

Web and HTTP

First, a review...

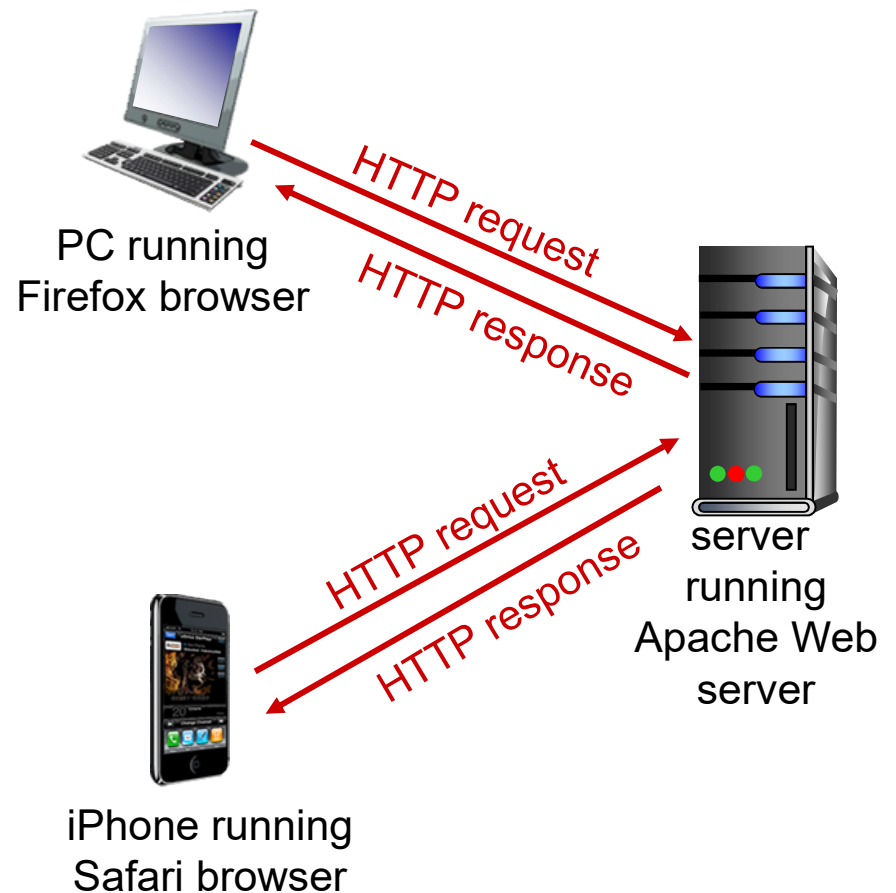
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,...
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL*, e.g.,

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



- surfing
- Search for objects and (fetch /display that object)
- Download some object
- Login – sending some information (userid/password)
- **Fill up forms**
- -----
- URL (address) – GET/download/display
- POST/PUT information –client to server
- ----
- See an image (rendered) jpeg – decoding/display
- Focus on message exchange (communication)
- HTTP - comm

•

HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80 -reserved
- 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
- **MEMORY-LESS**

aside

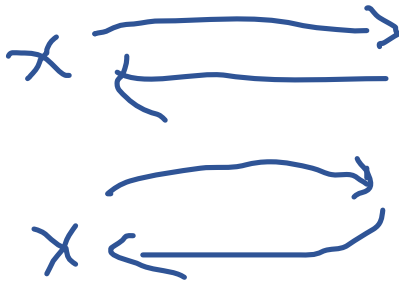
protocols that maintain “state” are complex!

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

~~HTTP connections~~

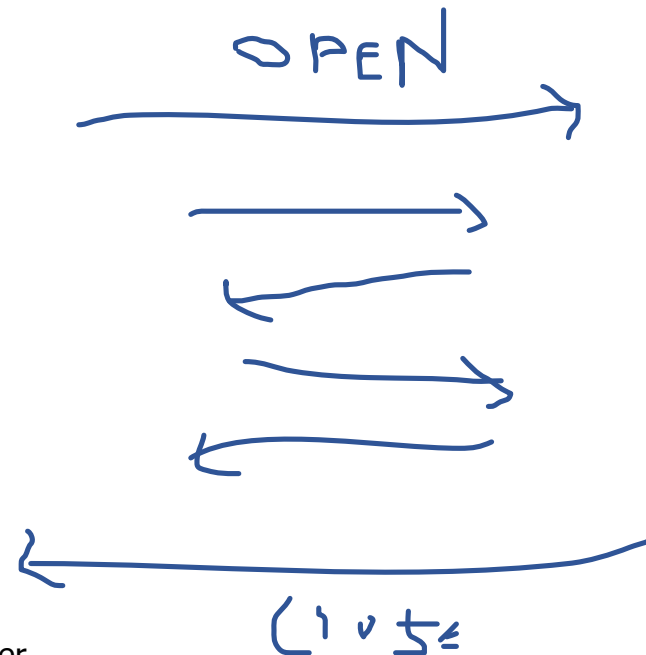
non-persistent HTTP

- 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

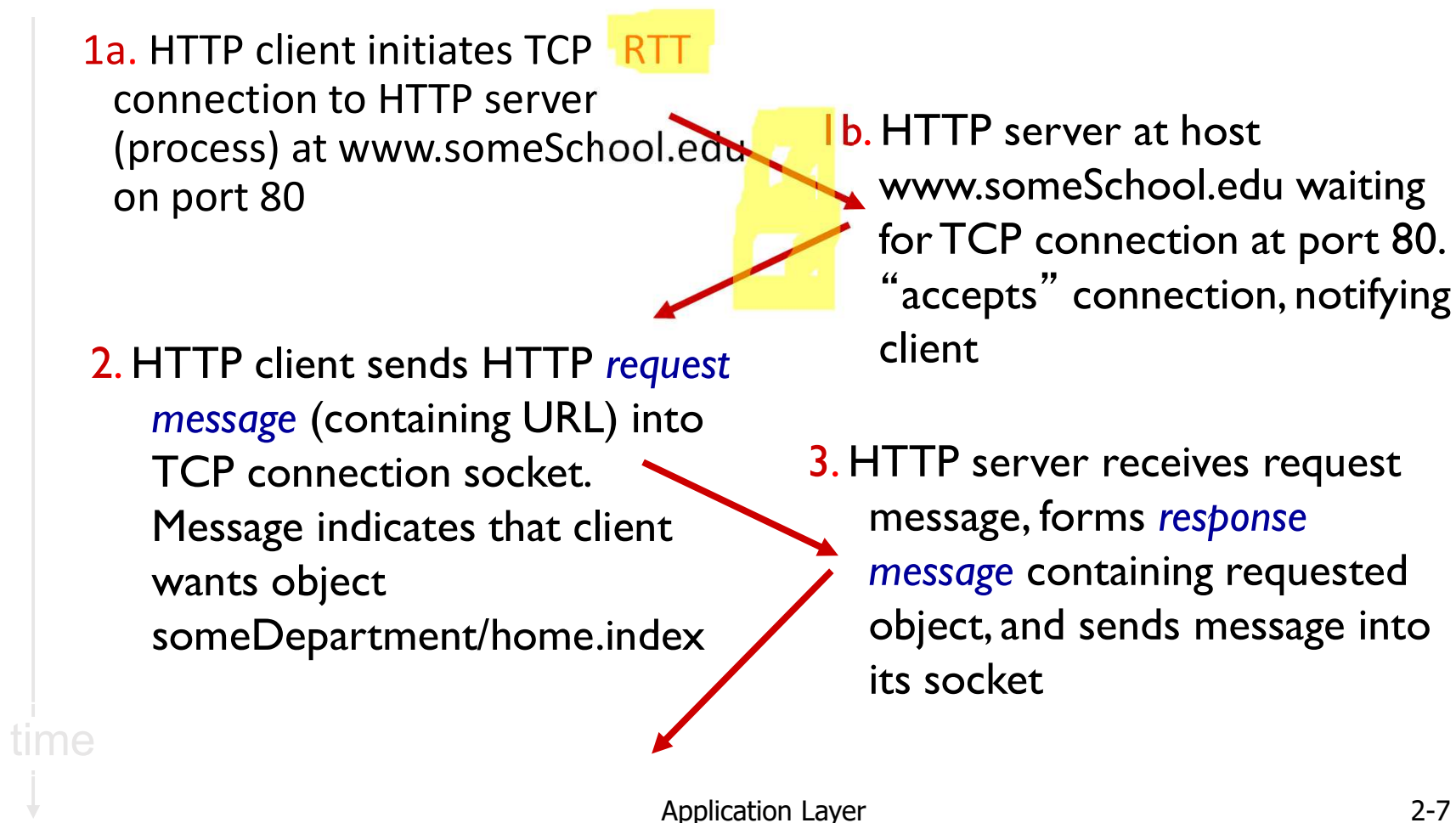


Non-persistent HTTP

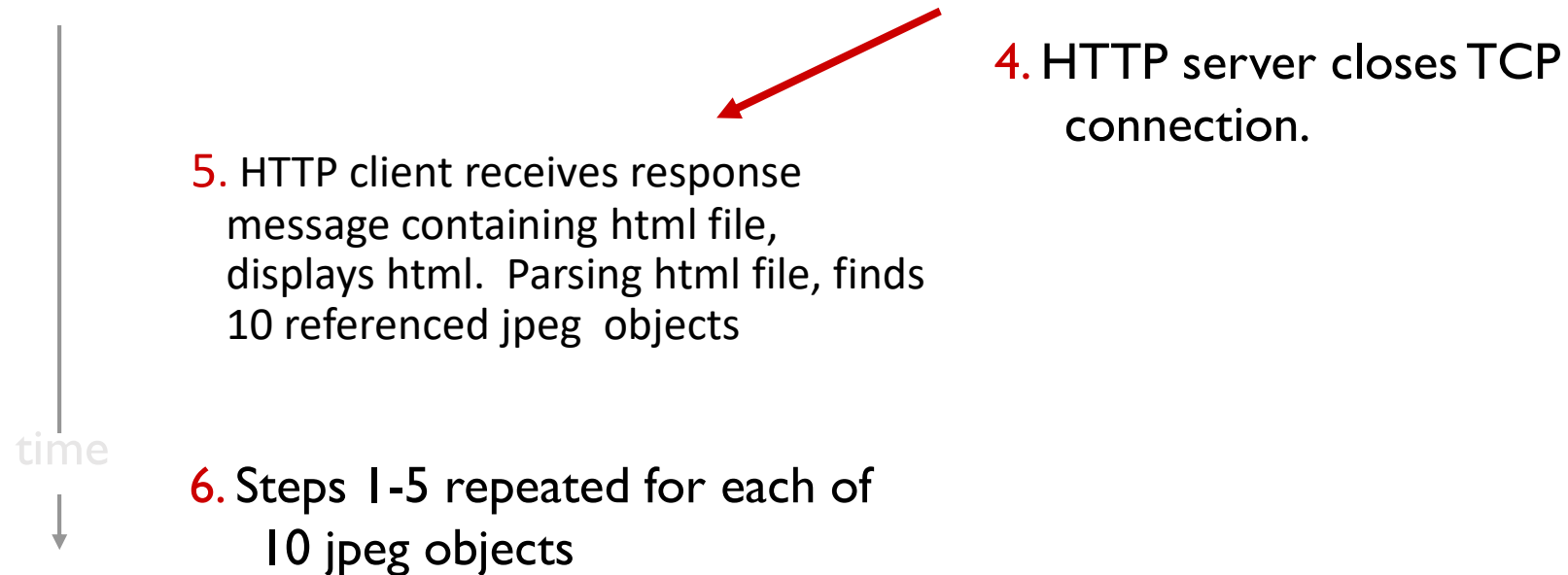
suppose user enters URL:

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

(contains text,
references to 10
jpeg images)



Non-persistent HTTP (cont.)

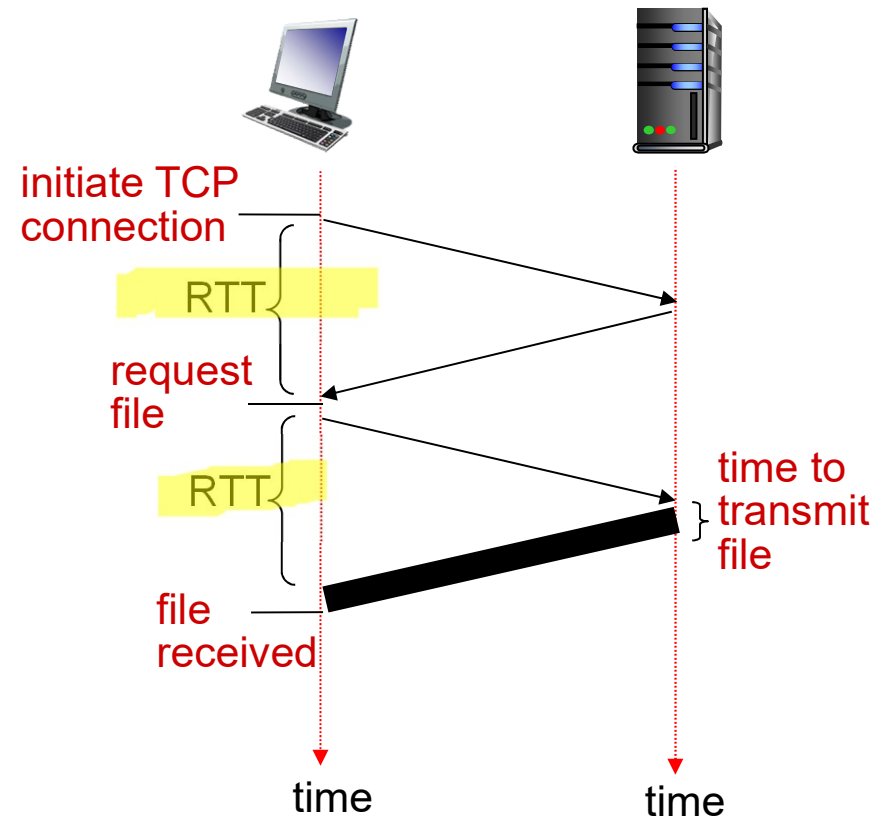


Non-persistent HTTP: response time

RTT (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 =
 $2\text{RTT} + \text{file transmission time}$



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)

request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

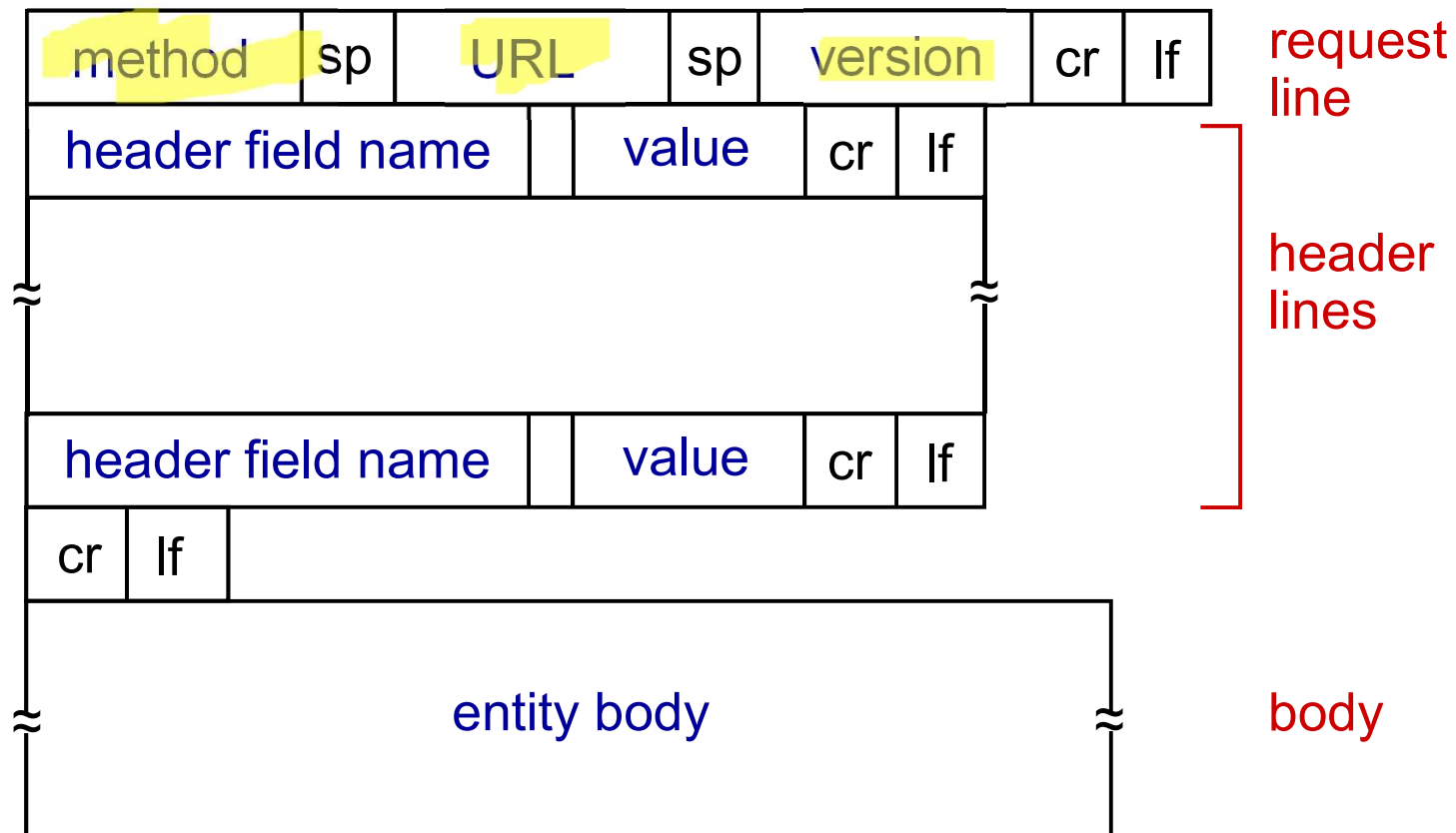
```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character
line-feed character

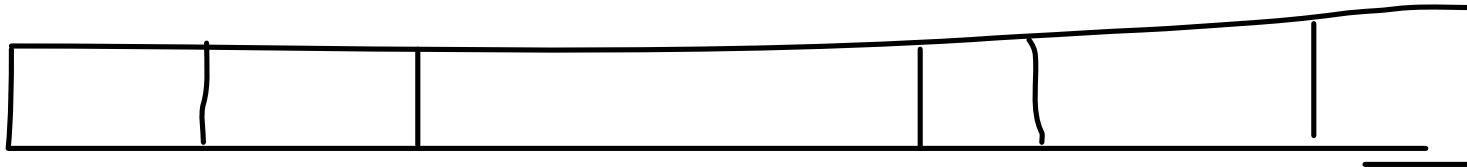
* Check out the online interactive exercises for more
examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Application Layer

~~HTTP request message: general format~~



- MY NAME IS SANJAY. What is your name?
- PARSE (syntax analysis)
- meaning (semantic analysis)



- Define structure of a message
- Field1 (command) – length 2 bytes,
- List of possible commands (**GET**, PUT)
- Next field – parse
- Protocol – structure, parameters, (Requests)
- Responses ()
- Error messages.

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)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

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

Application Layer

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

Trying out HTTP (client side) for yourself

LAB

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)