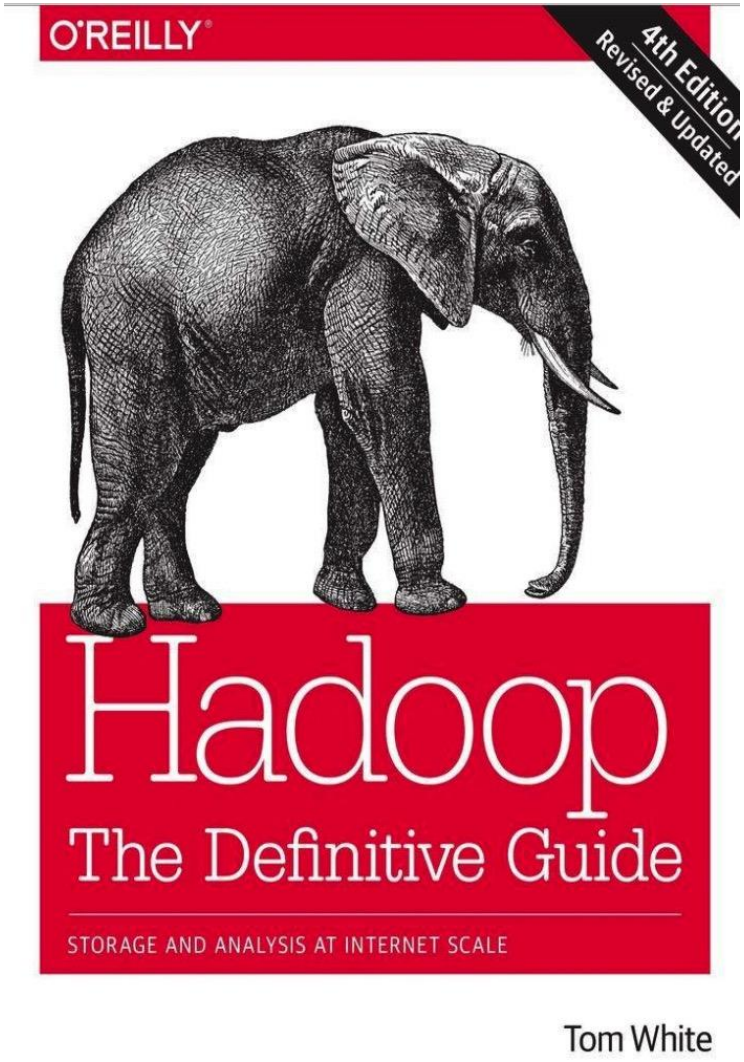
Hadoop MapReduce

Department of Computer Science,

National University of Computer & Emerging Sciences,

Islamabad Campus

An excellent Hadoop Reference



MapReduce

- To **speed up** the **processing**, we need to **run parts** of the **program** in **parallel**

- **Feasible to parallelize** the **processing**, **in practice is Challanging…**

- Using a **framework** like **Hadoop** to **take care of these issues** is a **great help**

MapReduce

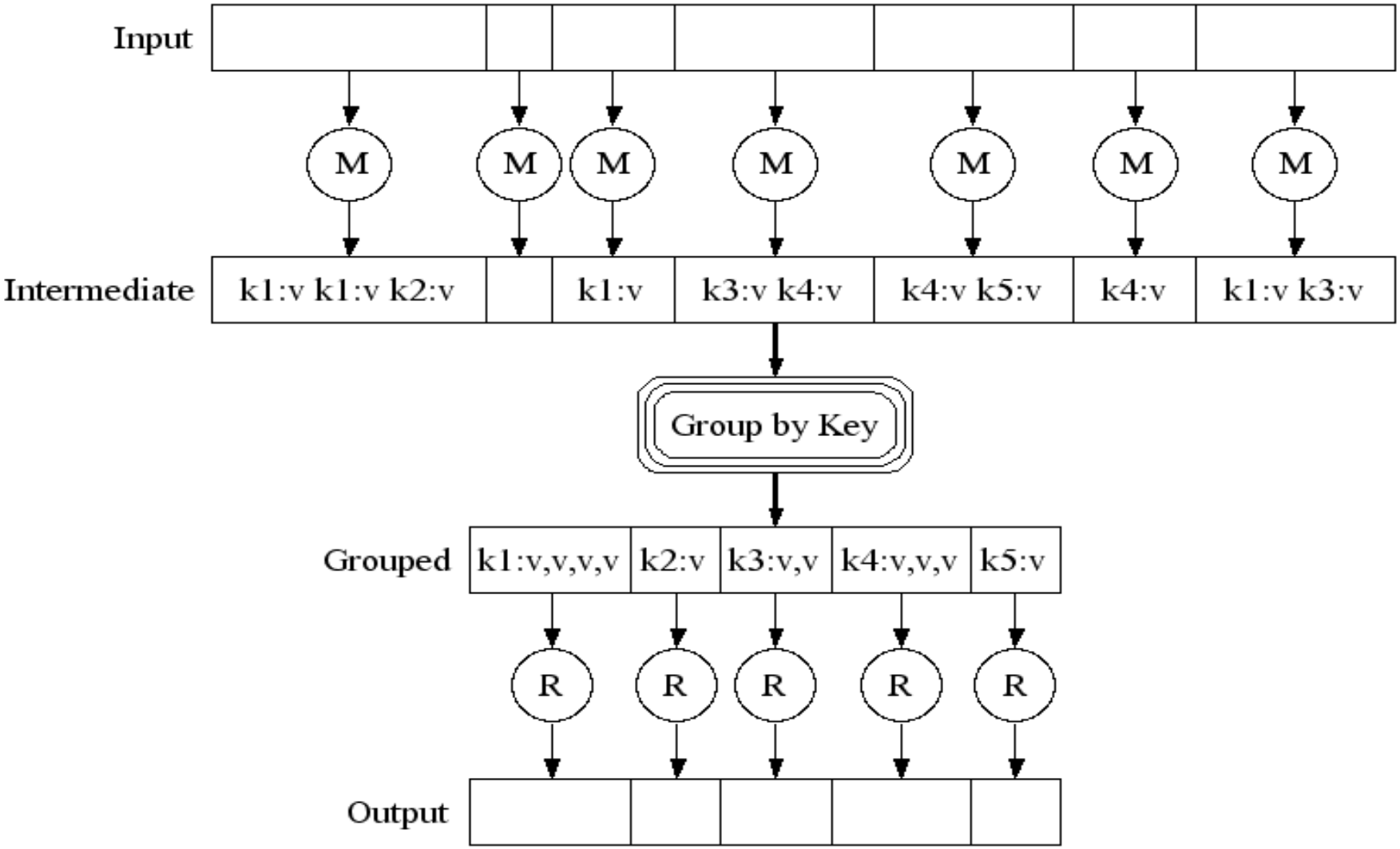
- **MapReduce** is a **programming model** for **data processing**.

- **MapReduce works** by **breaking** the **processing** into **two phases**:
  - ***map*** **phase** and the ***reduce*** **phase**.
  - **Each phase** has **key-value pairs** as **input** and **output**,
    - **Type** is **chosen** by the **programmer**.

Implementation: The **programmer specifies two functions**: the **map function** and the **reduce function**

Example: Word counting

- ”Consider the **problem** of **counting** the **number of occurrences** of **each word** in a **large collection of documents**”

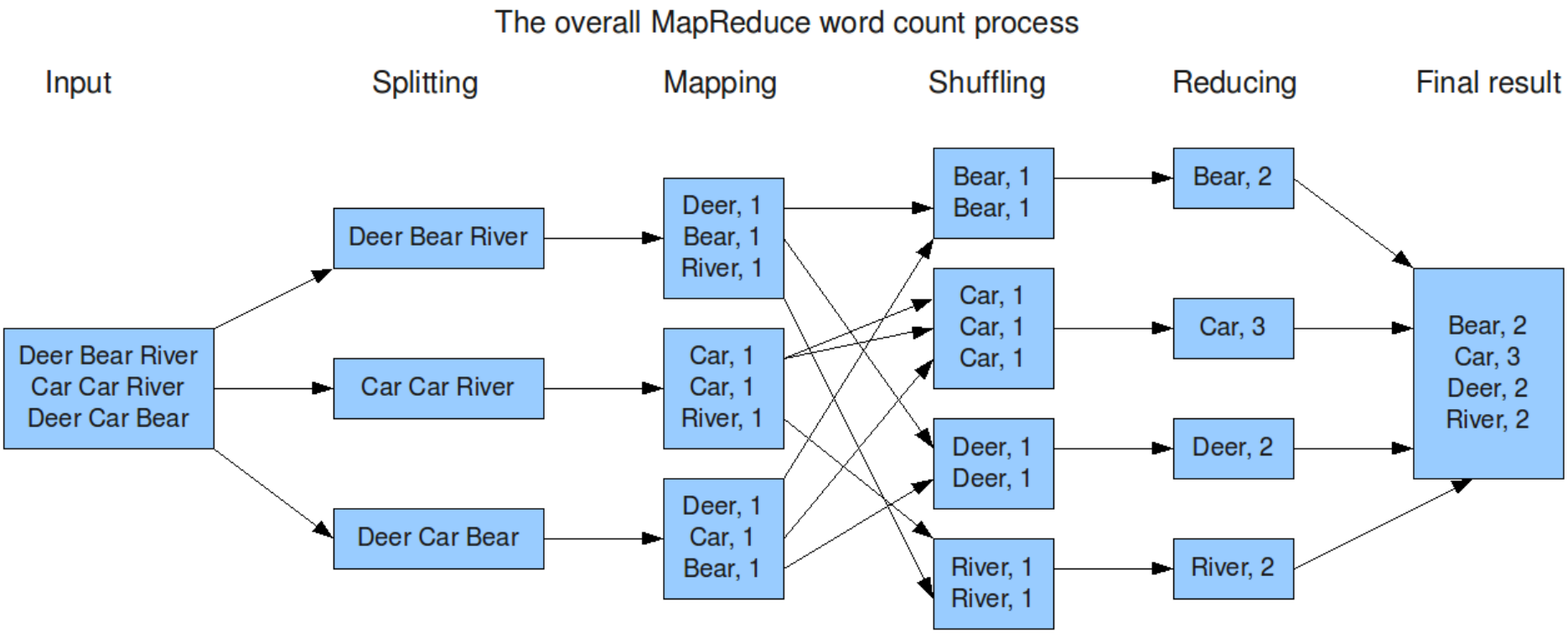

Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Word Count Example

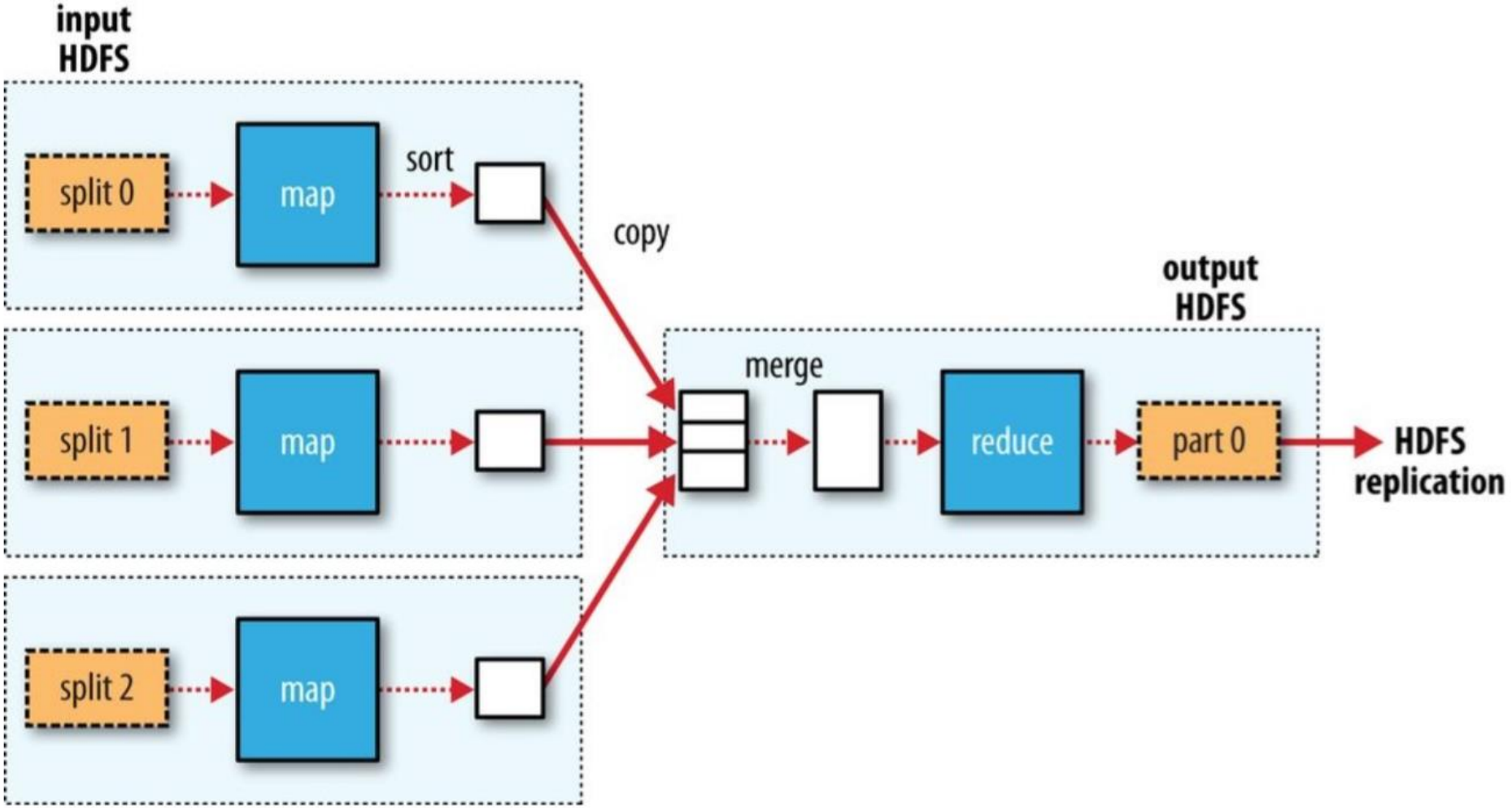
- **Mapper**
  - **Input**: ***key:*** **line offset,** ***value:*** **line of text**
  - **Output**: ***key:*** **word**, ***value:*** **1**

- **Reducer**
  - **Input**: ***key:*** **word**, ***value:*** **set of counts**
  - **Output**: ***key:*** **word**, ***value:*** **sum**

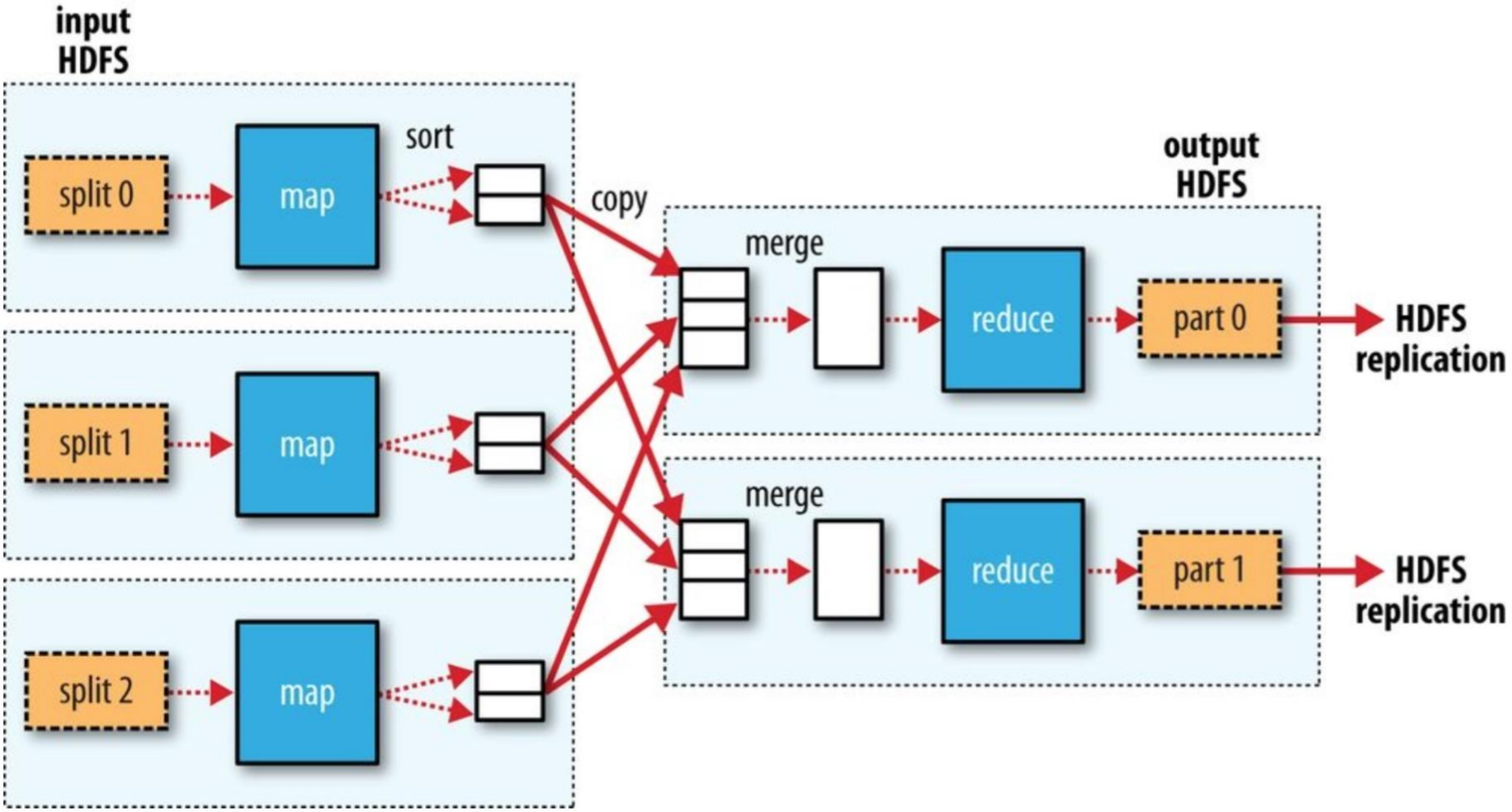
Word Count Dataflow



Data-flow with a single reduce task



Data-flow with multiple reduce tasks



A closer look

- **Input files: Can have different format**, usually **reside** on **HDFS**, tens of ***GBs in size***

- **InputFormat: How** these **input files are split** up and **read**. The **default InputFormat** is the ***TextInputFormat***.

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

A closer look

- **A MapReduce job** is a **unit** of **work** that the **client wants to be performed**: it consists of:
  1. the **input data**,
  2. the **MapReduce program**,
  3. and **configuration information**.

- **Hadoop runs** the **job** by **dividing** it into **tasks**, of **which there are two types**:
  - **map tasks** and **reduce tasks**.
  - **scheduled** using **YARN** and **run on nodes** in the **cluster**

A closer look

- **Hadoop divides** the **input** to a **MapReduce job into fixed-size pieces** called **input splits**, or just **splits**.

- **Hadoop creates one map task** for **each split**, which **executes user-defined map function** for **each record in the split**.

- **Splits size**, **parallelism**, and **overhead** ?

Data locality optimization

- **Hadoop** does its best to **run the map task** on a **node where** the **input data resides** in **HDFS**
  - To do **data locality optimization**.

- **Or** → **Next best solution**: **look** for a **free map slot** on **another node** **in the same rack**

- **Or** → **In worst case**: an **off-rack node** is **used**, which **results** in an **inter-rack network transfer**.

Typical Hadoop Cluster

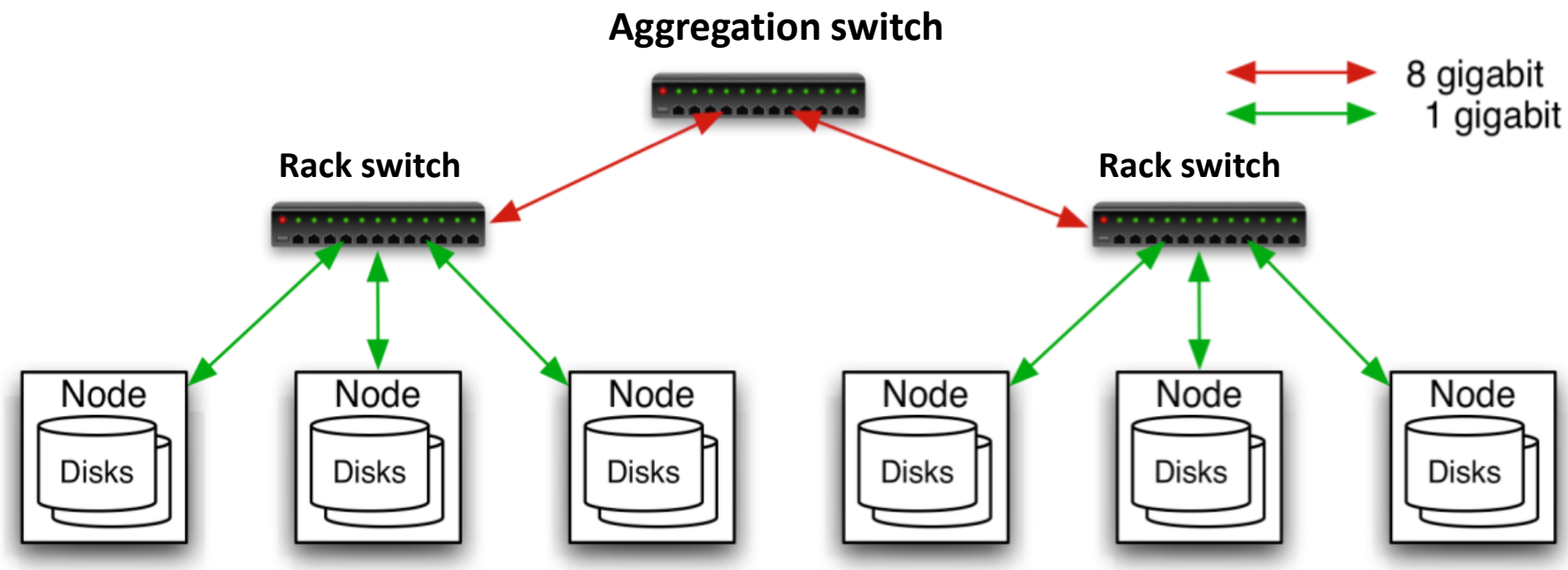
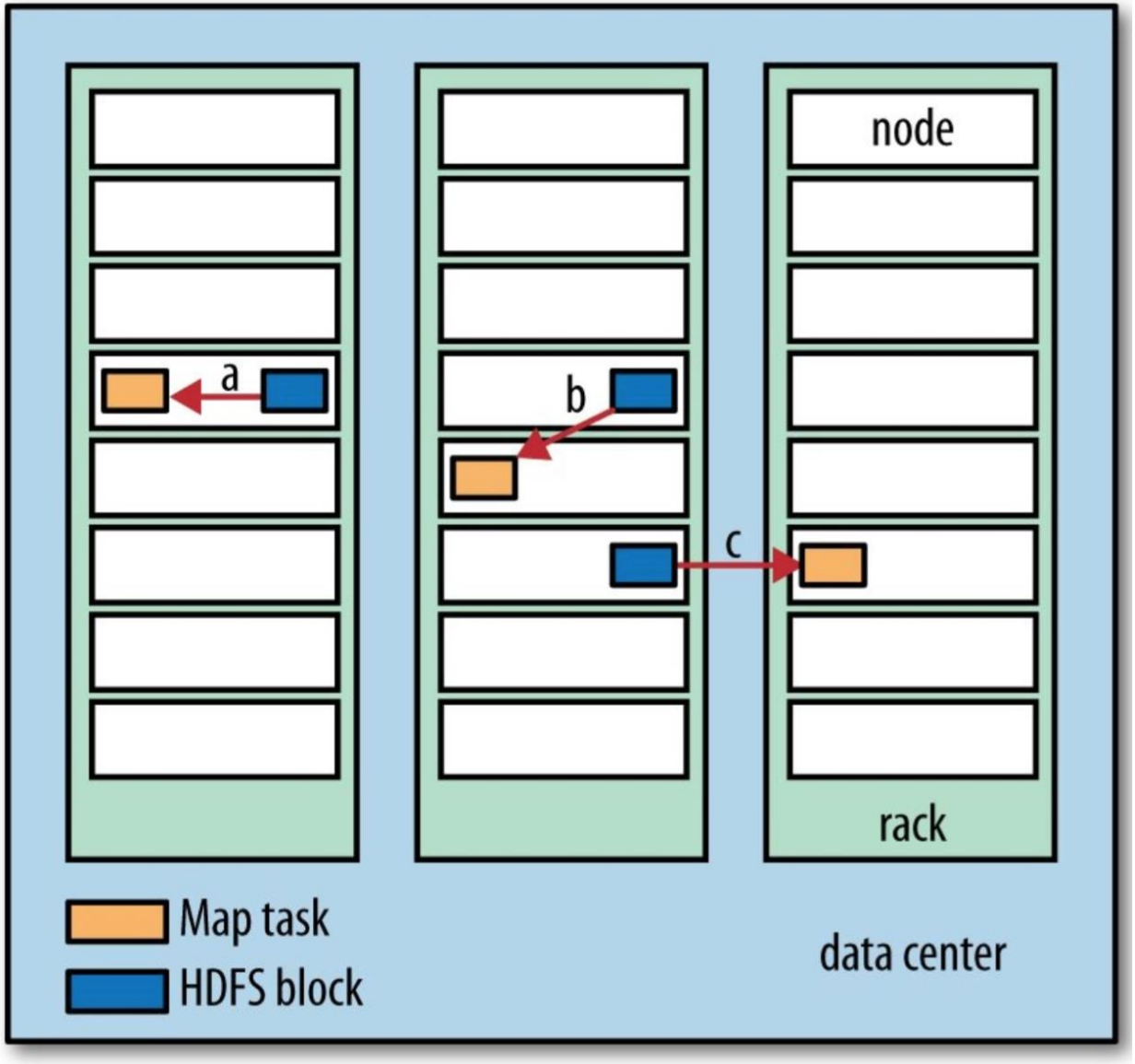


Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/YahooHadoopIntro-apachecon-us-2008.pdf

Data Locality Optimization



Reduce Tasks

- **Data Locality?**
  - **Reduce tasks don't have the advantage of data locality, because:**
    - **Input** to a **single reduce task** is **normally** the **output from all mappers**.

- The **reduce output** is **stored** in **HDFS** for **reliability**.

A real dataset

- **For our example:** we will **write** a **program** that **mines weather data.**

- **Weather sensors collect data every hour** at **many locations across** the **globe** and **gather a large volume** of log **data**

- A **good candidate** for **analysis** with **MapReduce**
  - **large semi-structured data**

Data Format

- The **data** we will use is from the **National Climatic Data Center**, or **NCDC**.

- The **data** is **stored** using a **line-oriented ASCII format**, in which **each line is a 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
```

Data Format

- **Datafiles** 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
```

Data Format

- There are **tens** of **thousands** of **weather stations**, so the **whole dataset** is made up of a **large number** of **relatively small files**.

- It's **generally easier** and **more efficient to process a smaller number of relatively large files**,
  - so the **data was preprocessed** so that each **year's readings were concatenated** into a **single file**.

Q. What's the highest recorded global temperature for each year in the dataset?

MapReduce

- **MapReduce** is a **programming model** for **data processing**.

- **MapReduce works** by **breaking** the **processing** into **two phases**:
  - **map phase** and the **reduce phase**.
  - **Each phase** has **key-value pairs** as **input** and **output**,
    - **Type** is **chosen** by the **programmer**.
  - The **programmer** also **specifies two functions**: the **map function** and the **reduce function**
  - **So? What should we do?**

How it can be done using Hadoop?

- **Some sample records …**

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

- **And or fields of interest**

```
6701199099999**1950**051507004…9999999N9+**0000**1+99999999999…)
0043011990999991950051512004…9999999N9+**0022**1+99999999999..
0043011990999991950051518004…9999999N9-**0011**1+99999999999..
0043012650999991949032412004…0500001N9+**0111**1+99999999999..
0043012650999991949032418004…0500001N9+**0078**1+99999999999..
```

The *map* Phase

- The **input** to our **map phase**?
  - **raw NCDC data**
  - **Shouldnt it be in Key/Value pair?**

- The **key** is the **line offset** from the **beginning** of the **file**, but as **we have no need for this**, **we ignore it**.

- **Each line** in the **dataset** is a **text value**.

The *map* Phase

- **What would the map phase do?**
  - **Produce** some **intermediate data** for **shuffle/reduce phase**
  - So **from** the **input line,** we **need to pull some values out**, which?
  - We **pull out** the **year** and the **air temperature**, because these are the **only fields** we are **interested** in.

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...
```

These **lines** are **presented** to the **map function** as the **key-value pairs**:

```
(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…)
```

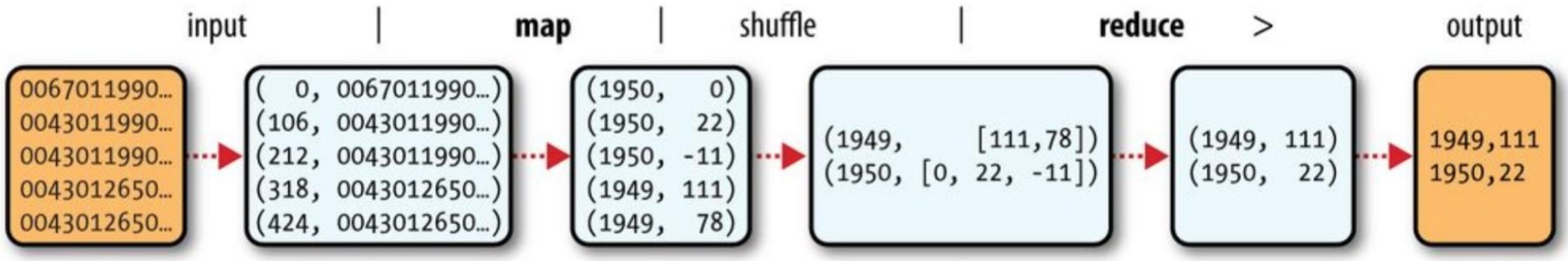
- The **map function** merely **extracts** the **year** and the **air temperature** (**integer**), and **emits** them **as its output**:

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

- The **output** from the **map function** is **processed** by the **MapReduce framework** before **being sent to the reduce function**. This processing **sorts and groups** the **key-value pairs** by **key**.

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

- **All** the **reduce function** has to do **now** is **iterate through** the **list** and **pick** up the **maximum reading**



Java MapReduce

- **We need three things**: a **map function**, a **reduce function**, and some **code** to **run the job**.

- The **map function** is **represented** by the **Mapper class**, which **declares** an **abstract map( ) method**.

- The **reduce function** is **similarly defined** using a **Reducer**

```java
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 MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, 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(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

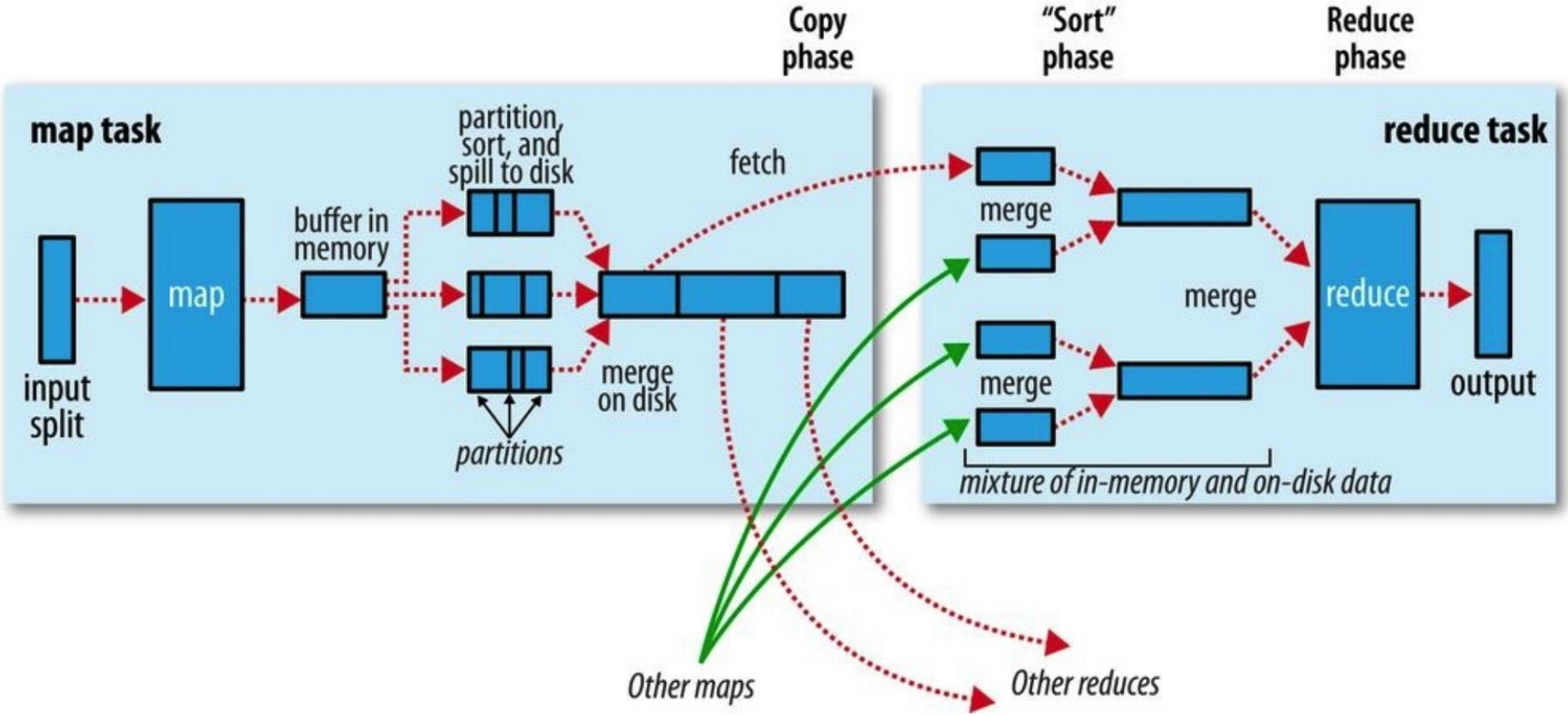
```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

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

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

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

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

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

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Number of reduce tasks

- The **number** of **reduce tasks** is **not governed** by the **size** of the **input**, but **instead** is **specified independently**.

- **When there are multiple reducers,**
  - the **map tasks partition** their **output**,
  - each **creating one partition for each reduce task**.
  - the **records** for **any given key** are **all in a single partition**.

A closer look



A refinement - Combiner

- **Many MapReduce jobs** are **limited** by the **bandwidth available** on the **cluster**

- It pays to **minimize** the **data transferred** between **map** and **reduce** **tasks**.

- **Hadoop allows** the **user to specify a combiner function** to be **run** on the **map output**, and the **combiner function's output** forms the **input** to the **reduce function**.

A refinement - Combiner Function

- **Combiner function** is an **optimization**
- **Hadoop does not provide a guarantee it invocation**

- **Calling** the **combiner function** ***zero***, ***one***, or ***many times*** should **produce** the **same output** from the **reducer**.

A refinement - Combiner Function

- **Assume** that for **our example**, **two maps process data** for **year 1950** (because they were in different splits). Imagine the **first map produced** the output: **(1950, 0), (1950, 20),(1950, 10)**

- And the **second split produced**, **(1950, 25), (1950, 15)**

- The **reduce function** would be **called with a list of all the values**: **(1950, [0, 20, 10, 25, 15])**

- **With output:** **(1950, 25)**

A refinement - Combiner Function

- **How to minimize** the **amount of data passed** to **Reduce**?

- Use a **combiner function** that, **just like the reduce function**, **finds** the **maximum temperature** for **each map output**.

max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25

Some Interesting Examples

Facebook friends using MapReduce

- **Facebook** has a **list of friends**
  - **bi-directional** thing, If **I'm your friend**, **you're mine.**
  - **You** and **me** have **230 friends** in **common**.
  - When you **visit someone's profile**, you **see a list of friends** that **you** have **in common**

We're going to use mapreduce so that we can calculate everyone's common friends…

Facebook friends using MapReduce

- Assume the **friends** are **stored** as **Person->[List of Friends]**, our **friends list** is then:

$A \rightarrow B\ C\ D$
$B \rightarrow A\ C\ D\ E$
$C \rightarrow A\ B\ D\ E$
$D \rightarrow A\ B\ C\ E$
$E \rightarrow B\ C\ D$

- **How it can be done?**
- **What would the Map phase do?**
- and **what would the reduce do?**

Facebook friends - The map phase

- **Each line** will be **an argument** to a **mapper**.
- For **every friend** in the **list of friends**, the **mapper** will **output a key-value pair**.
  - The **key** will be a **friend** along **with the person**.
  - The **value** will be the **list of friends**.
  - The **key will be sorted** so that the **friends are in order**,
  - **Causing** all **pairs of friends** to go to the **same reducer**.

Facebook friends - The map phase

For map(A → B C D) :

(A B) -> B C D
(A C) -> B C D
(A D) -> B C D

For map(B → A C D E) :

(A B) -> A C D E
(B C) -> A C D E
(B D) -> A C D E
(B E) -> A C D E

*Note that **A comes before B** in the **key***

Facebook friends - The map phase

For map(C → A B D E) :

(A C) -> A B D E
(B C) -> A B D E
(C D) -> A B D E
(C E) -> A B D E

For map(D → A B C E) :

(A D) -> A B C E
(B D) -> A B C E
(C D) -> A B C E
(D E) -> A B C E

For map(E → B C D):

(B E) -> B C D
(C E) -> B C D
(D E) -> B C D

The Sort/Shuffle (Group) Phase

- Before we **send** these **key-value pairs** to the **reducers**, we **group them** by their **keys** and **get**:

(A B) → (A C D E) (B C D)
(A C) → (A B D E) (B C D)
(A D) → (A B C E) (B C D)
(B C) → (A B D E) (A C D E)
(B D) → (A B C E) (A C D E)
(B E) → (A C D E) (B C D)
(C D) → (A B C E) (A B D E)
(C E) → (A B D E) (B C D)
(D E) → (A B C E) (B C D)

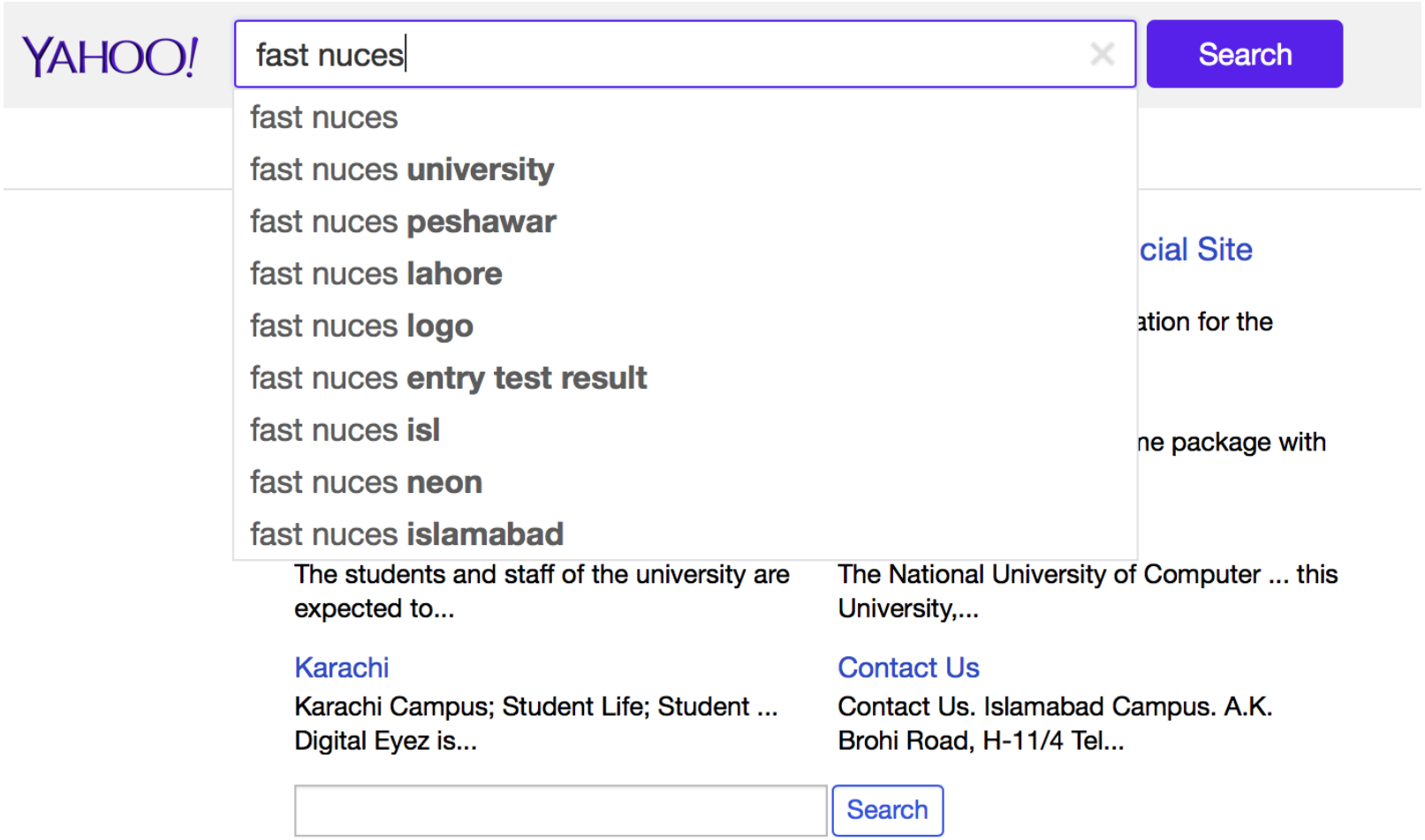
Facebook friends - The reduce phase

- Each **line** will be **passed as an argument** to a **reducer**.
- It will **intersect** the **lists** of **values** and **output the same key** with the **result** of the **intersection**.
- For example, **reduce((A B) -> (A C D E) (B C D))** will **output (A B) : (C D)** and **means** that **friends A** and **B have C and D** as **common friends**.

Facebook friends - Final outcome

(A B) → (C D)
(A C) → (B D)
(A D) → (B C)
(B C) → (A D E)
(B D) → (A C E)
(B E) → (C D)
(C D) → (A B E)
(C E) → (B D)
(D E) → (B C)

Yahoo Search Suggestions



Yahoo Search Suggestions

- **How does it work?**

- **Basic Idea - Related concepts appear close together**

Yahoo Search Suggestions

- **Input: Web pages**
  - **10 billions pages** having **10K bytes each** - **100TB** of input **data**
- **Output**: **List of words** and for **each word list of related words** - **List (Word, List(Related Words)**
- Not overly complicated to do that!!

How would it Work?

- **Map** - **Tokenize words** and **output pairs**
  - <word, nextWord>
  - <word, previousWord>

- **Group by Words** to have **<word, list(relatedWords)>**

- **Reduce** - **Add** **WordCount** to **output**

  <word, list(relatedWords), count>

Any Question?