

CS4450

Computer Networks: Architecture and Protocols

Lecture 18 DNS and HTTP

Spring 2018
Rachit Agarwal



What is DNS?

- **User has name of entity she/he wants to access**
 - E.g., www.cnn.com
 - Content, host, etc.
- **However, Internet routes and forwards requests based on IP addresses**
 - Need to convert name (e.g., www.cnn.com) to an IP address
- **Domain Name System (DNS)**
 - **Provides the mapping from name to IP address**
 - User asks DNS: what is the IP address for www.cnn.com
 - DNS responds: 157.166.255.18

Correctness Requirements

- **Addresses can change underneath**
 - Move www.cnn.com to 4.125.91.21
 - Humans/Applications should be unaffected
- **Name could map to multiple IP addresses**
 - www.cnn.com to multiple replicas to the Web site
 - To enable “load balancing” or reduced latency
 - Replicas may see different load (eg, due to geographic location)
 - Some replicas may be closer to the user
- **Multiple names for the same address**
 - E.g., www.cnn.com and cnn.com should map to same IP addresses

Goals and Approach

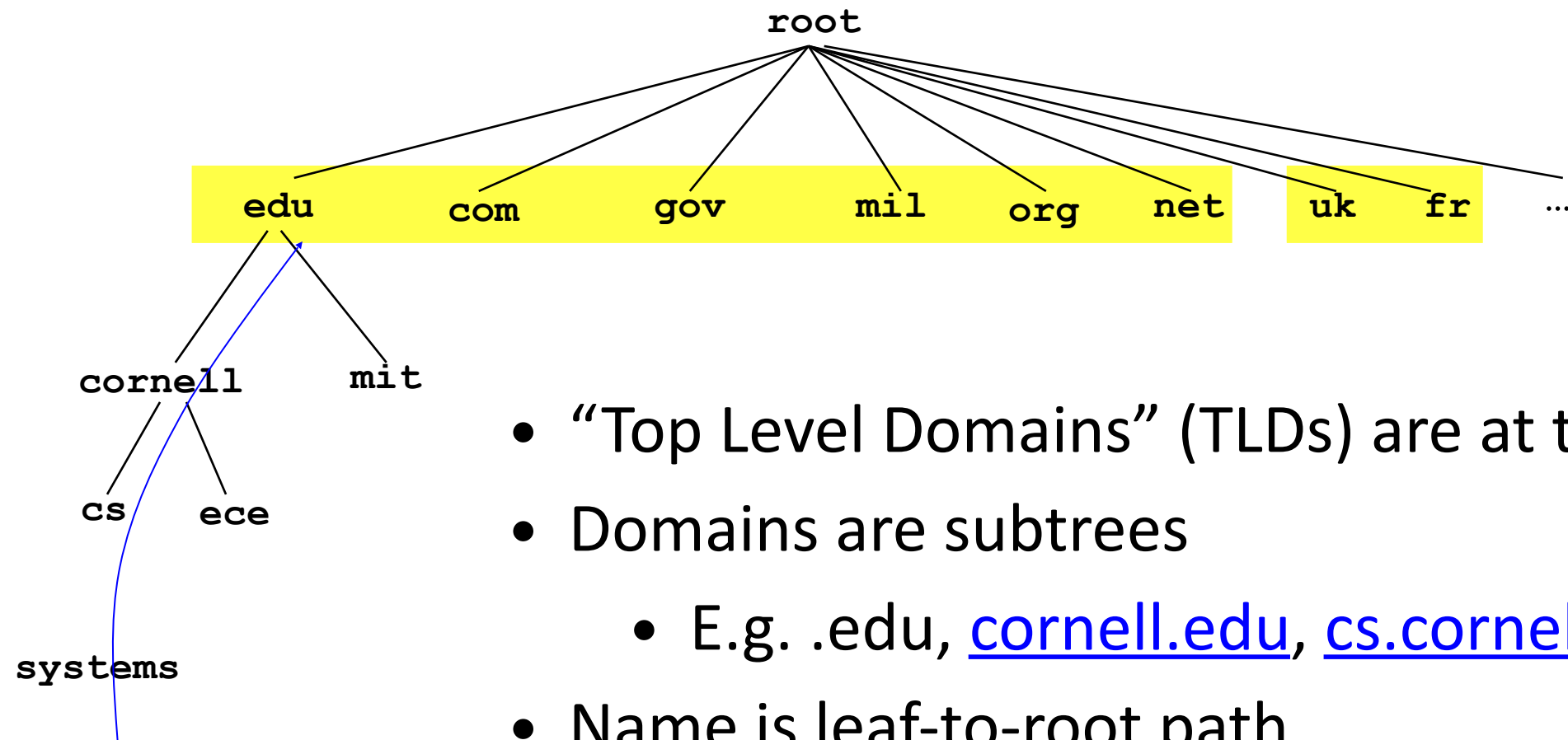
- **Goals**

- Correctness (from previous slide)
- Scaling (names, users, updates, etc.)
- Ease of management (uniqueness of names, etc.)
- Availability and consistency
- Fast lookups

- **Approach: Three intertwined hierarchies**

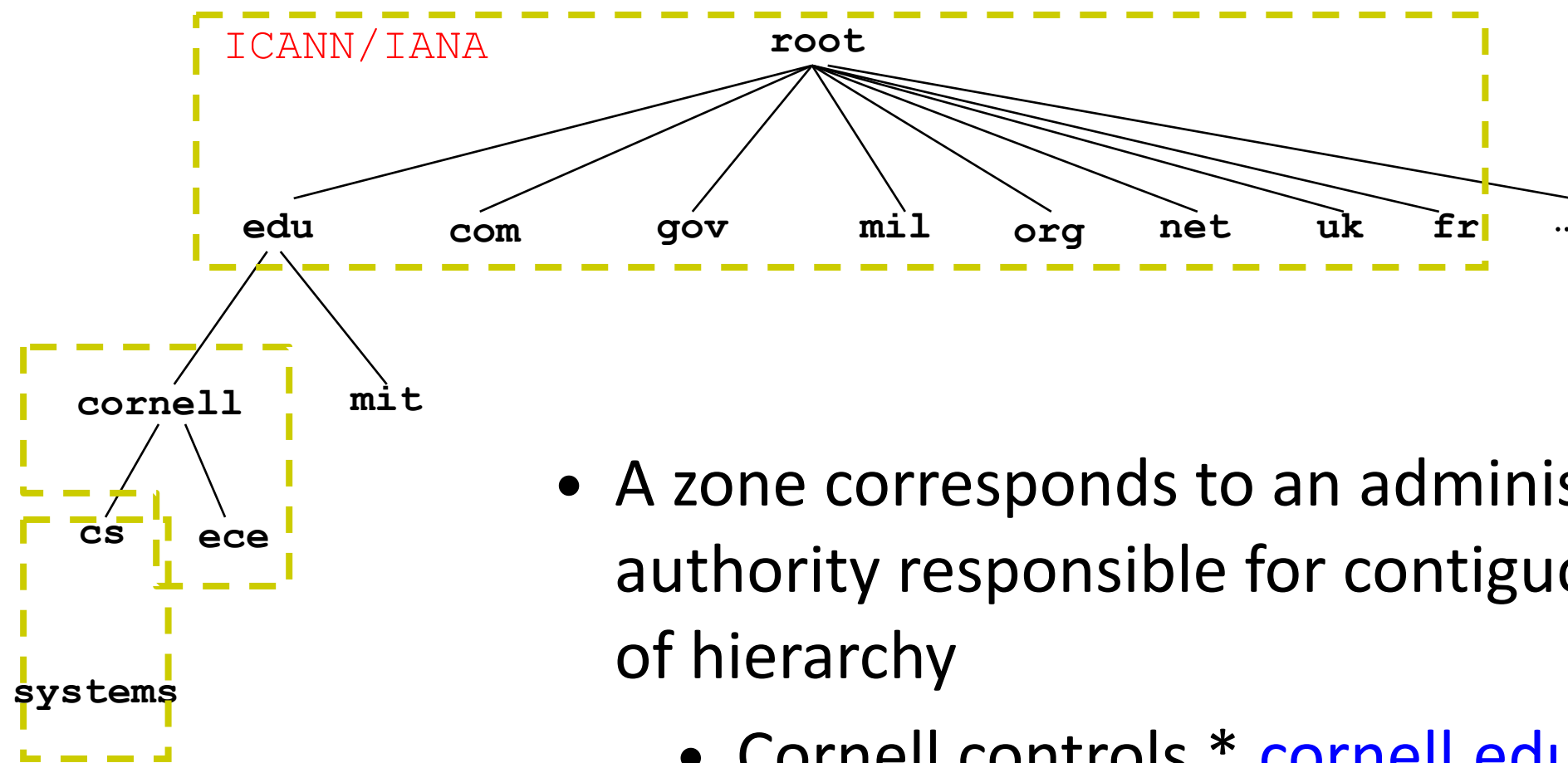
- Hierarchical Namespace: exploit structure in names
- Hierarchical Administration: hierarchy of authority over names
- Hierarchical Infrastructure: hierarchy of DNS servers

Hierarchical Namespace



- “Top Level Domains” (TLDs) are at the top
- Domains are subtrees
 - E.g. .edu, [cornell.edu](#), [cs.cornell.edu](#)
- Name is leaf-to-root path
 - [systems.cs.cornell.edu](#)

Hierarchical Administration



- A zone corresponds to an administrative authority responsible for contiguous portion of hierarchy
 - Cornell controls *.[cornell.edu](https://www.cornell.edu)
 - CS controls *.[cs.cornell.edu](https://www.cs.cornell.edu)
- Name collisions trivially avoided
 - Each domain can ensure this locally

Hierarchical Infrastructure

- Top of hierarchy: root
 - Location hardwired into other servers
- Next level: Top Level Domain (TLD) servers
 - .com, .edu, etc.
- Bottom level: Authoritative DNS servers
 - Actually do the mapping
 - Can be maintained locally or by a service provider

Per Domain Availability

- DNS Servers are **replicated**
 - Primary and secondary name servers are required
 - Name service available if at least one replica is up
 - Queries can be load-balanced among replicas
- Try alternate servers on timeout
 - **Exponential backoff** when retrying the same server

Who Knows What?

- Every server knows address of root name server
- Root servers know the address of all TLD servers
- Every node knows the address of all children
- An ***authoritative*** DNS server stores name-to-address mappings (“resource records”) for all DNS names in the domain that it has authority for
- Therefore, each server:
 - Stores only a subset of the total DNS database (scalable!)
 - Can discover server(s) for any portion of the hierarchy

Benefits of This Approach

- Scalable in names, updates, lookups, users
- Highly available: domains replicate independently
- Extensible: can add TLDs just by changing root db
- Autonomous administration:
 - Each domain manages own names and servers
 - And can further delegate
 - Easily ensures uniqueness of names
 - And consistency of databases

DNS Records (details)

- 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 (details continued)

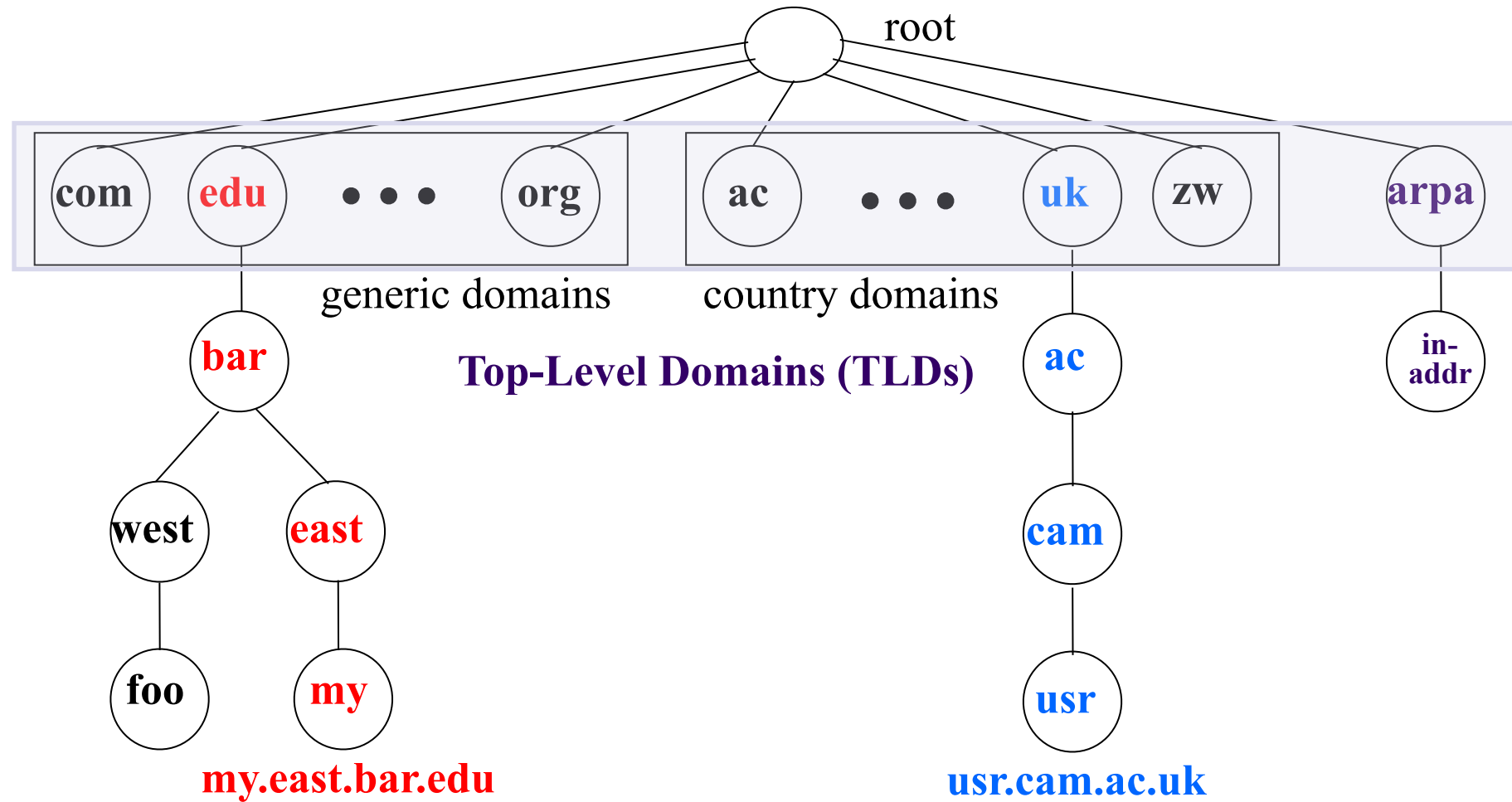
- Type = MXL (-> Main eXchanger)
 - name = domain in email address
 - value = name(s) of main server(s)
- Type = CNAME: (-> Canonical NAME)
 - Name = alias
 - Value is “canonical” name
- Type = PTR: (-> Pointer)
 - name is reversed IP
 - value is corresponding hostname

Inserting Resource Records into DNS

- Example: you just created company “FooBar”
- You get a block of IP addresses from your ISP
 - E.g., 212.44.9.128/25
- Register foobar.com at registrar (e.g., 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

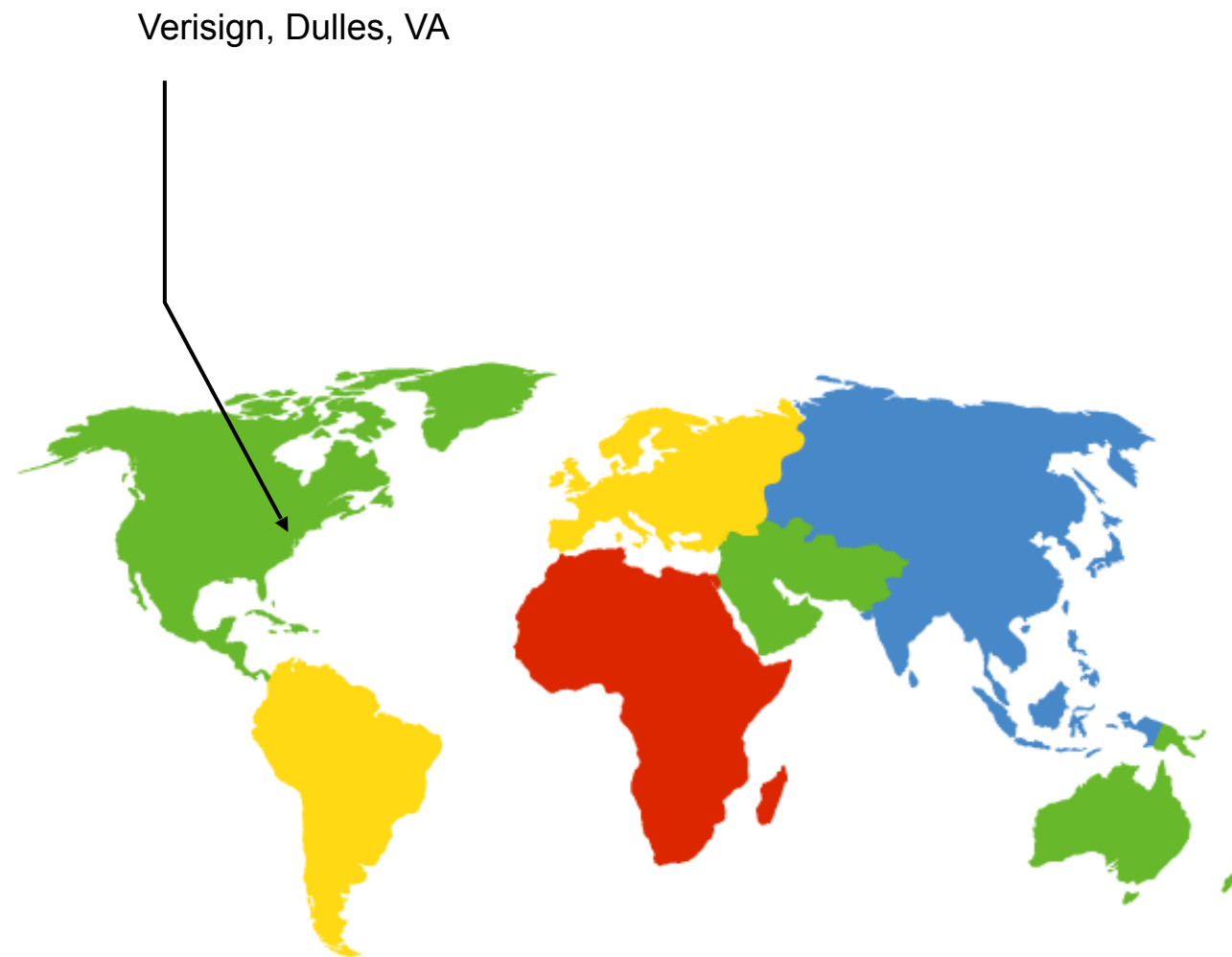
Distributed Hierarchical Database

Everything scales but the root!



DNS Root

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



DNS Root Servers

- 13 root servers (see <http://www.root-servers.org/>)
 - Labeled A through M
- How can we seamlessly scale this further?

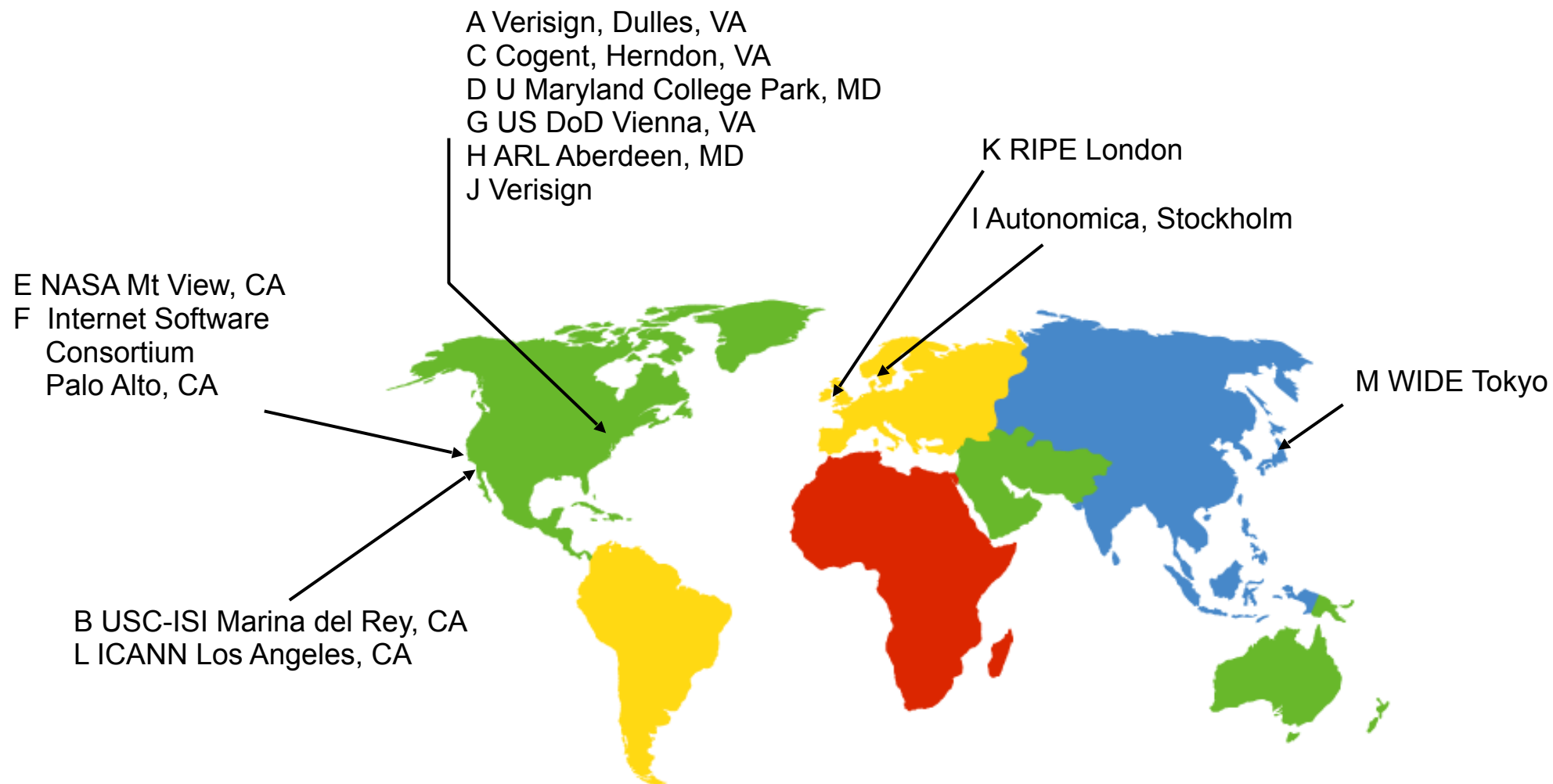


Anycast

- Routing finds shortest paths to destination
- If several locations are given the same address:
 - Network will deliver the packet to closest location with that address
- This is called “**anycast**”
 - No modification of routing is needed for this....
- Allows for seamless replication of resources
 - Any problems with this approach?

DNS Root Servers

- 13 root servers (see <http://www.root-servers.org/>)
 - Labeled A through M
- Replication via any-casting (localized routing for addresses)

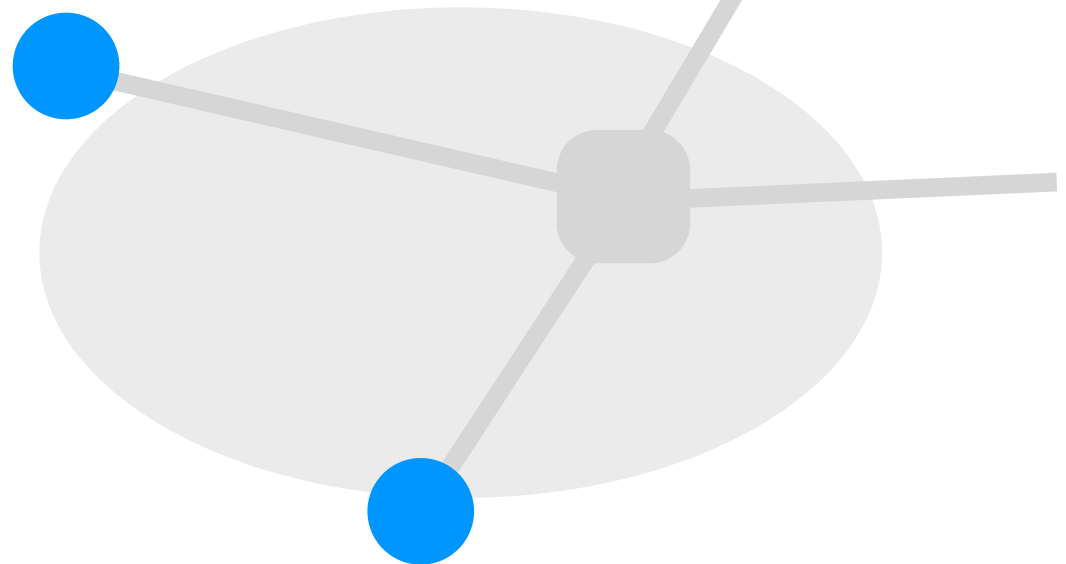


Using DNS

- Two components
 - Local DNS servers
 - Resolver software on hosts
- Local DNS server (“default name server”)
 - Usually near the end hosts that use it
 - Local hosts configured with local server (e.g, /etc/resolv.conf) or learn server via DHCP
- Client application
 - Obtain DNS name (e.g., from the URL)
 - Do **gethostbyname()** to trigger resolver code
 - Which then sends request to local DNS server

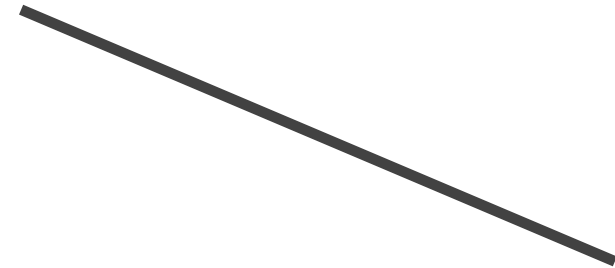
**Local DNS
server**

(mydns.cornell.edu)



DNS client
(me.cs.cornell.edu)

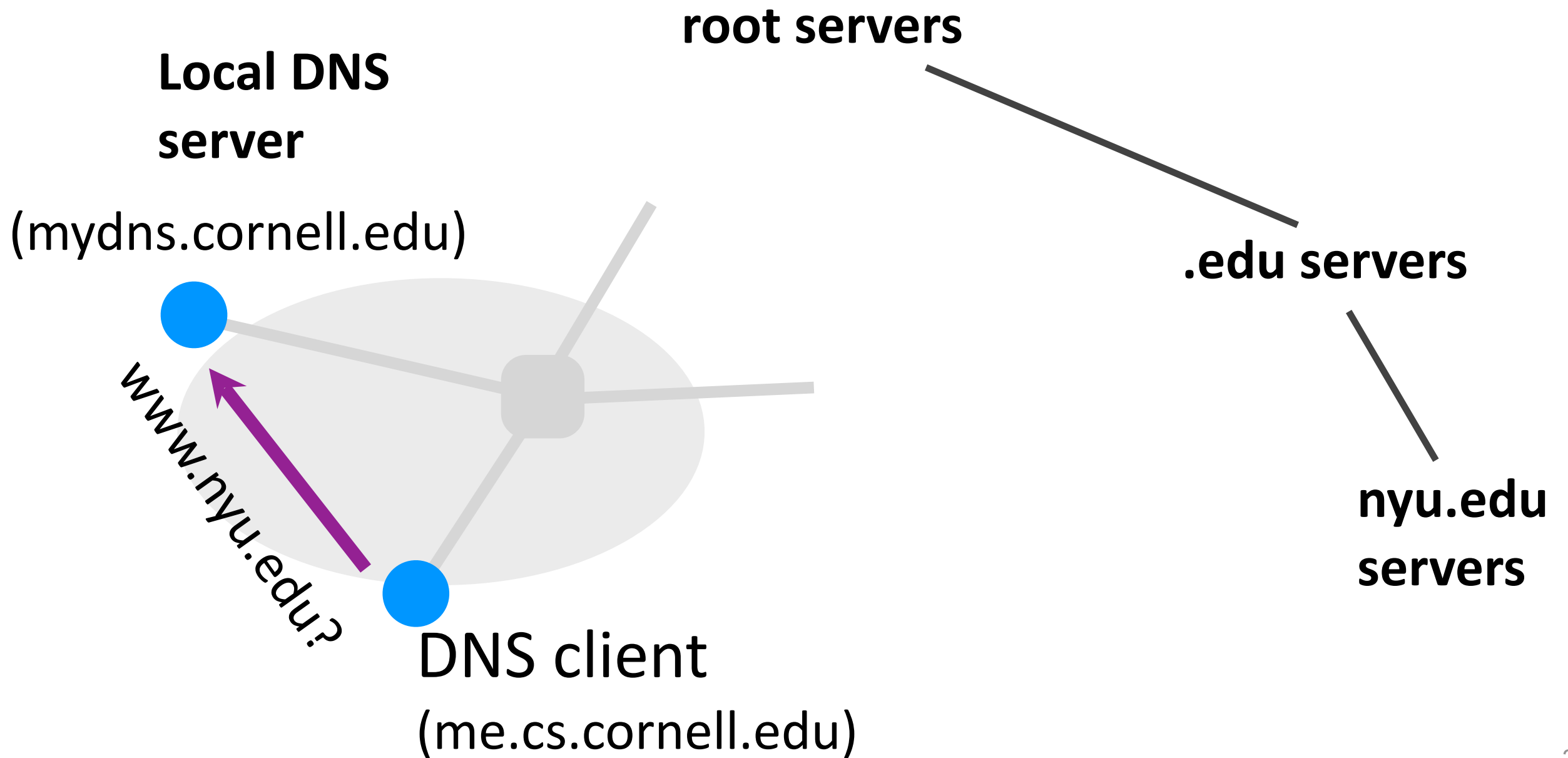
root servers

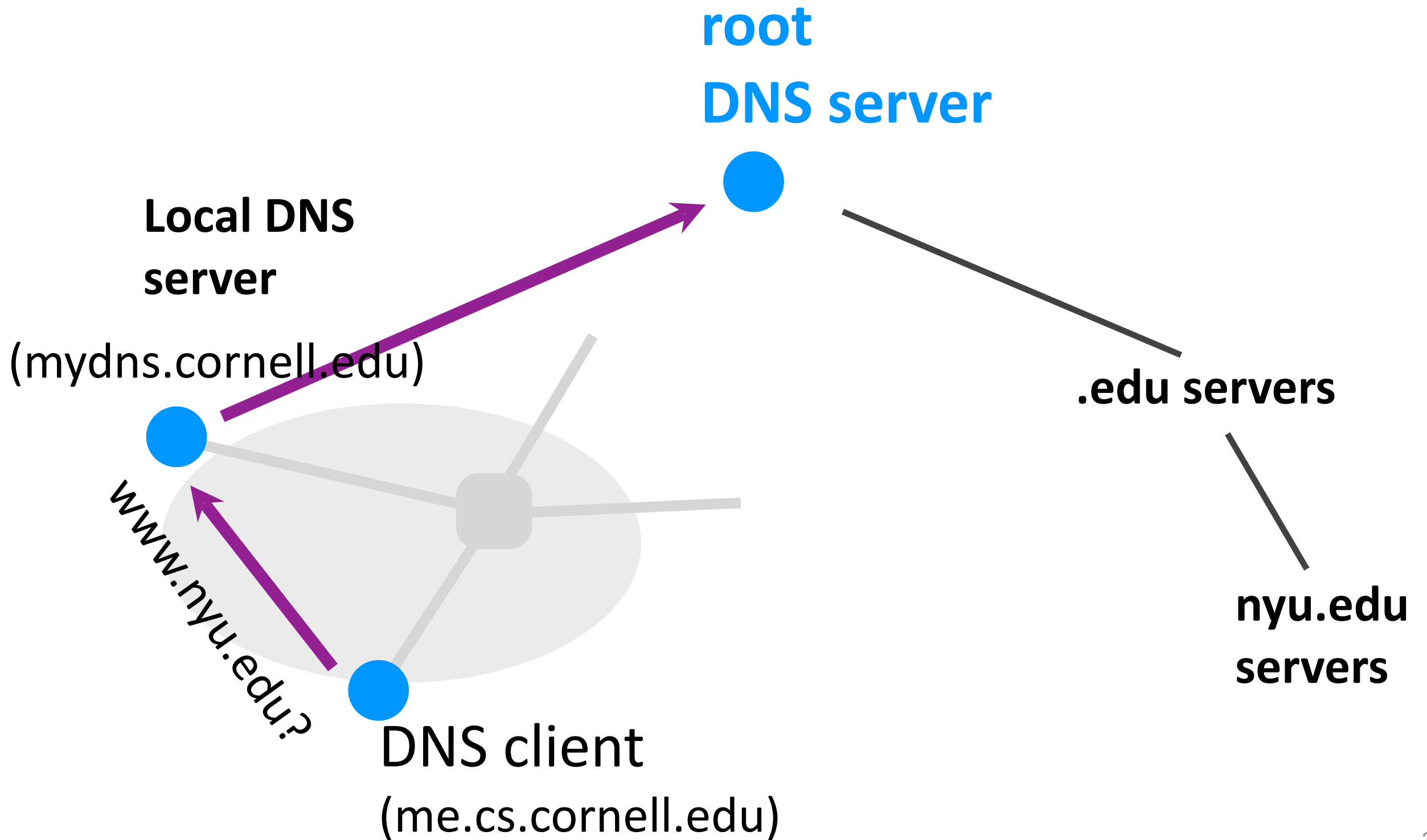


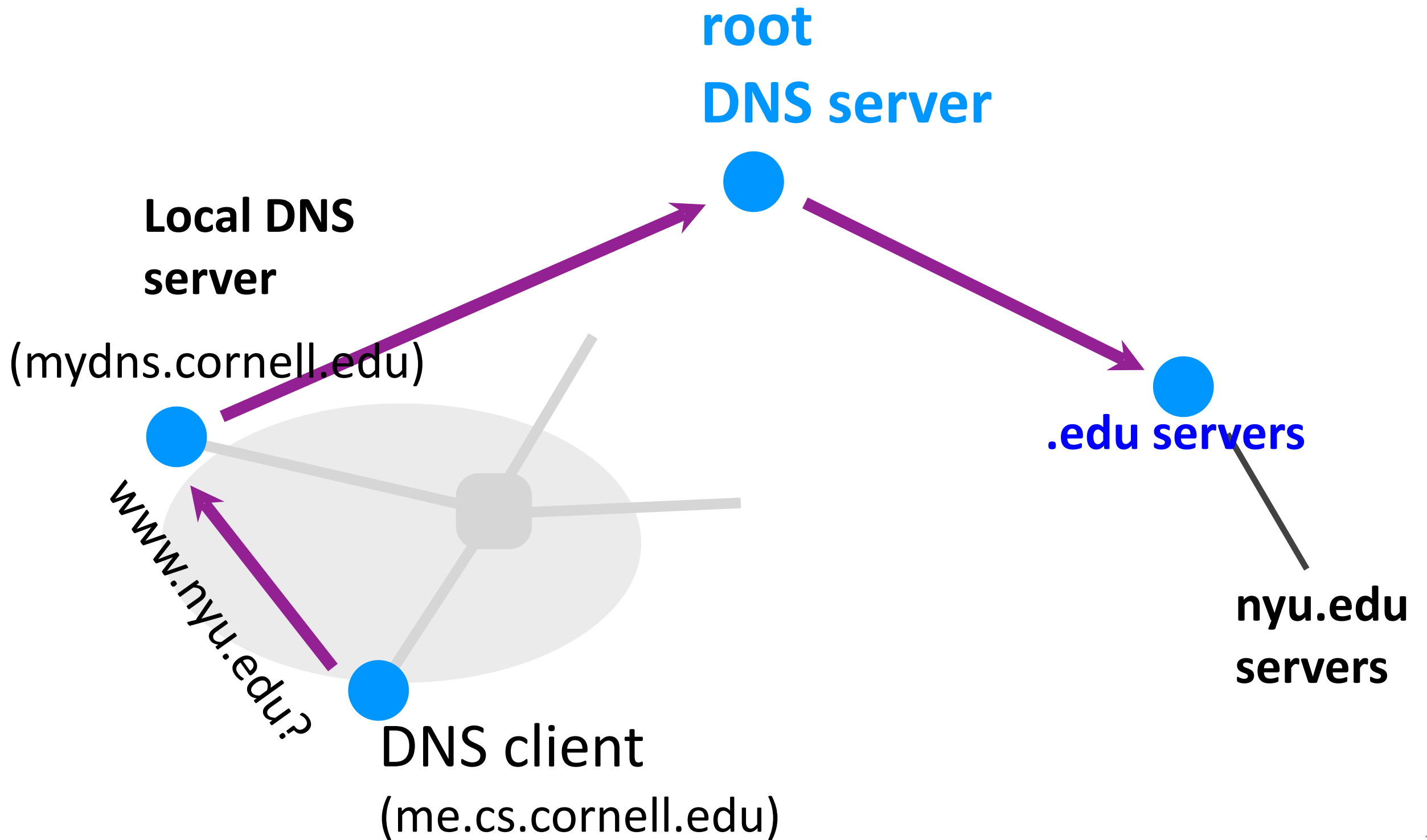
.edu servers



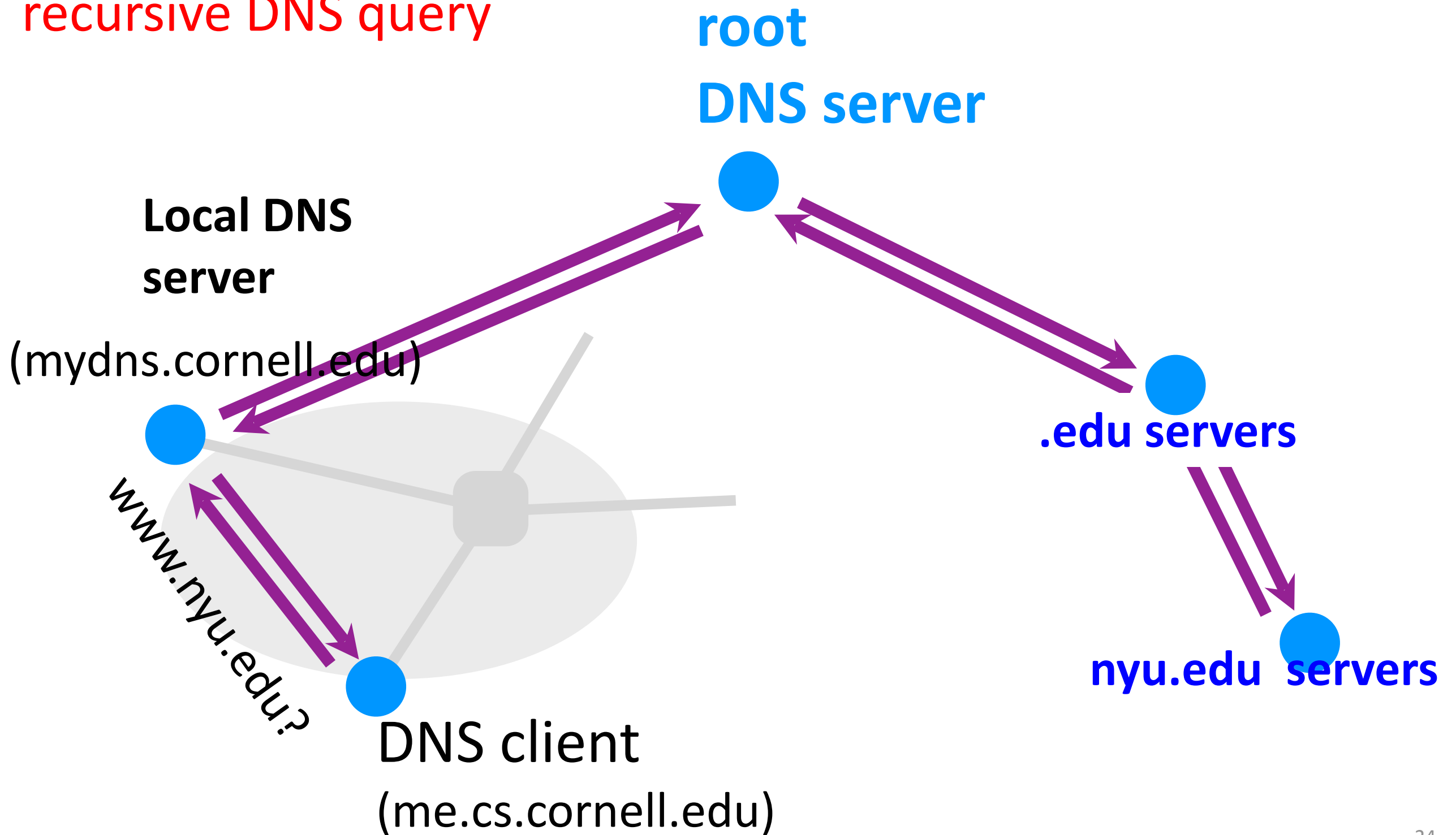
**nyu.edu
servers**





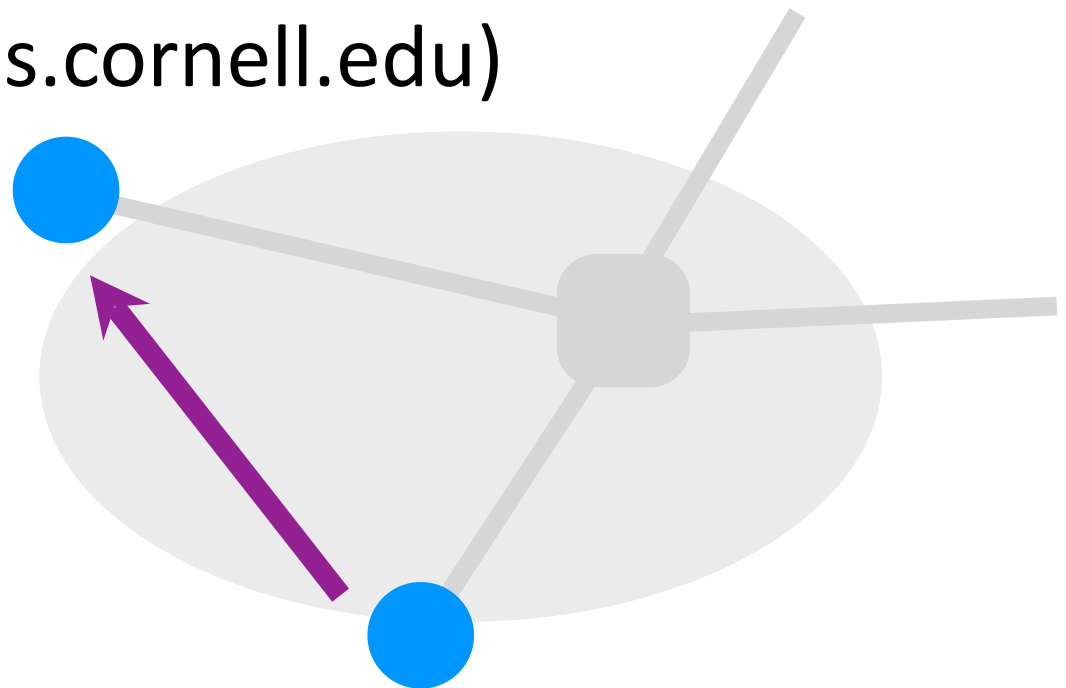


recursive DNS query



**Local DNS
server**

(mydns.cornell.edu)



DNS client
(me.cs.cornell.edu)

**root
DNS server**



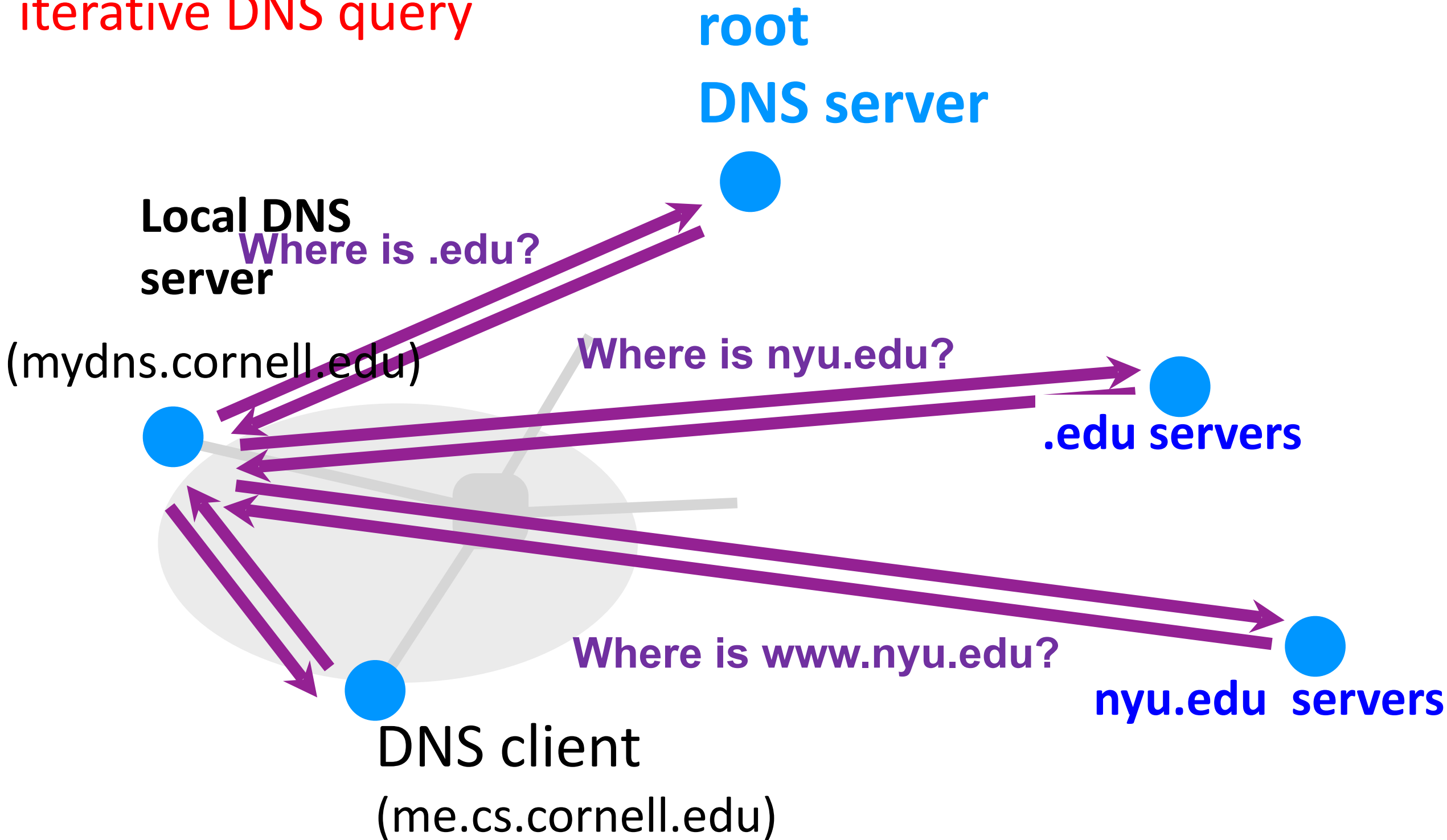
.edu servers



nyu.edu servers



iterative DNS query



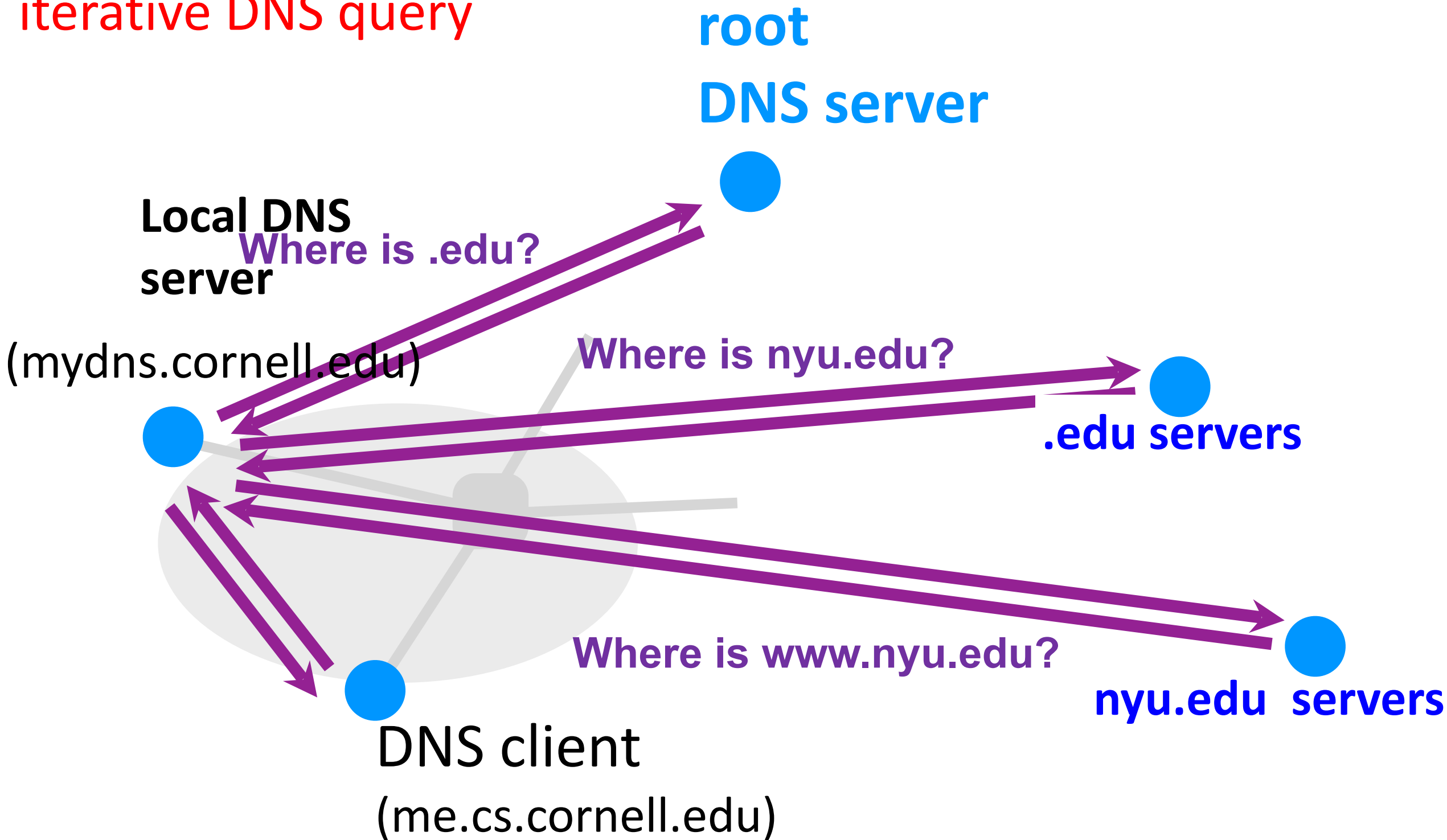
DNS Protocol

- Query and Reply messages
 - Both with the same message format
 - *see text for details*
- Client-Server interaction on UDP Port 53
 - Spec supports TCP too, but not always implemented
 - Reliability via repeating requests on timeout
- Resolution is almost always “iterative”

Goals

- Scaling (names, users, updates, etc.)
 - Yes
- Ease of management (uniqueness of names, etc.)
 - Yes
- Availability and consistency
 - Yes
- Fast lookups
 - ??

iterative DNS query



DNS Caching

- 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
- Why caching is effective
 - The top-level servers very rarely change
 - Popular sites visited often -> local DNS server often has the information cached

Questions?

The Web

The Web – Precursor

- 1967, Ted Nelson, Xanadu:
 - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
 - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term “Hypertext”
 - Influenced research community
 - Who then missed the web.....



Ted Nelson

The Web – Precursor

- Physicist trying to solve real problem
 - Distributed access to data
- World Wide Web (WWW): a distributed database of “pages” linked through **Hypertext Transport Protocol (HTTP)**
 - First HTTP implementation - 1990
 - Tim Berners-Lee at CERN
 - HTTP/0.9 – 1991
 - Simple GET command for the Web
 - HTTP/1.0 –1992
 - Client/Server information, simple caching
 - HTTP/1.1 - 1996



Tim Berners-Lee

Web Components

- Infrastructure:
 - Clients
 - Servers
 - Proxies
- Content:
 - Individual objects (files, etc.)
 - Web sites (coherent collection of objects)
- Implementation
 - URL: naming content
 - HTTP: protocol for exchanging content

URL Syntax

protocol://hostname[:port]/directorypath/resource

protocol http, ftp, https, smtp, rtsp, *etc.*

hostname DNS name, IP address

port Defaults to protocol's standard port
e.g. http: 80 https: 443

directory path Hierarchical, reflecting file system

resource Identifies the desired resource

Can also extend to program executions:

```
http://us.f413.mail.yahoo.com/ym/ShowLetter?
box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289
957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&he
ad=b
```

Web and DNS

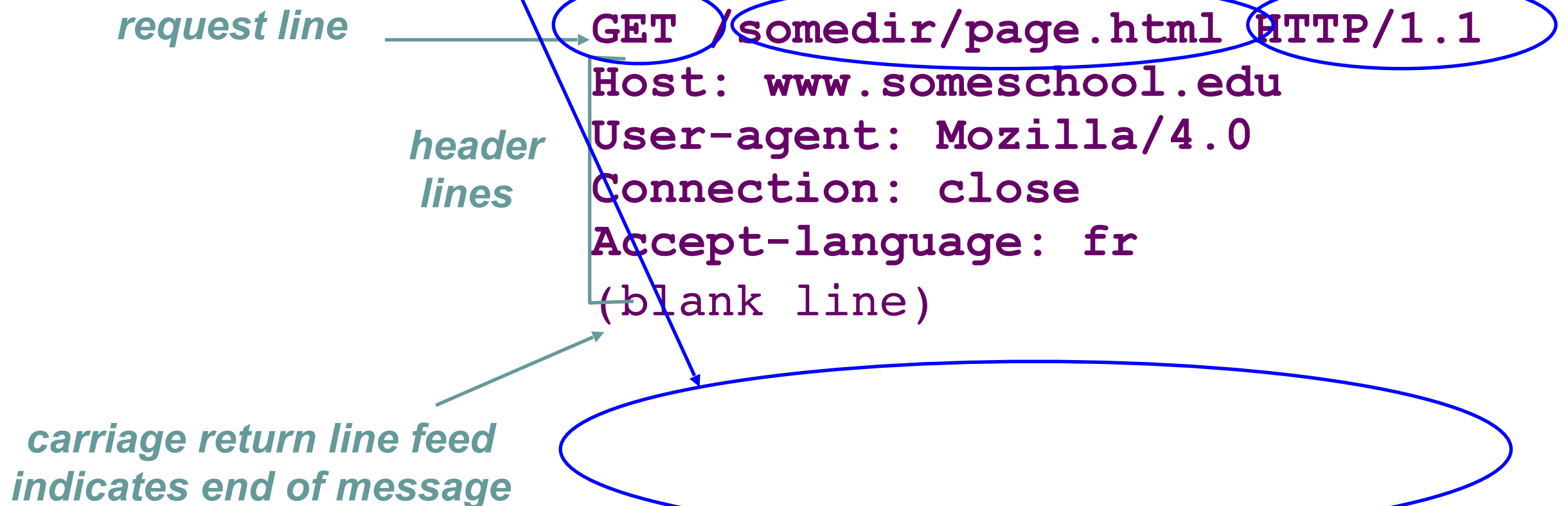
- URLs use hostnames
- Thus, content names are tied to specific hosts
- Why is this a problem?
- Makes persistence of names problematic...

Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
 - Server is “always on” and “well known”
 - Clients initiate contact to server
- Synchronous request/reply protocol
 - Runs on top of transport layer, Port 80
- Stateless
- ASCII format

HTTP request message

- Request line: method, resource, and protocol version
- Request headers: provide information or modify request
- Body: optional data (e.g., to “POST” data to the server)



HTTP response message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional data

status line

(protocol, status code, status phrase)

header lines

data

e.g., requested HTML file

HTTP/1.1 200 OK

Connection close

Date: Thu, 06 Aug 2006 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2006 ...

Content-Length: 6821

Content-Type: text/html

(blank line)

data data data data data ...

HTTP is Stateless

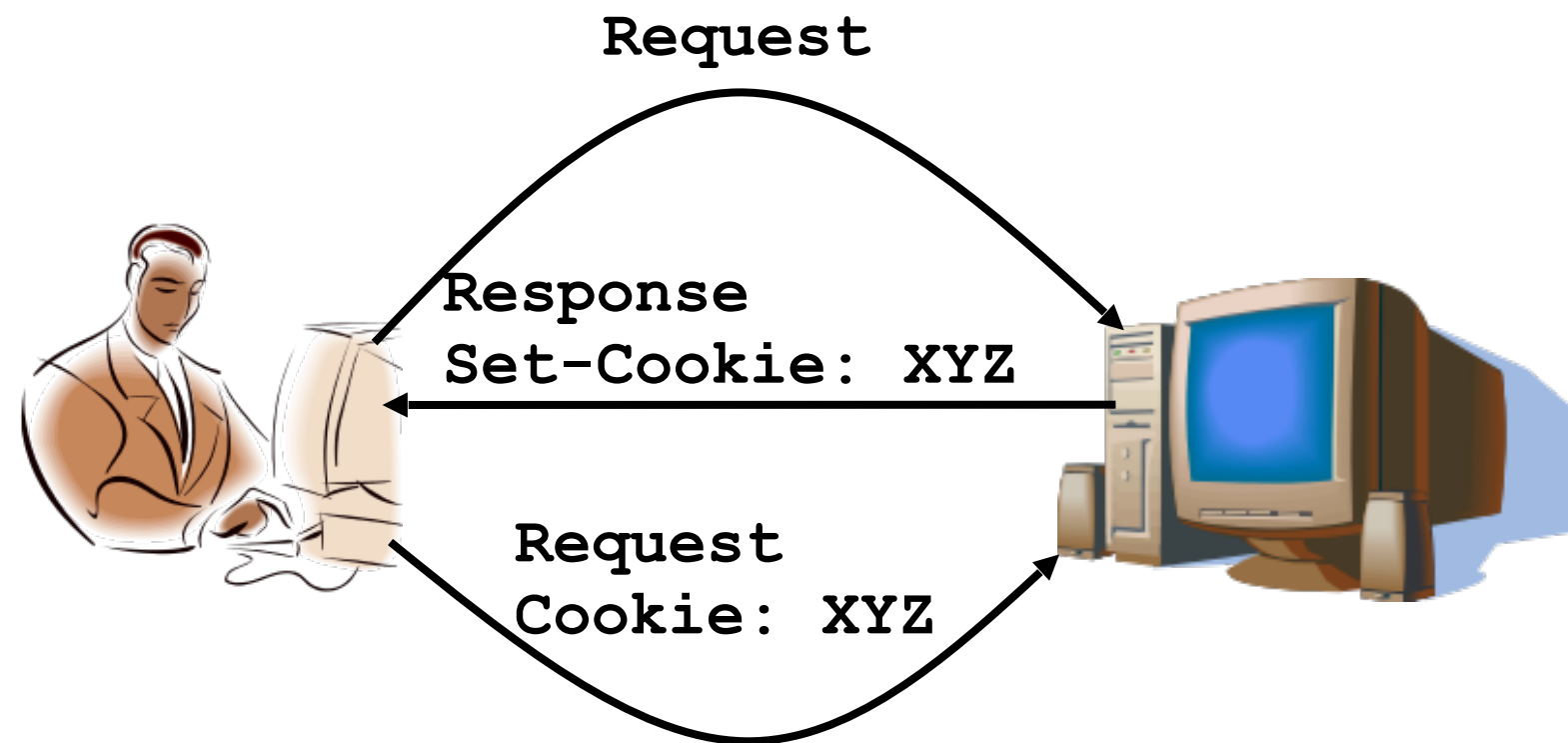
- Each request-response treated independently
 - Servers not required to retain state for HTTP
 - The application may have lots of state, but not HTTP
- **Good:** Improves scalability on the server-side
 - Failure handling is easier
 - Can handle higher rate of requests
 - Order of requests doesn't matter (to HTTP)
- **Bad:** Some applications need persistent state
 - Need to uniquely identify user or store temporary info
 - e.g., Shopping cart, user profiles, usage tracking, ...

Question

- How does a stateless protocol keep state?

State in a Stateless Protocol: Cookies

- Client-side state maintenance
 - Client stores small state on behalf of server
 - Client sends state in future requests to the server
- Can provide authentication



HTTP Performance Issues

Performance Goals

- User
 - Fast downloads
 - High availability
- Content provider
 - Happy users (hence, above)
 - Cost-effective infrastructure
- Network (secondary)
 - Avoid overload

**Caching and replication
resolve most of these issues**

HTTP Performance

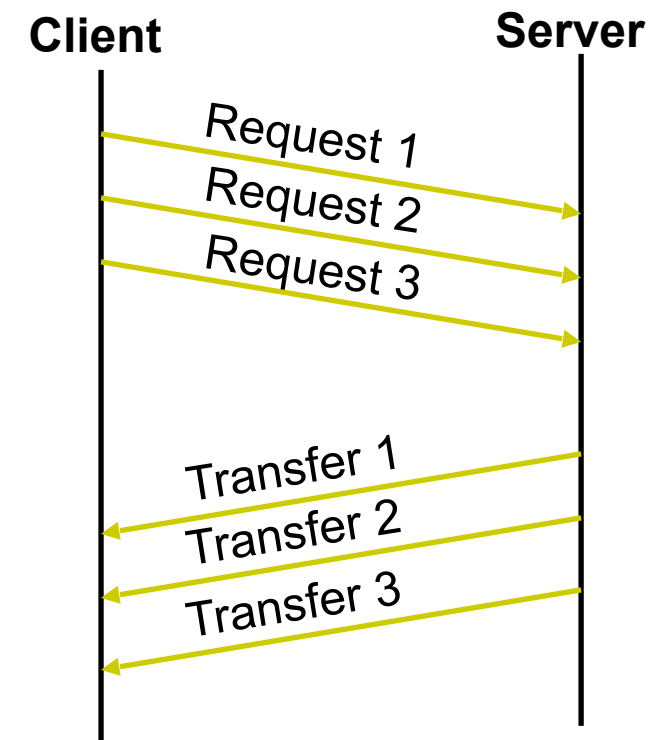
- Most Web pages have multiple objects
 - e.g., HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
 - One item at a time
- New connection per (small) object!
- Requires 2RTTs worth of latency per object

Improving HTTP Performance

- **Persistent Connections**
- Maintain connection across multiple requests
 - Including transfers subsequent to current page
 - Client or server can tear down connection
- Performance advantages:
 - Avoid overhead of connection set-up and tear-down
- Default in HTTP/1.1

Improving HTTP Performance

- **Pipelined Requests and Responses**
- Batch requests and responses to reduce the number of packets



Questions?

Improving HTTP Performance: Caching

- Why does caching work?
 - Exploits locality of reference
- How well does caching work?
 - Very well, up to a limit
- File popularity has high peak but long tail
 - Large overlap in highly popular content
 - But many unique requests
- A universal story!
 - Hit rate of cache grows logarithmically with size

Improving HTTP Performance: 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

Improving HTTP Performance: Caching - How?

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

Typical Caching Interaction

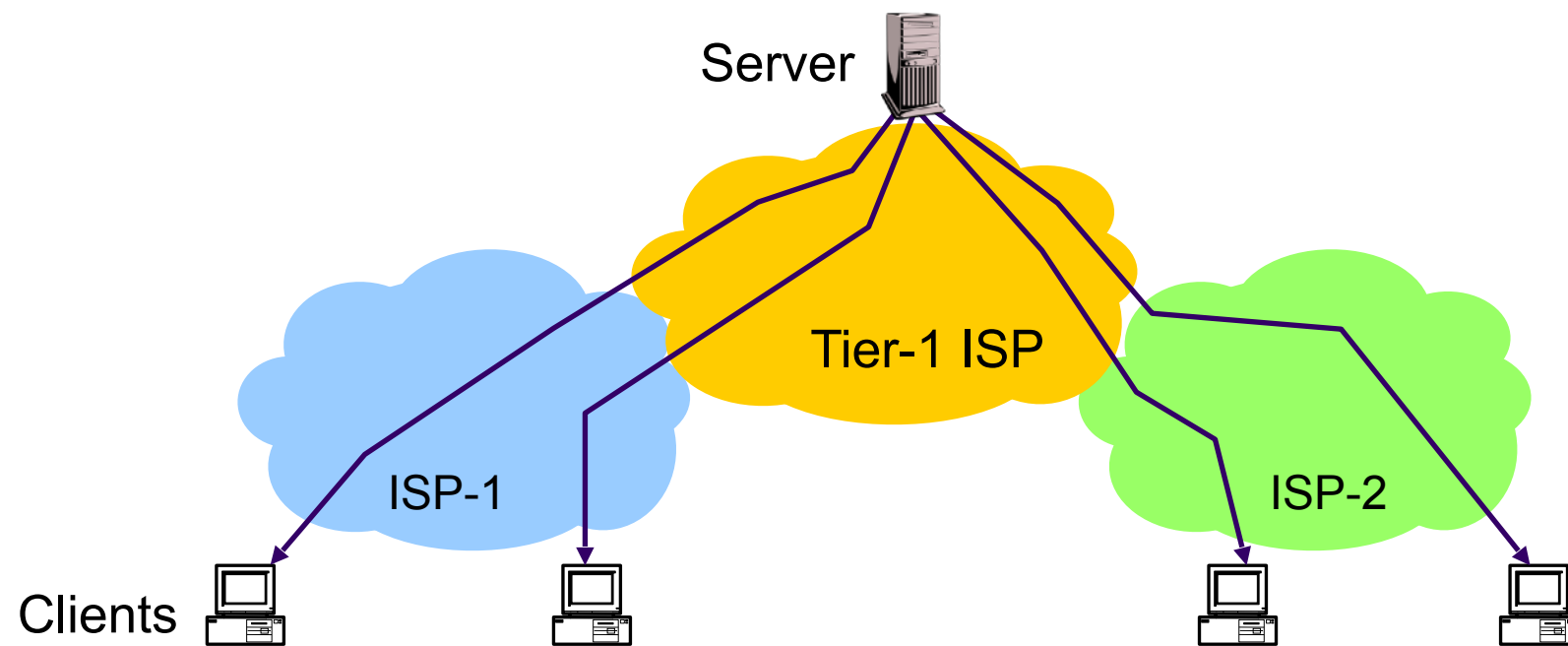
- Client issues request for object
- If it is in local client cache:
 - If within TTL, respond to client
 - If not within TTL, send if-modified-since to server
 - If server has updated copy, it sends it
 - If not, server responds saying that it doesn't
- If not in local client cache:
 - Send request to server
 - This request may pass through other caches, which use a similar algorithm

Improving HTTP Performance: Caching - Where?

- Options
 - Client
 - Forward proxies
 - Reverse proxies
 - Content Distribution Network

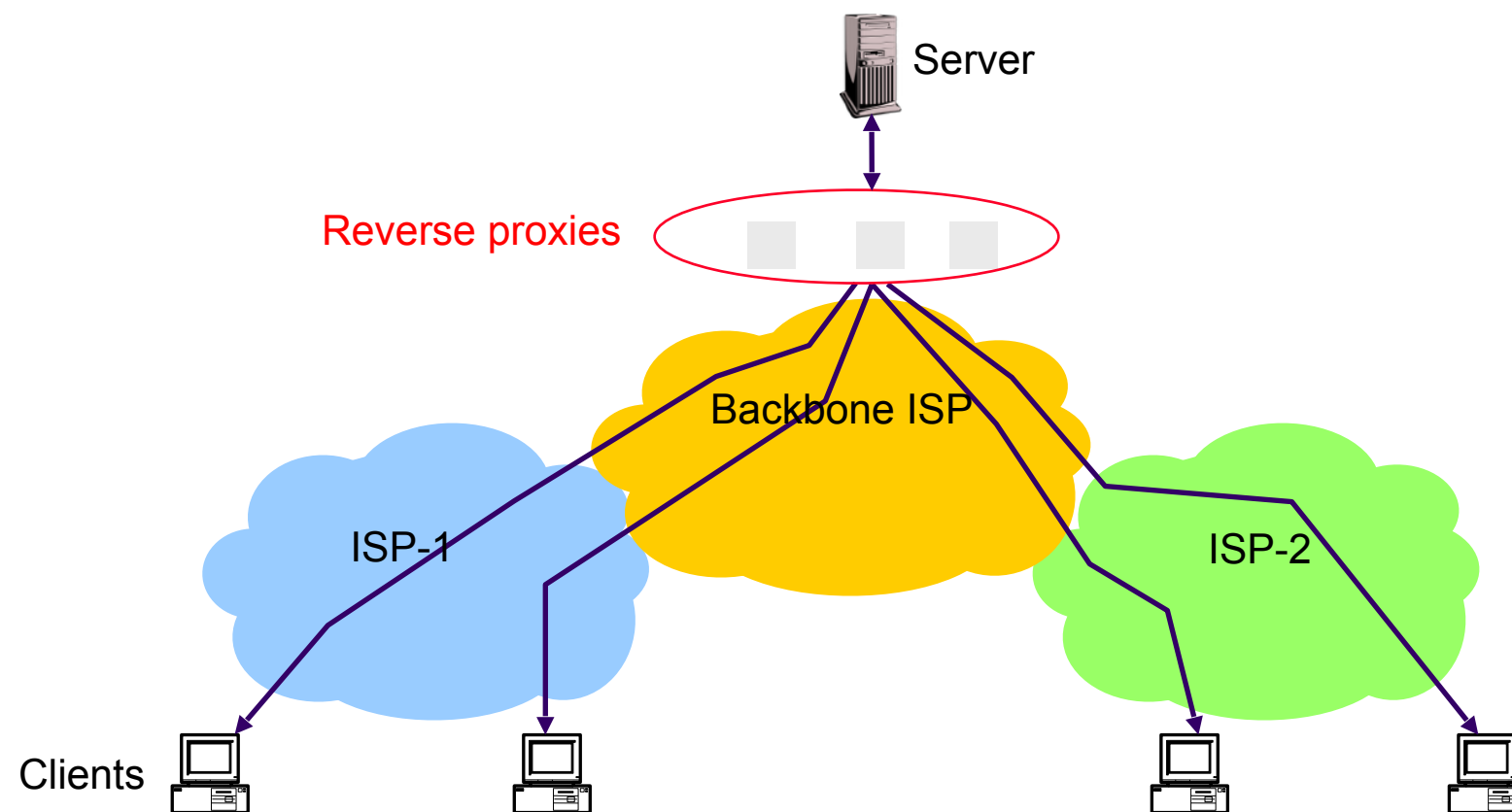
Improving HTTP Performance: Caching - Where?

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



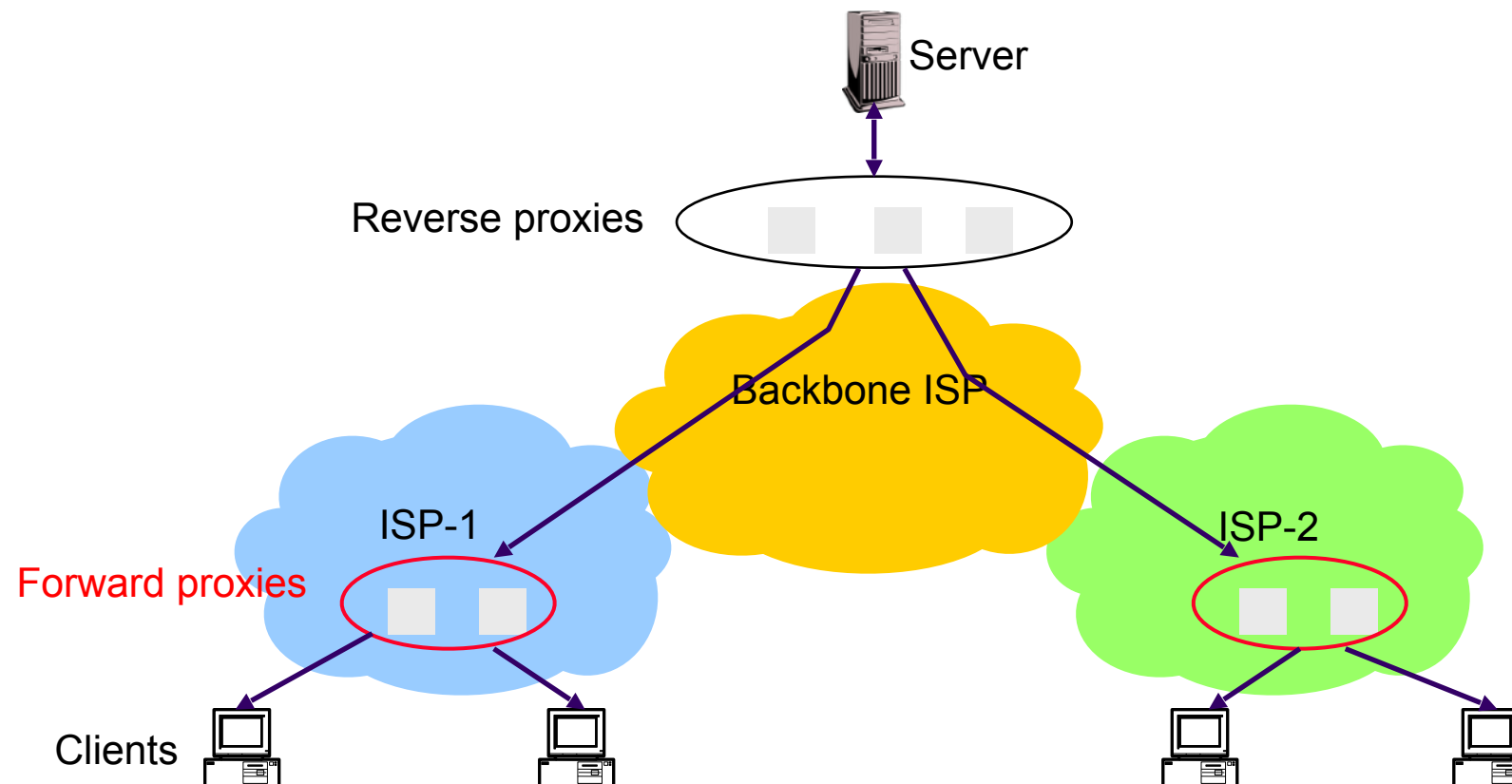
Caching with Reverse Proxies

- Cache documents close to server
 - Decrease server load
- Typically done by content provider



Caching with Forward Proxies

- Cache documents close to clients
 - Reduce network traffic and decrease latency
- Typically done by ISPs or enterprises



Improving HTTP Performance: Replication

- Replicate popular Web site across many machines
 - Spreads load on servers
 - Places content closer to clients
 - Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
 - Balance load across server replicas
 - Pair clients with nearby servers
- Common solution:
 - DNS returns different addresses based on client's geo location, server load, etc.

Content Distribution Networks

- **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 (pull) caching and (push) replication
 - **Pull:** Direct result of clients' requests
 - **Push:** Expectation of high access rate
- Also do some processing
 - Handle dynamic web pages
 - Transcoding

CDN Example — Akamai

- Akamai creates new domain names for each client
 - e.g., a128.g.akamai.net for cnn.com
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the 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...

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:
 - Web hosting companies
 - CDNs
 - Cloud infrastructure

Questions?

Backup slide: History of DNS

- Originally: per-host file hosts.txt in /etc/hosts
 - SRI (Menlo Park) kept master copy
 - Downloaded regularly
 - Flat namespaces
- As the Internet grew this system broke down
 - SRI couldn't handle the load
 - Conflicts in selecting names
 - Hosts had inaccurate copies of hosts.txt
- Domain Name System (DNS) invented to fix this
 - First server implementation done by 4 UCB students!

DNS Measurements (MIT data from 2000)

- What is being looked up?
 - ~60% requests for A records
 - ~25% for PTR records
 - ~5% for MX records
 - ~6% for ANY (wildcard) records
- How long does it take?
 - Median ~100msec (but 90th percentile ~500msec)
 - 80% have no referrals; 99.9% have fewer than four
- Query packets per lookup: ~2.4
 - But this is misleading....

DNS Measurements (MIT data from 2000)

- Does DNS give answers?
 - ~23% of lookups fail to elicit an answer!
 - ~13% of lookups result in NXDOMAIN (or similar)
 - Mostly reverse lookups
 - Only ~64% of queries are successful!
 - How come the web seems to work so well?
- ~ 63% of DNS packets in unanswered queries!
 - Failing queries are frequently retransmitted
 - 99.9% successful queries have ≤ 2 requests

Moral of the Story

- The Internet was designed to be highly resilient.
 - No matter what goes wrong, it tries to recover
- **In a highly resilient system, many things can be going wrong without you noticing it!**

DNS Measurements (MIT data from 2000)

- Top 10% of names accounted for ~70% of lookups
 - Caching should really help!
- 9% of lookups are unique
 - Cache hit rate can never exceed 91%
- Cache hit rates ~ 75%
 - But caching for more than 10 hosts doesn't add much

A Common Pattern.....

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
 - Large fraction of total metric in the top 10%
 - Sizable fraction ($\sim 10\%$) of total fraction in low values
- In an exponential distribution
 - Large fraction is in top 10%
 - But low values have very little of overall total
- Lesson: in networking, have to pay attention to both ends of distribution (high peak and long tail)
 - Here, caching helps, but not a panacea

Why not name content directly?

- How do you know where to send the request?
- How do you scale?
- How do you trust the response?
 - Requesting host
 - Network
- How would you design it?

Scorecard: Getting n Small Objects

- **Time dominated by latency**
- One at a time: $\sim 2n$ RTT
- M concurrent: $\sim 2\lceil n/m \rceil$ RTT
- Persistent: $\sim (n+1)$ RTT
- Pipelined: ~ 2 RTT
- Pipelined/Persistent: ~ 2 RTT first time, RTT later

Scorecard: Getting n Large Objects

- **Time dominated by bandwidth**
- One at a time: $\sim nF/B$
- M concurrent: it depends
 - If more flows get no additional bandwidth: $\sim nF/B$
 - If shared with large population of users: $\sim [n/m] F/B$
 - Where each TCP connection gets the same bandwidth
- Pipelined and/or Persistent: $\sim nF/B$
 - The only thing that helps is getting more bandwidth