Introduction to Big Data, MAP/REDUCE, Hadoop, HDFS

D. Tsoumakos
Data Analysis and Processing
2020

Department of Informatics
Ionian University

Material adapted from slides by **Jimmy Lin, The iSchool – University of Maryland**
And **from www.cloudcomputingchina.com**

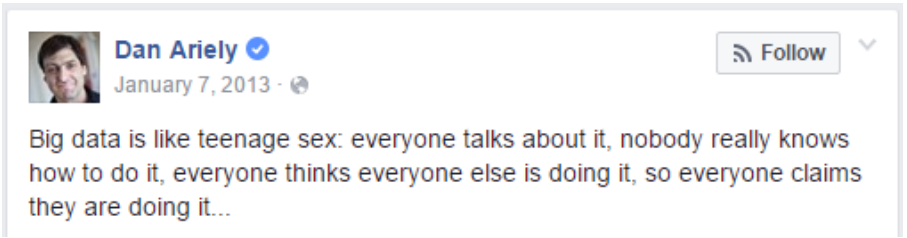
IONIAN UNIVERSITY
DEPARTMENT OF INFORMATICS

The Big Data era

What is Big Data?

- Big data is like teenage sex:
  - everyone talks about it
  - nobody really knows how to do it
  - everyone thinks everyone else is doing it
  - so everyone claims they are doing it...

--Dan Ariely, Professor at Duke University



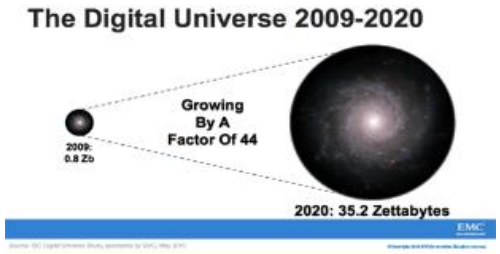
Big Data Definition

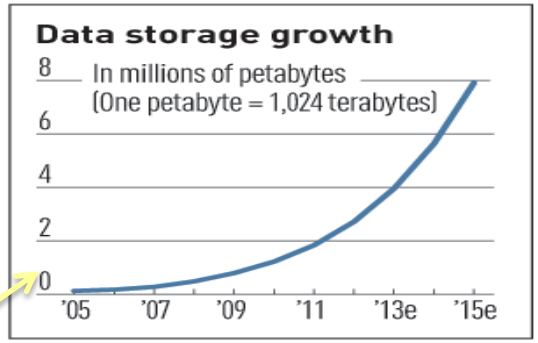
- No single standard definition…

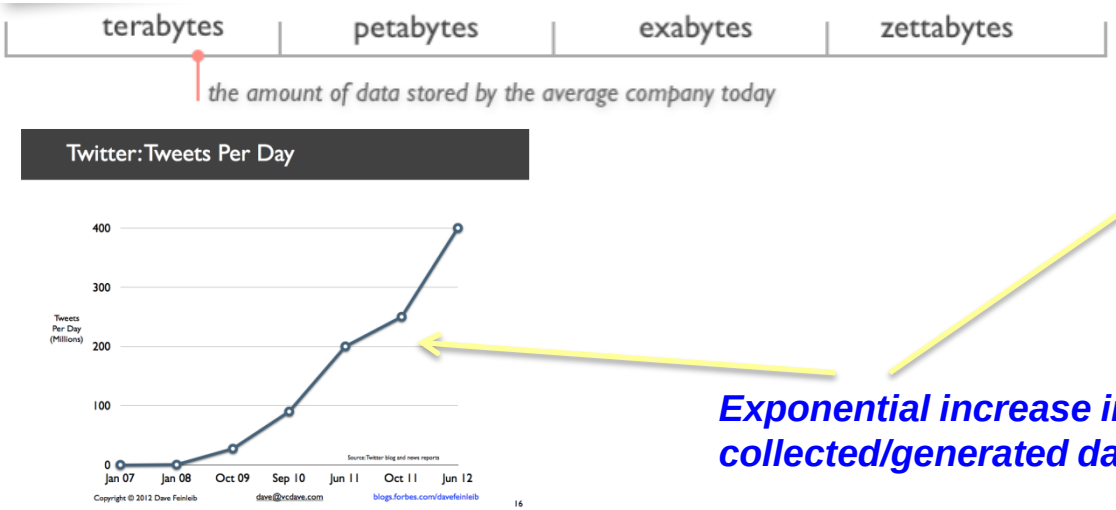
"***Big Data***" is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

Characteristics of Big Data:
1-Scale (Volume)

- **Data Volume**
  - 44x increase from 2009 2020
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially



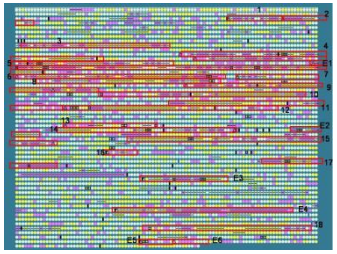


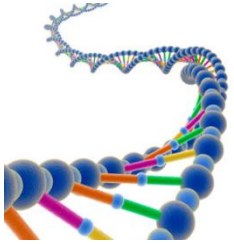


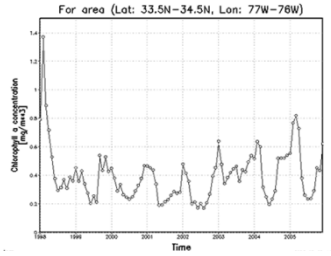
Exponential increase in collected/generated data

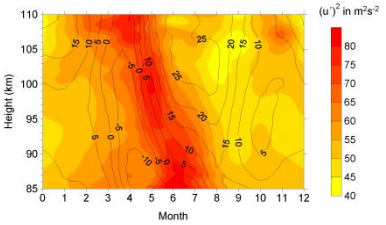
Characteristics of Big Data:
2-Complexity (Varity)

- Various formats, types, and structures
- Text, numerical, images, audio, video, sequences, time series, social media data, multi-dim arrays, etc…
- Static data vs. streaming data
- A single application can be generating/collecting many types of data







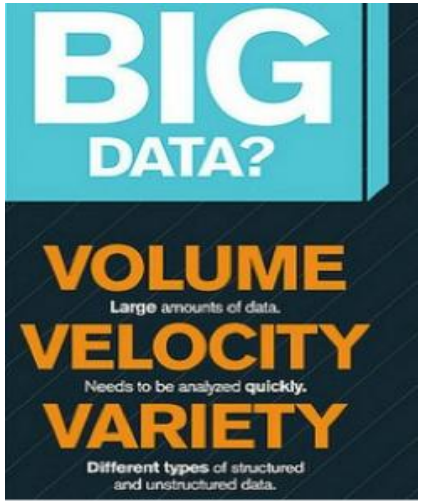


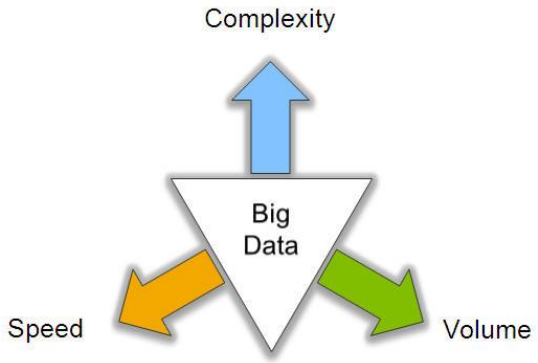
To extract knowledge➔ all these types of data need to linked together

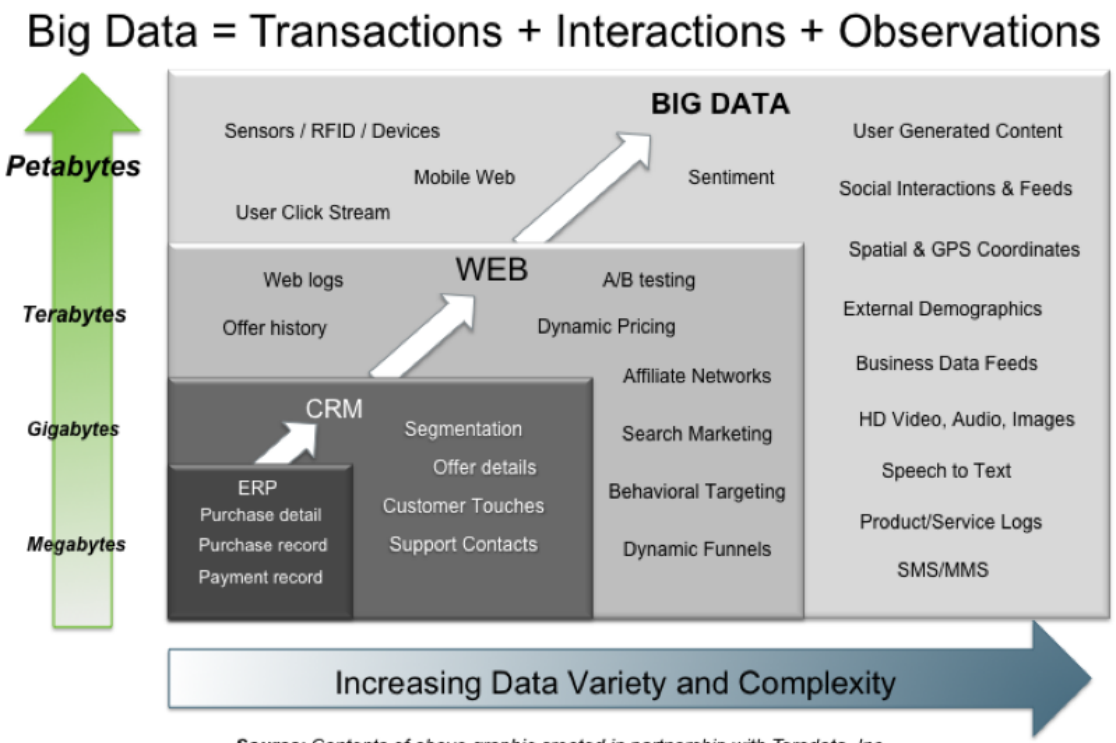
Characteristics of Big Data:
3-Speed (Velocity)

- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like ➔ send promotions right now for store next to you
  - **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction

Big Data: 3V's







Source: Contents of above graphic created in partnership with Teradata, Inc.

Extended Big Data Characteristics: 6V

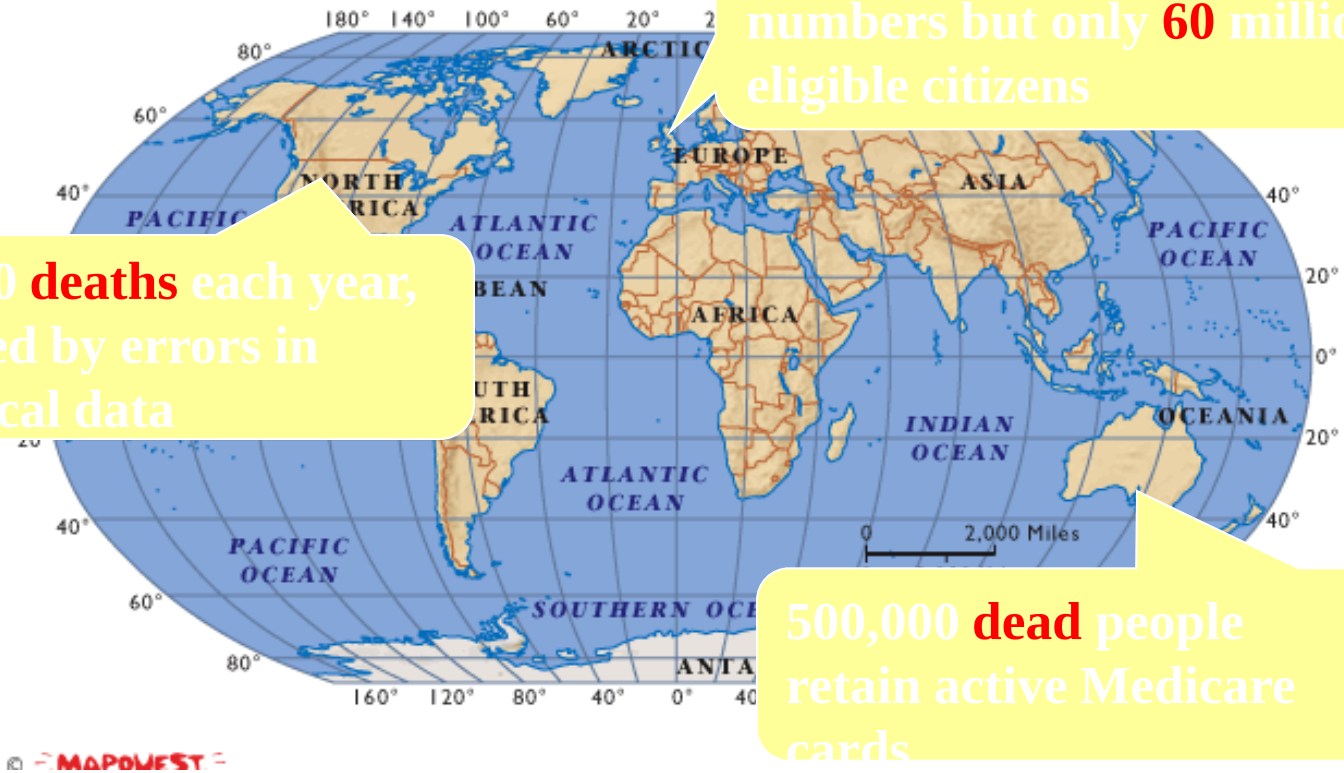
- Volume: In a big data environment, the amounts of data collected and processed are much larger than those stored in typical relational databases.
- Variety: Big data consists of a rich variety of data types.
- Velocity: Big data arrives to the organization at high speeds and from multiple sources simultaneously.

- Veracity: Data quality issues are particularly challenging in a big data context.
- Visibility/Visualization: After big data being processed, we need a way of presenting the data in a manner that's readable and accessible.
- Value: Ultimately, big data is meaningless if it does not provide value toward some meaningful goal.

Veracity (Quality & Trust)

- *Data = quantity + quality*
- When we talk about big data, we typically mean its quantity:
  - What capacity of a system provides to cope with the sheer size of the data?
  - Is a query feasible on big data within our available resources?
  - How can we make our queries tractable on big data?
  - . . .
- Can we trust the answers to our queries?
  - Dirty data routinely lead to misleading financial reports, strategic business planning decision $\Rightarrow$ loss of revenue, credibility and customers, disastrous consequences
- *The study of data quality is as important as data quantity*

Data in real-life is often dirty



Visibility/Visualization

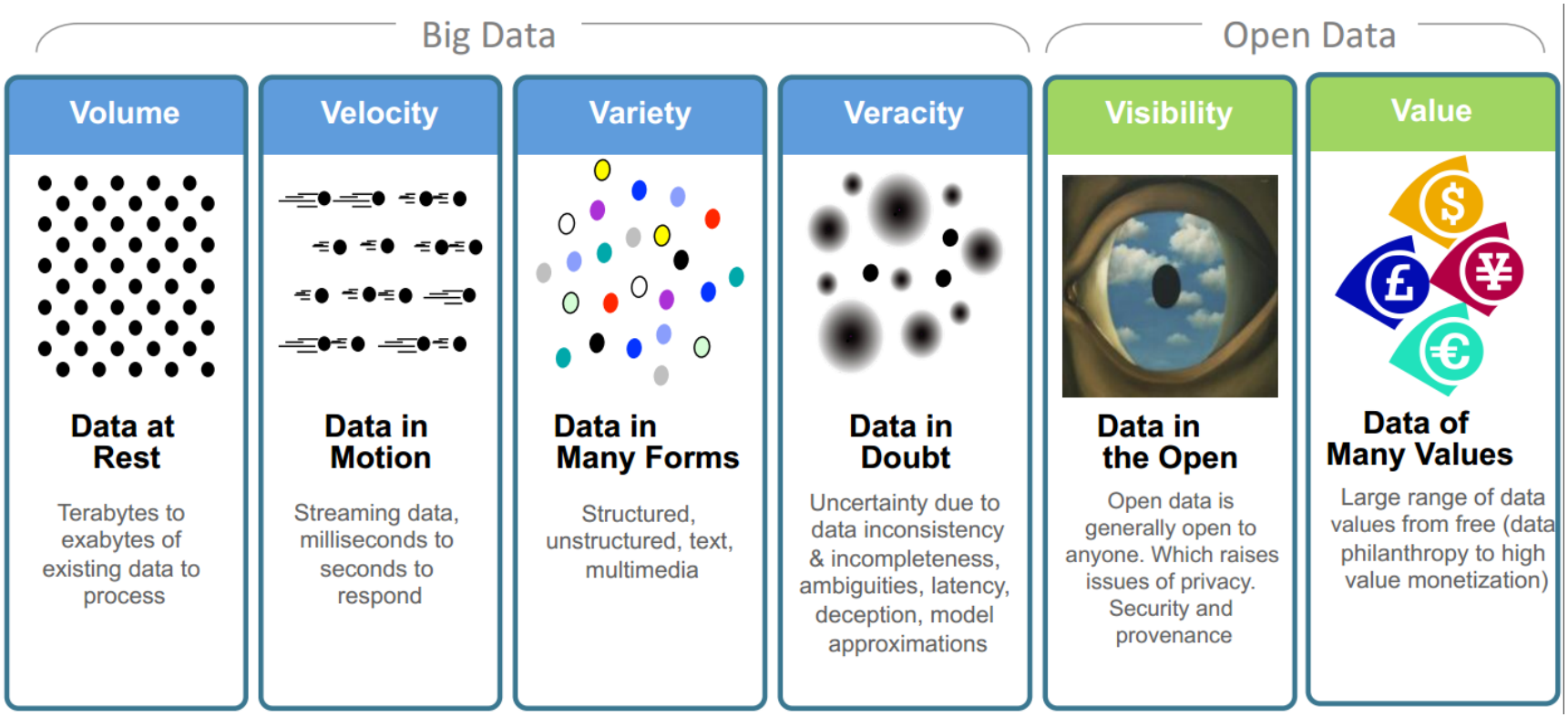
- Visible to the process of big data management
- Big Data – visibility = Black Hole?

- Big data visualization tools:

Value

- Big data is meaningless if it does not provide value toward some meaningful goal

Big Data: 6V in Summary

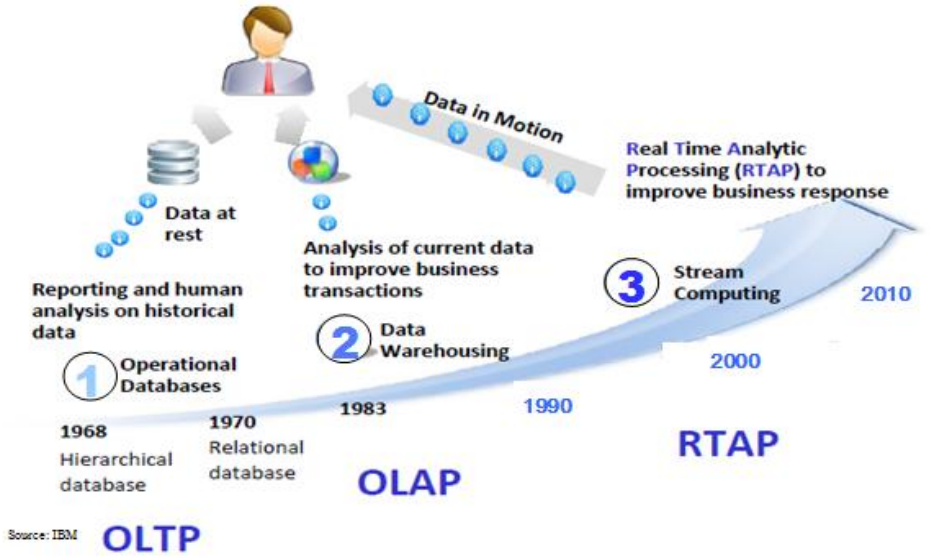


Transforming Energy and Utilities through Big Data & Analytics. By Anders Quitzau@IBM

Other V's

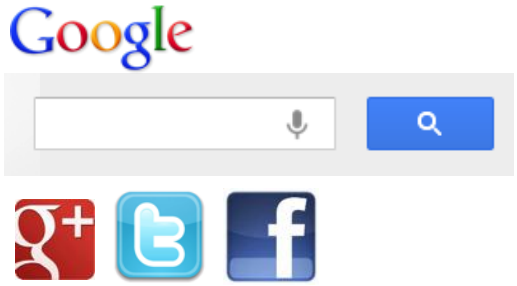
- Variability
  - Variability refers to data whose meaning is constantly changing. This is particularly the case when gathering data relies on language processing.
- Viscosity
  - This term is sometimes used to describe the latency or lag time in the data relative to the event being described. We found that this is just as easily understood as an element of Velocity.
- Virality
  - Defined by some users as the rate at which the data spreads; how often it is picked up and repeated by other users or events.
- Volatility
  - Big data volatility refers to how long is data valid and how long should it be stored. You need to determine at what point is data no longer relevant to the current analysis.
- More V's in the future …

Harnessing Big Data



- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

Who is generating Big Data?



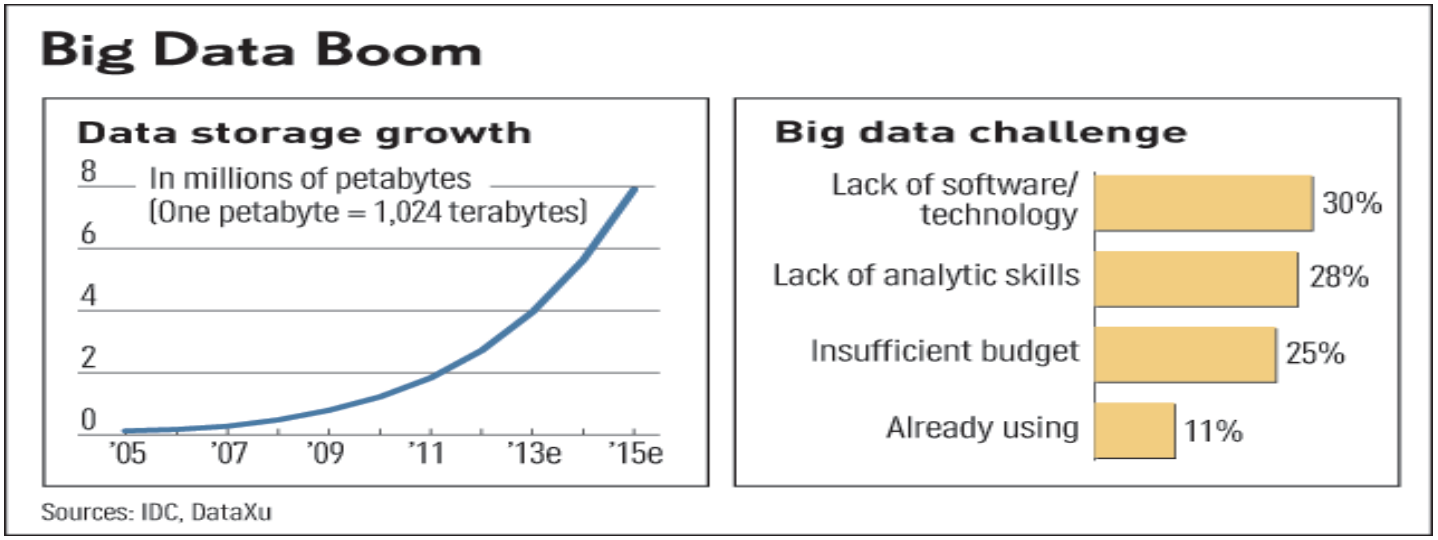
The Model Has Changed…

- **The Model of Generating/Consuming Data has Changed**

Old Model:

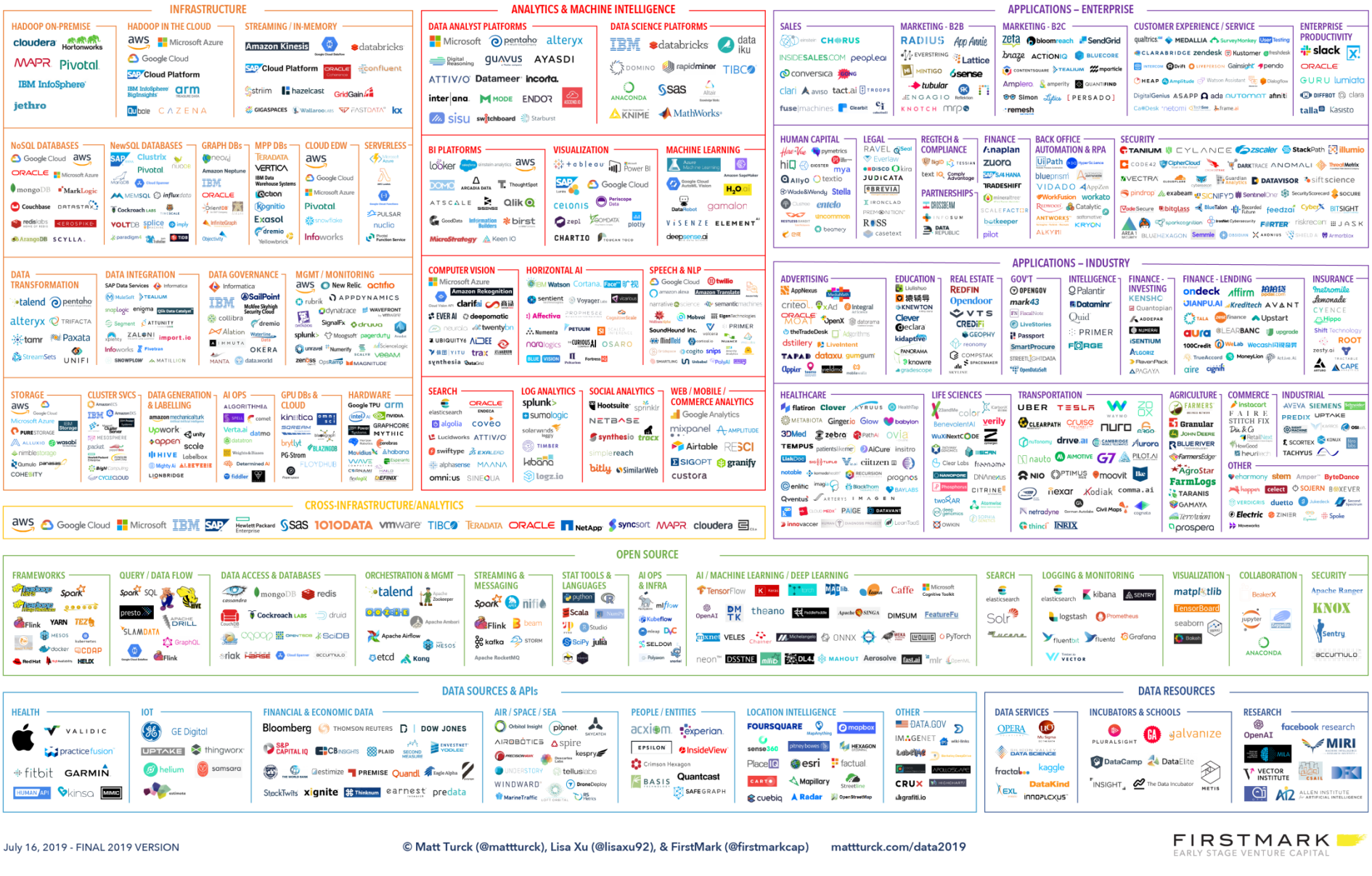
New Model:

Challenges in Handling Big Data



- **The Bottleneck is in technology**
  - New architecture, algorithms, techniques are needed
- **Also in technical skills**
  - Experts in using the new technology and dealing with big data

DATA & AI LANDSCAPE 2019



FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Google

Processes 20 PB a day (2008)
Crawls 20B web pages a day (2012)
Search index is 100+ PB (5/2014)
Bigtable serves 2+ EB, 600M QPS (5/2014)

INTERNET ARCHIVE

400B pages, 10+ PB (2/2014)

YAHOO!

Hadoop: 365 PB, 330K nodes (6/2014)

JPMorganChase

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

ebay

Hadoop: 10K nodes, 150K cores, 150 PB (4/2014)

300 PB data in Hive + 600 TB/day (4/2014)

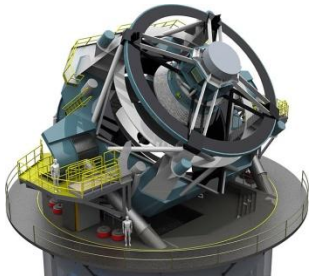
facebook

LHC: ~15 PB a year

CERN

amazon web services™

S3: 2T objects, 1.1M request/second (4/2013)



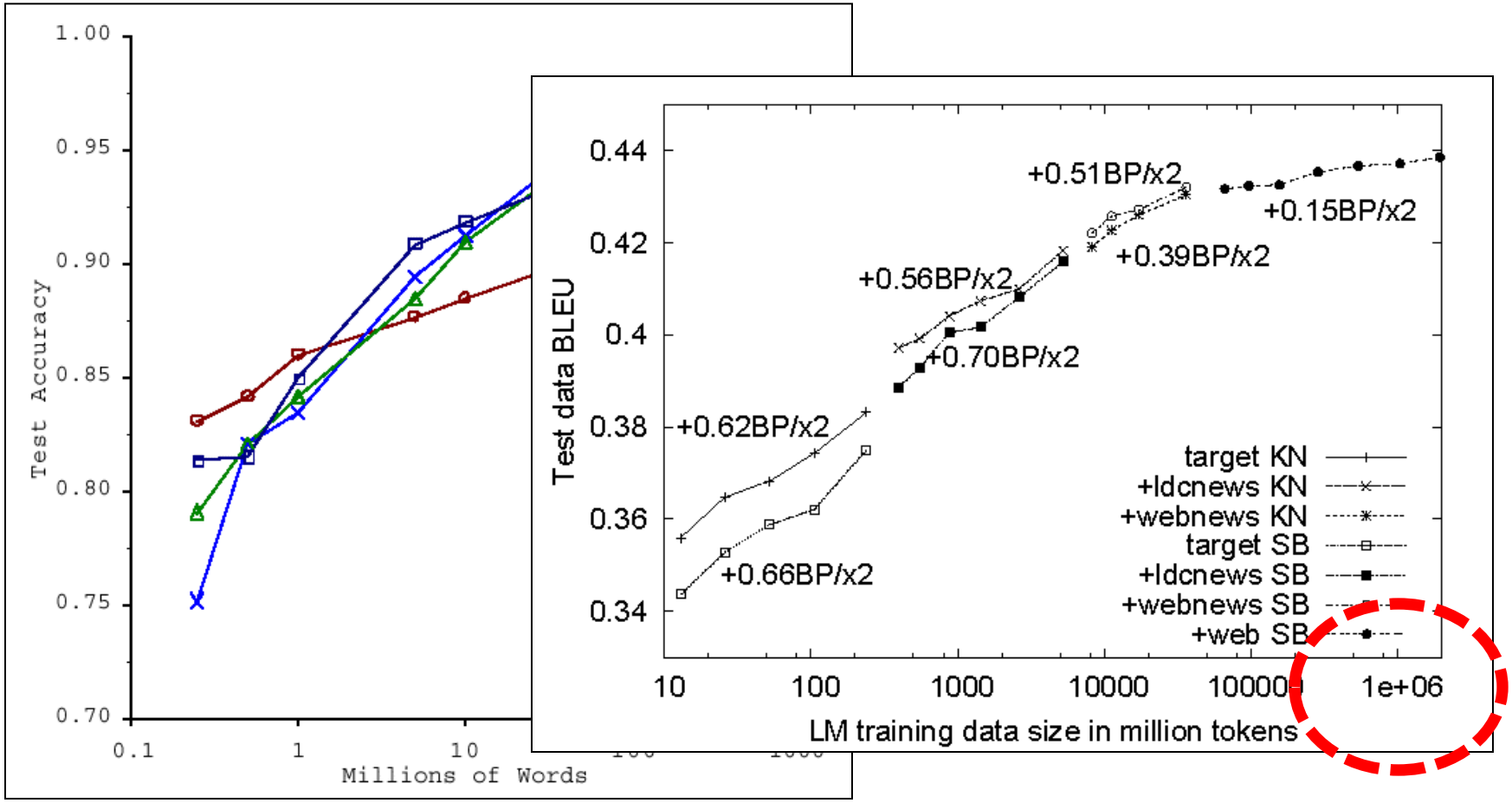
LSST: 6-10 PB a year (~2020)

SKA: 0.3 – 1.5 EB per year (~2020)

How much data?

No data like more data!

s/knowledge/data/g;



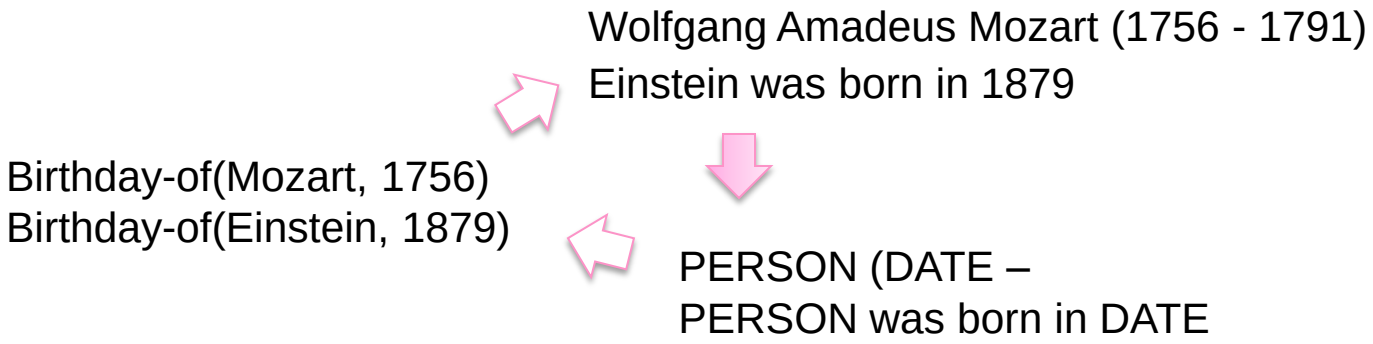
How do we get here if we're not Google?

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

What to do with more data?

- Answering 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; …)

Science

Emergence of the 4th Paradigm

Engineering

The unreasonable effectiveness of data

Count and normalize!


Know thy customers
Data → Insights → Competitive advantages
Commerce
EPSON
PEARL BAR

What is cloud computing?

Just a buzzword?

- Before clouds…
  - P2P computing
  - Grids
  - HPC
  - …
- Cloud computing means many different things:
  - Large-data processing
  - Rebranding of web 2.0
  - Utility computing
  - Everything as a service

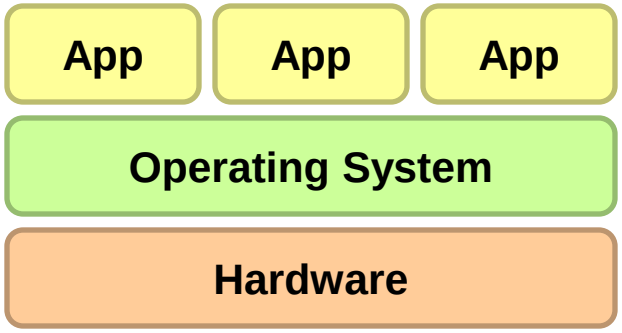
Rebranding of web 2.0

- 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

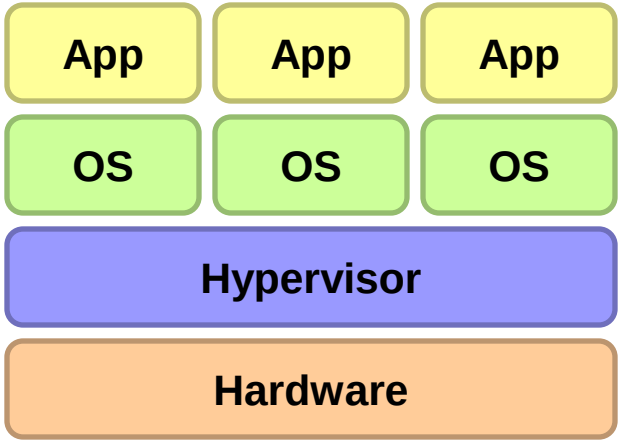
Utility Computing

- 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

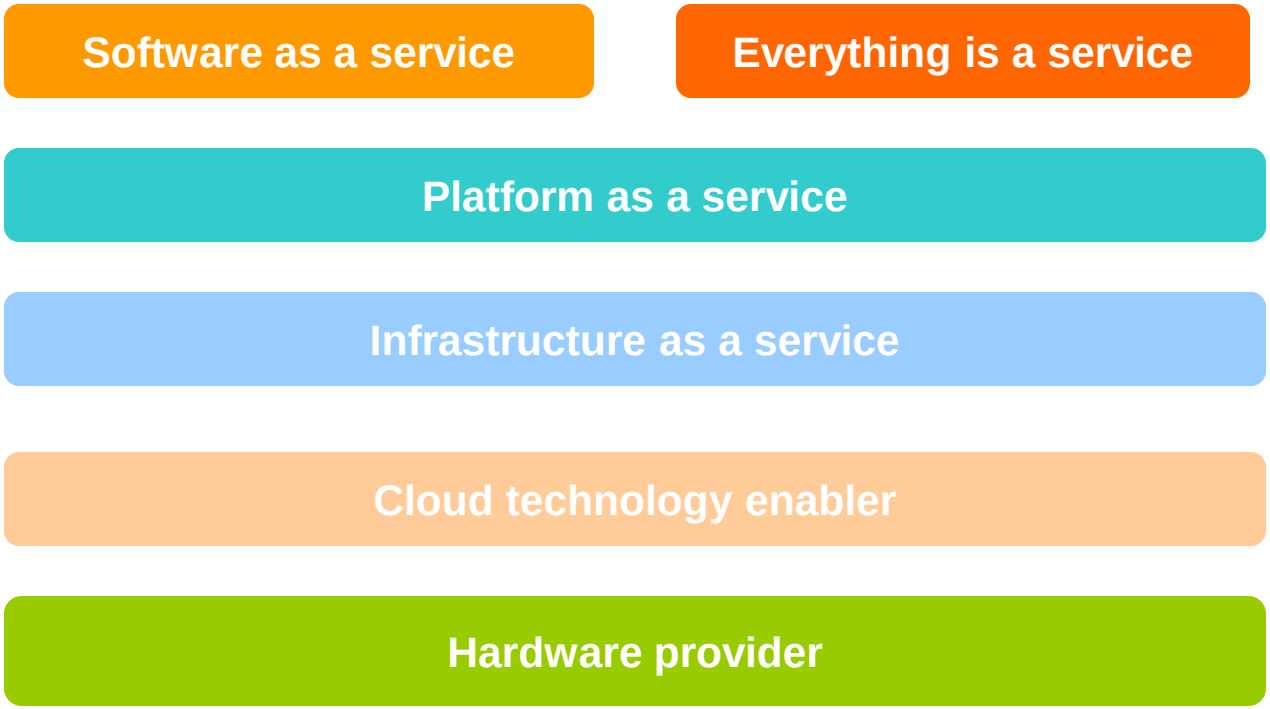


Traditional Stack



Virtualized Stack

Cloud computing market

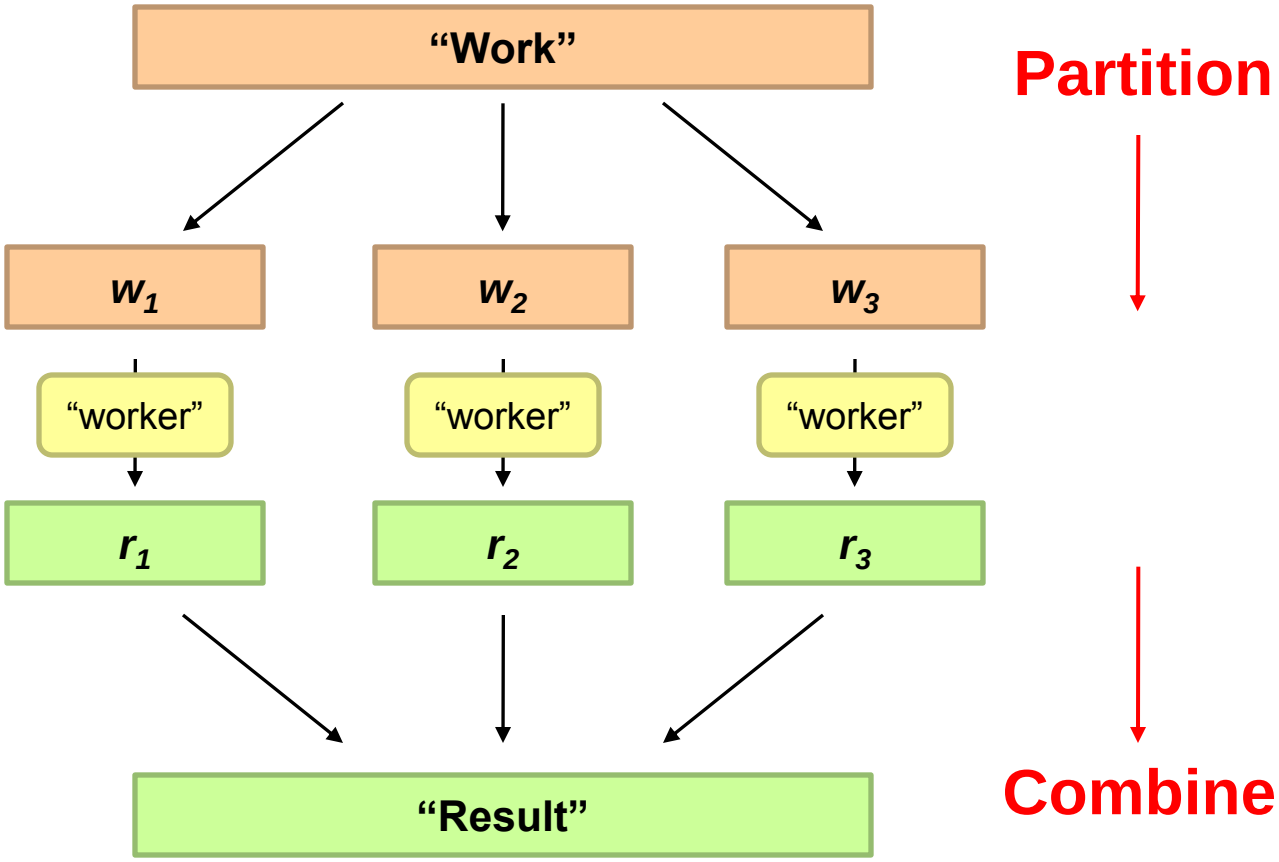


Everything as a Service

- 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

How do we scale up?

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?

Synchronization!

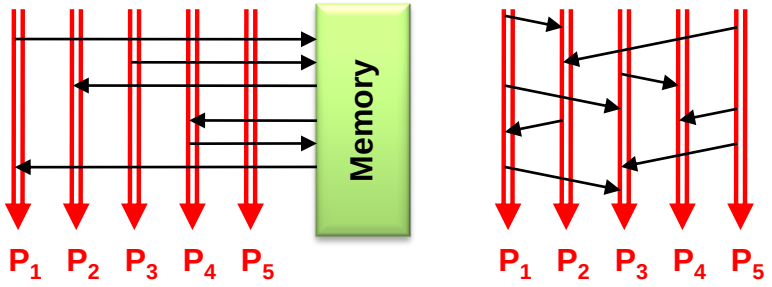
- Parallelization problems arise from:
  - Communication between workers (e.g., to exchange state)
  - Access to shared resources (e.g., data)
- Thus, we need a synchronization mechanism

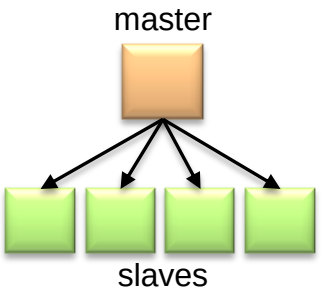
Managing Multiple Workers

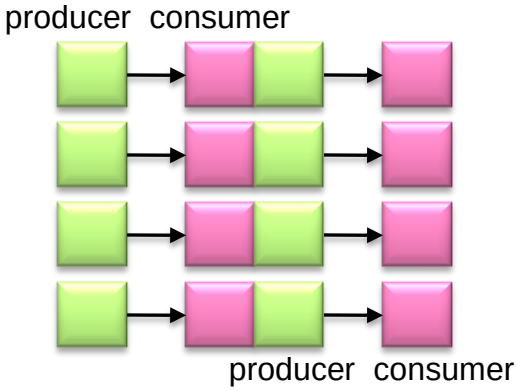
- Difficult because
  - We don't know the order in which workers run
  - We don't know when workers interrupt each other
  - We don't know the order in which workers access shared data
- Thus, we need:
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still,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

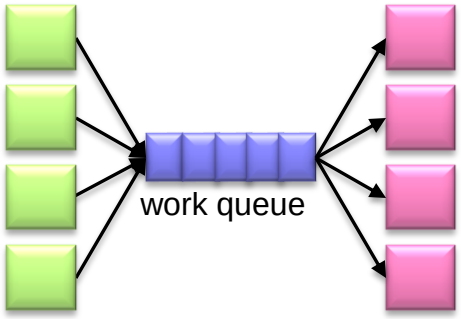
Current Tools

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







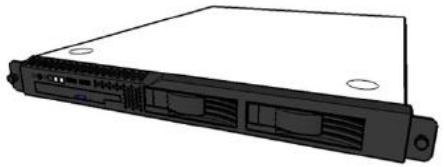


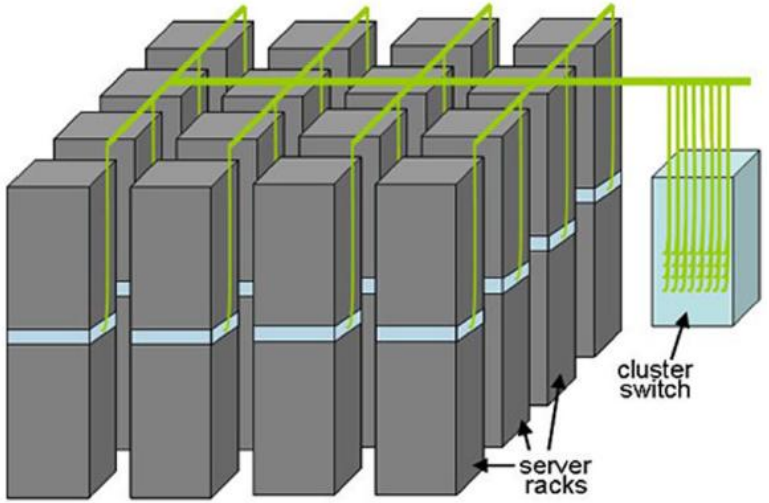
What's the point?

- 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!

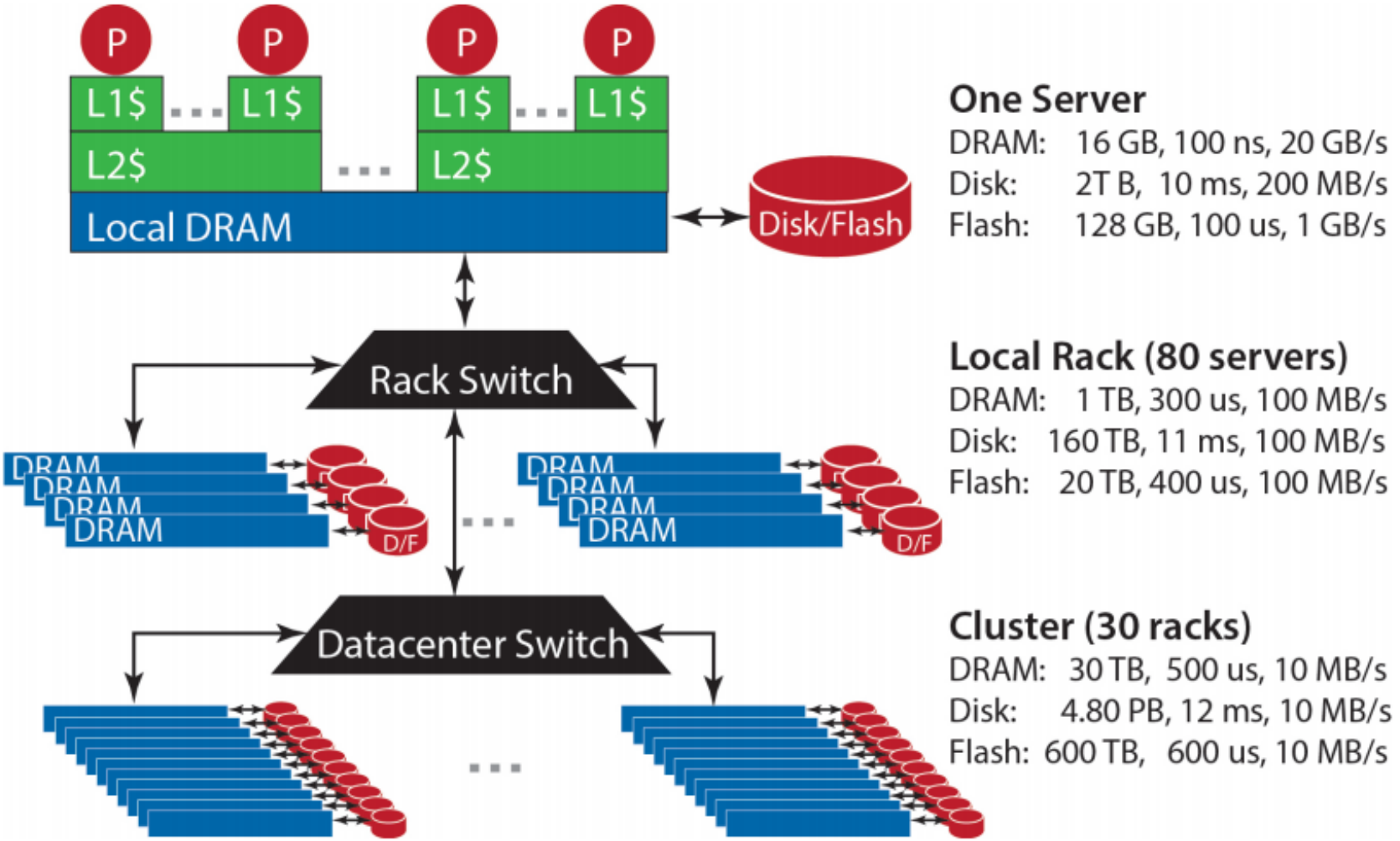
Building Blocks





Source: Barroso and Urs Hölzle (2009)

Storage Hierarchy



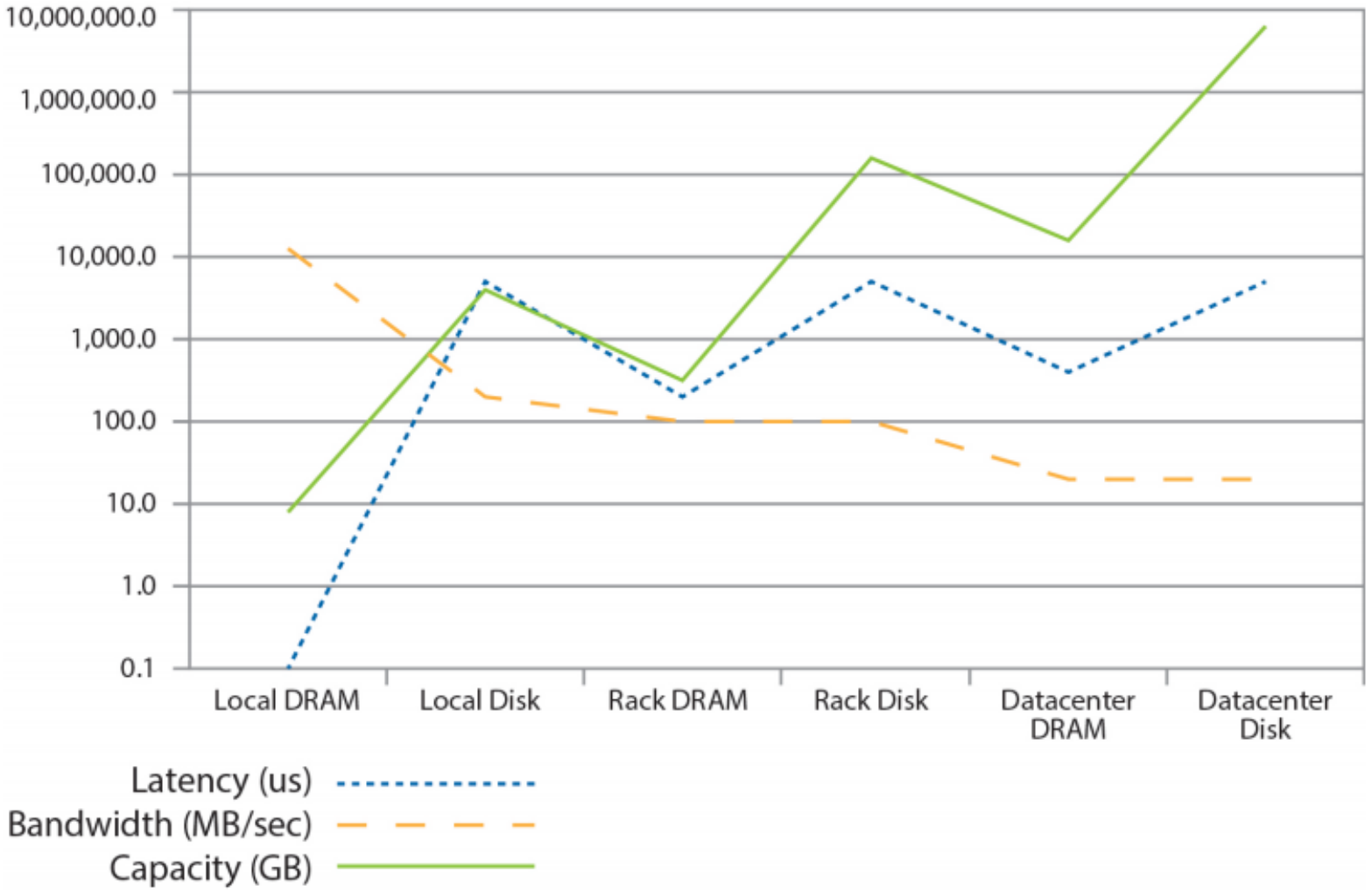
One Server
DRAM: 16 GB, 100 ns, 20 GB/s
Disk: 2T B, 10 ms, 200 MB/s
Flash: 128 GB, 100 us, 1 GB/s

Local Rack (80 servers)
DRAM: 1 TB, 300 us, 100 MB/s
Disk: 160 TB, 11 ms, 100 MB/s
Flash: 20 TB, 400 us, 100 MB/s

Cluster (30 racks)
DRAM: 30 TB, 500 us, 10 MB/s
Disk: 4.80 PB, 12 ms, 10 MB/s
Flash: 600 TB, 600 us, 10 MB/s

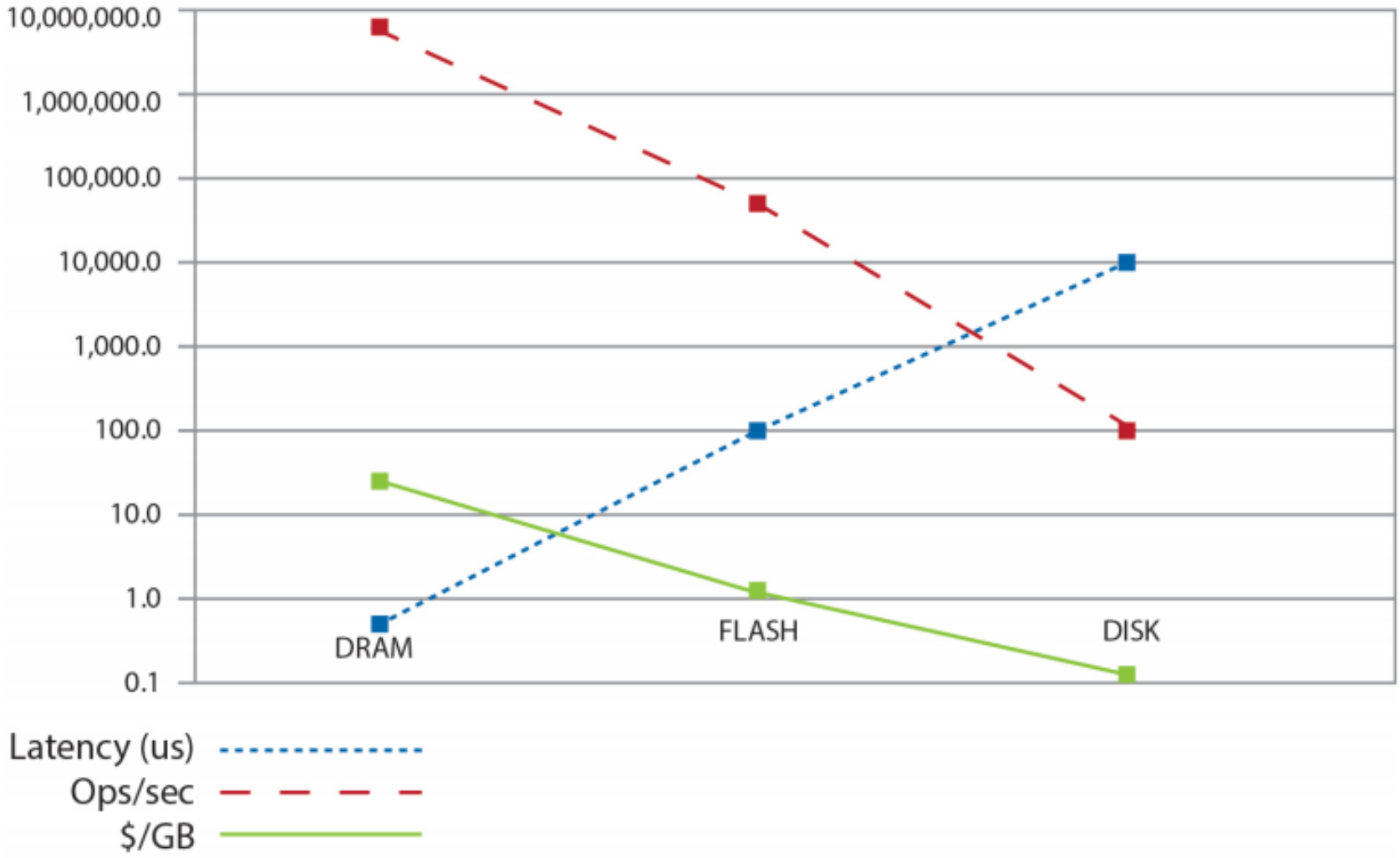
Source: Barroso and Urs Hölzle (2013)

Storage Hierarchy



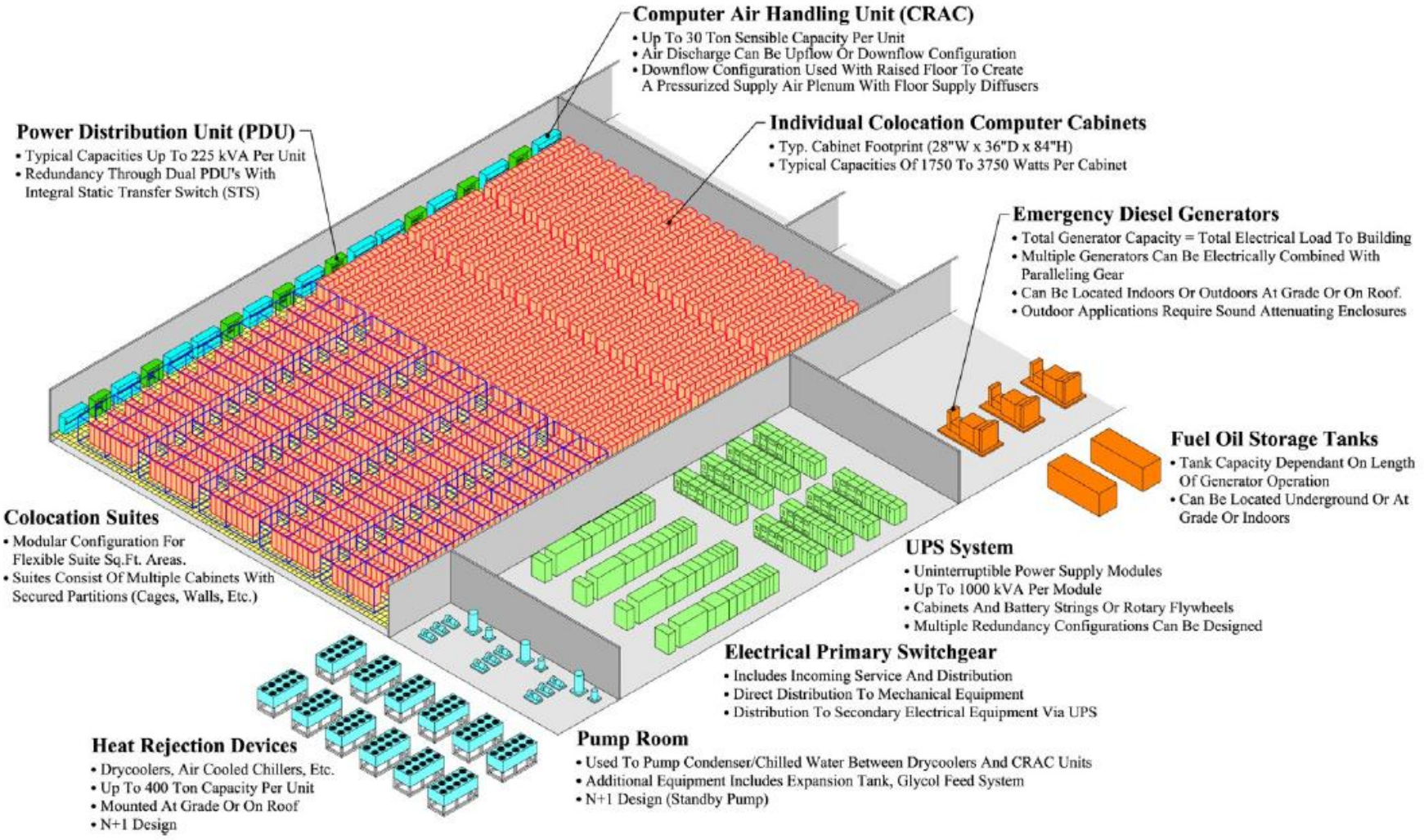
Source: Barroso and Urs Hölzle (2013)

Storage Hierarchy



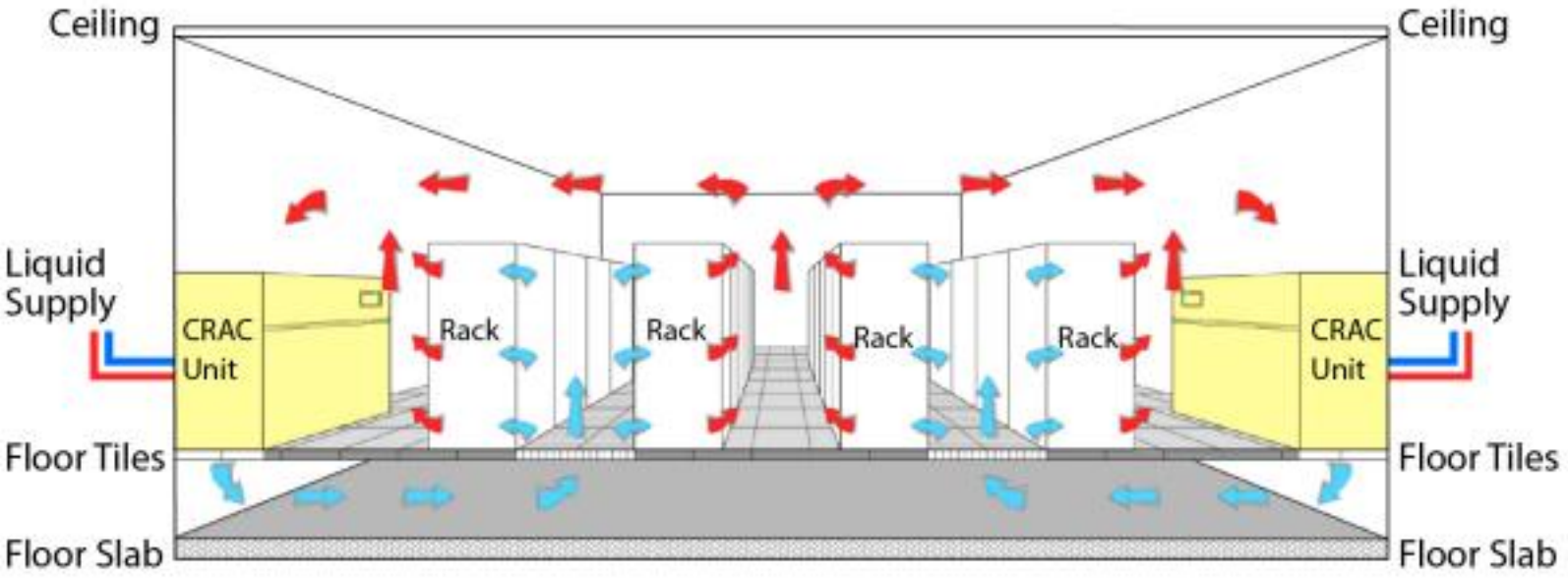
Source: Barroso and Urs Hölzle (2013)

Anatomy of a Datacenter



Source: Barroso and Urs Hölzle (2013)

Anatomy of a Datacenter



Source: Barroso and Urs Hölzle (2013)

Scaling “up” vs. “out”

- No single machine is large enough
  - Smaller cluster of large SMP machines vs. larger cluster of commodity machines (e.g., 16 128-core machines vs. 128 16-core machines)
- Nodes need to talk to each other!
  - Intra-node latencies: ~100 ns
  - Inter-node latencies: ~100 μs
- Let’s model communication overhead…

Source: analysis on this an subsequent slides from Barroso and Urs Hölzle (2009)

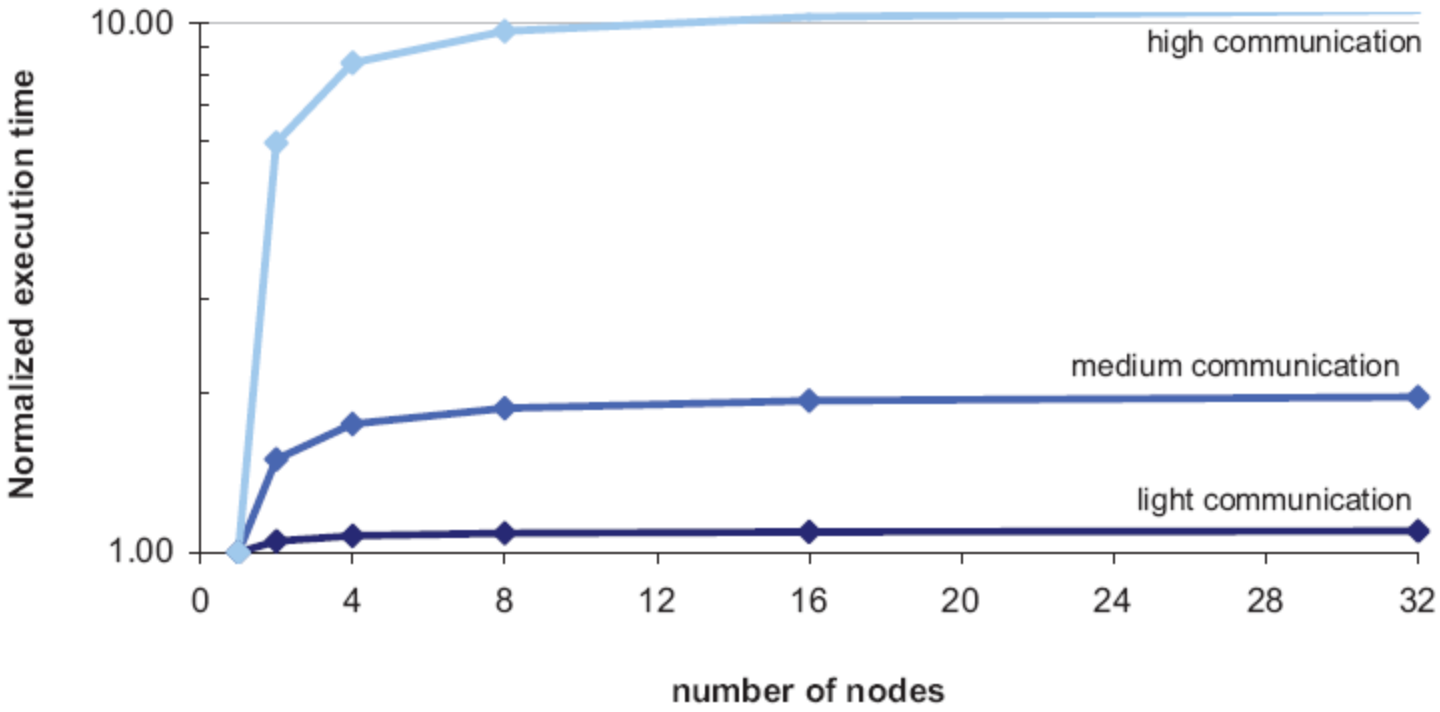
Modeling Communication Costs

- Simple execution cost model:
  - Total cost = cost of computation + cost to access global data
  - Fraction of local access inversely proportional to size of cluster
  - *n* nodes (ignore cores for now)

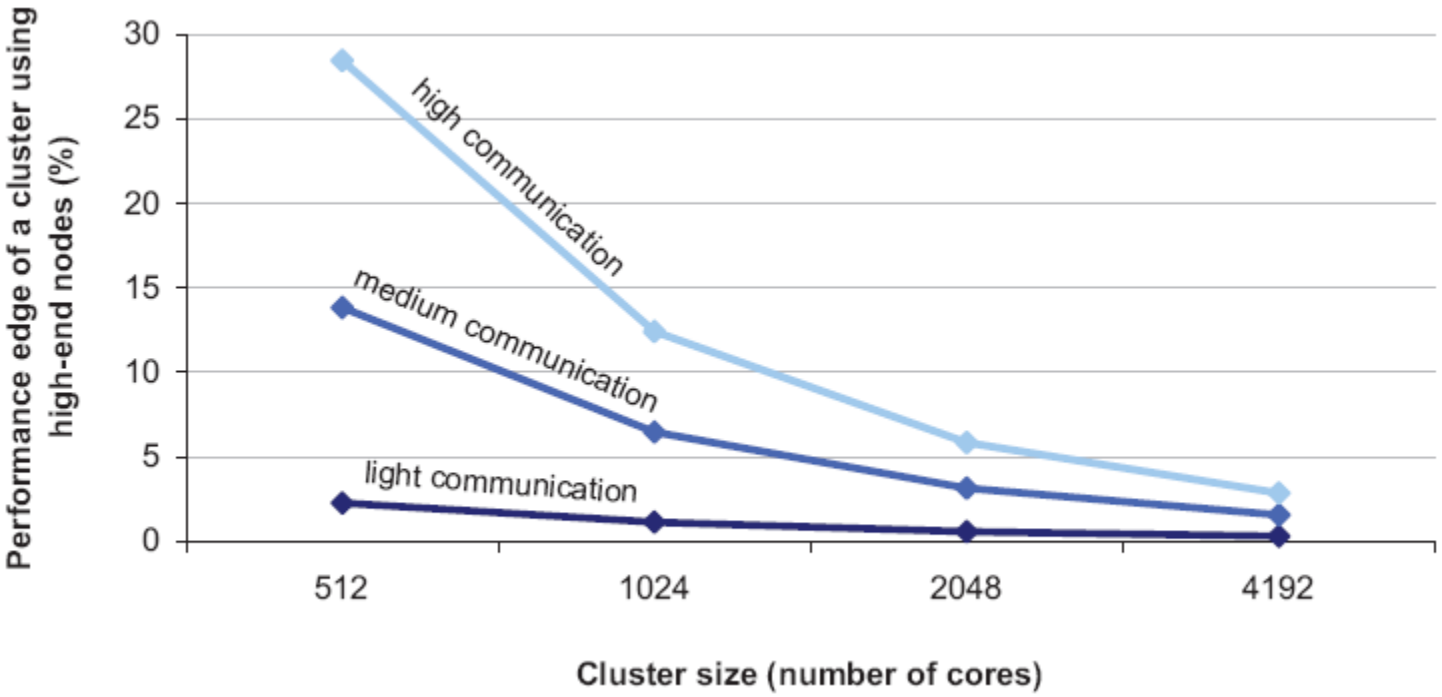
$$1\ \text{ms} + f \times [100\ \text{ns} \times (1/n) + 100\ \mu\text{s} \times (1 - 1/n)]$$

    - Light communication: $f = 1$
    - Medium communication: $f = 10$
    - Heavy communication: $f = 100$
- What are the costs in parallelization?

Cost of Parallelization



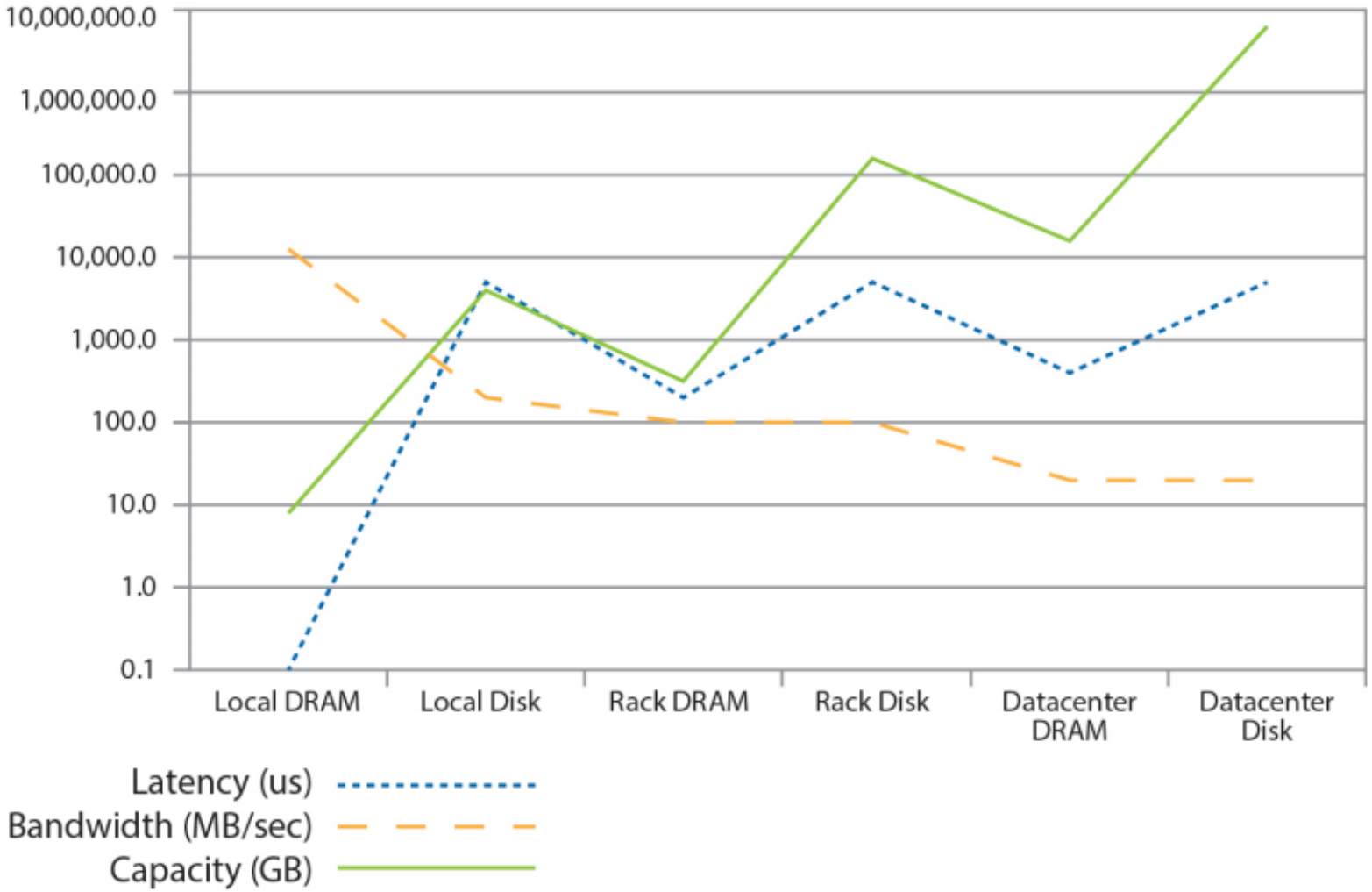
Advantages of scaling “up”



So why not?

Why does commodity beat exotic?

Moving Data Around



Source: Barroso and Urs Hölzle (2013)

Seeks vs. Scans

- Consider a 1 TB database with 100 byte records
  - We want to update 1 percent of the records
- Scenario 1: random access
  - Each update takes ~30 ms (seek, read, write)
  - 10^8 updates = ~35 days
- Scenario 2: rewrite all records
  - Assume 100 MB/s throughput
  - Time = 5.6 hours(!)
- Lesson: avoid random seeks!

Source: Ted Dunning, on Hadoop mailing list

“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

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

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

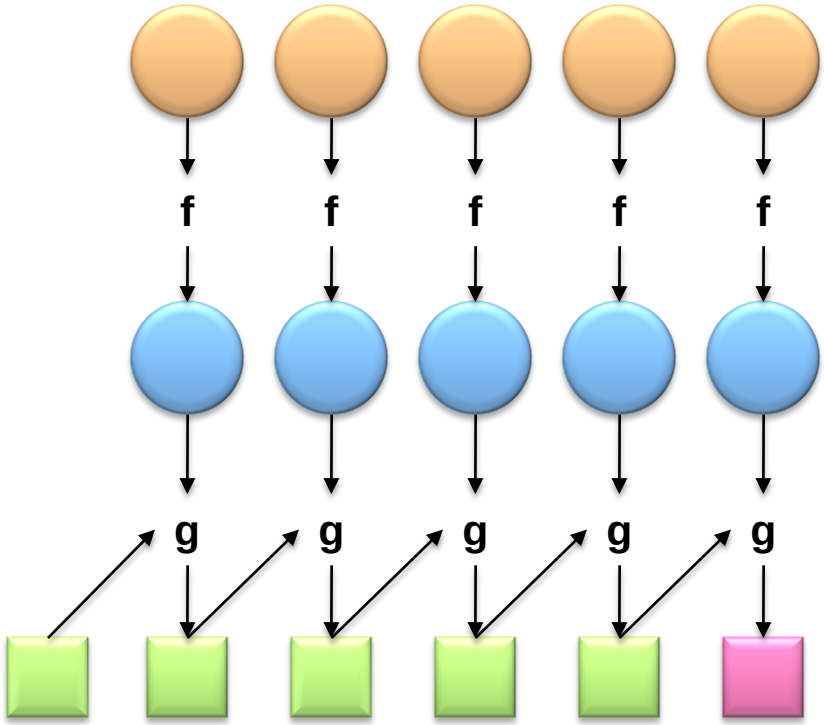
Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

Roots in Functional Programming

Map

Fold



Challenges

1. **Cheap nodes fail, especially if you have many**
   - Mean time between failures for 1 node = 3 years
   - Mean time between failures for 1000 nodes = 1 day
   - Solution: Build fault-tolerance into system

2. **Commodity network = low bandwidth**
   - Solution: Push computation to the data

3. **Programming distributed systems is hard**
   - Solution: Data-parallel programming model: users write “map” & “reduce” functions, system distributes work and handles faults

MapReduce

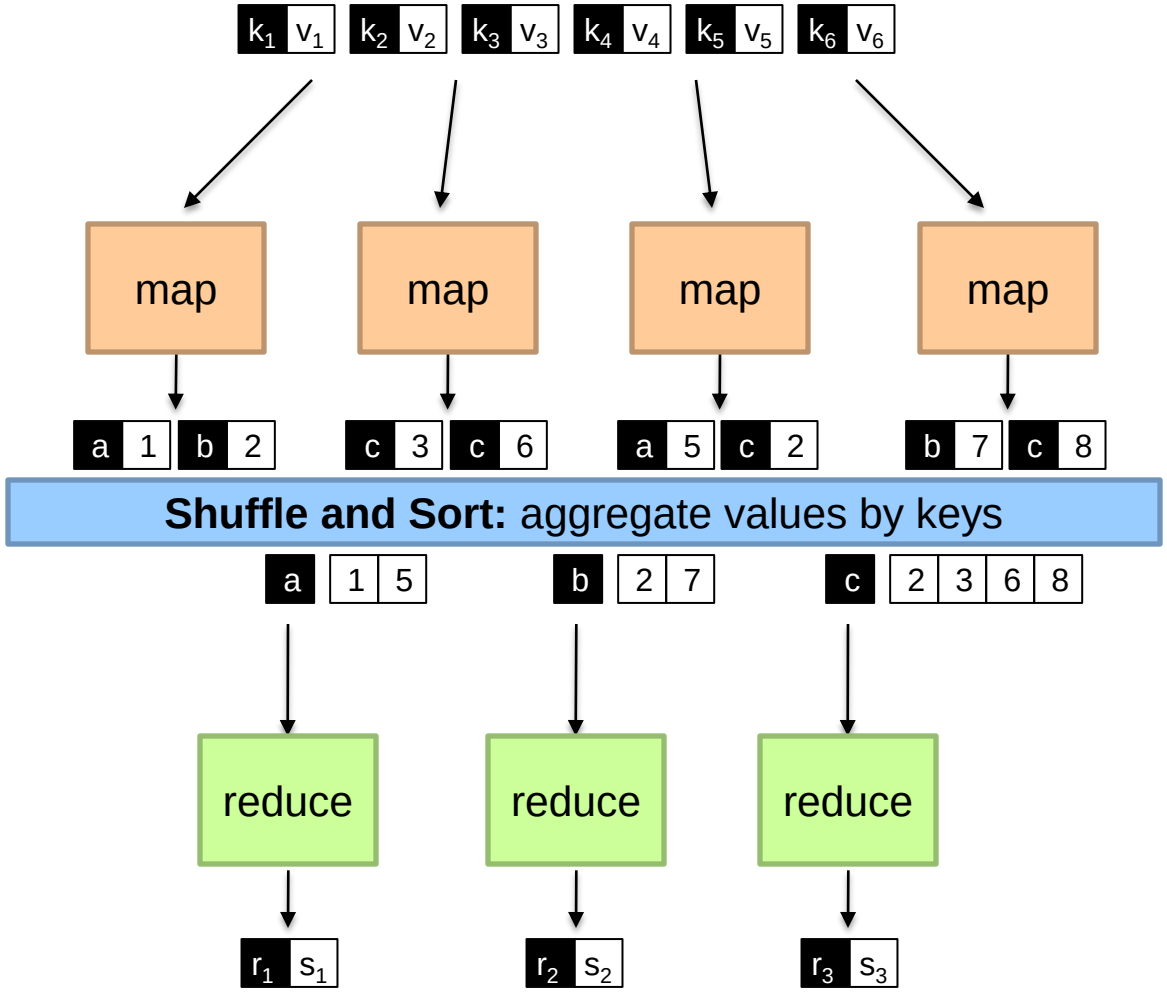
- Programmers specify two functions:

  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"?


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 “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

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…

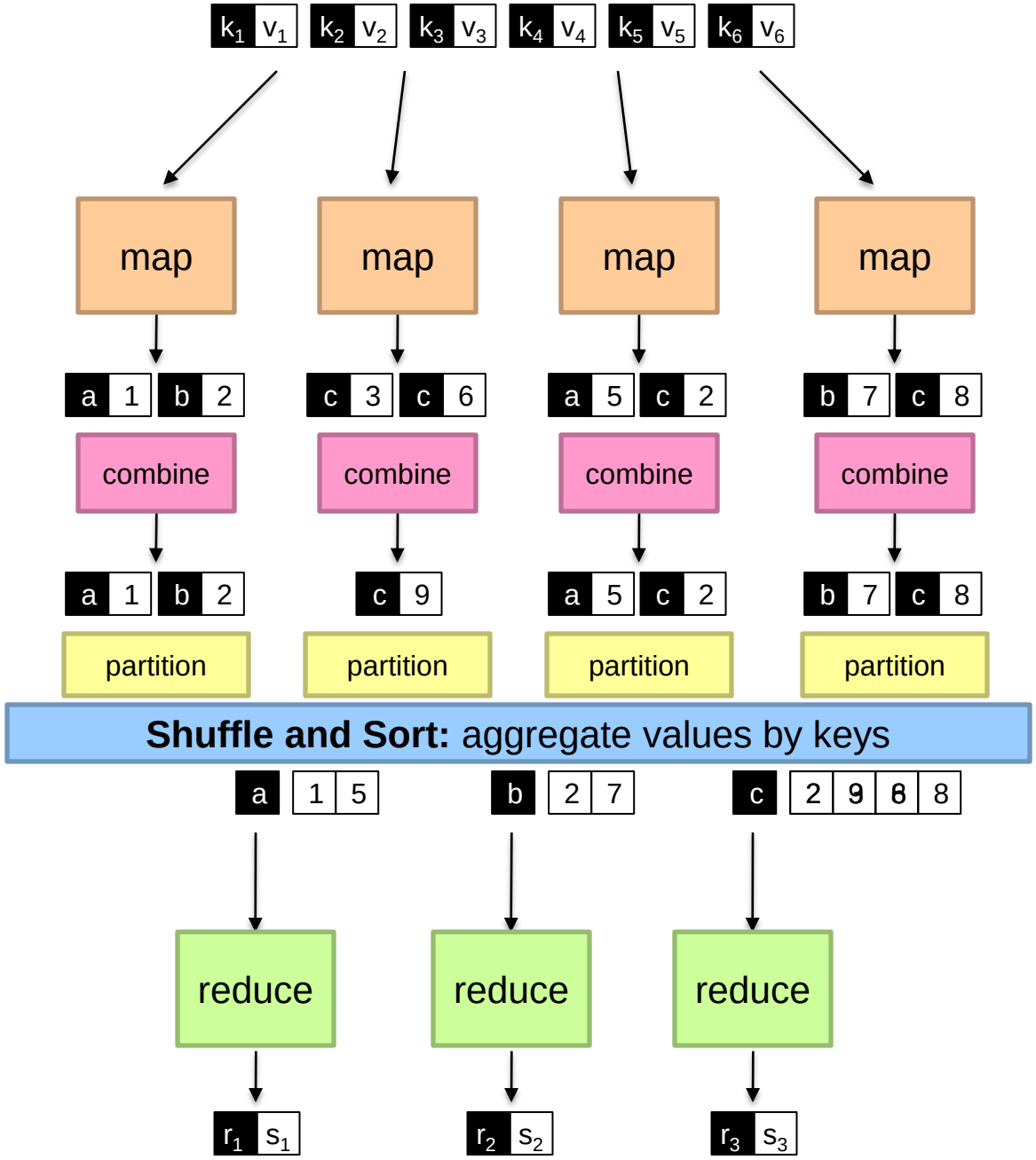
- Not quite…usually, programmers also specify:

  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

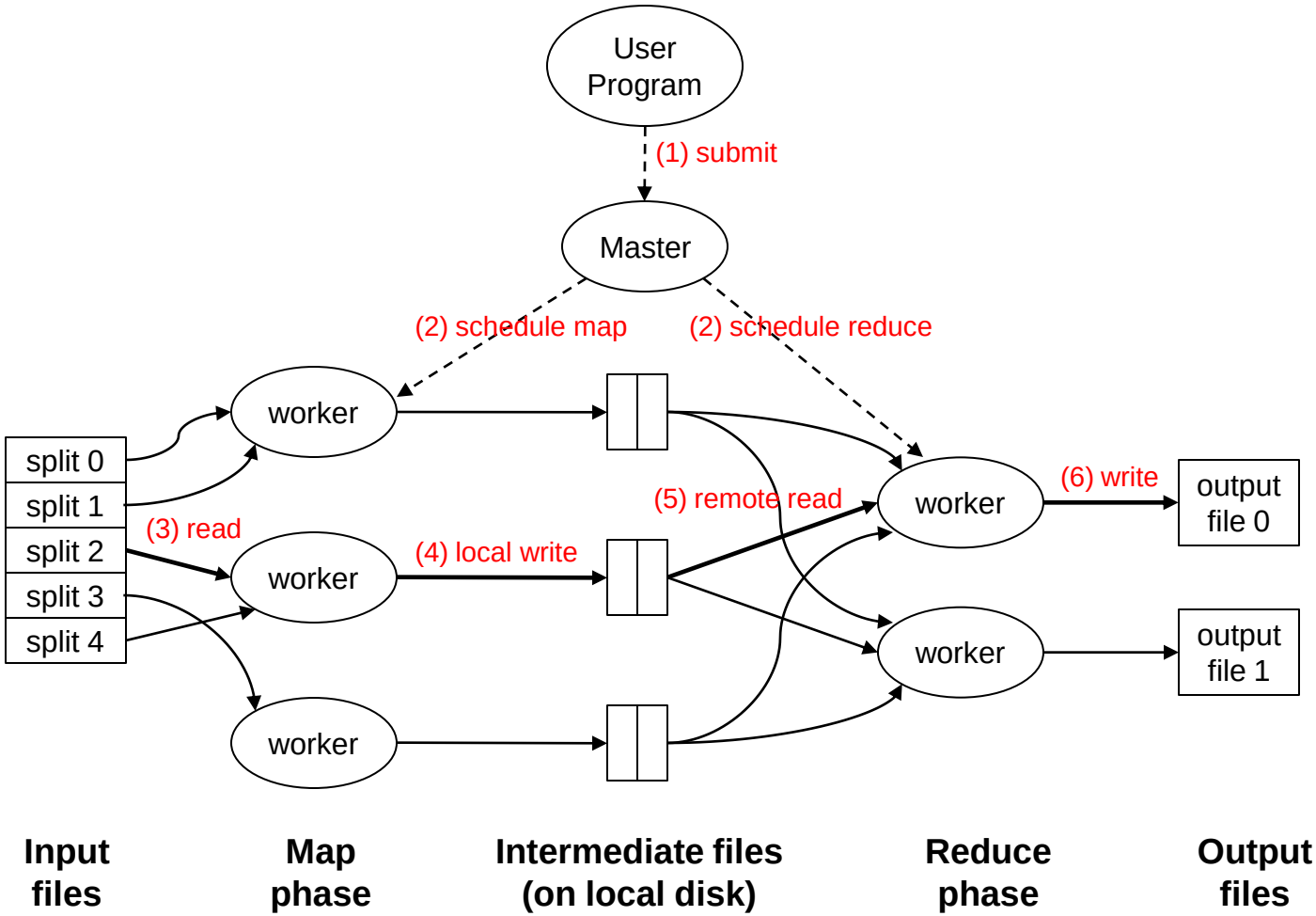

k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
combine
combine
combine
combine
a 1
b 2
c 9
a 5
c 2
b 7
c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

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

MapReduce Execution

- Single *master* controls job execution on multiple *slaves*
- Mappers preferentially placed on same node or same rack as their input block
  - Minimizes network usage
- Mappers save outputs to local disk before serving them to reducers
  - Allows recovery if a reducer crashes
  - Allows having more reducers than nodes



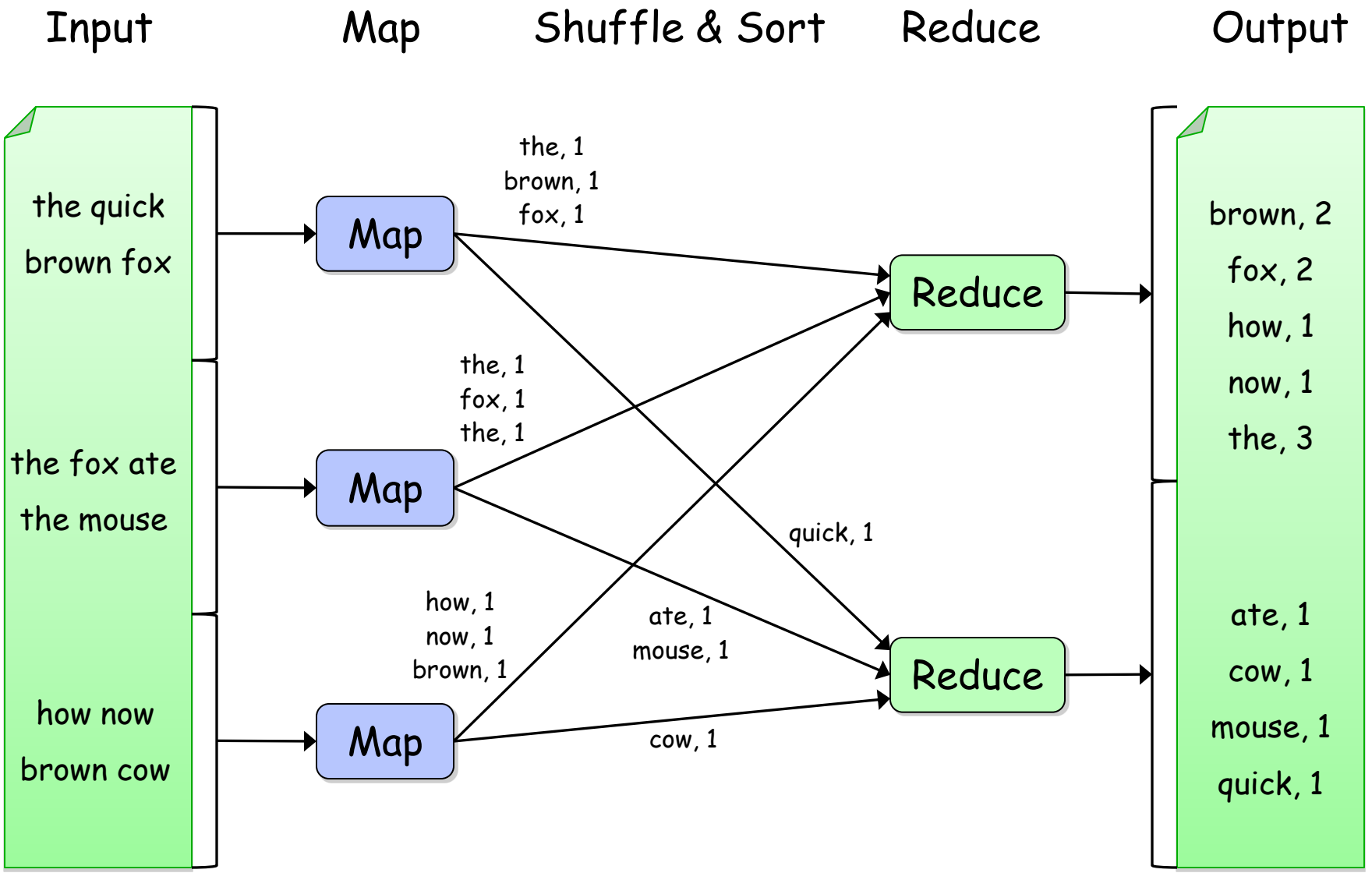
Adapted from (Dean and Ghemawat, OSDI 2004)

“Hello World”: Word Count

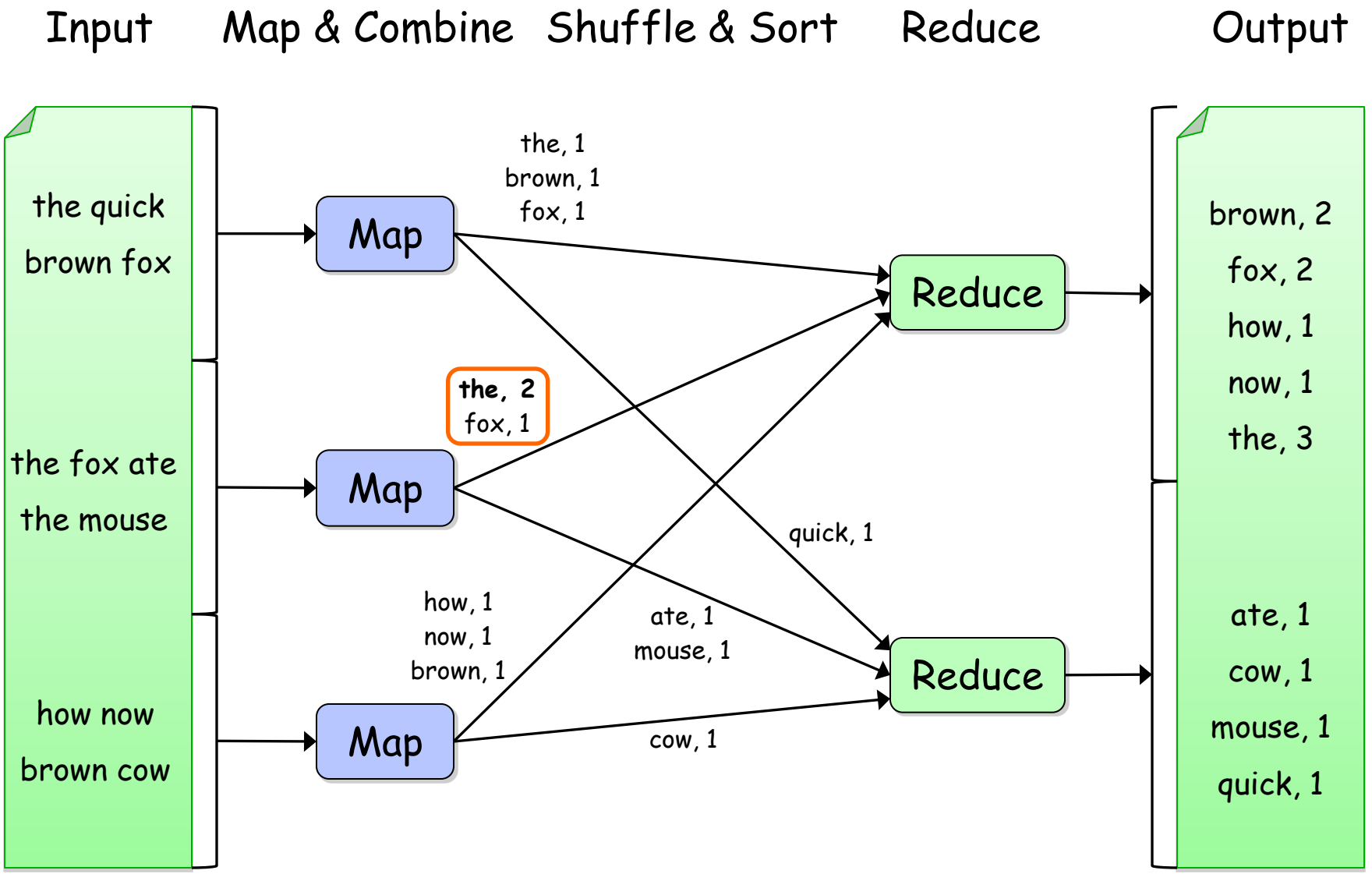
```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

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

Word Count Execution



Word Count with Combiner



Search Example

- **Input:** (lineNumber, line) records
- **Output:** lines matching a given pattern

- **Map:**

```
if(line matches pattern):
    output(line)
```

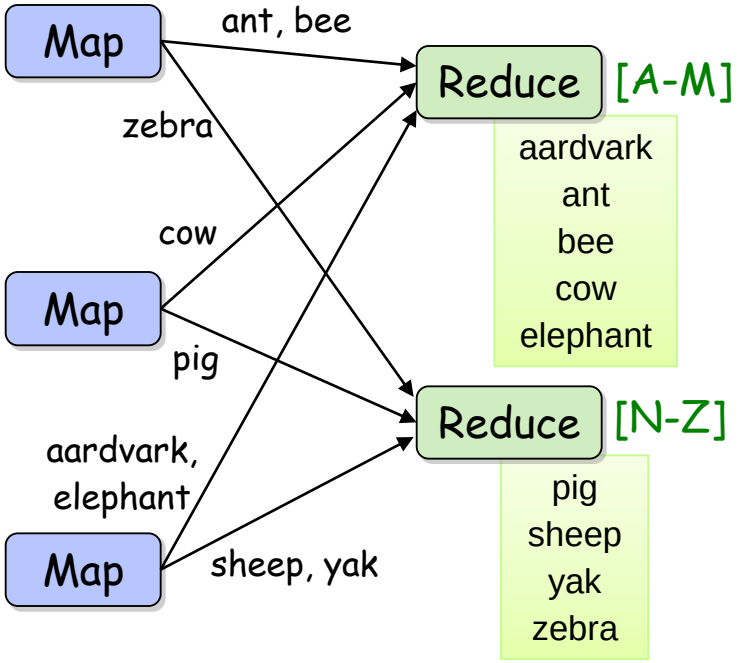
- **Reduce:** identify function
  - Alternative: no reducer (map-only job)

Sort Example

- **Input:** (key, value) records
- **Output:** same records, sorted by key

- **Map:** identity function
- **Reduce:** identify function

- **Trick:** Pick partitioning function h such that $k_1<k_2 \Rightarrow h(k_1)<h(k_2)$



Inverted Index Example

- **Input:** (filename, text) records
- **Output:** list of files containing each word

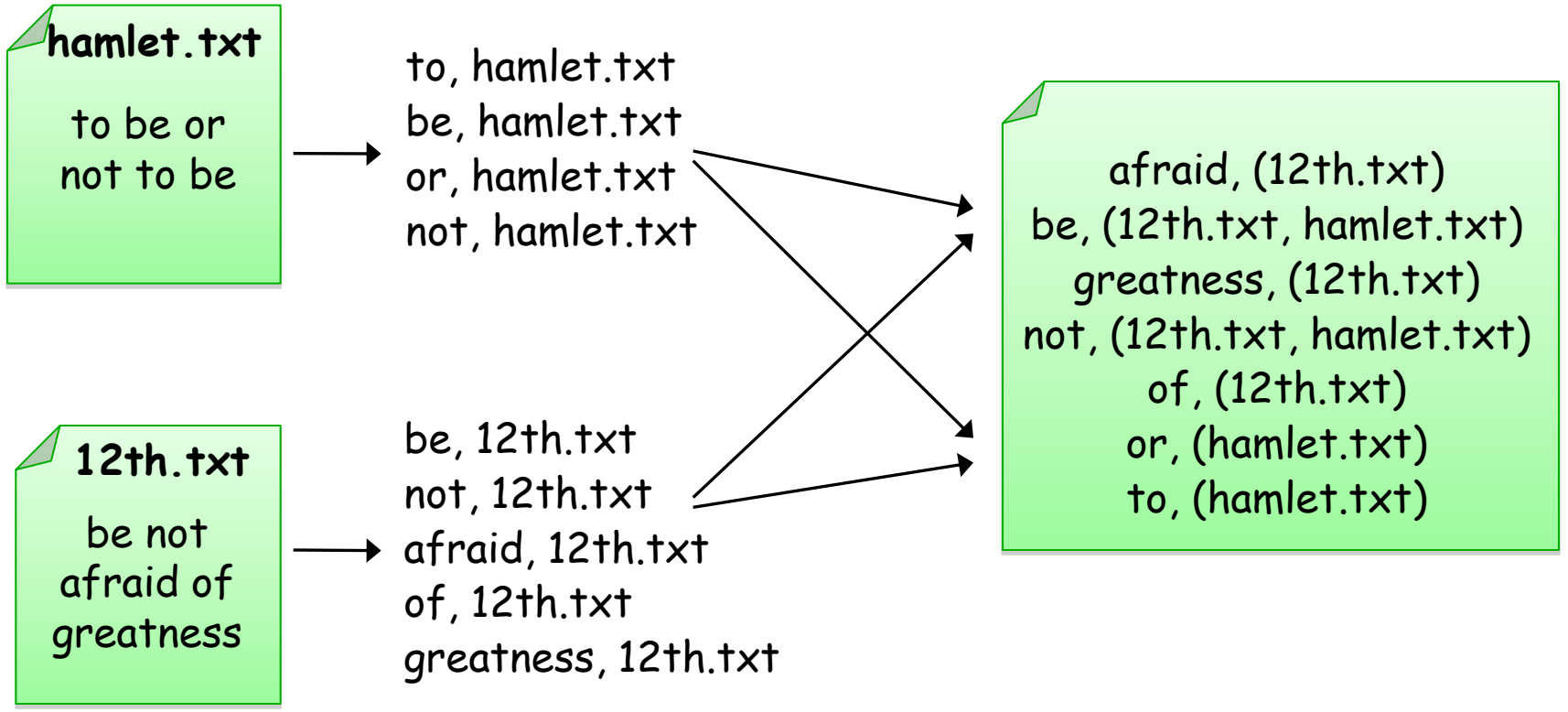
- **Map:**

```
foreach word in text.split():
    output(word, filename)
```

- **Combine:** uniquify filenames for each word
- **Reduce:**

```
def reduce(word, filenames):
    output(word, sort(filenames))
```

Inverted Index Example



Most Popular Words Example

- **Input:** (filename, text) records
- **Output:** top 100 words occurring in the most files

- Two-stage solution:
  - **Job 1:**
    - Create inverted index, giving (word, list(file)) records
  - **Job 2:**
    - Map each (word, list(file)) to (count, word)
    - Sort these records by count as in sort job

- Optimizations:
  - Map to (word, 1) instead of (word, file) in Job 1
  - Count files in job 1's reducer rather than job 2's mapper
  - Estimate count distribution in advance and drop rare words

Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     - OK for a map because it has no dependencies
     - OK for reduce because map outputs are on disk
   - If the same task fails repeatedly, fail the job or ignore that input block (user-controlled)

➢ **Note: For these fault tolerance features to work, *your map and reduce tasks must be side-effect-free***

Fault Tolerance in MapReduce

2. If a node crashes:
   - Re-launch its current tasks on other nodes
   - Re-run any maps the node previously ran
     - Necessary because their output files were lost along with the crashed node

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node ("speculative execution")
   - Take the output of whichever copy finishes first, and kill the other

- Surprisingly important in large clusters
  - Stragglers occur frequently due to failing hardware, software bugs, misconfiguration, etc
  - Single straggler may noticeably slow down a job

Takeaways

- By providing a data-parallel programming model, MapReduce can control job execution in useful ways:
  - Automatic division of job into tasks
  - Automatic placement of computation near data
  - Automatic load balancing
  - Recovery from failures & stragglers

- User focuses on application, not on complexities of distributed computing

Hadoop Components

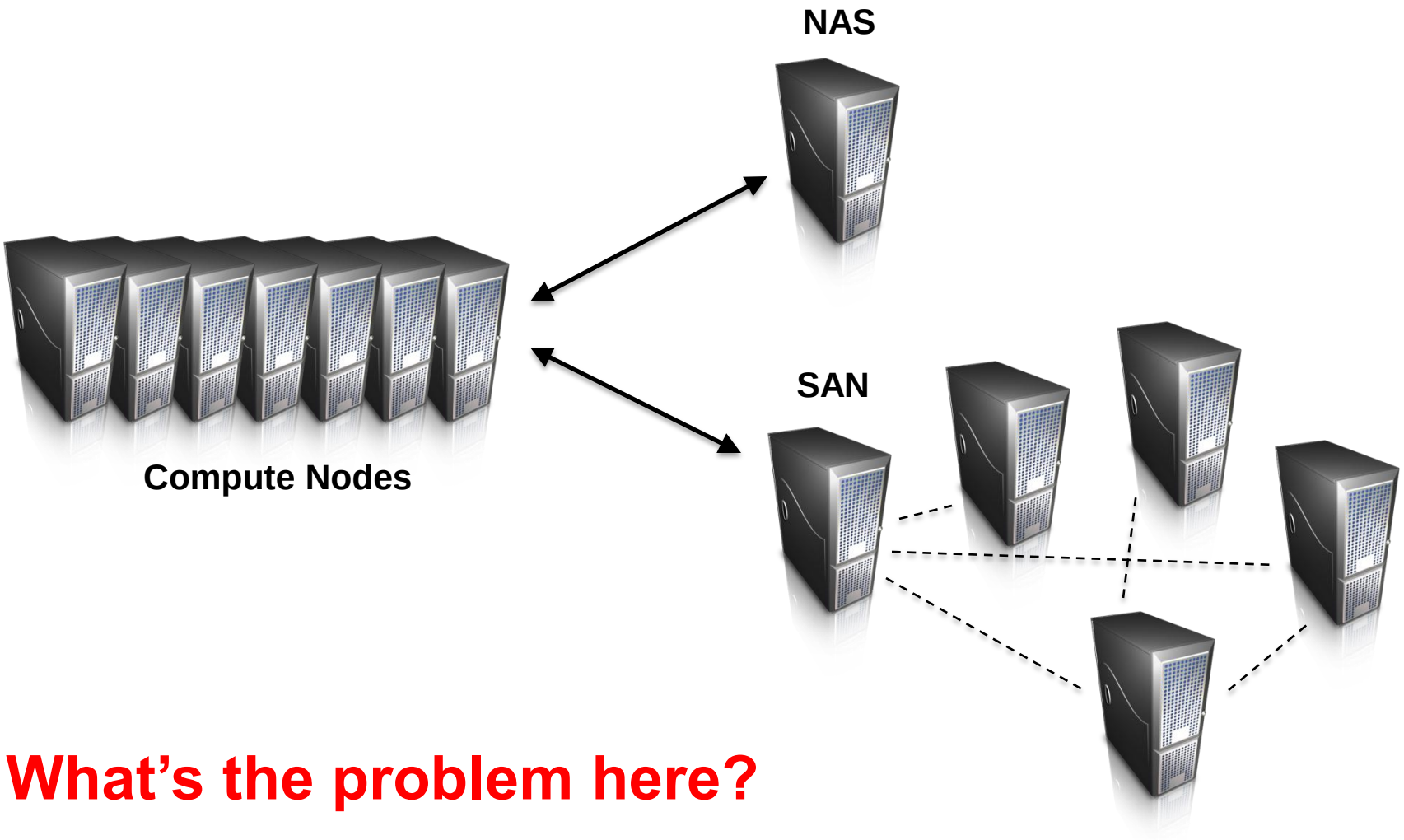
- **Distributed file system (HDFS)**
  - Single namespace for entire cluster
  - Replicates data 3x for fault-tolerance

- **MapReduce framework**
  - Executes user jobs specified as “map” and “reduce” functions
  - Manages work distribution & fault-tolerance

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
  - Large and expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.

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: 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)

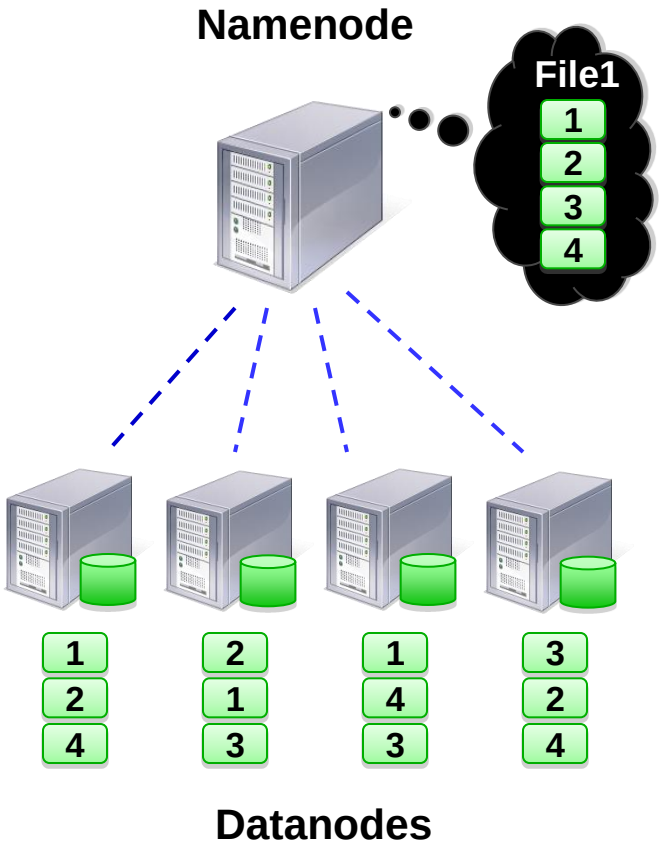
From GFS to HDFS

- 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…

Hadoop Distributed File System

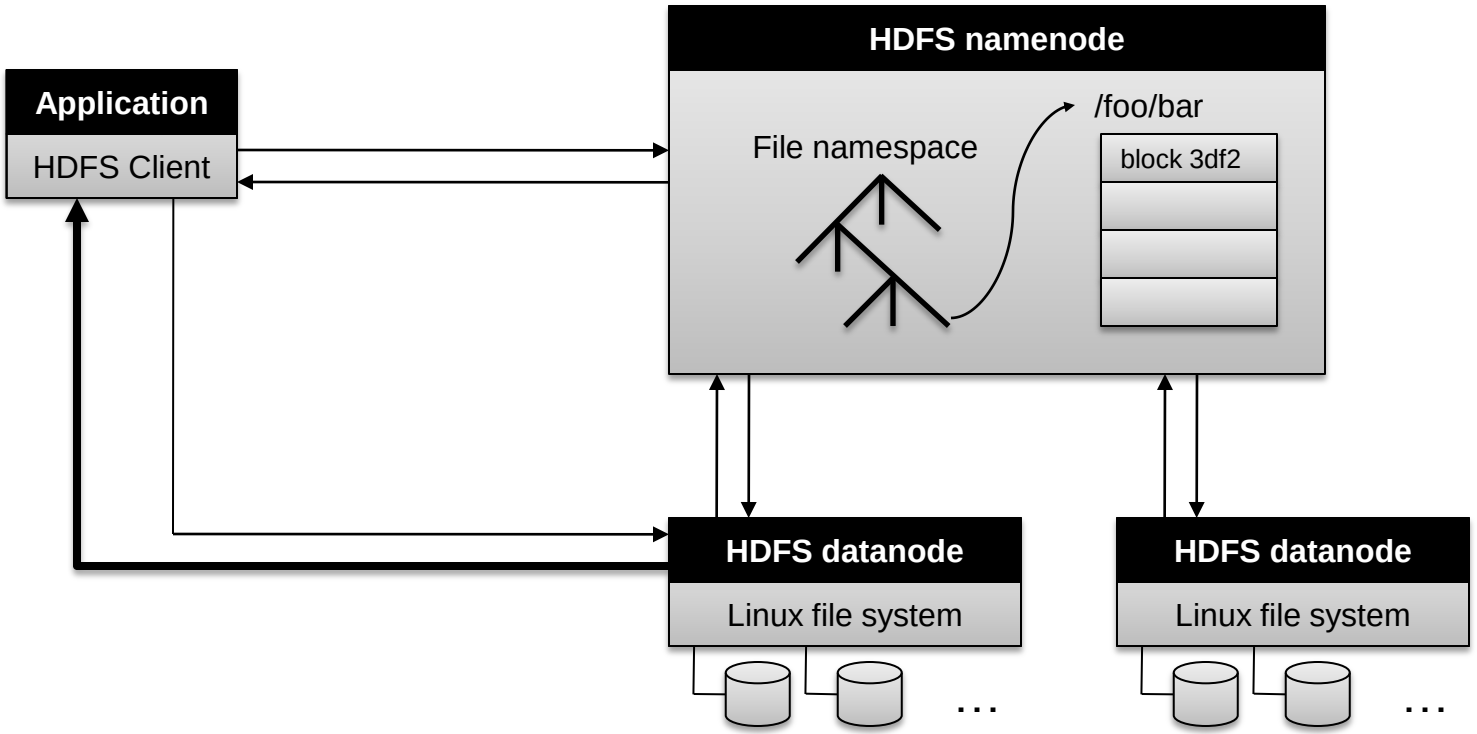
- Files split into 64MB *blocks*
- Blocks replicated across several *datanodes* (usually 3)
- Single *namenode* stores metadata (file names, block locations, etc)
- Optimized for large files, sequential reads
- Files are append-only



Namenode Responsibilities

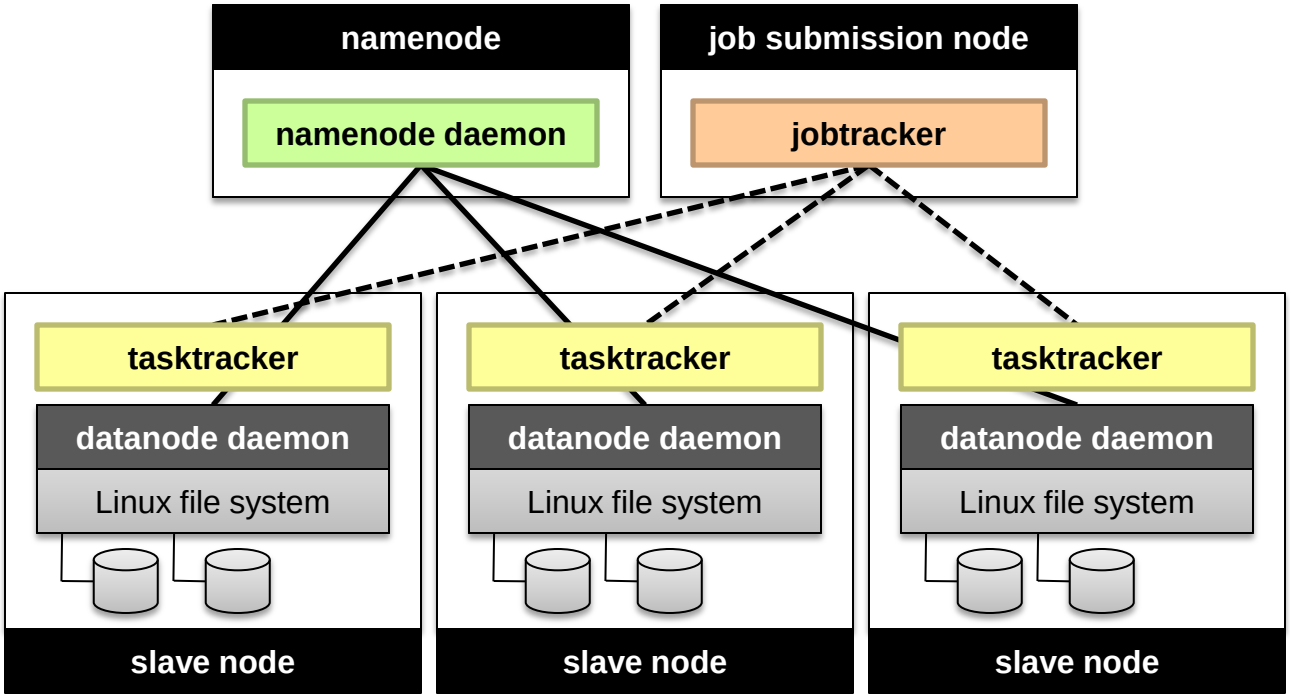
- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic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

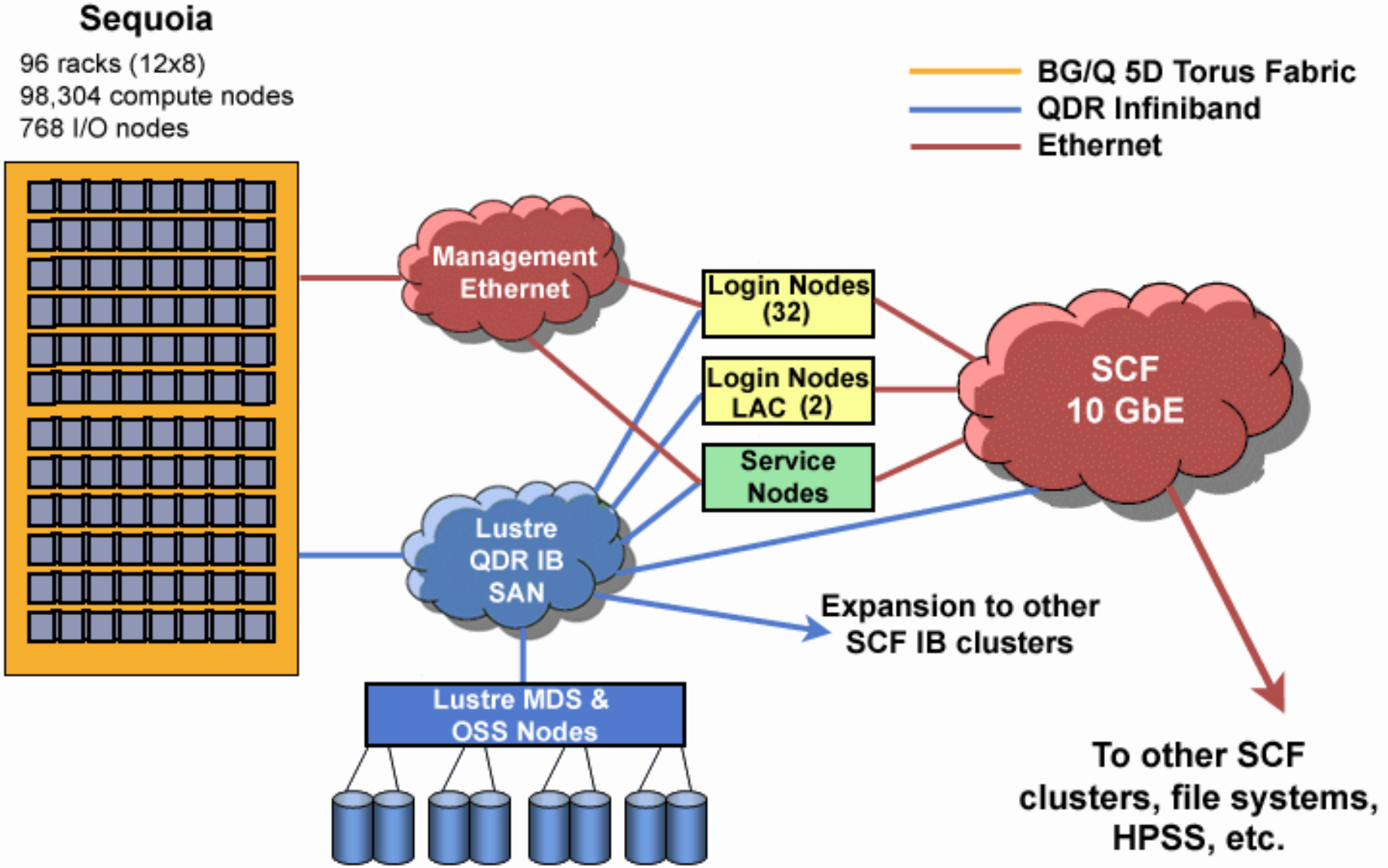
HDFS Architecture



Adapted from (Ghemawat et al., SOSP 2003)

Putting everything together…




Sequoia
96 racks (12x8)
98,304 compute nodes
768 I/O nodes
BG/Q 5D Torus Fabric
QDR Infiniband
Ethernet
Management Ethernet
Login Nodes (32)
Login Nodes LAC (2)
Service Nodes
SCF 10 GbE
Lustre QDR IB SAN
Expansion to other SCF IB clusters
Lustre MDS & OSS Nodes
To other SCF clusters, file systems, HPSS, etc.

Source: LLNL