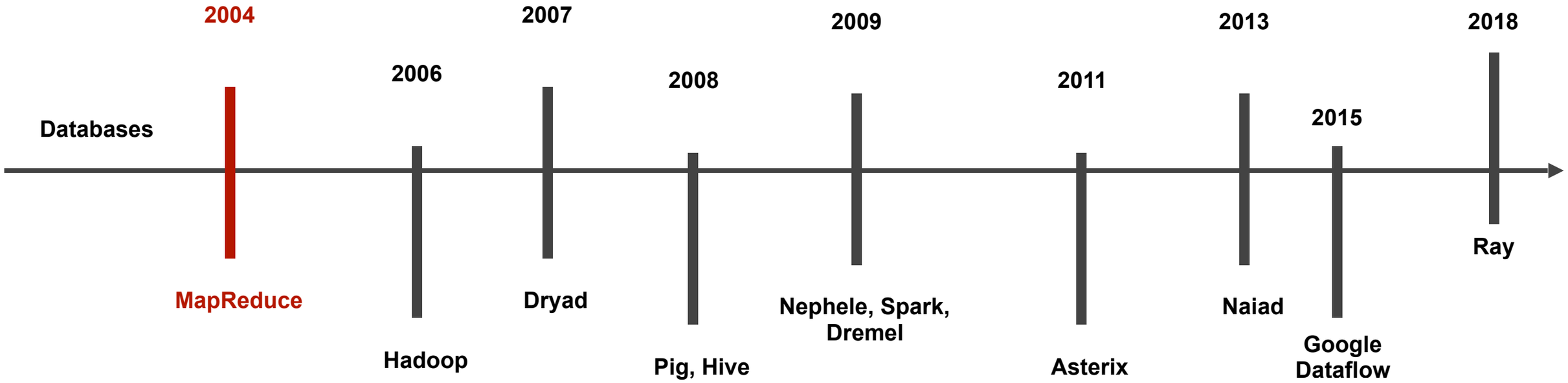
Distributed Systems

Spring Semester 2020

Lecture 2: MapReduce

John Liagouris
liagos@bu.edu

MapReduce


2004
2007
2009
2013
2018
2006
2008
2011
2015
Databases
MapReduce
Hadoop
Dryad
Pig, Hive
Nephele, Spark,
Dremel
Asterix
Naiad
Google
Dataflow
Ray

Simple Map Reduce: Case Study and Lab 1

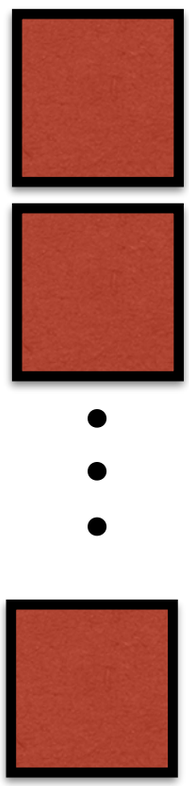
- Google
- Reusable infrastructure for doing big distributed computations that alleviates the burden of distributions from the app programer.
- Provides an abstraction
- Programmer focuses on the core of the app infrastructure does the rest

Computational Model

First Stage

— INPUT —

Massive amount of data in files



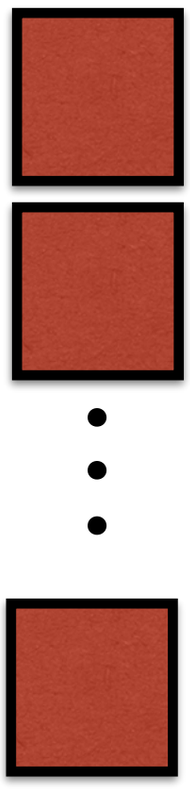
eg. 1000's of files containing text

Computational Model

First Stage

— INPUT —

Massive amount of data split into separate files



eg. 1000's of files containing text

Infrastructure invokes app specified Map functions on all files

MAP

MAP

MAP

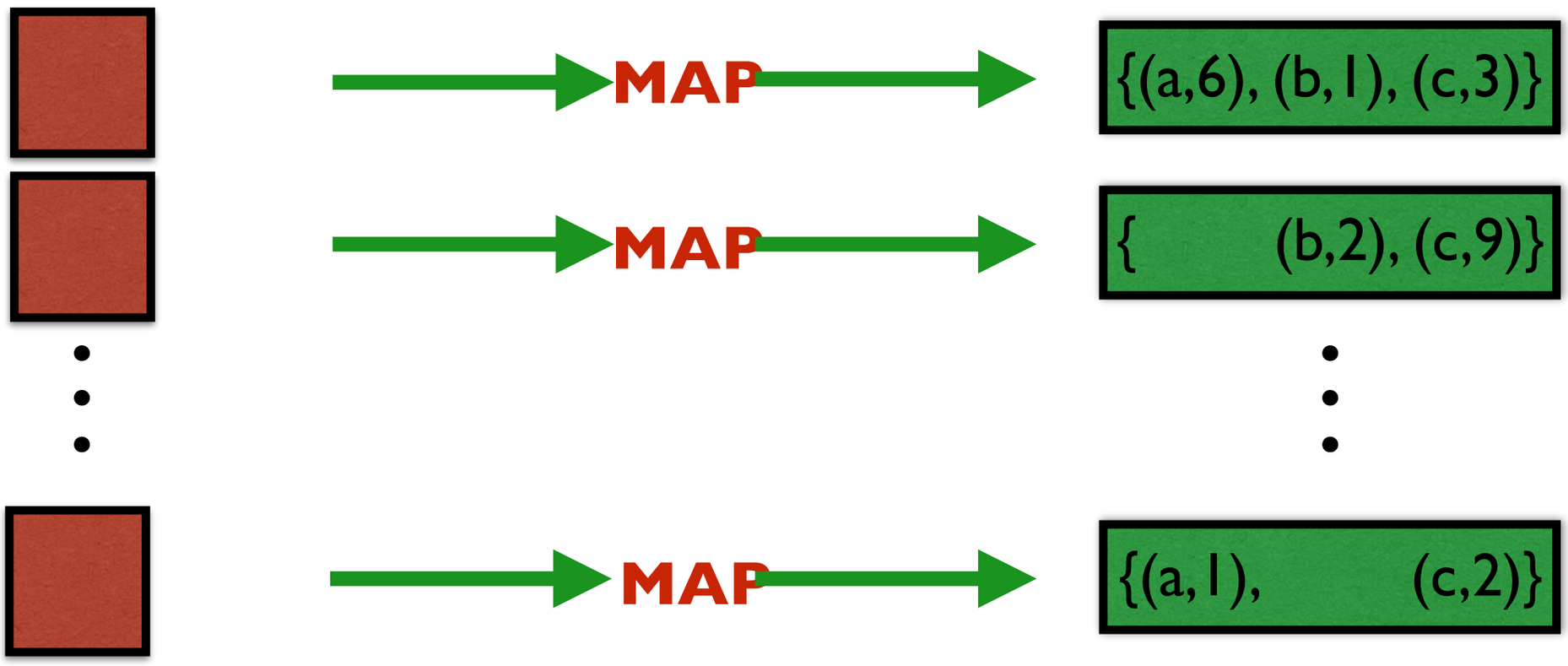
Computational Model

First Stage

— INPUT —
Massive amount
of data in files

Infrastructure
invokes app specified
Map functions on all files

Map function maps data
to key value pairs

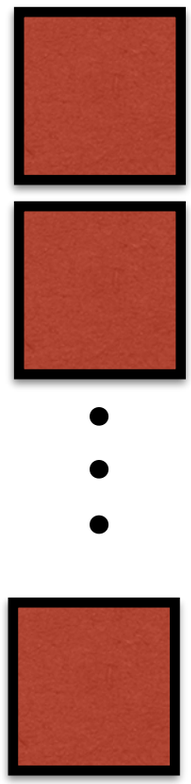


eg. 1000's of files
containing text

Computational Model

Second Stage

— INPUT —
Massive amount of data in files



eg. 1000's of files containing text

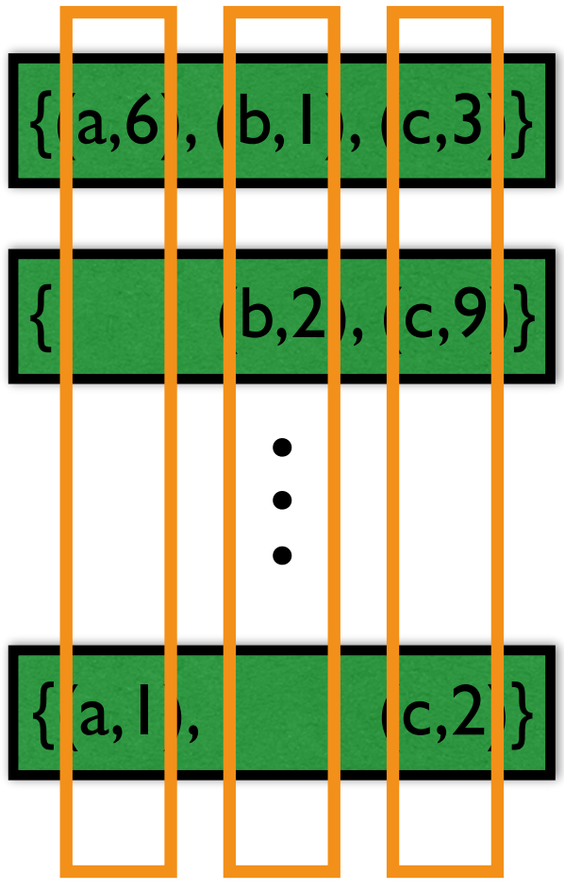
Infrastructure invokes app specified Map functions on all files

MAP

MAP

MAP

Map function maps data to key value pairs

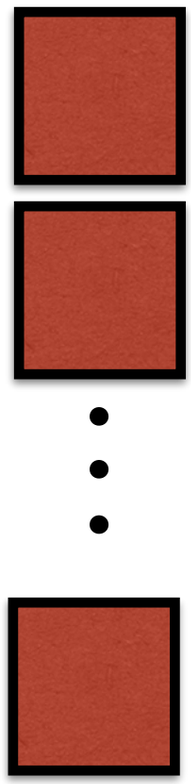


Organizes map output creating a file per key

Computational Model

Second Stage

— INPUT —
Massive amount of data in files



eg. 1000's of files containing text

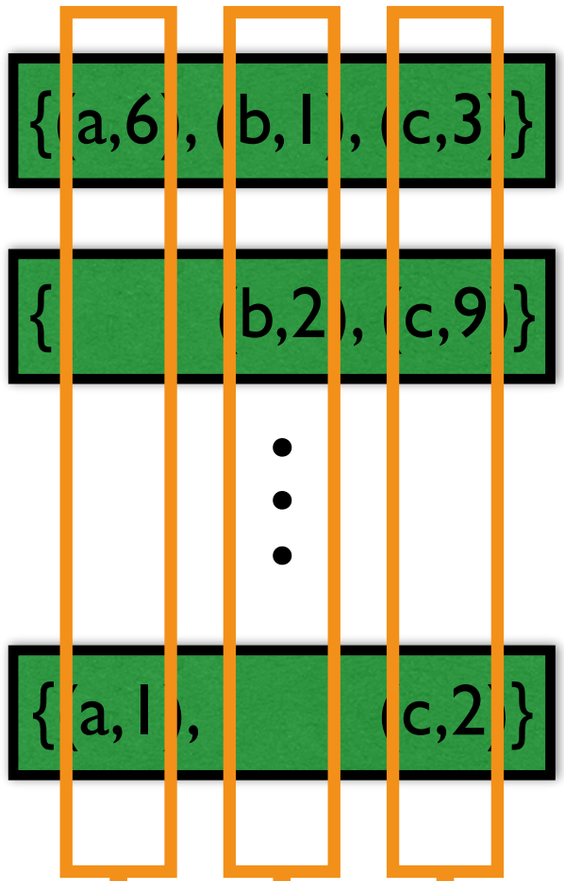
Infrastructure invokes app specified Map functions on all files

MAP

MAP

MAP

Map function maps data to key value pairs



Organizes map output creating a file per key

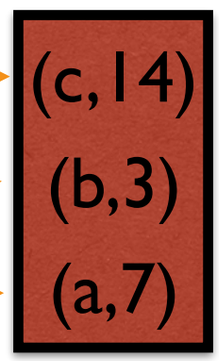
Infrastructure invokes app specified Reduce function on all files

REDUCE

REDUCE

REDUCE

— OUTPUT —
Merged



MapReduce

- Paper version is much more sophisticated
- But this is good place to start for the first Lab

MapReduce : Programming

- Programmer provides MAP and REDUCE function
- Infrastructure provides everything else!

Programmer visible model

Lab 1 Part B: Write map and reduce for word count

```
map(k,v) {
  split v into words
  for each word w
      emit(w,1)
}
```

```
reduce(k,v) {
   emit(len(v))
}
```

Typically simple functions — easy for app programmer

Example: URL access frequency

Input: request logs

```
GET /dumprequest HTTP/1.1
Host: rve.org.uk
Connection: keep-alive
Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML, like
Gecko) Ubuntu Chromium/25.0.1364.160
Chrome/25.0.1364.160 Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Output:

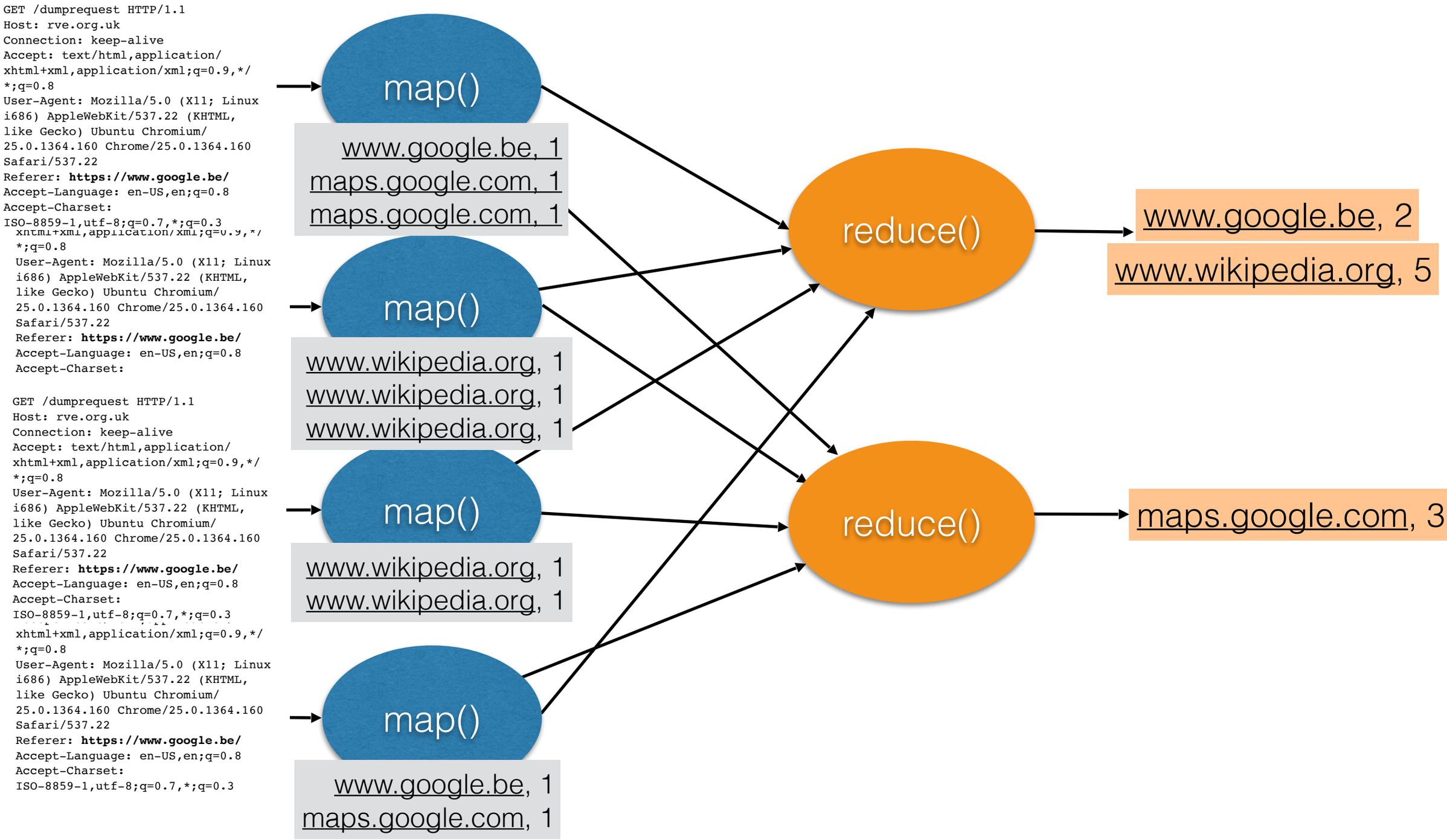
access count per URL

https://www.google.be/, 3567
http://maps.google.com/, 3564
http://www.facebook.com/, 1234

Example: URL access frequency

```
map(String key, String value):
  // key: document name
  // value: document contents
    for each URL u in value:
```

Example: URL access frequency



MapReduce: Implementation

Single master that organizes and orchestrates

Splits input into files stored in a shared networked file system

master

ready and waiting for work

worker computers

worker 0

Master hands out splits and map function to idle workers until all maps are done

worker 1

When work arrives the worker must read in the data and call the map function passing in the file name and contents

MapReduce: Implementation

Single master that organizes and orchestrates

Splits input into files stored in a shared networked file system

master

worker computers

bucket files

worker 0 → 1 2 3

worker 1 → 1 2 3

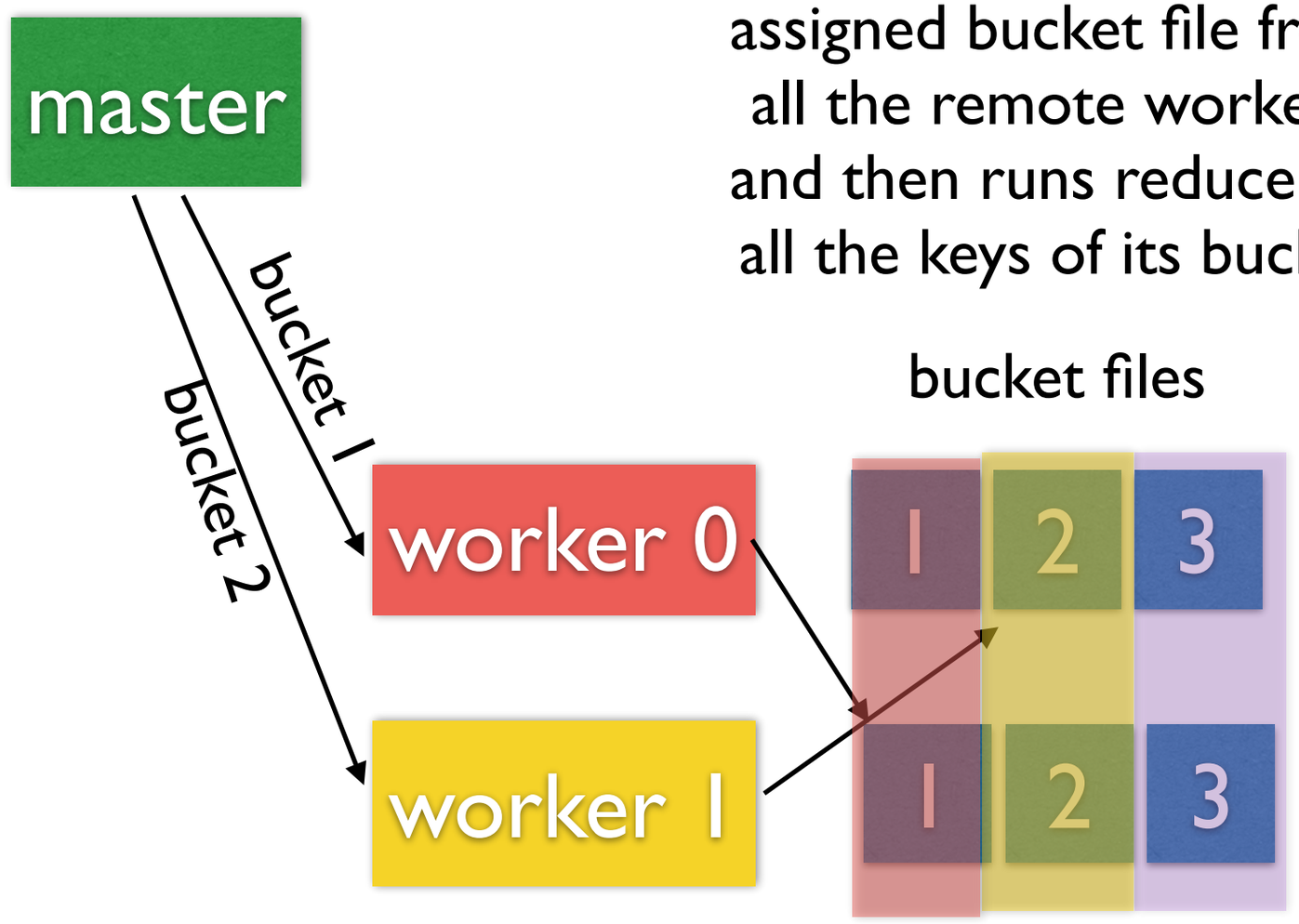
Master decides based on resources how many reduce tasks it will use to do the map processing. (in our case parameter)

tells workers to store map output in buckets of keys on its local disks eg. according to some hash function (one bucket for each reduce task that will be run)

MapReduce: Implementation

Single master that organizes and orchestrates

Master now sends workers reduce jobs specifying which bucket to work on. When jobs complete remaining jobs are handed out to idle workers



Workers fetch it's assigned bucket file from all the remote workers and then runs reduce on all the keys of its bucket

bucket files

Workers store output locally and when complete the master may coordinate a merge