POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Week 4-2

Hadoop II: MapReduce Programming

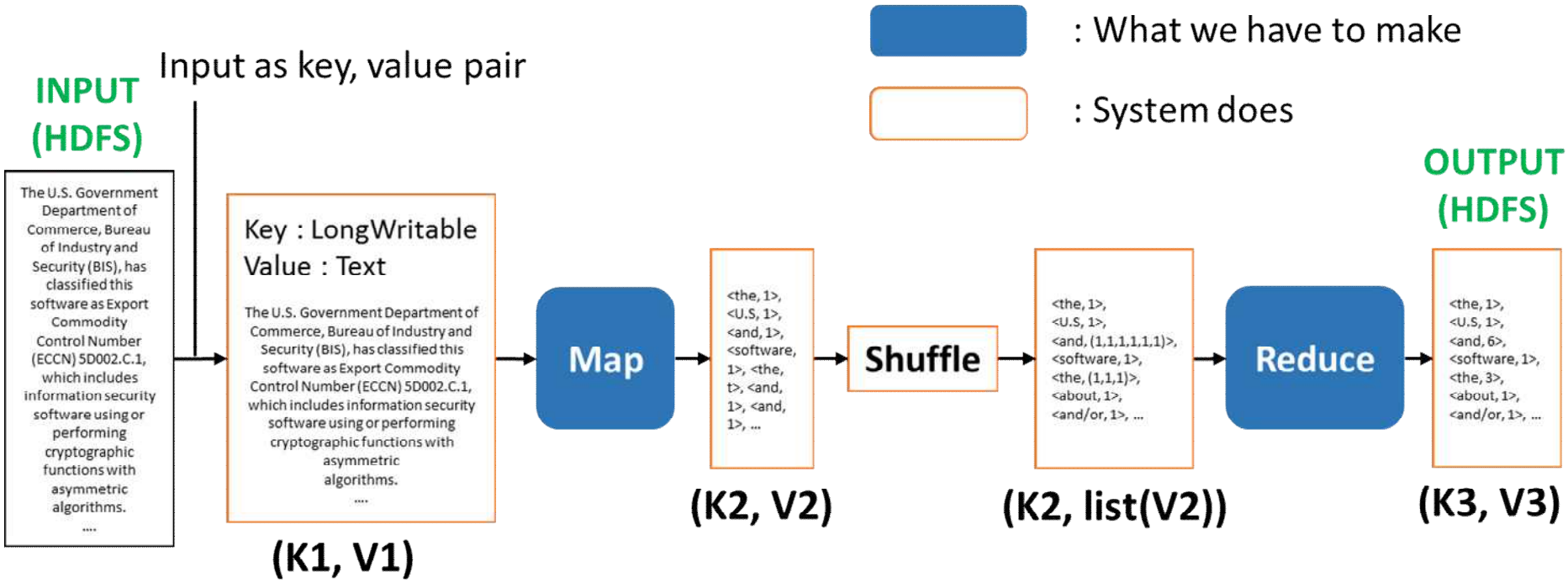
Big Data

Prof. Hwanjo Yu
POSTECH

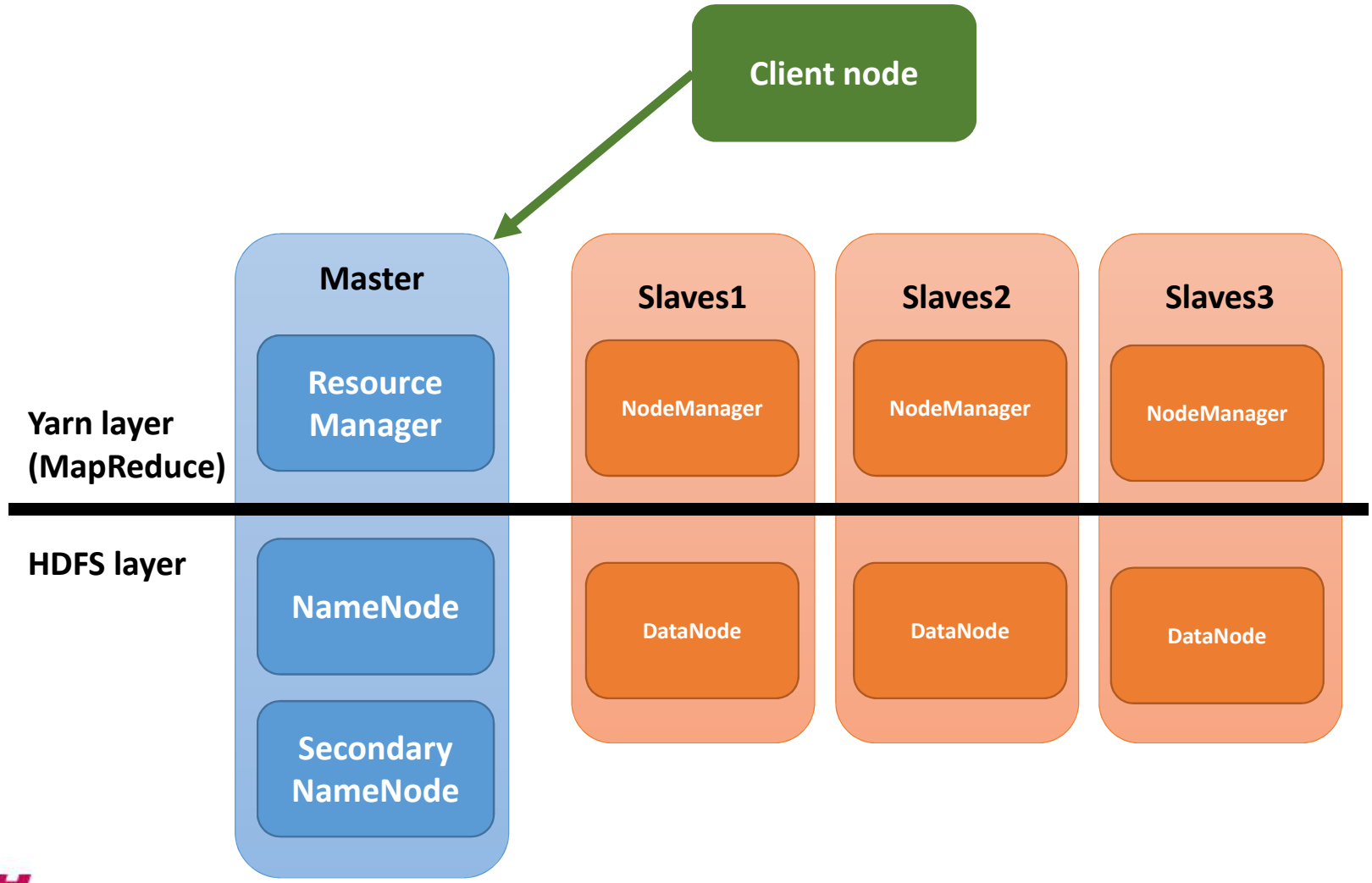
MapReduce

- Software framework for easily writing applications

  which process *vast amounts of data* (multi-terabyte data-sets) *in-parallel* on large clusters



Overall system



Word count example

- mkdir ~/wc_example
  - Make word count example directory

- cp /usr/local/hadoop-2.8.1/README.txt ~/wc_example/
  - Copy README.txt file to wc_example directory

- Make sample code WordCount.java
  - Copy WordCount Example code from next slide

Word count example code

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

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }


  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Compile

- $HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main WordCount.java
  - Compile and Create Classes file(WordCount, Mapper, Reducer)
- jar cf wc.jar WordCount*.class
  - Integrate required packages and Class file ➔ wc.jar file

- *hadoop jar ~/wc_example/wc.jar WordCount input/input.txt output*
  - Execute Hadoop program

Should not be exist prior

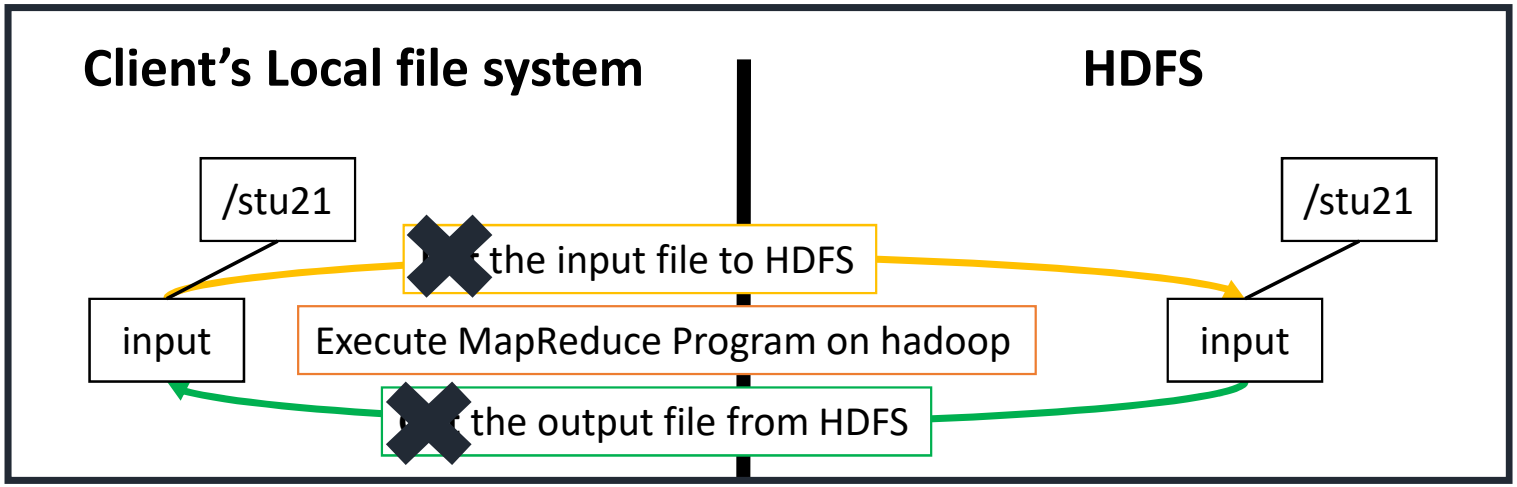
Input file in HDFS

Output directory which will store output file

- *hdfs dfs -ls output*
  - Check output folder
- *hdfs dfs -cat output/part-r-00000*
  - Read output file
- *hdfs dfs -get output/part-r-00000 output.txt*

Output file from HDFS

Copy the file to local



Structure

```
import some packages
public class WordCount{
    public static class MyMapper extends Mapper{
        public void map(Object key, Text value, Context context){
                        ...
        }
    }
    public static class MyReducer extends Reducer{
        public void reduce(Text key, Iterable<IntWritable> value, Context context){
                        ...
        }
    }
    public static void main(String [ ] args) throws Exception{
        initialize configuration
        create Job instance
        setting classes using Job instance

        execute the program
    }
}
```

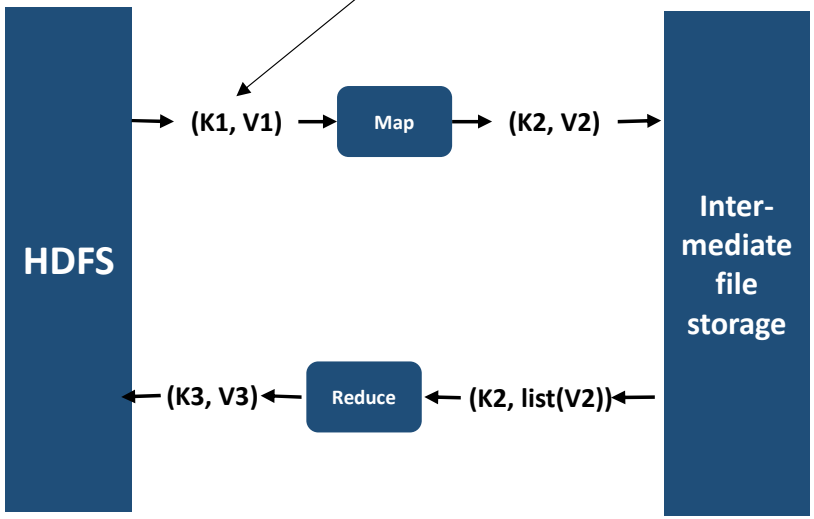
WritableComparable
(Writable + Comparable)

Provide a way to produce results

Writable
(serialization, deserialization)

TextInputFormat : default

- For text file
- One pair by one line
- Key : file offset (IntWritable)
- Value : string in line (Text)



Mapper - Overview

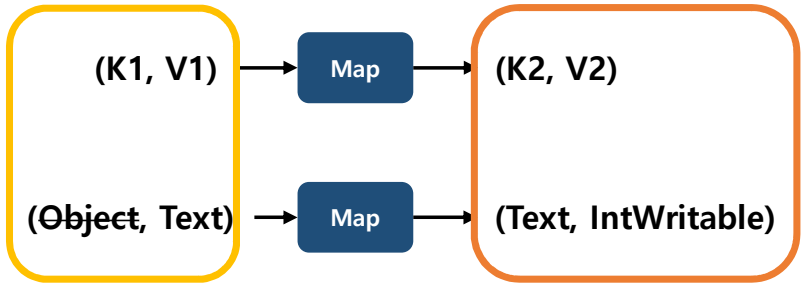
```
public static class MyMapper extends Mapper<K1, V1, K2, V2>{

        K2 k2 = new K2();

        V2 v2 = new V2();


        public void map(K1 key, V1 value, Context context) {

                …

                context.write(k2, v2);

        }
}
```

Mapper - Sample

Input type (k,v) Output type (k,v)

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);

    private Text word = new Text();

    public void map(Object key, Text value, Context context)
                    throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

For efficiency

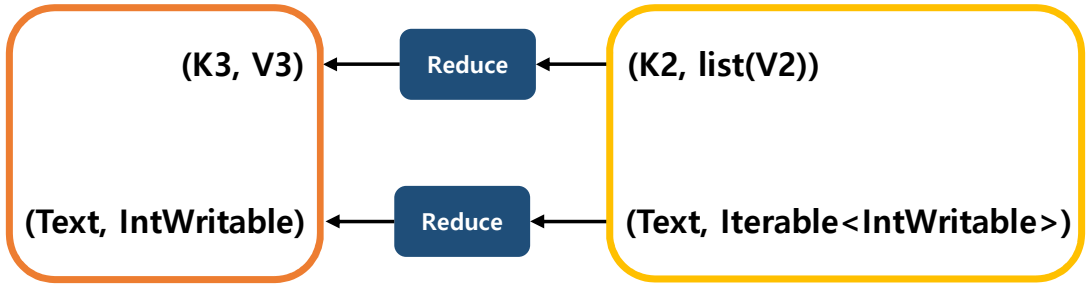


Reducer - Overview

```
public static class MyReducer extends Reducer<K2, V2, K3, V3> {

        K3 k3 = new K3();

        V3 v3 = new V3();


        public void reduce(K2 key, Iterable<V2> values, Context context) {

                // handle the values by iterating loops

                context.write(k3, v3);

        }

}
```

Reducer - Sample

```
public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {

        private IntWritable result = new IntWritable();


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

                int sum = 0;

                for (IntWritable val : values) {

                        sum += val.get();

                }

                result.set(sum);

                context.write(key, result);

        }

}
```



Main function

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

        //job.setInputFormatClass(TextInputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        //job.setOutputFormatClass(TextOutputFormat.class);

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

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

Job name

Local Reducer or Mini Reducer

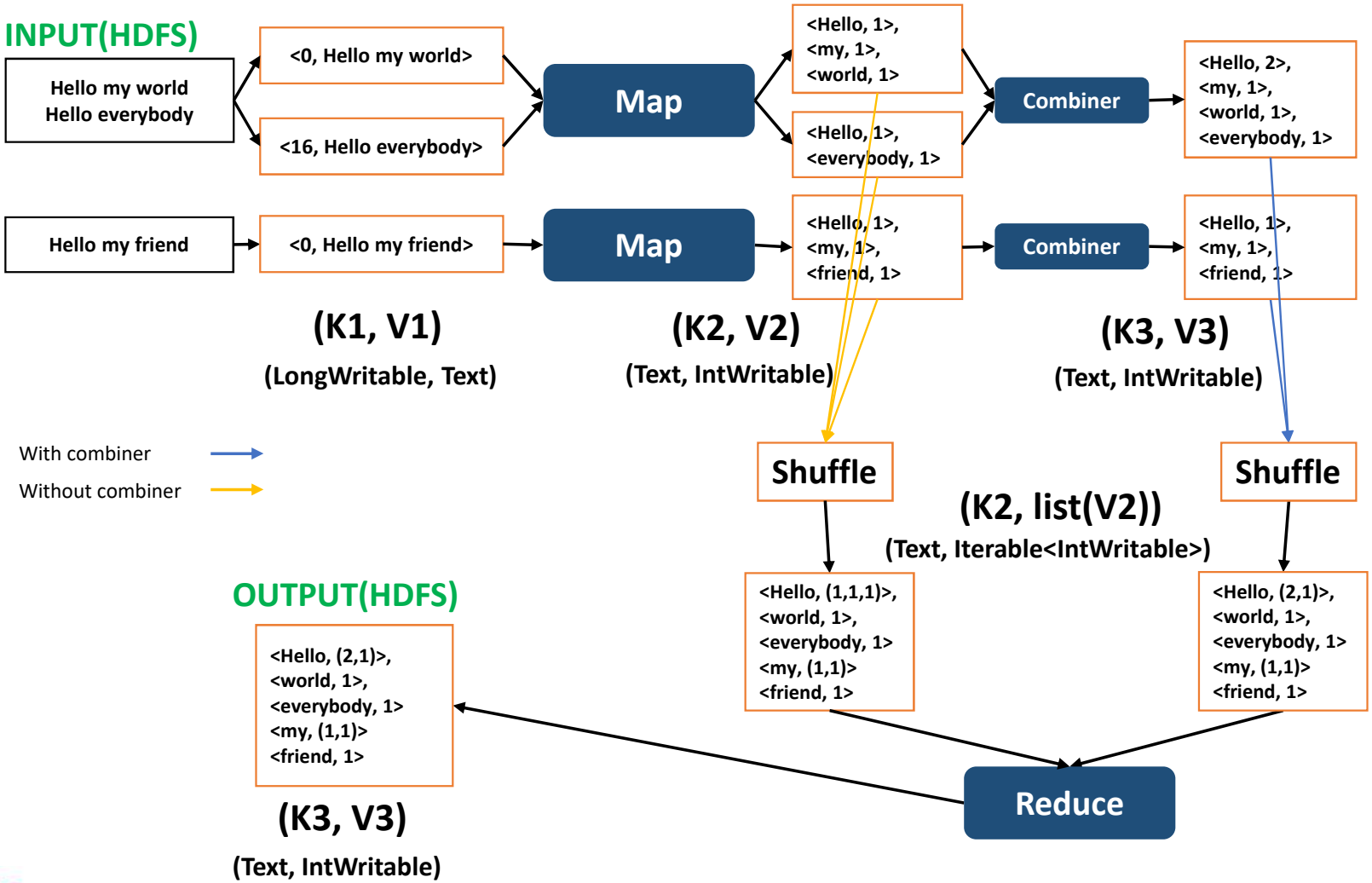
Why use it ??
To reduce input data of reducer (reduce network traffic)

Can we always use the combiner??
No!! Most job cannot use combiner

Final(Reducer) output key/value

What about Mapper's output??
Present explicitly only in case that output type of Map, Reduce are different

Start the job
Blocking function
(Not returned until job is completed)


INPUT(HDFS)
Hello my world
Hello everybody
<0, Hello my world>
<16, Hello everybody>
Map
<Hello, 1>,
<my, 1>,
<world, 1>
<Hello, 1>,
<everybody, 1>
Combiner
<Hello, 2>,
<my, 1>,
<world, 1>,
<everybody, 1>
Hello my friend
<0, Hello my friend>
Map
<Hello, 1>,
<my, 1>,
<friend, 1>
Combiner
<Hello, 1>,
<my, 1>,
<friend, 1>
(K1, V1)
(LongWritable, Text)
(K2, V2)
(Text, IntWritable)
(K3, V3)
(Text, IntWritable)
With combiner
Without combiner
Shuffle
Shuffle
(K2, list(V2))
(Text, Iterable<IntWritable>)
<Hello, (1,1,1)>,
<world, 1>,
<everybody, 1>
<my, (1,1)>
<friend, 1>
<Hello, (2,1)>,
<world, 1>,
<everybody, 1>
<my, (1,1)>
<friend, 1>
OUTPUT(HDFS)
<Hello, (2,1)>,
<world, 1>,
<everybody, 1>
<my, (1,1)>
<friend, 1>
(K3, V3)
(Text, IntWritable)
Reduce

Primitive types for Hadoop

- Text → String + WritableComparable
- IntWritable → Integer + WritableComparable
- LongWritable → Long + WritableComparable
- FloatWritable → Float + WritableComparable
- BooleanWritable → Boolean + WritableComparable
- ArrayWritable → Array + WritableComparable
- NullWritable → null + WritableComparable
- …

http://hadoop.apache.org/docs/current/api/org/apache/hadoop/io/package-summary.html
http://hadoop.apache.org/docs/current/api/overview-summary.html

Input format

- Determine
  - How to recognize the input file as?

    (How to read, and What type is the file??)
  - How to separate the file into the InputSplit

    (small piece of the file, unit of Map task)
  - How to separate (read) the InputSplit into the key and the value

Usually a block becomes a InputSplit
➔ the number of blocks == the number of Map task

TextInputFormat : **Default**

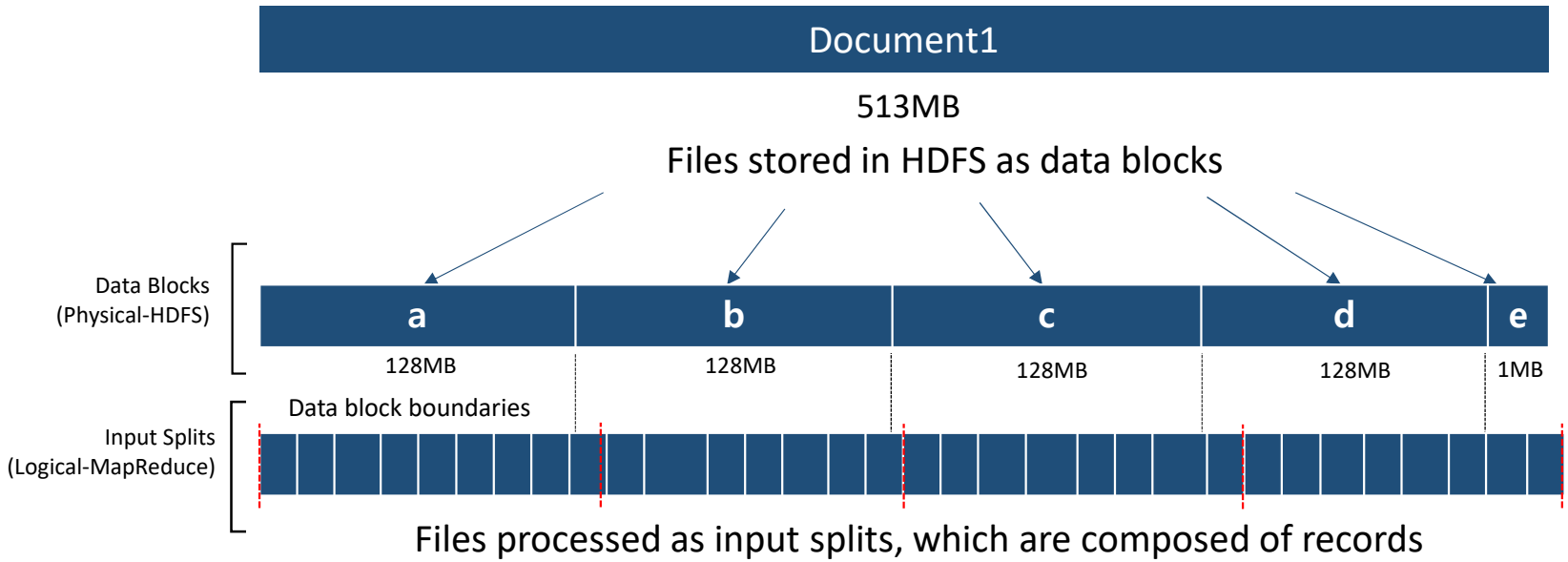
- For text file
- One pair by one line
- **Key** : file offset (LongWritable)
- **Value** : string in line (Text)

KeyValueTextInputFormat

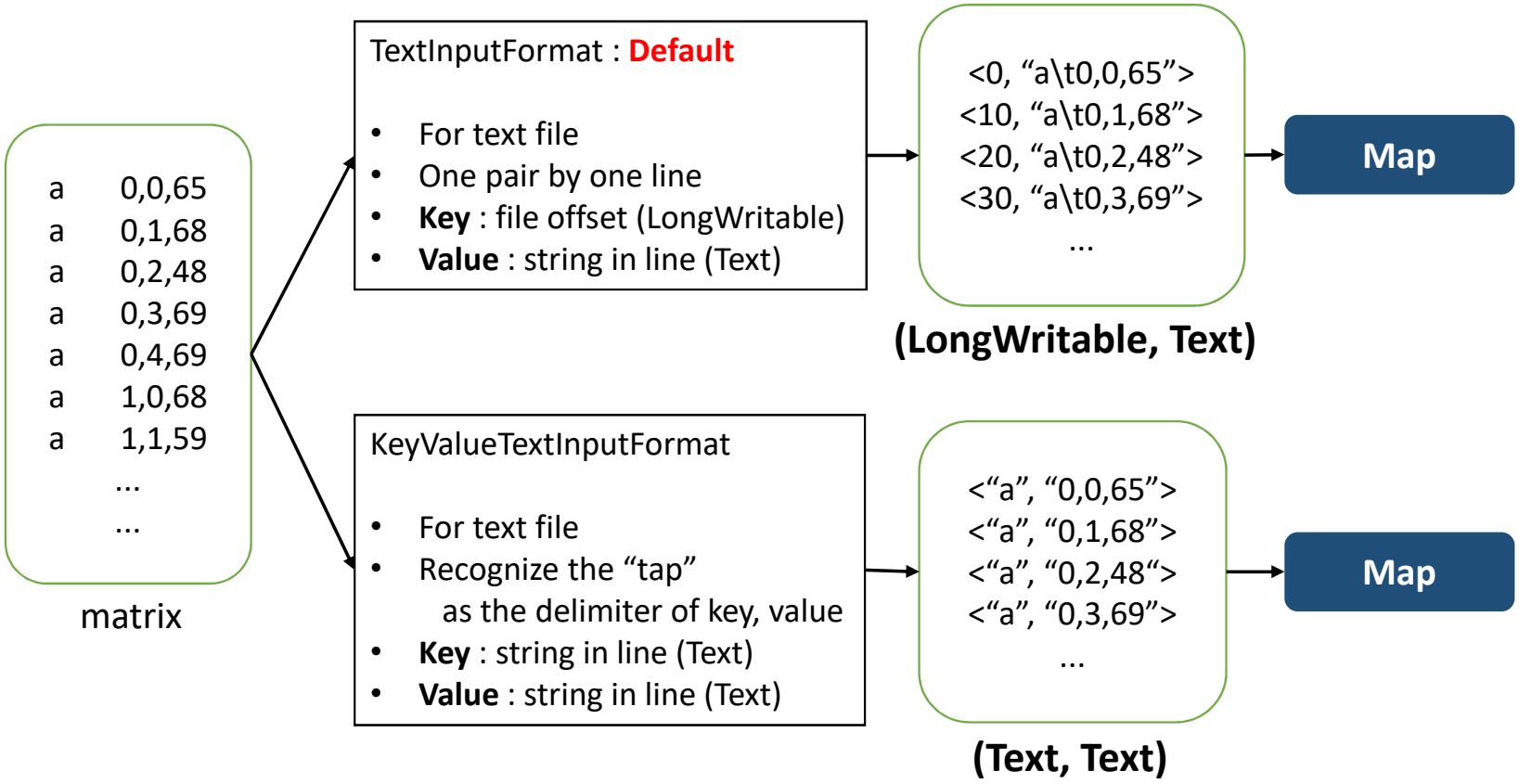
- For text file
- Recognize the "tap" as the delimiter of key, value
- **Key** : string in line (Text)
- **Value** : string in line (Text)

http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/package-summary.html

Input format – Cont'd



Input format – Cont'd



Inverted index examples

ID	Text
1	Baseball is played during summer months.
2	Summer is the time for picnics here.
3	Months later we found out why.
4	Why is summer so hot here

↑ Sample document data

Dictionary and posting lists →

Term	Freq	Document ids
baseball	1	[1]
during	1	[1]
found	1	[3]
here	2	[2], [4]
hot	1	[4]
is	3	[1], [2], [4]
months	2	[1], [3]
summer	3	[1], [2], [4]
the	1	[2]
why	2	[3], [4]

lucid IMAGINATION

http://www.slideshare.net/erikhatcher/introduction-to-solr-9213241

Simple inverted index example

hdfs dfs -rm -r output/invertedIndex1

Get Source code and input file

source code: https://drive.google.com/open?id=0B-ZY5QWwVNT_WmRQTHFnaDhWVnM

input file: https://drive.google.com/open?id=0B-ZY5QWwVNT_UVBWbzJKR2NEbHM

$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main InvertedIndex1.java

jar cf invertedIndex1.jar InvertedIndex1*.class

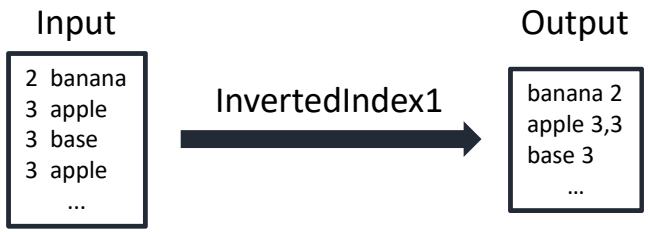
hdfs dfs –put document input/document

hadoop jar invertedIndex1.jar InvertedIndex1 input/document output/invertedIndex1

hdfs dfs -cat output/invertedIndex1/part-r-00000

Simple inverted index code

```
public static class InvertedIndexMapper extends Mapper<Text, Text, Text, Text> {
        public void map(Text docID, Text term, Context context) throws IOException, InterruptedException {
                context.write(term, docID);
        }
}

public static class InvertedIndexReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text term, Iterable<Text> docIDs, Context context) throws IOException, InterruptedException {

                StringBuilder documents = new StringBuilder();
                boolean first = true;

                for (Text docID : docIDs) {
                        if (!first)
                                documents.append(",");
                        else
                                first = false;

                        documents.append(docID.toString());
                }
                context.write(term, new Text(documents.toString()));
        }
}
```



Setup method (for your homework)

```
public void setup(Mapper.Context context) {



}
```

- *Called once at the beginning of the task*
  - Before map or reduce function
- Default : Do nothing
- You can customize what you want.

Cleanup method (for your homework)

```
public void cleanup(Mapper.Context context) {



}
```

- *Called once at the end of the task*
  - After map or reduce function
- Default : Do nothing
- You can customize what you want.

HW 2 - Hadoop MapReduce

- Please check the attached zip file.
- Input and template files are in the attached zip file.
- Unzip the attached zip file
  - `tar xvzf hw2_templates`

- You must…
  - Utilize the **MapReduce Framework**
  - **Test** on the ***Hadoop 2.8.1*** (Compilation, Execution), Compile error ➔ Your score is Zero.
  - Please read lecture note and **comments in template codes carefully**
  - **Do it Yourself**

HW 2.1 - Inverted Index

- Goal
  - Implement Inverted index for Multiple Files in a directory.

- Submission
  - `InvertedIndex.java`

HW 2.1 - Inverted Index

- Note
  - Input files are in the attached zip file.
  - Your program must read file names in a given directory.
  - Related files
    - `hw2_templates/InvertedIndex/*`
  - ***Inverted Index template file is not provided.***

- Tips
  - Use setup() and context in the Mapper to get the file name.
    - setup() is called once for each Map task (for each chunk) while map() is called multiple times (for each line).
  - Write a loop in the main() to add all input files in the directory to the job.

HW 2.2 - Join

- Goal
  - Implement a relational join as a MapReduce query

```
SELECT *
FROM order, line_item
WHERE order.order_id = line_item.order_id
```

- Submission
  - `Join.java`

HW 2.2 - Join

- Note
  - There are data, output samples, template files in Attached files
  - Related files
    - `hw2_join_templates/Join/*`

HW 2.2 - Join

- Input Data
  - `records`
  - There are “order” table and “line_item” table.
  - First column: table name
    Second column: order_id,
    Fifth column of “line_item” table: item_id
- InputFormat
  - `TextInputFormat`
- Output Data
  - `part-r-00000`
- Program parameter
  - 5 parameters
  - [Input file] [output path] [table names delimited by ,] [# of join index of first table] [# of join index of second table]

HW 2.2 - Join

- Constraint
  - Don’t touch main function and input/output key/value type
  - You can use some data structure for Caching in Reduce task
  - Result records are concatenated with records from “Order” and records from “line_item”

HW 2.2 - Join (Test?)

$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Join.java

jar cf join.jar Join*.class

hadoop jar join.jar Join **/user/input/records output/join order,line_item 1 1**

hdfs dfs -cat output/join/part-r-00000

Parameters

➔ **I'll check and score the homework in similar way!**

HW 2.3 - Matrix Multiplication

- Goal
  - Implement a MapReduce algorithm to compute the matrix multiplication a*b and a*b*c

- Submission
  - There must be 3 implemented source code files in compressed file.
    - `Multiplication1_1.java`
    - `Multiplication1_2.java`
    - `Multiplication2.java`

HW 2.3 - Matrix Multiplication

- Note
  - There are data, output samples, template files in the attached zip file.
  - Related files
    - `hw2_templates/MatrixMultiplication/*`

HW 2.3 - Matrix Multiplication

- Input Data
  - `matrix1_1, matrix1_2, matrix2`
  - a : **3 X 5**, b : **5 x 2**, c : **2 x 3**
  - Matrix name and element information is *separated by 'Tap'*

    matrix i,j,value
  - *It's sparse matrix format.*

    If the value is 0, than the related record is not presented explicitly

- Output Data
  - `output/multiple1_1/part-r-00000`
  - `output/multiple1_2/part-r-00000`
  - `output/multiple1_2/final/part-r-00000`
  - `output/multiple2/part-r-00000`

HW 2.3 - Matrix Multiplication

- Constraint
  - Don't touch main function and input/output key/value type
  - In Multiplication1_2.java, you must utilize Multiplication1_1 class. It means that you should implement Multiplication1_1 first before implementing Multiplication1_2
  - In Multiplication1_2.java, job1's output must be input of Matrix1_2_1_Mapper. If you don't modify the main function, this procedure works properly
  - Result must be the same as the given solution
    - Final result of Multiplication1_2 should be located in 'final' directory of the directory of intermediate result
    - Form of Multiplication1_1, Multiplication1_2 and Multiplication2's output are the same [row,column value] (separated by tap)

HW 2.3 - Matrix Multiplication1_1

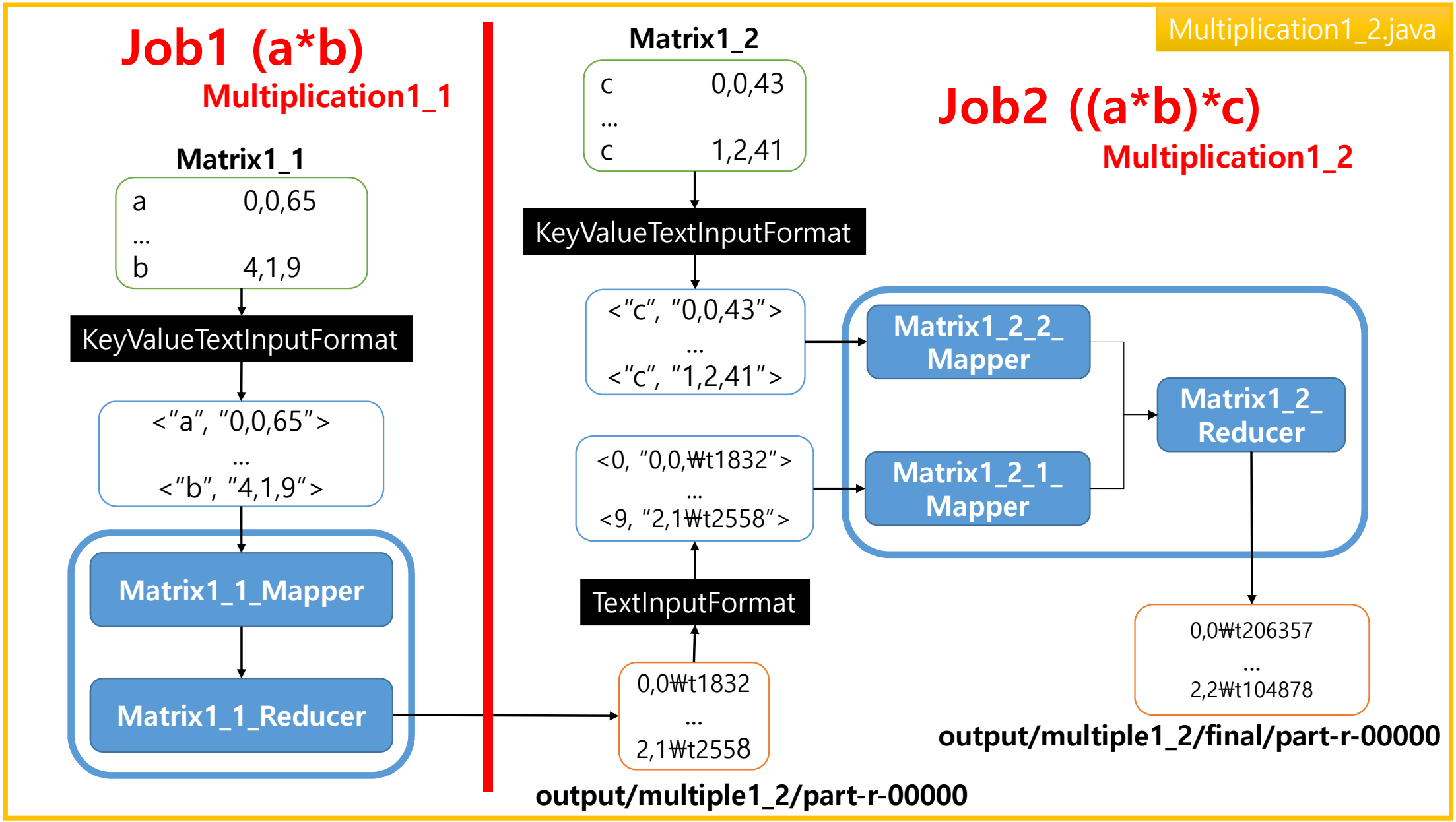
- Goal
  - Implement a MapReduce algorithm to compute the multiplication of **"two"** matrix which ***comes from single file***
- Source code
  - `Multiplication1_1.java`
- Input Data
  - `matrix1_1`
- InputFormat
  - `KeyValueTextInputFormat`
- Output Data
  - `output/multiple1_1/part-r-00000`
- Program parameter
  - 5 parameters
  - [Input file] [output path] [# of first matrix's rows] [# of first matrix's columns] [# of second matrix's columns]

HW 2.3 - Matrix Multiplication1_2

- Goal
  - Implement a MapReduce algorithm to compute the multiplication of **"three"** matrix which ***comes from two different files.***
  - Program compute a*b first using Multiplication1_1, and compute (a*b)*c using the algorithm in Multiplication1_2 ***sequentially***
- Source code
  - `Multiplication1_2.java`
  - *Multiplication1_1.java must be implemented beforehand*
- Input Data
  - `matrix1_1, matrix1_2`
- Output Data
  - `output/multiple1_2/part-r-00000`
  - `output/multiple1_2/final/part-r-00000`

HW 2.3 - Matrix Multiplication1_2

- MultipleInputs class and Multiple mappers
  - Because the program will *use two input files* which are different in format, program should use the MultipleInputs class
  - *Two different InputFormat is used*
  - *Two different Mapper is used* as well
- Program parameter
  - 7 parameters
  - [Input file1] [Input file2] [output path] [# of first matrix's rows] [# of first matrix's columns] [# of second matrix's columns] [# of third matrix's columns]
  - Intermediate result (a*b) will be written in [output path] and final result (a*b*c) will be written in [output path/final]


Job1 (a*b)
Multiplication1_1
Matrix1_1
a 0,0,65
...
b 4,1,9
KeyValueTextInputFormat
<"a", "0,0,65">
...
<"b", "4,1,9">
Matrix1_1_Mapper
Matrix1_1_Reducer
Matrix1_2
c 0,0,43
...
c 1,2,41
KeyValueTextInputFormat
<"c", "0,0,43">
...
<"c", "1,2,41">
<0, "0,0,₩t1832">
...
<9, "2,1₩t2558">
TextInputFormat
0,0₩t1832
...
2,1₩t2558
output/multiple1_2/part-r-00000
Multiplication1_2.java
Job2 ((a*b)*c)
Multiplication1_2
Matrix1_2_2_ Mapper
Matrix1_2_1_ Mapper
Matrix1_2_ Reducer
0,0₩t206357
...
2,2₩t104878
output/multiple1_2/final/part-r-00000

HW 2.3 - Matrix Multiplication2

- Goal
  - Implement a MapReduce algorithm to compute the multiplication of **"three"** matrix which ***comes from single file.***
- Source code
  - `Multiplication2.java`
- Input Data
  - `matrix2`
- Output Data
  - `output/multiple2`
- Program parameter
  - 6 parameters
  - [Input file] [output path] [# of first matrix's rows] [# of first matrix's columns] [# of second matrix's columns] [# of third matrix's columns]

HW 2.3 - Matrix Multiplication (Test?)

Put these 3 files in home directory (or sub directory of home) in the server's file system and compile the files , make jar file and run the 'jar' file like following commands

1. Multiplication1_1

$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Multiplication1_1.java

jar cf multiple1_1.jar Multiplication1_1*.class

hadoop jar multiple1_1.jar Multiplication1_1 **/user/input/matrix1_1 output/multiple1_1 3 5 2**

hdfs dfs –cat output/multiple1_1/part-r-00000

2. Multiplication1_2 (complies Multiplication1_1.java and Multiplication1_2.java simultaneously)

$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Multiplication1*.java

jar cf multiple1_2.jar Multiplication1*.class

hadoop jar multiple1_2.jar Multiplication1_2 **/user/input/matrix1_1 /user/input/matrix1_2 output/multiple1_2 3 5 2 3**

hdfs dfs -cat output/multiple1_2/part-r-00000

hdfs dfs -cat output/multiple1_2/final/part-r-00000

HW 2.3 - Matrix Multiplication (Test?)

3. Multiplication2

$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Multiplication2.java

jar cf multiple2.jar Multiplication2*.class

hadoop jar multiple2.jar Multiplication2 **/user/input/matrix2 output/multiple2 3 5 2 3**

hdfs dfs -cat output/multiple2/part-r-00000

Parameters

➔ **The homework will be graded in a similar way!**

```
A =

    65    68    48    69    69
    68    59     0    44     0
    43     0    69    70    64

>> B

B =

     1    14
     6     7
     1    20
     2     0
    17     9

>> C

C =

    43    32     0
    43    40    41

>> A*B*C

ans =

      206357      177304      121647
       80625       70920       55965
      167614      145200      104878

>> D=A*B

D =

        1832        2967
         510        1365
        1340        2558
```

Figure. Matlab example