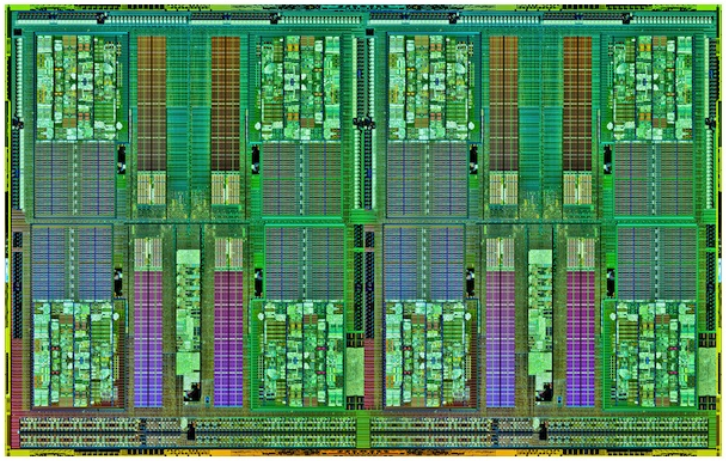
departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism
(Concorrência e Paralelismo – CP 11158)

Lecture 10

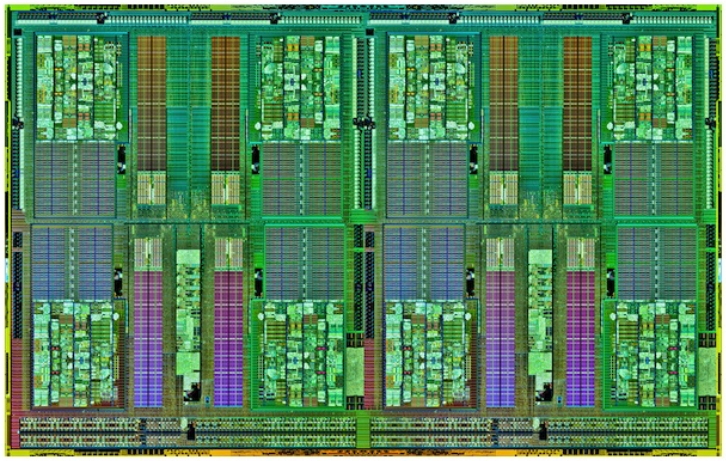
— Map-Reduce —

Map-Reduce



- AMD Opteron 6200
  - 16 cores

Map-Reduce



- AMD Opteron 6200
  - 16 cores

Google's Map-Reduce

What is MapReduce?

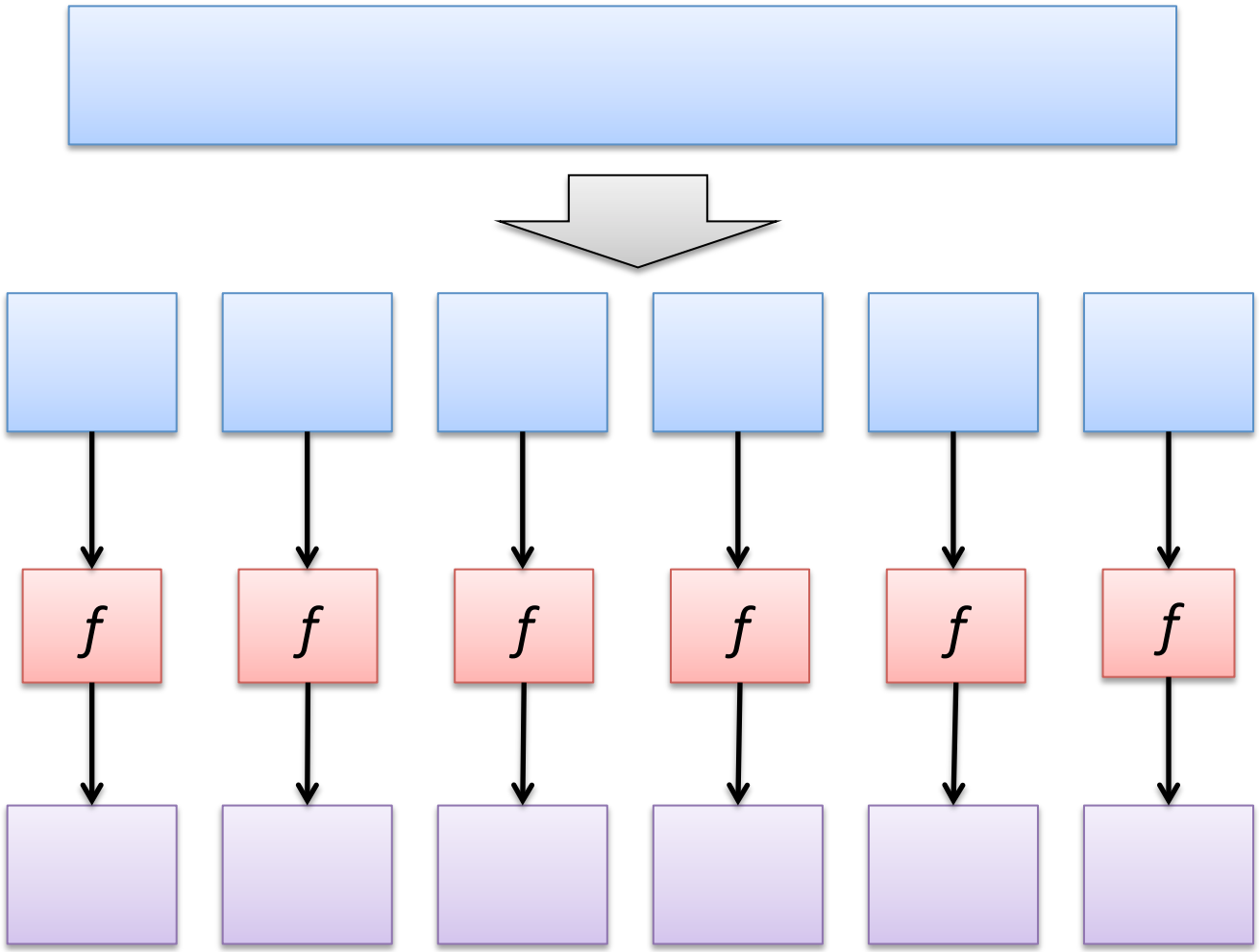
- A famous distributed programming model
- In many circles, considered *the* key building block for much of Google's data analysis
  - A programming language built on it: Sawzall, http://labs.google.com/papers/sawzall.html
  - *… Sawzall has become one of the most widely used programming languages at Google. … [O]n one dedicated Workqueue cluster with 1500 Xeon CPUs, there were 32,580 Sawzall jobs launched, using an average of 220 machines each. While running those jobs, 18,636 failures occurred (application failure, network outage, system crash, etc.) that triggered rerunning some portion of the job. The jobs read a total of 3.2×10^{15} bytes of data (2.8PB) and wrote 9.9×10^{12} bytes (9.3TB).*
  - Other similar languages: Yahoo's Pig Latin and Pig; Microsoft's Dryad
- Cloned in open source: Hadoop, http://hadoop.apache.org/

Motivation

- Large-Scale Data Processing
  - Want to use 1000s of CPUs
    - But don't want hassle of ***managing*** things
- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Some Problems are Embarrassingly Parallel (1)

Task: Convert 405K TIFF images (~4 TB) to PNG



Input: many TIFF images

Distribute images among K computers

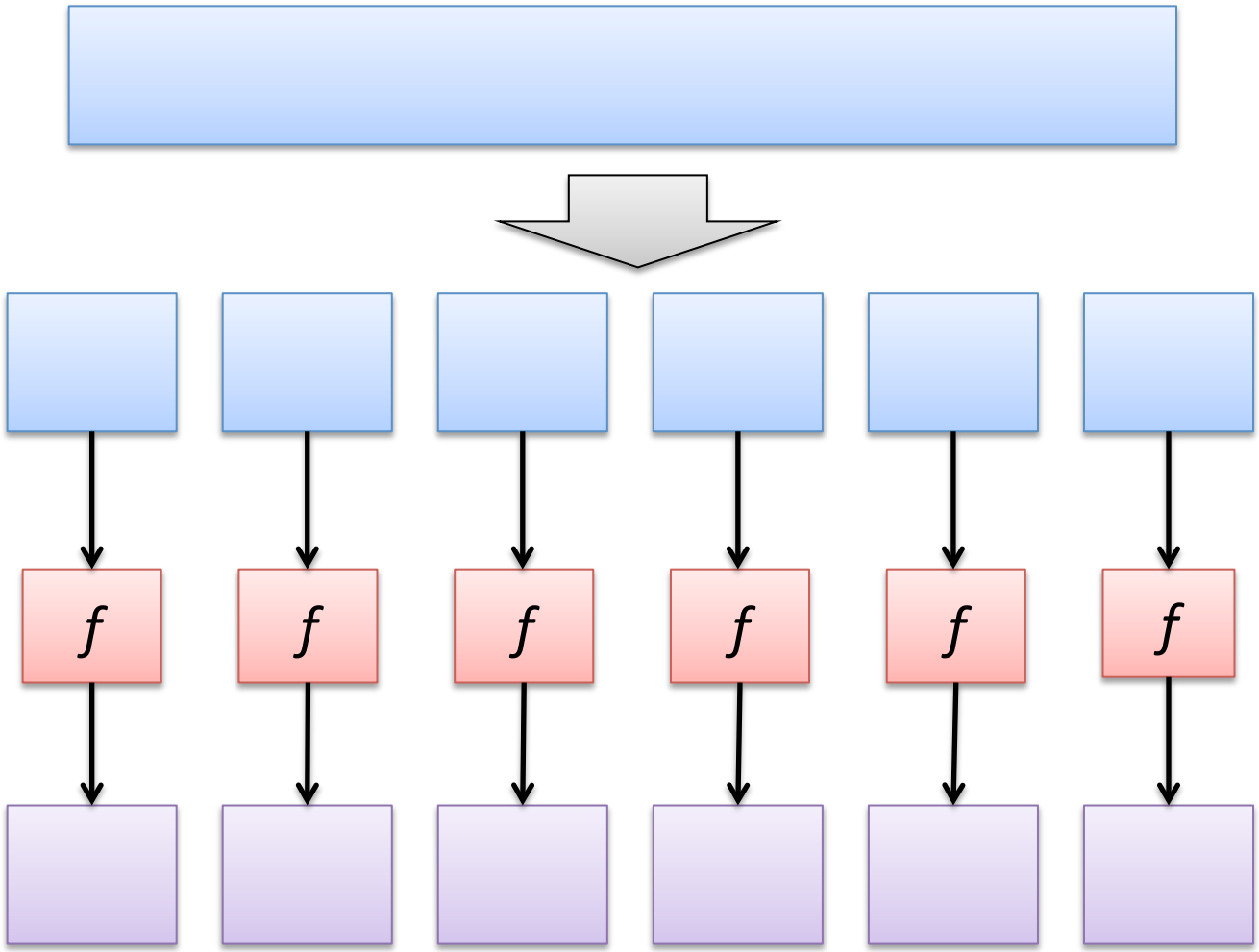
f is a function to convert TIFF to PNG; apply it to every item

Output: a big distributed set of converted images

http://open.blogs.nytimes.com/2008/05/21/the-new-york-times-archives-amazon-web-services-timesmachine/

Some Problems are Embarrassingly Parallel (2)

Task: Compute the word frequency of 5M documents



Input: millions of documents

Distribute documents among K computers

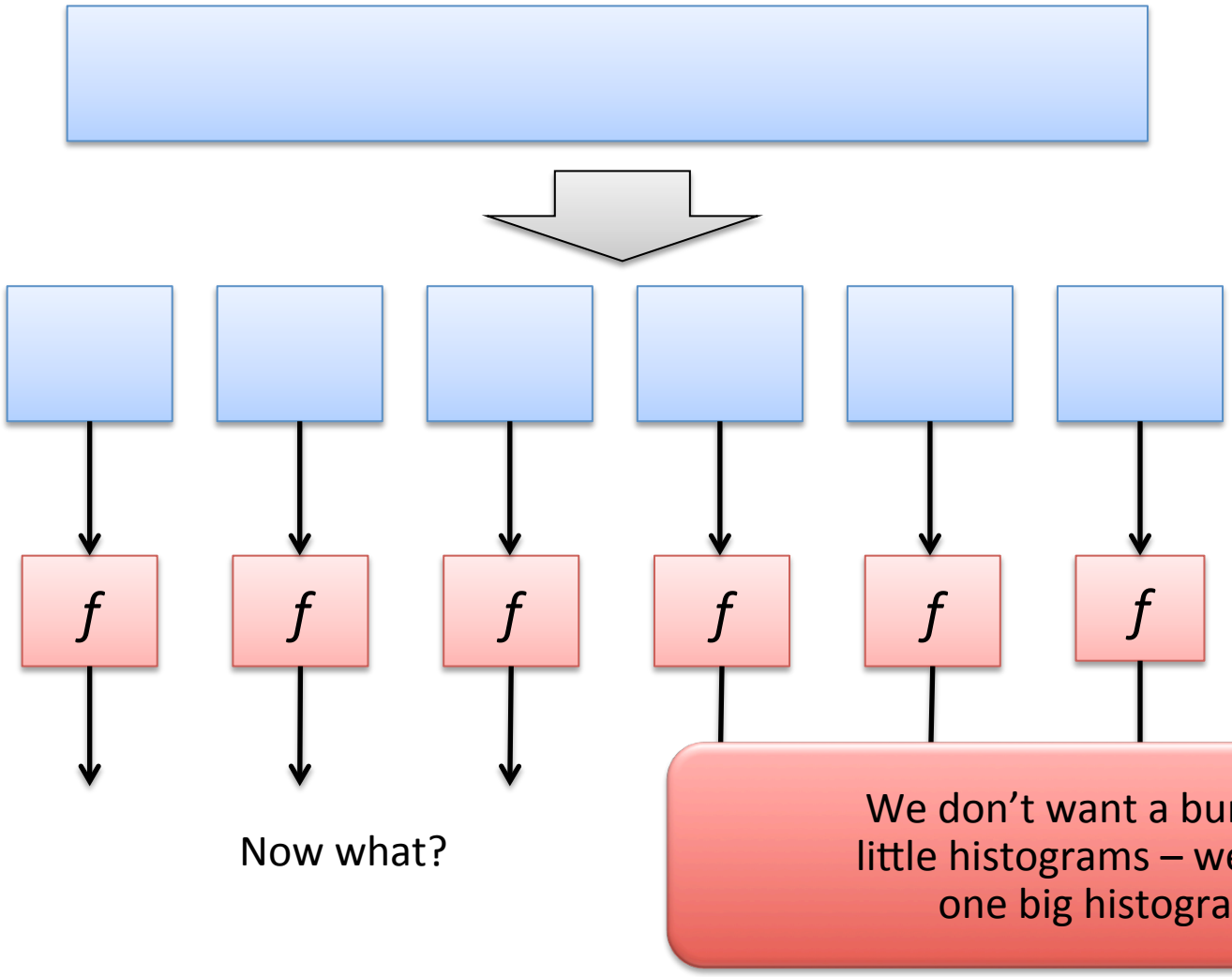
For each document *f* returns a set of <word, freq> pairs

Output: a big a big distributed list of *sets* of word freqs.

Adapted from slides by Bill Howe

Some Problems are Embarrassingly Parallel (3)

Task: Compute the word frequency across *all* documents



Input: millions of documents

Distribute documents among K computers

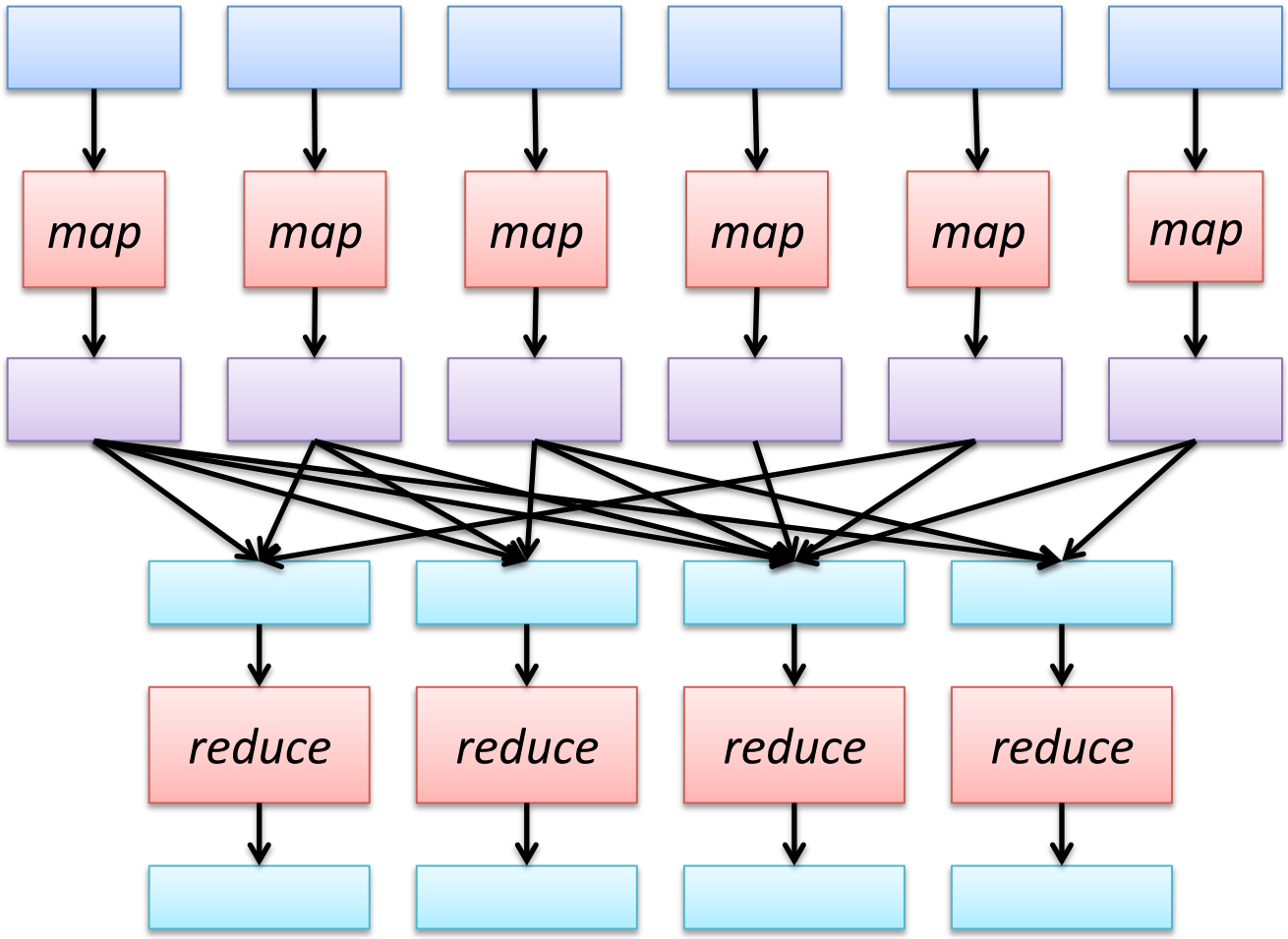
For each document *f* returns a set of <word, freq> pairs

Now what?

We don't want a bunch of little histograms – we want one big histogram

MapReduce

Task: Compute the word frequency across all documents



Distribute documents among K computers

For each document *f* returns a set of <word, freq> pairs

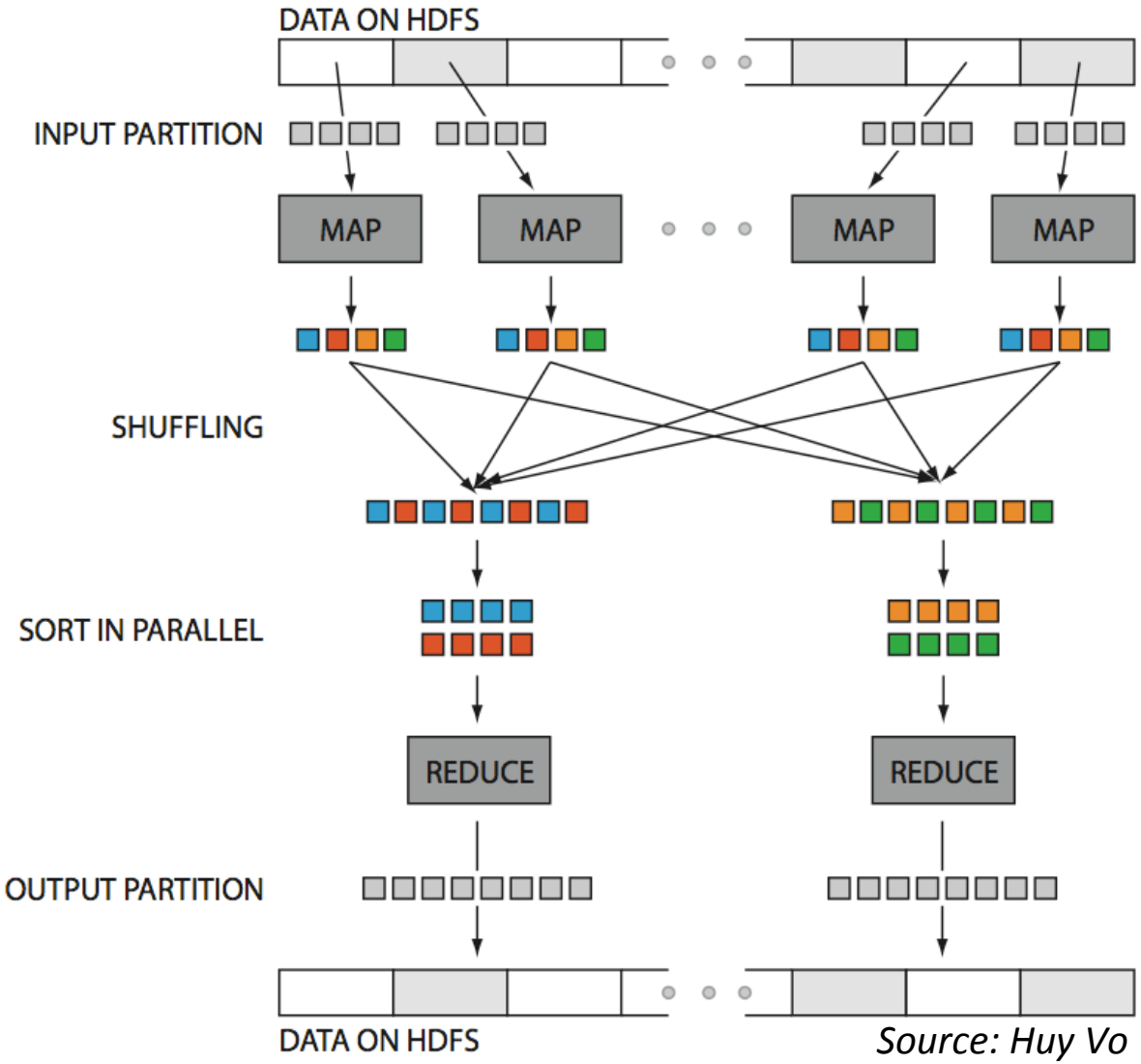
A big distributed list of sets of word freqs

Shuffle <word, freq> pairs so that all the counts for a word are sent to the same host

Add the counts of each word

Output: the distributed histogram

Hadoop on One Slide

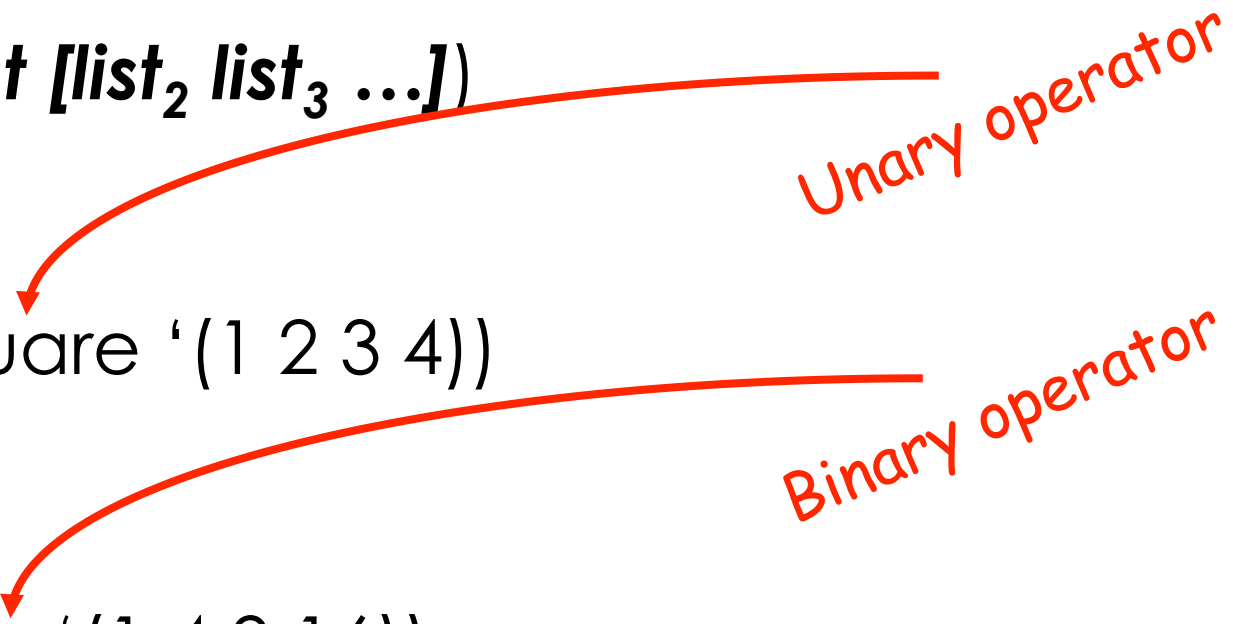


Source: Huy Vo

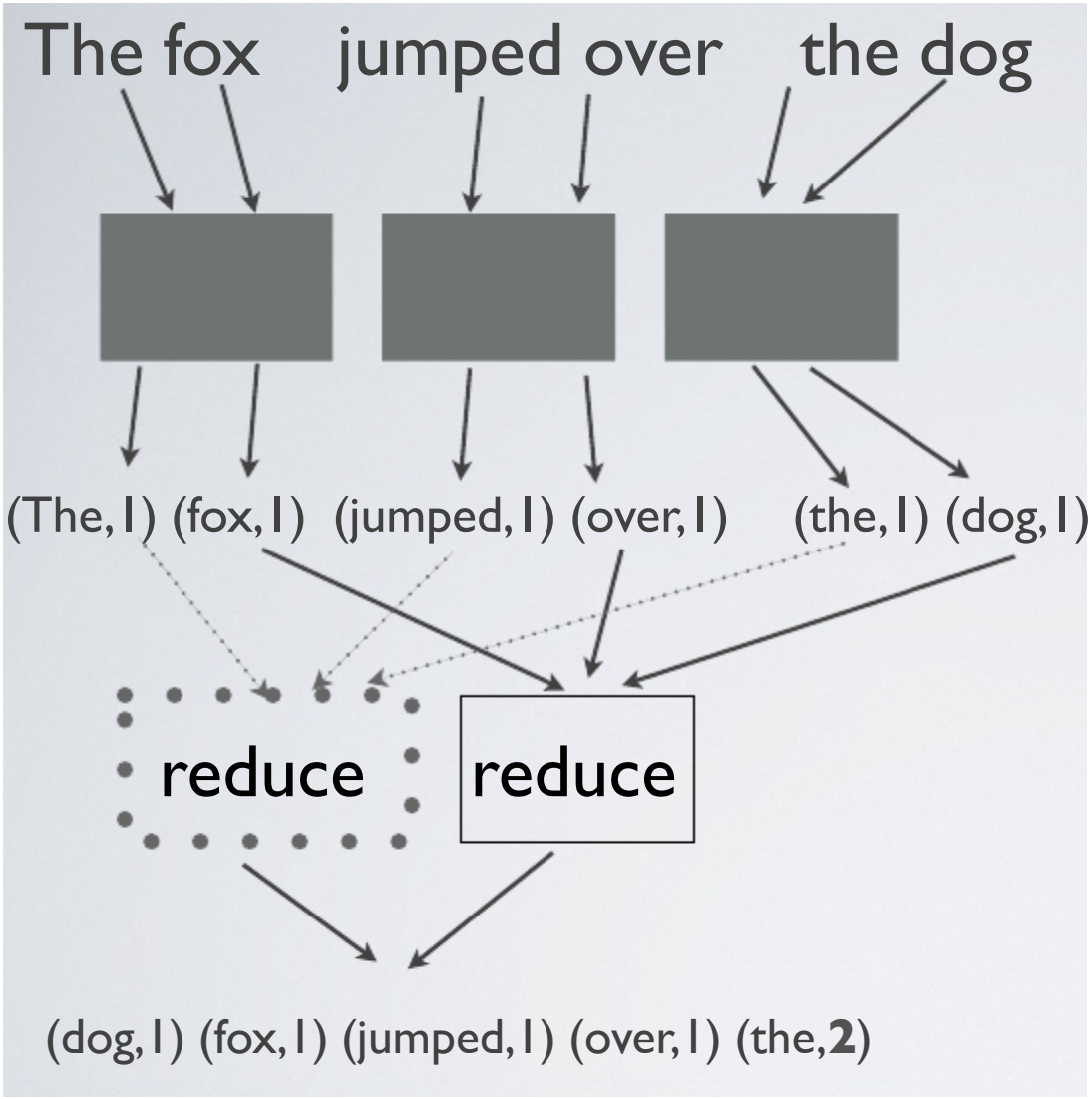
- MapReduce was invented at Google
  [Dean & Ghemawat, OSDI'04]
- Hadoop = open source implementation
- Data stored on HDFS distributed file system
  - Direct-attached storage
- Programmers write Map and Reduce functions
- Framework provides automated parallelization and fault tolerance
  - Data replication, restarting failed tasks
  - Scheduling Map and Reduce tasks on hosts with local copies of input data

Map in Lisp (Scheme)

- (map ***f list [list$_2$ list$_3$ …]***)
- (map square '(1 2 3 4))
  - (1 4 9 16)
- (reduce + '(1 4 9 16))
  - 30
- (reduce + (map square (map – l_1 l_2))))

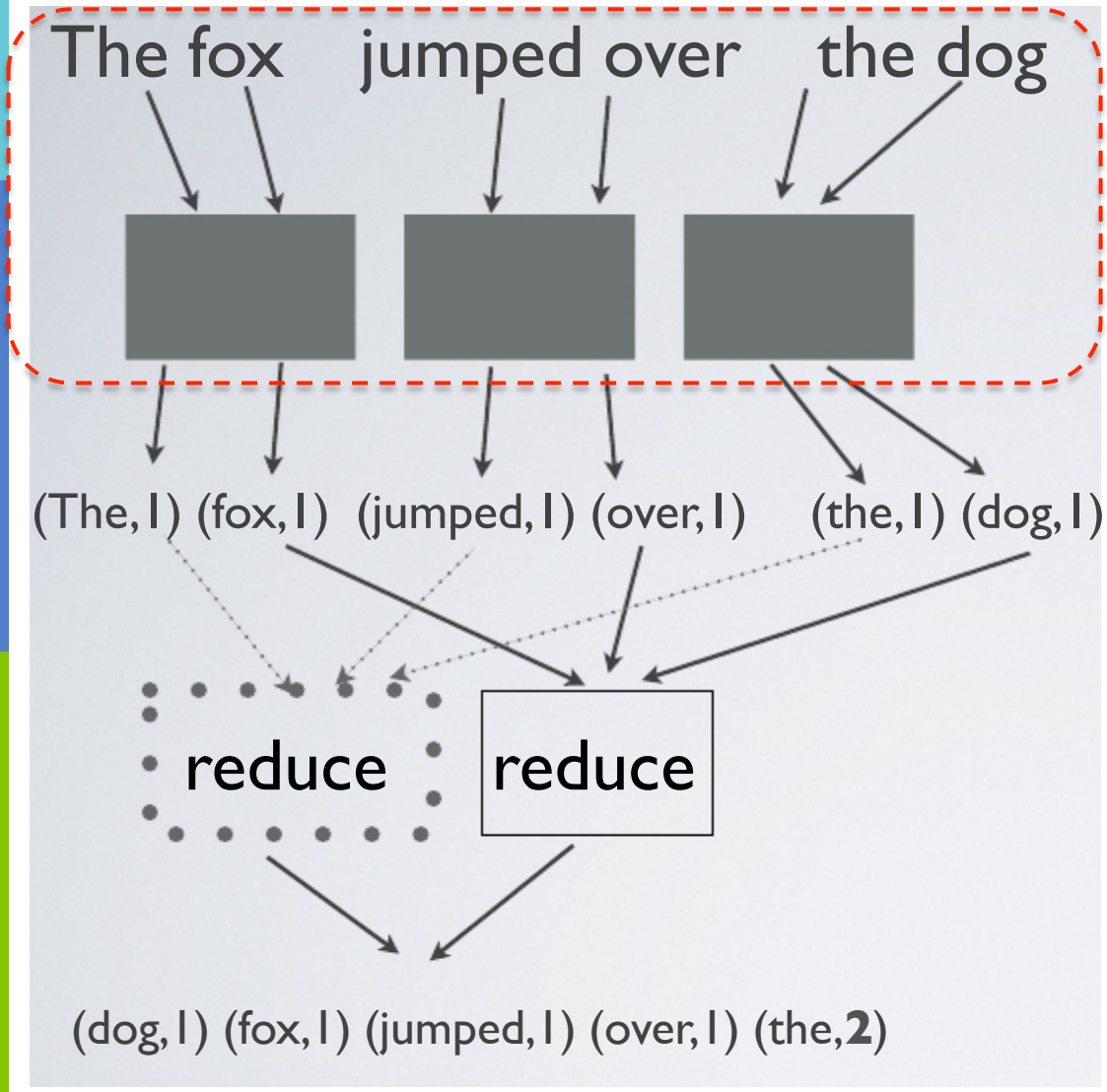


Google's Map-Reduce Model



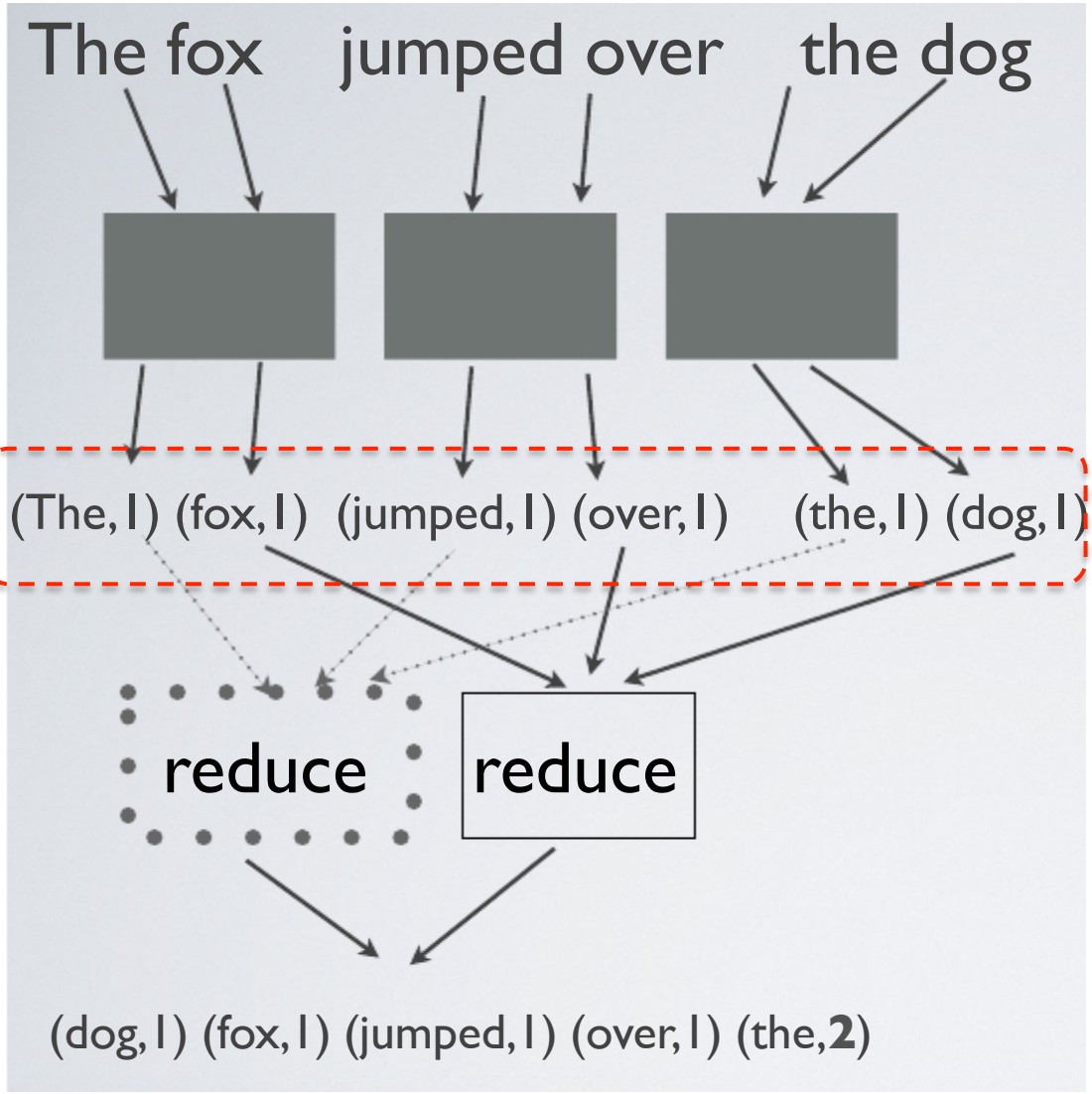
- MAP
  - Applied to each portion of input data
  - Emit intermediate pairs <key, value>
- REDUCE
  - Applied to all pairs with the same key
  - Sorted final output

Google's Map-Reduce Model



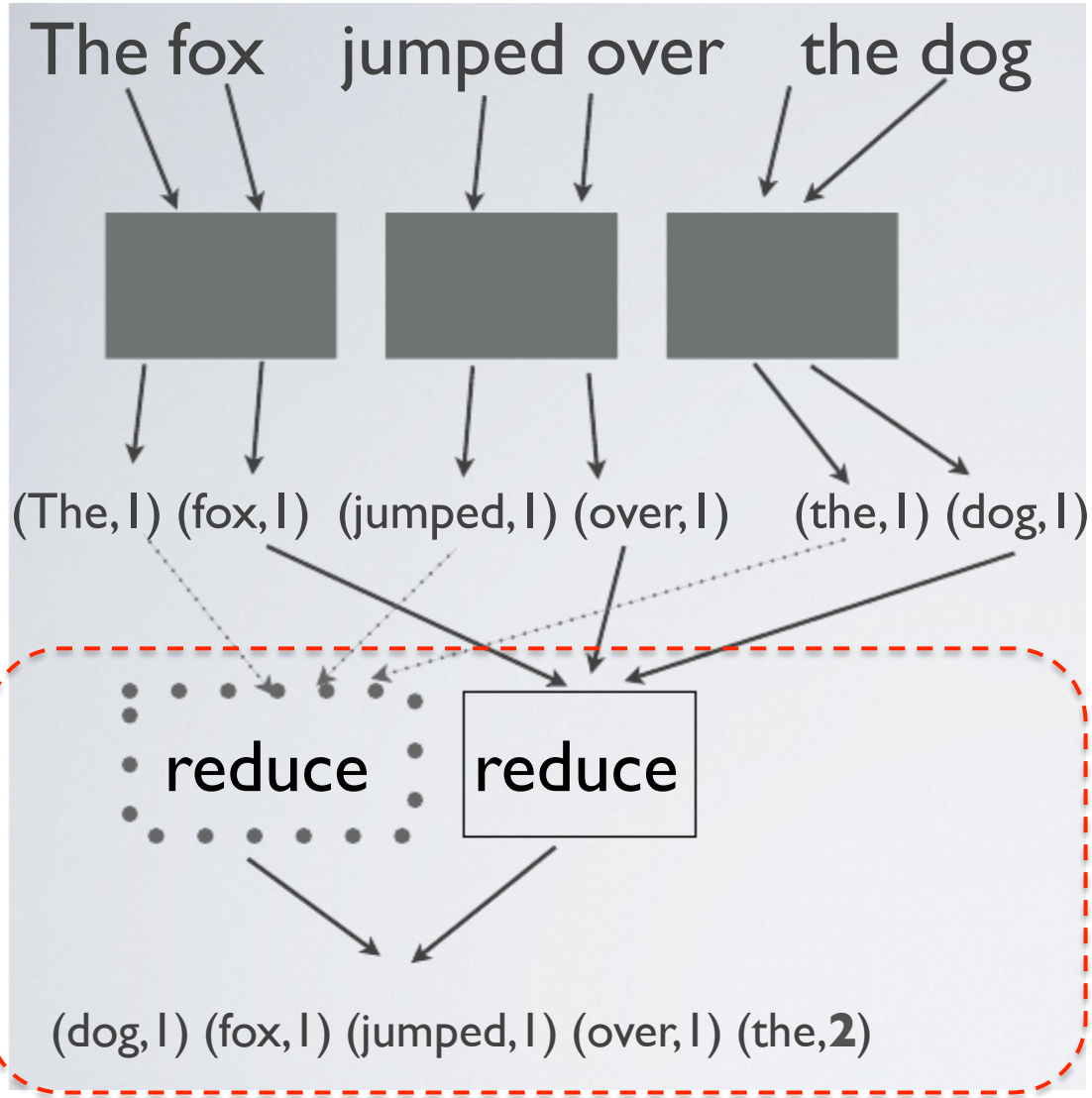
- MAP
  - Applied to each portion of input data
  - Emit intermediate pairs <key, value>
- REDUCE
  - Applied to all pairs with the same key
  - Sorted final output

Google's Map-Reduce Model



- MAP
  - Applied to each portion of input data
  - Emit intermediate pairs <key, value>
- REDUCE
  - Applied to all pairs with the same key
  - Sorted final output

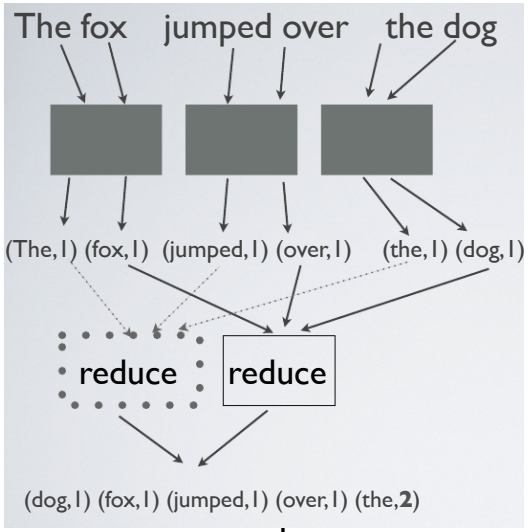
Google's Map-Reduce Model



- MAP
  - Applied to each portion of input data
  - Emit intermediate pairs <key, value>
- REDUCE
  - Applied to all pairs with the same key
  - Sorted final output

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Google's Map-Reduce Model



- Simplicity!!!
  - User only provides the MAP and REDUCE functions!!
- Map/Reduce model offers
  - Parallelization abstraction
    - Hiding parallelization communication issues among workers
    - Hiding scheduling
  - Resource management
  - Fault tolerance
    - Task monitoring
    - Task replication
    - Task re-execution

MapReduce Programming Model

- Iput & Output: each a set of key/value pairs
- Programmer specifies two functions:

```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

  - Processes input key/value pair
  - Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

- *Inspired by primitives from functional programming languages such as Lisp, Scheme, and Haskell*

Slide source: Google

Example: What Does This Do?

```
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: word
    // output_values: ????

    int result = 0;

    for each v in intermediate_values:

        result += v;

    EmitFinal(output_key, result);
```

Map/Reduce ala Google

- map(key, val) is run on each item in set
  - emits new-key / new-val pairs
- reduce(key, vals) is run for each unique key emitted by map()
  - emits final output
- Often, one application will need to run map/reduce many times in succession

Count words in docs

- Input consists of (url, contents) pairs
- map(key=url, val=contents):
  - For each word *w* in contents, emit (w, "1")
- reduce(key=word, values=uniq_counts):
  - Sum all "1"s in values list
  - Emit result "(word, sum)"

Reverse Web-Link Graph

- Map
  - For each URL linking to target, …
  - Output <target, source> pairs

- Reduce
  - Concatenate list of all source URLs
  - Outputs: <target, ***list*** (source)> pairs

Index maps words to files
Compute an *Inverted* Index

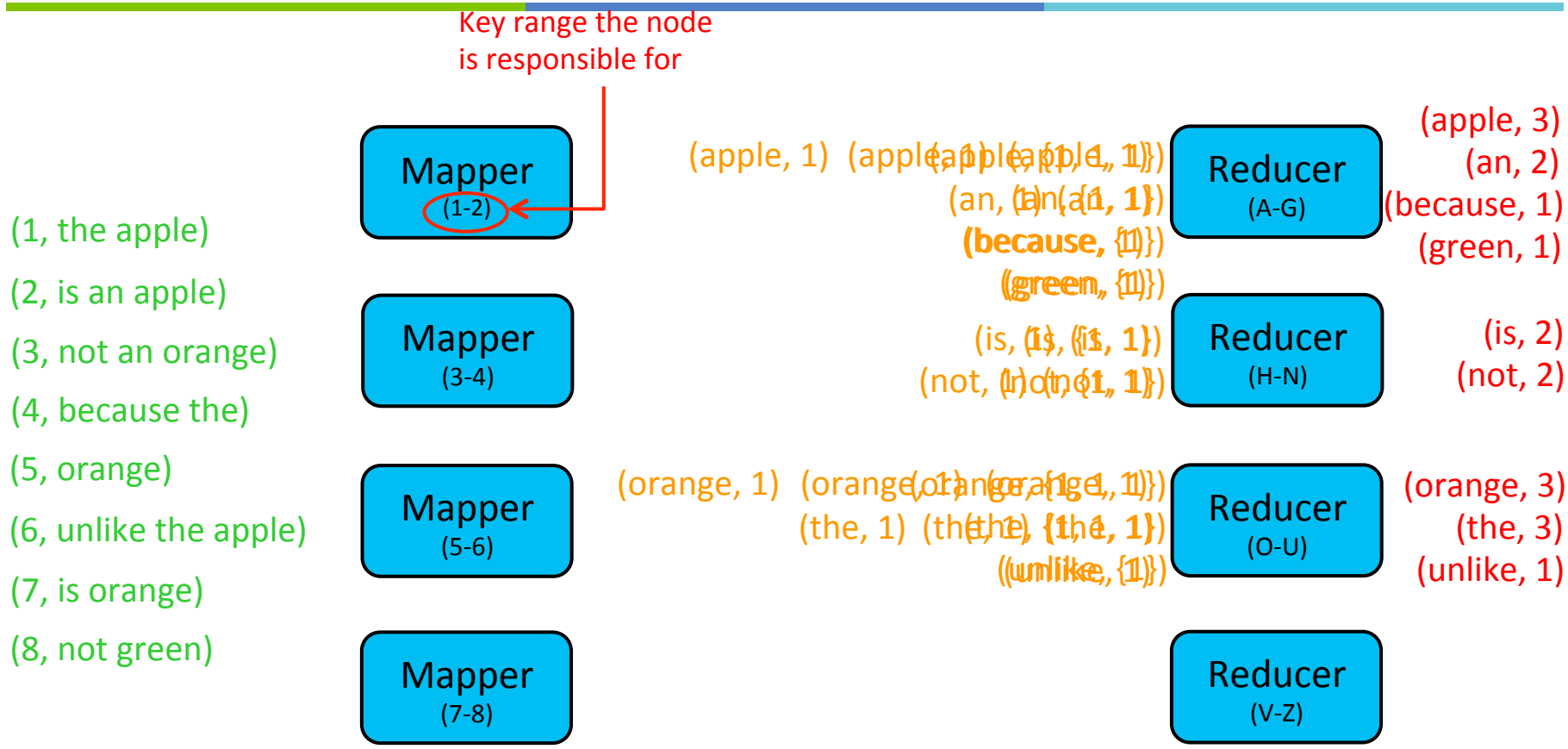
- Map

  For each file f and each word in the file w

    Output(w,f) pairs

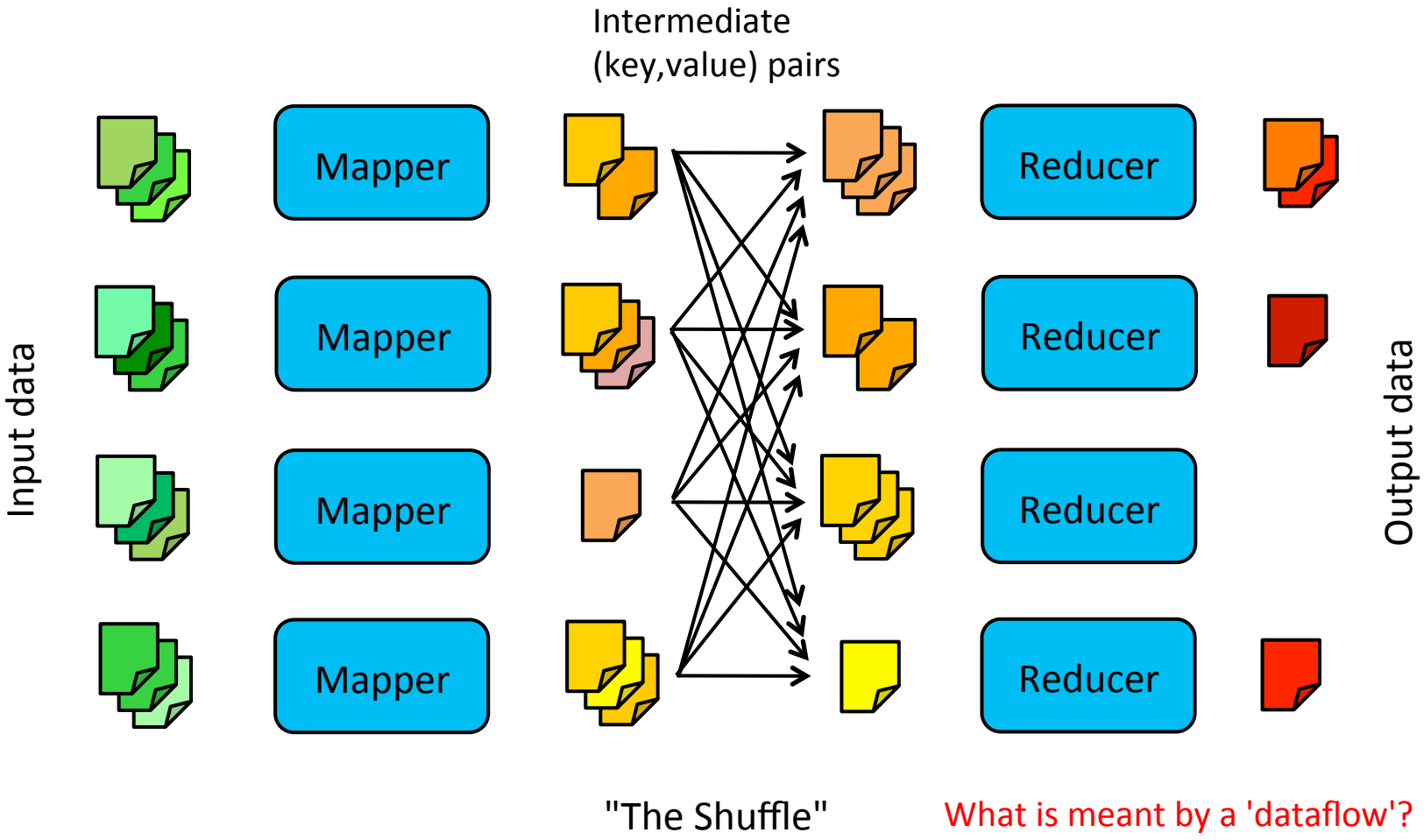
- Reduce
  - Merge, eliminating duplicates

Simple example: Word count



1. Each mapper receives some of the KV-pairs as input
2. The mappers process the KV-pairs one by one
3. Each KV-pair output by the mapper is sent to the reducer that is responsible for it
4. The reducers sort their input by key and group it
5. The reducers process their input one group at a time

MapReduce dataflow



What is meant by a 'dataflow'?
What makes this so scalable?

Common mistakes to avoid

- Mapper and reducer should be stateless
  - Don't use static variables - after `map` + `reduce` return, they should remember nothing about the processed data!
  - Reason: No guarantees about which key-value pairs will be processed by which workers!

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

Wrong!

- 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!

More common mistakes to avoid

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

Wrong!

```
reduce(key, value[]) {
  /* do some computation on
  all the values */
}
```

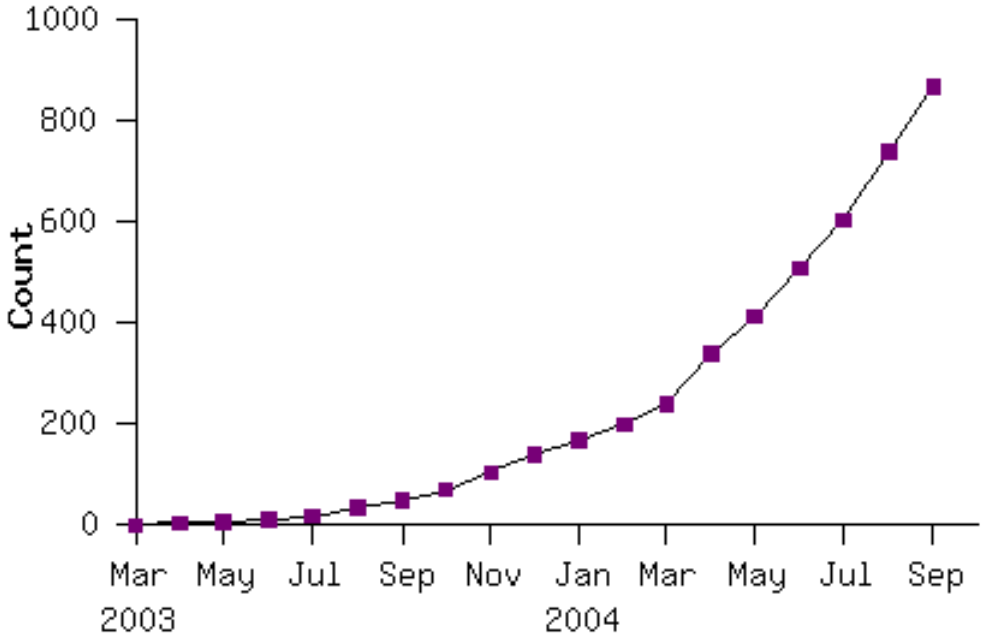
- Mapper must not map too much data to the same key
  - In particular, don't map *everything* to the same key!!
  - Otherwise the reduce worker will be overwhelmed!
  - It's okay if some reduce workers have more work than others
    - Example: In WordCount, the reduce worker that works on the key 'and'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

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
    - Example: Filtering out key-value pairs we don't need
  - `map` can emit more than one intermediate key-value pair for each incoming key-value pair
    - Example: Incoming data is text, `map` produces (word,1) for each word
  - `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, ...
- Need to get the intermediate format right!
  - If `reduce` needs to look at several values together, `map` must emit them using the same key!

Model is Widely Applicable

MapReduce Programs In Google Source Tree



Example uses:

distributed grep	distributed sort	web link-graph reversal
term-vector / host	web access log stats	inverted index construction
document clustering	machine learning	statistical machine translation
...	...	...

Implementation Overview

- Typical cluster:
  - 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
  - Limited bandwidth
  - Storage is on local disks
  - GFS: distributed file system manages data (SOSP'03)
  - Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines
- Implementation is a Java or C++ library linked into user programs

More info

- Good tutorial presentation & examples at:
  - http://research.google.com/pubs/pub36249.html
- The definitive paper:
  - http://labs.google.com/papers/mapreduce.html

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

The End