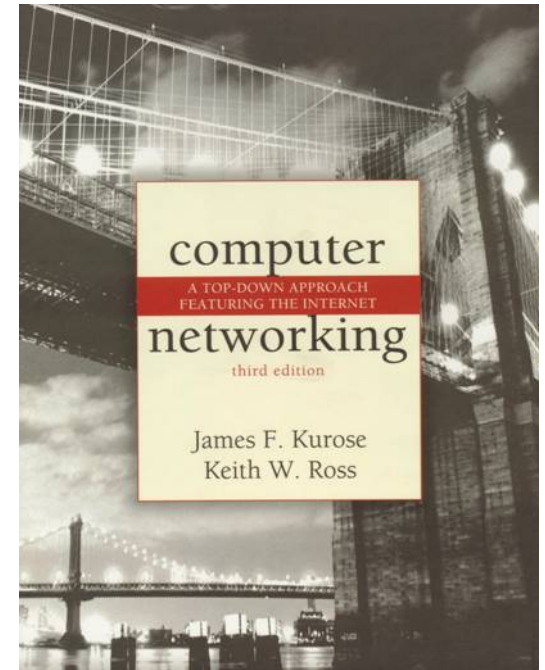


Chapter 2

Application Layer



*Computer Networking:
A Top Down Approach
Featuring the Internet,
3rd edition.*

*Jim Kurose, Keith Ross
Addison-Wesley, July
2004.*

Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
 - HTTP
 - FTP
 - SMTP / POP3 / IMAP
 - DNS

Some network apps

- ❑ E-mail
- ❑ Web
- ❑ Instant messaging
- ❑ Remote login
- ❑ P2P file sharing
- ❑ Multi-user network games
- ❑ Streaming stored video clips
- ❑ Internet telephone
- ❑ Real-time video conference
- ❑ Massive parallel computing

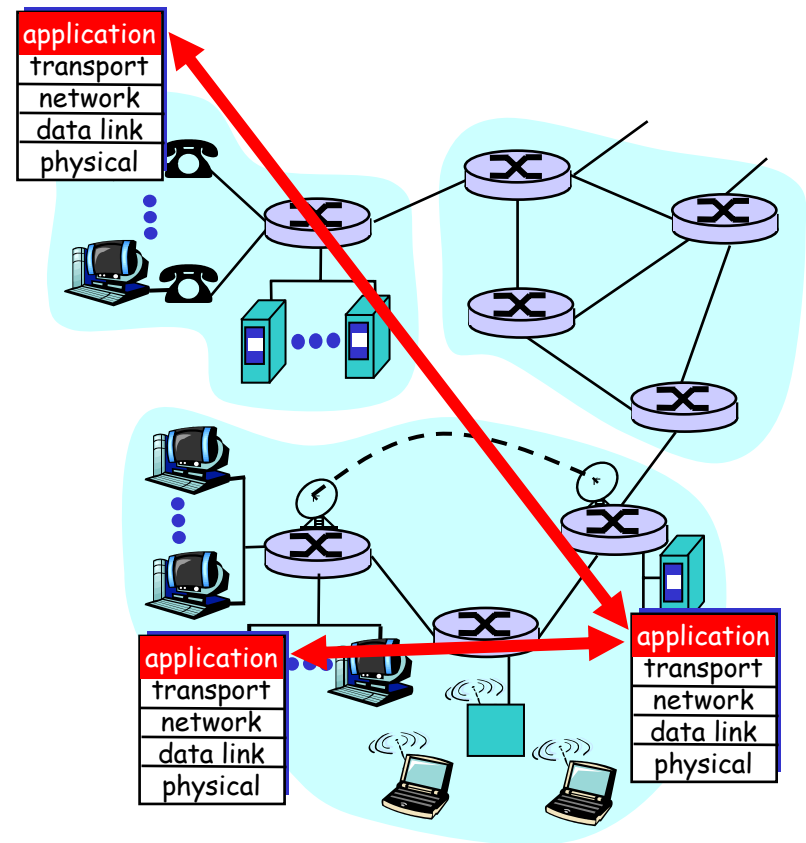
Creating a network app

Write programs that

- run on different end systems and
- communicate over a network.
- e.g., Web: Web server software communicates with browser software

No software written for devices in network core

- Network core devices do not function at app layer
- This design allows for rapid app development



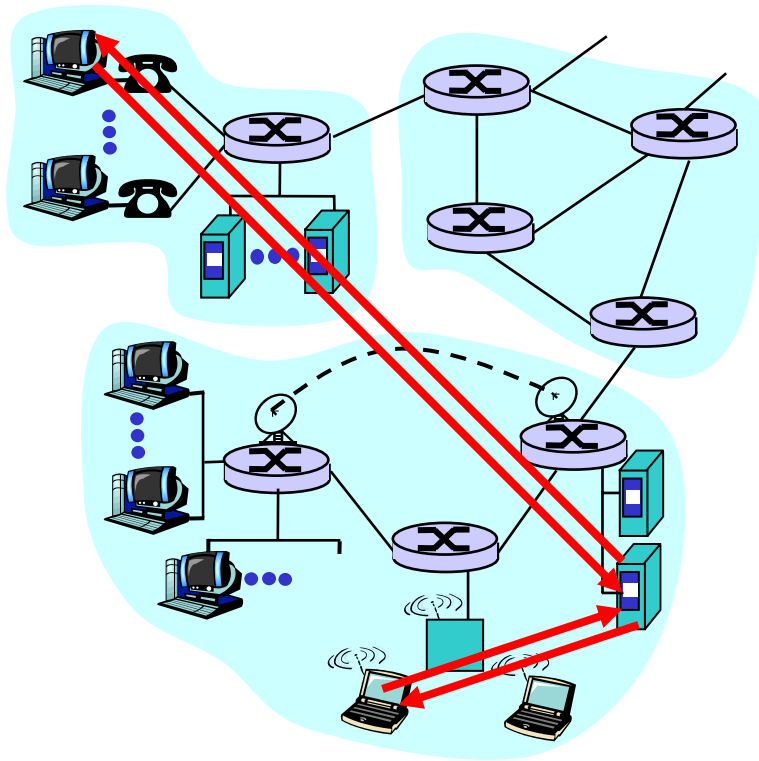
Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

Application architectures

- ❑ Client-server
- ❑ Peer-to-peer (P2P)
- ❑ Hybrid of client-server and P2P

Client-server architecture



server:

- always-on host
- permanent IP address
- "server farms" for scaling

clients:

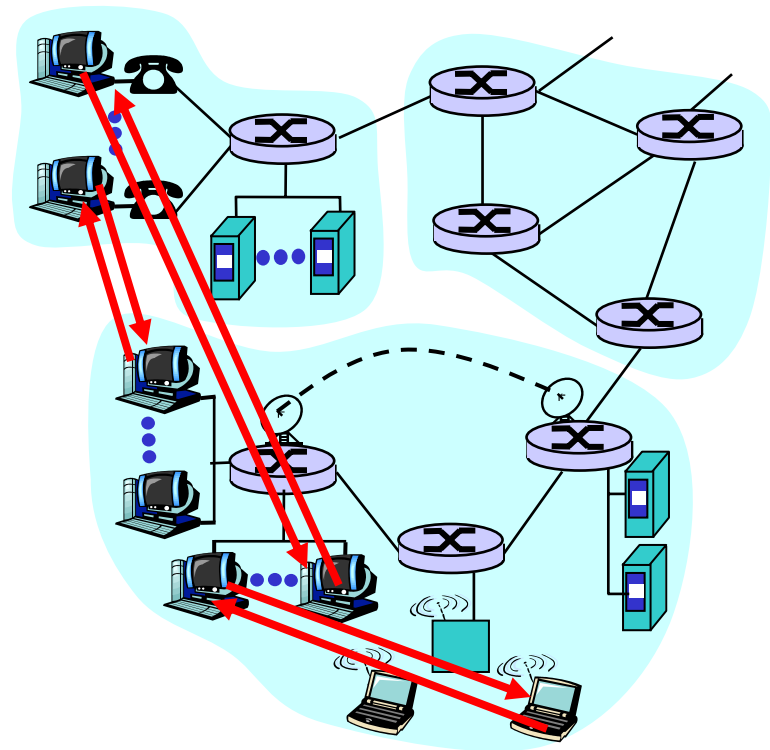
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- ❑ no always on server
- ❑ arbitrary end systems directly communicate
- ❑ peers are intermittently connected and may change IP addresses
- ❑ example: Gnutella

Highly scalable

But difficult to manage



Hybrid of client-server and P2P

Napster

- File transfer P2P
- File search centralized:
 - Peers register content at central server
 - Peers query same central server to locate content

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
 - User registers its IP address with central server when it comes online
 - User contacts central server to find IP addresses of buddies

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**

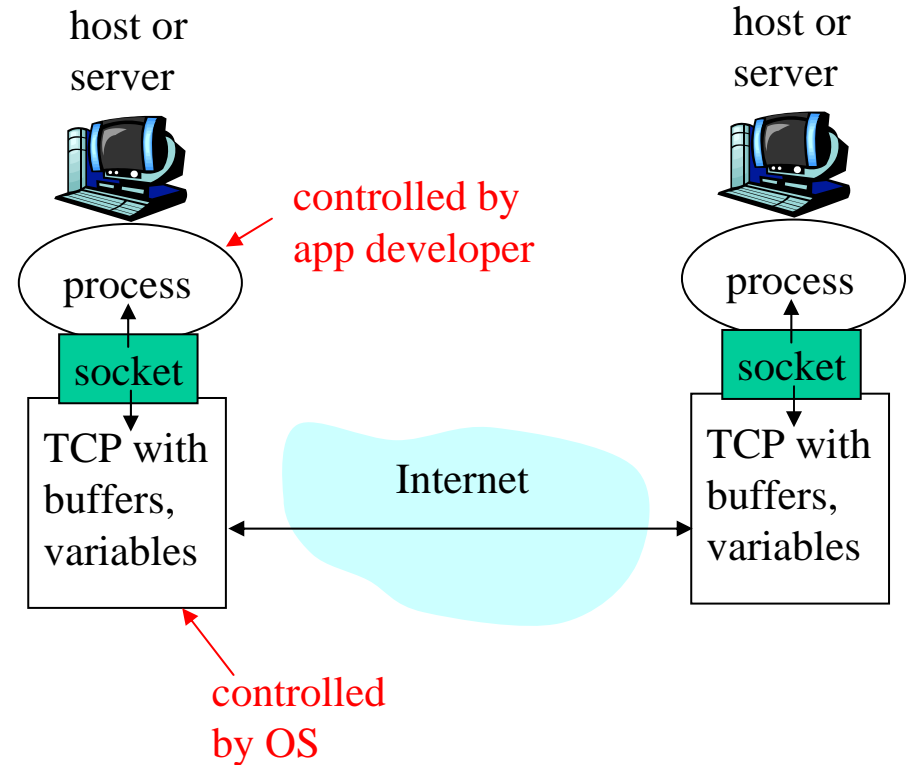
Client process: process that initiates communication

Server process: process that waits to be contacted

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

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process
- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)



Addressing processes

- ❑ For a process to receive messages, it must have an identifier
- ❑ A host has a unique 32-bit IP address
- ❑ **Q:** does the IP address of the host on which the process runs suffice for identifying the process?
- ❑ **Answer:** No, many processes can be running on same host
- ❑ Identifier includes both the IP address and **port numbers** associated with the process on the host.
- ❑ Example port numbers:
 - HTTP server: 80
 - Mail server: 25
- ❑ **More on this later**

App-layer protocol defines

- ❑ Types of messages exchanged, eg, request & response messages
- ❑ Syntax of message types: what fields in messages & how fields are delineated
- ❑ Semantics of the fields, ie, meaning of information in fields
- ❑ Rules for when and how processes send & respond to messages

Public-domain protocols:

- ❑ defined in RFCs
- ❑ allows for interoperability
- ❑ eg, HTTP, SMTP

Proprietary protocols:

- ❑ eg, KaZaA

What transport service does an app need?

Data loss

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

Timing

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

Bandwidth

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

Transport service requirements of common apps

Application	Data loss	Bandwidth	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	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

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

UDP service:

- ❑ unreliable data transfer between sending and receiving process
- ❑ does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
 - app architectures
 - app requirements
- ❑ 2.2 Web and HTTP
- ❑ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

Web and HTTP

First some jargon

- ❑ 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
- ❑ Example URL:

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

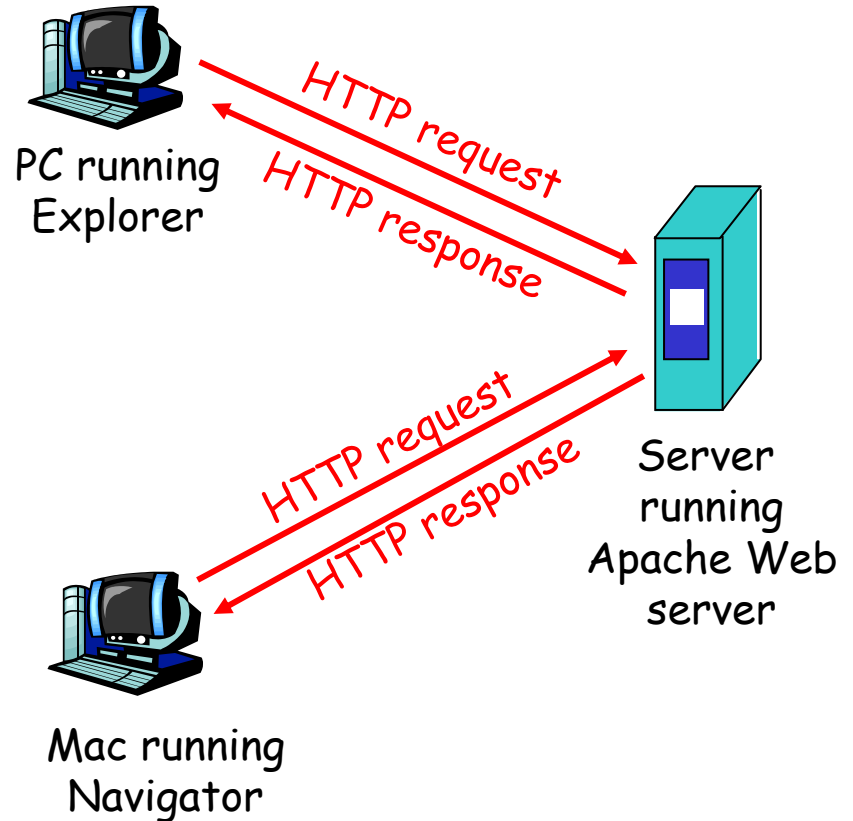
host name

path name

HTTP overview

HTTP: hypertext transfer protocol

- ❑ Web's application layer protocol
- ❑ client/server model
 - *client*: browser that requests, receives, "displays" Web objects
 - *server*: Web server sends objects in response to requests
- ❑ HTTP 1.0: RFC 1945
- ❑ HTTP 1.1: RFC 2068



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

Protocols that maintain "state" are complex! aside

- ❑ past history (state) must be maintained
- ❑ if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP

- ❑ At most one object is sent over a TCP connection.
- ❑ HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

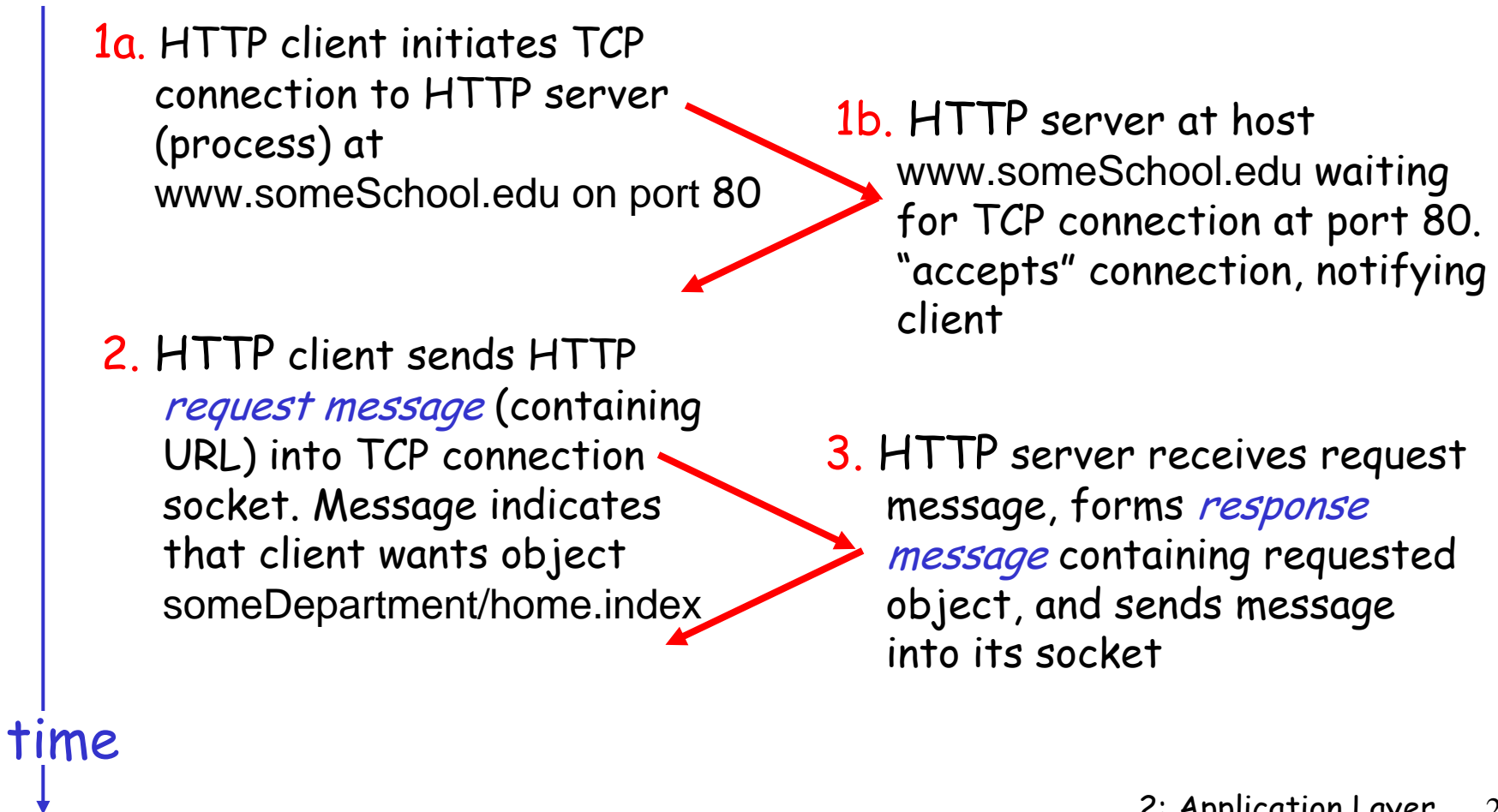
- ❑ Multiple objects can be sent over single TCP connection between client and server.
- ❑ HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

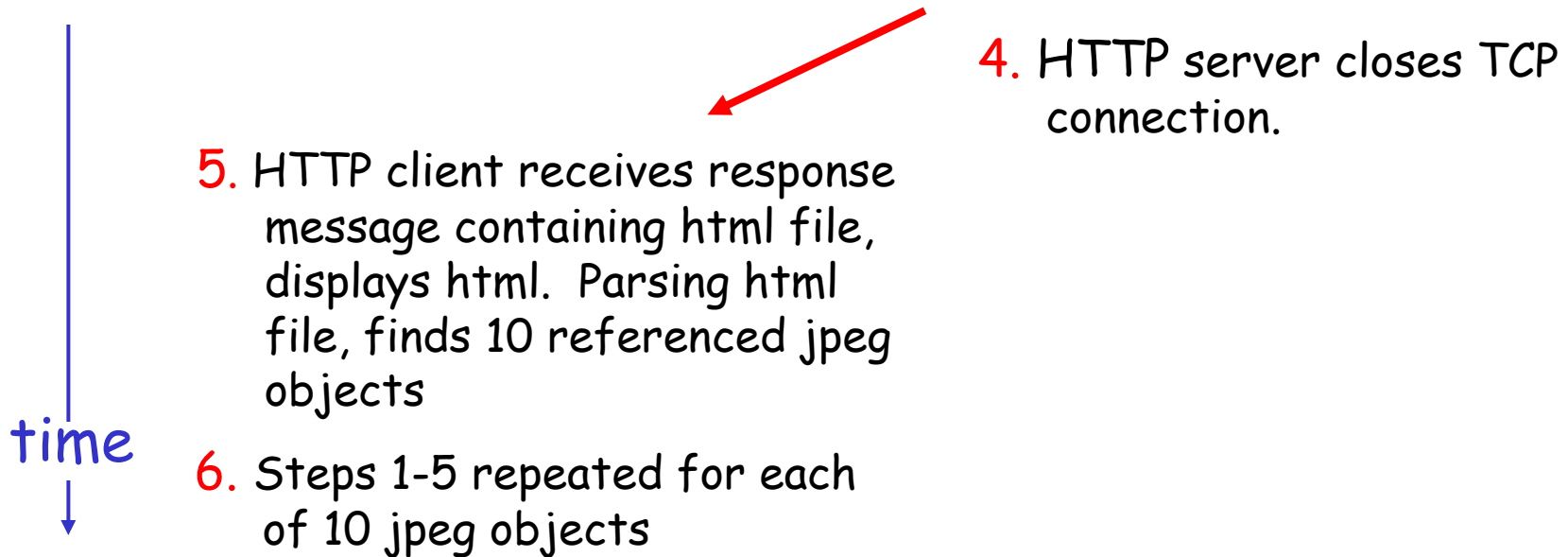
Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

(contains text,
references to 10
jpeg images)



Nonpersistent HTTP (cont.)



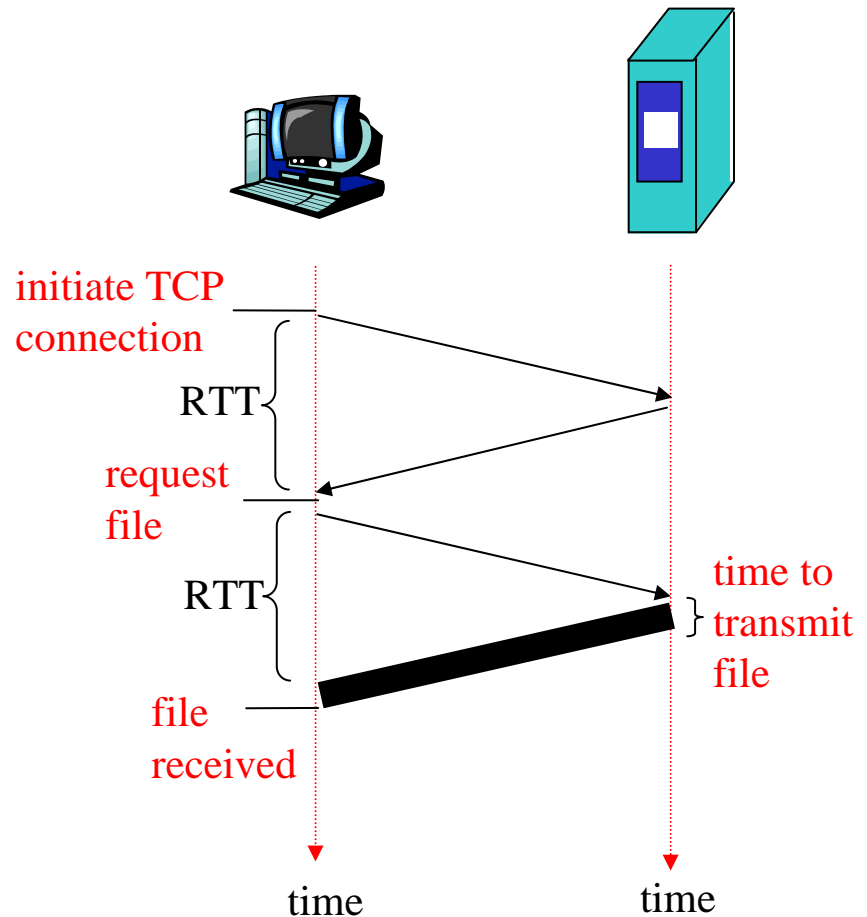
Response time modeling

Definition of RRT: time to send a small packet to travel from client to server and back.

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

total = $2RTT + \text{transmit time}$



Persistent HTTP

Nonpersistent HTTP issues:

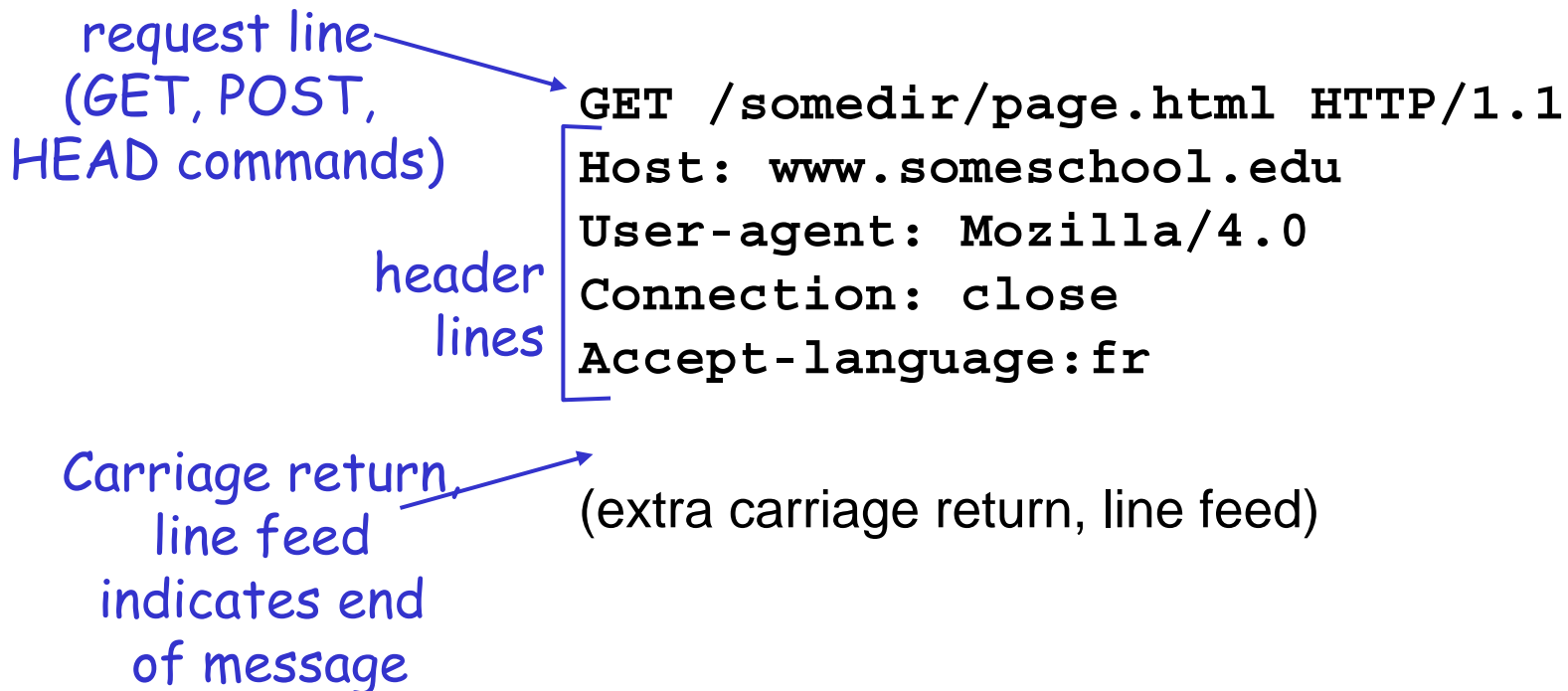
- ❑ requires 2 RTTs per object
- ❑ OS must work and allocate host resources for each TCP connection
- ❑ but browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- ❑ server leaves connection open after sending response
- ❑ subsequent HTTP messages between same client/server are sent over connection

HTTP request message

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



HTTP response message

status line
(protocol
status code
status phrase)

HTTP/1.1 200 OK

header
lines

Connection close

Date: Thu, 06 Aug 1998 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 1998

Content-Length: 6821

Content-Type: text/html

data, e.g.,
requested
HTML file

data data data data data ...

User-server state: cookies

Many major Web sites use cookies

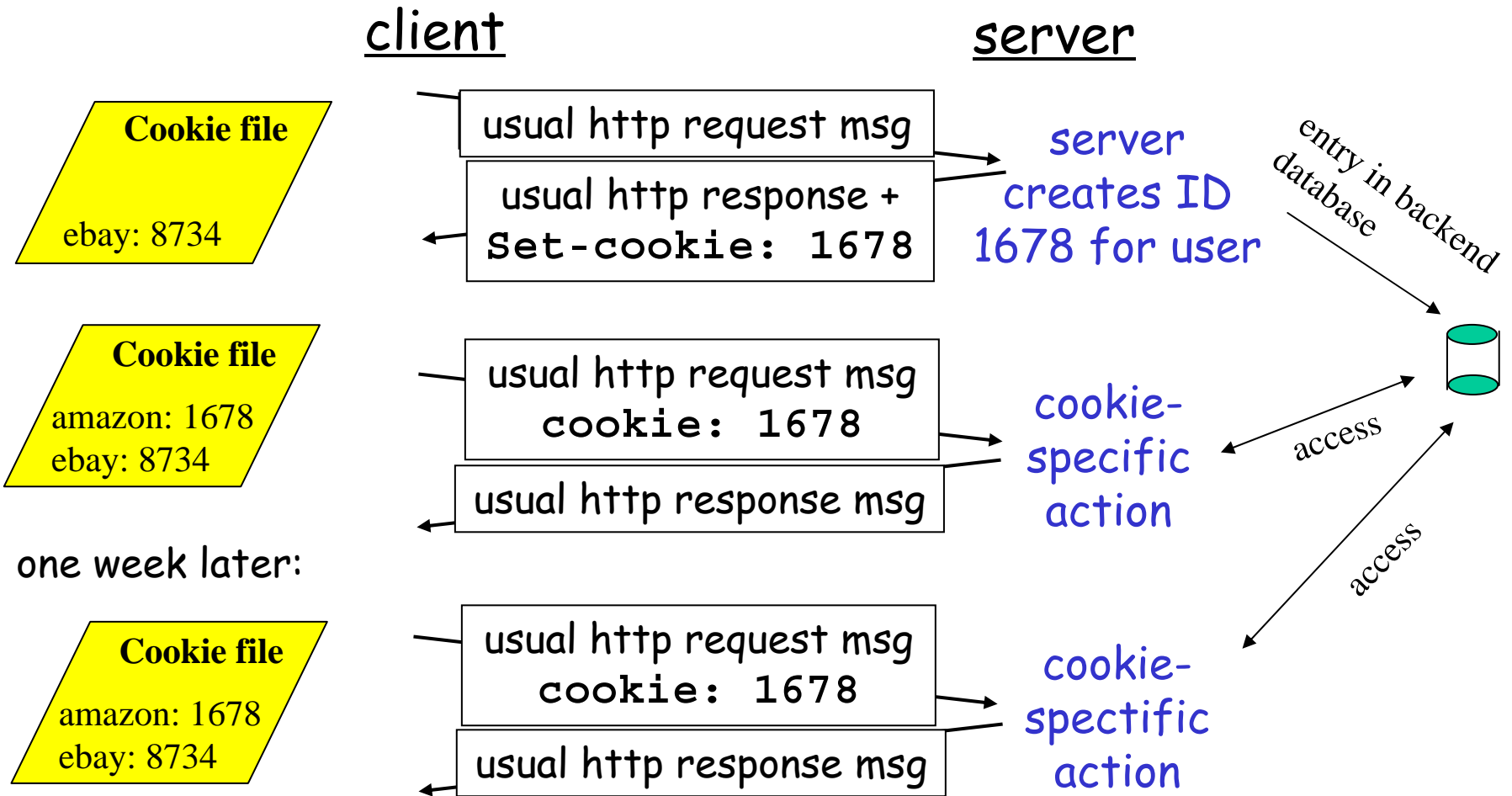
Four components:

- 1) cookie header line in the HTTP response message
- 2) cookie header line in HTTP request message
- 3) cookie file kept on user's host and managed by user's browser
- 4) back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

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

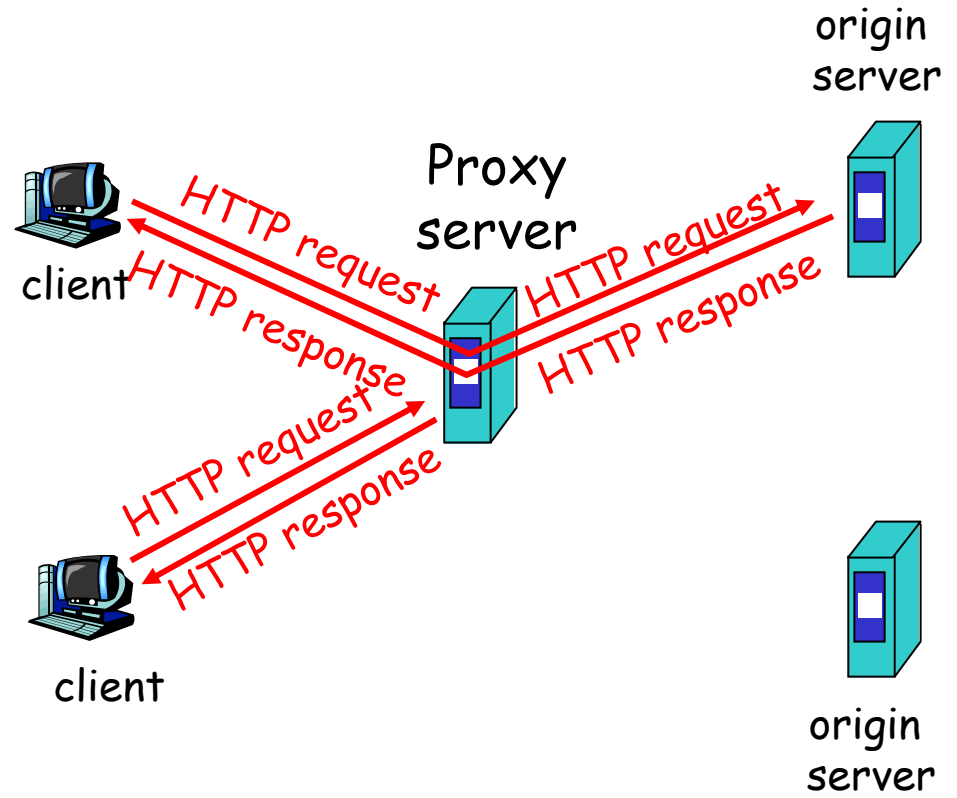
Cookies and privacy: aside

- ❑ cookies permit sites to learn a lot about you
- ❑ you may supply name and e-mail to sites
- ❑ search engines use redirection & cookies to learn yet more
- ❑ advertising companies obtain info across sites

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
- ❑ 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 (but so does P2P file sharing)

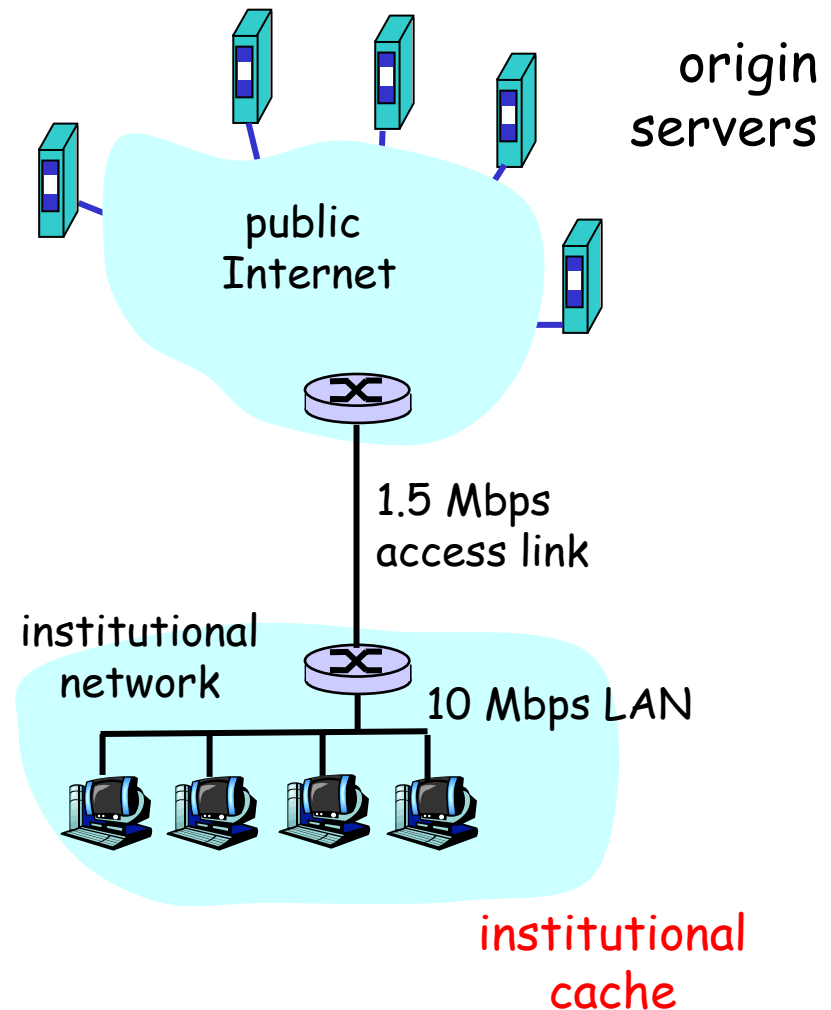
Caching example

Assumptions

- ❑ average object size = 100,000 bits
- ❑ avg. request rate from institution's browsers to origin servers = 15/sec
- ❑ delay from institutional router to any origin server and back to router = 2 sec

Consequences

- ❑ utilization on LAN = 15%
- ❑ utilization on access link = 100%
- ❑ total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + milliseconds



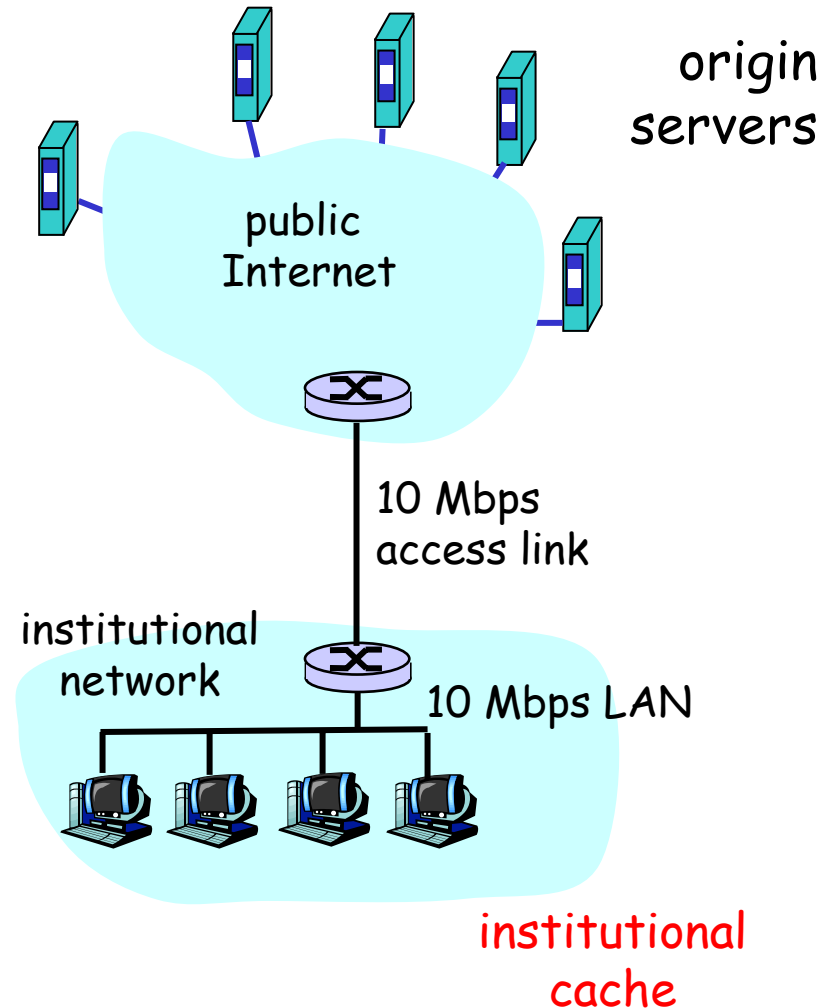
Caching example (cont)

Possible solution

- ❑ increase bandwidth of access link to, say, 10 Mbps

Consequences

- ❑ utilization on LAN = 15%
- ❑ utilization on access link = 15%
- ❑ Total delay = Internet delay + access delay + LAN delay
= 2 sec + msec + msec
- ❑ often a costly upgrade



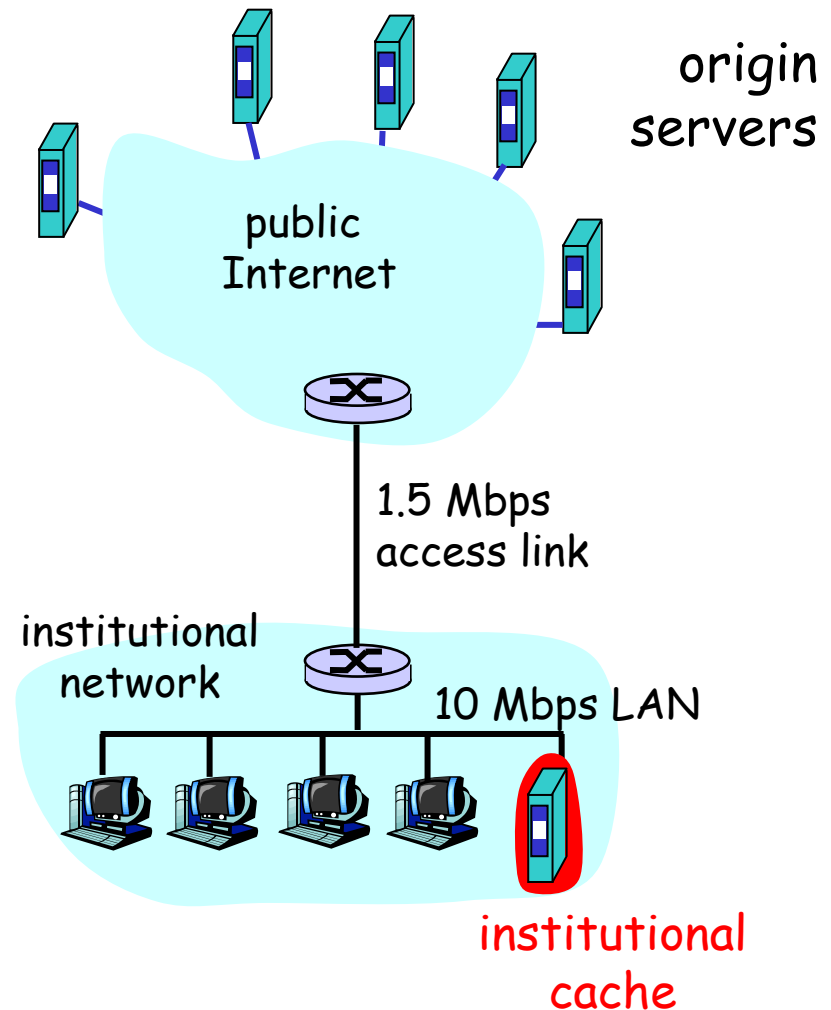
Caching example (cont)

Install cache

- suppose hit rate is .4

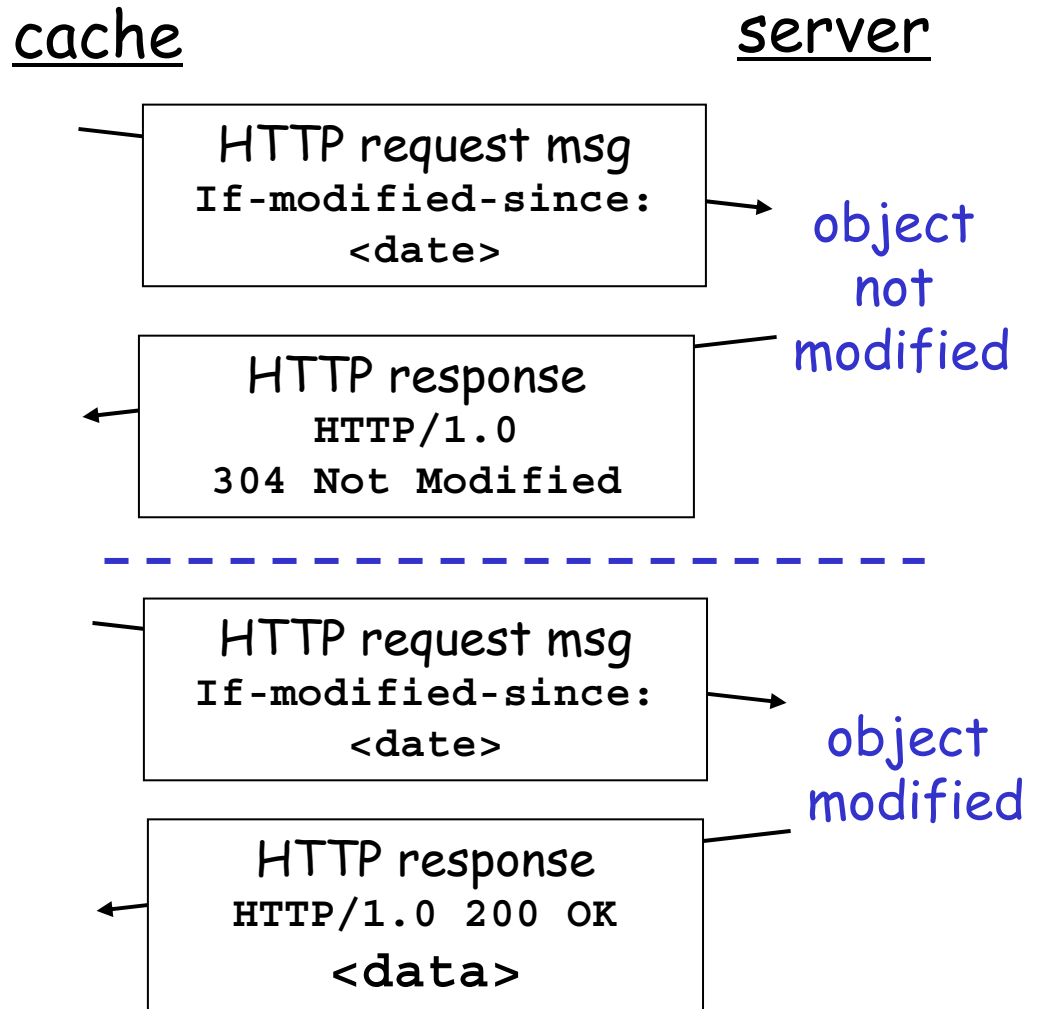
Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = $.6 * (2.01) \text{ secs} + \text{milliseconds} < 1.4 \text{ secs}$



Conditional GET

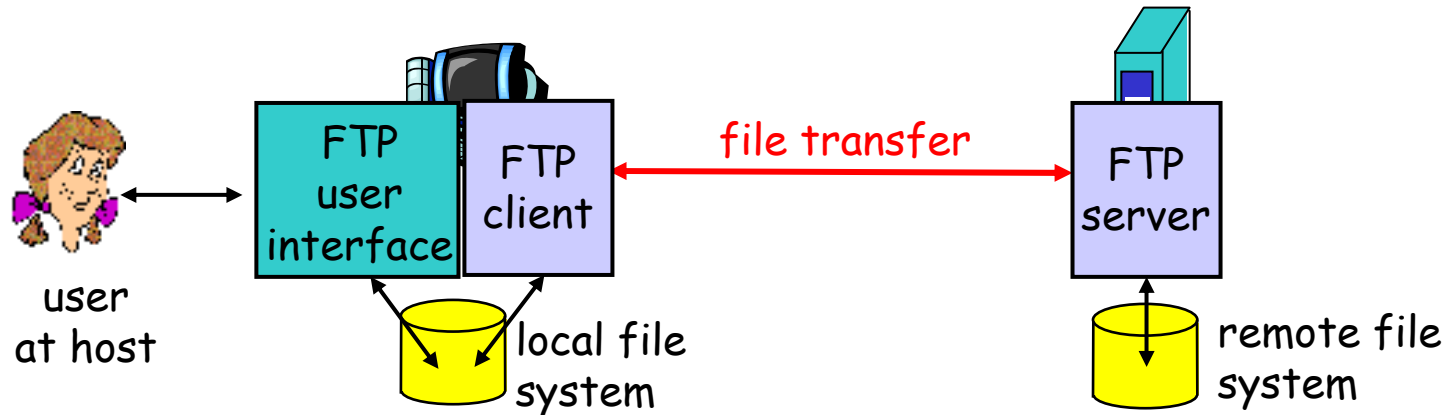
- **Goal:** don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
If-modified-since:
<date>
- server: response contains no object if cached copy is up-to-date:
HTTP/1.0 304 Not Modified



Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

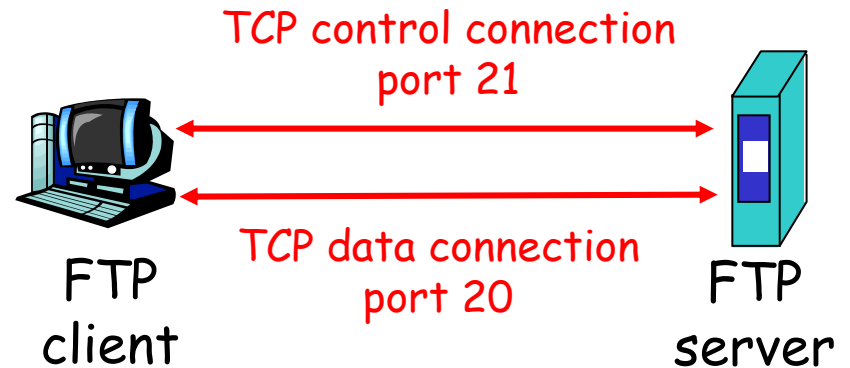
FTP: the file transfer protocol



- ❑ transfer file to/from remote host
- ❑ client/server model
 - *client*: side that initiates transfer (either to/from remote)
 - *server*: remote host
- ❑ ftp: RFC 959
- ❑ ftp server: port 21

FTP: separate control, data connections

- ❑ FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- ❑ Client obtains authorization over control connection
- ❑ Client browses remote directory by sending commands over control connection.
- ❑ When server receives a command for a file transfer, the server opens a TCP data connection to client
- ❑ After transferring one file, server closes connection.



- ❑ Server opens a second TCP data connection to transfer another file.
- ❑ Control connection: "out of band"
- ❑ FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

Sample commands:

- ❑ sent as ASCII text over control channel
- ❑ USER *username*
- ❑ PASS *password*
- ❑ LIST return list of file in current directory
- ❑ RETR *filename* retrieves (gets) file
- ❑ STOR *filename* stores (puts) file onto remote host

Sample return codes

- ❑ status code and phrase (as in HTTP)
- ❑ 331 Username OK, password required
- ❑ 125 data connection already open; transfer starting
- ❑ 425 Can't open data connection
- ❑ 452 Error writing file

Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

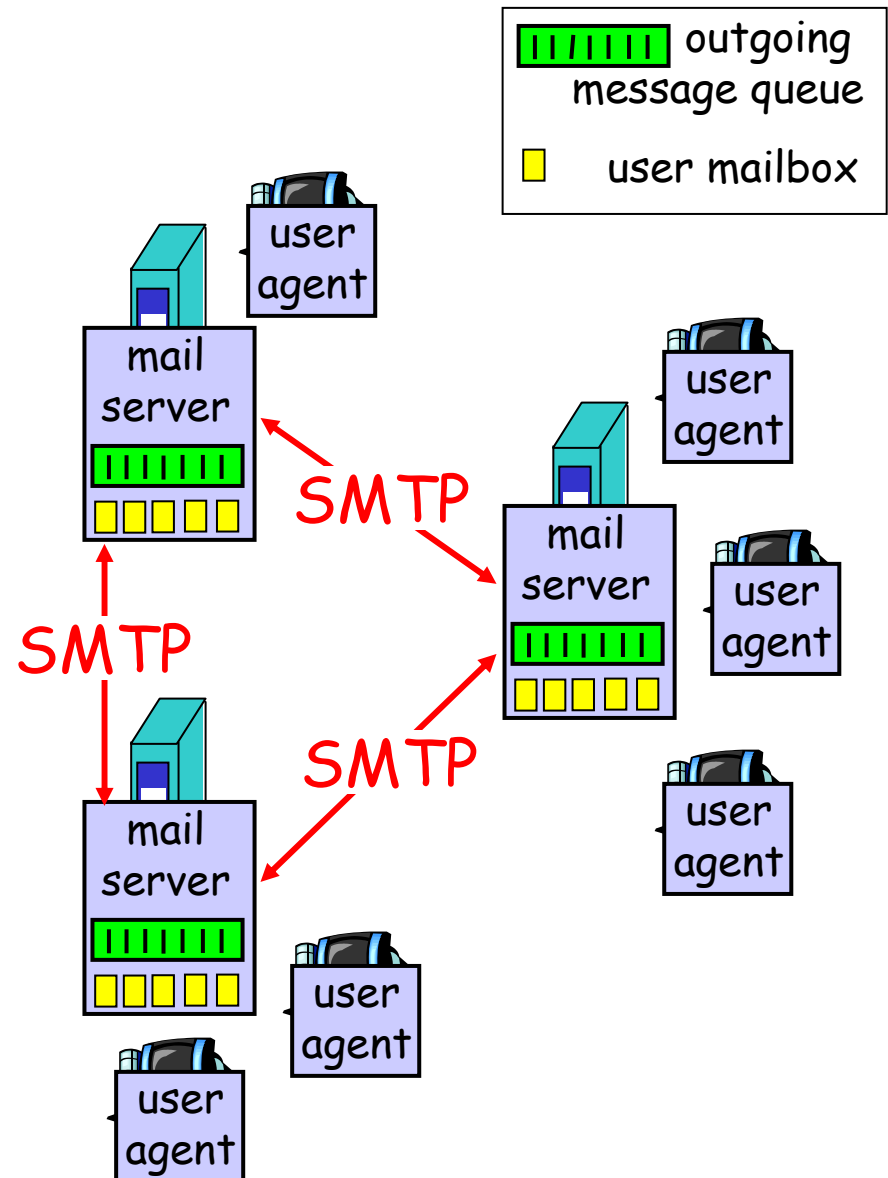
Electronic Mail

Three major components:

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

User Agent

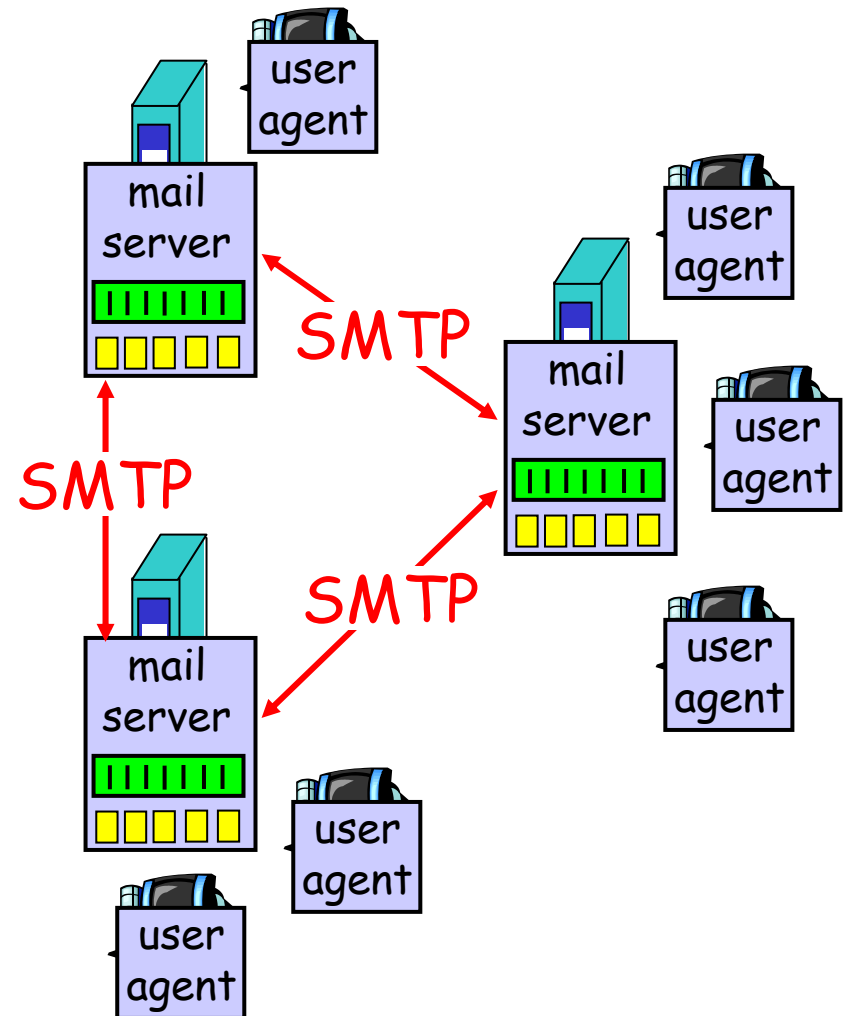
- ❑ a.k.a. "mail reader"
- ❑ composing, editing, reading mail messages
- ❑ e.g., Eudora, Outlook, elm, Netscape Messenger
- ❑ outgoing, incoming messages stored on server



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

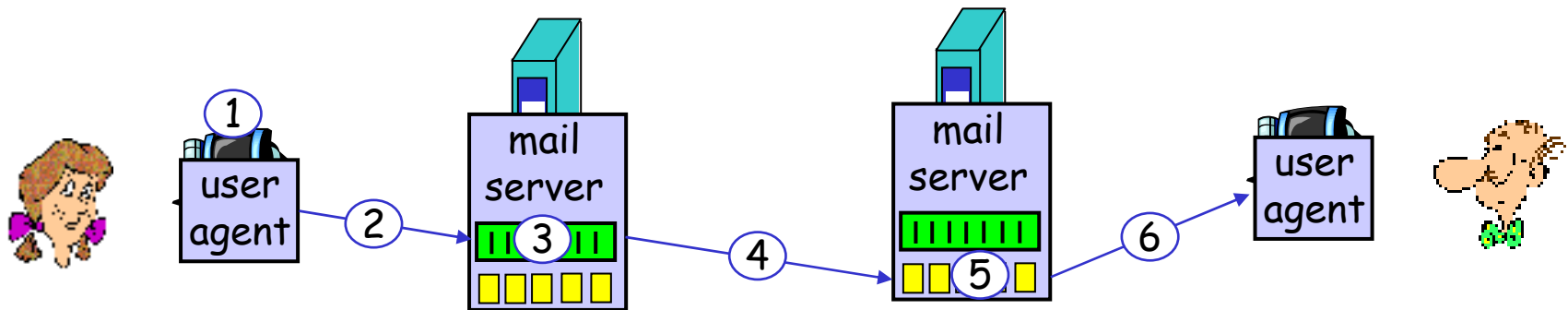


Electronic Mail: SMTP [RFC 2821]

- ❑ 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
 - **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 and "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:

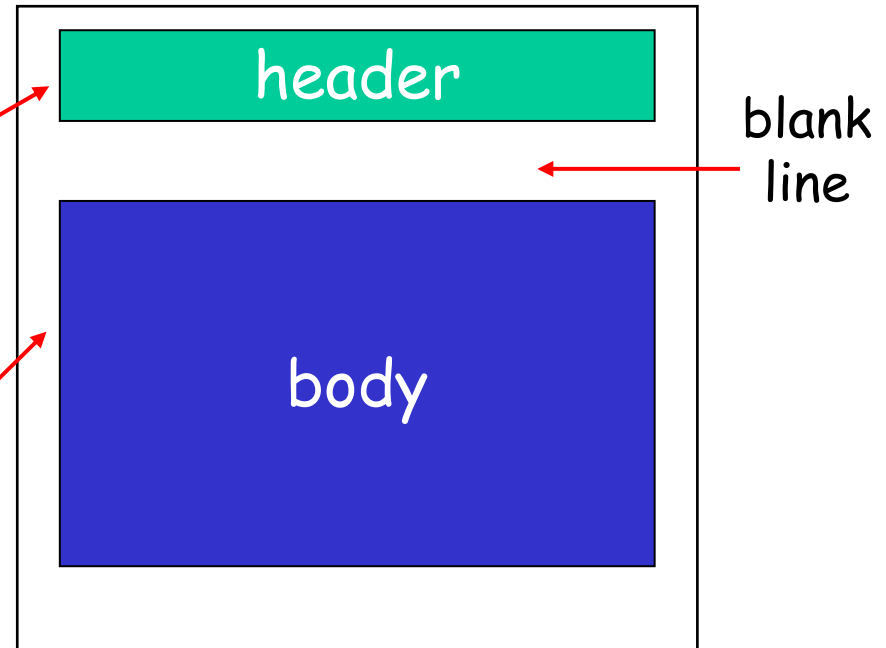
- ❑ HTTP: pull
- ❑ SMTP: push
- ❑ both have ASCII command/response interaction, status codes
- ❑ HTTP: each object encapsulated in its own response msg
- ❑ SMTP: multiple objects sent in multipart msg

Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
 - To:
 - From:
 - Subject:*different from SMTP commands!*
- body
 - the "message", ASCII characters only



Message format: multimedia extensions

- ❑ MIME: multimedia mail extension, RFC 2045, 2056
- ❑ additional lines in msg header declare MIME content type

MIME version

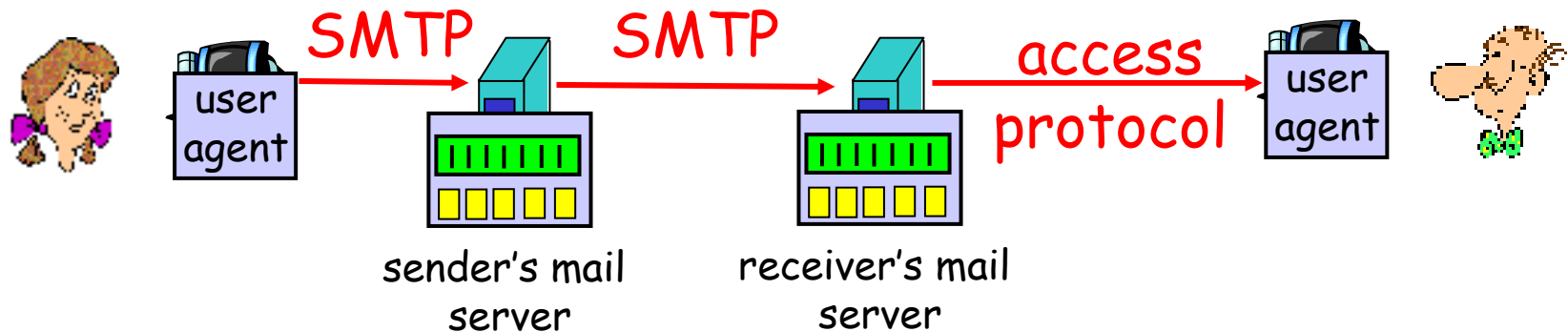
method used
to encode data

multimedia data
type, subtype,
parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.....
.....base64 encoded data
```

Mail access protocols



- ❑ SMTP: delivery/storage to receiver's server
- ❑ Mail access protocol: retrieval from server
 - POP: Post Office Protocol [RFC 1939]
 - authorization (agent <-->server) and download
 - IMAP: Internet Mail Access Protocol [RFC 1730]
 - more features (more complex)
 - manipulation of stored msgs on server
 - HTTP: 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 “download and delete” mode.
- ❑ Bob cannot re-read e-mail if he changes client
- ❑ “Download-and-keep”: copies of messages on different clients
- ❑ POP3 is stateless across sessions

IMAP

- ❑ Keep all messages in one place: the server
- ❑ Allows user to organize messages in folders
- ❑ IMAP keeps user state across sessions:
 - names of folders and mappings between message IDs and folder name