Data-Intensive Computing with MapReduce

Session 1: Introduction to MapReduce

Jimmy Lin
University of Maryland
Thursday, January 24, 2013

What is this course about?

- Computing on “big data”
- Focus on applications and algorithm design
- MapReduce… and beyond

Source: Wikipedia (All Souls College, Oxford)

... to building sh*t that works

Source: Wikipedia (Factory)


UNIVERSITY OF
18
56
MARYLAND
... and back.
Source: Wikipedia (All Souls College, Oxford)

@lintool

Source: Wikipedia (Hard disk drive)

Google™

processes 20 PB a day (2008)
crawls 20B web pages a day (2012)

JPMorganChase

150 PB on 50k+ servers
running 15k apps (6/2011)

ebay

>10 PB data, 75B DB
calls per day (6/2012)

INTERNET ARCHIVE

Wayback Machine: 240B web
pages archived, 5 PB (1/2013)

>100 PB of user data +
500 TB/day (8/2012)

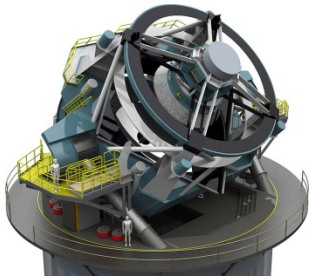
facebook

LHC: ~15 PB a year

CERN

amazon web services™

S3: 449B objects, peak 290k
request/second (7/2011)
1T objects (6/2012)



LSST: 6-10 PB a year
(~2015)

SKA: 0.3 – 1.5 EB
per year (~2020)

How much data?

Why big data?

Science
Engineering
Commerce

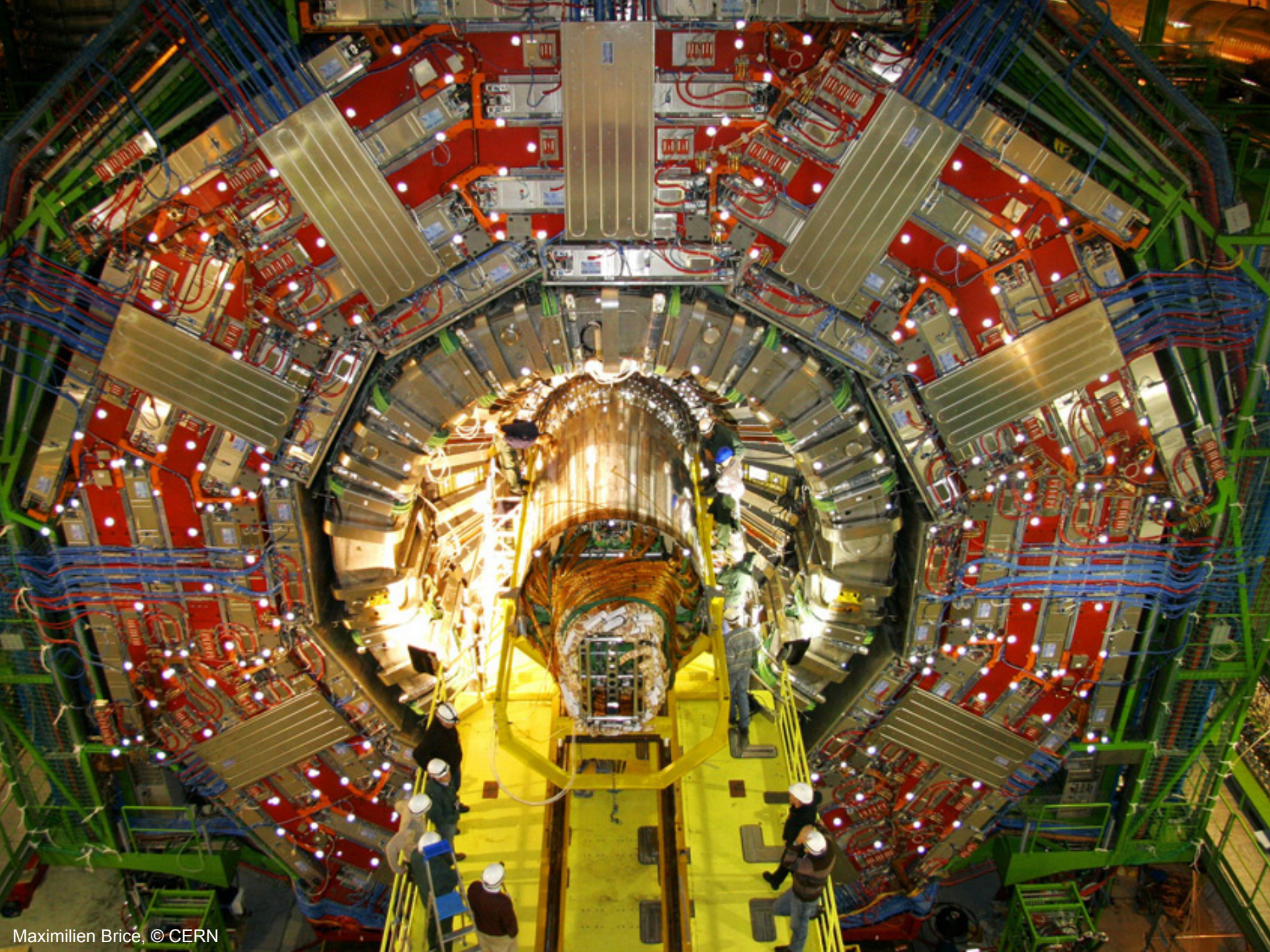
Source: Wikipedia (Everest)

Science

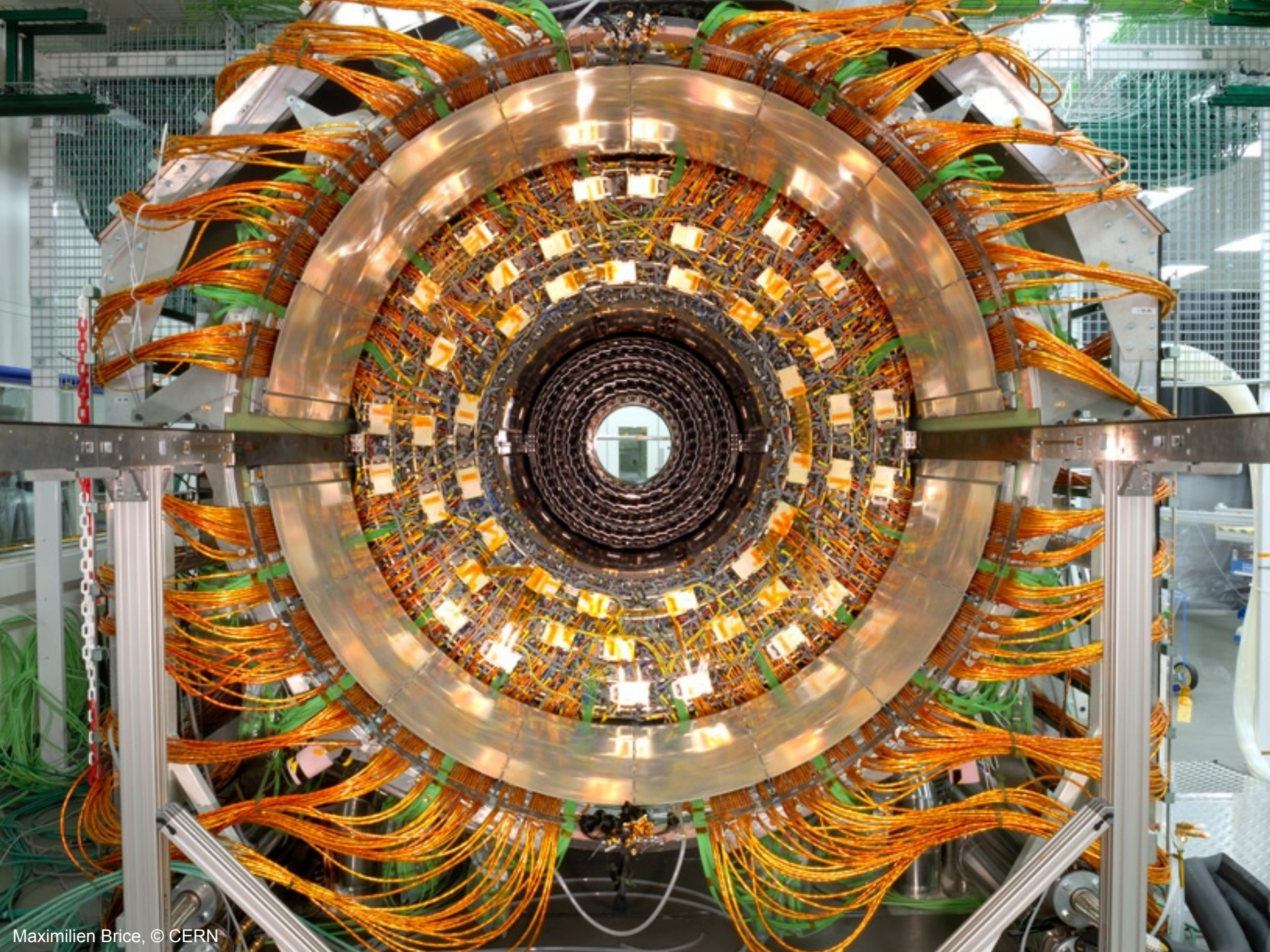
Emergence of the 4th Paradigm

Data-intensive e-Science

Maximilien Brice, © CERN



Maximilien Brice, © CERN



Maximilien Brice, © CERN

Engineering

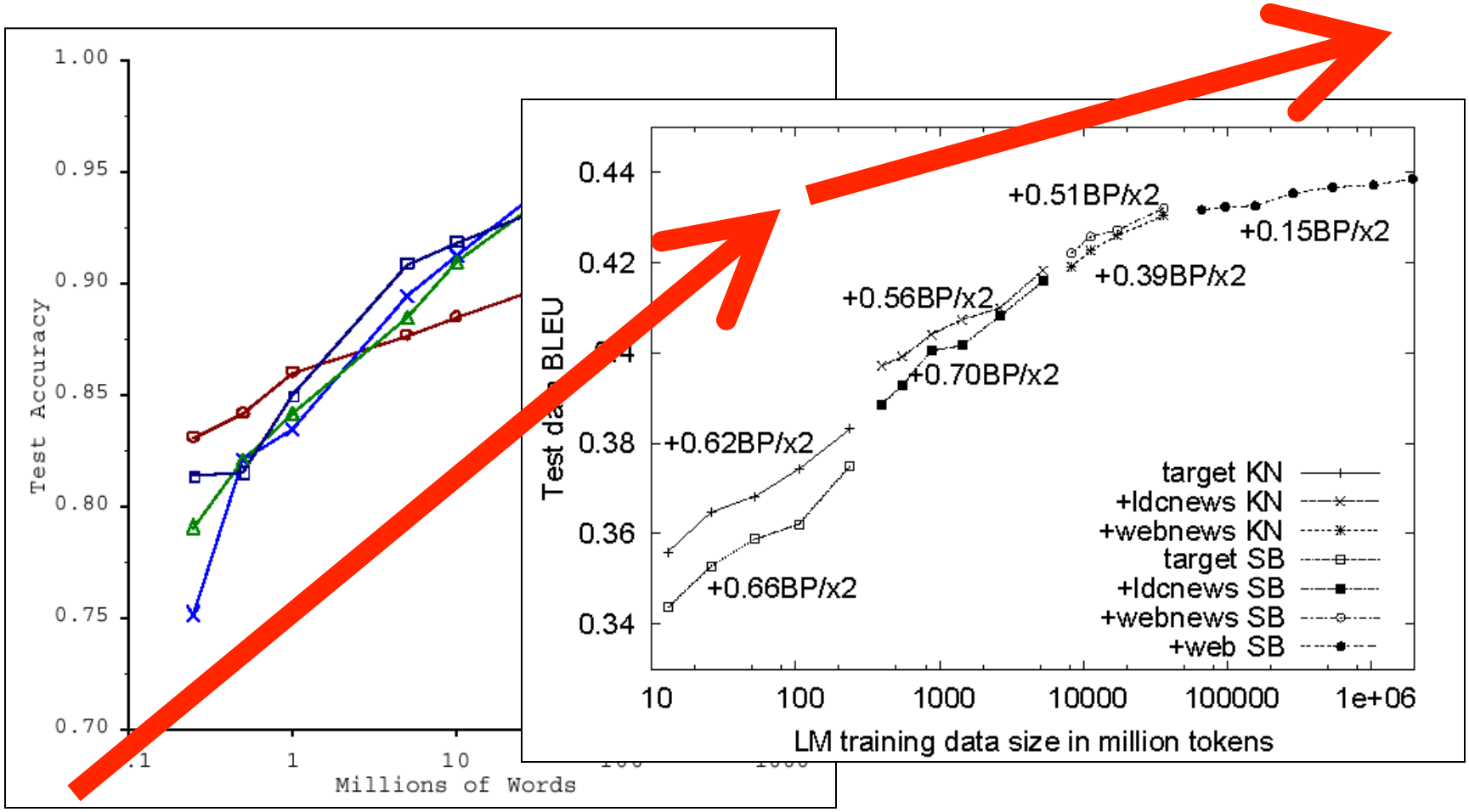
The unreasonable effectiveness of data

Count and normalize!

Source: Wikipedia (Three Gorges Dam)

No data like more data!

s/knowledge/data/g;



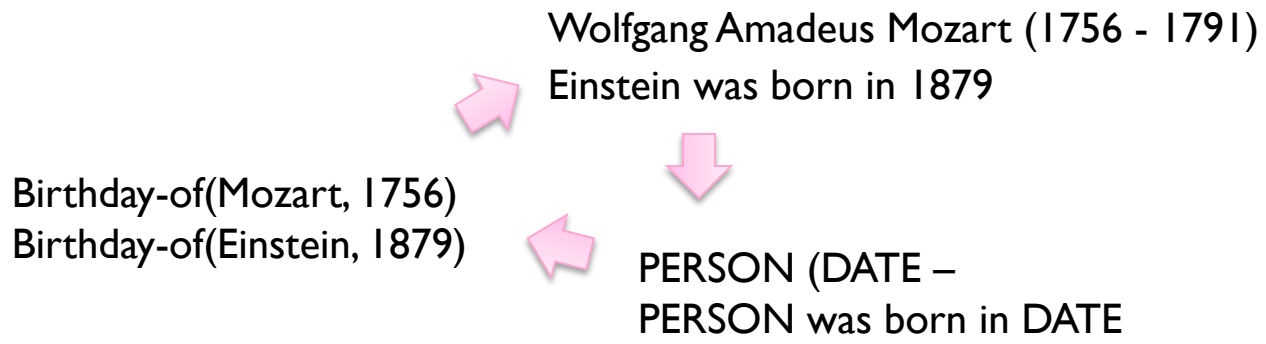
(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)

What to do with more data?

- Answering factoid questions
  - Pattern matching on the Web
  - Works amazingly well

Who shot Abraham Lincoln? → X shot Abraham Lincoln

- Learning relations
  - Start with seed instances
  - Search for patterns on the Web
  - Using patterns to find more instances



(Brill et al., TREC 2001; Lin, ACM TOIS 2007)
(Agichtein and Gravano, DL 2000; Ravichandran and Hovy, ACL 2002; …)


Know thy customers
Data → Insights → Competitive advantages
Commerce
EPSON
PEARL BAR
Source: Wikiedia (Shinjuku, Tokyo)

Source: Wikipedia (Noctilucent cloud)

Course Administrivia

Source: http://www.flickr.com/photos/artmind_etcetera/6336693594/

Course Pre-requisites (I)

- Competent Java programming
  - But this course is *not* about programming
  - Focus on "thinking at scale" and algorithm design
  - You're expected to pick up Hadoop with minimal help
- Be good at debugging
  - Google it!

Course Pre-requisites (II)

- Basic knowledge of
  - Probability and statistics, discrete math
  - Computer architecture
- No previous experience necessary in
  - MapReduce
  - Parallel and distributed programming
- Curiosity

How will I actually learn Hadoop?

- Next class session
- Hadoop: The Definitive Guide
- RTFM
- RTFC(!)

This course is not for you...

- If you're not genuinely interested in the topic
- If you can't put in the time
- If you're uncomfortable with uncertainty, unpredictability, etc. that comes with bleeding-edge software

Otherwise, this will be a rewarding and fun course!

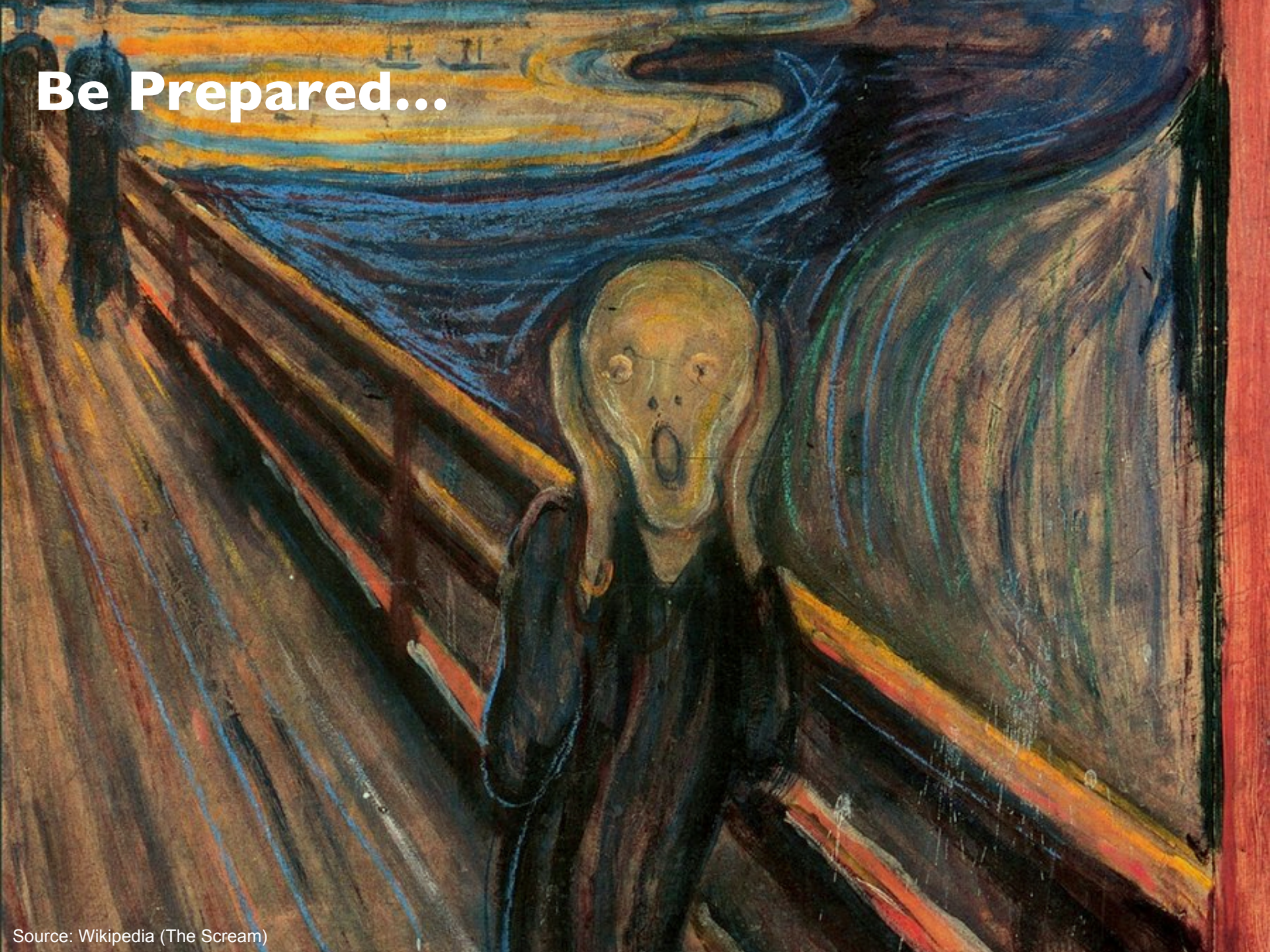
Details, Details…

- Make sure you're on the mailing list!
- Textbooks
- Components of the final grade:
  - Assignments
  - Midterm
  - Final exam
  - Final project
- I am unlikely to accept the following excuses:
  - "Too busy"
  - "It took longer than I thought it would take"
  - "It was harder than I initially thought"
  - "My dog ate my homework" and modern variants thereof

Hadoop Resources

- Hadoop on your local machine
- Hadoop in a virtual machine on your local machine
- Hadoop on Amazon EC2

Be Prepared...



Source: Wikipedia (The Scream)

“Hadoop Zen”

- This is bleeding-edge technology (= immature!)
  - We’ve come a long way since 2007, but still far to go…
  - Bugs, undocumented “features”, inexplicable behavior, data loss(!)
- Don’t get frustrated (take a deep breath)…
  - Those W$*#T@F! moments
- Be patient…
  - We will inevitably encounter “situations” along the way
- Be flexible…
  - We will have to be creative in workarounds
- Be constructive…
  - Tell me how I can make everyone’s experience better

“Hadoop Zen”

Source: Wikipedia (Japanese rock garden)

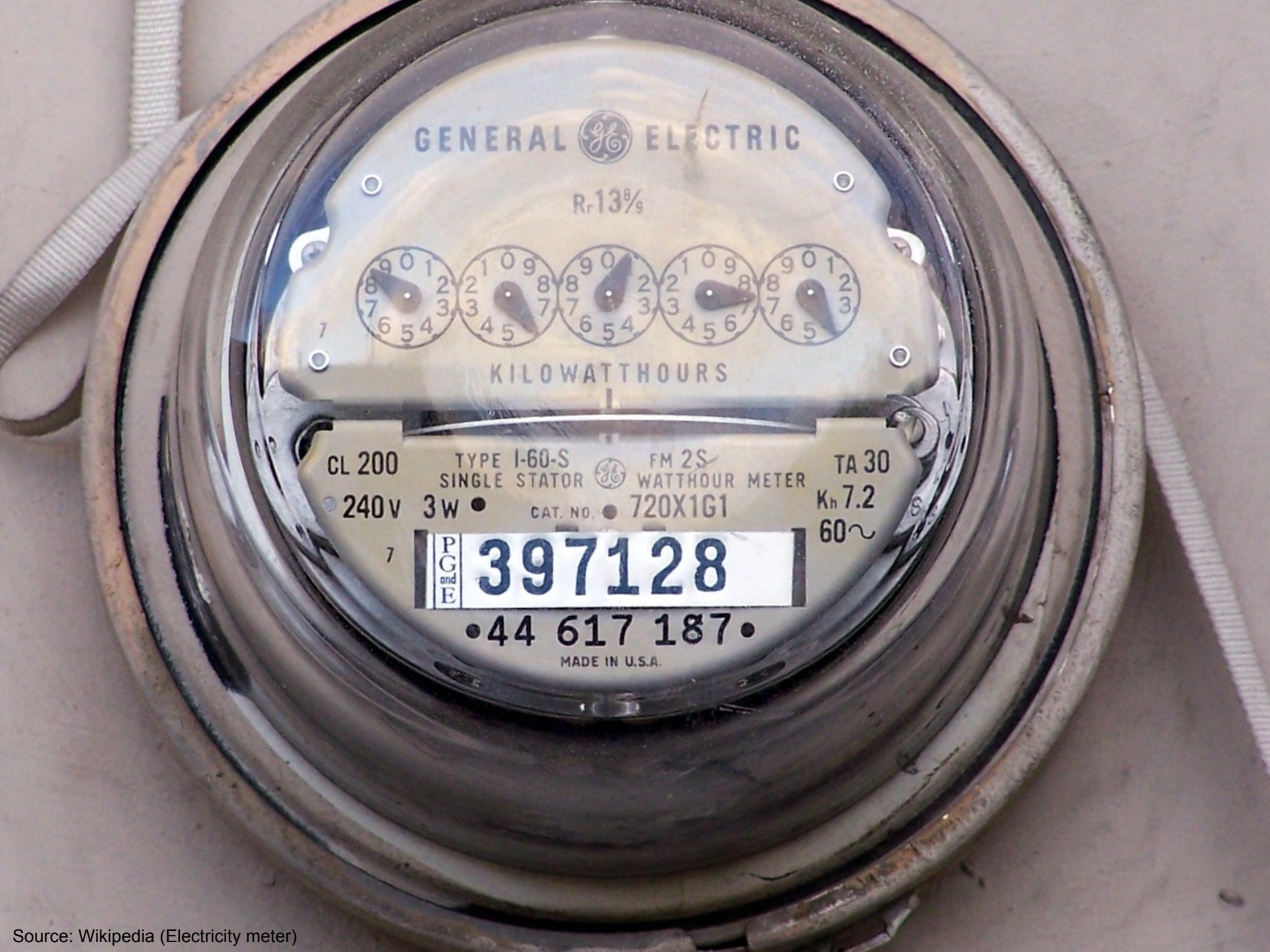
Source: Wikipedia (Clouds)

The best thing since sliced bread?

- Before clouds…
  - Grids
  - Connection machine
  - Vector supercomputers
  - …
- Cloud computing means many different things:
  - Big data
  - Rebranding of web 2.0
  - Utility computing
  - Everything as a service

Rebranding of web 2.0

- Rich, interactive web applications
  - Clouds refer to the servers that run them
  - AJAX as the de facto standard (for better or worse)
  - Examples: Facebook, YouTube, Gmail, …
- "The network is the computer": take two
  - User data is stored "in the clouds"
  - Rise of the netbook, smartphones, etc.
  - Browser *is* the OS



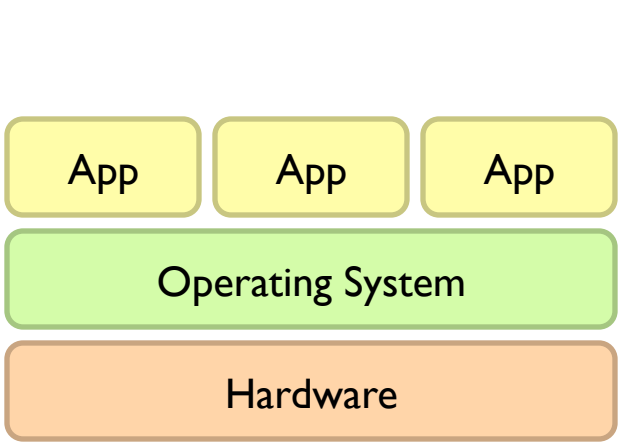
Source: Wikipedia (Electricity meter)

Utility Computing

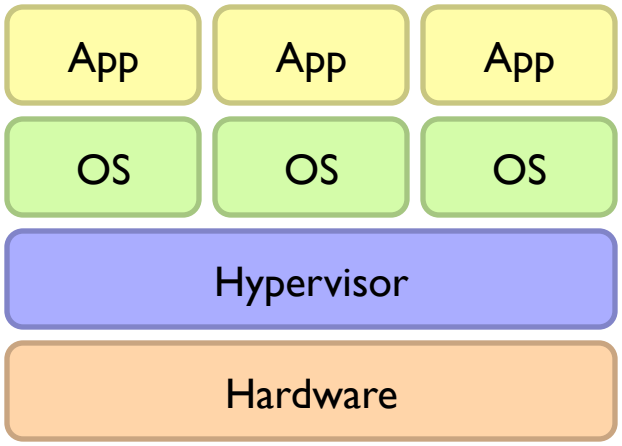
- What?
  - Computing resources as a metered service ("pay as you go")
  - Ability to dynamically provision virtual machines
- Why?
  - Cost: capital vs. operating expenses
  - Scalability: "infinite" capacity
  - Elasticity: scale up or down on demand
- Does it make sense?
  - Benefits to cloud users
  - Business case for cloud providers

I think there is a world market for about five computers.

Enabling Technology: Virtualization



Traditional Stack



Virtualized Stack

Everything as a Service

- Utility computing = Infrastructure as a Service (IaaS)
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, Rackspace
- Platform as a Service (PaaS)
  - Give me nice API and take care of the maintenance, upgrades, …
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Salesforce

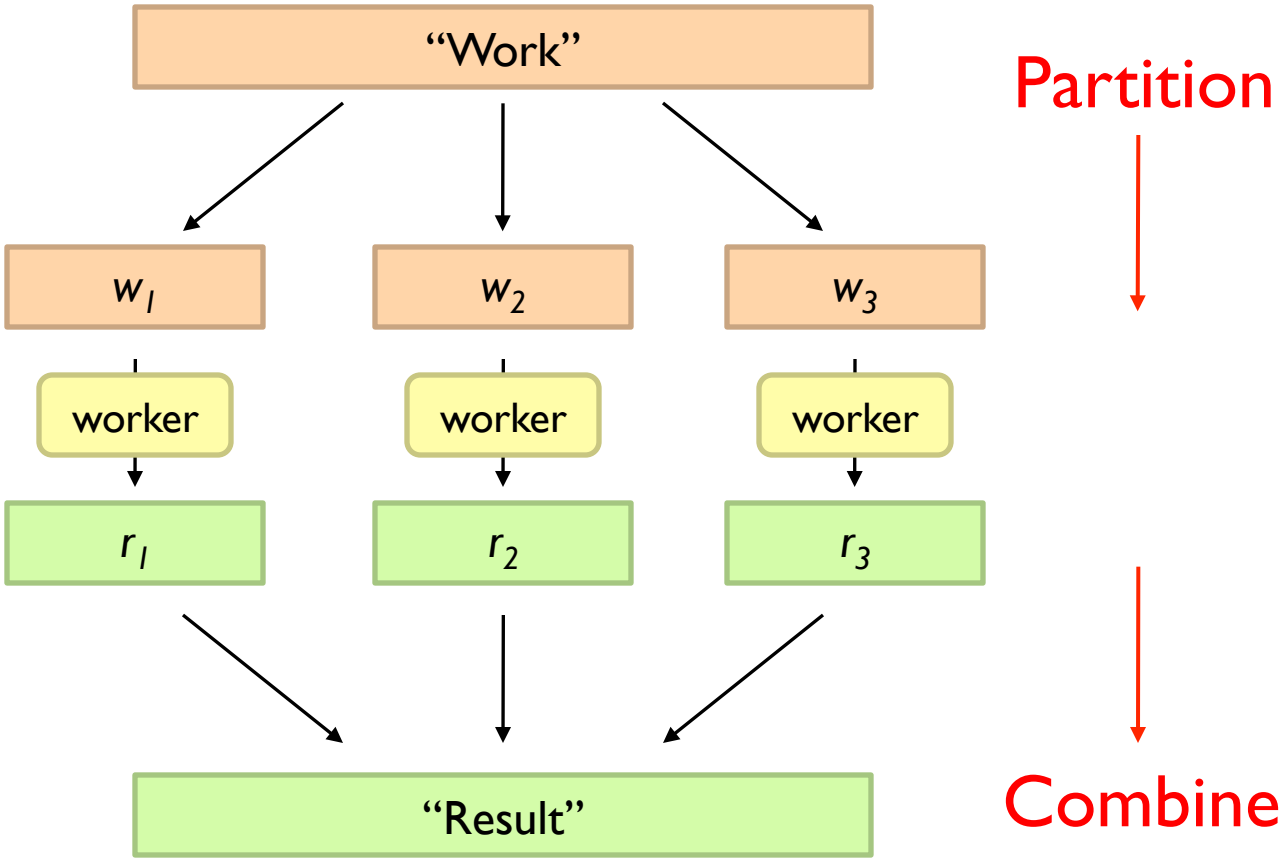
Who cares?

- Ready-made big data problems
  - Social media, user-generated content = big data
  - Examples: Facebook friend suggestions, Google ad placement
  - Business intelligence: gather everything in a data warehouse and run analytics to generate insight
- Utility computing provides:
  - Ability to provision Hadoop clusters on-demand in the cloud
  - lower barrier to entry for tackling big data problems
  - Commoditization and democratization of big data capabilities

Source: Google

Source: http://www.flickr.com/photos/22897666@N00/5711185410/

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's 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

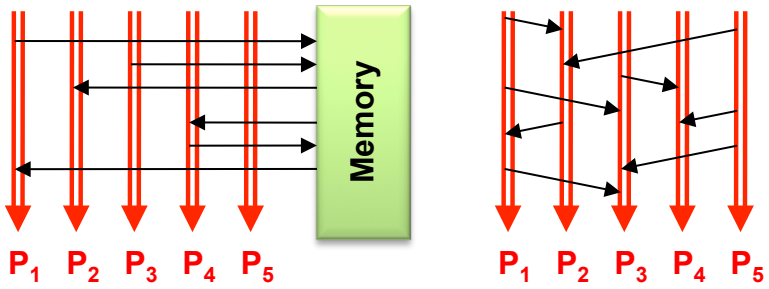
Source: Ricardo Guimarães Herrmann

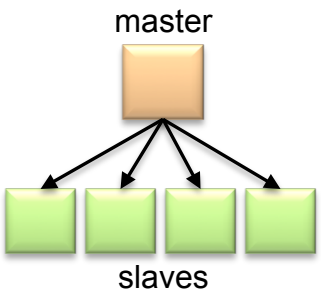
Managing Multiple Workers

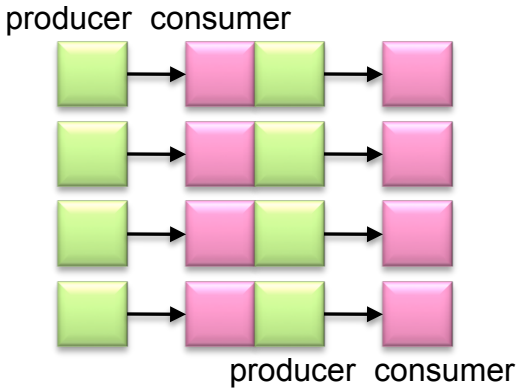
- Difficult because
  - We don't know the order in which workers run
  - We don't know when workers interrupt each other
  - We don't know when workers need to communicate partial results
  - We don't know the order in which workers access shared data
- Thus, we need:
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still, lots of problems:
  - Deadlock, livelock, race conditions...
  - Dining philosophers, sleeping barbers, cigarette smokers...
- Moral of the story: be careful!

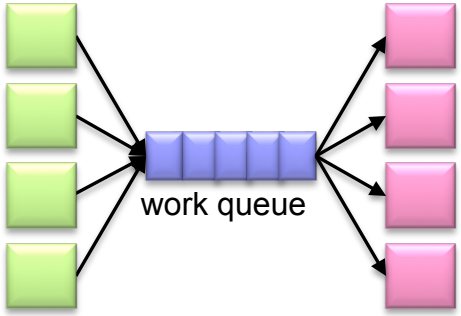
Current Tools

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





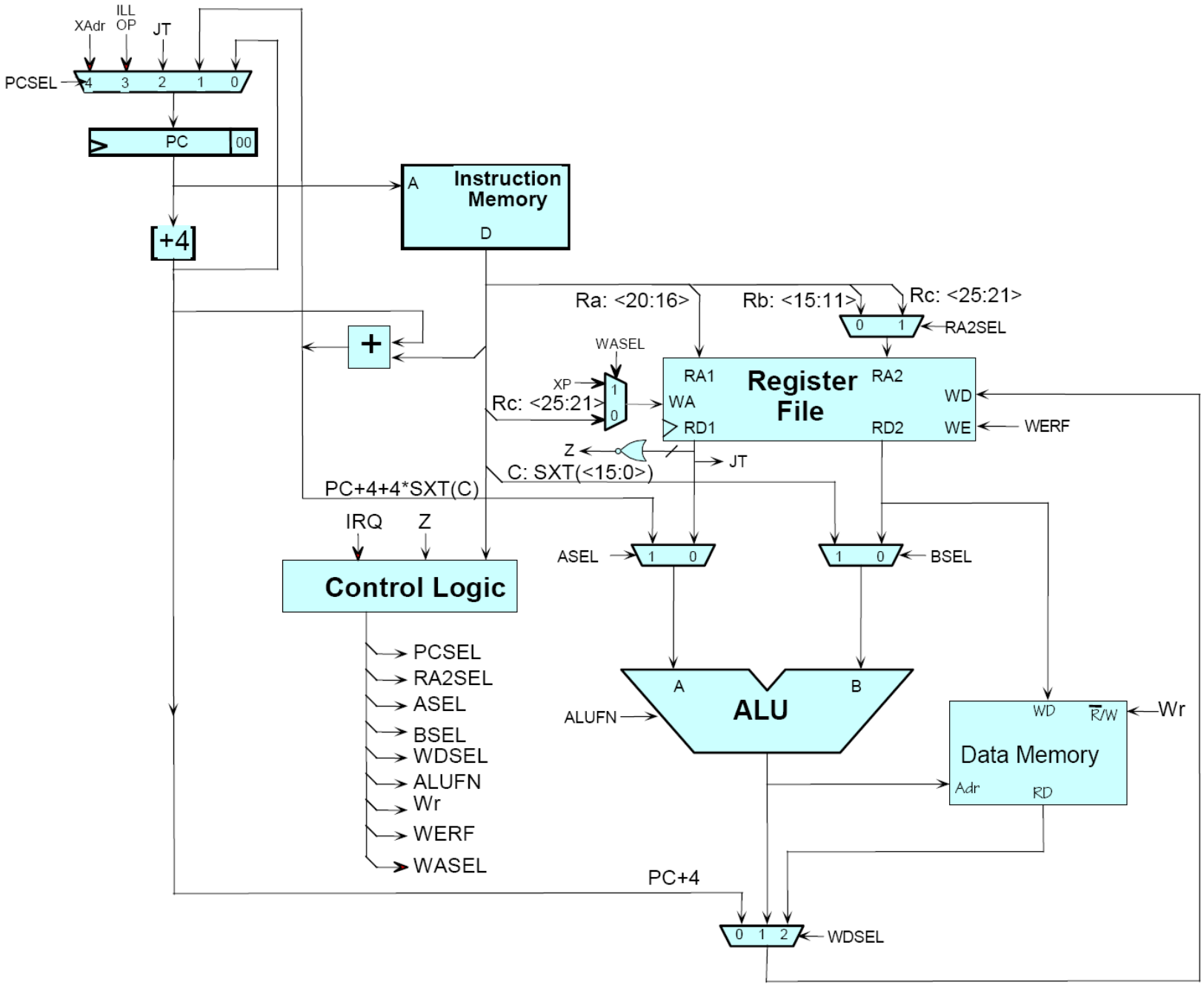




Where the rubber meets the road

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
  - At the scale of datacenters and across datacenters
  - In the presence of failures
  - In terms of multiple interacting services
- Not to mention debugging…
- The reality:
  - Lots of one-off solutions, custom code
  - Write you own dedicated library, then program with it
  - Burden on the programmer to explicitly manage everything

Source: Wikipedia (Flat Tire)



Source: MIT Open Courseware

Source: MIT Open Courseware

Source: Google

What's the point?

- It's all about the right level of abstraction
  - Moving beyond the von Neumann architecture
  - We need better programming models
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

The datacenter *is* the computer!

“Big Ideas”

- Scale “out”, not “up”
  - Limits of SMP and large shared-memory machines
- Move processing to the data
  - Cluster have limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - From the mythical man-month to the tradable machine-hour

Source: Google

Typical Big Data Problem

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

Map

Reduce

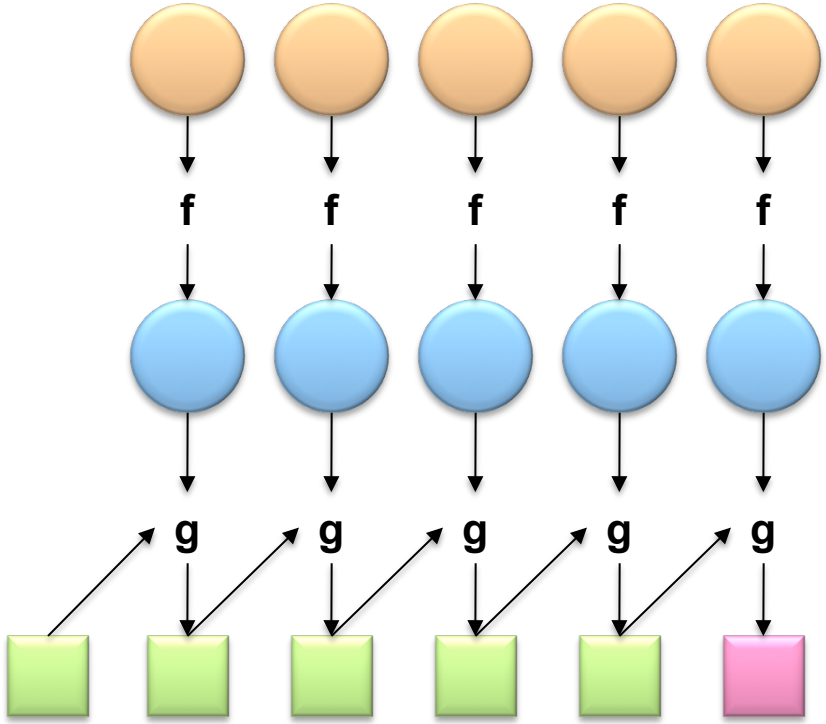
Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

Roots in Functional Programming

Map

Fold



MapReduce

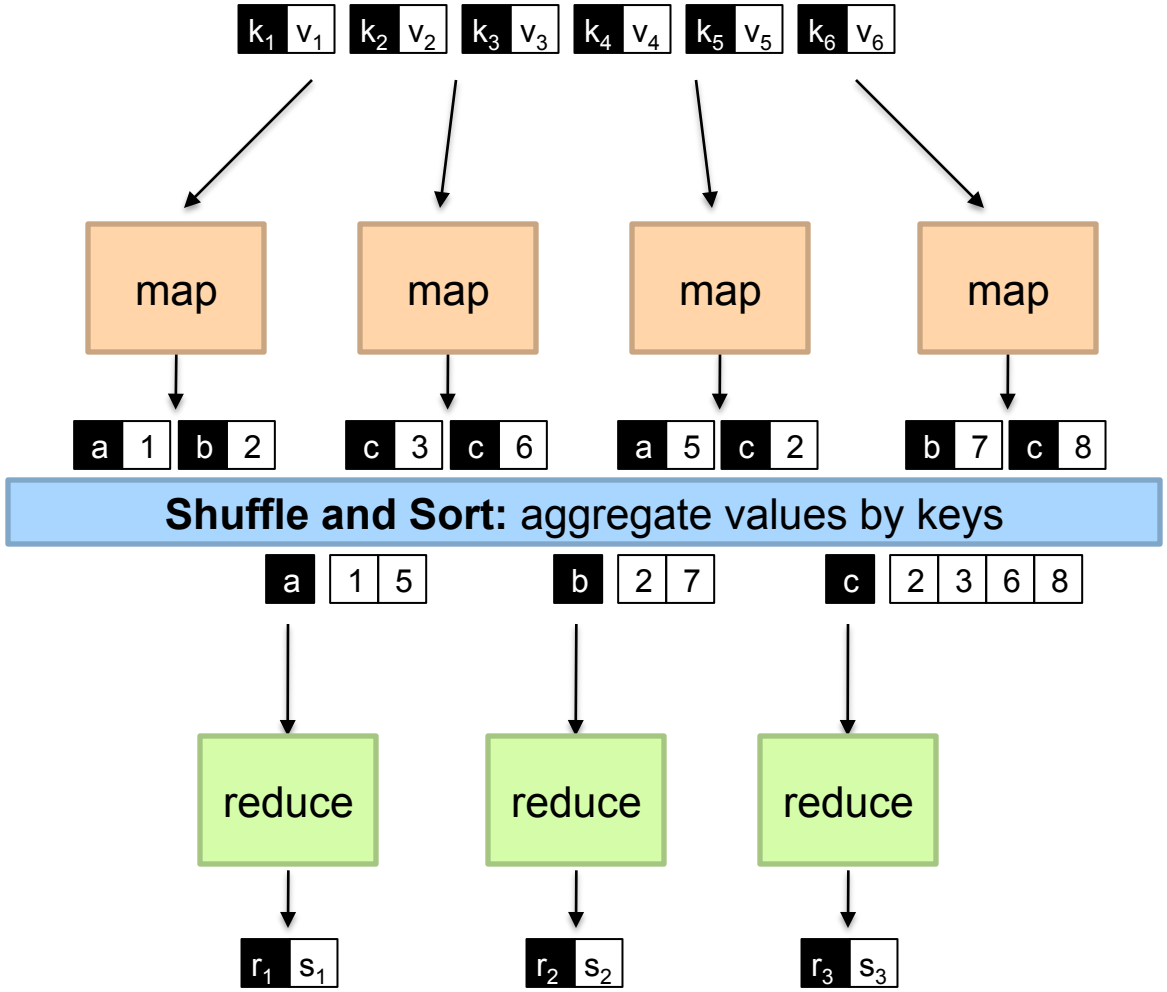
- Programmers specify two functions:

  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…


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

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…

What's "everything else"?

MapReduce "Runtime"

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

MapReduce

- Programmers specify two functions:

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

  - All values with the same key are reduced together

- The execution framework handles everything else…

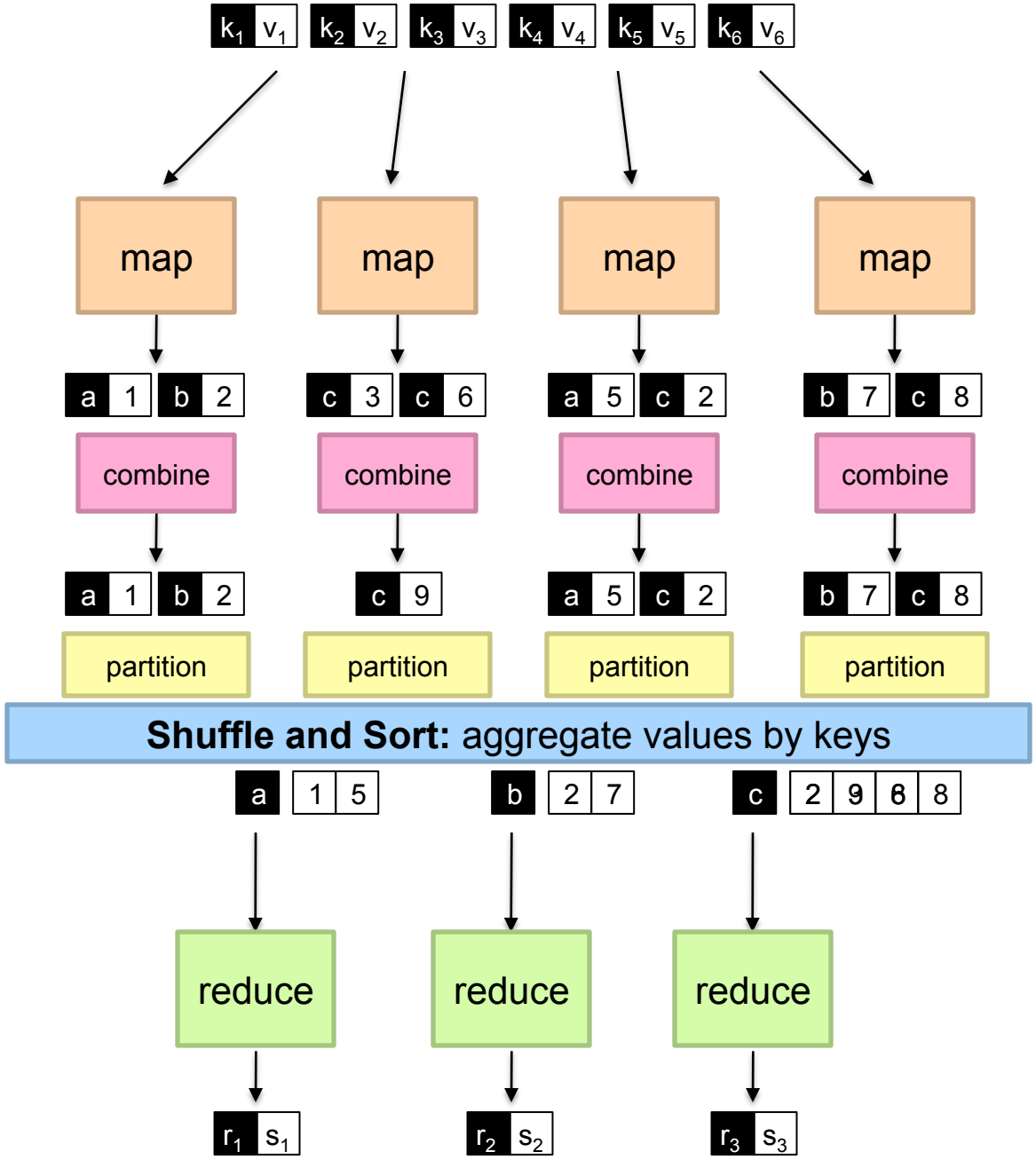
- Not quite…usually, programmers also specify:

  partition (k', number of partitions) → partition for k'

  - Often a simple hash of the key, e.g., hash(k') mod n
  - Divides up key space for parallel reduce operations

  combine (k', v') → <k', v'>*

  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
combine
combine
combine
combine
a 1
b 2
c 9
a 5
c 2
b 7
c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Two more details…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers

“Hello World”: Word Count

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```

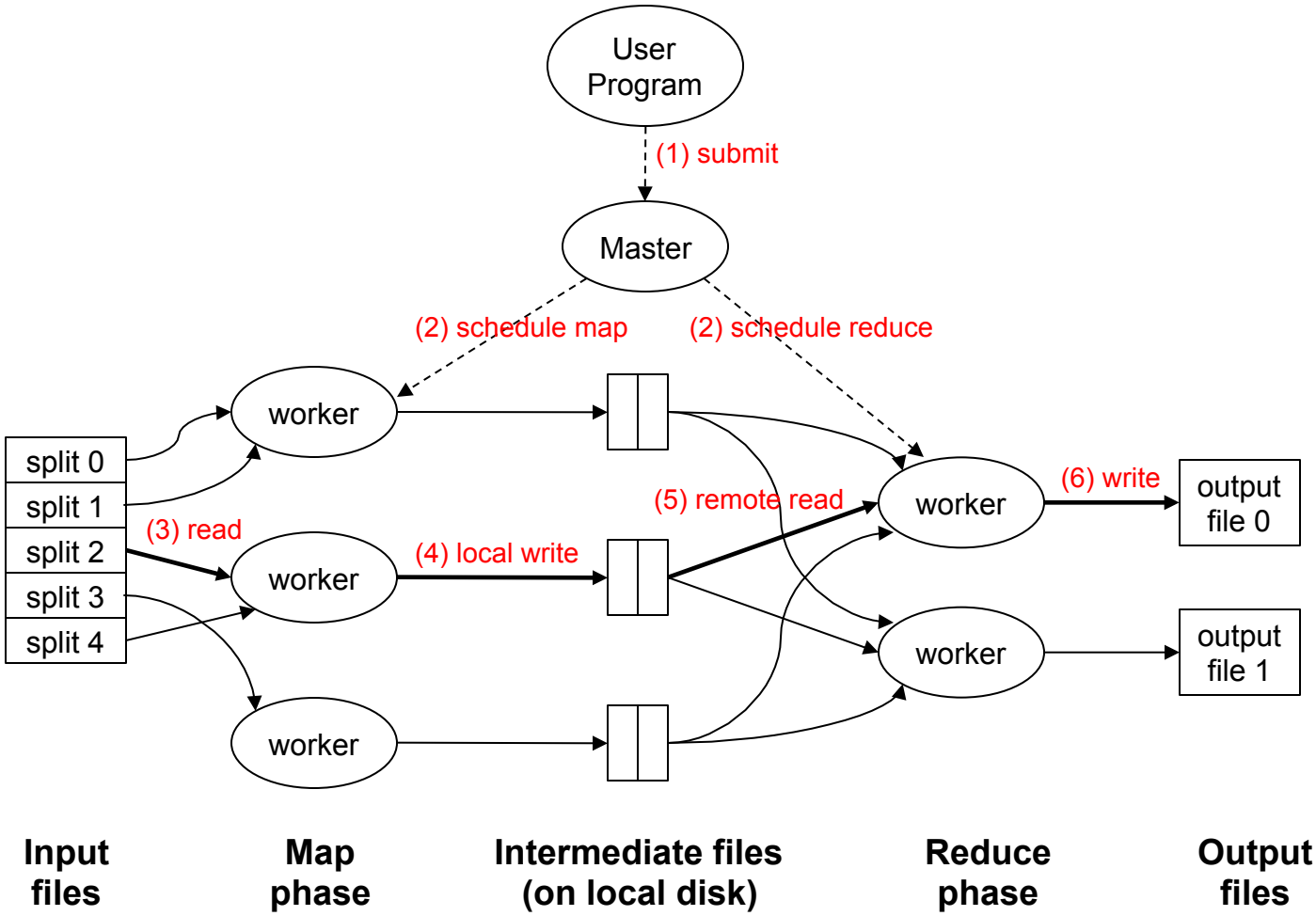
MapReduce can refer to…

- The programming model
- The execution framework (aka “runtime”)
- The specific implementation

Usage is usually clear from context!

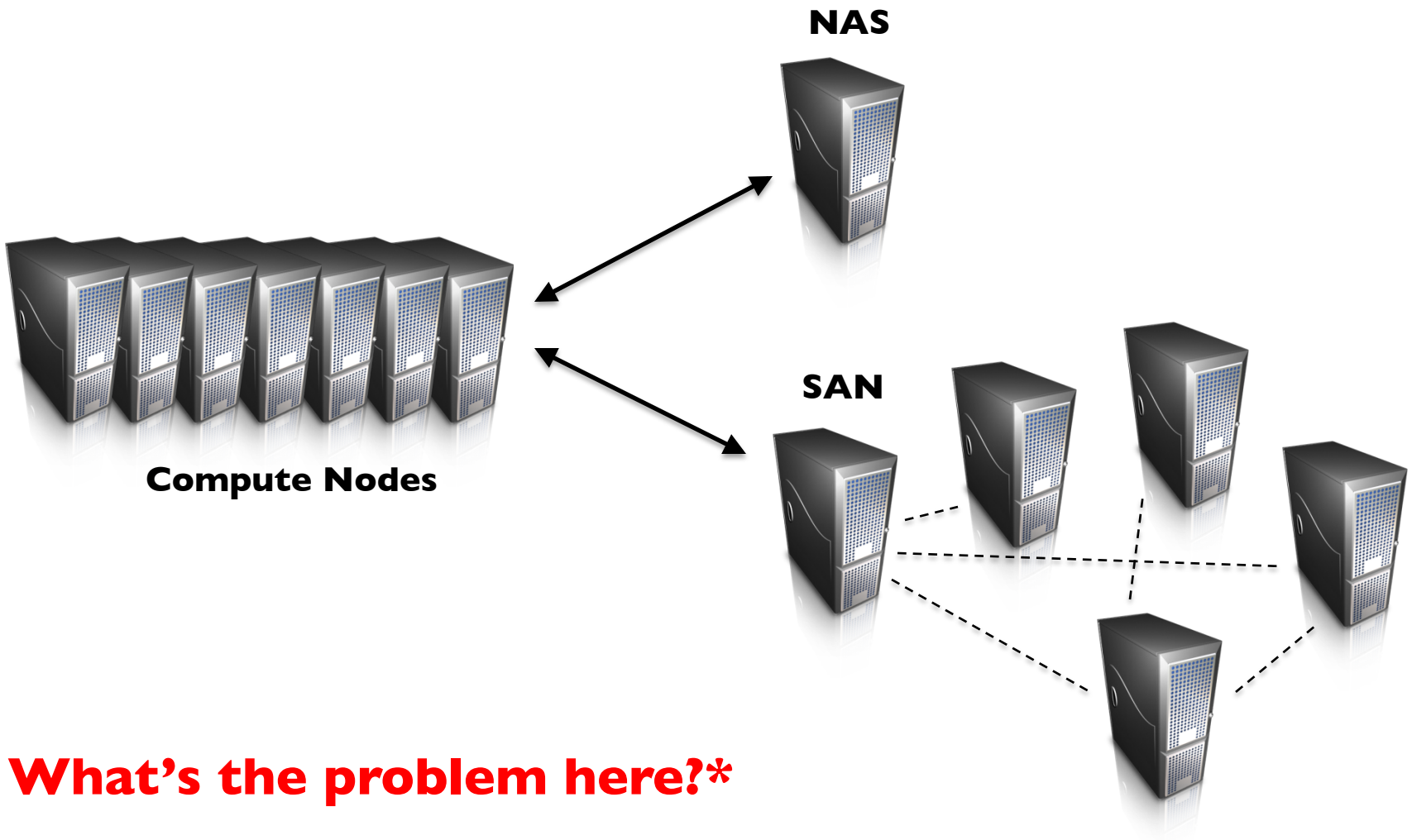
MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, now an Apache project
  - Used in production at Yahoo, Facebook, Twitter, LinkedIn, Netflix, …
  - The *de facto* big data processing platform
  - Rapidly expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.



Adapted from (Dean and Ghemawat, OSDI 2004)

How do we get data to the workers?



What's the problem here?*

* Really a problem?

Distributed File System

- Don't move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google's MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

GFS: Assumptions

- Commodity hardware over "exotic" hardware
  - Scale "out", not "up"
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large datasets, streaming reads
- Simplify the API
  - Push some of the issues onto the client (e.g., data layout)

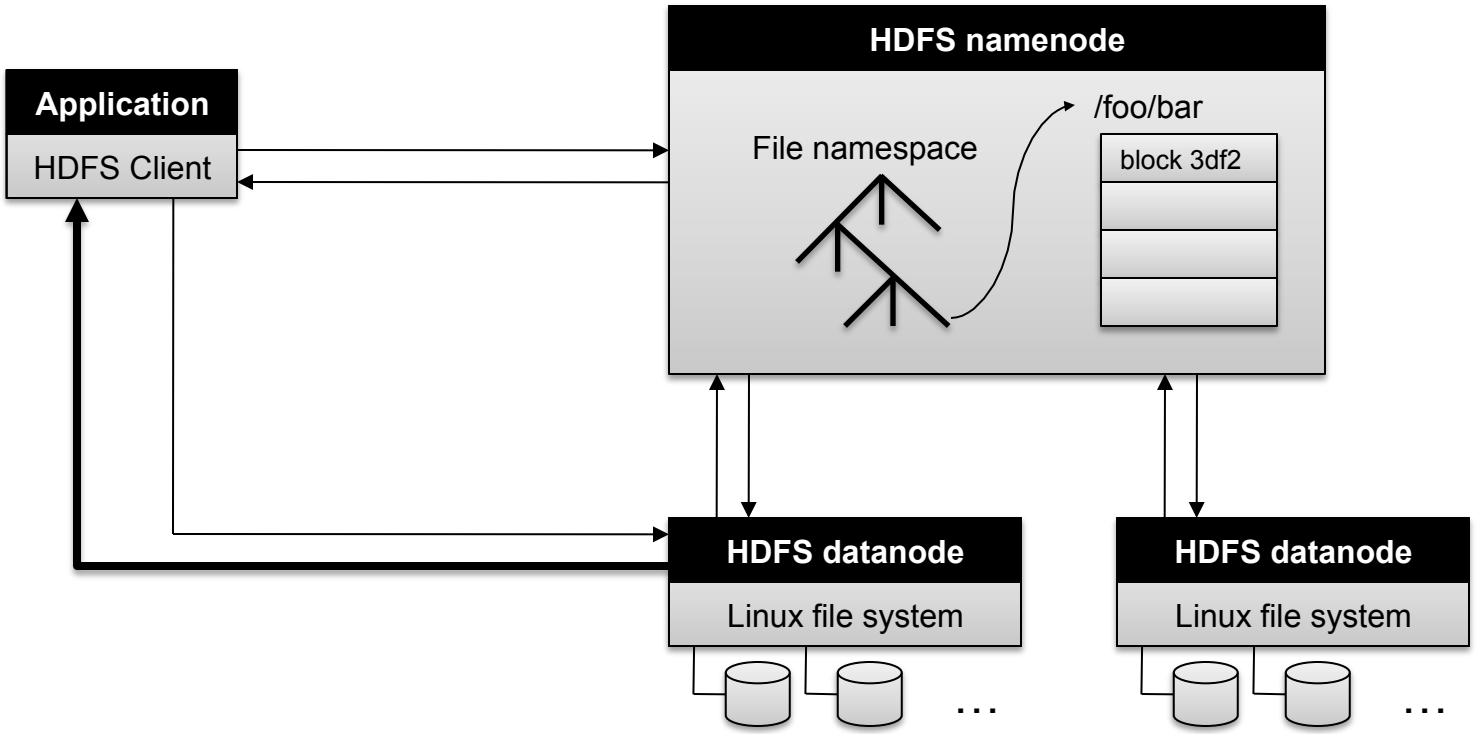
HDFS = GFS clone (same basic ideas)

From GFS to HDFS

- Terminology differences:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Differences:
  - Different consistency model for file appends
  - Implementation
  - Performance

For the most part, we'll use Hadoop terminology…

HDFS Architecture

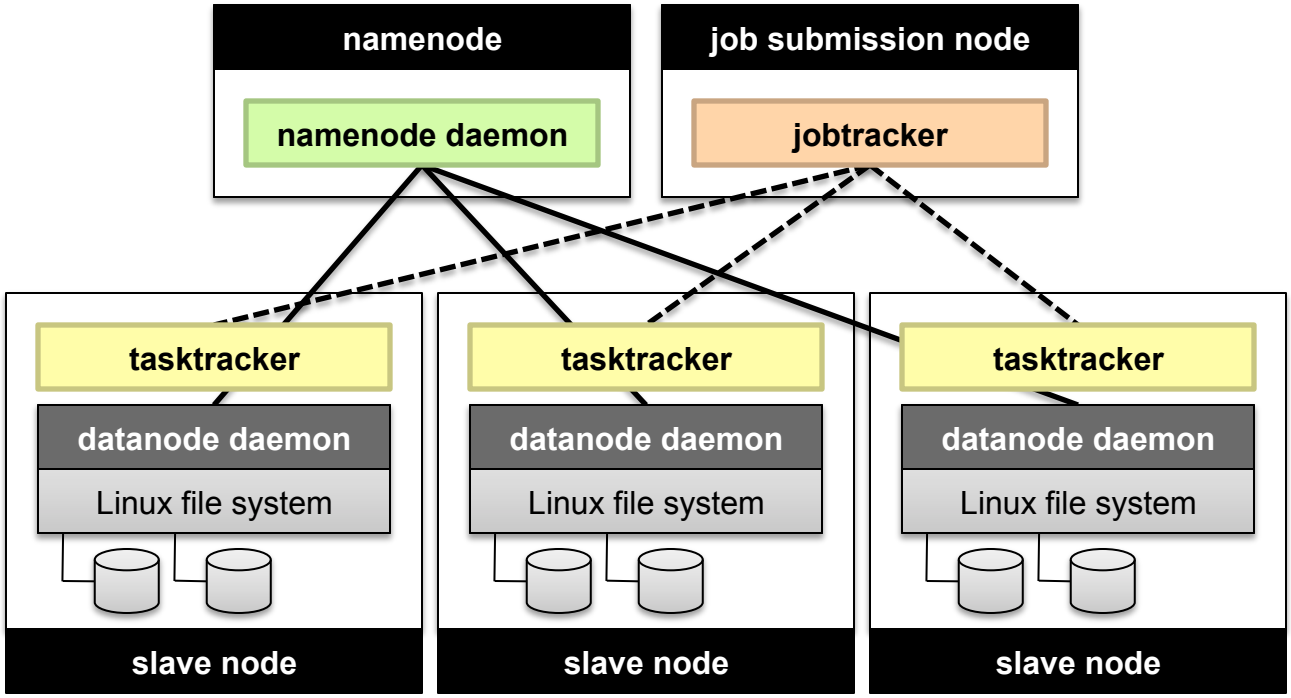


Adapted from (Ghemawat et al., SOSP 2003)

Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

Putting everything together…



(Not Quite… We'll come back to YARN later)

Source: Wikipedia (Japanese rock garden)