BIG DATA

FRAMEWORK AND INFRASTRUCTURE

Adapted from the slides by Dr. Zareen Alamgir

Acknowledgement
Content obtained from many sources, including: Agrawal et al., VLDB 2010 tutorial; Shim, VLDB 2012 tutorial; Jeff Ullman, Jimmy Lin's notes

Books: Data-Intensive Text Processing with MapReduce and Mining of Massive Data Sets

Big Data: What is the Big deal?

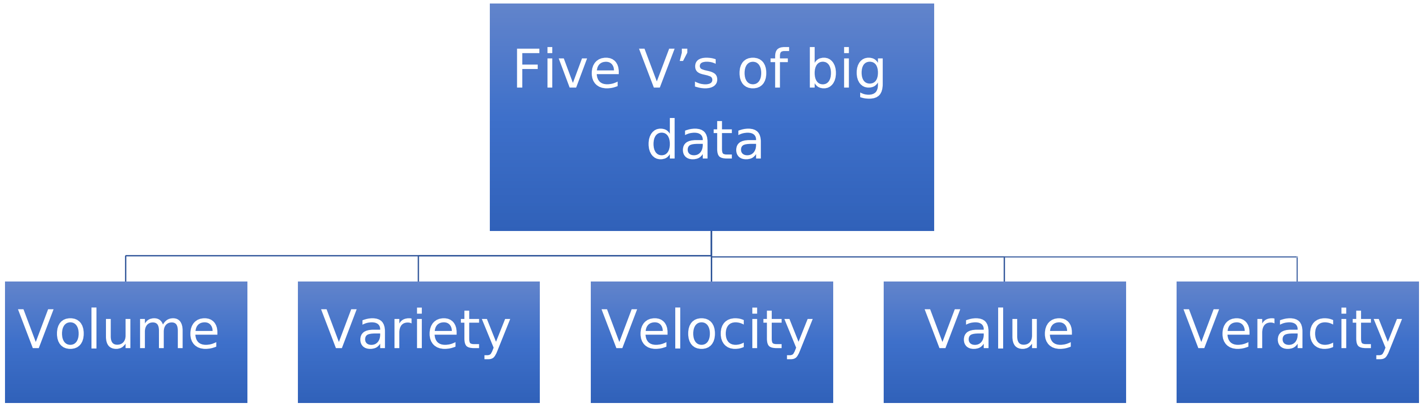
- Many success stories
  - *Google: many billions of pages indexed, products, structured data*
  - *Facebook: 1.1 billion users using the site each month*
  - *Twitter: 517 million accounts, 250 million tweets/day*
- This is changing society

Data Volumes are Exploding

Data is currency: companies are profiting from knowledge from Big Data

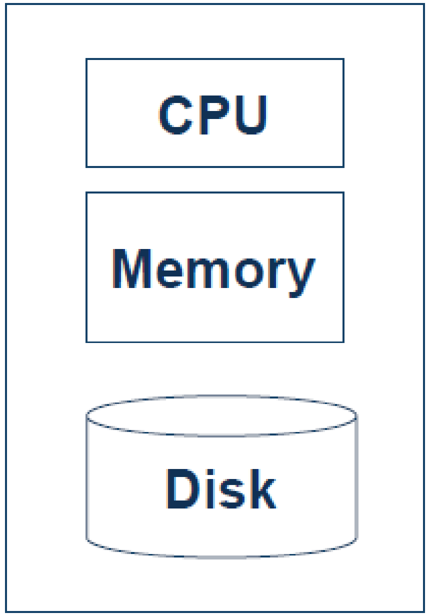
What is Massive/Big Data?

- *Too big*: petabyte-scale collections or lots of (not necessarily big) data sets
- *Too hard*: does not fit neatly in an existing tool
  - *Data sets that need to be cleaned, processed and integrated*
  - *E.g., Twitter, news, customer transactions*
- *Too fast*: needs to be processed quickly



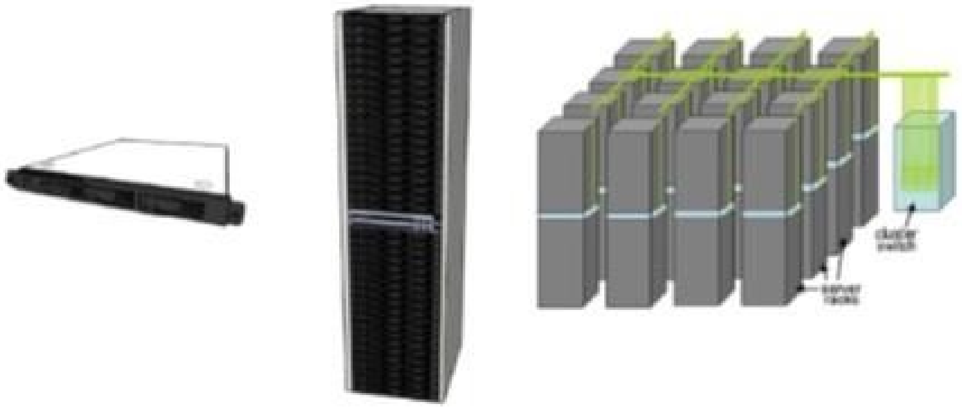
Single-node architecture

- Data Analysis
- Data Mining
- Machine Learning



Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - *~4 months to read the web*
- ~1,000 hard drives to store the web
- Takes even more to do something useful with the data!
- A standard architecture for such problems is emerging
  - *Cluster of commodity Linux nodes*
  - *Commodity network (ethernet) to conne*

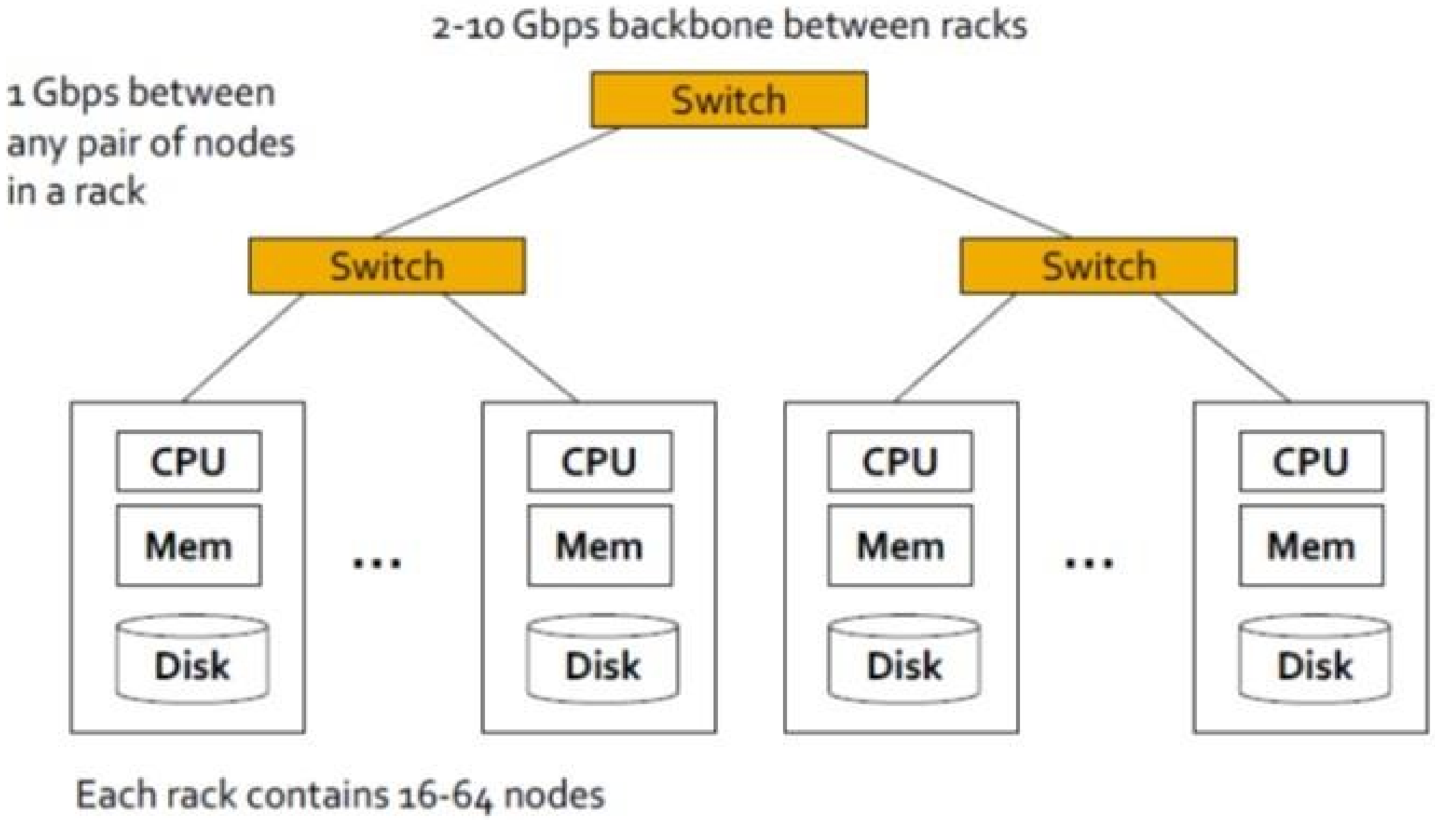


Platforms for Large-scale Data Analysis

- Parallel DBMS technologies
  - *Proposed in the late eighties*
  - *Matured over the last two decades*
  - *Multi-billion dollar industry: Proprietary DBMS Engines intended as Data Warehousing solutions for very large enterprises*

- Map Reduce
  - *pioneered by Google*
  - *popularized by Yahoo (open-source Hadoop)*

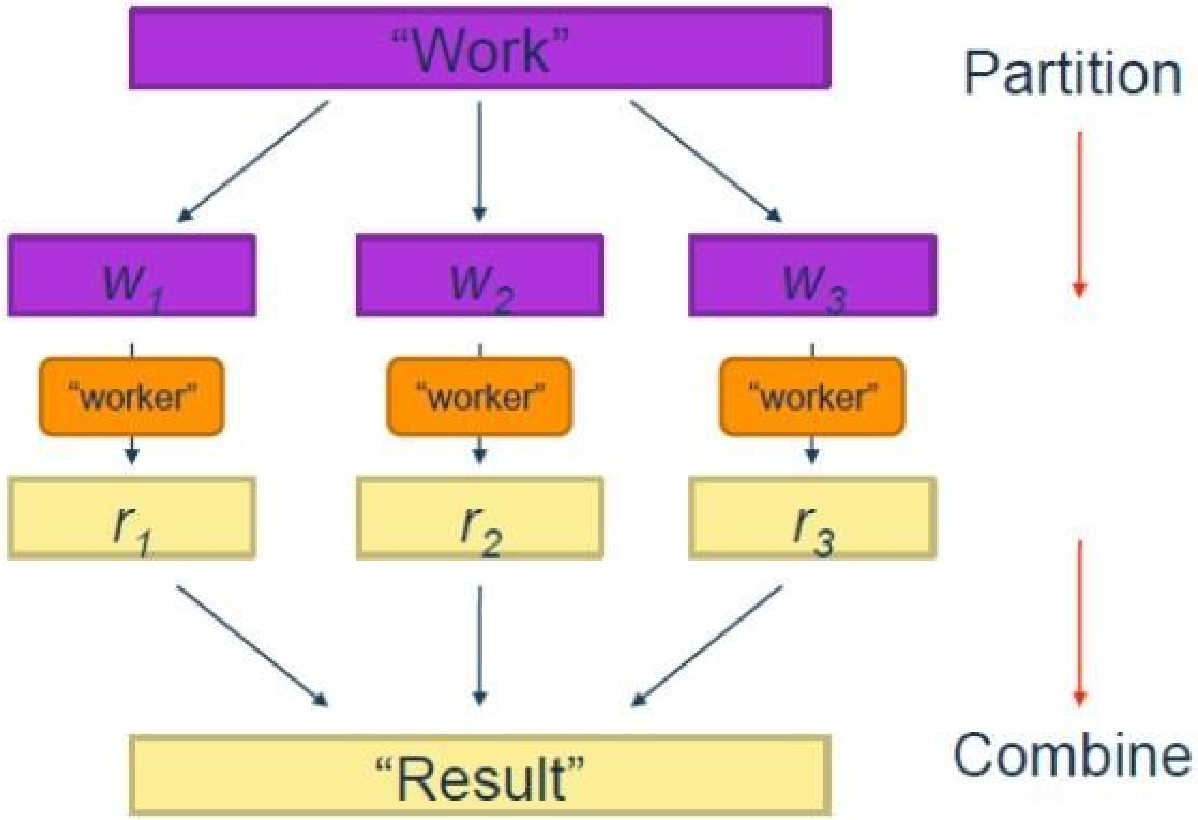
Cluster Architecture



In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RC

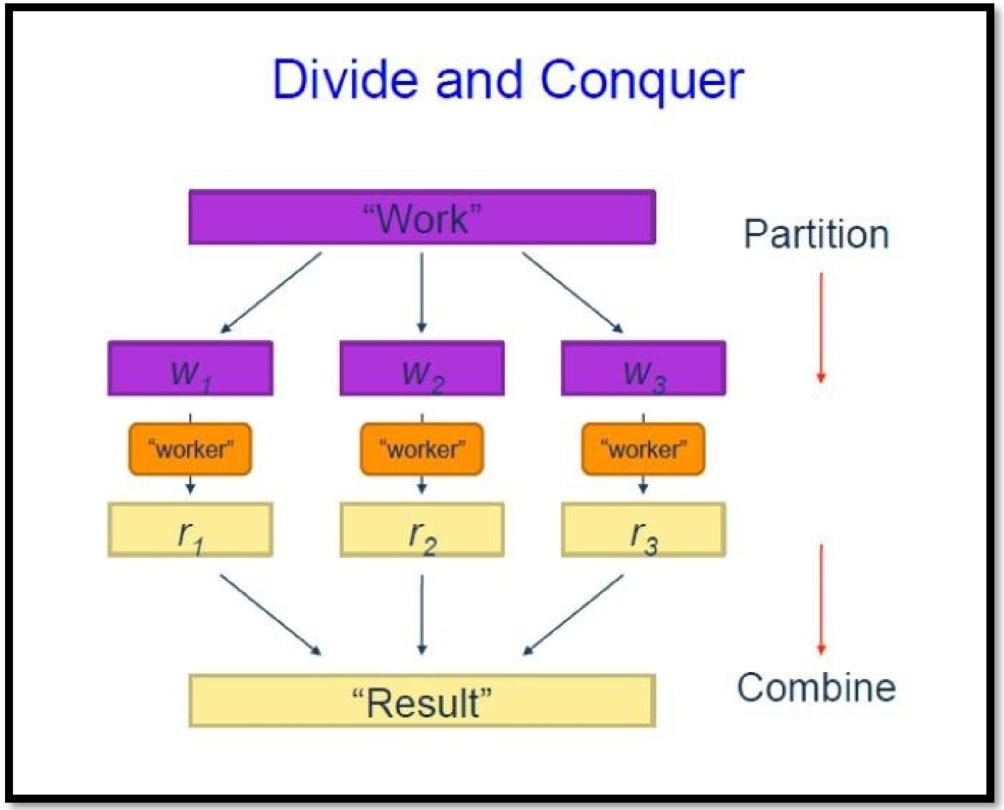

Tackling Big Data
Source: Google

Divide and Conquer



Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?
- What is the common theme of all of these problems



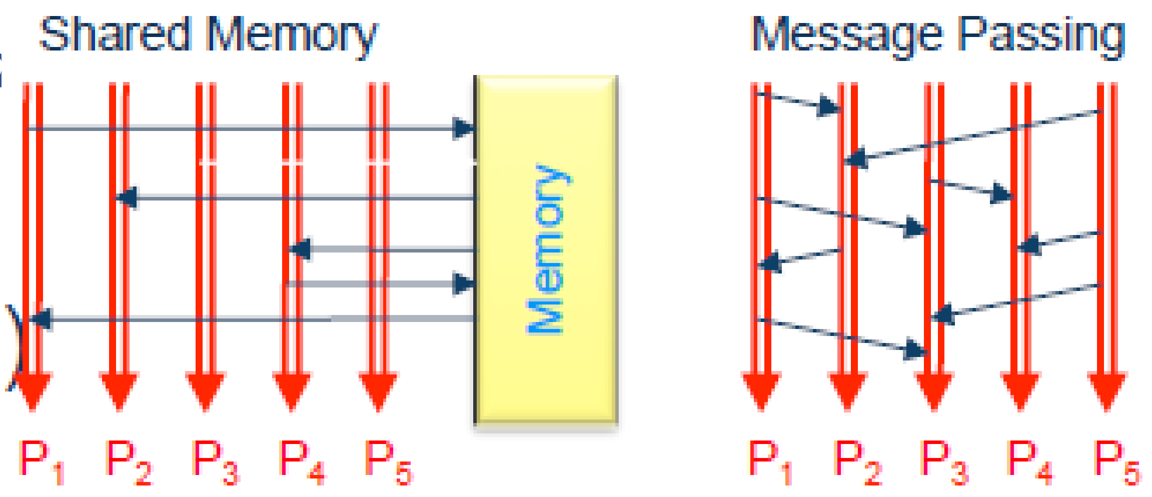
Common Theme?

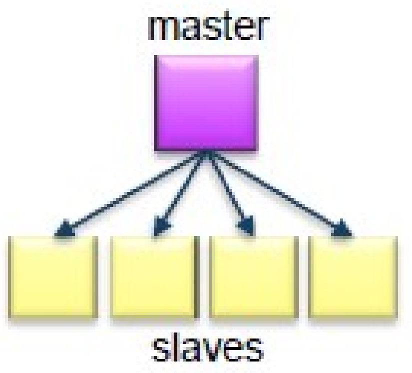
- Parallelization problems arise from:
  - *Communication between workers (e.g., to exchange state)*
  - *Access to shared resources (e.g., data)*
- Thus, we need a synchronization mechanism

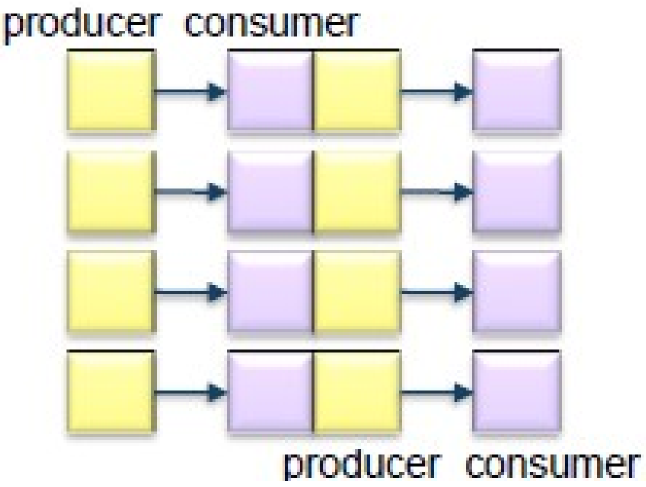
Semaphores (lock, unlock)
Conditional variables (wait, notify, broadcast)
Barriers
Still, lots of problems:
Deadlock, livelock, race conditions...
Dining philosophers, sleeping barbers, cigarette smokers...

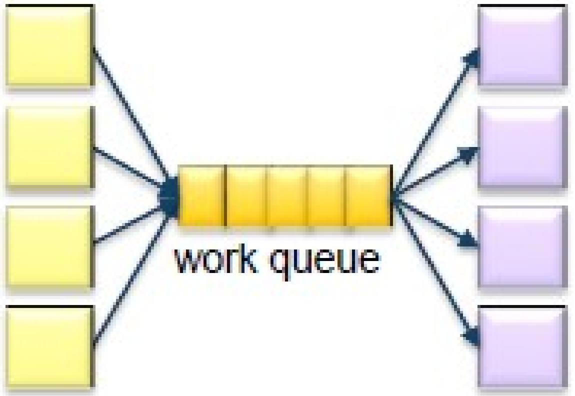
Current Tools

- Programming models
  - Shared memory (pthreac
  - Message passing (MPI)
- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues









Infrastructure for Big Data

- Data Centers
  - *Commodity hardware*
  - *Many machines connected in a netw*
- Challenges
  - *How to distribute computation?*
  - *Machines fail regularly*

- MapReduce is designed to handle these challenges

Big Ideas: Abstract System-Level Details

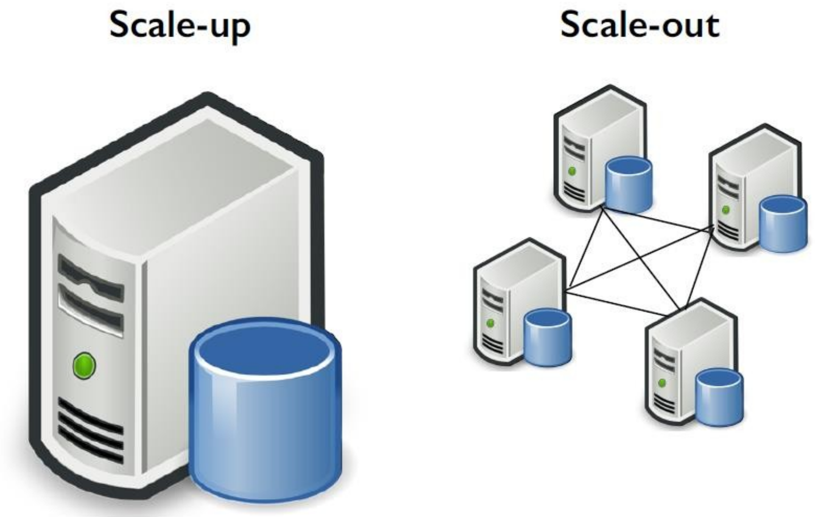
- It’s all about the right level of abstraction
  - *Moving beyond the von Neumann architecture*
  - *The datacenter is the computer !*

- MapReduce isolates developers from System level details
- Separating the what from the how
  - *Programmer defines what computations are to be performed*
  - *MapReduce execution framework takes care of how the computations are carried out*

Big Ideas: Scale Out vs. Scale Up

- Scale up: small number of high-end servers
  - *Symmetric multi-processing (SMP) machines, large shared memory*
  - *Not cost-effective – cost of machines does not scale linearly; and no single SMP machine is big enough*
- Scale out: Large number of commodity low-end servers is more effective for data-intensive applications

- 8 128-core machines vs. 128 8-core machines

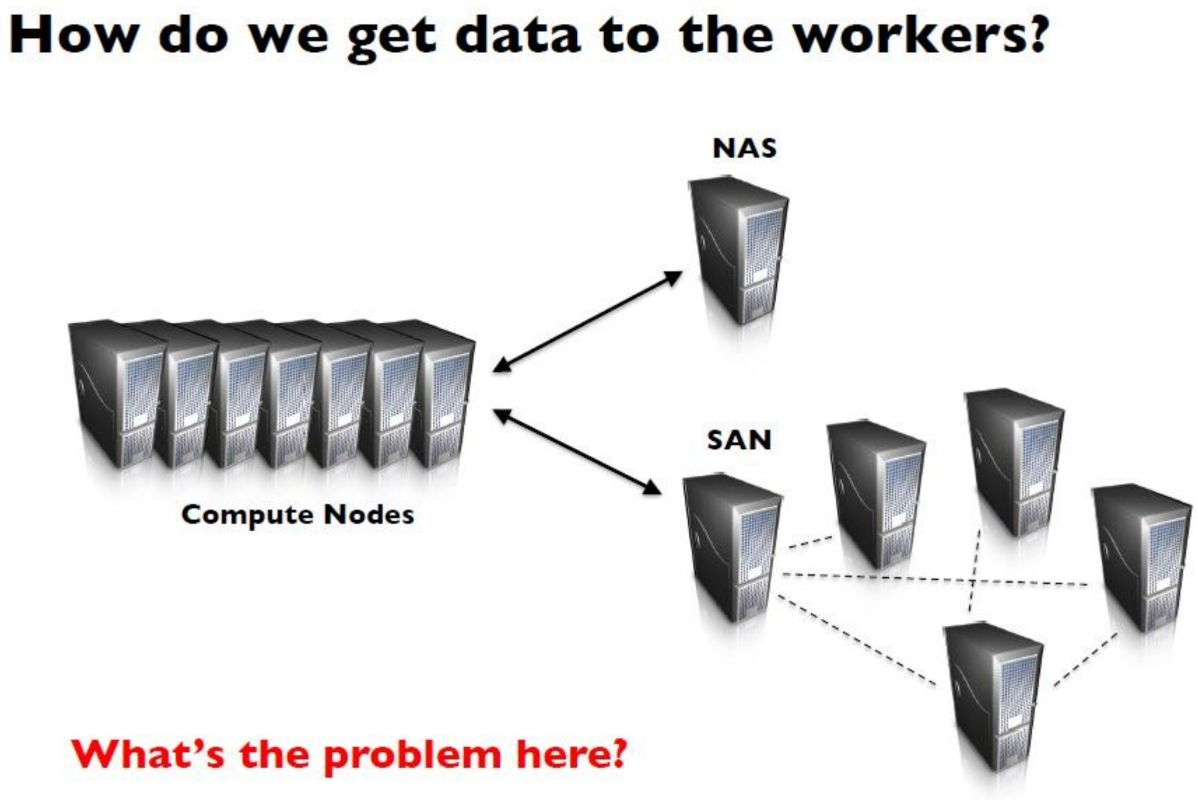


Big Ideas: Failures are Common

- Suppose a cluster is built using machines with a *mean-time between failures (MTBF) of 1000 days*
- For a 10,000 server cluster, there are on average 10 failures per day!
- MapReduce implementation cope with failures
  - *Automatic task restarts*

Big Ideas: Move Processing to Data

- Supercomputers often have processing nodes and storage nodes
  - *Computationally expensive tasks*
  - *High-capacity interconnect to move data around*
  - *Data movement leads to a bottleneck in the network!*



Why does this make sense for compute-intensive tasks?

What's the issue for data-intensive tasks?

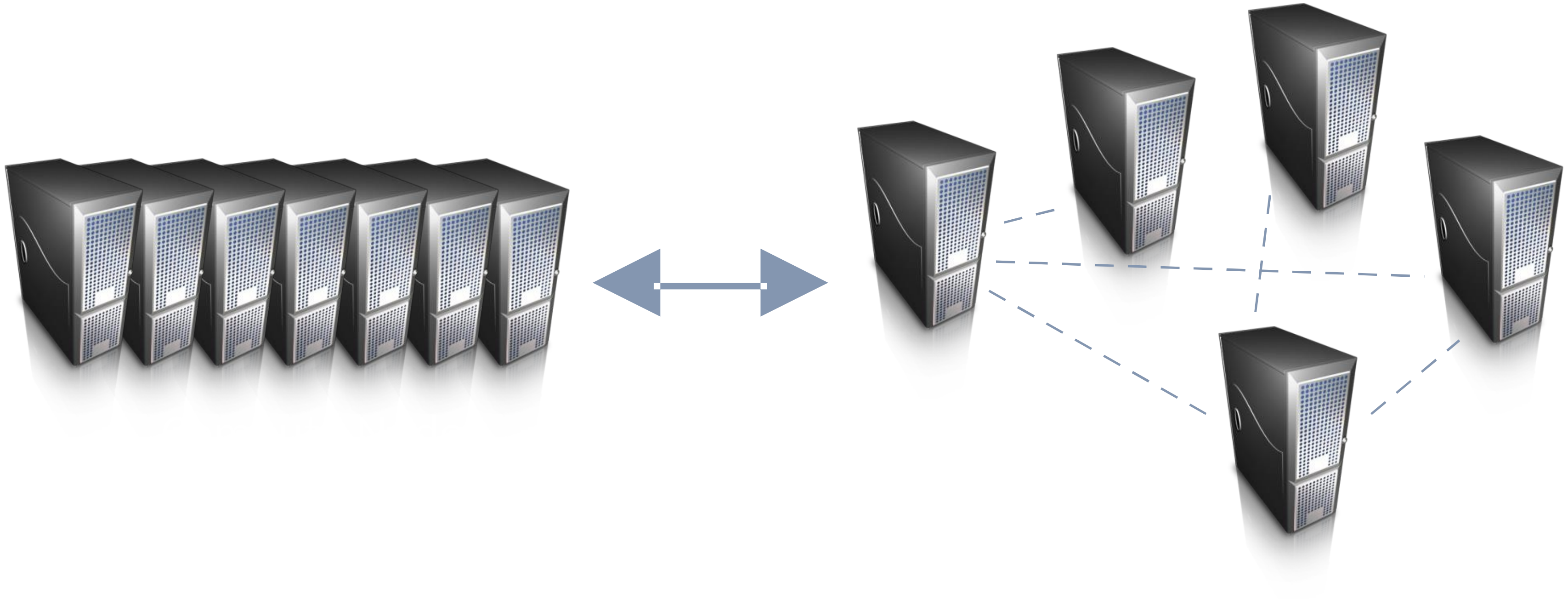
Many data-intensive applications are not very processor-demanding

What’s the solution?

Don’t move data to workers… move workers to the data!

Key idea: co-locate storage and compute

Start up worker on nodes that hold the data



What’s the solution?

Don’t move data to workers… move workers to the data!

Key idea: co-locate storage and compute

Start up worker on nodes that hold the data

We need a distributed file system for managing this

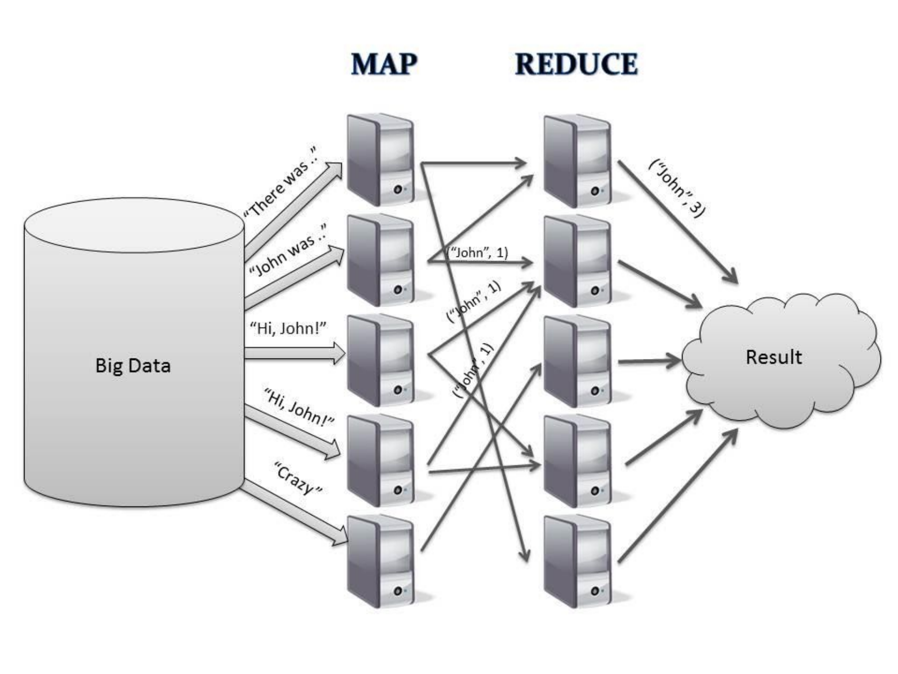
GFS (Google File System) for Google’s MapReduce

HDFS (Hadoop Distributed File System) for Hadoop

Big Ideas: Avoid Random Access

- Disk seek times are determined by mechanical factors
  - *Read heads can only move so fast and platters can only spin so rapidly*
- Example:
  - *1 TB database containing 1010 100 byte records*
  - *Random access: each update takes ~30ms (seek ,read, write)*
    - Updating 1% of the records takes ~35 days
  - *Sequential access: 100MB/s throughput*
    - Reading the whole database and rewriting all the records, takes 5.6 hours

MapReduce was designed for batch processing---
organize computations into long streaming operations

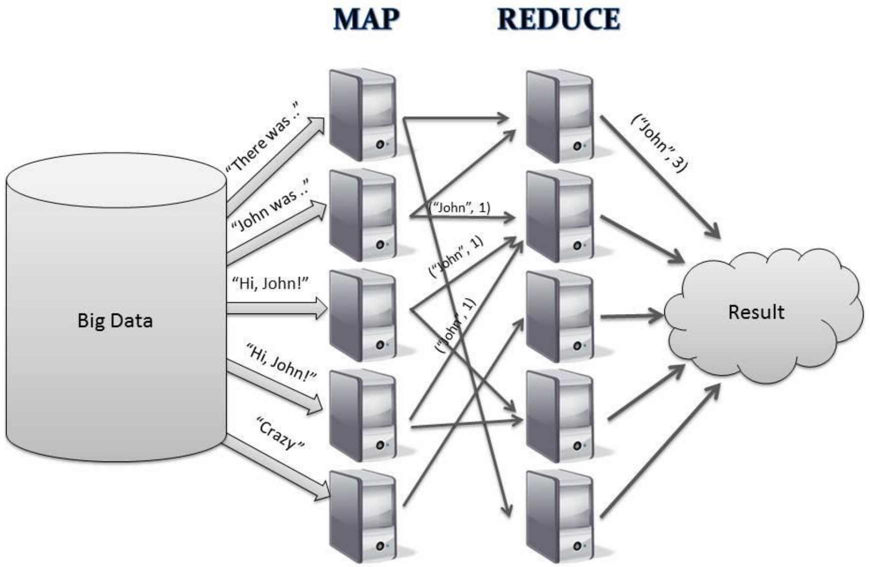


HADOOP AND MAP REDUCE

Map Reduce

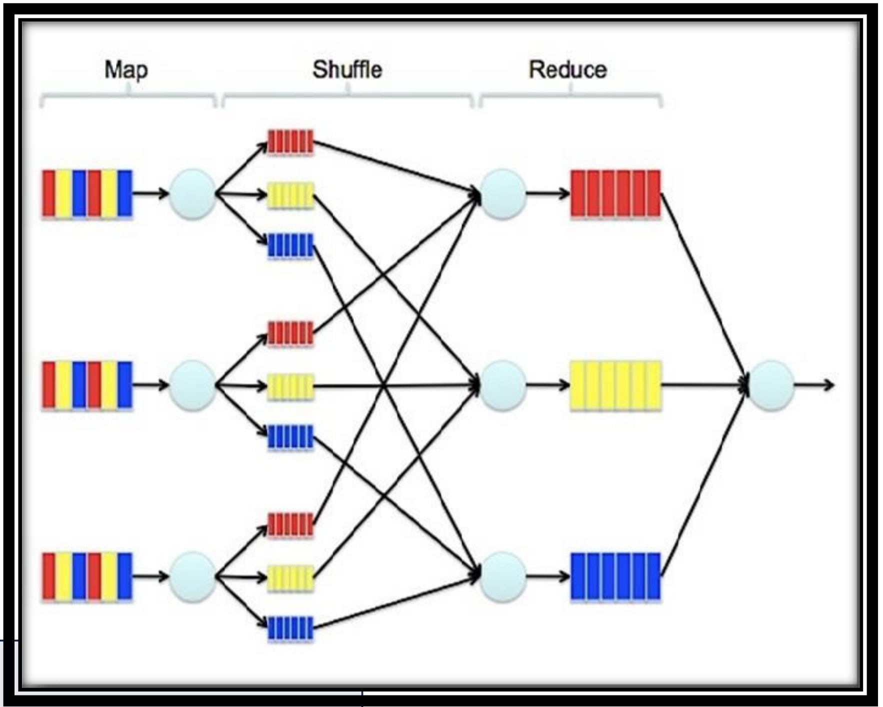
- Idea:
  - *Store data redundantly for reliability*
  - *Bring computation close to the data*
  - *Provide unified programming model to simplify parallelism*

- Builds on Distributed File Systems



Typical Large-Data Problem

- Iterate over a large number of records
- Extract something of interest from each Map
- Shuffle and sort intermediate results
- Aggregate intermediate results Reduce
- Generate final output



Programmers specify two functions:

map (k, v) → <k', v'>*

reduce (k', v') → <k', v'>*

The execution framework handles everything else…

(Dean and Ghemawat, OSDI 2004)

MapReduce - Word Count

Warm-up task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- Sample application:
  - *Analyze web server logs to find popular URLs*

Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
      emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```

MapReduce Example - WordCount

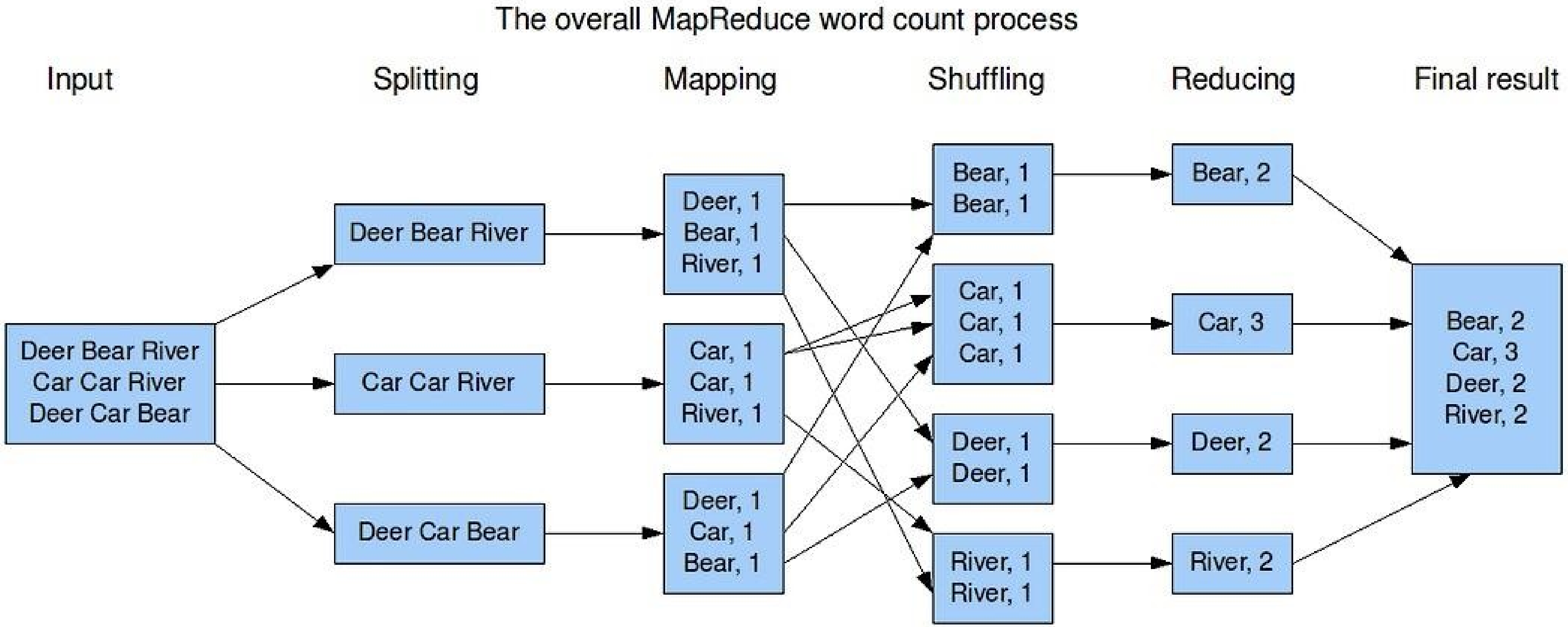
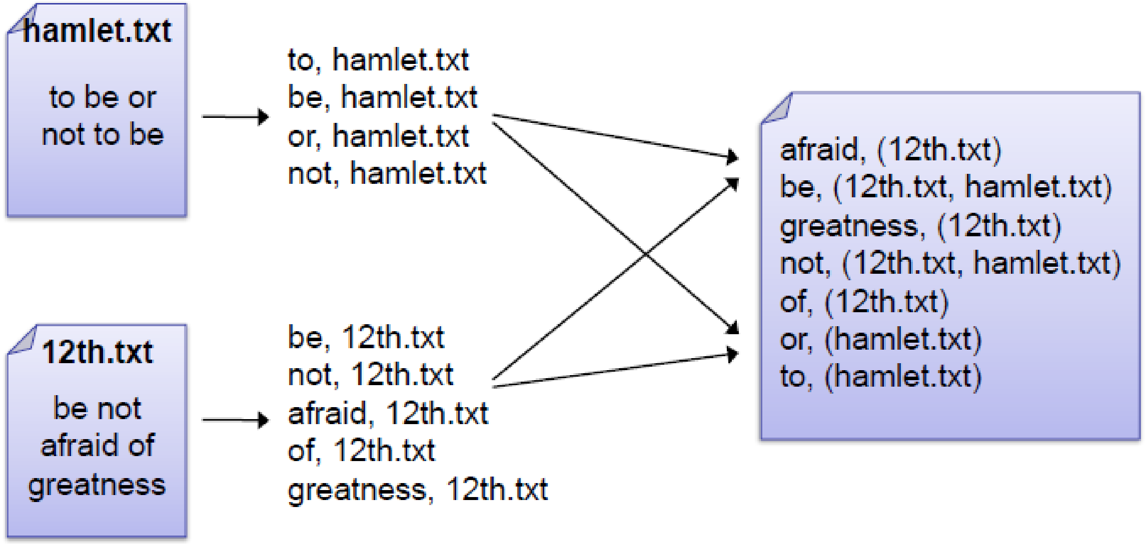


Image from: http://blog.jteam.nl/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png

Inverted Index Example

- This was the original Google's usecase
- Generate an inverted index of words from a given set of files

- Map:
  - parses a document and emits <word, docId> pairs
- Reduce:
  - takes all pairs for a given word, sorts the docId values, and emits a <word,list(docId)> pair



MapReduce Implementations

- Google has a proprietary implementation in C++
  - *Bindings in Java, Python*
- Hadoop is an open-source implementation in Java
  - *Development led by Yahoo, used in production*
  - *Now an Apache project*
  - *Rapidly expanding software ecosystem*
- Lots of custom research implementations
  - *For GPUs, cell processors, etc.*

Map and Reduce

- The idea of Map, and Reduce is 40+ year old
  - *Present in all Functional Programming Languages.*
  - *See, e.g., APL, Lisp and ML*
- Alternate names for Map: Apply-All
- Higher Order Functions
  - *take function definitions as arguments, or*
  - *return a function as output*
  - *Map and Reduce are higher-order functions.*

Map Examples in Haskell

- map (+1) [1,2,3,4,5]

  == [2, 3, 4, 5, 6]
- map (toLower) "abcDEFG12!@#“

  == "abcdefg12!@#“
- map (`mod` 3) [1..10]

  == [1, 2, 0, 1, 2, 0, 1, 2, 0, 1]

Fold-Right in Haskell

- Examples
  - *foldr (+) 0 [1..5] ==15*
  - *foldr (+) 10 [1..5] == 25*
- Definition
  - *foldr f z [] = z*
  - *foldr f z (x:xs) = f x (foldr f z xs)*

- Typically, map and fold are used in combination.
  - *Example: compute the sum of squares of a list of integers*

MapReduce

- Programmers specify two functions:
  - *map (k, v) → <k’, v’>**
  - *reduce (k’, v’) → <k’, v’>**
  - *All values with the same key are sent to the same reducer*

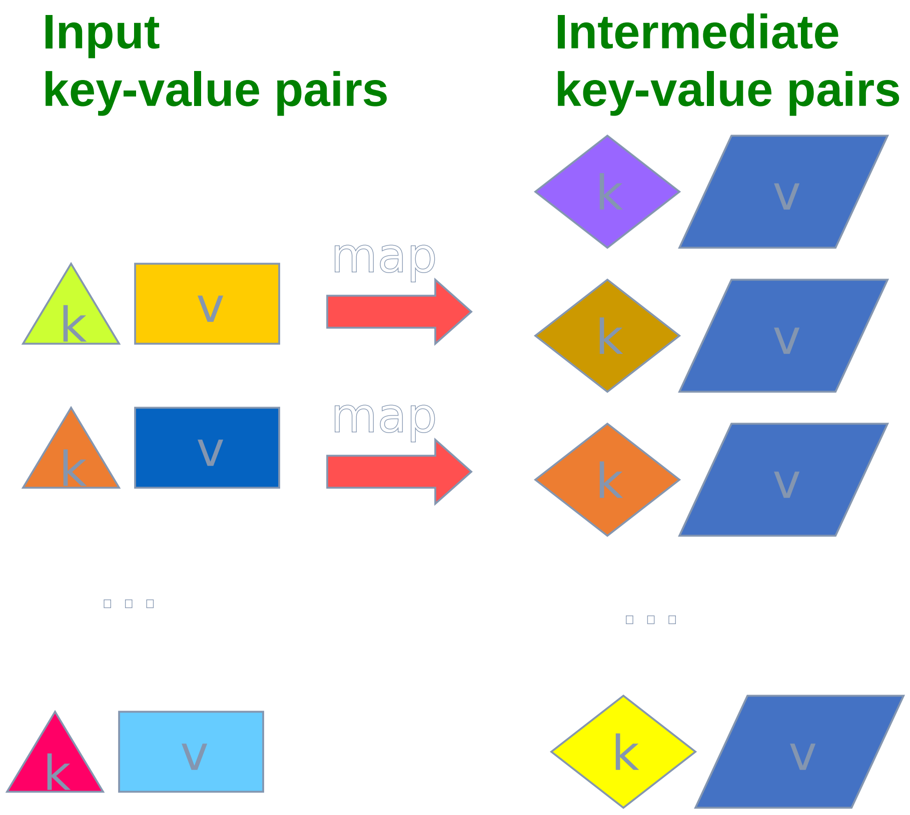
- The execution framework handles everything else…

MapReduce: Overview

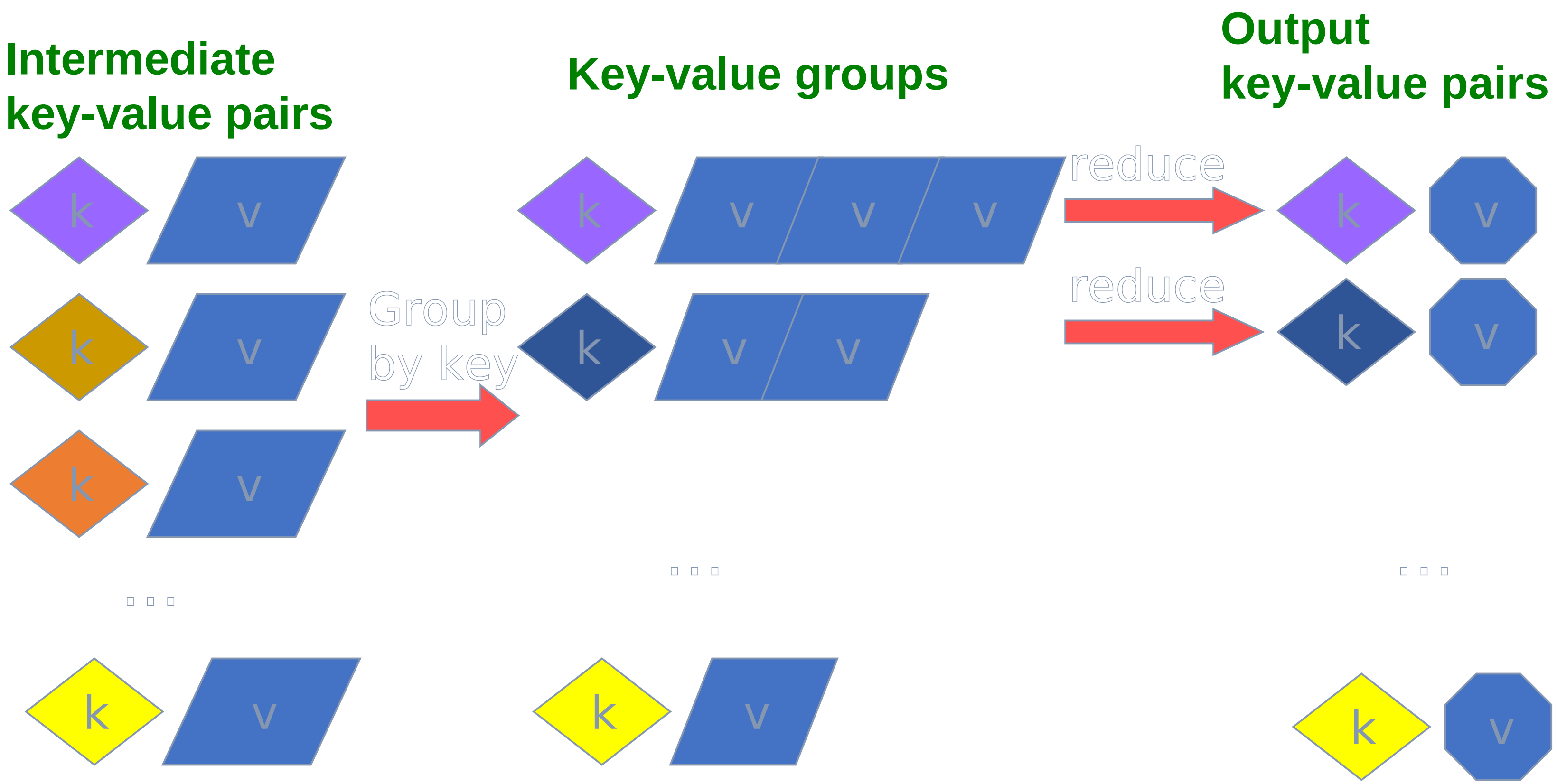
- Sequentially read a lot of data
- Map:
  - *Extract something you care about*
- Group by key: Sort and Shuffle
- Reduce:
  - *Aggregate, summarize, filter or transform*
- Write the result

Outline stays the same, Map and Reduce change to fit the problem

MapReduce: The Map Step



MapReduce: The Reduce Step



Example: Language modeling

- Statistical machine translation:
  - *Need to count number of times every 5-word sequence occurs in a large corpus of documents*

- How to implement in MapReduce:
  - *Map: extract (5-word sequence, count) from document*
  - *Reduce: combine counts*

Example: Distributed Grep

- Find all occurrences of the given pattern in a very large set of files.

- Map:
  - *Apply grep on assigned documents*
  - *Emit list of documents that contain term*
- Reduce:
  - *Merge lists*

Example: Shakemaps

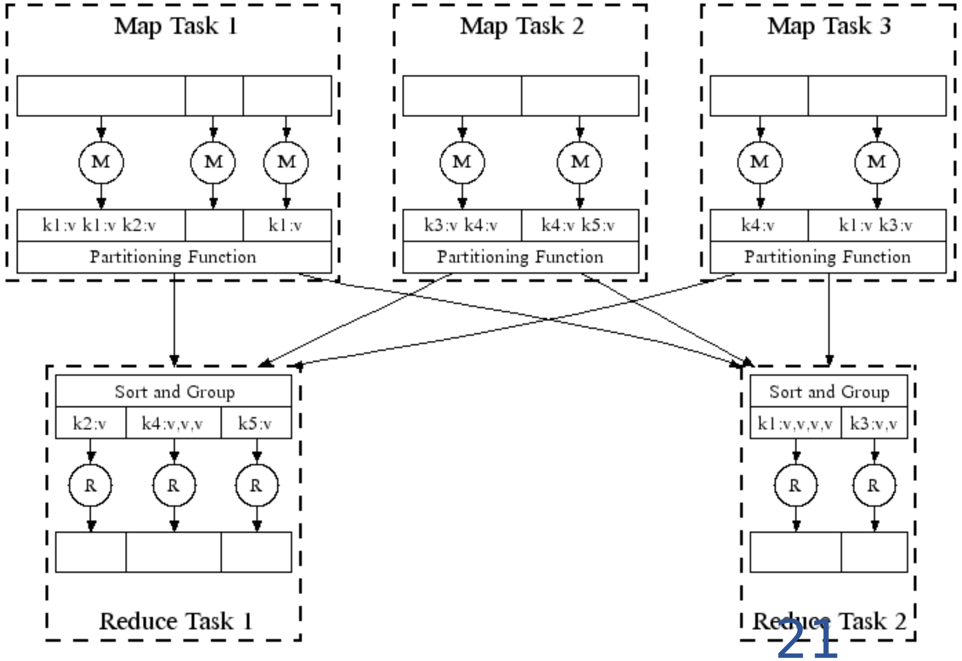
- Want to figure out how strongly different regions are shaken through earthquakes
- Input
  - *Each line: epicenter location; magnitude*
- Map
  - *Reads a line of input and simulate the earthquake*
  - *Output: (region ID, earthquake ID, amount of shaking)*
- Reduce
  - *Collect the region IDs and compute average (or maximum etc.) amount of shaking*

Refinement: Combiners

- A Map task may produce many pairs of the form *(k,v_1), (k,v_2), …* for the same key *k*
  - *E.g., popular words in the word count example*

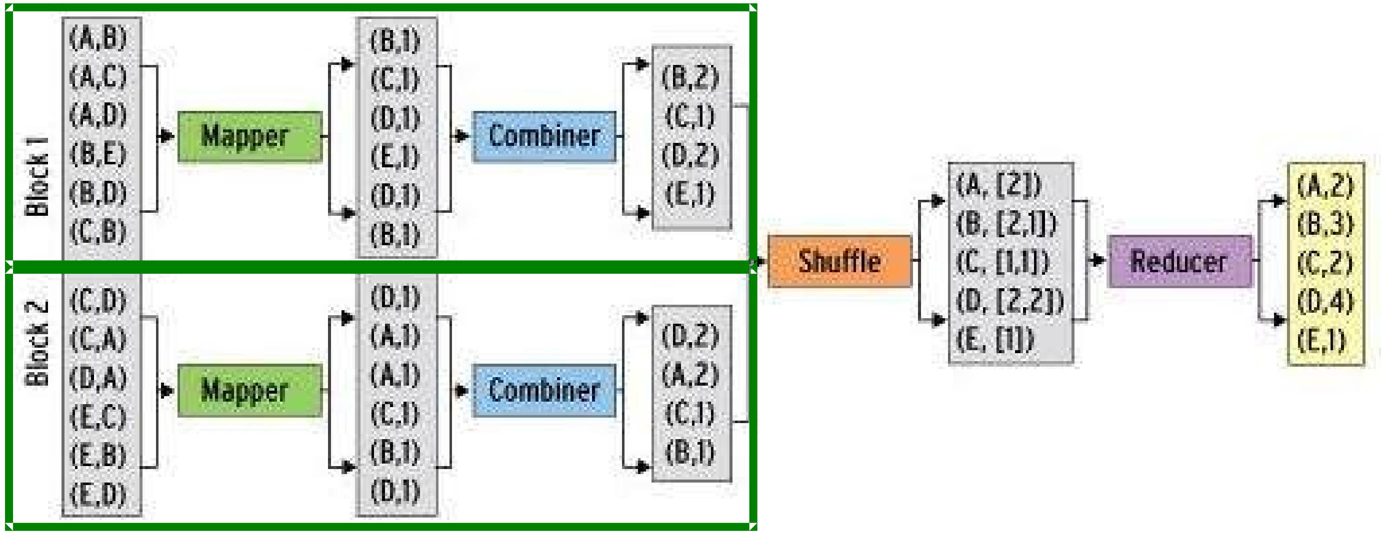
- Can save network time by pre-aggregating values in the mapper:
  - *combine(k, list(v_1))* $\rightarrow$ *v_2*
  - *Combiner is usually same as the reduce function*

- Works only if reduce function is commutative and associative



Refinement: Combiners

- Back to our word counting example:
  - *Combiner combines the values of all keys of a single mapper (single machine):*



  - *Much less data needs to be copied and shuffled!*

Combiner is usually same as the reduce function

Reading

- Chapter 1& 2: Data-Intensive Text Processing with Map Reduce