

# CS246: Mining Massive Datasets

## Intro & MapReduce

CS246: Mining Massive Datasets  
Jure Leskovec, Stanford University  
<http://cs246.stanford.edu>



# What is Data Mining?

## Knowledge discovery from data

---

**\$600** to buy a disk drive that can  
store all of the world's music

**5 billion** mobile phones  
in use in 2010

**30 billion** pieces of content shared  
on Facebook every month

**40%** projected growth in  
global data generated  
per year vs.

**5%**  
growth in global  
IT spending

**\$5 million vs. \$400**

Price of the fastest supercomputer in 1975<sup>1</sup>  
and an iPhone 4 with equal performance

**235** terabytes data collected by  
the US Library of Congress  
by April 2011

**15 out of 17**  
sectors in the United States have  
more data stored per company  
than the US Library of Congress



Data contains value and knowledge

# Data Mining

- But to extract the knowledge data needs to be
  - Stored
  - Managed
  - And **ANALYZED** ← this class

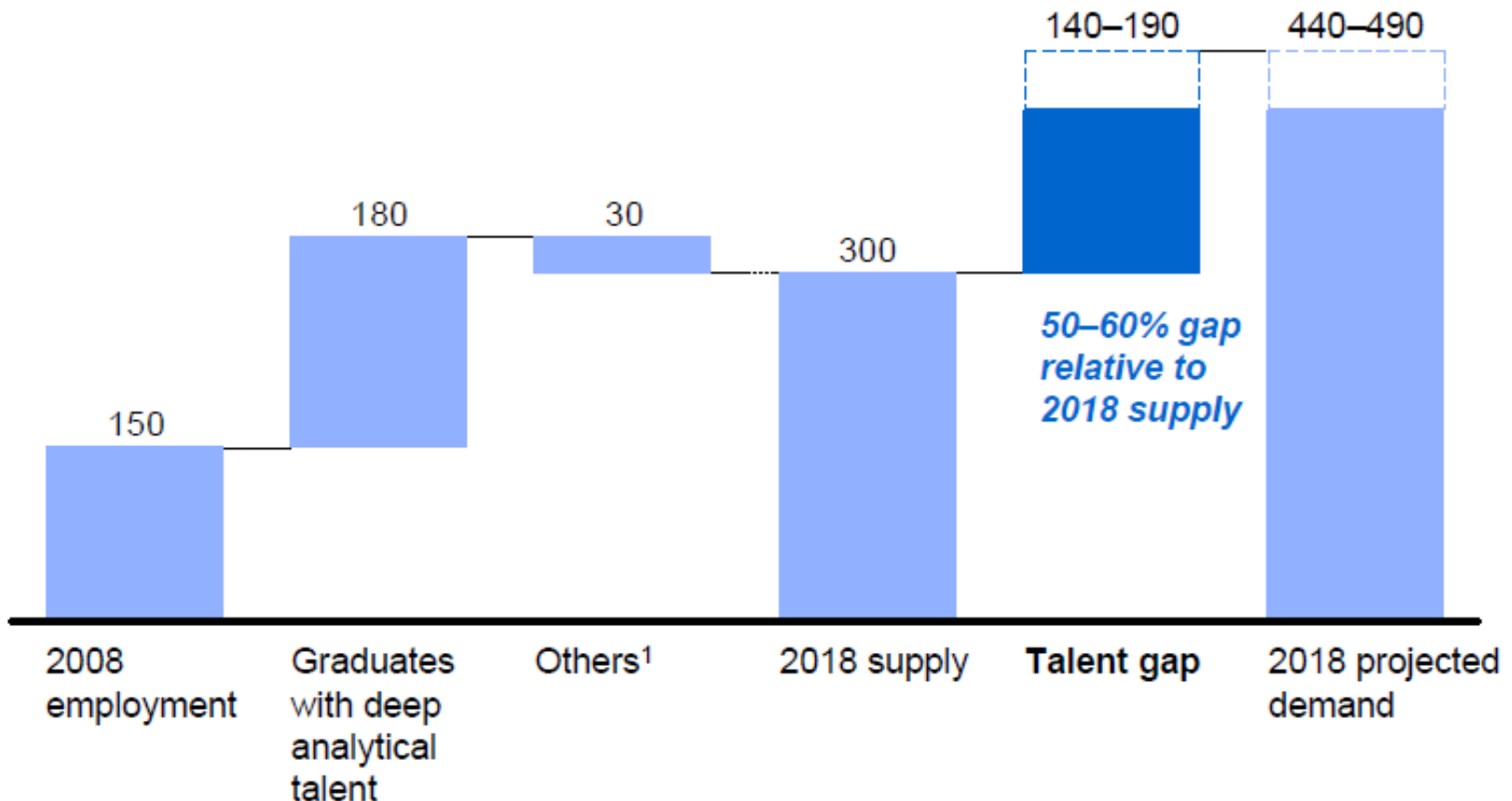
**Data Mining  $\approx$  Big Data  $\approx$   
Predictive Analytics  $\approx$  Data Science**

# Good news: Demand for Data Mining

**Demand for deep analytical talent in the United States could be 50 to 60 percent greater than its projected supply by 2018**

Supply and demand of deep analytical talent by 2018

Thousand people



<sup>1</sup> Other supply drivers include attrition (-), immigration (+), and reemploying previously unemployed deep analytical talent (+).

SOURCE: US Bureau of Labor Statistics; US Census; Dun & Bradstreet; company interviews; McKinsey Global Institute analysis

# What is Data Mining?

- Given lots of data
- Discover patterns and make predictions that are:
  - **Valid:** hold on new data with some certainty
  - **Useful:** should be possible to act on the item
  - **Unexpected:** non-obvious to the system
  - **Understandable:** humans should be able to interpret the pattern

# Data Mining Tasks

- **Descriptive methods**

- Find human-interpretable patterns that describe the data
  - **Example:** Clustering

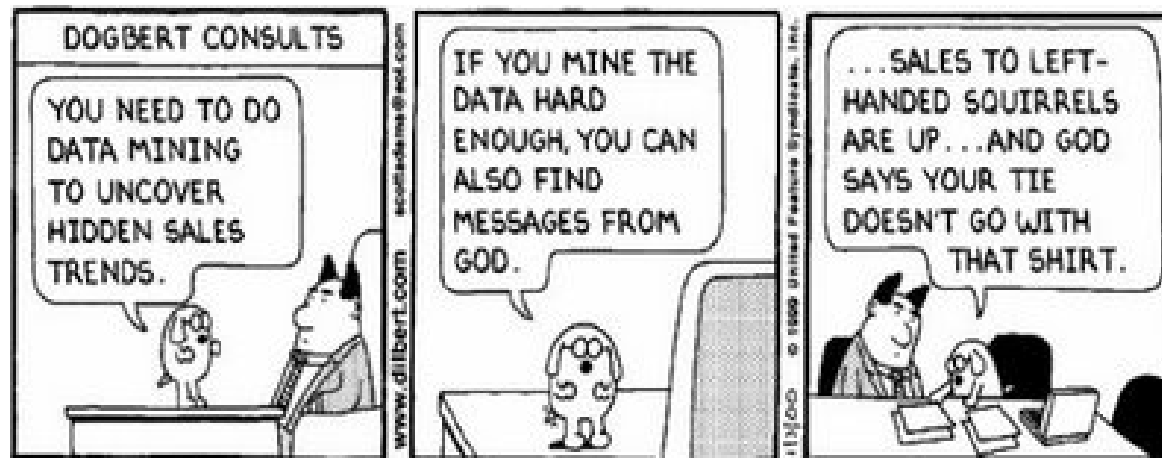
- **Predictive methods**

- Use some variables to predict unknown or future values of other variables
  - **Example:** Recommender systems

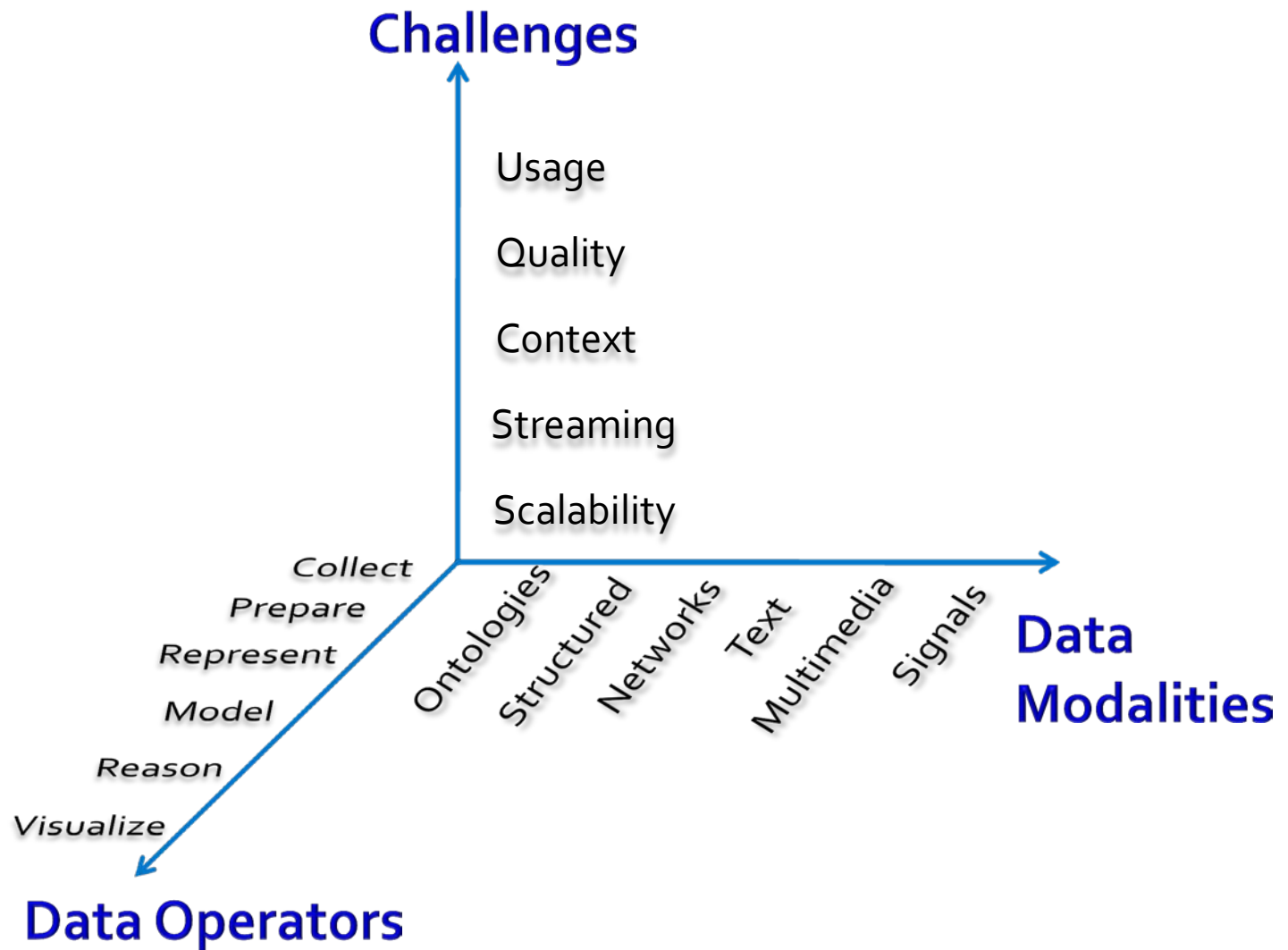


# Meaningfulness of Analytic Answers

- A risk with “Data mining” is that an analyst can “discover” patterns that are meaningless
- Statisticians call it **Bonferroni’s principle**:
  - Roughly, if you look in more places for interesting patterns than your amount of data will support, you are bound to find crap

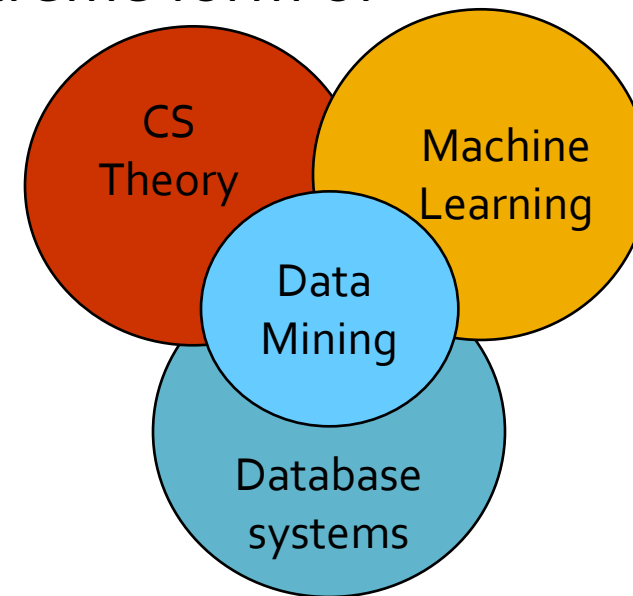


# What matters when dealing with data?



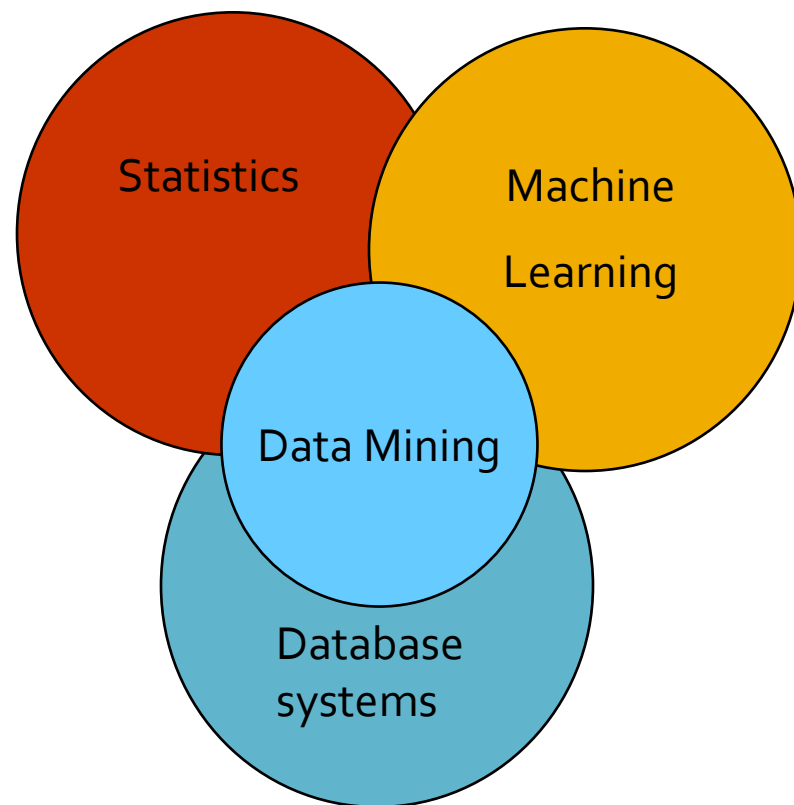
# Data Mining: Cultures

- **Data mining overlaps with:**
  - **Databases (DB):** Large-scale data, simple queries
  - **Machine Learning (ML):** Small data, Complex models
  - **CS Theory:** (Randomized) Algorithms
- **Different cultures:**
  - To a DB person, data mining is an extreme form of **analytic processing** – queries that examine large amounts of data
    - Result is the query answer
  - To a ML person, data-mining is the **inference of models**
    - Result is the parameters of the model
- **In this class we will do both!**



# This Class: CS246

- This class overlaps with machine learning, statistics, artificial intelligence, databases but more stress on
  - **Scalability** (big data)
  - **Algorithms**
  - **Computing architectures**
  - Automation for handling **large data**



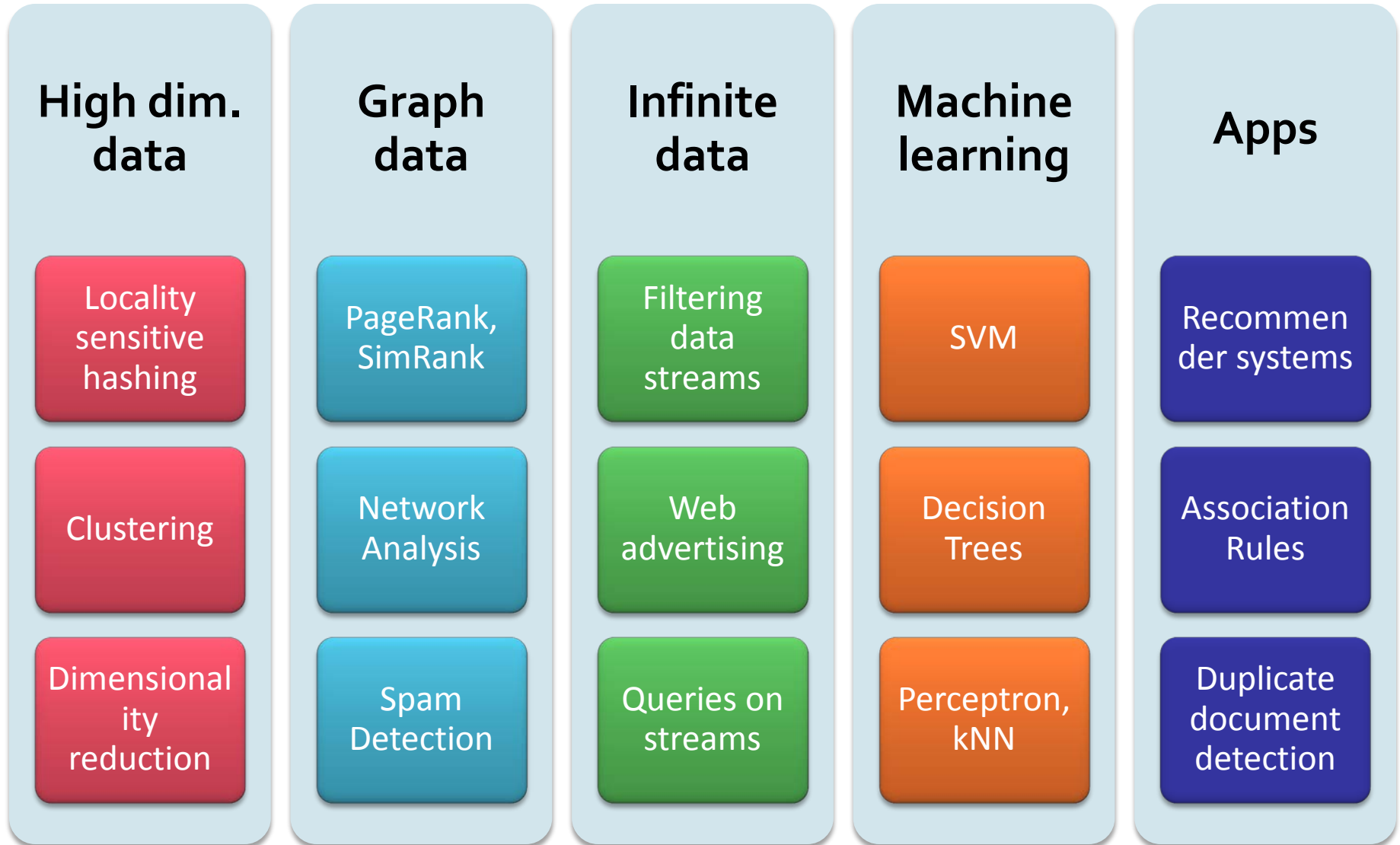
# What will we learn?

- We will learn to **mine different types of data:**
  - Data is high dimensional
  - Data is a graph
  - Data is infinite/never-ending
  - Data is labeled
- We will learn to **use different models of computation:**
  - MapReduce
  - Streams and online algorithms
  - Single machine in-memory

# What will we learn?

- **We will learn to solve real-world problems:**
  - Recommender systems
  - Market Basket Analysis
  - Spam detection
  - Duplicate document detection
- **We will learn various “tools”:**
  - Linear algebra (SVD, Rec. Sys., Communities)
  - Optimization (stochastic gradient descent)
  - Dynamic programming (frequent itemsets)
  - Hashing (LSH, Bloom filters)

# How the Class Fits Together





# How do you want that data?



# About the Course

# CS246 Course Staff

## ■ TAs:

### ■ We have 9 great TAs!

- Hristo Paskov, Clifford Huang, Dima Brezhnev, James Liu, Negar Rahmati, Clement Ntwari Nshuti, Jean-Yves Stephan, Janice Lan, Nat Roth

## ■ Office hours:

- **Jure:** Wednesdays 9-10am, Gates 418
- See course website for TA office hours
  - We start Office Hours next week
- For SCPD students we will use **Google Hangout**
  - We will post Hangout link on Piazza 1min before the OH

# Website and Readings

- **Course website:** <http://cs246.stanford.edu>
  - Lecture slides (at least 30min before the lecture)
  - Homeworks, solutions
  - Readings
- **Readings: Mining of Massive Datasets** by A. Rajaraman, J. Ullman, and J. Leskovec  
**Free online:** <http://mmds.org>  
Note there are **2** editions:
  - Use the **2<sup>nd</sup>** edition for class readings
  - Gradiance quizzes often refer to chapters of the **1<sup>st</sup>** edition

# Logistics: Communication

- **Piazza Q&A website:**

- <https://piazza.com/class#winter2015/cs246>

- Use Piazza for all questions and public communication
  - Search the forum before asking a question
  - Please tag your posts and please no one-liners
- Jure will give **extra credit** for Piazza participation

- **For e-mailing course staff always use:**

- [cs246-win1415-staff@lists.stanford.edu](mailto:cs246-win1415-staff@lists.stanford.edu)

- **We will post course announcements to Piazza (make sure you check it regularly)**

**Auditors are welcome to sit-in & audit the class**

# Work for the Course

- **4 longer homeworks: 40%**
  - Theoretical and programming questions
  - Hadoop tutorial has just been posted
  - Assignments take lots of time (+20h). **Start early!!**
- **How to submit?**
  - **Homework write-up:**
    - Submit via Scoryst
  - **Everyone uploads code:**
    - Put all the code for 1 question into 1 file and submit at: <http://snap.stanford.edu/submit/>

# Work for the Course

- **Short weekly Gradianc quizzes: 20%**
  - Quizzes are posted on **Fridays**
    - Due **exactly** 7 days later (Friday 23:59 PT) **No late days!**
    - First quiz has already been posted
  - To register on Gradianc please use **SUNetID** or **<legal first name>\_<legal last name>** for username
    - We have to be able to match your Gradianc ID and SUNetID
- **Final exam: 40%**
  - **Friday, March 20 12:15pm-3:15pm**
  - Alternate final: Monday, March 16 7-10pm

# Course Calendar

## ■ Homework schedule:

Date (5pm)	Out	In
01/01, Tue	Tutorial	
01/08, Thu	HW1	
01/13, Tue		Tutorial
01/22, Thu	HW2	HW1
02/05, Thu	HW3	HW2
02/19, Thu	HW4	HW3
03/05, Thu		HW4

- **2 late “days” (late periods) for HWs for the quarter:**
  - 1 late period expires on Tuesday 5pm
  - **You can use max 1 late period per assignment**

# Prerequisites

- **Algorithms** (CS161)
  - Dynamic programming, basic data structures
- **Probability** (CS109 or Stat116)
  - Moments, typical distributions, MLE, ...
- **Programming** (CS107 or CS145)
  - Your choice, but Java will be very useful
- **We provide some background at recitation sessions, but the class will be fast paced**



# Recitation Sessions

- **Recitation sessions:**
  - **Review of probability & stats:** Fri 1/9, at 4:15pm
  - **Review of linear algebra:** Fri 1/16, at 4:15pm
  - All sessions will be held in **Gates B01**
  - Sessions will be video recorded

# CS246H: Hadoop Labs

- **CS246H** covers practical aspects of Hadoop and other distributed computing architectures
  - HDFS, Combiners, Partitioners, Hive, Pig, Hbase, ...
  - 1 unit course, optional homeworks
- **CS246H runs (somewhat) parallel to CS246**
  - CS246 discusses theory and algorithms
  - CS246H tells you how to implement them
- **Instructor:** Daniel Templeton (Cloudera)
- First session of CS246H will give a Hadoop tutorial
  - Think of it as CS246 Hadoop recitation
- CS246H lectures are recorded (available via SCPD)

# What's after the class

- **Data Science / Info Seminar (CS545):**
  - <http://i.stanford.edu/infoseminar>
  - Great industrial & academic speakers
  - Topics include data mining, ML, data science
  - Open to SCPD students (videos recorded)
- **CS341: Project in Data Mining (Spring 2014)**
  - Research project on big data
  - Groups of 3 students
  - We provide interesting data, computing resources (Amazon EC2) and mentoring
- **My group has RA positions open:**
  - See <http://snap.stanford.edu/apply/>

# Final Thoughts

- **CS246 is fast paced!**
  - Requires programming maturity
  - Strong math skills
    - SCPD students tend to be rusty on math/theory
- **Course time commitment:**
  - Homeworks take about 20h
  - Gradiance quizzes take about 1-2h
- **Form study groups**
- **It's going to be fun and hard work. 😊**

# 4 To-do items

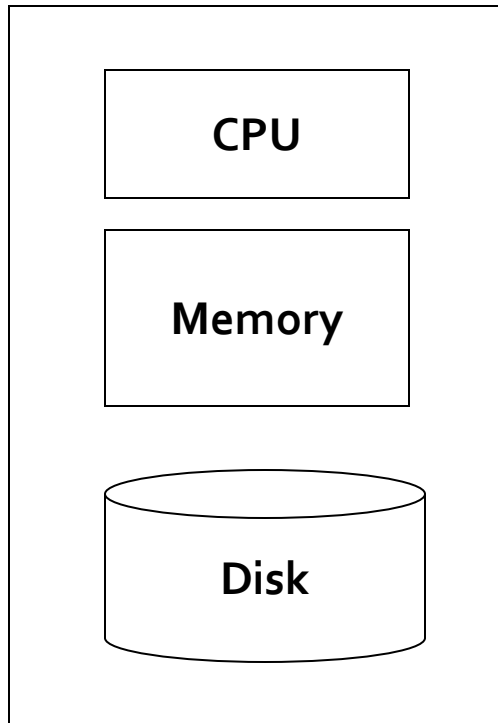
- **4 to-do items for you:**
  - **Register to Piazza**
  - **Register to Scoryst**
  - **Register to Gradiane and complete the first quiz**
    - **Use your SUNet ID to register!** (so we can match grading records)
    - Complete the first quiz (it is already posted)
  - **Complete Hadoop tutorial**
    - Tutorial should take you about 2 hours to complete  
(Note this is a “toy” homework to get you started. Real homeworks will be much more challenging and longer)
- **Additional details/instructions at**  
**<http://cs246.stanford.edu>**

# Computational Model: MapReduce

# MapReduce

- Much of the course will be devoted to **large scale computing for data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard
- **MapReduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

# Single Node Architecture



**Machine Learning, Statistics**

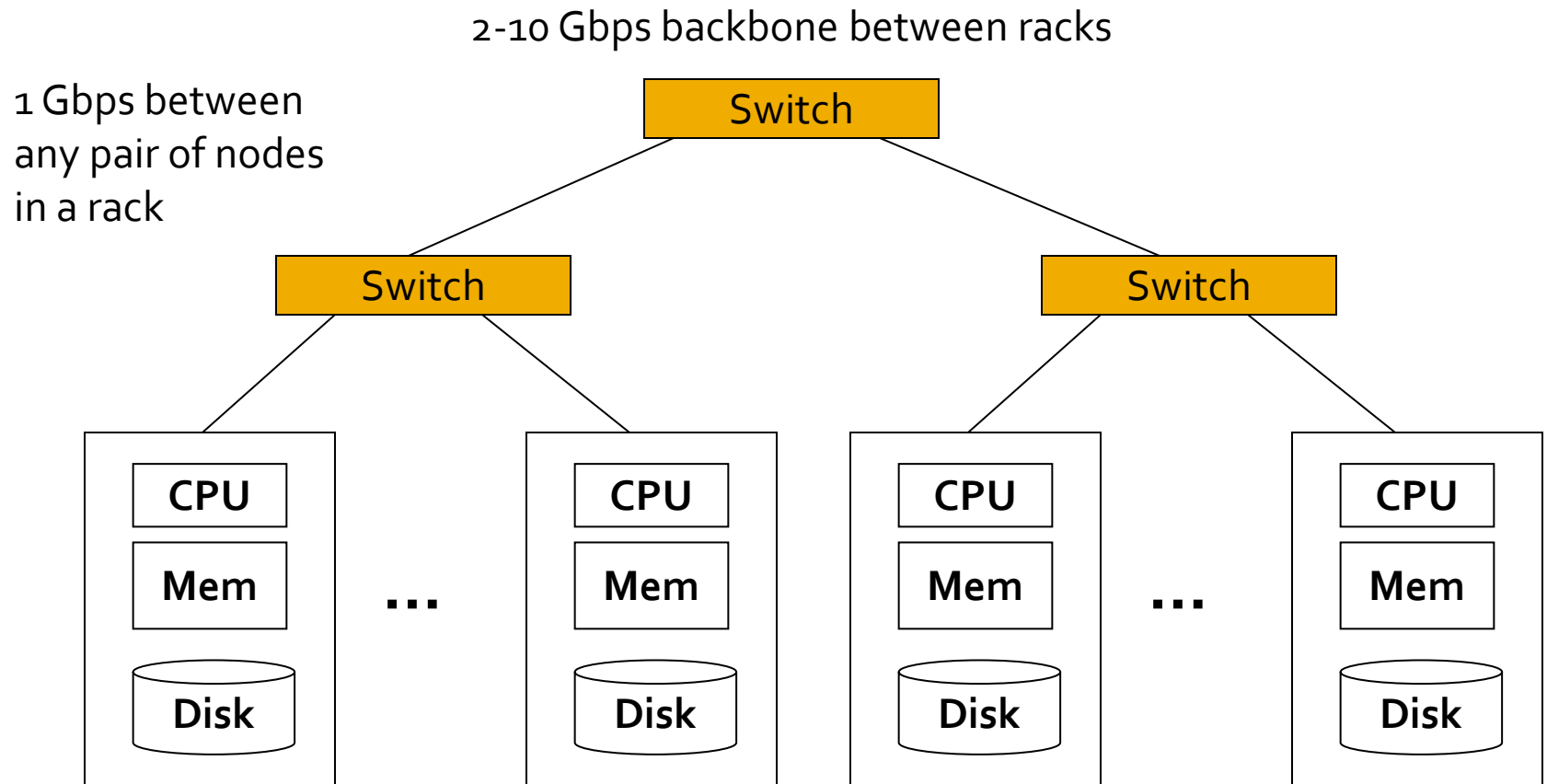
**“Classical” Data Mining**



# 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!**
- **Recently standard architecture for such problems emerged:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

# Cluster Architecture



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, <http://bit.ly/Shh0RO>



# Large-scale Computing

- **Large-scale computing for data mining problems on commodity hardware**
- **Challenges:**
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - With 1M machines 1,000 machines fail every day!

# Idea and Solution

- **Issue:** Copying data over a network takes time
- **Idea:**
  - Bring computation to data
  - Store files multiple times for reliability
- **MapReduce** addresses these problems
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - MapReduce



# Storage Infrastructure

- **Problem:**

- If nodes fail, how to store data persistently?

- **Answer:**

- **Distributed File System**

- Provides global file namespace

- **Typical usage pattern:**

- Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

# Distributed File System

## ■ **Chunk servers**

- File is split into contiguous chunks
- Typically each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

## ■ **Master node**

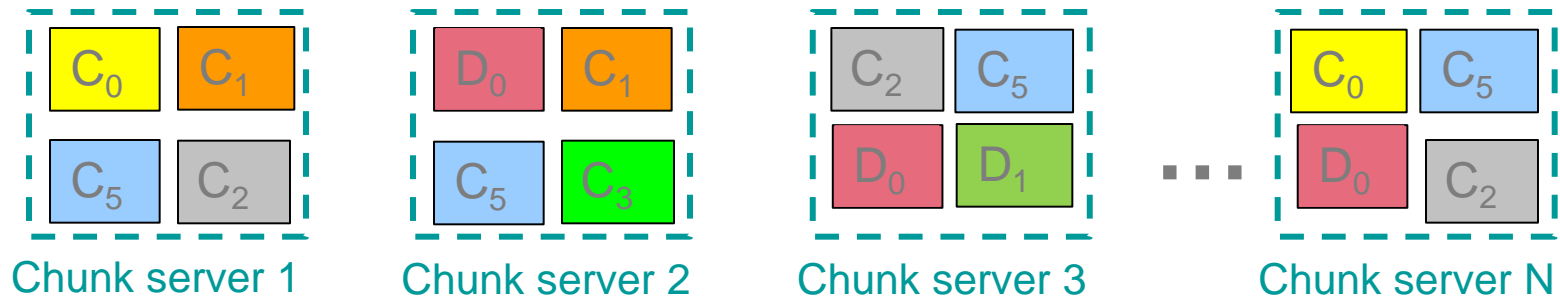
- a.k.a. Name Node in Hadoop's HDFS
- Stores metadata about where files are stored
- Might be replicated

## ■ **Client library for file access**

- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

# Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk **replicated** on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Chunk servers also serve as compute servers



# Programming Model: MapReduce

## 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

# Task: Word Count

## Case 1:

- File too large for memory, but all `<word, count>` pairs fit in memory

## Case 2:

- Count occurrences of words:
  - `words(doc.txt) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one word per a line
- **Case 2** captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

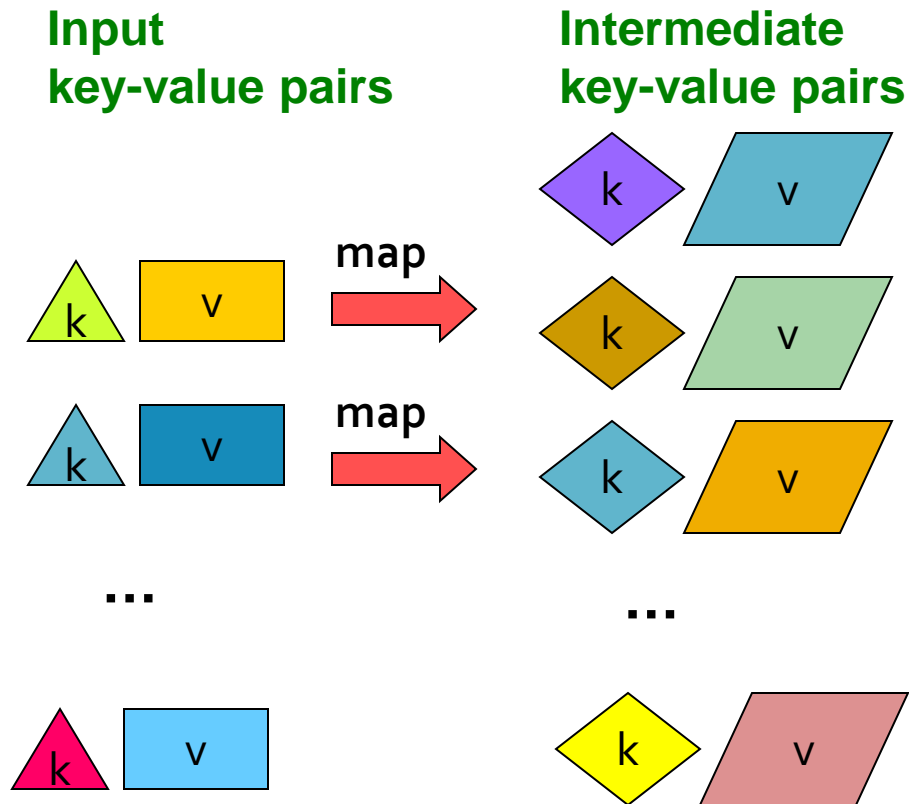
# MapReduce: Overview

## 3 steps of MapReduce

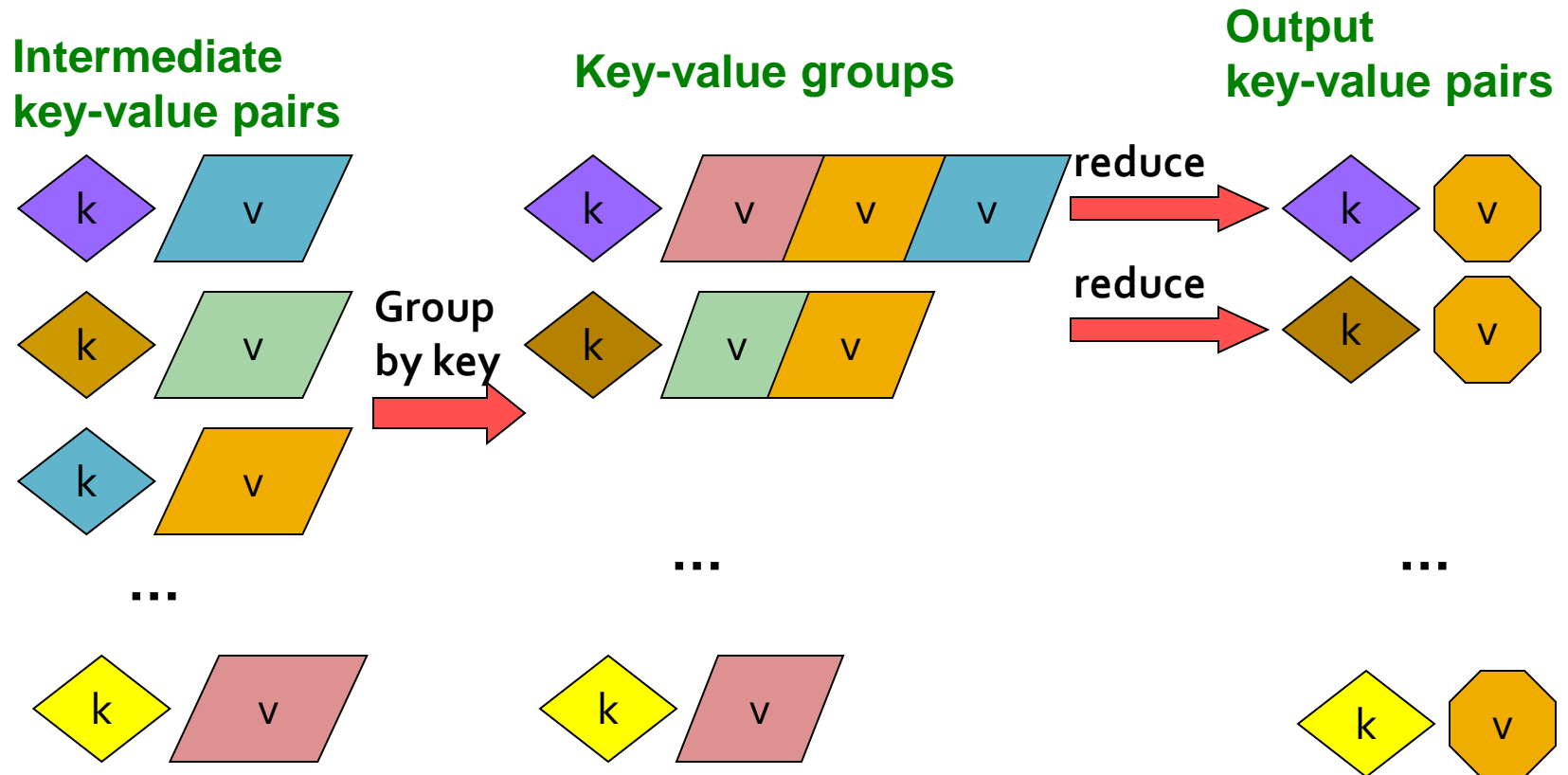
- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and shuffle
- **Reduce:**
  - Aggregate, summarize, filter, or transform
- Output the result

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

# MapReduce: The Map Step



# MapReduce: The Reduce Step



# More Specifically

- **Input:** a set of key-value pairs
- **Programmer specifies two methods:**
  - **Map( $k, v$ )**  $\rightarrow \langle k', v' \rangle^*$ 
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every  $(k, v)$  pair
  - **Reduce( $k', \langle v' \rangle^*$ )**  $\rightarrow \langle k', v'' \rangle^*$ 
    - All values  $v'$  with same key  $k'$  are **reduced** together and processed in  $v'$  order
    - There is one Reduce function call per unique key  $k'$

# Map-Reduce: A diagram

## MAP:

Read input and produces a set of key-value pairs

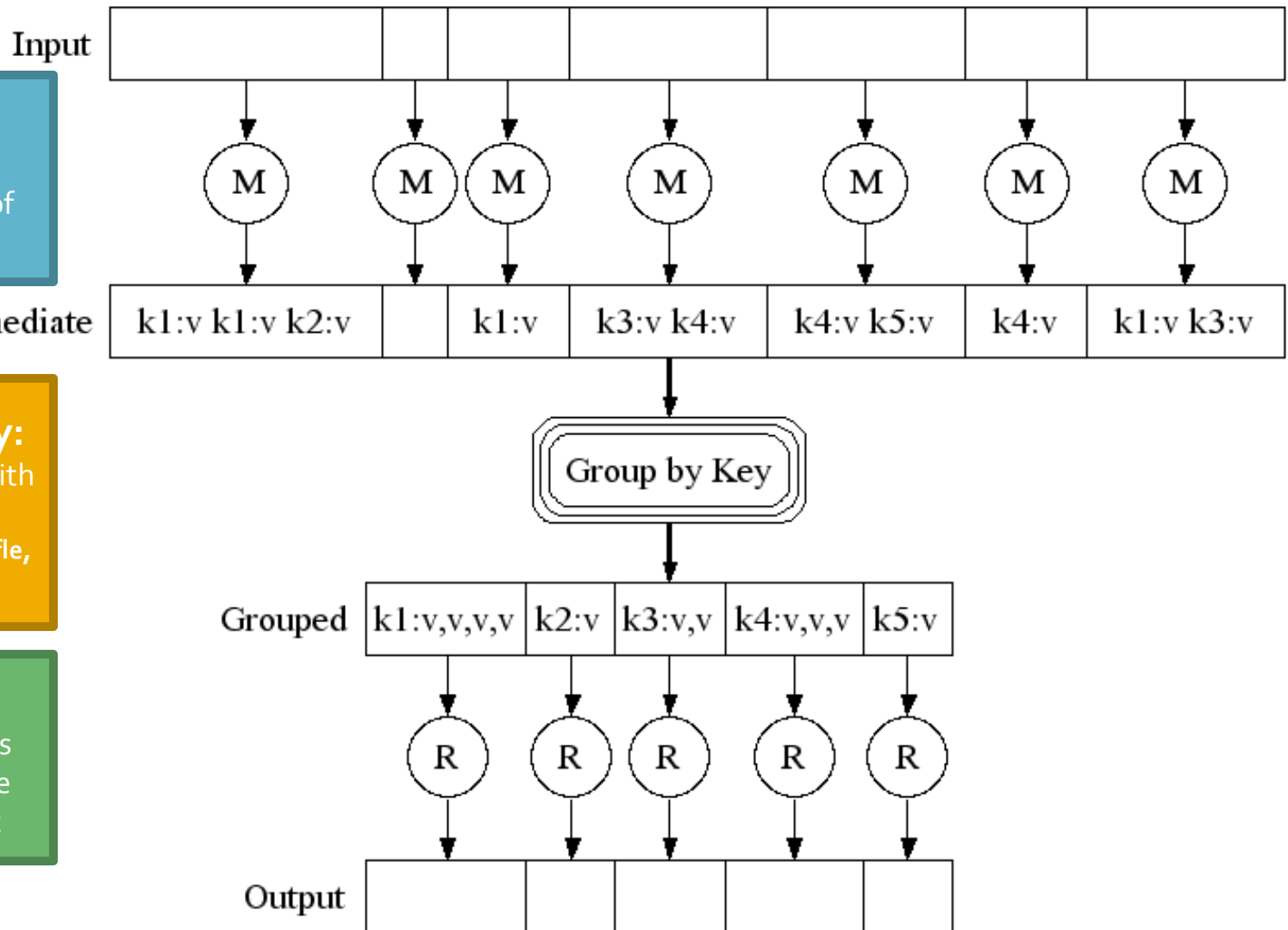
Intermediate

## Group by key:

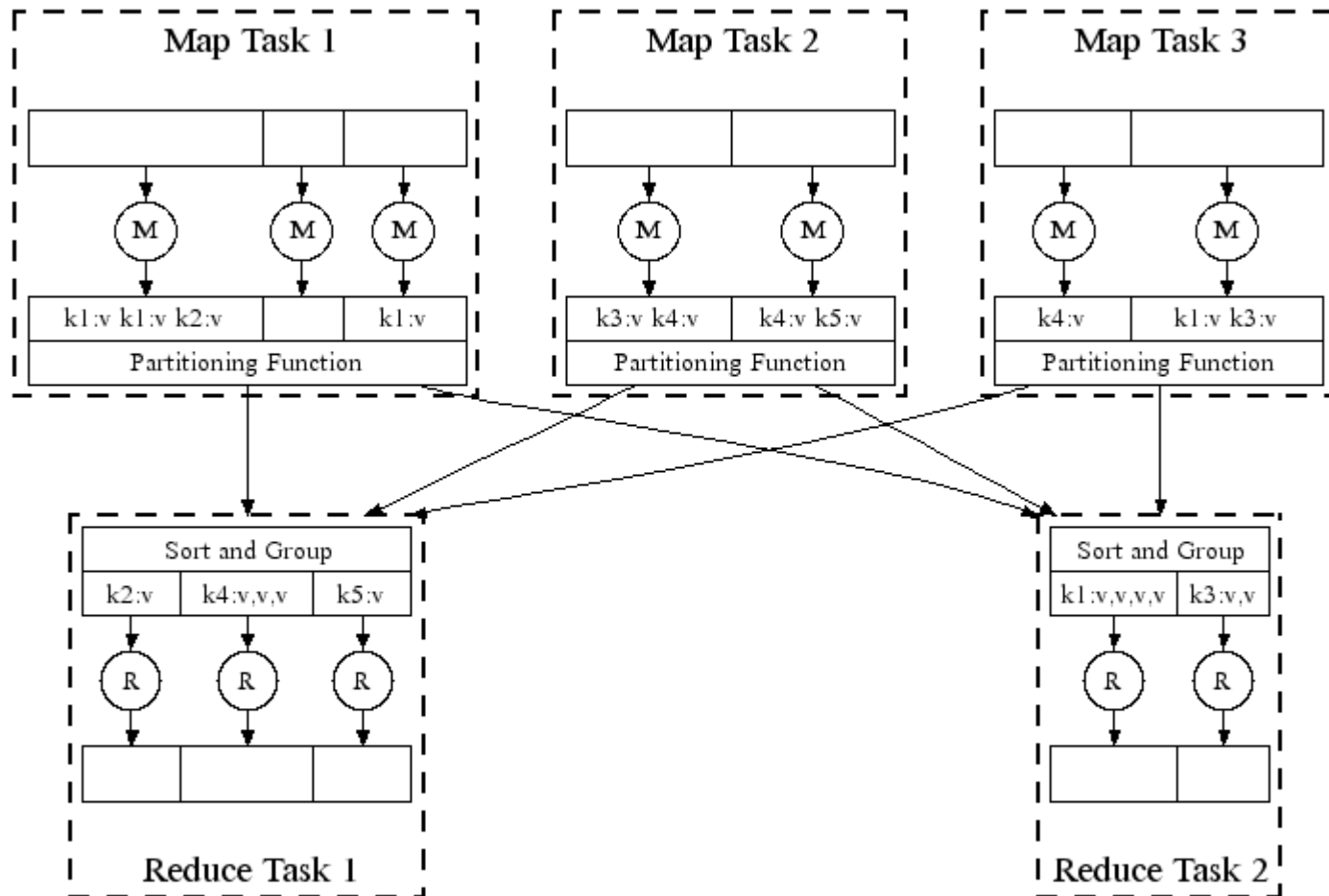
Collect all pairs with same key  
(Hash merge, Shuffle, Sort, Partition)

## Reduce:

Collect all values belonging to the key and output



# Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work



# MapReduce: Word Counting

Provided by the  
programmer

## MAP:

Read input and  
produces a set of  
key-value pairs

(The, 1)

(crew, 1)

(of, 1)

(the, 1)

(space, 1)

(shuttle, 1)

(Endeavor, 1)

(recently, 1)

....

(key, value)

## Group by key:

Collect all pairs  
with same key

(crew, 1)

(crew, 1)

(space, 1)

(the, 1)

(the, 1)

(the, 1)

(shuttle, 1)

(recently, 1)

...

(key, value)

Provided by the  
programmer

## Reduce:

Collect all values  
belonging to the  
key and output

(crew, 2)

(space, 1)

(the, 3)

(shuttle, 1)

(recently, 1)

...

(key, value)

Only sequential reads

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need .....

Big document

# 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: Environment

## MapReduce environment takes care of:

- **Partitioning** the input data
- **Scheduling** the program's execution across a set of machines
- Performing the **group by key** step
  - In practice this is the bottleneck
- Handling machine **failures**
- Managing required inter-machine **communication**

# Data Flow

- Input and final output are stored on a distributed file system (HDFS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local file system of Map and Reduce workers
- Output is often input to another MapReduce task

# Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

# Dealing with Failures

## ■ Map worker failure

- Map tasks completed or in-progress at worker are reset to idle
- Reduce workers are notified when task is rescheduled on another worker

## ■ Reduce worker failure

- Only in-progress tasks are reset to idle
- Reduce task is restarted

## ■ Master failure

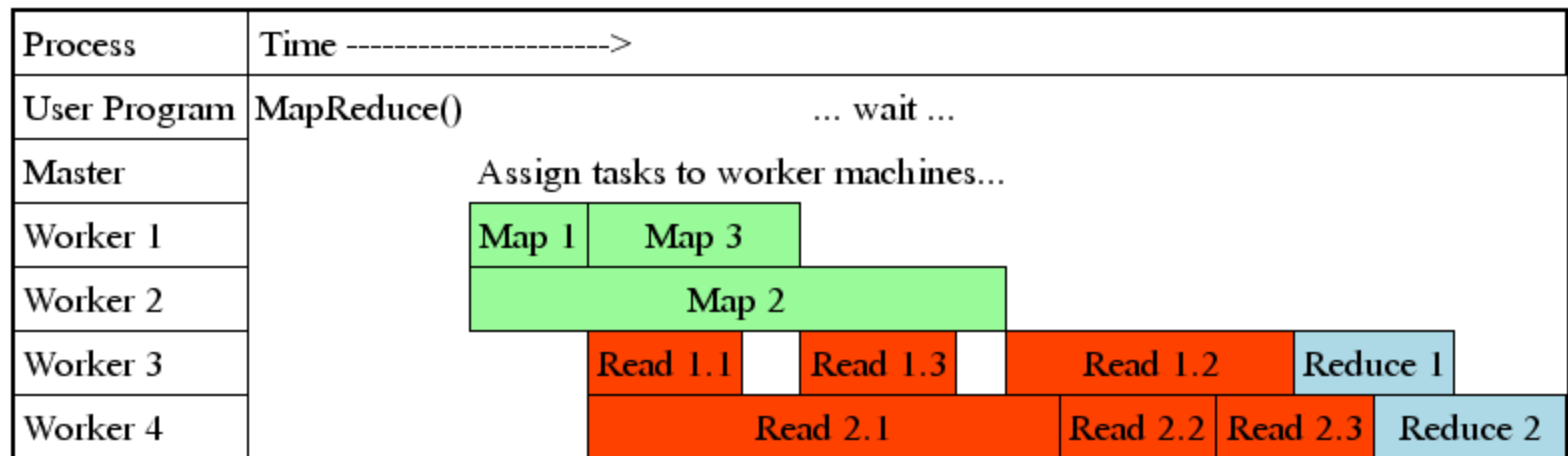
- MapReduce task is aborted and client is notified

# How many Map and Reduce jobs?

- $M$  map tasks,  $R$  reduce tasks
- **Rule of a thumb:**
  - Make  $M$  much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually  $R$  is smaller than  $M$** 
  - Because final output is spread across  $R$  files

# Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks  $\gg$  machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing





# Refinements: Backup Tasks

## ■ Problem

- Slow workers significantly lengthen the job completion time:
  - Other jobs on the machine
  - Bad disks
  - Weird things

## ■ Solution

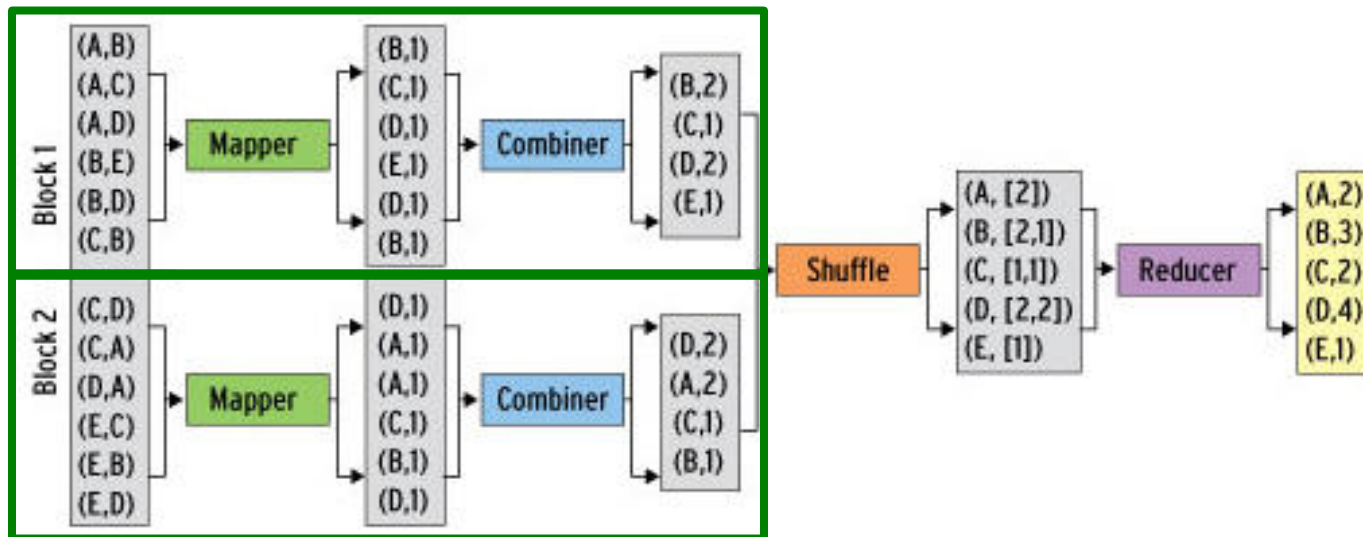
- Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first “wins”

## ■ Effect

- Dramatically shortens job completion time

# 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!
- Works if reduce function is **commutative and associative**

# Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - $\text{hash}(\text{key}) \bmod R$
- **Sometimes useful to override the hash function:**
  - E.g.,  $\text{hash}(\text{hostname}(\text{URL})) \bmod R$  ensures URLs from a host end up in the same output file

# Problems Suited for MapReduce

# Example: Host size

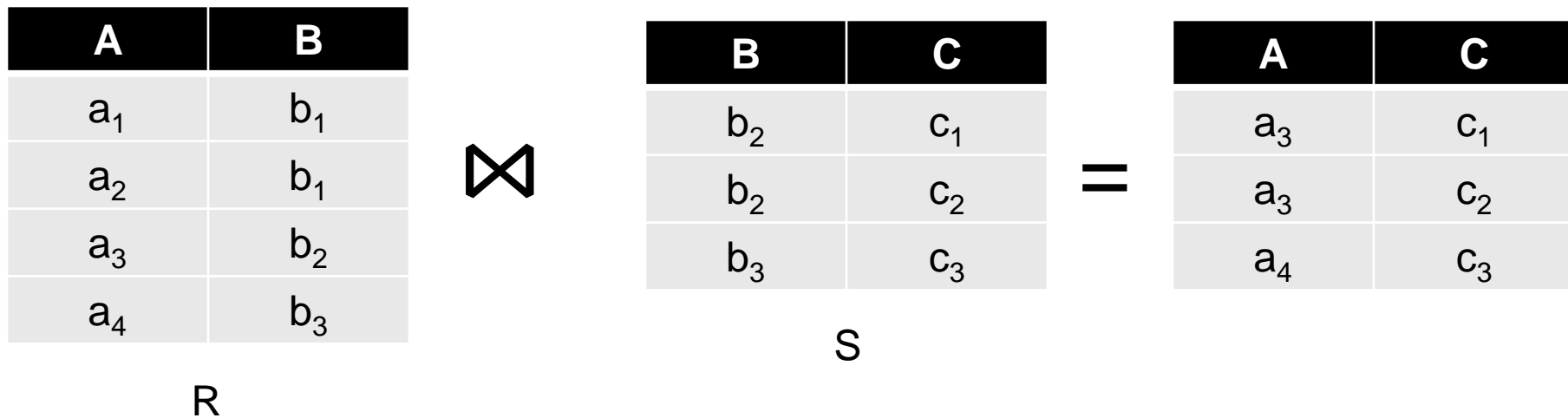
- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, ...)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host
- **Other examples:**
  - Link analysis and graph processing
  - Machine Learning algorithms

# Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

# Example: Join By Map-Reduce

- Compute the natural join  $R(A,B) \bowtie S(B,C)$
- $R$  and  $S$  are each stored in files
- Tuples are pairs  $(a,b)$  or  $(b,c)$



# Map-Reduce Join

- Use a hash function  $h$  from B-values to  $1...k$
- **A Map process turns:**
  - Each input tuple  $R(a,b)$  into key-value pair  $(b,(a,R))$
  - Each input tuple  $S(b,c)$  into  $(b,(c,S))$
- **Map processes** send each key-value pair with key  $b$  to Reduce process  $h(b)$ 
  - Hadoop does this automatically; just tell it what  $k$  is.
- Each **Reduce process** matches all the pairs  $(b,(a,R))$  with all  $(b,(c,S))$  and outputs  $(a,b,c)$ .



# Problems NOT suitable for MapReduce

- **MapReduce is great for:**
  - Problems that require sequential data access
  - Large batch jobs (**not** interactive, real-time)
- **MapReduce is inefficient for problems where random (or irregular) access to data required:**
  - **Graphs**
  - Interdependent data
    - Machine learning
    - Comparisons of many pairs of items

# Cost Measures for Algorithms

- In MapReduce we quantify the cost of an algorithm using
  1. *Communication cost* = total I/O of all processes
  2. *Elapsed communication cost* = max of I/O along any path
  3. (*Elapsed*) *computation cost* analogous, but count only running time of processes

Note that here the big-O notation is not the most useful (adding more machines is always an option)

# Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost** = input file size +  $2 \times$  (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - **Elapsed communication cost** is the sum of the largest input + output for any map process, plus the same for any reduce process

# What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other
- Total cost tells what you pay in rent from your friendly neighborhood cloud
- Elapsed cost is wall-clock time using parallelism

# Cost of Map-Reduce Join

- **Total communication cost**  
 $= O(|R| + |S| + |R \bowtie S|)$
- **Elapsed communication cost**  $= O(s)$ 
  - We're going to pick  $k$  and the number of Map processes so that the I/O limit  $s$  is respected
  - We put a limit  $s$  on the amount of input or output that any one process can have.  **$s$  could be:**
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost

# Pointers and Further Reading

# Implementations

- **Google's MapReduce**
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: <http://lucene.apache.org/hadoop/>
- **Aster Data**
  - Cluster-optimized SQL Database that also implements MapReduce

# Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- Amazon's "Elastic Compute Cloud" (EC2)
- Aster Data and Hadoop can both be run on EC2
- **For CS341 (offered next quarter) Amazon will provide free access for the class**



# Reading

- Jeffrey Dean and Sanjay Ghemawat:  
MapReduce: Simplified Data Processing on  
Large Clusters
  - <http://labs.google.com/papers/mapreduce.html>
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - <http://labs.google.com/papers/gfs.html>

# Resources

- Hadoop Wiki
  - Introduction
    - <http://wiki.apache.org/lucene-hadoop/>
  - Getting Started
    - <http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop>
  - Map/Reduce Overview
    - <http://wiki.apache.org/lucene-hadoop/HadoopMapReduce>
    - <http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses>
  - Eclipse Environment
    - <http://wiki.apache.org/lucene-hadoop/EclipseEnvironment>
- Javadoc
  - <http://lucene.apache.org/hadoop/docs/api/>

# Resources

- Releases from Apache download mirrors
  - <http://www.apache.org/dyn/closer.cgi/lucene/hadoop/>
- Nightly builds of source
  - <http://people.apache.org/dist/lucene/hadoop/nightly/>
- Source code from subversion
  - [http://lucene.apache.org/hadoop/version\\_control.html](http://lucene.apache.org/hadoop/version_control.html)

# Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]