EN.601.414/614 Computer Networks

CDN and DNS

Xin Jin

Spring 2019 (MW 3:00-4:15pm in Shaffer 301)

JOHNS HOPKINS UNIVERSITY

https://github.com/xinjin/course-net

Question on differences between connection and circuit

- **Connection**
  - A transport layer concept
  - Resources are reserved at end hosts (sender & receiver)
  - Need the underlying network layer to send data

- **Circuit**
  - A network layer concept
  - Resources are reserved at each hop
  - Circuit switching is one way to support a connection; packet switching is the other way

Agenda

- **CDN: Content Distribution Network**
- **DNS: Domain Name System**

Recap:
Improving HTTP performance

- **Optimizing connections using three “P”s**
  - Persistent connections
  - Parallel/concurrent connections
  - Pipelined transfers over the same connection

Caching

- **Why does caching work?**
  - ➢ Exploit locality of reference
- **How well does caching work?**
  - ➢ Very well, up to a limit
  - ➢ Large overlap in content
  - ➢ But many unique requests
    - Empirical result: effectiveness of caching (cache hit ratio) grows logarithmically with user size

Caching: How

- **Modifier to GET requests:**
  - `If-modified-since` – returns "not modified" if resource not modified since specified time

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
If-modified-since: Wed, 18 Jan 2017 10:25:50 GMT
(blank line)
```

Caching: How

- **Modifier to GET requests:**
  - `If-modified-since` – returns “not modified” if resource not modified since specified time
- **Client specifies “`if-modified-since`” time in request**
- **Server compares this against “last modified” time of resource**
- **Server returns “Not Modified” if resource has not changed**
- **…. or a “OK” with the latest version otherwise**

Caching: How

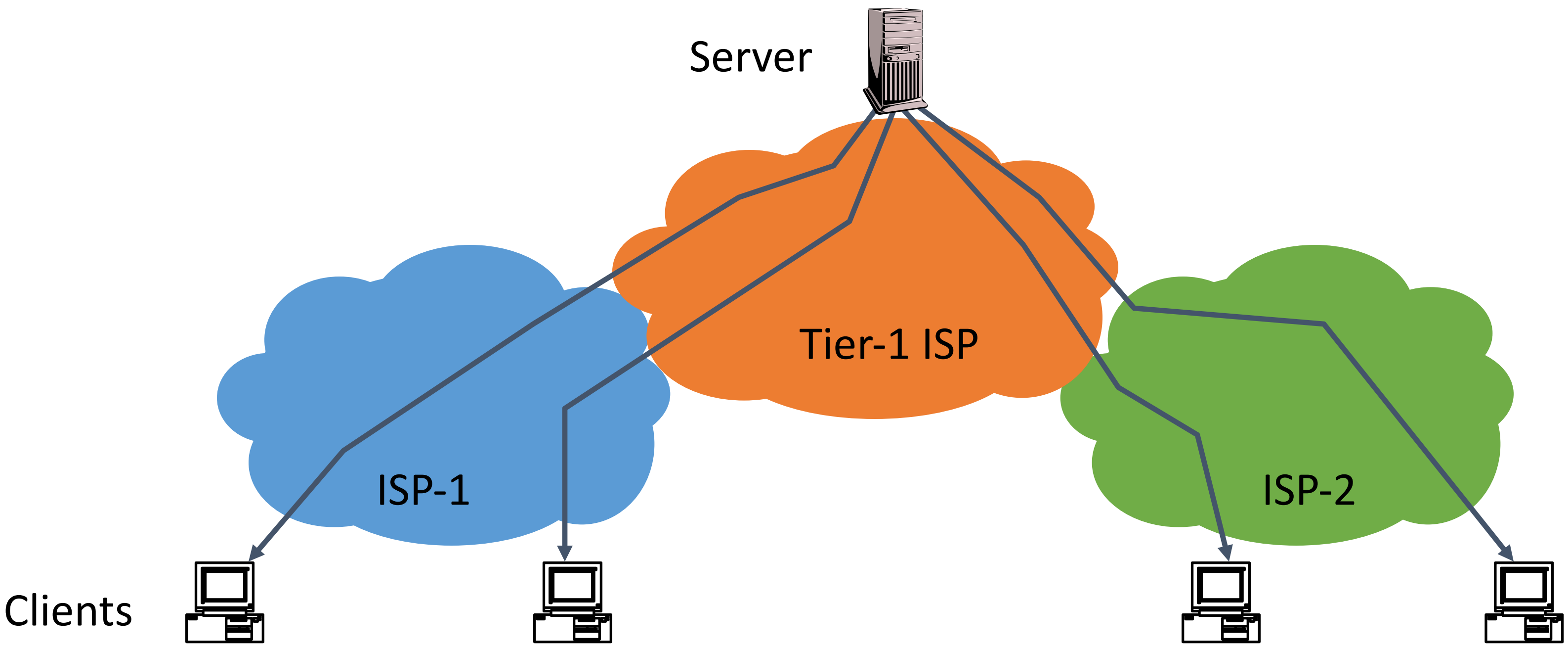
- **Modifier to GET requests:**
  - `If-modified-since` – returns "not modified" if resource not modified since specified time
- **Response header:**
  - `Expires` – how long it's safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

Caching: Where?

- **Options**
  - ➢ Client (browser)
  - ➢ Forward proxies
  - ➢ Reverse proxies
  - ➢ Content Distribution Network

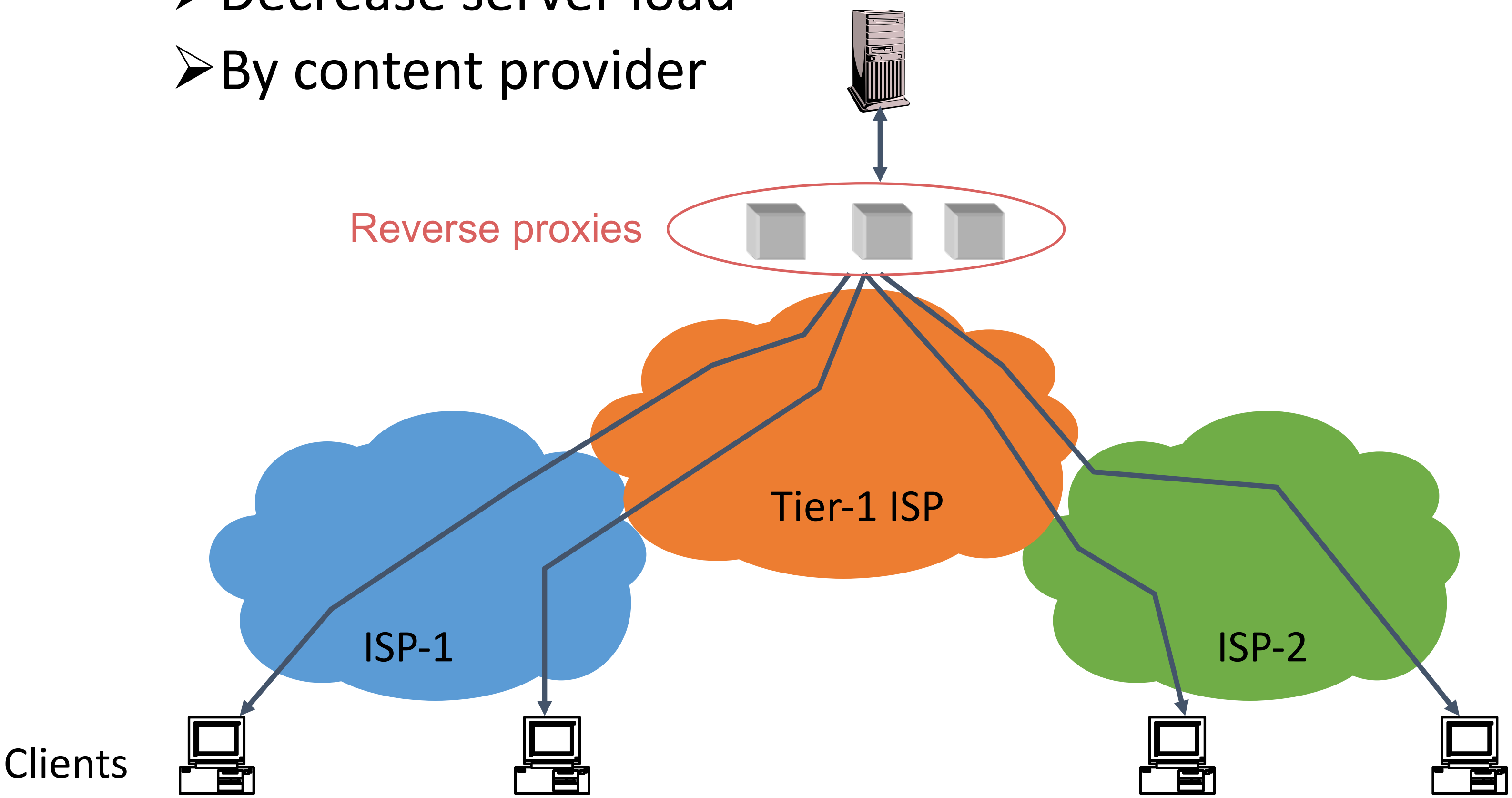
Caching: Where?

- **Many clients transfer same information**
  - Generate unnecessary server and network load
  - Clients experience unnecessary latency



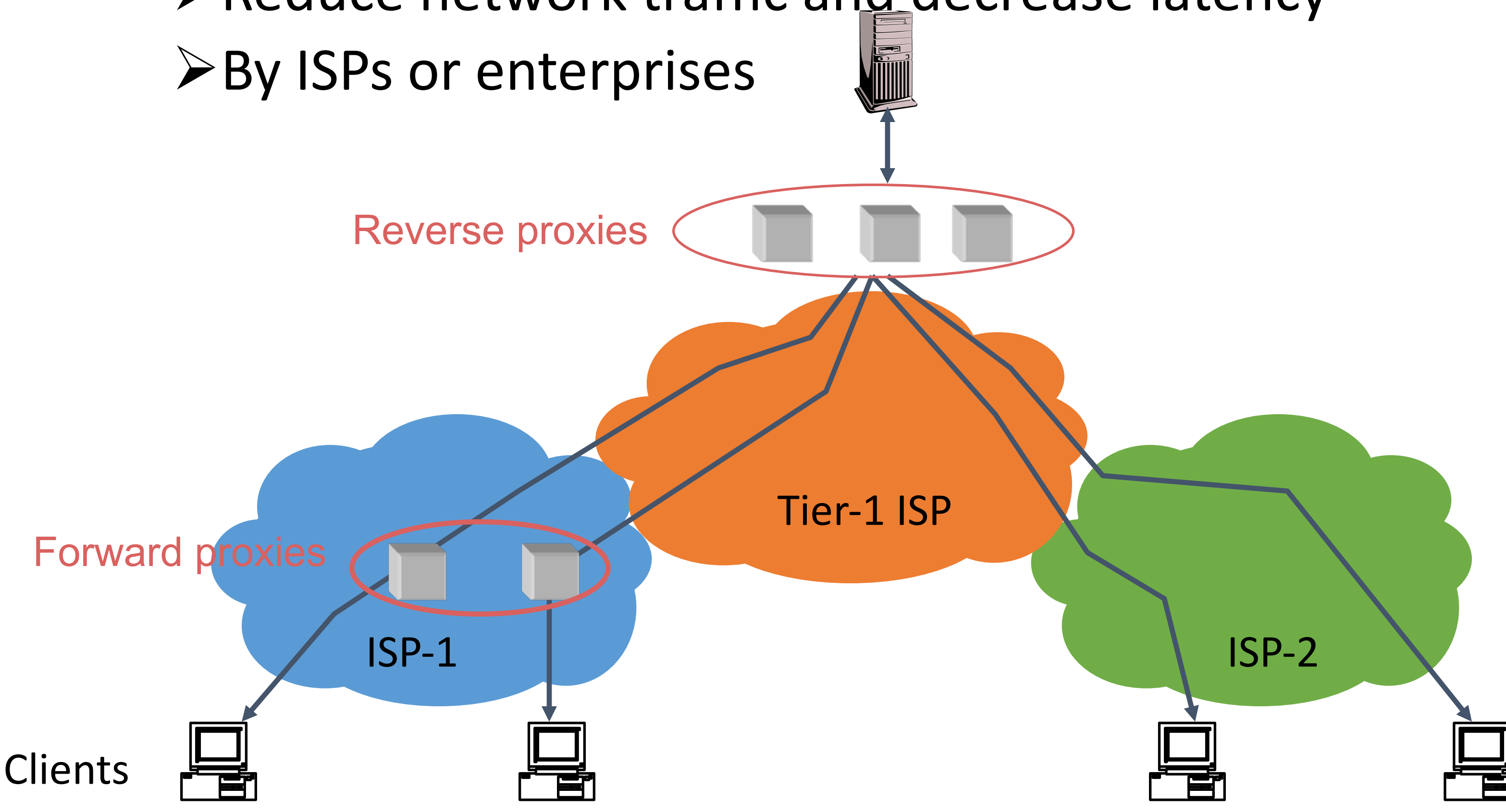
Caching with Reverse Proxies

- **Cache documents close to server**
  - Decrease server load
  - By content provider



Caching with Forward Proxies

- **Cache documents close to clients**
  - Reduce network traffic and decrease latency
  - By ISPs or enterprises



Replication

- **Replicate popular Websites across many machines**
  - Spread load across servers
  - Place content closer to clients
  - Help when content isn’t cacheable

Recap: Improving HTTP performance

- **Optimizing connections using three "P"s**
  - Persistent connections
  - Parallel/concurrent connections
  - Pipelined transfers over the same connection
- **Caching**
  - Forward proxy: close to clients
  - Reverse proxy: close to servers
- **Replication**

Content Distribution Networks (CDN)

- **Caching and replication as a service**
- **Large-scale distributed storage infrastructure (usually) administered by one entity**
  - e.g., Akamai has servers in 20,000+ locations
- **Combination of caching and replication**
  - Pull: Direct result of clients' requests (caching)
  - Push: Expectation of high access rate (replication)
- **Can do some processing to handle dynamic webpage content**

Cost-effective content delivery

- **General theme: multiple sites hosted on shared physical infrastructure**
  - ➢ Efficiency of statistical multiplexing
  - ➢ Economies of scale (volume pricing, etc.)
  - ➢ Amortization of human operator costs
- **Examples:**
  - ➢ CDNs
  - ➢ Web hosting companies
  - ➢ Cloud infrastructure

CDN example – Akamai

- **Akamai creates new domain names for each client**
  - e.g., a128.g.akamai.net for cnn.com
- **The client content provider modifies content so that embedded URLs reference new domains**
  - "Akamaize" content
  - e.g., http://www.cnn.com/image-of-the-day.gif becomes http://a128.g.akamai.net/image-of-the-day.gif
- **Requests now sent to CDN's infrastructure**

How to direct clients to particular replicas?

- **In order to**
  - ➢Balancing load across server replicas
  - ➢Pairing clients with nearby servers to decrease latency and overall bandwidth usage

DNS: Domain name system

Internet names & addresses

- **Machine addresses: e.g., 141.212.113.143**
  - Router-usable labels for machines
  - Conforms to network structure (the "where")
- **Machine names: e.g., cs.jhu.edu**
  - Human-usable labels for machines
  - Conforms to organizational structure (the "who")
- **The Domain Name System (DNS) is how we map from one to the other**
  - A directory service

Why?

- **Convenience**
  - Easier to remember www.google.com than 216.58.216.100
- **Provides a level of indirection!**
  - Decoupled names from addresses
  - Many uses beyond just naming a specific host

DNS: History

- **Initially all host-address mappings were in a `hosts.txt` file (in `/etc/hosts`):**
  - ➢Maintained by the Stanford Research Institute (SRI)
  - ➢Changes were submitted to SRI by email
  - ➢New versions of `hosts.txt` periodically FTP'd from SRI

DNS: History (cont'd)

- **As the Internet grew this system broke down**
  - ➢SRI couldn't handle the load
  - ➢Names were not unique
  - ➢Hosts had inaccurate copies of `hosts.txt`
- **The Domain Name System (DNS) was invented to fix this**

Goals

- **Uniqueness: no naming conflicts**
- **Scalable**
  - Many names and frequent updates
- **Distributed, autonomous administration**
  - Ability to update my own (machines') names
  - Don't have to track everybody's updates
- **Highly available**
- **Lookups are fast**
- **Perfect consistency is a non-goal**

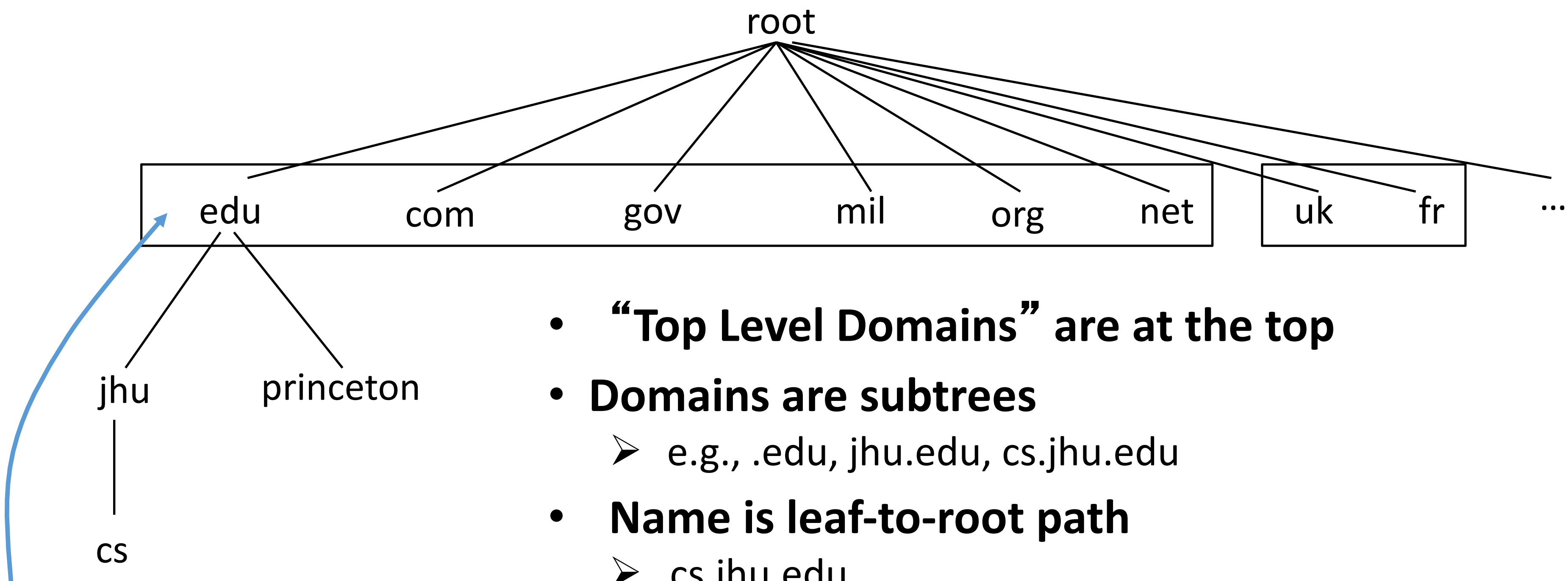
How?

- **Partition the namespace**
- **Distribute administration of each partition**
  - ➢Autonomy to update my own (machines') names
  - ➢Don't have to track everybody's updates
- **Distribute name resolution for each partition**
- **How should we partition things?**

Key idea: Hierarchy

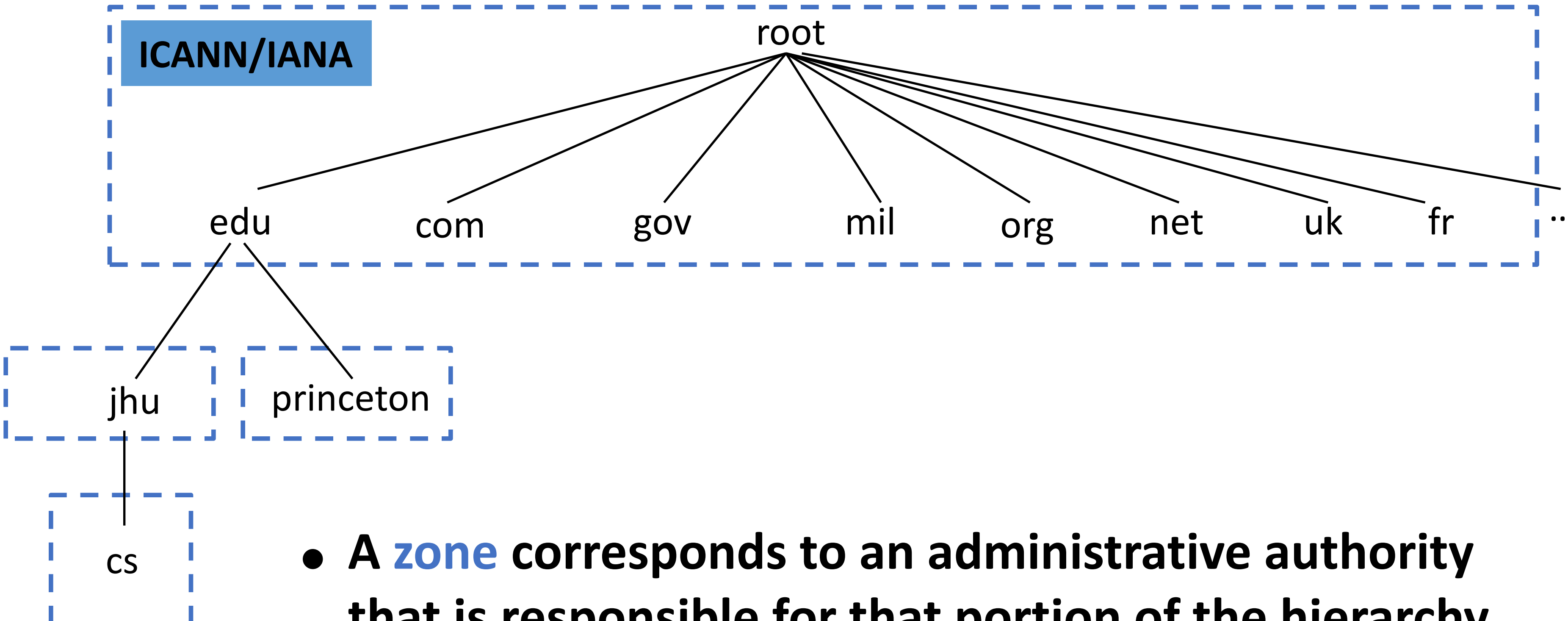
- **Three intertwined hierarchies**
  - Hierarchical namespace
    - As opposed to original flat namespace
  - Hierarchically administered
    - As opposed to centralized
  - (Distributed) hierarchy of servers
    - As opposed to centralized storage

Hierarchical namespace



- "Top Level Domains" are at the top
- **Domains are subtrees**
  - e.g., .edu, jhu.edu, cs.jhu.edu
- **Name is leaf-to-root path**
  - cs.jhu.edu
- **Depth of tree is arbitrary (limit 128)**
- **Name collisions trivially avoided**
  - Each domain is responsible

Hierarchical administration



- A **zone** corresponds to an administrative authority that is responsible for that portion of the hierarchy
  - e.g., JHU controls names: *.jhu.edu
  - e.g., CS controls names: *.cs.jhu.edu

Server hierarchy

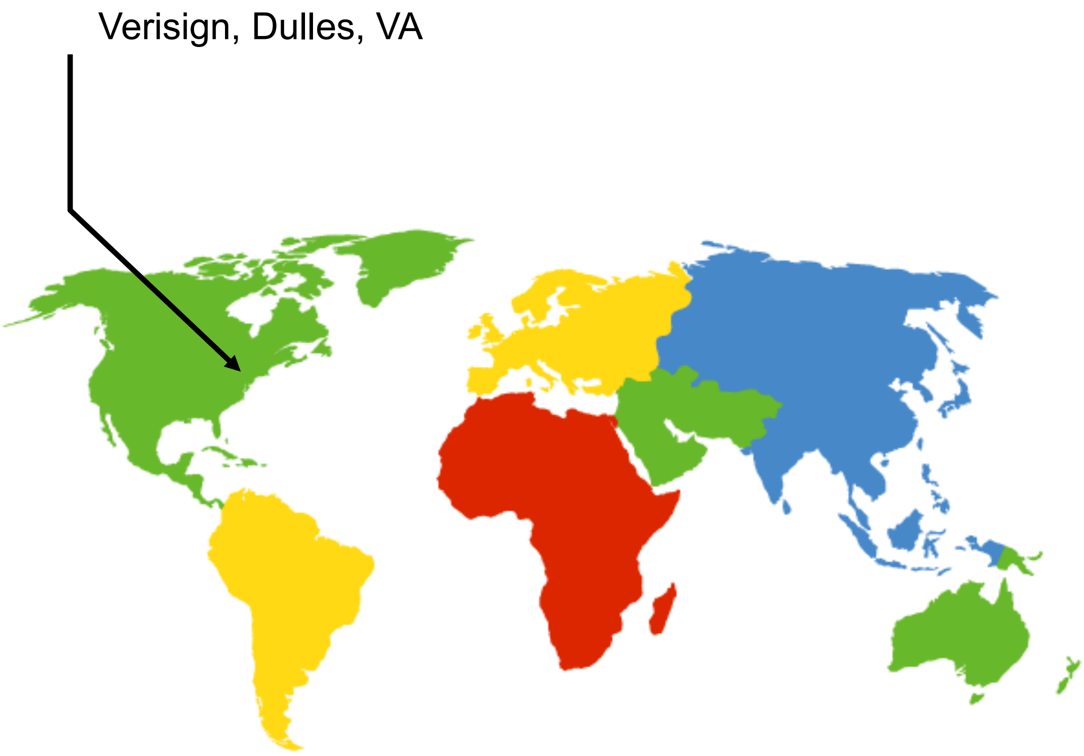
- **Top of hierarchy: Root servers**
  - Location hardwired into other servers
- **Next Level: Top-level domain (TLD) servers**
  - .com, .edu, etc.
  - Managed professionally
- **Bottom Level: Authoritative DNS servers**
  - Actually store the name-to-address mapping
  - Maintained by the corresponding administrative authority

Server hierarchy

- **Each server stores a (small!) subset of the total DNS database**
- **An authoritative DNS server stores "resource records" for all DNS names in the domain that it has authority for**
- **Each server needs to know other servers that are responsible for the other portions of the hierarchy**
    - Every server knows the root
    - Root server knows about all top-level domains

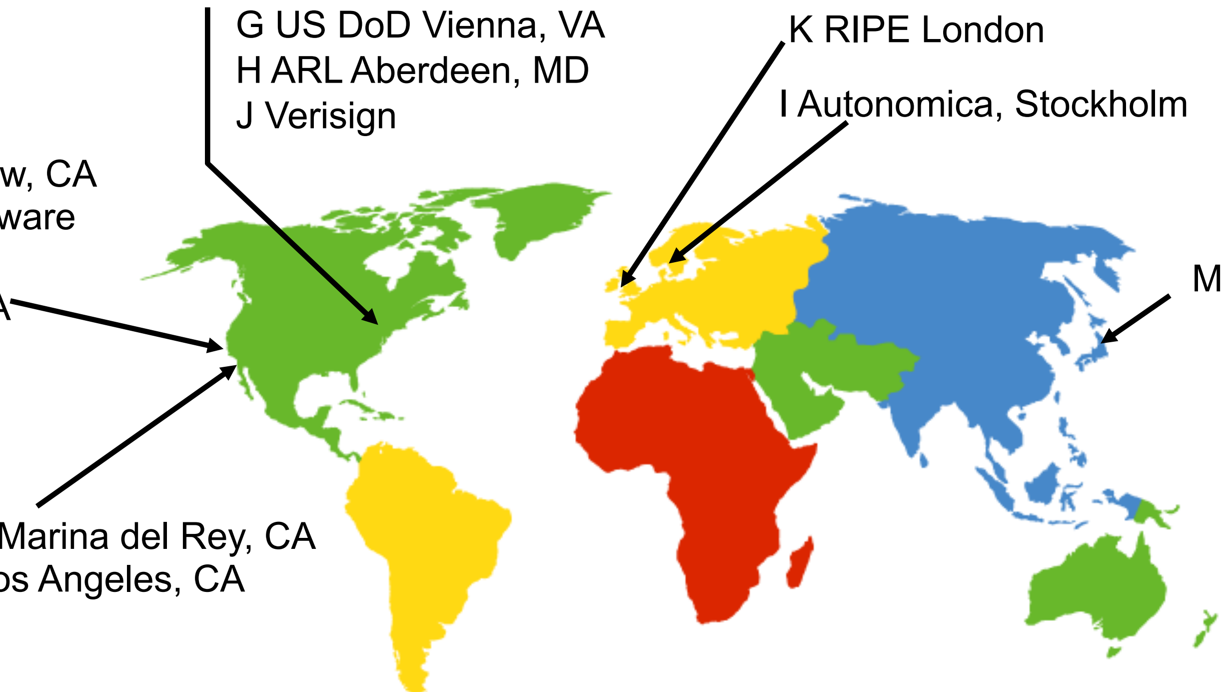
DNS root

- **Located in Virginia, USA**
- **How do we make the root scale?**



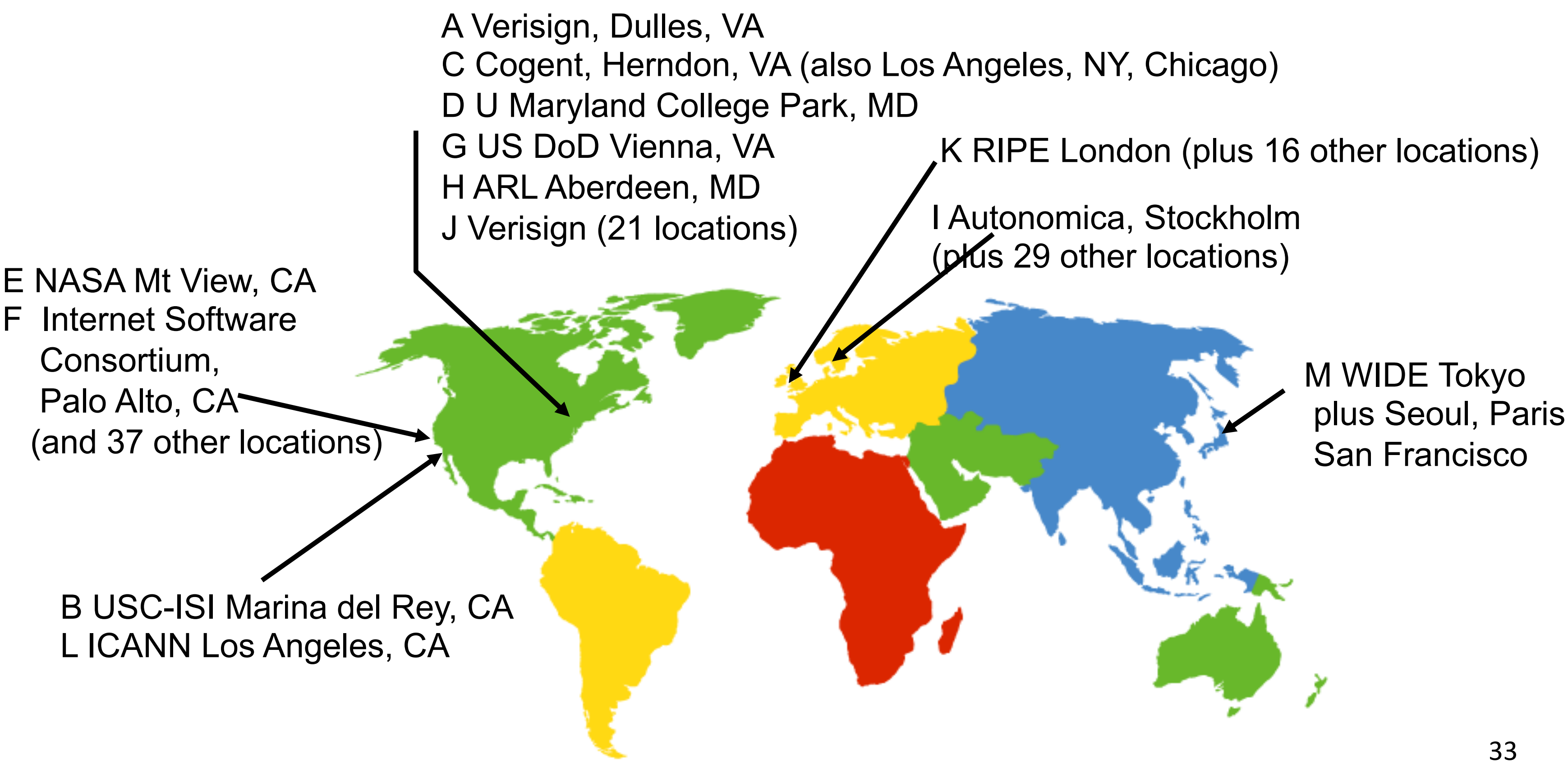
DNS root servers

- **13 root servers (labeled A-M; see http://www.root-servers.org/)**



DNS root servers

- **13 root servers replicated via anycast**



Anycast in a nutshell

- **Routing finds shortest paths to destination**
- **If several locations are given the same address, then the network will deliver the packet to the closest location with that address**

- **Characteristics**
  - ➢Very robust
  - ➢Requires no modification to routing algorithms

DNS records

- **DNS servers store resource records (RRs)**
  - ➢RR is (name, value, type, TTL)
- **Type = A: (→ Address)**
  - ➢name = hostname
  - ➢value = IP address
- **Type = NS: (→ Name Server)**
  - ➢name = domain
  - ➢value = name of DNS server for domain

DNS records (cont'd)

- **Type = CNAME: (→ Canonical Name)**
  - name = alias name for some "canonical" (real) name
    - e.g., documents.example.com is really docs.example.com
  - value = canonical name
- **Type = MX: (→ Mail eXchanger)**
  - name = domain in email address
  - value = name(s) of mail server(s)

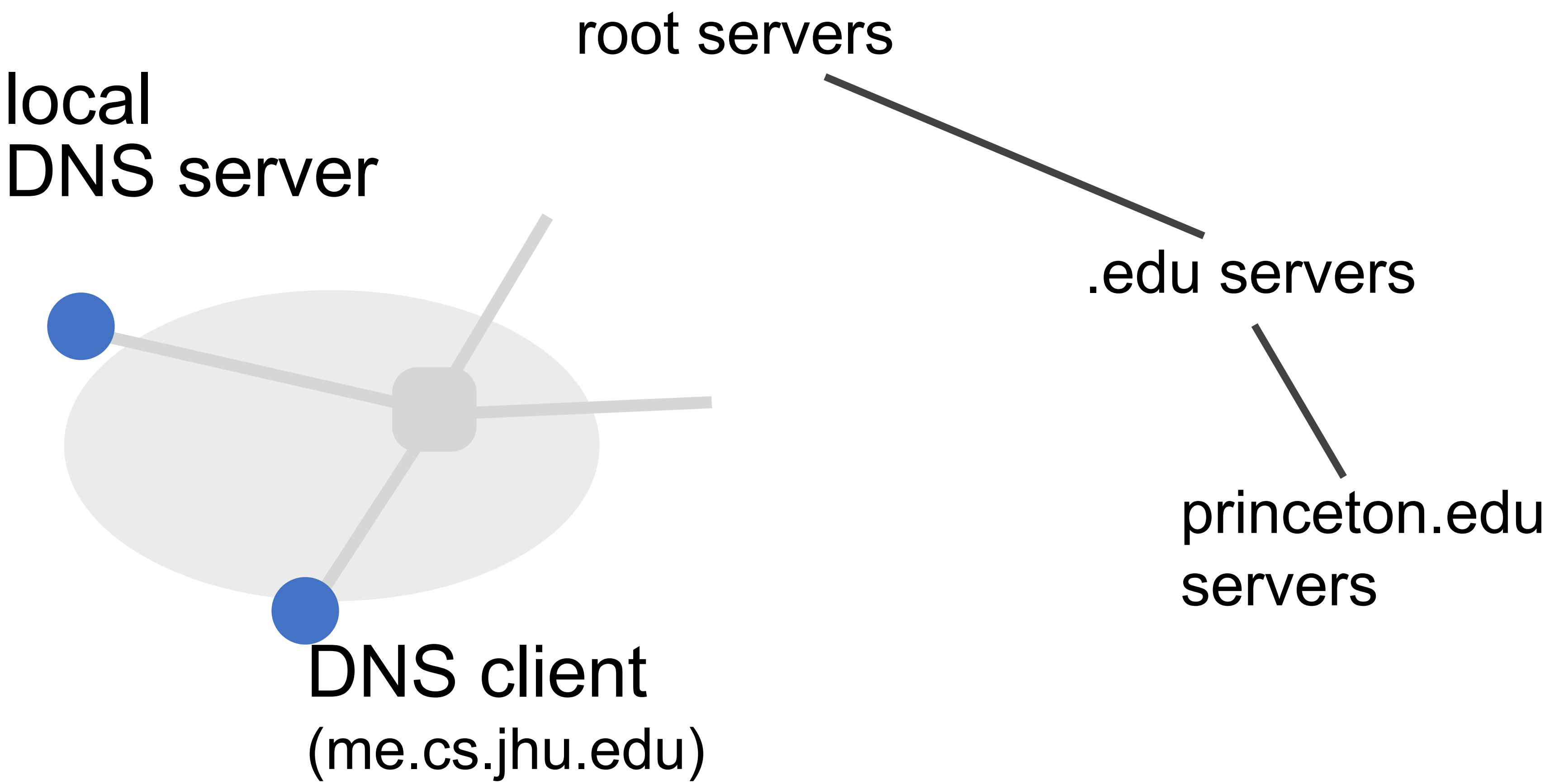
Inserting Resource Records into DNS

- **Register foobar.com at registrar (GoDaddy)**
  - Provide registrar with names and IP addresses of your authoritative name server(s)
  - Registrar inserts RR pairs into the .com TLD server:
    - (foobar.com, dns1.foobar.com, NS)
    - (dns1.foobar.com, 212.44.9.129, A)
- **Store resource records in your server dns1.foobar.com**
  - e.g., type A record for www.foobar.com
  - e.g., type MX record for foobar.com

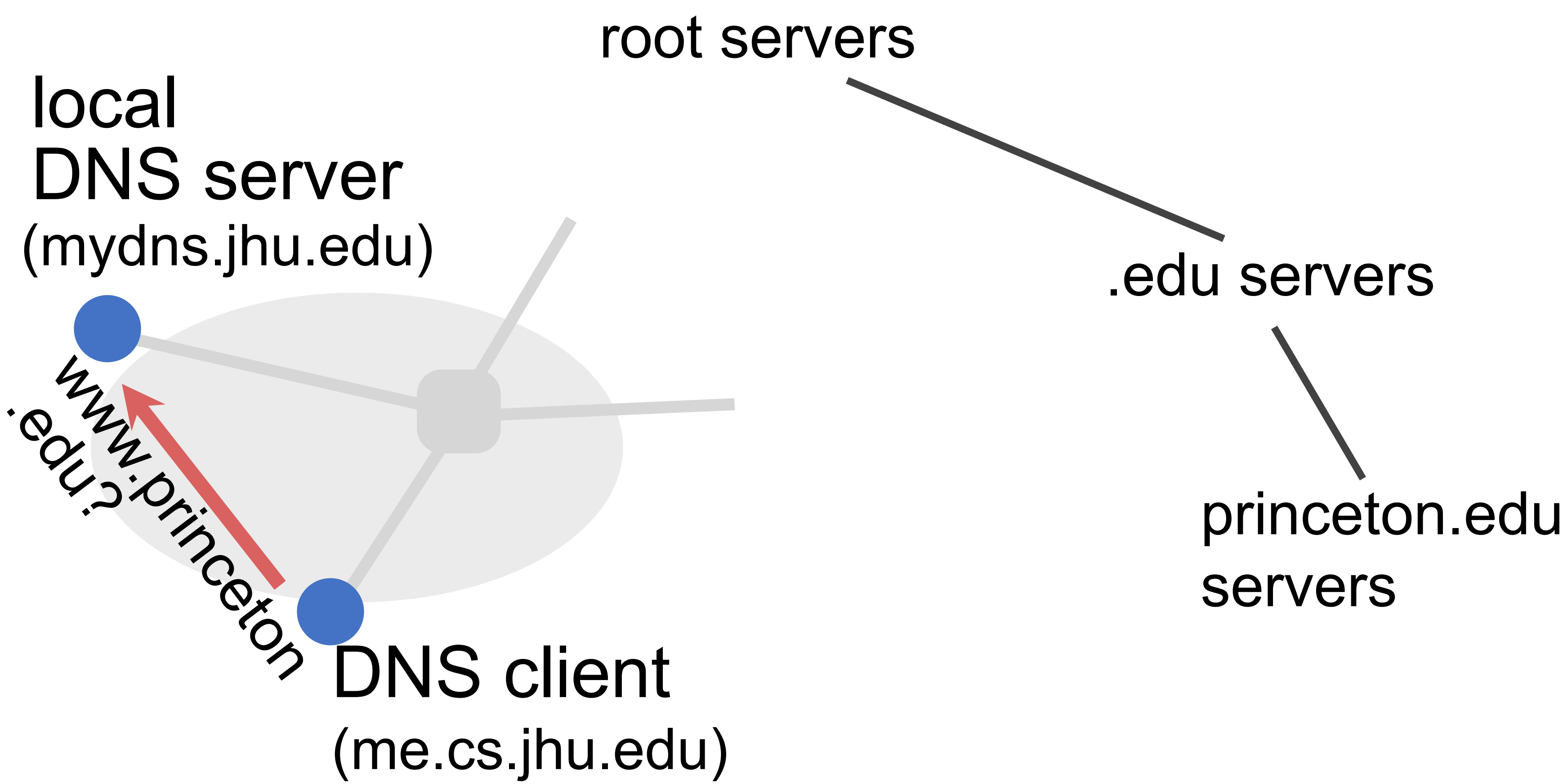
Using DNS (Client/App View)

- **Two components**
  - ➢ Local DNS servers
  - ➢ Resolver software on hosts
- **Local DNS server (“default name server”)**
  - ➢ Clients configured with default server’s address OR learn it via a host configuration protocol (e.g., DHCP)
- **Client application**
  - ➢ Obtain DNS name (e.g., from URL)
  - ➢ Do `gethostbyname()` to trigger DNS request to its local DNS server

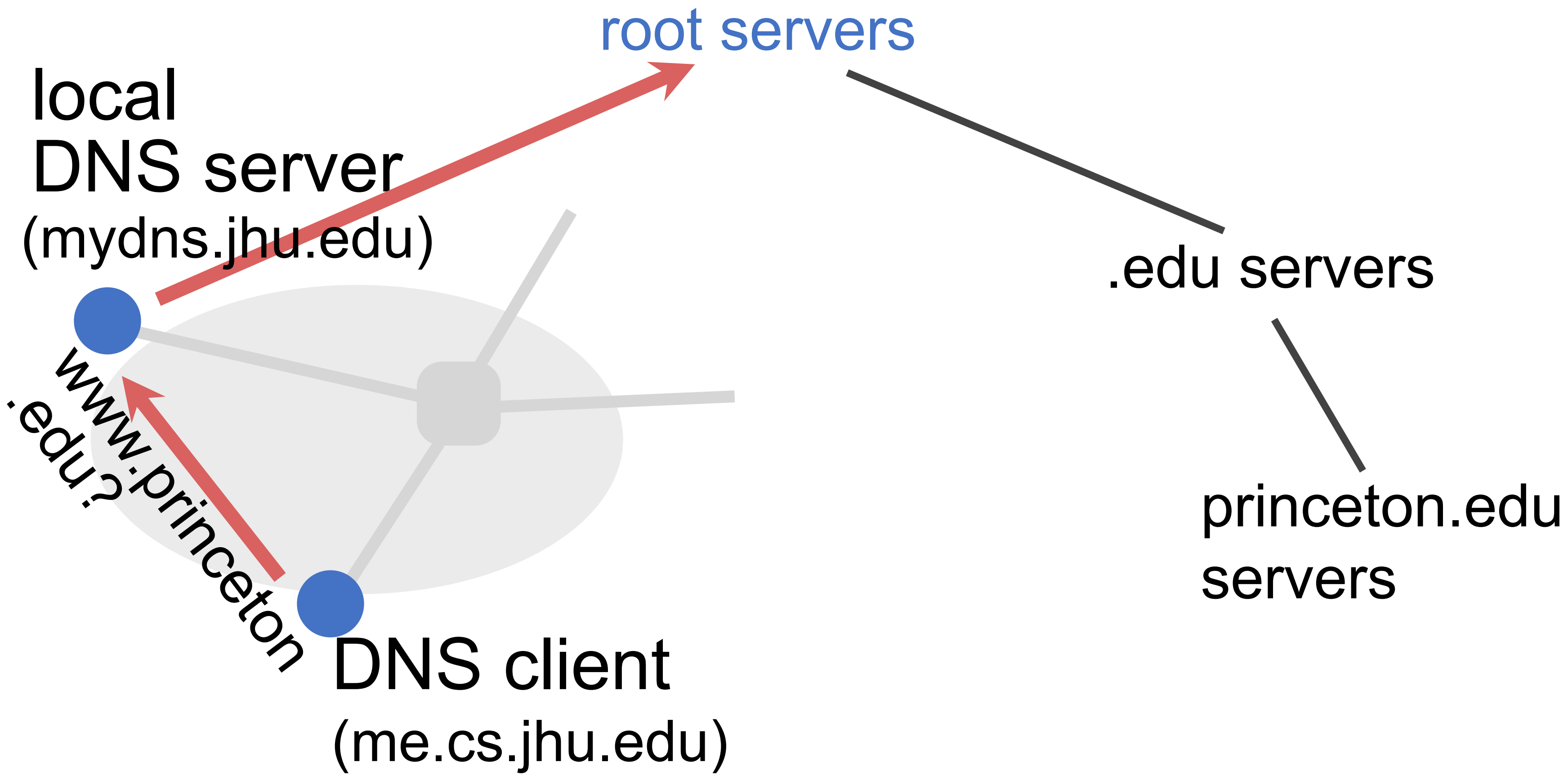
Name resolution



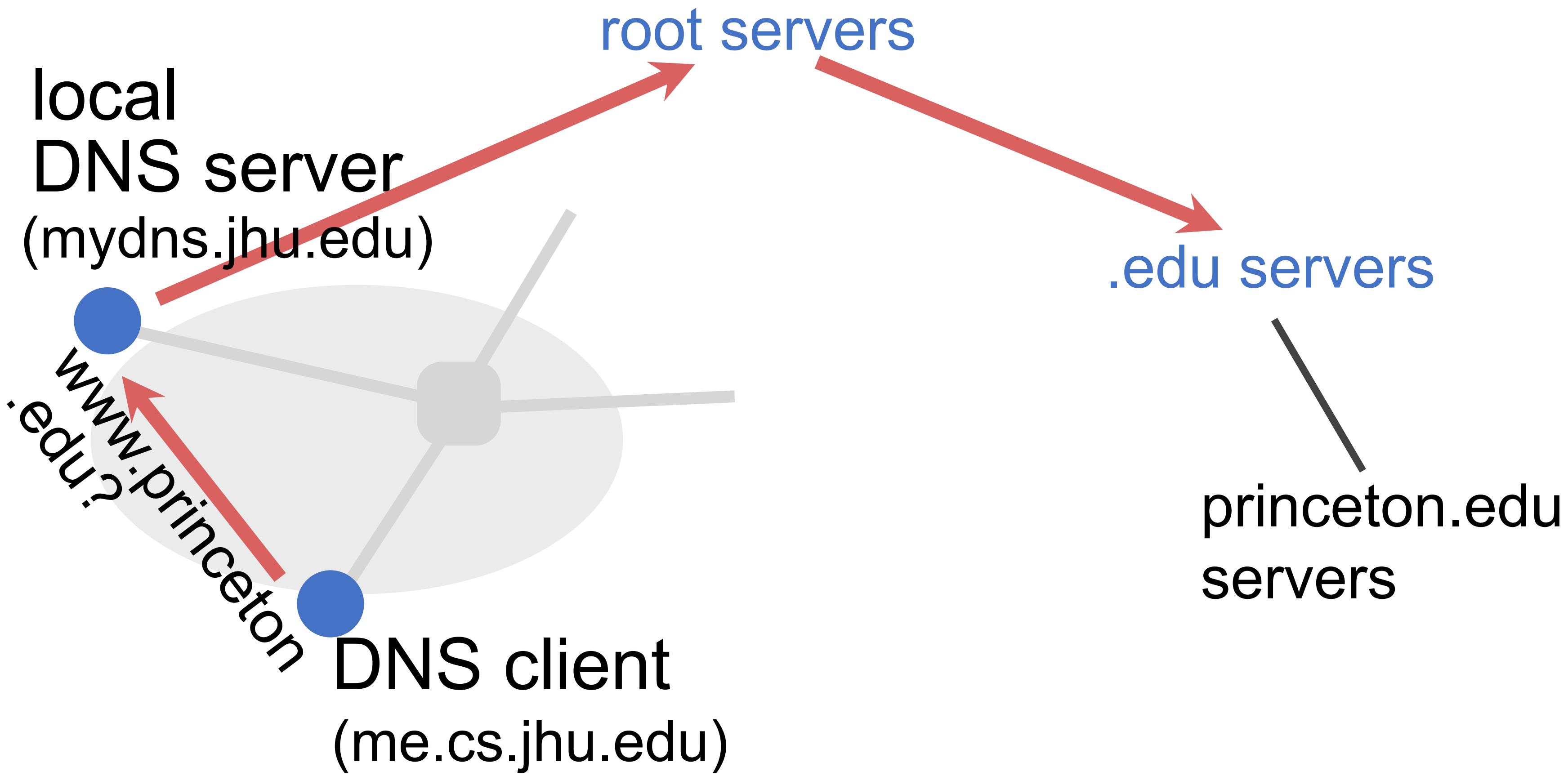
Name resolution



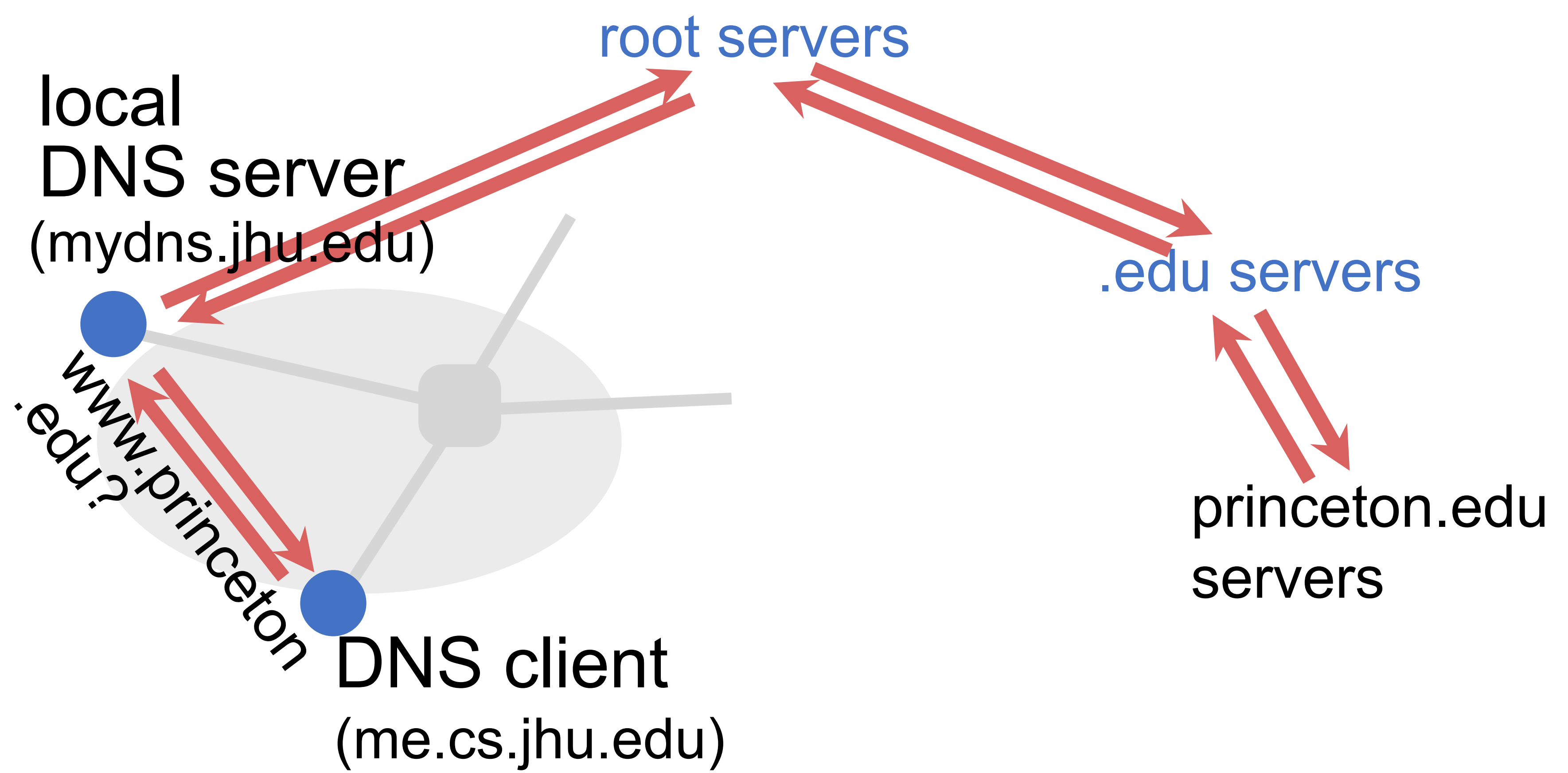
Name resolution



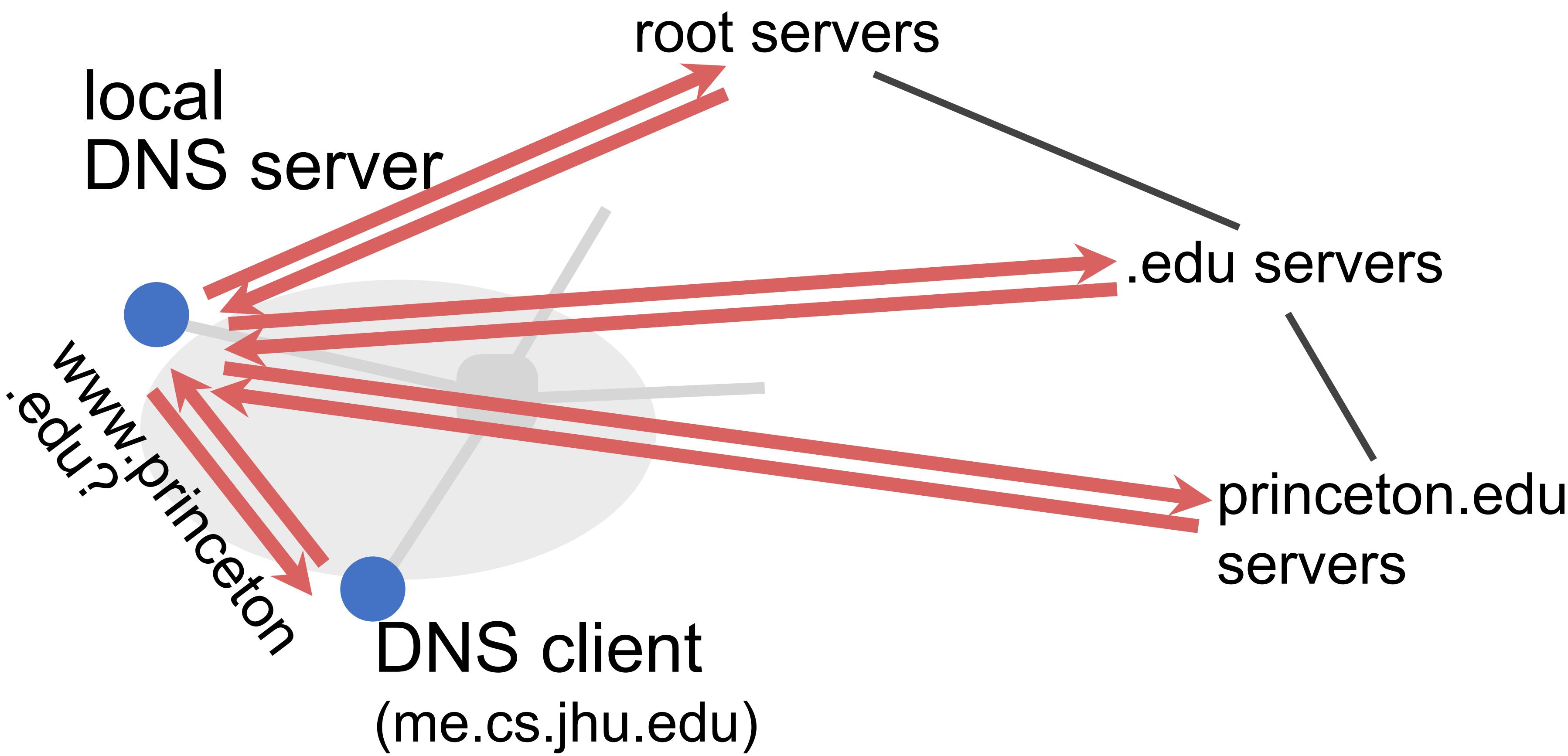
Name resolution



Name resolution: Recursive



Name resolution: Iterative



Two ways to resolve a name

- **Recursive name resolution**
  - Ask server to do it for you
- **Iterative name resolution**
  - Ask server who to ask next
- **The iterative example we saw is a mix of both!**

DNS protocol

- **Query and Reply messages; both with the same message format**
  - Header: identifier, flags, etc.
  - Plus resource records
  - See text for details
- **Client–server interaction on UDP Port 53**
  - Spec supports TCP too, but not always implemented

Goals: Are we there yet?

- **Uniqueness: No naming conflicts**
- **Scalable**
- **Distributed, autonomous administration**
- **Highly available?**

Reliability

- **Replicated DNS servers (primary/secondary)**
  - ➢Name service available if at least one replica is up
  - ➢Queries can be load-balanced between replicas
- **Usually, UDP used for queries**
  - ➢Reliability, if needed, must be implemented on UDP
- **Try alternate servers on timeout**
  - ➢Exponential backoff when retrying same server
- **Same identifier for all queries**
  - ➢Don’t care which server responds

Goals: Are we there yet?

- **Uniqueness: No naming conflicts**
- **Scalable**
- **Distributed, autonomous administration**
- **Highly available**
- **Fast lookups?**

DNS caching

- **Performing all these queries takes time**
  - Up to 1-second latency before starting download
- **Caching can greatly reduce overhead**
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- **How DNS caching works**
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field
  - Server deletes cached entry after TTL expires

Negative caching

- **Remember things that don't work**
  - ➢Misspellings like www.cnn.comm and www.cnnn.com
  - ➢These can take a long time to fail the first time
  - ➢Good to remember that they don't work so the failure takes less time the next time around
- **Negative caching is optional**
  - ➢Not widely implemented

Important properties of DNS

- **Administrative delegation and hierarchy enables:**
  - Easy unique naming
  - “Fate sharing” for network failures
  - Reasonable trust model
  - Caching increases scalability and performance

DNS provides indirection

- **Addresses can change underneath**
  - Move www.cnn.com to 4.125.91.21
- **Name could map to multiple IP addresses**
  - Load-balancing (CDN)
  - Reducing latency by picking nearby servers (CDN)
- **Multiple names for the same address**
  - E.g., many services (mail, www) on same machine
  - E.g., aliases like www.cnn.com and cnn.com
- **This flexibility applies only within domain!**

Summary

- **CDNs improve web performance**
  - Via replication and caching
  - Good server selection
- **DNS allows us to go to webpages without having to memorize IP addresses**
  - Allows a level of indirection that enables many functionalities including CDN server selection

- **Exercise and lab session this Wednesday**
- **Assignment 1 is due this Friday**

Thanks!
Q&A