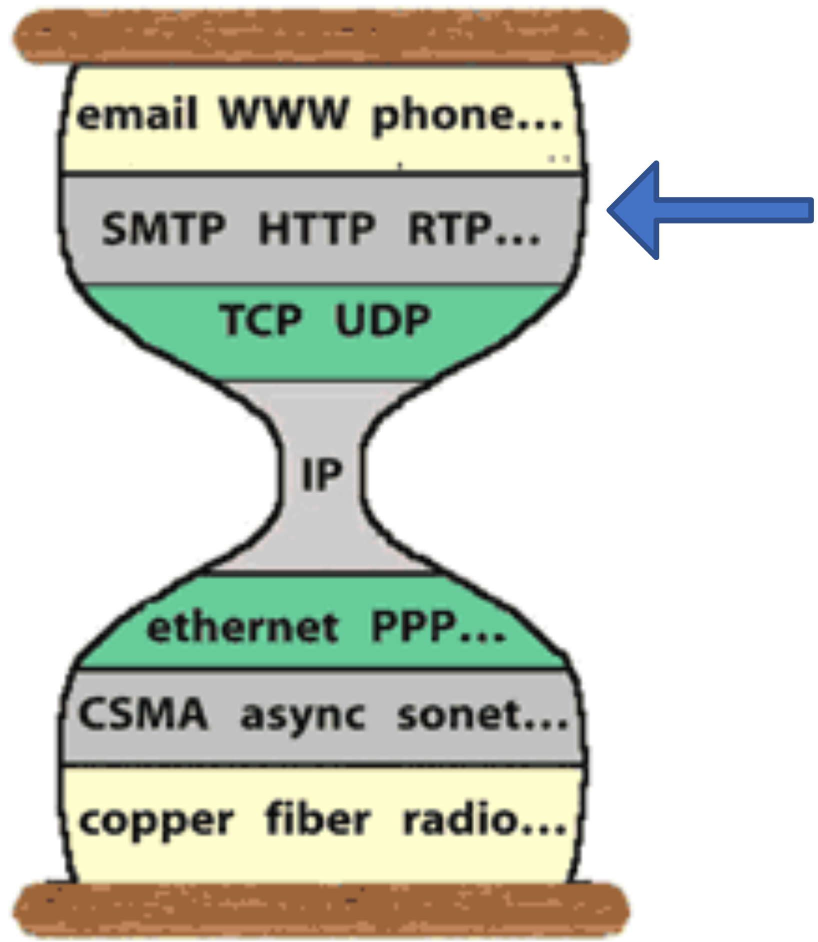
Lecture-3: HTTP and DNS

- HTTP/2 and Cookie

  (FYI)

- Domain Name System
  - Why we need it
  - What it is
  - How it works
  - What it has been used for

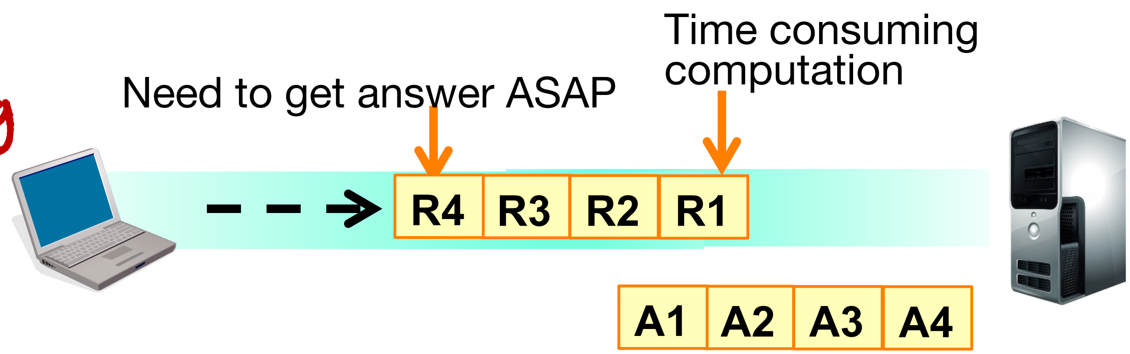


HTTP/1.1's performance issues

important

1. Head-of-line blocking: HTTP/1.1 handles all requests in strict sequential order
   - A request for a large file, or some dynamic computation, can take time, blocking all requests following it

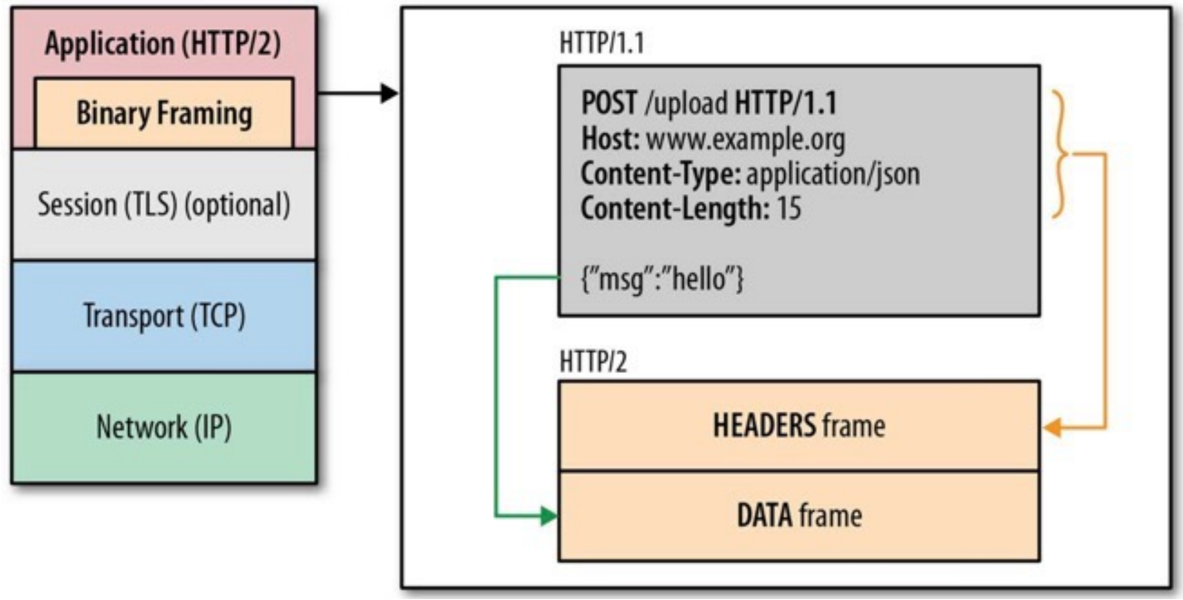
Header-of-line blocking

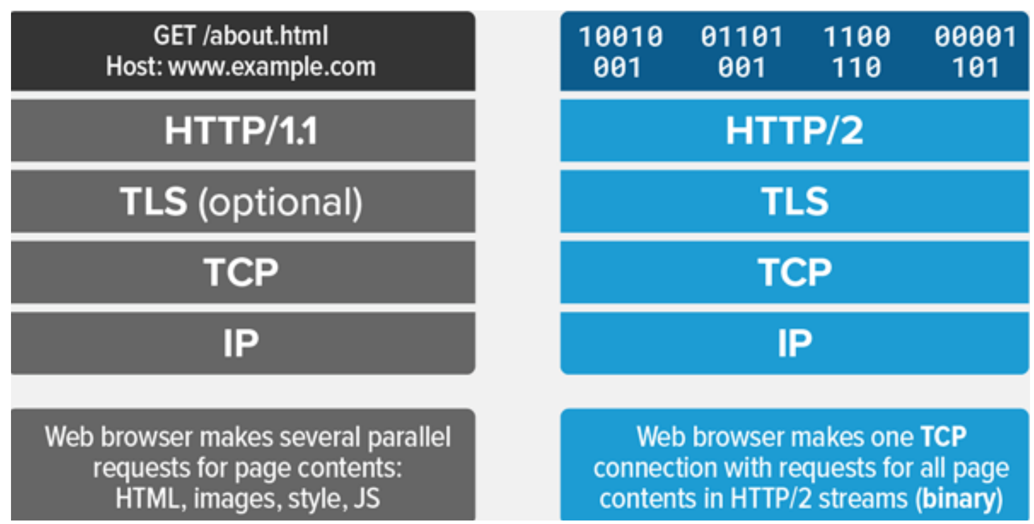


   - Work-around: open multiple TCP connections
2. Big size HTTP header with repetitive information carried in queries
   - No work-around

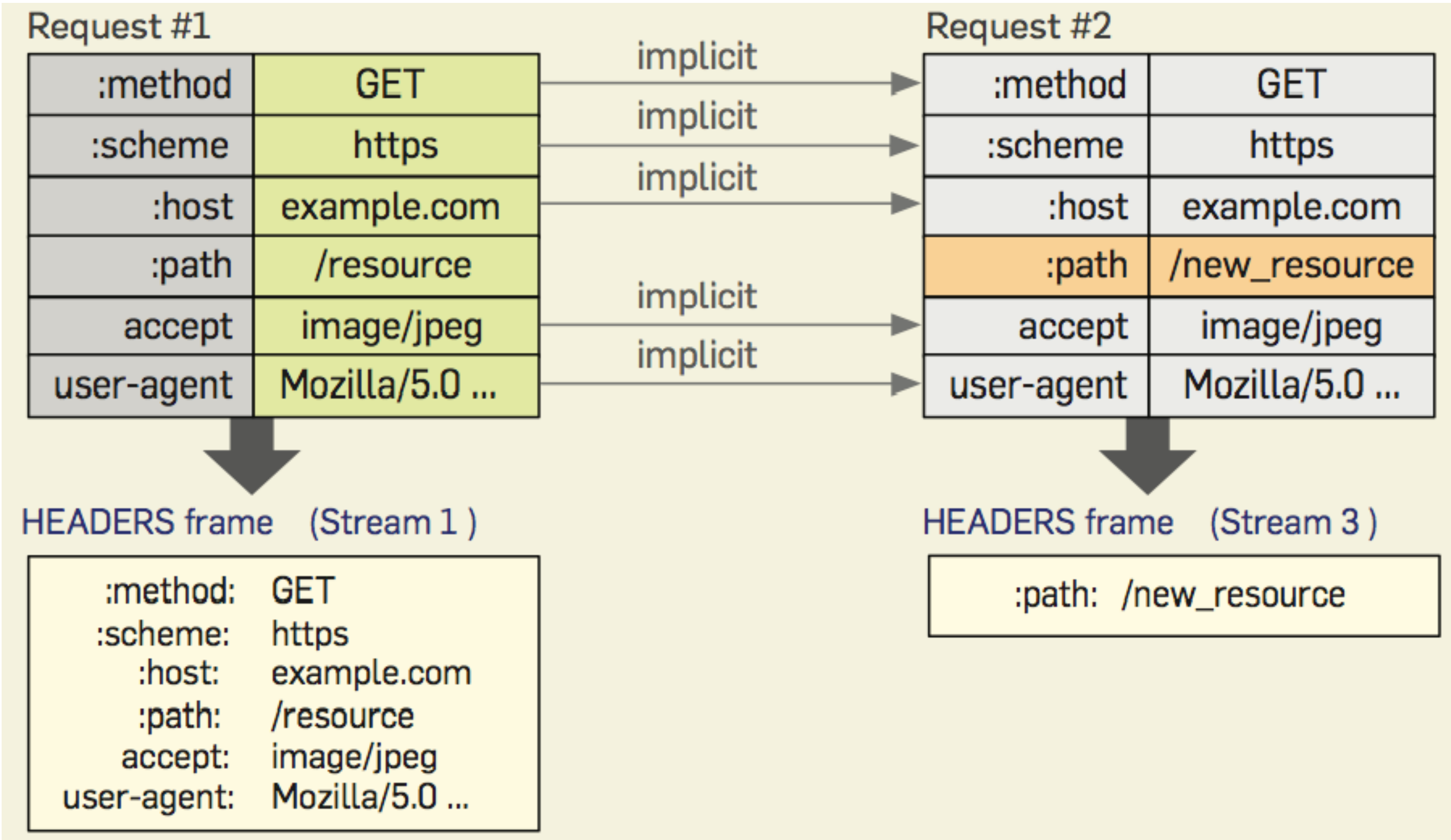
HTTP/2's major new features

- Binary encoding
- Header compression
- "frame" as the basic unit
- Use a single TCP connection between browser—server
  - Each HTTP request → a stream
  - streams are multiplexed, in priority order
- Server push





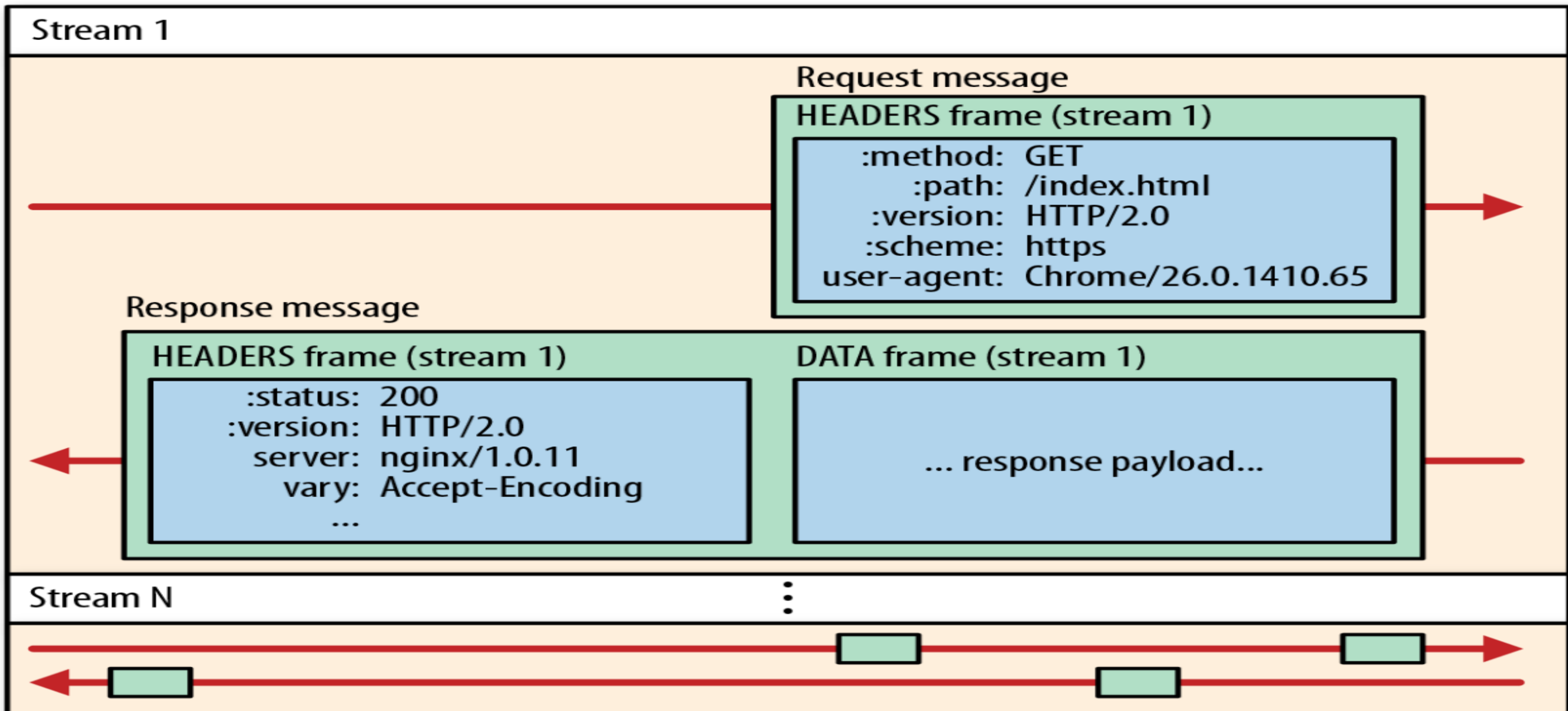
HTTP/2: Header Compression



- Both browser & server keep a header table until the TCP connection closes

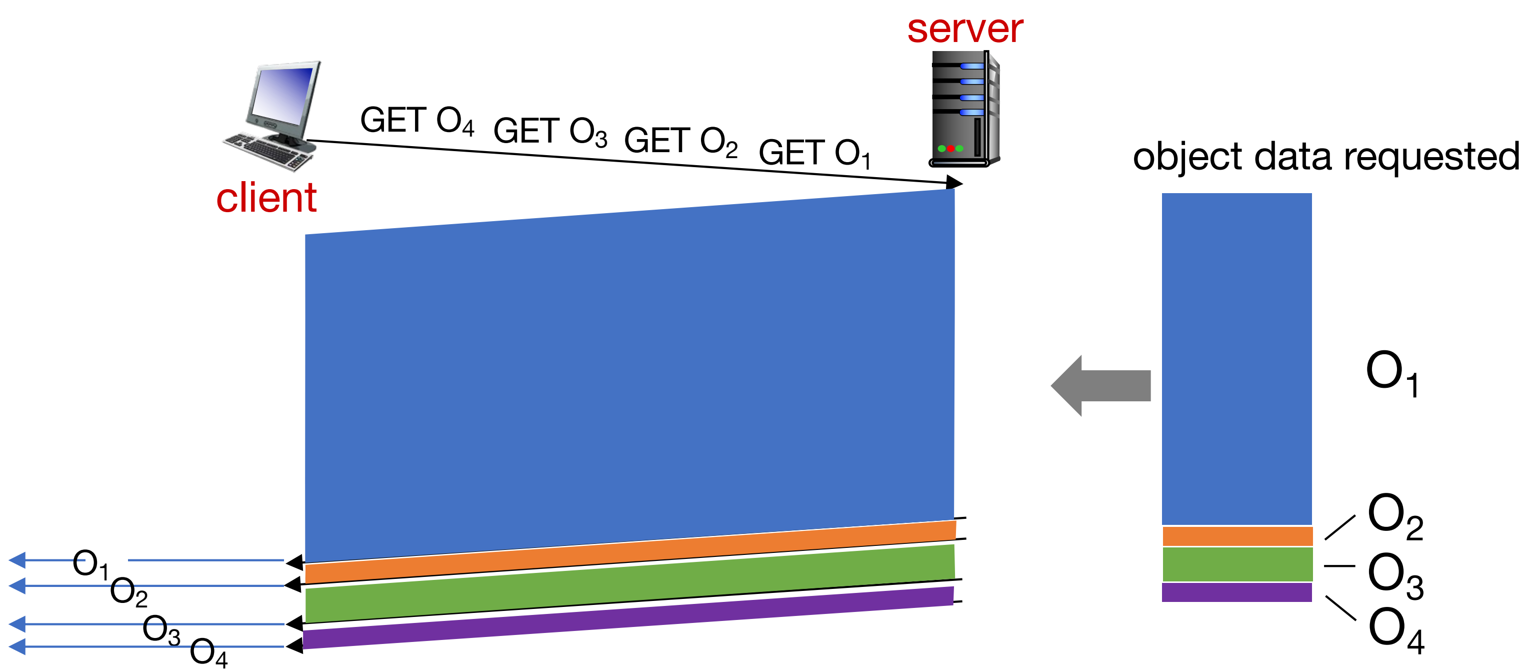
HTTP/2.0: Frame, Message, Stream

- Frame: basic communication unit
- Message: an HTTP request, or response
  - encoded in one or multiple frames
- Stream: a virtual channel with priority, carrying frames in both directions



HTTP/2: Mitigating HOL blocking

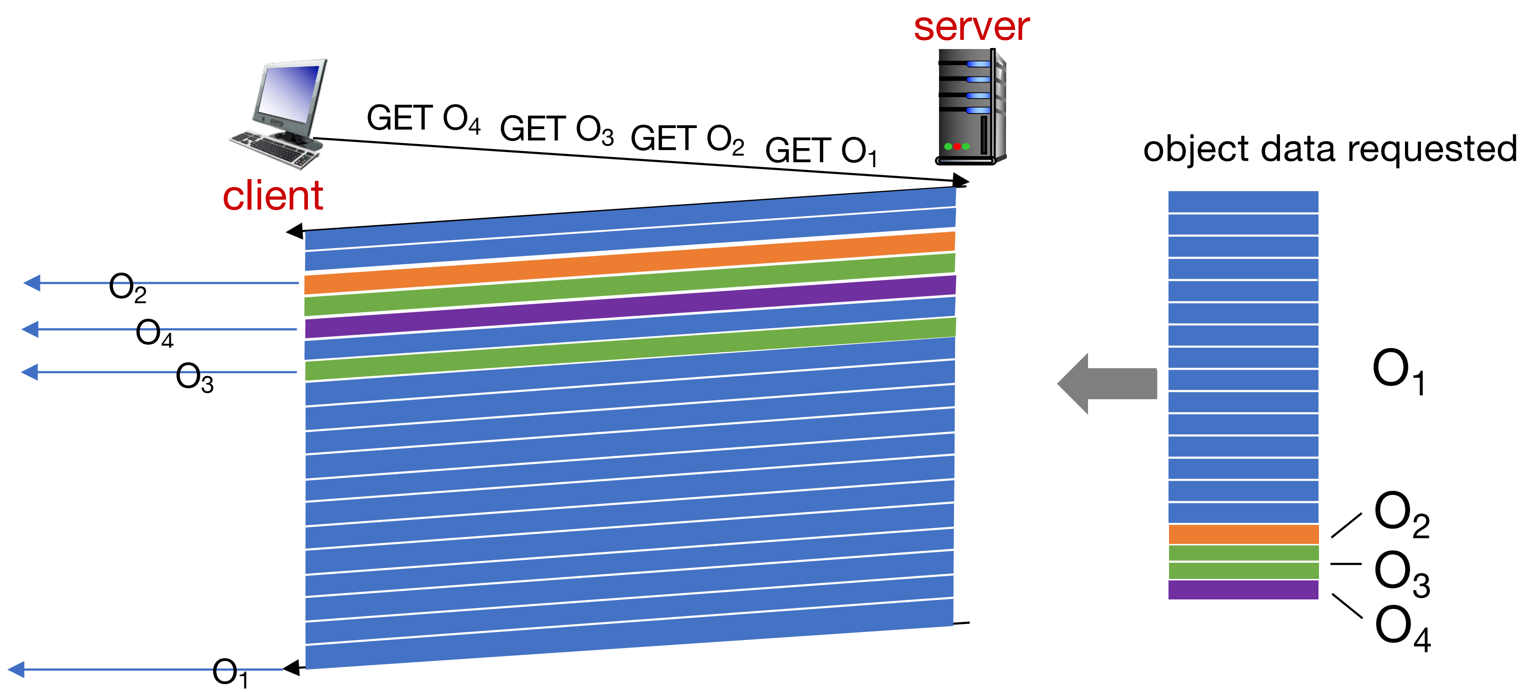
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: Mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



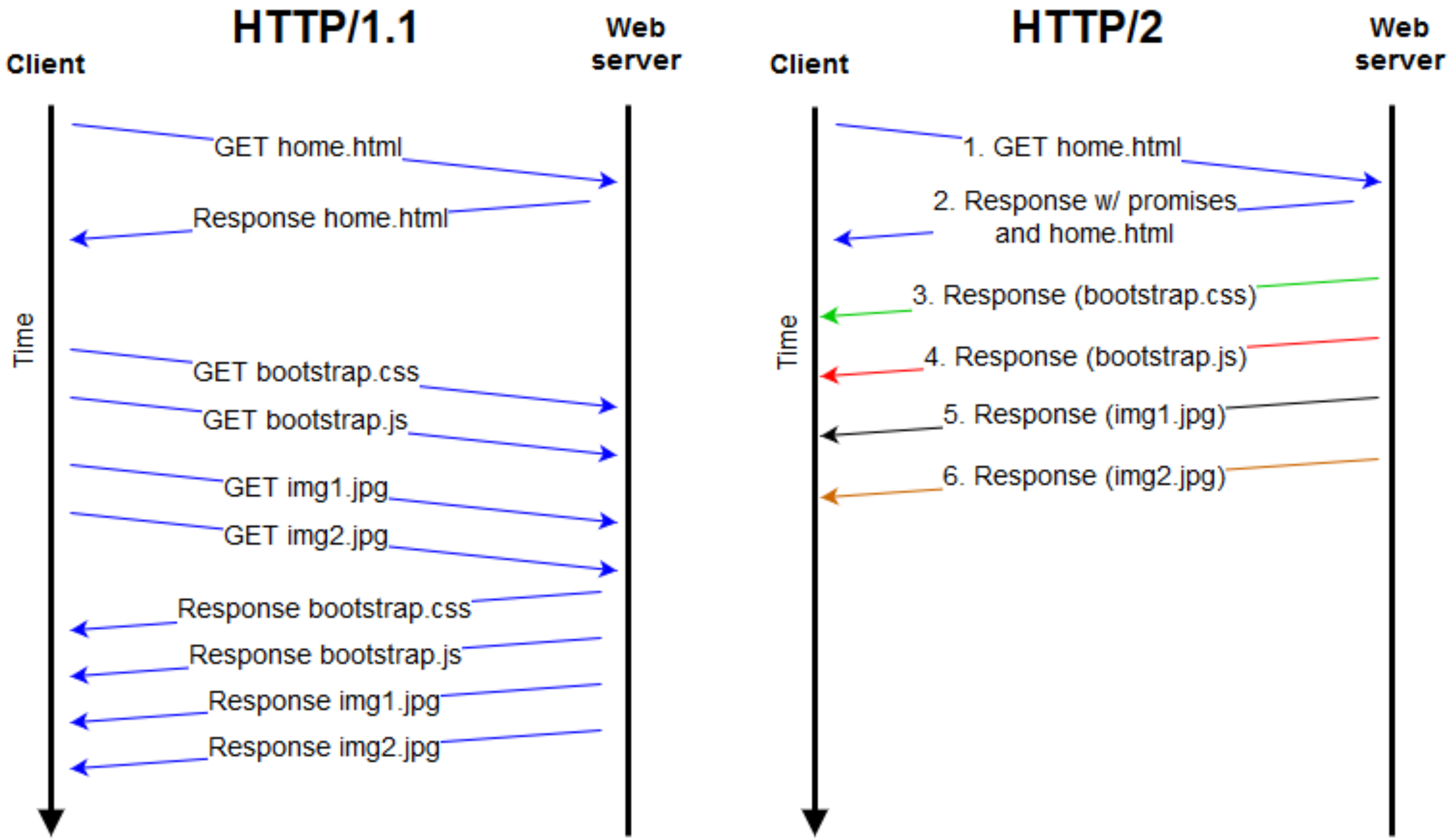
O_2, O_3, O_4 delivered quickly, O_1's finish-time slightly delayed

- What if 2^{nd} frame of O_1 gets lost: can O_2, O_3, and O_4 be delivered to the browser app before the loss is recovered?

HTTP/2 Performance Improvements

- Reduced HTTP header overhead
  - Binary encoding
  - Header compression
- Attempted to remove head-of-line blocking
  - Multiple streams, one for each http request/reply
  - Big messages are broken down to multiple frames
  - Frames from all streams can be interleaved
- Above approaches avoids HOL *at HTTP level*
  - Single TCP connection between client-server → packet losses still lead to head-of-line blocking

HTTP/2 server push



HTTP/2 to HTTP/3

FYI

Decreased delay in multi-object HTTP requests

HTTP/2 over single TCP connection means:

- Recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- No security over vanilla TCP connection
- HTTP/3: adds security , per object error and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer

FYI

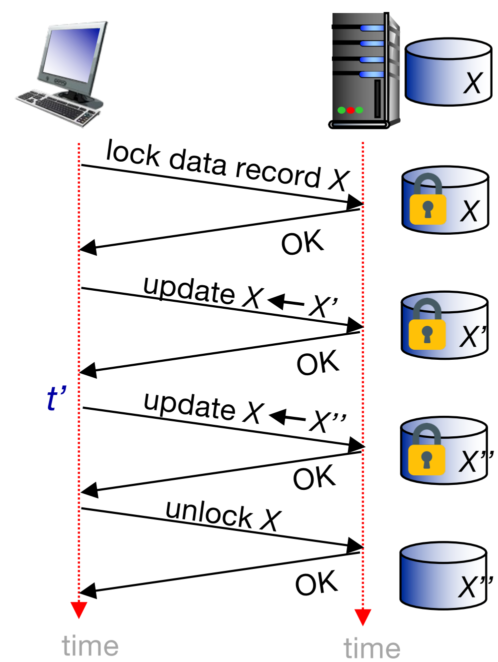
Tracking Web Clients via HTTP Cookies

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- No notion of multi-step exchanges of HTTP messages to complete a Web session
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed session

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t'* ?

Maintaining user/server state: cookies

Websites and client browser use *cookies* to maintain some state between sessions

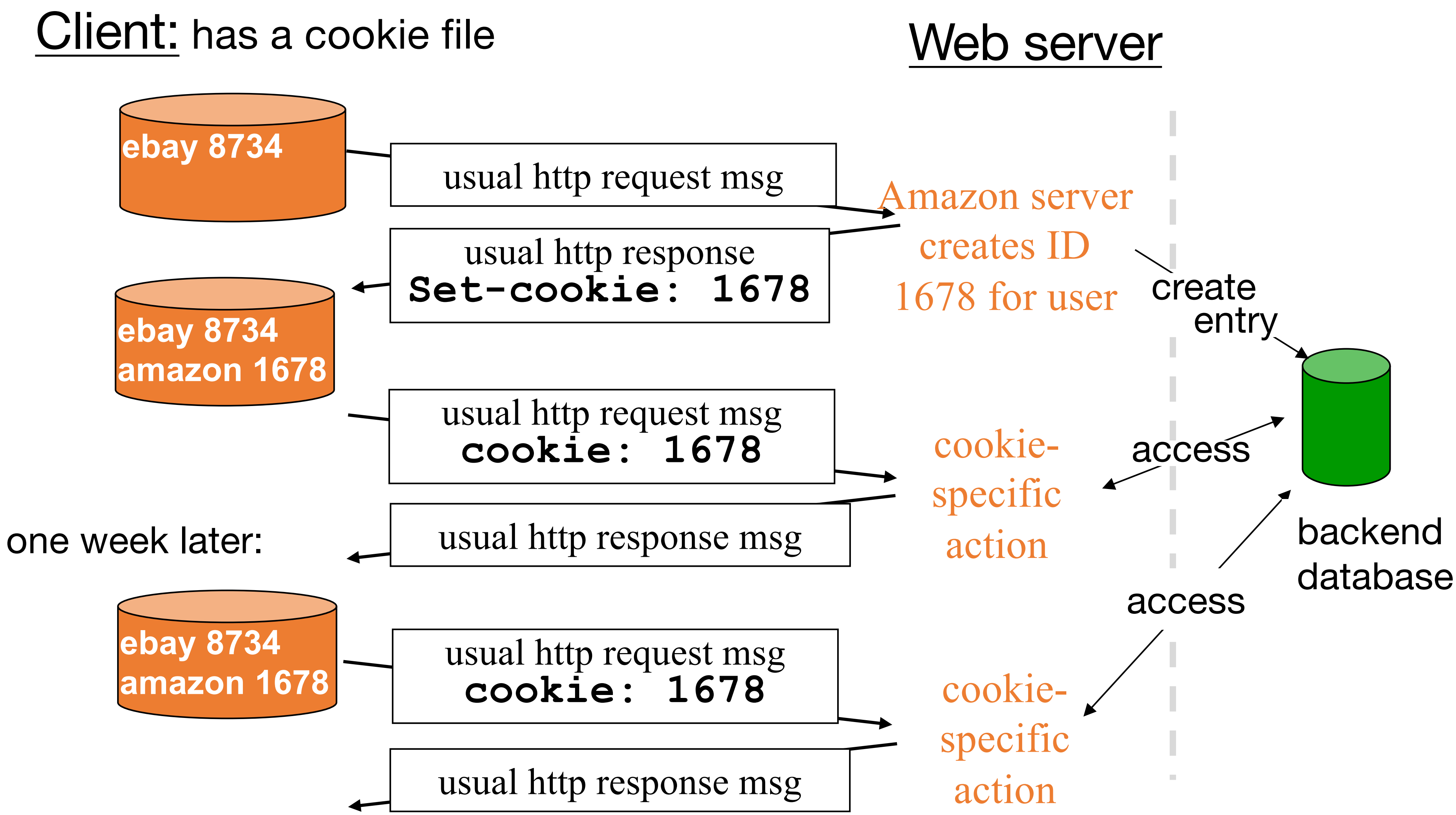
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) backend database at Web site

Example:

- Alice uses browser on laptop, visits specific Amazon for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Alice to this site will contain cookie ID value, allowing site to "identify" Amazon

User-server interaction: cookies



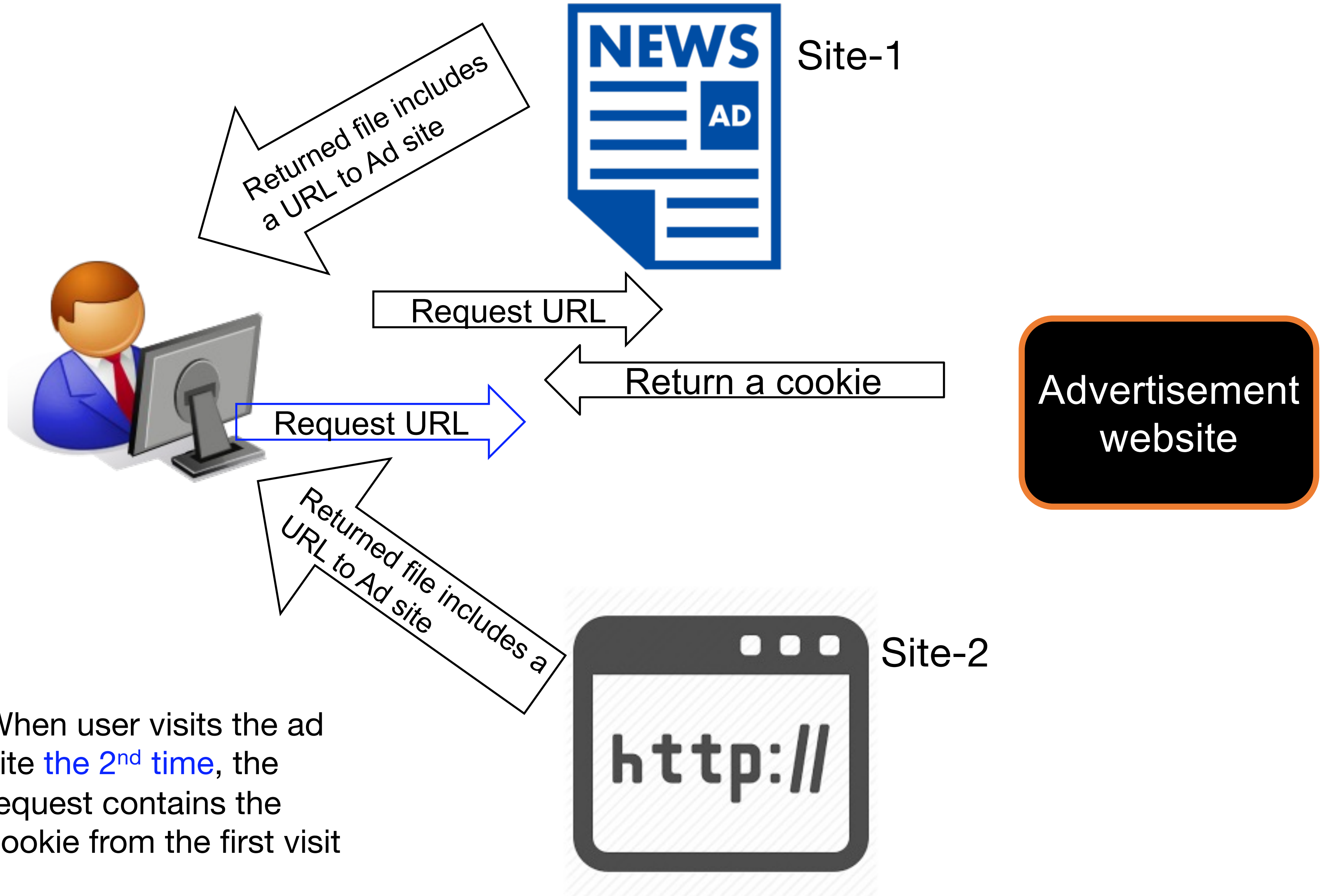
Cookies: usefulness vs privacy exposure

The use of cookies can

- Bring convenience to you
- Bring relevant recommendations

- Permit a website to learn your online behavior
- Advertising companies can obtain user info across multiple sites

Third Party Cookies



When user visits the ad site the 2nd time, the request contains the cookie from the first visit

Why is DNS needed?

- Application: host-to-host, process-to-process communication
  - Process identifier: **IP address and port number**
  - **HTTP server-1: http://173.194.204.99:80**
  - **HTTP server-2: http://176.32.103.205:80**
- But, how can we know and remember the destination IP address?
  - Google?
  - Amazon?
  - Facebook?

how to map between IP address and name, and vice versa ?

Domain Name System

- Why Internet needs DNS:
  - apps use name, IP needs address to deliver packets
  - name → IP address translation
    - One can also map IP address → name
- DNS: works in the *query-reply* pattern (like HTTP)
  - Your browser sends a DNS query with a name:
  - DNS server sends back a reply:

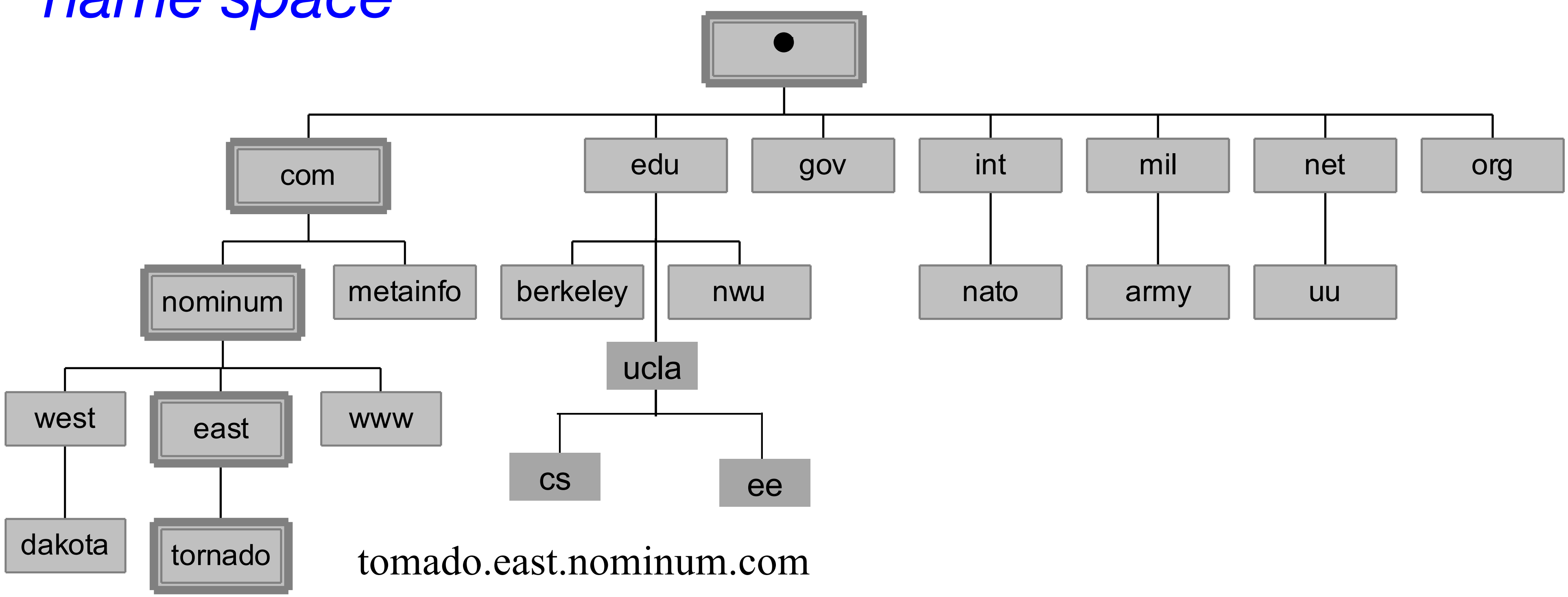
`web.cs.ucla.edu` → `131.179.128.29`

- DNS runs over UDP (unreliable transport) by default
  - DNS handles packet losses
  - DNS can also run over TCP

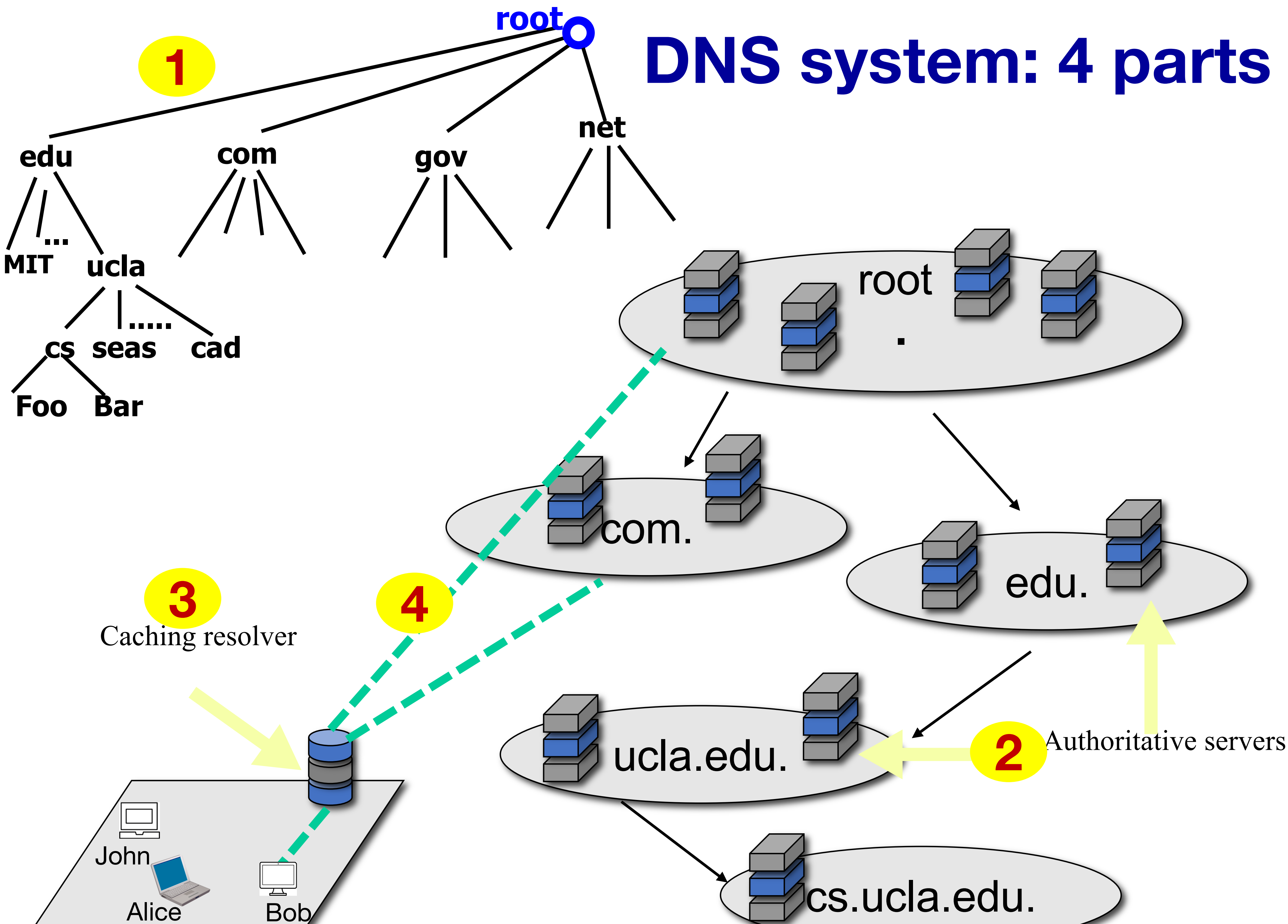
Q: Does DNS need to do anything different if running over TCP?

Domain Names

- A *domain name* is the sequence of labels from a node to the root, separated by dots (“.”s), read left to right
  - Domain names are limited to 255 characters in length
- A node’s domain name identifies its position in the *name space*



DNS system: 4 parts

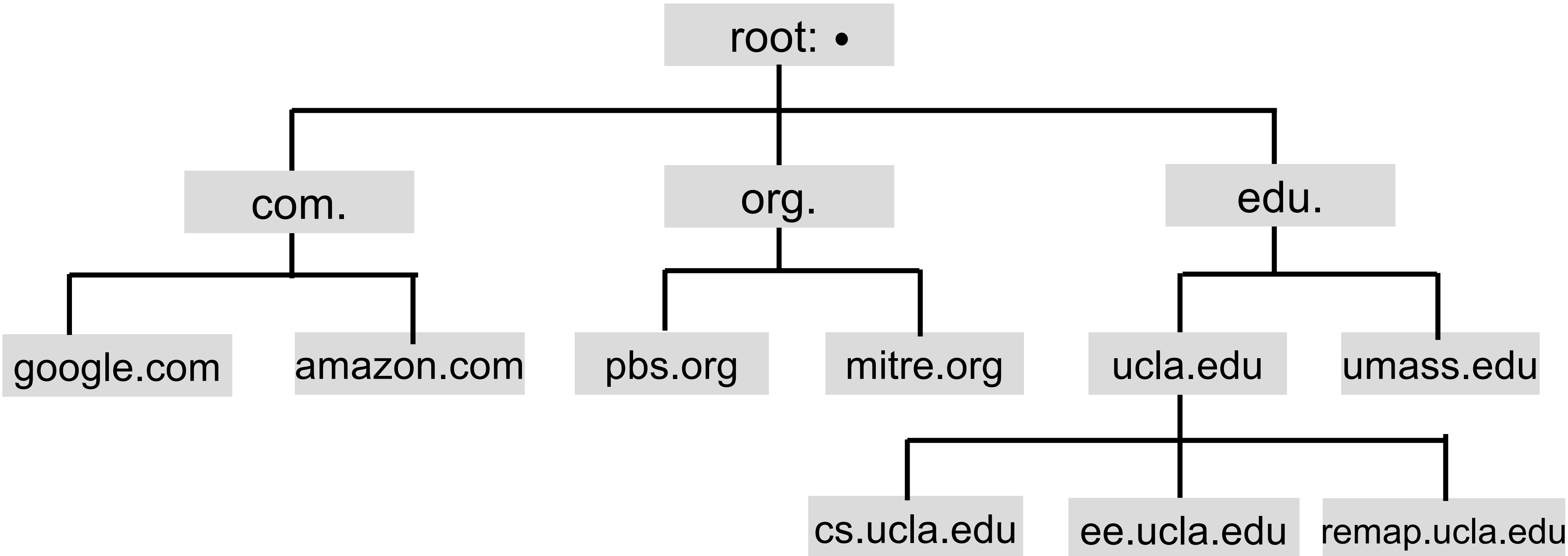


DNS: 4 Major Parts

important

1. A hierarchically structured *name space*
2. A *distributed (federated) database*, maintained by a hierarchy of *authoritative servers*
   - provided by individual domain owners
3. *Local DNS servers* (also called *caching resolvers*)
   - Each host runs a resolver routine (*stub resolver*), which talks to caching resolver provided by the host's Internet service provider
     - *Used to be the case, being changed now*
4. *DNS query protocol* used by local caching resolvers to query authoritative servers
   - also used for stub to caching resolver communication

DNS: defines a hierarchical name space



- starting from the root, growing downward, *variable depth*
- Each leaf node is a (DNS) name
- each ***non-leaf*** node in the tree is a ***domain***
  - Each domain belongs to an administrative authority
  - *delegated* domain can set up sub-domains, the tree depth limit: 127
- DNS name hierarchy: *independent from topological connectivity*

DNS Namespace Governance

- Internet Corporation for Assigned Names and Numbers (ICANN, https://www.icann.org/) oversees the management of
  - Assignment of Top Level Domains (TLDs)
  - Delegation of TLD managements
  - Operation of the root *name servers*
- TLD operators
  - Running TLD name servers
  - allocate 2nd level domain names
    - e.g.: `edu` allocates the name `ucla.edu` to UCLA
- 2nd level domain owners assign 3rd level names
  - `ucla.edu` allocates `cs.ucla.edu` to the CS dept

Top-Level Domains

- Generic TLDs (gTLD)
  - Old ones: .com, .org, .net, .mil, .gov, .edu, .arpa
  - New ones: .kim, .bar, .coffee, .dance, .lol, and more (1000+ and counting)
- Country code TLDs (ccTLD)
  - e.g.: .us, .kr, .ru, .cn
  - Internationalization ccTLDs (I18n ccTLD)
    - .한국 (South Korea), .рф (Russia), .中国 (China), …
- https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains

Second-level Domains

FYI

- Example 2nd-level domain names under gTLDs
  - ucla.edu, mit.edu
  - google.com, apple.com
  - ca.gov, mass.gov
- Examples under ccTLDs
  - .ac.uk, gov.uk
  - edu.cn, gov.cn
- DNS names of additional levels:
  - 3rd level: cs.ucla.edu
  - 4th level: sec.cs.ucla.edu
- No defined limit on the level of DNS names
  - Practical limit: the max length of a DNS name: 255 bytes

DNS Name Servers

- *Authoritative servers*: serving queries for a given domain
  - A domain has multiple authoritative name servers
    - Master – maintaining the master zone file
    - Slave – replicated copies of the master file
  - These servers should be placed in different networks
- *Caching resolvers* (“local DNS servers”)
  - query authoritative servers on users behalf
  - cache the data from DNS replies
- *Stub resolvers*: inside each host
  - configured with the IP address of (local) caching resolver(s)
  - send DNS queries to the caching resolver

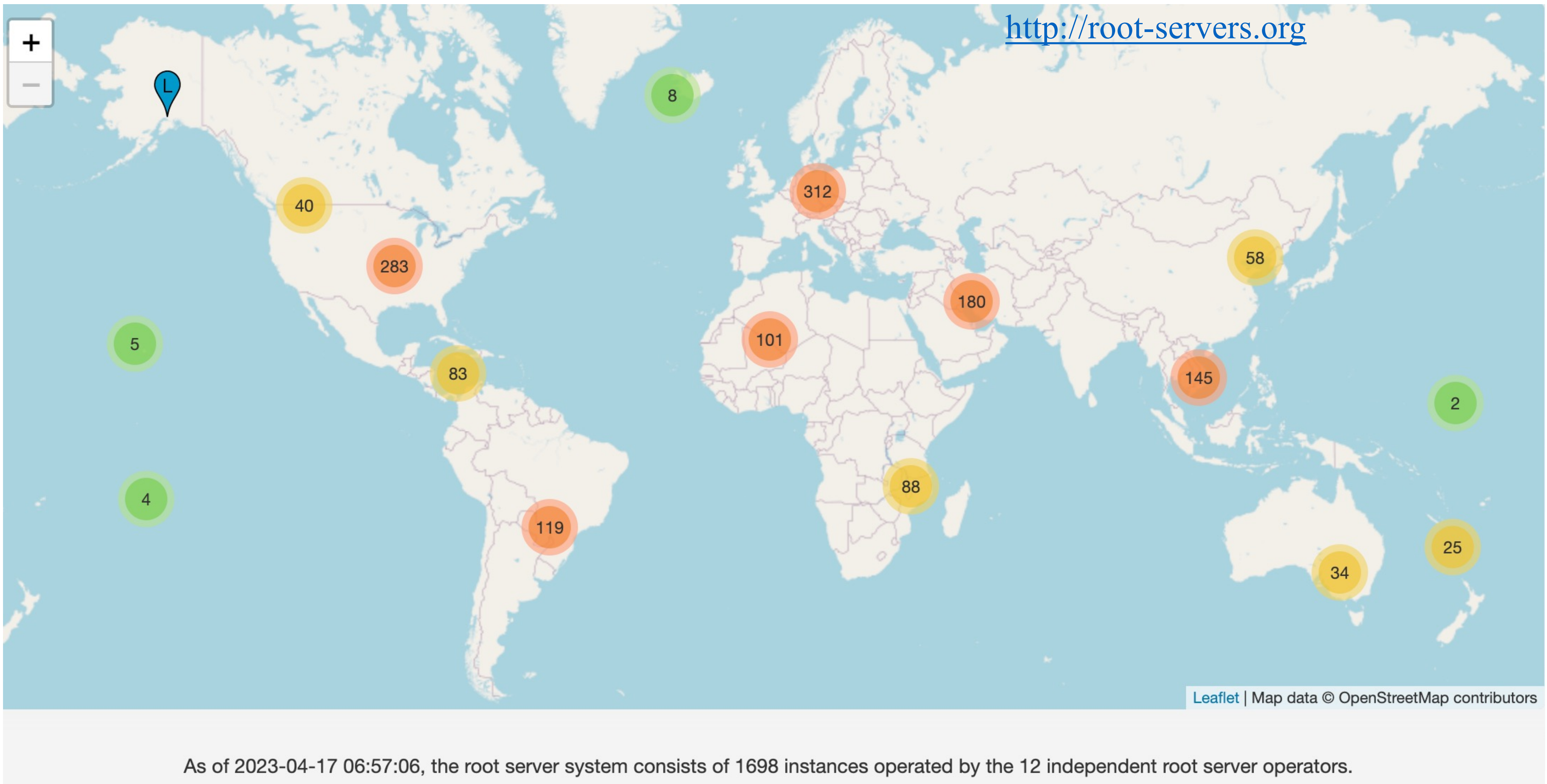
The Root Nameservers

https://root-servers.org/

- The root domain file: contains the names and IP addresses of the authoritative DNS servers for all the top-level domains (TLDs)
- This root domain file is published on 13 root DNS servers, named “A” through “M”, provided by *volunteer efforts* of diverse organizations

DNS Root Name Servers

- 13 root name servers operated by various parties on a coordinated, volunteering basis, all have multiple instances via anycast



List of Root Servers

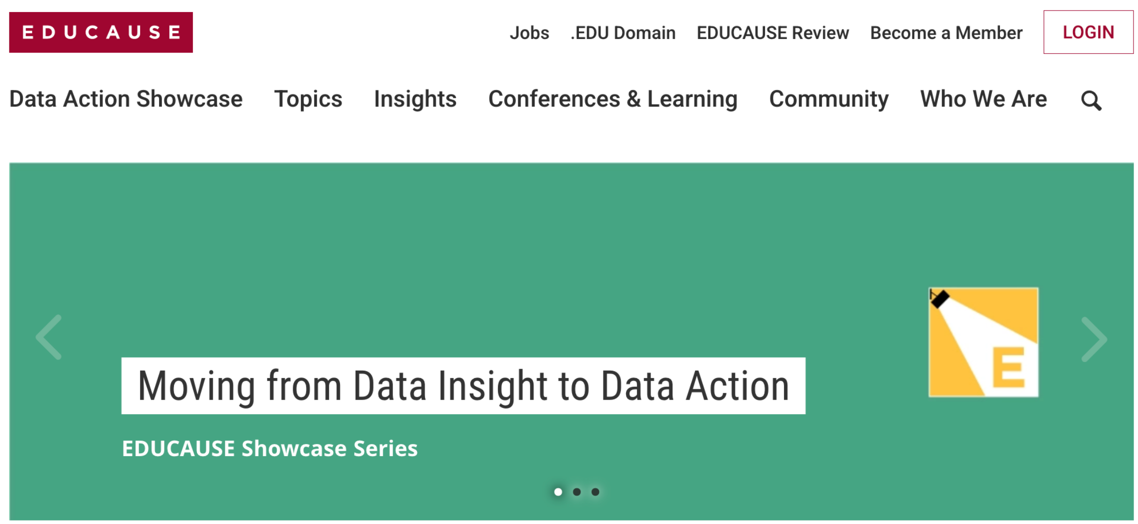
HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

IPv4 address IPv6 address

TLD nameservers

- each country's government provides ccTLD authoritative name servers
- gTLD name servers: ICANN delegates the management of each gTLDs to a specific organization
  - .edu is delegated to EDUCAUSE, which runs *authoritative servers* for .edu, allocates names to US higher education institutions

https://www.educause.edu/



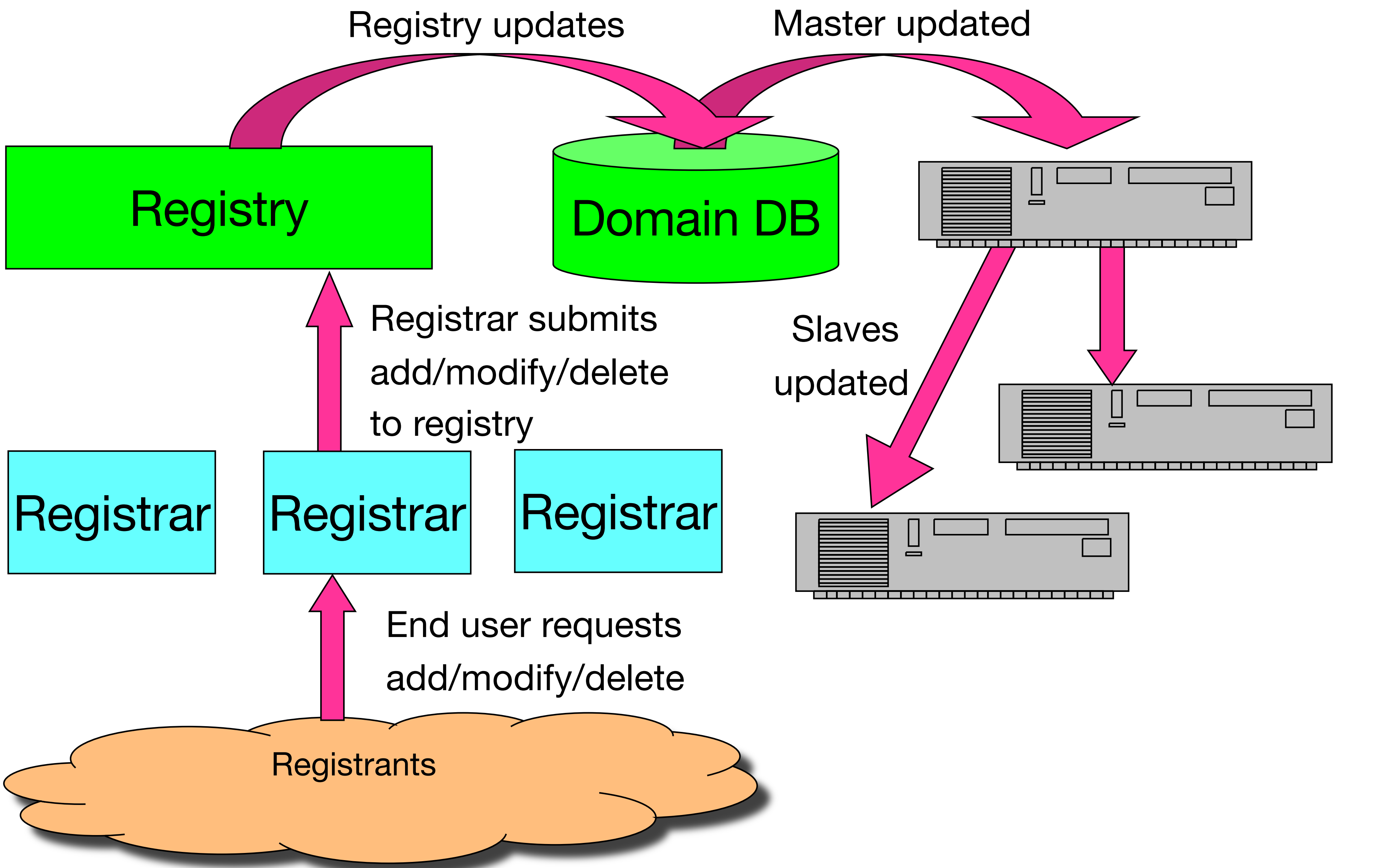
  - ucla.edu is delegate to UCLA, which runs authoritative servers to serve queries for names in ucla.edu domain

FYI

Another example: Verisign

- ICANN delegates the management of .com to Verisign
- Verisign operates *authoritative name servers* for .com domain
- Verisign contracts registrars to sell domain names to public
  - Example registrars
    - GoDaddy (US)
    - 22net (in China)
    - CoolOcean (India)
- There exist a *very* large number of registrars

Registries, registrars, registrants



Registries, registrars, registrants

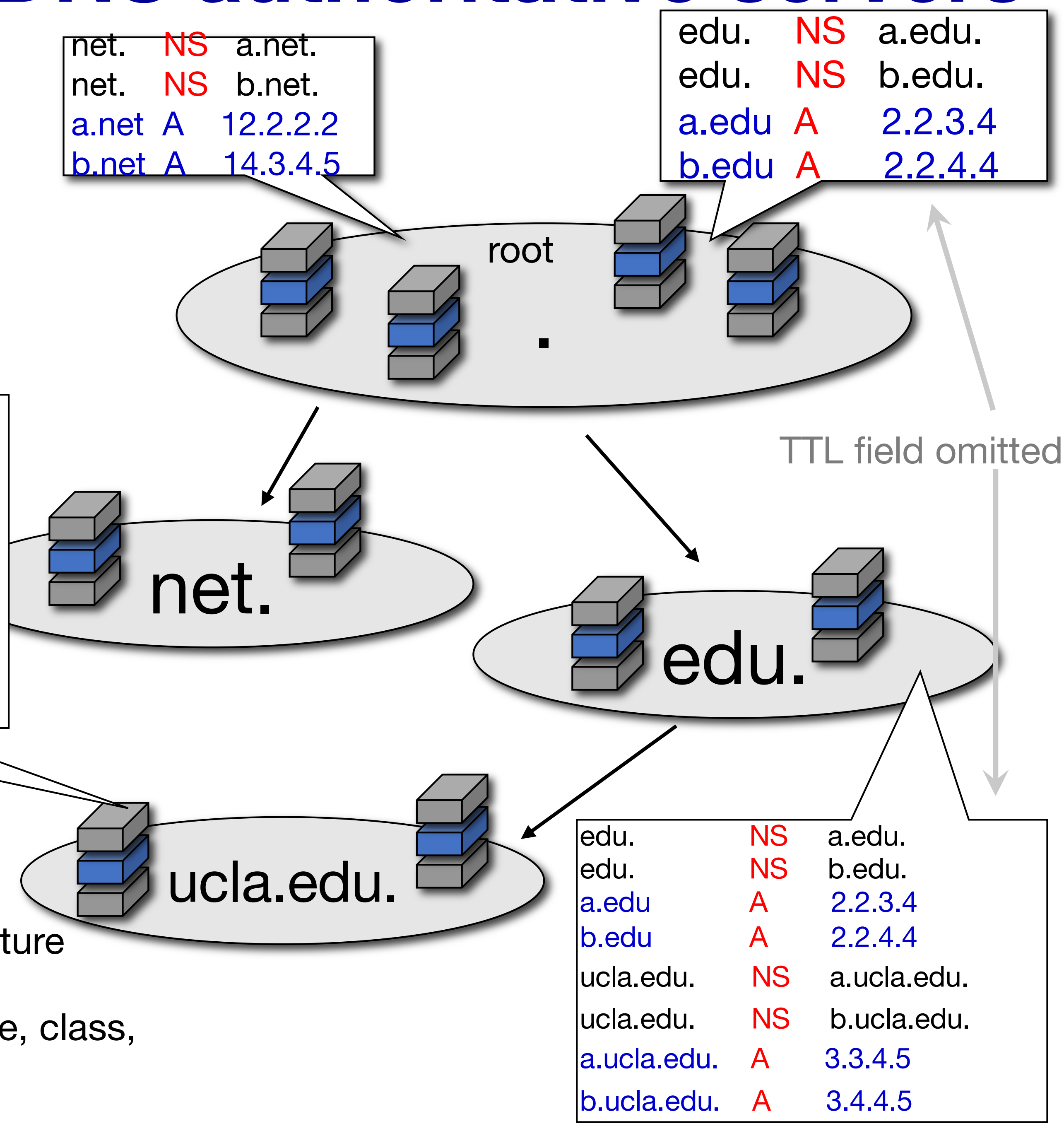
FYI

- Registry
  - An organization that manages a DNS namespace
    - Allocate names, or work with a registrar for name allocations
    - Run name servers
- Registrar
  - An organization that sells domain names to the public
  - Submits change requests to the registry on behalf of the registrant
- Registrant:
  - Person or company who registers a domain name
  - A registrant can manage its domain name's settings through its own registrar.

Glue together DNS authoritative servers

Each NS RR of zone Z and the corresponding *glue RR* is stored in both Z's own and its parent's zone files

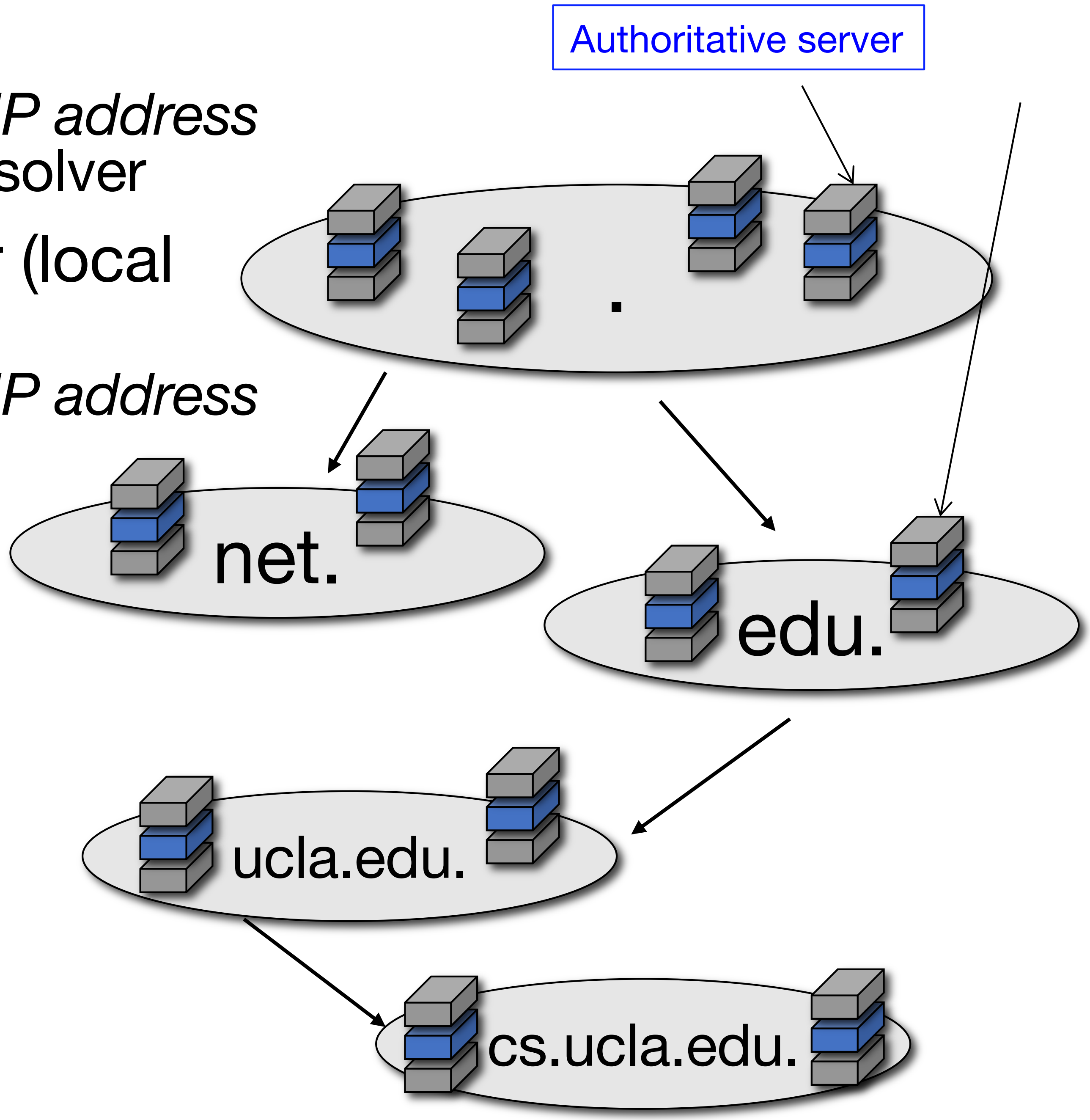
NAME	TYPE	TTL	VALUE
ucla.edu	NS	824	a.ucla.edu
ucla.edu	NS	824	b.ucla.edu
a.ucla.edu	A	600	3.3.4.5
b.ucla.edu	A	900	3.4.4.5
www.ucla.edu	A	1700	3.2.2.2
mail.ucla.edu	A	3100	3.3.3.3
….			

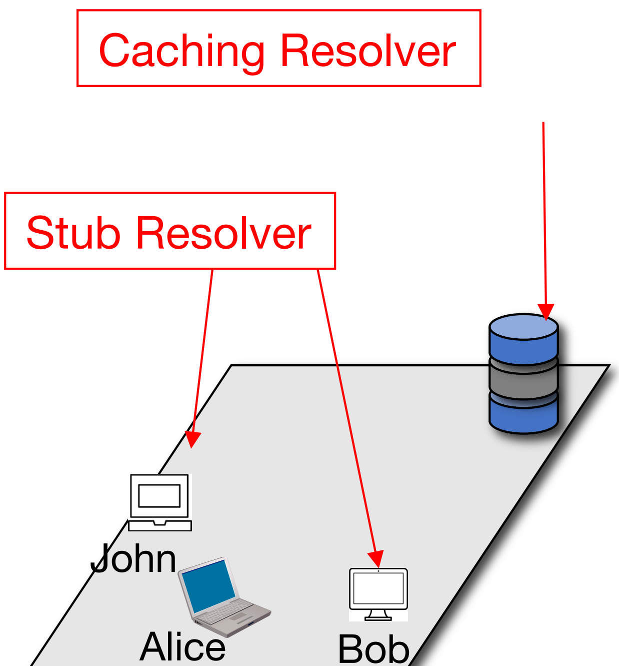


- All DNS data stored in a data structure called "*resource record*" (RR)
- An RR contains 5 fields: name, type, class, TTL, value

Bootstrapping DNS lookup

- Stub resolvers
  - must know the *IP address* for a caching resolver
- Caching resolver (local DNS server)
  - must know the *IP address* for root server



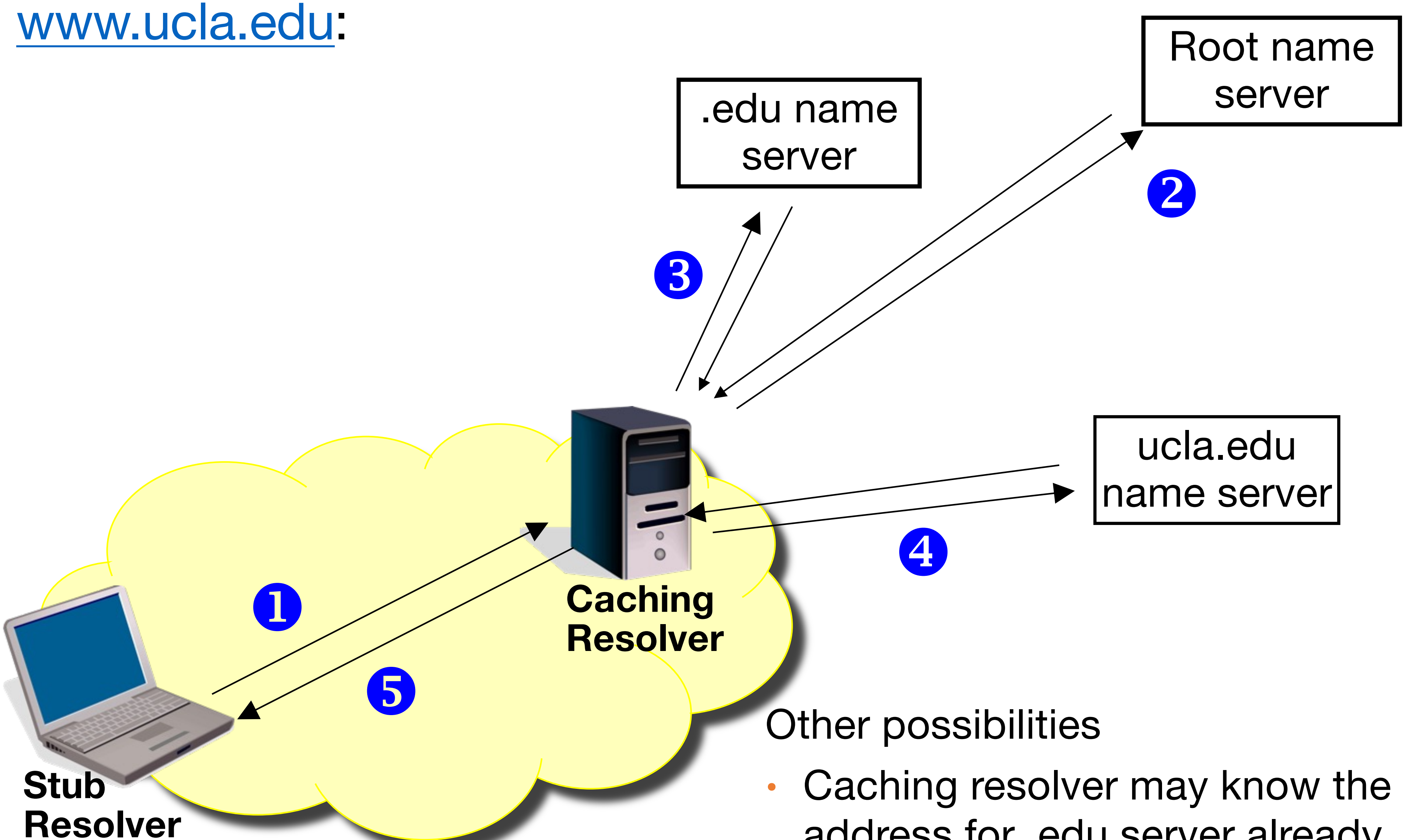


DNS Resolution

- Whenever an app needs to communicates: first call DNS to translate the name to IP address, then open socket with the destination address
  - System call `getaddrinfo(), gethostbyname()`
- Stub resolver
  - configured with the IP address of the caching resolver(s)
  - send DNS queries to local caching resolvers
- Caching resolver (local DNS server)
  - Has the *IP address* of root servers, hard-coded in
  - Query authoritative servers, cache the data from replies

Example of DNS Lookup

Your browser needs IP address for www.ucla.edu:

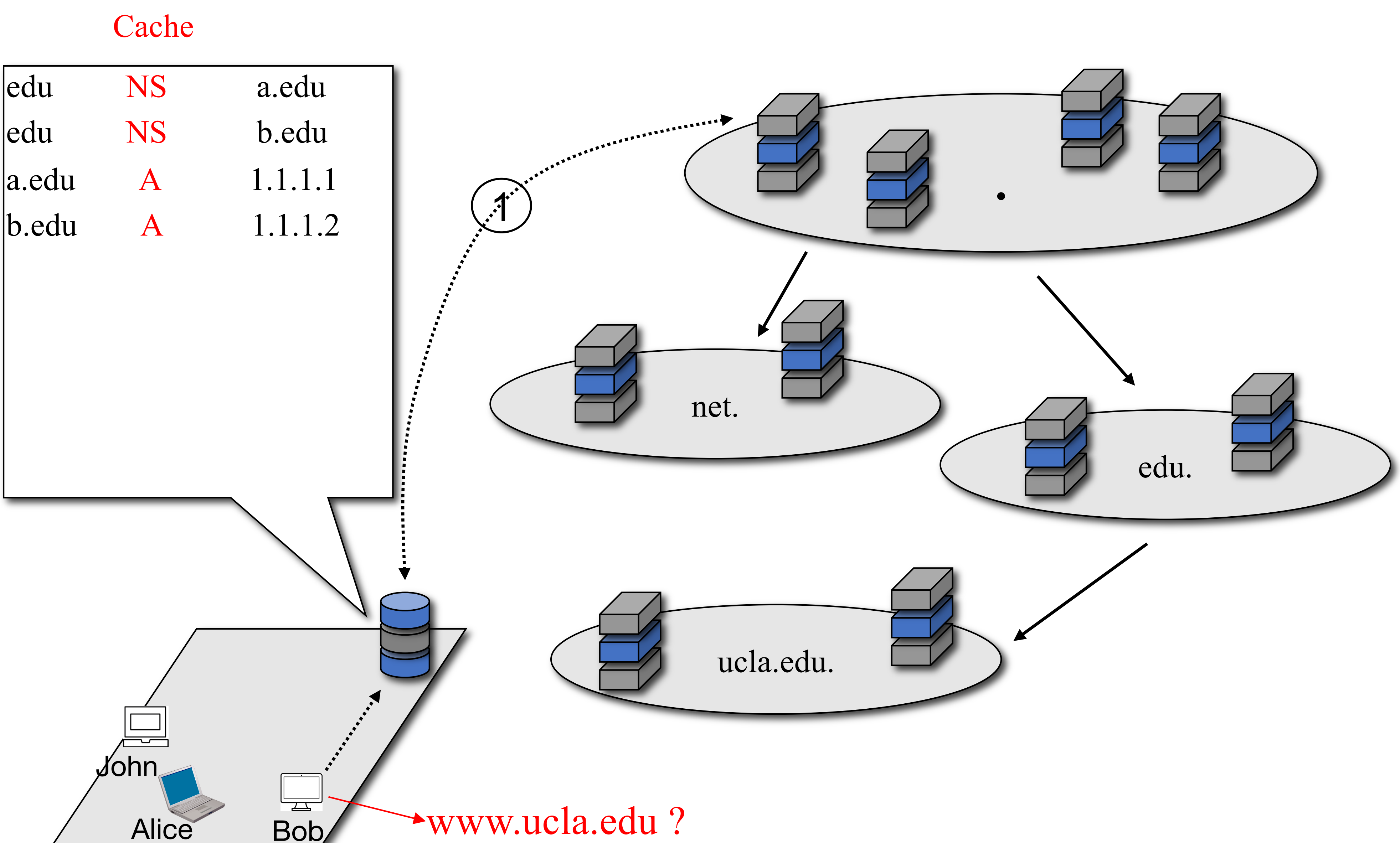


Other possibilities

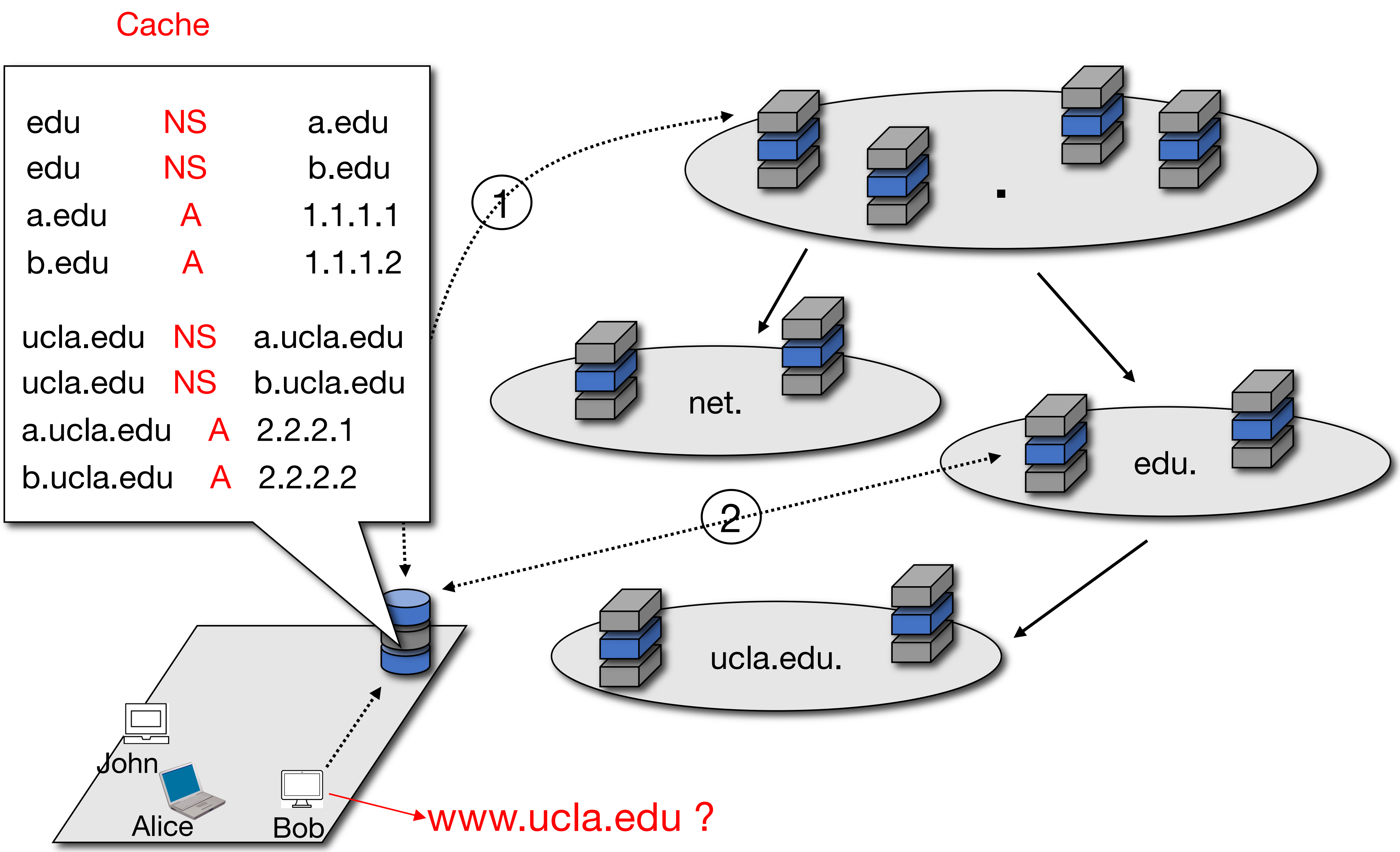
- Caching resolver may know the IP address for .edu server already

Steps of Actions in Resolving a Name

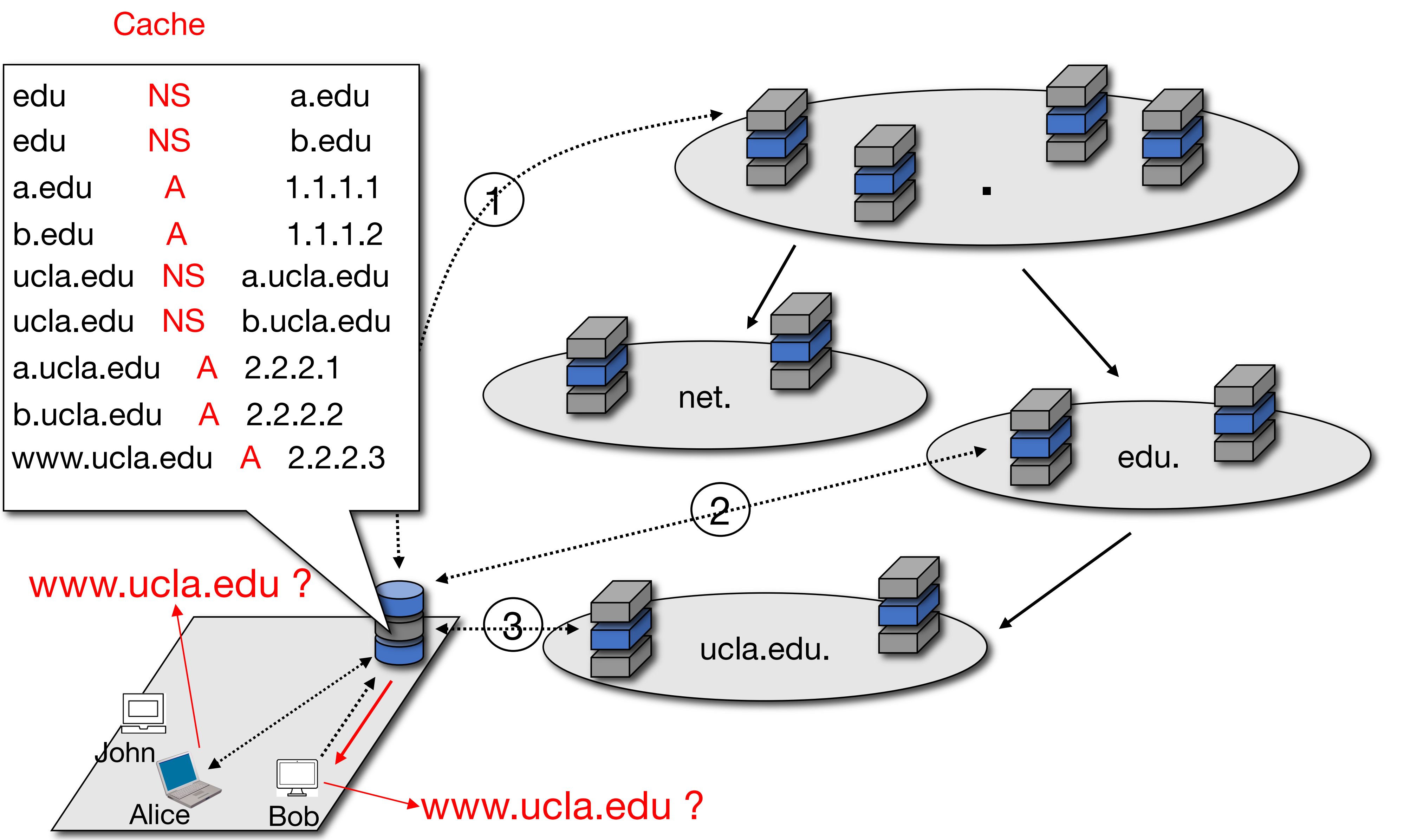
important



Steps of Actions in Resolving a Name



Steps of Actions in Resolving a Name



Summary: How a DNS name gets resolved?

1. A user host sends a query for `www.ucla.edu` (asking for its IP address) to a local DNS caching resolver
   - provided by your ISP
     - In recent years: provided by Google (8.8.8.8), CloudFlare (1.1.1.1), etc
2. The caching resolver either finds a *relevant* answer in its cache,
   - any of the following are relevant to `www.ucla.edu`
     - An exact match: `www.ucla.edu`'s IP address
     - ucla.edu DNS server IP address: go to step-5
     - .edu DNS server IP address: go to step-4

   otherwise sends the query to one of the root servers
3. The root server replies with pointers to `.edu` servers
4. The caching resolver queries `.edu` DNS server, which replies with pointers to `ucla.edu` DNS servers
5. The caching resolver queries `ucla.edu` DNS server to get the IP address for `www.ucla.edu`, and sends the answer back to user host

Exploring DNS

- dig
  - Should be available by default on macOS
  - Part of "bind" package on Linux (and if brave enough, on Windows)

https://www.digwebinterface.com/

```
tianyuan% dig . NS

; <<>> DiG 9.10.6 <<>> . NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38900
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;. IN NS

;; ANSWER SECTION:
. 16232 IN NS e.root-servers.net.
. 16232 IN NS h.root-servers.net.
. 16232 IN NS l.root-servers.net.
. 16232 IN NS i.root-servers.net.
. 16232 IN NS a.root-servers.net.
. 16232 IN NS d.root-servers.net.
. 16232 IN NS c.root-servers.net.
. 16232 IN NS b.root-servers.net.
. 16232 IN NS j.root-servers.net.
. 16232 IN NS k.root-servers.net.
. 16232 IN NS g.root-servers.net.
. 16232 IN NS m.root-servers.net.
. 16232 IN NS f.root-servers.net.
```

```
tianyuan% dig a.root-servers.net (A)

; <<>> DiG 9.10.6 <<>> a.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR,
id: 30471
;; flags: qr rd ra; QUERY: 1, ANSWER: 1,
AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;a.root-servers.net. IN A

;; ANSWER SECTION:
a.root-servers.net. 604800 IN A 198.41.0.4
```

```
tianyuan% dig a.root-servers.net aaaa
......
;; QUESTION SECTION:
;a.root-servers.net. IN AAAA

;; ANSWER SECTION:
a.root-servers.net. 604800 IN AAAA
2001:503:ba3e::2:30
```

- 2nd level domains:
  - UCLA runs its own DNS servers
- 3rd level domains: CS dept runs its own DNS servers

```
tianyuan% dig ucla.edu ns
.......
;; QUESTION SECTION:
;ucla.edu. IN NS

;; ANSWER SECTION:
ucla.edu. 917 IN NS ns2.dns.ucla.edu.
ucla.edu. 917 IN NS ns3.dns.ucla.edu.
ucla.edu. 917 IN NS ns4.dns.ucla.edu.
ucla.edu. 917 IN NS ns1.dns.ucla.edu.

;; ADDITIONAL SECTION:
ns1.dns.ucla.edu. 10093 IN A 192.35.225.7
ns2.dns.ucla.edu. 17620 IN A 54.2
ns2.dns.ucla.edu. 19766 IN AAAA 2
ns3.dns.ucla.edu. 11775 IN A 54.2
ns4.dns.ucla.edu. 21258 IN A 3.10
```

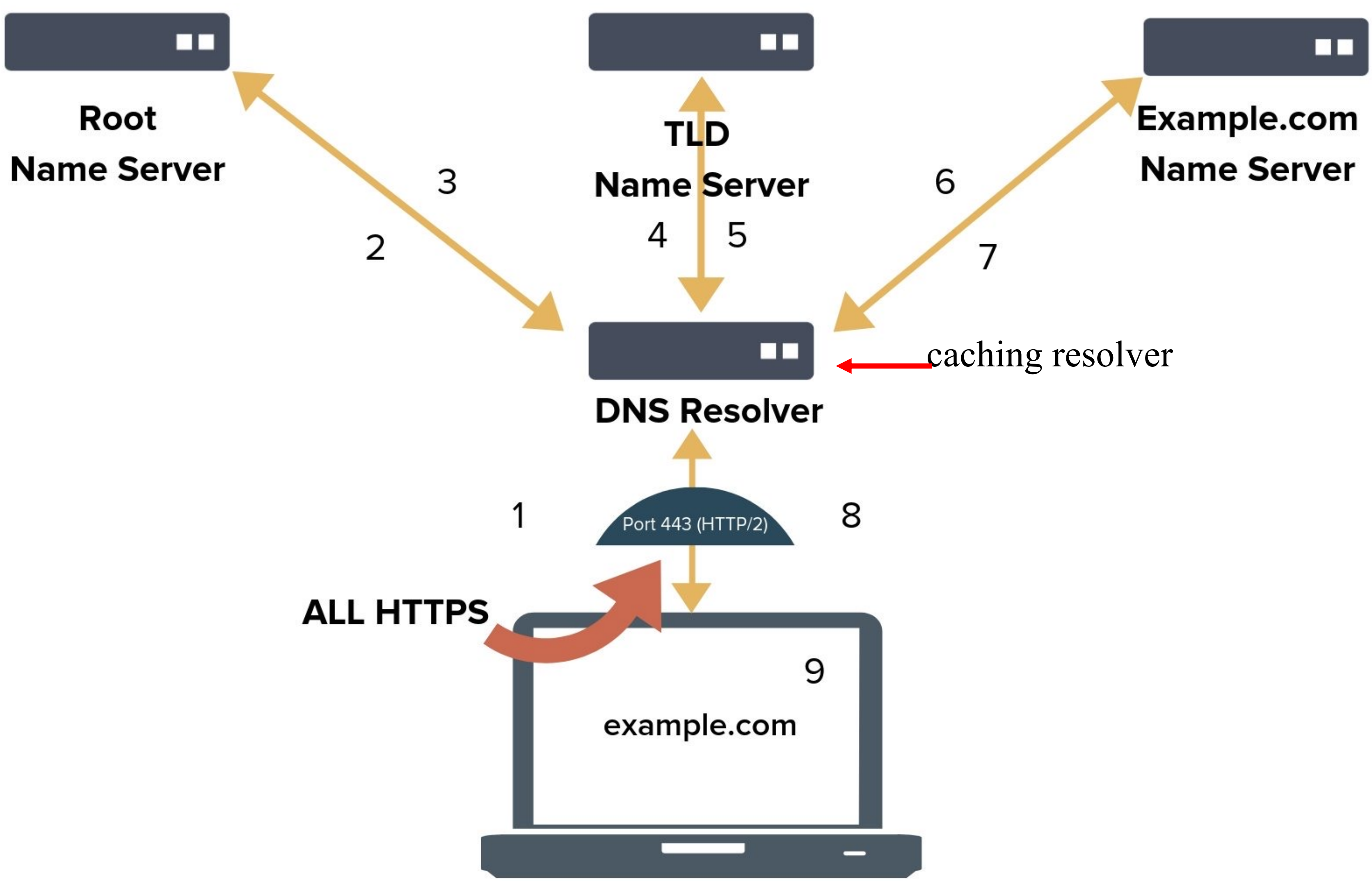
```
tianyuan% dig cs.ucla.edu ns

;; QUESTION SECTION:
;cs.ucla.edu.                IN      NS

;; ANSWER SECTION:
cs.ucla.edu.       14400     IN      NS      NS0.cs.ucla.edu.
cs.ucla.edu.       14400     IN      NS      NS3.cs.ucla.edu.
cs.ucla.edu.       14400     IN      NS      NS2.DNS.ucla.edu.
cs.ucla.edu.       14400     IN      NS      NS2.cs.ucla.edu.
cs.ucla.edu.       14400     IN      NS      NS3.DNS.ucla.edu.
cs.ucla.edu.       14400     IN      NS      NS1.cs.ucla.edu.
```

Latest change: DNS over HTTPS (DoH)

FYI



FYI

Inserting records into DNS

- Example: assume creating a new "Foo University"
- Register name foo.edu at EDU registrar
  - Need to provide registrar with names and IP addresses of your authoritative name servers (primary and secondary)
  - Registrar inserts two RRs into the edu TLD server:

```
(foo.edu, a.foo.edu, NS)
(a.foo.edu, 1.1.1.1, A)
```

- Put in authoritative server Type A record for www.foo.edu, and Type MX record for foo.edu

How do people get the IP address of Web site `www.foo.edu`?

FYI

Example Configuration