Distributed Systems

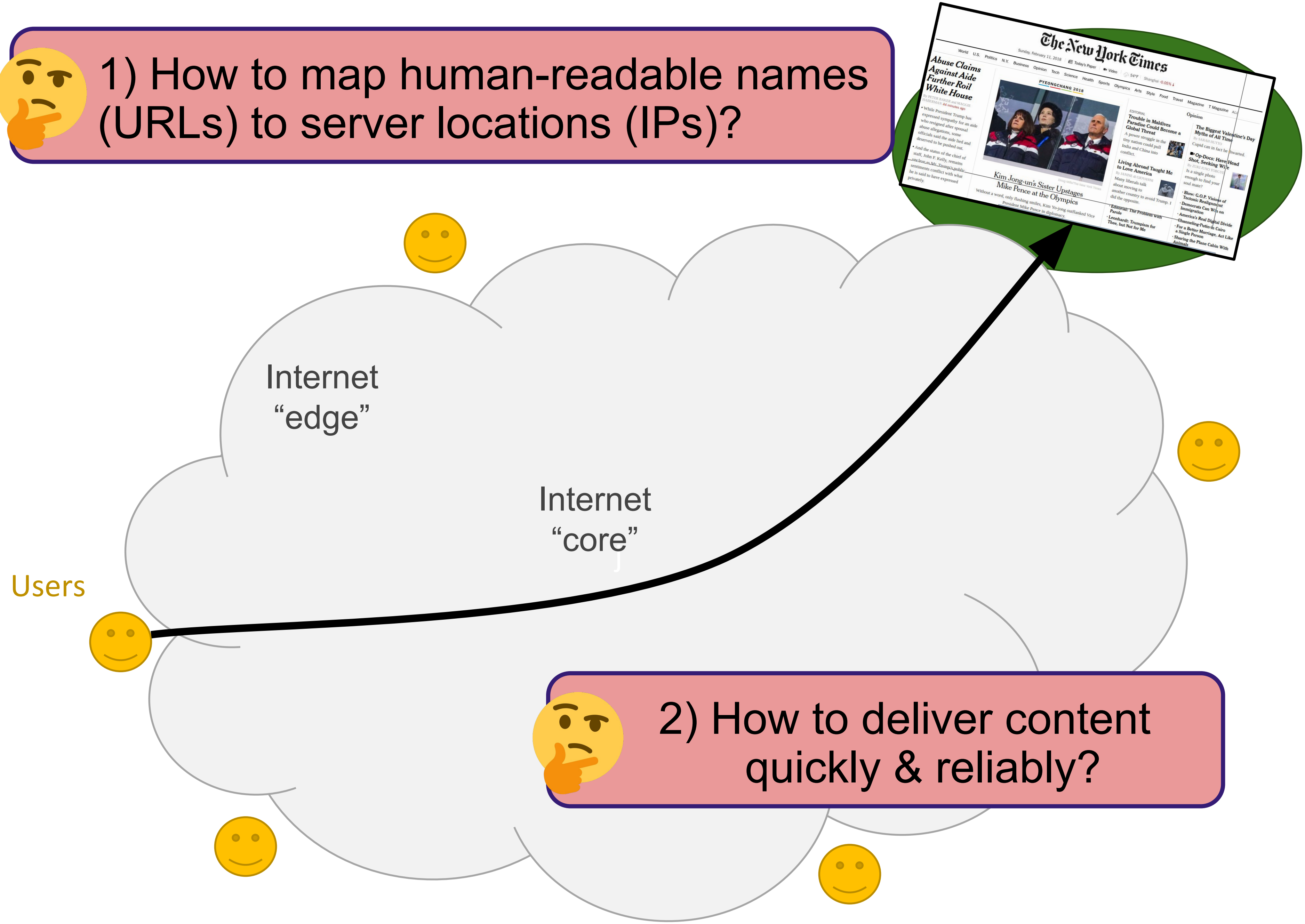
15-440/640

Fall 2018

18 – Internet Content Delivery Case Study: DNS & CDNs

Readings: Tanenbaum 5.1-5.5, 7.6. Optional readings: readings linked from website

1) How to map human-readable names (URLs) to server locations (IPs)?



2) How to deliver content quickly & reliably?

Topics Today

1) Naming at Internet Scale
    - DNS - one of the world's largest databases
    - DNS Architecture
    - Robustness and Security Implications
2) Content Distribution at Internet Scale
    - CDNs - some of the world's largest distributed systems
    - Design Decisions
    - Consistent Hashing for Scaling and Load Balancing

Why Naming is Important

Naming enables

Passing of references to objects

Deferring decision on meaning/binding

Examples

- User names → dsberger
- Email → dsberger@cmu.edu
- File name → /usr/dsberger/foo.txt
- URLs → http://www.funnycatsite.com

Name Discovery

Well-known name
- www.google.com, port 80…

Broadcast
- Advertise name → e.g. 802.11 Beacons

Query
- Use google

Broadcast query
- Ethernet ARP

Use another naming system
- DNS returns IP addresses

Physical rendezvous
- Exchange info in the real world

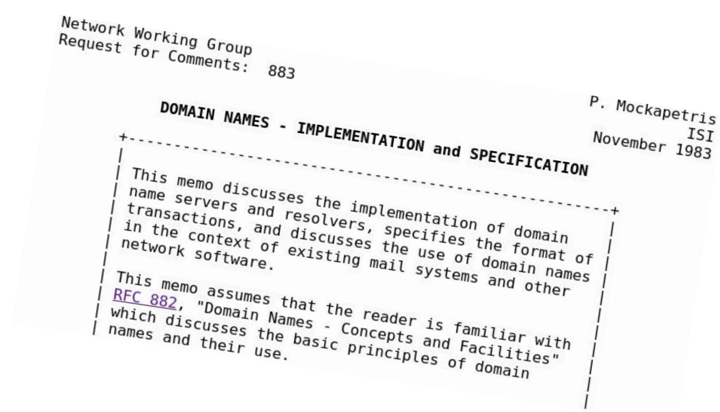
What would you demand from an Internet naming system?

Internet Name Discovery

Challenges/Goals:

- Scalability
- Decentralized maintenance
- Robustness
- Global scope
  - Names mean the same thing everywhere

Domain Name System, 1984



Network Working Group
Request for Comments: 883

P. Mockapetris
ISI
November 1983

DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION

This memo discusses the implementation of domain name servers and resolvers, specifies the format of transactions, and discusses the use of domain names in the context of existing mail systems and other network software.

This memo assumes that the reader is familiar with RFC 882, "Domain Names - Concepts and Facilities" which discusses the basic principles of domain names and their use.

DNS trades off consistency
for all these goals

DNS-RPC Format

Conceptually, we use RPCs to query a database with billions of resource records (RR).

RR format: **(class, name, value, type, ttl)**

Basically, only one class: Internet (IN)

Types for IN class:

- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is name of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

Properties of DNS Host Entries

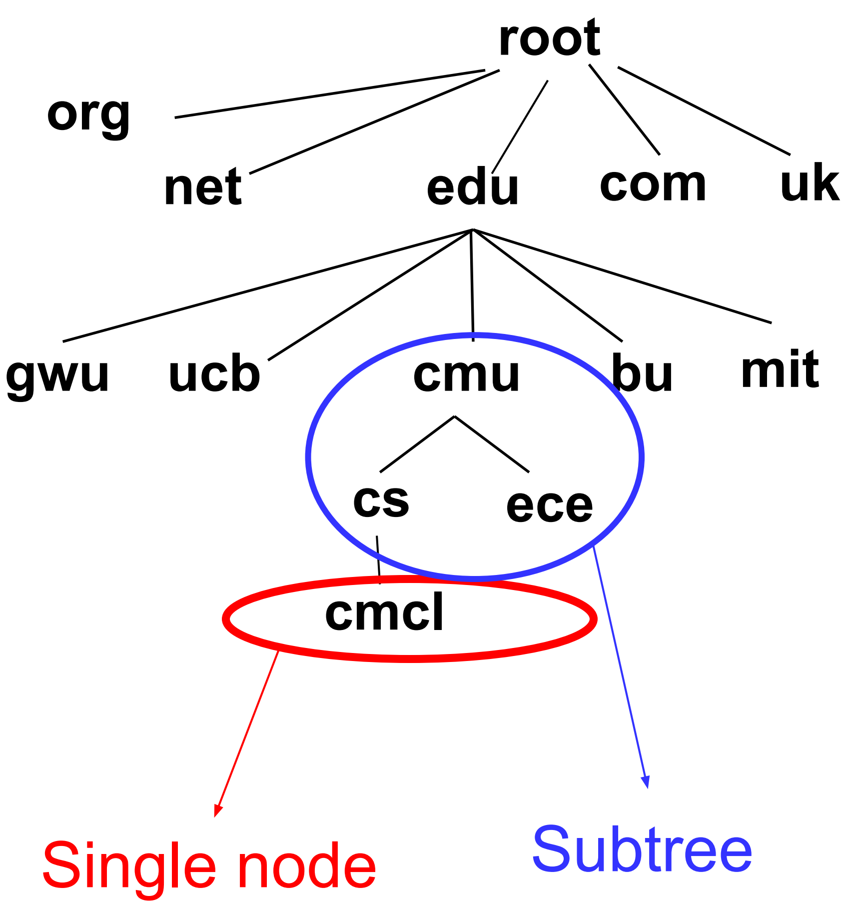
Many kinds of mappings are possible:

- Simple case: 1-1 mapping - domain name to IP
  - `kittyhawk.cmcl.cs.cmu.edu` maps to `128.2.194.242`
- Multiple domain names - same IP:
  - `eecs.mit.edu, cs.mit.edu` both map to `18.62.1.6`
- Single domain name - multiple IPs:
  - `nytimes.com` maps to 4 different IP addresses

When could this be useful?

- Some valid domain names don't map to any IP
  - for example: `cmcl.cs.cmu.edu`

The DNS Hierarchy



Each node in hierarchy stores a list of names that end with same suffix

- Suffix = path up tree

Each edge is implemented via a DNS record of type NS.

Zone = contiguous section of name space

- E.g., Complete tree, single node or subtree

A zone has an associated set of name servers

- Must store list of names and tree links

DNS Design: Zone Delegation

Zones are created by convincing owner node to create/delegate a subzone

- Records within zone stored in multiple redundant name servers (master/slave)
- Slaves updated by zone transfer of name space
  - Zone transfer is a bulk transfer of the “configuration” of a DNS server – uses TCP to ensure reliability

Example:

- CS.CMU.EDU created by CMU.EDU administrators
- Who created CMU.EDU or .EDU?

DNS: Root Name Servers

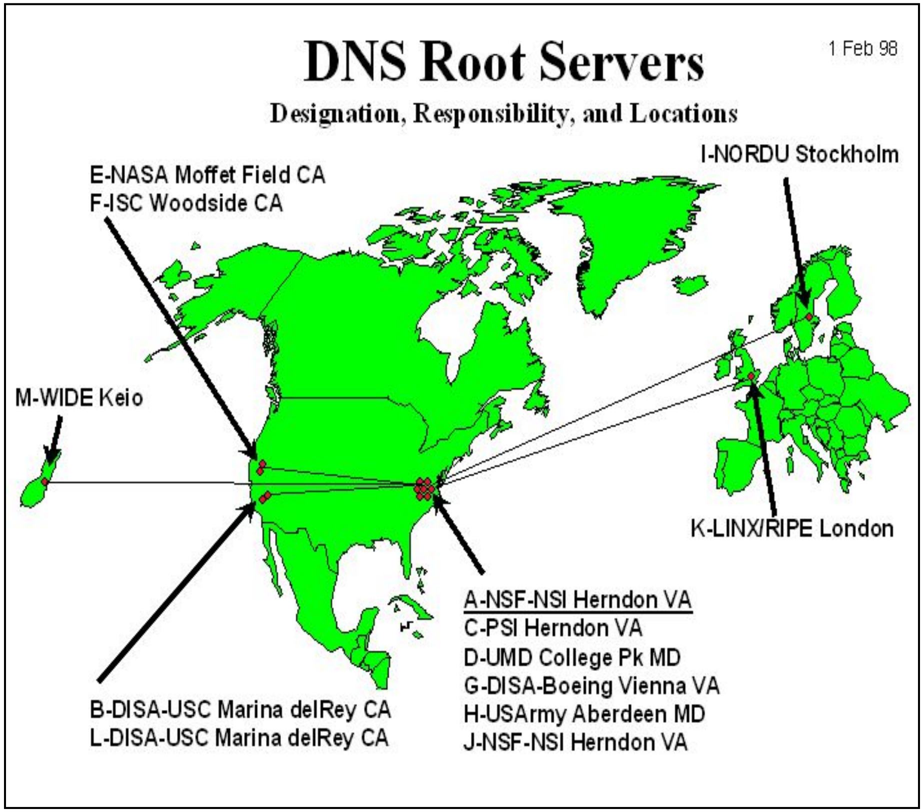
Responsible for “root” zone

~13 root name servers

- Currently {a-m}.root-servers.net

Local name servers contact root servers when they cannot resolve a name

- Configured with well-known root servers
- Newer picture → www.root-servers.org



Architecture and Robustness

DNS servers are replicated

- Available if ≥1 replica up
- Load balance replicas

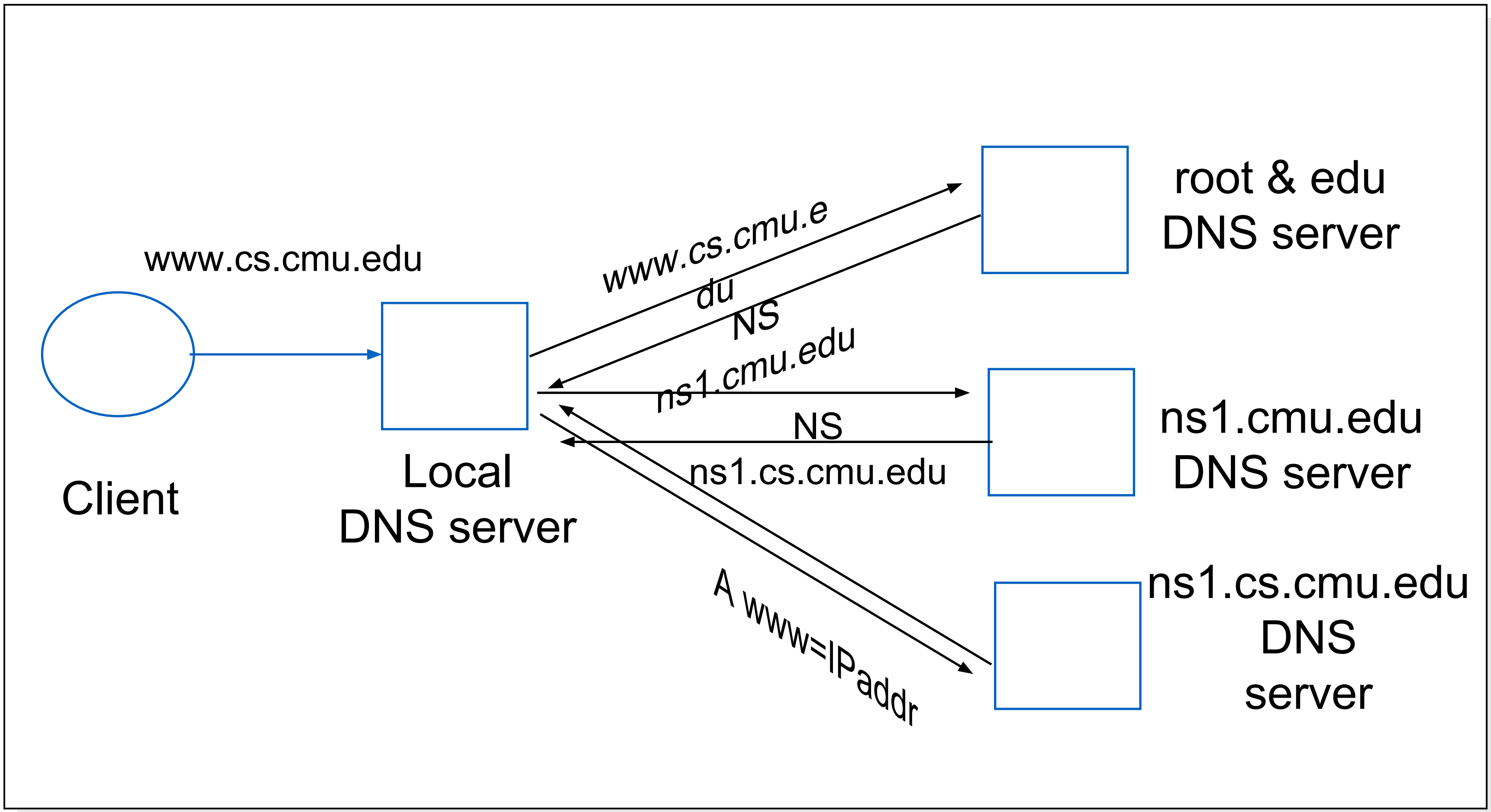
UDP used for queries

- RPC semantic of DNS?

Each host has a resolver

- Typically a library that applications can link to
- Local name servers hand-configured (e.g. /etc/resolv.conf)

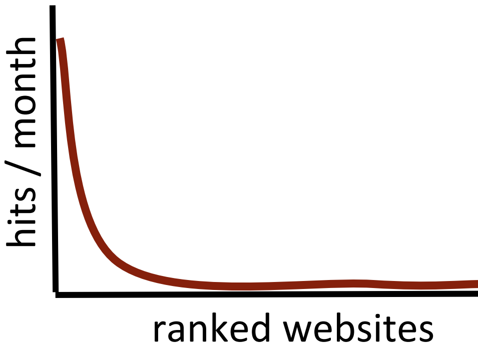
Typical Resolution



Workload and Caching

Are all servers/names likely to be equally popular?

- Why might this be a problem?
- How can we solve this problem?



DNS responses are cached

- Quick response for repeated translations
- Other queries may reuse some parts of lookup
    - NS records for domains

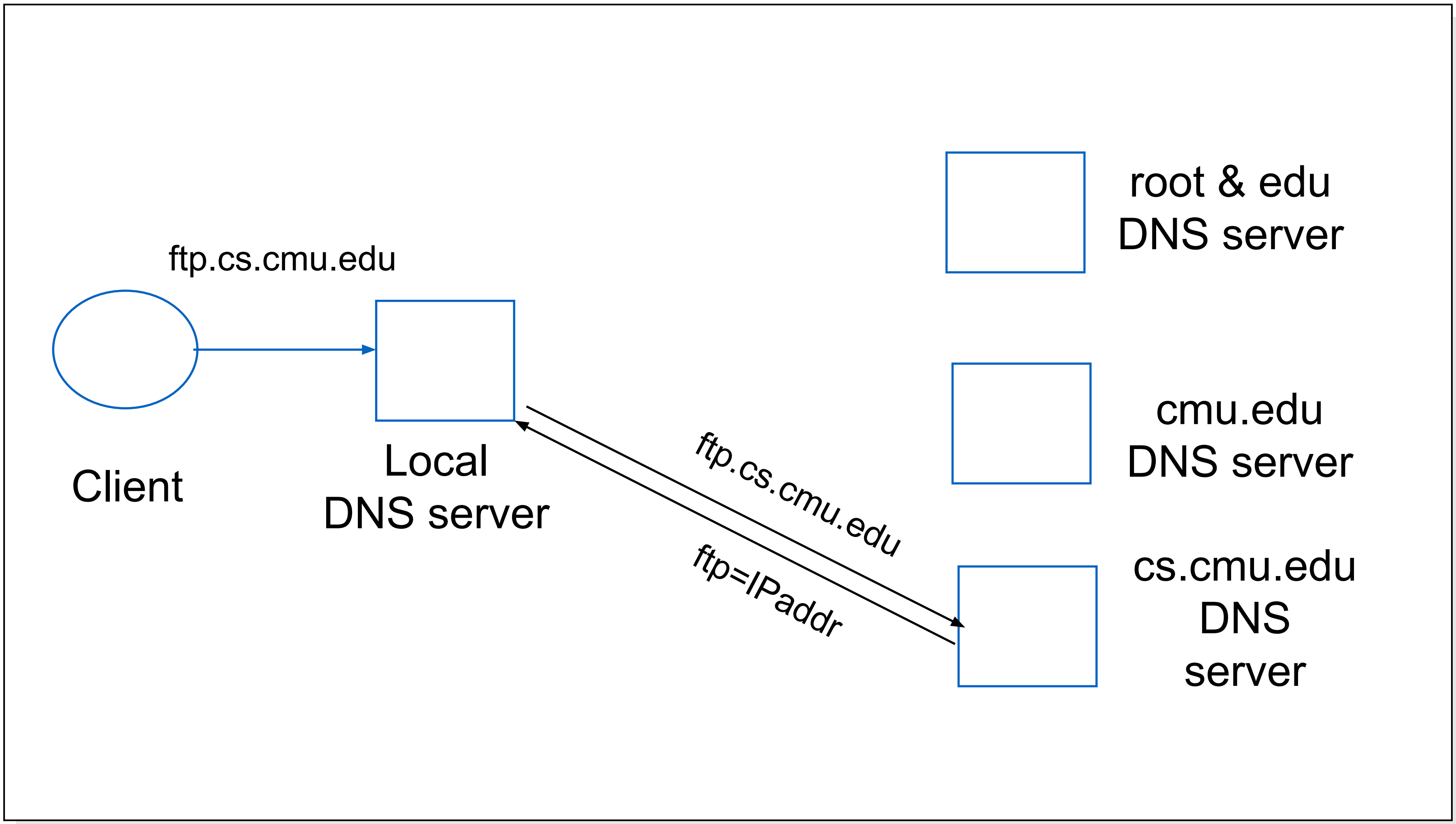
DNS negative queries are cached

- Don't have to repeat past mistakes
- E.g. misspellings, search strings in resolv.conf

Cached data periodically times out

- Lifetime (TTL) of data controlled by owner of data
- TTL passed with every record

Subsequent Lookup Example



Choosing the Time-To-Live

Common practices

Top-level NS records: very high TTL

- alleviate load on root

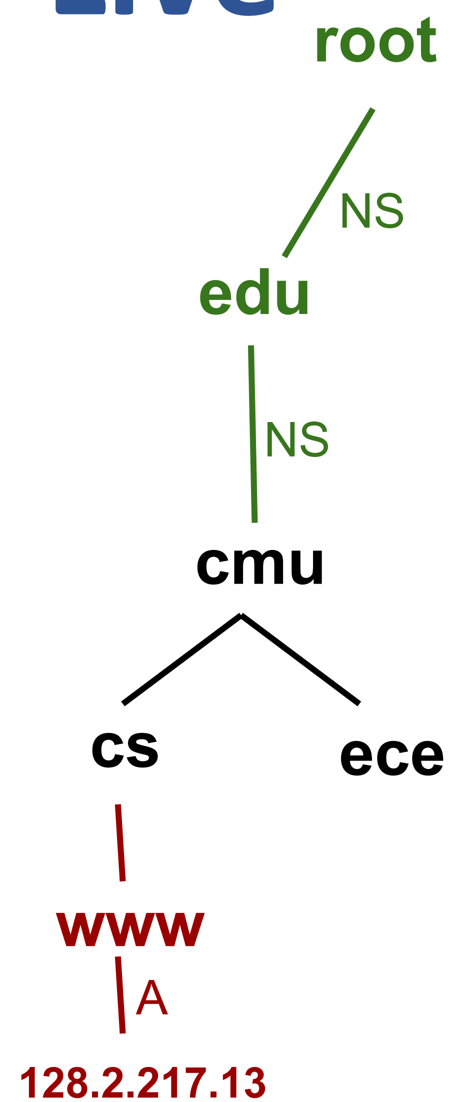
Intermediary NS records: high TTL

A records: small TTL (<7200s)

- consistency concerns

Some A records: tiny TTL (<30s)

- fault tolerance, load balancing



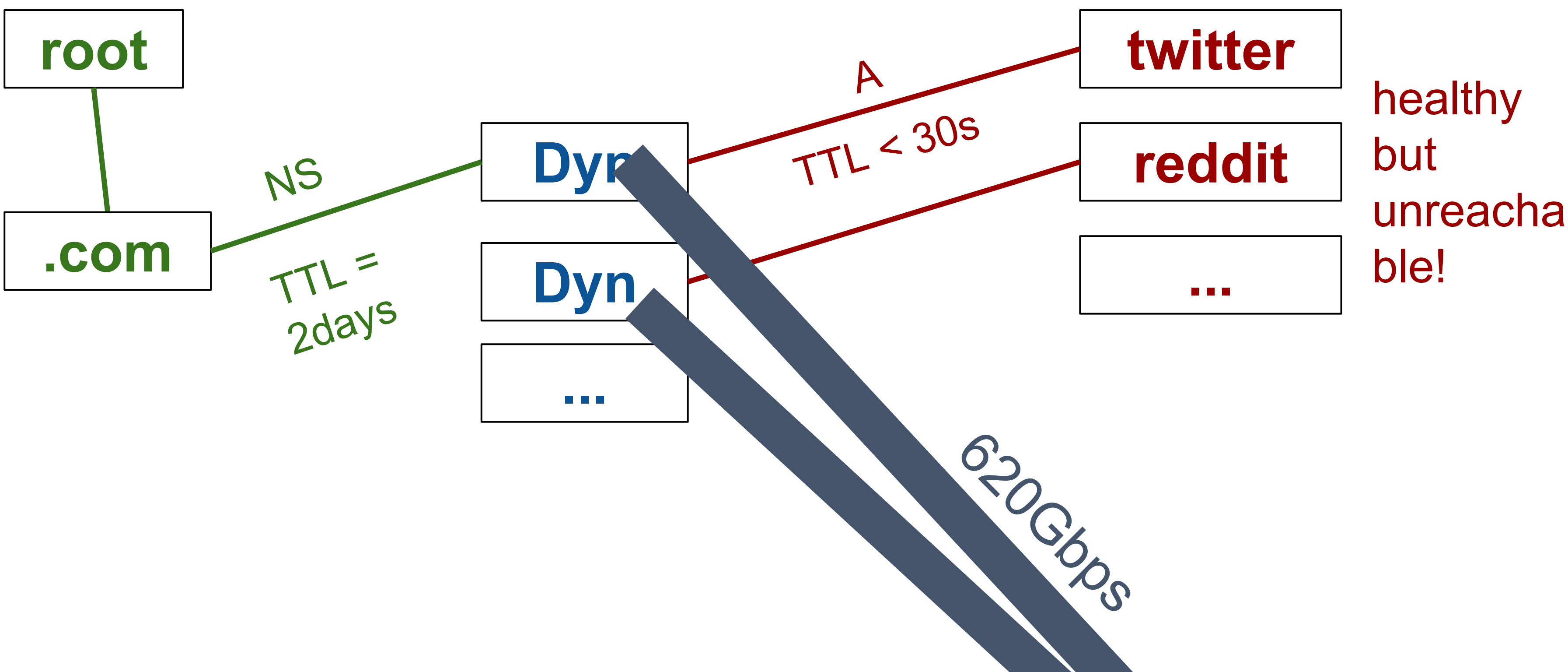
Do small TTLs give better availability and consistency?

What Happened on 10/21/2016?

- DDoS attack on Dyn
- Dyn provides naming service for Twitter, CNN, AirBnB, Spotify, Reddit, ...

- Why didn't DNS defense mechanisms work in this case?
- Let's take a look at the DNS records…

DNS at time ot Dyn Attack



Source: Mirai botnet (bad IoT devices)

- White-labeled DVR and IP camera electronics
- username: root and password: **xc3511**
- password hardcoded into the device firmware

Solutions?

Main culprit: no ideal TTL!

Could lower TTLs on NS records

- Redirect traffic faster to another DNS service
- Cost: increased load

Is trust in DNS consistency mechanism (TTL) overrated?

Dyn customers

- Going to backup DNS providers
- Signing up with alternatives after the attacks (PayPal, Amazon, etc)

DNS (Summary)

- Motivations → large distributed database
  - Scalability
  - Independent update
  - Robustness
- Hierarchical database structure
  - Zones
  - How is a lookup done
- Caching and consistency in practice
- What are the steps to creating and securing your own domain?

Topics Today

1) Naming at Internet Scale

    DNS - one of the world's largest databases

    DNS Architecture

    Robustness and Security Implications

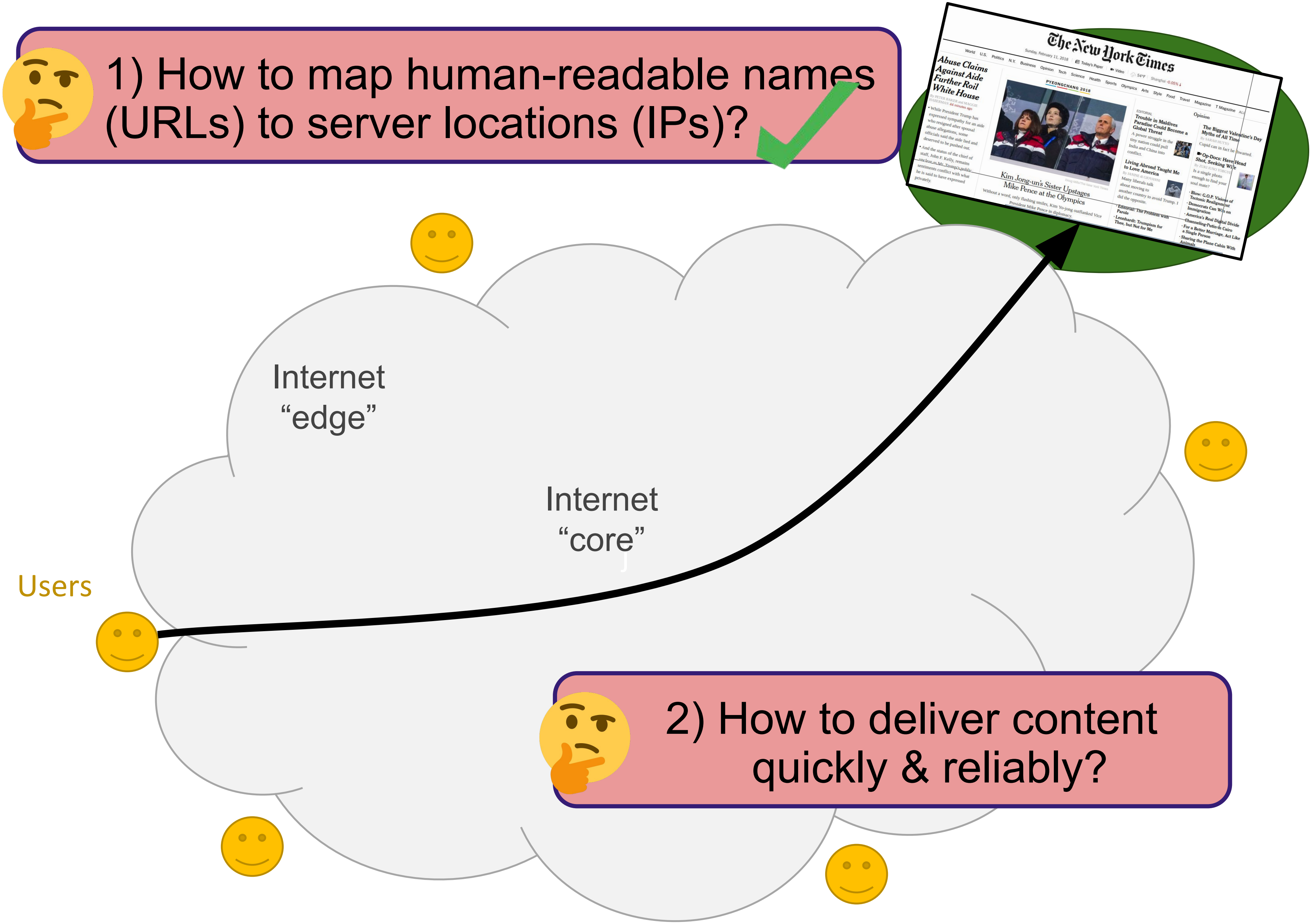
2) Content Distribution at Internet Scale

    CDNs - some of the world's largest distributed systems

    Design Decisions

    Consistent Hashing for Scaling and Load Balancing

1) How to map human-readable names (URLs) to server locations (IPs)?



2) How to deliver content quickly & reliably?

Typical Web Workload

- Many (typically small) objects per page
- File sizes are heavy-tailed
- Embedded references

Lots of objects & TCP
- 3-way handshake
- Lots of slow starts
- Even worse: TLS

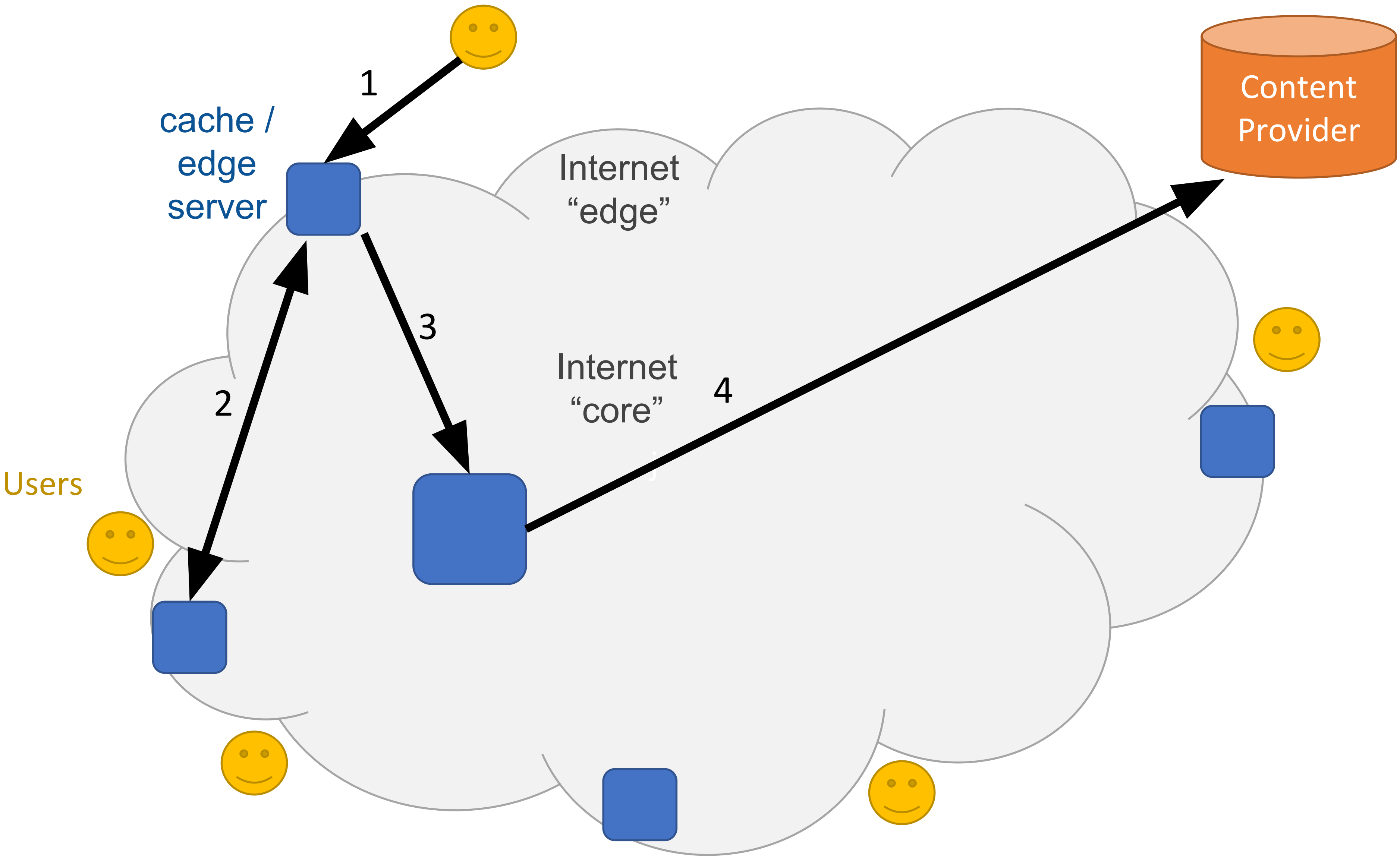
Why does this matter for performance?

- Content Delivery Network (CDNs)
  - The world's largest distributed caching systems
  - Key for Internet performance
  - Explosive growth

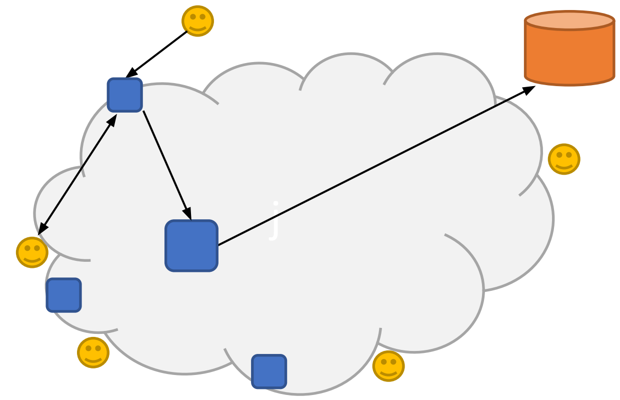
Technique to reduce latency in a DS?

CDNs will carry **71% of Internet traffic** in 2021, up from 52% in 2016. Source: CISCO Visual Networking Index 2016-2021. Sept 15, 2017.

A Typical CDN

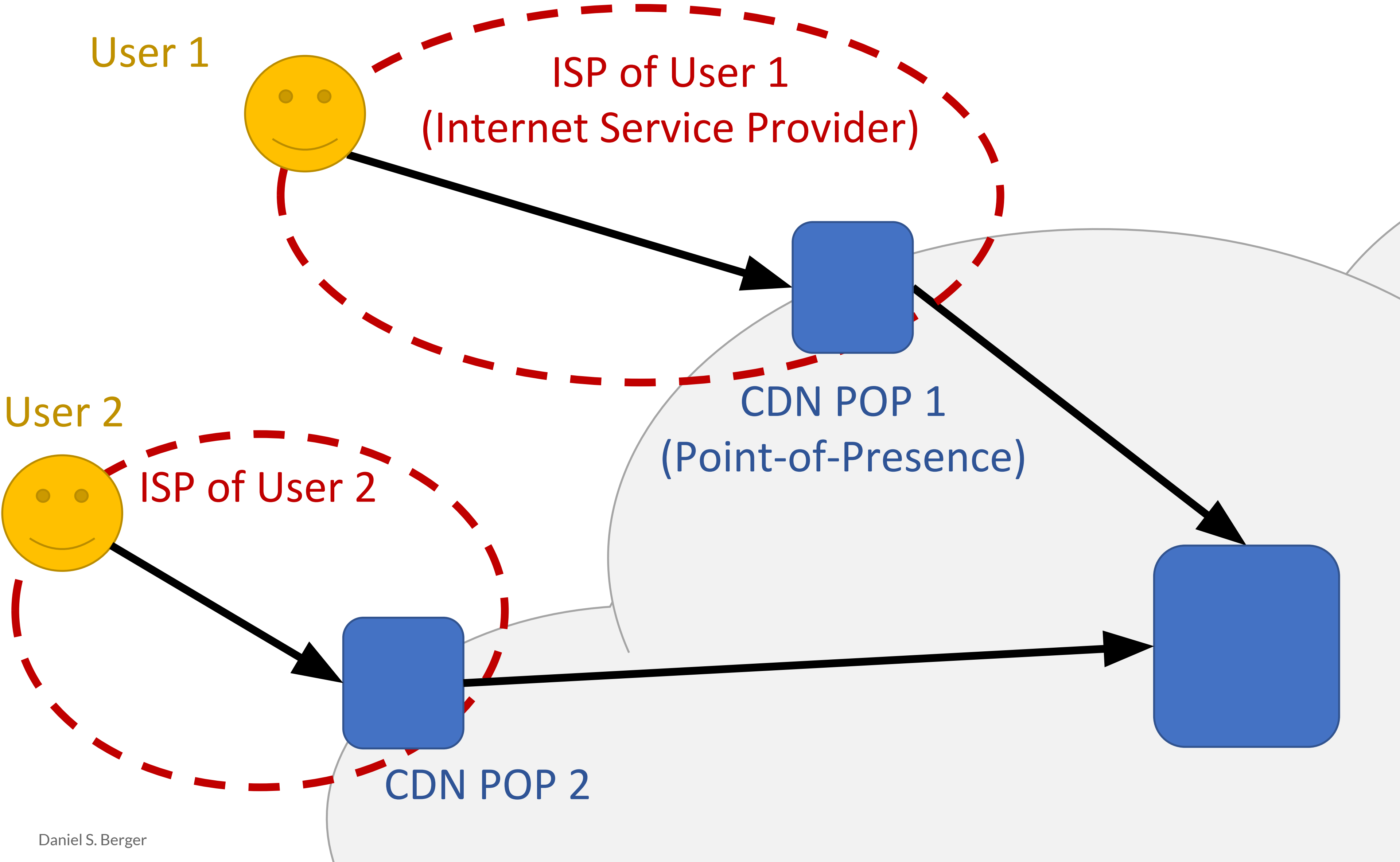


CDN Design Decisions



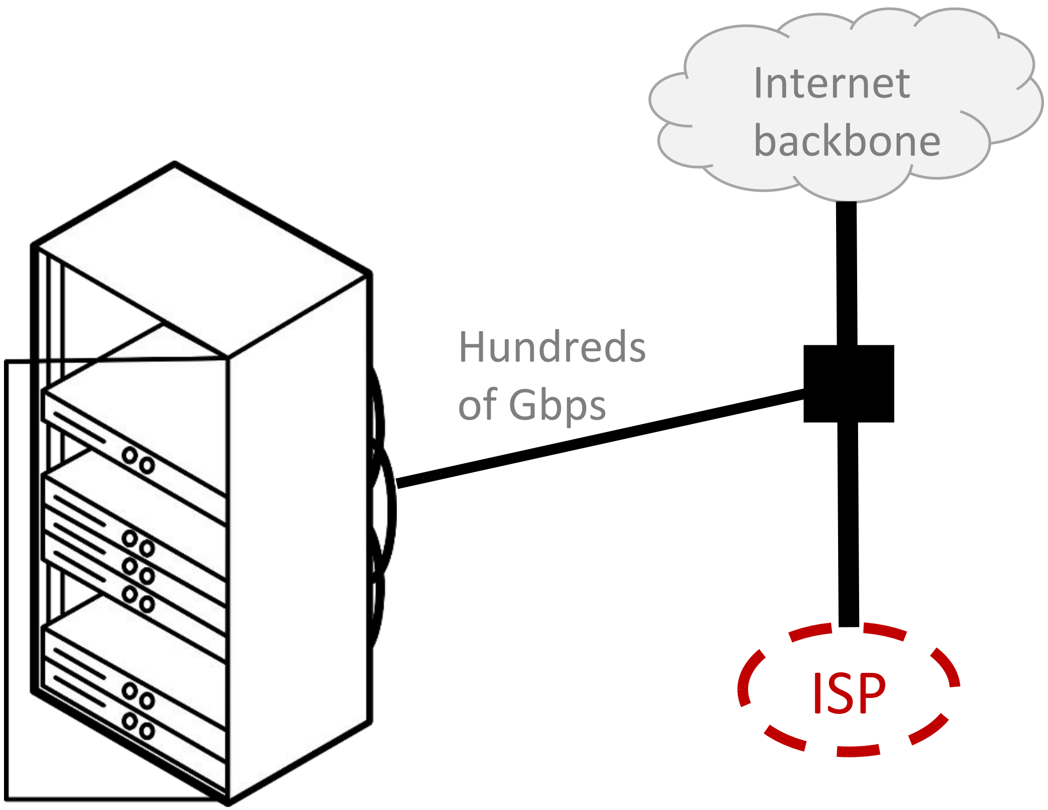
- Where to replicate content
- How to replicate content
- How to find content and how to direct clients towards a CDN PoP
- How to choose a CDN server within a PoP, and how to deal with failures
- How to propagate updates (CDN cache consistency)

Where to Replicate Content

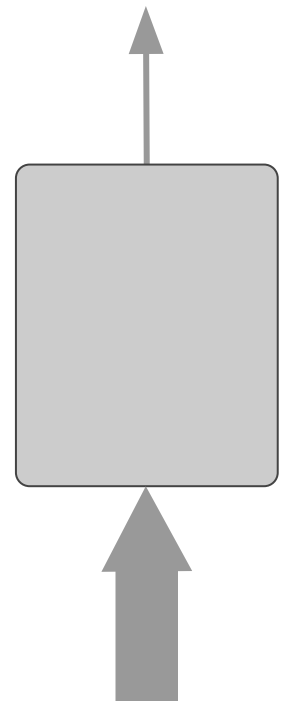


Where and How to Replicate

Rack(s) of edge servers



“Pull-based” edge servers



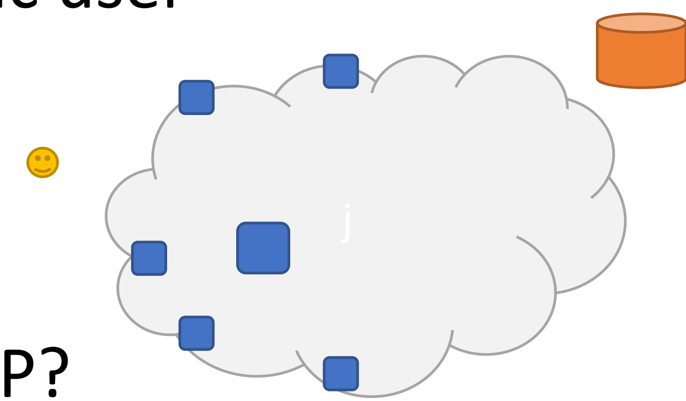
First check local cache

If cache miss, fetch from content provider

40 Gbps, 10k-100k reqs / sec

Directing Users to CDNs

- Which PoP?
  - Best “performance” for this specific user
    - Based on Geography? RTT?
    - Throughput? Load?



- How to direct user requests to the PoP?
  - As part of routing → anycast (= as part of IP routing)
  - As part of application → HTTP redirect
  - **As part of naming → DNS**

    (e.g., CNAME that is resolved via CDN’s name server)

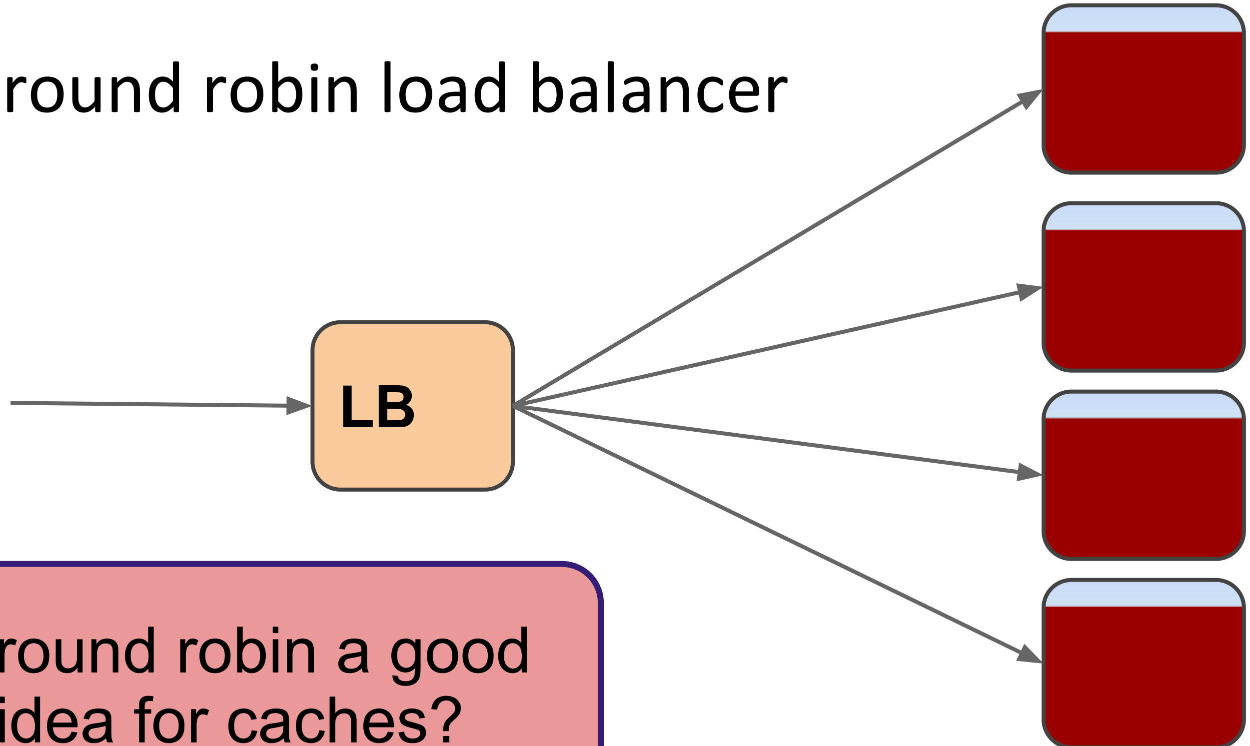
DNS-Based Client Routing

- Client does name lookup for service
- **CDN high-level name server** chooses appropriate regional PoP
  - Chooses “best” PoP for client
  - Return NS-record of low-level CDN name server
  - Large TTL (why?)
- **CDN low-level name server** chooses specific caching server within its PoP
  - Choose edge server that is likely to cache file, and is alive
  - Small TTL (why?)

How do we choose an edge server (that has file in cache and is alive)?

CDN Scaling and Load Balancing

Idea 1: round robin load balancer



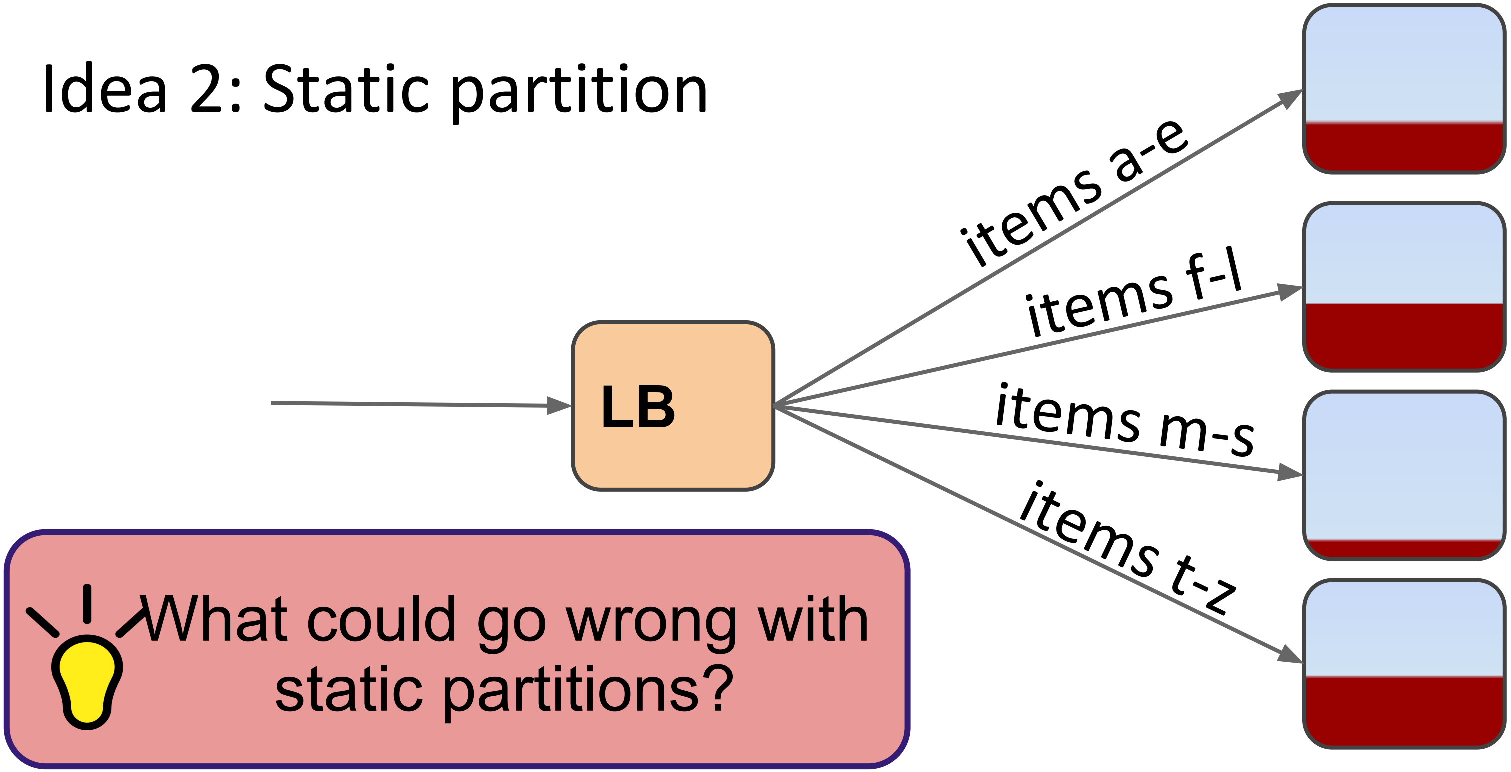
Is round robin a good idea for caches?

Consider an overall working set of size 16TB.

What is the working set at every cache with round robin?

Better CDN Load Balancer

Idea 2: Static partition



- If you used the server name: what if “cowpatties.com” had 1000000 pages, but “zebras.com” had only 10?
- Could fill up the bins as they arrive

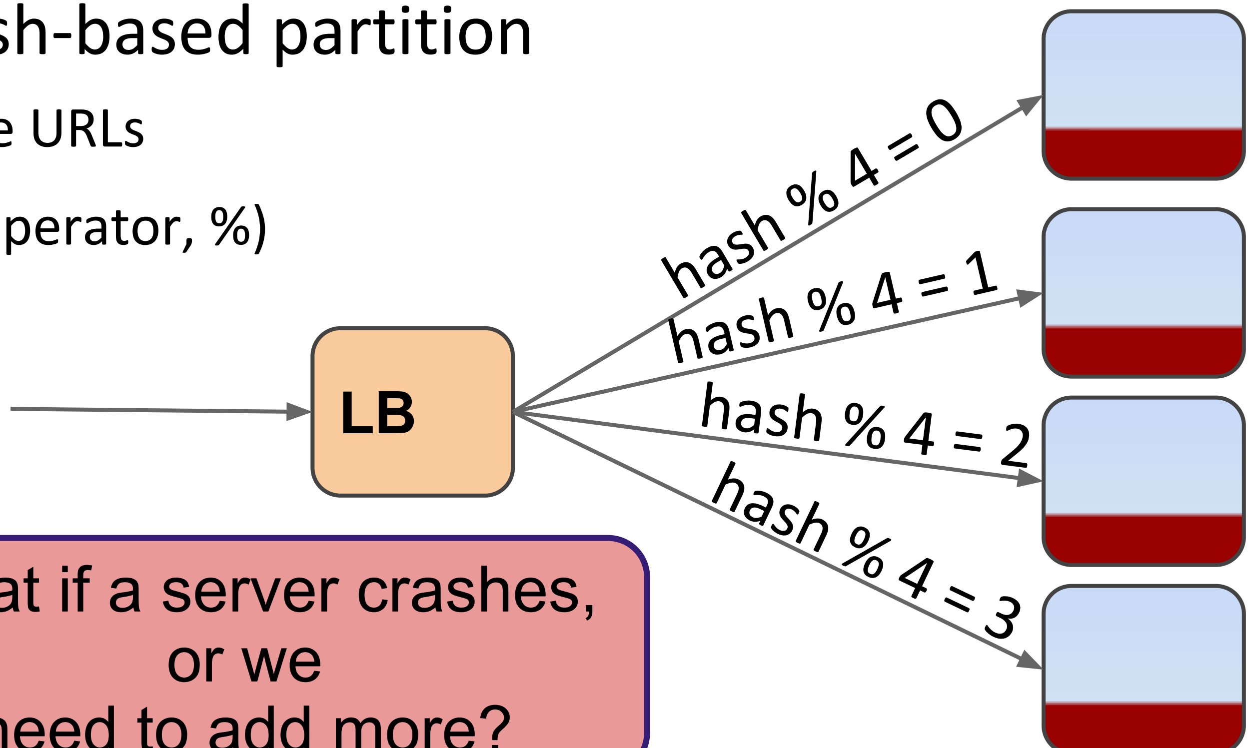
→ Requires tracking the location of **every** object at LB

Hash-Partitioned Load Balancer

Idea 3: Hash-based partition

(e.g., hash the URLs

use modulo operator, %)



What if a server crashes, or we need to add more?

- Problem 1: no data duplication → all servers need to be up!
- Problem 2: what if there are several LBs and they have different views of which servers are up/down?
- Problem 3: adding/removing servers is hard! Why?

Hash-Partitioning Problems

Idea 3: Hash-based partition (cntd)

Consider 90 documents

- Before: hash-partitioned to nodes 1..9
- Now: node 10 which was dead is alive again

How many documents are on the wrong server?

- Before: server = id%9 (for 9 servers)
- Now: server = id%10 (for 10 servers)

All objects with id > 9 need to move (slightly better with integer div)

☹ Disruption coefficient > ½

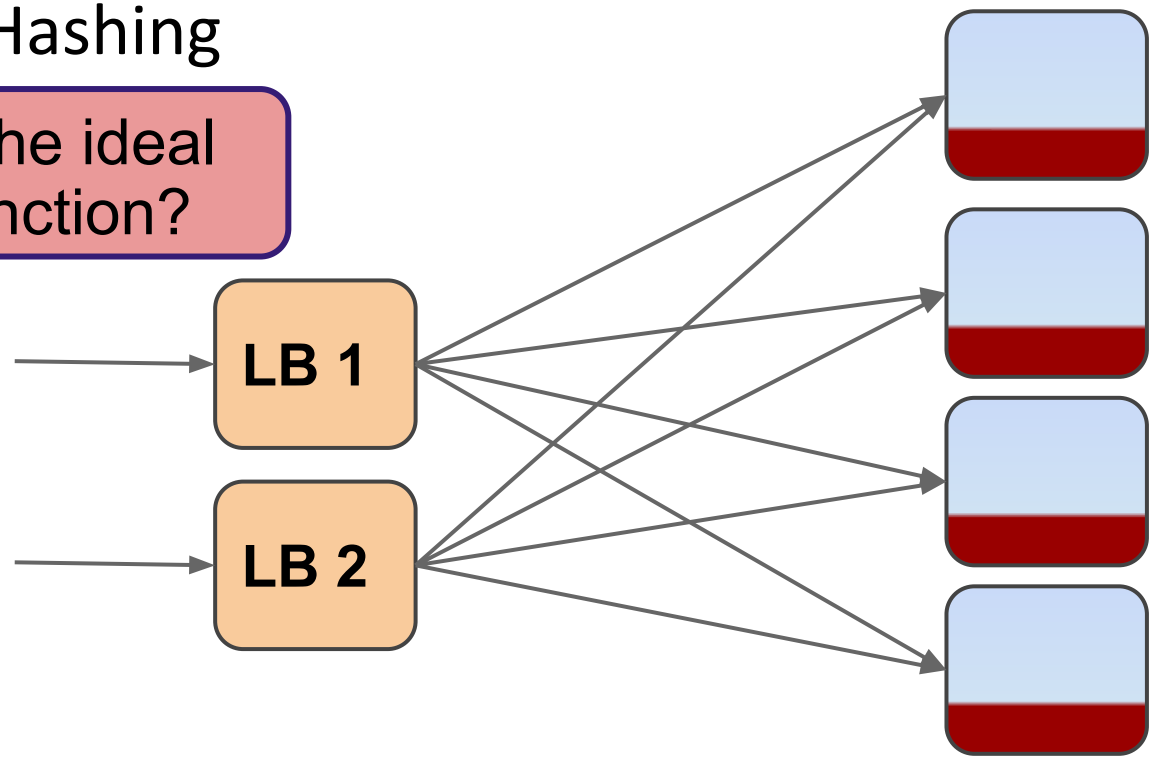
How do we fix hash-based partitioning?

Actual CDN Load Balancer

Idea 4: Consistent Hashing

Properties of the ideal CDN hash function?

"View" = subset of all servers that are visible to LB



Desired properties

Load: over all views, # of objects / server is small (and ~uniform)

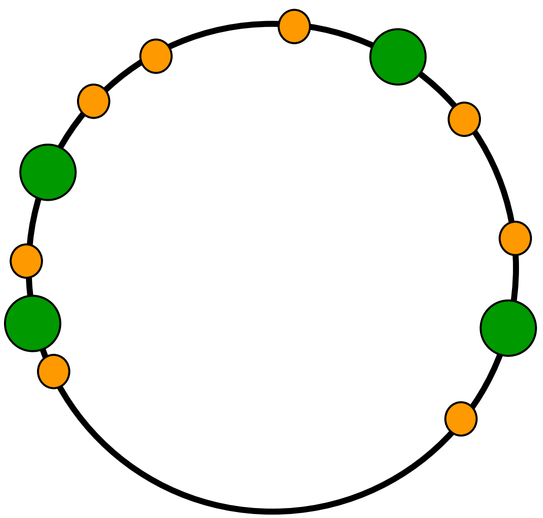
Spread: over all views, # of servers / obj is small (and ~uniform)

Smoothness: little impact when servers are added/removed

Implementing Consistent Hashing

- **Main idea:**
  - map both keys and nodes to the same (metric) identifier space

Ring is one option.



Consistent Hashing Identifiers

The consistent hash function assigns each node and key an *m*-bit identifier using SHA-1 as a base hash function.

Node identifier: SHA-1(IP address)

IP="198.10.10.1" $\xrightarrow{\text{SHA-1}}$ ID=123

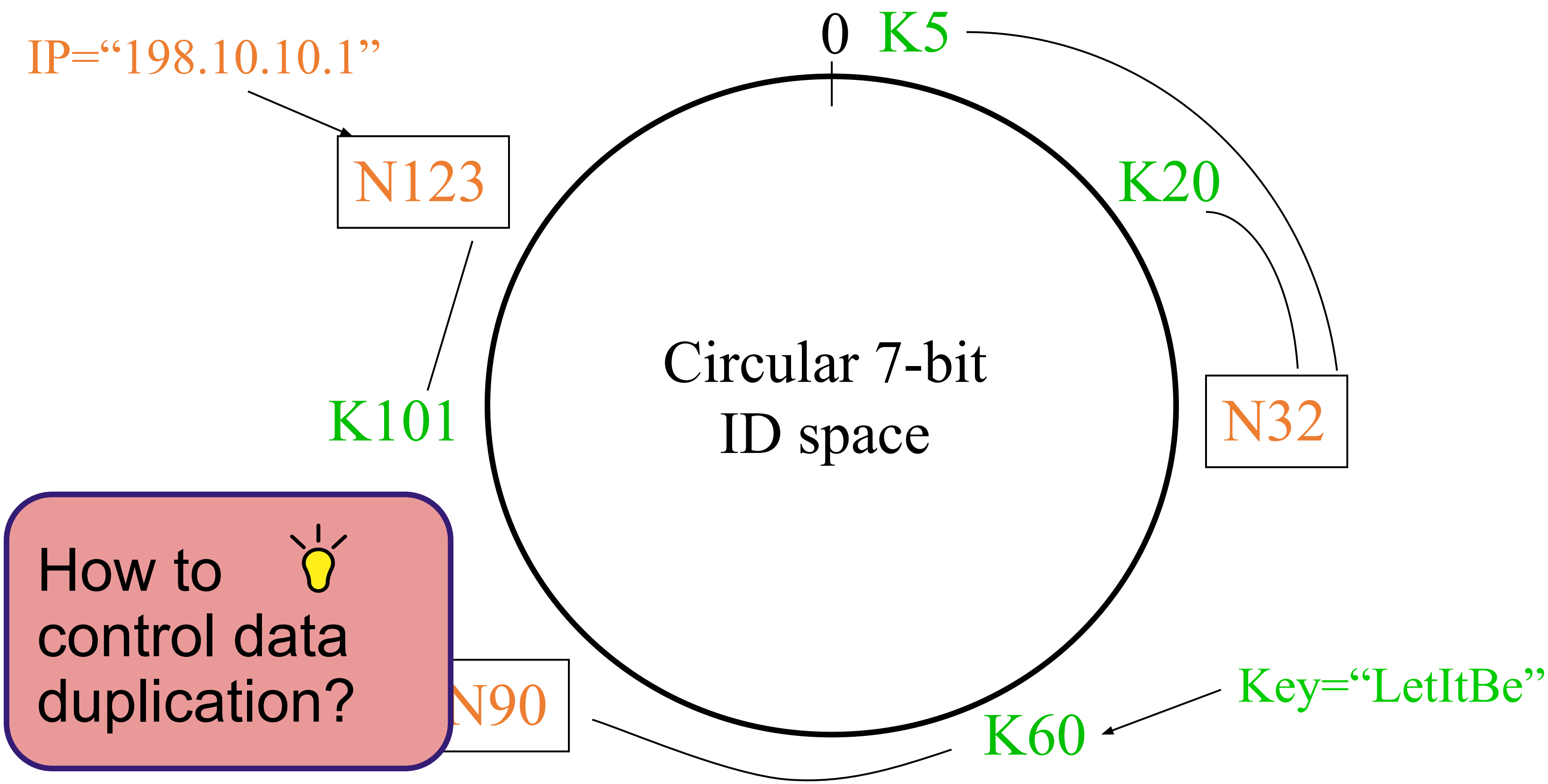
Key identifier: SHA-1(key)

key="LetItBe" $\xrightarrow{\text{SHA-1}}$ ID=60

How to map key ids to node ids?

Consistent Hashing Example

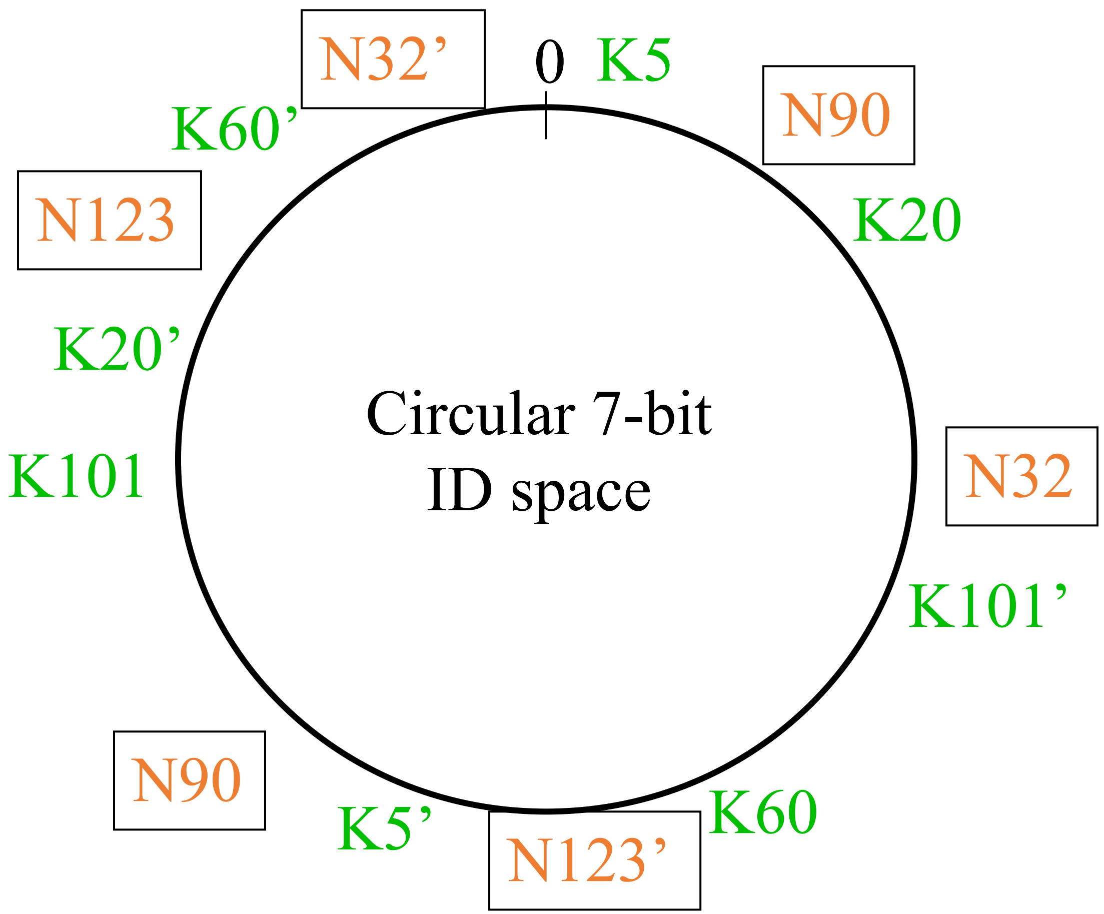
Rule: A key is stored at its successor: node with next higher or equal ID



How to control data duplication?

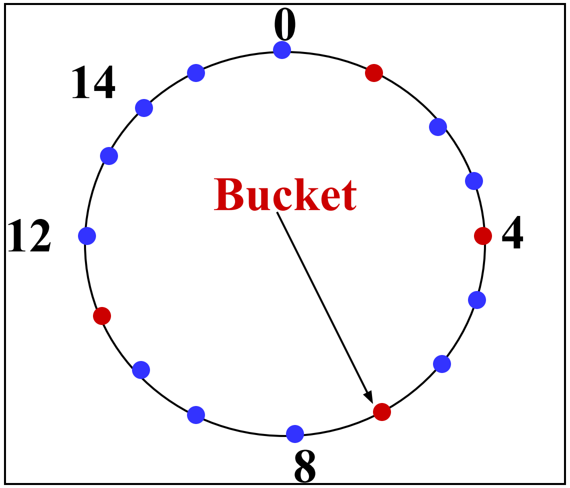
Consistent Hashing Example II

Add virtual nodes and keys to improve smoothness, load and spread.



Consistent Hash – Properties

Ring-based construction using hash of key and node with c different views.



- Load → no machine gets more than O(log c) times the average number of keys

- Spread → No key is stored in more than O(log c) caches.

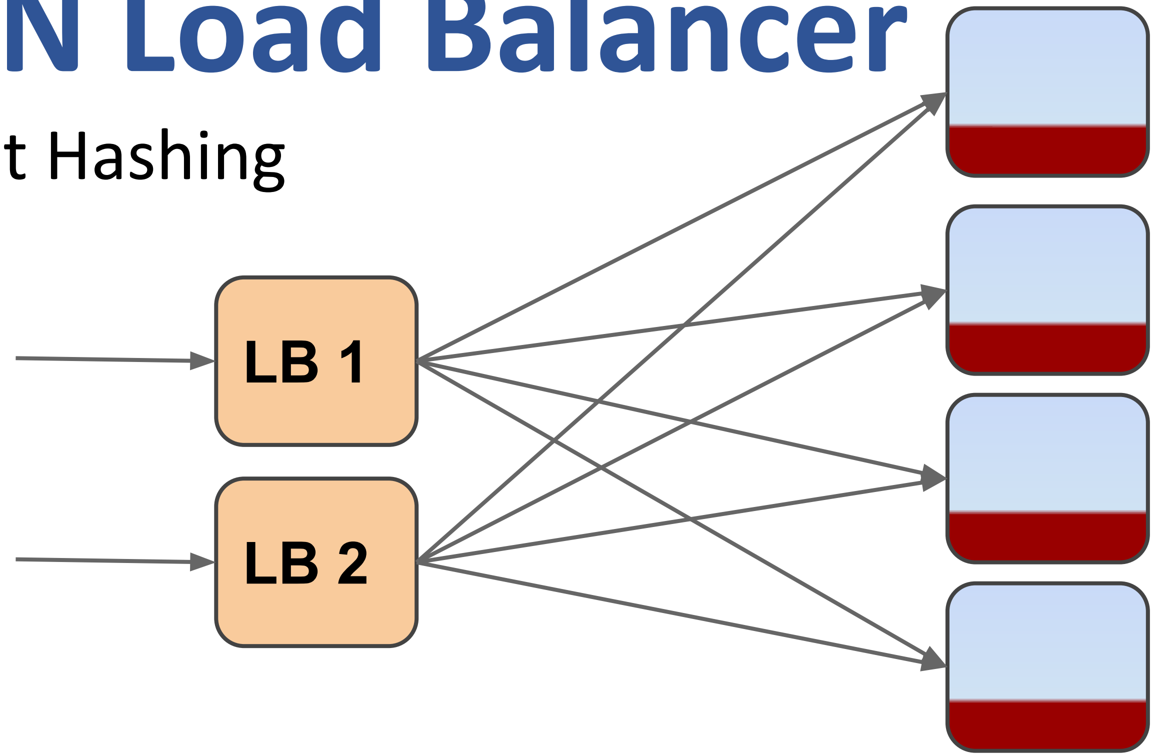
- Smoothness → addition of bucket does not cause much movement between existing buckets

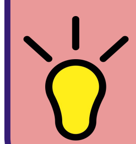
Back to CDNs...

(Consistent) hashing has many applications in DS

Actual CDN Load Balancer

Idea 4: Consistent Hashing





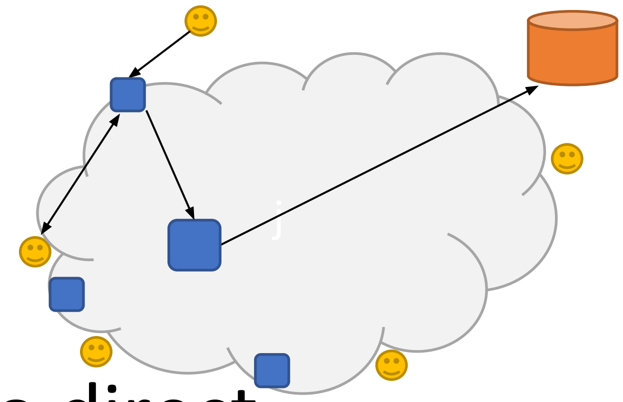
How to direct users to specific server?
(i.e., how to build the LB)

DNS-Based Client Routing

- Client does name lookup for service
- **CDN high-level name server** chooses appropriate regional PoP
  - Chooses “best” PoP for client
  - Return NS-record of low-level CDN name server
  - Large TTL (why?)
- **CDN low-level name server** chooses specific caching server within its PoP
  - Use **consistent hashing** to choose the edge server that has is responsible for this URL, and is alive
  - Small TTL (why?)

41

CDN Design Decisions

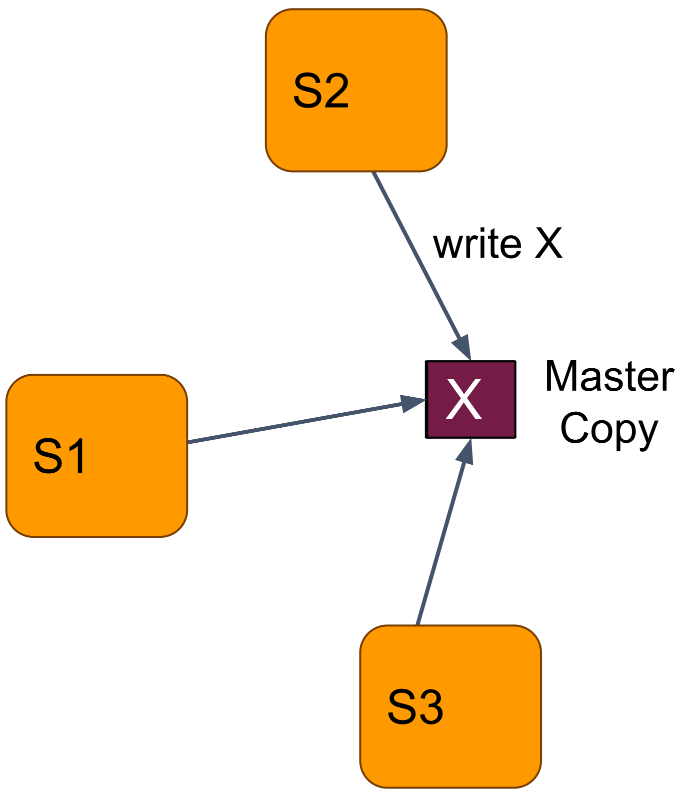


- ~~Where to replicate content~~
- ~~How to replicate content~~
- ~~How to find content and how to direct clients towards a CDN PoP~~
- ~~How to choose a CDN server within a PoP, and how to deal with failures~~
- How to propagate updates (CDN cache consistency)

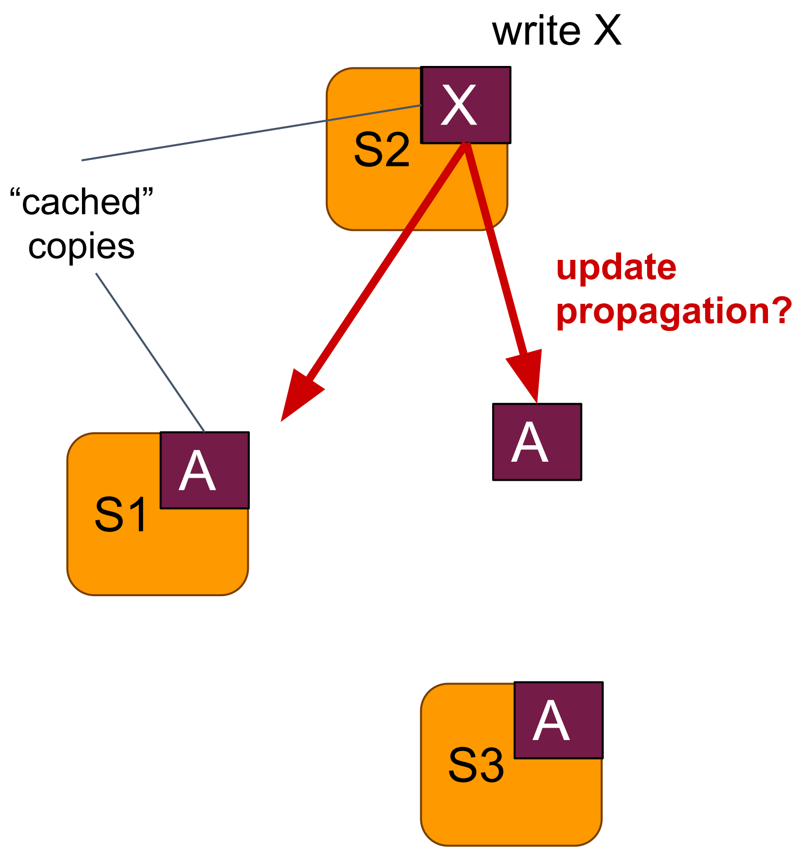
42

Cache Update Propagation Techniques

Ideal World: One-Copy Semantics

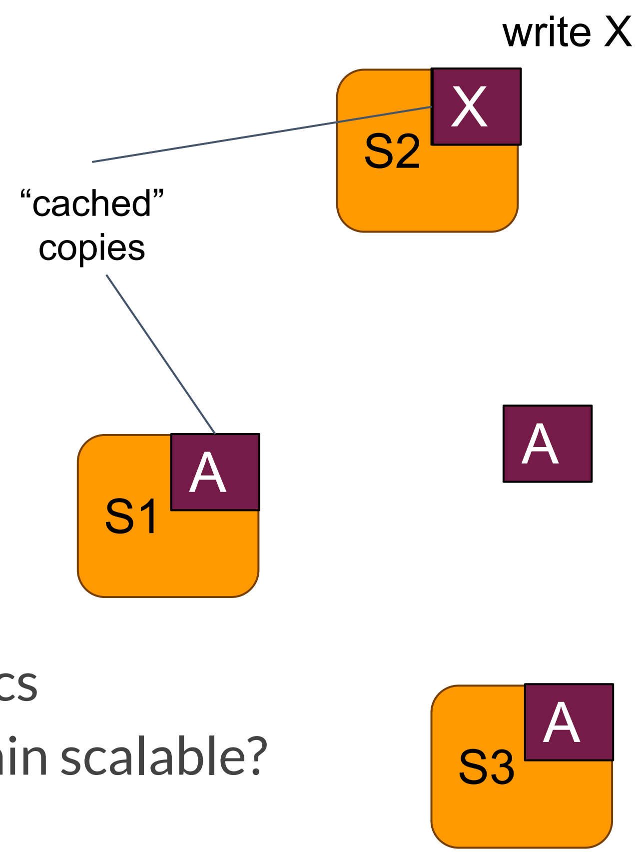


Caching Reality



Cache Update Propagation Techniques

1. Enforce Read-Only (Immutable Objects) ✓
2. **Broadcast Invalidations**
3. **Check on Use**
4. Callbacks ✓
5. **TTLs ("Faith-based Caching")**



All of these approximate one-copy-semantics

- how little can you give up, and still remain scalable?
- how complex is the implementation?

2. Broadcast Invalidations

Every potential caching site notified on every update

- No check to verify caching site actually contains object
- Notification includes specific object being invalidated
- Effectively broadcast of address being modified
- At each cache site, next reference to object will cause a miss

Usage: e.g., in CDNs

+

- Simple to implement
- No race conditions (with blocking writes)

+

- Wasted traffic if no readers
- Limited scalability (in blocking implementation)

3. Check On Use

Reader checks master copy before each use

- conditional fetch, if cache copy stale
- has to be done at coarse granularity (e.g. entire file)
- otherwise every read is slowed down excessively

Usage: e.g., AFS-1, HTTP (Cache-control: must-revalidate)

+

- Simple to implement
- No server state (no need to know caching node)

+

- Wasted traffic if no updates
- Very slow if high latency
- High load

5. TTLs ("Faith-based Caching")

Assume cached data is valid for a while

- Check after timer expires: Time-to-Live (TTL) field
- No communication during trust (TTL) period

Usage: e.g., CDNs, DNS, HTTP (Cache-control: max-age=30)

+

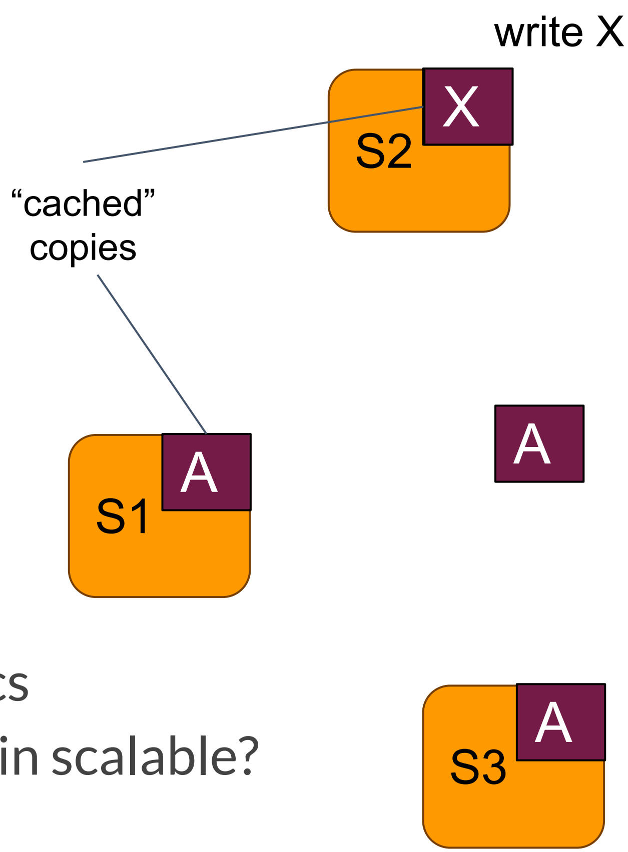
- Simple to implement
- No server state (no need to know caching node)

+

- Use visible inconsistency
- Less efficient than callback-based schemes

Cache Update Propagation Techniques

1. Enforce Read-Only (Immutable Objects) ✓
2. **Broadcast Invalidations**
3. **Check on Use**
4. Callbacks ✓
5. **TTLs ("Faith-based Caching")**



All of these approximate one-copy-semantics

- how little can you give up, and still remain scalable?
- how complex is the implementation?

CDN Update Propagation

Static Web Objects ("1st-gen CDNs" from 1998)

- Images & Photos, static websites, CSS, JS, ...
- Consistency via TTL (set by content owner)

Dynamic Content ("2nd-gen CDNs" from 2010)

- Support for dynamic web content at edge
- Broadcast invalidation "purge" objects 10ms

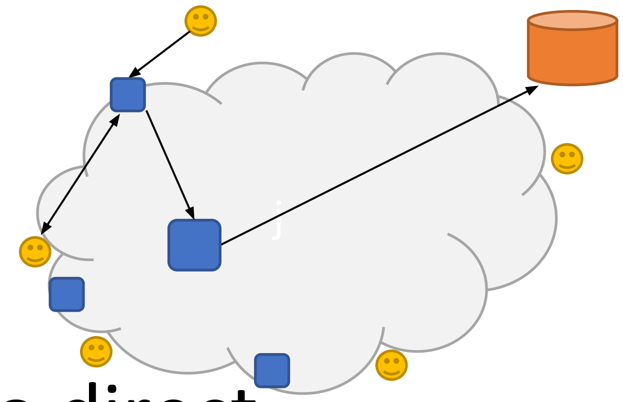
Edge Applications (only partial adoption)

- Applications run on edge servers
- Paxos-based data replication (at Akamai)

Bypass caches

- Forward data to data center, TCP/TLS at edge

CDN Design Decisions



- ~~Where to replicate content~~
- ~~How to replicate content~~
- ~~How to find content and how to direct clients towards a CDN PoP~~
- ~~How to choose a CDN server within a PoP, and how to deal with failures~~
- ~~How to propagate updates (CDN cache consistency)~~

Does every CDN look like ours?

So far, we've discussed Akamai

- Akamai is one of the world's largest CDNs
  - Evolved out of MIT research on consistent hashing
  - Serves 15-30% of all Internet traffic
  - 170K++ servers deployed worldwide
- But there are many more: CloudFront, CloudFlare, Fastly, ChinaNet, Edgecast, Limelight, Lvl3, GCD, ..
- Current developments:
  - Automation in performance tuning
  - Large content providers deploy their own CDNs
  - Many open problems (performance and security)

51

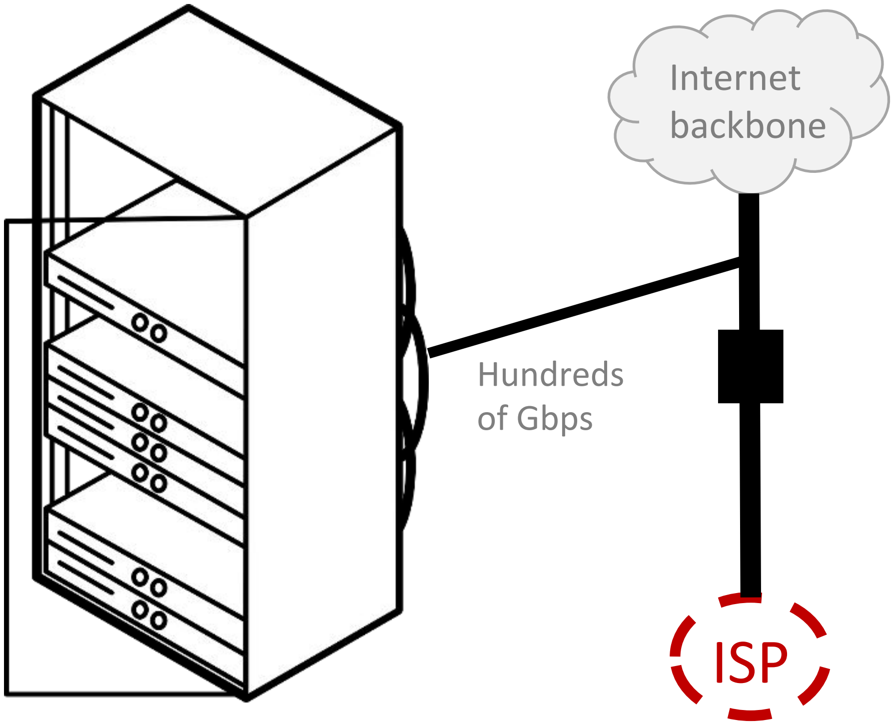
Summary on CDNs

- Across wide-area Internet: **caching is the only way** to improve latency
- CDNs move data closer to user
- CDNs balance load and fault tolerance
- Many design decisions, including cache consistency
- Use consistent hashes and many other DS techniques

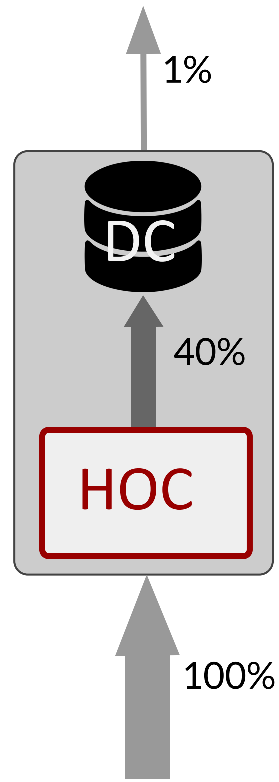
What if load is larger than CDN can handle?

More Detailed PoP View

Rack(s) of cache servers



Details of a cache server



Typical server:

- Disk Cache

  8 x 1TB SSD

- Hot Object Cache

  64GB RAM

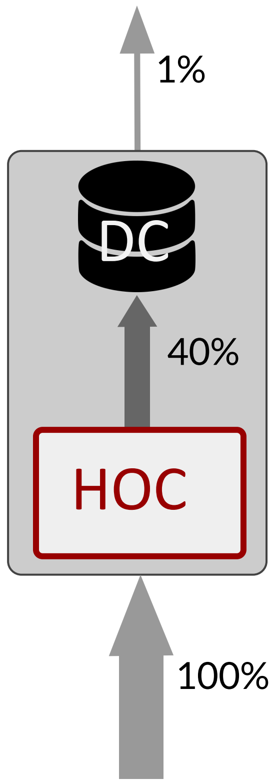
40 Gbps, 10k-100k reqs / sec

Caching Challenges I

- DC can't serve traffic at 40 Gbps and up
  - Write-intensive workload
  - Mostly random I/Os

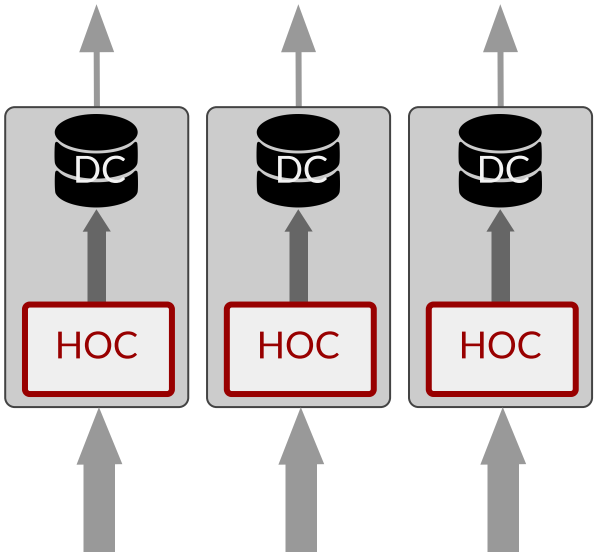
- HOC needs to serve majority of requests
  - But HOC is small
  - Cache management needs to deal with variability
  - No time for decisions

    10 micro seconds / req



Caching Challenges II

- Each POP has many CDN servers
- Currently use consistent hashing
- But different traffic types don't mix
  - Live streaming events: high temporal variability
  - Software downloads (think iOS release): dominate everything for short amount of time, very large files
  - Gaming/interactive web apps: very small files, latency sensitive



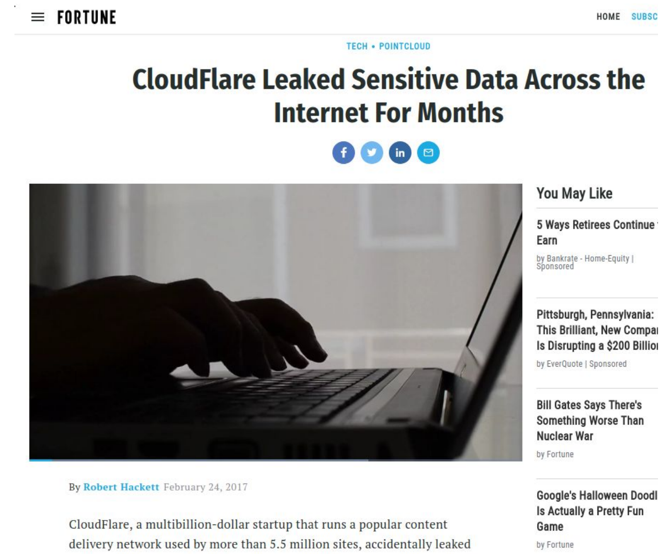
- We need automated classification and request routing

Caching Challenges III

- CDN servers do more than just caching
- HTTPs termination, Image rescaling,…
- 2017/2: CloudFlare information leak
- SW bug exposed cookies, auth codes
- How do we build safe and robust

  CDN server software?
  - Very critical user data

    (passwords, visitor stats, etc.)
  - High-performance low latency
  - Very specialized code bases, in-house code development



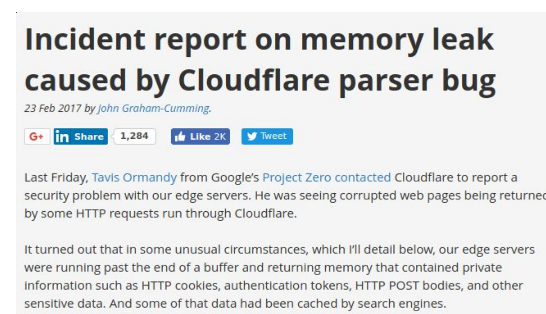
FORTUNE

TECH • POINTCLOUD

CloudFlare Leaked Sensitive Data Across the Internet For Months

By Robert Hackett February 24, 2017

CloudFlare, a multibillion-dollar startup that runs a popular content delivery network used by more than 5.5 million sites, accidentally leaked



Incident report on memory leak caused by Cloudflare parser bug

23 Feb 2017 by John Graham-Cumming.

Last Friday, Tavis Ormandy from Google's Project Zero contacted Cloudflare to report a security problem with our edge servers. He was seeing corrupted web pages being returned by some HTTP requests run through Cloudflare.

It turned out that in some unusual circumstances, which I'll detail below, our edge servers were running past the end of a buffer and returning memory that contained private information such as HTTP cookies, authentication tokens, HTTP POST bodies, and other sensitive data. And some of that data had been cached by search engines.