


Computer Networks

CMSC 417 : Spring 2024

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

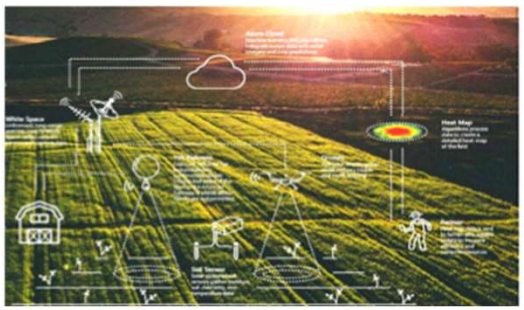
Topics:
BGP Attacks (continued…) (Textbook chapter 4)
Application layer (Textbook chapter 2)

Nirupam Roy

Tu-Th 2:00-3:15pm
CSI 2117

May 7th, 2024

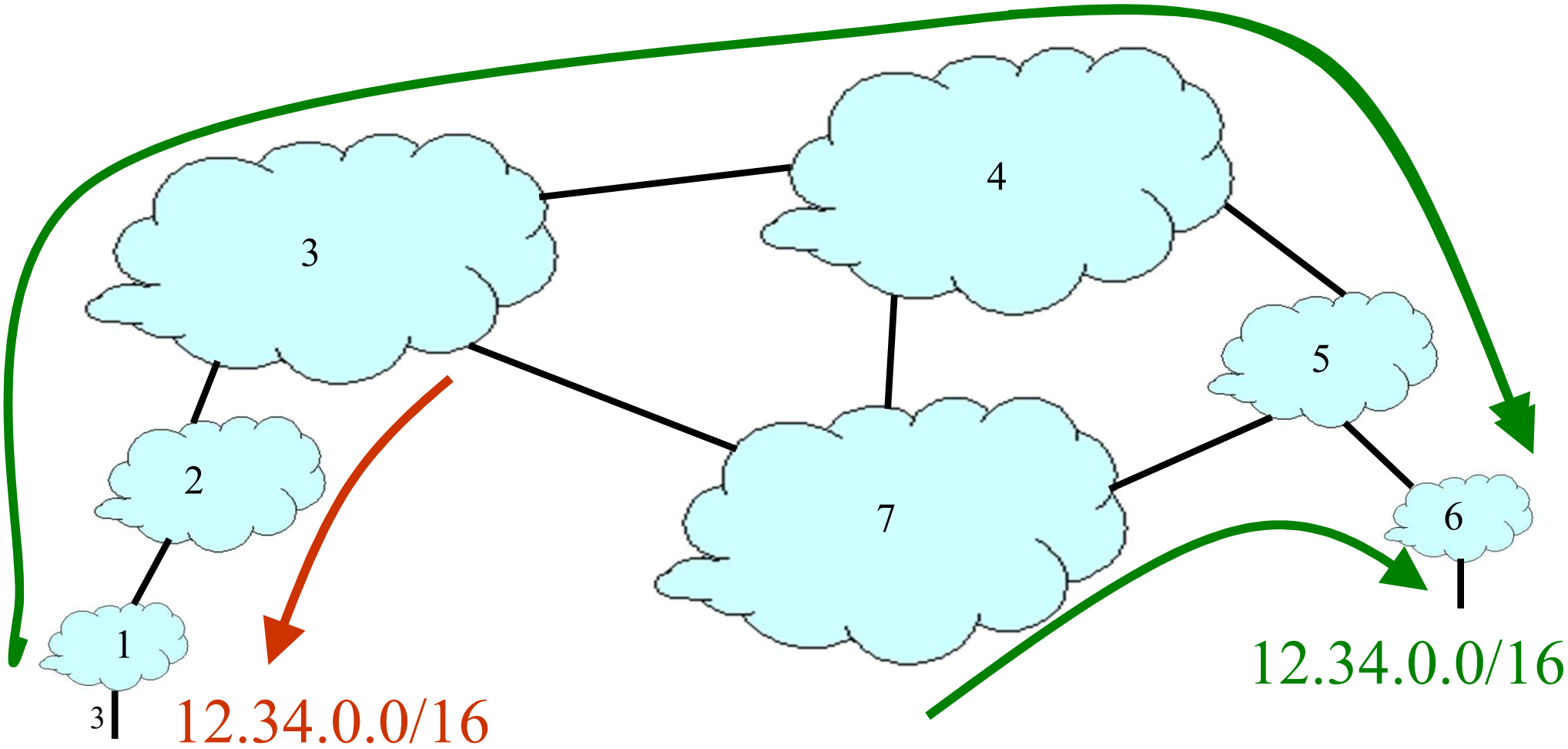





BGP Attacks/Misconfiguration (cont…)

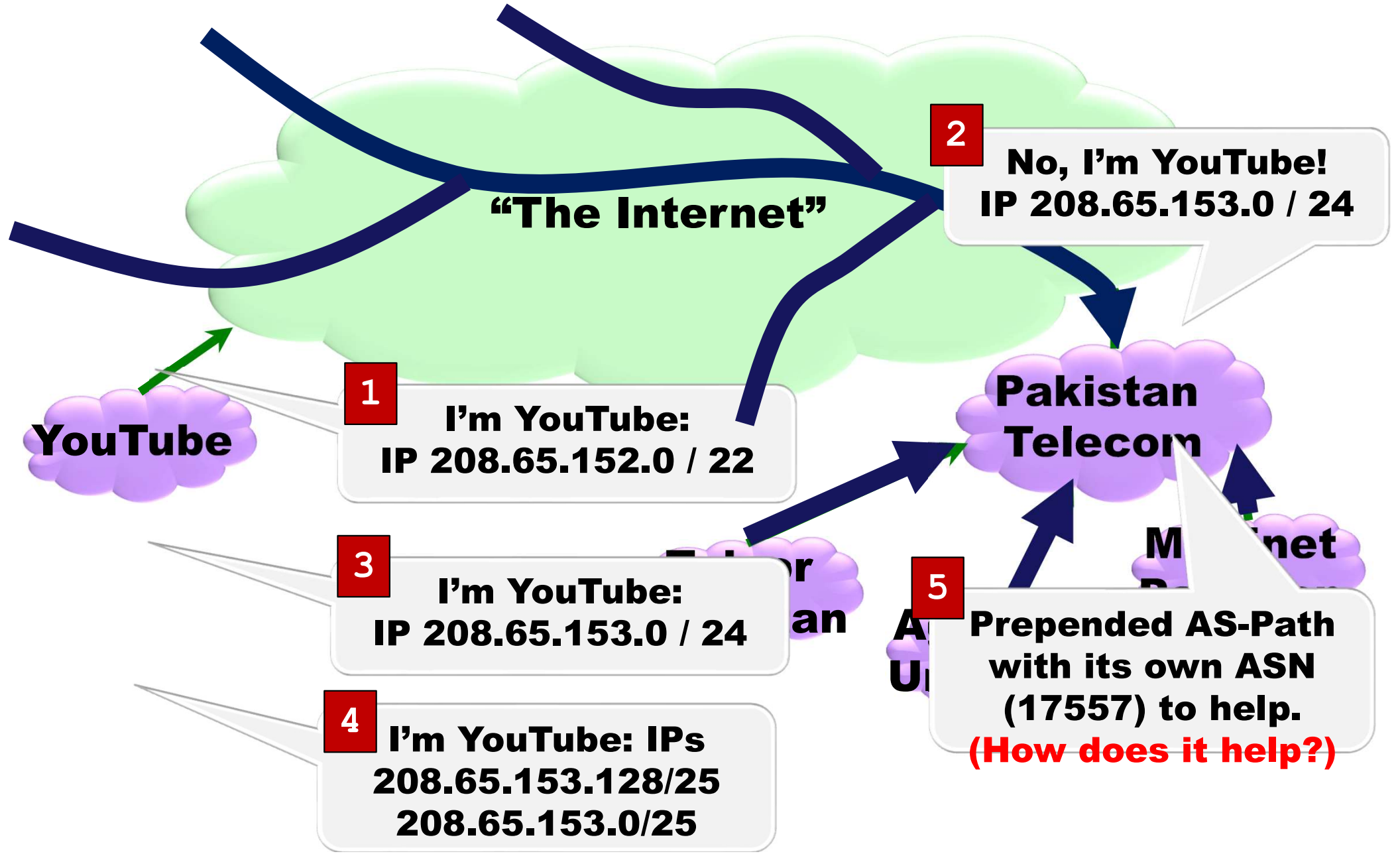
Prefix Hijacking

- **Originating someone else's prefix**
  - **What fraction of the Internet believes it?**



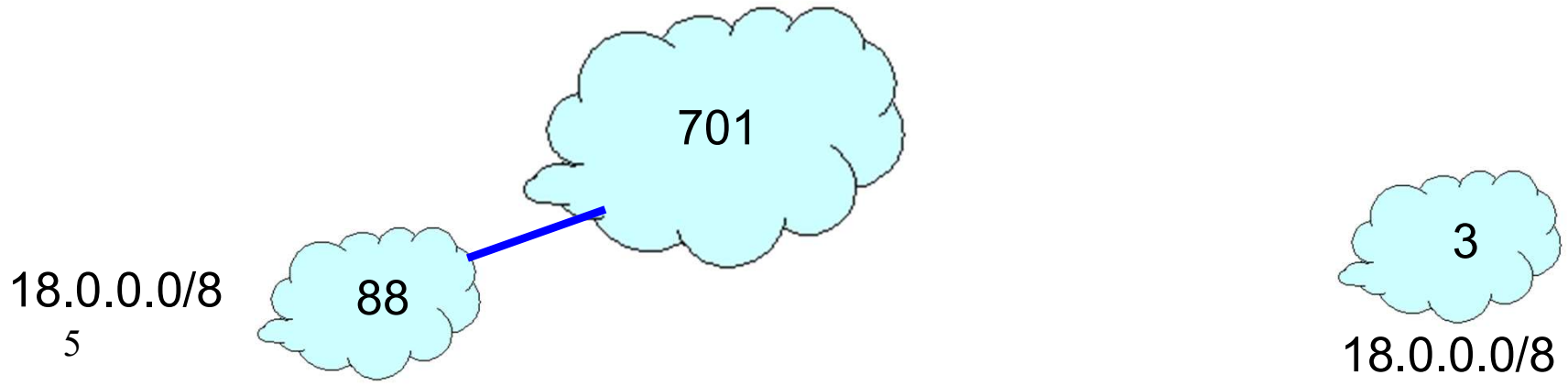
Sub-prefix hijack

But here's what Pakistan ended up doing…



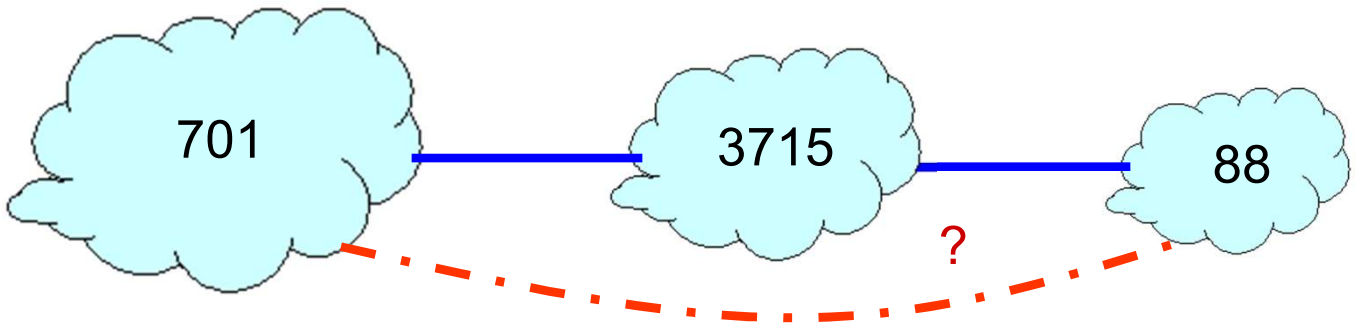
Bogus AS Paths to Hide Hijacking

- **Adds AS hop(s) at the end of the path**
  - **E.g., turns "701 88" into "701 88 3"**
- **Motivations**
  - **Evade detection for a bogus route**
  - **E.g., by adding the legitimate AS to the end**
- **Hard to tell that the AS path is bogus…**
  - **Even if other ASes filter based on prefix ownership**

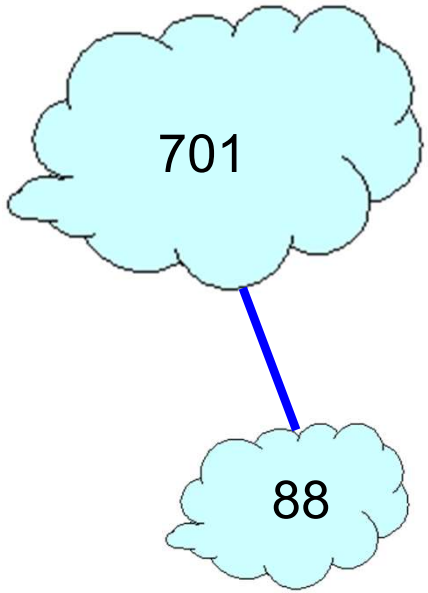


Path-Shortening Attacks

- **Remove ASes from the AS path**
  - **E.g., turn "701 3715 88" into "701 88"**
- **Motivations**
  - **Make the AS path look shorter than it is**
  - **Attract sources that normally try to avoid AS 3715**
  - **Help AS 88 look like it is closer to the Internet's core**
- **Who can tell that this AS path is a lie?**
  - **Maybe AS 88 *does* connect to AS 701 directly**



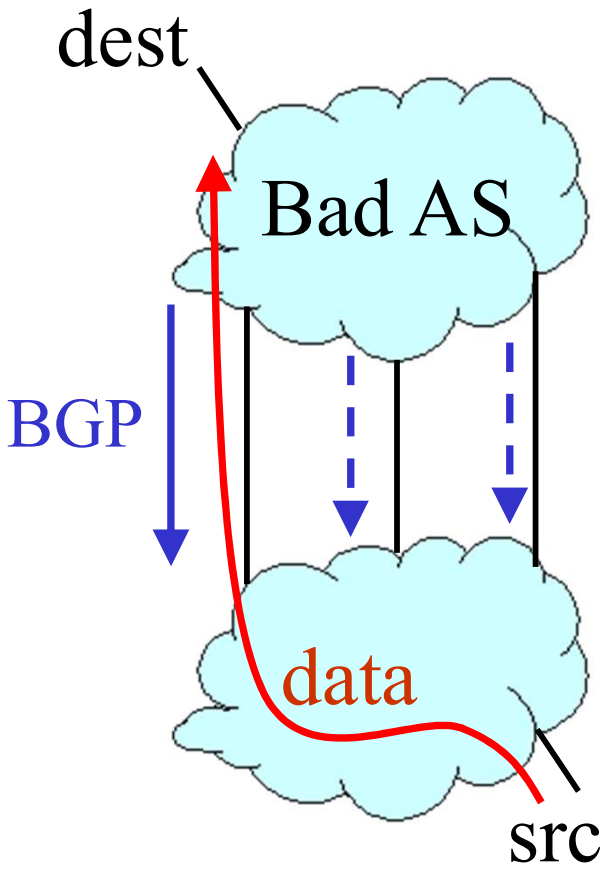
Attacks that Add a Bogus AS Hop



- **Add ASes to the path**
  - **E.g., turn "701 88" into "701 3715 88"**
- **Motivations**
  - **Trigger loop detection in AS 3715**
    - **Denial-of-service attack on AS 3715**
    - **Or, blocking unwanted traffic coming from AS 3715!**
  - **Make your AS look like is has richer connectivity**
- **Who can tell the AS path is a lie?**
  - **AS 3715 could, if it could see the route**
  - **AS 88 could, but would it really care as long as it received data traffic meant for it?**

Violating "Consistent Export" to Peers

- **Peers require consistent export**
  - **Prefix advertised at all peering points**
  - **Prefix advertised with same AS path length**
- **Reasons for violating the policy**
  - **Trick neighbor into "cold potato"**
  - **Configuration mistake**
- **Main defense**
  - **Analyzing BGP updates**
  - **… or data traffic**
  - **… for signs of inconsistency**



Other Attacks

- **Attacks on BGP sessions**
  - **Confidentiality of BGP messages**
  - **Denial-of-service on BGP session**
  - **Inserting, deleting, modifying, or replaying messages**
- **Resource exhaustion attacks**
  - **Too many IP prefixes (e.g., BGP “512K Day”)**
  - **Too many BGP update messages**
- **Data-plane attacks**
  - **Announce one BGP routes, but use another**

Solution Techniques

- **Protective filtering**
  - **Know your neighbors**
- **Anomaly detection**
  - **Suspect the unexpected**
- **Checking against registries**
  - **Establish ground truth for prefix origination**
- **Signing and verifying**
  - **Prevent bogus AS PATHs**
- **Data-plane verification**
  - **Ensure the path is actually followed**




Computer Networks

CMSC 417 : Spring 2024

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

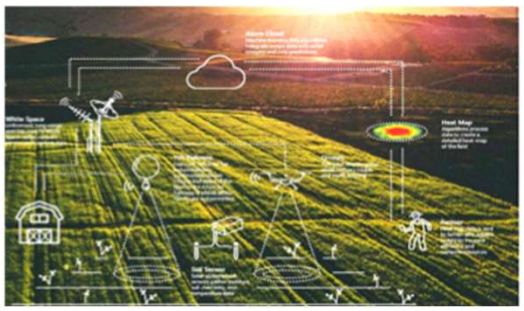
Topic: Application Layer Protocols Part-1
(Textbook chapter 9)

Nirupam Roy

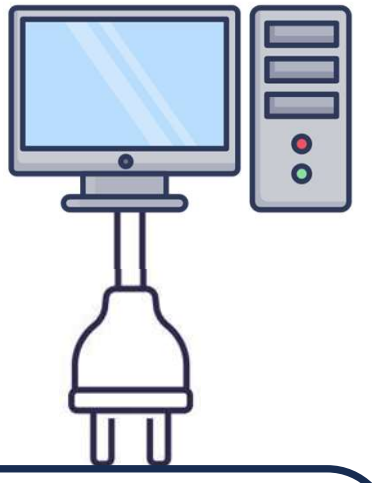
Tu-Th 2:00-3:15pm
CSI 2117

May 7th, 2024






Applications

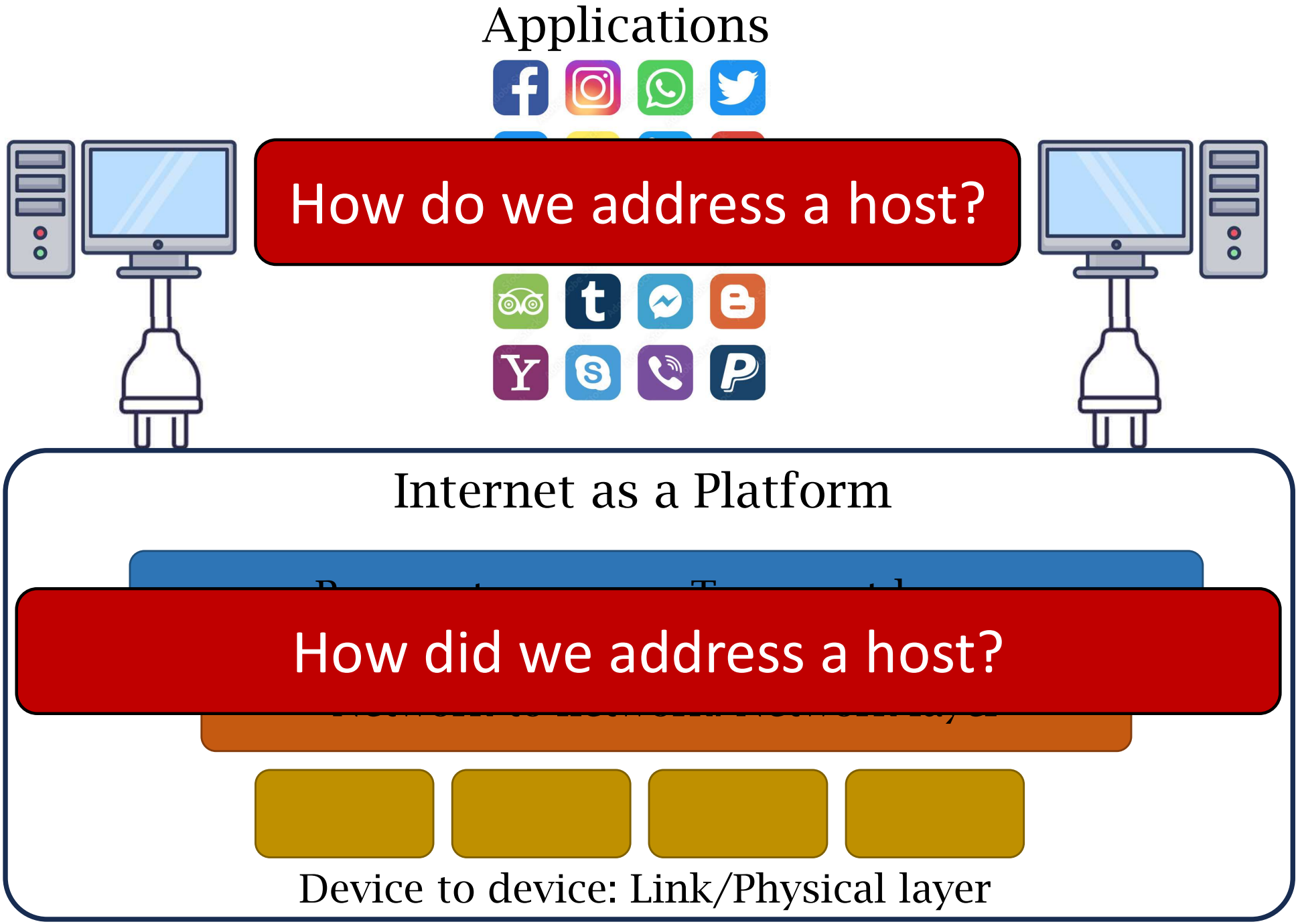


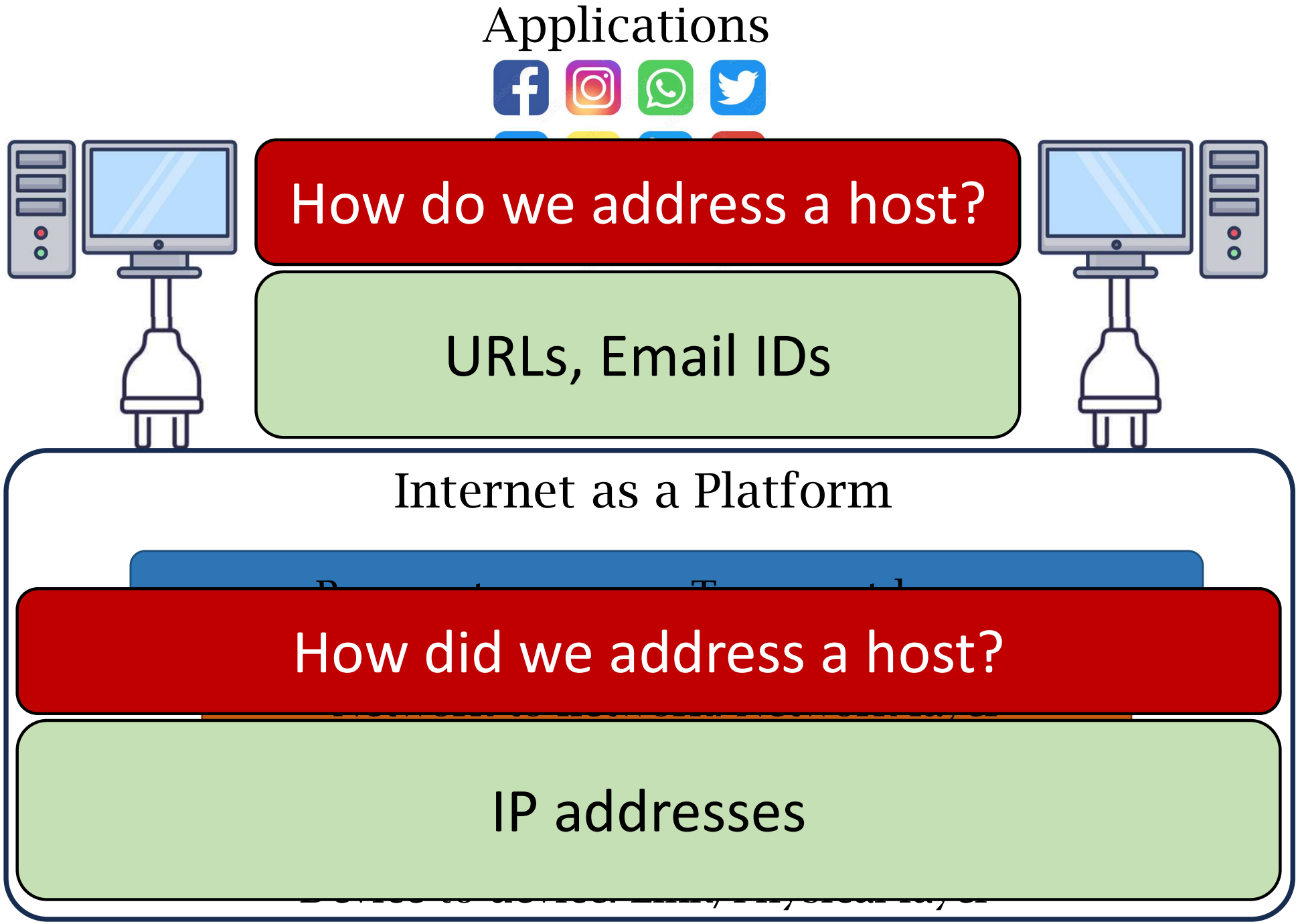
Internet as a Platform

Process to process: Transport layer

Network to network: Network layer

Device to device: Link/Physical layer


Applications
How do we address a host?
Internet as a Platform
How did we address a host?
Device to device: Link/Physical layer


Applications
How do we address a host?
URLs, Email IDs
Internet as a Platform
How did we address a host?
IP addresses

Applications (World wide web)

https://www.cs.umd.edu

Webserver

HTTP protocol responses

Internet as a Platform

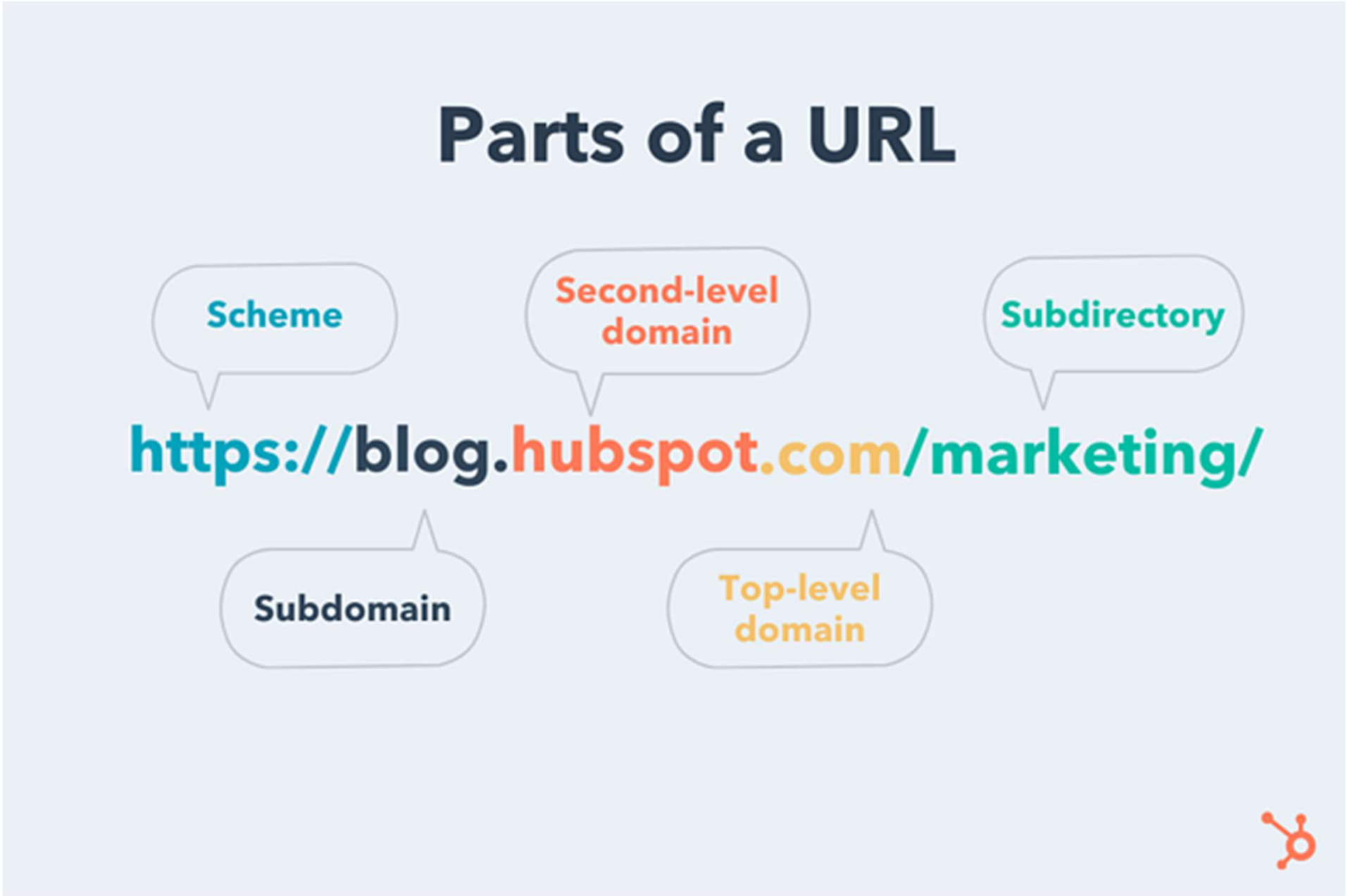
How did we address a host?

IP addresses

We need a database on the Internet
for
Human readable domain name → IP address mapping

Questions:

(1) Centralized or distributed?
(2) Hierarchical or flat?
(3) At the application layer or at a lower layer?



PC: https://blog.hubspot.com/marketing/parts-url

Anatomy of a domain name

https://www.cs.umd.edu

https://www.ece.umd.edu

ftp://ftp.example.com/Batch_File

nirupam@cs.umd.edu

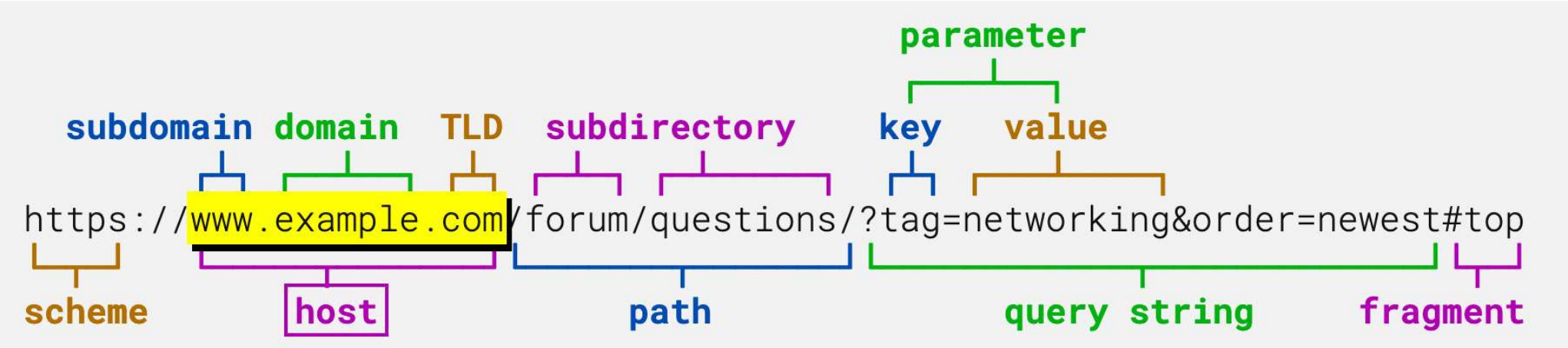
Anatomy of a domain name

https://www.cs.umd.edu

https://www.ece.umd.edu

ftp://ftp.example.com/Batch_File

nirupam@cs.umd.edu



[Picture] https://strategylab.ca/parts-of-a-url/

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

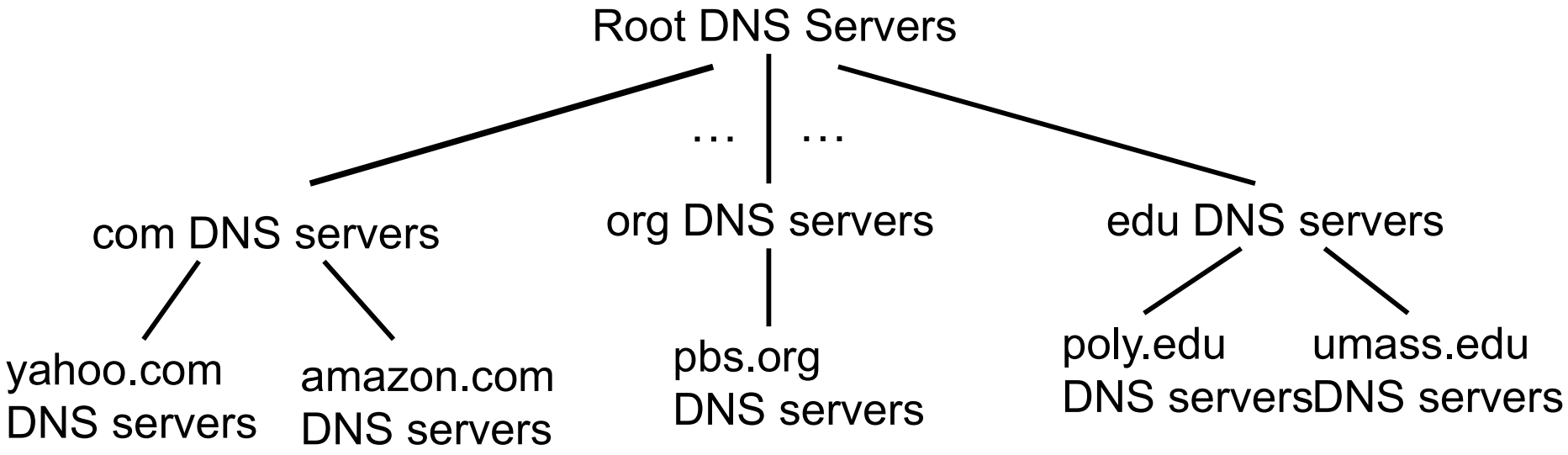
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

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

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: a distributed, hierarchical database

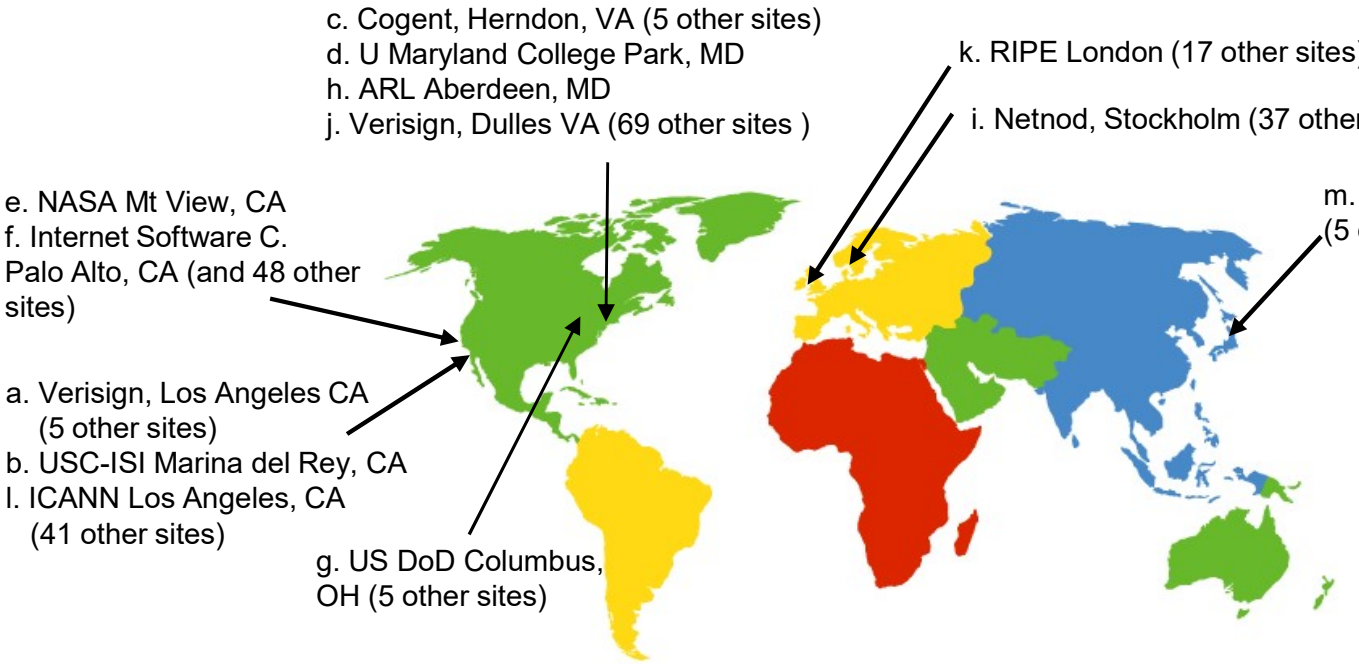


client wants IP for www.amazon.com; 1st approximation:

- 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

- Contacted by local name server that cannot resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server



13 logical root name "servers" worldwide

- *each "server" replicated many times*

TLD, Authoritative DNS Servers

Top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

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 server

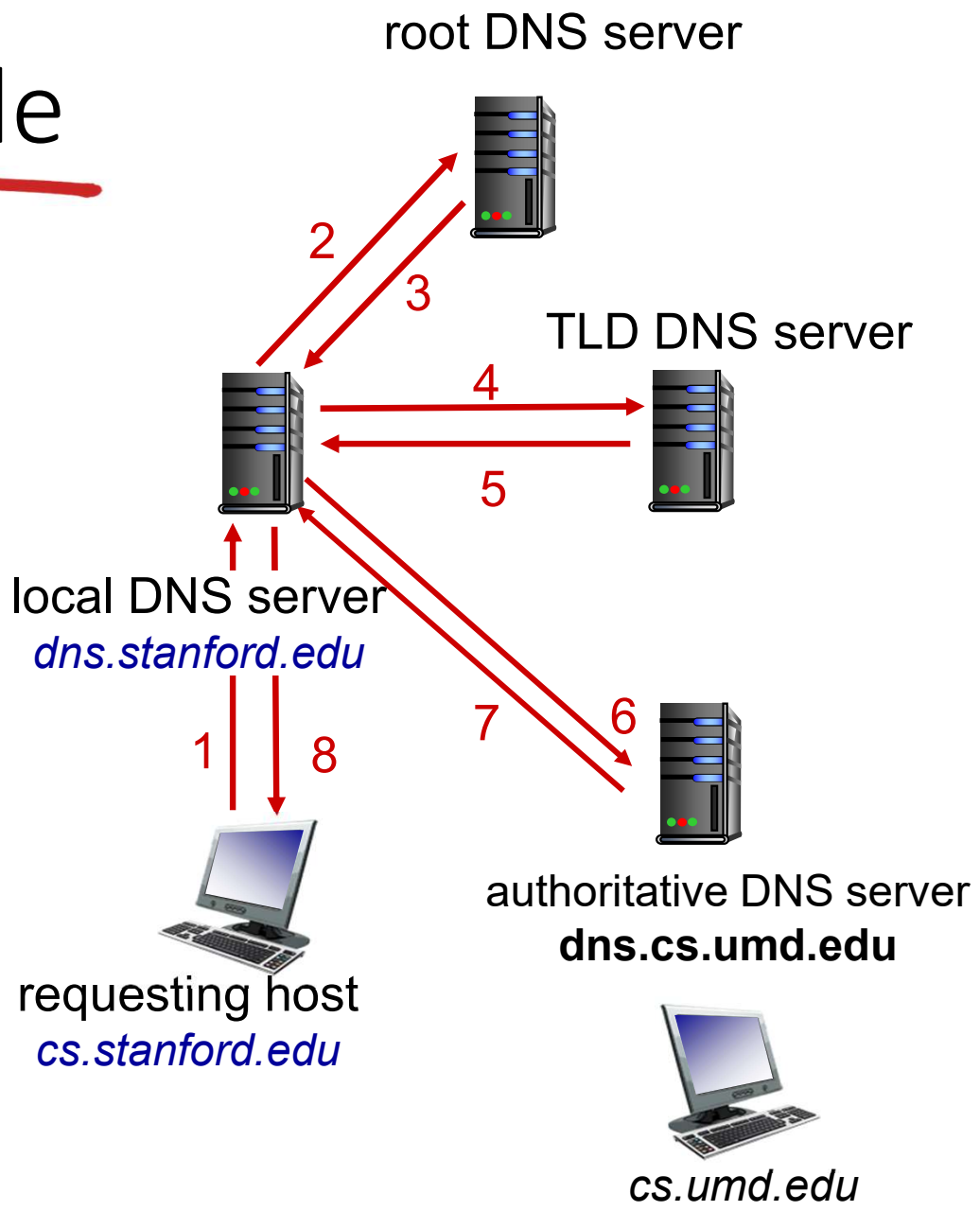
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at cs.stanford.edu wants IP address for cs.umd.edu

iterated query:

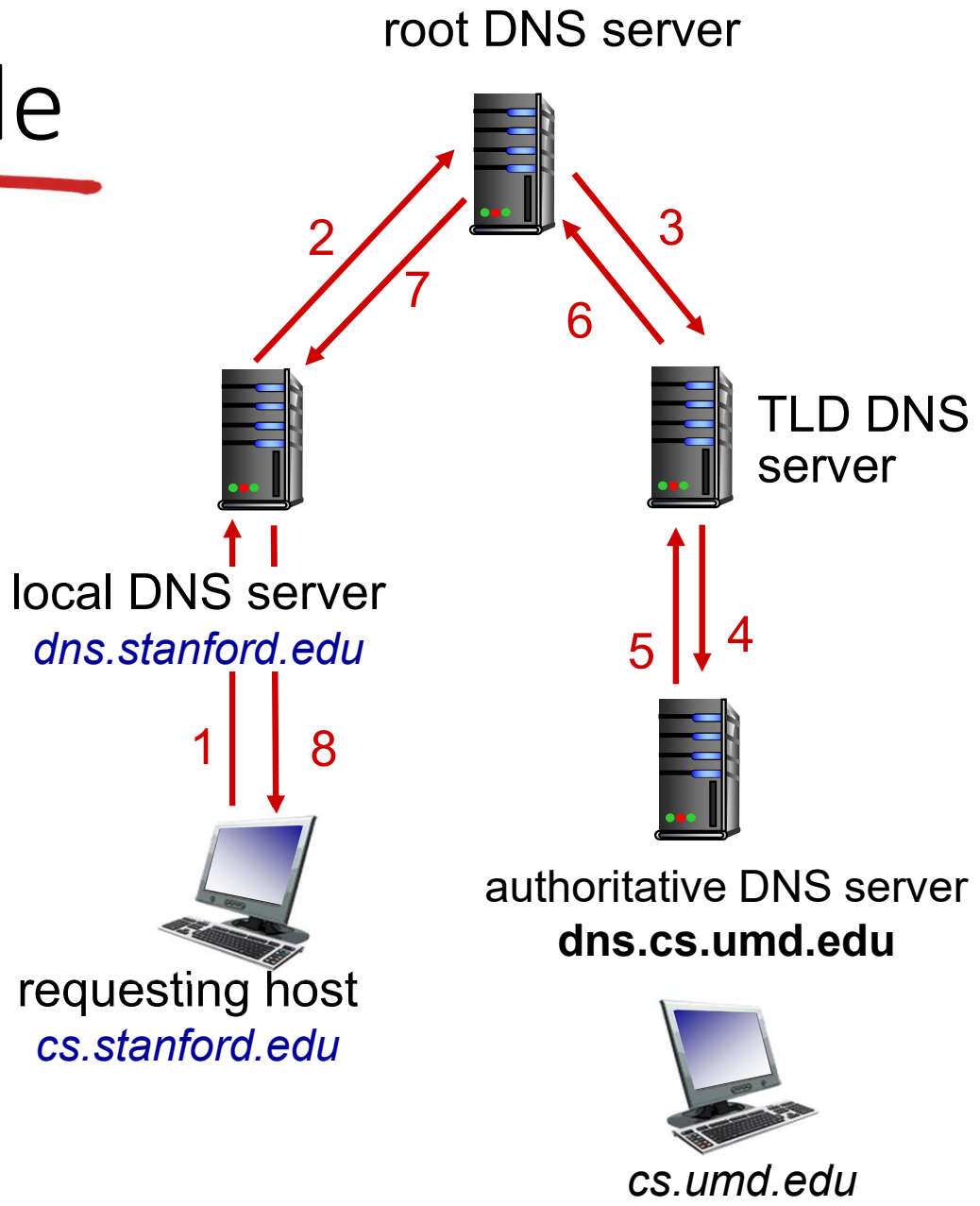
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

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

why not centralize DNS?

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

A: doesn‘t scale!

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

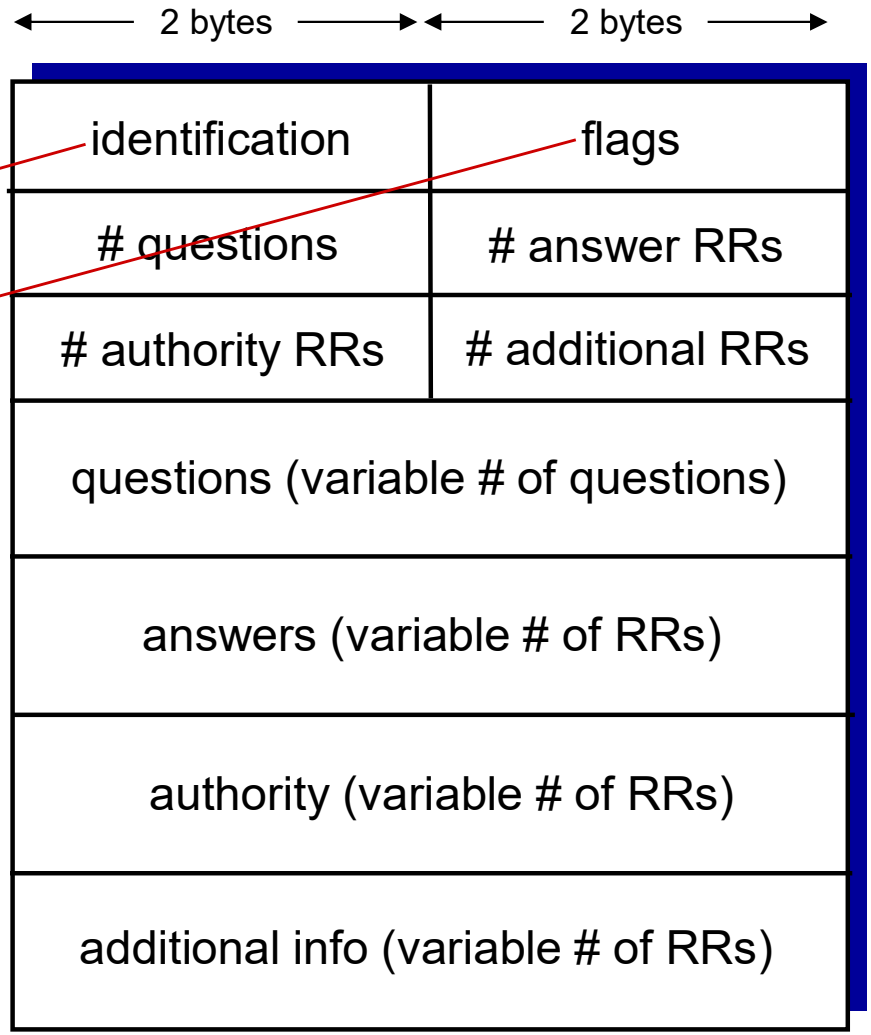
- **value** is name of mail server associated with **name**

DNS protocol, messages

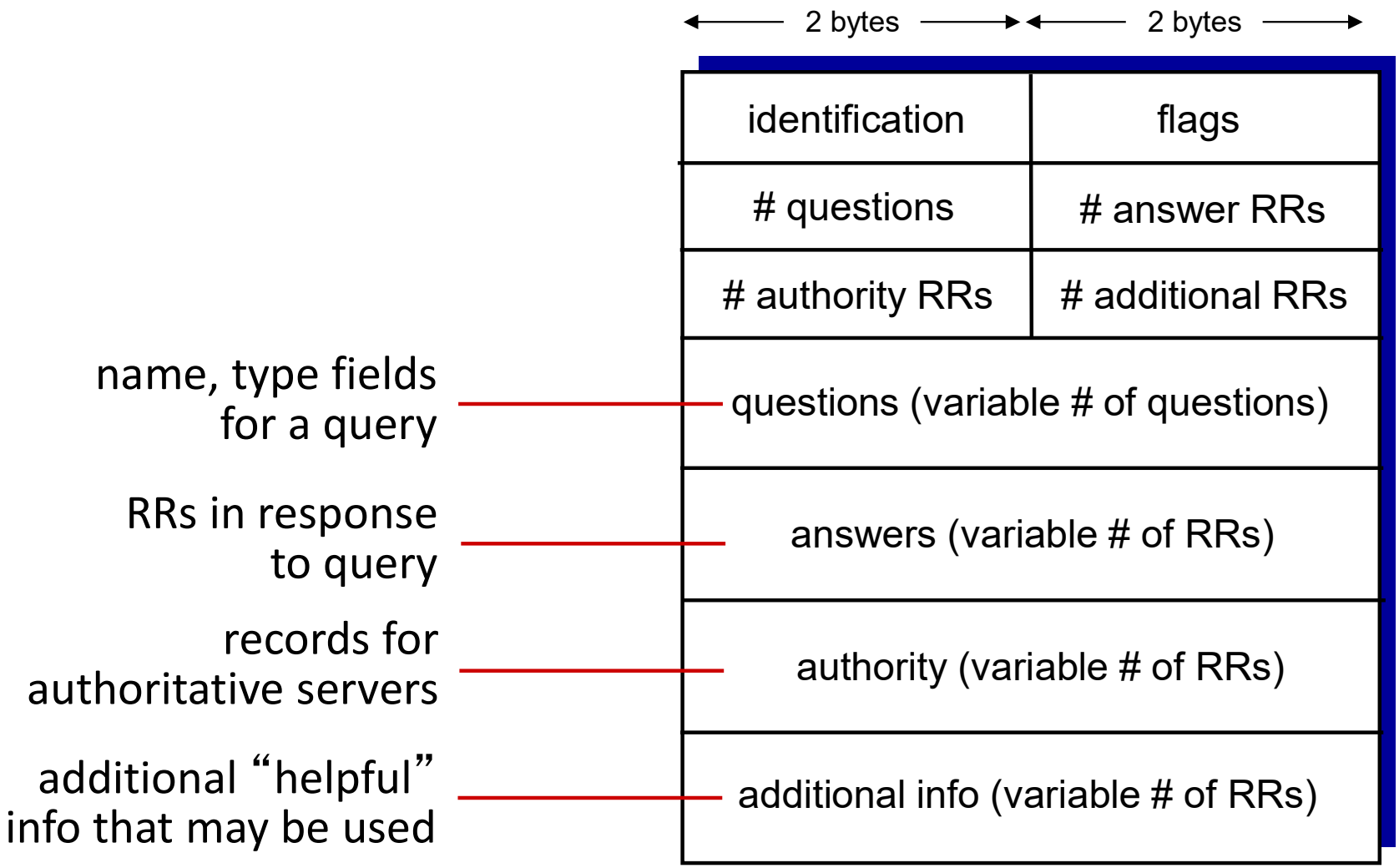
- *query* and *reply* messages, both with same *message format*

message header

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



DNS protocol, messages



Inserting records into 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 two RRs into .com TLD server:
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server type A record for www.networkuptopia.com; type NS record for networkutopia.com

HTTP
(Hyper-Text Transfer Protocol)

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

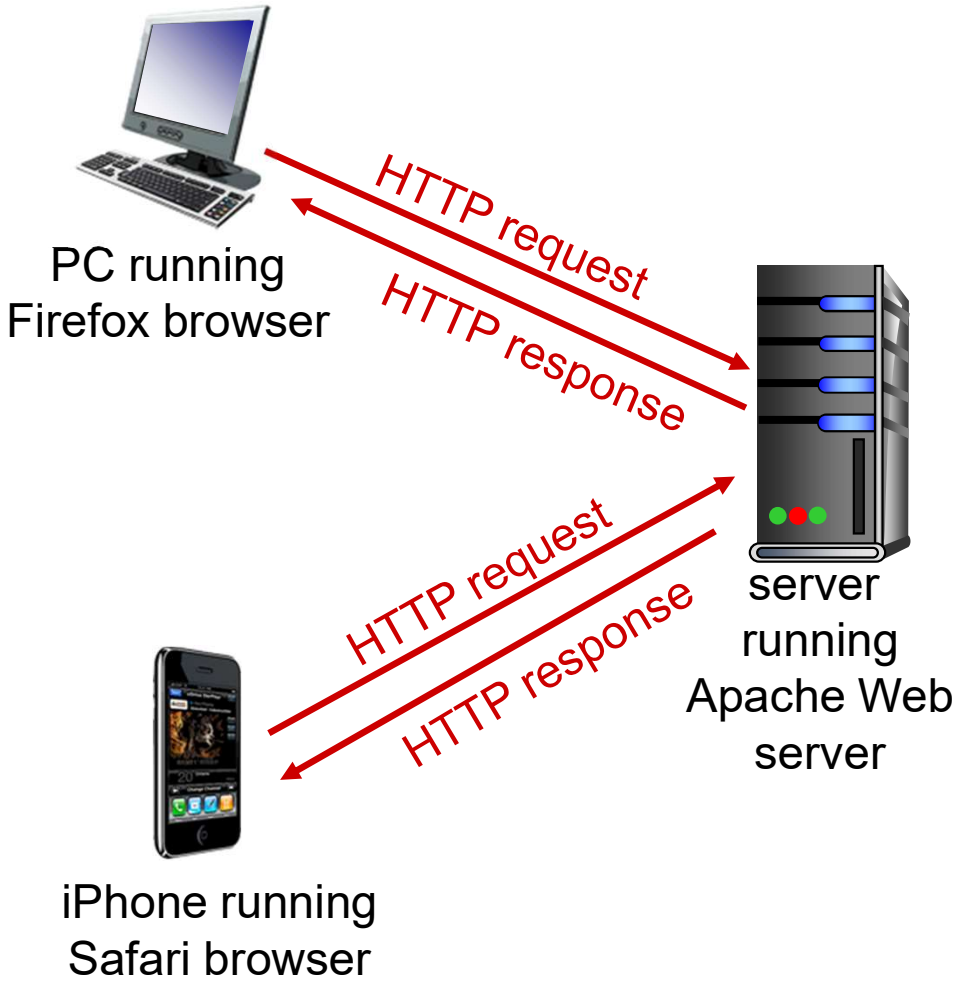
`www.someschool.edu/someDept/pic.gif`

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



HTTP overview

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

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

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

Non-persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index

(contains 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

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

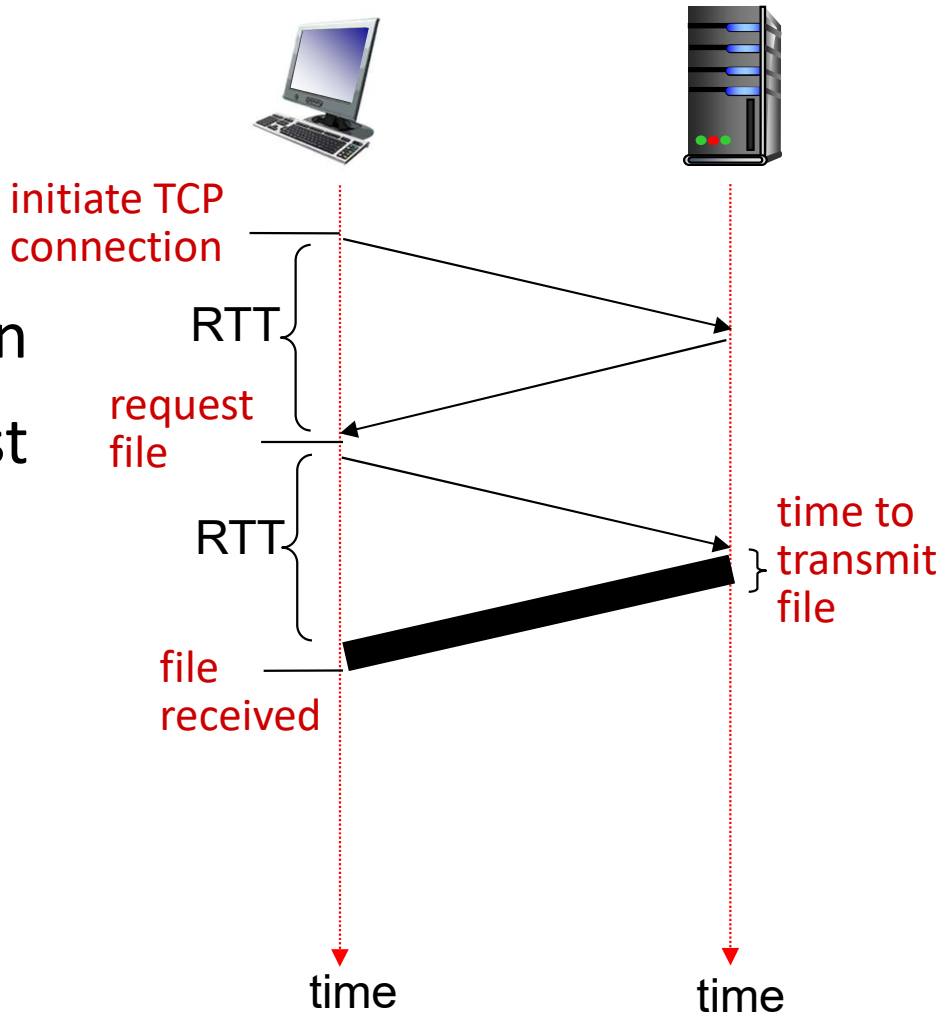
Non-persistent HTTP: response time

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

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time

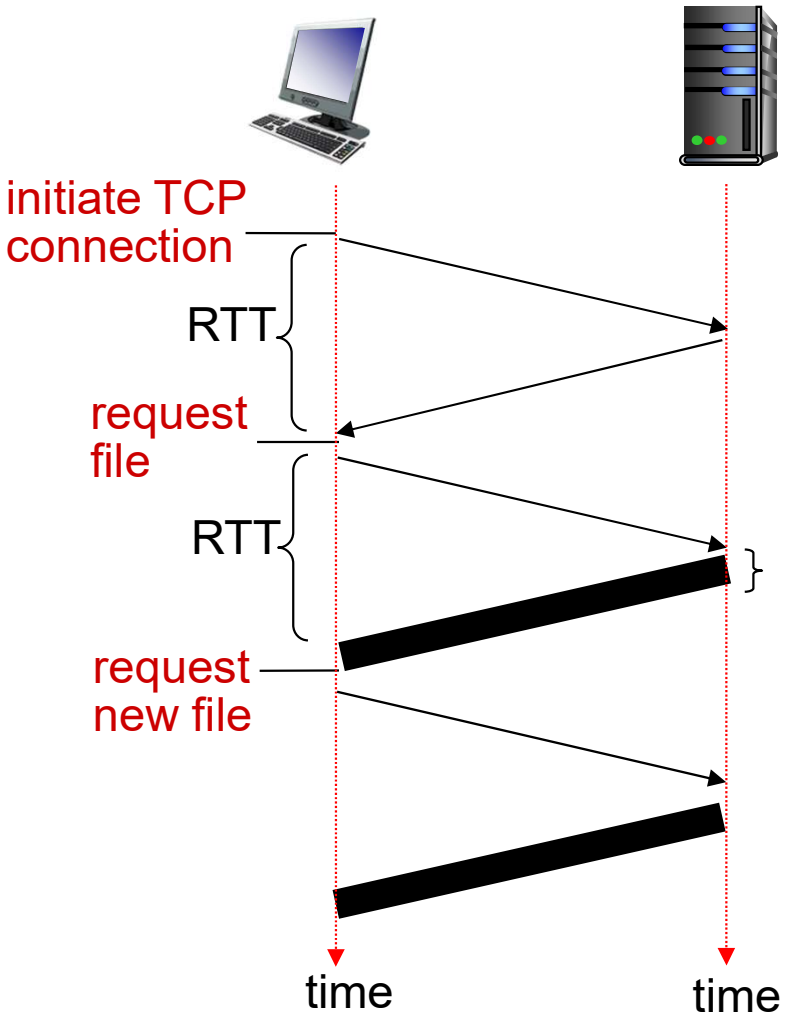
  = 2RTT+ file transmission time



Persistent HTTP

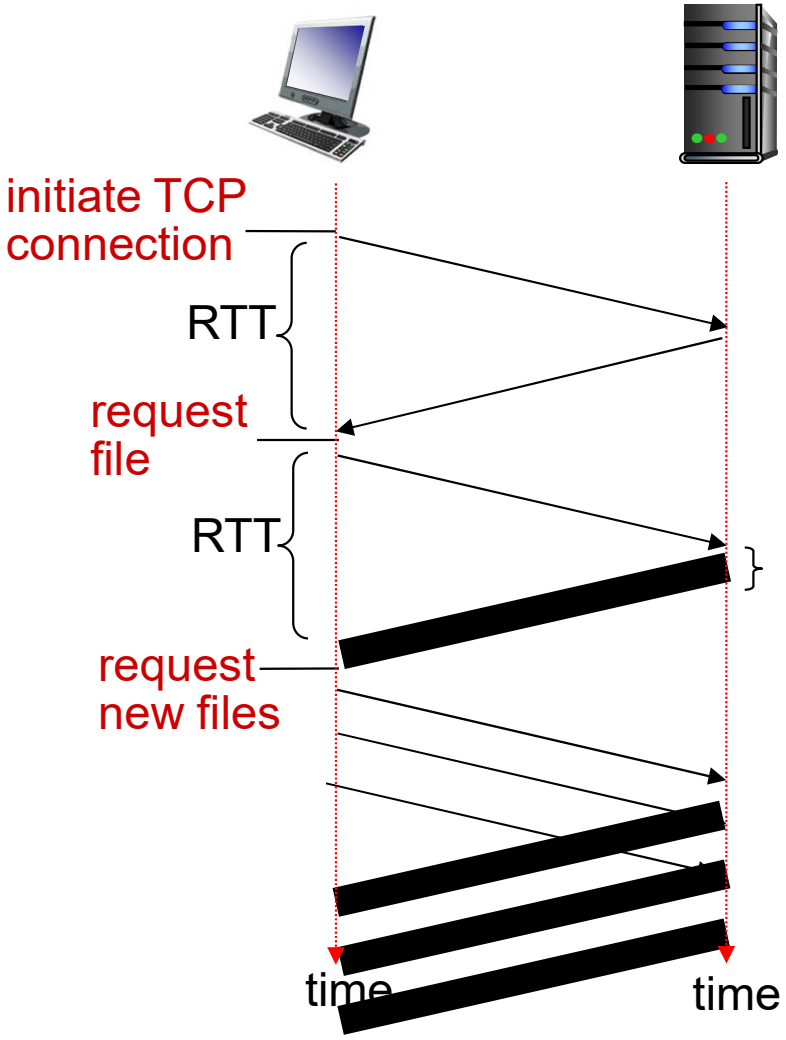
Persistent HTTP:

- 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



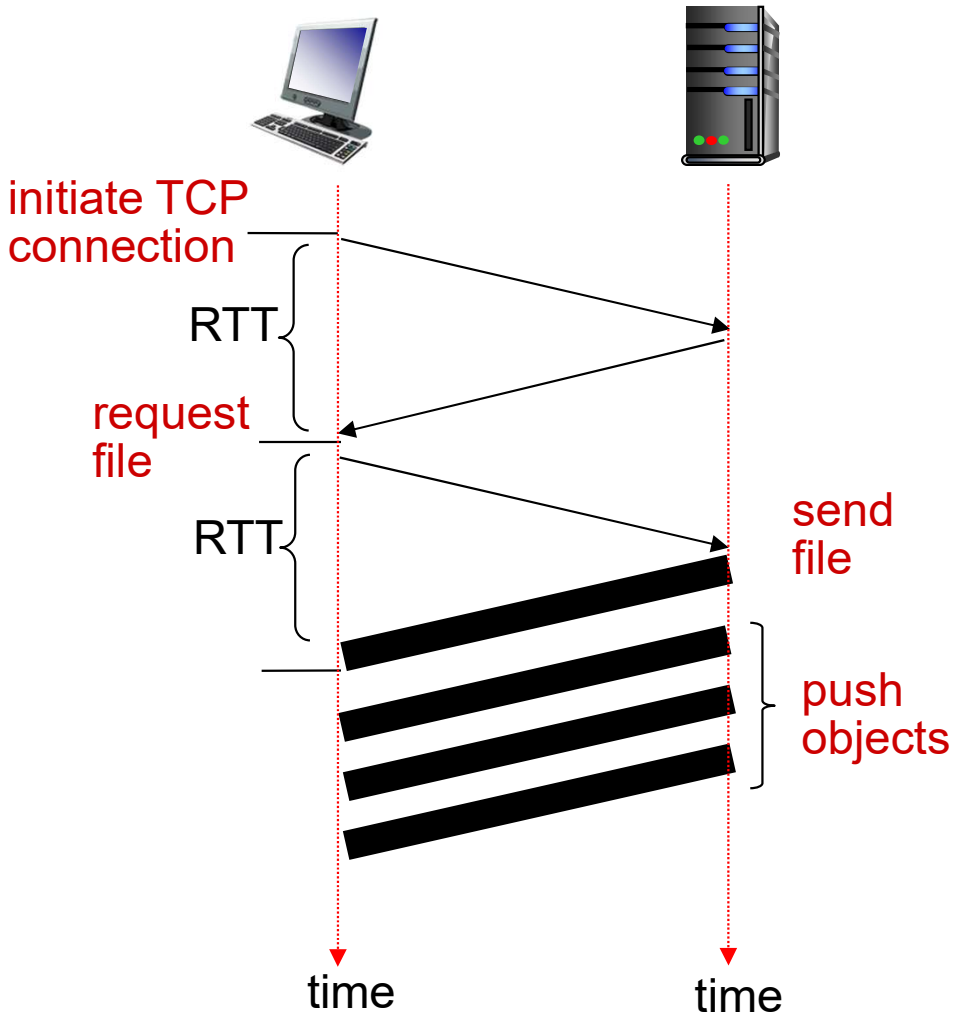
Other optimizations

- Pipelining
  - Send several requests at once



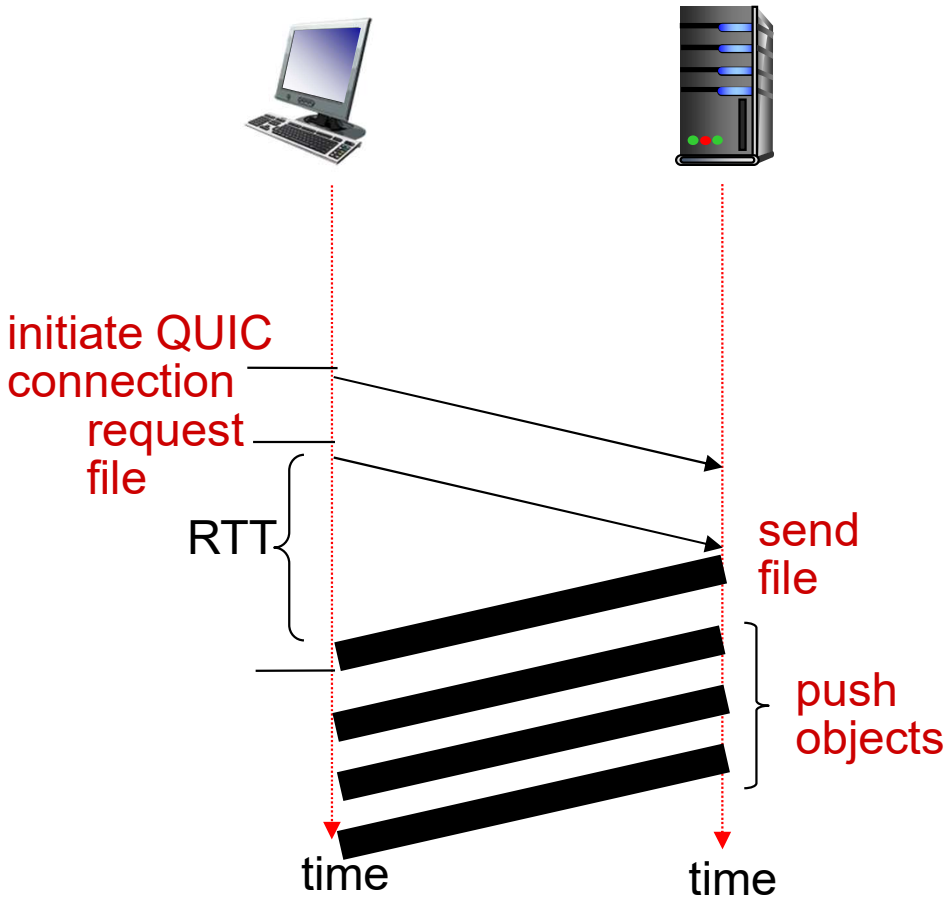
Other optimizations

- Pipelining
  - Send several requests at once
- HTTP/2
  - Push resources



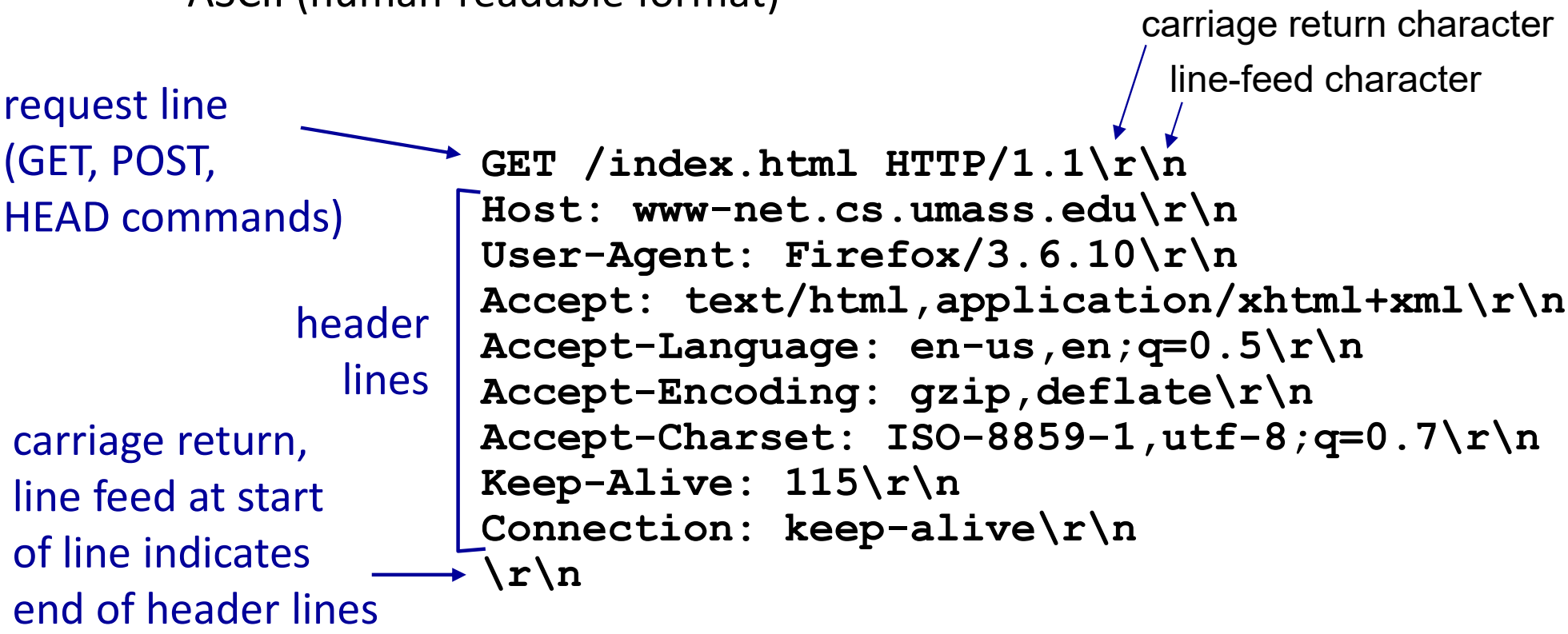
Other optimizations

- Pipelining
  - Send several requests at once
- HTTP/2
  - Push resources
- QUIC
  - Eliminate first RTT

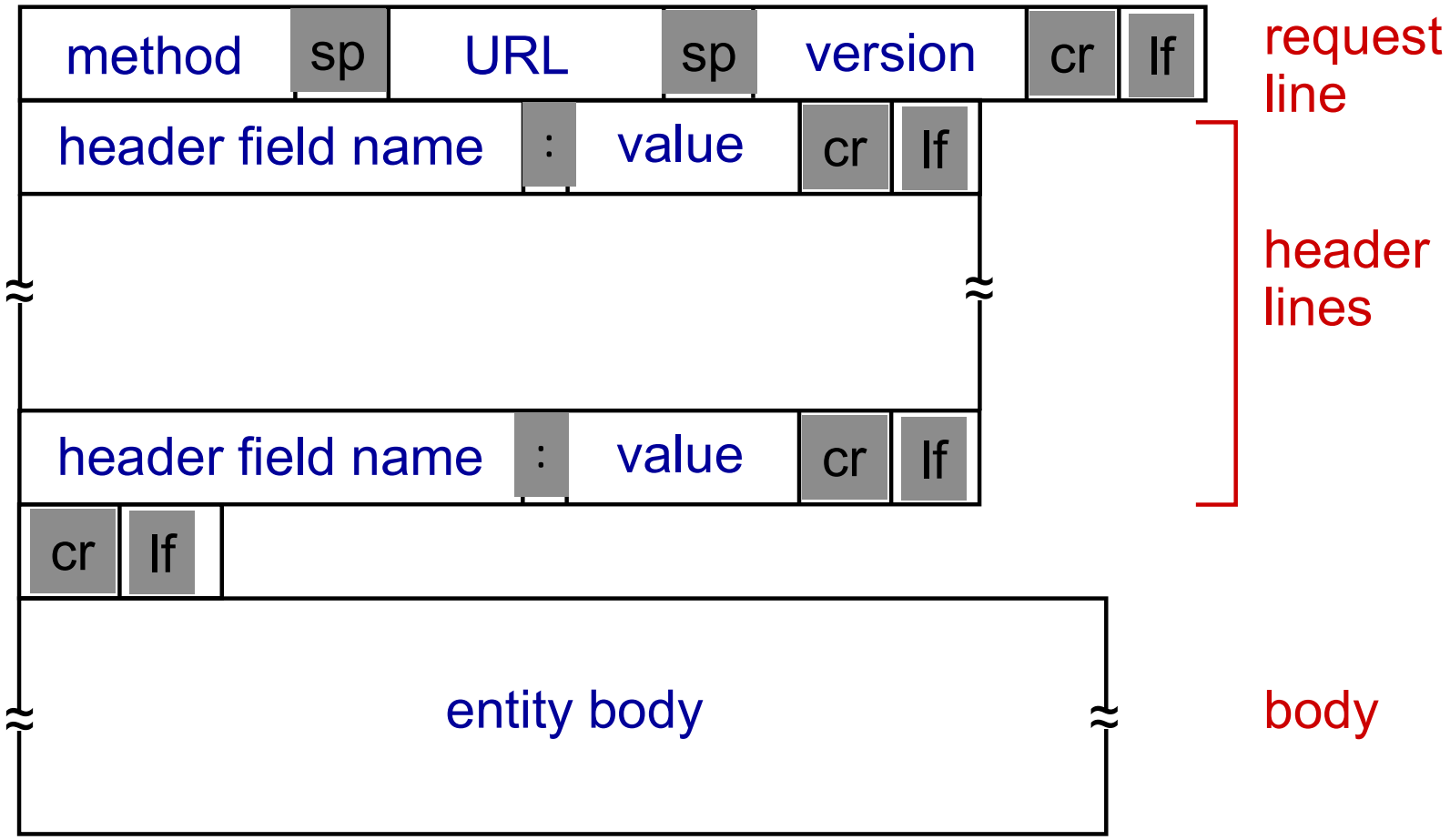


HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP request message: general format



Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

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 msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Complete Student Experience Feedback

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

User-server state: cookies

Many Web sites use cookies

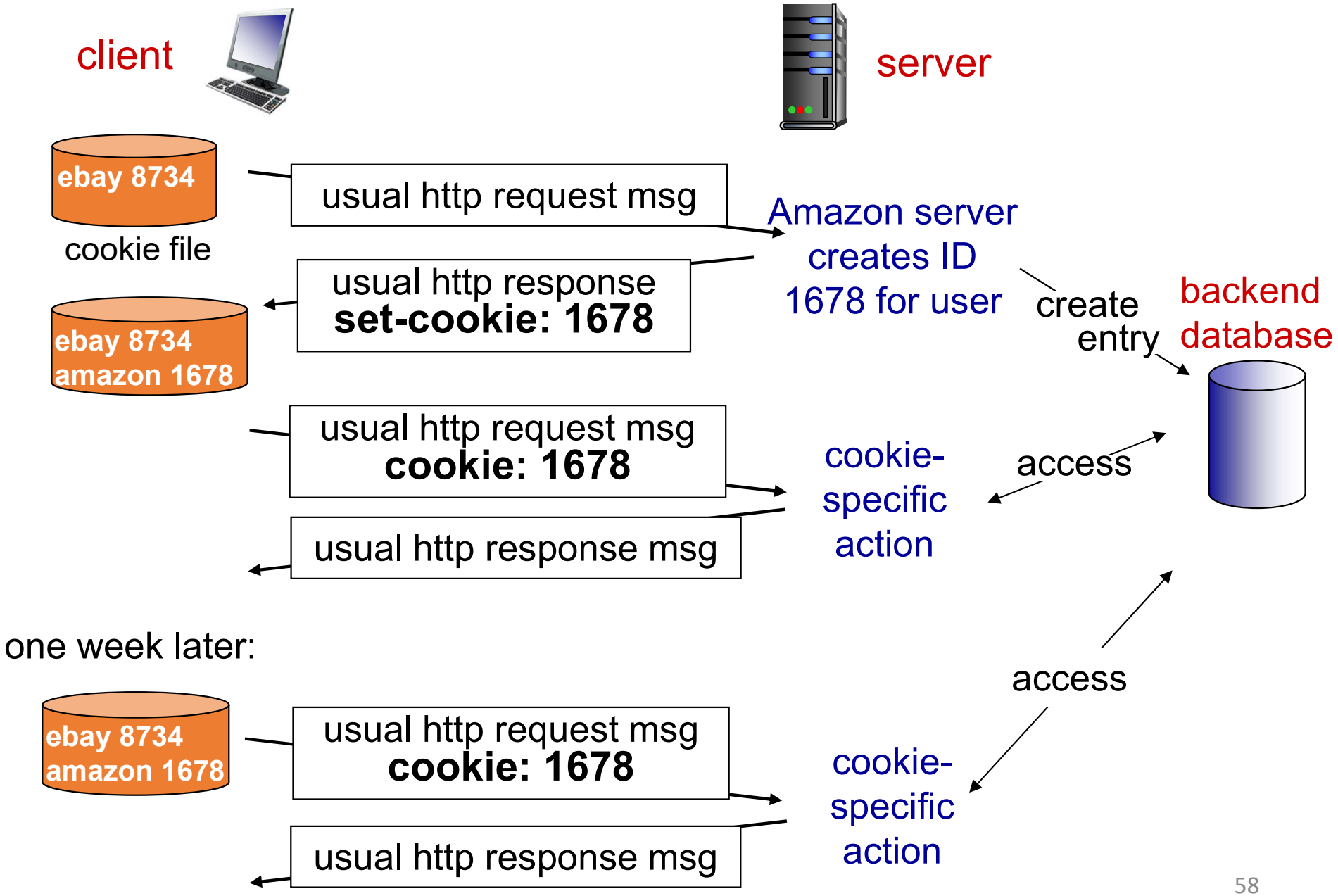
four components:

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 Website

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

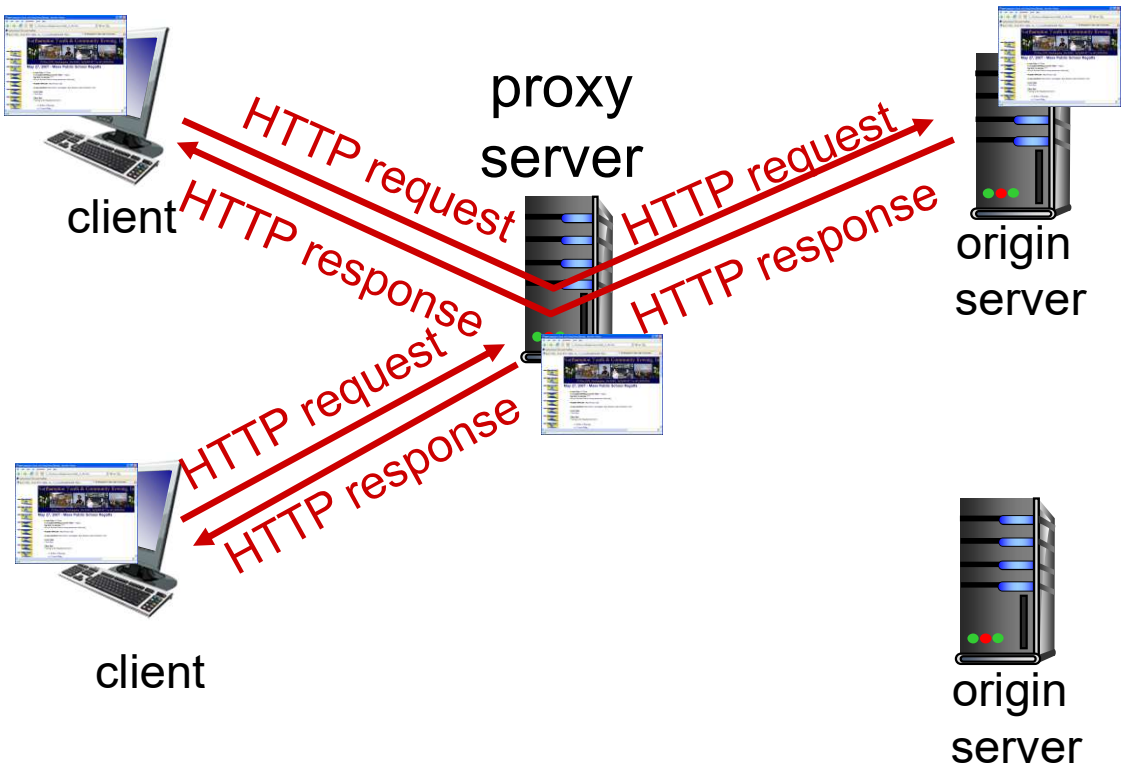
how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content

Outline

- ✓ Principles of network applications
- ✓ Web and HTTP
- ❑ Electronic Mail (SMTP, POP3, IMAP)
- ❑ DNS

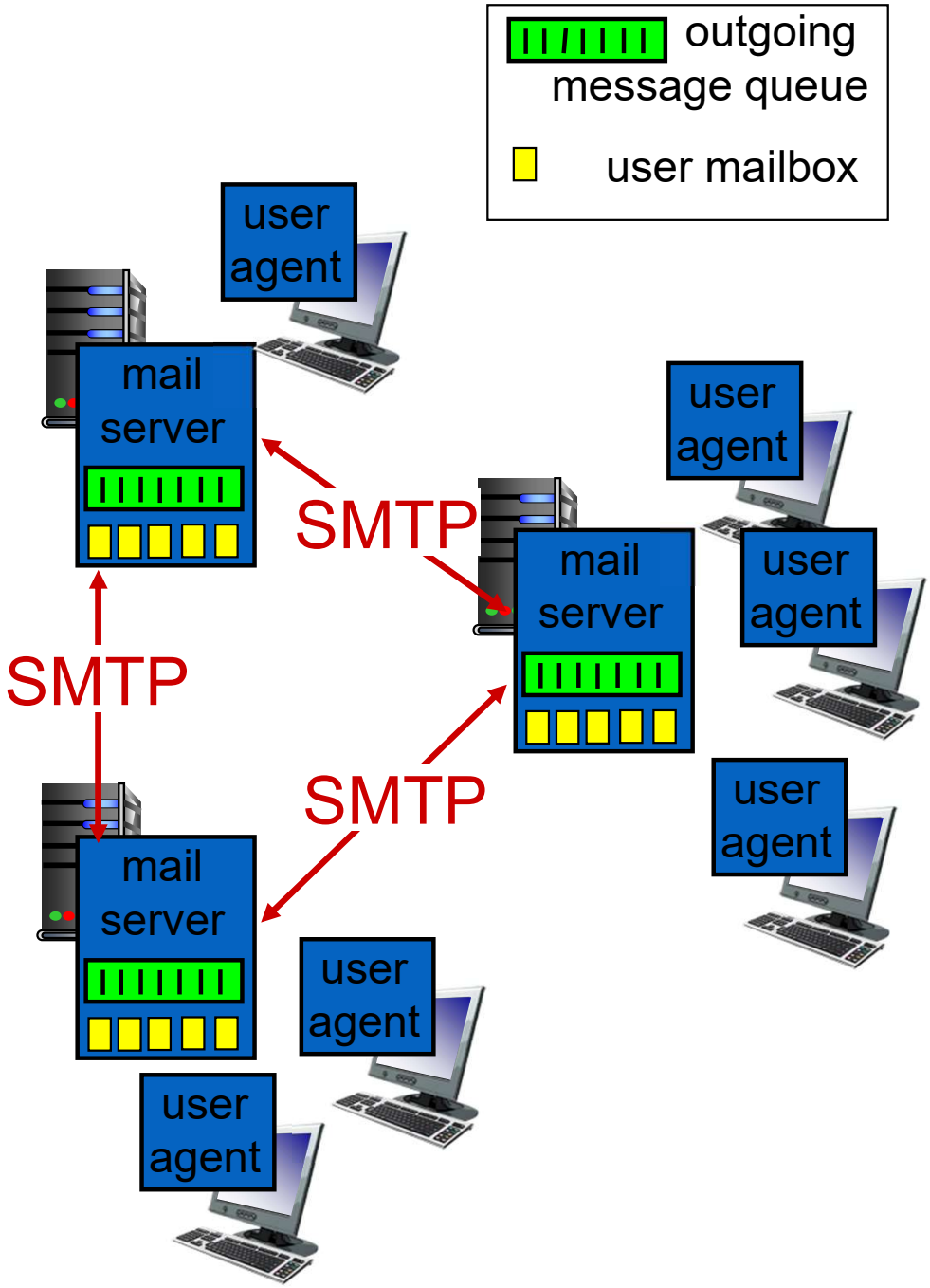
Electronic mail

Three major components:

- user agents
- mail servers
- SMTP: Simple Mail Transfer Protocol

User Agent

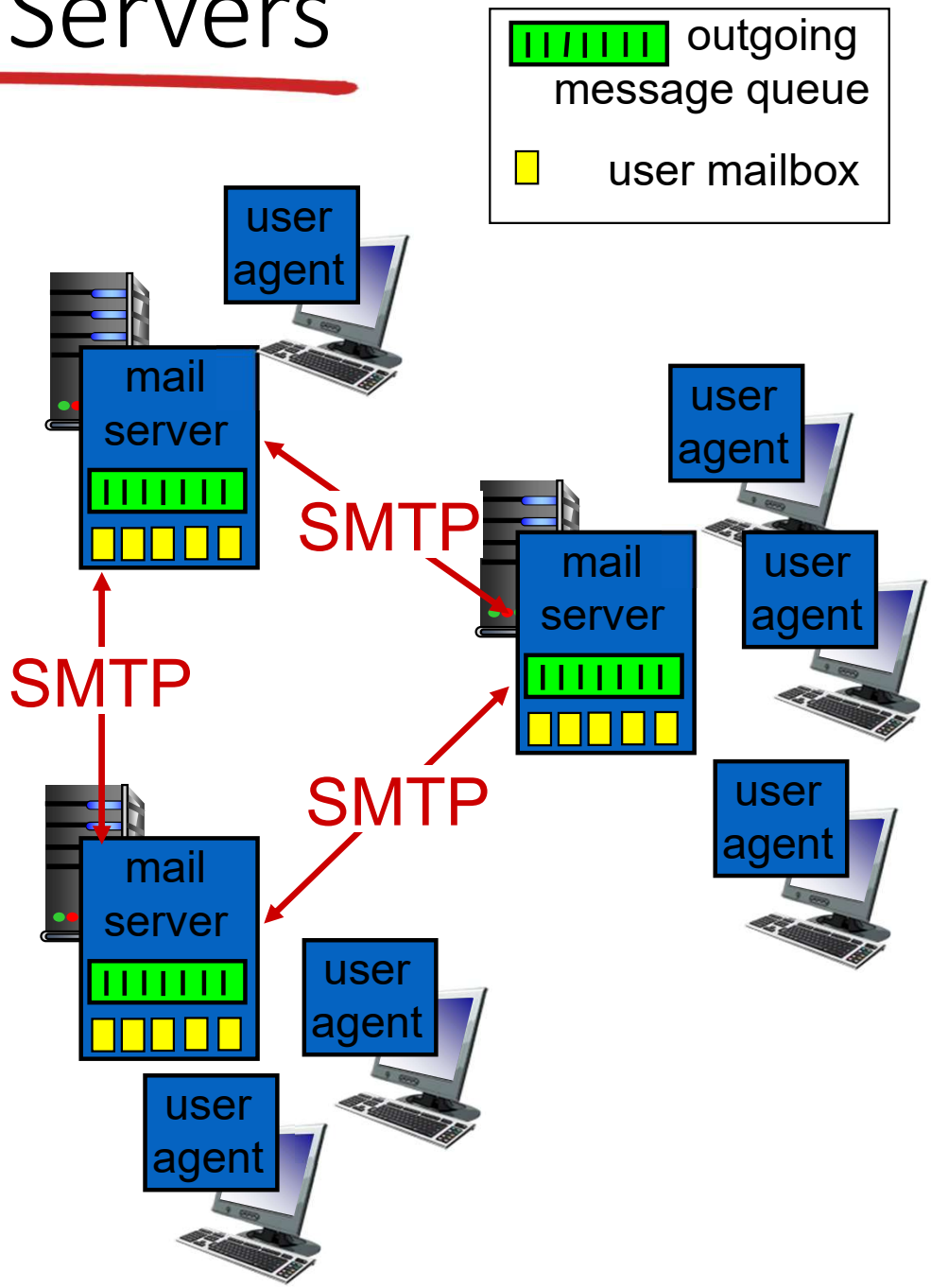
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic 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



Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

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

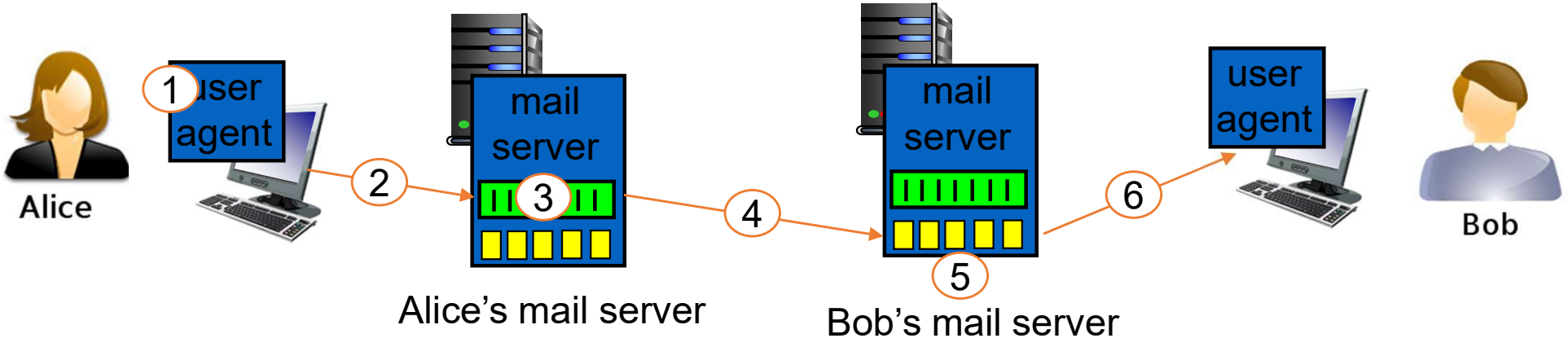
2) Alice's UA sends message to her mail server; message placed in message queue

3) client side of SMTP 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



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: final words

- 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

comparison with HTTP:

- HTTP: pull
- SMTP: 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

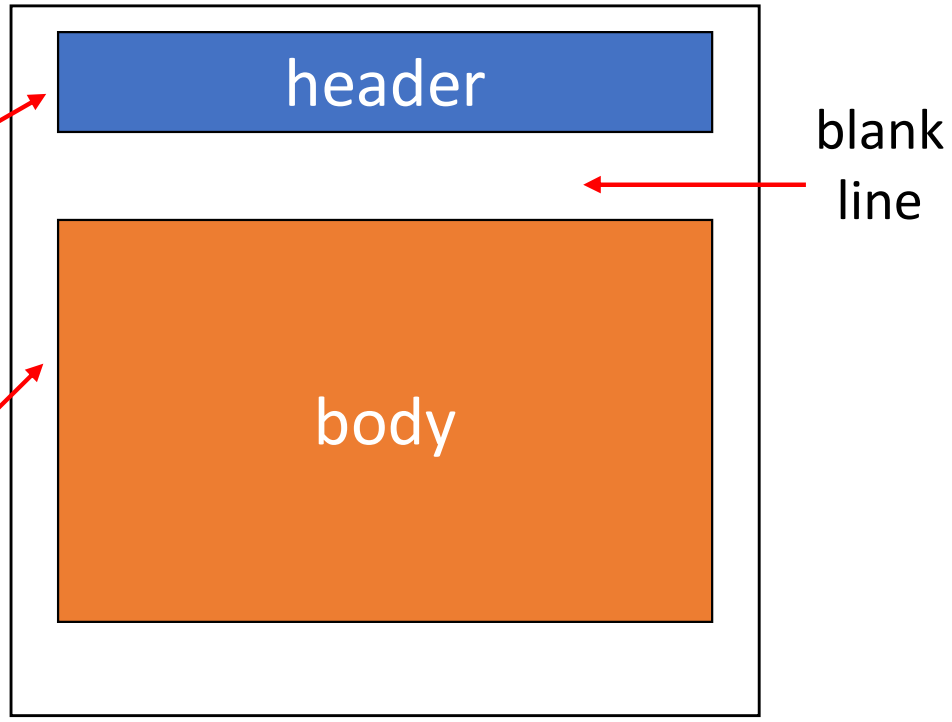
Mail message format

SMTP: protocol for exchanging email messages

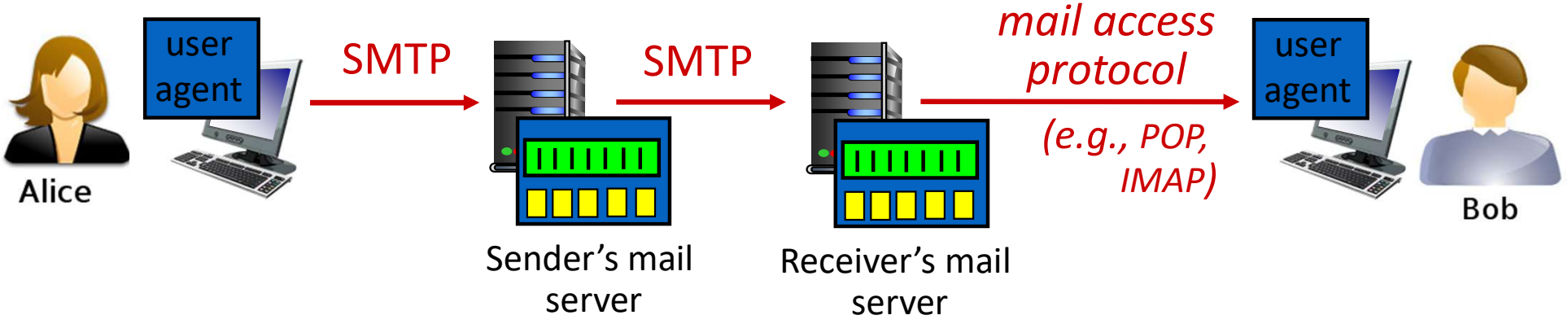
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



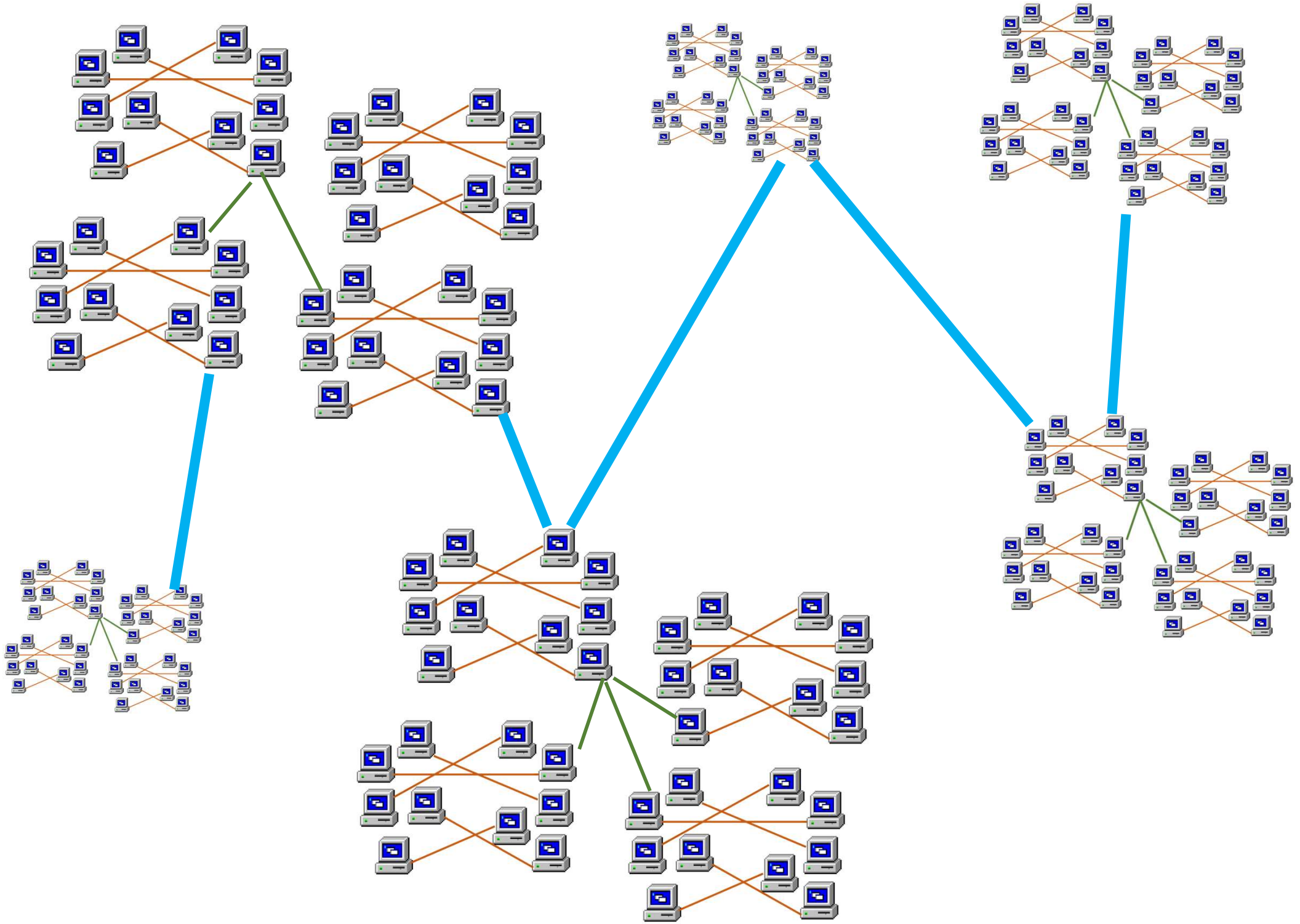
Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Outline

- ✓ Principles of network applications
- ✓ Web and HTTP
- ✓ Electronic Mail (SMTP, POP3, IMAP)
- ❑ DNS



What’s the Internet?