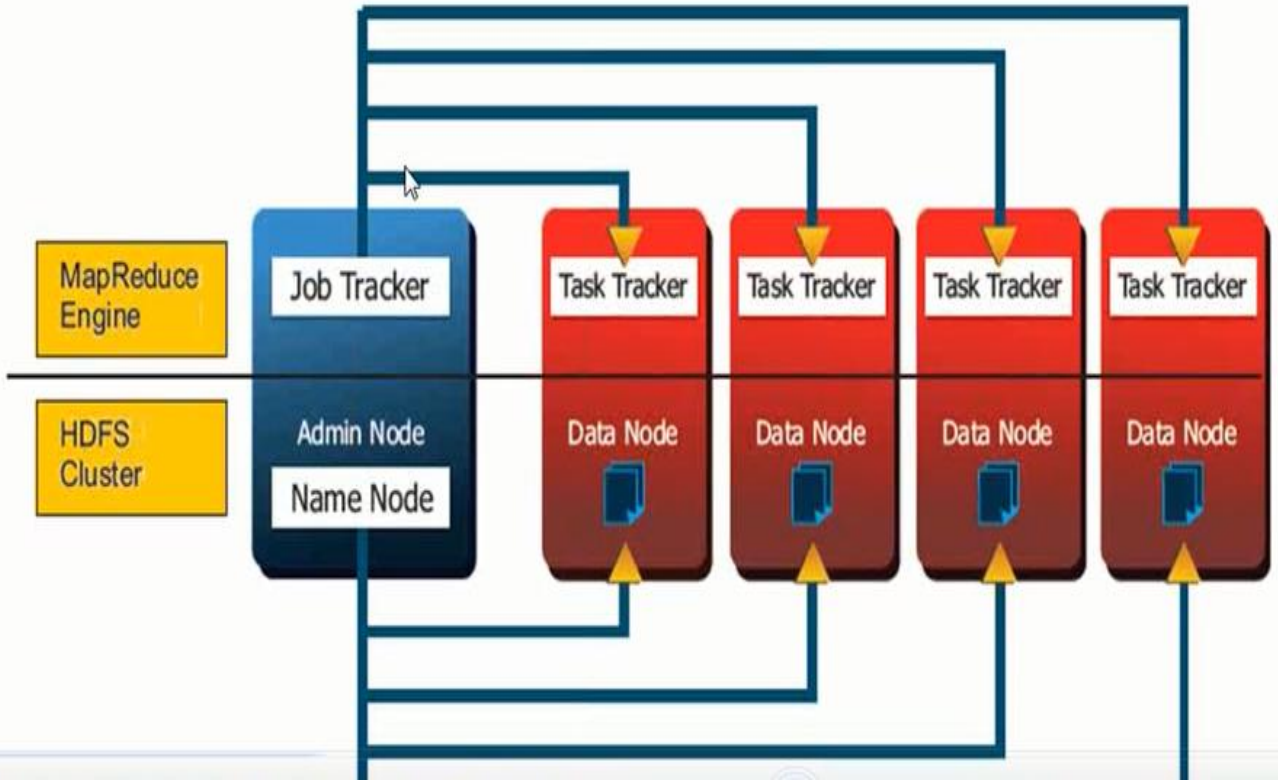
- ✓ Apache Hadoop is a framework that allows for the distributed processing of large data sets across clusters of commodity computers using a simple programming model.

- ✓ Companies using Hadoop:
  - Yahoo
  - Google
  - Facebook
  - Amazon
  - AOL
  - IBM
  - And many more at

http://wiki.apache.org/hadoop/PoweredBy

HDFS CORE COMPONENTS

- ✓ HDFS – Hadoop Distributed File System (storage)
- ✓ MapReduce (processing)



HDFS

HDFS - Hadoop Distributed File System

- ✓ Highly fault-tolerant
- ✓ High throughput
- ✓ Suitable for applications with large data sets
- ✓ Streaming access to file system data
- ✓ Can be built out of commodity hardware

DESIGN OF HDFS

HDFS is a file system designed for storing very large files with streaming data access patterns, running clusters on commodity hardware.

AREAS WHERE HDFS IS NOT GOOD

- Low-latency data access
- Lots of small files
- Multiple writers, arbitrary file modifications

HDFS COMPONENTS

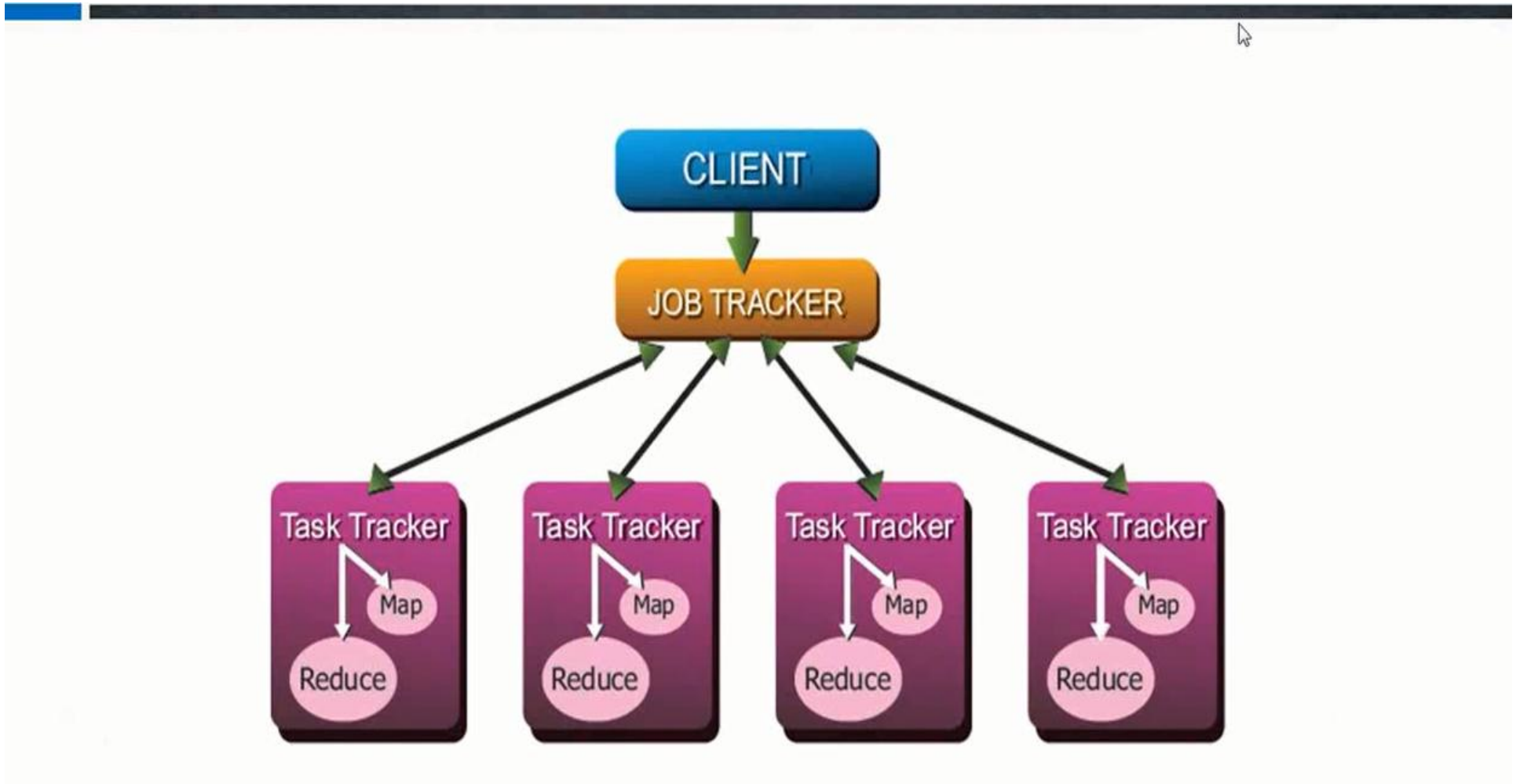
- Namenodes
- Datanodes

MAIN COMPONENTS OF HDFS

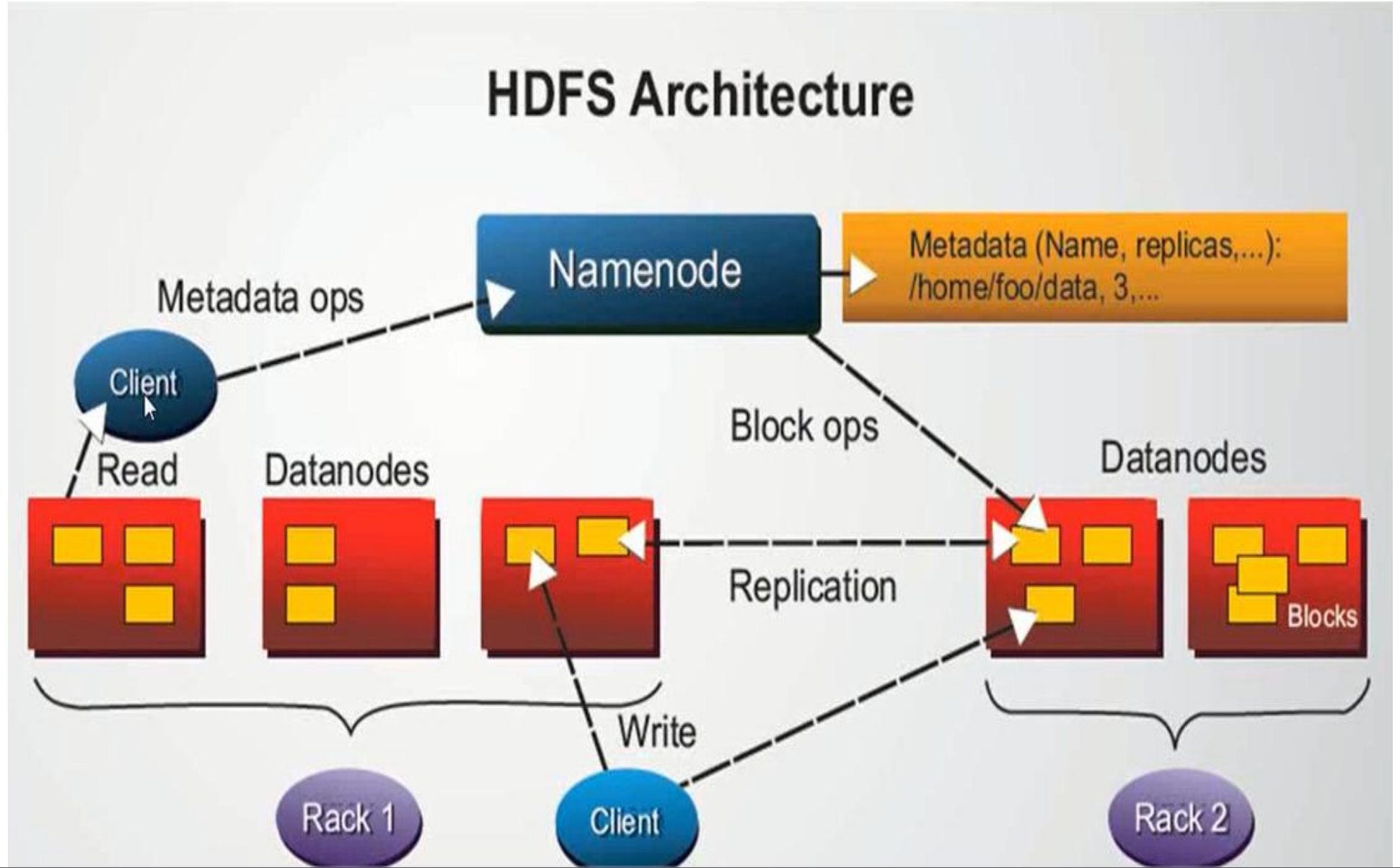
- ✓ NameNode:
  - ✓ master of the system
  - ✓ maintains and manages the blocks which are present on the DataNodes

- ✓ DataNodes:
  - ✓ slaves which are deployed on each machine and provide the actual storage
  - ✓ responsible for serving read and write requests for the clients

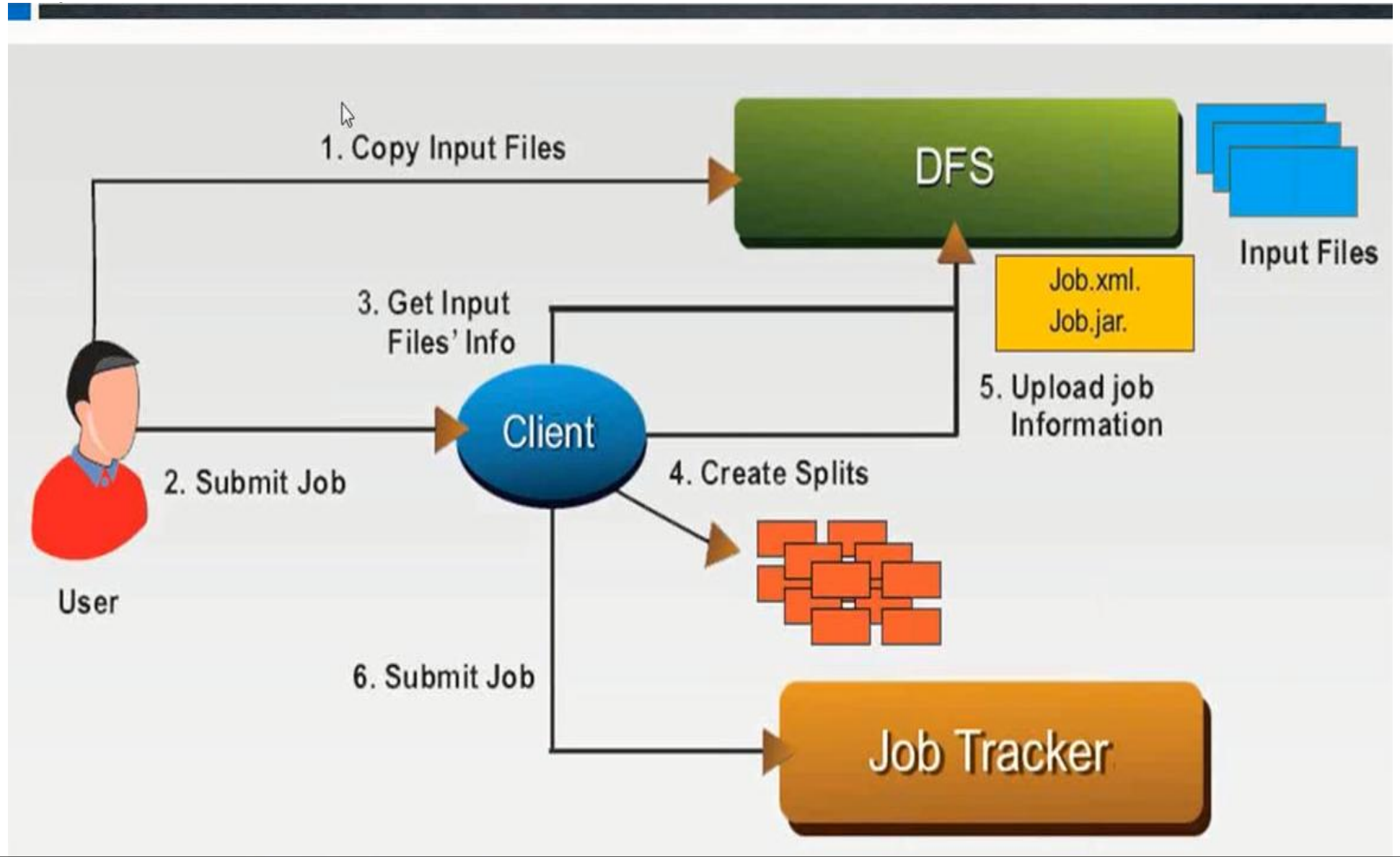
JOB TRACKER & TASK TRACKER



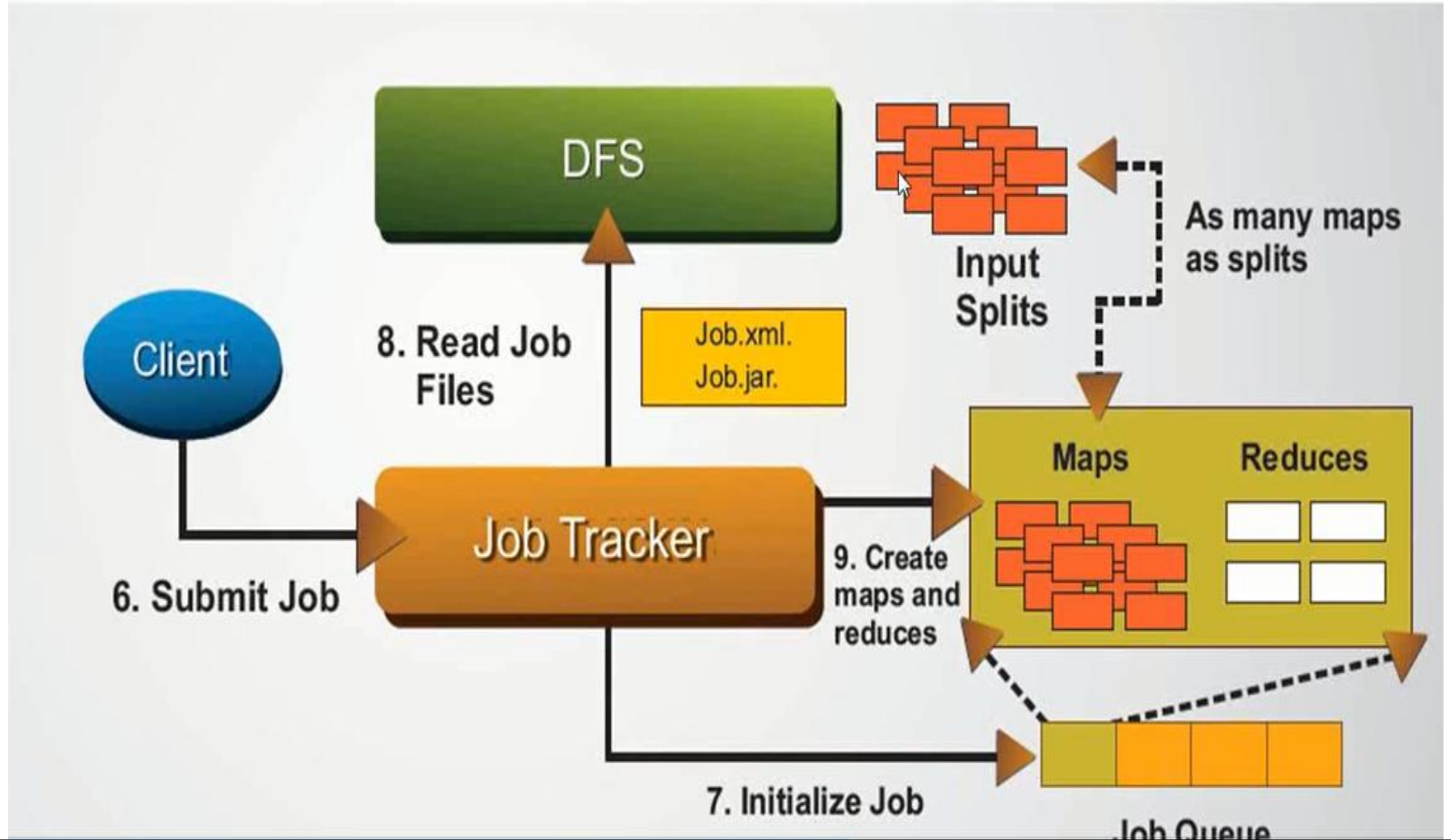
HDFS ARCHITECTURE



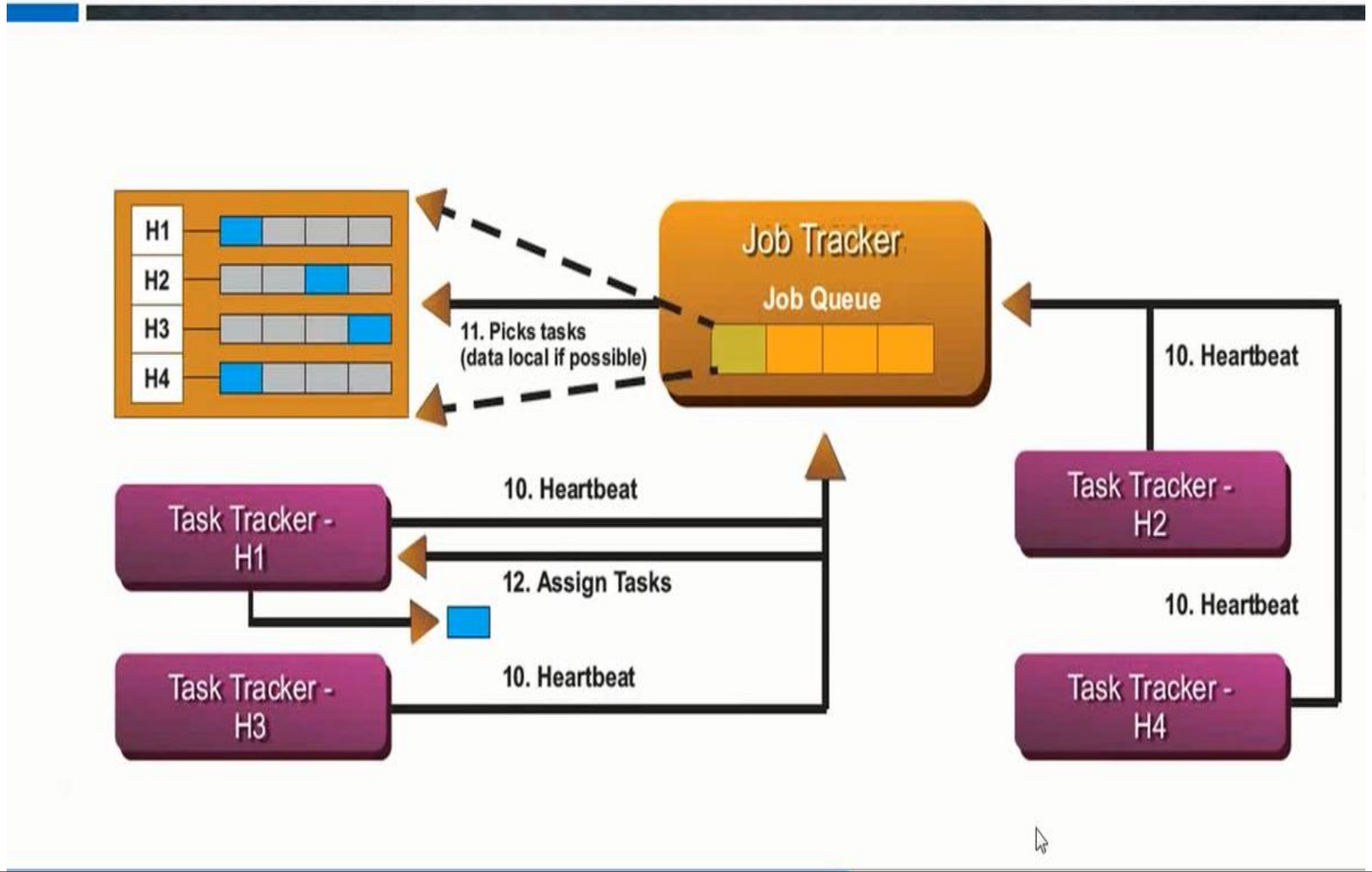
JOB TRACKER


1. Copy Input Files
DFS
Input Files
Job.xml.
Job.jar.
3. Get Input
Files' Info
5. Upload job
Information
Client
2. Submit Job
4. Create Splits
User
6. Submit Job
Job Tracker

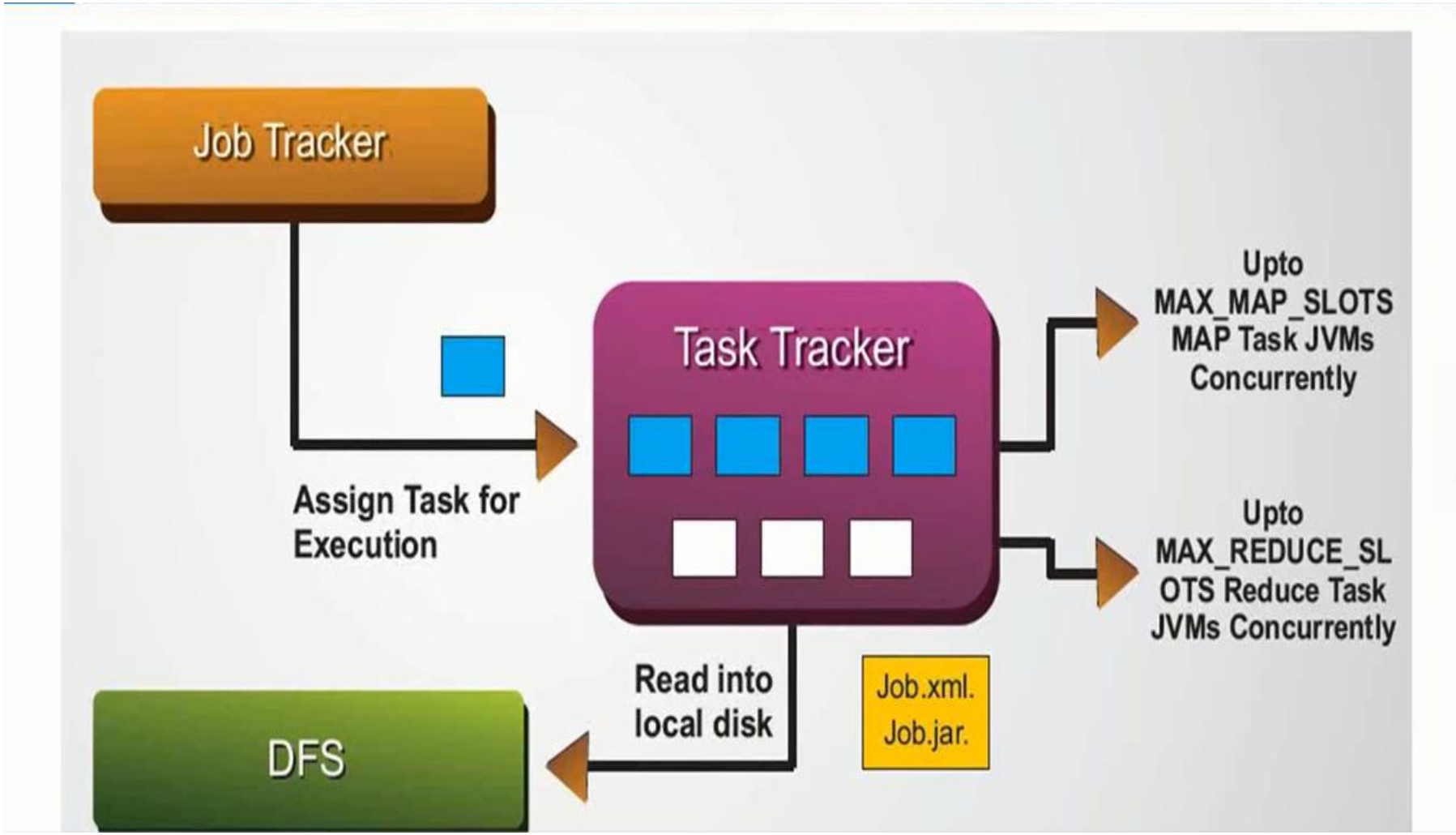
JOB TRACKER CONTD…….



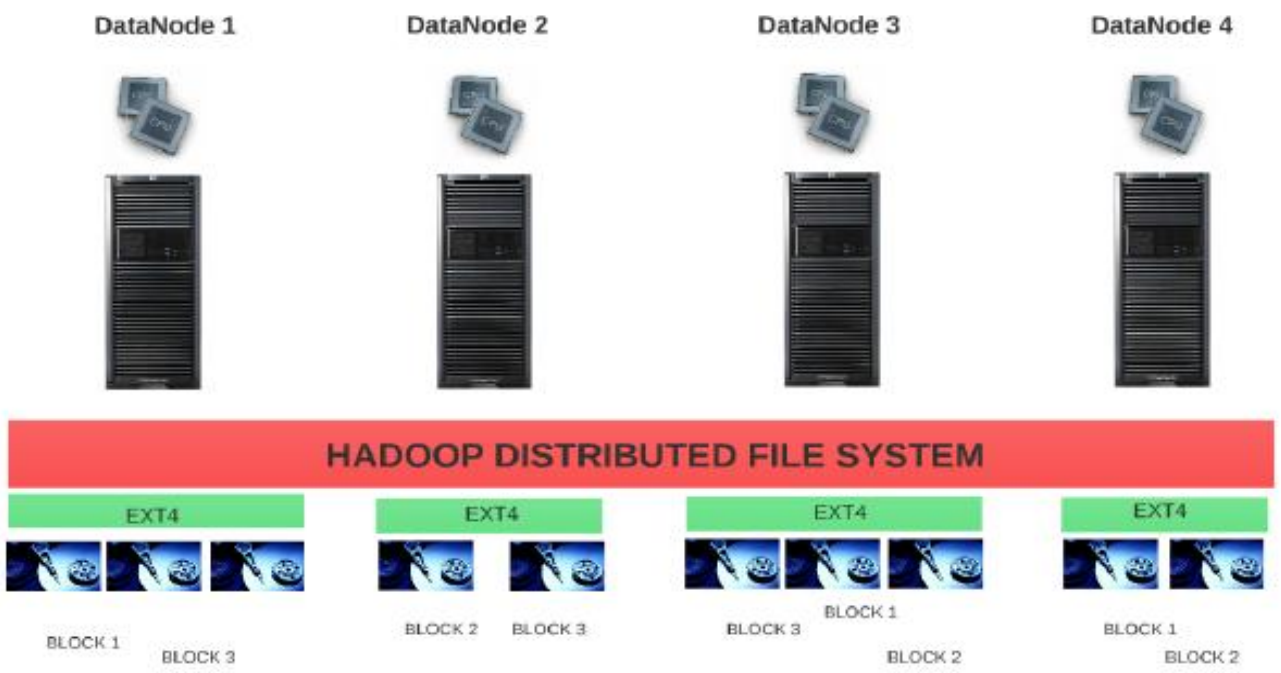
JOB TRACKER CONTD….



JOB TRACKER CONTD…….



DATANODE


DataNode 1
DataNode 2
DataNode 3
DataNode 4
HADOOP DISTRIBUTED FILE SYSTEM
EXT4
EXT4
EXT4
EXT4
BLOCK 1
BLOCK 3
BLOCK 2
BLOCK 3
BLOCK 1
BLOCK 3
BLOCK 2
BLOCK 1
BLOCK 2

NAMENODE

aka Master

File Name - MyDatasetInHDFS
File Size - 350 MB
Replication Factor - 3
Blocks - BLK_0045732, BLK_9610590, BLK_8851209
Block Locations
Permissions
Created By, Created On
Lasted Modified By, Last Modified On

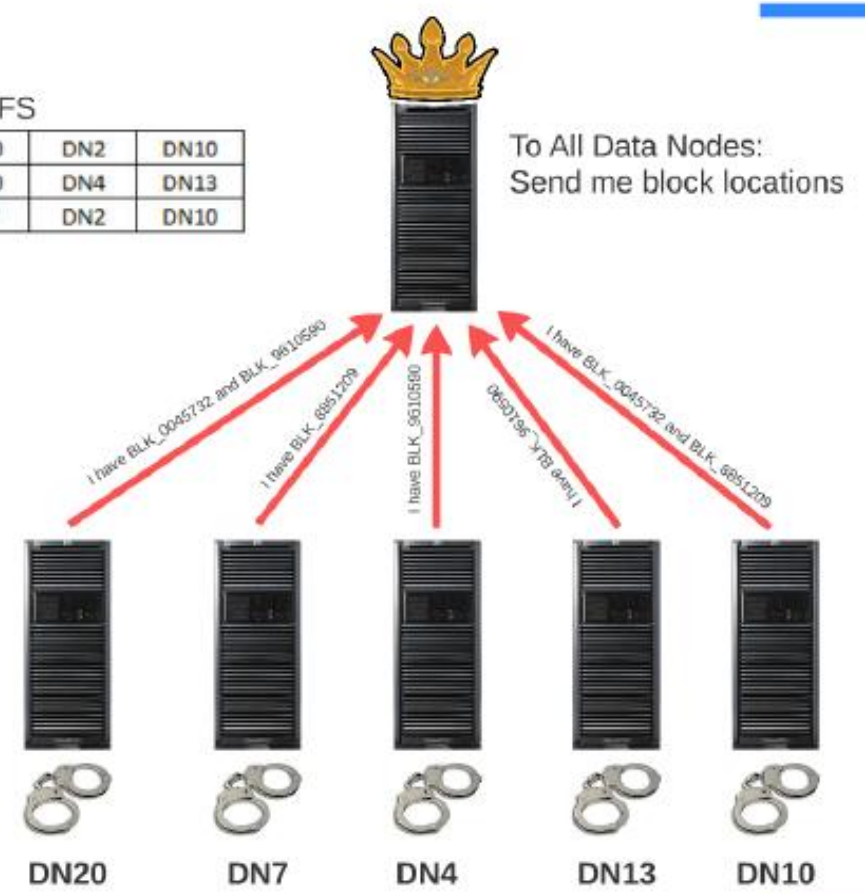
Name Node

In Disk - Metadata of Files & Folders

In Memory - Block locations

MyDatasetInHDFS

BLK_0045732	DN20	DN2	DN10
BLK_9610590	DN20	DN4	DN13
BLK_8851209	DN7	DN2	DN10



HADOOP CLUSTER

NODE	RACK	CLUSTER	FACEBOOK	YAHOO!	DATA CENTER

NODE

CPU + RAM + DISK

Name Node Configuration

Processors: 2 Quad Core CPUs running @ 2 GHz
RAM: 128 GB
Disk: 6 x 1TB SATA
Network: 10 Gigabit Ethernet

Node

Data Node Configuration

Processors: 2 Quad Core CPUs running @ 2 GHz
RAM: 64 GB
Disk: 12-24 x 1TB SATA
Network: 10 Gigabit Ethernet

RACK

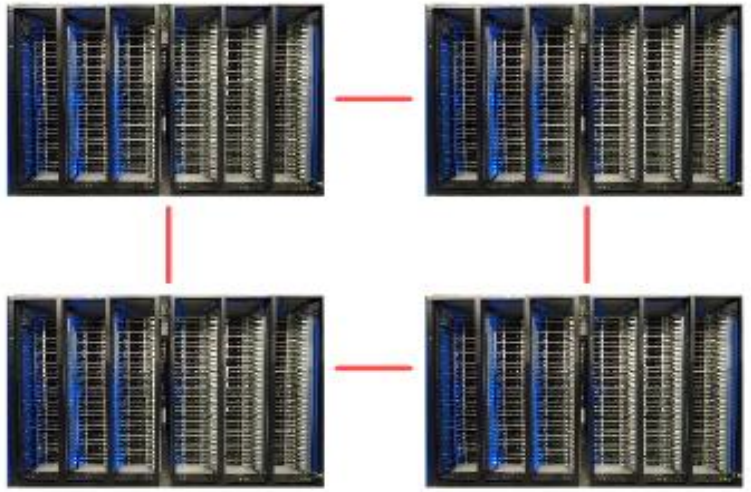
CLUSTER

Rack

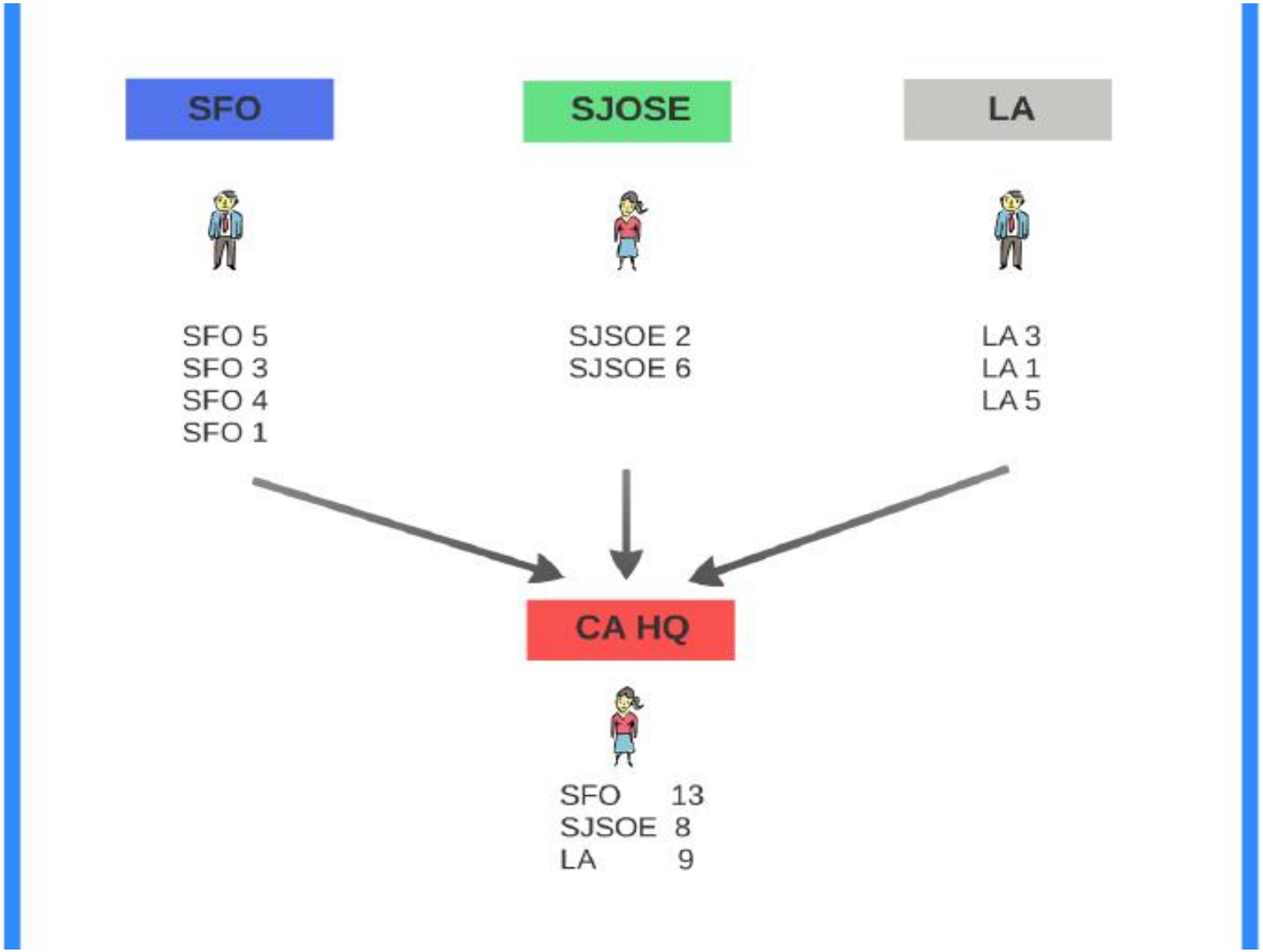
Collection of nodes

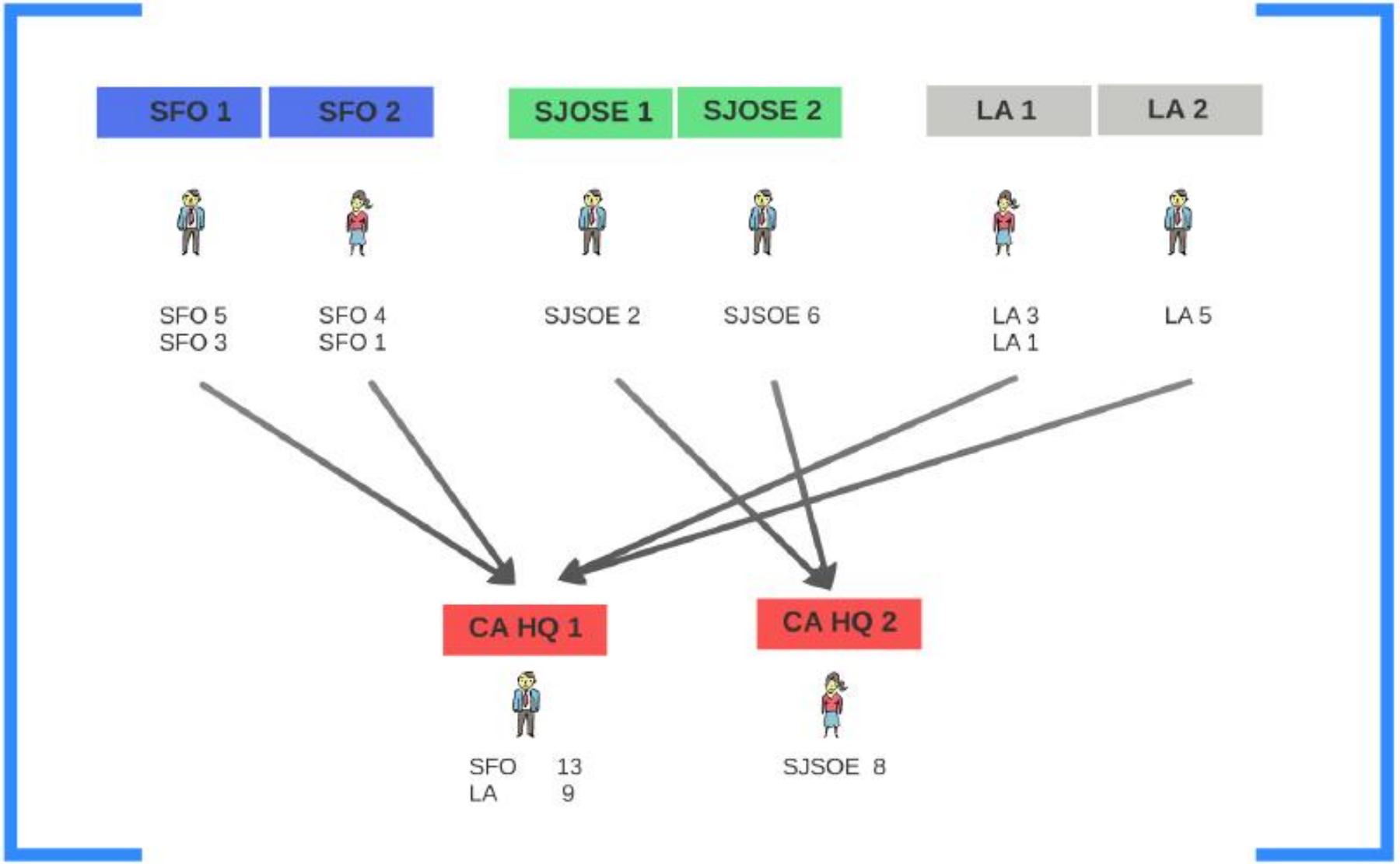
Cluster

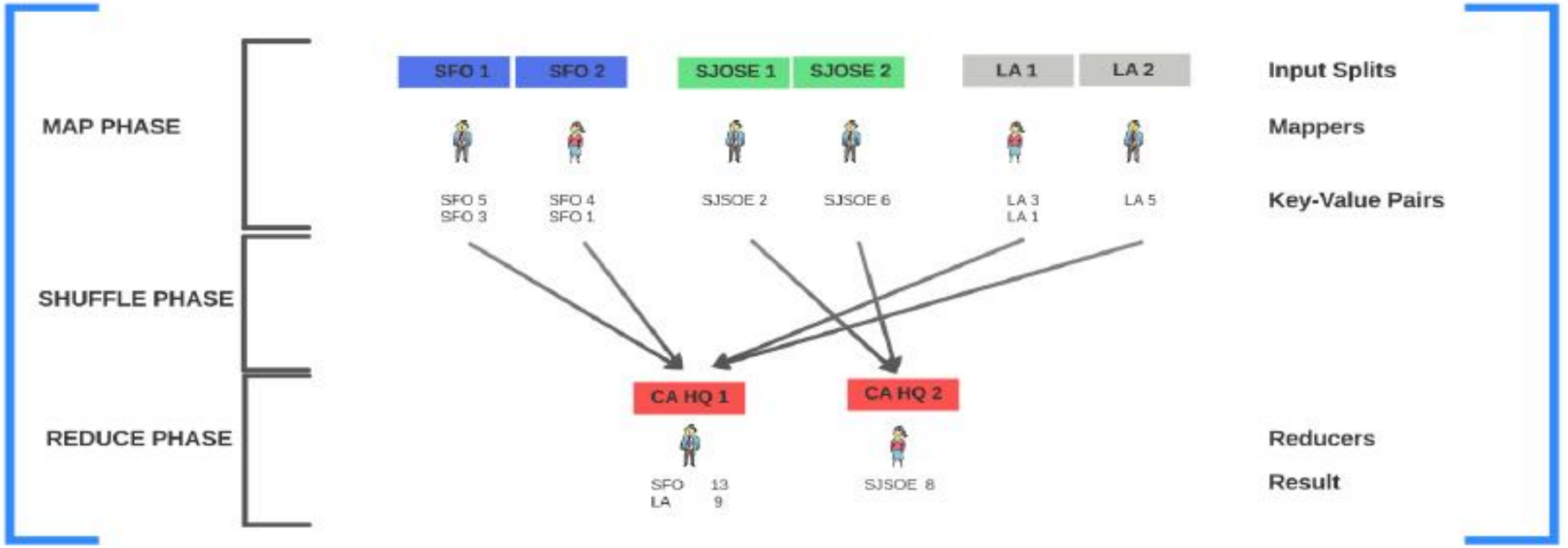
Racks interconnected over network



MAPREDUCE


SFO
SJOSE
LA
SFO 5
SFO 3
SFO 4
SFO 1
SJSOE 2
SJSOE 6
LA 3
LA 1
LA 5
CA HQ
SFO 13
SJSOE 8
LA 9


SFO 1
SFO 2
SJOSE 1
SJOSE 2
LA 1
LA 2
SFO 5
SFO 3
SFO 4
SFO 1
SJSOE 2
SJSOE 6
LA 3
LA 1
LA 5
CA HQ 1
CA HQ 2
SFO 13
LA 9
SJSOE 8


MAP PHASE
SHUFFLE PHASE
REDUCE PHASE
SFO 1
SFO 2
SJOSE 1
SJOSE 2
LA 1
LA 2
Input Splits
Mappers
Key-Value Pairs
SFO 5
SFO 3
SFO 4
SFO 1
SJSOE 2
SJSOE 6
LA 3
LA 1
LA 5
CA HQ 1
CA HQ 2
Reducers
Result
SFO 13
LA 9
SJSOE 8

WHAT IS MAPREDUCE?

- Distributed Programming model for processing large data sets
- Conceived at Google
- Can be implemented in any programming language
- MapReduce is NOT a programming language
- Hadoop implements MapReduce
- MapReduce System (Hadoop) - Manage communications, data transfers, parallel execution across distributed servers

DISSECTING MAPREDUCE COMPONENTS

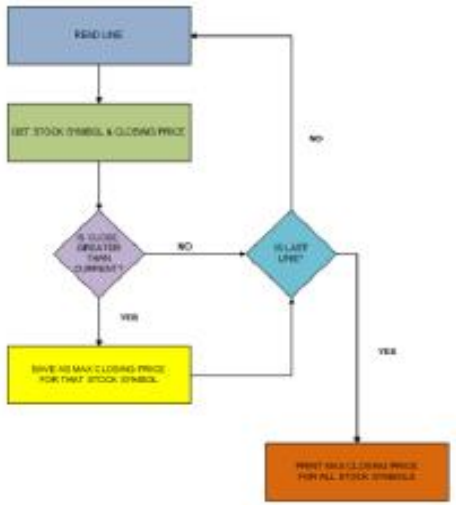
SAMPLE BIG DATA PROBLEM

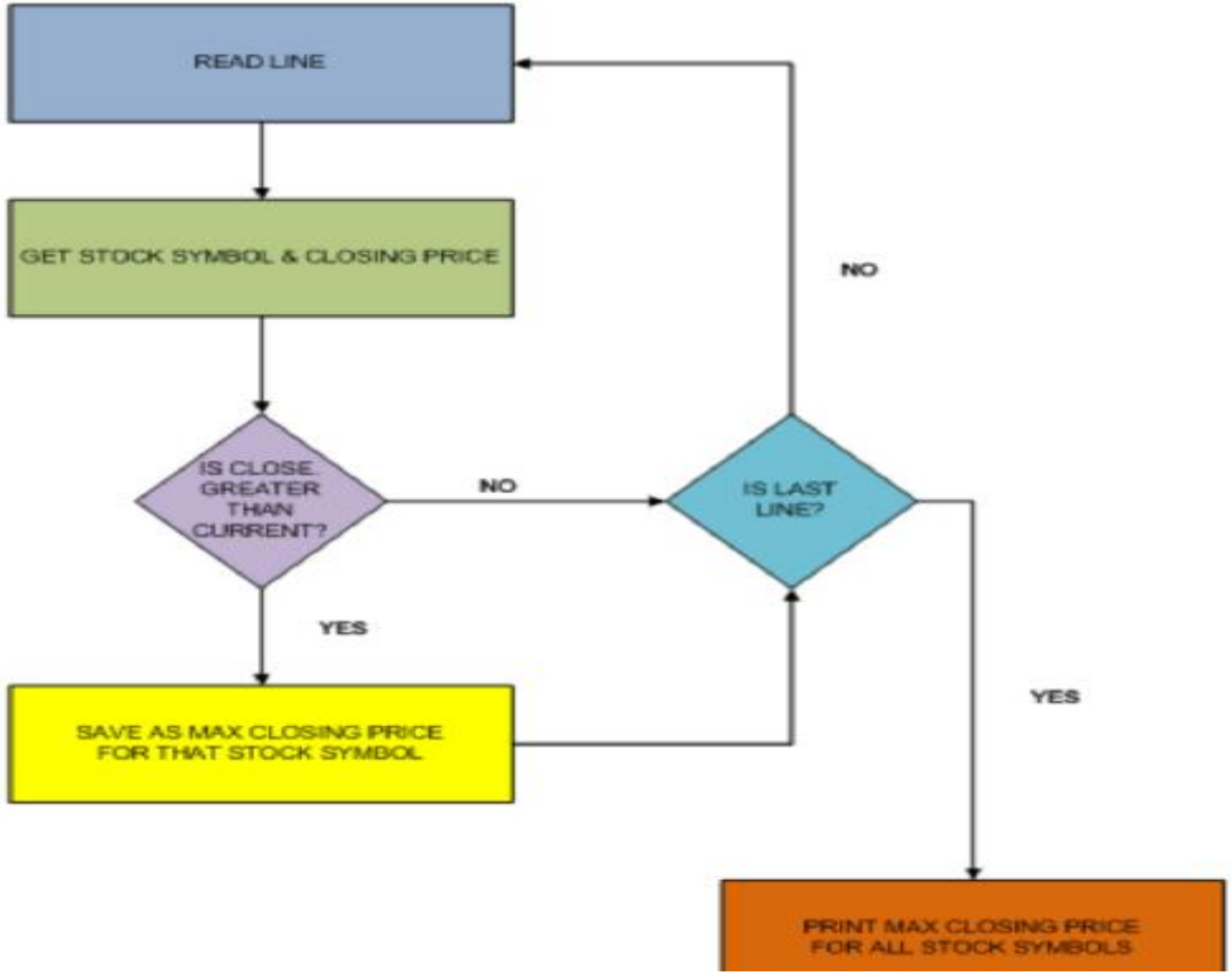
- Sample Stocks Dataset
- Each record has symbol, date, open, close...
- Find Maximum Closing Price for each symbol

```
ABCSE,B7J,2008-10-28,6.48,6.74,6.22,6.72,44300,5.79
ABCSE,B7J,2008-10-27,6.21,6.78,6.21,6.40,55200,5.51
ABCSE,B7J,2008-10-24,6.39,6.66,6.21,6.40,67400,5.51
ABCSE,B7J,2008-10-23,6.95,6.95,6.50,6.59,59400,5.68
ABCSE,B7J,2008-10-22,6.92,7.17,6.80,6.80,55300,5.86
ABCSE,B7J,2008-10-21,7.20,7.30,7.10,7.10,54400,6.11
ABCSE,B7J,2008-10-20,6.94,7.31,6.94,7.12,45700,6.13
ABCSE,B7J,2008-10-17,6.43,6.93,6.42,6.90,57700,5.94
ABCSE,B7J,2008-10-16,6.61,6.69,6.21,6.53,83200,5.62
ABCSE,B7J,2008-10-15,6.84,6.90,6.36,6.36,78900,5.48
ABCSE,B7J,2008-10-14,7.15,7.32,6.93,6.96,74700,5.99
ABCSE,B7J,2008-10-13,6.00,6.57,6.00,6.57,75700,5.66
ABCSE,B7J,2008-10-10,5.05,5.72,4.79,5.72,158400,4.93
ABCSE,B7J,2008-10-09,6.30,6.41,6.00,6.02,140500,5.18
ABCSE,B7J,2008-10-08,5.60,6.47,5.60,6.28,292000,5.41
ABCSE,B7J,2008-10-07,7.59,7.59,6.66,6.69,89900,5.76
ABCSE,B7J,2008-10-06,7.83,7.90,7.00,7.40,159600,6.37
```

MAX CLOSING PRICE ALGORITHM

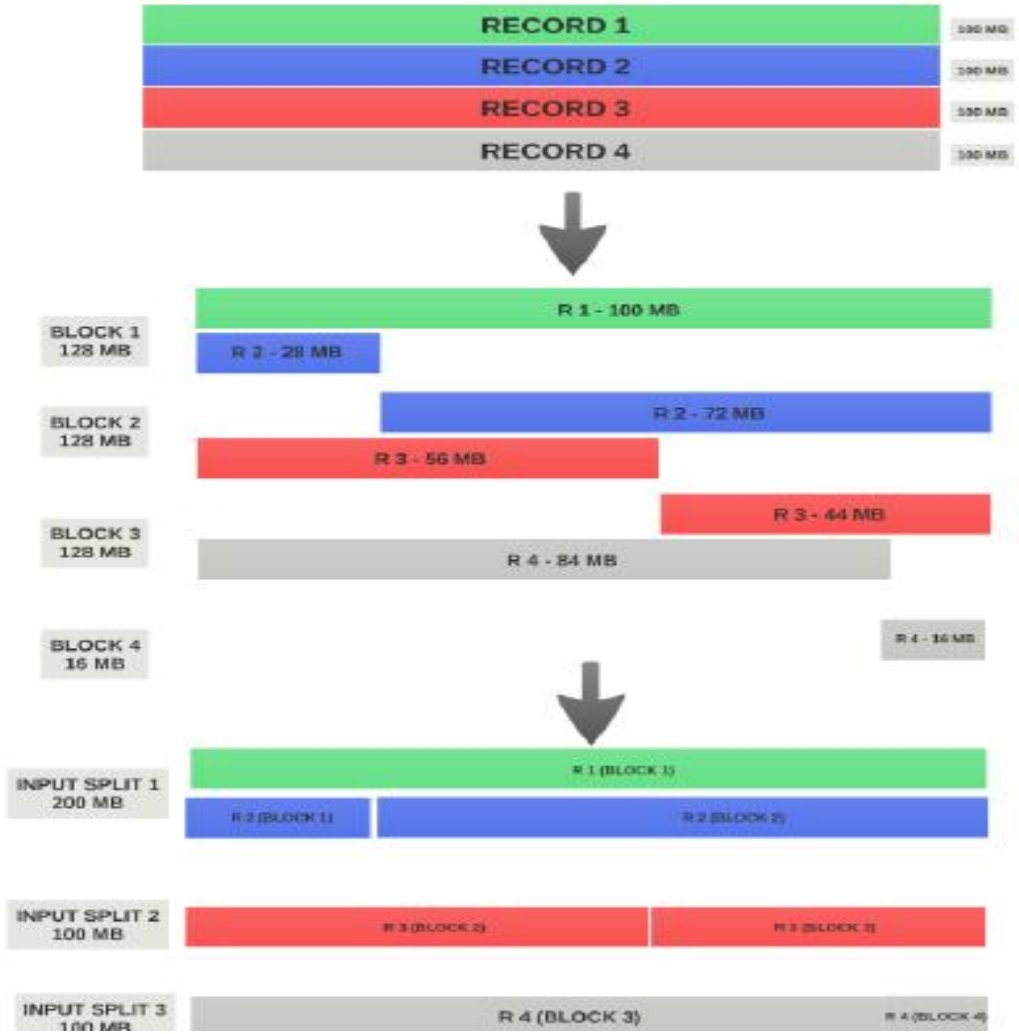
- One Node
- Not Distributed

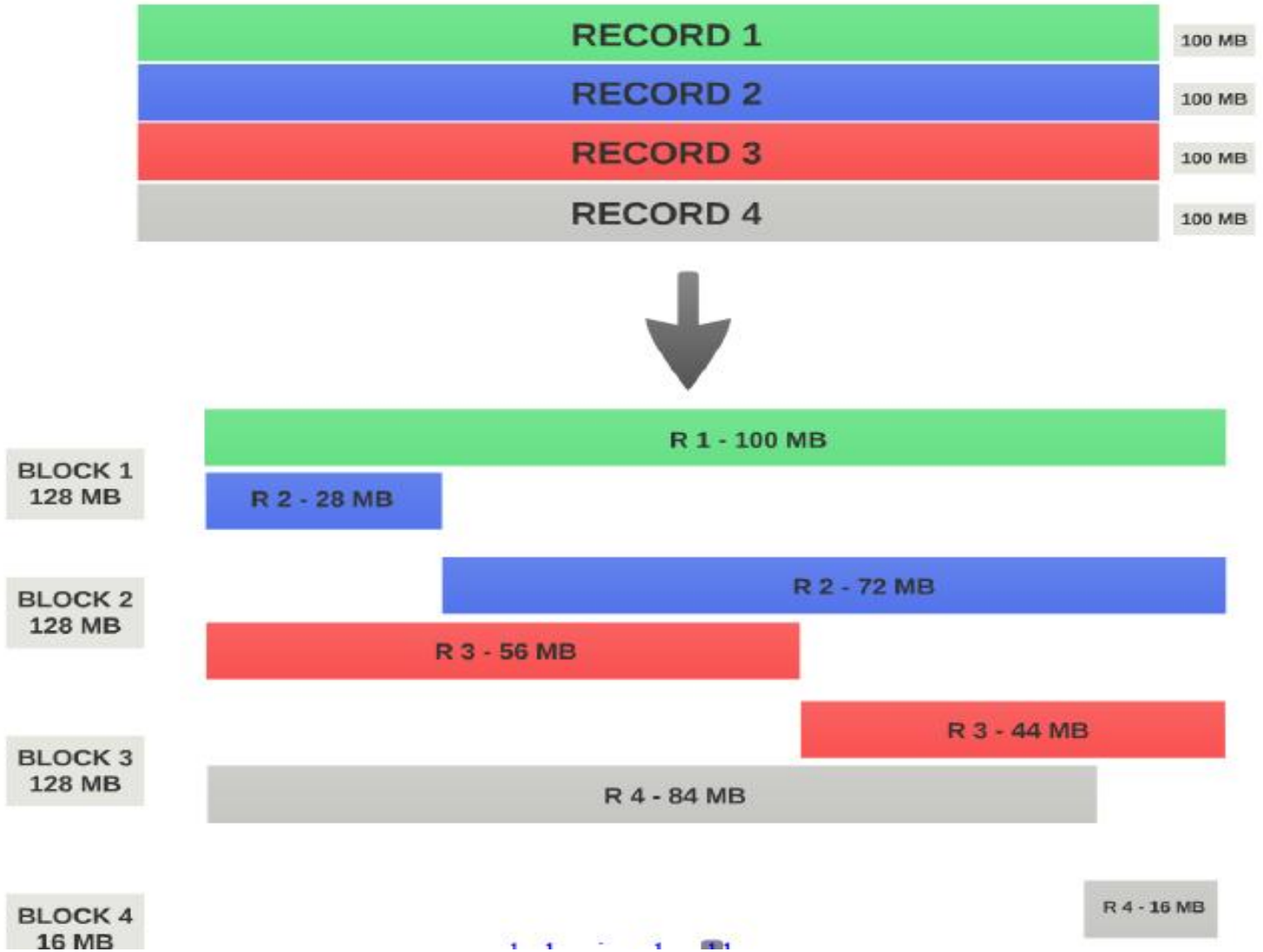


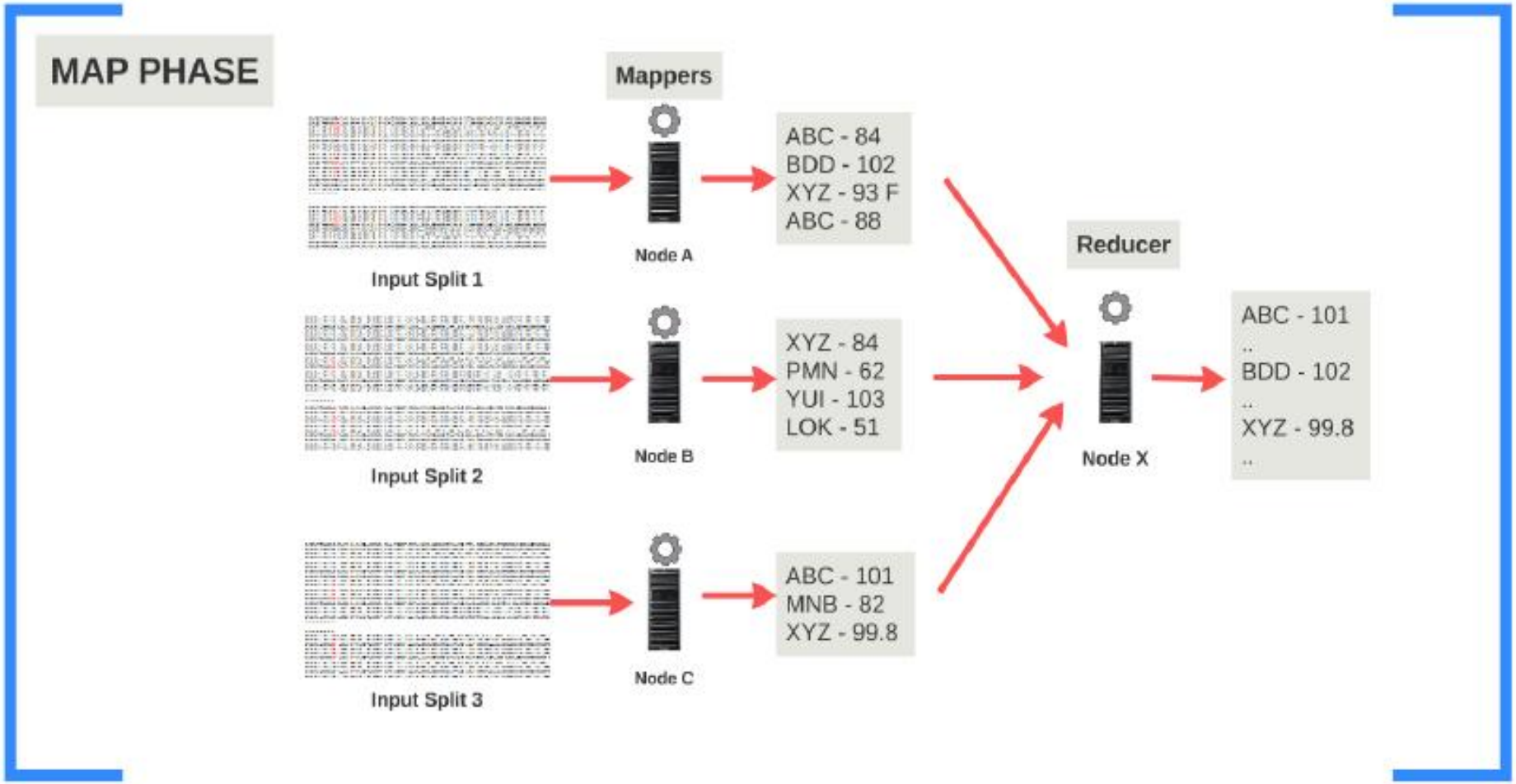

READ LINE
GET STOCK SYMBOL & CLOSING PRICE
IS CLOSE GREATER THAN CURRENT?
NO
YES
SAVE AS MAX CLOSING PRICE FOR THAT STOCK SYMBOL
IS LAST LINE?
NO
YES
PRINT MAX CLOSING PRICE FOR ALL STOCK SYMBOLS

DISTRIBUTED

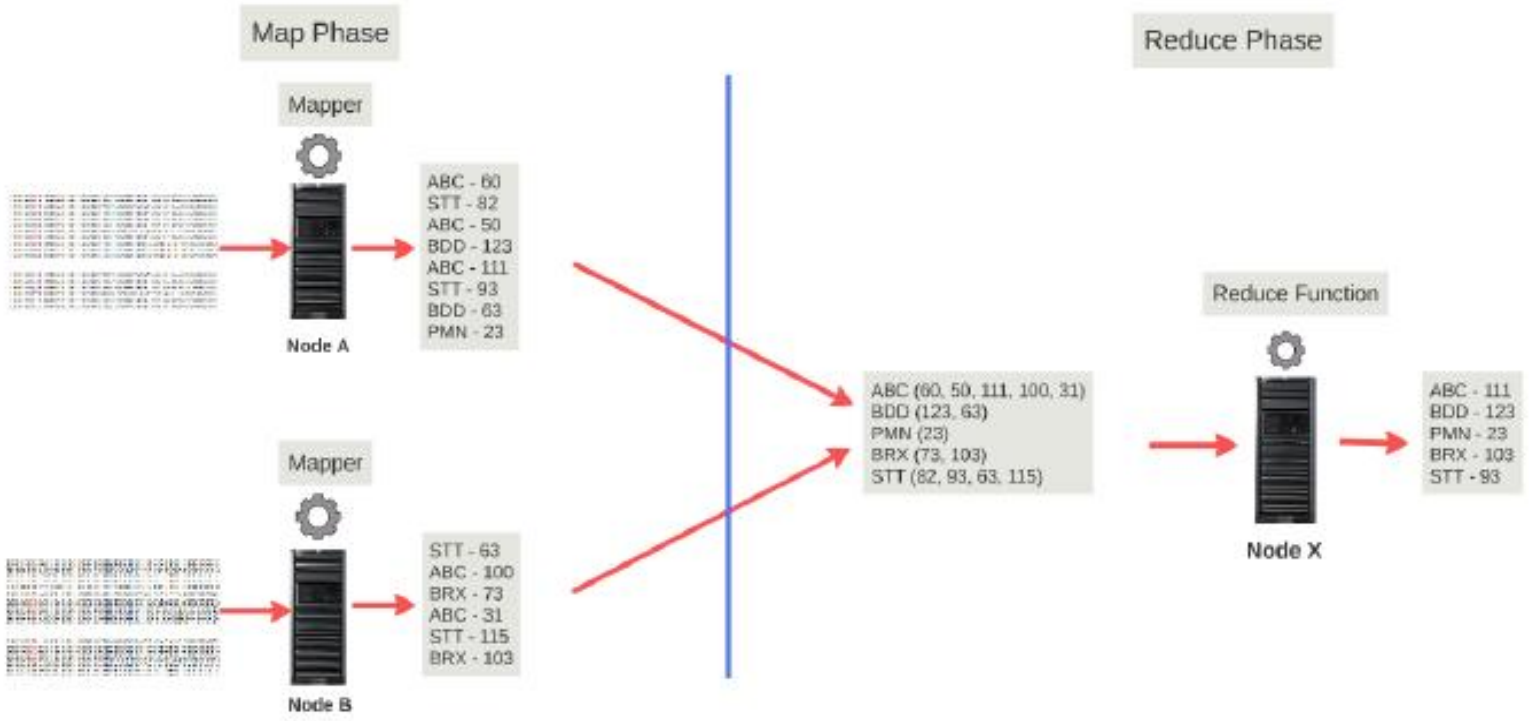
BLOCKS vs. INPUT SPLIT



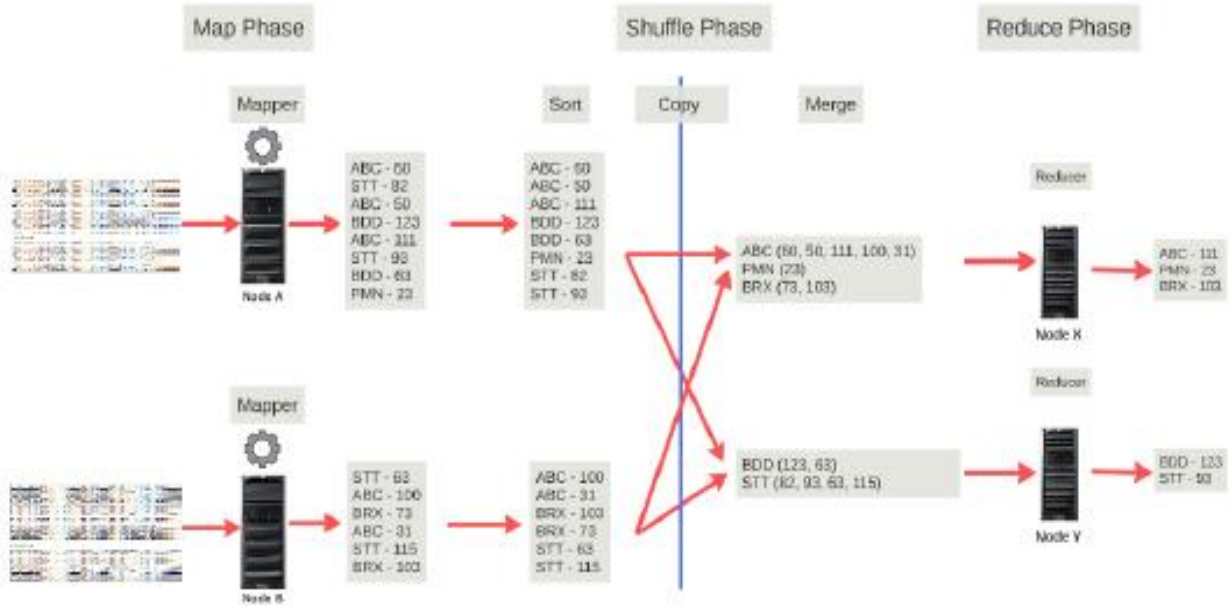

RECORD 1
100 MB
RECORD 2
100 MB
RECORD 3
100 MB
RECORD 4
100 MB
BLOCK 1
128 MB
R 1 - 100 MB
R 2 - 28 MB
BLOCK 2
128 MB
R 2 - 72 MB
R 3 - 56 MB
BLOCK 3
128 MB
R 3 - 44 MB
R 4 - 84 MB
BLOCK 4
16 MB
R 4 - 16 MB


MAP PHASE
Mappers
Input Split 1
Node A
ABC - 84
BDD - 102
XYZ - 93 F
ABC - 88
Input Split 2
Node B
XYZ - 84
PMN - 62
YUI - 103
LOK - 51
Input Split 3
Node C
ABC - 101
MNB - 82
XYZ - 99.8
Reducer
Node X
ABC - 101
..
BDD - 102
..
XYZ - 99.8
..

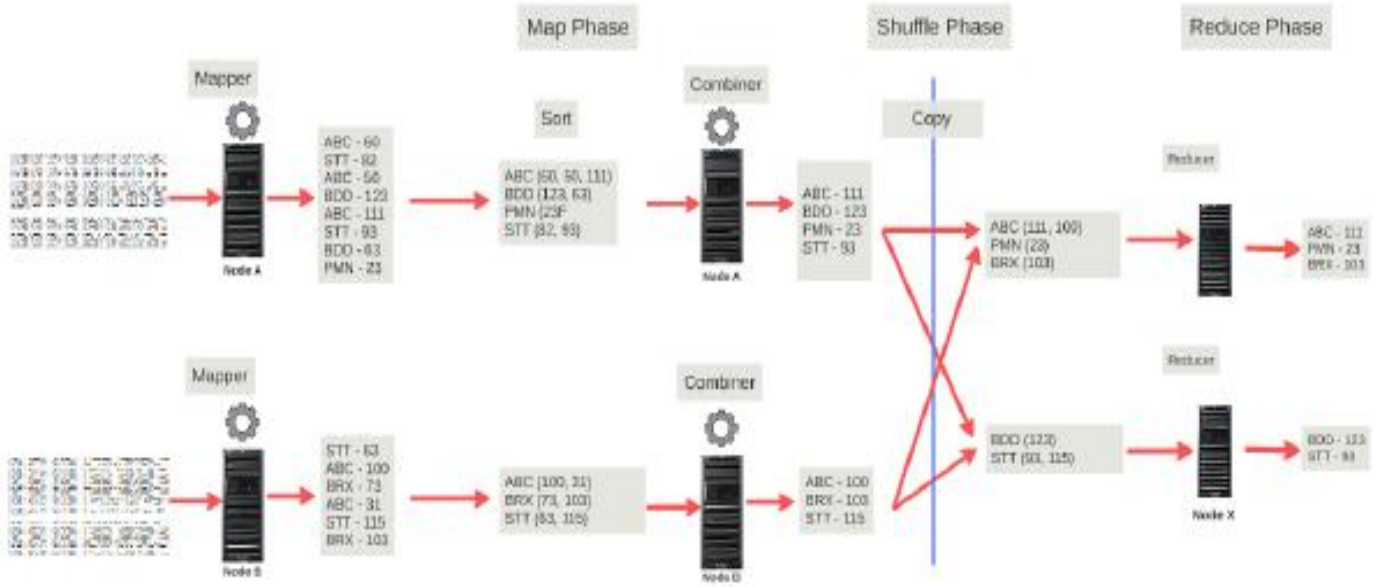
REDUCE PHASE



MULTIPLE REDUCERS



COMBINER (OPTIONAL)



STEPS FOR WRITING HADOOP MAP REDUCE PROGRAM

Step-1: Creating a File on Local System (file.txt)

$ cat > file.txt
Hi how are you
How is your class
How is your Big Data
What is time for lecture
What is the strength of Hadoop and Spark

Note: Use ctrl+d to save and make exit.

Step-2: Loading file.txt from local system to HDFS

$ hadoop-2.7.7/bin/hadoop fs –put file.txt file

Step-3: Writing Programs for Processing File

DriverCode.java
MapperCode.java
ReducerCode.java

Step-4: Compiling all .java Programs:

$javac –classpath $HADOOP_HOME/hadoop-core.jar *.java

Step-5: Creating jar Files:

$ jar cvf test.jar *.class

Step-6: Running test.jar on file (Already on HDFS)

$ hadoop jar test.jar DriverCode file TestHadoop

Word Count Program With Map Reduce and Java:

1. Download and Install Eclipse IDE

Eclipse is the most popular Integrated Development Environment (IDE) for developing Java applications. It is robust, feature-rich, easy-to-use and powerful IDE which is the #1 choice of almost Java programmers in the world. And it is totally FREE.

- As of now (fall 2018), the latest release of Eclipse is Photon. Click the following link to download Eclipse:

  http://www.eclipse.org/downloads/eclipse-packages

- The following file will get downloaded for Linux:

  eclipse-committers-photon-R-linux-gtk.tar.gz

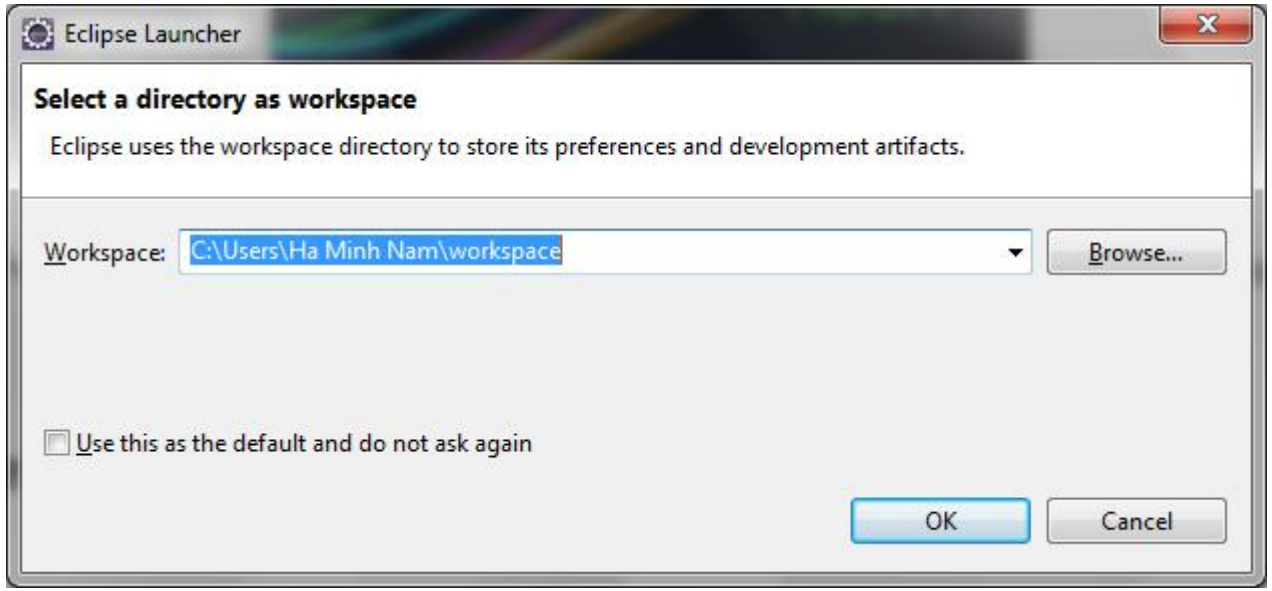
- Unzip the file using following command:

  $ tar –zxvf eclipse-committers-photon-R-linux-gtk.tar.gz

- Move eclipse folder in Home Directory
- Open eclipse-> eclipse

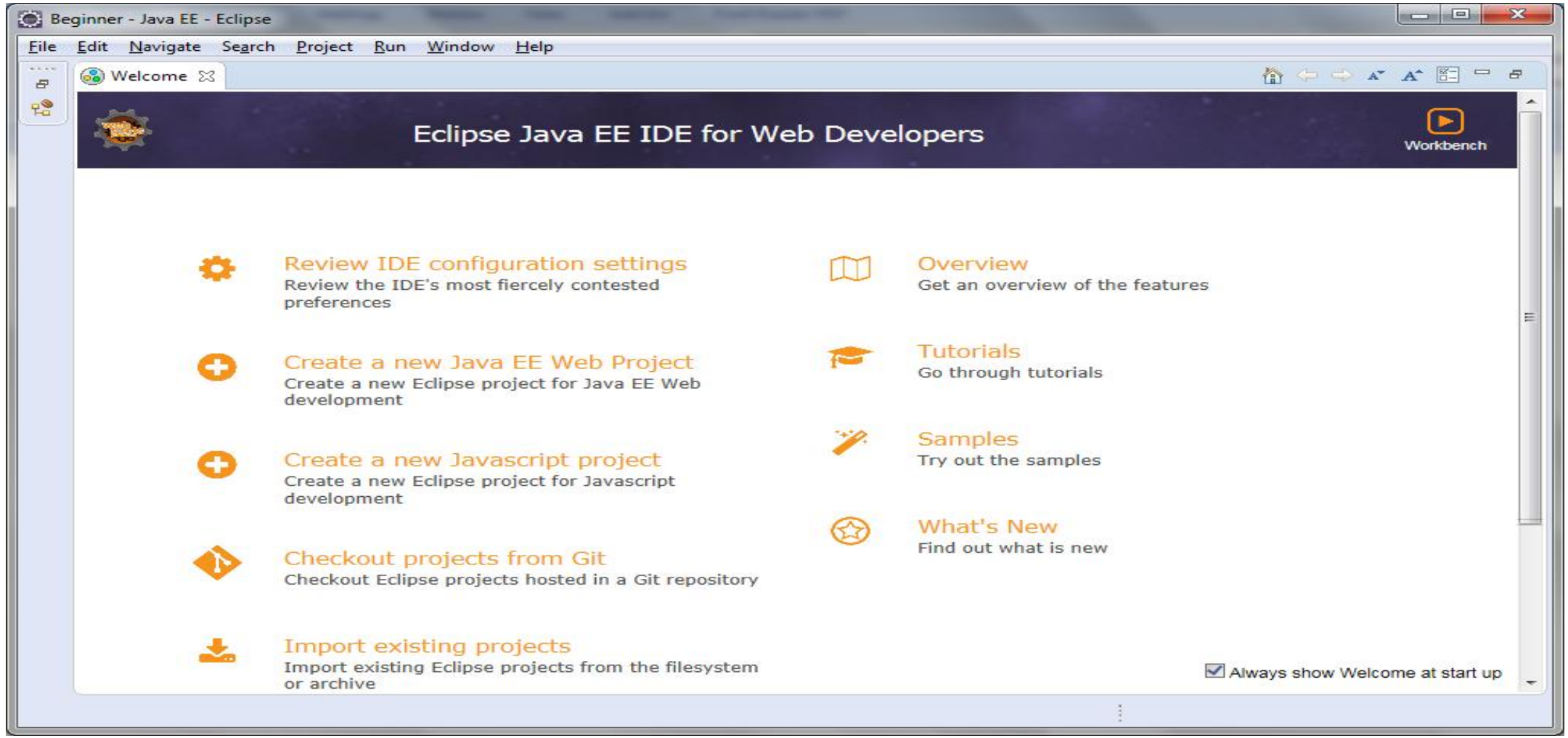
2. Choose a Workspace Directory

Eclipse organizes projects by workspaces. A workspace is a group of related projects and it is actually a directory on your computer. That's why when you start Eclipse, it asks to choose a workspace location like this:



If you want to choose another directory, click **Browse**

Click **OK**. You should see the welcome screen:


Beginner - Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
Eclipse Java EE IDE for Web Developers
Workbench
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Create a new Java EE Web Project
Create a new Eclipse project for Java EE Web development
Create a new Javascript project
Create a new Eclipse project for Javascript development
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Always show Welcome at start up

3. Create Java Project in Package Explorer:

- File > New > Java Project >(Name it – WordCountProject) > Finish
- Right Click > New > Package (Name it - WordCountPackage) > Finish
- Right Click on Package > New > Class (Name it - WordCount)
- Add Following Reference Libraries –
  1. hadoop-core-1.2.1.jar
  2. commons-cli-1.2.jar
- Right Click on Project > Build Path> Add External Archivals
  Downloads/hadoop-core-1.2.1.jar
  Downloads/commons-cli-1.2.jar

4. Type following Java Program for Driver, Mapper and Reducer :

```
package WordCountPackage;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
     Text outputKey = new Text(word.toUpperCase().trim());
  IntWritable outputValue = new IntWritable(1);
  con.write(outputKey, outputValue);
}
}
}
```

```
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException
{
int sum = 0;
  for(IntWritable value : values)
  {
  sum += value.get();
  }
  con.write(word, new IntWritable(sum));
}
}
}
```

The program consist of 3 classes:

- Driver class (Public void static main- the entry point)
- Map class which extends public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Map function.
- Reduce class which extends public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

5. Make Jar File:

Right Click on Project> Export> Java>Select export destination as **Jar File** > next> Finish

Jar File Path: /home/mamoon/temp.jar

6. Create a Text File and Move it to HDFS:

File Contents:

Apple, Banana, Mango,Apple,Mango, Banana, Apple, Banana, Mango,Apple,Mango, Banana

To Move this into Hadoop directly, open the terminal and enter the following commands:

$ hadoop-2.7.7/bin/hadoop fs –put /home/mamoon/fruits.txt demofruits.txt

7. **Run Jar file**

$ hadoop-2.7.7/bin/hadoop jar temp.jar WordCountPackage.WordCount demofruits.txt testfruitsoutput

8. Display Output:

$ hadoop-2.7.7/bin/hadoop fs -cat testfruitsoutput/part-r-00000

APPLE 7
BANANA 7
MANGO 7

Explanation of Map Reduce Computation:

Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel cross a cluster of servers. The results of tasks can be joined together to compute final results.

MapReduce consists of 2 steps:

Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).
Example – (Map function in Word Count)

put	Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),

Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.
Example – (Reduce function in Word Count)

Input (output of Map function)	Set of Tuples	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)
Output	Converts into smaller set of tuples	(BUS,7), (CAR,7), (TRAIN,4)

Work Flow of Program

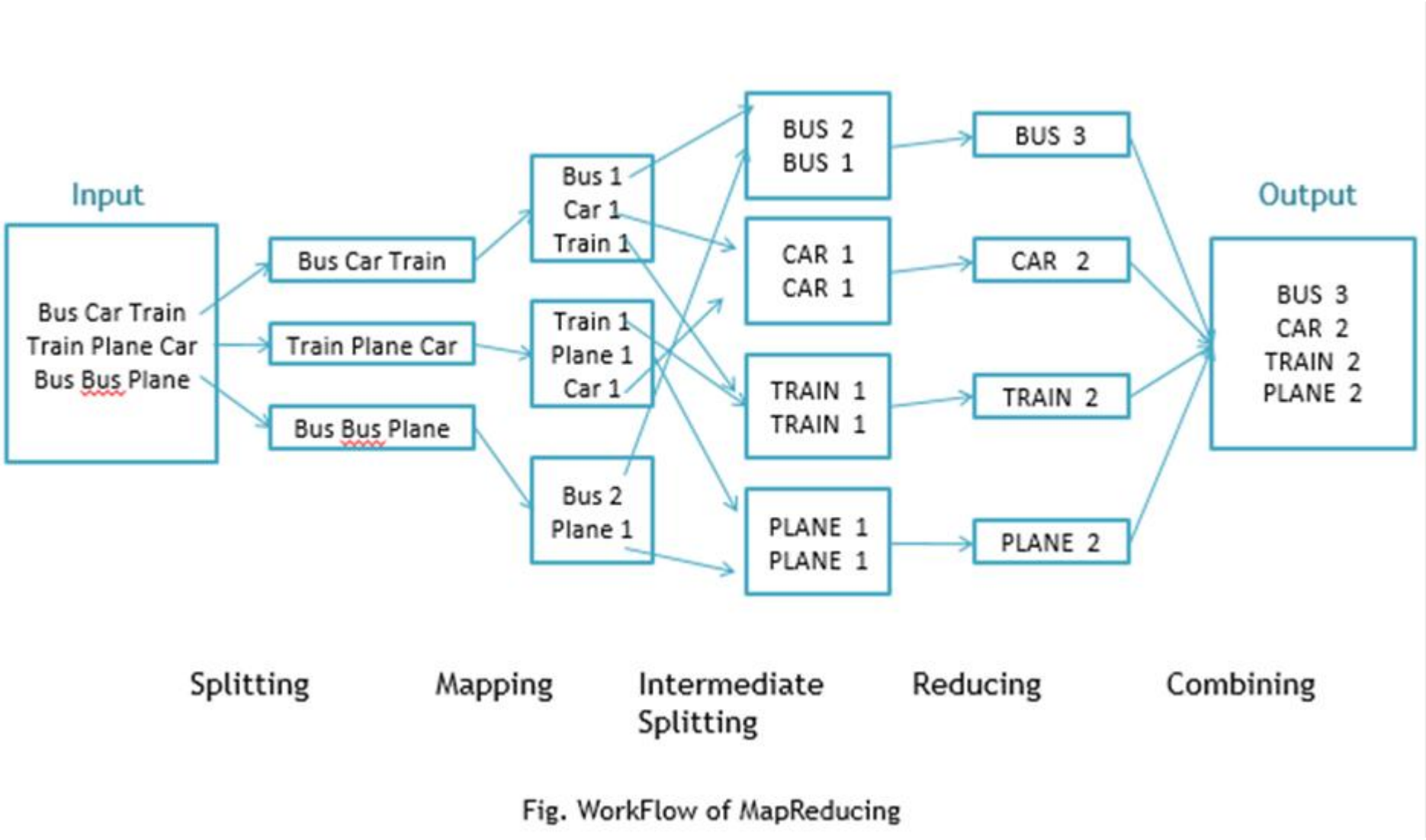


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps:

Splitting – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).

Mapping – as explained above

Intermediate splitting – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on same cluster.

Reduce – it is nothing but mostly group by phase

Combining – The last phase where all the data (individual result set from each cluster) is combine together to form a Result

MapReduce Design Pattern:

To solve any problem in MapReduce, we need to think in terms of MapReduce. It is not necessarily true that every time we have both a map and reduce job.

Input-Map-Reduce-Output
Input-Map-Output
Input-Multiple Maps-Reduce-Output
Input-Map-Combiner-Reduce-Output

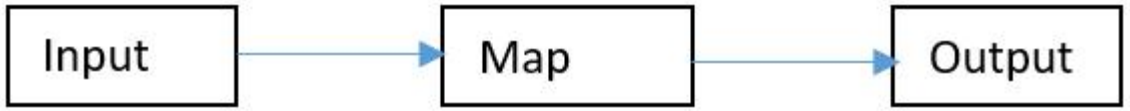
Following is a real time scenario to understand when to use which design pattern.

- Input-Map-Reduce-Output

If we want to do some aggregation then this pattern is used:

Scenario	Counting gender total/ average salary of employees
Map (Key, Value)	Key: Gender Value: Their Salary
Reduce	Group by Gender And Take Total of salary for each group

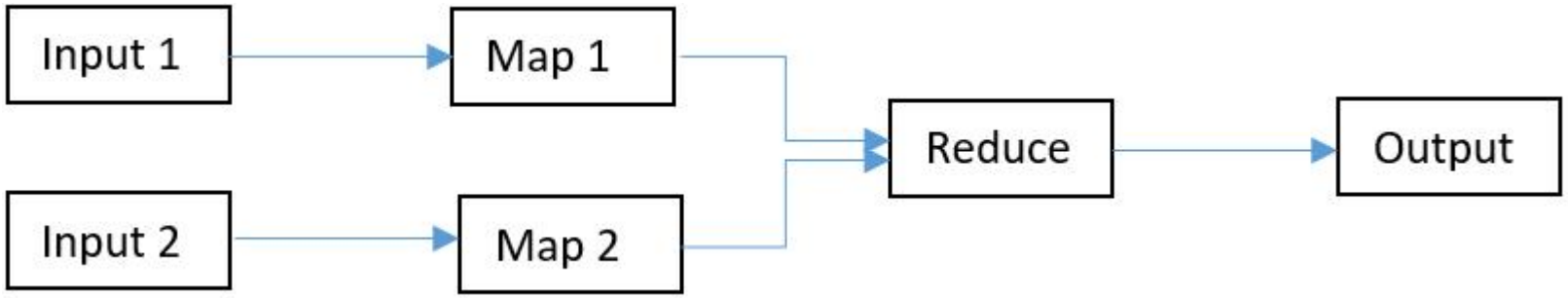
- **Input-Map-Output:**



If we want to change only the format of data then this pattern is used:

Scenario	Some employees have gender entry as "Female", "F","f",O
Map (Key, Value)	Key : Employee Id Value : Gender -> if Gender is Female/ F/ f/ O then converted to F else if Gender is Male/M/m/1 then convert to M

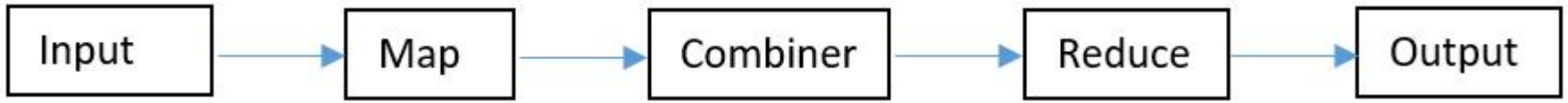
- **Input-Multiple Maps-Reduce-Output**



In this design pattern, our input is taken from two files which have a different schema:

We have to find the total of gender-wide salary. But he have 2 files with different schema. Input File 1 Gender is given as a prefix to Name Eg. Ms. Shital Katkar Mr. Krishna Katkar Input File 2 There is a different column for gender. However, the format is mixed. Eg. Female/Male, O/1, F/M	
Map (Key, Value)	**Map 1 (For input 1)** We need to write a program to split prefix from Name and, according to the prefix, determine the gender. Then prepare the Key value pair (Gender,Salary). **Map 2 (For input 2)** Here the program will be straightforward. Resolve the mixed format and make a key value pair (Gender,Salary).
Reduce	Group by Gender And Take Total of salary for each group

- Input-Map-Combiner-Reduce-Output:



A Combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class. Purpose of the combiner is to reduce workload of Reducer.

In MapReduce program, 20% of the work is done in the Map Stage, which is also known as the data preparation stage, which works in parallel.

80% of the work is done in Reduce stage which is known as the calculation stage, and it is not parallel. Therefore it is slower than the Map phase. To reduce time, some work in the Reduce phase can be done in the combiner phase.

Scenario
There are 5 departments. And we have to calculate the gender-wide total salary. However, there are certain rules to calculate the total. After calculating gender wise total for each department, if the salary is greater than 200K, add 20K in total, if the salary is greater than 100K, add 10K in total

Input Files (for each department there is 1 file)	Map (Parallel) (, Value = Salary)	Combiner (Parallel)	Reducer (Not Parallel)	Output
Dept 1	Male<10,20,25,45,15,45,25,20> Female <10,30,20,25,35>	Male <250,20> Female <120,10>	Male < 250,20,155, 10,90,90,30> Female <120,10,175,10,135, 10,110,10,130,10>	Male <645> Female <720>
Dept 2	Male<15,30,40,25,45> Female <20,35,25,35,40>	Male <155,10> Female <175,10>		
Dept 3	Male<10,20,20,40> Female <10,30,25,70>	Male <90,00> Female <135,10>		
Dept 4	Male<45,25,20> Female <30,20,25,35>	Male <90,00> Female <110,10>		
Dept 5	Male<10,20> Female <10,30,20,25,35>	Male <30,00> Female <130,10>		