In-class Quiz

Put phones/smart watches on table at front of the room

Write legibly

Keep your eyes on your own paper

CS 43: Computer Networks

03: HTTP

September 9, 2026

Slides adapted from Kurose & Ross, Kevin Webb, Vasanta Chaganti

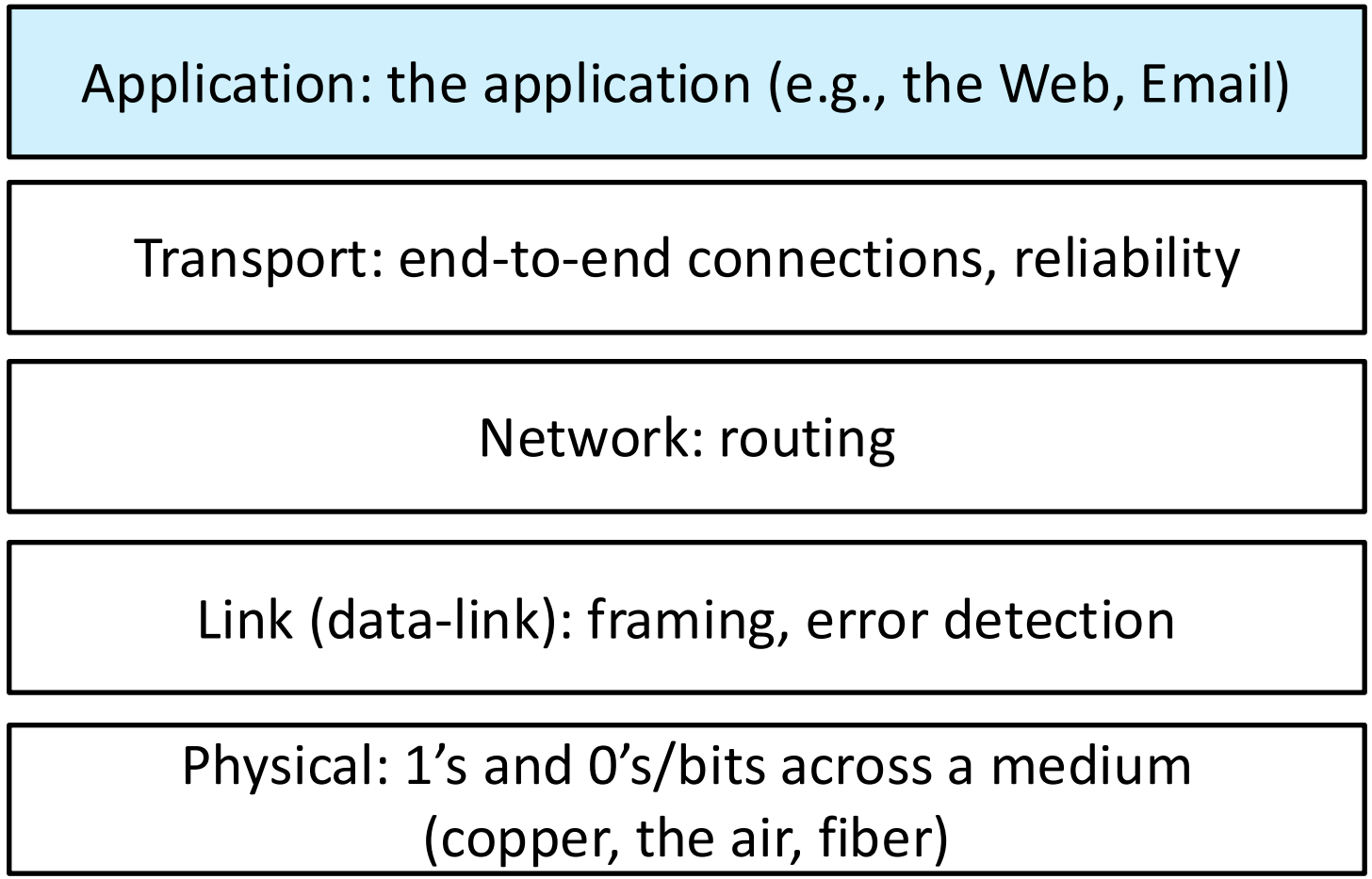
Announcements

- Register your Clicker: https://join.iclicker.com/DQLN

Today

- HTTP
  - What happens when you click www.nytimes.com into a web browser?
  - Why is HTTP considered a stateless protocol and what does that mean for server design?
  - How do changes from version 1.0 to 2.0 improve HTTP performance?

Five-Layer Internet Model

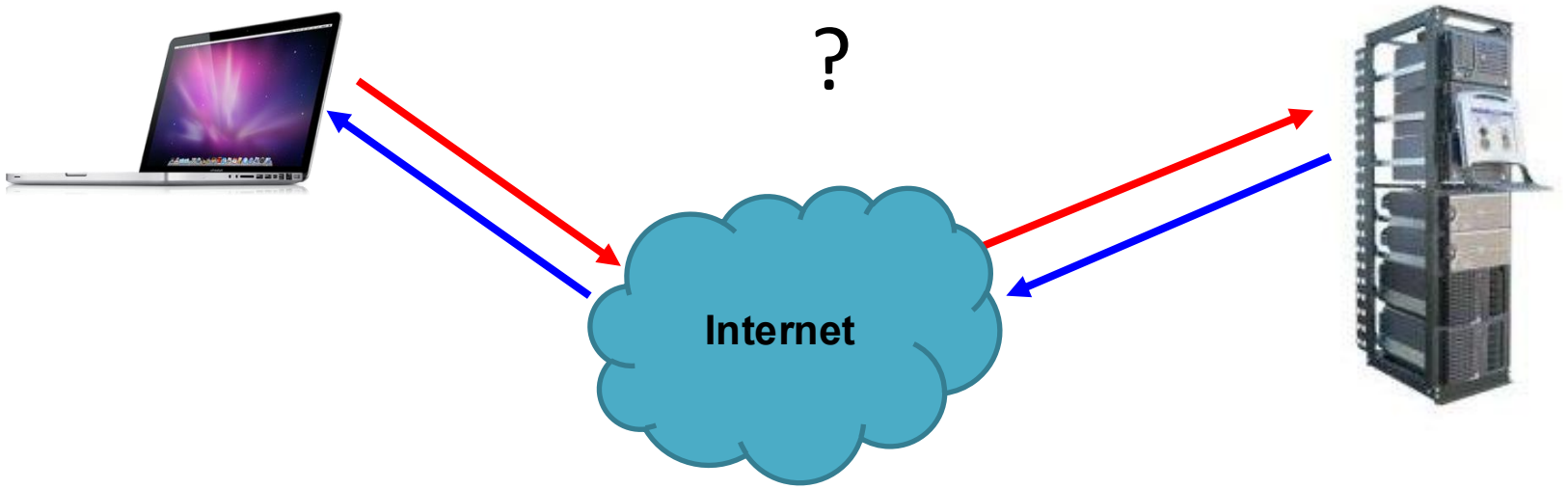


ROUGHLY: what happens when I type google.com into my browser?

Web servers are always on, waiting for requests.

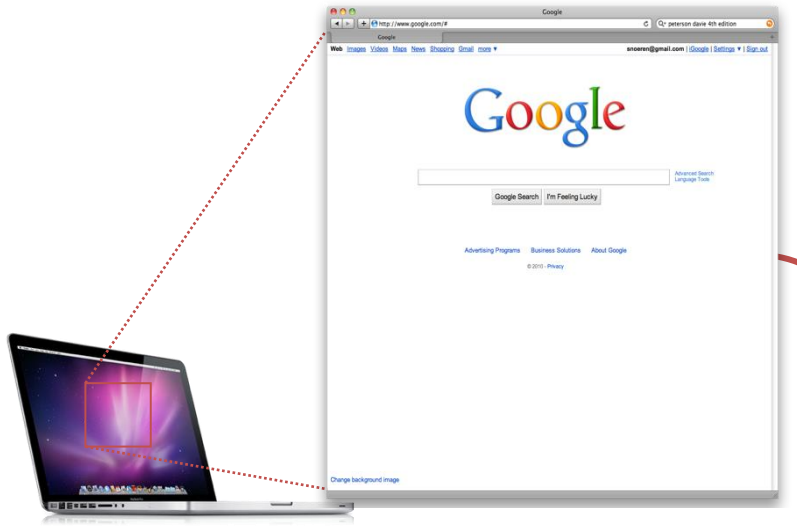
My computer

www.google.com



Application Layer: Web request (HTTP)

- Turn click into HTTP request




```
GET /index.html HTTP/1.1
Host: www.google.com
...
```

Can two websites have the same domain name?

A. Yes
B. No

Can two websites have the same domain name?

A. Yes

B. No

How are they assigned? We'll learn about this in a future lecture…

Can two web servers have the same domain name?

A. Yes
B. No

Can two web servers have the same domain name?

A. Yes

B. No

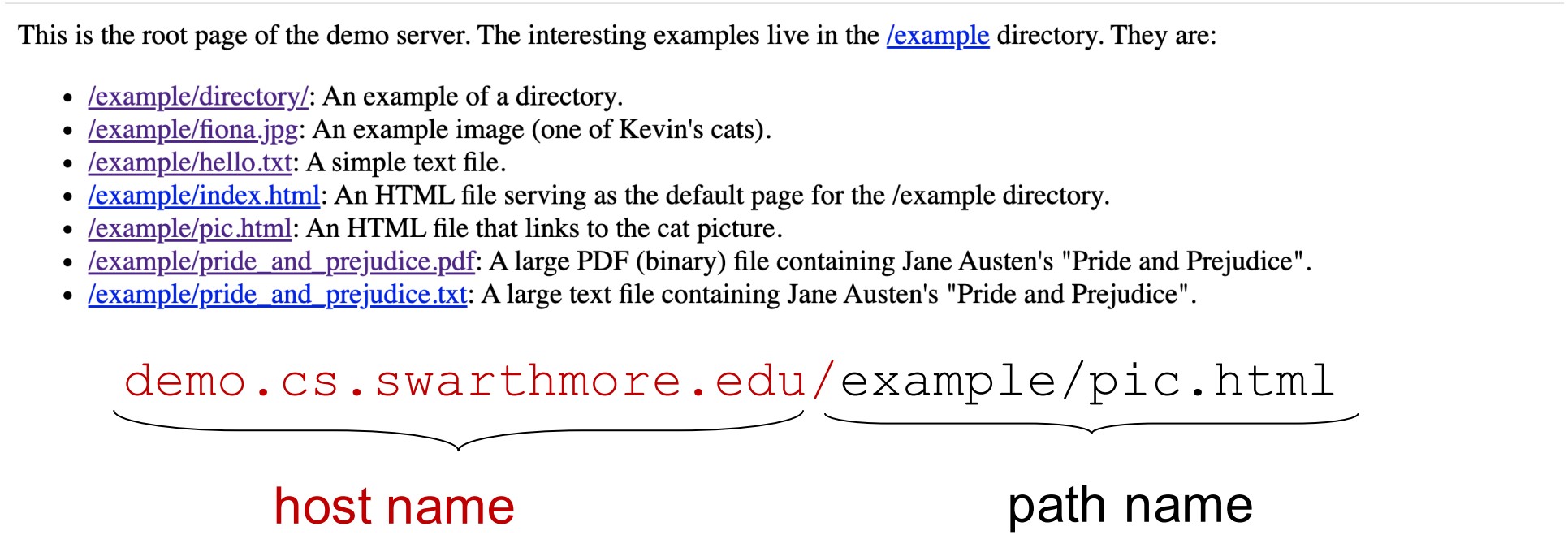
There are usually many web servers serving content for the same domain name so one web server isn't responsible for handling every web request.

Application Layer: Name resolution (DNS)

- Where is www.google.com?

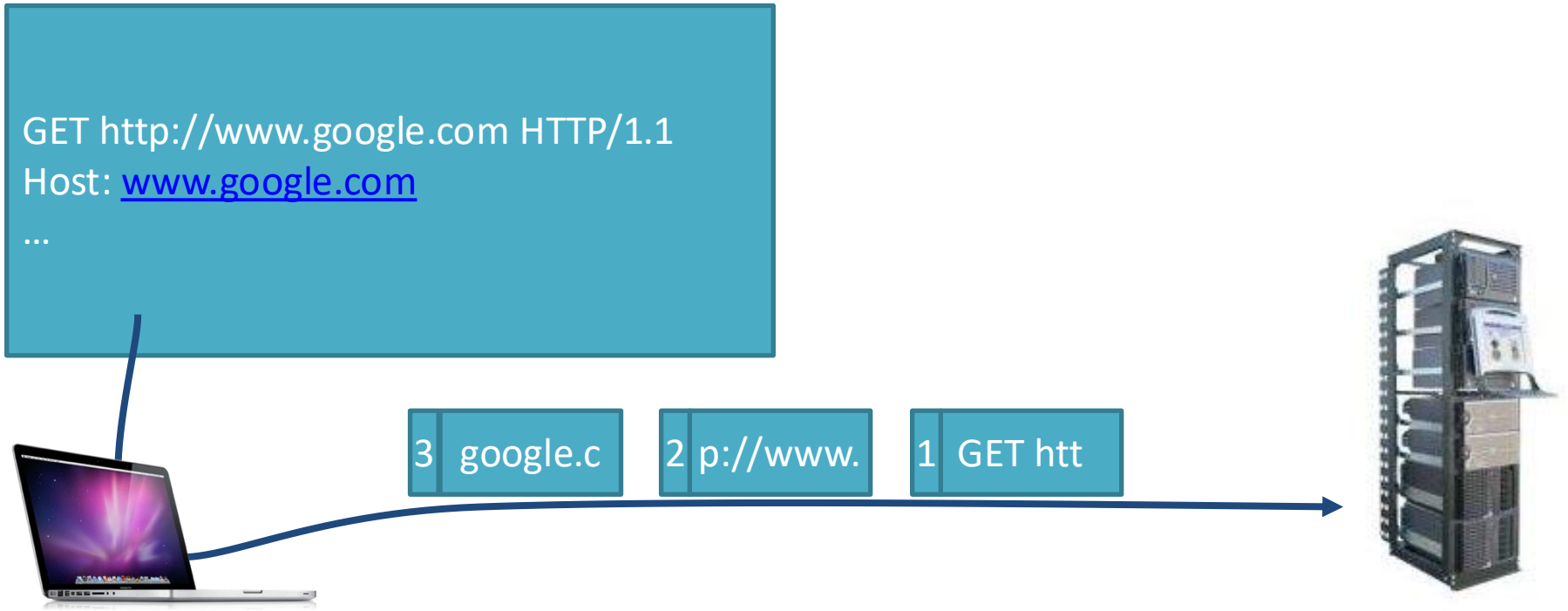
Networking Systems Design Concept: Naming

- How do I identify the objects on any webserver?
- Objects are uniquely addressable from URLs



Transport Layer: TCP

- Break message into packets (TCP segments)
- Should be delivered reliably & in-order



TCP distinguishes processes based on port numbers. How doe the Web client know which port number to connect to?

- There are standard port numbers for various protocols
- HTTP is usually at port 80 and HTTPS at 443

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

- There are cybersecurity risks of having always on servers at well-known port numbers…more on this later in the semester…

Let’s look at HTTP in a browser…

HTTP request message

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

carriage return character
line-feed character

request line (GET, POST,
HEAD commands)

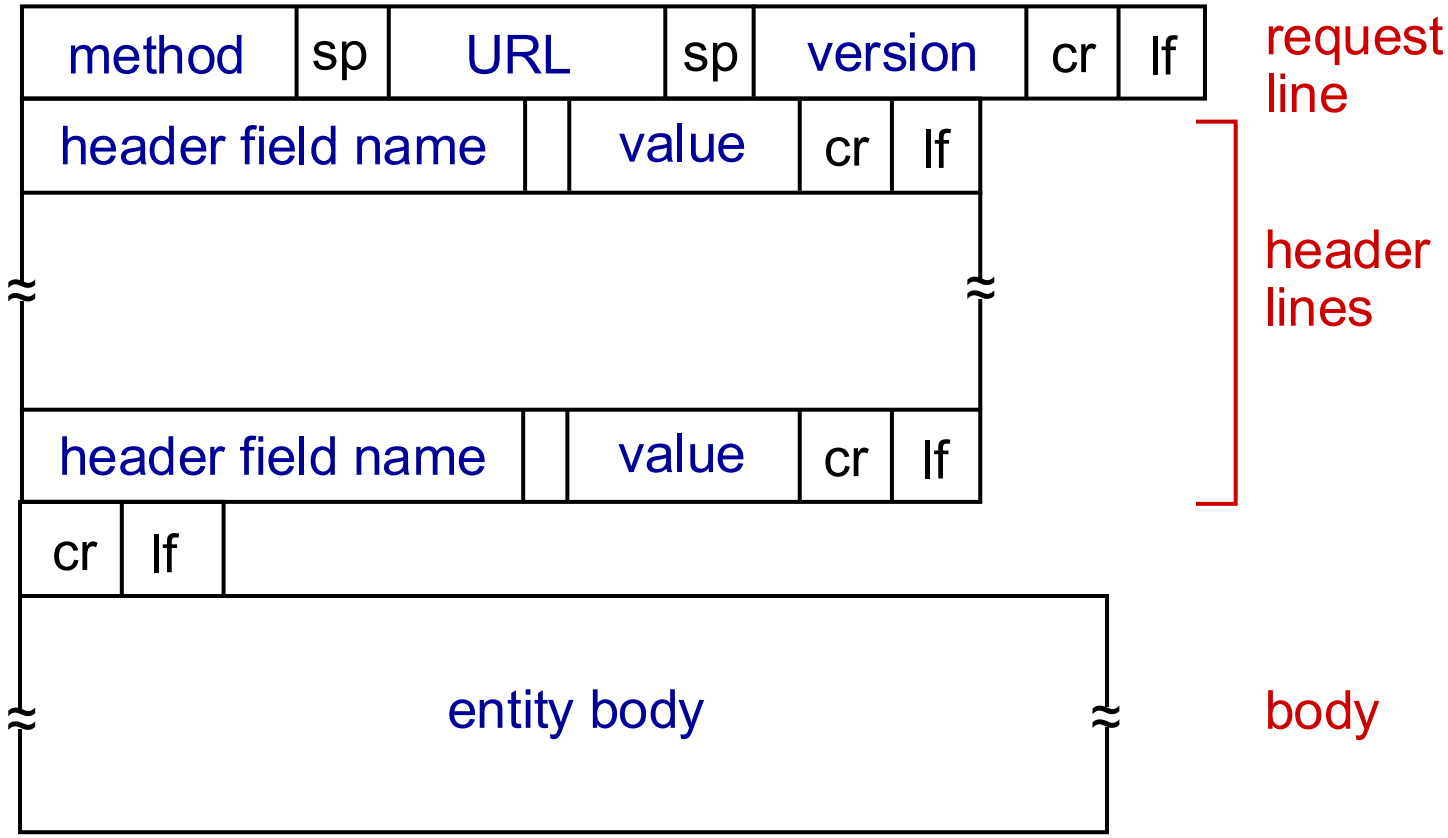
carriage return, line feed
at start of line indicates
end of header lines

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

Why CRLF?

- HTTP/1.x is a text protocol with no length field in the header
- Need an unambigous marker for “the header line is done”
- \r (carriage return) and \n (line feed) come from teletype machines where the two actions were separate

HTTP request message: general format



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

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

```
www.somesite.com/animalsearch?monkeys&banana
```

HEAD method:

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

PUT method:

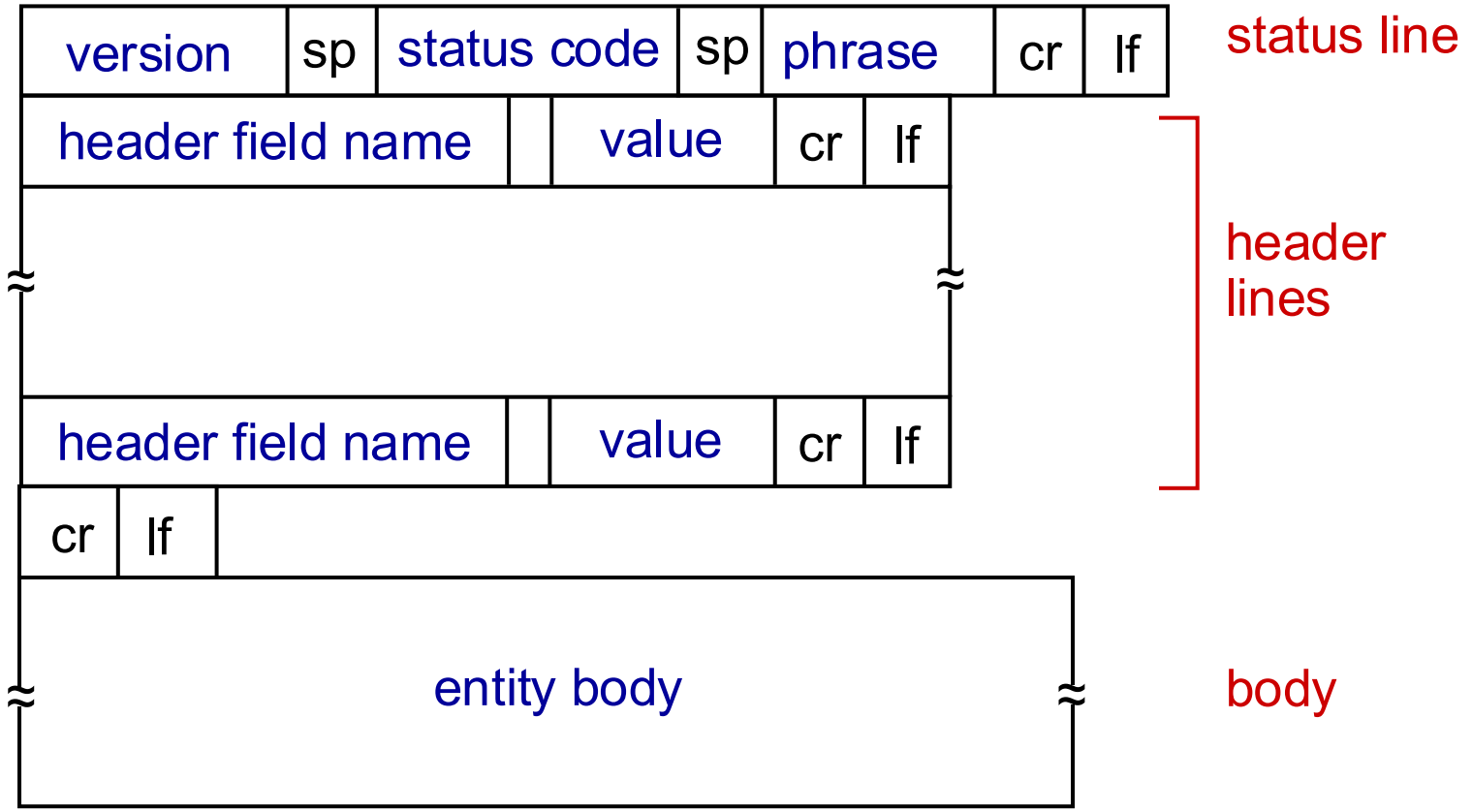
- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
```

HTTP response message: general format



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

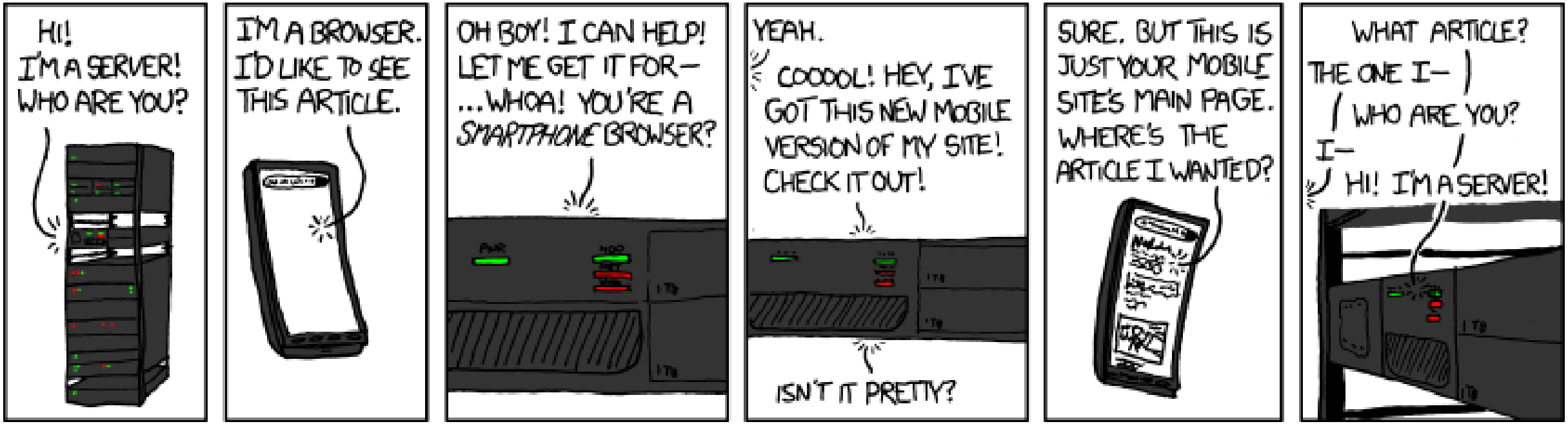
- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

State(less)



(XKCD #869, “Server Attention Span”)

HTTP called a stateless protocol because…

A. The server maintains no information about past client requests
B. The connection is closed after every response
C. HTTP messages contain no header fields
D. The server does not know the client's IP address
E. HTTP provides no reliability guarantees

HTTP called a stateless protocol because…

A. **The server maintains no information about past client requests**
B. The connection is closed after every response
C. HTTP messages contain no header fields
D. The server does not know the client's IP address
E. HTTP provides no reliability guarantees

State(less)

- Original web: simple document retrieval
- Maintain State? Server is not required to keep state between connections

  ...often it might want to though
- Authentication: Client is not required to identify itself
  - server might refuse to talk otherwise though

User-server state: cookies

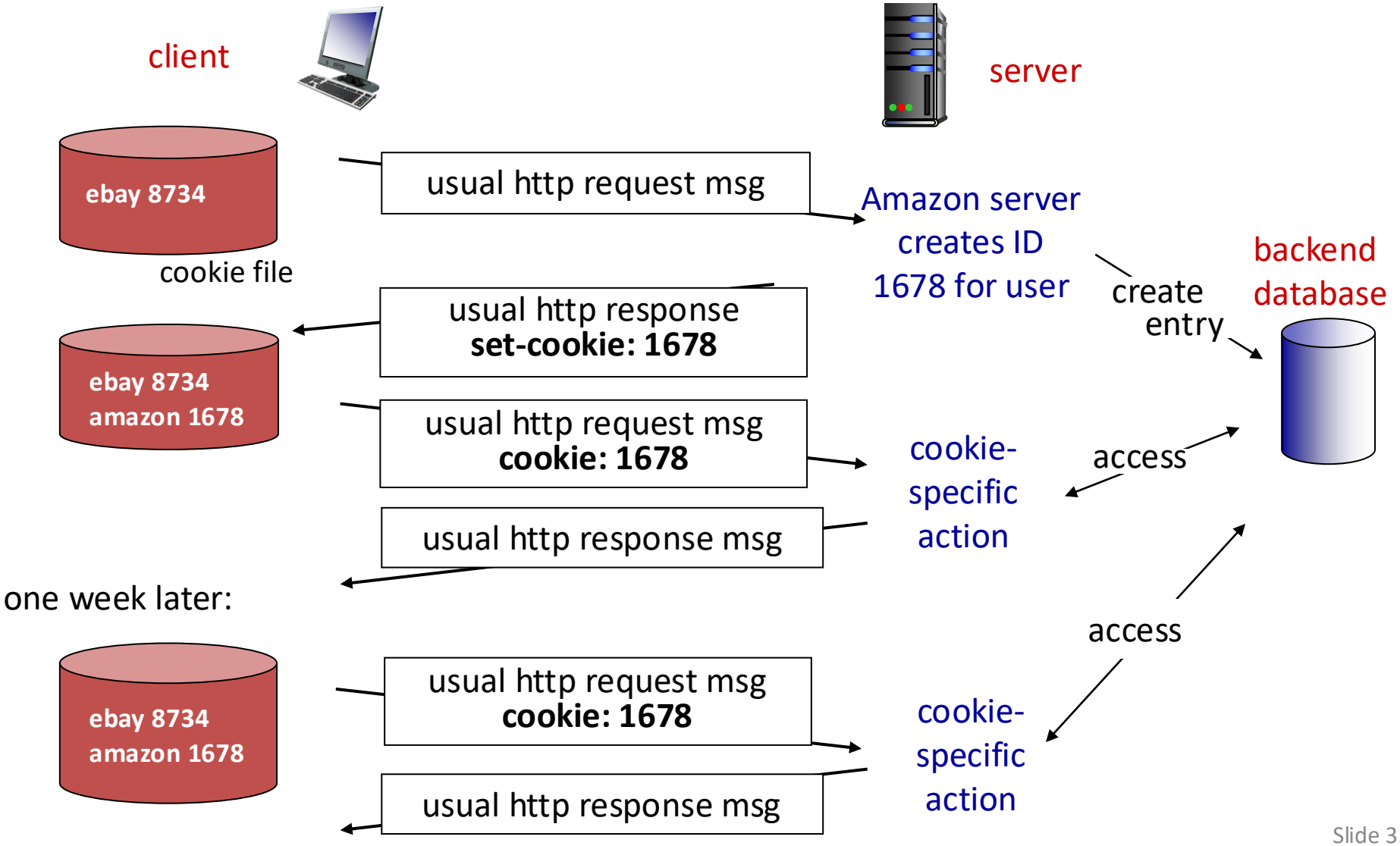
What cookies can be used for:

- 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

Cookies: keeping "state" (cont.)



User-server state: cookies

Many web sites use cookies

Four components:

1) cookie header line of HTTP response message

2) cookie header line in next HTTP request message

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

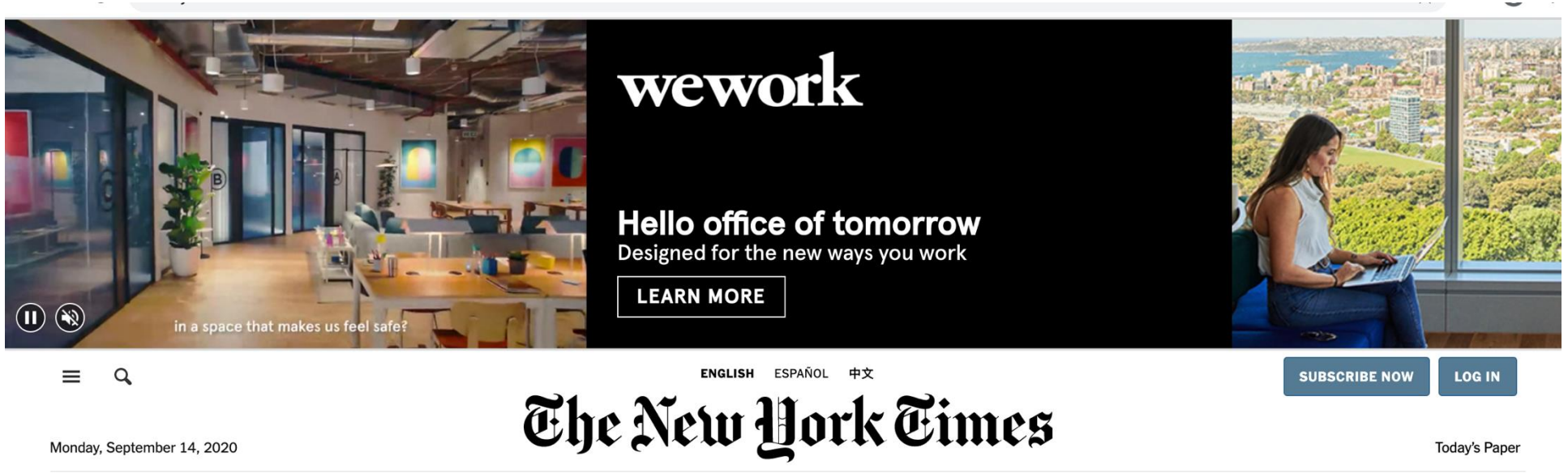
4) back-end database at Web site

Cookies and Privacy

Cookies permit sites to learn a lot about you

supply name and e-mail to sites (and more!)

third-party cookies (ad networks) follow you across multiple sites.



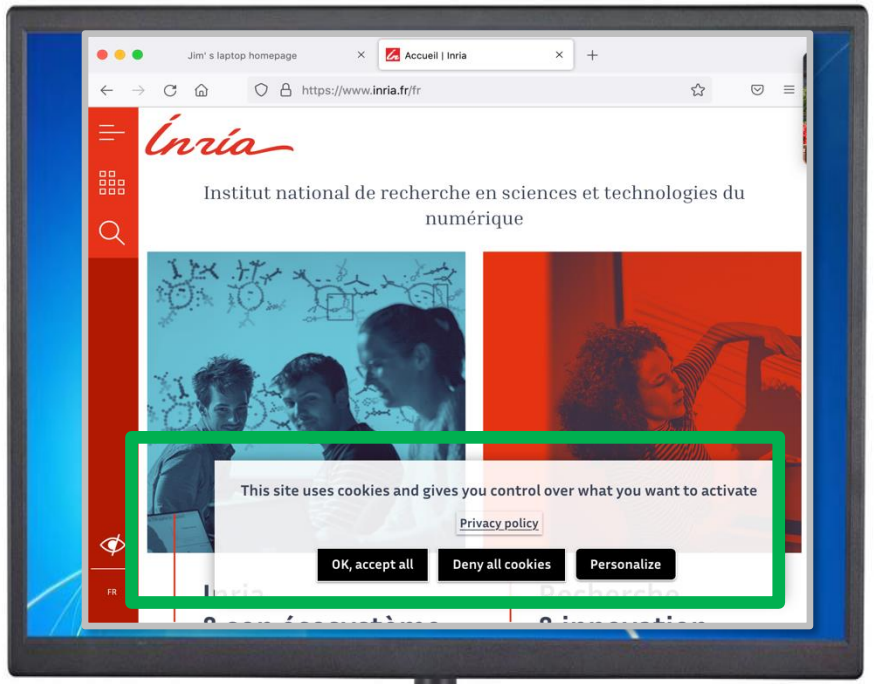
GDPR (EU General Data Protection Regulation) and cookies

"Natural persons may be associated with online identifiers [...] such as internet protocol addresses, cookie identifiers or other identifiers [...].

This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them."

GDPR, recital 30 (May 2018)

when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations



User has explicit control over whether or not cookies are allowed

HTTP connections

Non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects requires multiple connections

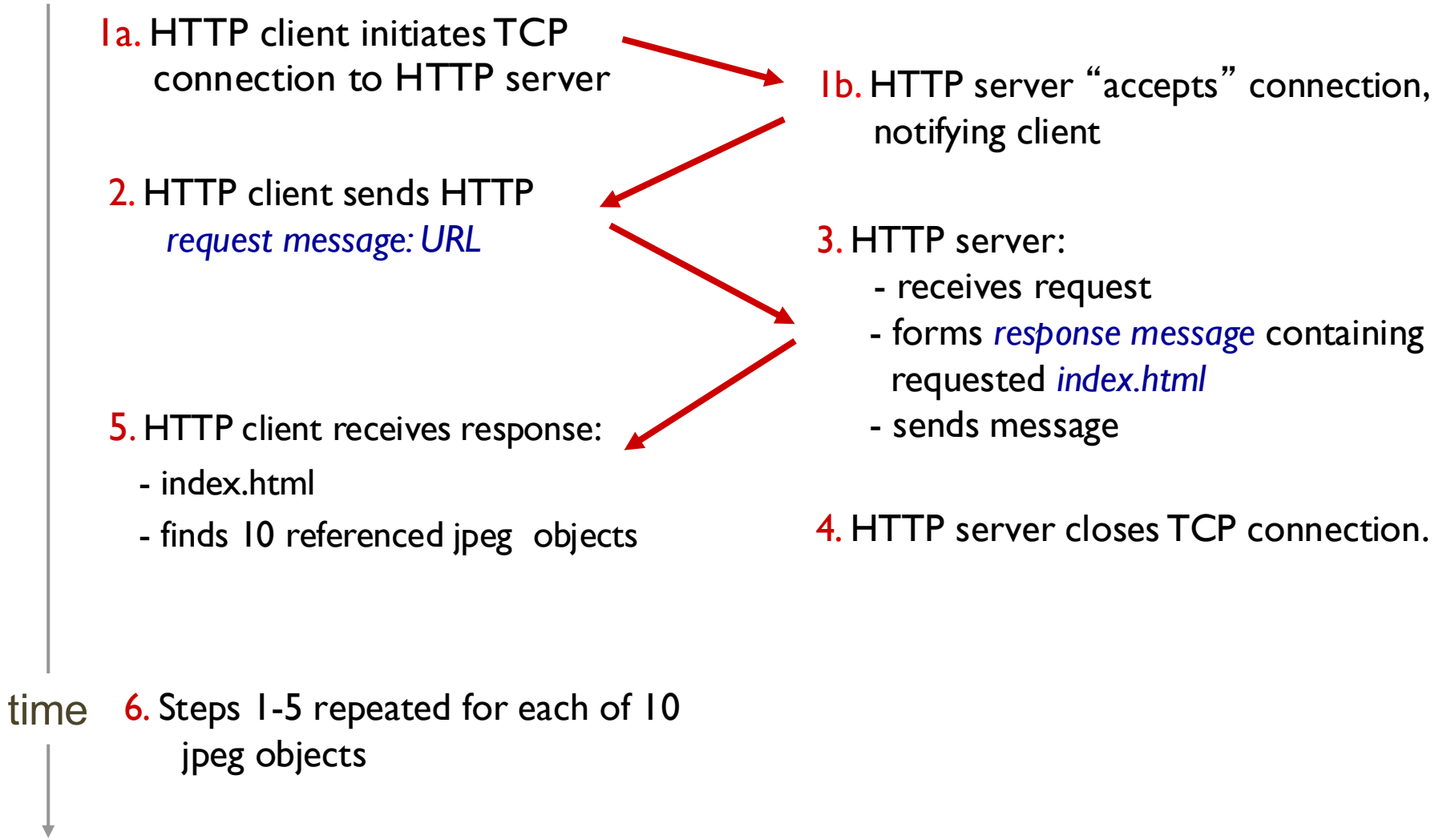
Persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

object: image, script, stylesheet, etc.

Non-persistent HTTP

suppose user enters URL: contains references to 10 jpeg images



Pseudocode Example

non-persistent HTTP

```
for object on web page:
    connect to server
    request object
    receive object
    close connection
```

persistent HTTP

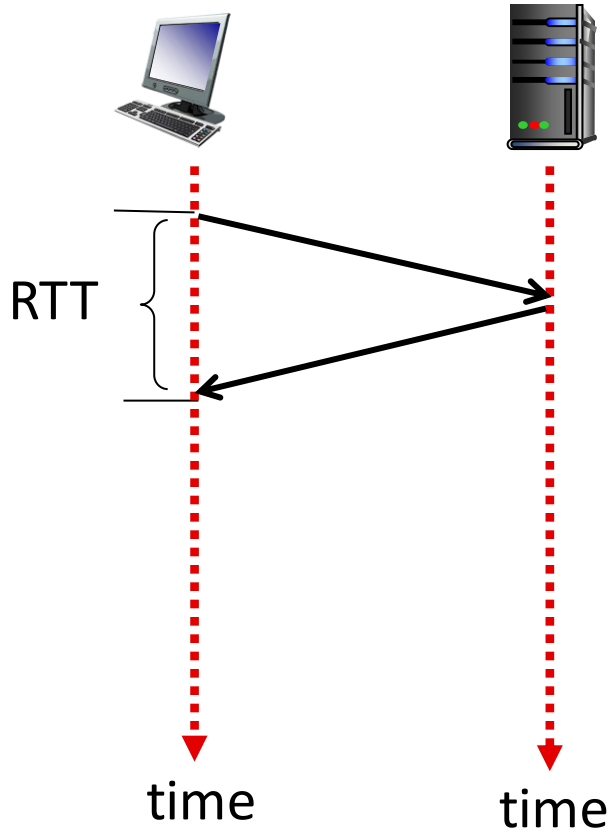
```
connect to server
for object on web page:
    request object
    receive object
close connection
```

Round Trip Time

Round Trip Time (RTT):

- time for a small packet to travel from client to server and response to come back.

- Connection establishment (via TCP) requires one RTT.



Non-Persistent HTTP Connections can download a website with several objects in…

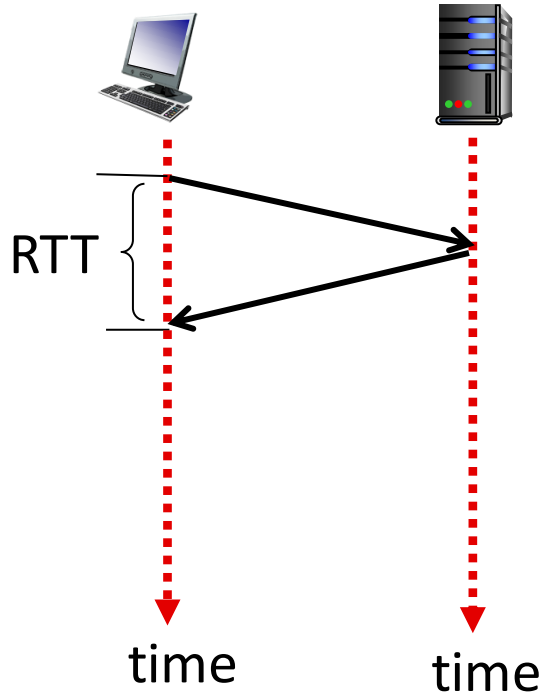
A. One RTT + (File transfer time per object)

B. (One RTT + File transfer time) per object

C. Two RTTs

D. Two RTTs + (File transfer time per object)

E. (Two RTTS + File transfer time) per object



Non-Persistent HTTP Connections can download a website with several objects in…

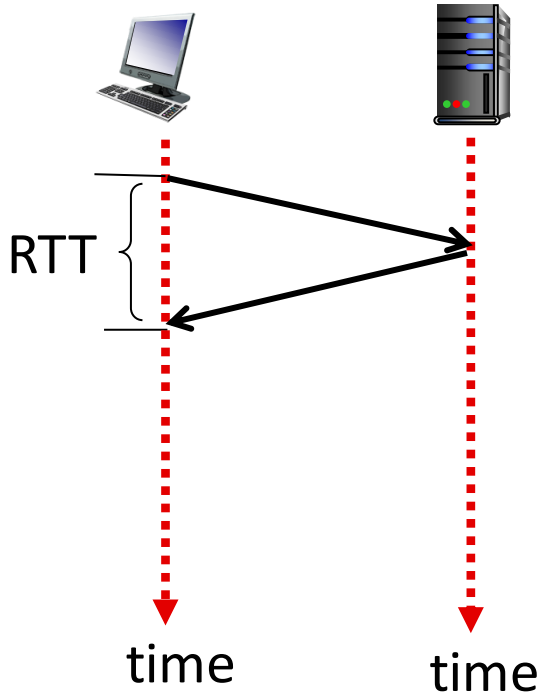
A. One RTT + (File transfer time per object)

B. (One RTT + File transfer time) per object

C. Two RTTs

D. Two RTTs + (File transfer time per object)

E. (Two RTTS + File transfer time) per object



Non-persistent HTTP: response time

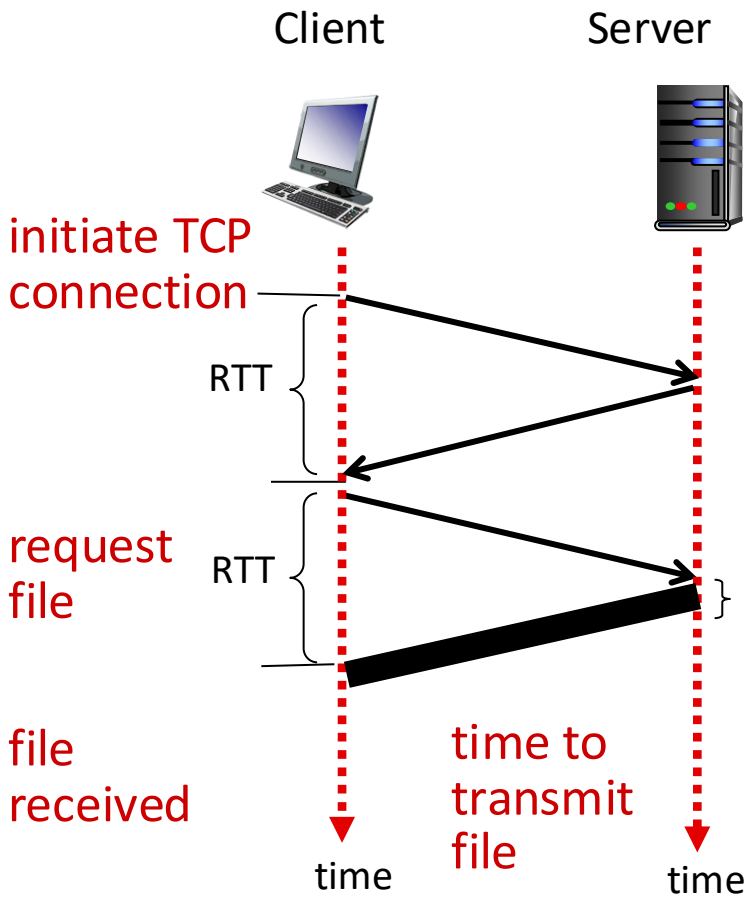
Round Trip Time (RTT): time for a small packet to travel from client to server and back

HTTP response time:

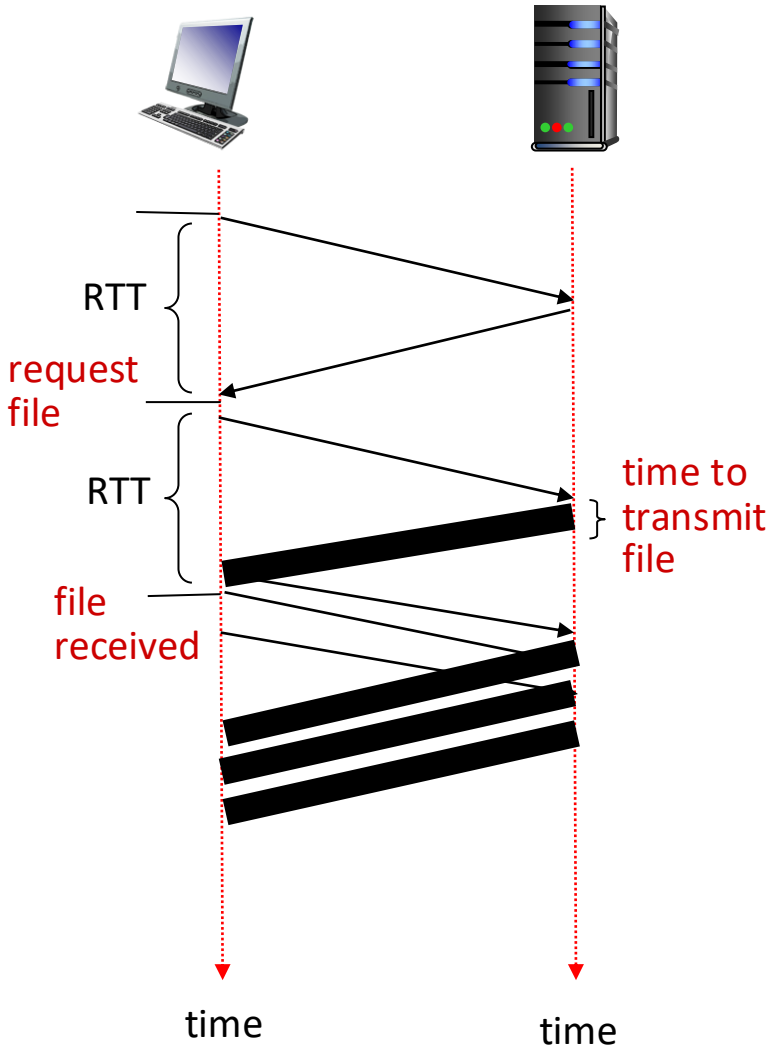
- 1-RTT to initiate TCP connection
- 1-RTT for HTTP request + first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2-RTT+ file transmission time

  For each object



Persistent Connection



Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

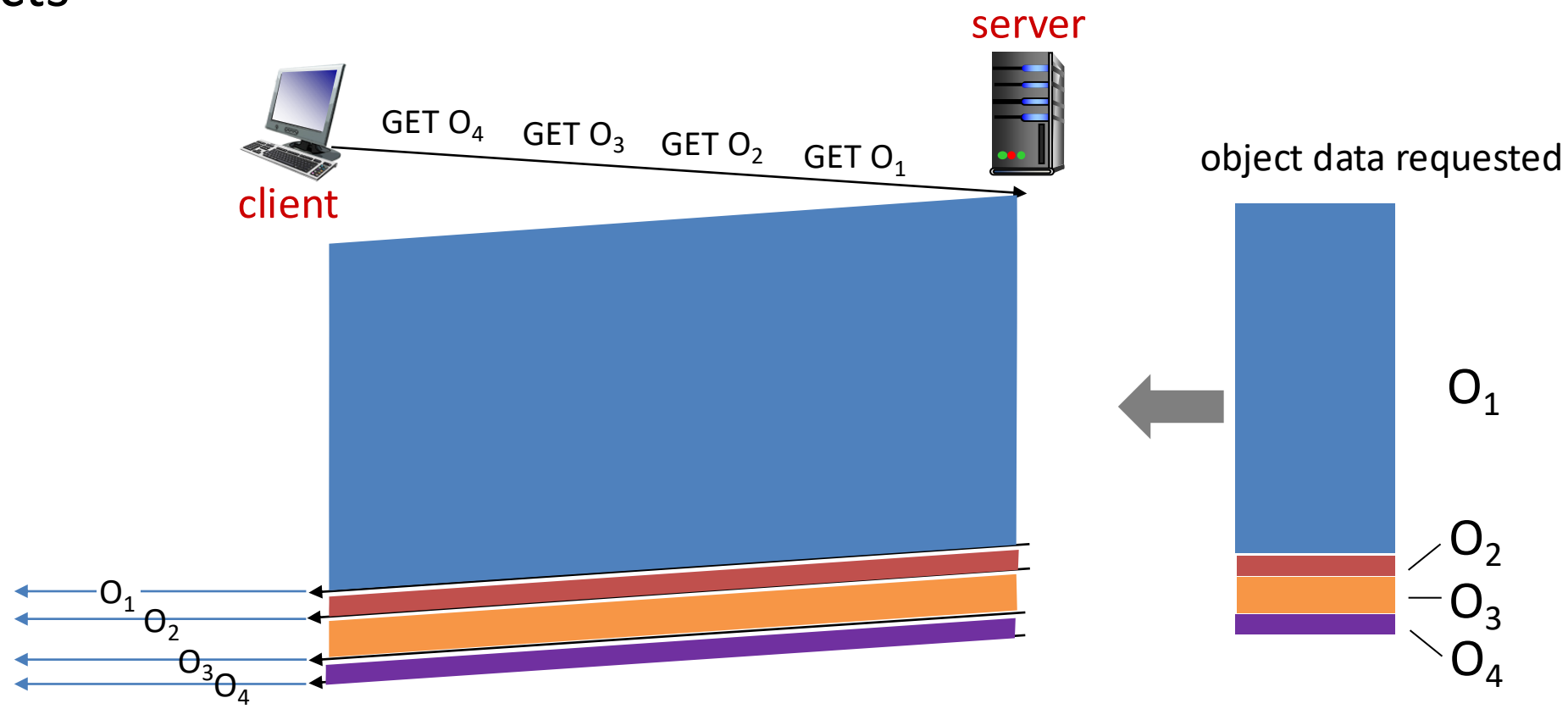
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

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)

New Problem: Head-of-Line-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

Can I just open a parallel TCP connection for each object to solve head-of-line blocking?" Which response is most accurate?

A. Yes, and it's free: each connection is independent, so there's no downside to opening as many as you have objects
B. It helps, but each flow still has to do connection setup which is costly
C. No, parallel connections do nothing for HOL blocking
D. No, opening a separate connection per object is impossible

Can I just open a parallel TCP connection for each object to solve head-of-line blocking?" Which response is most accurate?

A. Yes, and it's free: each connection is independent, so there's no downside to opening as many as you have objects
B. It helps, but each flow still have to do connection setup which is costly
C. No, parallel connections do nothing for HOL blocking
D. No, opening a separate connection per object is impossible

Many browsers do this anyway!

This is also costly because of "congestion" and "unfairness" which we will talk more about when we talk about the transport layer

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

HTTP/2

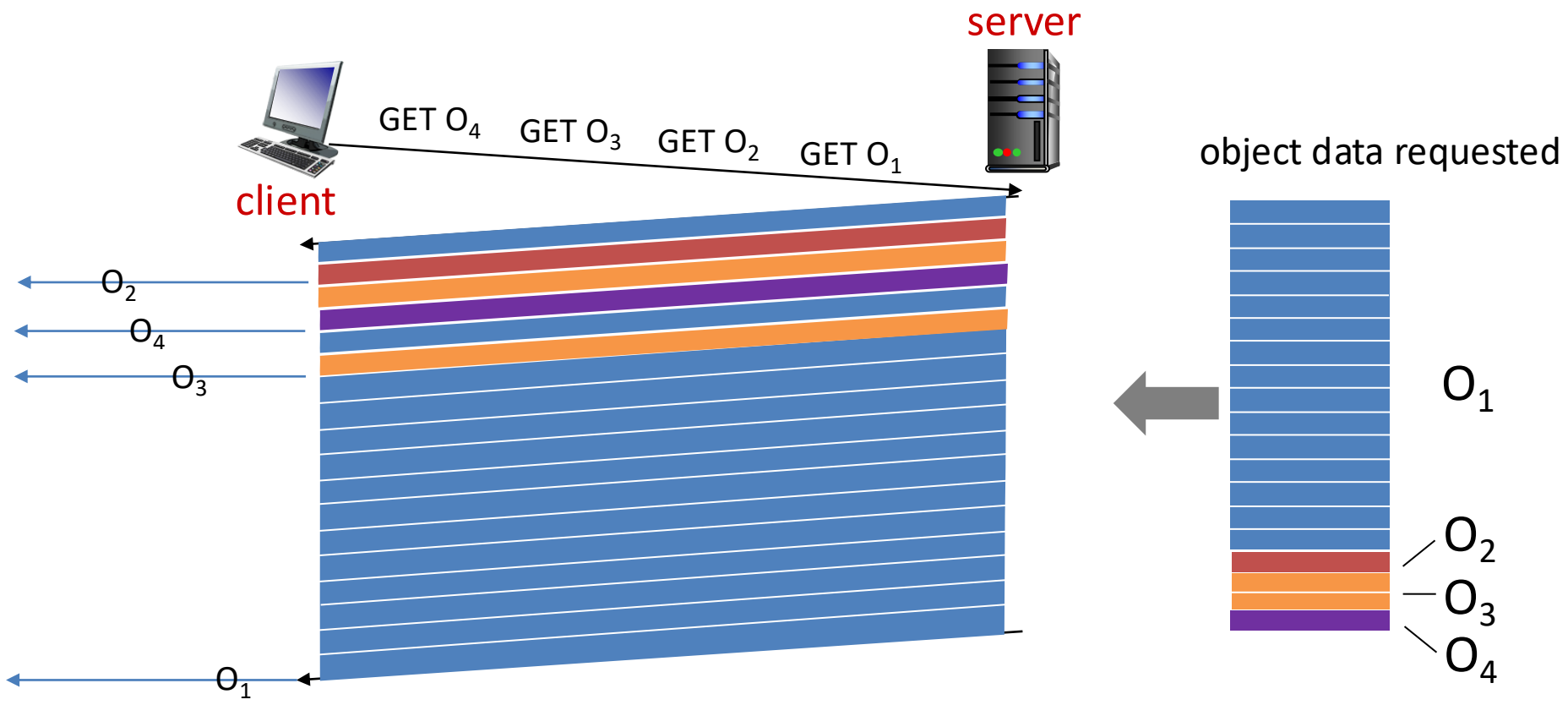
Key goal: decreased delay in multi-object HTTP requests

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/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

So far to improve HTTP performance…

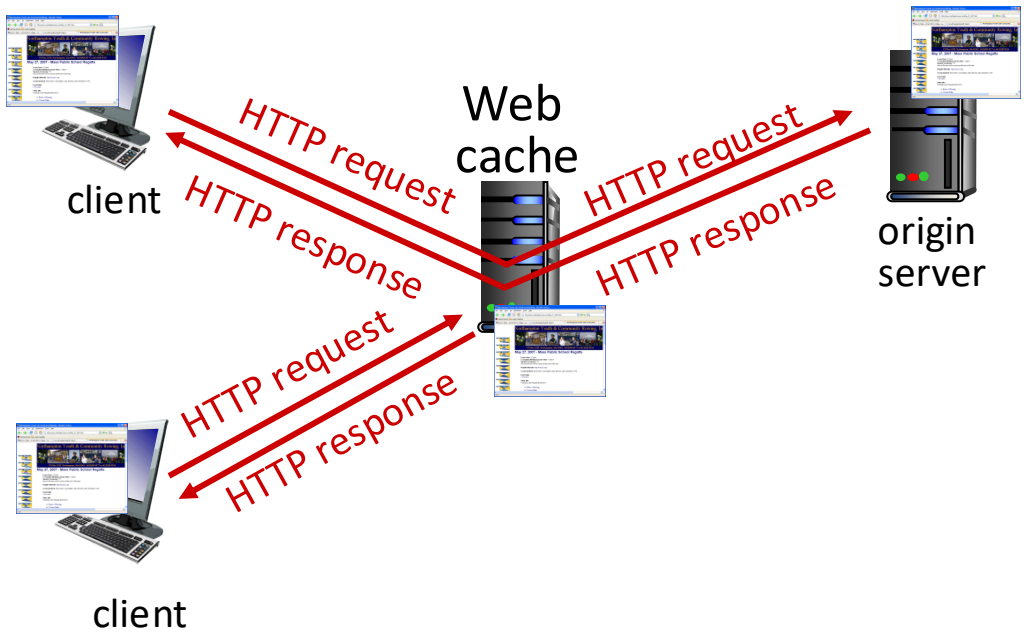
- Persistent connections
- Break objects into frames
- Push objects on a page before they are even requested

What if I didn’t have to request the object from the web server at all?

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Browser caching: Conditional GET

Goal: don't send object if browser has up-to-date cached version

- no object transmission delay (or use of network resources)

- *client:* specify date of browser-cached copy in HTTP request

  If-modified-since: <date>

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

  HTTP/1.0 304 Not Modified