gethostbyname and struct hostent

```
struct hostent *hp;
struct sockaddr_in serv_addr;

hp = gethostbyname("nif-c12.eecs.wsu.edu");
/* from an internal host you could just use "nif-c12" */
/* take the usual precautions about failure:  */
/* gethostbyname returns 0 if the host was not found */

(void) memset((void *)&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
/* now, copy the primary host address to serv_addr */
(void) memcpy( (void *)&serv_addr.sin_addr,
               hp->h_addr, hp->h_length );
serv_addr.sin_port = htons( SERVER_PORT );
```

Chapter 2: Application Layer

Chapter goals:

- conceptual + implementation aspects of network application protocols
  - client server paradigm
  - service models
- learn about protocols by examining popular application-level protocols

More chapter goals

- specific protocols:
  - http
  - ftp
  - smtp
  - pop
  - dns
- programming network applications
  - socket programming

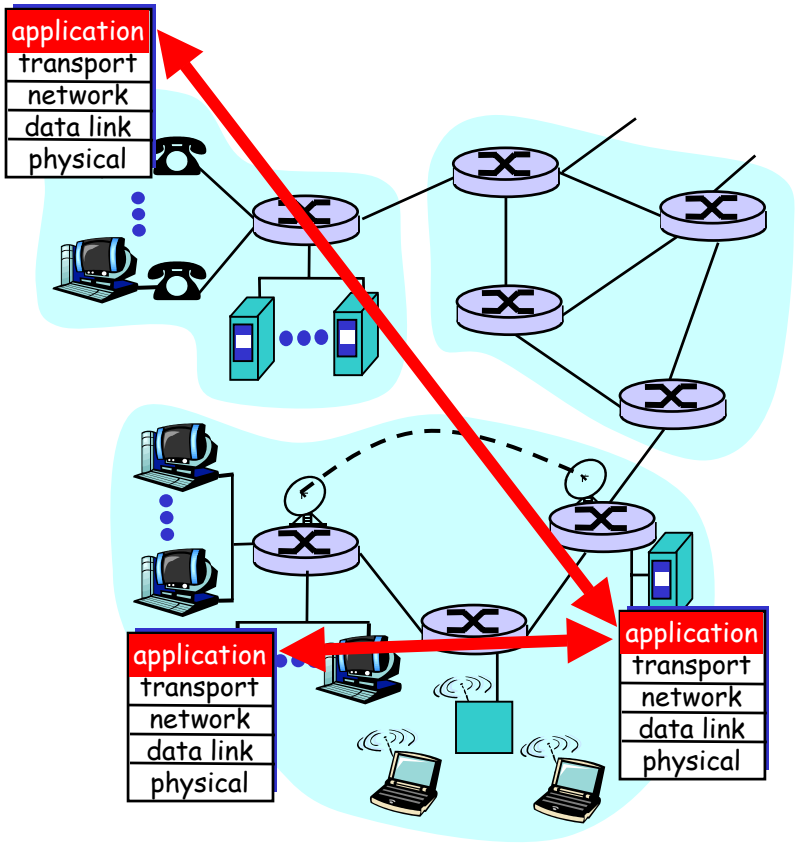
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use services provided by lower layer protocols



Network applications: some jargon

- A process is a program that is running within a host.
- Within the same host, two processes communicate with an application-layer protocol running over an interprocess communication mechanism defined by the OS.
- Processes running in different hosts communicate with an application-layer protocol running over a transport protocol

- A user agent (or client program) is an interface between the user and the network application.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player

Client-server paradigm

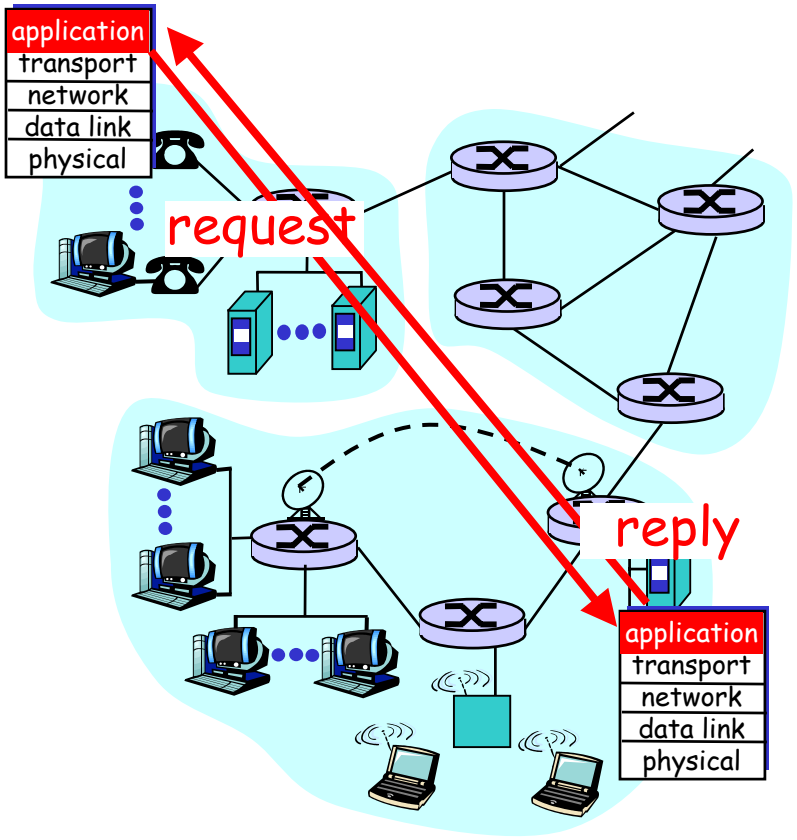
Typical network app has two pieces: *client* and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Application-layer protocols (cont).

API: application programming interface

- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?

- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… lots more on this later.

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

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

Timing

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

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	loss-tolerant	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	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
financial apps	no loss	elastic	yes and no

Services provided by Internet transport protocols

TCP service:

- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not providing:* 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: their protocols and transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

The Web: some jargon

- Web page:
  - consists of "objects"
  - addressed by a URL – uniform resource locator
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects.
- Typical URL has three components: protocol, host name and path name:

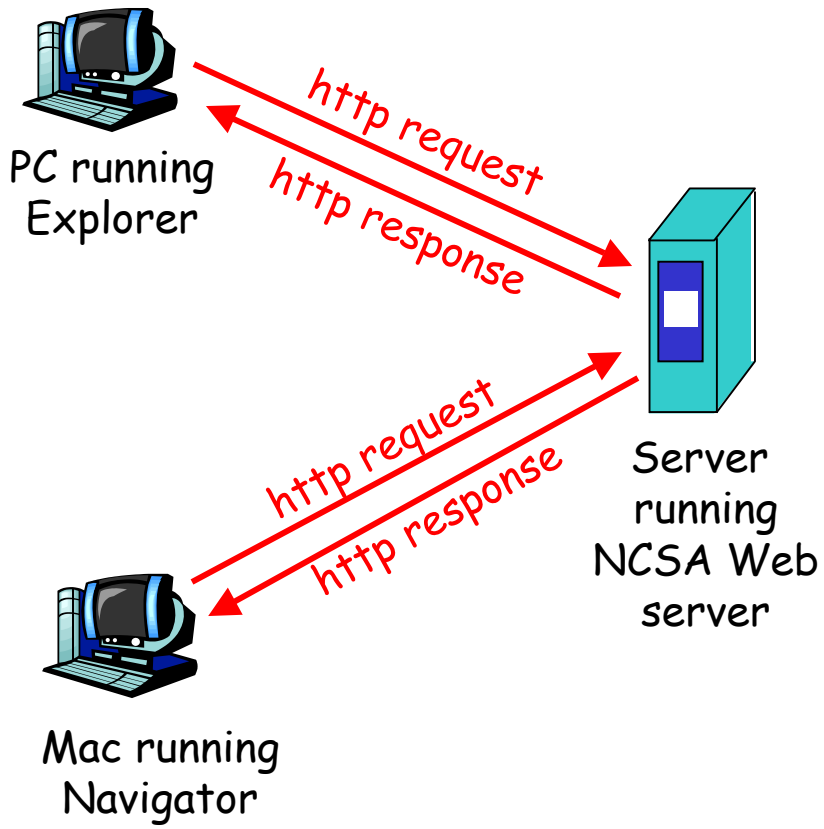
- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
  - Opera
  - Lynx
  - wget
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server

http://www.someSchool.edu/someDept/pic.gif

The Web: the http protocol

http: hypertext transfer protocol

- Web's application layer protocol: defined by W3C – World Wide Web Consortium
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068



The http protocol: more

http: TCP transport service:

- 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

State-ful protocols

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

http example

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

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

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time

http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced image objects

6. Steps 1-5 repeated for each of 10 image objects

time

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start – what does this mean?

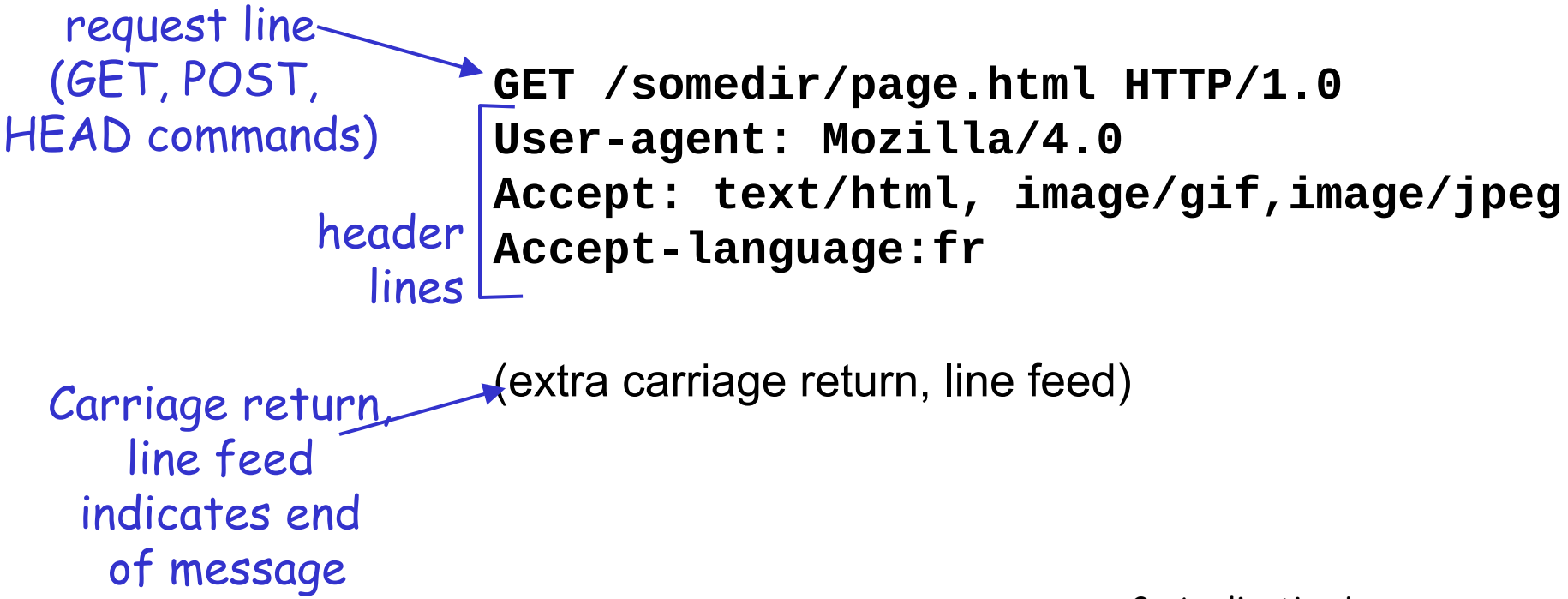
But most 1.0 browsers use concurrent TCP connections.

Persistent

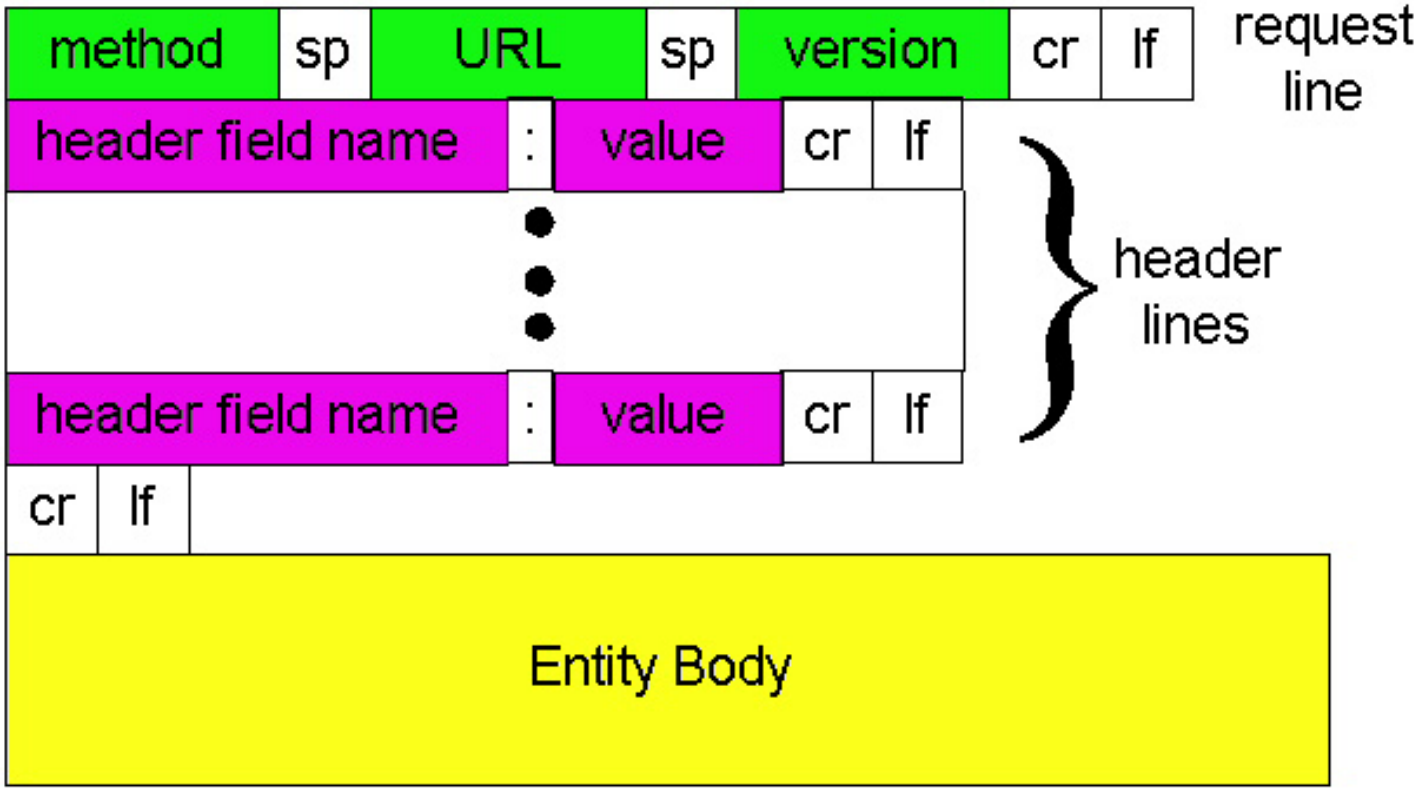
- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

http message format: request

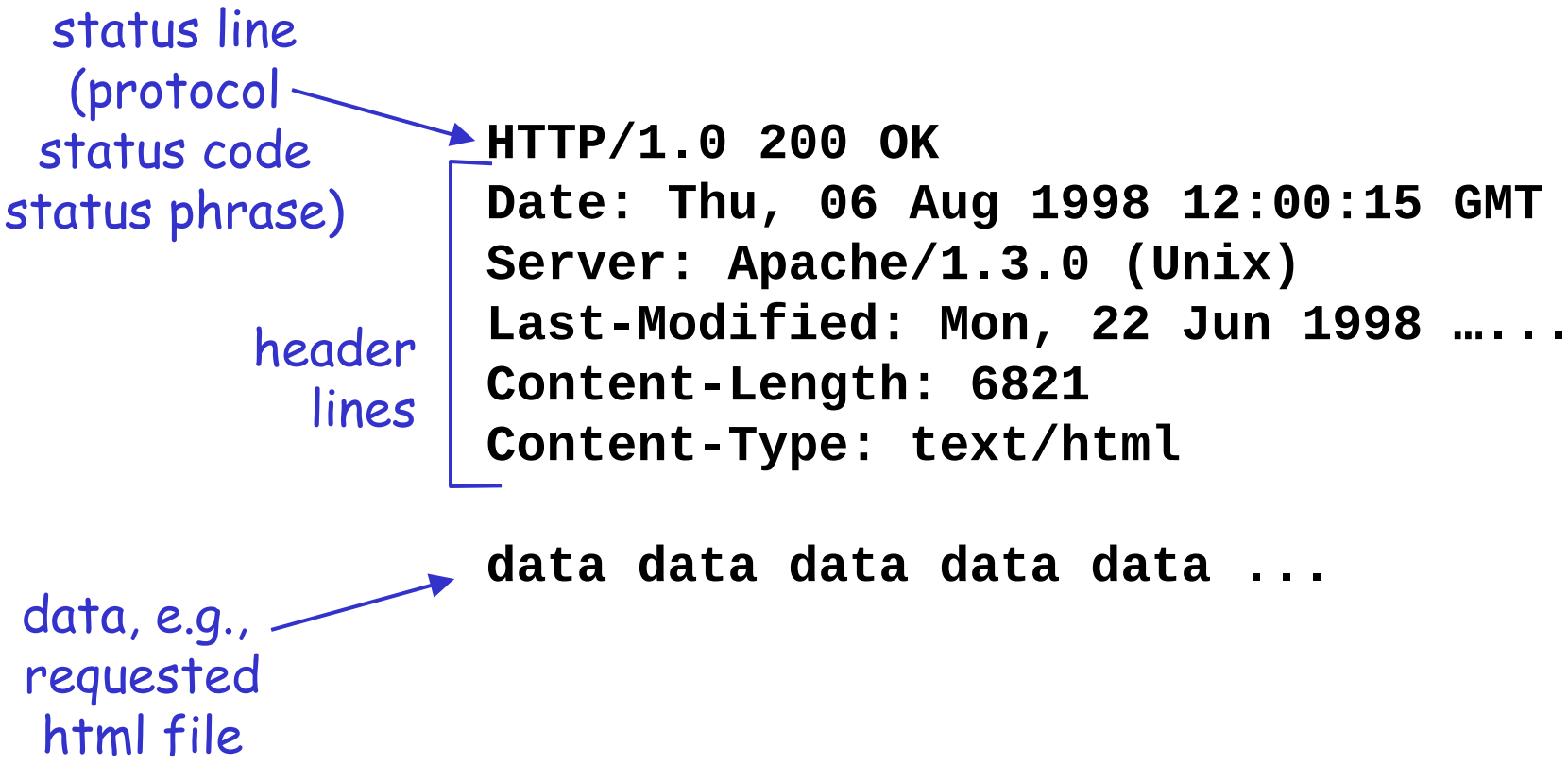
- two types of http messages: *request, response*
- http request message:
  - ASCII (human-readable format)



http request message: general format



http message format: respone



http response status codes

In first line in server->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.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

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!

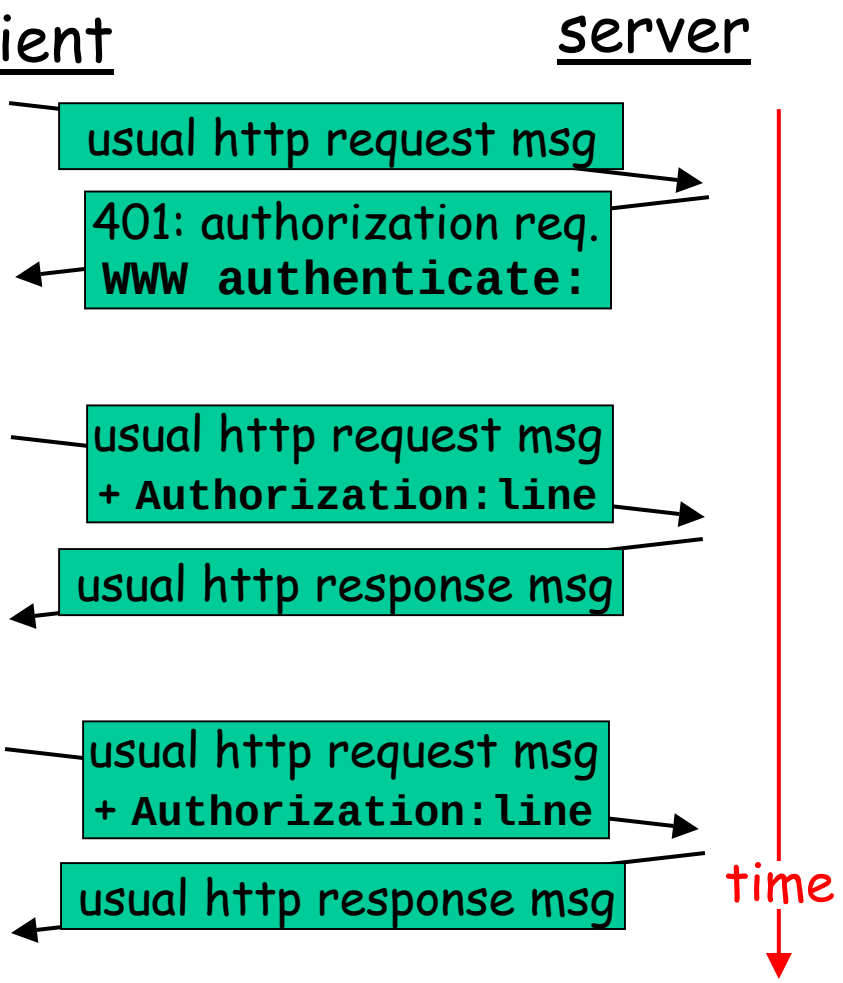
User-server interaction: authentication

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - **`authorization:`** header line in request
  - if no authorization presented, server refuses access, sends

    `WWW authenticate:`

    header line in response



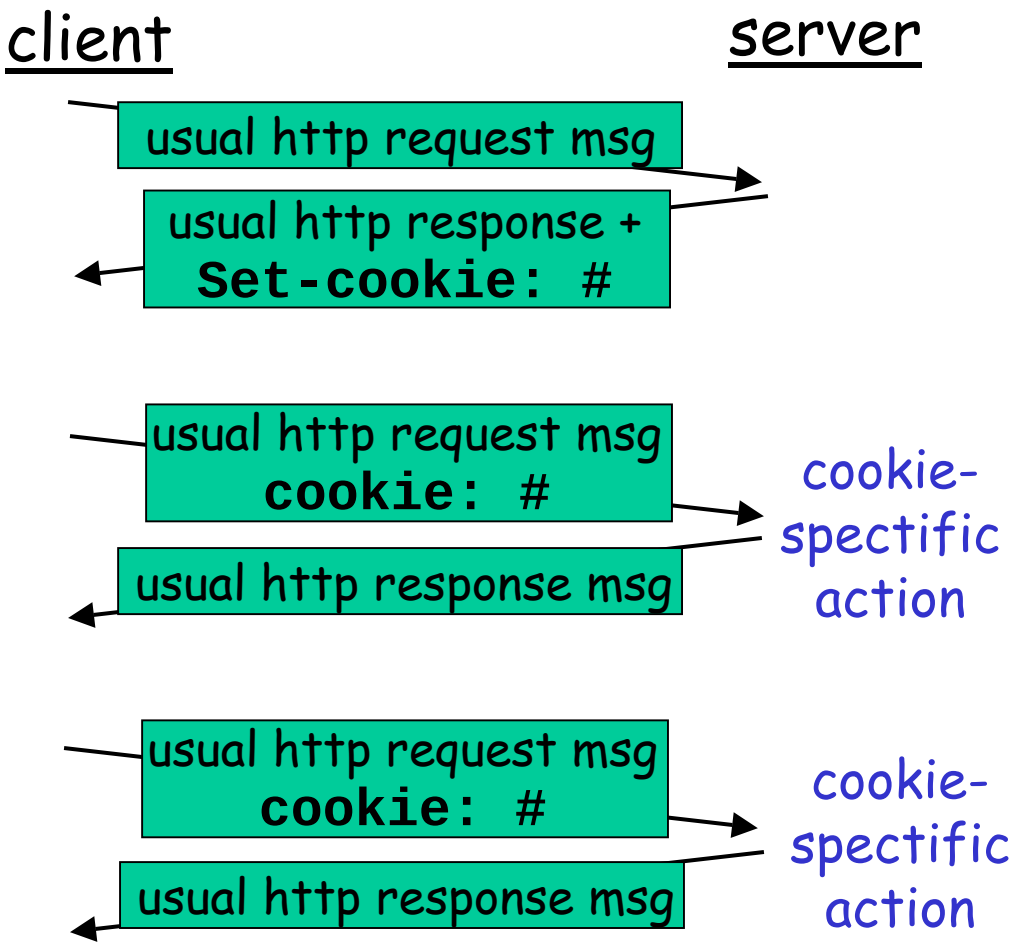
Browser caches name & password so that user does not have to repeatedly enter it.

User-server interaction: cookies

- server sends "cookie" to client in response mst

  `Set-cookie: 1678453`
- client presents cookie in later requests

  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices

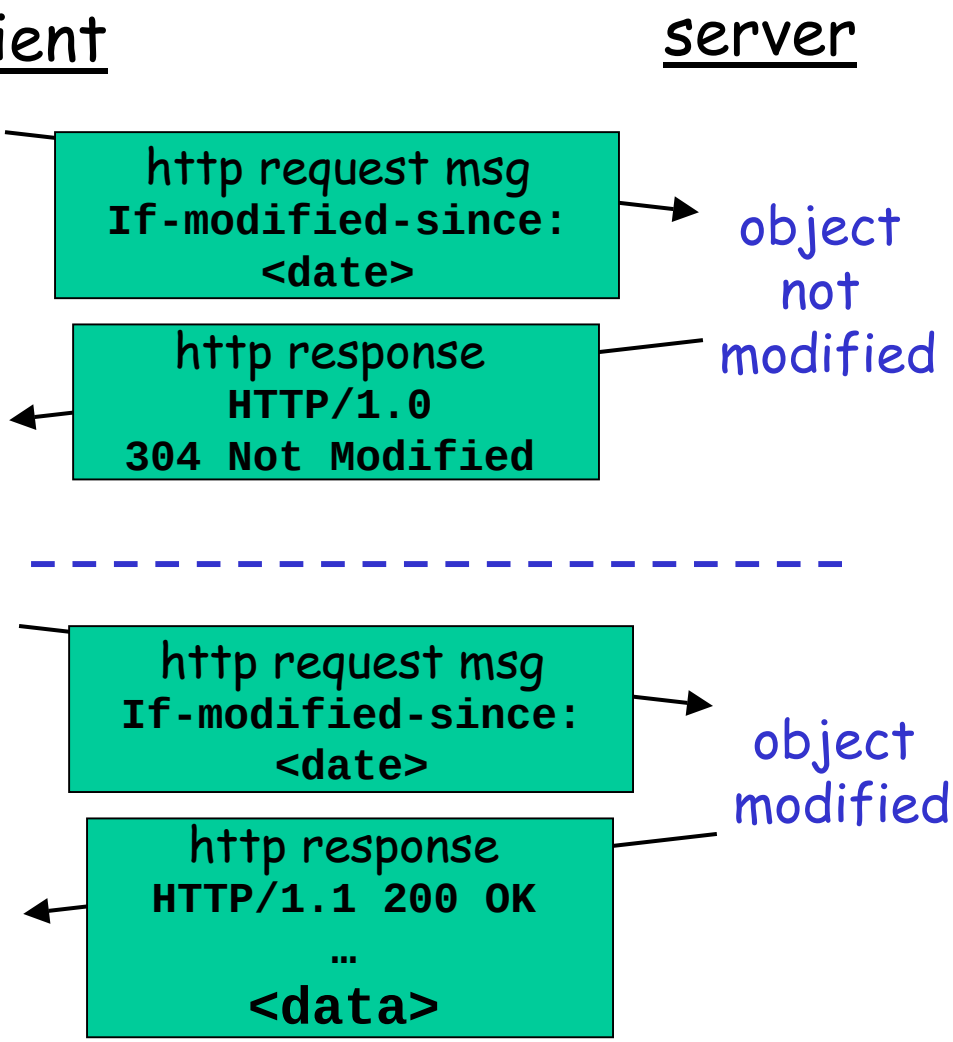


User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

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

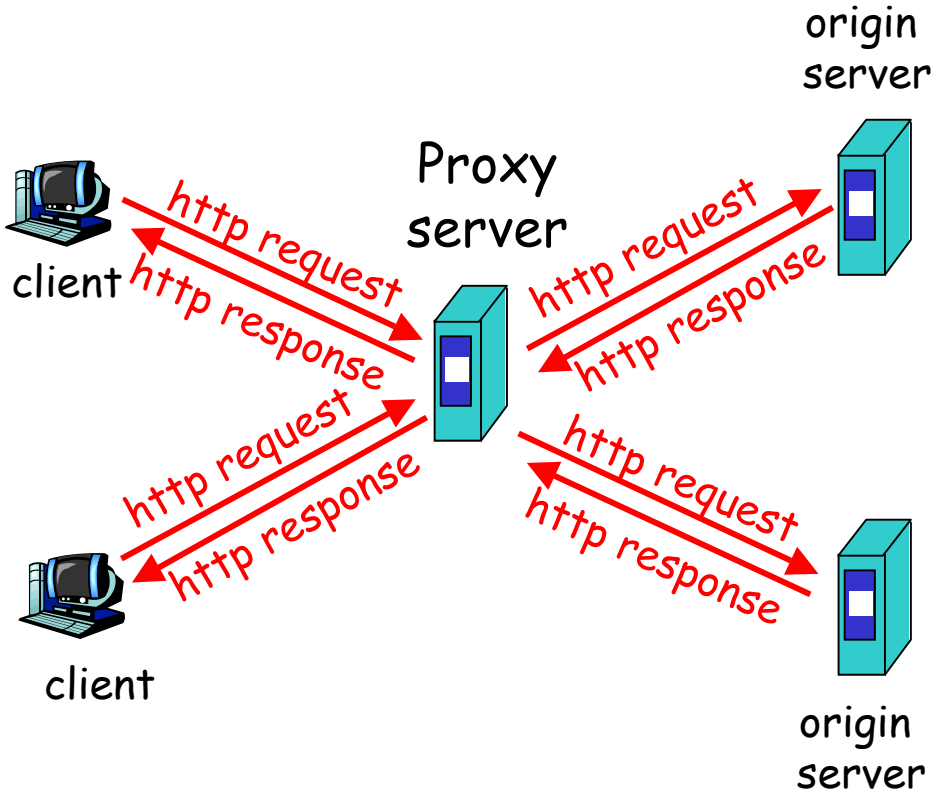
  HTTP/1.0 304 Not Modified



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



Why Web Caching?

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck

