# Computer Networks (CS F303)


Virendra Singh Shekhawat
Department of Computer Science and Information Systems

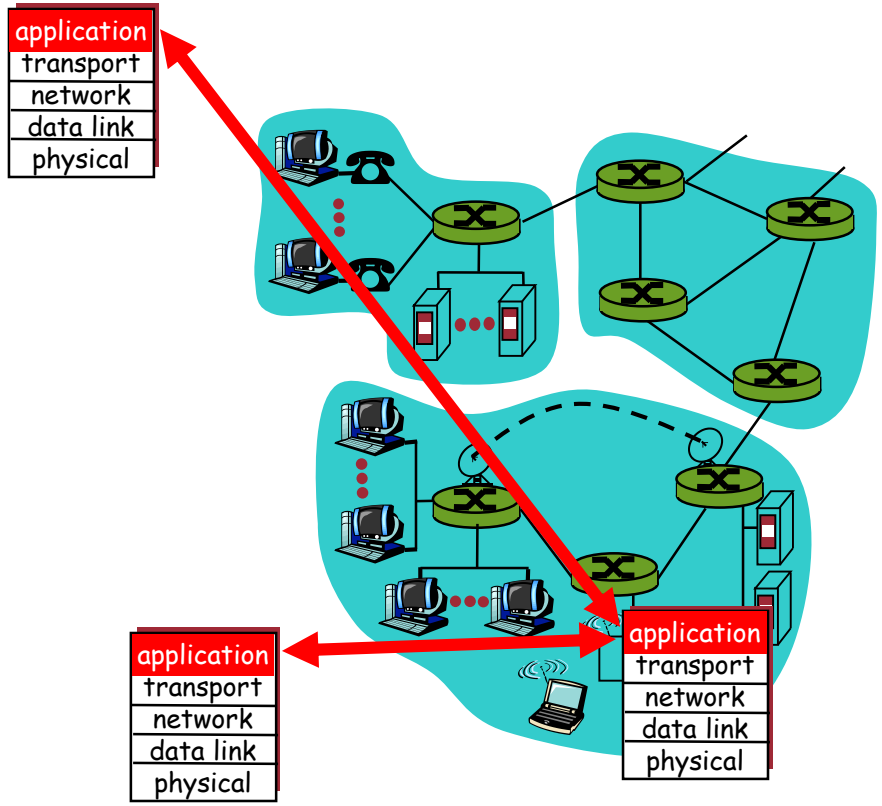
BITS Pilani
Pilani Campus

# Application Layer

- Network Application Architectures
  - Client-Server, Peer-to-Peer, Hybrid (CS+P2P)
- Client Server Applications
  - Web Browser/Web Server, File Transfer, Email, Telnet.
- Peer-to-Peer Applications
  - BitTorrent, eDonkey, Emule, Gnutella etc.
- Hybrid Applications
  - Skype
- Cloud Based and CDN Based Applications
  - YouTube, MS Teams, Zoom, Google Meet, etc.
- Internet Directory Structure
  - DNS Protocol

# What is a Network Application?

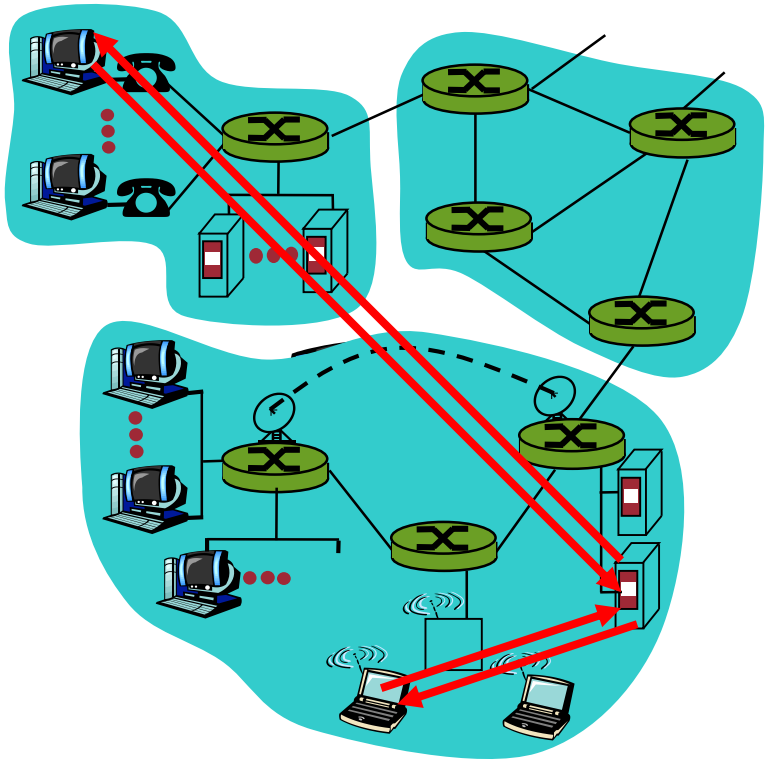
- Programs that run on different end systems and communicate over a network
  - e.g., Web: Web server app. communicates with browser app.
- Network core devices do not run user application code
- Application on end systems allows for rapid application development



# Application architectures

- Client-server
- Peer-to-Peer (P2P)
- Hybrid of client-server and P2P

# Client-Server Architecture



## Server:

- "always-on" host
- Permanent IP address
- For scaling, data center is used to create large powerful virtual server

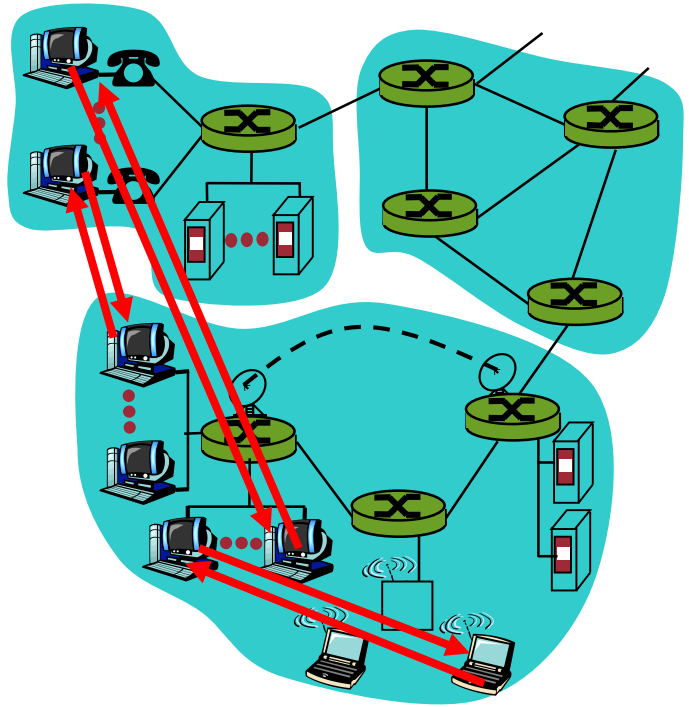
## Clients:

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

# Pure P2P Architecture

- No “always-on” server
- Arbitrary end systems directly communicate
- Peers are connected and change IP addresses
  - Example: Freenet and BitTorrent (File Sharing Apps)

Highly scalable but difficult to manage!!!



# Hybrid of CS and P2P



## Skype

- Internet telephony application
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

## Instant messaging

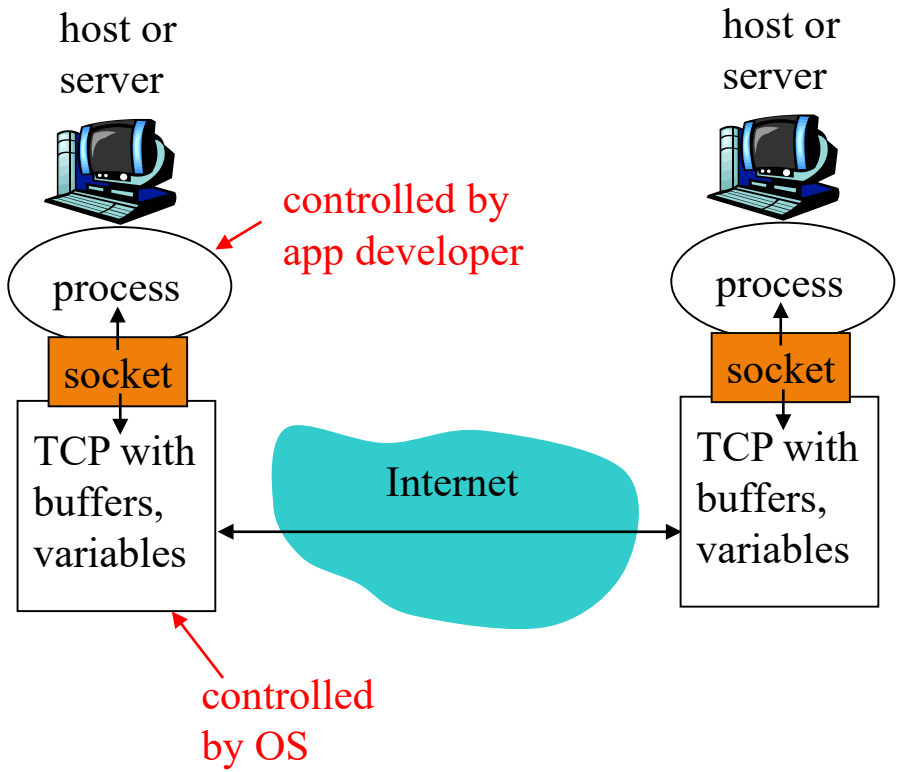
- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

# How do Network Applications Communicate?

- **Process** sends/receives messages to/from its **Socket**
  - **Socket** is the interface between the application layer and the transport layer within the host

- Within same host, two **processes** communicate using **inter-process communication**

- **Processes** in different hosts communicate by exchanging **messages**

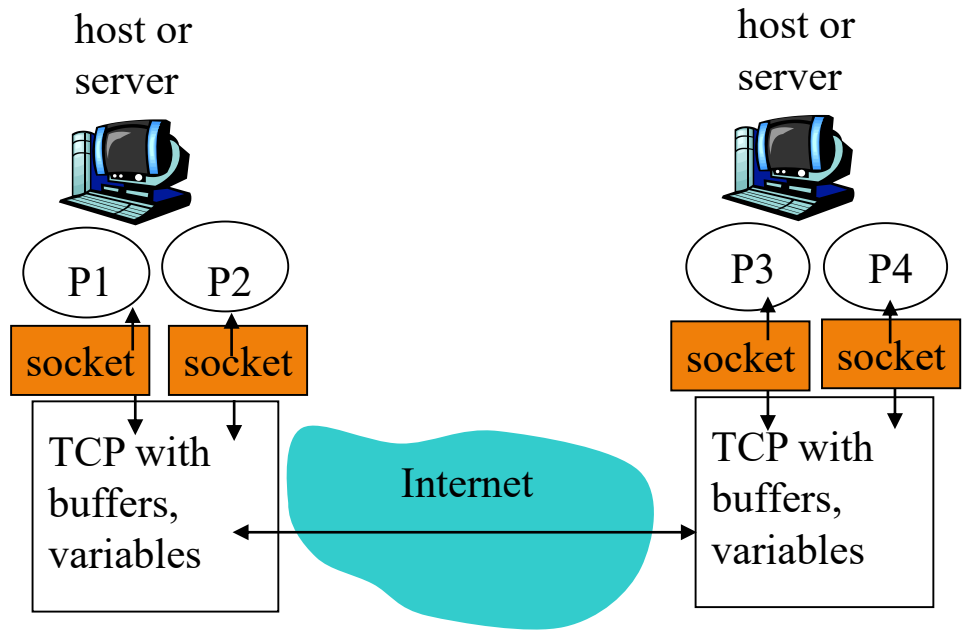


# How to identify a process running on a machine?

- To receive messages, process must have *identifier*

- Is IP address of host sufficient for identifying the process?

- *Process identifier* = IP address + port number
  - e.g., HTTP server: 80, Mail server (SMTP): 25
  - List of well known port numbers is available at **http://www.iana.org**



# What transport service does an app need?



- Data loss
  - Some apps (e.g., audio, video) can tolerate some loss
  - Other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- Bandwidth
  - Some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
  - Other apps ("elastic apps") make use of whatever bandwidth they get
  - ex. E-mail, File Transfer
- Timing
  - Some apps (e.g., Internet telephony, interactive games) require low **delay** to be "effective"

# Web and HTTP [1994]

## Web page consists of objects

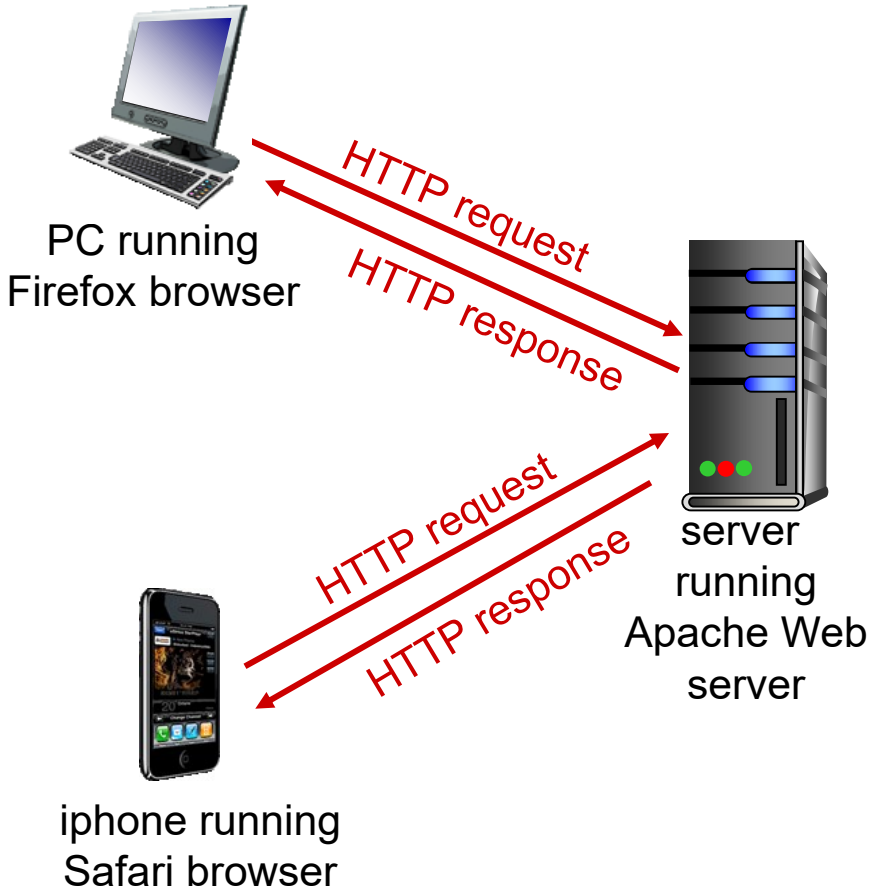
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URLs:

https://www.bits-pilani.ac.in/pilani/computerscience/ProgrammesOffered

https://www.bits-pilani.ac.in/pilani/computerscience/Faculty

# HTTP Protocol Overview [.1]

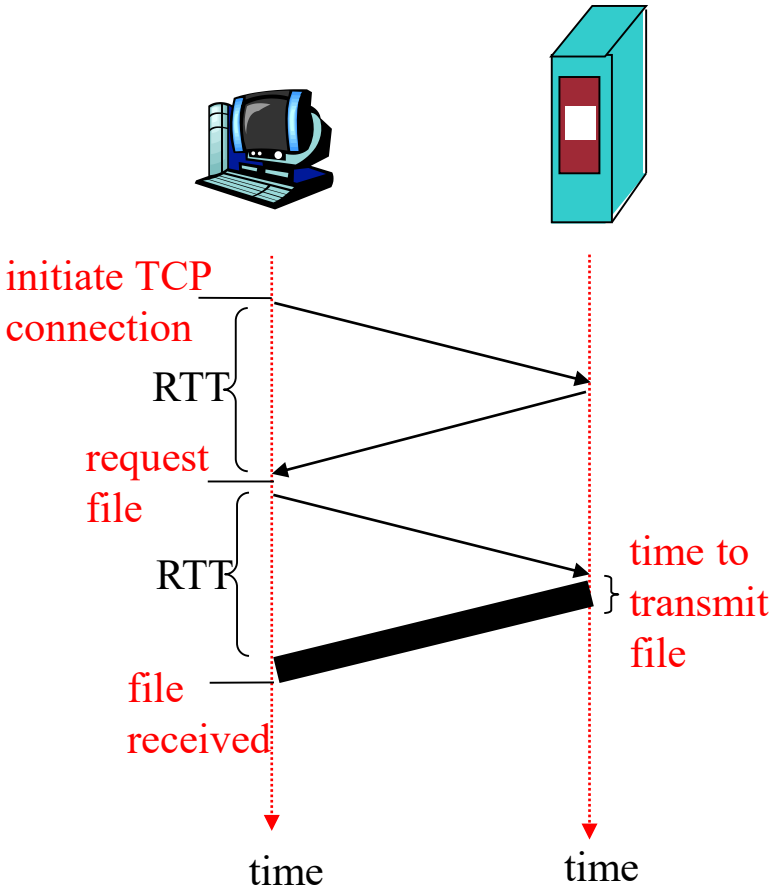
- Types of messages exchanged
  - e.g., request, response
- Message syntax:
  - What fields in messages & how fields are delineated
- Message semantics
  - Meaning of information in fields
- Rules for when and how processes send & respond to messages



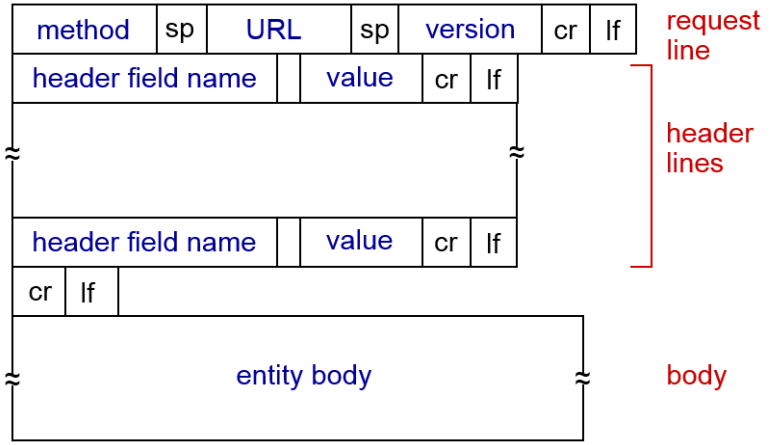
# HTTP Protocol Overview [..2]

## Uses TCP:

- Client initiates TCP connection (creates socket) to server at port 80
- Server accepts TCP connection from client
- HTTP messages exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed



# HTTP Request Message




request line (GET, POST, HEAD commands)

header lines

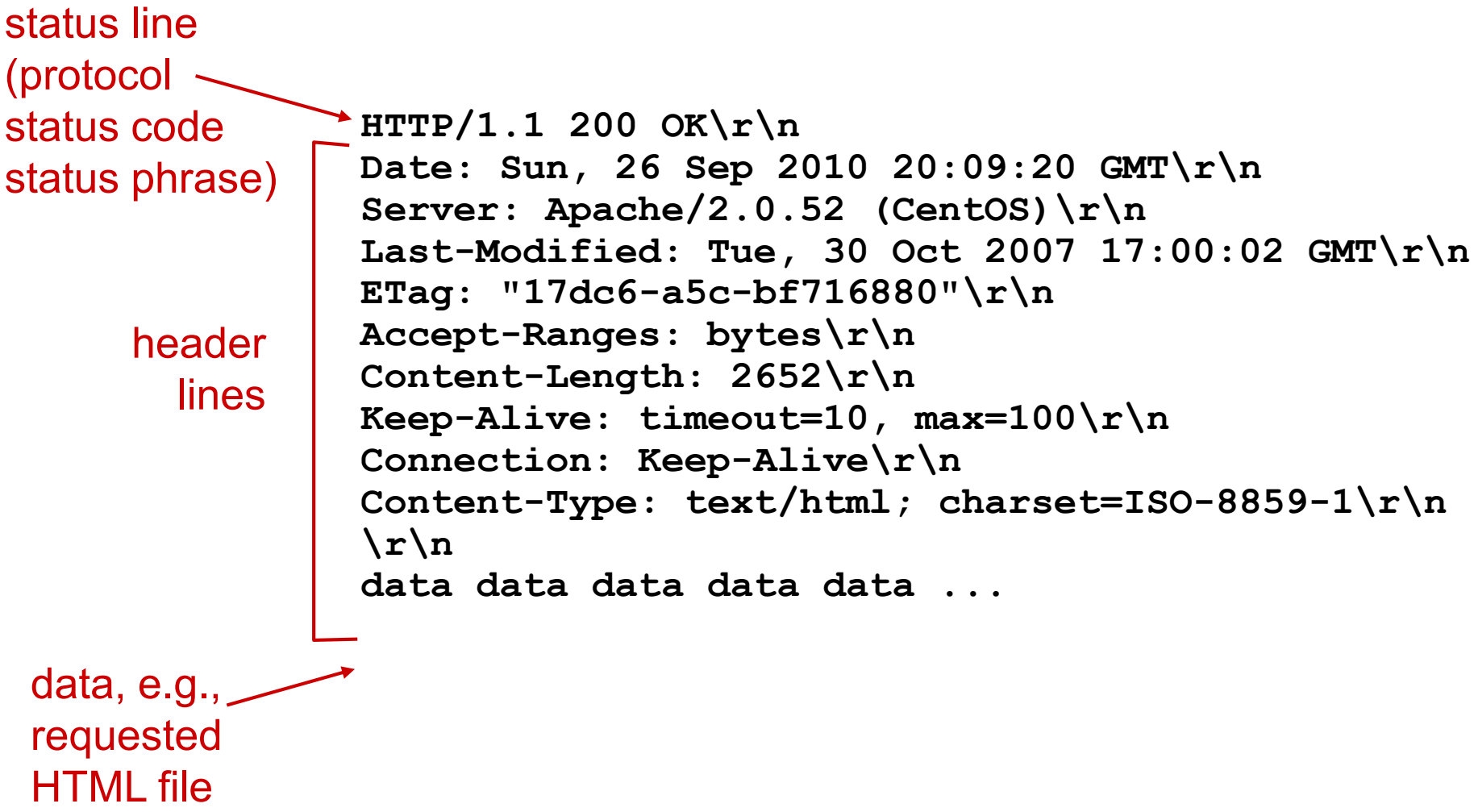
carriage return, line feed at start of line indicates end of header lines

carriage return character

line-feed character

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\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
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
data data data data data ...
```

# Response Message



# HTTP Method Types

## HTTP/1.0

- GET
  - Include user data in URL field of HTTP GET request message (following a '?'):
    - https://www.bitsadmission.com/bitsatmain.aspx?id=11012016
- POST
  - User input sent from client to server in entity body of HTTP POST request message
- HEAD
  - Asks server to leave requested object out of response

## HTTP/1.1

- Additional methods
- PUT
  - Uploads file in entity body to path specified in URL field
- DELETE
  - Deletes file specified in the URL field

# HTTP Response status Codes



**200 OK**

- request succeeded, requested object later in this msg

**301 Moved Permanently**

- requested object moved, new location specified later in this msg (Location:)

**400 Bad Request**

- request msg not understood by server

**404 Not Found**

- requested document not found on this server

**505 HTTP Version Not Supported**

- the **HTTP** version used in the request is not supported by the server.

# How does a Webpage transfer?



- Let's assume a web page consists of a base HTML file and 5 JPEG images.

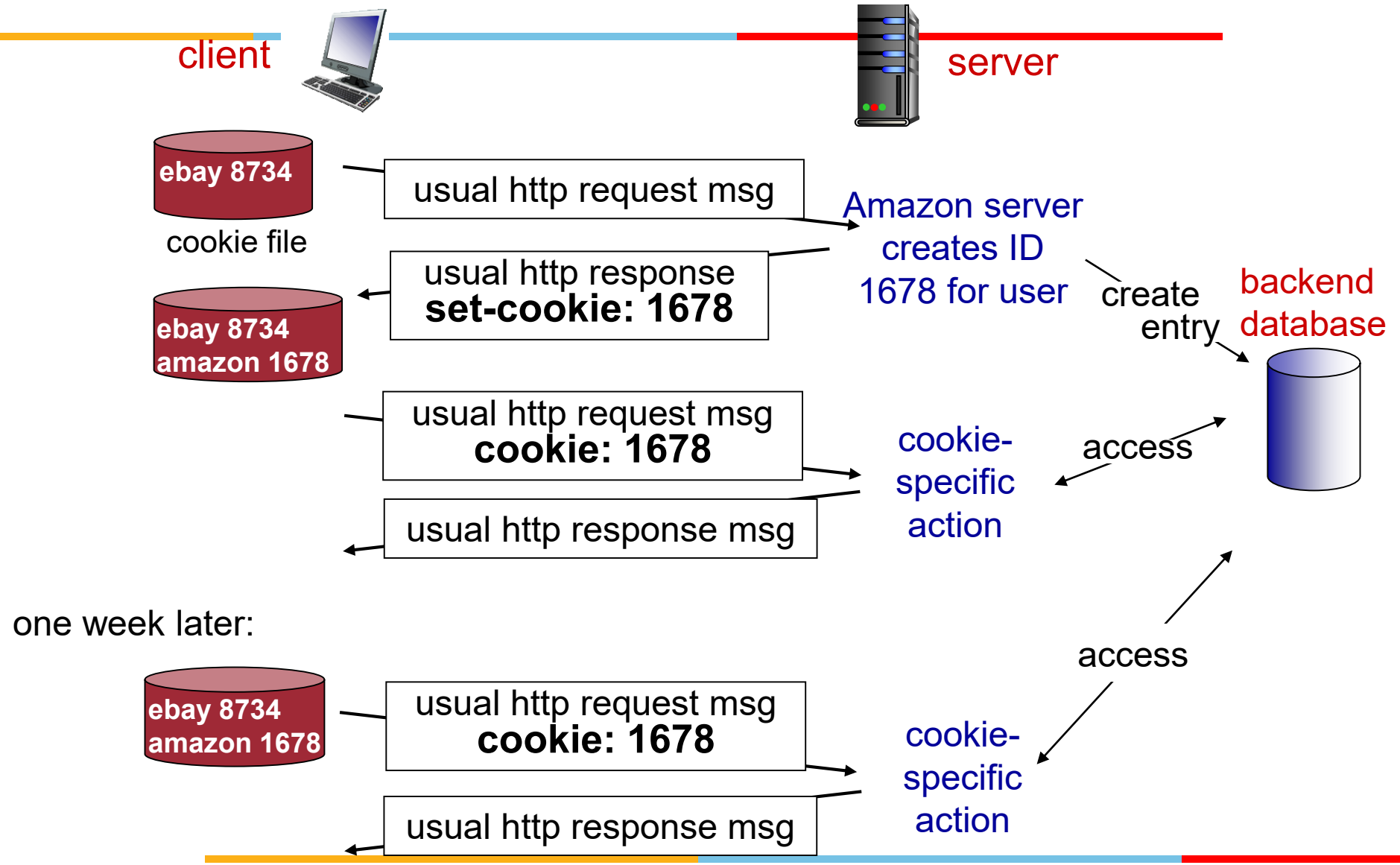
## Non-persistent HTTP

- At most one object is sent over a TCP connection
- HTTP/1.0 uses non-persistent HTTP

## Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- Persistent with Pipeline vs. Persistent without Pipeline
- HTTP/1.1 uses persistent connections in default mode

# State in HTTP using "Cookies"



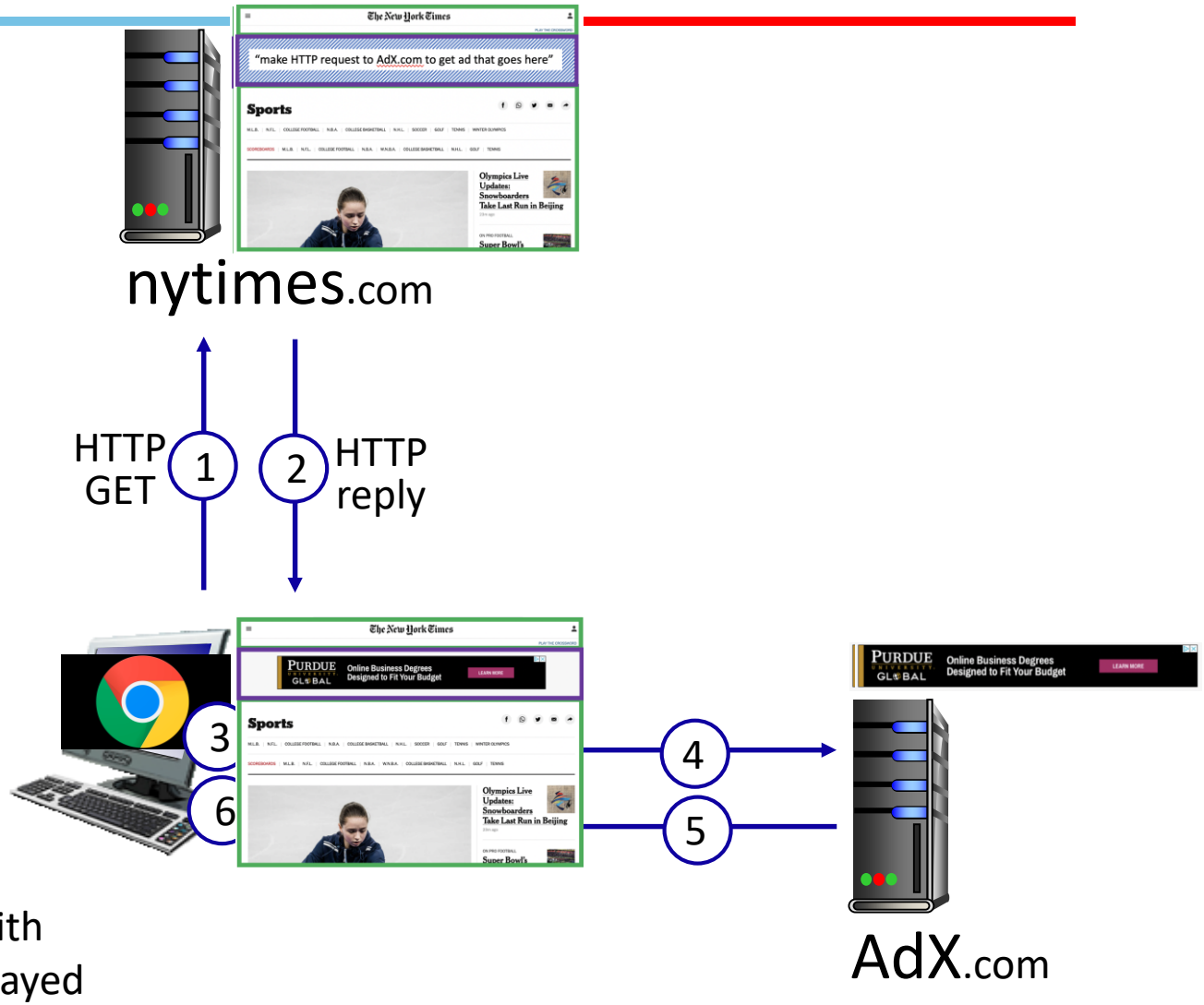
# Example: displaying a NY Times web page



1 2 GET base html file from nytimes.com

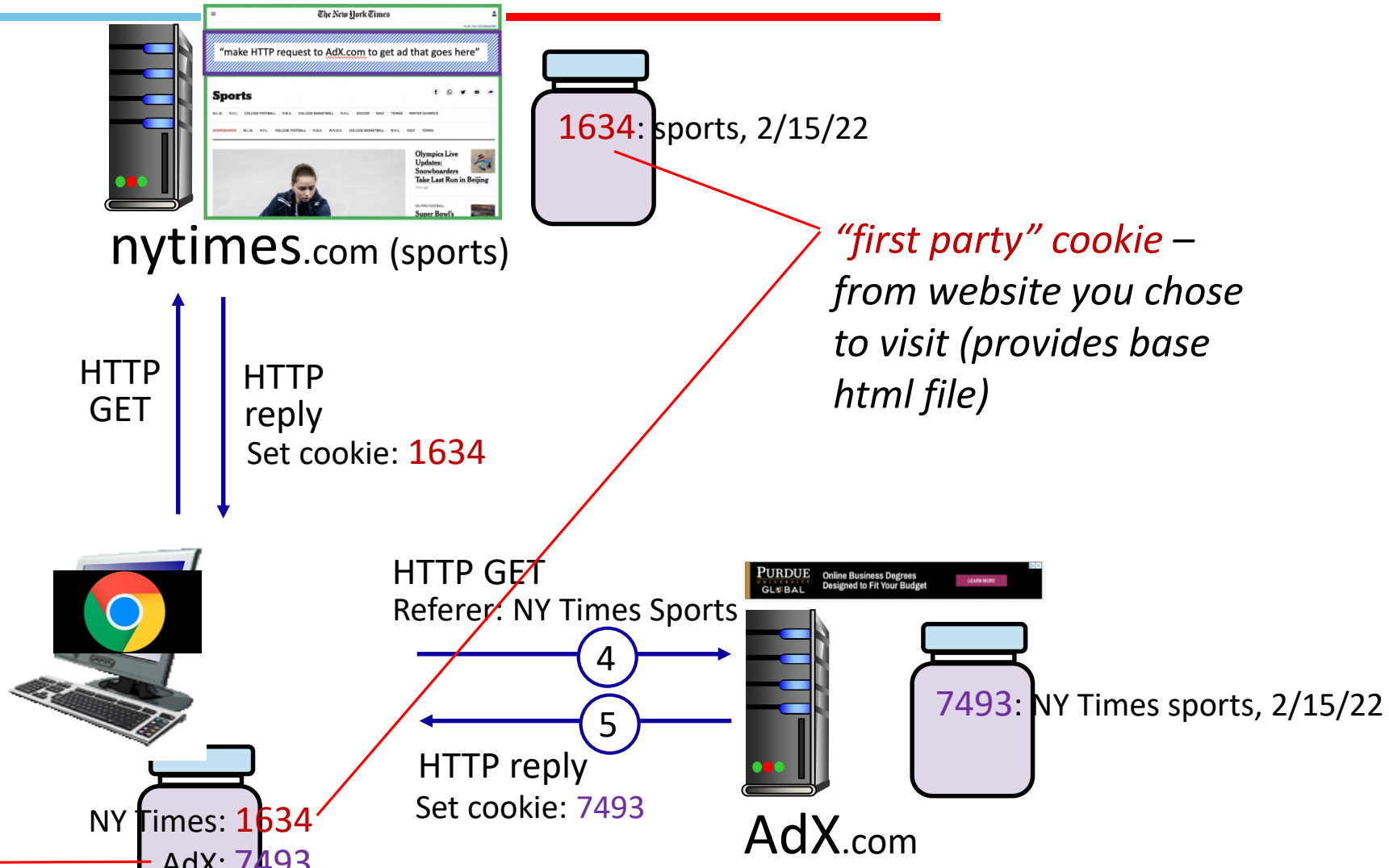
4 5 fetch ad from AdX.com

7 display composed page



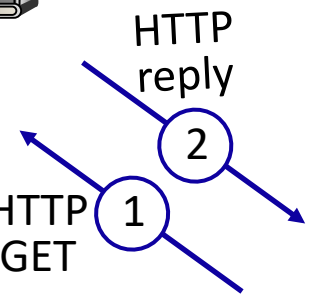

NY times page with embedded ad displayed

# Cookies: Tracking a user's browsing behavior

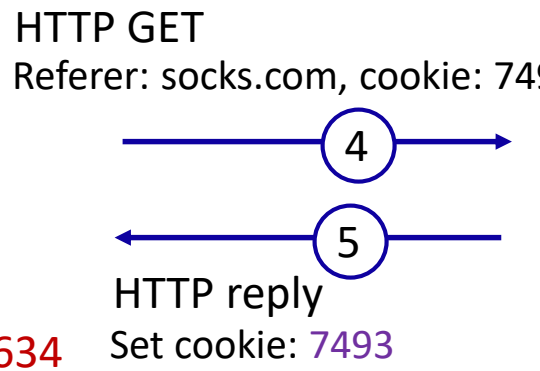


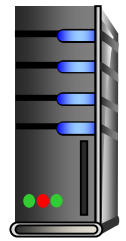
# Cookies: tracking a user's browsing behavior


socks.com







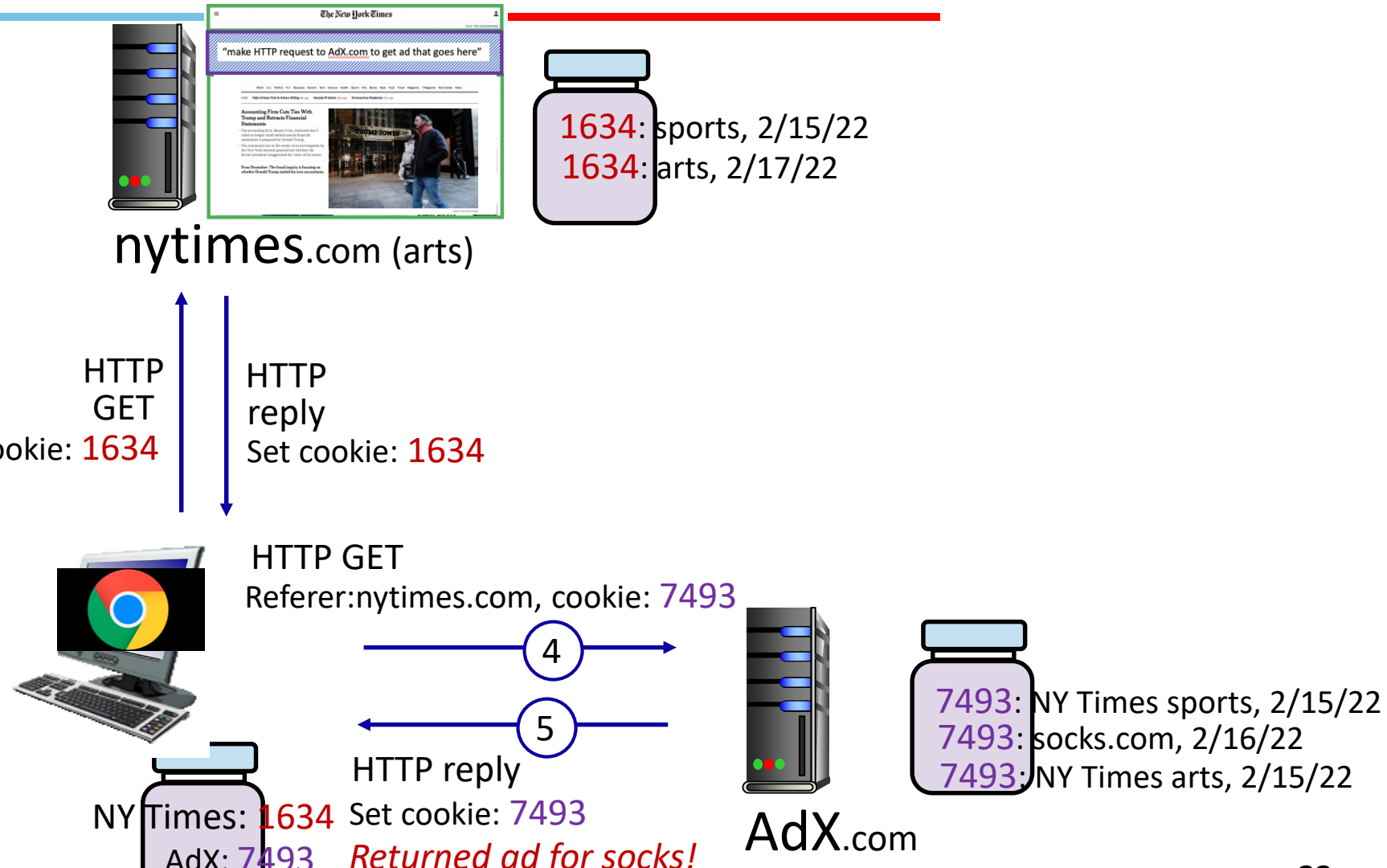




AdX.com

AdX:

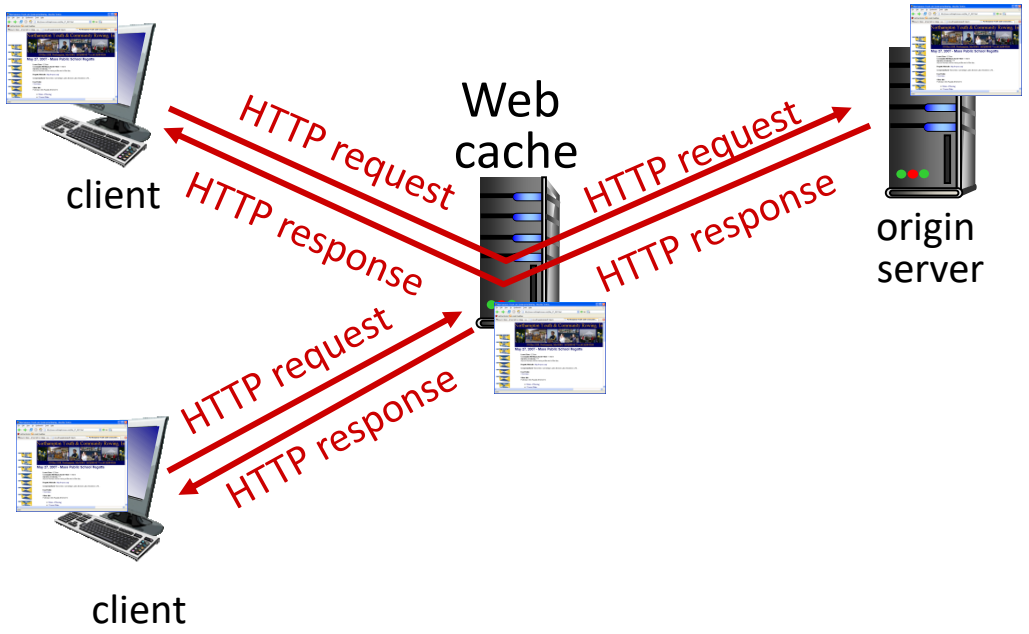
- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

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



# Web Caches (aka Proxy Server)

*Goal:* satisfy client requests without involving origin server

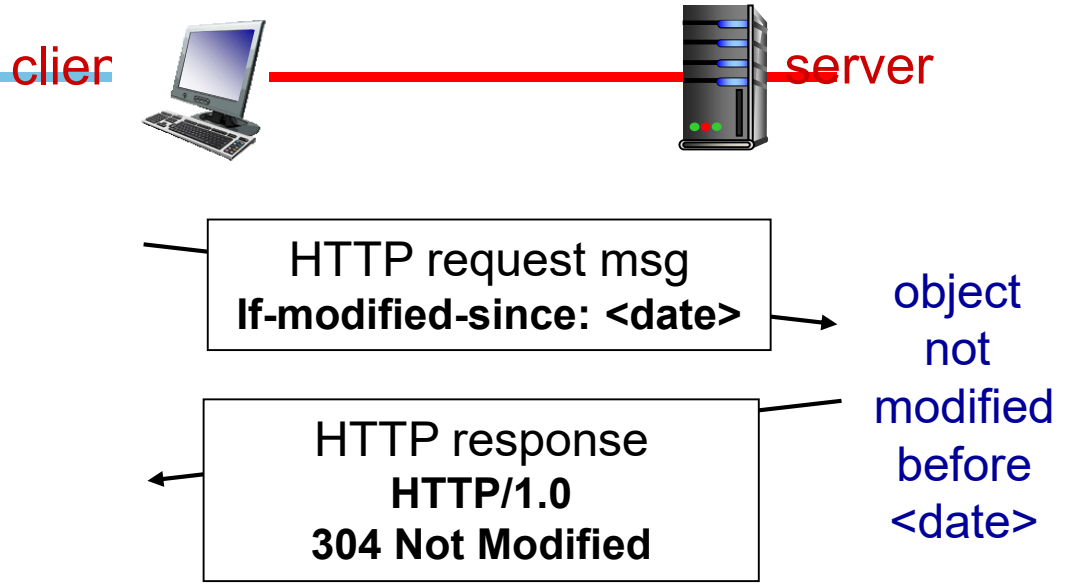


# Conditional GET

- ***Goal:*** don't send object if cache has up-to-date cached version
- ***cache:*** specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- ***server:*** response contains no object if cached copy is up-to-date:

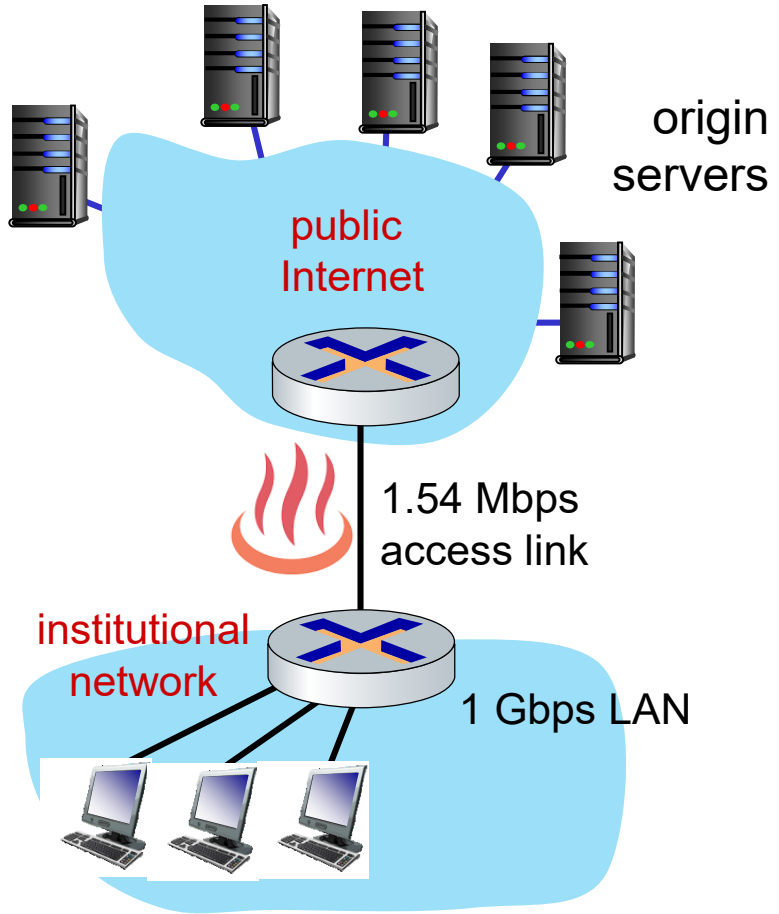
  `HTTP/1.0 304 Not Modified`



# Caching example

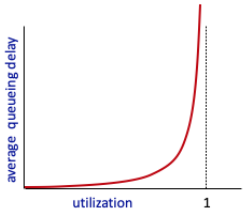
*Scenario:*

- 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

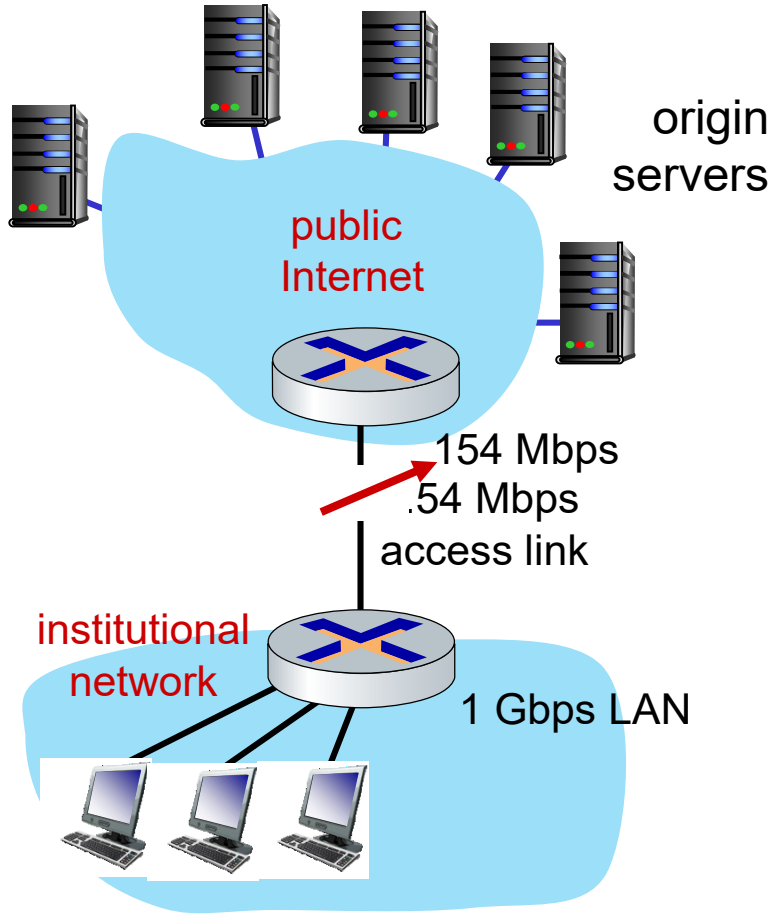



*problem:* large queueing delays at high utilization!

# Option 1: buy a faster access link

*Scenario:*

- access link rate: 154 Mbps → 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 = → .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + → + usecs → msecs

*Cost:* faster access link (expensive!)

# Option 2: install a web cache



*Scenario:*

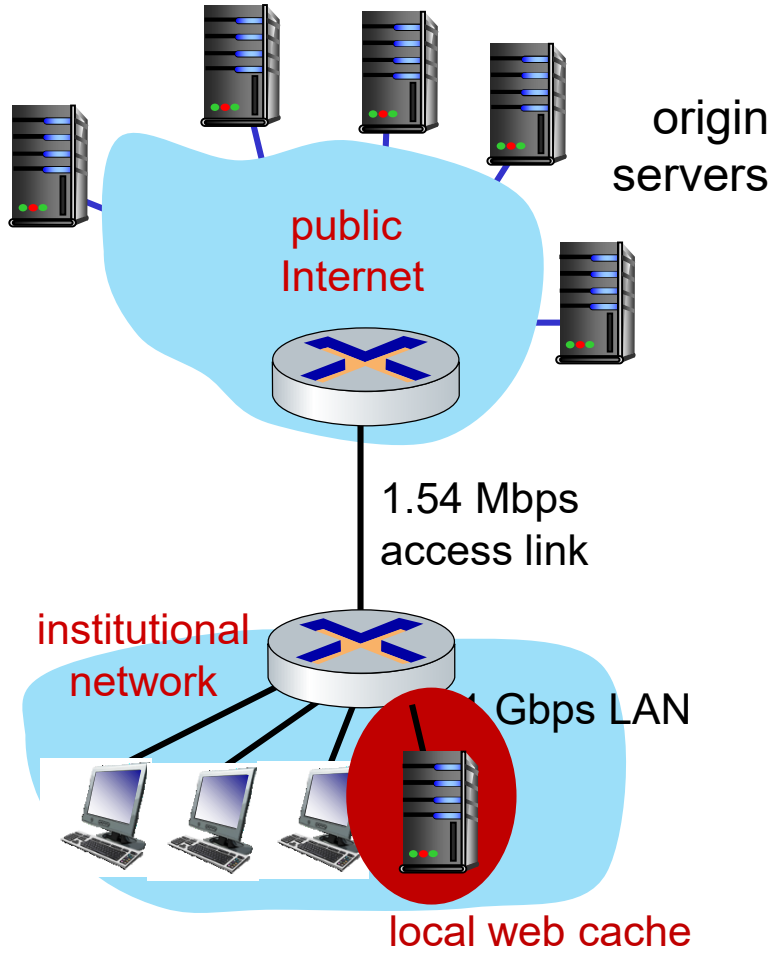
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

*Cost:* web cache (cheap!)

*Performance:*

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

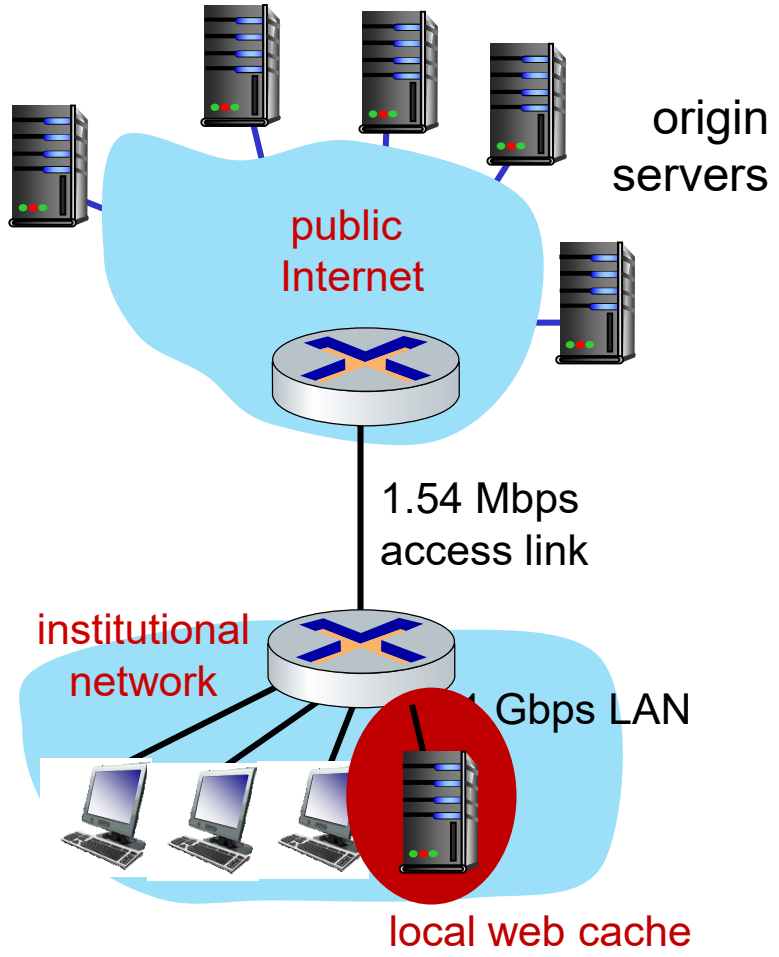
*How to compute link utilization, delay?*



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

Suppose cache hit rate is 0.4:

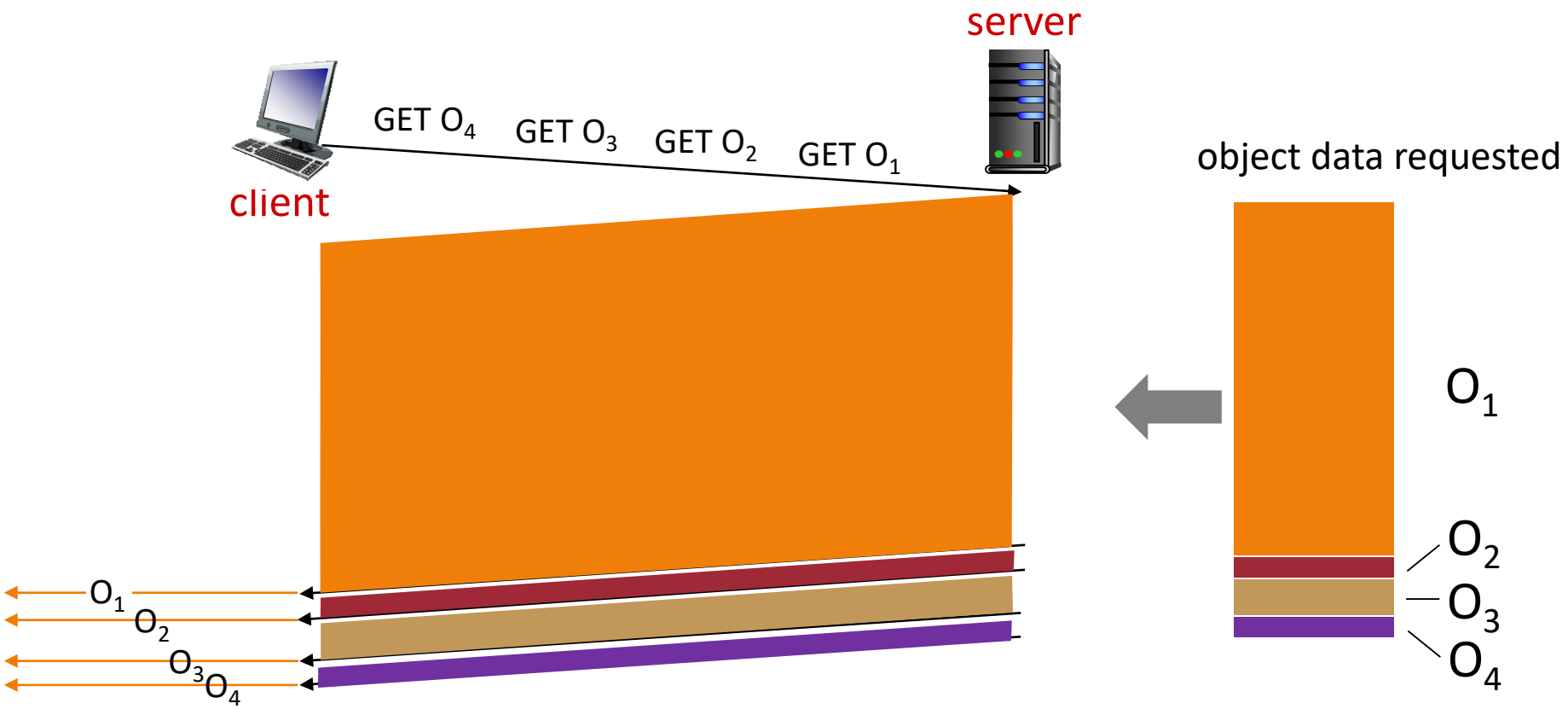
- 40% requests served by cache, with low (msec) delay
  - 60% requests satisfied at origin
    - rate to browsers over access link
      = 0.6 * 1.50 Mbps = .9 Mbps
    - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
  - average end-end delay:
    = 0.6 * (delay 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!)*

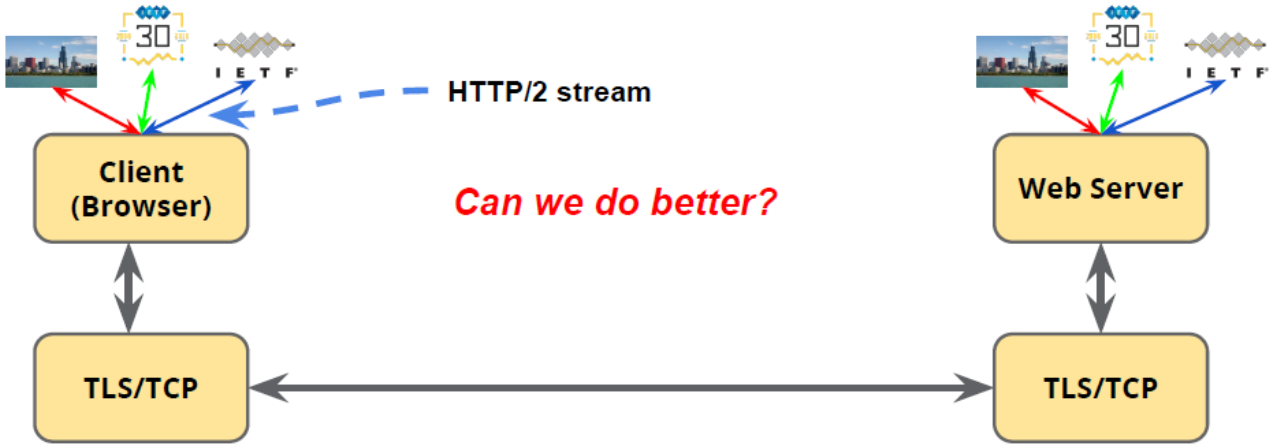
# HTTP/1.1: HOL blocking

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




Server responds *in-order* (FCFS) to multiple GET requests

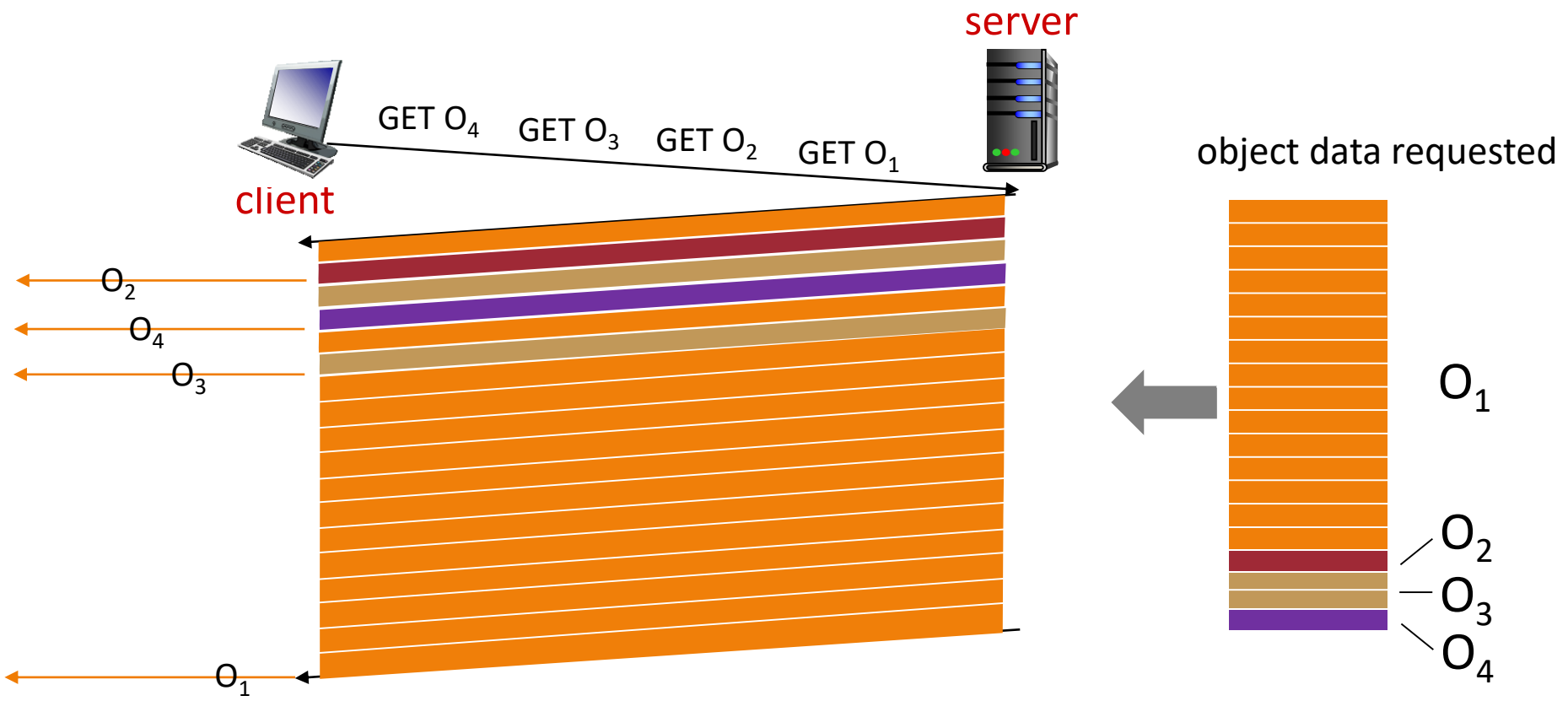
# HTTP/2: Stream Multiplexing




- What is a ***stream***?
  - Bi-directional sequence of **frames** sent over the HTTP/2 protocol exchanged between the server and client

# 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



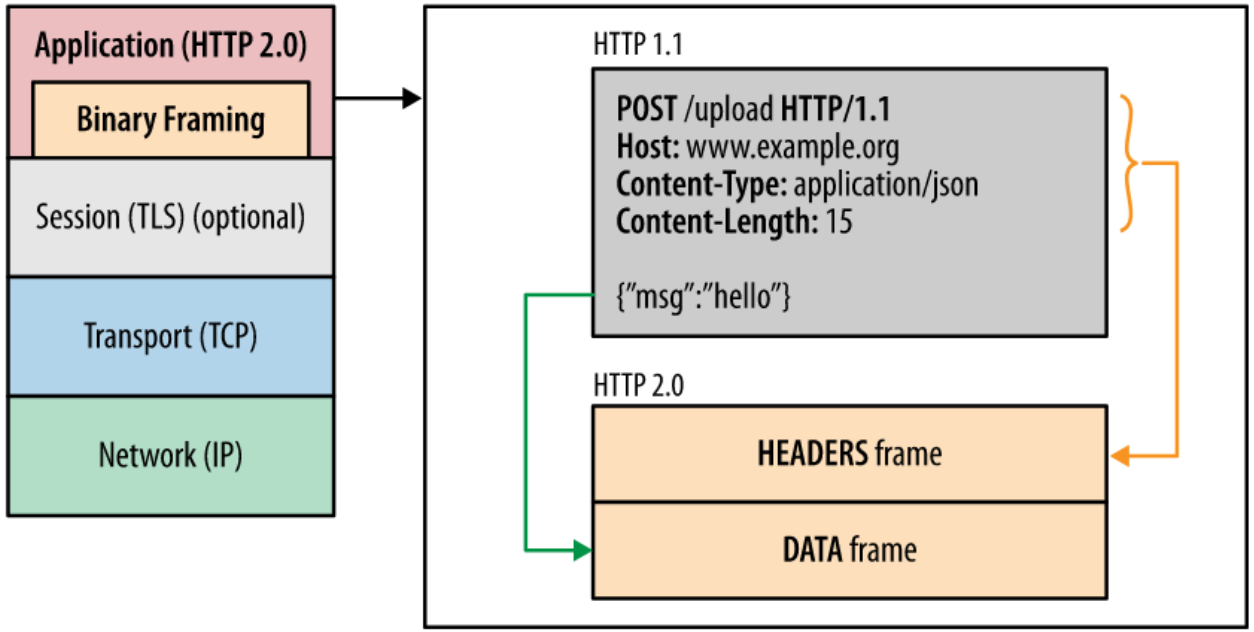
*Key goal:* decreased delay in multi-object HTTP requests

*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
- ***push*** unrequested objects to client
- Divide objects into frames, schedule frames to mitigate HOL blocking

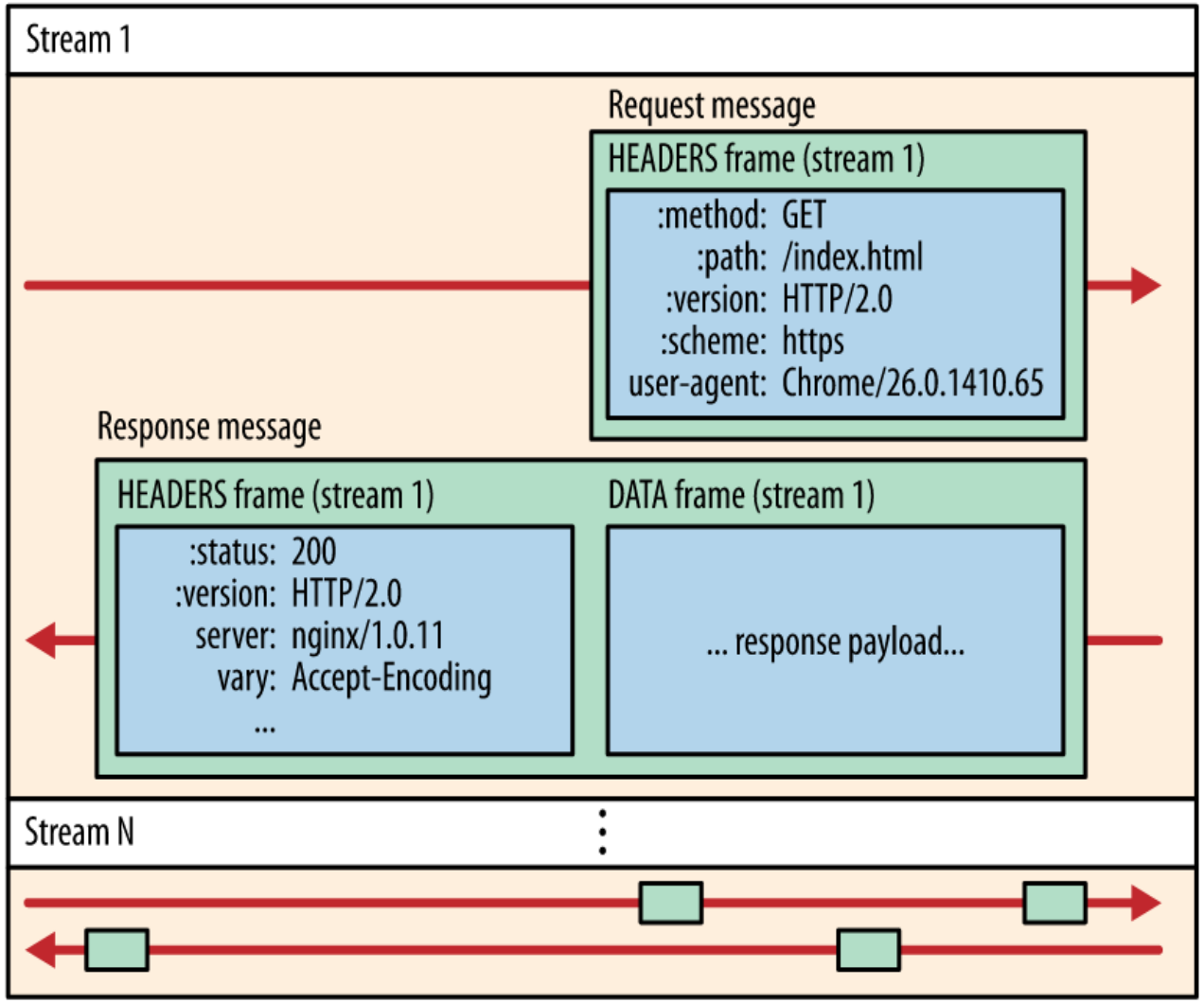
# Binary Framing Layer

- HTTP/2 allows transmission of parallel multiplexed requests and responses
  - HTTP/2 breaks down the HTTP protocol communication into an exchange of binary-encoded **frames**, which are then mapped to **messages** that belong to a particular **stream**, all of which are multiplexed within a single TCP connection.



# HTTP/2.0 Connection

- ***Stream***: A bidirectional flow of bytes within an established connection, which may carry one or more messages.
- ***Message***: A complete sequence of frames that map to a logical request or response message.
- ***Frame***: The smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs.



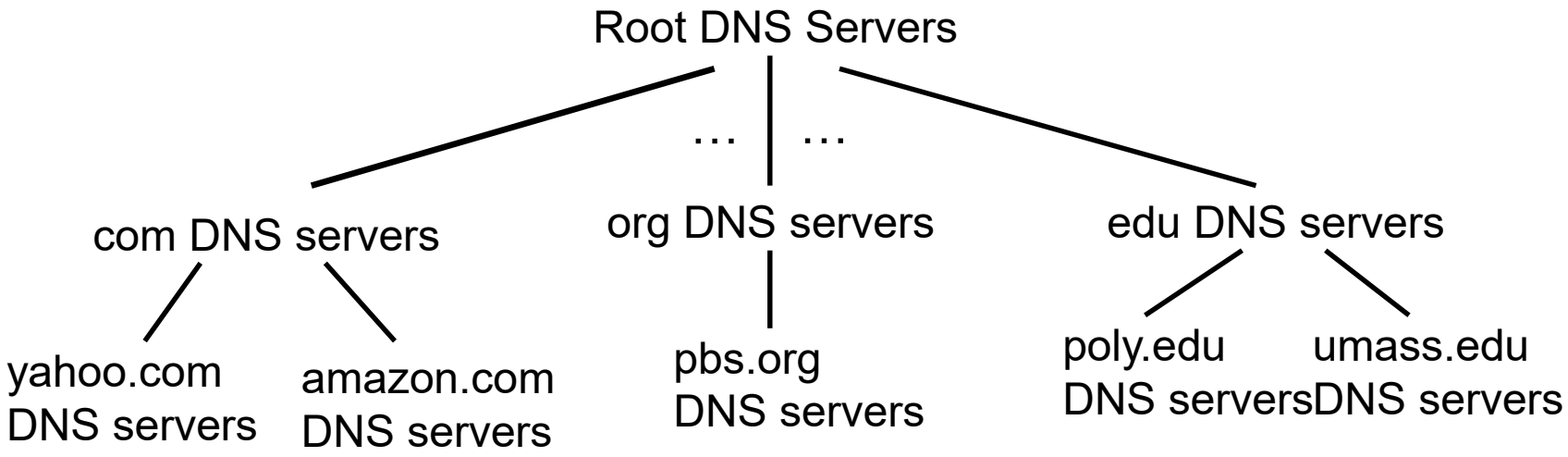
# Domain Name System (DNS)

- The **domain name system** maps the **name** people use to locate a website to the IP address that a computer uses to locate a website.

- Why do we need the mapping between host name and IP address?

- *Application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)

# DNS Structure – Distributed Hierarchical Database



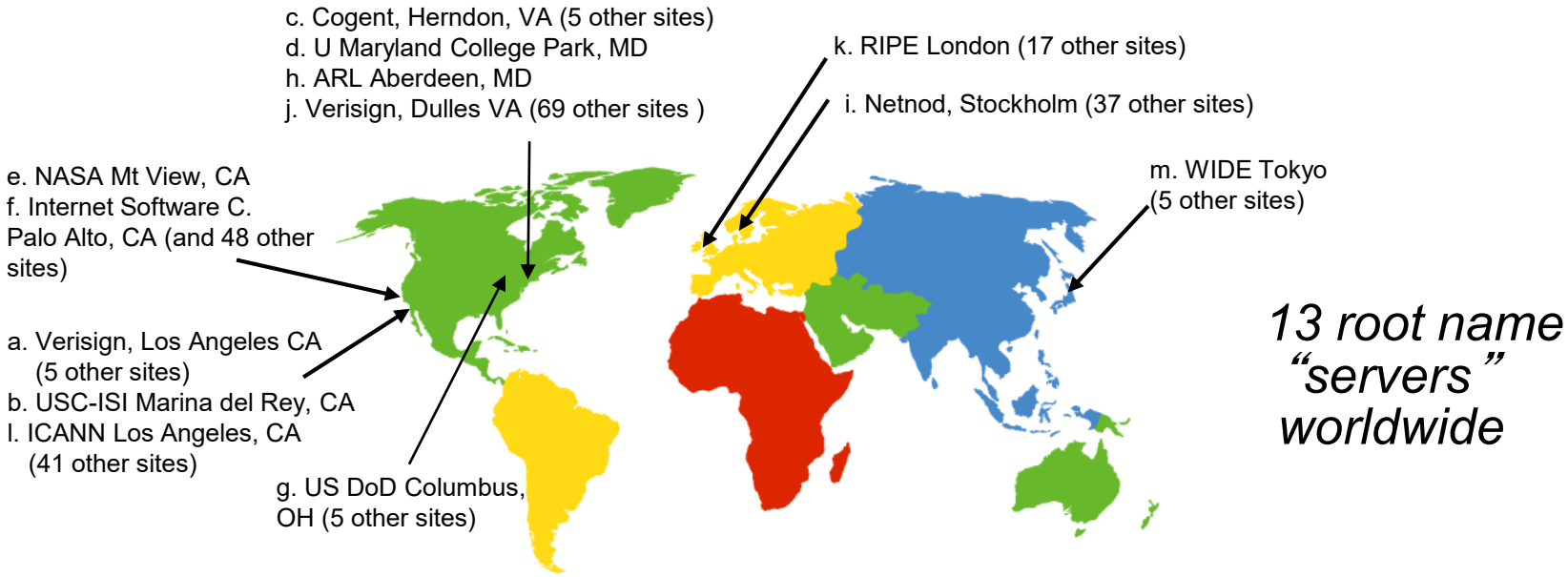

*Client wants IP for www.amazon.com; 1st approx:*

- Client queries root server to find com DNS server
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

List of all top level domain servers is available at: ***https://www.icann.org/resources/pages/tlds-2012-02-25-en***

# Root Name Servers

- Root name server:
  - Total 13 server, mostly located in North America.
  - Each server is actually a network of replicated servers (~200 servers in US)
  - ICANN manages root DNS domain

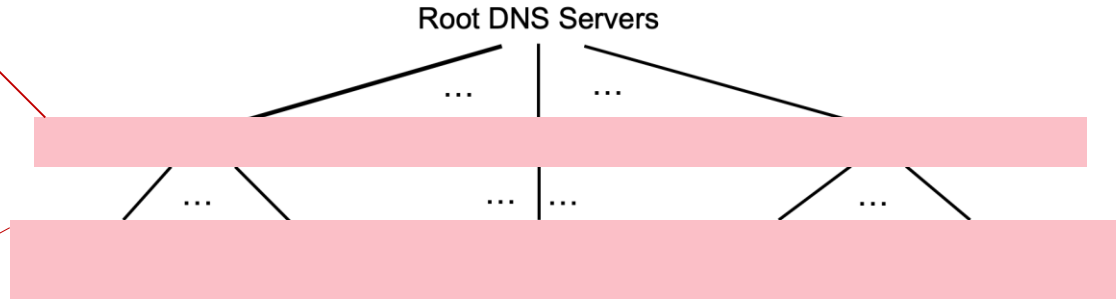



*13 root name "servers" worldwide*

# Top-Level Domain, and Authoritative Servers

## Top-Level Domain (TLD) servers:

- Responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD




## Authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

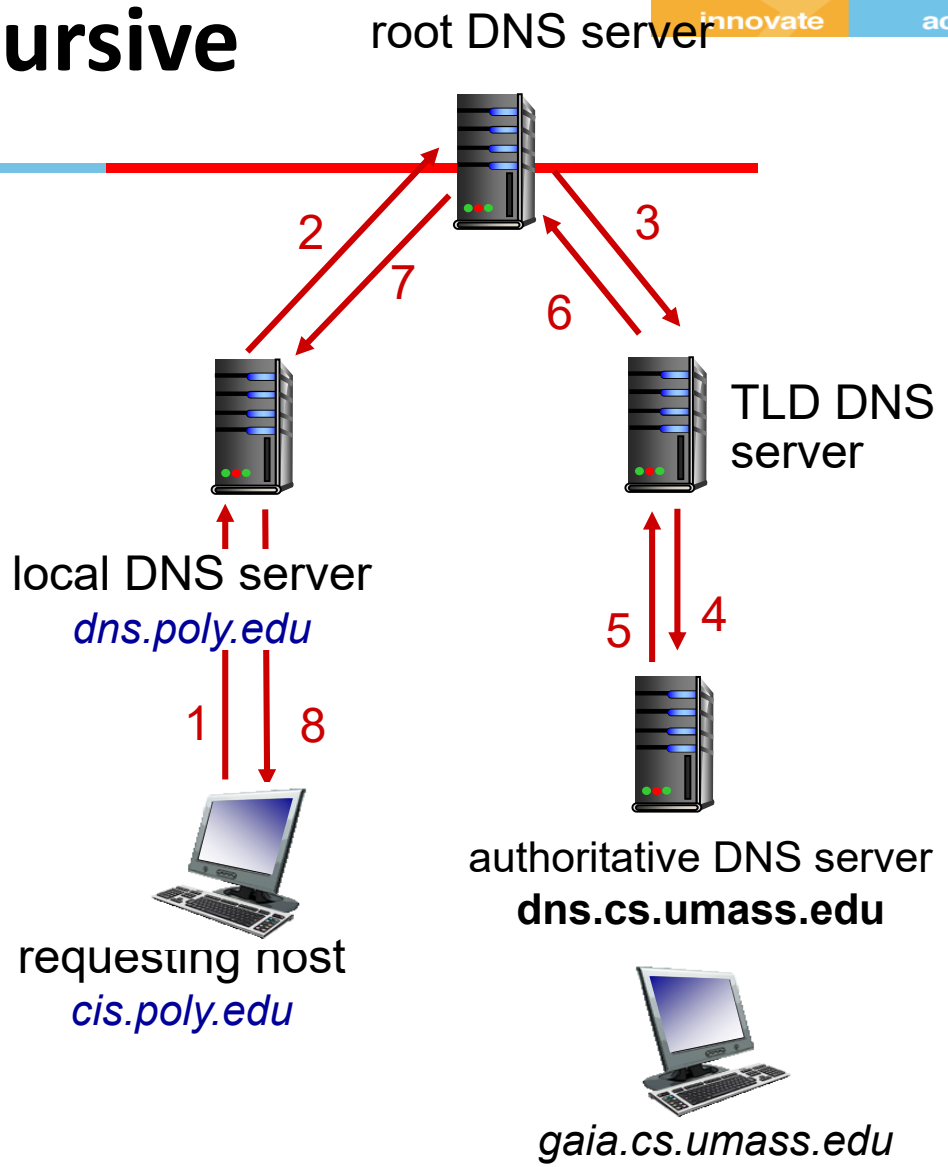
# DNS Services

- Hostname to IP address translation
  - Host name to IP address mapping
- Host aliasing
  - Canonical name to alias name(s) mapping
- Mail server aliasing
  - Host name to mail server mapping
- Load distribution
  - Replicated Web servers: many IP addresses correspond to one name

# DNS Query Processing - **Recursive**

## Recursive query:

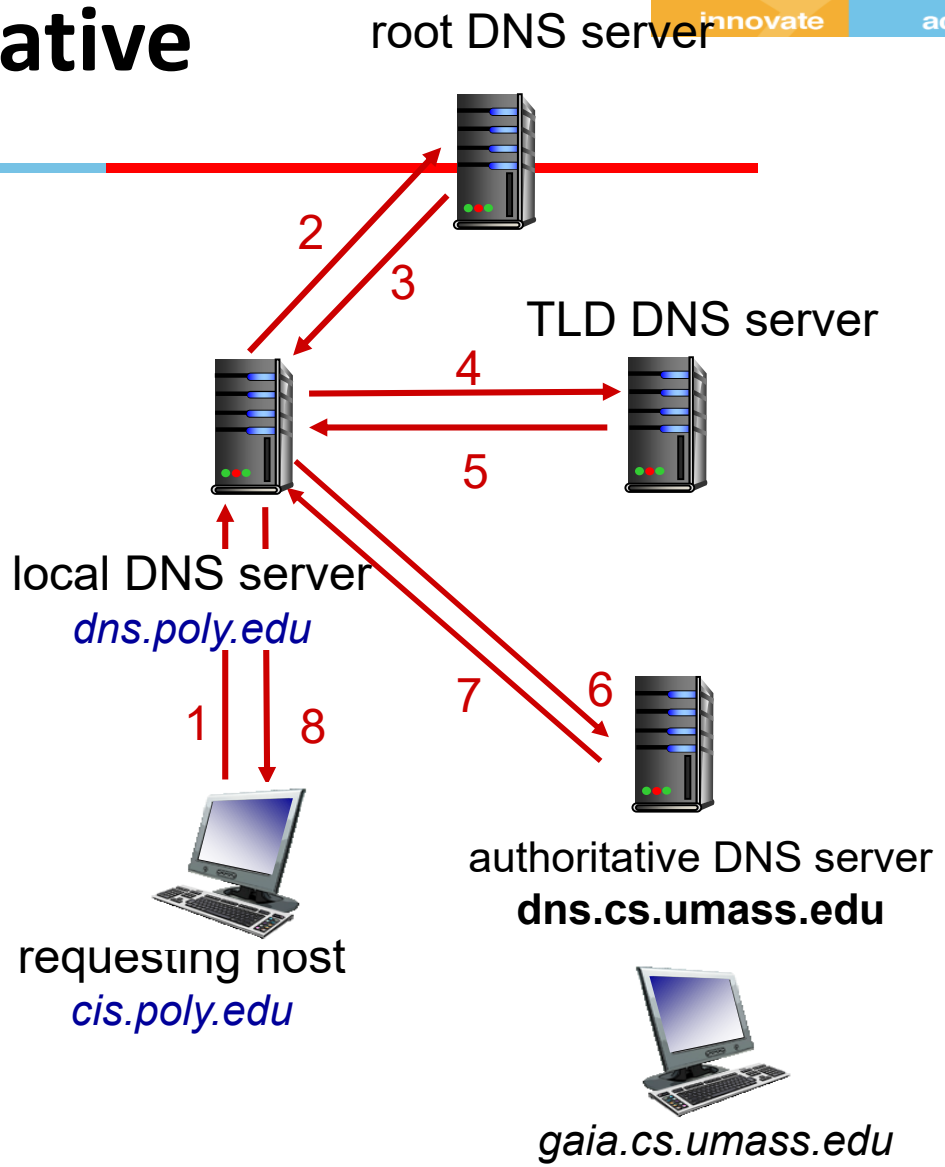
- Puts burden of name resolution on contacted name server
- Heavy load at upper levels of hierarchy?



# DNS Query Processing - **Iterative**

- TLD server may know only of an intermediate DNS server for the hostname, which in turn knows the authoritative DNS server for the hostname.

- DNS responses are usually cached to improve the delay performance and to reduce the number of DNS messages
  - e.g., Local DNS server caches the TLD server information



# DNS Records

*DNS:* distributed database for storing resource records (RR)

RR format: `(name, value, type, ttl)`

## type=A

- **`name`** is hostname
- **`value`** is IP address

## type=NS

- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

## type=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

## type=MX

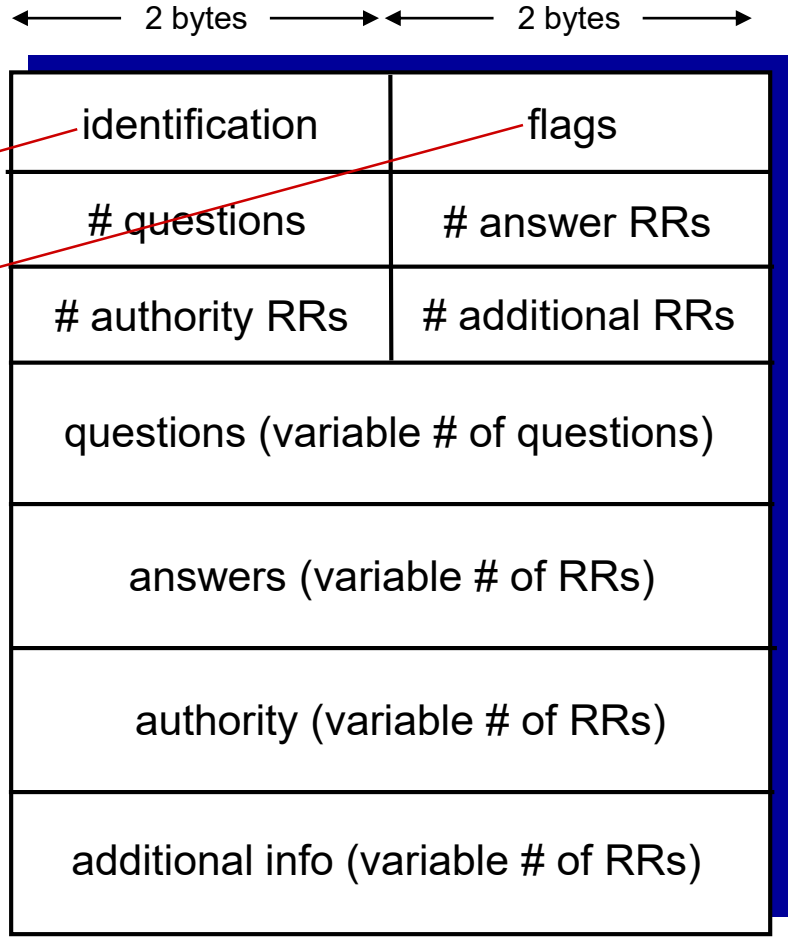
- **`value`** is name of mailserver associated with **`name (host name, i.e., mail server alias)`**

# DNS Messages

- *Query* and *reply* messages, both with same *message format*
- *Explore DNS protocol in Lab Session #2*

msg header

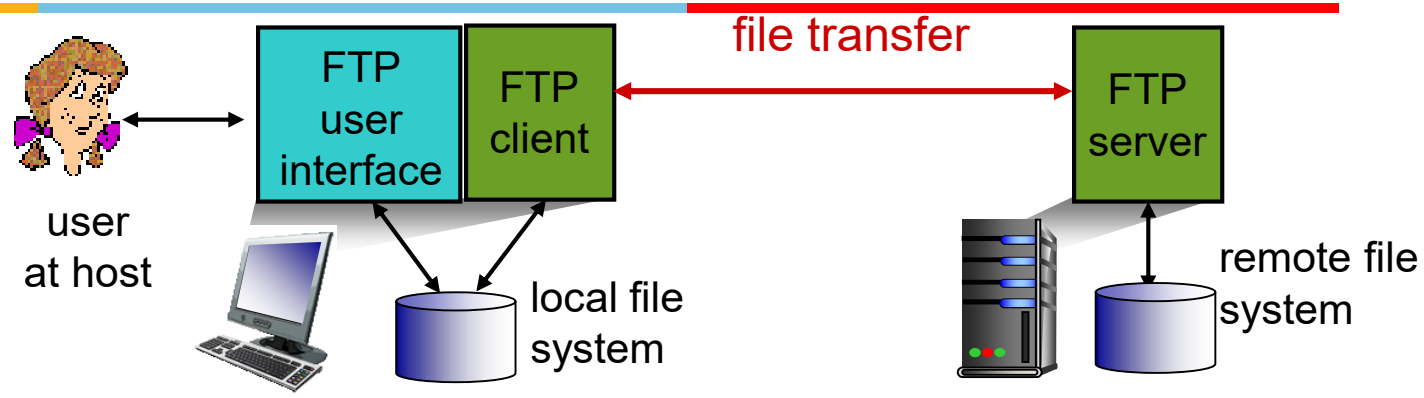
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



# Inserting Records into DNS

- A newly created domain name should be first registered at a registrar
  - Internet Cooperation of Assigned Names and Numbers (ICANN) accredits the **registrars**
  - Accredited **registrar** list is available at www.internic.net
  - **Registrar** is a commercial entity that verifies the uniqueness of the domain name.

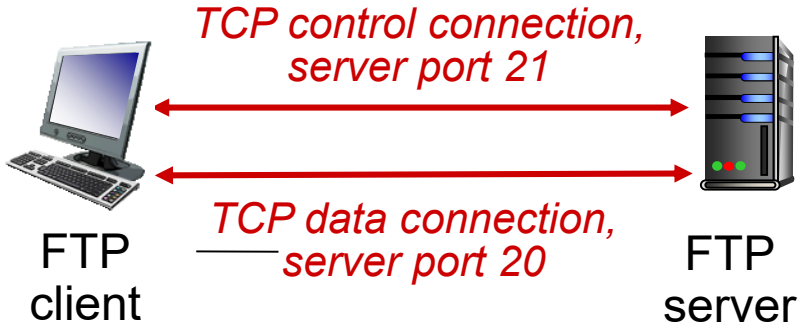
# FTP: File Transfer Protocol




- Transfer file to/from remote host
- Client/server model
  - *Client:* side that initiates transfer (either to/from remote)
  - *Server:* remote host
- ftp: RFC 959
- ftp server: port 21

# FTP: Connections

- Control connection
  - Authorization, directory listing etc.
- When server receives file transfer command,
  - *Server* opens *2nd* TCP data connection (for file) *to* client
- After transferring one file, server closes data connection



# FTP Commands and Responses



## *Sample commands:*

- Sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

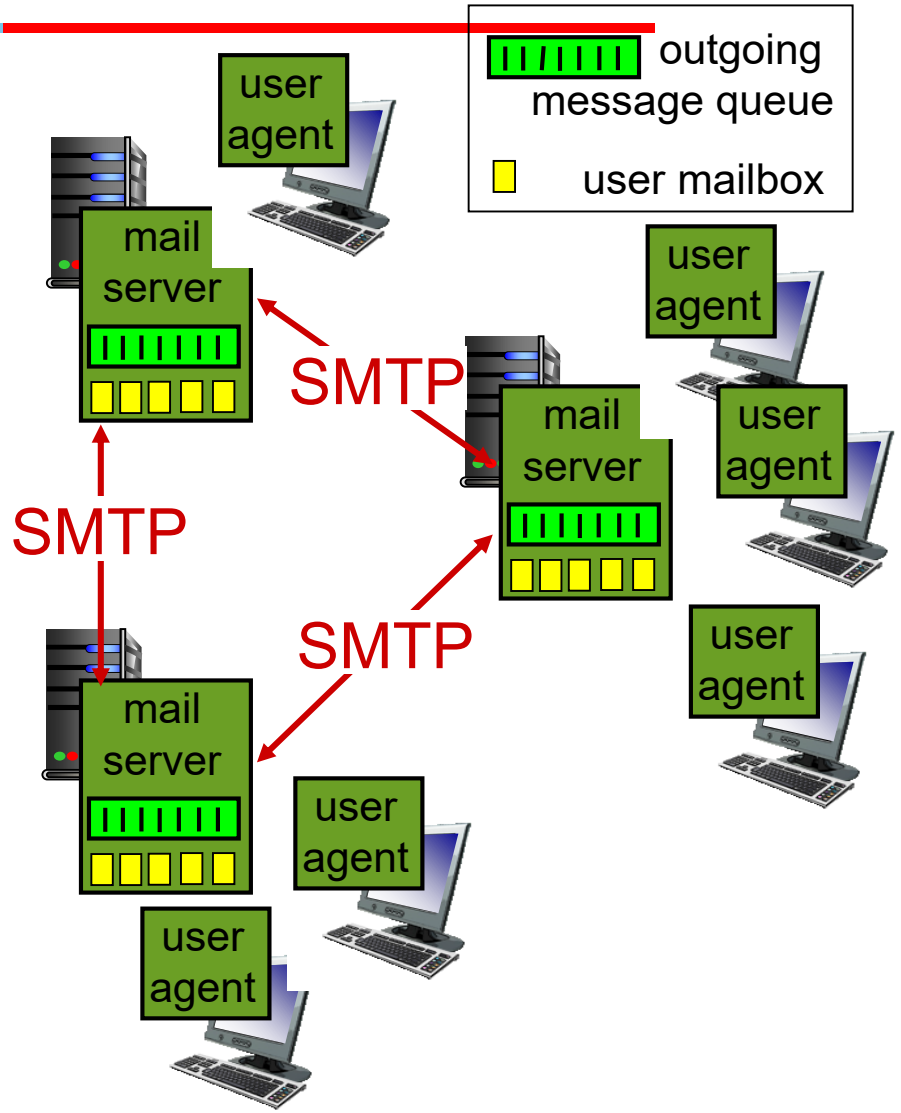
## *Sample return codes*

- **Status code and phrase (as in HTTP)**
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

# eMail

## Three major components:

- User agents
  - e.g., Outlook, Thunderbird
- Mail servers
  - Contains incoming messages for user
- Simple mail transfer protocol:
  - SMTP

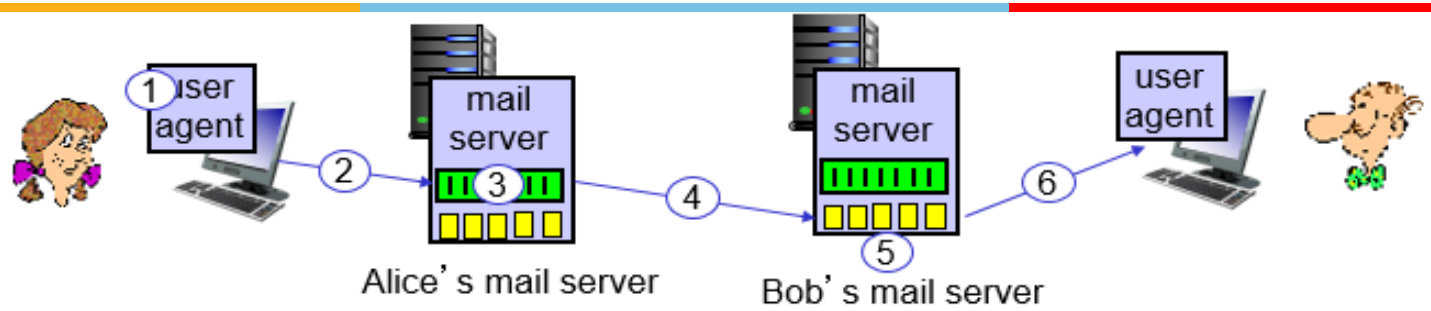


# SMTP [RFC 5321, Original RFC 821]



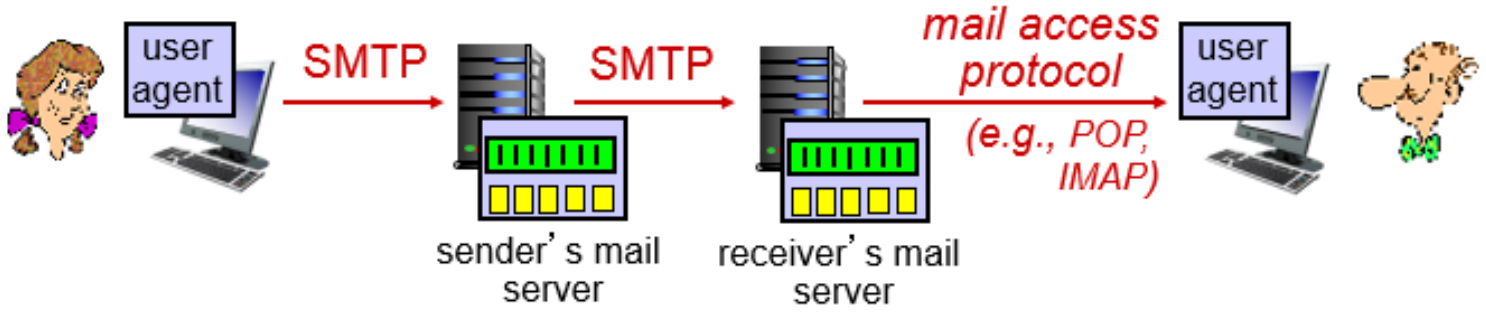
- Uses TCP to reliably transfer email message from client to server, port 25
- Direct transfer: sender's mail server to receiver's mail server
- Three phases of transfer
  - Handshaking (greeting)→Transfer of messages→Connection Closure
- Command/response interaction (like HTTP, FTP)
  - Commands: ASCII text
  - Response: status code and phrase
- Messages must be in 7-bit ASCII
  - Painful for multimedia data

# Mail Transfer Process




```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# Mail Access Protocols




- Mail access protocol: retrieval from server
  - POP3 [Port:110]: Post Office Protocol [RFC 1939]: authorization, download and keep, download and delete
    - User can create folders and move the messages into them locally.
    - Stateless across the sessions
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
    - Allows to create remote folders and maintains user state information across IMAP sessions
    - Permit a user agent to obtain components of messages. Good for low bandwidth connections.

# POP3 Protocol

## *Authorization phase*

- Client commands:
  - **`user:`** declare username
  - **`pass:`** password
- Server responses
  - **`+OK`**
  - **`-ERR`**

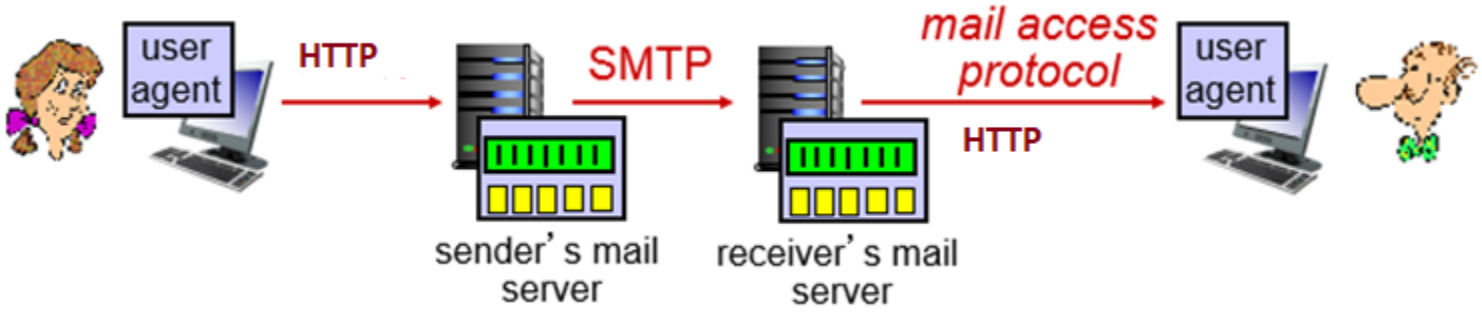
## *Transaction phase,* client:

- **`list:`** list message numbers
- **`retr:`** retrieve message by number
- **`dele:`** delete
- **`quit`**

```
S: +OK POP3 server ready
C: user alex
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# Web based E-Mail

- Hotmail introduced Web-based access in the 1990s



# Thank You!