Application Layer

Instructor: C. Pu (Ph.D., Assistant Professor)

Lecture 07

puc@marshall.edu

User-Server State: Cookies

- HTTP server is ***stateless***
  - simplifies server design
  - prompt to dev. high-performance server handling simultaneous TCP conn.
- However, it is desirable for server to ***identify users***
  - either wish to restrict user access
  - or want to serve content as a function of the user identity
- HTTP cookies
  - allow sites to keep track of users
  - major commercial Web sites use cookies

User-Server State: Cookies

- Four components with cookie:

  1) a cookie header line in HTTP ***response*** message

  2) a cookie header line in HTTP ***request*** message

  3) a ***cookie file*** kept on user's end system, managed by user's browser

  4) a back-end ***database*** at Web site

User-Server State: Cookies

- Example:
  - Susan always accesses Internet always from PC
  - in the past she has visited the eBay.com
  - visit Amazon.com for first time
  - when initial HTTP request arrives at server site, server site creates:
    - **unique ID** (or **identification number)**
    - **entry** in back-end database for **ID**
      - entry is indexed by the ID
  - Amazon.com server responds HTTP response to Susan's browser
    - including a *Set-cookie: header*
      - contains the ID

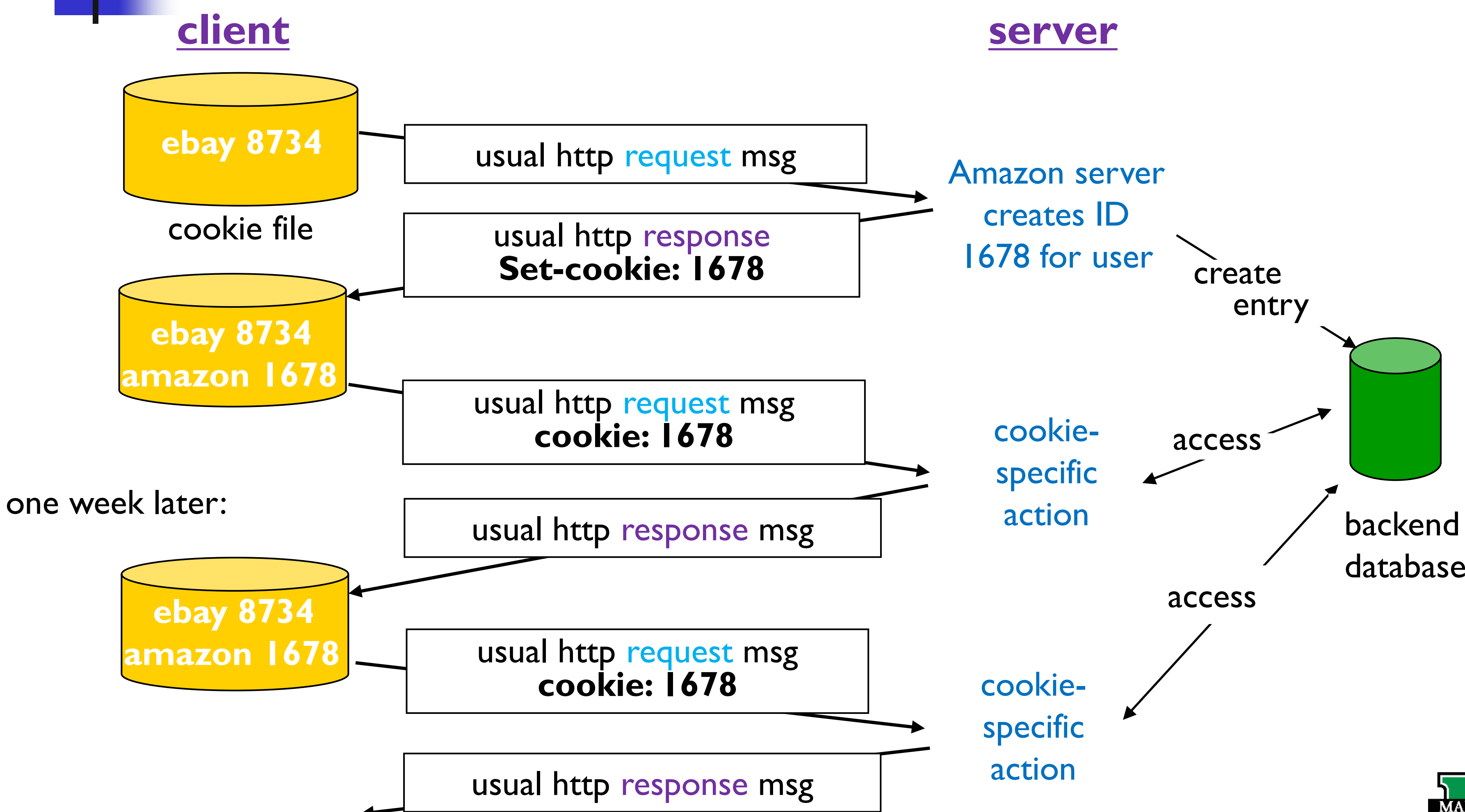
```
Set-cookie: 1678
```

User-Server State: Cookies

- Example:
  - when Susan's browser receives the HTTP response
    - appends a line to the cookie file
      - hostname for server + the ID in the *Set-cookie: header*
    - cookie file already has an entry for eBay
  - as Susan continues to browser Amazon.com
    - each time she requests a Web page
      - consults cookie file
      - extracts the ID for the site
      - put a cookie header line with the ID in the HTTP request
    - each HTTP request includes the header line

```
Cookie: 1678
```

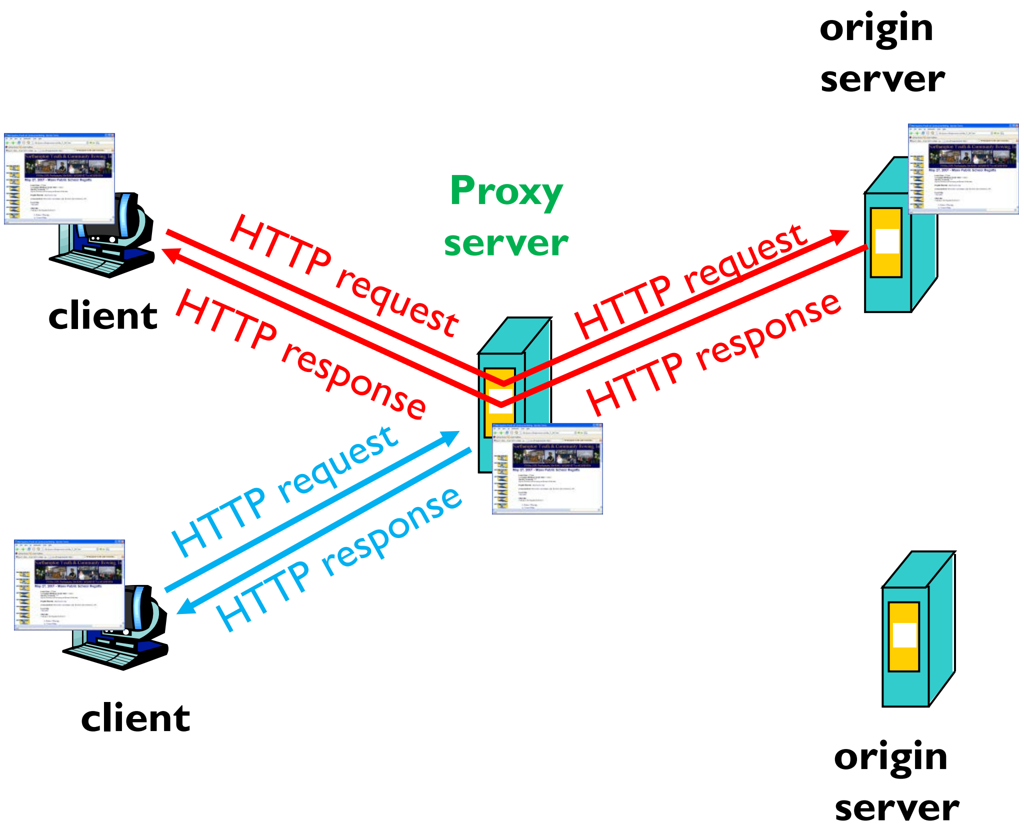
Cookies: Keeping "state" (cont.)



Web Caches (Proxy Server)

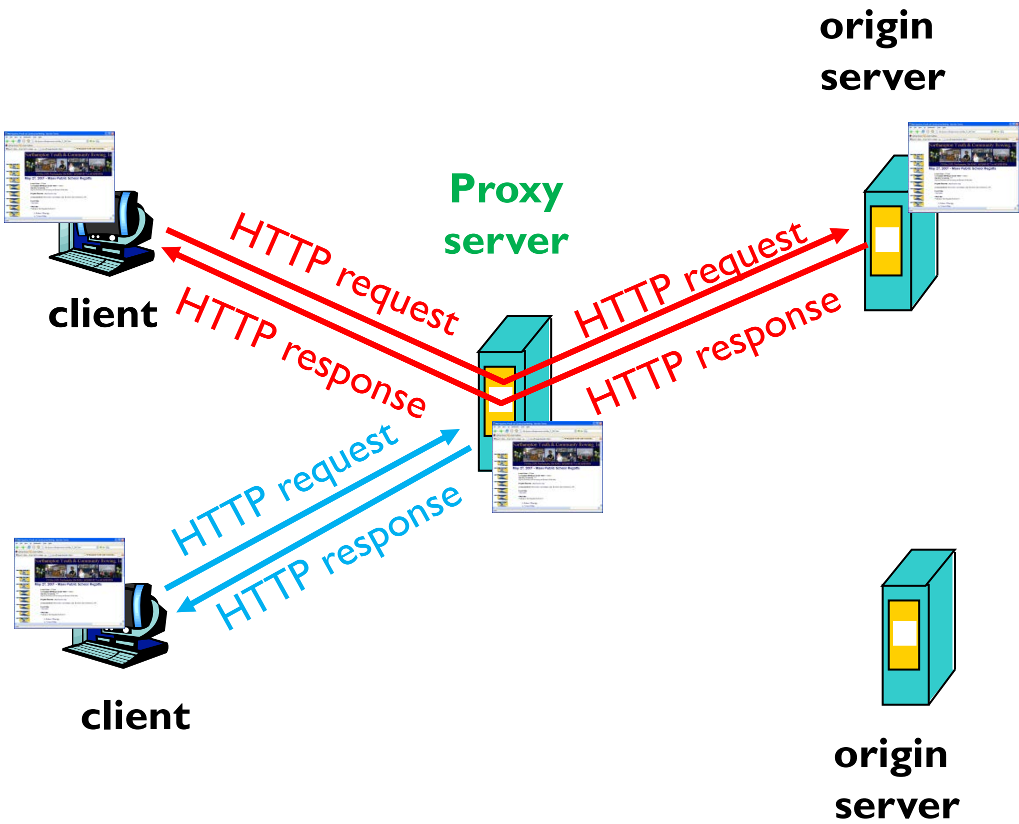
Goal: satisfy client request without involving origin server

- **Web cache**: a network entity satisfies HTTP requests on the behalf of an origin Web server
  - has its own disk storage
  - keeps copies of recently requested objects in the storage
- user can configure browser: Web accesses via Web cache
- browser sends all HTTP requests to Web cache
  - if, object in Web cache: Web cache returns object
  - else, Web cache requests object from **origin server**, then store a copy and returns object to client



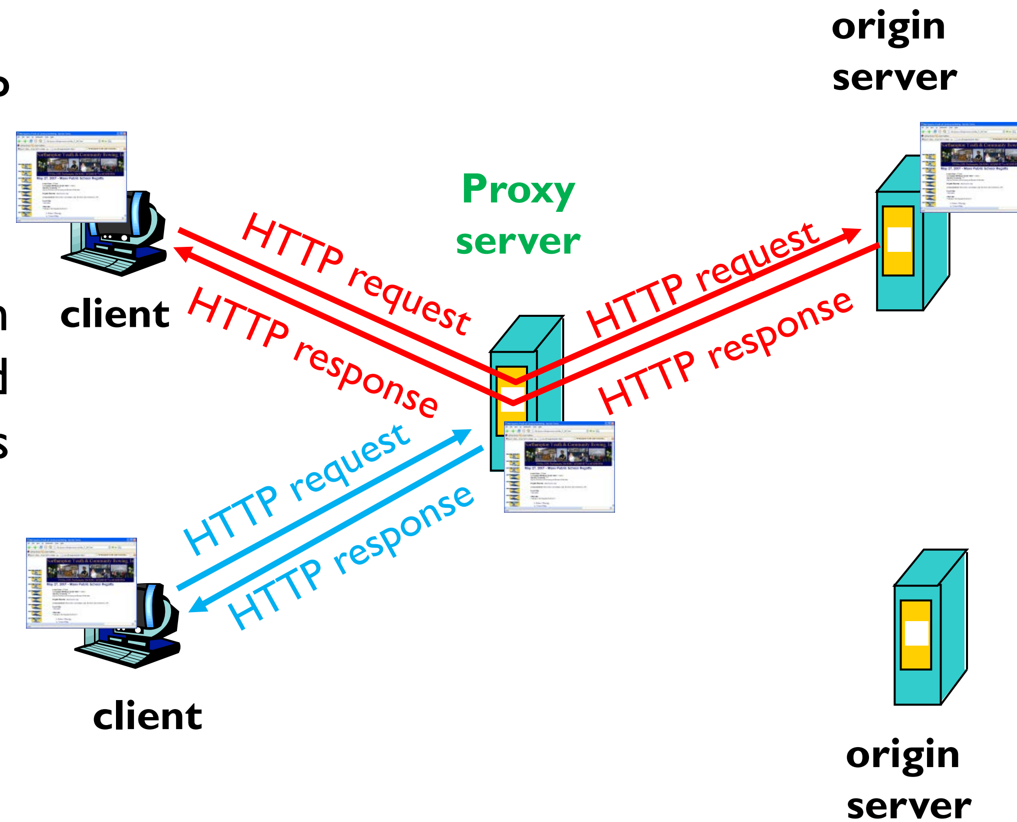
More About Web Caching

- Web cache acts as both *client* and *server* at the same time
  - *server*: when receiving request from client and sending response to a browser
  - *client*: when sending request to and receiving response from origin server

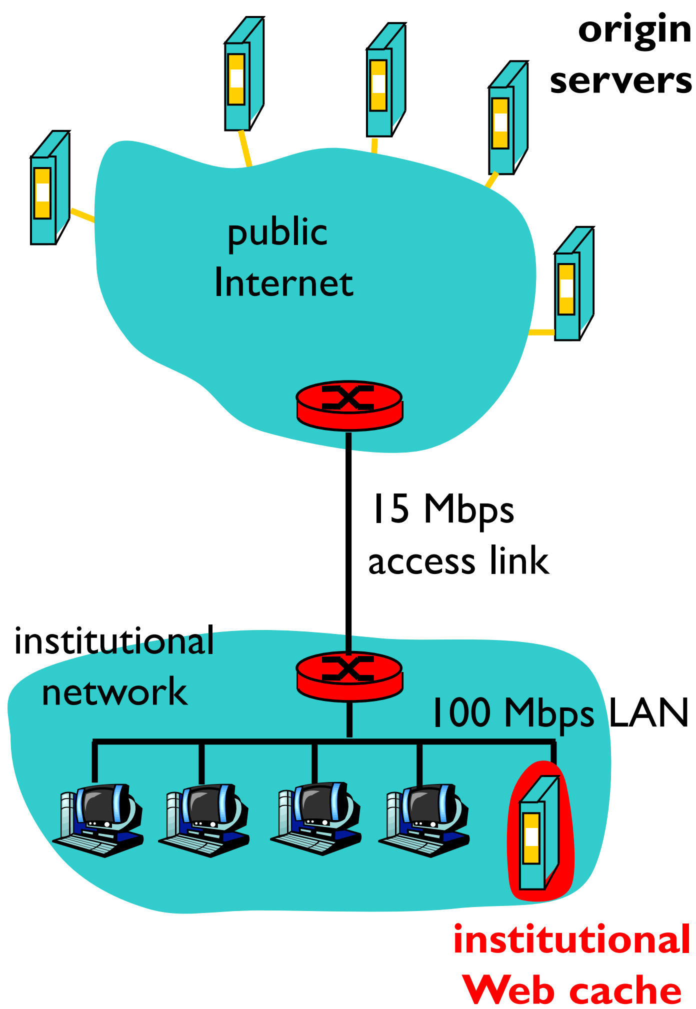


More About Web Caching

- typically cache is installed by ISP (university, company, residential ISP)
- E.g.,
  - Marshall Univ. install a cache on its campus network and configure all of the campus browsers to point to the cache



Caching Example (cont.)



Conditional GET

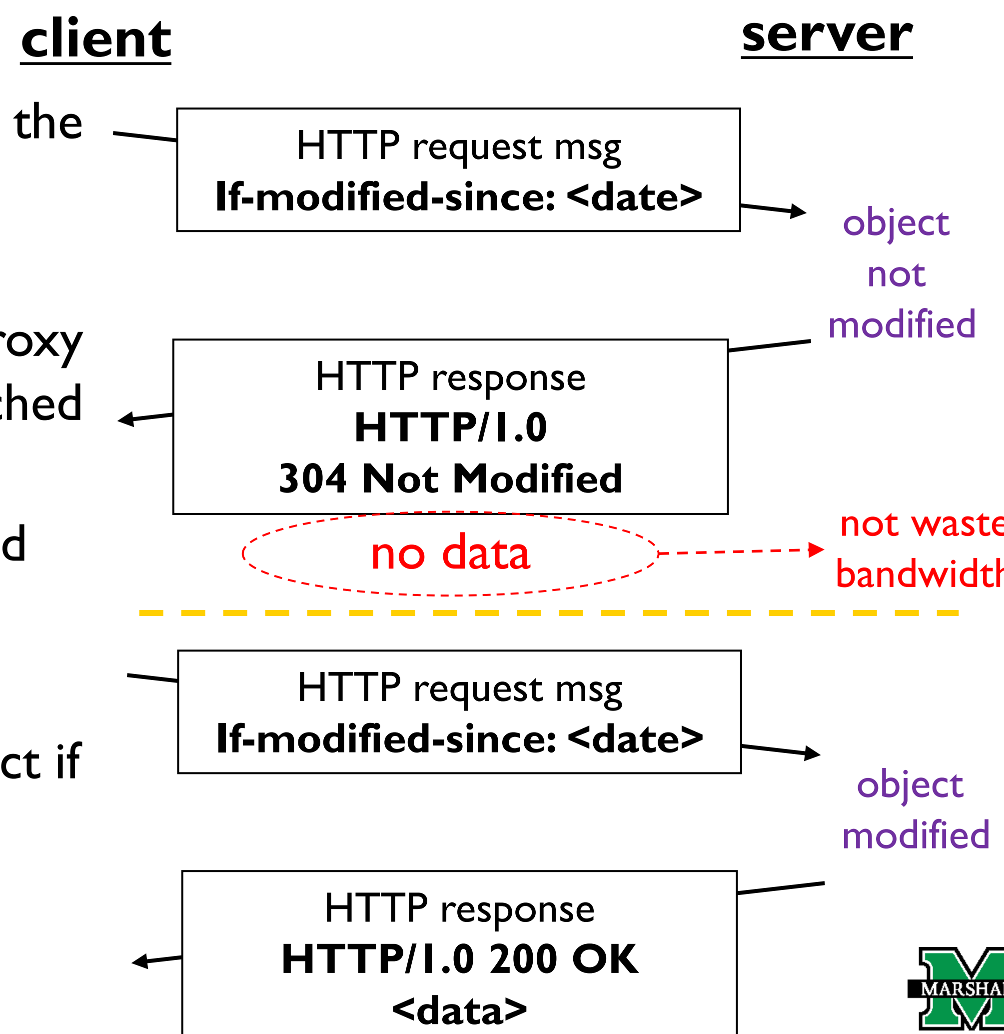
- Caching can reduce user-perceived response times, but introduces a new problem
  - the copy of object residing in the cache may be ***stale***
    - the object in the Web server may have been *modified* since the copy was cached at the client
- HTTP has a mechanism to verify that the objects are up to date
  - **conditional GET**
- an HTTP request message is called **conditional GET** message if
  - (i) the request message uses the **GET** method and
  - (ii) the request message includes an **If-Modified-Since:** *header line*

Conditional GET

- The copy of an object residing in the proxy cache may be stale
  - Need to verify
- Goal: don’t send object if proxy cache has **up-to-date** cached version
- Proxy cache: specify date of cached copy in HTTP request

  If-modified-since: <date>
- server: response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



DNS: Domain Name System

- people has many identifiers
  - SSN, name, passport #
- so too can Internet hosts
  - **hostname**
    - e.g., www.google.com, used by humans
    - difficult to process by router
  - **IP address** (32 bit)
- Q: how to map between IP addresses and name, and vice versa?
  - a directory service that translates hostnames to IP address

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *DNS servers*
- *application-layer protocol* allows hosts to query the distributed database for name
- DNS servers are often UNIX machines running the Berkeley Internet Name Domain software
- The DNS protocol runs over **UDP** and users port **53**

DNS: Domain Name System

- DNS is commonly used by other app. layer protocols to
  - translate user-supplied ***hostnames*** to ***IP address***
- E.g.: requesting the URL *www.someschool.edu*
  1. the same user machine runs the client side of the DNS application.
  2. the browser extracts the hostname, www.someschool.edu, from the URL and passes the hostname to the client side of the DNS application.
  3. the DNS client sends a query containing the hostname to a DNS server.
  4. the DNS client eventually receives a reply, which includes the IP address for the hostname.
  5. once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

DNS: Domain Name System (cont.)

DNS services

- **hostname to IP address translation**
- **host aliasing**
  - **canonical names**
    - `relay1.west-coast.enterprise.com`
  - **alias names**
    - `enterprise.com` or
    - `www.enterprise.com`
- **load distribution**
  - perform load distribution among replicated Web servers
    - busy sites, e.g., cnn.com, are replicated over multiple servers, each server running on a different end system with a different IP address
  - rotate the ordering of address within each reply

Simple design for DNS:

- one DNS server
  - containing all mappings
  - centralized design

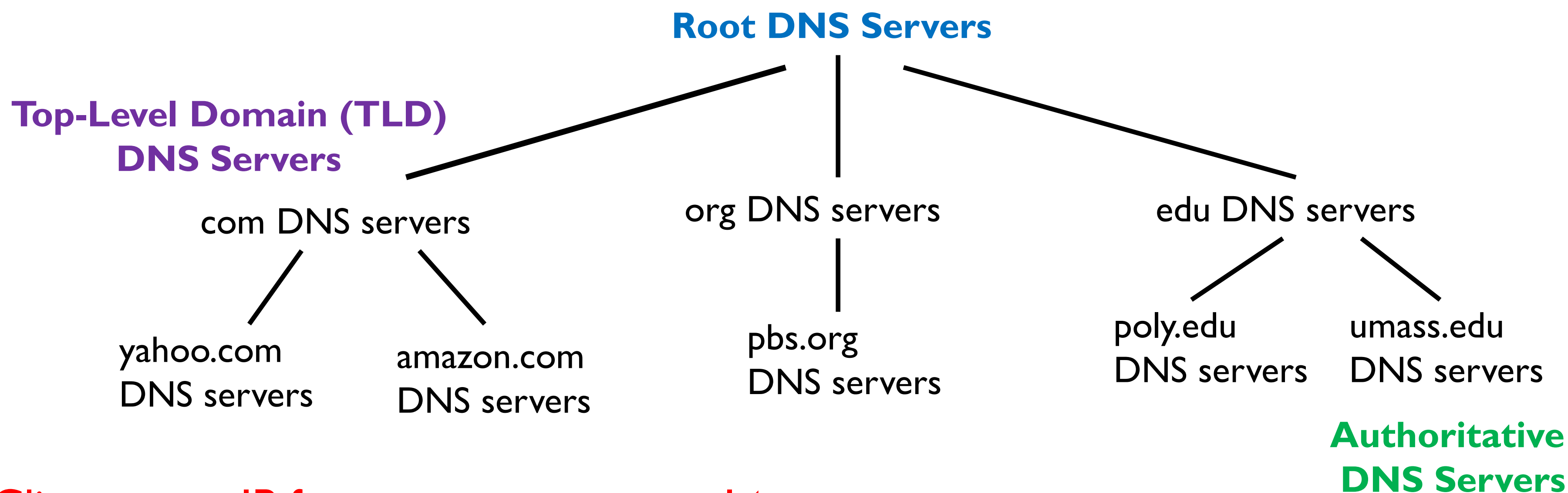
Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn't scale!

Distributed, Hierarchical Database

No single DNS server has all of the mappings for all of the hosts in the Internet



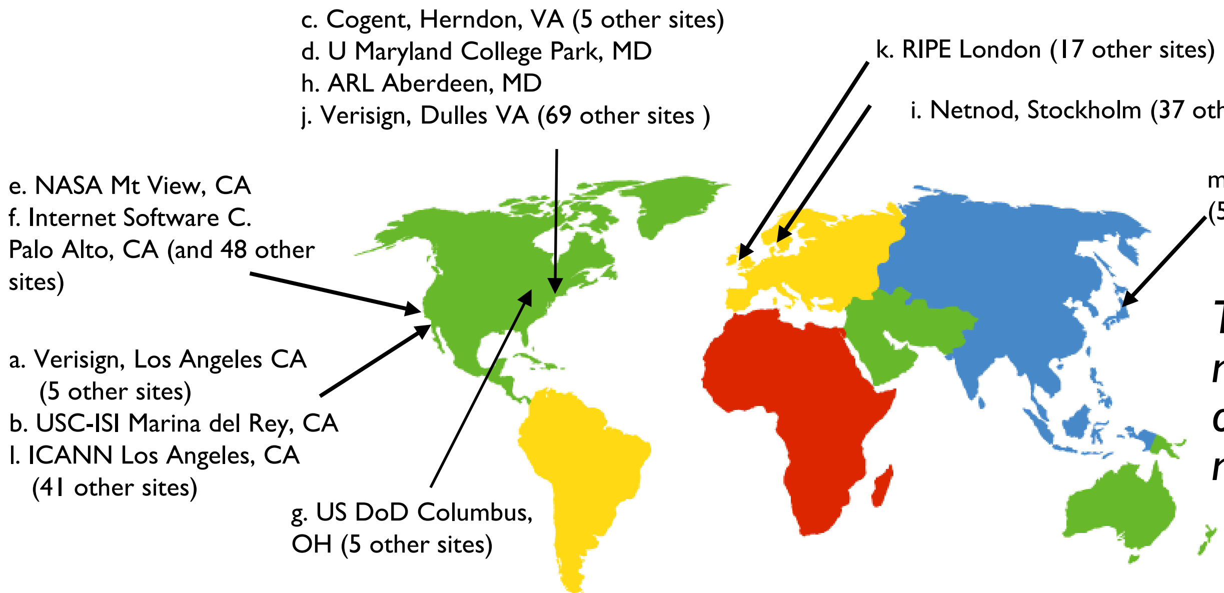
Client wants IP for www.amazon.com; 1st approx:

- client queries a root server, which returns IP addresses for TLD servers for top-level domain `com`
- client queries TLD server, which returns IP address of authoritative server for `amazon.com`
- client queries `authoritative` DNS server to get IP address for `www.amazon.com`

DNS: Root Name Servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts lower-level name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

over 400 root name "servers" worldwide, each "server" replicated many times



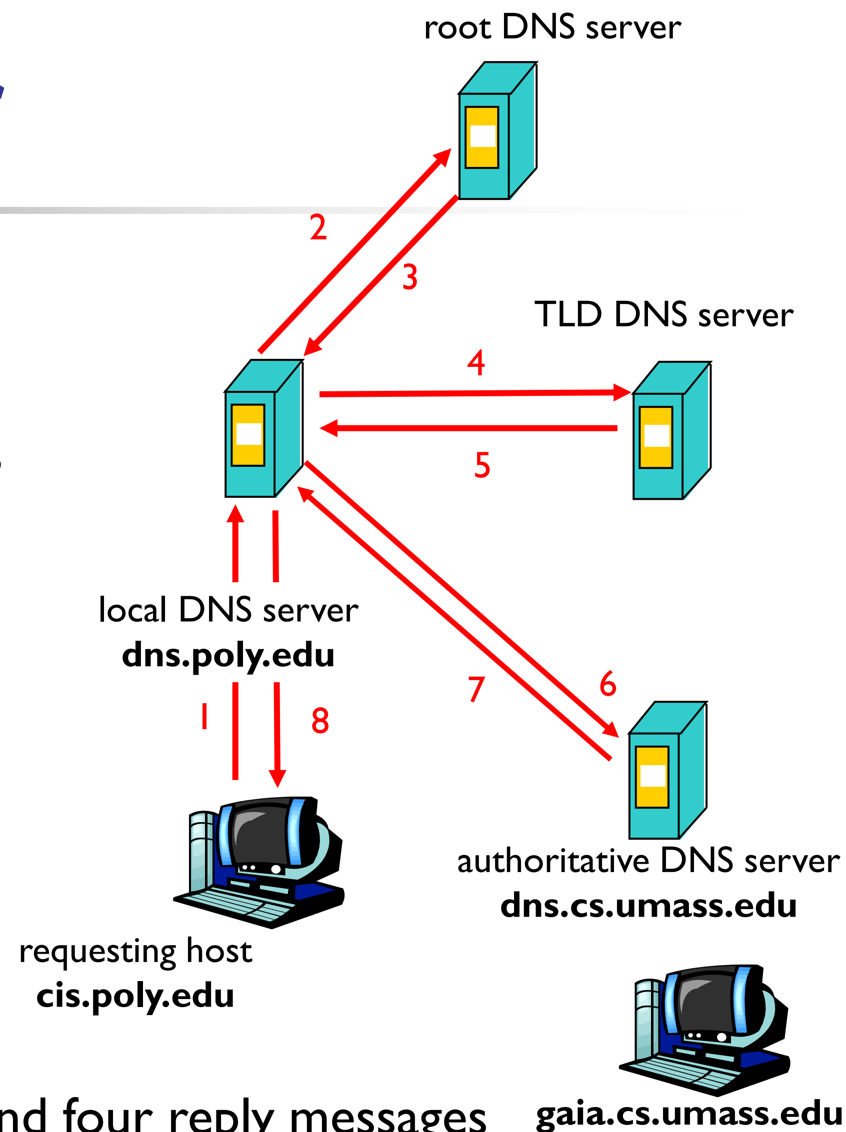
These root name servers are managed by 13 different organizations (labeled a through m).

TLD and Authoritative Servers

- top-level domain (TLD) servers:
  - responsible for `com`, `org`, `net`, `edu`, etc, and all top-level country domains `uk`, `fr`, `ca`, `jp`.
  - Verisign Global Registry Services - maintains servers for `com` TLD
  - Educause - for `edu` TLD
- authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - can be maintained by organization or service provider
  - most universities and large companies implement and maintain their authoritative DNS server

Local DNS Server

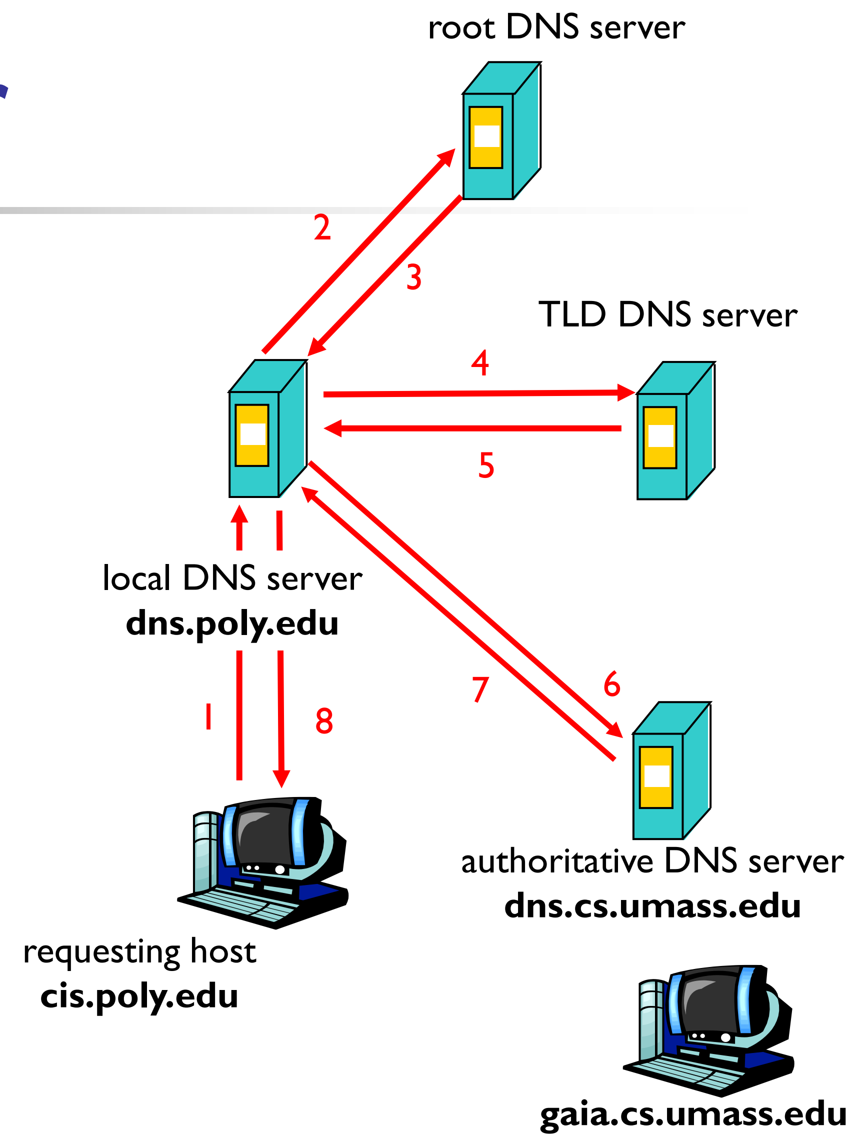
- a **local DNS server** does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one local DNS server
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy



eight DNS messages sent: four query message and four reply messages

Local DNS Server

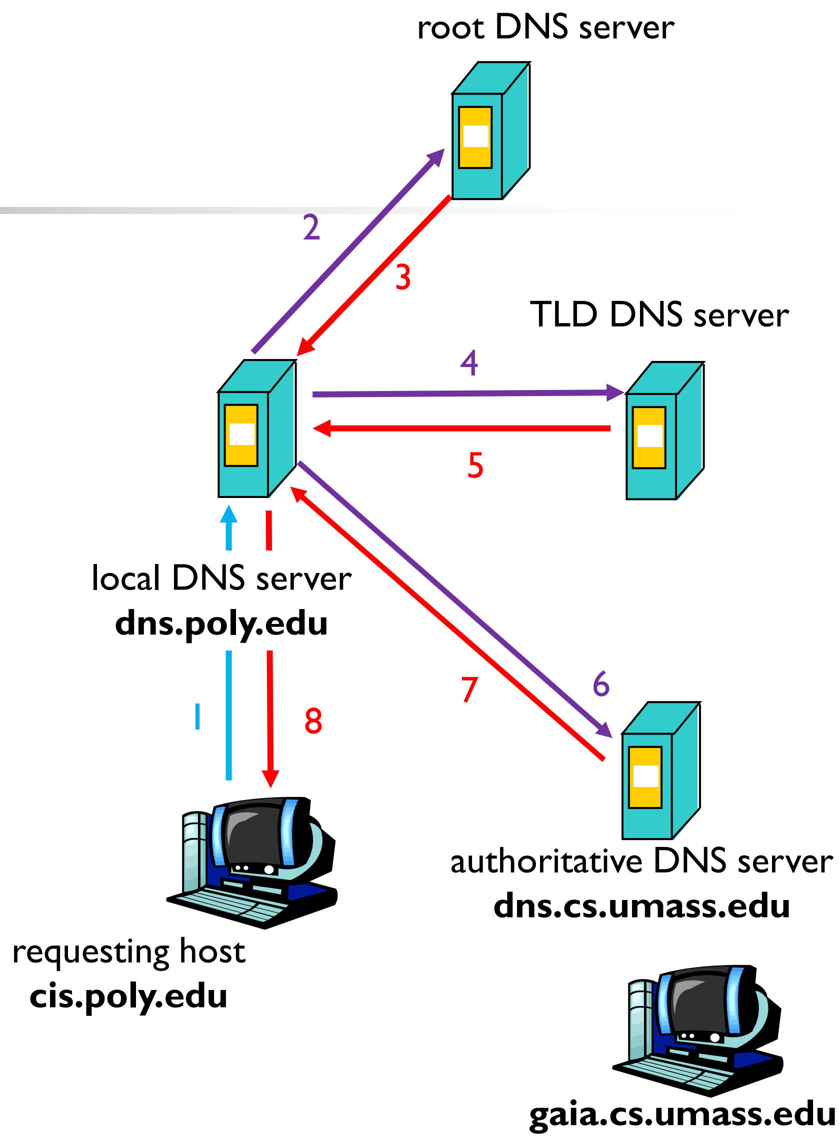
- DNS name resolution example
  - Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
  - **iterated query**:
    - contacted server replies with name of server to contact
    - "I don't know this name, but ask this server"
  - **recursive query**:
    - puts burden of name resolution on contacted name server



MARSHALL

Local DNS Server

- DNS name resolution example
  - Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
  - **iterated query**:
    - contacted server replies with name of server to contact
    - "I don't know this name, but ask this server"
  - **recursive query**:
    - puts burden of name resolution on contacted name server



DNS: Caching and Updating Records

- when a DNS server receives a DNS reply (containing a mapping from a host name to an IP address), it can cache the mapping in its local cache
  - a local DNS server can cache the IP addresses of TLD servers
    - allow the local DNS server can bypass the root DNS servers in a query chain
    - thus root name servers not often visited
  - cache entries timeout (disappear) after some time