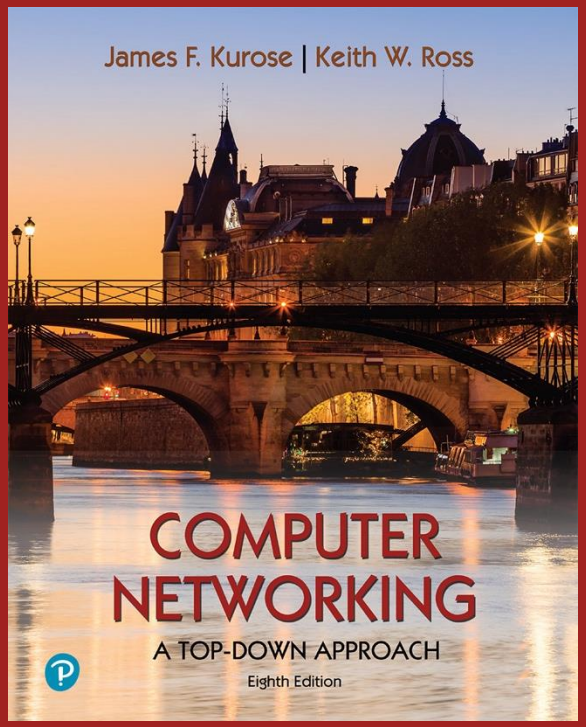
Communication Networks
W. Tavernier



Chapter 2
Application Layer

Computer Networking: A Top-Down Approach
8th Edition, 2020, Pearson,
James F. Kurose, Keith W. Ross

Chapter 2 outline

2.1 Principles of network applications

2.2 Web and HTTP

2.3 Electronic mail

- SMTP, POP3, IMAP

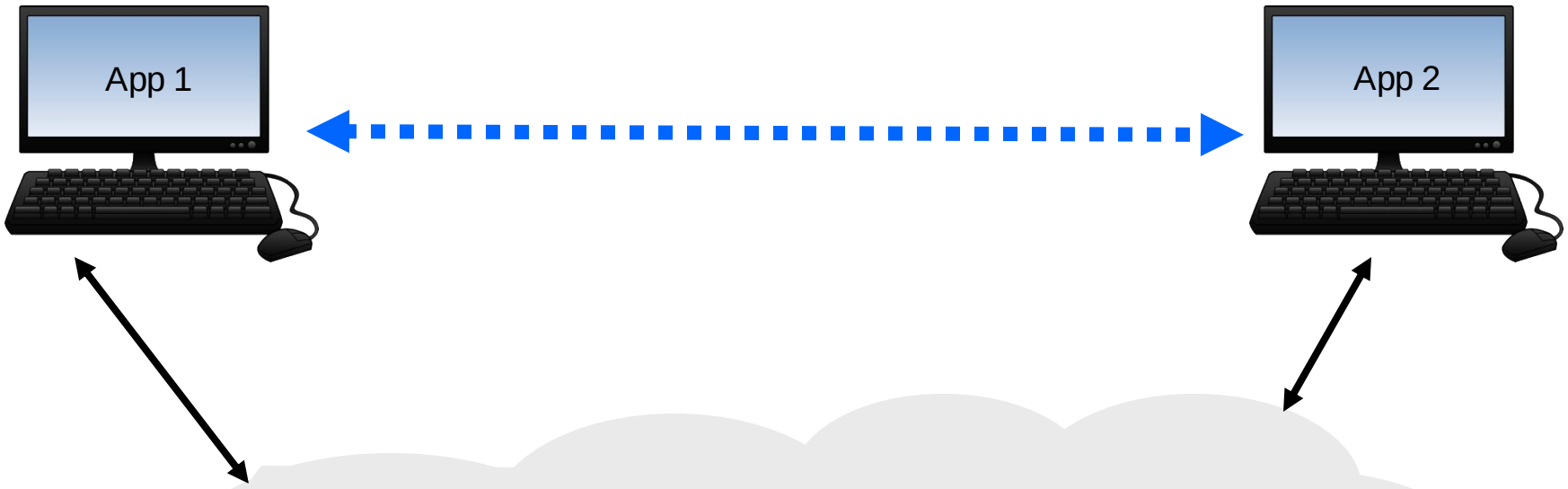
2.4 DNS

2.5 P2P applications

[**2.6** video streaming and content distribution networks]

2.7 Socket programming with UDP and TCP

Focus : Application Layer



Q: How to communicate among remote processes?

Network Applications

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

YouTube

skype™

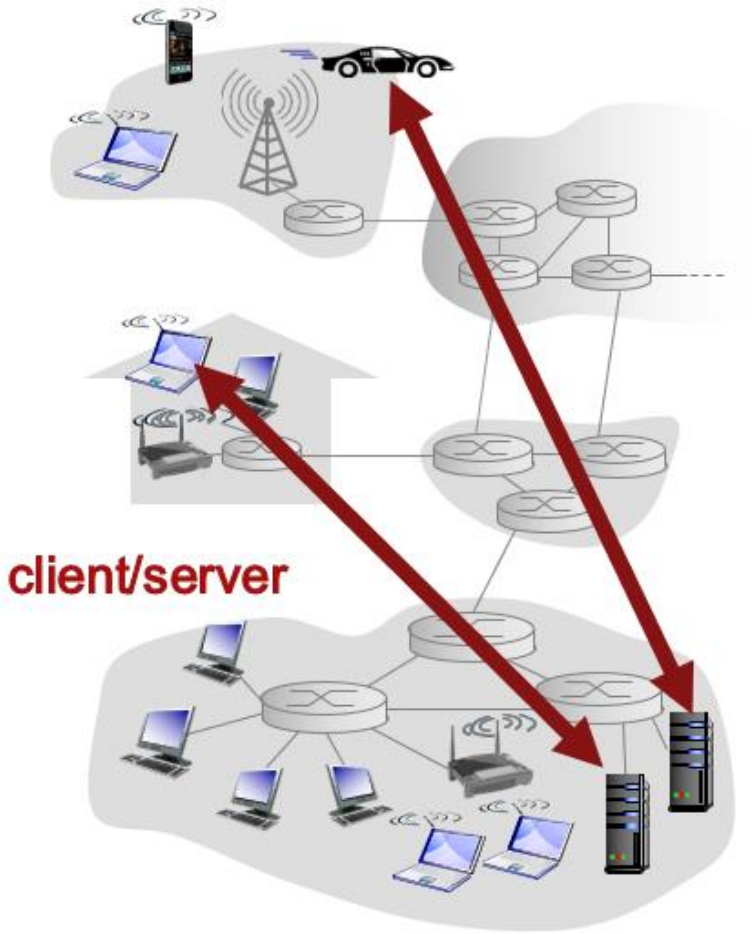
µTorrent®

Q: Which application architectures can we identify?

Client-Server Architecture

server:

- always-on host
- permanent IP address
- data centers for scaling



clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Application Client - Server

CLIENT : "*active open*"

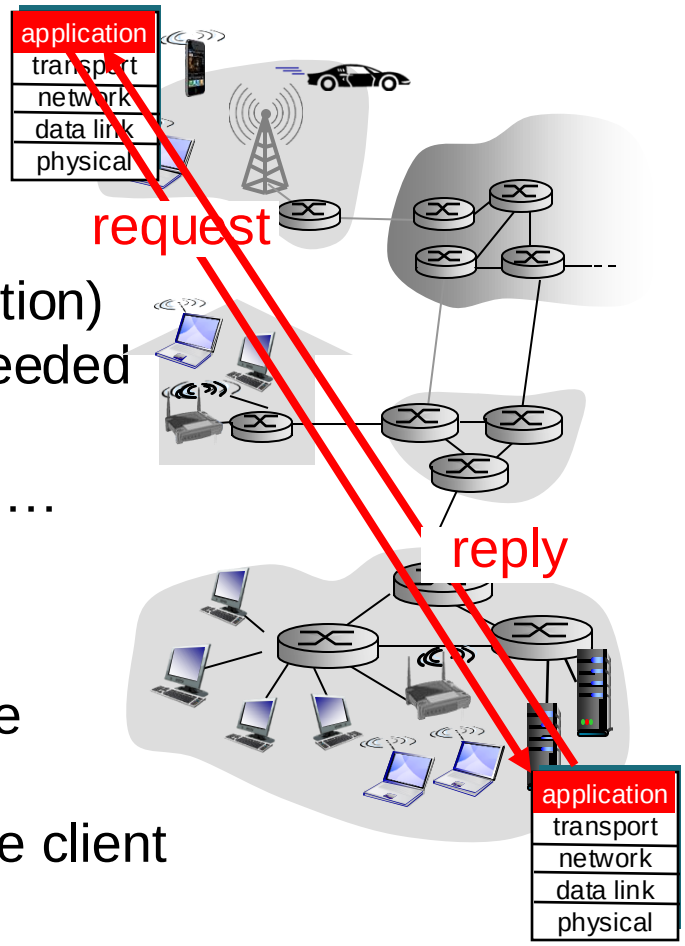
- invoked directly by user
- local on user's personal computer
- actively initiates contact with server
- one session at a time (per (instance of) application)
- access multiple application(s) (instances) as needed
- simple hardware and software

e.g. : Thunderbird, Outlook, Chrome, Firefox, Safari, …

SERVER : "*passive open*"

- special purpose program for one application
- can handle multiple remote clients at same time
- runs on a shared computer
- waits passively for contact from arbitrary remote client
- publicly known address, port
- powerful hardware and sophisticated operating system
- server program also called daemon (e.g. FTP daemon)

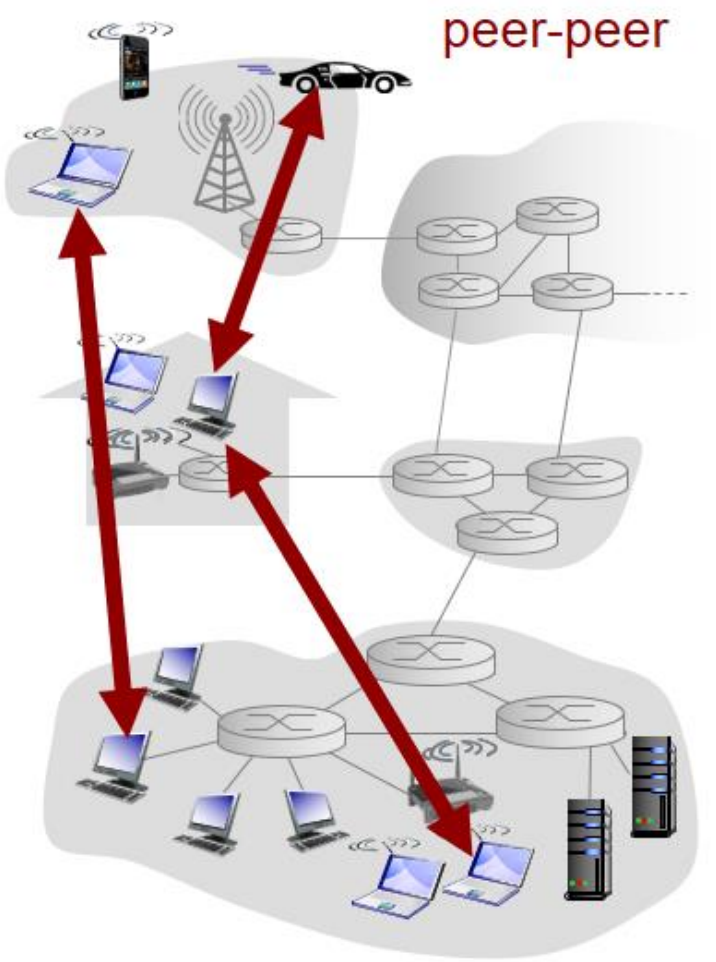
e.g.: Apache, IIS, Sendmail, Postfix, …



Information can flow in both directions between client and server
Some application programs act as client and server

P2P Architecture

- **no** always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - **self scalability** – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Hybrid of client-server and P2P

- Skype
  - voice-over-IP P2P application
  - centralized server: finding address of remote party
  - client-client connection: direct (not through server)
- Instant messaging
  - chatting between two users is P2P
  - centralized service: client presence detection/location
    - user registers its IP address with central server when it comes online
    - user contacts central server to find IP addresses of contacts

Focus : Application Layer

Q: What do we need for communication between two remote applications?

What is needed for application-layer communication?

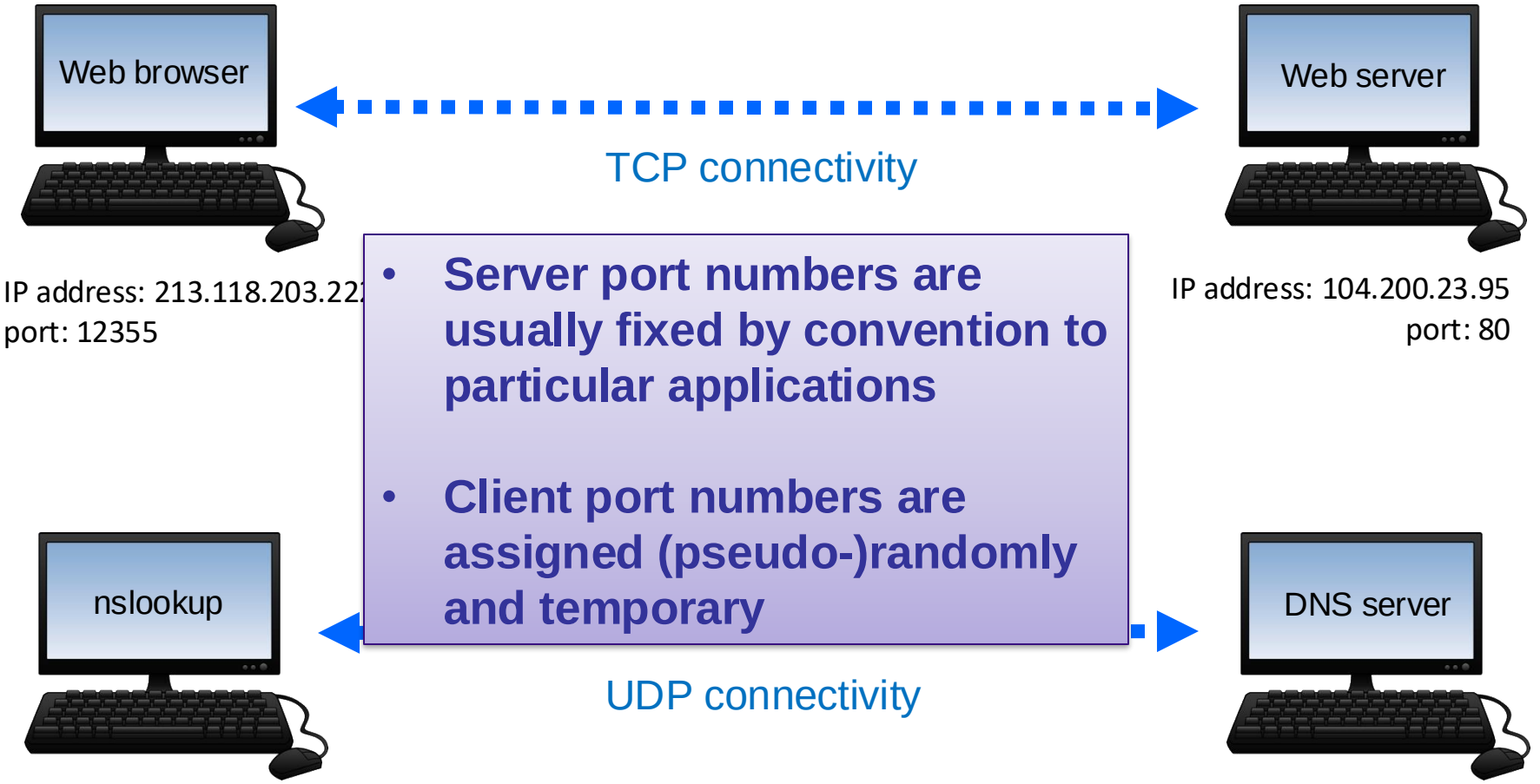
1. Addressing of remote processes

- Nodes (i.e., interface) on the Internet are addressed using IP addresses (32-bit IPv4 address or 128-bit IPv6 address)
- Applications (i.e., OS processes) on the considered nodes are addressed using a port number (16-bit value, i.e. 0-65535)

2. Type of connectivity

- Provided by the lower layer = transport layer protocol
- Two (most used) types:
  - Reliable byte stream service = TCP
  - Best-effort datagram service = UDP

Examples: web browsing & DNS



Common server port mappings for applications

Application	Usual server port	Transport Layer Protocol
FTP	20,21	TCP
SSH	22	TCP
Telnet	23	TCP
SMTP (email transfer server)	25	TCP
DNS	53	UDP/TCP
HTTP (web)	80	TCP
POP (email server)	109, 110	TCP

Note that these are assigned by convention (reserved as per RFC 1700).
Port numbers > 1024 can be freely used (e.g., web server at port 443)

```
$ sudo netstat -plnt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      3686/mysqld
tcp        0      0 :::443                  :::*                    LISTEN      2218/httpd
tcp        0      0 :::80                   :::*                    LISTEN      2218/httpd
tcp        0      0 :::22                   :::*                    LISTEN      1051/sshd
```

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

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

timing

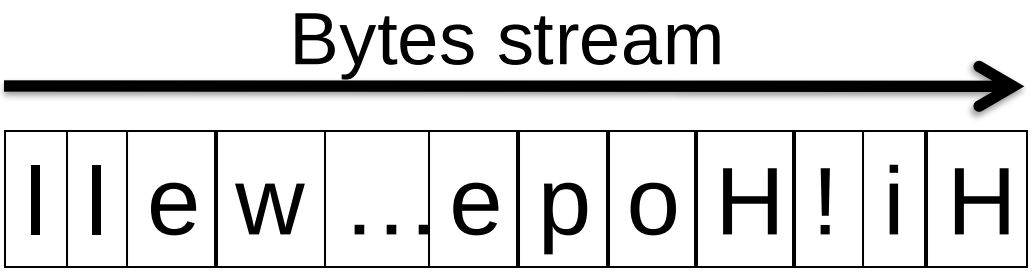
- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

security

- encryption, data integrity, …

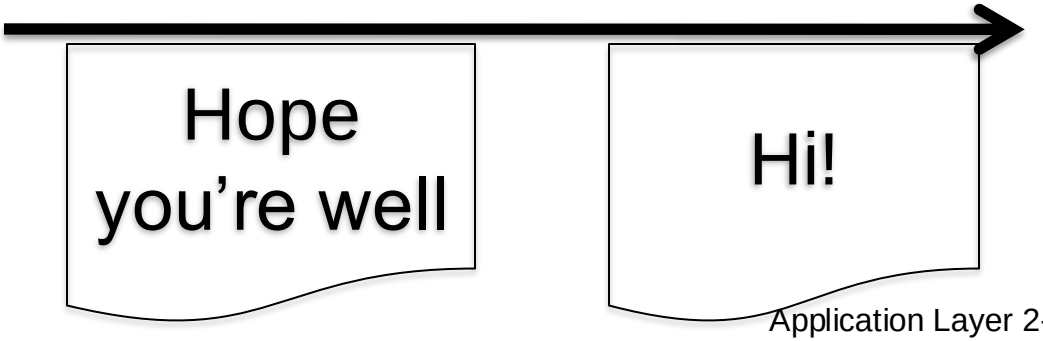
Type of connectivity (transport layer)

- TCP = reliable byte stream service
  - Connection setup before communication (3-way handshake)
  - Ensures every byte has arrived in order (acknowledgements)

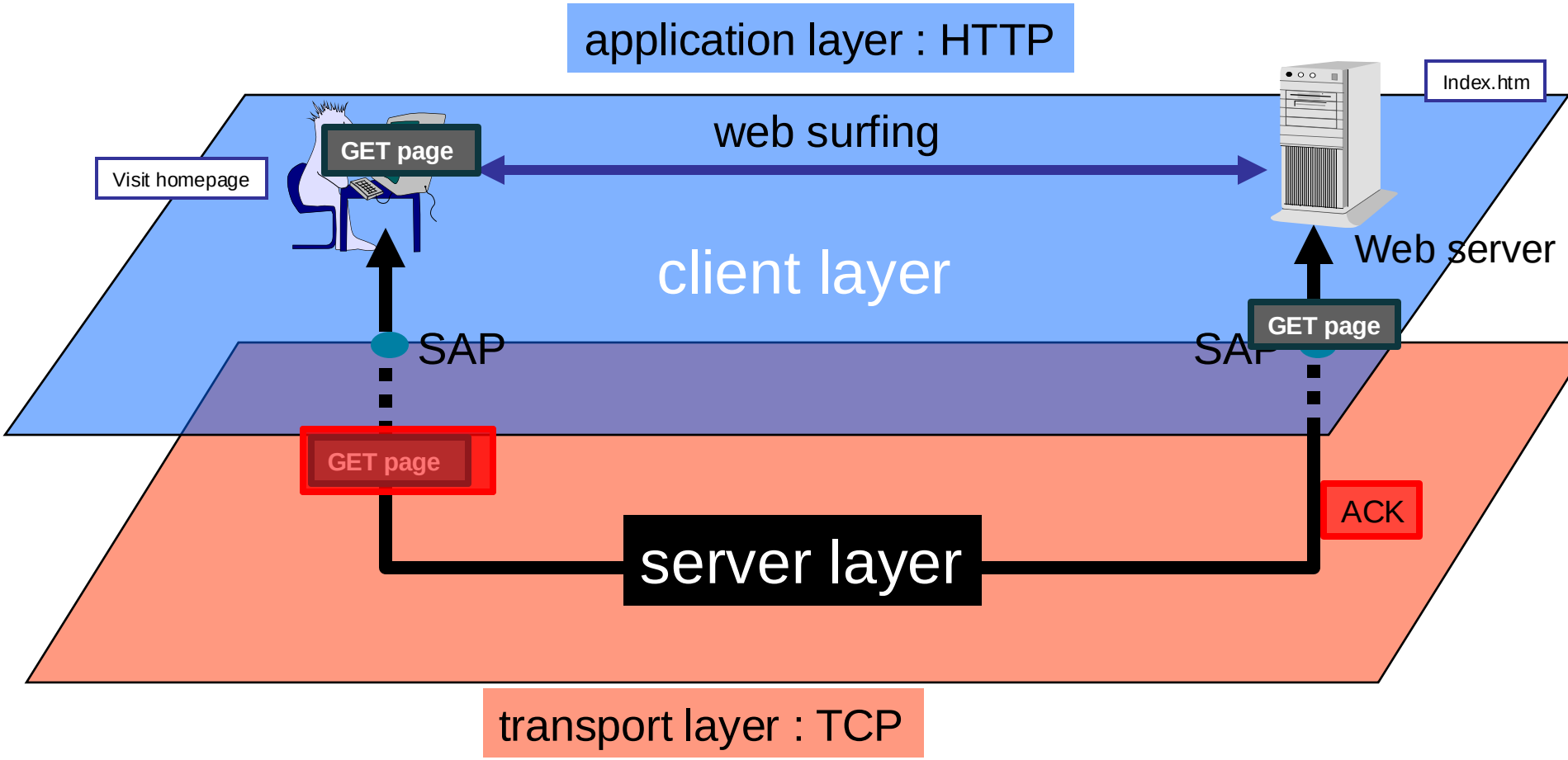


When sending "Hi!" and "Hope you're well"

- UDP = best-effort datagram service
  - UDP treats them as separate messages (of max 65507 bytes)
  - Application itself must handle lost messages



Application/Transport Layer Interaction

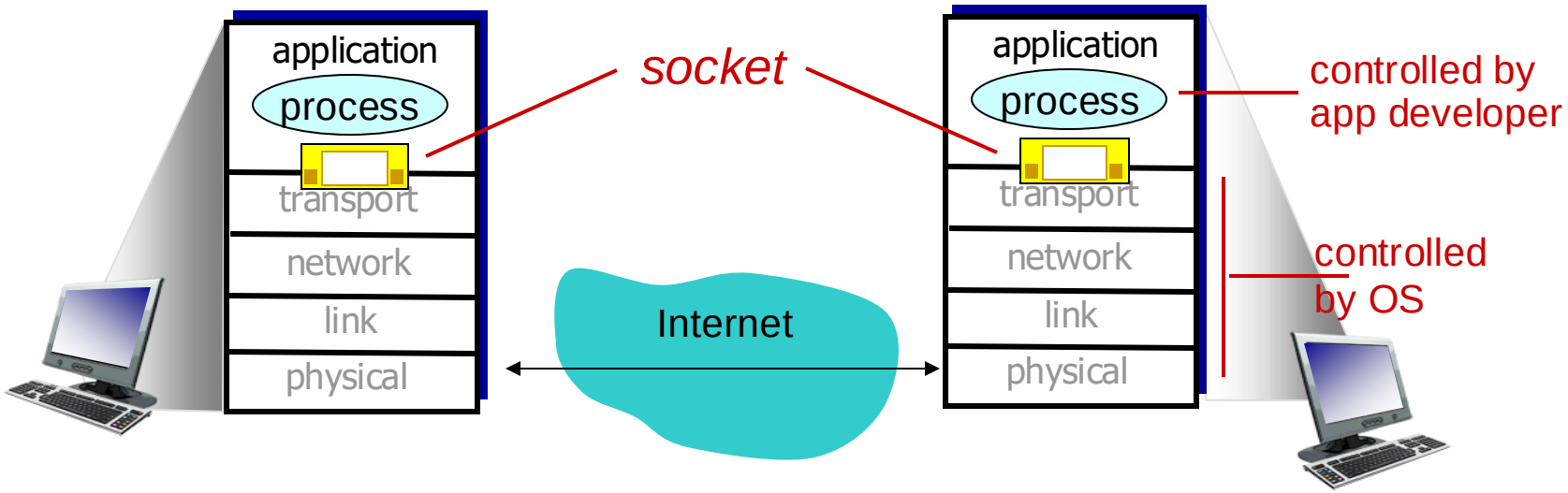


SAP : Service Access point

Remark : client - server

Network programming using sockets

- Network communication from one process to another remote process relies on **Operating System Calls for network programming**
- Most common API is based on **Berkeley Sockets**:
  - Socket = communication endpoint, referring to the “door” between the application process and the end-to-end transport protocol



Socket Example (in Python 3)

TCP Hello Server

```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.bind(('127.0.0.1', 9999))
s.listen(5)

while True:
    c, addr = s.accept()
    g = bytes("Hello %s" % a[0])
    c.send(bytes(g,'utf-8')
    c.close()
```

TCP Hello Client

```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(('127.0.0.1', 9999))

s.send(b'Hello, world')
data = s.recv(1024)

s.close()

print 'Received', data
```

```
# start server first

$ python tcp-server.py
```

```
# next start client

$ python tcp-client.py
Received b'Hello 127.0.0.1\n'
```

Clarification of the socket primitives in notes

Applications and application-layer protocols

- Network Application : communicating, distributed processes (e.g. Web, e-mail, P2P file sharing, …)
  - Applications are responsible for WHAT is sent between the processes

Q: How to enable different user interfaces for the same application (e.g., Chrome vs. Firefox, Outlook vs. Thunderbird)?

- Applications use an **application-layer protocol** to implement these services
- The **application layer** is the collection of these protocols (FTP, SMTP, POP, IMAP, HTTP, DNS, SNMP, etc.)
  - **Example**: HTTP protocol: GET and POST messages

App-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:

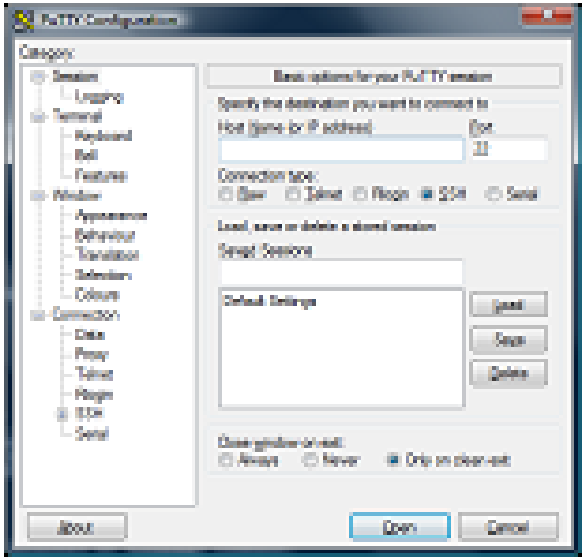
- defined in RFCs
- allows for interoperability
- e.g. HTTP, SMTP

proprietary protocols:

- e.g. Skype, BitTorrent

Telnet client

- You can use a **`telnet`** client (*software*) to play with the protocols: HTTP, SMTP and others.
- The **`telnet`** client is a 'generic' TCP client:
  - Sends whatever you type to the TCP socket.
  - Prints whatever comes back through the TCP socket
  - Useful for testing TCP servers (ASCII based protocols).



Putty (Windows)

telnet command line (Linux)

Investigate simple protocols using Telnet

Some application-layer protocols which can be tested with the telnet client:

- ECHO (port 7)
- DAYTIME (port 13)
- TELNET (port 23)
- HTTP (port 80)
- …

Ask creation of a socket to 157.193.40.10 at port 13

Daytime example

```
$ telnet 157.193.40.10 13
Trying 157.193.40.10...
Connected to eduserv2.ugent.be.
Mon Oct 2 14:27:43 1998
Connection closed by foreign host.
```

Echo example

```
$ telnet 157.193.40.10 7
Trying 157.193.40.10...
Connected to eduserv2.ugent.be.
Hallo
Hallo
^C
```

We type Hallo and see it on the display (Telnet operation)
We receive it a second time on the display (Echo operation)
^C will close the connection

Telnet protocol example

```
$ telnet 157.193.40.10 23
Trying 157.193.40.10...
Connected to eduserv2.ugent.be.
login:wtaverni
passwd:****
Last login: Mon 23 17:12:23 CET
2019 from 213.118.203.222 on pts/15
> ls
test.txt
```

Chapter 2 outline

2.1 Principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

[**2.6** video streaming and content distribution networks]

2.7 Socket programming with UDP and TCP

HTTP overview

HTTP: HyperText Transfer Protocol

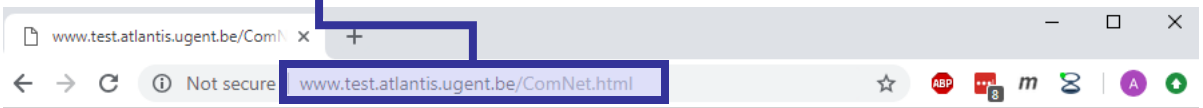
- Web's application layer protocol to exchange Web objects (e.g., html pages, videos, pictures)
- client/server model
  - client: browser that requests, receives, and "displays" Web objects
  - server: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068

text-based

- HTTP 2.0: RFC 7540
- HTTP 3.0: RFC 9114

Web object example

URL : <IP address> : port / <filepath of object>
<domain/host name>



ComNet.html

Base HTML-file which includes **several referenced objects**

ietf.gif

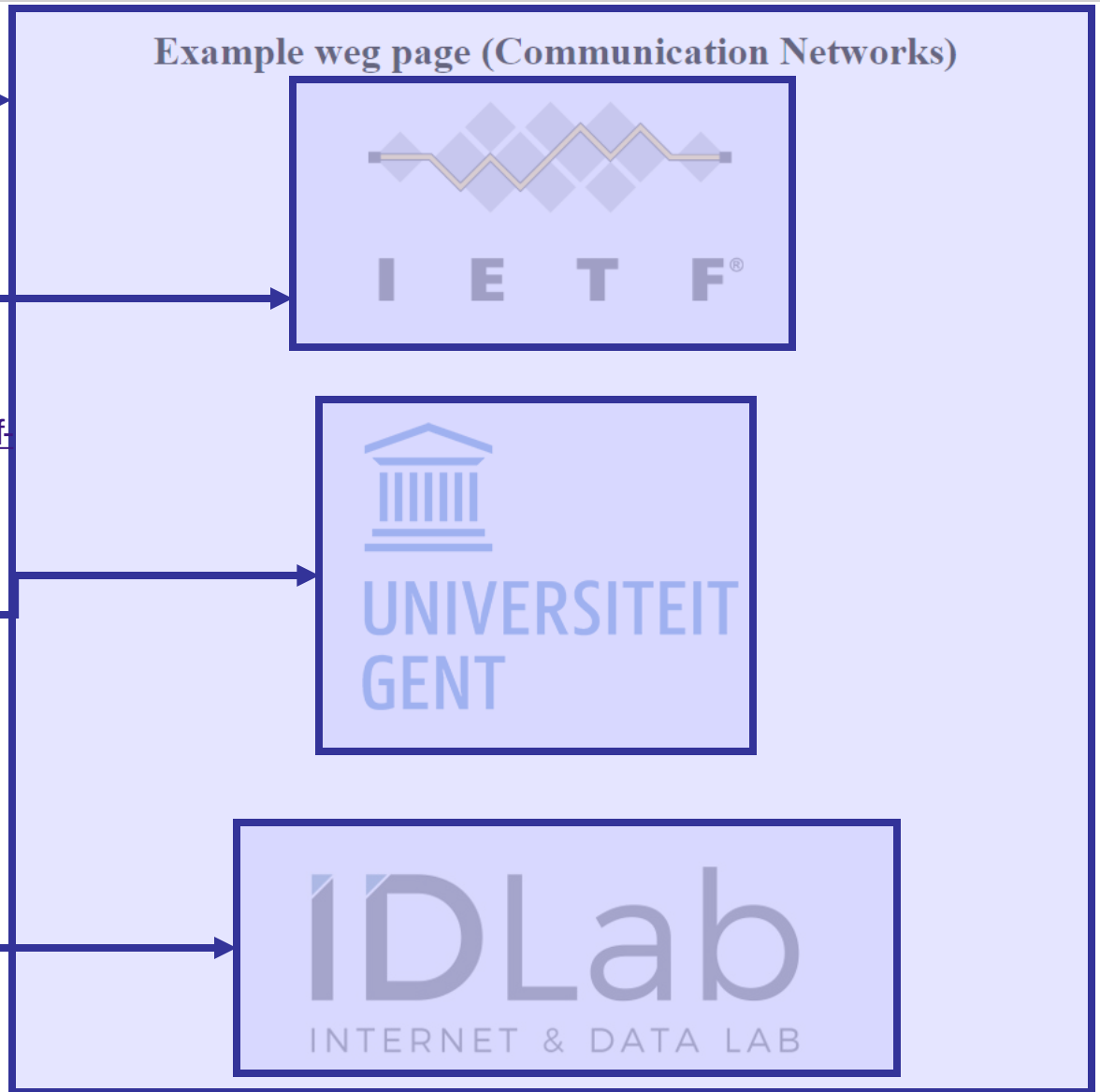
First referenced object (with own url: https://ietf.org/media/images/ietf-logo.original.png)

ugent.png

Second referenced object …

IDLab.jpg

Third referenced object …



HTTP overview

uses TCP:

- server is listening (open socket, port 80)
- client initiates TCP connection (creates socket, port >1024) 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

HTTP is "stateless"

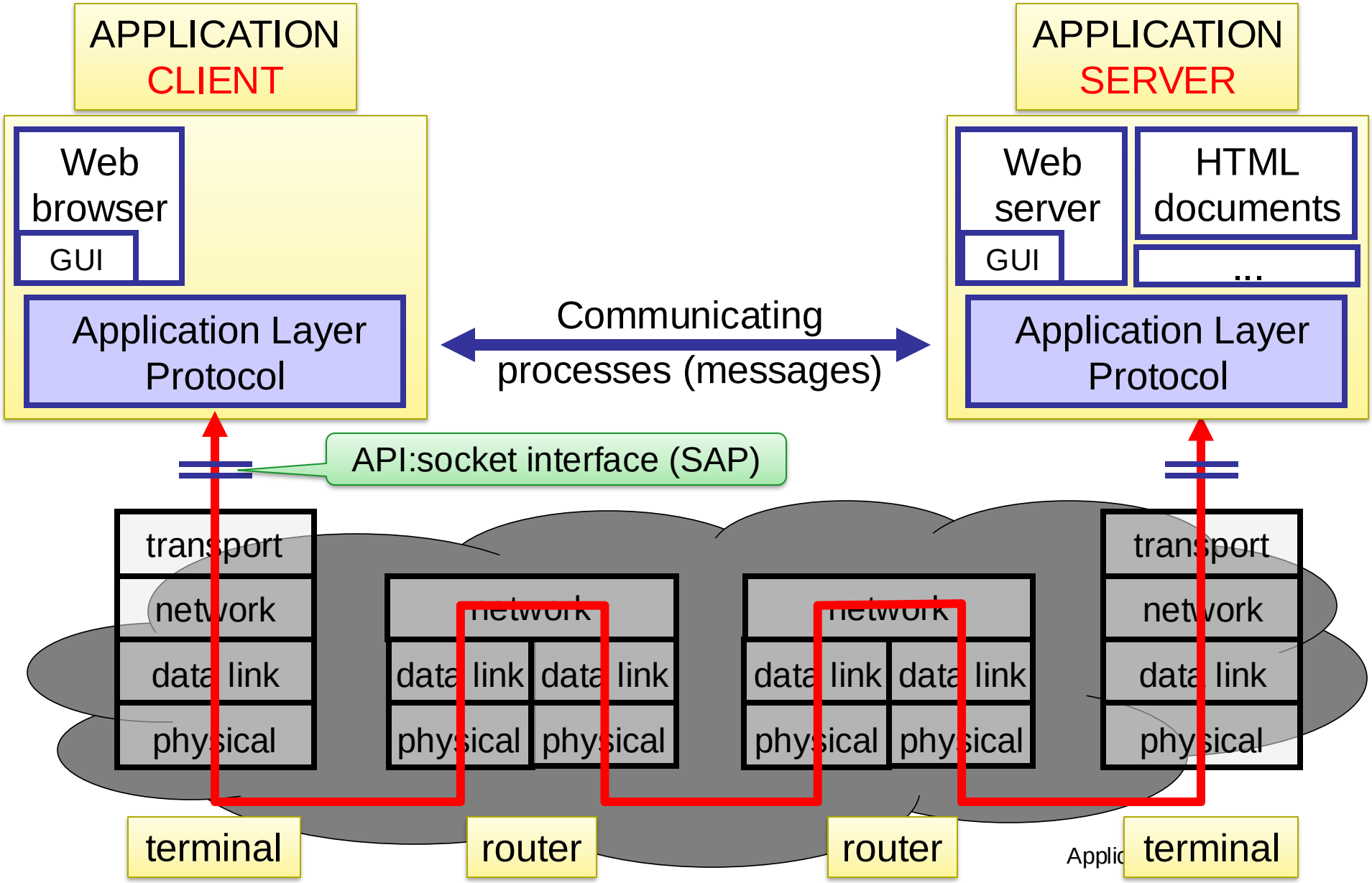
- server maintains no information about past client requests

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 in the global picture



What a browser does in background …

```
❯ telnet ugent.be 80
Trying 157.193.43.50...
Connected to ugent.be.
Escape character is '^]'.
GET / HTTP/1.1
Host: ugent.be

HTTP/1.1 301 Moved Permanently
Date: Mon, 16 Sep 2024 14:06:01 GMT
Server: Apache
Location: https://ugent.be/
Content-Length: 225
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://ugent.be/">here</a>.</p>
</body></html>

Connection closed by foreign host.
```

Set up a TCP connection

Issue HTTP requests

Receive HTTP response

Close TCP connection

Q: Try this at home … why are all modern webservers returning 301 ?

HTTP Connection Persistance

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects requires multiple connections
- HTTP/1.0 uses non-persistent HTTP

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server
- HTTP/1.1 uses persistent connections in default mode

Non-persistent HTTP

(contains text, references to 3 jpeg images)

Suppose user enters in the browser the URL

`http://www.test.atlantis.ugent.be/ComNet.html`

1a. HTTP client initiates TCP connection to HTTP server (process) at idlab.technology on port 80

1b. HTTP server at host `www.test.atlantis.ugent.be` waiting for TCP connection at port 80 "accepts" connection, notifying client

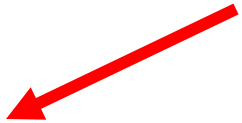
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `ComNet.html`

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

time

Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.



5. HTTP client receives response message containing html file, displays html and closes TCP connection. Parsing html file, finds 3 referenced .gif objects

6. Steps 1-5 repeated for each of 3 jpg objects

time

Non-persistent 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

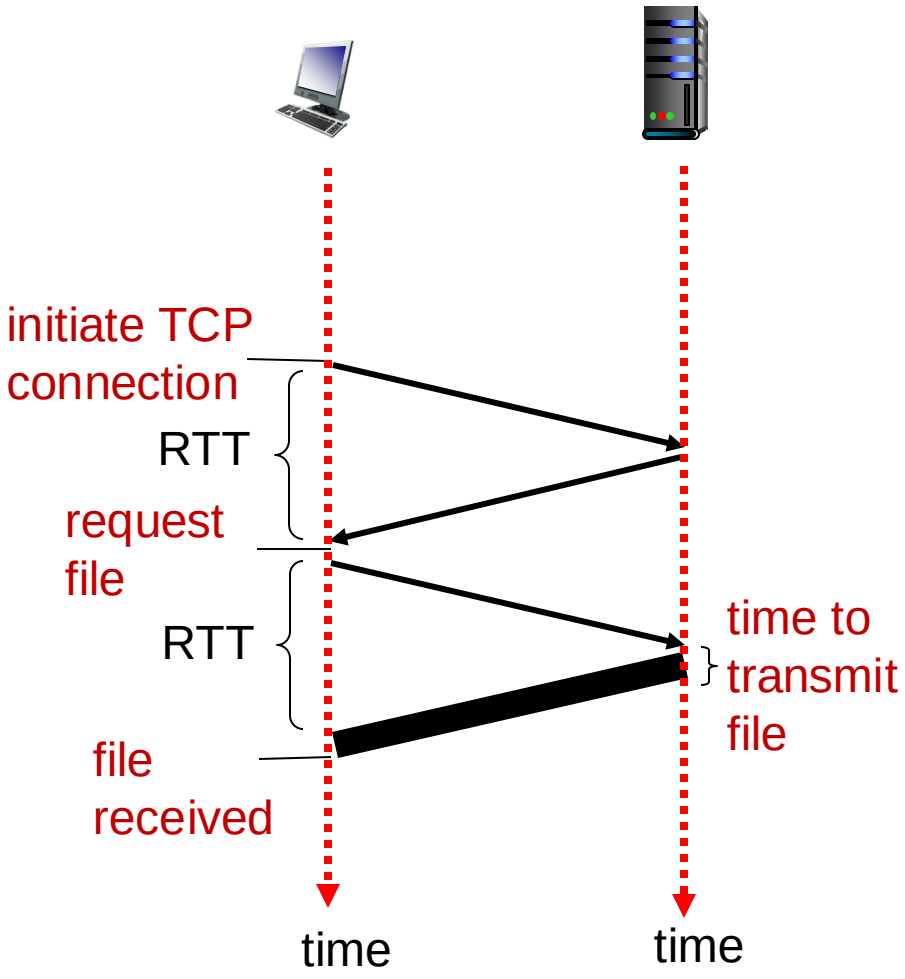
total = 2RTT+file transmit time

In practice:

1 KB @ 80 Kbps & 50 ms RTT

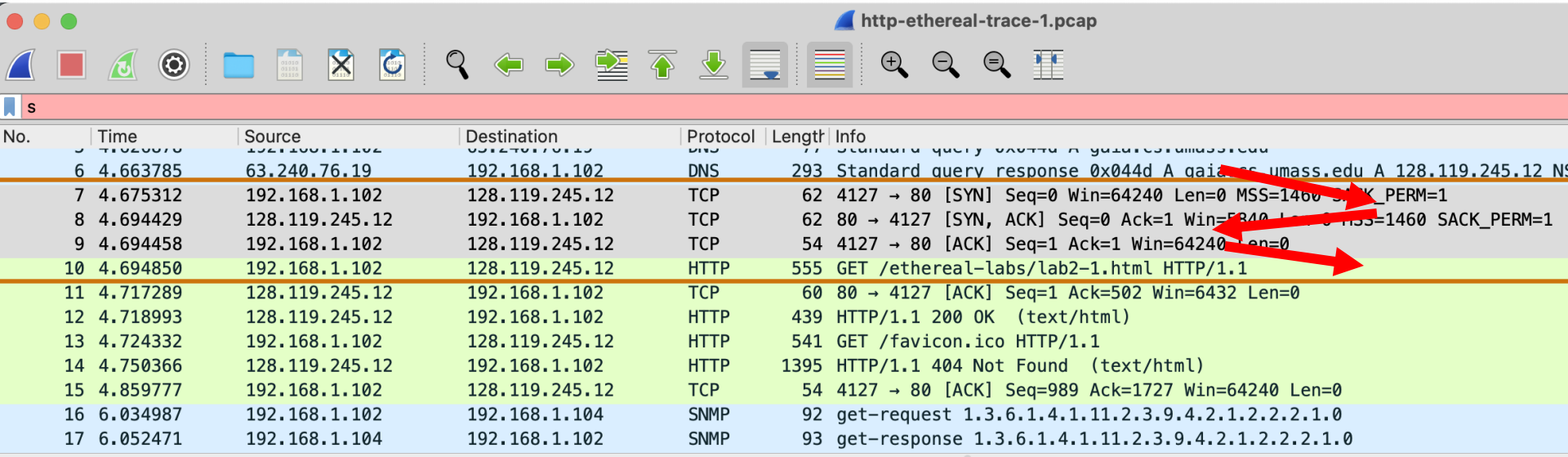
(10KBps x 1KB) + 2x0,05s = 0,2s

Effective throughput = 40 Kbps



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

TCP connection setup with webserver in Wireshark



Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but 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 are sent over connection

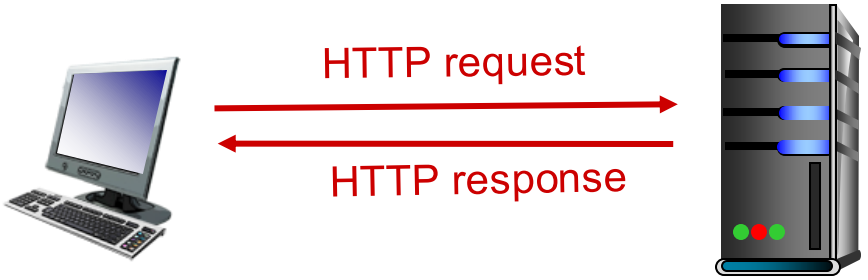
Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP messages



Request message

- Client -> Server
- Message structure

Method + url	headers	body

- **Request methods**:
  - GET
  - POST
  - HEAD
  - PUT
  - DELETE

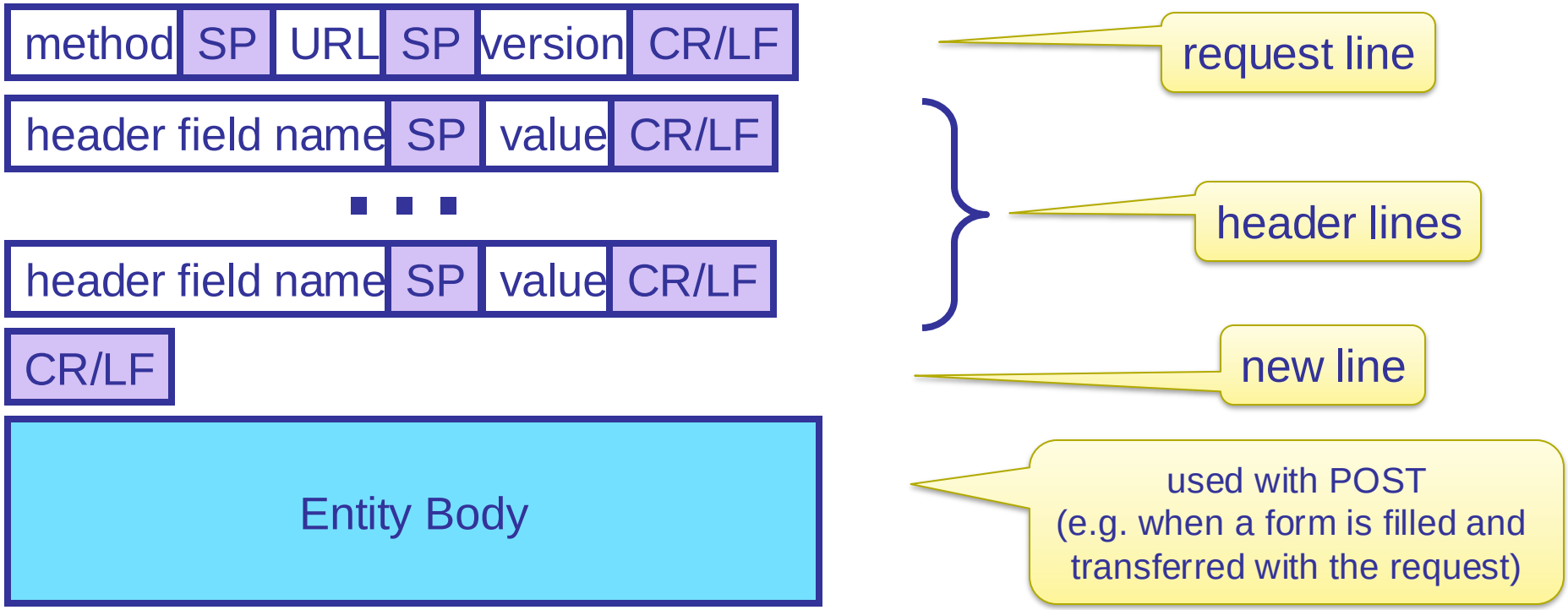
Response message

- Server -> Client
- Message structure

status	headers	body

- **Status codes:**
  - **200**
  - **301**
  - **400**
  - **404**
  - **505**

HTTP : Request Message




```
GET / HTTP/1.1
Host: neverssl.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.4 Safari/605.1.15
Accept-Language: en-GB,en;q=0.9
```

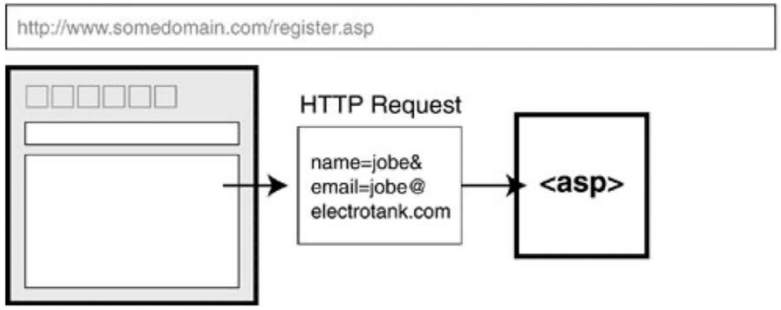
Request line

Header lines

Uploading Form Input

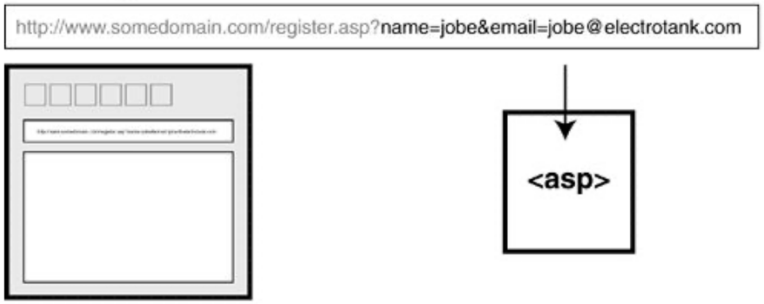
POST method:

- web page often includes form input
- input is uploaded to server in entity body

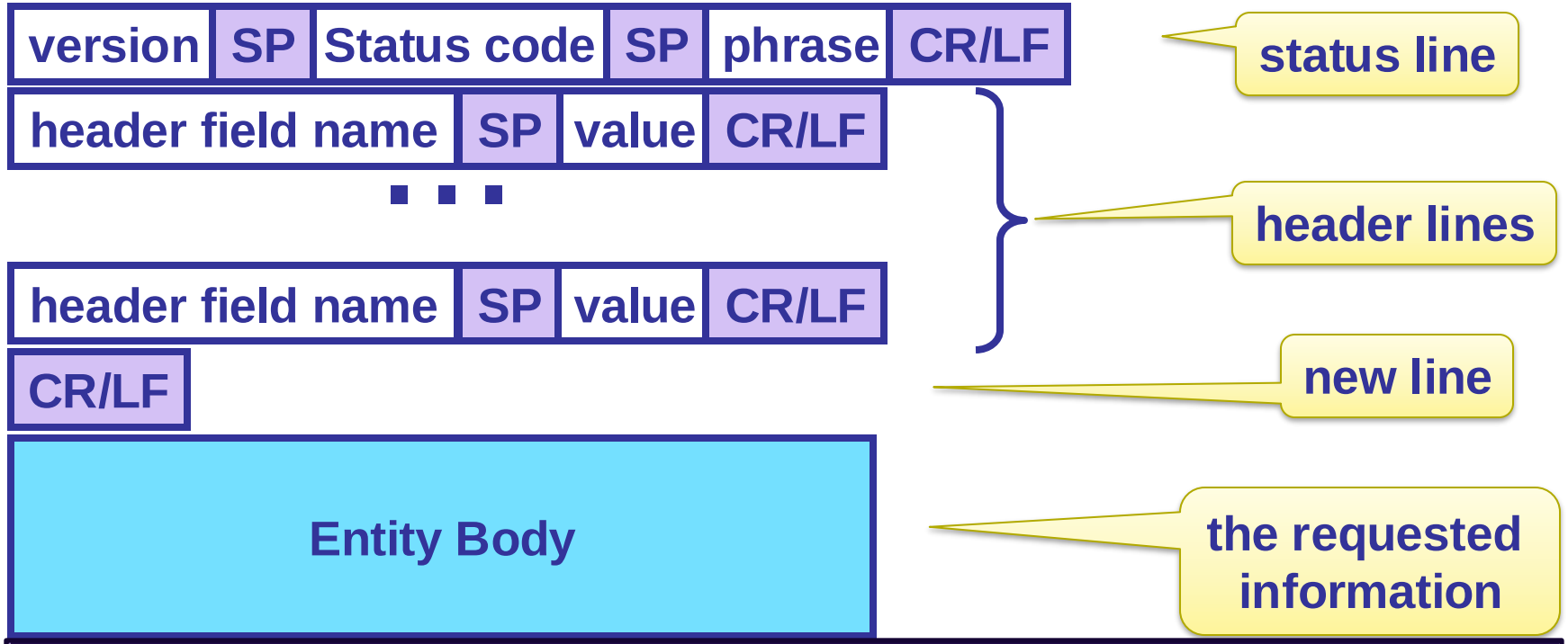


URL method:

- uses GET method
- input is uploaded in URL field of request line:



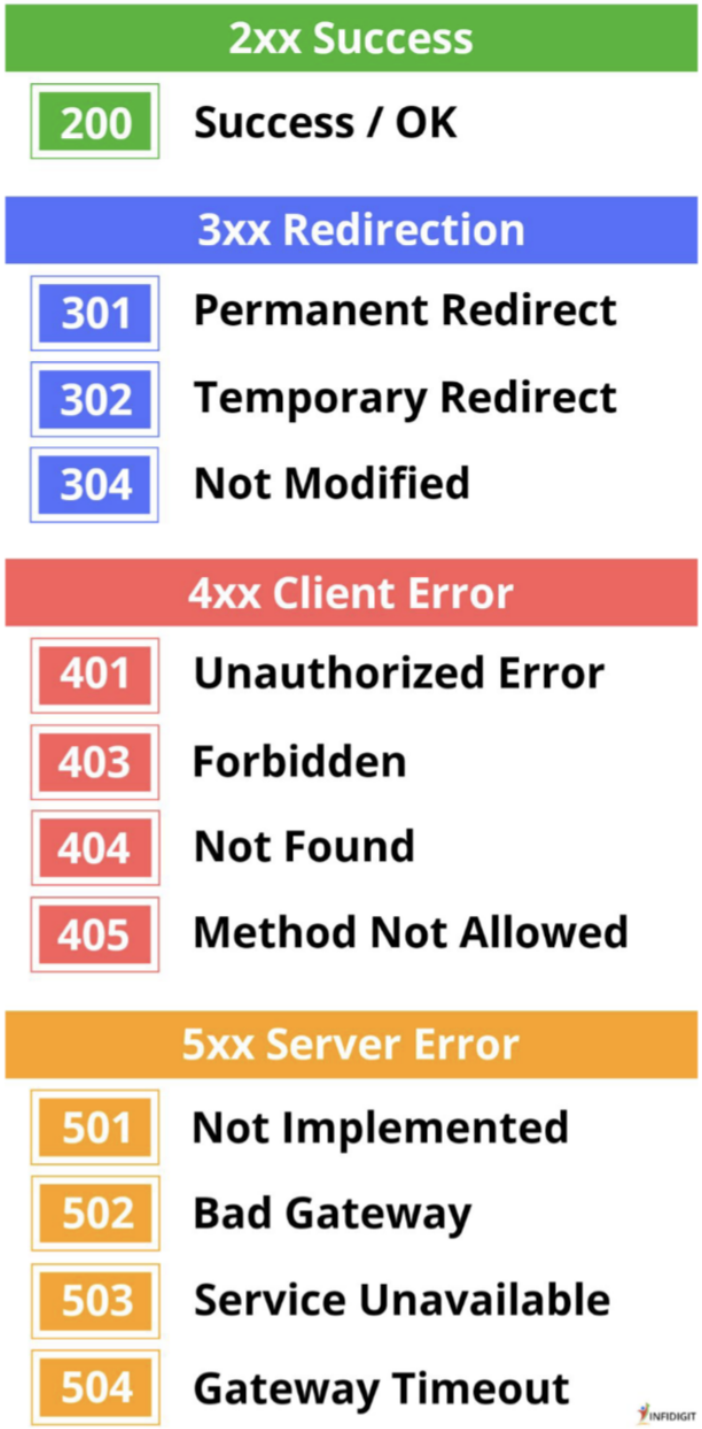
HTTP : Response Message




```
HTTP/1.1 200 OK                                          Status line
Date: Mon, 16 Sep 2024 14:32:25 GMT
Server: Apache/2.4.58 ()
Last-Modified: Wed, 29 Jun 2022 00:23:22 GMT
ETag: "8be-5e28b29291e10-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding                                    Header lines
Content-Encoding: gzip
Content-Length: 1173
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

HTTP Response Status

- Status code appears in 1st line in server-to-client response message.
- Most common = 200 – success
- Redirect
  - 301 permanent redirect
    - E.g. from http to https-page
  - 302 temporary
- 4xx error
  - Client did something wrong
- 5xx error
  - The server did something wrong



HTTP example (as in the old days)

```
❯ telnet neverssl.com 80
Trying 34.223.124.45...
Connected to neverssl.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: neverssl.com

HTTP/1.1 200 OK
Date: Mon, 16 Sep 2024 14:43:14 GMT
Server: Apache/2.4.58 ()
Upgrade: h2,h2c
Connection: Upgrade
Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT
ETag: "f79-5e28b29d38e93"
Accept-Ranges: bytes
Content-Length: 3961
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

<html>
…
        <h2>How?</h2>
        <p>neverssl.com will never use SSL (also
  known as TLS). No encryption, no strong
  authentication, no HTTP/2.0, just plain old
  unencrypted HTTP and forever stuck in the dark
  ages of internet security.</p>
…
</html>
Connection closed by foreign host
```

connection set-up

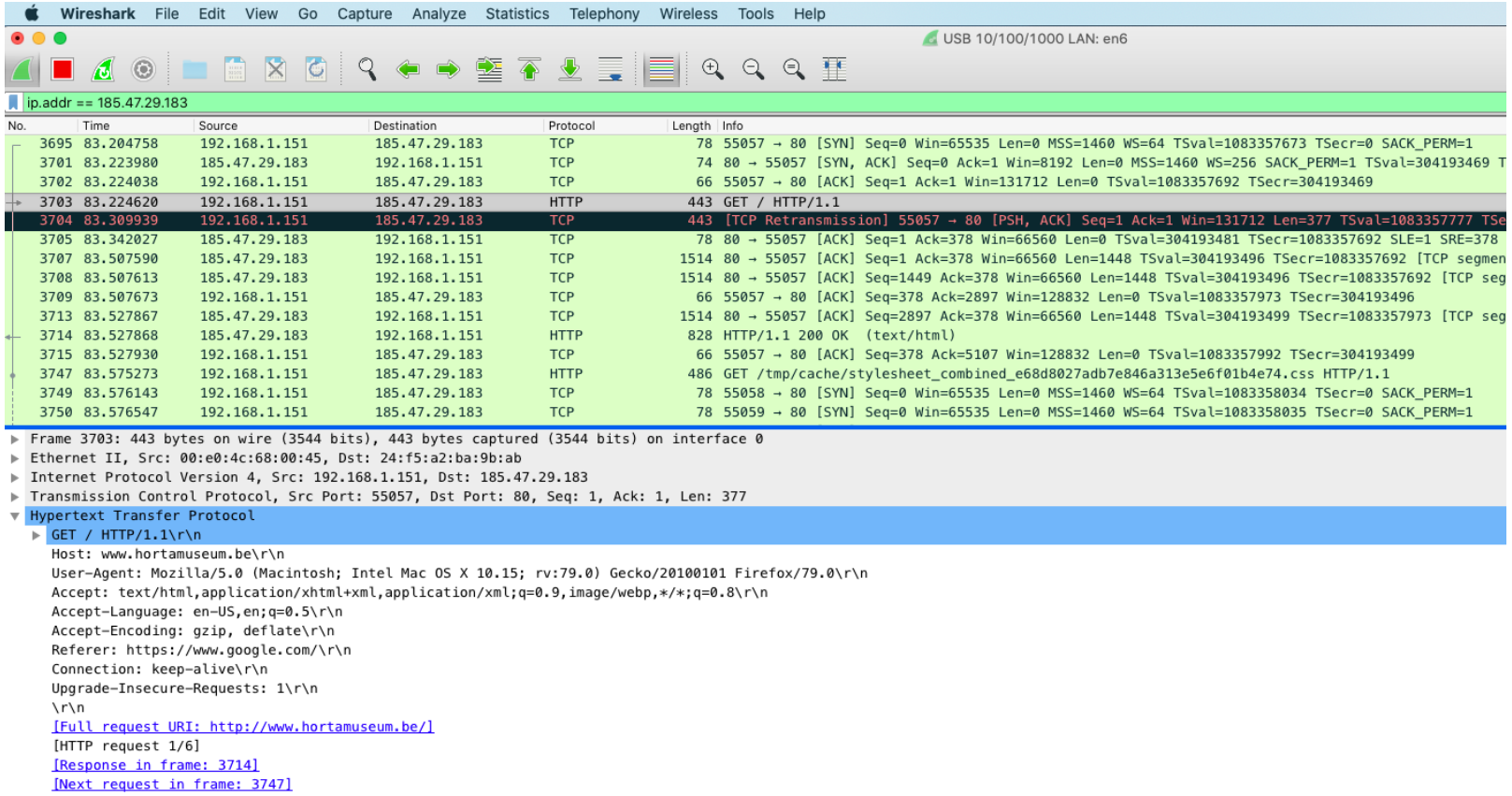
request to get homepage

reply header

HTML document neverssl homepage

close connection

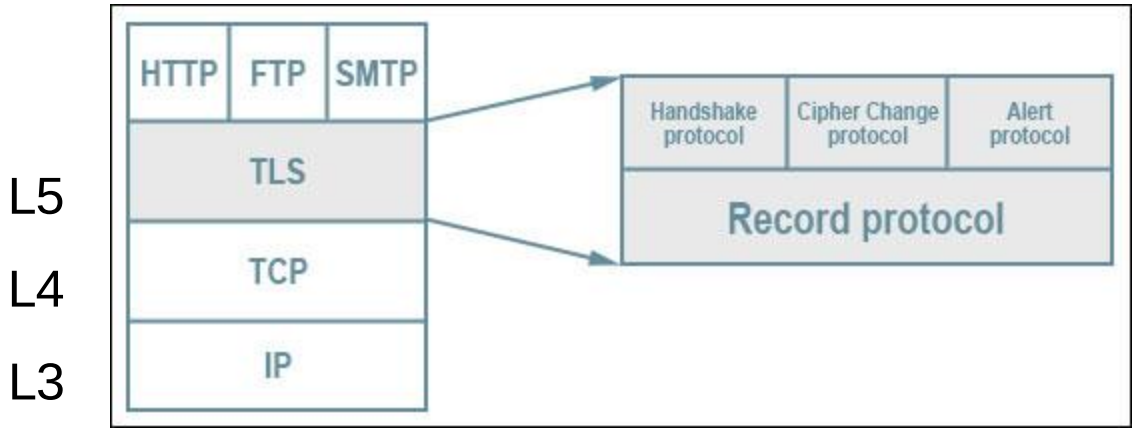
HTTP GET IN WIRESHARK



Transport Layer Security (TLS)

- (plain) HTTP relies directly on TCP
  - All application layer communication is sent unencrypted over TCP over the Internet
  - “anyone” can monitor what web pages you visit, etc.

- Alternative = Transport Layer Security (also Secure Sockets Layer):
  - Set up an encrypted connection between client/server first over TCP -> Cfr. Chapter 8
  - Exchange all application layer traffic (HTTP) over the encrypted channel -> HTTPS

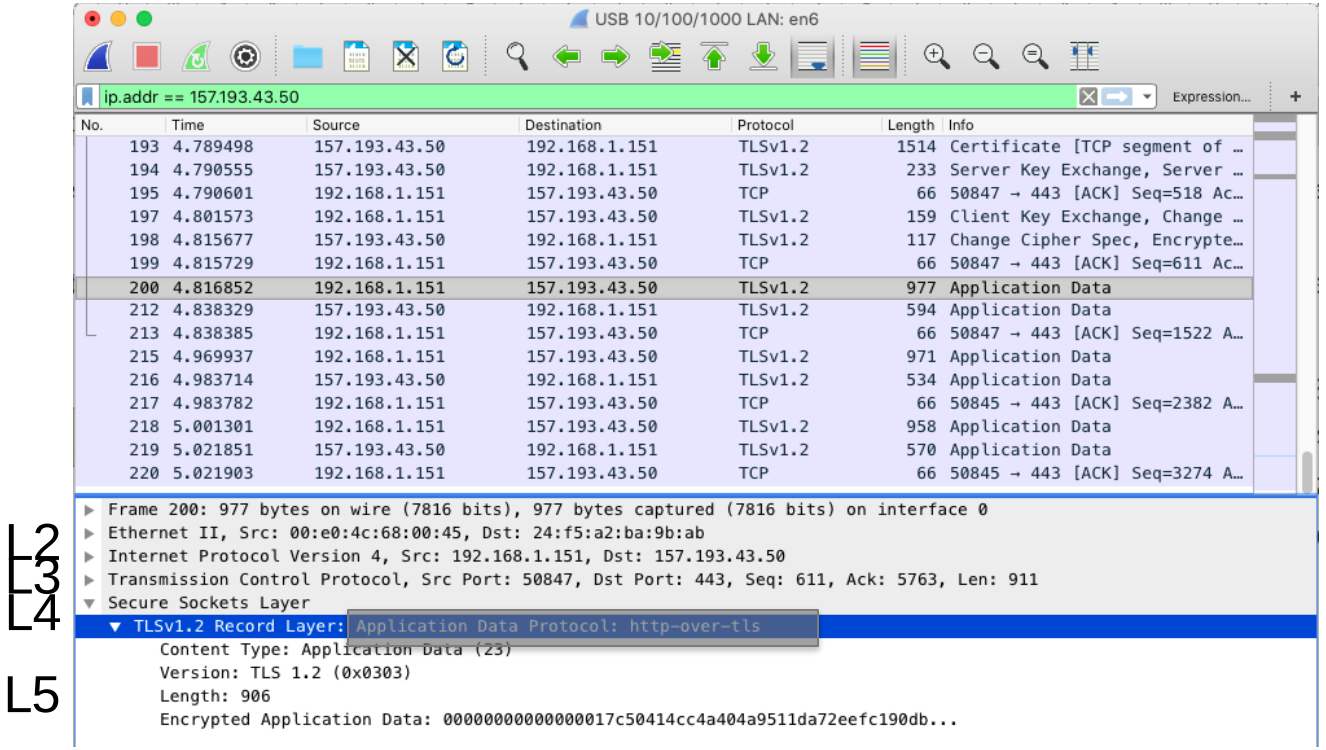


HTTP today (using HTTPS)

```
$ openssl s_client -crlf -connect www.ugent.be:443

<certification handshake>

GET / HTTP/1.1

HTTP/1.1 200 OK
Date: Thu, 23 Sep 2021 09:35:38 GMT
Server: waitress
…
Accept-Ranges: bytes
Via: 1.1 www.ugent.be
Set-Cookie: ROUTEID=.1; path=/

<html>
   <head>
   <title>UNIVERSITEIT GENT - UNIVERSITY OF
 GHENT</title>
   <style type="text/css">
   ...
   </style>
</head>

<body bgcolor="#000066" link="#cccccc"
 vlink="#cccccc" alink="#666666">
   ...
</body>
</html>
Connection closed by foreign host
```

secure connection set-up using TLS Chapter 8

Web session to ugent.be -> HTTPS




HOME PAGE
TODAY'S PAPER
VIDEO
MOST POPULAR
U.S. Edition
Try a Digital Subscription
Log In
Register No
THE WORLD'S CITI. IT'S WHEREVER YOU ARE.
The New York Times
Monday, October 14, 2013
Last Update: 1:42 PM ET
citi
The World's Citi™
Search
Follow Us
Personalize Your Weathe
Why is the homepage of my newspaper "aware" of my interest in bank loans and intentions to buy a house?
WHEREVER THERE'S OPPORTUNITY, WE'RE THERE TO HELP MAKE IT REAL.
citi
The World's Citi™
WORLD
U.S.
POLITICS
NEW YORK
BUSINESS
DEALBOOK
TECHNOLOGY
SPORTS
SCIENCE
HEALTH
ARTS
STYLE
OPINION
Autos
Blogs
FISCAL CRISIS
Seeking Deal to Avert Default, Lawmakers to Meet Obama
By JONATHAN WEISMAN
12:51 PM ET
Senate talks on reopening the government and raising the debt ceiling were expected to accelerate Monday afternoon, and President Obama was to meet with top Congressional
RETRO REPORT | VIDEO
Portents of Doom in a Sheep Named Dolly
The Opinion Pages
How the G.O.P. Is Shaping the Midterms
Republicans have alienated the public. In 2014, can independents capitalize on that?
ROOM for DEBATE
MORE IN OPINION
· Editorial: False Equality in Michigan
· Stiglitz: Inequality Is a Choice
· Op-Ed: The Bay of Bengal
TURNING THE PAGE
How We Called It, Down Through the Years
A look back at a selection of columns, essays and editorials that were published in The International Herald Tribune.
· Visit the Special Section »
OP-ED COLUMNISTS
· Keller: Obamacare, the Rest of the Story
· Krugman: The Dixiecrat Solution

User-Server State: Cookies

Many Web sites use cookies

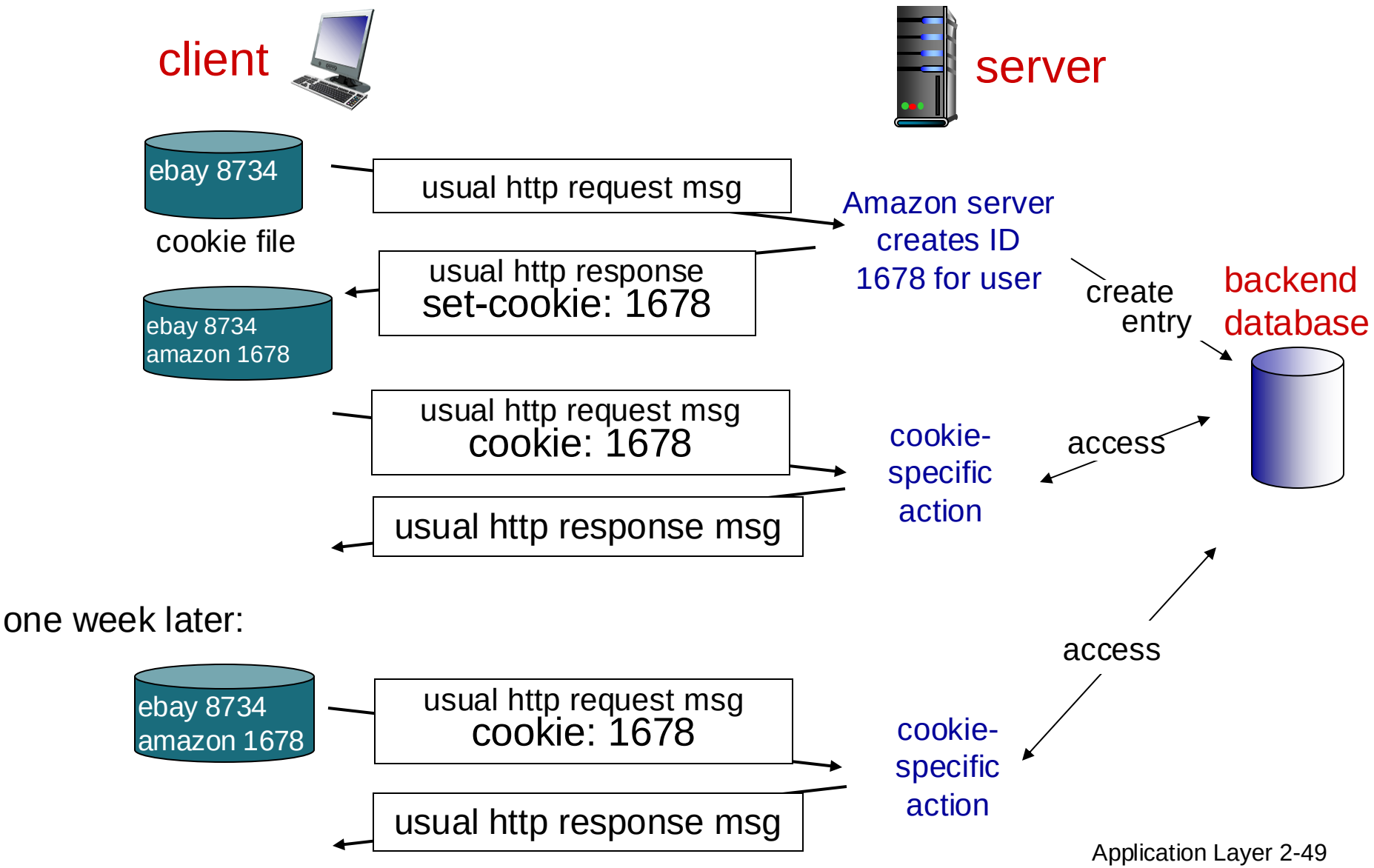
Cookies rely on four components:

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 always accesses Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state"



Cookies

what cookies can be used for:

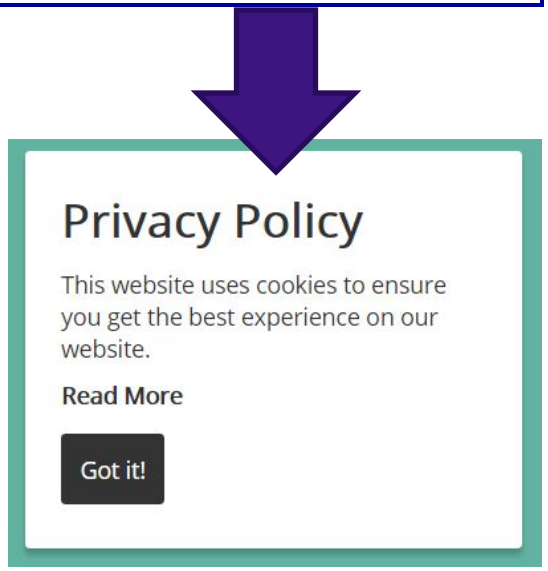
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

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

cookies and privacy: **aside**

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- EU GDPR law requires explicit agreement for use of cookies since 2018



Conditional GET: client-side caching

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

- no object transmission delay
- lower link utilization

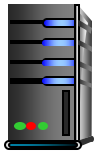
- client: specify date of cached copy in HTTP request

```
If-modified-since:
   <date>
```

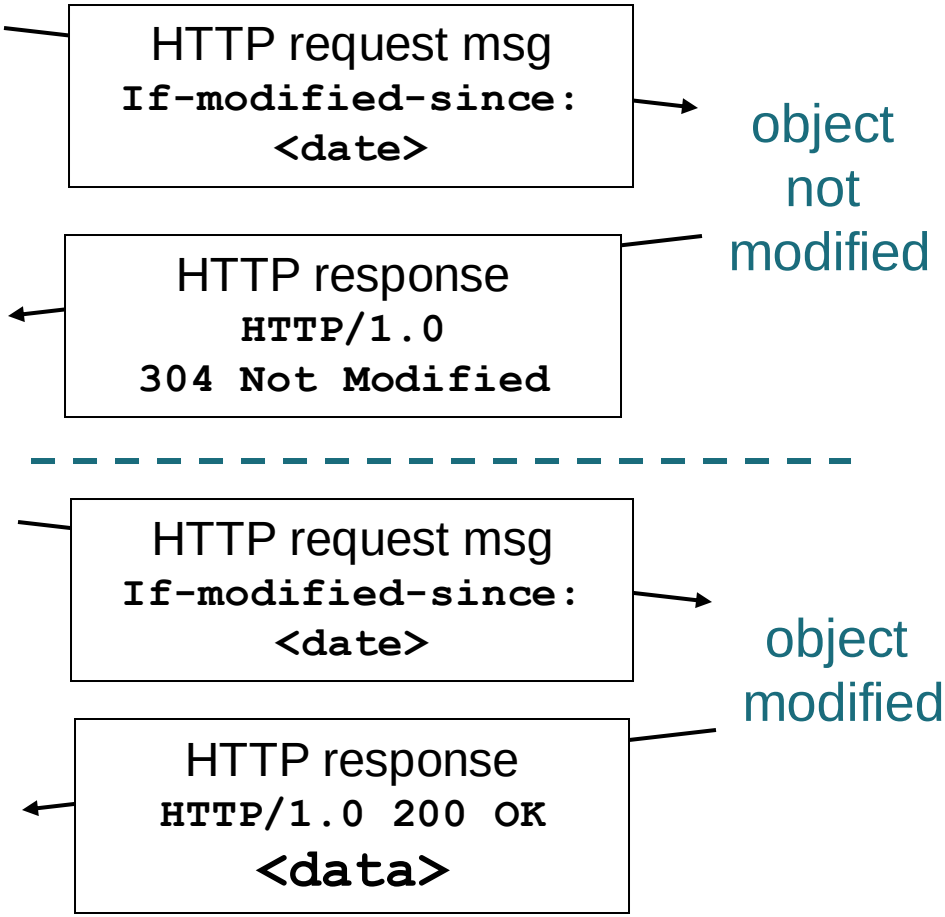
- server: response contains no object if cached copy is up-to-date:

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

client



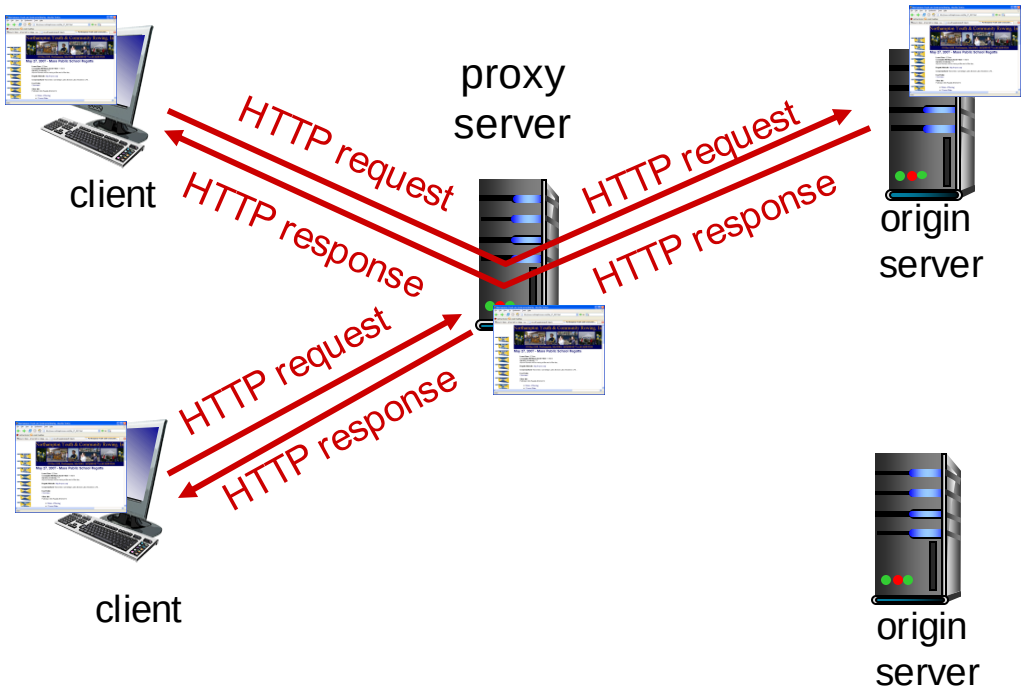
server



Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More About Web Caching

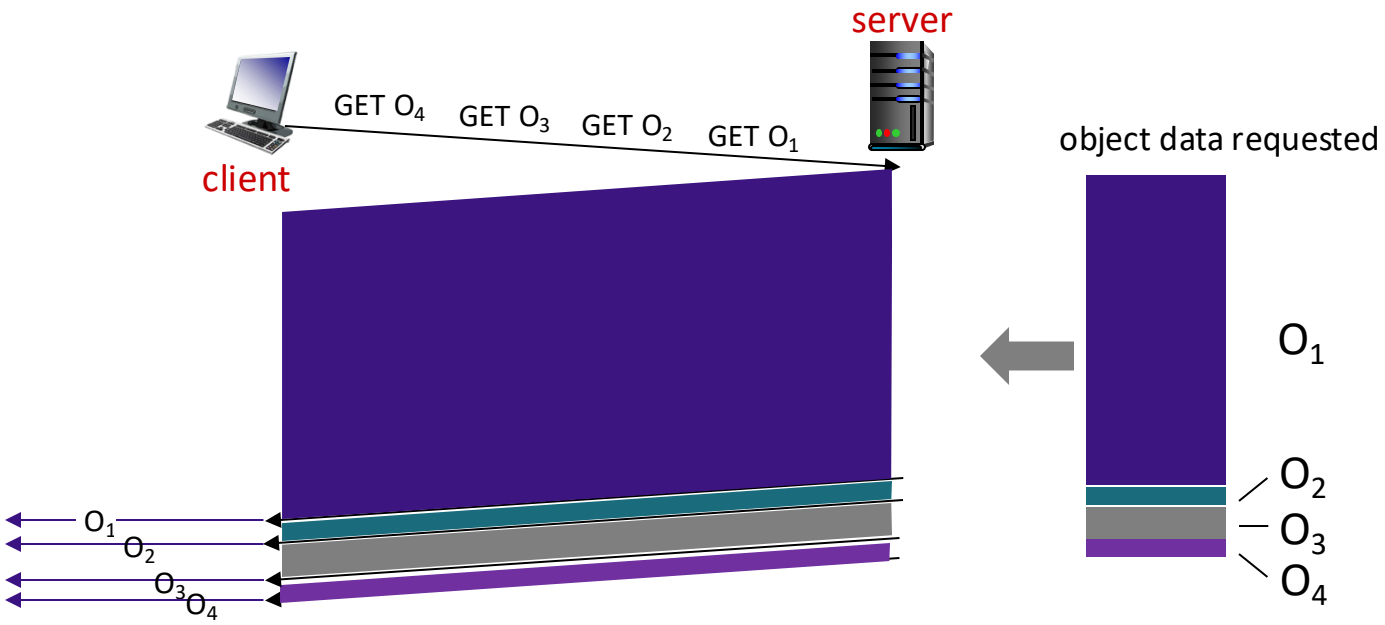
- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content
  (so too does P2P file sharing)

HTTP 1.1 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

HTTP/2

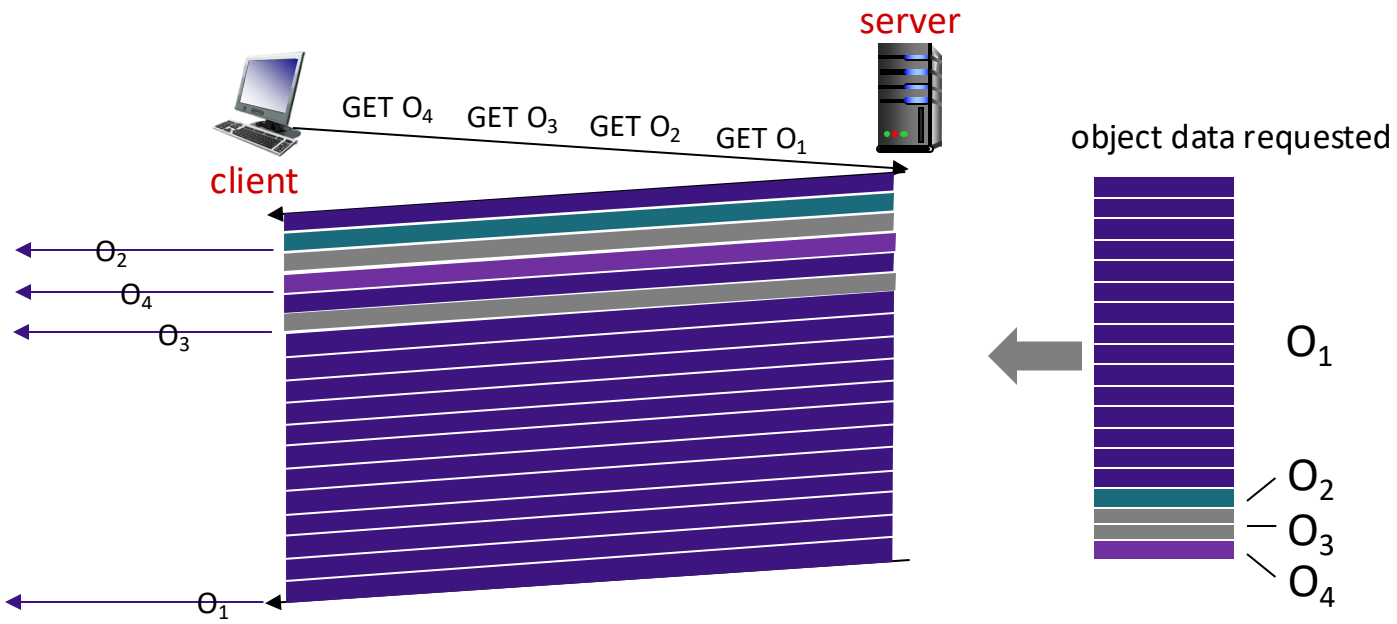
Key goal: decreased delay in multi-object HTTP requests

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

HTTP evolution and versions

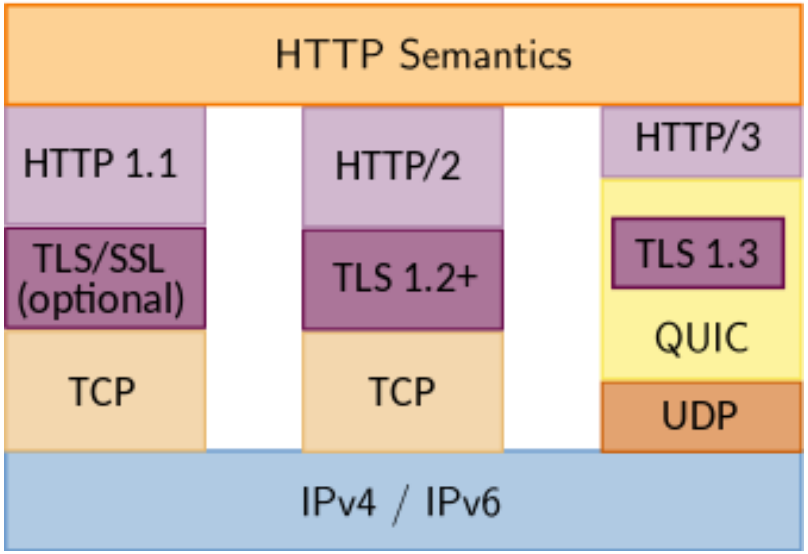
Overview
On Youtube

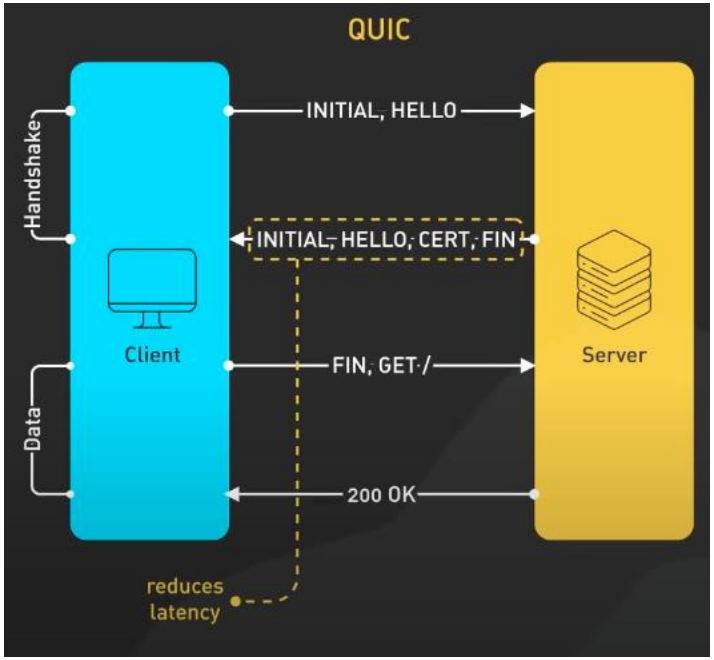
HTTP/1.0	HTTP/1.1	HTTP2.0
GET, POST	Persistent connections	Improve page load speed (decreasing latency) • Data compression of HTTP headers • Multiplexing multiple requests • HTTP/2 Server Push
HEAD asks server to leave requested object out of response	OPTIONS method Used to determine abilities of the server.	Longer lived (~permanent) HTTP connections
PUT uploads file in entity body to path specified in URL field DELETE deletes file specified in the URL field	Additional caching, authenti-cation & compression options	HTTP/2 is a binary protocol instead of text-based (as 1.*)

HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

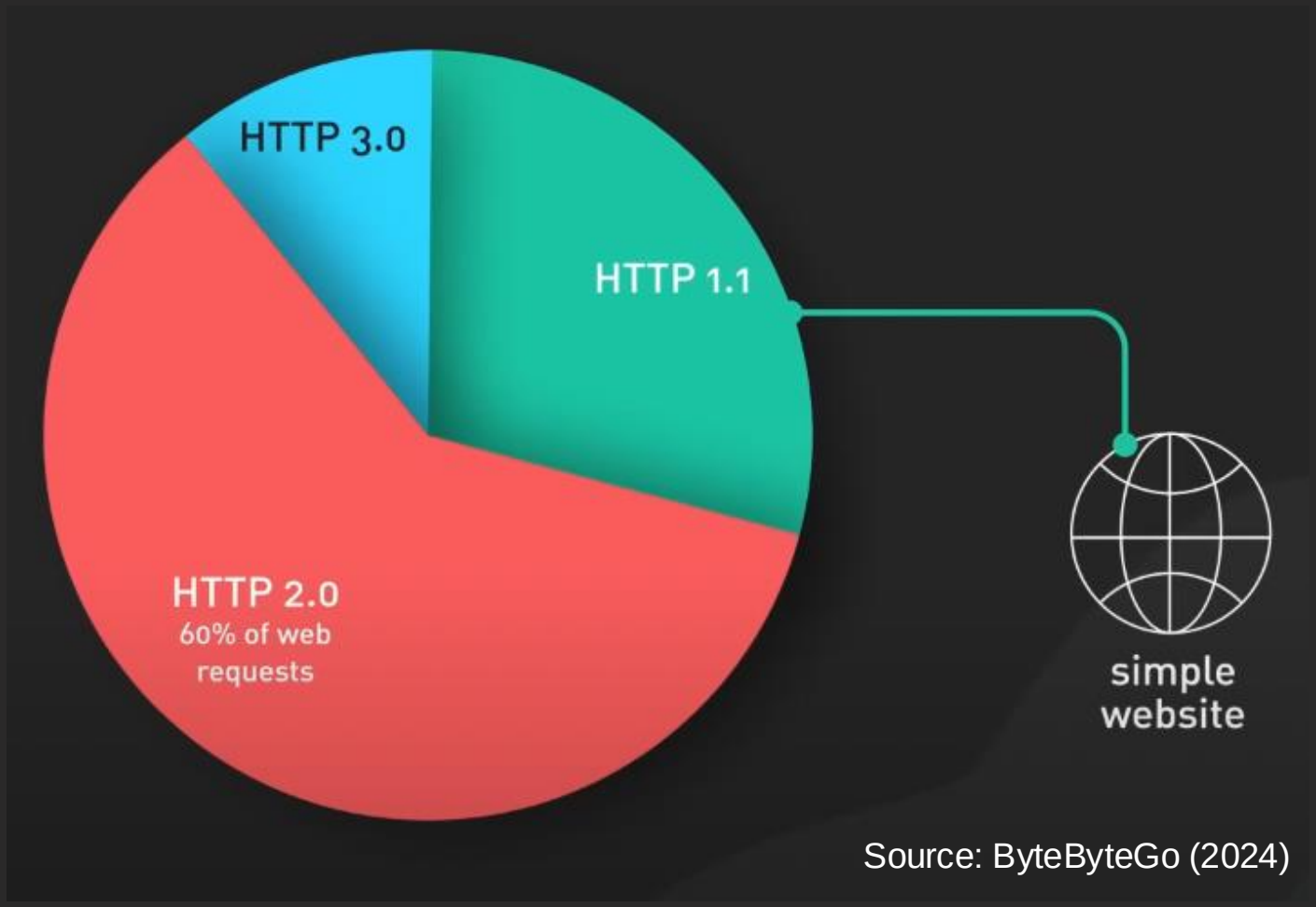
- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security, per object error- and congestion-control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





Overview On Youtube

Adoption



Source: ByteByteGo (2024)

Experiment in Kathará

- Chapter 2 -> http experiment
  - Involves commonly used Apache 2 webserver(s)
  - Use wget to fetch page including images
- Fetch homepage of server on client
  - Which HTTP version?
  - Which HTTP messages?
  - How many TCP connections?