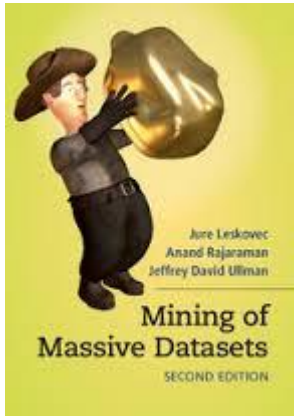
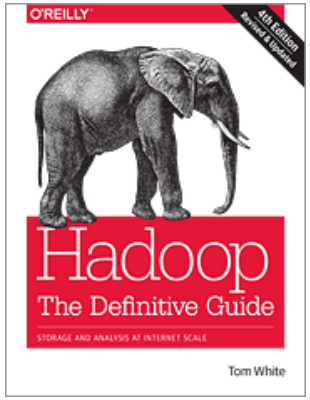
Chapter 2: Data Mining New Software Stack



Chapter 2: MapReduce



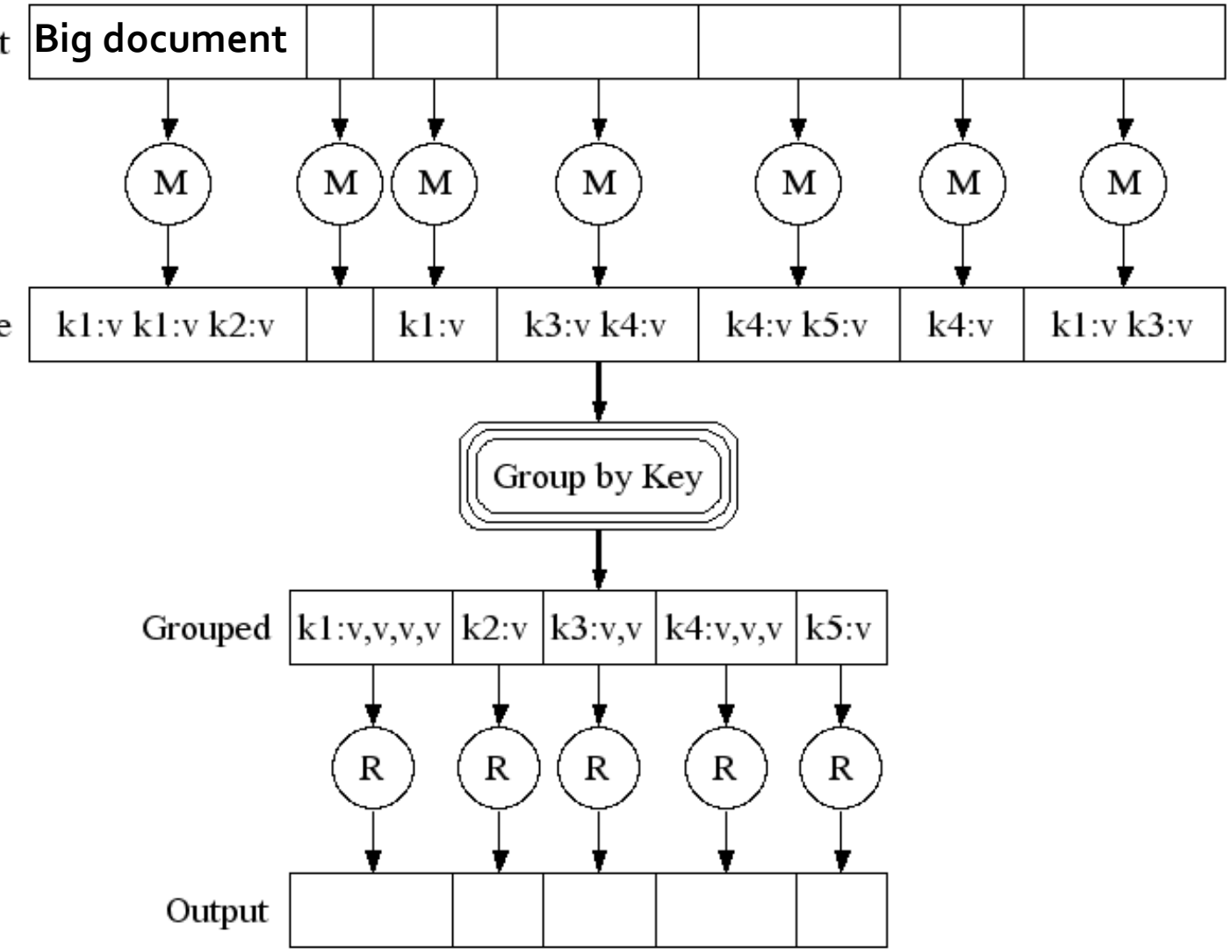
Instructor: Mehmet S. Aktaş

Map-Reduce: A diagram

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



A Weather Dataset

- Program that mines weather data
  - Weather sensors collect data every hour at many locations across the globe
  - They gather a large volume of log data, which is good candidate for analysis with MapReduce
- Data Format
  - Data from the National Climate Data Center(NCDC)
  - Stored using a line-oriented ASCII format, in which each line is a record

Example 2-1. 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
```

A Weather Dataset

- Data Format
  - Data files are organized by date and weather station.
  - There is a directory for each year from 1901 to 2001, each containing a gzipped file for each weather station with its readings for that year.

```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

  - The whole dataset is made up of a large number of relatively small files since there are tens of thousands of weather station.
  - The data was preprocessed so that each year's readings were concatenated into a single file.

Analyzing the Data with Unix Tools

- What's the highest recorded global temperature for each year in the dataset?
- A Unix Shell script program for processing line-oriented data

Example 2-2. A program for finding the maximum recorded temperature by year from records

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`"\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
         END { print max }'
done
```

```
% ./max_temperature.sh
1901    317
1902    244
1903    289
1904    256
1905    283
...
```

Maximum temperature is 31.7℃ for 1901.

- The complete run for the century took 42 minutes in one run on a single EC2 High-CPU Extra Large Instance.

Analyzing the Data with Unix Tools

- To speed up the processing, run parts of the program in parallel
- Problems for parallel processing
  - Dividing the work into equal-size pieces isn't always easy or obvious.
    - The file size for different years varies
    - The whole run is dominated by the longest file
    - A better approach is to split the input into fixed-size chunks and assign each chunk to a process
  - Combining the results from independent processes may need further processing.
  - Still limited by the processing capacity of a single machine, handling coordination and reliability for multiple machines
- It's feasible to parallelize the processing, though, it's messy in practice.

Analyzing the Data with Map and Reduce

- Map and Reduce
  - MapReduce works by breaking the processing into 2 phases: the map and the reduce.
  - Both map and reduce phases have key-value pairs as input and output.
  - Programmers have to specify two functions: map and reduce function.
  - The **input to the map phase** is the raw NCDC data.
    - Here, the key is the offset of the beginning of the line and the value is each line of the data set.
  - The **map function** pulls out the year and the air temperature from each input value.
  - The **reduce function** takes <year, temperature> pairs as input and produces the maximum temperature for each year as the result.

Analyzing the Data with Hadoop – Map and Reduce

- Original NCDC Format

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

- Input file for the map function, stored in HDFS

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

- Output of the map function, running in parallel for each block

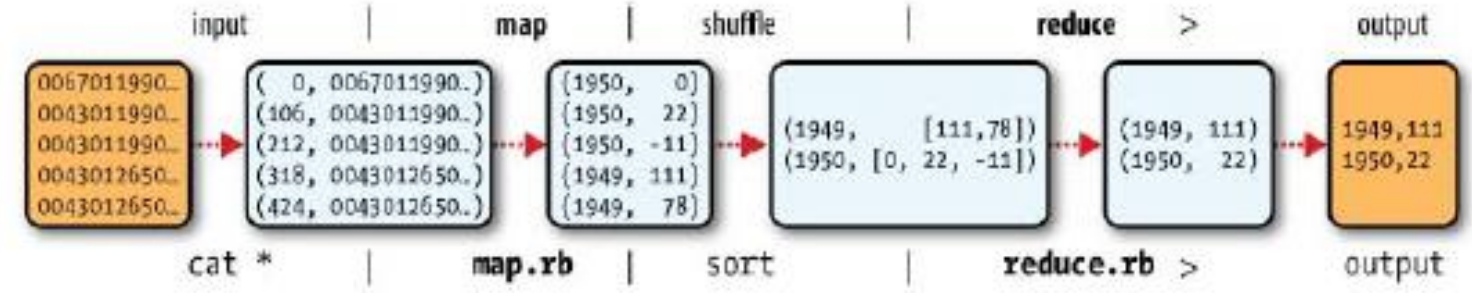
```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

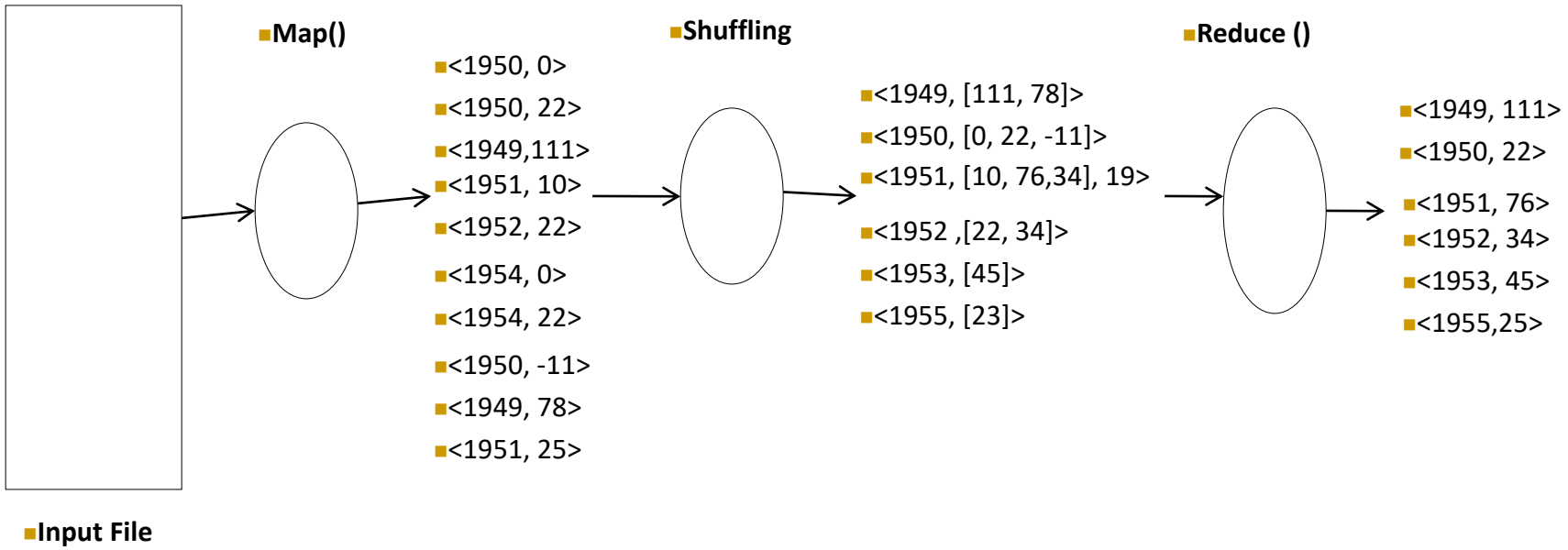
- Input for the reduce function & Output of the reduce function

```
(1949, [111, 78])          (1949, 111)
(1950, [0, 22, -11])       (1950, 22)
```

Analyzing the Data with Hadoop – Map and Reduce

- The whole data flow





Map function

- Takes an input pair
- Produces a set of intermediate **key/value** pairs
- The MapReduce library groups together all intermediate values associated with the same intermediate key i and passes them to the reduce function

Analyzing the Data with Hadoop – Java MapReduce

- Having run through how the MapReduce program works, express it in code
  - A map function, a reduce function, and some code to run the job are needed.
- Map function

```
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.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class MaxTemperatureMapper extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {

    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]")) {
      output.collect(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

Analyzing the Data with Hadoop – Java MapReduce

- Reduce function

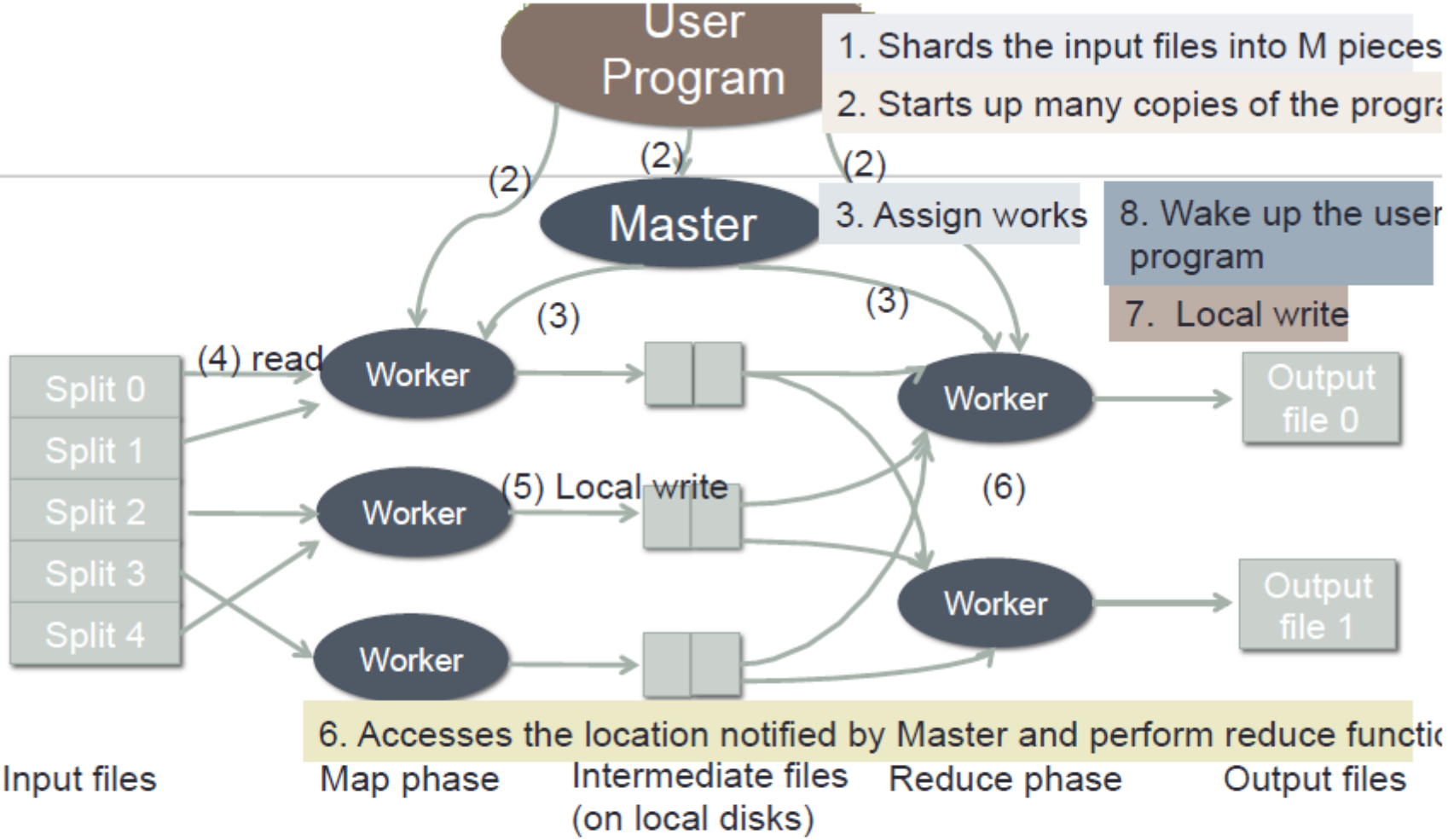
```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MaxTemperatureReducer 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 maxValue = Integer.MIN_VALUE;
    while (values.hasNext()) {
      maxValue = Math.max(maxValue, values.next().get());
    }
    output.collect(key, new IntWritable(maxValue));
  }
}
```

Analyzing the Data with Hadoop – Java MapReduce

- Main function for running the MapReduce job

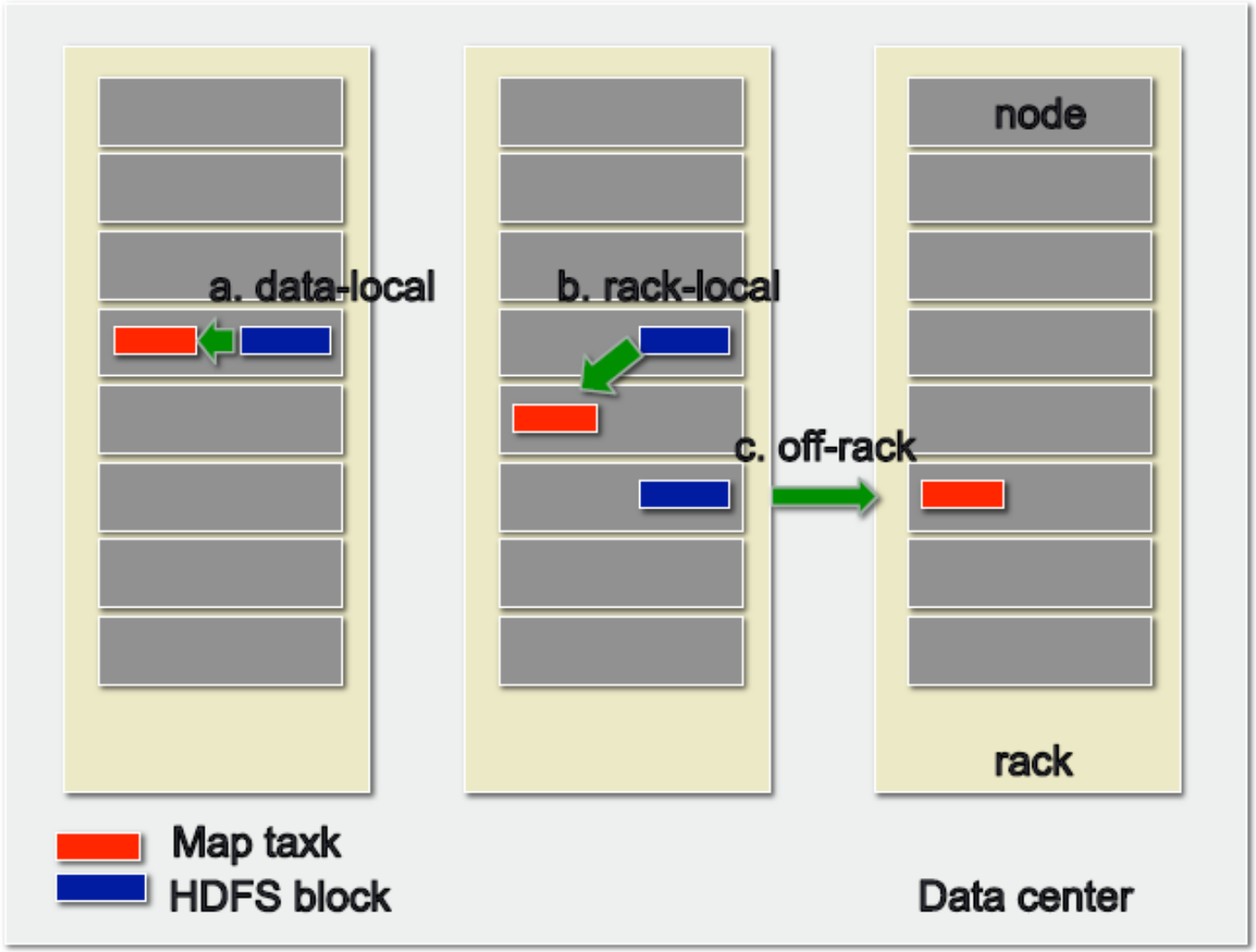
```
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;

public class MaxTemperature {

  public static void main(String[] args) throws IOException {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    JobConf conf = new JobConf(MaxTemperature.class);
    conf.setJobName("Max temperature");

    FileInputFormat.addInputPath(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setMapperClass(MaxTemperatureMapper.class);
    conf.setReducerClass(MaxTemperatureReducer.class);

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

    JobClient.runJob(conf);
  }
}
```

Map Reduce Data Flow



4. Read contents of the corresponding input shard
Parse and passes the key-value pair to the Map function

5. Buffered pairs are written to local disk
Location is reported to the Master and the Master forwards them to the reduce worker


node
a. data-local
b. rack-local
c. off-rack
rack
Map taxk
HDFS block
Data center

Data locality optimization

- Hadoop tries to run the map task on a node where the input data resides
  - It minimizes the usage of cluster bandwidth
- If all replication nodes are running other map tasks
  - The job scheduler will look for a free map slot on a node in the same rack

Scaling Out

- Data Flow – single reduce task
  - Reduce tasks don't have the advantage of data locality – the input to a single reduce task is normally the output from all mappers.
  - All map outputs are merged across the network and passed to the user-defined reduce function.
  - The output of the reduce is normally stored in HDFS.

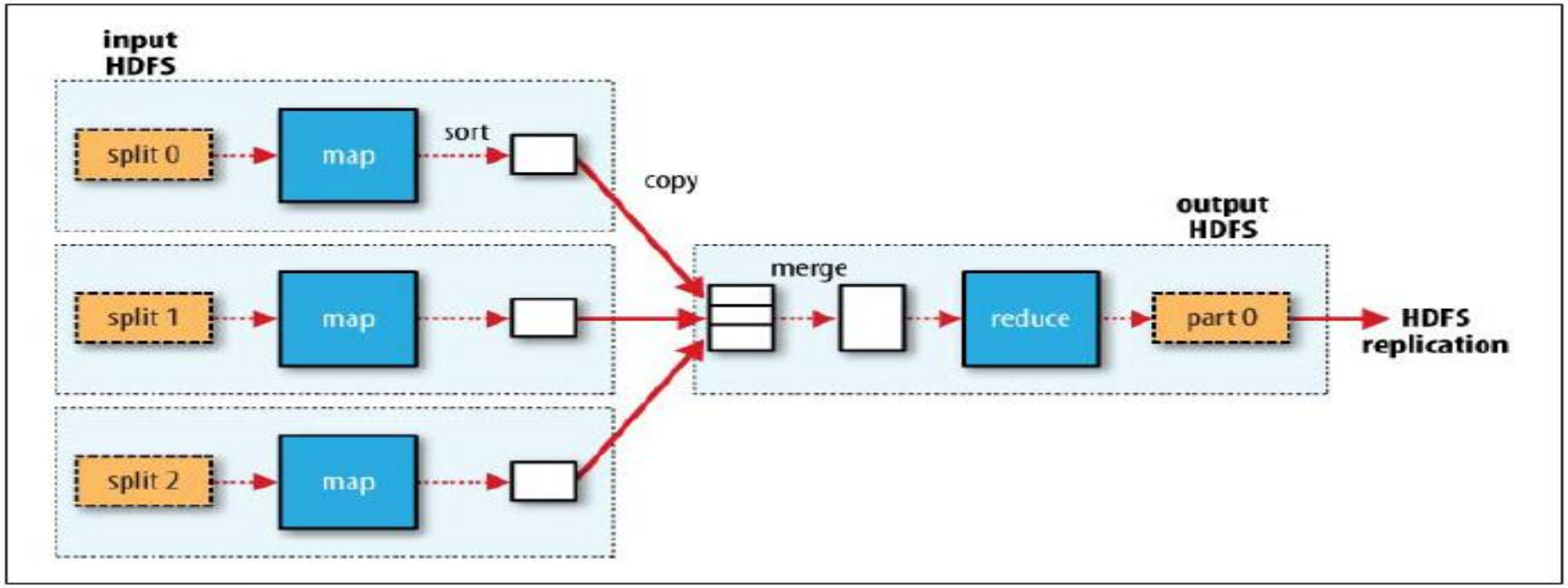


Figure 2-2. MapReduce data flow with a single reduce task

Scaling Out

- Data Flow – multiple reduce tasks
  - The number of reduce tasks is specified independently not governed by the input size.
  - The map tasks partition their output by keys, each creating one partition for each reduce task.
  - There can be many keys and their associated values in each partition, but the records for any key are all in a single partition.

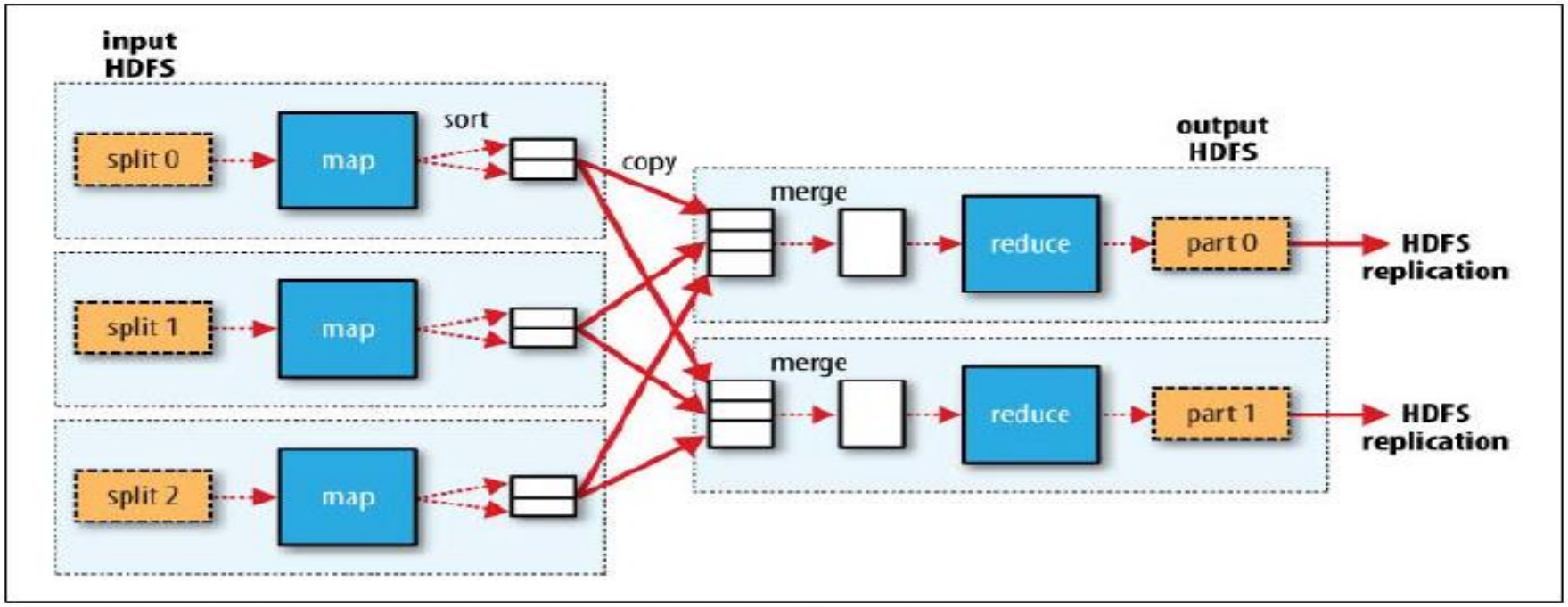


Figure 2-3. MapReduce data flow with multiple reduce tasks

Scaling Out

- Data Flow – zero reduce task

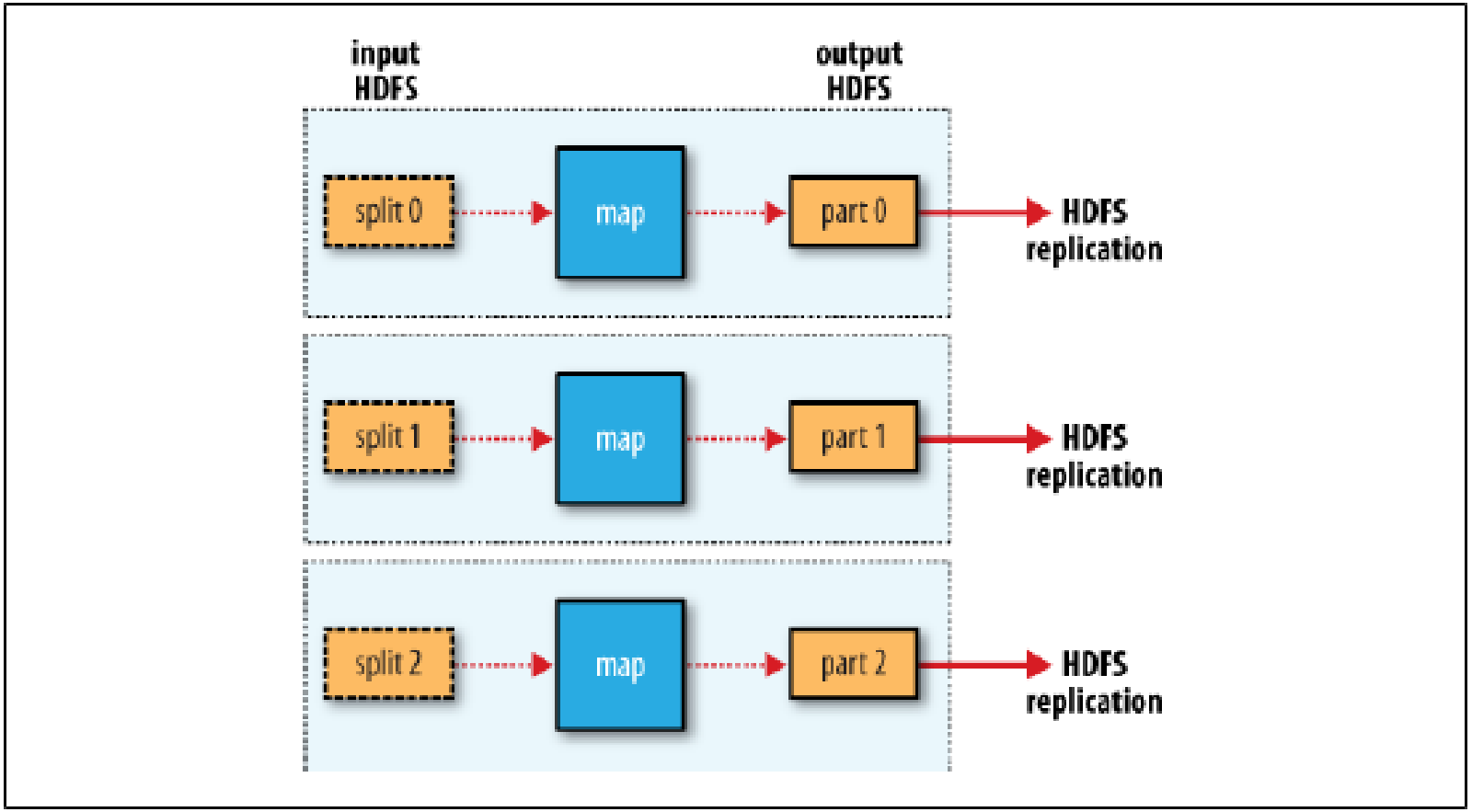


Figure 2-4. MapReduce data flow with no reduce tasks

Shuffle

- The process by which the system performs the sort and transfers the map outputs to the reducers as inputs
  - MapReduce makes the guarantee that the input to every reducer is sorted by key

Combiner functions

- Minimize the data transferred between map and reduce tasks

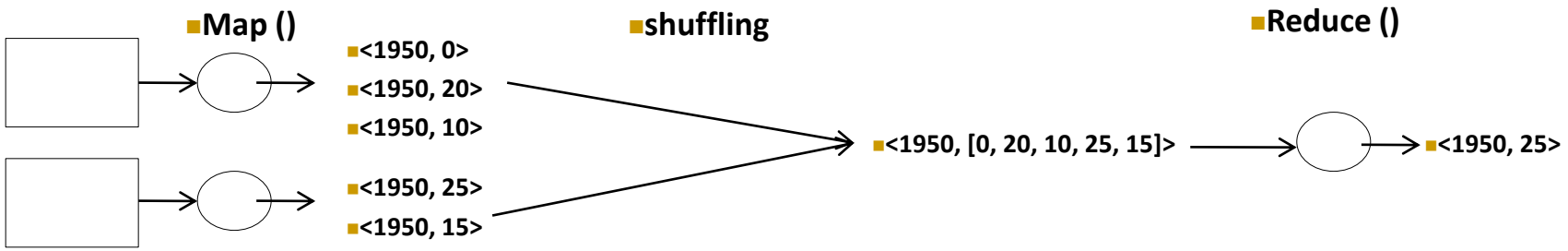
- Users can specify a ***combiner function***
  - To be run on the map output
  - To replace the map output with the combiner output

Scaling Out

- Combiner Functions
  - The function calls on the temperature values can be expressed as follows:
    - Max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25
  - Calculating ‘mean’ temperatures couldn’t use the mean as the combiner function
    - mean(0, 20, 10, 25, 15) = 14
    - mean(mean(0, 20, 10), mean(25, 15)) = mean(10, 20) = 15.
  - The combiner function doesn’t replace the reduce function.
  - It can help cut down the amount of data shuffled between the maps and the reduces

Scaling Out

- Combiner Functions
  - Example without a combiner function



  - Example with a combiner function, finding maximum temperature for a map