

Where Map-Reduce is used ?

✓ Weather Forecasting



✓ Health Care



✓ Oil & Gas Industry



Weather Data

ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/

← → ↻ ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/

Index of /pub/data/uscrn/prod

Name	Size
[parent directory]	
2000/	
2001/	
2002/	
2003/	
2004/	
2005/	
2006/	
2007/	
2008/	
2009/	
2010/	
2011/	
2012/	
2013/	
Daily01_New_IR_Sensor_2012-10-24.txt	1.1 kB
File_name_change.04112011.txt	668 B
README.txt	13.7 kB
obsolete/	
snapshots/	
updates/	

← → ↻ ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/produ

Index of /pub/data/uscrn/products

Name	Size
[parent directory]	
CRND0103-2013-AK_Barrow_4_ENE.txt	34.5 kB
CRND0103-2013-AK_Fairbanks_11_NE.txt	34.5 kB
CRND0103-2013-AK_Gustavus_2_NE.txt	34.5 kB
CRND0103-2013-AK_Kenai_29_ENE.txt	34.5 kB
CRND0103-2013-AK_King_Salmon_42_SE.txt	34.5 kB
CRND0103-2013-AK_Metlakatla_6_S.txt	34.5 kB
CRND0103-2013-AK_Port_Alsworth_1_SW.txt	34.5 kB
CRND0103-2013-AK_Red_Dog_Mine_3_SSW.txt	34.5 kB
CRND0103-2013-AK_Sand_Point_1_ENE.txt	34.5 kB
CRND0103-2013-AK_Sitka_1_NE.txt	34.5 kB
CRND0103-2013-AK_St_Paul_4_NE.txt	34.5 kB
CRND0103-2013-AK_Tok_70_SE.txt	34.5 kB
CRND0103-2013-AL_Brewton_3_NNE.txt	34.5 kB
CRND0103-2013-AL_Clanton_2_NE.txt	34.5 kB
CRND0103-2013-AL_Courtland_2_WSW.txt	34.5 kB
CRND0103-2013-AL_Cullman_3_ENE.txt	34.5 kB
CRND0103-2013-AL_Fairhope_3_NE.txt	34.5 kB
CRND0103-2013-AL_Gadsden_19_N.txt	34.5 kB
CRND0103-2013-AL_Gainesville_2_NE.txt	34.5 kB
CRND0103-2013-AL_Greensboro_2_WNW.txt	34.5 kB
CRND0103-2013-AL_Guntersville_2_SW.txt	34.5 kB
CRND0103-2013-AL_Highland_Home_2_S.txt	34.5 kB
CRND0103-2013-AL_Muscle_Shoals_2_N.txt	34.5 kB
CRND0103-2013-AL_Northport_2_S.txt	34.5 kB
CRND0103-2013-AL_Russellville_4_SSE.txt	34.5 kB
CRND0103-2013-AL_Scottsboro_2_NE.txt	34.5 kB
CRND0103-2013-AL_Selma_13_WNW.txt	34.5 kB
CRND0103-2013-AL_Selma_6_SSE.txt	34.5 kB
CRND0103-2013-AL_Talladega_10_NNE.txt	34.5 kB
CRND0103-2013-AL_Thomasville_2_S.txt	34.5 kB

Use Case in HealthCare

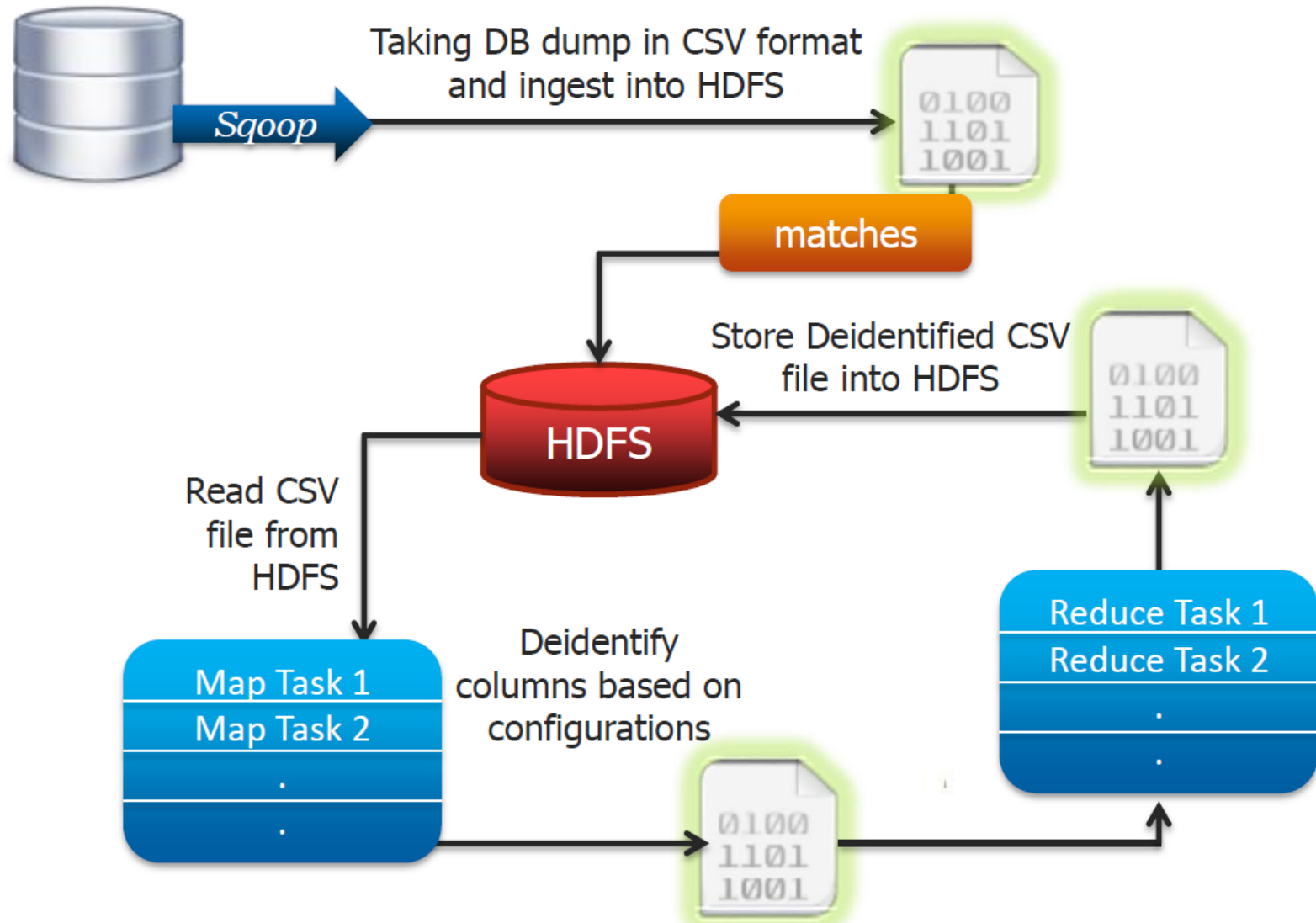
Problem Statement:

- ✓ De-identify personal health information.

Challenges:

- ✓ Huge amount of data flows into the systems daily and there are multiple data sources that we need to aggregate data from.
- ✓ Crunching this huge data and de-identifying it in a traditional way had problems.

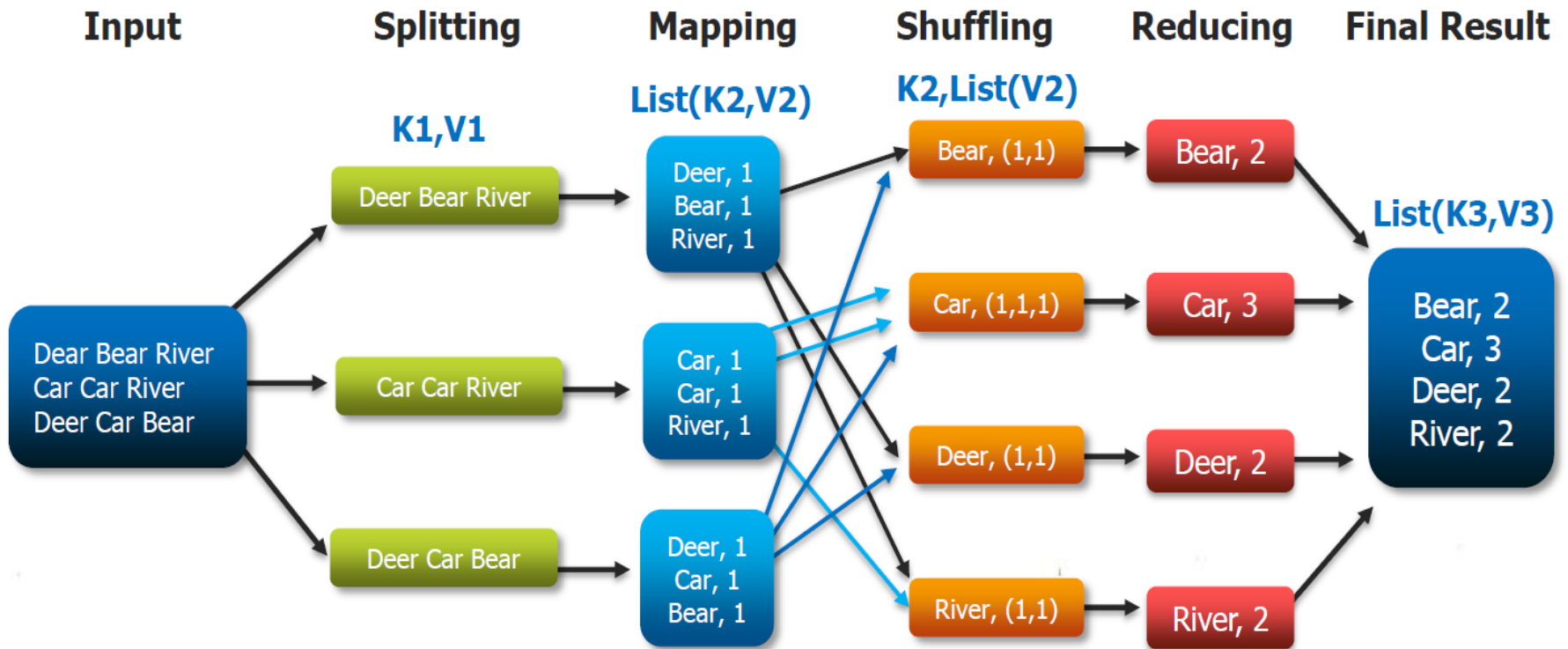
Architecture



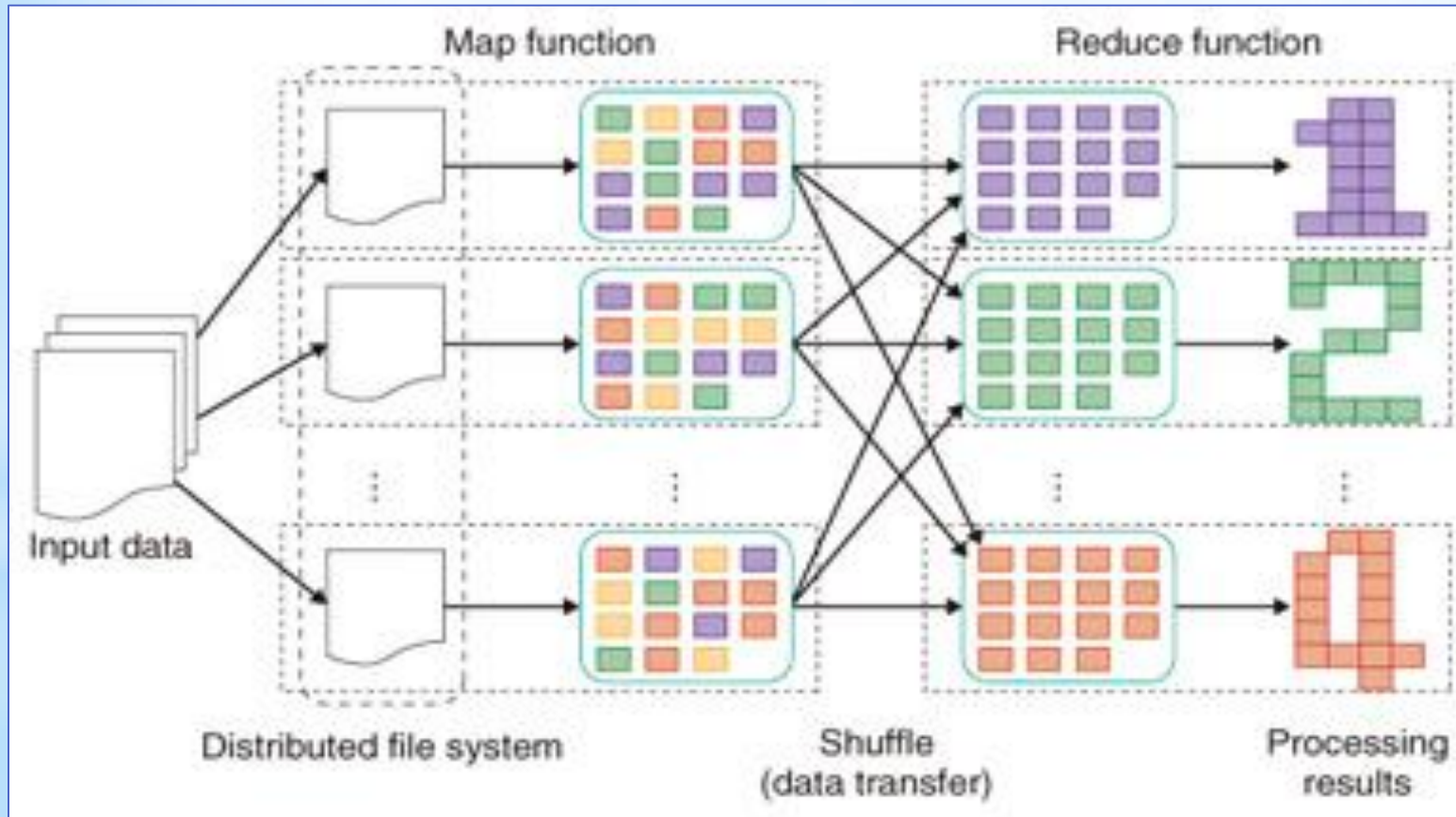
Traditional Way

Flow of Map Reduce Process

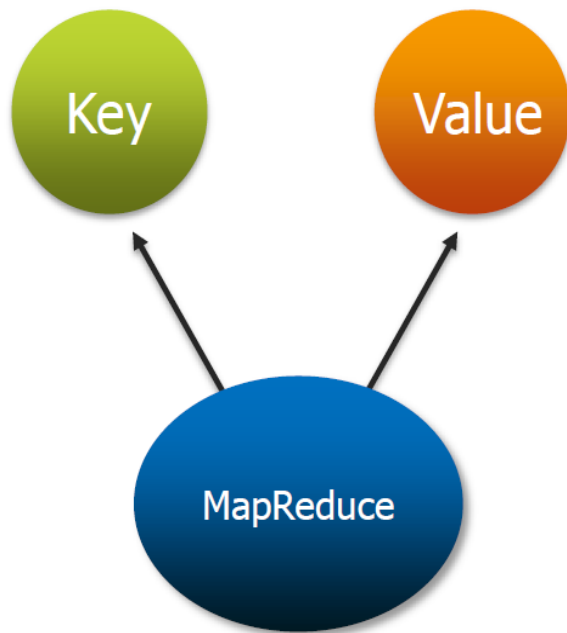
The Overall MapReduce Word Count Process



Flow of Map Reduce Process



Anatomy of Map-Reduce Program



Map:

(K1, V1)

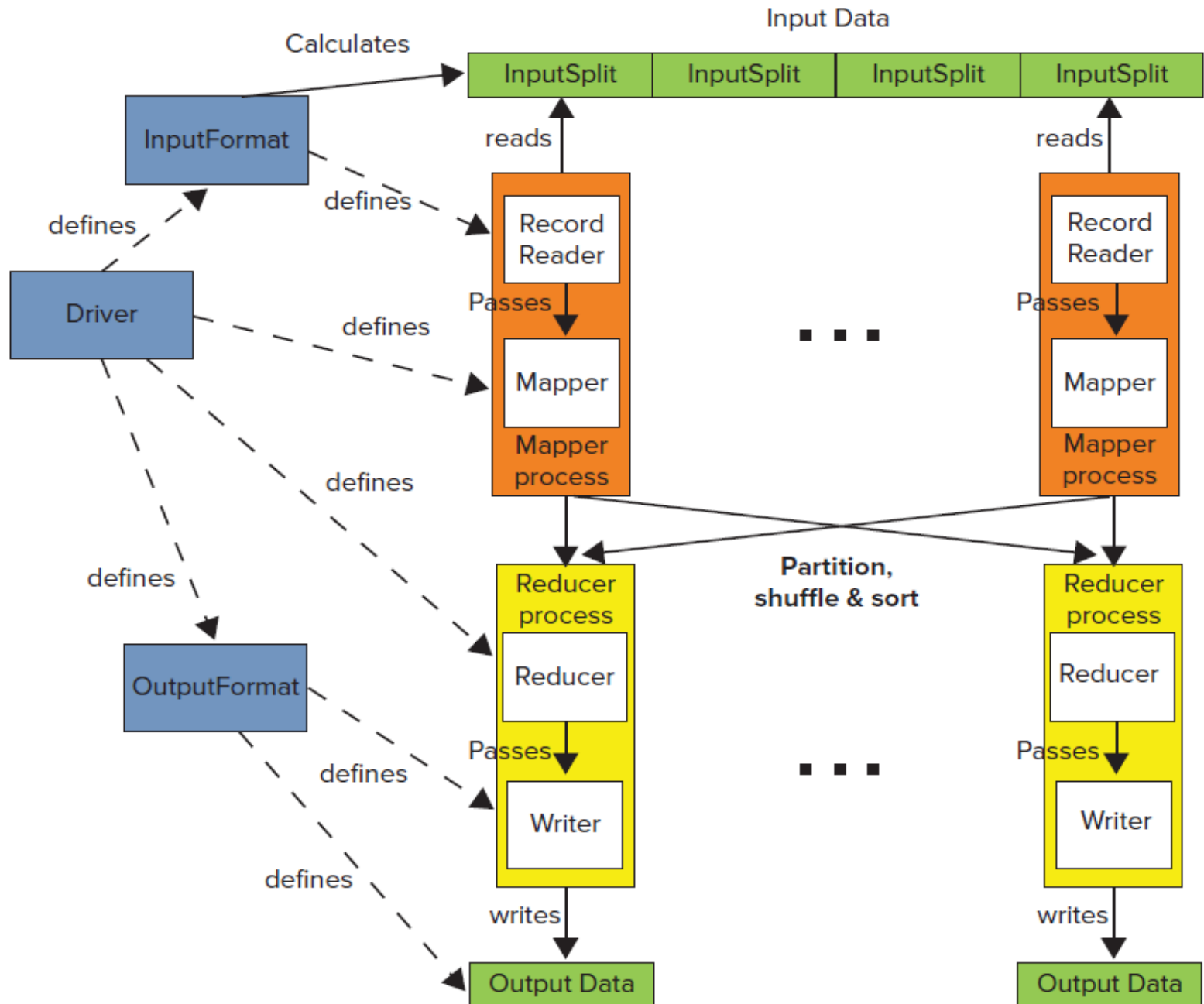
List (K2, V2)

Reduce:

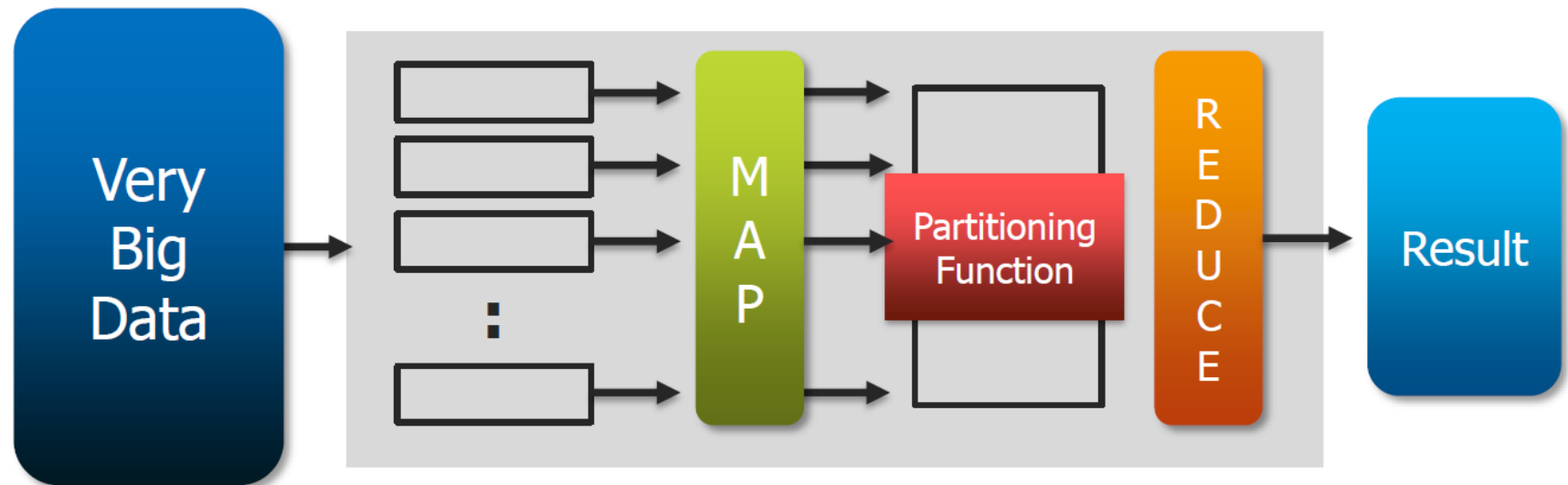
(K2, list (V2))

List (K3, V3)

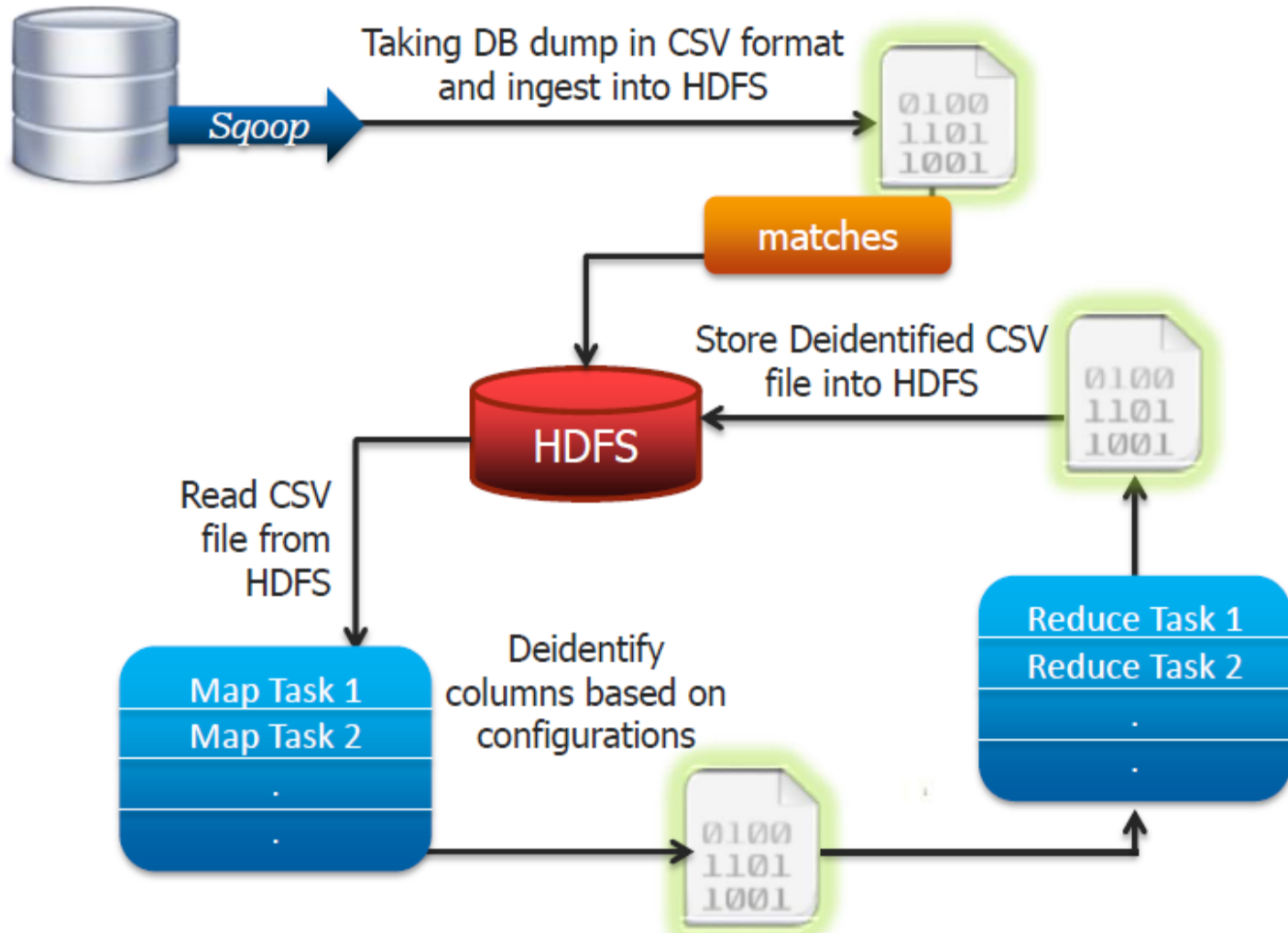
Map-Reduce Flow :



Map-Reduce Way



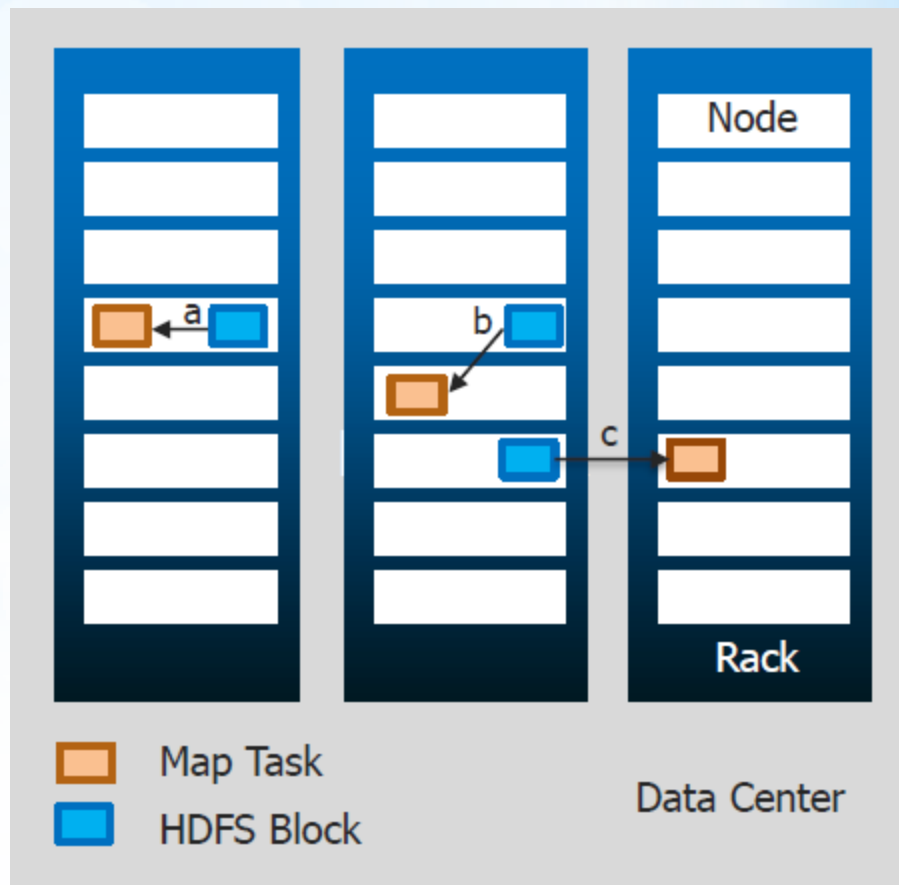
Revisit de-identification Architecture



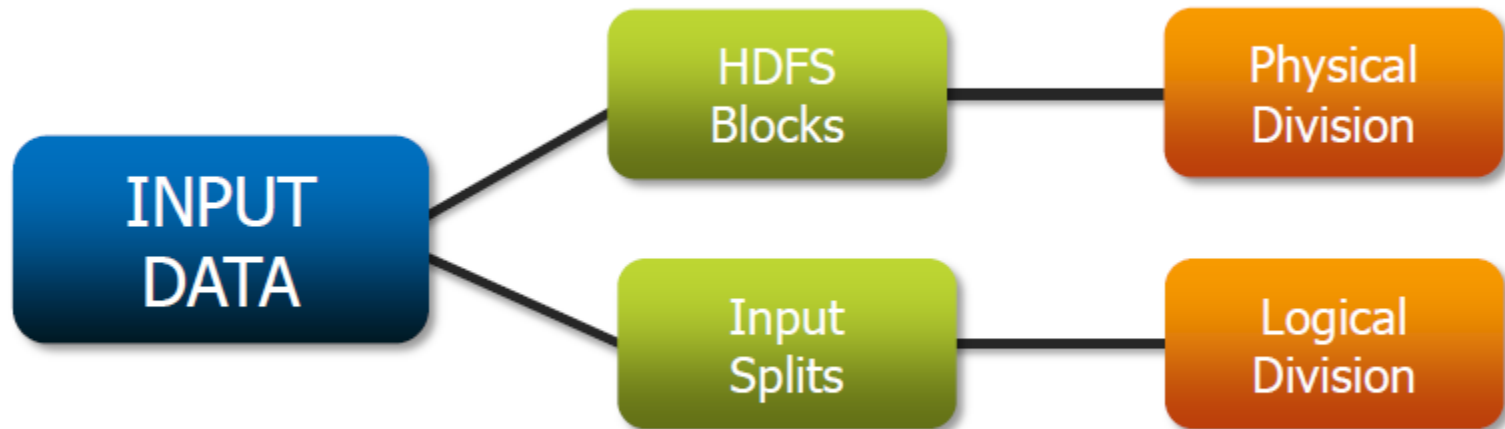
Why Map-Reduce ?

Two biggest Advantages:

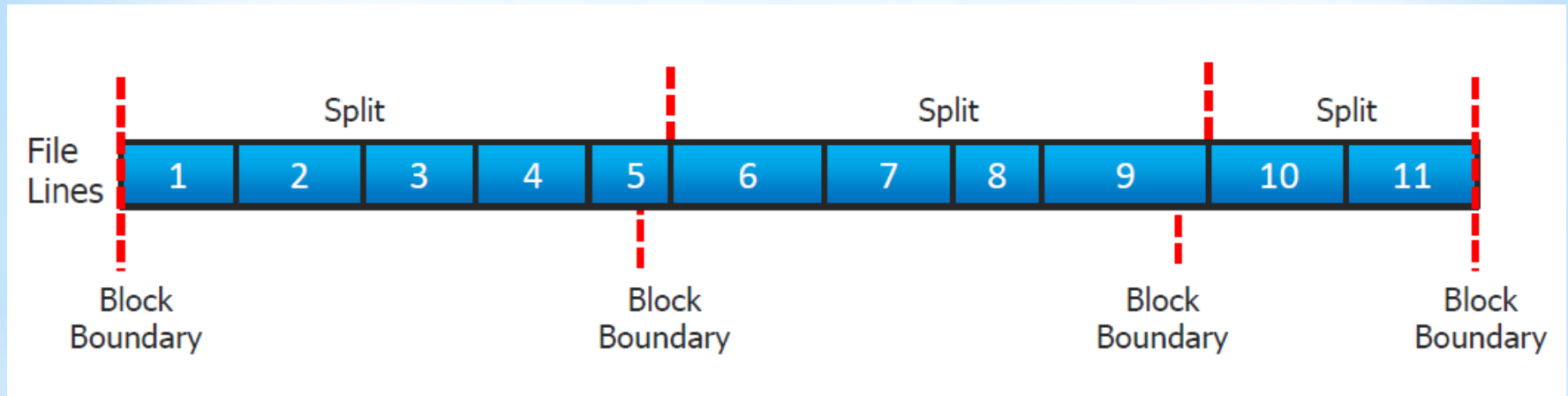
- ✓ Taking processing to the data
- ✓ Processing data in parallel



Input Splits



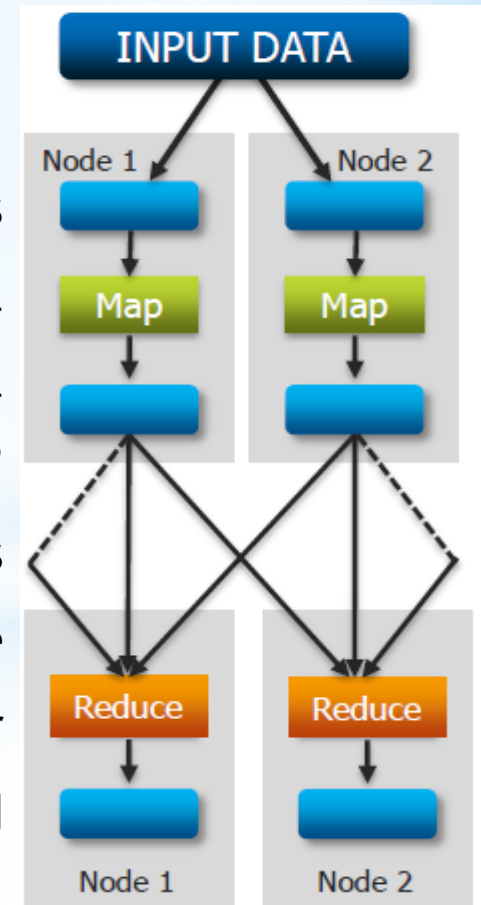
Relation Between Input splits and HDFS Blocks



- ✓ Logical records do not fit neatly into the HDFS blocks.
- ✓ Logical records are lines that cross the boundary of the blocks.
- ✓ First split contains line 5 although it spans across blocks.

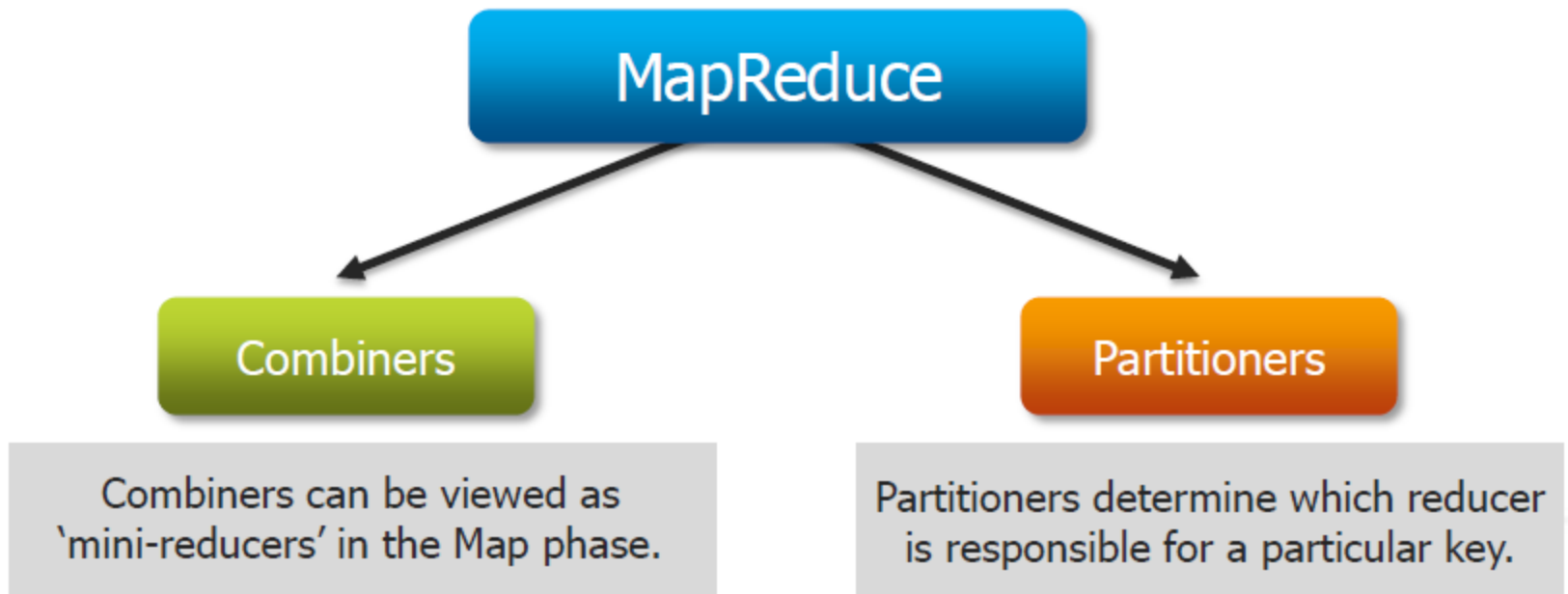
Map-Reduce Flow

Input data is distributed to nodes
Each map task works on a “split” of data
Mapper outputs intermediate data
Data exchange between nodes in a “shuffle”
process
Intermediate data of the **same key** goes to the
same reducer
Reducer output is stored

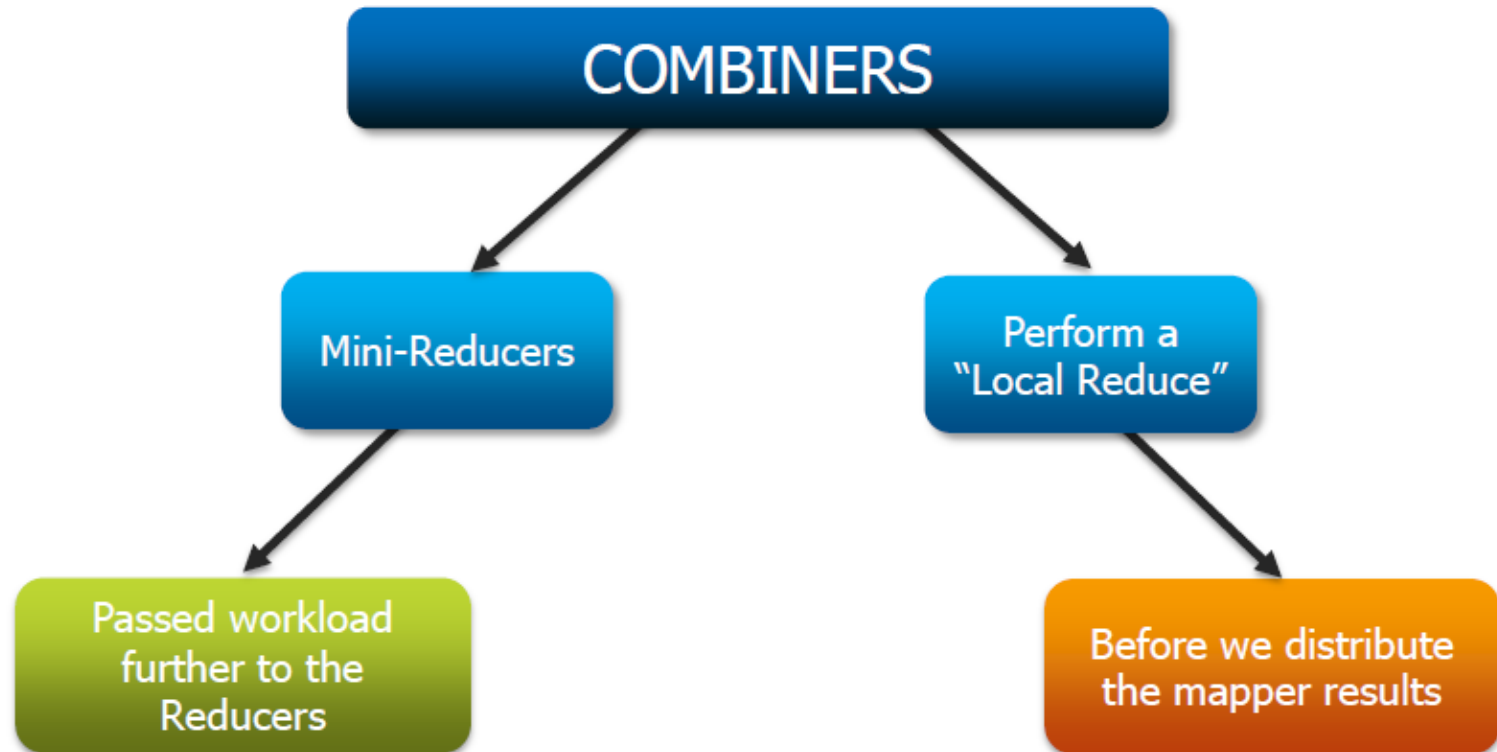


Overview of Map-Reduce

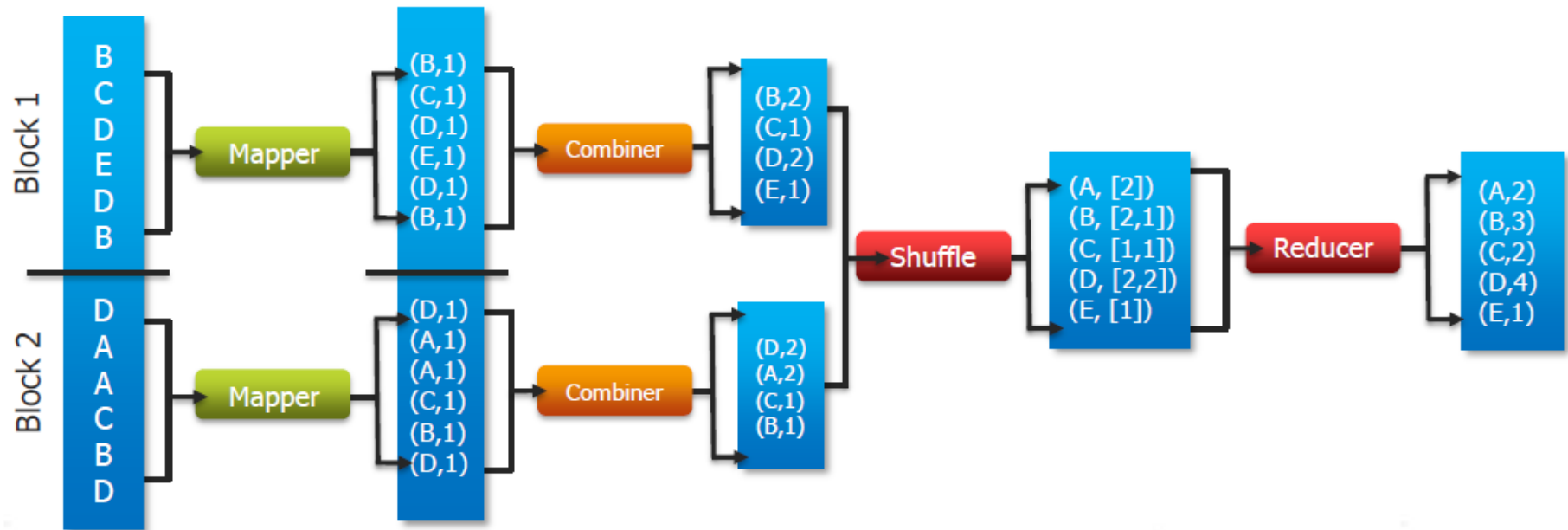
Complete view of MapReduce, illustrating combiners and partitioners in addition to Mappers and Reducers



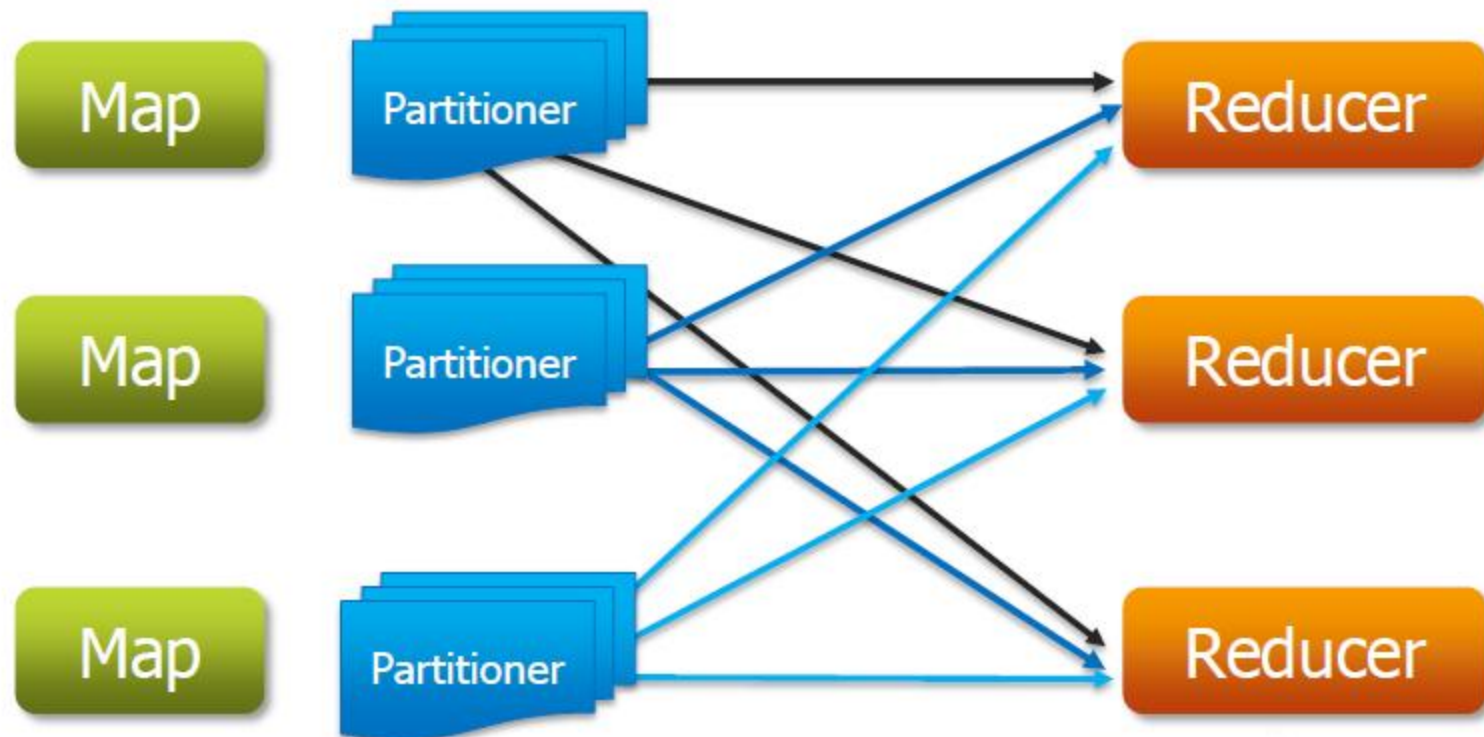
Combiner – Local Reduce



Combiner



Partitioner – Redirecting Output from Mapper



Input Formats

Each line in the text files is a **record**.

KeyValueTextInputFormat



Text Files

SequenceFileInputFormat



Sequence Files

Advanced Map Reduce

Advanced Map Reduce - Topics

- Combiners and Partitions
- Sequence file
- Distributed Cache
- Map Side Join
- Reduce Side Join
- Custom Input Formats
- Counter(system and User Defined)
- MR Unit(for Map Reduce Unit Testing)

Combiners and Partitioners

```
public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
```

```
@Override
```

```
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 class MyPartitioner implements Partitioner<Text, IntWritable>
{
```

```
@Override
```

```
public int getPartition(Text key, IntWritable value, int numPartitions) {
```

```
String myKey = key.toString().toLowerCase();
```

```
if (myKey.equals("hadoop")) {
return 0;
}
```

```
if (myKey.equals("data")) {
return 1;
}
```

```
else {
return 2;
}
}
```

```
@Override
```

```
public void configure(JobConf arg0) {
```

```
// Gives you a new instance of JobConf if you want to change Job
// Configurations}
```

```
public static void main(String[] args) throws Exception {
```

```
JobConf conf = new JobConf(WithPartitioner.class);
conf.setJobName("wordcount");
```

```
// Forcing program to run 3 reducers
conf.setNumReduceTasks(3);
```

```
conf.setMapperClass(Map.class);
conf.setCombinerClass(Reduce.class);
conf.setReducerClass(Reduce.class);
conf.setPartitionerClass(MyPartitioner.class);
```

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.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);
}
```

```
public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
```

```
@Override
```

```
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()) {
value.set(tokenizer.nextToken());
output.collect(value, new IntWritable(1));
}
}
```

Sequence File

File Based Data structures provided by Hadoop

❖ Sequence File

❖ MapFile

Key	Value
Datetimestamp1	Log file 1
Datetimestamp1	Log file 2
Datetimestamp1	Log file 3
Datetimestamp1	Log file ..n

Advantages:

Compression possible :

1. None
2. Records Level
3. Block Level

- ☐ So it takes less space
- ☐ Less IO
- ☐ Less Bandwidth
- ☐ Splittable

- Small files are not good for Hadoop since its consumes NameNode's RAM as a part of huge Metadata....
- The Solution to this problem is - Sequence file...Where you can store huge no of small files and it can be merged-compressed at block level and processed like one huge big file

Sequence File

Why Do We Need A Sequence File?

1) Dealing With Binary Data

Some applications require the usage of specialized data structures. Though Hadoop typically works on text data, we might have to deal with binary data (images, videos etc.) sometimes or treat an entire text file as a single record.

2) Problem With Small Files

HDFS is a distributed file system, mainly designed for batch processing of large volumes of data .

In case the size of a file is much smaller than this default block size, there is a dramatic degradation of performance, because of large number of seeks and lots of hopping from one datanode to another to retrieve a small file, which is inefficient.

In case of MapReduce, when file size is very small, the input for each process is very little and there are large number of map tasks. For example, a 10GB file broken up into files of size 100KB each, use a map of their own. Thus the time taken to finish the job considerably increases.

For solving these two problems, we need a Sequence file. A Sequence file is a data structure for binary key-value pairs. it can be used as a common format to transfer data between MapReduce jobs.

Another important advantage of a sequence file is that it can be used as an archive to pack smaller files. This avoids the above mentioned problems with small files.

Sequence File

How Is It Stored Internally?

A sequence file is composed of a header and one or more records. The first three bytes of a sequence file are the bytes SEQ, which acts as a magic number, followed by a single byte representing the version number. The header contains other fields, including the names of the key and value classes, compression details, user-defined metadata etc. Each file has a randomly generated sync marker, whose value is stored in the header. Sync markers appear between records in the sequence file, not necessarily between every pair of records.

The internal format of the records depends on whether compression is enabled, and if it is, whether it is record compression or block compression. If no compression is enabled (the default), each record is made up of the record length (in bytes), the key length, the key, and then the value.

The format for record compression is almost identical to no compression, except the value bytes are compressed using the codec defined in the header. Keys are not compressed.

Block compression compresses multiple records at once, it is therefore more compact than and should generally be preferred over record compression because it has the opportunity to take advantage of similarities between records. Records are added to a block until it reaches a minimum size in bytes, defined by the `io.seqfile.compress.blocksize` property, the default is 1 million bytes. A sync marker is written before the start of every block. The format of a block is a field indicating the number of records in the block, followed by four compressed fields: the key lengths, the keys, the value lengths, and the values.

Map File

Map File is a Directory and NOT a file. It has two files

- a) Index file - its sequence file and
- b) Data File - it is also sequence file..

```
% hadoop fs -text numbers.map/data | head
1      One, two, buckle my shoe
2      Three, four, shut the door
3      Five, six, pick up sticks
4      Seven, eight, lay them straight
5      Nine, ten, a big fat hen
6      One, two, buckle my shoe
7      Three, four, shut the door
8      Five, six, pick up sticks
9      Seven, eight, lay them straight
10     Nine, ten, a big fat hen
```

The *index* file contains a fraction of the keys, and contains a mapping from the key to that key's offset in the *data* file:

```
% hadoop fs -text numbers.map/index
1      128
129    6079
257    12054
385    18030
513    24002
641    29976
769    35947
897    41922
```

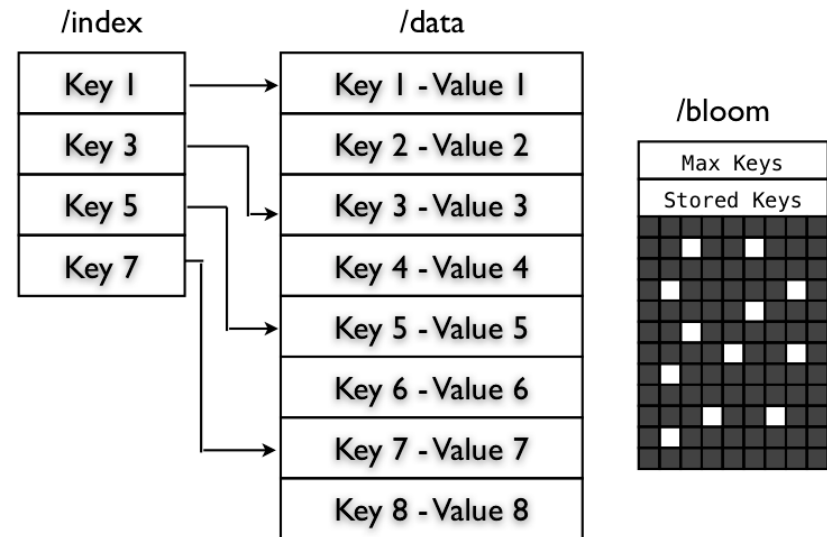
Example : Search key - 496

Last-Key <= 496 → index key found is - 385 with the value of 18030 (offset in the data file)

Reader seeks from the offset until it reaches to search Key,496

Advantage :

GOOD for Random access



Counters

- ✓ Counters are lightweight objects in Hadoop that allow you to keep track of System progress during Map and Reduce phase
- ✓ By Default, Hadoop defines a number of standard counters in “groups”
- ✓ These show up information such as “Map input records”, “Map Output records”

	Counter	Map	Reduce	Total
File Input Format Counters	Bytes Read	0	0	0
Job Counters	SLOTS_MILLIS_MAPS	0	0	56,303
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	2
	SLOTS_MILLIS_REDUCE	0	0	0
File Output Format Counters	Bytes Written	0	0	1,000,000,000
FileSystemCounters	HDFS_BYTES_READ	167	0	167
	FILE_BYTES_WRITTEN	108,282	0	108,282
	HDFS_BYTES_WRITTEN	1,000,000,000	0	1,000,000,000
Map-Reduce Framework	Map input records	0	0	10,000,000
	Physical memory (bytes) snapshot	0	0	290,058,240
	Spilled Records	0	0	0
	Total committed heap usage (bytes)	0	0	76,480,512
	CPU time spent (ms)	0	0	16,240
	Map input bytes	0	0	10,000,000
	Virtual memory (bytes) snapshot	0	0	1,034,915,840
	SPLIT_RAW_BYTES	167	0	167
	Map output records	0	0	10,000,000

Counters...

Counters can be used to find out good records or Bad records. For ex: in order data : In some records, product details are missing or amount is coming as zero etc






Distributed Cache

- During complex MR Job, Some program might need some data to be referred to complete the job like dependency(set-of-small-data)/configuration/property
- For Ex : Distributed application, would require a list of current Dollar values for all the tasks running on the cluster OR
- While Processing salaries, grade details are also required...
- When we want to distribute some common data across all task trackers, we can go for Distributed Cache
- The Distributed Cache can contain small data files needed for initialization or libraries of code that may be needed to be accessed on all nodes in the cluster

Map Reduce Unit Testing – MRUnit

1. You need to download apache-mrunit-1.0.0-hadoop1 which contains the MR-Unit testing api.

<http://psg.mtu.edu/pub/apache/mrunit/mrunit-1.0.0/>

 commons-logging-1.1.1	10/17/2012 3:03 PM	Executable Jar File	60 KB
 hamcrest-core-1.1	10/17/2012 3:03 PM	Executable Jar File	75 KB
 junit-4.10	10/17/2012 3:03 PM	Executable Jar File	248 KB
 mockito-all-1.8.5	10/17/2012 3:03 PM	Executable Jar File	1,387 KB
 mrunit-1.0.0-hadoop1	4/3/2013 3:31 AM	Executable Jar File	100 KB

2. Import all jars from the lib folder of the downloaded file into eclipse project.
3. select all jar to add them to your testing project

Testing Mappers

1. Instantiate an instance of the MapDriver class parameterized exactly as the mapper under test.
2. Add an instance of the Mapper you are testing in the withMapper call
3. In the *.withInput* call pass in your key and input value, here LongWritable with an arbitrary value and a Text object that contains a line “First day,Sunday,abhay,holiday”.
4. Specify the expected output in the withOutput call, here we are expecting a Text object as the key with value as “Sunday” and the count of it as the value as “1”.
5. The last call runTest feeds the specified input values into the mapper and compares the actual output against the expected output set in the ‘withOutput’ method.

MR Unit Testing

```
import java.io.IOException;
import junit.framework.TestCase;
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 org.apache.hadoop.mapreduce.*;
import org.junit.Test;

public class TestExample extends TestCase {

    public static class maptest extends
        Mapper<LongWritable, Text, Text, IntWritable>{

        Text day=new Text();
        public void map(LongWritable key,Text value,Context ct)
            throws IOException,InterruptedException
        {
            String[] line=value.toString().split(",");
            int val=Integer.parseInt(line[0]);

            if (val==1)
            {
                day.set(line[1]);
                ct.write(day,new
                    IntWritable(val));
            }
        }
    }

    MapDriver<LongWritable, Text, Text, IntWritable>
    mapDriver;

    public void setUp() {
        new maptest();
        mapDriver = MapDriver.newMapDriver( new maptest());
    }
```

```
@Test
public void testMapper() {
    try {
        mapDriver.withInput(new LongWritable(), new
            Text("1,sunday,abhay,holiday"))
            .withOutput(new Text("sunday"), new
                IntWritable(1))

        .runTest();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

MapDriver : Allows you to test a Mapper instance. You provide the input (k, v)* pairs that should be sent to the Mapper, and outputs you expect to be sent by the Mapper to the collector for those inputs.

By calling runTest(), MapDriver will deliver the input to the Mapper and will check its outputs against the expected results.

Note: The MapDriver only allows one input and output per test. You can call withInput and withOutput multiple times if you want, but the MapDriver will overwrite the existing values with the new ones, so you will only ever be testing with one input/output at any time.