CS 4530 Software Engineering

Module 9: Distributed Systems Principles and Requirements

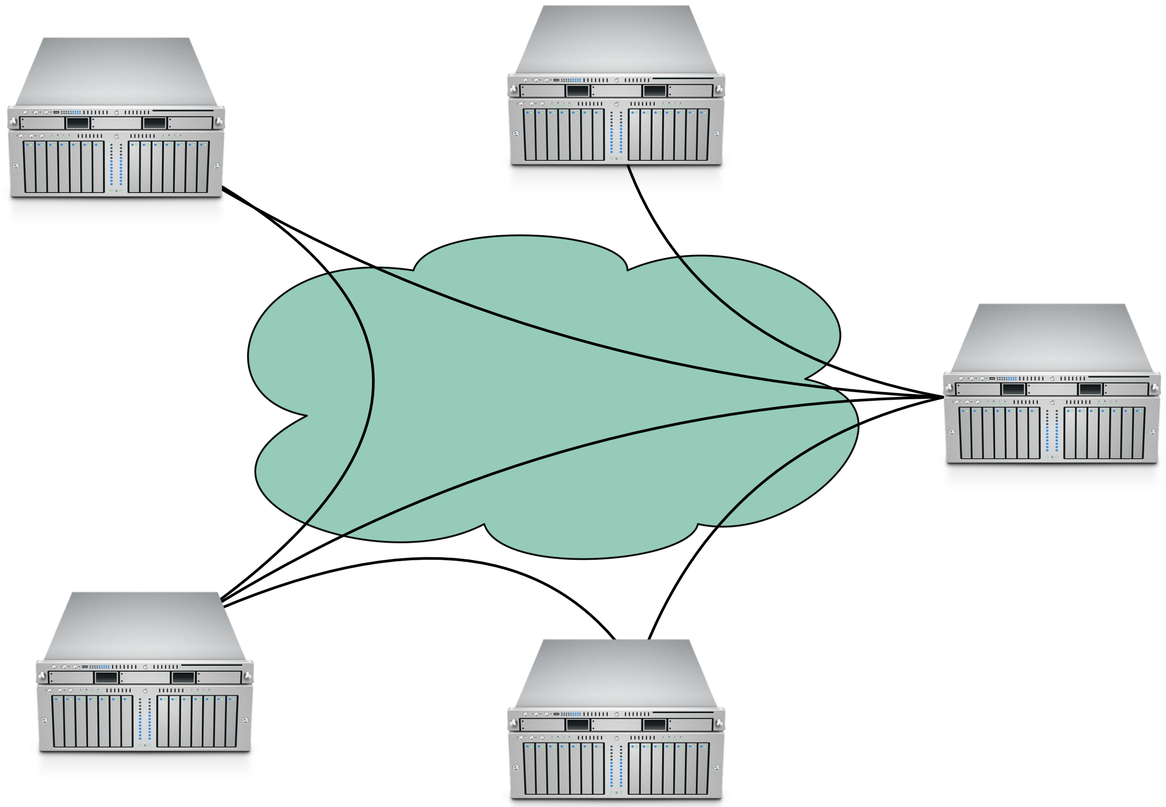
Jonathan Bell, Adeel Bhutta, Mitch Wand
Khoury College of Computer Sciences

Learning Objectives for this Lesson

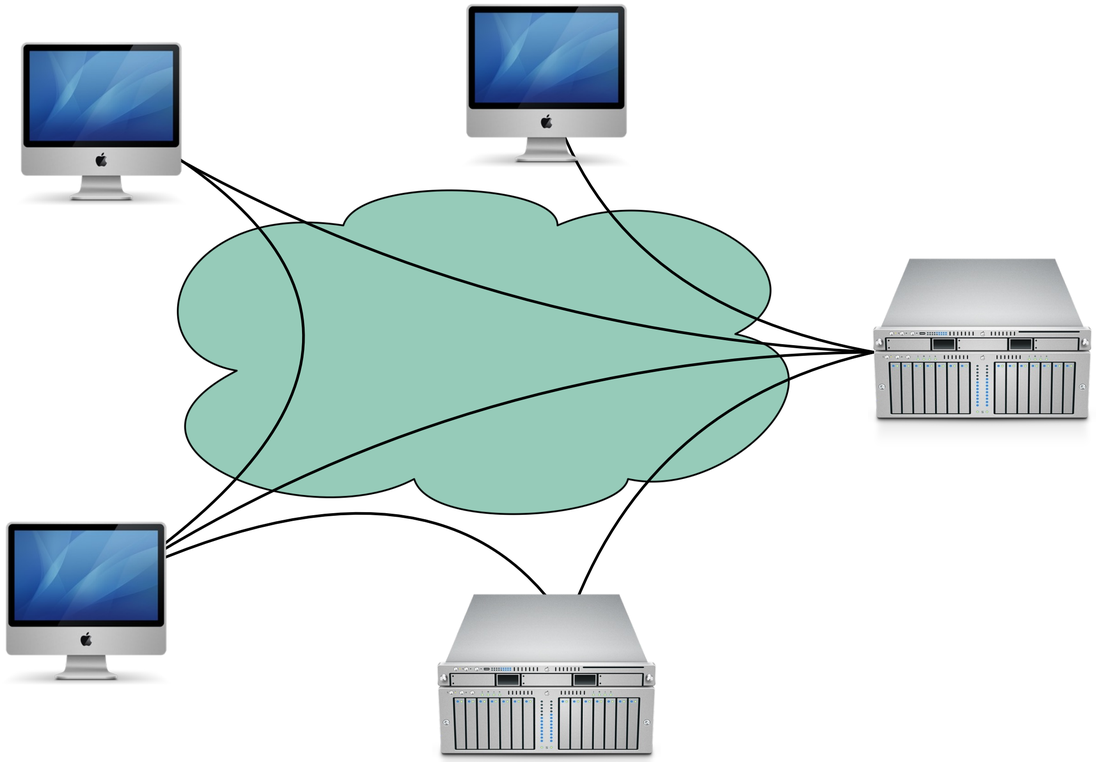
By the end of this lesson, you should be able to…

- Describe 5 key goals of distributed systems
- Understand the fundamental constraints of distributed systems
- Understand the roles of replication and partitioning in the context of the DNS infrastructure

What is a distributed system?



Model:
Many servers talking through a network



Model:
Many servers and clients talking through a network

Why expand to distributed systems?

(Here is why) - performance, availability, fault tolerance

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance

“Distributed Systems for Fun and Profit”, Takada

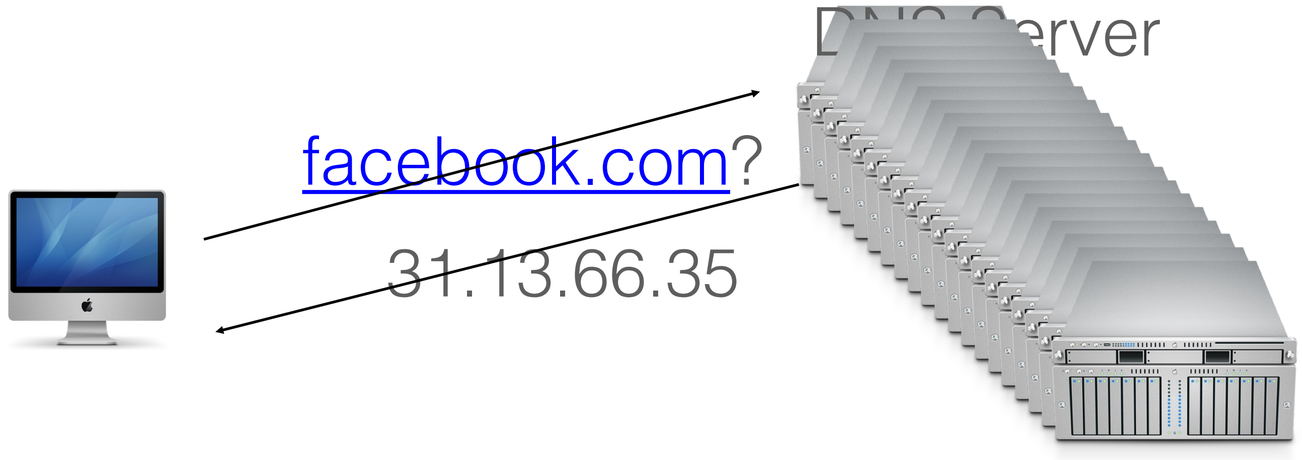
Example: Domain Name System (DNS)

Problem Statement

- Nodes (hosts) on a network are identified by IP addresses
- E.g.: 142.251.41.4
- We humans prefer something easier to remember: calendar.google.com, facebook.com, www.khoury.northeastern.edu
- We need to keep a directory of domain names and their addresses
- We also need to make sure everybody gets directed to the correct host

Example: Domain Name System (DNS)

- Need to handle millions of DNS queries per second
- Not immediately obvious how to scale: how do we maintain replication, some measure of consistency?



Domain Name System

- Strawman solution: Use a Local file
  - Keep local copy of mapping from all hosts to all IPs (e.g., /etc/hosts)
  - Hosts change IPs regularly: Download file frequently
  - Lot of constant internet bandwidth use
  - IPv4 space is now full
    - 32-bits: 4,294,967,296 addresses
    - At 1 byte per address, file would be 4GB
    - Not a lot of disk space (now, DNS introduced in the late 80s)

Domain Name System

- Strawman solution: Use a Local file
  - Keep local copy of mapping from all hosts to all IPs (e.g., /etc/hosts)
  - Hosts change IPs regularly: Download file frequently
  - Lot of constant internet bandwidth use
  - IPv4 space is now full
    - 32-bits: 4,294,967,296 addresses
    - At 1 byte per address, file would be 4GB
    - Not a lot of c ed

We need 200x of these to hold 4GB: $270K+


your Tandy 1000s/3000/4000 or PC Compatible on a user-installable card. Includes manual and software for easy installation. The easy way to get hard disk power for your computer system. **25-4059** **799.00**

Add an External Hard Disk Drive

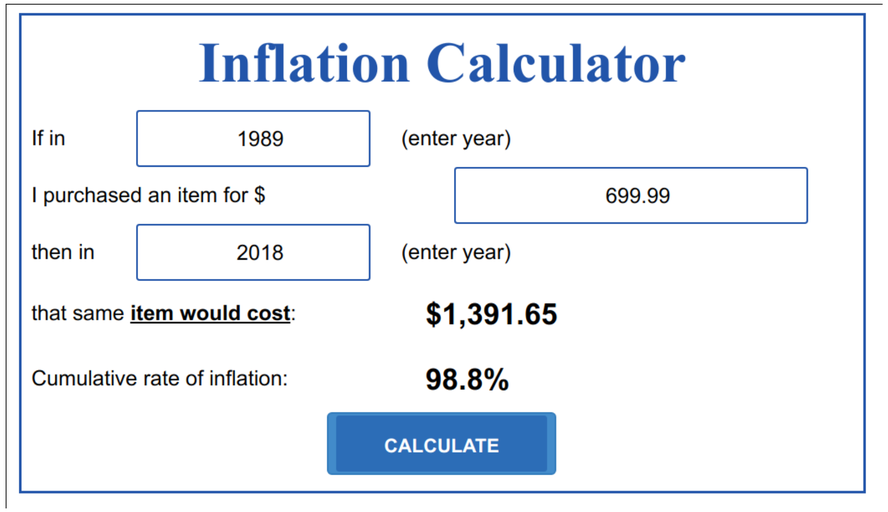
699.00 Low As $40 Per Month*

- **Alternative Expansion Option**
- **Allows Greater Data Storage**

20-Megabyte External Hard Disk Drive. (Cable Kit and installation required for secondary unit.) Requires Hard Disk Controller Board (25-1007).
25-1041 . . . . . **699.00**

Hard Disk Controller Board. For Tandy 1000 SX/SL and original Tandy 1000 only. Allows you to add hard disk drives for up to 40 million characters of storage. Includes cable for use with 10 or 20-megabyte hard disks.
25-1007 . . . . . **299.95**

180 PRICES APPLY AT PARTICIPATING RA

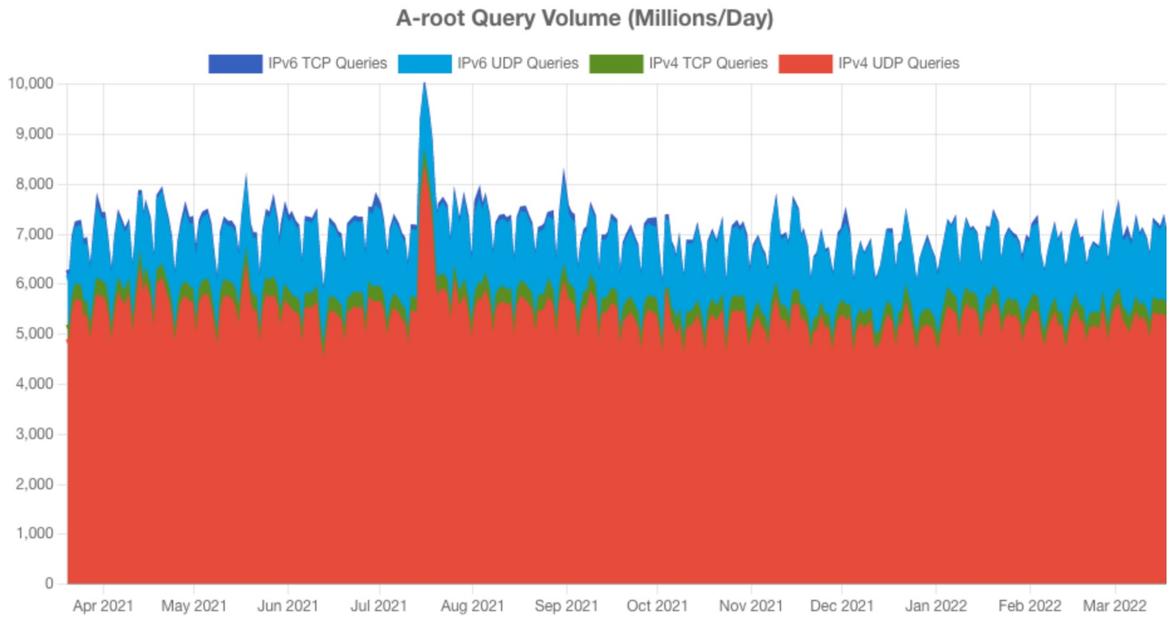


Domain Name System

- Strawman solution: Use a Local file
  - Keep local copy of mapping from all hosts to all IPs (e.g., /etc/hosts)
  - Hosts change IPs regularly: Download file frequently
  - Lot of constant internet bandwidth use
  - IPv4 space is now full
    - 32-bits: 4,294,967,296 addresses
    - At 1 byte per address, file would be 4GB
    - Not a lot of disk space (now, DNS introduced in the late 80s)
    - But a lot of constant internet bandwidth
  - More names than IPs
    - Aliases
  - **Not scalable!**

Domain Name System

- Another Strawman: Well-known centralized server. All requests made to this server:
  - Single point of failure
  - Bottleneck for throughput and access time
  - Bottleneck for administration (adding/changing records?)
  - Ultimately, **not scalable**!



https://a.root-servers.org/metrics

DNS as a distributed system

- We need a **scalable** solution
  - New hosts keep being added
  - Number of users increases
  - Need to maintain speed/responsiveness
- We need our service to be **available** and **fault tolerant**
  - It is a crucial basic service
  - A problematic node shouldn’t “crash the internet”
  - Reads are more important that writes: far more queries to resolve records than to update them
- Global in scope
  - Domain names mean the same thing everywhere

Distributed Systems Goals

- **Scalability**
- Performance
- Latency
- Availability
- Fault Tolerance

"the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth."

"Distributed Systems for Fun and Profit", Takada

Distributed Systems Scale “Horizontally”

- “Vertical” scaling: add more resources to existing server
  - Faster CPUs, more CPU cores, more RAM, more storage
  - Ineffective once: Clock speed plateaus; difficult to write applications that utilize 256 CPU cores (adding 2TB RAM to a server *can* often help)
- “Horizontal” scaling: add more servers
  - Rely on “commodity” servers rather than state-of-the-art hardware
  - Allows for dynamic addition of resources as needed by load

Distributed Systems Goals

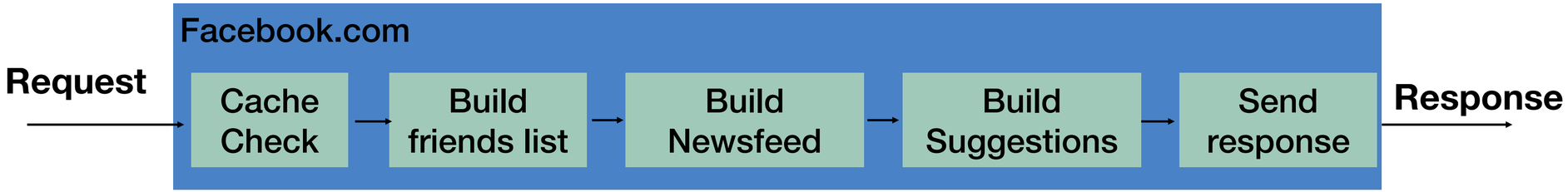
- Scalability
- **Performance**
- Latency
- Availability
- Fault Tolerance

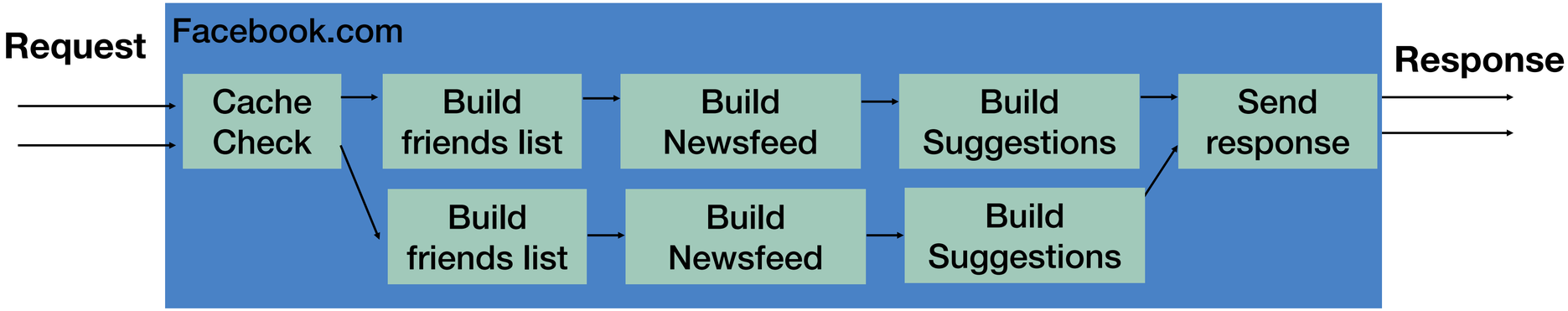
"is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used."

"Distributed Systems for Fun and Profit", Takada

Improve Throughput With Concurrency

Throughput: total requests that can be processed per unit-time





Distributed Systems Goals

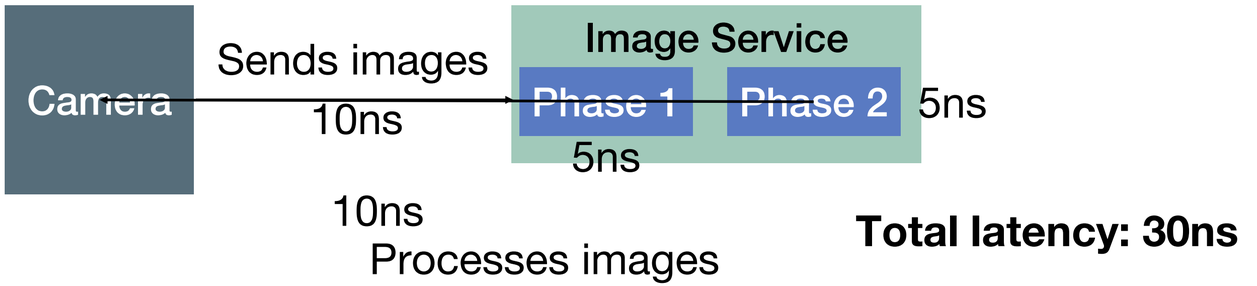
- Scalability
- Performance
- **Latency**
- Availability
- Fault Tolerance

"The state of being latent; delay, a period between the initiation of something and the it becoming visible."

"Distributed Systems for Fun and Profit", Takada

Latency: Delay in Receiving a Response

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
- Adding pipelined components -> latency is cumulative



Improving Latency with Distributed Systems

- Often more challenging than increasing throughput
  - Examples:
    - Physical - Speed of light (network transmissions over long distances)
    - Algorithmic - Looking up an item in a hash table is limited by hash function
    - Economic - Adding more RAM gets expensive
- Distributed systems can reduce latency by moving the data/server closer to the user

Networks Introduce Significant Latency

- We still haven't surpassed the speed of light: ~1 CPU cycle ~= 10cm of transmission distance
- Chicago <-> NYC is a valuable link due to commodities markets in Chicago and equities markets in NYC
- High frequency trading created a market for private, lower-latency alternatives to public internet
- Extremely expensive, not a general solution (and still 8.5ms!)

ORIGINAL CABLE

Technology
Buried fiber-optic cable

Completion
Mid-1980s

Path length
~ 1,000 miles

Round-trip time for data
14.5
milliseconds and up

Approach
Multiple routes followed the easiest rights-of-way—along rail lines. But that means time-sucking jogs and detours.

SPREAD NETWORKS

Technology
Buried fiber-optic cable

Completion
August 2010

Path length
825 miles

Round-trip time for data
13.1
milliseconds

Approach
Spread bought its own rights-of-way, avoiding a Philadelphia-ward dip in favor of a shorter path northwest through central Pennsylvania.

MCKAY BROTHERS

Technology
Microwave beams through air

Completion
July 4, 2012

Path length
744 miles

Round-trip time for data
9
milliseconds

Approach
Microwaves generally move faster than photons in optical fiber, and McKay's network uses just 20 towers on a nearly perfect great circle.

TRADEWORX

Technology
Microwave beams through air

Completion
Winter 2012

Path length
~ 731 miles

Round-trip time for data
8.5
milliseconds (est.)

Approach
Tradeworx is highly secretive, but the company is open about the price of a subscription: $250,000 a year.

https://www.zerohedge.com/news/chicago-new-york-and-back-85-milliseconds

Distributed Systems Goals

- Scalability
- Performance
- Latency
- **Availability**
- Fault Tolerance

"the proportion of time a system is in a functioning condition. If a user cannot access the system, it is said to be unavailable."

Availability = uptime / (uptime + downtime).

Often measured in "nines"

Availability %	Downtime/year
90%	>1 month
99%	< 4 days
99.9%	< 9 hours
99.99%	<1 hour
99.999%	5 minutes
99.9999%	31 seconds

"Dis ada

Distributed Systems Challenges: Availability

More machines, more problems

- Say there's a 1% chance of having some hardware failure occur to a machine in a given month (power supply burns out, hard disk crashes, etc)
- Now I have 10 machines
  - Probability(at least one fails during the month) = 1 - Probability(no machine fails) = $1-(1-.01)^{10} = 10\%$
- 100 machines -> 63% chance that at least one fails
- 200 machines -> 87% chance that at least one fails (!)
- Implication: System as a whole must tolerate component failures

Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- **Fault Tolerance**

"ability of a system to behave in a well-defined manner once faults occur"

What kind of faults?

Disks fail

Power supplies fail

Networking fails

Security breached

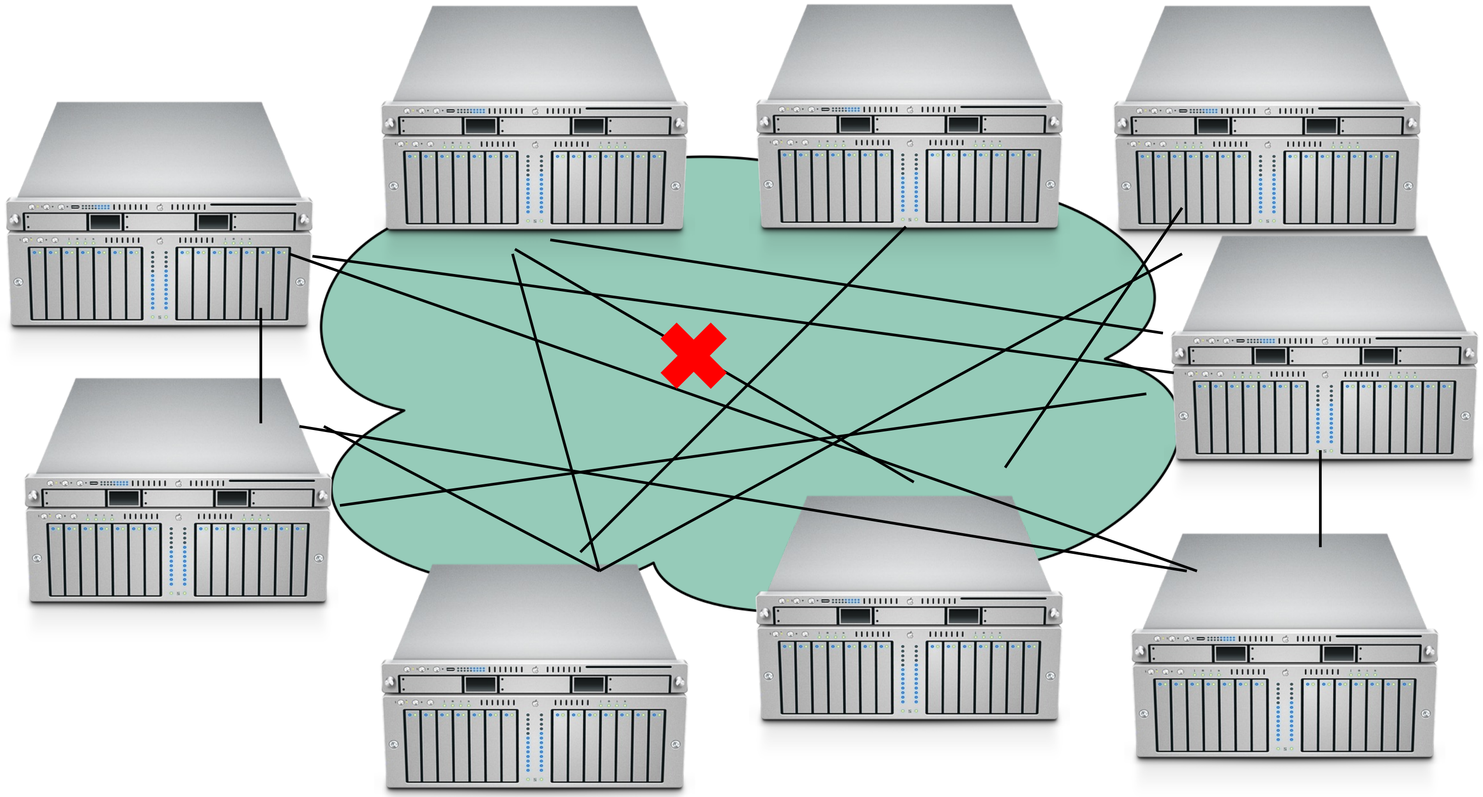
Datacenter goes offline

Power goes out

"Distributed Systems for Fun and Profit", Takada

Distributed Systems Challenges: Performance, Availability

Number of nodes + distance between them



Challenge: Distributed Systems Rely on Networks

- Can not expect network to be a perfect analog for communication within a single computer because:
  - Speed of light (1 foot/nanosecond)
  - Communication links exist in uncontrolled/hostile environments
  - Communication links may be bandwidth limited (tough to reach even 100MB/sec)
- In contrast to a single computer, where:
  - Distances are measured in mm, not feet
  - Physical concerns can be addressed all at once
  - Bandwidth is plentiful (easily GB/sec)

And still more challenges

We still rely on other administrators, who are not infallible

Amazon Web Services outage takes a portion of the internet down with it

Zack Whittaker

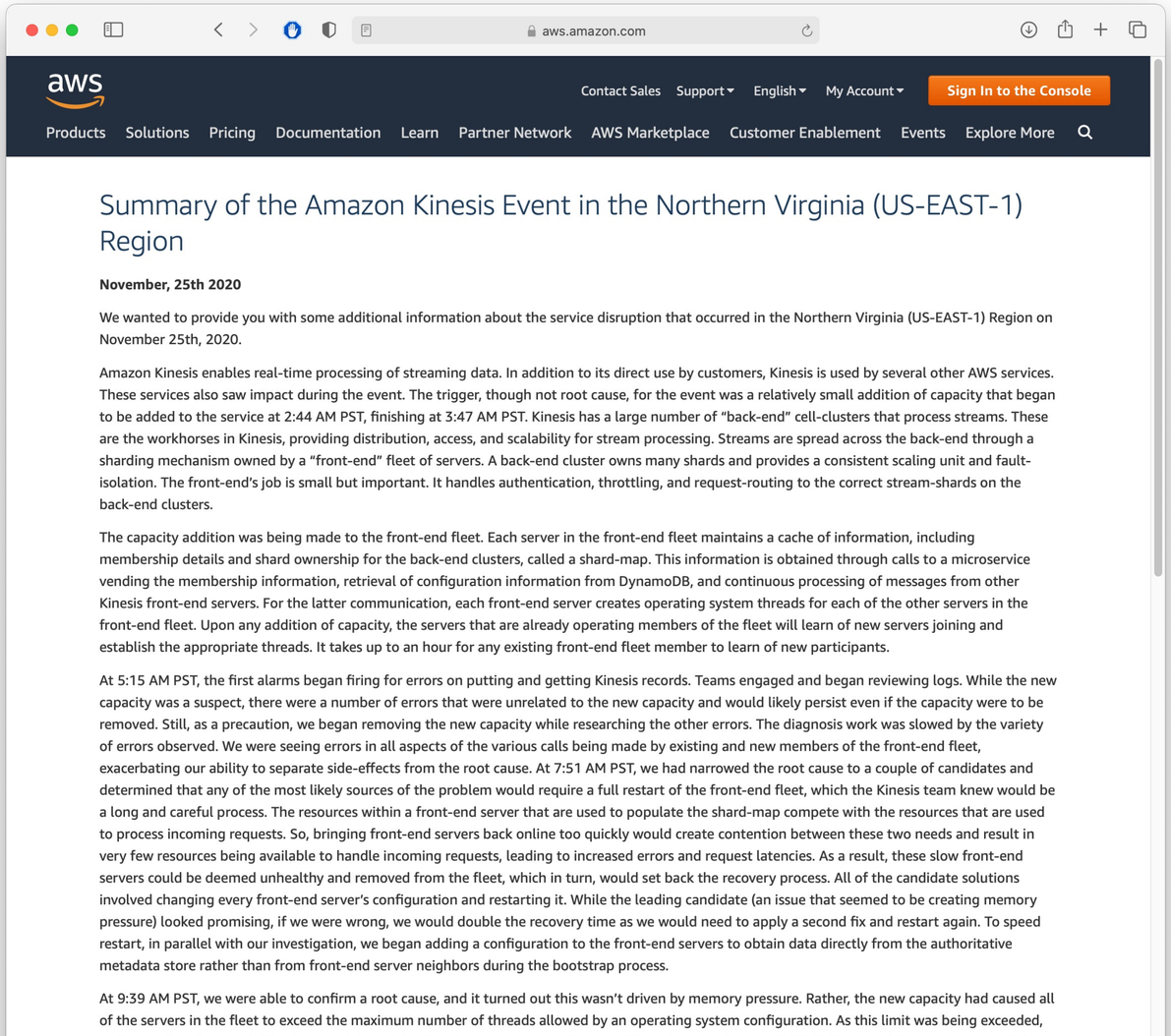
@zackwhittaker / 12:32 PM EST • November 25, 2020

Comment

Image Credits: David Becker / Getty Images

Amazon Web Services is currently having an outage, taking a chunk of the internet down with it.

Several AWS services were experiencing problems as of early Wednesday, according to its status page. That means any app, site or service that relies on AWS might also be down, too. (As I found out the hard way this morning when



System Design Follows Requirements

Domain Name System

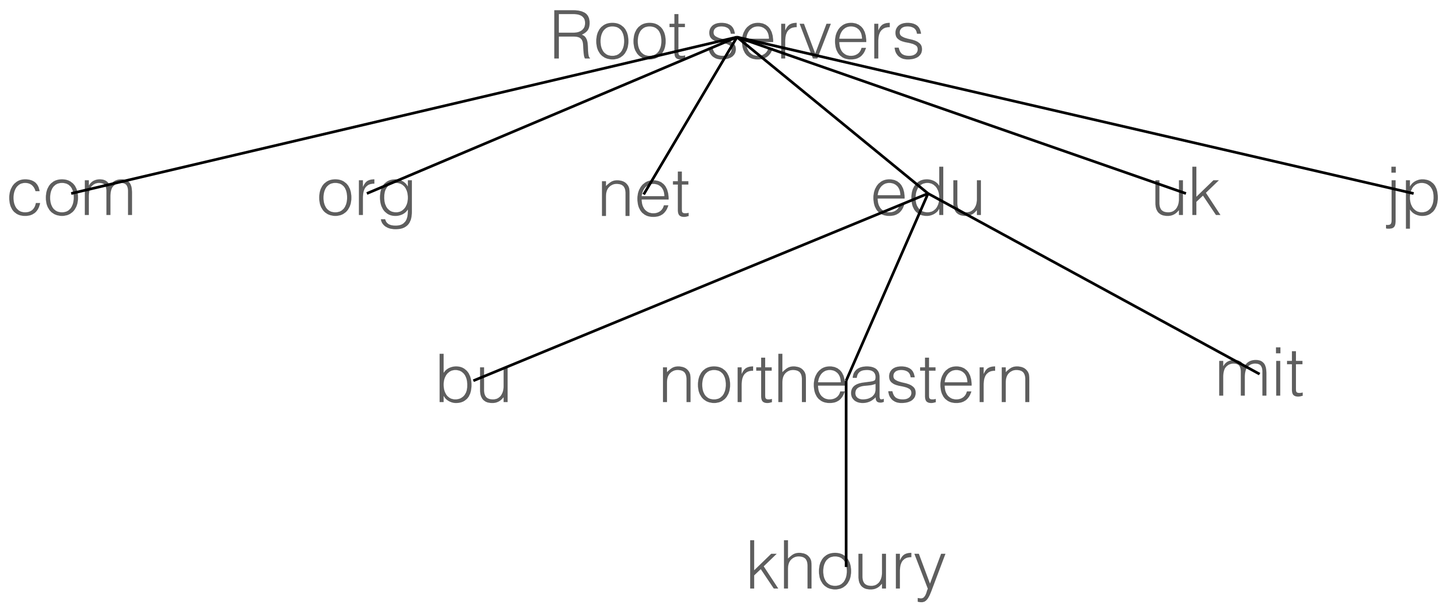
- DNS requirements for querying records:
  - Scalability - grow in number of names
  - Performance - grow in number of requests
  - Latency - provide answers with little delay
  - Fault tolerance & availability - always provide a response
- DNS requirements for updating/creating records:
  - Latency - OK if it takes minutes/hours for an update to take full effect
  - Performance - Expect far fewer writes than reads

Preliminary Design: How to organize data in the system?

- This depends to a large degree on whether there is shared state
- Usually, there is some shared state
- How important is it to synchronize?
- What about our DNS example?
  - Domains can be split (e.g., .com, .edu, .info, .eu, .jp, …)
  - Huge volume of requests – multiple nodes need to provide the same mappings, consistently

How to organize DNS

Idea: break apart responsibility for each part of a domain name (**zone**) to a different group of servers



Each zone is a continuous section of name space
Each zone has an associate set of name servers

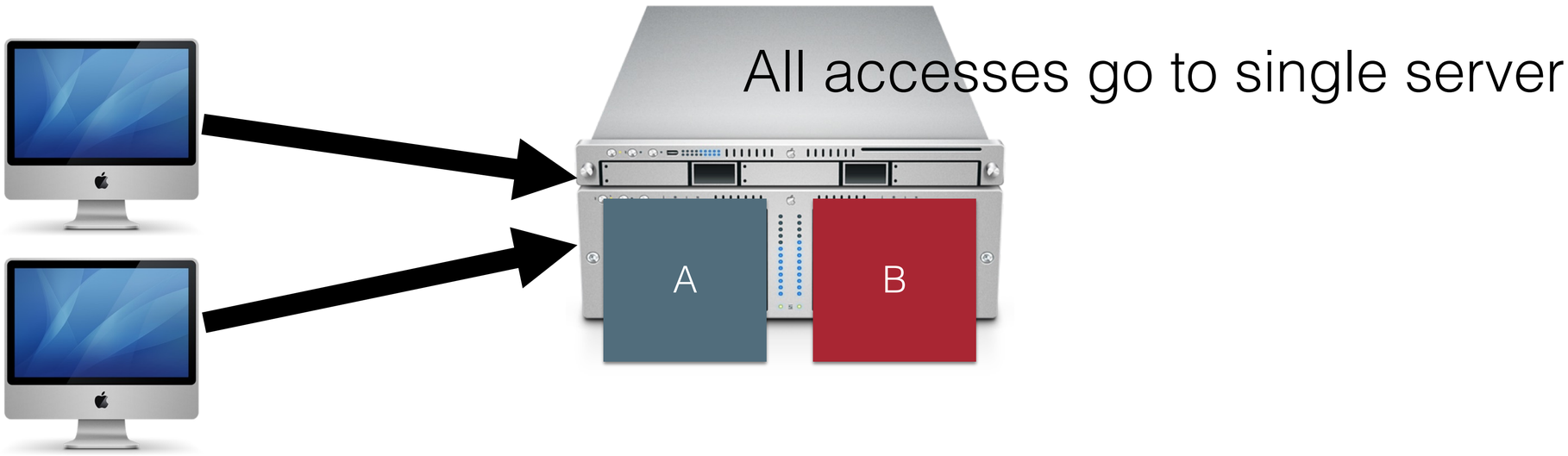
How to organize DNS

Idea: break apart responsibility for each part of a domain name (**zone**) to a different group of servers

In other words, we **partition** the domain names according to the top-level domain.

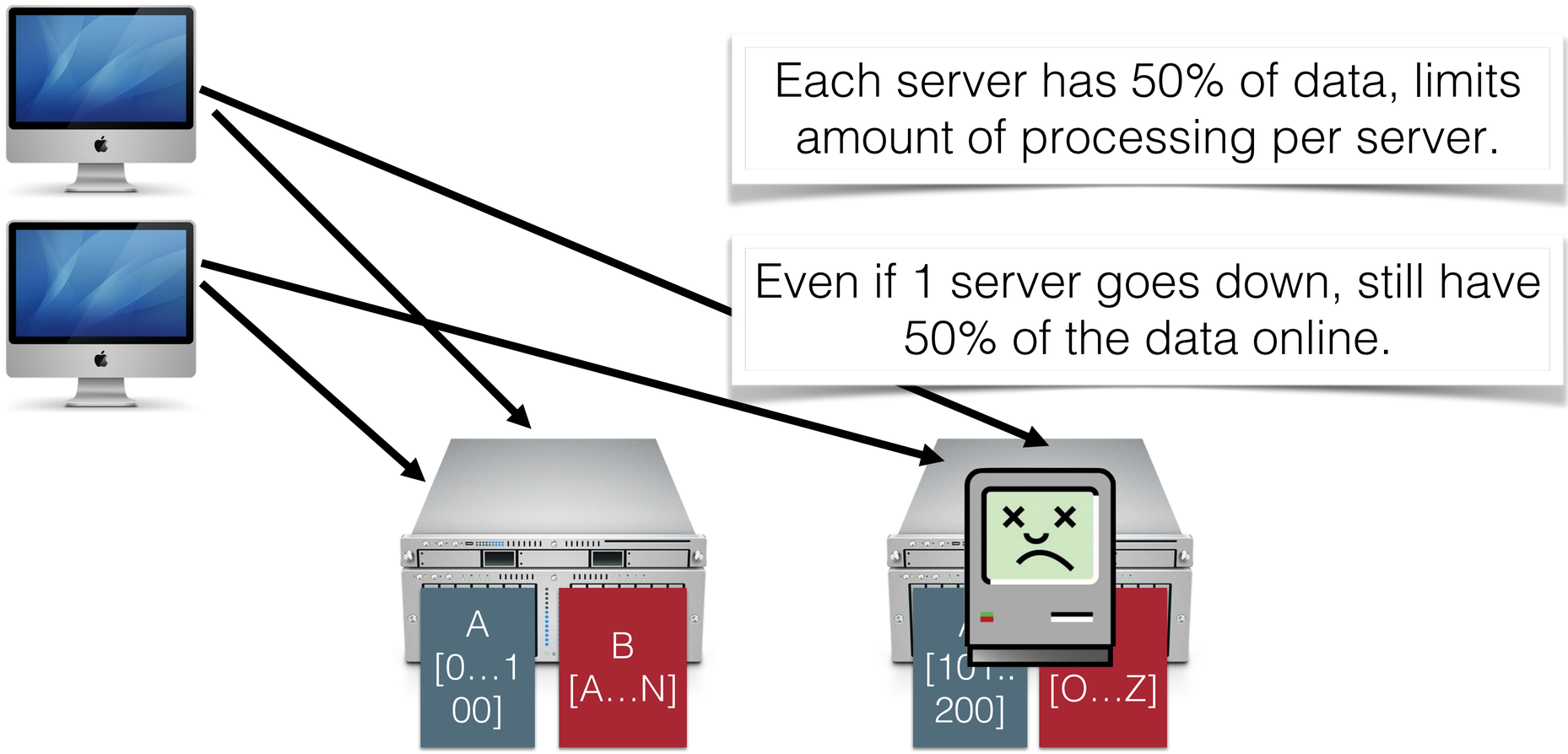
Recurring Solution #1: Partitioning

- Partitioning is a common strategy to distributing a system and its data
- Starting from a non-distributed system:

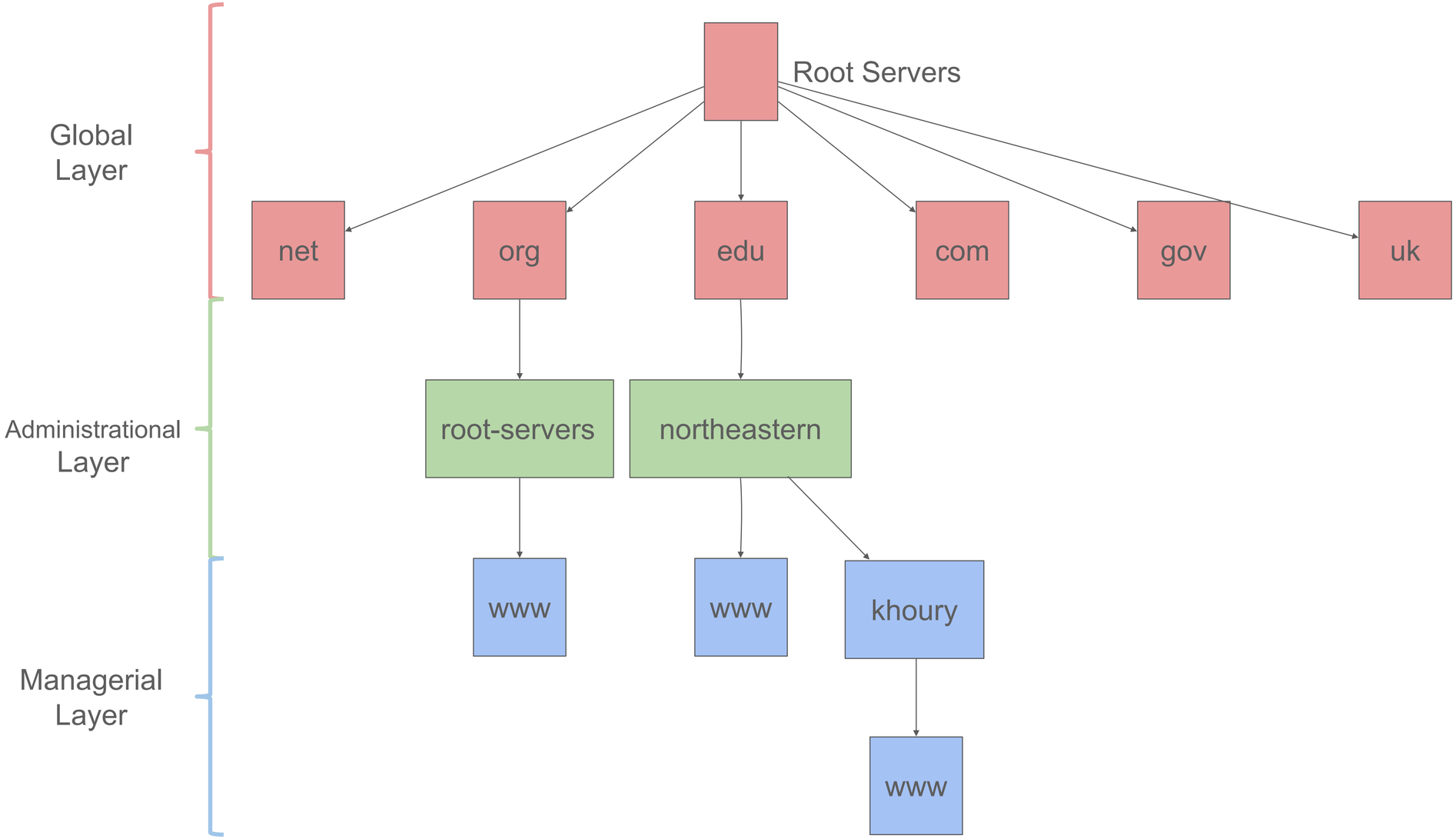


Recurring Solution #1: Partitioning

- Divide data up in some (hopefully logical) way
- Makes it easier to process data concurrently (cheaper reads)

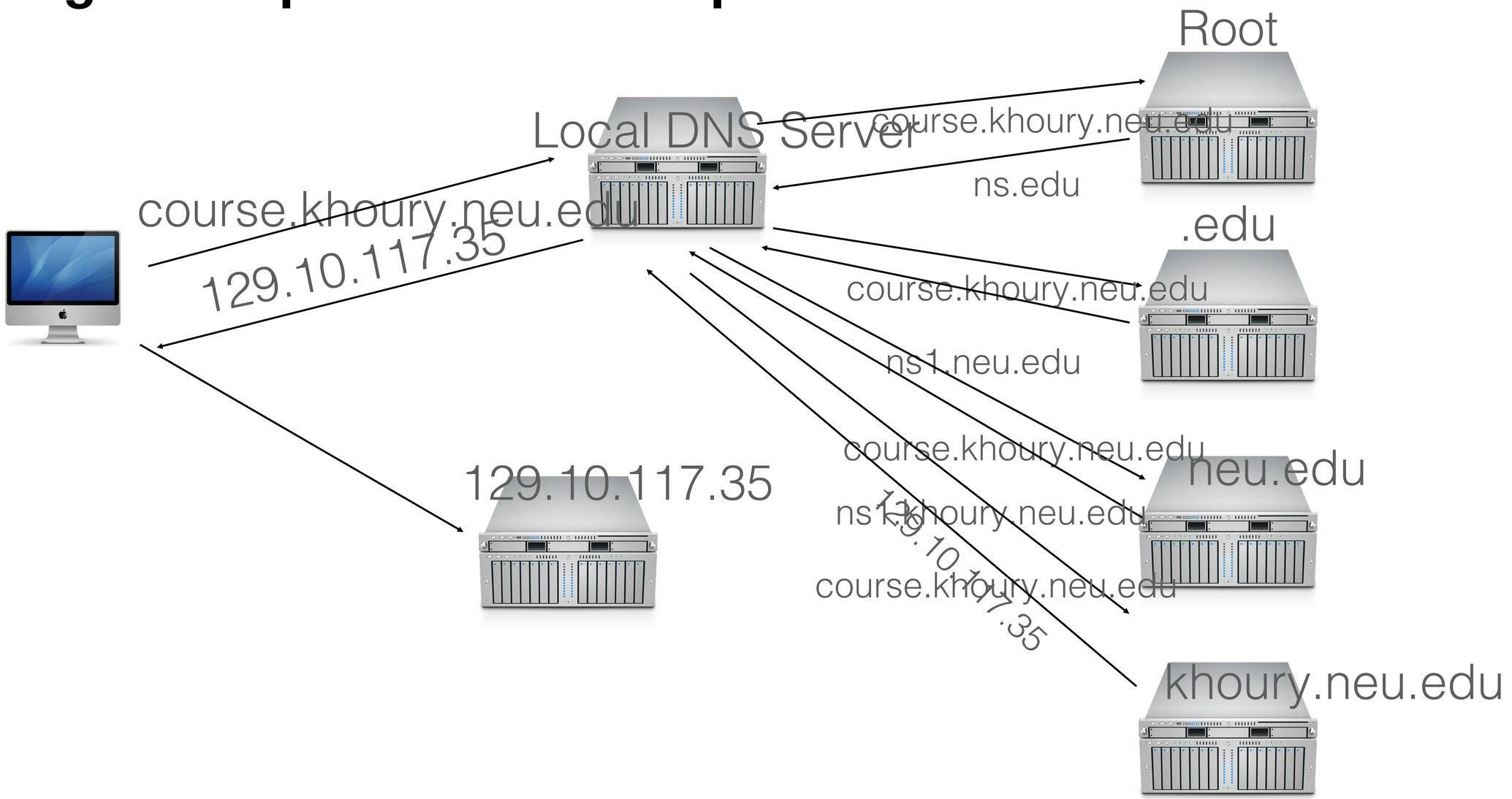


Partitioning DNS



DNS: Example

What might a request look like in practice?

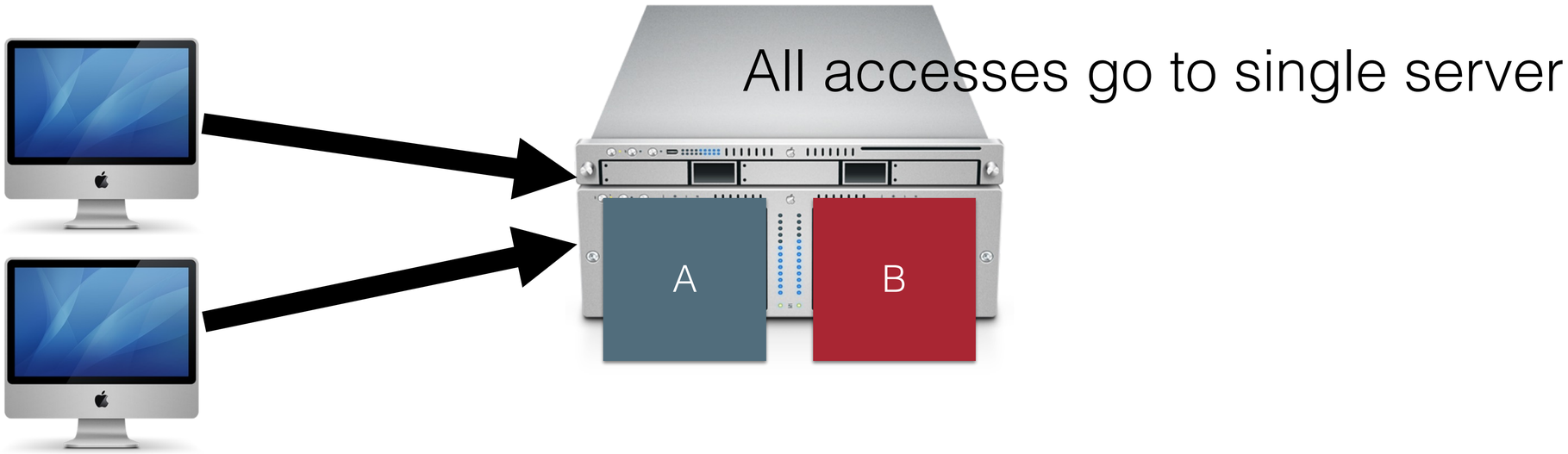


How to deal with volume?

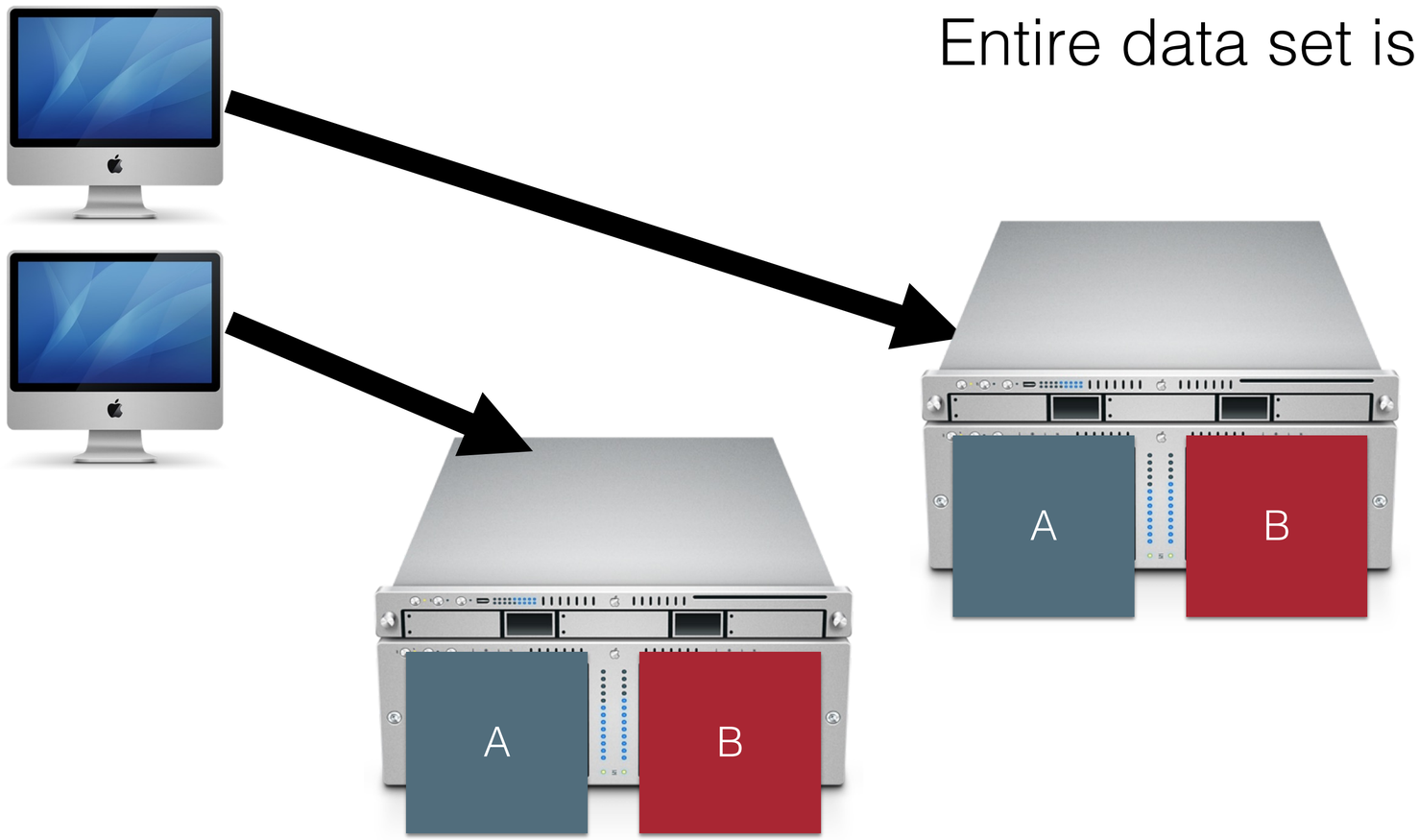
- We successfully distributed requests following the hierarchical nature of domain names
- However, e.g., .com is a very popular TLD – there might be (hundreds of) thousands of requests happening at any given time
- We may need several nodes just servicing .com
- This leads to **replication**

Recurring Solution #2: Replication

- Goal: Any node should be able to process any request
- Again, starting from a non-distributed system:



Recurring Solution #2: Replication



Entire data set is copied

Recurring Solution #2: Replication

- Improves performance:
  - Client load can be evenly shared between servers
  - Reduces latency: can place copies of data nearer to clients
- Improves availability:
  - One replica fails, still can serve all requests from other replicas

Replication in DNS – Root Servers

- 13 root servers
  - `[a-m].root-servers.org`
  - E.g., `d.root-servers.org`
- Handled by 12 distinct entities
  - (“a” and “j”) are both Verisign
    - Don’t ask why.

Verisign, Inc.	a
Information Sciences Institute	b
Cogent Communications	c
University of Maryland	d
NASA Ames Research Center	e
Internet Systems Consortium, Inc.	f
U.S. DOD Network Information Center	g
U.S. Army Research Lab	h
Netnod	i
Verisign, Inc.	j
RIPE NCC	k
ICANN	l
WIDE Project	m

There is replication even within the root servers

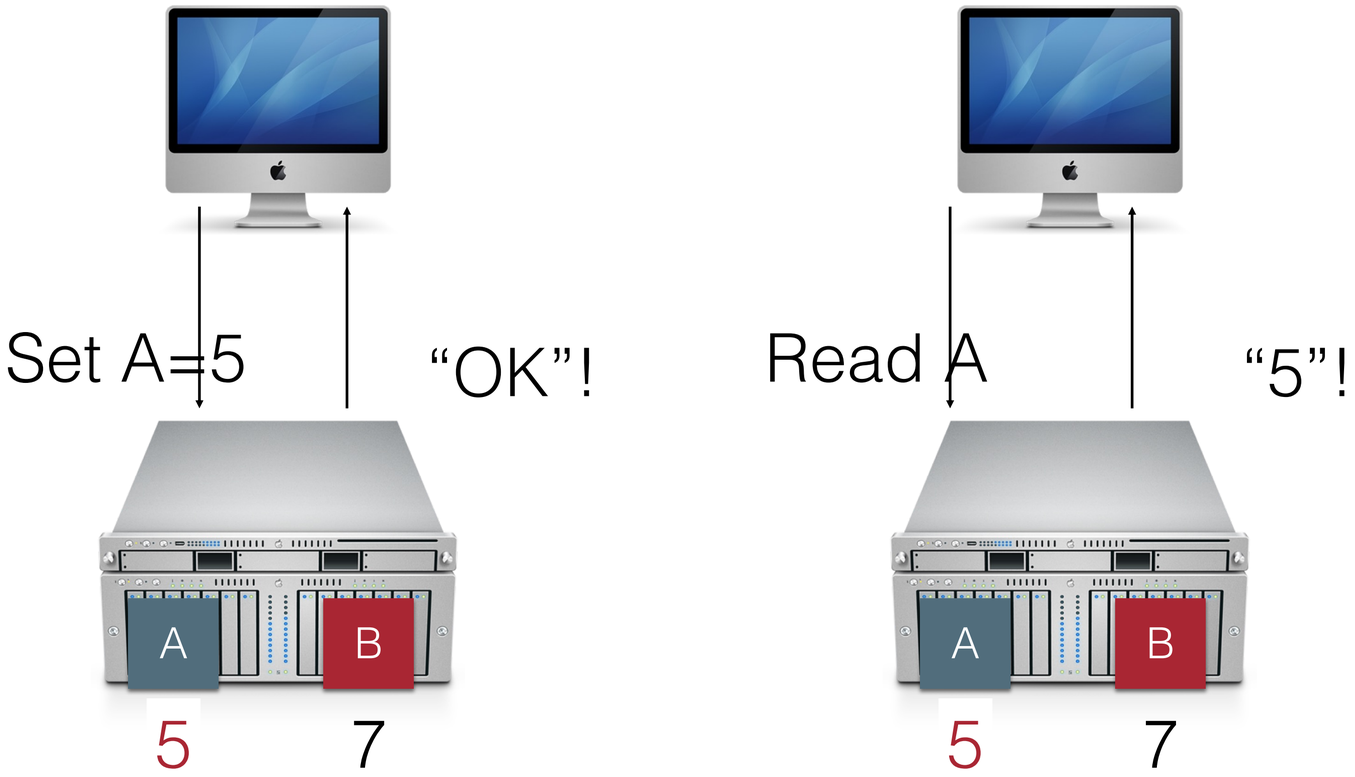
- 13 root servers
  - `[a-m].root-servers.org`
  - E.g., `d.root-servers.org`
- But each root server has multiple copies of the database, which need to be kept in sync.
- Somewhere around 1500 replicas in total.

Partitioning + Replication

- So, DNS combines both partitioning and replication
- As do most distributed systems

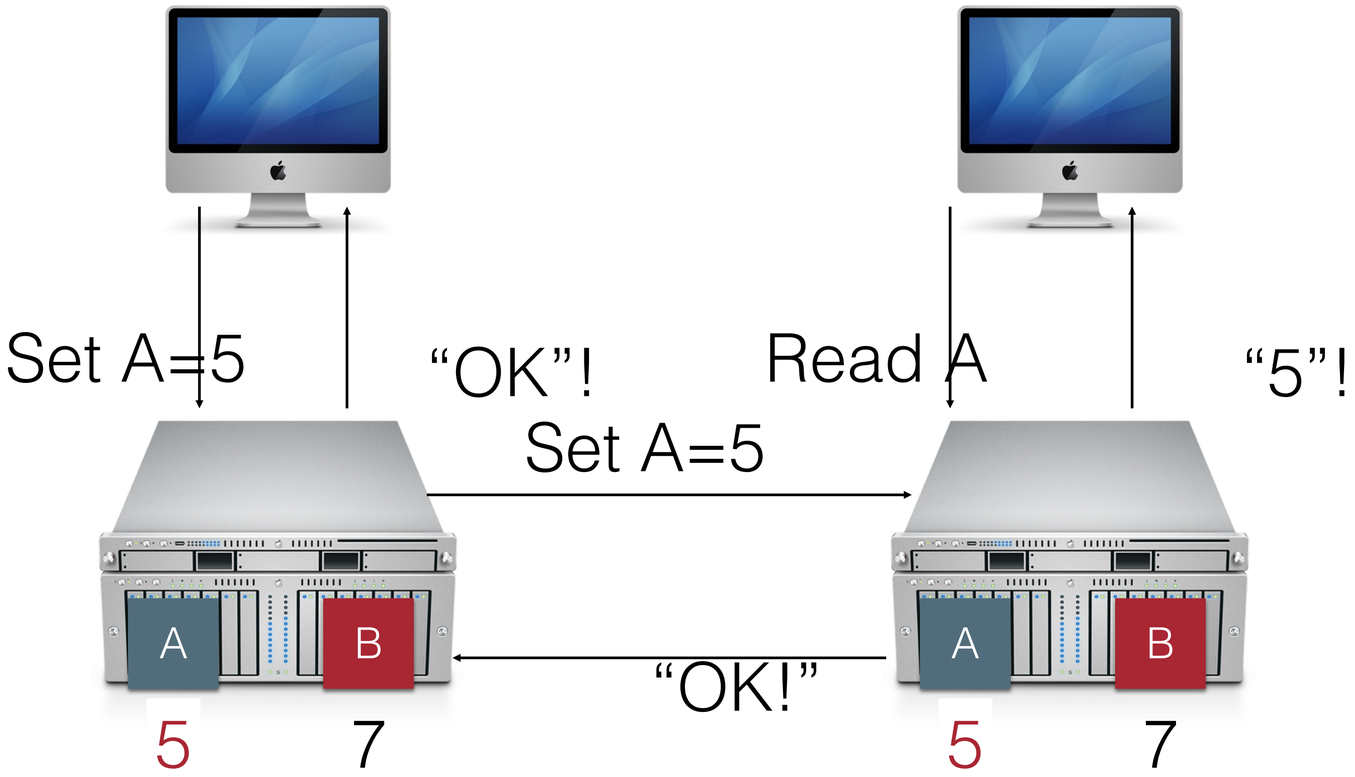
Replication Problem: Consistency

We probably want our system to work like this



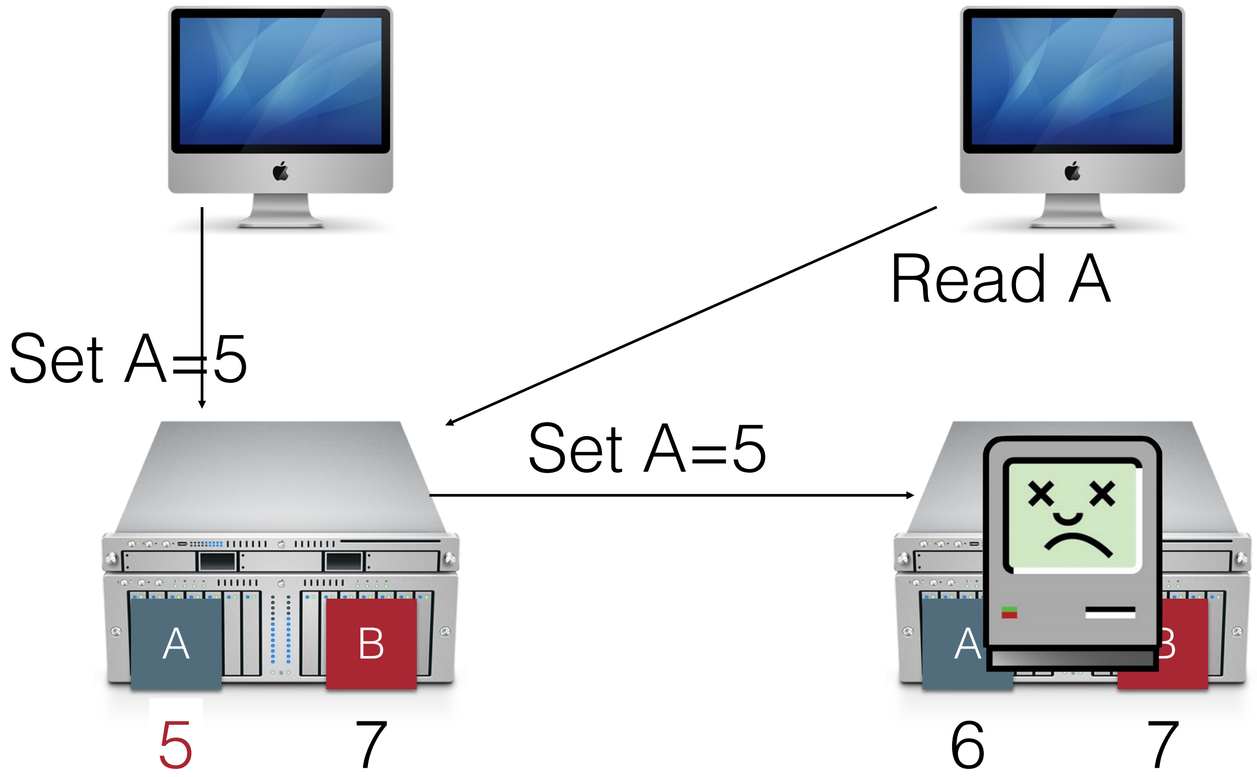
Sequential Consistency

AKA: Behaves like a single machine would



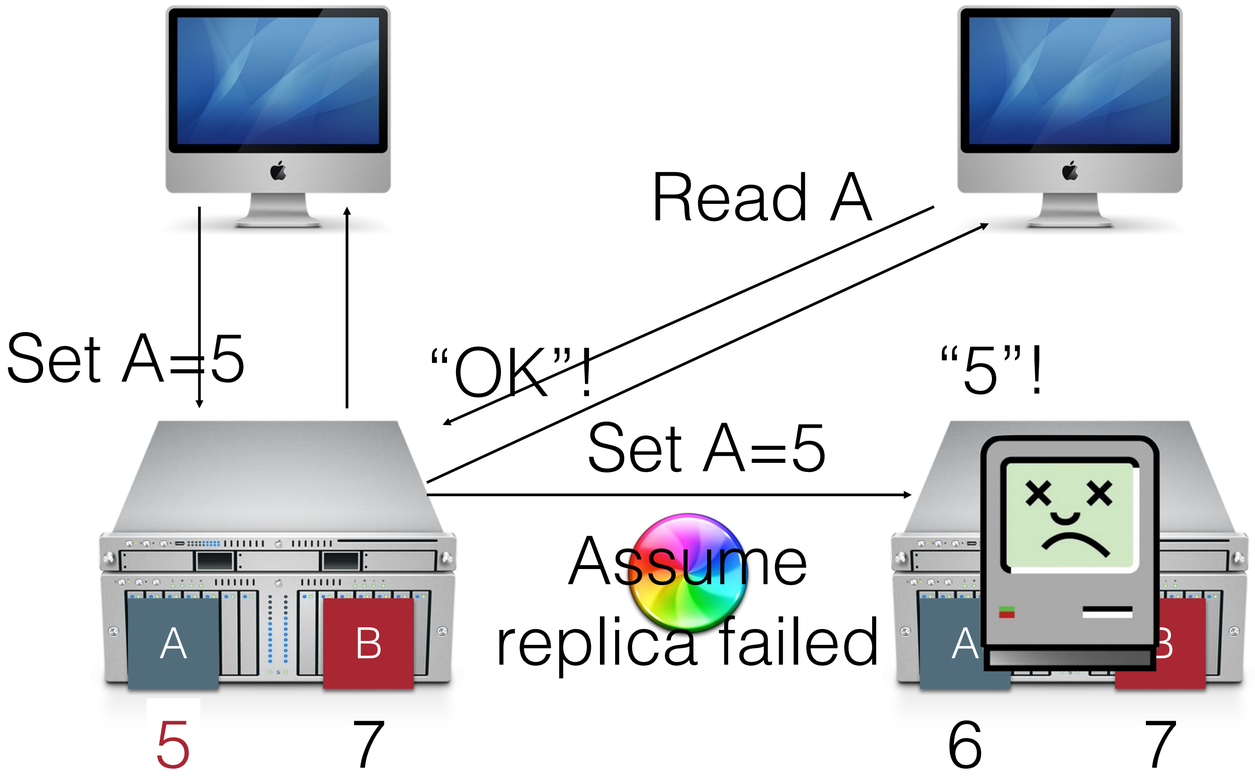
Availability

If at least one node is online, can we still answer a request?

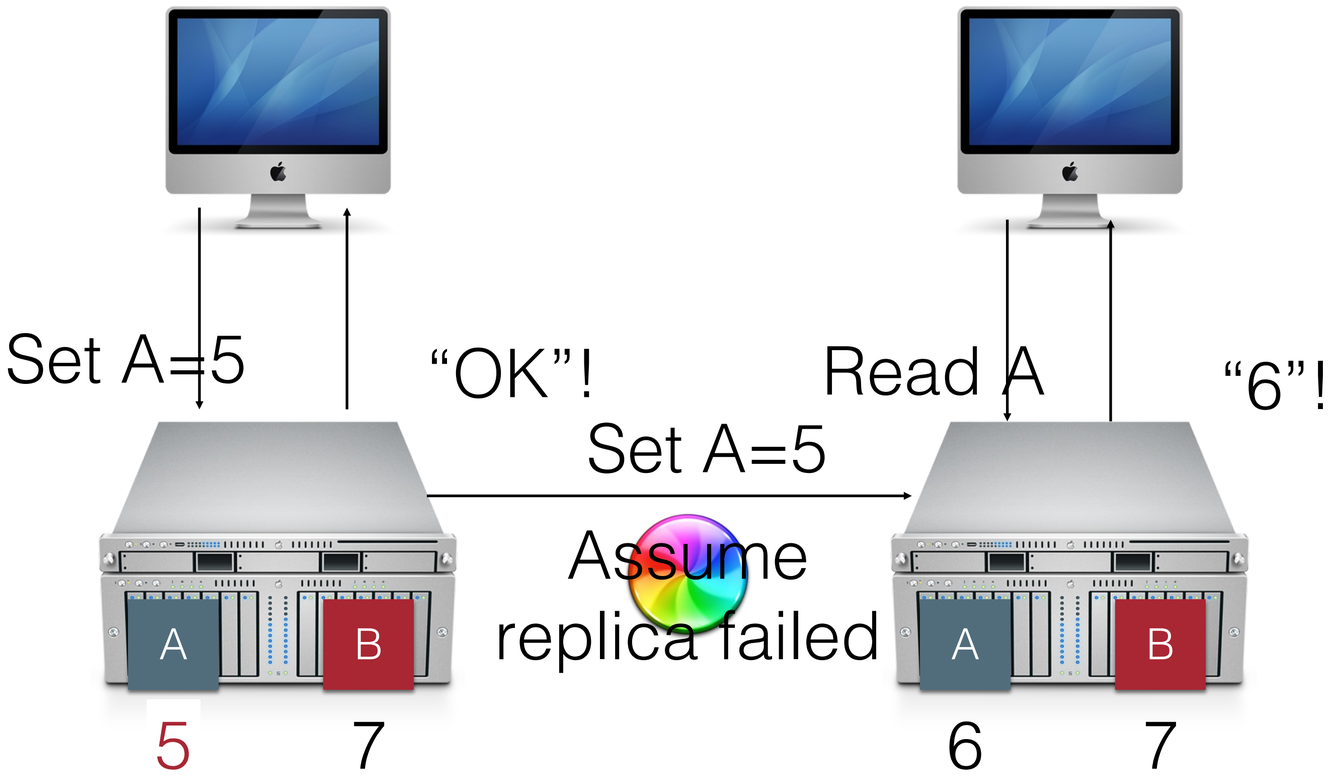


Consistent + Available

On timeout, assume node is crashed



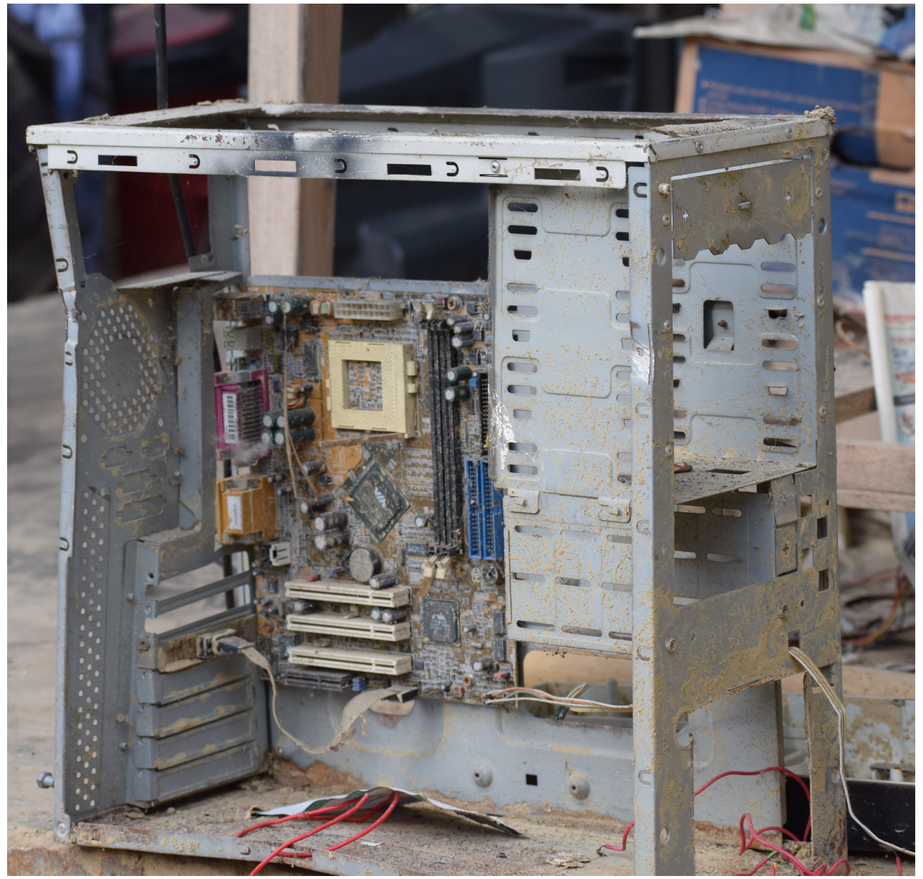
What if *the network* fails?



Shared Fate

Are you still there?

- Two methods/threads/processes running on the same computer generally have **shared fate** [Crashed/not]
- When two machines in a distributed system can’t talk to each other, how do we know if the other is crashed?
- We call this a **split brain** problem



CAP Theorem: Consistency or Availability

- Pick two of three:
  - Consistency: All nodes see the same data at the same time (strong consistency)
  - Availability: Individual node failures do not prevent survivors from continuing to operate
  - Partition tolerance: The system continues to operate despite message loss (from network and/or node failure)
    - Can’t drop this for a DS - networks can always fail

DNS uses a relaxed consistency model

- Recall - DNS requirements for updating/creating records:
  - Latency - OK if it takes minutes/hours for an update to take full effect
  - Performance - Expect far fewer writes than reads
- DNS does not attempt to provide strong consistency
- Updates are made in a best-effort approach
- OK for clients to see the “old” value for a short time after it was updated

Distributed Software Engineering Abstractions

Key Question: Consistency vs Availability

- Distributed system will never match exact semantics of non-distributed system
- For replication do we value more: guaranteed consistency (looks like a single machine) or guaranteed availability (sometimes read stale data)?
  - For a lock server?
  - For the order of tweets on twitter?
- For partitioning: Where can we draw the line? How do we determine how to organize the partitions?
- Next module: common architectures for distributed systems

Review: Learning Objectives for this Lesson

By the end of this lesson, you should be able to…

- Describe 5 key goals of distributed systems
- Understand the fundamental constraints of distributed systems
- Understand the roles of replication and partitioning in the context of the DNS infrastructure