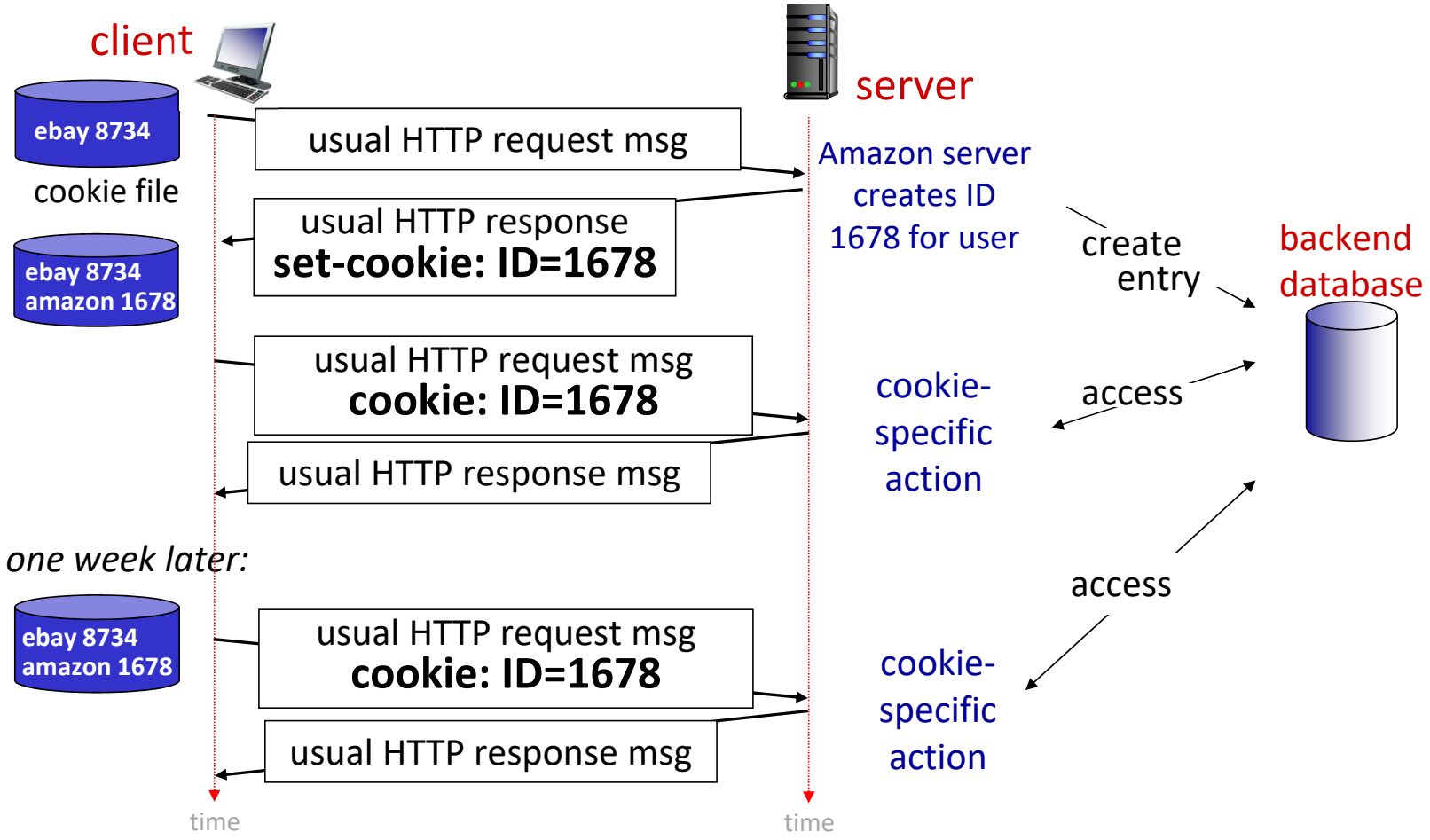
Maintaining user/server state: cookie

- *state* is very helpful for:
  - identification/authorization
  - shopping carts
  - recommendations

- Challenge: HTTP itself is *stateless*

- Where to keep state?
  - at protocol endpoints
    - client side: browser
    - server side: backend database
  - in messages
    - HTTP messages carry state
- How exactly?
  - *cookie*
  - local storage
  - IndexedDB data
  - Session storage
  - …

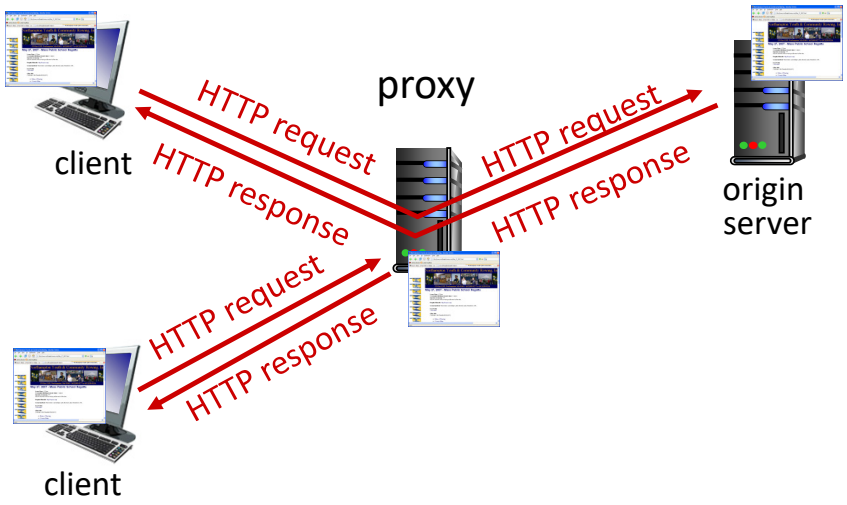
Maintaining user/server state: cookie



Web cache (or called proxy)

- *Goal:* satisfy client requests on behalf of origin server
  - user configures browser to point to a (local) *Web cache*
- *Why* web caching?
  - reduce response time for client request
    - cache is closer to client
  - reduce traffic on ISP's access link
  - enables "poor" content providers to more effectively deliver content
  - anonymity

- browser/client sends all HTTP requests to proxy
  - *if* object not in proxy: proxy requests object from origin server, caches received object, then returns object to client
  - *else* proxy returns object to client



Check freshness by conditional GET

Goal: don't send object if proxy has cached up-to-date version

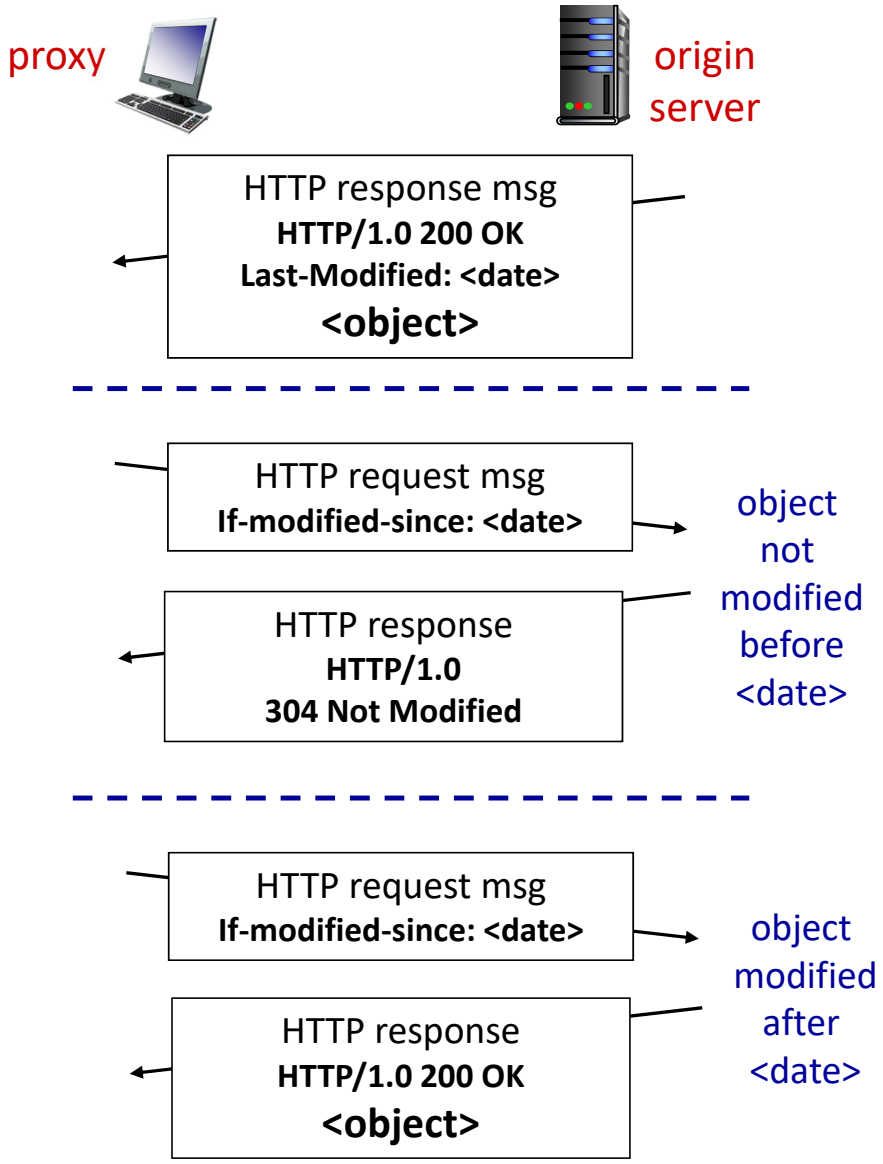
- "If-modified-since" header
- "Expires" or "max-age" header

- *proxy:* specify date of cached copy in HTTP request

  If-modified-since: <date>

- *origin server:* response contains no object if cached copy is up-to-date:

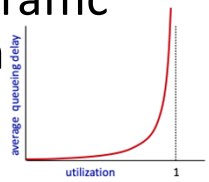
  HTTP/1.0 304 Not Modified

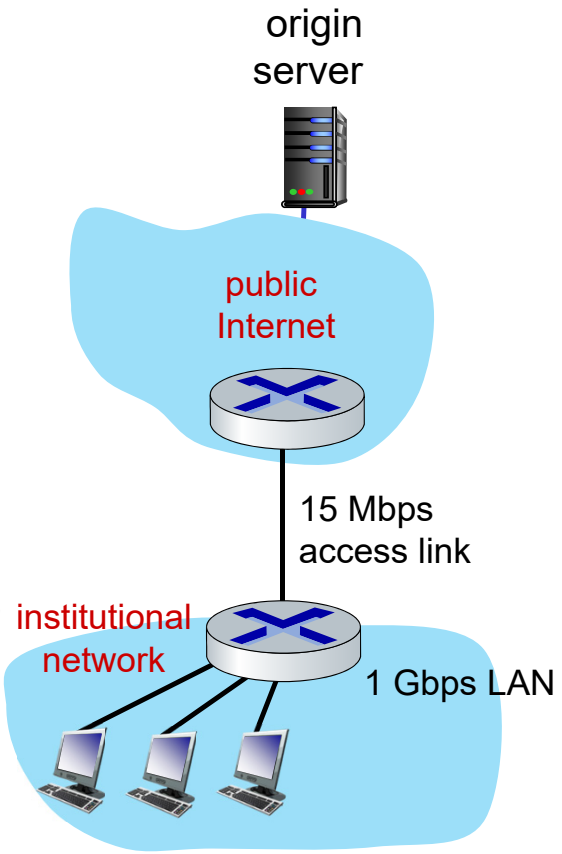


Caching example

Scenario:

- access link rate: 15 Mbps
- rate of demand over access link/LAN = ?
  - web object size: 0.1 Mb
  - average request rate: 150 requests/sec
- one-way delay in public Internet: 2 seconds
- link/LAN delay is 10 ms at low or medium traffic
  is minutes at high utilization





Performance:

- access link utilization = 15/15 = 1
- LAN utilization: 15M/1G = 0.015
- end-to-end delay
  = Internet delay + access link delay + LAN delay
  ≈ 2 sec + minutes + 10 ms

access link is the bottleneck

Option 1: buy a faster access link

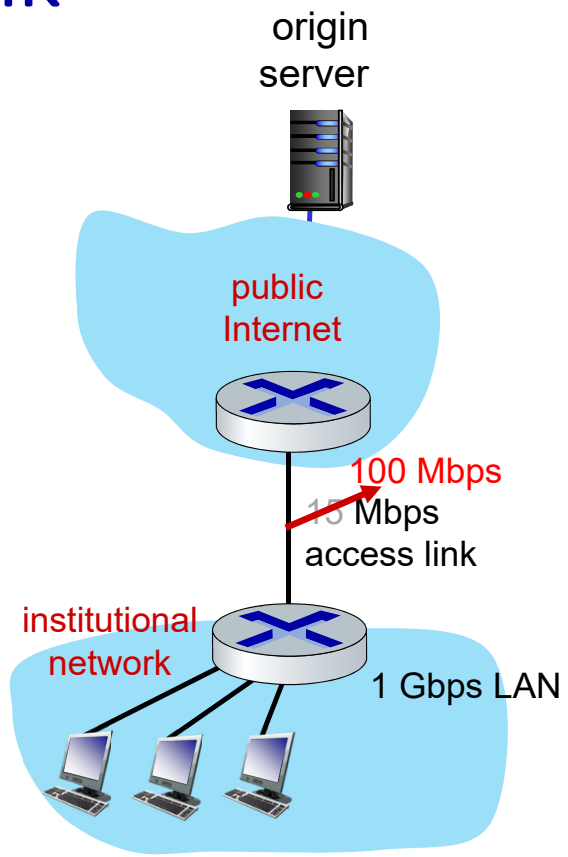
Scenario:

- access link rate: ~~15~~ 100 Mbps
- rate of demand over access link/LAN = 15 Mbps
  - web object size: 0.1 Mb
  - average request rate: 150 requests/sec
- one-way delay in public Internet: 2 seconds
- link/LAN delay is 10 ms at low or medium traffic is minutes at high utilization

Performance:

- access link utilization = ~~1~~ 15/100 = 0.15
- LAN utilization: 0.015
- end-to-end delay
  = Internet delay + access link delay + LAN delay
  ≈ 2 sec + ~~minutes~~ 10 ms + 10 ms

Cost: faster access link (expensive!)



Option 2: install a proxy

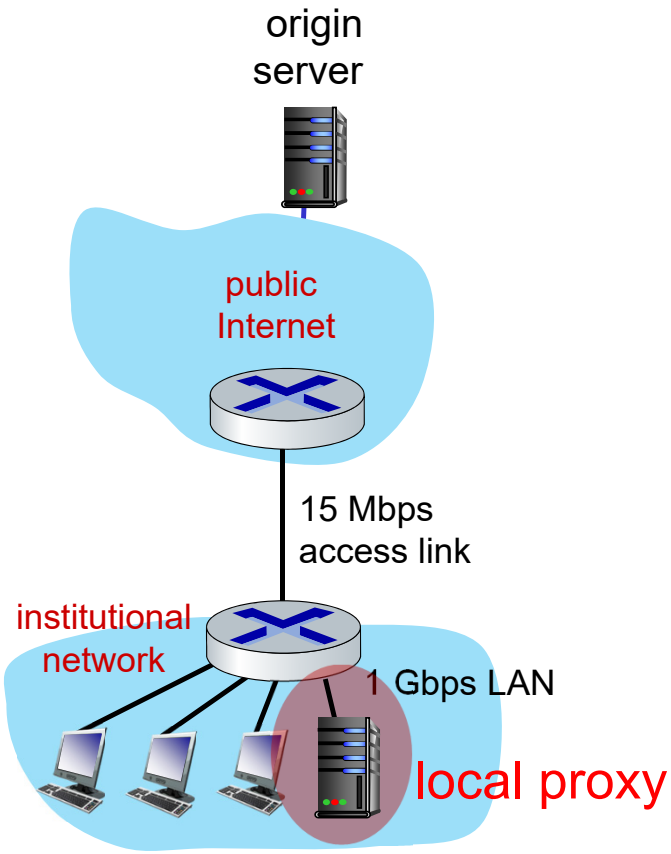
Scenario:

- access link rate: 15 Mbps
- rate of demand over access link/LAN = 15 Mbps
  - web object size: 0.1 Mb
  - average request rate: 150 requests/sec
- one-way delay in public Internet: 2 seconds
- link/LAN delay is 10 ms at low or medium traffic is minutes at high utilization

Cost: proxy (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

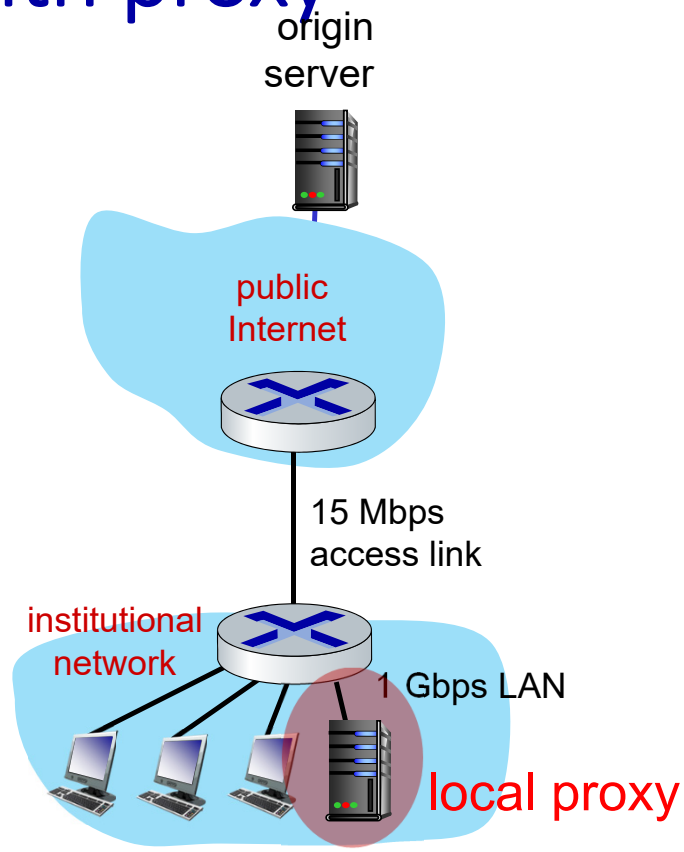


Calculating end-to-end delay with proxy

suppose cache hit rate is 0.4:

- each request satisfied by proxy takes
  ≈ 10 ms = 0.01 s
- each request served by origin server takes
  - access link utilization = 0.6 * 15 / 15 = 0.6
  - access link delay ≈ 10 ms
  - end-to-end delay
    ≈ 2 s + 10 ms + 10 ms ≈ 2.02 s
- average end-end delay:
  = 0.6 * (delay from origin server)
  + 0.4 * (delay when satisfied by proxy)
  ≈ 0.6 * 2.02 s + 0.4 * 0.01 s
  ≈ 1.2 secs

lower avg end-end delay than with 100 Mbps link (and cheaper too!)

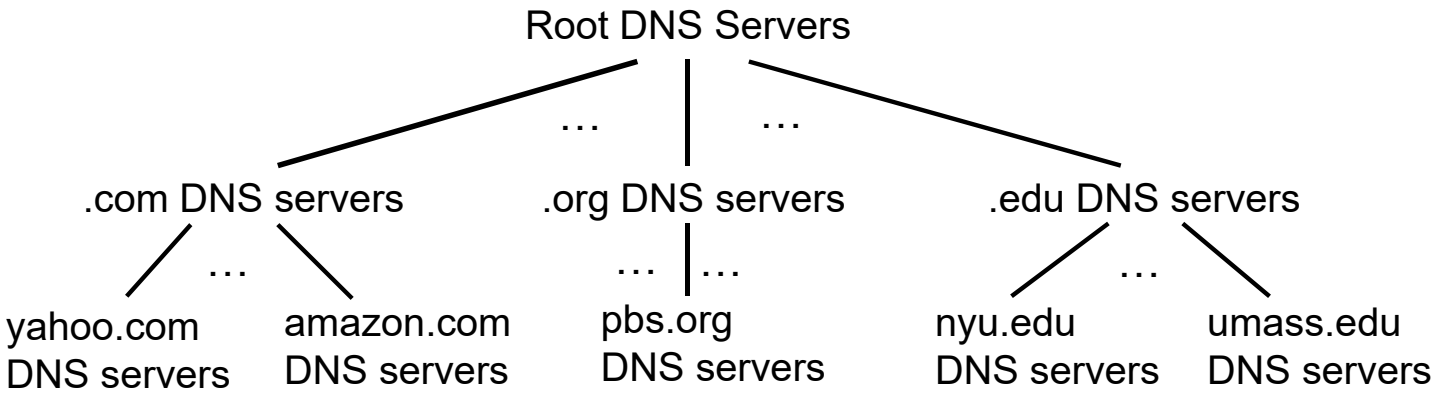


Brief introduction to Domain Name System (DNS)

- For addressing
  - people use identifiers
    - Ex. www.nthu.edu.tw
  - Internet hosts and routers use IP addresses
    - Ex. 140.114.69.135
- Translation between IP address and name needs DNS

- DNS
  - distributed database
    - implemented in hierarchy of many name servers
  - application-layer protocol
    - hosts and DNS servers communicate to resolve names (address/name translation)
  - complexity at network's "edge"
    - traffic volume
      - Ex. Akamai DNS servers alone: 2.2T DNS queries/day

DNS: a distributed, hierarchical database

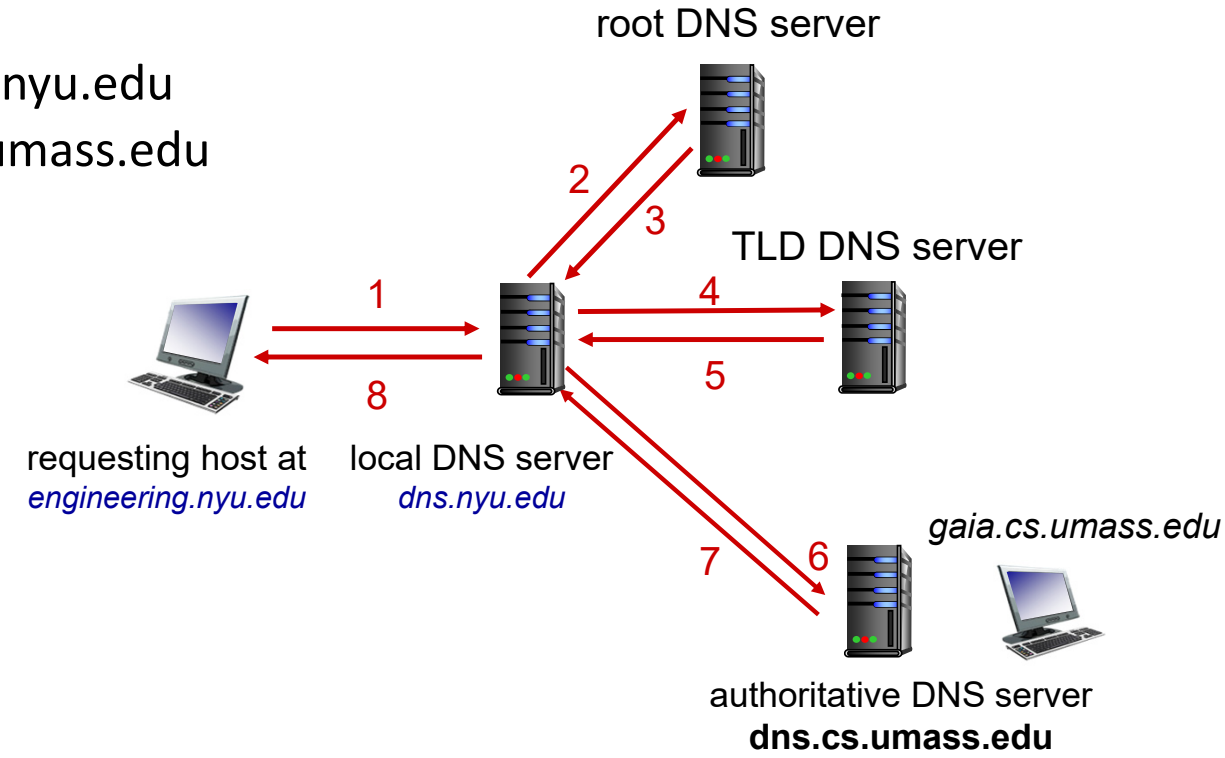


Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS name resolution: iterated query

Example: host at engineering.nyu.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"

Can have more layers (in-between TLD and authoritative)

- Ex. root, .edu, nthu.edu, dns.cs.nthu.edu.tw

More on DNS

- Can have more layers (in-between TLD and authoritative)
  - Ex. root, .edu, nthu.edu, dns.cs.nthu.edu.tw
- DNS caching
  - once (any) name server learns mapping, it caches mapping, and immediately returns a cached mapping in response to a query
    - caching improves response time
    - cache entries timeout (disappear) after some time (TTL)
    - TLD servers typically cached in local name servers