Map Reduce Programming

By
Aparna.Kumari@nirmauni.ac.in

Contents

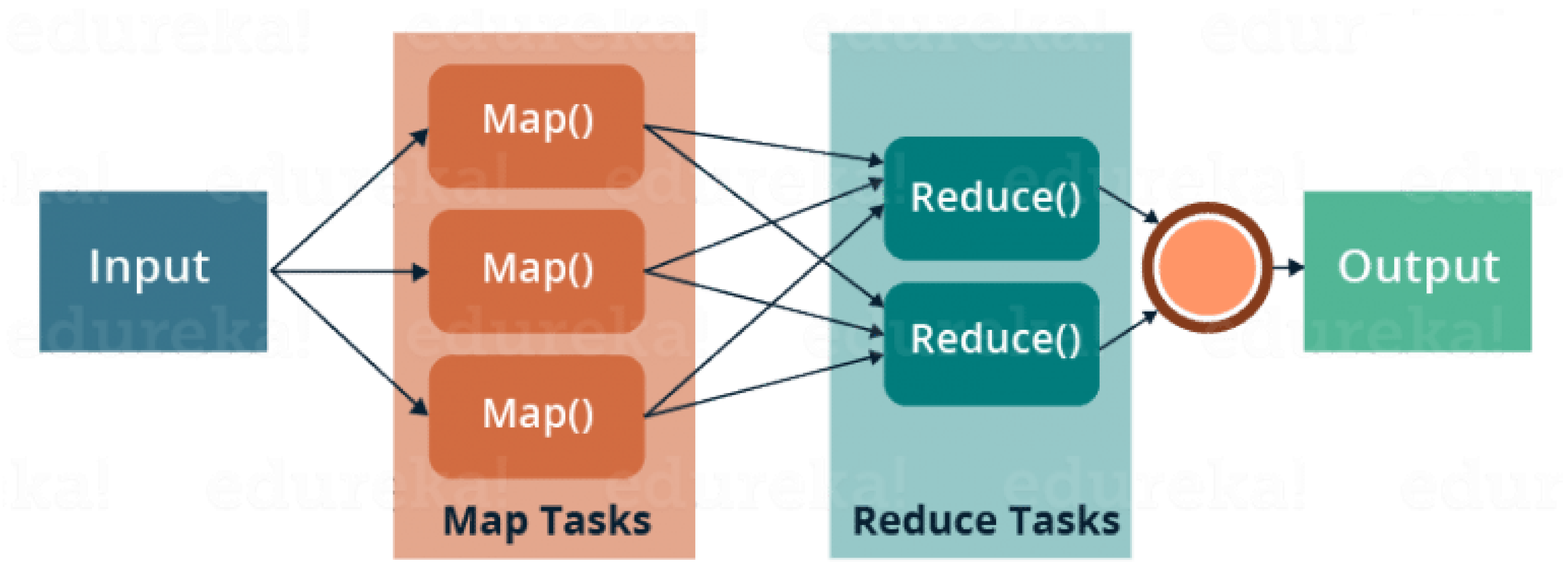
Map Reduce

- MapReduce is a programming model which is associated with the implementation of data processing of extensive data sets.
- Users specify a map function that processes a Key/Value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.
- Many real world tasks are expressible in this model.

Map Reduce

- Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines
- The run-time system takes care of the details of partitioning the input data, scheduling the program’s execution across a set of machines, handling machine failures and managing the required inter-machine communication.
- Typical MapReduce computation processes many terabytes of data on thousands of machines.

What is Map Reduce?



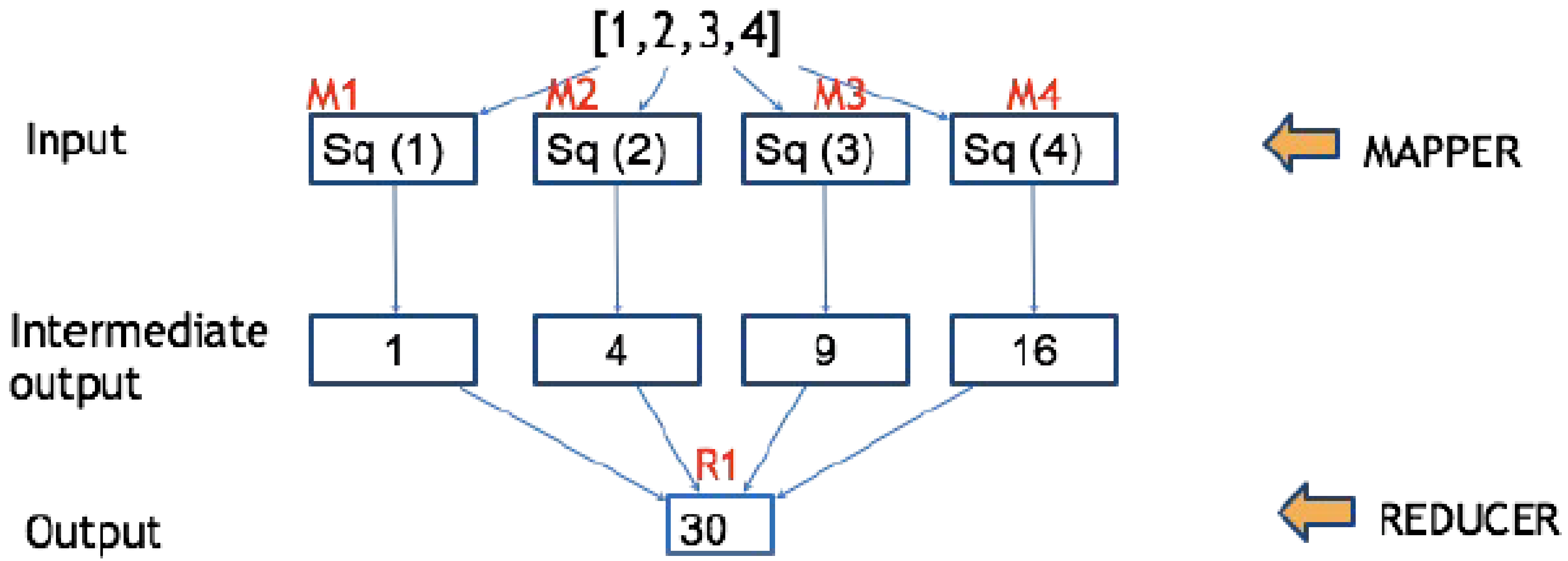
Map Reduce

- MapReduce consists of two distinct tasks – Map and Reduce.
- As the name MapReduce suggests, the reducer phase takes place after the mapper phase has been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.

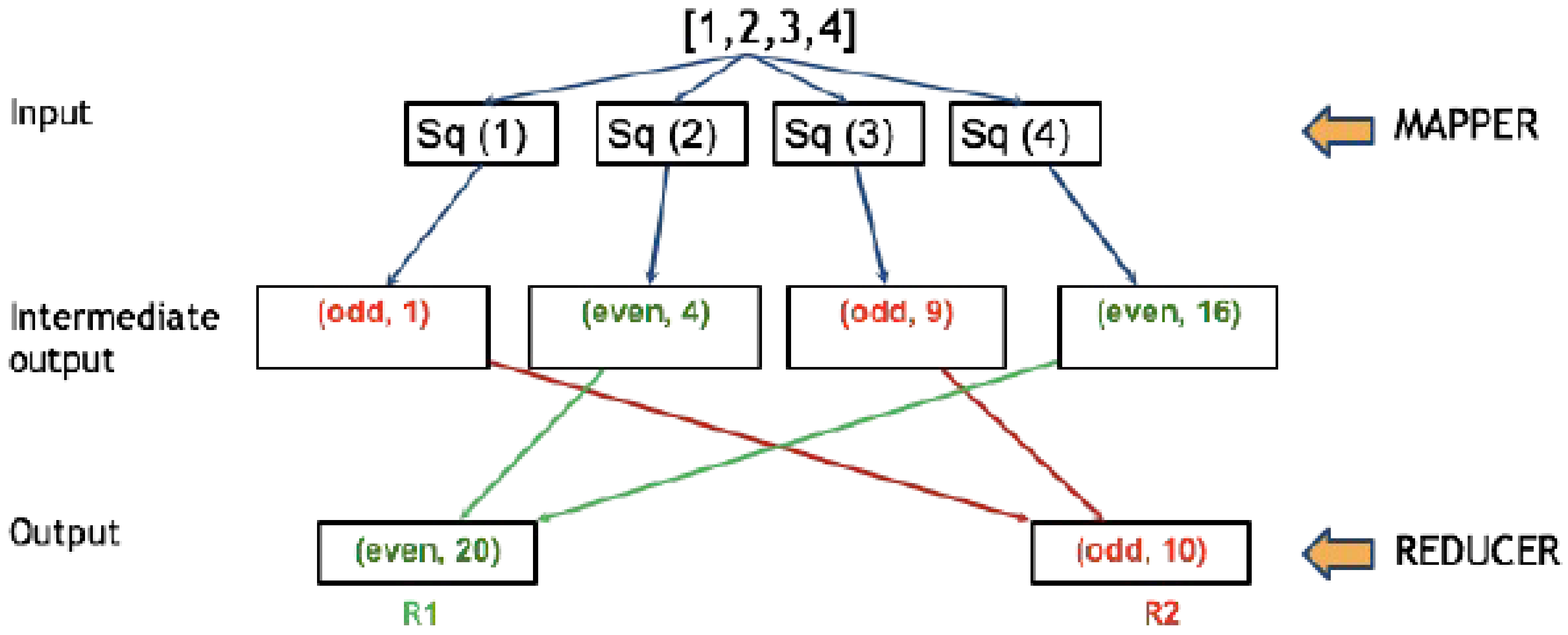
Examples

- Input : Dear, Bear, River, Car, Car, River, Deer, Car and Bear
- Sum of Squares of input: [1,2,3,4]

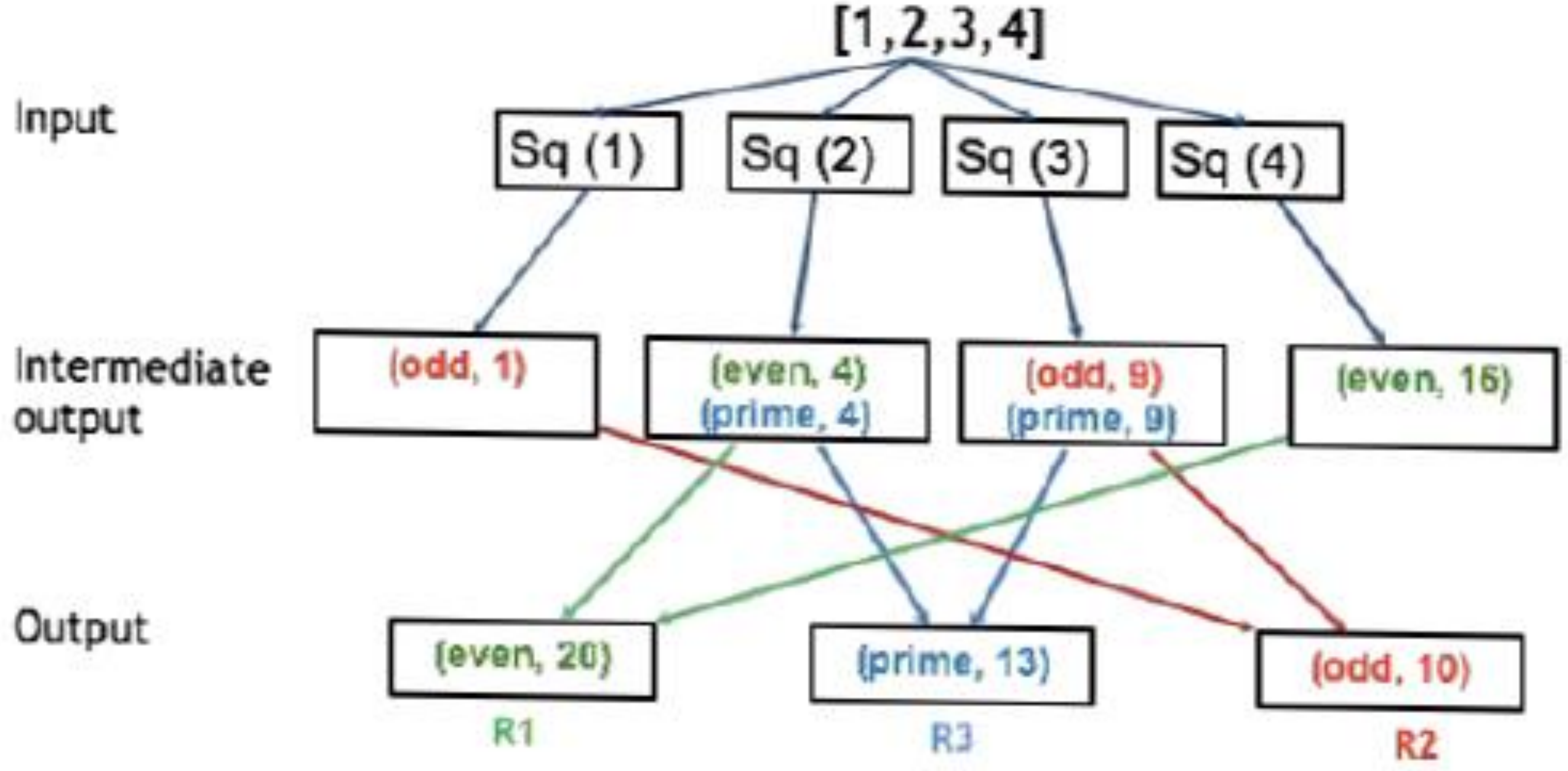
Examples : Sum of Squares



Examples : Sum of Squares of odd and even numbers



Examples : Sum of Squares of prime and non-prime numbers



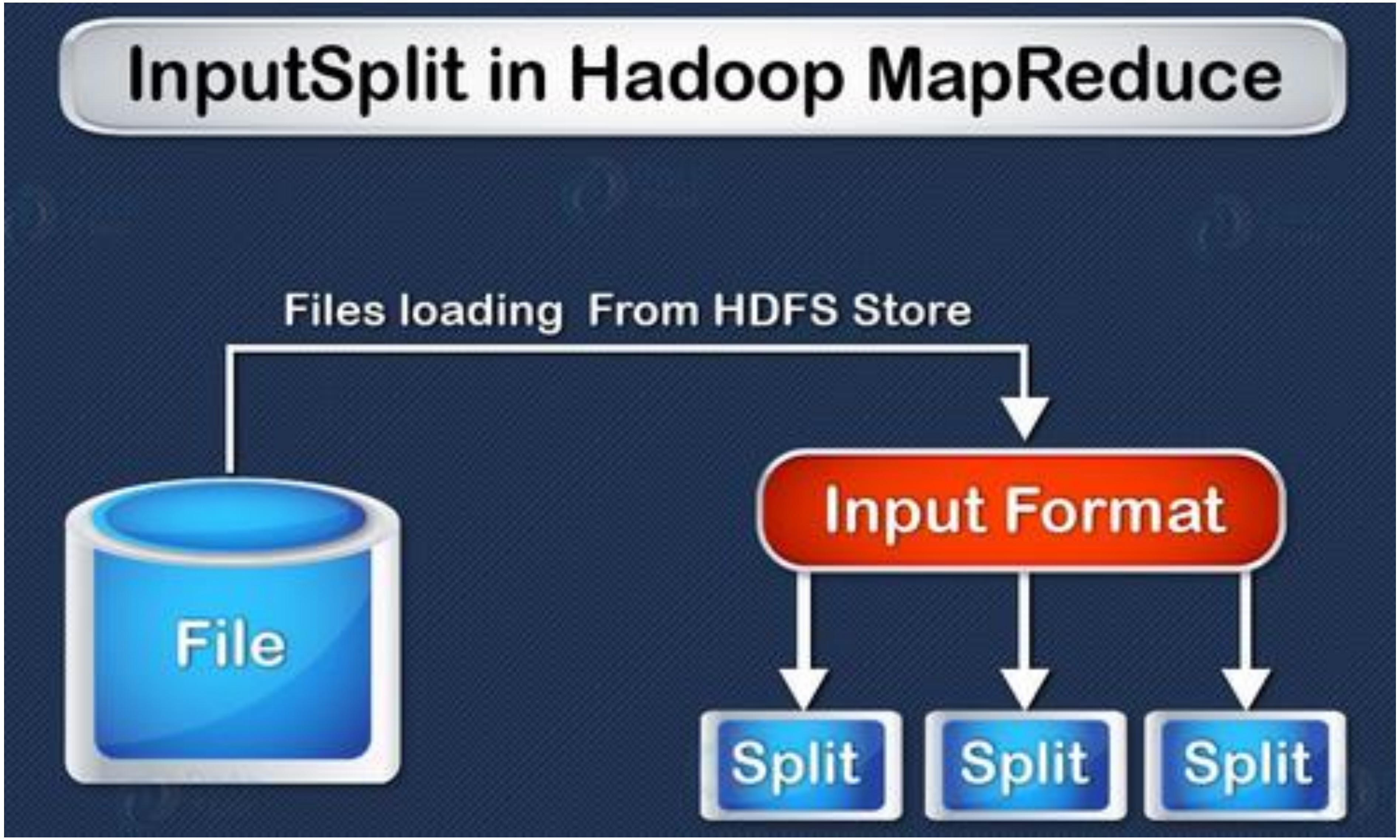
Mapper Class

- The first stage in Data Processing using MapReduce is the **Mapper Class.** Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store and saves this intermediate data into the local disk.
- **Input Split**

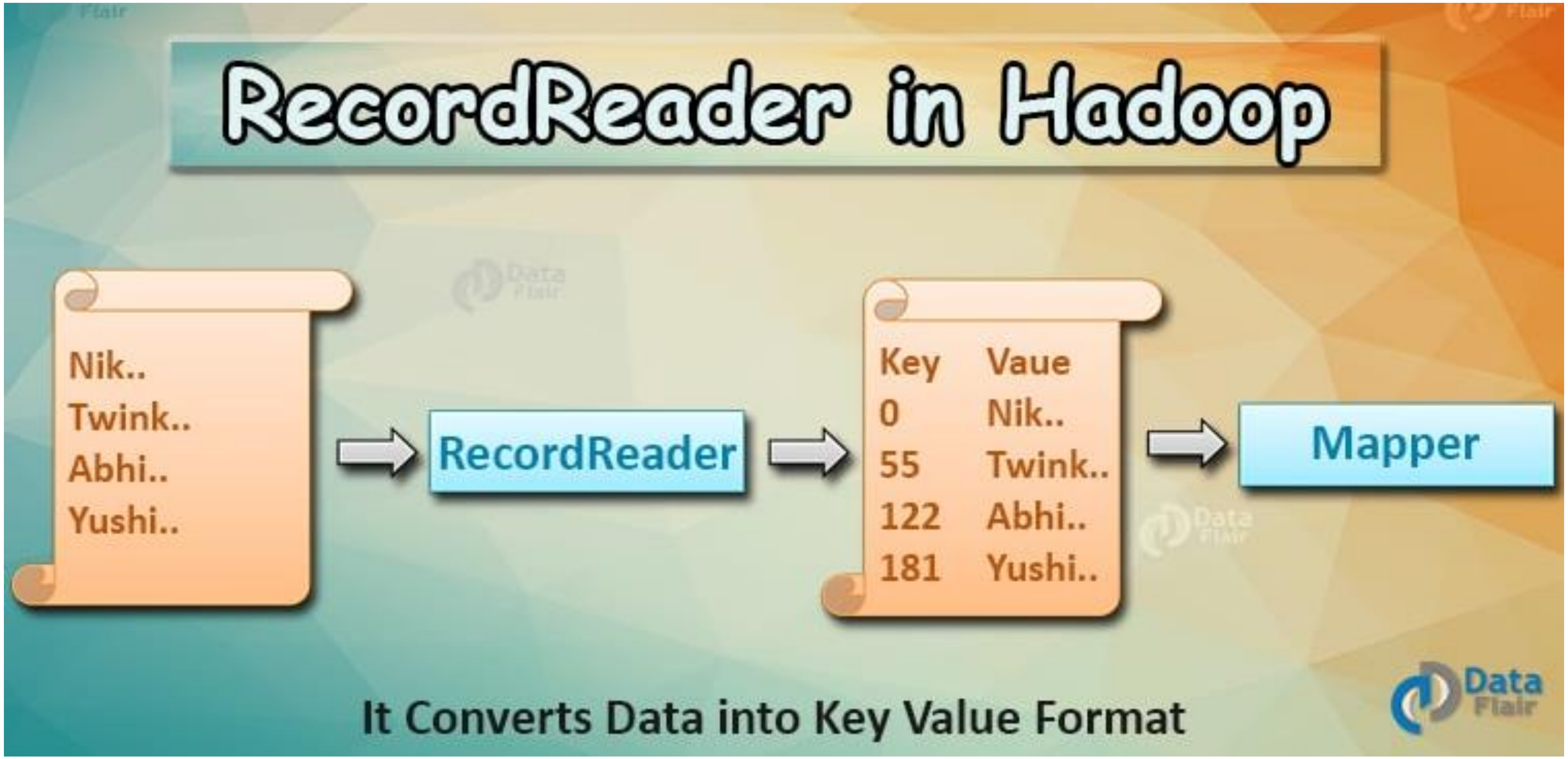
  It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
- **RecordReader**

  It interacts with the Input split and converts the obtained data in the form of **Key-Value** Pairs.

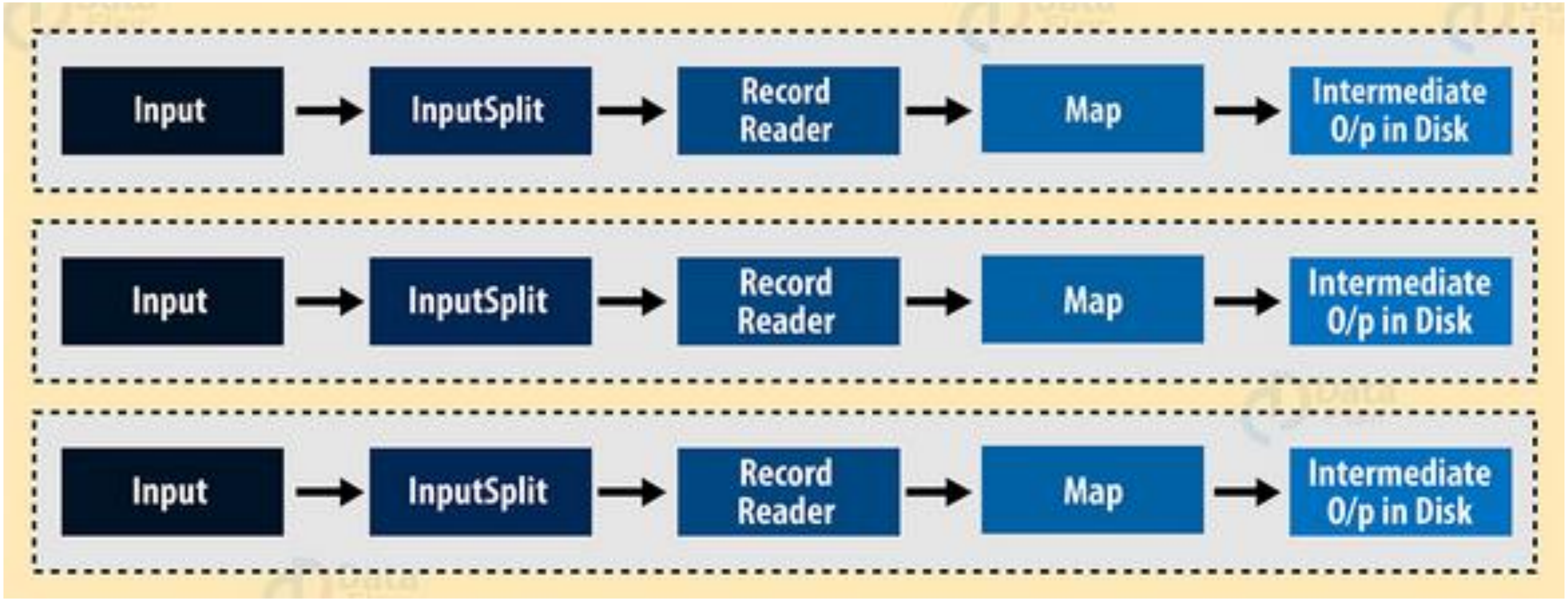
Mapper Class



Mapper Class



Mapper Class



Input Split

- Apache Hadoop can process any arbitrary data like log files, text files, structured data etc.
- In HDFS, InputSplit perform the logical partition of the data.
- InputSplit is the chunk of data that processed by single map (i.e. one Mapper can process one InputSplit at a time)
- Now each split is required to divide into the records
- It do not contain actual data rather they contain the references to the actual data (HDFS blocks).
- InputFormat is responsible for validating the input data, creating the inputsplit and divide them into the records.
- RecordReader reads the data from inputsplit (record) and converts them into key-value pair for the input to the Mapper class.

Record Reader

- RecordReader uses the data within the boundaries, defined by InputSplit.
- It creates key-value pairs for the mapper.
- It “start” is the byte position in the file, thus RecordReader starts generating key-value at ‘start.
- Then, the “end” is where it should stop reading records. In MapReduce, RecordReader load data from its source and it converts the data into key-value pairs suitable for reading by the mapper.
- RecordReader communicates with the inputsplit until it does not read the complete file.
- The MapReduce framework defines RecordReader instance by the InputFormat.

TextInputFormat provides 2 types of RecordReader

- 1) LineRecordReader-
- LineRecordReader in Hadoop is the default RecordReader that TextInputFormat provides.
- Hence, each line of the input file is the new value and key is byte offset.

2) SequenceFileRecordReader-

- It reads data specified by the header of a sequence file.

Reducer Class

- The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the **HDFS.**

Driver Class

- The major component in a MapReduce job is a **Driver Class.** It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of **Mapper** and **Reducer** Classes long with data types and their respective job names.

Advantages of MapReduce

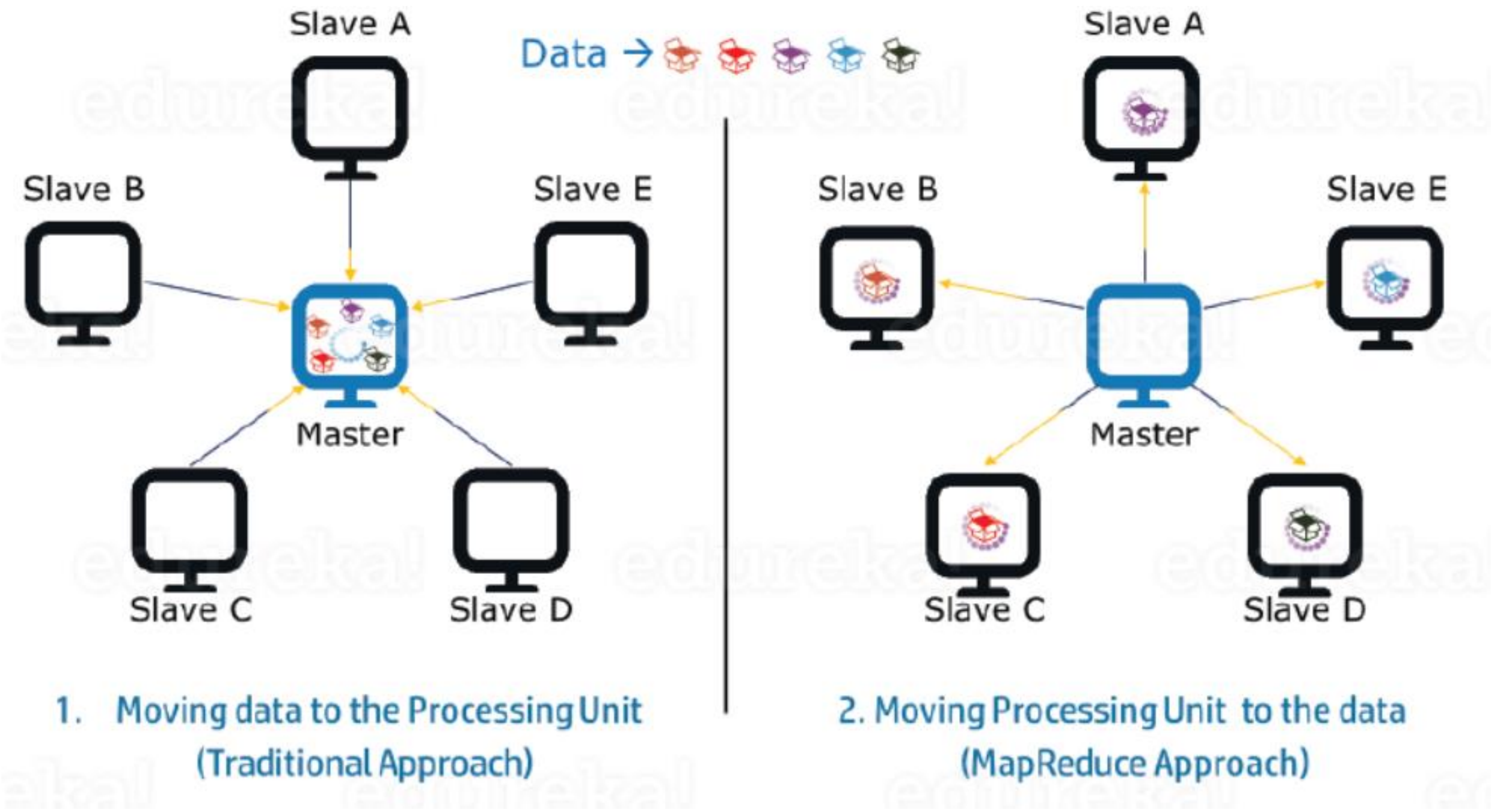
1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

2. Data Locality:

Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. It handle following issues: <u>data processing is costly and deteriorates the network performance., high processing time, over-burdened master node and may fail.</u>

Advantages of MapReduce


Slave A
Data →
Slave B
Slave E
Master
Slave C
Slave D
Slave A
Slave B
Slave E
Master
Slave C
Slave D
1. Moving data to the Processing Unit (Traditional Approach)
2. Moving Processing Unit to the data (MapReduce Approach)

Word count Job

File.txt
Size ::
500MB

Input : Text file
Output : count of words

Hi how are you?
how is your job?
how is your family
how is your sister
how is your brother
what is the time now
what is the strength of the
Hadoop

Hi how are you
how is your job

how is your family
how is your sister

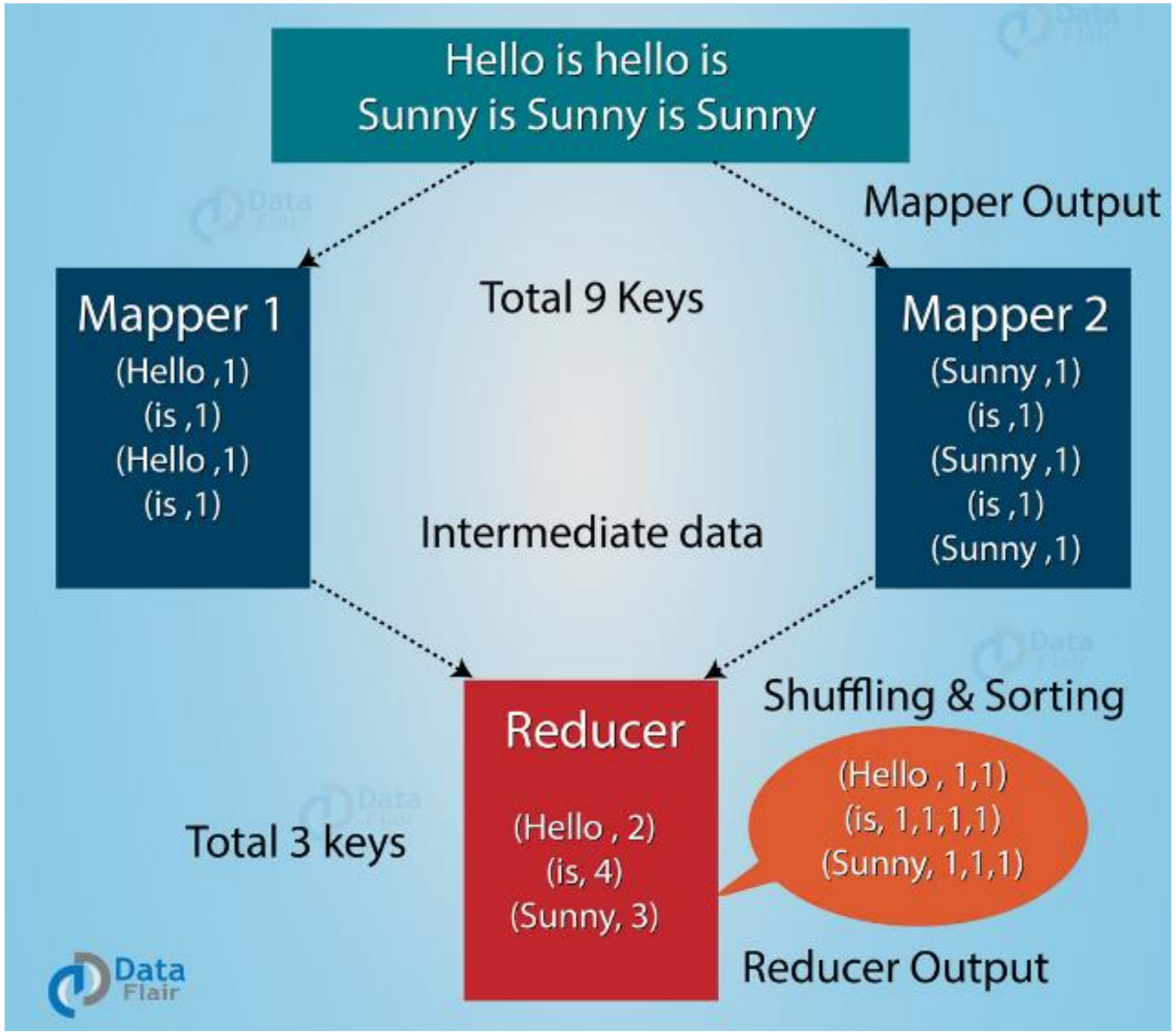
How is your brother
what is the time now

what is the strength of the
Hadoop

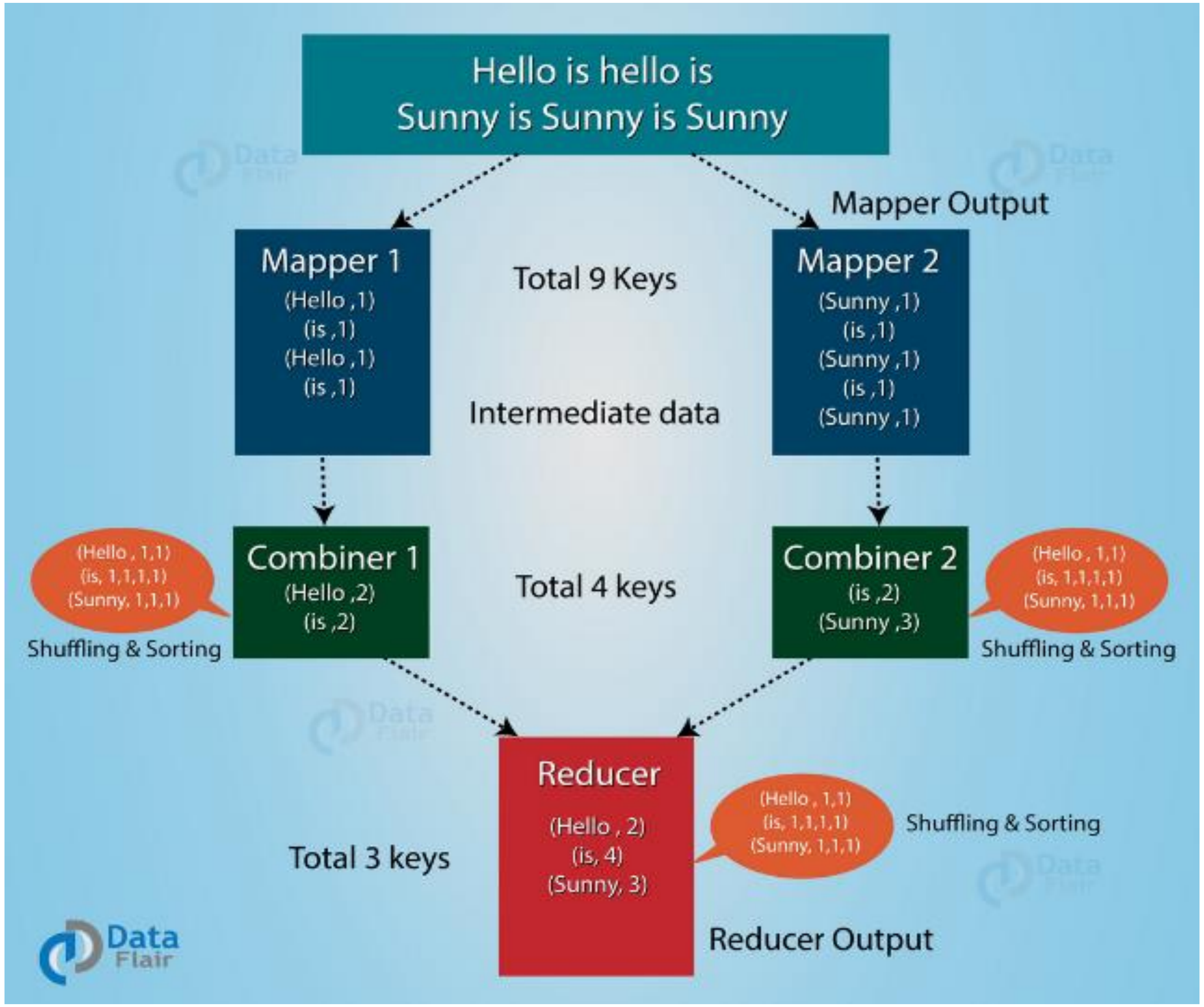
Hadoop/Map Reduce Combiners

- On a large dataset when we run **MapReduce job**, large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion. MapReduce framework provides a function known as **Hadoop Combiner** that plays a key role in reducing network congestion.
- The combiner in MapReduce is also known as ‘Mini-reducer’. The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer and its use is optional.

MapReduce program without Combiner



MapReduce program with Combiner



Advantages of MapReduce Combiner

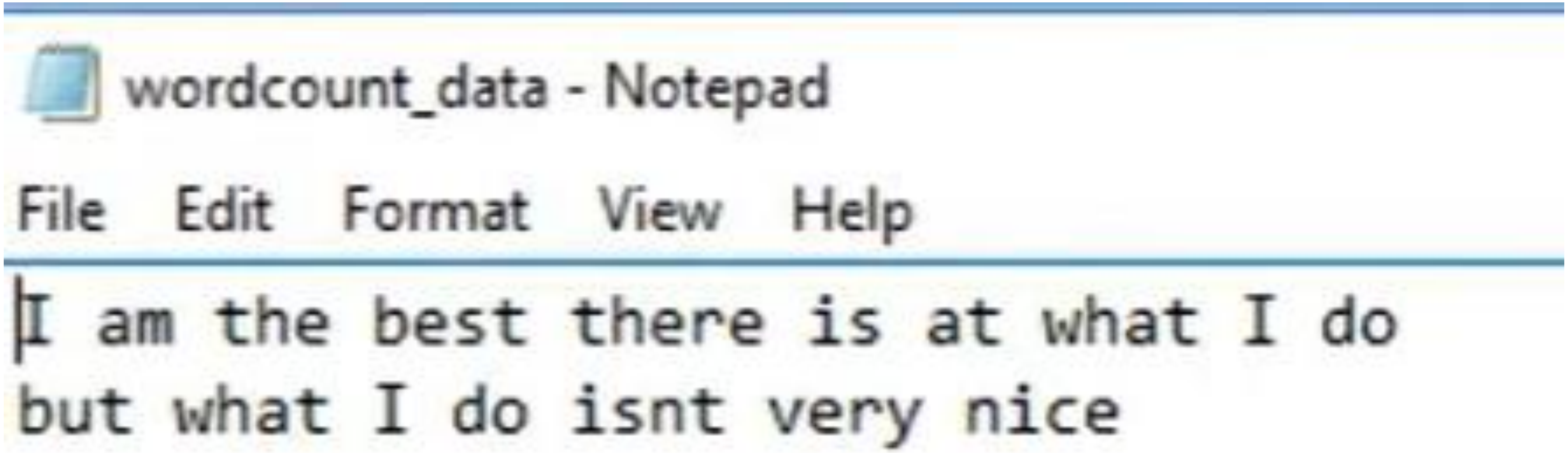
- Hadoop Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needed to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

Disadvantages of MapReduce Combiner

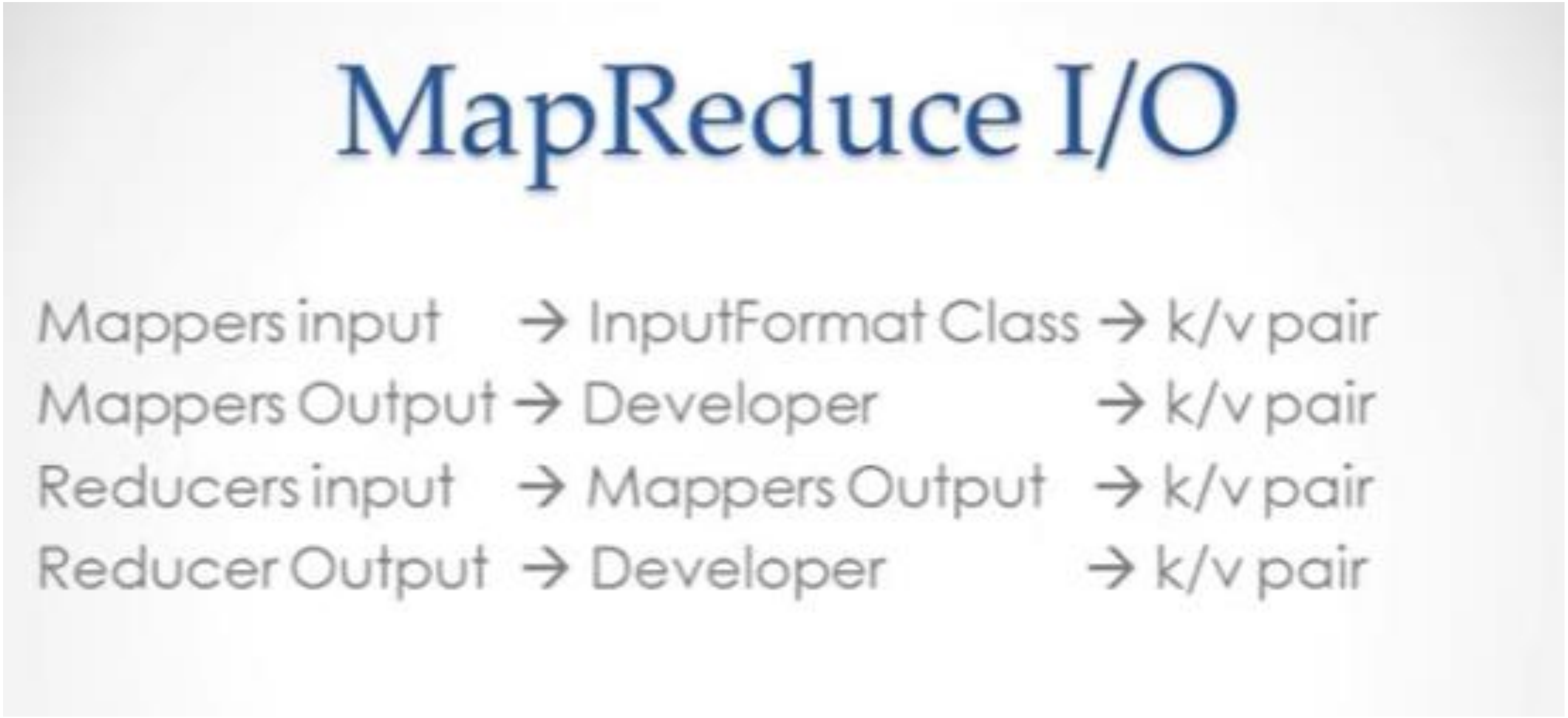
- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

Working of Mapper and Reducer using Example Program

- Word Count Problem: Sample Data File



```
I am the best there is at what I do
but what I do isnt very nice
```



MapReduce I/O

Mappers input	→ InputFormat Class	→ k/v pair
Mappers Output	→ Developer	→ k/v pair
Reducers input	→ Mappers Output	→ k/v pair
Reducer Output	→ Developer	→ k/v pair

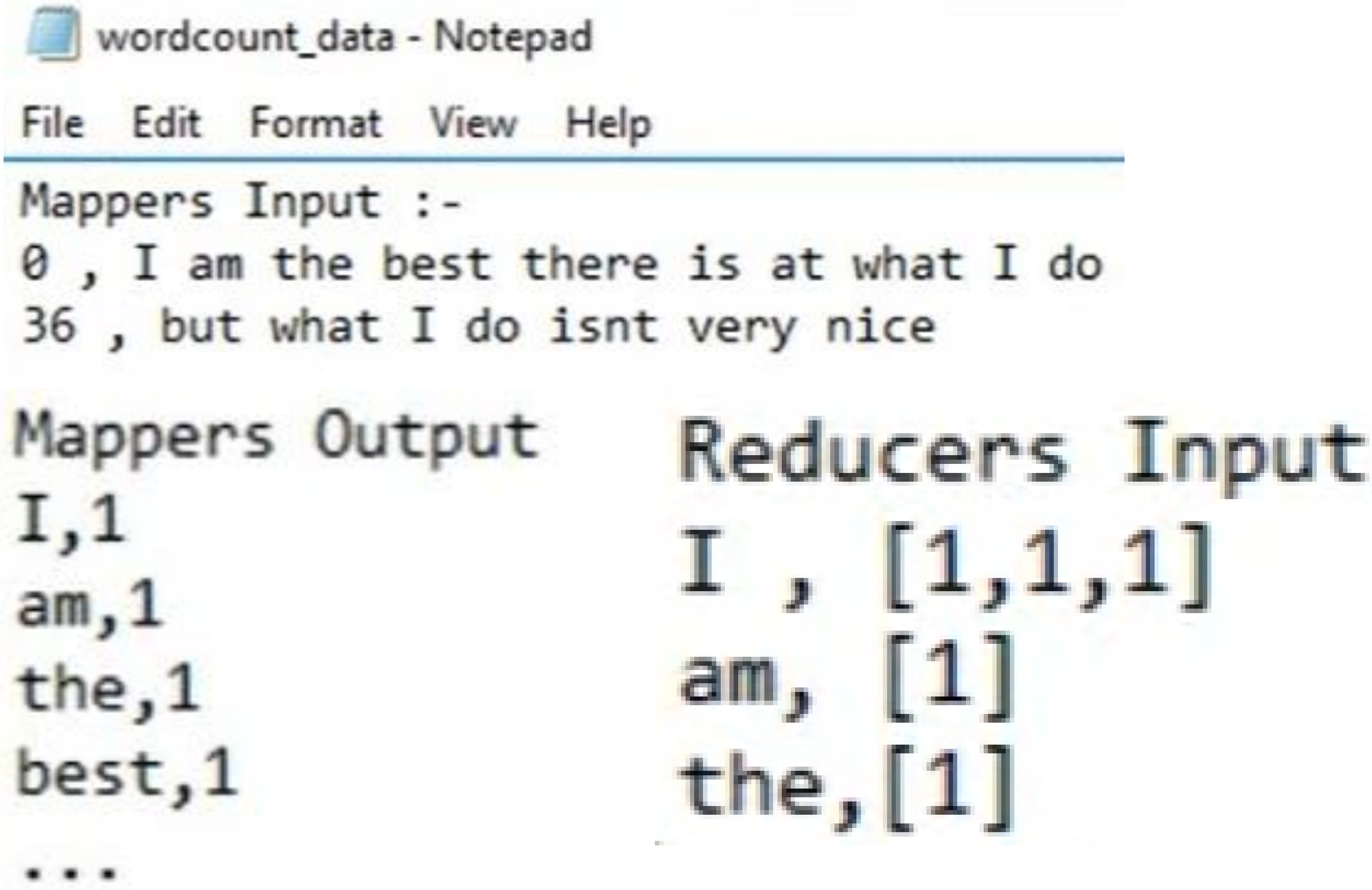
MapReduce I/O

Mappers Input : k1 , v1

Mappers Output : list (K2 , V2)

Reducers Input : k2 , list(v2)

Reducers output : k3 , v3



```
Mappers Input :-
0 , I am the best there is at what I do
36 , but what I do isnt very nice
```

```
Mappers Output
I,1
am,1
the,1
best,1
...
```

```
Reducers Input
I , [1,1,1]
am, [1]
the,[1]
```

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
```

```
public class WordCount {

• public static class WCMapper extends MapReduceBase implements
  Mapper<LongWritable, Text, Text, IntWritable>{
•       private final static IntWritable one = new IntWritable(1);
•       private Text word = new Text();
•       public void map(LongWritable key, Text value,
  OutputCollector<Text,IntWritable> output,
•             Reporter reporter) throws IOException{
•           String line = value.toString();
•           StringTokenizer  tokenizer = new StringTokenizer(line);
•           while (tokenizer.hasMoreTokens()){
•              word.set(tokenizer.nextToken());
•              output.collect(word, one); }   }  }
```

```
 public static class WCReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output,
         Reporter reporter) throws IOException
 {
        int sum=0;
        while (values.hasNext()) {
        sum+=values.next().get();
        }
        output.collect(key,new IntWritable(sum));
    }                    }
```

```
public static void main(String[] args) throws IOException{
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WCMapper.class);
        conf.setCombinerClass(WCReducer.class);
        conf.setReducerClass(WCReducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new Path(args[1]));
        JobClient.runJob(conf);         }     }
```

Execution of the WordCount Program

- Create jar file from Eclipse → Export -> Jar and save in your local directory, e.g: D:\BigData\abc.jar

- Start the Hadoop demons, check all services are up.

- Execute the Jar file:
  - hadoop jar D:\BigData\abc.jar /input /output
  - Hadoop jar D:\BigData\hadoop-3.2.1\share\hadoop\mapreduce\hadoop-mapreduce-examples-3.2.1.jar wordcount /input /output

- Explore the file in Namenode information