# Computer Networks CS3001 (Section BDS-7A) Lecture 09

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science

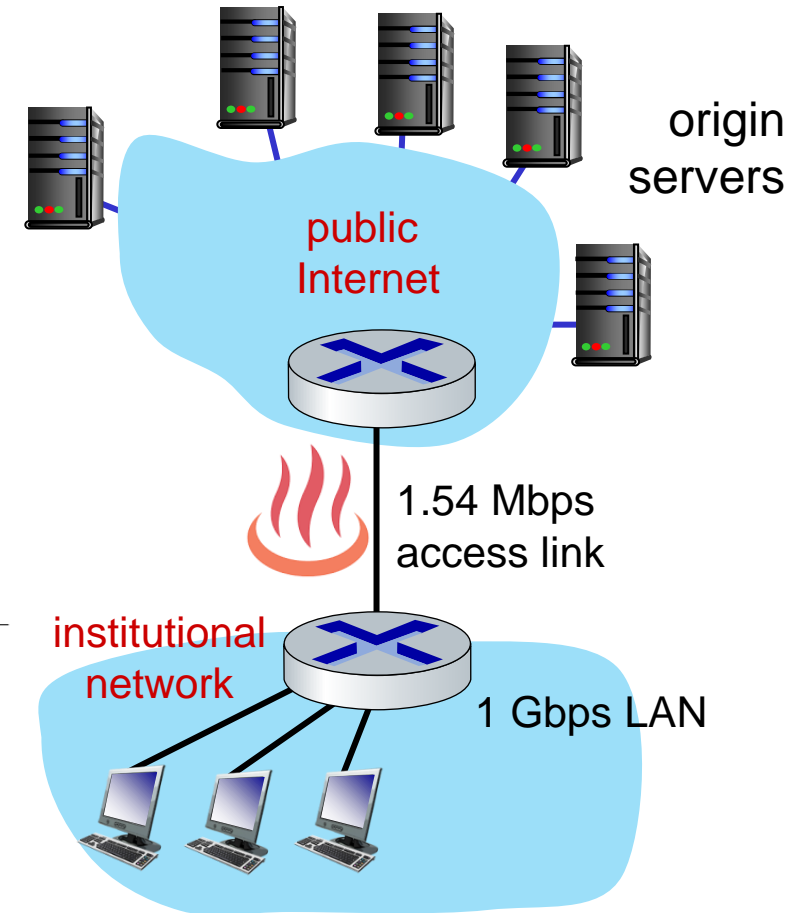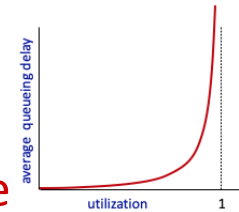21 ~~19~~ September, 2023

# Caching example

*Scenario:*

- access link rate: 1.54 Mbps
- RTT from institutional 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

origin servers

public Internet

1.54 Mbps access link

institutional network

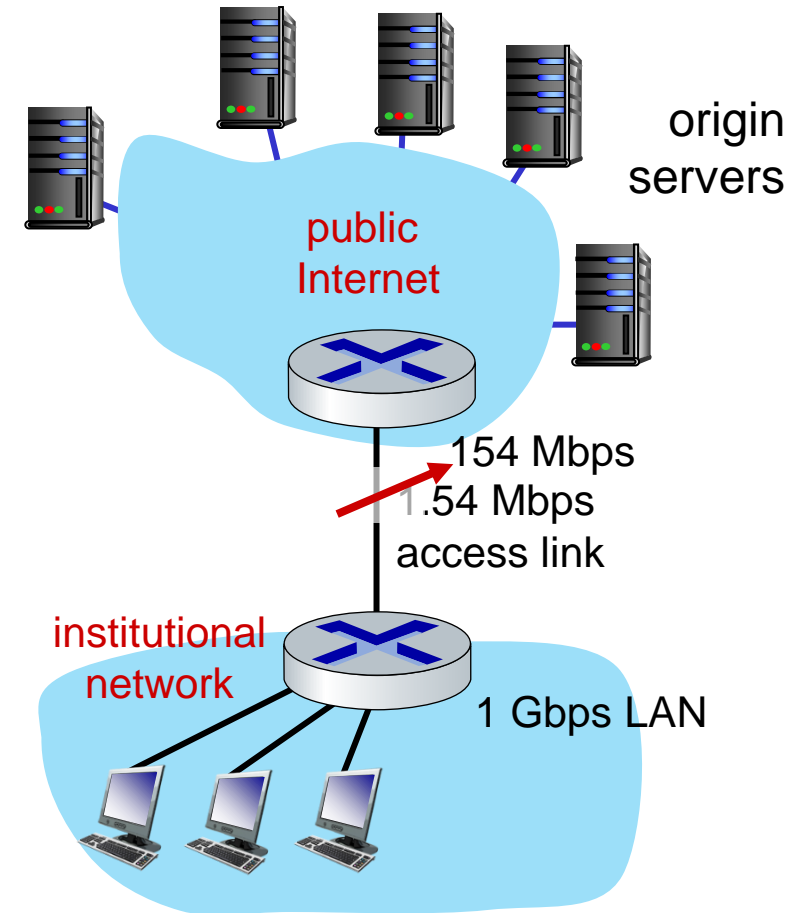1 Gbps LAN

# Option 1: buy a faster access link

*Scenario:*

- access link rate: ~~1.54~~ **154 Mbps** Mbps
- RTT from institutional 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 institutional 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!)

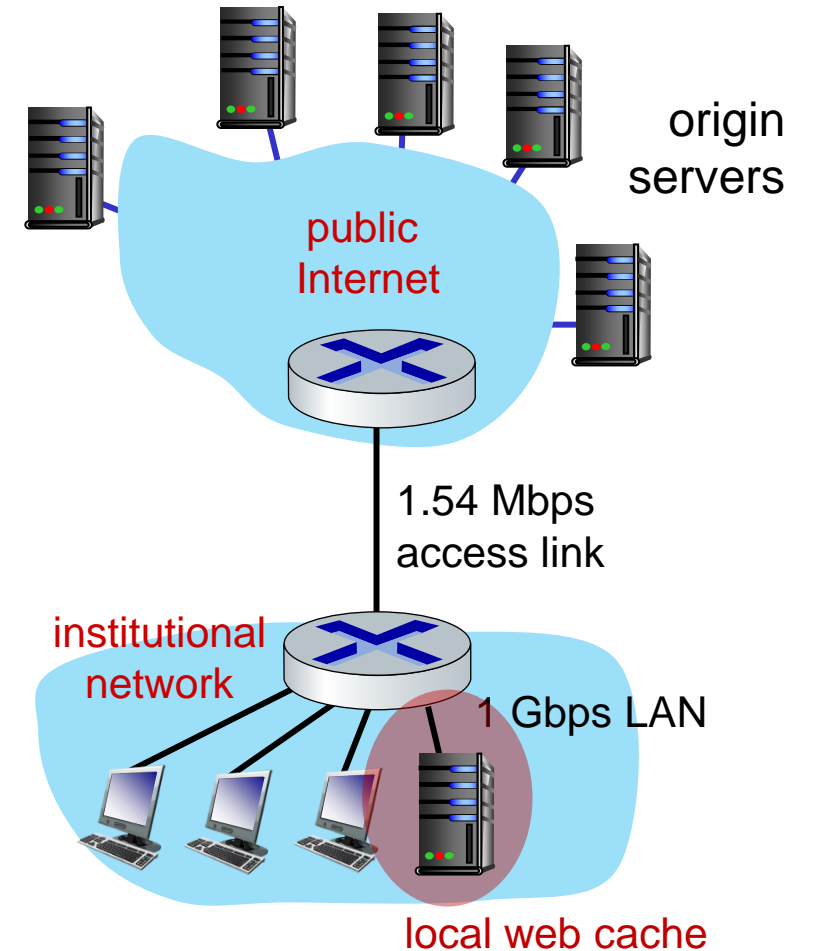*Performance:*

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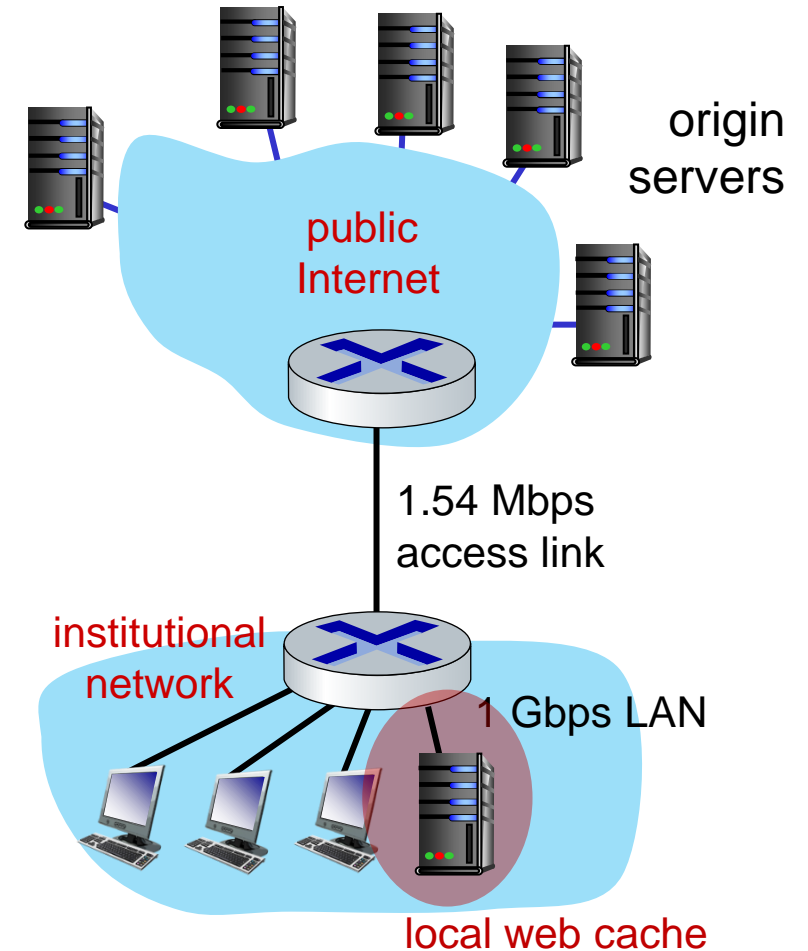
# Calculating access link utilization, end-end delay with cache:

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

# 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
before
<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>**
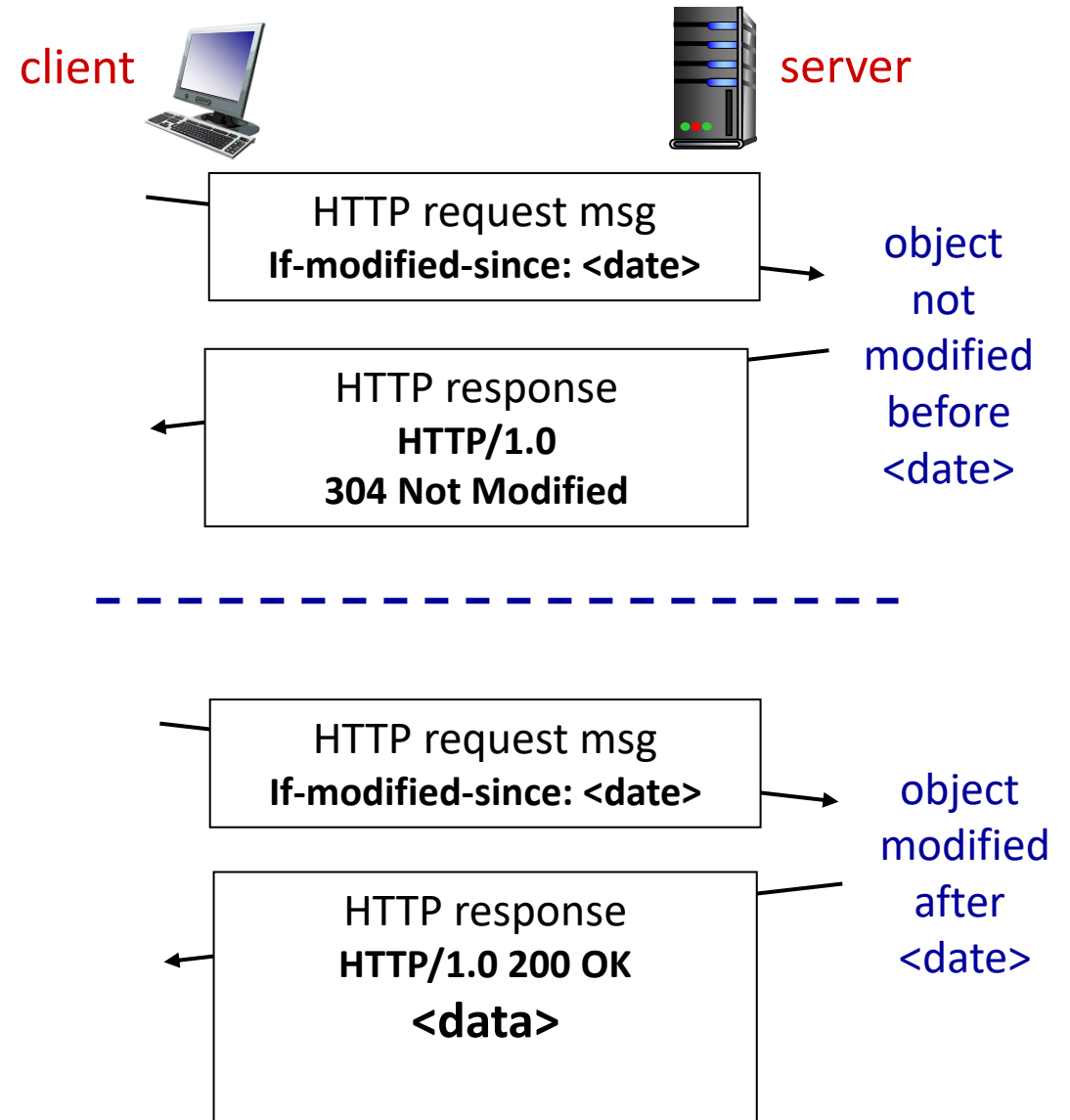
# HTTP/2

*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
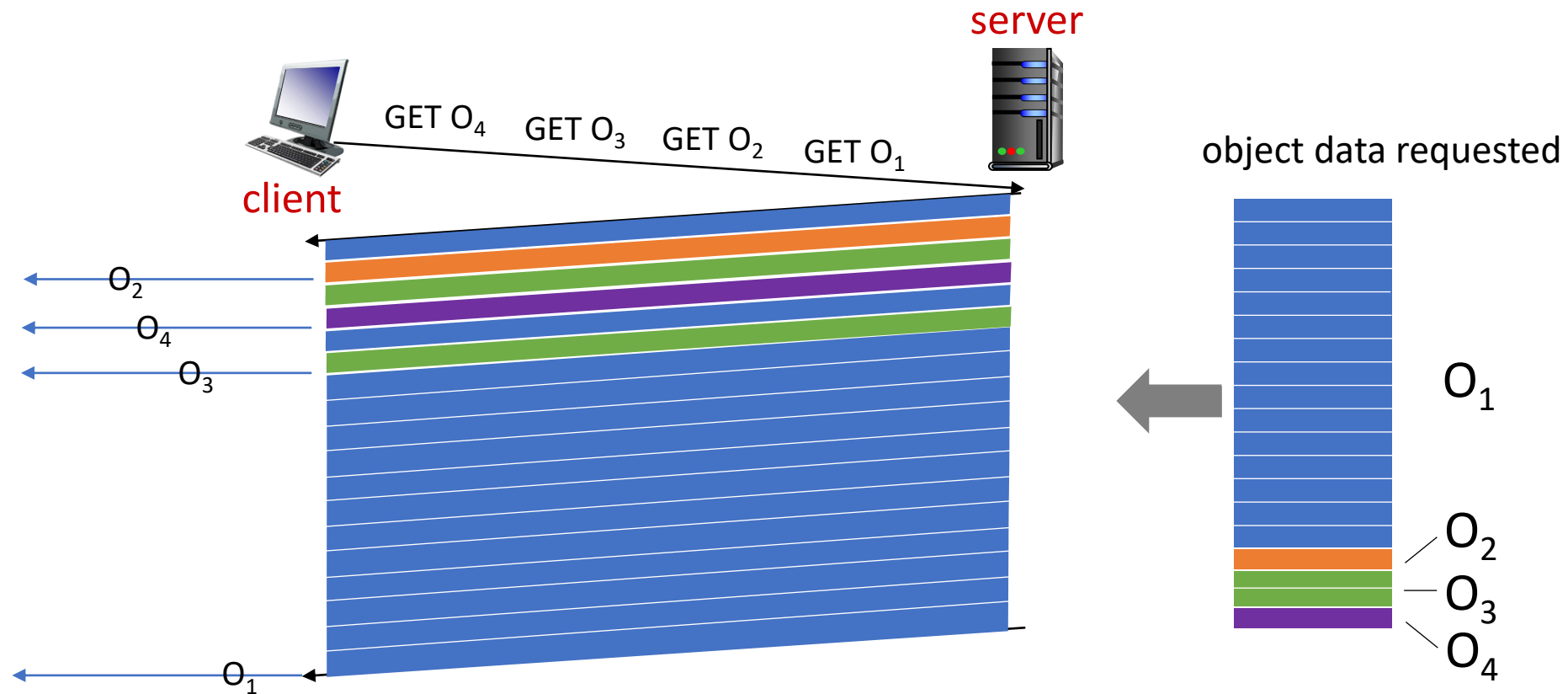
# HTTP/2: mitigating HOL blocking

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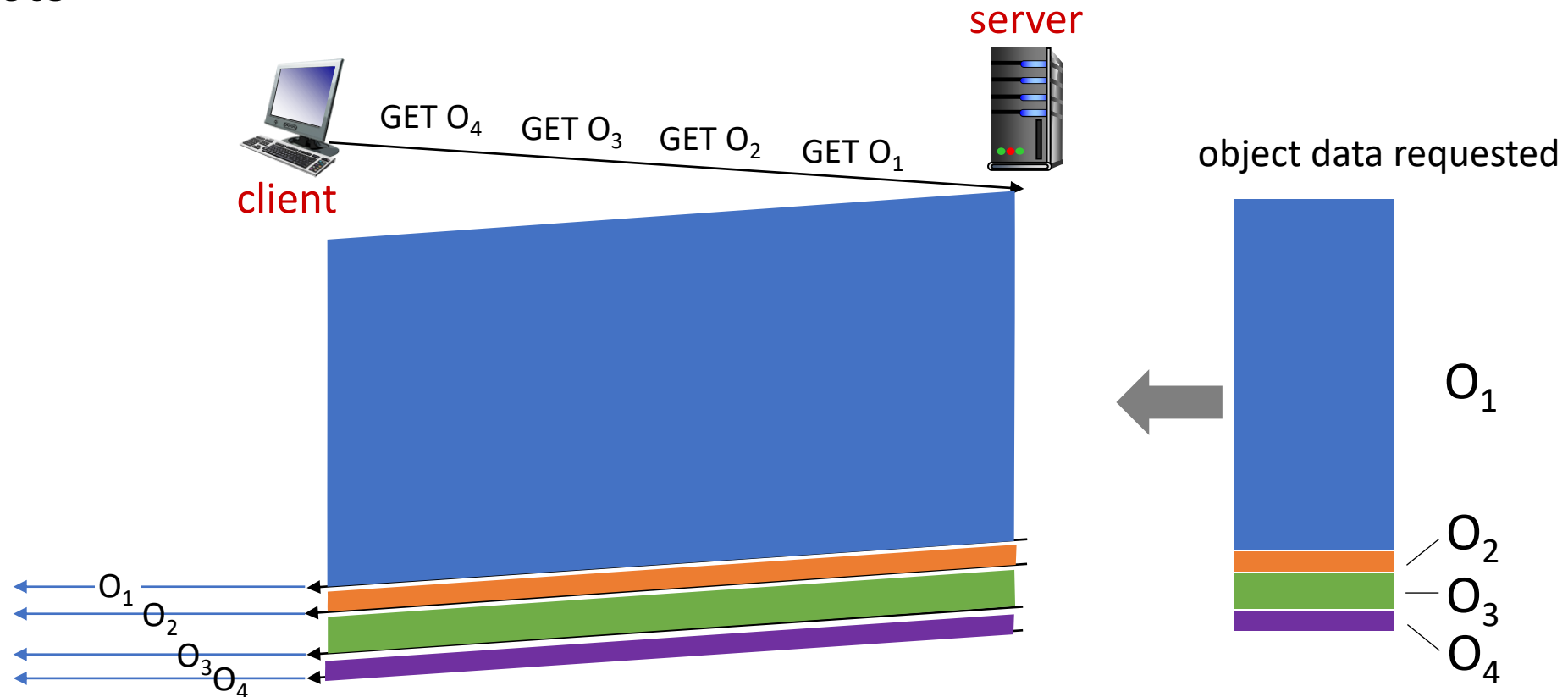


*objects delivered in order requested: O₂, O₃, O₄ wait behind O₁*

# 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*

# HTTP/2 to HTTP/3

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

# 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

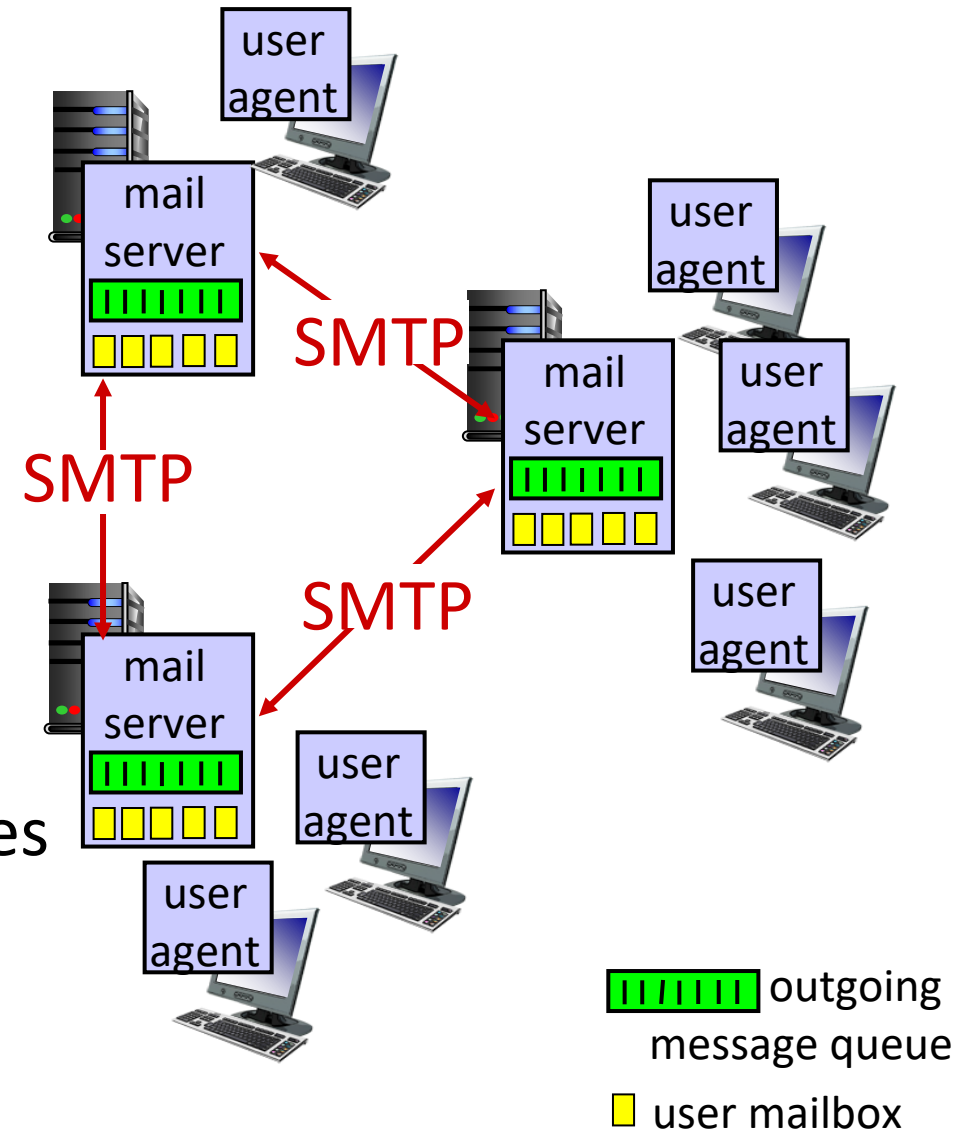- socket programming with UDP and TCP

# 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
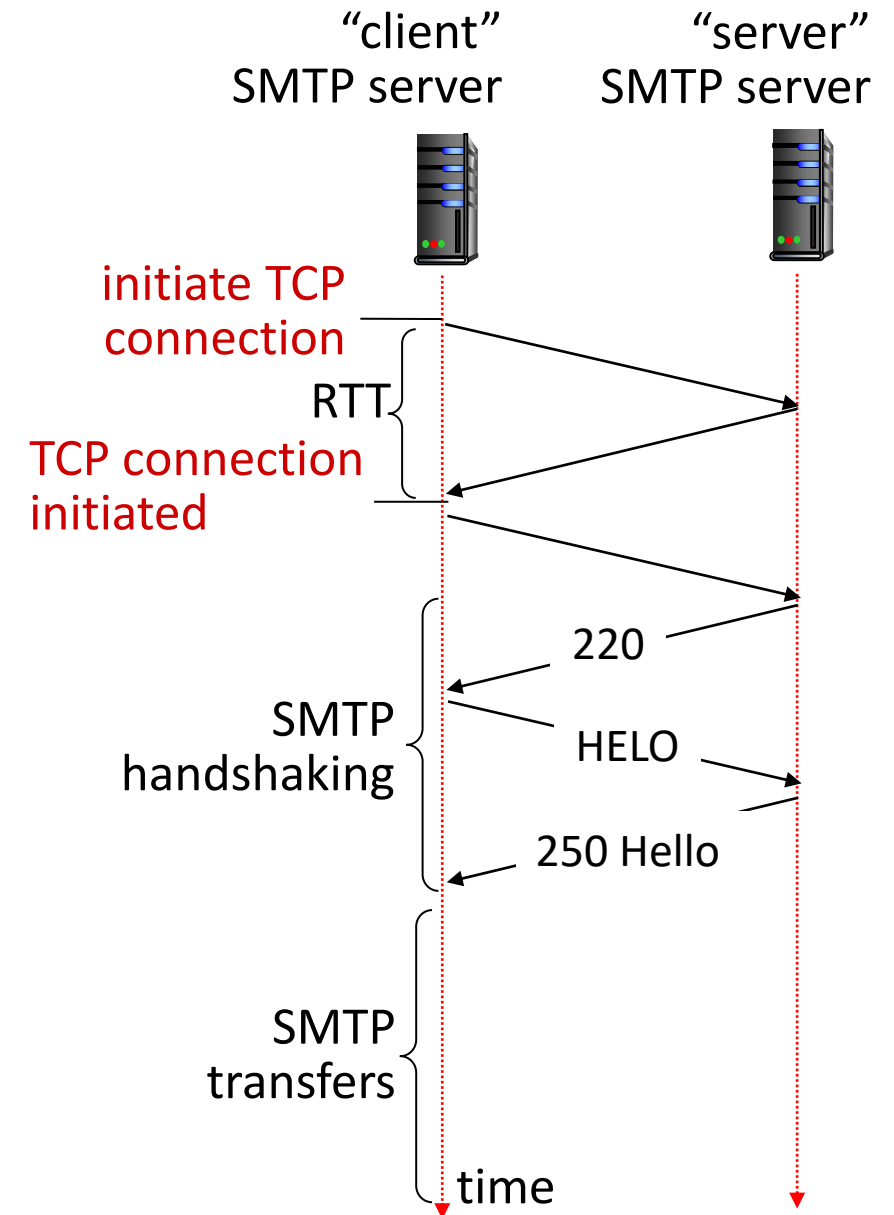
- *message queue* of outgoing (to be sent) mail messages

**SMTP protocol** between mail servers to send email messages

- **client:** sending mail server

- **"server":** receiving mail server



outgoing message queue

user mailbox
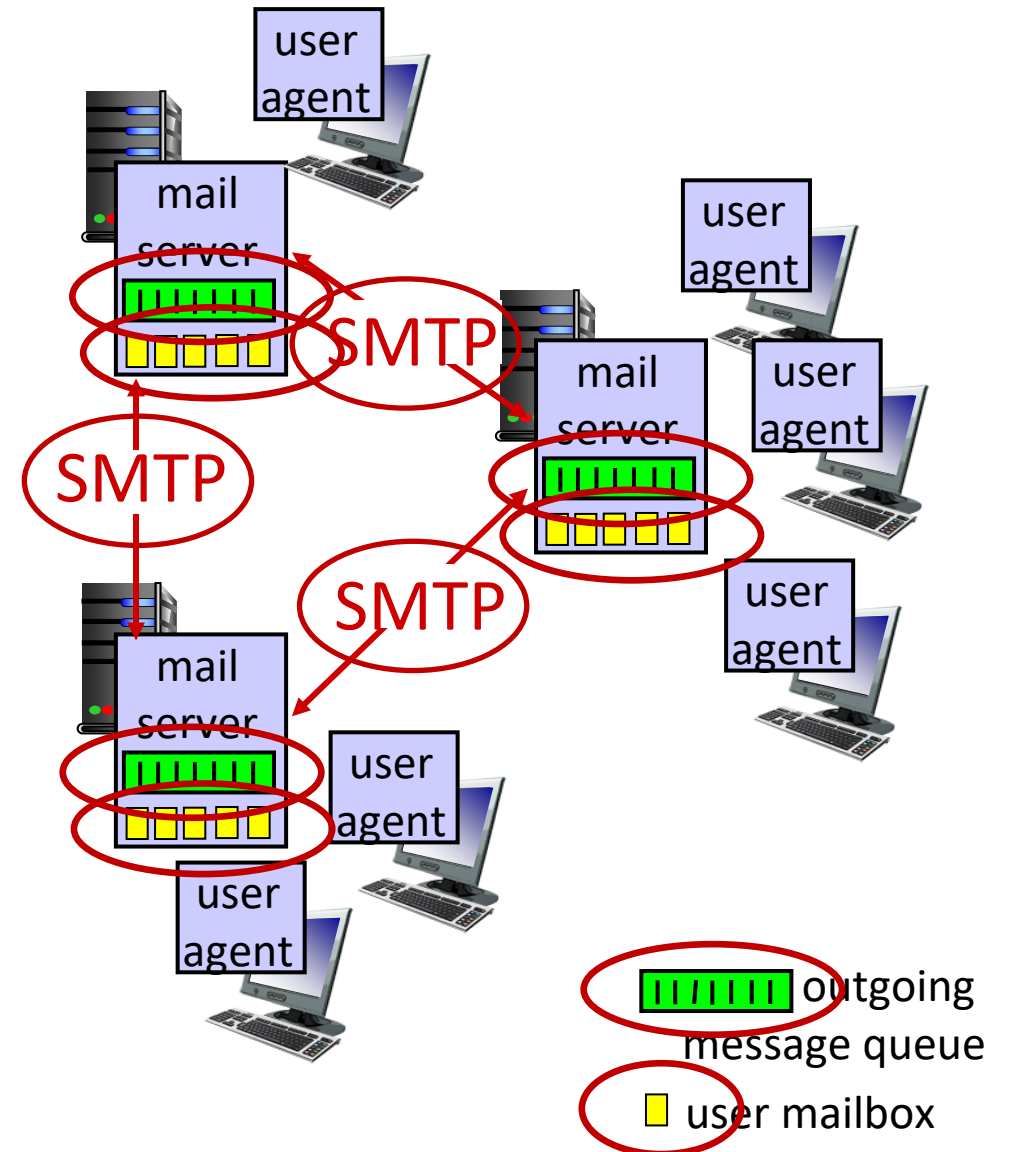
# SMTP RFC (5321)

- 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
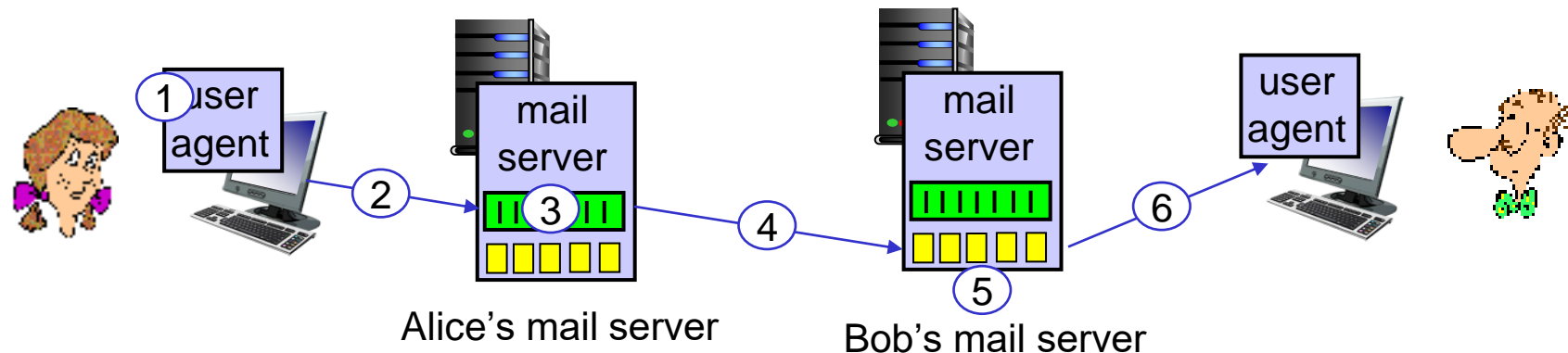
# Sample SMTP interaction

```
S: 220 hamburger.edu
```

# 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:
  - From:
  - Subject:
  
  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the "message" , ASCII characters only

| header |
|--------|

blank line

| body |
|------|
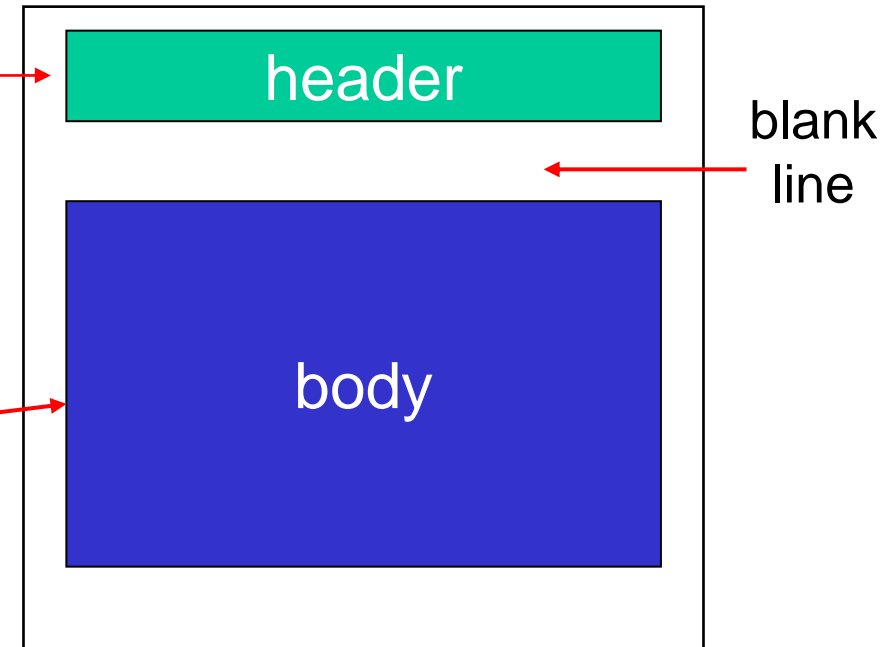
# Retrieving email: mail access protocols
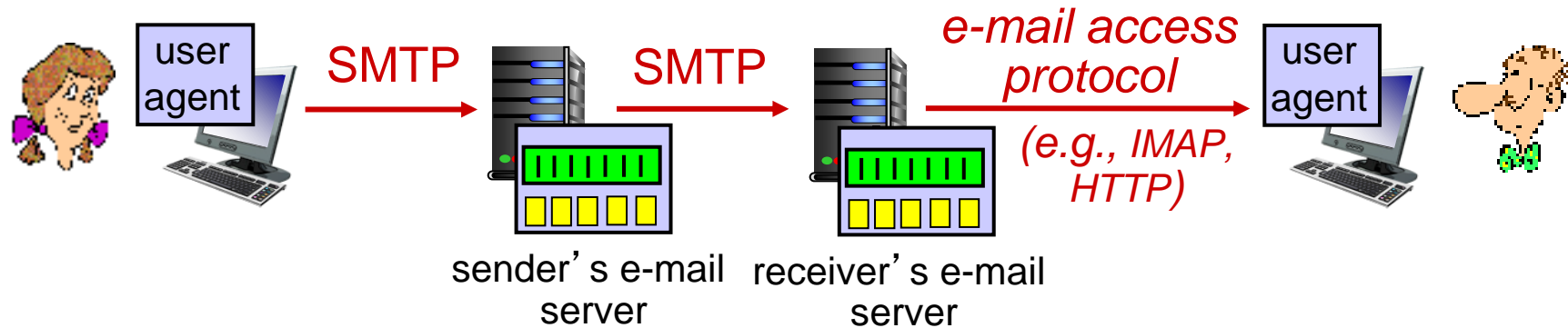


- **SMTP:** delivery/storage of e-mail messages to receiver's server

- **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 STMP (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

- mail server aliasing

- load distribution
  - replicated Web servers: many IP addresses correspond to one name

*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
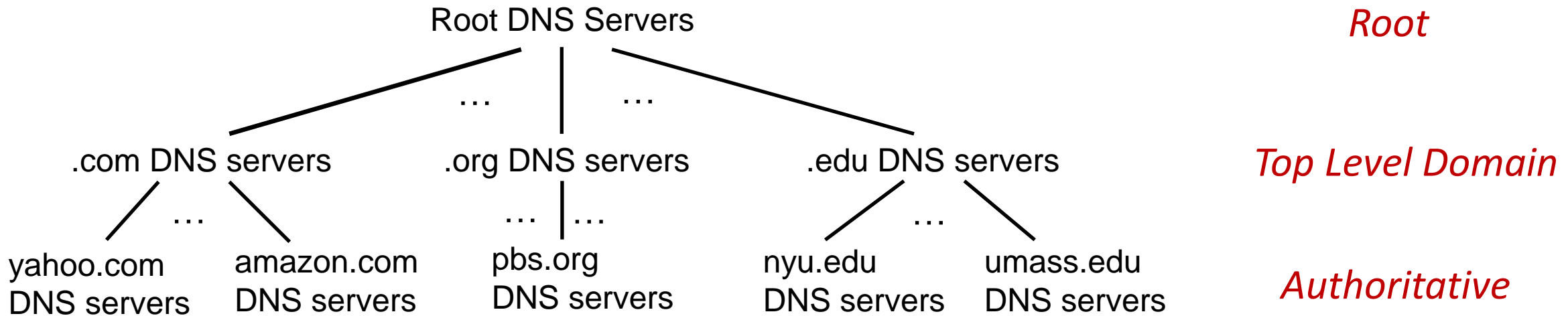- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

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

**"bulletproof": reliability, security**

# DNS: a distributed, hierarchical database

Root DNS Servers                                                    *Root*

.com DNS servers          .org DNS servers          .edu DNS servers          *Top Level Domain*

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

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