UNIT - III

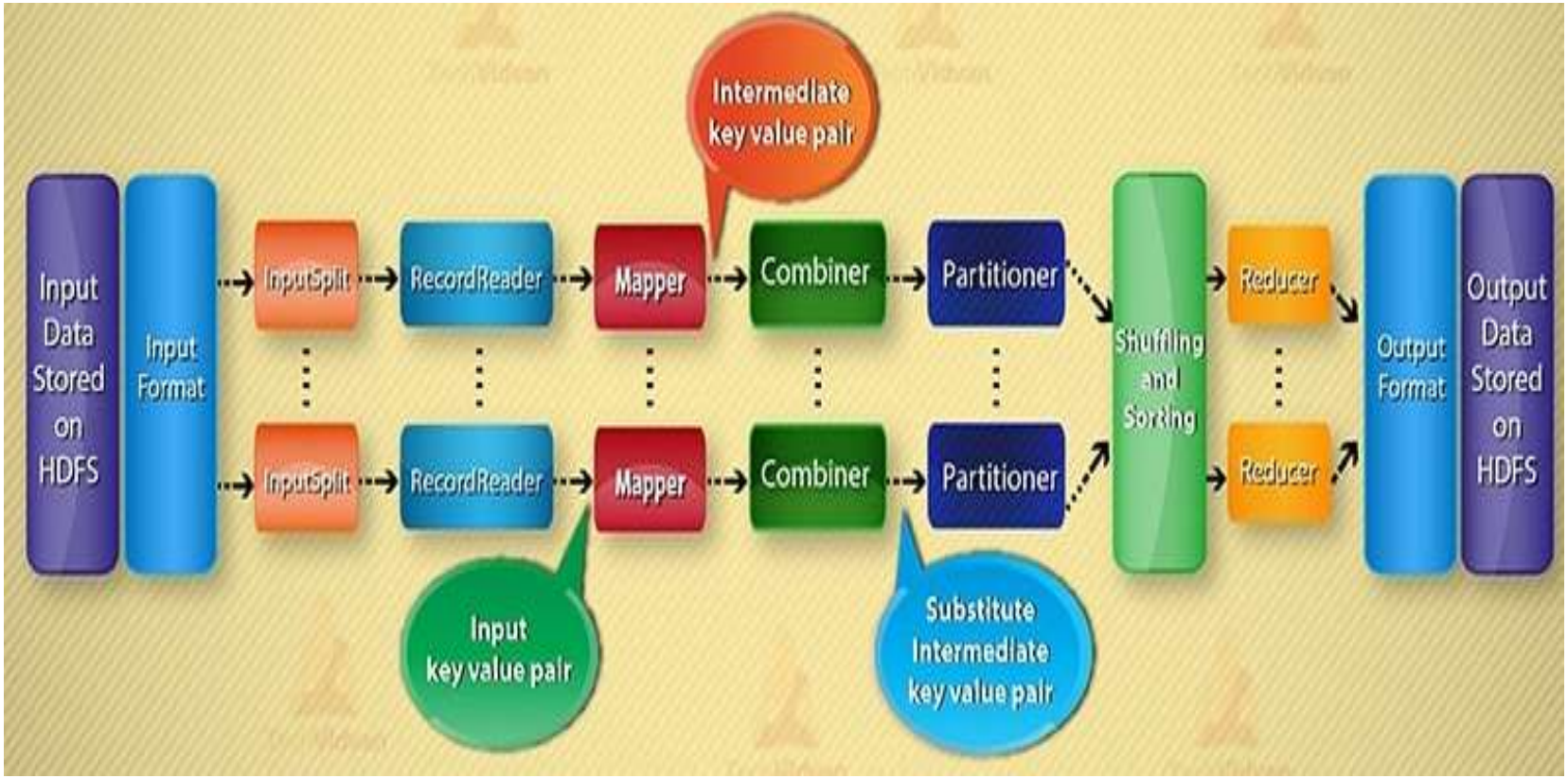
Writing MapReduce Programs

Dr.G.Dileep Kumar, Associate Professor
Dept. of CSE, KITS - Guntur

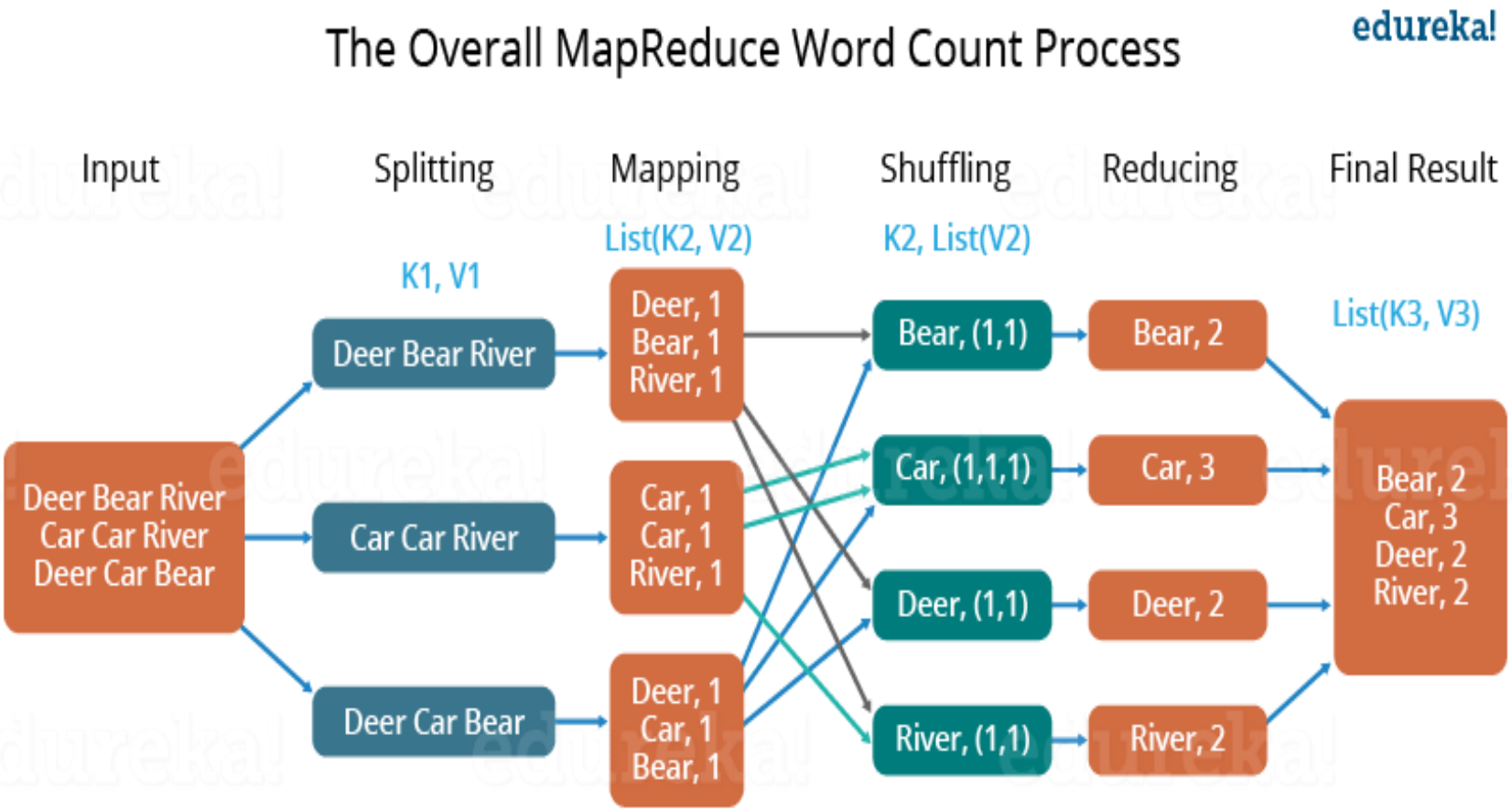
MapReduce:

- MapReduce is a software framework and programming model used for processing huge amounts of data.
- **MapReduce** program work in two phases, namely, Map and Reduce.
- Map tasks deal with splitting and mapping of data.
- Reduce tasks shuffle and reduce the data.
- The input to each phase is **key-value** pairs.

MapReduce Process Flow:



WordCount:


The Overall MapReduce Word Count Process
edureka!
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1, V1
List(K2, V2)
K2, List(V2)
List(K3, V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

WordCountMapper.java

```
package Count.BigData101;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
public class WordCountMapper extends 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,Context context) throws IOException,
              InterruptedException
               {
                        String line=value.toString();
                        StringTokenizer Tokens=new StringTokenizer(line);
                        while(Tokens.hasMoreTokens())
                                 {
                                          word.set(Tokens.nextToken());
                                          context.write(word,one);

                                 }
               }
}
```

WordCountReducer.java

```
package Count.BigData101;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.*;
public class WordCountReducer extends Reducer<Text,IntWritable,Text,IntWritable>
{
    private IntWritable totalWordCount=new IntWritable();
    public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,
    InterruptedException
    {
        int sum=0;
        Iterator<IntWritable> iterator=values.iterator();
        while(iterator.hasNext())
        {
            sum+=iterator.next().get();
        }
        totalWordCount.set(sum);
        context.write(new Text(key),totalWordCount);
    }
}
```

WordCountMain.java

```
package Count.BigData101;
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCountMain
{
    public static void main(String[] args) throws IOException, InterruptedException,
    ClassNotFoundException
    {
    if(args.length!=2)
    {
            System.out.println("Please provide the input and output path");
            System.exit(-1);
     }
    Configuration confg=new Configuration();
    @SuppressWarnings("deprecation")
    Job job=new Job(confg,"WordCountMain");
    job.setJobName("WordCountMain");
```

```
    job.setJarByClass(WordCountMain.class);
    job.setNumReduceTasks(2);
    FileInputFormat.addInputPath(job,new Path(args[0]));
    FileOutputFormat.setOutputPath(job,new
    Path(args[1]));
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Steps To Execute MR Program:

- Open Eclipse IDE.
- Create File -> New -> Java Project (WordCount)
- In Window open Library Tab -> Add External JARs
- Add External JARs - > Finish

Required JAR Files to Load:

- File System -> usr-> lib -> hadoop
  - hadoop-annotations.jar
  - hadoop-auth.jar
  - Hadoop-common.jar
- File System -> usr-> lib -> hadoop -> client-0.20
  - All jar Files
- File System -> usr-> lib -> hadoop -> lib
  - slf4j.-api-1.7.5.jar
  - commons-httpclient-3.1.jar

Weather Data

- Write a program that mines weather data.
- Weather sensors collecting data every hour at many locations across the globe.
- Gather a large volume of log data.
- The data we will use is from the National Climatic Data Center (NCDC, http://www .ncdc.noaa.gov/).
- The data is stored using a line-oriented ASCII format, in which each line is a record.

Ex: Input Data

To visualize the way the map works, consider the following sample lines of input data :

0067011990999991950051507004...9999999N9+00001+99999999999...

0043011990999991950051512004...9999999N9+00221+99999999999...

0043011990999991950051518004...9999999N9-00111+99999999999...

0043012650999991949032412004...0500001N9+01111+99999999999...

0043012650999991949032418004...0500001N9+00781+99999999999...

Format of a National Climate Data Center record

0057
332130 # USAF weather station identifier
99999 # WBAN weather station identifier
19500101 # observation date
0300 # observation time
4
+51317 # latitude (degrees x 1000)
+028783 # longitude (degrees x 1000)
FM-12
+0171 # elevation (meters)
99999
V020
320 # wind direction (degrees)
1 # quality code

N 0072
1
00450 # sky ceiling height (meters)
1 # quality code
C
N
010000 # visibility distance (meters)
1 # quality code
N
9
-0128 # air temperature (degrees Celsius x 10)
1 # quality code
-0139 # dew point temperature (degrees Celsius x 10)
1 # quality code
10268 # atmospheric pressure (hectopascals x 10)
1 # quality code

MaxTemperatureMapper.java

```
Import org.apache.hadoop.io.IntWritable;
Import org.apache.hadoop.io.LongWritable;
Import org.apache.hadoop.io.Text;
Import org.apache.hadoop.mapreduce.Mapper;
Import java.io.IOException;
Public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    Private static final int MISSING= 9999;
    Public void map(LongWritable key, Text value, Context context)  throwsIOException,
    InterruptedException {
    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if(line.charAt(87) == '+')
    {
            // parseInt doesn't like leading plus signs
             airTemperature = Integer.parseInt(line.substring(88, 92));
     }
    else
    {
                    airTemperature = Integer.parseInt(line.substring(87, 92));
     }
    String quality = line.substring(92, 93);
     if(airTemperature != MISSING&&quality.matches("[01459]")) {
      context.write(newText(year), newIntWritable(airTemperature));
    }
  }
```

MaxTemperatureReducer.java

```
Import org.apache.hadoop.io.IntWritable;
Import org.apache.hadoop.io.Text;
Import org.apache.hadoop.mapreduce.Reducer;
Import java.io.IOException;
Public class MaxTemperatureReducer extends Reducer<Text, IntWritable,
    Text, IntWritable>
{
 public void reduce(Text key, Iterable<IntWritable>values, Context context)
    throwsIOException, InterruptedException
{
        int maxValue = Integer.MIN_VALUE;
        for(Int Writable value : values)
        {
                maxValue = Math.max(maxValue, value.get());
        }
         context.write(key, new IntWritable(maxValue));
}
}
```

JobLauncher.java

```
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.lib.input.FileInputFormat;
Import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
Import java.io.IOException;
 public class JobLauncher
{
Public static void main(String[] args) throwsIOException,
    ClassNotFoundException, InterruptedException
    {
        Configuration conf = newConfiguration();
        Job job = newJob(conf, "Max Temparature");
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
```

```
        job.setJarByClass(JobLauncher.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job,
                new Path(args[0]));
        FileOutputFormat.setOutputPath(job,
                new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0: 1);
}
}
```

Understanding Hadoop API for MapReduce Framework (Old and New):

Hadoop OLD API (0.20) and New API (1.X or 2.X)
Hadoop Latest API Version is 3.1.1

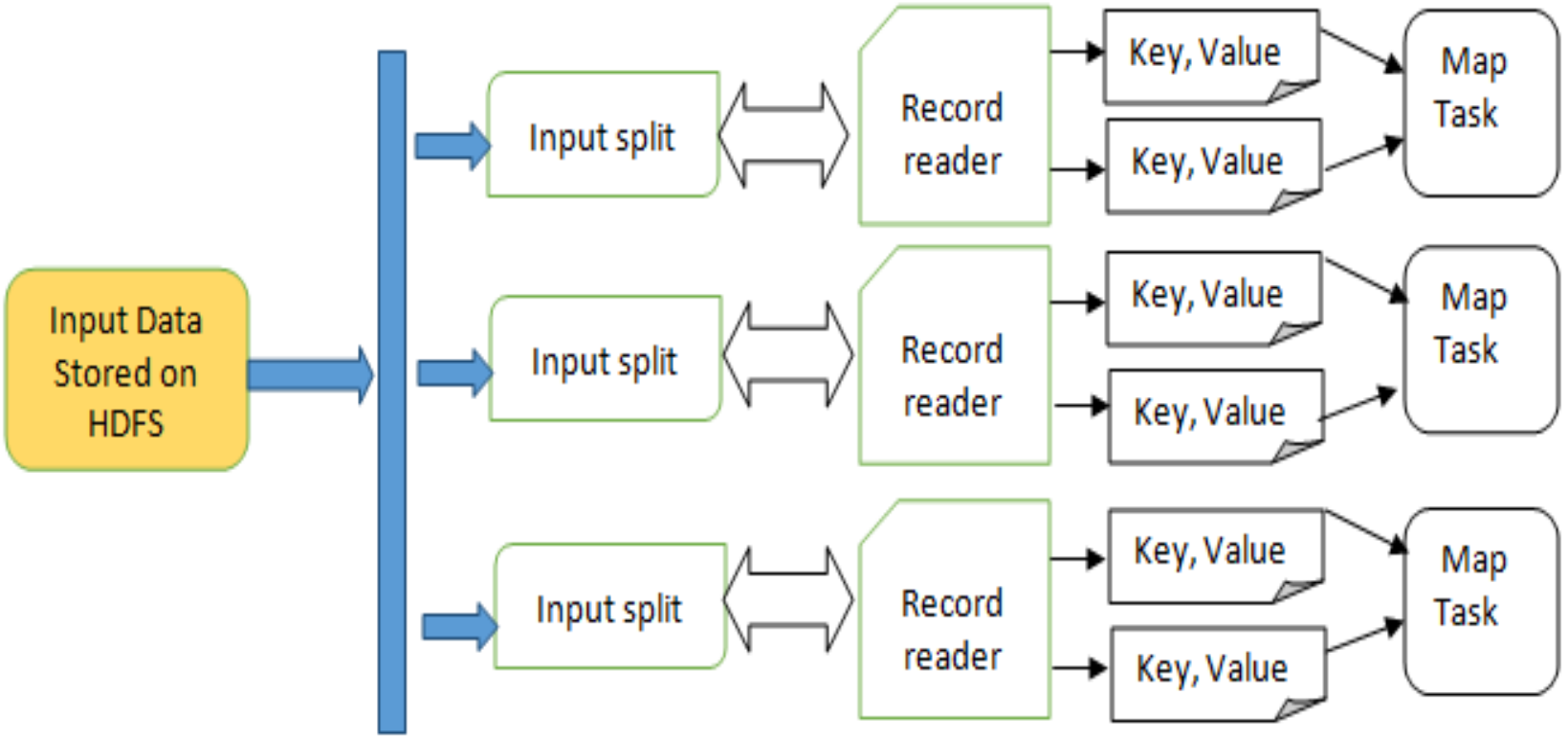
Diffrence	New API	OLD API
Mapper & Reducer	New API useing Mapper and Reducer as *Class* So can add a method (with a default implementation) to an abstract class without breaking old implementations of the class	IN OLD API used Mapper & Reduceer as *Interface* (still exist in New API as well)
Package	new API is in the *org.apache.hadoop.mapreduce* package	old API can still be found in *org.apache.hadoop.mapred.*
User Code to commnicate with MapReduce Syaterm	use "*context*" object to communicate with mapReduce system	*JobConf*, the *OutputCollector*, and the *Reporter* object use for communicate with Map reduce System

Control Mapper and Reducer execution	new API allows both mappers and reducers to control the execution flow by ***overriding the run()*** method.	Controlling mappers by writing a ***MapRunnable***, but no equivalent exists for reducers.
JOB control	Job control is done through the ***JOB*** class in New API	Job Control was done through ***JobClient*** (not exists in the new API)
Job Configuration	Job Configuration done through ***Configuration*** class via some of the helper methods on Job.	**jobconf** objet was use for Job configuration.which is extension of Configuration class. java.lang.Object extended by org.apache.hadoop.conf.Configuration extended by org.apache.hadoop.mapred.JobConf
OutPut file Name	In the new API map outputs are named ***part-m-nnnnn***, and reduce outputs are named ***part-r-nnnnn*** (where nnnnn is an integer designating the part number, starting from zero).	in the old API both map and reduce outputs are named ***part-nnnnn***
reduce() method passes values	In the new API, the reduce() method passes values as a ***java.lang.Iterable***	In the Old API, the reduce() method passes values as a ***java.lang.Iterator***

Basic Programs of Hadoop MapReduce:

- Driver Code
- Mapper Code
- Reducer Code
- Record Reader
- Combiner
- Partitioner

RecordReader:



org.apache.hadoop.mapreduce

Class RecordReader<KEYIN,VALUEIN>

java.lang.Object

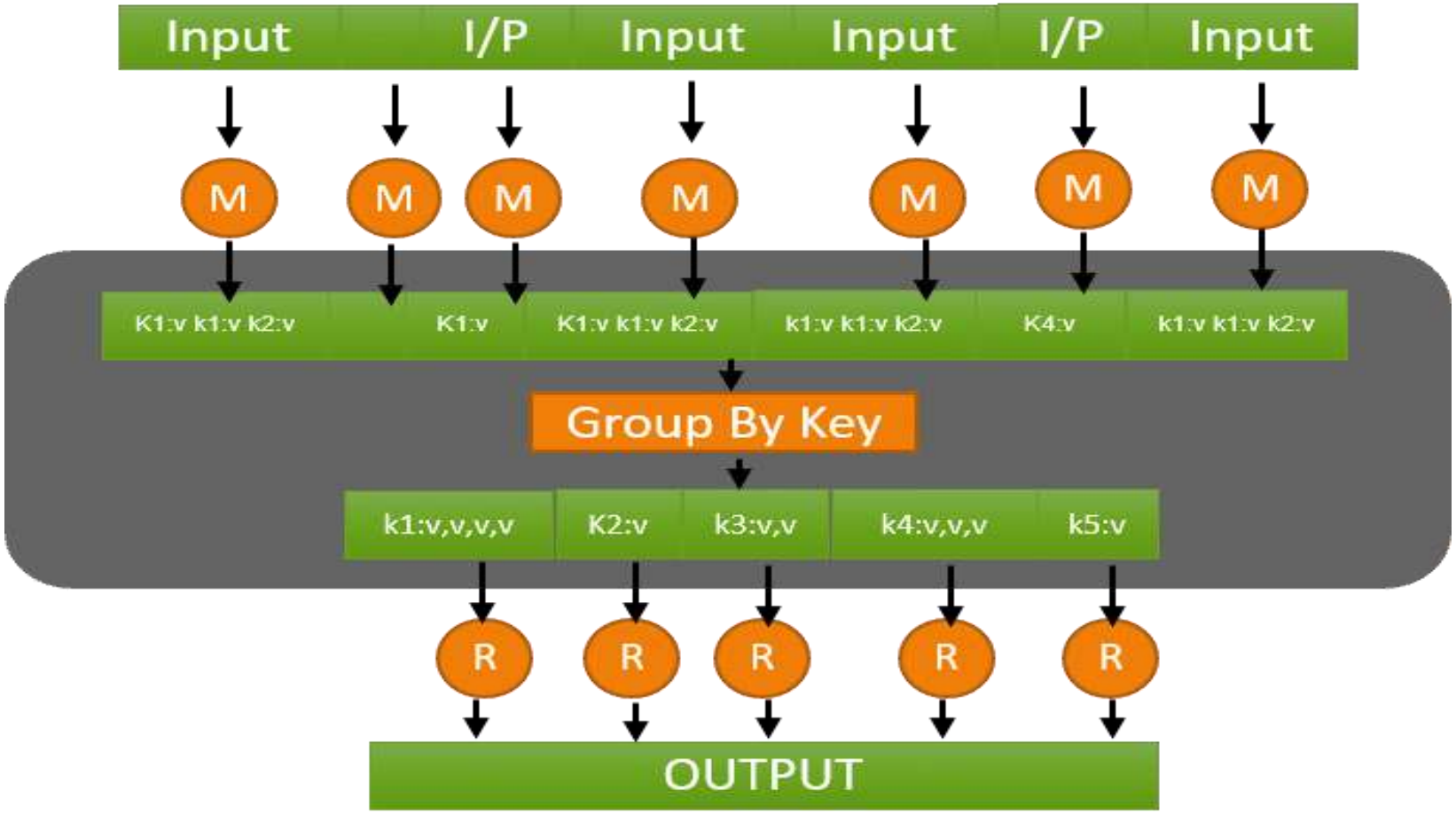
- org.apache.hadoop.mapreduce.RecordReader<KEYIN,VALUEIN>

- **Type Parameters:**KEYIN -VALUEIN -
- ***Constructor***
- public RecordReader()

Methods:

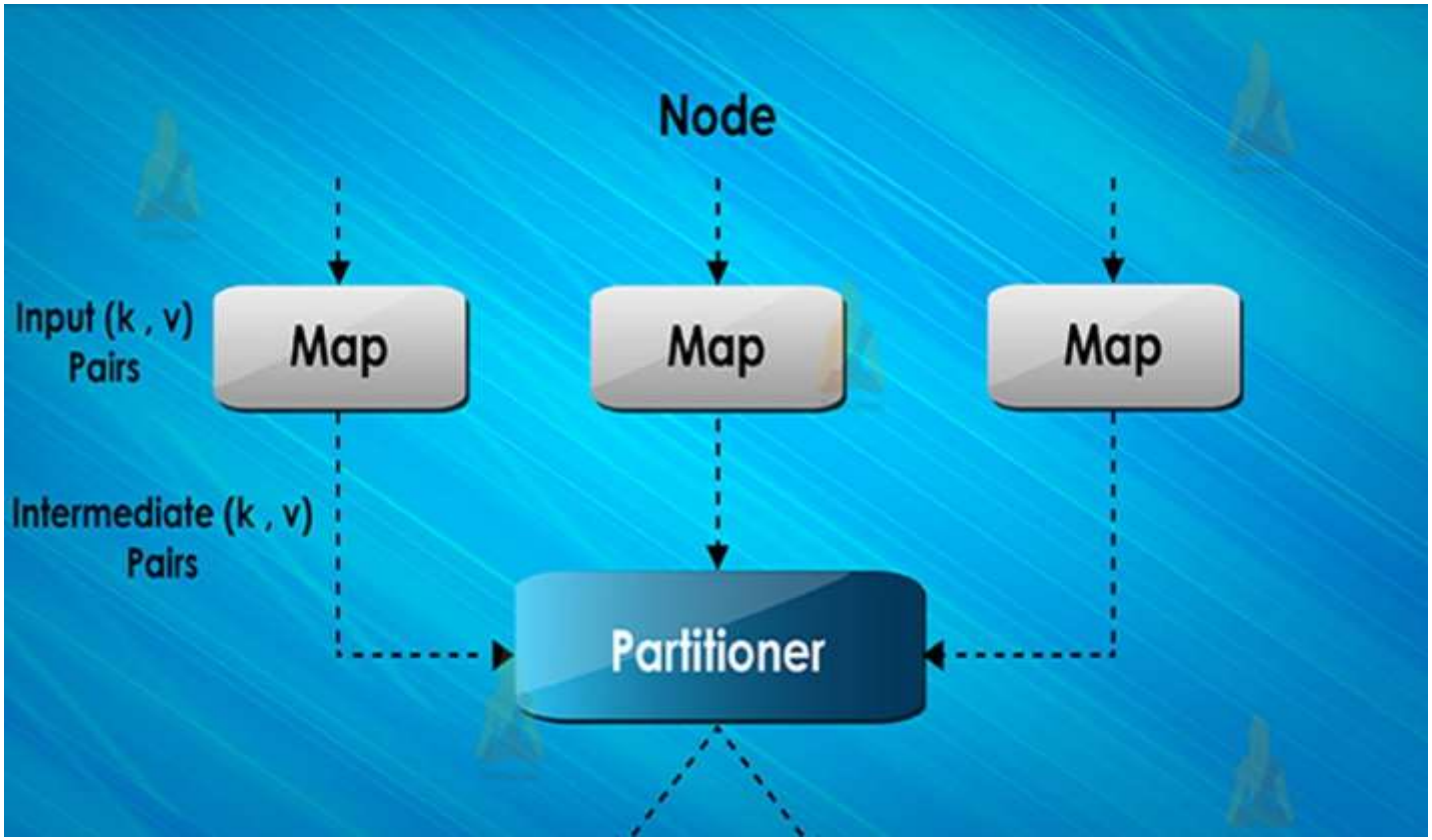
Method	Description
close()	Close the record reader.
getCurrentKey()	Get the current key
getCurrentValue()	Get the current value.
getProgress()	The current progress of the record reader through its data.
initialize(InputSplit split, TaskAttemptContext context)	Called once at initialization.
nextKeyValue()	Read the next key, value pair.

Combiner:


Input
I/P
Input
Input
I/P
Input
M
K1:v k1:v k2:v
K1:v
K1:v k1:v k2:v
k1:v k1:v k2:v
K4:v
k1:v k1:v k2:v
Group By Key
k1:v,v,v,v
K2:v
k3:v,v
k4:v,v,v
k5:v
R
OUTPUT

- A Combiner, also known as a semi-reducer.
- It is an optional class that operates by accepting the inputs from the Map class.
- It passes the output key-value pairs to the Reducer class.
- The main function of a Combiner is to summarize the map output records with the same key.

Partitioner:



org.apache.hadoop.mapreduce

Class Partitioner<KEY,VALUE>

java.lang.Object

- org.apache.hadoop.mapreduce.Partitioner<KEY,VALUE>

getPartition(**KEY** key, **VALUE** value, int numPartitions)

- Get the partition number for a given key (hence record) given the total number of partitions.