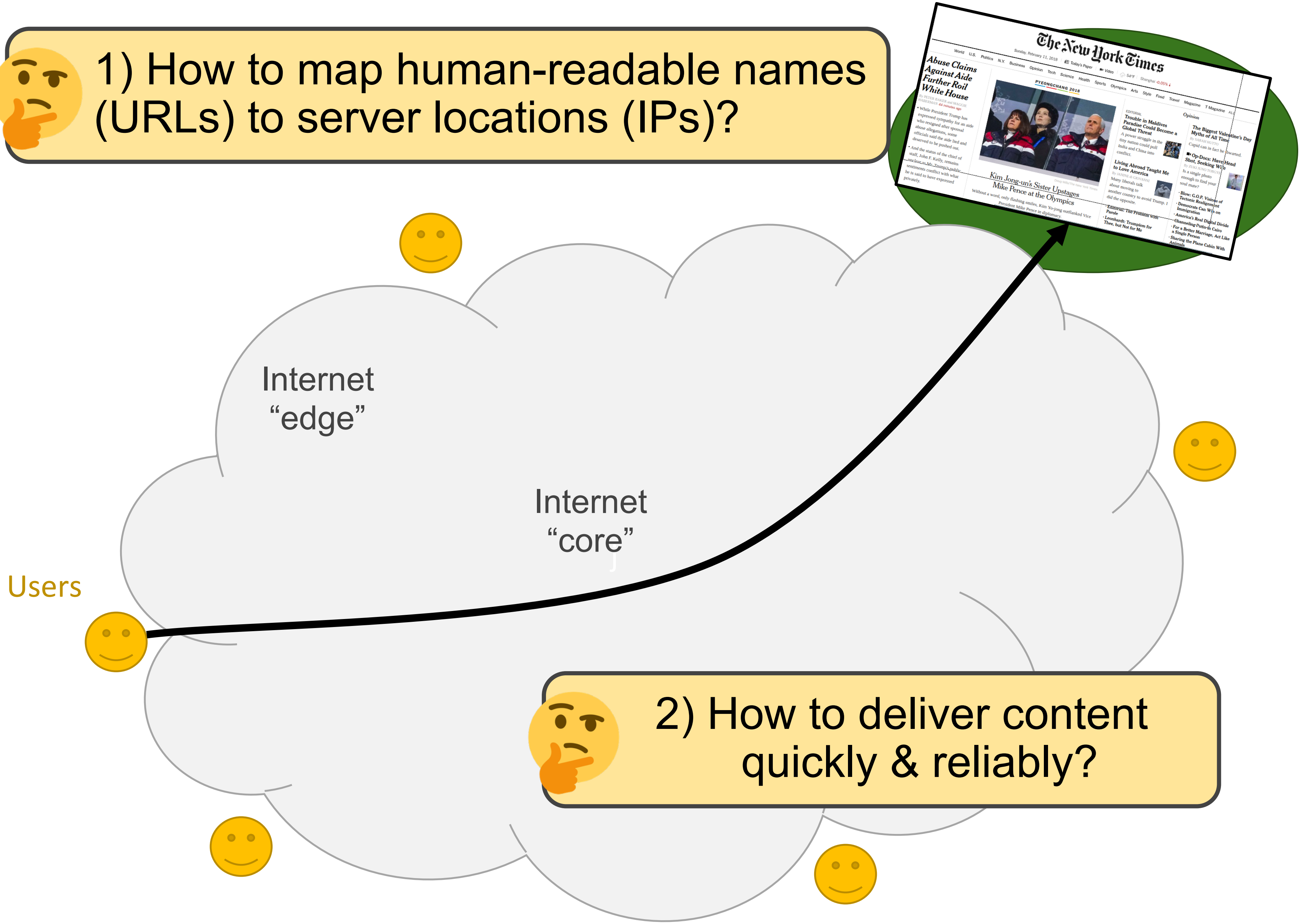
15-440/640
Distributed Systems

Internet Content Delivery

The Domain Name System
& Content Delivery Networks

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

2. Content Distribution at Internet Scale

    CDNs - some of the world's largest distributed systems

    Design Decisions

    Consistent Hashing for Scaling and Load Balancing

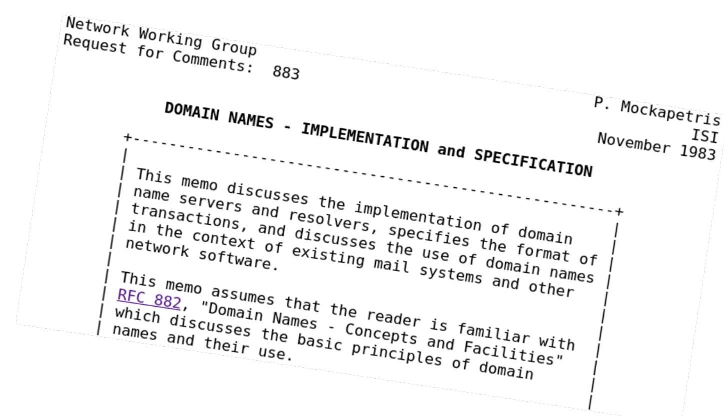
Internet Name Discovery

Challenges/Goals:

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

Domain Name System, 1984

DNS trades off consistency
for all these goals


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-RPC Format

RPC-queries to the DNS database with billions of resource records (RR)

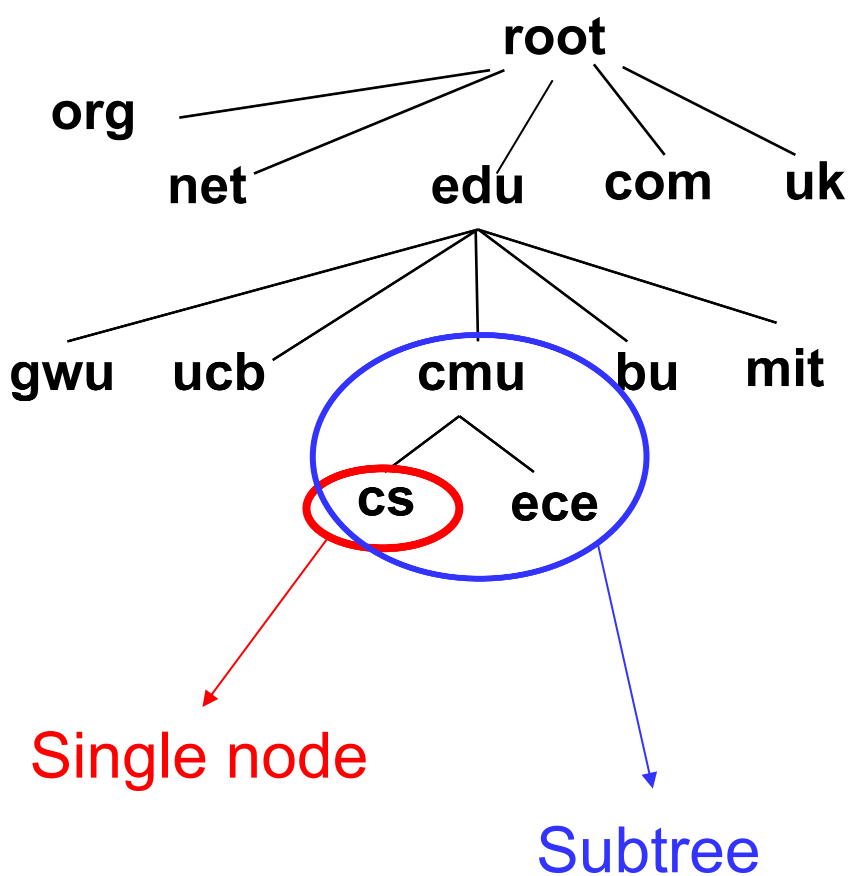
RR format: **(class, name, value, type, ttl)**

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**

The DNS Hierarchy

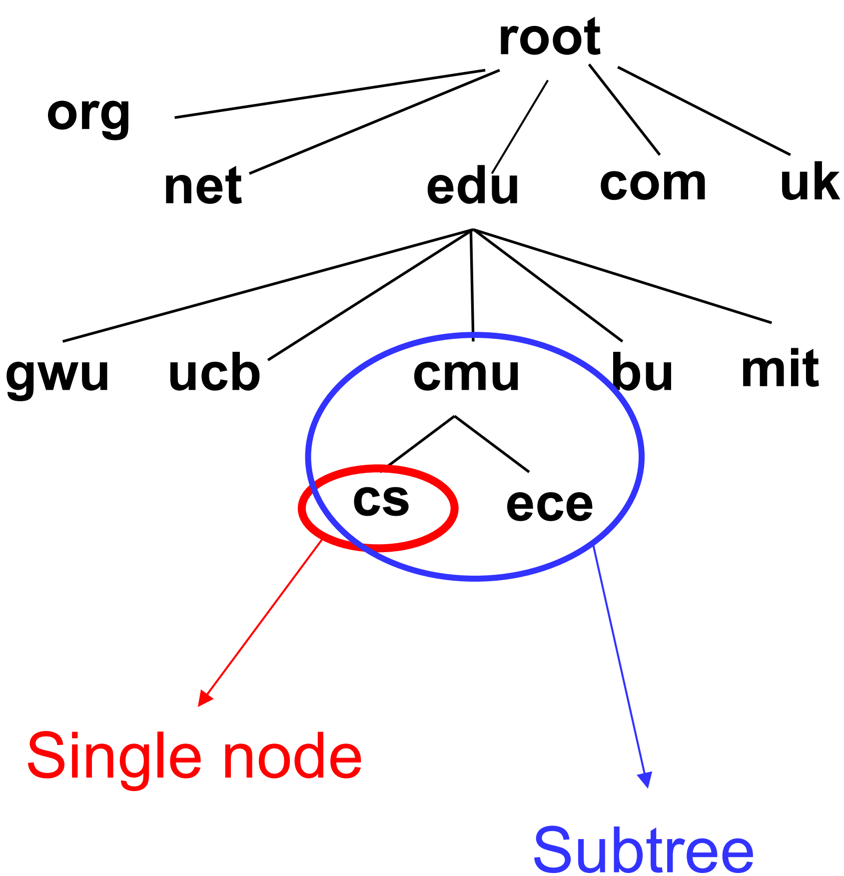


Each node in hierarchy stores information for names that end with same suffix

- Suffix = path up tree

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

The DNS Hierarchy



Zone

- distinct contiguous section of name space
  - E.g., Complete tree, single node or subtree
- Managed by a specific organization or administrator
- Has an associated set of name servers
  - Holds trusted, correct DNS records for that zone

DNS Design: Zone Delegation

Zones are created by delegating the administration for a part of the DNS namespace

- Records within zone stored in multiple redundant name servers (primary/secondary)
- Secondary 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

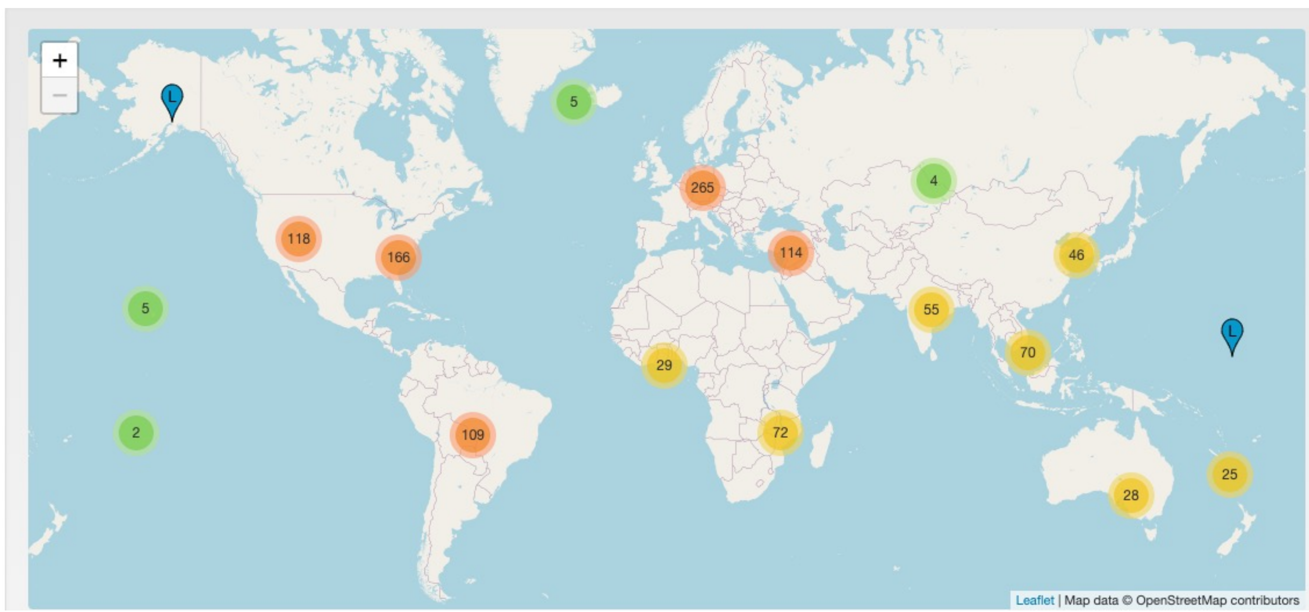
DNS: Root Name Servers

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
- www.root-servers.org



Architecture and Robustness

DNS servers are replicated

- Available if ≥1 replica up
- Load balance replicas

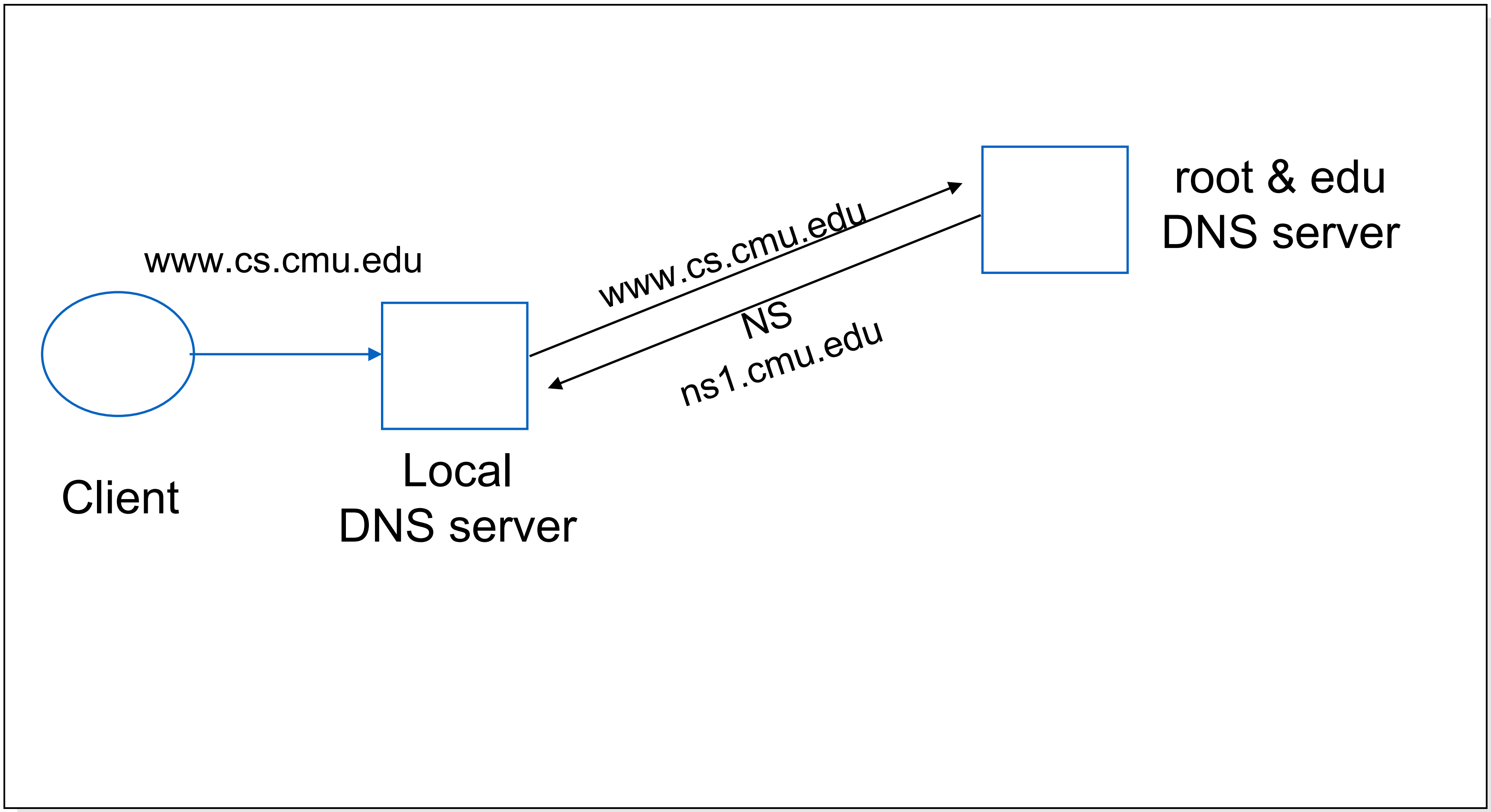
UDP used for queries

RPC semantic of DNS?

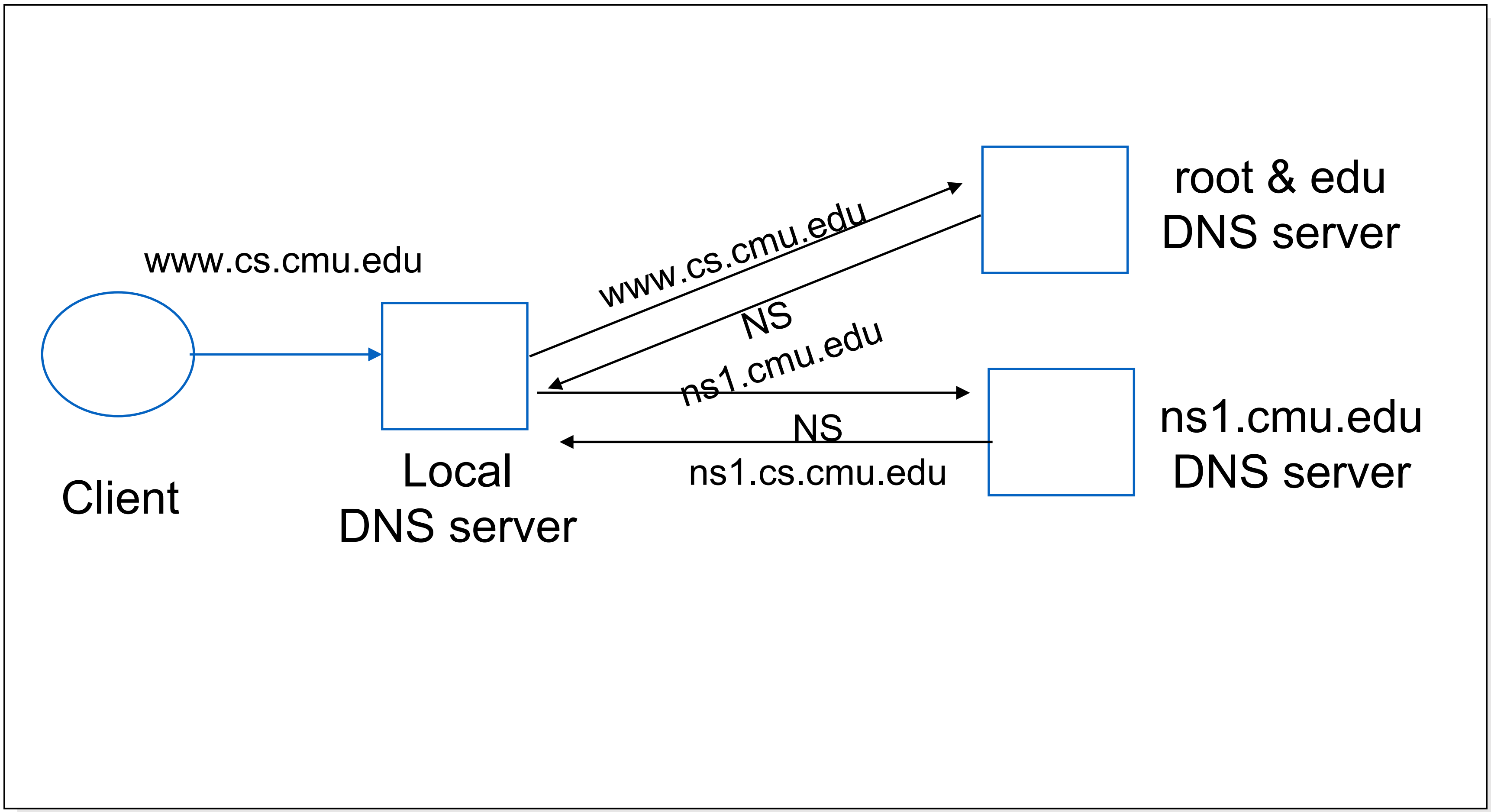
Each host has a resolver

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

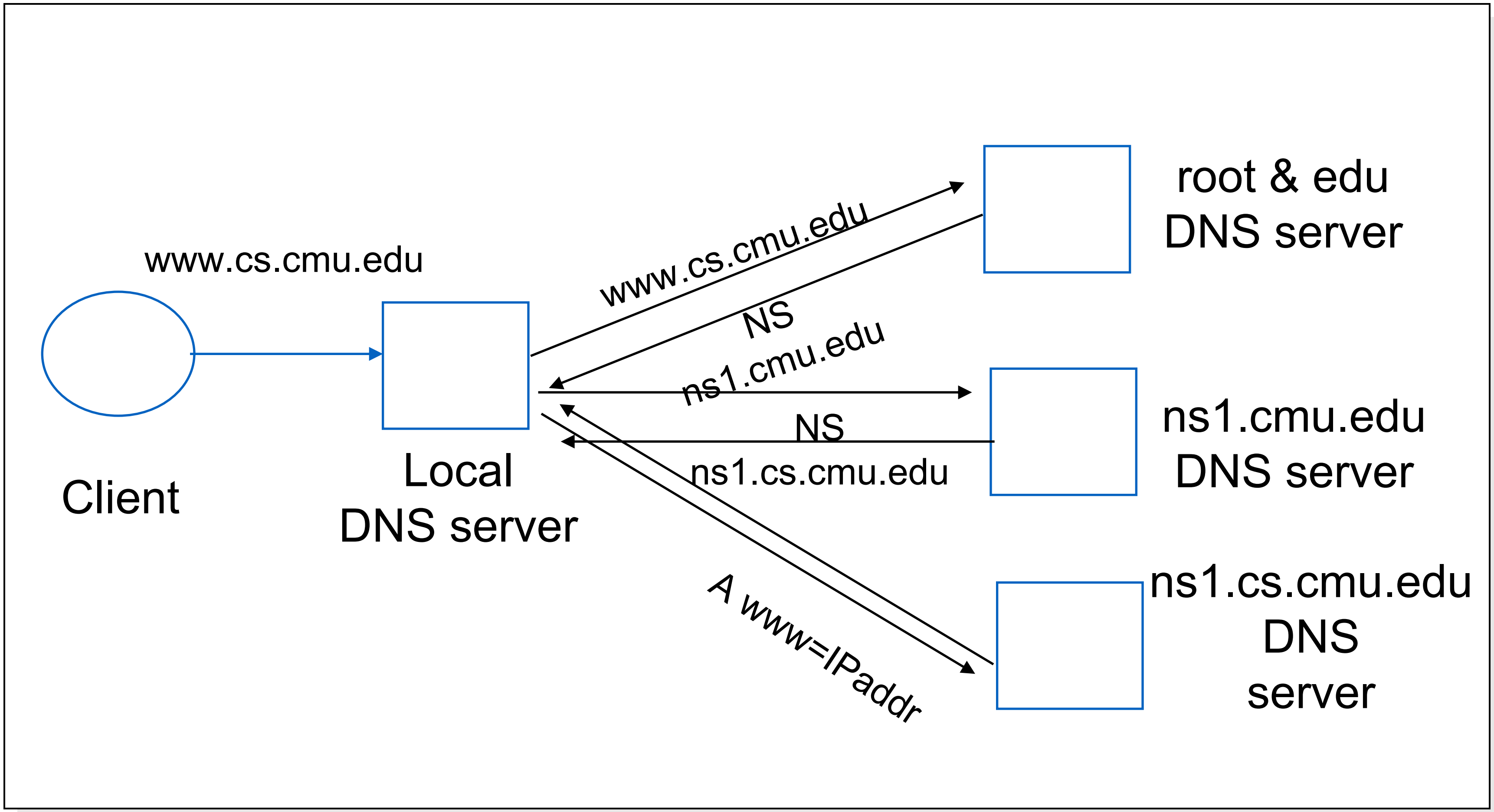
Typical Resolution



Typical Resolution



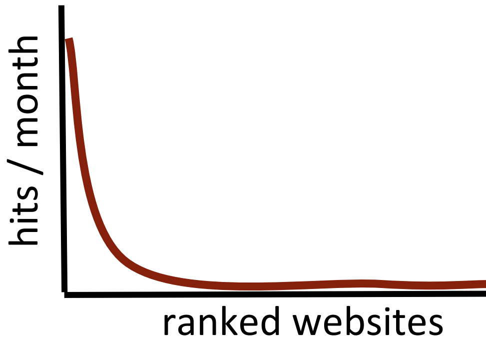
Typical Resolution



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 of data controlled by owner of data
- Time-to-live (TTL) passed with every record

Choosing the Time-To-Live

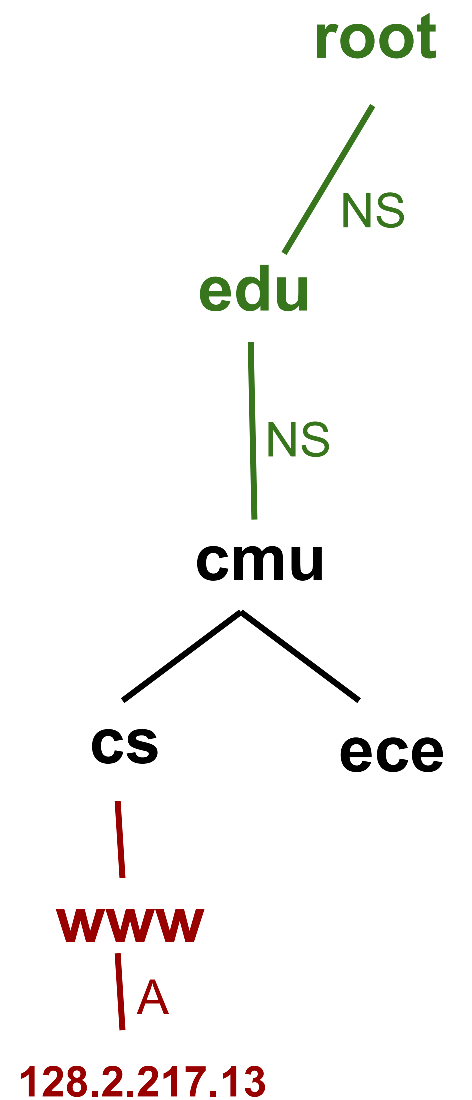
Common practices

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



Think about the effect of TTL

DNS (Summary)

- Motivations → large distributed database
  - Scalability
  - Independent update
  - Robustness
- Hierarchical database structure
  - Zones
  - Lookup query flow
- Caching and consistency in practice
  - Role of TTL

Topics Today

1. Naming at Internet Scale

    DNS - one of the world's largest databases

    DNS Architecture

2. Content Distribution at Internet Scale

    CDNs - some of the world's largest distributed systems

    Design Decisions

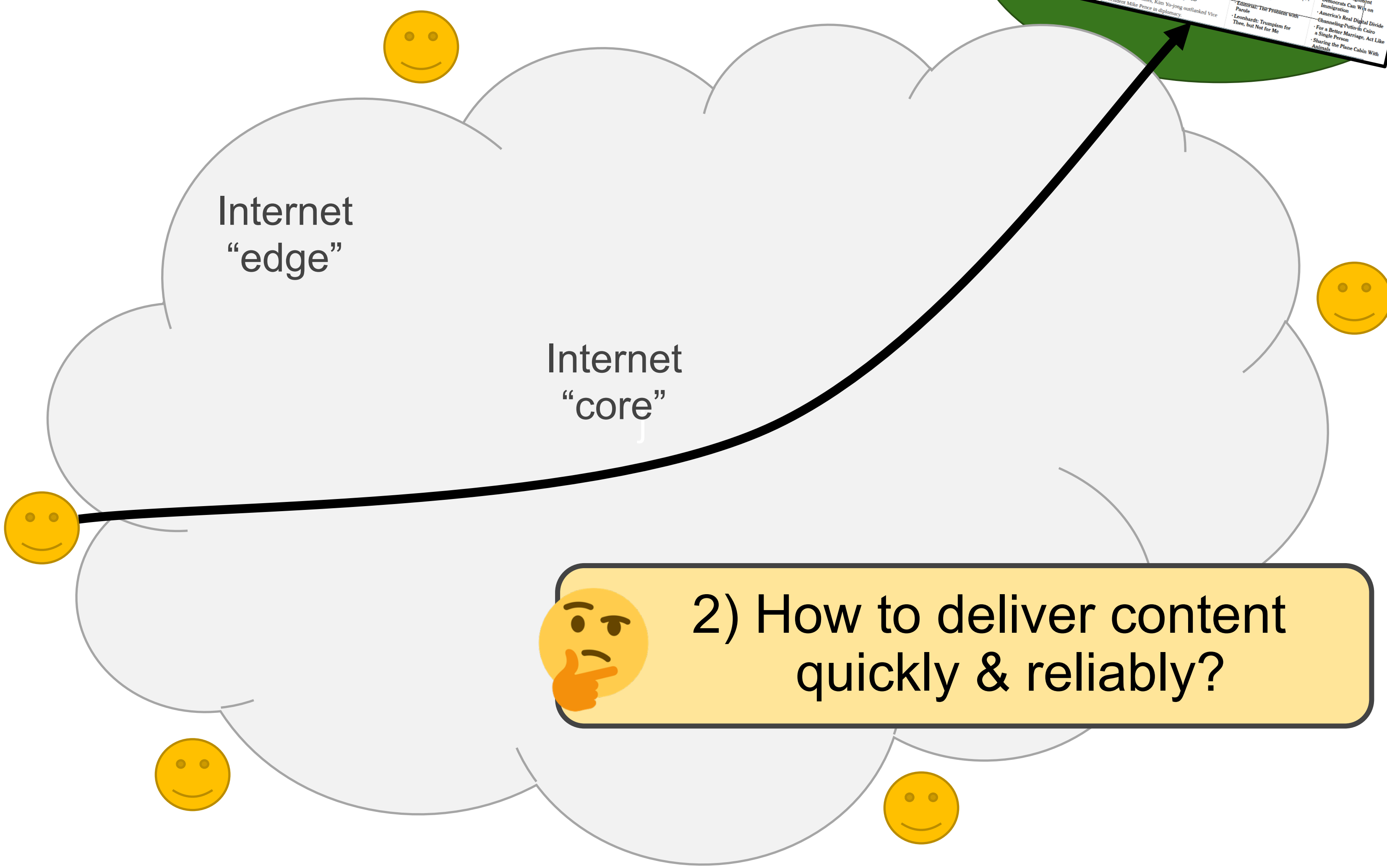
    Consistent Hashing for Scaling and Load Balancing

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


The New York Times

Abuse Claims Against Aide Further Roil White House

Kim Jong-un's Sister Upstages Mike Pence at the Olympics



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

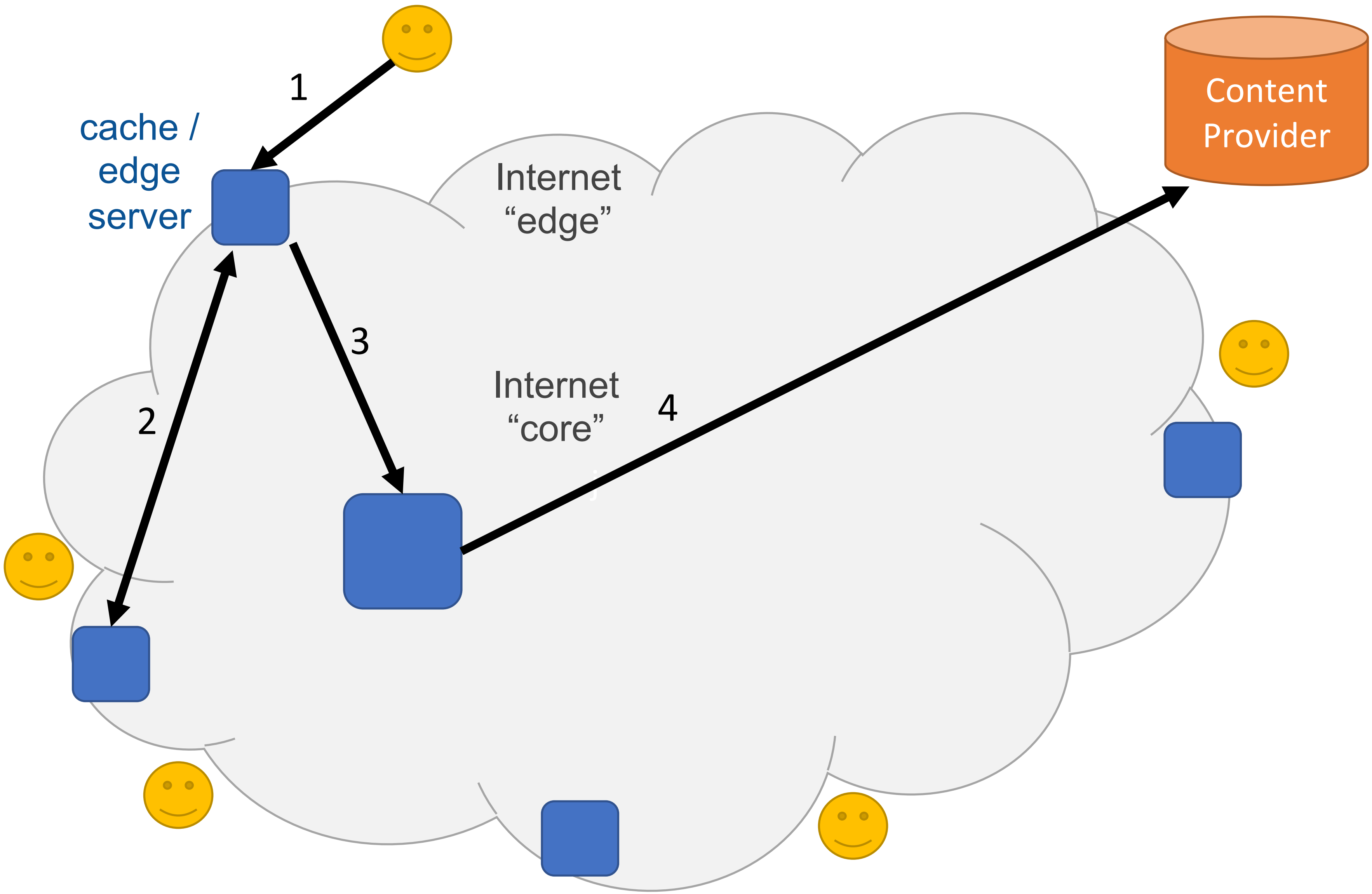
Why does this matter for performance?

Technique to reduce latency in a DS?

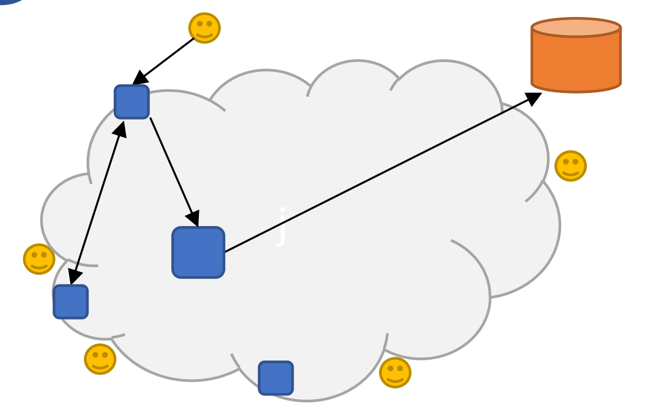
- Content Delivery Network (CDNs)
  - The world's largest distributed **caching** systems
  - Key for Internet performance
  - Explosive growth

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

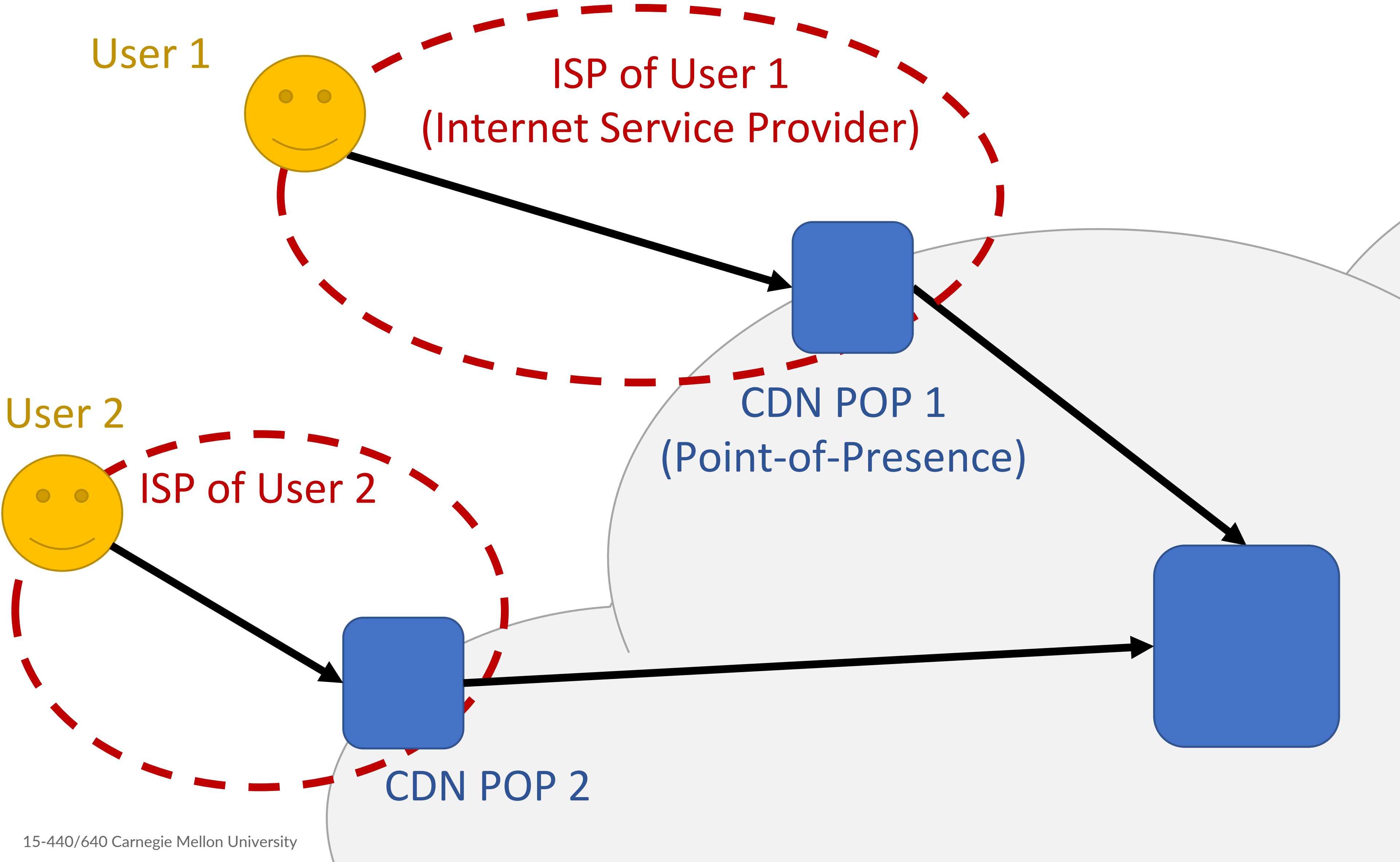


Some Key CDN Design Decisions



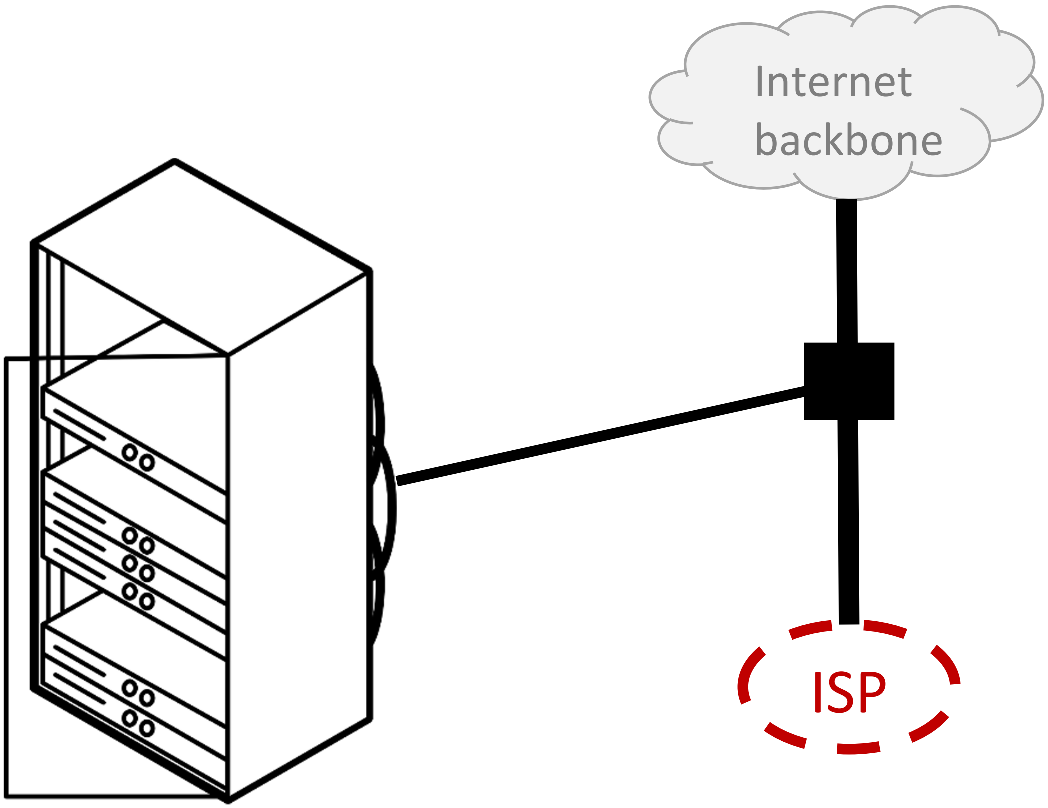
- Where and how to replicate content
- How to direct clients towards a CDN Point-of-Presence (PoP)
- How to choose a CDN server within a PoP
- How to propagate updates (CDN cache consistency)

Where to Replicate Content

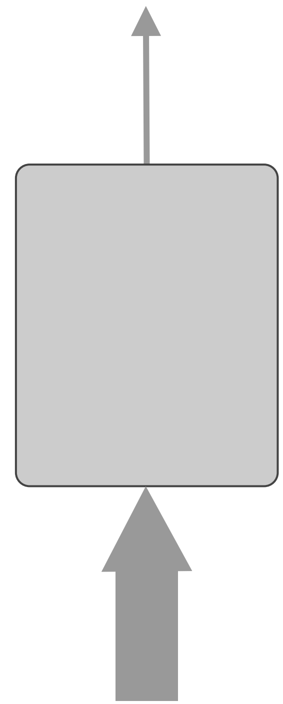


Where and How to Replicate

Rack(s) of edge servers



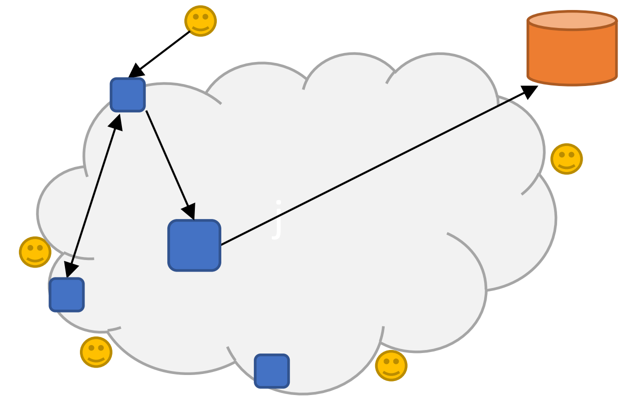
"Pull-based" edge servers



First check local cache

If cache miss, fetch from content provider

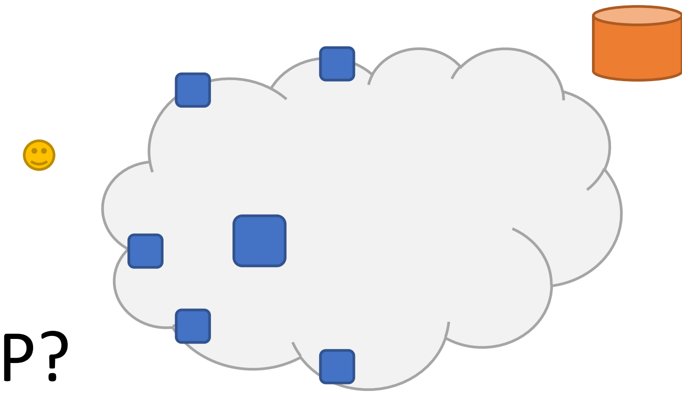
Some Key CDN Design Decisions



- Where and how to replicate content
- **How to direct clients towards a CDN Point-of-Presence (PoP)**
- How to choose a CDN server within a PoP
- How to propagate updates (CDN cache consistency)

Directing Users to CDNs

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



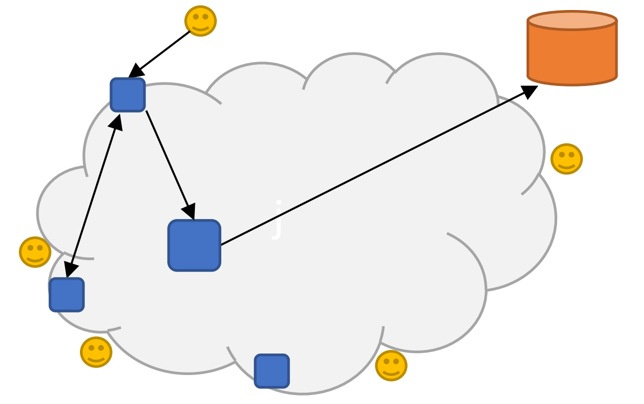
- How to direct user requests to the PoP?
  - Multiple ways
  - Examples:
    - As part of naming → DNS
      (e.g., CNAME that is resolved via CDN's name server)
    - As part of IP routing → anycast

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)?

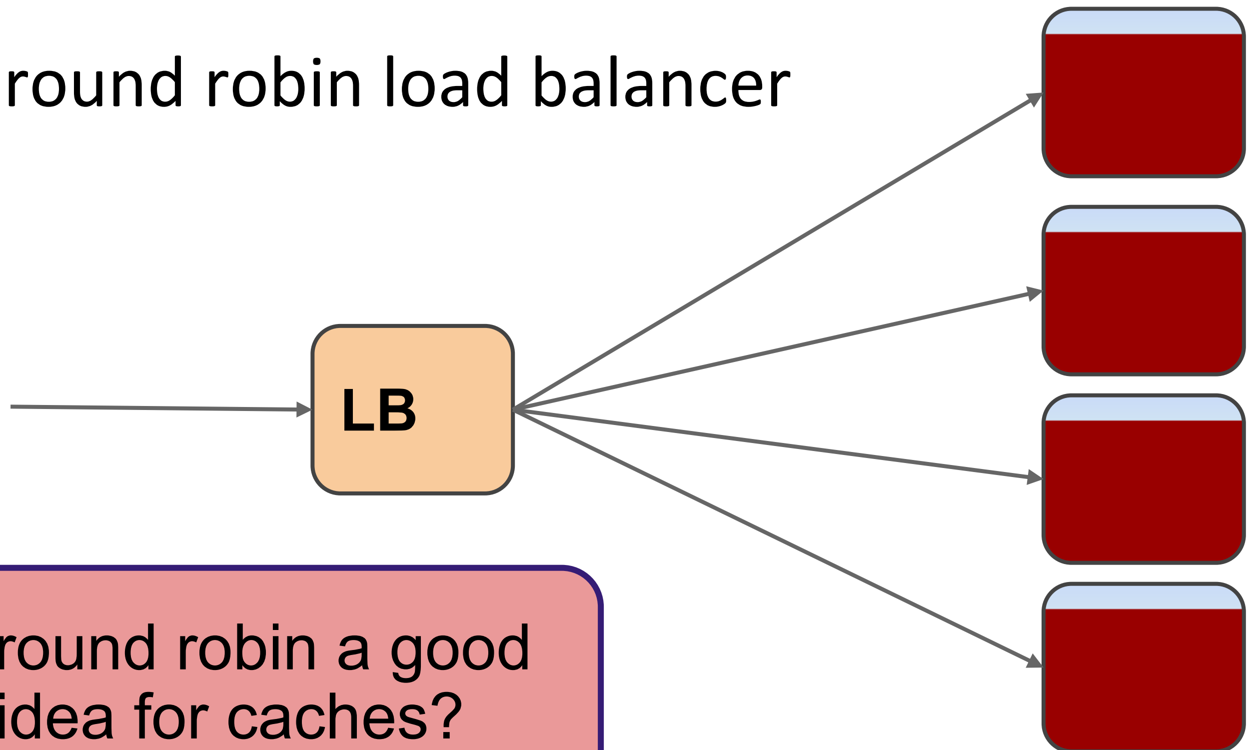
Some Key CDN Design Decisions



- Where and how to replicate content
- How to direct clients towards a CDN Point-of-Presence (PoP)
- **How to choose a CDN server within a PoP**
- How to propagate updates (CDN cache consistency)

CDN Scaling and Load Balancing

Idea 1: round robin load balancer



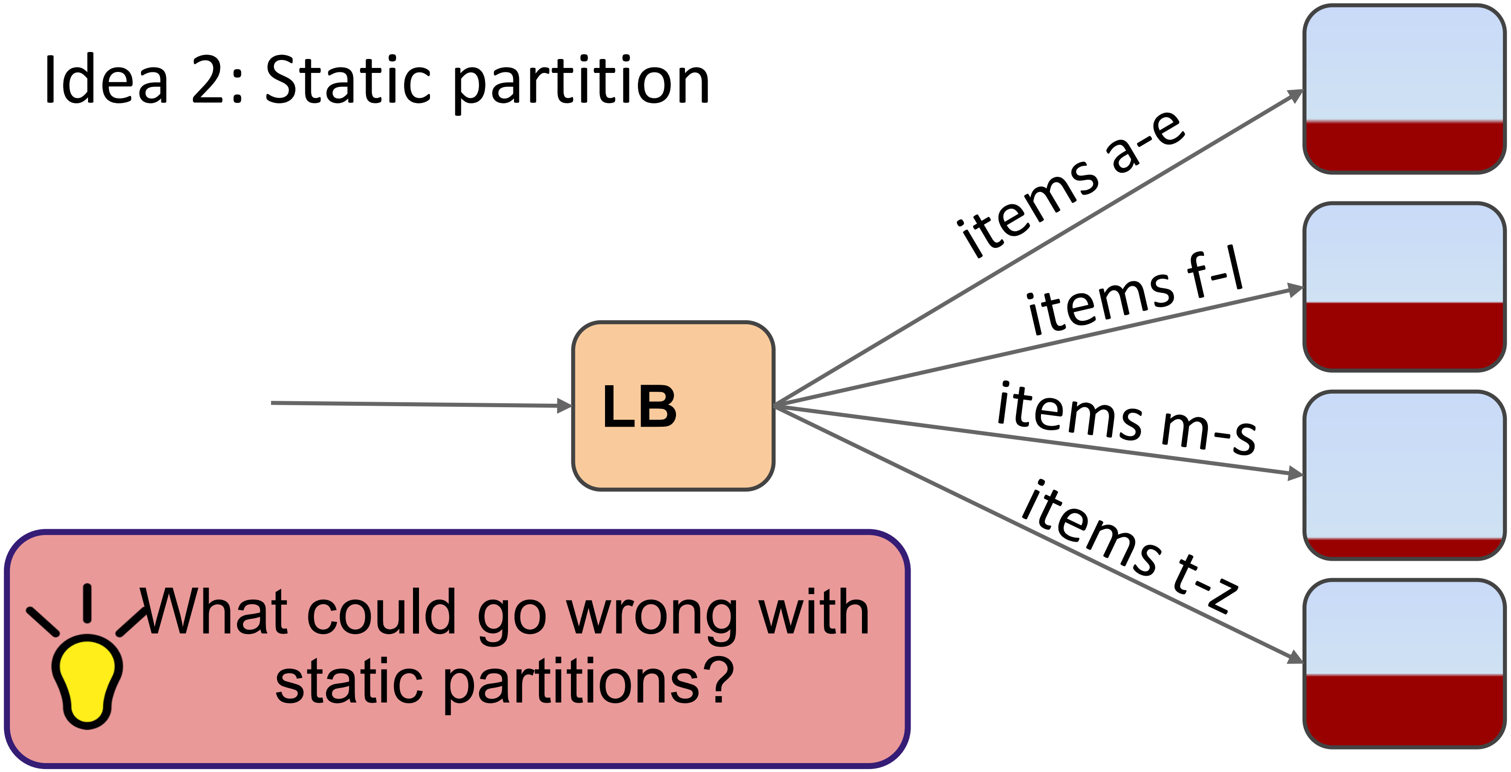
Is round robin a good idea for caches?

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 “tigers.com” had 1000000 pages, but “zebras.com” had only 10?
- Could fill up the bins as they arrive

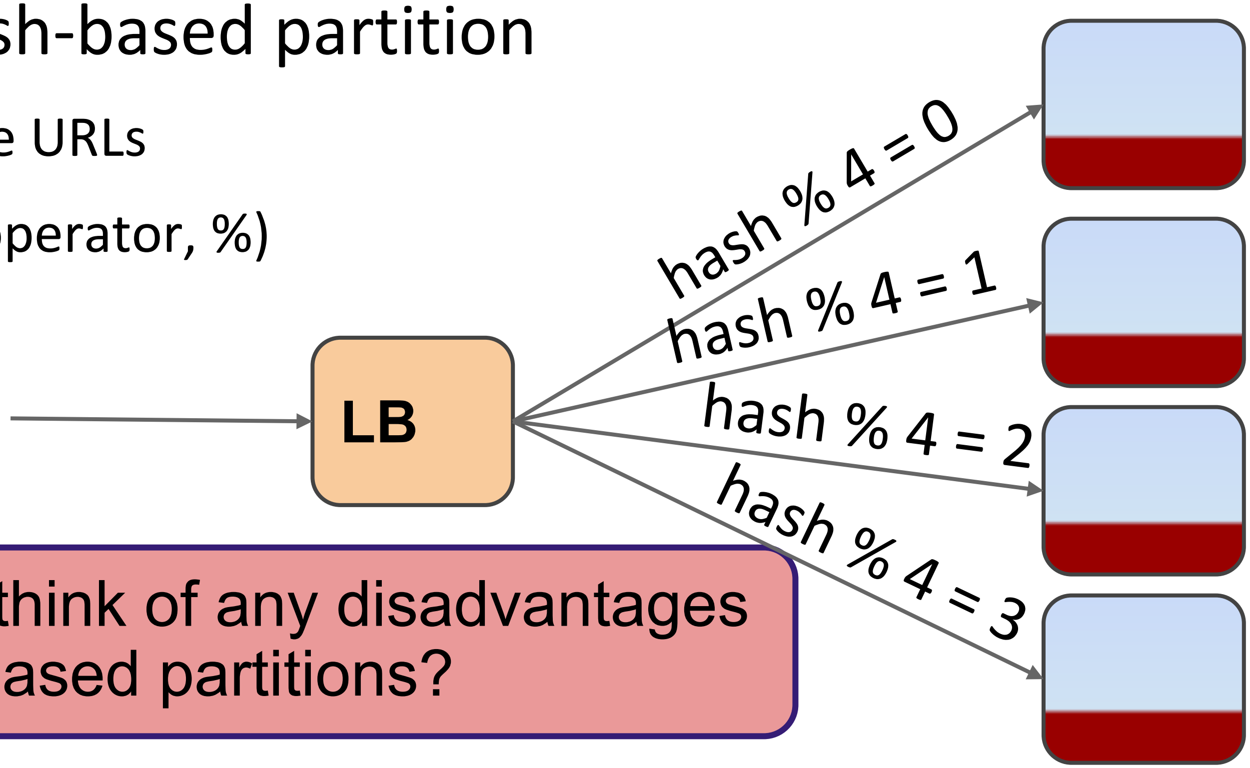
→ Requires tracking the location of **every** object at LB

Hash-Partitioned Load Balancer

Idea 3: Hash-based partition

(e.g., hash the URLs

use modulo operator, %)



Can you think of any disadvantages of hash-based partitions?

- 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 is added

How many documents are on the wrong server?

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

A large fraction of objects need to move!

=> Cache misses

How do we fix hash-based partitioning?

Solution: Consistent Hashing

Idea 4: Consistent Hashing

- Special type of hashing
- Can resize table without shuffling all entries
- On average only $1/n^{th}$ of entries will be moved when adding/removing a node
  - (where n = total number of nodes)

Consistent Hashing

- Key idea: map both nodes and keys to the same (metric) identifier space
  - E.g., Hash to a m-bit identifier

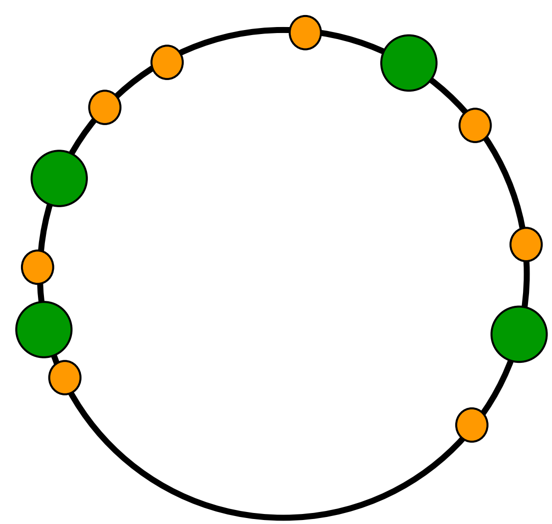
Node identifier: SHA-1(IP address)

IP="198.10.10.1" —SHA-1→ ID=123

Key identifier: SHA-1(key)

key="LetItBe" —SHA-1→ ID=60

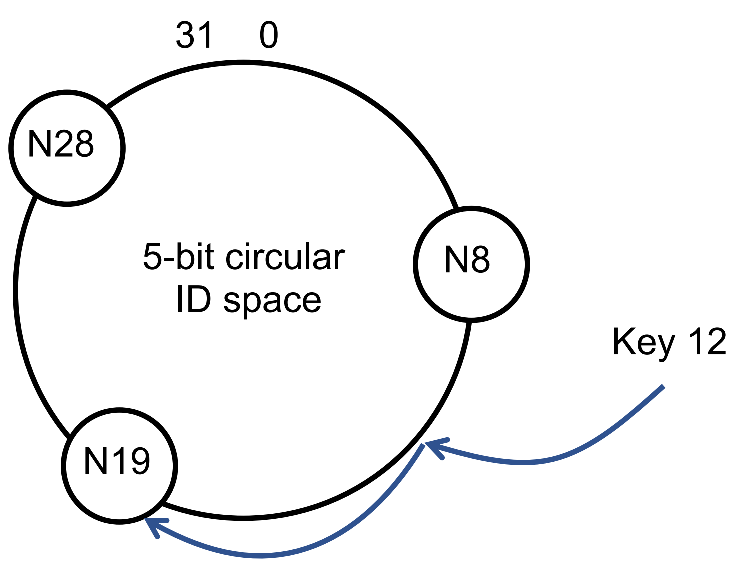
- Identifier space organized as ring



Consistent Hashing

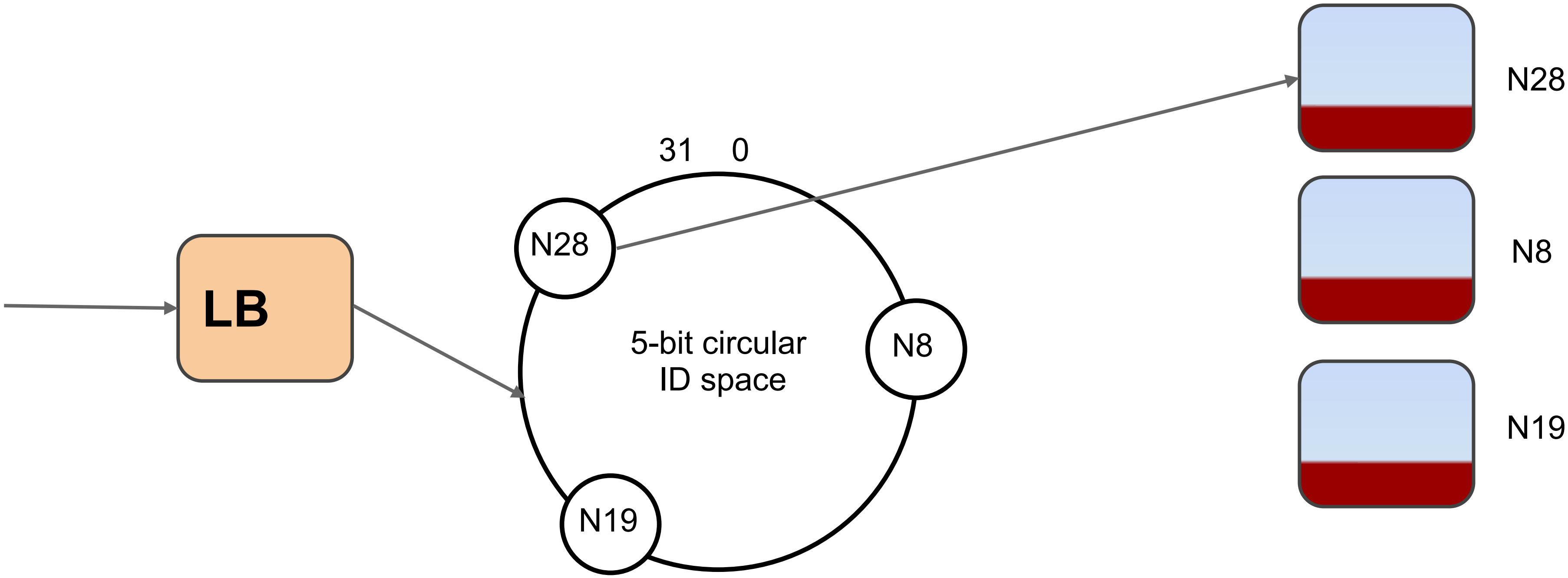
How to map key IDs to node IDs?

- Keys mapped to the successor node
  - Node with immediately next higher ID

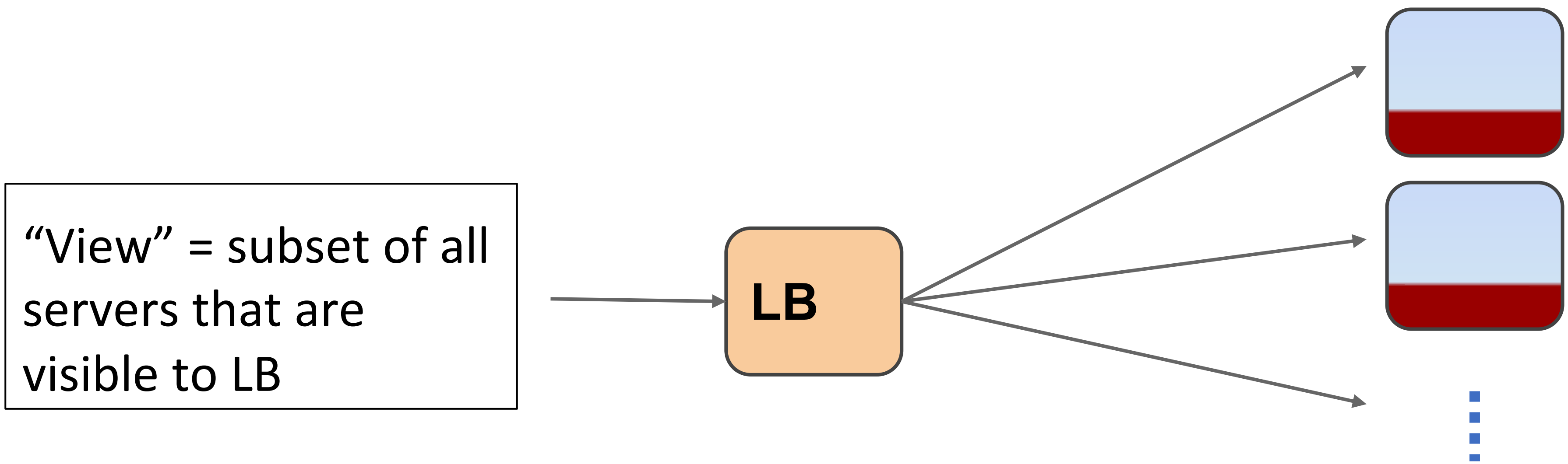


Note: circular ID space
so 29-31, 0-8 map to N8

Consistent Hashing



Properties of Consistent Hashing



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

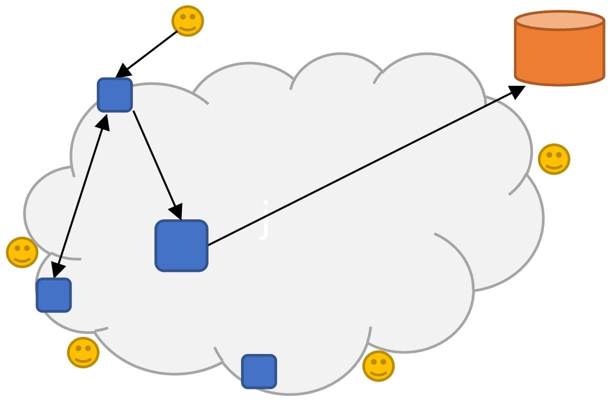
Very useful in other distributed systems too
E.g., Distributed Hash Tables in peer-to-peer systems

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?)

35

Some Key CDN Design Decisions



- Where and how to replicate content
- How to direct clients towards a CDN Point-of-Presence (PoP)
- How to choose a CDN server within a PoP
- **How to propagate updates (CDN cache consistency)**

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

Edge Applications (only partial adoption)

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

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:
  - Optimizing resource consumption
  - Automation in performance tuning
  - Large content providers deploy their own CDNs
  - Many open problems (performance and security)

39

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
- Use consistent hashes and many other DS techniques