Map-Reduce

Everything Data

CompSci 290.01 Spring 2014

DUKE
COMPUTER SCIENCE

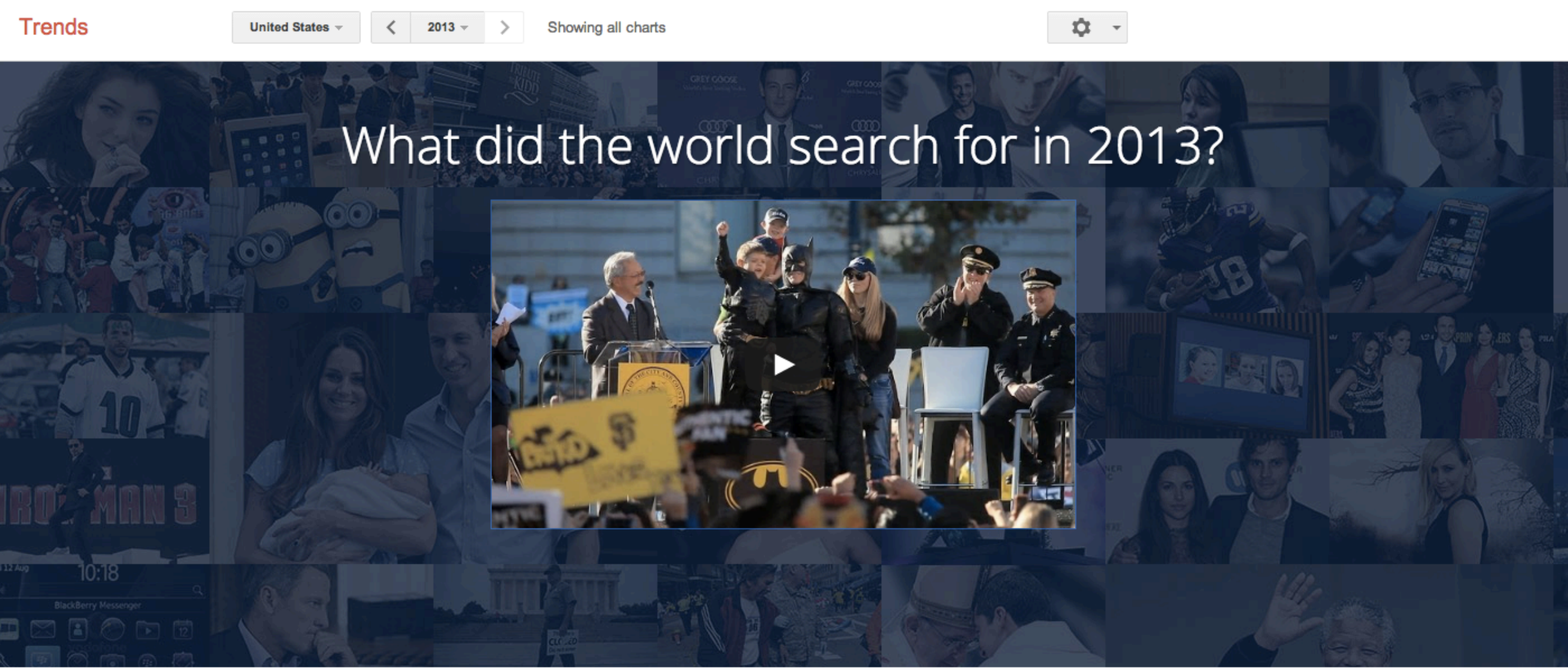
Announcements (Thu. Feb 27)

- **Homework #8** will be posted by noon tomorrow.

- **Project deadlines**:
  - 2/25: Project team formation
  - 3/4: Project Proposal is due.
  - 3/4: 2 minute presentation in class

2,161,530,000,000 searches in 2013

Google



Size of the entire corpus??

Web Images Videos Books Maps More Search tools

About 131,000,000 re **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 ...

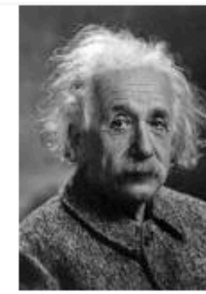
[News for **einstein**](#)

[**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

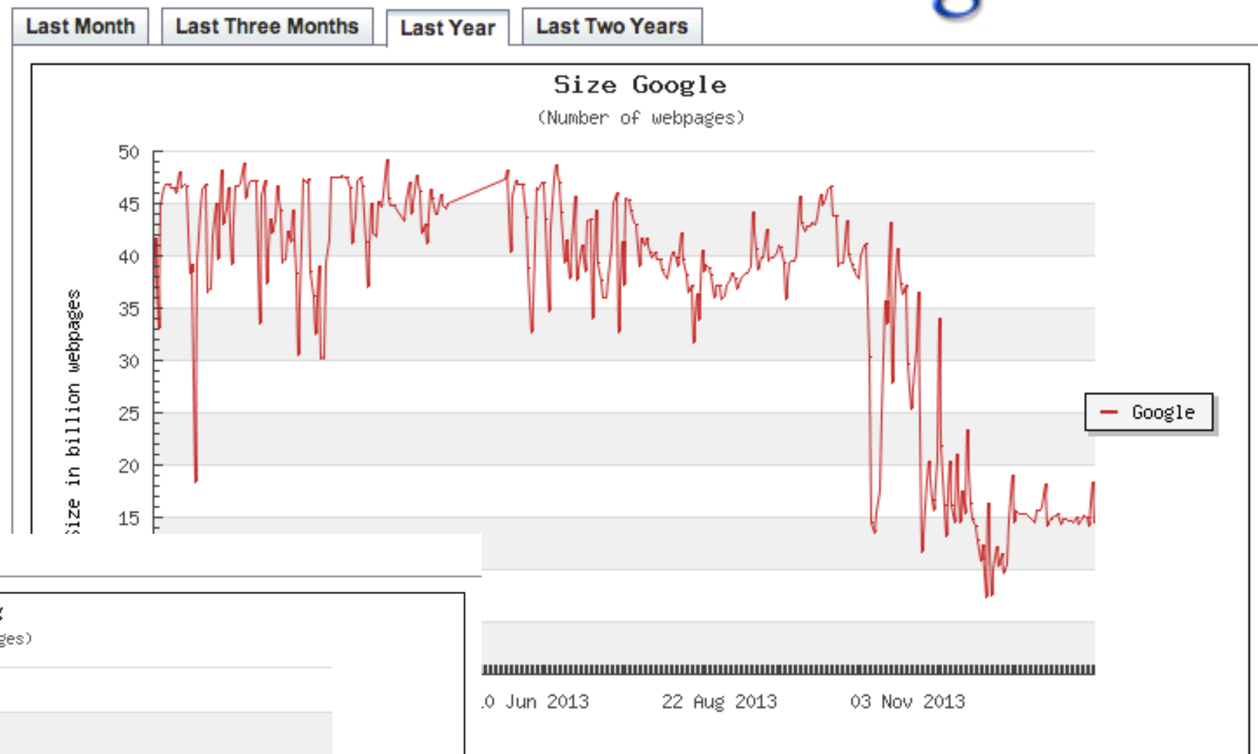

Stephen Hawking

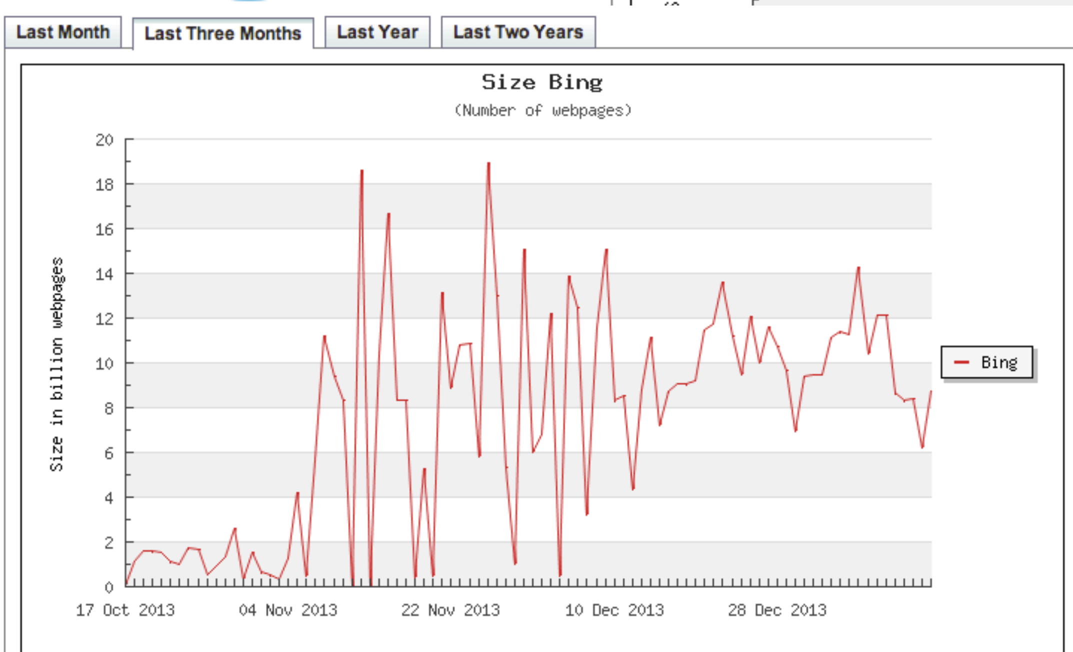

Thomas Edison


Galileo Galilei


Marie Curie

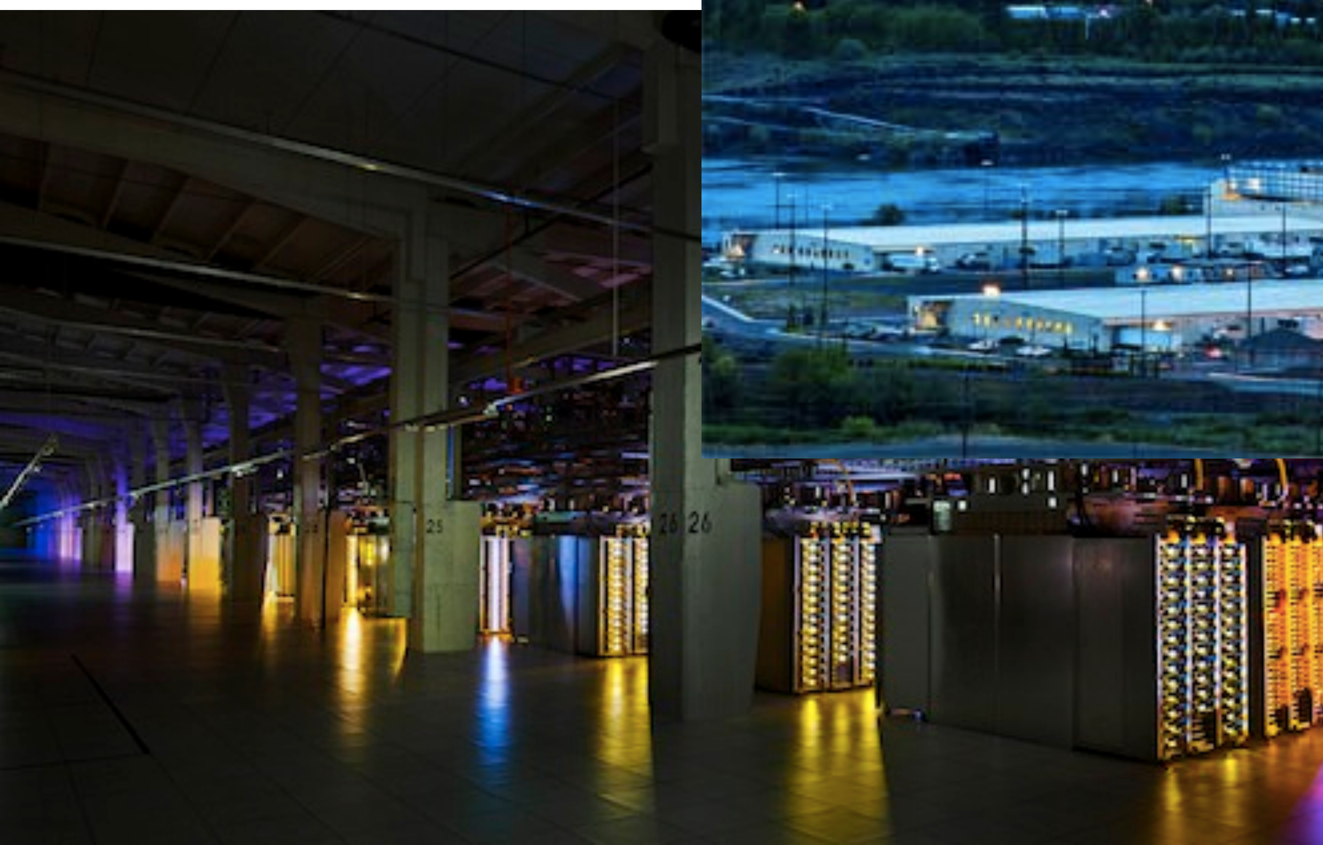
Size of the entire corpus??





http://www.worldwidewebsize.com/

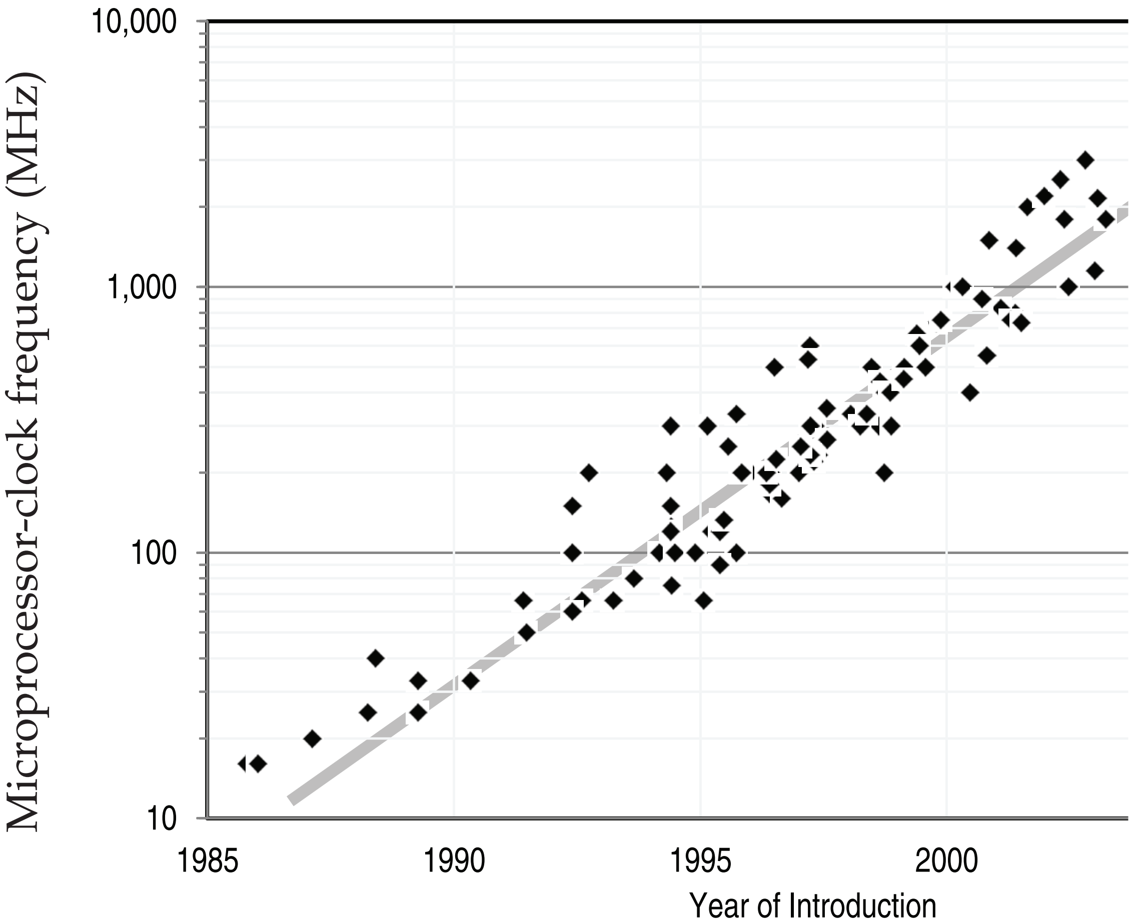
Trend 1: Data centers



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*



http://www.**nap.edu**/catalog.php?record_id=12980

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
- …

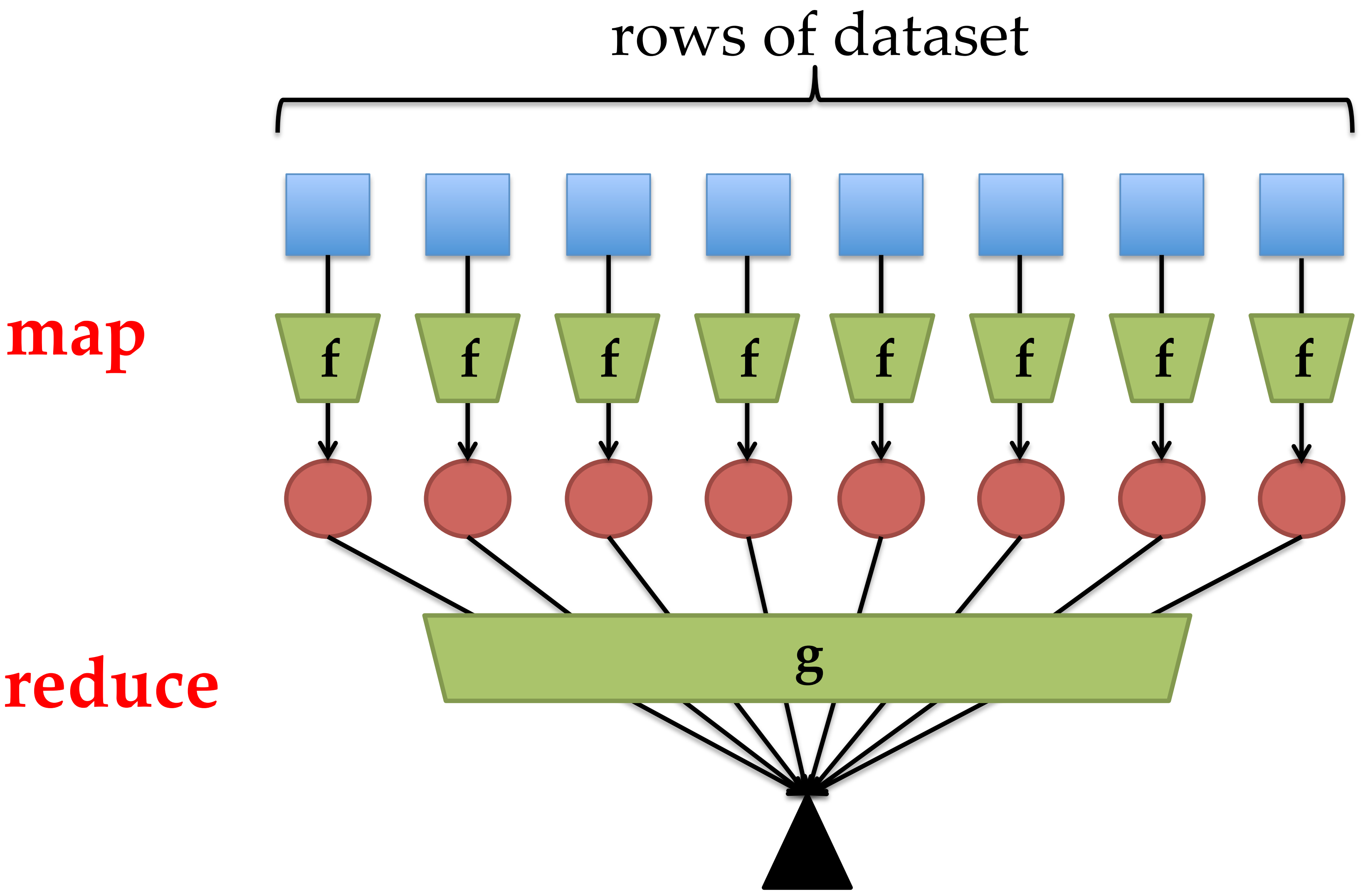
Map-Reduce

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
  - …

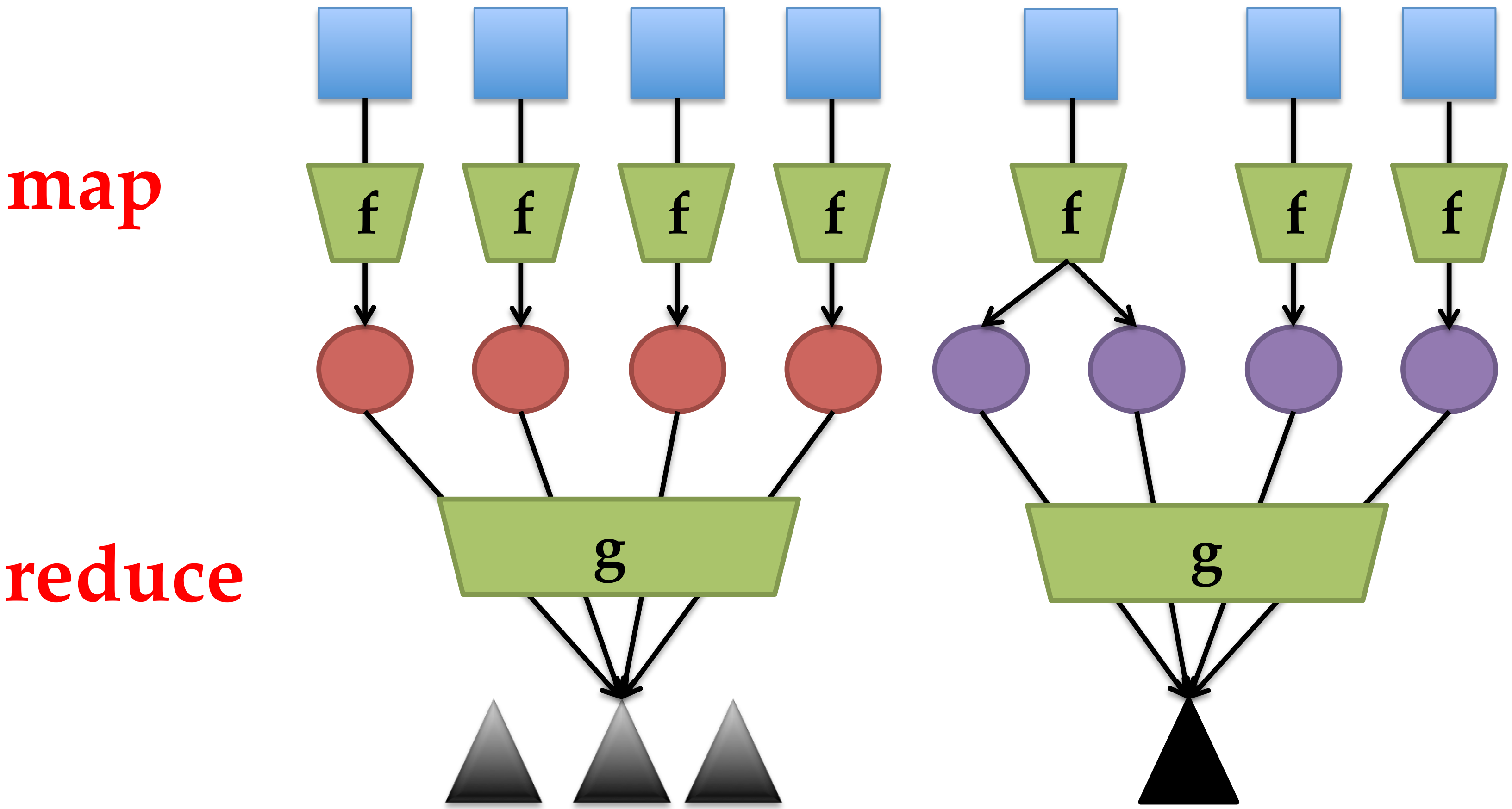
Map Reduce Programming Model



Map-Reduce Programming Model

$$\mathrm{map}(k_1, v_1) \rightarrow \mathrm{list}(k_2, v_2)$$

$$\mathrm{reduce}(k_2, \mathrm{list}(v_1)) \rightarrow \mathrm{list}(k_3, v_3)$$



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 <w, c>
  - c is the number of times w appears across all documents.

Word Count: Map

<docid, {list of words}> → {list of <word, 1>}

- *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

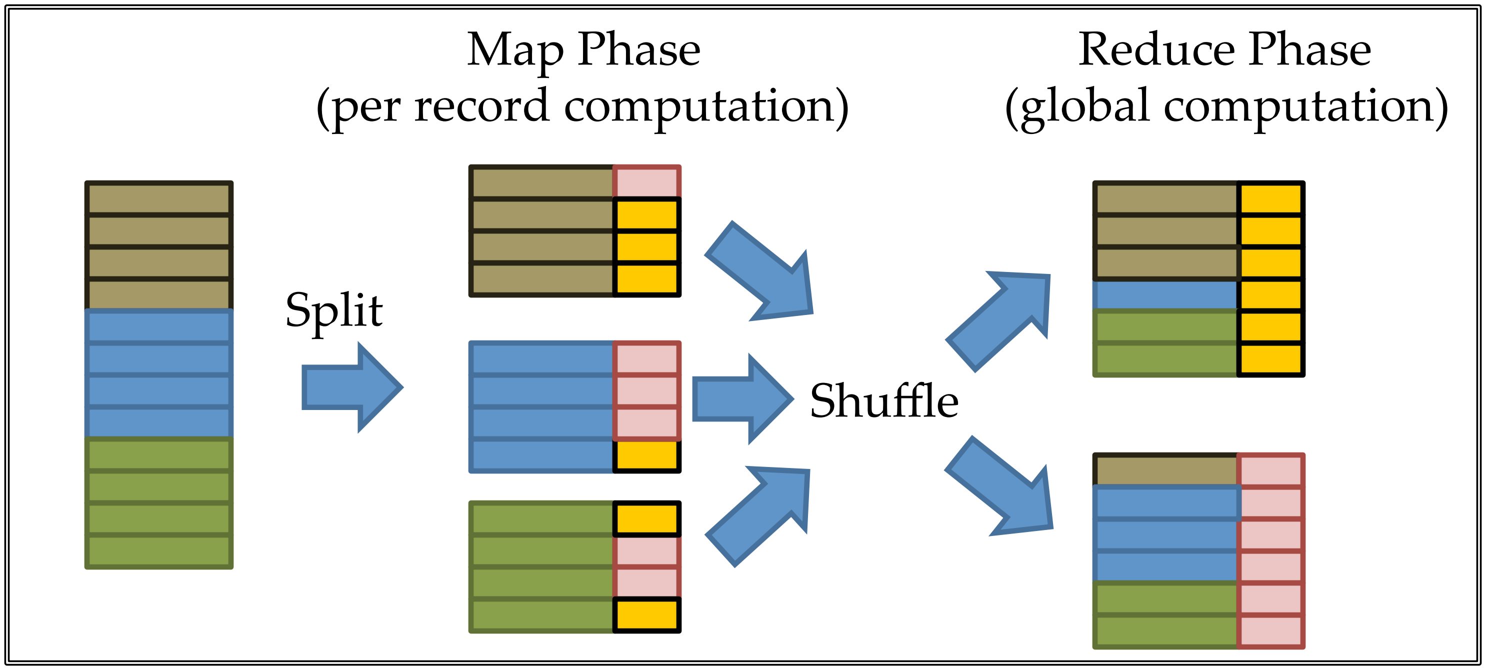
<word, {list of counts}> → <word, sum(counts)>

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

Map-Reduce Implementation

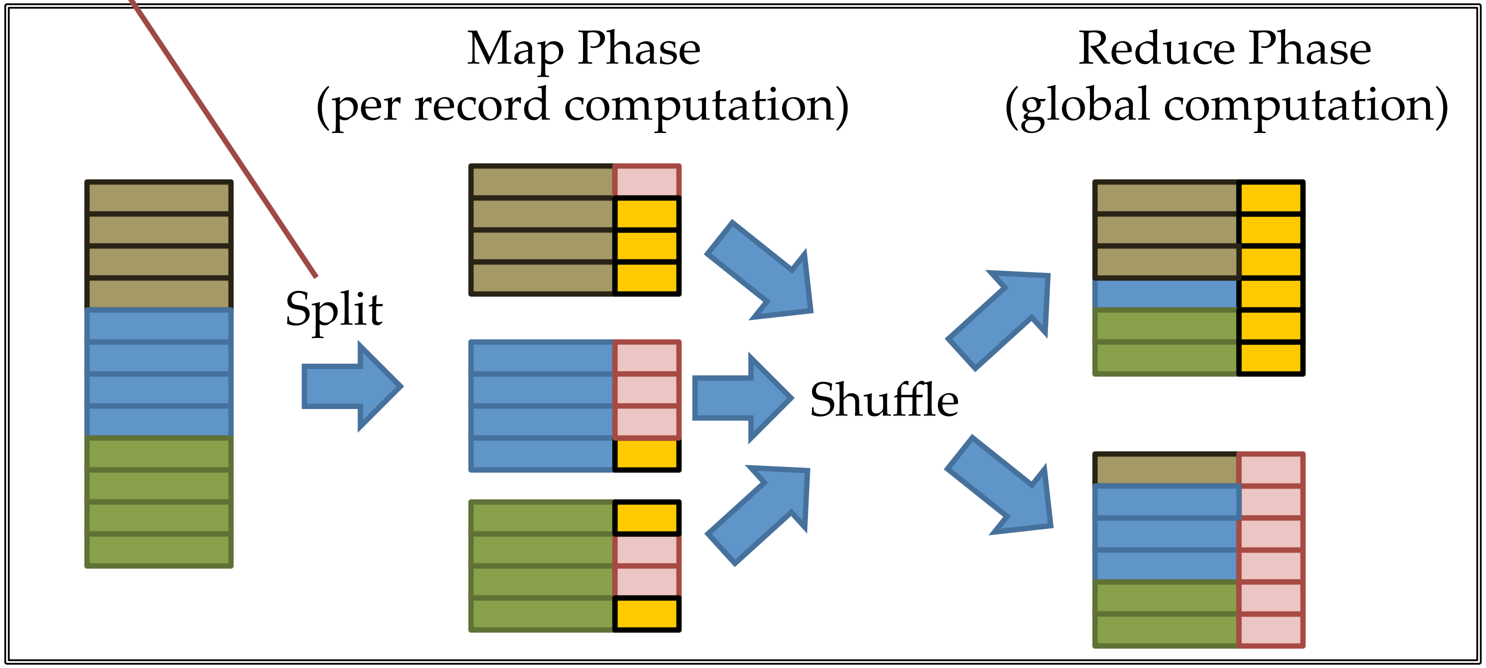
$$\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$

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



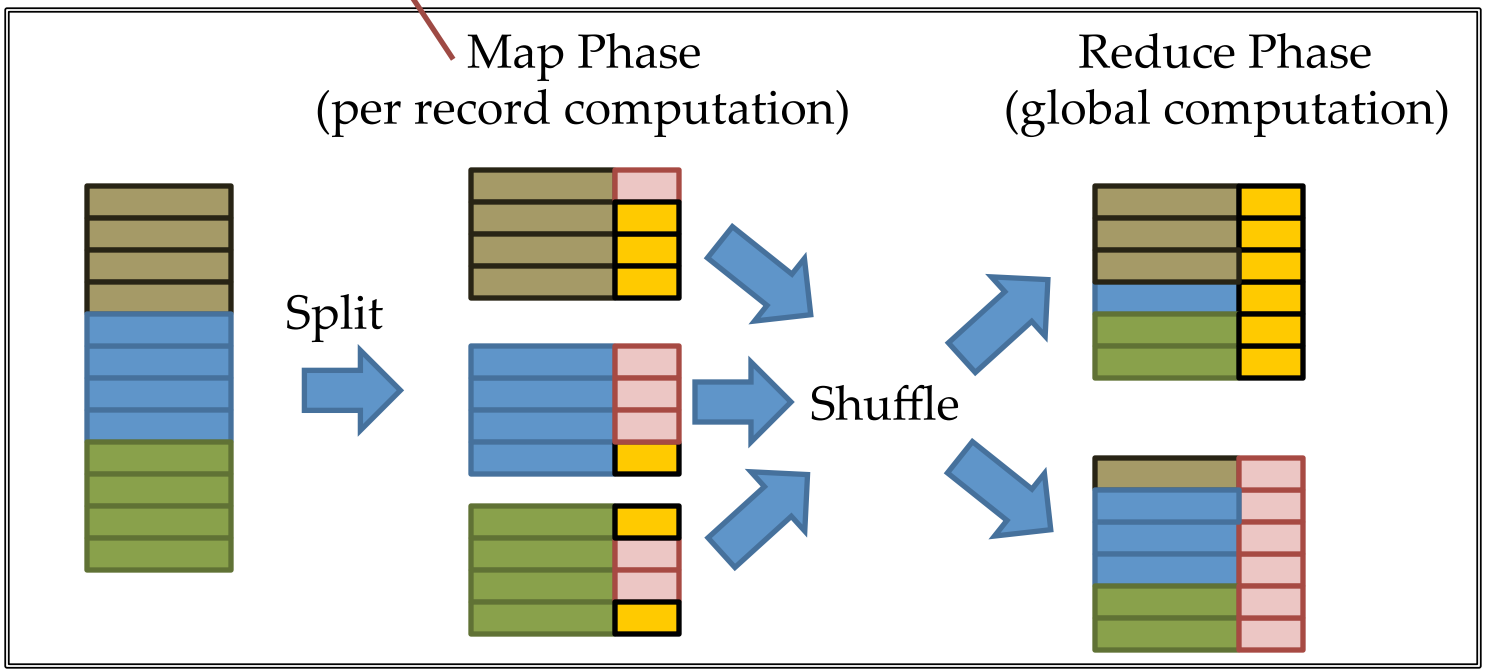
Map-Reduce Implementation

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



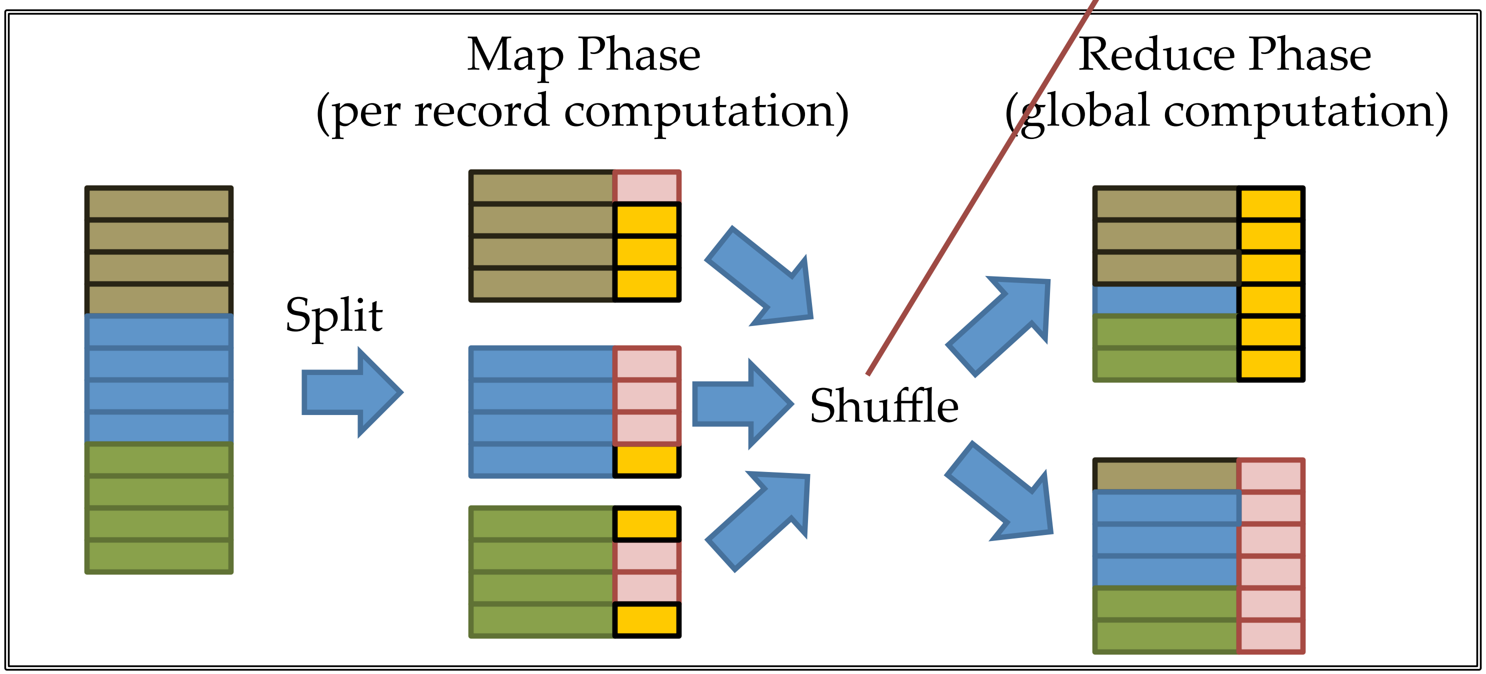
Map-Reduce Implementation

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



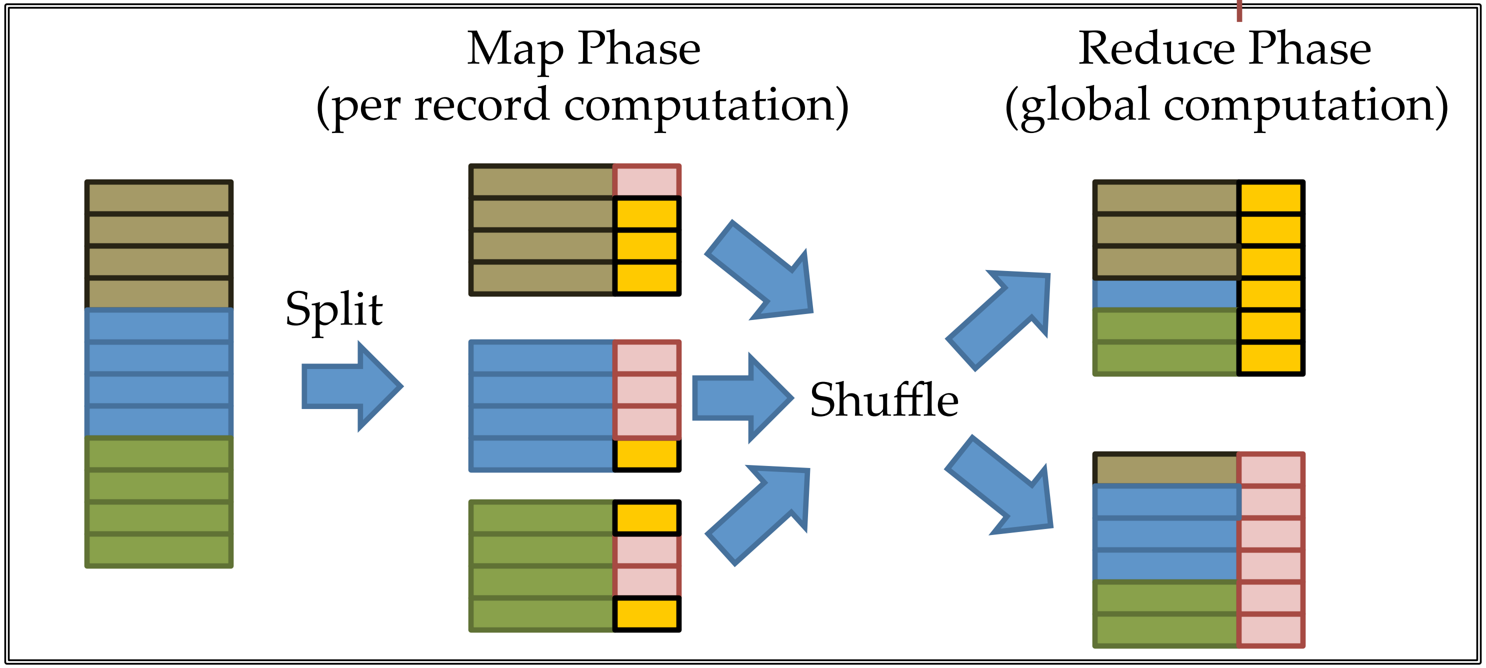
Map-Reduce Implementation

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.

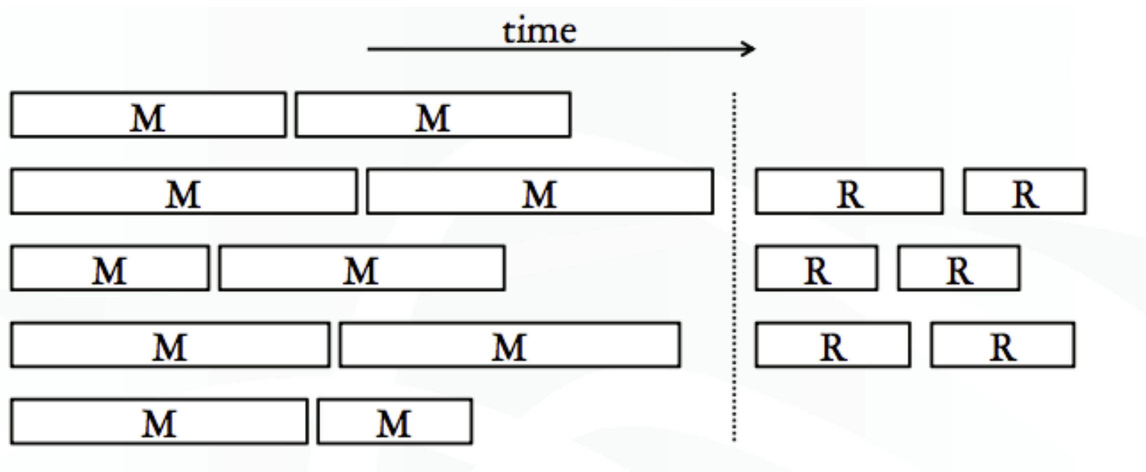
- More workers implies …

  … 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:
  <docid, {list of words}> → {list of <word, 1>}
- Reduce:
  <word, {list of counts}> → <word, sum(counts)>

- *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*

Mapper

- Map:
  <docid, {list of words}> → {list of <word, 1>}

- Combine:
  <word, {list of counts}> → <word, sum(counts)>

- Reduce:
  <word, {list of counts}> → <word, sum(counts)>

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:
  <docid, {list of words}> → {list of <word, 1>}

- Reduce:
  <word, {list of counts}> → <_ , (word, sum(counts))>

- Reduce:
  <_ , {list of (word, count) pairs}> →
  <_ , {list of words with k most frequent counts}>

Example 3: Distributed Grep

- Input: String
- Output: {list of lines that match the string}

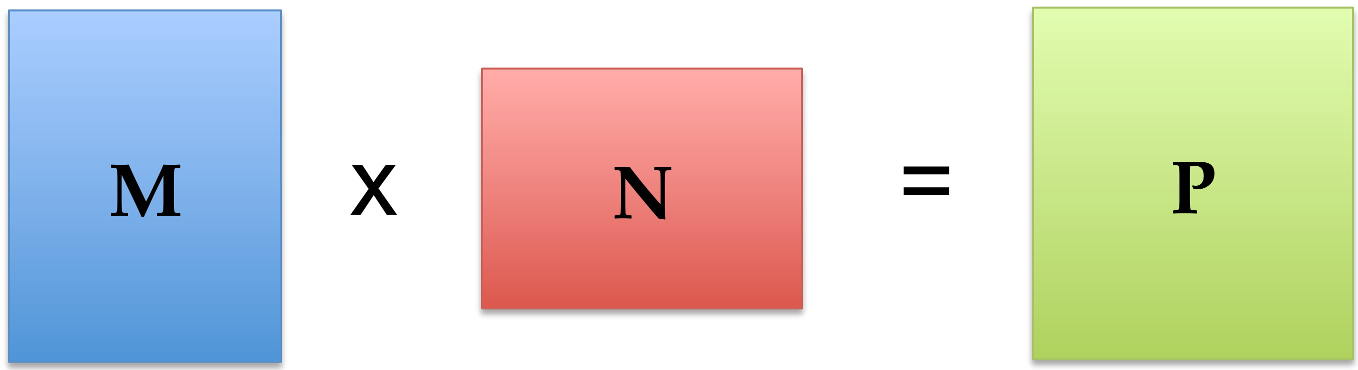
- Map:

  <lineid, line> → <lineid, line> // *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 *<matrix id, row, col, entry>*
    - E.g.: $<M, i, j, m_{ij}>$

- Map:

  $<\_\ , (M, i, j, m_{ij})> \rightarrow <(i,k), (M, i, m_{ij})>$ … *for all k*
  $<\_\ , (N, j, k, n_{jk})> \rightarrow <(i,k), (N, k, n_{jk})>$ … *for all i*

- Reduce:

  $<(i,k), \{(M, i, j, m_{ij}), (N, j, k, n_{jk}) \ldots\}>$
  $\rightarrow \{ <(i,k), \Sigma_j m_{ij} n_{jk}> \}$

Example 5: Join two tables

- Input: file1 and file2, with schema <*key, value*>
- Output: *keys* appearing in both files.

- Map:

  <_ , (file1, key, value)> → (key, file1)

  <_ , (file2, key, value)> → (key, file2)
- Reduce:

  <key , {list of fileids}> → <_, key>

  // *if list contains both* file1 *and* file2.

Map-side Join

- Suppose file2 is very small …

Send contents of file2 to all the mappers

- Map:

  <_ , (file1, key, value, {keys in file2})> → (_ , key)

  // *If key is also in* file2

- Reduce: // *do nothing*

  <_ , {list of keys}> → <_ , {list of keys}>

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:
  <_ , (User, id, age)> → (id, (User, key, value))
  <_ , (Log, userid, url)> → (userid, (Log, userid, url))
- Reduce:
  <id, {list of records}>
  → <_, {records from User} x {records from Log}>
  // if list contains records both from User *and* Log.
- Map:
  <_ , (User, Log, id, age, userid, url)> → (url, (User, Log, id, age, userid, url))
  <_ , (Page, url, category)> → (url, (Page, url, category))
- Reduce:
  <url, {list of records}>
  → <_, {records from User x Log} x {records from Page}>
  // 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.
  - User 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.