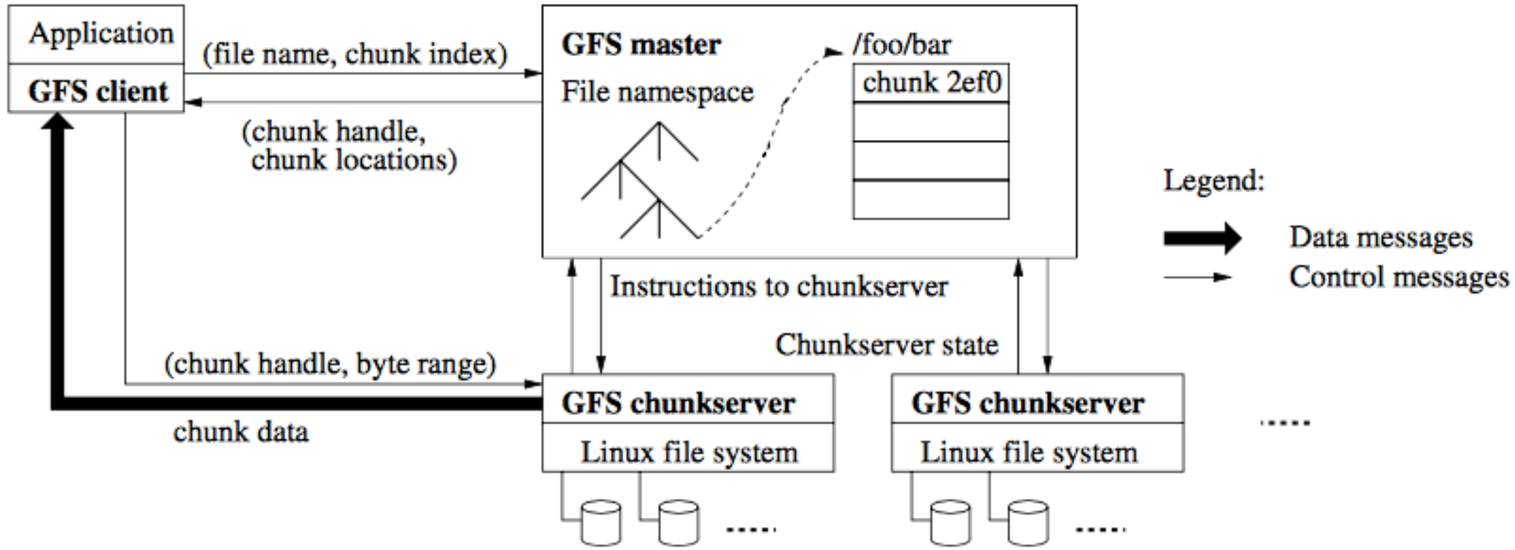
Sharding & CDNs

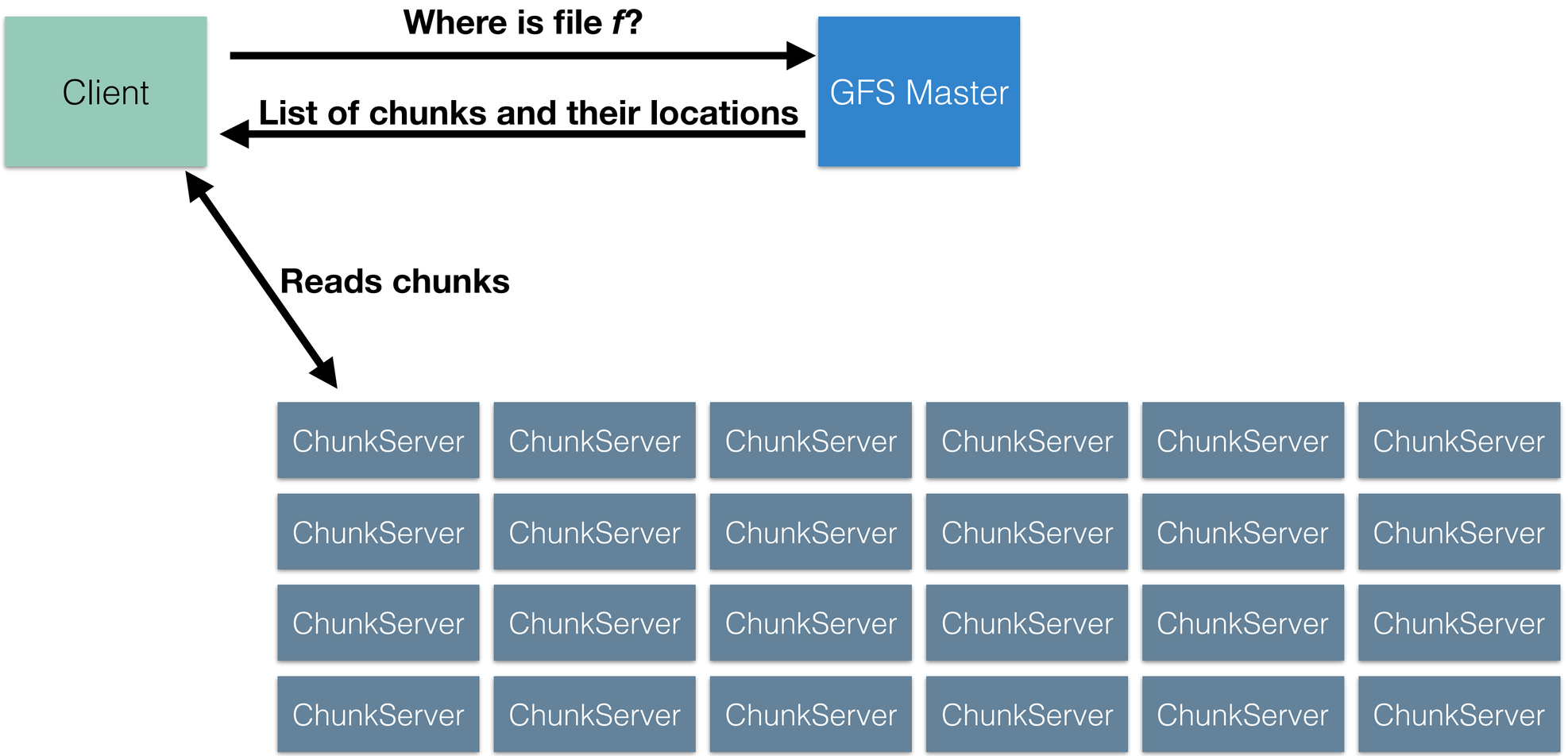
CS 475, Fall 2019
Concurrent & Distributed Systems

GEORGE MASON UNIVERSITY

Review: GFS Architecture



Review: GFS - Reads



Today

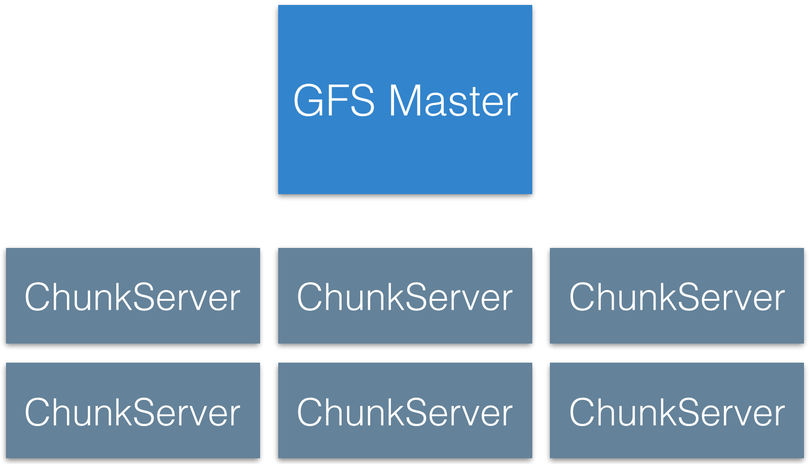
- How do we get rid of the “master” server that keeps track of metadata?
  - Sharding & CDNs
- Reminder - Project is out next week!
  - Fault-tolerant, sequentially consistent replicated key value store
  - Can do in a group (1 to **3** students per group)

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?

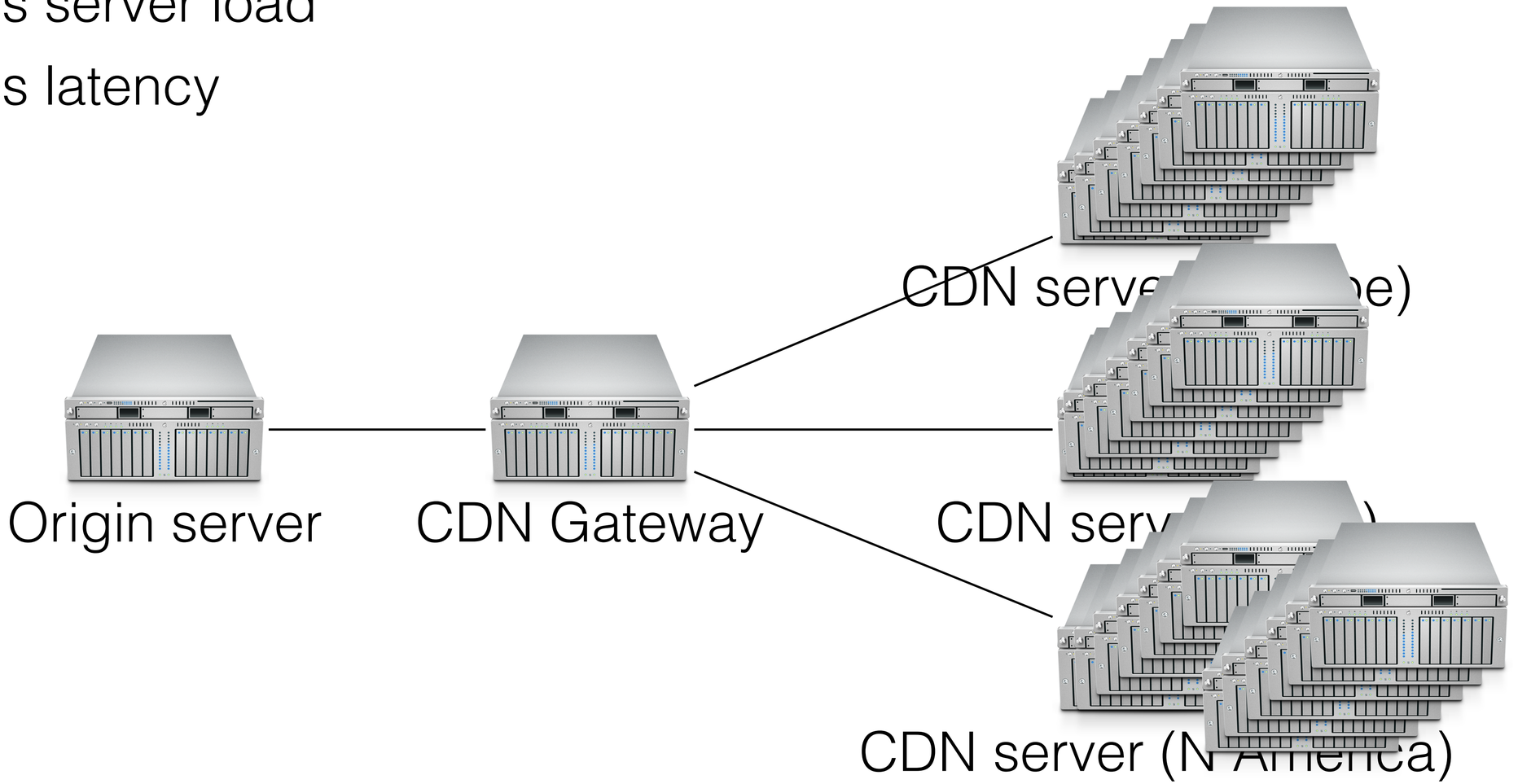
Why not use a central server to find data?

- 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

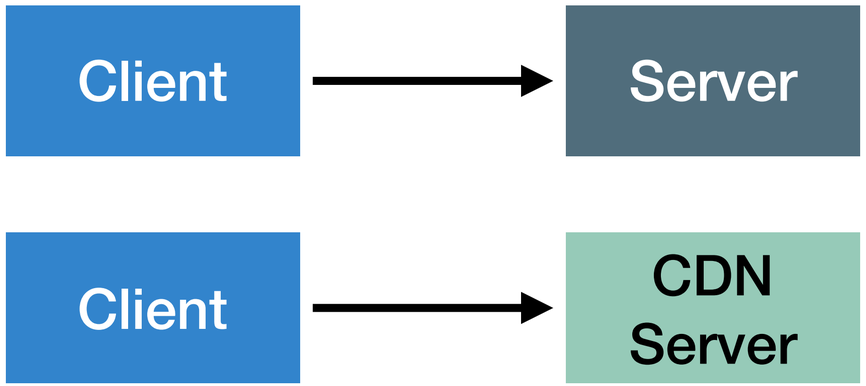
- Motivating scenario:
  - Prof Bell has a popular web page, doesn't want to be limited by his server's capacity or location
- What is the high level problem?
  - We will scatter caches across the internet, direct browser to nearest cached copy
  - If not cached nearby, fetch and store in cache
- Why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

Typical Web Workload

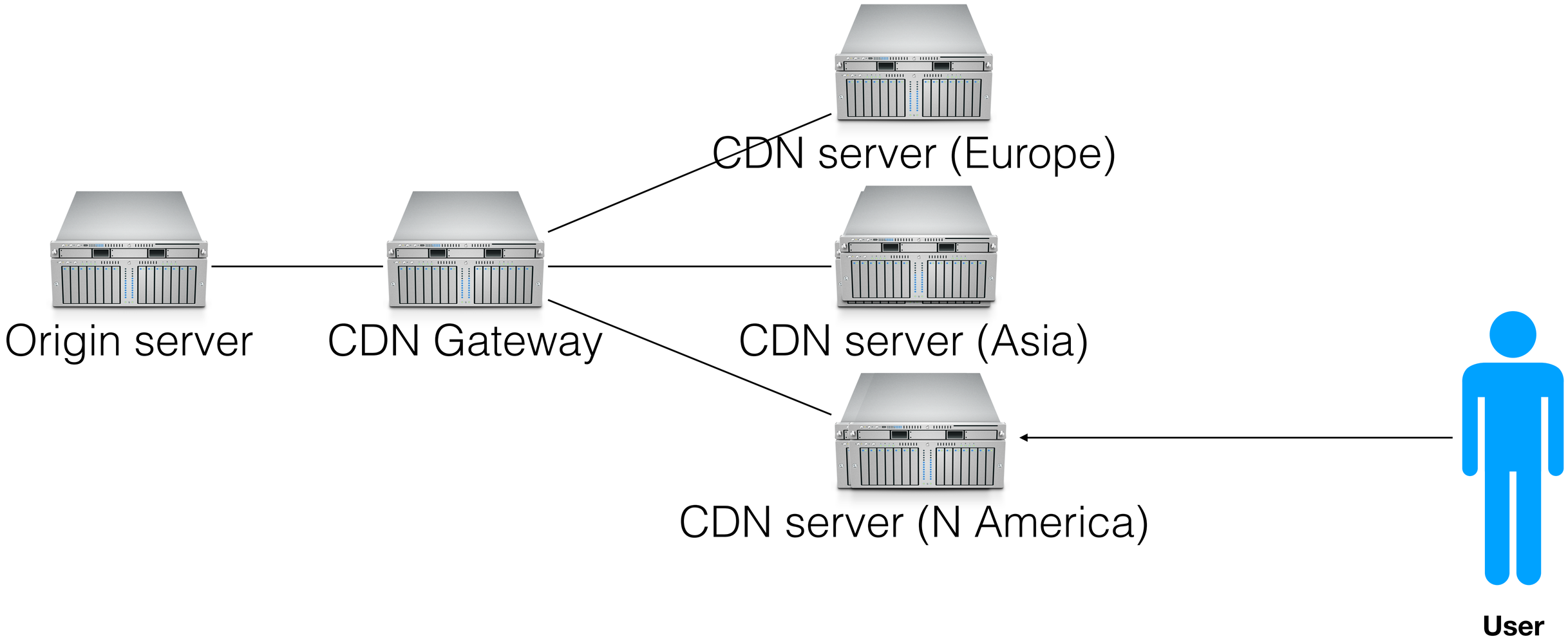
- Many small objects per page, but “long tail” of file size
- Pages have embedded references to content
- Implications?
  - Lots of requests! -> potential for latency issues
  - Some big requests -> potential for throughput issues

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 Idea


CDN server (Europe)
Origin server
CDN Gateway
CDN server (Asia)
CDN server (N America)
User

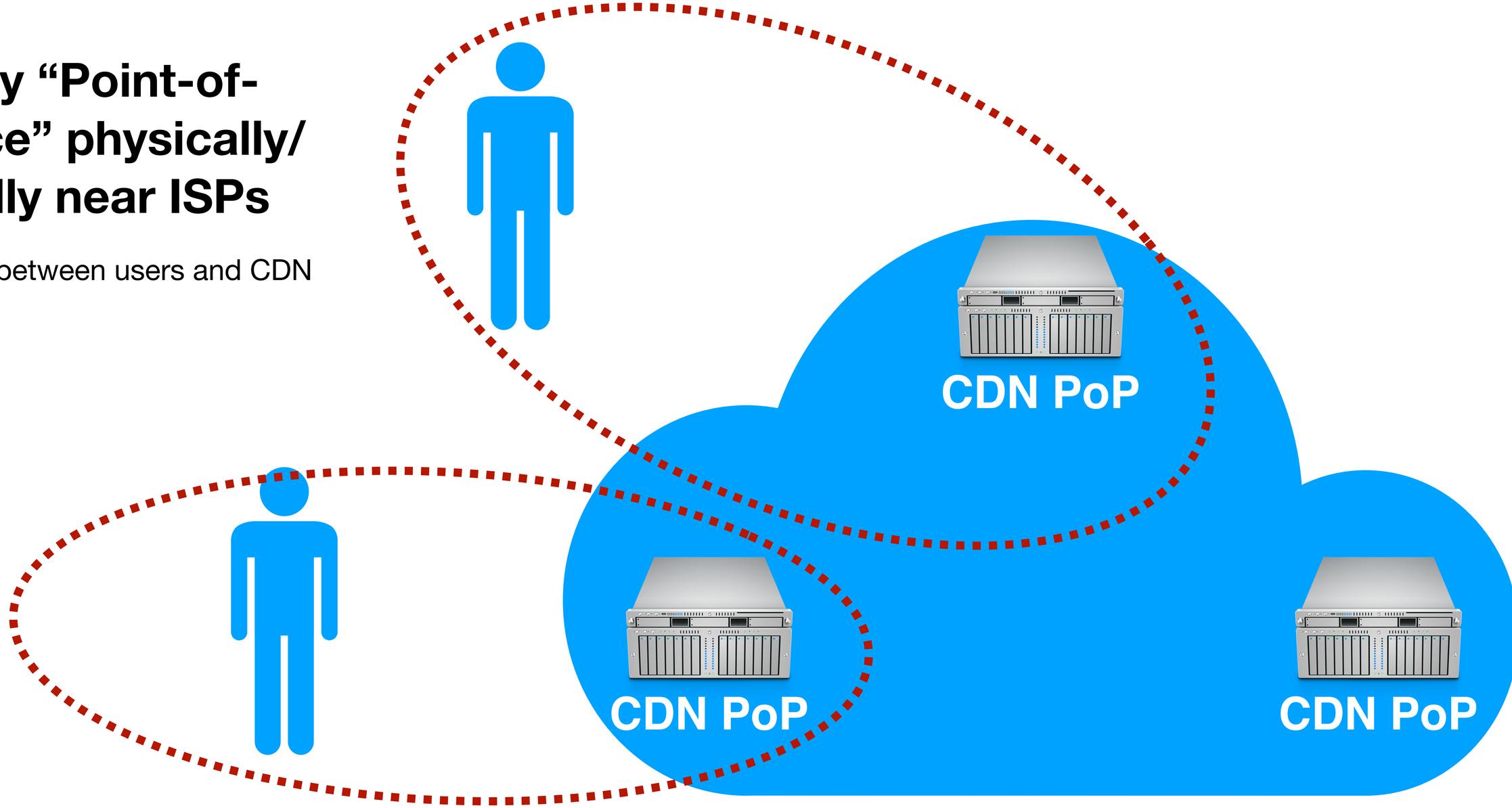
CDN Challenges

- 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

Where to Replicate

Deploy "Point-of-Presence" physically/ logically near ISPs

Reduce hops between users and CDN



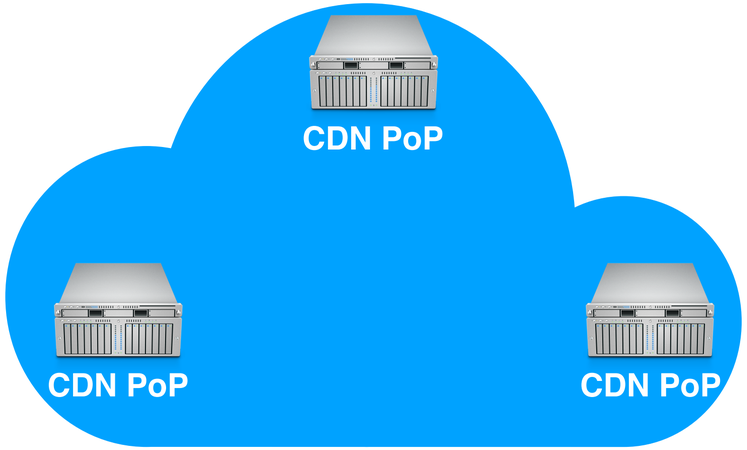
How to Replicate

Rack(s) of servers with lots of RAM
Directly connected to internet backbone
(hundreds of Gbps)

CDN PoP

How to find PoP

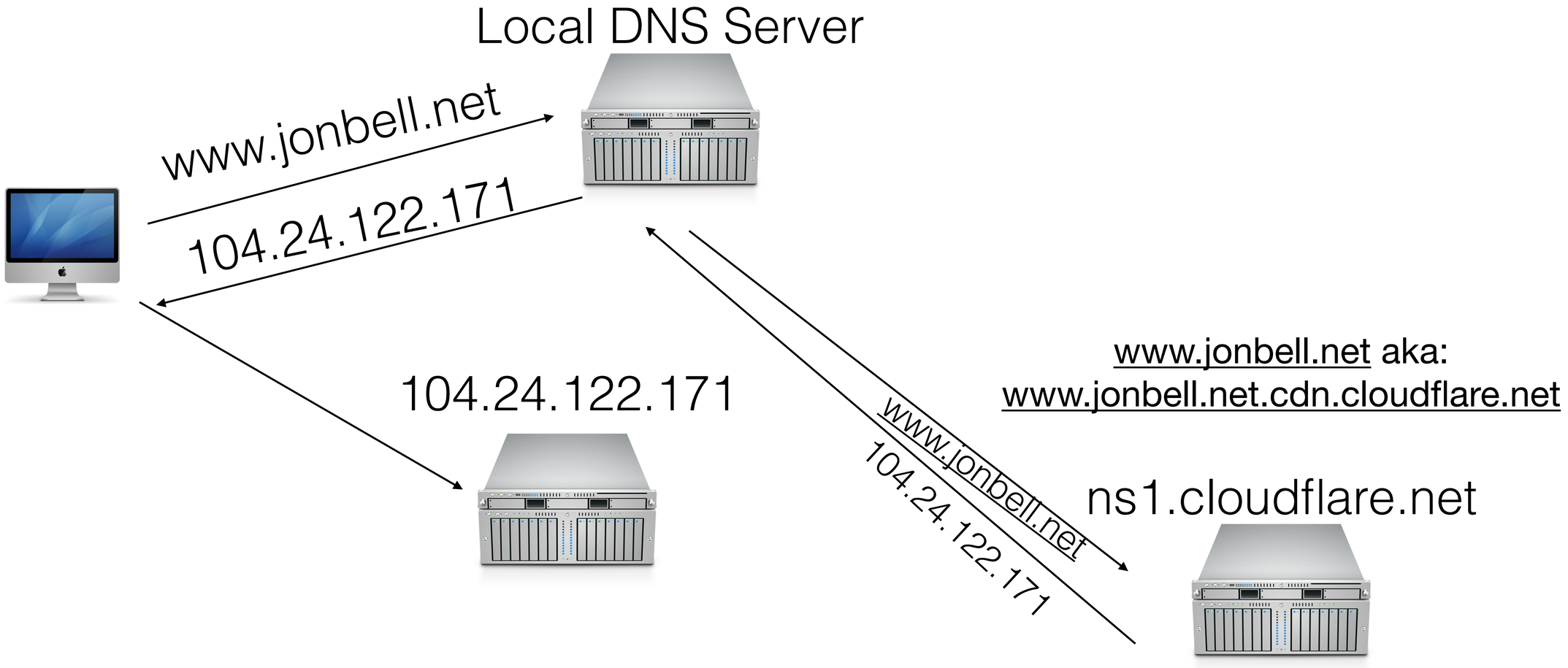
- Which point-of-presence to pick?
  - Based on geography? Round trip time?
  - Throughput of each PoP? Load?
- How to do the redirection?
  - As part of the application? (redirect requests)
  - As part of naming? (DNS alias record)



How to find PoP

- Client does normal DNS lookup
- DNS is setup to map to regionally best PoP
  - How? Return a Name Server record for the correct PoP
  - Large (hours) time-to-live on this record (want lots of caching)
- Regional PoP DNS server resolves to a specific server within that PoP
  - Want server most likely to have the page cached already
  - Very small (<5 minutes) TTL

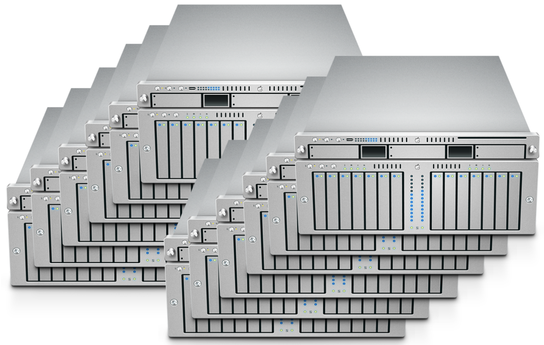
How to Find PoP



How to find content inside of PoP?

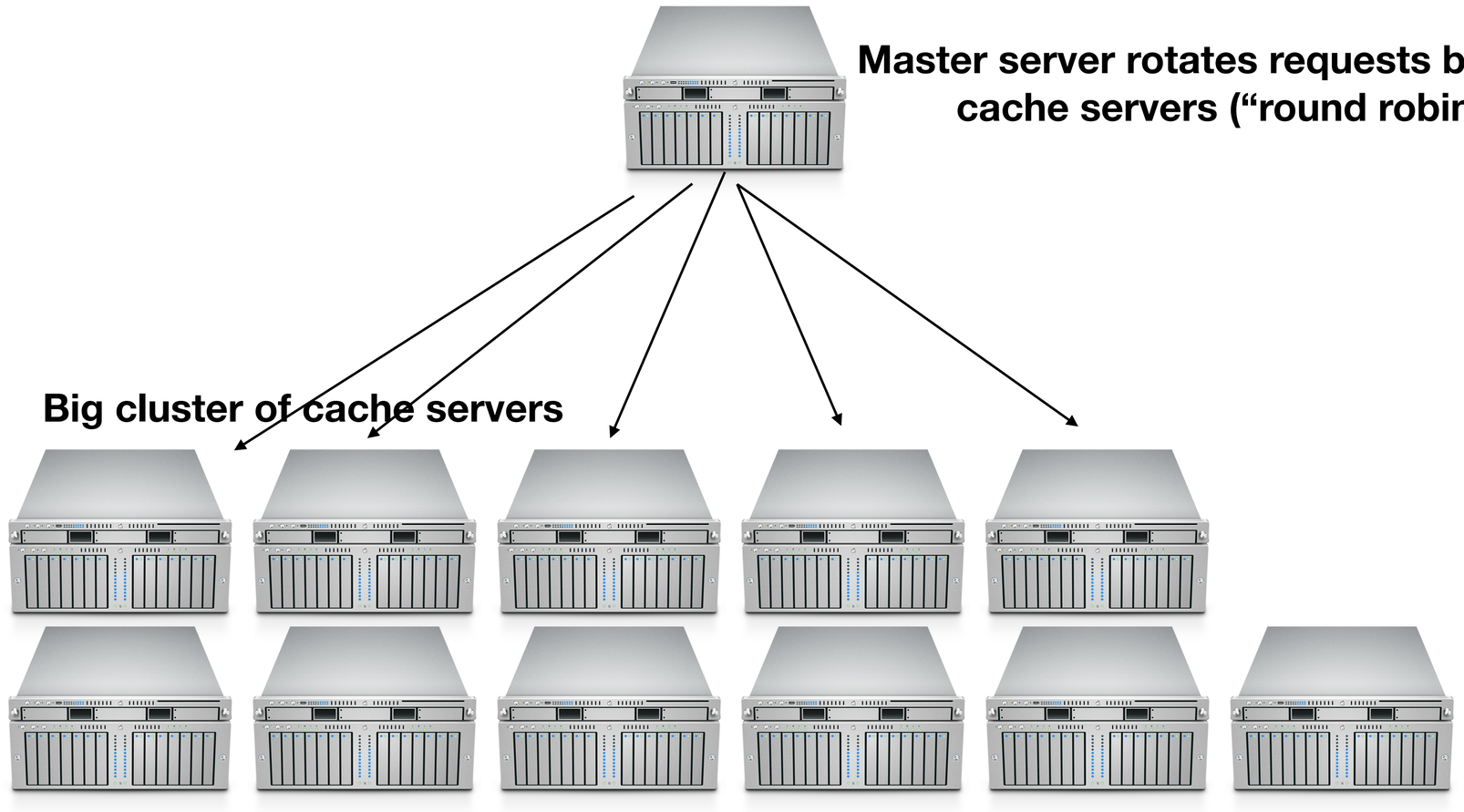
- Desire super, super low latency
- Serving a single request is probably very very cheap (e.g. read a 1KB file from memory/SSD and return it)
- But we want to serve billions of these requests at once
- VERY important that we can route requests fast

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: No specialization - each cache server at worst case caches entire internet!

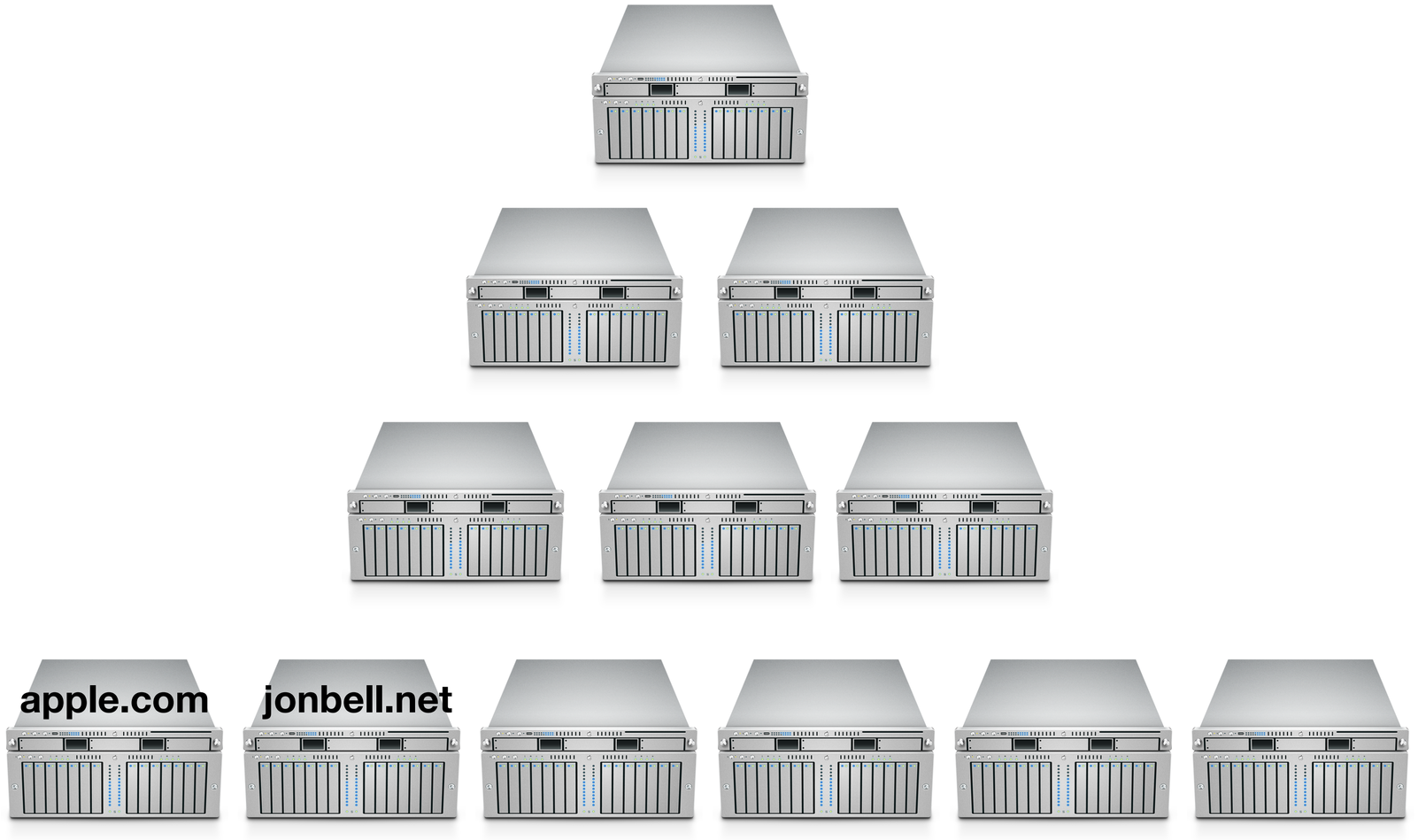
CDN: How to find content? (Strawman)

Master server directs all requests to appropriate cache server

Big cluster of cache servers

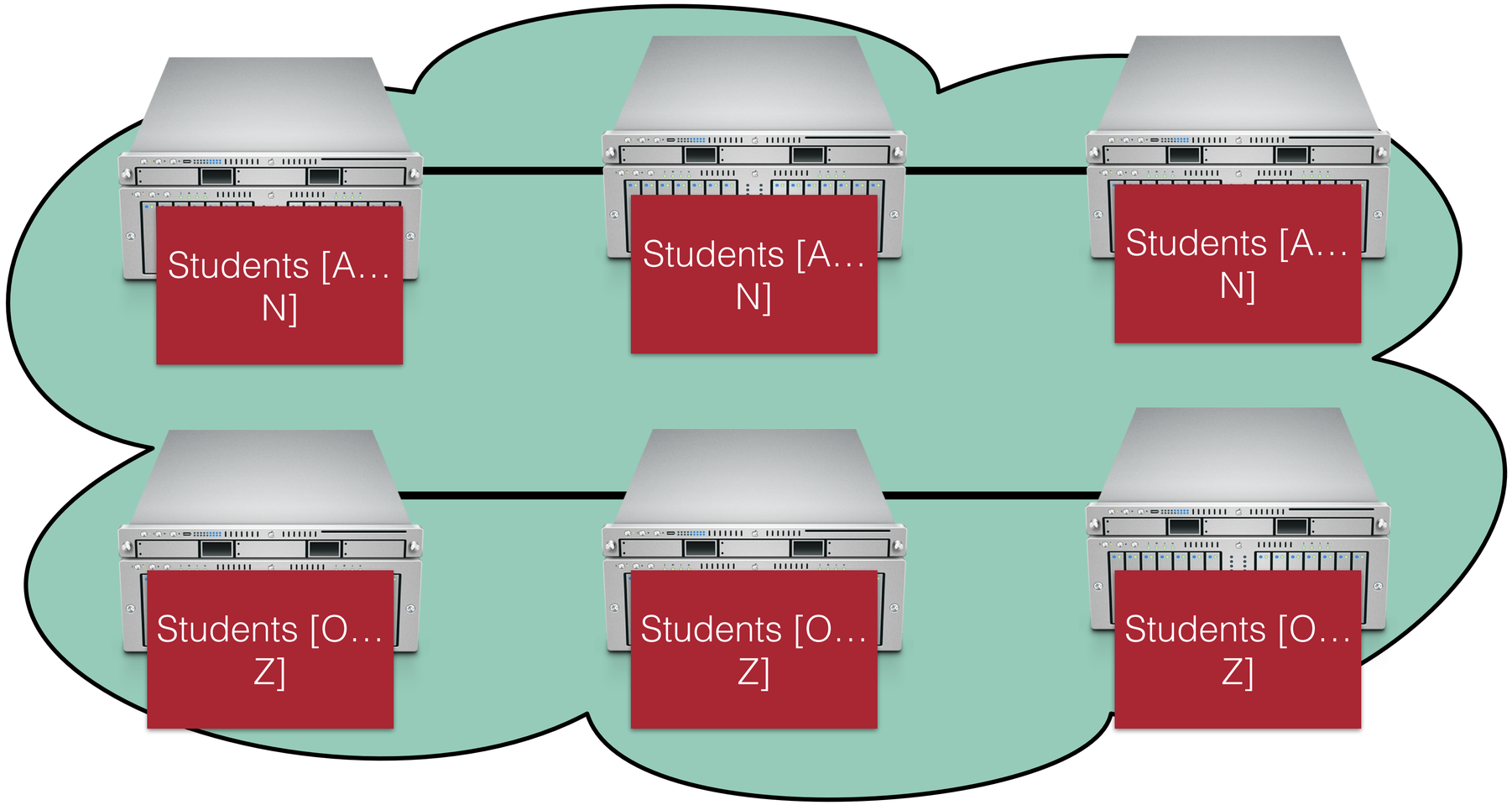
Problem: Master becomes a huge bottleneck
Millions of requests, each request needs to be processed incredibly fast

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

Strawman: Sharding (Partitioning by Key)



Strawman: Sharding (Partitioning by Key)

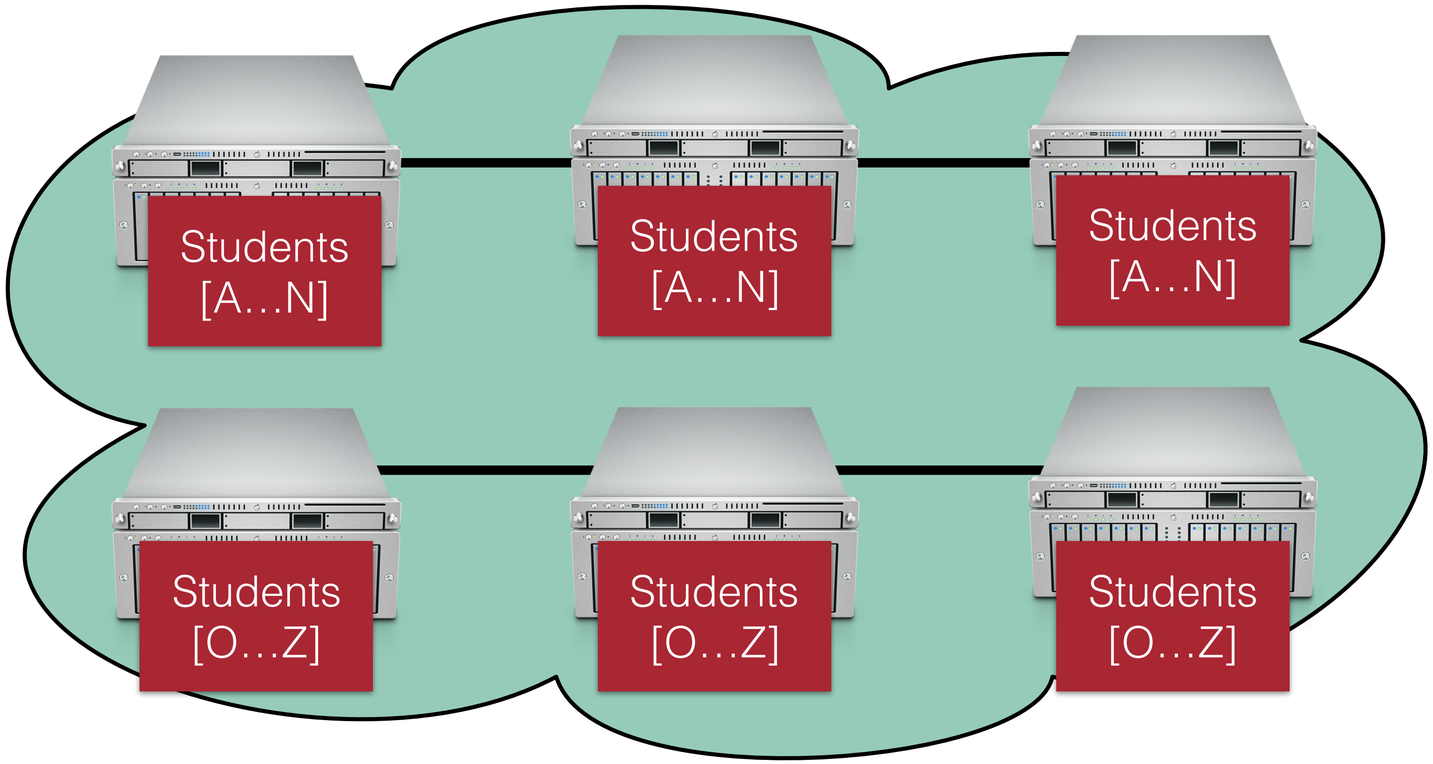
- We can solve our **discovery** problem if we can define a **consistent** way to store our data and share those rules
- Create "buckets," and use a "hash function" to map from a key to a bucket
- Example: All files starting with the letter "A" are stored on servers 1,2,3; all files starting with the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

Strawman Sharding Scheme

In this class:

38 students

28 students



Hashing

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

Hashing

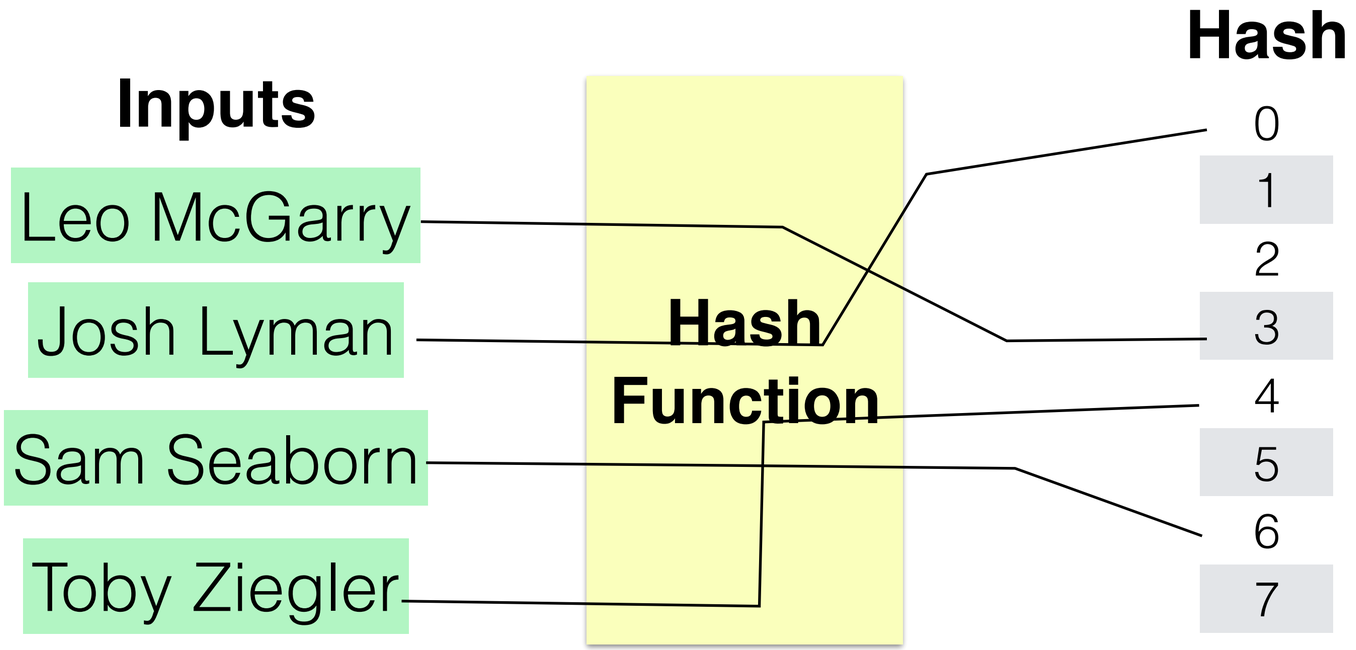
- *Idea: create a function hash(key), that for any key returns the server that stores key*
- This is called a **hash** function
- Problems?
  - No notion of duplication (what if a server goes down?)
  - What if nodes go down/come up?

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

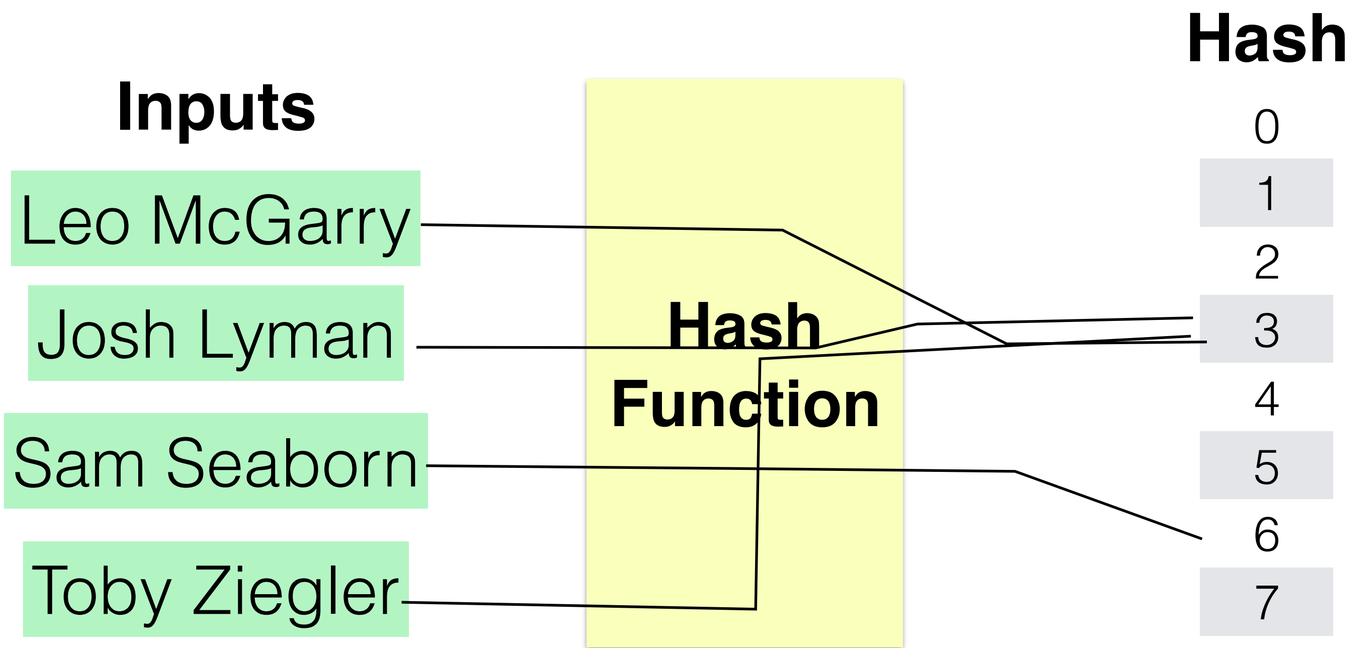
Hashing

- 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



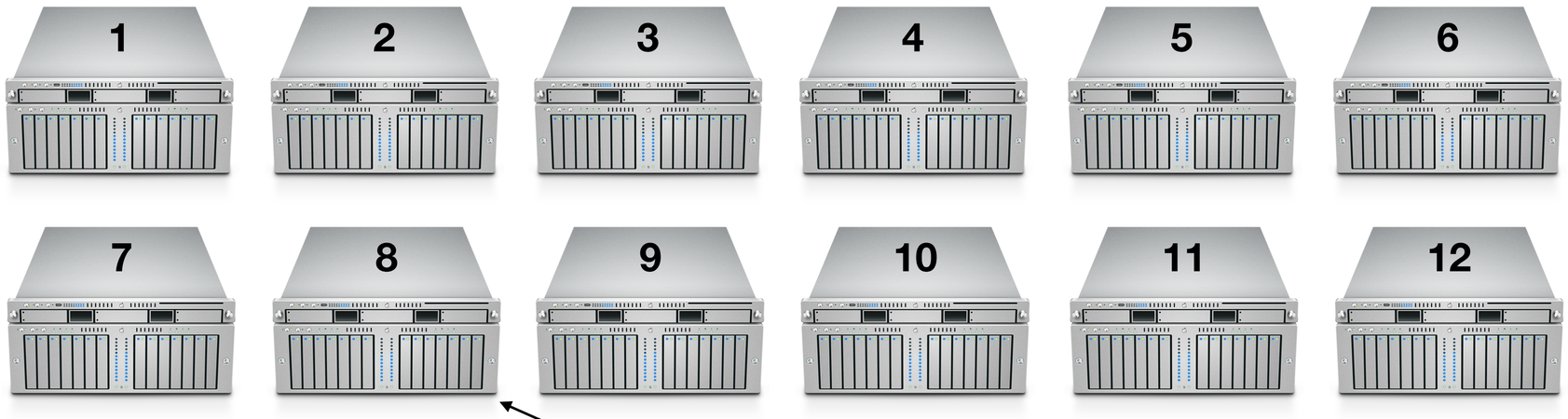
Hashing

- 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

CDN: Finding Content



hash(https://www.jonbell.net/gmu-cs-475-fall-2019/ **)=8**

Master server takes hash of entire URL

Hash-Partitioning

- Problems:
  - No data duplication (what if crash?)
  - Who keeps track of which servers are up/down?
  - Adding/removing servers is very hard

hash(https://www.jonbell.net/gmu-cs-475-fall-2019/ **)=8**

Master server takes hash of entire URL

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

What happens if we change to 21?

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

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

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

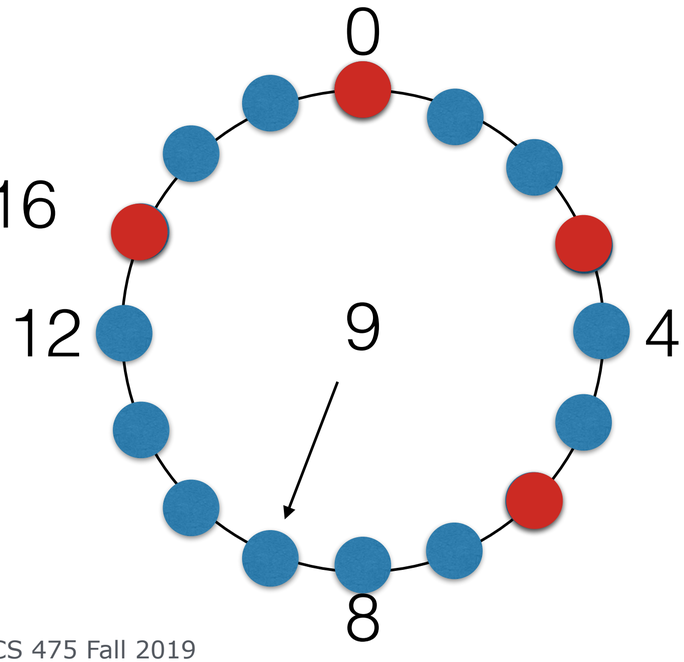
- Construction:
  - Assign each of C hash buckets to random points on mod 2^n circle, where hash key size = n
  - Map object to pseudo-random position on circle
  - Hash of object is the closest clockwise bucket

Example: hash key size is 16

Each ● is a value of hash % 16

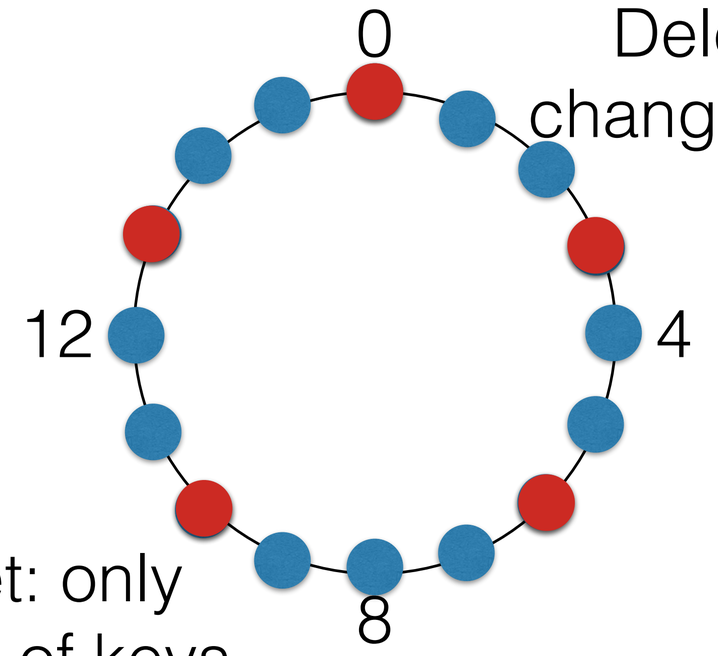
Each ● is a bucket

Example: bucket with key 9?



Consistent Hashing

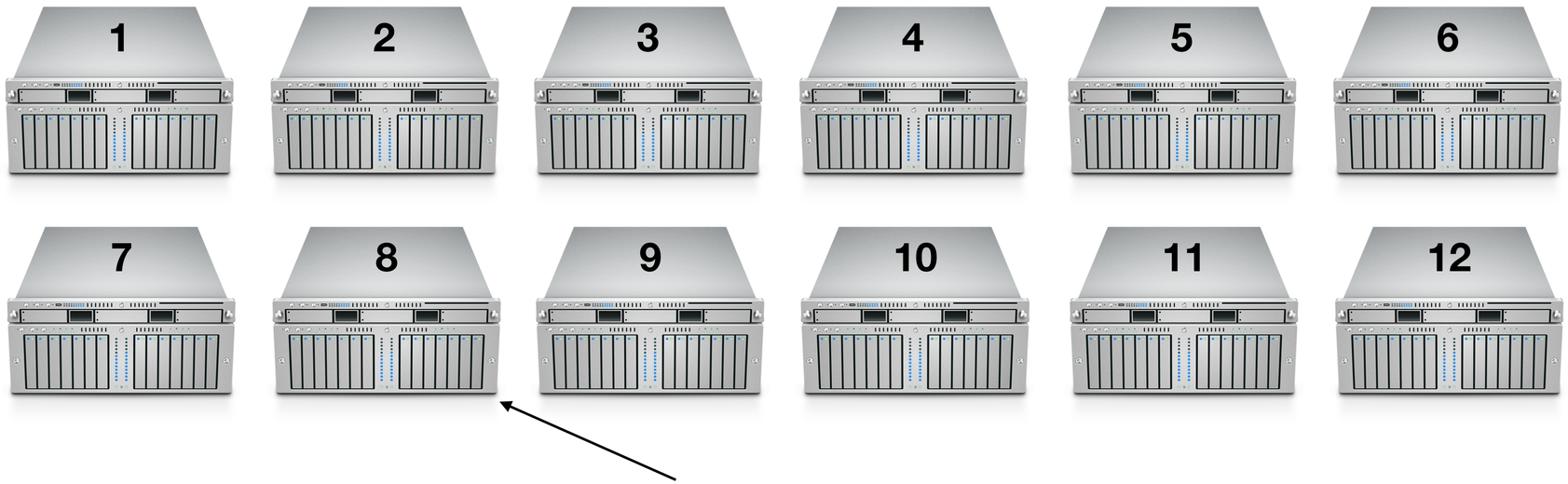
It is relatively smooth: adding a new bucket doesn't change that much



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

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

CDN: Finding Content



consistenthash(https://www.jonbell.net/gmu-cs-475-fall-2019/ **)=8**

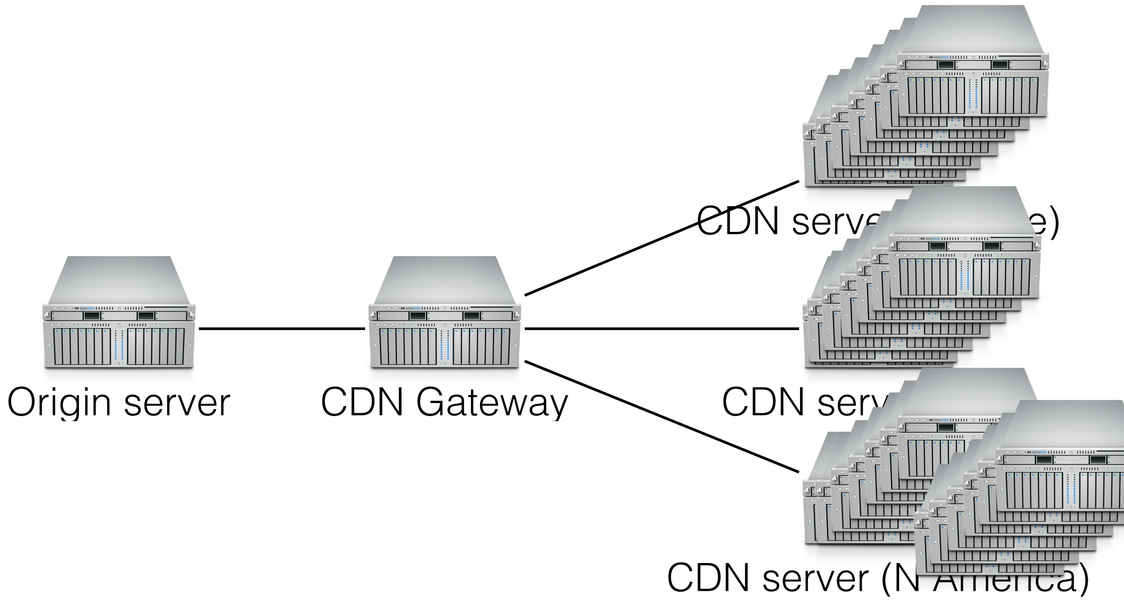
Master server takes hash of entire URL

Finding the replicas with DNS

- Client does normal DNS lookup
- DNS is setup to map to regionally best PoP
  - How? Return a Name Server record for the correct PoP
  - Large (hours) time-to-live on this record (want lots of caching)
- Regional PoP DNS server resolves to a specific server within that PoP
  - Want server most likely to have the page cached already
  - Very small (<5 minutes) TTL
  - **Uses consistent hashing to choose the local server for the URL**

Cache Consistency

- Remember, goal: make it look like the origin server is just really fast, but:
  - We have lots of servers caching that data
  - How do we keep them all up-to-date, or at least consistent?
  - Saving grace: data can only be written *in one place*
- Tradeoffs:
  - Complexity of implementation
  - Scalability
  - Consistency



Cache Consistency Approach: Broadcast Invalidations

- Every time any data is updated, each potential caching site is notified
  - Doesn't matter if site has data cached or not
  - Clears out caches, does not put new object in there though
- Pros:
  - Simple to implement
- Cons:
  - Wasted traffic (if data not cached already)
  - Hard to scale if require invalidates to be acknowledged before allowing write

Cache Consistency Approach: Check on Use

- Reader checks the origin copy before each use
  - Fetch only if the cache is stale
  - Has to be done at level of entire file, for each read
- Pros:
  - Simple to implement
  - No state needed
- Cons:
  - Wasted traffic if not updated
  - Defeats the performance goals (throughput? latency?)

Cache Consistency Approach: TTLs

- Assume that cached data will be valid for some amount of time
- Each page has a TTL (time-to-live)
- No communication during the TTL period, then communicate to re-establish
- Pros:
  - Simple to implement, no server state
- Cons:
  - May allow for divergence in consistency

CDN Update Propagation

- Static content
  - Images, photos, static sites, CSS files, JS files, etc.
  - Consistency set by TTL, set by the owner of the content
- Dynamic content
  - Pages that update, e.g. blog
  - Broadcast invalidation to “purge” objects
- Edge-based applications
  - The entire application runs in the CDN using consistent replication (future)

CDN Summary

- Caching is the only way to improve latency across the internet (can not beat speed of light)
- CDNs move data closer to the user, balance load and failures
- Use consistent hashing
- Many design decisions for cache consistency

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