After Midterm

Lecture 1: Application Layer and Http

05506015 Data Communication and Computer Networks

Dr. Rungrat Wiangsripanawan

Outline

- What is the Internet?
- What is a protocol?
- Application Layer Protocol

เนื้อหาใน Slide 95% นำมาจาก Slide บทที่ 1 และ บทที่ 2 ของ หนังสือ
Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

First Question

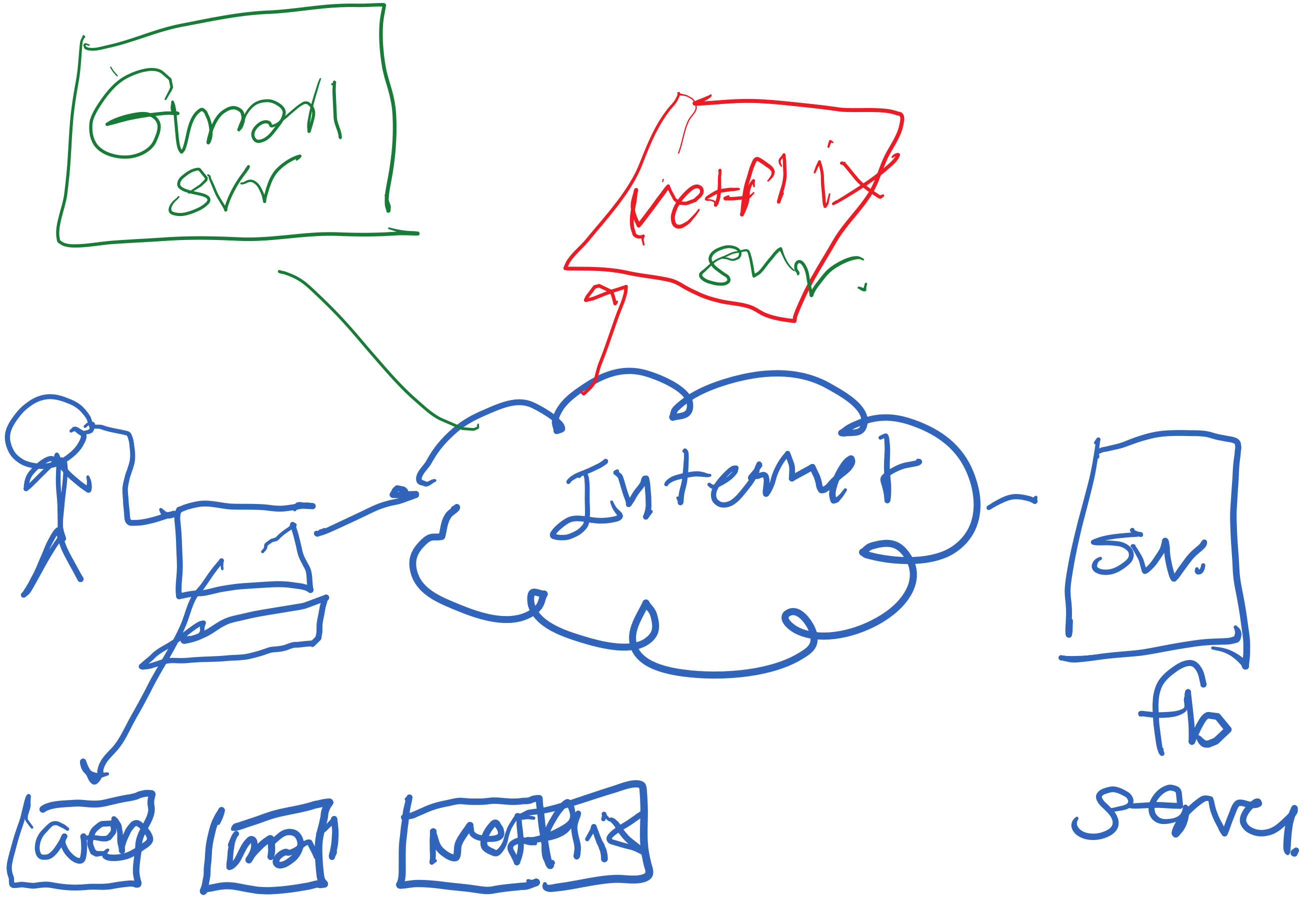
the Internet = WWW??
Discuss
5 mins in Chat

What is the internet?

What is the internet?

an infrastructure that provides services to applications.

- Traditional applications
  - Web/ email
- Messaging Service,
- Netflix Disney
- Internet Messaging



Some network apps

- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

Q: *your* favorites?

Applications

- Network application
- Distributed application
- Application run on End Hosts/Systems
- 2 Architectures
  - Client Server
  - Peer-to-Peer
- Socket API

Summary: What is the Internet ?

- *Infrastructure* that provides services to applications:
  - Web, streaming video, multimedia teleconferencing, email, games, e-commerce,
- provides *programming interface* to distributed applications:
  - "hooks" allowing sending/receiving apps to "connect" to, use Internet transport service
  - provides service options, analogous to postal service

Protocol ในการส่งอีเมลหา อ รุ่งรัตน์

1. ต้องใช้อีเมล์สถาบันเท่านั้น
2. เนื้อความต้องมี รูปแบบดังต่อไปนี้ (มีคำขึ้นต้น มี เนื้อหา และ มีคำลงท้าย) ถ้ามีไม่ครบครูไม่อ่านและไม่ตอบ

Subject: [รหัสวิชา] ขอเรียนสอบถาม/ปรึกษา เรื่อง ....

Content:

เรียน อ รุ่งรัตน์

ดิฉัน/ผม นาย/นางสาว .... รหัสนักศึกษา นักศึกษาชั้นปีที่ .... ในวิชา *[ชื่อวิชา]* กลุ่ม .... ขอเรียนสอบถาม ....*เนื้อความที่สอบถาม*

ขอแสดงความนับถือ

ชื่อ นามสกุล เต็ม

Protocol คืออะไร

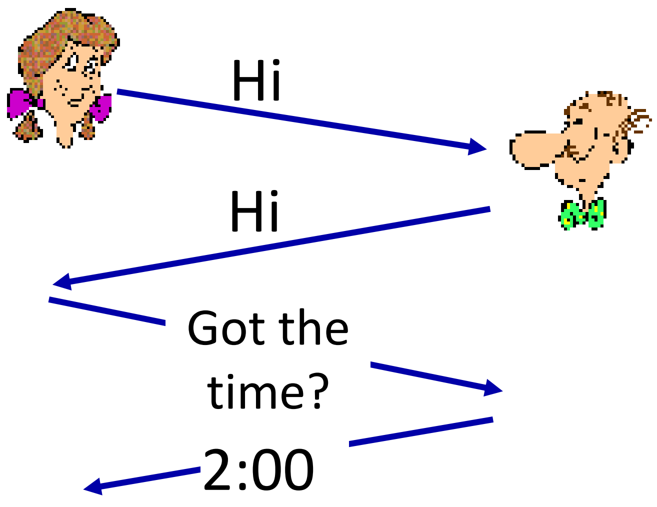
กฎเกณฑ์ในการติดต่อสื่อสาร

Rules for:

... specific messages sent

รูปแบบของข้อความที่ใช้ส่ง

... specific actions taken when message received, or other events เมื่อได้รับข้อความนั้นแล้วจะทำอะไร



Computer Networking: A Top-Down Approach
8th edition n
Jim Kurose, Keith Ross
Pearson, 2020

Application Layer

An application-layer protocol defines:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype, Zoom

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

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

throughput

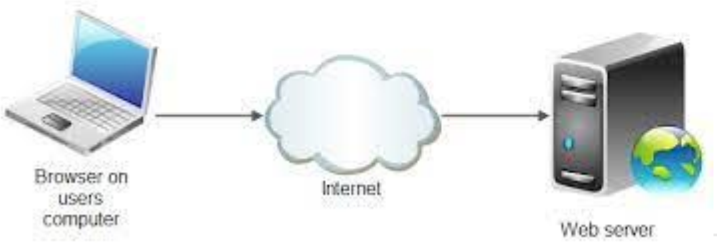
- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

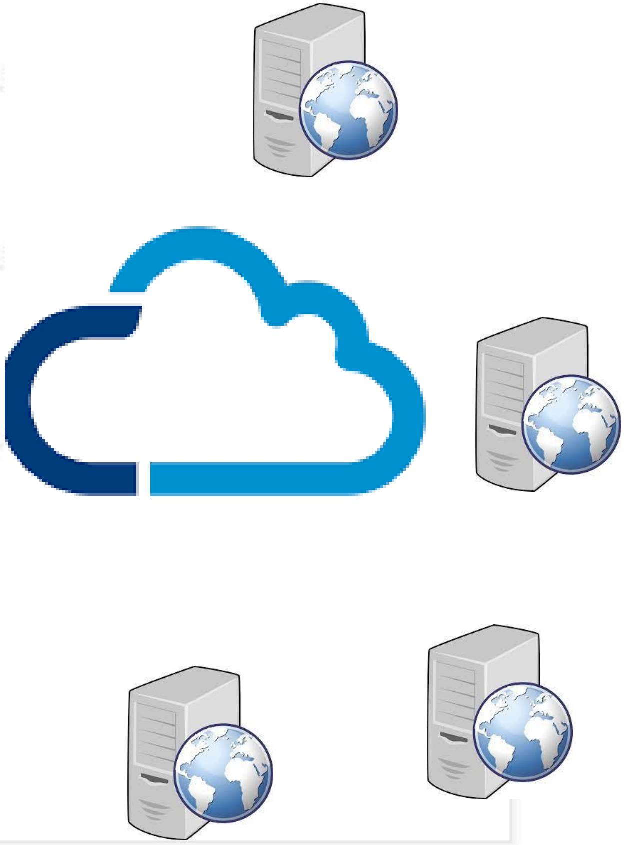
- encryption, data integrity, …

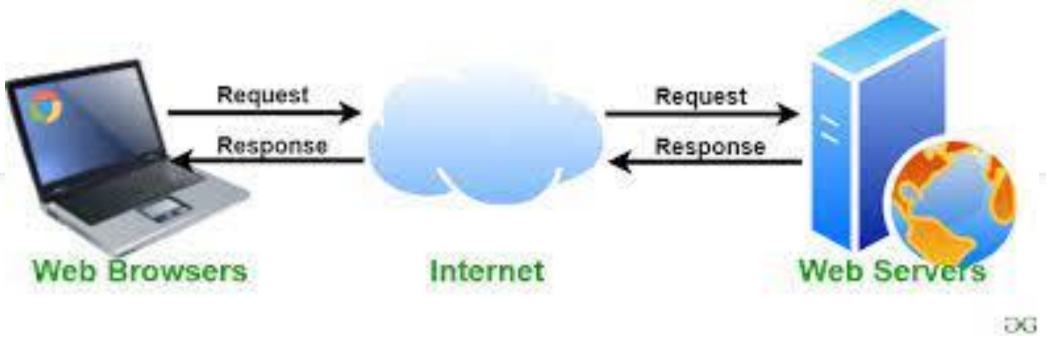
Transport service requirements: common apps

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no



Application Layer protocol : http (web protocol)



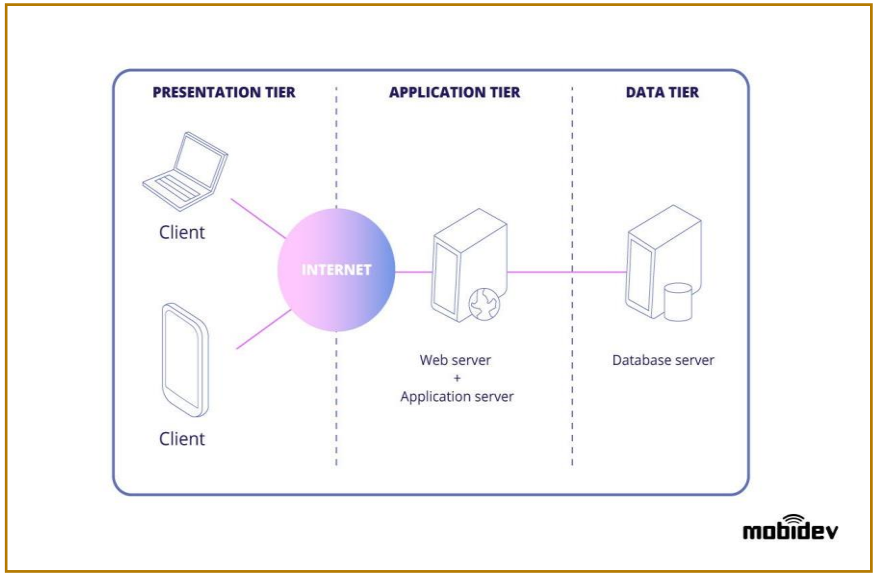


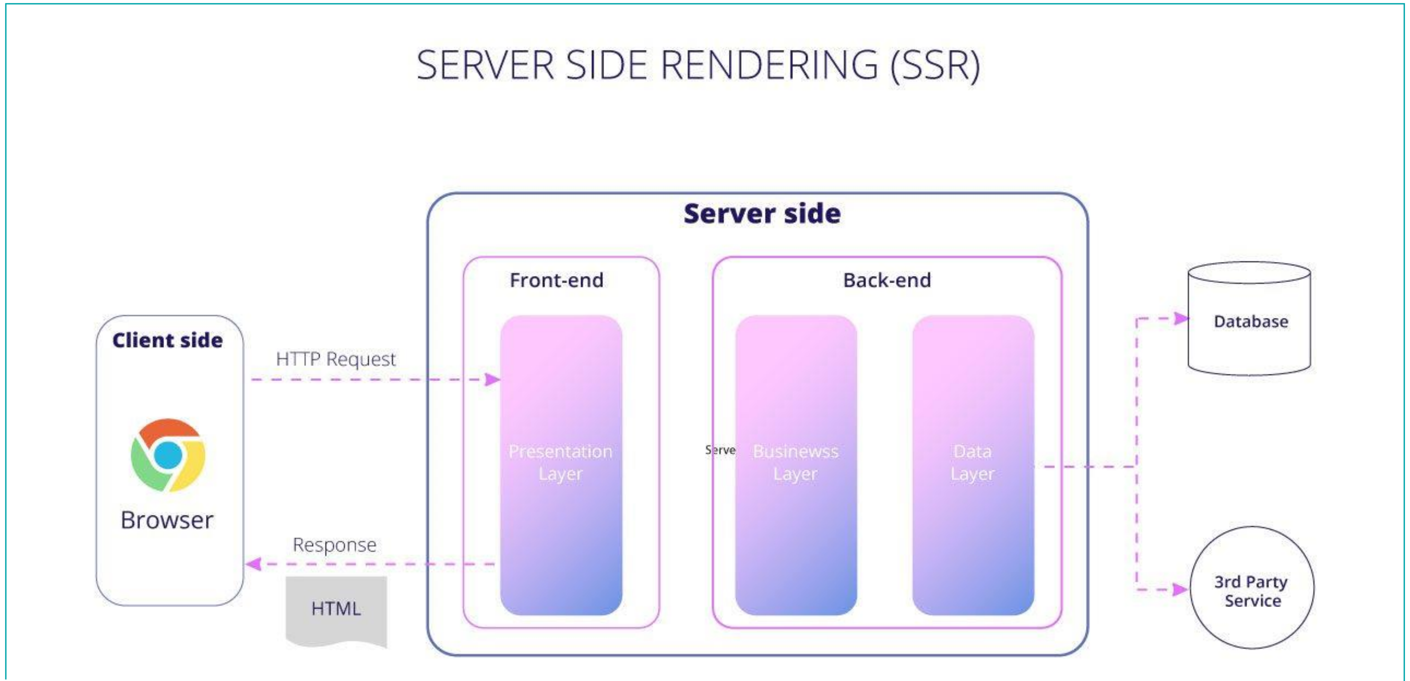
Web and HTTP

First, a quick review…

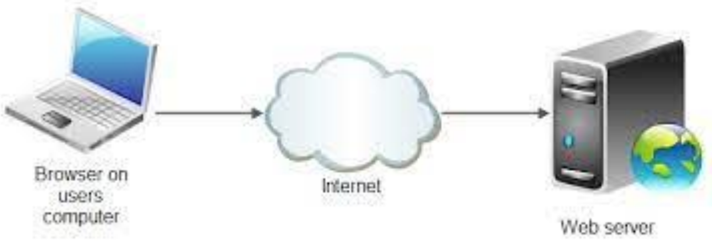
- web page consists of *objects,*
  - each of which can be stored on different Web servers
- object can be
  - HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$





- Client – Browser
- Web Server
- HTTP PRotocol



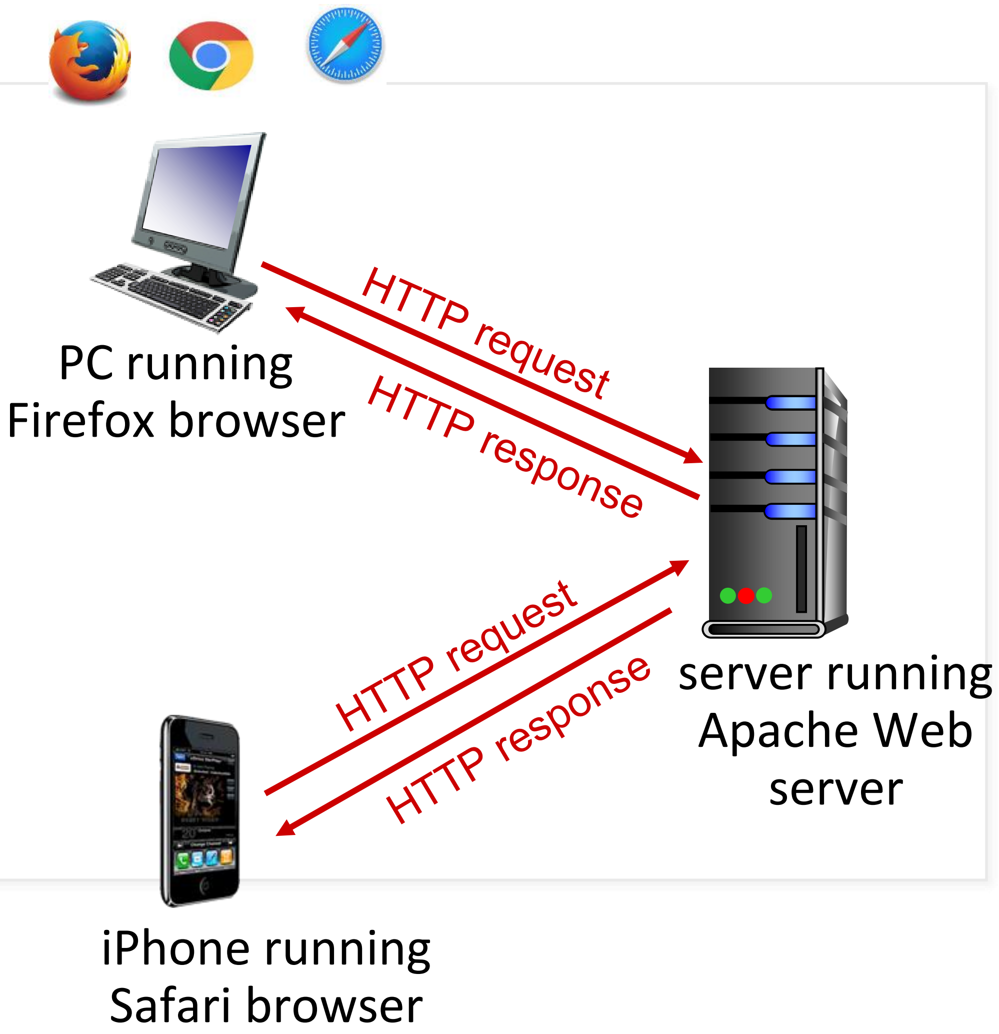
HTTP Versions

Year	Version
[1991	HTTP/0.9]
1996	HTTP/1.0
1997	HTTP/1.1 *
2015	HTTP/2
2020 (draft)	HTTP/3

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, 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

PC running Firefox browser

server running Apache Web server

HTTP overview (continued)

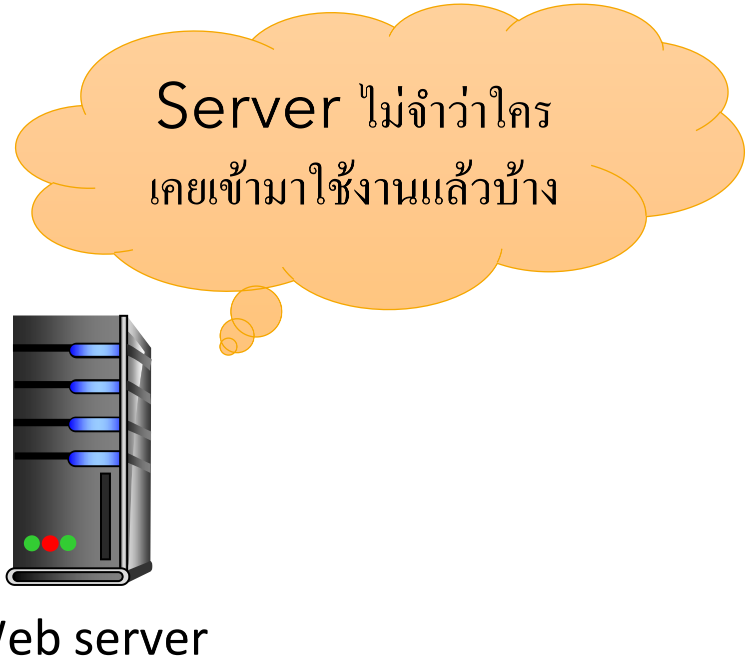
HTTP is "stateless"

- server maintains *no* information about past client requests

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 Request

HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)
  - Request Line / Header lines/ empty line (CR LF)

carriage return character

line-feed character

1. request line (GET, POST, HEAD commands)

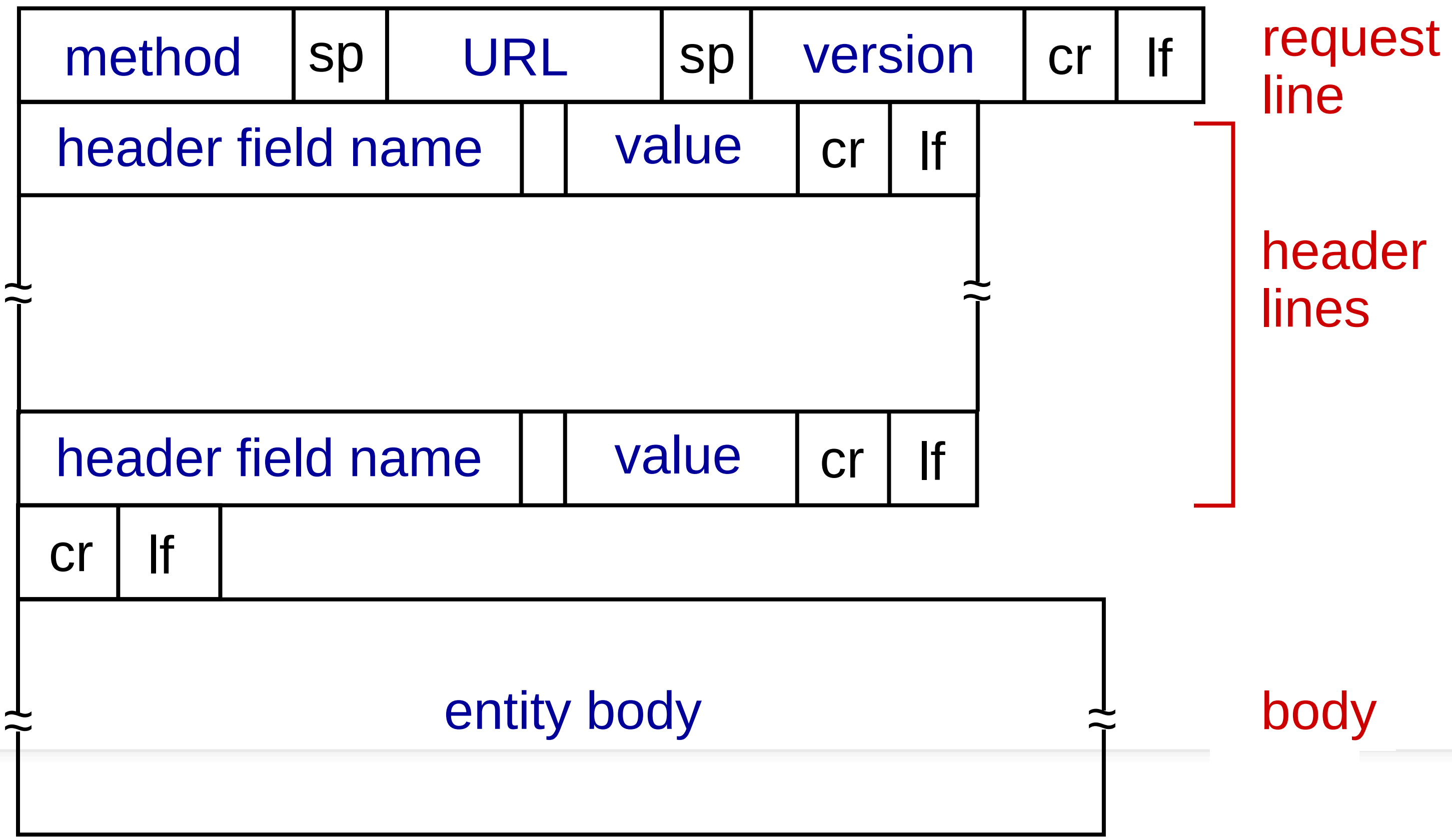
2. header lines

```
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
   10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \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
Connection: keep-alive\r\n
\r\n
```

3. 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/

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

Response syntax[edit]

- A server sends *response messages* to the client, which consist of:[23]
  - a status line, consisting of
    - the protocol version, a space, the response status code, another space, a possibly empty reason phrase, a carriage return, and a line feed
    - (e.g. *HTTP/1.1 200 OK*);
  - **Header lines**
    - zero or more response header fields,
    - each consisting of
      - the case-insensitive field name, a colon, optional leading whitespace, the field value, and optional trailing whitespace (e.g. *Content-Type: text/html*), and ending with a carriage return and a line feed;
  - an empty line, consisting of a carriage return and a line feed;
  - an optional message body.

HTTP response message

status line (protocol status code status phrase) → `HTTP/1.1 200 OK`

Example http response

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>
```

When *Connection: close* is sent, it means that the web server will close the TCP connection immediately after the transfer of this response.

The first digit of the status code defines its class:

- 1XX (informational)
  - The request was received, continuing process.
- 2XX (successful)
  - The request was successfully received, understood, and accepted.
- 3XX (redirection)
  - Further action needs to be taken in order to complete the request.
- 4XX (client error)
  - The request contains bad syntax or cannot be fulfilled.
- 5XX (server error)
  - The server failed to fulfill an apparently valid request.

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

Persistent (http 0.9/ 1.0) Non Persistent (http/1.1 เป็นต้นมา)

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

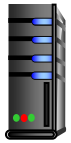
Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

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

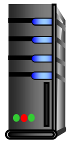
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

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/home.index`**
(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

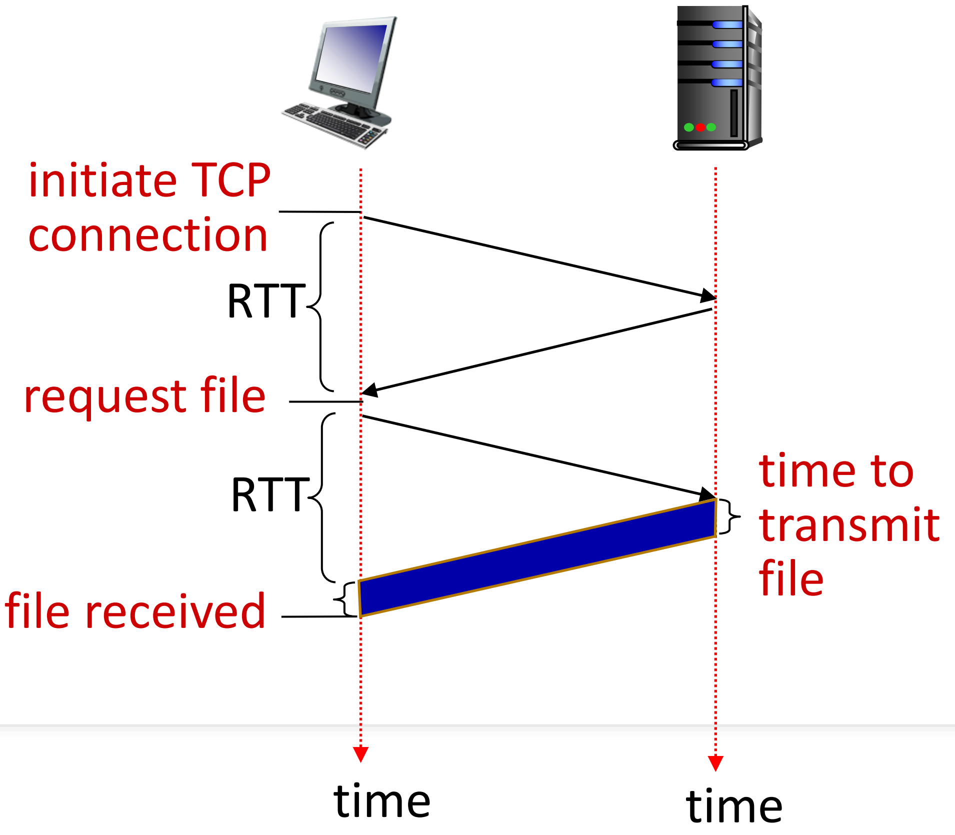
time

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
- obect/file transmission time



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

Cookies

Recall: HTTP GET/response interaction is *stateless*

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

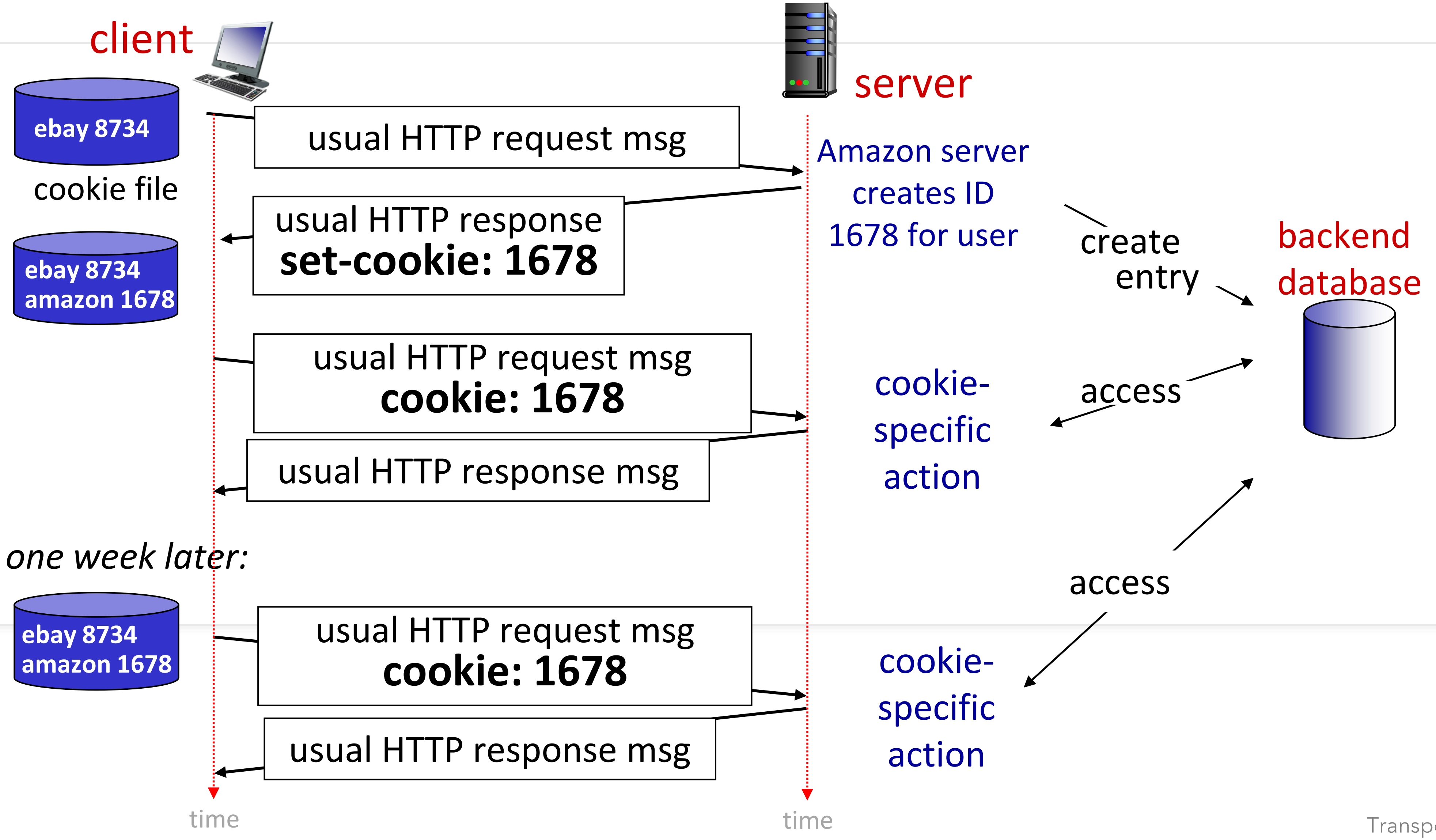
four components:

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

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Challenge: How to keep state?

- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies inHTTP messages carry state

aside

cookies and privacy:

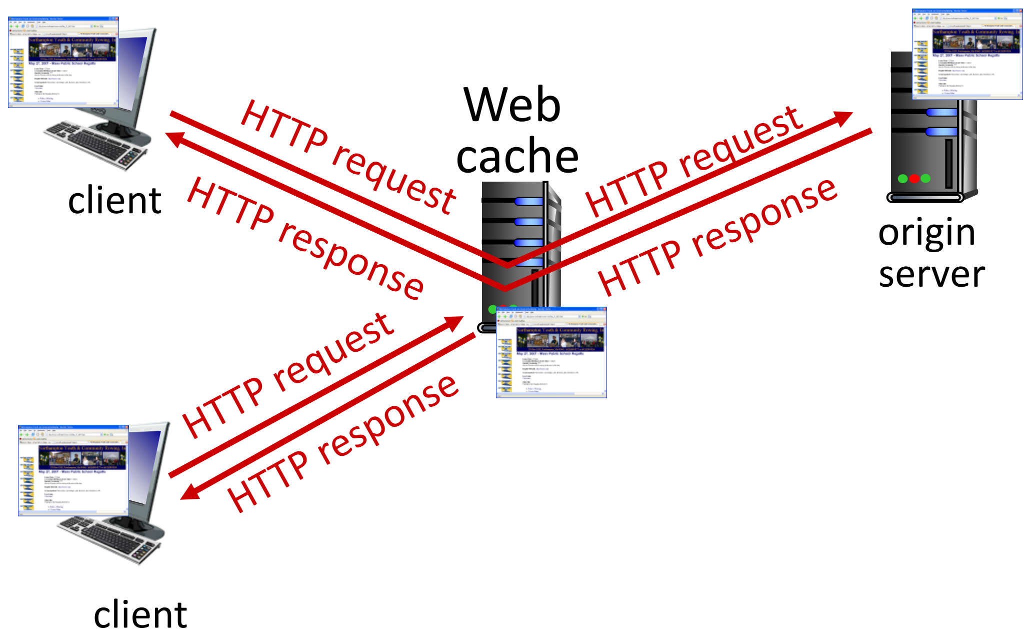
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Web cache

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



Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object’s allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution’s access link
- Internet is dense with caches
  - enables “poor” content providers to more effectively deliver content

Caching example

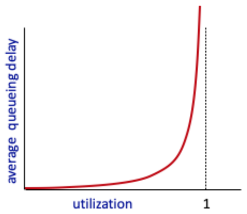
Scenario:

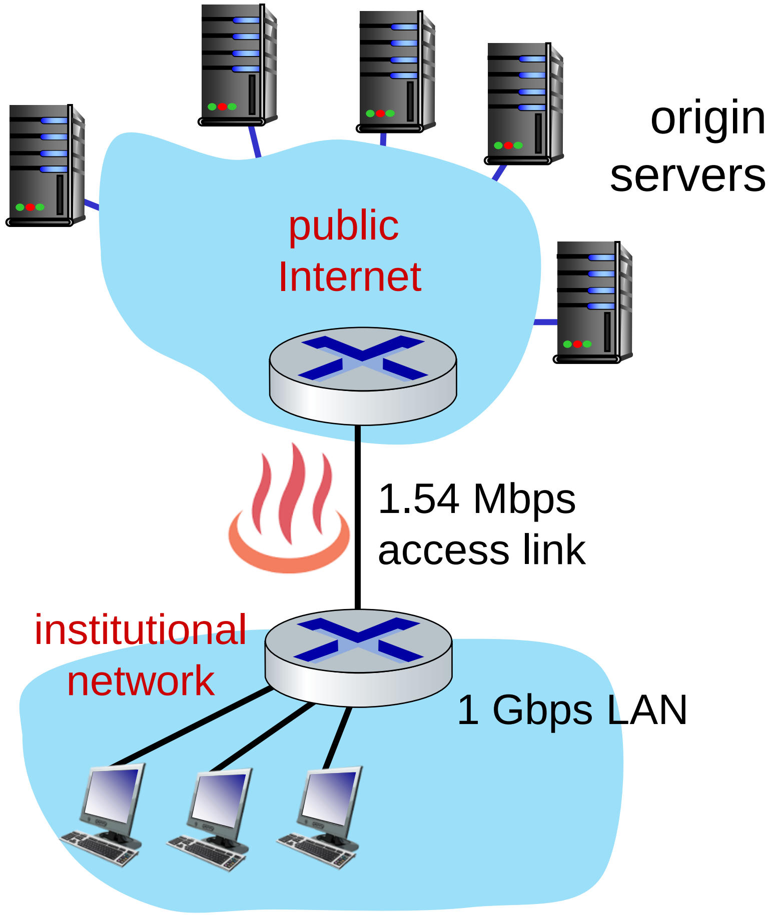
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs

problem: large queueing delays at high utilization!





Option 1: buy a faster access link

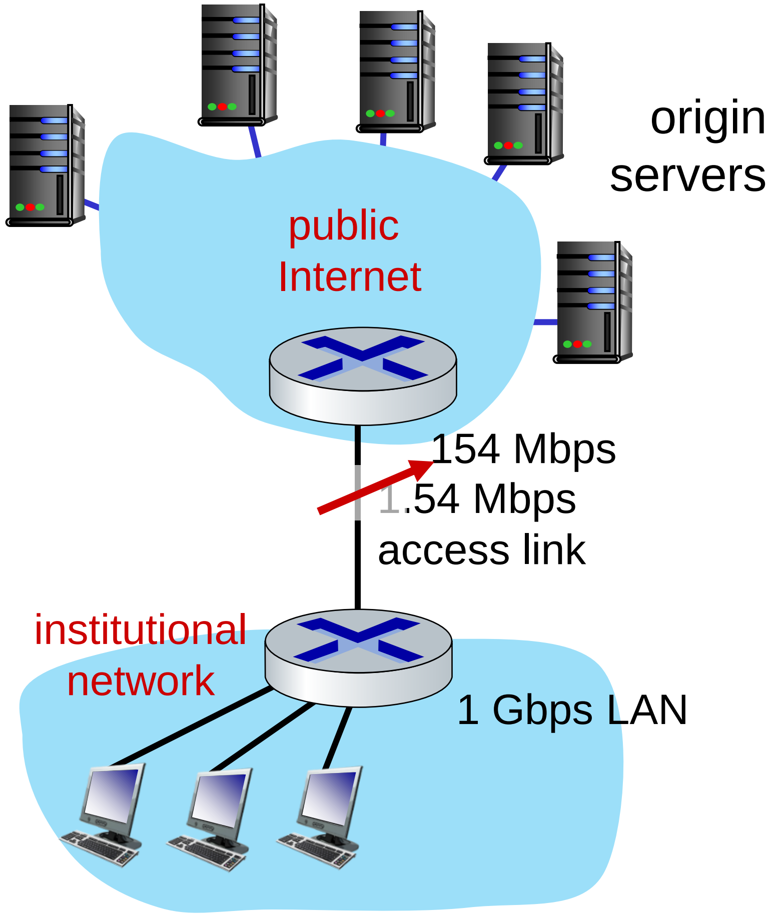
Scenario:

- access link rate: ~~1.54~~ 154 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ → .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs

Cost: faster access link (expensive!)



Option 2: install a web cache

Scenario:

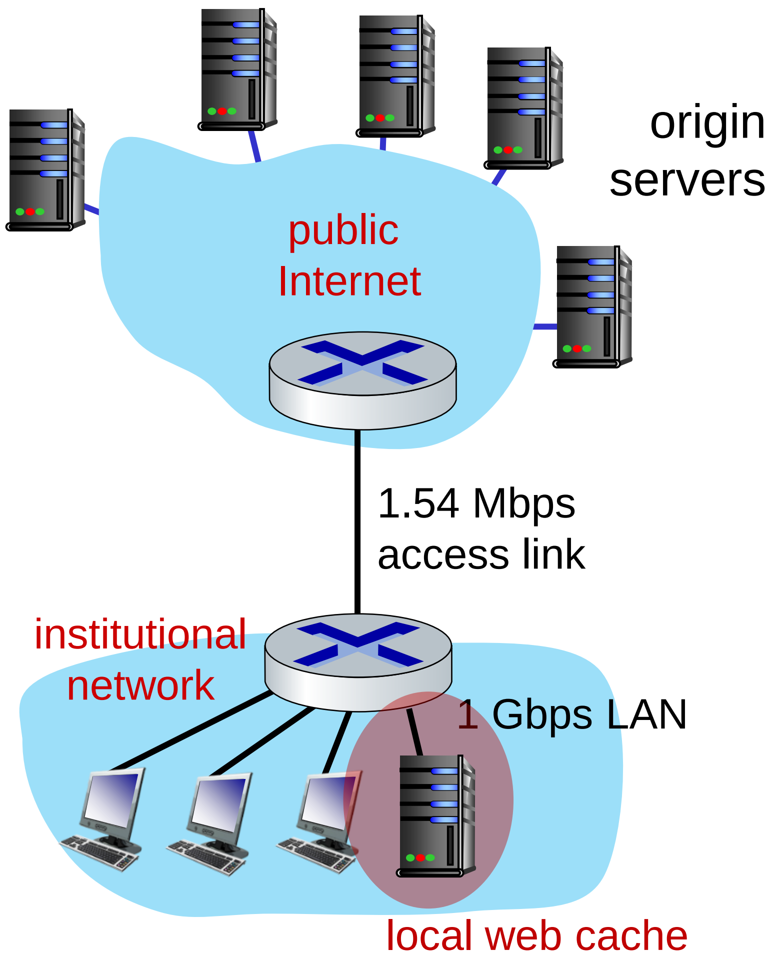
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

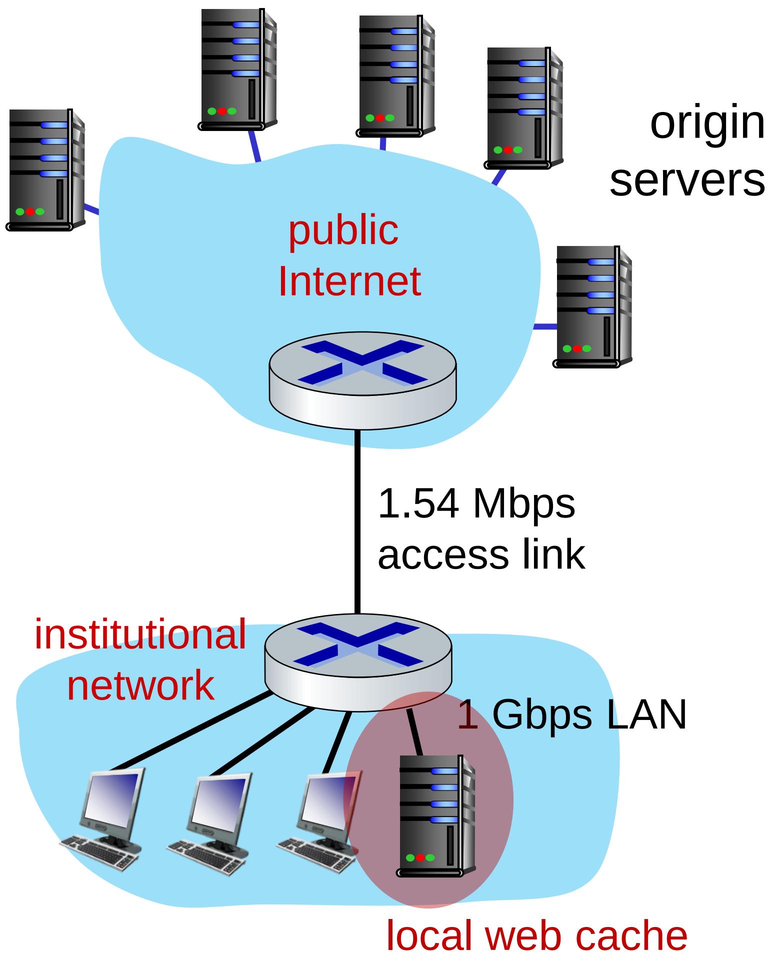
How to compute link utilization, delay?



Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link
    = 0.6 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- average end-end delay:
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



lower average end-end delay than with 154 Mbps link (and cheaper too!)

Conditional GET

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

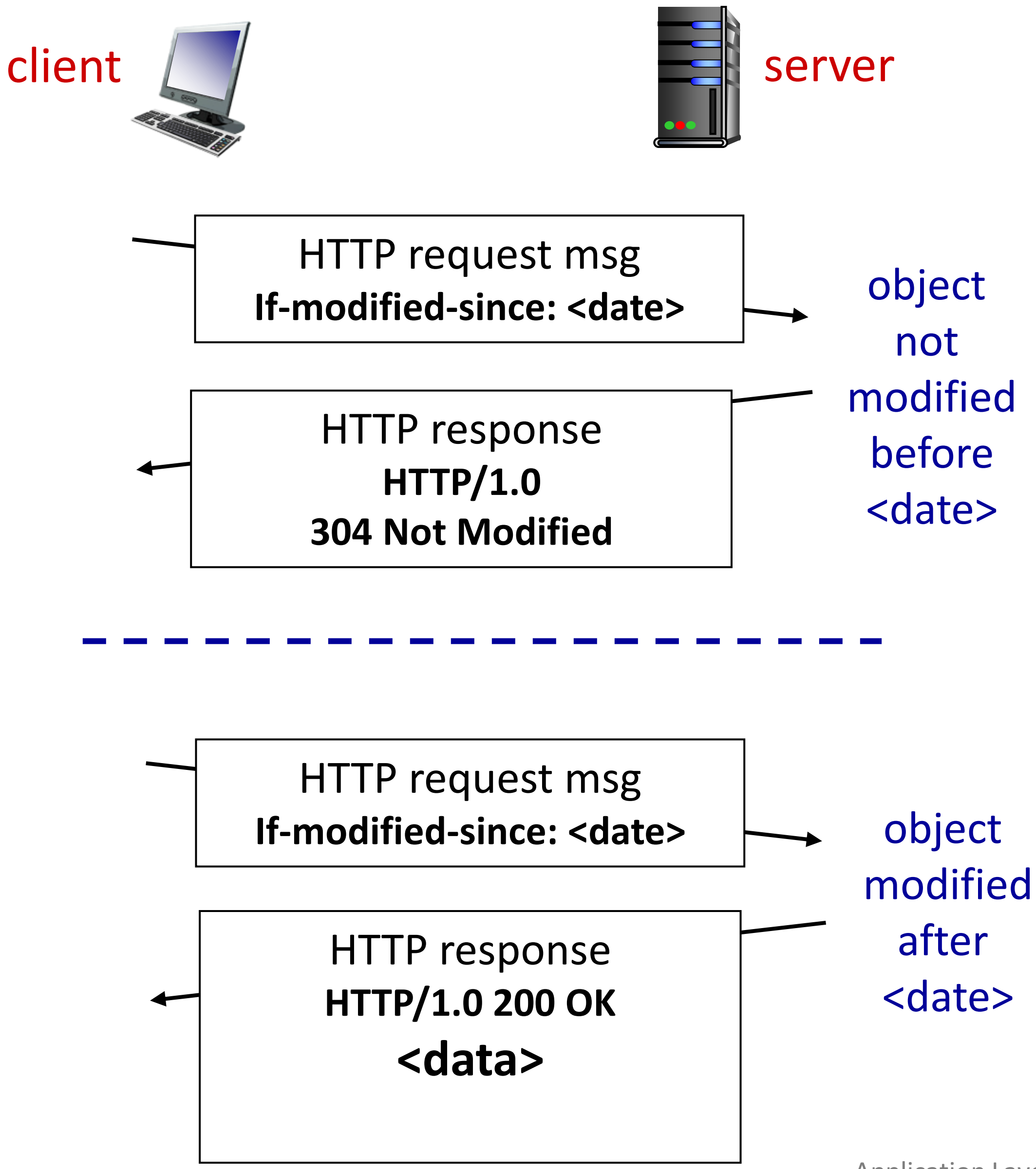
- no object transmission delay (or use of network resources)

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

  If-modified-since: <date>

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

  HTTP/1.0 304 Not Modified



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

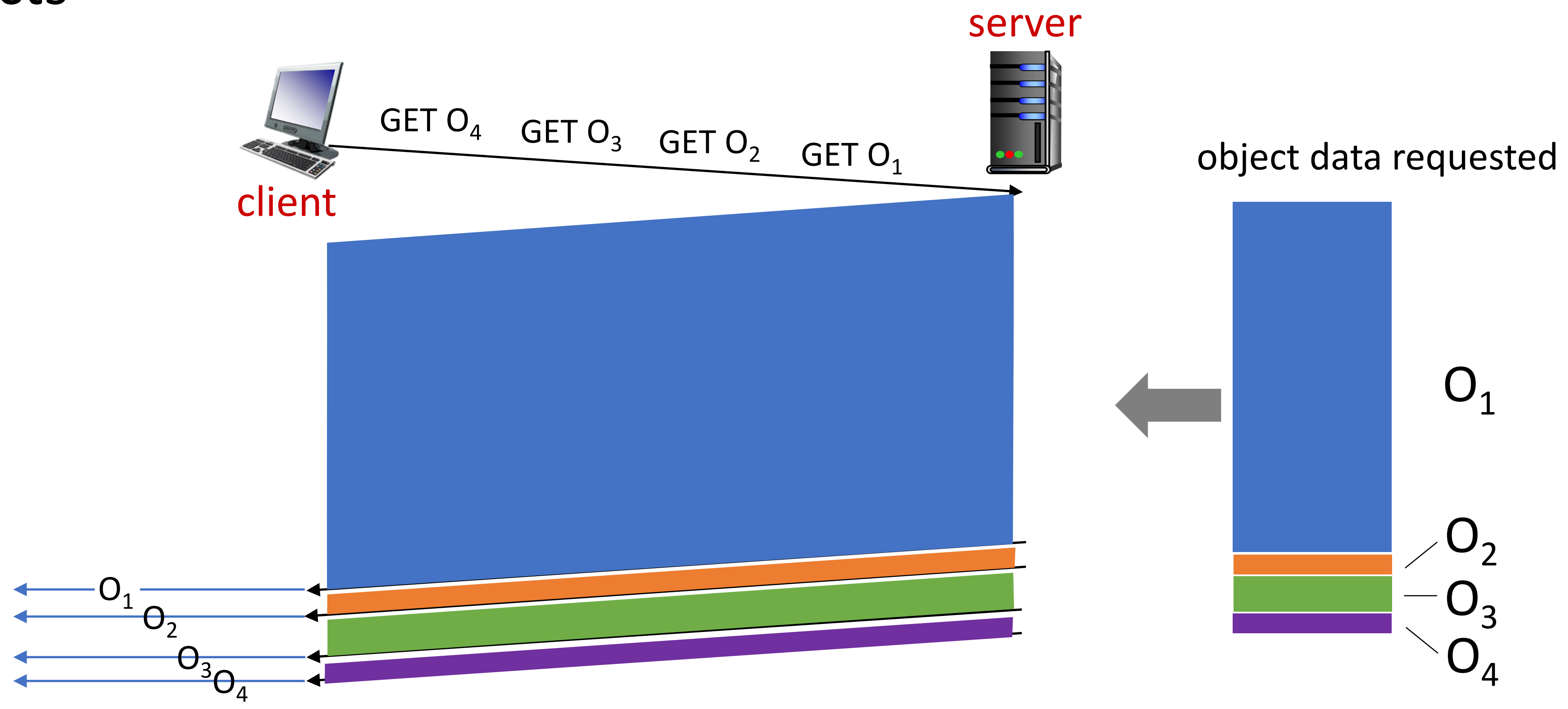
HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- 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

Head of Line (HOL) Blocking

HTTP/2: mitigating HOL blocking

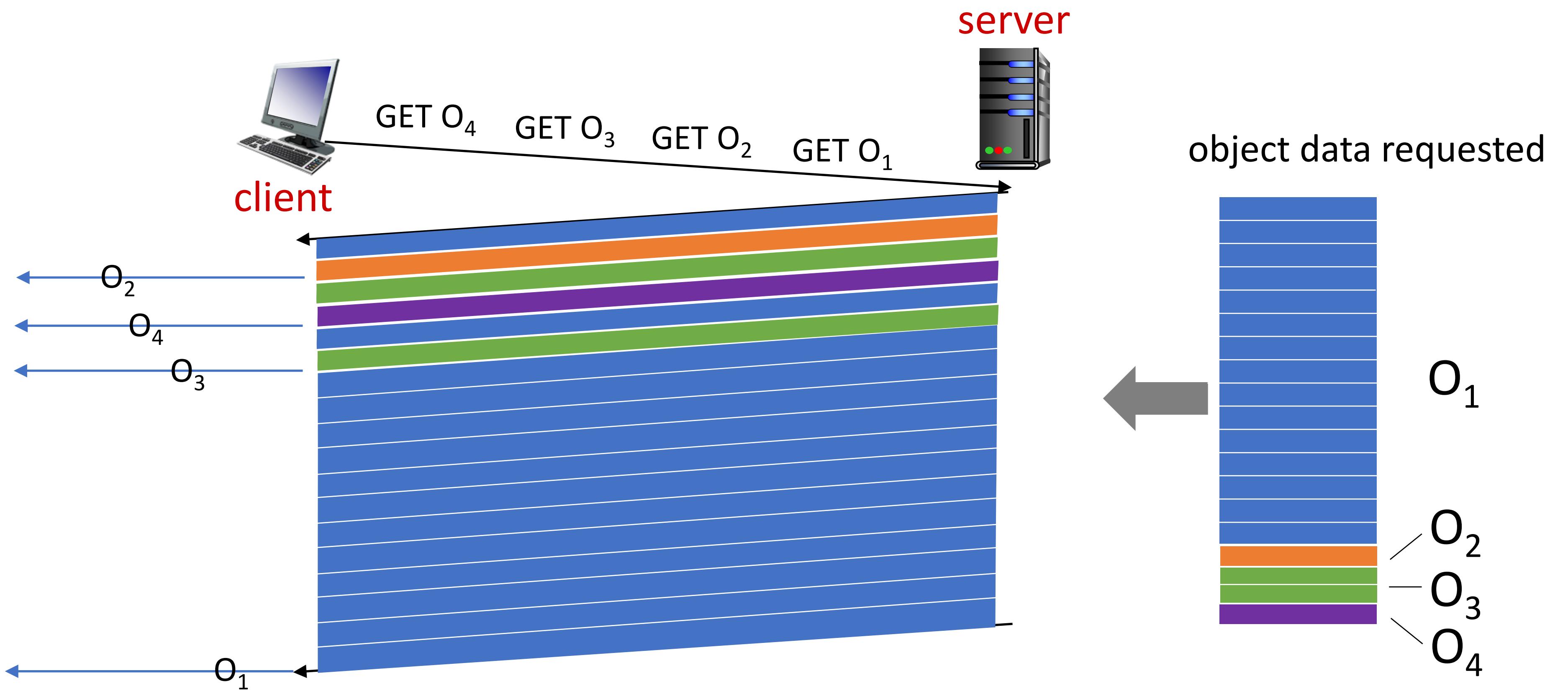
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



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/2 to HTTP/3

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security, per object error- and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer

HW. Q1: Postal Service - กลุ่มละ 5 คน การส่ง เขียนส่ง 1 หน้ากระดาษ A4 ก่อนเข้าเรียน ส่งทาง Go.edu

1. ถ้านักศึกษาต้องการเขียนจดหมายถึง Lisa Blackpink หรือ CR7 นักศึกษาต้องทำอย่างไรบ้าง จดหมายถึงจะไปถึง Lisa หรือ โด้
2. การบริการของไปรษณีย์ช่วยให้จดหมายไปถึงผู้รับได้อย่างไร
3. นักศึกษาจะมั่นใจได้อย่างไรว่า จดหมายไปถึง Lisa หรือ โด้แน่ๆ ไม่สูญหายระหว่างทาง

HW. Q2: ทำการลงโปรแกรม Wireshark

1. ทำการลงโปรแกรม Wireshark บนเครื่องของนักศึกษาคนใดคนหนึ่งในกลุ่ม
2. เมื่อลงสำเร็จ ให้ถ่ายรูป นักศึกษา กับ เครื่องคอมพิวเตอร์ ที่เปิดหน้าจอโปรแกรม Wireshark เรียบร้อยแล้ว

References:

- [1] CSCI262 Lecture Notes by Dr. Luke McEvan, University of Wollongong Australia.
- [2] Computer Security: Principles and Practice, W. Stalling and L. Brown, 1st edition, Pearson Education, 2008.
- [3] Computer Security, D. Gollman. 2nd edition, John Wiley & Sons, 2006.
- [4] Wikipedia.org