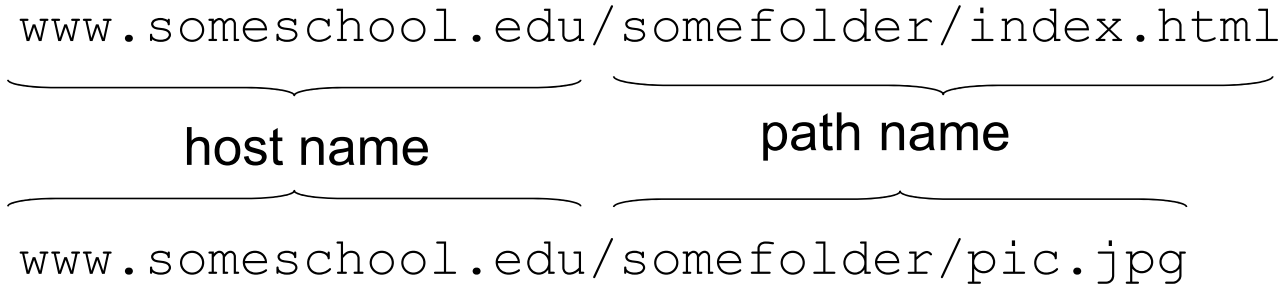
Application layer: overview

- Principles of network applications
- **Web and HTTP**
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Web and HTTP

web page (or webpage)

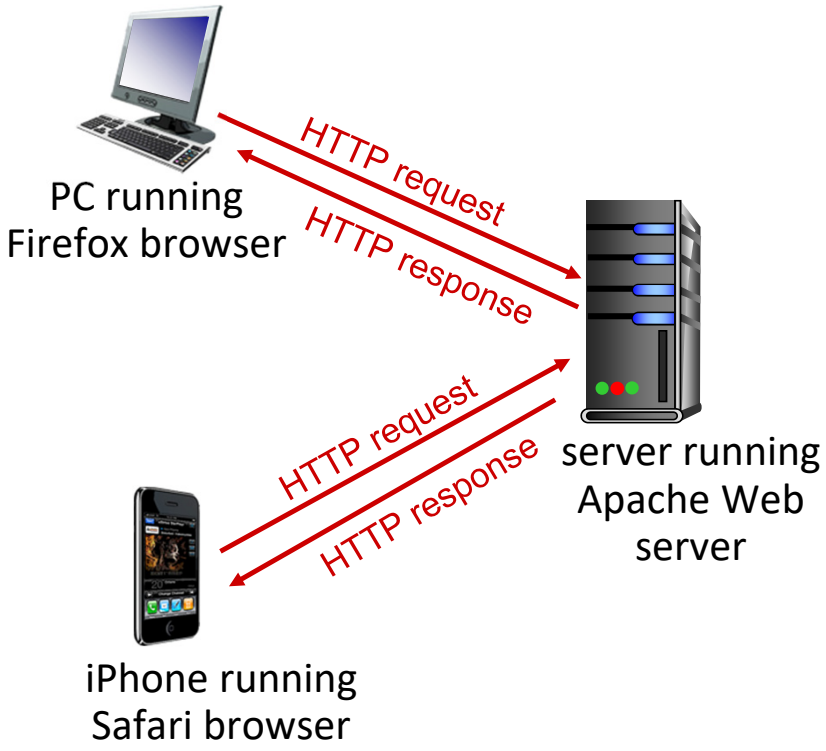
- web page consists of *objects*
  - object is a file such as a HTML file, image, video clip, …
  - objects can be stored on different web servers
- most web pages consist of a *base HTML-file* and several *referenced objects*
  - *each* addressable by a *URL* (Uniform Resource Locator), e.g.,



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)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server
  - server typically runs on port 80
- server accepts the TCP connection
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

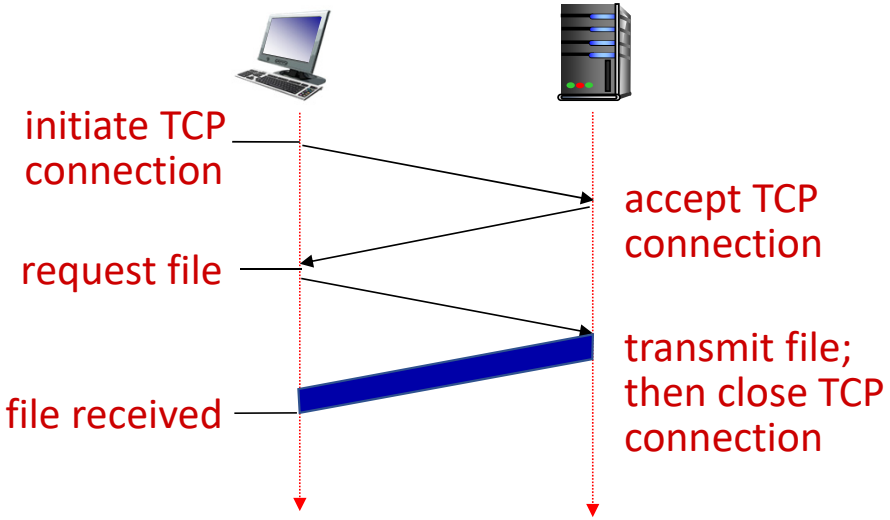
HTTP is "stateless"

- HTTP server maintains *no* information about past client requests
- other mechanisms take care of "state"

HTTP connections: two types

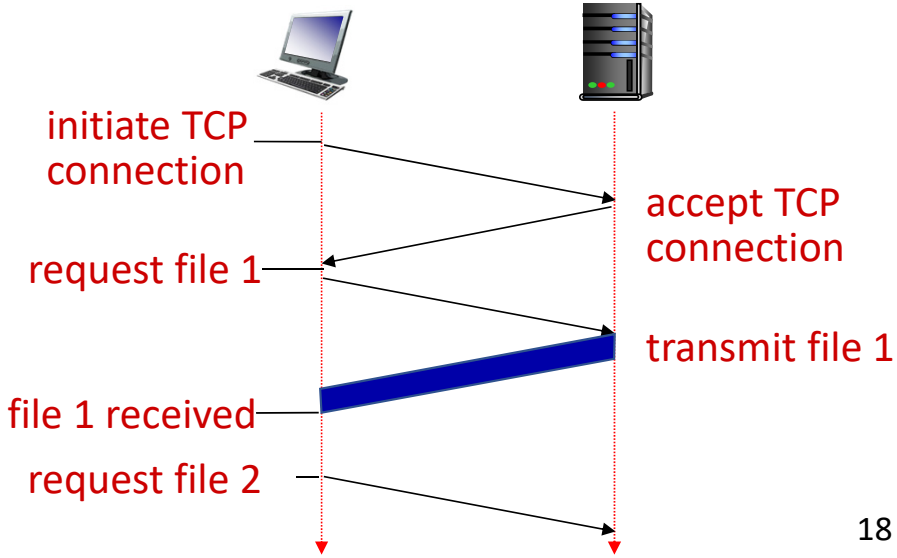
Non-persistent HTTP

- open TCP connection
- send an object over TCP connection
- close TCP connection



Persistent HTTP

- open TCP connection to a server
- multiple objects can be sent over the *single* TCP connection
- close TCP connection



downloading multiple objects requires multiple connections

Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.html`**
(containing 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.html

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/homepage.html`**
(containing text, references to 10 jpeg images)

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

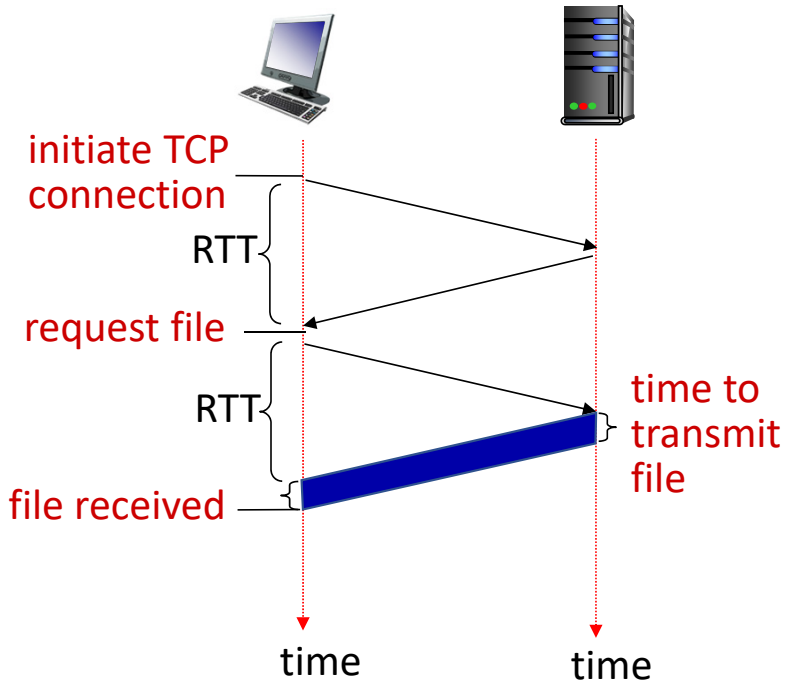
time

Non-persistent HTTP: response time

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

HTTP response time (per object):

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



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

Persistent HTTP (HTTP1.1):

- 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 (cutting response time in half)

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line [HOL] blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

multiple parallel TCP connections: one for a single object

- drawback: high overhead (# of sockets maintained at servers), unfair

HTTP/2

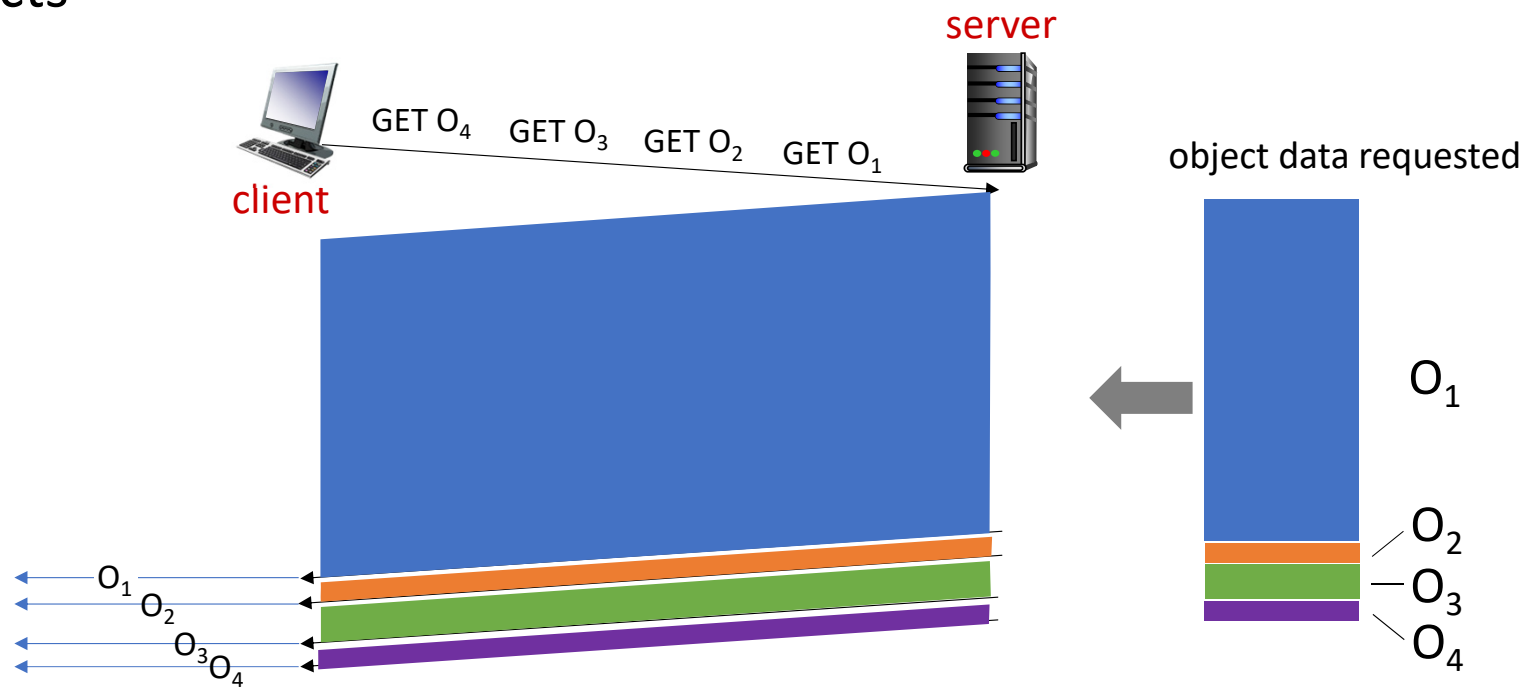
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

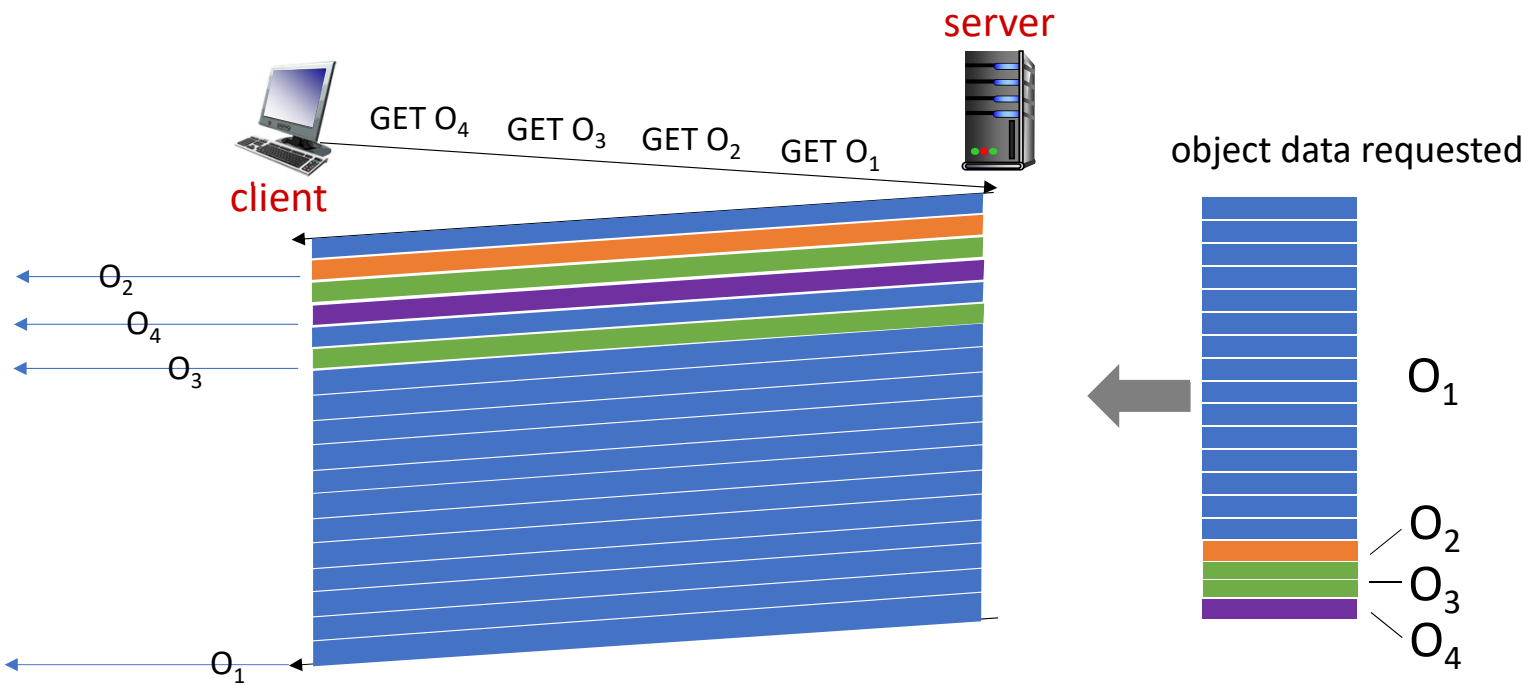
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

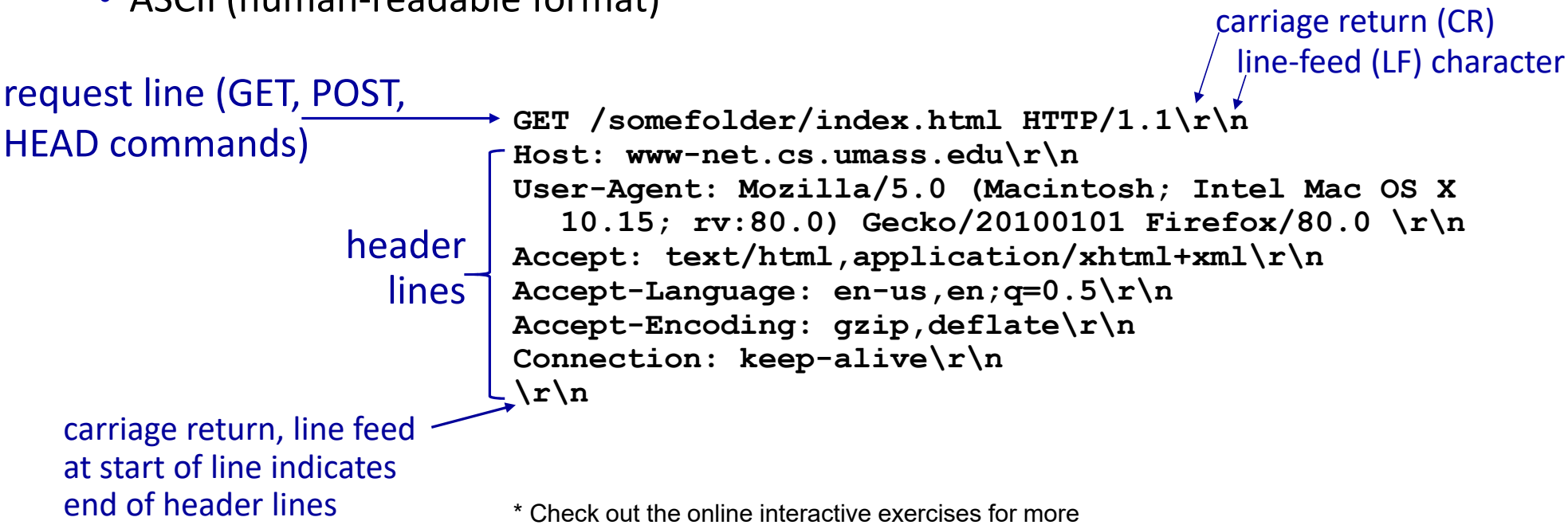
HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

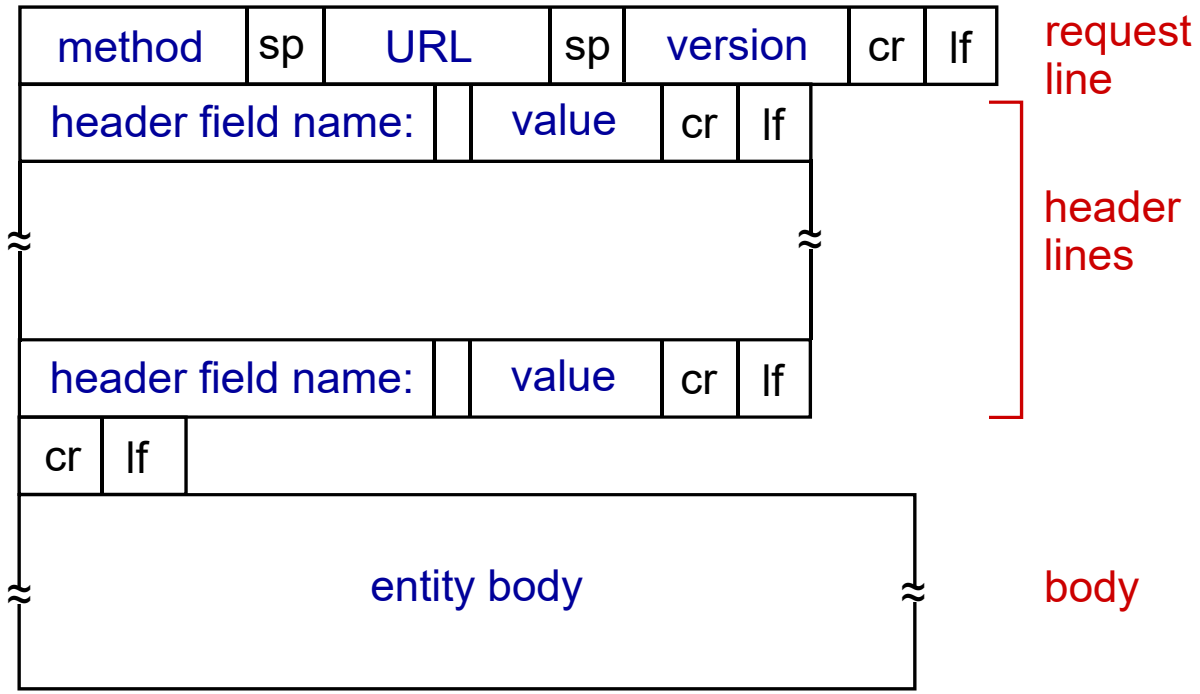
HTTP request message

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



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



Other HTTP request messages

POST method:

- web page might include form input or allow to upload files
- user input (sent from client to server) is put in entity body of HTTP POST request message

GET method (for sending data to server):

- include information in URL field of HTTP GET request message (following a '?'):

```
www.google.com/search?q=http+get&lr=lang_en
```

- entity body ignored or rejected

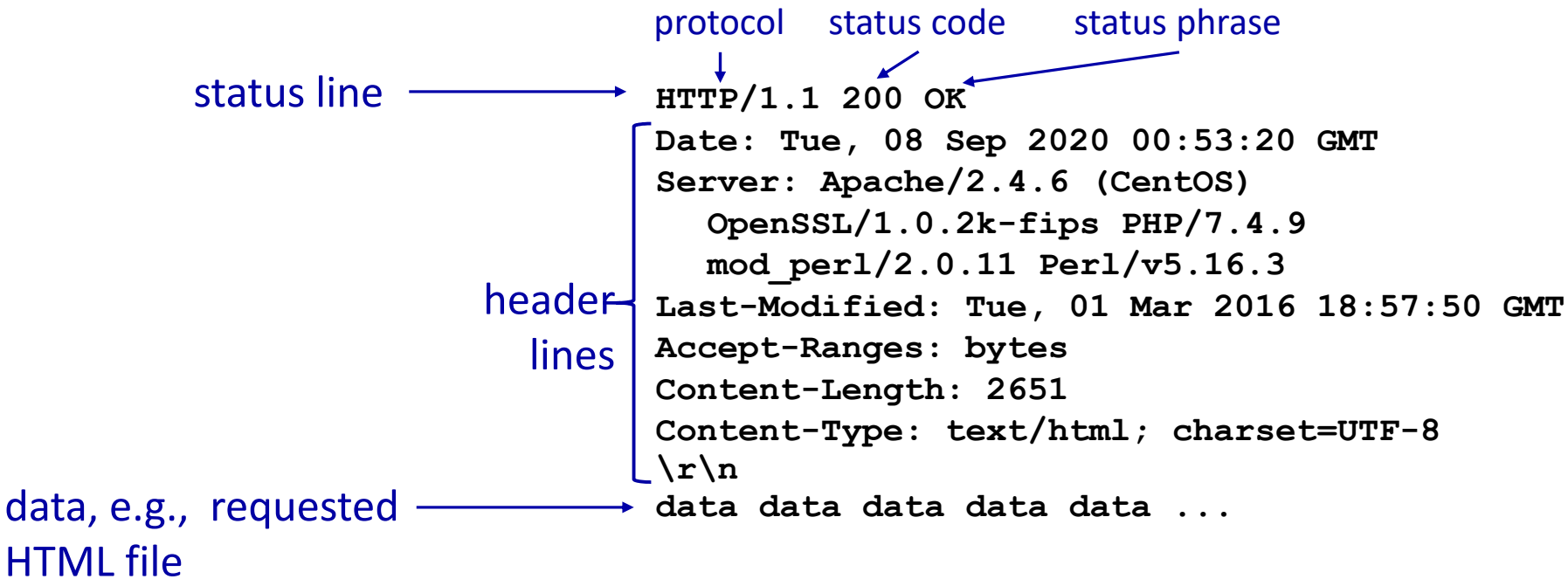
HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.
- often for debugging

PUT method:

- create or update a file to server
  - with content in entity body
  - at specific URL

HTTP response message



* Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/

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 message

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

  400 Bad Request
  - request msg not understood by server

  404 Not Found
  - requested document not found on this server

  505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. telnet to your favorite Web server:

```
% telnet www.cs.nthu.edu.tw 80
```

- opens TCP connection to port 80 (default HTTP server port) at www.cs.nthu.edu.tw.
- anything typed in will be sent to port 80 at www.cs.nthu.edu.tw

2. type in a GET HTTP request:

```
GET /~jungchuk/test.html HTTP/1.1
Host: www.cs.nthu.edu.tw
```

- 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)