

Map-Reduce

Everything Data

CompSci 290.01 Spring 2014



DUKE
COMPUTER SCIENCE

Announcements (Wed. Mar 25)

- **Homework #10** will be posted by noon tomorrow.
- **Project deadlines:**
 - 3/30: Mid-Term Report is due.
- Monday 3/30: Project work
 - No Lab exercise
 - Use the time to get our input on your project.



2,095,100,000,000 searches in 2014

What do these searches say about us?

EXPLORE THE YEAR IN SEARCH

Trending

Searches

- 1 Robin Williams
- 2 World Cup
- 3 Ebola
- 4 Malaysia Airlines

Trending

People

- 1 Jennifer Lawrence
- 2 Kim Kardashian
- 3 Julie Gayet
- 4 Tracy Morgan

Trending

Athletes

- 1 James Rodriguez
- 2 Michael Schumacher
- 3 Ray Rice
- 4 Luis Suarez

Size of the entire corpus??

Web Images Videos Books Maps More Search tools

About 131,000,000 results (0.27 seconds) **131,000,000 pages mentioning Einstein**

[Albert Einstein - Wikipedia, the free encyclopedia](#)

[en.wikipedia.org/wiki/Albert_Einstein](#) ▾ Wikipedia ▾

Albert **Einstein** was a German-born theoretical physicist. He developed the general theory of relativity, one of the two pillars of modern physics (alongside ...

[Hans Albert Einstein](#) - [General relativity](#) - [Religious views](#) - [Brain](#)

[Albert Einstein - Biographical - Nobelprize.org](#)

[www.nobelprize.org/nobel_prizes/physics/.../einstein-bio.htm...](#) ▾ Nobel Prize ▾

Albert **Einstein** was born at Ulm, in Württemberg, Germany, on March 14, 1879. Six weeks later the family moved to Munich, where he later on began his ...

[News for einstein](#)



[Einstein's lost theory uncovered](#)

Nature.com - 19 hours ago

A manuscript that lay unnoticed by scientists for decades has revealed that Albert **Einstein** once dabbled with an alternative to the Big Bang ...

[Albert Einstein's Lost Theory Resurfaces, Shows His Resistance To Big Bang...](#)

Huffington Post - 19 minutes ago

[Albert Einstein Biography - Facts, Birthday, Life Story - Biography.com](#)

[www.biography.com](#) ▾ People ▾ The Biography Channel ▾

Biography.com offers a glimpse into the life of Albert **Einstein**, the most influential physicist of the 20th century, who developed the theory of relativity.

Albert Einstein

Theoretical Physicist

Albert Einstein was a German-born theoretical physicist. He developed the general theory of relativity, one of the two pillars of modern physics.

[Wikipedia](#)

Born: March 14, 1879, [Ulm, Germany](#)

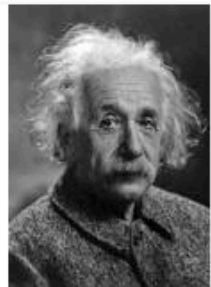
Died: April 18, 1955, [Princeton, NJ](#)

Children: [Eduard Einstein](#), [Hans Albert Einstein](#), [Lieserl Einstein](#)

Spouse: [Elsa Einstein](#) (m. 1919–1936), [Mileva Marić](#) (m. 1903–1919)

Education: [University of Zurich](#) (1905), [More](#)

Awards: [Nobel Prize in Physics](#), [Copley Medal](#), [Franklin Medal](#), [More](#)



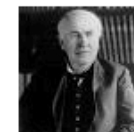
People also search for



Isaac
Newton



Stephen
Hawking



Thomas
Edison

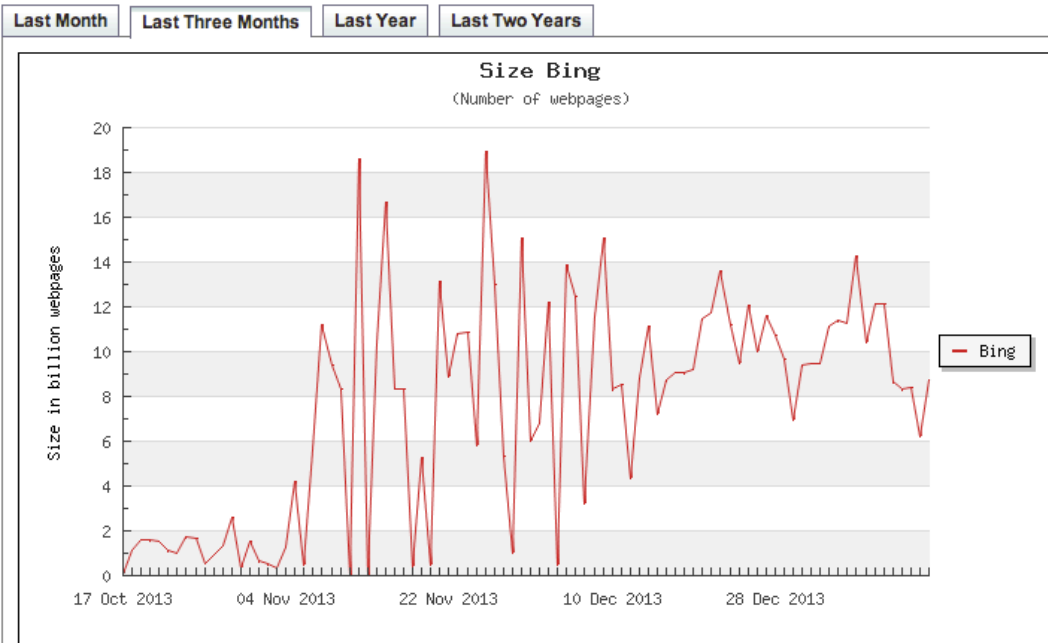
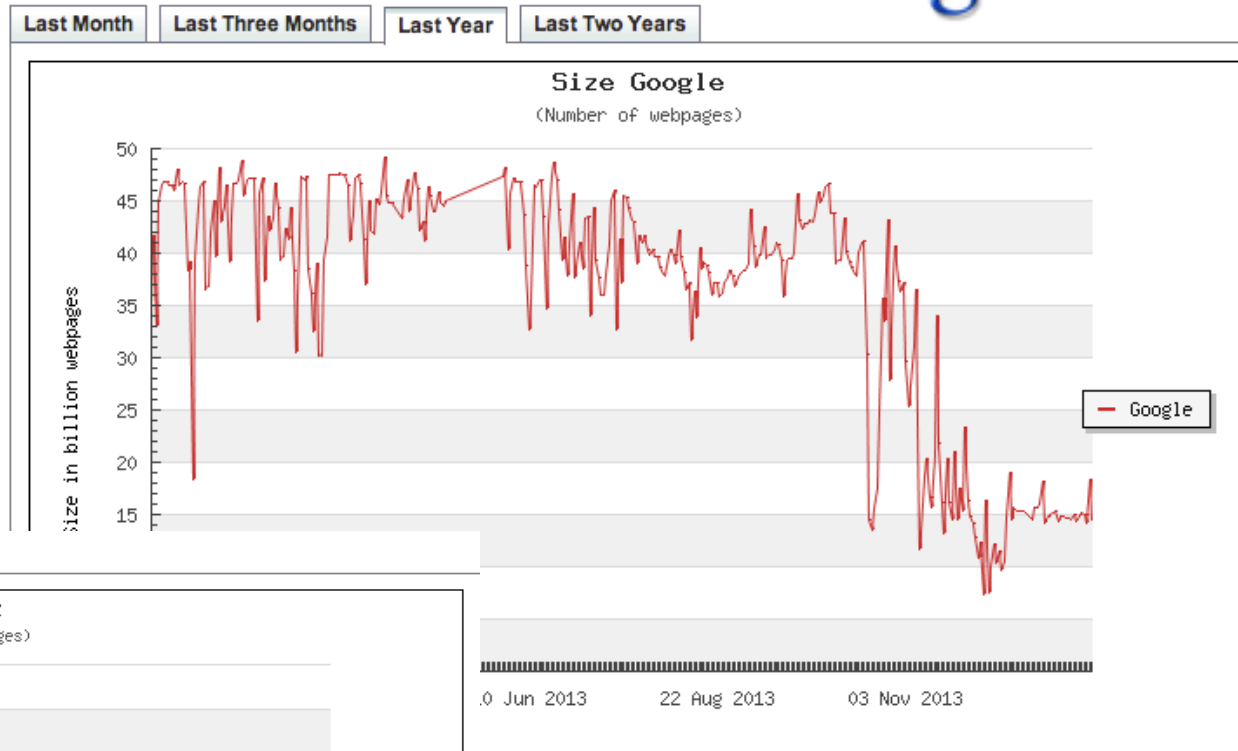


Galileo
Galilei



Marie Curie

Size of the entire corpus??



<http://www.worldwidewebsite.com/>

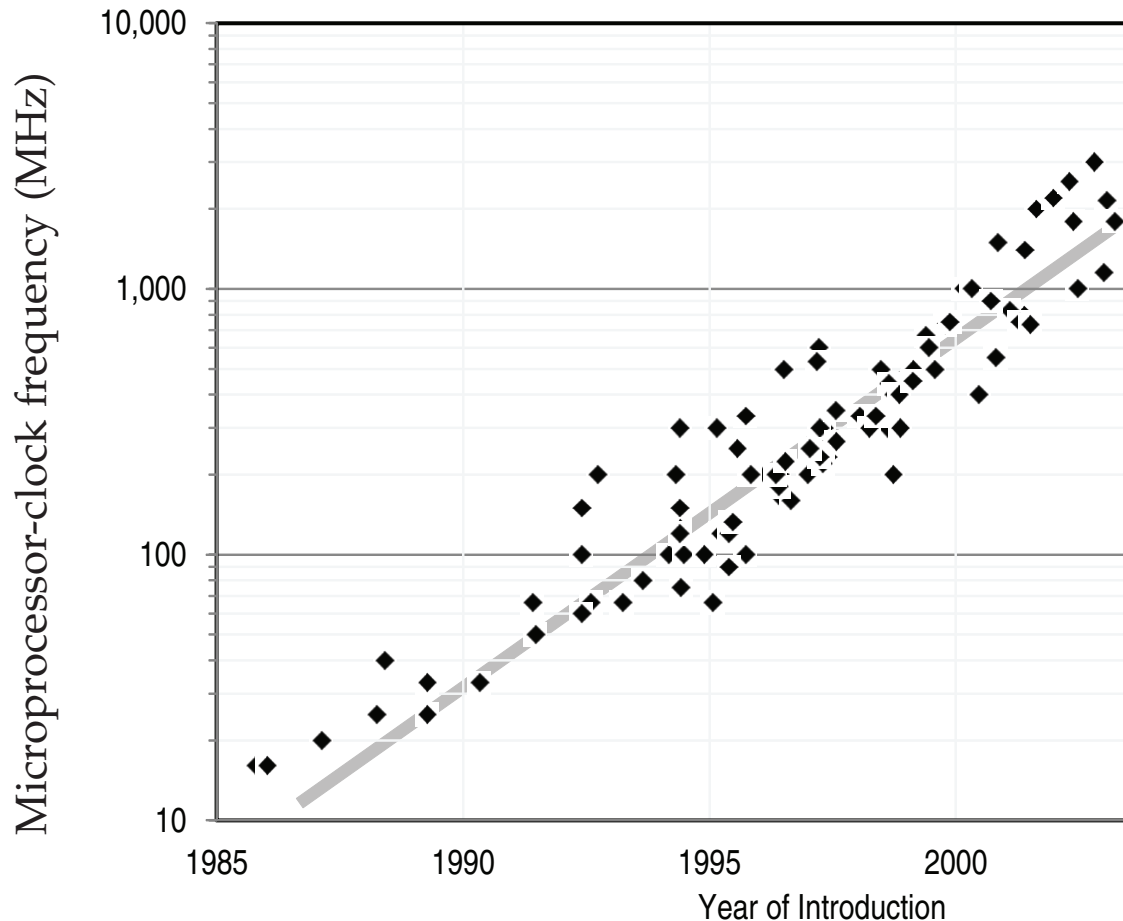
Trend 1: Data centers



[http://**designtaxi.com**/news/353991/Where-The-Internet-Lives-A-Glimpse-Inside-Google-s-Private-Data-Centers/](http://designtaxi.com/news/353991/Where-The-Internet-Lives-A-Glimpse-Inside-Google-s-Private-Data-Centers/)

Trend 2: Multicore

Moore's Law: *# transistors on integrated circuits doubles every 2 years*



Need to think “parallel”

- Data resides on different machines
- Split computation onto different machines/cores

But ... parallel programming is hard!

Low level code needs to deal with a lot of issues ...

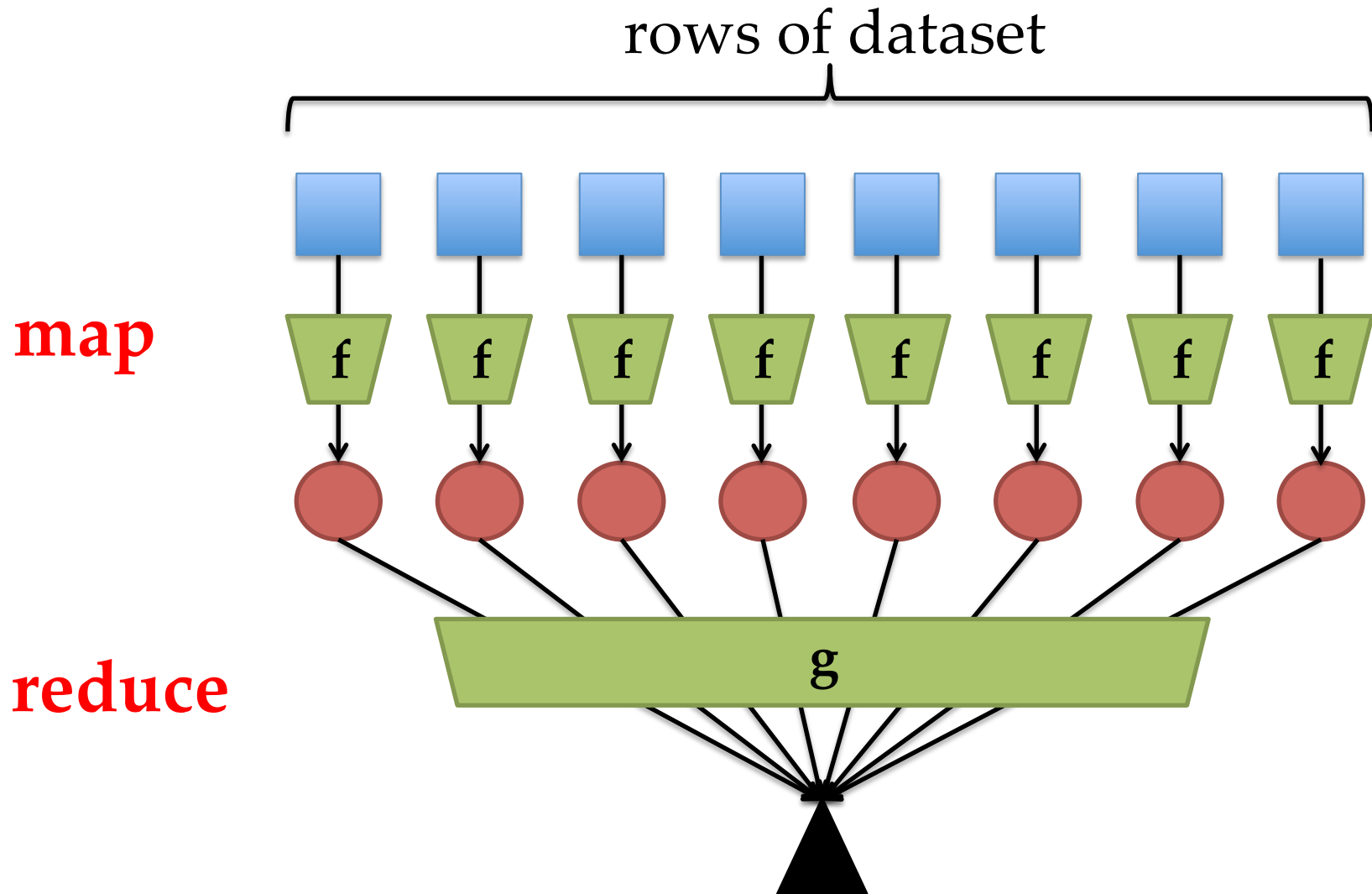
- Failures
 - Loss of computation
 - Loss of data
- Concurrency
- ...

Map-Reduce

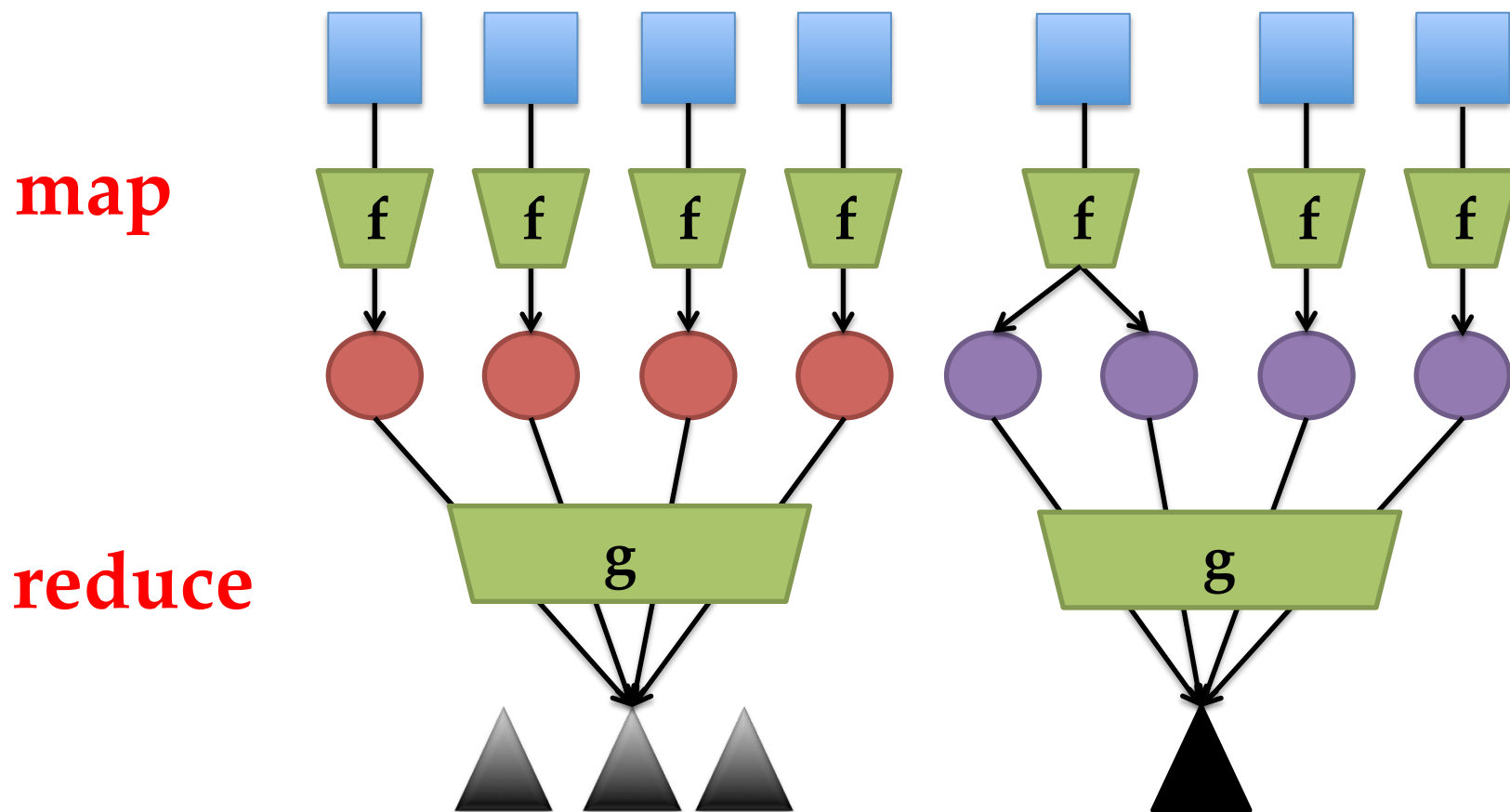
Programming Model + Distributed System

- Simple model
 - Programmer only describes the logic
- Works on commodity hardware
 - Scales to thousands of machines
 - Ship code to the data, rather than ship data to code
 - Hides all the hard systems problems from the programmer
 - Machine failures
 - Data placement
 - ...

Map-Reduce Programming Model



Map-Reduce Programming Model

$$\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$
$$\text{reduce}(k_2, \text{list}(v_1)) \rightarrow \text{list}(k_3, v_3)$$


Example 1: Word Count

- Input: A set of documents, each containing a list of words
 - Each document is a row
 - E.g., search queries, tweets, reviews, etc.
- Output: A list of pairs $\langle w, c \rangle$
 - c is the number of times w appears across all documents.

Word Count: Map

$\langle \text{docid}, \{\text{list of words}\} \rangle \rightarrow \{\text{list of } \langle \text{word}, 1 \rangle\}$

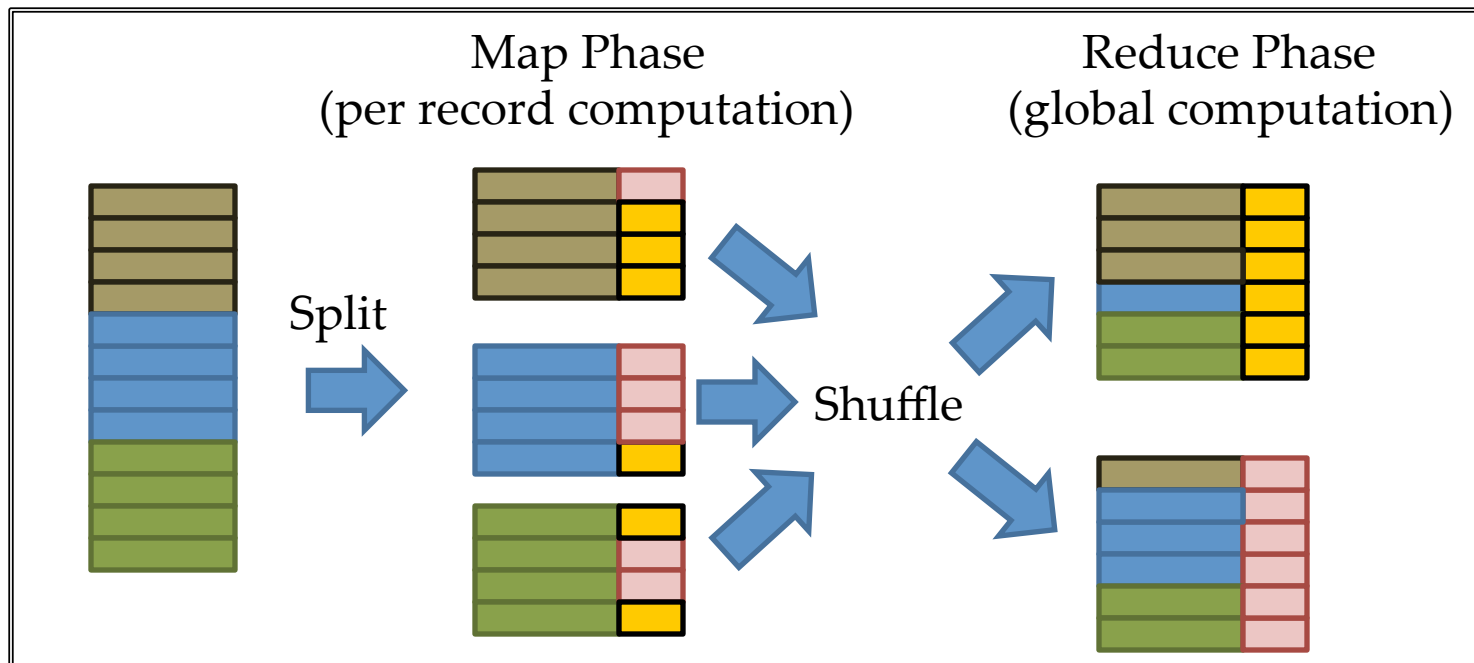
- *The mapper takes a document d and creates n key value pairs, one for each word in the document.*
- *The output key is the word*
- *The output value is 1*
 - *(count of each appearance of a word)*

Word Count: Reduce

$\langle \text{word}, \{\text{list of counts}\} \rangle \rightarrow \langle \text{word}, \text{sum}(\text{counts}) \rangle$

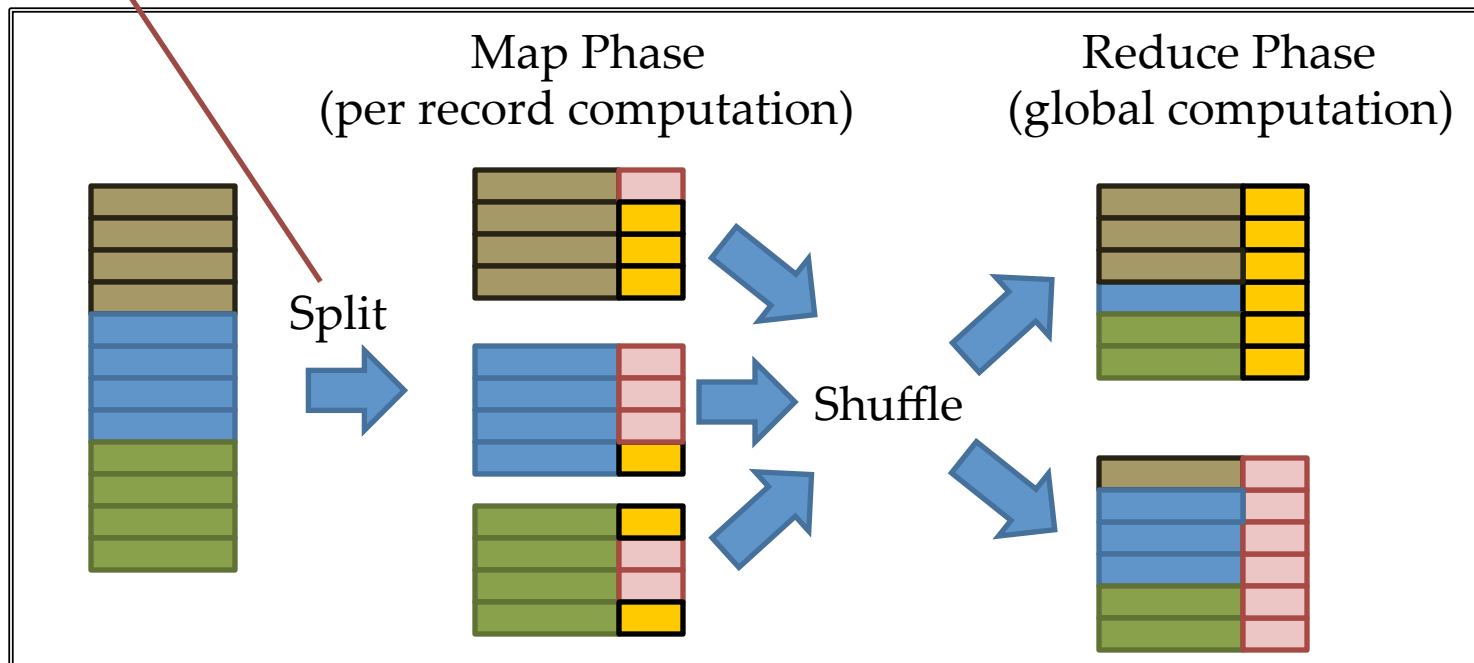
- *The reducer aggregates the counts (in this case 1) associated with a specific word.*

Map-Reduce Implementation

$$\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$
$$\text{reduce}(k_2, \text{list}(v_1)) \rightarrow \text{list}(k_3, v_3)$$


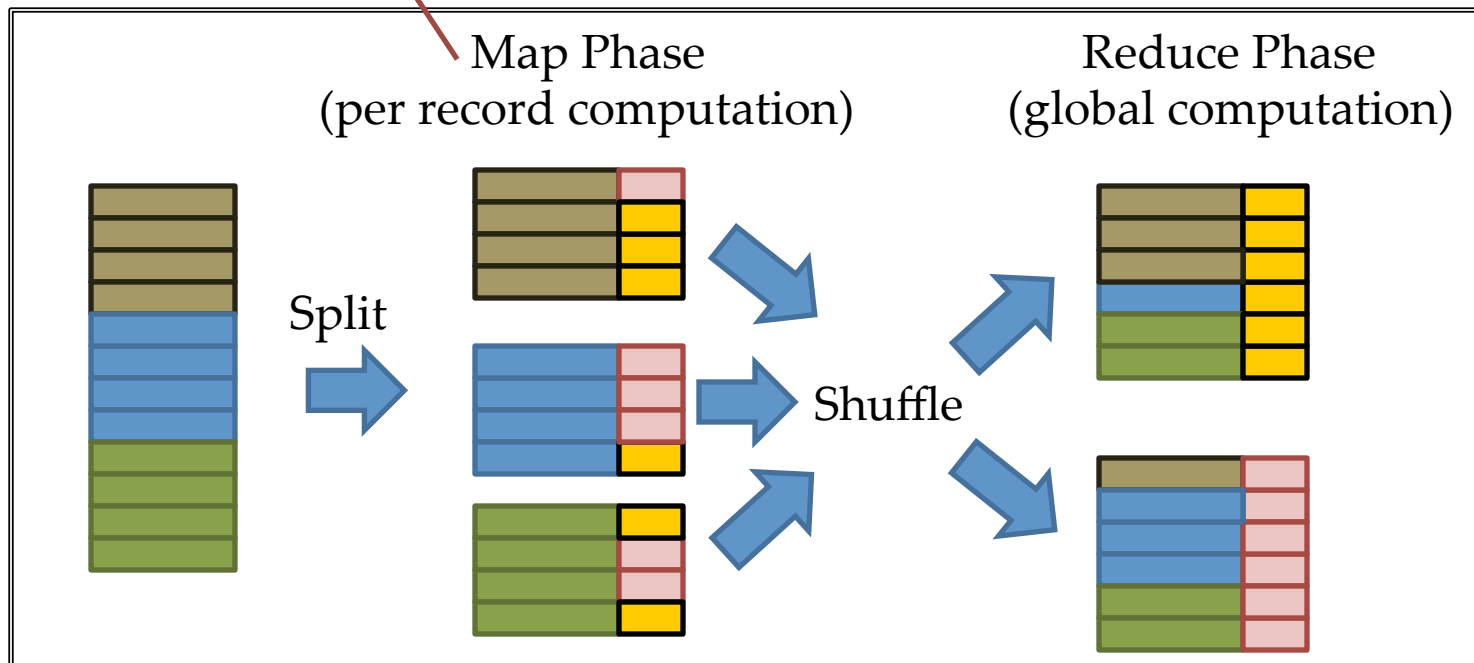
Map-Reduce Implementation

Split phase partitions the data across different mappers (... *think different machines*)



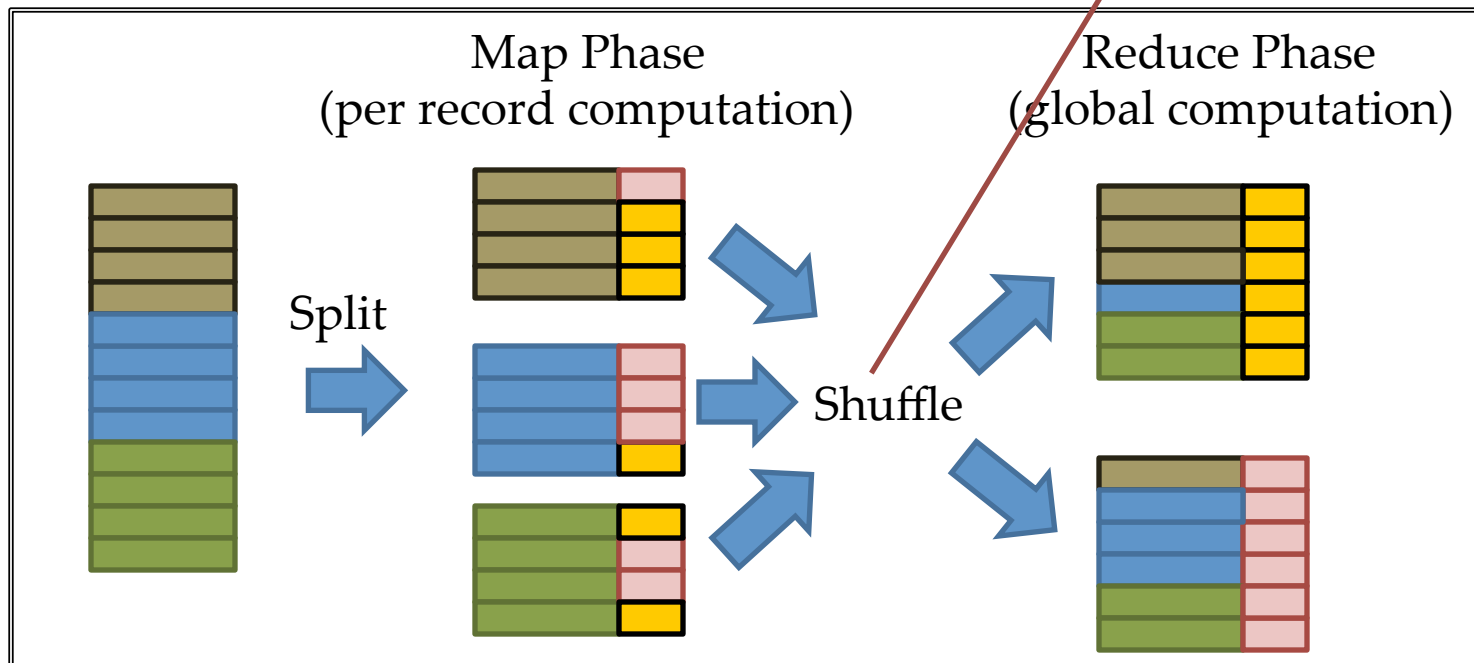
Map-Reduce Implementation

Each mapper executes user defined map code on the partitions in parallel



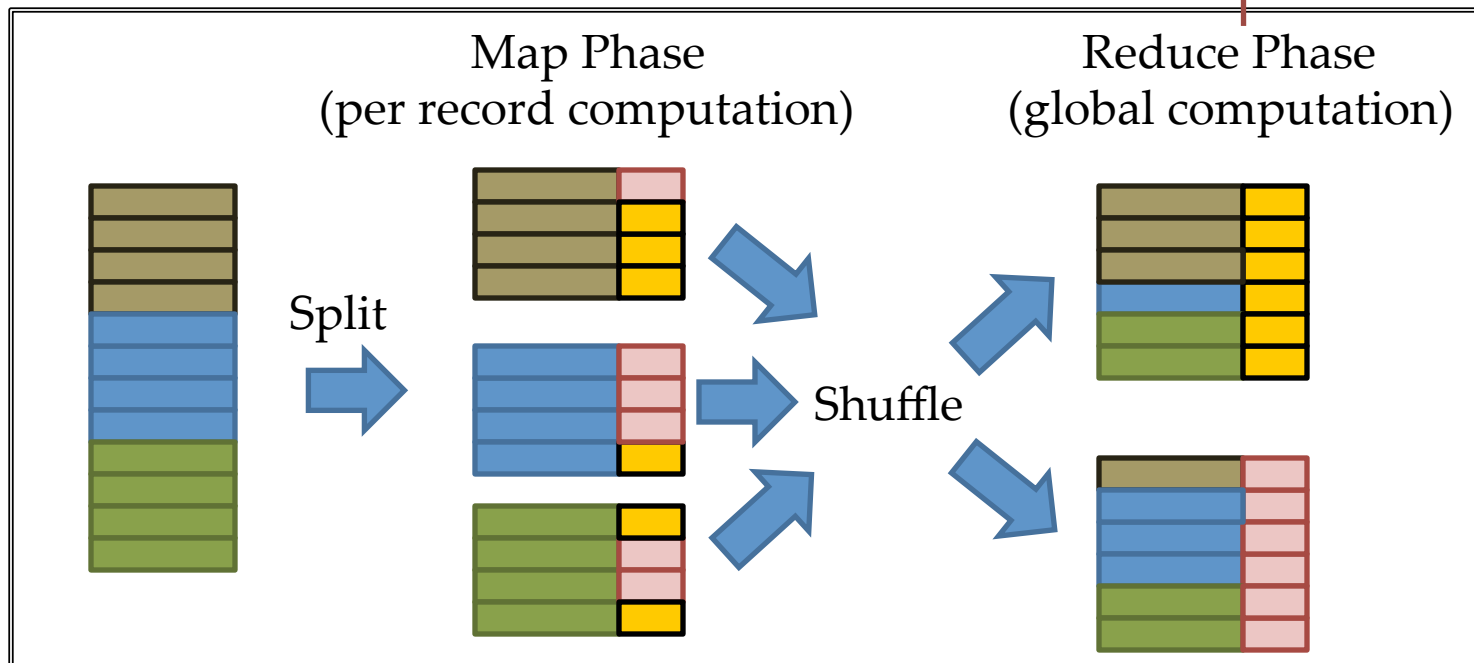
Map-Reduce Implementation

Data is shuffled such that there is one reducer per output key (... *again think different machines*)



Map-Reduce Implementation

Each reducer executes the user defined reduce code in parallel.



Map Reduce Implementation

- After every map and reduce phase, data is written onto disk
 - If machines fail during the reduce phase, then no need to rerun the mappers.

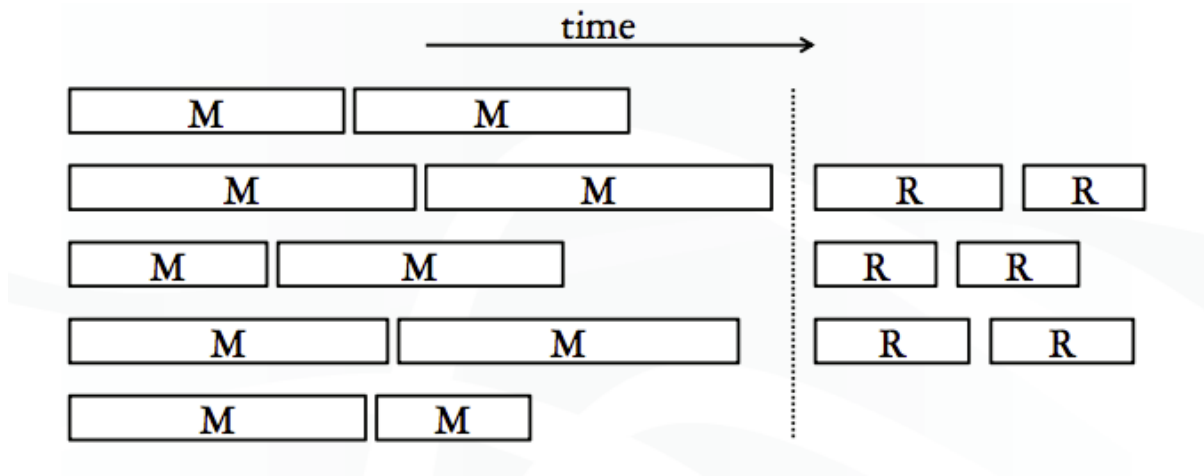
Writing to disk is *slow*.
Should minimize number of map-reduce phases.

Mappers, Reducers and *Workers*

- Physical machines are called *workers*
- Multiple mappers and reducers can run on the same worker.
- More workers implies ...
 - ... *more parallelism (faster computation)* ...
 - ... *but more (communication) overhead* ...

Map Reduce Implementation

- All reducers start only after all the mappers complete.



- Straggler: A mapper or reducer that takes a long time

Back to Word Count

- Map:
 $\langle \text{docid}, \{\text{list of words}\} \rangle \rightarrow \{\text{list of } \langle \text{word}, 1 \rangle\}$
- Reduce:
 $\langle \text{word}, \{\text{list of counts}\} \rangle \rightarrow \langle \text{word}, \text{sum}(\text{counts}) \rangle$
- *Number of records output by the map phase equals the number of words across all documents.*

Map-Combine-Reduce

- Combiner is a mini-reducer within each mapper.
 - Helps when the reduce function is commutative and associative.
- Aggregation within each mapper reduces the communication cost.

Word Count ... *with combiner*

- Map: **Mapper**
 $\langle \text{docid}, \{\text{list of words}\} \rangle \rightarrow \{\text{list of } \langle \text{word}, 1 \rangle\}$

- Combine:
 $\langle \text{word}, \{\text{list of counts}\} \rangle \rightarrow \langle \text{word}, \text{sum}(\text{counts}) \rangle$

- Reduce:
 $\langle \text{word}, \{\text{list of counts}\} \rangle \rightarrow \langle \text{word}, \text{sum}(\text{counts}) \rangle$

Reducer

Word Count ... *in python*

```
"""The classic MapReduce job: count the frequency of words.
"""
```

```
from mrjob.job import MRJob
import re
```

```
WORD_RE = re.compile(r"[\w']+")
```

```
class MRWordFreqCount(MRJob):
```

```
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)
```

```
    def combiner(self, word, counts):
        yield (word, sum(counts))
```

```
    def reducer(self, word, counts):
        yield (word, sum(counts))
```

```
if __name__ == '__main__':
    MRWordFreqCount.run()
```

One of the many MapReduce
libraries for python

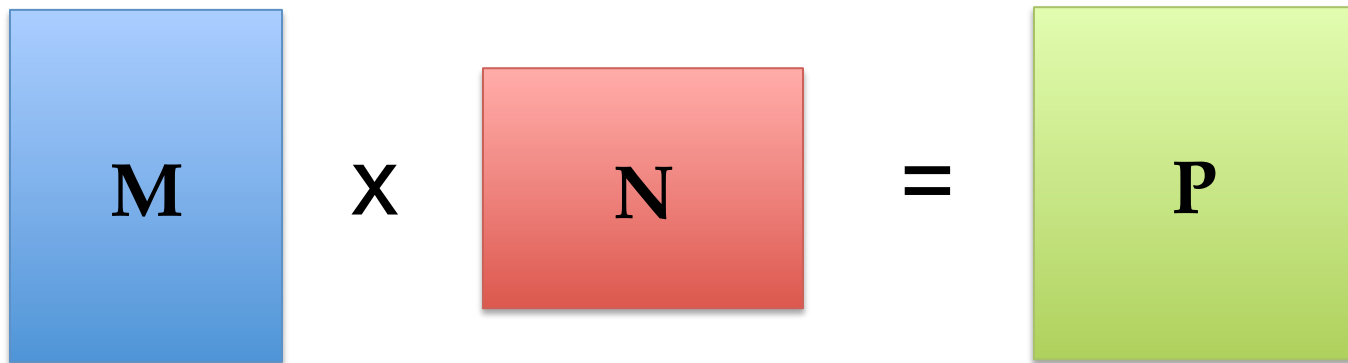
Example 2: K most frequent words

- Need multiple Map-Reduce steps
- Map:
 $\langle \text{docid}, \{\text{list of words}\} \rangle \rightarrow \{\text{list of } \langle \text{word}, 1 \rangle\}$
- Reduce:
 $\langle \text{word}, \{\text{list of counts}\} \rangle \rightarrow \langle _ , (\text{word}, \text{sum}(\text{counts})) \rangle$
- Reduce:
 $\langle _ , \{\text{list of } (\text{word}, \text{count}) \text{ pairs}\} \rangle \rightarrow$
 $\langle _ , \{\text{list of words with k most frequent counts}\} \rangle$

Example 3: Distributed Grep

- Input: String
- Output: {list of lines that match the string}
- Map:
 $\langle \text{lineid}, \text{line} \rangle \rightarrow \langle \text{lineid}, \text{line} \rangle$ *// if line matches string*
- Reduce:
// do nothing

Example 4: Matrix Multiplication



$$p_{ik} = \sum_j m_{ij} n_{jk}$$

Matrix Multiplication

- Assume the input format is $\langle \text{matrix id}, \text{row}, \text{col}, \text{entry} \rangle$
 - E.g.: $\langle M, i, j, m_{ij} \rangle$
- Map:

$$\langle _ , (M, i, j, m_{ij}) \rangle \rightarrow \langle (i,k), (M, i, m_{ij}) \rangle \dots \text{for all } k$$

$$\langle _ , (N, j, k, n_{jk}) \rangle \rightarrow \langle (i,k), (N, k, n_{jk}) \rangle \dots \text{for all } i$$
- Reduce:

$$\langle (i,k), \{(M, i, j, m_{ij}), (N, j, k, n_{jk}) \dots\} \rangle$$

$$\rightarrow \{ \langle (i,k), \sum_j m_{ij} n_{jk} \rangle \}$$

Example 5: Join two tables

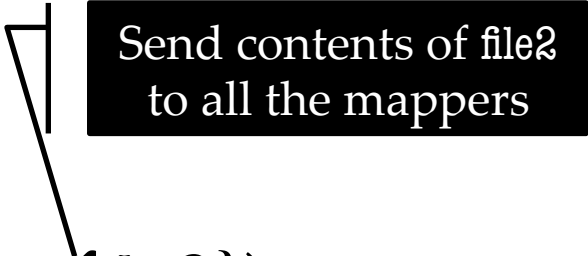
- Input: file1 and file2, with schema $\langle key, value \rangle$
- Output: *keys* appearing in both files.
- Map:
 - $\langle _, (file1, key, value) \rangle \rightarrow (key, file1)$
 - $\langle _, (file2, key, value) \rangle \rightarrow (key, file2)$
- Reduce:
 - $\langle key, \{list\ of\ fileids\} \rangle \rightarrow \langle _, key \rangle$
 - // if list contains both file1 and file2.*

Map-side Join

- Suppose file2 is very small ...

- Map:

$\langle _, (\text{file1}, \text{key}, \text{value}, \{\text{keys in file2}\}) \rangle$
 $\rightarrow (_, \text{key})$



Send contents of file2
to all the mappers

// If key is also in file2

- Reduce: *// do nothing*

$\langle _, \{\text{list of keys}\} \rangle \rightarrow \langle _, \{\text{list of keys}\} \rangle$

Example 5: Join 3 tables

- Input: 3 Tables
 - User (id:int, age:int)
 - Page (url:varchar, category:varchar)
 - Log (userid: int, url:varchar)
- Output: Ages of users and types of urls they clicked.

Multiway Join

- `Join(Page, Join (User , Log))`
- Map:
 - $\langle _, (User, id, age) \rangle \rightarrow (id, (User, key, value))$
 - $\langle _, (Log, userid, url) \rangle \rightarrow (userid, (Log, userid, url))$
- Reduce:
 - $\langle id, \{list\ of\ records\} \rangle$
 $\rightarrow \langle _, \{records\ from\ User\} \times \{records\ from\ Log\} \rangle$
// if list contains records both from User and Log.
- Map:
 - $\langle _, (User, Log, id, age, userid, url) \rangle \rightarrow (url, (User, Log, id, age, userid, url))$
 - $\langle _, (Page, url, category) \rangle \rightarrow (url, (Page, url, category))$
- Reduce:
 - $\langle url, \{list\ of\ records\} \rangle$
 $\rightarrow \langle _, \{records\ from\ User\ x\ Log\} \times \{records\ from\ Page\} \rangle$
// if list contains records both from User x Log and Page.

Summary

- Map-reduce is a programming model + distributed system implementation that make parallel programming easy.
 - Programmer does not need to worry about systems issues.
- Computation is a series of Map and Reduce jobs.
 - Parallelism is achieved within each Map and Reduce phase.