CS4225/CS5425 Big Data Systems for Data Science

Logistics + Introduction

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg

School of Computing

Schedule

Week	Date	Topics	Tutorial	Due Dates
1	18 Aug	Overview and Introduction		
2	25 Aug	MapReduce - Introduction		
3	1 Sep	Polling Day Public Holiday		
4	8 Sep	MapReduce and Databases	Tutorial: MapReduce	Assignment 1 released
5	15 Sep	NoSQL Overview 1		
6	22 Sep	NoSQL Overview 2	Tutorial: NoSQL	
Recess				
7	6 Oct	Apache Spark 1		
8	13 Oct	Apache Spark 2	Tutorial: Spark	Assignment 1 due (15 Oct 11.59pm), Assignment 2 released
9	20 Oct	Stream Processing 1		
10	27 Oct	Stream Processing 2	Tutorial: Stream Processing	
11	3 Nov	Large Graph Processing 1		
12	10 Nov	NUS Well-Being Day		
13	17 Nov	Large Graph Processing 2	Tutorial: Graph Processing	Assignment 2 due (19 Nov 11.59pm)
	29 Nov	**Final Exam**		

Assessment

- 2 assignments on Hadoop and Spark (25% each)
- Final exam (50%) – held in-person.
- (Note: no weightage for in-lecture-quizzes)

Tutorials

- Start from Week 4
- Not counted for final grade
- Tutorial questions will be available on Canvas before the tutorial: we recommend attempting questions before tutorial
- Some questions are samples for tests

Final Exam

- Date: 29 Nov
  - Held in class; open book + notes, but no electronics usage
- Focus is on understanding and application, not facts / memorization
- Example questions
  - Integrative: Require you to combine knowledge from different chapters of the textbook
  - Application: Require you to apply your knowledge of fundamental concepts to reasonably practical scenarios.
  - "Why not": Example: Tommy proposed a solution A to solve problem B. Explain the problem with solution A and how to overcome this problem

What will we learn?

- **We will learn to process different types of data:**
  - Large data volume
  - Graph data
  - Stream data (infinite/never-ending)
- **We will learn to use different models of computation:**
  - MapReduce/Spark
  - Large graph processing engines
  - Streams and online algorithms

What is Data Science?

- **Standard definition**:

"Data science is an interdisciplinary field about processes and systems to extract knowledge or insights from data in various forms."

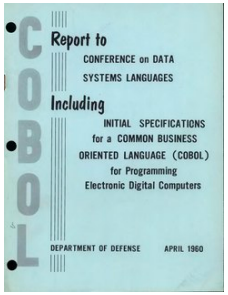
What is Data Science?

- **Standard definition**:

"Data science is an interdisciplinary field about processes and systems to extract knowledge or insights from data in various forms."

- **Historical view**: in 1962, statistician John Tukey described a field called "data analysis", which emerged out of statistics, but focusing not just on testing hypotheses, but on computing aspects of data analysis: including collecting, storing, analyzing, presenting data, etc.

Early transistor computers (TX-0, IBM 608, 7000 series)



Early DBMS, statistical software

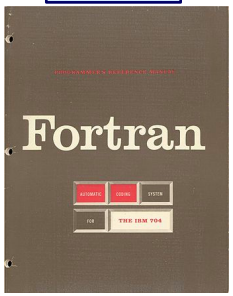
COBOL

1950 UNIVAC I

1957

1960

1962



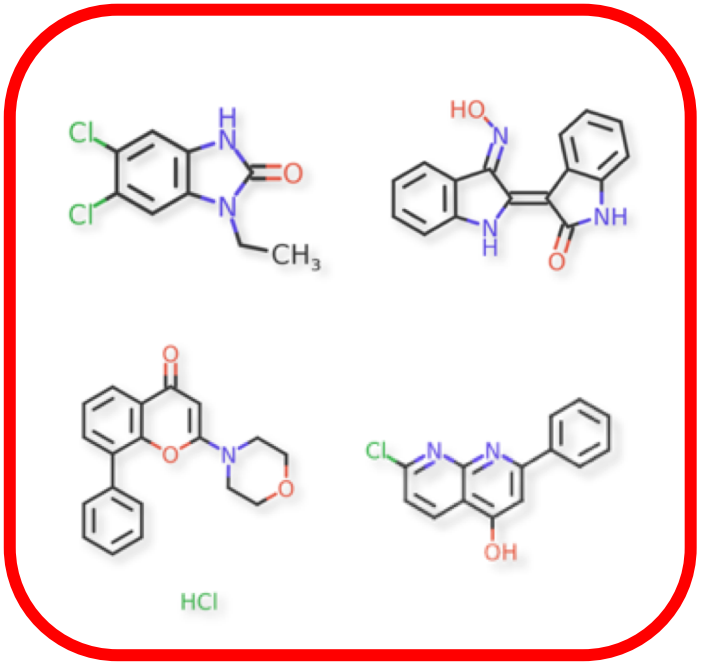
THE FUTURE OF DATA ANALYSIS[1]

BY JOHN W. TUKEY

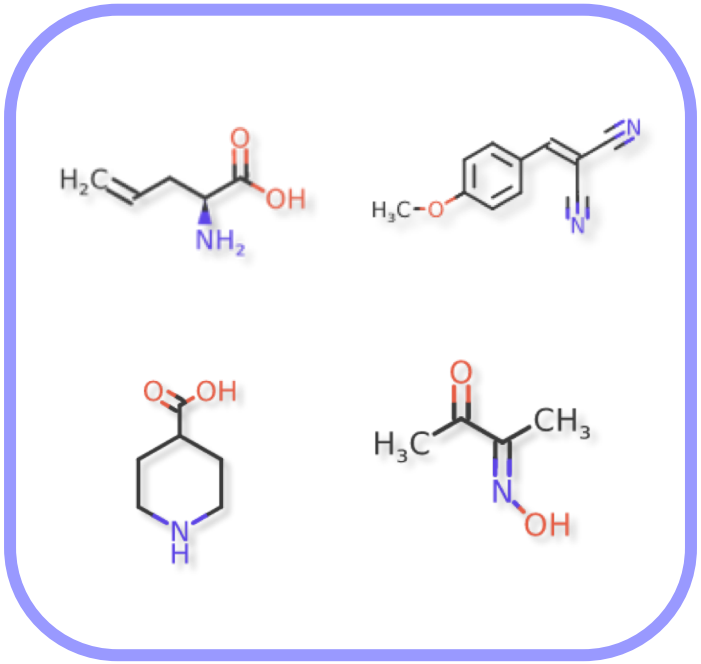
Princeton University and Bell Telephone Laboratories

Q: What rule characterizes toxic molecules?

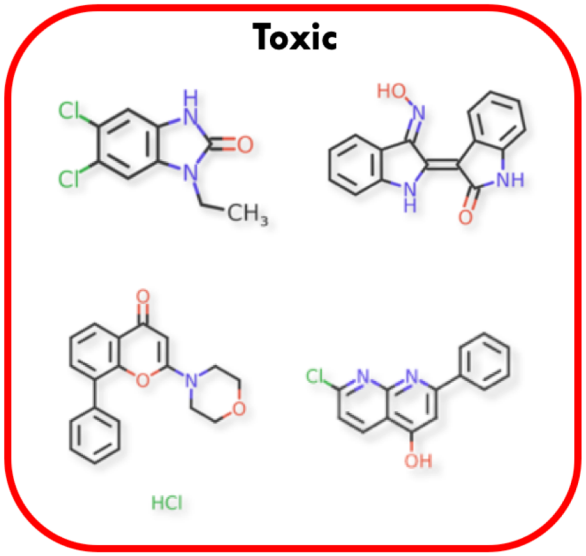
Toxic

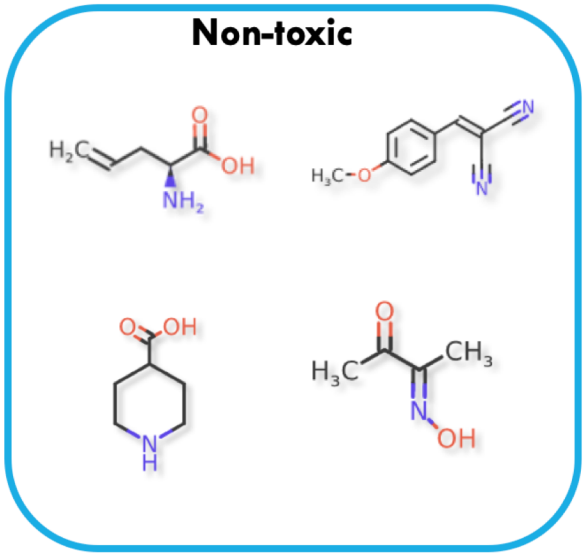


Non-toxic



Q: What rule characterizes toxic molecules?



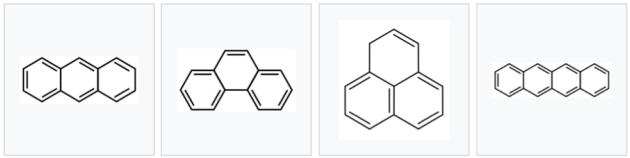


if  > 1:
    toxic
else
    nontoxic

Polycyclic aromatic hydrocarbon

From Wikipedia, the free encyclopedia

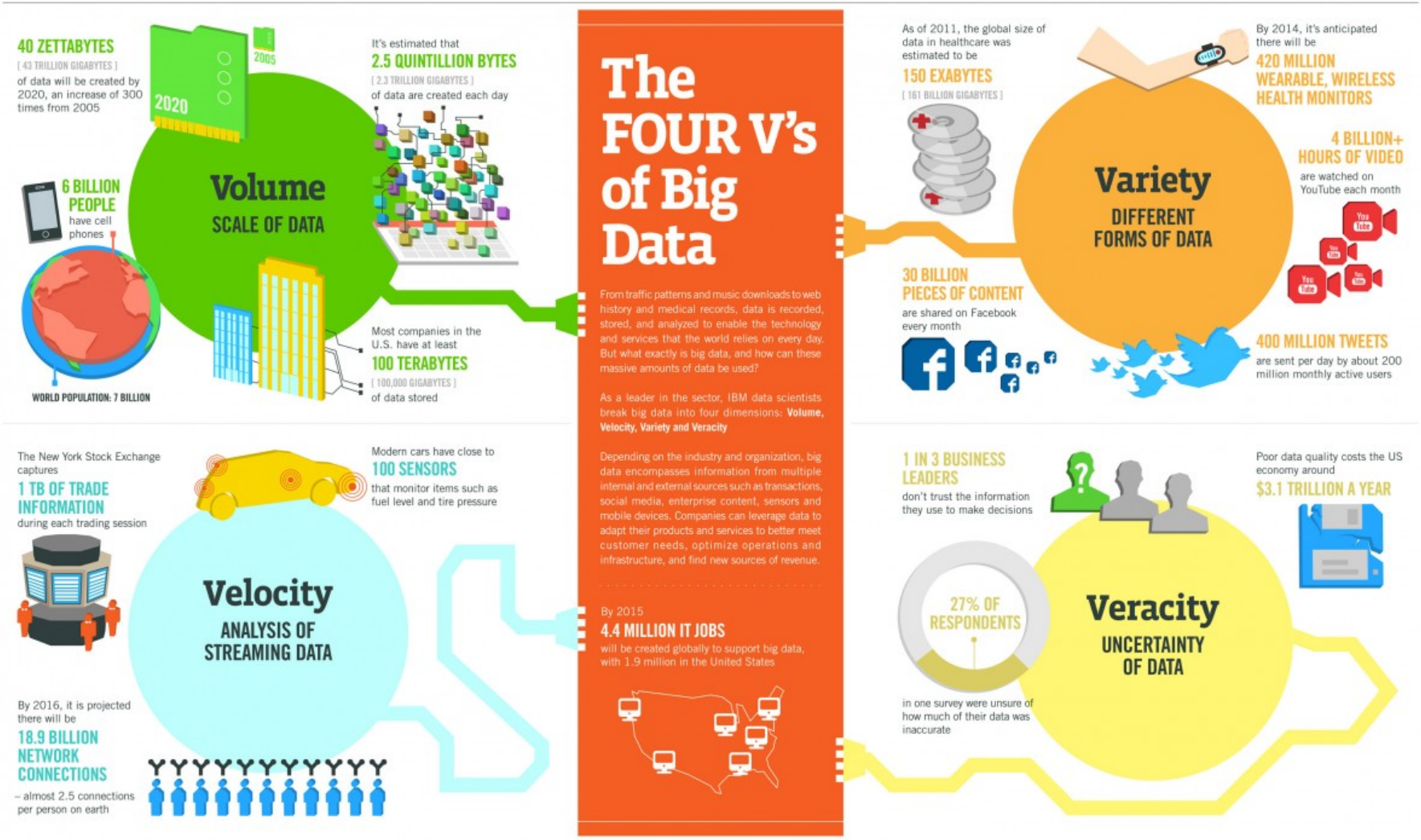
Principal PAH Compounds



Cancer [edit]

PAHs have been linked to skin, lung, bladder, liver, and stomach cancers in well-established animal model studies.[72] human carcinogens are identified in the section "Regulation and Oversight" below.

Challenges of Big Data: the 4 'V's



Veracity: ~3.3% of data in some of the most popular datasets are mislabelled

Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks

Curtis G. Northcutt* ChipBrain, MIT
Anish Athalye MIT
Jonas Mueller Amazon

[1] Northcutt, Curtis G., Anish Athalye, and Jonas Mueller. "Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks." NeurIPS 2021

Veracity: Importance of Data Quality

Forbes

ENTERPRISE & CLOUD

Andrew Ng Launches A Campaign For Data-Centric AI

Data is eating the world so Andrew Ng wants to make sure we radically improve its quality. "Data is food for AI," says Ng, and he is launching a campaign to shift the focus of AI practitioners from model/algorithm development to the quality of the data they use to train the models.

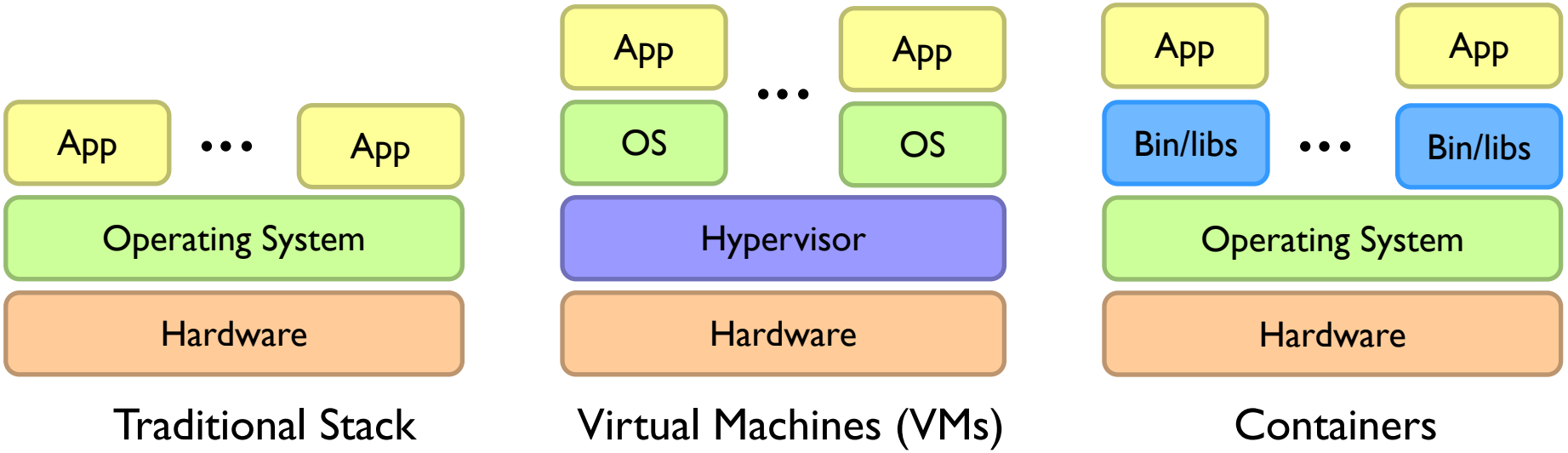
Source: Wikipedia (Clouds)

Utility Computing

- What?
  - Computing resources as a metered service ("pay as you go")
  - Ability to dynamically provision virtual machines
- Why?
  - Scalability: "infinite" capacity
  - Elasticity: scale up or down on demand

Thomas J. Watson (attributed?)

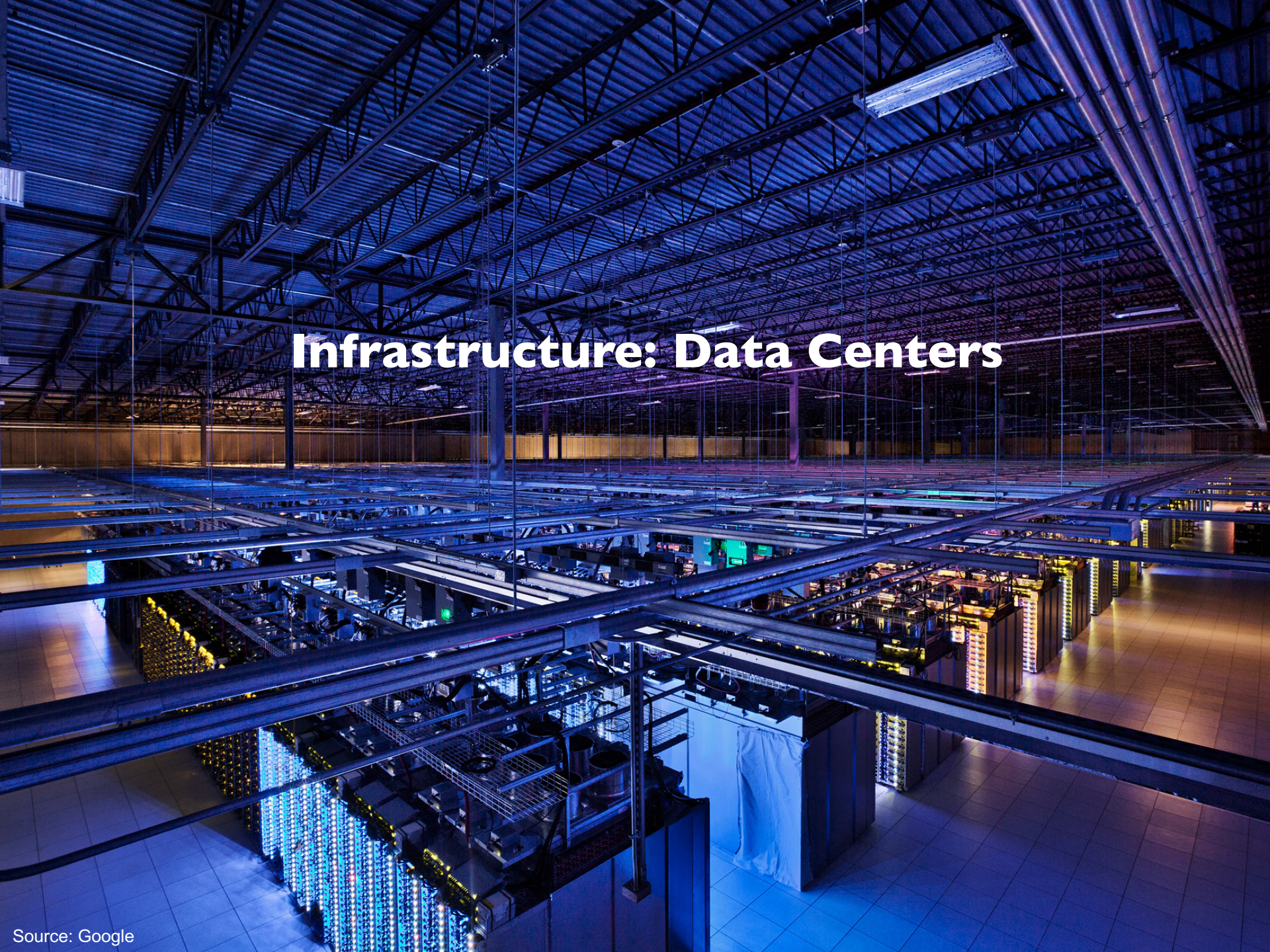
Enabling Technology: Virtualization and Containers



- **Virtual Machines**: enable sharing of hardware resources by running each application in an isolated virtual machine.
  - **High overhead** as each VM has its own OS.
- **Containers**: enable lightweight sharing of resources, as applications run in an isolated way, but still share the same OS.
  - A container is a **lightweight software package** that encapsulates an application and its environment.

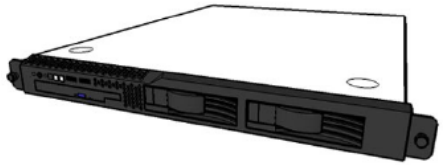
Everything as a Service

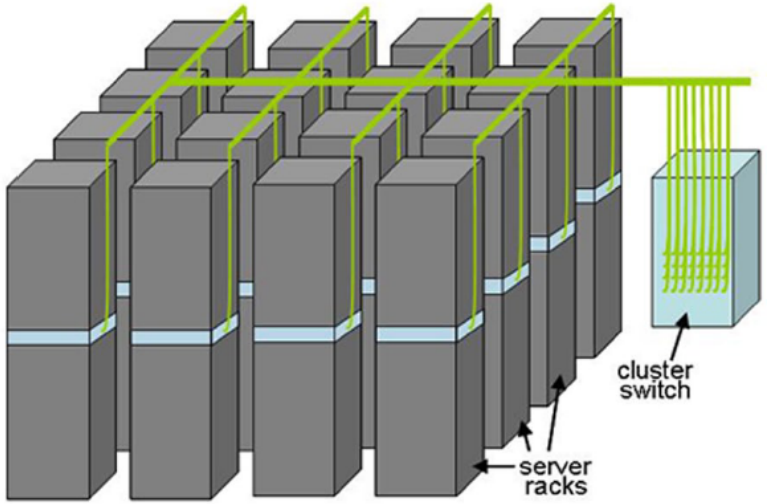
- Infrastructure as a Service (IaaS): Utility Computing
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, Rackspace, Google Compute Engine
- Platform as a Service (PaaS)
  - Provides hosting for web applications and takes care of the hardware maintenance, upgrades, …
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Dropbox, Zoom



Source: Google

Building Blocks



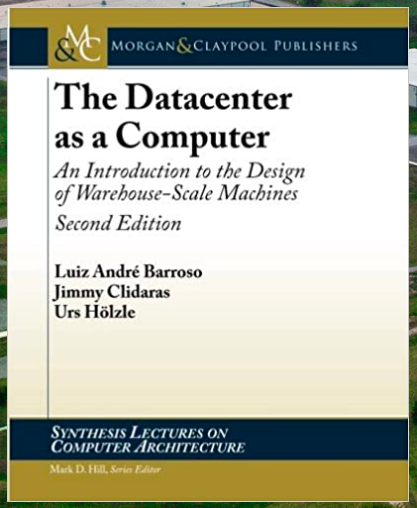


Source: Barroso and Urs Hölzle (2009)

Source: Google

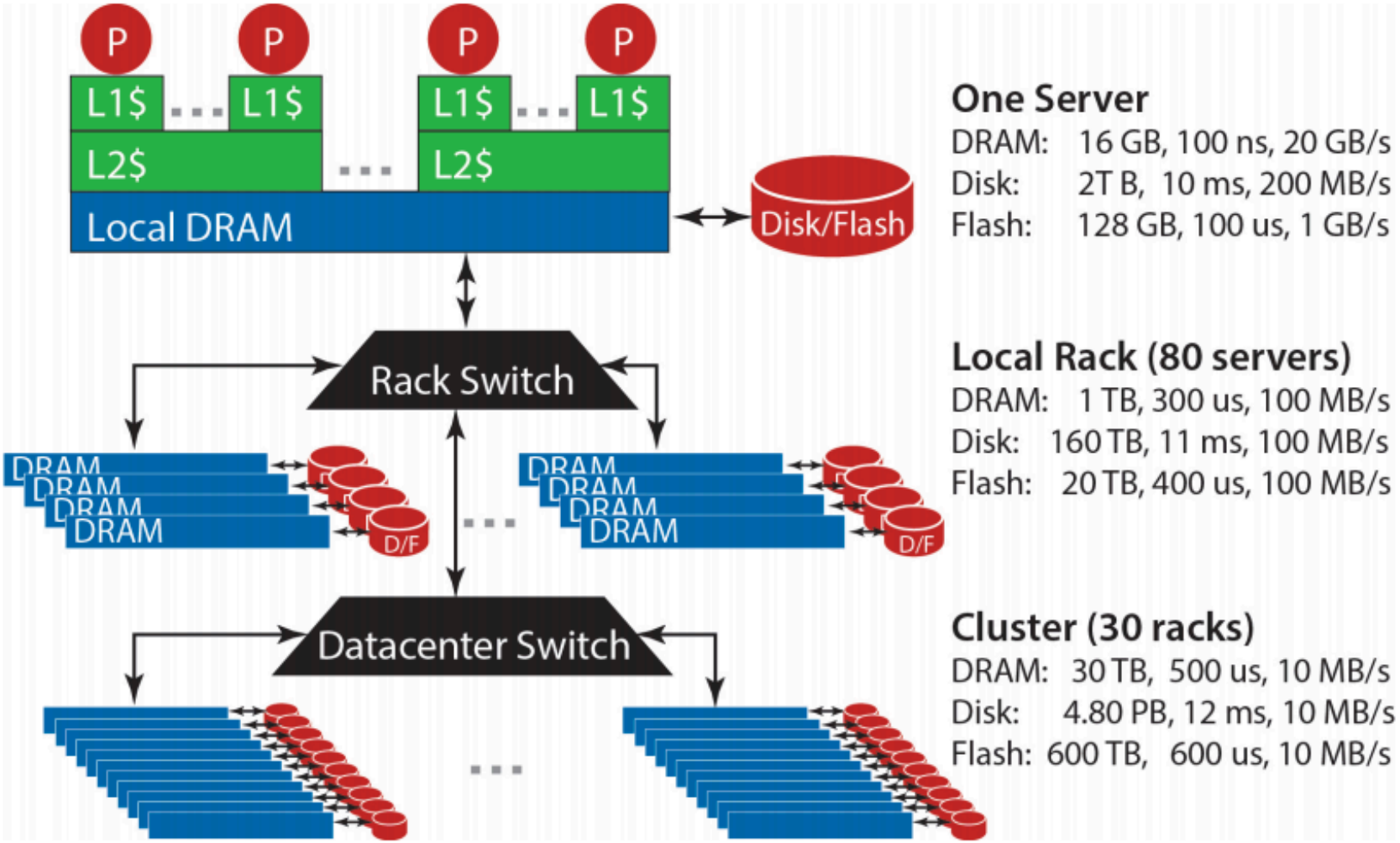
Key Idea: the datacenter *is* the computer

(Meaning: we can think of the machines in a data center as a big "processing unit" being used to solve a problem, rather than just as independent machines)



Source: Google

Storage Hierarchy



Source: Barroso and Urs Hölzle (2013)

Bandwidth vs Latency

- **Bandwidth**: maximum amount of data that can be transmitted per unit time (e.g. in GB/s)
- **Latency**: time taken for 1 packet to go from source to destination (*one-way*) or from source to destination back to source (*round trip*), e.g. in ms
- When transmitting a *large* amount of data, bandwidth tells us roughly how long the transmission will take.
- When transmitting a very *small* amount of data, latency tells us how much delay there will be.
- **Throughput** is similar to bandwidth, but instead of referring to capacity, it refers to the rate at which some data was *actually transmitted* across the network during some period of time.

Low Bandwidth

High Bandwidth

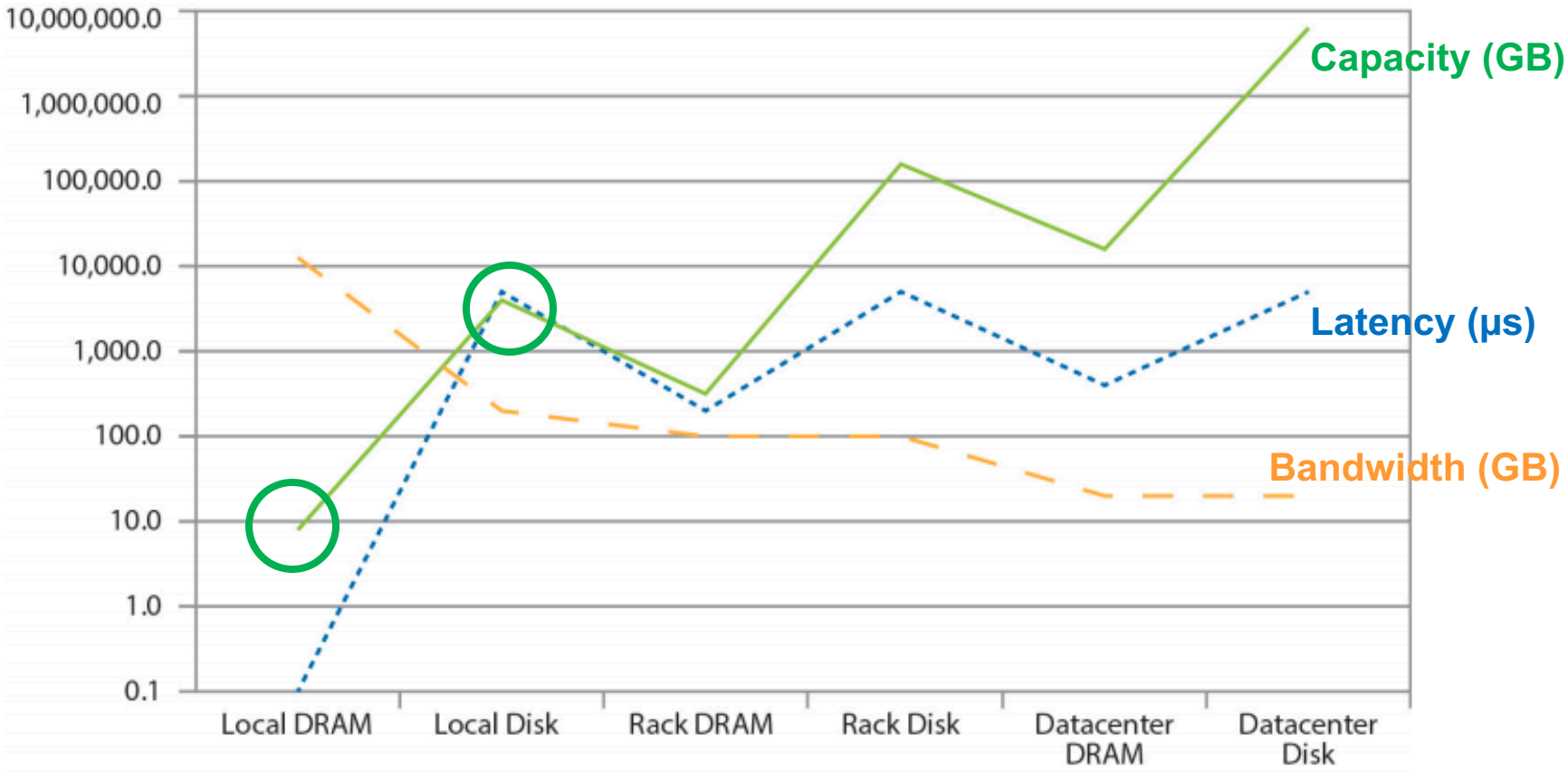
Low Latency

High Latency

Adding many lanes to a highway: increases bandwidth, but does not decrease latency

Credit: Katie Hempenius (via Jake Archibald)

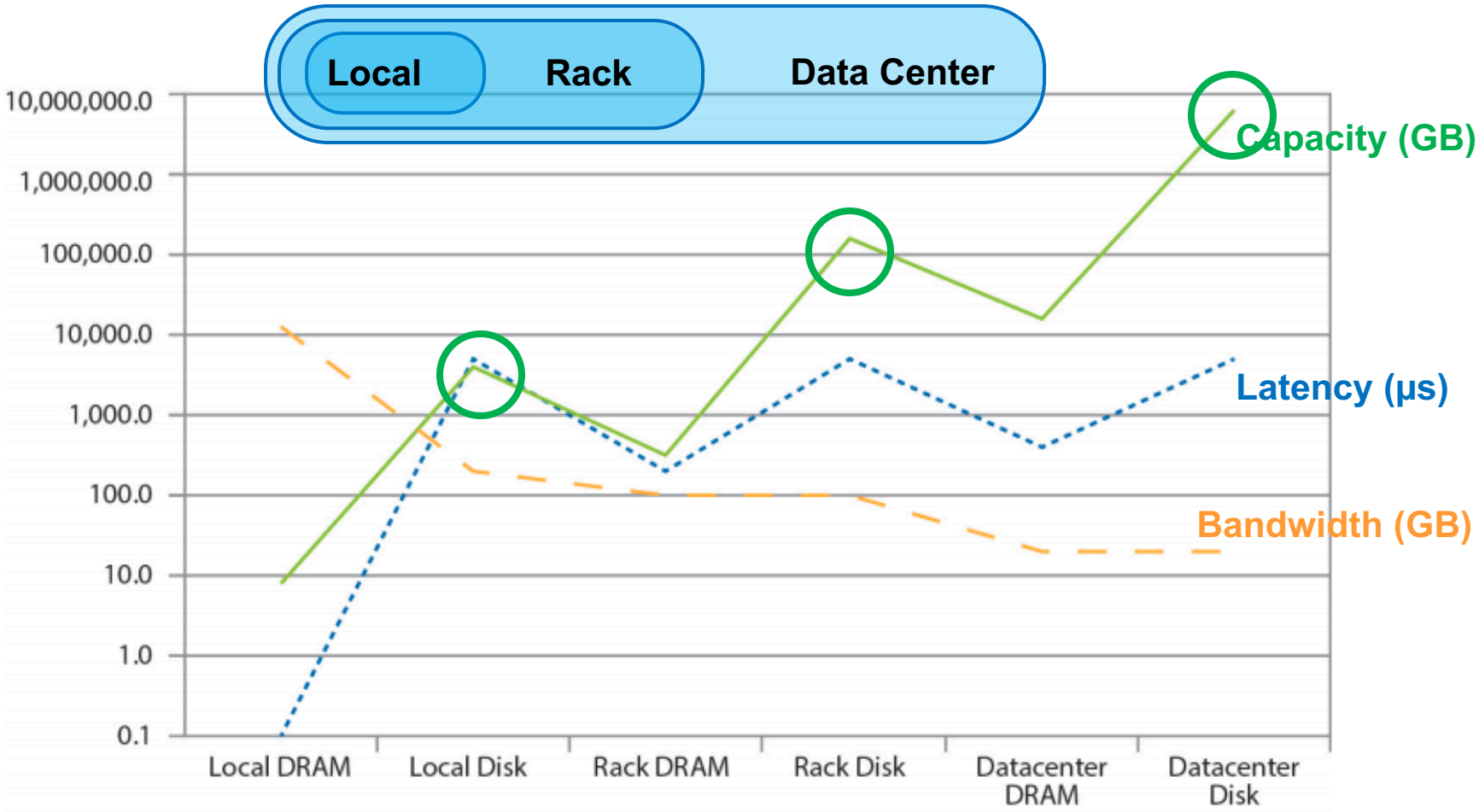
Capacity of Storage Hierarchy



1. **Disk has much higher capacity than DRAM**

Source: Barroso and Urs Hölzle (2013)

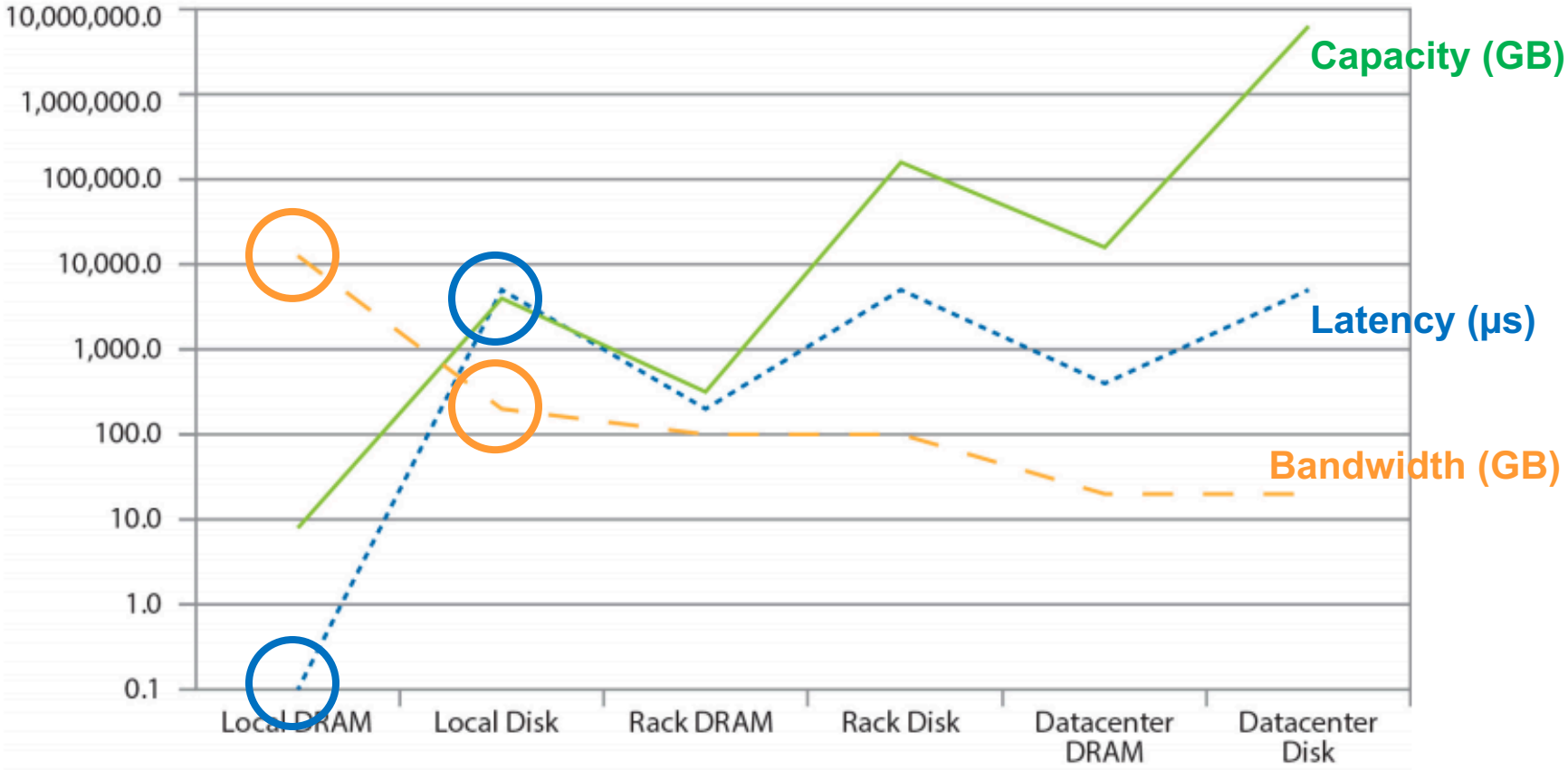
Capacity of Storage Hierarchy



2. **Storage hierarchy**: *capacity* increases as we go from Local Server, to Rack, to Datacenter.

Source: Barroso and Urs Hölzle (2013)

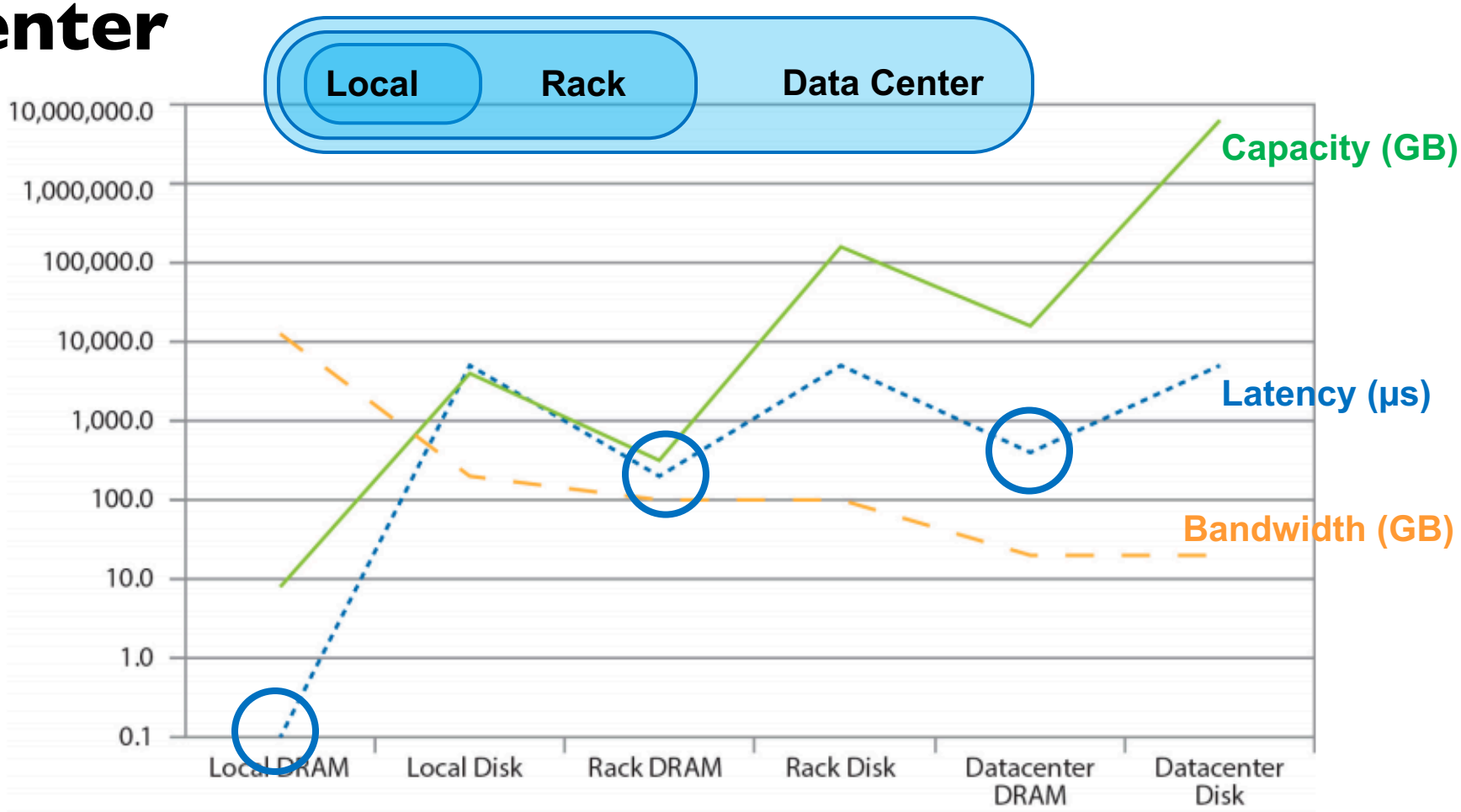
Speed of Moving Data Around Data Center



3. **Disk reads are much more expensive than DRAM**, both in terms of with **higher latency** and **lower bandwidth**.

Source: Barroso and Urs Hölzle (2013)

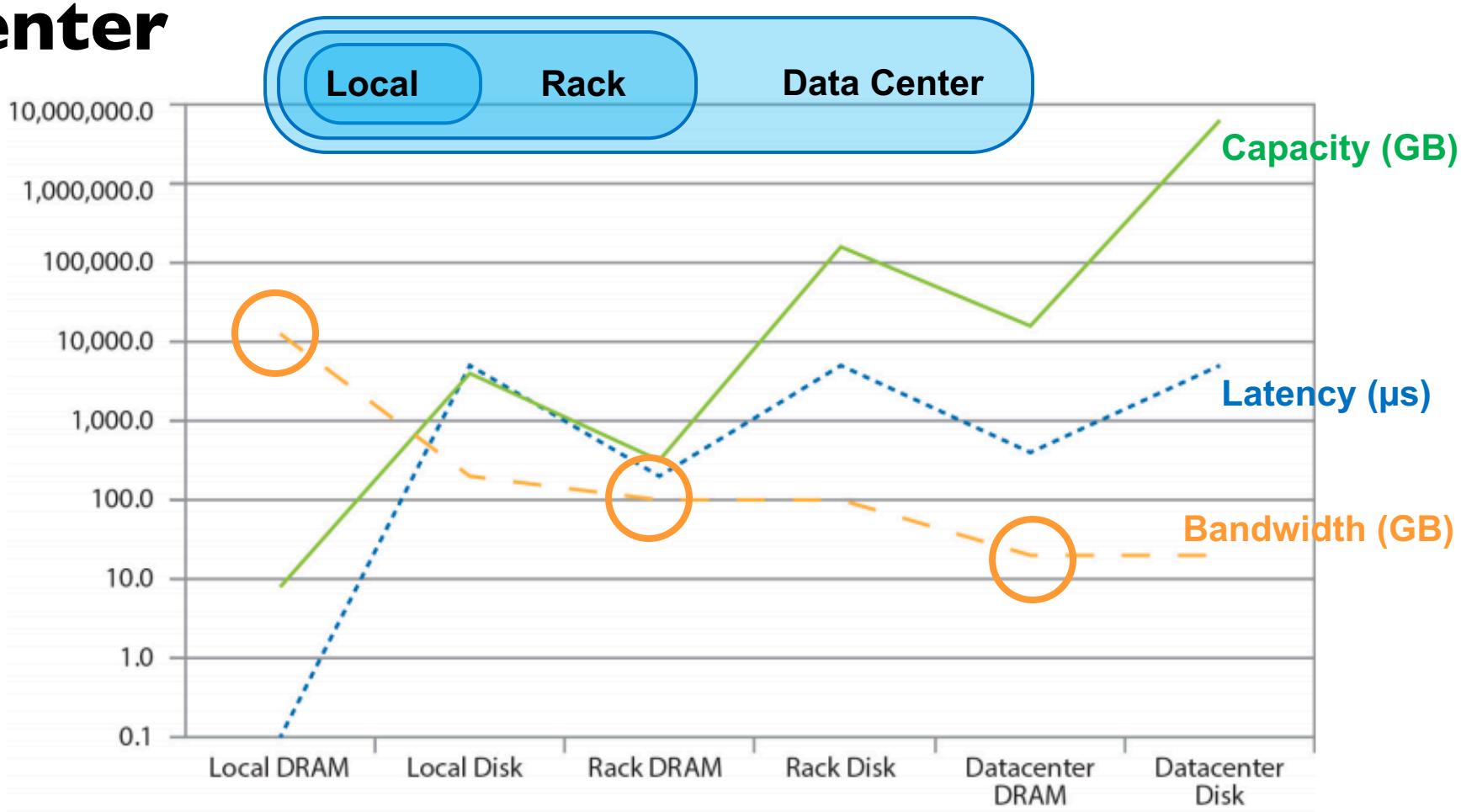
Speed of Moving Data Around Data Center



4. **Costs increase over the storage hierarchy**: **latency increases** as we go from Local to Rack to Datacenter.

Source: Barroso and Urs Hölzle (2013)

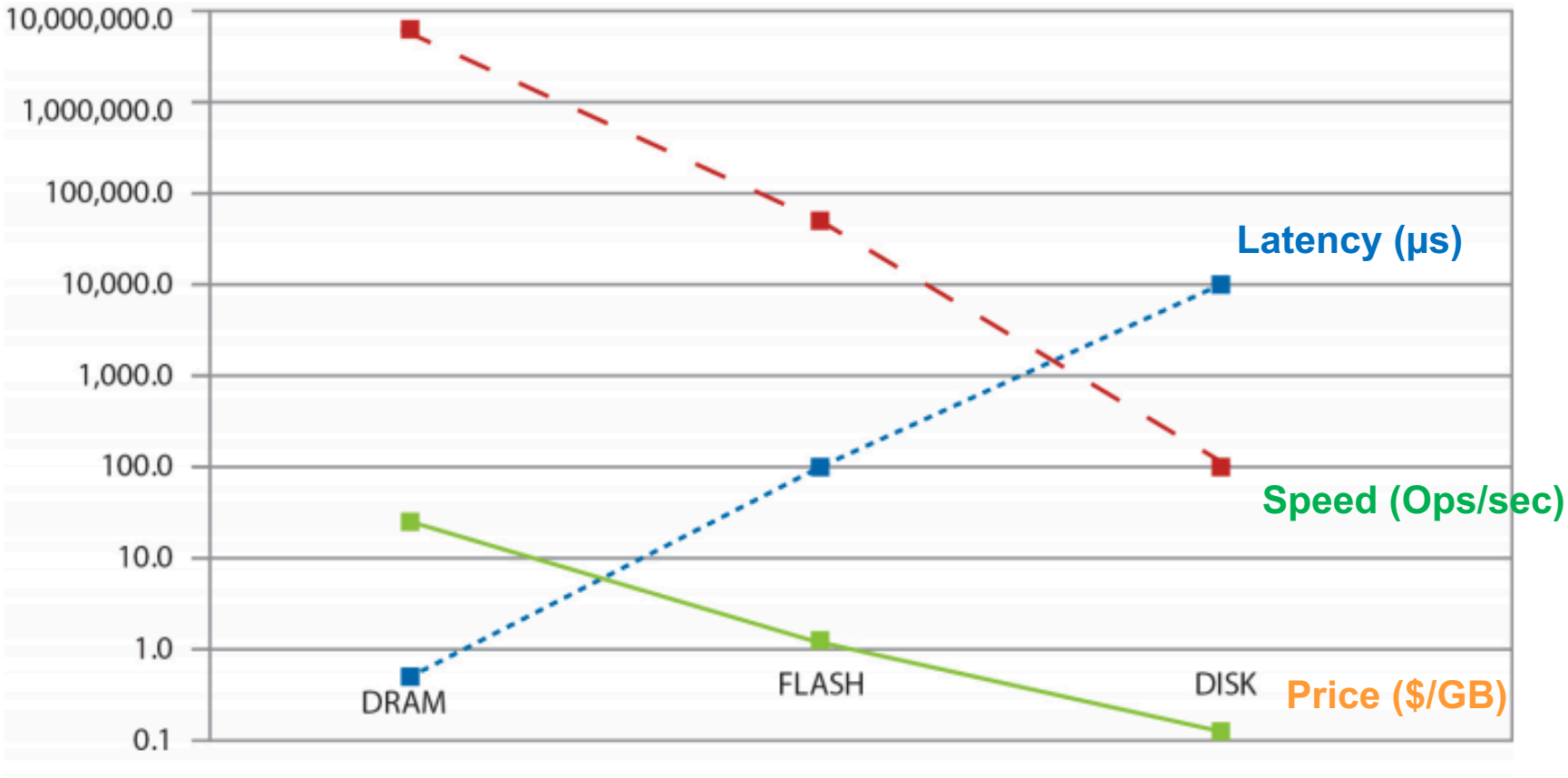
Speed of Moving Data Around Data Center



5. **Costs increase over the storage hierarchy**: **bandwidth decreases** as we go from Local to Rack to Datacenter.

Source: Barroso and Urs Hölzle (2013)

Price vs Speed Tradeoff



DRAM: fast but most expensive

DISK: slow but cheapest

Source: Barroso and Urs Hölzle (2013)

"Big Ideas" of Massive Data Processing in Data Centers

- Scale "out", not "up"
  - scale 'out' = combining many cheaper machines; scale 'up': increasing the power of each individual machine
  - Also called 'horizontal' vs 'vertical' scaling
- Move processing to the data
  - Clusters have limited bandwidth: we should move the task to the machine where the data is stored
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - E.g. if processing a certain dataset takes 100 machine hours, ideal scalability is to use a cluster of 10 machines to do it in about 10 hours.

Q: What type of application domains are you most interested in?

1. Business
2. Scientific
3. Biomedicine
4. Engineering
5. Finance / Economics
6. Web / E-commerce
7. Education
8. Other

MapReduce

a. **Motivation**

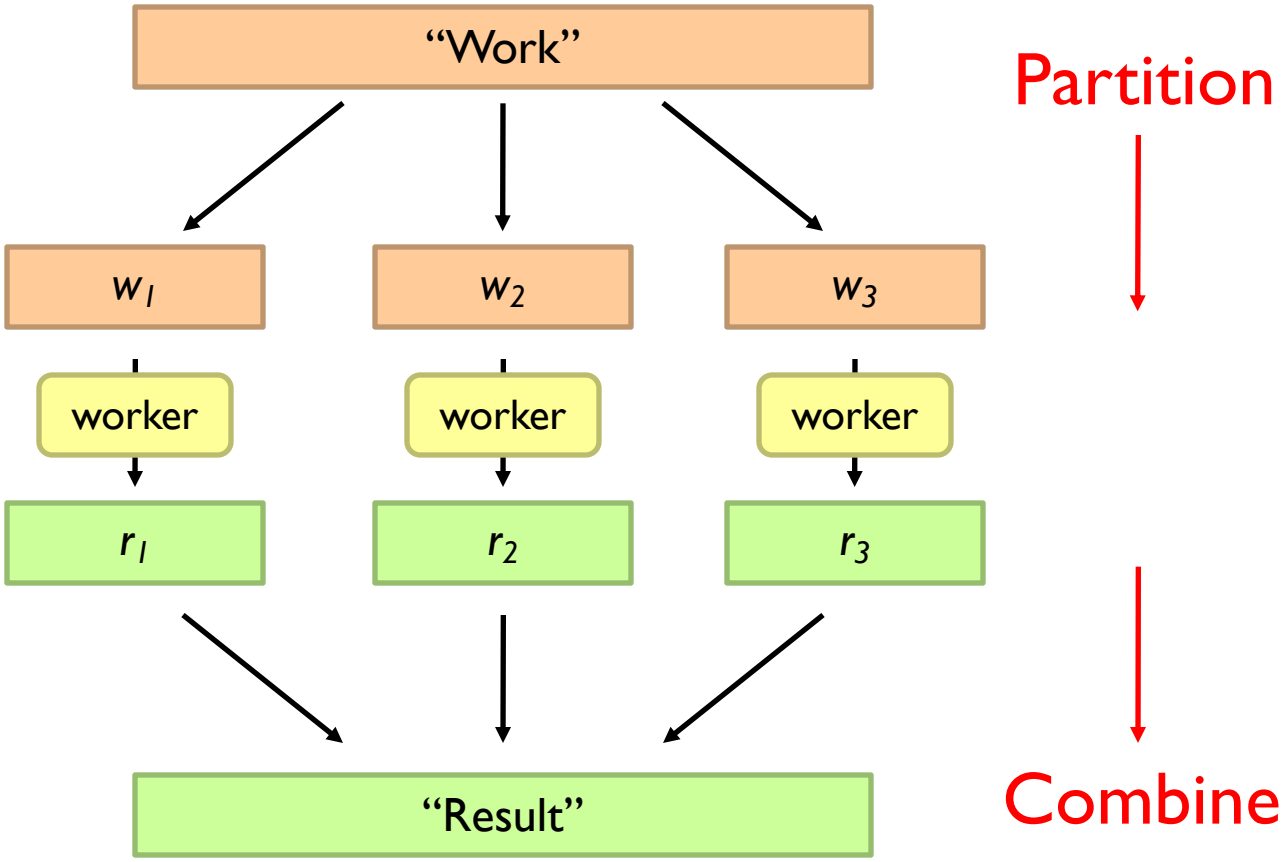
b. Basic MapReduce

Source: Google

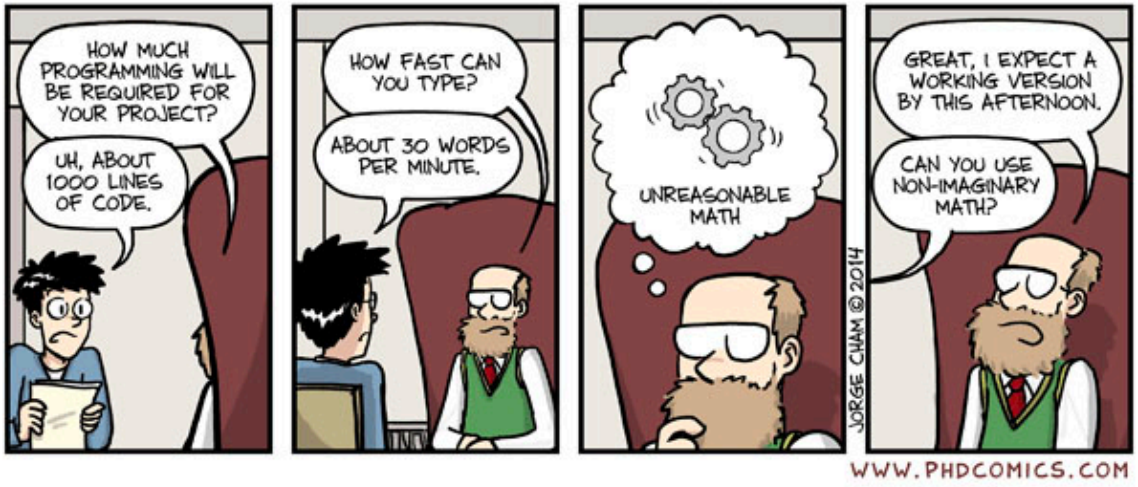
Motivation: Google Example

- 20+ billion web pages: ~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!
- Infrastructure: cloud + data centers
  - Cluster of commodity nodes
  - Commodity network (ethernet) to connect them
- Solution:
  - Parallelization + divide and conquer

Divide and Conquer



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/fail?

Challenge 1: Machine Failures

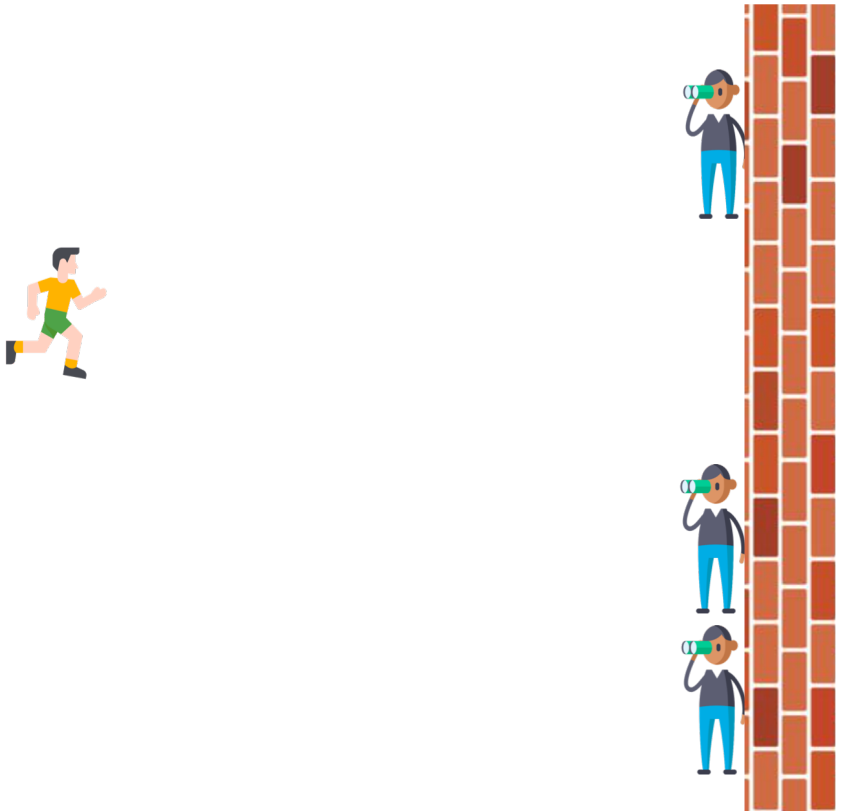
- One server may stay up 10 years (~3650 days)
- If you have 3650 servers, expect to loose 1/day
- People estimated Google had ~2.5 million machines in 2016
  - ~700 machines fail every day!

Challenge 2: Synchronization

- 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 control mechanisms, such as **barriers**

Barrier

- Simple tool used when multiple processes are running at the same time
- It ensures that every process must stop at this point until all processes have reached the barrier



Challenge 3: Programming Difficulty

- Concurrency is 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

The datacenter *is* the computer

- It's all about the right level of abstraction
  - Moving beyond the single machine architecture
  - What's the "instruction set" (or "API") of the datacenter computer?
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
  - No need to explicitly worry about reliability, fault tolerance, etc.
- Separating the *what* from the *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

MapReduce

a. Motivation

b. **Basic MapReduce**

Source: Google

Typical Big Data Problem

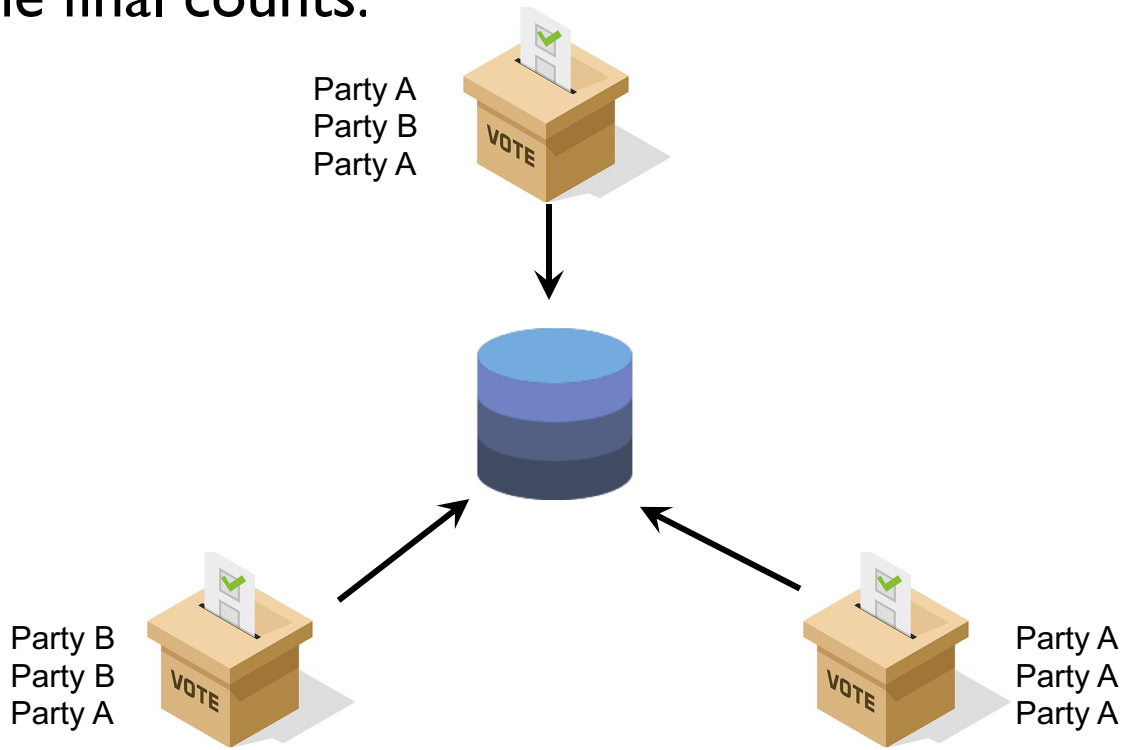
- Iterate over a large number of records
- Extract something of interest from each **Map**
- Shuffle and sort intermediate results **Shuffle**
- Aggregate intermediate results **Reduce**
- Generate final output

Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

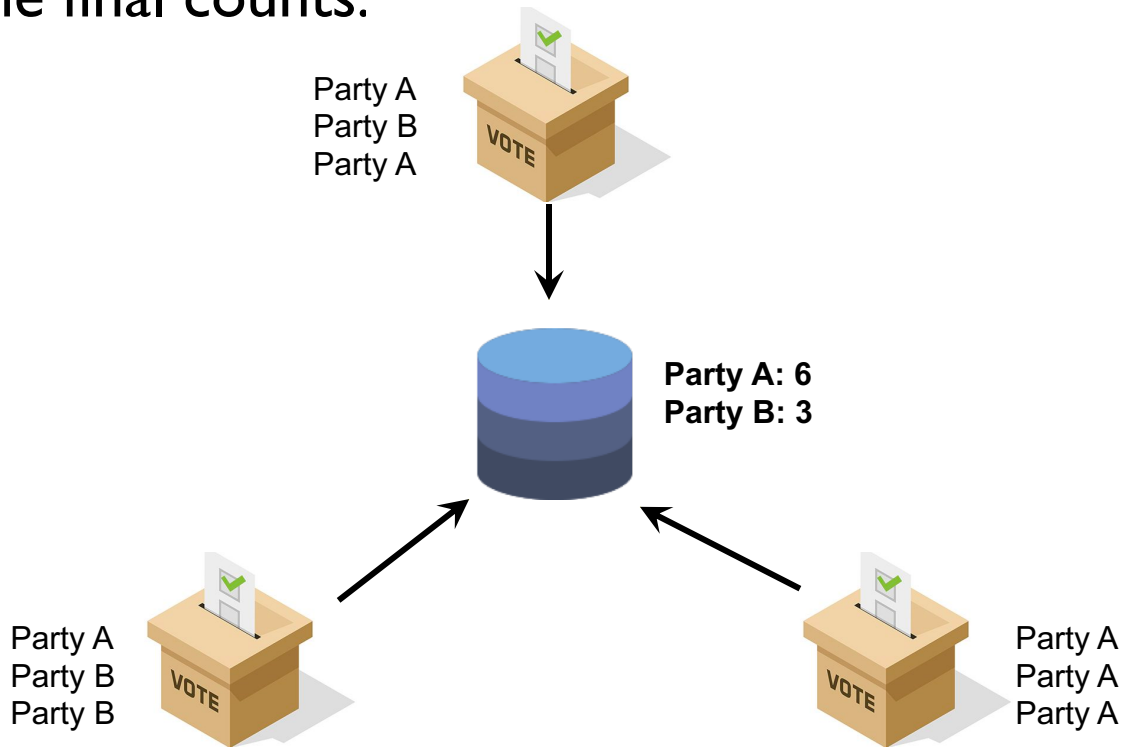
Basic Example: Tabulating Election Results from Multiple Polling Sites

- Imagine you are hired to develop the software for Singapore's election counting system.
- You need to **aggregate** vote counts from multiple stations into the final counts.



Basic Example: Tabulating Election Results from Multiple Polling Sites

- Imagine you are hired to develop the software for Singapore's election counting system.
- You need to **aggregate** vote counts from multiple stations into the final counts.



Tabulating Election Results: MapReduce

Map

Shuffle

Reduce

Party A
Party B
Party A

Party A
Party B
Party B

Party A
Party A
Party A

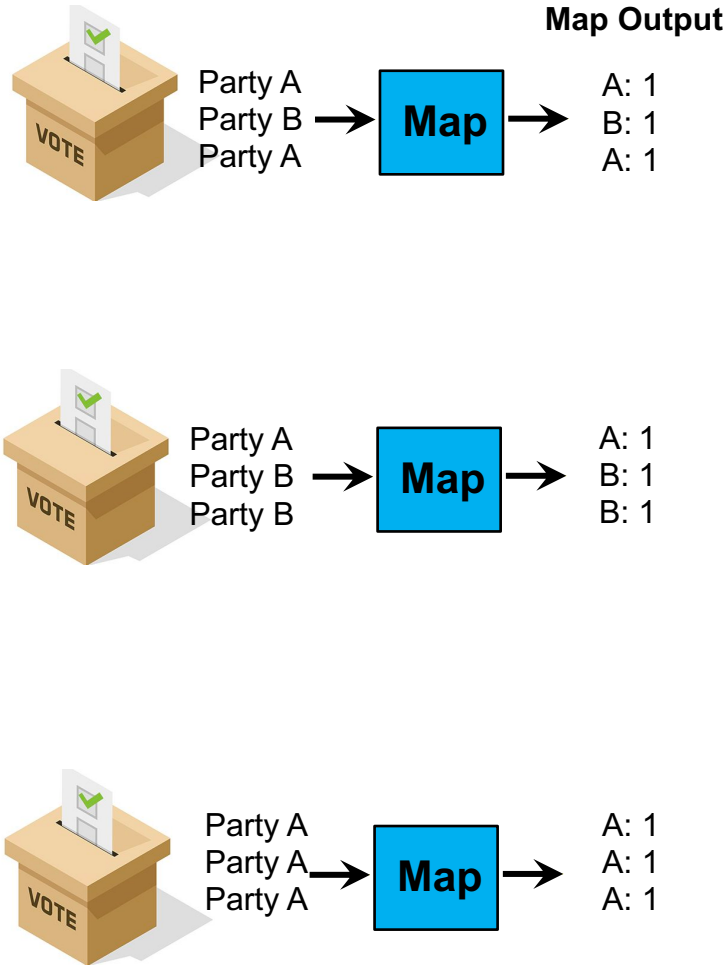
Party A: 6
Party B: 3

Tabulating Election Results: MapReduce

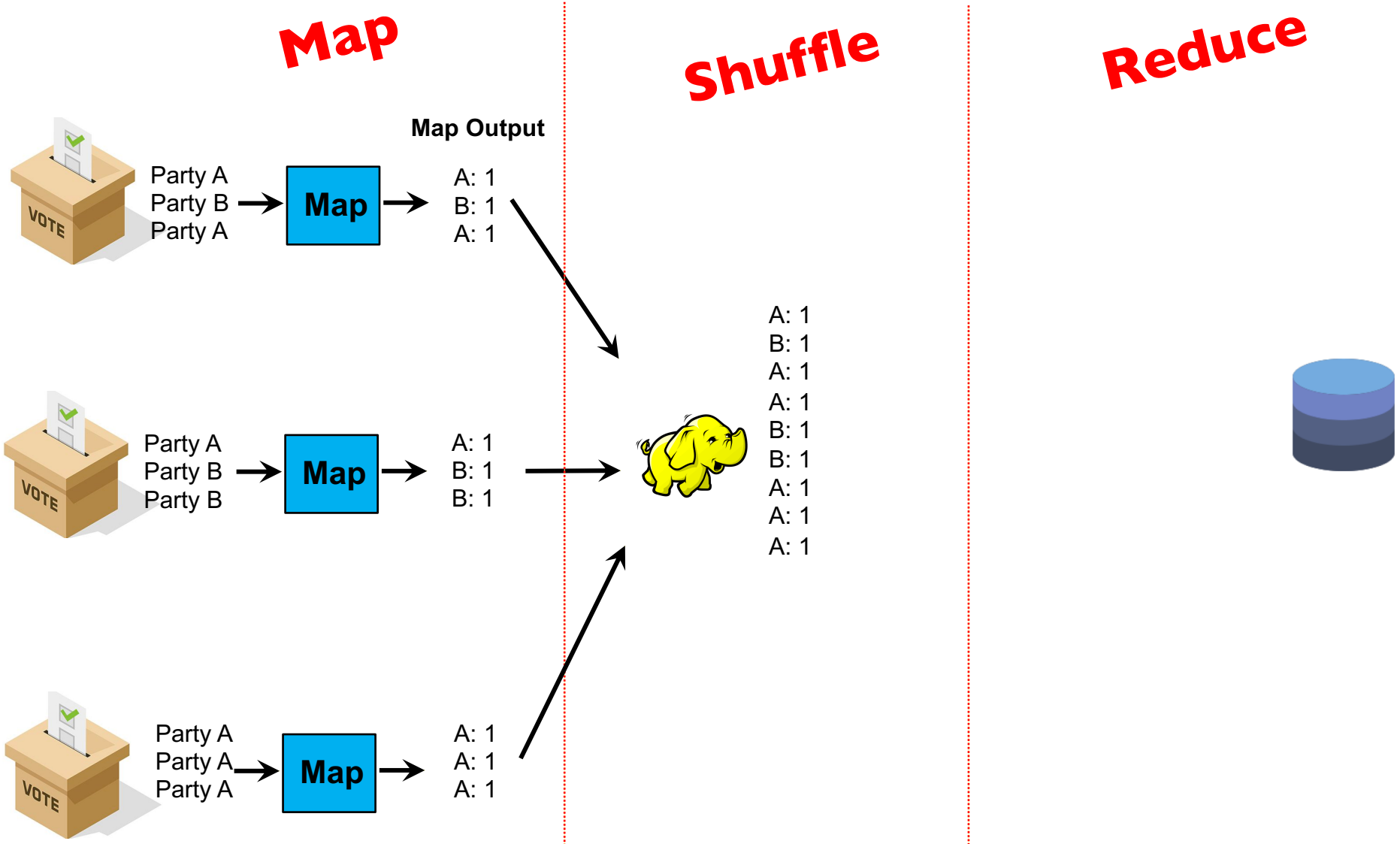
Map

Shuffle

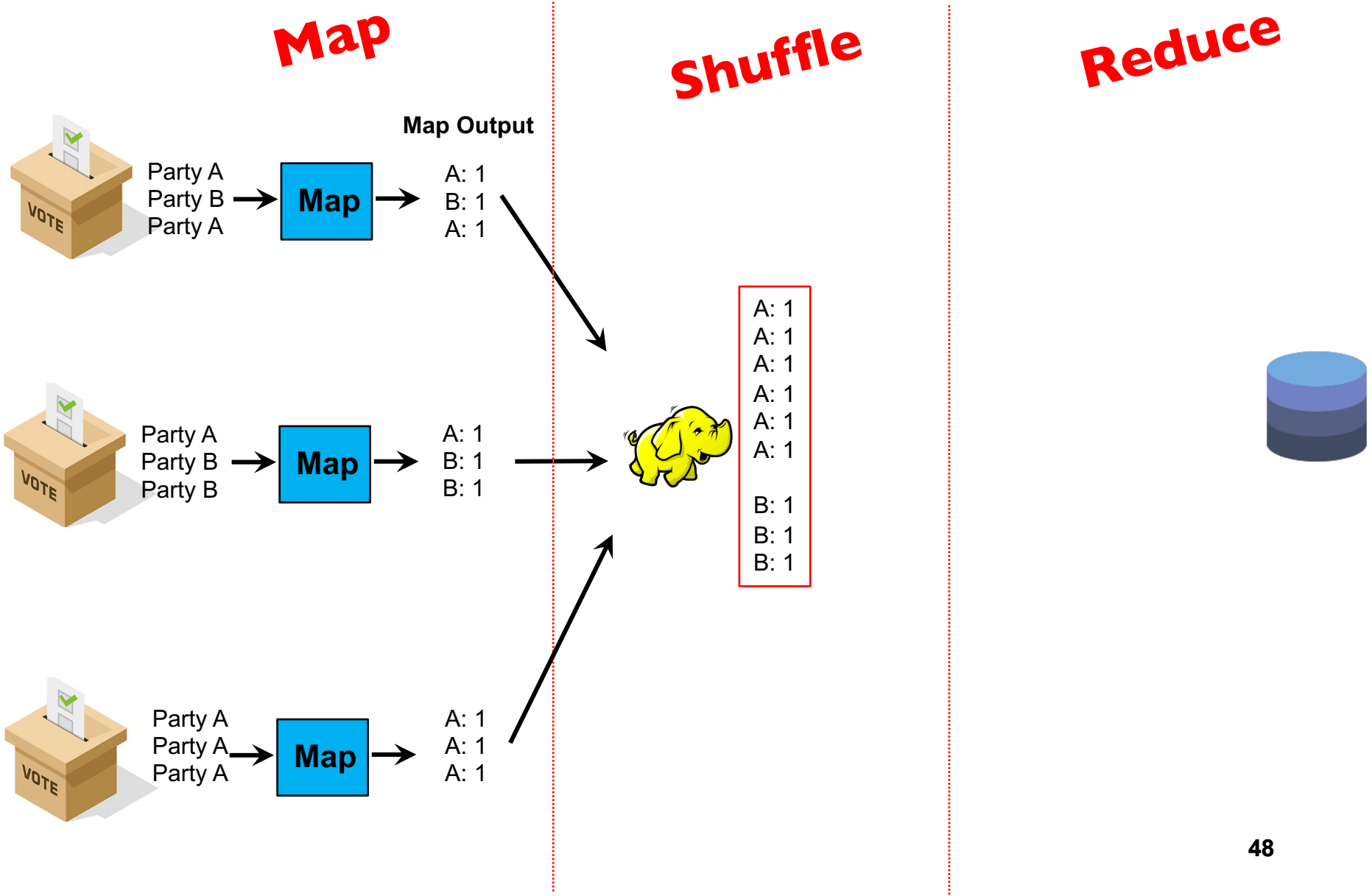
Reduce



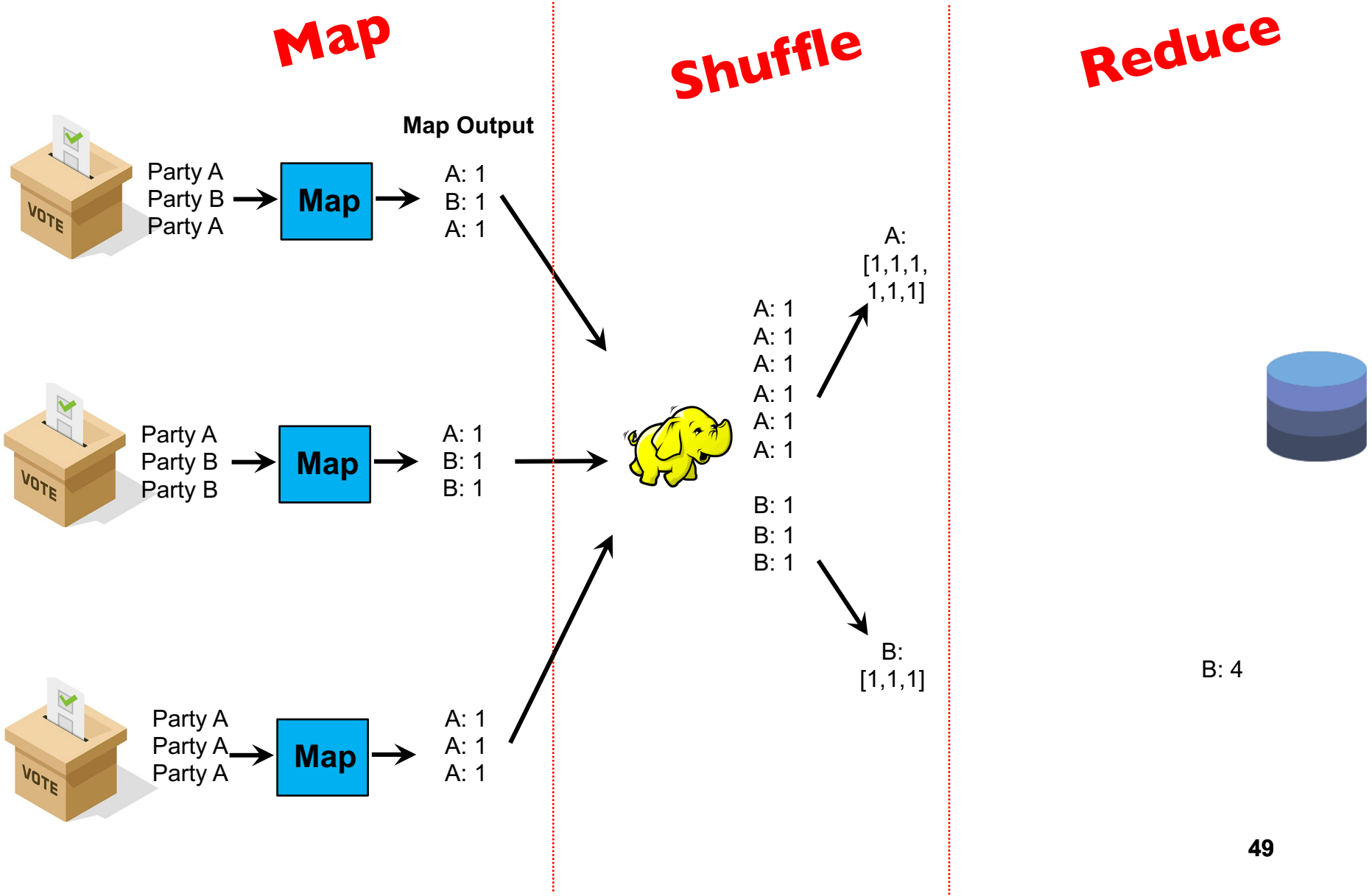
Tabulating Election Results: MapReduce



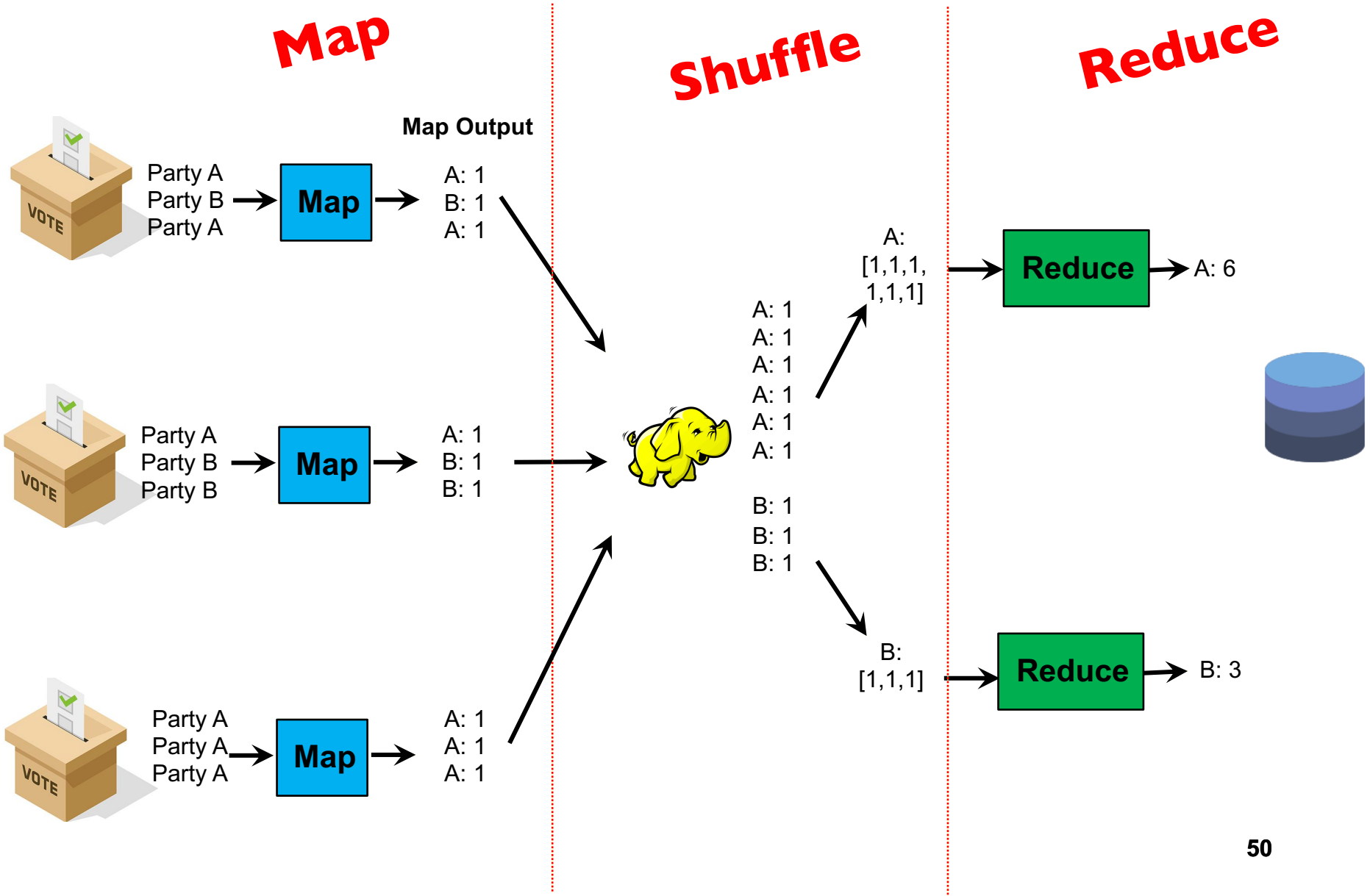
Tabulating Election Results: MapReduce



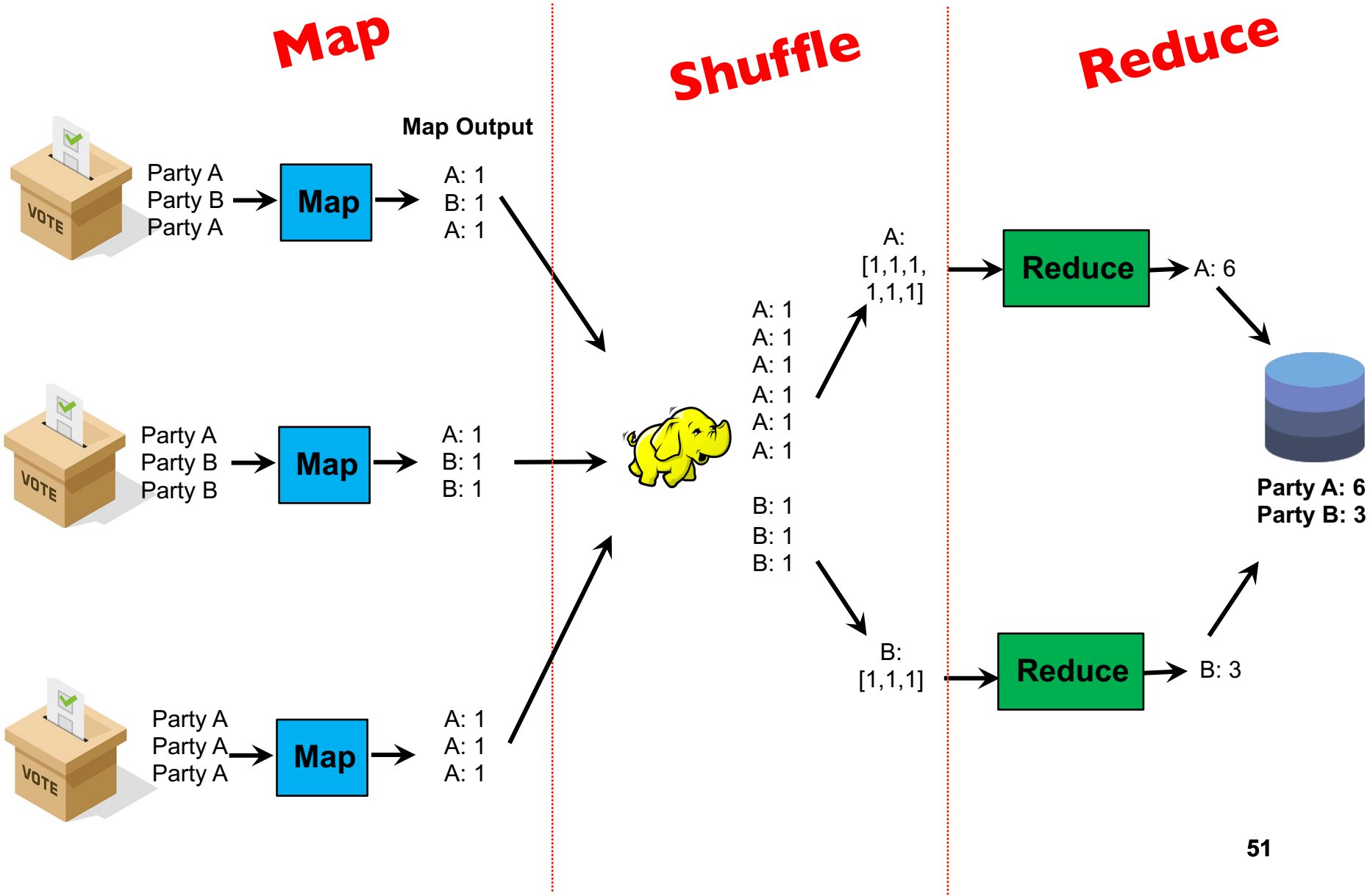
Tabulating Election Results: MapReduce



Tabulating Election Results: MapReduce



Tabulating Election Results: MapReduce



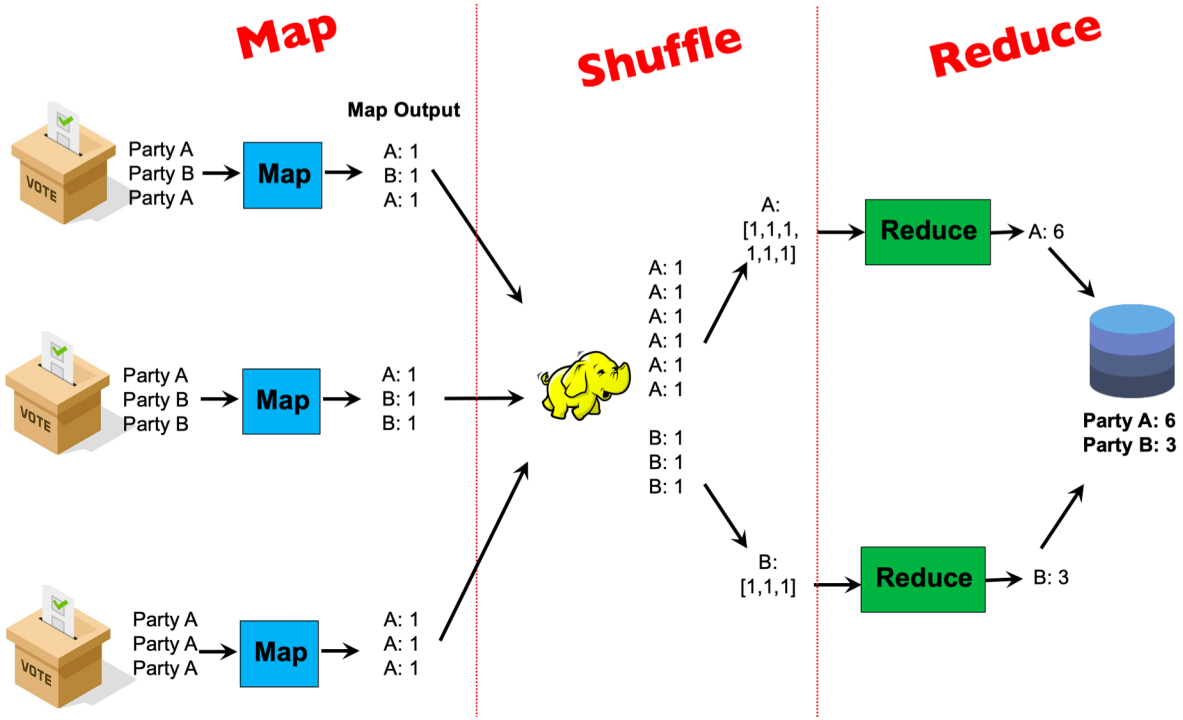
Writing MapReduce Programs

- **Typical Interface:** Programmers specify two functions:

  map $(k_1, v_1) \rightarrow \text{List}(k_2, v_2)$

  reduce $(k_2, \text{List}(v_2)) \rightarrow \text{List}(k_3, v_3)$

  - All values with the same key are sent to the same reducer



Writing MapReduce Programs

- **Typical Interface:** Programmers specify two functions:

  map $(k_1, v_1) \rightarrow \text{List}(k_2, v_2)$

  reduce $(k_2, \text{List}(v_2)) \rightarrow \text{List}(k_3, v_3)$

  - All values with the same key are sent to the same reducer

- The execution framework handles many challenging issues…

  - 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/fail?

MapReduce Execution Framework

- 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 file system (later)