Google big data techniques

Lecturer: Jiaheng Lu

Fall 2016

- MapReduce model (this lecture)
- Google File System (next lecture)
- Bigtable data storage platform (next lecture)

Introduction to MapReduce

MapReduce: Insight

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

- Use Count-min sketch for approximation.

- But if we need the accurate value, how would you do it in parallel ?

Simple example: Word count

(Trump)

(Donald Trump)

Mapper
(1-2)

Reducer
(A-G)

(Trump Clinton)

(President election)

Mapper
(3-4)

Reducer
(H-N)

(USA President)

(Donald Trump)

Mapper
(5-6)

Reducer
(O-U)

1 Each mapper receives some of documents as input

Simple example: Word count

(Trump)

(Donald Trump)

Mapper
(1-2)

(Trump, 1)

(Donald, 1), (Trump, 1)

Reducer
(A-G)

(Trump Clinton)

(President election)

Mapper
(3-4)

(Trump, 1), (Clinton, 1)

(President,1),(election, 1)

Reducer
(H-N)

(USA President)

(Donald Trump)

Mapper
(5-6)

(USA, 1), (President, 1)

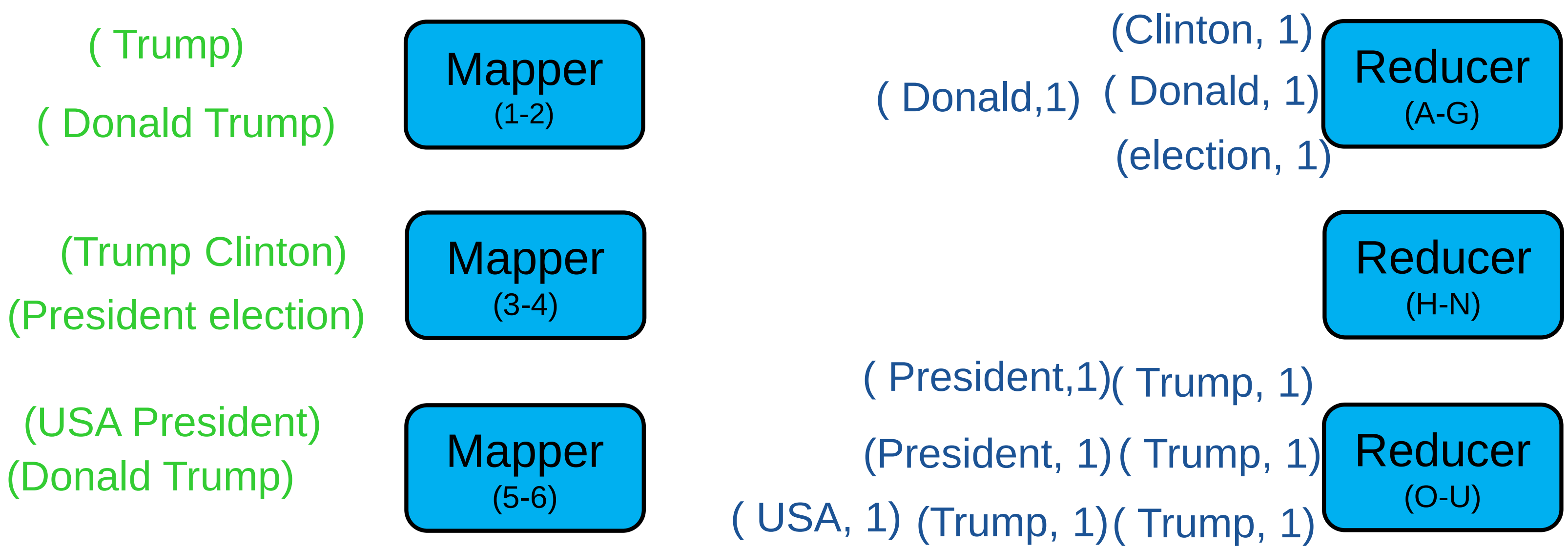
(Donald,1),(Trump, 1)

Reducer
(O-U)

1 Each mapper receives some of documents as input

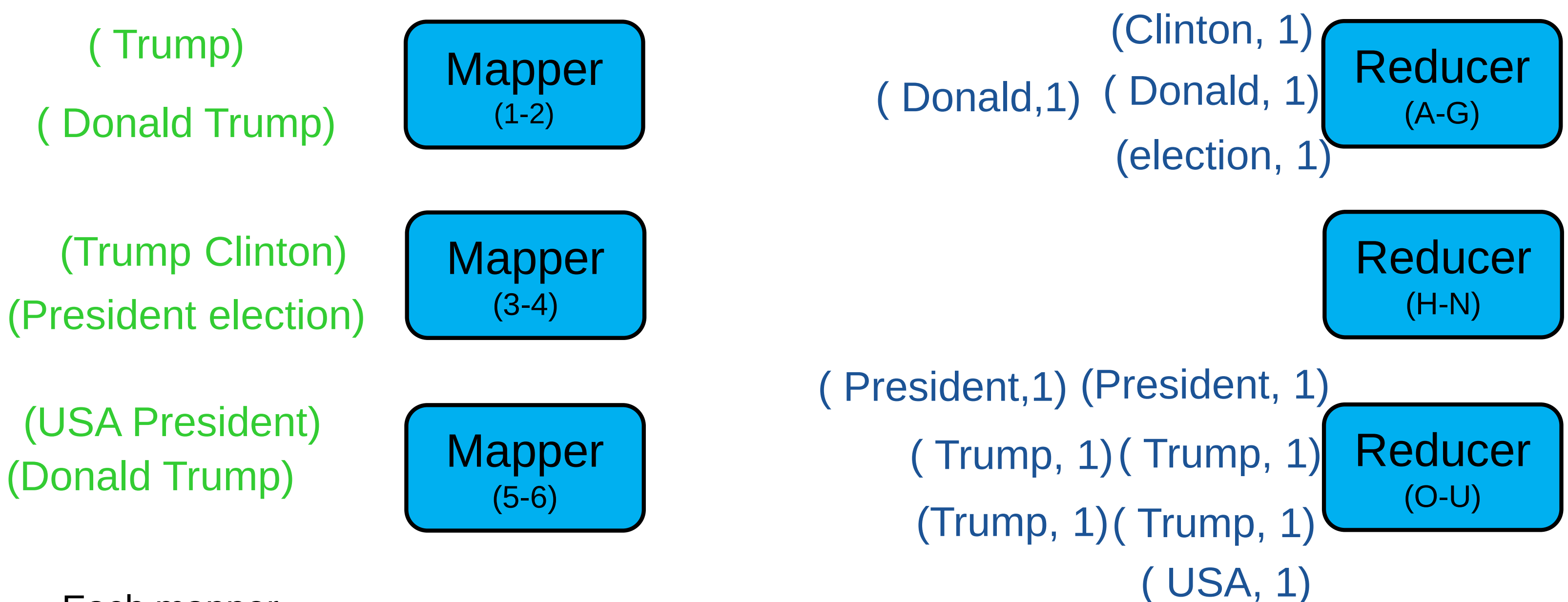
2 Mappers process the KV-pairs.

Simple example: Word count



1. Each mapper receives some of documents as input
2. Mappers process the KV-pairs.
3. Each KV-pair output by the mapper is sent to the reducer

Simple example: Word count



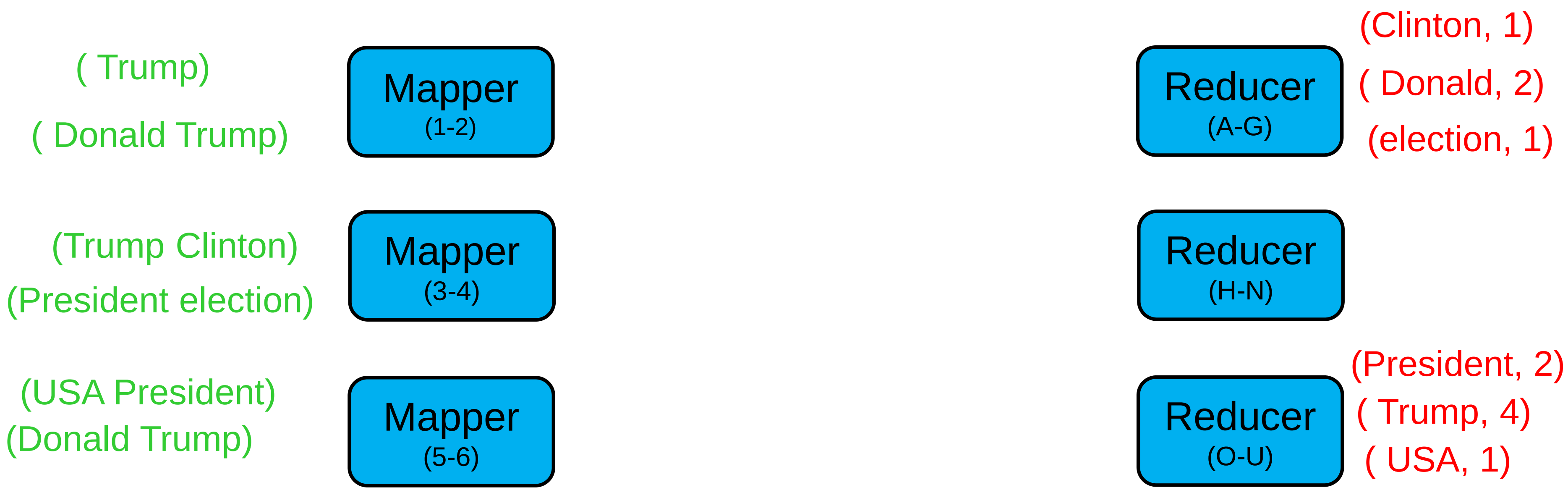
1 Each mapper receives some of documents as input

2 Mappers process the KV-pairs.

3 Each KV-pair output by the mapper is sent to the reducer

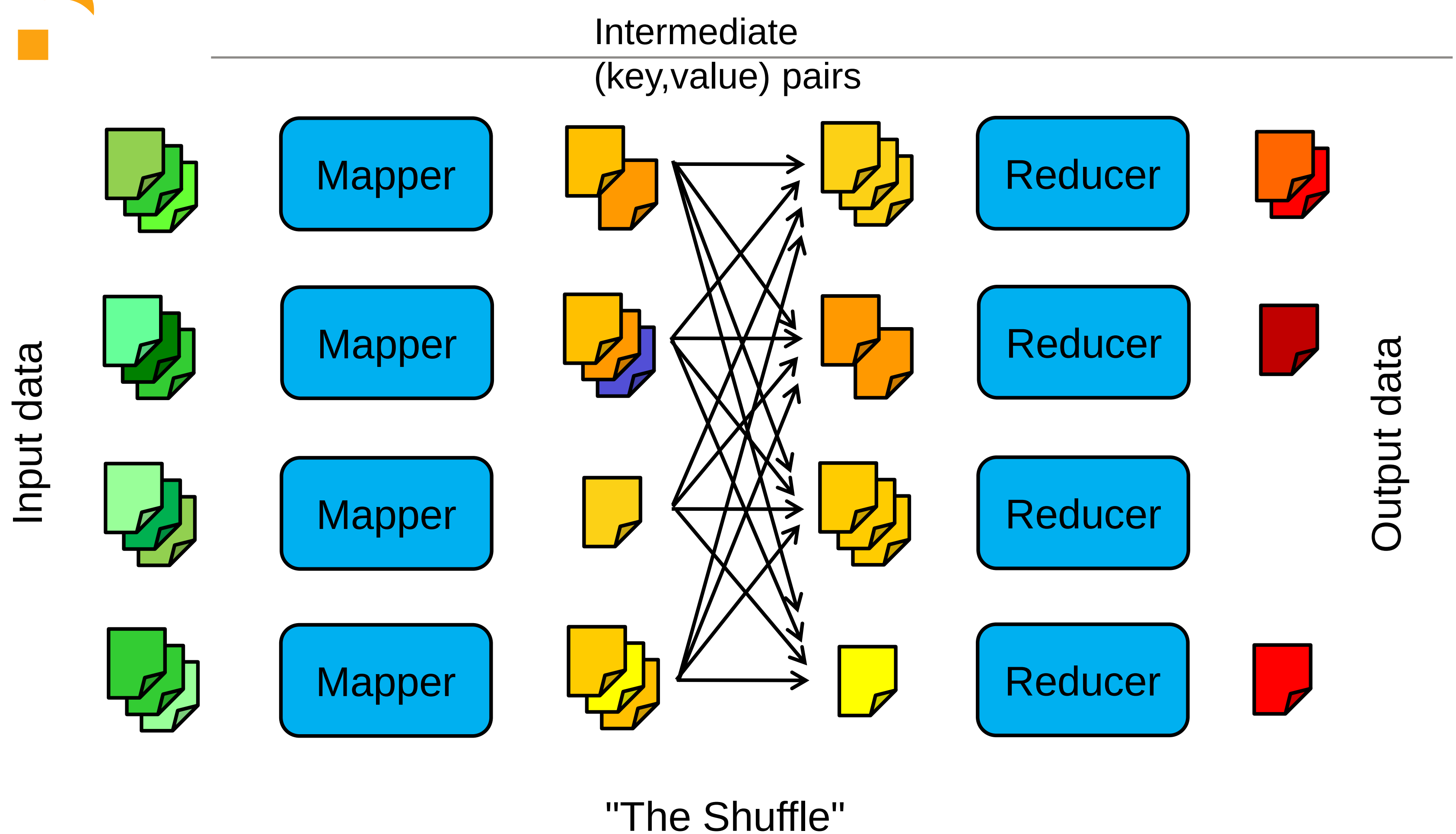
4 The reducers sort their input by key

Simple example: Word count



1. Each mapper receives some of documents as input
2. Mappers process the KV-pairs.
3. Each KV-pair output by the mapper is sent to the reducer
4. The reducers sort their input by key
5. The reducers process their input

MapReduce dataflow

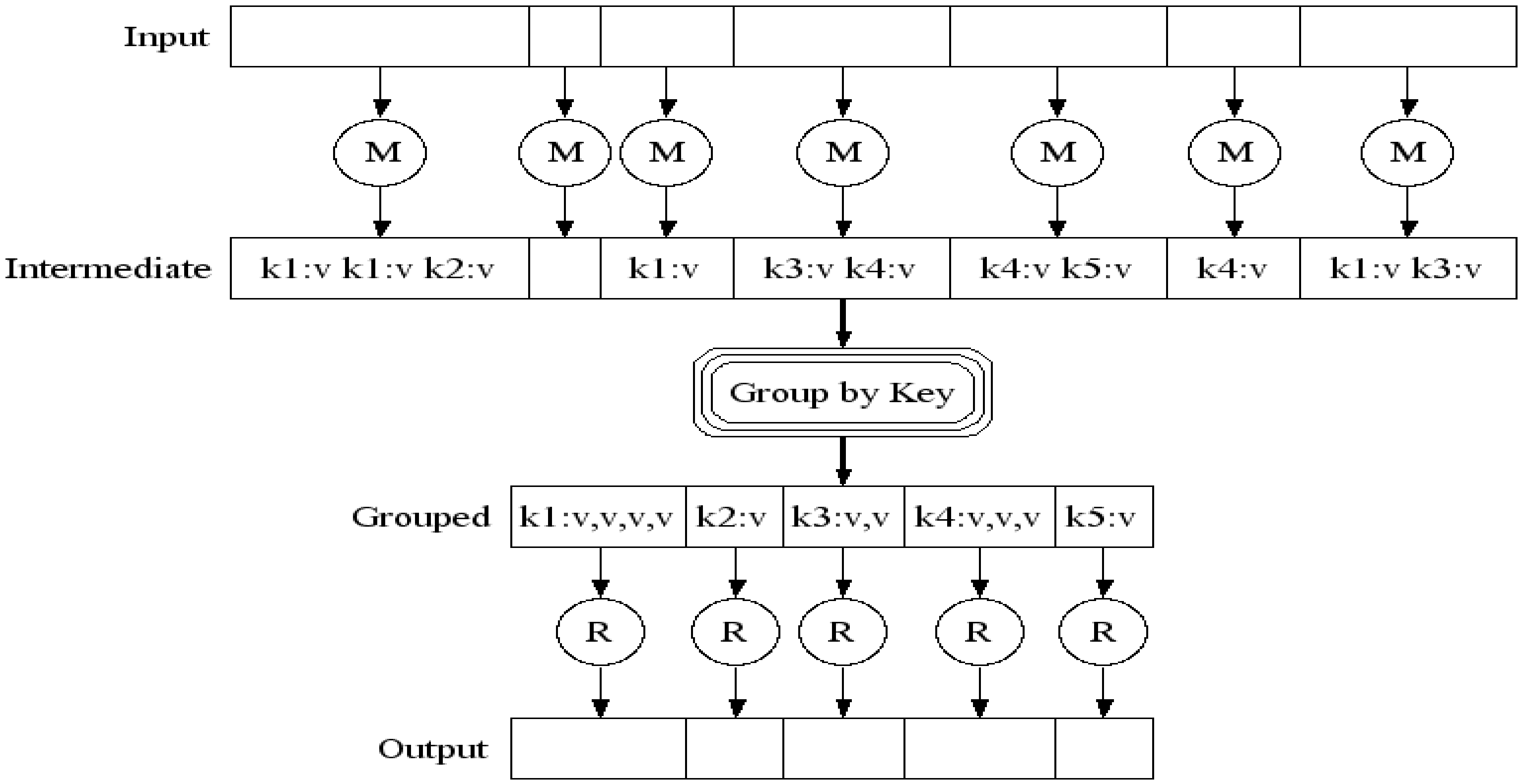


Pseudo-code

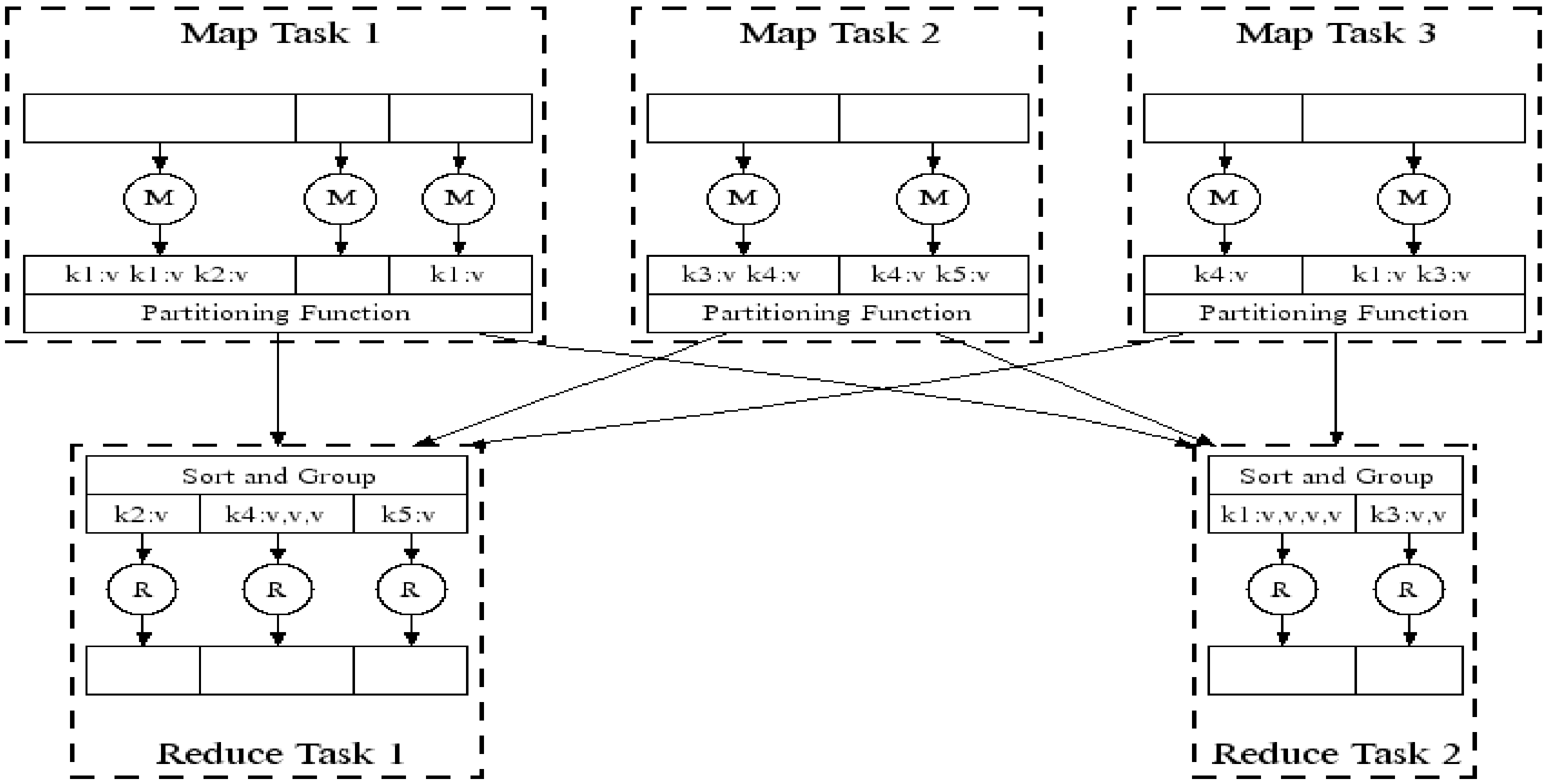
```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

MapReduce: Example



MapReduce in Parallel: Example



More examples

- Count URL access frequency
  - Map: output each URL as key, with count 1
  - Reduce: sum the counts
- Reverse web-link graph
  - Map: output (target,source) pairs when link to target found in source
  - Reduce: concatenates values and emits (target,list(source))
- Inverted index
  - Map: Emits (word,documentID)
  - Reduce: Combines these into (word,list(documentID))

Avoid the following common mistakes on MapReduce programs!

Common mistakes: Use static variables

- Don't use static shared variables for mappers
- After `map` + `reduce` return, they should remember nothing about the processed data!

```
HashMap h = new HashMap();
map(key, value) {
  if (h.contains(key)) {
    h.add(key,value);
    emit(key, "X");
  }
}
```

Wrong!

Common mistakes: Do your own I/O

- Don't try to do your own I/O!
  - Don't try to read from, or write to, files in the file system
  - The MapReduce framework does all the I/O for you:
    - All the incoming data will be fed as arguments to map and reduce
    - Any data your functions produce should be output via emit

```
map(key, value) {
  File foo =
    new File("xyz.txt");
  while (true) {
    s = foo.readLine();
    ...
  }
}
```

Wrong!

Common mistakes: Too much data on the same key

- Mapper must not map too much data to the same key
  - In particular, don't map *everything* to the same key!!
  - Otherwise the reduce worker will be overwhelmed!
  - It's okay if some reduce workers have more work than others
    - – Example: In WordCount, the reduce worker that works on the frequent key has a lot more work than the reduce worker that works on the rare key

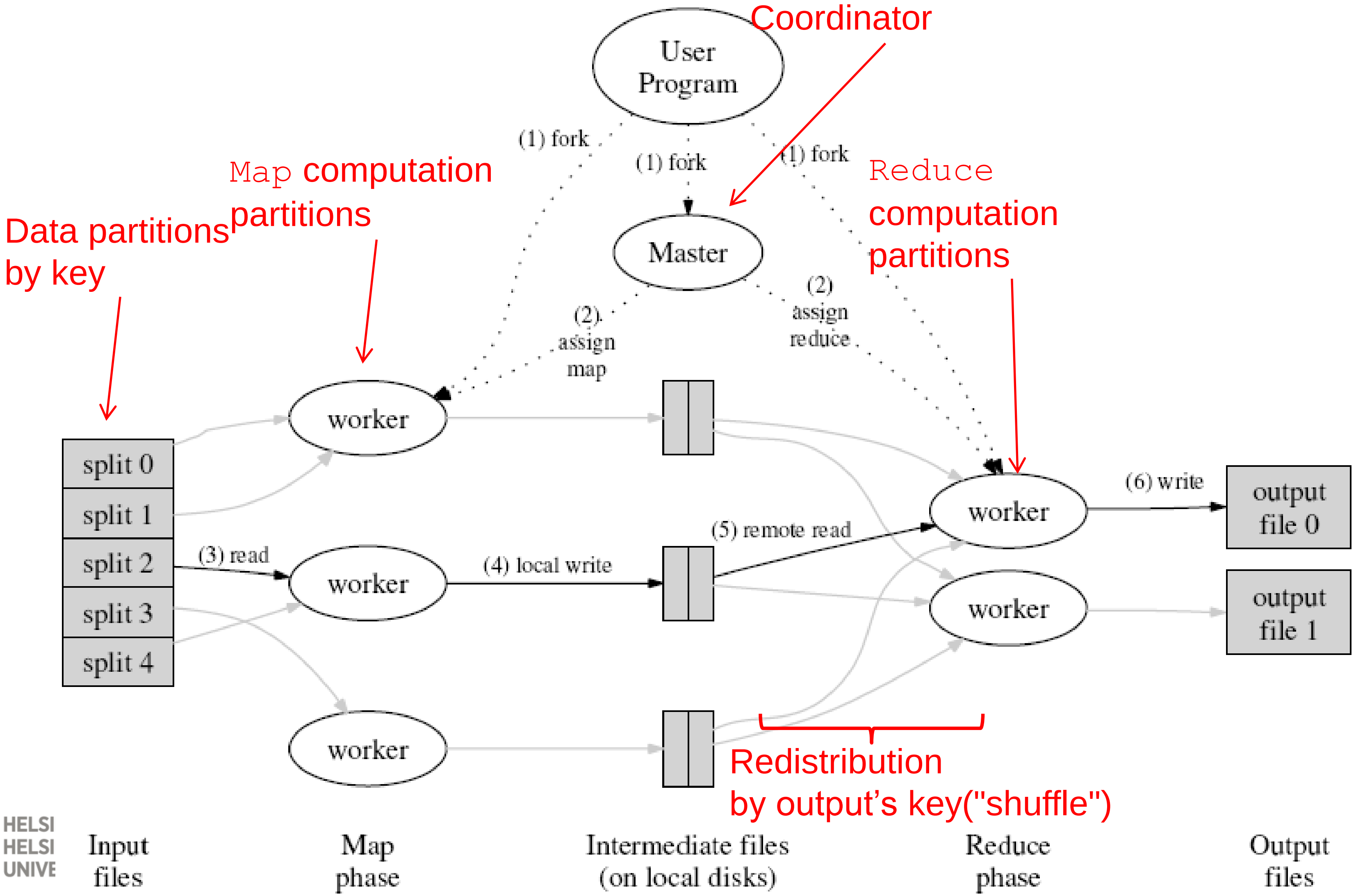
```
map(key, value) {
  emit("FOO", key + " " + value);
}
```

Wrong!

Designing MapReduce algorithms

- Key decision: What should be done by `map`, and what by `reduce`?
  - `map` can do something to each individual key-value pair, but it can't look at other key-value pairs
  - `reduce` can aggregate data; it can look at multiple values, as long as `map` has mapped them to the same (intermediate) key
    - Example: Count the number of words, add up the total cost, ...

More details on the MapReduce data flow



Some additional details

- To make this work, we need a few more parts in Hadoop HDFS system
- The file system (distributed across all nodes):
  - Stores the inputs, outputs, and temporary results
- The driver program (executes on one node):
  - Specifies where to find the inputs, the outputs
  - Specifies what mapper and reducer to use
  - Can customize behavior of the execution
- The runtime system (controls nodes):
  - Supervises the execution of tasks

Java MapReduce code on Apache Hadoop 2.7.2

Understand WordCount and extend it for Exercise 3!

MapReduce Program

- A MapReduce program consists of the following 3 parts :
- Driver → main (would trigger the map and reduce methods)
- Mapper
- Reducer

- It is better to include the map reduce and main methods in 3 different classes

Mapper

- 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);
- }
- }
- }

Mapper

Interface Mapper<K1,V1,K2,V2> , the first pair is the input key/value pair, the second is the output key/value pair

```
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);
      }
    }
  }
```

Mapper

Keys are the position in the file, and values are the line of text. Context emits the output.

```
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);
    }
  }
}
```

Reducer

```
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

Given the Mapper and Reducer code, the main() starts the MapReduce running.

- 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);
- }

Main function

Configurations are specified by resources. A resource contains a set of name/value pairs as XML data.

- 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);
- }

Main function

Normally the user creates the application, describes various facets of the job via Job and then submits the job and monitor its progress.

```
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);
}
```

Main function

CombinerClass is a mini reducer in a single node.

- 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);
- }

Combiner class

- Combiner class "mini-reduce"
- machine A emits <the, 1>, <the, 1>
- machine B emits <the, 1>.
- a Combiner on machine A emits <the, 2>. This value, along with the <the, 1> from machine B will both go to the Reducer node

- We have now saved bandwidth, but preserved the computation.

- Watch a video
- https://www.youtube.com/watch?v=fHWXRxB3UqU&t=654s

- Answer the questions in the learning-objectives form

Answers the questions and you will be good at MapReduce

More MapReduce examples

Mapreduce Example

- Generate term co-occurrence matrix for a text collection
  - M = N x N matrix (N = vocabulary size)
  - M_{ij}: number of times *i* and *j* co-occur in some context
    (for concreteness, let's say context = sentence)

e.g., Mohammad and Hirst (EMNLP, 2006)

First Try: “Pairs”

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit (a, b) → count

- Reducers sums up counts associated with these pairs

“Pairs” Analysis

- Advantages
  - Easy to implement, easy to understand
- Disadvantages
  - Lots of pairs to sort and shuffle around (upper bound?)

Another Try: “Stripes”

- Idea: group together pairs into an associative array

$(a, b) \rightarrow 1$
$(a, c) \rightarrow 2$
$(a, d) \rightarrow 5$
$(a, e) \rightarrow 3$
$(a, f) \rightarrow 2$

$a \rightarrow \{ b: 1, c: 2, d: 5, e: 3, f: 2 \}$

$a \rightarrow \{ b: 1, \quad d: 5, e: 3 \}$
$a \rightarrow \{ b: 1, c: 2, d: 2, \quad f: 2 \}$
$a \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \}$

- Each mapper takes a sentence:
    - Generate all co-occurring term pairs

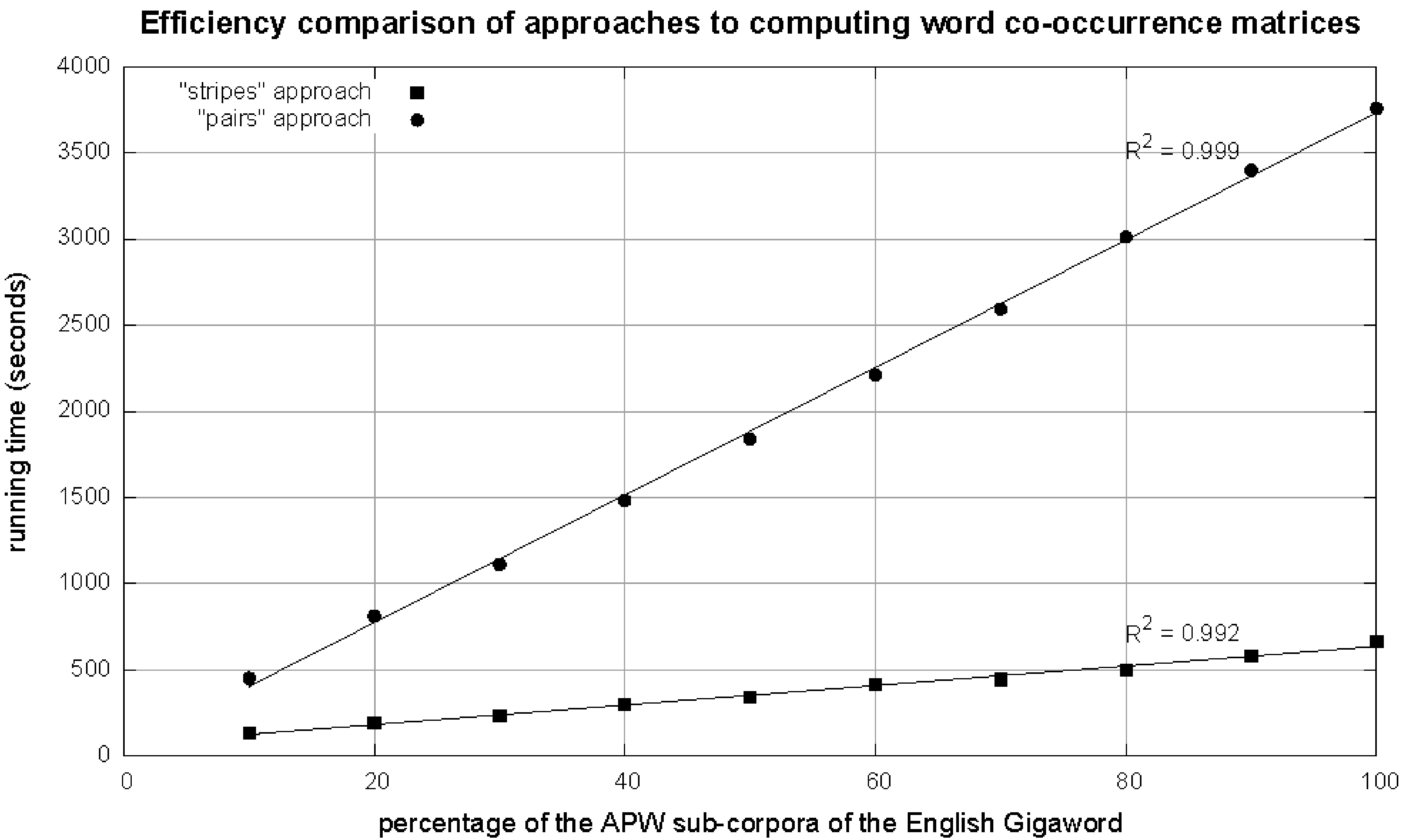
Another Try: "Stripes"

- Reducers perform element-wise sum of associative arrays

$$
\begin{array}{rl}
 & a \rightarrow \{\, b{:}\ 1, \qquad\quad d{:}\ 5,\ e{:}\ 3 \,\} \\
+ & a \rightarrow \{\, b{:}\ 1,\ c{:}\ 2,\ d{:}\ 2, \qquad\quad f{:}\ 2 \,\} \\
\hline
 & a \rightarrow \{\, b{:}\ 2,\ c{:}\ 2,\ d{:}\ 7,\ e{:}\ 3,\ f{:}\ 2 \,\}
\end{array}
$$

"Stripes" Analysis

- Advantages
  - Far less sorting and shuffling of key-value pairs
  - Can make better use of combiners
- Disadvantages
  - More difficult to implement
  - Underlying object is more heavyweight



Cluster size: 38 cores

Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

Joins in MapReduce

- (1) Reduce-side join
- (2) Broadcast join
- (3) Map-side filtering and Reduce-side join
  - a Bloom filter

Reduce-side join

- Map
  - output <key, value>
  - key>>join key, value>>tagged with data source
- Reduce
  - do a full cross-product of values
  - output the combination results

Example

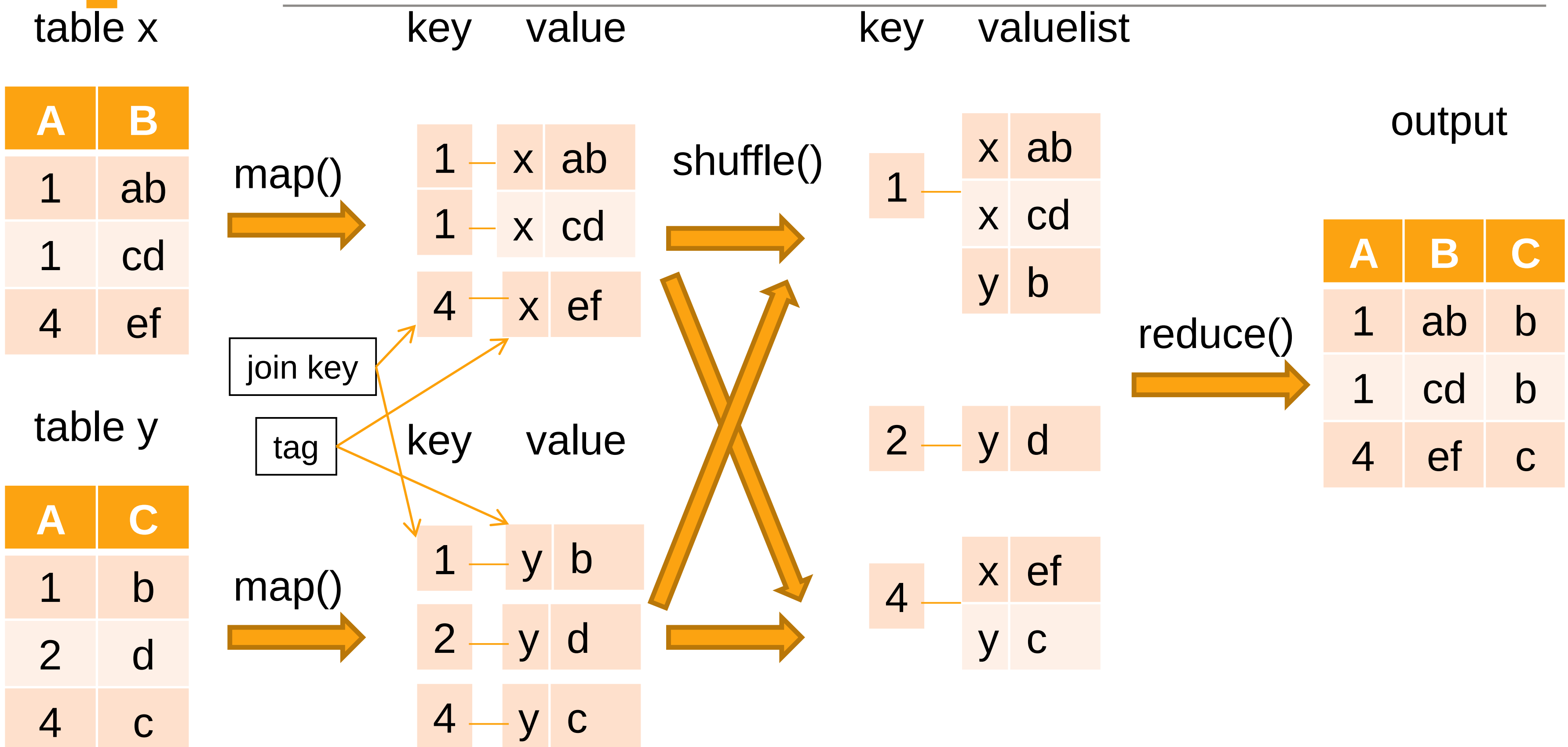


table x
A B
1 ab
1 cd
4 ef
table y
A C
1 b
2 d
4 c
map()
map()
join key
tag
key value
1 x ab
1 x cd
4 x ef
key value
1 y b
2 y d
4 y c
shuffle()
key valuelist
1 x ab
x cd
y b
2 y d
4 x ef
y c
reduce()
output
A B C
1 ab b
1 cd b
4 ef c

Broadcast join (*replicated join)*

- Using distributed cache to broadcast the smaller table

- DistributedCache can be used to distribute simple, read-only data needed by applications.

- MapReduce copies all the cache files in the local file system of all the nodes before any task for the job starts on that node.

- Do join in Map()

Map-side filtering and Reduce-side join

- If table Y is not small, copying Y to every node (in DistributedCache) will take a lot of IO and network overheads.

- Can we use Bloom filter?

Map-side filtering and Reduce-side join

- A smaller representation of joined keys of table Y in a bloom filter

- Replicate this bloom filter to each node.

- At the map side, the bloom filter can be used to filter the records in table X and reduce the size of records in the shuffle and reduce phases.

- How to handle false positives of a bloom filter?

Exercise 3

- Write an executable MapReduce program to perform the table join in Exercise 3

Summary

- MapReduce is a scalable software programming framework

- Hadoop HDFS is the platform to support MapReduce program （next lecture）