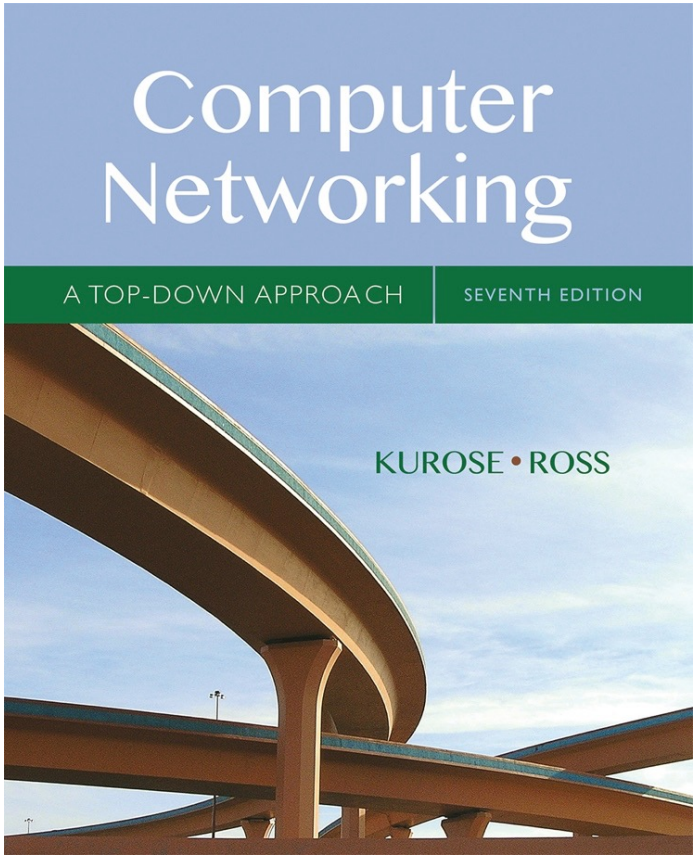
Chapter 2
Application Layer



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.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Administrivia

- Tentative Quiz/Exam Dates
  - Sep 30, Quiz 2
  - Oct 18, Midterm
  - Nov 8, Quiz 3
  - Nov 22, Quiz 4
- Homework 2 on Wireshark Basics
  - Due on Sep 16
  - No late submissions accepted

Chapter 2: application layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
  - content distribution networks
- learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

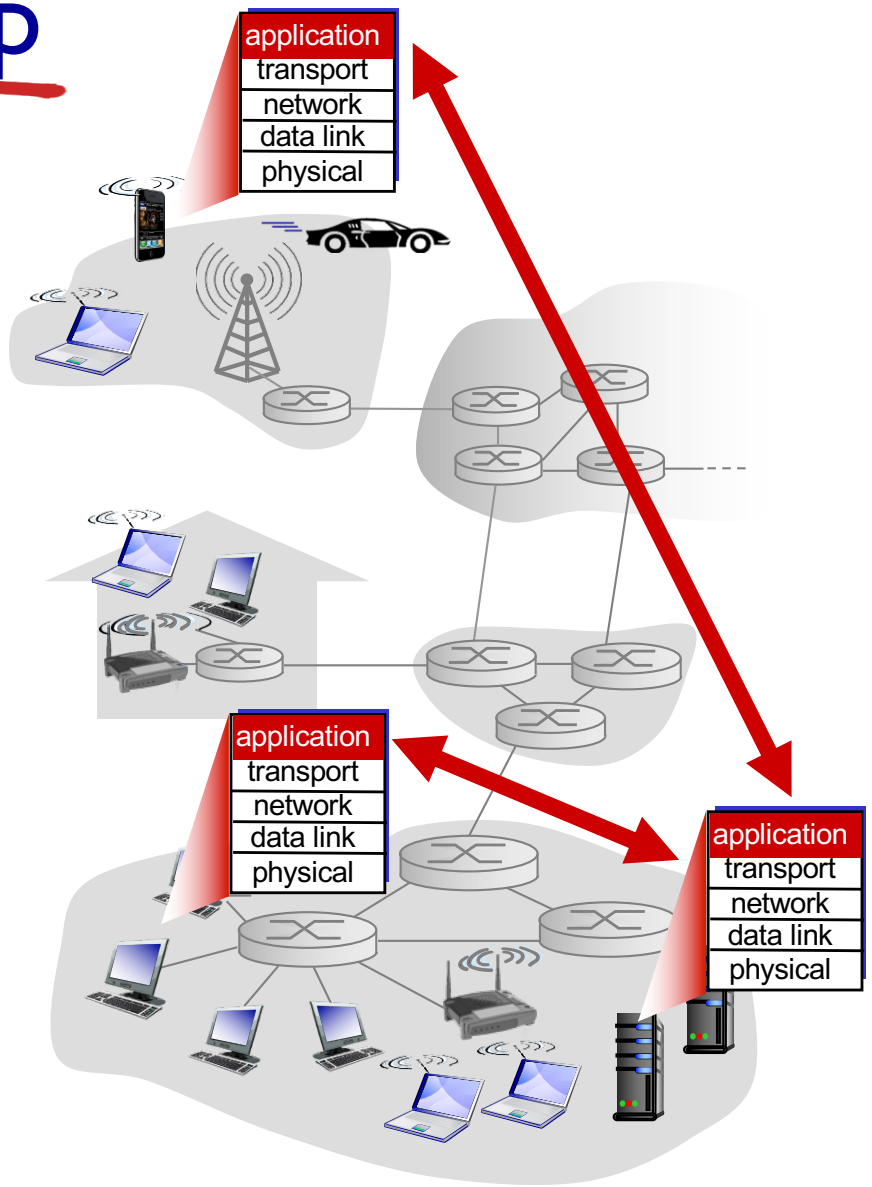
Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

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

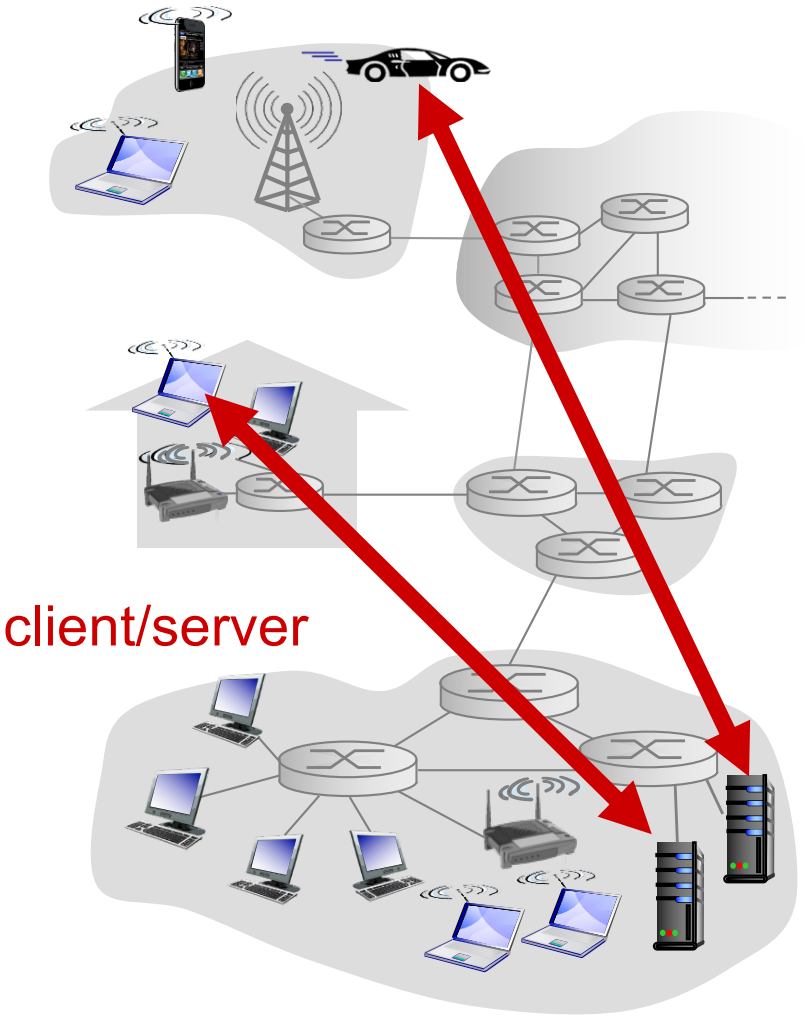
- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

Application architectures

Possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



server:

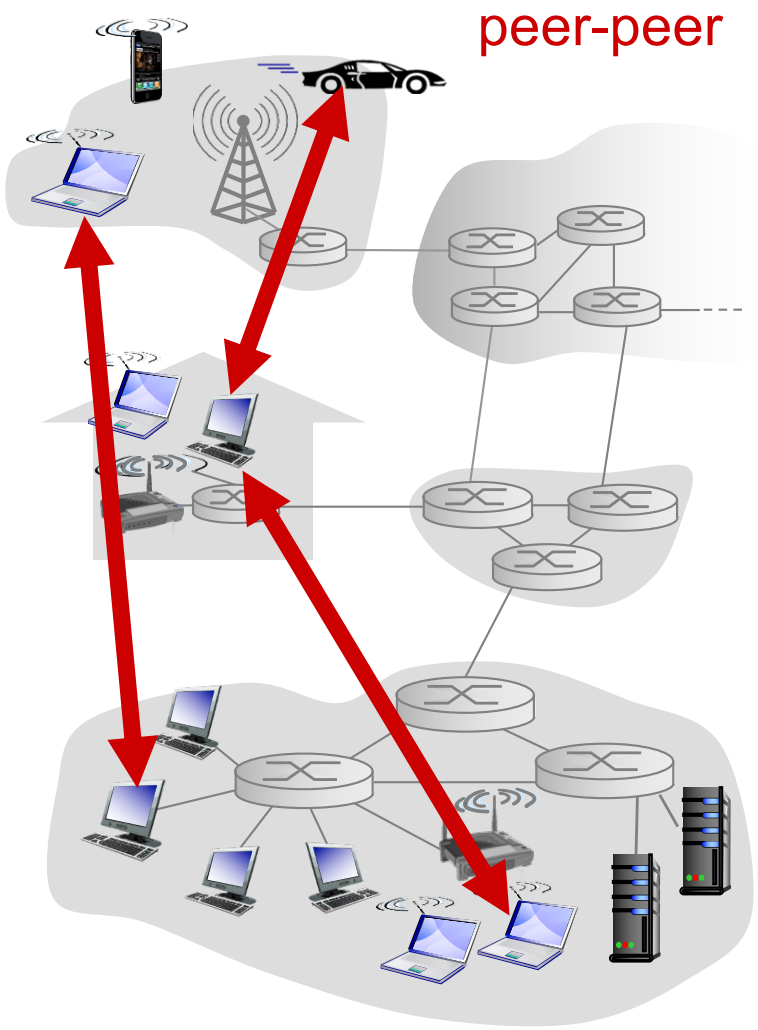
- always-on host
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P 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



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

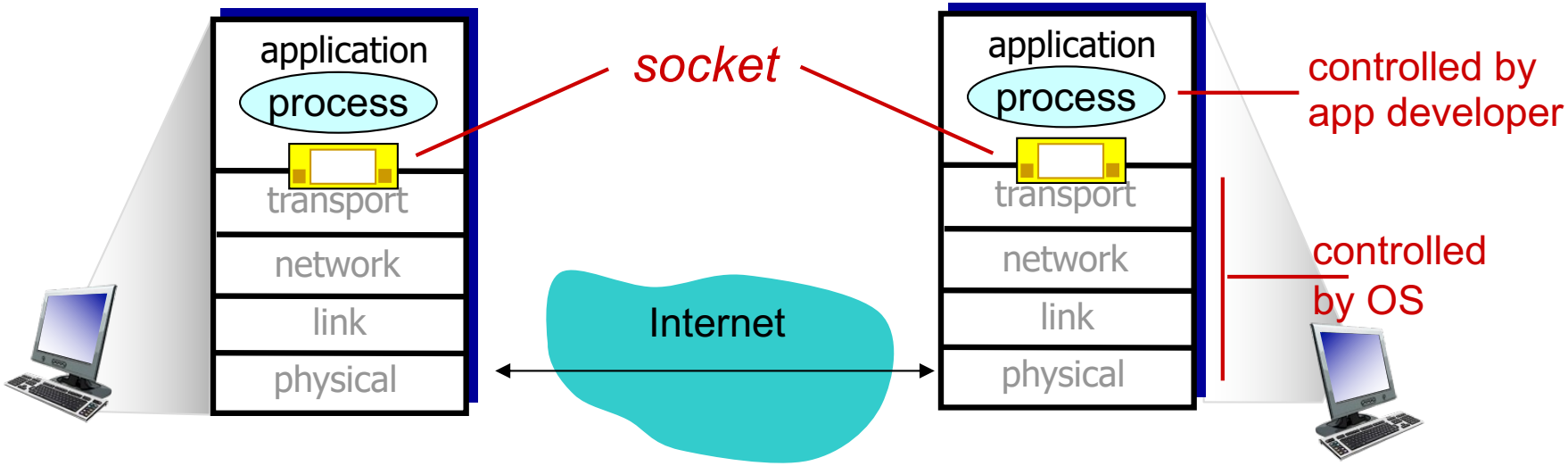
client process: process that initiates communication

server process: process that waits to be contacted

- aside: 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 outdoor
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



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 csce.uark.edu web server:
  - IP address: 130.184.76.39
  - port number: 80
- more shortly…

App-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
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

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	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, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec 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
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

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 apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Securing TCP

TCP & UDP

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext

SSL

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

SSL is at app layer

- apps use SSL libraries, that "talk" to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted

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 (continued)

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

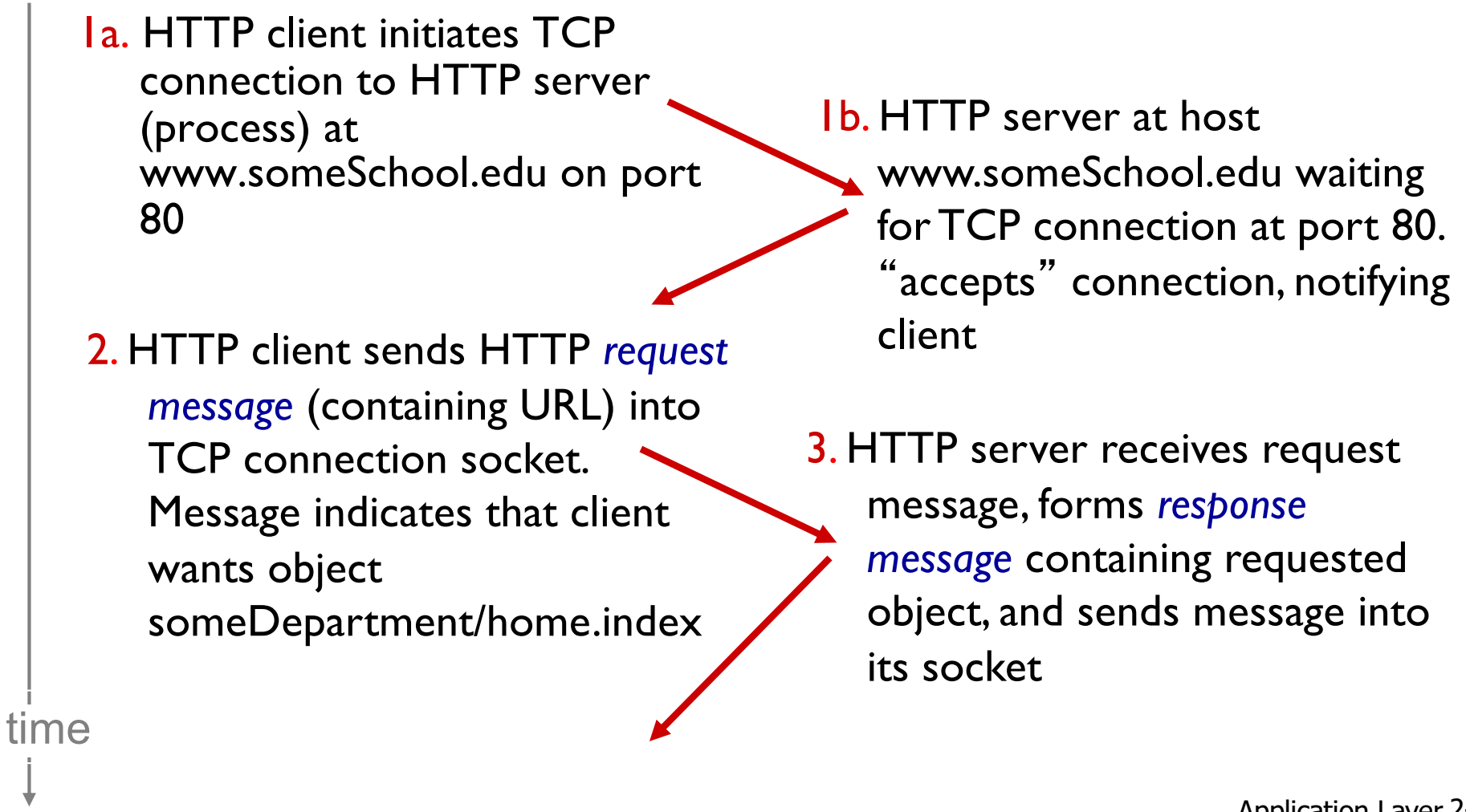
persistent HTTP

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

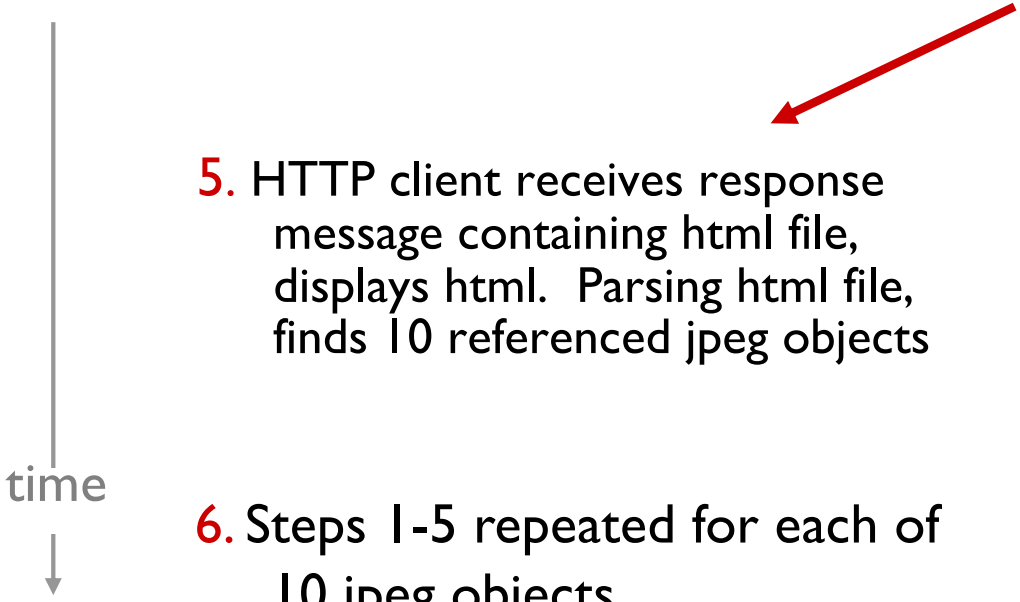
Non-persistent HTTP

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

(contains text, references to 10 jpeg images)



Non-persistent HTTP (cont.)



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

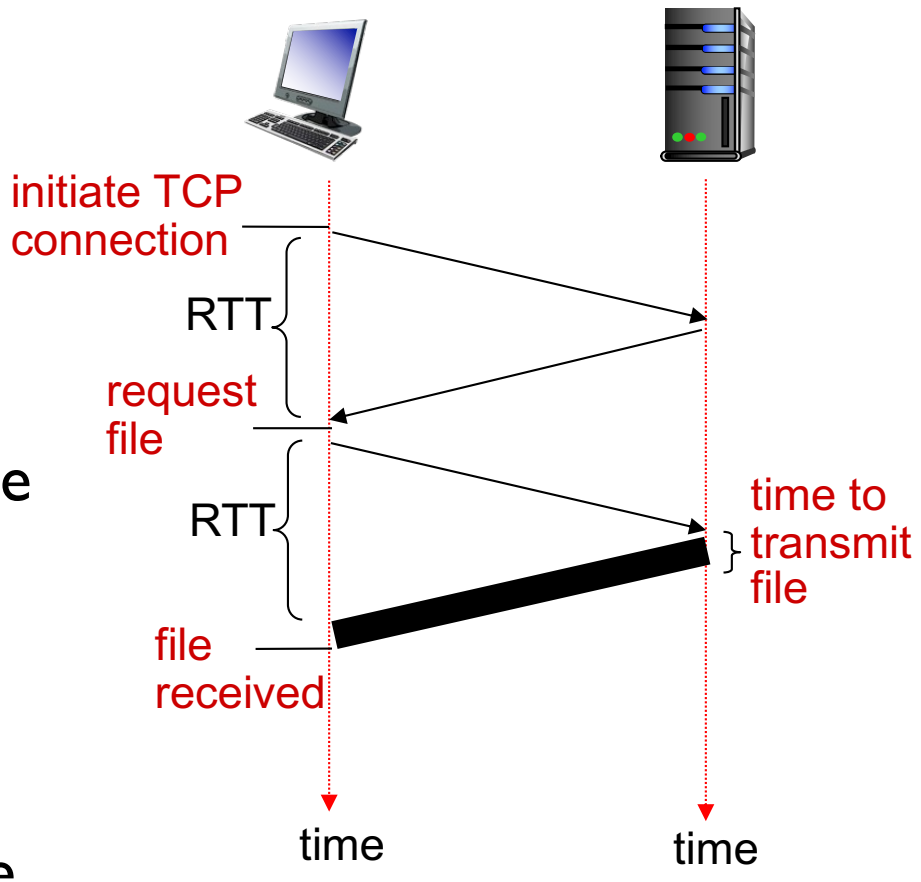
Non-persistent HTTP: response time

RTT- Round Trip Time (definition):
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

non-persistent HTTP issues:

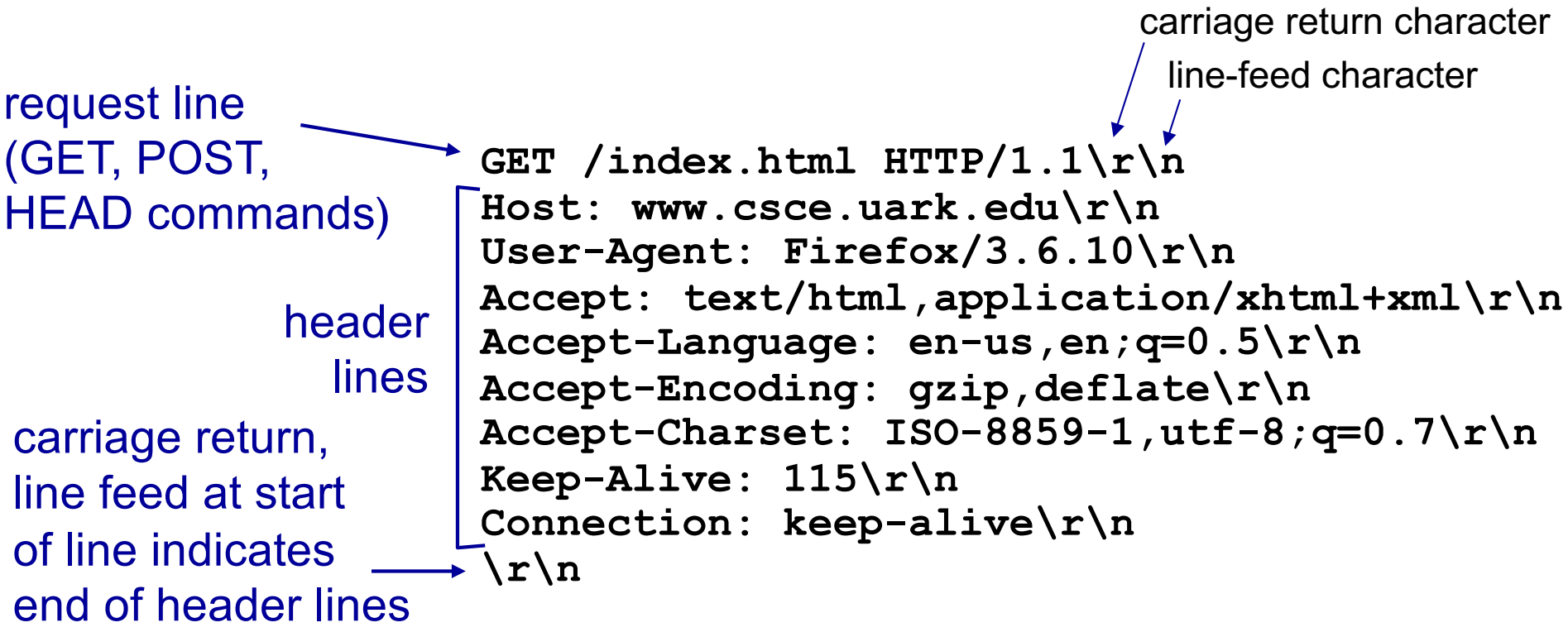
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

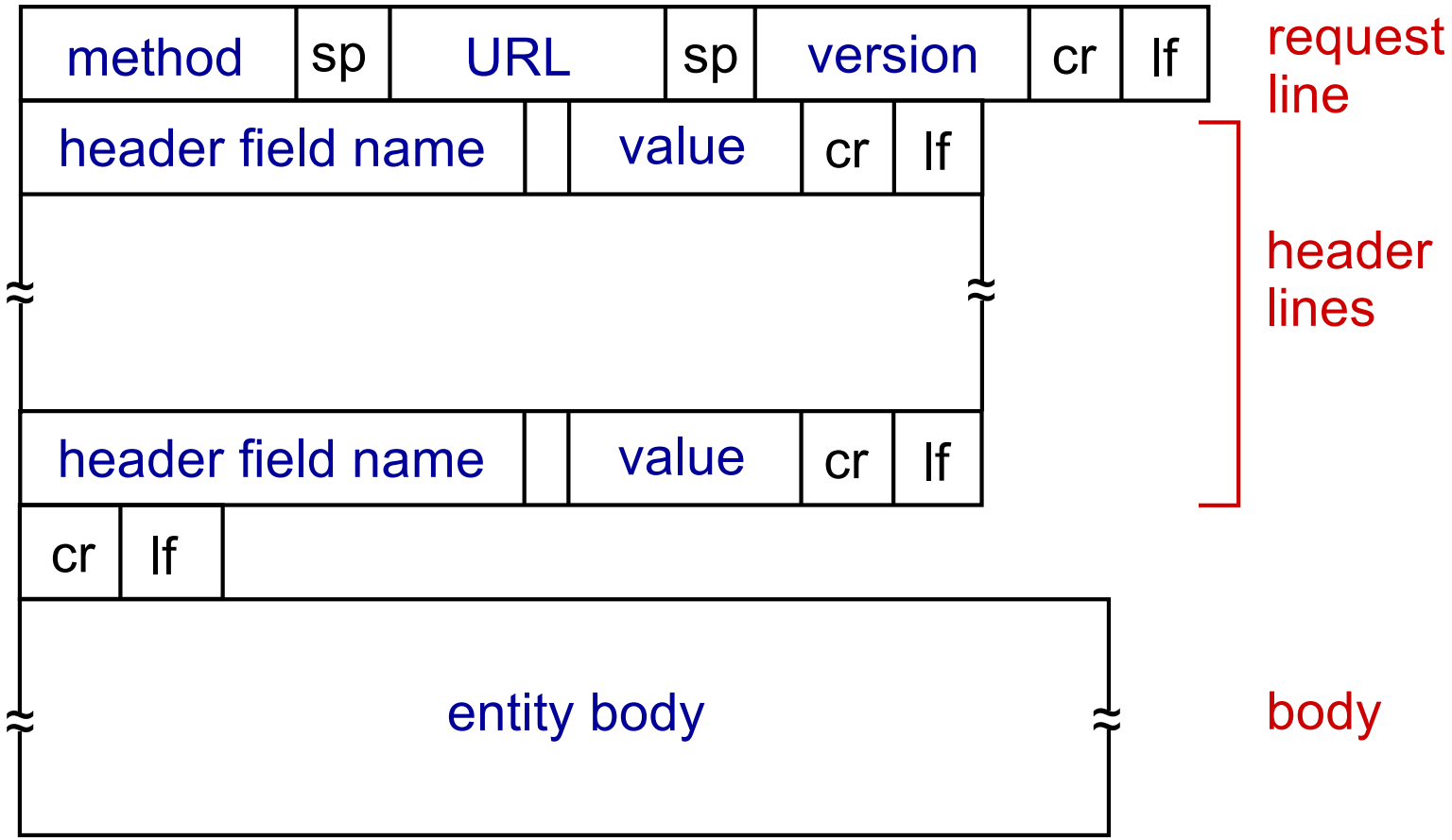
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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



HTTP request message: general format



Method types

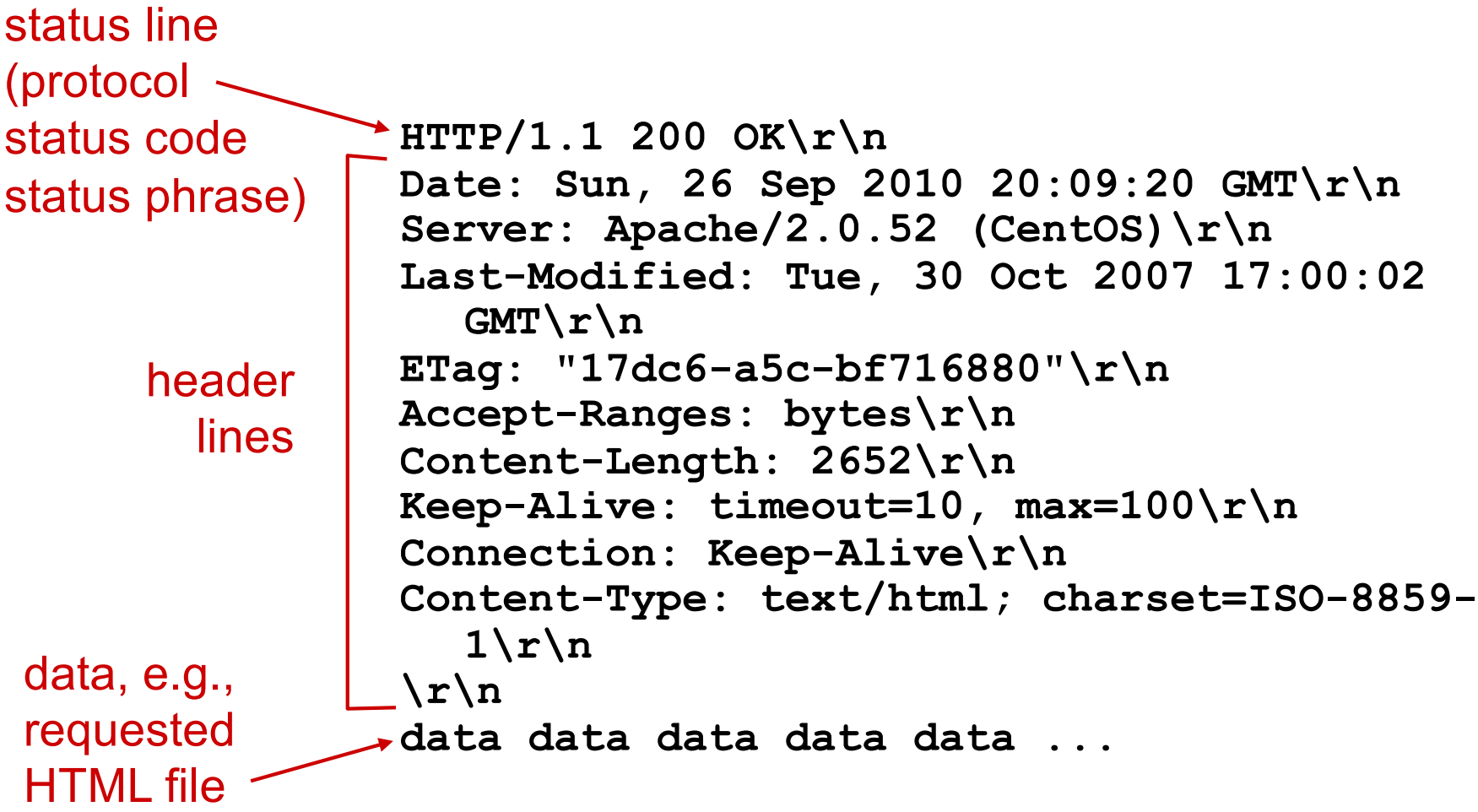
HTTP/1.0:

- 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



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

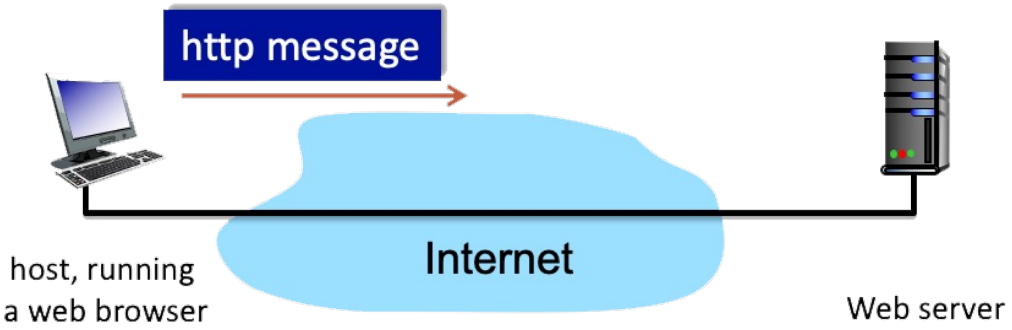
404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

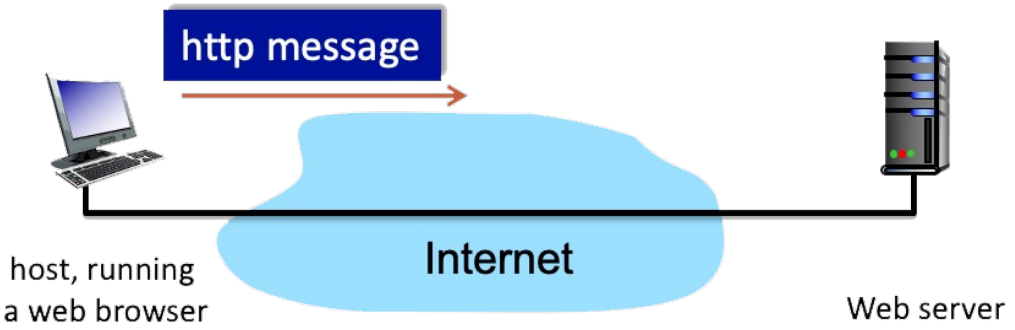
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q1: What is the name of the file that is being retrieved in this GET message?

Ans: quotation2.htm

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

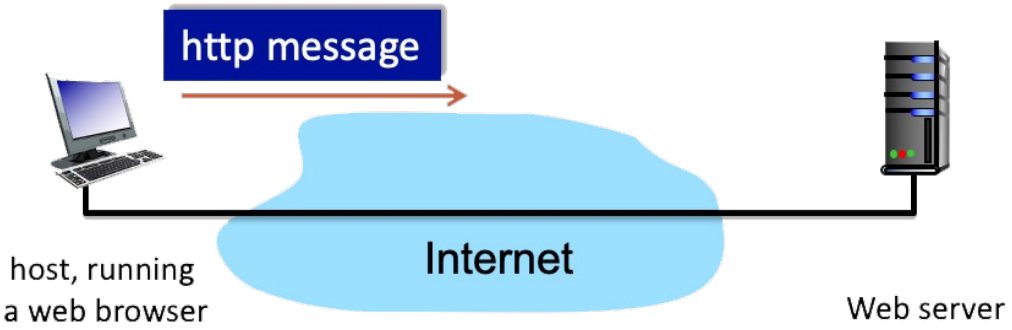
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q2: What version of HTTP is the client running?

Ans: HTTP/1.1

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

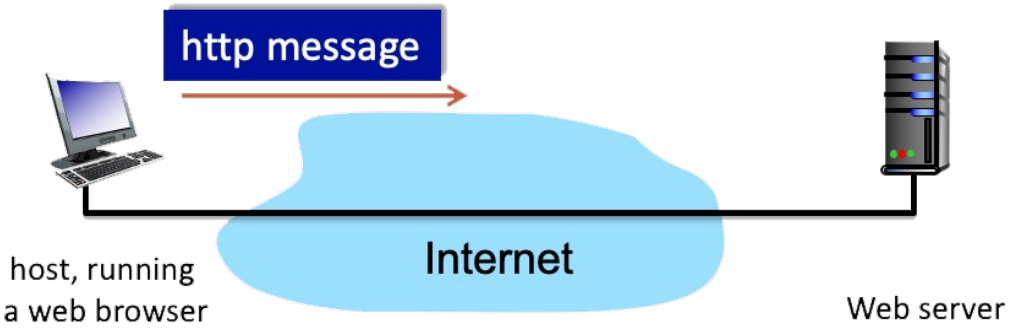
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q3: True or False: The client will accept html files.

Ans: True. In the 'Accept' field, the client includes 'text/html' files.

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

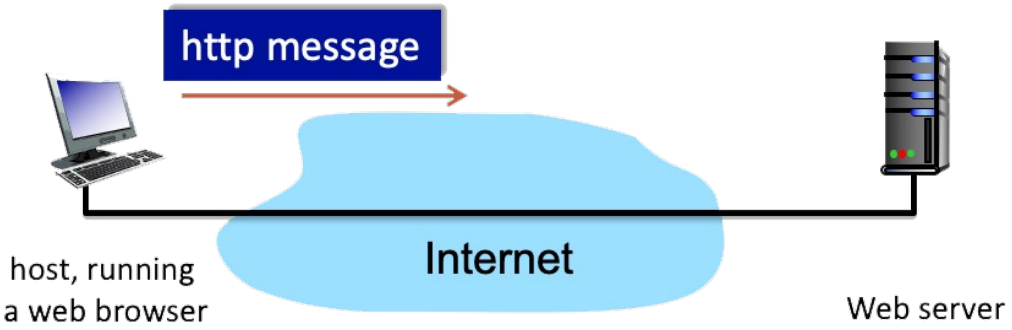
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q4: True or False: The client will accept jpeg images.

Ans: False. The client does NOT include 'image/jpeg' in its 'Accept' field.

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

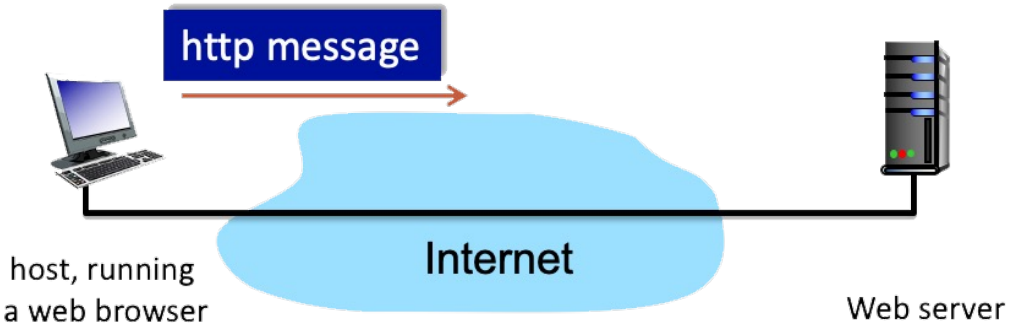
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q5: What is the client's preferred version of English?

Ans: American English.
Any language without a defined q value has a default value of 1.

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

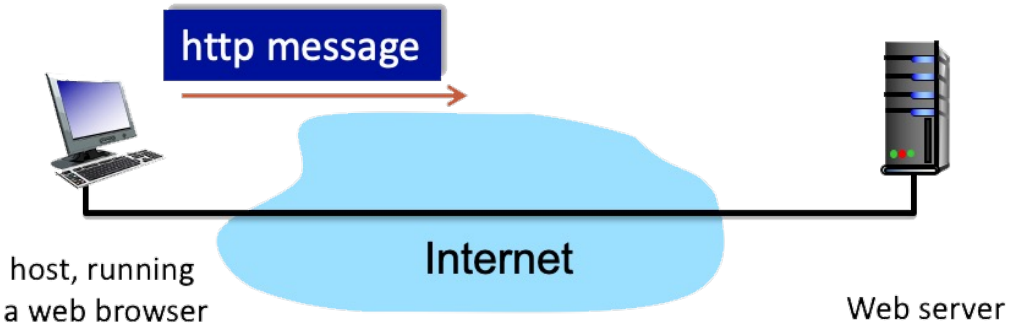
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q6: What is the client's least preferred version of English?

Ans: English, because it has the lowest q value.

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

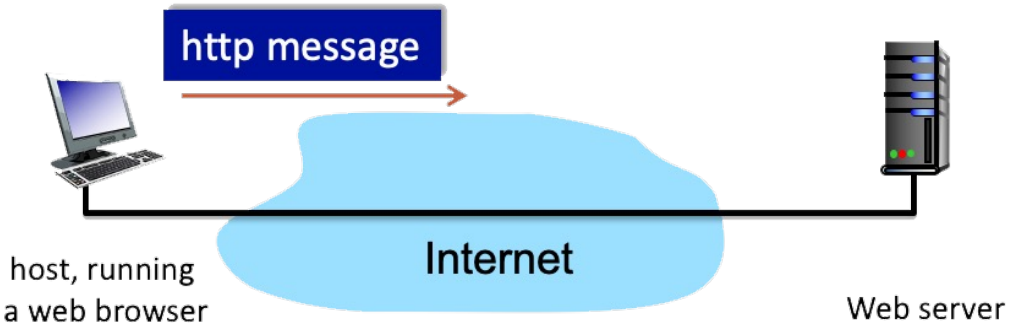
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q7: True or False: The client will accept the German language.

Ans: False. The client does NOT include German in its 'Accepted-Language' field.

Exercises: HTTP Get Message

A client is sending an HTTP GET message to a web server



GET /computernetworks/quotation2.htm HTTP/1.1

Host: test.csce.uark.edu

Accept: text/plain, text/html, image/gif, image/png, audio/basic, audio/vnf.wave, video/mp4, video/wmv,

Accept-Language: en-us, en-gb;q=0.8, en;q=0.4, fr, fr-ch, zh

If-Modified-Since: Tue, 27 Aug 2024 09:45:39 -0700

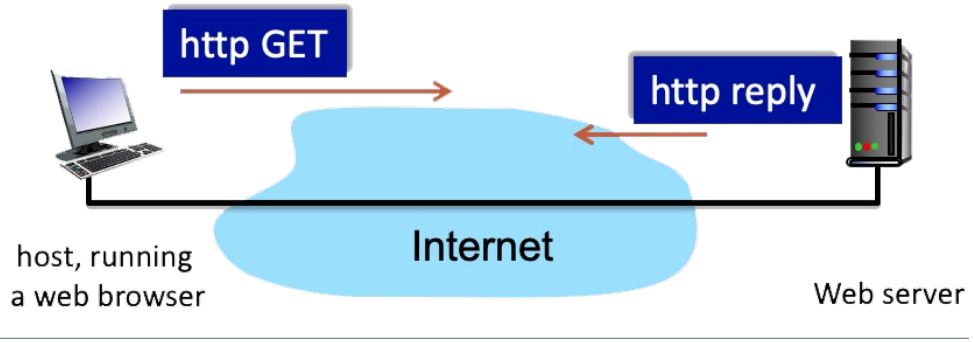
User Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)

Q8: True or False: The client already has a cached copy of the file.

Ans: True. The client has a cached copy of the file that was updated on: Tue, 27 Aug 2024 09:45:39 -0700

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

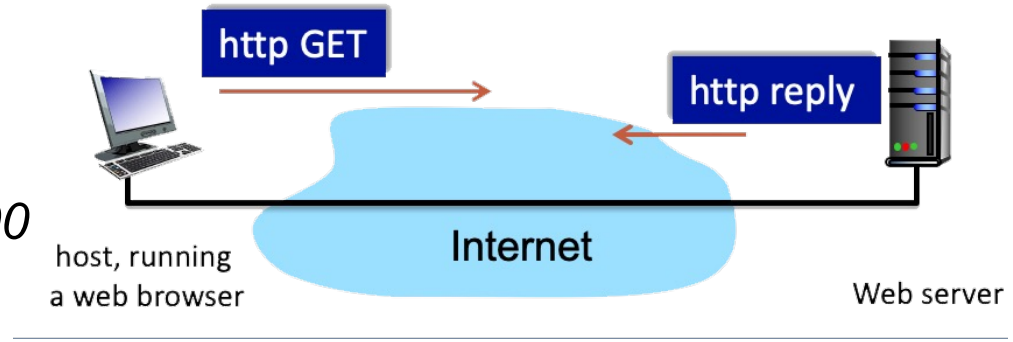
Content-type: image/html

Q1: Is the response message using HTTP 1.0 or HTTP 1.1?

Ans: HTTP/1.1

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

Content-type: image/html

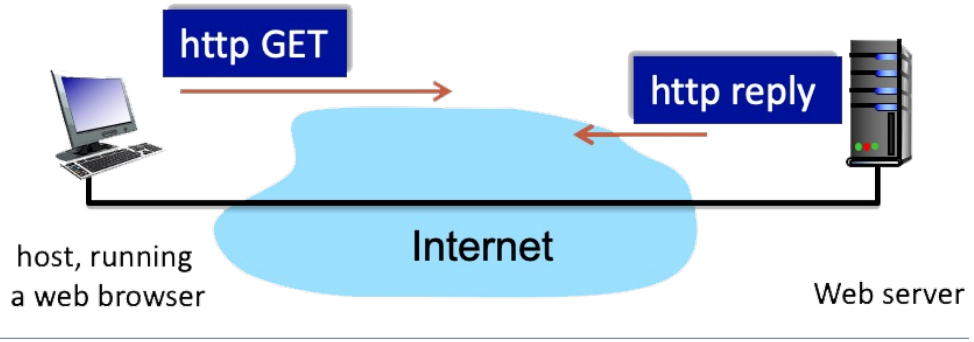
Q2: Was the server able to send the document successfully? Yes or No

Ans: Yes.

Since the response code is 200 OK, the document was received successfully.

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

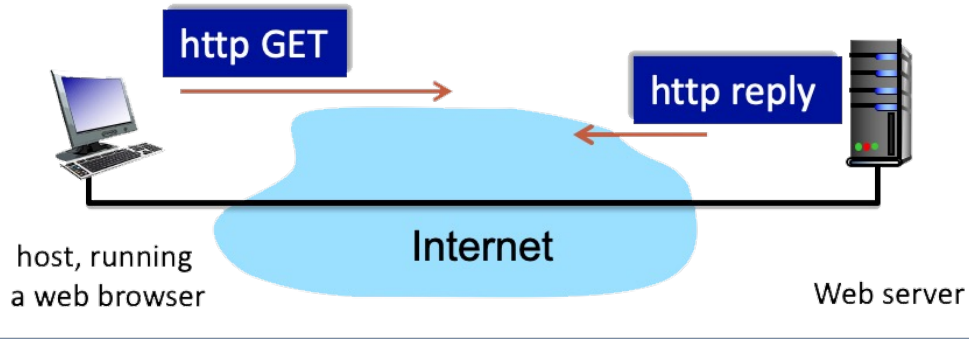
Content-type: image/html

Q3: How big is the document in bytes?

Ans: 3860 bytes

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

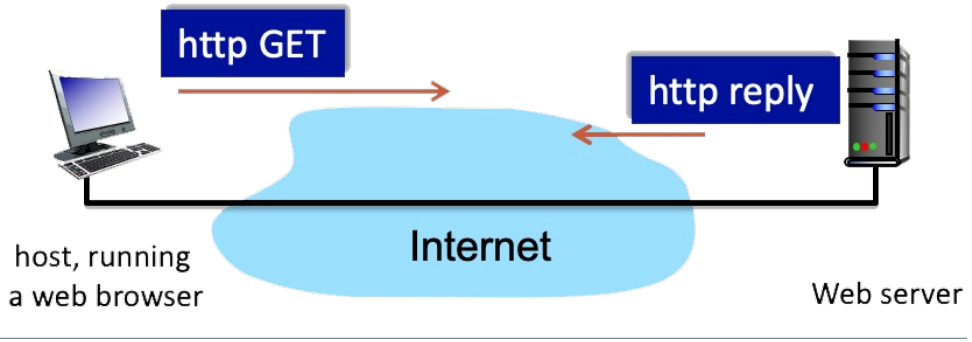
Content-type: image/html

Q4: Is the connection persistent or nonpersistent?

Ans: persistent

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

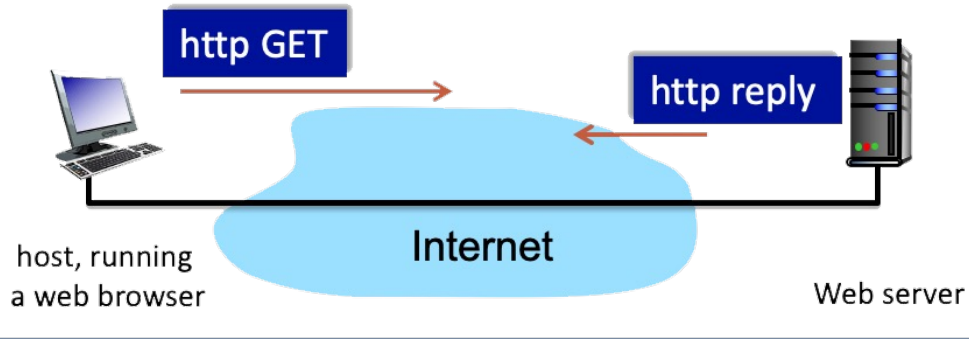
Content-type: image/html

Q5: What is the type of file being sent by the server in response?

Ans: image/html

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

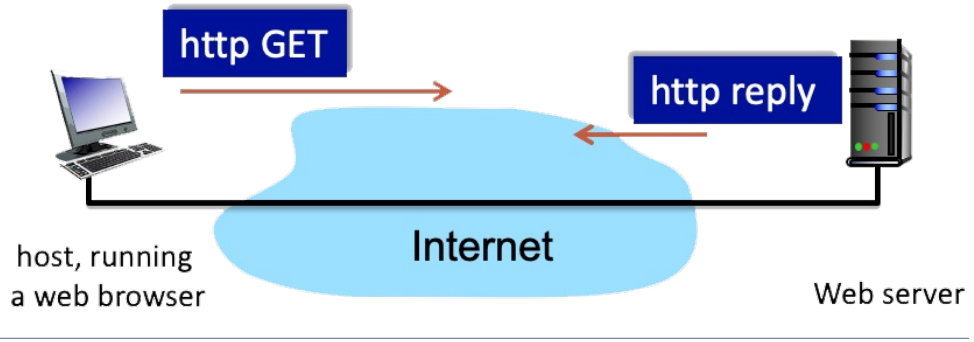
Content-type: image/html

Q6: What is the name of the server and its version?

Ans: Apache/2.2.3

Exercises: HTTP RESPONSE

A server is sending an HTTP RESPONSE message back to a client



HTTP/1.1 200 OK

Date: Tue, 27 Aug 2024 17:00:09 +0000

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 27 Aug 2024 17:40:09 +0000

ETag:17dc6-a5c-bf716880.

Content-Length: 3860

Keep-Alive: timeout=23, max=93

Connection: Keep-alive

Content-type: image/html

Q7: Will the ETag change if the resource content at this particular resource location changes? Yes or No

Ans: Yes.

The Etag is a string that uniquely identifies a resource. If a resource is updated, the Etag will change.

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
```

DEMO: 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

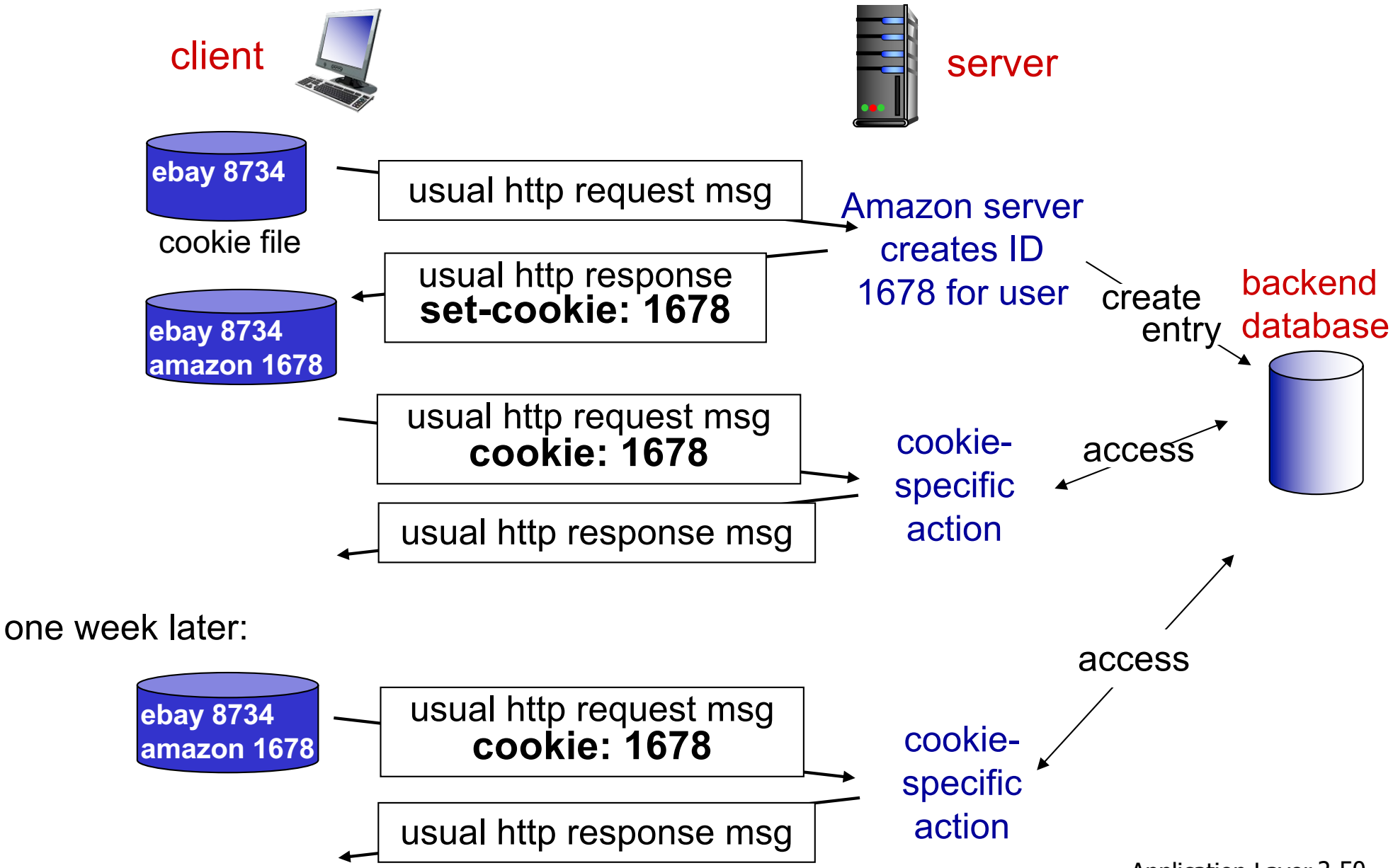
four components:

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 Web site

example:

- Susan always accesses 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

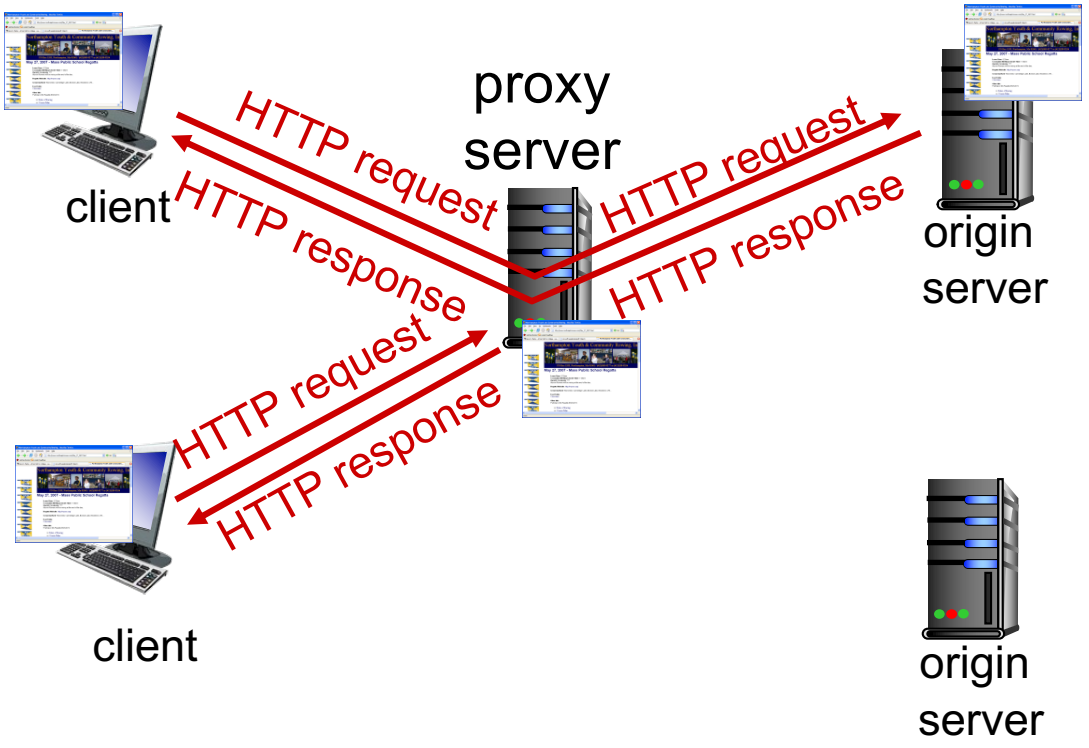
how to keep "state":

- 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 (so too does P2P file sharing)

Caching example:

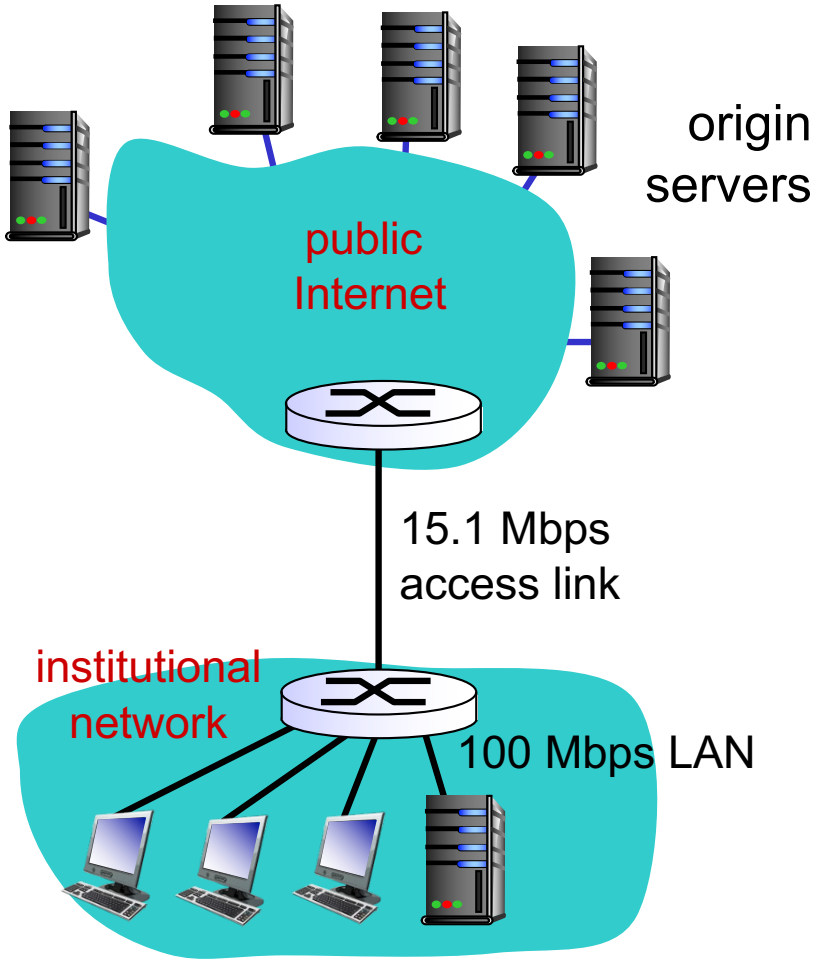
assumptions:

- avg object size: 1M bits per request
- avg request rate from browsers to origin servers: 15 requests/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15.1 Mbps

consequences:

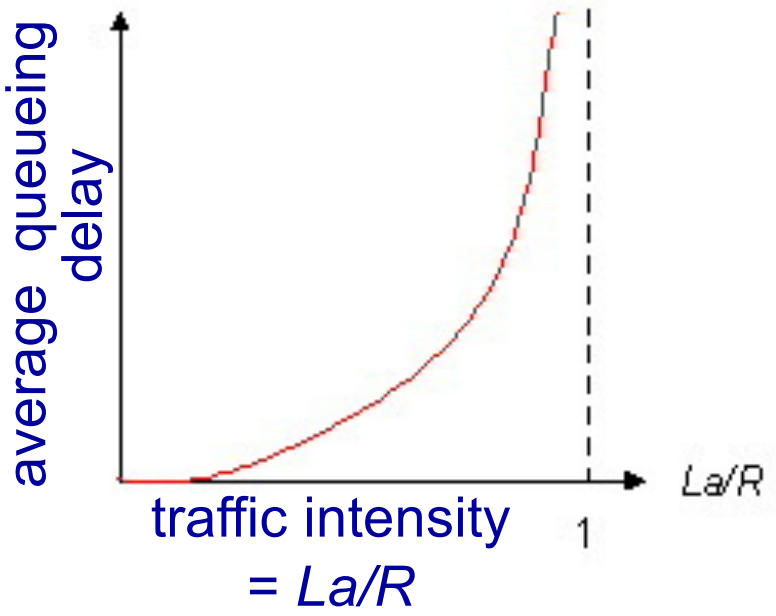
- LAN utilization: 15% *problem!*
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + msecs



Queueing delay (learned in Chapter 1)

- *R*: link bandwidth (bps)
- *L*: packet length (bits)
- a: average packet arrival rate



- *La/R* ~ 0: avg. queueing delay small
- *La/R* -> 1: avg. queueing delay large
- *La/R* > 1: more "work" arriving than can be serviced, average delay infinite!

La/R ~ 0

La/R -> 1

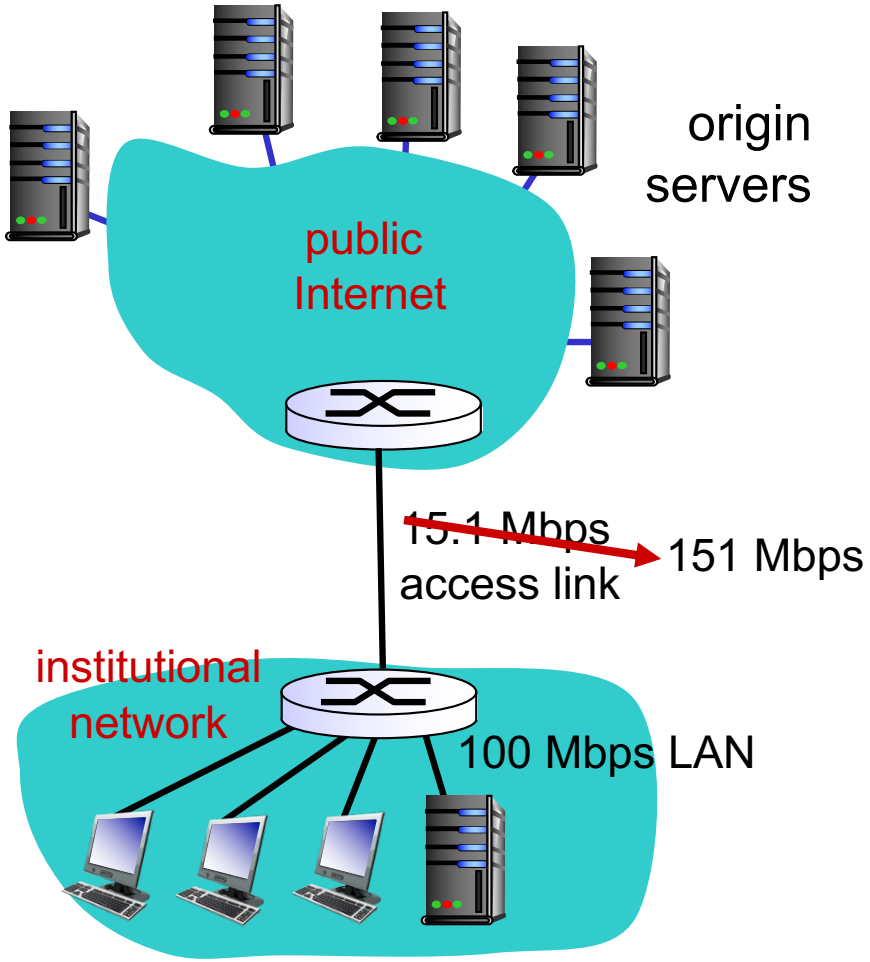
Caching example: fatter access link

assumptions:

- avg object size: 1M bits per request
- avg request rate from browsers to origin servers: 15 requests/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15.1 Mbps~~ 151 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + msecs
  msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

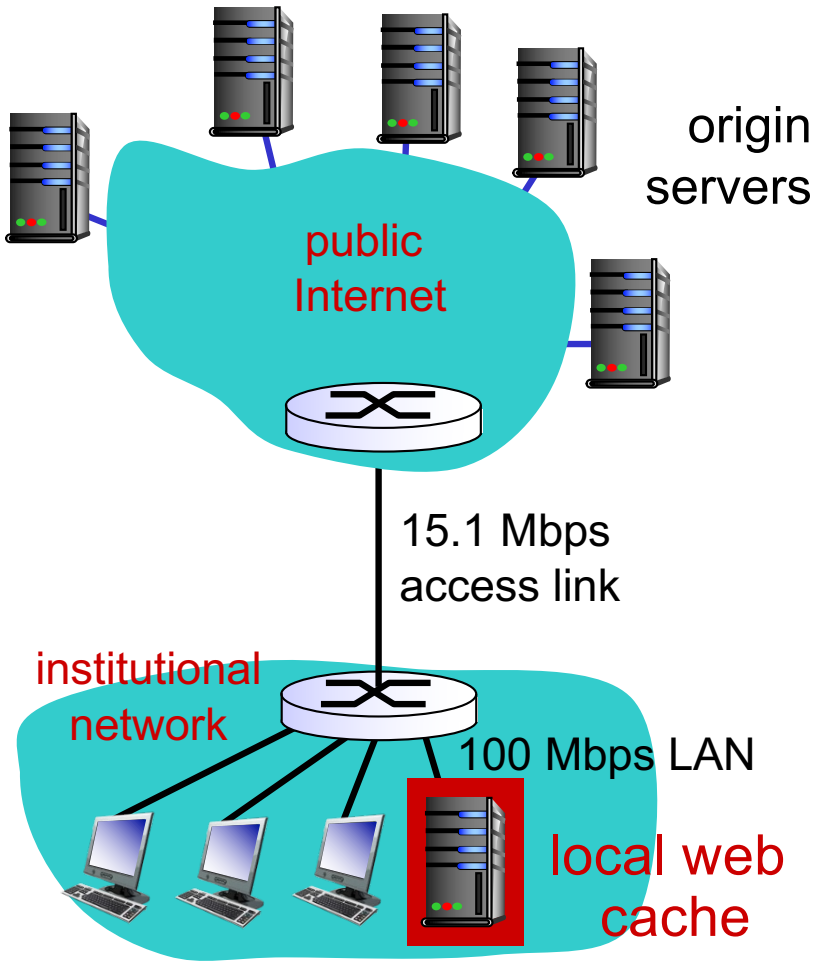
- avg object size: 1M bits per request
- avg request rate from browsers to origin servers: 15 requests/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15.1 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

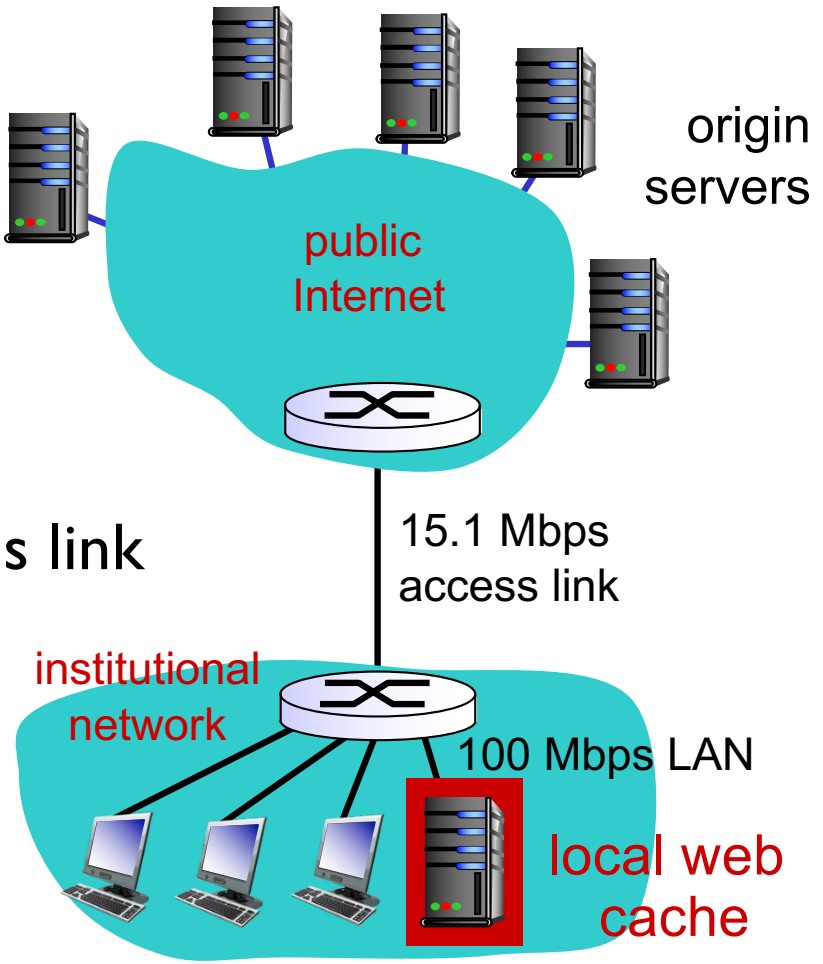
Cost: web cache (cheap!)



Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache
    60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link
  = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/15.1 = 59.6%
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6*(2.0) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 151 Mbps link
    (and cheaper too!)

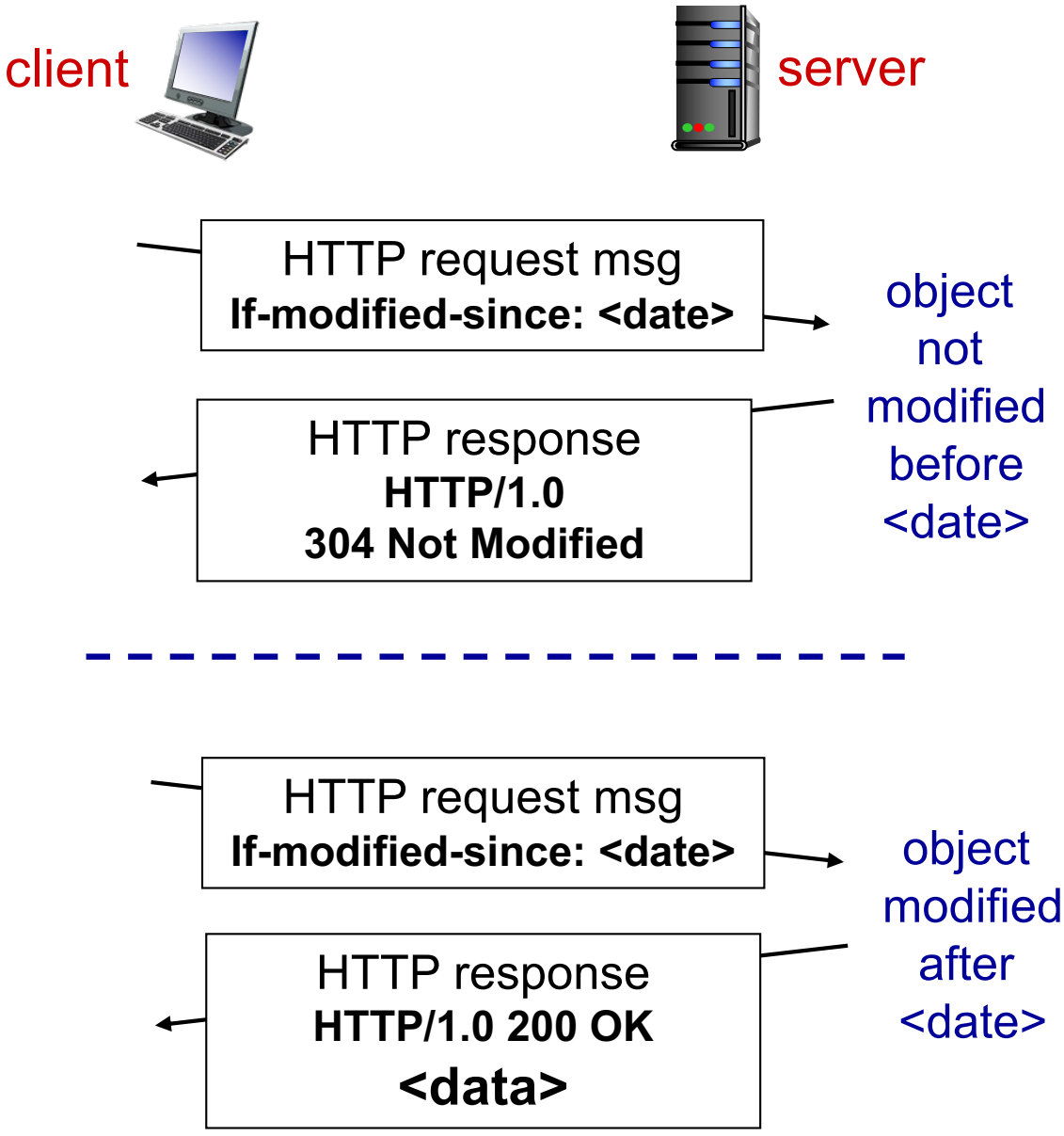


Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

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

  `HTTP/1.0 304 Not Modified`



Electronic Mail

- Origins – 1960-70s; method for users to send messages on time-sharing systems; using @
- 70s / ARPANET – exchange messages between different machines ; simple formats, no standardized formats across networks
- SMTP (80s) – first step towards standardizing email message communication across ARPANET
- 90s – present: Multimedia, Encryption, Spam prevention, etc.

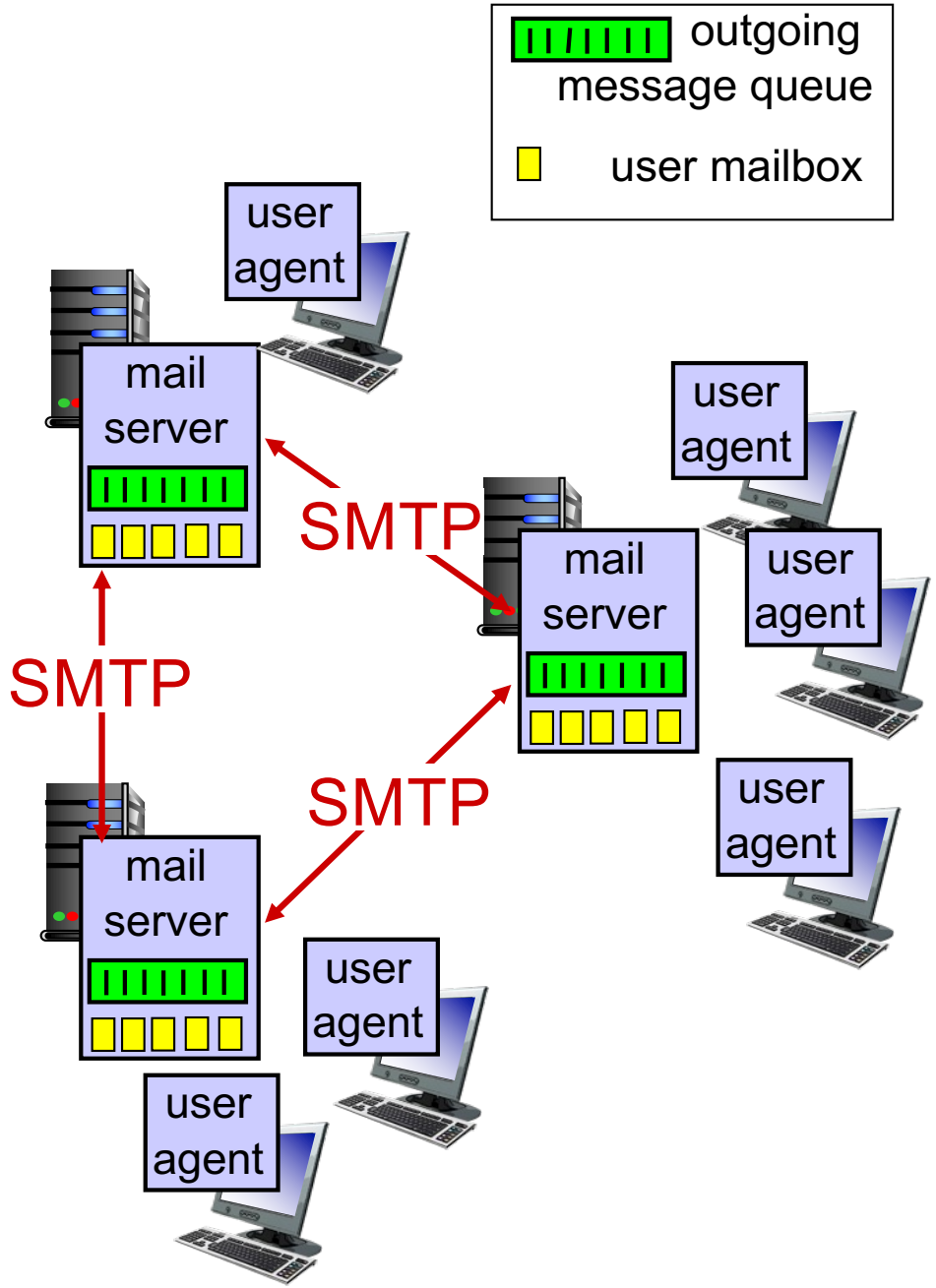
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

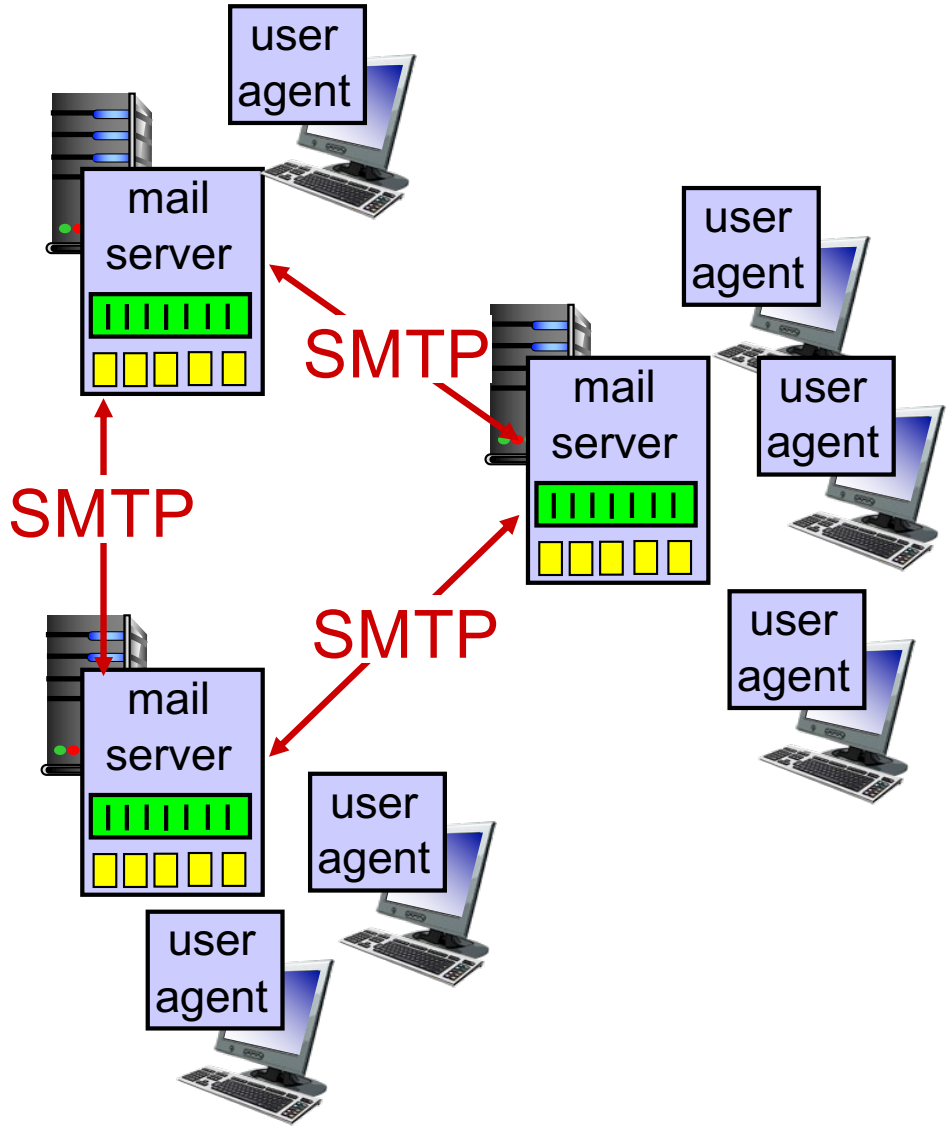
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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

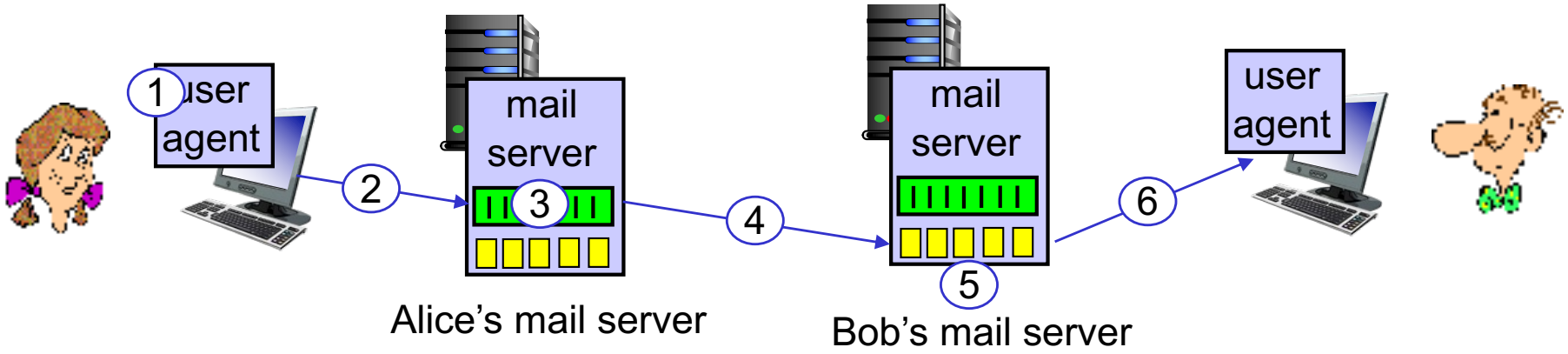


Electronic Mail: SMTP [RFC 2821]

- 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*
  - *Q: how do we send binaries?*

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

handshaking (greeting)

transfer of messages

closure

```
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
```

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

Note: Almost all current SMTP servers require Transport Layer Security (TLS) ; port 587

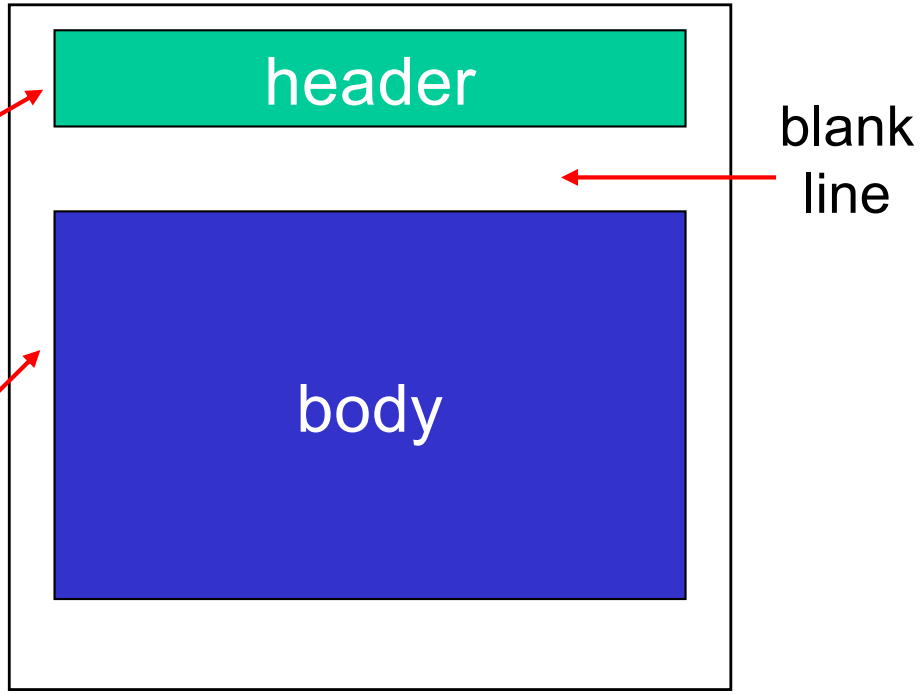
Mail message format

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



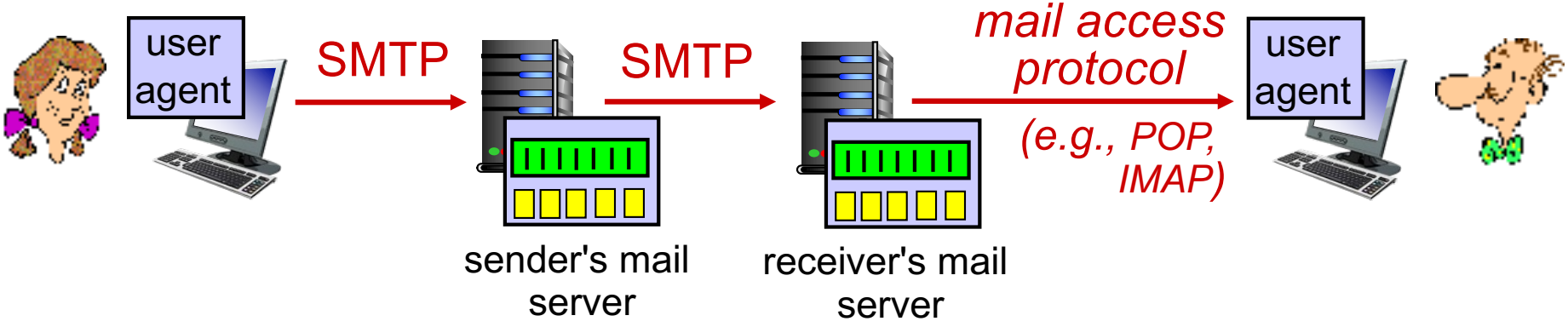
SMTP: final words

- 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 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.

POP3 protocol

authorization phase

- client commands:
  - `user:` declare username
  - `pass:` password
- server responses
  - `+OK`
  - `-ERR`

transaction phase, client:

- `list:` list message numbers
- `retr:` retrieve message by number
- `dele:` delete
- `quit`

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

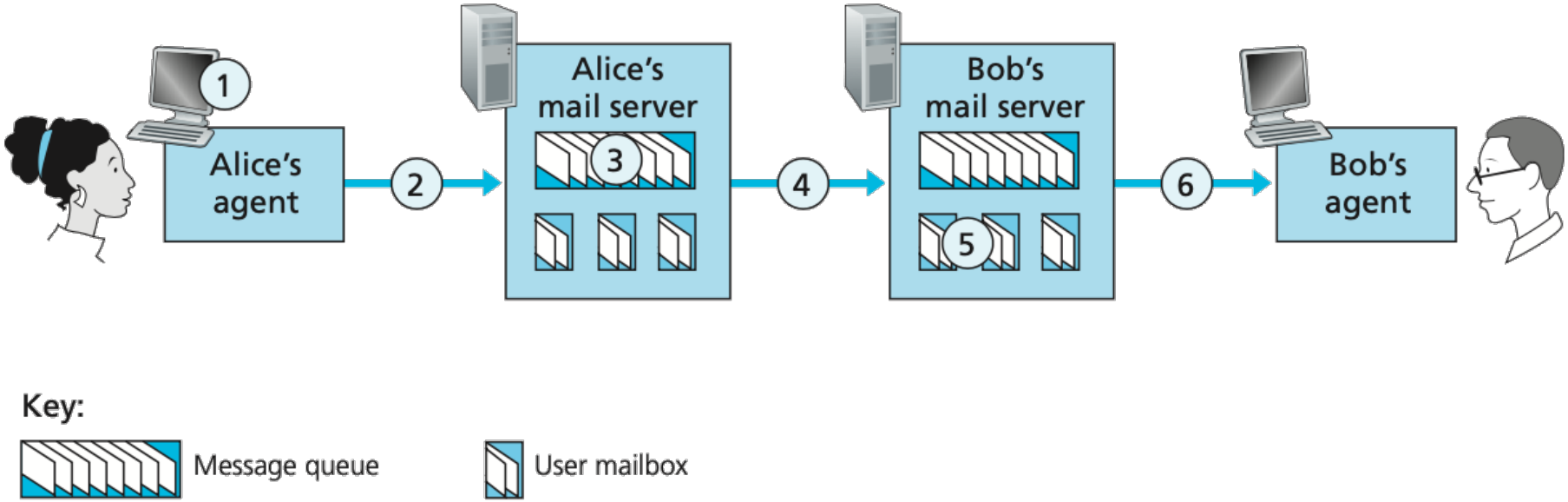
- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

Need only a **local** copy while deleting from server

Access and Manage your email from **Multiple** devices

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

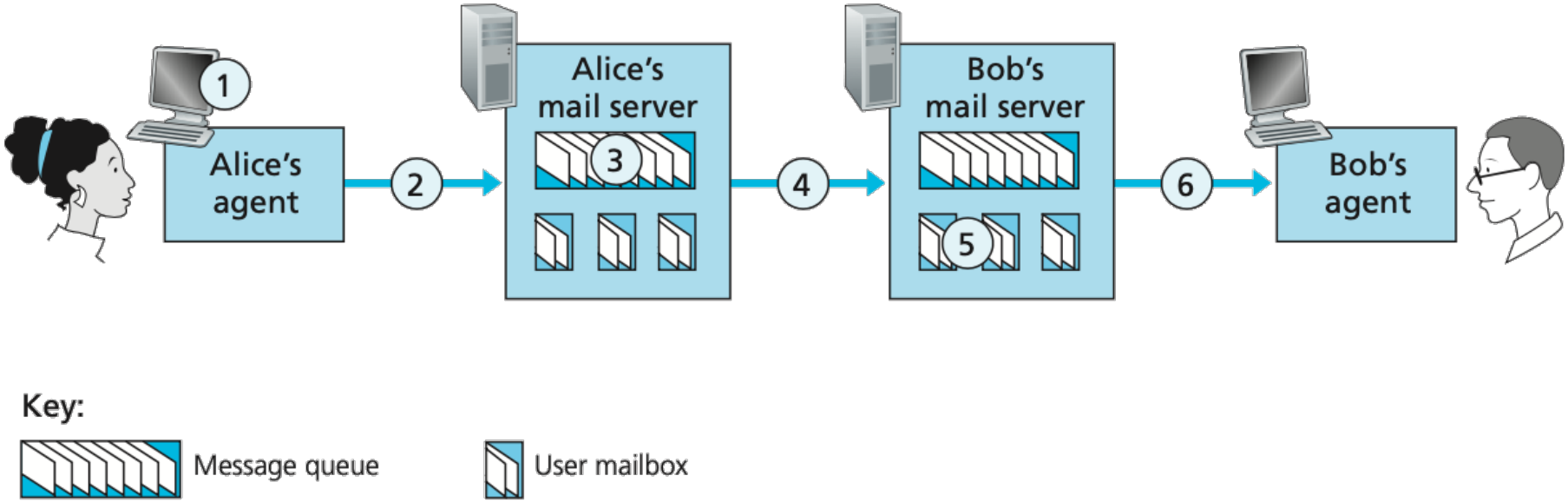


Q1: At point 2 in the diagram, what protocol is being used?

Ans: SMTP

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

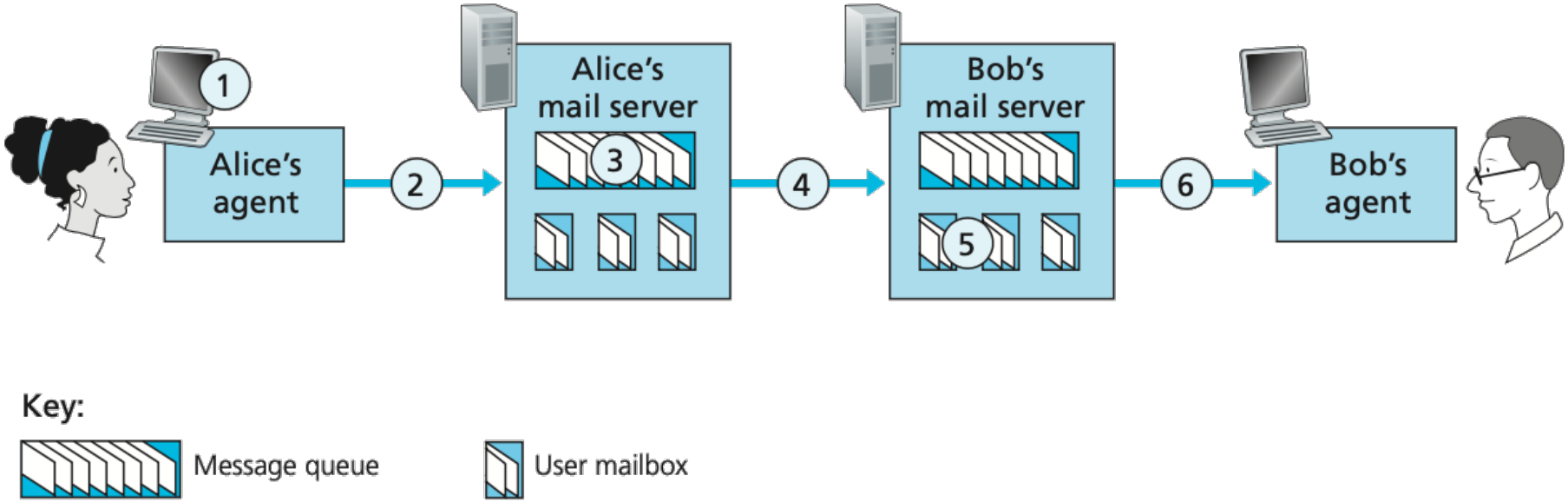


Q2: At point 4 in the diagram, what protocol is being used?

Ans: SMTP

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

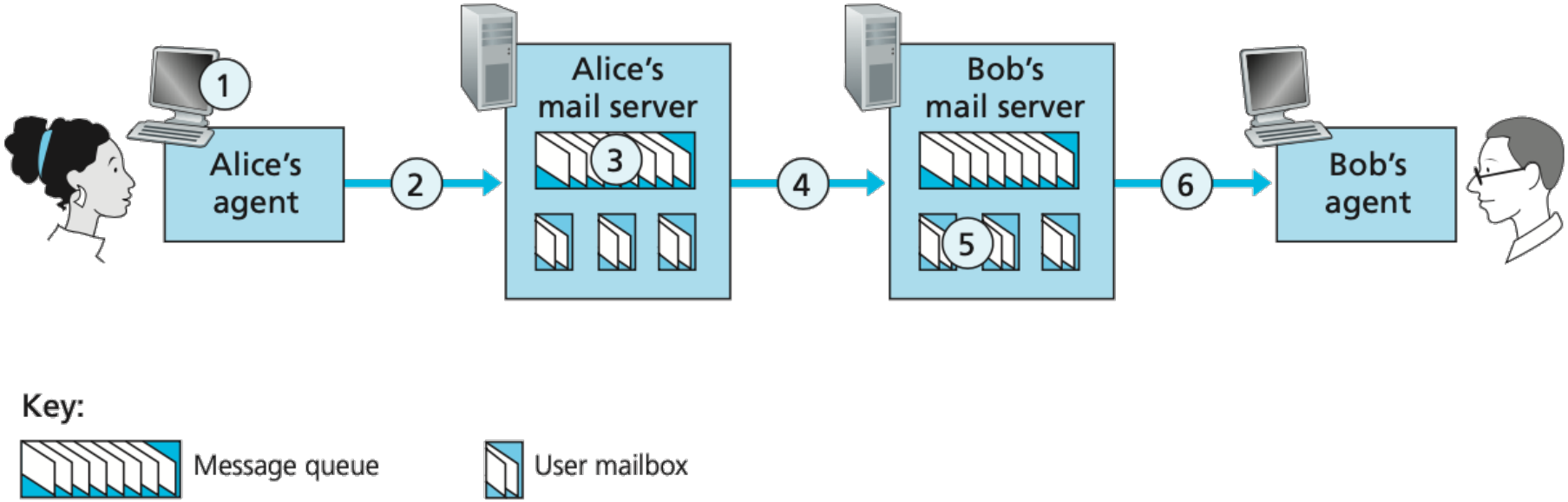


Q3: At point 6 in the diagram, what protocol is being used?

Ans: POP3

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

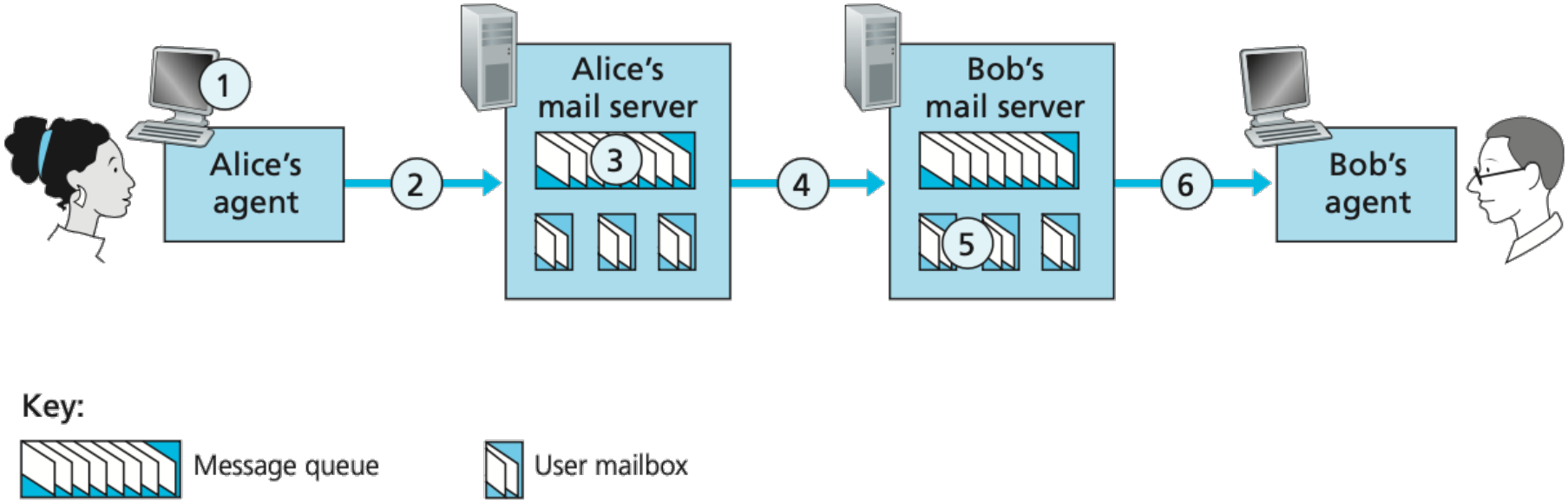


Q4: Does SMTP use TCP or UDP?

Ans: TCP

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

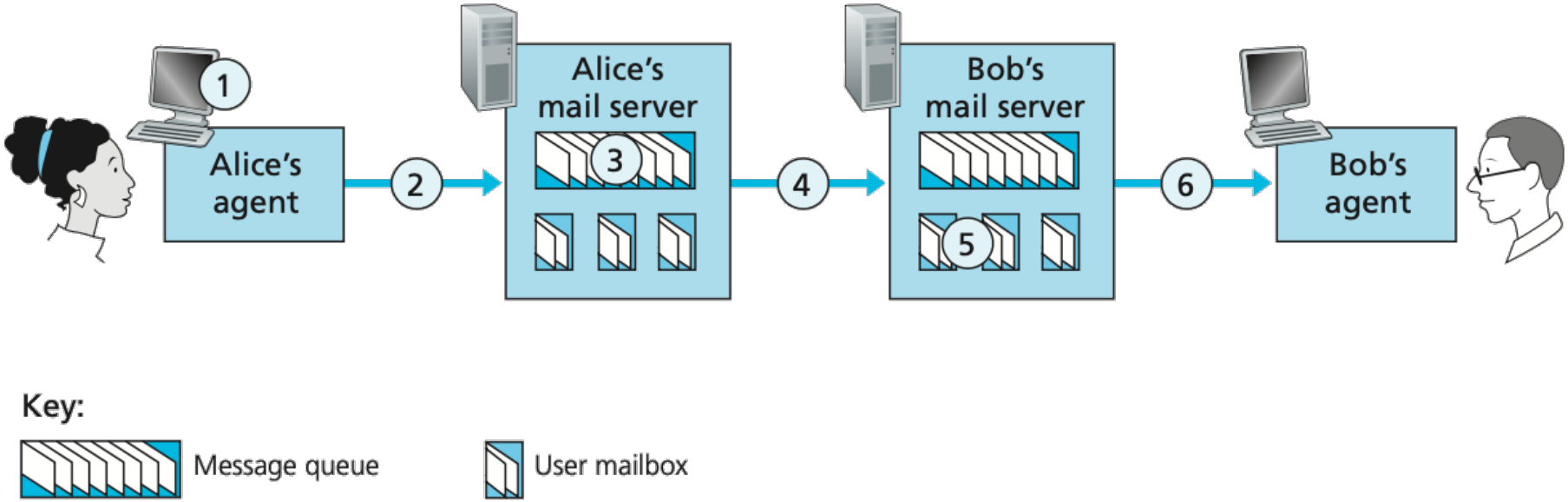


Q4: Does SMTP use TCP or UDP?

Ans: TCP

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

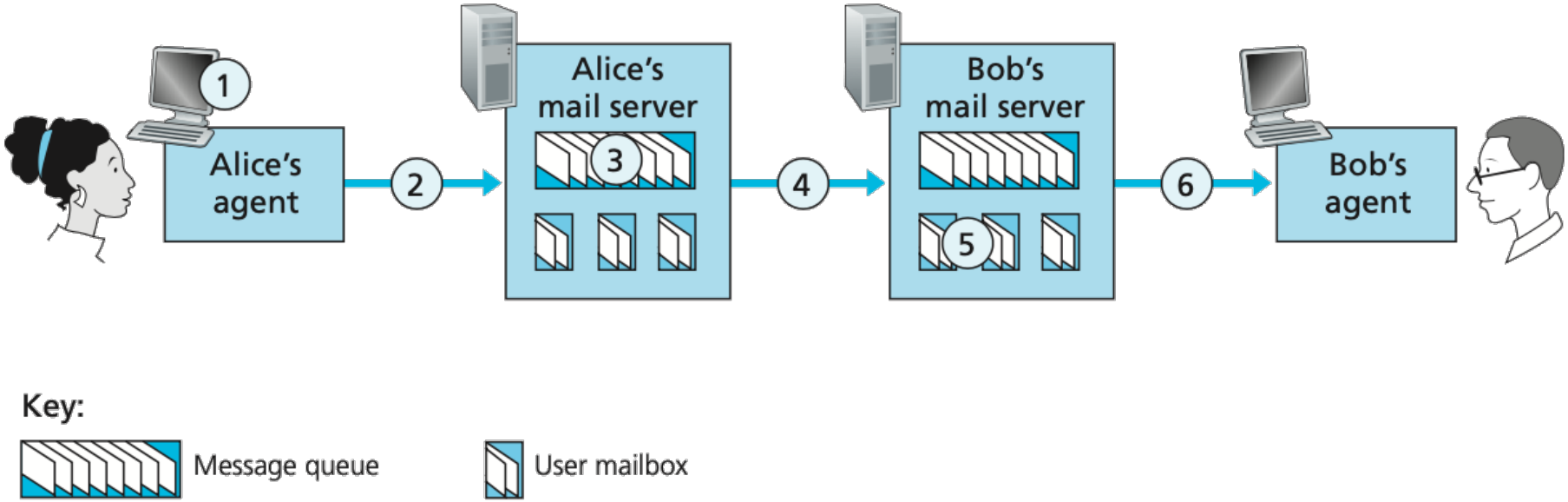


Q5: Is SMTP a 'push' or 'pull' protocol?

Ans: push

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

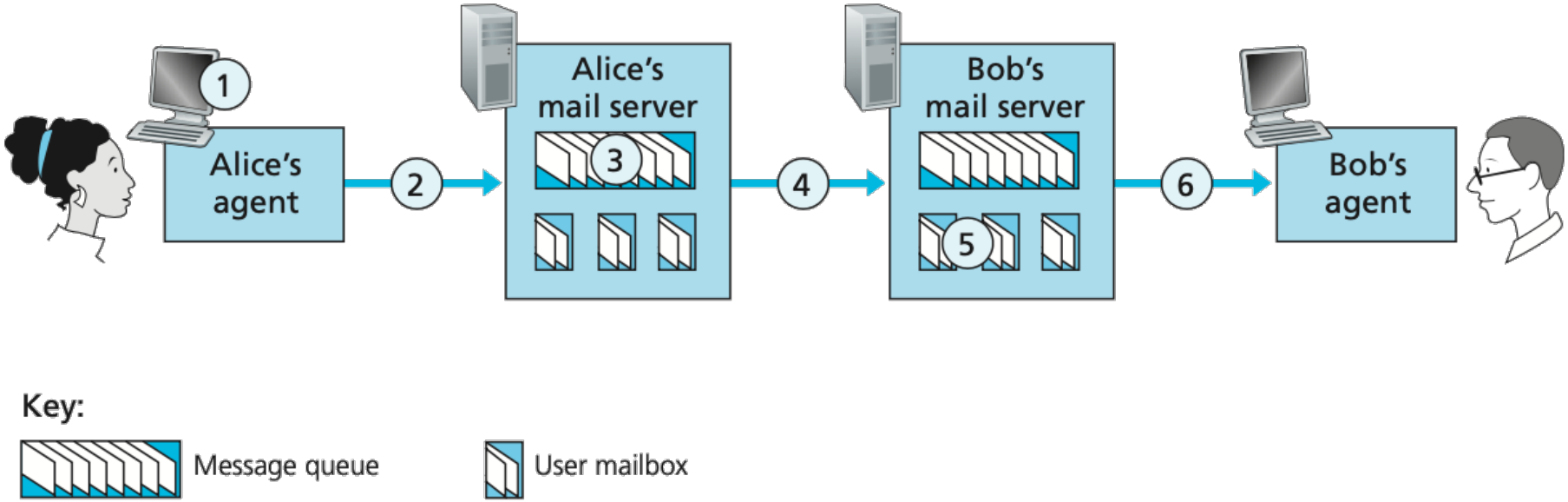


Q6: Is POP3 a 'push' or 'pull' protocol?

Ans: pull

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.

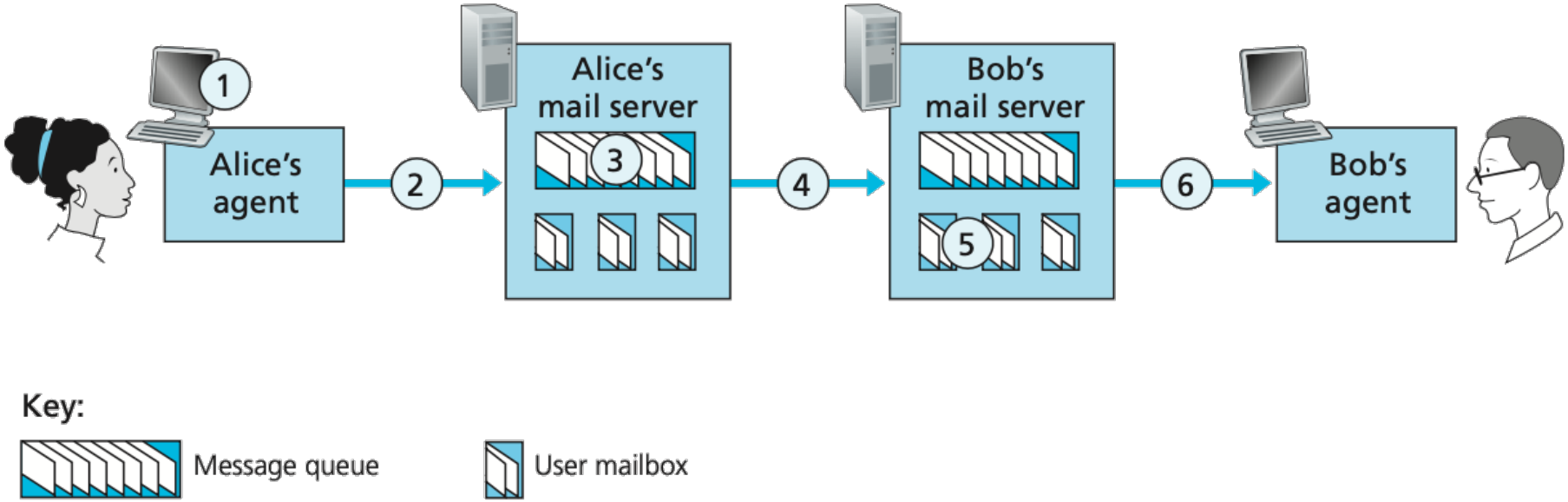


Q7: What port does SMTP use?

Ans: port 25

Exercises: Email and SMTP

Alice sends an email to Bob. Assume both Bob's and Alice's user agents use the POP3 protocol.



Q7: What port does POP3 use?

Ans: port 110

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.google.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: services, structure

DNS services

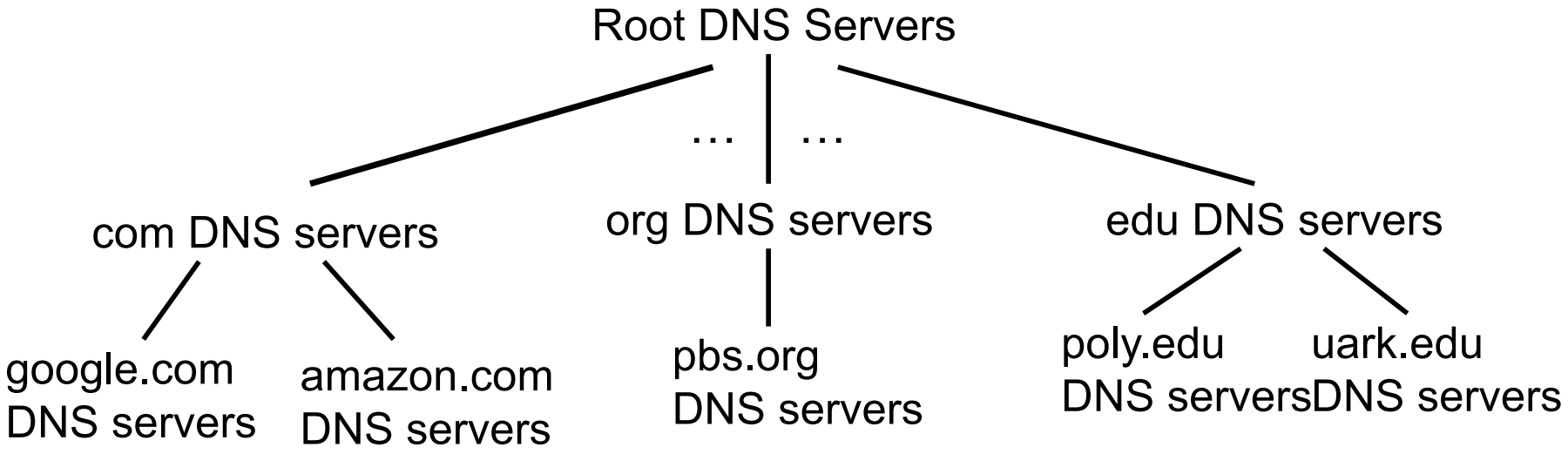
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

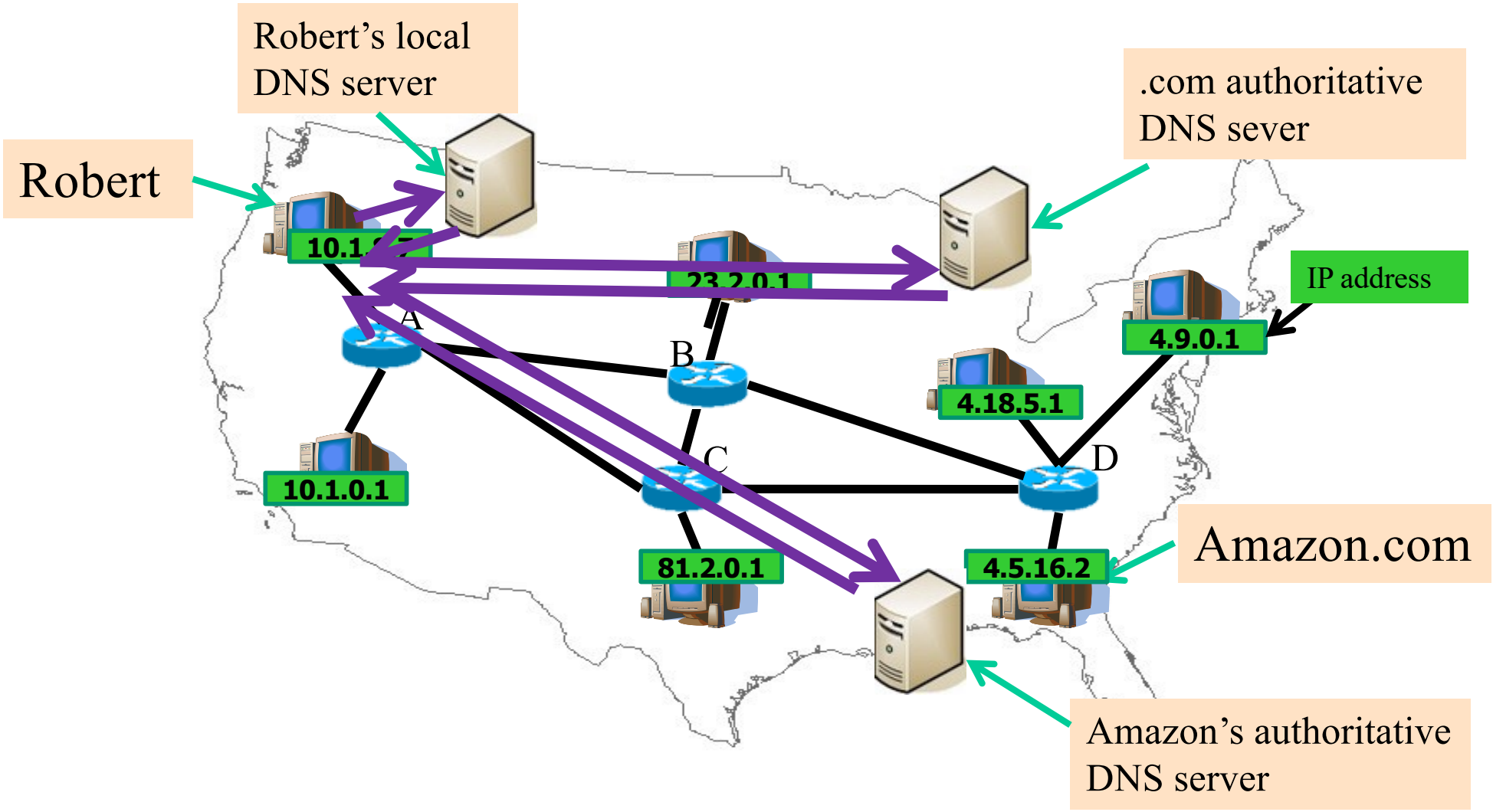
A: doesn't scale!

DNS: a distributed, hierarchical database



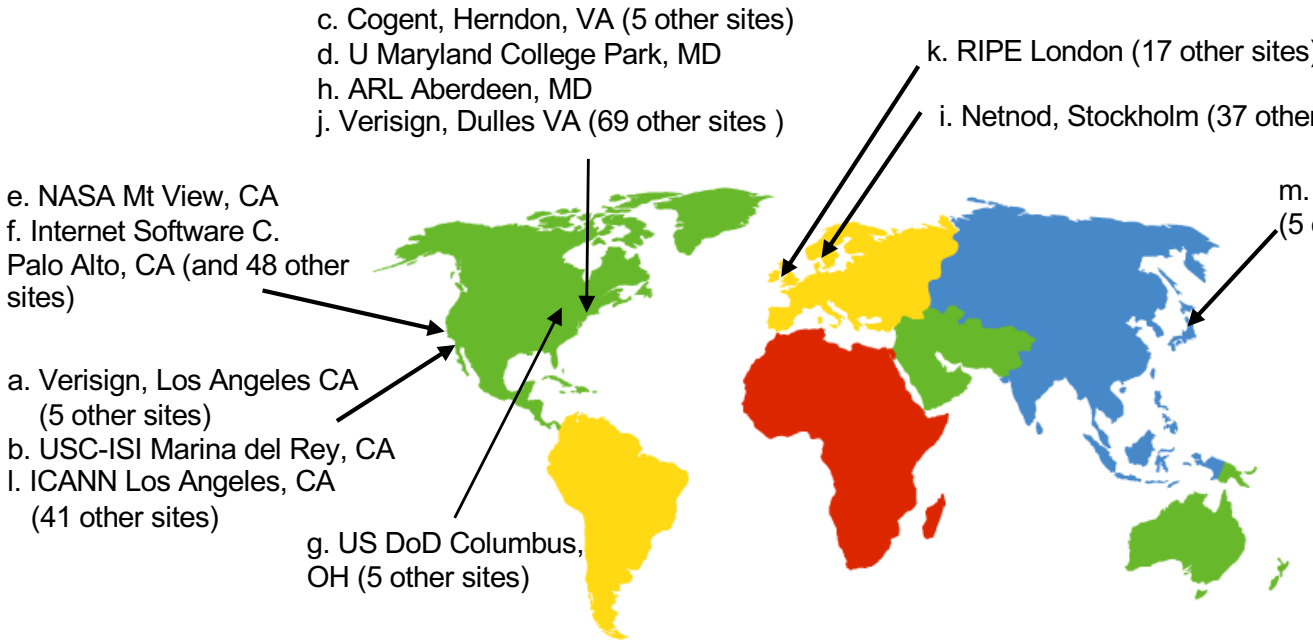
client wants IP for www.amazon.com; 1st approximation:

- 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


Robert's local DNS server
.com authoritative DNS sever
Robert
IP address
A
B
C
D
10.1.0.1
4.9.0.1
4.18.5.1
81.2.0.1
4.5.16.2
Amazon.com
Amazon's authoritative DNS server

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 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

Demo: WHOIS Lookup database

- https://who.is/

Local DNS name server

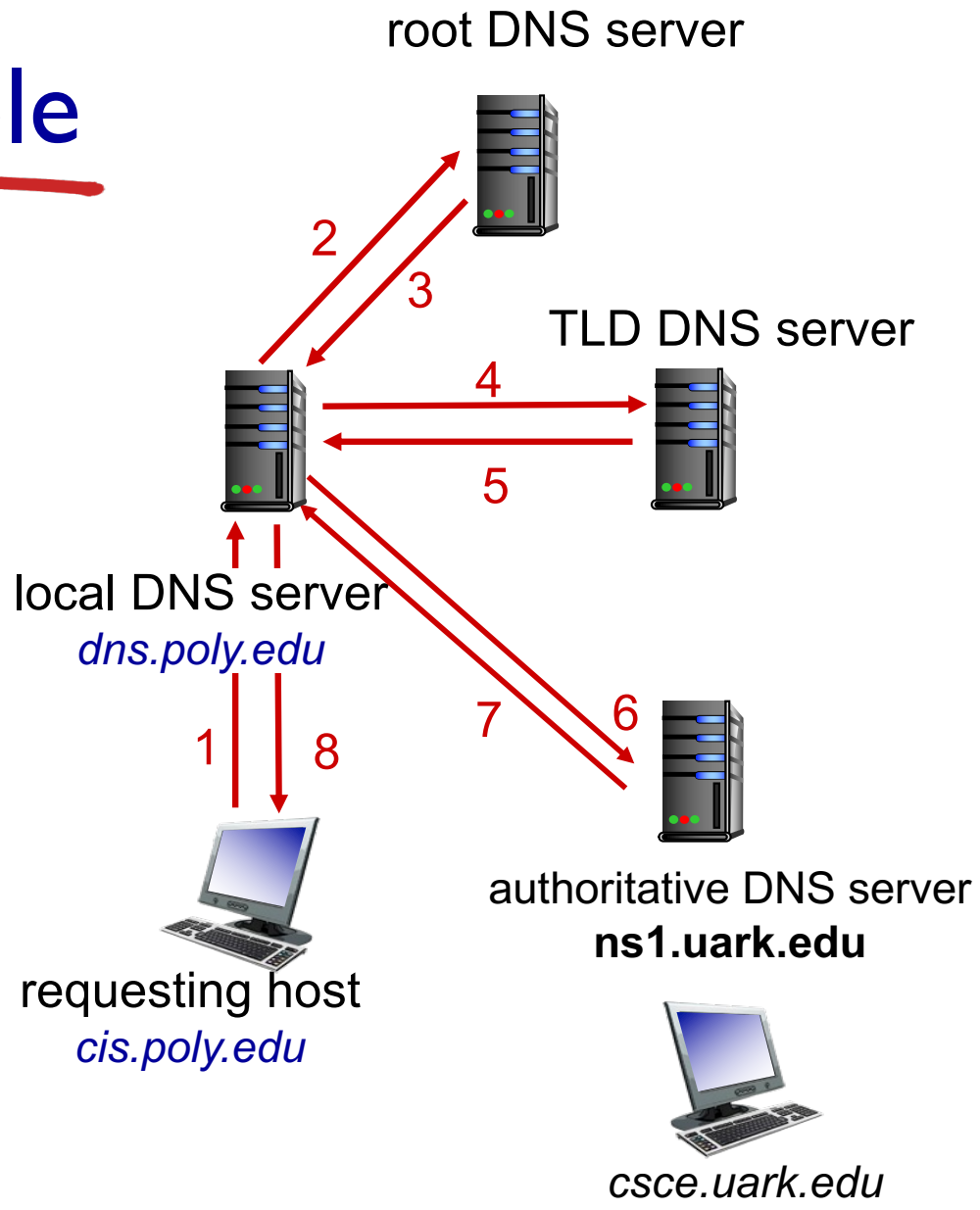
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

DNS name resolution example

- host at cis.poly.edu wants IP address for csce.uark.edu

iterated query:

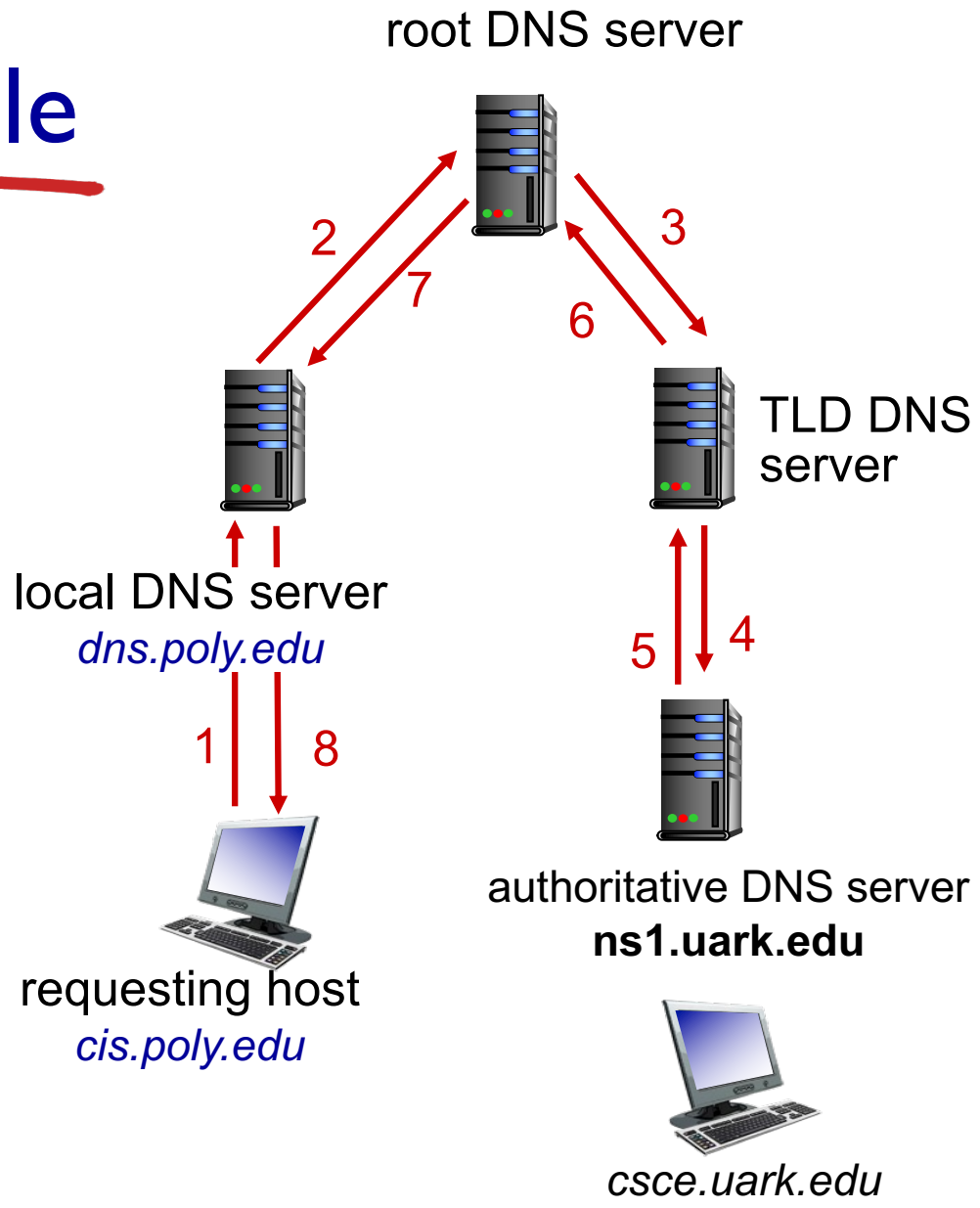
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



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 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

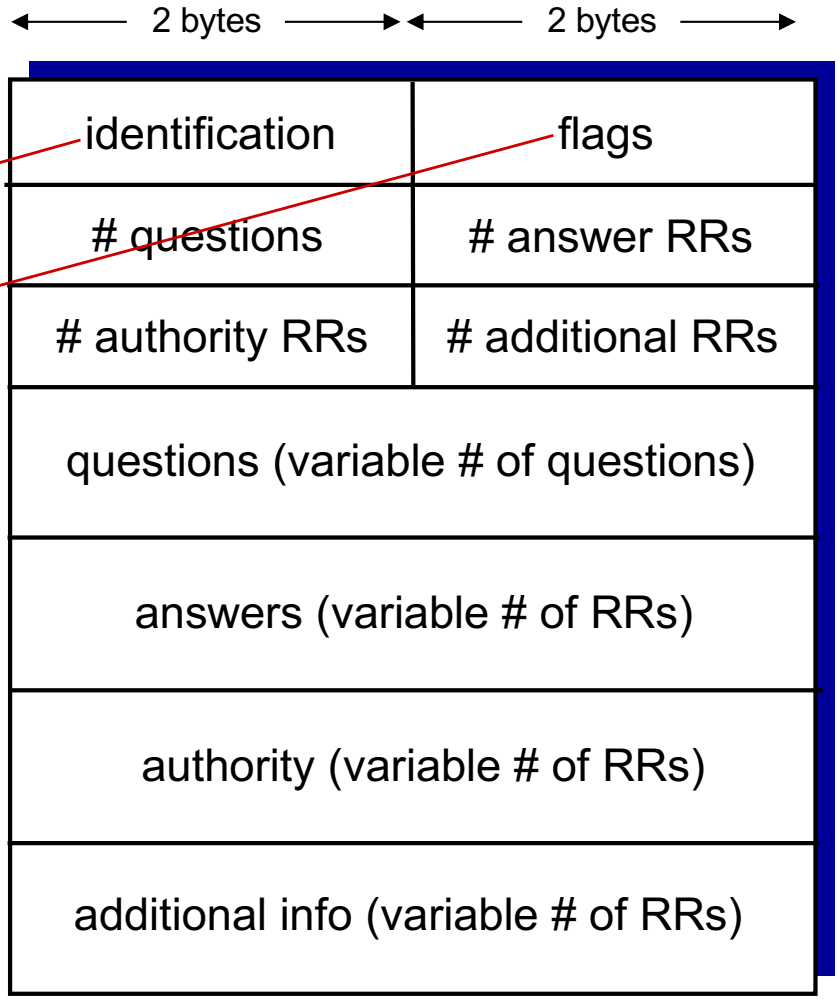
- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

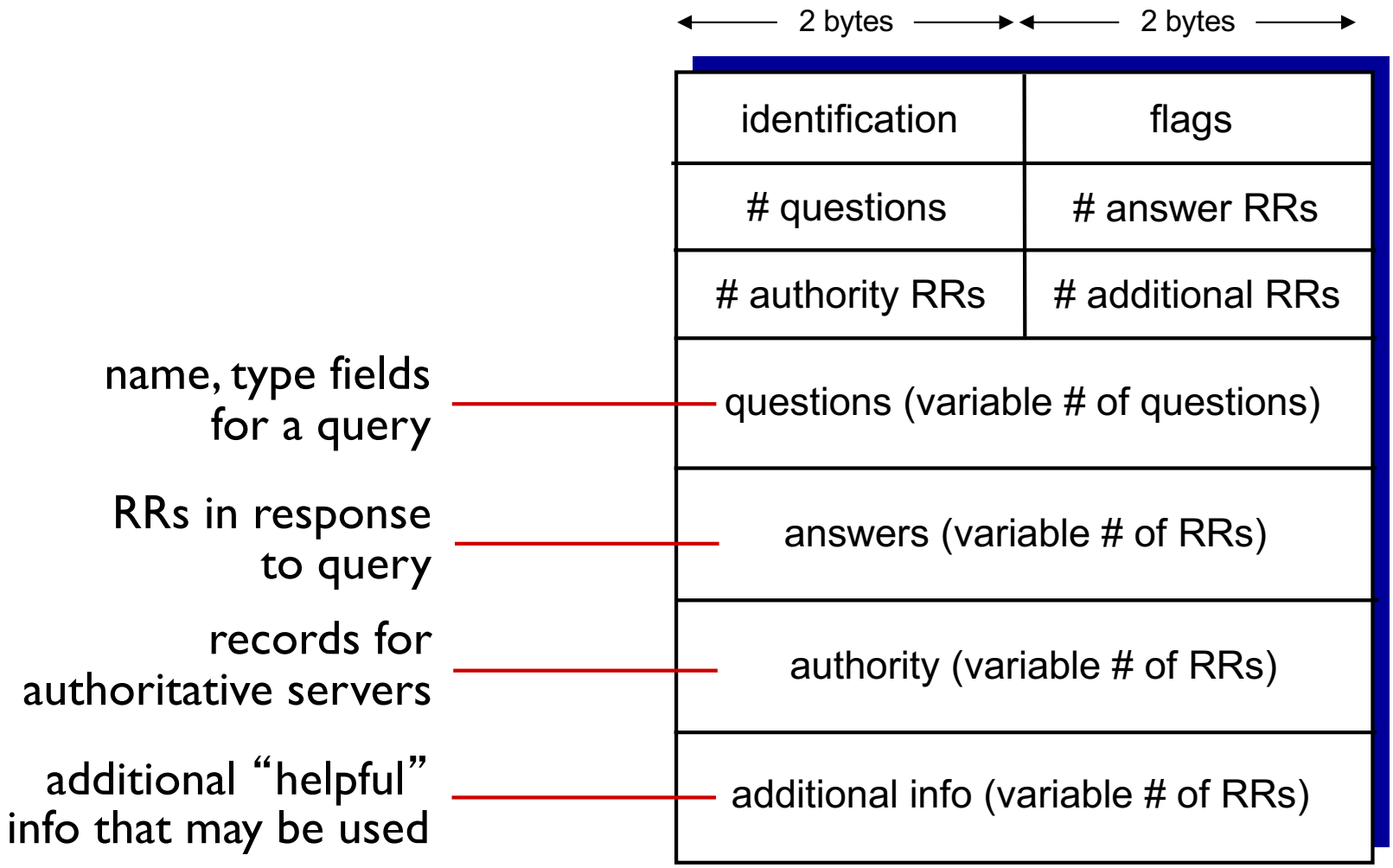
- *query* and *reply* messages, both with same *message format*

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 MX record for networkutopia.com

Attacking DNS

DDoS attacks

- bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
  - potentially more dangerous

redirect attacks

- man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification

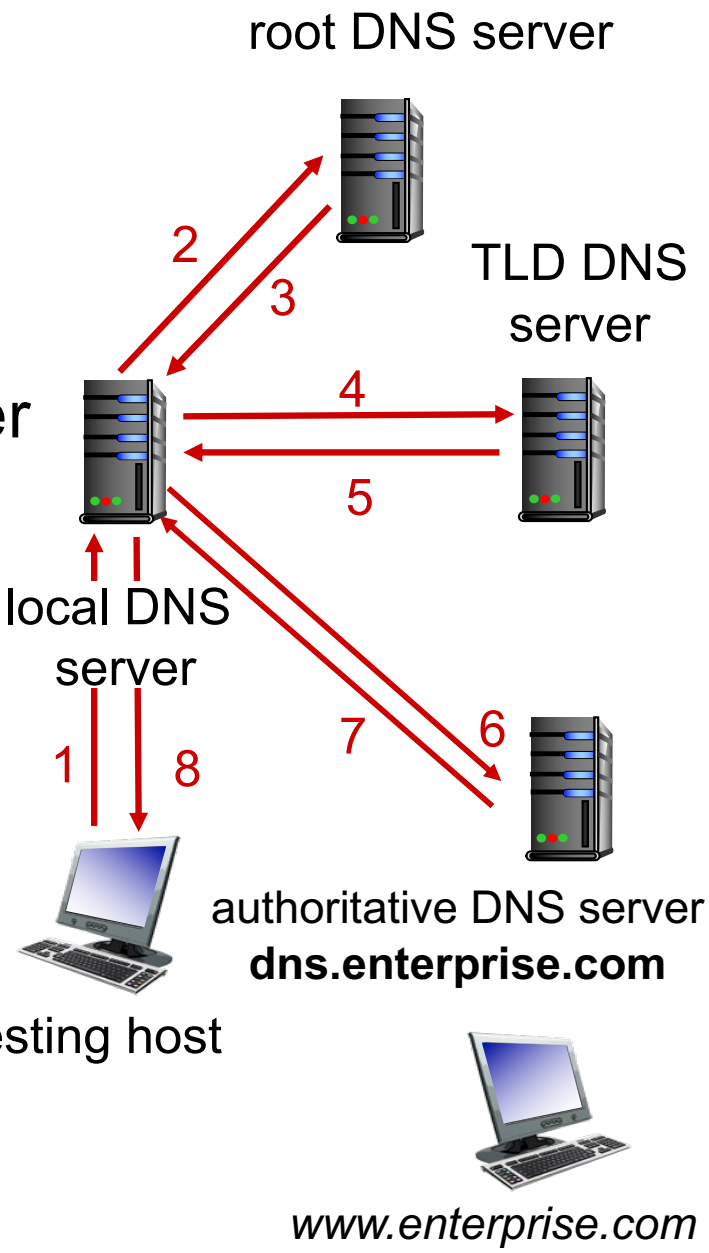
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q1. What transport protocol(s) does DNS use: TCP, UDP, or Both?

Ans: Both

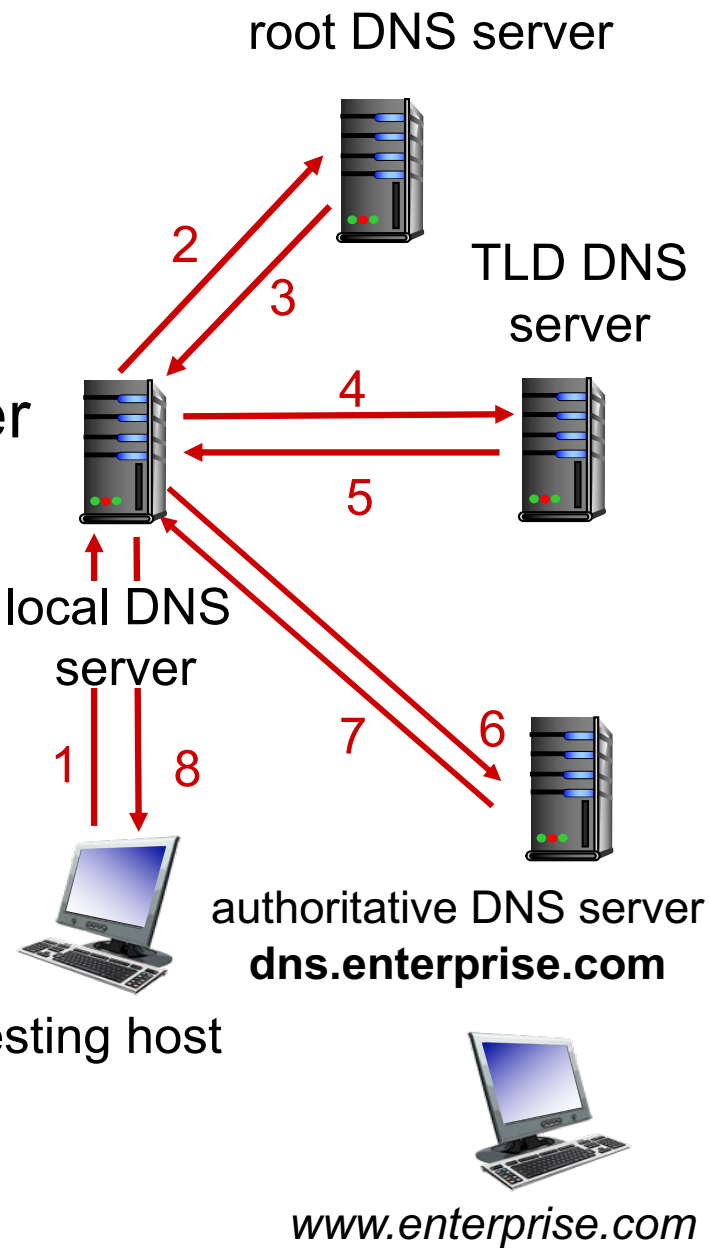
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q2. What well-known port does DNS use?

Ans: port 53

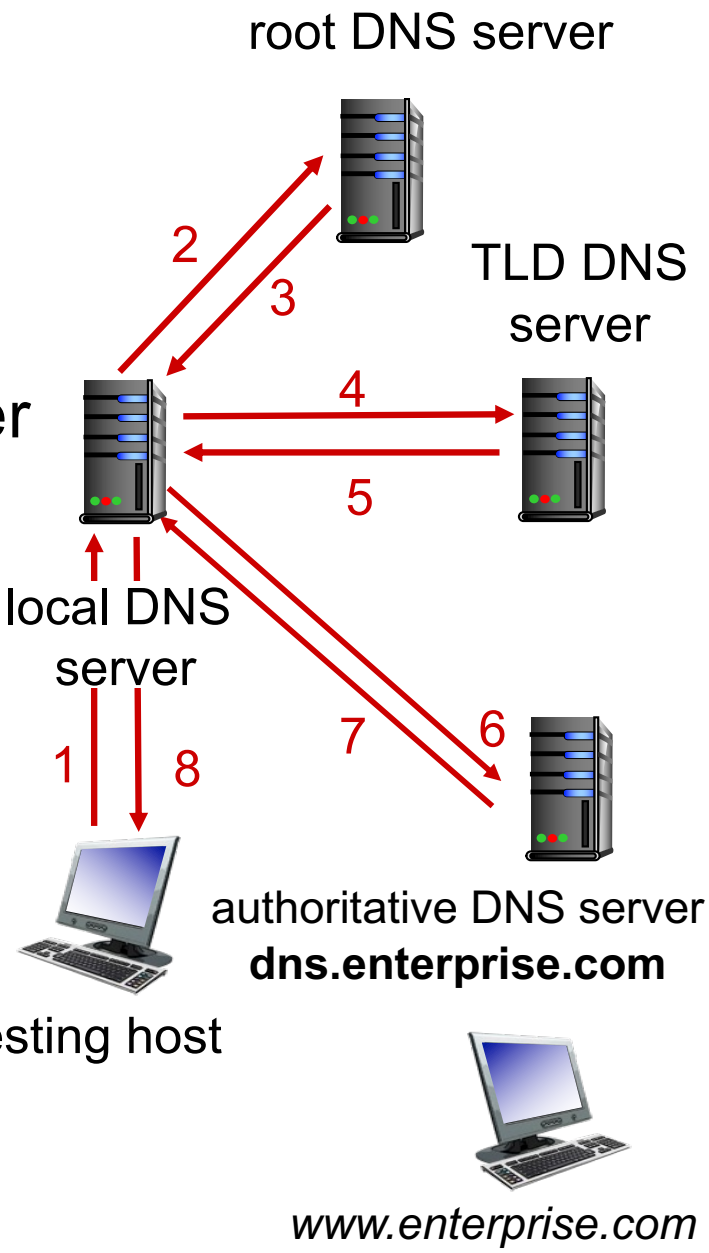
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q3. Can you send multiple DNS questions and get multiple RR answers in one message? (Yes or No)

Ans: Yes

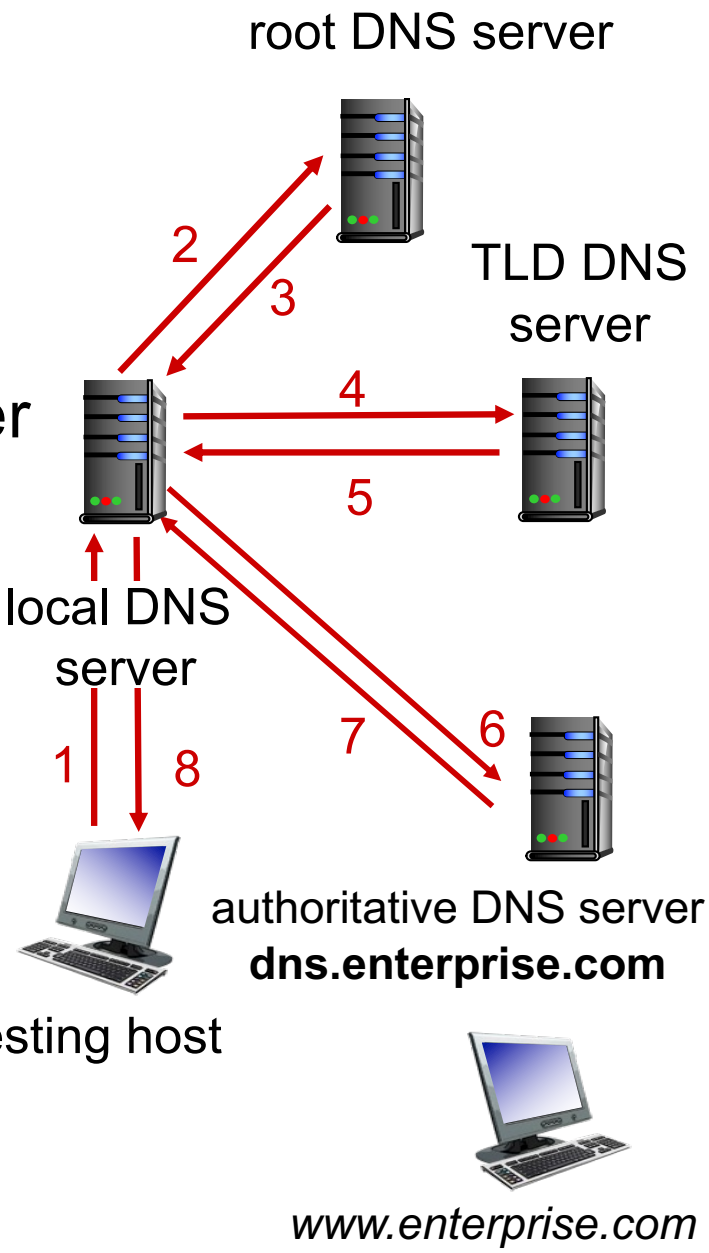
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q4. To which DNS server does a host send their requests to?

Ans: Local DNS server, which acts on behalf of the host.

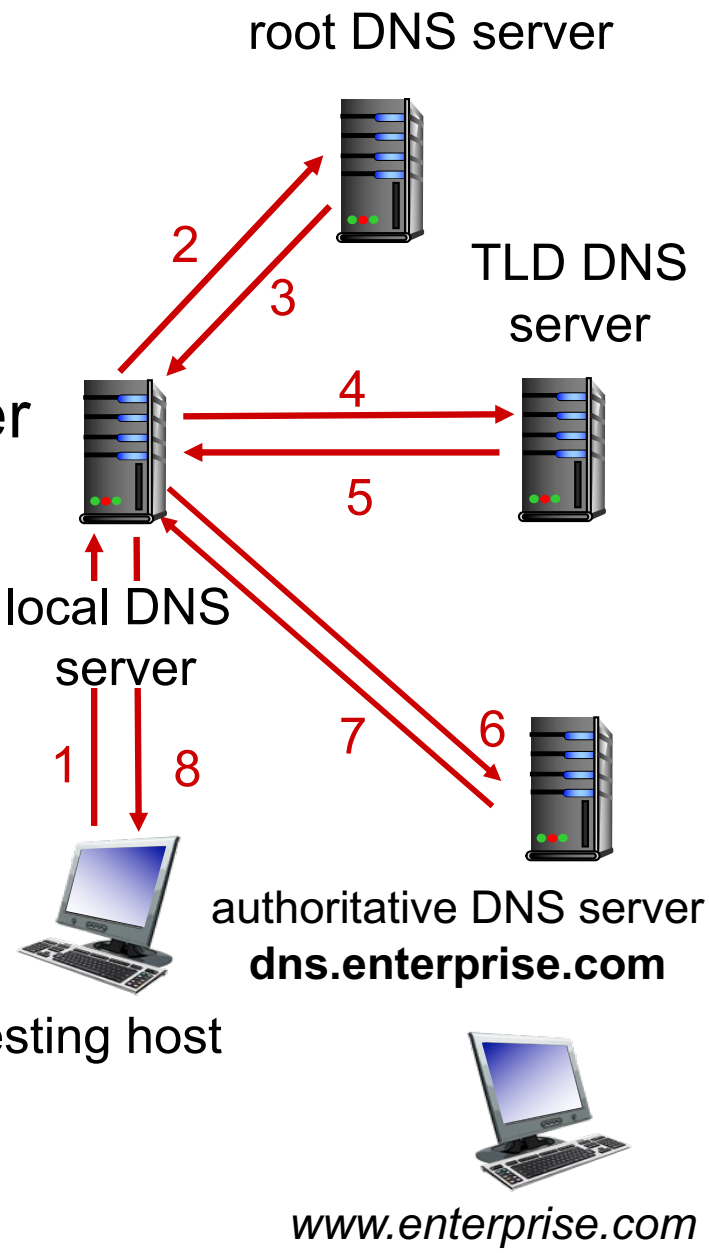
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q5. Which type of DNS server holds a company's DNS records?

Ans: company's Authoritative DNS server

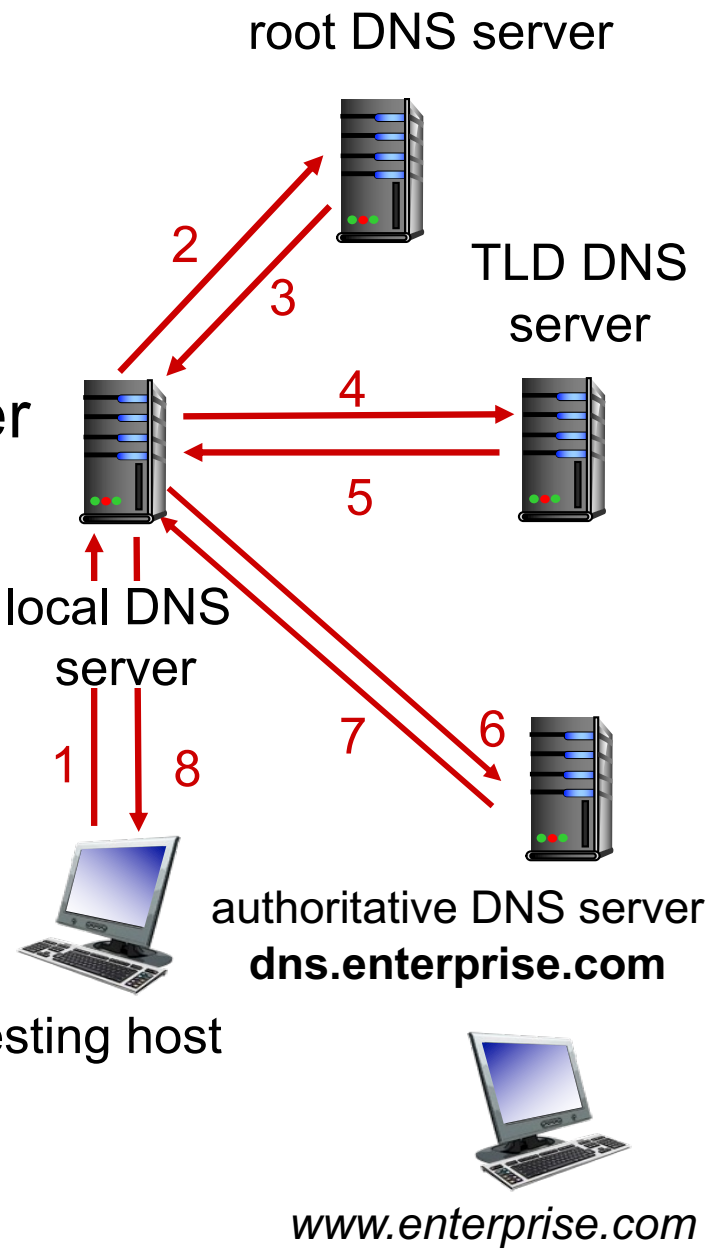
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q6. What is the name of the DNS server for enterprise.com?

Ans: dns.enterprise.com

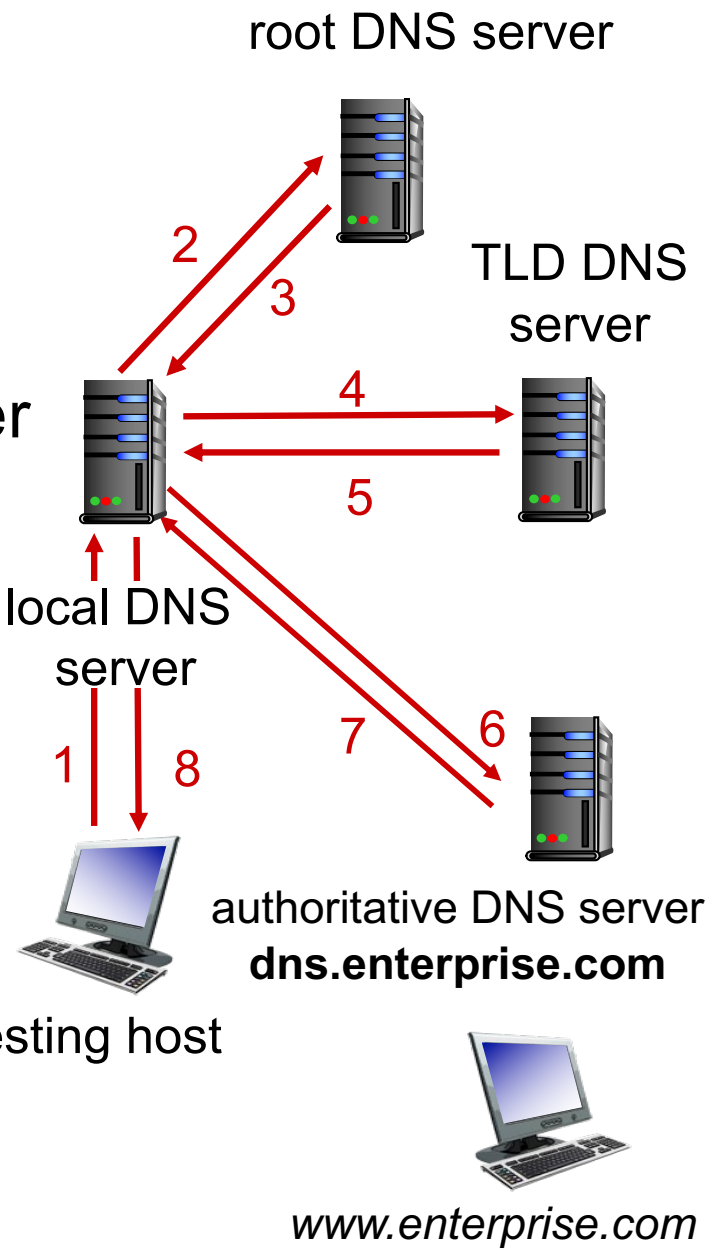
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q7. When the local DSN server contacts the TLD server, how many answers (RR) are returned?

Ans: 2 records returned
a NS record, and an A record for the DNS server

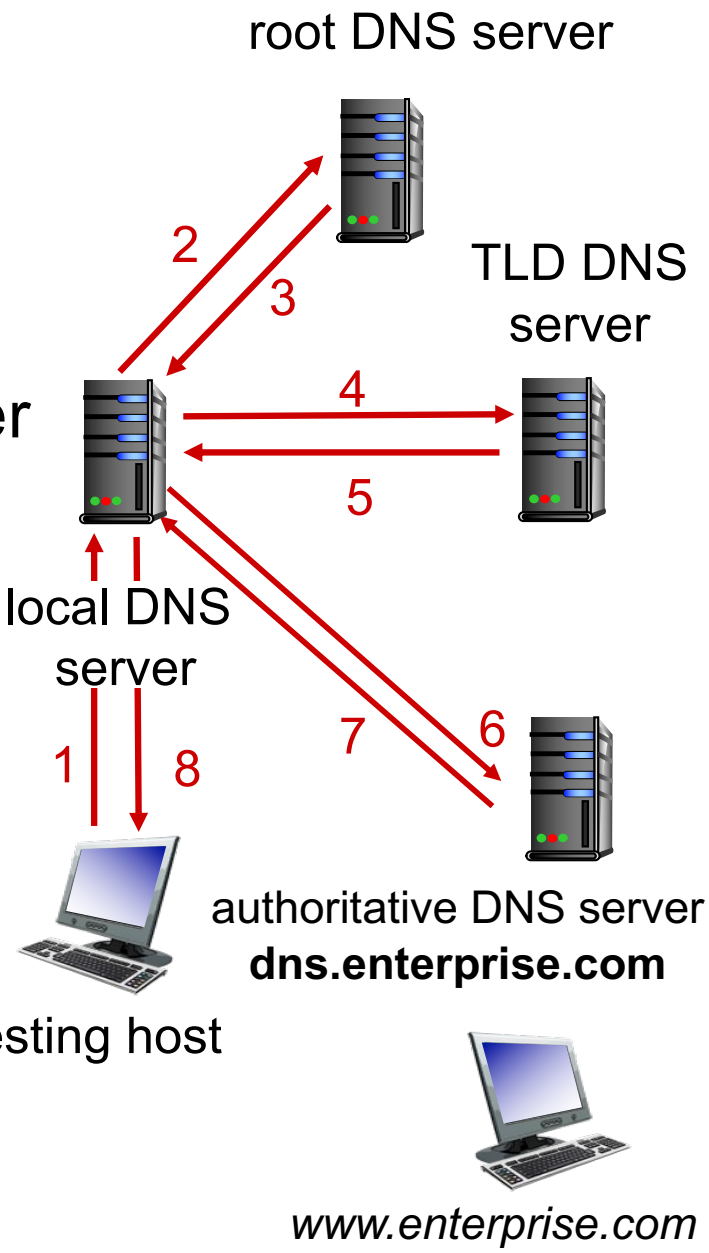
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q8. What was the content of the A record?
Answer with the format: "name, value"

Ans: The A record has contents:
(dns.enterprise.com, 146.54.154.34)

Exercises: DNS

Records on the TLD DNS server

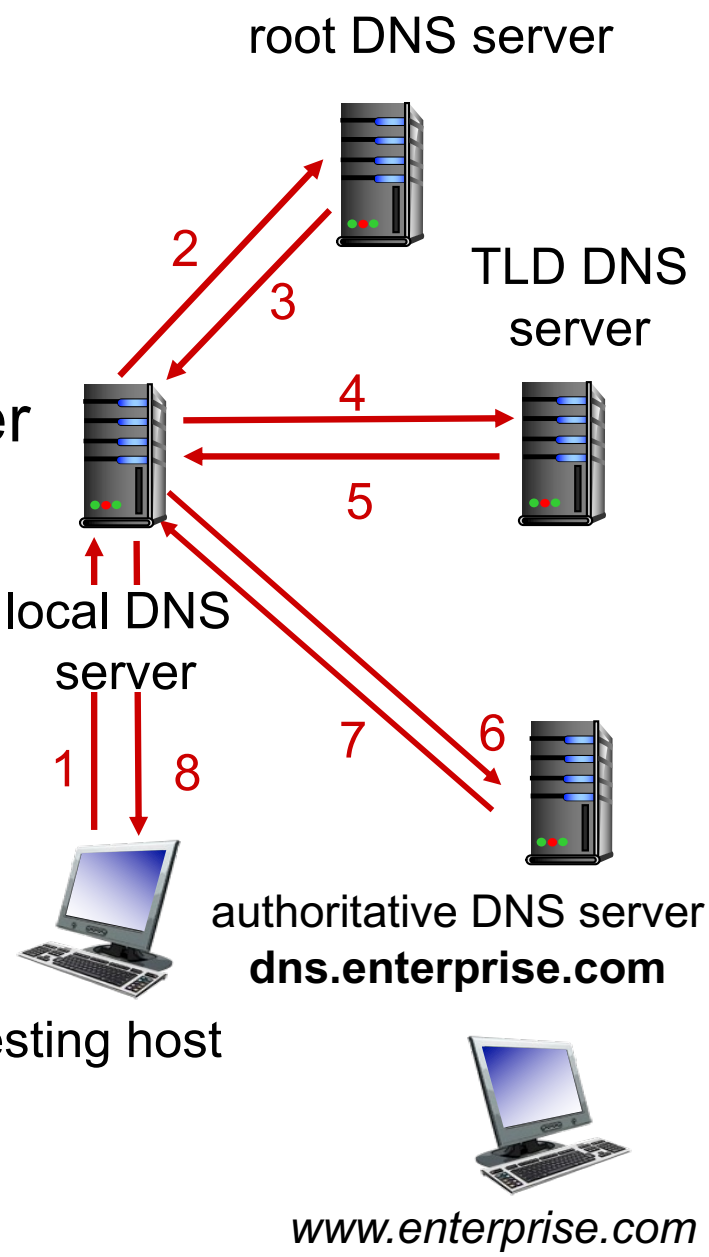
- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)

Q9. Assume that the enterprise.com website is actually hosted on west3.enterprise.com, what type of record is needed for this?

Ans: a CNAME record



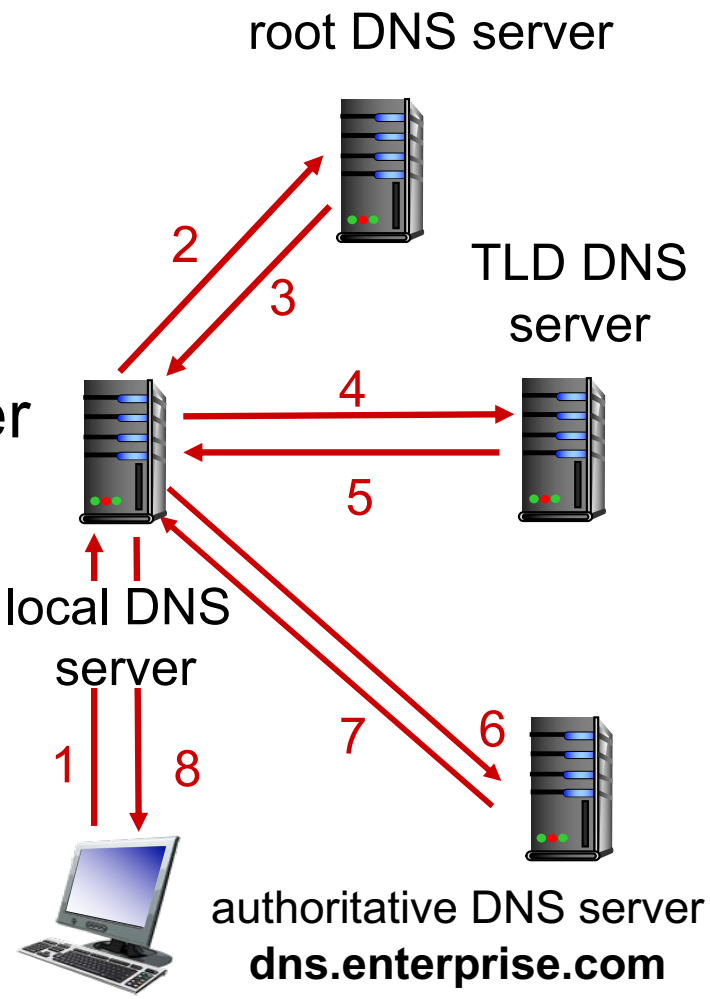
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q10. We try to send an email to admin@enterprise.com with the mail server mail.enterprise.com. What type of record will contain the name of the enterprise.com domain and the name of its mailserver(s)?

Ans: An MX record will be returned.

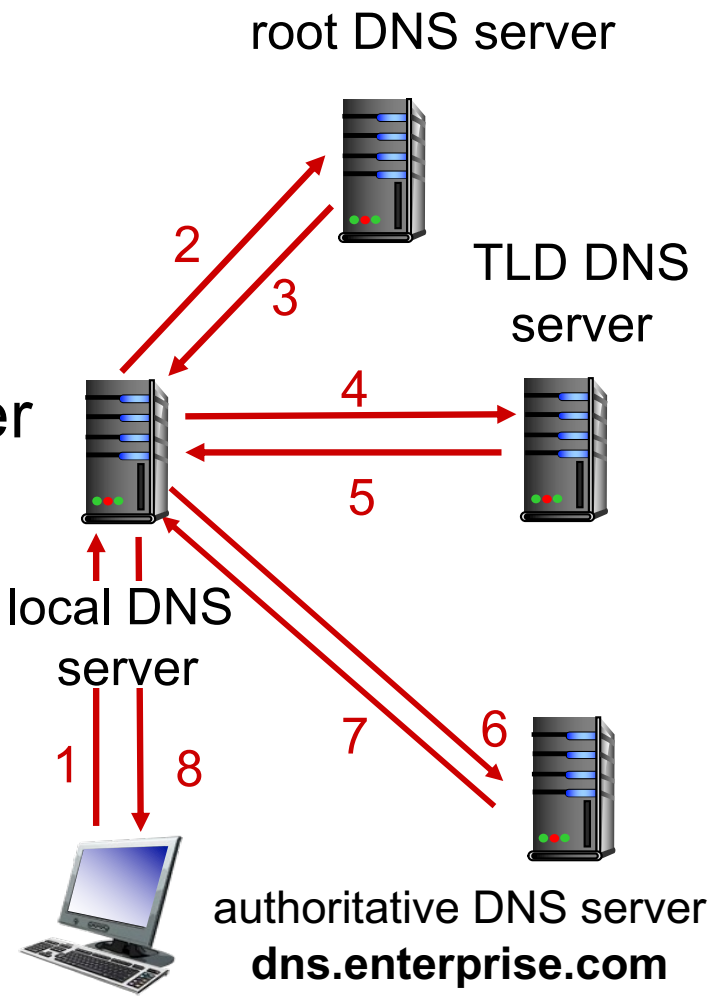
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q11. In that MX record, what are the contents? Answer with the format: "name, value"

Ans: The MX record has contents: (enterprise.com, mail.enterprise.com)

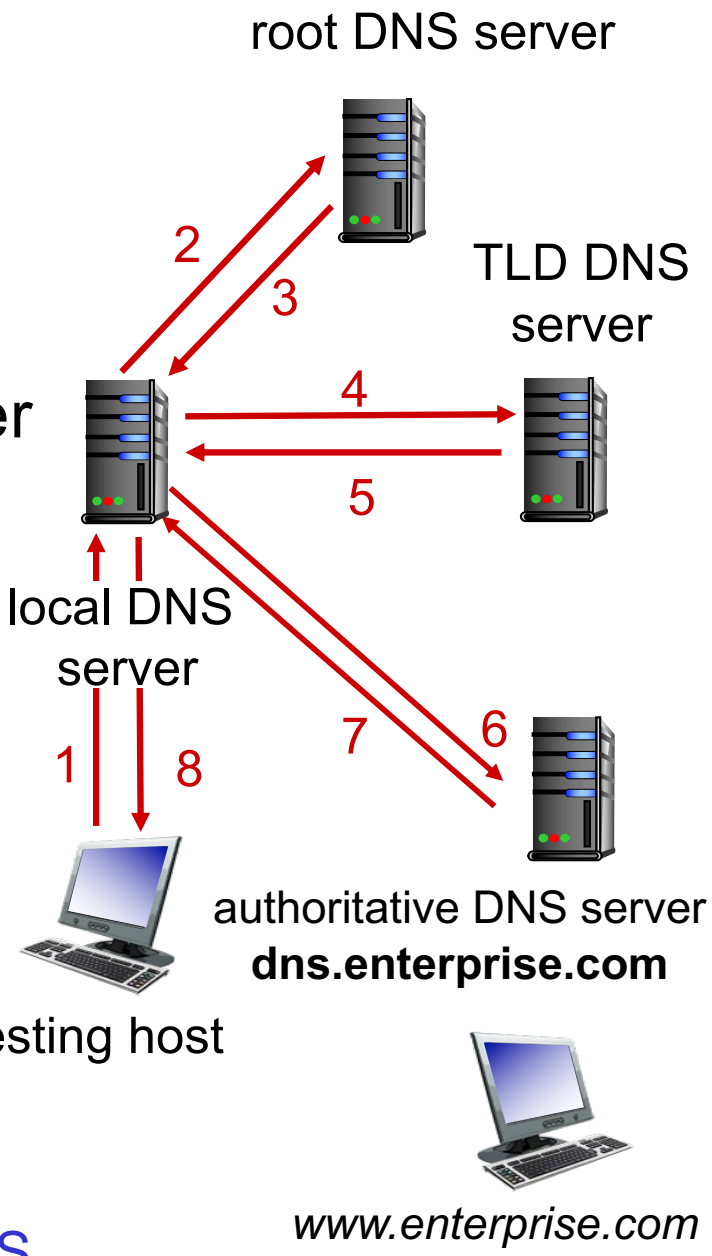
Exercises: DNS

Records on the TLD DNS server

- (www.enterprise.com, dns.enterprise.com, NS)
- (dns.enterprise.com, 146.54.154.34, A)

Records on the enterprise.com DNS server

- (www.enterprise.com, west3.enterprise.com, CNAME)
- (west3.enterprise.com, 142.81.17.206, A)
- (enterprise.com, mail.enterprise.com, MX)
- (mail.enterprise.com, 247.29.102.42, A)



Q12. Does your local DNS server take advantage of caching similar to web requests? (Yes or No)

Ans: Yes, DNS servers (especially your Local DNS server) cache records for faster retrieval.

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
- socket programming:

  TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

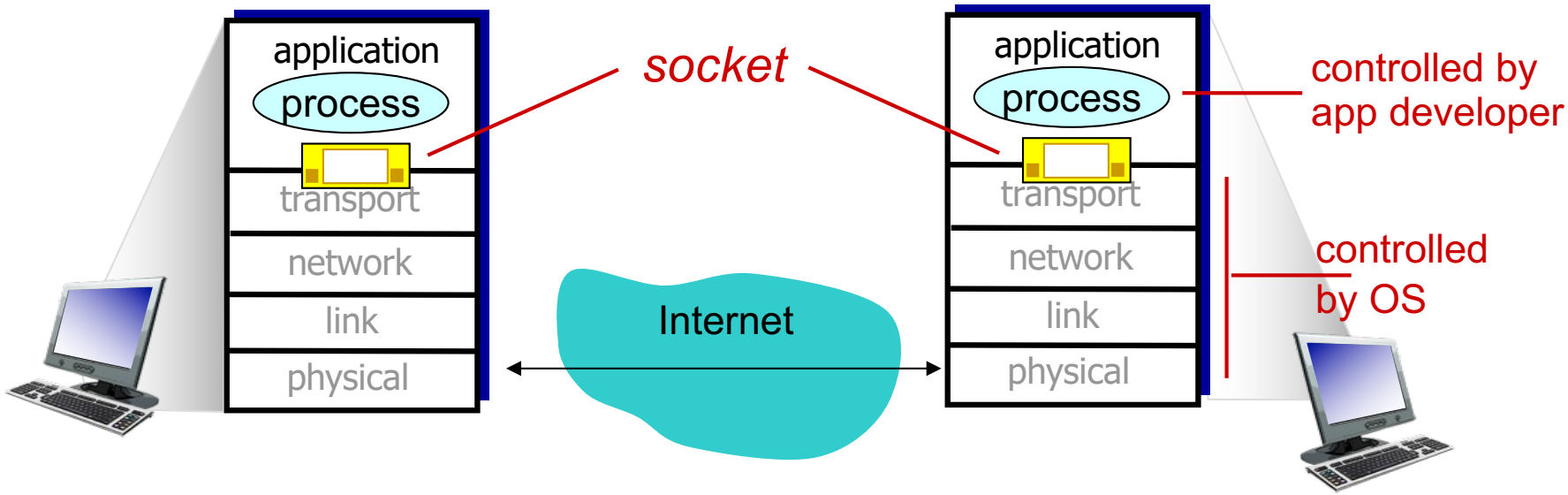
important themes:

- control vs. messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- "complexity at network edge"

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket programming *with UDP*

UDP: no "connection" between client & server

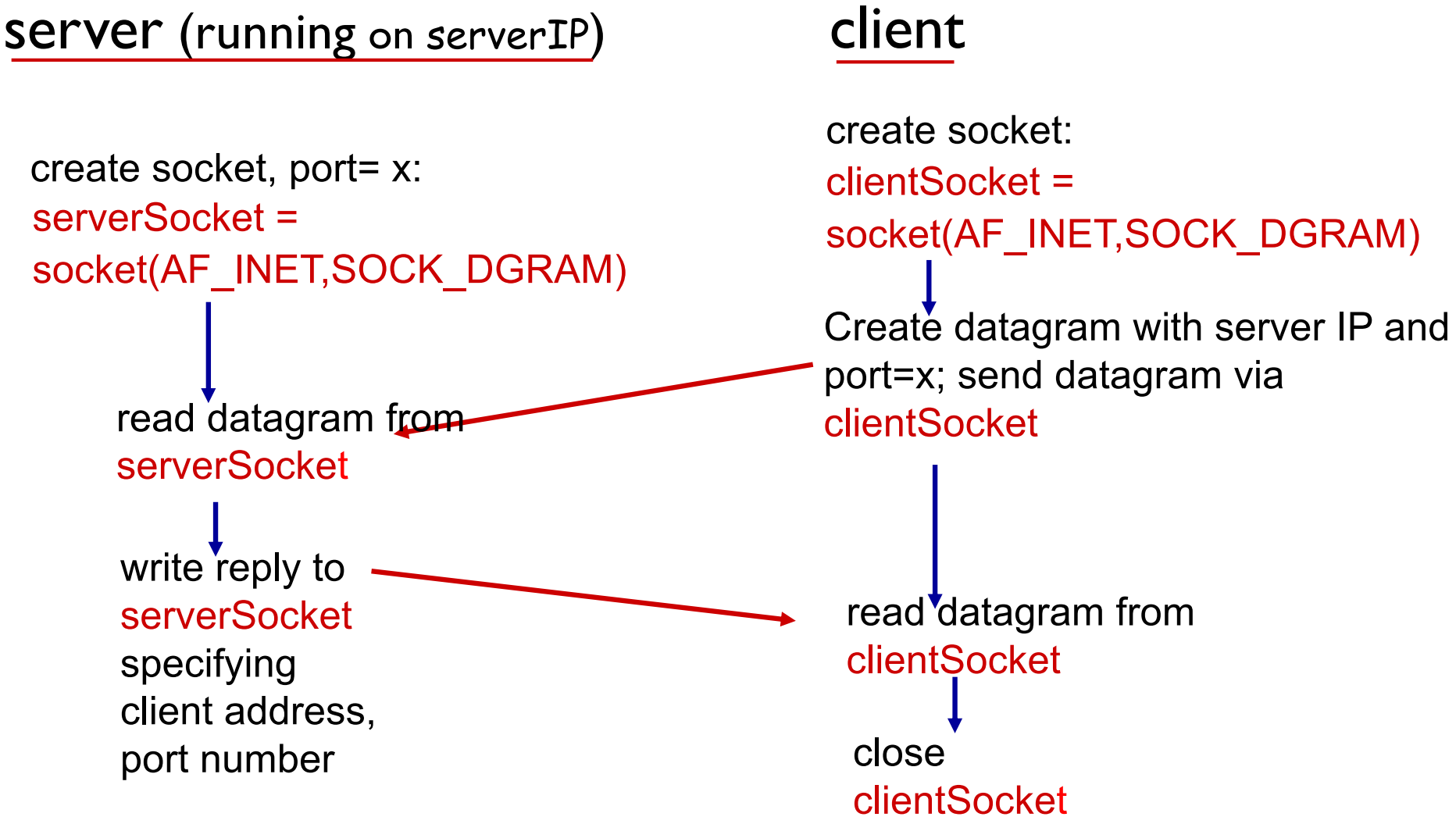
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



Example app: UDP client

Python UDPClient

include Python's socket library

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```

get user keyboard input

```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage.decode()
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

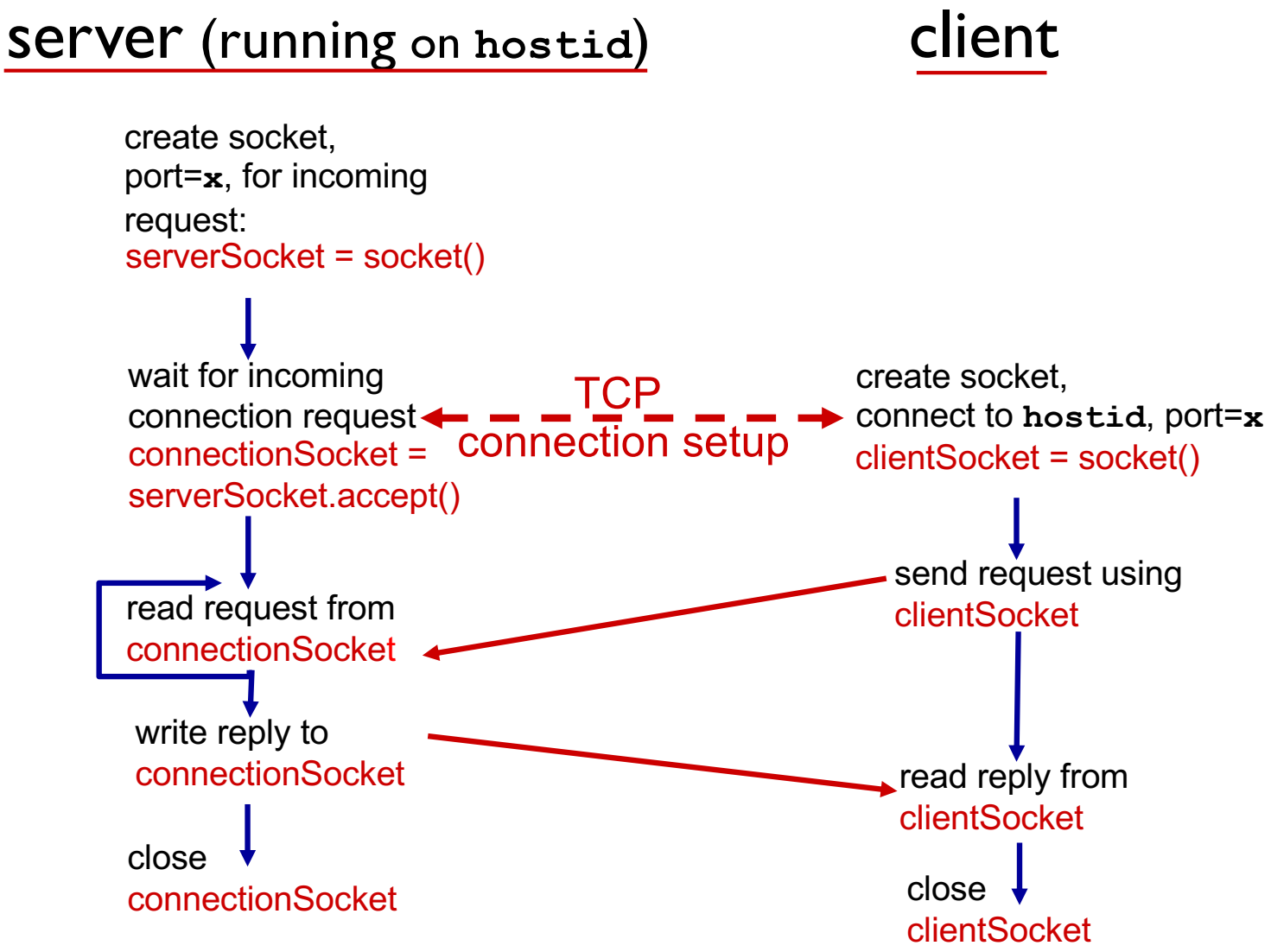
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                          encode())

    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - SMTP, POP, IMAP
  - DNS
- socket programming:

  TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

important themes:

- control vs. messages
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable message transfer
- "complexity at network edge"