Computer Networks
COL 334/672

Congestion Control

Tarun Mangla

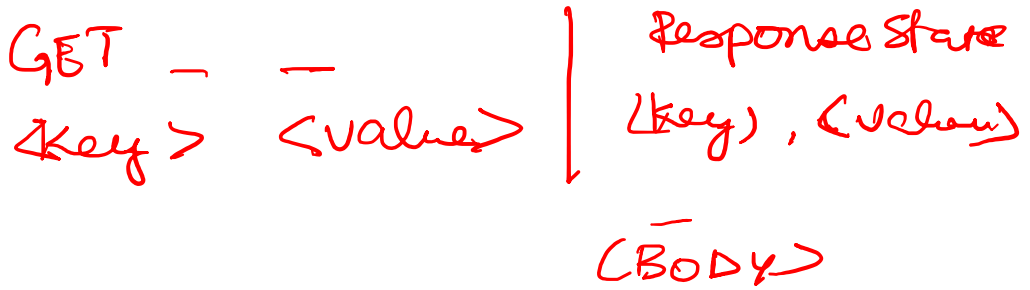
Slides adapted from KR

Sem 1, 2024-25

Quiz password: http

Recap: Application Layer

- HTTP
  - Overview of HTTP
  - Request/response message format
  - State management
  - Caching
  - Request pipelining
- Email
- DNS
- P2P
- Video streaming

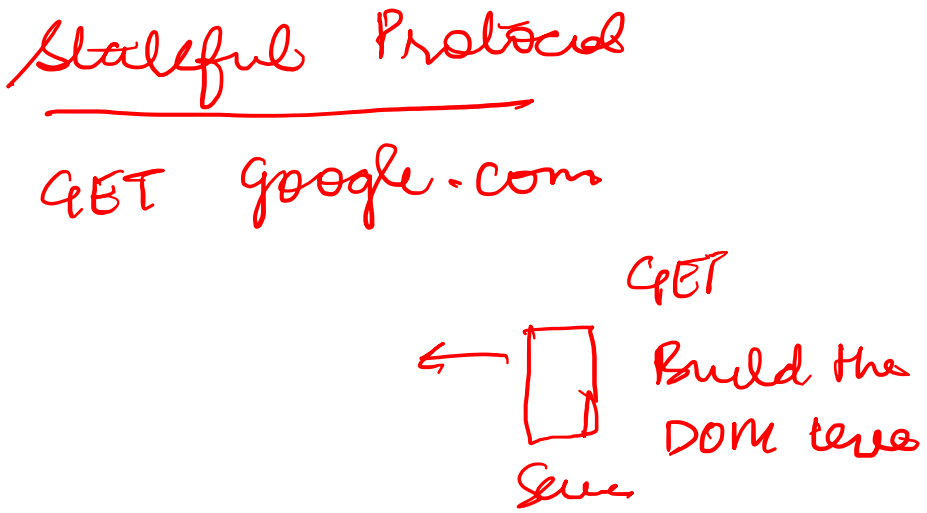


HTTP is stateless

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction

Makes the design of HTTP servers much simpler

One still needs to maintain state in a few cases



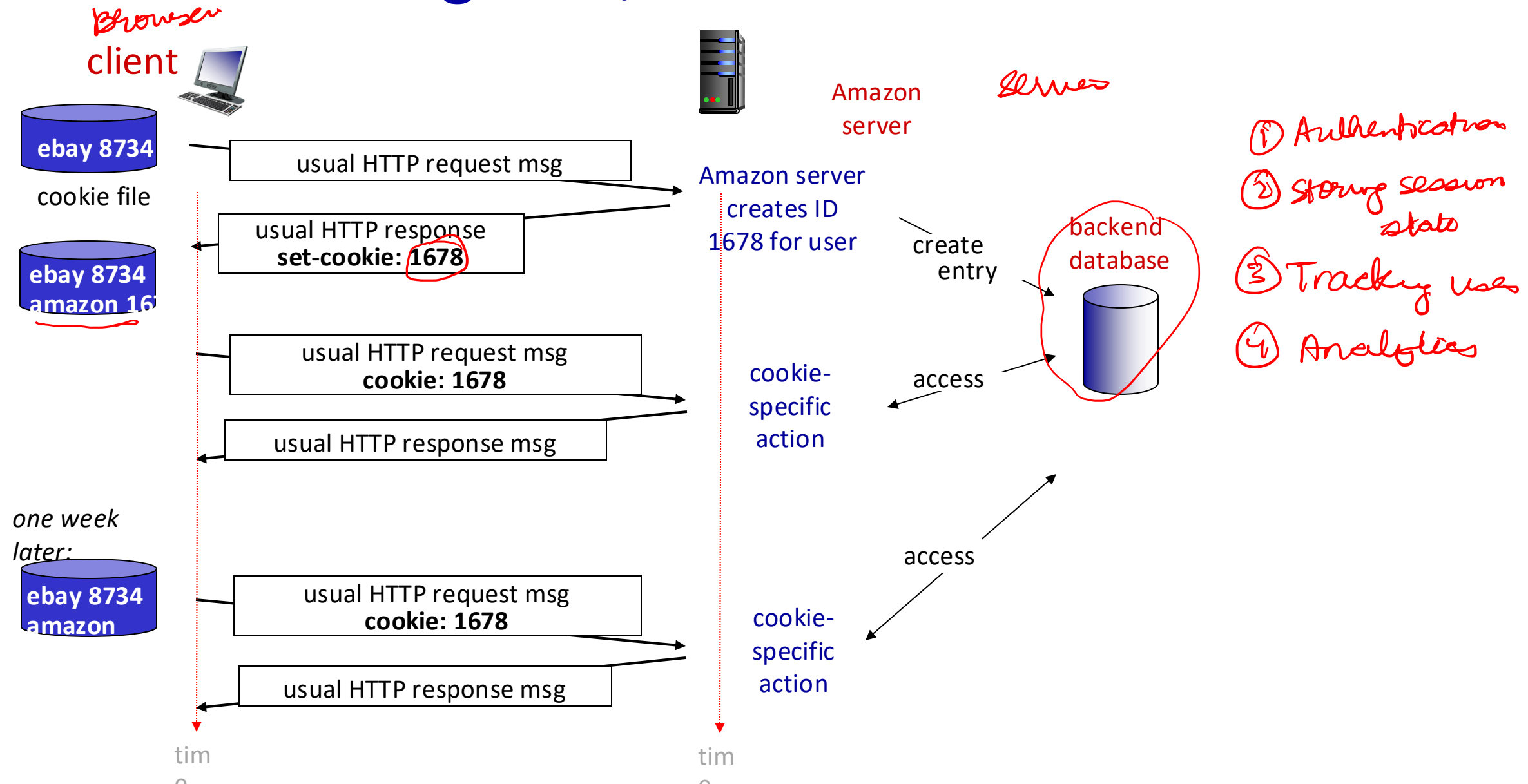
Maintaining user/server state: cookies

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

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)

aside

cookies and privacy:

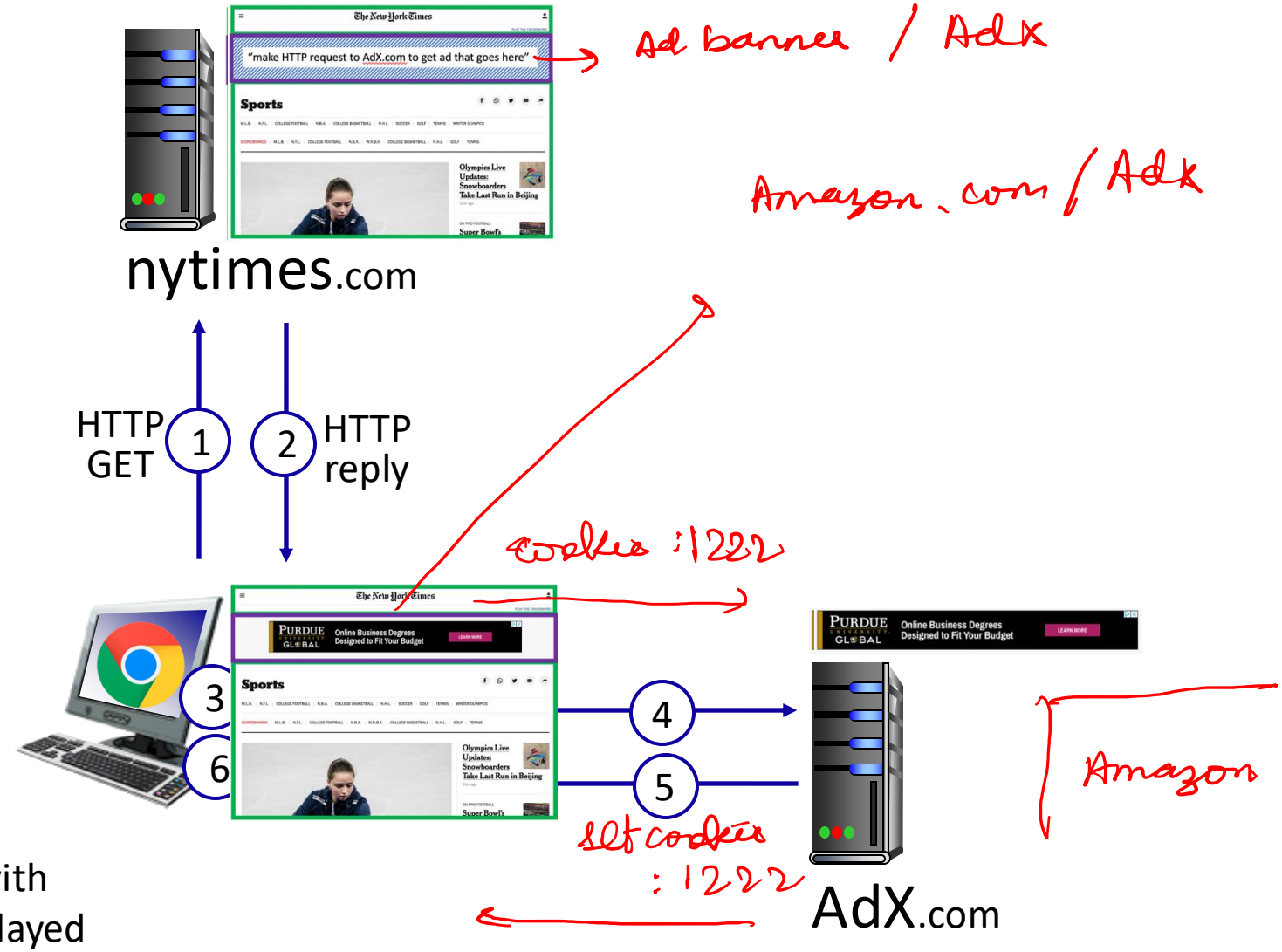
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Example: displaying a NY Times web page

1 2 GET base html file from nytimes.com

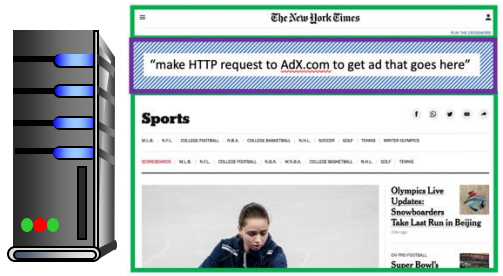
4 5 fetch ad from AdX.com

7 display composed page

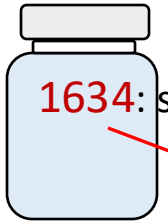


NY times page with embedded ad displayed

Cookies: tracking a user's browsing behavior



nytimes.com (sports)



1634: sports, 2/15/22

"first party" cookie – from website you chose to visit (provides base html file)

HTTP GET

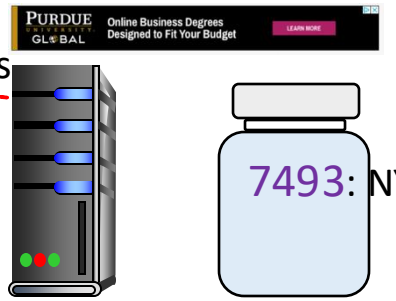
HTTP reply
Set cookie: 1634

HTTP GET
Referrer: NY Times Sports

4

5

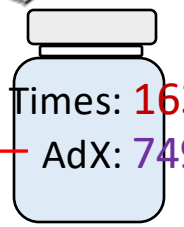
HTTP reply
Set cookie: 7493



AdX.com

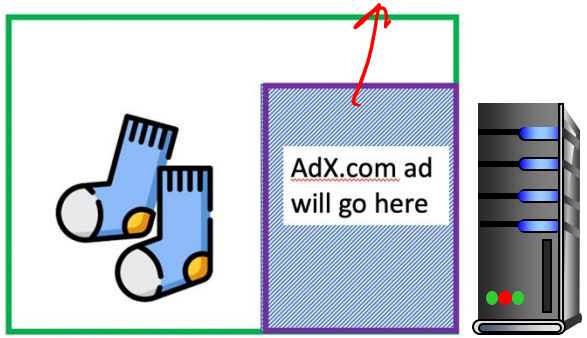
7493: NY Times sports, 2/15/22

"third party" cookie – from website you did not choose to visit

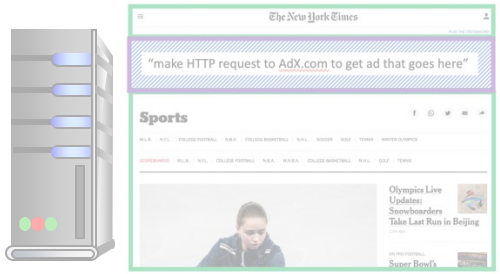


NY Times: 1634
AdX: 7493

Cookies: tracking a user's browsing behavior



socks.com



nytimes.com

1634: sports, 2/15/22

HTTP reply

2

HTTP GET

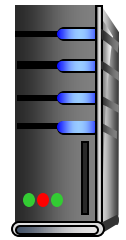
1

HTTP GET
Referrer: socks.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493



AdX.com

7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22

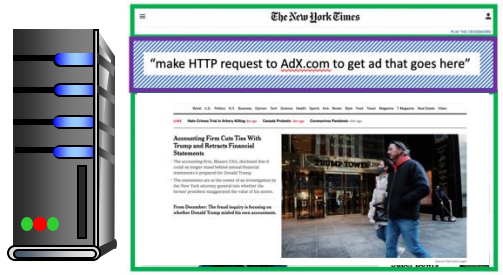
NY Times: 1634
AdX: 7493

AdX:

- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

Cookies: tracking a user's browsing behavior (one day later)

socks.com



nytimes.com (arts)

1634: sports, 2/15/22
1634: arts, 2/17/22

HTTP GET
cookie: 1634

HTTP reply
Set cookie: 1634

HTTP GET
Referrer: nytimes.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493
Returned ad for socks!

NY Times: 1634
AdX: 7493

AdX.com

7493: NY Times sports, 2/15/22
7493: socks.com, 2/16/22
7493: NY Times arts, 2/15/22

Cookies: tracking a user's browsing behavior

Cookies can be used to:

- track user behavior on a given website (first party cookies)
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
  - rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

Data protection laws and cookies

CCPA : California Data Protection

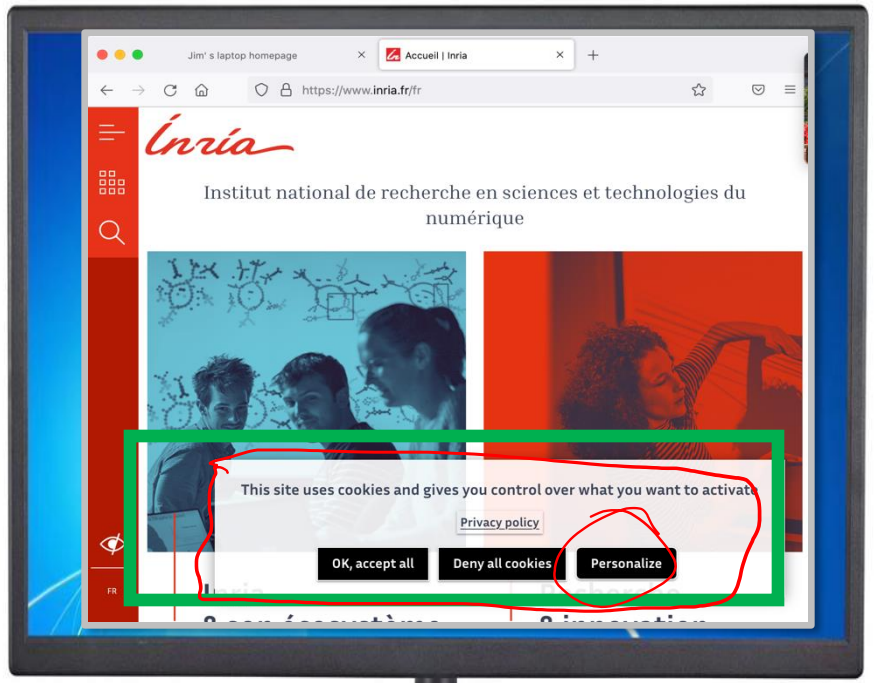
"Natural persons may be associated with online identifiers [...] such as internet protocol addresses, cookie identifiers or other identifiers [...].

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)

India's data protection law: DPDPA

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



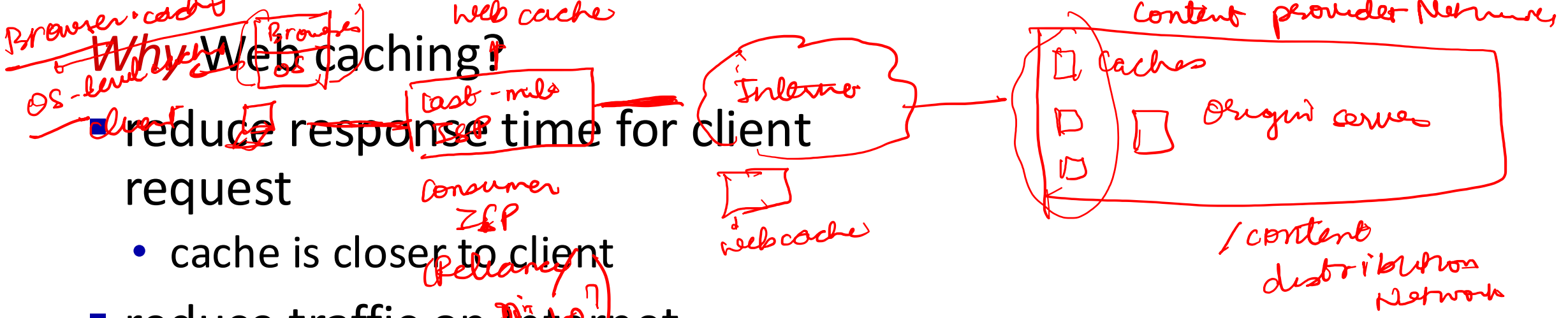
User has explicit control over whether or not cookies are allowed

HTTP

- Overview of HTTP
- Request/response message format
- State management
- **Caching**
- Request pipelining

Caching

Goal: satisfy client requests without involving origin server



Why Web caching?

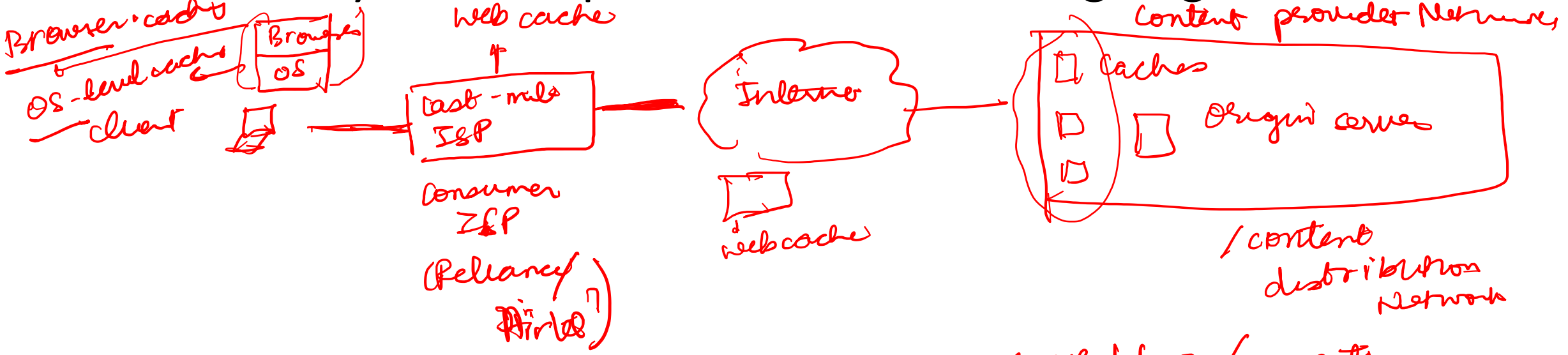
- reduce response time for client request
  - cache is closer to client
- reduce traffic on Internet backbone
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Expiry date:
If-modified since 9:00 AM

Caching

NYTIMES.com → Day 0, Time 9:00 AM
Caching Mech
→ Day 0, Time 5:00 PM

Goal: satisfy client requests without involving origin server

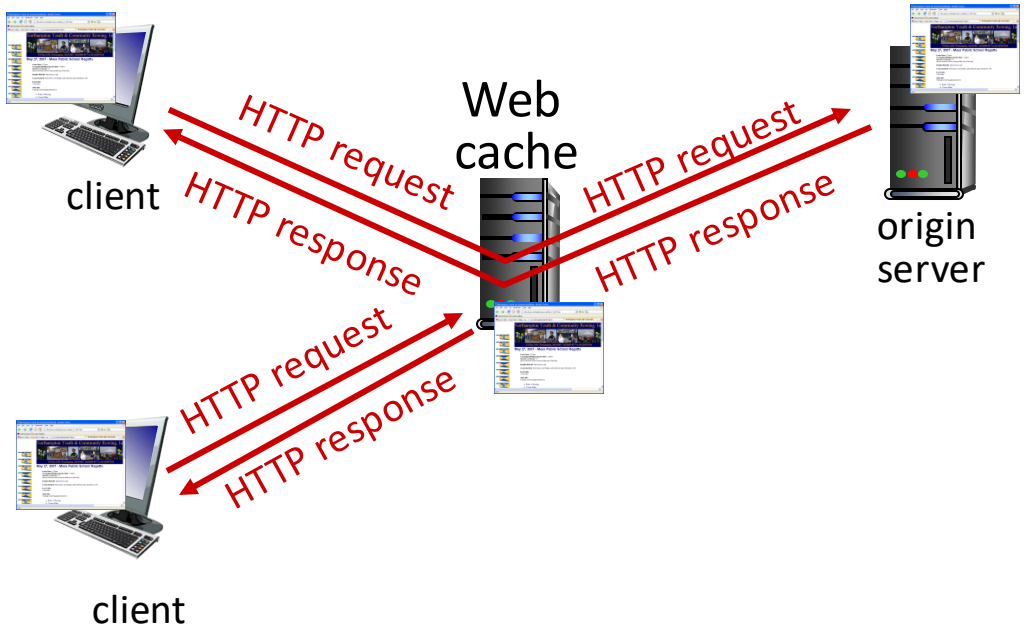


→ Save b/w / costs
→ improve user experience
① End-to-end encrypted HTTPS)
① content is more dynamic
② No need / b/w has improved
③ Network latency

Where in the network path can caching occur?

Forward Cache 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



HTTP caching mechanism: server tells cache about object's allowable caching in response header:

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

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

Browser caching: Conditional GET

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

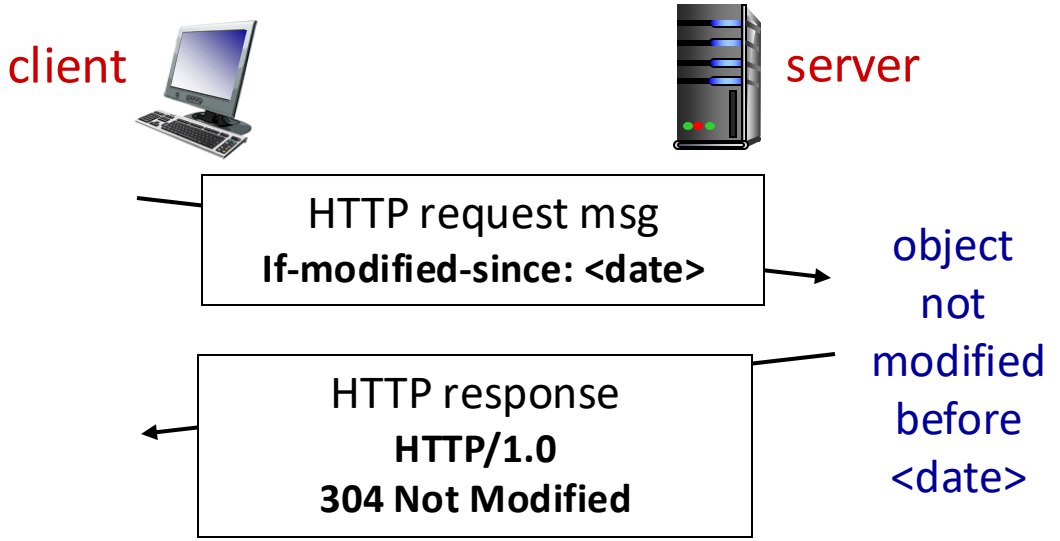
- no object transmission delay (or use of network resources)

- *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



HTTP

- Overview of HTTP
- Request/response message format
- State management
- Caching
- **Request pipelining**

Request Pipelining

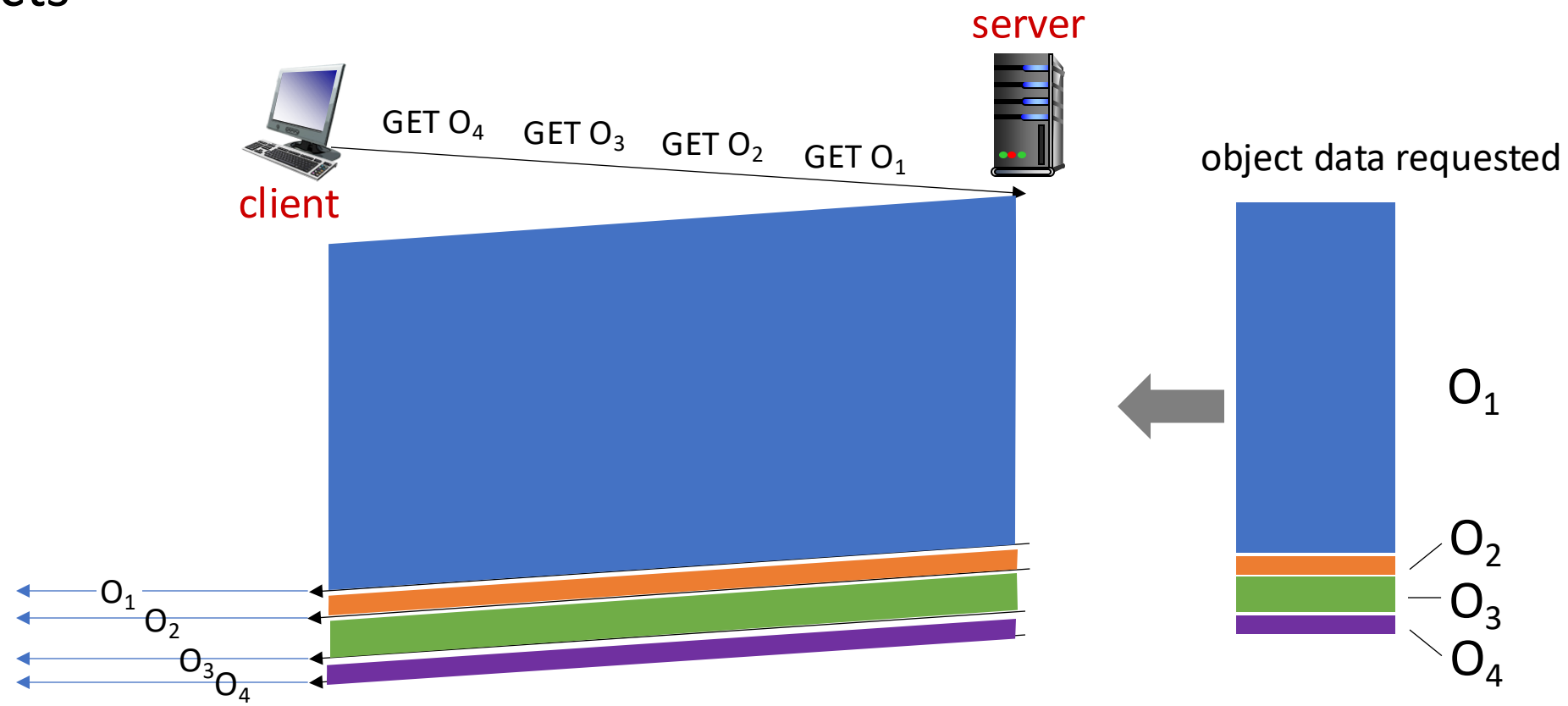
Key goal: decreased delay in multi-object HTTP requests

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 1.1: FCFS Pipelining

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

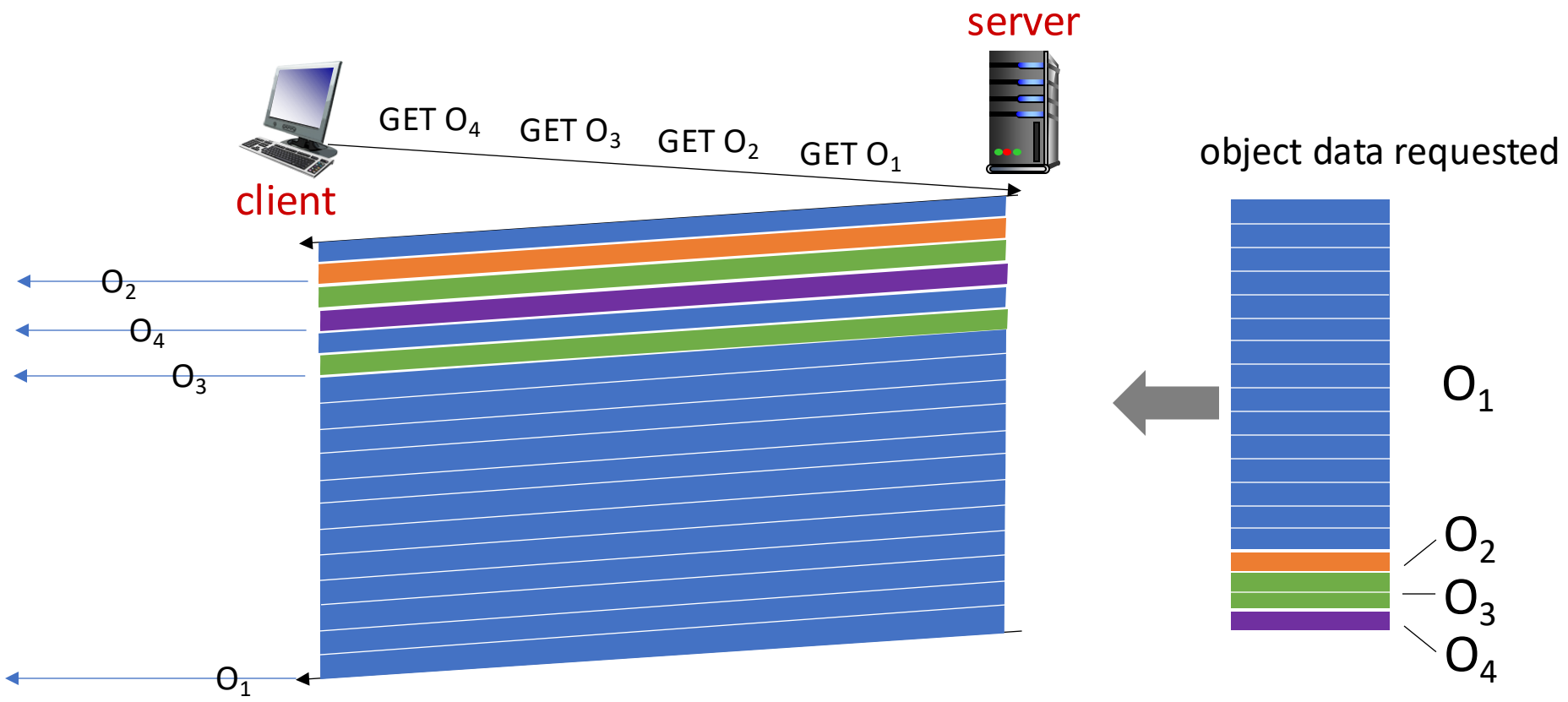
Key goal: decreased delay in multi-object HTTP requests

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

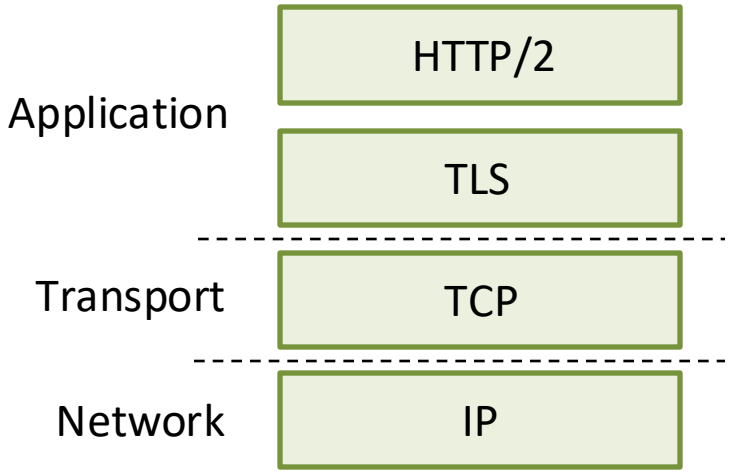


O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HOL blocking due to TCP still there!

QUIC: Quick UDP Internet Connections aka HTTP/3

- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)



HTTP/2 over TCP

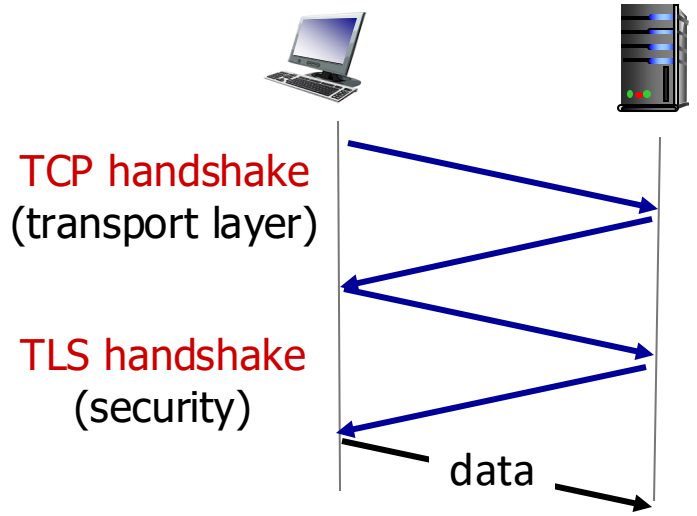
QUIC: Quick UDP Internet Connections

adopts approaches we've studied for connection establishment, error control, congestion control

- **error and congestion control:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well-known TCP ones." [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT

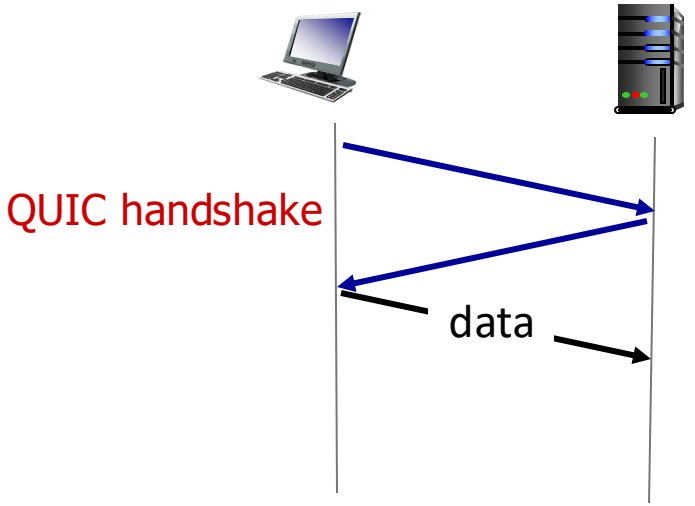
- multiple application-level "streams" multiplexed over single QUIC connection
  - separate reliable data transfer, security
  - common congestion control

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

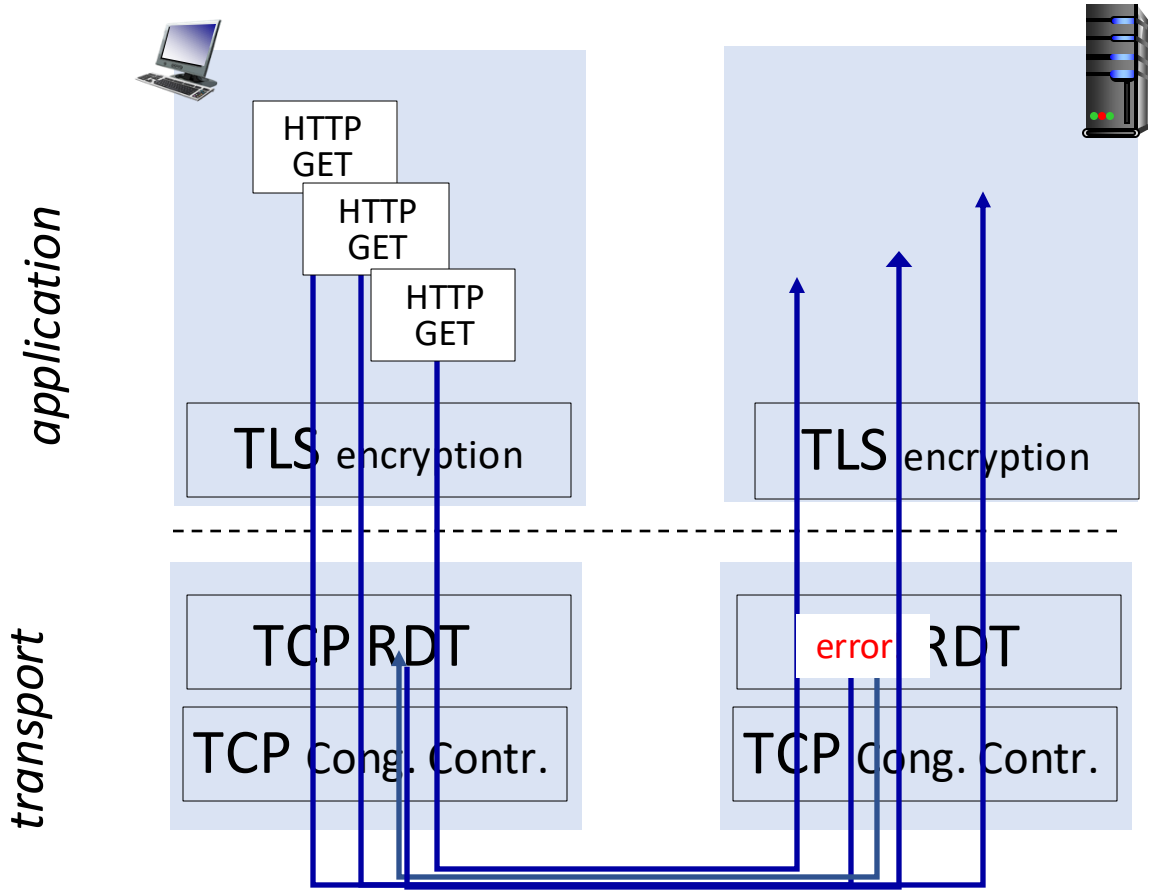
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

QUIC: streams: parallelism, no HOL blocking



(a) HTTP 2.0