Computer Network

2023 Fall

Chapter 2
Application Layer

A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR
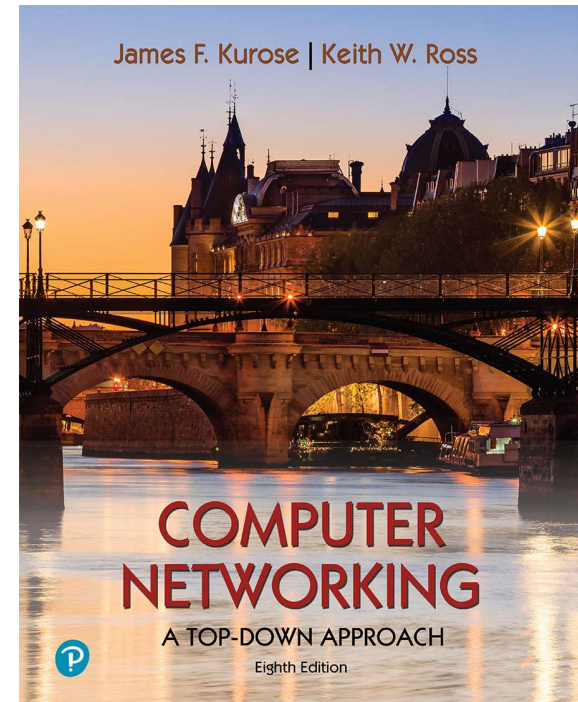
Computer Networking: A Top-Down Approach
8th edition   n
Jim Kurose, Keith Ross
Pearson, 2020

Some network apps

- 1970-80
  - e-mail
  - remote login
  - File transfer

- 1990
  - Internet search
  - Web
  - E-commerce

- 21C
  - voice over IP (e.g., Skype)
  - social networking
  - multi-user network games
  - streaming stored video (YouTube, Hulu, Netflix)
  - real-time video conferencing (e.g., Zoom)
  - P2P file sharing

Applications

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Creating a network app

write programs that:

- run on (different) end systems

- communicate over network

- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications

- applications on end systems allows for rapid app development, propagation
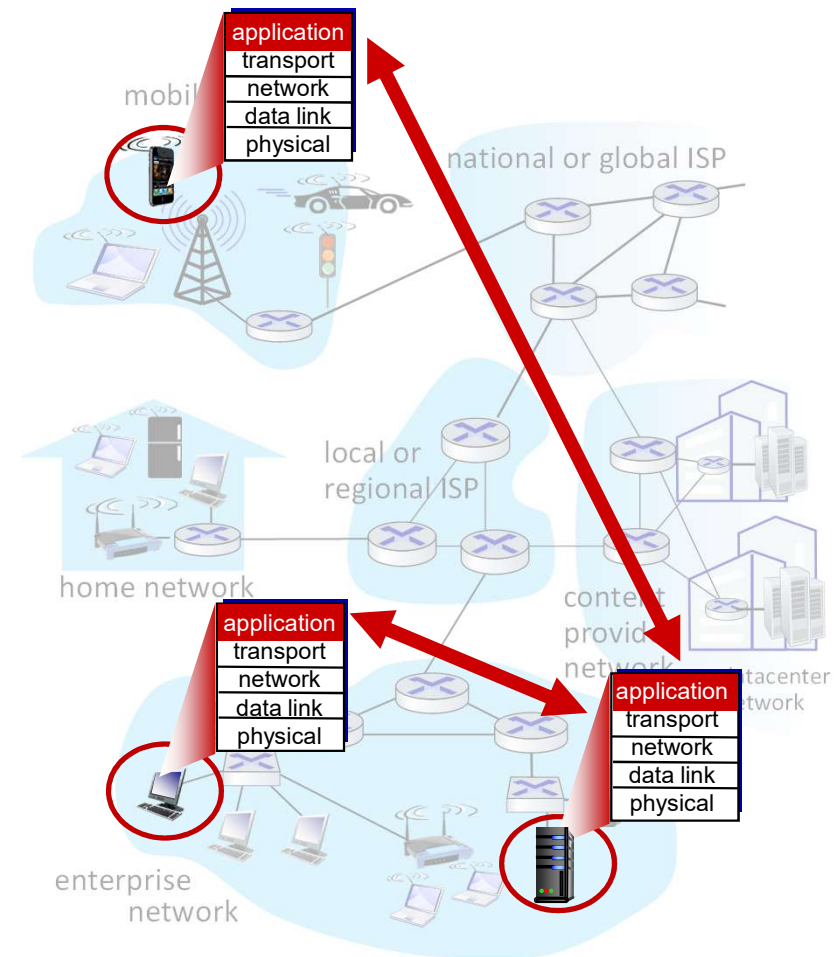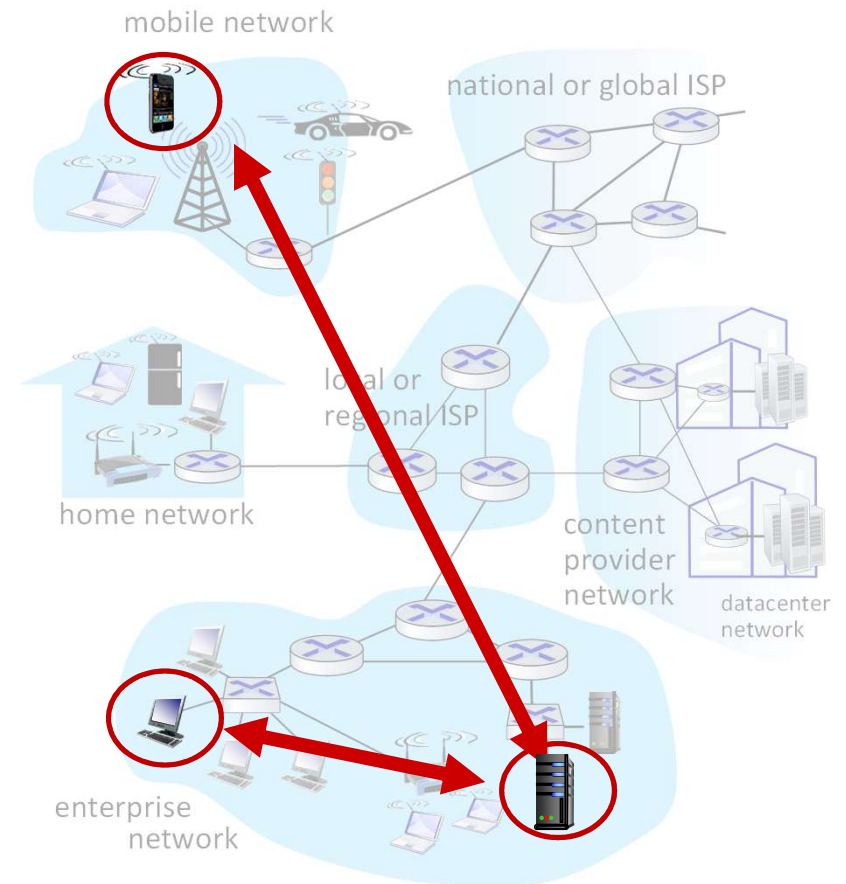
Client-server paradigm

server:
- always-on host
- permanent IP address
- often in data centers, for scaling

clients:
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP

Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- example: P2P file sharing [BitTorrent]
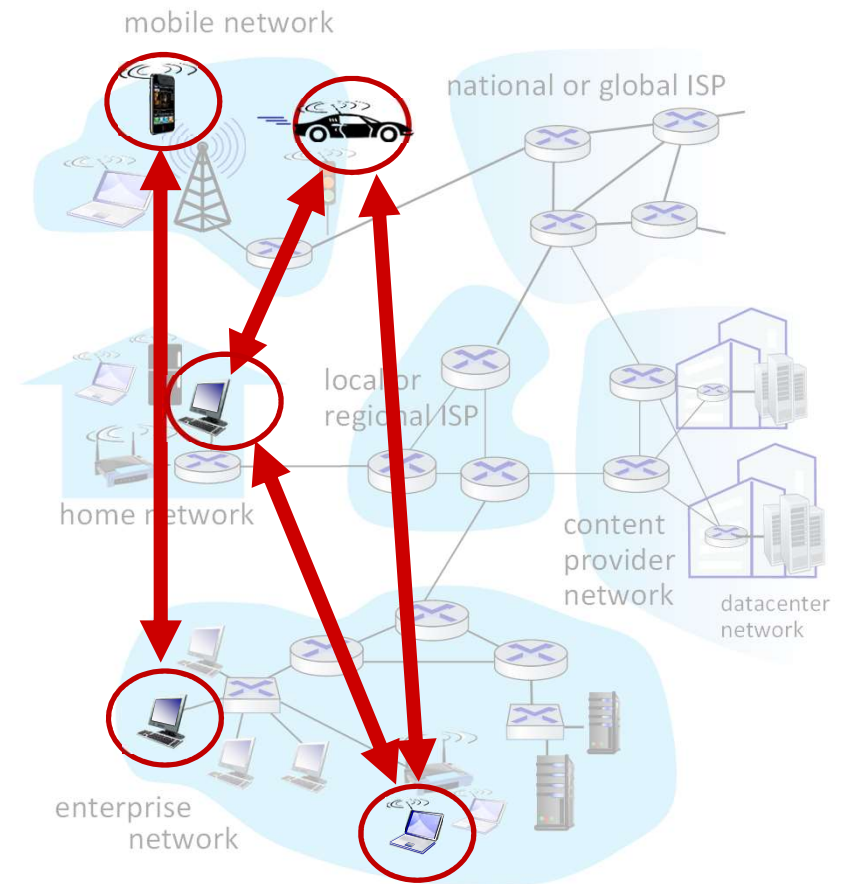
Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)

- processes in different hosts communicate by exchanging messages

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

- note: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its socket

- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side
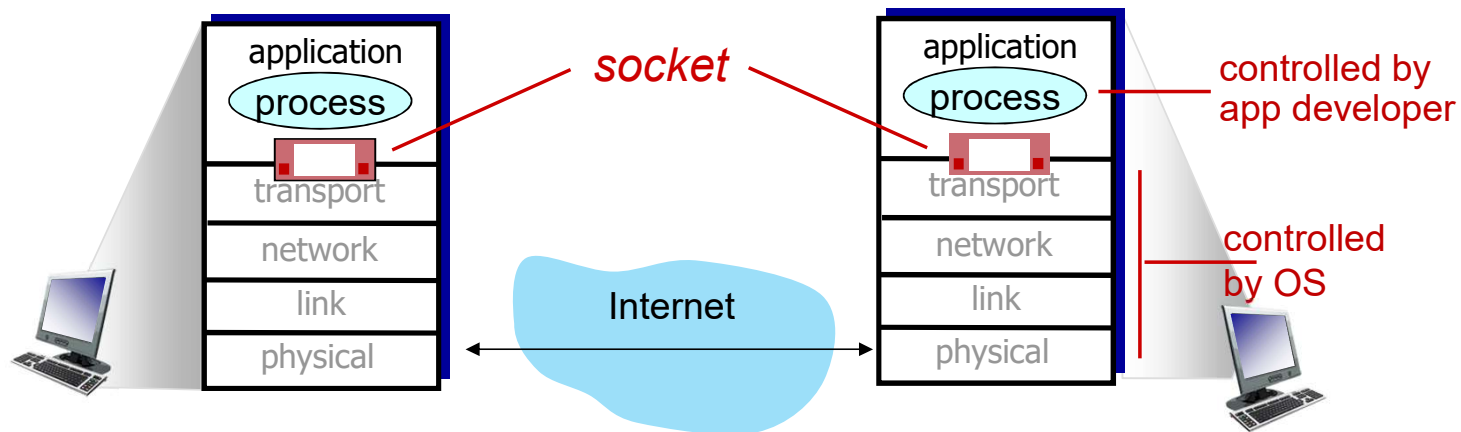
Addressing processes

- to receive messages, process must have *identifier*

- host device has unique 32-bit IP address

- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.

- example port numbers:
  - HTTP server: 80
  - mail server: 25

- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

- more shortly…

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, ...

Transport service requirements: common apps

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process

- *flow control:* sender won't overwhelm receiver

- *congestion control:* throttle sender when network overloaded

- *connection-oriented:* setup required between client and server processes

- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *unreliable data transfer* between sending and receiving process

- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? *Why is there a UDP?*

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Question

- Why do HTTP, FPT, SMTP, and POP3 run on top of TCP rather than on UDP?

Securing TCP

Vanilla TCP & UDP sockets:

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)

- provides encrypted TCP connections
- data integrity
- end-point authentication

TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into "socket" traverse Internet *encrypted*
- more: Chapter 8

An application-layer protocol defines:

- **types of messages exchanged,**
  - e.g., request, response
- **message syntax:**
  - what fields in messages & how fields are delineated
- **message semantics**
  - meaning of information in fields
- **rules** for when and how processes send & respond to messages

open protocols:
- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:
- e.g., Skype, Zoom

Application layer: overview

- Principles of network applications
- **Web and HTTP**
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Web and HTTP

First, a quick review...

- web page consists of *objects,* each of which can be stored on different Web servers

- object can be HTML file, JPEG image, Java applet, audio file,...

- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name   path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

PC running
Firefox browser

HTTP request
HTTP response

HTTP request
HTTP response

server running
Apache Web
server

iPhone running
Safari browser
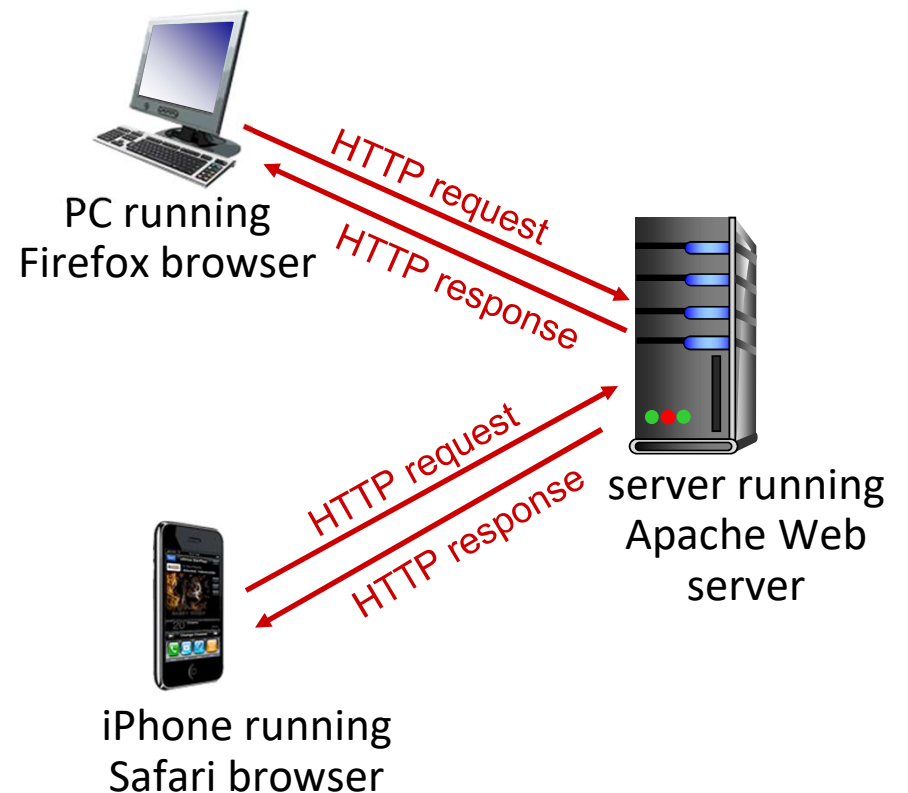
HTTP overview (continued)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80

- server accepts TCP connection from client

- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

- TCP connection closed
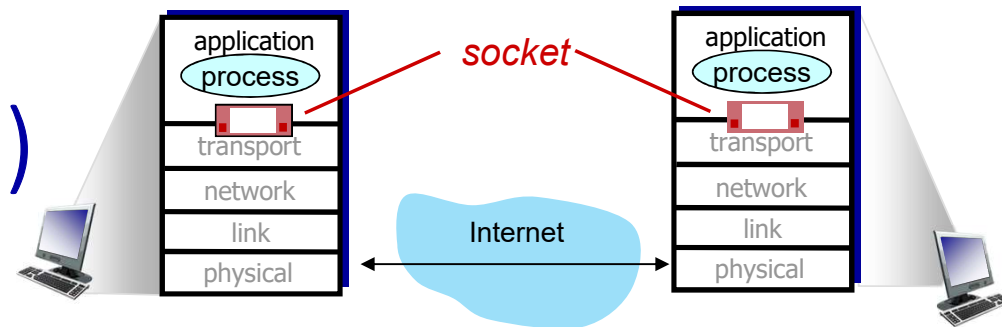
HTTP is "stateless"

- server maintains *no* information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained

- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened

2. at most one object sent over TCP connection

3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server

- multiple objects can be sent over *single* TCP connection between client, and that server

- TCP connection closed

Non-persistent HTTP: example

User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: `www.someSchool.edu/someDepartment/home.index`

(containing text, references to 10 jpeg images)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects
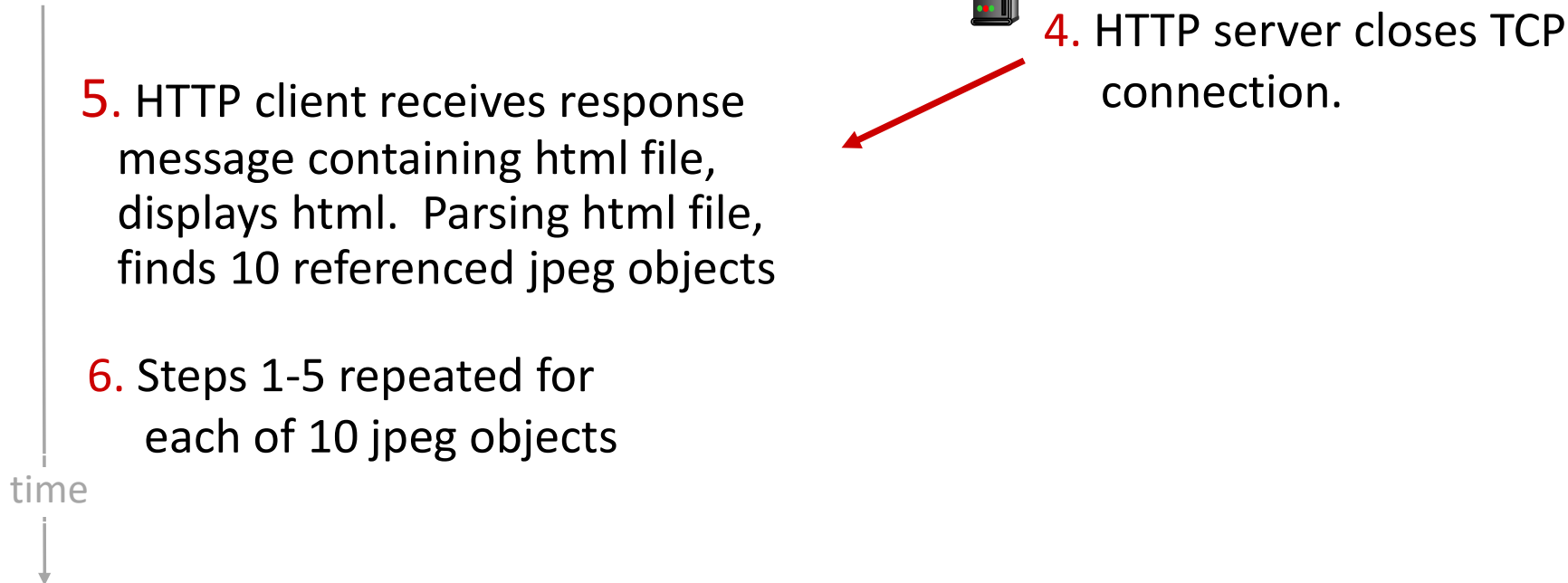
time

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time

initiate TCP connection

RTT

request file

RTT

time to transmit file

file received
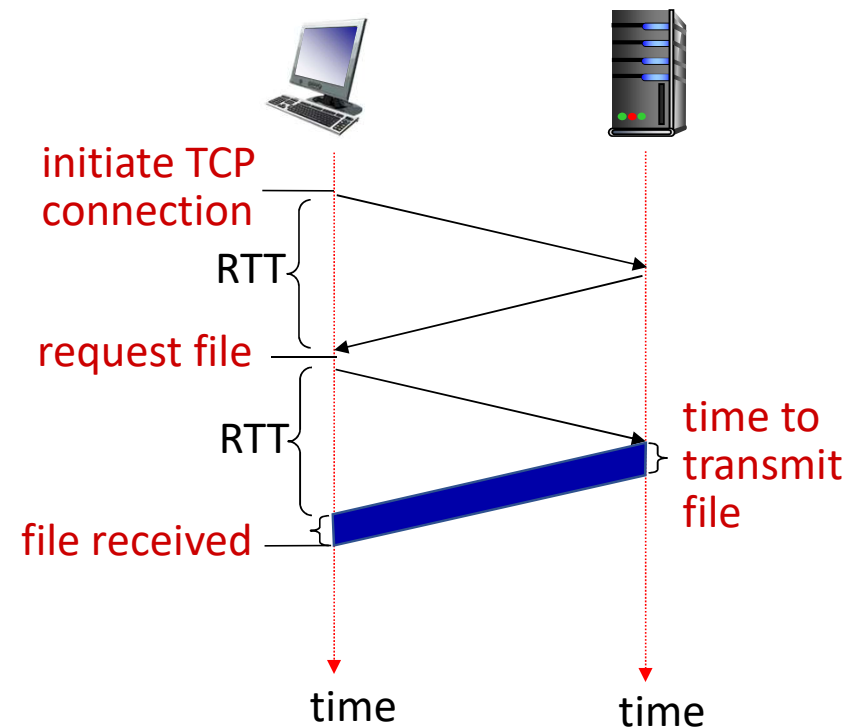
time   time

Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

- requires 2 RTTs per object

- OS overhead for *each* TCP connection

- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response

- subsequent HTTP messages between same client/server sent over open connection

- client sends requests as soon as it encounters a referenced object

- as little as one RTT for all the referenced objects (cutting response time in half)

HTTP request message

- two types of HTTP messages: *request, response*

- HTTP request message:
  - ASCII (human-readable format)

carriage return character
line-feed character

request line (GET, POST, HEAD commands)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
    10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

header lines

carriage return, line feed at start of line indicates end of header lines
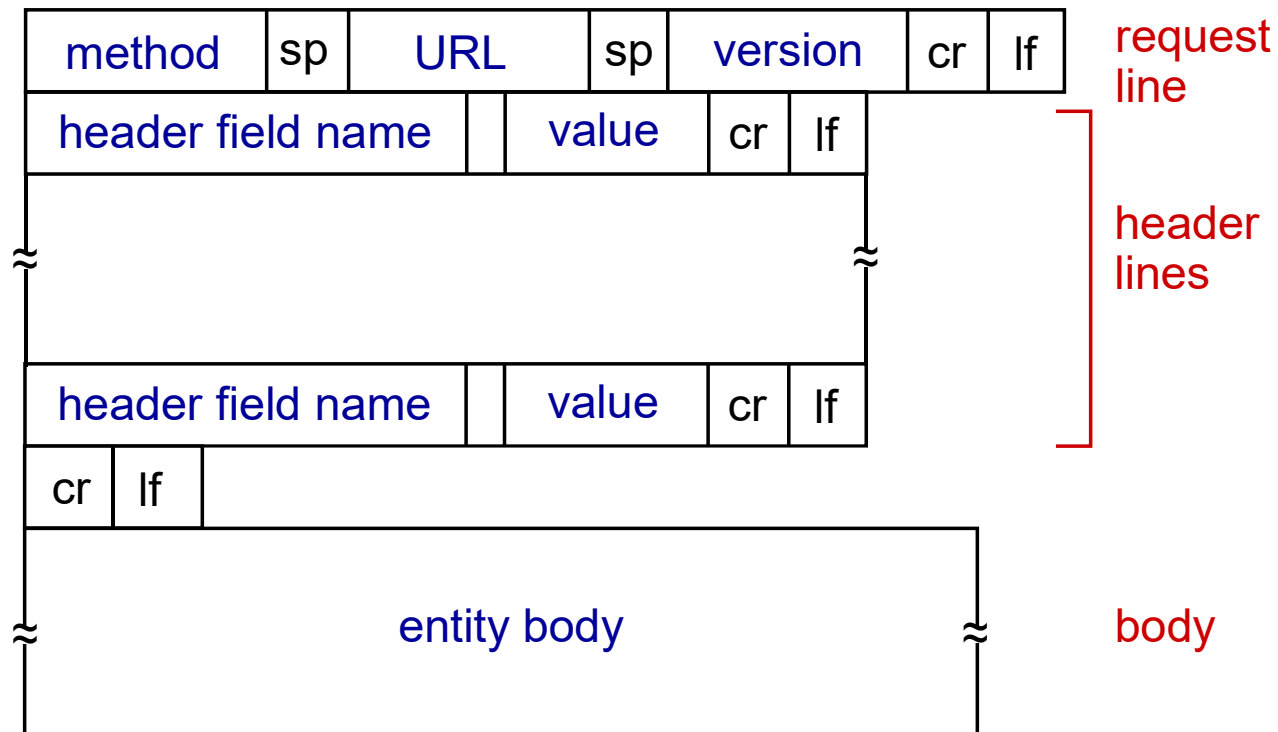
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message
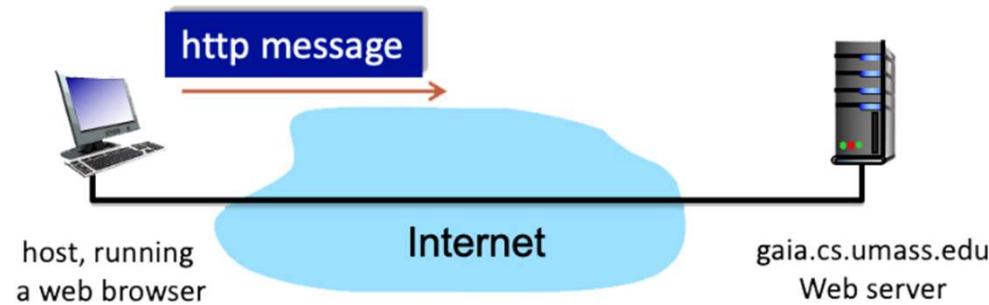
HTTP request message: general format

| method | sp | URL | sp | version | cr | lf |

request line

| header field name | | value | cr | lf |

≈ ≈

| header field name | | value | cr | lf |

header lines

| cr | lf |

≈ entity body ≈

body

Problems

- A client is sending an HTTP GET message to a web server: gaia.cs.umass.edu

GET /kurose_ross_sandbox/interactive/quotation8.htm HTTP/1.1
Host: gaia.cs.umass.edu
Accept: text/plain, text/html, image/jpeg, image/png, audio/basic, audio/mpeg, video/wmv, video/mpeg,
Accept-Language: en-us, en-gb;q=0.1, en;q=0.6, fr, fr-ch, zh, da, de, fi, ar
If-Modified-Since: Mon, 25 Sep 2023 02:29:25 -0700
User Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1

1) What is the name of the file that is being retrieved in this GET message?

2) The client will accept gif images? (true or false)

3) The client already has a cached copy of the file? (true or false)

Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

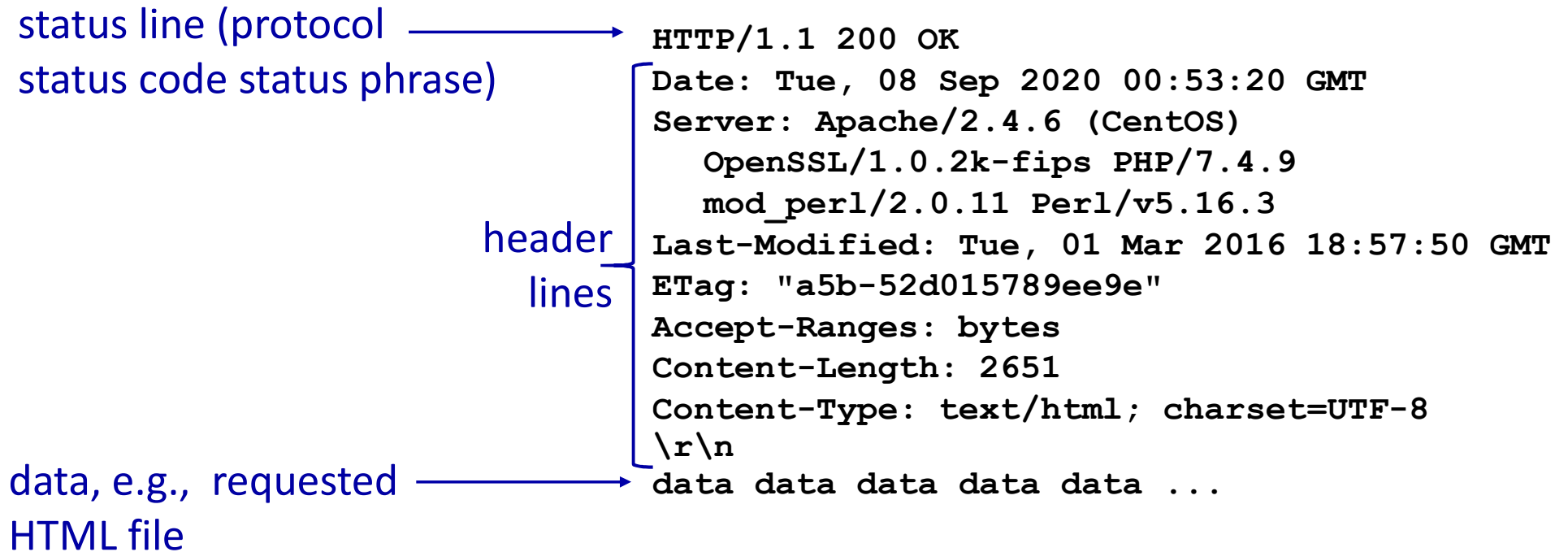**https://www.google.com/search?q=monkeys&sca_esv=5668...**

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message

status line (protocol status code status phrase)   ⟶

data, e.g., requested HTML file   ⟶

header lines

```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2020 00:53:20 GMT
Server: Apache/2.4.6 (CentOS)
   OpenSSL/1.0.2k-fips PHP/7.4.9
   mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8
\r\n
data data data data data ...
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (in Location: field)
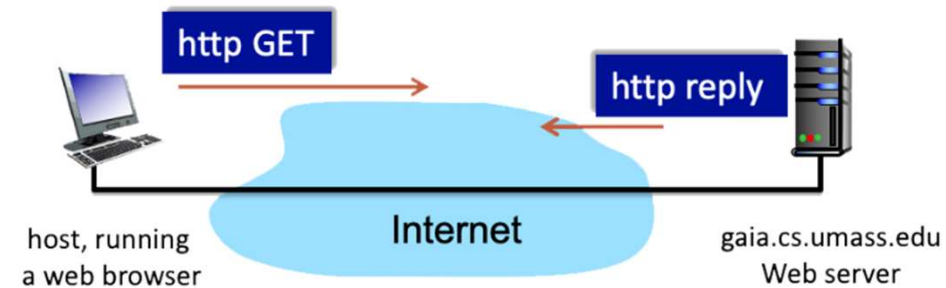
400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Problems

- A server is sending a HTTP response message back to the client

> *HTTP/1.0 404 Not Found*
>
> *Date: Mon, 25 Sep 2023 10:12:25 +0000*
>
> *Server: Apache/2.2.3 (CentOS)*
>
> *Content-Length: 706*
>
> *Connection: Close*
>
> *Content-type: text/html*

1) Was the server able to send the document successfully? (yes or no)

2) How big is the document in bytes?

3) Is the connection persistent or nonpersistent?

Maintaining user/server state: cookies

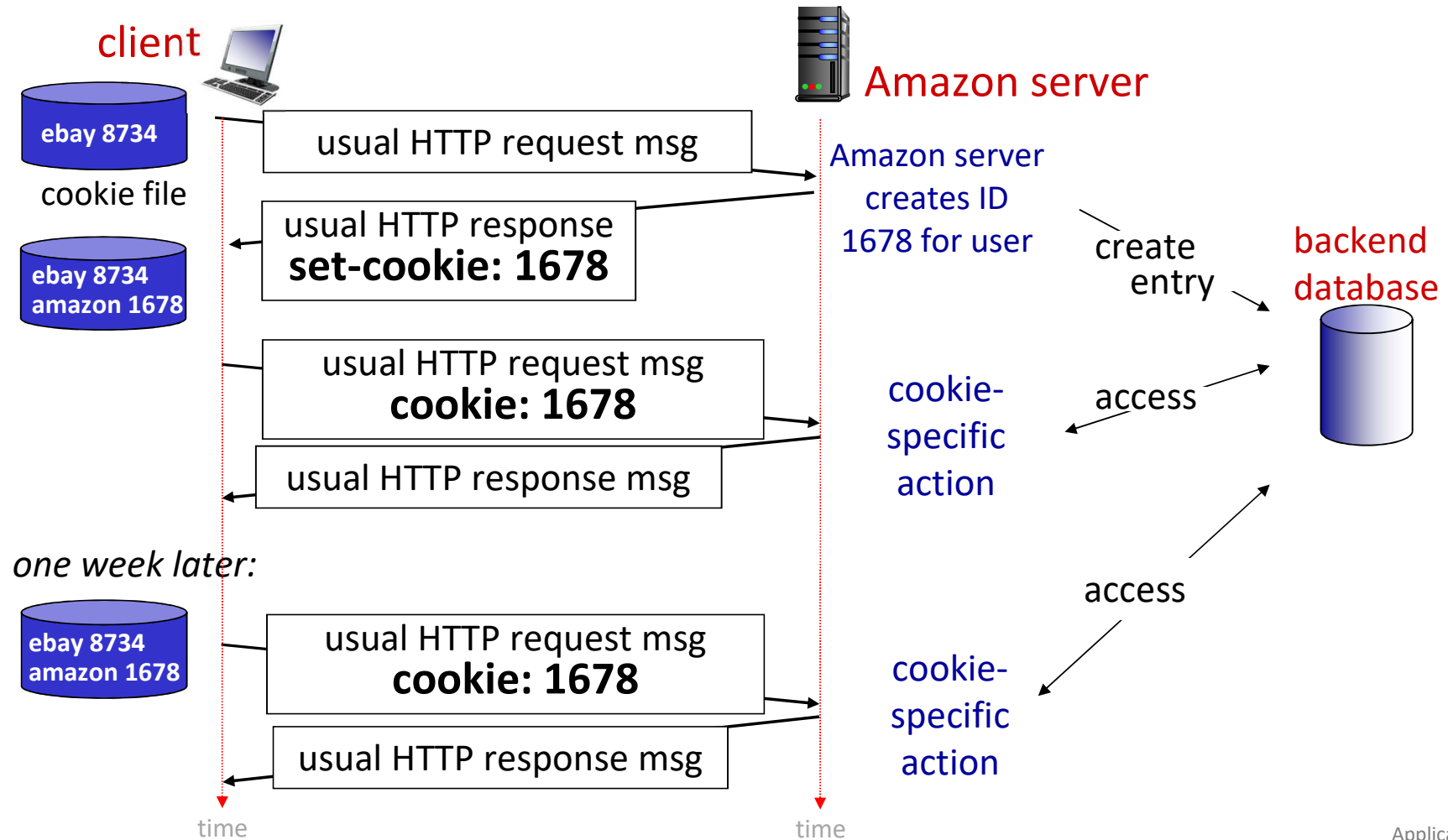Web sites and client browser use *cookies* to maintain some state between transactions

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) back-end database at Web site

Example:

- Susan uses browser on laptop, visits specific Amazon site 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 Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies

client

ebay 8734

cookie file

**ebay 8734
amazon 1678**

Amazon server

usual HTTP request msg → Amazon server creates ID 1678 for user

usual HTTP response **set-cookie: 1678**

create entry → backend database

usual HTTP request msg **cookie: 1678** → cookie-specific action

usual HTTP response msg ← access ← backend database

one week later:

**ebay 8734
amazon 1678**

usual HTTP request msg **cookie: 1678** → cookie-specific action

usual HTTP response msg ← access

time   time

HTTP cookies: comments

What cookies can be used for:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Challenge: How to keep state?
- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies in HTTP messages carry state
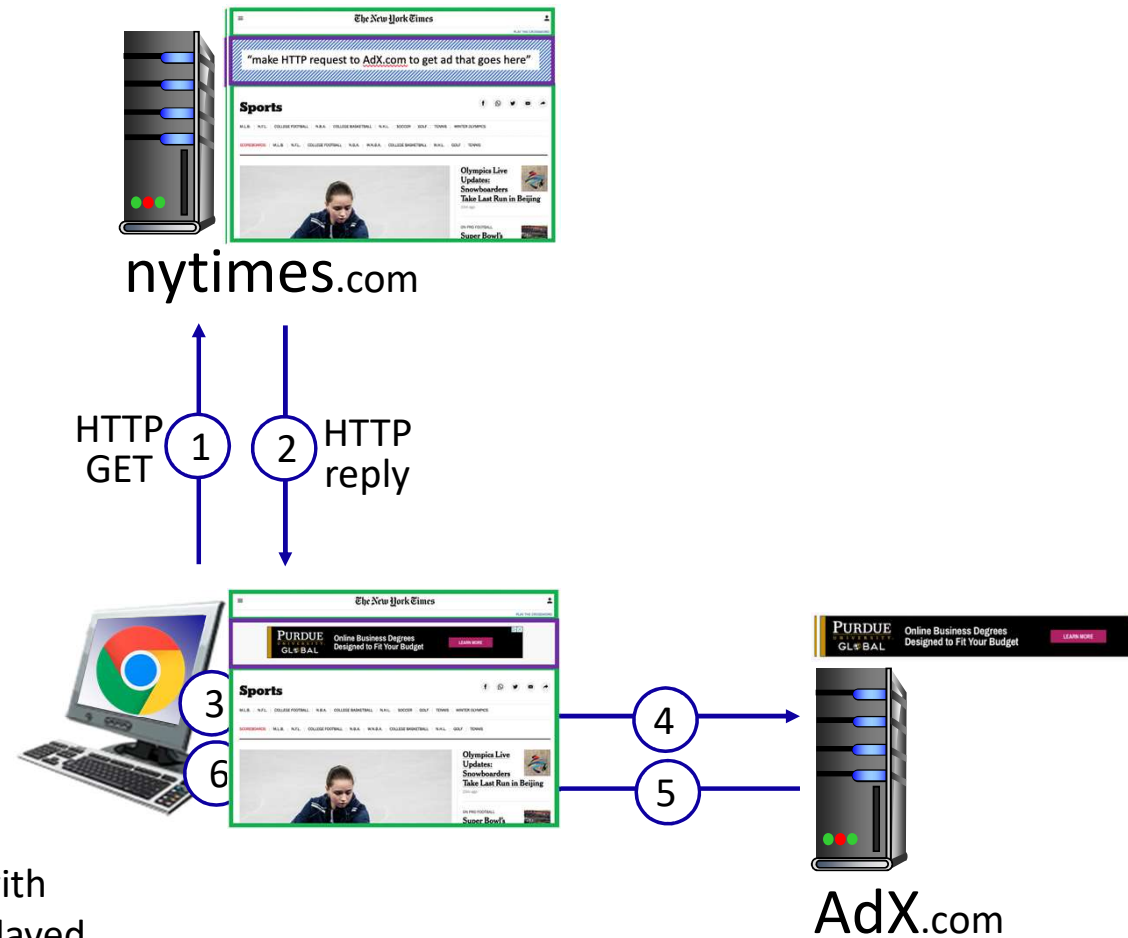
cookies and privacy:
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

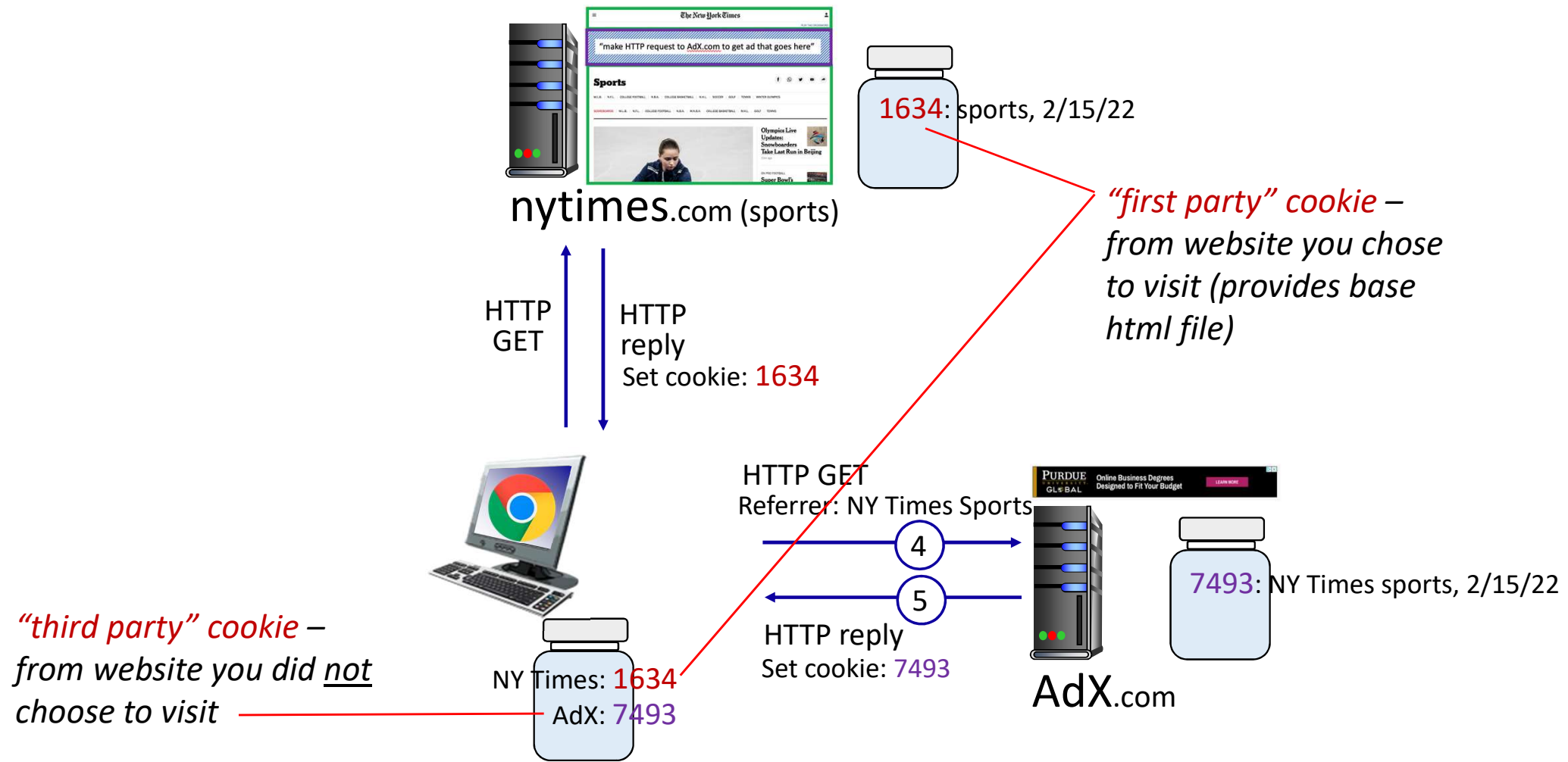# Example: displaying a NY Times web page

① ② GET base html file from nytimes.com
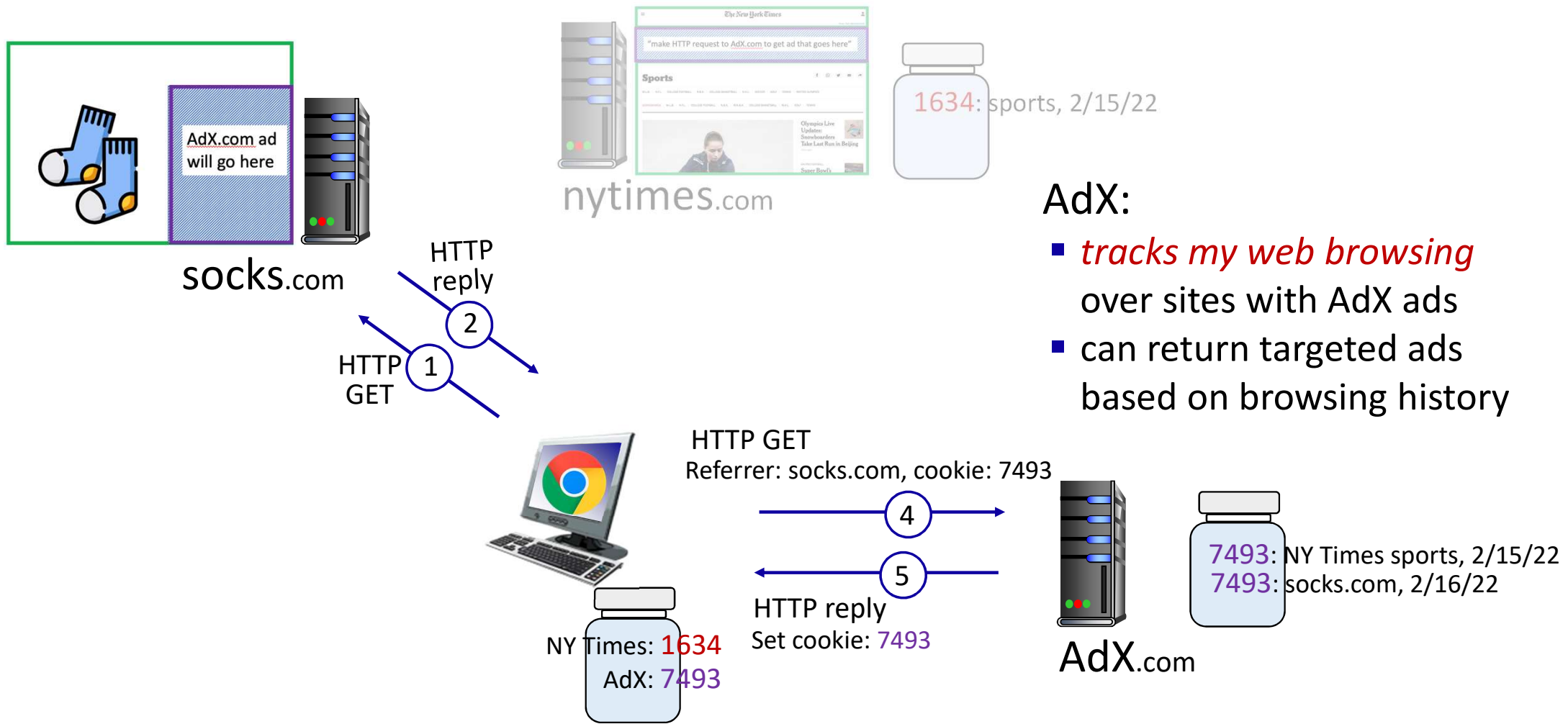
④ ⑤ fetch ad from AdX.com
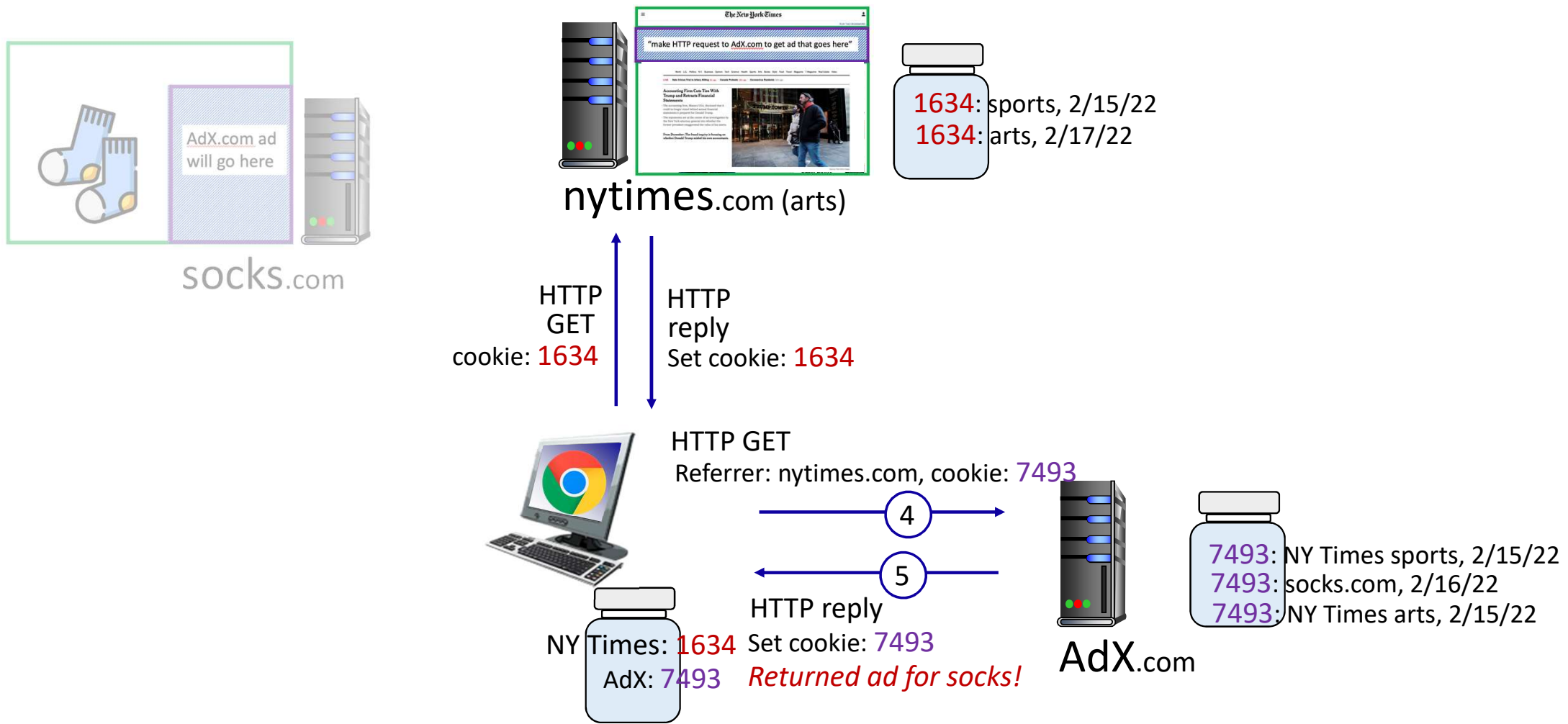
⑦ display composed page

nytimes.com

HTTP GET ① ② HTTP reply

NY times page with embedded ad displayed

AdX.com

Cookies: tracking a user's browsing behavior

nytimes.com (sports)

"make HTTP request to AdX.com to get ad that goes here"

Sports

1634: sports, 2/15/22

"first party" cookie – *from website you chose to visit (provides base html file)*

HTTP GET

HTTP reply
Set cookie: 1634

HTTP GET
Referrer: NY Times Sports

4

5

HTTP reply
Set cookie: 7493

7493: NY Times sports, 2/15/22

AdX.com

"third party" cookie – *from website you did not choose to visit*

NY Times: 1634
AdX: 7493

Cookies: tracking a user's browsing behavior

socks.com

The New York Times
"make HTTP request to AdX.com to get ad that goes here"
Sports
nytimes.com

1634: sports, 2/15/22

AdX:
- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

HTTP reply ②

HTTP GET ①

HTTP GET
Referrer: socks.com, cookie: 7493

④

⑤

HTTP reply
Set cookie: 7493

NY Times: 1634
AdX: 7493

AdX.com

7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22

Cookies: tracking a user's browsing behavior (one day later)

socks.com

AdX.com ad will go here

nytimes.com (arts)

"make HTTP request to AdX.com to get ad that goes here"

1634: sports, 2/15/22
1634: arts, 2/17/22

HTTP GET
cookie: 1634

HTTP reply
Set cookie: 1634

HTTP GET
Referrer: nytimes.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493
Returned ad for socks!

NY Times: 1634
AdX: 7493

AdX.com

7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22
7493: NY Times arts, 2/15/22

Cookies: tracking a user's browsing behavior

Cookies can be used to:

- track user behavior on a given website (first party cookies)
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
  - rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

third party tracking via cookies:

- disabled by default in Firefox, Safari browsers
- to be disabled in Chrome browser in 2023

GDPR (EU General Data Protection Regulation) and cookies

"Natural persons may be associated with online identifiers […] such as internet protocol addresses, cookie identifiers or other identifiers […].

This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them."

GDPR, recital 30 (May 2018)

when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations
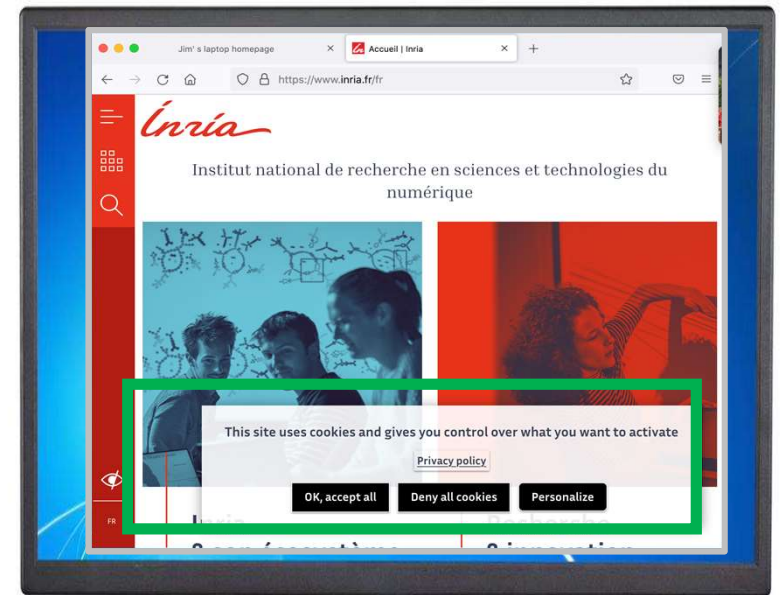
User has explicit control over whether or not cookies are allowed

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*

- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client
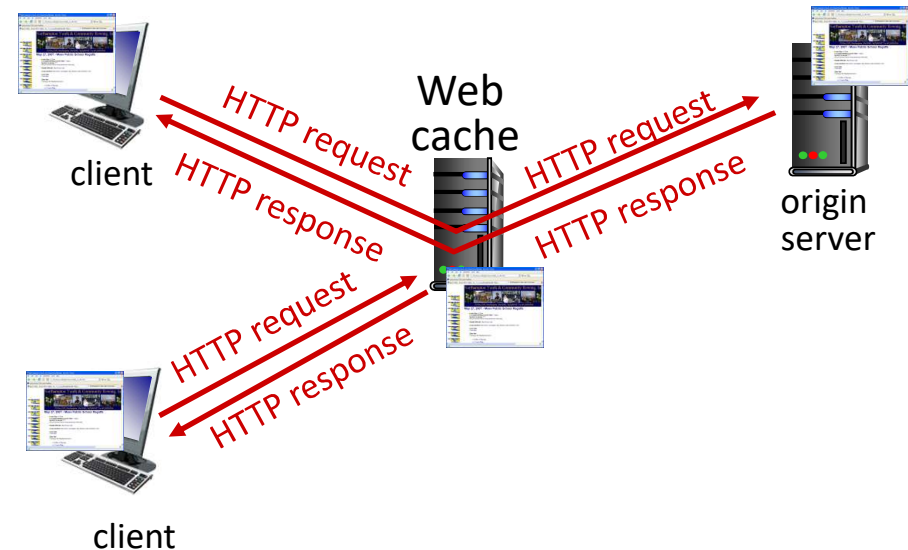
Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content
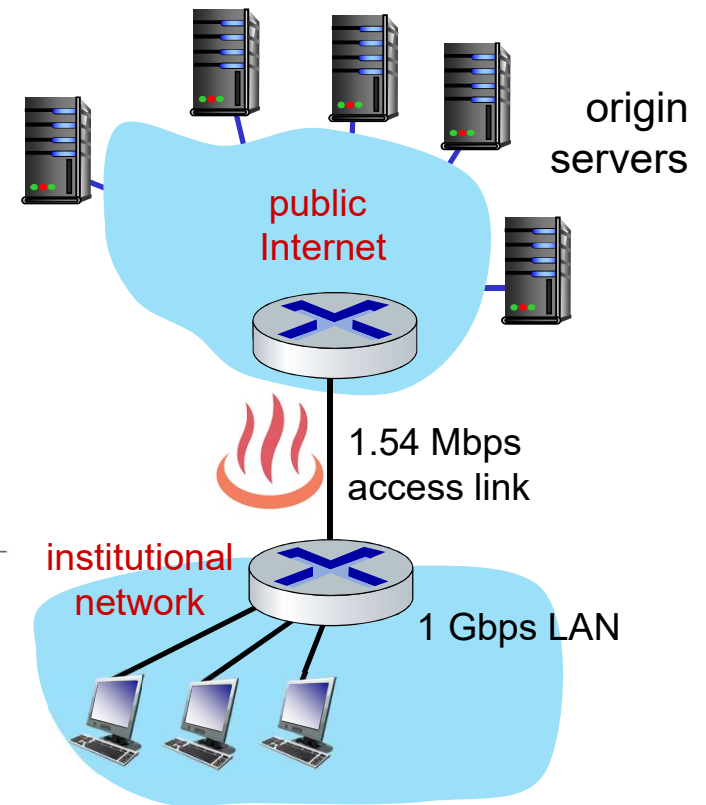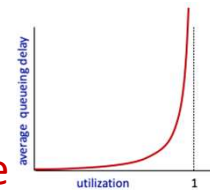
Caching example

Scenario:

- access link rate: 1.54 Mbps
- RTT from internet-side router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97   *problem:* large queueing delays at high utilization!
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay

  = 2 sec + minutes + usecs
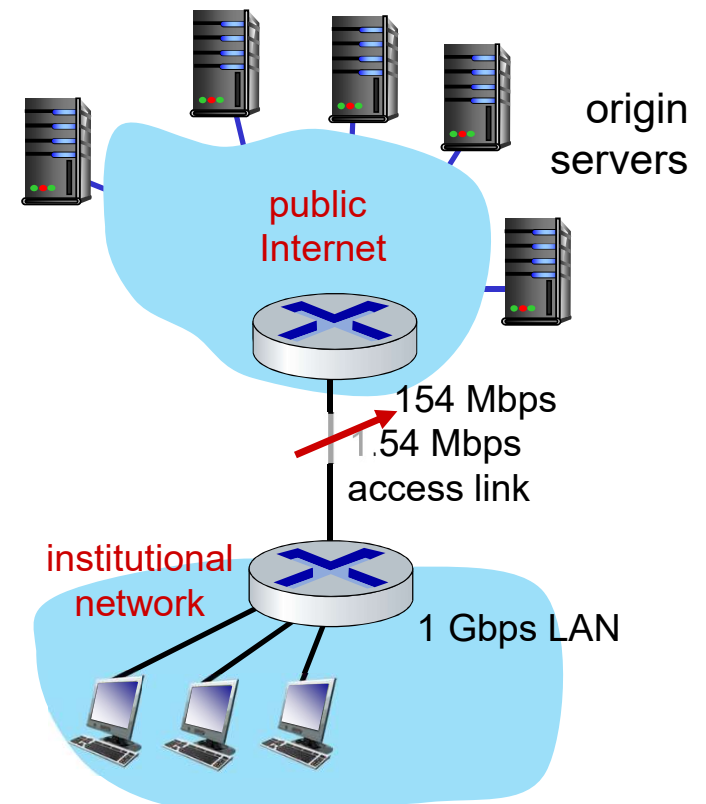
Option 1: buy a faster access link

Scenario:

- access link rate: ~~1.54~~ **154 Mbps**
- RTT from internet-side router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ → .0097
- LAN utilization: .0015
- end-end delay = Internet delay +
  access link delay + LAN delay

  = 2 sec + ~~minutes~~ + usecs → msecs

Cost: faster access link (expensive!)

origin servers

public Internet

154 Mbps
~~1.54 Mbps~~
access link

institutional network

1 Gbps LAN

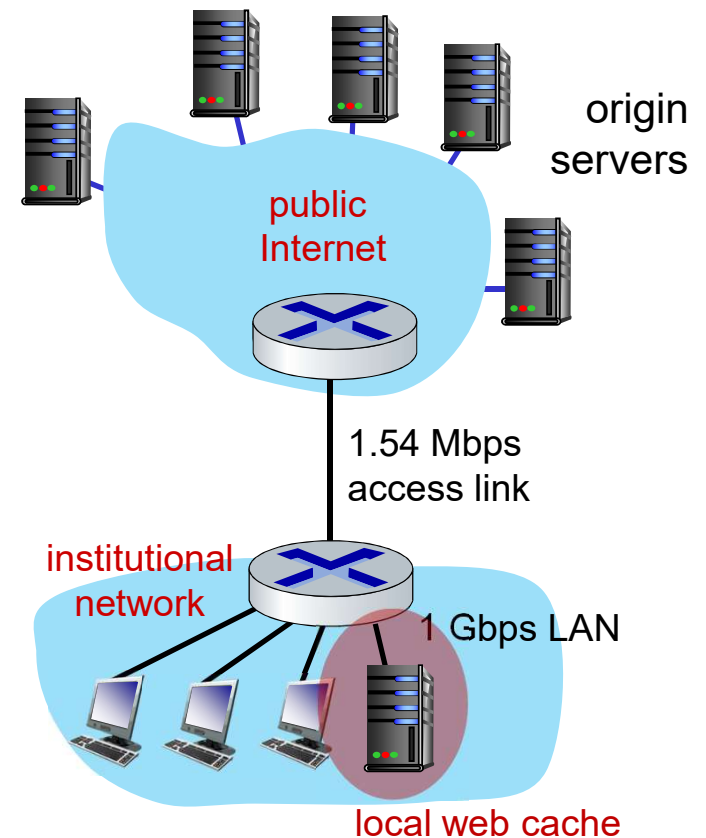# Option 2: install a web cache

Scenario:
- access link rate: 1.54 Mbps
- RTT from internet-side router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

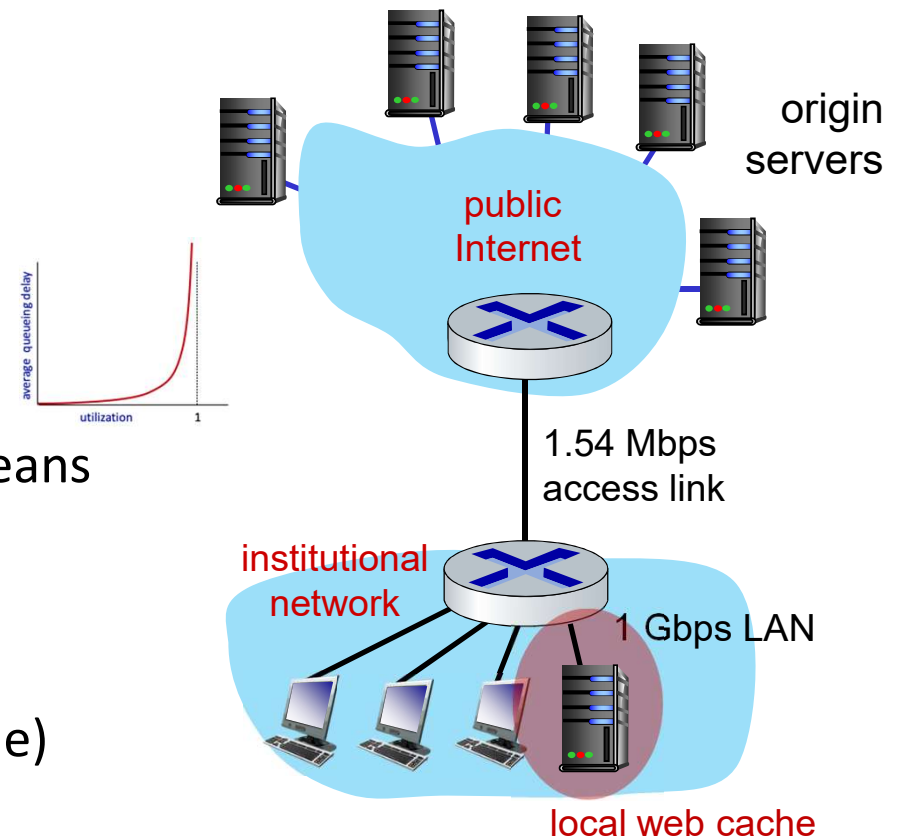*Cost:* web cache (cheap!)

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

How to compute link utilization, delay?

origin servers

public Internet

1.54 Mbps access link

institutional network

1 Gbps LAN

local web cache

Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link
    = 0.6 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- average end-end delay:
  = 0.6 * (delay from origin servers)
      + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

lower average end-end delay than with 154 Mbps link (and cheaper too!)

origin servers

public Internet

1.54 Mbps access link

institutional network

1 Gbps LAN

local web cache

average queueing delay / utilization

Browser caching: Conditional GET

Goal: don't send object if browser has up-to-date cached version
- no object transmission delay (or use of network resources)

■ *client:* specify date of browser-cached copy in HTTP request

  If-modified-since: <date>

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

  HTTP/1.0 304 Not Modified

client   server

HTTP request msg
If-modified-since: <date>

object not modified after <date>

HTTP response
**HTTP/1.0
304 Not Modified**

- -

HTTP request msg
If-modified-since: <date>

object modified after <date>

HTTP response
**HTTP/1.0 200 OK
<data>**
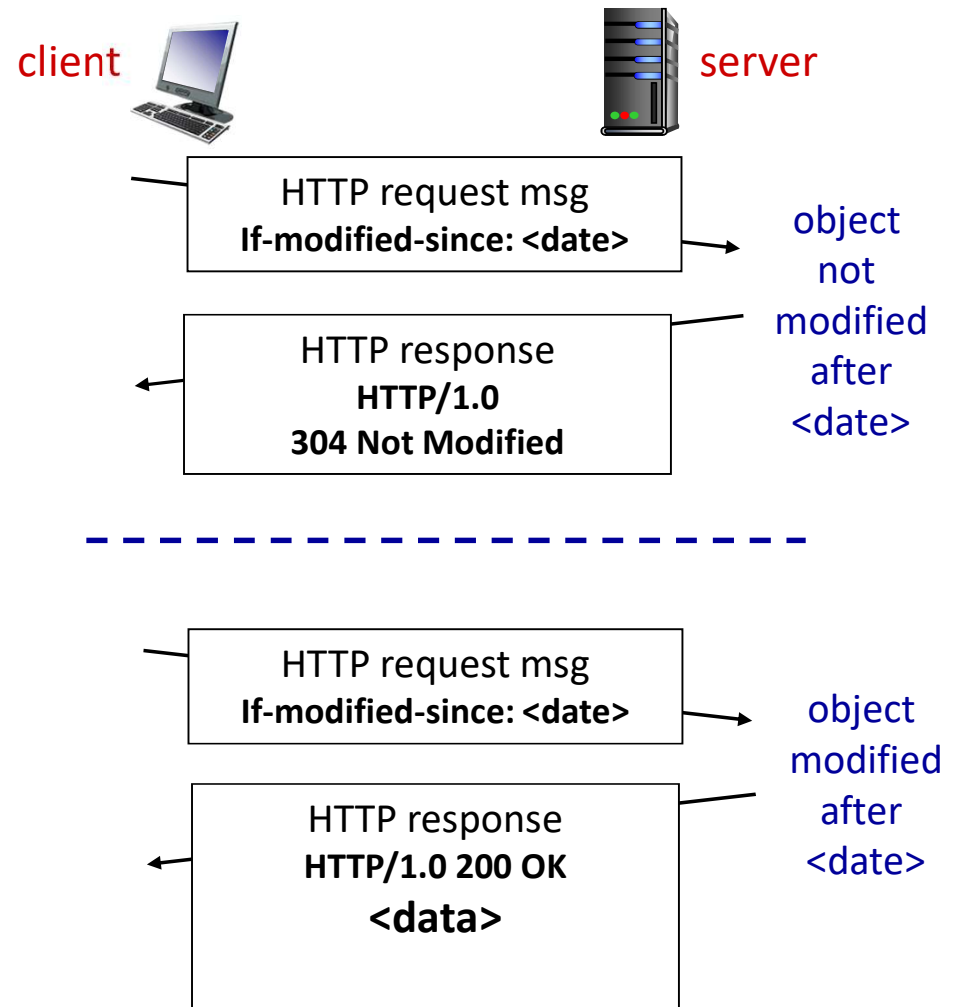
HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1

- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)

- *push* unrequested objects to client

- divide objects into frames, schedule frames to mitigate HOL blocking
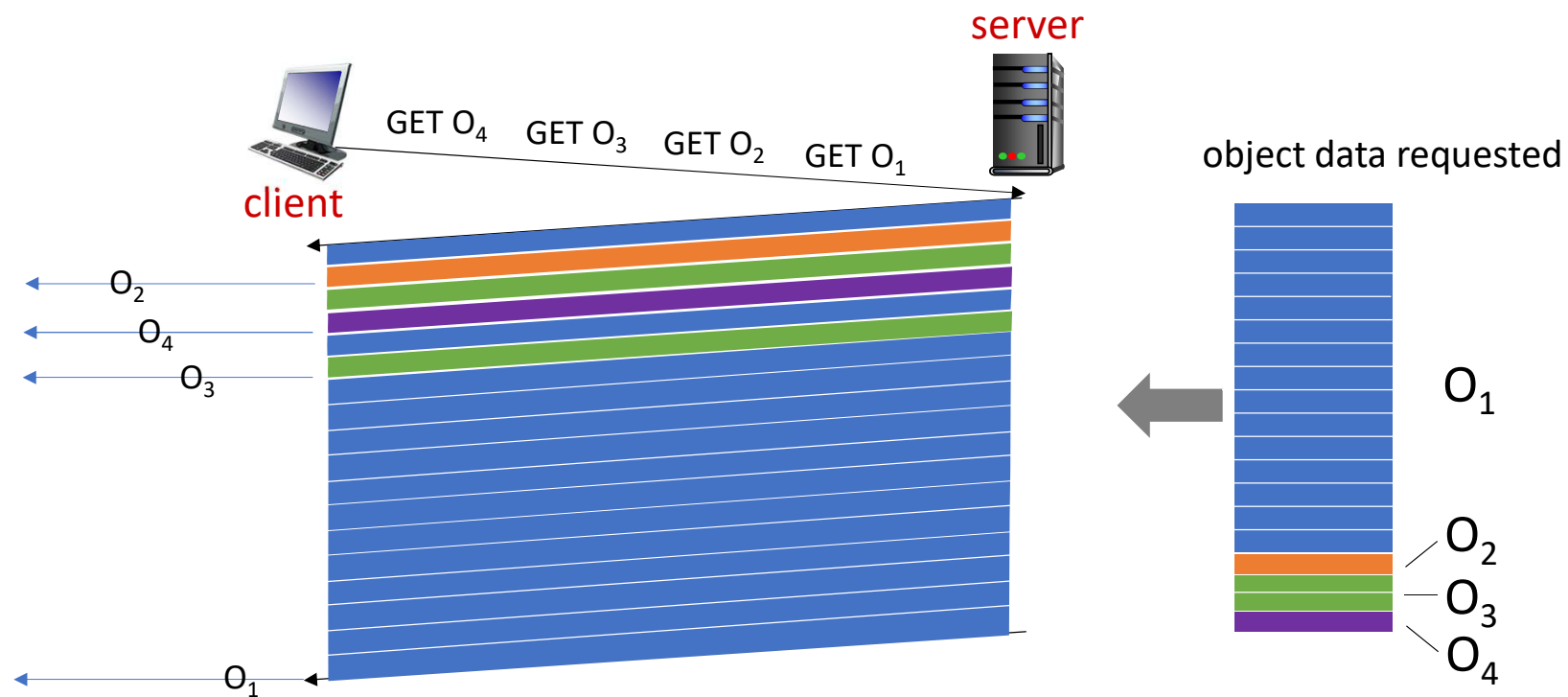
HTTP/2: mitigating HOL blocking

HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects
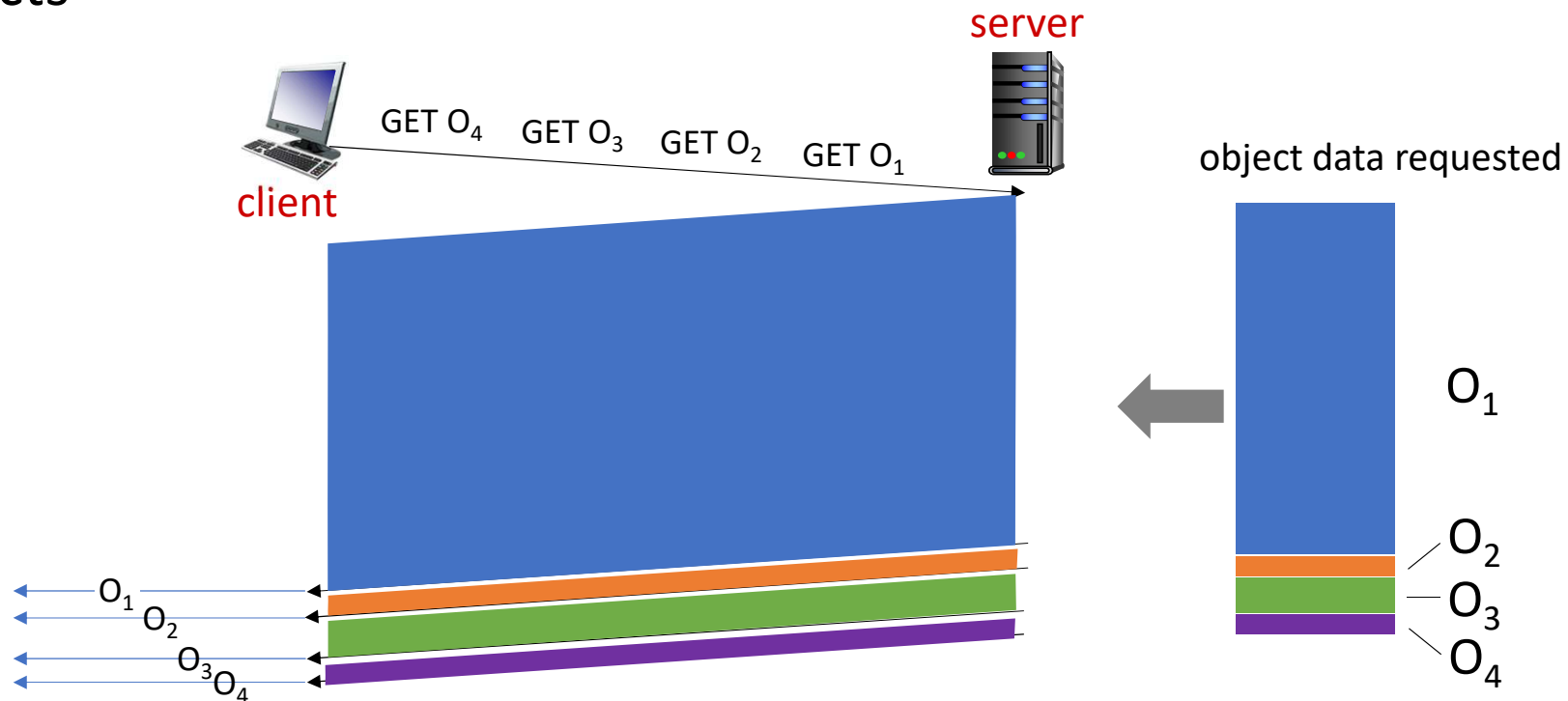
objects delivered in order requested: O₂, O₃, O₄ wait behind O₁

HTTP/2: mitigating HOL blocking

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

O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

Application layer: overview

- Principles of network applications

- Web and HTTP

- **E-mail, SMTP, IMAP**

- The Domain Name System DNS

- P2P applications

- video streaming and content distribution networks

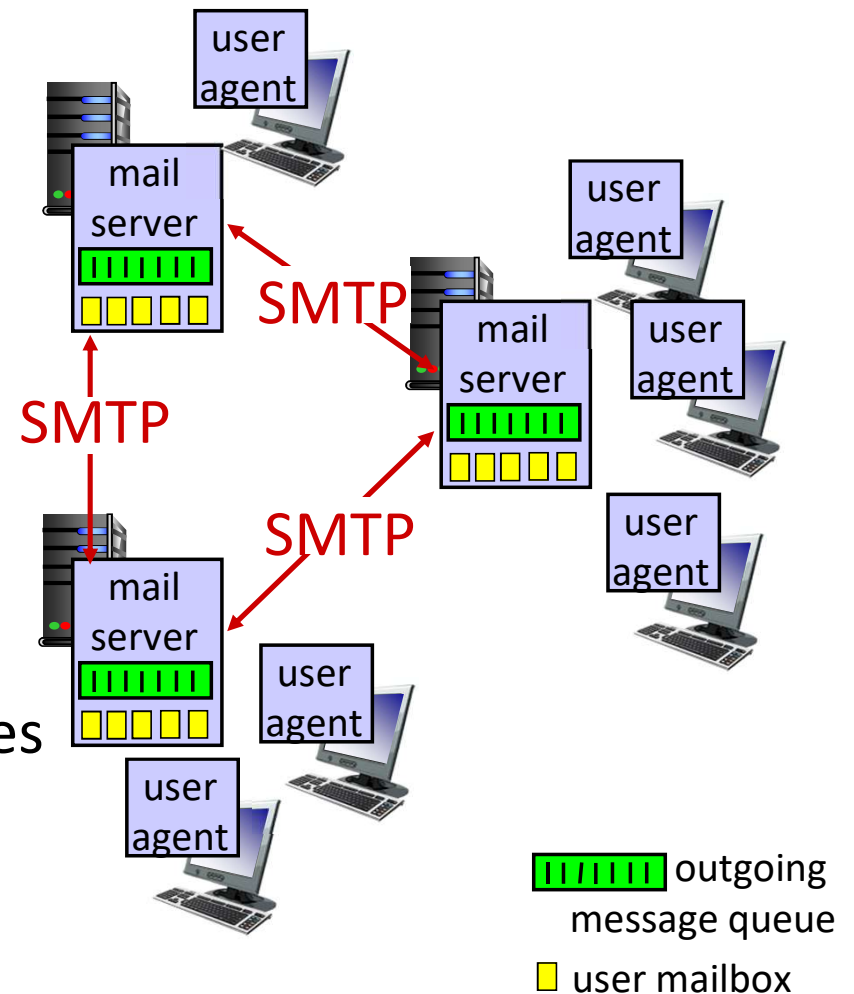- socket programming with UDP and TCP

E-mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
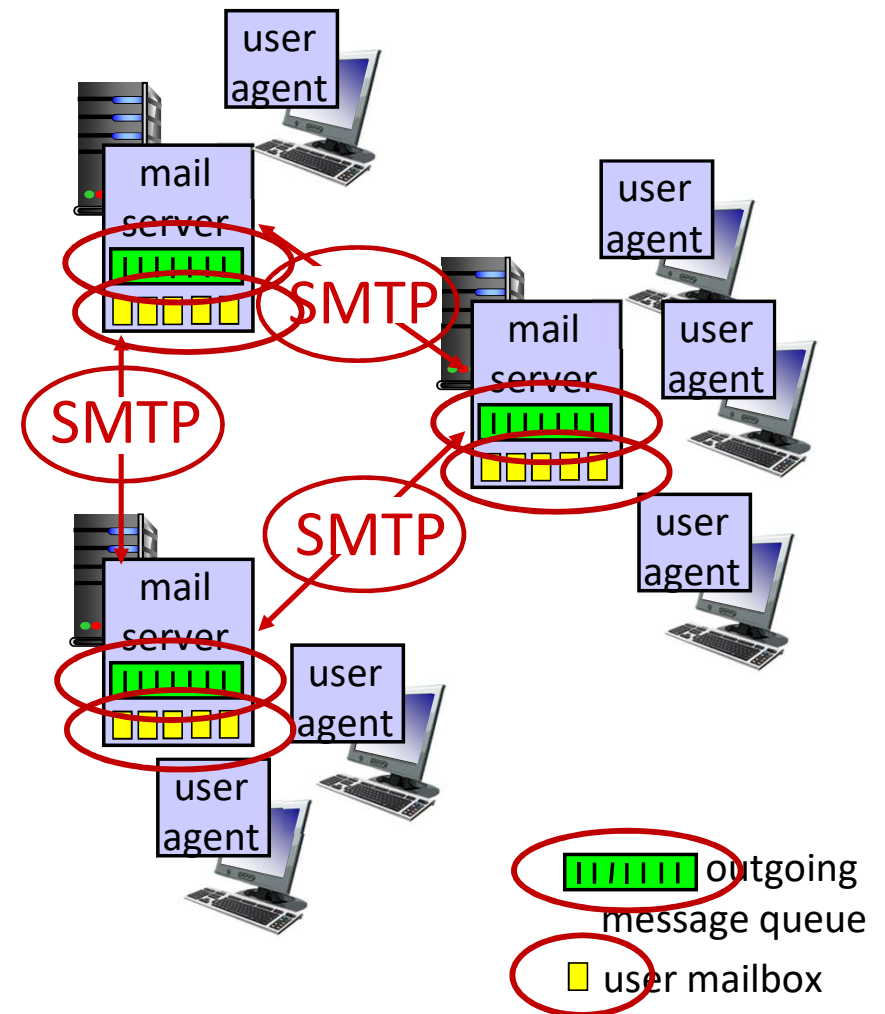- outgoing, incoming messages stored on server

outgoing message queue

user mailbox

E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
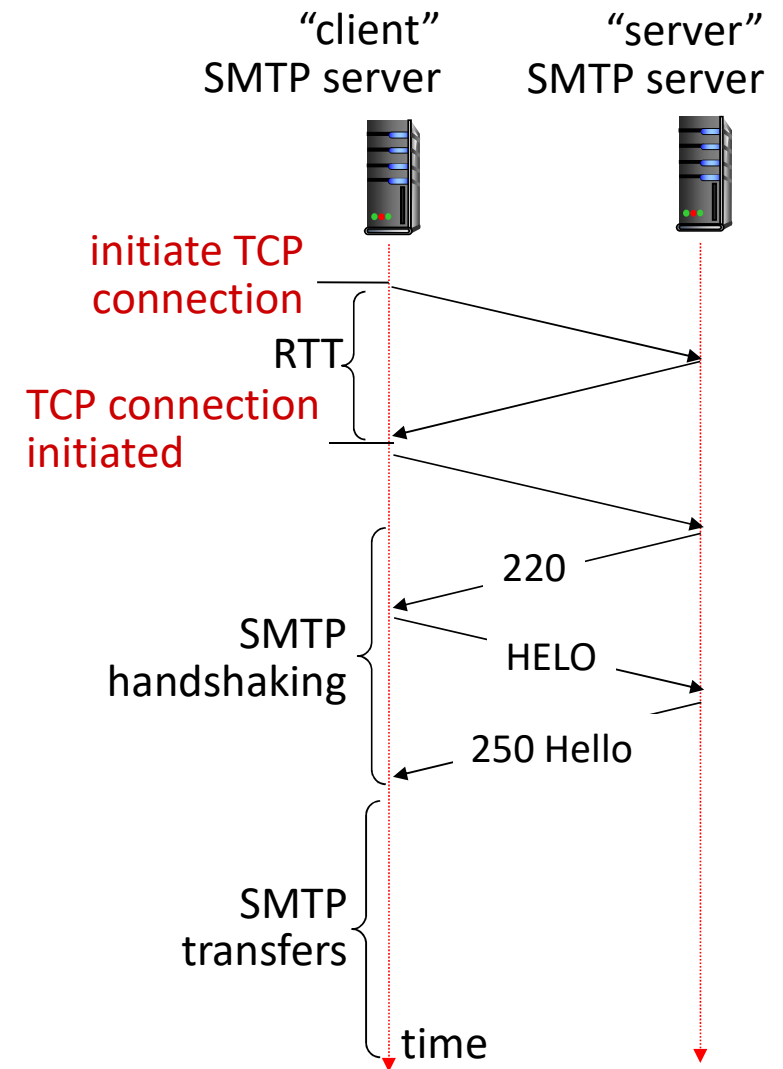
- *message queue* of outgoing (to be sent) mail messages

SMTP protocol between mail servers to send email messages

- client: sending mail server
- "server": receiving mail server

SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase

"client"
SMTP server

"server"
SMTP server

initiate TCP connection

RTT

TCP connection initiated

220

SMTP handshaking

HELO

250 Hello

SMTP transfers

time

Scenario: Alice sends e-mail to Bob

1) Alice uses UA to compose e-mail message "to" bob@someschool.edu

2) Alice's UA sends message to her mail server using SMTP; message placed in message queue

3) client side of SMTP at mail server opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message

Alice's mail server   Bob's mail server
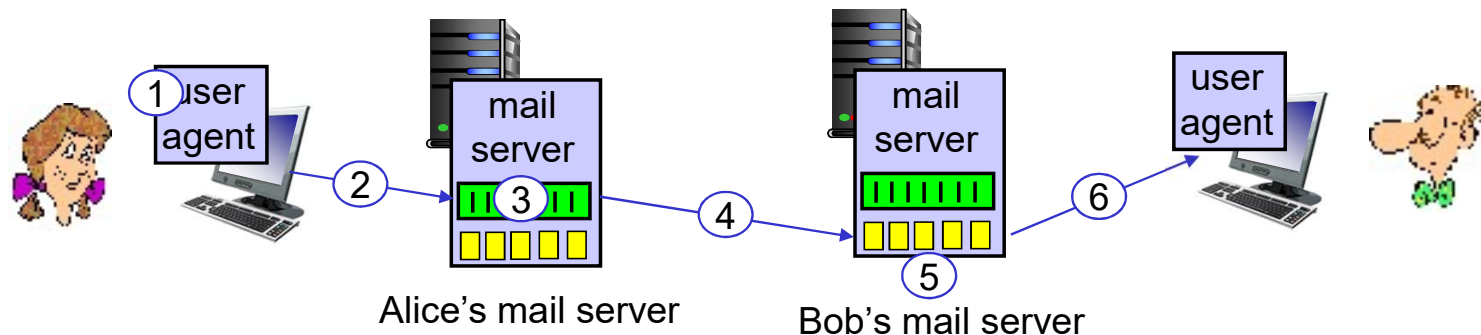
Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

SMTP: observations

*comparison with HTTP:*

- HTTP: client pull

- SMTP: client push

- both have ASCII command/response interaction, status codes

- HTTP: each object encapsulated in its own response message

- SMTP: multiple objects sent in multipart message

- SMTP uses persistent connections

- SMTP requires message (header & body) to be in 7-bit ASCII

- SMTP server uses CRLF.CRLF to determine end of message

Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

- header lines, e.g.,
  - `To: alice@crepes.fr`
  - `From: bob@hamberger.edu`
  - `Subject: Searching for the meaning..`

- Body: the "message" , ASCII characters only

header

blank line

body
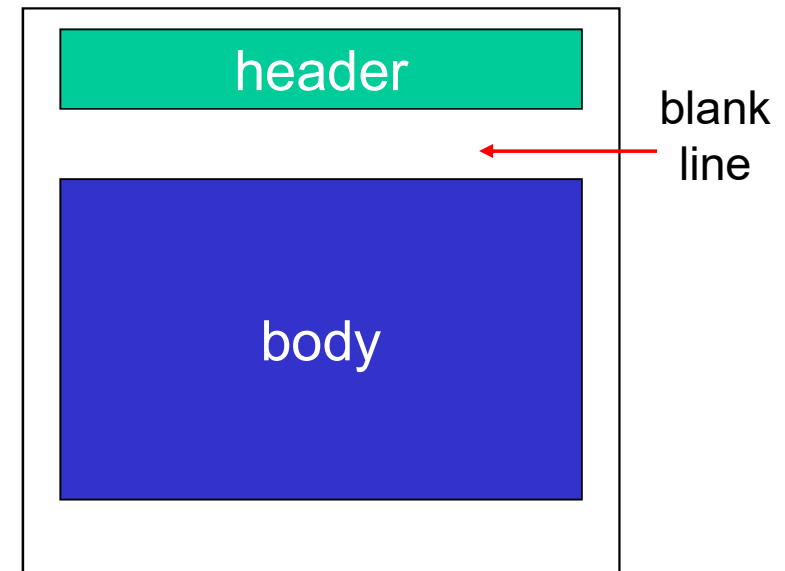
Retrieving email: mail access protocols
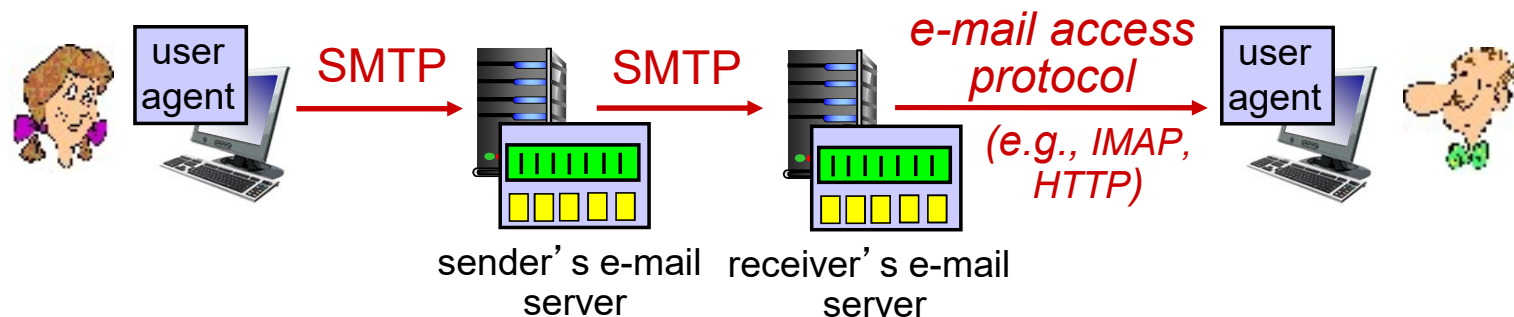
- SMTP: delivery/storage of e-mail messages to receiver's server

- mail access protocol: retrieval from server

  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server

  - HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP) to retrieve e-mail messages

Application Layer: Overview

- Principles of network applications

- Web and HTTP

- E-mail, SMTP, IMAP

- **The Domain Name System DNS**

- P2P applications

- video streaming and content distribution networks

- socket programming with UDP and TCP

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - used by humans

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

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *name servers*

- *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)

  - *note:* core Internet function, implemented as application-layer protocol

  - complexity at network's "edge"

DNS: services, structure

DNS services:

- hostname-to-IP-address translation

- host aliasing
  - canonical, alias names
    - enterprise.com → relay1.west-coast.enterprise.com

- mail server aliasing
  - hotmail.com → relay1.west-coast.hotmail.com

- load distribution
  - replicated Web servers: many IP addresses correspond to one name
    - cnn.com → IP1, IP2, IP3, …

Q: Why not centralize DNS?

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

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

humongous distributed database:
- ~ billion records, each simple
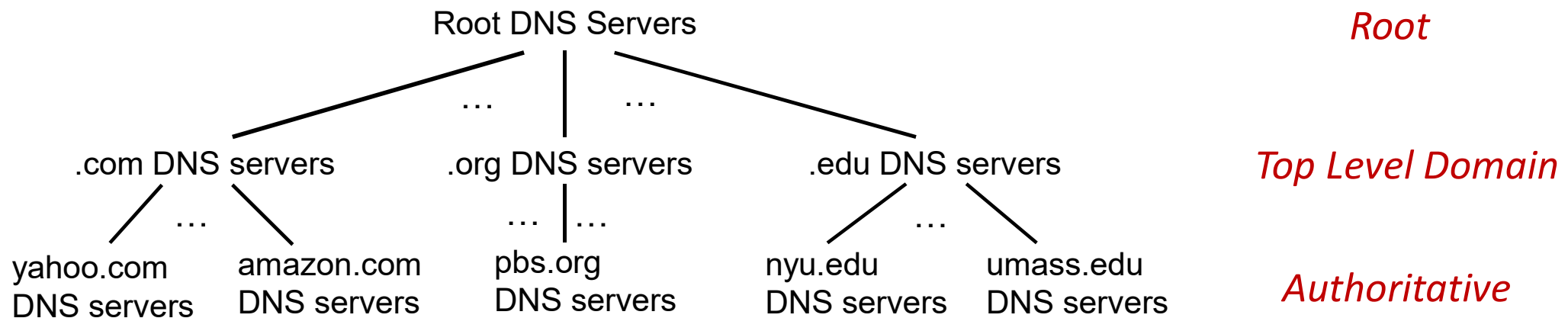
handles many *trillions* of queries/day:
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:
- millions of different organizations responsible for their records

NOT SO EASY

DNS: a distributed, hierarchical database

Root DNS Servers

Root

.com DNS servers   .org DNS servers   .edu DNS servers

Top Level Domain

yahoo.com DNS servers   amazon.com DNS servers   pbs.org DNS servers   nyu.edu DNS servers   umass.edu DNS servers
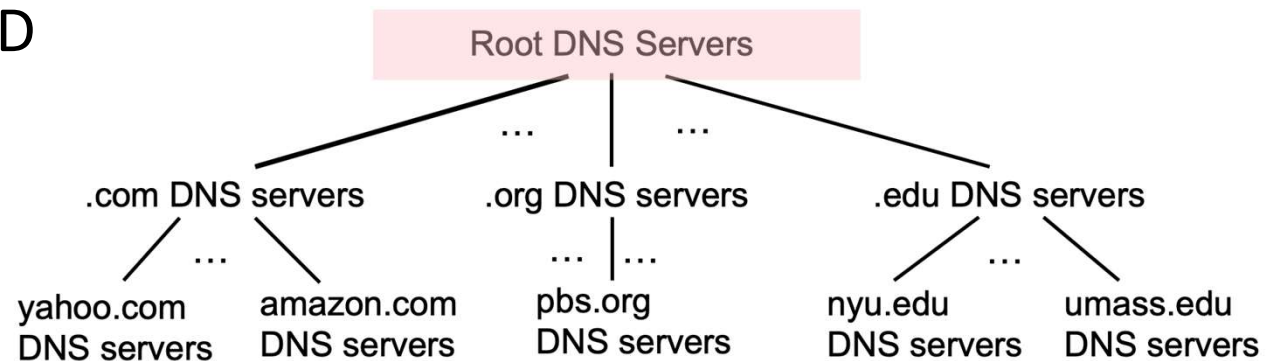
Authoritative

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: root name servers

- provide IP addresses of TLD servers

DNS: root name servers

- *incredibly important* Internet function
  - Internet couldn't function without it!
  - DNSSEC – provides security (authentication, message integrity)

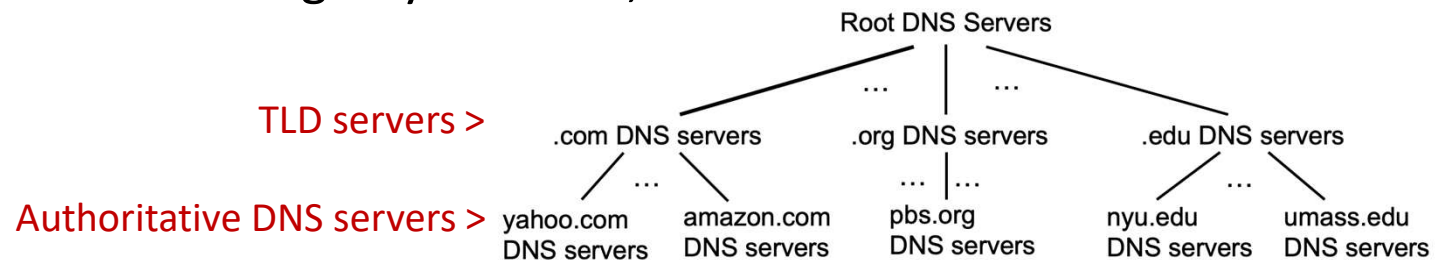- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)

Key:
- 0 Servers
- 1–10 Servers
- 11–20 Servers
- 21+ Servers
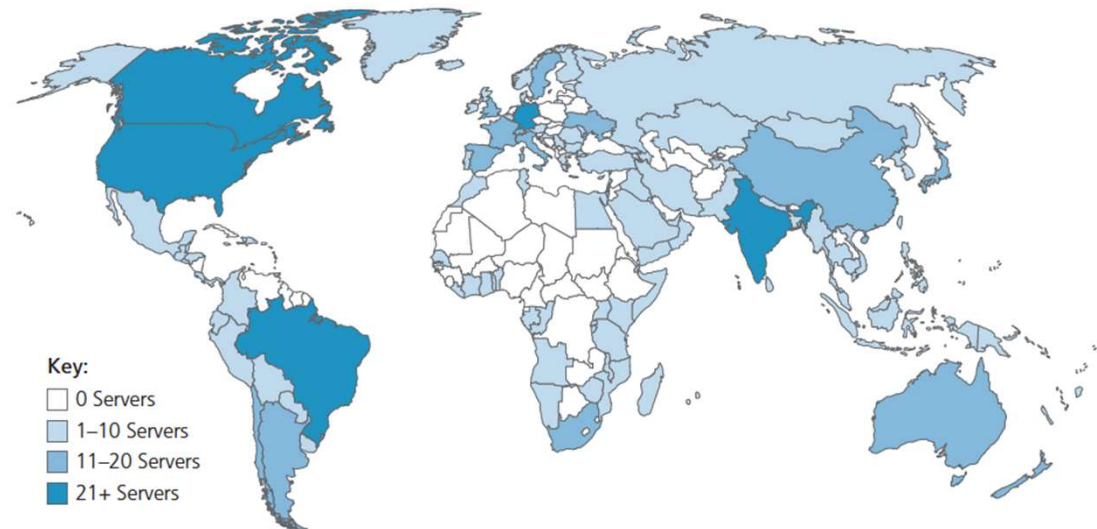
Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

- provide IP addresses for authoritative DNS server
- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD

blog.naver.com

Sub → Second-level → Top-level

Root DNS Servers

TLD servers >   .com DNS servers   ...   ...   .org DNS servers   .edu DNS servers

Authoritative DNS servers >   yahoo.com DNS servers   amazon.com DNS servers   pbs.org DNS servers   nyu.edu DNS servers   umass.edu DNS servers

Authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name servers

- when host makes DNS query, it is sent to its *local* DNS server
  - Local DNS server returns reply, answering:
    - from its local cache of recent name-to-address translation pairs (possibly out of date!)
    - forwarding request into DNS hierarchy for resolution
  - each ISP has local DNS name server; to find yours:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`

- local DNS server doesn't strictly belong to hierarchy

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"

root DNS server

2

3

TLD DNS server

1

4

8

5

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu

7

6

authoritative DNS server
dns.umass.edu
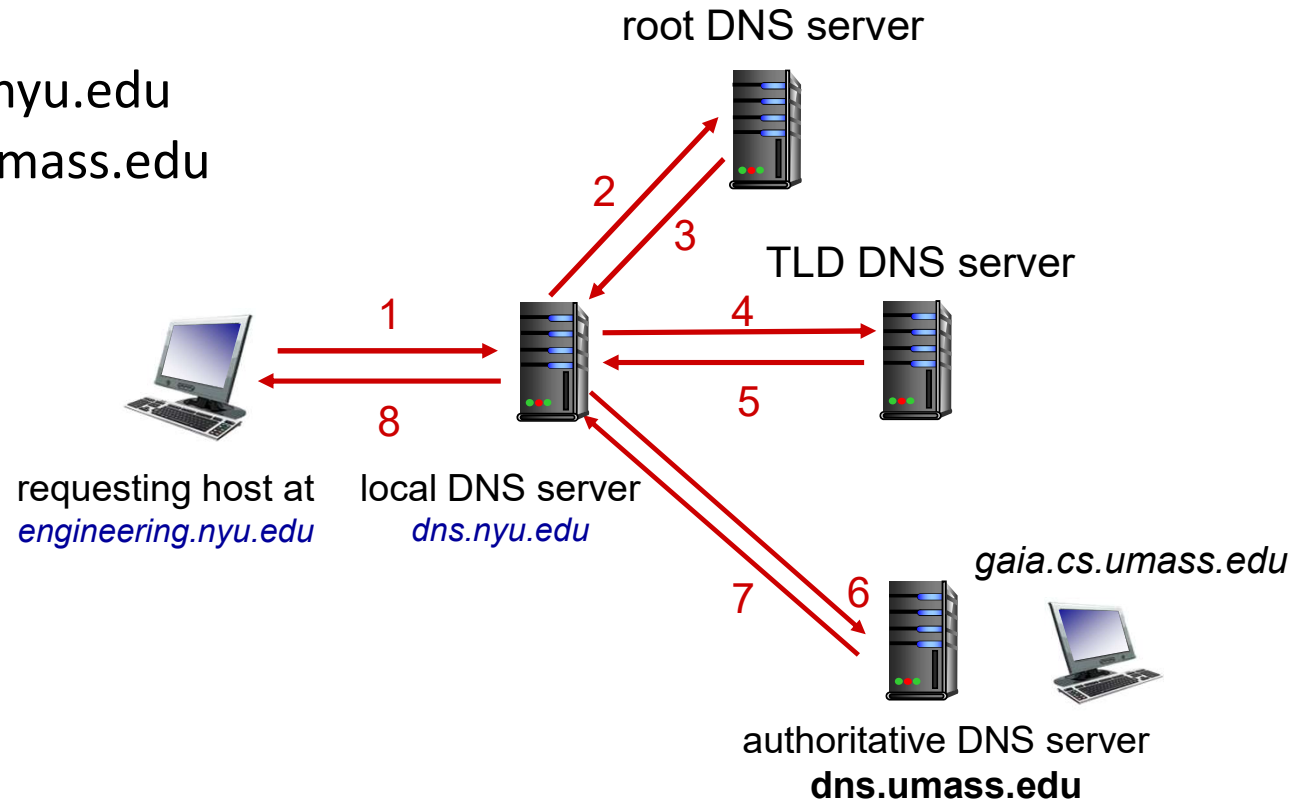
DNS name resolution: recursive query
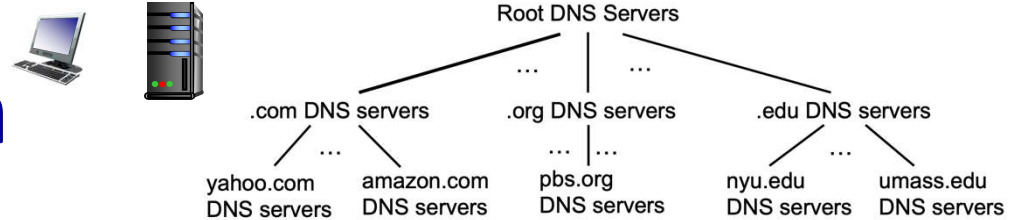
Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

root DNS server

TLD DNS server

local DNS server
dns.nyu.edu

requesting host at
engineering.nyu.edu

authoritative DNS server
dns.umass.edu

gaia.cs.umass.edu

1 2 3 4 5 6 7 8
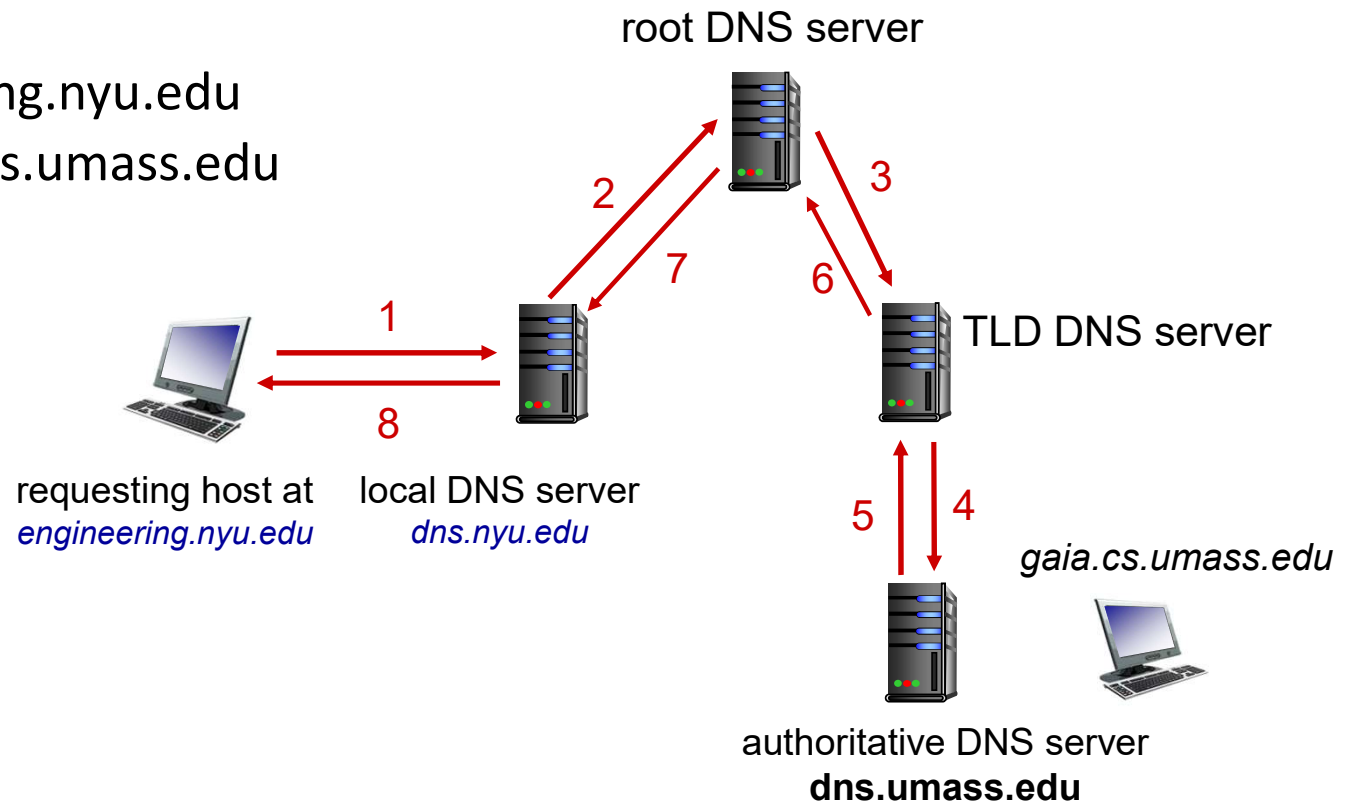
Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and i*mmediately* 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

- cached entries may be *out-of-date*
  - if named host changes IP address, may not be known Internet-wide until all TTLs expire!

DNS records

DNS: distributed database storing resource records (RR)

RR format: (`name`, `value`, `type`, `ttl`)

type=A
- `name` is hostname
- `value` is IP address

type=NS
- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME
- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

type=MX
- `value` is name of SMTP mail server associated with `name`

DNS records

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu

root DNS server

TLD DNS server:
- (umass.edu, dns.umass.edu, NS)
- (dns.umass.edu, 128.119.40.111, A)

TLD DNS server

Authoritative server:
- (gaia.cs.umass.edu, 128.119.245.12, A)

1
2
3
4
5
6
7
8

requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu

gaia.cs.umass.edu

Authoritative DNS server
dns.umass.edu
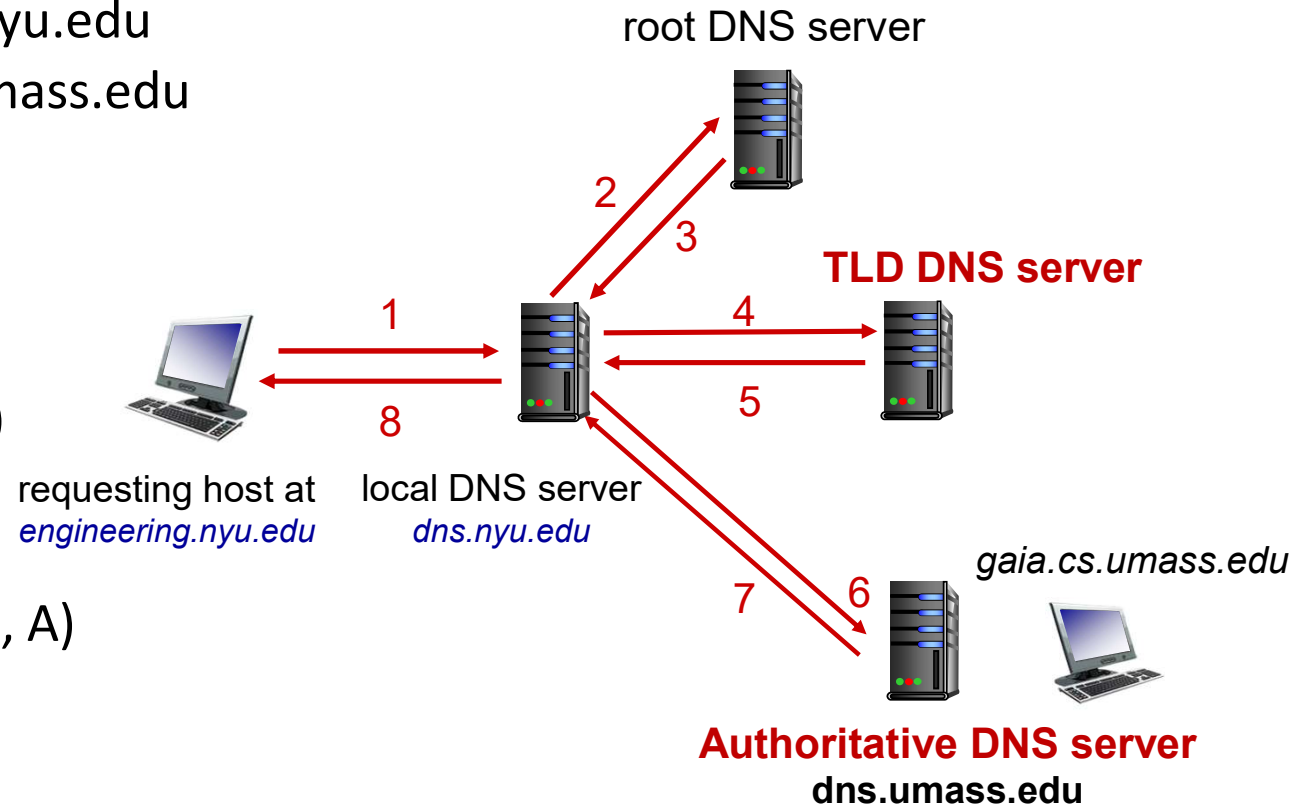
DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

message header:
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

← 2 bytes →	← 2 bytes →
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	
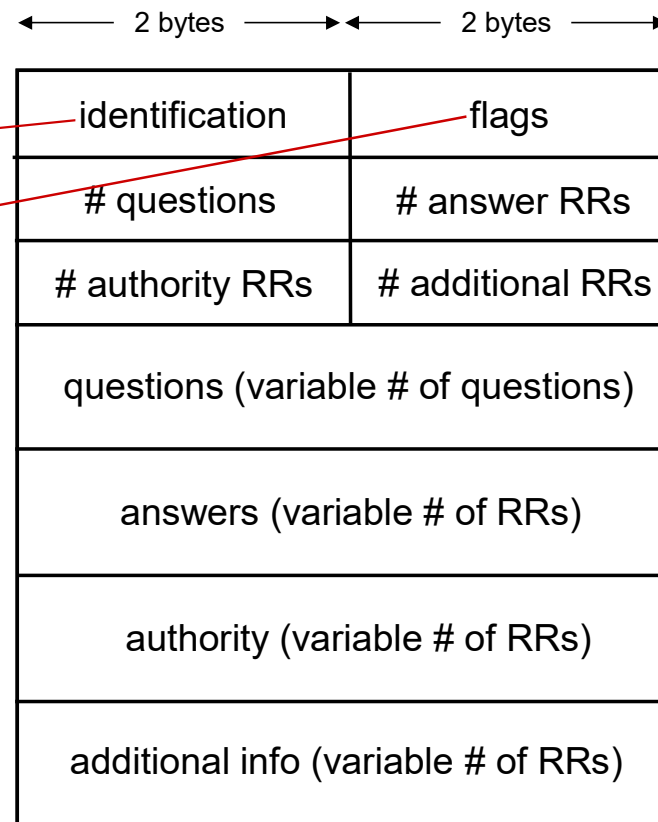
DNS protocol messages

DNS *query* and *reply* messages, both have same *format:*

← 2 bytes →	← 2 bytes →
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query —— questions (variable # of questions)

RRs in response to query —— answers (variable # of RRs)

records for authoritative servers —— authority (variable # of RRs)

additional " helpful" info that may be used —— additional info (variable # of RRs)
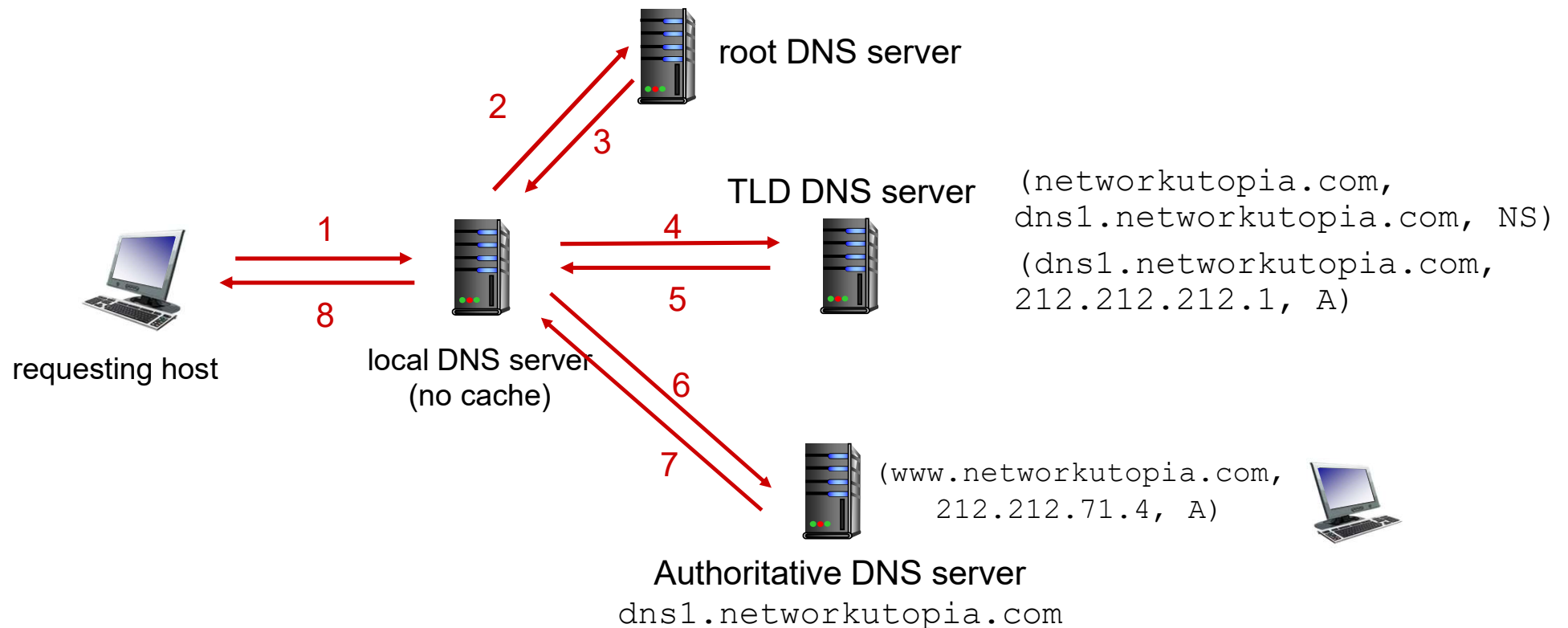
Getting your info into the DNS

example: new startup "Network Utopia"

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS, A RRs into .com TLD server:
    ```
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
    ```
- create authoritative server locally with IP address `212.212.212.1`
  - type A record for www.networkuptopia.com
  - type MX record for mail.networkutopia.com

Receiving records

Example: Alice wants to see *www.networkutopia.com*

root DNS server

2

3

TLD DNS server   `(networkutopia.com, dns1.networkutopia.com, NS)`

4

1

5   `(dns1.networkutopia.com, 212.212.212.1, A)`

8

requesting host

local DNS server
(no cache)

6

7   `(www.networkutopia.com, 212.212.71.4, A)`

Authoritative DNS server
`dns1.networkutopia.com`

Problems

- Assume the following records are on the TLD DNS server:
  - (www.enterprise.com, dns.enterprise.com, NS)
  - (dns.enterprise.com, 146.54.136.45, A)

- Assume the following records are on the enterprise.com DNS server:
  - (www.enterprise.com, east1.enterprise.com, CNAME)
  - (east1.enterprise.com, 142.81.17.206, A)
  - (enterprise.com, mail.enterprise.com, MX)
  - (mail.enterprise.com, 247.29.128.97, A)

Root DNS server

Local DNS server

TLD DNS server

Authoritative DNS server
dns.enterprise.com

Requesting host
127.0.0.1

www.enterprise.com

1. When local DNS server contacts TLD server, how many answers (RR) are returned?
   a. What is the content of A-type record? "name, value"

2. What is the actual name of enterprise.com? What type of record is needed for this?

3. What type of record will contain the name of mailserver?