ELEC / COMP 177 – Fall 2011

Computer Networking
→ Application Layer (HTTP)

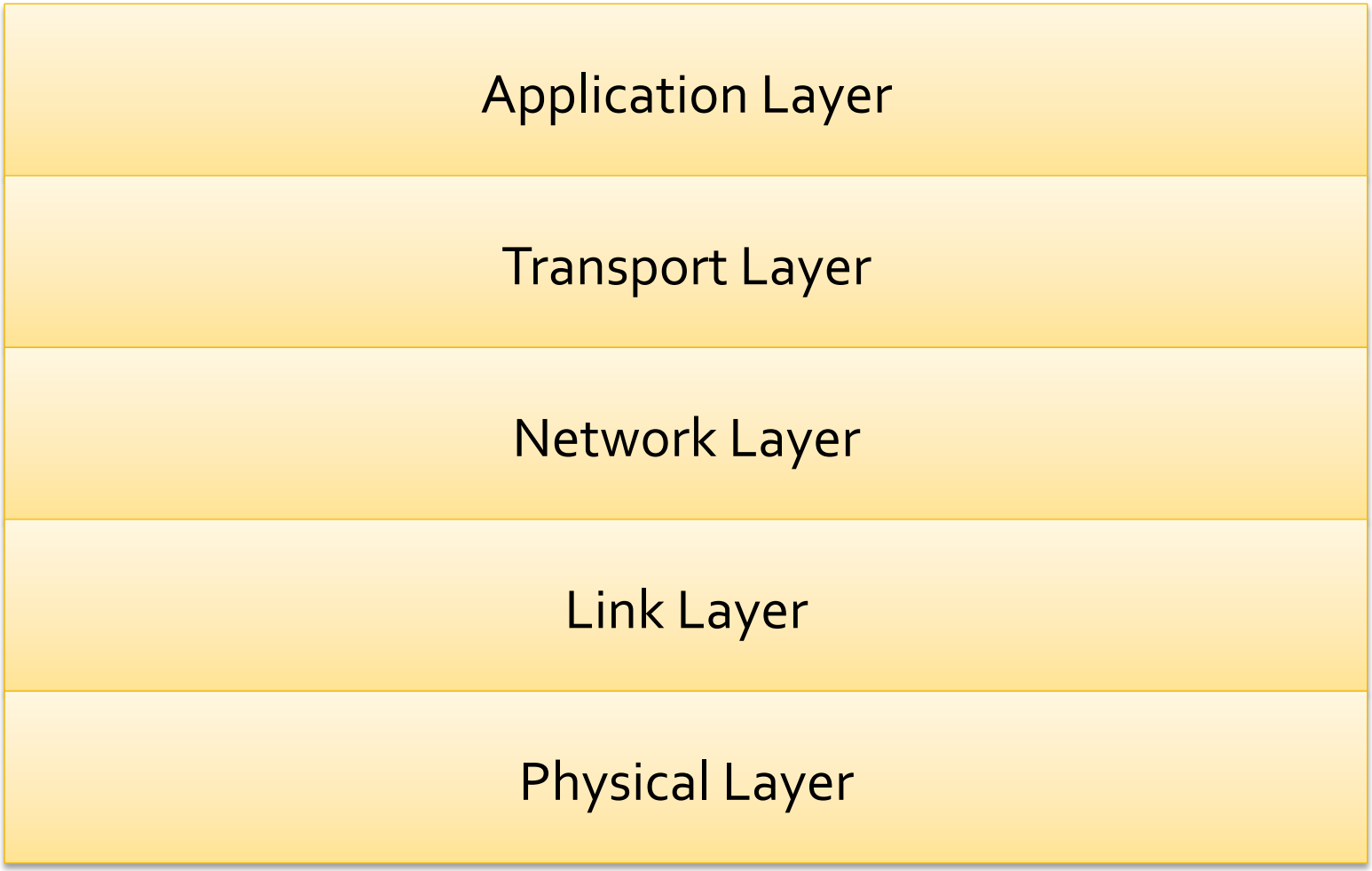
Some slides from Kurose and Ross, *Computer Networking*, 5th Edition

Upcoming Schedule

- **Homework #1**
  - Assigned today
  - Due in one week
  - Application layer: DNS, HTTP, protocols
    - **Recommend you start early so we can discuss questions on Tuesday**
  - Submit PDF file online via Sakai
- **Lab #1**
  - Today (2pm) in Baun 212

Application Layer

Recap – Network Model



Recap – Physical Layer

Application Layer

Transport Layer

Network Layer

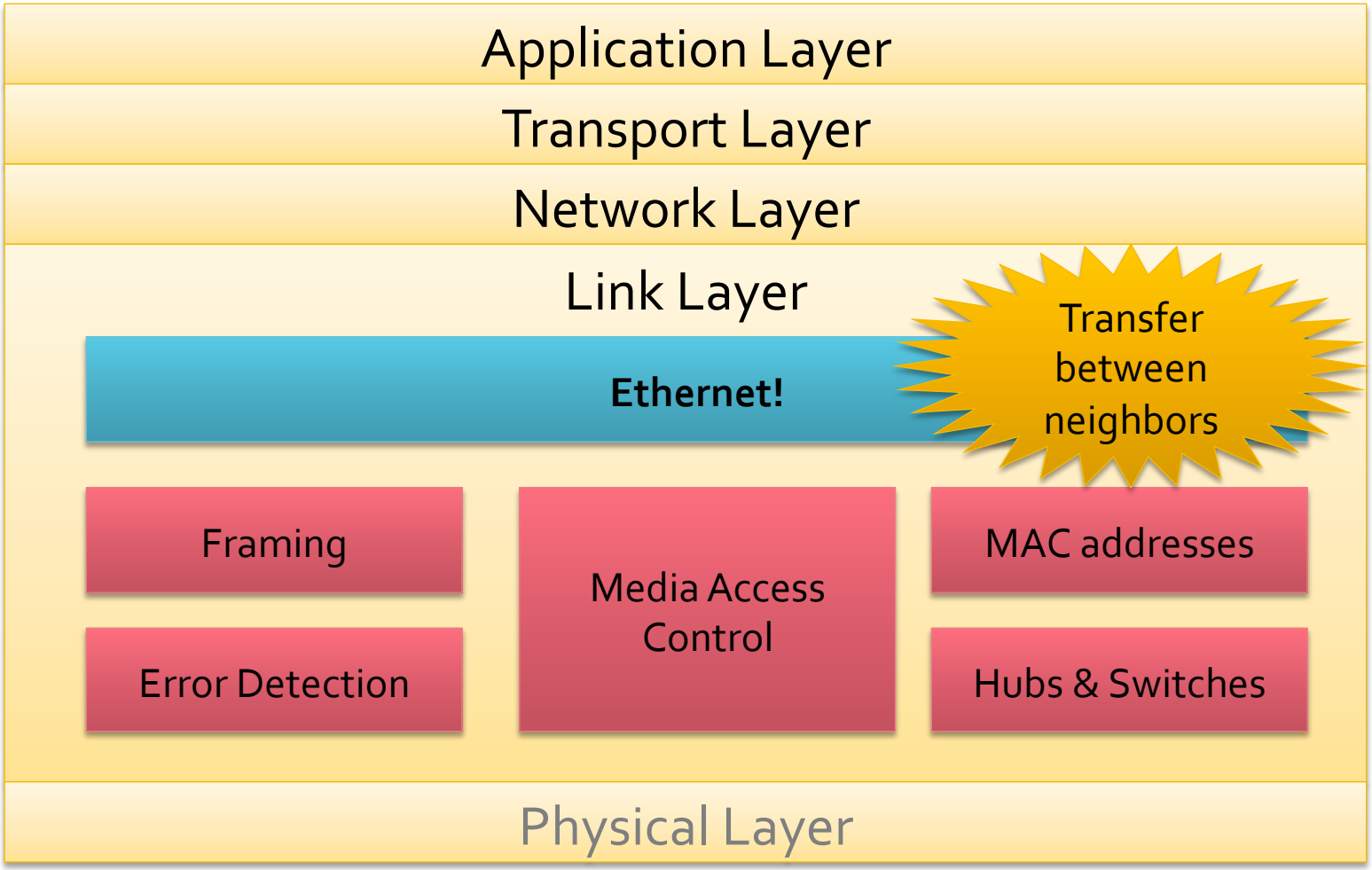
Link Layer

Physical Layer

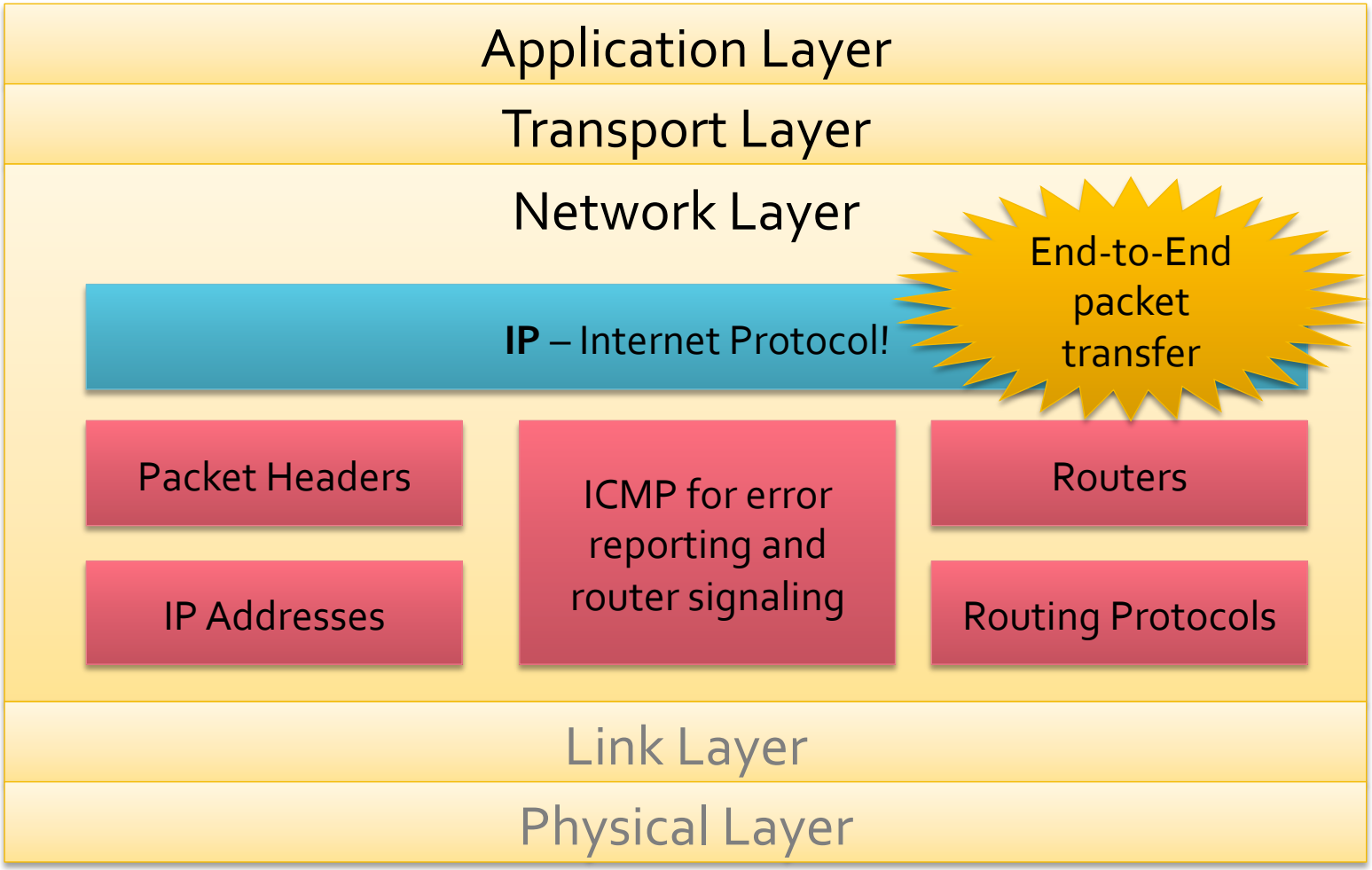
"Bits on a wire"

Encoding schemes fight:
attenuation
distortion
clock skew

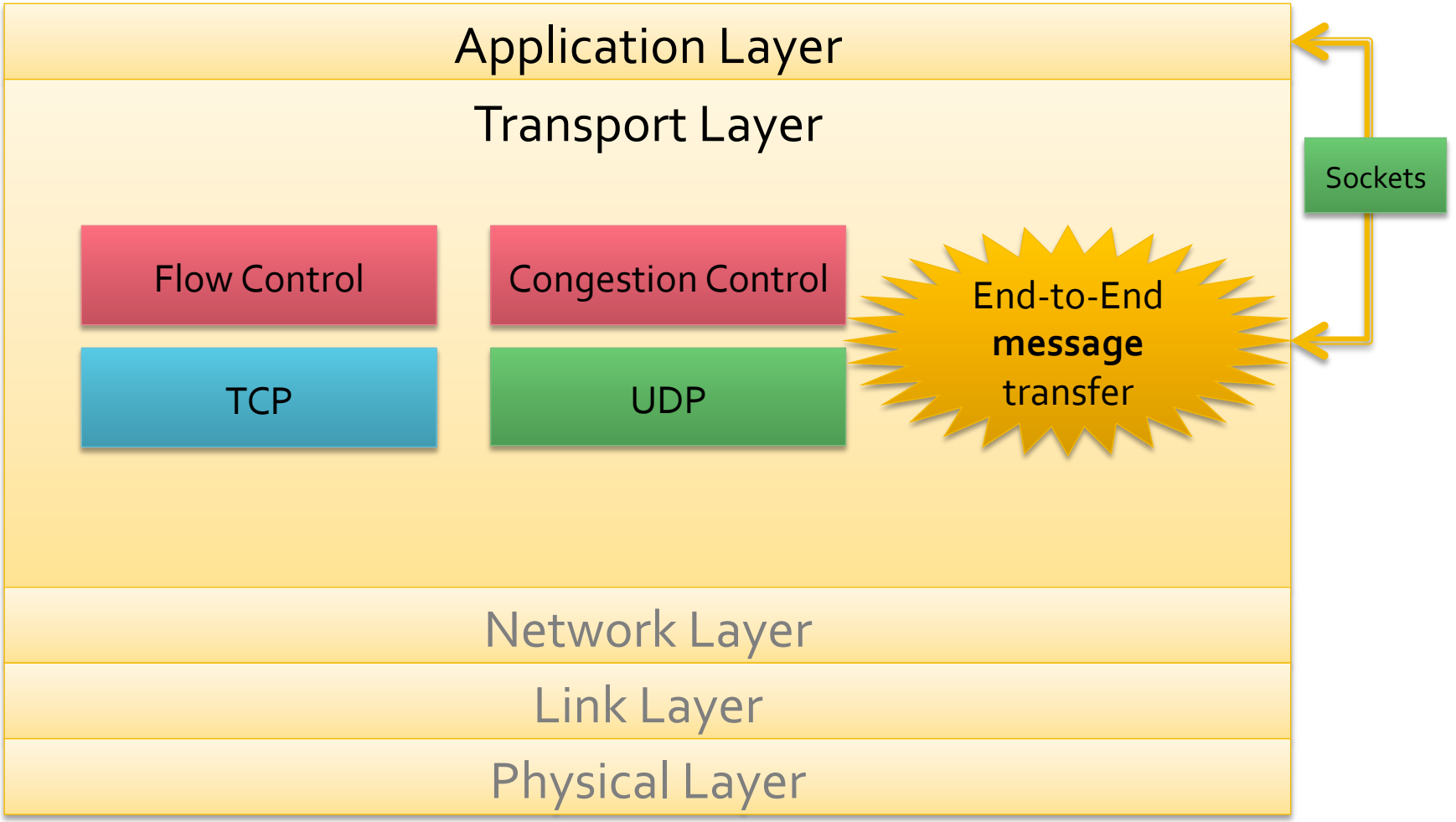
Recap – Link Layer



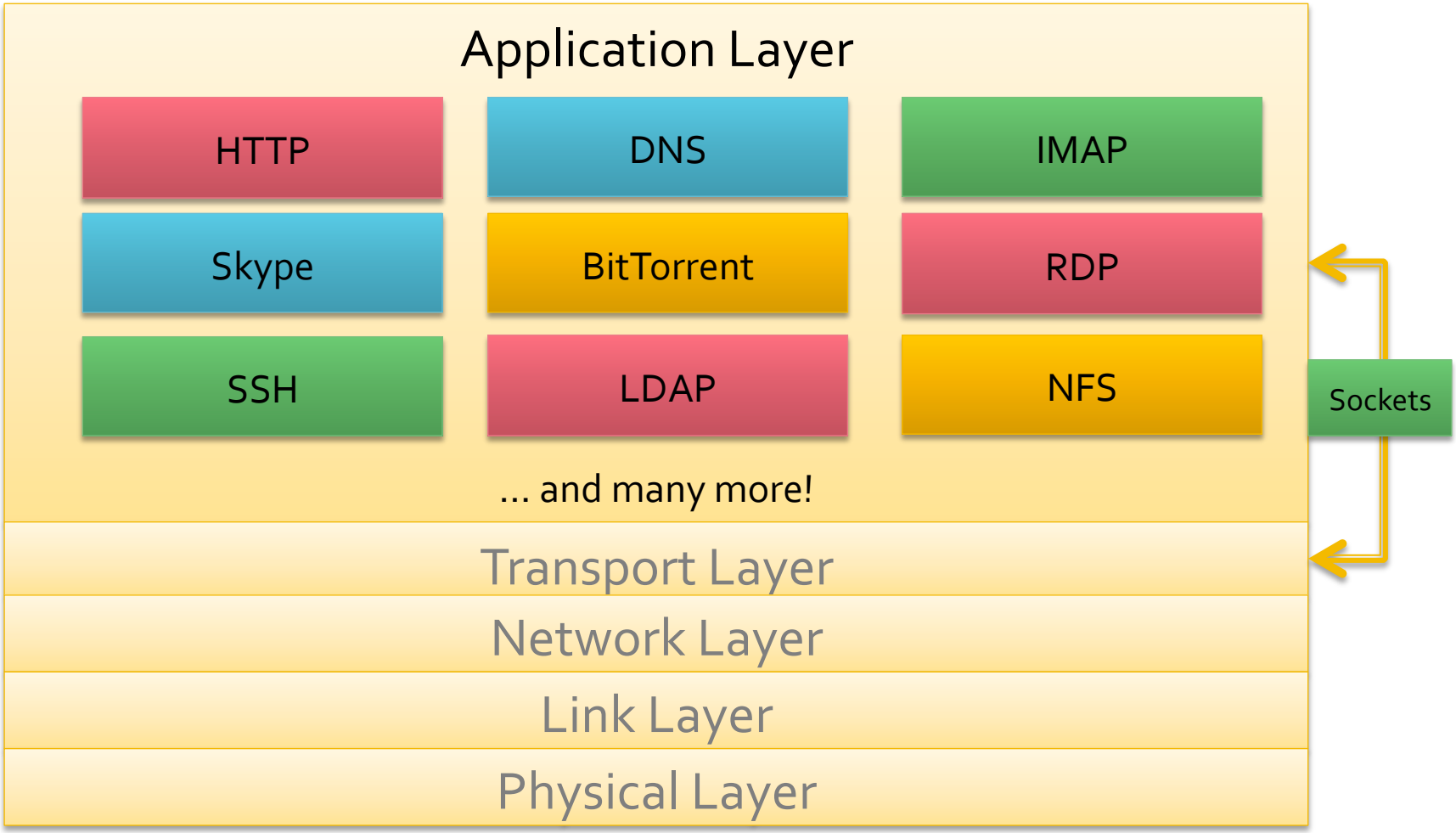
Recap – Network Layer



Recap – Transport Layer



Introducing the Application Layer



Goals

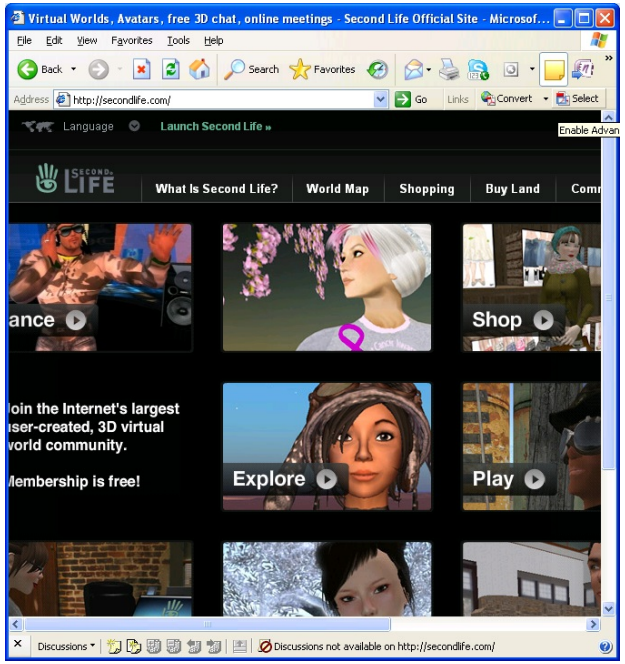
- Concepts and implementation aspects of network application protocols
  - Transport-layer service models
  - Client-server paradigm
  - Peer-to-peer paradigm
- Learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- Programming network applications
  - Socket API

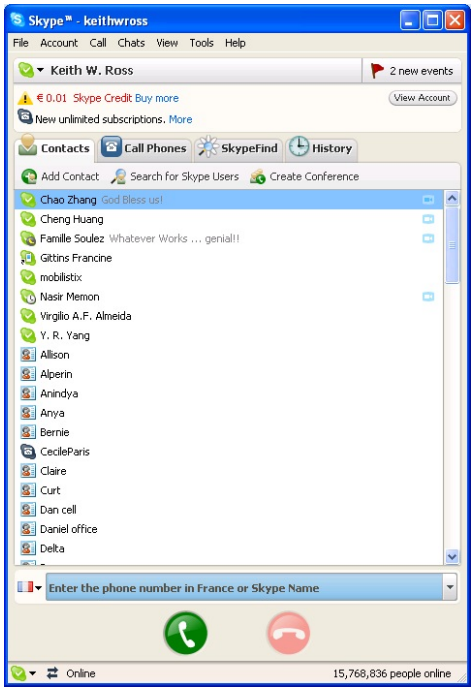
Network Applications

- What programs do you run that use the Internet?

Some Network Apps

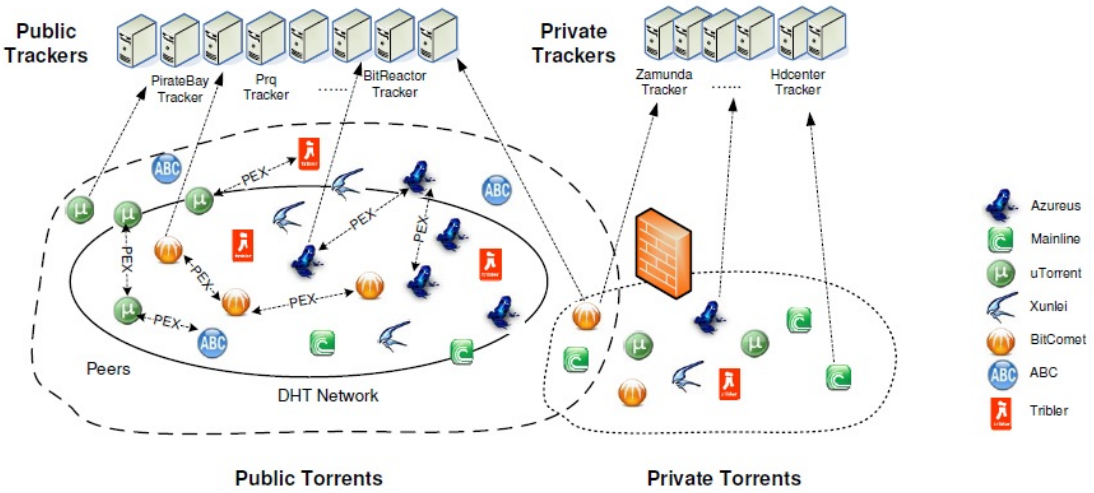
- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Social networks
- Voice over IP
- Real-time video conferencing
- Grid computing

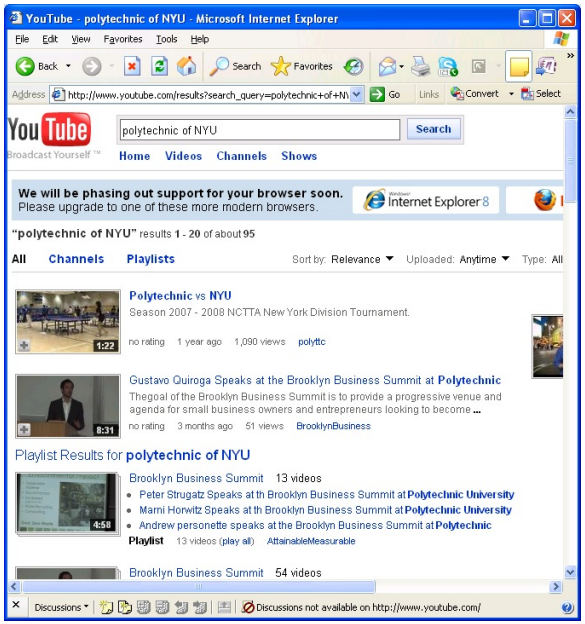

Virtual Worlds, Avatars, free 3D chat, online meetings - Second Life Official Site - Microsof...
Launch Second Life »
What Is Second Life?
World Map
Shopping
Buy Land
Shop
Explore
Play
Discussions not available on http://secondlife.com/


Skype™ - keithwross
Keith W. Ross
2 new events
Contacts
Call Phones
SkypeFind
History
Chao Zhang
Cheng Huang
Famille Soulez
Gittins Francine
mobilistix
Nasir Memon
Virgilio A.F. Almeida
Y. R. Yang
Allison
Alperin
Anindya
Anya
Bernie
CecileParis
Claire
Curt
Dan cell
Daniel office
Delta
Enter the phone number in France or Skype Name
Online
15,768,836 people online


Facebook | Keith Ross - Microsoft Internet Explorer
facebook
Keith Ross
Wall
Info
Photos
Boxes
Video
What's on your mind?
Keith Ross Katie playing Blackbird at the 2009 LFNY recital.
KatieGuitar2009
Information
Networks:
NYU Poly Faculty
Relationship Status:
Married
Discussions not available on http://www.facebook.com/

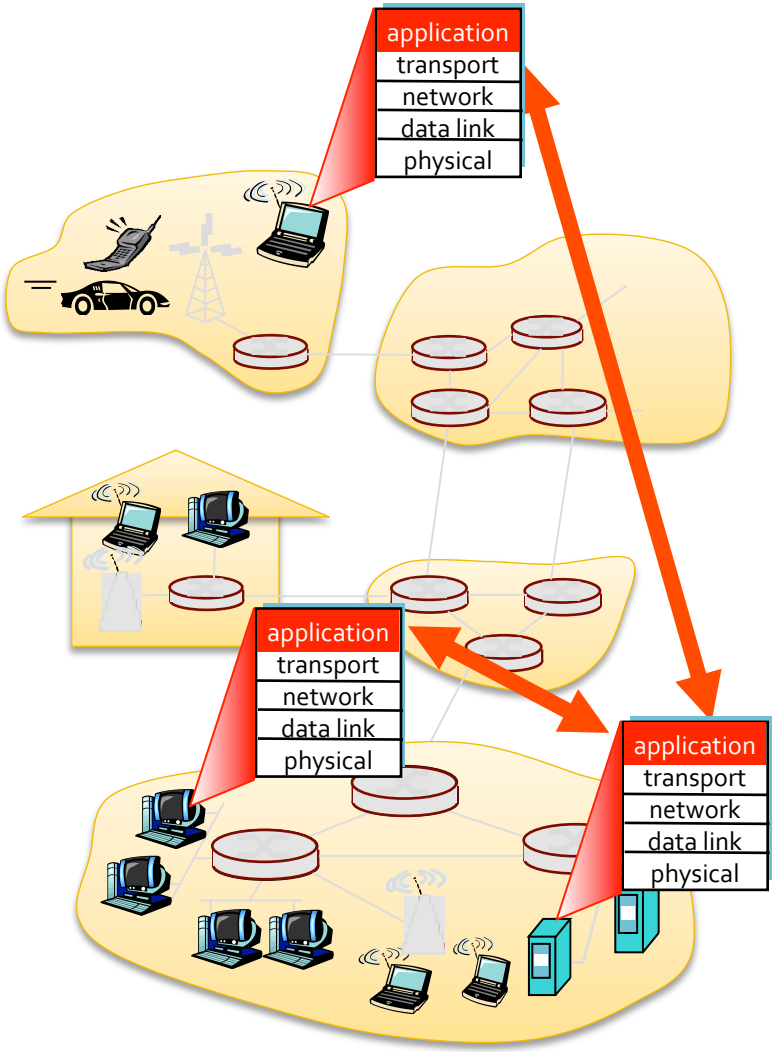

Public Discovery Sites
Private Discovery Sites
torrentspy
TorrentPortal
SHARE-IT
Demonoid.com
HDBits.org
cinematik.net
TVTORRENTS.com


Public Trackers
PirateBay Tracker
Prq Tracker
BitReactor Tracker
Private Trackers
Zamunda Tracker
Hdcenter Tracker
PEX
Peers
DHT Network
Azureus
Mainline
uTorrent
Xunlei
BitComet
ABC
Tribler
Public Torrents
Private Torrents


YouTube - polytechnic of NYU - Microsoft Internet Explorer
YouTube
polytechnic of NYU
Search
Home
Videos
Channels
Shows
We will be phasing out support for your browser soon.
Please upgrade to one of these more modern browsers.
Internet Explorer 8
"polytechnic of NYU" results 1 - 20 of about 95
All
Channels
Playlists
Polytechnic vs NYU
Season 2007 - 2008 NCTTA New York Division Tournament.
Gustavo Quiroga Speaks at the Brooklyn Business Summit at Polytechnic
Playlist Results for polytechnic of NYU
Brooklyn Business Summit 13 videos
Peter Strugatz Speaks at th Brooklyn Business Summit at Polytechnic University
Marni Horwitz Speaks at th Brooklyn Business Summit at Polytechnic University
Andrew personette speaks at the Brooklyn Business Summit at Polytechnic
Brooklyn Business Summit 54 videos
Discussions not available on http://www.youtube.com/

Creating a Network App

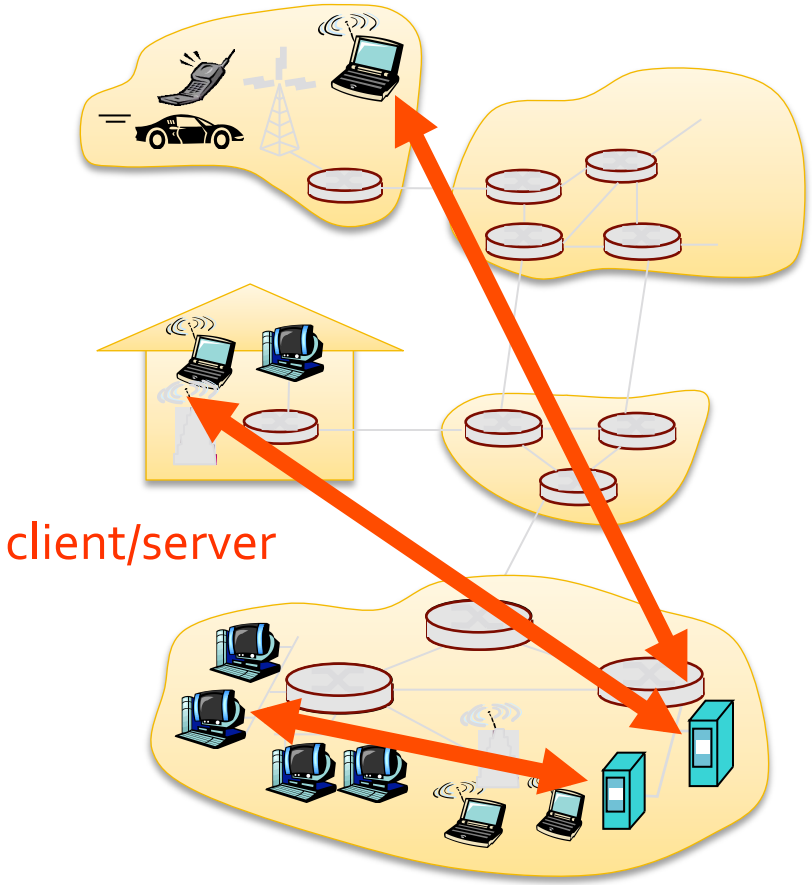
- Write programs that
  - Run on (different) end systems
  - Communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for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



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

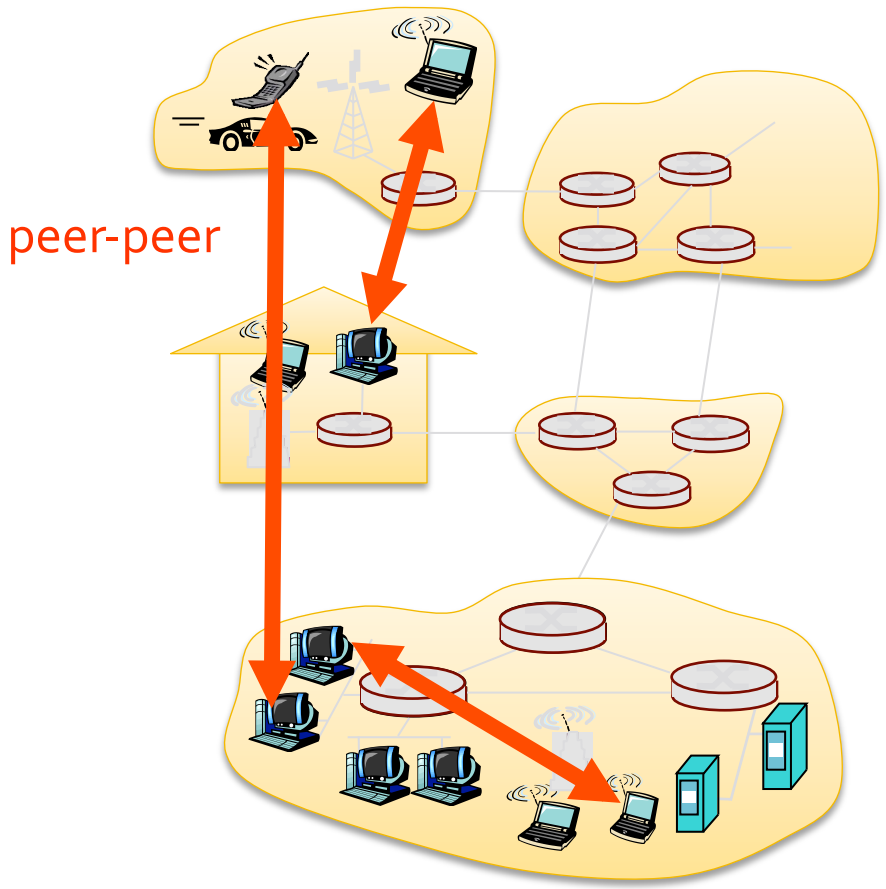
Google Data Centers

- Estimated cost of data center: $600M
- Google spent $2.4B in 2007 on new data centers
- Each data center uses 50-100 megawatts of power

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**



Hybrid of Client-Server and P2P

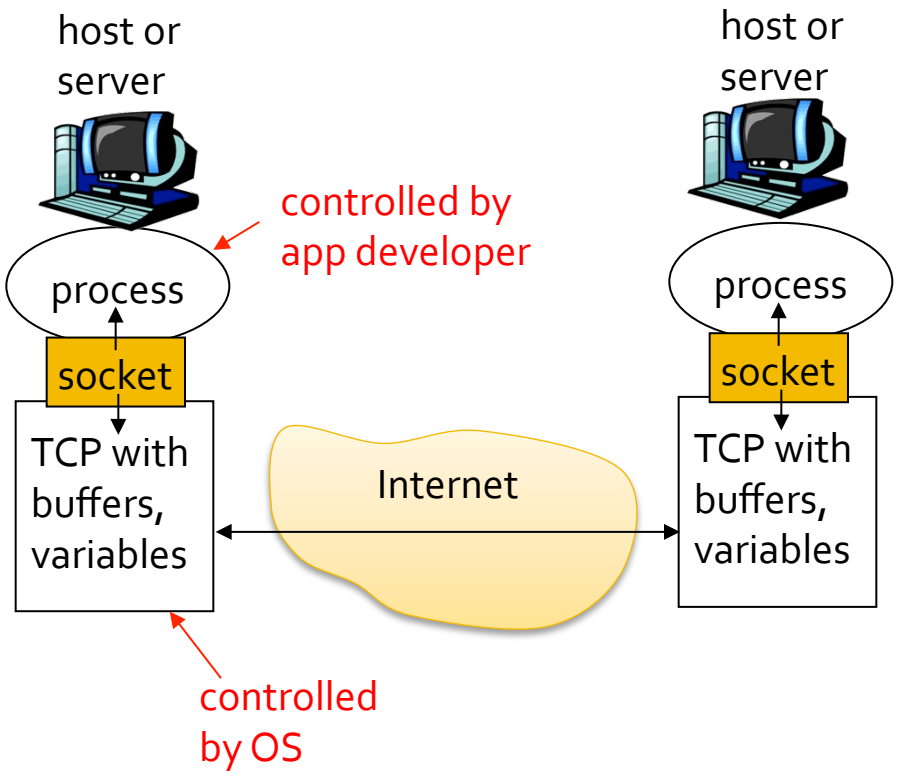
- Skype
  - Voice-over-IP P2P application
  - Centralized server: finding address of remote party
  - Client-client connection: direct (not through server)
- Instant messaging
  - Chatting between two users is P2P
  - Centralized service: client presence detection/location
    - User registers its IP 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
  - Sending process relies on transport infrastructure on other side of door which brings 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?

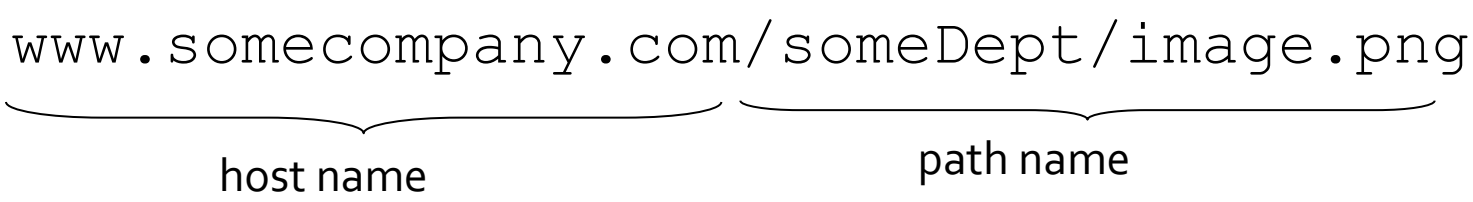
Transport Service Requirements for Common Apps

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)

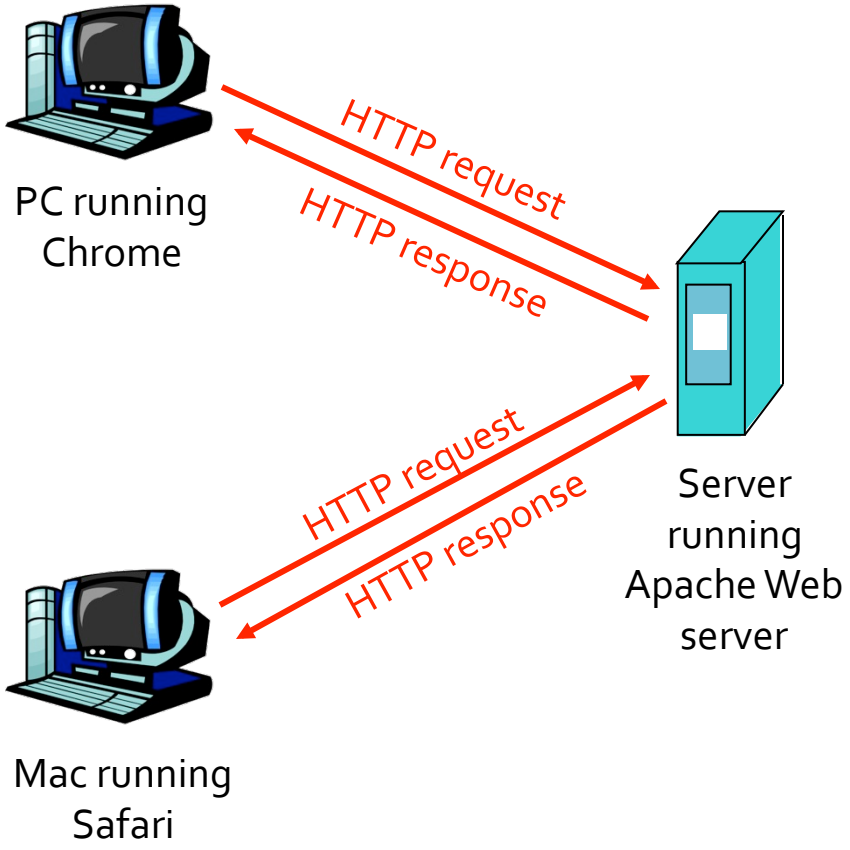
Web and HTTP

- 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:

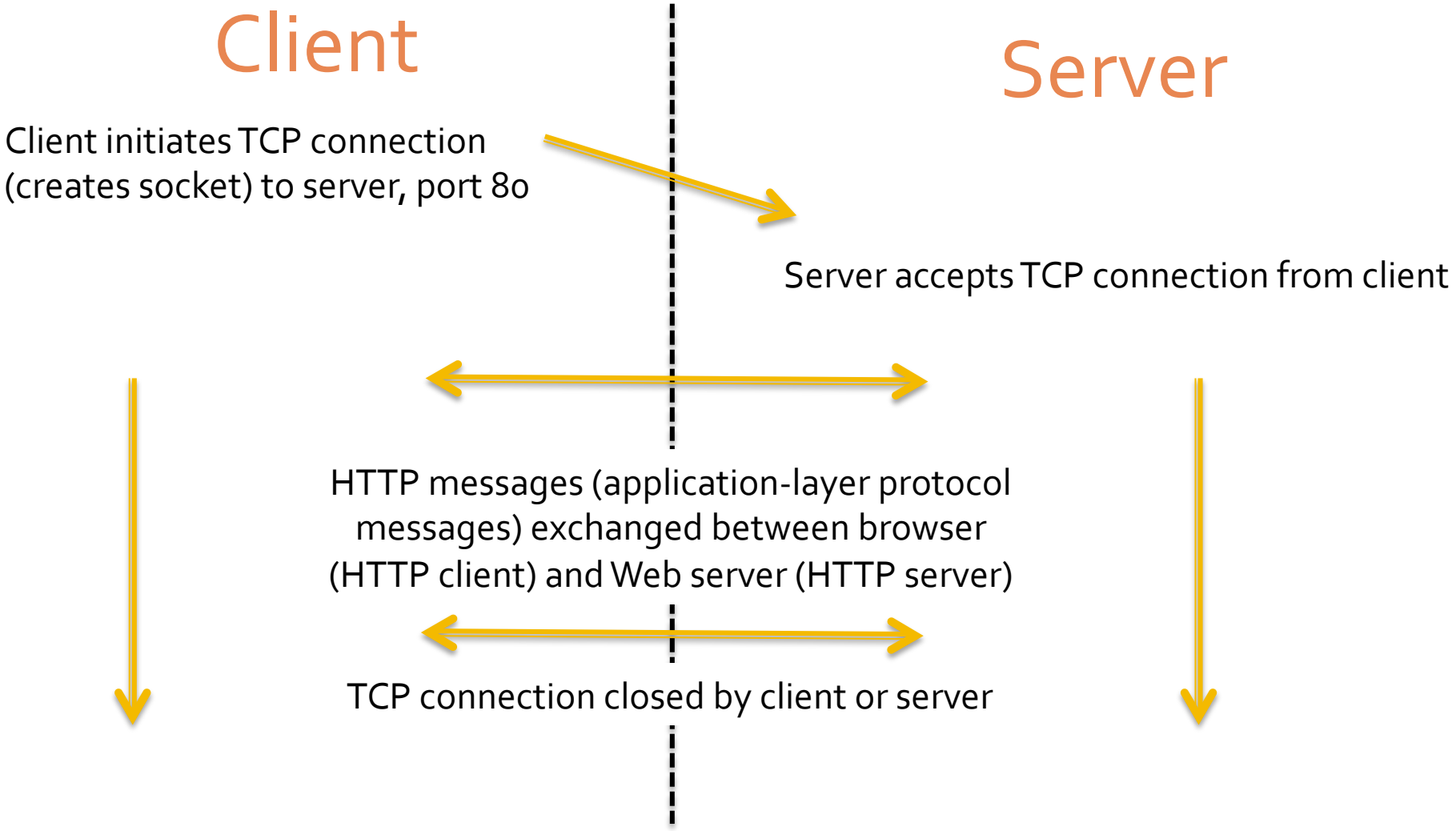


Hypertext Transfer Protocol Overview

- **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



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, references to 10 jpeg images)

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

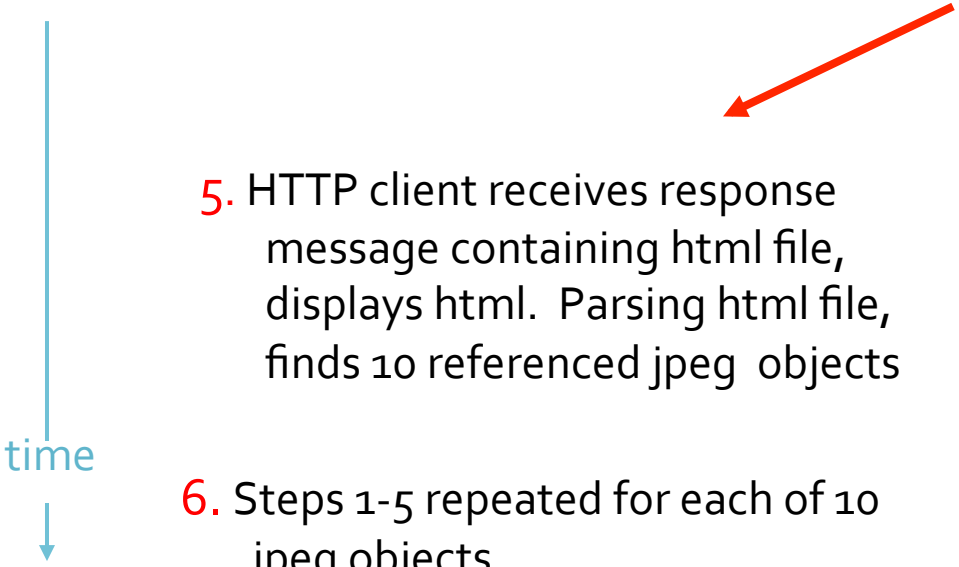
1b. HTTP server at host www.someCompany.com waiting for TCP connection at port 80. "accepts" connection, notifying client

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.

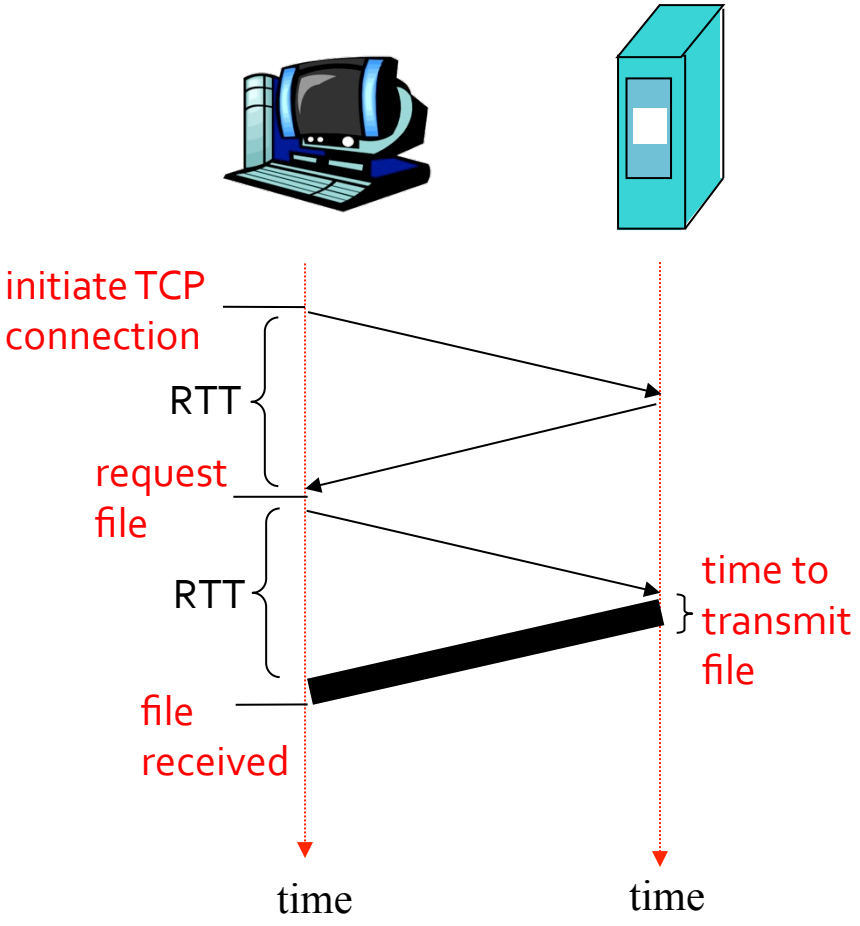
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

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!)**

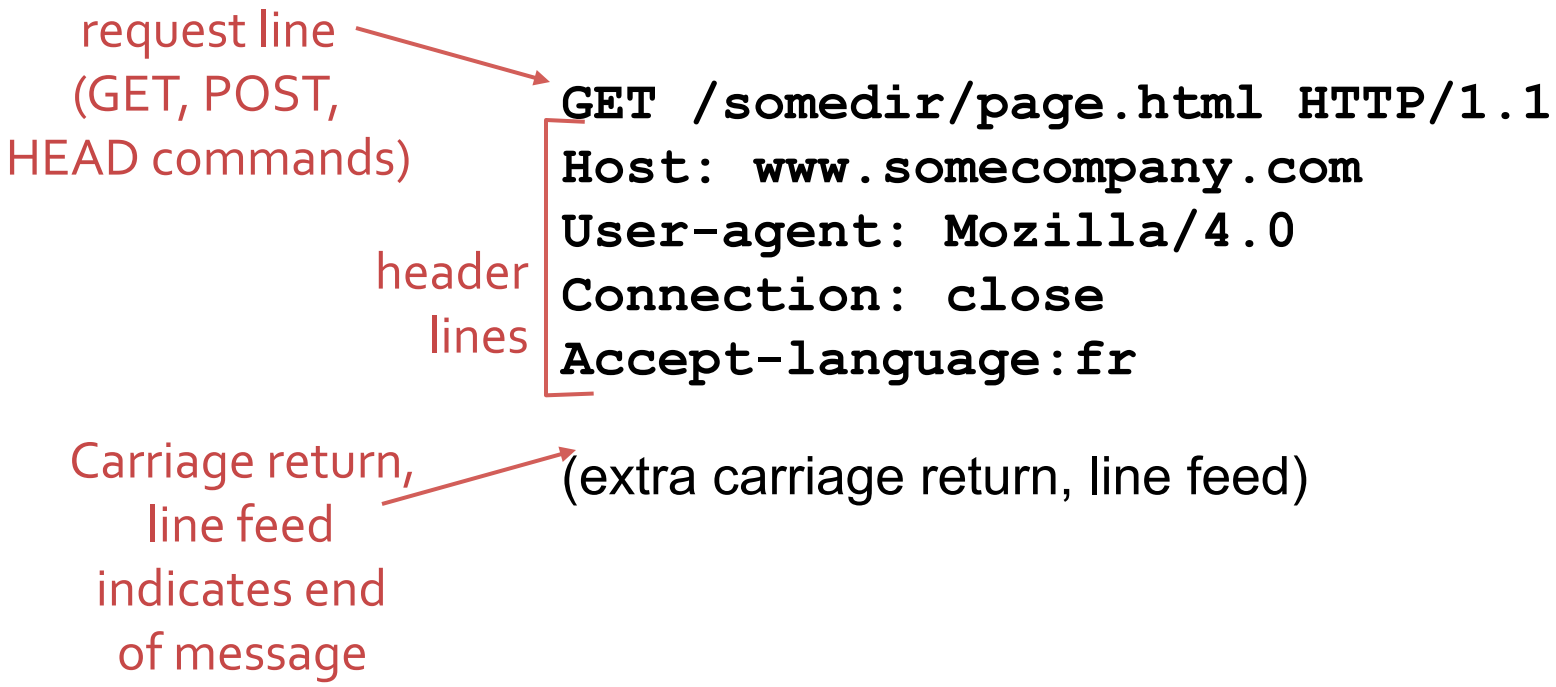


Persistent vs Non-Persistent HTTP

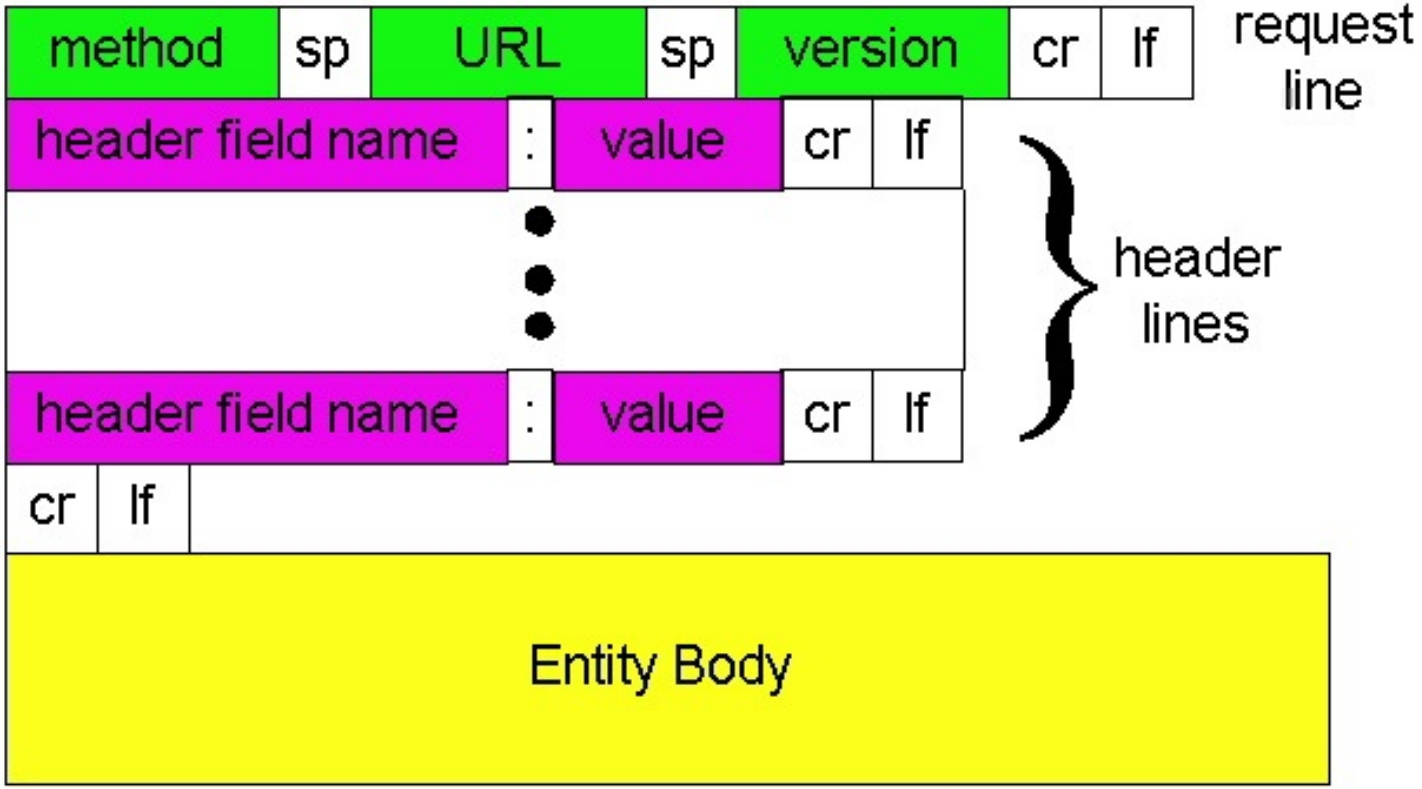
- **Non-Persistent HTTP issues**
  - Requires 2 RTTs per object
  - OS overhead for each TCP connection
  - Browsers often open parallel TCP connections to fetch referenced objects (more overhead)
- **Persistent HTTP**
  - Server leaves connection open after sending response
  - Subsequent HTTP messages 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)



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

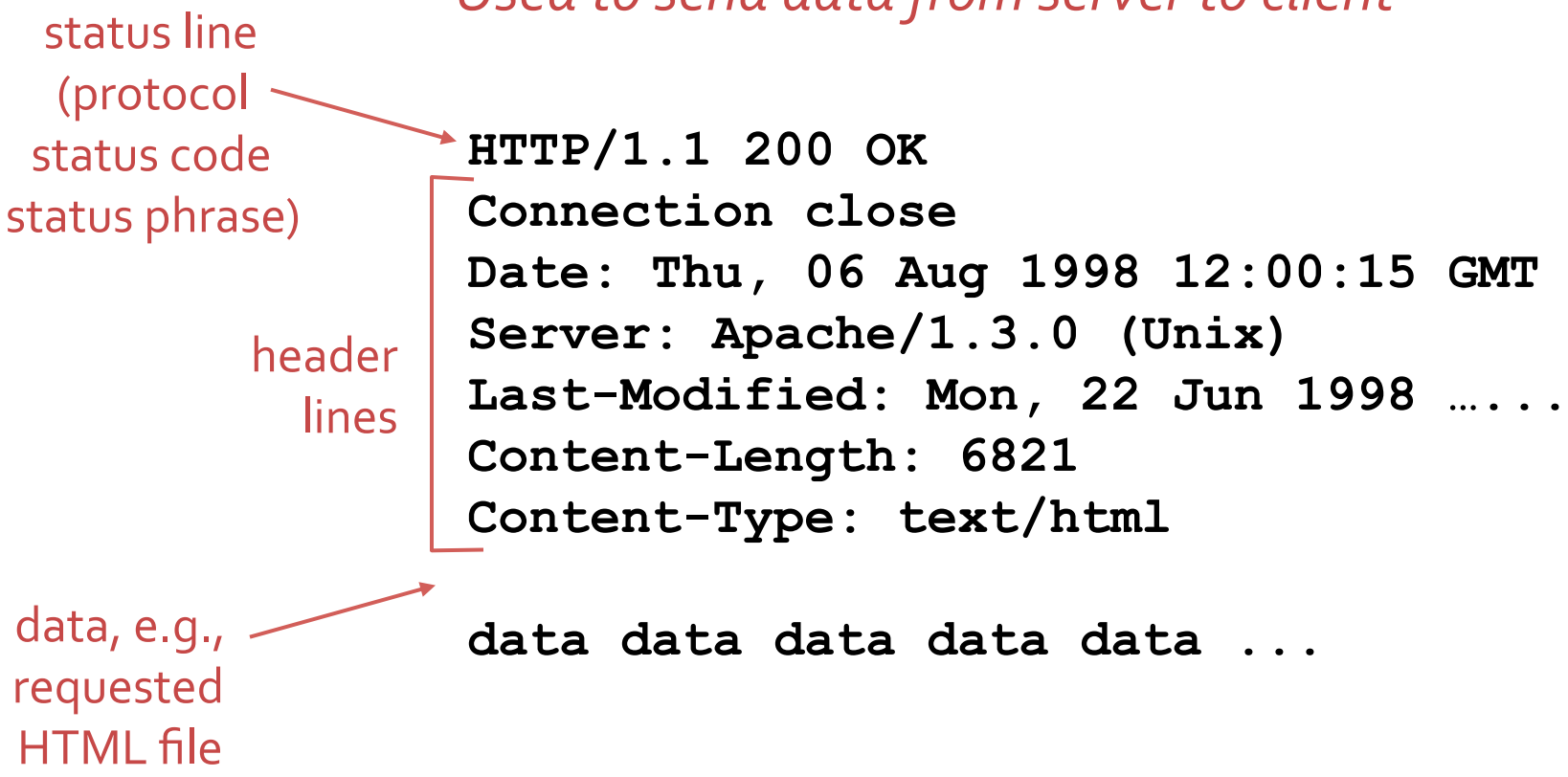
```
www.somecompany.com/page.php?variable1=testData
```

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



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 /support/ 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

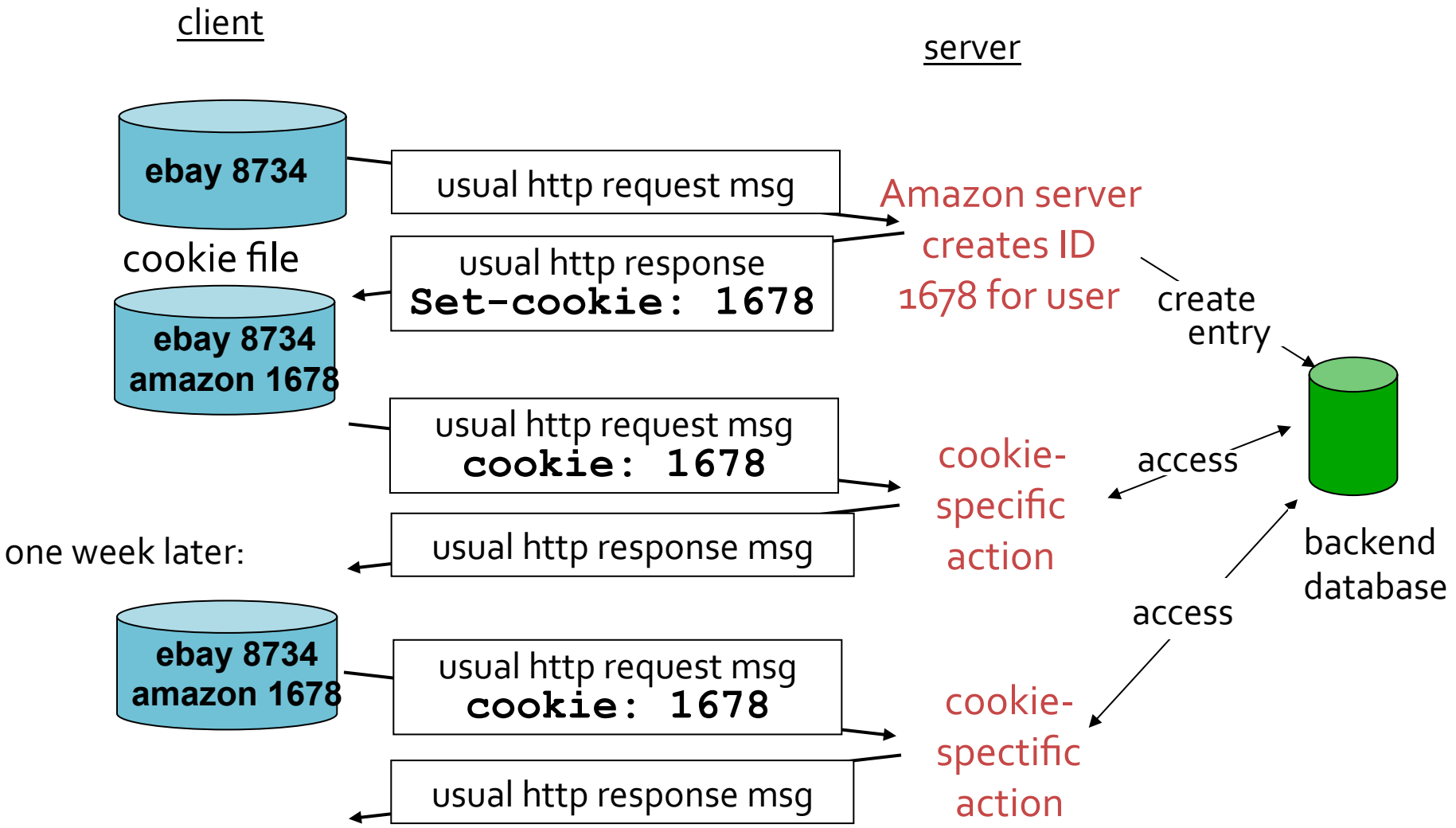
3. Look at response message sent by HTTP server!

User-Server State: Cookies

- HTTP is stateless, but that is not always desirable
- Solution? Cookies!
- Most major Web sites use 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

- What cookies store
  - Key -> Value pairs
- What can I do with this?
  - Authorization
  - Shopping carts
  - User session state (Web e-mail)
  - etc…
- 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

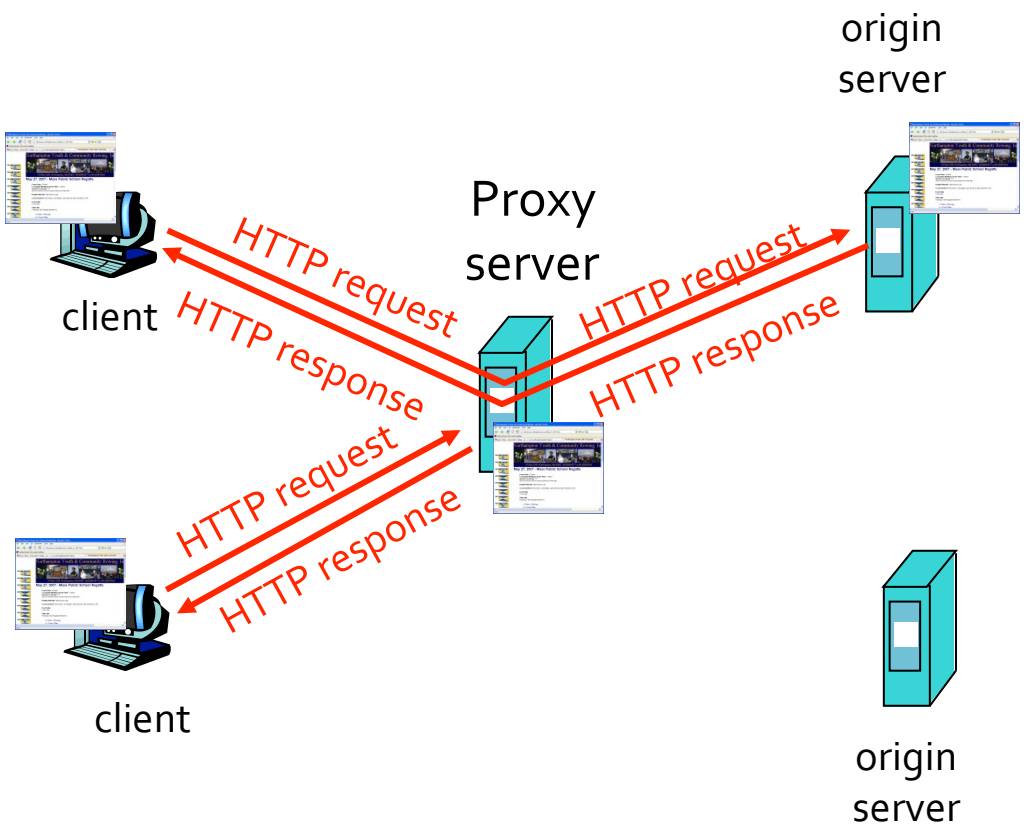
EverCookie

- **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 (coming soon)
- See http://samy.pl/evercookie/

Web Caches (Proxy Server)

Goal: satisfy client request without involving origin server

- User sets browser to send all Web accesses to cache
- Browser sends all HTTP requests to cache
  - Object in cache? Cache returns object
  - Object not in cache? Cache requests object from origin server, then returns object to client



Web Caching

- Cache acts as both client and server
- Cache is typically installed by ISP (university, company, residential ISP)

- Why Web caching?
  - Reduce response time for client request
  - Reduce traffic on an institution's access link

Caching Benefits

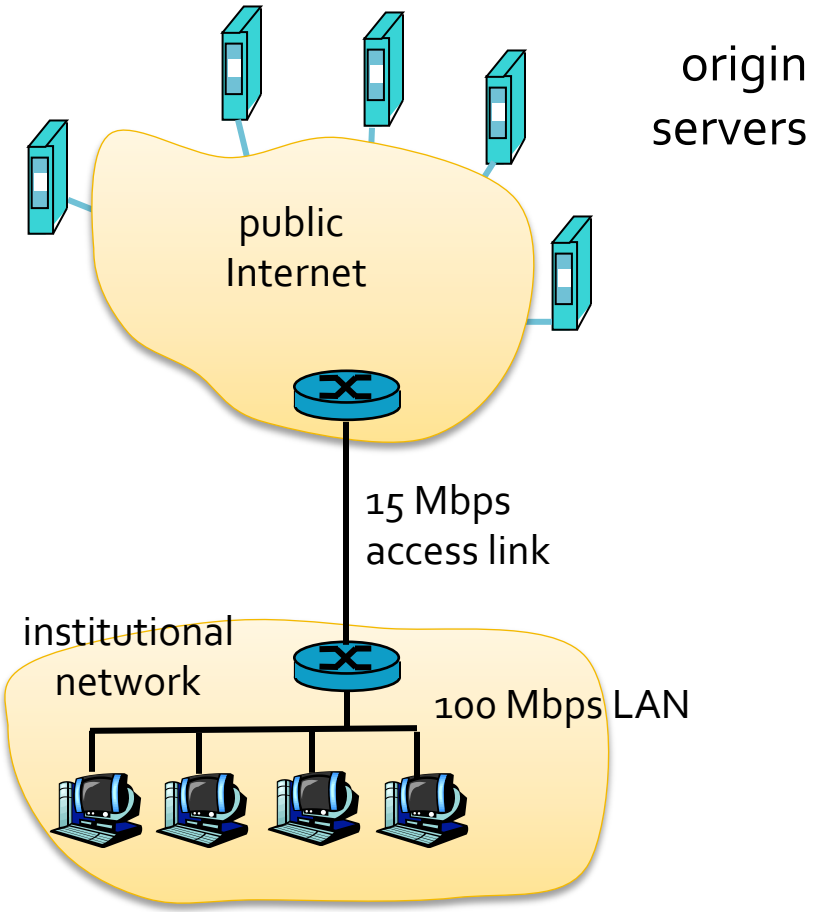
Assumptions

- Average object size = 1 Mbit
- Average request rate from institution's browsers to origin servers = 15/sec
- Delay from institutional router to any origin server and back to router = 2 sec

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching Benefits

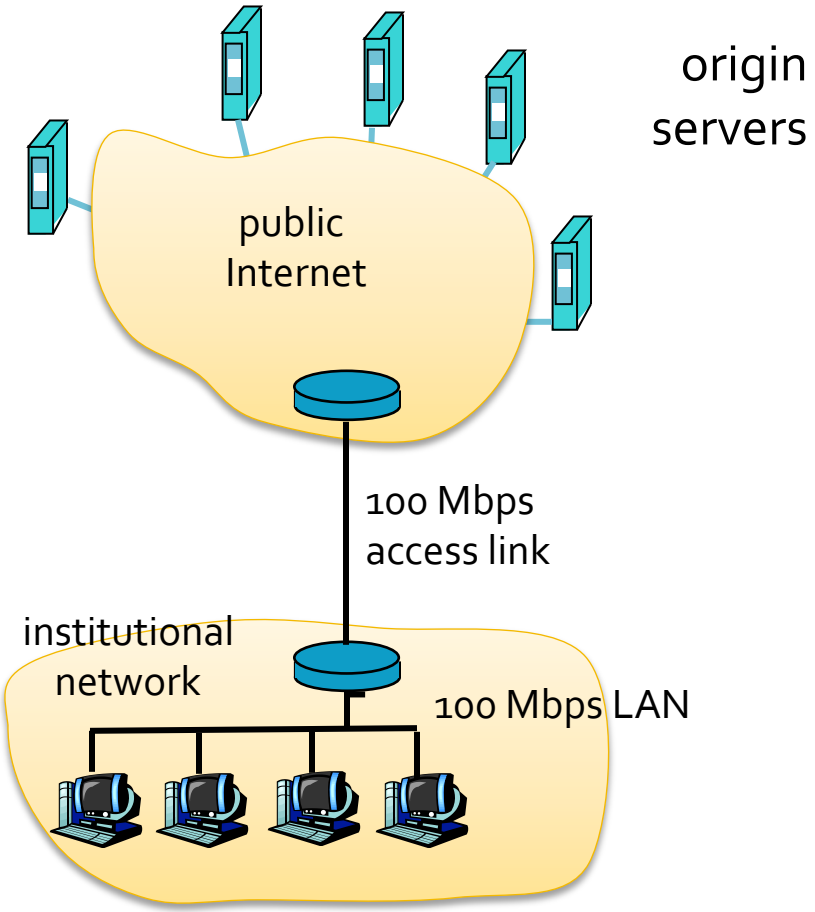
Possible solution

- Buy more link bandwidth!
- Say, 100 Mbps

Consequence

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- Often a costly upgrade



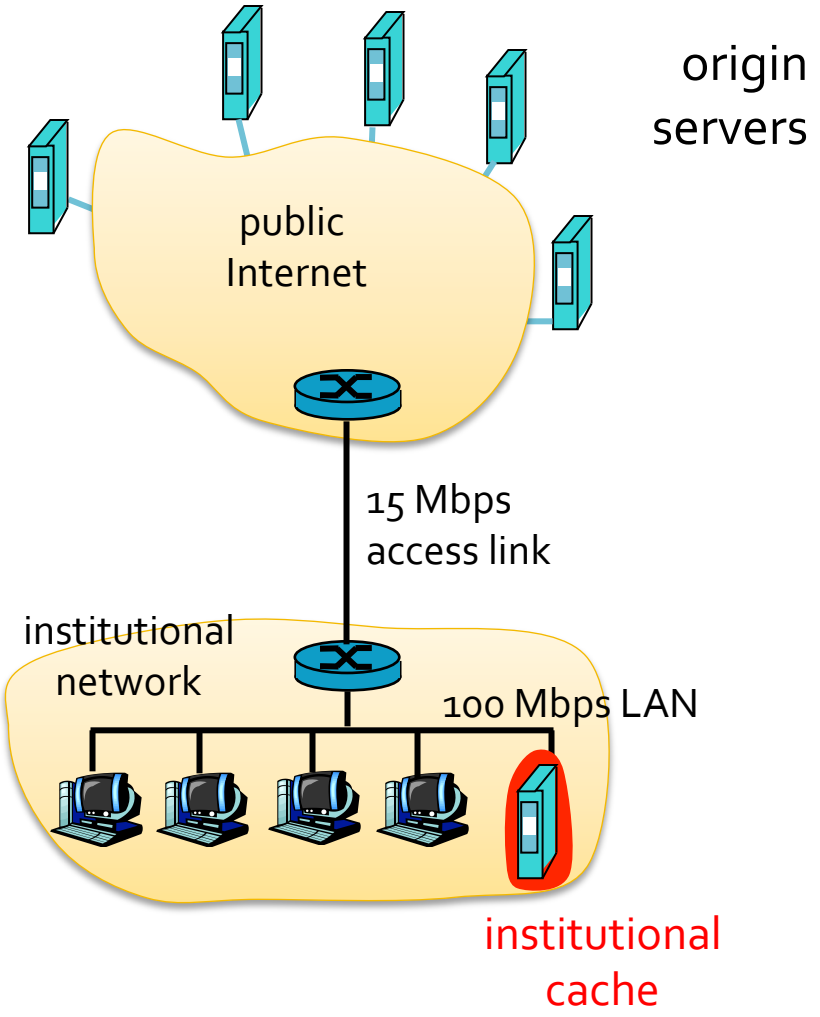
Caching Benefits

Another solution: install cache

- Suppose hit rate is 0.4

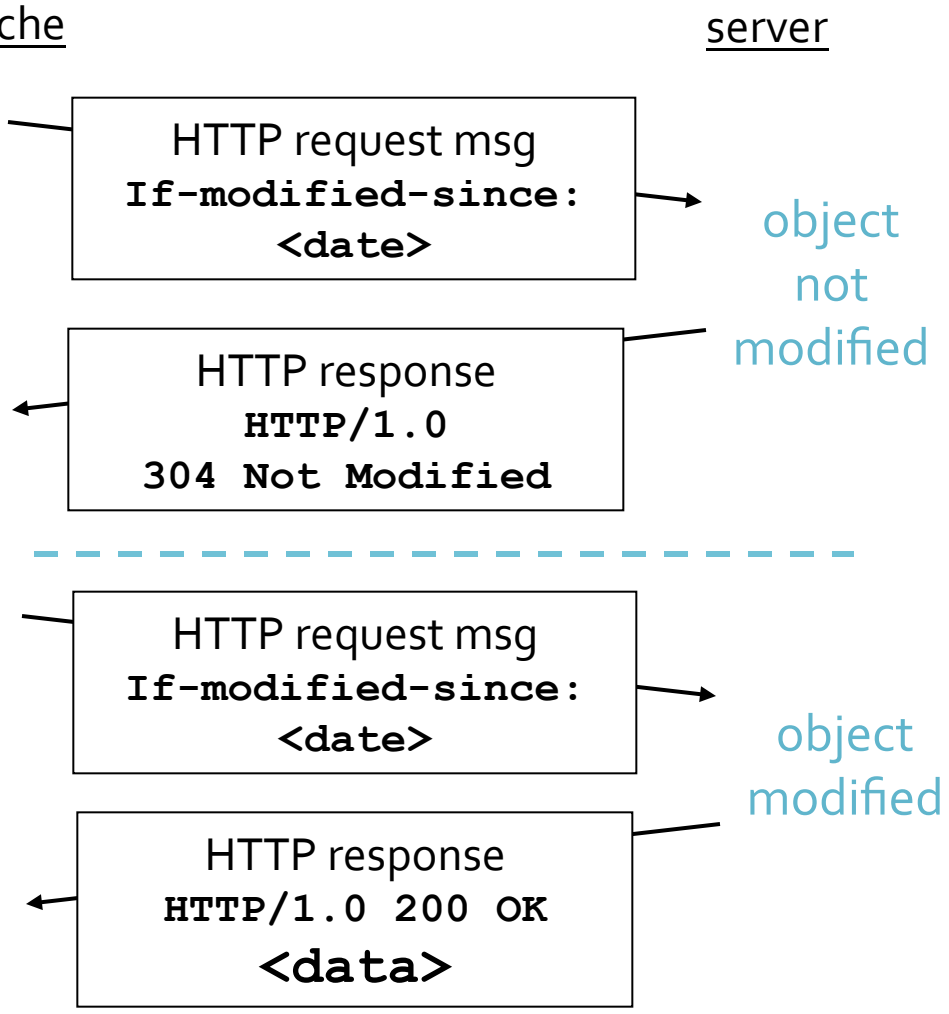
Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- Total avg delay
  - = Internet delay + access delay + LAN delay
  - = .6*(2.01) secs + .4*milliseconds
  - = Less than 1.4 secs
- **Faster and cheaper than buying more bandwidth!**



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`



Other Caching Uses

- How do large websites uses caches?
  - How do I use a cache on my website?