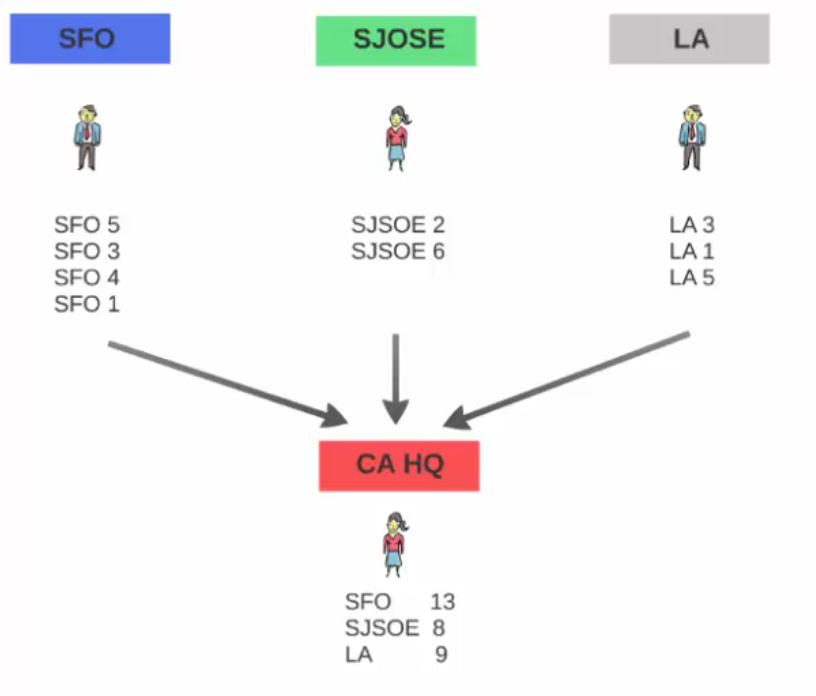
# MAP Reduce Intro

Lets take an example , you are the head of census and has been asked to find the population of all cities in california, you can use as much resuorce as you want, but have been provided with time line of only 4 month.

**How this operation can be done?**

Single person cannot calculate the population of cmplte state, so best solution would be to divide teh state by city and make individual incharge of each city.

Each individual will collect the data from individual city and provide the input to state HQ. Where they will finally consolidate the data and calculate the population for the entire state.

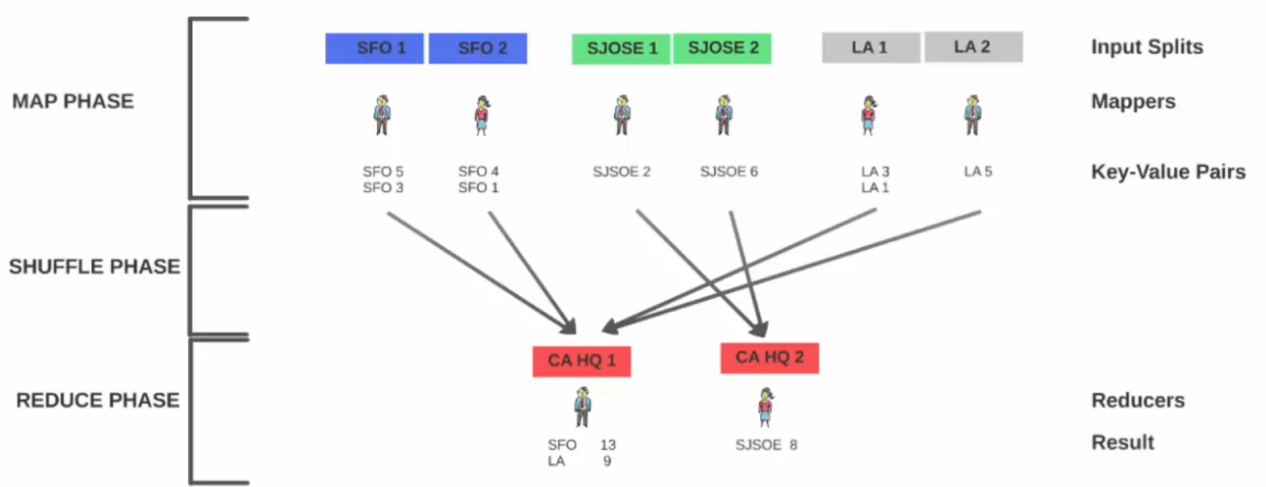


Next time the same operatino is been asked to perform with a time limit of 2 month.

In this case, you will simply double the number of person doing the task, 2 person assigned to each city. Each member will divide the locality in the city and get the information and share the colected info to 1 of the state HQ.

**Note: each person assigned to same city, shoud share the input to same state HQ. Cannot send the data to different HQ.**

**This is a map redcuce model, so next year if asked to get the data in a month, add more member to get the sensus and consolidate it.**



# What is MapReduce?

<https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm>

**MapReduce is a framework** using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

**MapReduce is a processing technique** and a program model for distributed computing based on java. **The MapReduce algorithm contains two important tasks, namely Map and Reduce.**

**Map** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

Secondly, **reduce task**, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

**Advantage:**

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes.

Under the MapReduce model, the data processing primitives are called mappers and reducers.

Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, **once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change**.

This simple scalability is what has attracted many programmers to use the MapReduce model.

# The Algorithm

Generally MapReduce paradigm is based on sending the computer to where the data resides!

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.

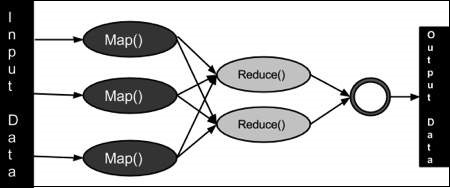
**Reduce stage** − This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

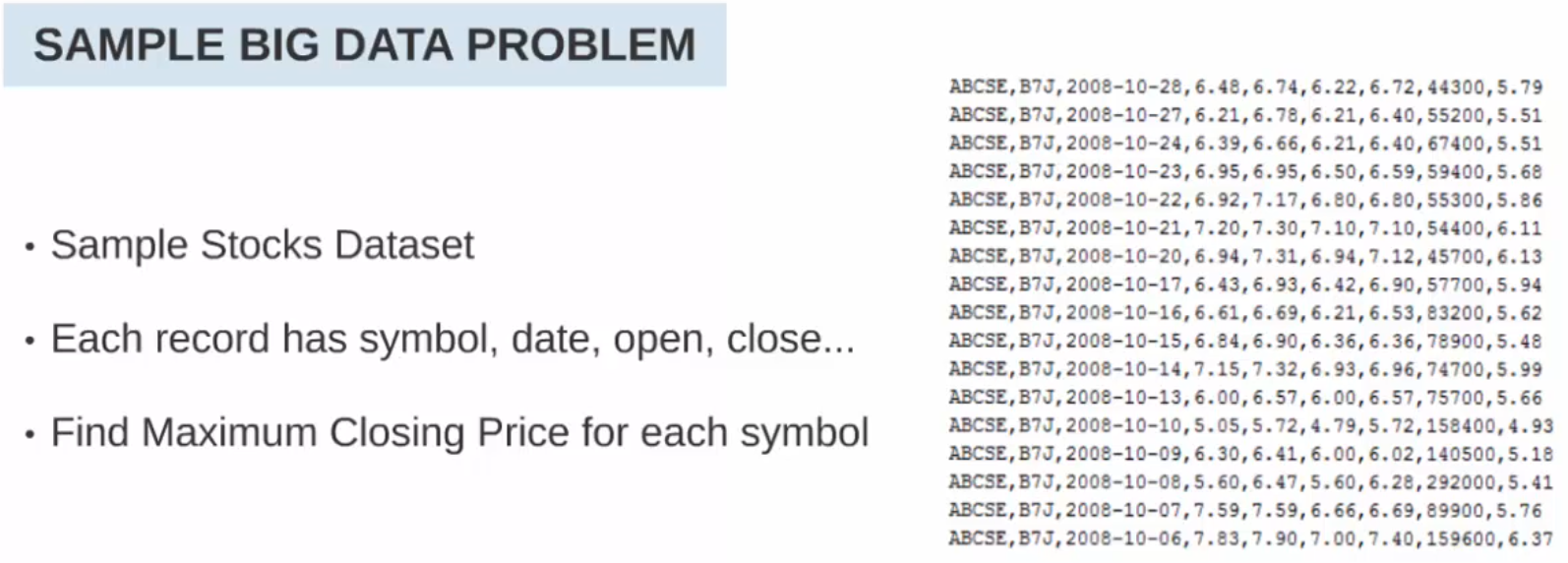
Most of the computing takes place on nodes with data on local disks that reduces the network traffic.

After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

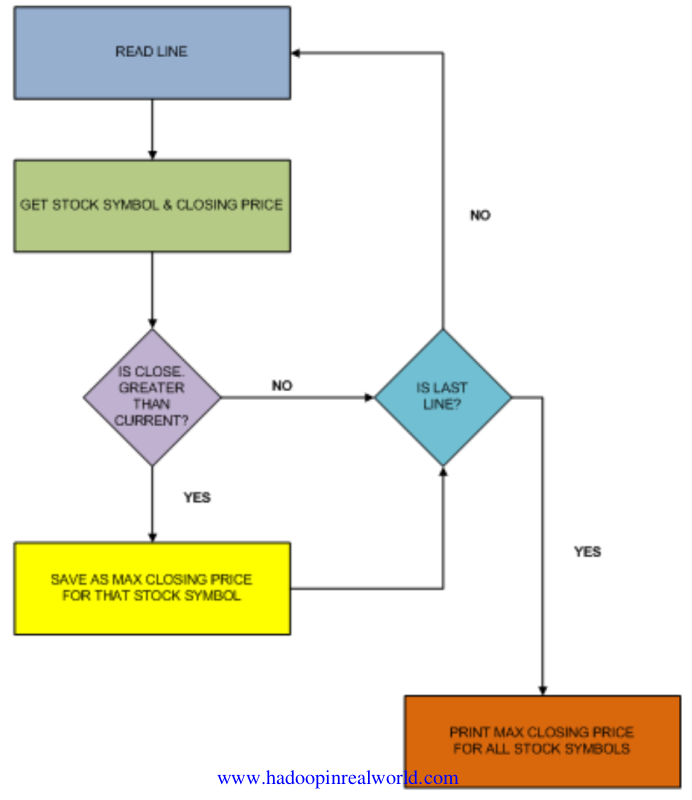


# Map reduce using exampl:

Lets assume you have a sample data set of stock exchange and need to find the maximum closing price of each stock.



If you want to solve this problem without the map reduce solution, the flow chart will be like following



If you have a huge data set, then this step will take a long to compute.

Let see how the same proble can be solved using the map reduce solution.

## Map phase:

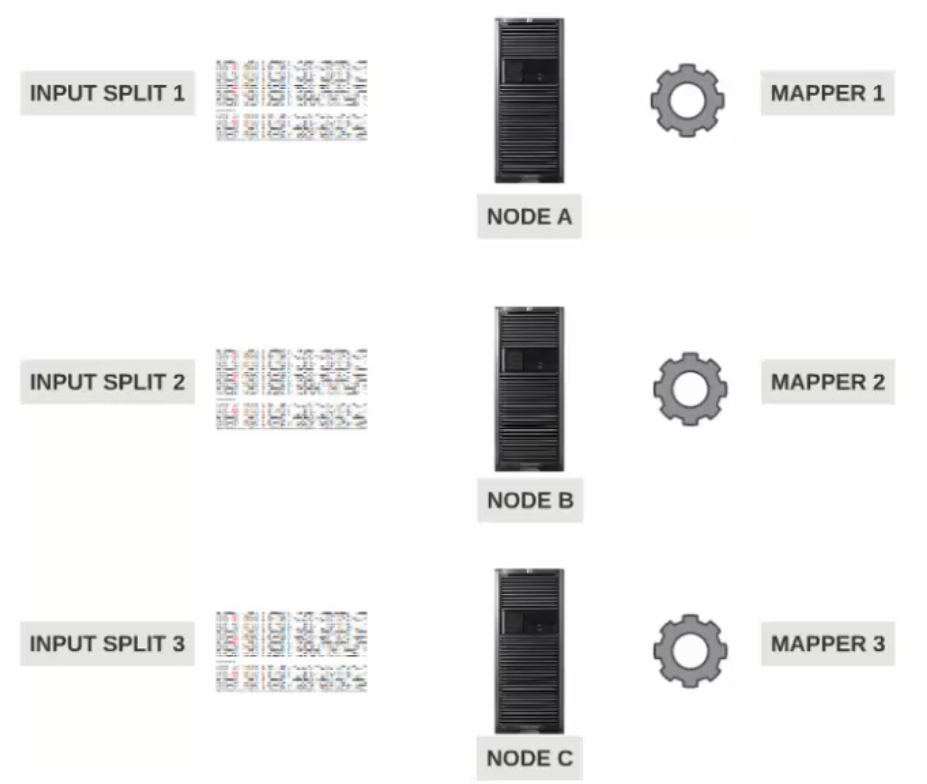
- Divide the data in to chunks known as **input splits.**

**-** the process working on the input splits are called **mappers**

**-** each mapper process one recod at a time.

**-** each mapper will execute the same code on each record**.**

**-** the output of mapper is a **key value pair.**



**What is Input split?**

**Input split is not same as HDFS block.**

A hdfs block can contain more than 1 record

Assuming the record size is 100 MB. And each block is of size 128 MB

1st block will contain 100MB of first record and 28 MB of second record

2nd block will contain 72 mb of 2nd record and 56 mb of third record

3rd block will contain 44mb of 3rd record and 84 mb of 4th record

And so on...

**In this case if we provide the block 1 to mapper, then record 2 cannot be processed by mapper. Since the record 2 is not complete.**

This problem is solved by input split.

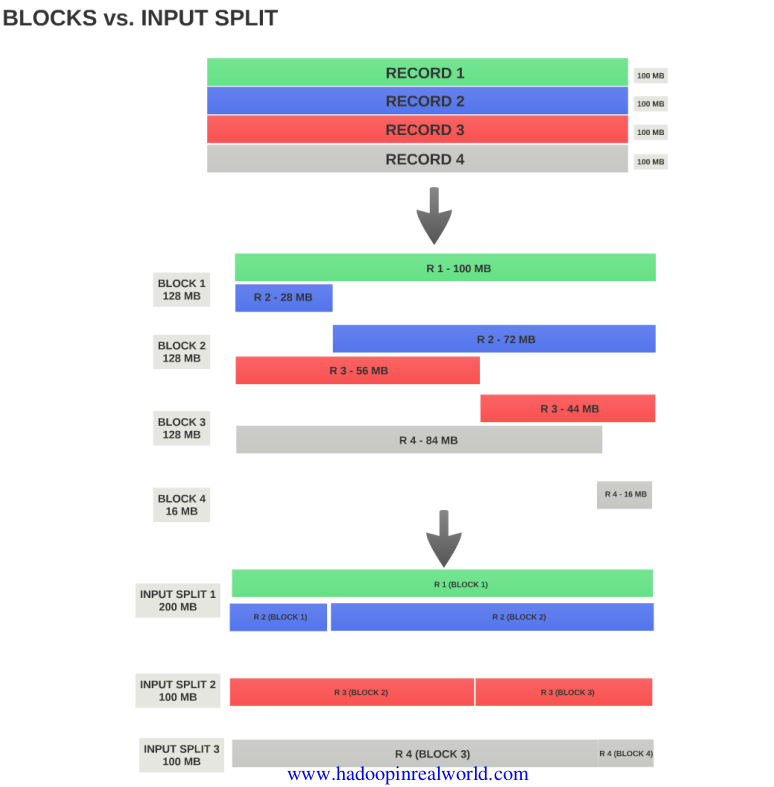
Input split contains the complete record, so the mapper can process the record completly.

**Input split is not pyhysical chunks of data**. They are the start and end pointers within blocks. So when the mapper tries to read the data, it clearly knows where to start and where to end.

The inut split can start in 1 block and end in another block.

Input split respect logical record boundary.

During map reduce process executed, hadoop scans the block and create input split which respect logical record boundary.



Lets go back to map phase

The mapper in hadoop can be written in many language, java scala, python etc....

The mapper is invoked once for every record in a input split. If the inpuot split has 100 record then the mapper code is invoked 100 time.

**How many mapper does hadoop created?**

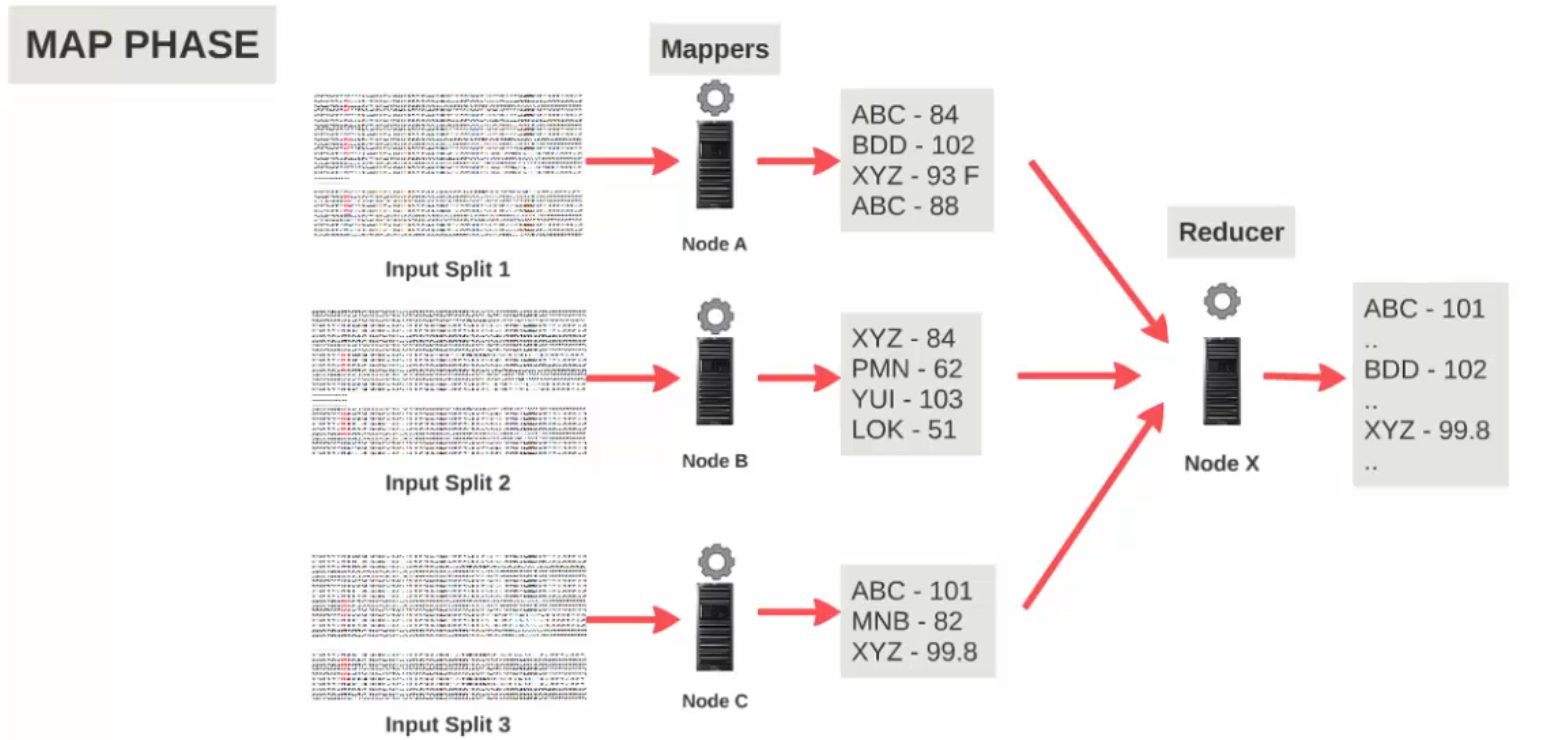
The number of mapper is completely dependent on the number of inout splits.

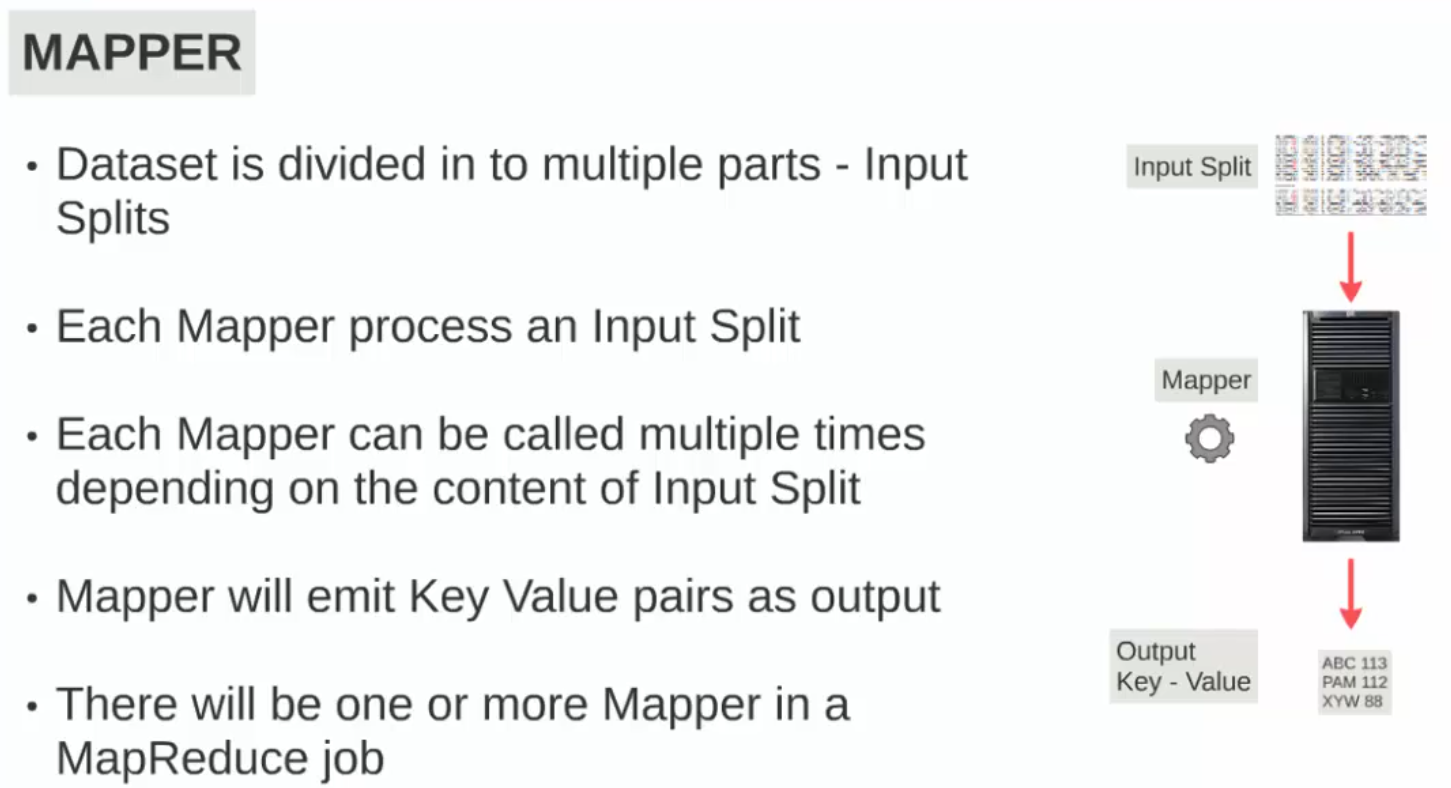
If there are 10 input splits then there are 10 mapper created by hadoop

The output of the mapper is a key value pair. For the stock problem, the ouput value for each record is the stock and the closing price for that record.

**What should be the key and what should be the value?**

This depend on what to reduce, check the reduce phase section.





## Reduce phase:

the reducer work on the output of mapper, The reducer will recieve a key and the list of values for that key as input. The keys are grouped before sending to reducer(this happen in **shuffle phase**).

Suppose we have 10 stock with 100 closing price for each. Then there are 1000 records in the input split.

The output of the mapper will be 1000 key value pair

The inout to the reducer will be 10 key value pairs, 1 record for each stock and the list of closing price as values, this is sufficient for the reducer to find the maximum closing price for each stock

**What should be the key and what should be the value?**

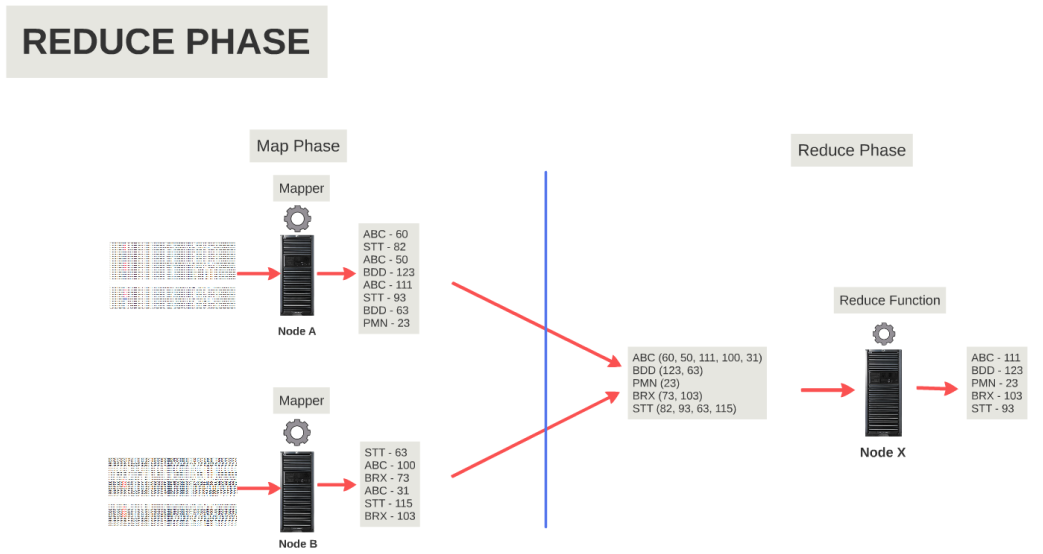
Think what need to be reduced.

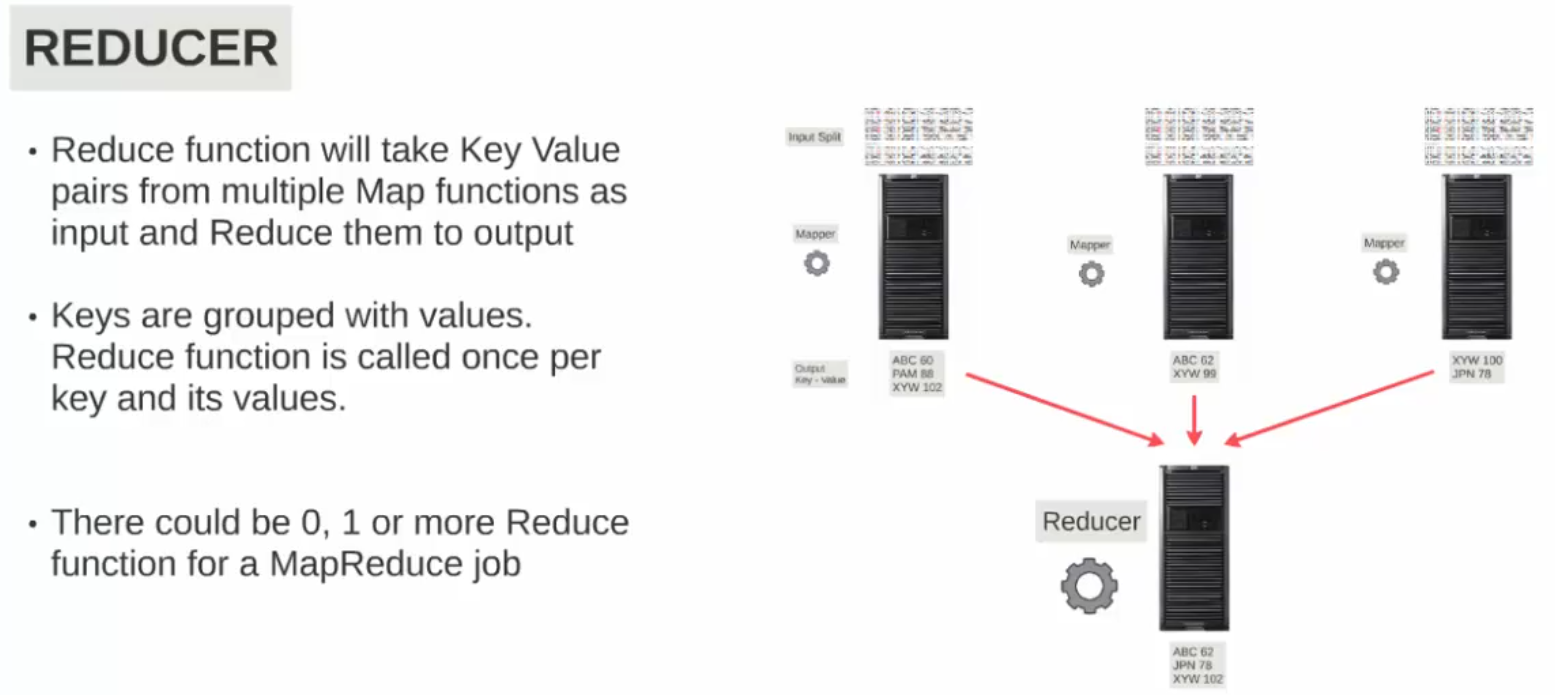
In this stock problem, we know that if the reducer has the stock and the list of closing price, the reducer can calculate the maximum closing price for each stock.

Also the reducer to be called once for each stock. That is why the stock was made as key and closing price and the value.

**We know that the number of mapper is same as number of input splits, this cannot be controlled by users, but the number of reducer can be set by users. Can have a map reduce process with no reducers.**

**If there is only 1 reducer which is processing input from many mapper say 100, this can be a performane bottle neck. It is always advisable to have more than 1 reducer for large data set.**





## Shuffle phase:

We know the individual output from each mapper and group by key and provided as input to reducers. This is done in shuffle phase.

Shuffle phase is a key component in map reduce,

The shuffle phase is a process in which the map output is transfered to reducer.

**In the map phase each key is assigned to a partition.**

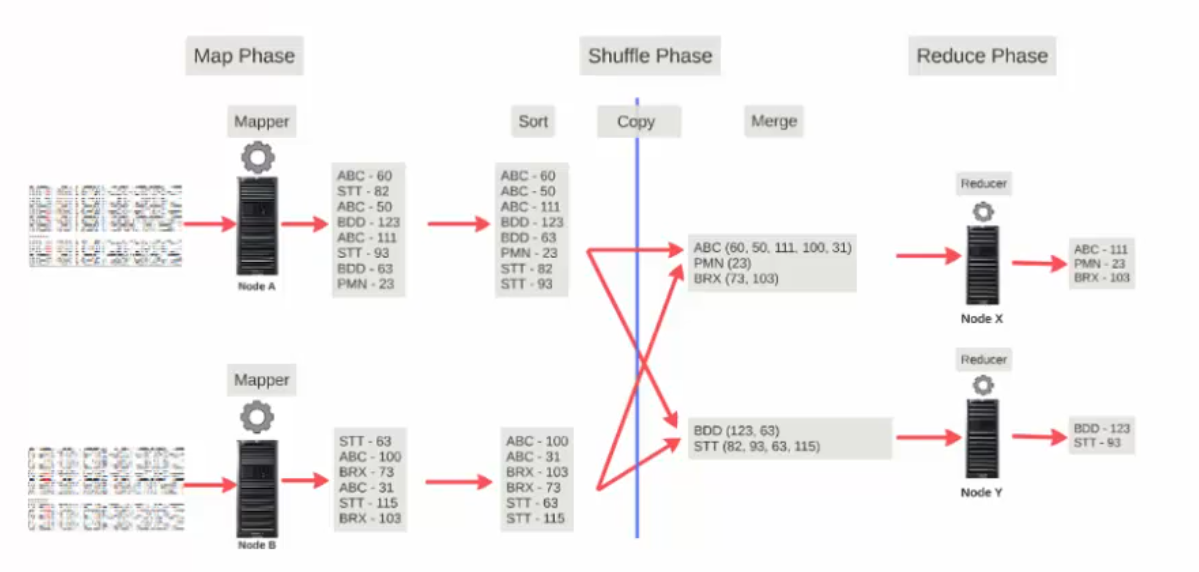
**So if there are 3 reducers, there are 3 patition.**

**The output from each mapper can have same key in a key value pair, similar key are assigned to same paritition.**

**During the shuffle phase the key value pair data in the same partition are sorted based on key**

**Once the key value pair are sorted in a partition, now its ready to copy each parition to appropriate reducer. During copy similar key are merged/grouped and key values pair are formed.**

**Then the key value pairs are send to single reducer for processing.**

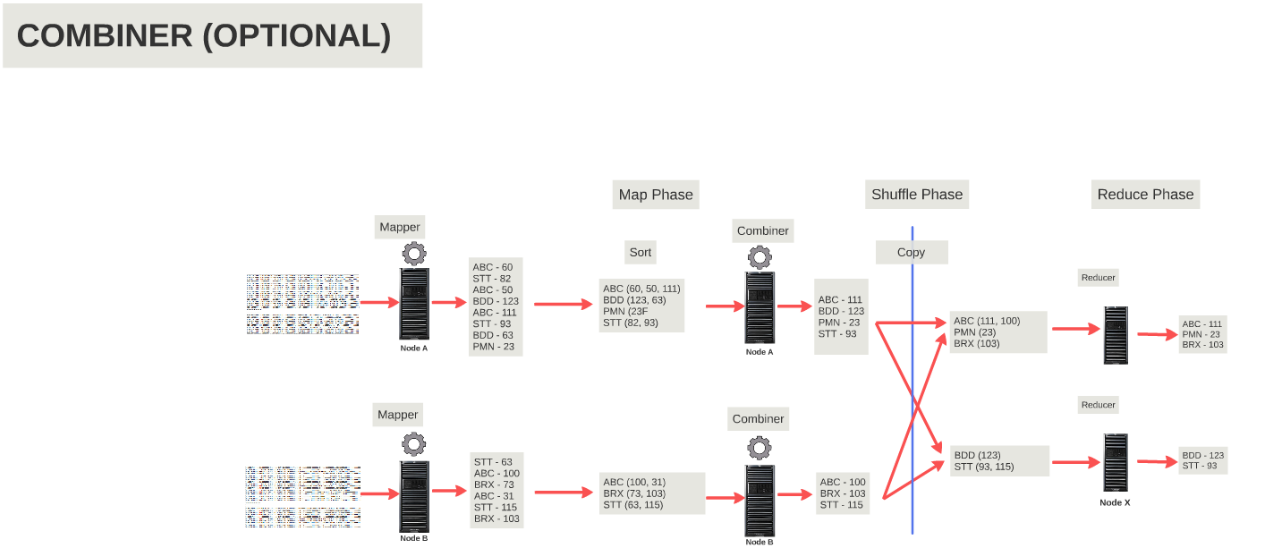


## Combiners:

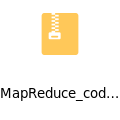
Combiners are otional in map phase

Combiners can be considerd as mini reducers in map phase, they are used to reduce the amount of data which is sent to reduce phase.

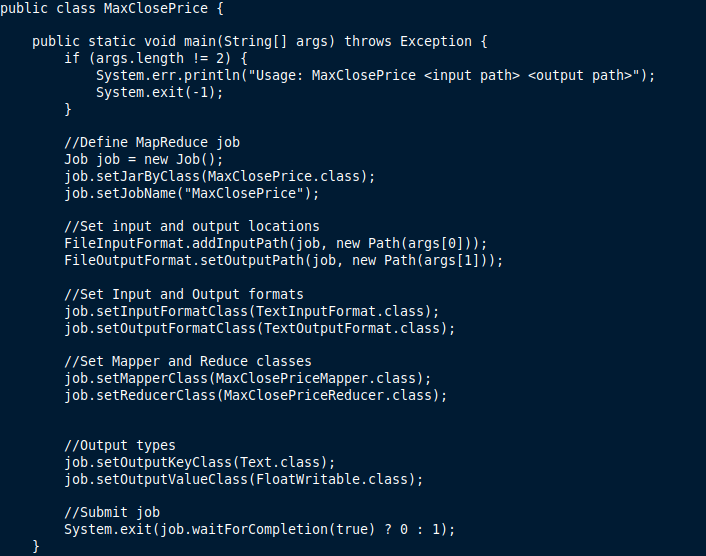
In the stock example we can see that each mapper has multiple key value pair wirh same key. So the combiner can be used to get the single key value pair from each mapper and then send to reducer. This will reduce the amount of data sent to reducer, which increases the reducer performance.



# Code for map reduce process



**Driver Class:**



This is the driver class.

- specify the niout and output folder

- define a hadoop job which specify the class which need execute.

- set the inout and output path for the job

- set the format for input and output file

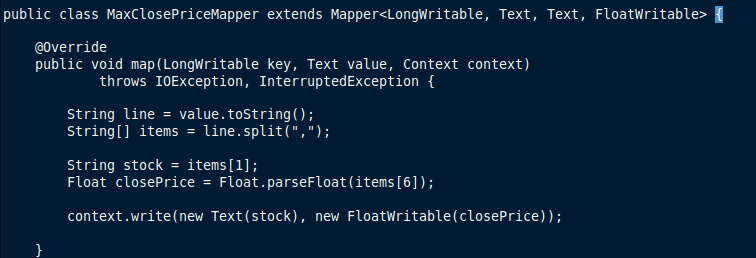
- set the mapper and reducer class

- set the key and value pair types for both mapper and reducer

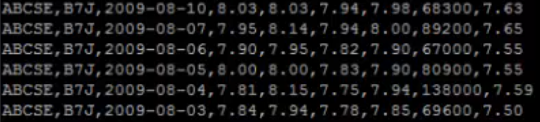
Note: value is a hadoop writable since the data need to be transfered over network as a byte stream (serialization in java), to make the serilization fst over hadoop a new type is introduced.

- job is submitted and wait for completion

**Mapper class:**



This class is excuted by mapper for each input record.



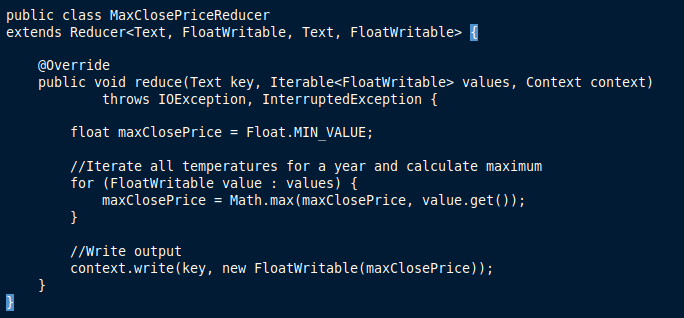
For the stock exchnge problem, it split the input line with “,”

Then get the stock name and the closng price from each record

Create key value pair as stock name as key and close price as value.

Context object is set with key value pair.hadoop wil take over from here.

**Reduce Class:**

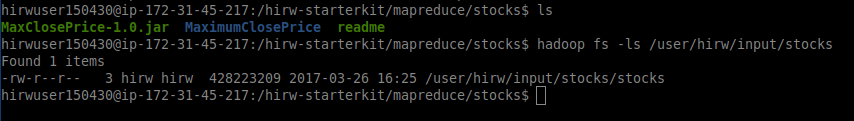


The reduce clas will get the key and values pair, using this the max closing price of each stock name can be calculated

Then creates the final key value pair for eahc key with maximum close price

**Execution in hadoop:**

1. Need compile the code and buil a jar file
2. Copy the jar to hadoop cluster using scp or winscp
3. The input file for stock is present in HDFS



1. Excute the jar file using hadoop

hirwuser150430@ip-172-31-45-217:/hirw-starterkit/mapreduce/stocks$ hadoop jar /hirw-starterkit/mapreduce/stocks/MaxClosePrice-1.0.jar com.hirw.maxcloseprice.MaxClosePrice /user/hirw/input/stocks output/mapreduce/stocks

20/08/04 18:40:11 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-45-216.ec2.internal/172.31.45.216:8032

20/08/04 18:40:12 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

20/08/04 18:40:12 INFO input.FileInputFormat: Total input paths to process : 1

20/08/04 18:40:12 INFO mapreduce.JobSubmitter: number of splits:4

20/08/04 18:40:12 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1592756896525\_1013

20/08/04 18:40:12 INFO impl.YarnClientImpl: Submitted application application\_1592756896525\_1013

20/08/04 18:40:12 INFO mapreduce.Job: The url to track the job: http://ec2-54-92-244-237.compute-1.amazonaws.com:8088/proxy/application\_1592756896525\_1013/

20/08/04 18:40:12 INFO mapreduce.Job: Running job: job\_1592756896525\_1013

20/08/04 18:40:19 INFO mapreduce.Job: Job job\_1592756896525\_1013 running in uber mode : false

20/08/04 18:40:19 INFO mapreduce.Job: map 0% reduce 0%

20/08/04 18:40:31 INFO mapreduce.Job: map 25% reduce 0%

20/08/04 18:40:36 INFO mapreduce.Job: map 75% reduce 0%

20/08/04 18:40:38 INFO mapreduce.Job: map 100% reduce 0%

20/08/04 18:40:39 INFO mapreduce.Job: map 100% reduce 100%

20/08/04 18:40:39 INFO mapreduce.Job: Job job\_1592756896525\_1013 completed successfully

20/08/04 18:40:39 INFO mapreduce.Job: Counters: 54

File System Counters

FILE: Number of bytes read=8908

FILE: Number of bytes written=638174

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=428236041

HDFS: Number of bytes written=8357

HDFS: Number of read operations=15

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Killed map tasks=1

Launched map tasks=4

Launched reduce tasks=1

Data-local map tasks=4

Total time spent by all maps in occupied slots (ms)=208852

Total time spent by all reduces in occupied slots (ms)=22600

Total time spent by all map tasks (ms)=52213

Total time spent by all reduce tasks (ms)=5650

Total vcore-milliseconds taken by all map tasks=52213

Total vcore-milliseconds taken by all reduce tasks=5650

Total megabyte-milliseconds taken by all map tasks=53466112

Total megabyte-milliseconds taken by all reduce tasks=5785600

Map-Reduce Framework

Map input records=7461349

Map output records=7461349

Map output bytes=59248591

Map output materialized bytes=8926

Input split bytes=544

Combine input records=7461349

Combine output records=895

Reduce input groups=836

Reduce shuffle bytes=8926

Reduce input records=895

Reduce output records=836

Spilled Records=1790

Shuffled Maps =4

Failed Shuffles=0

Merged Map outputs=4

GC time elapsed (ms)=1422

CPU time spent (ms)=23750

Physical memory (bytes) snapshot=2448711680

Virtual memory (bytes) snapshot=6884646912

Total committed heap usage (bytes)=2143813632

Peak Map Physical memory (bytes)=613994496

Peak Map Virtual memory (bytes)=1383821312

Peak Reduce Physical memory (bytes)=194990080

Peak Reduce Virtual memory (bytes)=1380237312

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

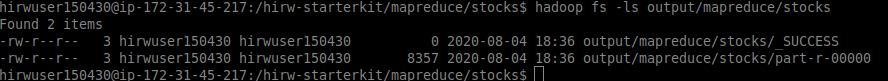
Bytes Read=428235497

File Output Format Counters

Bytes Written=8357

hirwuser150430@ip-172-31-45-217:/hirw-starterkit/mapreduce/stocks$

1. After the job is executed the output can be found in hdfs directory



1. Can view the output of file in HDFS

Each stock name has a maximum closing price

