Computer Networks
COL 334/672

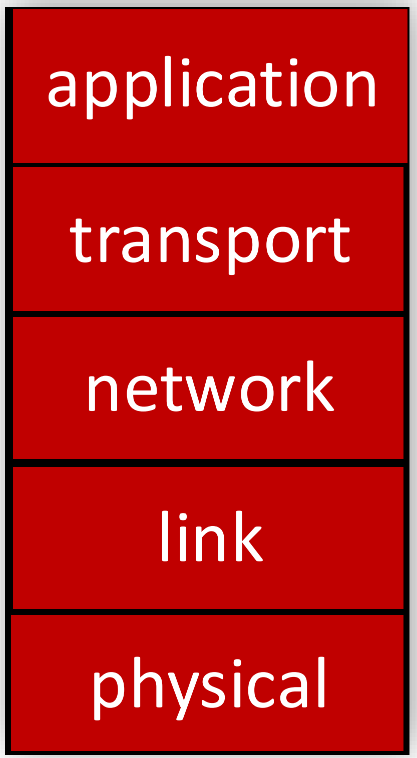
Application Layer

Slides adapted from KR

Sem 1, 2025-26

Internet Layered Architecture

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits “on the wire”



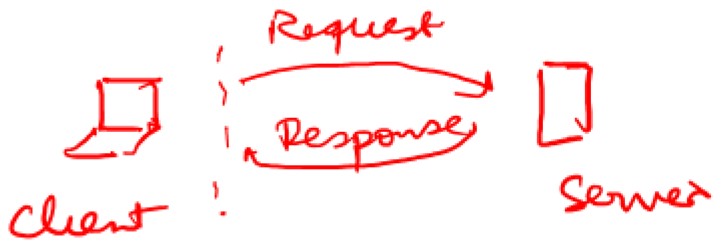
How to design a networked application?

① What is that application : chat application .

② scale : 100 users (low budget)

For N/w communication : Socket APIs → UDP / → TCP socket → reliability

N/w app architecture



[Intermittent connectivity / May be behind a NAT]

Server
→ dedicated machine (always ON)
→ publicly accessible (Public IP, port)

P2P architecture

Peer to peer

Client
Client

(Both are intermittently connected)

↳ Establishing connectivity is challenge (Tough)

↳ No resources / self-scaling n/w

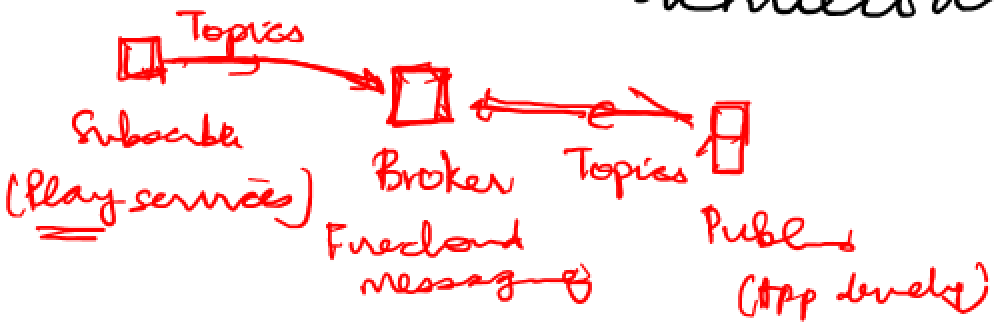
↳ Reliability / Churn

Networked Application Architecture

- Publish subscribe architecture

Android Notification

Publish / Subscribe architecture



What we are going to cover

Killer Applications

- HTTP ②
- Email ①
- DNS → Infrastructure Appn
- P2P ③
- Video Streaming ④

Web and HTTP

Browser downloads base HTML which has info about all other objects

- World Wide Web or Web was the 2nd killer app over the Internet
- Goal of Web: organize and retrieve information over Internet
- Web is based on two sister protocols: HTML and HTTP
- HTML: HyperText Markup Language
  - language to create and structure web page
  - web page consists of *objects,* each of which can be stored on different Web servers
  - base HTML-file which includes several referenced objects, each addressable by a URL

$$\underbrace{\texttt{www.wikipedia.org}}_{\text{host name}}\underbrace{\texttt{/wiki/http}}_{\text{path name}}$$

example.com

HTTP overview

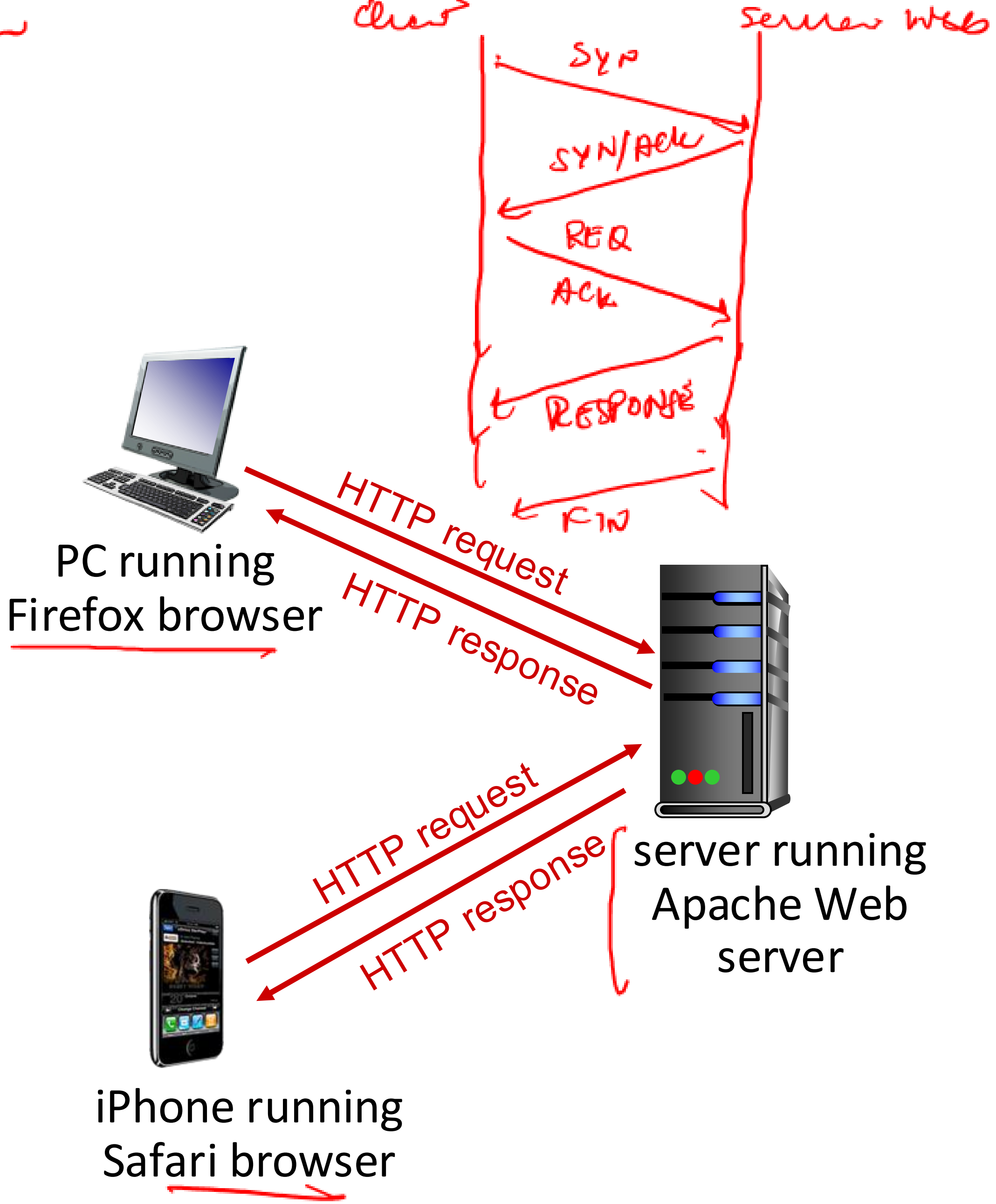
HTTP: hypertext transfer protocol

- Request/response protocol
- Stateless protocol [scalability Simple]
- Which transport?

TCP! Reliability

↳ Congestion control

2 RTTs per object



PC running Firefox browser

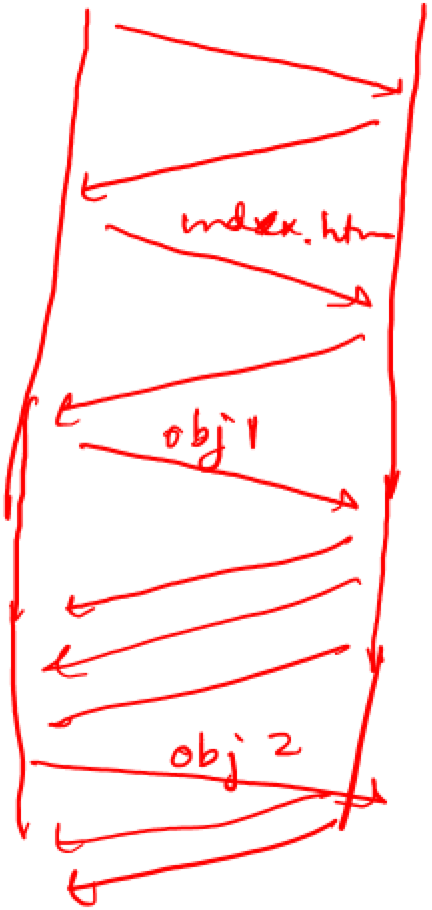
server running Apache Web server

iPhone running Safari browser

Downloading Multiple Objects

10 objects :- 20 RTTs (Non-persistent connection)

11 RTTs (persistent connection)



Request (keep Alive]

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)

HTTP

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

cse.iitd.ac.in/~cs...

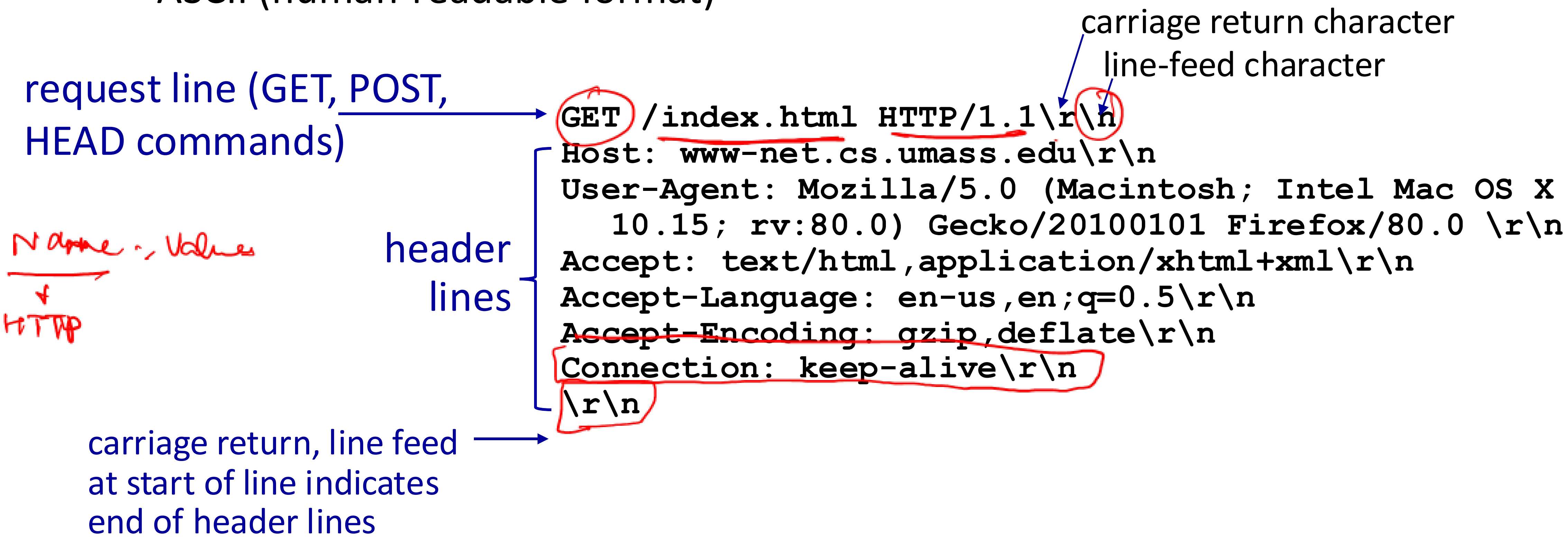
↓ Find server (DNS)

IP address

↓ Download index.html

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)

```
GET /index.html HTTP/1.1\r\n
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
```

header lines

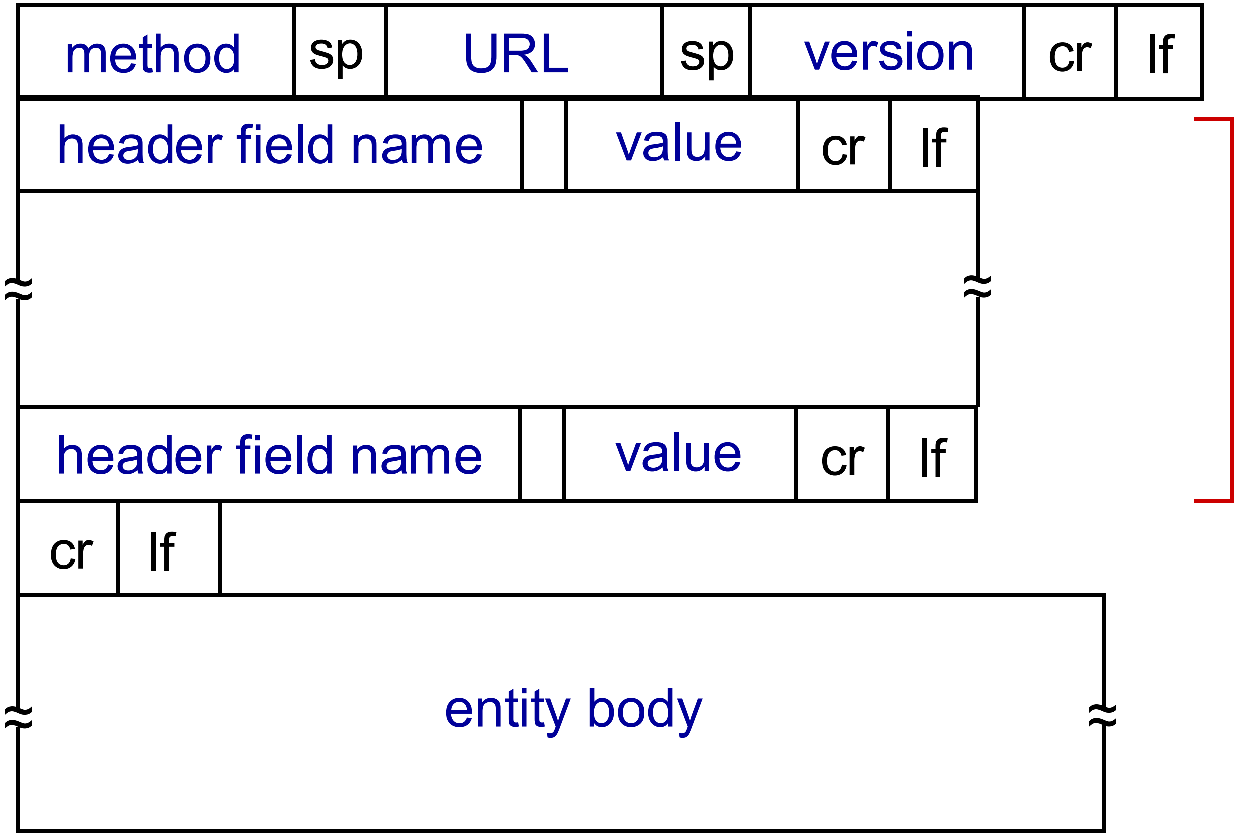
carriage return, line feed at start of line indicates end of header lines

Name : Value

↓

HTTP

HTTP request message: general format



method sp URL sp version cr lf

header field name value cr lf

header field name value cr lf

cr lf

entity body

request line

header lines

body

Other HTTP request messages

post data

POST method:

1

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

2

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:

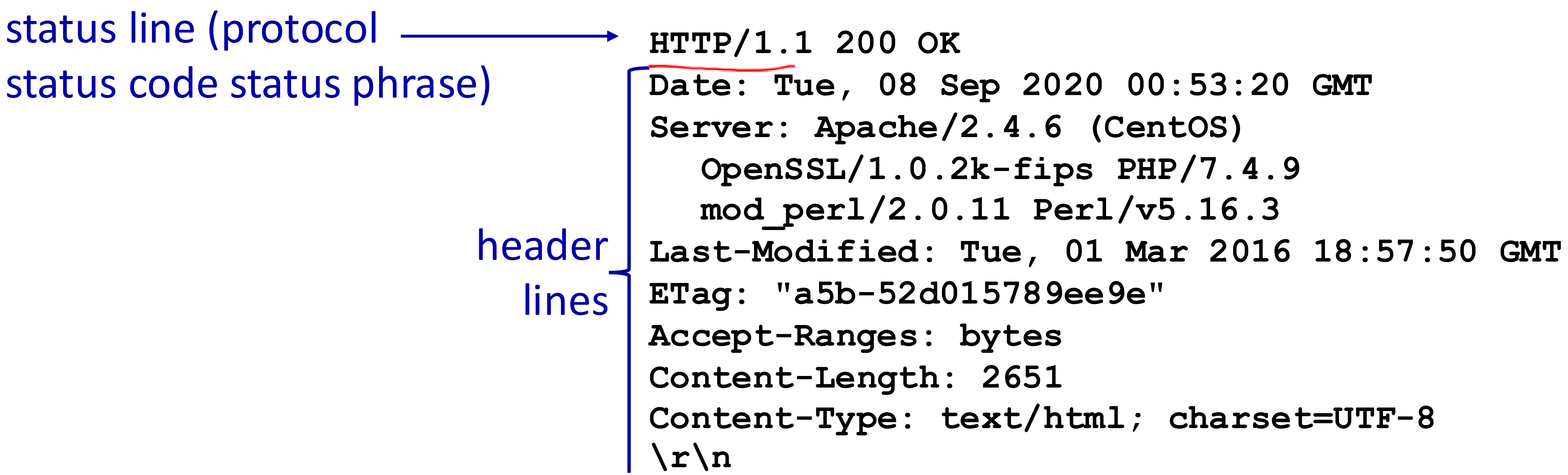
- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

2

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)

header lines

```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2020 00:53:20 GMT
Server: Apache/2.4.6 (CentOS)
   OpenSSL/1.0.2k-fips PHP/7.4.9
   mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8
\r\n
```

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

HTTP Recap

1. HTTP : client-server, request-response protocol

2. HTTP is stateless

Extending HTTP

- **How to remember state?**
- **How to improve performance?**
- **How to scale?**

HTTP is stateless but how does Instagram remember my login?

Cookies are the mechanism

↓

Auxilliary to HTTP → [still stateless]

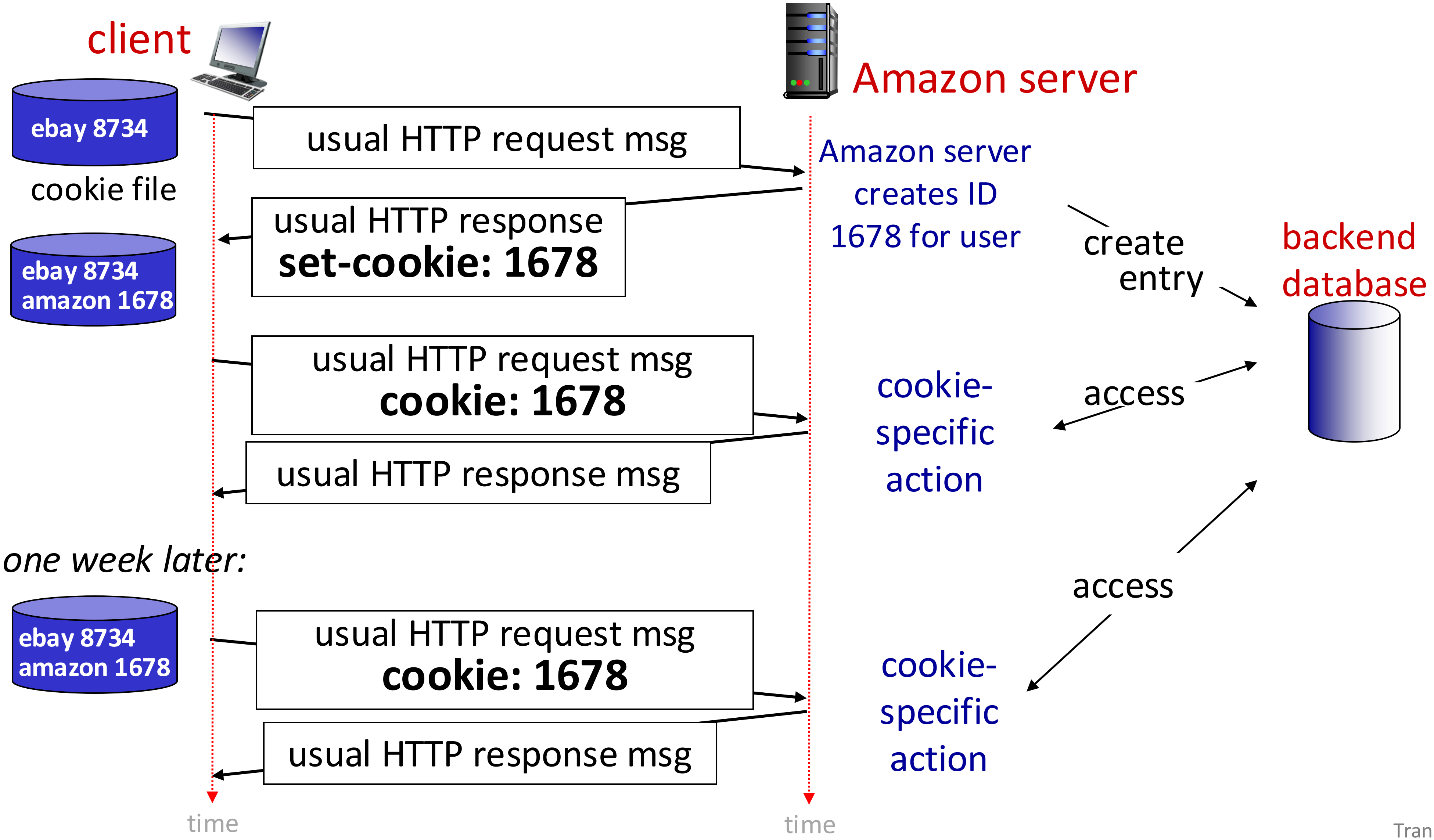
Maintaining user/server state: cookies

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

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

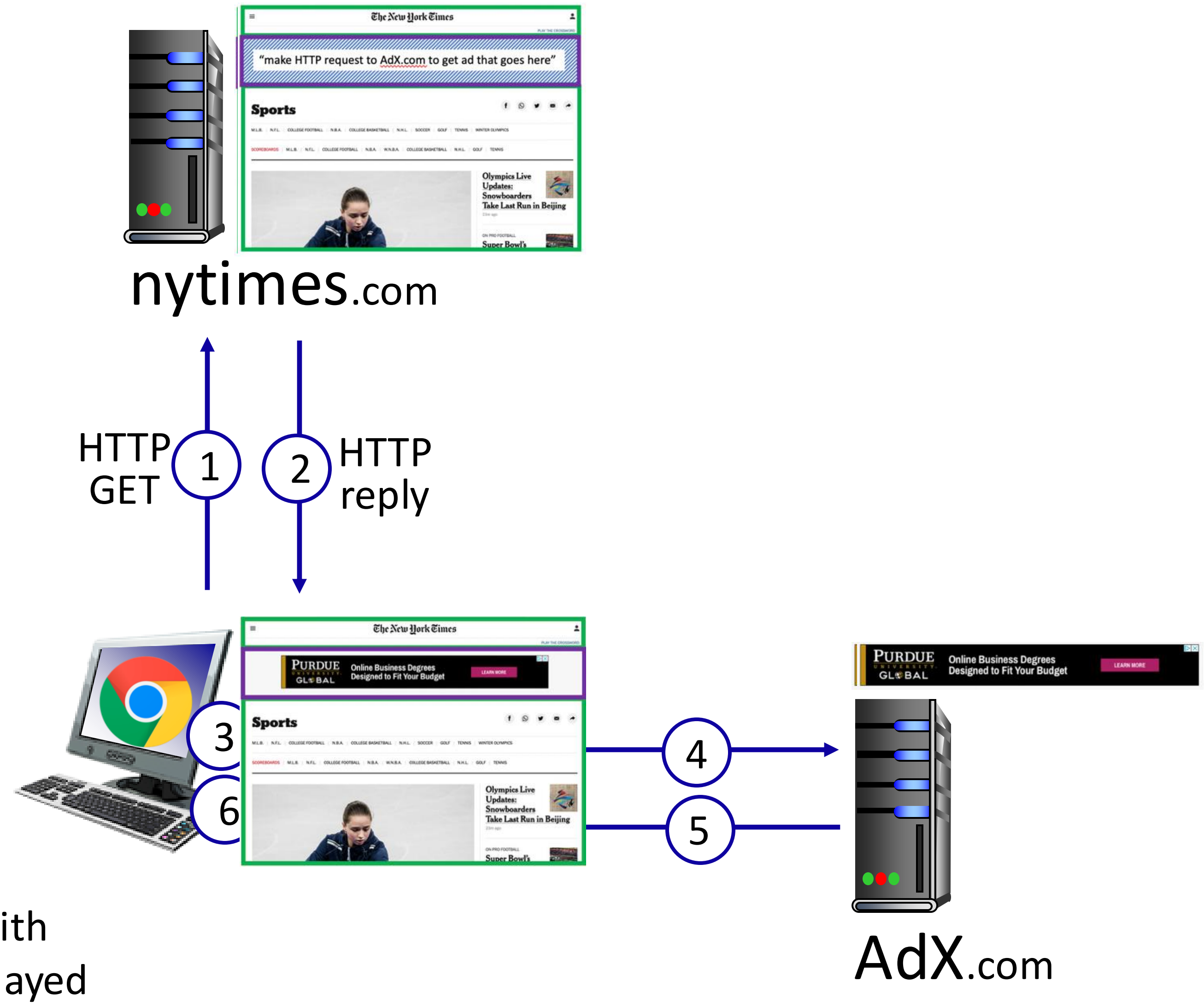
- authorization
- shopping carts
- recommendations (also ads)
- user session state (Web e-mail)

Example: displaying a NY Times web page

1 2 GET base html file from nytimes.com

4 5 fetch ad from AdX.com

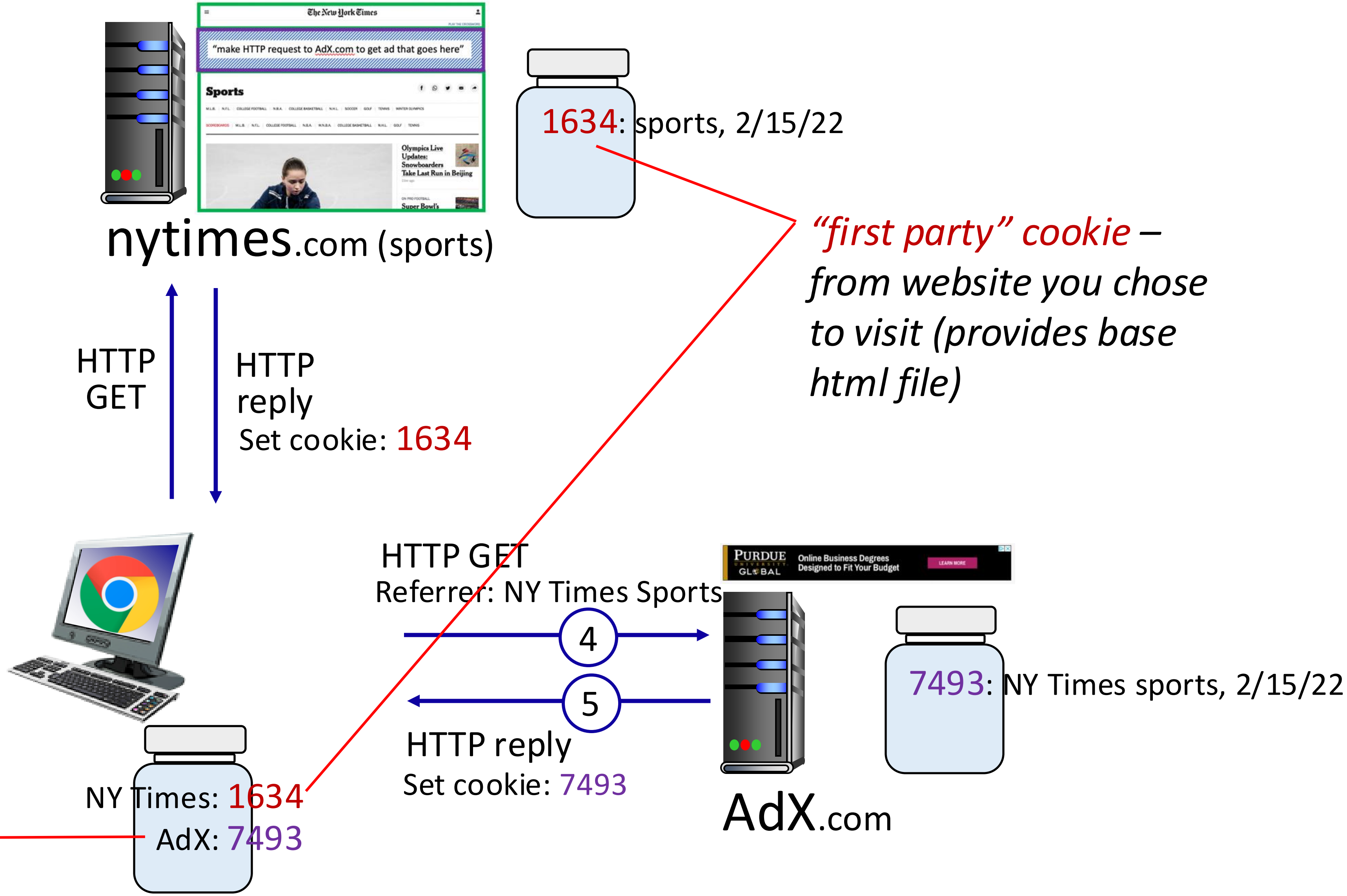
7 display composed page



“make HTTP request to AdX.com to get ad that goes here”

NY times page with embedded ad displayed

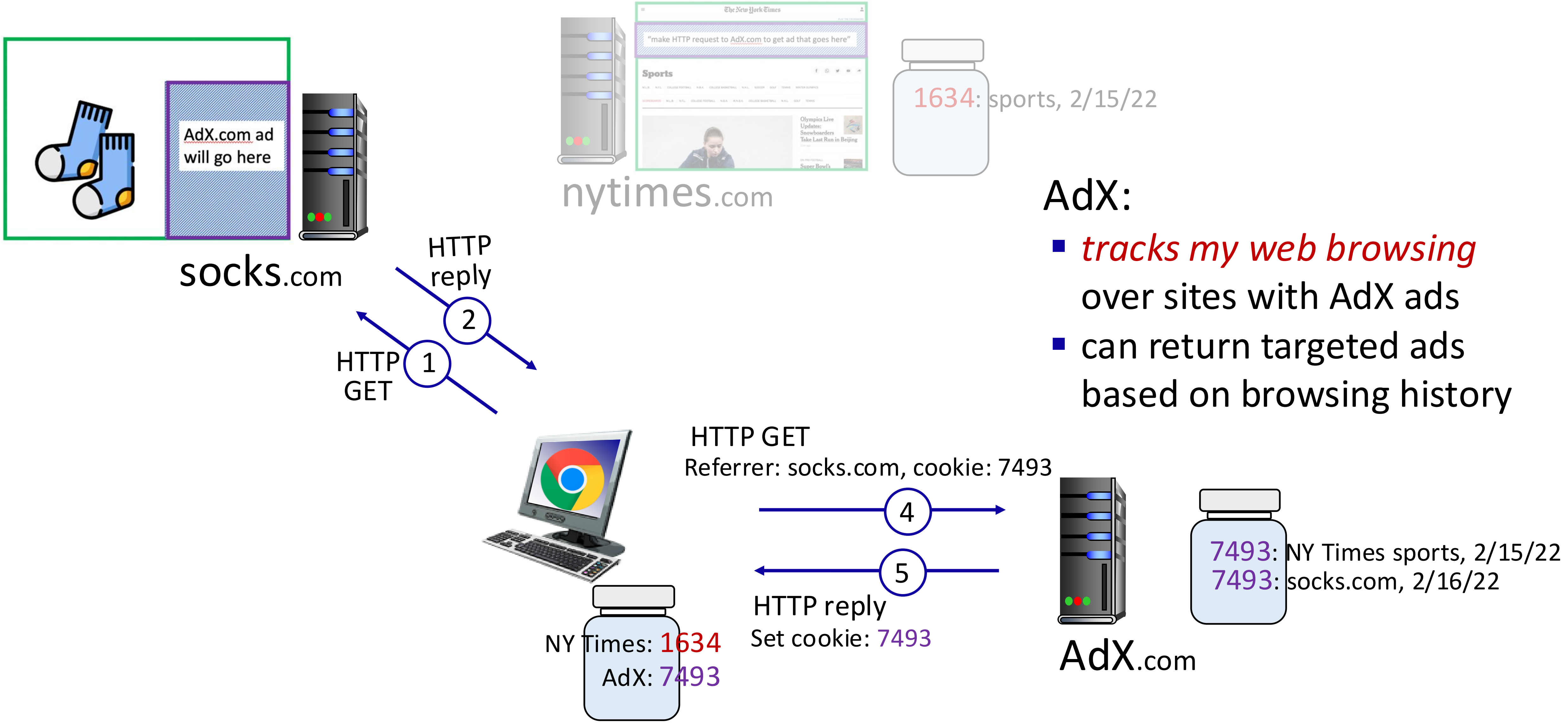
Cookies: tracking a user's browsing behavior


The New York Times
"make HTTP request to AdX.com to get ad that goes here"
Sports
Olympics Live Updates: Snowboarders Take Last Run in Beijing
Super Bowl's
nytimes.com (sports)
1634: sports, 2/15/22
HTTP GET
HTTP reply
Set cookie: 1634
HTTP GET
Referrer: NY Times Sports
4
5
HTTP reply
Set cookie: 7493
PURDUE UNIVERSITY GLOBAL
Online Business Degrees Designed to Fit Your Budget
LEARN MORE
7493: NY Times sports, 2/15/22
AdX.com
NY Times: 1634
AdX: 7493

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

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

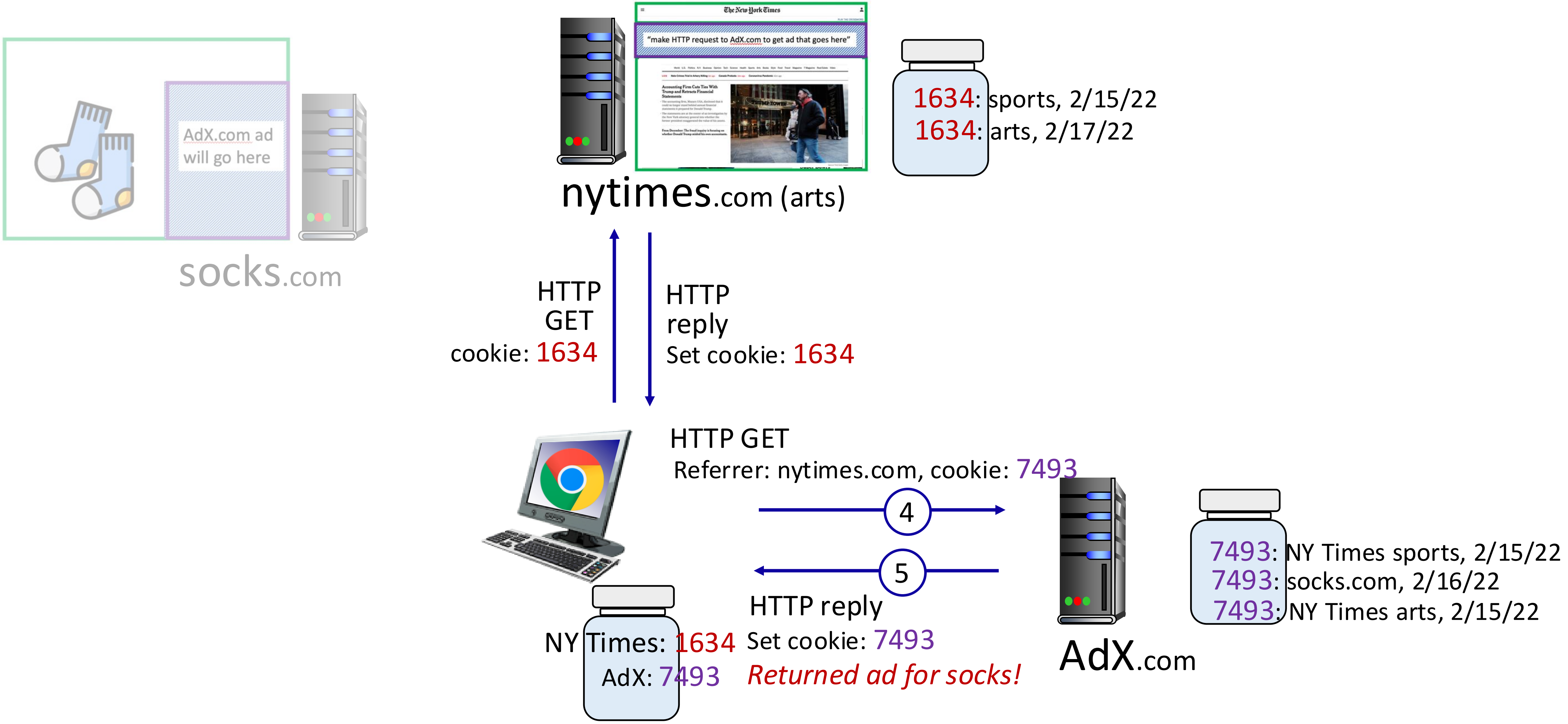
Cookies: tracking a user’s browsing behavior



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)



AdX.com ad will go here

socks.com

"make HTTP request to AdX.com to get ad that goes here"

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:

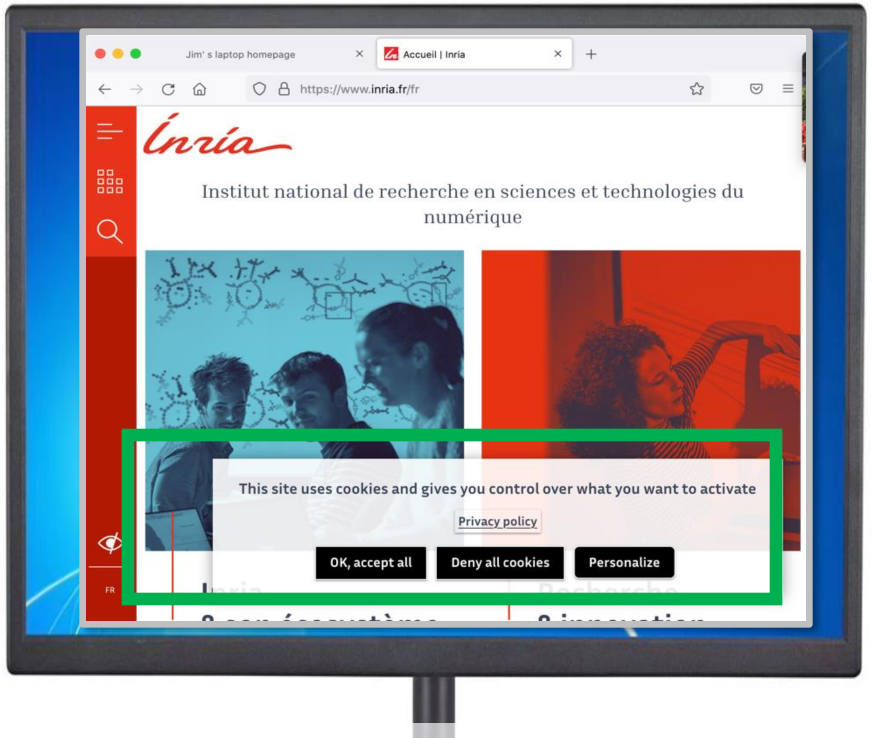
- track user behavior on a given website (first party cookies)
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
  - rather than displayed ad triggering HTTP GET to tracker, could be an invisible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

Data protection laws and cookies

> “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

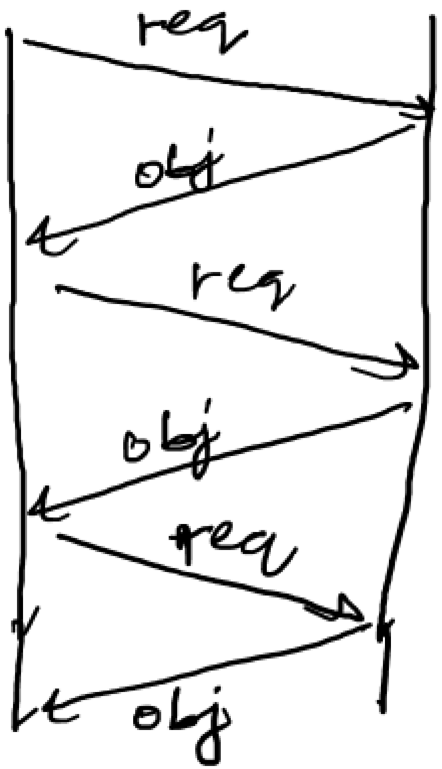


Institut national de recherche en sciences et technologies du numérique

This site uses cookies and gives you control over what you want to activate

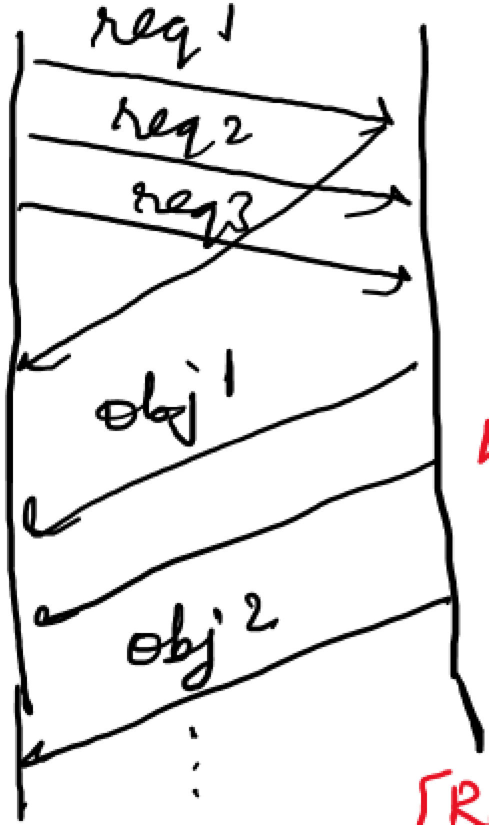
User has explicit control over whether or not cookies are allowed

Improving HTTP



serial download
(Inefficient)

→



request pipelining
(allowed in HTTP 1.1)

HOL blocking
what if
obj 1 is
too big
↑
[Response sent
FCFS]
↑

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

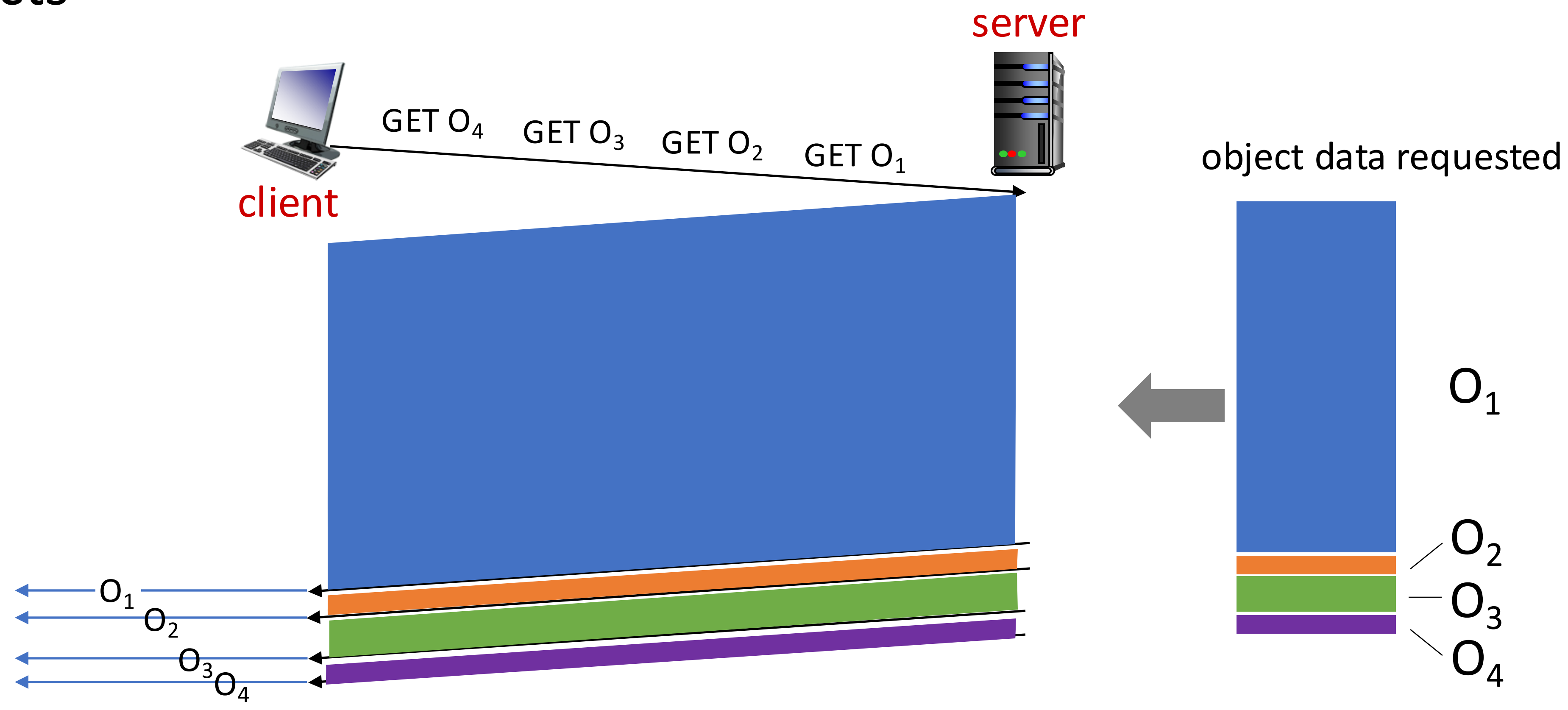
Key goal: decreased delay in multi-object HTTP requests

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



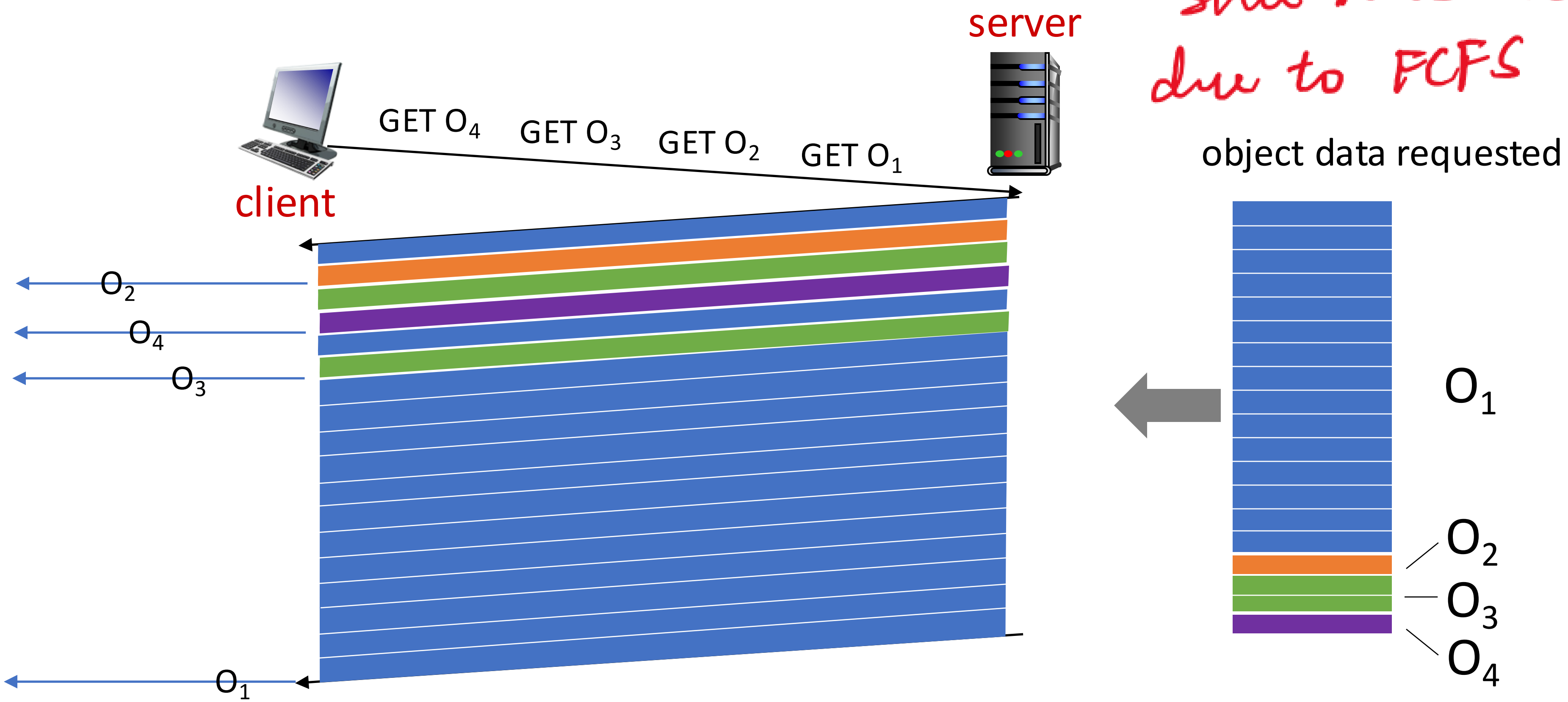
objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

↦ QUIC (or HTTP/3) over UDP [Mitigates TCP HOL blocking]

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved

still has HOL blocking due to FCFS in TCP



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

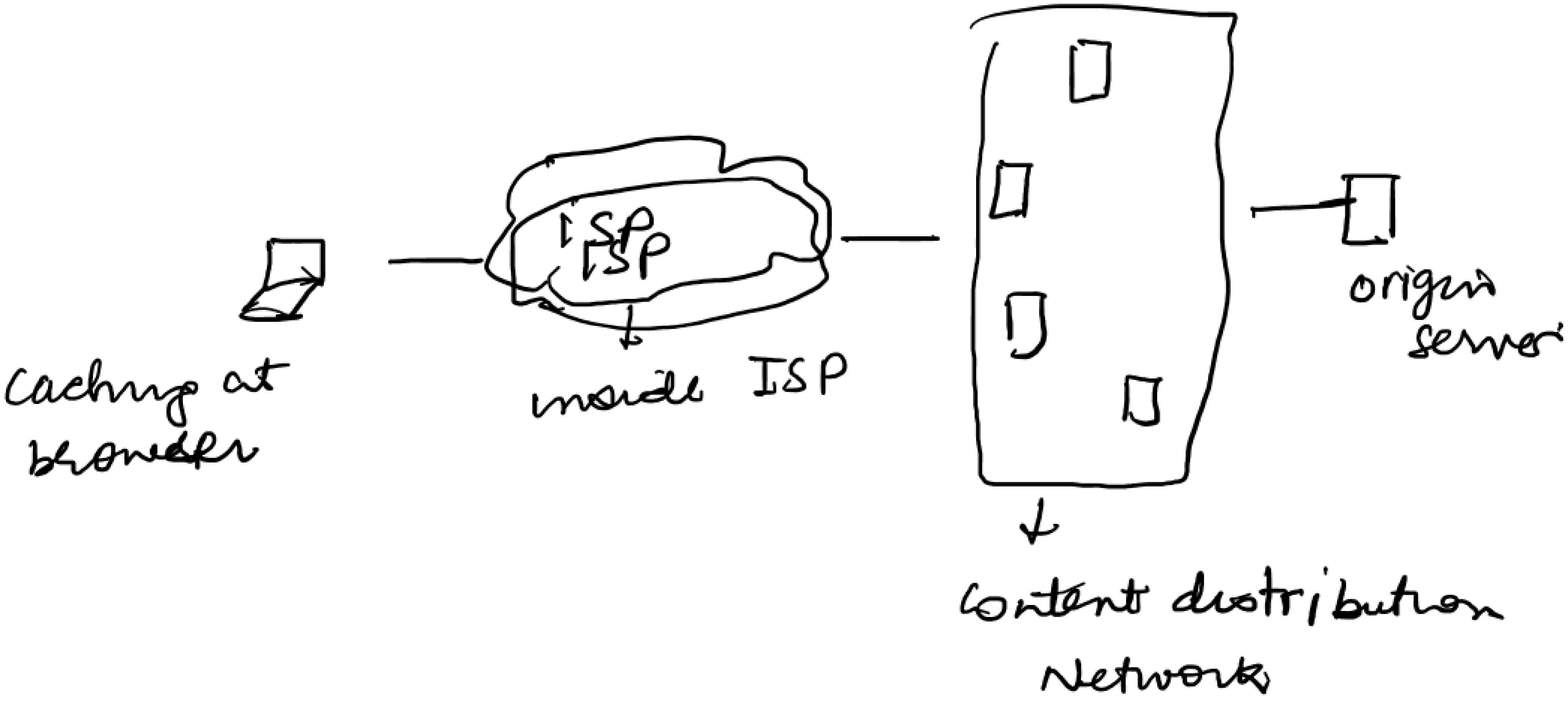
Improving HTTP

Goal: Scale content distribution

Caching: satisfy client requests without involving origin server

[A single server does not scale]

Where in the network path can caching occur?



Browser caching: Conditional GET

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

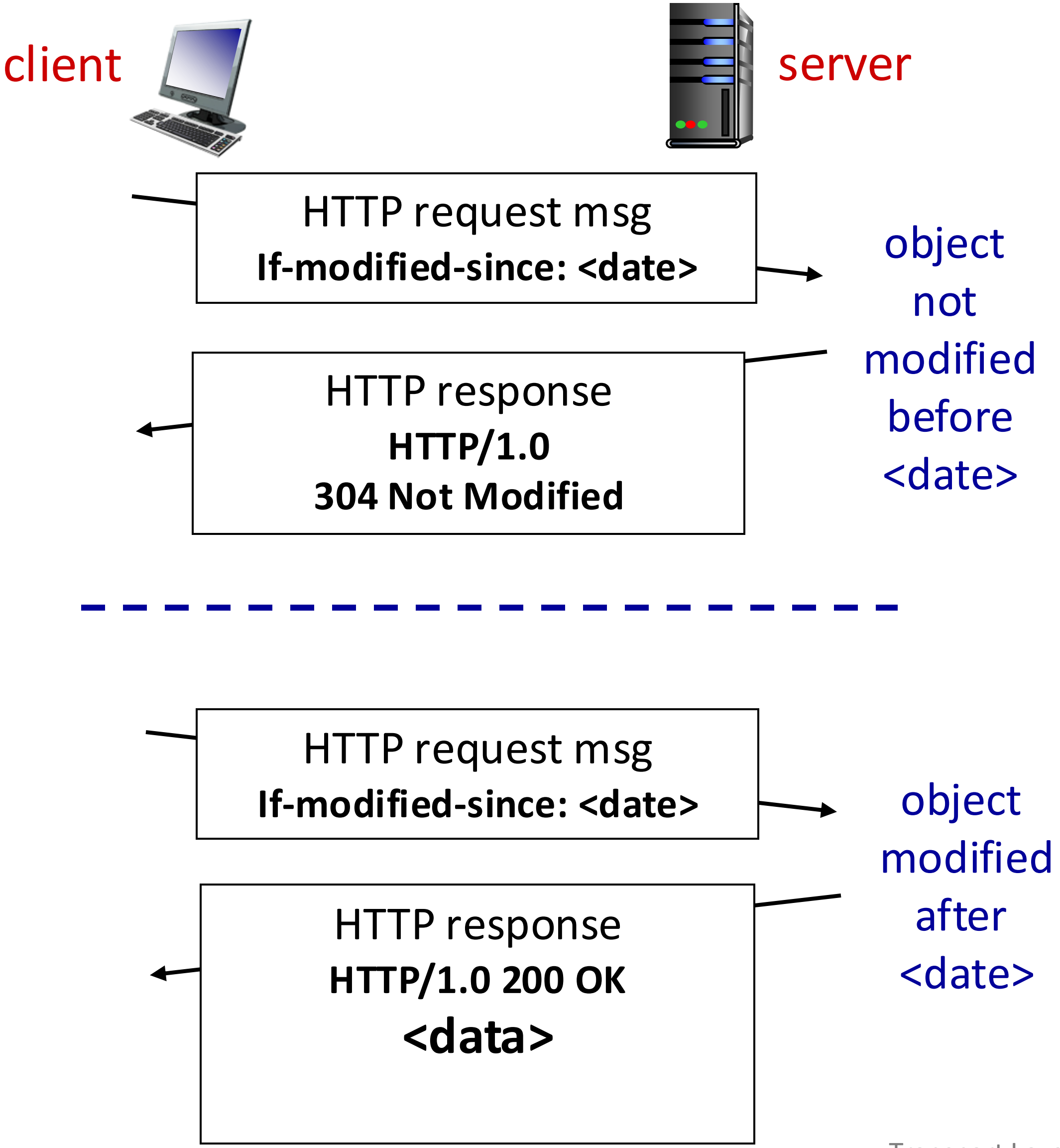
  - no object transmission delay (or use of network resources)

- *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

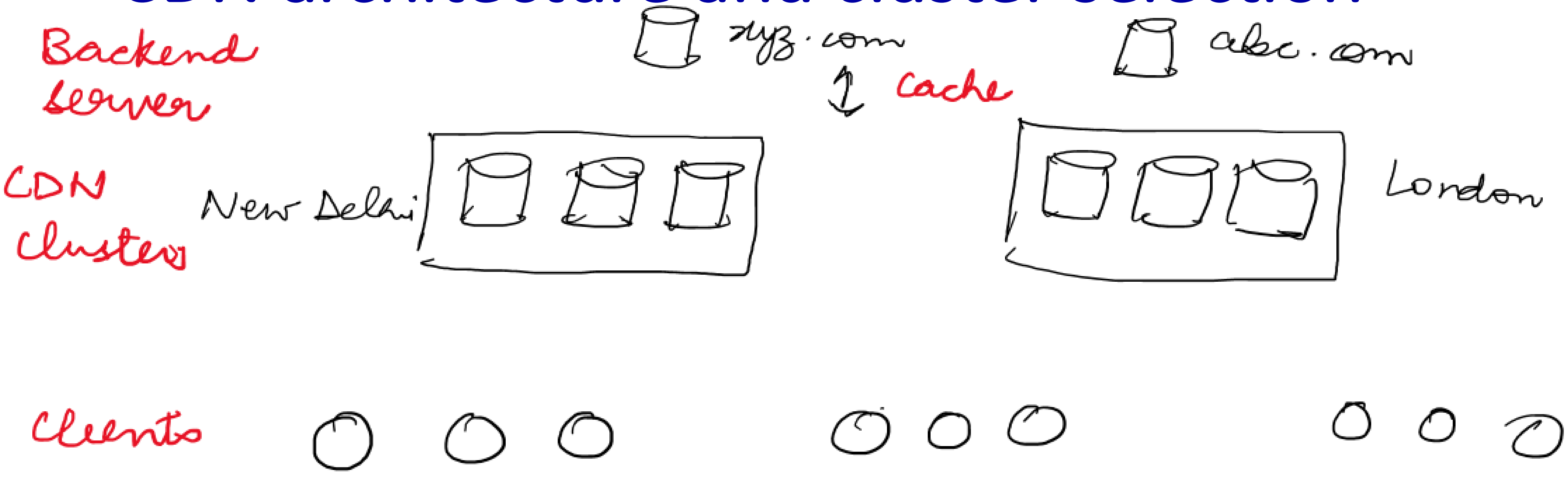


Content distribution networks (CDNs)

- CDN: geographically distribute collection of *server surrogates*
- Servers can be leased by many customers
- Popular CDNs: Limelight, Akamai, Level3
- Two kinds of server placement policies:

- *enter deep:* push CDN servers deep into many access networks
  - close to users
  - Akamai: 240,000 servers deployed in > 120 countries (2015)
- *bring home:* smaller number (10’s) of larger clusters in POPs near access nets
  - used by Limelight

CDN architecture and cluster selection

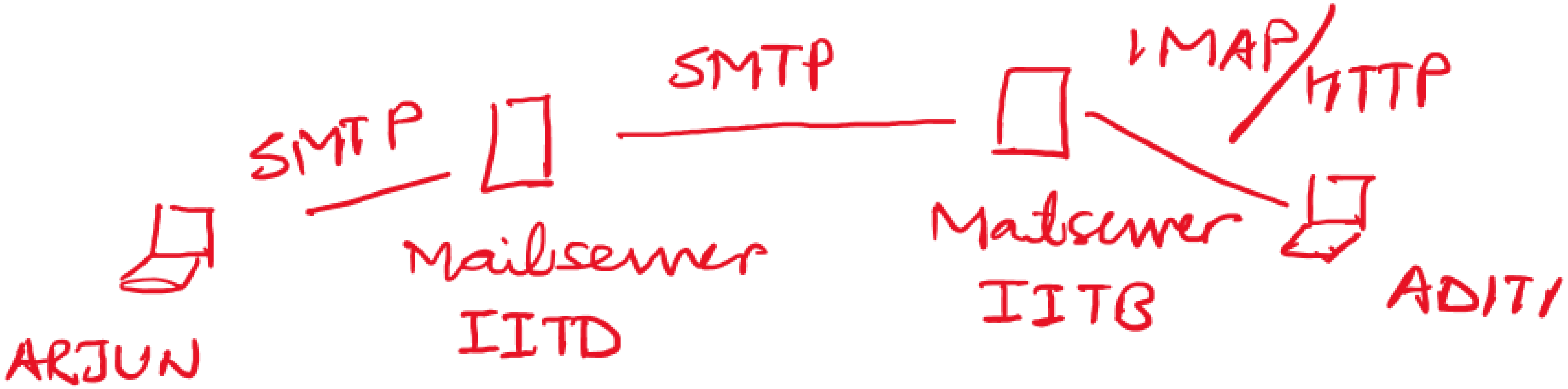


Policies and mechanisms for routing a client request to a cluster?

Journey of an Email

- Scenario: Arjun from IITD wants to send an email to his friend Aditi in IITB

What are the major components involved in email application?



Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

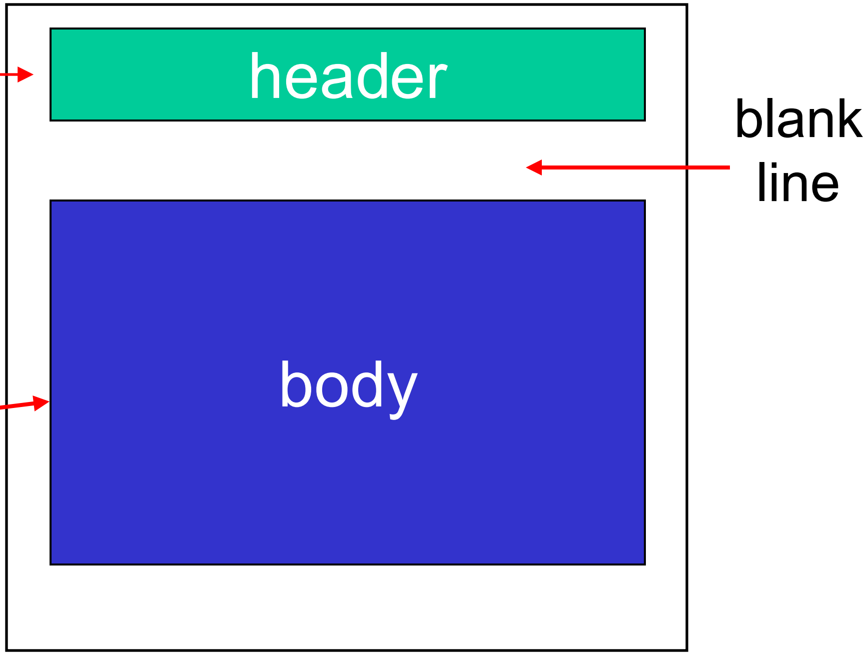
Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

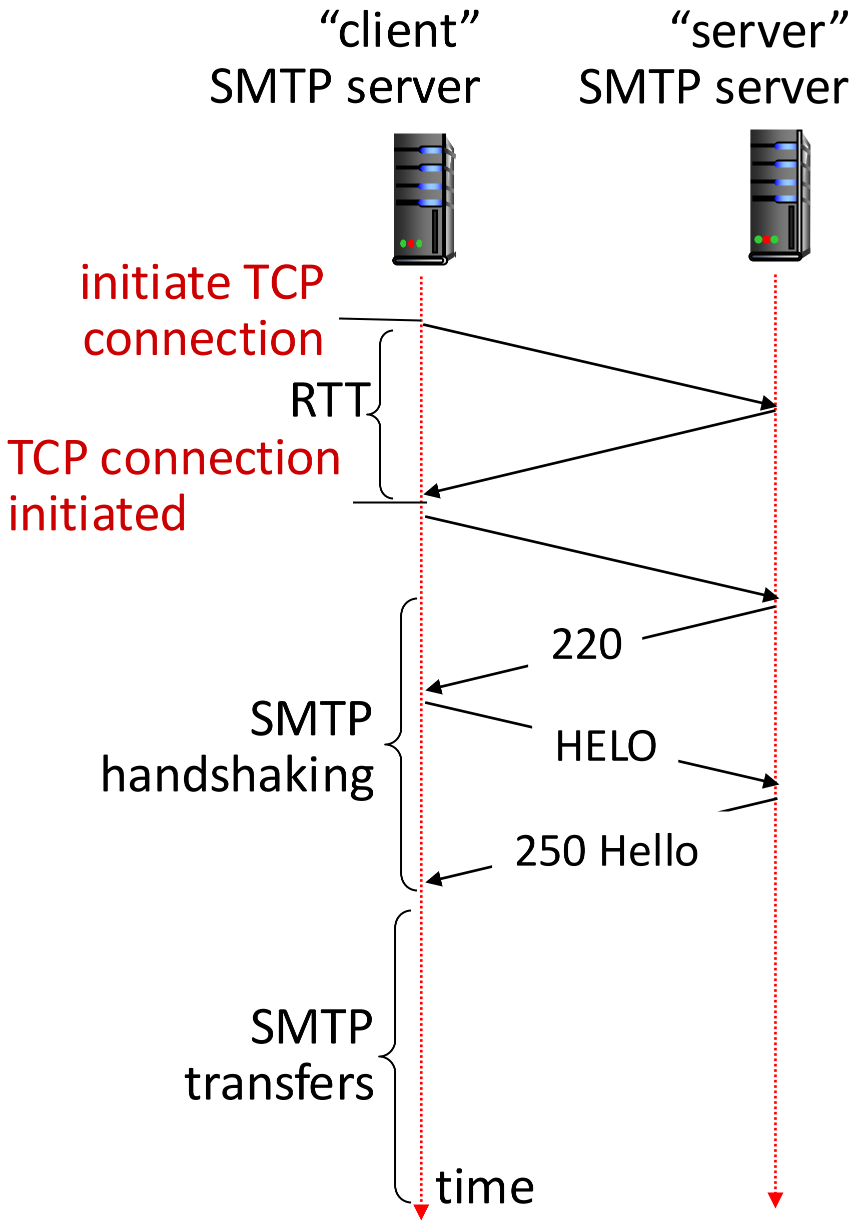
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the “message” , ASCII characters only



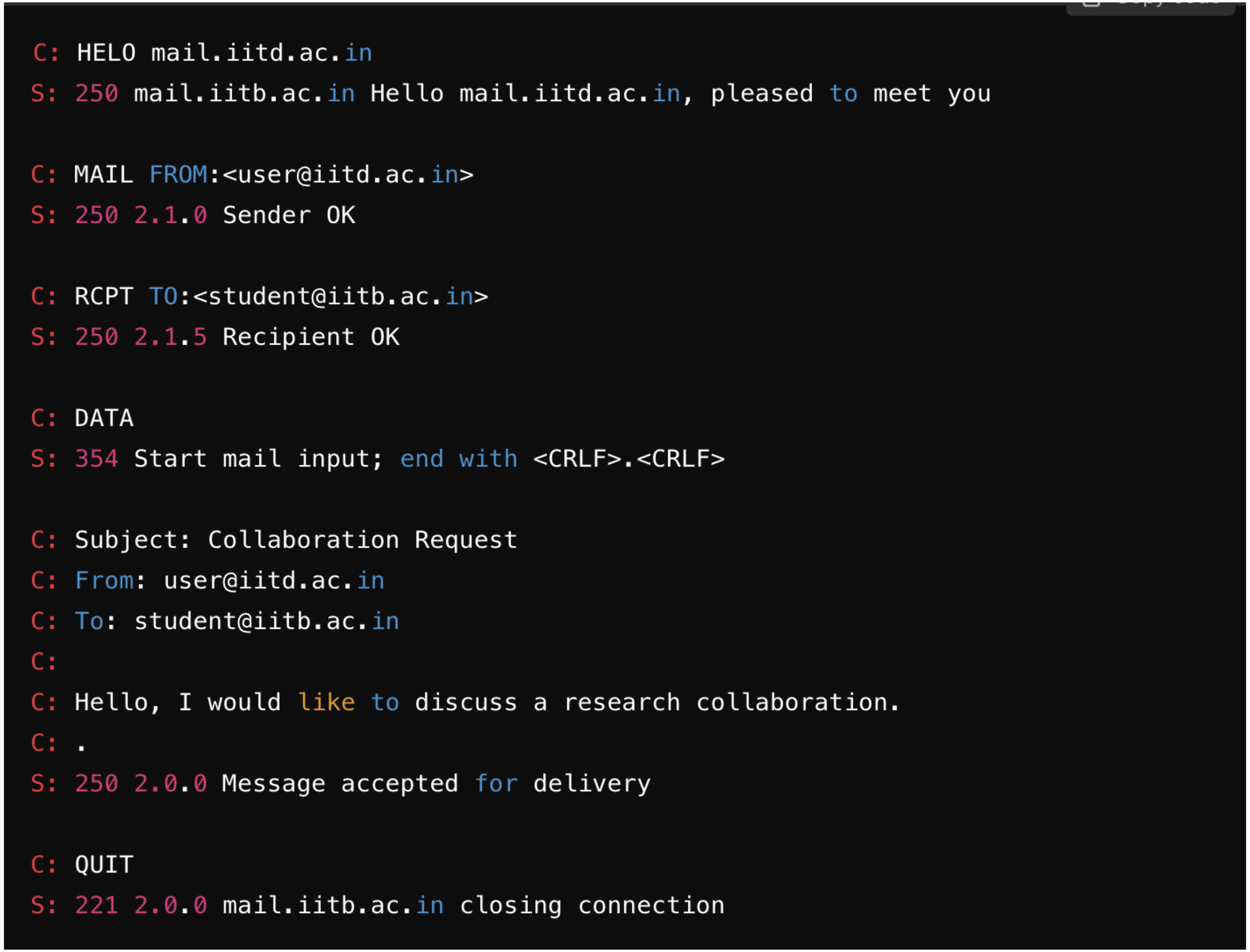
SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase



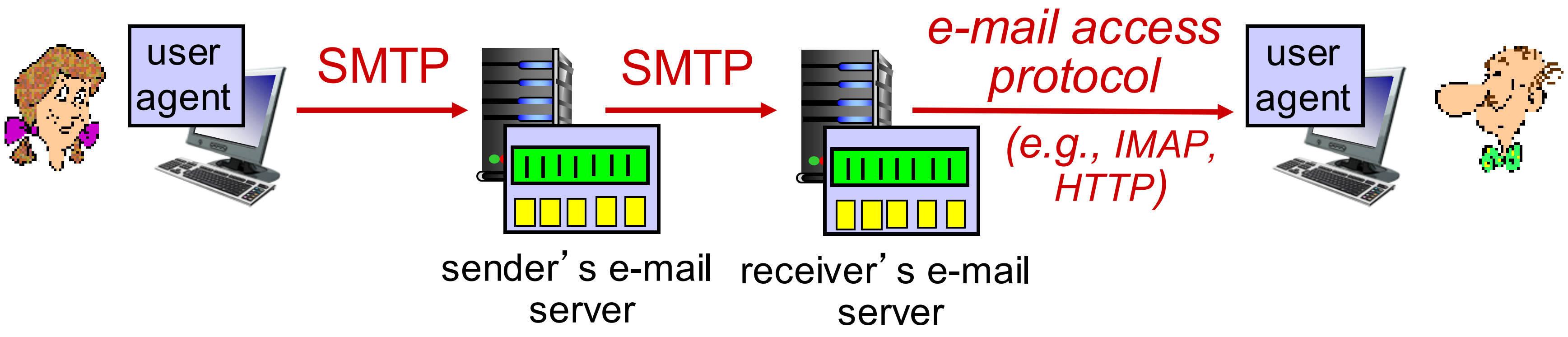
Sample SMTP Interaction

```
S: 220 mail.iitb.ac.in
```



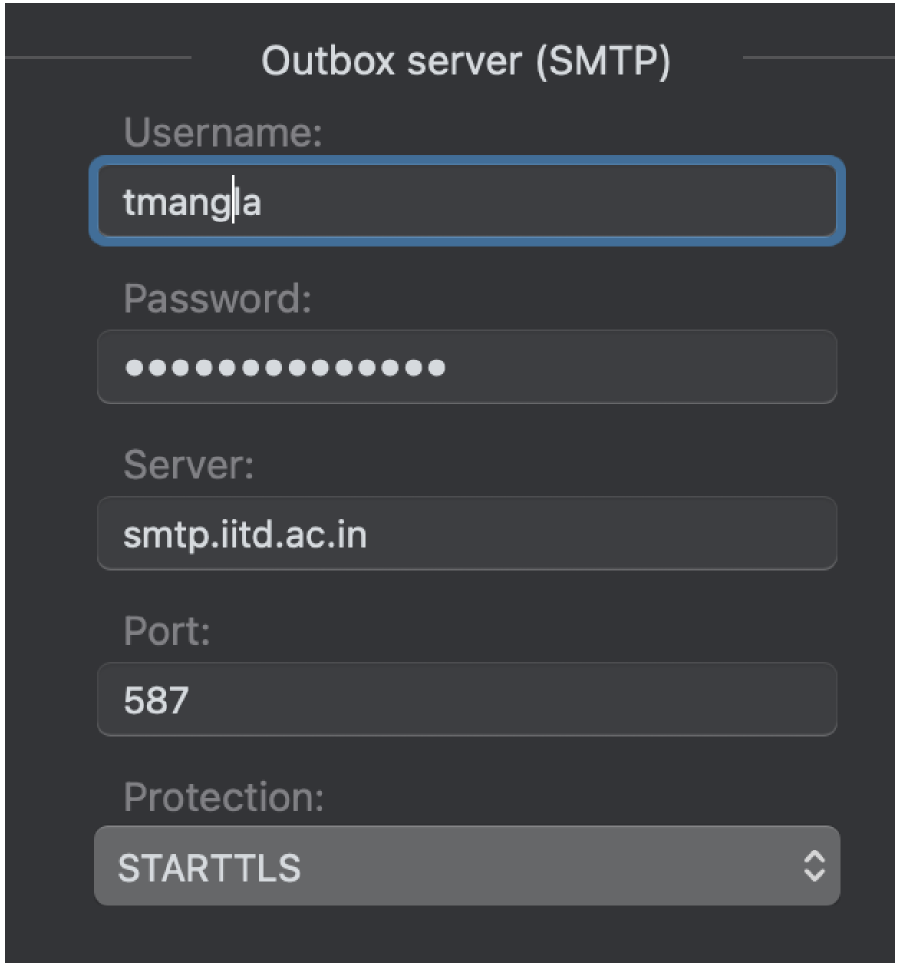
```
C: HELO mail.iitd.ac.in
S: 250 mail.iitb.ac.in Hello mail.iitd.ac.in, pleased to meet you

C: MAIL FROM:<user@iitd.ac.in>
S: 250 2.1.0 Sender OK

C: RCPT TO:<student@iitb.ac.in>
S: 250 2.1.5 Recipient OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>

C: Subject: Collaboration Request
C: From: user@iitd.ac.in
C: To: student@iitb.ac.in
C:
C: Hello, I would like to discuss a research collaboration.
C: .
S: 250 2.0.0 Message accepted for delivery

C: QUIT
S: 221 2.0.0 mail.iitb.ac.in closing connection
```

Retrieving email: mail access protocols



- SMTP: delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

Configuring Mailbox



Outbox server (SMTP)

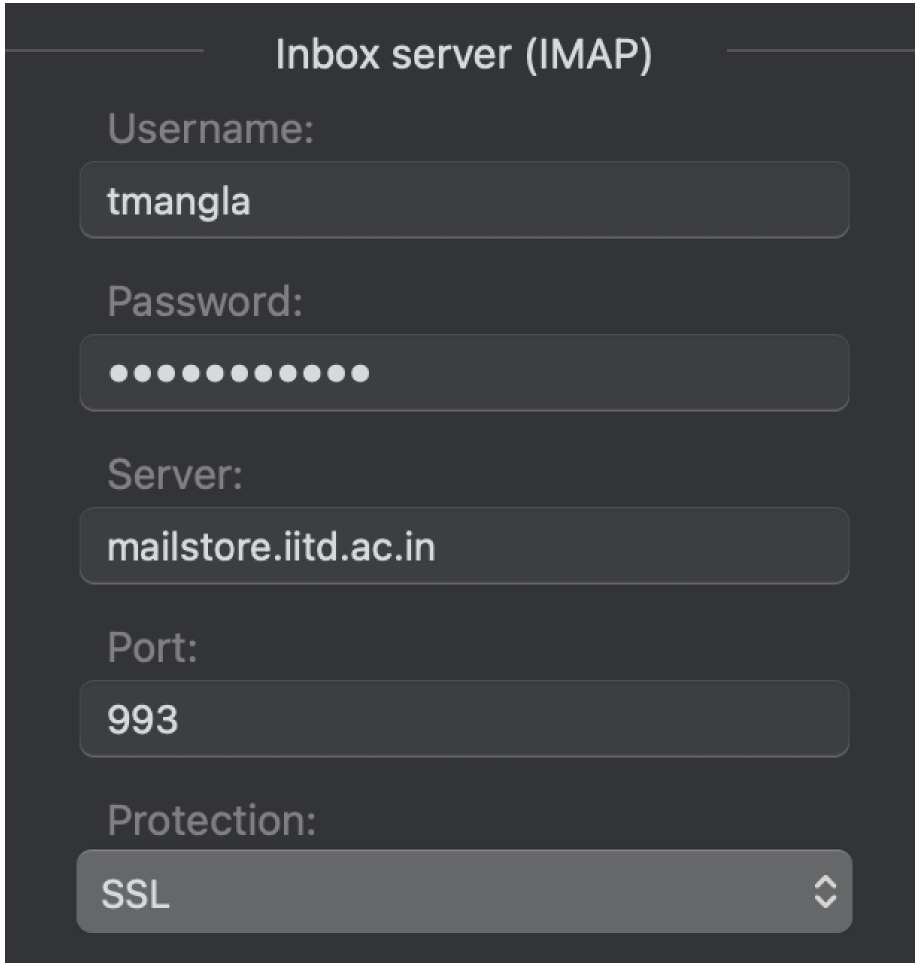
Username: tmangla

Password: ••••••••••••••

Server: smtp.iitd.ac.in

Port: 587

Protection: STARTTLS



Inbox server (IMAP)

Username: tmangla

Password: ••••••••••••

Server: mailstore.iitd.ac.in

Port: 993

Protection: SSL

SMTP: observations

comparison with HTTP:

- HTTP: client pull
- SMTP: client push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

SMTP/Email:

- Open standards
- Interoperability among email clients
- Exemplify the design spirit of Internet