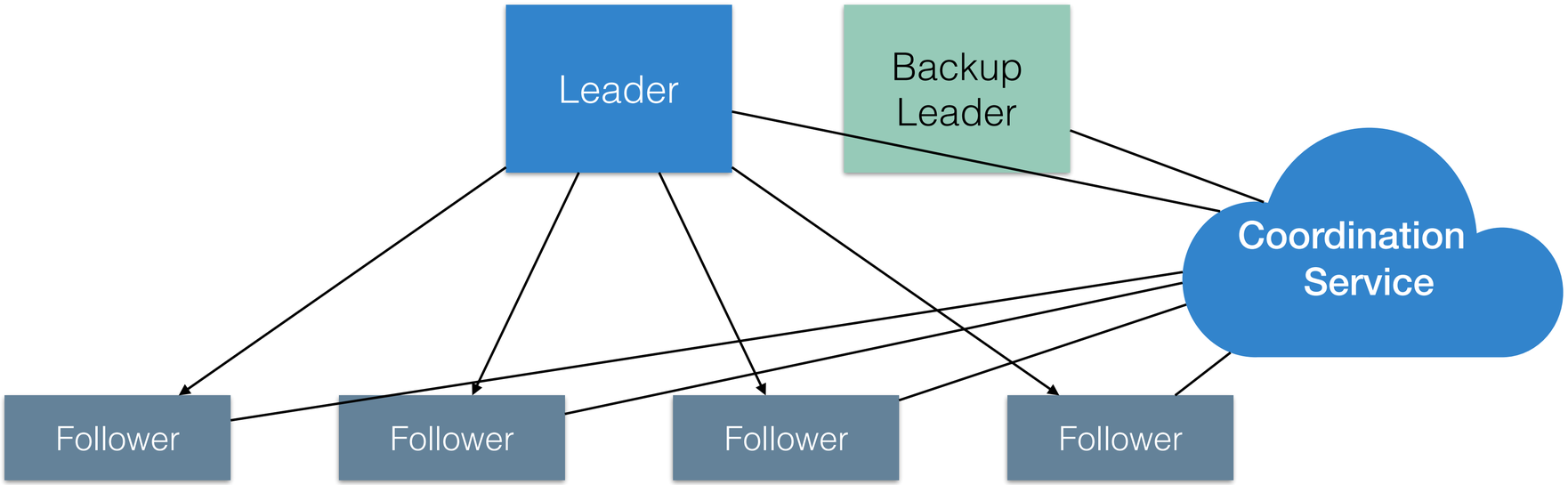
Sharding & CDNs

CS 475, Spring 2019
Concurrent & Distributed Systems

GEORGE MASON UNIVERSITY

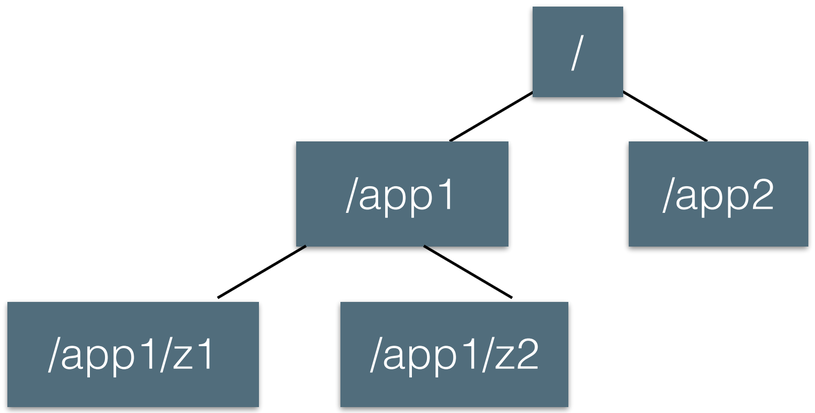
Review: Fault Tolerant Leader/Follower System



Coordination service handles all of those tricky parts.
But can't the coordination service fail?

Review: ZooKeeper - Data Model

- Provides a hierarchical namespace
- Each node is called a znode
- ZooKeeper provides an API to manipulate these nodes



Review: ZooKeeper - ZNodes

- In-memory data
- NOT for storing general data - just metadata (they are replicated and generally stored in memory)
- Map to some client abstraction, for instance - locks
- Znodes maintain counters and timestamps as metadata

Review: ZooKeeper - Znode Types

- Regular znodes
  - Can have children znodes
  - Created and deleted by clients explicitly through API
- Ephemeral znodes
  - Cannot have children
  - Created by clients explicitly
  - Deleted by clients OR removed automatically when client session that created them disconnects

Review: ZooKeeper - API

- Clients track changes to znodes by registering a **watch**
- Create(path, data, flags)
  Delete(path, version)
  Exists(path, watch)
  getData(path, watch)
  setData(path, data, version)
  getChildren(path, watch)
  Sync(path)

Review: ZooKeeper - Recipes

- Why figure out how to re-implement this low level stuff (like locks)?
- Recipes: https://zookeeper.apache.org/doc/r3.3.6/recipes.html
  - And in Java: http://curator.apache.org
- Examples:
  - Locks
  - Group Membership

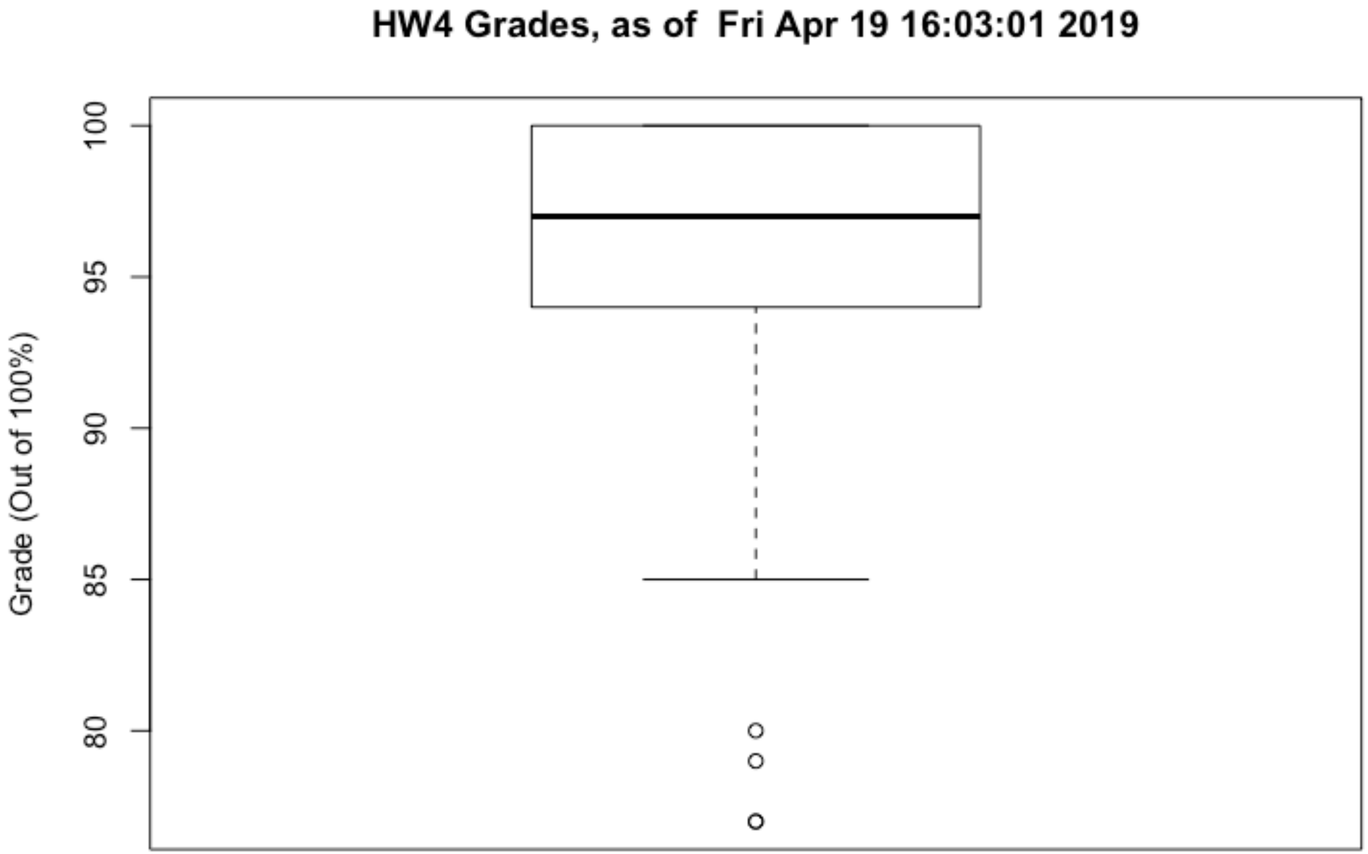
Review: Failure Handling in ZK

- Just using ZooKeeper does not solve failures
- Apps using ZooKeeper need to be aware of the potential failures that can occur, and act appropriately
- ZK client will guarantee consistency **if it is connected to the server cluster**

Review: ZK: Handling Reconnection

- What should we do when we reconnect?
- Re-issue outstanding requests?
  - Can't assume that outstanding requests didn't succeed
  - Example: create /leader (succeed but disconnect), re-issue create /leader and fail to create it because you already did it!
- Need to check what has changed with the world since we were last connected

HW4 Discussion



Today

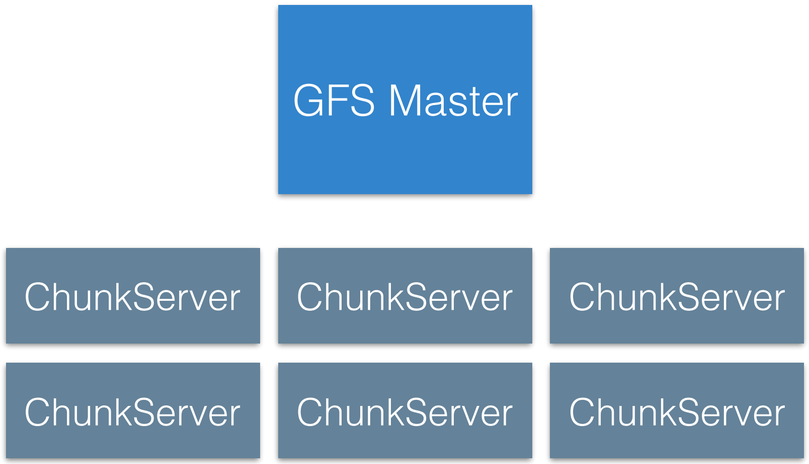
- This week: How do we get rid of the “master” server that keeps track of metadata?
- Today:
  - Sharding & CDNs
- Wednesday:
  - Pure peer-to-peer systems
- Reminder - Project is out!
  - Fault-tolerant, sequentially consistent replicated key value store
  - Can do in a group (1 to 3 students per group)

Locating Data

- How do we find data?
- Every answer so far has required some sort of central server
  - DNS lets us resolve names, going through the root servers
  - GFS lets us find chunks that match to files, but need to go through master server
- Why not use the central server to find data?

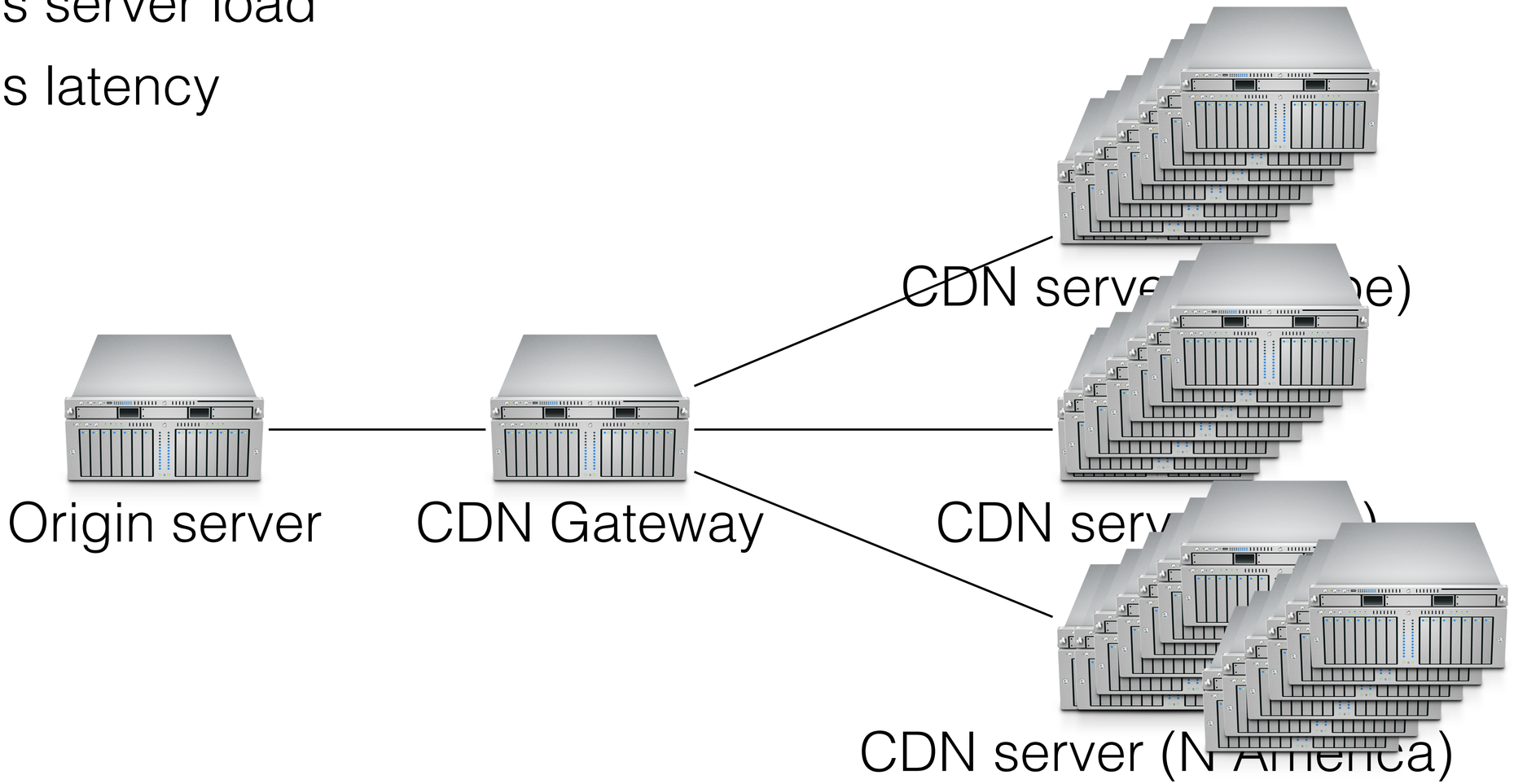
Why not use a central server to find data?

- Central server is:
  - Point of failure
  - Performance bottleneck
  - Requires bootstrapping



Motivating Problem: CDN

- Goal: replicate web content to many servers
- Reduces server load
- Reduces latency

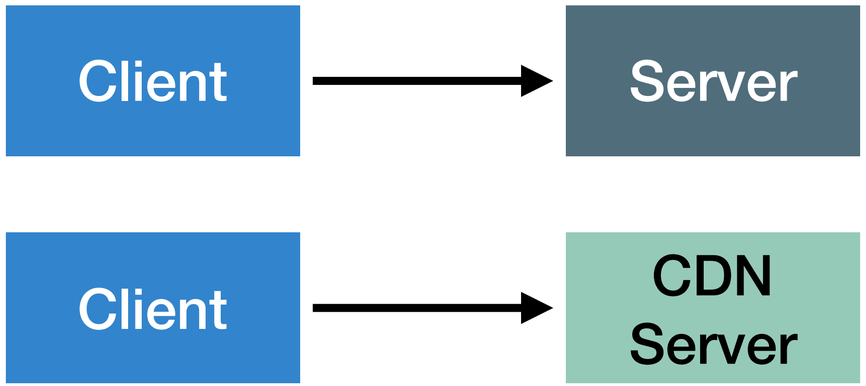


CDNs

- Motivating scenario:
  - Prof Bell has a popular web page, doesn't want to be limited by his server's capacity or location
- What is the high level problem?
  - We will scatter caches across the internet, direct browser to nearest cached copy
  - If not cached nearby, fetch and store in cache
- Why does this help?
  - Reduces server load
  - Reduces latency

CDNs

- Constraints:
  - No support from browser
  - No support from the server we are caching
  - Different pages will have different popularities
- What can we do?
  - We can change DNS lookups and see the HTTP requests from the clients



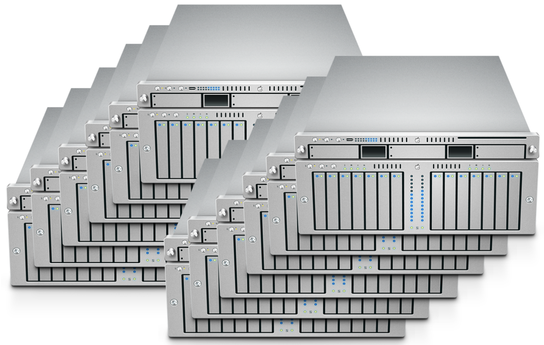
CDN Challenges

- How do we replicate the content?
  - Assume: only static content
- Where do we replicate the content?
  - Assume: infinite money
- How to choose amongst known replicas?
  - Lowest load? Best performance?
- How to find the replicated content?
  - Tricky

CDN Challenges Finding Content

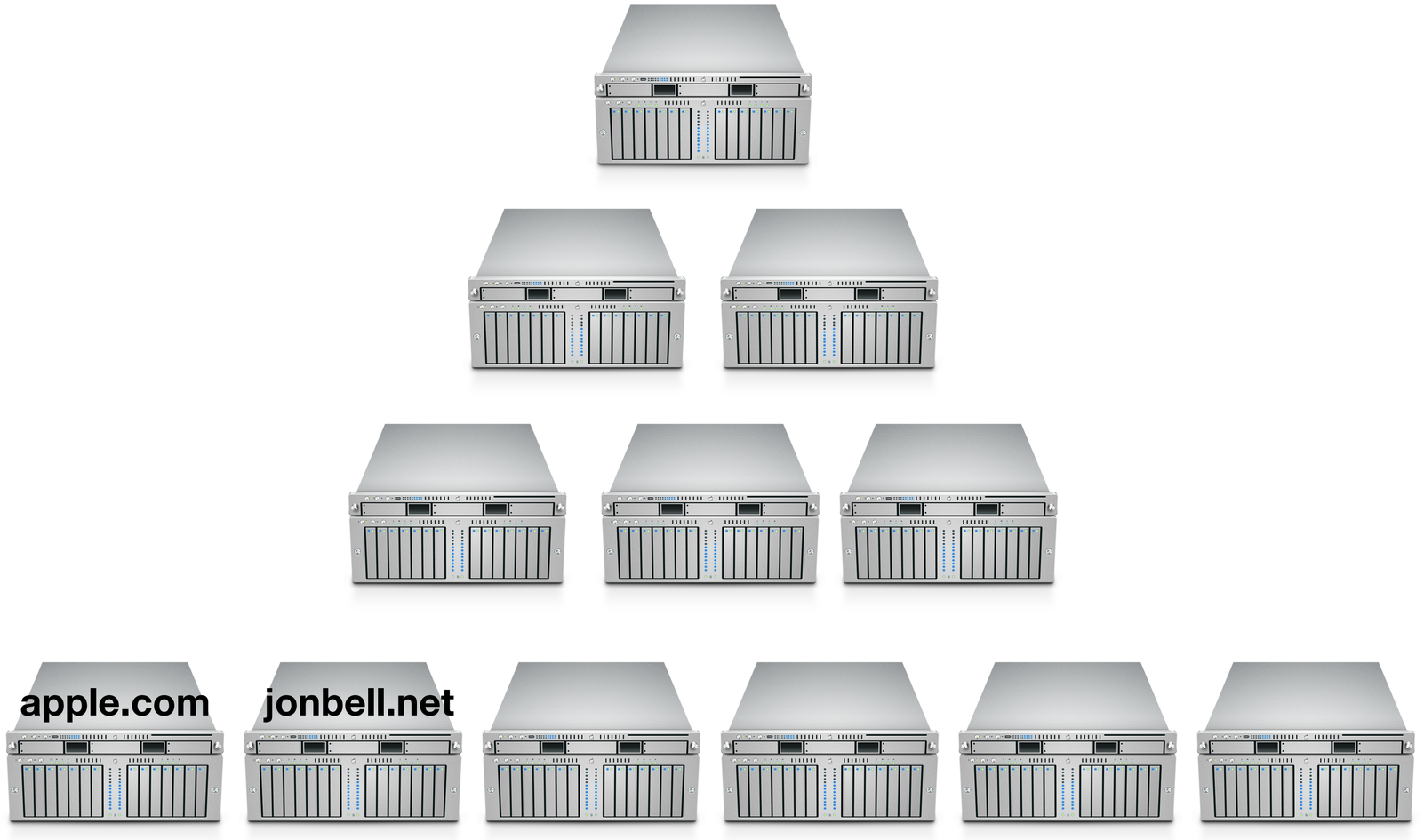
- Desire super, super low latency
- Serving a single request is probably very very cheap (e.g. read a 1KB file from memory/SSD and return it)
- But we want to serve billions of these requests at once
- VERY important that we can route requests fast

CDN: How to find content?



Problem: how the heck do we figure out what data should go where?

CDN: How to find content? (Strawman)



Problem: How do we organize the tiers/shortcut from URLs to servers?
1 server per domain doesn't help balance load

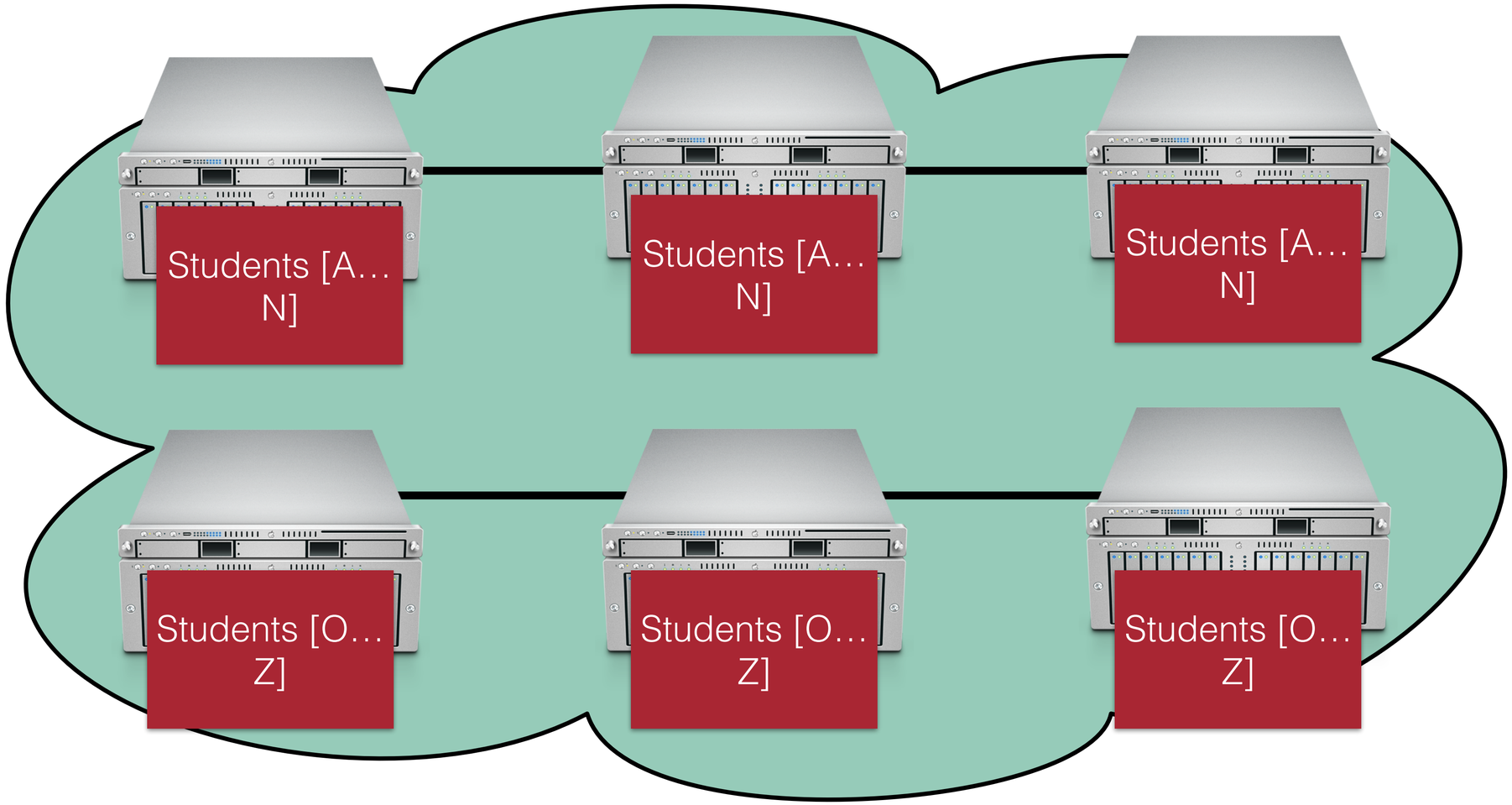
CDN: How to find content? (Strawman)

Master server directs all requests to appropriate cache server

Big cluster of cache servers

Problem: Master becomes a huge bottleneck
Millions of requests, each request needs to be processed incredibly fast

Strawman: Sharding (Partitioning by Key)



Strawman: Sharding (Partitioning by Key)

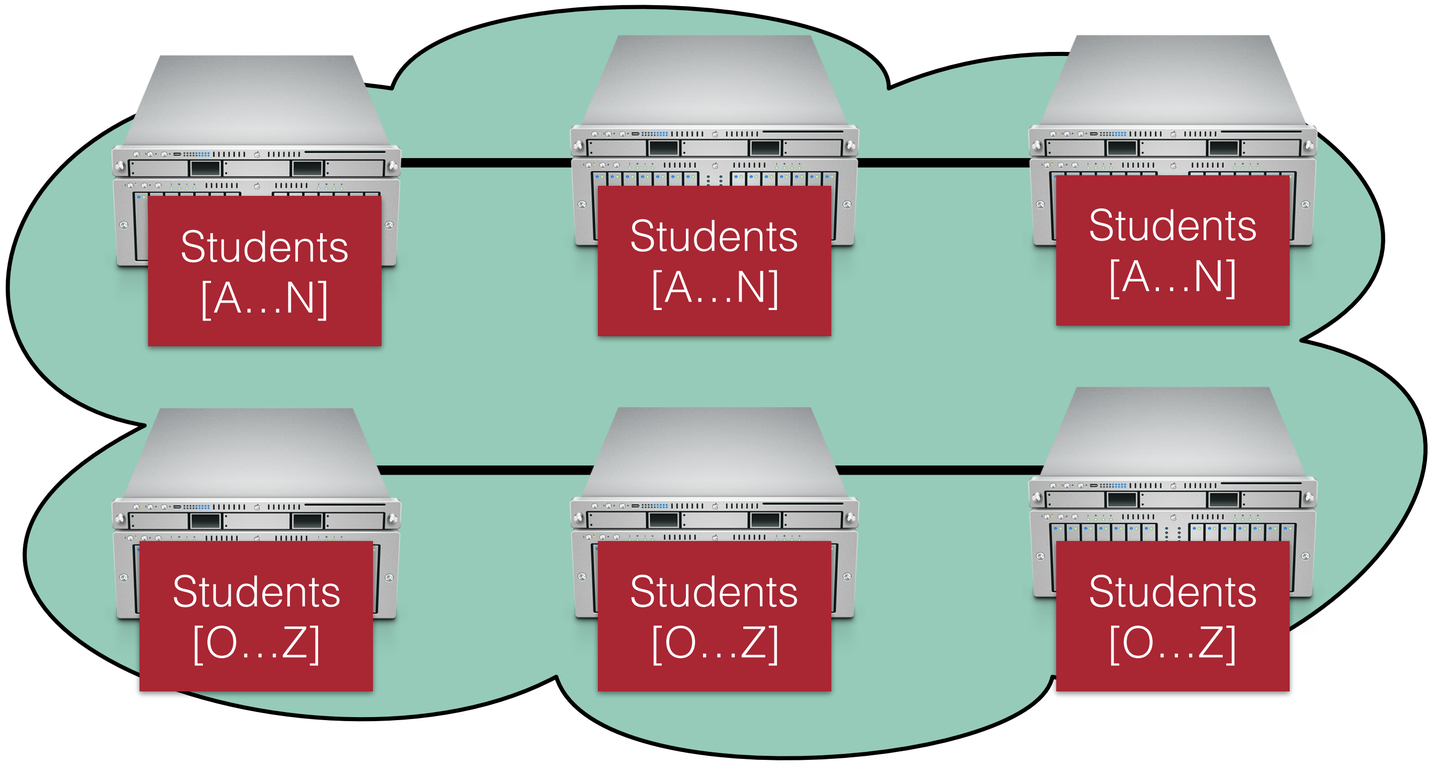
- We can solve our **discovery** problem if we can define a **consistent** way to store our data and share those rules
- Create "buckets," and use a "hash function" to map from a key to a bucket
- Example: All files starting with the letter "A" are stored on servers 1,2,3; all files starting with the letter "B" are stored on severs 4,5,6, etc.

Strawman Sharding Scheme

In this class:

40 students

16 students



Hashing

- BitTorrent & many other modern p2p systems use content-based naming
- Content distribution networks such as Akamai use consistent hashing to place content on servers
- Amazon, Linkedin, etc., all have built very large-scale key-value storage systems (databases--) using consistent hashing

Hashing

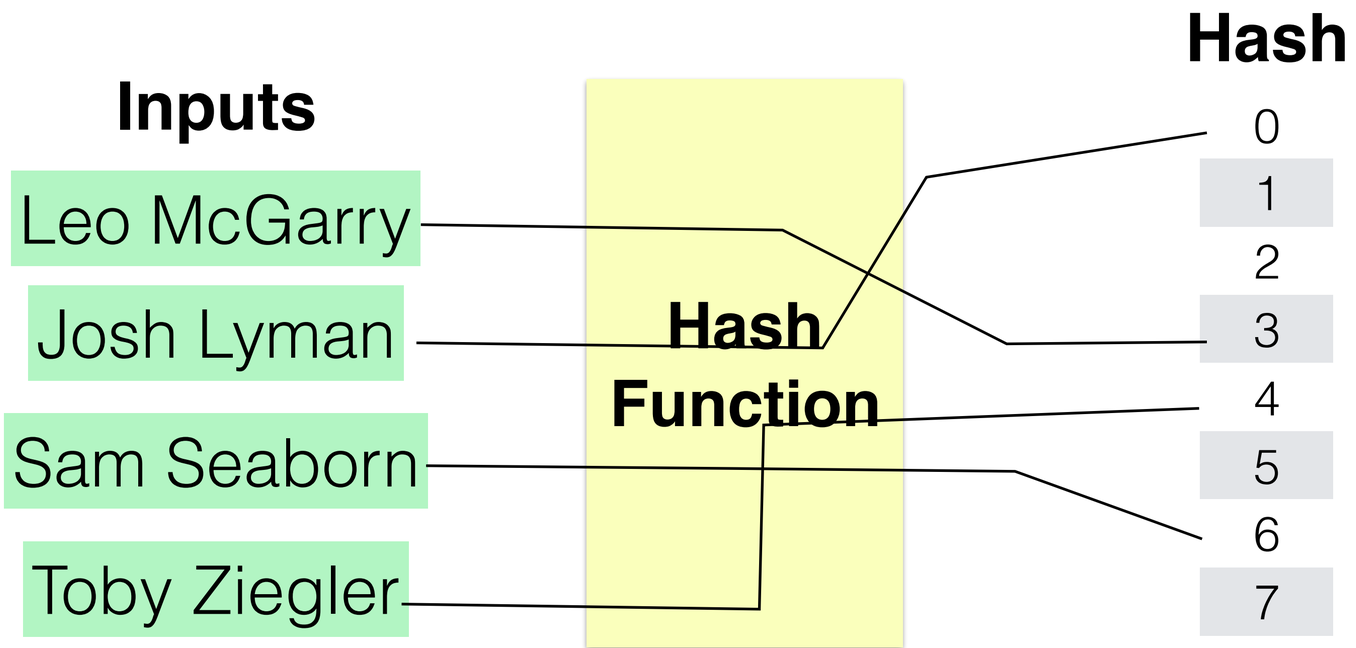
- *Idea: create a function hash(key), that for any key returns the server that stores key*
- This is called a **hash** function
- Problems?
  - No notion of duplication (what if a server goes down?)
  - What if nodes go down/come up?

Hashing

- Input: Some arbitrarily large data (bytes, ints, letters, whatever)
- Output: A fixed size value
- Rule: Same input gives same output, always; "unlikely" to have multiple inputs map to the same output

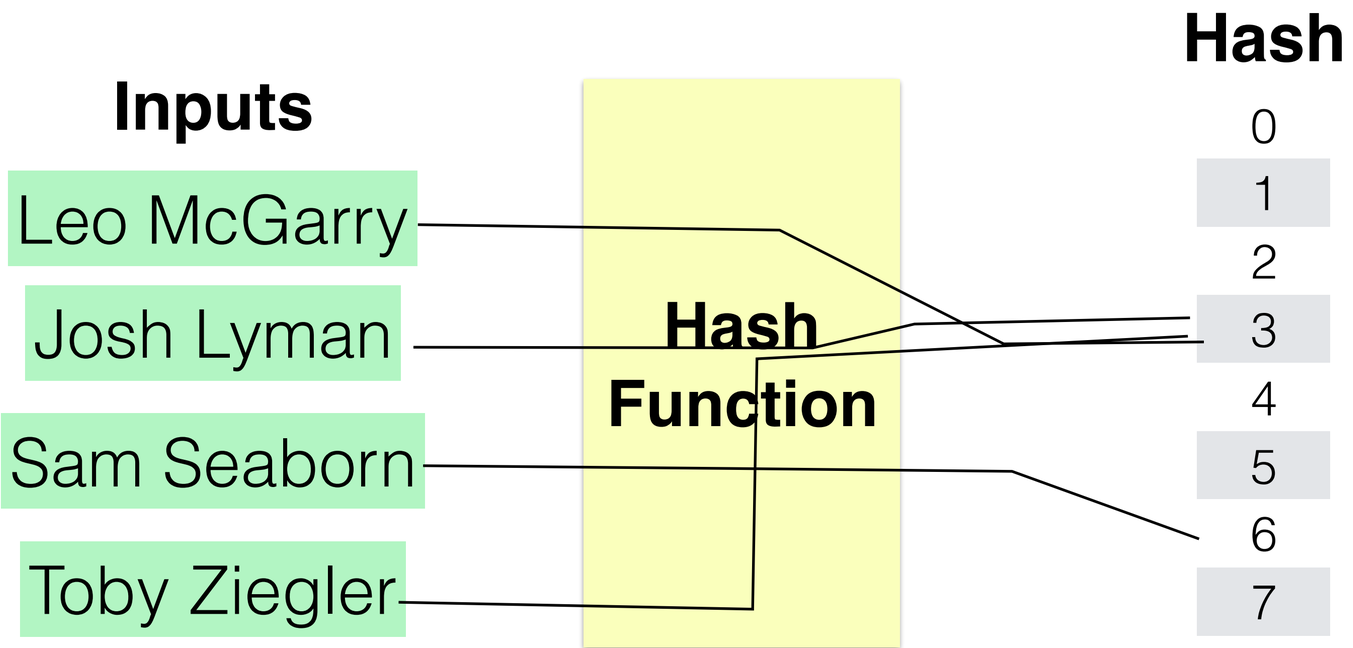
Hashing

- Compresses data: maps a variable-length input to a fixed-length output
- Relatively easy to compute
- Example:



Hashing

- The last one mapped every input to a different hash
- Doesn't have to, could be collisions



Hashing

- Hashes have tons of other uses too:
  - Verifying integrity of data
  - Hash table
  - Cryptography
  - Merkle trees (git, blockchain)

Hashing for Partitioning

Input		Hash Result		Server ID
Some big long piece of text or database key	*hash()=*	900405	*% 20 =*	5

Conventional Hashing + Sharding

- In practice, might use an off-the-shelf hash function, like sha1
- sha1(url) -> 160 bit hash result % 20 -> server ID (assuming 20 servers)
- But what happens when we add or remove a server?
  - Data is stored on what *was* the right server, but now that the number of servers changed, the right server changed too!

Conventional Hashing

Assume we have 10 keys, all integers

0, 3, 6, 9 1, 4, 7 2, 5, 8

Adding a new server

Conventional Hashing

Assume we have 10 keys, all integers

Adding a new server

8/10 keys had to be reshuffled!
Expensive!

Consistent Hashing

- Problem with regular hashing: very sensitive to changes in the number of servers holding the data!
- Consistent hashing will require on average that only K/n keys need to be remapped for K keys with *n* different slots (in our case, that would have been 10/4 = 2.5 [compare to 8])

Consistent Hashing

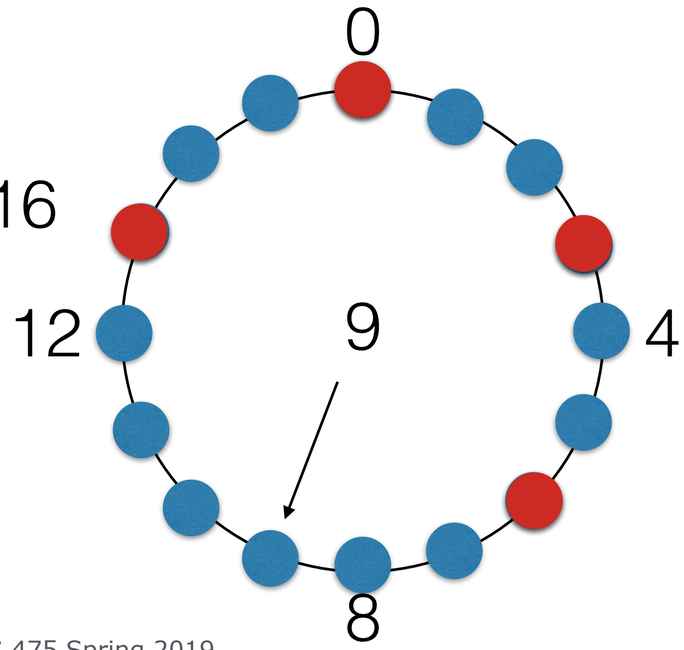
- Construction:
  - Assign each of C hash buckets to random points on mod 2^n circle, where hash key size = n
  - Map object to pseudo-random position on circle
  - Hash of object is the closest clockwise bucket

Example: hash key size is 16

Each ● is a value of hash % 16

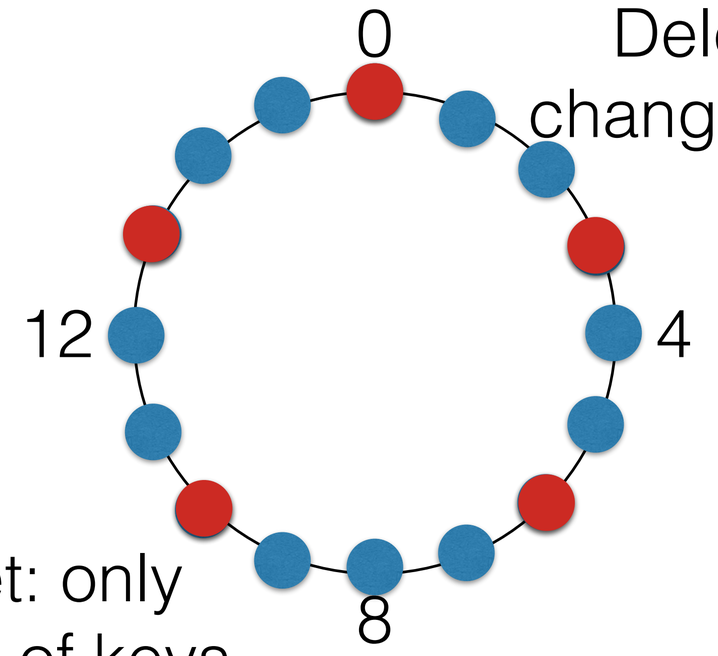
Each ● is a bucket

Example: bucket with key 9?



Consistent Hashing

It is relatively smooth: adding a new bucket doesn't change that much



Delete bucket: only changes location of keys 1,2,3

Add new bucket: only changes location of keys 7,8,9,10

CDN: How to find content?

Master server directs all requests to appropriate cache server

Big cluster of cache servers

Problem: Master becomes a huge bottleneck
Millions of requests, each request needs to be processed incredibly fast

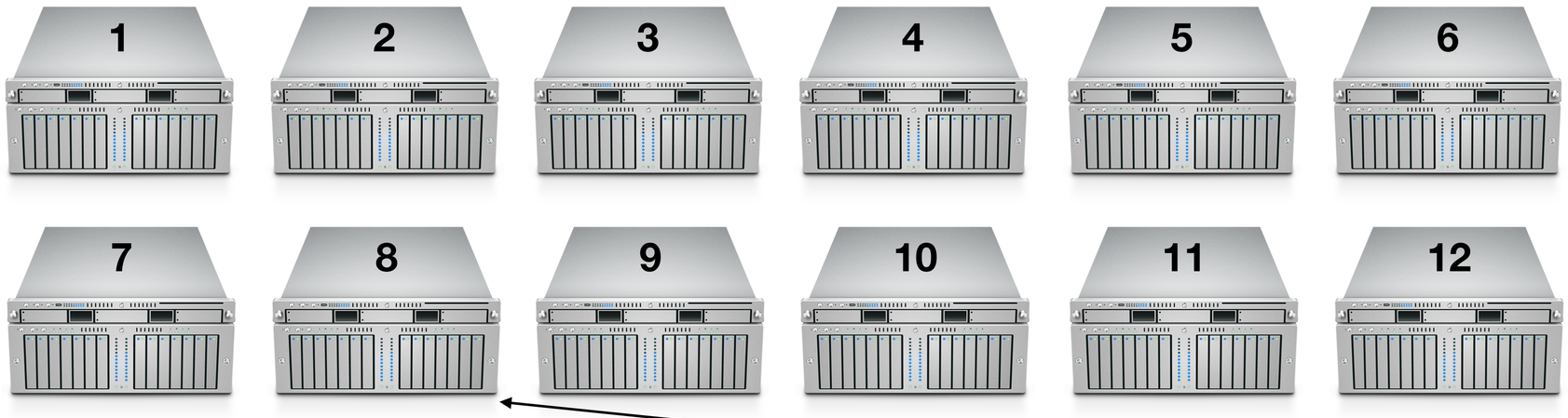
Finding the replicas

- Maintain 1000's of data centers across the internet, each with many servers
- Hash(URL's domain) maps to a server
- Akamai maintains their own DNS infrastructure, with two tiers (global and regional)
- Lookup apple.com -> akamai.net -> g.akamaitech.net -> actual cache server

Finding the replicas

- Lookup apple.com -> akamai.net -> g.akamaitech.net -> actual cache server
- The address returned for g.akamaitech.net is one that is near the client (geographically)
- The address returned by that regional server is determined by which local server has the content
  - As determined by consistent hashing

CDN: Finding Content



Consistent hash(http://www.jonbell.net/gmu-cs-475-spring-2018/homework-3/ **)=8**

This work is licensed under a Creative Commons Attribution-ShareAlike license