Hadoop Setup

Hadoop Setup

- Setting up Hadoop development environment can be a tedious and time-consuming task.
- In order to better utilize time, we will be using a pre-configured Ubuntu Operating System (OS) running on a Virtual Machine (VM)

Hadoop Setup

- We uses a virtualization product called Virtual Box
- It is feature-rich, high performance, and free
- To set up Hadoop Training Virtual Machine (VM) in Virtual Box.
  - Virtual Box can run in most modern Operating Systems; therefore, you need to have a laptop/desktop
  - The minimal specifications for your machine are 4G RAM, Core2 Duo Processor, and 20G of empty hard drive space

Hadoop Setup

1. Download Virtual Box

   Download the latest version of Virtual Box for your Operating System: https://www.virtualbox.org/wiki/Downloads

2. Install Virtual Box

   Double click on the Virtual Box Installer and follow installation instructions. When the installation is complete, launch the application.

3. Download Hadoop Training Ubuntu Image

   Download Hadoop Training Virtual Machine (VM). The file is ~1.5G so it may take few minutes to download.

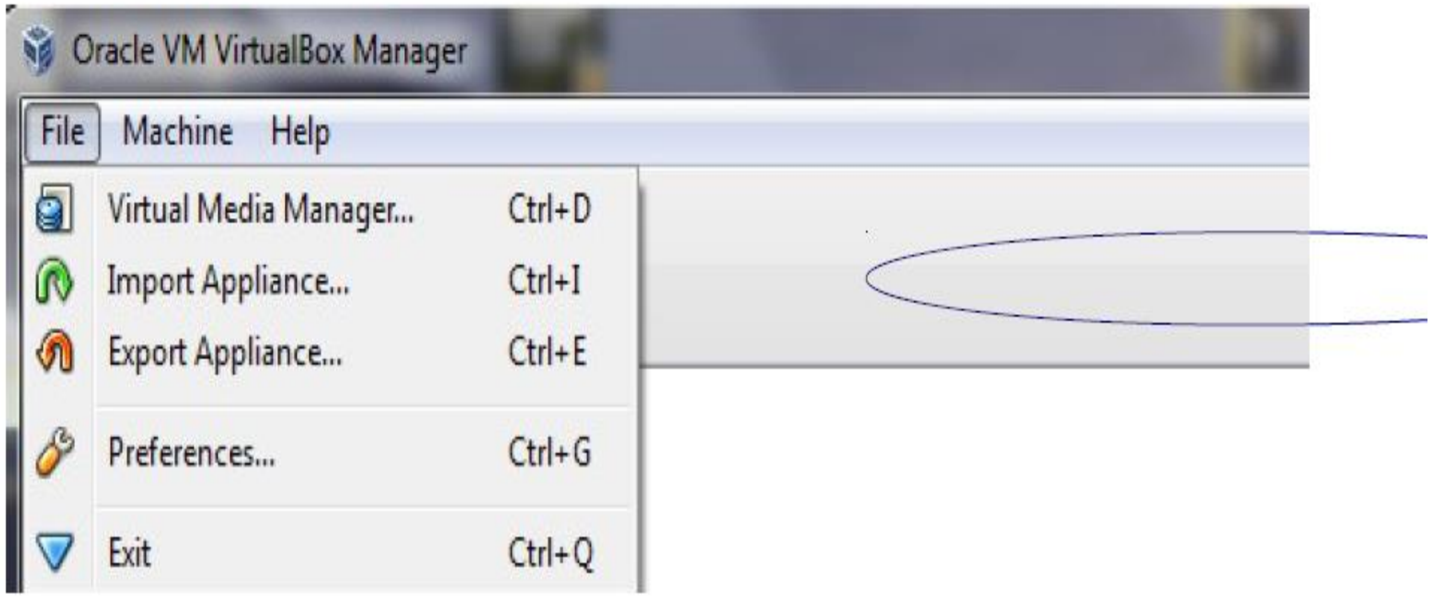
   After all, you are downloading Ubuntu Operating System with Hadoop installed.

https://www.dropbox.com/s/sbyr0baonjwf62v/HadoopTraining_v1.0.ova

Hadoop Setup

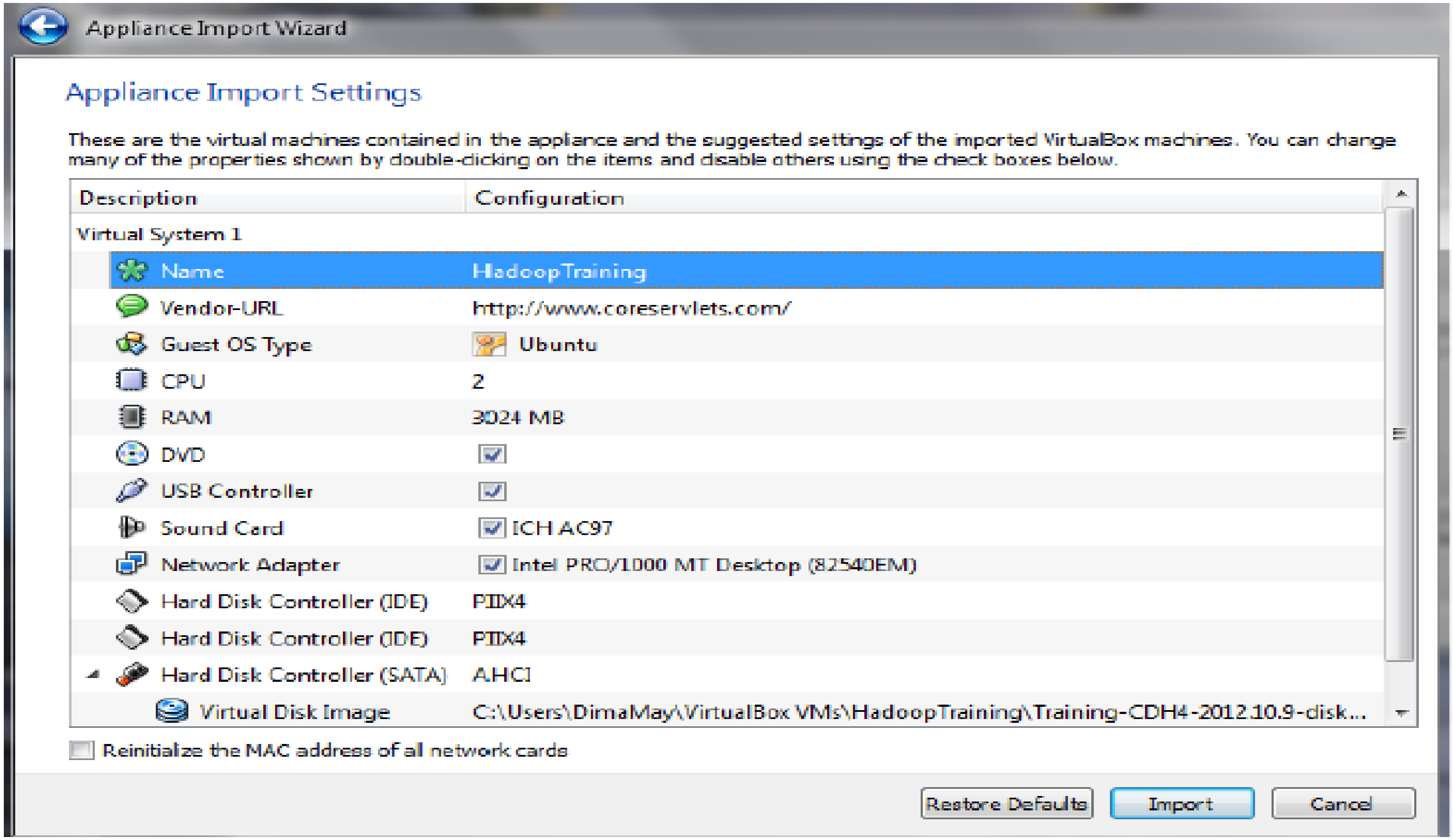
Import *HadoopTraining.ova* into your Virtual Box installation.

(1) File → Import Appliance → Choose → Browse and select *HadoopTraining.ova*


Oracle VM VirtualBox Manager
File
Machine
Help
Virtual Media Manager... Ctrl+D
Import Appliance... Ctrl+I
Export Appliance... Ctrl+E
Preferences... Ctrl+G
Exit Ctrl+Q

Hadoop Setup

(2) Click Next and "Application Import Settings" dialog will pop-up



Appliance Import Wizard

Appliance Import Settings

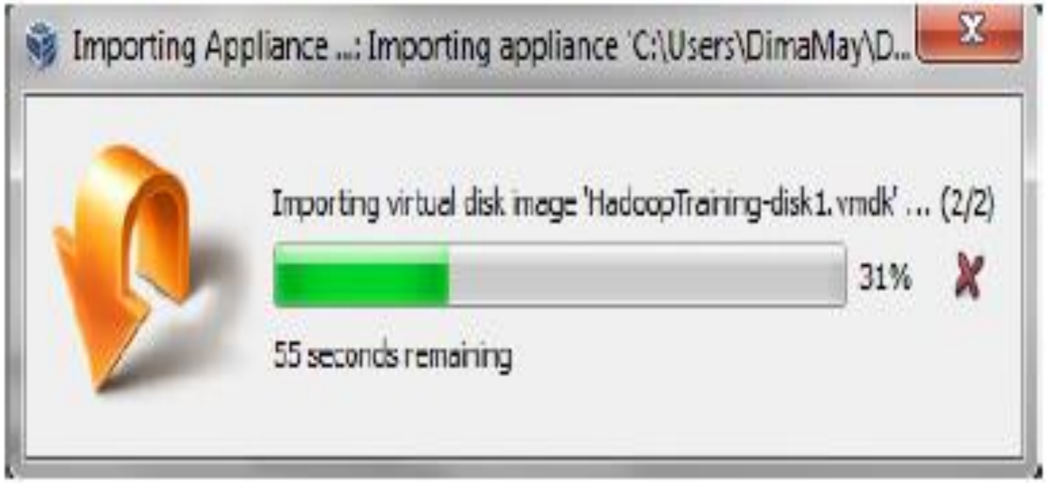
These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Description	Configuration
Virtual System 1	
Name	HadoopTraining
Vendor-URL	http://www.coreservlets.com/
Guest OS Type	Ubuntu
CPU	2
RAM	3024 MB
DVD	✓
USB Controller	✓
Sound Card	✓ ICH AC97
Network Adapter	✓ Intel PRO/1000 MT Desktop (82540EM)
Hard Disk Controller (IDE)	PIIX4
Hard Disk Controller (IDE)	PIIX4
Hard Disk Controller (SATA)	AHCI
Virtual Disk Image	C:\Users\DimaMay\VirtualBox VMs\HadoopTraining\Training-CDH4-2012.10.9-disk...

☐ Reinitialize the MAC address of all network cards

Hadoop Setup

(3) By default there is 3G of RAM and 1 CPU. If you think you have more to spare then it's a good idea to increase RAM and number of CPUs (it will make your development experience faster and more pleasant). To adjust these settings, double click on the number and it will become editable. In addition, you can always change these settings later. Click **Import** button when you are ready. It may take a few minutes.

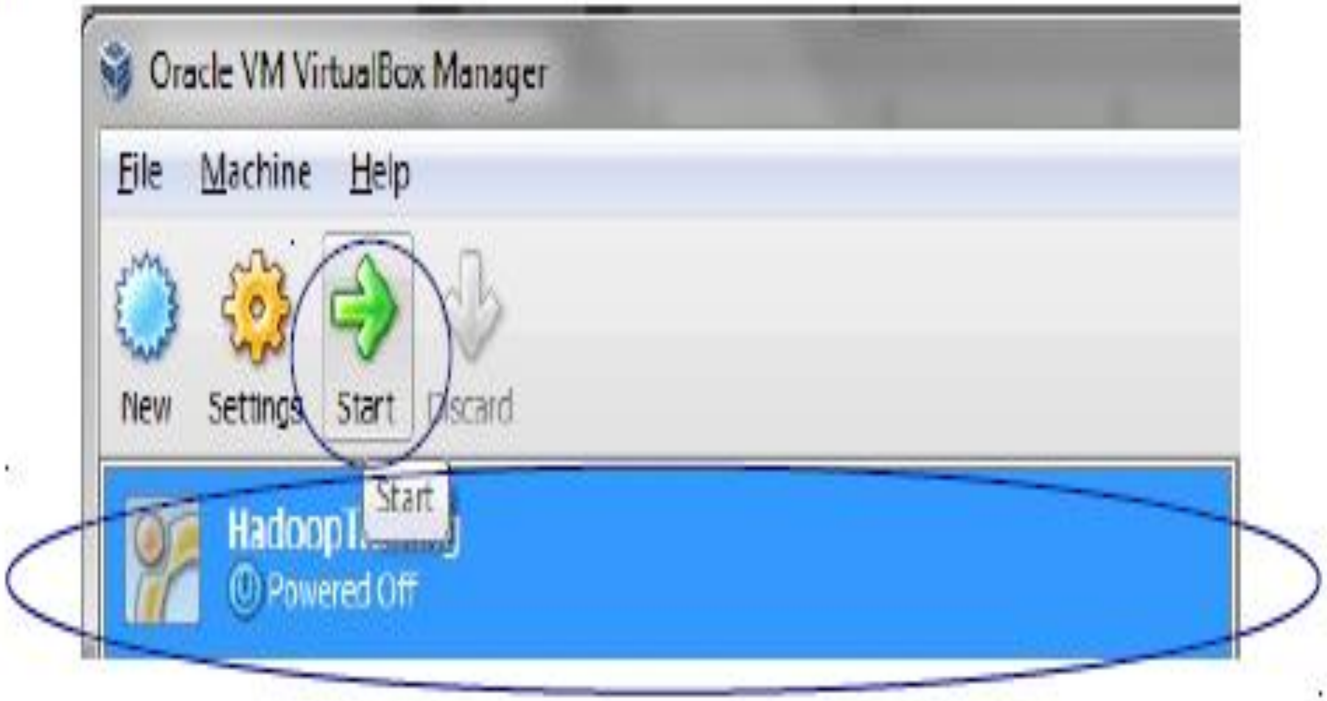

Importing Appliance ...: Importing appliance 'C:\Users\DimaMay\D...
31%

Importing virtual disk image 'HadoopTraining-disk1.vmdk' ... (2/2)

55 seconds remaining

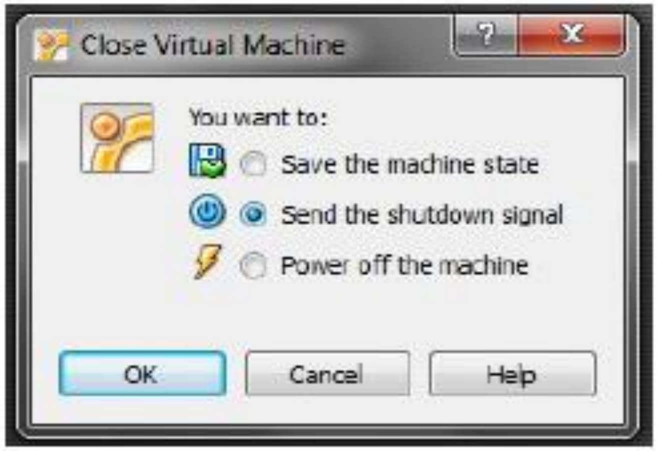
Hadoop Setup

(4) Start Hadoop Training Virtual Machine by selecting **HadoopTraining** and clicking the **Start** button.



Hadoop Setup

(5) Congratulations! You are now running guest OS Ubuntu in the Virtual Box. When closing you VM, a choice will be presented:



You want to:

- ( ) Save the machine state
- [✓ Send the shutdown signal]
- ( ) Power off the machine

"Save the machine state" will preserve all of your programs and windows as is. The next time VM is started it will be in the previously saved state. "Send the shutdown signal" will notify guest OS to initiate shutdown. "Power off the machine" is kind of like pressing the power button.

Logging in

- **Username: hadoop**
- **Password: hadoop**

VM excercise

Perform

1. Start Virtual Machine
2. Open Command Line Terminal
3. Start ALL Hadoop installed products
4. Stop ALL Hadoop installed products
5. Verify there are any leftover Java processes

Solution

1. Open Virtual Box, select training VM and click start
2. Within VM double click on Terminal icon
3. In the Command Line Terminal type *startCDH.sh*
4. In the Command Line Terminal type stopCDH.sh
5. jps

Hadoop Shell Commands

Perform

1. Start HDFS and verify that it's running
2. Create a new directory /exercise1 on HDFS
3. Upload a.txt file into HDFS under /exercise1directory
4. View the content of the /exercise1directory
5. Print the first 25 lines to the screen from a.txt on HDFS
6. Copy a file in HDFS with another name in HDFS
7. Copy a.txt to local file system and name it a_copy.txt
8. Delete a.txt from HDFS
9. Delete the /exercise1 directory from HDFS
10. Take a second to look at other available shell options

Hadoop Commands

1. Start HDFS and verify that it's running

```
$ startCDH.sh
```

2. Create a new directory /exercise1 on HDFS

```
$ hdfs dfs -mkdir /exercise
```

3. Upload a.txt file into HDFS under /exercise1 directory

```
$ hdfs dfs -put  exercise1/ a.txt
```

Hadoop Commands

4. View the content of the /exercise1directory

```
$ hdfs dfs -ls /exercise1/
```

5. Print the first 2 lines to the screen from a.txt on HDFS

```
$ hdfs dfs -cat /ANU/a1.txt | head -n 2
```

6. Copy a file in HDFS with another name in HDFS

```
$ hdfs dfs -cp /ANU/a1.txt /ANU/a1_hdfsCopy.txt
```

Hadoop Commands

7. Copy a.txt to local file system and name it a_copy.txt

```
$ hdfs dfs -get /exercise1/a.txt a_copy.txt
```

8. Delete a.txt from HDFS

```
$ hdfs dfs -rm /exercise1/a.txt
```

9. Delete the /exercise1 directory from HDFS

```
$ hdfs dfs -rm -r /exercise1
```

10. Take a second to look at other available shell option

```
$ hdfs dfs -help
```

Thank You

Solving the census problem

- Calculating the population of all cities in a big state like California is not an easy task for any one person
- divide the state by city and make individuals incharge of each city to calculate the population of each city he/she is incharge of
- illustration purpose, lets focus on only 3 cities – San Fransisco, San JOSE, and LA

Solving the census problem

- Person 1 will be incharge of SFO
- Person 2 will be incharge of San Jose
- Person 3 will be incharge of LA
- Incharge is responsible for finding the population of the assigned city

Solving the census problem

- Task of each incharge
  - Go to a home,
  - knock on the door and
  - when someone answers the door.
  - ask how many people live in the home and note it down.

Solving the census problem

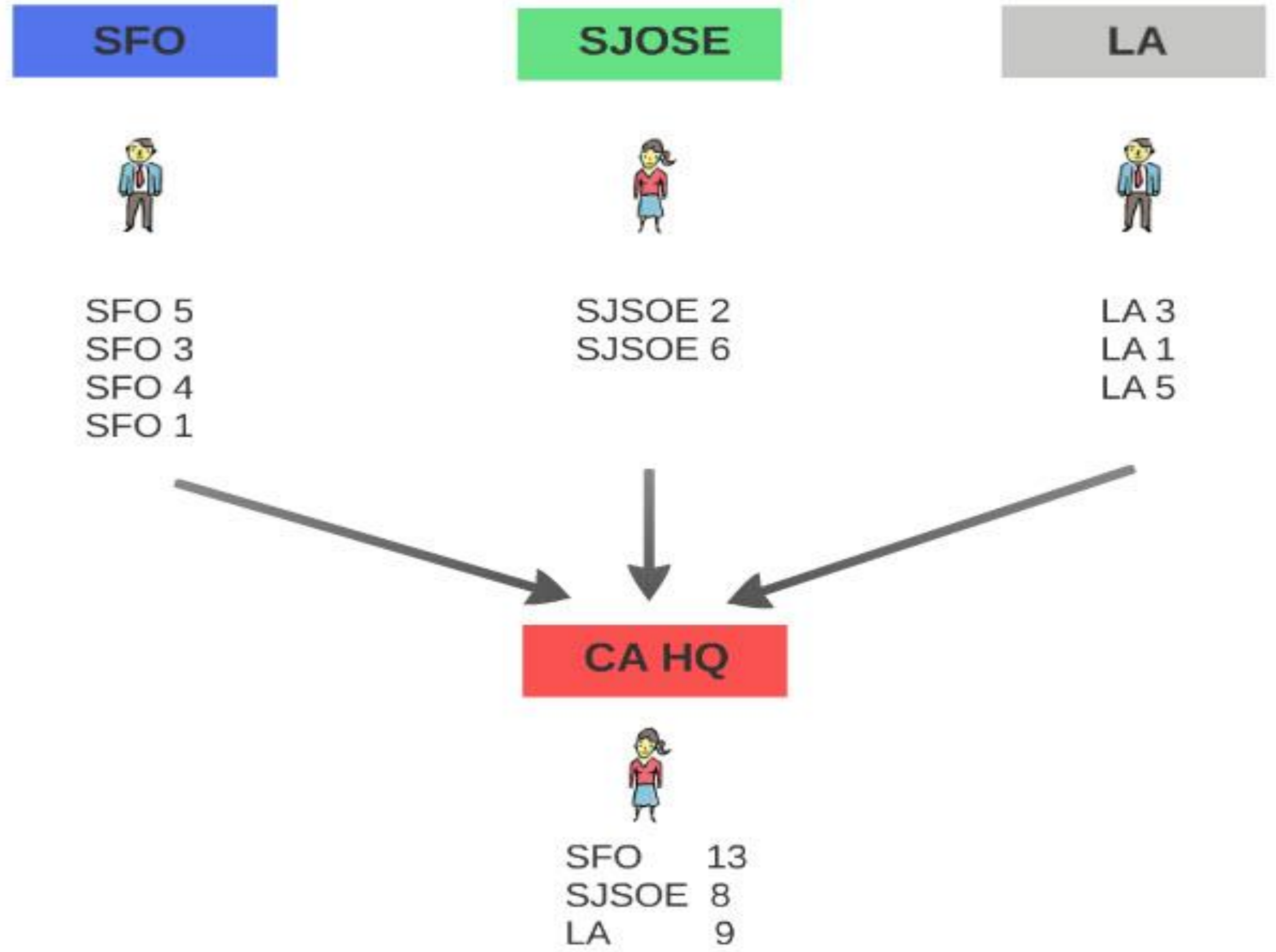
- Task of each incharge

  - You are instructing each one to note down the city they are responsible for and the number of people live in the home

  - Then the incharge has to go to the next home and repeat the same process until he covers all homes in the assigned city.

Solving the census problem

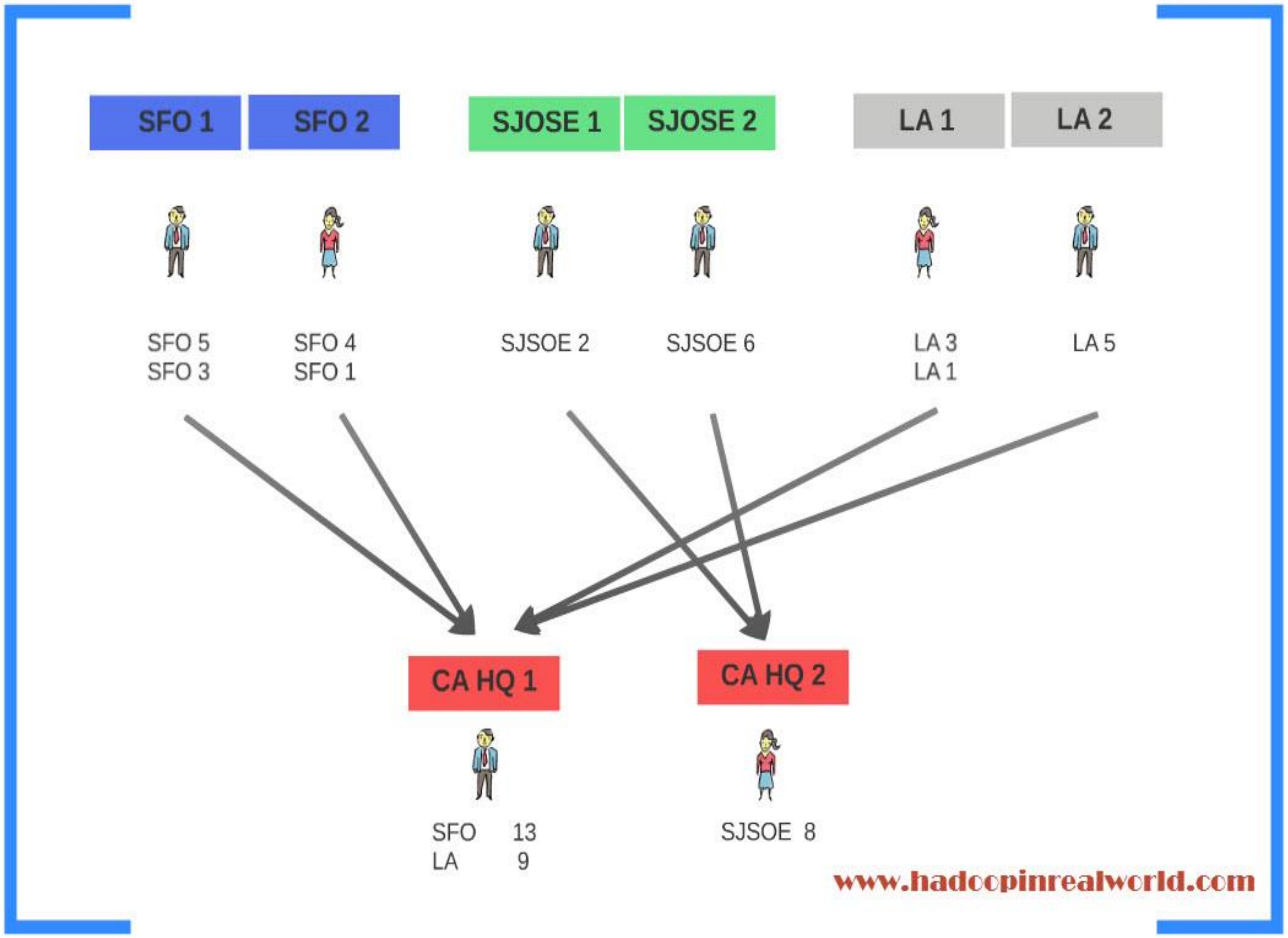
- Task of each incharge
  - An incharge who covers SFO
  - he goes to the first home there are 5 people in the home so he will note down "SFO 5".
  - 3 people are living in the 2nd home and he will note down "SFO 3".
  - Then the incharge has to go to the next home and repeat the same process until he covers all homes in the assigned city.


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

- When each person is done with their assigned city you will ask them to submit their result to the state’s head quarters.
- You will have a person in the head quarters to receive the results from all cities and aggregate them by city to come up with population of each city for the entire state.
- Assume 4 months to complete this job

- Next year around you are asked to do the same job again
- you have all the resources you want
- this time you have 2 months to finish the task.
- What would you do?

- So you would simply double the number of people to perform the tasks.
- You will divide SFO in to 2 divisions and add one person to each division.
- You will do the same thing for San Jose and same for LA.
- You can also do the same thing at the head quarters.
- Let’s divide the head quarters in to two. CA HQ1 and CA HQ2 and one person to each division.


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

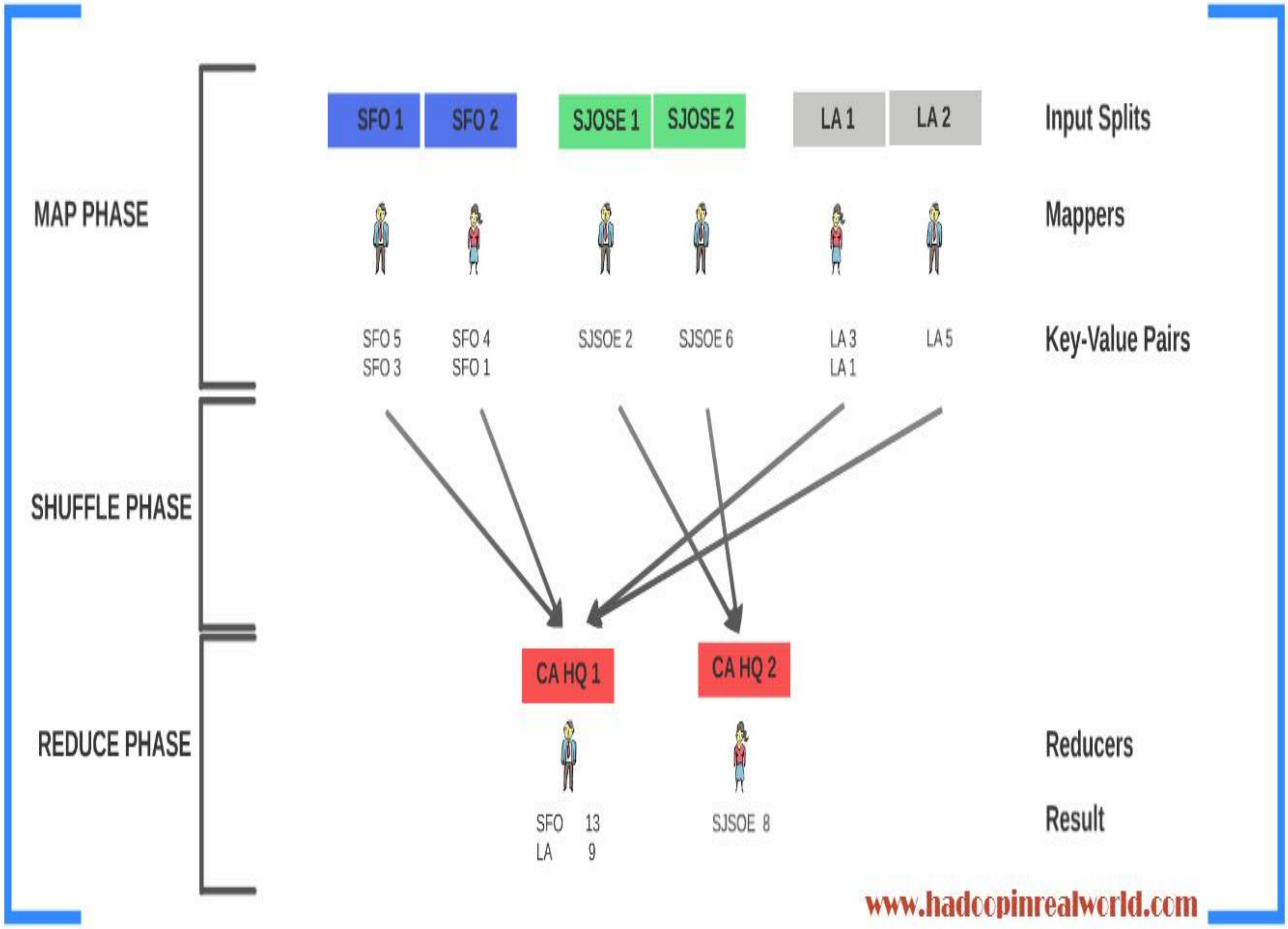
- With twice as much people you can finish the task in half the time.
- But there is one small problem. You want the census takers for SFO – SFO1 and SFO2 send their results to either CA HQ1 or CA HQ2.
- You don't want SFO1 sending their results to CA HQ1 and
- SFO2 sending their results to CA HQ2 because this would result in population count for SFO divided between Head quarters 1 and 2.
- This is not ideal because we want the consolidated population count by city, not partial counts.
- So what can we do?

- Simple, we can instruct census takers in SFO1 and 2 to send to thier result to either headquarters 1 or headquarters 2.
- Similarly we should instruct census takers for San Jose and LA; they should send to either HQ 1 or HQ 2.
- Problem solved!

- You try with this model and again you did it.
- You were able to complete the census calculation in 2 months.
- If next year, if you were asked to do the same thing in 1 month, you know exactly what to do.
- You can simply double the resources and apply your model and it will work like a charm.
- You now have a good enough model. not only the model works but it can also scale.
- That’s it. the Model you have is called Map Reduce.

Introducing MapReduce & its’ phases

- MapReduce is a programming model for distributed computing.
- It is not a programming language
- It is a model which you can use to process huge datasets in a distributed fashion.
- Now lets look at the phases involved in MapReduce.


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

Map Phase

- The phase where individuals calculate the population of their assigned city or part of city is called the Map Phase
- The individual person involved in the actual calculation is called the Mapper
- The city or part of the city he is working with is known as the Input Split
- The output from each Mapper is a Key Value pair.

Reduce Phase

- The phase where you aggregate the intermediate results from each city or Mappers in the Head Quarters is called the reduce phase
- The individuals who work in the head quarters are known as the reducers because they reduce or consolidate the output from many different mappers
- Each reducer will produce a result set.

Shuffle Phase

- The phase in which the values from many different mappers are copied or transfered to the reducers is known as the shuffle phase.
- The shuffle phase comes in between Map and Reduce phase.
- Map Phase, Shuffle Phase and Reduce Phase are the 3 phases of Map-Reduce.

Real World Example

- we will take a real example and walk through the process involved working with Map-Reduce.
- We have a dataset with information about several fictitious stocks symbol.
- Each line in the dataset we have information about a stock, stock symbol for a day… information like the opening price, closing price, low, high, volume and adjusted closing price for a given day.

Stock Dataset

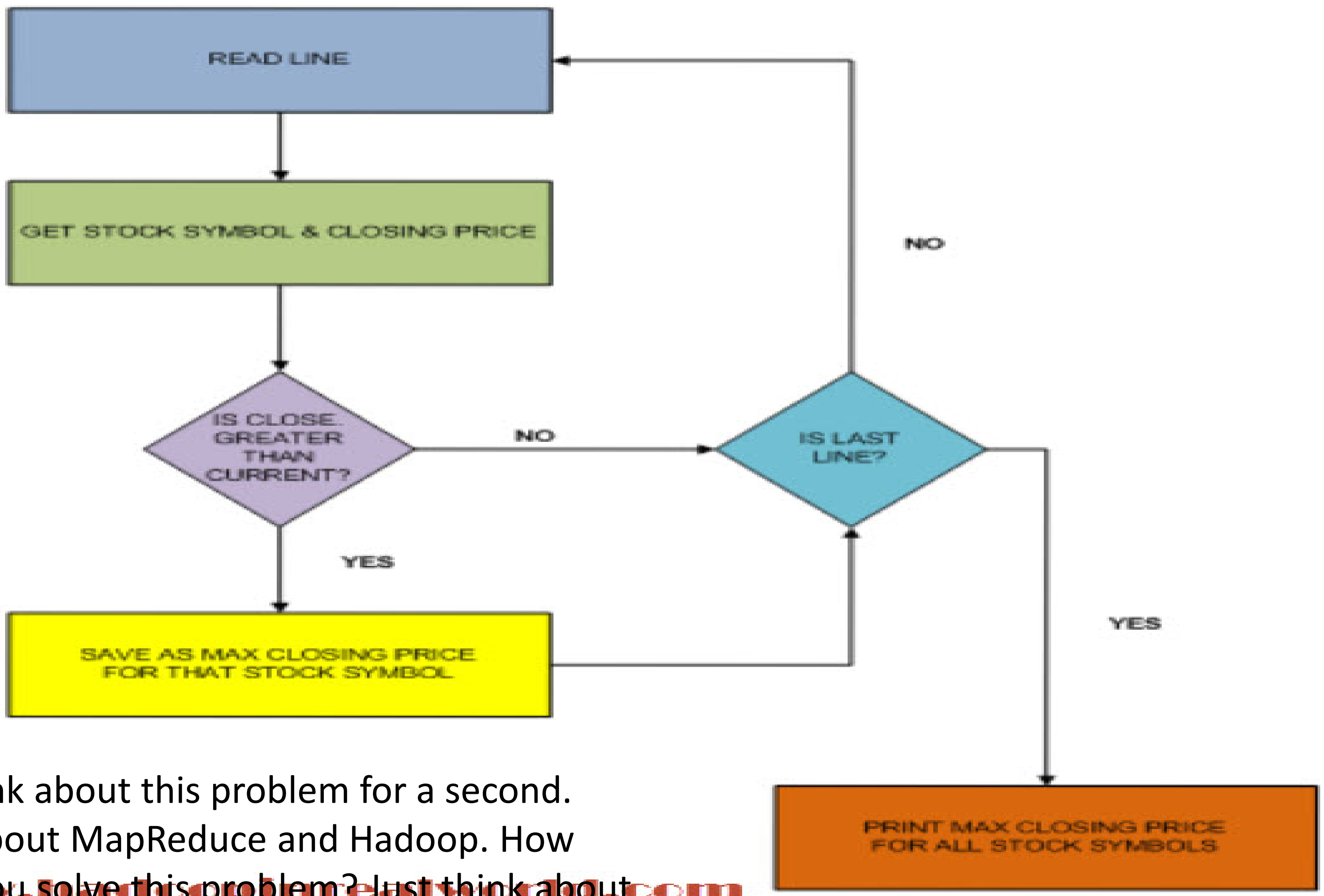
- ABCSE,B7J,2017-01-20,8.93,9.09,8.93,9.09,119300,9.09
- ABCSE,B7J,2017-01-19,8.99,9.04,8.91,9.02,132300,9.02
- ABCSE,B7J,2017-01-15,8.98,9.00,8.90,8.99,78000,8.99
- ABCSE,B7J,2017-01-14,9.03,9.03,8.93,9.01,142700,9.01
- ABCSE,B7J,2017-01-13,8.96,9.00,8.92,8.98,101600,8.98
- ABCSE,B7J, 2017-01-12,8.91,8.95,8.86,8.91,70600,8.91
- ABCSE,B7J, 2017-01-11,8.99,9.00,8.95,8.97,159500,8.97
- ABCSE,B7J, 2017-01-08,8.90,8.95,8.88,8.94,153100,8.94
- ABCSE,B7J, 2017-01-07,8.91,8.94,8.88,8.94,120700,8.94

Lets pick a record in this dataset, in this record we have inform
ation for symbol. B7J. for date 2017-01-20 and we have the op
ening price, low, high, close price, volume etc.

Stocks Problem

- Assume the size of this data is slightly over 400 MB.
- Not too big but good enough for our experimentation.
- Now lets talk about the problem we would like to solve with this dataset.
- For every stock symbol in the dataset we would like to find out its maximum closing price across several days. Simple use case right?

Simple flow chart


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

Now think about this problem for a second. forget about MapReduce and Hadoop. How would you solve this problem? Just think about the the algorithm.

Algorithm

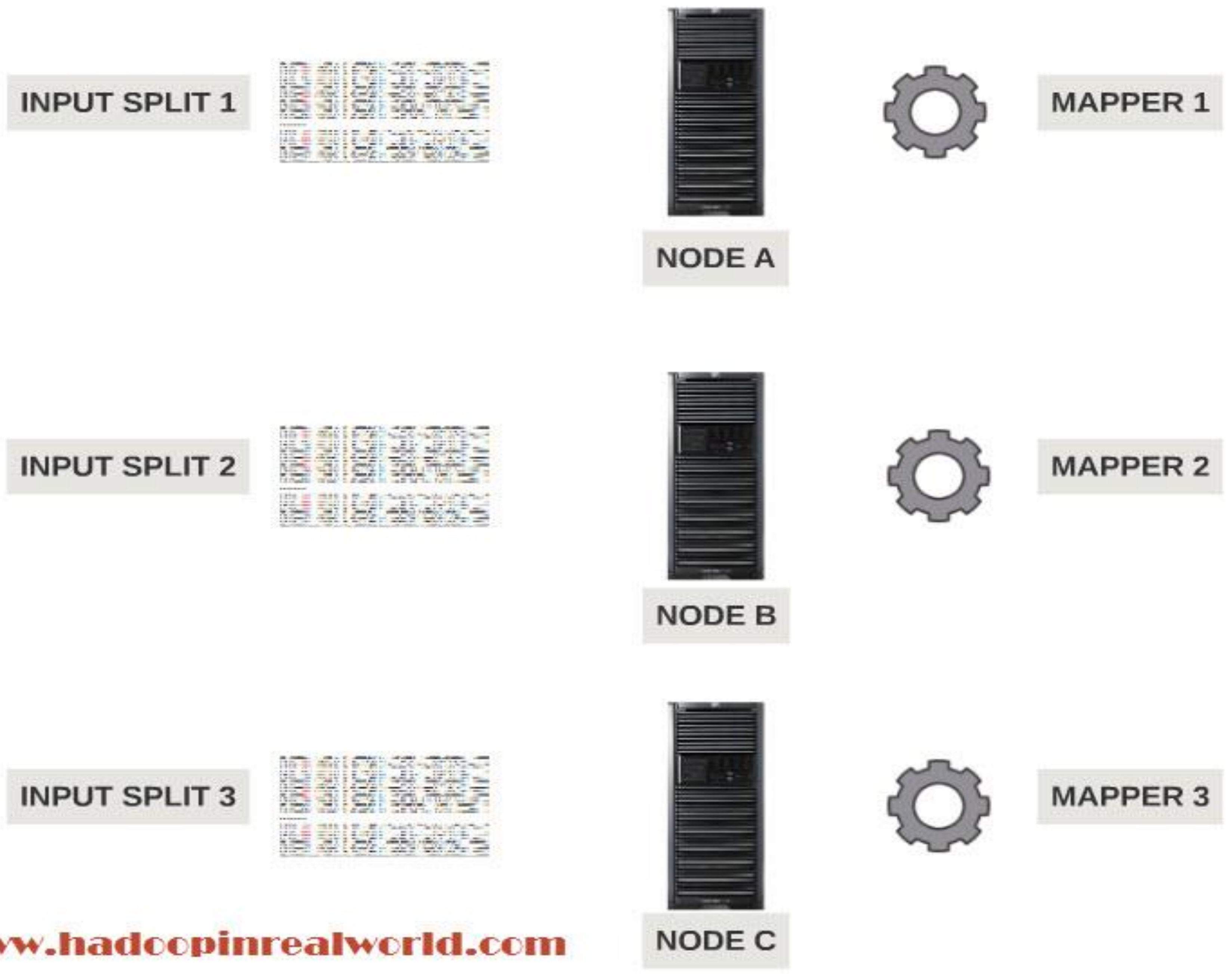
- We will read a line
- get the symbol and closing price from the line.
- Then check, is this closing price greater than the closing price we have for the symbol.
- If not go process the next line.
- If it is the greater than the closing price, save the closing price as the maximum closing price for that symbol
- move on to the next record in the dataset. If you reached the end of the file, print the results.

Problem with the approach

- Problem is that there is no parallelization.
- If you have a huge dataset you will have extremely long computation time which is not ideal.
- Now lets see how we have work out the same problem in the Map-Reduce world.

- We go over each phase and see the technical details involved in the
- Map Phase, Reduce Phase and the Shuffle phase.
- Lets first talk about the Map Phase.
- The central idea behind MapReduce is distributed processing

- So the first thing is divide the dataset in to chunks
- you have separate process working on the dataset on every chunk of data.
- Lets assign some technical Jargons now the chunks are called *input splits* and the process working on the chunks are called *Mappers*.
- Every mapper execute the same set of code and for each record they process in the *input split*, they could O/P a key value pair.



INPUT SPLIT 1
NODE A
MAPPER 1
INPUT SPLIT 2
NODE B
MAPPER 2
INPUT SPLIT 3
NODE C
MAPPER 3

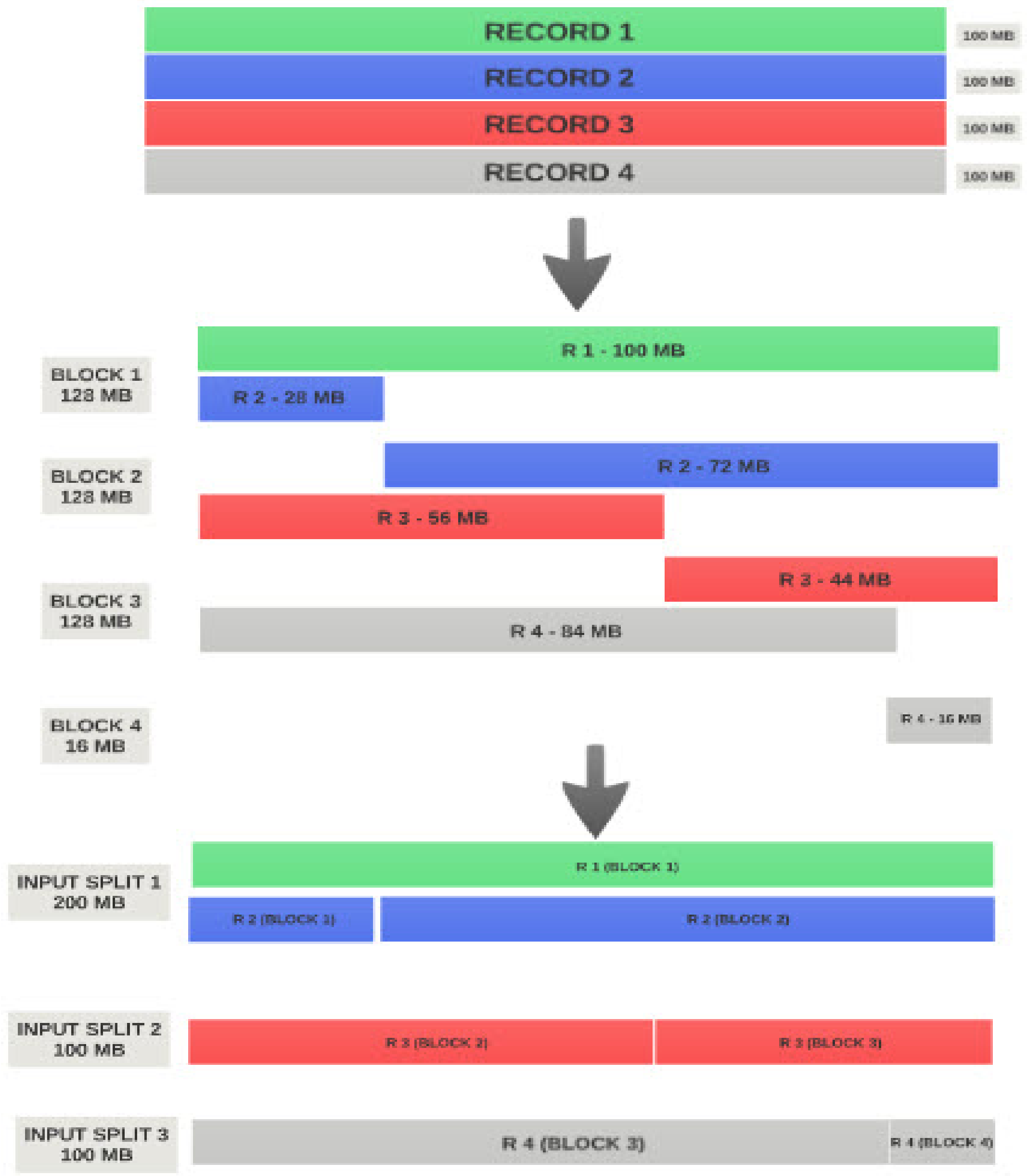
Block vs. InputSplit

- First what is an Input Split?
  How many of you think Input Split is same as the Block??
- InputSplit is not same as the block.
- A block is a hard division of data at the block size.
- If the block size is 128 MB. Each block for the dataset will be 128 MB except for the last block which could be less than the block size if the file size is not entirely divisible by the block size.

Block vs. InputSplit

- Since block is a hard cut at the block size block can end even before a record ends.
- Your block size is 128 MB.
- Each record in your file is about 100 Mb. Yes, imagine huge records.
- So the first record will perfectly fit in the block no problem since the record size 100 MB is well with in the block size which is 128 MB.
- However the 2nd record can not fit in the block. So the record number 2 will start in block 1 and will end in block 2.

BLOCKS vs. INPUT SPLIT



Block vs. InputSplit

- If you assign a mapper to a block 1, in this case, the Mapper can not process Record 2 because block 1 does not have the complete record 2.
- That is exactly the problem *Inputsplit* solves.
- In this case Input Split 1 will have both record 1 and record 2.
- Input Split 2 does not start with Record 2 since Record 2 is already included in the Input Split 1.
- So Input Split 2 will have only record 3.
- As you can see record 3 is divided between Block 2 and 3. Input Split is not physical chunks of Data.
- It is a Java class with pointers to start and end locations with in blocks.

Block vs. InputSplit

- So when Mapper tries to read the data it clearly knows where to start and where to end.
- The start location of an input split can start in a block and end in another block.
- That is why we have a concept of Input Split.
- Input Split respect logical record boundary.
- During MapReduce execution Hadoop scans through the blocks and create InputSplits which respects the record boundaries.
- Now that we understand the input to each Mapper lets talk about the Mapper itself.

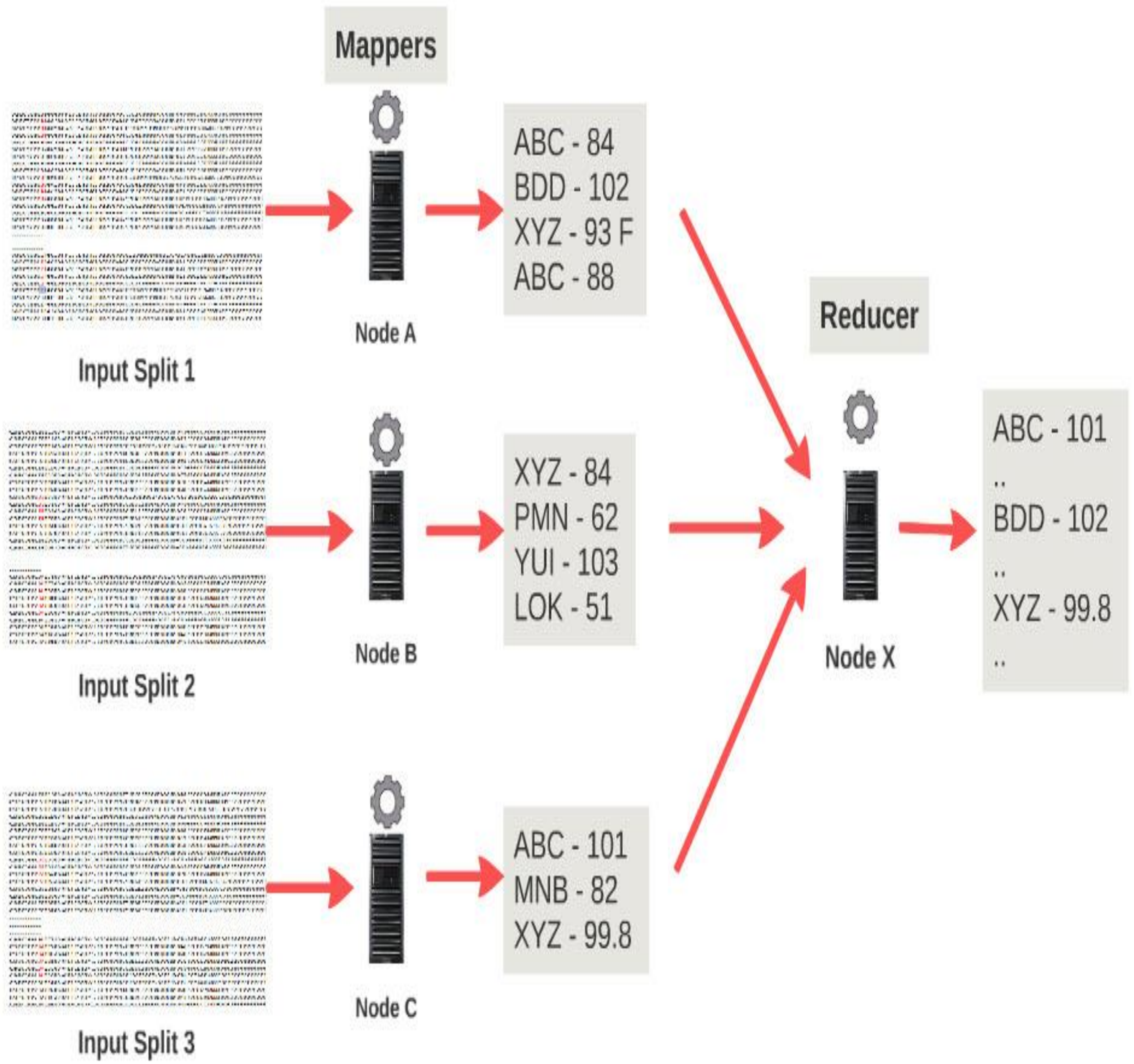
Map Phase

- Mapper is a Java program which is invoked by the Hadoop framework once for every record in the Input Split
- If you have 100 records in a Input Split the Mapper processing the split will be executed 100 times.
- How many mappers do you think Hadoop will create to process a dataset?

Map-Phase

- That is entirely dependent on the number of Input Splits.
- If there are 10 Input Splits there will be 10 mappers.
- If there are 100 input splits there will be 100 mappers.
- So a mapper is invoked for once every single record in the input split.
- The output of the Mapper should be a key value pair

MAP PHASE

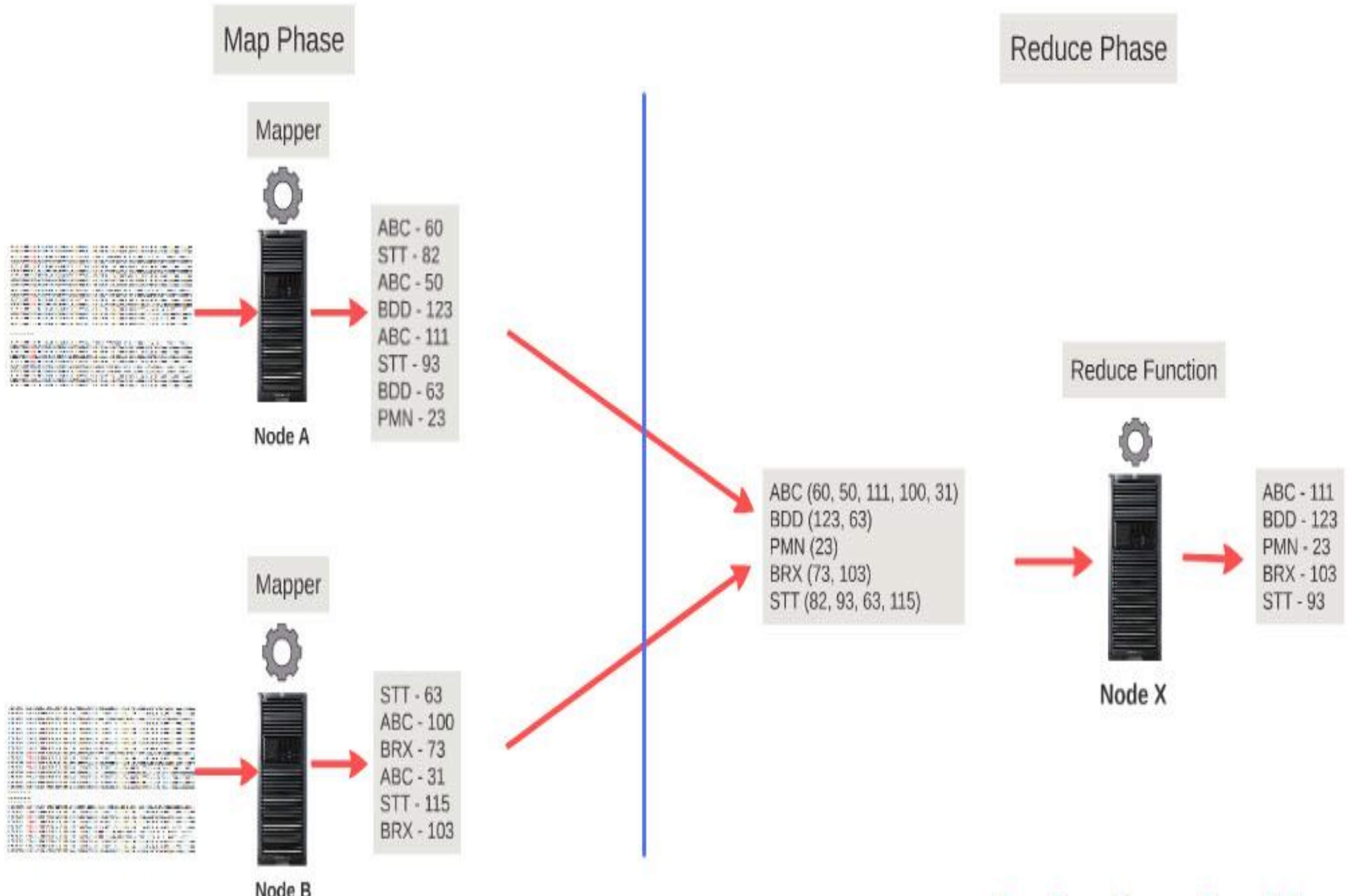

Mappers
Reducer
Input Split 1
Input Split 2
Input Split 3
Node A
Node B
Node C
Node X
ABC - 84
BDD - 102
XYZ - 93 F
ABC - 88
XYZ - 84
PMN - 62
YUI - 103
LOK - 51
ABC - 101
MNB - 82
XYZ - 99.8
ABC - 101
..
BDD - 102
..
XYZ - 99.8
..

- In our sample stock dataset, every line is a record for us
- we need to parse the record to get the stock symbol and the closing price.
- So stock symbol and closing price becomes the output.
- Symbol is the key and closing price is the value.
- How do you decide what should be the key and what should be symbol?

Reduce Phase

- The reducers work on the output of the Mappers.
- The ouput of the individual Mappers are grouped by Key in our case the stock symbol and list of values passed to the reducer.
- Reduce will receive a Key and a List of values for input.
- The keys will be grouped and lets say our dataset has stock information about 10 stocks symbols and 100 records for each symbol.
- That is 1000 records.
- So you will get 1000 key value pairs from all the Mappers combined

REDUCE PHASE



Shuffle Phase

- Lets say in our MapReduce job we decided to use 3 reducers.
- Lets say you have data for Apple in the stock dataset and we have 10 input splits to process, which means we will need 10 mappers.
- we can have records for apple in more than one input split and let’s assume the records for apple is spread out in all 10 input splits.

Shuffle Phase

- This means that each Mapper will produce Key Value pairs for Apple in its output.
- When you have more than one reducer you don’t want the Key Value pairs for Apple to spread out between the 3 reducers.
- That will be bad for our use case because we won’t be able to calculate a consolidated Max closing price for apple.

Shuffle Phase

- we want each key or symbol in our case to be assigned to a reducer and stick to it.
- Each mapper will process all the records in the input splits and will output a key value pair for each record.
- If you look at the output we have symbol for key and closing price as value. We have ABC 60, STT 82. Same with all the other mappers too.
- You may also note that the symbols in Mapper 1 can also found in Mapper2 . Look at the symbol STT for instance.
- Then in the shuffle phase with in each mapper the key value pairs will be assigned to a partition.
- With in each partition the key value pairs will be sorted by key.
- As you can see in the slide the ouput key value pairs are nicely sorted by key.

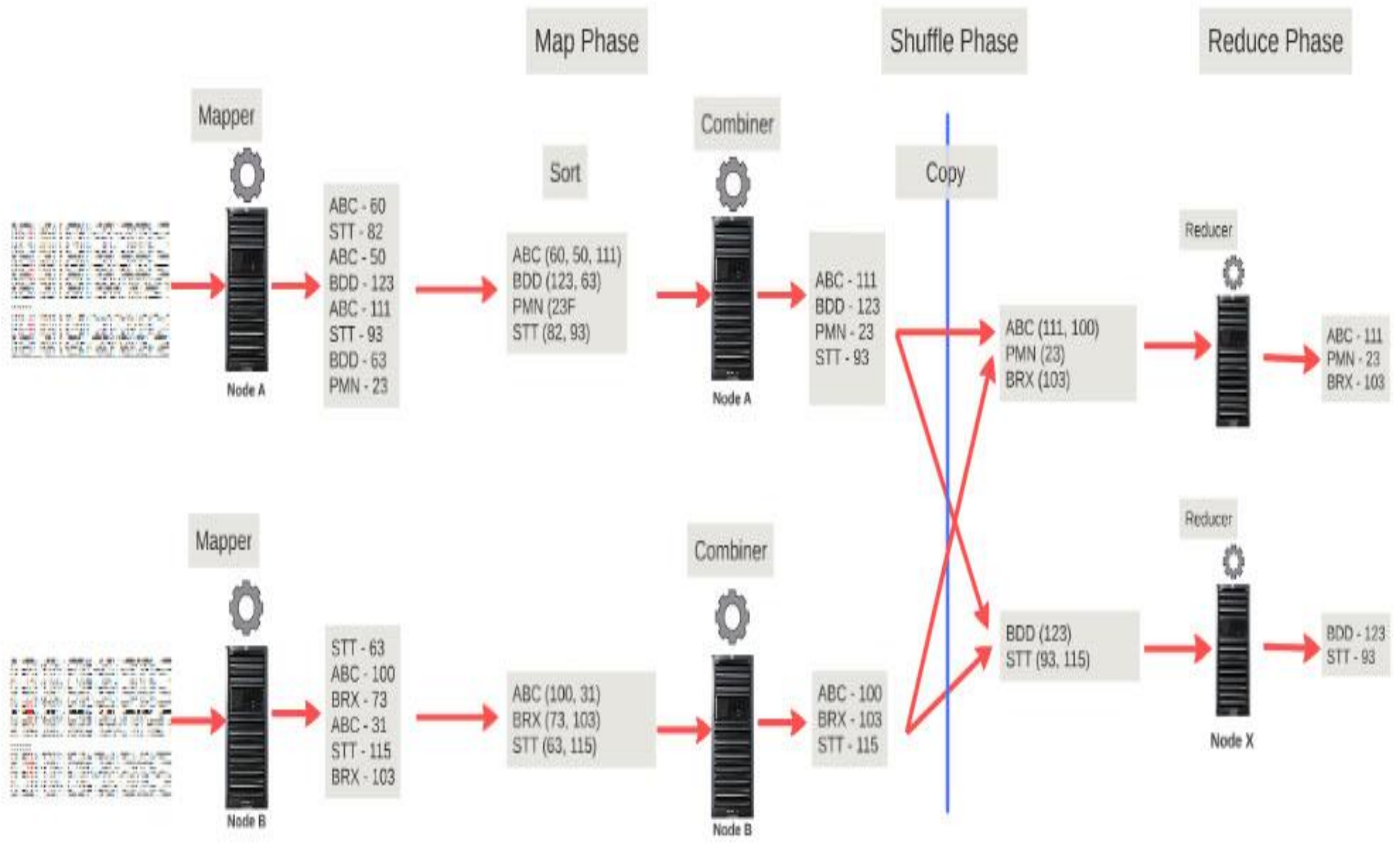
Shuffle Phase

- Then the key value pairs from each mapper will be copied over to the reduce phase to the appropriate reducers.
- At each reducer the the key value pairs coming from different mappers will be merged maintainng the sort order.
- There are 2 things to note in the fig. First is the the symbols are unique to each reducer. Second is the sort order by key is maintained.
- Then the job for the reducer is simple. The reducer 1 will run 3 times once for each symbol and reducer 2 will run 2 times once for each symbol. Each run will output the symbol and its maximum closing price.
- That is the end to end process.

Combiner

- We could have an optional Combiner at the Map Phase.
- Combiners can be used to reduce the amount of data that is sent to the reduce phase.
- In our example there is no reason to send all the closing prices for each symbol from each mapper. For eg. In mapper 1 we have 3 records for ABC so we have 3 closing prices for ABC – 60, 50 111.
- Since we are calculating the maximum closing price we don’t have the send the key value pairs with closing price 50 and 60. Since they are less than 111.
- So all we need is to send the key value pair with closing price 111 for symbol ABC.

COMBINER (OPTIONAL)



Combiner

- Combiner is like mini reducer that is executed at the Map Phase.
- Combiners can be very helpful to reduce the load on the reduce side
- Yes, you are reducing the amount of data that are being sent to the reducers, there by increasing the performance.
- Combiner is optional.

- We are going to write 3 programs – Mapper, Reducer and a Driver program.
- We know what a Mapper and what is a Reducer but what is a Driver/Runner program ?
- A driver program will provide all the needed information and bring all the needed information together to Submit a MapReduce job.

Driver Code

Job object refers to a MapReduce job and Instantiate a new object

- *JobConf conf = new JobConf(MaxClosePrice.class);*

Give a name to the job

conf.setJobName("stocksmax");

Driver Code

- set the input path where you can find the input dataset

FileInputFormat.setInputPaths(conf,new Path(args[0]));

- the output path where you want your reducers to write the output

FileOutputFormat.setOutputPath(conf,new Path(args[1]));

To specify the format of your input and output dataset.

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

InputFormat

InputFormat is responsible 3 main tasks

- First, it Validate inputs, meaning make sure the dataset actually exists in the location that you specified.
- Next, Split-up the input file(s) into logical InputSplit(s), each of which is then assigned to an individual Mapper.
- Finally and this is important, InputFormat provides. *RecordReader* implementation to extract input records from the logical InputSplit for processing by the Mapper.

OutputFormat

- Similar to InputFormat, OutputFormat validate output specifications and has RecordWriter implementation to be used to write out the output files of the job.
- Hadoop comes with several OutputFormat implementations, infact for every InputFormat you can find a corresponding OutputFormat and also you can write custom implementations of OutputFormat.

- Next set the Mapper and Reducer classes for this MapReduce job.

conf.setMapperClass(MaxClosePriceMapper.class);

conf.setReducerClass(MaxClosePriceReducer.class);

conf.setCombinerClass(MaxClosePriceReducer.class);

- Setting the combiner class is optional

DriverCode

- Next, set the output key and the value types for both your Mapper and Reducer.
- The key is of type *Text* and value is of type *FloatWritable*

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(FloatWritable.class);
```

- Run the job

```
JobClient.runJob(conf);
```

Writable wrappers in Hadoop

- These O/P types look new doesn’t it? Yes they are new.
- There are writable wrappers in Hadoop for all major Java primitive types.
- For example, the Writable implemenation for

  int is IntWritable.

  float is FloatWritable,

  boolean is BooleanWritable

  String it is Text.

But why new datatypes? when we already have well defined datatypes in Java?

- Writables are used when ever there is a need to transfer data between tasks
- That is when the data is given as input and output to and from the mapper
- They are used also when the data is given as input and output to and from the reducer
- Mappers and Reducers distributed in many different nodes and this mean you will have a lot of data being transferred between nodes

Serialization

- Transfer data over the network between nodes the objects must be turned in to byte stream and this process is called serialization.
- Hadoop is designed to process is million and billions of records so there is a lot of data transferred over the network
- Serialization should be fast, compact and effective

- Authors of Hadoop felt the Java’s out of the box serailzation was not that effective in terms of speed and size
- Java serialization writes the class name of each object which is being serialized to the byte stream
- This is to know the object’s type so that we will be able to deserialize the object from the byte stream

- Every subsequent instance of the class should have a reference to the first occurrence of the class name which this clearly takes up space
- This reference result in two problems

  - first one is space and hence it is not compact

  - second problem is the reference handles introduce a problem during sorting records in a serialized stream, since only the first record will have the class name and must be given special care.

- Writables was introduced to make the Serialization fast and compact.
- How though?
- By simply not writing the class name to the stream.
- Then how would you know the type during deserialization?
- The assumption is that the client always knows the type and this is usually true.

- There is one more benefit of using Writables as opposed to regular Java types.
- With standard Java types, when the objects are reconstructed from a byte stream during deserialization process, a new instance for each object has to be created.
- Where as with writables, the same object can be reused which improves processing efficiency and speed.

Mapper Program

- Can you remember when a Mapper gets called ?
- A mapper is called once for every record in your dataset
- dataset has 100 records a Mapper will be called 100 times
- In our dataset each line is a record so the Mapper is called once for each line.
- What is going to be the output of our Mapper?

Mapper Program

- We are finding the maximum closing price of each stock symbol
- This means that we have to group the records by symbol so that we can calculate the maximum closing price by symbol.
- So we will output Stock Symbol as the key and close price as the value for each record
- We now know what is going to be the Map’s input and what is going to be the maps output.

Mapper Program

- We will extend the Mapper class to make our class a mapper

public class MaxClosePriceMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,FloatWritable>

- Add the appropriate import statements
- Look at the signature of the Mapper class, there are 4 parameters listed
- The first 2 parameter dictate the input to the Mapper as key and value

- The third and the fourth parameter tells us the output key and value from the mapper.
- So the input key to the Mapper is of type *LongWritable* and input value is of type *Text.*
- The output key from the Mapper is of type *Text* and the output value from the Mapper is of type *FloatWritable*.

- In the program, we have the Map method.
- *public void map(LongWritable key, Text value, OutputCollector<Text,FloatWritable> output, Reporter reporter)*
- This method will be called for every record that is passed to the Mapper.
- The first two arguments to the Map method varies based on the Input format that defines your dataset.
- The second parameter is the text, which is the actual line from the file.
- The first argument is LongWritable but what does it represent?

- The first argument is the byte offset with in the file of the beginning of the each line
- the second argument which is the actual record from the file.
- Since the record is delimited by comma, we are using the string split method to get all the columns in to the array.

```
String line = value.toString();
String[] items = line.split(",");
```

- We know our dataset very well so we know that the 2nd column is the symbol and the 7th column is the closing price.

String stock = items[1];

Float closePrice = Float.parseFloat(items[6]);

- But how do we submit the output from the Mapper?
- We will use the output collect to collect the output

output.collect(new Text(stock), new FloatWritable(closePrice));

Reducer Program

- From here on, Hadoop will take care of your mapper output.
- The output along with other key value pairs from the mappers will be sorted and partitioned by key.
- which will then be copied to the appropriate reducer.
- then at each reducer the output from several mappers will be merged.
- The values for a key will be group and the input to the reducer will be a key and a list of values for that key.
- The reducer will be called once per key.

Reducer Program

- First start by extending the Reducer class.
- You have to specify 4 type parameters, the first 2 parameters defines the input to the reducer
- The 3rd and 4th defines the output from the reducer

public class MaxClosePriceReducer extends MapReduceBase implements Reducer<Text,FloatWritable,Text,FloatWritable>

- Next, the reduce method
- This will take Text as the key which is nothing but a stock symbol
- Iterator list of float writables which is nothing but the list of closing prices for that particular stock symbol as arguments

```
public void reduce(Text key, Iterator<FloatWritable>
values,OutputCollector<Text,FloatWritable> output,
                Reporter reporter)
```

- Once you have the list of closing prices
- it is easier to calculate the maximum closing price using the while loop

```
float maxClosePrice = Float.MIN_VALUE;
        while (values.hasNext()) {
 maxClosePrice = Math.max(maxClosePrice, values.next().get());
                                   }
```

- Finally once you have the output,simply submit the output by calling the write method on the context. Make sure the type of input arguments to the reduce method and the types that you wrote to the context match the type parameters defined above. Now we have the program ready. it is now time to run package this project as a .jar and execute the job in our Hadoop cluster.

- But how do we submit the output from the Reducer?
- We will use the output collect to collect the output

output.collect(key, new FloatWritable(maxClosePrice));