Network Applications: HyperText Transfer Protocol

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

9/27/2018

Admin

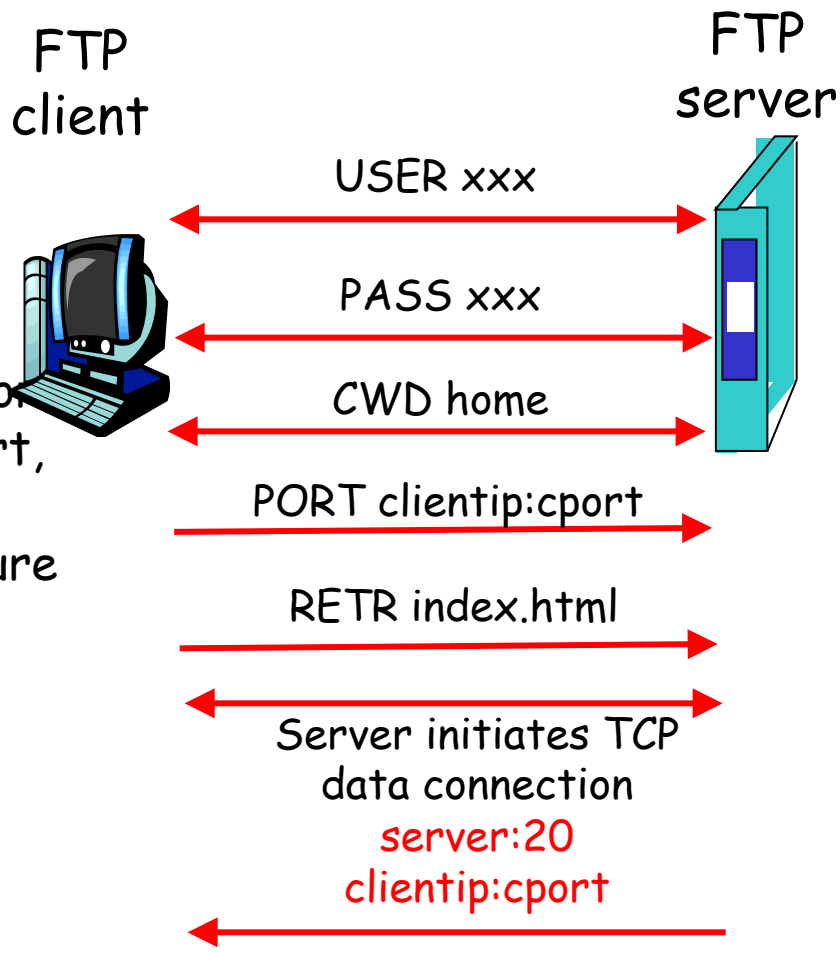
- Assignment 2 status?

Recap: Basic Socket Programming

- UDP: DatagramSocket, MulticastSocket
- TCP: ServerSocket, Socket
- The main function of socket is multiplexing/demultiplexing to application processes
  - UDP uses (dst IP, port) -- roughly
  - TCP uses (src IP, src port, dst IP, dst port)
    - Welcome socket, ServerSocket, Socket
- Always pay attention to encoding/decoding

Recap: FTP Features

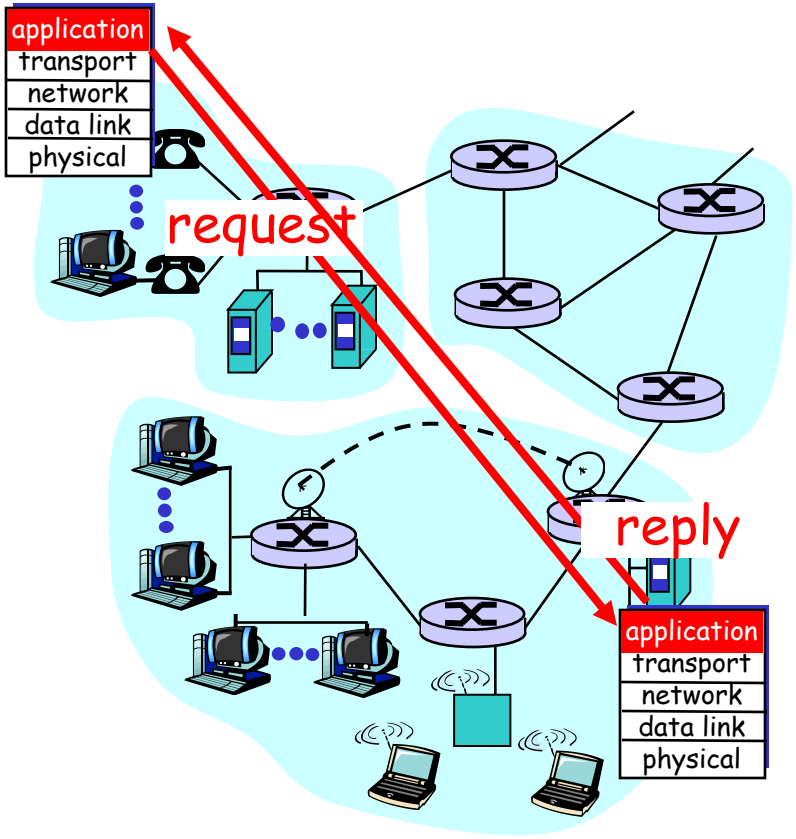
- A stateful protocol
  - state established by commands such as
    - USER/PASS, CWD, TYPE
- Multiple TCP connections
  - A control connection
    - commands specify parameters for the data connection: (1) data port, transfer mode, representation type, and file structure; (2) nature of file system operation e.g., store, retrieve, append, delete, etc.
  - Data connections
    - Two approaches: PORT vs PASV to establish data connections



Recap: FTP Evaluation

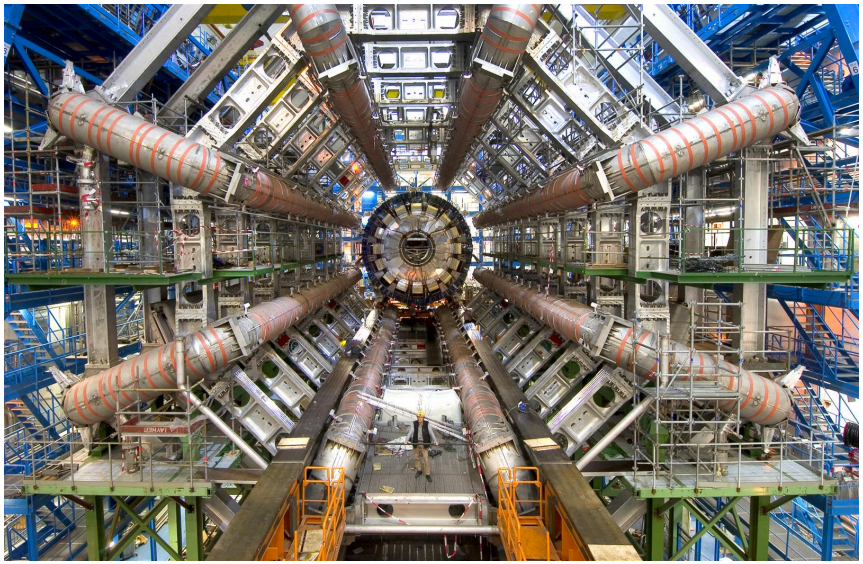
Key questions to ask about a C-S application

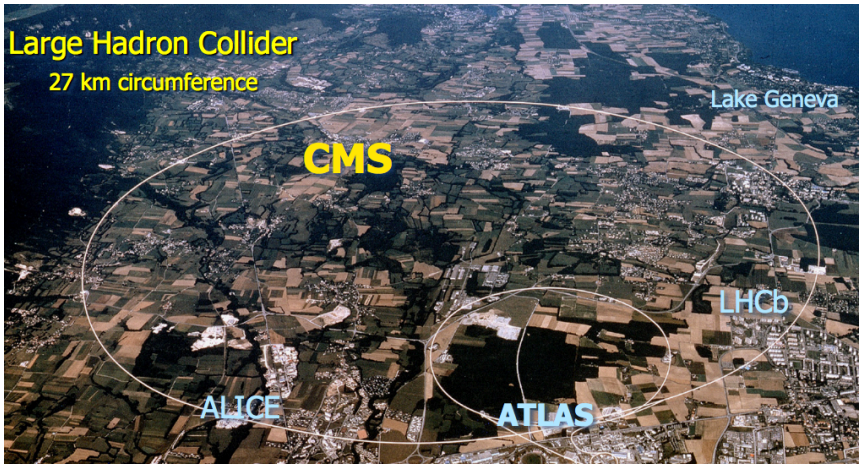
- Is the application **extensible**?
- Is the application **scalable**?
- How does the application handle server failures (being **robust**)?
- How does the application provide **security**?



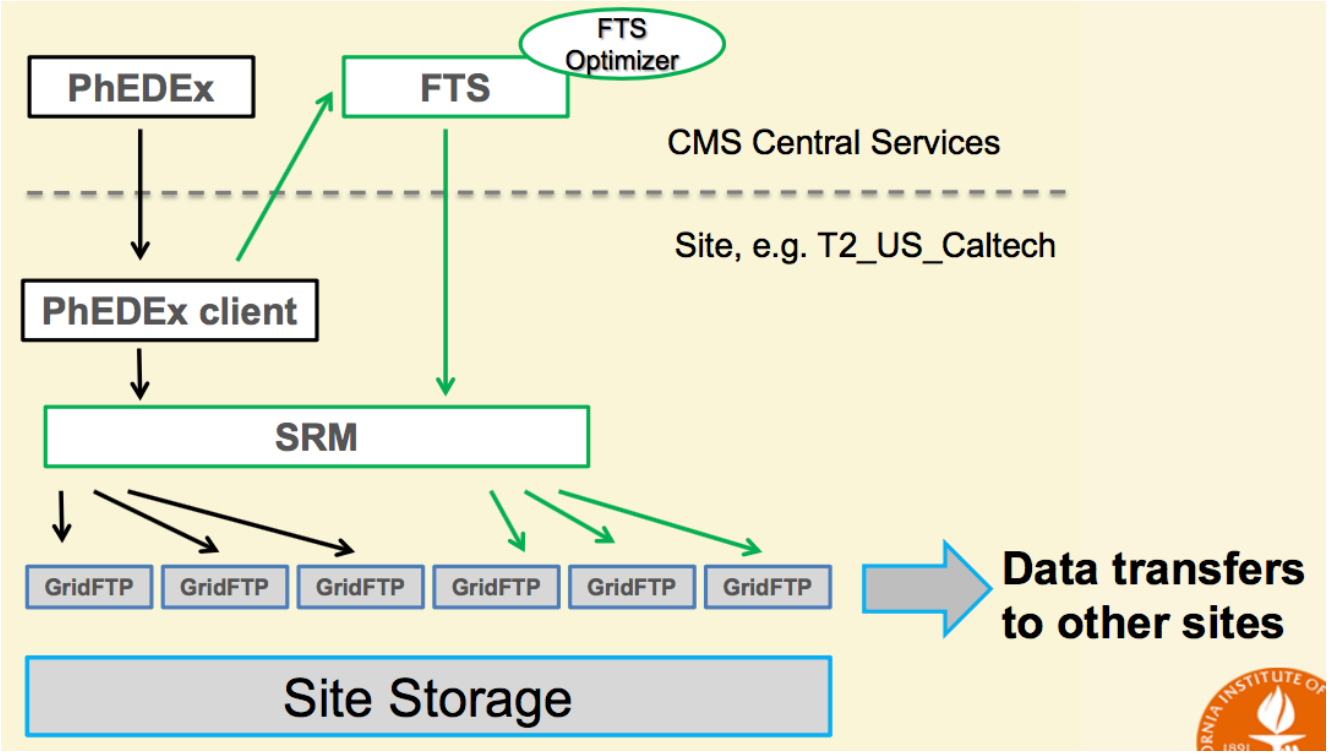
FTP Extensions

- FTP with extensions are being used extensively in large data set transfers (e.g., LHC)





Data Transfer Structure



- See GridFTP to FTP extensions
  - http://www.ogf.org/documents/GFD.20.pdf
  - http://www.ogf.org/documents/GFD.21.pdf
- Goal of GridFTP: allow parallel, high-throughput data transfer
  - Discussion: What features do we need to add to FTP to allow parallel transfers?

Outline

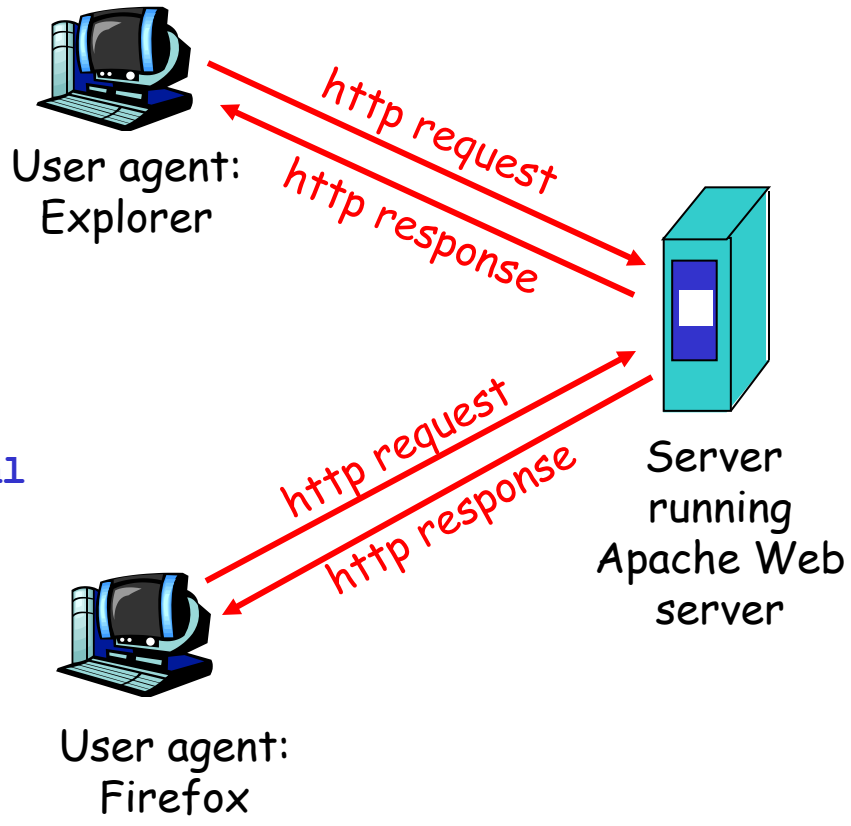
- Admin and recap
- HyperText Transfer Protocol (HTTP)

HTTP Basic Service: From Opaque Files to Web Pages

- Web page:
  - authored in HTML
  - addressed by a URL
    - URL has two components:
      - host name, port number and
      - path name

`http://www.cs.yale.edu:80/index.html`

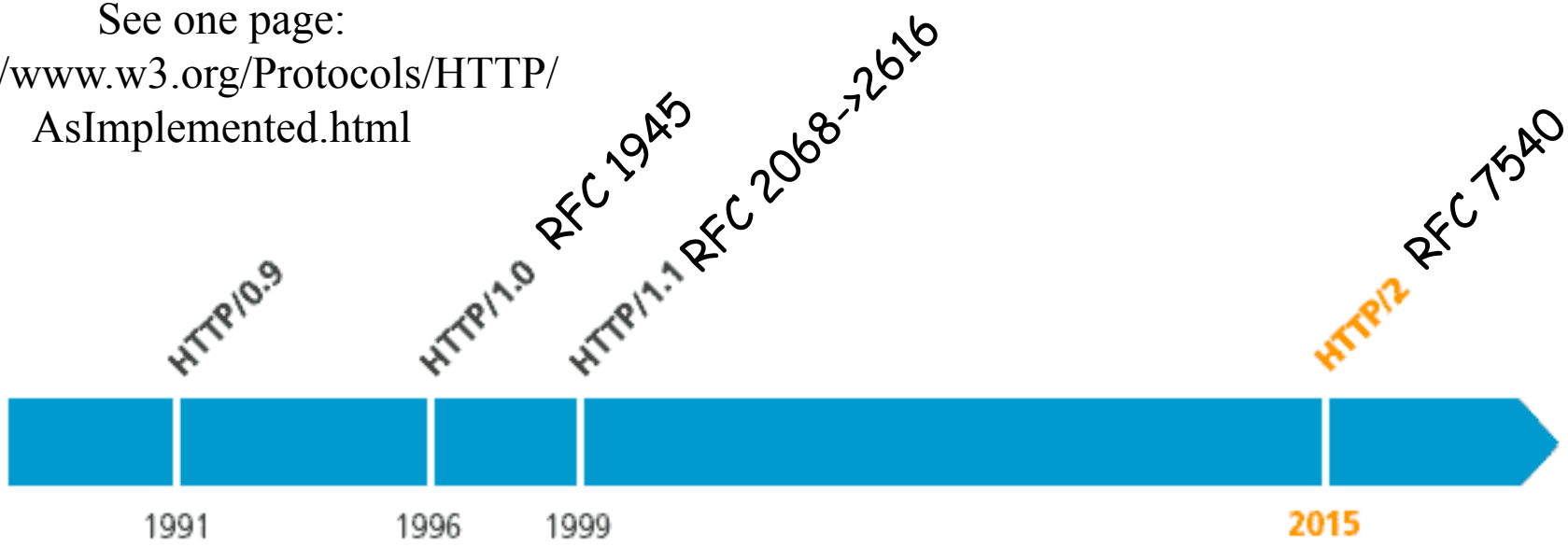
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects (e.g., js, css, images)



The Web pages are requested through HTTP: hypertext transfer protocol

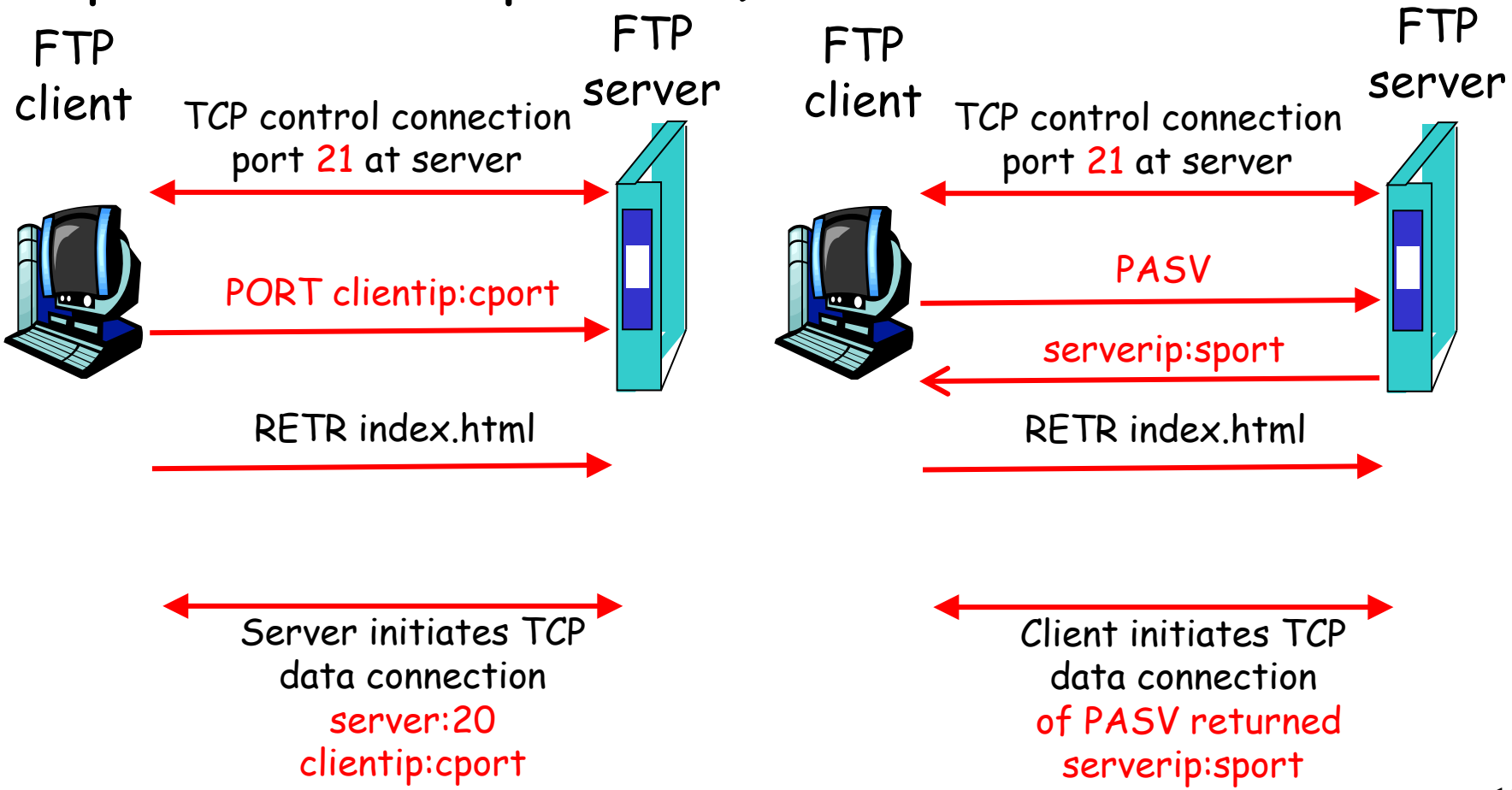
HTTP Design Evolution

See one page:
https://www.w3.org/Protocols/HTTP/AsImplemented.html



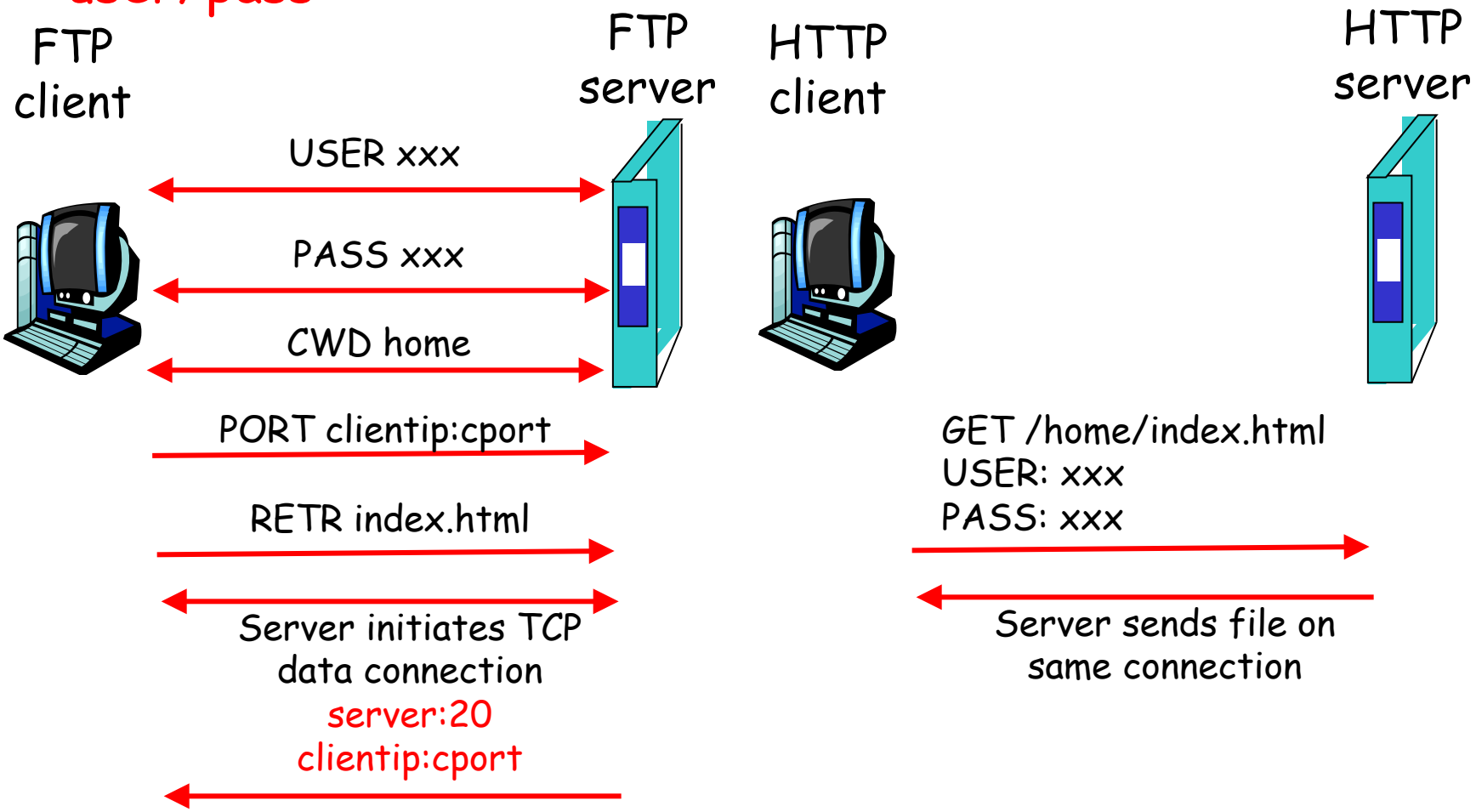
Thought Experiment

- How about we use FTP as (underlying protocol to implement) HTTP?



HTTP Msg Flow vs FTP Msg Flow

- HTTP1.0 servers are stateless servers: each request is self-contained; FTP assumes always starts with user/pass



HTTP Requires Content Management Capabilities (e.g., Content Negotiation)

When request /somedir/page.html

Specify, content negotiation properties such as
- User-agent (Mozilla/4.0)
- Format: text/html, image/gif, image/jpeg,…
- Locality: English, …

In response:

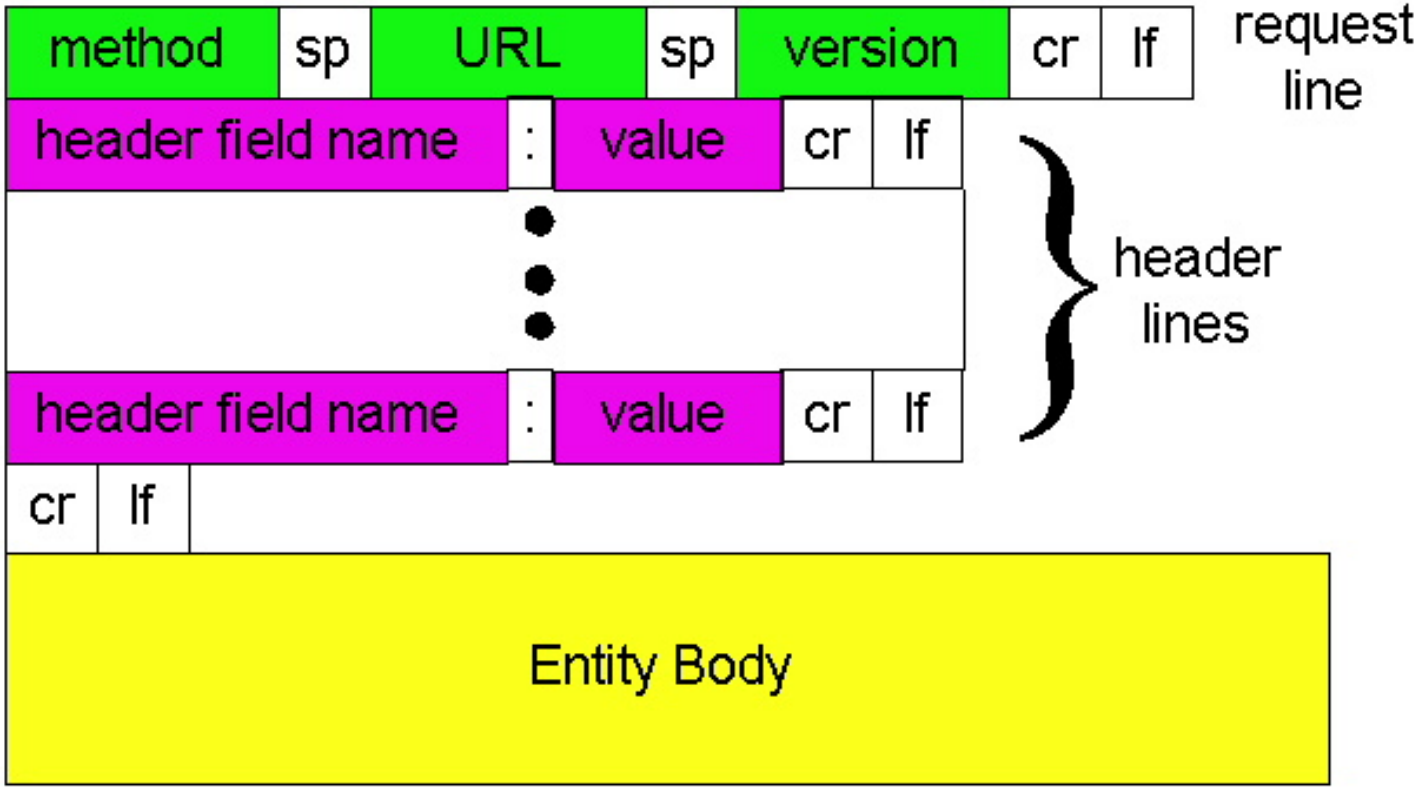
Provide content meta information such as
- Format: text/html, image/gif, image/jpeg,…
- Last modification time,

Outline

- Admin and recap
- HTTP
  - Basic service
  - Basic protocol

HTTP Request Message: General Format

- Simple methods
- Rich headers to support functions such as content negotiation



Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.cs.yale.edu 80
```

Opens TCP connection to port 80 (default http server port) at www.cs.yale.edu. Anything typed in sent to port 80 at www.cs.yale.edu

2. Type in a GET http request:

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

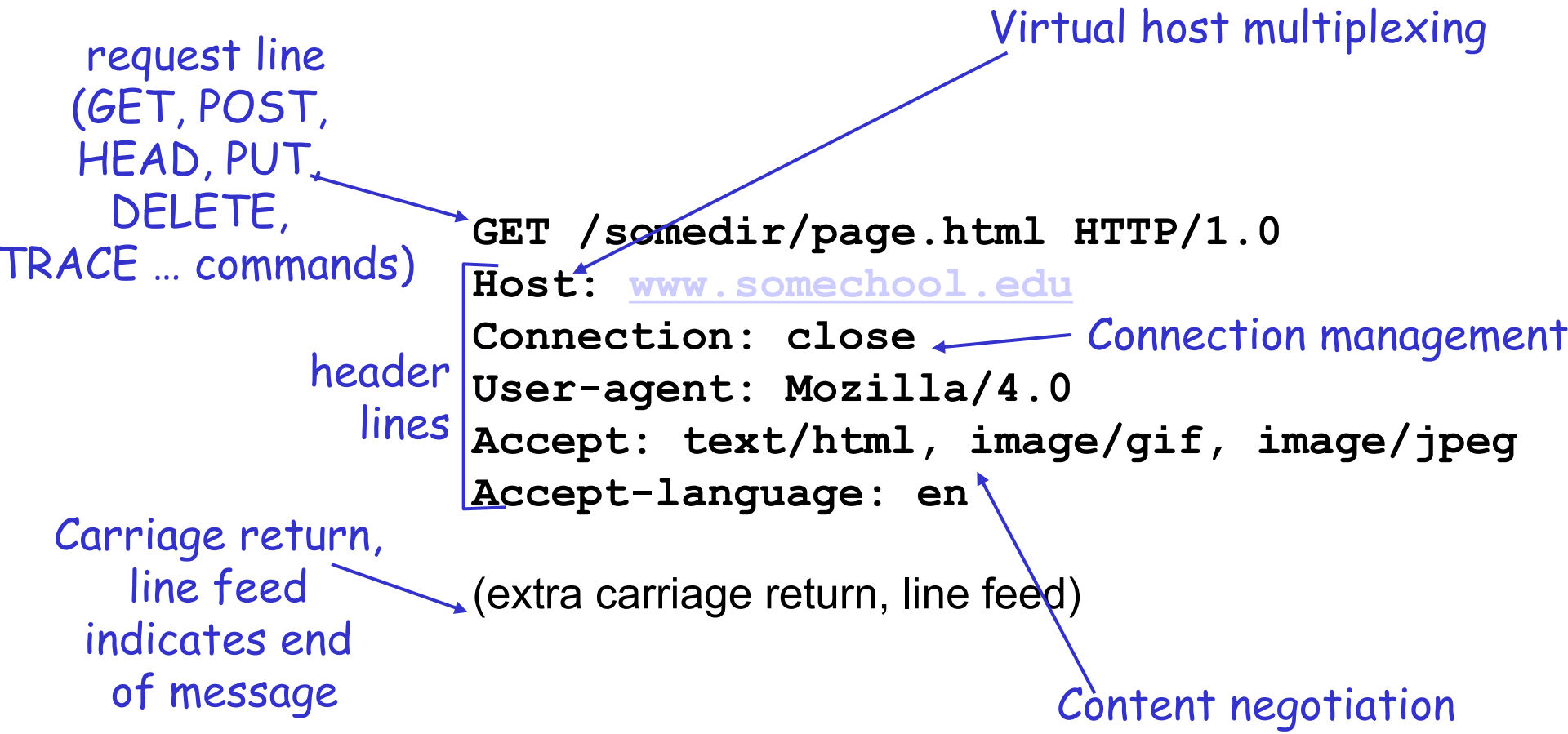
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by the http server.

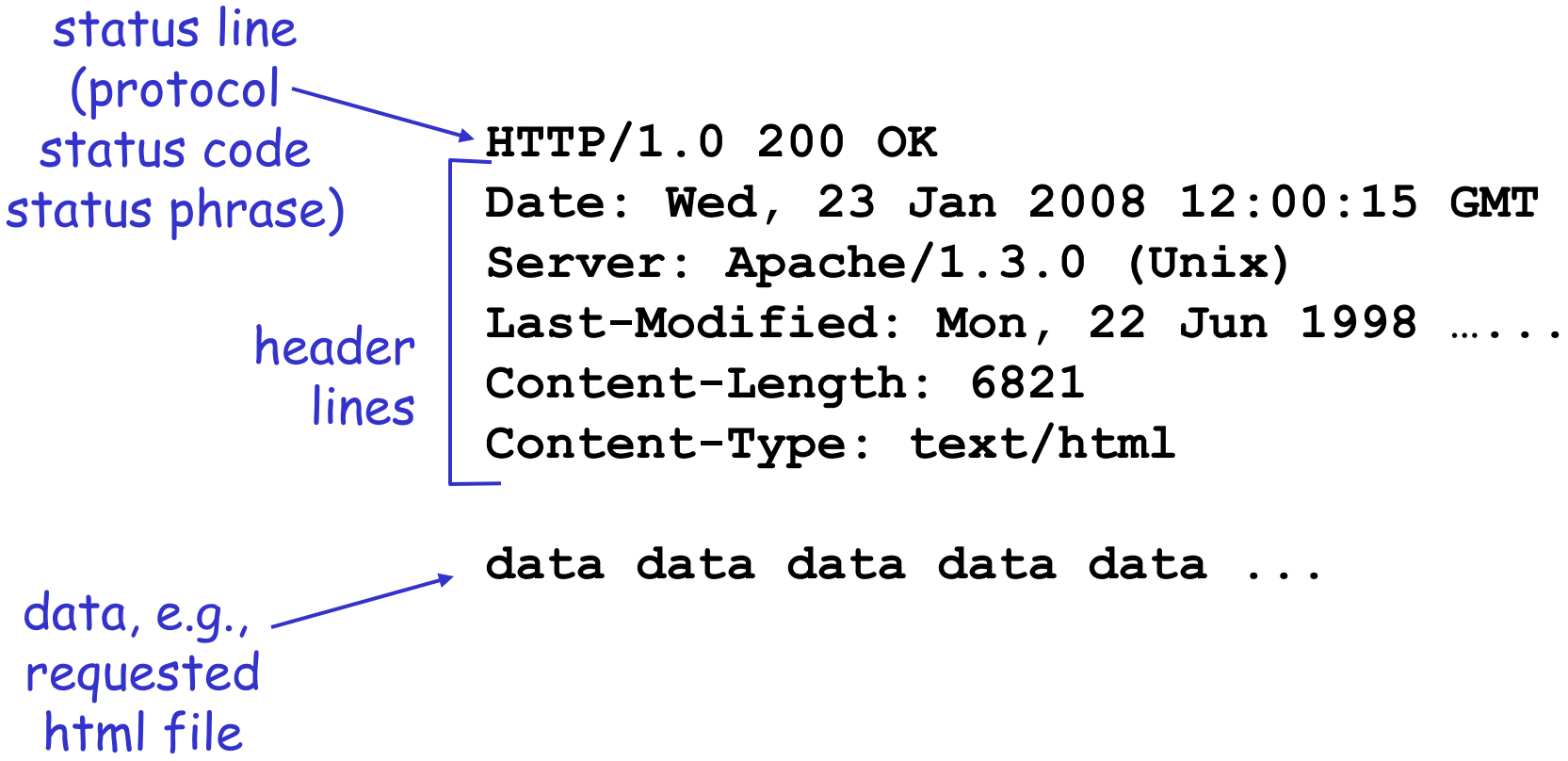
Trying out HTTP (client side) for yourself

- Try telnet GET on
  - zoo.cs.yale.edu to fetch class home page
  - www.cs.yale.edu
  - www.yale.edu

HTTP Request Message Example: GET



HTTP Response Message



HTTP Response Status Codes

In the first line of the 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

Outline

- Admin and recap
- HTTP
  - Basic service
  - Basic protocol
  - Basic HTTP client/server workflow

Simple Design Exercise

- Workflow of an HTTP client/server processing

HTTP Basic Message Flow

Suppose user enters URL
www.cs.yale.edu/index.html

0. http server at host www.cs.yale.edu waiting for TCP connection at port 80.

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

1b. server "accepts" connection, ack. 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 (index.html), sends message into socket

time

HTTP Basic Message Flow (cont.)

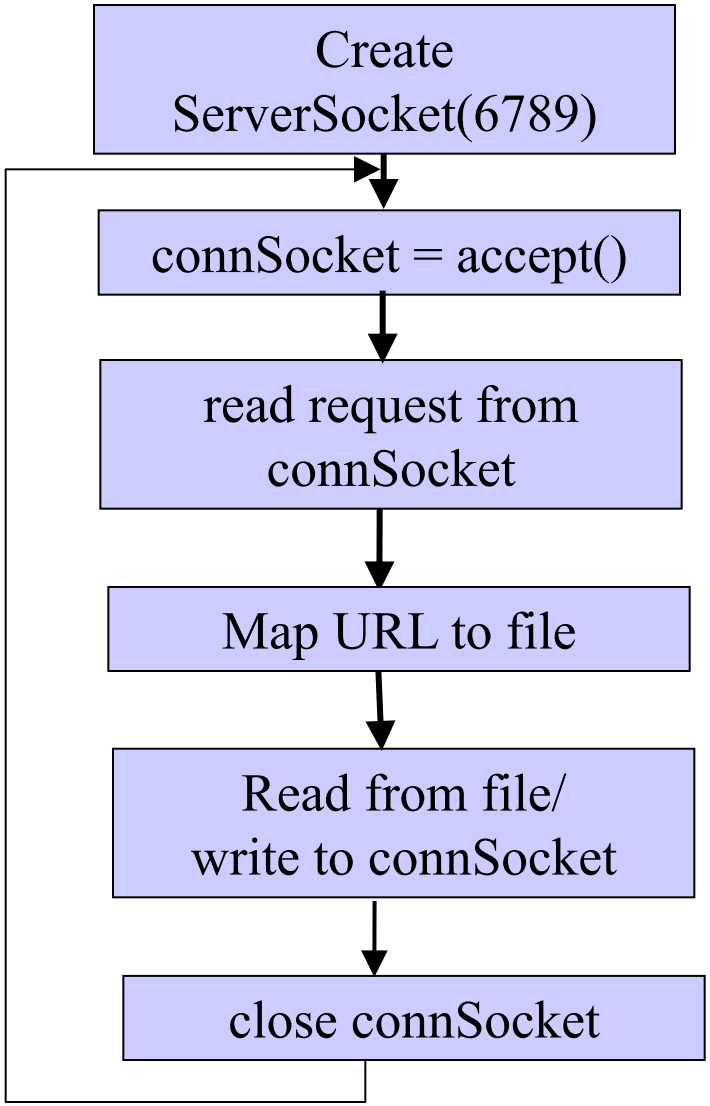
4. http server closes TCP connection.

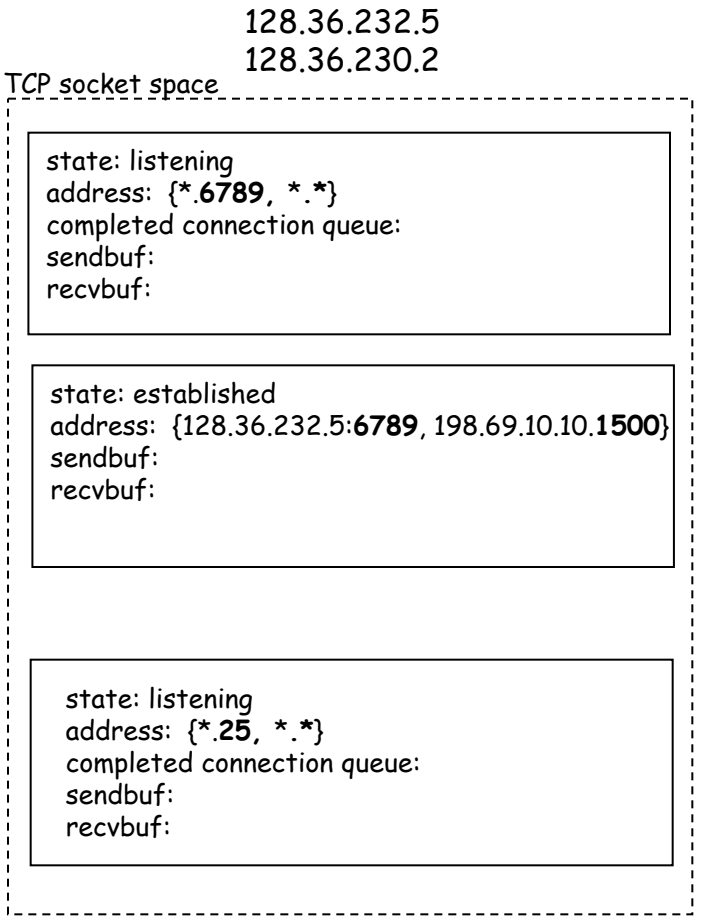
5. http client receives response message containing html file, parses html file, finds embedded image

time

6. Steps 1-5 repeated for each of the embedded images

Basic HTTP Server Workflow





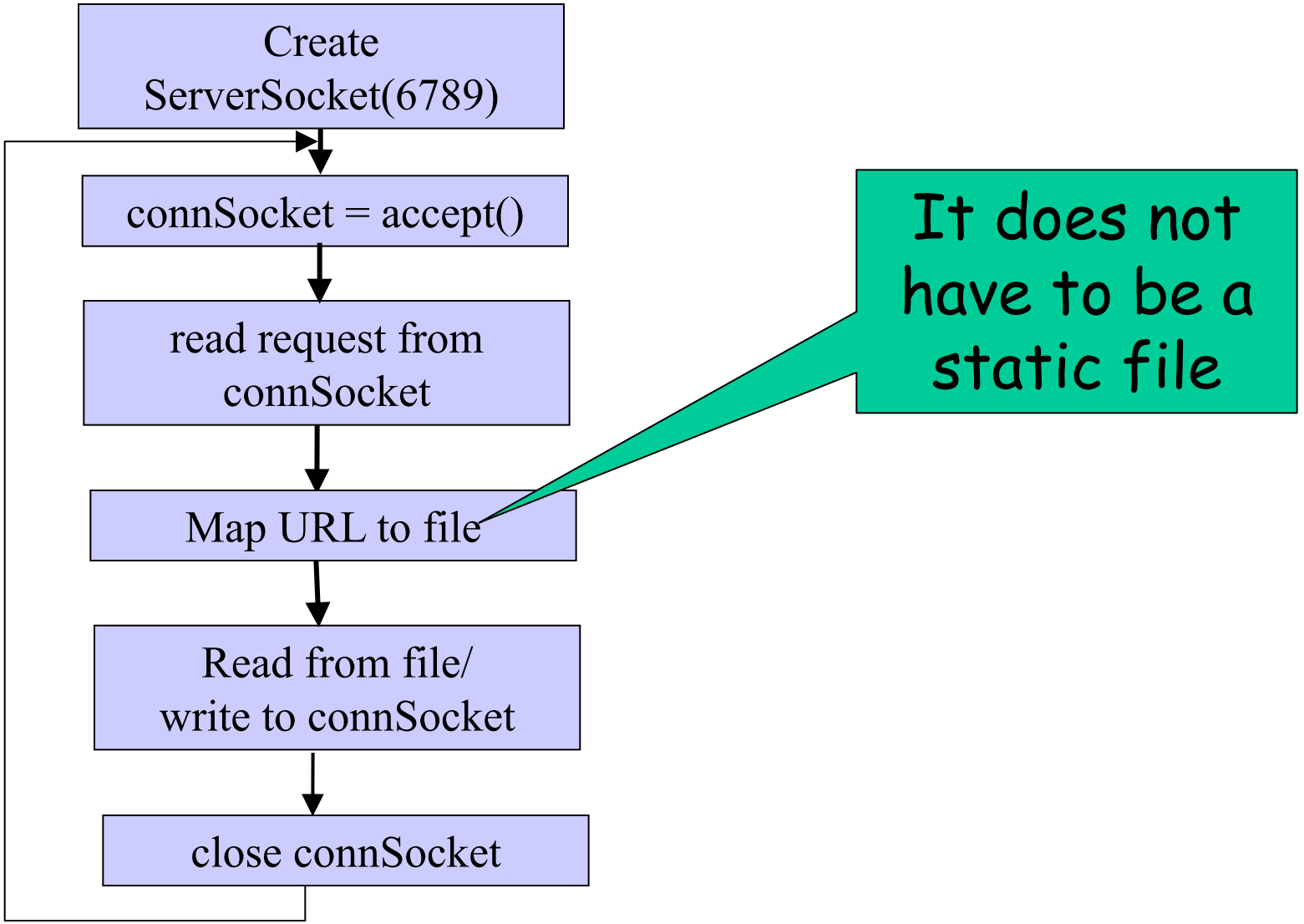
Example Code

- See BasicWebServer.java

- Try using telnet and real browser, and fetch
  - file1.html
  - index.html

  what difference in behavior?

URL as Content Abstraction



Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
  - Embed code into pages (server side include)
    - http server includes an interpreter for the type of pages
  - Invoke external programs (http server is agnostic to the external program execution)

```
http://www.cs.yale.edu/index.shtml
http://www.cs.yale.edu/cgi-bin/ureserve.pl
http://www.google.com/search?q=Yale&sourceid=chrome
```

Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, …

Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, …

- To enable ssi, need configuration to tell the web server (see conf/apache-htaccess)
  - https://httpd.apache.org/docs/2.2/howto/htaccess.html (Server Side Includes example)

CGI: Invoking External Programs

- Two issues
  - Input: Pass HTTP request parameters from server to the external program
  - Output: Get external program output to server (to client)

Example: Typical CGI Implementation

- Starts the executable as a child process
  - Passes HTTP request as environment variables
    - `http://httpd.apache.org/docs/2.2/env.html`
    - CGI standard: http://www.ietf.org/rfc/rfc3875
    - Example
      - `GET /search?q=Yale&sourceid=chrome HTTP/1.0`
      - setup environment variables, in particular $QUERY_STRING=`q=Yale&sourceid=chrome`
  - Redirects input/output of the child process to the socket
    - Read this before Assignment Three

https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

https://httpd.apache.org/docs/2.2/howto/htaccess.html
(CGI Example)

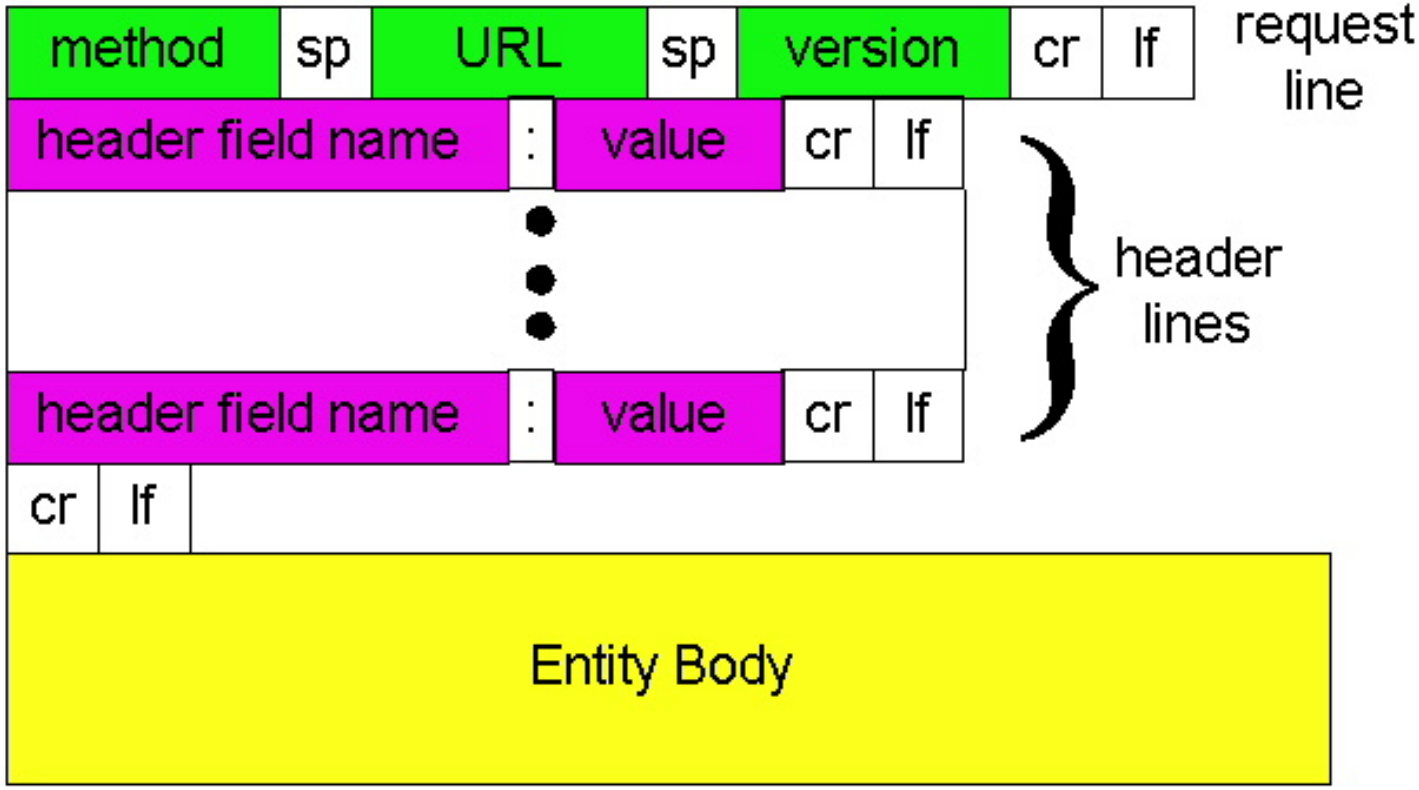
Example

- http://172.28.229.215/BasicWebServer/cgi/price.cgi?appl

```
#!/usr/bin/perl -w

$company = $ENV{'QUERY_STRING'};
print "Content-Type: text/html\r\n";
print "\r\n";

print "<html>";
print "<h1>Hello! The price is ";

if ($company =~ /appl/) {
  my $var_rand = rand();
  print 450 + 10 * $var_rand;
} else {
  print "150";
}

print "</h1>";
print "</html>";
```

HTTP: POST

- If an HTML page contains forms or parameter too large, they are sent using POST and encoded in message body



HTTP: POST Example

```
POST /path/script.cgi HTTP/1.0
User-Agent: MyAgent
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

item1=A&item2=B
```

Example using nc:
programming/examples-java-socket/BasicWebServer/nc/

Outline

- Admin and recap
- HTTP
  - Basic service
  - Basic protocol
  - Basic HTTP client/server workflow
  - Stateful interaction using stateless HTTP

Stateful User-server Interaction: Cookies

Goal: no explicit application level session

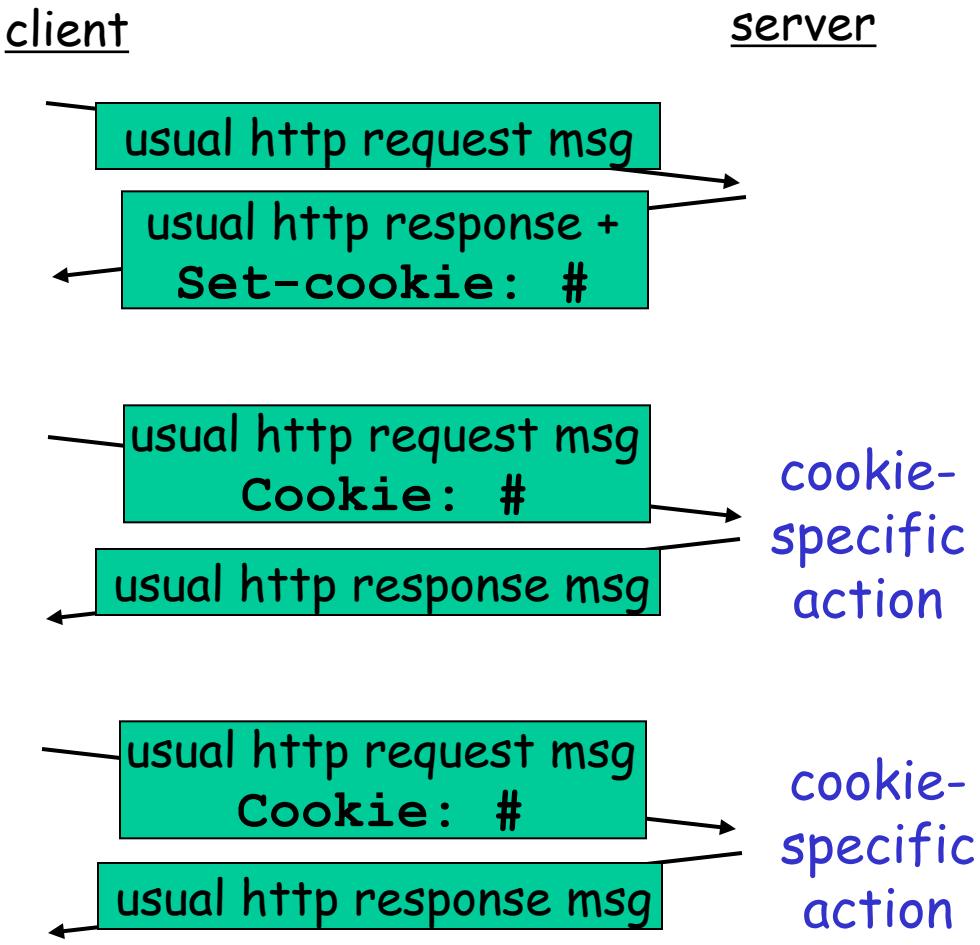
- Server sends "cookie" to client in response msg

  `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



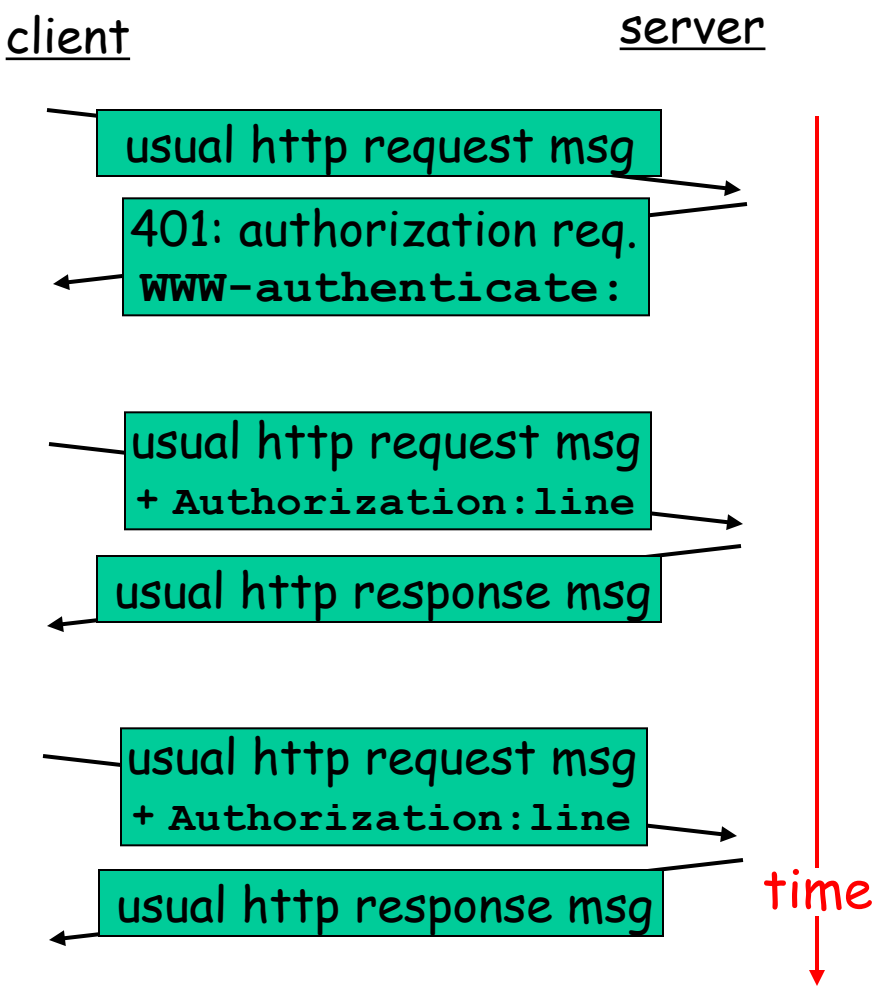
Authentication of Client Request

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:`

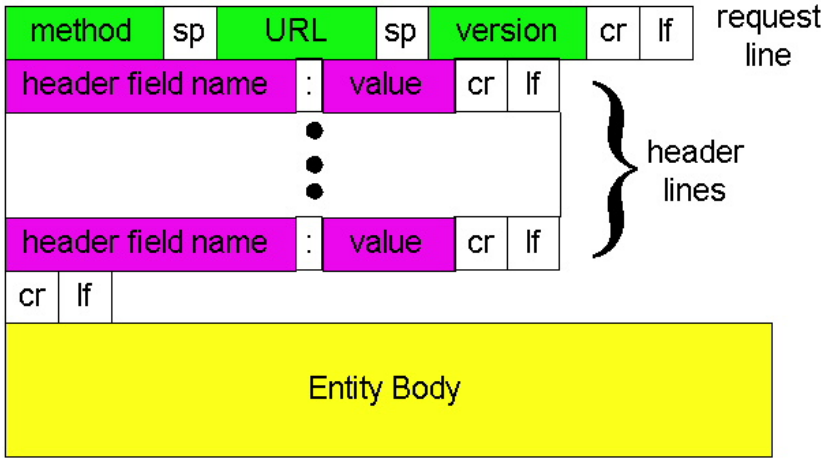
    header line in response

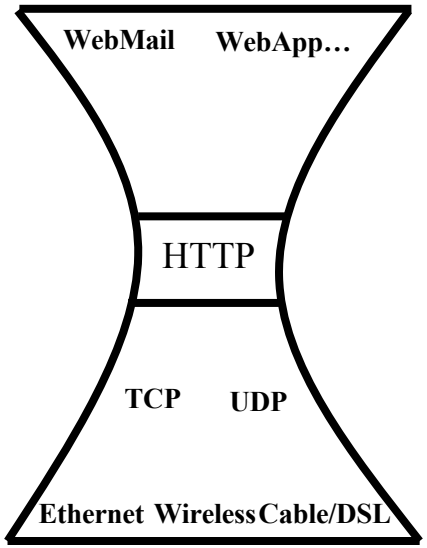


Browser caches name & password so that user does not have to repeatedly enter it.

Recap So Far: HTTP

- C-S app serving Web pages
  - message format
    - request/response line, header lines, entity body
    - simple methods, rich headers
  - message flow
    - stateless server, thus states such as cookie and authentication are needed in each message
- Wide use of HTTP for Web applications
  - Example: RESTful API http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm



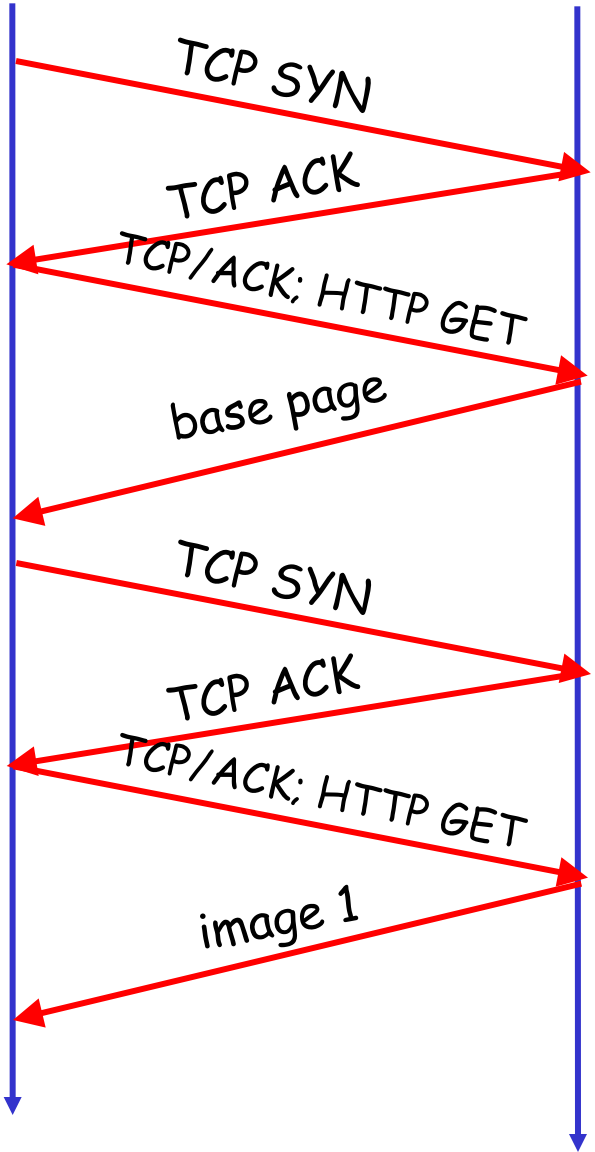


Outline

- Admin and recap
- HTTP
  - Basic service
  - Basic protocol
  - Basic HTTP client/server workflow
  - Stateful interaction using stateless HTTP
  - HTTP "acceleration"

Latency of Basic HTTP Protocol Msg Flow

- >= 2 RTTs per object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)



Web Request Structures

- https://httparchive.org/reports/page-weight
  - 75 requests, 1525KB /page
    - 5 css requests, 50 KB
    - 4 font requests, 97KB
    - 5 HTML requests, 28KB
    - 33 image requests, 651KB
    - 20 javascript requests, 406KB
    - 2 video requests, 2662KB
    - 3 others, 0.5KB

Discussion: How to reduce HTTP latency?

Substantial Efforts to Speedup Basic HTTP/1.0

- Reduce the number of objects fetched [Browser cache]

- Reduce data volume [Compression of data]
- Header compression [HTTP/2]

- Reduce the latency to the server to fetch the content [Proxy cache]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]

- Increase concurrency [Multiple TCP connections]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]

- Server push [HTTP/2]

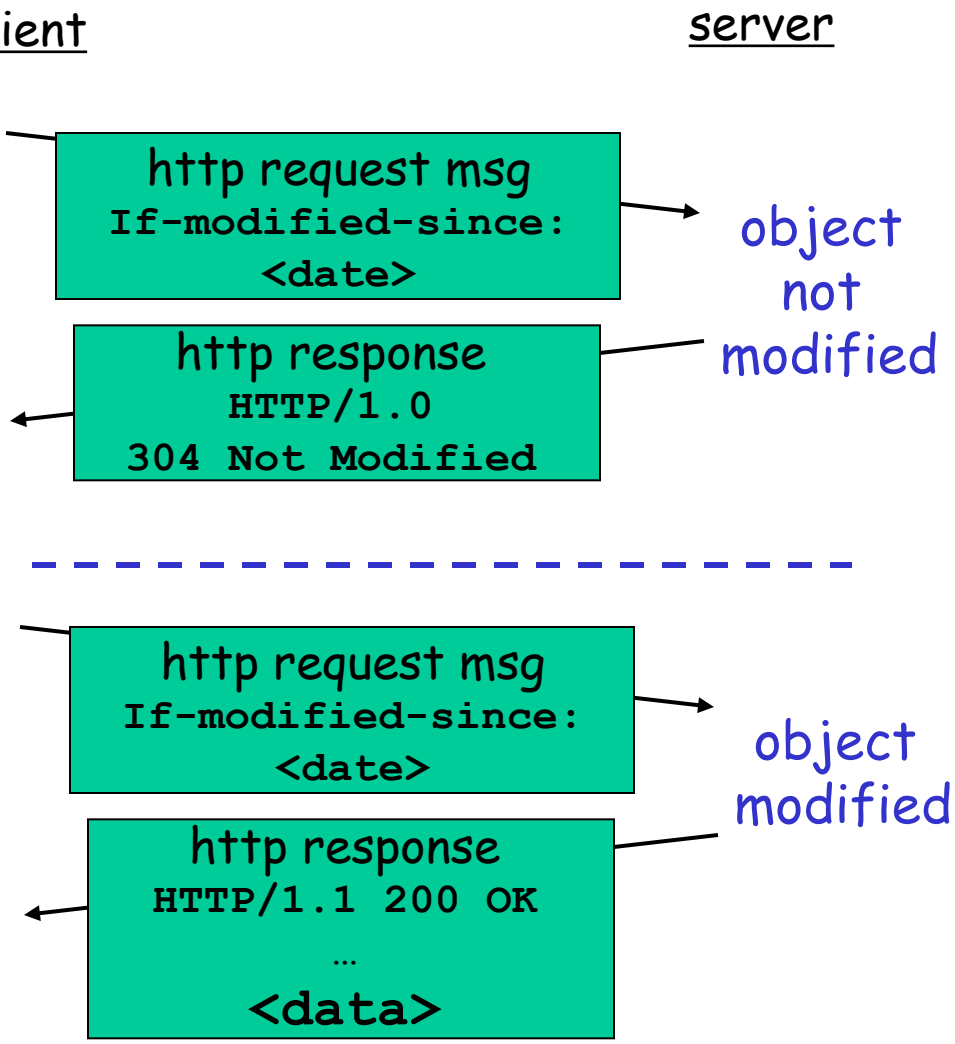
Browser Cache and Conditional GET

- 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>
```

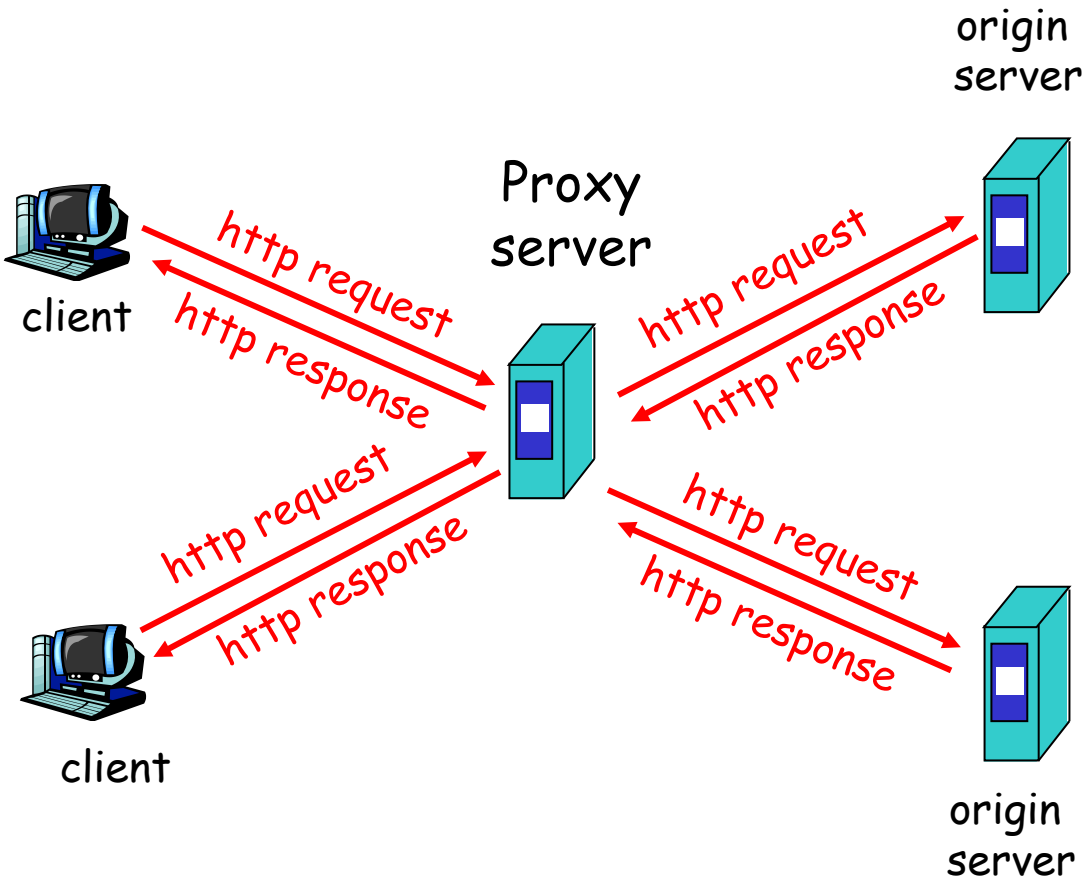
- server: response contains no object if cached copy up-to-date:

```
HTTP/1.0 304 Not
   Modified
```



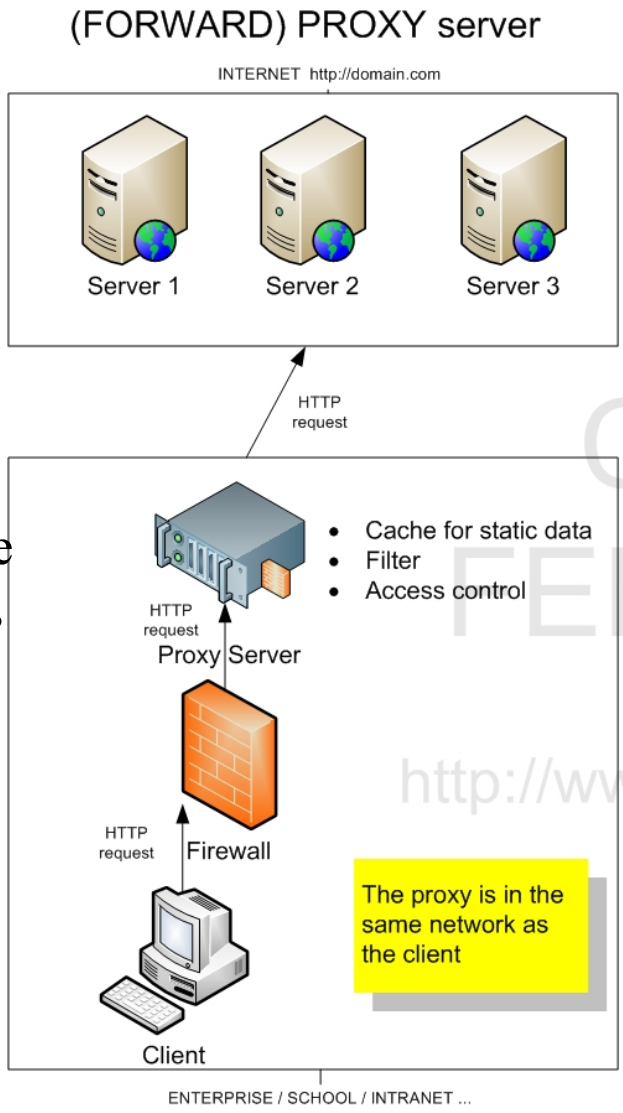
Web Caches (Proxy)

Goal: satisfy client request without involving origin server

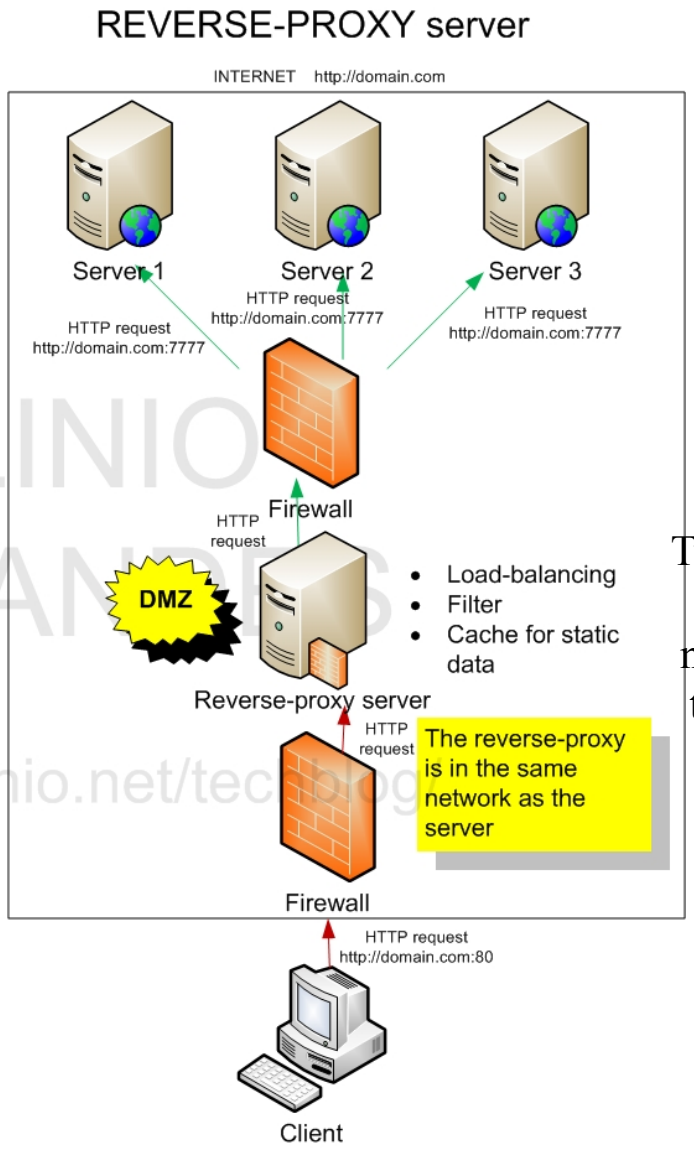


http://www.celinio.net/techblog/?p=1027

Two Types of Proxies



Typically in the same network as the client

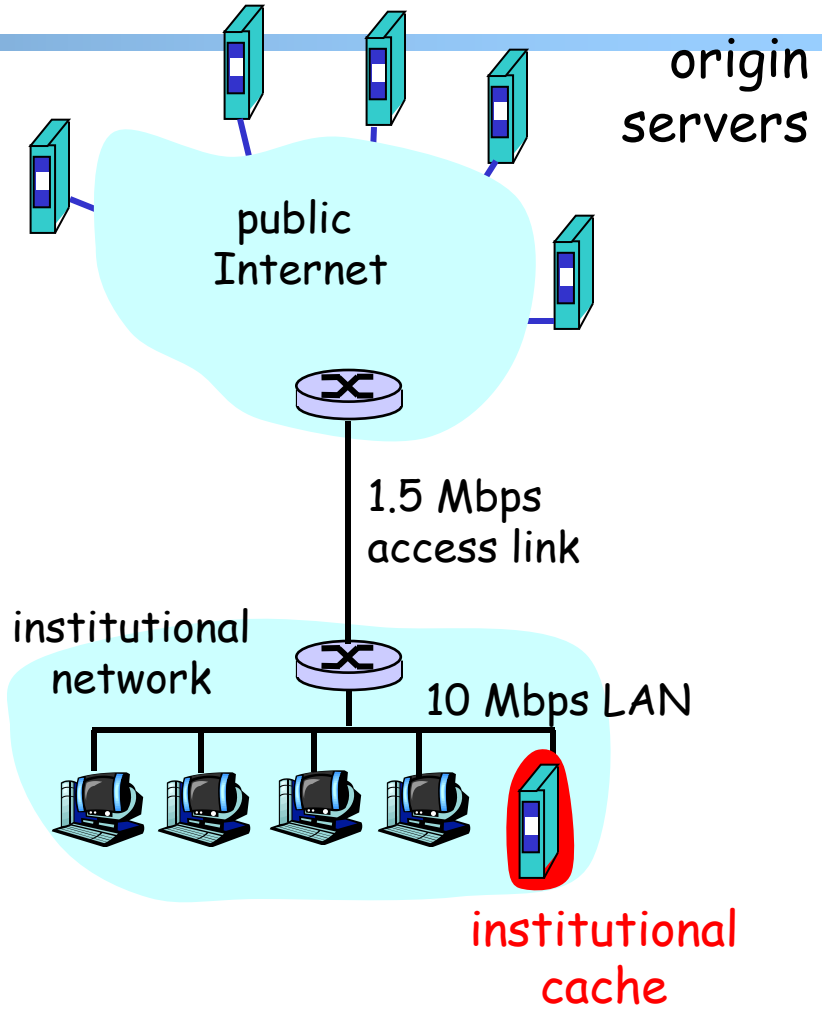


Typically in the same network as the server

Benefits of Forward Proxy

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 is bottleneck



No Free Lunch: Problems of Web Caching

- The major issue of web caching is how to maintain consistency
- Two ways
  - pull
    - Web caches periodically pull the web server to see if a document is modified
  - push
    - whenever a server gives a copy of a web page to a web cache, they sign a lease with an expiration time; if the web page is modified before the lease, the server notifies the cache

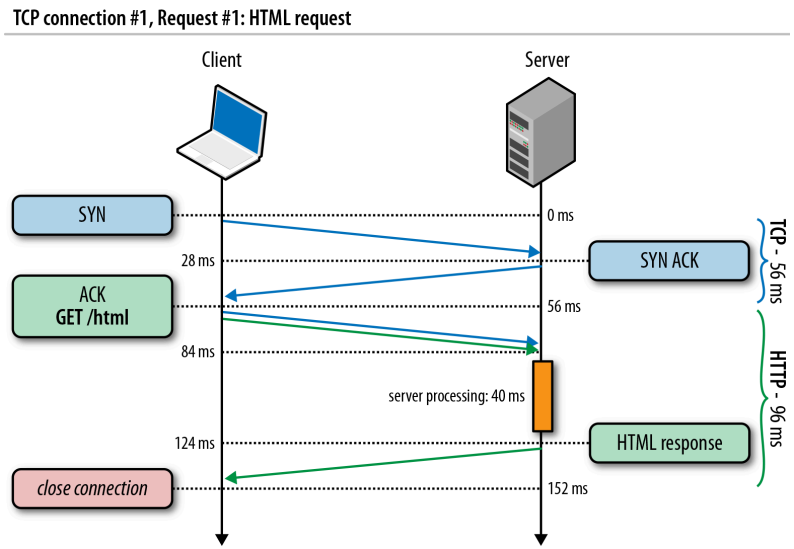
Demo

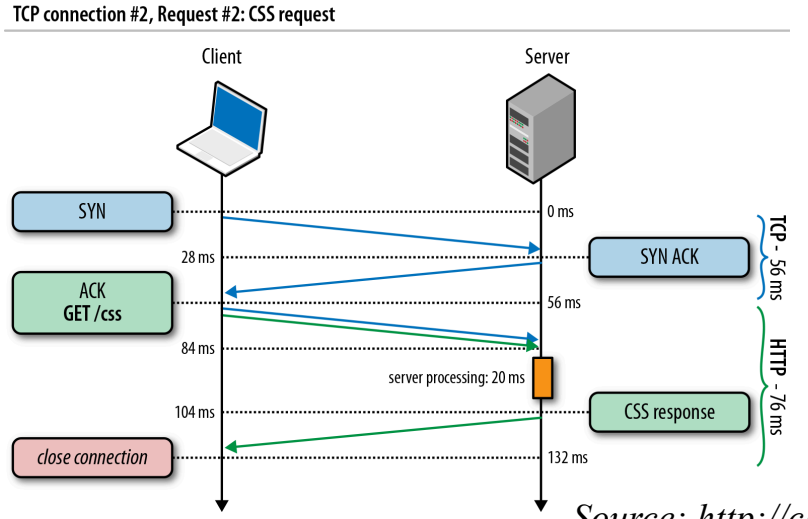
- Load class home page zoo.cs.yale.edu and capture traffic using wireshark

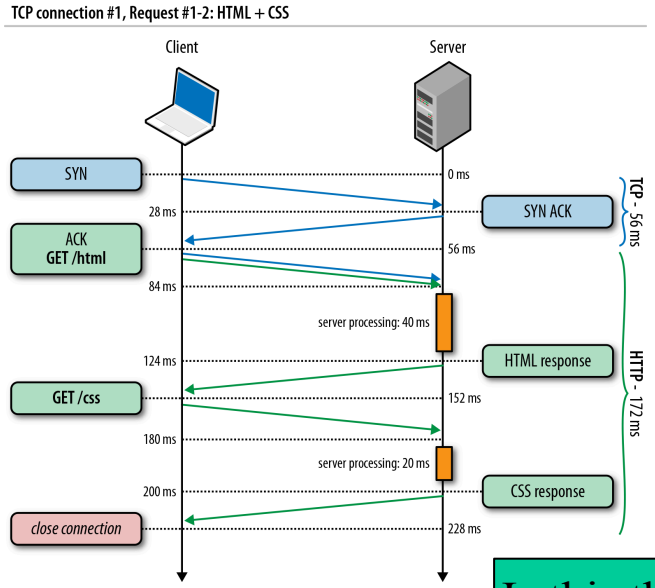
HTTP/1.1: Persistent (keepalive/pipelining) HTTP

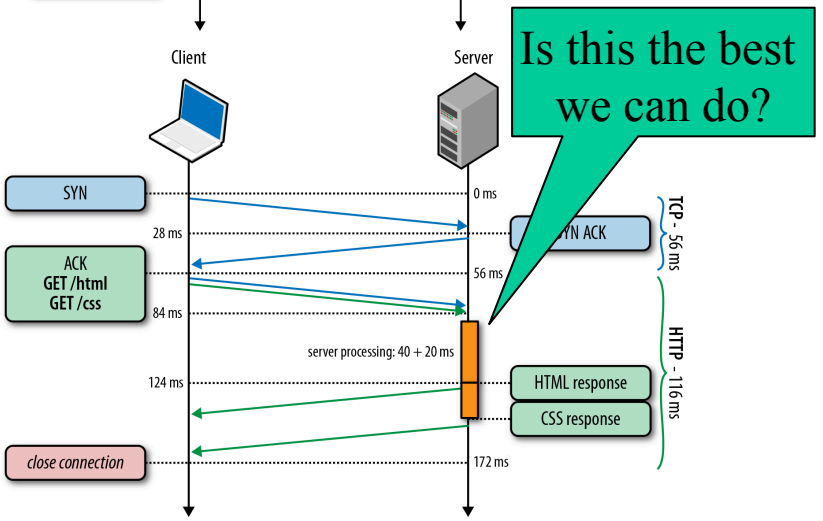
- HTTP/1.0 allows a single request outstanding, while HTTP/1.1 allows request pipelining
  - On same TCP connection: server parses request, responds, parses new request, …
  - Client sends requests for all referenced objects as soon as it receives base HTML
- Benefit
  - Fewer RTTs
  - See Joshua Graessley WWDC 2012 talk: 3x within iTunes

HTTP/1.0, Keep-Alive, Pipelining



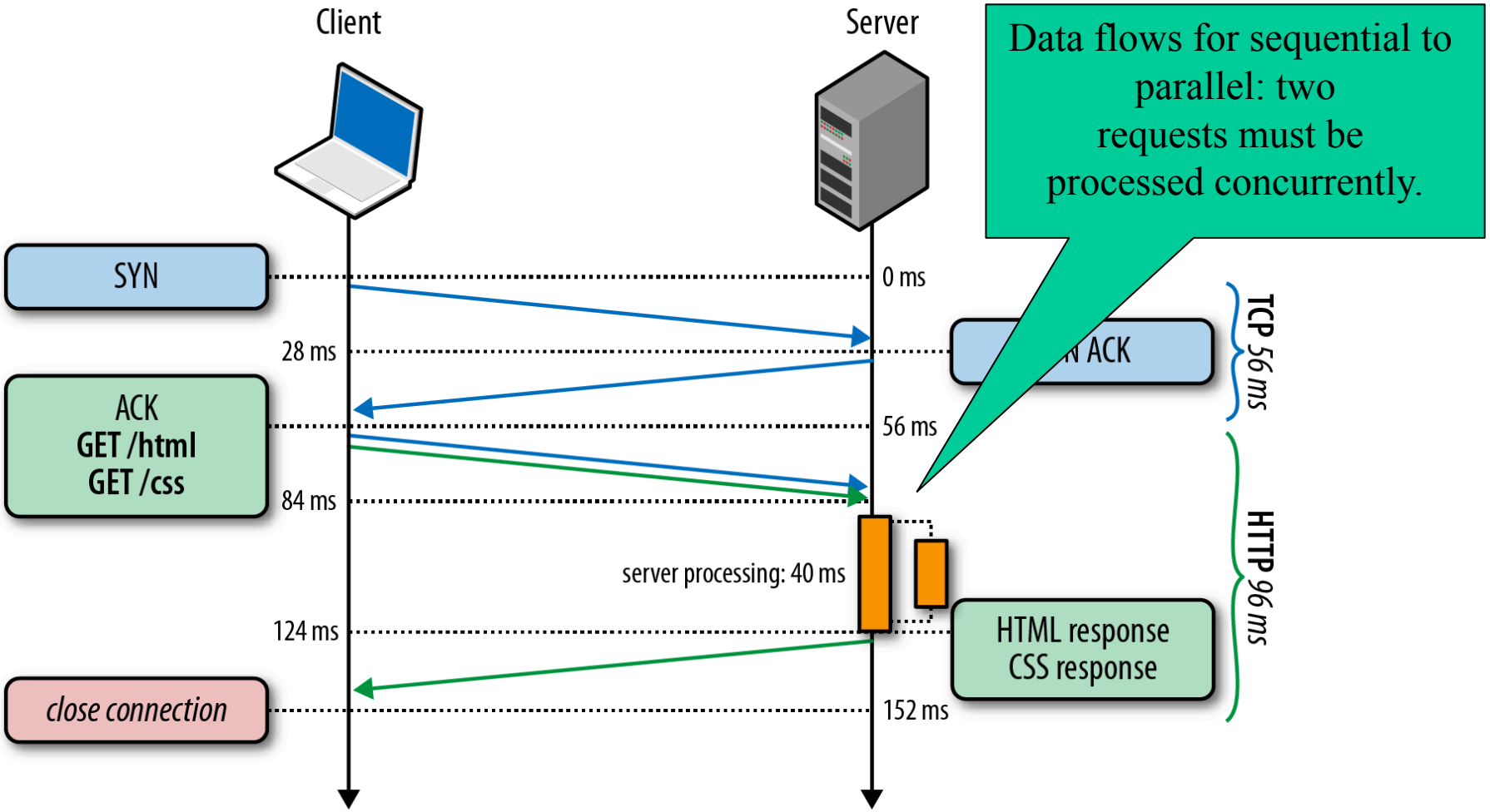






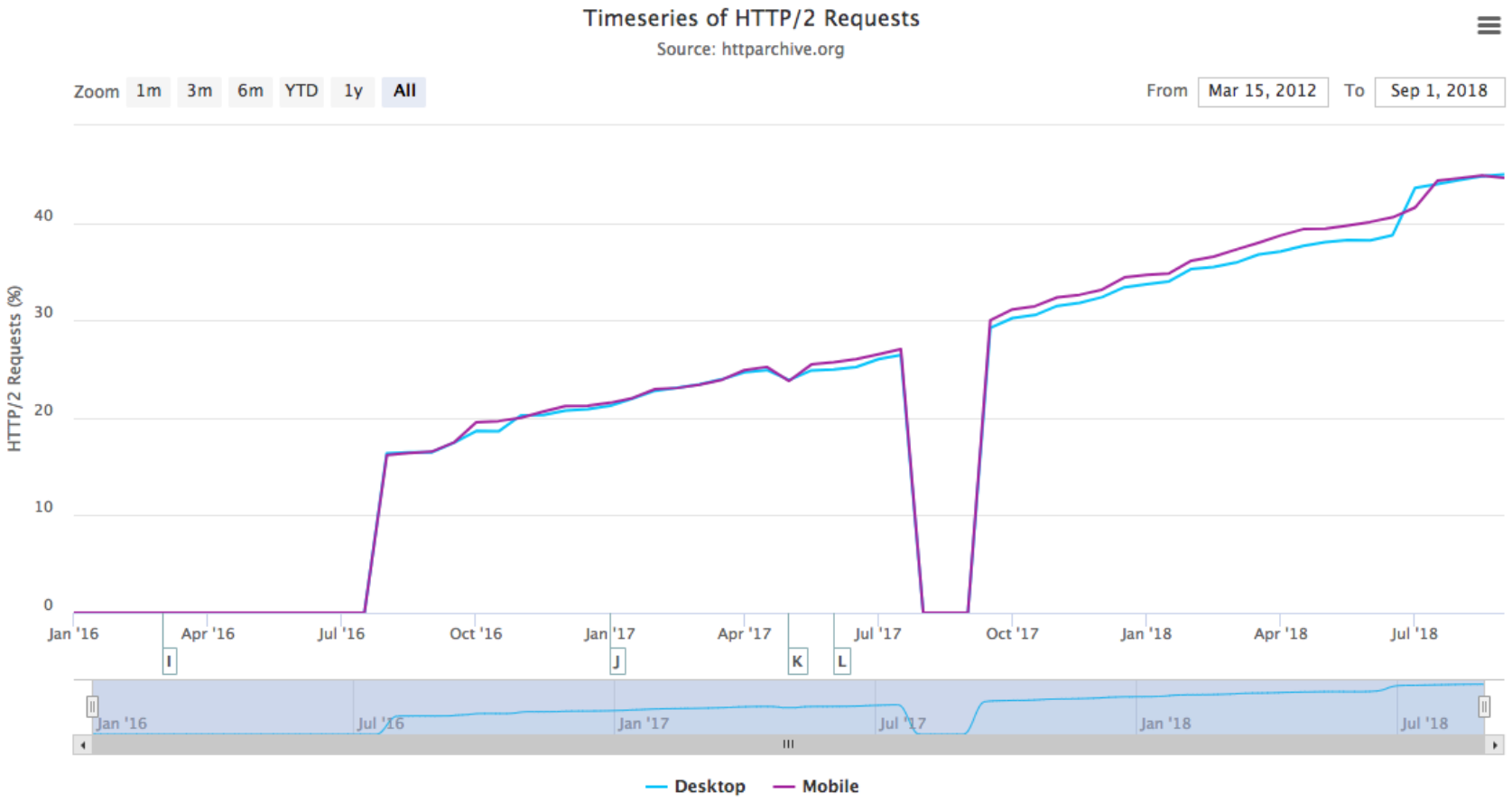
Source: http://chimera.labs.oreilly.com/books/1230000000545/ch11.html

HTTP/2 Basic Idea: Remove Head-of-Line Blocking in HTTP/1.1



Source: http://chimera.labs.oreilly.com/books/1230000000545/ch11.html

HTTP/2 Adoption



https://httparchive.org/reports/state-of-the-web#h2