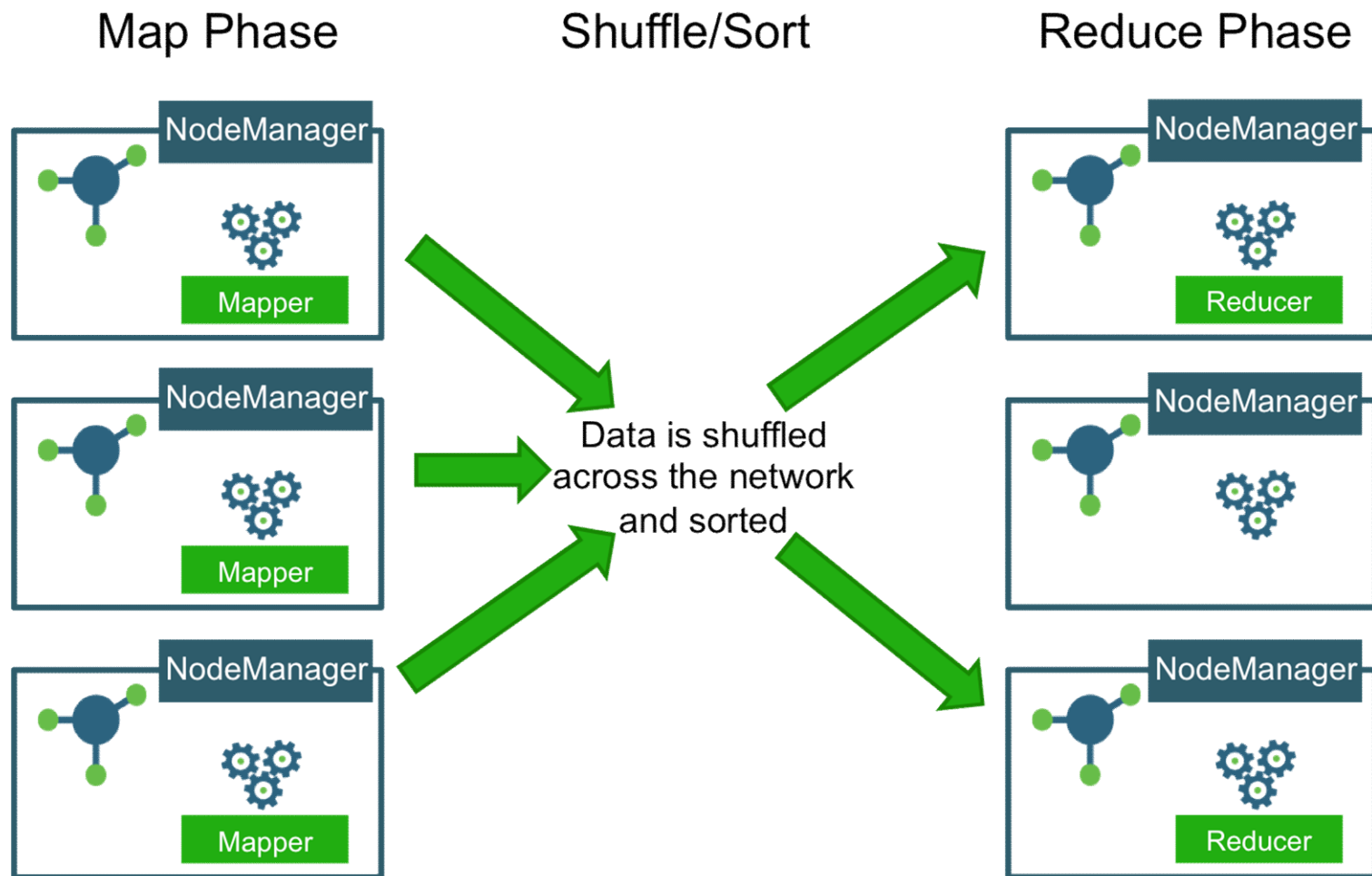


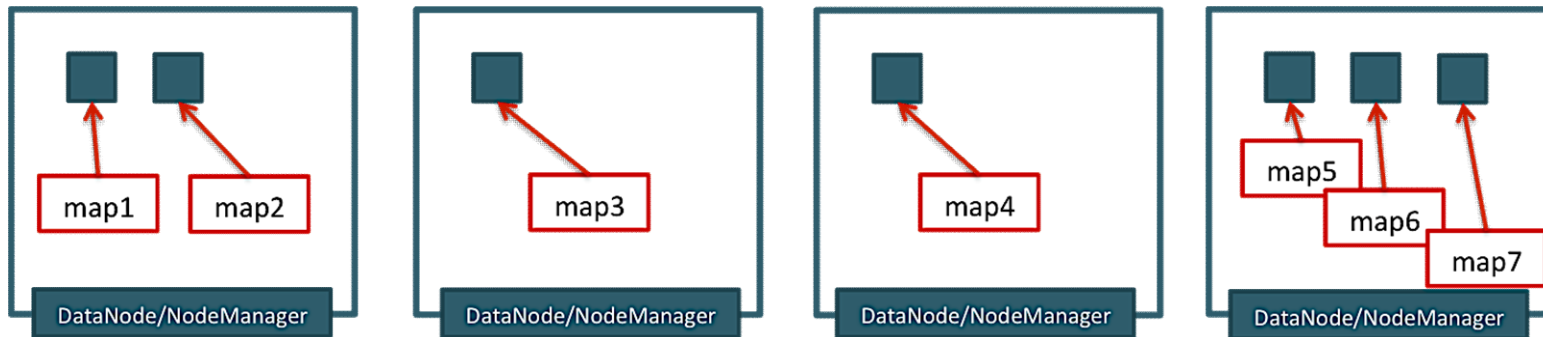
# Writing MapReduce Applications





# Understanding MapReduce

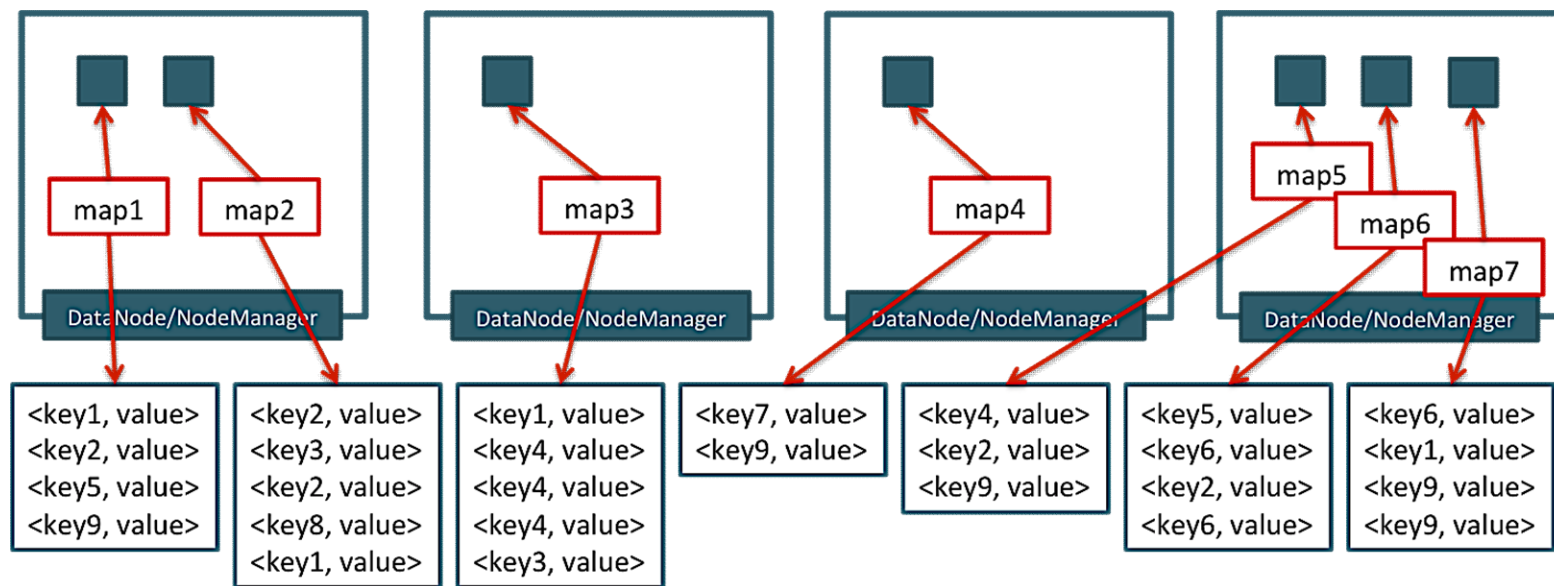
1. Suppose a file is the input to a MapReduce job. That file is broken down into blocks stored on DataNodes across the Hadoop cluster.



2. During the Map phase, map tasks process the input of the MapReduce job, with a map task assigned to each Input Split. The map tasks are Java processes that ideally run on the DataNodes where the blocks are stored.

# Understanding MapReduce - cont.

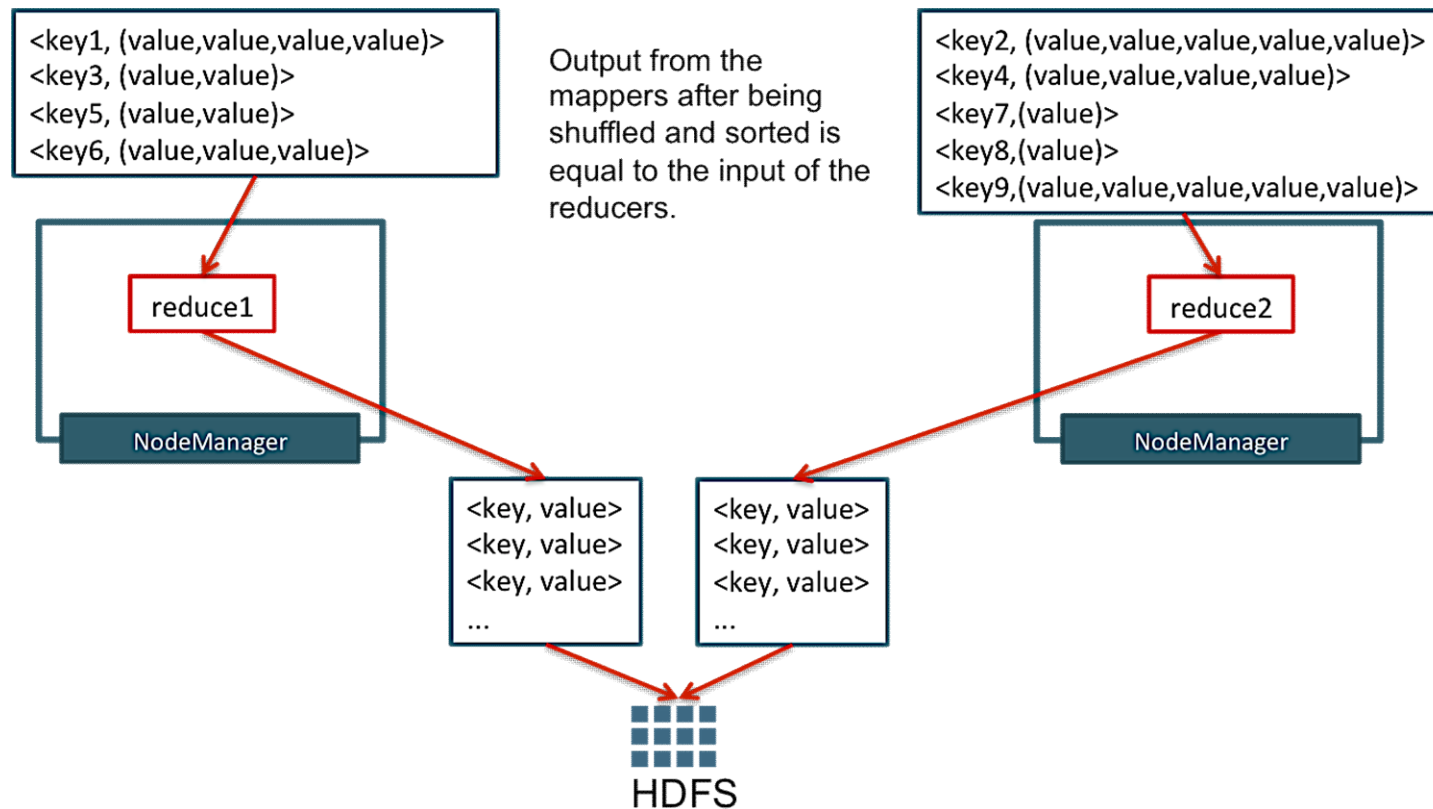
3. Each map task processes its Input Split and outputs records of <key, value> pairs.



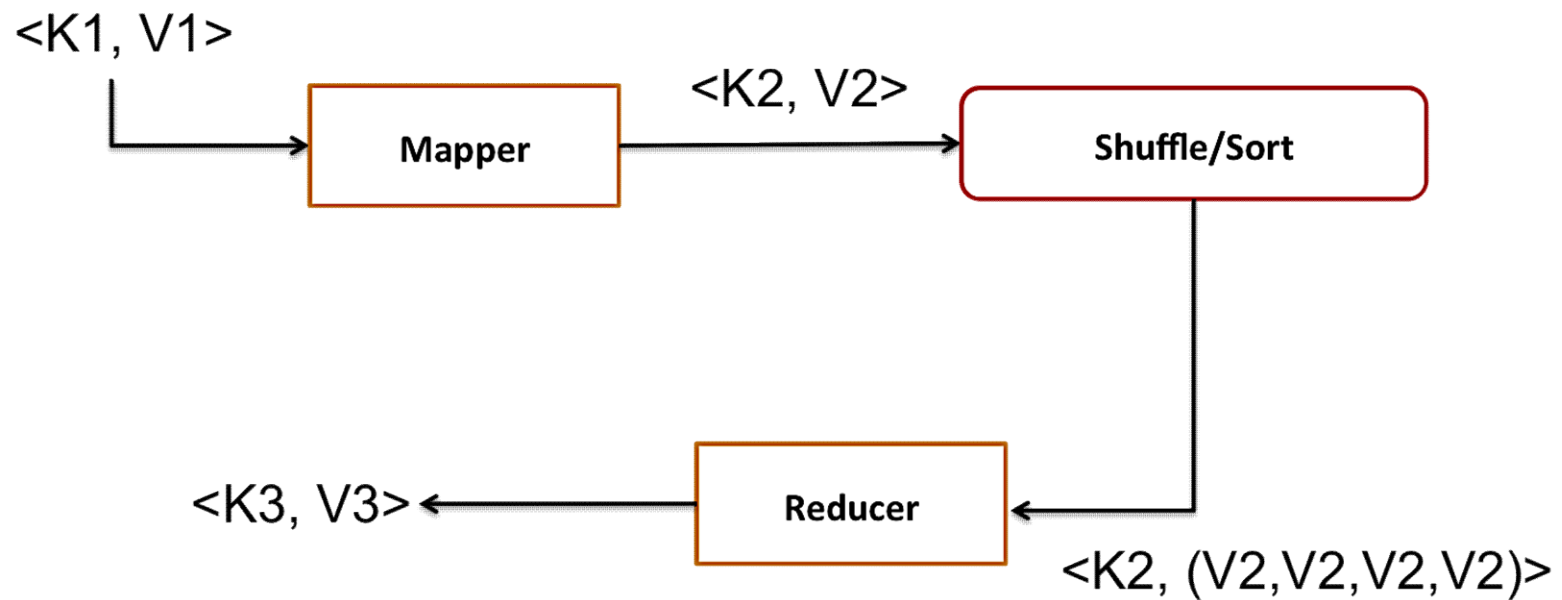
4. The <key,value> pairs go through a shuffle/sort phase, where records with the same key end up at the same reducer. The specific pairs sent to a reducer are sorted by key, and the values are aggregated into a collection.

# Understanding MapReduce - cont.

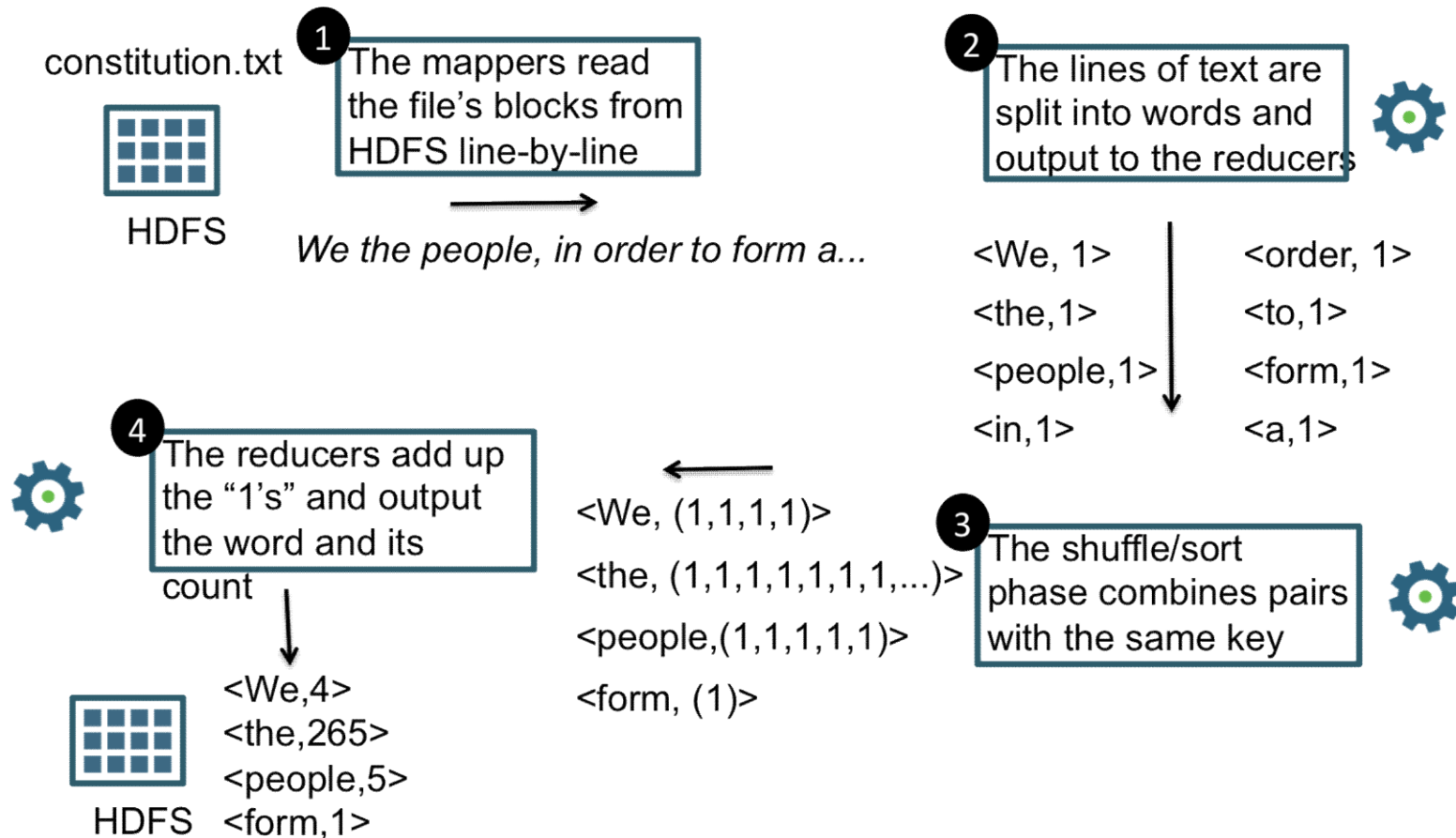
5. Reduce tasks run on a NodeManager as a Java process. Each Reducer processes its input and outputs <key,value> pairs that are typically written to a file in HDFS.



# The Key/Value Pairs of MapReduce



# WordCount in MapReduce



# Demonstration

Understanding MapReduce



# WordCountMapper

```
public class WordCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value,
                        Context context)
        throws IOException, InterruptedException {
        String currentLine = value.toString();
        String [] words = currentLine.split(" ");
        for(String word : words) {
            Text outputKey = new Text(word);
            context.write(outputKey, new IntWritable(1));
        }
    }
}
```

# WordCountReducer

```
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text,
IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable>
values,
                        Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable count : values) {
            sum += count.get();
        }
        IntWritable outputValue = new IntWritable(sum);
        context.write(key, outputValue);
    }
}
```

# WordCountJob

```
Job job = Job.getInstance (getConf(), "WordCountJob");
Configuration conf = job.getConfiguration();
job.setJarByClass (getClass());

Path in = new Path(args[0]);
Path out = new Path(args[1]);
FileInputFormat.setInputPaths (job, in);
FileOutputFormat.setOutputPath (job, out);
job.setMapperClass (WordCountMapper.class);
job.setReducerClass (WordCountReducer.class);
job.setInputFormatClass (TextInputFormat.class);
job.setOutputFormatClass (TextOutputFormat.class);
job.setMapOutputKeyClass (Text.class);
job.setMapOutputValueClass (IntWritable.class);
job.setOutputKeyClass (Text.class);
job.setOutputValueClass (IntWritable.class);
return job.waitForCompletion (true) ? 0 : 1;
```

# Running a MapReduce Job

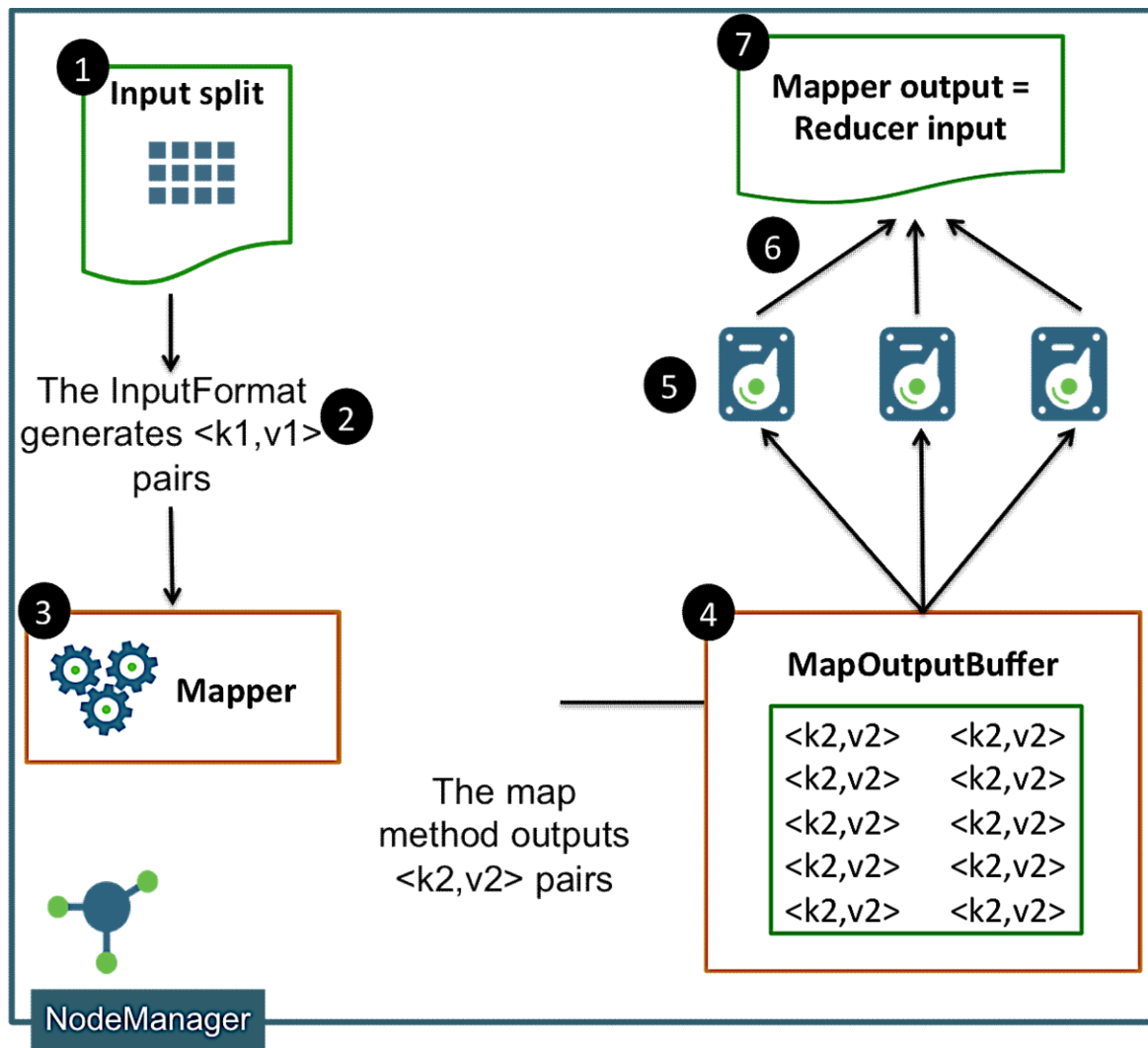
To run a job, perform the following steps:

- Put the input files into HDFS
- If the output directory exists, delete it
- Use **hadoop** to execute the job
- View the output files

```
yarn jar wordcount.jar  
my.WordCountJob input/file.txt result
```

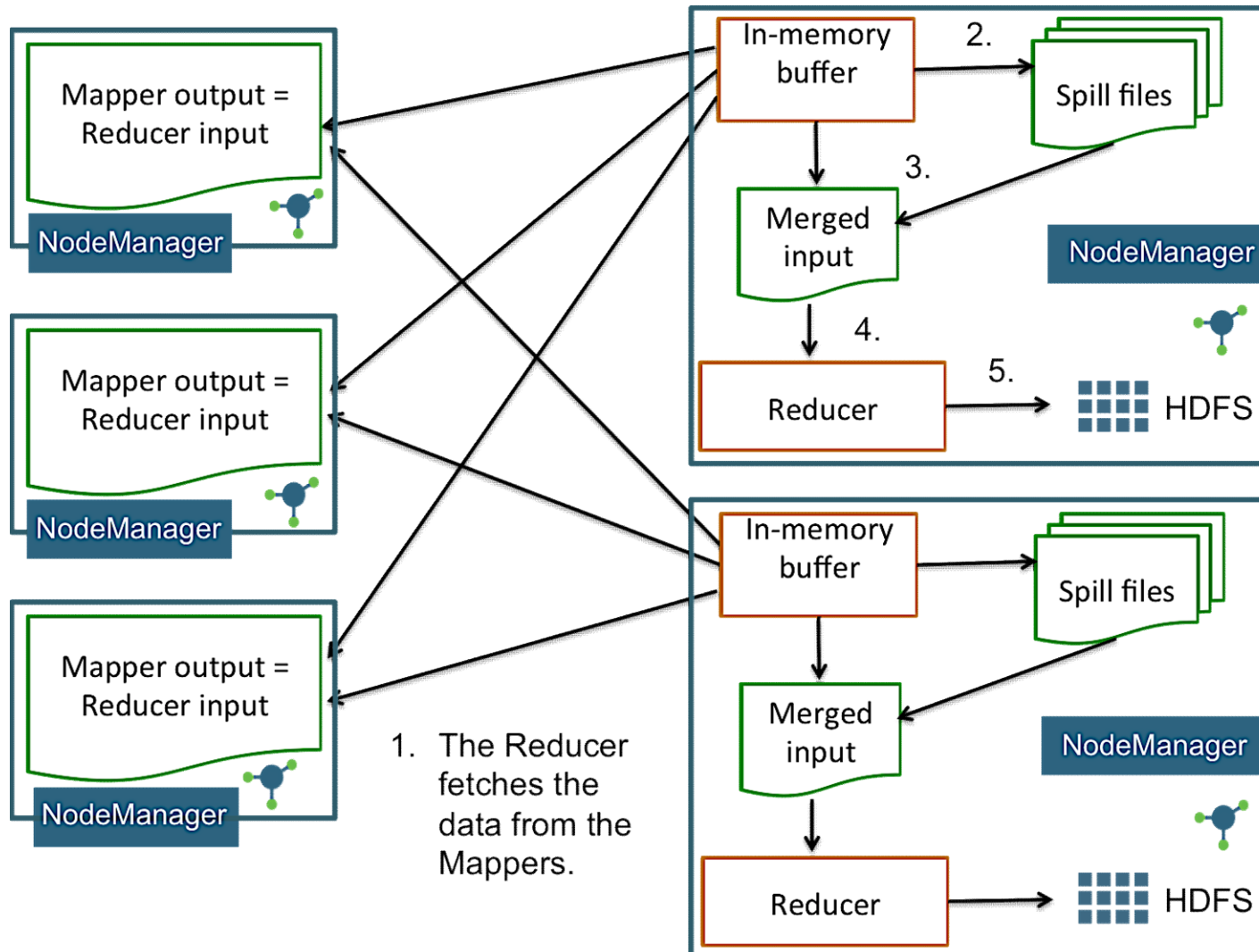
# Hands-On

Word Count



Spill files are merged into a single file

Records are sorted and spilled to disk when the buffer reaches a threshold





# About YARN

YARN = Yet Another Resource Negotiator

YARN splits up the functionality of the JobTracker in Hadoop 1.x into two separate processes:

- **ResourceManager**: for allocating resources and scheduling applications
- **ApplicationMaster**: for executing applications and providing failover

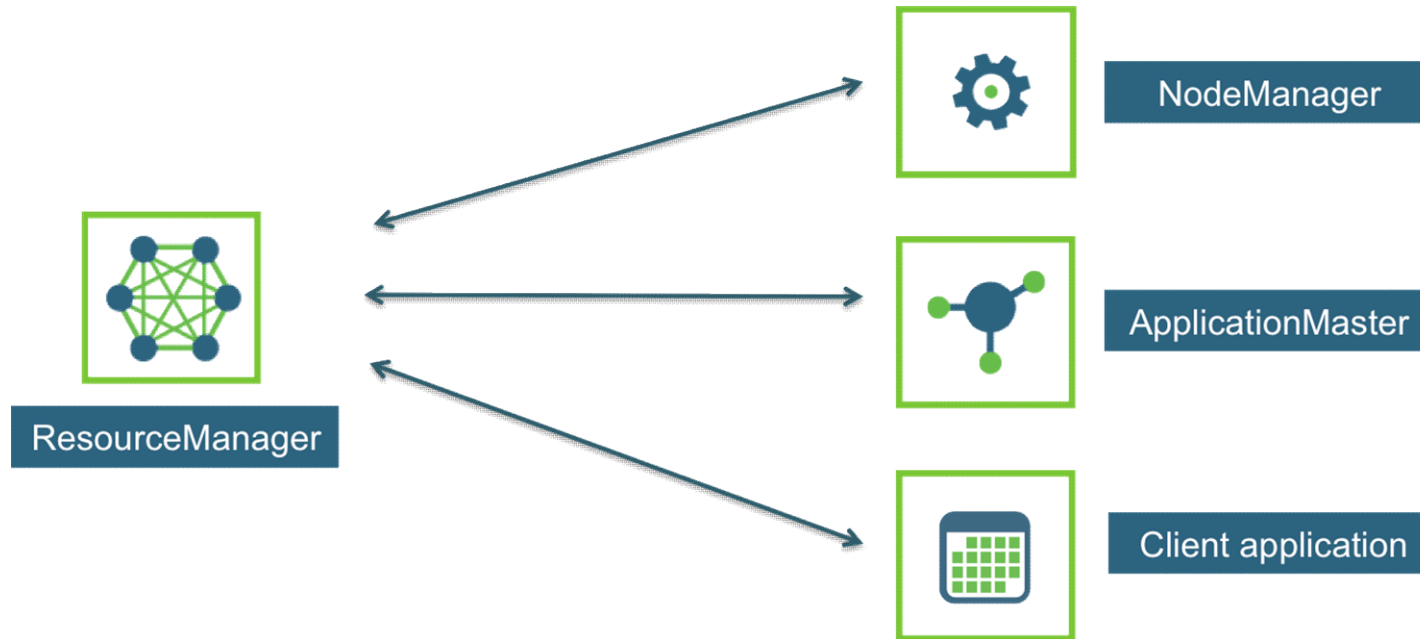


# Open-Source YARN Use Cases

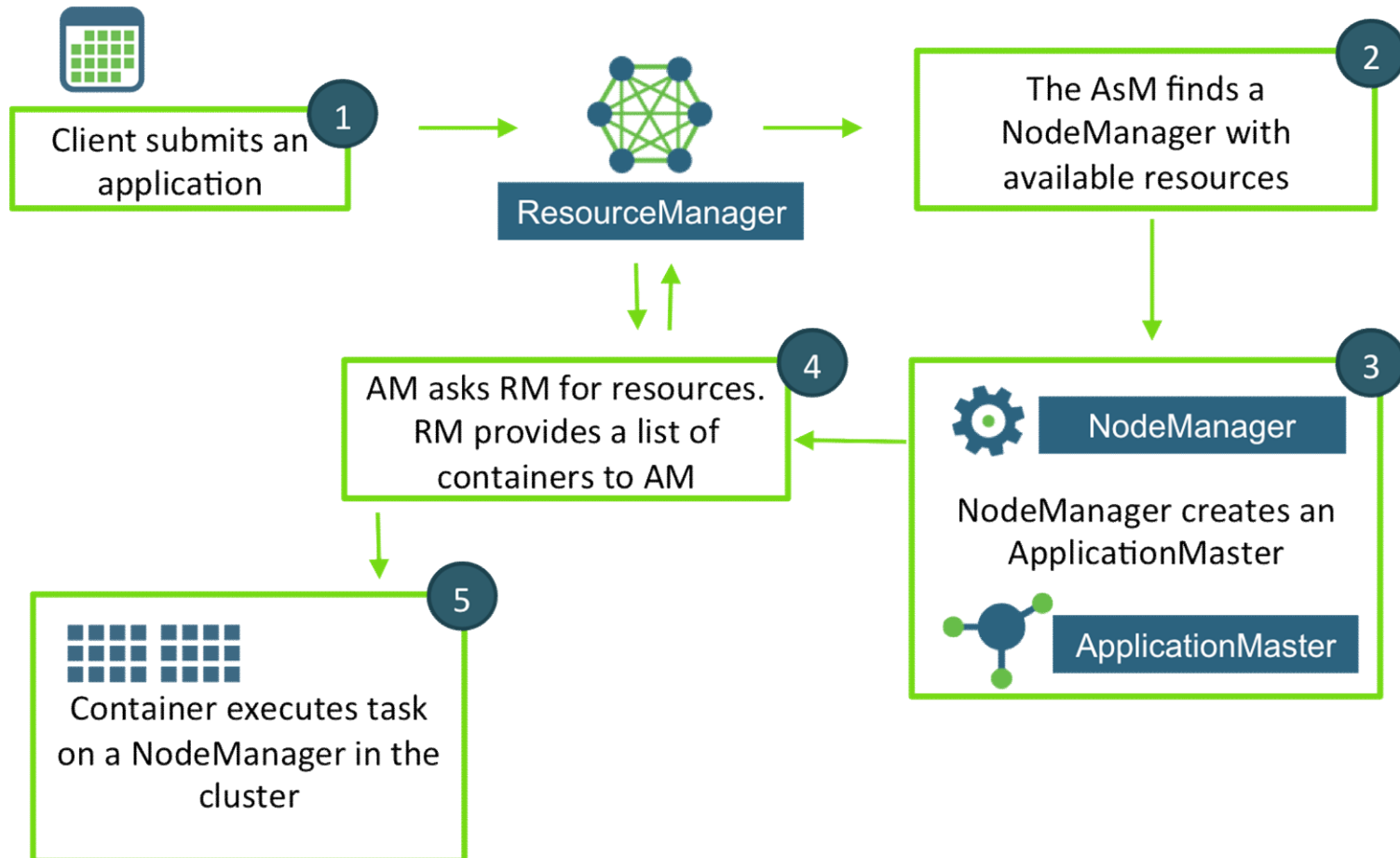
- **Tez:** improves the execution of MapReduce jobs
- **Slider:** for deploying existing distributed applications onto YARN
- **Storm:** for real-time computing
- **Spark:** a MapReduce-like cluster computing framework designed for low-latency iterative jobs and interactive use from an interpreter
- **Open MPI:** a high-performance Message Passing Library that implements MPI-2
- **Apache Giraph:** a graph processing platform

# The Components of YARN

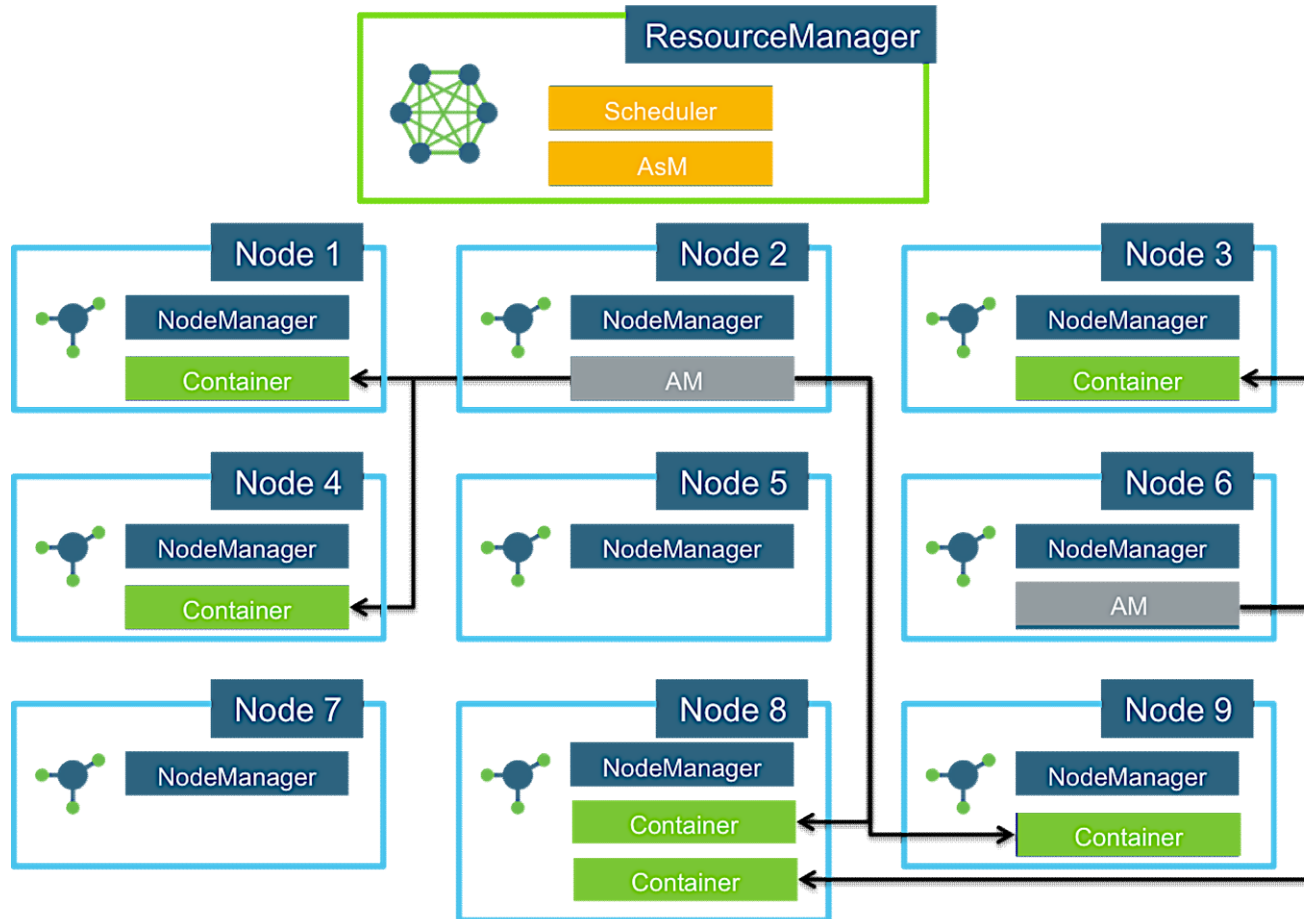
The **ResourceManager** communicates with the **NodeManagers**, **ApplicationMasters**, and **Client** applications.

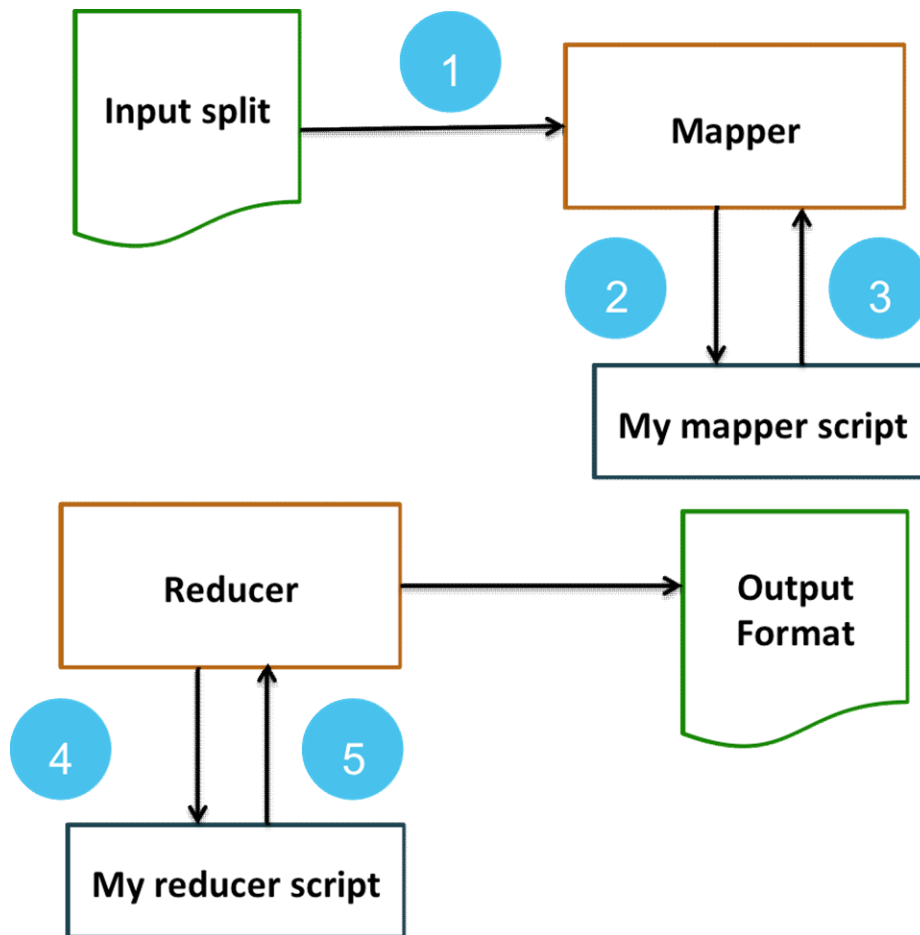


# Lifecycle of a YARN Application



# A Cluster View Example





1. The InputFormat generates  $\langle k1, v1 \rangle$  pairs
2. Converts the  $\langle k1, v1 \rangle$  pairs to lines and sends them to the stdin of the process
3. The stdout of the process is converted into  $\langle k2, v2 \rangle$  pairs
4. Converts  $\langle k2, (v2, v2, \dots) \rangle$  pairs to lines and send them to the stdin of the process
5. The stdout of the process is converted into  $\langle k3, v3 \rangle$  pairs

# Running a Hadoop Streaming Job

A Streaming job is a MapReduce job defined in the `hadoop-streaming.jar` file:

```
hadoop jar $HADOOP_HOME/lib/hadoop-streaming.jar  
  -input input_directories  
  -output output_directories  
  -mapper mapper_script  
  -reducer reducer_script
```

# Review

1. What are the three main phases of a MapReduce job?
2. Suppose the Mappers of a MapReduce job output `<key,value>` pairs that are of type `<integer,string>`. What will the pairs that are processed by the corresponding Reducers look like?
3. What happens if all the `<key,value>` pairs output by a Mapper do not fit into the memory of the Mapper?
4. What determines the number of Mappers of a MapReduce job?
5. What determines the number of Reducers of a MapReduce job?
6. **True or False:** The shuffle/sort phase sorts the keys and values as they are passed to the Reducer.
7. What are the four main components of YARN?
8. What happens if a container fails to complete its task in a YARN application?

# Hands-On

Distributed Grep

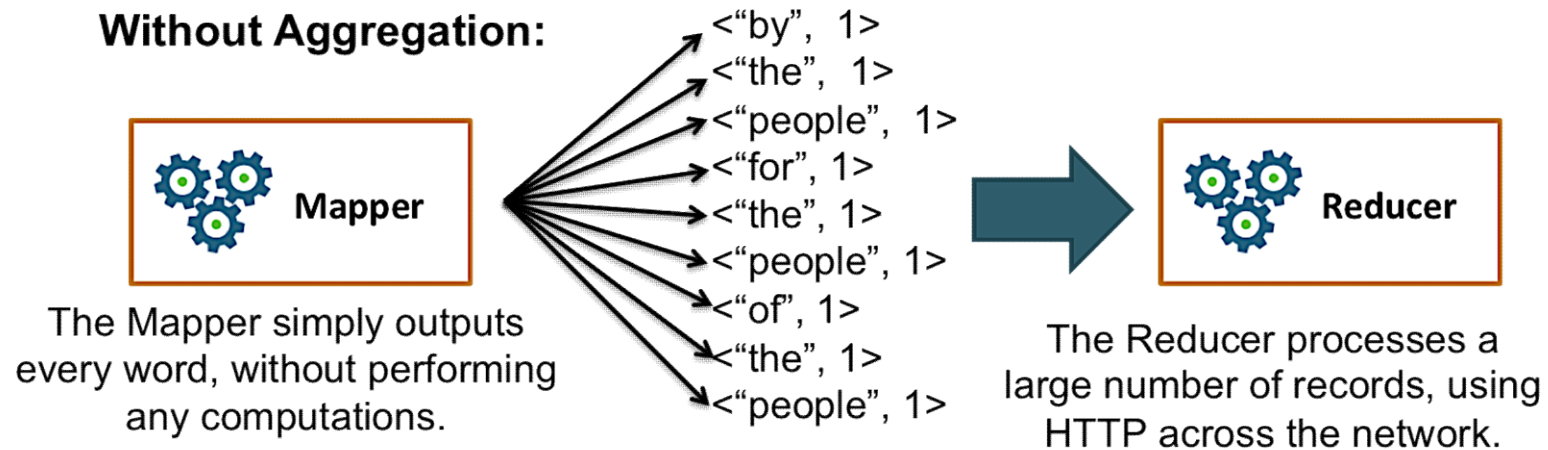


# Hands-On

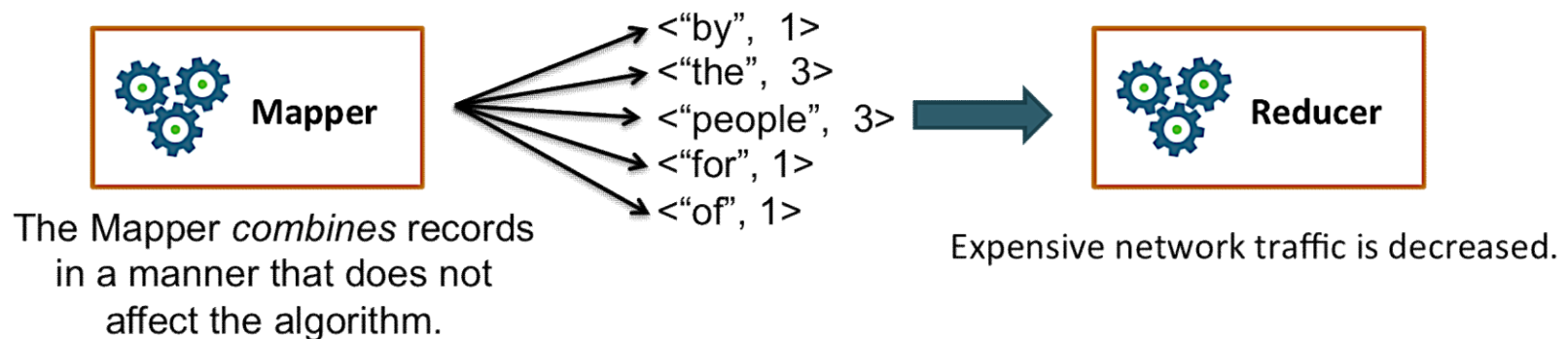
Inverted Index

# Map Aggregation

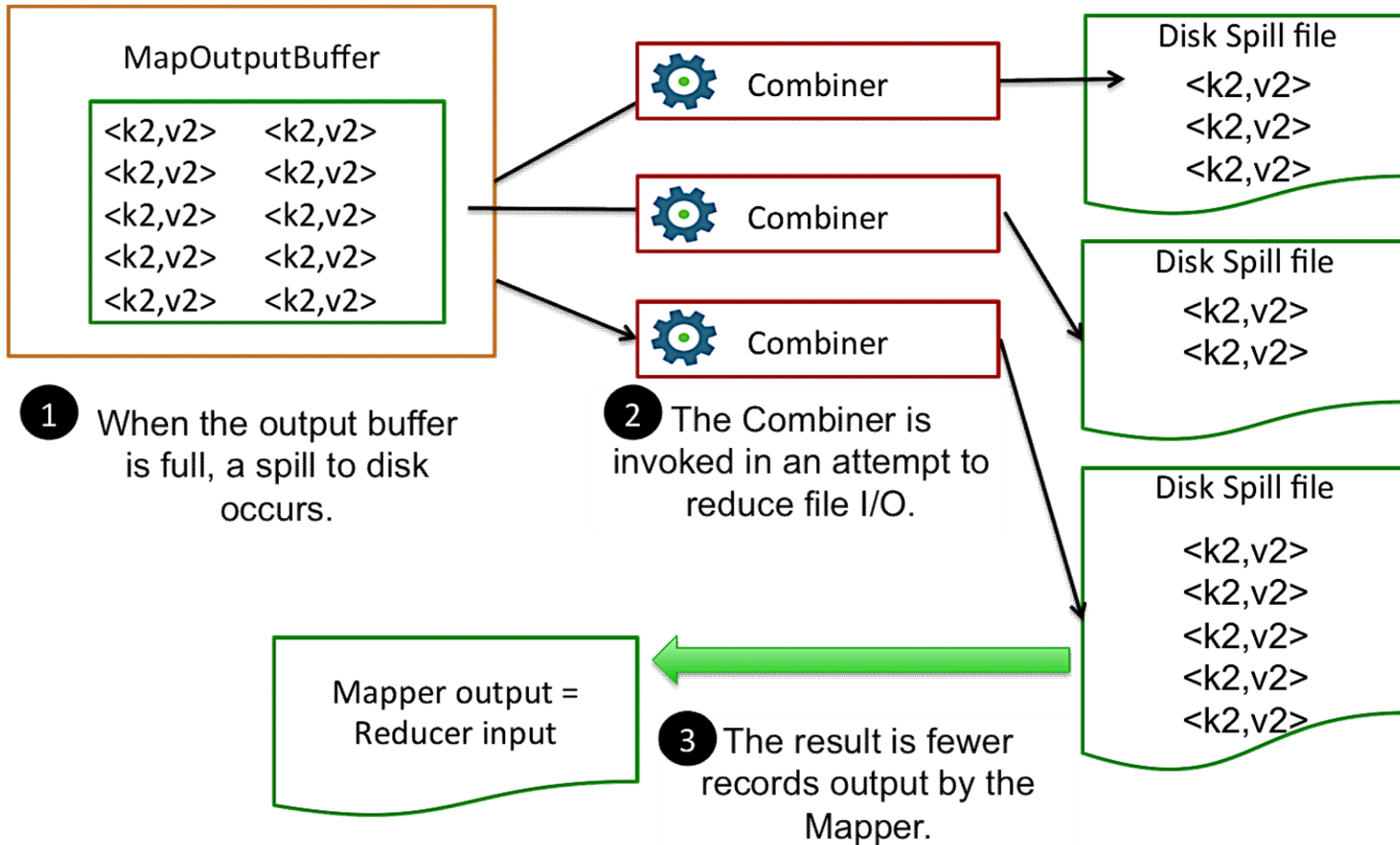
### Without Aggregation:



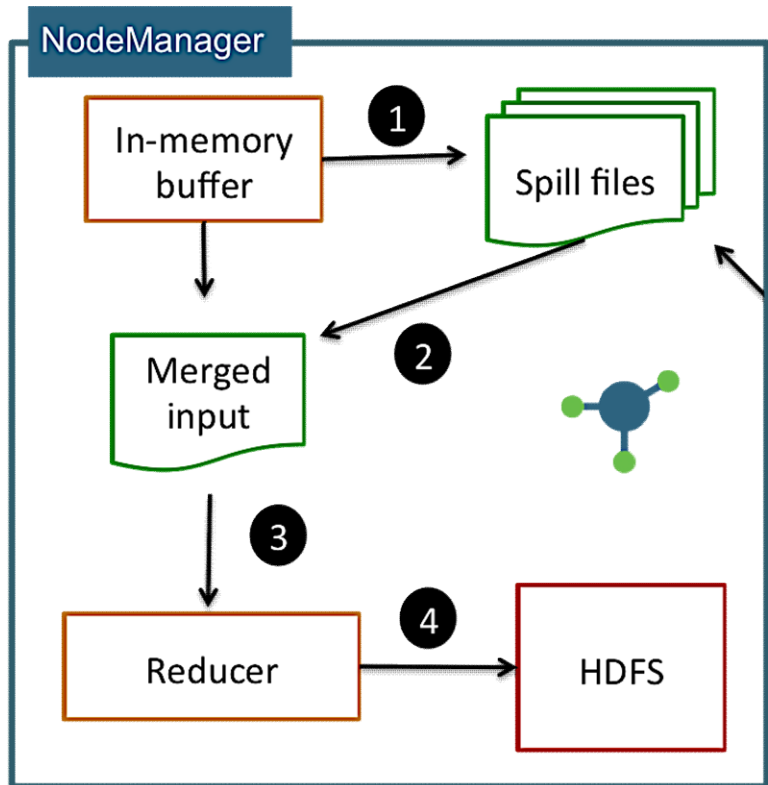
### With Aggregation:



# Overview of Combiners



# Reduce-side Combining



The combiner is also used in the reduce phase if the intermediate <key,value> pairs from Mappers are spilled to disk.

# Example of a Combiner

```
public class WordCountCombiner
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable outputValue = new IntWritable();

    @Override
    protected void reduce(Text key,
        Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable count : values) {
            sum += count.get();
        }
        outputValue.set(sum);
        context.write(key, outputValue);
    }
}
```