# Lecture 5: Application Layer

## Overview and HTTP

COMP 332, Spring 2018

Victoria Manfredi

WESLEYAN
UNIVERSITY

**Acknowledgements**: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved.

# Today

1. Announcements
   - homework 2 due Wed. by 11:59p
   - virtual machines and networking
     - may want to turn networking on/off if having issues (or restart)
     - good way to check for internet (and dns) access: ping website

2. Application layer
   - overview
   - Web and HTTP

3. HTTP protocol
   - requests, responses, error codes
   - cookies

# Application Layer

## OVERVIEW

# Application layer: where apps live

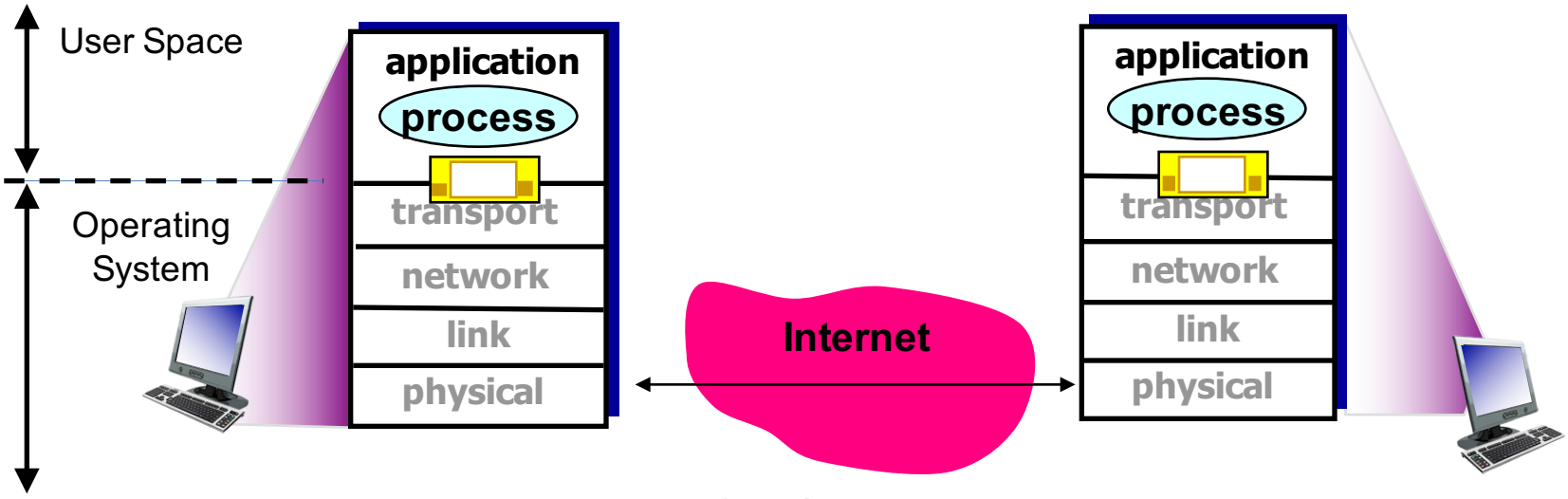
## Application software

– processes running different hosts, communicate via messages

## Application architecture

– client-server vs. peer-to-peer vs. hybrid

– overlaid on network architecture



# Application layer protocols

Provide specific services to application

## Define

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

## Rely on transport layer

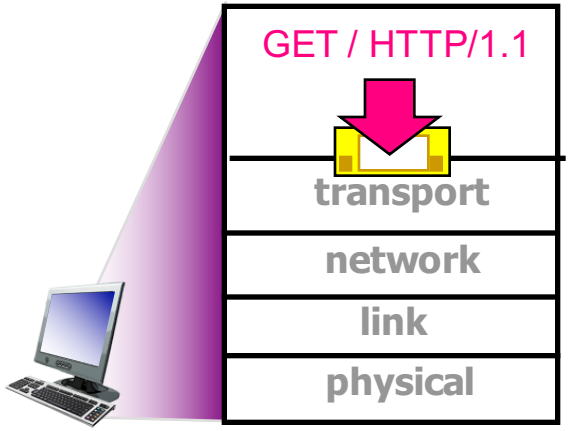
- to get messages from process on one host to process on another host

## Open protocols

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

## Proprietary protocols

- e.g., Skype



# Application requirements

Dictate what transport layer services application needs
TCP or UDP (or SSL/TCP or QUIC if you're Google)?

| Service | App requirements |
|---|---|
| Reliable data transfer: does all data need to be received? | Loss-tolerant? E.g. video? |
| Throughput: does data need to be delivered quickly? Is app sending lots of data? | Bandwidth sensitive? E.g., video<br>Elastic traffic? E.g., use as much/little bandwidth as available |
| Timing: does data need to be delivered at certain min rate? | Time-sensitive? E.g., voice, video need low delay |
| Security: does data need to be secured from eavesdroppers and modification? | Encryption?<br>Data integrity?<br>Endpoint authentication?<br>Confidentiality? |

# Services provided by Internet transport protocols

## TCP service

- connection-oriented
  - setup required between client and server processes
- reliable transport
  - messages delivered to destination process without error and in-order
- congestion control
  - sender reduces sending rate when network is overloaded
- flow control
  - sender reduces sending rate when destination is overloaded
- does not provide
  - timing, minimum throughput or delay guarantee, security

## UDP service

- unreliable data transfer
  - best-effort service between sender and destination processes
- does not provide
  - reliability
  - flow control
  - congestion control
  - timing
  - throughput guarantee
  - security
  - connection setup

Q: why bother? Why is there a UDP?

# Transport service requirements: common apps

| Application | Data loss | Throughput | Time sensitive |
|---|---|---|---|
| File transfer | no loss | elastic | no |
| E-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| Real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100's msec |
| Stored audio/video | loss-tolerant | same as above | yes, few secs |
| Interactive games | loss-tolerant | few kbps up | yes, 100's msec |
| Text messaging | no loss | elastic | yes and no |

Q: other apps you can think of?

# Internet apps: application, transport protocols

Associated with each app is an app layer protocol: depending on app requirements, runs over specific transport protocols

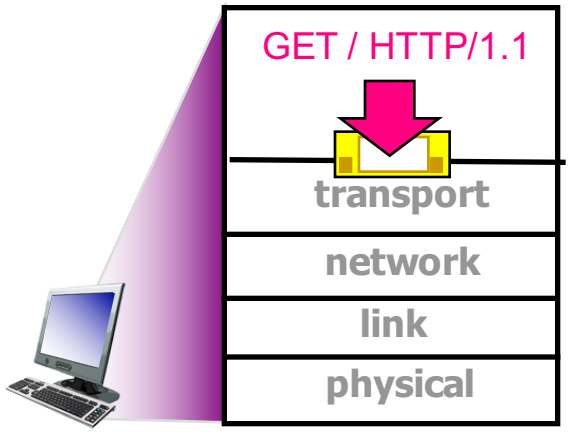
| Application | Application layer protocol | Underlying transport protocol |
|---|---|---|
| E-mail | SMTP [RFC 2821] | TCP |
| Remote terminal access | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| File transfer | FTP [RFC 959] | TCP |
| Streaming multimedia | HTTP (e.g., YouTube), RTP [RFC 1889] | TCP or UDP |
| Internet telephony | SIP, RTP, proprietary (e.g., Skype) | TCP or UDP |

Q: where does security come into play?

# Securing TCP

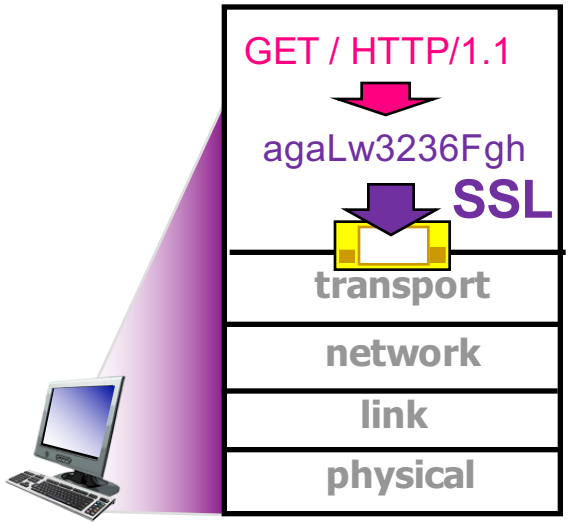
## TCP & UDP

- no encryption: cleartext passwords sent into socket traverse Internet in cleartext




## TLS/SSL

- at app layer
  - apps use SSL libraries, that "talk" to TCP
- provides encrypted TCP connection
  - data integrity
  - end-point authentication




## TLS/SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted

Q: Why does SSL run over TCP?
How is TLS/SSL related to OSI model?

# Network Applications

## WEB AND HTTP

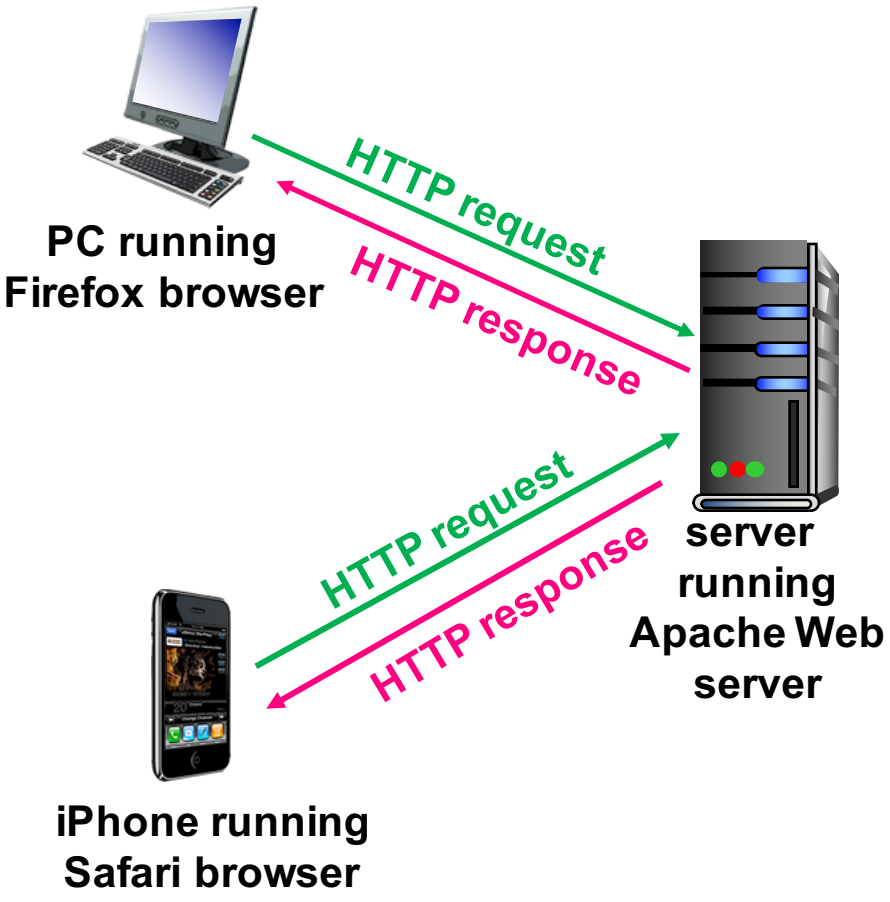
# Web's application layer protocol

## HTTP

- **H**yper**T**ext **T**ransfer **P**rotocol

## Client/server model

- client
  - browser that requests, receives, (using HTTP protocol) and "displays" Web objects
- server
  - Web server sends (using HTTP protocol) objects in response to requests



# HTTP overview

## When you click on a link

1. client initiates TCP connection
   - creates socket to server on port 80
2. server accepts TCP connection from client
3. HTTP messages exchanged between browser (HTTP client) and Web server (HTTP server)
4. TCP connection closed

## Two types of HTTP messages

- request, response

## Stateless

- server maintains no information about past client requests

## Q: Why stateless?

- Stateful protocols are complex
  - storage
    - state must be maintained for potentially many clients
  - server/client crashes
    - views of state may be inconsistent, must be reconciled
  - workaround
    - cookies

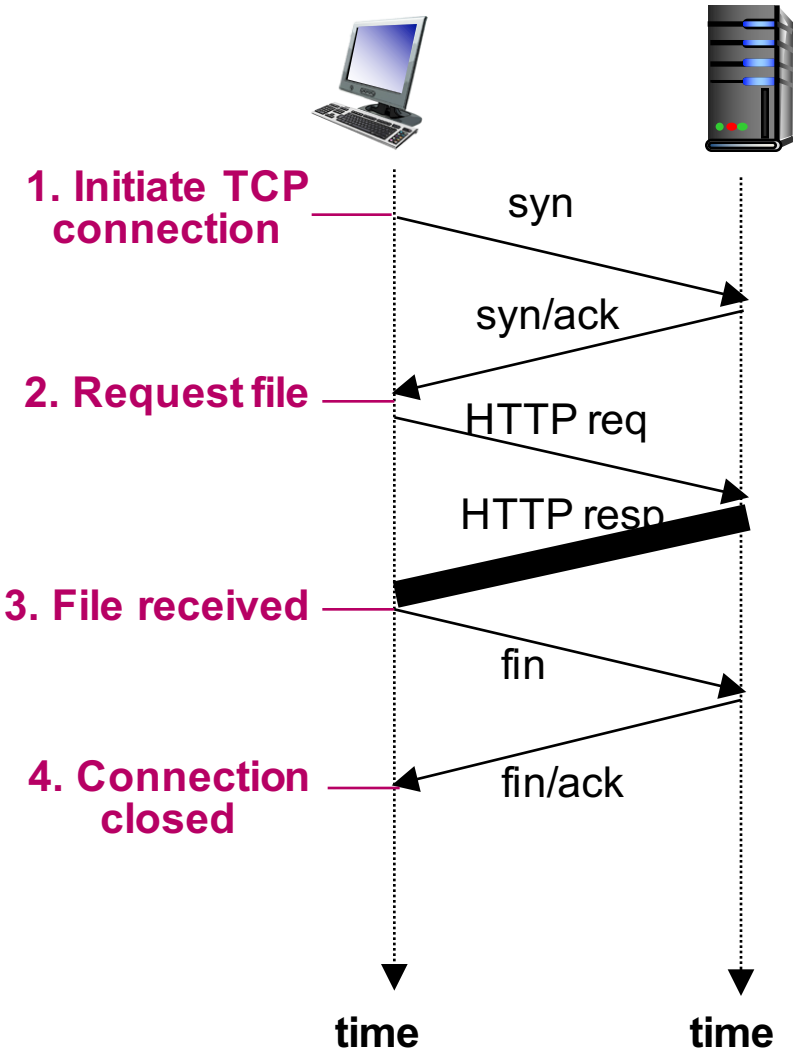
# HTTP overview

## When you click on a link

1. client initiates TCP connection
   - creates socket to server on port 80
2. server accepts TCP connection from client
3. HTTP messages exchanged between browser (HTTP client) and Web server (HTTP server)
4. TCP connection closed

## Two types of HTTP messages

- request, response



# Format of a webpage

Web page consists of objects

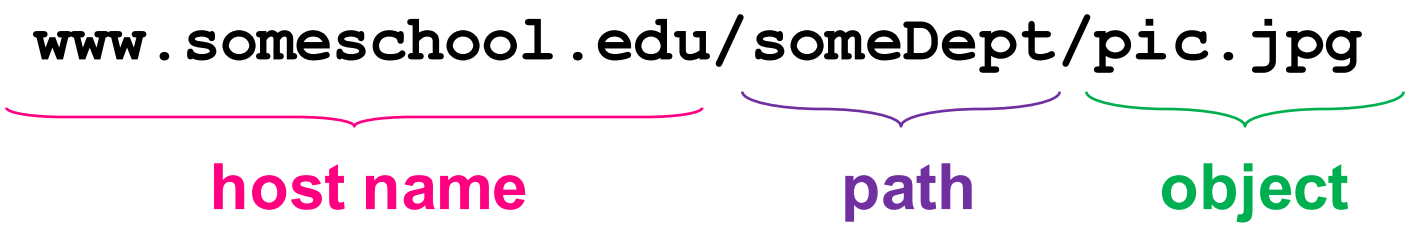
- object can be HTML file, JPEG image, Java applet, audio file,…
- typically includes base HTML-file and several referenced objects

1. index.html

2. pic.jpg
3. HWK.pdf

All 3 objects must be requested from server in order to fully load webpage

Each object is addressable by URL, e.g.,




Q: How do we download multiple objects using HTTP?

# HTTP connections

Two ways to use HTTP requests to get objects from web server

## Non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- for each object, setup and use separate TCP connection
  - downloading multiple objects requires multiple connections
- HTTP/1.0

## Persistent HTTP

- multiple objects can be sent over single TCP connection between client, server
- reuse same TCP connection to download multiple objects
- HTTP/1.1: by default

Q: Which is faster? Which is better?

# Non-persistent HTTP

Suppose user enters URL:

`www.wesleyan.edu/mathcs/index.html`

1a. HTTP client initiates TCP connection to HTTP server (process) at `www.wesleyan.edu` on port 80

1b. HTTP server at host `www.wesleyan.edu` waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates client wants object `mathcs/index.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. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for the 10 referenced jpeg objects referred to in `index.html`

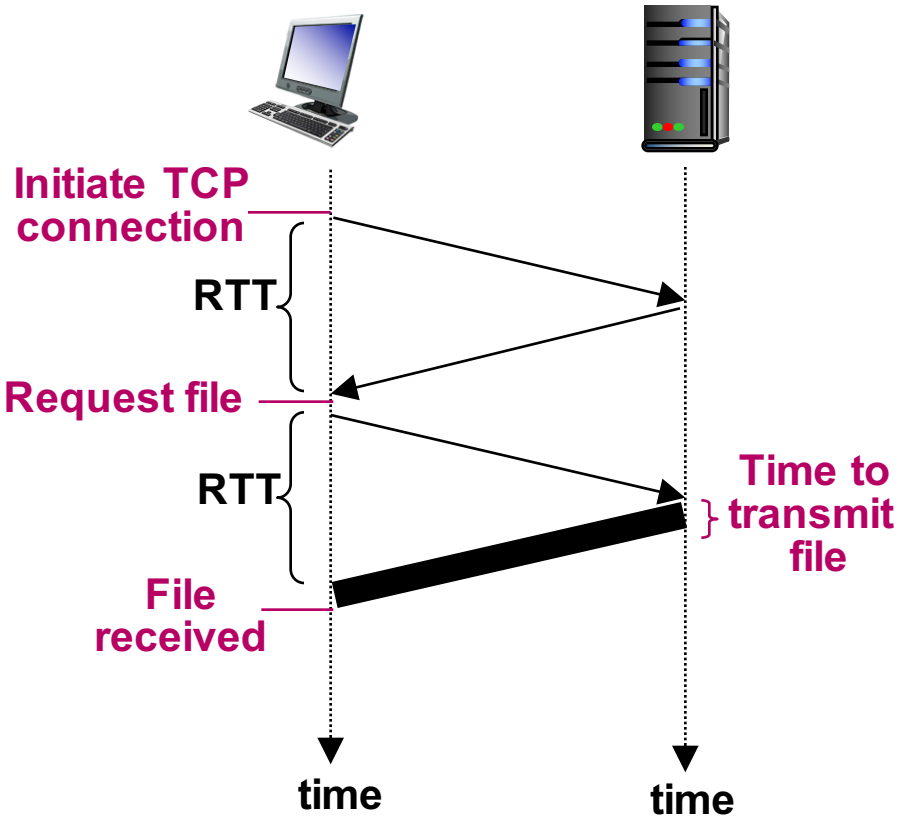
# Non-persistent HTTP: response time

## Round-trip-time (RTT)

- time for small packet to travel from client to server and back

## HTTP response time

- 1 RTT
  - to initiate TCP connection
- 1 RTT
  - for HTTP request and first few bytes of HTTP response to return
- file transmission time




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

# Problems with non-persistent HTTP

Delay and resource usage

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Q: Can we do better?

# Persistent HTTP

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

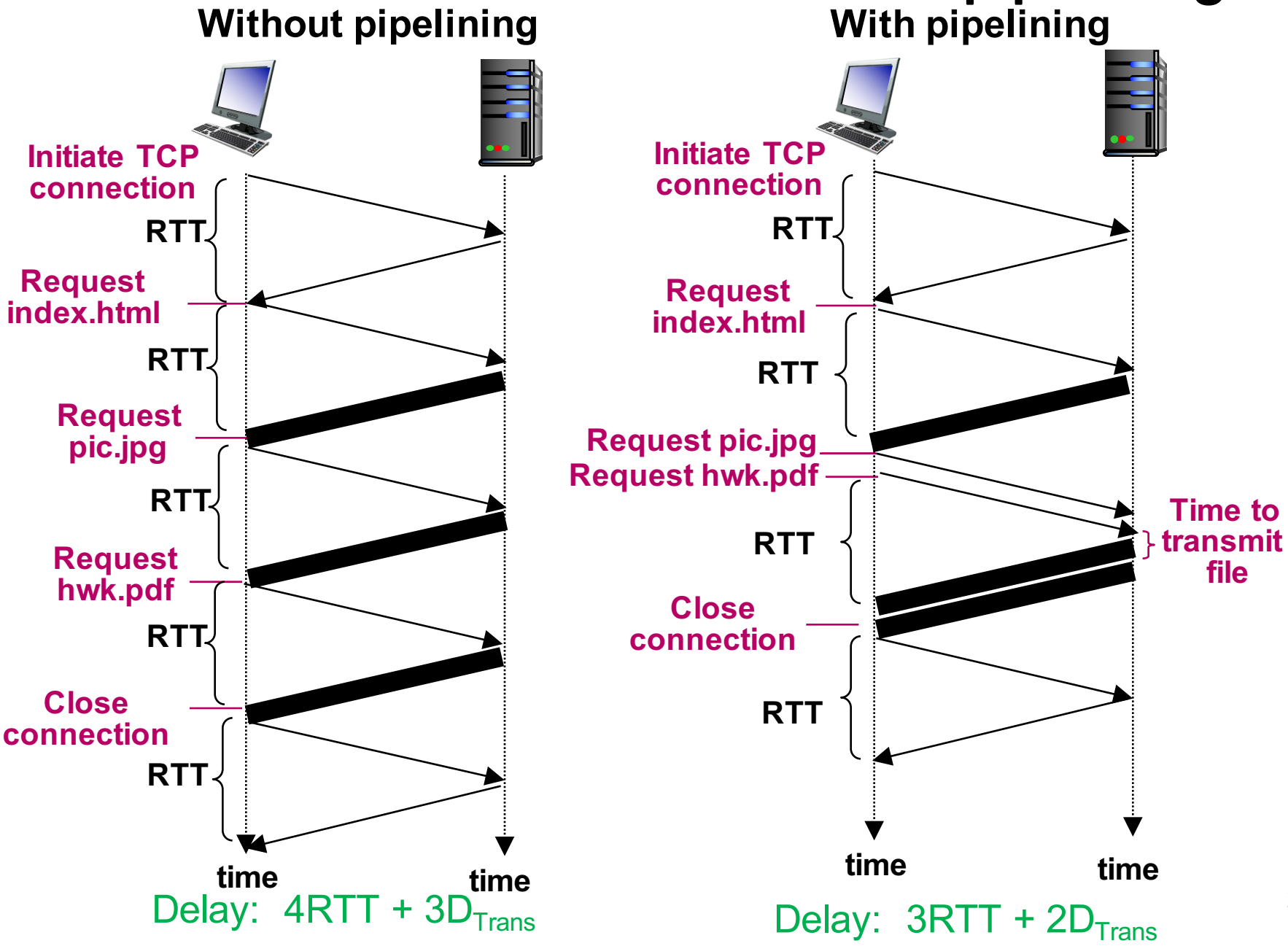
Persistent without pipelining

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

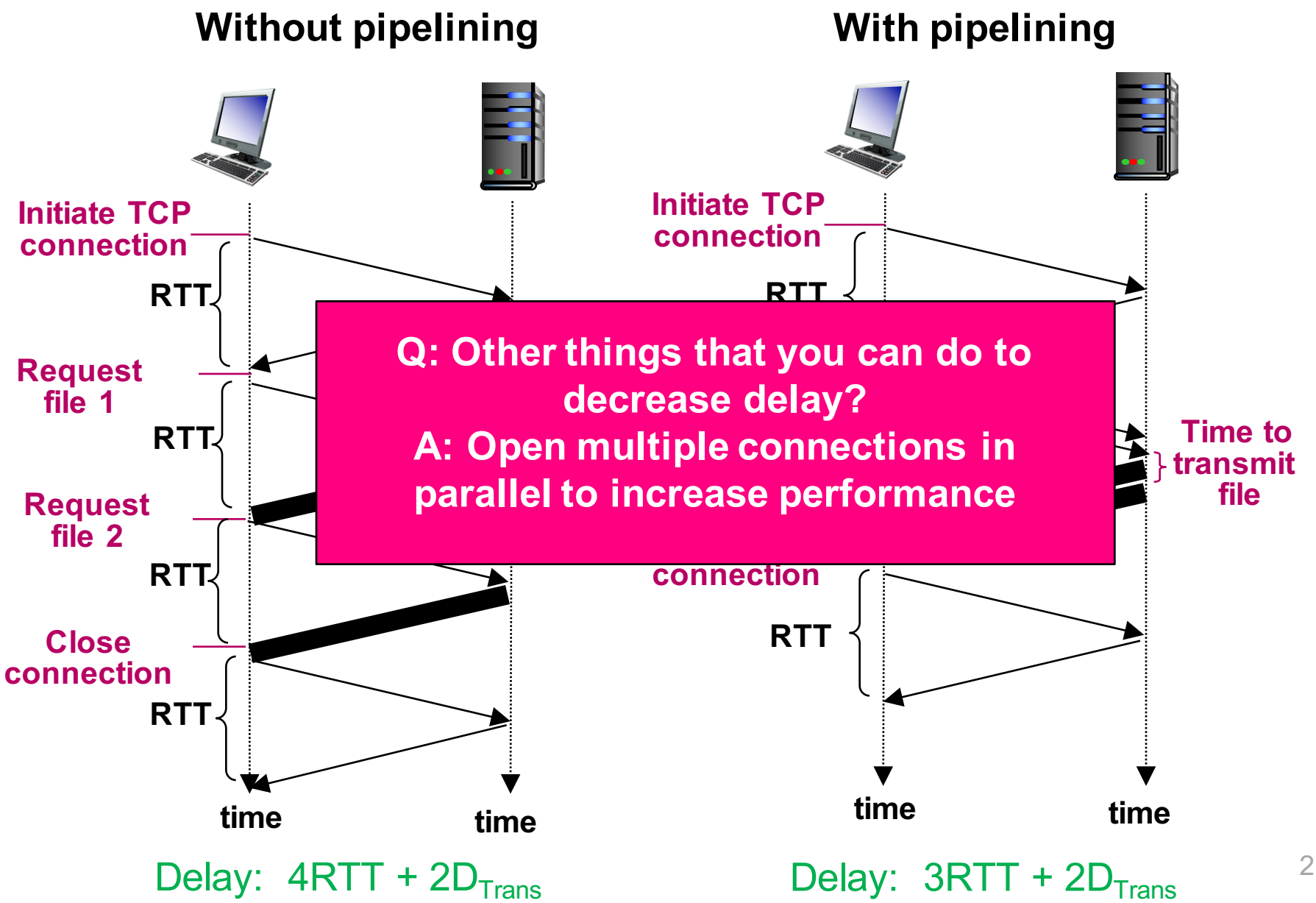
Persistent with pipelining

- client sends requests as soon as it encounters referenced object
- as little as 1 RTT for all referenced objects
- default in HTTP/1.1

# Persistent HTTP: with and without pipelining



# Persistent HTTP: with and without pipelining




Delay: $4RTT + 2D_{Trans}$

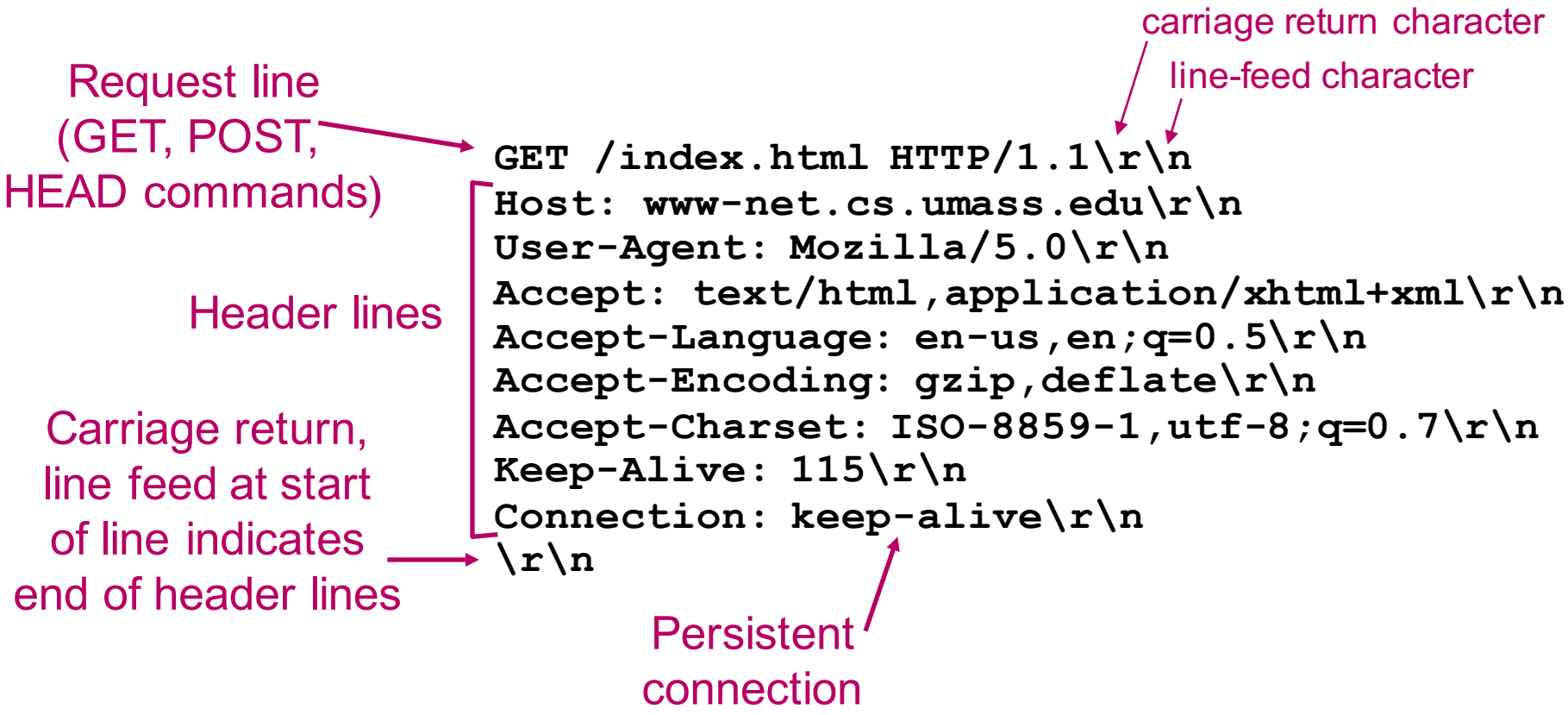
Delay: $3RTT + 2D_{Trans}$

# HTTP Protocol

## REQUESTS, RESPONSES, ERRORS

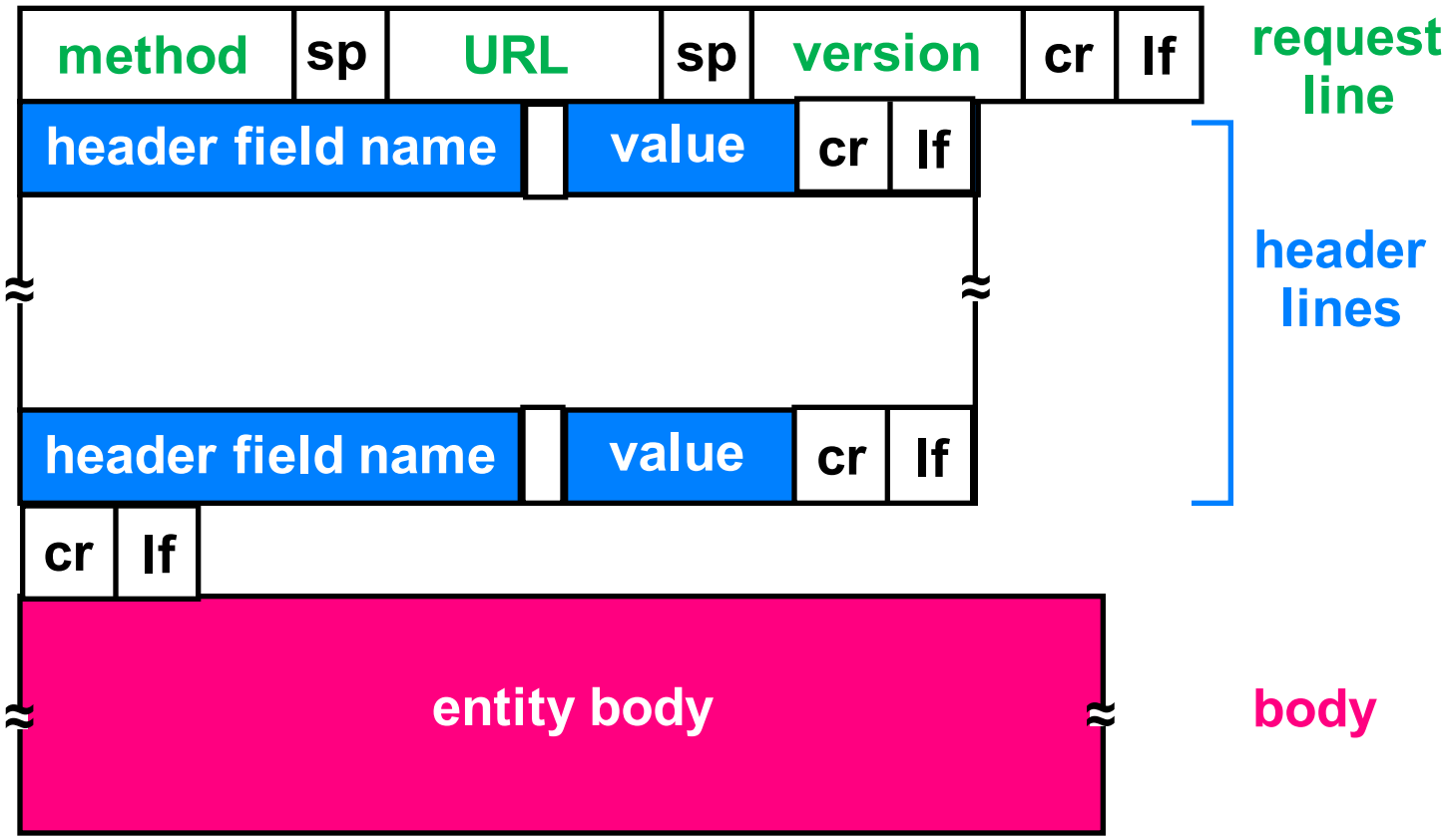
# HTTP request message

ASCII (human-readable format)




Q: What info can server use to fingerprint you, without even using cookies?

# HTTP request message: general format



# 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:

```
www.somesite.com/animalsearch?monkeys&banana
```

# HTTP response message

Status line (protocol status code status phrase)

Header lines

Use to determine end of message

Data, e.g., requested HTML file (may be split across multiple pkts

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

# HTTP response status codes

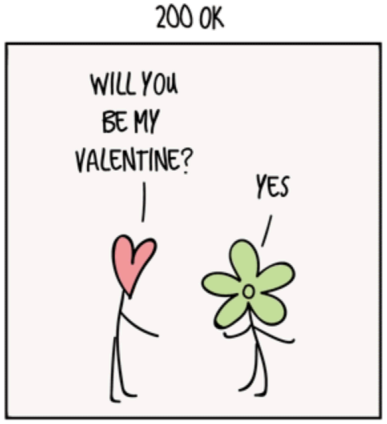
## Status code

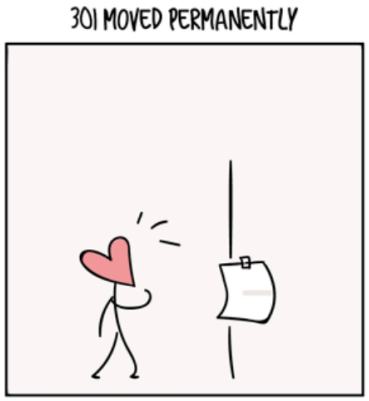
- appears in 1st line in server-to-client response message.

## Some sample codes

- 200 OK
  - request succeeded, requested object later in this 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
- 500 Server error
- 505 HTTP Version Not Supported

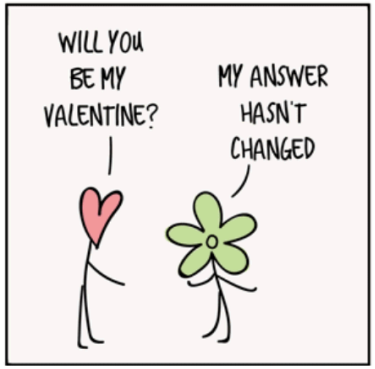
# HTTP status codes as Valentine's Day cartoons



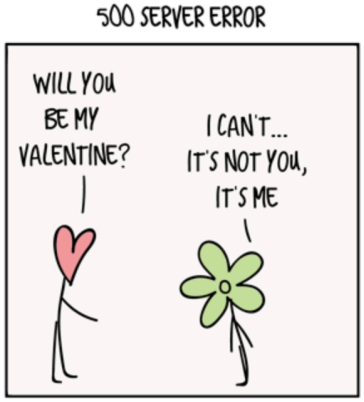










From https://medium.com/@hanilim/http-codes-as-valentines-day-comics-8c03c805faa0

# Try out HTTP (client side) for yourself

1. Open tcp connection using netcat:

```
nc wesleyan.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at wesleyan.edu. Anything typed in will be sent to port 80 at wesleyan.edu

2. type in a GET HTTP request:

```
GET /mathcs/index.html HTTP/1.1
Host: wesleyan.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

# HTTP Protocol

## COOKIES

# User-server state: cookies

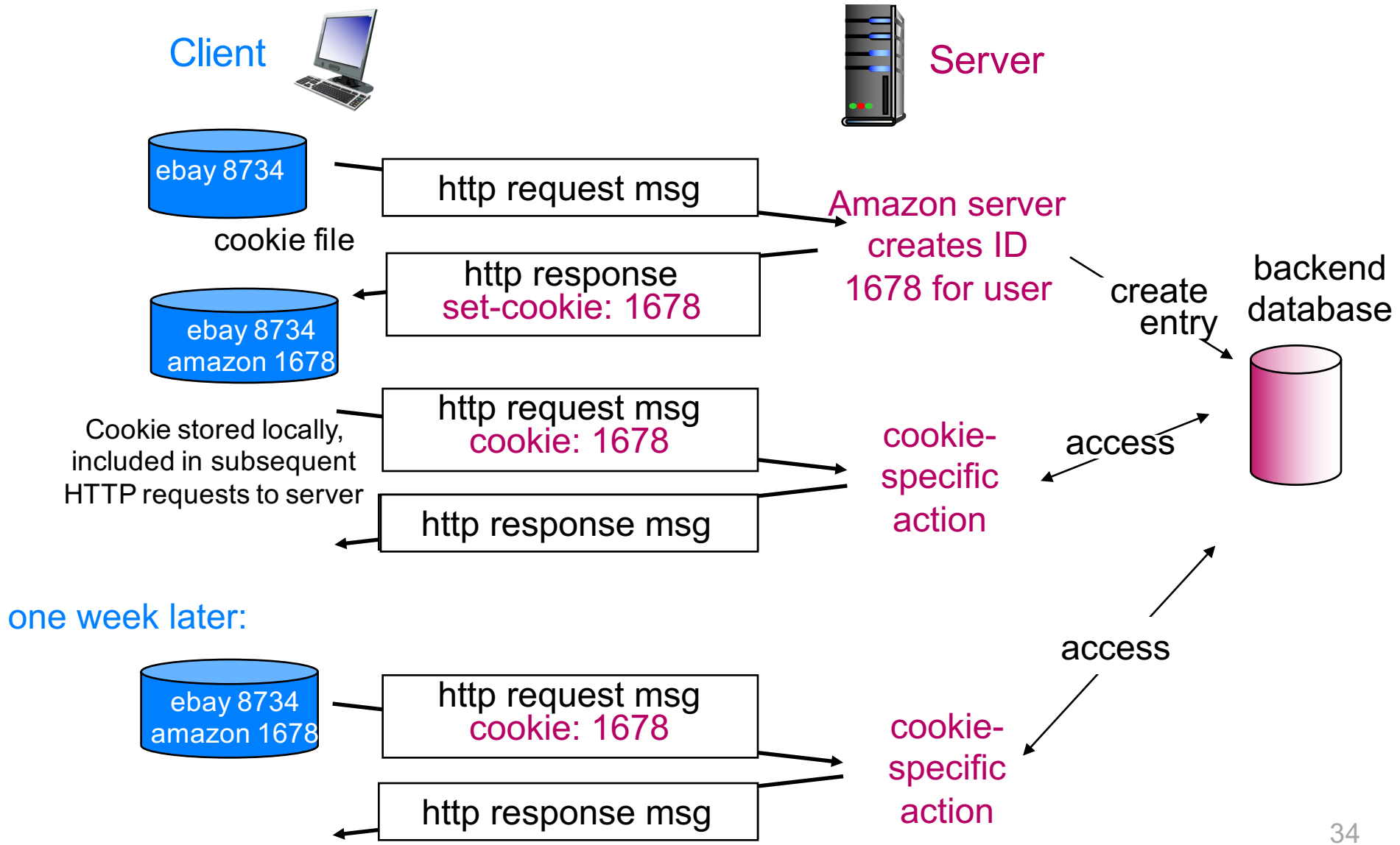
## Many Web sites use cookies

- allow HTTP to store state across multiple interactions by user
- Q: why?
  - distinguish different users
  - authorization
  - shopping cart when not signed in
  - recommendations, preferences
  - user session state (Web e-mail)

## 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

# Cookie example



# Downsides to cookies

## For client

- privacy vulnerabilities
  - cookies permit sites to learn a lot about you
  - suppose cookie set but also signed into google
- track user behavior
- ad tracking (third-party cookies)
  - cookie set by website different than one you are on

## For server

- users can easily delete cookies
- storage needed on server too

## For adversary

- cookie injection attacks
  - man-in-middle HTTP connection
- other attacks

# Look at your own cookies

## Firefox (Mac OS)

~/Library/Application\Support//Firefox/Profiles/h1svqqk3.default/cookies.sqlite

## Chrome (Mac OS)

~/Library/Application\ Support/Google/Chrome/Safe\ Browsing\ Cookies

## Use sqlite to parse file.

- may want to make a copy of file in your home dir to open

## Is your browser safe from tracking?

- https://panopticlick.eff.org/