

DNS

Autumn 2024
cs168.io

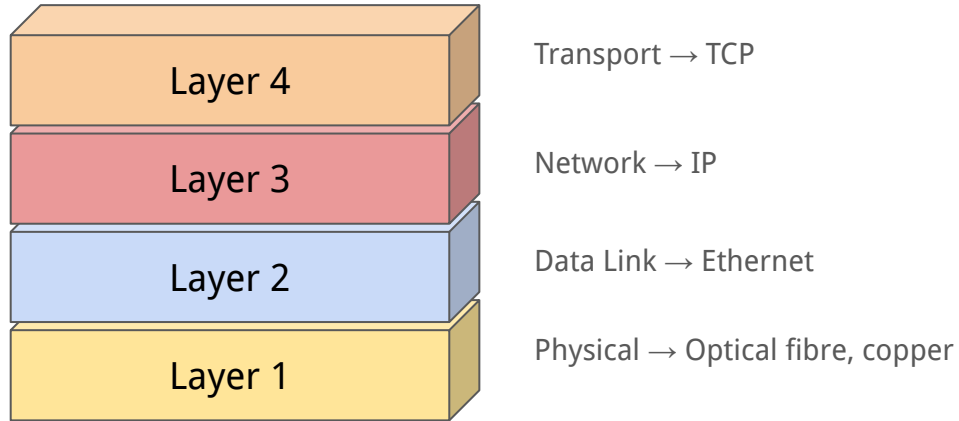
Rob Shakir

Thanks to Murphy McCauley for some of the material!

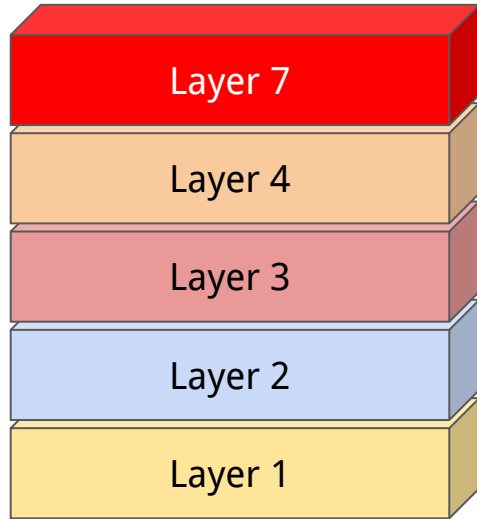
Where are we?

- Talked about foundations and principles.
 - e.g., packet switching, end-to-end
- How the Internet is made up, and
 - Intra-domain routing
 - Inter-domain routing
- Talked about IP and TCP
 - What do packets look like?
 - How do we make an unreliable network look reliable?
 - How do we deal with congestion?
- Lots of “plumbing”!

Visually



Visually



Transport → TCP

Network → IP

Data Link → Ethernet (*we've mentioned this, but we'll come back to it*)

Physical → Optical fibre, copper

We are here! Application Layer

Lots of different applications – but we'll focus on some common and critical ones.

Thinking back...

- On the Internet and ARPANET, three killer applications.
 - Remote terminal
 - Connect to someone else's machine remotely – like SSH today.
 - `telnet <remote host>`
 - File transfer
 - Copy files across the network
 - `ftp <remote host>`
 - Email
 - Send a message to another user
 - `mail <user>@<remote host>`
- Remembering the remote host addresses is difficult for humans.
 - `mail alice@46.0.1.2?!`

Avoiding numerical addresses.

- Rather than use numerical addresses have a **hostname**.
- Record this hostname and its mapping to an IP address on the Internet in `hosts.txt`.
- e.g., UCB-ARPA → 10.0.0.78
 - Now rather than using mail `mosher@10.0.0.78` - one can mail `mosher@ucb-arpa`

Maintaining the lists of hosts

- Originally maintained by Elizabeth Jocelyn “Jake” Feinler.
 - Give her a call and she would add an entry to the hosts file!
 - From 1982...

```
UCB-ARPA 10.0.0.78 ARPA Mosher, David A.  
(Mosher@BERKELEY)  
University of California  
Computer Systems Research Group  
457 Evans Hall  
Berkeley, California 94720  
(415) 642-7780  
  
CPUtype: VAX-11/780(UNIX)
```

“I remembered that back then we simply xeroxed the hosts.txt file and put it into the Arpanet Directory, so I copied that.” – [Elizabeth Feinler](#)

Introducing hosts.txt

- Originally, the list of hostnames ("hosts.txt") was human readable.

ARPANET DIRECTORY
NIC 19275
Jan. 1974

HOST NAMES

HOST NAMES

HOSTNAME	HOST ADDR (Dec)	LIAISON	STATUS
AFWL-TIP	176	D Hyde (505)247-1711 x3803	TIP, Up 3-74
ALOHA-TIP	164	R Binder (808)948-7066	TIP
AMES-11	208	J Hart (415)965-5935	USER, up 12-73
AMES-67	16	W Hathaway (415)965-6033	SERVER
AMES-TIP	144	W Hathaway (415)965-6033	TIP
ANL	?	L Amiot (312)739-7711 x4309	SERVER, up 2-74
ARPA-DMS	28	S Crocker (202)694-5037	USER, Agency use only
ARPA-TIP	156	S Crocker (202)694-5037	TIP

Keeping hosts.txt up-to-date.

- Every site maintained their own copy of hosts.txt.
 - Leading to differing mappings at each site.
 - See [RFC606](#) (1973) – *“it seems about time to put an end to the absurd situation where each site on the network must maintain a different, generally out-of-date, host list”*.
- First step – make the list machine readable.
 - Format defined in [RFC608](#).

Introducing hosts.txt

- Originally, the list of hostnames (“hosts.txt”) was human readable.
- Eventually converted to a machine-readable format ([example](#) from 1983).

```
NET : 44.0.0.0 : AMPRNET :
```

```
NET : 45.0.0.0 : C3-PR :
```

```
NET : 46.0.0.0 : UCB-ETHER :
```

```
NET : 47.0.0.0 : SAC-PR-TEMP :
```

```
HOST : 46.0.0.4 : UCBARPA : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

```
HOST : 46.0.0.5 : UCBCAD : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

```
HOST : 46.0.0.6 : UCBERNIE : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

```
HOST : 46.0.0.7 : UCBMONET : VAX-11/750 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

```
HOST : 46.0.0.9 : UCBEVAX : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

```
HOST : 46.0.0.10 : UCBVAX : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,UDP :
```

Improved situation for humans!

- Rather than copying a file via FTP from 46.0.0.10...
- Can now copy a file from UCBVAX.
 - Berkeley had a /8!
- But this situation didn't scale for NIC, and was fragile.
 - Significant work to keep adding to the hosts file (especially as there were more workstations)!
 - Each location had to copy a version of the hosts file from the NIC (via FTP).
 - And it was getting bigger!
 - Could end up with a partial hosts file.

Questions?

The Domain Name System

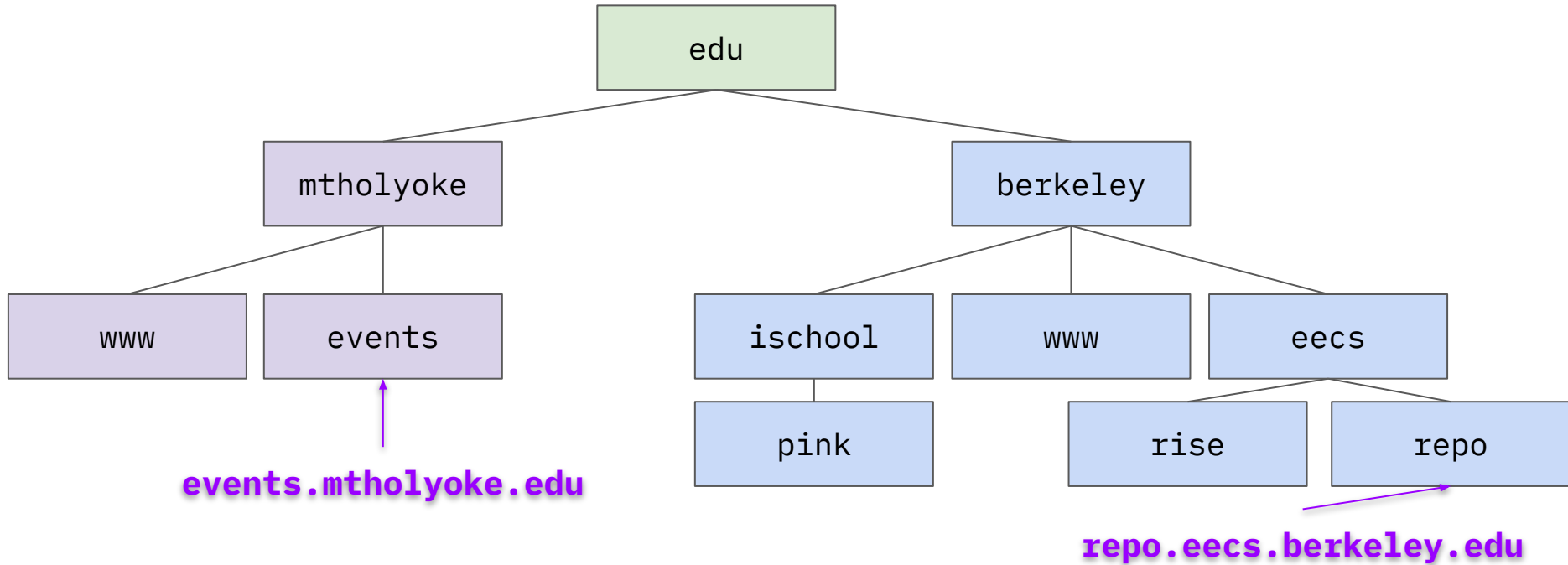
- Rather than have a centralised file that defines all the hosts – have a system that allows human-friendly names → IP addresses.
- Must deal with scale!
 - Many hosts/names.
 - Many lookups.
 - Many updates.
- Highly available.
 - No single point of failure (like the FTP server)
- Perform well
 - Communication generally starts with a name lookup!

The Domain Name System (DNS)

- Proposed in RFC882 (1983).
- We use this system (with some modifications – but not many) today!

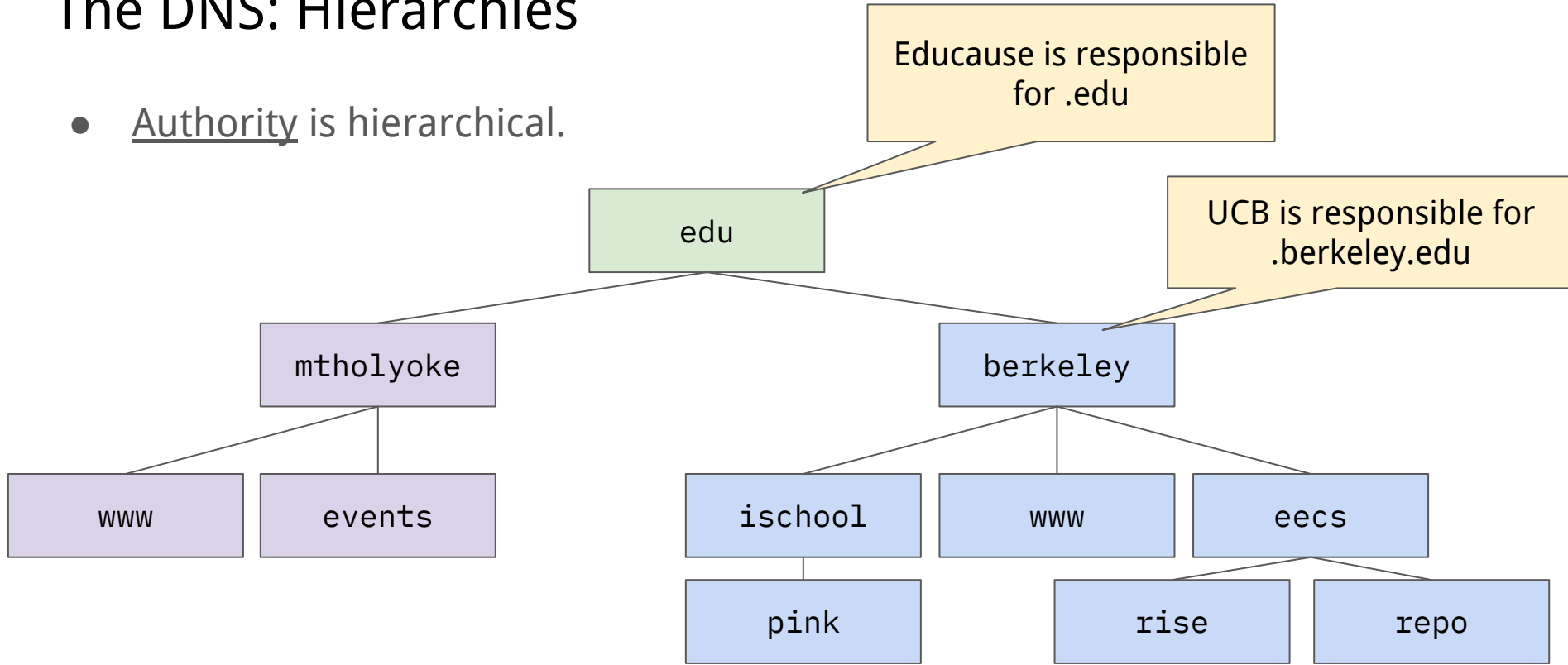
The DNS: Hierarchies

- Names are hierarchical.



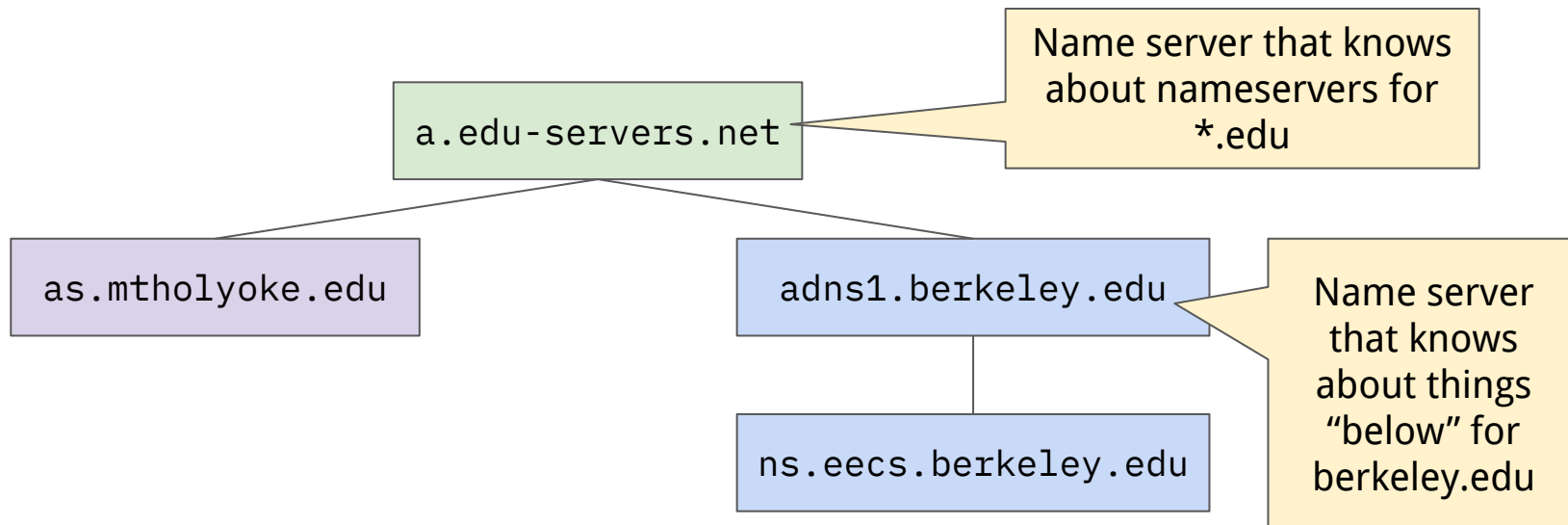
The DNS: Hierarchies

- Authority is hierarchical.



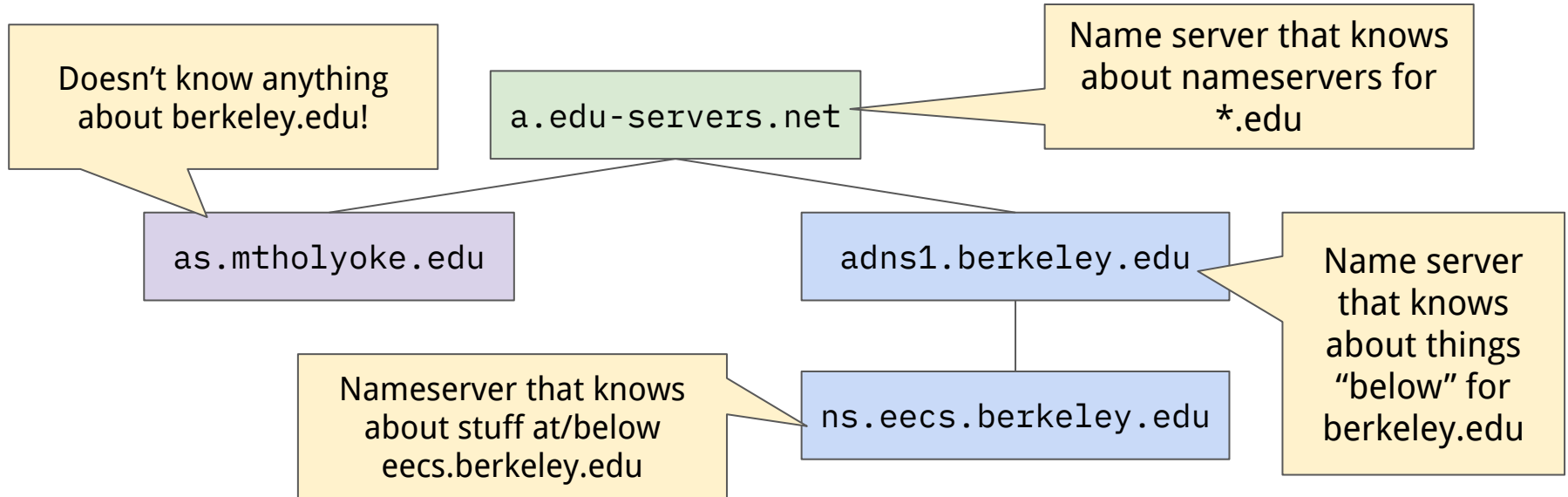
The DNS: Hierarchies

- Infrastructure is hierarchical.
- Not just one server that knows all the names.
 - Hierarchy of *name servers* which know parts of the tree.



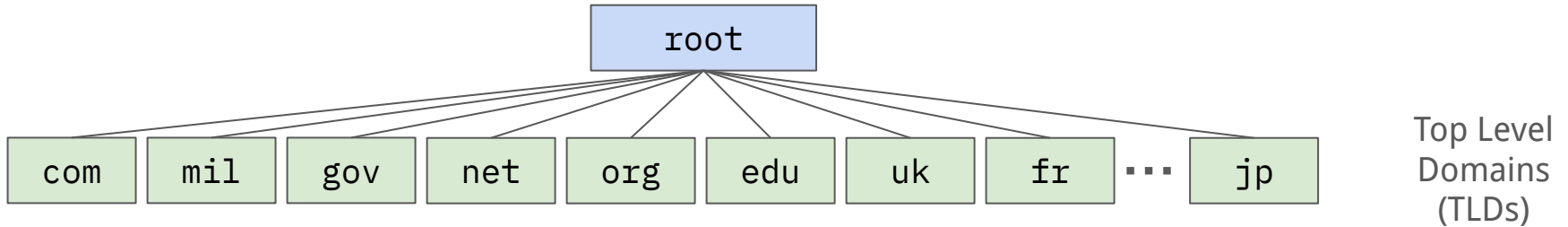
The DNS: Hierarchies

- Infrastructure is hierarchical.
- Not just one server that knows all the names.
 - Hierarchy of *name servers* which know parts of the hierarchy.



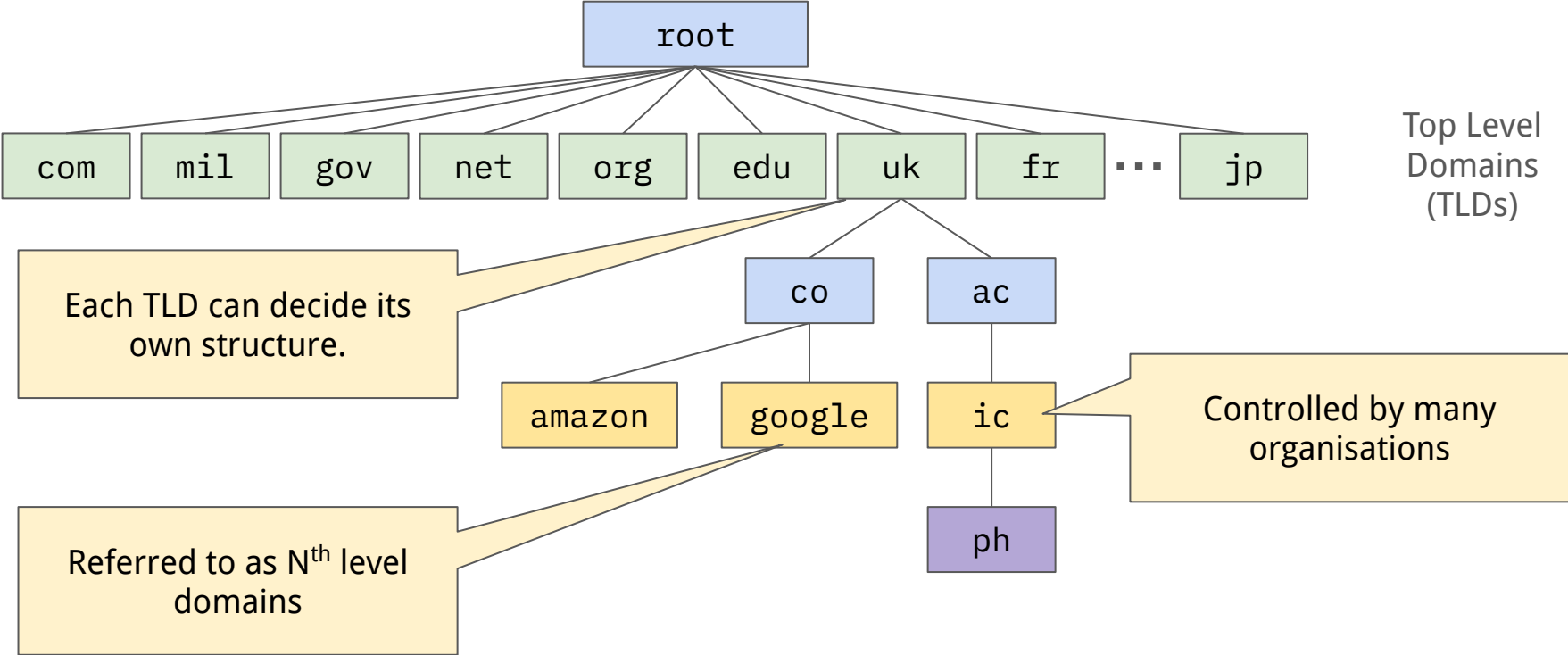
Questions?

DNS: The Bigger Picture



- DNS root
 - Controlled by ICANN.
- Top Level Domains (TLDs)
 - Controlled by various parties.
 - Historically, relatively few (.com, .net, .org, and country specific) but more recently many more!
 - 1590 as of 2024-03!

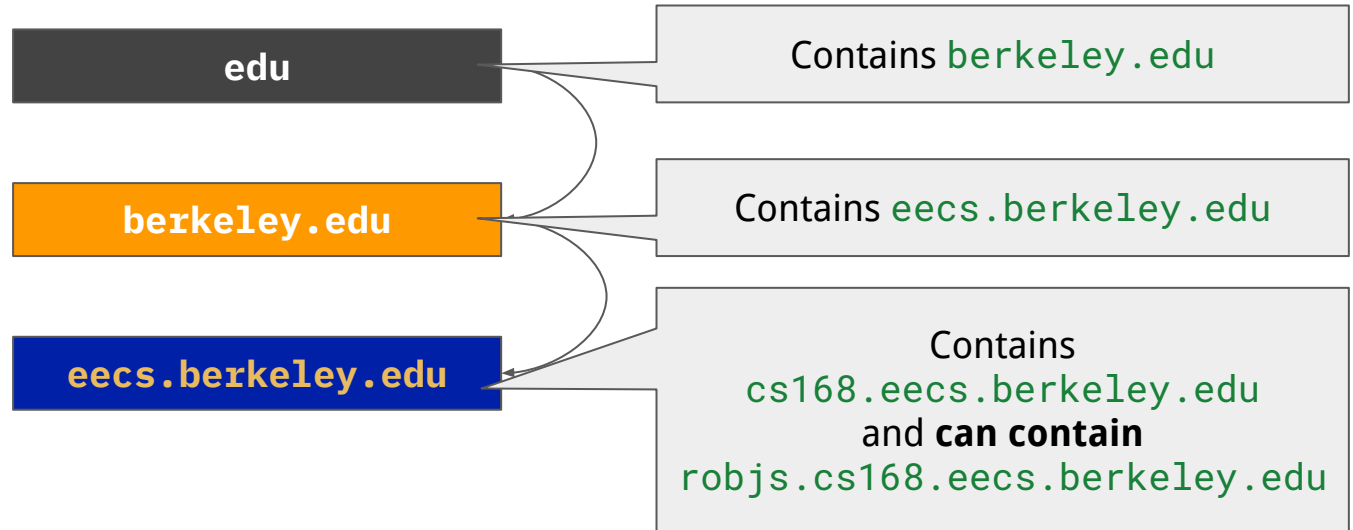
DNS: The Bigger Picture



DNS: Zones, Authority and Delegation

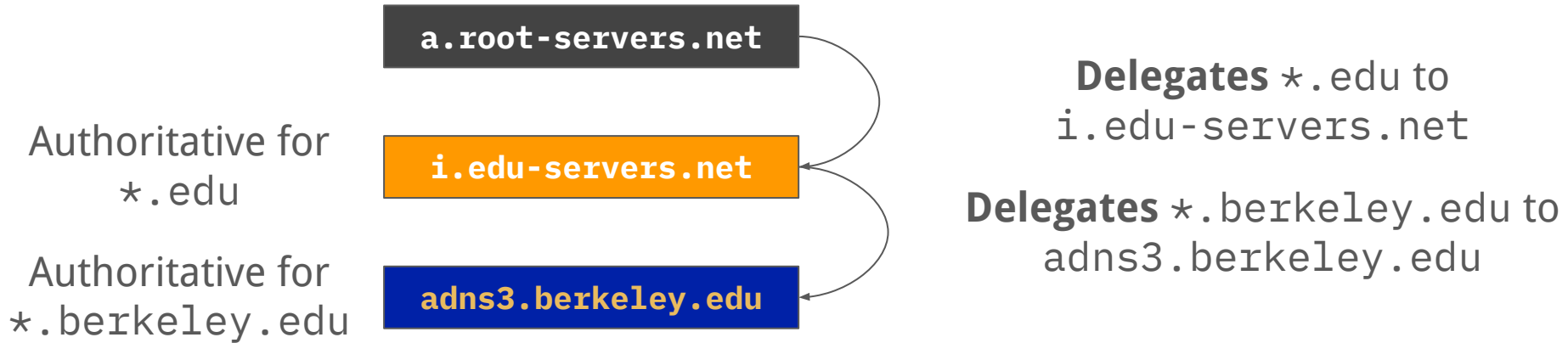
- A *zone* is the basic unit of the DNS (e.g., **edu** is a zone).
 - Each zone contains records.
- Each zone is controlled by an administrative authority responsible for some part of the hierarchy.
- A zone is authoritative for how names within that part of the hierarchy are controlled.
- You can choose to delegate authority to another nameserver within a zone.

DNS: Zones, Authority and Delegation



- Each zone defines entries that are only beneath it in the hierarchy.
 - May contain multiple levels of the hierarchy.


DNS: Zones, Authority and Delegation



- Allows different administrative authorities to be responsible for different parts of the hierarchy.
 - e.g., Educause (*.edu) don't have to be aware of what is happening in berkeley.edu.

Questions?

DNS: Name Lookup

- Iterative resolution process.
 - Start with root name server.
 - Ask for the name you want.
 - If it has an answer, you're done!
 - If not, it will redirect you to the next nameserver to ask.
- 

DNS: Name Lookup

- Example: let's look up eecs.berkeley.edu.
 - You can do this from your laptop.
 - `dig +trace @a.root-servers.net eecs.berkeley.edu`

```
.           518400  IN  NS  e.root-servers.net.
.           518400  IN  NS  h.root-servers.net.
.           518400  IN  NS  l.root-servers.net.

edu.       172800  IN  NS  a.edu-servers.net.
edu.       172800  IN  NS  b.edu-servers.net.
edu.       172800  IN  NS  c.edu-servers.net.
```

Where to find the root zone (".")

.edu is **delegated** to these nameservers – go ask them.

```
;; Received 1176 bytes from 2001:7fd::1#53(k.root-servers.net)
```

Answer from one of the root servers

DNS: Name Lookup

- Example: let's look up eecs.berkeley.edu.
 - You can do this from your laptop.
 - `dig +trace @a.root-servers.net eecs.berkeley.edu`

```
berkeley.edu.      172800   IN      NS      adns1.berkeley.edu.  
berkeley.edu.      172800   IN      NS      adns2.berkeley.edu.  
berkeley.edu.      172800   IN      NS      adns3.berkeley.edu.
```

```
;; Received 377 bytes from 2001:503:1::30#53(i.edu-servers.net) in 16 ms
```

Answer from
i.edu-servers.net

berkeley.edu is delegated to
these nameservers – go ask
them

DNS: Name Lookup

- Example: let's look up eecs.berkeley.edu.
 - You can do this from your laptop.
 - `dig +trace @a.root-servers.net eecs.berkeley.edu`

```
eecs.berkeley.edu. 86400      IN      A       141.193.213.10
eecs.berkeley.edu. 86400      IN      A       141.193.213.11
```

```
;; Received 78 bytes from 192.107.102.142#53(adns3.berkeley.edu) in 27 ms
```

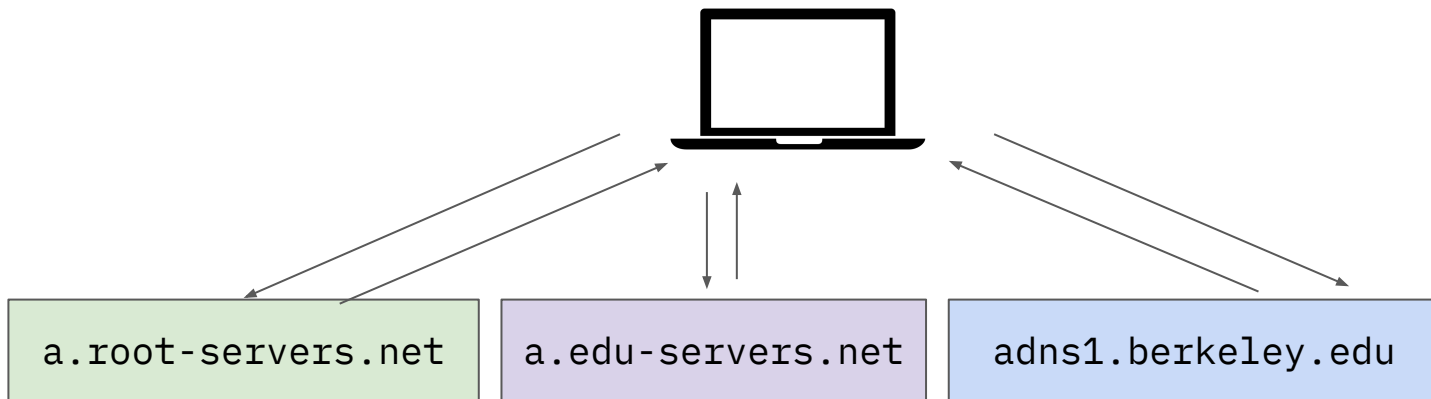
I'm authoritative for
berkeley.edu! Here's the IP
address...

Answer from Berkeley's
name servers

Questions?

Who does these lookups?

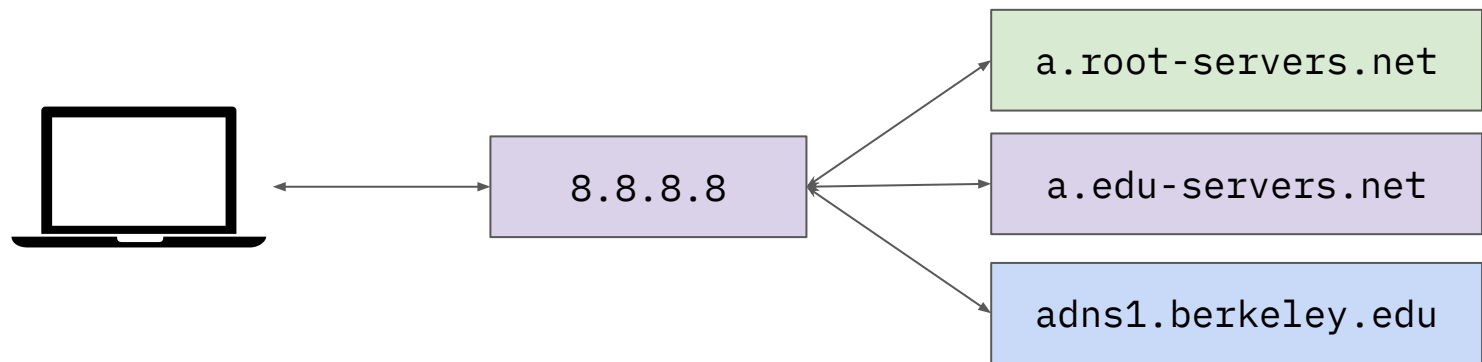
- Originally – done by the end host.



Iterative queries (what `+trace` did for us!)

Who does these lookups?

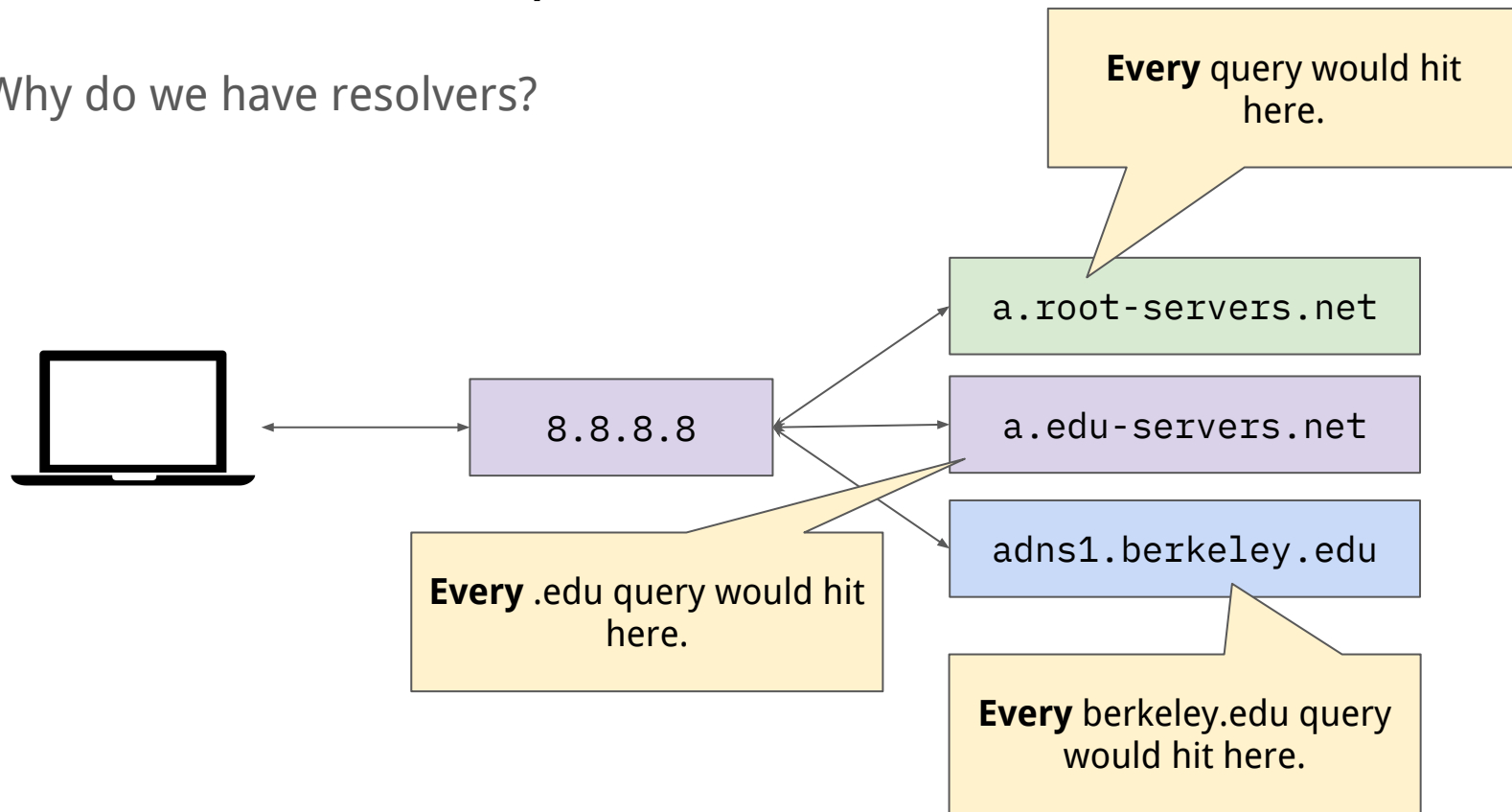
- Today, usually done by a resolving name server (“resolver”).



Recursive query made to a resolver.
Well known resolvers – 1.1.1.1 (CloudFlare), 8.8.8.8 (Google).

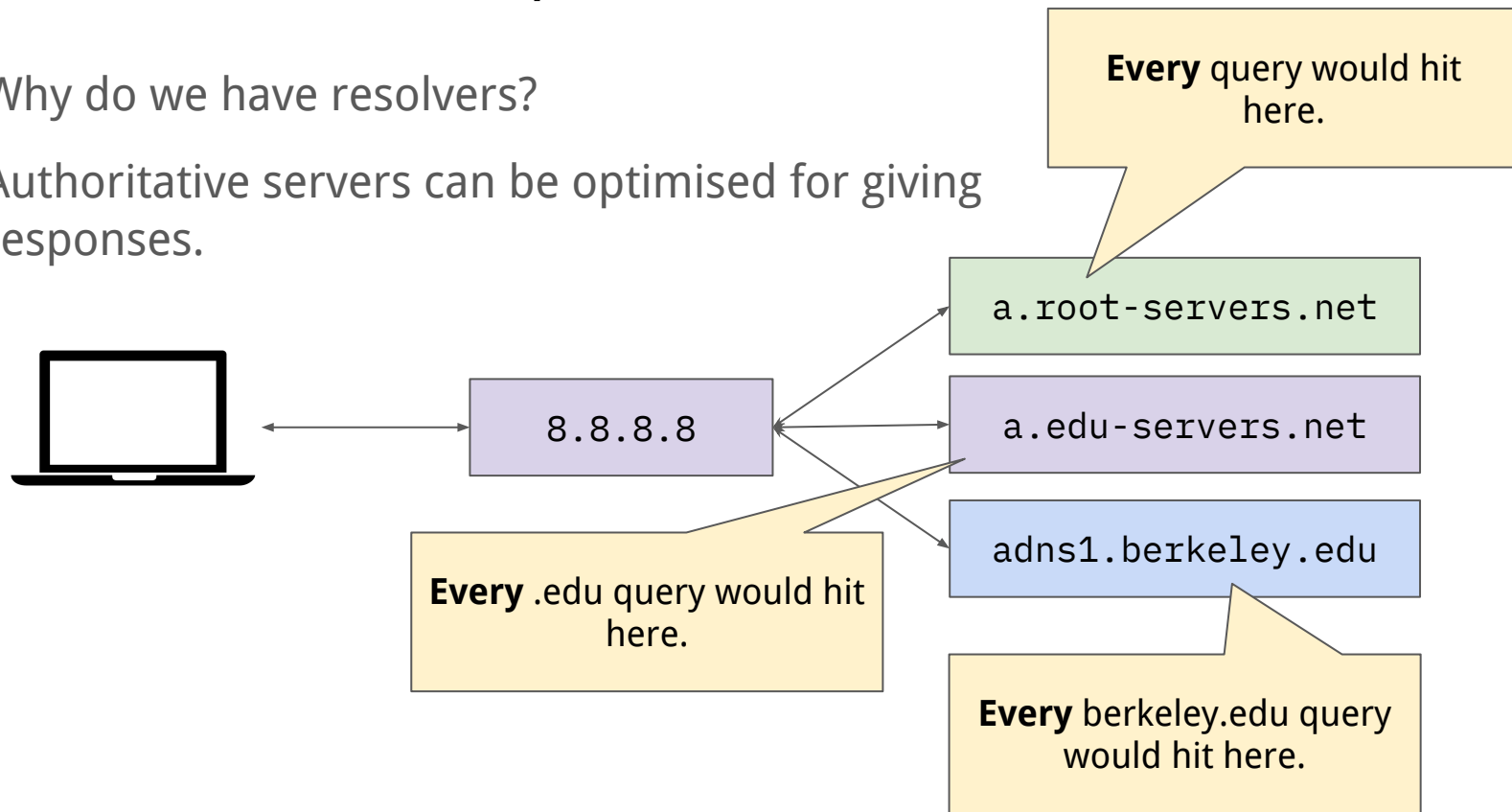
Who does these lookups?

- Why do we have resolvers?



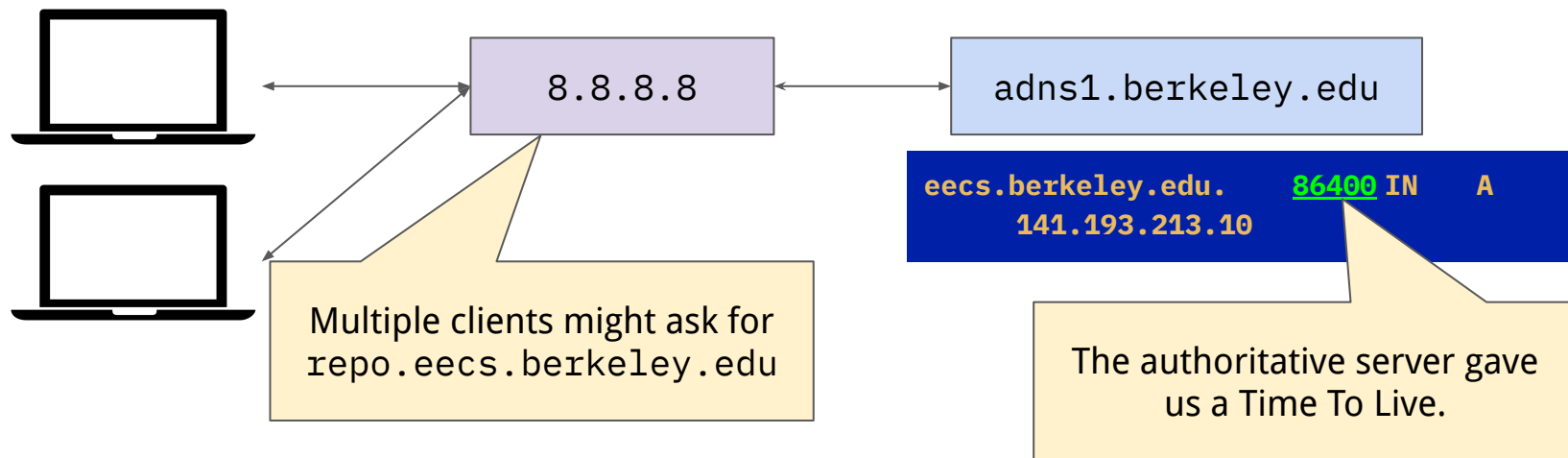
Who does these lookups?

- Why do we have resolvers?
- Authoritative servers can be optimised for giving responses.



Who does these lookups?

- Why do we have resolvers?
- Resolvers can cache entries according to the record's TTL – reduces the load on authoritative servers.



Chicken-and-egg: How do we know addresses?

- For clients – we need to know our resolver address.
 - Either programmed yourself – well-known, memorable addresses – e.g., 8.8.8.8.
 - Or learnt when addresses are assigned, DHCP, SLAAC (we'll come back to this!)
- For resolvers – where are the root servers?
 - Root hints file – <https://www.internic.net/domain/named.root>
 - Allows software that needs to make DNS queries and can't rely on a resolver to know where to start.

```
; OPERATED BY RIPE NCC
;
.           3600000      NS      K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET. 3600000      A       193.0.14.129
K.ROOT-SERVERS.NET. 3600000      AAAA    2001:7fd::1
```

Questions?

DNS: Zone Availability

- We've been pretending that each zone just has one name server.
- Zones "must" have two authoritative nameservers.
 - Ensures availability of that zone.
- Such nameservers work in a primary/secondary model – one server is the primary, and the zone contents is transferred to the secondaries.

```
▶ dig berkeley.edu NS
```

```
;; ANSWER SECTION:
```

```
berkeley.edu.      10800      IN        NS        adns2.berkeley.edu.  
berkeley.edu.      10800      IN        NS        adns1.berkeley.edu.  
berkeley.edu.      10800      IN        NS        adns3.berkeley.edu.
```

DNS: Root Server Availability

- If the root servers were unavailable, this would be a huge problem!
- After the TTL of a record timed out we would not know where to go to find any zone!
 - Including TLDs like `.edu` being redirected to their nameservers.
- Root servers are therefore highly-available.
 - There are 13 root servers.
 - But there are many, many instances of these root servers.
- We use a trick called “anycast” – use the same IP address in many places on the Internet.
 - So one address for `k.root-servers.net` might be many servers in many different places.

DNS: k.root-servers.net

```
K.root-servers.net. 3600000 IN A 193.0.14.129  
K.root-servers.net. 3600000 IN AAAA 2001:7fd::1
```

Very long TTL - ~42 days.

One IPv4 address!

One IPv6 address!

DNS: k.root-servers.net

K.root-servers.net. 3600000 IN A 193.0.14.129

K.root-servers.net. 3600000 IN AAAA 2001:7fd::1



DNS: f.root-servers.net

K.root-servers.net. 3600000 IN A 193.0.14.129

K.root-servers.net. 3600000 IN AAAA 2001:7fd::1

Which instance am I actually using?

► dig +short +norec @k.root-servers.net hostname.bind chaos txt
"ns1.us-mia.k.ripe.net"

DNS: Root Server Availability

- Duplication of server infrastructure means that the root servers are extremely highly available.
- `f.root-servers.net` has >3,000 instances.
- Lots of cooperation between network operators to keep the root servers highly available.

Questions?

The DNS Protocol

DNS Details

- APIs – how does DNS look from an developer perspective?
- Servers – what software acts as an authoritative or recursive DNS server?
- The DNS protocol – what do DNS network packets look like?

DNS: APIs

- Relatively simple, common APIs that are available in almost all languages.
- `result = gethostbyname("foo.com")`
 - Limited to IPv4.
 - Deprecated but very common.
- `error = getaddrinfo("example.com", NULL, NULL, &result)`
 - Replacement API
 - Supports more than IPv4.
- Usually just make requests to the OS' configured resolving DNS server.
 - All the complexities of the DNS hidden from the end developer.

DNS Servers

- Two types of servers that we might need to run...
- Authoritative – usually run by application providers or “hosting” providers.
 - A core part of owning a domain name.
 - Lots of use of AWS Route53, or nameservers of “domain registries” (Verisign).
- Recursive – usually run by ISPs.
 - As a core part of their Internet service.
 - Some popular recursive servers are run by large application providers (Cloudflare, Google).

DNS Servers: BIND

- Huge amount of DNS history here at Berkeley!
- First DNS server written for Unix was **BIND** (1984).
 - And berkeley.edu is the oldest .edu!
- Many alternatives - with optimisations for authoritative and recursive functionality.

The Berkeley Internet Name Domain Server

Douglas B. Terry, Mark Painter, David W. Riggle, and Songnian Zhou

Computer Systems Research Group
Computer Science Division

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

The Berkeley Internet Name Domain (BIND) Server allows a

The DNS Protocol

- Client/server design.
 - Client is often a user host, but can be a recursive server.
- Client sends a query, server replies with a response.
- Server typically listens on a well-known **UDP** port: 53.

The DNS Protocol

- Why UDP?
 - Saves RTT for TCP connection establishment.
 - TCP requires servers to keep state per connection – lots of connections.
 - No need for ordered stream abstraction - a single packet is often all that is required!

```
▶ tshark -T fields -E separator=, -e ipv6.src -e ipv6.dst -e frame.len -e _ws.col.Info  
udp port 53
```

```
2001:5a8:429e:9f00:8003:4ac:9eb5:8e97,2001:5a8:429e:9f00:3e28:6dff:fe67:7f19,101,Standard  
d query 0x3d90 A google.com OPT
```

```
2001:5a8:429e:9f00:3e28:6dff:fe67:7f19,2001:5a8:429e:9f00:8003:4ac:9eb5:8e97,117,Standard  
d query response 0x3d90 A google.com A 142.251.214.142 OPT
```

The DNS Protocol

- Why UDP?
 - Saves RTT for TCP connection establishment.
 - TCP requires servers to keep state per connection – lots of connections.
 - No need for ordered stream abstraction - a single packet is often all that is required!

Query is one 101 byte packet.

```
▶ tshark -T fields -E separator=, -e ipv6.src -e ipv6.dst -e info  
udp port 53
```

```
2001:5a8:429e:9f00:8003:4ac:9eb5:8e97,2001:5a8:429e:9f00:3e28:6dff:fe67:7f19,101,Standard query 0x3d90 A google.com OPT
```

```
2001:5a8:429e:9f00:3e28:6dff:fe67:7f19,2001:5a8:429e:9f00:8003:4ac:9eb5:8e97,117,Standard query response 0x3d90 A google.com A 142.251.214.142 OPT
```

Response is one 117 byte packet

The DNS Protocol: UDP?

- Wait... isn't UDP unreliable? What happens if packets are dropped?
- Simple timeout/retry mechanism.
- Varies from OS-to-OS, but can be fairly slow.
- Ensuring that your resolver is available, and performant is important.
 - Often this is a function your home router might provide.
 - And you can have multiple resolvers to fall back between.

The DNS Protocol: TCP and further.

- UDP also causes us complexities if we have very large responses.
- Generally, we don't!
 - Remember – our google.com query was <120 bytes!
- Transferring zones between primary authoritative servers and secondary ones results in larger responses.
 - These queries are often done over TCP.
- Recent advances in DNS start to look how to implement it over encrypted transport.
 - TCP and UDP are “plaintext” – someone in the middle can see what you're doing!

Questions?

The DNS Protocol

- All messages share the same basic format.
- Messages may be:
 - A query - QR bit in header is 0
 - A response - QR bit in header is 1
- Theoretically, there are different query types.
 - **IQUERY**.
 - Show me the names for this address.
 - Obsoleted in 2002 (RFC3425) – not implemented, or disabled.
 - **STATUS** - not really defined.
 - **QUERY** - used for basically everything.
- The **RD** bit indicates *recursion desired* - do a recursive lookup.

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)
- Many different types of record.
- Focusing on the primary goal is to map names to IP addresses.
- Three types of records we need:
 - **A** records (IPv4 address)
 - **AAAA** records (IPv6 address)
 - **NS** records (name server)

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)
- For **A** and **AAAA** records, this is the hostname of interest.
 - e.g., `www.google.com`.

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)
- The actual value associated with the record.
- For **A** records, this is the IPv4 address.
- For **AAAA** records, this is the IPv6 address.
- For **NS** records, this is the name server's name.

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)
- How long in seconds the record response is valid for.

DNS: Resource Records

- DNS zone data is stored in *resource records* (RRs)
- These are essentially a tuple:
 - (type, name, value, ttl, class)
- We'll ignore this – intended to be used for places where the DNS was used outside of the Internet.
 - Not aware of any cases where this is different!

Back to our `repo.eecs.berkeley.edu` query.

- What records do we see?

DNS: Name Lookup

name - the root zone is called "."

ttl - how long this response is valid

type - in this case a nameserver (NS) record

```
.           518400    IN      NS      e.root-servers.net.
.           518400    IN      NS      h.root-servers.net.
.           518400    IN      NS      l.root-servers.net.

edu.        172800    IN      NS      a.edu-servers.net.
edu.        172800    IN      NS      b.edu-servers.net.
edu.        172800    IN      NS      c.edu-servers.net.
```

value - the name of the next nameserver we are being referred to.

We can have multiple of each type of record associated with the same name.

```
;; Received 1176 bytes from 2001:7fd::1#53(k.root-servers.net)
```


DNS: Name Lookup

Name that we were looking up - with associated TTL and value.

Huh? New record type CNAME - this is an **alias** to another name.

```
;; ANSWER SECTION:
```

```
repo.eecs.berkeley.edu. 21600 IN CNAME repo-2.eecs.berkeley.edu.  
repo-2.eecs.berkeley.edu. 21600 IN A 128.32.138.46
```

A record with the value of the IPv4 address for repo-2.eecs...

DNS: Name Lookup

```
;; QUESTION SECTION:  
;repo-2.eecs.berkeley.edu. IN AAAA  
  
;; AUTHORITY SECTION:  
eecs.berkeley.edu. 1791 IN SOA ns.eecs.berkeley.edu. dns.eecs.berkeley.edu.  
100012225 10887 3600 604800 86400
```

When we ask for the **AAAA** record, one doesn't exist.

So we only see the **Authority** section of the response (what zone version is this...), and no **Answer** section.

Questions?

DNS: Email

- How do we know where to send an email when we write to sylvia@eecs.berkeley.edu?
- We need a different record type.
 - **MX** - mail exchanger record – tells us where to send email messages.

```
▶ dig eecs.berkeley.edu MX
```

```
;; ANSWER SECTION:
```

```
eecs.berkeley.edu. 10549      IN      MX      1 aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      5 alt1.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      5 alt2.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      10 alt4.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      10 alt3.aspmx.l.google.com.
```

DNS: Email

- How do we know where to send an email when we write to sylvia@eecs.berkeley.edu?
- We need a different record type.
 - **MX** - mail exchanger record – tells us where to send email

Where to connect to when sending an email to this domain – doesn't have to be within the zone.

```
▶ dig eecs.berkeley.edu MX
```

```
;; ANSWER SECTION:
```

```
eecs.berkeley.edu. 10549      IN      MX      1 aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      5 alt1.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      5 alt2.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      10 alt4.aspmx.l.google.com.  
eecs.berkeley.edu. 10549      IN      MX      10 alt3.aspmx.l.google.com.
```

DNS: Email

- How do we know where to send an email when we write to sylvia@eecs.berkeley.edu?
- We need a different record type.
 - **MX** - mail exchanger record – tells us where to send email

Where to connect to when sending an email to this domain – doesn't have to be within the zone.

```
▶ dig eecs.berkeley.edu MX
```

```
;; ANSWER SECTION:
```

```
eecs.berkeley.edu. 10549 IN MX 1 aspmx.l.google.com.  
eecs.berkeley.edu. 10549 IN MX 5 alt1.aspmx.l.google.com.  
eecs.berkeley.edu. 10549 IN MX 5 alt2.aspmx.l.google.com.  
eecs.berkeley.edu. 10549 IN MX 10 alt4.aspmx.l.google.com.  
eecs.berkeley.edu. 10549 IN MX 10 alt3.aspmx.l.google.com.
```

DNS: Email

- How do we know where to send an email when we write to sylvia@eecs.berkeley.edu?
- We need a different record type.
 - **MX** - mail exchanger record – tells us where to send email

Where to connect to when sending an email to this domain – doesn't have to be within the zone.

```
▶ dig eecs.berkeley.edu MX
```

Priority within the value – MX records allow us to understand which mail servers to try in which order.

```
IN      MX      1  aspmx.l.google.com.  
TN      MX      5  alt1.aspmx.l.google.com.  
IN      MX      5  alt2.aspmx.l.google.com.  
IN      MX      10 alt4.aspmx.l.google.com.  
eecs.berkeley.edu. 10549 IN      MX      10 alt3.aspmx.l.google.com.
```

What happens with multiple responses?

- In some cases, there are multiple responses for an **A** or **AAAA** query.

```
;; ANSWER SECTION:
```

```
microsoft.com.      1999 IN    A        20.112.250.133  
microsoft.com.      1999 IN    A        20.231.239.246  
microsoft.com.      1999 IN    A        20.76.201.171  
microsoft.com.      1999 IN    A        20.70.246.20  
microsoft.com.      1999 IN    A        20.236.44.162
```

- The value does not contain a precedence for **A/AAAA** (different to **MX**).
- Client can therefore pick one, the server shuffles the order.
- Coarse-grained load-balancing, and simple resiliency.

DNS as a Simple Load-Balancer

- When we look at `google.com`... from my home ISP

```
▶ dig google.com +short  
142.251.46.238
```

DNS as a Simple Load-Balancer

- When we look at `google.com`... from my home ISP

```
▶ dig google.com +short  
142.251.46.238
```

- When we look at `google.com`... from a machine in Oregon.

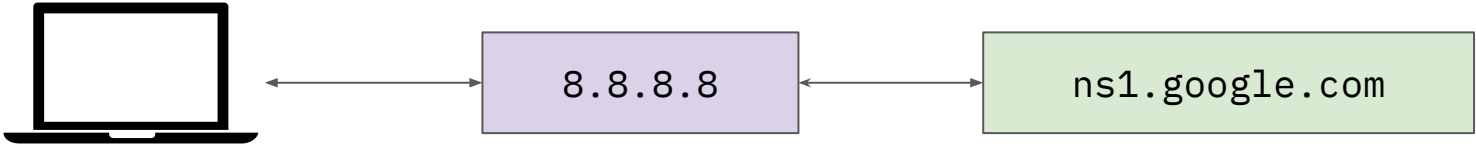
```
rjs@jumhost:~$ dig google.com +short  
74.125.135.113  
74.125.135.100  
74.125.135.102  
74.125.135.101  
74.125.135.138  
74.125.135.139
```

DNS as a Simple Load Balancer

- Sometimes DNS gives us responses based on where we are querying from.
- This means the authoritative server has some logic to say “if this query is from X then respond with Y”.
- What could X be?
 - The recursive resolver that is querying the authoritative server.
 - Most clients don't query the authoritative server directly
 - The end client that is querying the resolver
 - Requires extension to DNS to carry the *client subnet*.
 - The geographical location of the end user – needs us to map from IP address to physical location.
 - Needs a mapping database like [MaxMind](#).

How does geographical load balancing work?

Query: youtube.com AAAA?
Client is from: 2001:5a8:429e::/48



Your IP Address	2001:5a8:429e:9f00:8003:4ac:9eb5:8e97
Location -----	San Francisco, California, United States (US), North America

youtube.com. 300 IN AAAA
2607:f8b0:4005:80d::200e

How does geographical load balancing work?

- Client information is used to determine something about the “right” place to route the client to.
- Response is then given directing that client to the “nearest” server.
- Some guessing involved.
 - We don't know how close from a network perspective that user is to the geographical location.
 - We don't know what the performance to different servers is.
- Some proprietary logic required at the authoritative server.

DNS: Geographical Load Balancing Results

Use a different DNS record type – **PTR** – lets us assign a name to an IP address.

```
▶ host 2607:f8b0:4005:80d::200e
e.0.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.d.0.8.0.5.0.0.4.0.b.8.f.7.0.6.2.ip6.arpa domain
name pointer sfo03s25-in-x0e.1e100.net.
```

Mapped to **sfo...** – pretty close!

- Machine in Oregon got a different result – 2607:f8b0:400e:c0c::88.
- Comparing performance:
 - My laptop → SF result = 20 msec RTT.
 - My laptop → Oregon result = 35 msec RTT.
- Mapping logic gave us a way to map a client to a better performing server.

Recap

- The DNS was created to allow for name to IP address resolution – helping humans to access things on the Internet.
- It is a hierarchical system – where both zones and nameservers have a hierarchy that allows for delegation to authoritative entities.
- DNS is a simple protocol - most often used over UDP - that provides a way to query for particular record types.
- It extends beyond just address resolution into service resolution - e.g., mail servers - and can be used for load-balancing.