Project 3a is out!

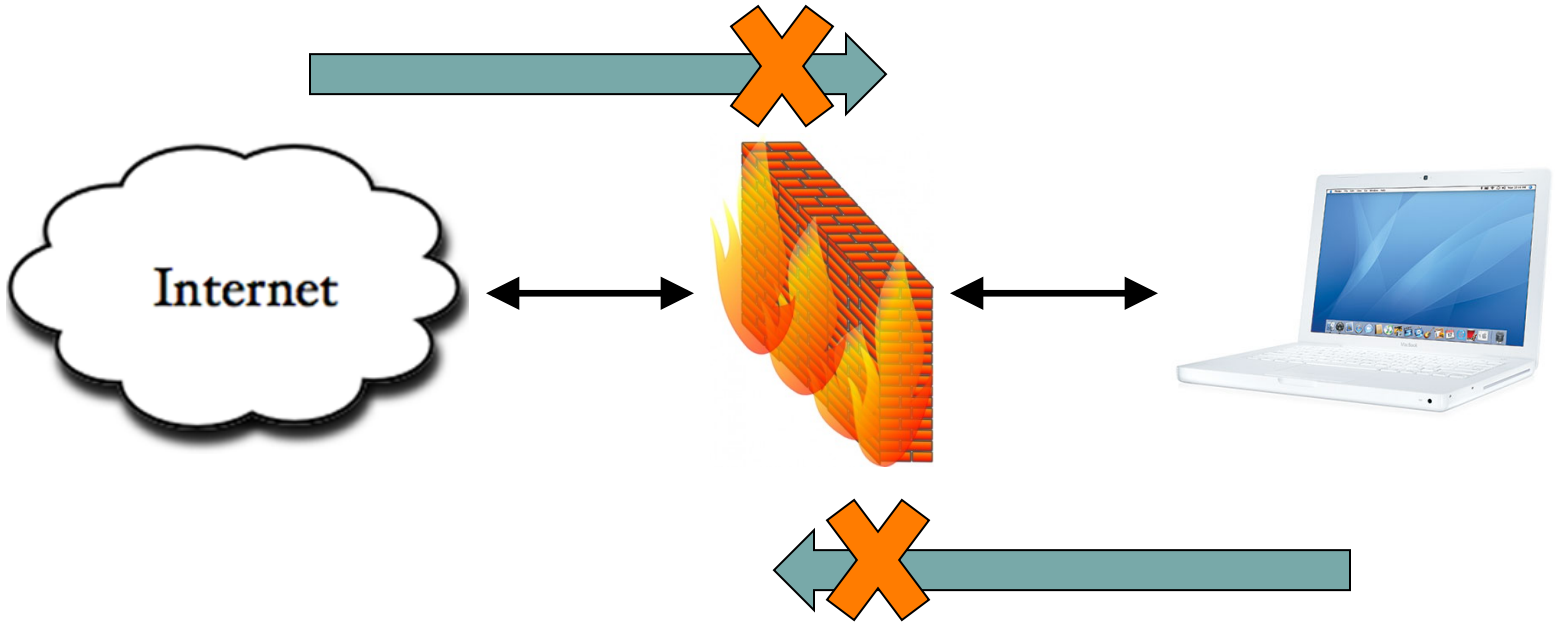
Goal: implement a basic network firewall

- We give you the VM & framework.
- You implement the firewall logic.

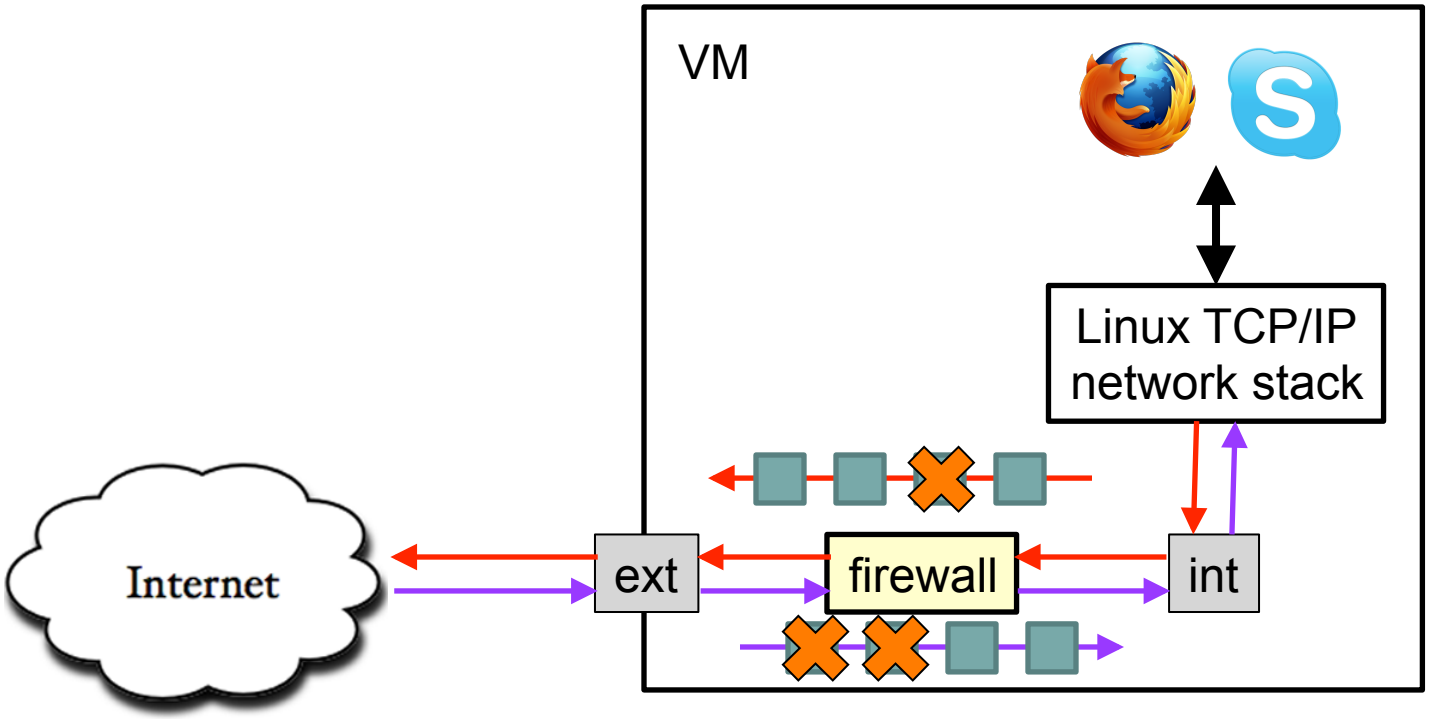
Due: Nov 17 @ noon

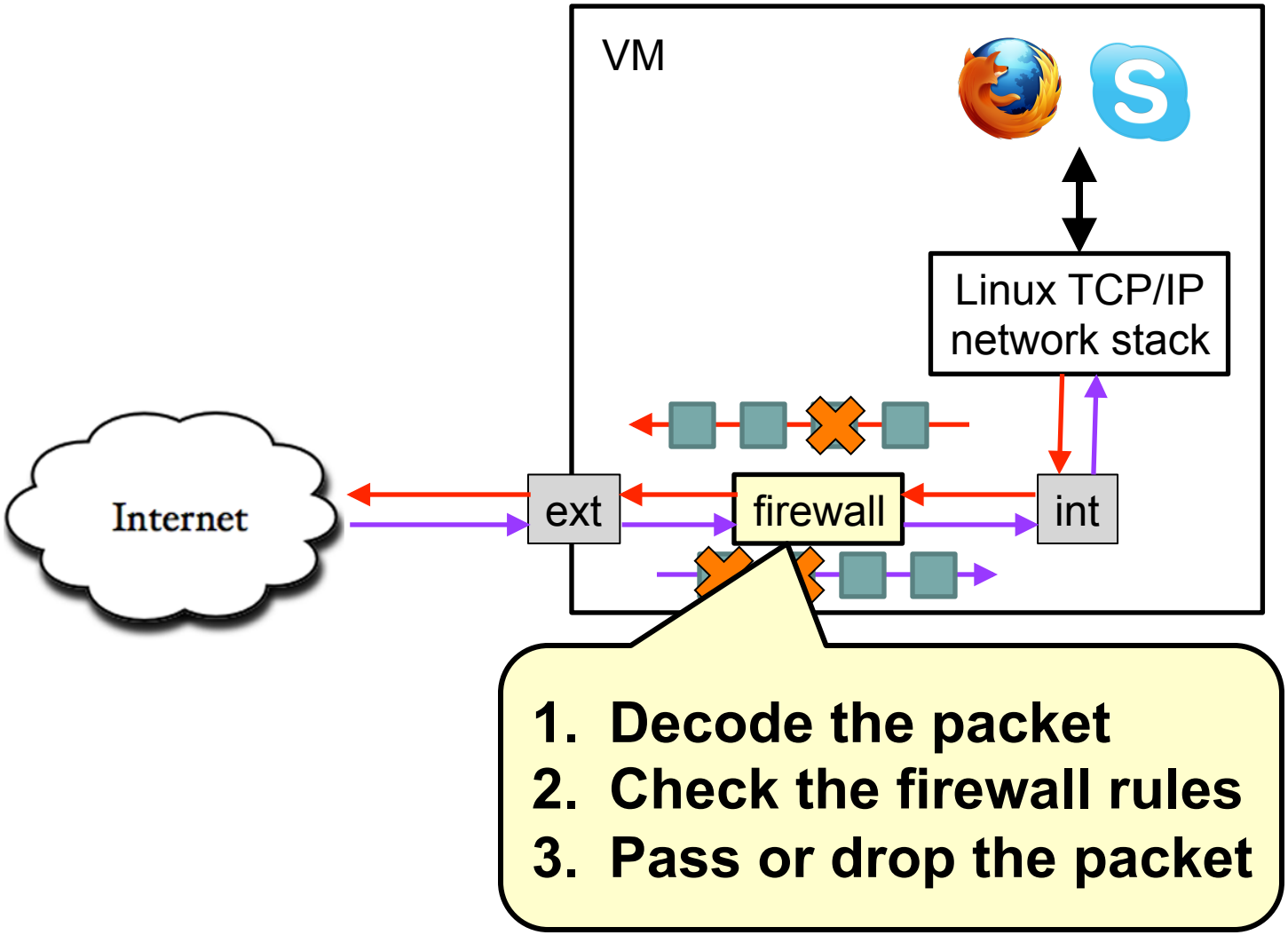
What Is Firewall?

Blocks malicious traffic



Blocks unauthorized traffic


VM
Linux TCP/IP
network stack
Internet
ext
firewall
int


VM
Linux TCP/IP
network stack
Internet
ext
firewall
int
1. Decode the packet
2. Check the firewall rules
3. Pass or drop the packet

Packets on wire look like this…

```
52 54 00 12 35 02 18 ba  dd c0 ff ee 08 00 45 00
00 ac 06 5a 40 00 40 06  72 a0 0a 00 02 0f 5b bd
59 86 b0 90 00 50 ca af  9a 10 00 00 fa 02 50 18
39 08 d2 5b 00 00 47 45  54 20 2f 76 30 2f 73 6f
75 72 63 65 73 20 48 54  54 50 2f 31 2e 31 0d 0a
48 6f 73 74 3a 20 76 69  64 65 6f 73 65 61 72 63
68 2e 75 62 75 6e 74 75  2e 63 6f 6d 0d 0a 55 73
65 72 2d 41 67 65 6e 74  3a 20 55 6e 69 74 79 20
56 69 64 65 6f 20 4c 65  6e 73 20 52 65 6d 6f 74
65 20 53 63 6f 70 65 20  76 36 2e 38 2e 30 0d 0a
43 6f 6e 6e 65 63 74 69  6f 6e 3a 20 4b 65 65 70
2d 41 6c 69 76 65 0d 0a  0d 0a
```

and your firewall should decode this.

Firewall rules

Type 1: a combination of

- Protocol (TCP/UDP/ICMP)
- IP address or country (e.g., Canada)
- Port number

Type 2: domain names

- E.g., block DNS queries for *.facebook.com

NO CHEATING

WE RUN COPY CHECKER

Questions?

- General questions
  - Ask your favorite GSI
- Project-specific questions
  - **Chang Lan (main)**
  - **Shoumik Palkar**
  - **Sangjin Han**
- **Start Early!**

Today

- Router-assisted Congestion Control (wrap up)
- Application layer: DNS

Router-Assisted Congestion Control

- Recall: Three tasks for CC:
  - Fairness
  - Rate adjustment
  - Detecting congestion

Router-Assisted Congestion Control

- Recall: Three tasks for CC:
  - Fairness → e.g., Fair Queuing
  - Rate adjustment → e.g., Rate Control Protocol
  - Detecting congestion → e.g., ECN

Fairness: General Approach

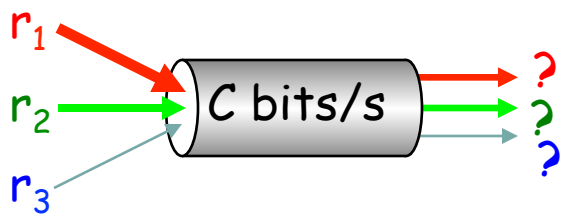
- Routers classify packets into “flows”
  - Assume for now flows ~ TCP connections
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion

Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

$$a_i = \min(f, r_i)$$

where f is the unique value such that Sum(a_i) = C

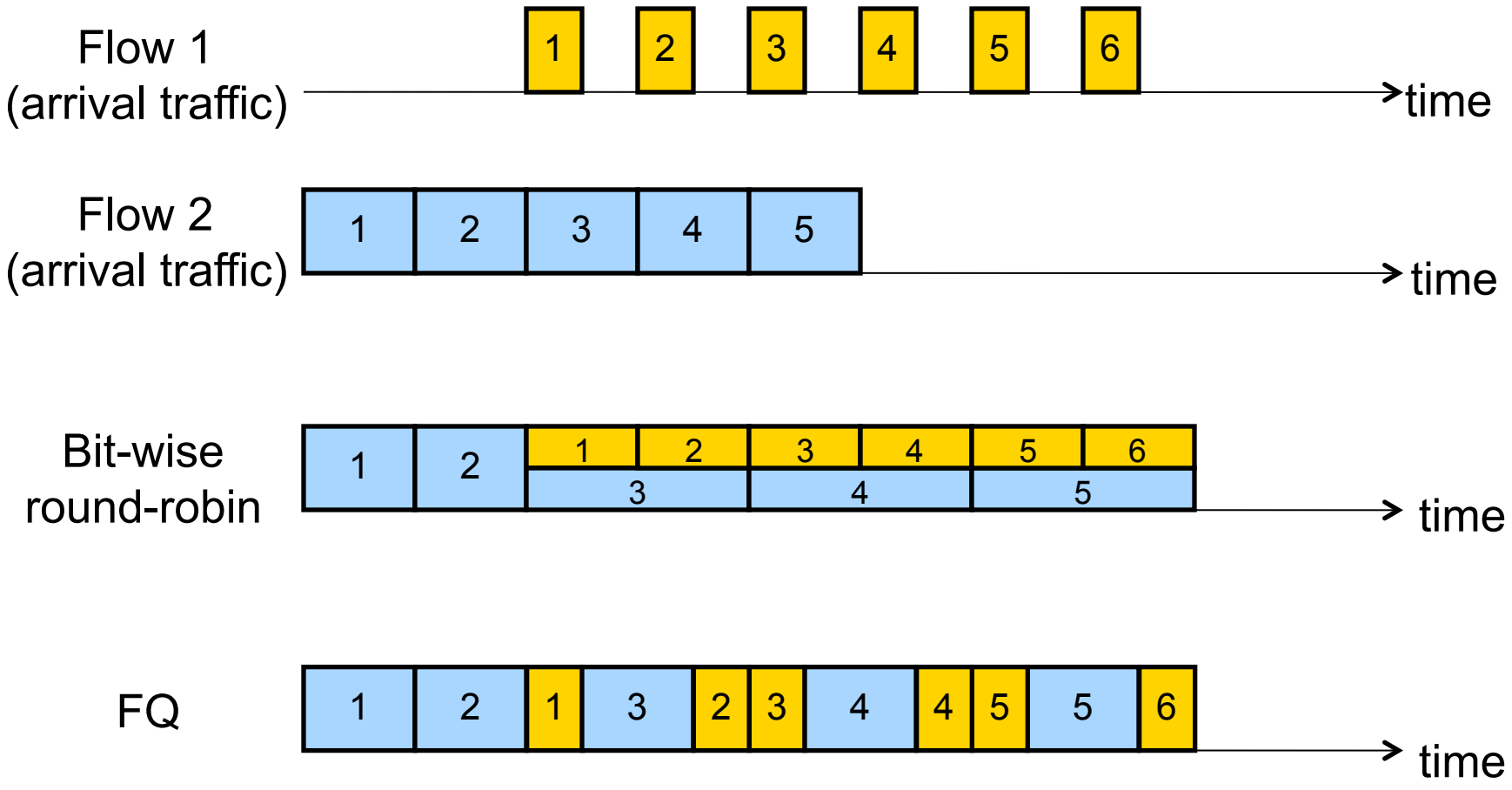


- This is what round-robin service gives if all packets are the same size

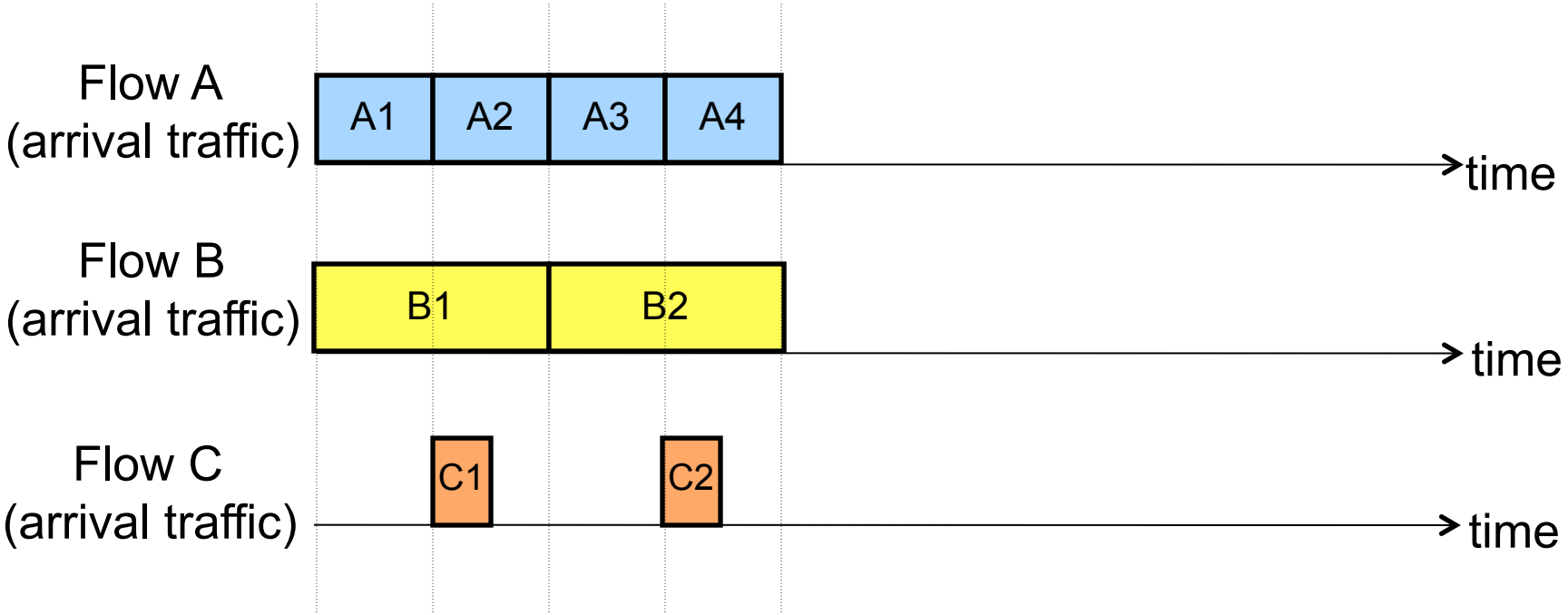
Fair Queuing is how we deal with packets of different sizes

- Mental model: Bit-by-bit round robin
- For each packet, compute its “deadline”
  - deadline → the time at which the last bit of a packet would have left the router *if* flows are served bit-by-bit
- Transmit packets in increasing order of deadlines

Example


Flow 1
(arrival traffic)
1
2
3
4
5
6
time
Flow 2
(arrival traffic)
1
2
3
4
5
time
Bit-wise
round-robin
1
2
1
2
3
4
5
6
3
4
5
time
FQ
1
2
1
3
2
3
4
4
5
5
6
time

Exercise



- What's the packet order at the output for four scheduling disciplines?
  - FIFO
  - Round-Robin
  - Fair Queuing
  - Priority
- Assume Flow A > B > C in priority and for breaking ties

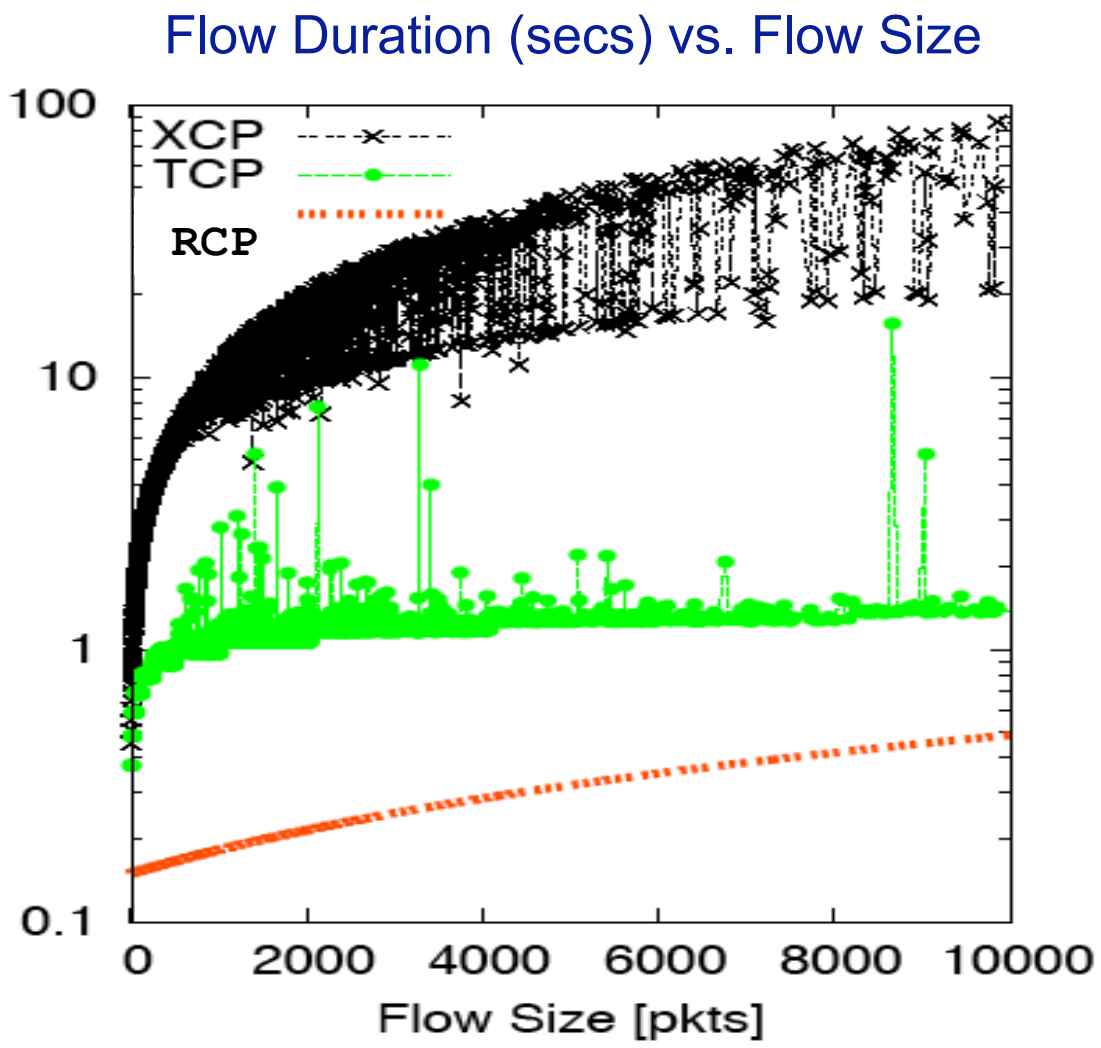
Router-Assisted Congestion Control

- Recall: Three tasks for CC:
  - Fairness → e.g., Fair Queuing
  - Rate adjustment → e.g., Rate Control Protocol
  - Detecting congestion → e.g., ECN

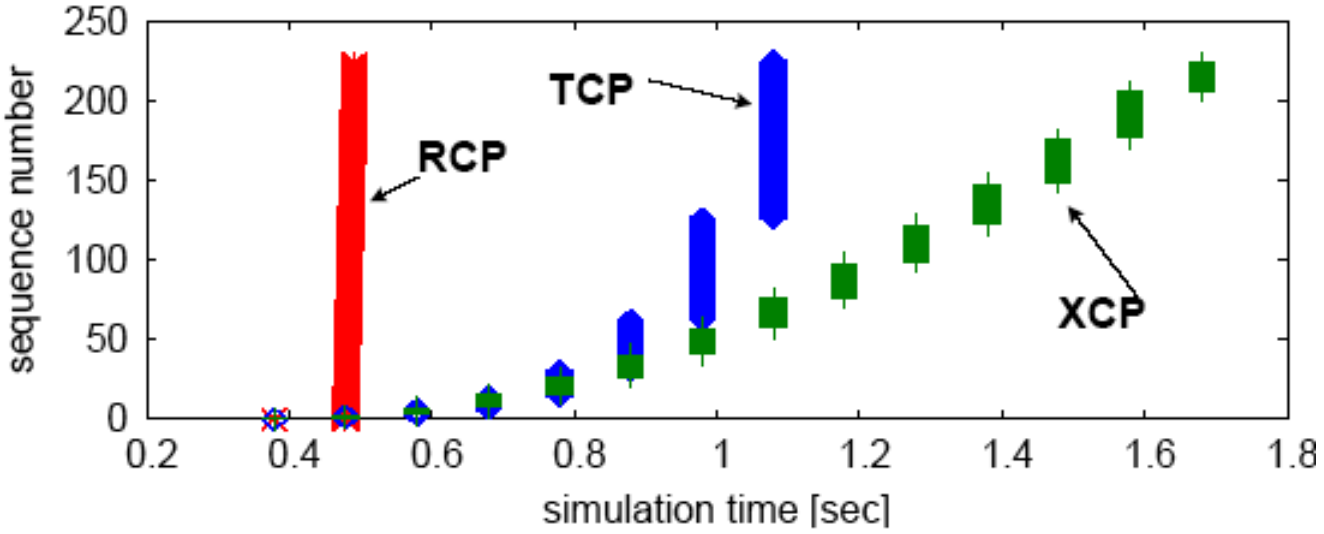
Idea: Let routers tell endhosts what rate they should use

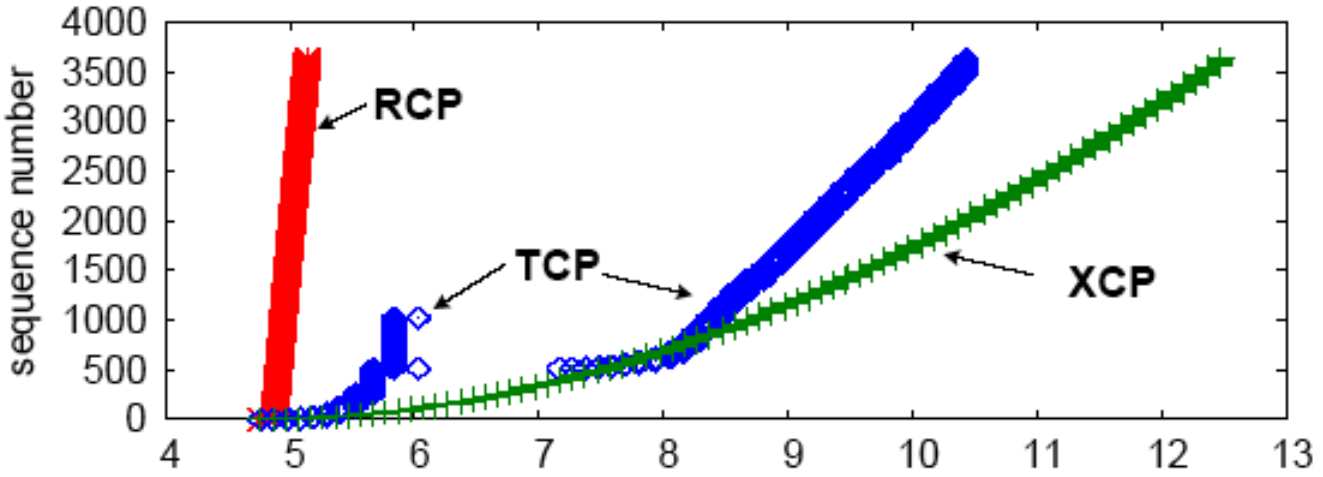
- Packets carry “rate field”
- Routers insert “fair share” *f* in packet header
- End-hosts set sending rate (or window size) to *f*
  - Note: still assumes end-hosts play by the rules
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati&McKeown in 2007

Flow Completion Time: TCP vs. RCP (Ignore XCP)



Why the improvement?





Why is RCP important?

- Congestion control had long focused on safety (primary) and fairness (secondary)
  - TCP does a good enough job at both
- RCP highlighted CC's impact on performance
  - performance matters a *lot* in today's environment
- RCP shows we can do better but is still fairly complex and not incrementally deployable

RC3 – *Mittal et al., 2014*

- RC3: Recursively Cautious Congestion Control
- Senders transmit at max rate from the get-go
- But packets are marked with priorities
  - CWND packets are sent at priority#1
  - The remaining at low priority
- Routers implement priority scheduling (exists today)
  - If no congestion, all packets get through (high performance)
  - else, low priority packets dropped (safety and fairness)

Router-Assisted Congestion Control

- Recall: Three tasks for CC:
  - Fairness → e.g., Fair Queuing
  - Rate adjustment → e.g., Rate Control Protocol
  - Detecting congestion → e.g., ECN

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - Many options for when routers set the bit
  - Tradeoff between (link) utilization and (packet) delay
- Receiver relays ECN bit in its ACK to the sender
  - Congestion semantics can be exactly like that of drop
  - e.g., endhost reacts as though it saw a drop
- Advantages:
  - Don’t confuse corruption with congestion
  - Serves as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
  - Today: defined in RFC 3168 using ToS/DSCP bits in the IP header

A final proposal: Charge people for congestion!

- Use ECN as congestion markers
- Whenever I get an ECN bit set, I have to pay $$
- Now, there's no debate over what a flow is, or what fair is…
- Idea started by Frank Kelly at Cambridge
  - "optimal" solution, backed by much math
  - Great idea: simple, elegant, effective
  - Unclear that it will impact practice

Recap!

- TCP
  - allows apps to assume reliable bytestream delivery
  - with (mostly) fair and safe sharing of network resources

- Congestion control
  1. avoid overloading the network to the point of collapse
  2. while sharing resources fairly
  3. and maximizing performance
  - *TCP does a good enough job at the first two but the last is more iffy*

Taking Stock

- Course so far
  - Concepts *(links, delays, switches)*
  - Overall architecture (*layers, protocols, principles*)
  - Network layer (*how we interconnect hosts and networks*)
  - Transport layer (*how we make best-effort usable to apps*)

- What's left?
  - Application layer (DNS, HTTP) ← this week
  - Lower layers (Ethernet, Wireless) ← next
  - Advanced topics (datacenters, SDN, IPv6) ←last ~2 weeks

Some very special guest lectures

- Nov 10: Yahel Ben-David (*Wireless*)
- Nov 19: Stephen Strowes (*IPv6*)
- Nov 24: Scott Shenker (*SDN)*
- Dec 1: Panel: Careers in networking (*tentative)*

Application Layer

- Domain Name System (DNS)
  - What's behind (e.g.) instr.eecs.berkeley.edu?
- HTTP and the Web
  - What happens when you click on a link?

Host Names & Addresses

- Host addresses: *e.g., 169.229.131.109*
  - a number used by protocols
  - conforms to network structure (the “where”)
- Host names: *e.g., instr.eecs.berkeley.edu*
  - mnemonic name usable by humans
  - conforms to organizational structure (the “who”)
- The Domain Name System (DNS) is how we map from one to the other
  - a directory service for hosts on the Internet

Why bother?

- Convenience
  - Easier to remember www.google.com than 74.125.239.49
- Provides a level of indirection!
  - Decoupled names from addresses
  - Many uses beyond just naming a specific host

DNS: Early days

- Mappings stored in a hosts.txt file (in /etc/hosts)
  - maintained by the Stanford Research Institute (SRI)
  - new versions periodically copied from SRI (via FTP)

- 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

- The Domain Name System (DNS) was invented to fix this
  - First name server implementation done by 4 UCB students!

Goals?

- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available
- Correct
  - no naming conflicts (uniqueness)
  - consistency
- Lookups are fast

How?

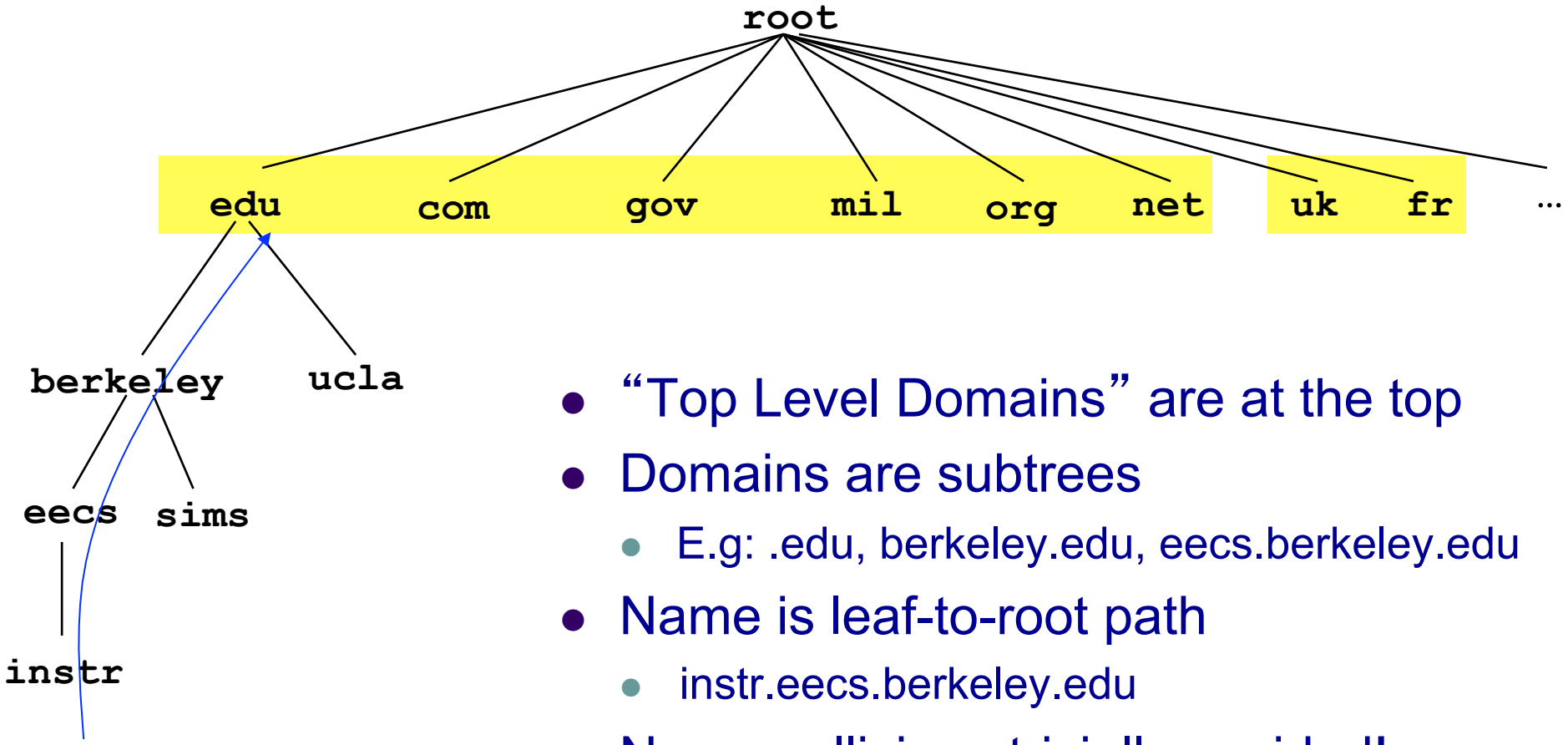
- Partition the namespace
- Distribute administration of each partition
  - Autonomy to update my own (machines') names
  - Don't have to track everybody's updates
- Distribute name resolution for each partition
- *How should we partition things?*

Key idea: hierarchical distribution

Three intertwined hierarchies

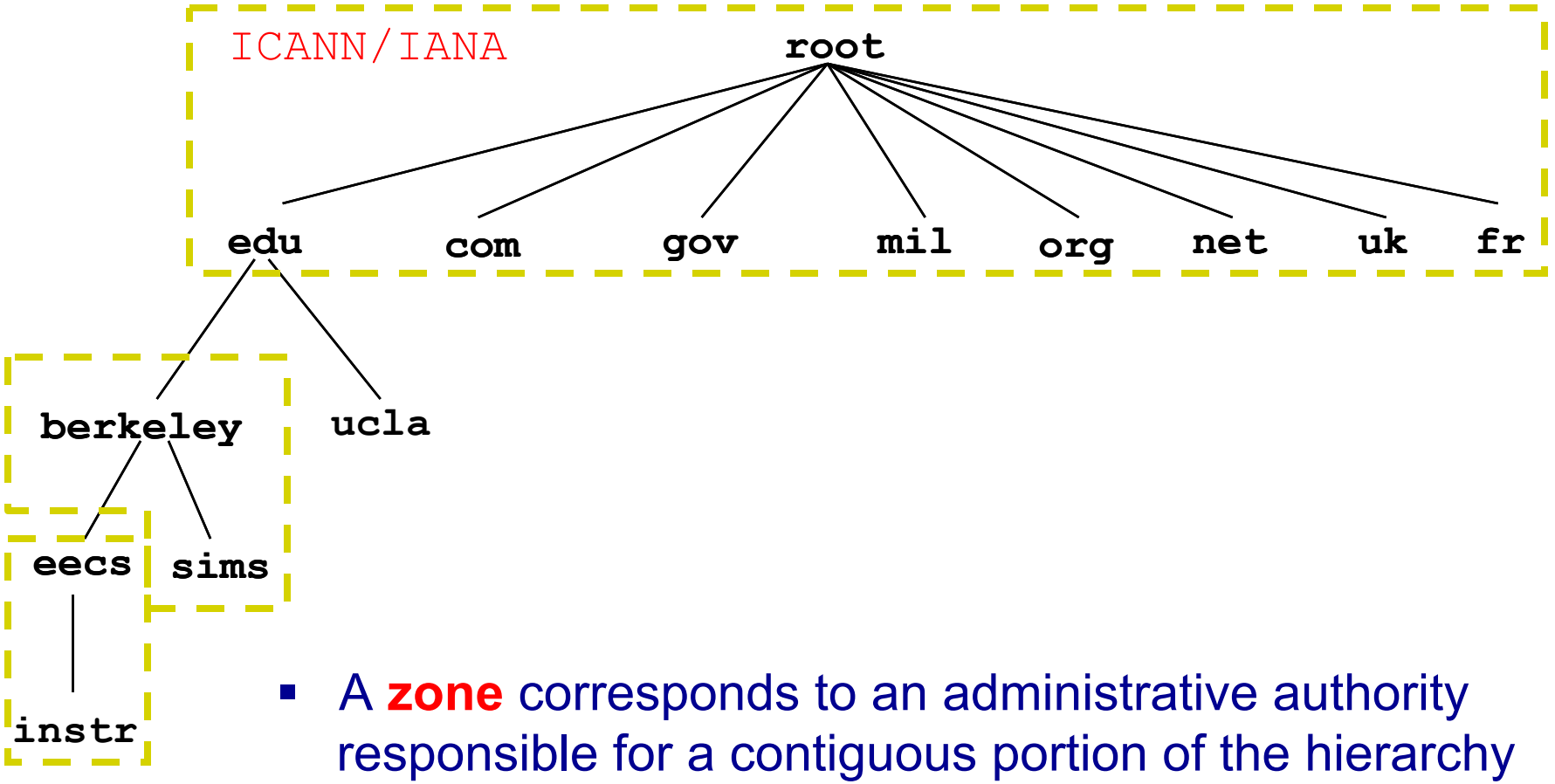
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralized administrator
- Hierarchy of servers
  - As opposed to centralized storage

Hierarchical Namespace



- "Top Level Domains" are at the top
- Domains are subtrees
  - E.g: .edu, berkeley.edu, eecs.berkeley.edu
- Name is leaf-to-root path
  - instr.eecs.berkeley.edu
- Name collisions trivially avoided!
  - each domain's responsibility

Hierarchical Administration



- A **zone** corresponds to an administrative authority responsible for a contiguous portion of the hierarchy
- E.g.: UCB controls **.berkeley.edu* and **.sims.berkeley.edu* while EECS controls **.eecs.berkeley.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

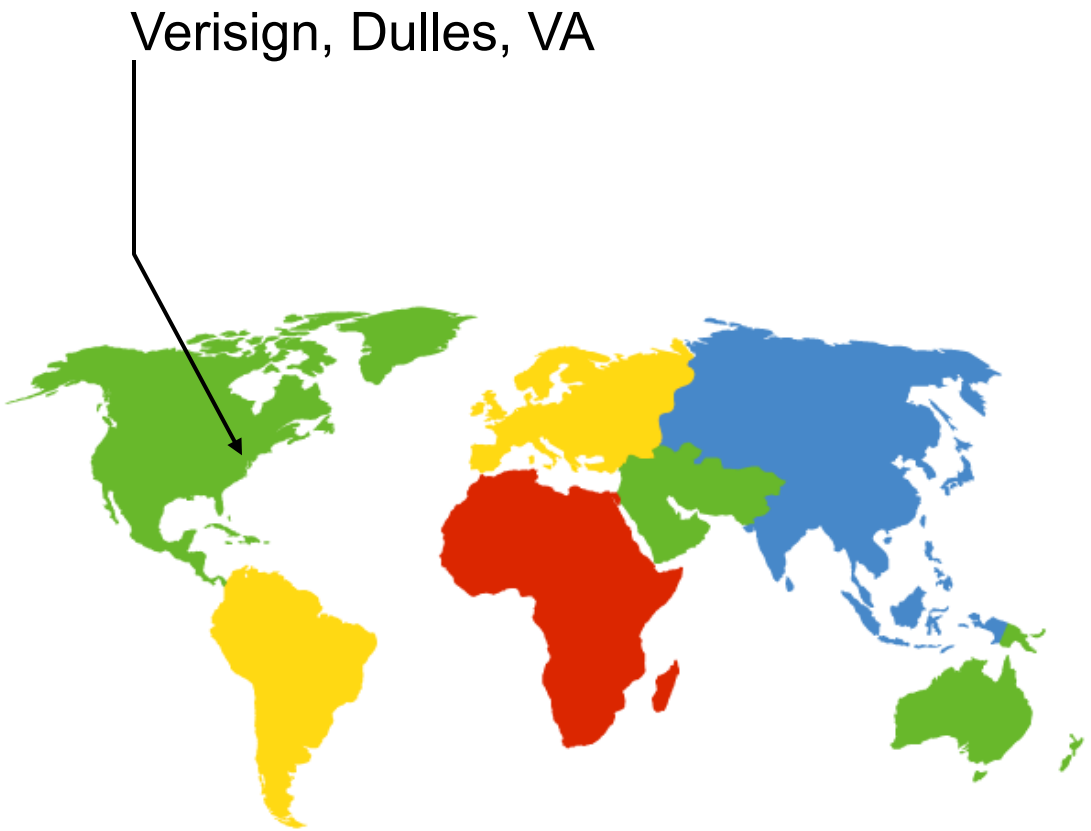
- Every server knows the address of the root name server
- Root servers know the address of all TLD servers
- …
- An authoritative DNS server stores name-to-address mappings (“resource records”) for all DNS names in the domain that it has authority for

→ Each server stores a subset of the total DNS database

→ Each server can discover the server(s) responsible for any portion of the hierarchy

DNS Root

- Located in Virginia, USA



DNS Root Servers

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

DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)
- Replicated via any-casting

Anycast in a nutshell

- Routing finds shortest paths to destination
- What happens if multiple machines advertise the same address?
- The network will deliver the packet to the closest machine with that address
- This is called “anycast”
  - 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 = MX: *(→ Mail eXchanger)*
  - name = domain in email address
  - value = name(s) of mail server(s)

Inserting Resource Records into DNS

- Example: you just created company "FooBar"
- You get a block of IP addresses from your ISP
  - say 212.44.9.128/25

- Register foobar.com at registrar (e.g., Go Daddy)
  - 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)**

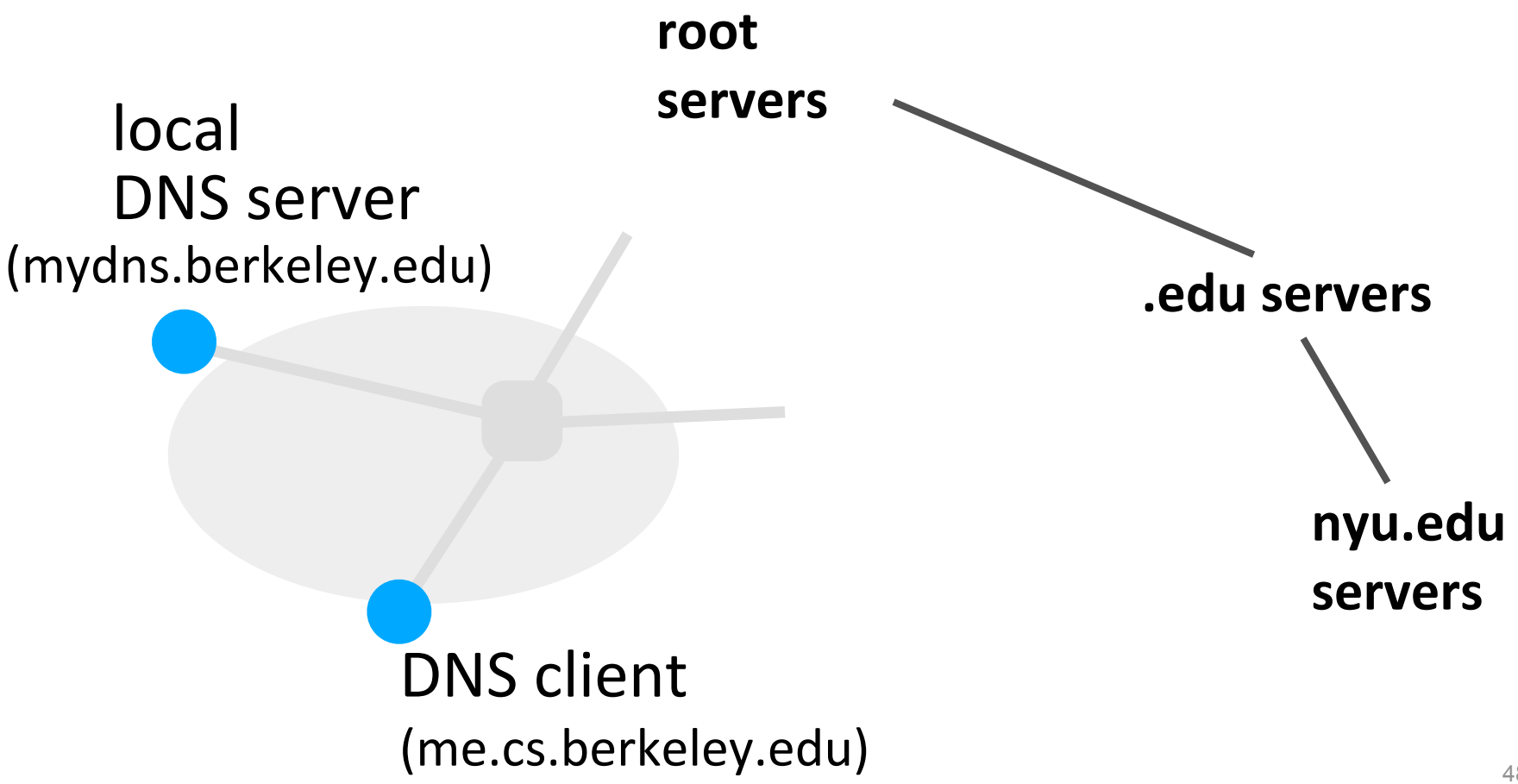
- Store resource records in your server dns1.foobar.com
  - e.g., type A record for www.foobar.com
  - e.g., type MX record for foobar.com

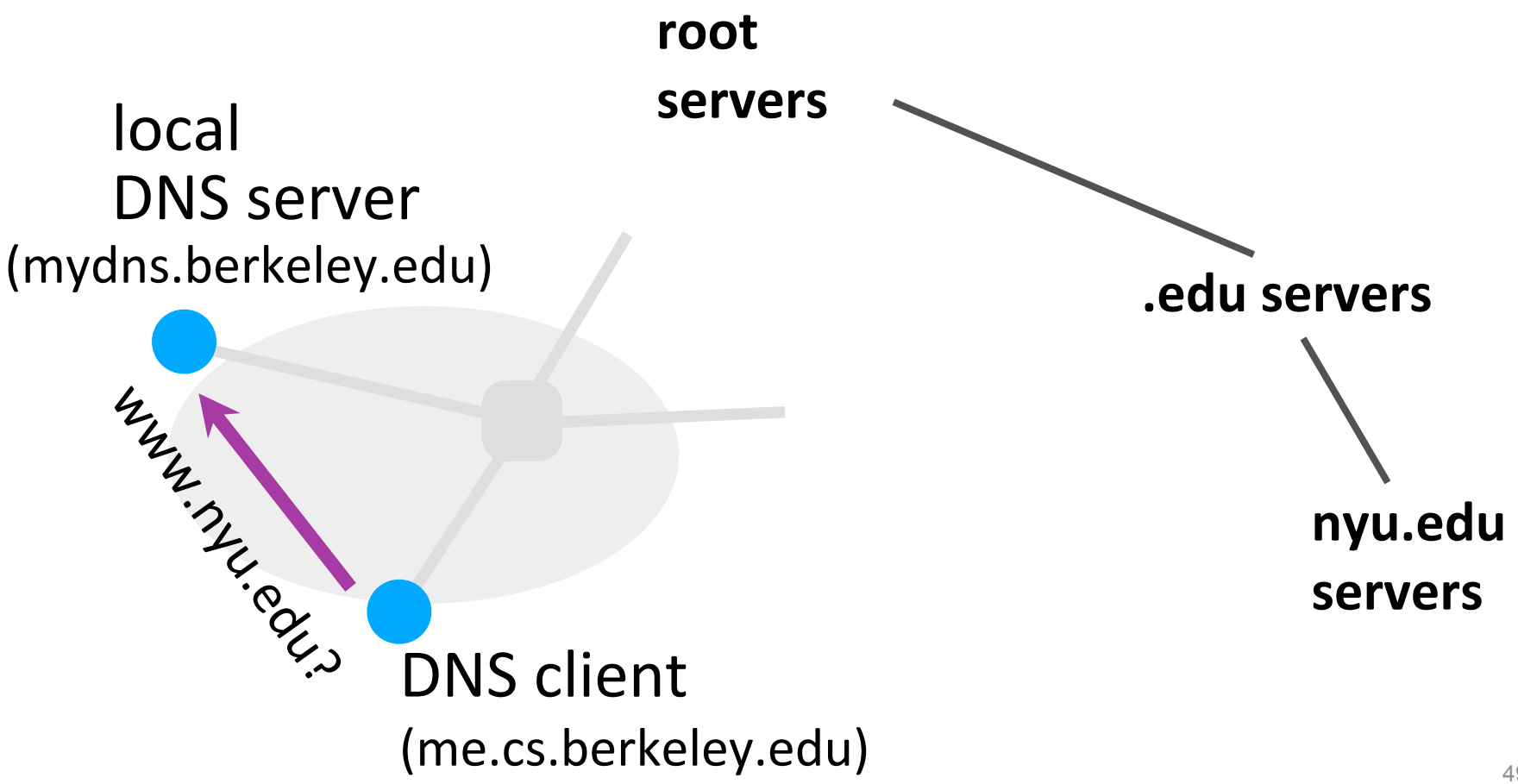
Using DNS (Client/App View)

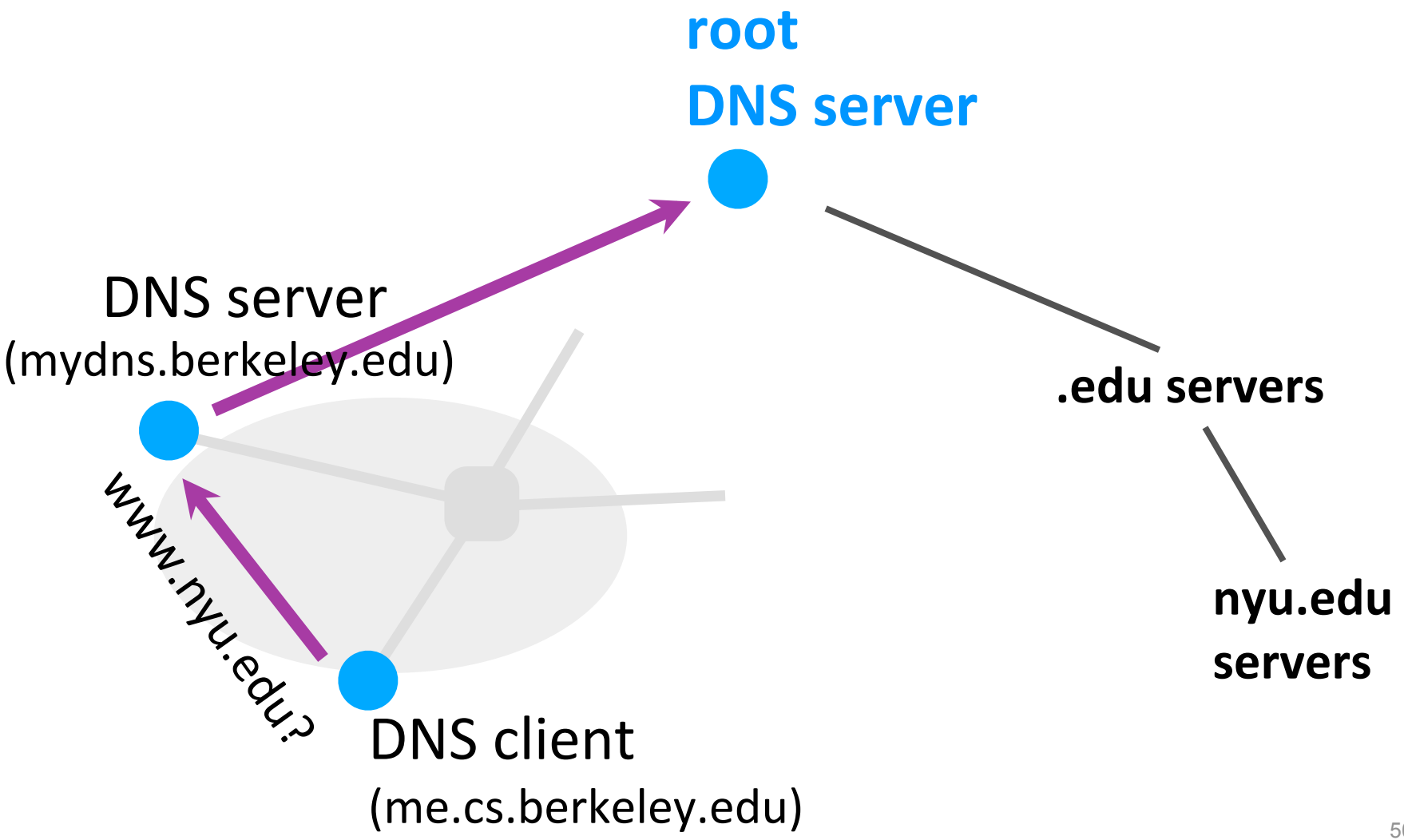
- Two components
  - Local DNS servers
  - Resolver software on hosts

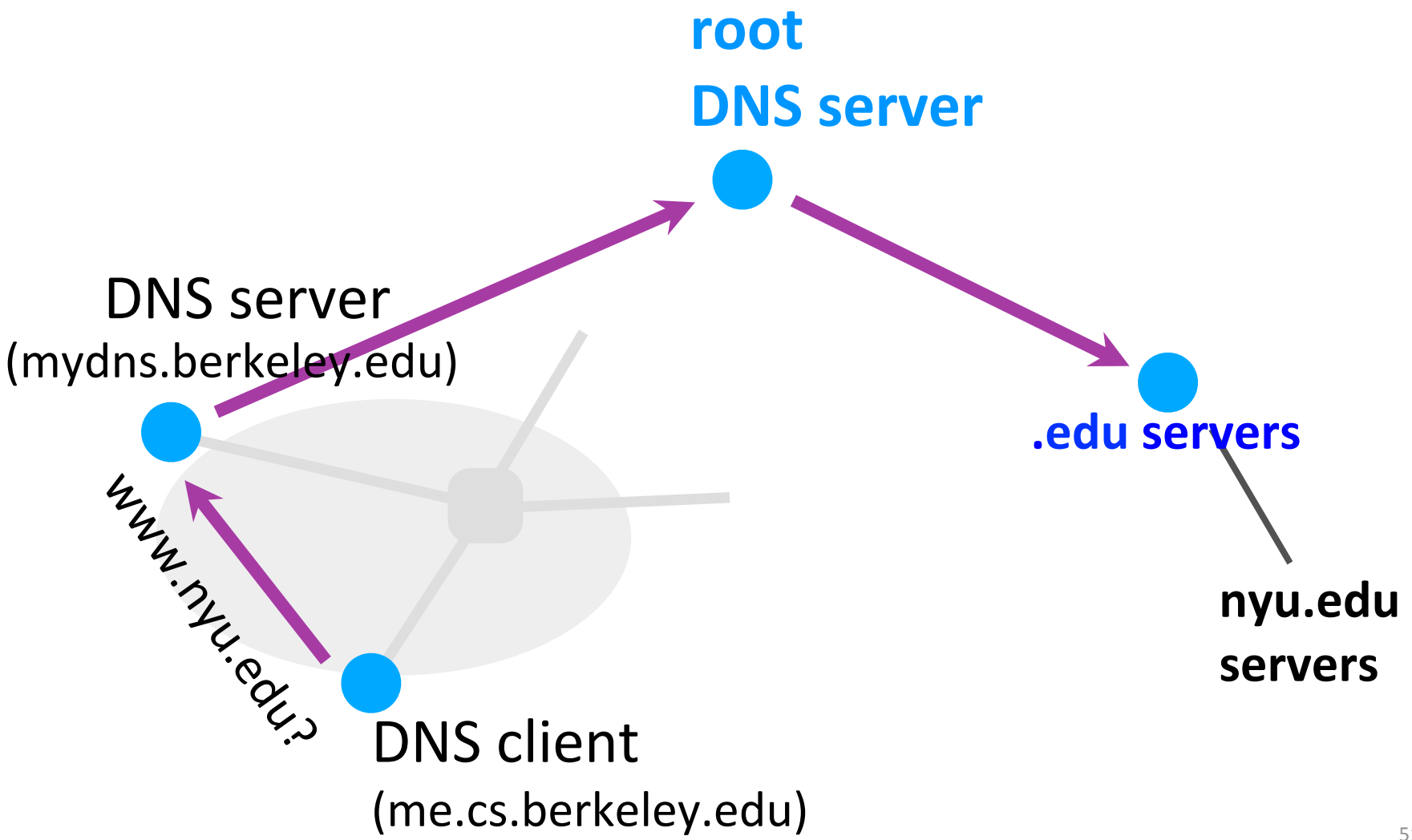
- Local DNS server ("default name server")
  - Clients configured with the default server's address or learn it via a host configuration protocol (e.g., DHCP – *future lecture*)

- Client application
  - Obtain DNS name (e.g., from URL)
  - Triggers DNS request to its local DNS server

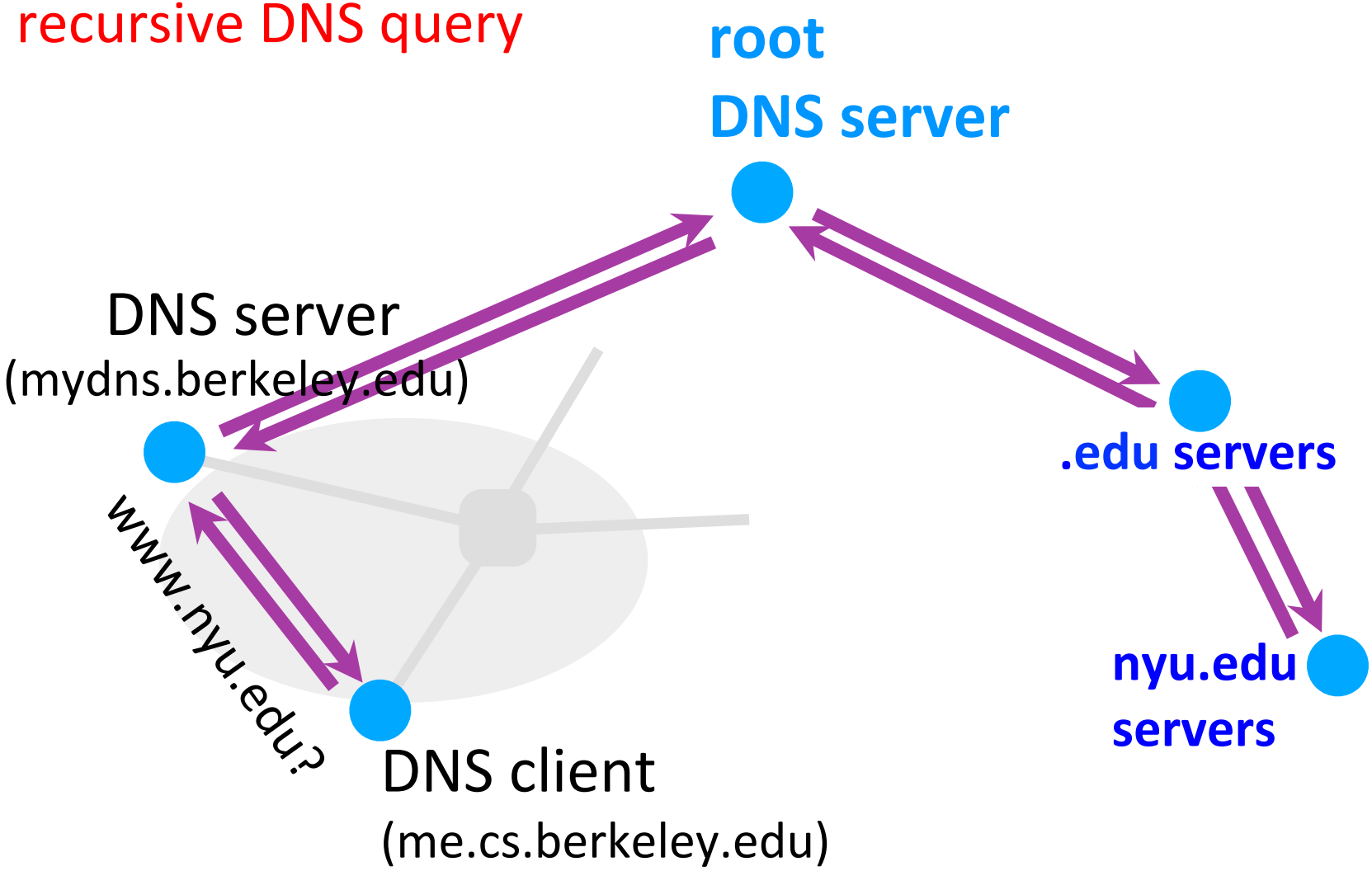

root
servers
local
DNS server
(mydns.berkeley.edu)
.edu servers
nyu.edu
servers
DNS client
(me.cs.berkeley.edu)

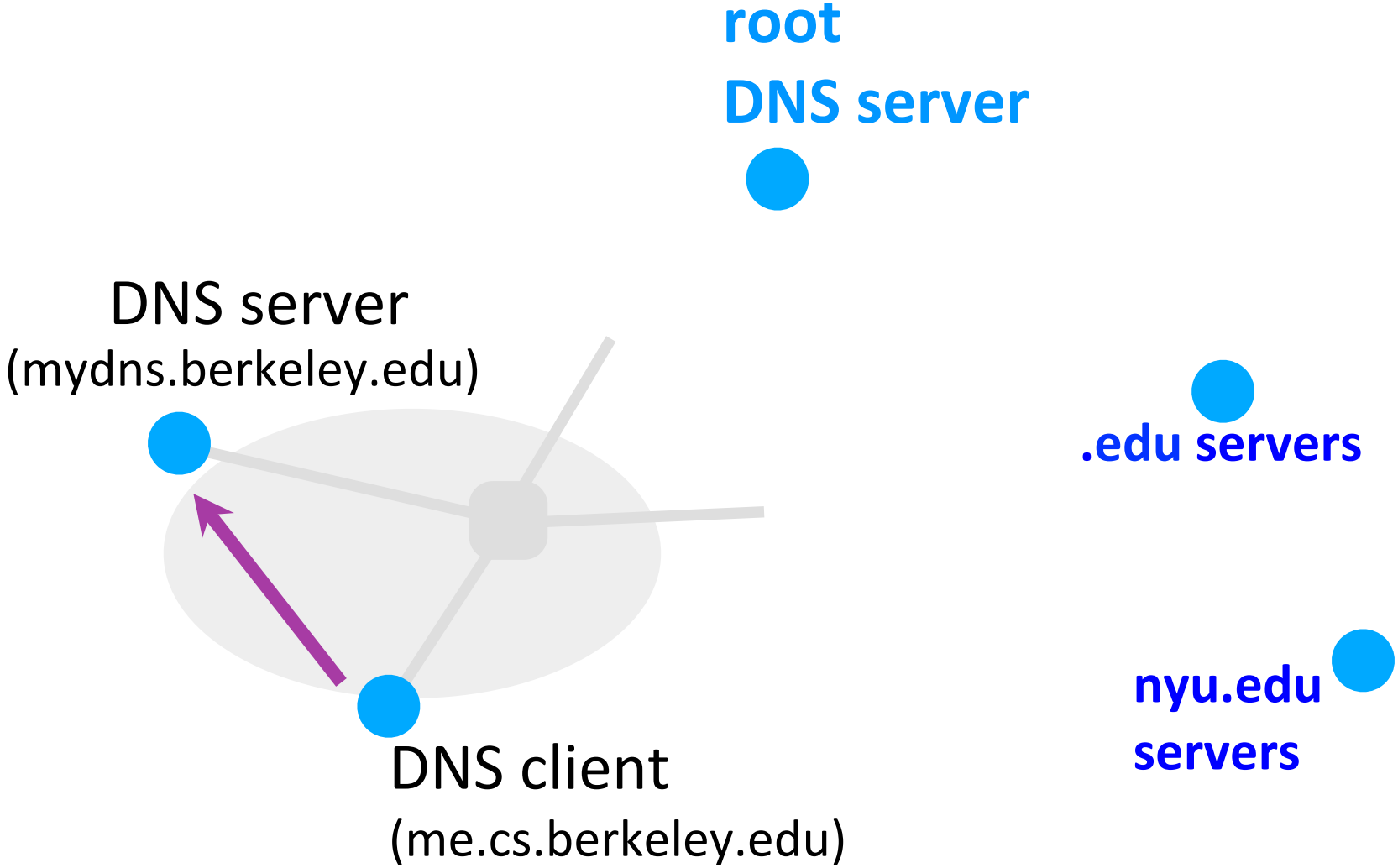

root
servers
local
DNS server
(mydns.berkeley.edu)
.edu servers
nyu.edu
servers
www.nyu.edu?
DNS client
(me.cs.berkeley.edu)


root
DNS server
DNS server
(mydns.berkeley.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.berkeley.edu)

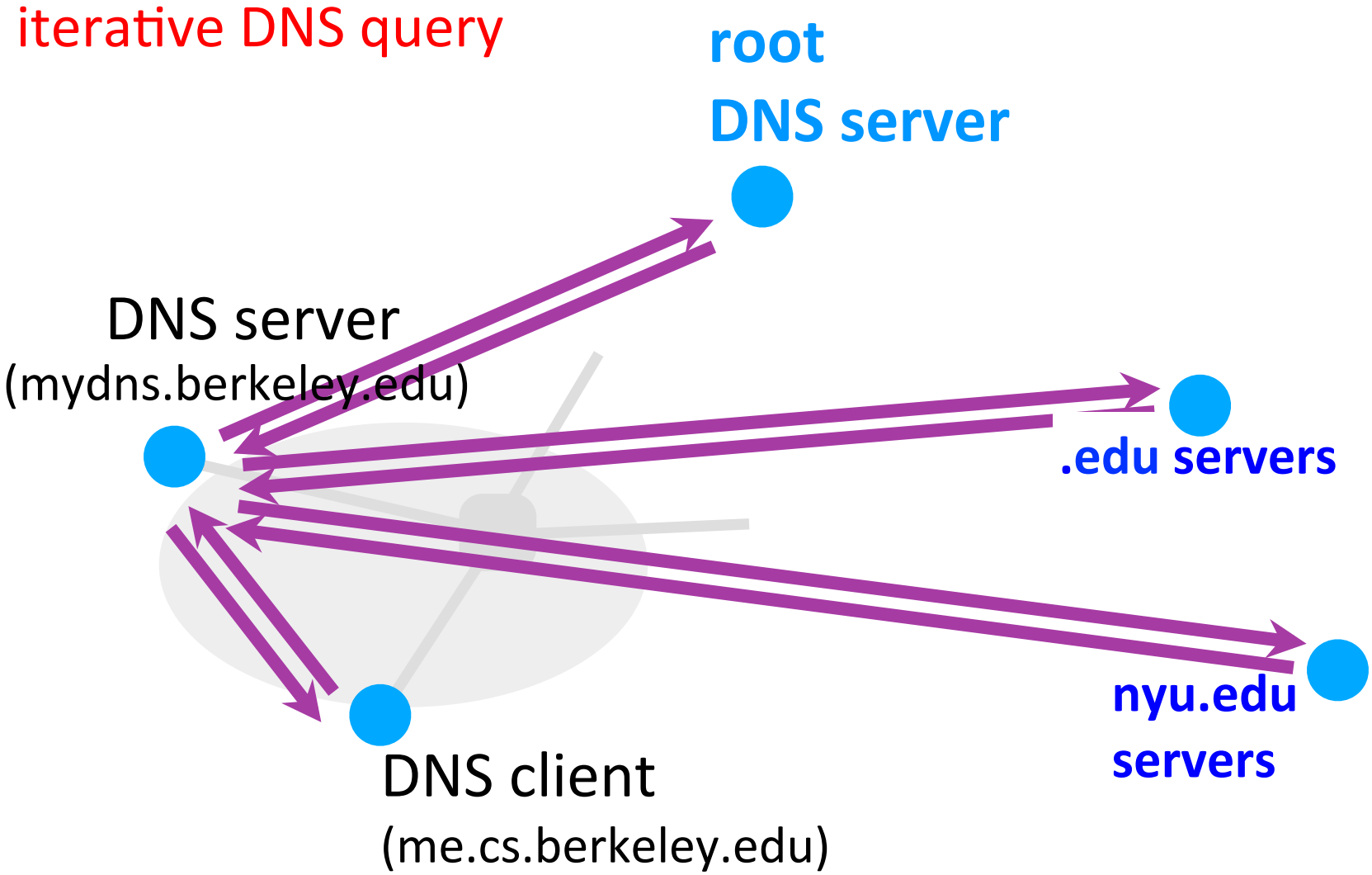

root
DNS server
DNS server
(mydns.berkeley.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.berkeley.edu)

recursive DNS query




root
DNS server
DNS server
(mydns.berkeley.edu)
.edu servers
nyu.edu
servers
DNS client
(me.cs.berkeley.edu)

iterative DNS query



DNS Protocol

- Query and Reply messages; both with the same message format
  - *see text/section for details*
- Client-Server interaction on UDP Port 53
  - Spec supports TCP too, but not always implemented

Goals – how are we doing?

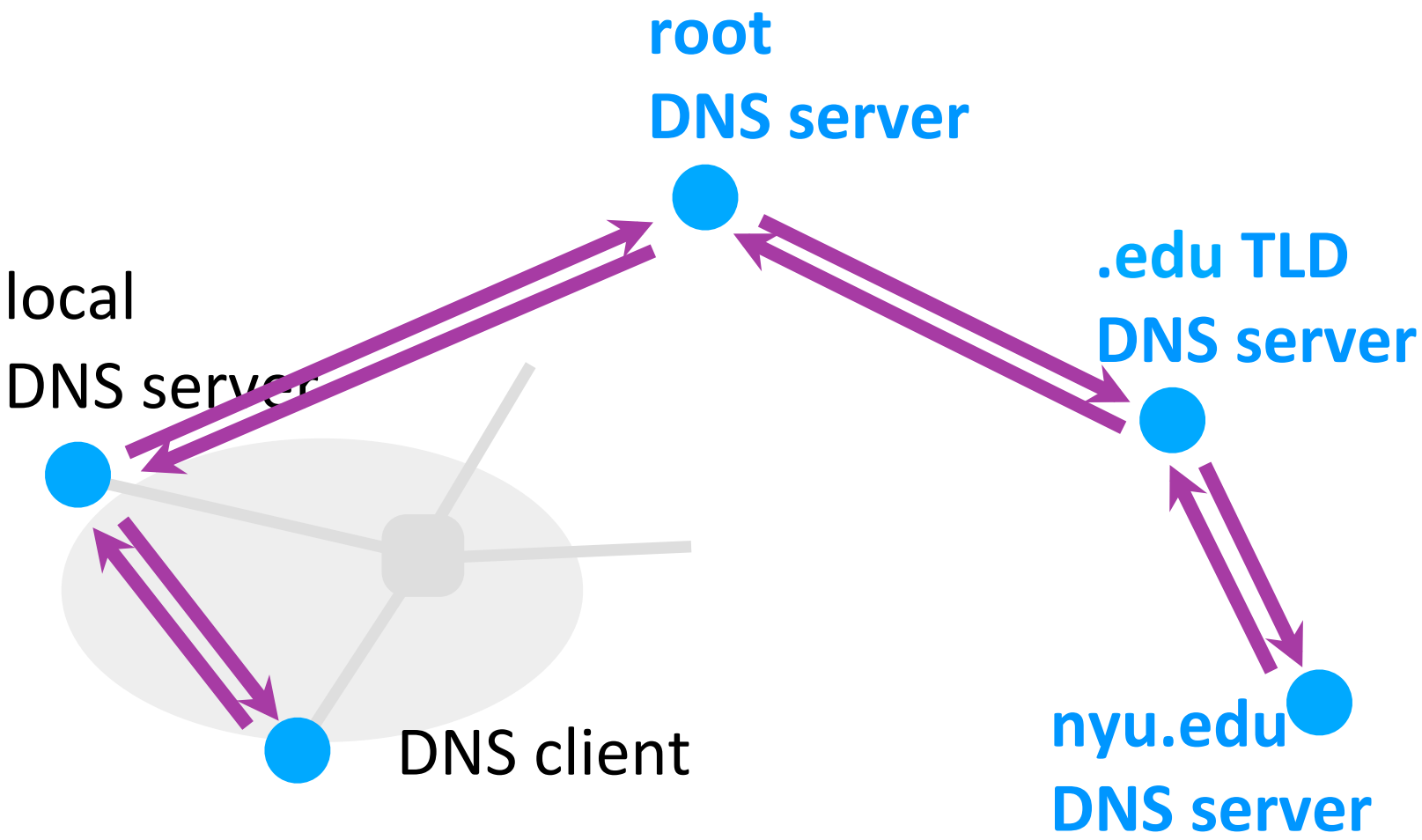
- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available

Per-domain availability

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

Goals – how are we doing?

- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available
- Correct
  - no naming conflicts (uniqueness)
  - consistency
- Lookups are fast


root
DNS server
.edu TLD
DNS server
local
DNS server
DNS client
nyu.edu
DNS server

Caching

- Caching of DNS responses at all levels

- Reduces load at all levels
- Reduces delay experienced by DNS client

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

Negative Caching

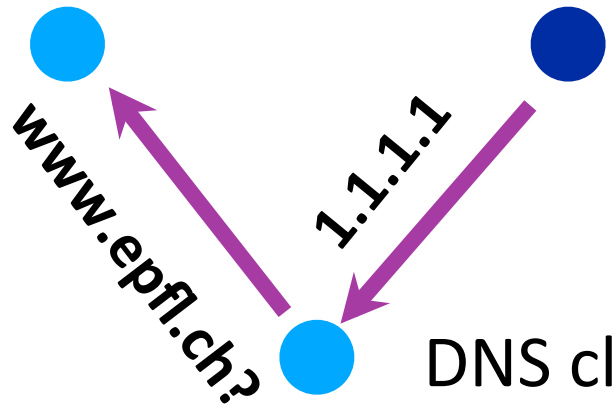
- Remember things that don't work
  - Misspellings like *www.cnn.comm* and *www.cnnn.com*
  - These can take a long time to fail the first time
  - Good to remember that they don't work
  - … so the failure takes less time the next time around
- Negative caching is optional

How can one attack DNS?

root
DNS server

.edu TLD
DNS server

local
DNS server



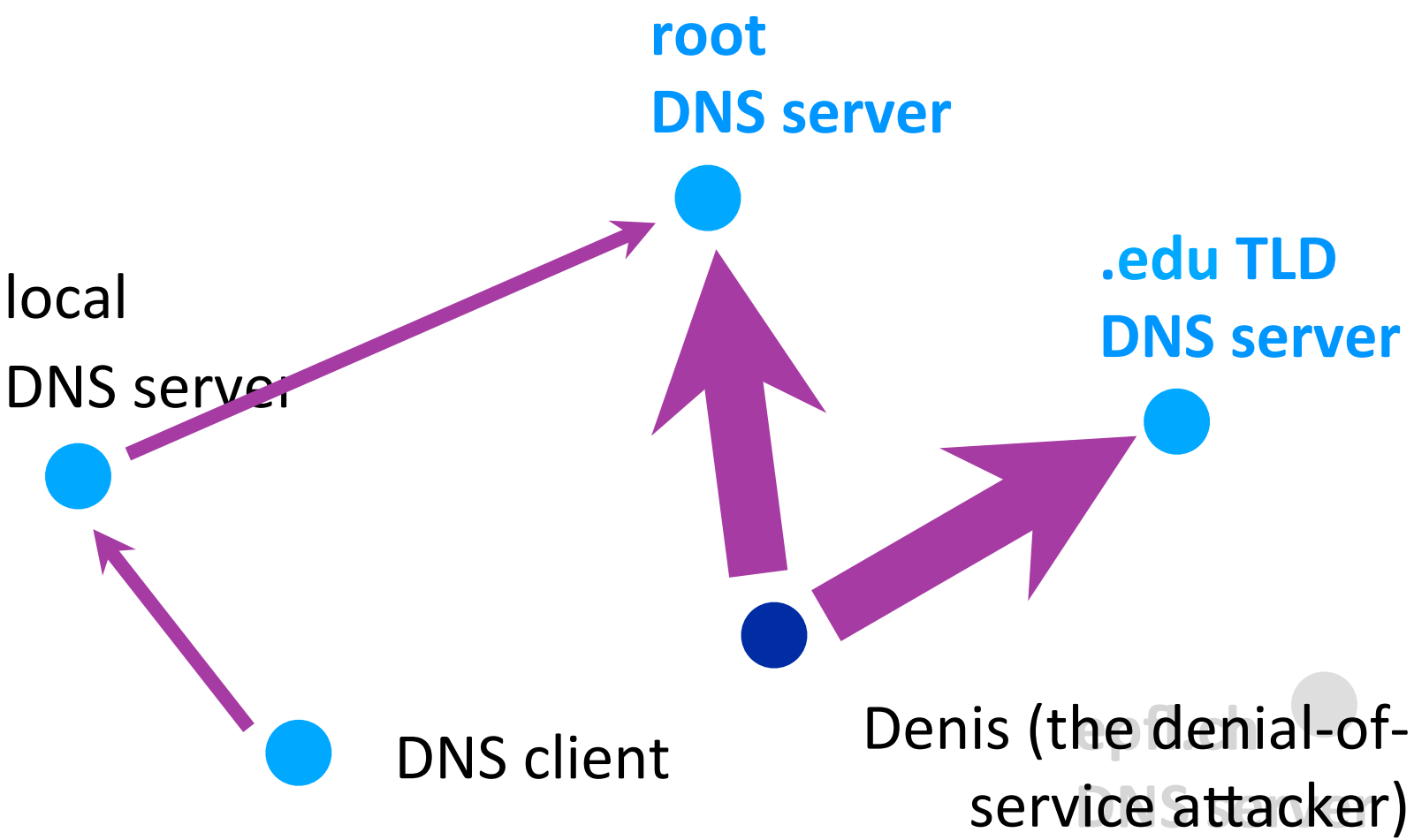
Persa (the impersonator)

DNS client

nyu.edu
DNS server

How can one attack DNS?

- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*


root
DNS server
.edu TLD
DNS server
local
DNS server
DNS client
Denis (the denial-of-
service attacker)

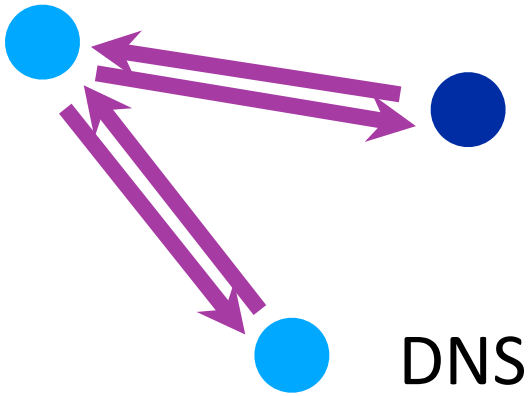
How can one attack DNS?

- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*
- Denial-of-service the root or TLD servers
  - *make them unavailable to the rest of the world*

root
DNS server

.edu TLD
DNS server

local
DNS server



Casha

DNS client

nyu.edu
DNS server

How can one attack DNS?

- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*
- Denial-of-service the root or TLD servers
  - *make them unavailable to the rest of the world*
- Poison the cache of a DNS server
  - *increase the delay experienced by DNS clients*

Important Properties of DNS

Administrative delegation and hierarchy results in:

- Easy unique naming
- “Fate sharing” for network failures
- Reasonable trust model
- Caching lends scalability, performance

DNS provides Indirection

- Addresses can change underneath
  - Move www.cnn.com to a new IP address
  - Humans/apps are unaffected
- Name could map to multiple IP addresses
  - Enables load-balancing
- Multiple names for the same address
  - E.g., many services (mail, www, ftp) on same machine
- Allowing “host” names to evolve into “service” names

Next Lecture (Scott Shenker lecturing)

- HTTP and the Web
- (Broadcast) Ethernet