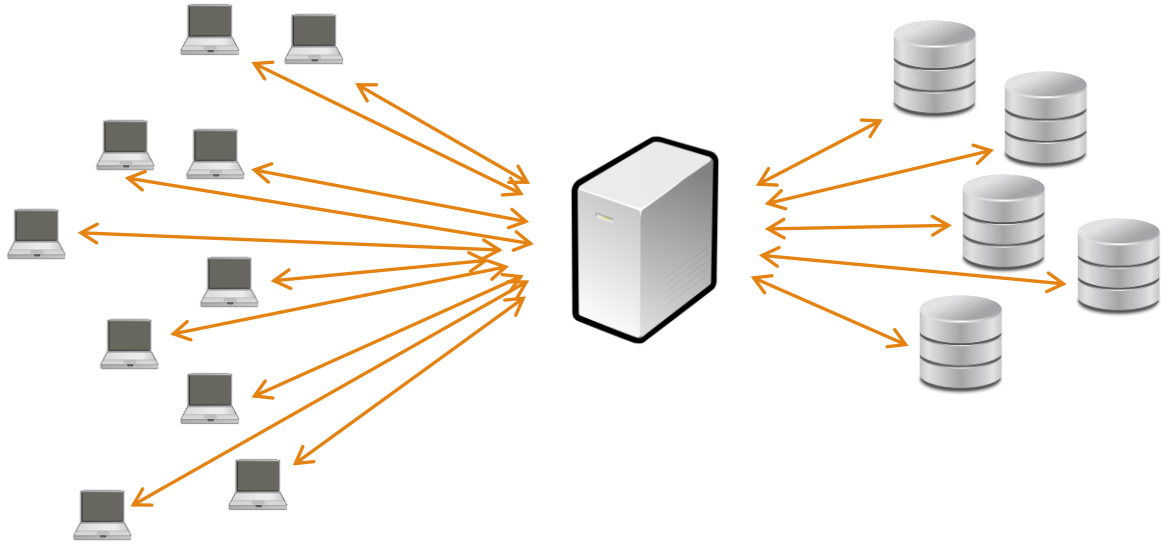
Basics of MapReduce

NOT REQUIRED FOR THE EXAM

Overview over this video

This video discusses the basics of the MapReduce framework

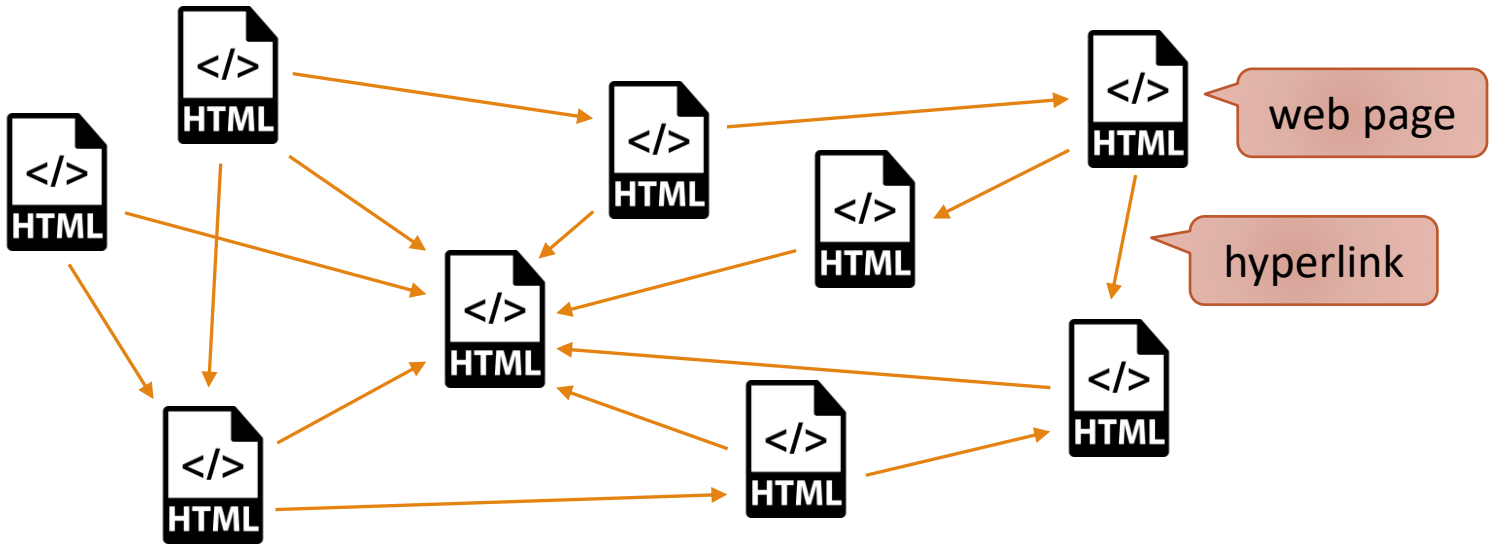
Relational Databases are Powerful



Allow us to store, manage, and analyse large datasets

Some data analysis applications go beyond what such systems can handle…

Importance of Web Pages



Graph with all relevant HTML pages on the web

Task: rank pages by “importance”

- Basis of Google Search
- ≈ an iterated matrix product (number of rows = number of pages)

Problem: millions to billions of web pages

"Big Data"

Finding Patterns in Large Datasets

Predictive maintenance, e.g.

- Which parts of a system are likely to fail or cause certain effects?
- Learn from sensor data, infrastructure data, etc.

Detect interesting patterns, e.g.

- Interesting phenomena in space, on earth, under water, in stock markets, etc.
- Recognise plants with certain diseases
- Estimate the likelihood of archeological artifacts at a certain site based on archeological data, geological parameters, etc.

Understand the structure/dynamics of large networks, e.g.

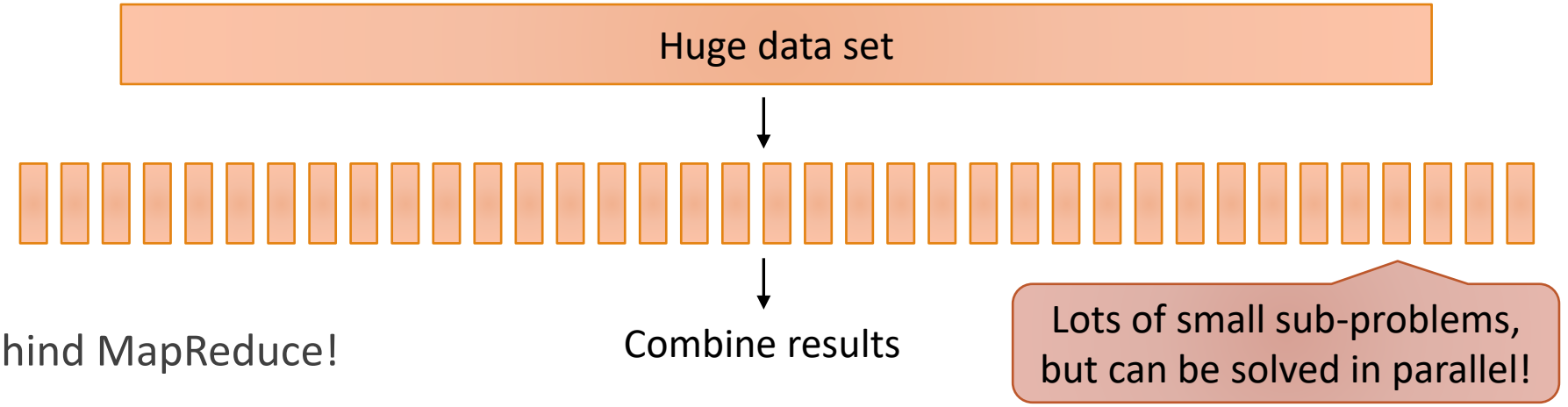
- Train networks
- Social networks

In these applications, the data is potentially huge.

Divide and Conquer

Often such problems can be solved as follows:

- Divide data into smaller chunks
- Perform computations on smaller chunks
- Combine results



Idea behind MapReduce!

MapReduce

Programming framework

- Provides simple means to create programs that can scale to terabytes of data

Users only implement two methods

- **Map**: the computation on the smaller chunks of data
- **Reduce**: how the results are combined to the final result
- MapReduces manages the rest!

Implementations:

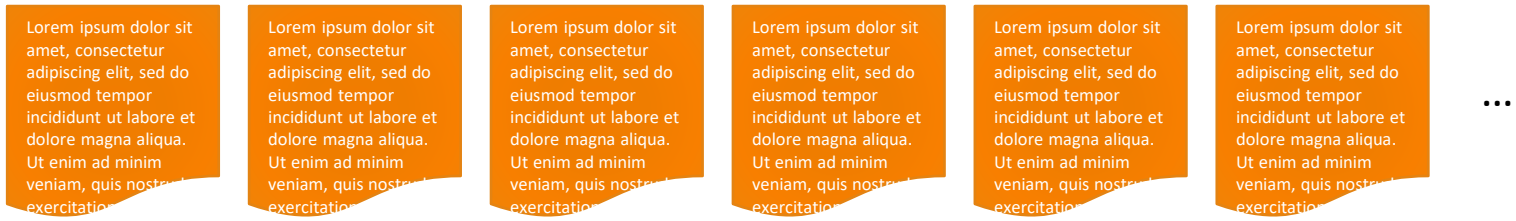
- MapReduce: Google's original implementation
- Apache Hadoop: open-source

Counting Words in Documents

or: Hello World!

The Task

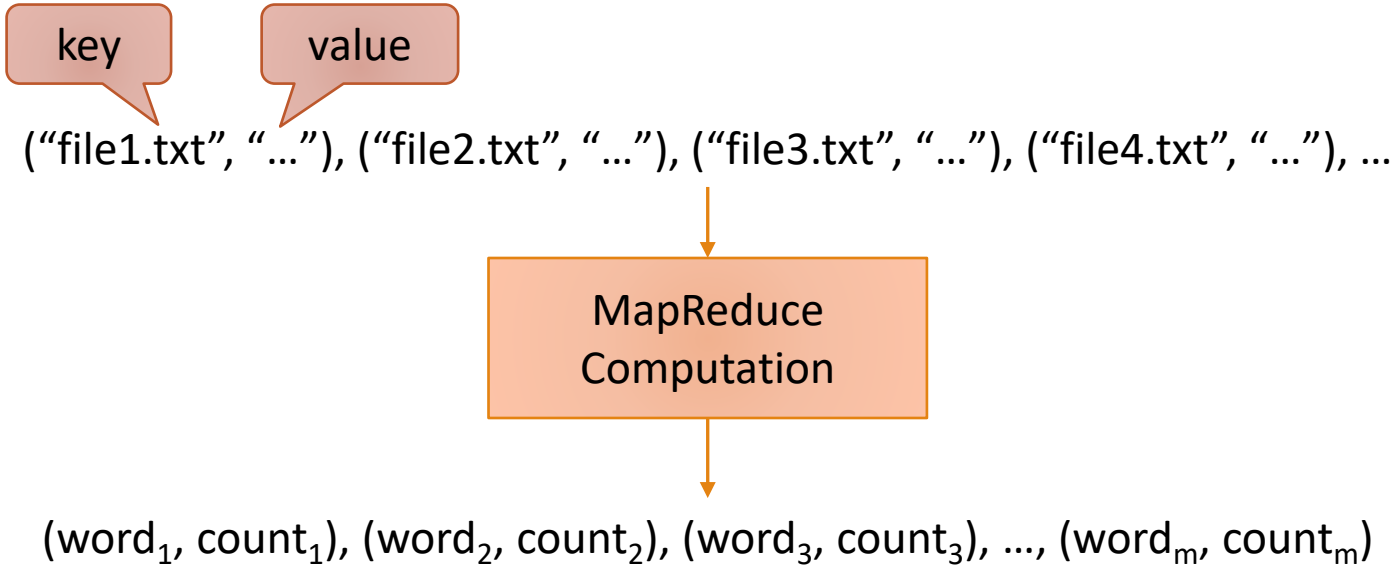
Given: a collection of “not too large” text files



Task: for each word **w**, count how often **w** occurs in these files

How To Do This In MapReduce?

The MapReduce computation:



To implement this computation, we have to write two functions: **Map** & **Reduce**

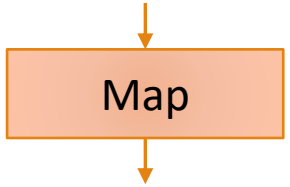
The Map Function

The Map function is applied to a single key/value pair
and produces a list of zero or more key/value pairs

```
Map(String filepath, String contents):
    for each word w in contents:
        output pair (w, "1")
```

Example:

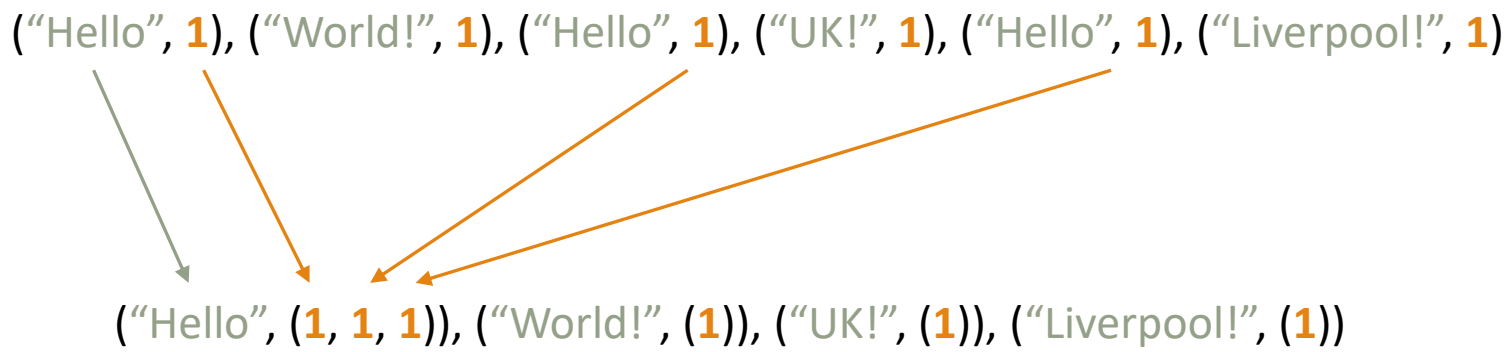
("file1.txt", "Hello World! Hello UK! Hello Liverpool!")



("Hello", "1"), ("World!", "1"), ("Hello", "1"), ("UK!", "1"), ("Hello", "1"), ("Liverpool!", "1")

Grouping

After the Map functions have been applied, we group all values by key



The list of values for a key is the input to Reduce

The Reduce Function

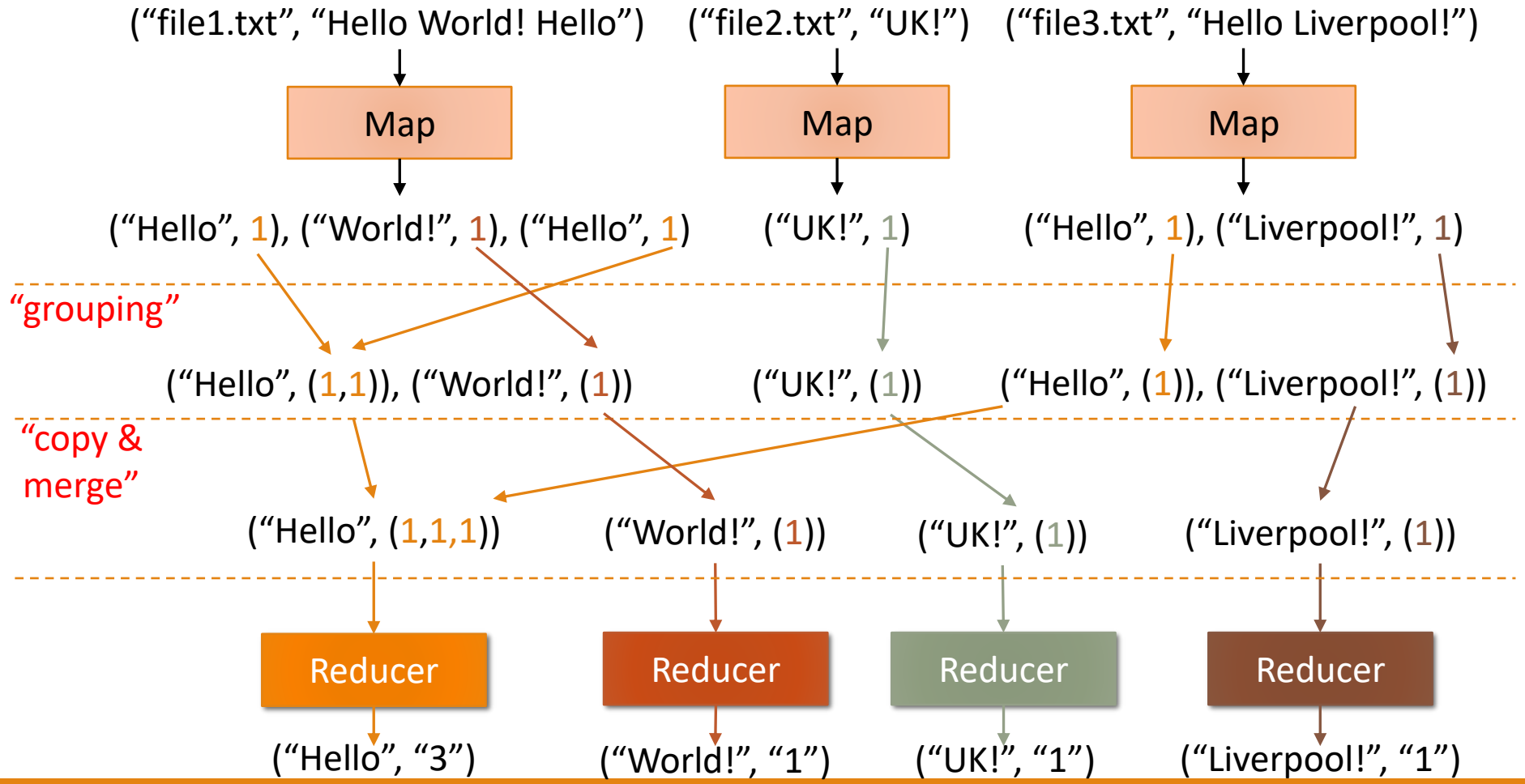
The Reduce function takes a key and a list of values as input and outputs a list of key/value pairs

```
Reduce(String word, Iterator<String> values):
    int count = 0
    for each v in values:
        count = count + parseInt(v)
    output pair (word, toString(count))
```

Example:

- Reduce(“Hello”, (“1”,”1”,”1”)) outputs (“Hello”, “3”)

Putting Things Together



The Beauty of It…

We only had to implement these two functions…

```
Map(String filepath, String contents):
    for each word w in contents:
        output pair (w, "1")
```

```
Reduce(String word, Iterator<String> values):
    int count = 0
    for each v in values:
        count = count + parseInt(v)
    output pair (word, toString(count))
```

MapReduce takes care of everything else:

- Parallel execution, including allocation of Map/Reduce to machines
- Failure

Summary

MapReduce (or Hadoop) is a powerful tool to deal with divide and conquer style problems