ELEC / COMP 177 – Fall 2013

Computer Networking → Application Layer (HTTP)

Some slides from Kurose and Ross, *Computer Networking*, 5th Edition

Upcoming Schedule

- **Project #1**
  - Starts next Thursday
  - **Is your Linux environment all ready?**
  - **Bring your laptop – Work time after quick discussion of project goals**
  - **Two weeks – Don't delay!**

Application Layer

Recap – Network Model

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Recap – Physical Layer

Application Layer

Transport Layer

Network Layer

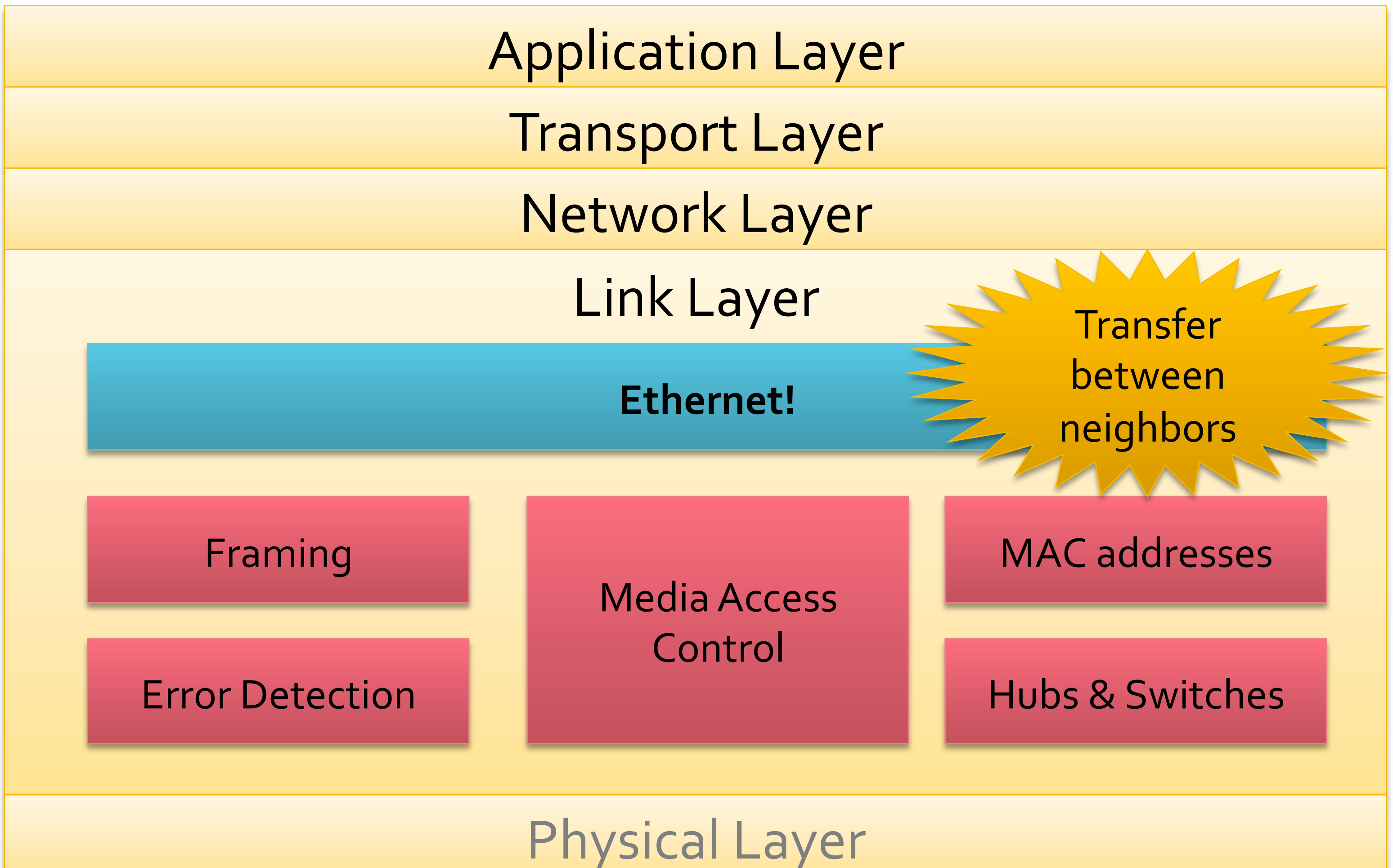
Link Layer

Physical Layer

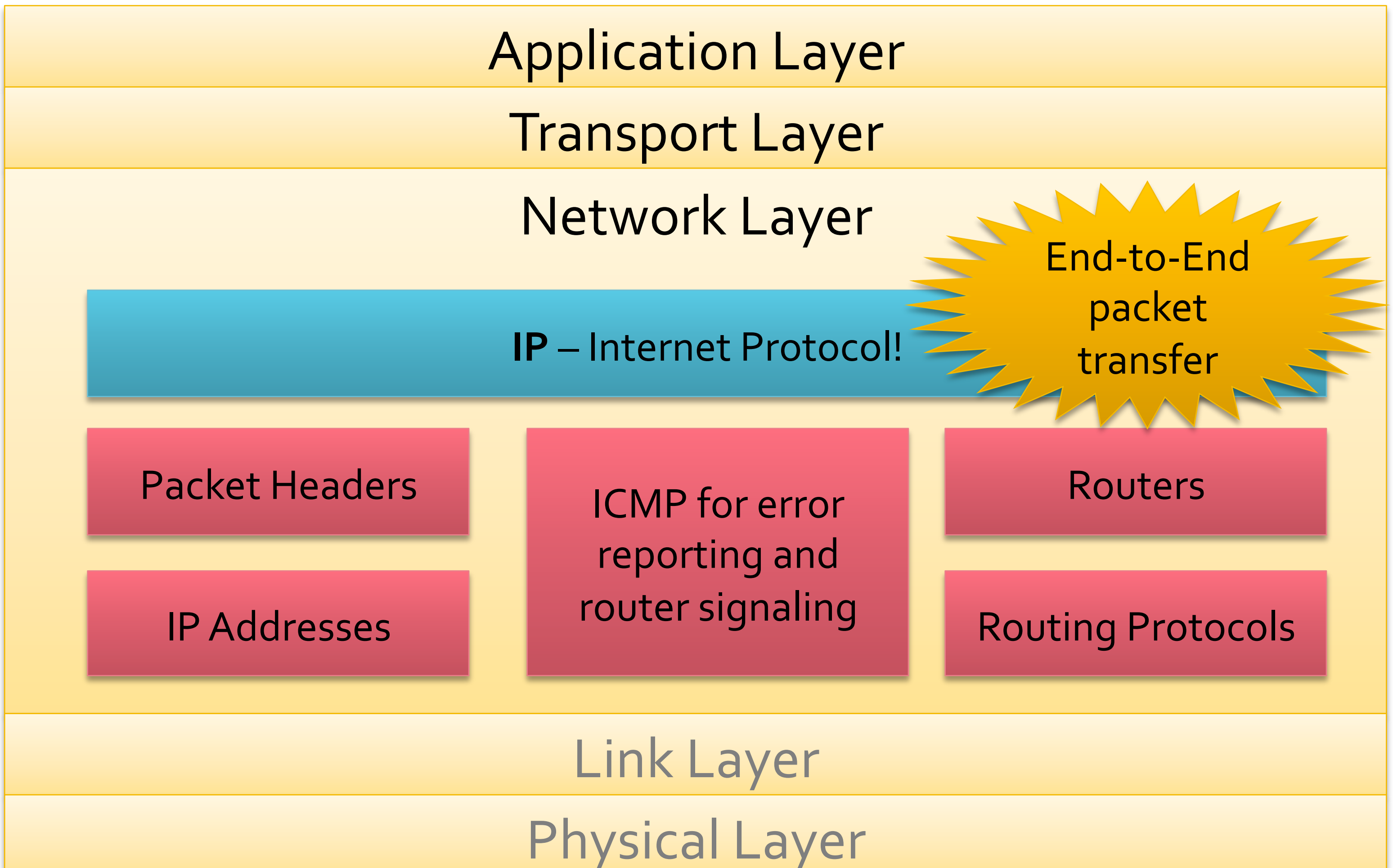
"Bits on a wire"

Encoding schemes fight:
attenuation
distortion
clock skew

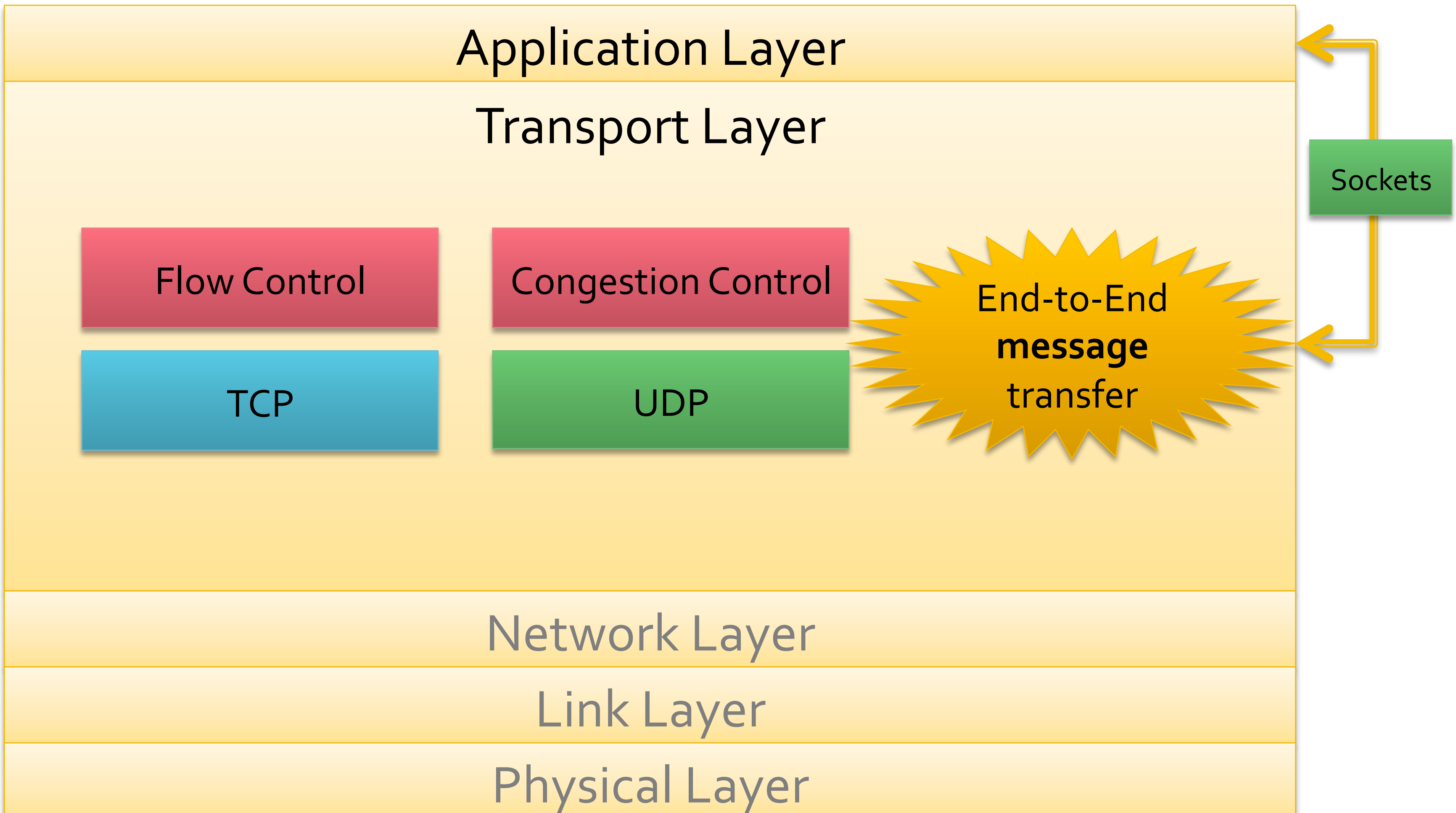
Recap – Link Layer



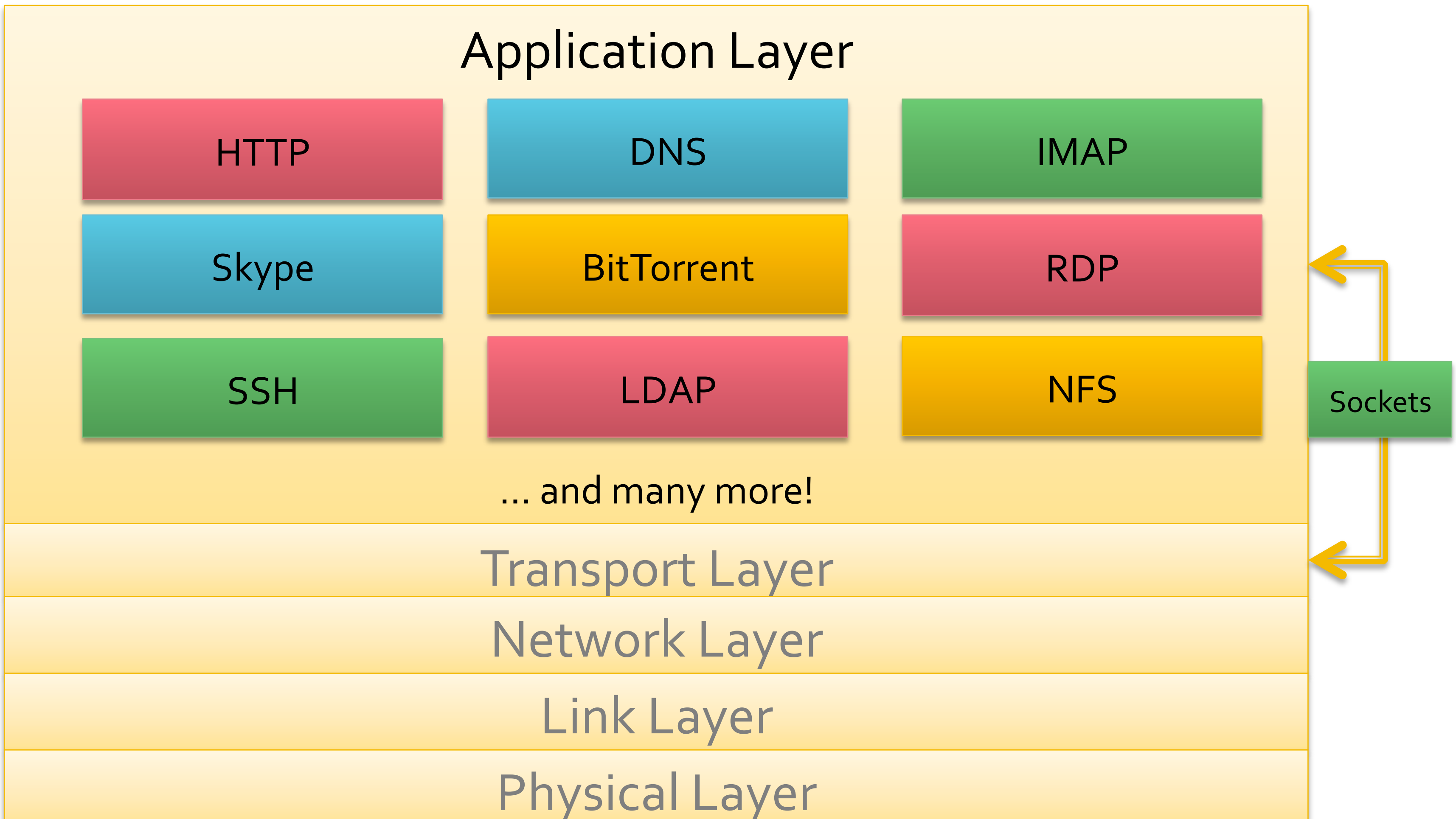
Recap – Network Layer



Recap – Transport Layer



Introducing the Application Layer



Topics

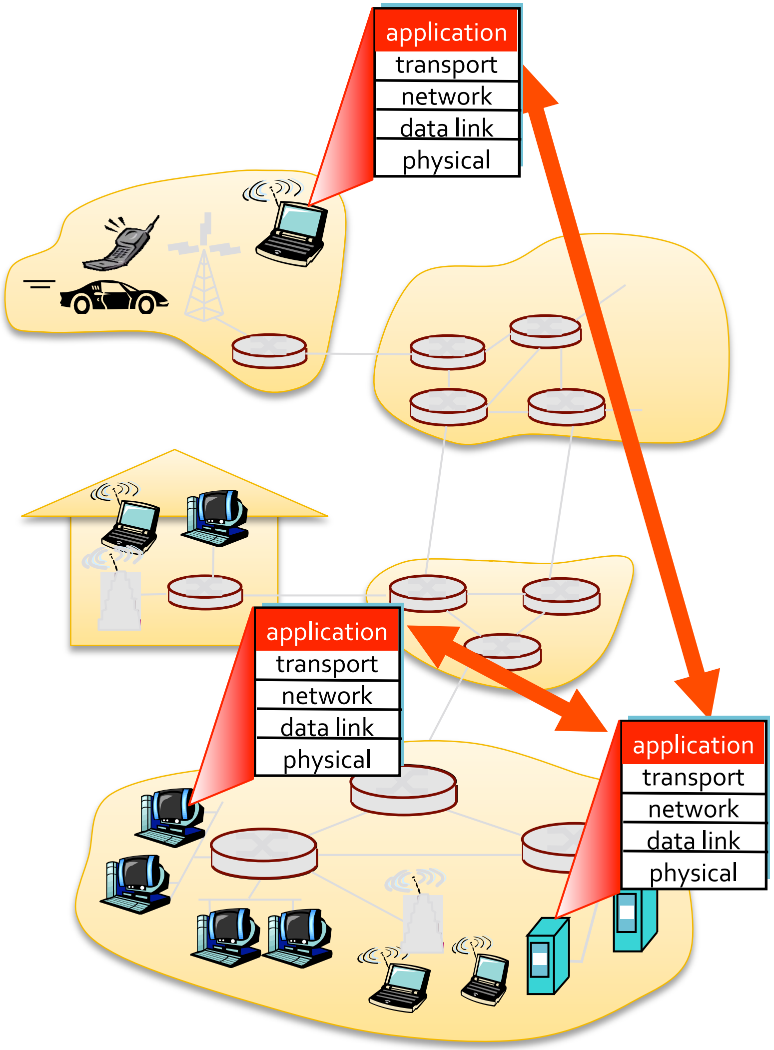
- Transport-layer service models
  - TCP and UDP
- Communication paradigms
  - Client-server
  - Peer-to-peer
- Examine popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- Program network applications
  - Socket API

Network Applications

- What programs do you run that use the Internet?

Creating a Network App

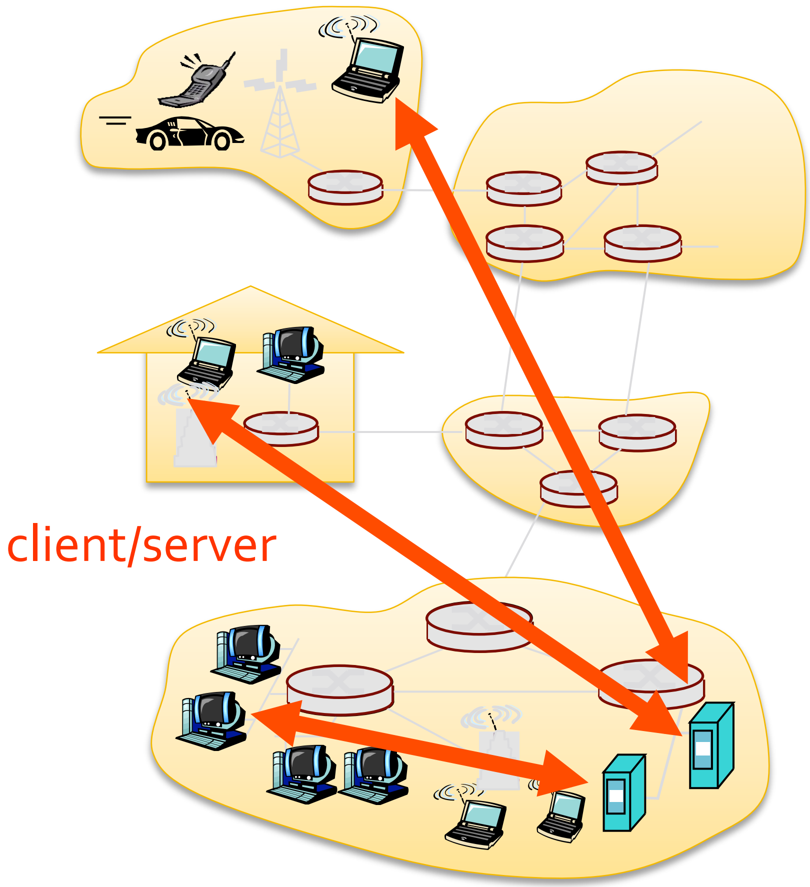
- Write programs that
  - Run on (different) end systems
  - Communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for rapid app development and distribution



Application architectures

- Client-server
  - Including data centers / cloud computing
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-Server Architecture



- Server:
  - Always-on host
  - Permanent IP address
  - Lots of bandwidth
  - Server farms for scaling
- Clients:
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

The Datacenter

The Datacenter

Google Datacenter (1 of many…)

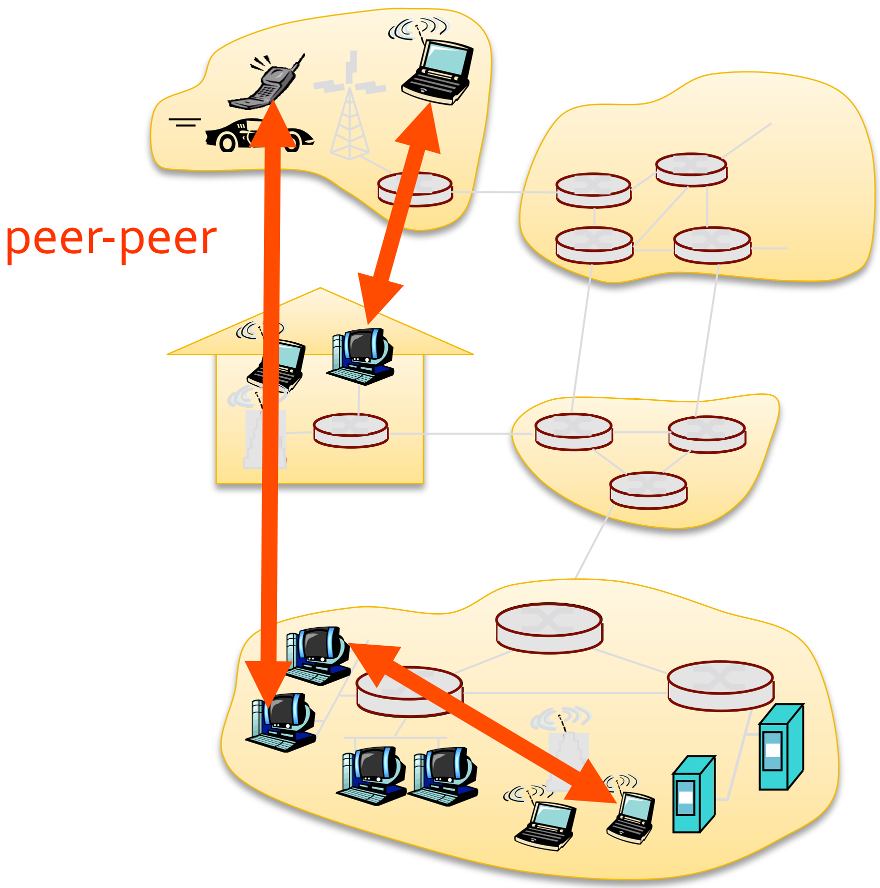
Microsoft Datacenter (Dublin, Ireland)

NSA Datacenter (Bluffdale, Utah. 2+ Billion $$)

Pure P2P architecture

- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

- No central point of failure
- **Highly scalable but difficult to manage**



P2P "Datacenter"

Hybrid of Client-Server and P2P

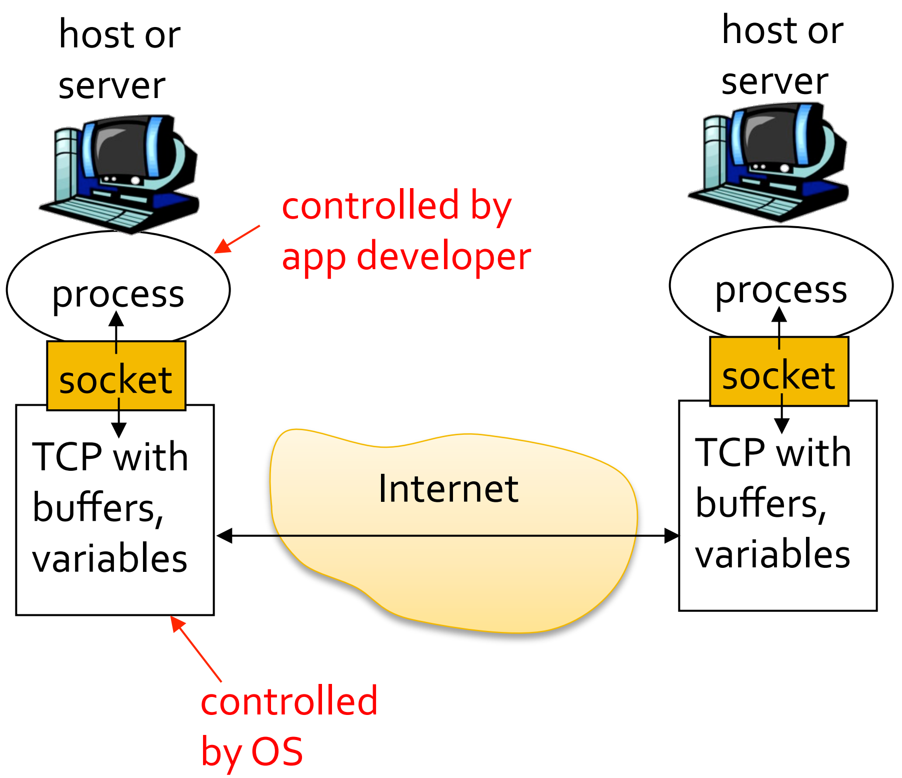
- Skype
  - Voice-over-IP P2P application
  - Centralized server: finding address of remote party
  - Client-client connection: direct (not through server)
- Instant messaging
  - Chatting between two users is P2P
  - Centralized service: client presence detection/location
    - User registers its IP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

Processes Communicating

- Process: program running within a host
  - Within same host, two processes communicate using inter-process communication (defined by OS)
  - Processes in different hosts communicate by exchanging messages
- **Client** process: process that initiates communication
- **Server** process: process that waits to be contacted
- Applications with P2P architectures have both client *and* server processes!

What is a Socket?

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Transport infrastructure on other side of door carries message to socket at receiving process
  - **Imagine you are just writing to a file…**
- API allow customization of socket
  - Choose transport protocol
  - Choose parameters of protocol



Application-Layer Protocol

- Sockets just allow us to send raw messages between processes on different hosts
  - Transport service takes care of moving the data
- **What** exactly is sent is up to the application
  - An application-layer protocol
  - HTTP, IMAP, Skype, etc…

Application-Layer Protocol

- Both the client and server speaking the protocol must agree on
  - 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

Application-Layer Protocol

- **Public-domain** protocols:
  - Defined in RFCs (Request for Comment)
  - Allows for interoperability
  - Examples: HTTP, SMTP, BitTorrent
- **Proprietary** protocols
  - Examples: Skype

Transport Service

- What kind of transport service do applications need?
- **Data loss – OK or forbidden?**
  - Some apps can tolerate some loss
  - Other apps requires 100% reliable data transfer
- **Latency – OK, or bad?**
  - Some apps require low delay to be effective
- **Throughput**
  - Some apps require minimum amount of throughput to be effective
  - Other apps (“elastic apps”) utilize whatever throughout is available
- **Security?**
  - Some apps require encyption

Transport Service Requirements for Common Apps

What do you think?

Application	Data Loss? (OK or not?)	Throughput? (Min required or elastic?)	Time Sensitive? (Low delay required?)
File transfer			
Email			
Web pages			
Real-time audio / video			
Stored audio/video			
Gaming			
Instant messaging			

Transport Service Requirements for Common Apps

Application	Data Loss? (OK or not?)	Throughput? (Min required or elastic?)	Time Sensitive? (Low delay required?)
File transfer	No data loss	Elastic	"Normal" delay OK
Email	No data loss	Elastic	"Normal" delay OK
Web pages	No data loss	Elastic	"Normal" delay OK
Real-time audio / video	Loss tolerant	Minimum	Time sensitive
Stored audio/video	Loss tolerant	Minimum	"Normal" delay OK
Gaming	No data loss	Minimum	Time sensitive
Instant messaging	No data loss	Elastic	"Normal" delay OK

Internet Transport Protocols

TCP SERVICE

- Connection-oriented
  - Setup required between client and server processes
- Reliable transport between sending and receiving process
- Flow control
  - Sender won't overwhelm **receiver**
- Congestion control
  - Sender won't overwhelm the **network**
- Does not provide
  - Timing, minimum throughput guarantees, security

UDP SERVICE

- Unreliable data transfer between sending and receiving process
- Does not provide
  - Connection setup
  - Reliability
  - Flow control
  - Congestion control
  - Timing
  - Throughput guarantee
  - Security

Why bother with UDP then?

Transport Service Requirements for Common Apps

Application	Data Loss? (OK or not?)	Throughput? (Min required or elastic?)	Time Sensitive? (Low delay required?)	Transport Protocol
File transfer	No data loss	Elastic	"Normal" delay OK	TCP
Email	No data loss	Elastic	"Normal" delay OK	TCP
Web pages	No data loss	Elastic	"Normal" delay OK	TCP
Real-time audio / video	Loss tolerant	Minimum	Time sensitive	UDP
Stored audio/video	Loss tolerant	Minimum	"Normal" delay OK	TCP or UDP
Gaming	No data loss	Minimum	Time sensitive	UDP
Instant messaging	No data loss	Elastic	"Normal" delay OK	TCP

Hypertext Transport Protocol (HTTP)

Web and HTTP

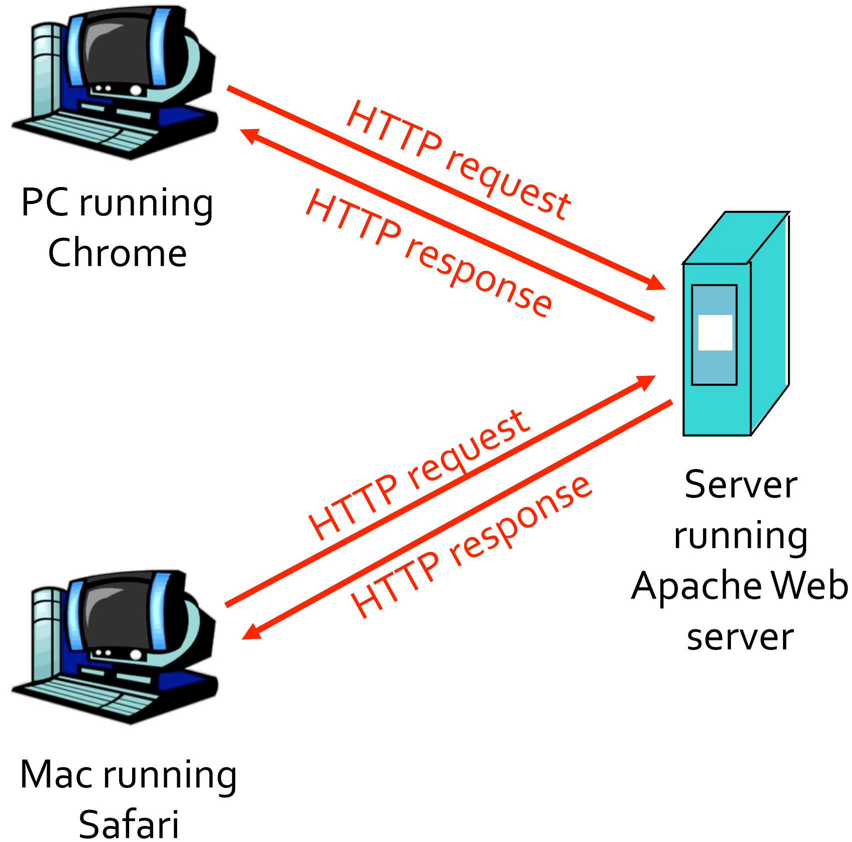
- Web **page** consists of base HTML file and (potentially) many referenced **objects**
  - HTML file, JPEG image, Flash video, …
- Each object is addressable by a **URL**
- Example URL:

```
www.somecompany.com/someDept/image.png
```

host name — path name

Hypertext Transfer Protocol Overview

- **HTTP** is the *application layer protocol* for the web
- It is how the client and server communicate
- Client/server model
  - **Client**: browser that requests, receives, "displays" Web objects
  - **Server**: Web server sends objects in response to requests



HTTP Overview

Client

Server

Client initiates TCP connection (creates socket) 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 by client or server

HTTP Overview

- HTTP is "stateless"
- Server maintains no information about past client requests

- Why no state?
  - Protocols that maintain "state" are complex!
  - Past history (state) must be maintained
  - If server/client crashes, their views of "state" may be inconsistent and must be reconciled

HTTP Connections

- **Nonpersistent HTTP**
  - At most one object is sent over a TCP connection.
- **Persistent HTTP**
  - Multiple objects can be sent over single TCP connection between client and server.

Nonpersistent HTTP

Suppose user enters URL `www.someCompany.com/someDept/index.html`
(contains text and references to 10 jpeg images)

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

1b. HTTP server at host `www.someCompany.com` 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 `someDept/index.html`

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

time

Nonpersistent HTTP

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

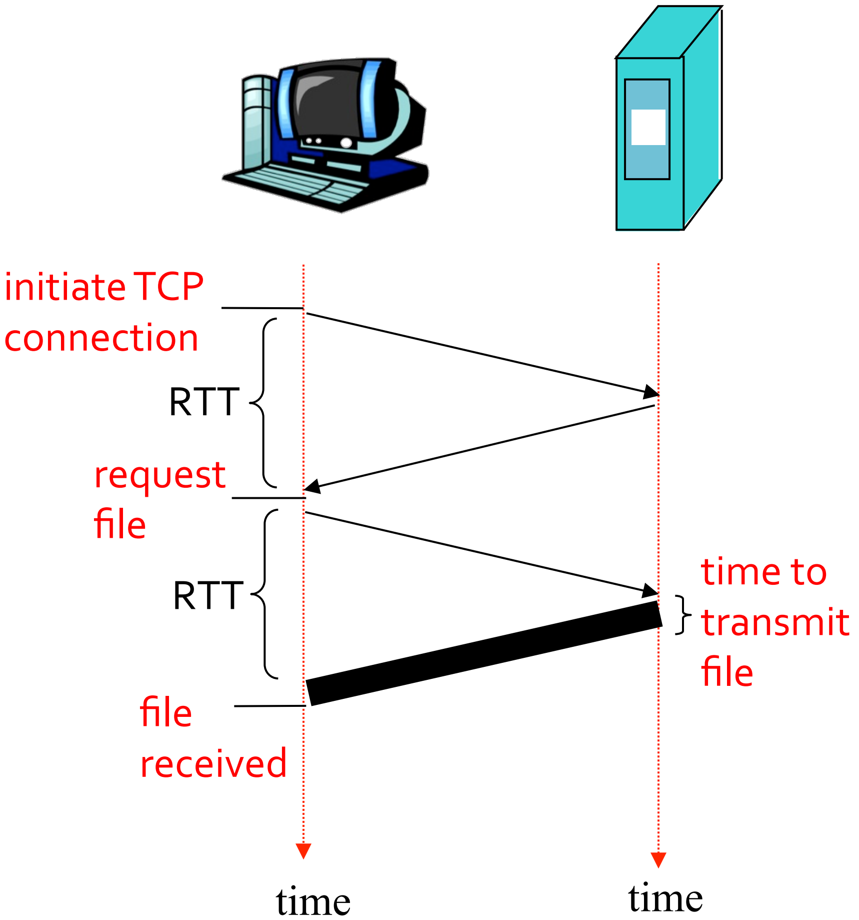
time

6. Steps 1-5 repeated for each of 10 jpeg objects

Why is this approach considered slow?

Non-Persistent HTTP: Response Time

- **RTT (Round Trip Time):**
  - Time for a small packet to travel from client to server and back.
- **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+transmit time (per object!)**



Persistent vs Non-Persistent HTTP

- **Non-Persistent HTTP issues**
  - Requires 2 RTTs per object
  - OS overhead for each TCP connection
  - Browsers often open parallel TCP connections to fetch referenced objects (more overhead)
- **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
  - As little as one RTT for all the referenced objects

HTTP Request Message

- HTTP request messages
  - Used to send data from client to server
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

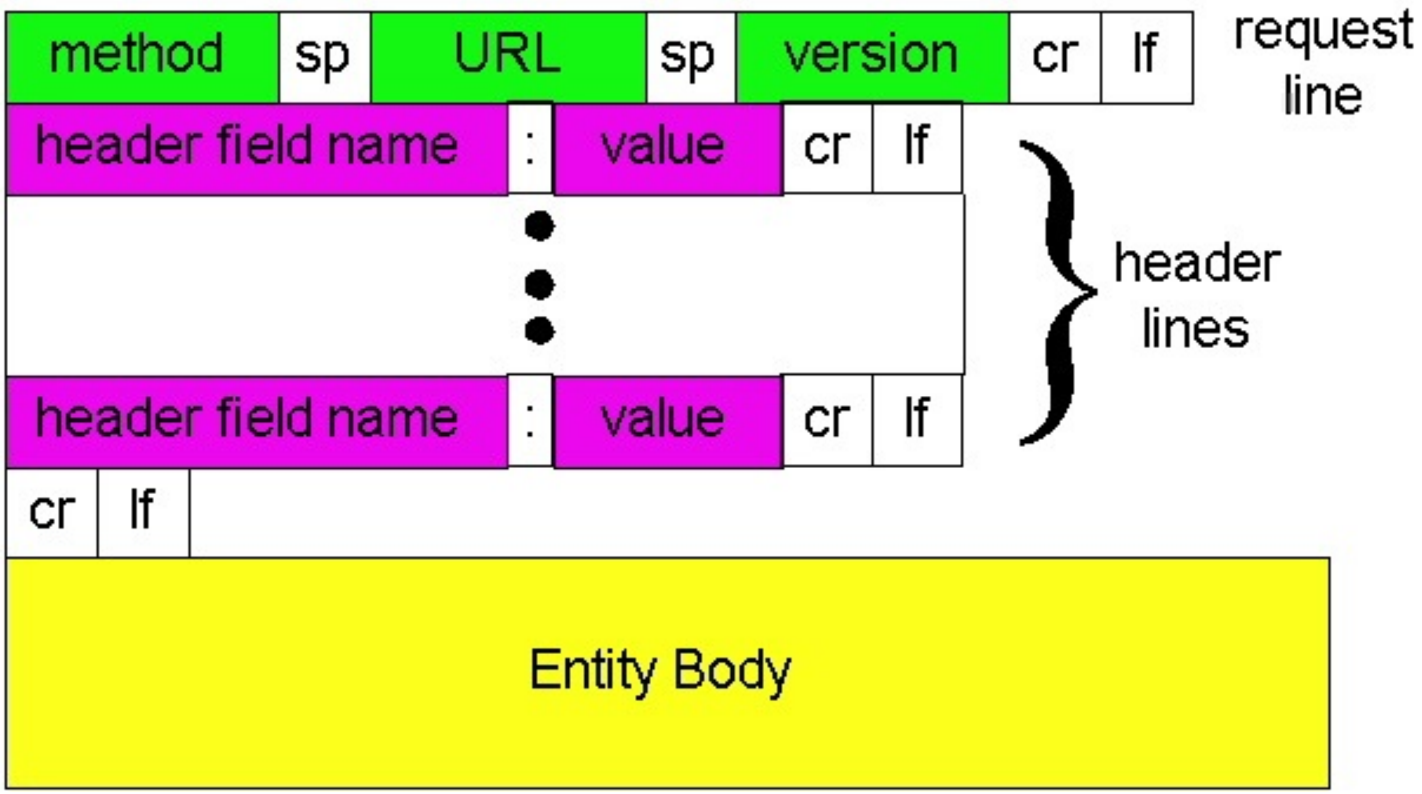
header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.somecompany.com
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP Request Message: General Format



Demo Time!

TELNET DEMO

- Manual file request

WIRESHARK DEMO

- Filtering on protocol headers
- Viewing request/response
- HTTP conversation analysis of all captured packets

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.somecompany.com/page.php?variable1=testData
```

Method Types

- **HTTP/1.0**
  - GET
  - POST
  - HEAD
    - asks server to leave requested object out of response
- **HTTP/1.1**
  - GET, POST, HEAD
  - PUT
    - uploads file in entity body to path specified in URL field
  - DELETE
    - deletes file specified in the URL field

HTTP Response Message

Used to send data from server to client

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP Response Status Codes

In first line in server->client response message.
A few 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 (Location:)

`400 Bad Request`

- request message not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

Trying out HTTP (Client side) for Yourself

1. Telnet to your favorite Web server:

```
telnet www.google.com 80
```

Opens TCP connection to port 80 (default HTTP server port) at `www.google.com`
Anything typed in sent to port 80 at `www.google.com`

2. Type in a GET HTTP request:

```
GET /about/ HTTP/1.1
Host: www.google.com
```

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

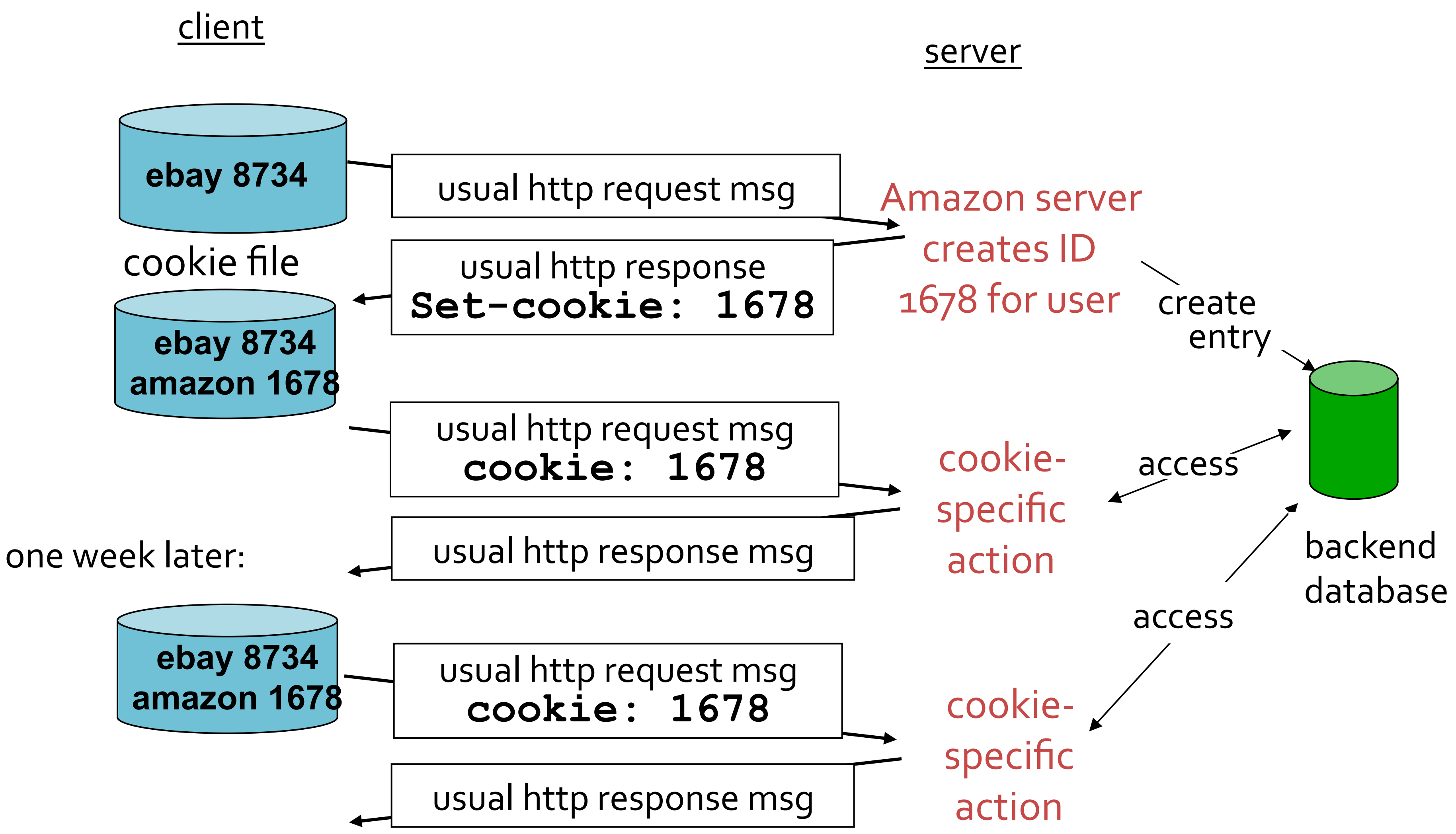
3. Look at response message sent by HTTP server!

User-Server State: Cookies

- HTTP is stateless
  - State is sometimes desired
- Solution? Cookies!
  - Created when you visit a site for the first time
  - When initial HTTP requests arrives at site, site creates:
    - Unique ID
    - Entry in backend database for ID

- Four components
  1. **Cookie header line** of HTTP *response* message
  2. Cookie header line in HTTP *request* message
  3. Cookie file kept on **user's host**, managed by user's browser
  4. **Back-end database** at Web site

Cookies: keeping "state"



Cookies

- Cookies store **Key -> Value pairs**
- What can I do with this?
  - Authorization, shopping carts, user session state (Web e-mail)
- How to keep "state":
  - Protocol endpoints (sender/receiver) both have to maintain data over multiple transactions
  - Cookies: http messages carry state
- Tension between users and websites
  - **Websites:** If I can track you, I can make money from marketers
  - **Users:** I don't want to be tracked (and thus can delete cookies)

Introducing the EverCookie

http://arstechnica.com/web/news/2010/09/evercookie-escalates-the-zombie-cookie-war-by-raising-awareness.ars

EverCookie

- **Clings to your computer – hard to remove**
- Stores a user ID and cookie data in eight different places!
  - Standard HTTP cookies
  - Flash cookies
  - RGB values of force-cached PNGs
  - Your Web history
  - Several HTML5 local storage features
  - Silverlight
  - Java
- See http://samy.pl/evercookie/

Conditional GET

- How do I know if the cache is up-to-date?
  - Solution: **Conditional Get**
  - 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`