Big Data Analysis Platforms

SHYI-CHYI CHENG

Slides credited to Cloudera Developer Training for Apache Hadoop

Outline

- Review of Virtual Machine (虛擬機器回顧)
- Hadoop Platform (運算分析系統架構)
- MapReduce
- Introduction to Python (Python入門簡介)
- Python Spark Platform (Python Spark運算分析架構)
- Parallel Programming With Spark

MapReduce

Original Slides by
Owen O'Malley (Yahoo!)
&
Cloudera Developer Training for Apache Hadoop

Introduction to MapReduce

- What you will learn
  - The concepts behind MapReduce
  - How data flows through MapReduce stages
  - Typical uses of Mappers
  - Typical uses of Reducers

MapReduce - What?

- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

MapReduce Features (1)

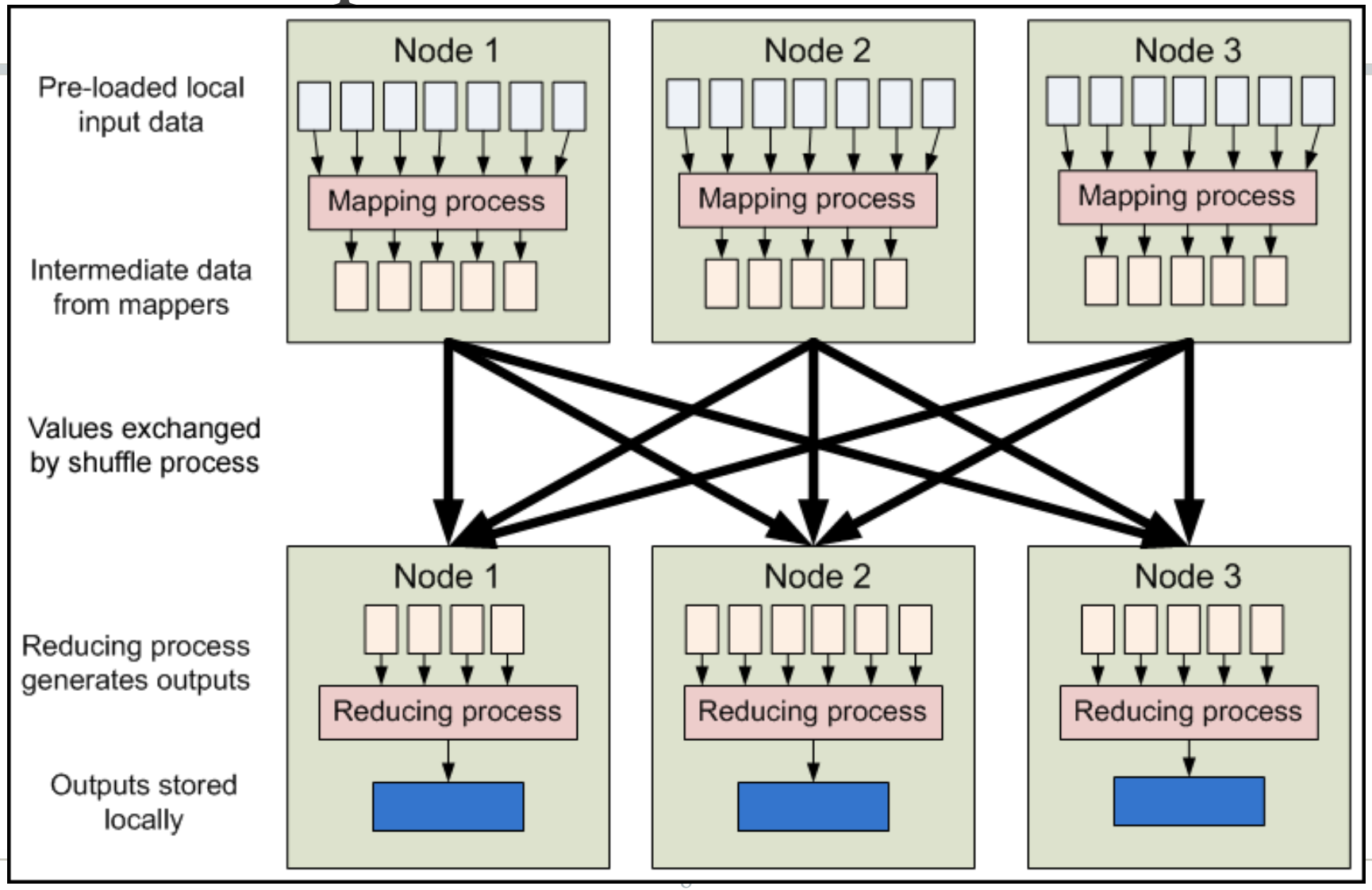
- Automatic parallelization and distribution
- Fault-tolerance
- A clean abstraction for programmers
  - MapReduce programs are usually written in Java
    - Can be written in any language using Hadoop streaming
    - All of Hadoop is written in Java
- MapReduce abstracts all the "housekeeping" away from the developer
  - Developer can simply concentrate on writing the Map and Reduce functions

Hadoop Streaming is a utility which allows users to create and run jobs with any executables (e.g. shell utilities) as the mapper and/or the reducer.

MapReduce Features (2)

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

MapReduce - Dataflow

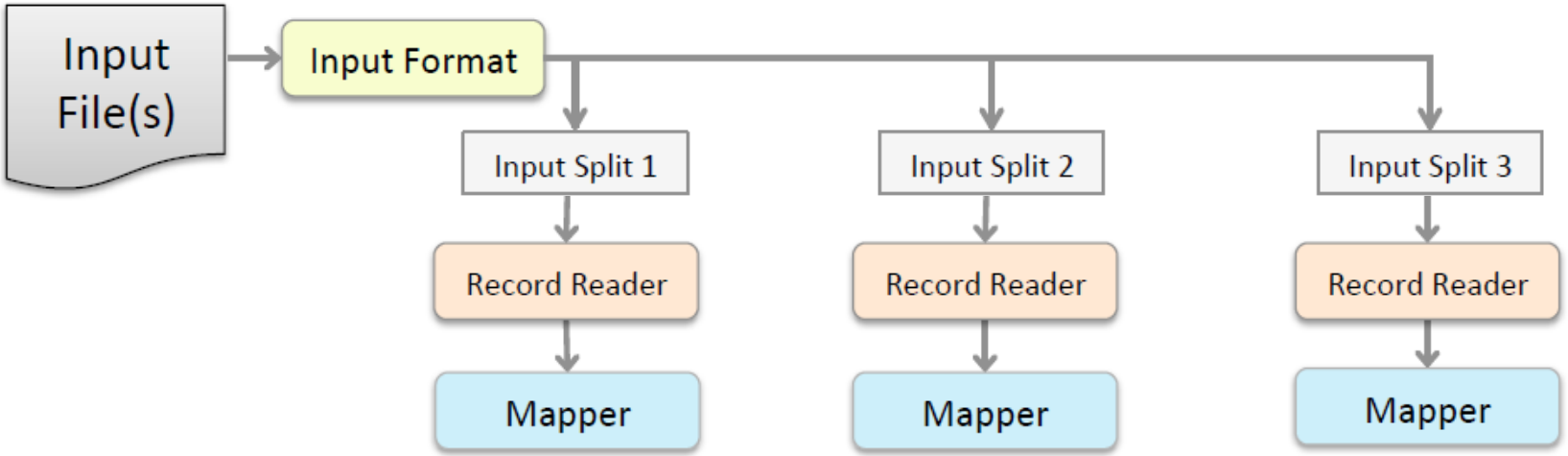


Key MapReduce Stages

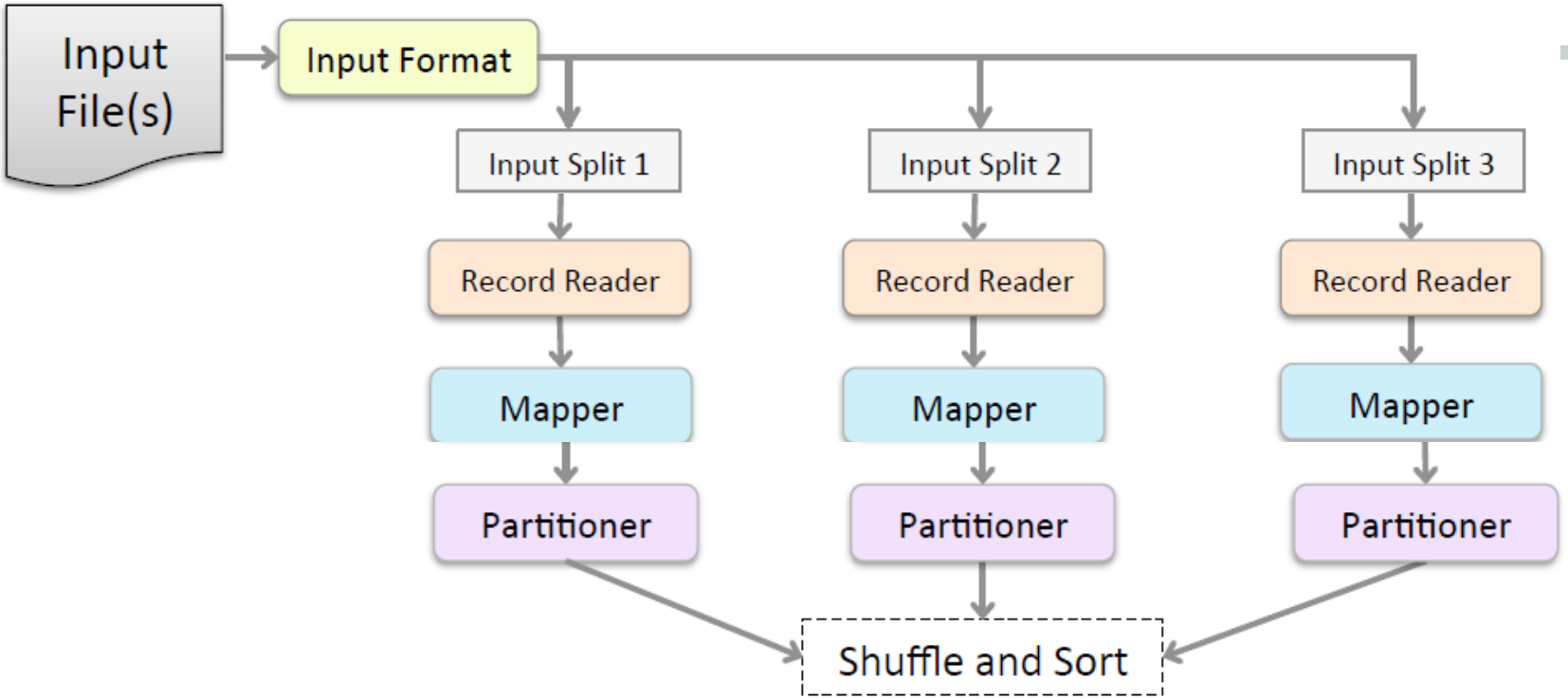
- The Mapper
  - Each Map task (typically) operates on a single HDFS block
  - Map tasks (usually) run on the node where the block is stored
- Shuffle and Sort
  - Sorts and consolidates intermediate data from all mappers
  - Happens after all Map tasks are complete and before Reduce tasks start
- The Reducer
  - Operates on shuffled/sorted intermediate data (Map task output)
  - Produces the final output

The MapReduce Flow

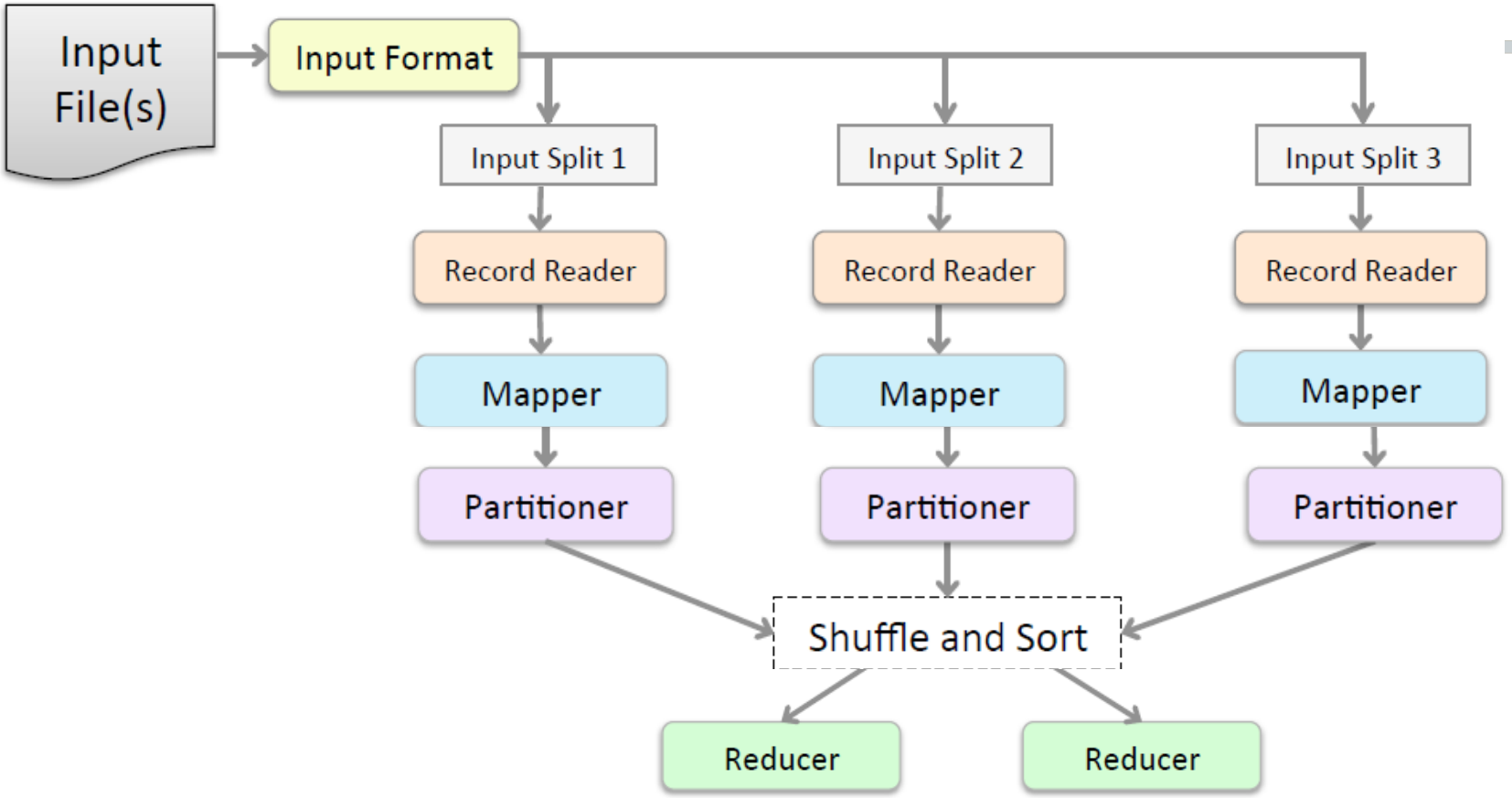
The MapReduce Flow



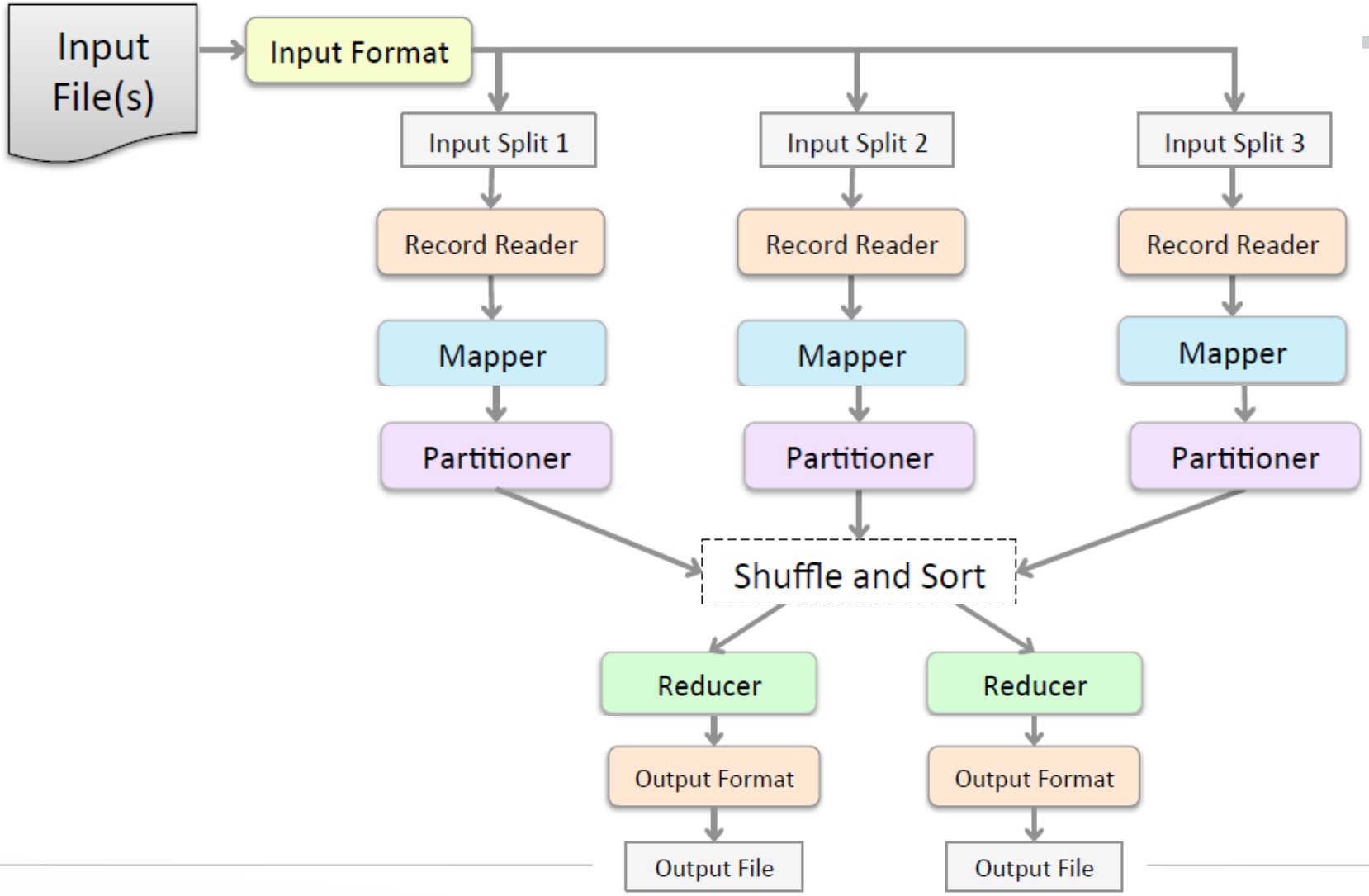
The MapReduce Flow



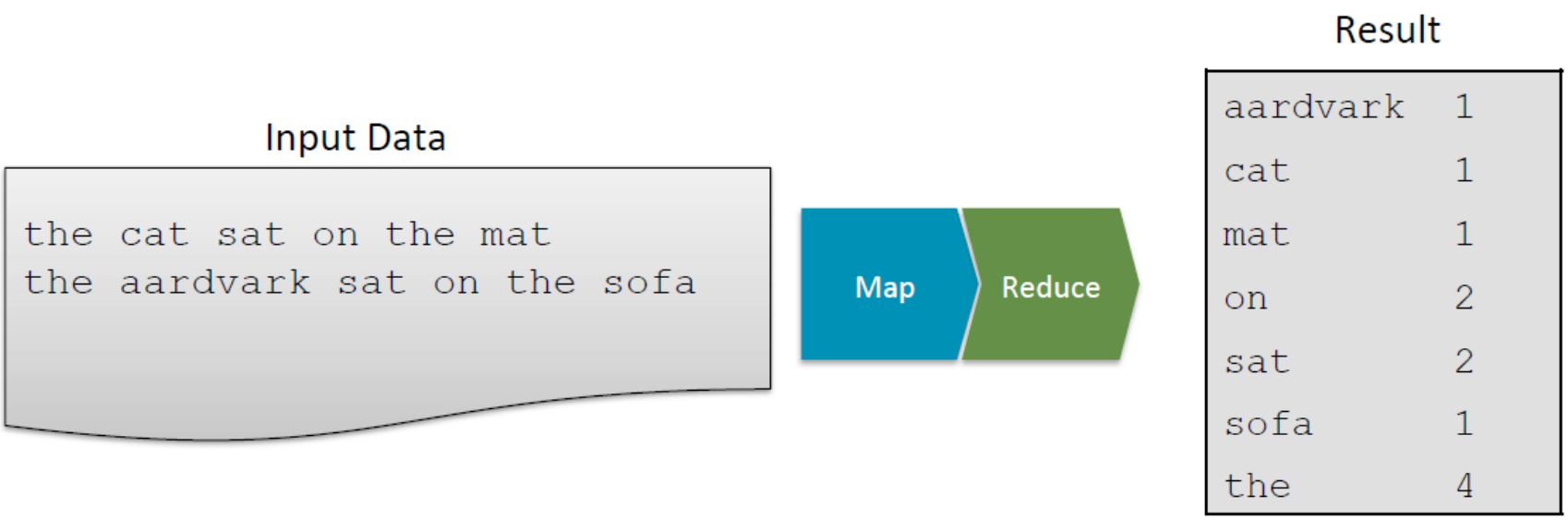
The MapReduce Flow



The MapReduce Flow



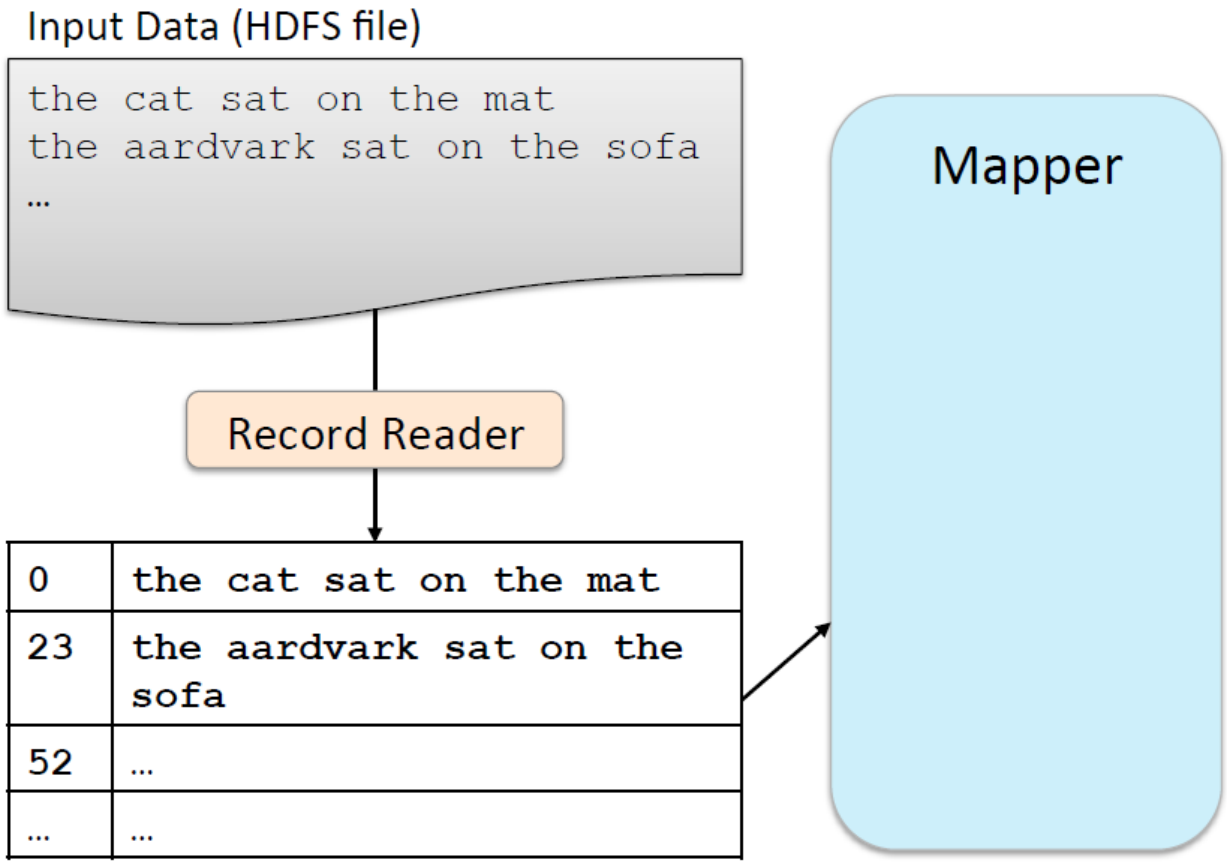
Word Count Example (1)



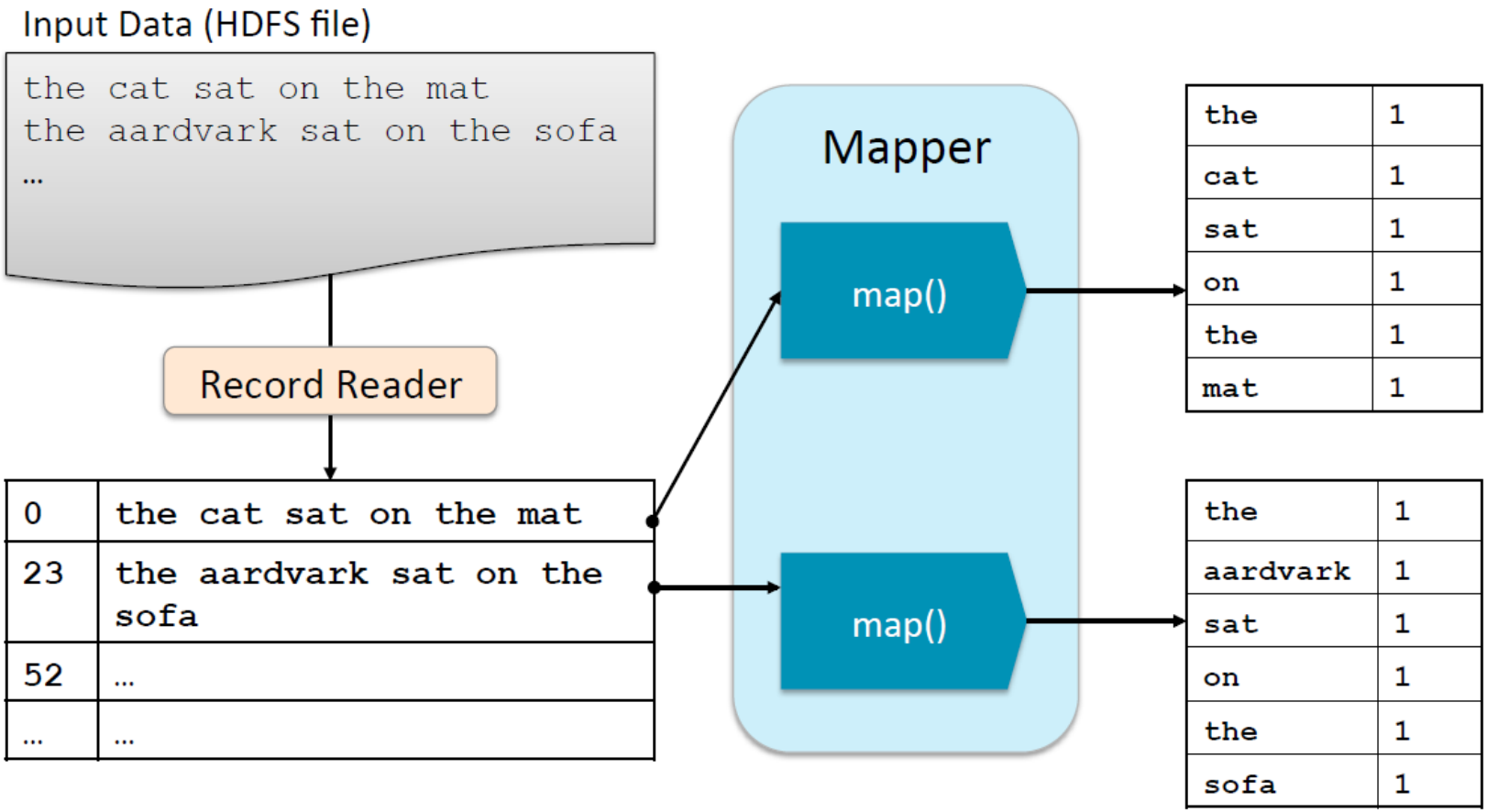
Word Count Example (2)

- Mapper
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- Reducer
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- Launching program
  - Defines this job
  - Submits job to cluster

Word Count Example (3)

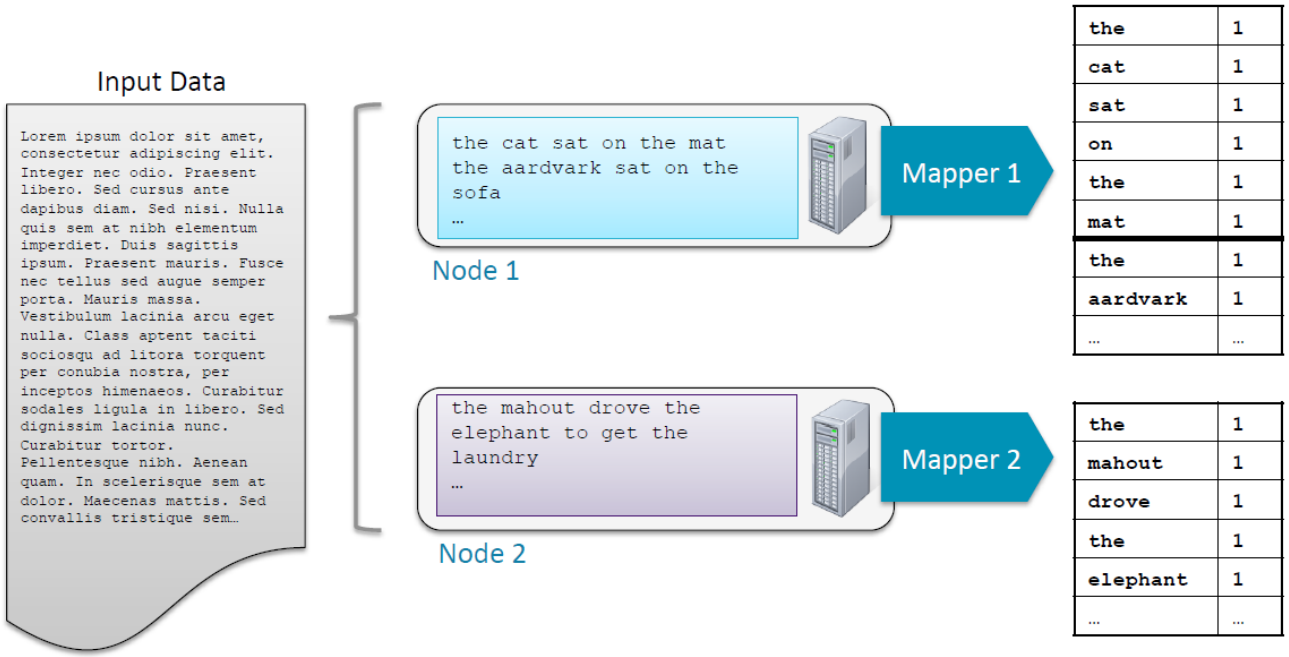


Word Count Example (4)

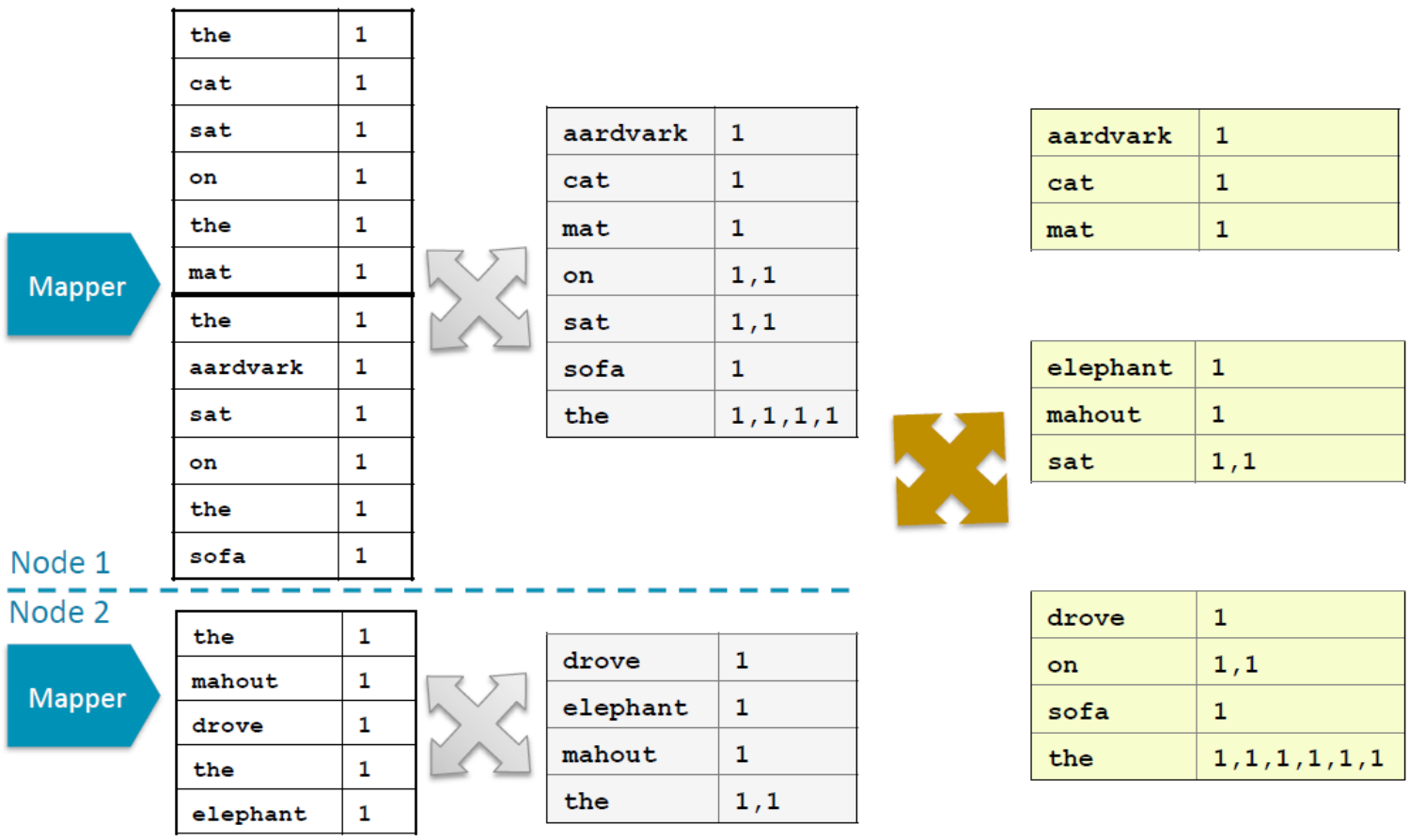


Word Count Example (5)

- Hadoop runs Map tasks on the node storing the data (when possible)
  - Minimizes network traffic
  - Many Mappers can run in parallel



Shuffle and Sort



the	1
cat	1
sat	1
on	1
the	1
mat	1

the	1
aardvark	1
sat	1
on	1
the	1
sofa	1

aardvark	1
cat	1
mat	1
on	1,1
sat	1,1
sofa	1
the	1,1,1,1

aardvark	1
cat	1
mat	1

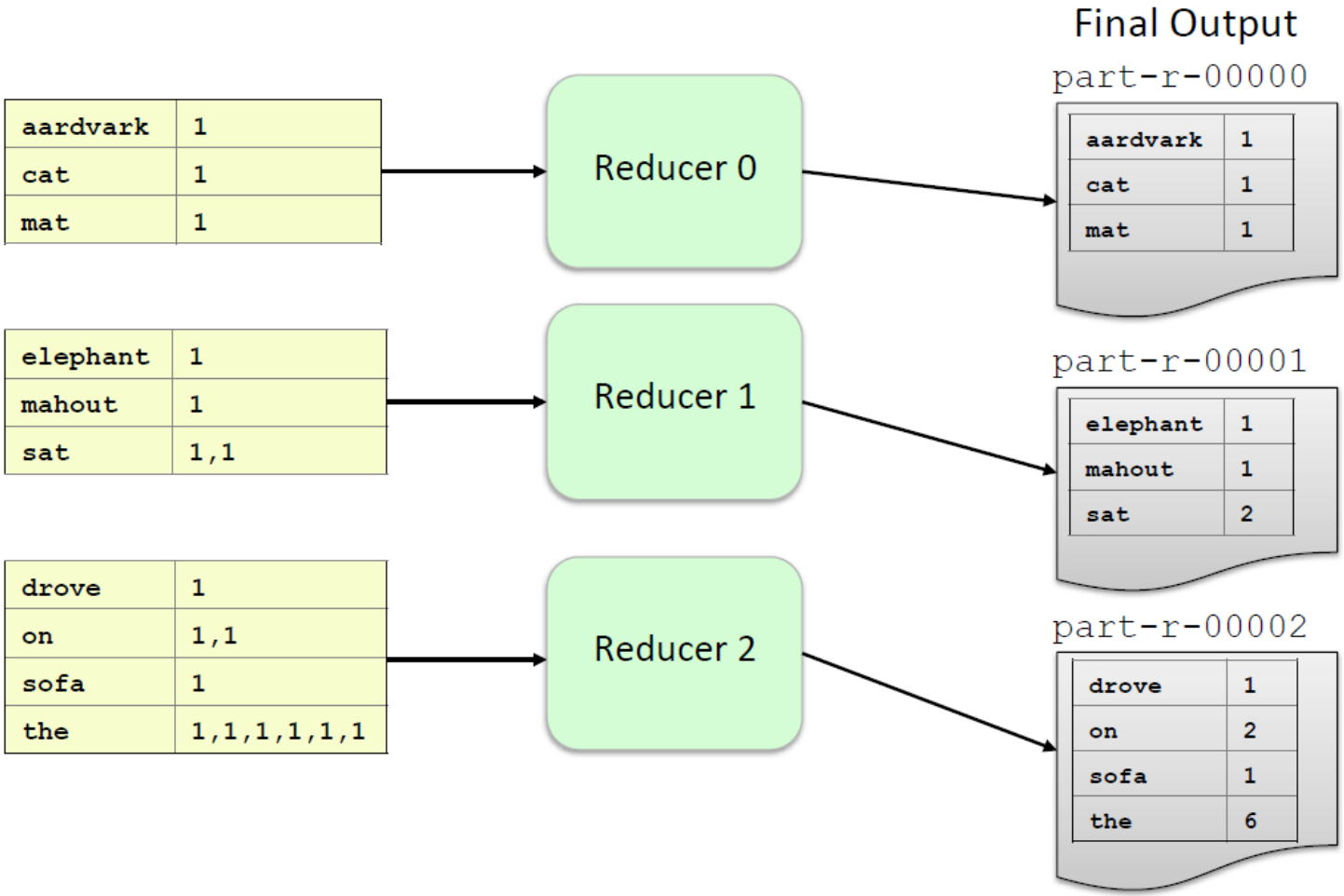
elephant	1
mahout	1
sat	1,1

the	1
mahout	1
drove	1
the	1
elephant	1

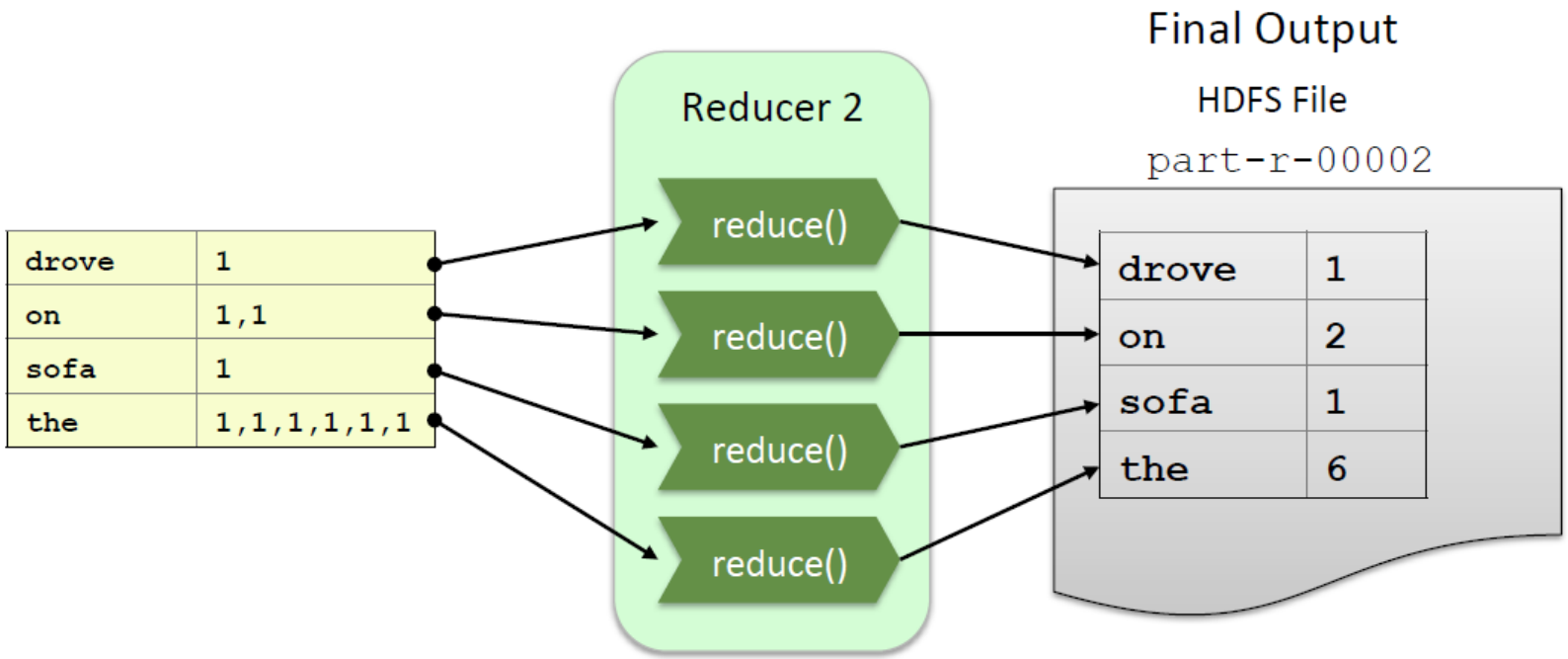
drove	1
elephant	1
mahout	1
the	1,1

drove	1
on	1,1
sofa	1
the	1,1,1,1,1,1

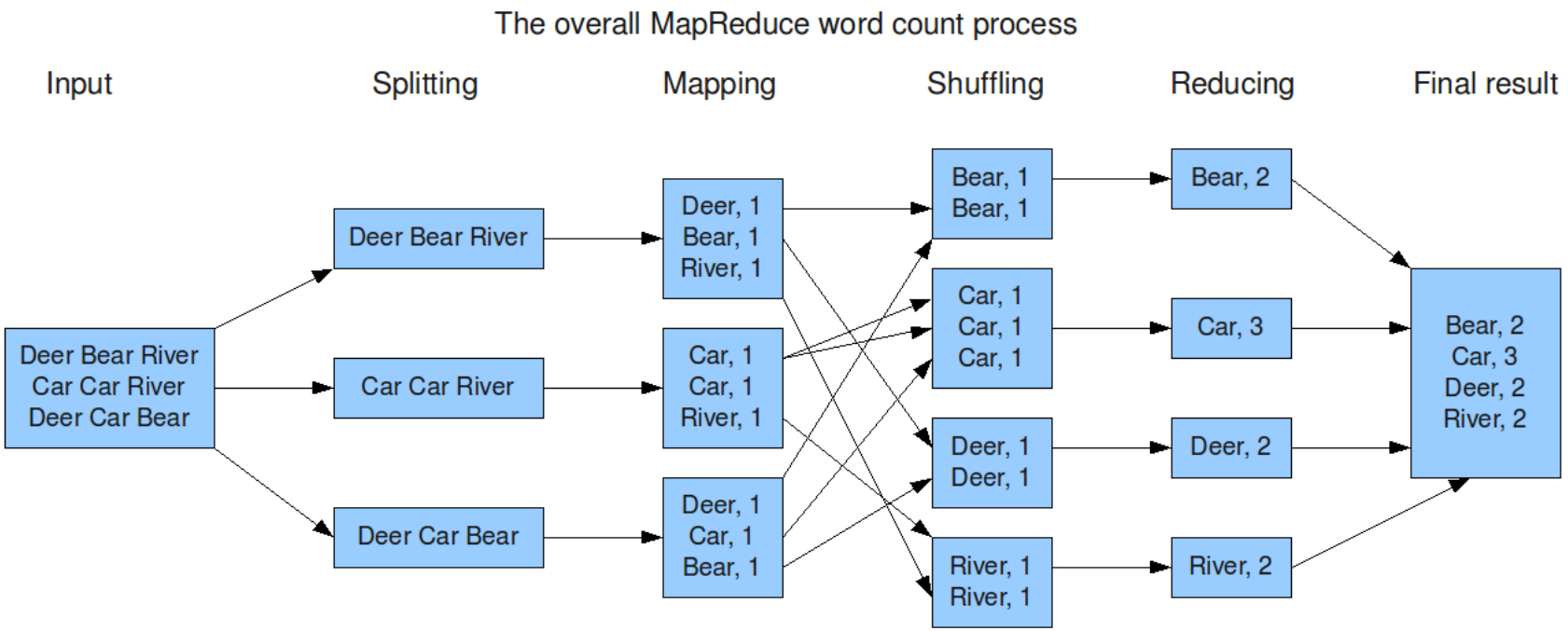
Example: SumReducer (1)



Example: SumReducer (2)



The Final Word Count Dataflow



Another Example

Input Data

```
...
2013-03-15 12:39 - 74.125.226.230 /common/logo.gif 1231ms - 2326
2013-03-15 12:39 - 157.166.255.18 /catalog/cat1.html 891ms - 1211
2013-03-15 12:40 - 65.50.196.141 /common/logo.gif 1992ms - 1198
2013-03-15 12:41 - 64.69.4.150 /common/promoex.jpg 3992ms - 2326
...
```

Map

FileTypeMapper output

...	...
gif	1231
html	891
gif	1992
jpg	3992
html	788
gif	3997
...	...

Intermediate Data after Shuffle and Sort

html	891,788,344,2990...
gif	1231,1992,3997,872...
jpg	3992,7881,2999...
png	919,890,3441,444...
txt	344,325,444,421...

Reduce

AverageReducer output

html	888.6
gif	1886.4
jpg	888.6
png	1201.0
txt	399.1

Word Count Mapper

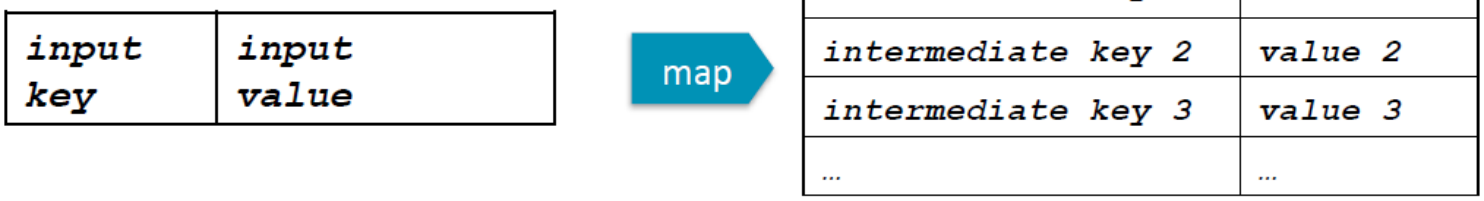
```
public static class Map extends MapReduceBase implements
    Mapper<LongWritable,Text,Text,IntWritable> {
  private static final IntWritable one = new IntWritable(1);
  private Text word  = new Text();

  public static void map(LongWritable key, Text value,
    OutputCollector<Text,IntWritable> output, Reporter reporter) throws
    IOException {
     String line = value.toString();
     StringTokenizer = new StringTokenizer(line);
     while(tokenizer.hasNext()) {
        word.set(tokenizer.nextToken());
        output.collect(word,one);
        }
     }
}
```

MapReduce: The Mapper (1)

- The Mapper
  - Input: key/value pair
  - Output: A list of zero or more key value pairs

```
map(in_key, in_value) →
                        (inter_key, inter_value) list
```



MapReduce: The Mapper (2)

- Input: one line of a file at a time in key/value format
  - The key is the byte offset into the file at which the line starts
  - The value is the contents of the line itself
- Output: in the forms of key/value pairs
  - Input: key/value pair
  - Output: A lit of zero or more key value pairs

23	the aardvark sat on the sofa

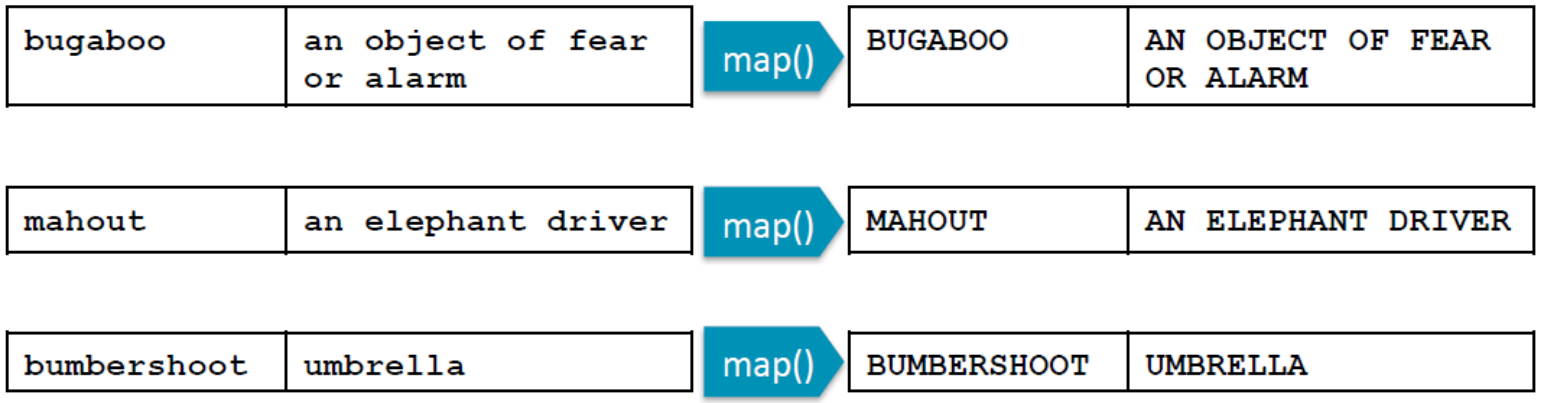
map

the	1
aardvark	1
sat	1
on	1
the	1
sofa	1

Example Mapper: Upper Case Mapper

- Turn input into upper case (pseudo-code)

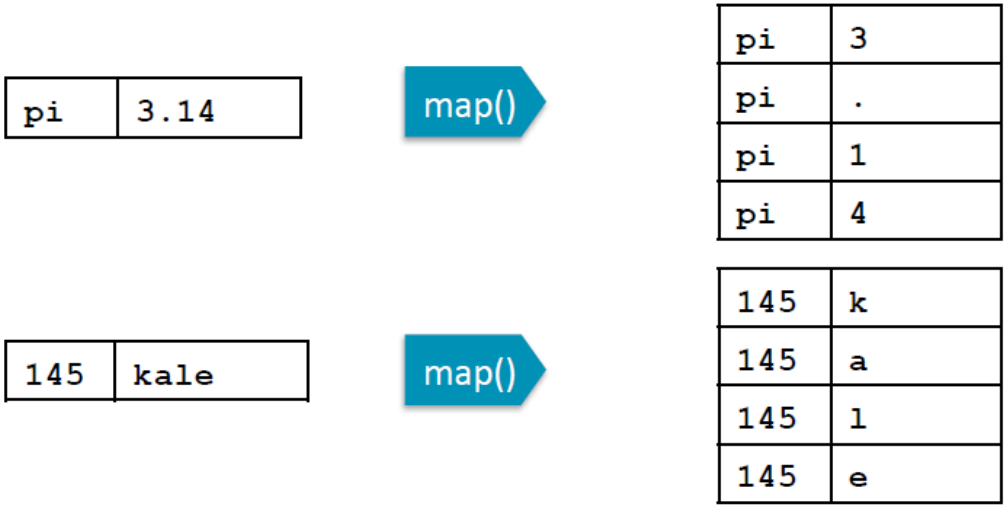
```
let map(k, v) =
    emit(k.toUpper(), v.toUpper())
```



Example Mapper: 'Explode' Mapper

- Output each input character separately (pseudo-code)

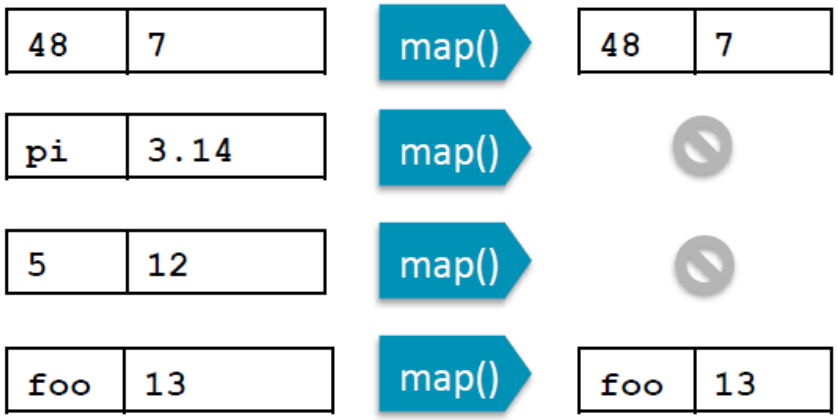
```
let map(k, v) =
    foreach char c in v:
        emit (k, c)
```



Example Mapper: 'Filter' Mapper

- Only output key/value pairs where the input value is a prime number (pseudo-code)

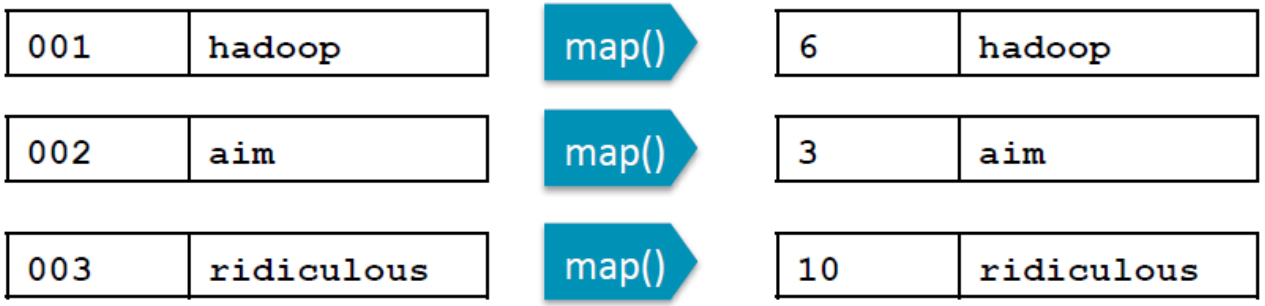
```
let map(k, v) =
    if (isPrime(v)) then emit(k, v)
```



Example Mapper: Changing Keyspaces

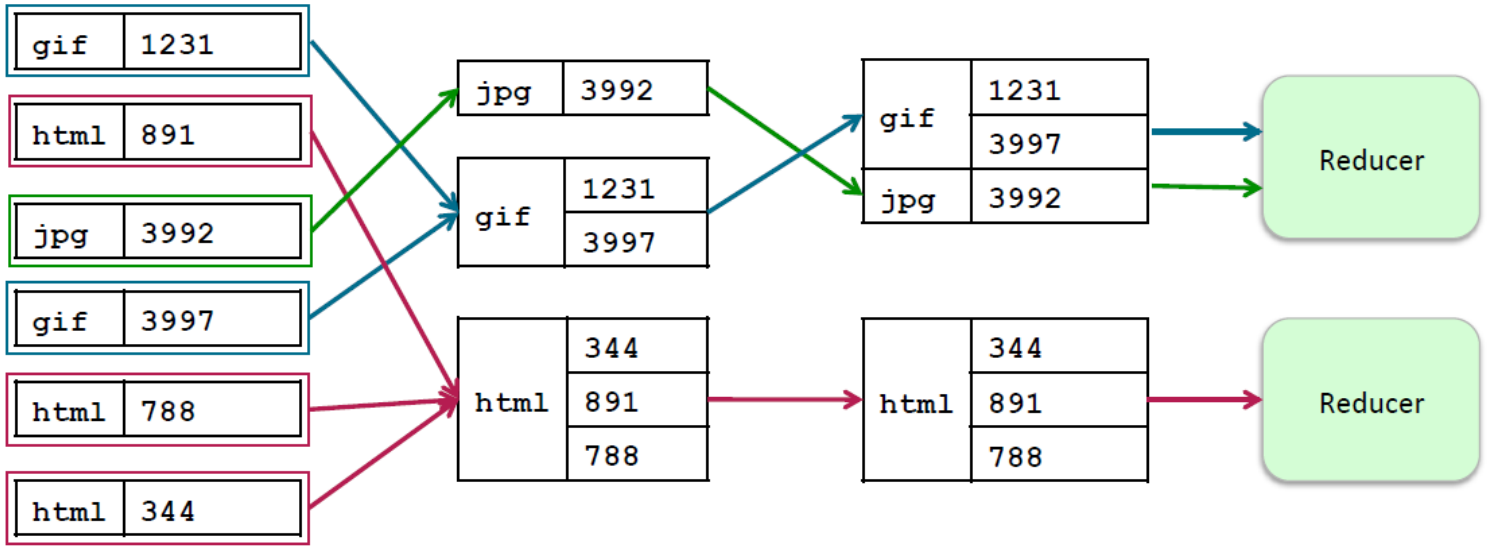
- Output the word length as the key (pseudo-code)

```
let map(k, v) =
    emit(v.length(), v)
```



Shuffle and Sort

- After the Map phase is over, all intermediate values for a given intermediate key are grouped together
- Each key and value list is passed to Reducer
  - All values for a particular intermediate key go to the same Reducer
  - The intermediate key/value lists are passed in sorted key order



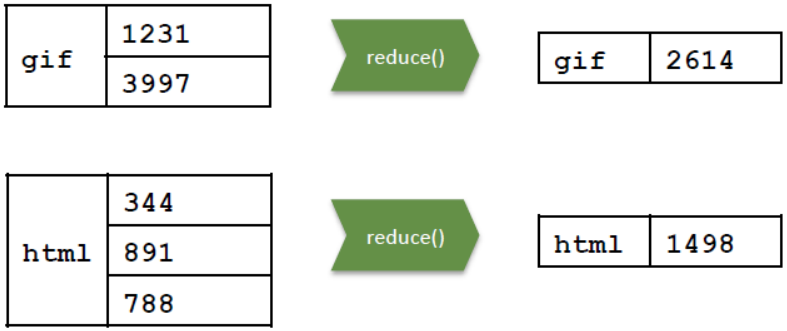
Word Count Reducer

```
public static class Reduce extends MapReduceBase implements
    Reducer<Text,IntWritable,Text,IntWritable> {
public static void map(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));
    }
  }
```

The Reducer

- The Reducer outputs zero or more final key/value pairs
  - In practice, usually emits a single key/value pair for each input key
  - These are written into HDFS

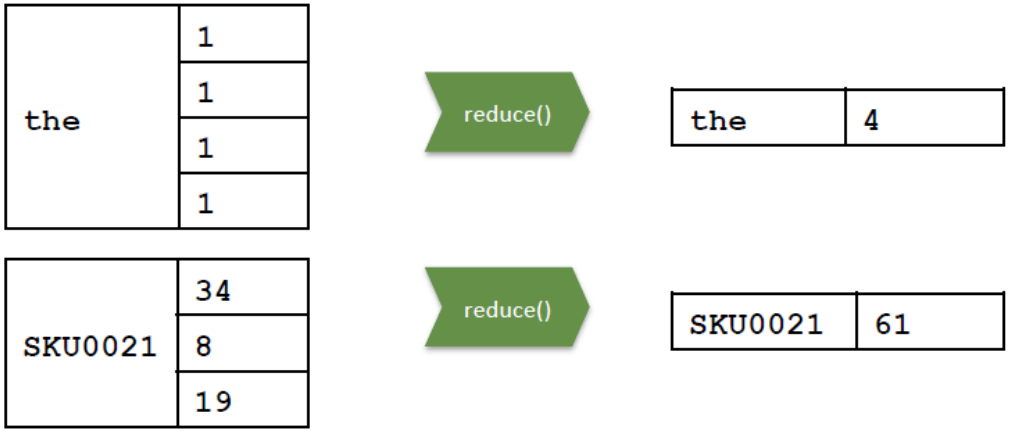
```
reduce(inter_key, [v1, v2, …]) →
                        (result_key, result_value)
```



Example Reducer: Sum Reducer

- All up all the values associated with each intermediate key (pseudo-code)

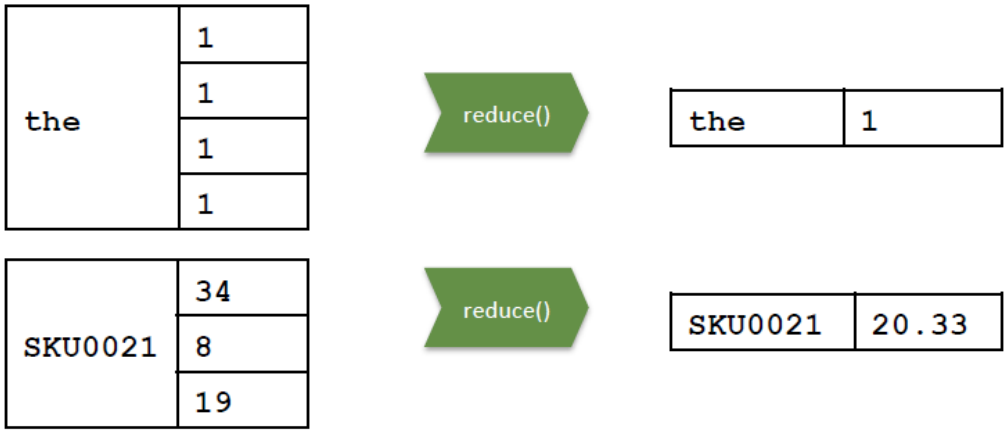
```
let reduce(k, vals) =
    sum = 0
    foreach int i in vals:
        sum += i
    emit(k, sum)
```



Example Reducer: Average Reducer

- Find the mean of all the values associated with each intermediate key (pseudo-code)

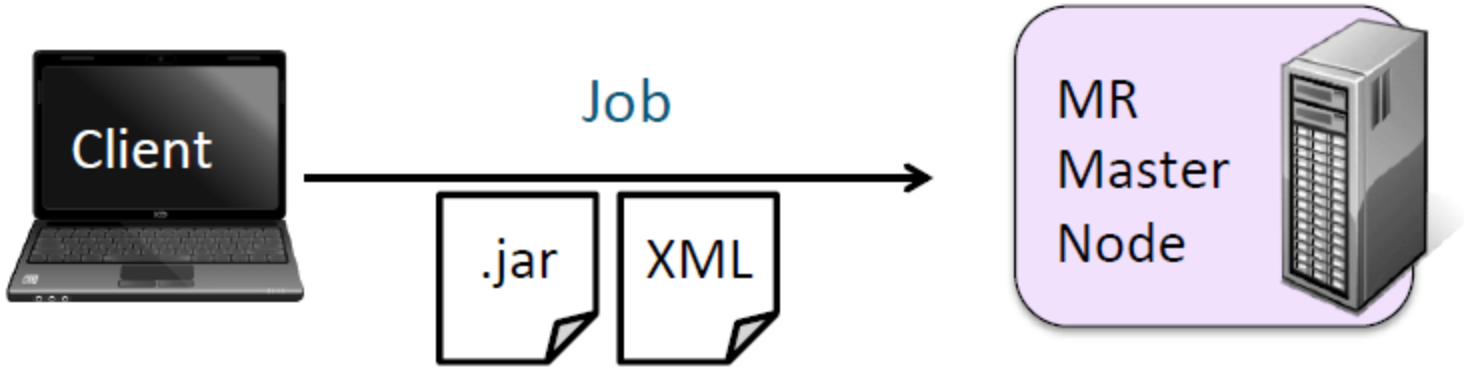
```
let reduce(k, vals) =
    sum = 0; counter = 0;
    foreach int i in vals:
        sum += i; counter += 1;
    emit(k, sum/counter)
```



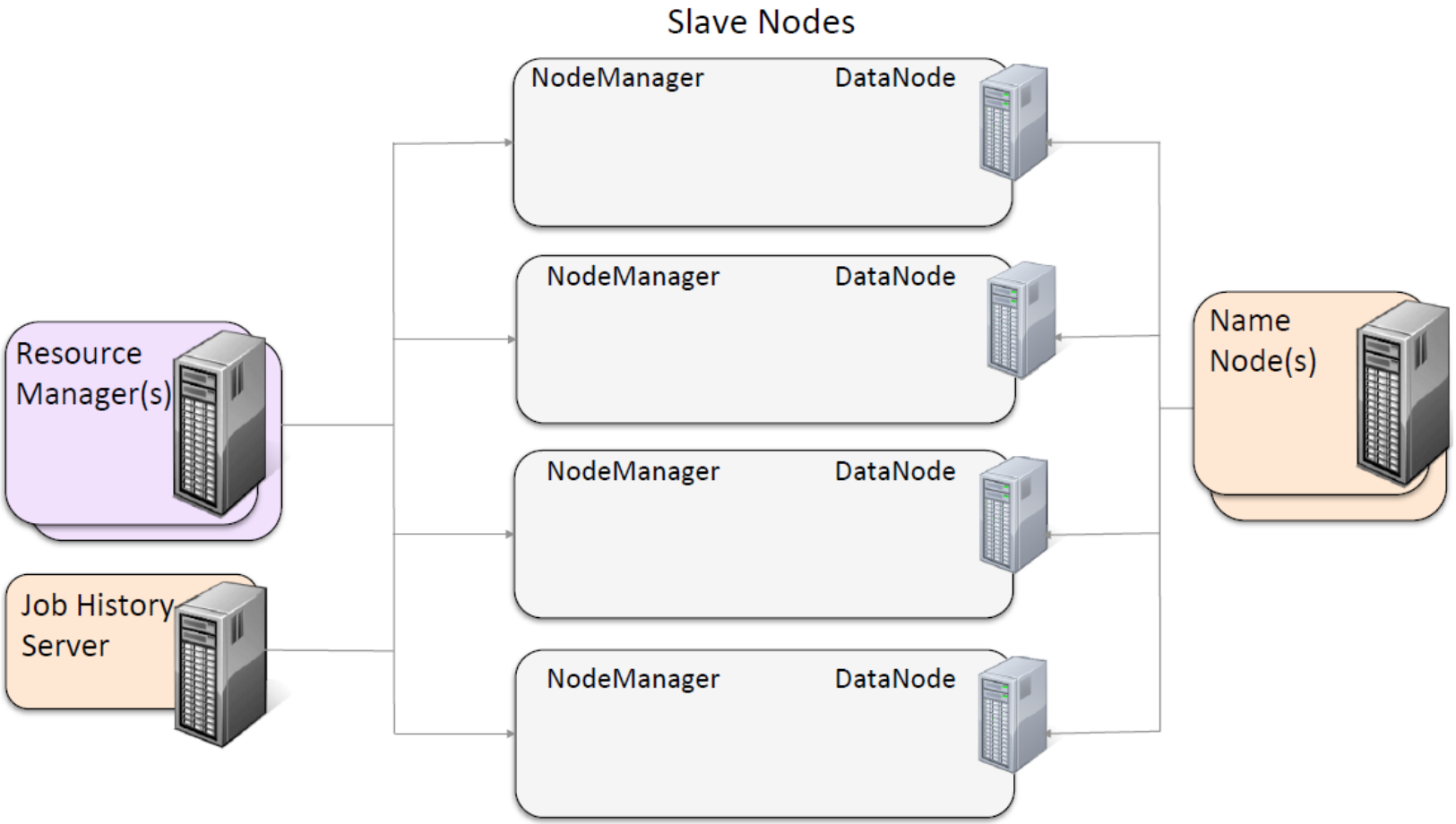
Hadoop Jobs and Tasks

- A job is a 'full program'
  - A complete execution of Mappers and Reducers over a dataset
- A task is the execution of a single Mapper or Reducer over a slice of data
- A task attempt is a particular instance of an attempt to execute a task
  - Number of task attempt >= number of tasks
  - If a task attempt fails, another will be started by the JobTracker or ApplicationMaster
  - Speculative execution can also result in more task attempts than completed tasks

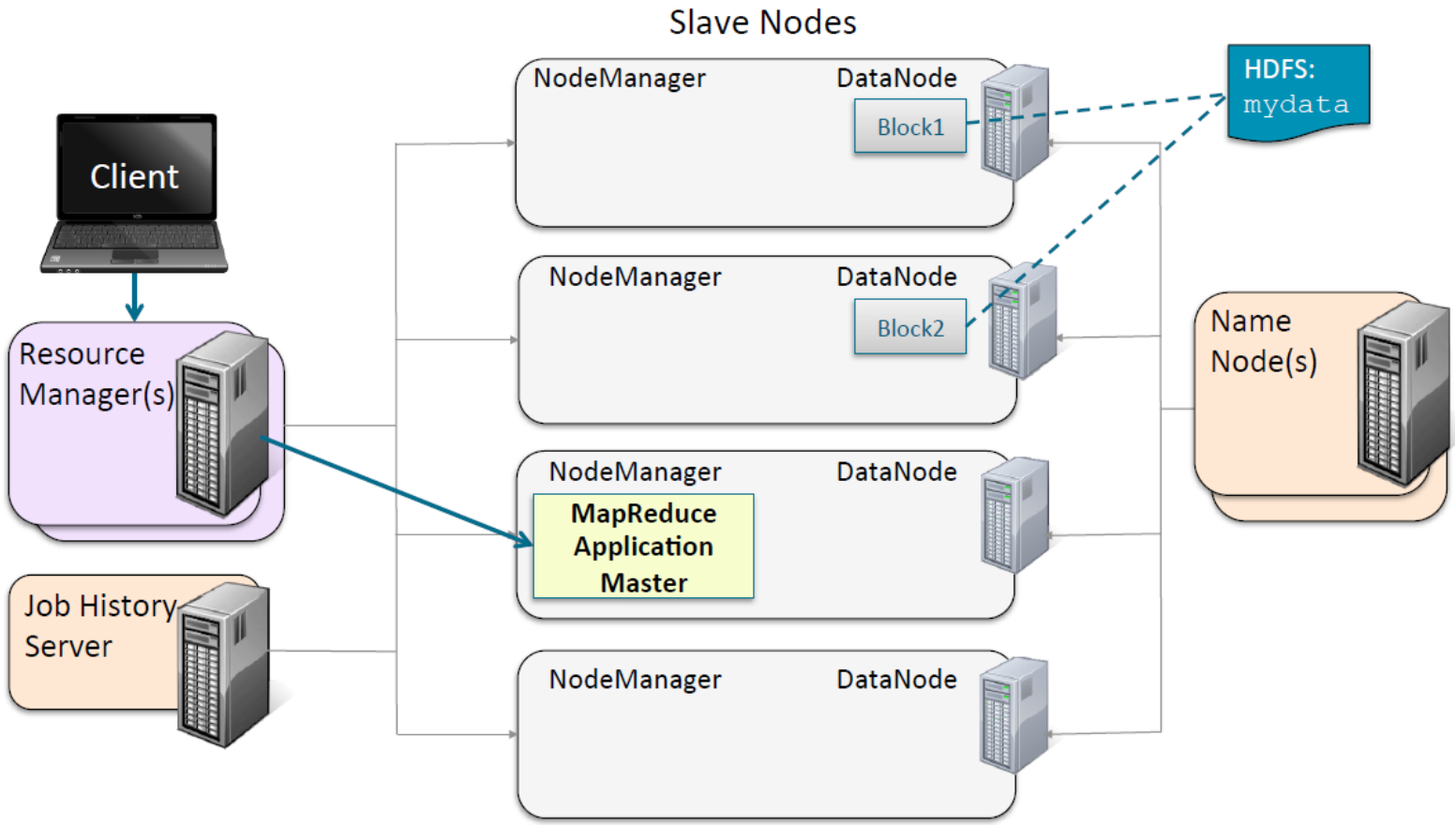
Submitting A Job



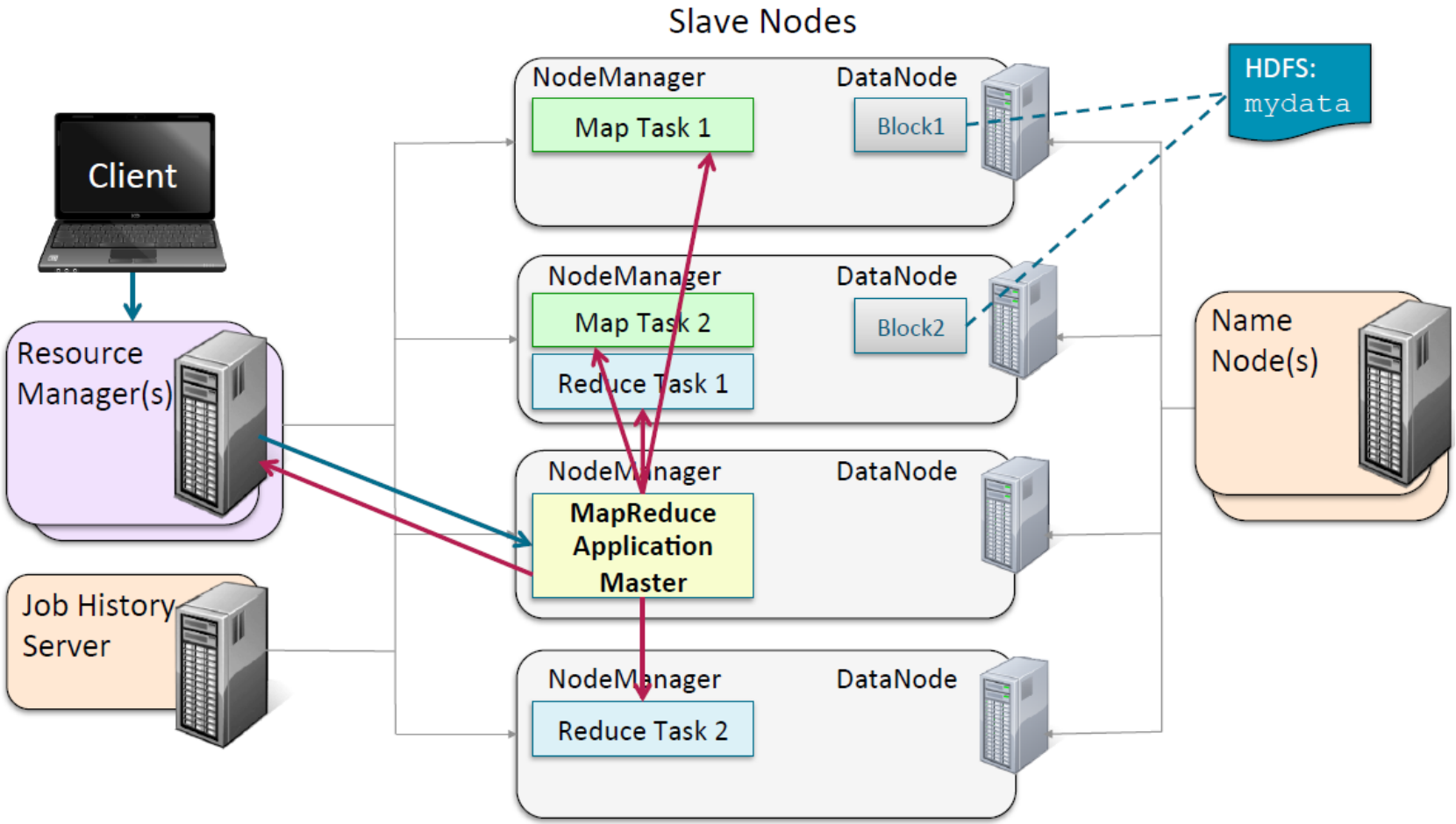
A MapReduce v2 Cluster



Running a Job on a MapReduce v2 Cluster (1)

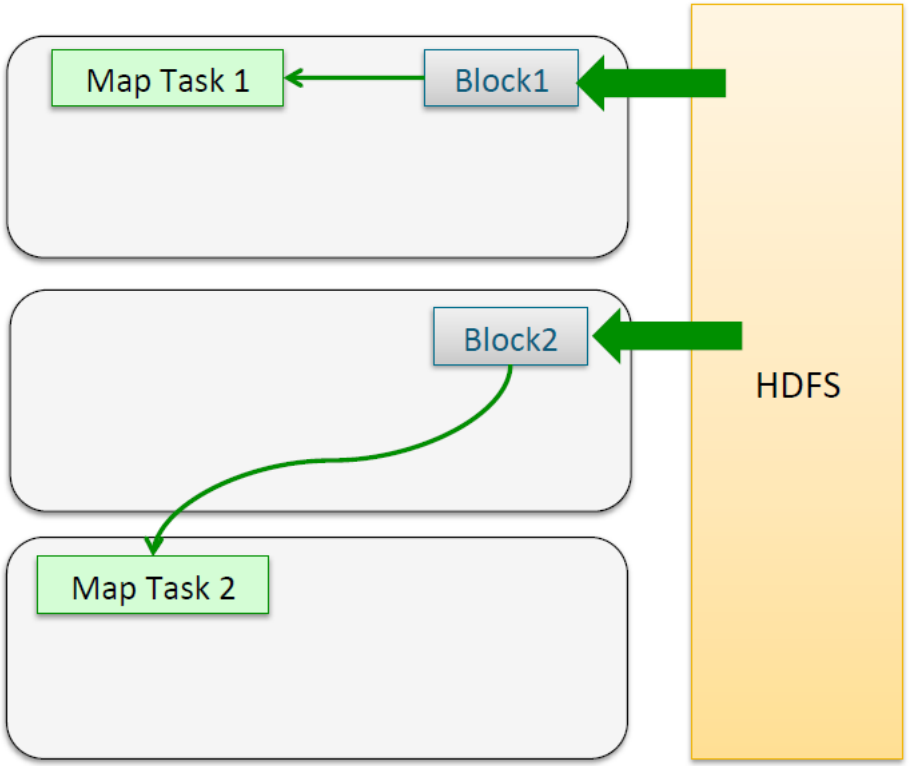


Running a Job on a MapReduce v2 Cluster (2)



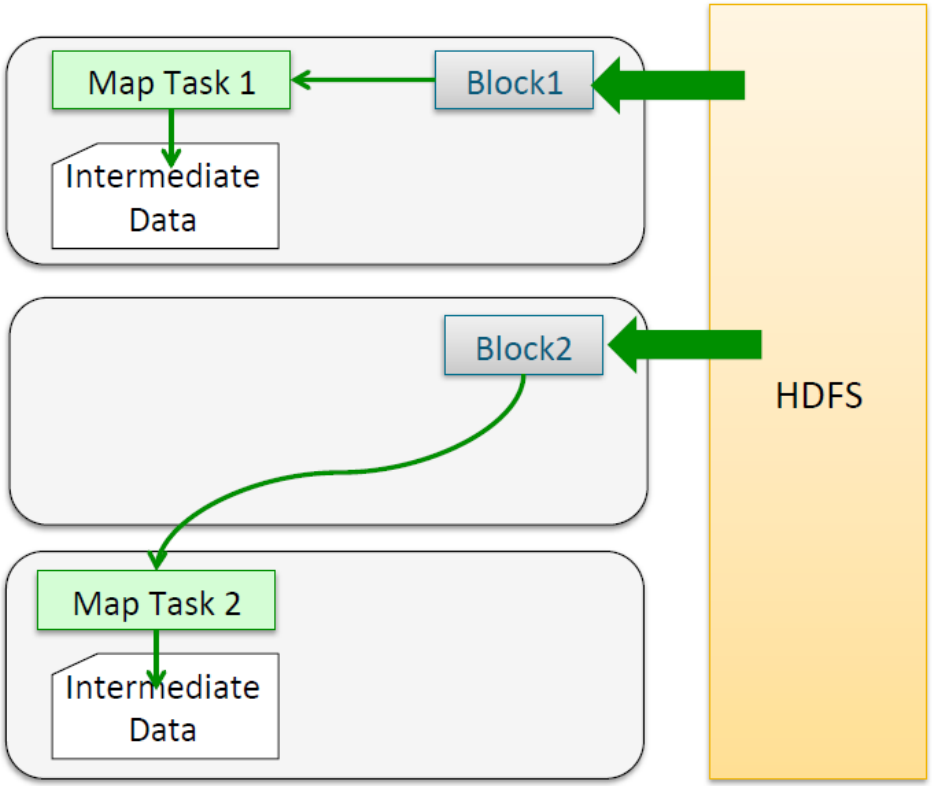
Job Data: Mapper data Locality

- When possible, Map tasks run on a node where a block of data is stored locally
- Otherwise, the Map tasks will transfer the data across the network as it possesses that data



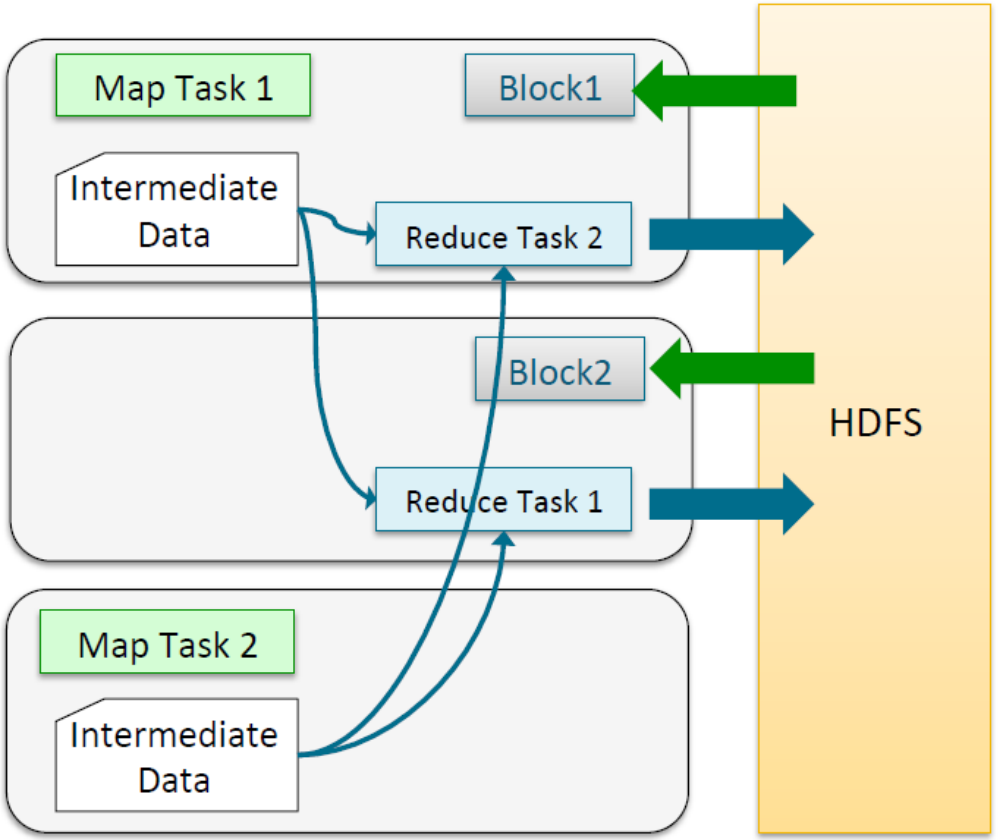
Job Data: Intermediate Data

- The intermediate data of Map tasks is stored on the local disk (not HDFS)



Job Data: Shuffle and Sort

- Intermediate data is transferred across the network to the Reducers
- Reducers write their output to HDFS



Why Shuffle and Sort is a Bottleneck?

- The reduce method in the Reducers cannot start until all Mappers have finished
- In practice, Hadoop will start to transfer and sort data from the Mappers to Reducers as soon as the Mappers finish work
  - This improves the bottleneck a little

Is a Slow Mapper a Bottleneck?

- Faulty hardware or slow machine might degrade the performance
- Hadoop uses speculative execution to mitigate against this by creating a new instance of the Mapper to replace the old one, operating on the same data

Creating and Running a MapReduce Job

- Write the Mapper and Reducers classes
- Write a Driver class that configures the job and submits to the cluster (discussed later)
- Compile the Mapper, Reducer, and Driver classes

```
$ javac -classpath `hadoop classpath` MyMapper.java
  MyReducer.java MyDriver.java
```

- Create a jar file with the Mapper, Reducer, and Driver classes

```
$ jar cf MyMR.jar MyMapper.class MyReducer.class
  MyDriver.class
```

- Run the Hadoop jar command to submit the job to the Hadoop cluster

```
$ hadoop jar MyMR.jar MyDriver in_file out_dir
```

To Write a MapReduce Program in Java

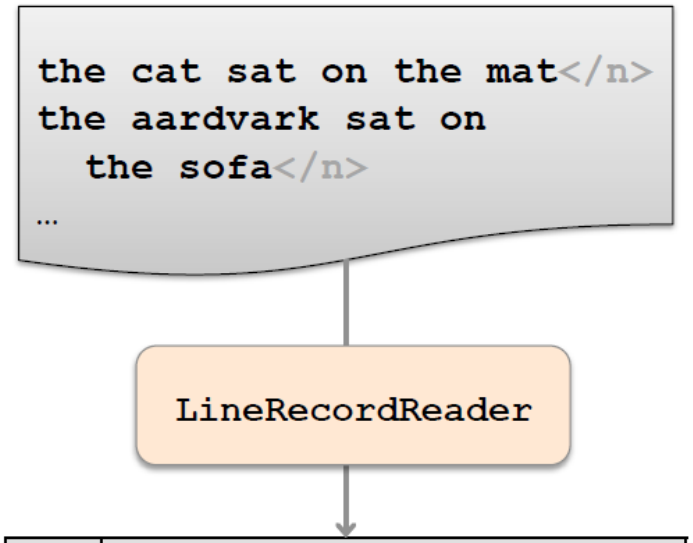
- The driver code
  - Code that runs on the client to configure and submit the job
- The Mapper
- The Reducer

Getting Data to the Mapper

- The data passed to the Mapper is specified by an InputFormat
  - Specified in the driver code
  - Defines the location of the input data, a file or a dictionary
  - Determines how to split the input data into input splits
    - Each mapper deals with a single input split
  - Creates a RecordReader object
    - RecordReader passes the input data into key/value pairs to pass to the Mapper

Example: TextInputFormat

- TextInputFormat
  - The default
  - Creates LineRecordReader objects
  - Treats each \n-terminated line of a file as a value
  - Key is the byte offset of that line within a file



key	value
0	the cat sat on the mat
23	the aardvark sat on the sofa
52	…
…	…

Other Standard InputFormat

- FileInputFormat
  - Abstract base class for all file-based InputFormats
- KetValueInputFormat
  - Maps \n-terminated lines as 'key [separator] value'
    - By default, [separator] is a tab
- SequenceFileInputFormat
  - Binary file of (key,value) pairs with some additional metadata
- SequenceFileAsTextInputFormat
  - Similar, but maps (key.toString(), value.toString())

Keys and Values are Objects

- Keys and Values in Hadoop are Java objects
  - Not primitives
- Values are objects which implement Writable
- Keys are objects which implement WriteableComparable

Hadoop Writable Classes

- IntWritable for ints
- LongWritable for longs
- FloatWritable for floats
- DoubleWritable for doubles
- Text for strings
- Etc.

The Driver: Complete Code (1)

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
public class WordCount {
    public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.out.printf("Usage: WordCount <input dir> <output dir>\n");
        System.exit(-1);
    }
    Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
```

The Driver: Complete Code (2)

```
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

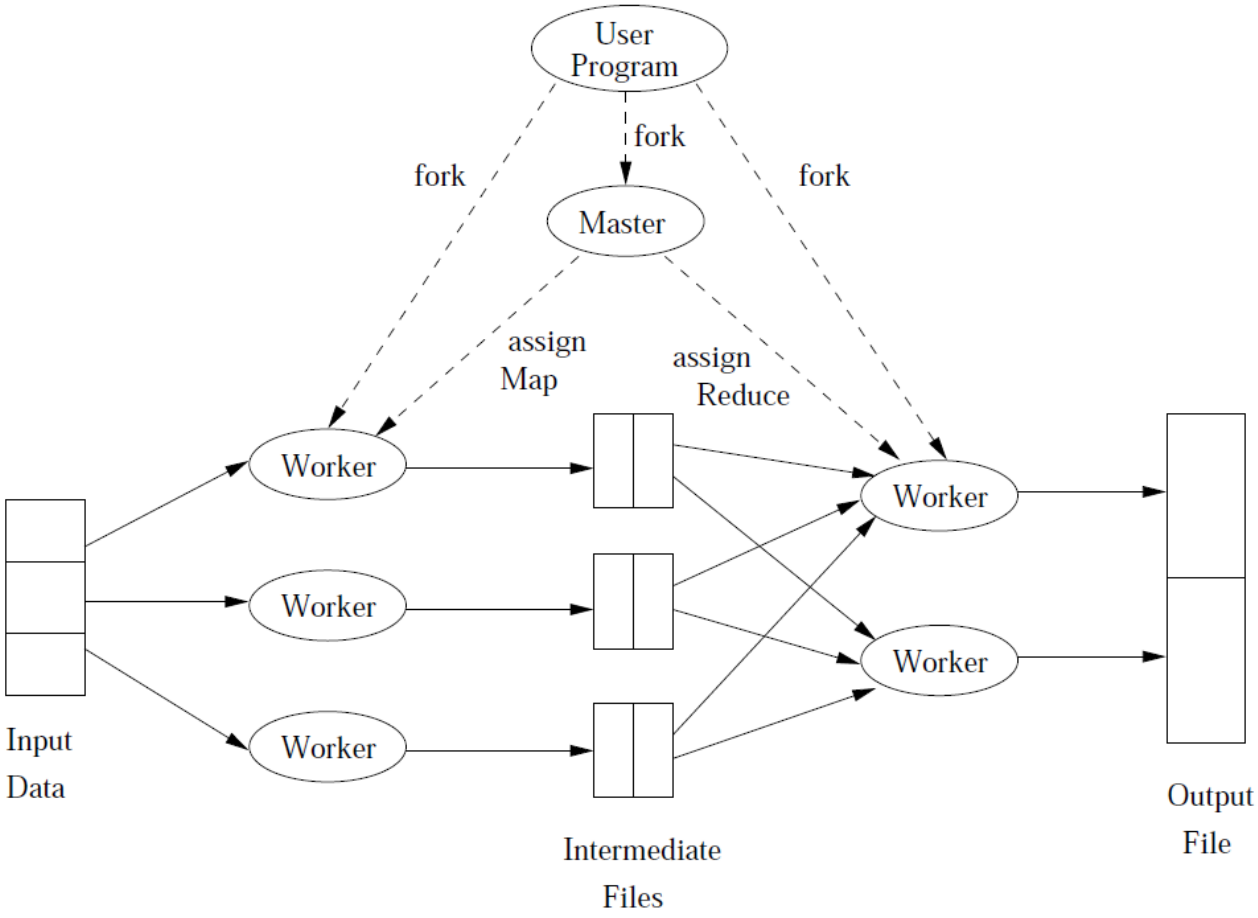
The Mapper Complete Code

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String line = value.toString();
        for (String word : line.split("\\W+")) {
            if (word.length() > 0) {
                context.write(new Text(word), new IntWritable(1));
            }
        }
    }
}
```

The Reducer Complete Code

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class SumReducer extends Reducer<Text, IntWritable, Text,
IntWritable>
{
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int wordCount = 0;
        for (IntWritable value : values) {
            wordCount += value.get();
        }
        context.write(key, new IntWritable(wordCount));
    }
}
```

Execution of a MapReduce Program



Writing Hadoop Code Using Eclipse

- Eclipse is an integrated development environment (IDE)
  - Open source
  - Very popular among Java developers
  - Has plug-ins to speed development in several different languages

Homework3: Writing MapReduce Programs

- Writing a MapReduce Java Program
  - Average Word Length
- More Practice with MapReduce Java programs
  - Log File Analysis
- Writing a report to describe your work

Any Questions?