Data-Intensive Information Processing Applications — Session #1

Introduction to MapReduce

Jimmy Lin
University of Maryland

Tuesday, January 26, 2010

What is this course about?

- Data-intensive information processing
- Large-data (“web-scale”) problems
- Focus on applications
- MapReduce… and beyond

What is MapReduce?

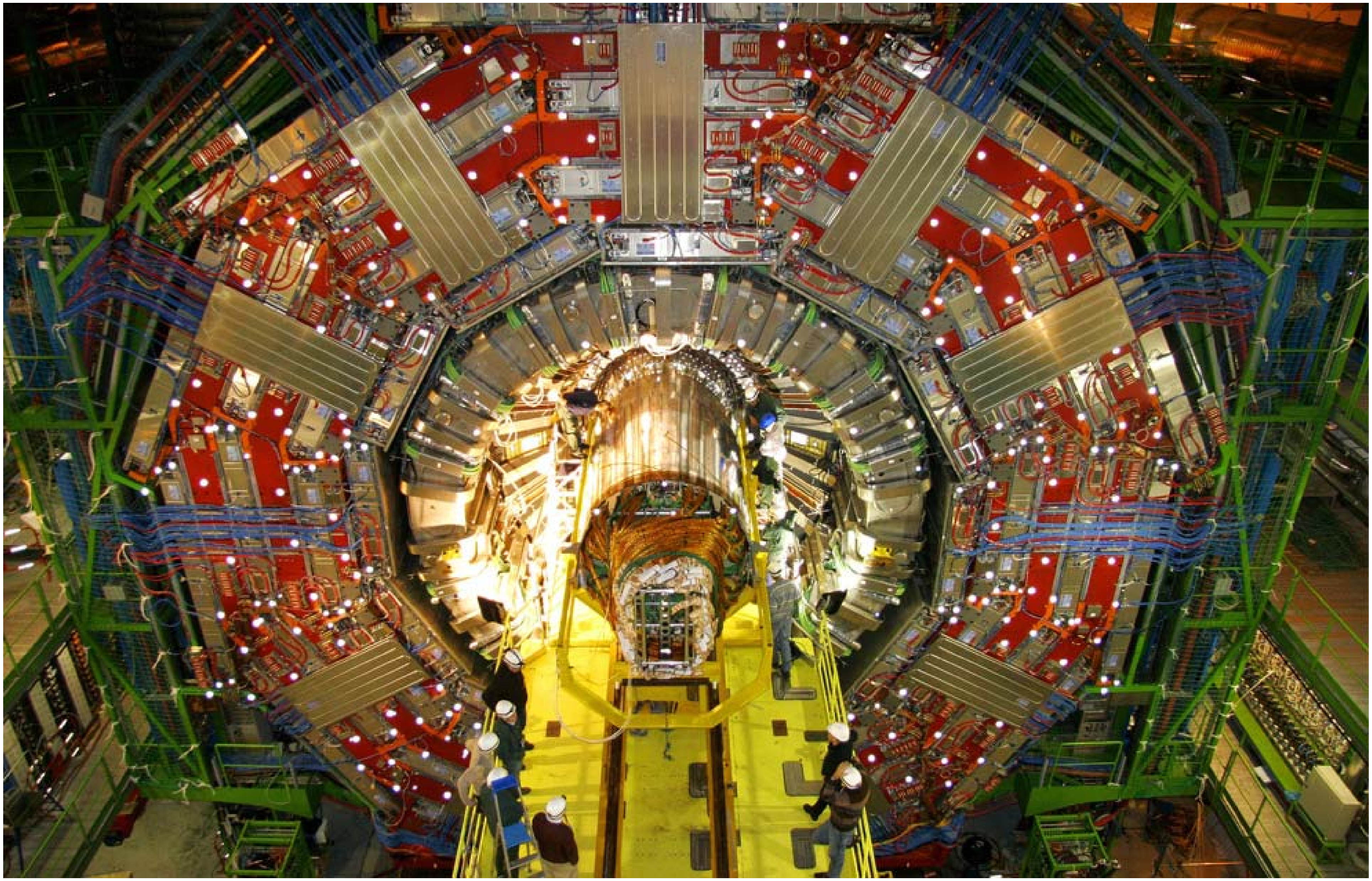
- Programming model for expressing distributed computations at a massive scale
- Execution framework for organizing and performing such computations
- Open-source implementation called Hadoop

Why large data?

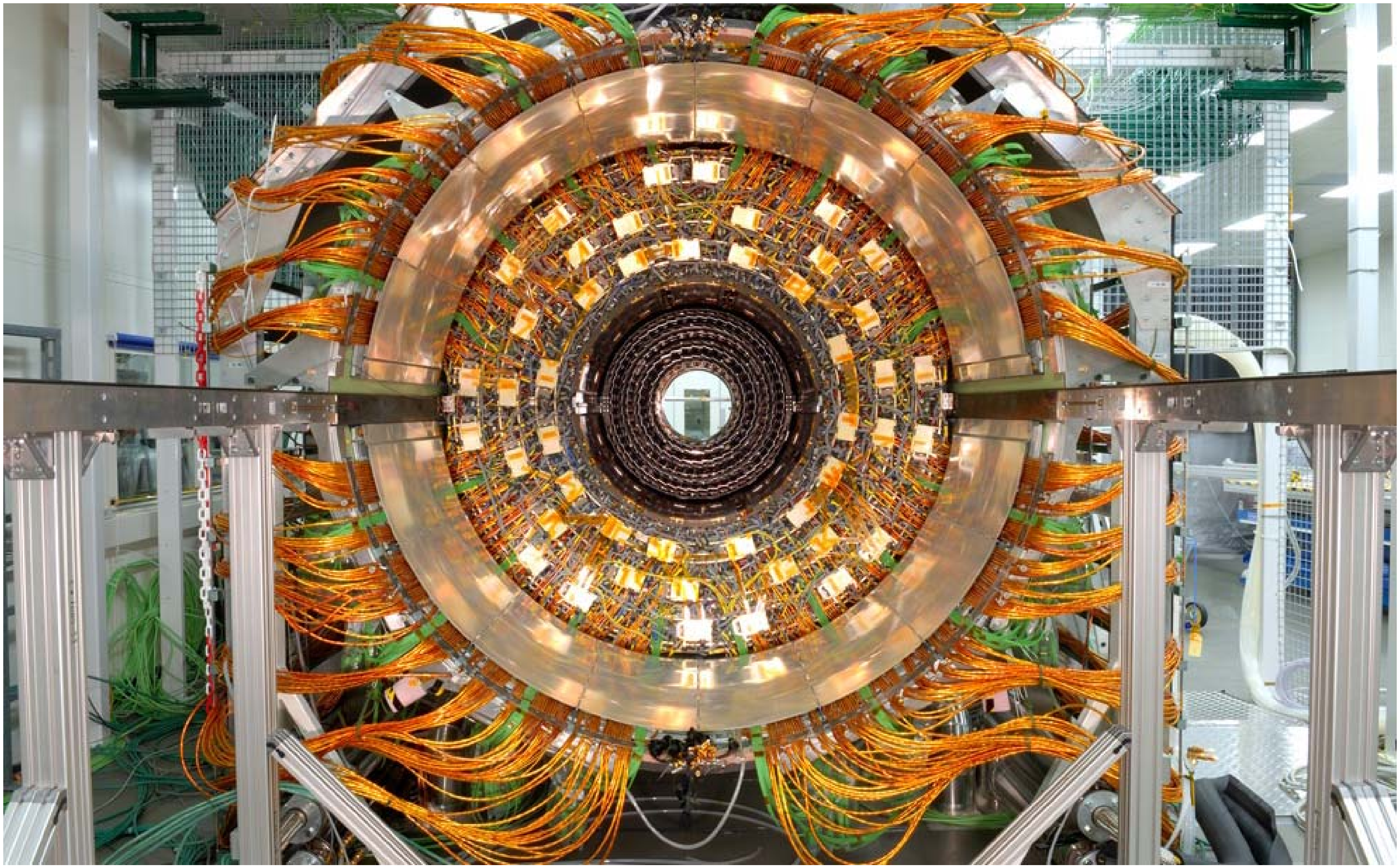
Source: Wikipedia (Everest)

How much data?

- Google processes 20 PB a day (2008)
- Wayback Machine has 3 PB + 100 TB/month (3/2009)
- Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
- eBay has 6.5 PB of user data + 50 TB/day (5/2009)
- CERN's LHC will generate 15 PB a year (??)



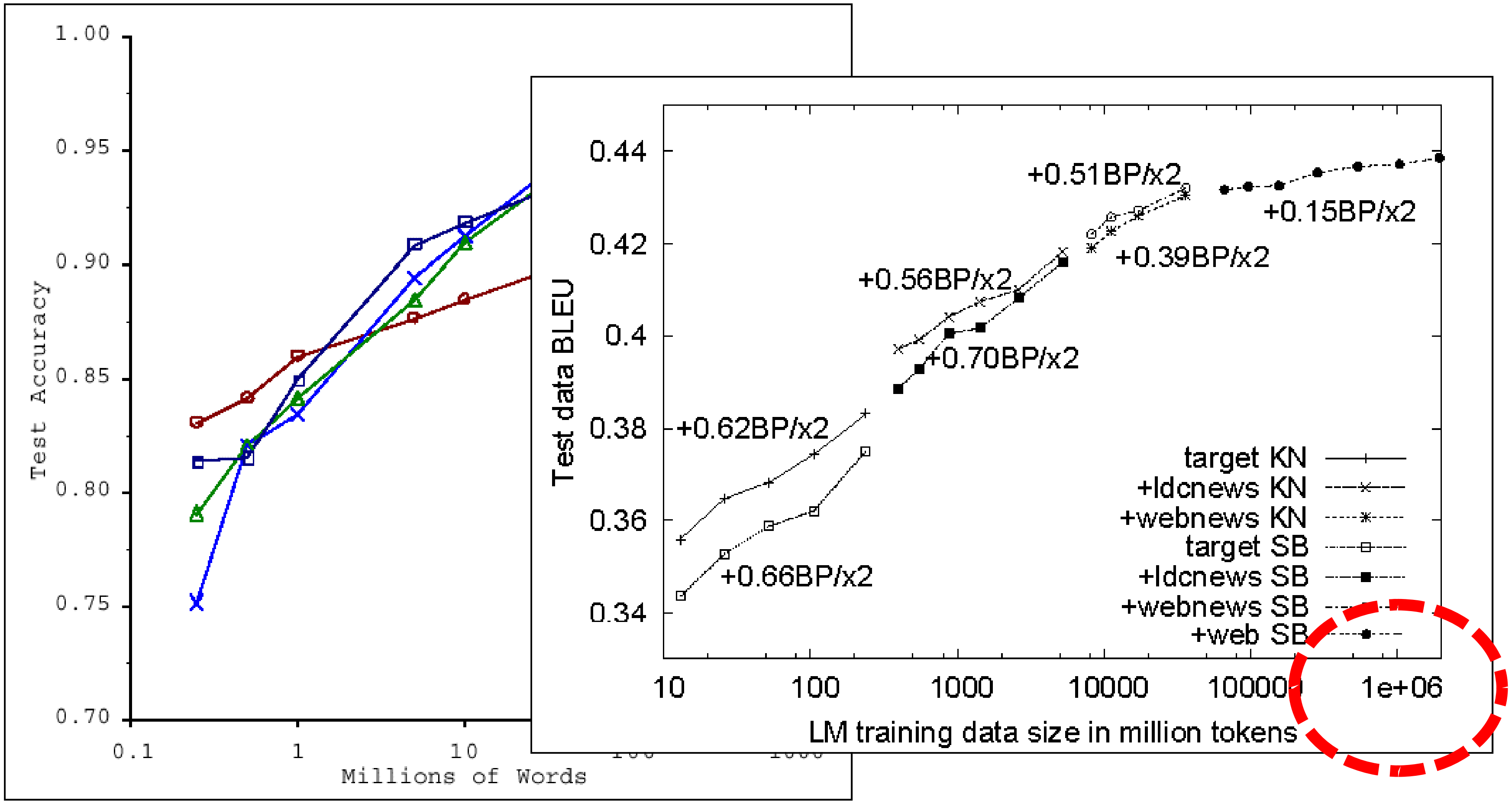
Maximilien Brice, © CERN



Maximilien Brice, © CERN

No data like more data!

s/knowledge/data/g;



How do we get here if we're not Google?

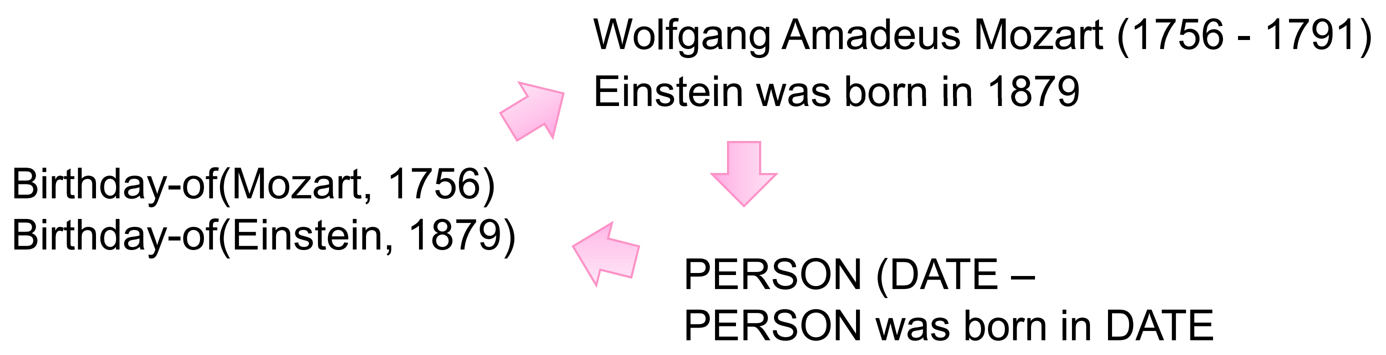
(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)

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


Wolfgang Amadeus Mozart (1756 - 1791)
Einstein was born in 1879
PERSON (DATE –
PERSON was born in DATE
Birthday-of(Mozart, 1756)
Birthday-of(Einstein, 1879)

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

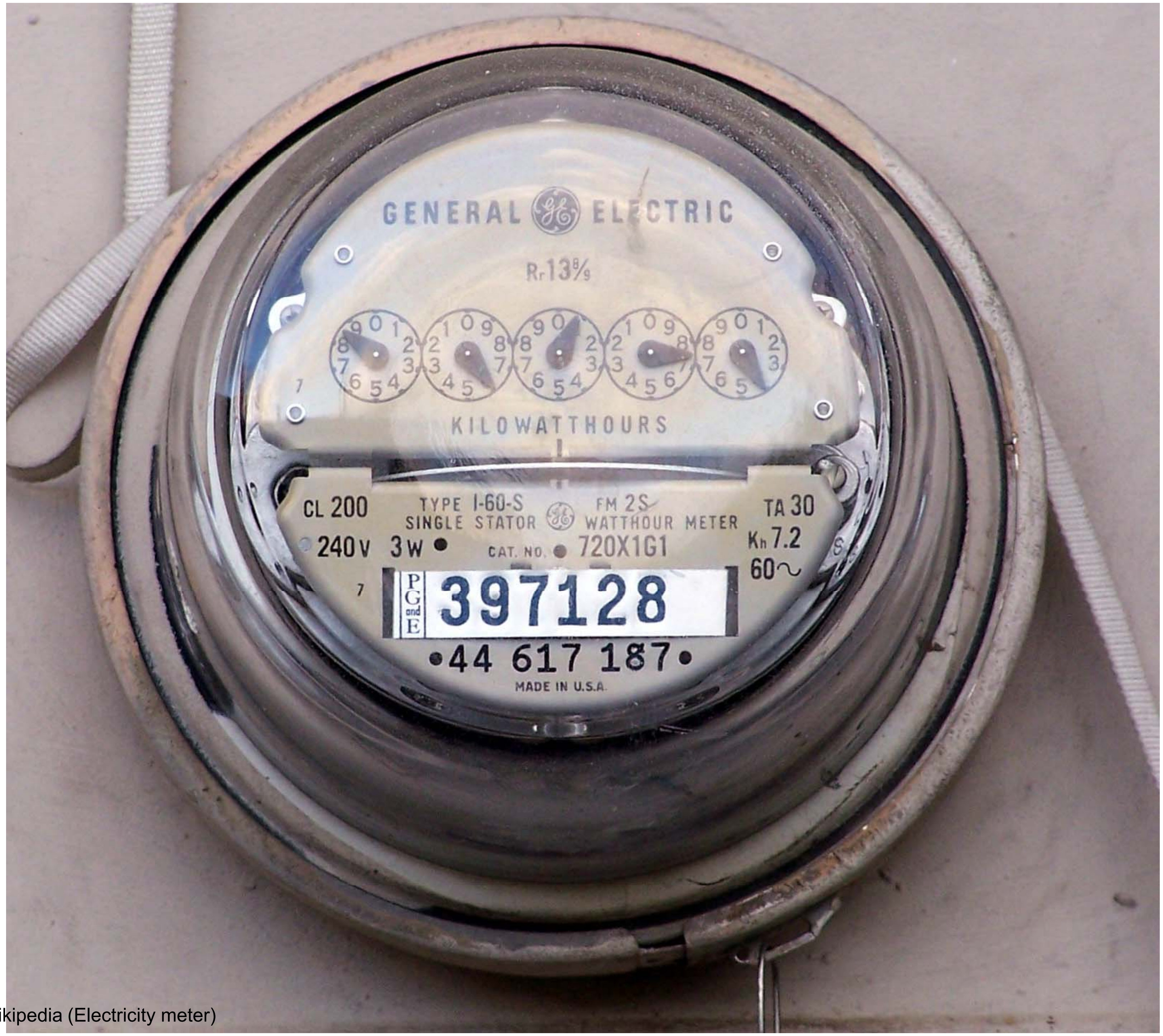
What is cloud computing?

The best thing since sliced bread?

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

Rebranding of web 2.0

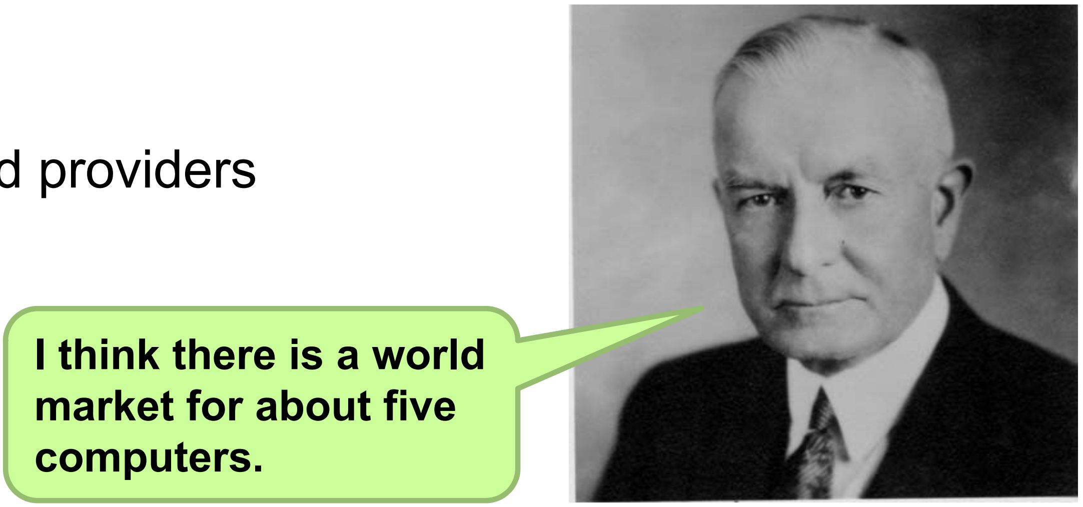
- Rich, interactive web applications
  - Clouds refer to the servers that run them
  - AJAX as the de facto standard (for better or worse)
  - Examples: Facebook, YouTube, Gmail, …
- “The network is the computer”: take two
  - User data is stored “in the clouds”
  - Rise of the netbook, smartphones,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



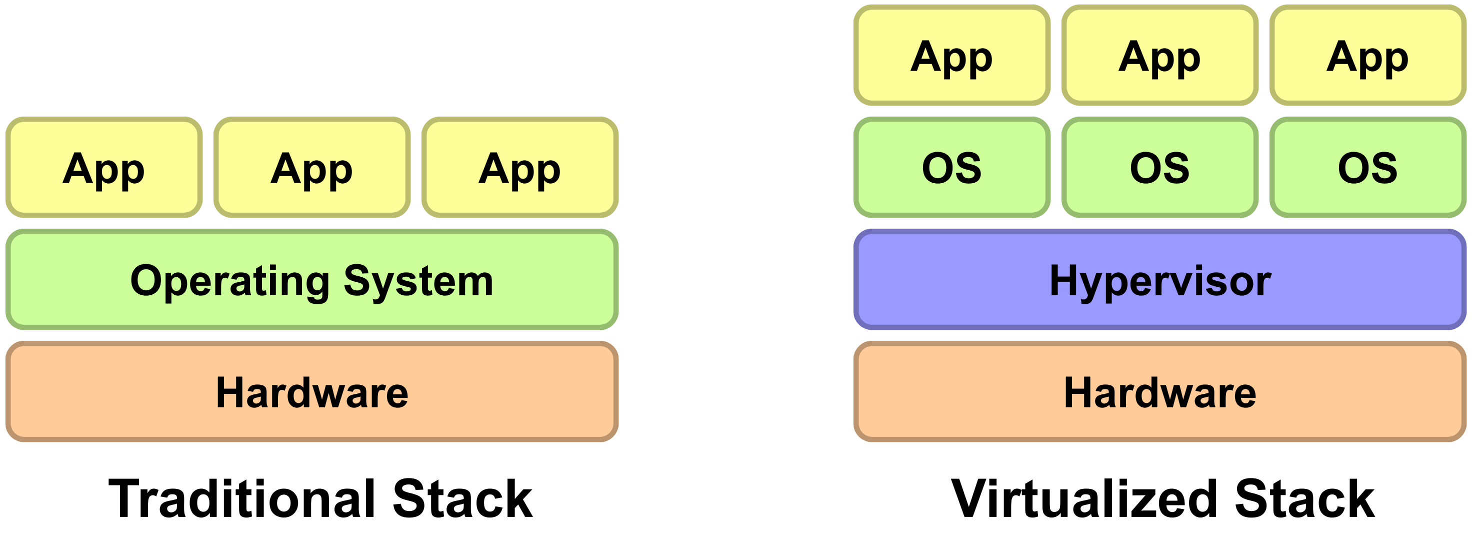
Source: Wikipedia (Electricity meter)

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



Enabling Technology: Virtualization



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 large-data problems
  - Lots of user-generated content
  - Even more user behavior data
  - Examples: Facebook friend suggestions, Google ad placement
  - Business intelligence: gather everything in a data warehouse and run analytics to generate insight
- Utility computing
  - Provision Hadoop clusters on-demand in the cloud
  - Lower barrier to entry for tackling large-data problem
  - Commoditization and democratization of large-data capabilities

Course Administrivia

Course Pre-requisites

- Strong Java programming
  - But this course is *not* about programming: we'll expect you to pick up Hadoop (quickly) along the way
  - Focus on "thinking at scale" and algorithm design
- Solid knowledge of
  - Probability and statistics
  - Computer architecture
- No previous experience necessary in
  - MapReduce
  - Parallel and distributed programming

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 not ready to do a lot of work
- If you're not open to thinking about computing in new ways
- If you can't cope with the uncertainty, unpredictability, etc. that comes with bleeding edge software

Otherwise, this will be a richly rewarding course!

Course components

- Textbooks
- Components of the final grade:
  - Assignments (important, but not worth much)
  - Midterm and final exams
  - Final project (of your choice, in groups of ~3)
  - Class participation
- 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

Cloud Resources

- Hadoop on your local machine
- Hadoop in a virtual machine on your local machine
- Hadoop in the clouds with Amazon EC2
- Hadoop on the Google/IBM cluster

Important Aside

- Usage agreement for EC2
- Usage agreement for Google/IBM cluster
- Stay tuned for more details over email…

Source: Wikipedia (Japanese rock garden)

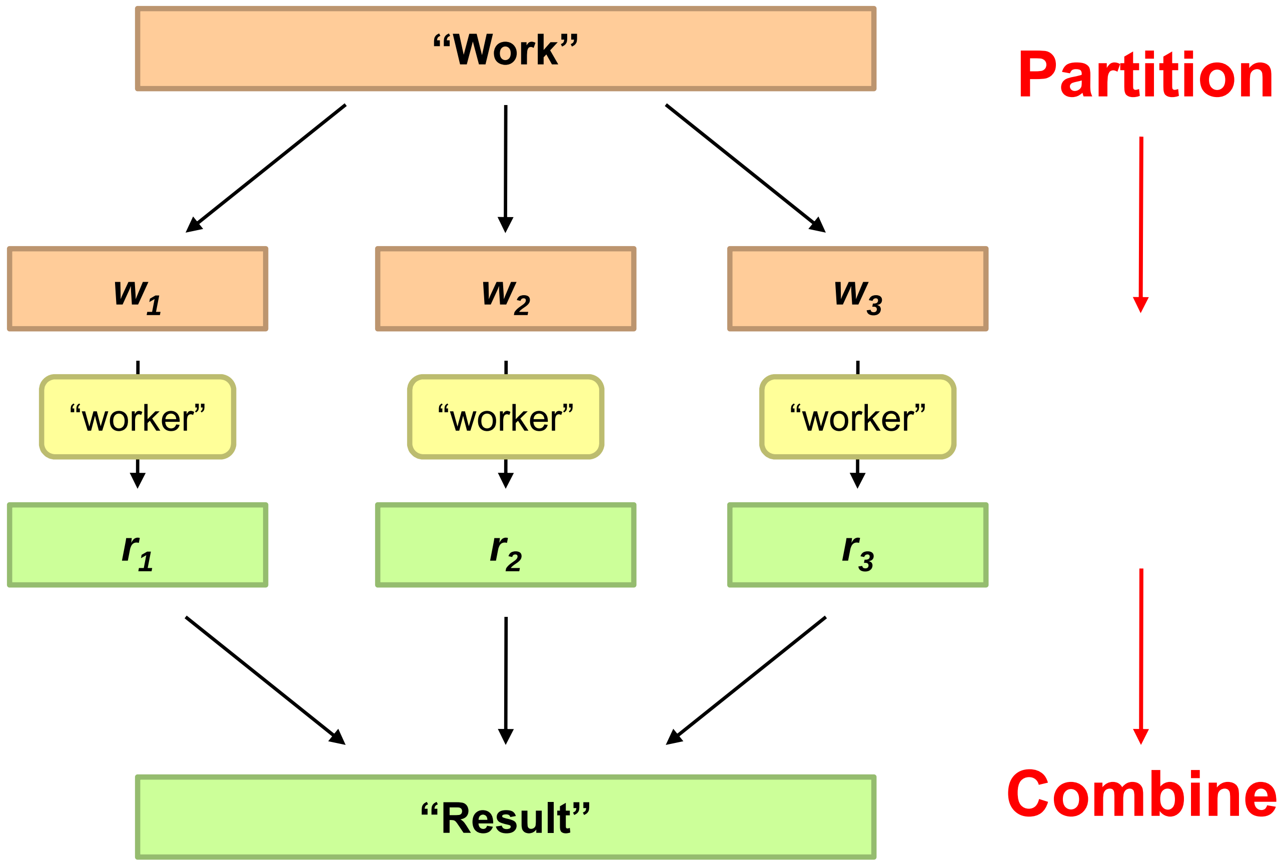
Hadoop Zen

- This is bleeding edge technology (= immature!)
  - 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

How do we scale up?

Source: Wikipedia (IBM Roadrunner)

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

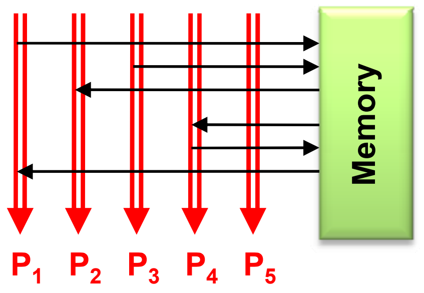
Source: Ricardo Guimarães Herrmann

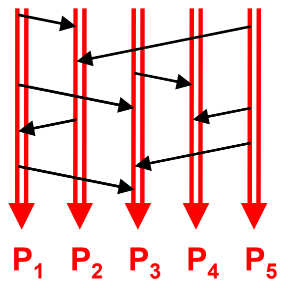
Managing Multiple Workers

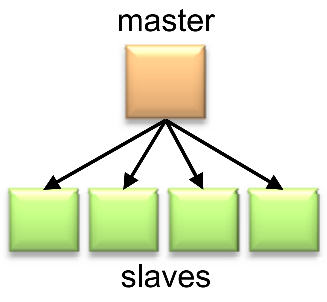
- Difficult because
  - We don’t know the order in which workers run
  - We don’t know when workers interrupt each other
  - We don’t know the order in which workers access shared data
- Thus, we need:
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still,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

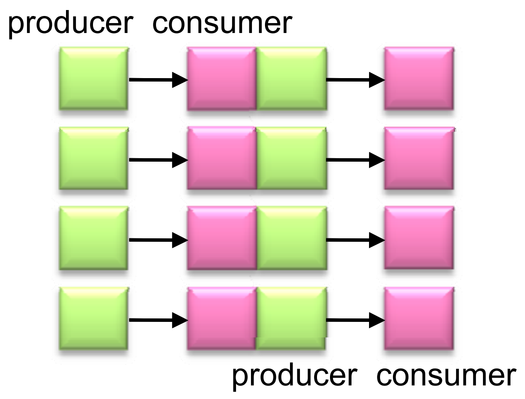
Current Tools

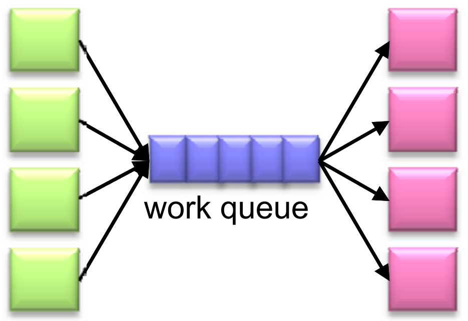
- Programming models
  - Shared memory (pthreads)
  - Message passing (MPI)
- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues







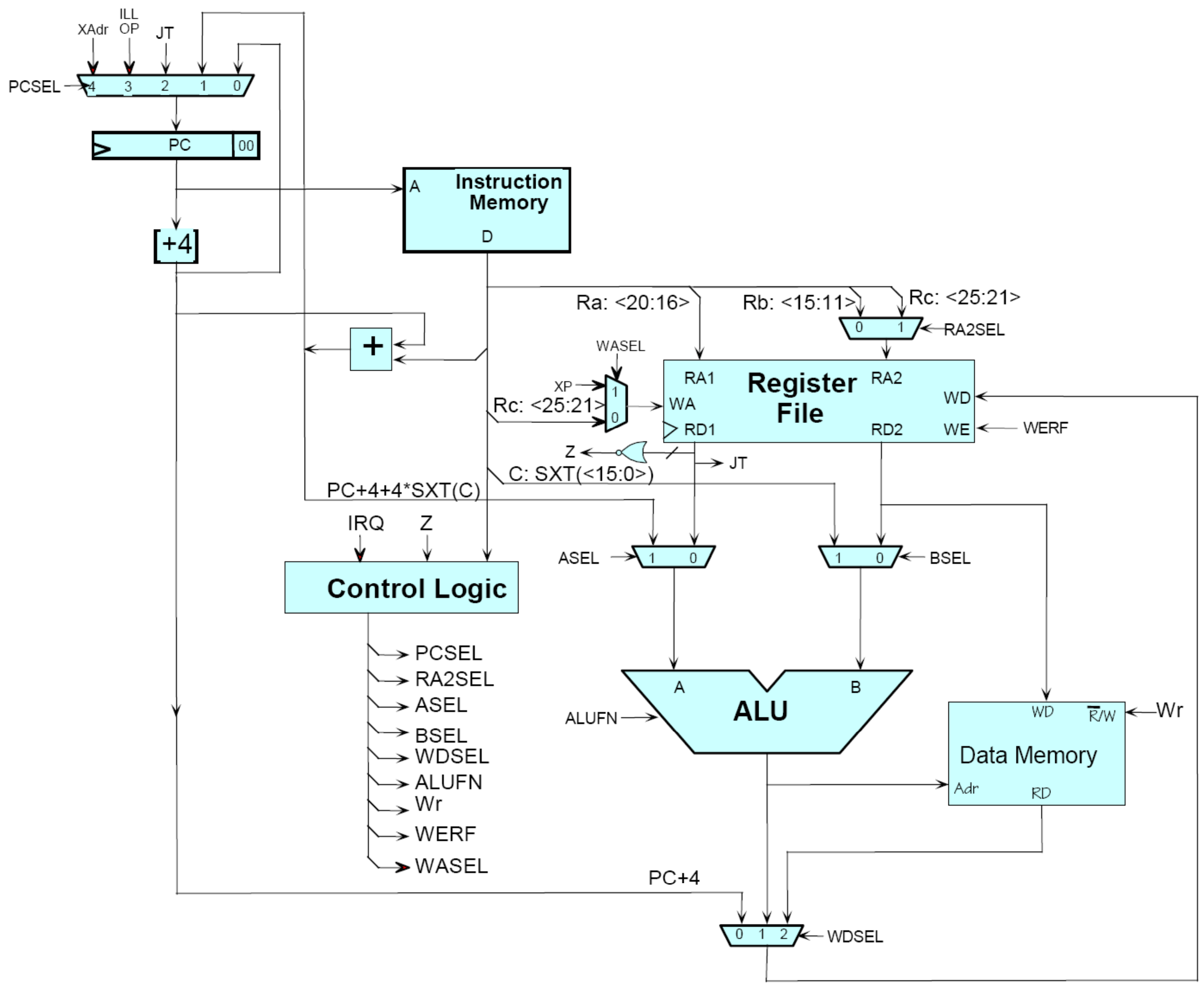




Where the rubber meets the road

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
  - At the scale of datacenters (even 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

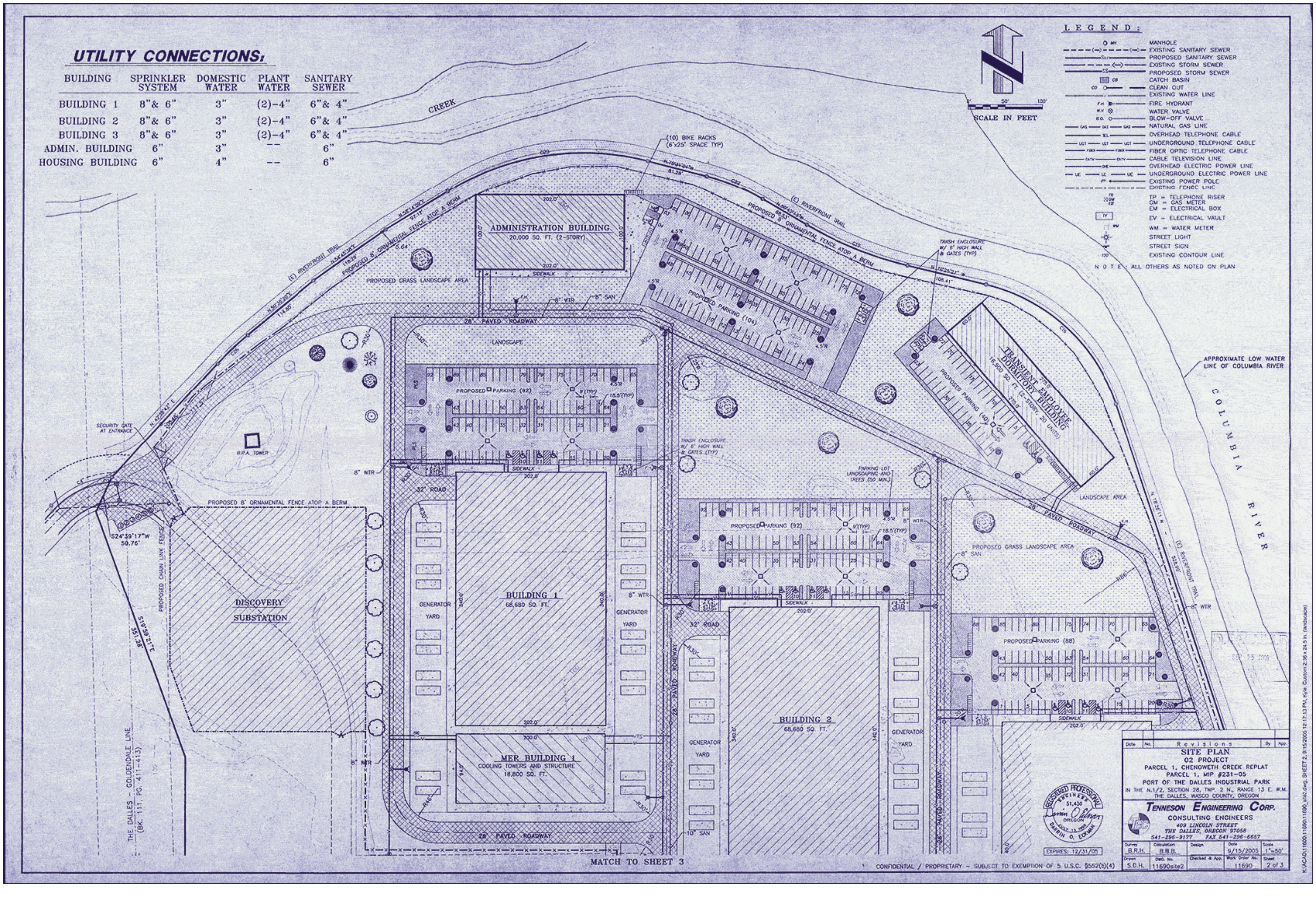
Source: Wikipedia (Flat Tire)



Source: MIT Open Courseware

PCSEL
PC
Instruction Memory
Register File
Control Logic
ALU
Data Memory
ASEL
BSEL
WDSEL
ALUFN
WERF
WASEL
RA2SEL
PC+4

Source: MIT Open Courseware


CREEK
SCALE IN FEET
(10) BIKE RACKS (6'x25' SPACE (TYP)
(E) RIVERFRONT TRAIL
PROPOSED 8' ORNAMENTAL FENCE ATOP A BERM
ADMINISTRATION BUILDING
20,000 SQ. FT. (2-STORY)
PROPOSED GRASS LANDSCAPE AREA
PROPOSED PARKING (104)
TRASH ENCLOSURE W/ 6' HIGH WALL & GATES (TYP)
TRANSIENT EMPLOYEE DORMITORY BUILDING
16,500 SQ. FT. (2-STORY, 20 UNITS)
PROPOSED PARKING (40)
APPROXIMATE LOW WATER LINE OF COLUMBIA RIVER
COLUMBIA RIVER
28' PAVED ROADWAY
LANDSCAPE
PROPOSED PARKING (92)
SECURITY GATE AT ENTRANCE
B.P.A. TOWER
PARKING LOT LANDSCAPING AND TREES (50 MIN.)
LANDSCAPE AREA
32' ROAD
DISCOVERY SUBSTATION
PROPOSED CHAIN LINK FENCE
S24°39'17"W 50.76'
GENERATOR YARD
BUILDING 1
68,680 SQ. FT.
MER BUILDING 1
COOLING TOWERS AND STRUCTURE
18,800 SQ. FT.
BUILDING 2
68,680 SQ. FT.
PROPOSED PARKING (88)
THE DALLES - GOLDENDALE LINE (BK. 111, PG. 411-413)
REGISTERED PROFESSIONAL ENGINEER 51,430 OREGON DARRIN O. ECKMAN
EXPIRES: 12/31/05
MATCH TO SHEET 3
CONFIDENTIAL / PROPRIETARY - SUBJECT TO EXEMPTION OF 5 U.S.C. §552(b)(4)

UTILITY CONNECTIONS:

BUILDING	SPRINKLER SYSTEM	DOMESTIC WATER	PLANT WATER	SANITARY SEWER
BUILDING 1	8"& 6"	3"	(2)-4"	6"& 4"
BUILDING 2	8"& 6"	3"	(2)-4"	6"& 4"
BUILDING 3	8"& 6"	3"	(2)-4"	6"& 4"
ADMIN. BUILDING	6"	3"	--	6"
HOUSING BUILDING	6"	4"	--	6"

LEGEND:

MH	MANHOLE
	EXISTING SANITARY SEWER
	PROPOSED SANITARY SEWER
	EXISTING STORM SEWER
	PROPOSED STORM SEWER
CB	CATCH BASIN
CO	CLEAN OUT
	EXISTING WATER LINE
F.H.	FIRE HYDRANT
W.V.	WATER VALVE
B.O.	BLOW-OFF VALVE
GAS	NATURAL GAS LINE
TEL	OVERHEAD TELEPHONE CABLE
UGT	UNDERGROUND TELEPHONE CABLE
FIBER	FIBER OPTIC TELEPHONE CABLE
CATV	CABLE TELEVISION LINE
OHE	OVERHEAD ELECTRIC POWER LINE
UE	UNDERGROUND ELECTRIC POWER LINE
PP	EXISTING POWER POLE
	EXISTING FENCE LINE

TR GM EB	TP = TELEPHONE RISER
	GM = GAS METER
	EM = ELECTRICAL BOX

EV	EV = ELECTRICAL VAULT
WM	WM = WATER METER
	STREET LIGHT
	STREET SIGN
100	EXISTING CONTOUR LINE

NOTE: ALL OTHERS AS NOTED ON PLAN

Date	No.	Revisions	By	App.

SITE PLAN
02 PROJECT
PARCEL 1, CHENOWETH CREEK REPLAT
PARCEL 1, MIP #231-05
PORT OF THE DALLES INDUSTRIAL PARK
IN THE N.1/2, SECTION 28, TWP. 2 N., RANGE 13 E. W.M.
THE DALLES, WASCO COUNTY, OREGON

TENNESON ENGINEERING CORP.
CONSULTING ENGINEERS
409 LINCOLN STREET
THE DALLES, OREGON 97058
541-296-9177 FAX 541-296-6657

Survey	Calculation	Design	Date	Scale
B.R.H.	B.B.B.		9/15/2005	1"=50'
Drawn	**Dwg. No.**	**Checked & App.**	**Work Order No.**	**Sheet**
S.D.H.	11690site2		11690	2 of 3

Source: Harper's (Feb, 2008)

What’s the point?

- It’s all about the right level of abstraction
  - The von Neumann architecture has served us well, but is no longer appropriate for the multi-core/cluster environment
- 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

MapReduce

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

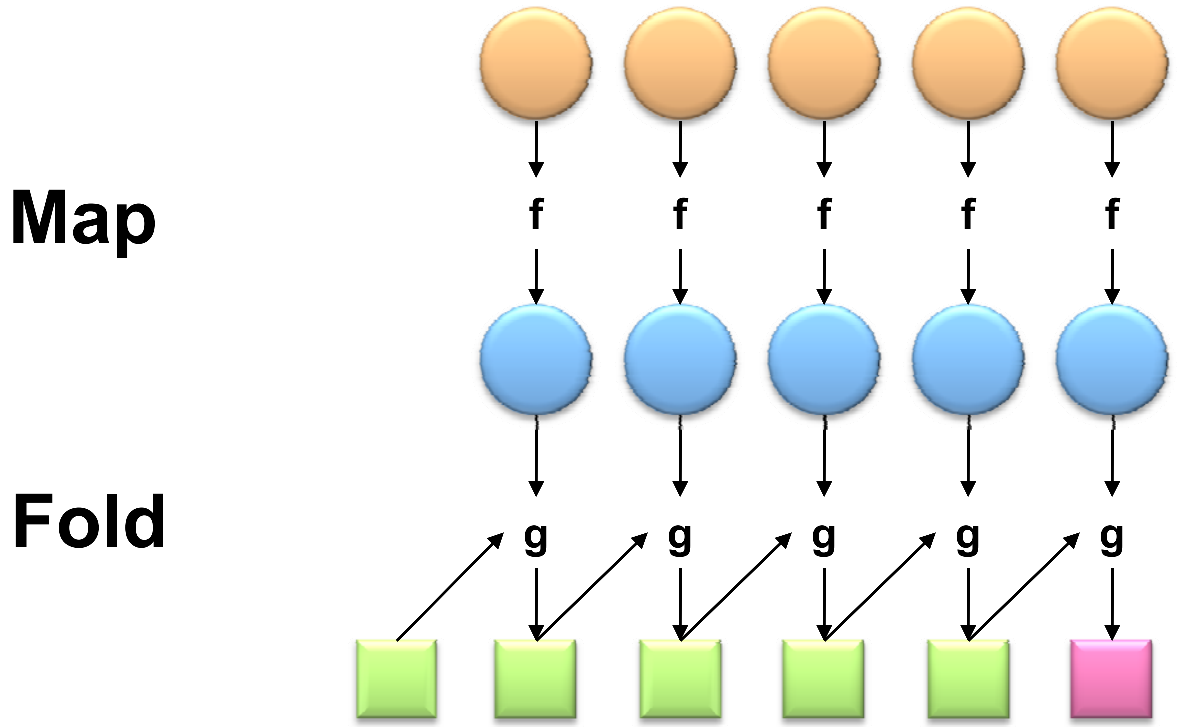
Reduce

- Generate final output

Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

Roots in Functional Programming



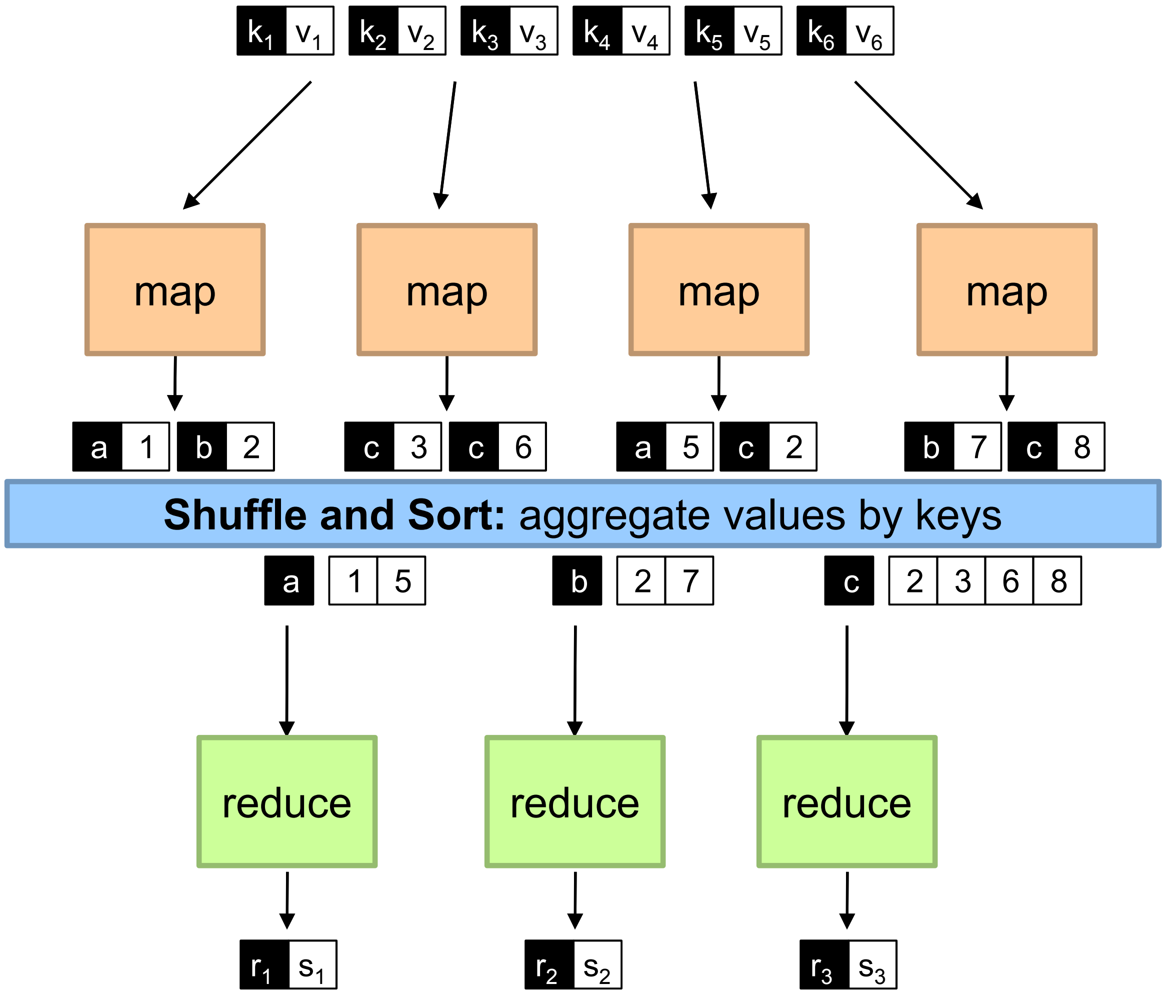
MapReduce

- 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…


k_1 v_1
k_2 v_2
k_3 v_3
k_4 v_4
k_5 v_5
k_6 v_6
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
r_1 s_1
r_2 s_2
r_3 s_3

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'>*

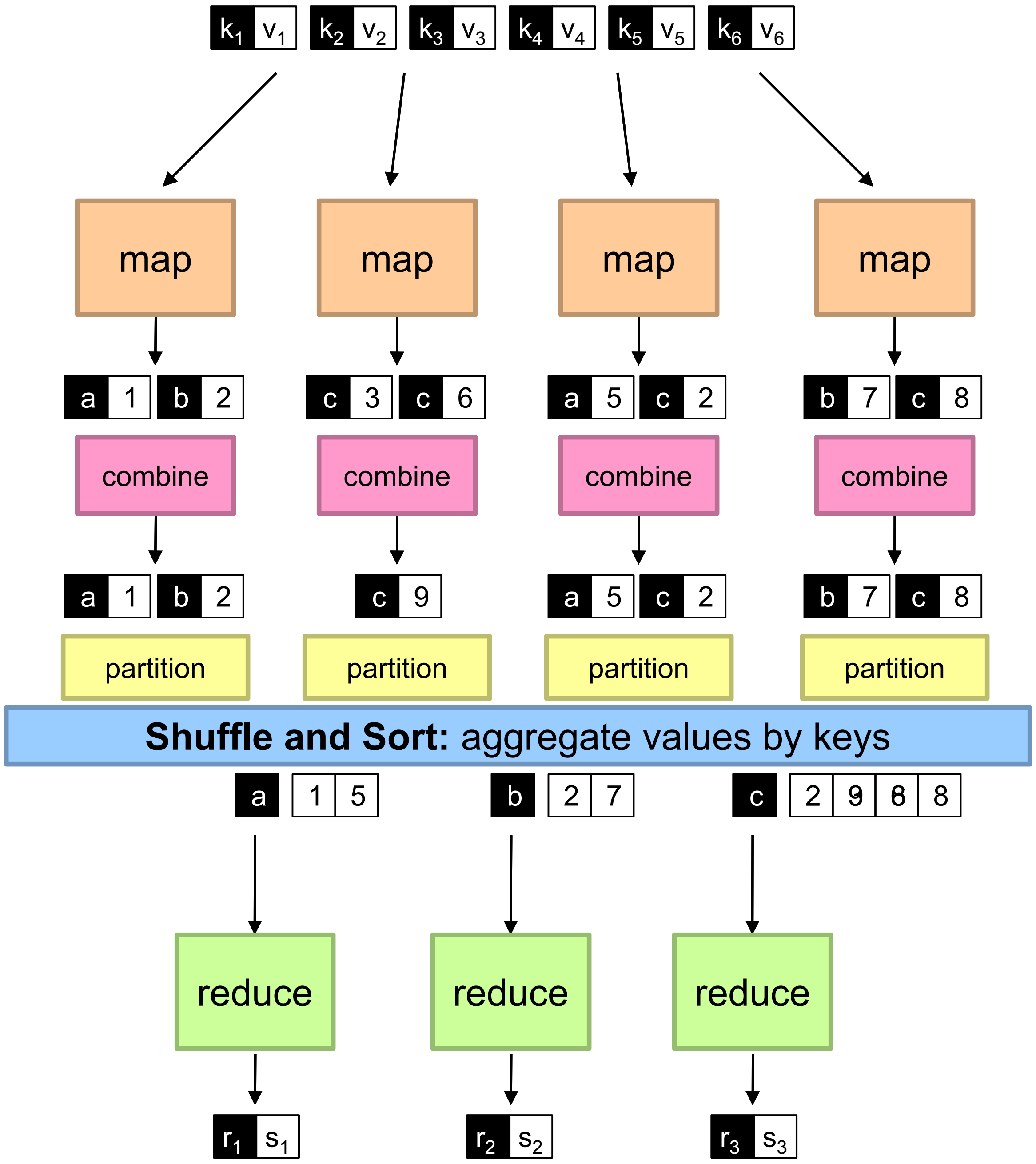
  - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers also specify:

  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


k_1 v_1
k_2 v_2
k_3 v_3
k_4 v_4
k_5 v_5
k_6 v_6
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 8 8
reduce
reduce
reduce
r_1 s_1
r_2 s_2
r_3 s_3

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);
```

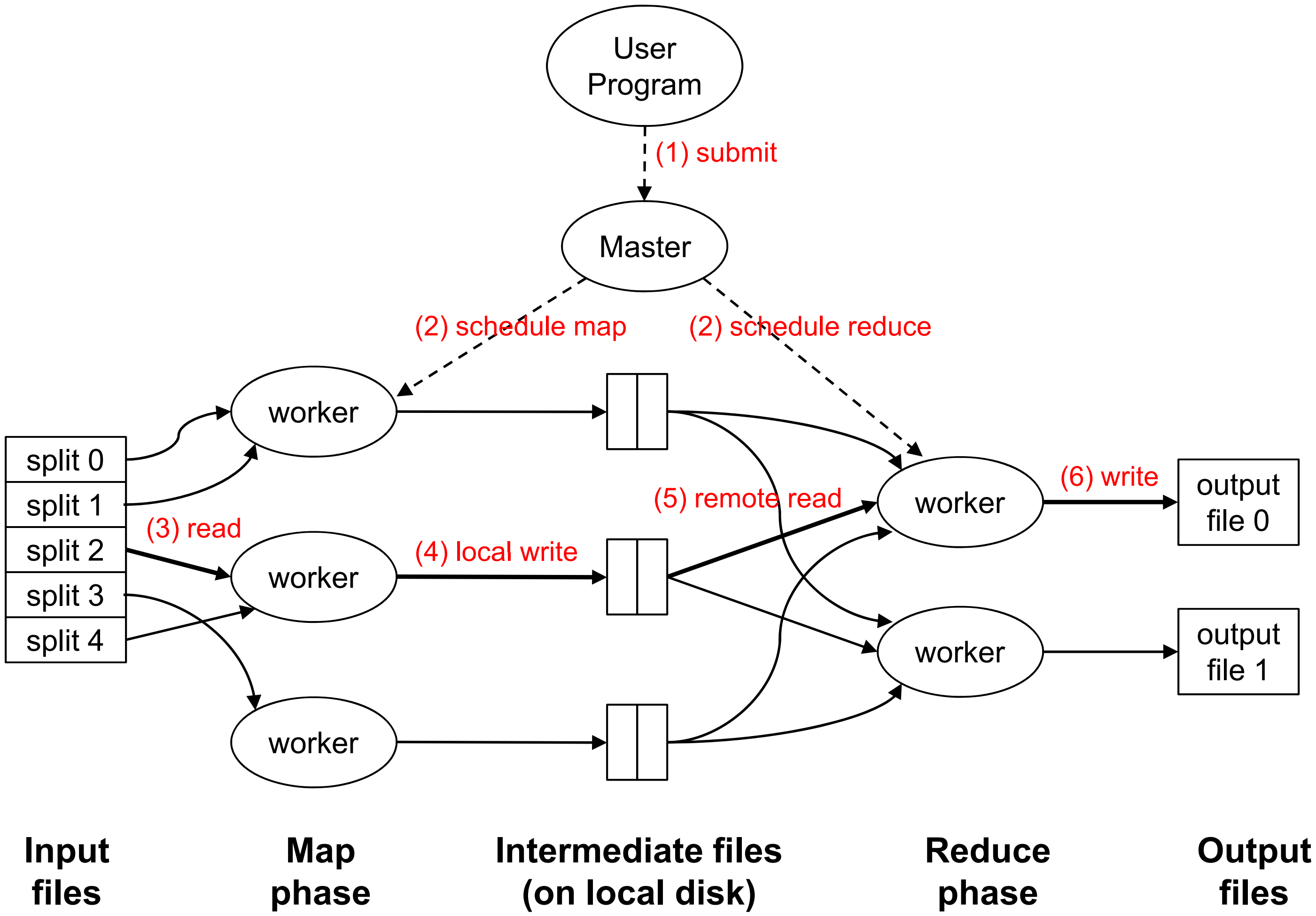
MapReduce can refer to…

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

Usage is usually clear from context!

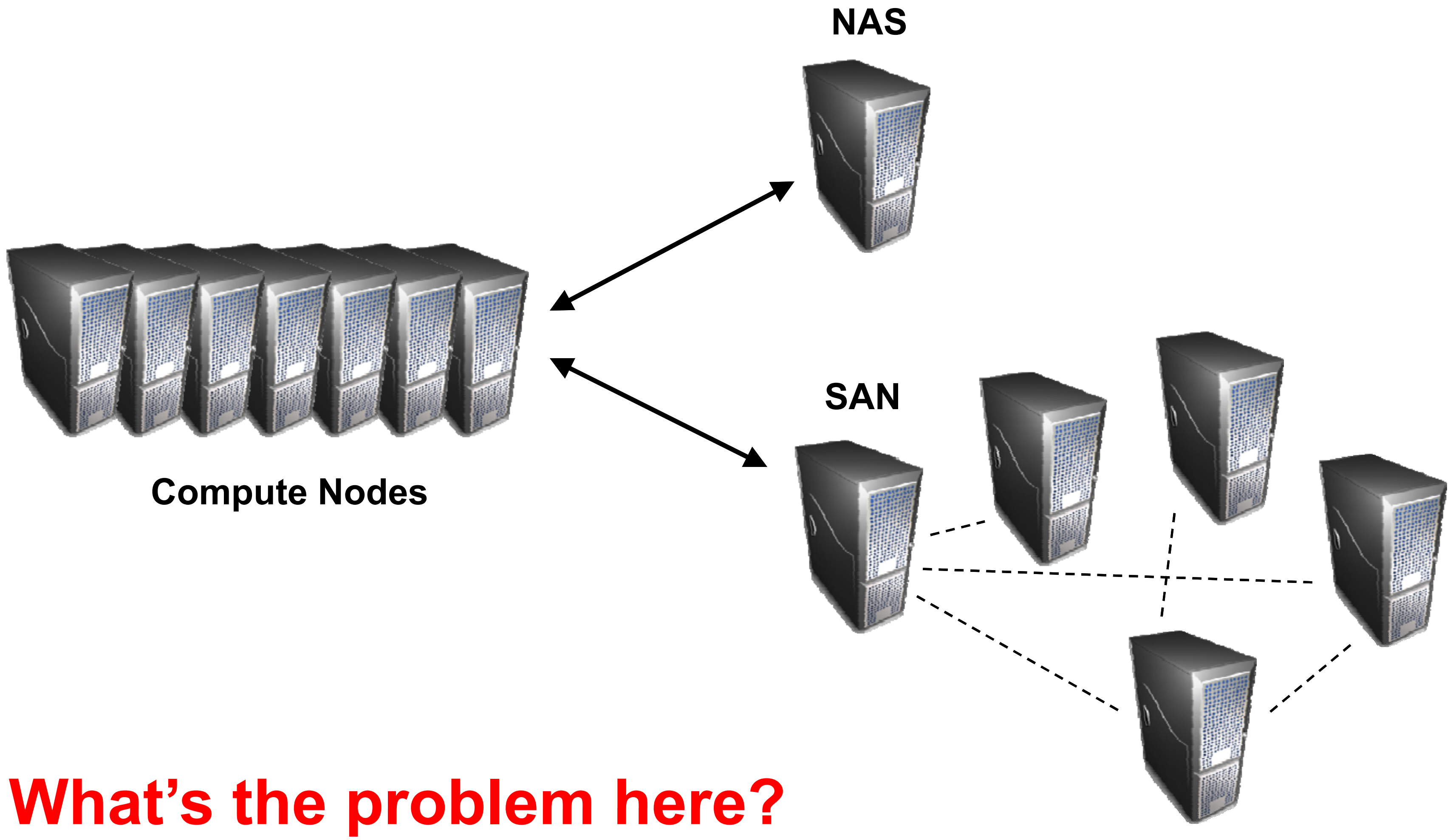
MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - 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?

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 slides adapted from material by (Ghemawat et al., SOSP 2003)

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)

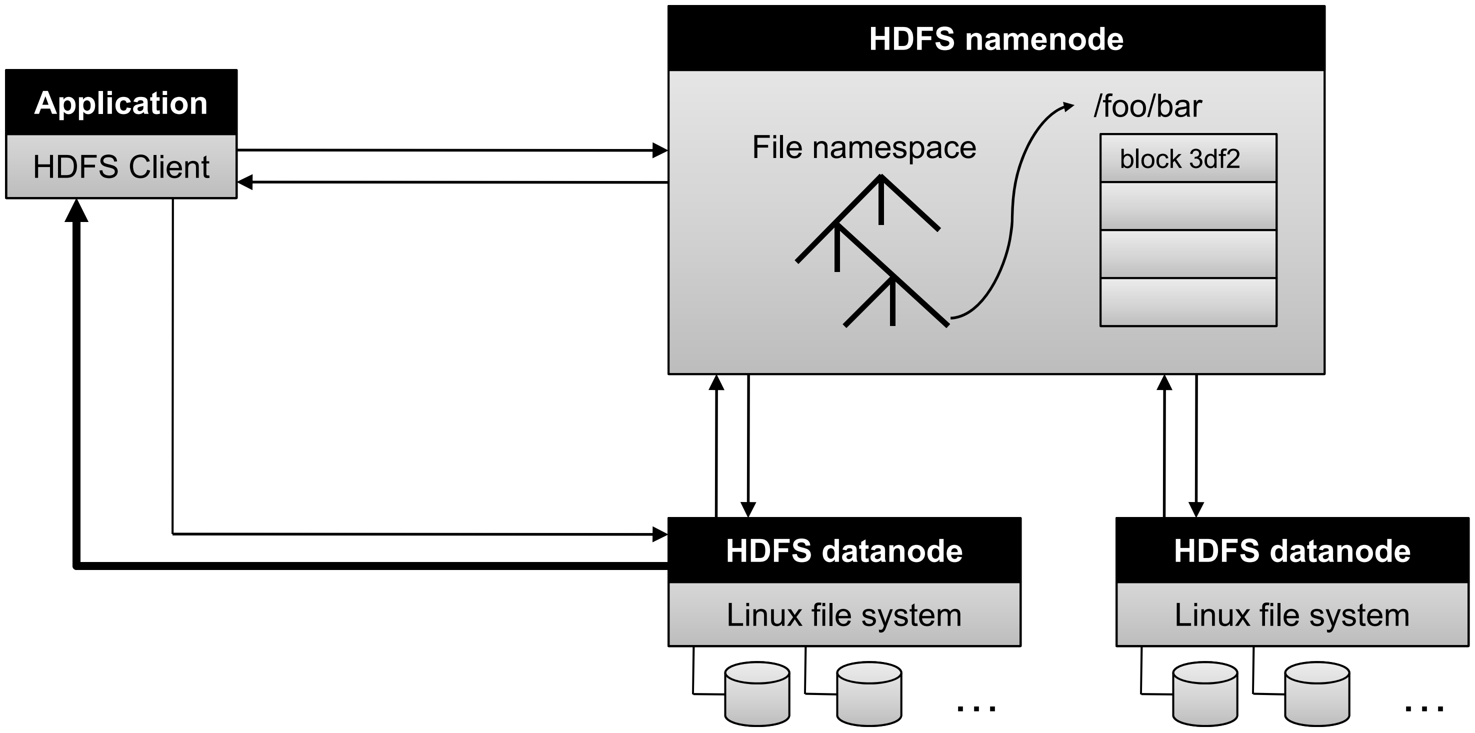
HDFS = GFS clone (same basic ideas)

From GFS to HDFS

- Terminology differences:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Functional differences:
  - No file appends in HDFS (planned feature)
  - HDFS performance is (likely) slower

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

HDFS Architecture

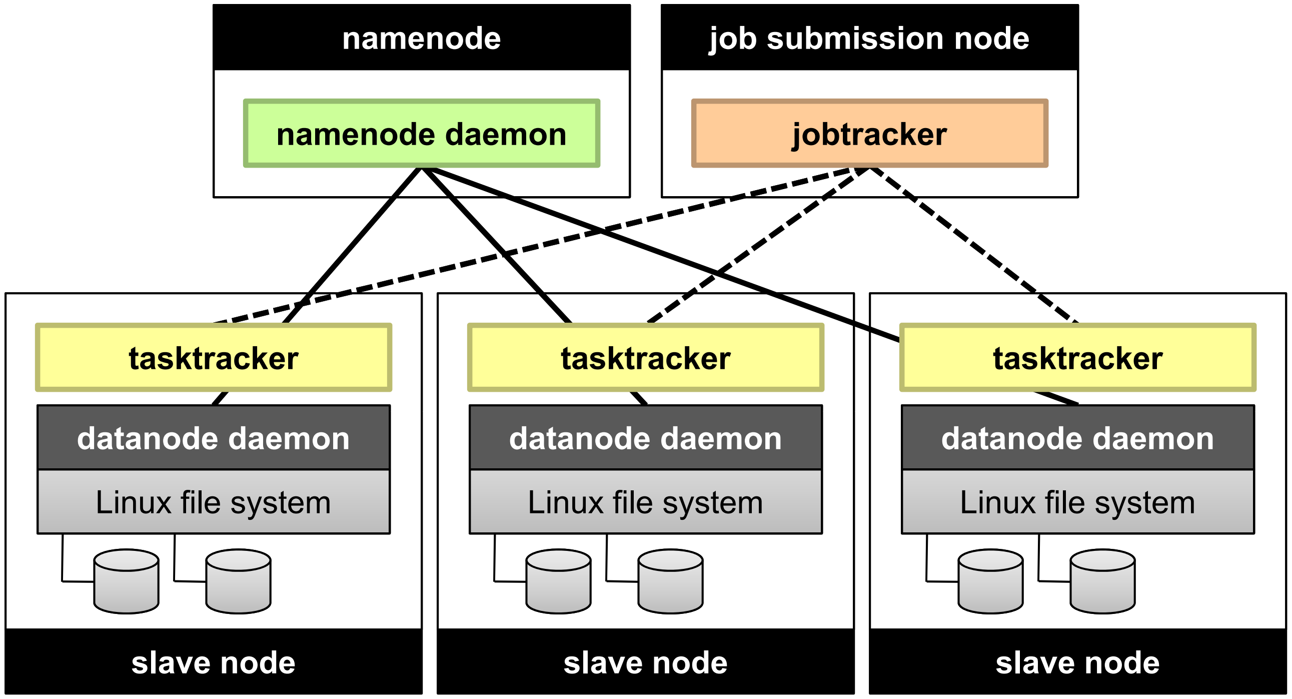


Adapted from (Ghemawat et al., SOSP 2003)

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…



Recap

- Why large data?
- Cloud computing and MapReduce
- Large-data processing: “big ideas”
- What is MapReduce?
- Importance of the underlying distributed file system

Questions?

Photo credit: Jimmy Lin