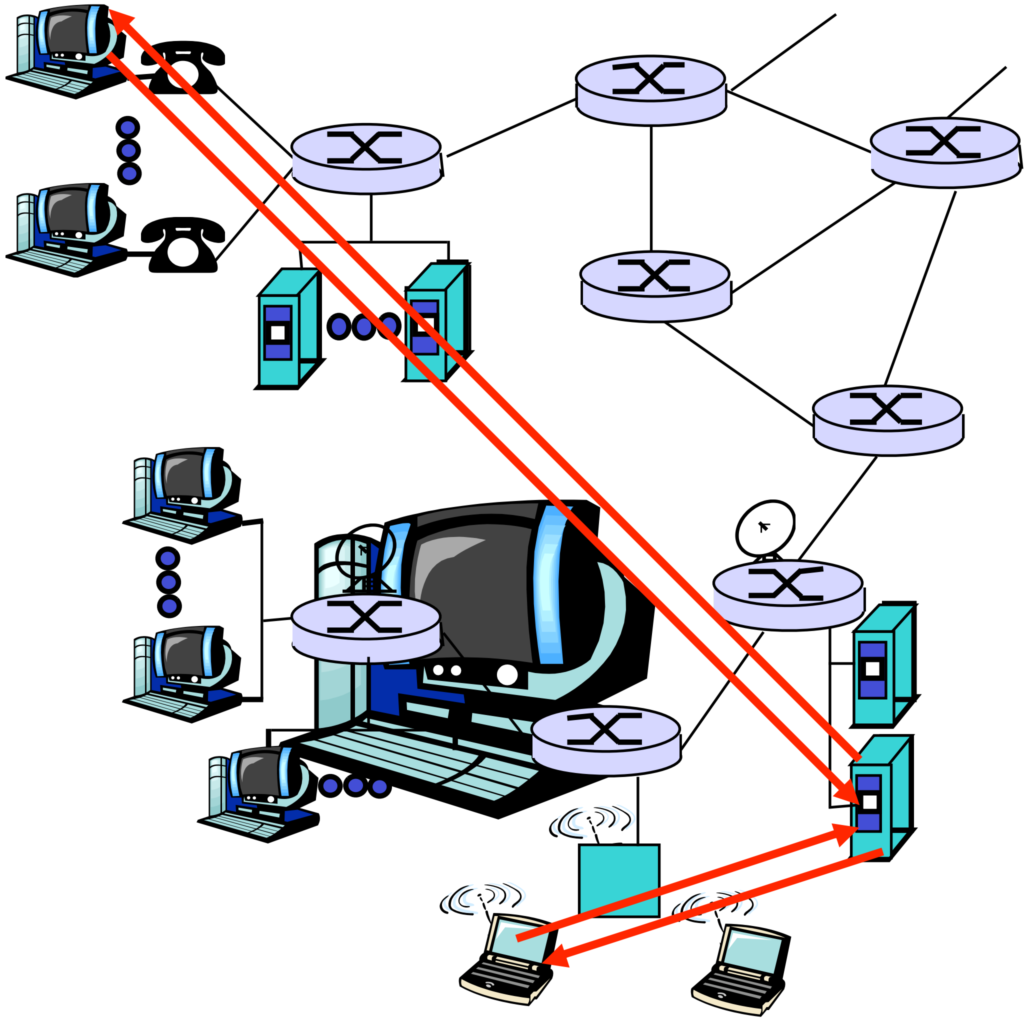
CMSC 332
Computer Networking
Web and FTP

Professor Szajda

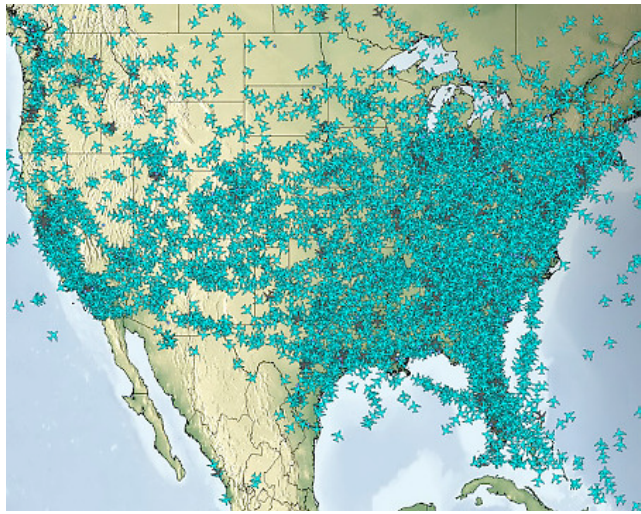
Review

- In the last slide set, we talked about principles of network applications
  - End-to-end argument
  - Network architectures (Client/Server, P2P)
  - Service requirements



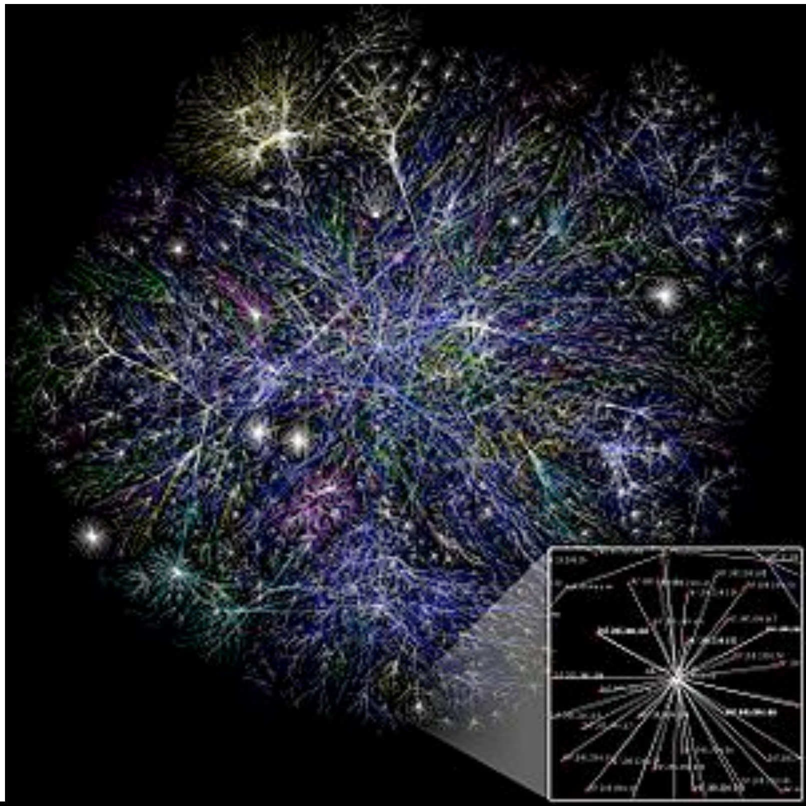
More Info: Bandwidth-Delay Product

- A student asked about the "bandwidth-delay product".
- This is simply the bandwidth of a link multiplied by the end-to-end delay (in seconds).
  - It tells us how many bits are "in the pipe".
- Example: If we have a 10Mbps link between here and Berkeley (with a 100ms delay), what is the bandwidth-delay product?
  - 10Mbps * 1/10sec = 1 Mb



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7-2.8 Sockets
- 2.9 Building a webserver



Web and HTTP

First some jargon

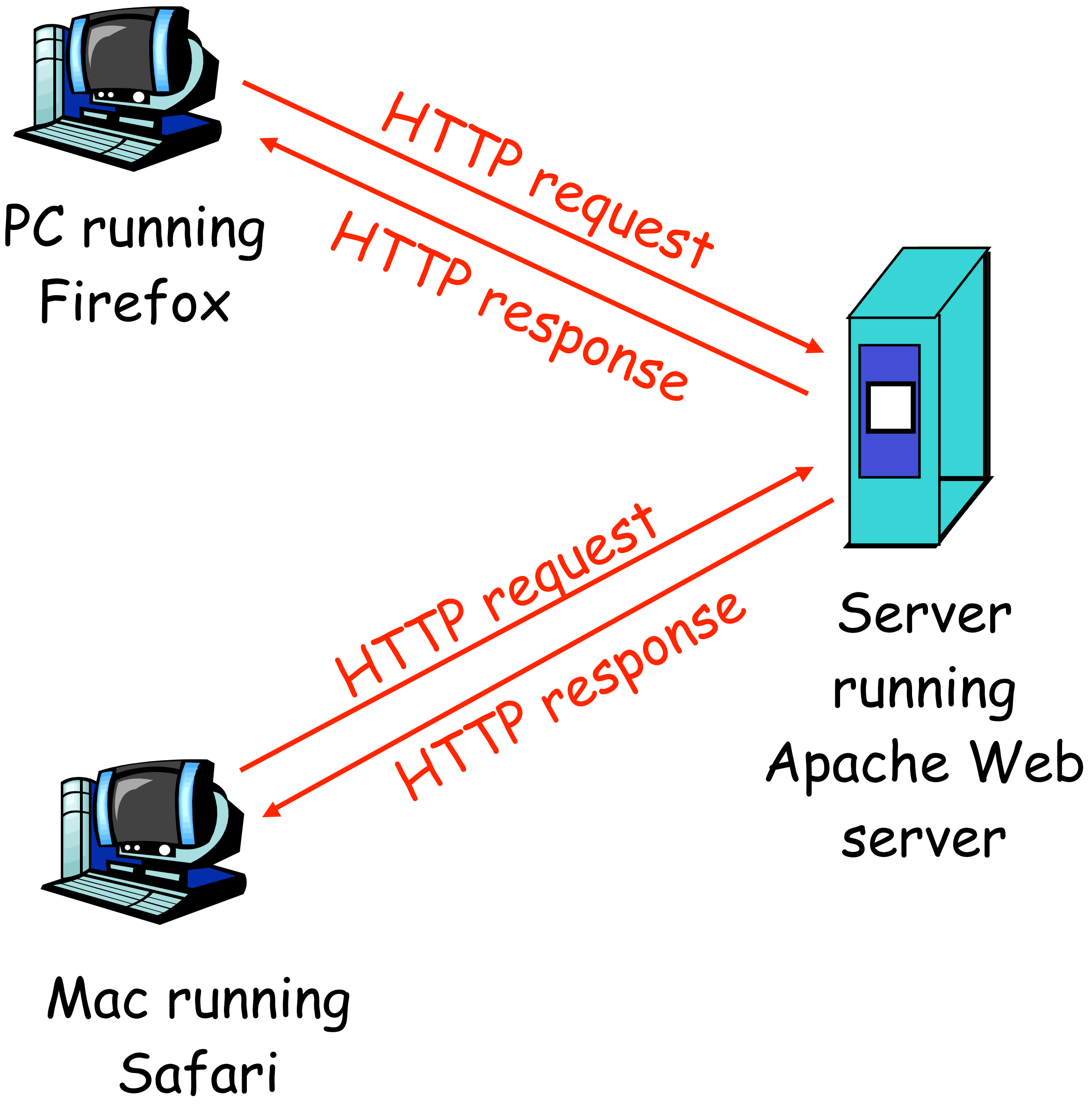
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file, …
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol (RFCs 1945, 2616)
- client/server model
  - client: browser that requests, receives, "displays" Web objects
  - server: Web server sends 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
  - Well, maybe (see next slide)

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

Nonpersistent HTTP

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

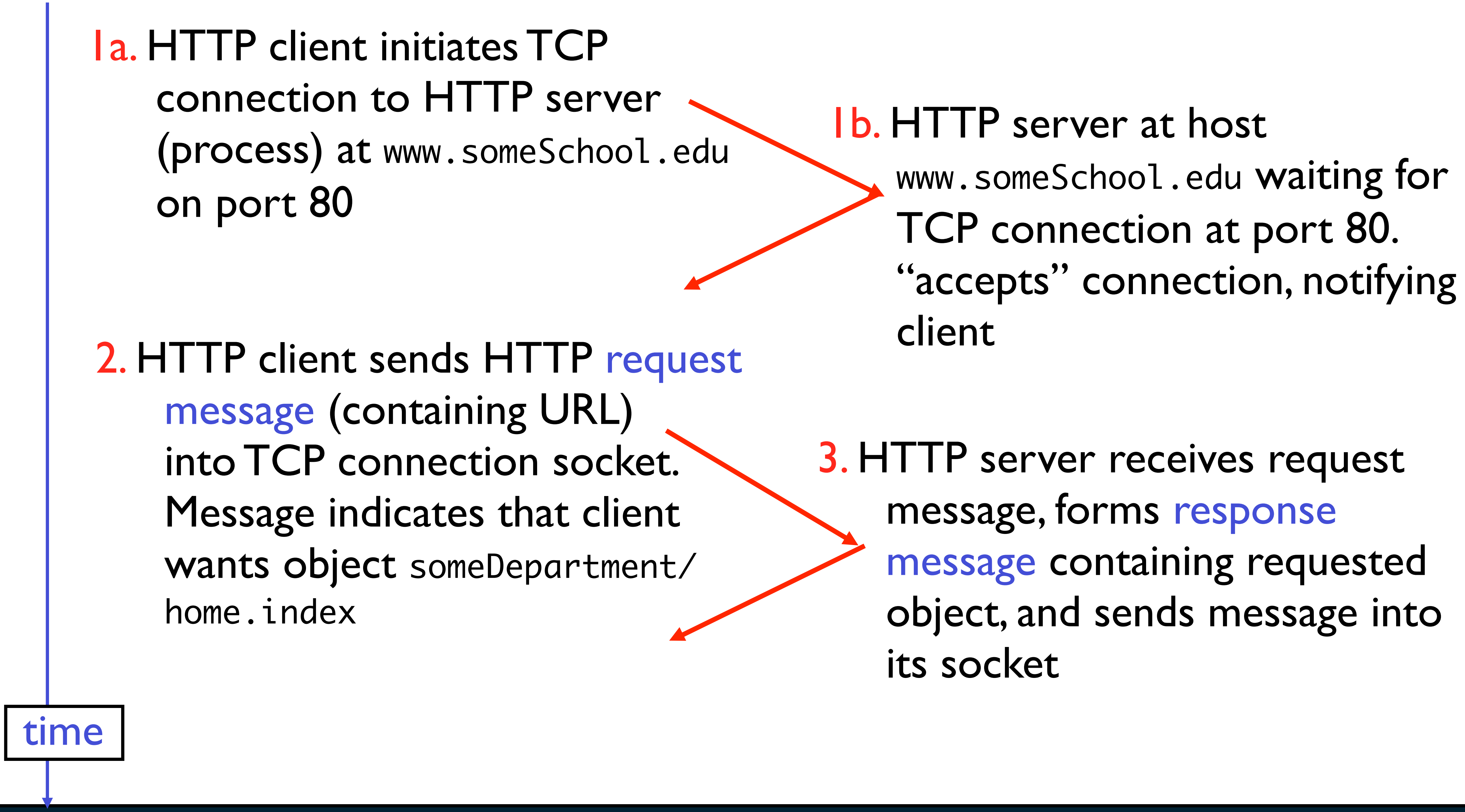
Persistent HTTP

- 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)



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

Nonpersistent 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

time

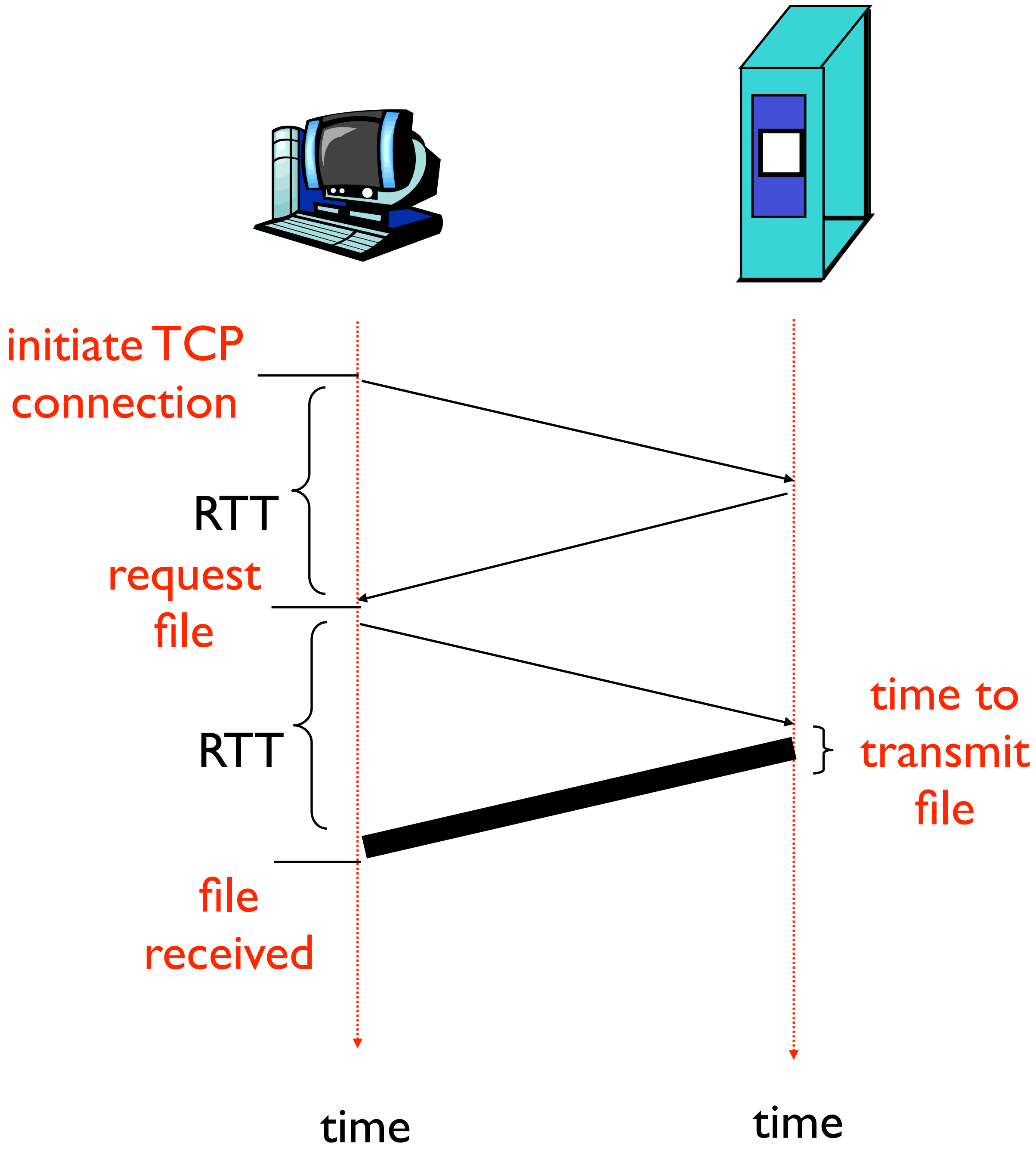
Non-Persistent HTTP: Response time

Definition of RTT: 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+transmit time



Persistent HTTP

Nonpersistent 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

Persistent without pipelining:

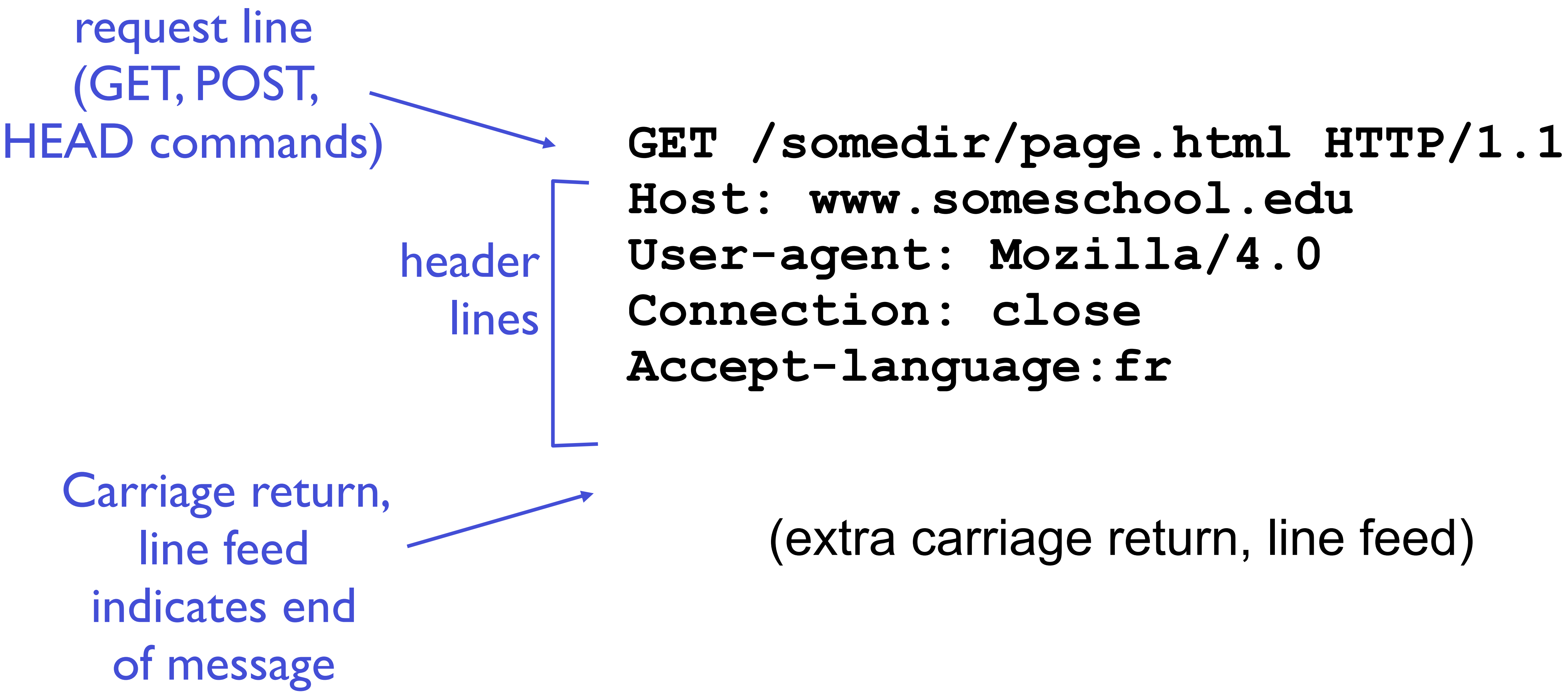
- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT total for all the referenced objects

HTTP request message

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



request line (GET, POST, HEAD commands)

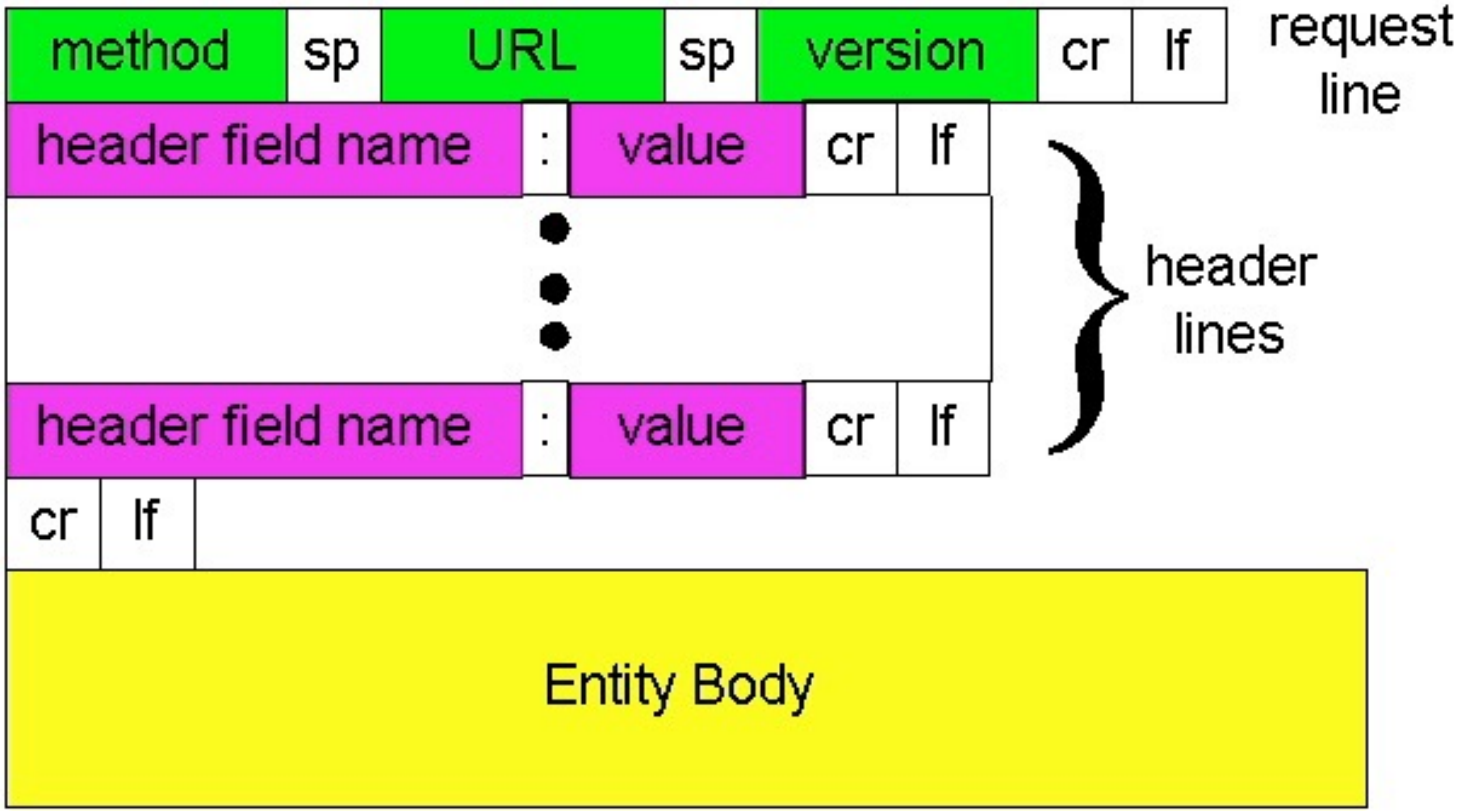
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

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

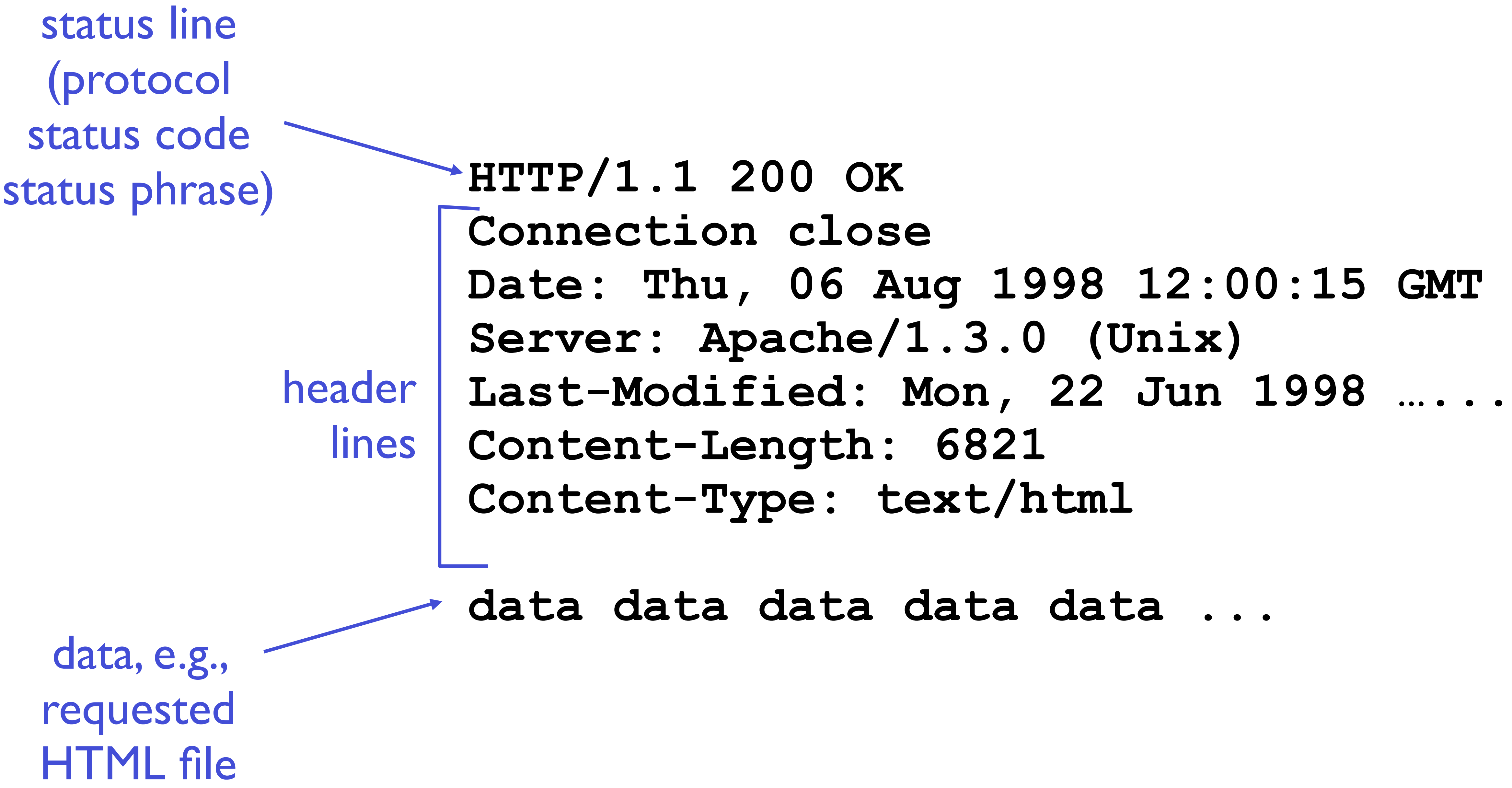
HTTP/1.0

- 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)

```
HTTP/1.1 200 OK
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 data data data data ...
```

header lines

data, e.g., requested HTML file

HTTP response status codes

In first line in server to client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message 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 www.richmond.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at `www.richmond.edu`. Anything typed in sent to port 80 at `www.richmond.edu`

2. Type in a GET HTTP request:

```
GET /~dszajda/classes/cs332/
Spring_2012/index.html HTTP/1.1
Host: www.richmond.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!

User-server state: cookies

Many major Web sites use cookies

Four components:

1) cookie header line of HTTP *response* message

2) cookie header line in HTTP *request* message

3) cookie file kept on user's host, managed by user's browser

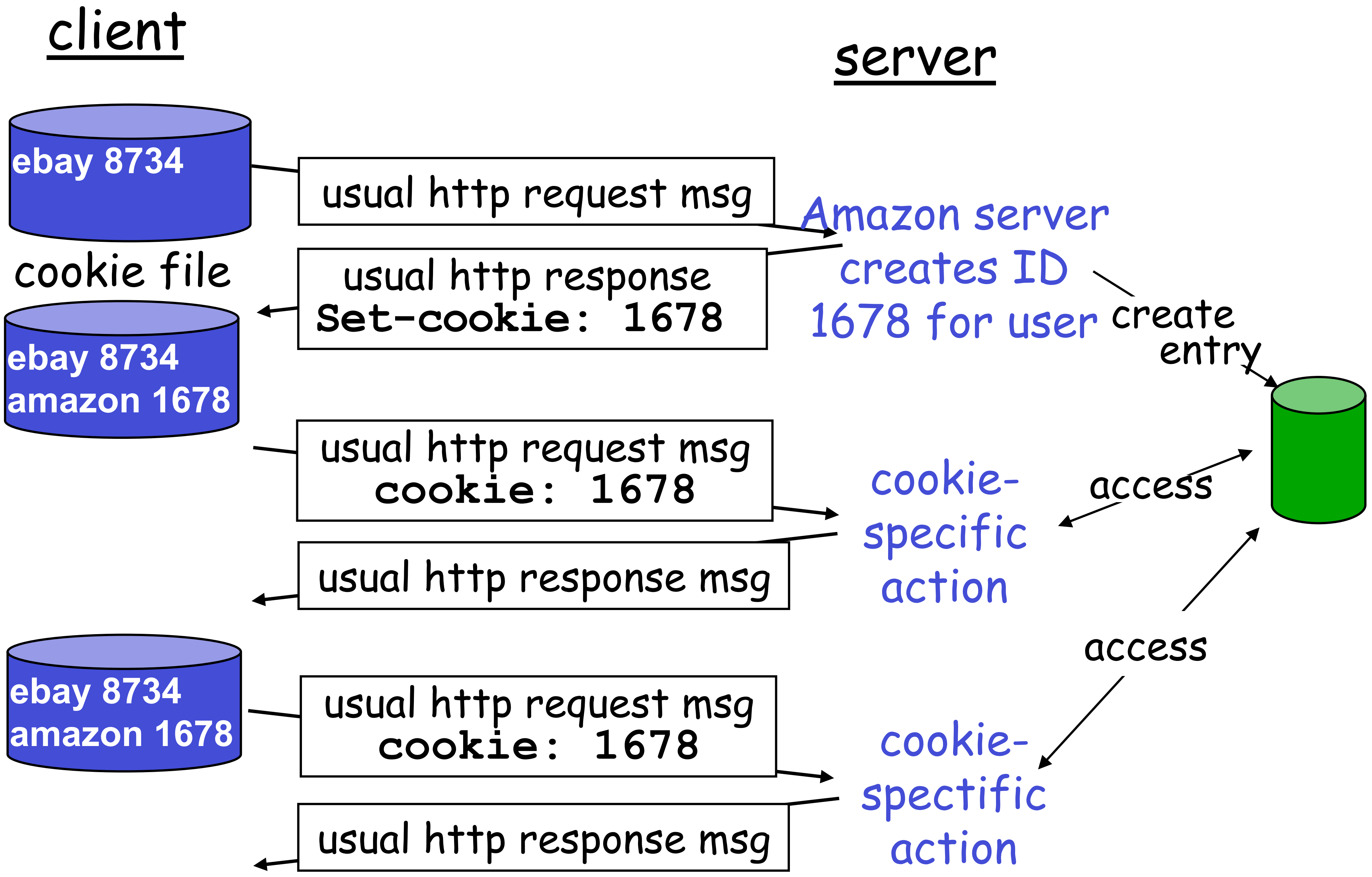
4) back-end database at Web site

Example:

- Susan accesses 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

The Anatomy of Our Cookie
Extra Virgin Olive Oil No Trans-Fats! No Hydrogenated Oils!
Absolutely ZERO Processed or Refined Ingredients In These Cookies!
Fresh Ground Flour is Rich in Nutrients and Rich in Fiber!
Organic Raisins! Free-Range Eggs! Organic Oatmeal!
We use Evaporated Cane Juice. It is Diabetic Friendly. Does not Spike Insulin!
Our Chocolate Chips Have NO Dairy, NO Wax and NO Refined Sugar!

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

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

How to keep "state":

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

aside

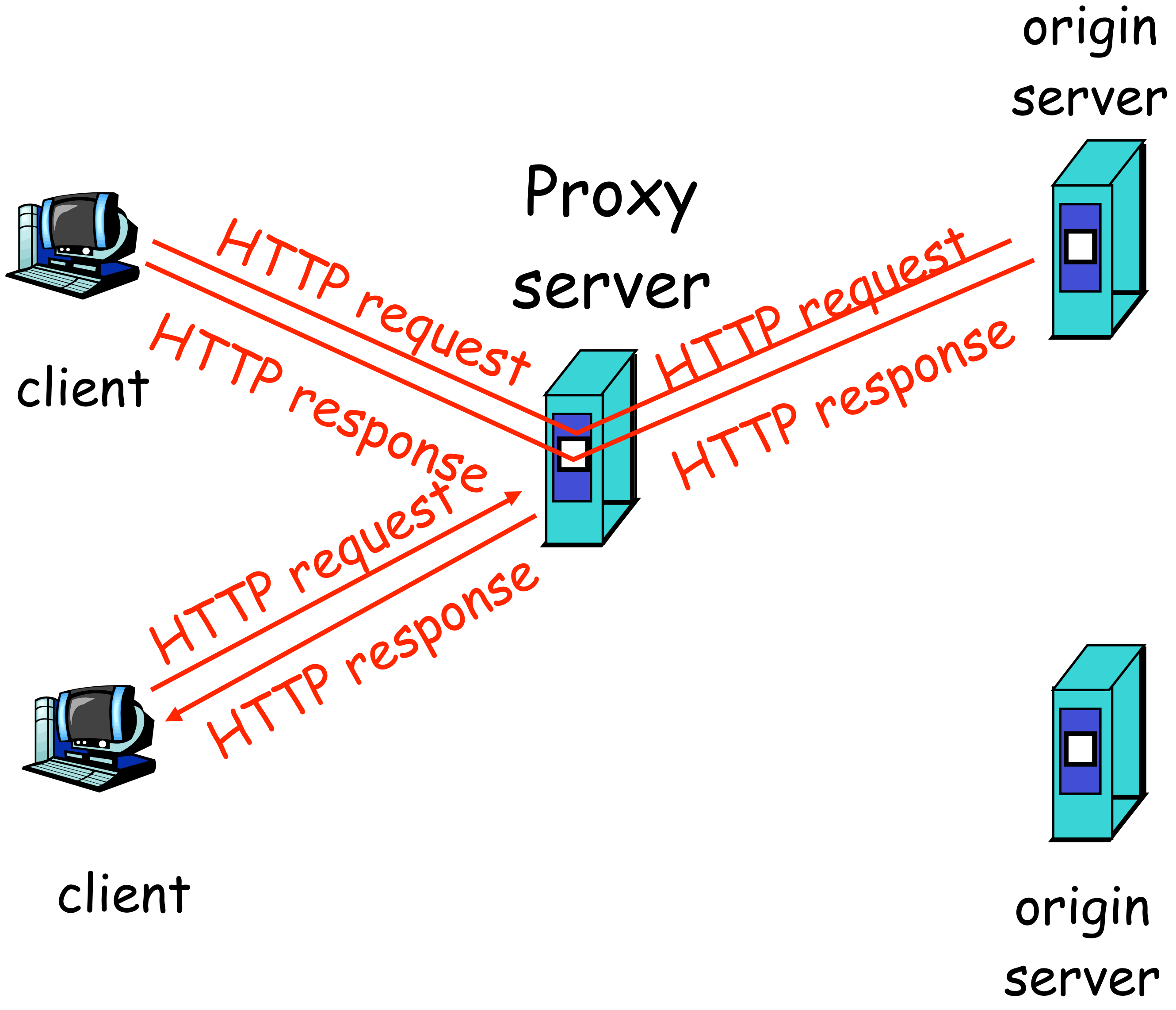
Cookies and privacy:

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

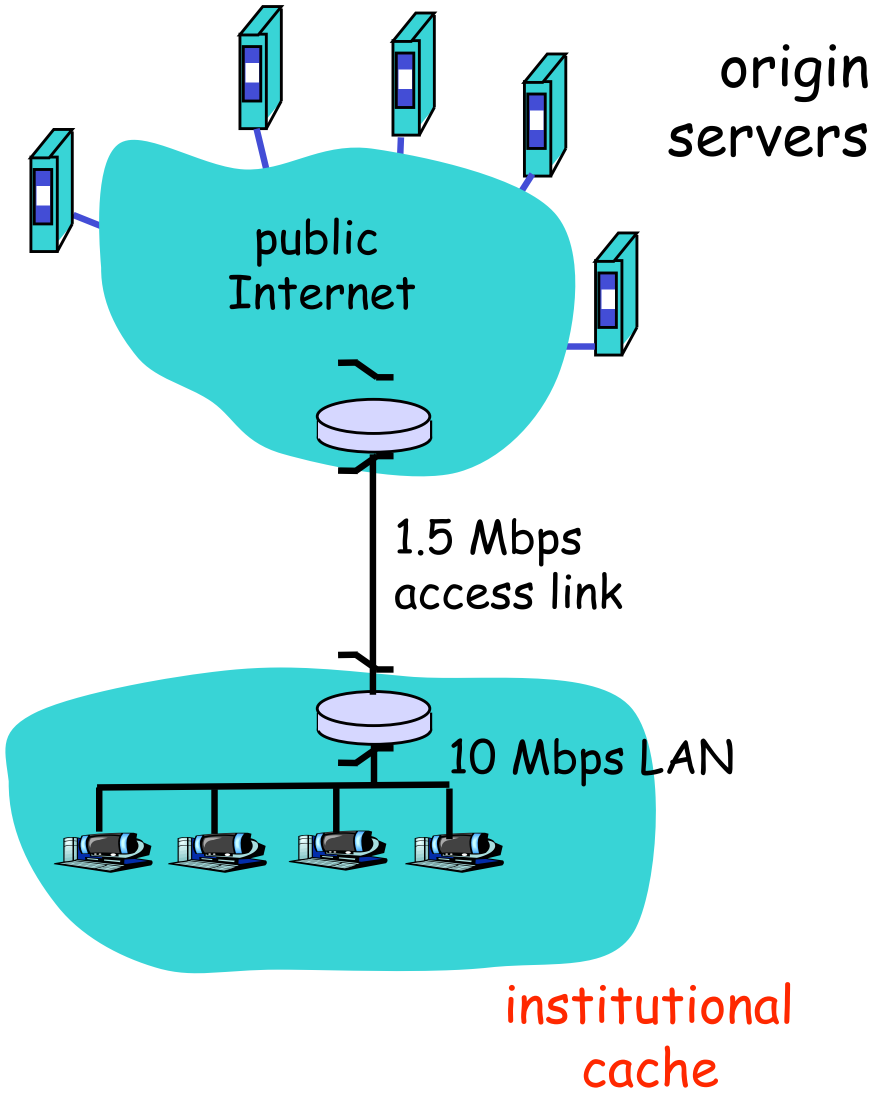
Assumptions

- 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

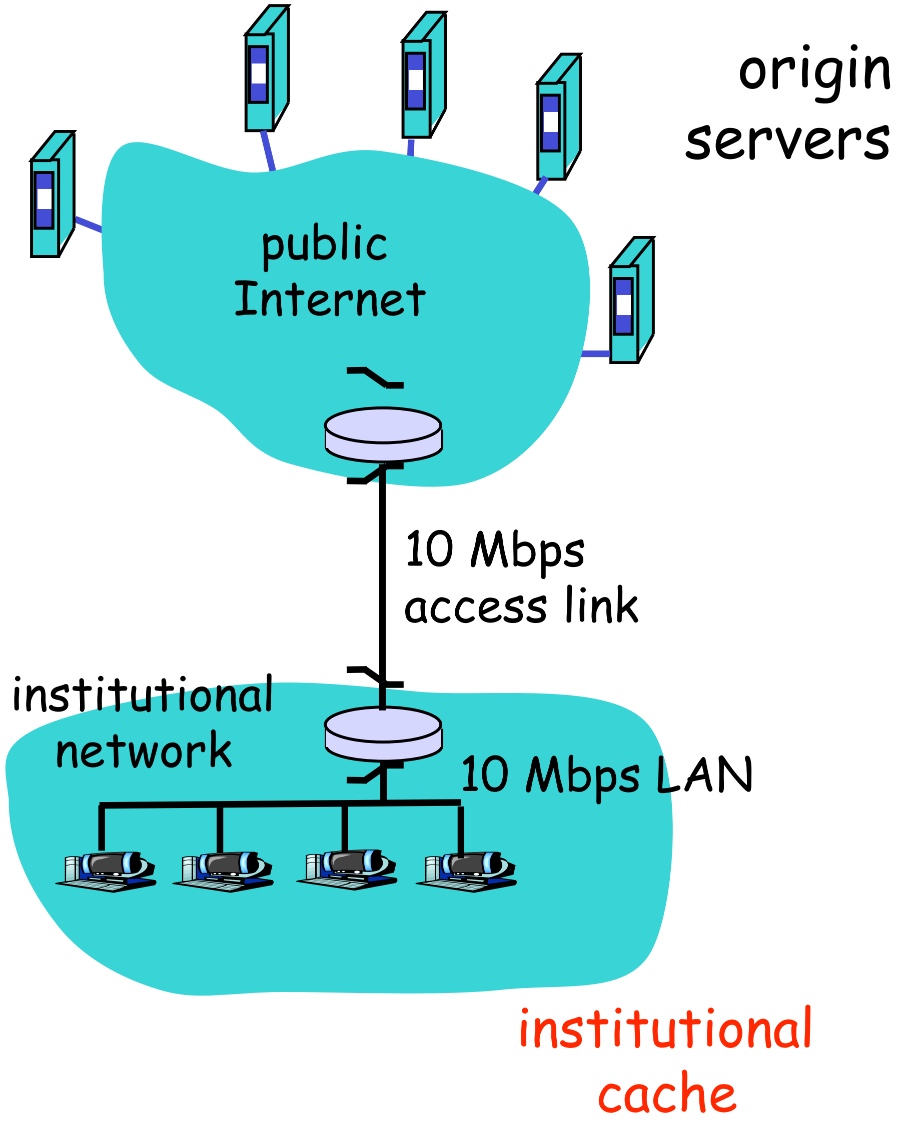
- increase bandwidth of access link to, say, 10 Mbps

Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs

- often a costly upgrade



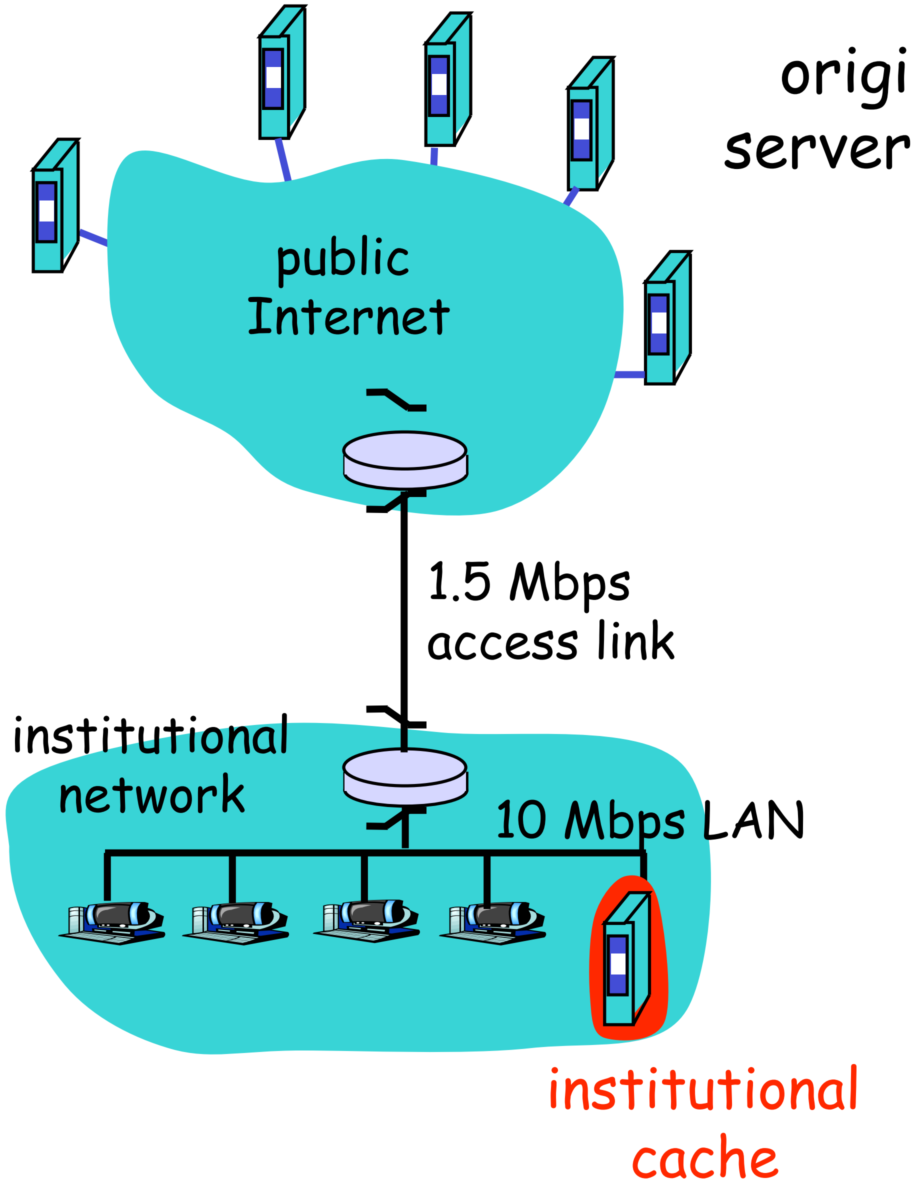
Caching example (cont)

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 = 0.6*(2.01) secs + .4*milliseconds < 1.4 secs



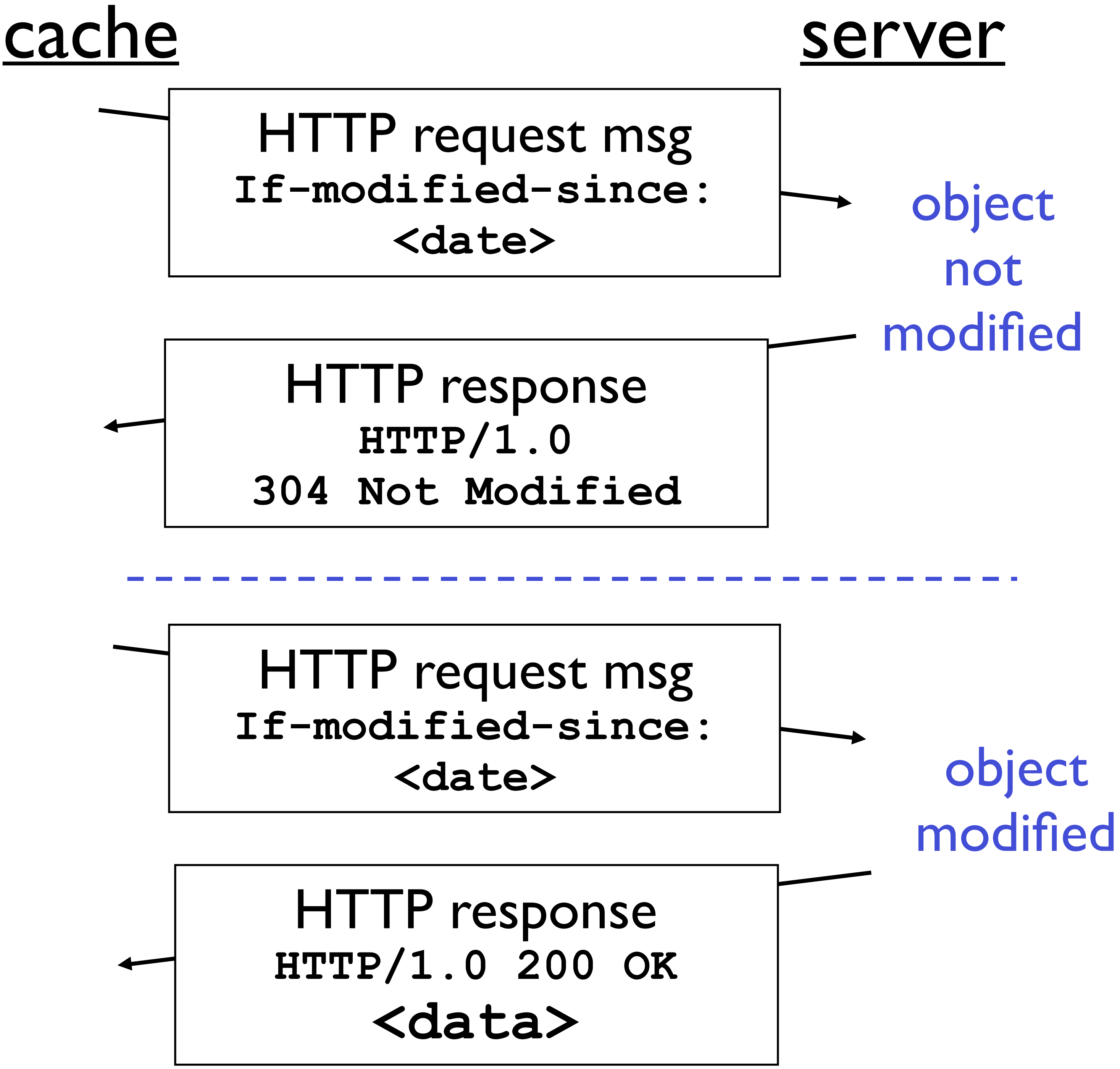
Conditional GET

- 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`



cache

server

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

object not modified

HTTP response
HTTP/1.0
304 Not Modified

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

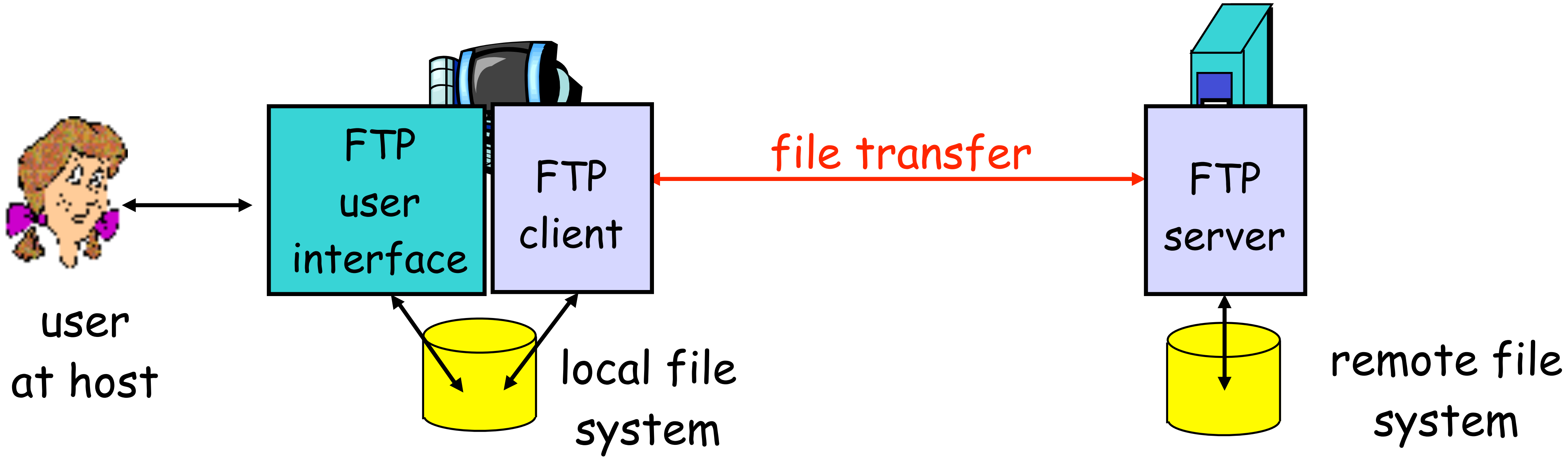
object modified

HTTP response
HTTP/1.0 200 OK
<data>

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7-2.8 Sockets
- 2.9 Building a webserver

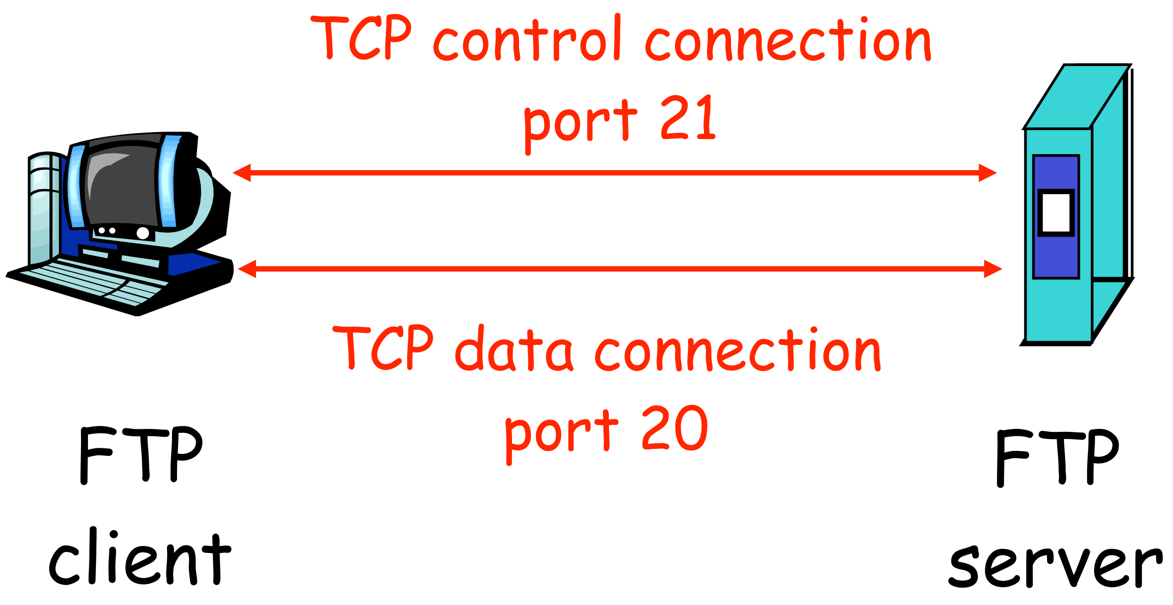
FTP: the file transfer protocol



- 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 file transfer command, server opens 2nd TCP connection (for file) to client
- After transferring one file, server closes data connection.



- Server opens another 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`**

Next Time

- We will cover Email and DNS
  - Read Sections 2.4 and 2.5
- Reminder:
  - Project 1 has been posted

Next is what?™
SAMSUNG mobile