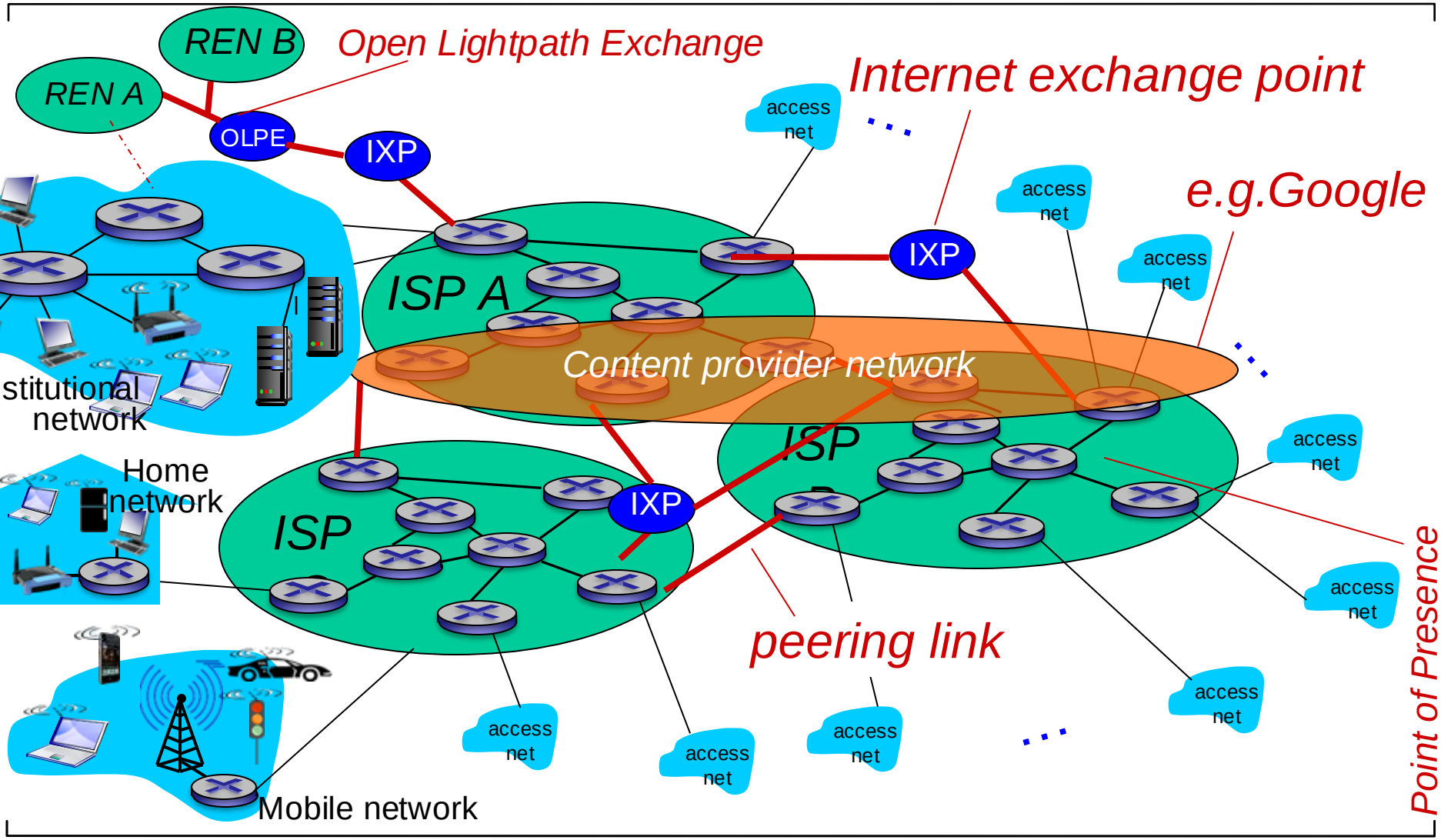
Principles of network applications the Web

CE 352, Computer Networks

Salem Al-Agtash

Lecture 4

Recap (Internet, CPN, REN)

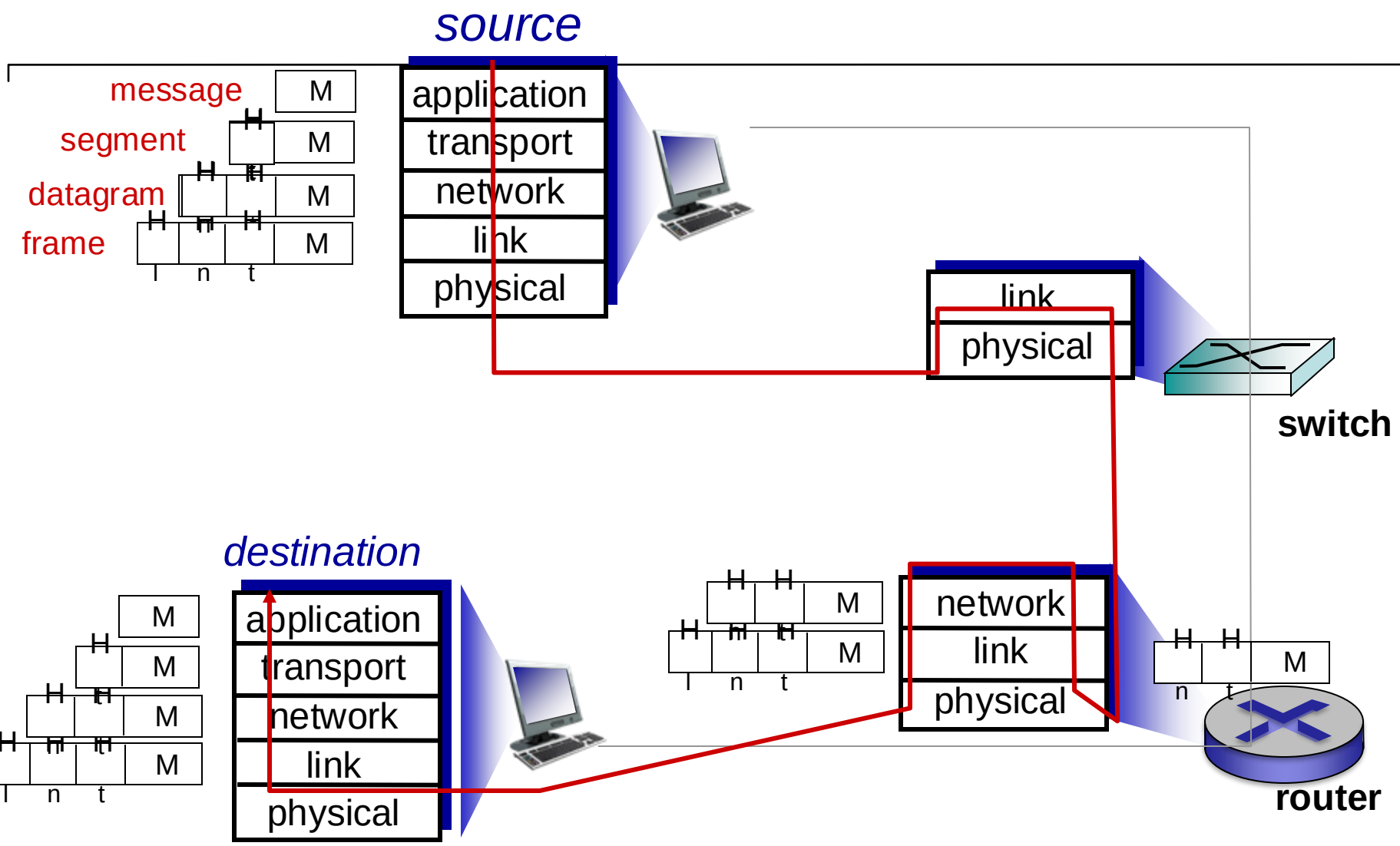


Recap (Physical links)

- bit: propagates between transmitter/receiver pairs
- physical link: what lies between transmitter & receiver (UTP, Fiber, Radio)

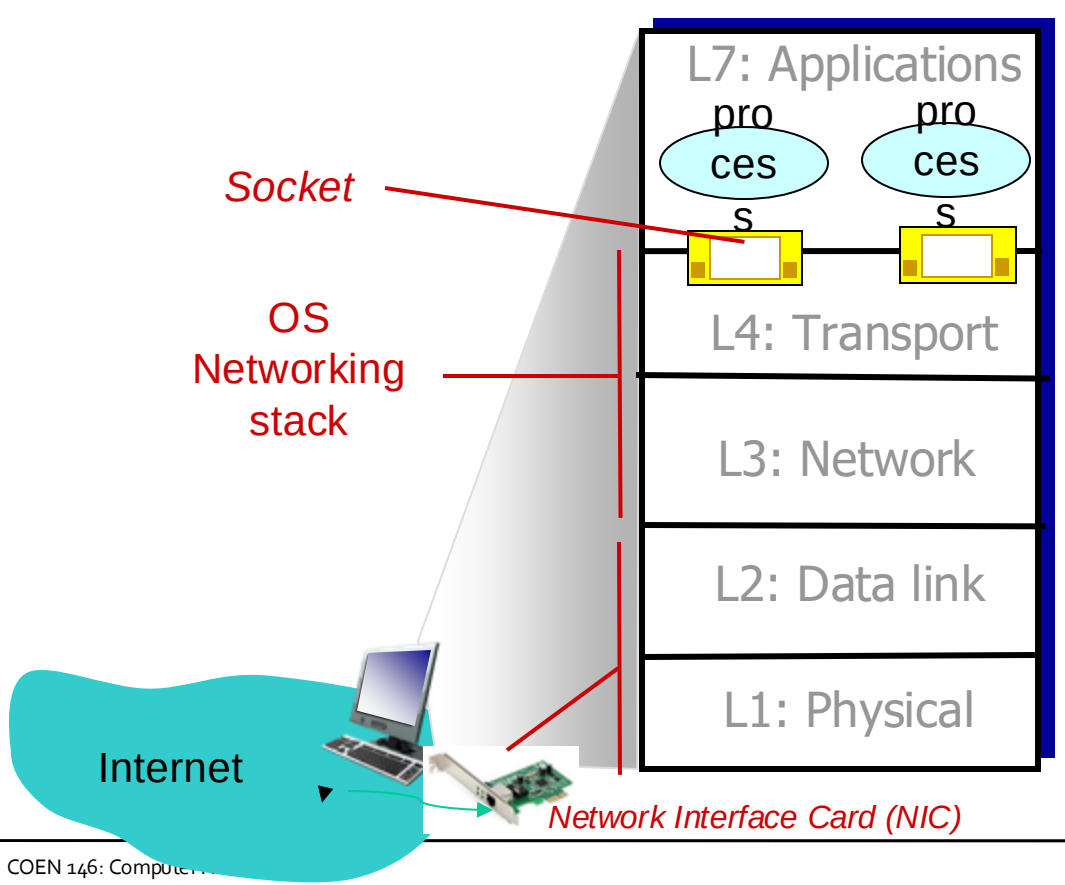
- Packet switching: no reservation of link, allows more users to use the network
- Circuit switching: Links reserved in advance, less users
- Network Performance:
  - Bandwidth: capacity in bits per second (width of link)
  - Latency (delay): length of link
  - Packet delay: nodal processing, queueing delay, transmission delay (L/R), propagation delay (d/s)
  - Packet loss: queueing buffer size and processing capabilities
  - Throughput: bits per second transfer rate between sender/ receiver

Recap (message, segment, packet, frame)



Recap (Layers)

- Protocols, reference models (TCP/IP and OSI), layers
- Internet protocol stack



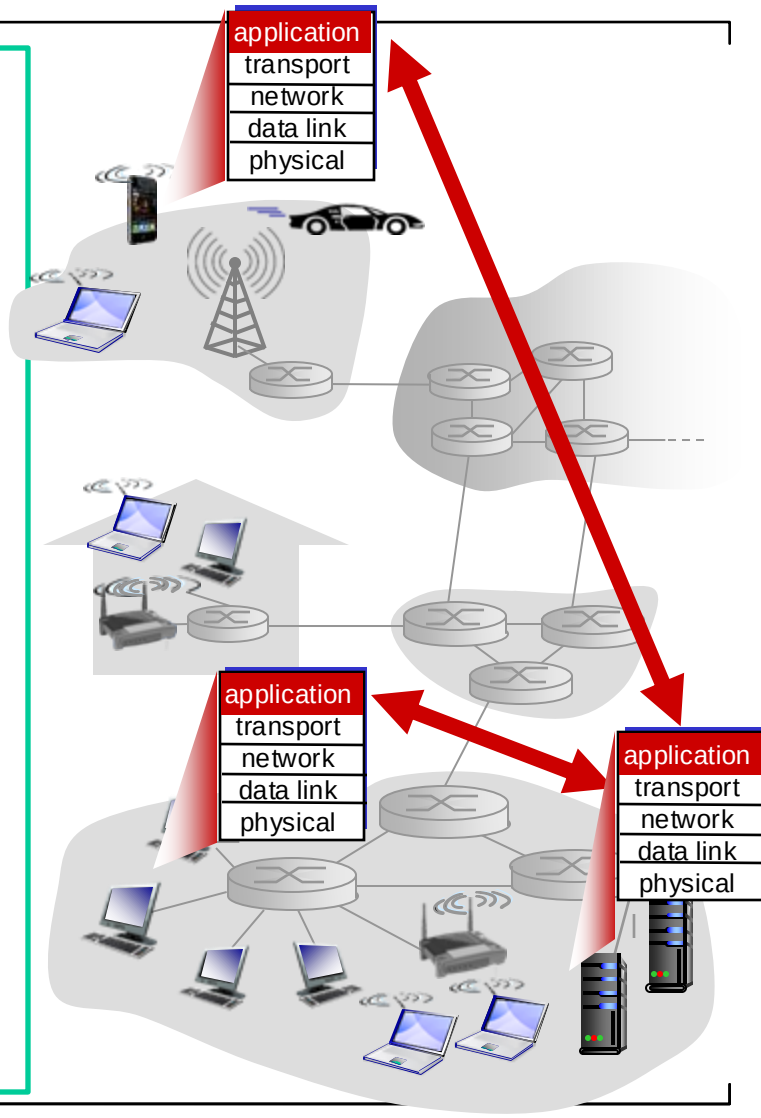
Web Browser (HTTP) Email (SMTP, IMAP, POP3) File Transfer (FTP, TFTP) Remote Login (Telnet)	Port: 16
Transmission Control Protocol (TCP) User Datagram Protocol (UDP)	
Internet Protocol IP ARP, ICMP	IPv4: 32 IPv6: 128
Ethernet, Wifi, Token Ring FDDI WAN Protocols	MAC : 48
Copper Twisted Pair Fiber Optic Radio	

Topics of Today

- Application layer
- Client – Server, Peer-to-Peer
- Communication:
  - On same machine – IPC
  - On network – Socket API
- Application protocols (http, FTP, ..)
- Transport protocol (TCP, UDP)
- The Web – WWW

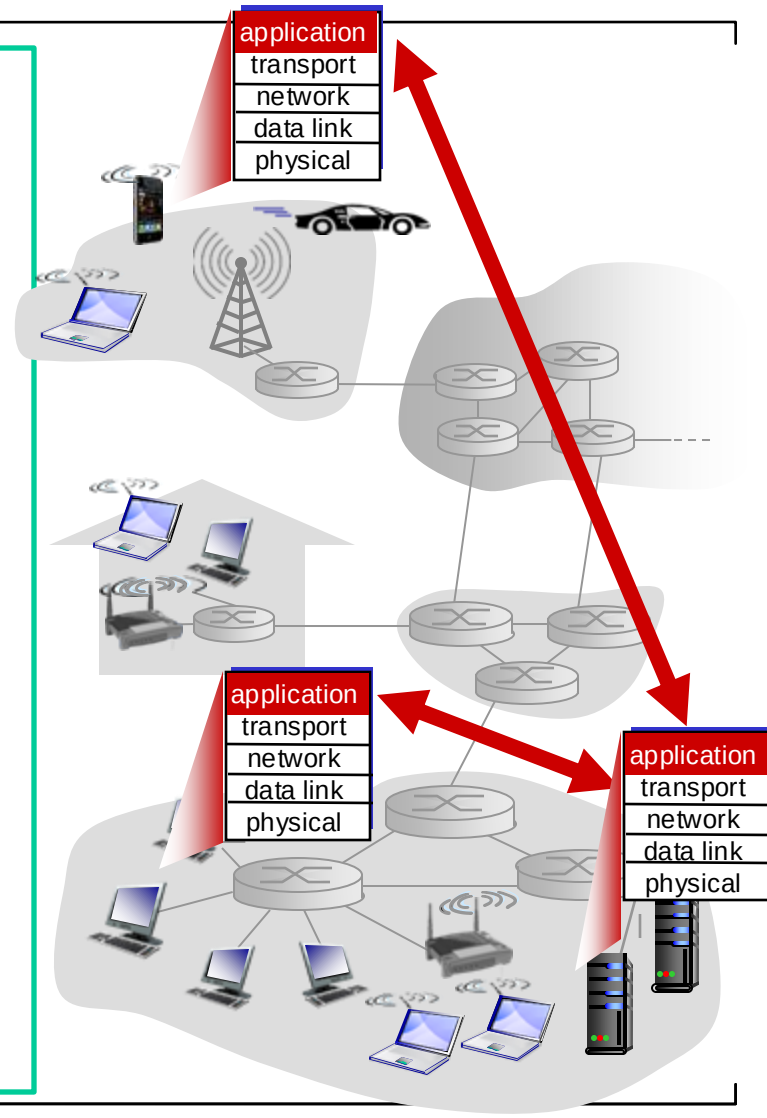
Applications on the Network

- End-to-end system programs/ applications
  - communicate over network
- no need to write software for network-core devices
- Examples of end-to-end applications:
  - ?



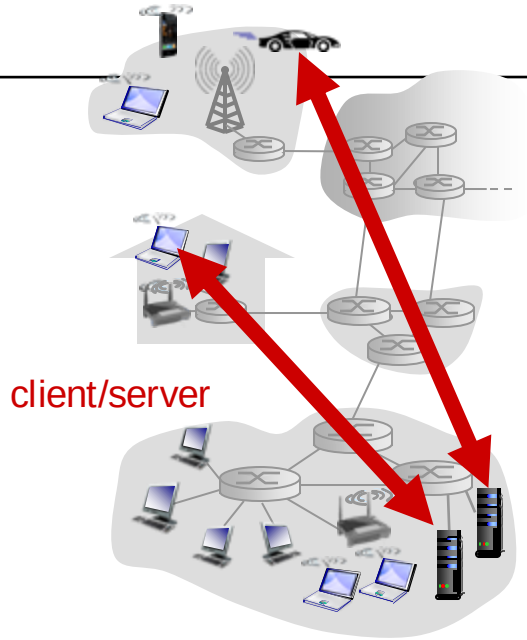
Applications on the Network

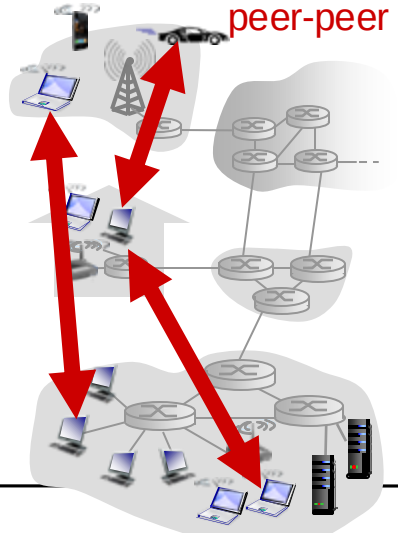
- End-to-end system programs/ applications
  - communicate over network
- no need to write software for network-core devices
- Examples of end-to-end applications:
  - Web, e-mail, text messaging, remote login, file transfer
  - social networking, multi-user network games
  - VoIP, streaming stored video (YouTube, Hulu, Netflix)



Application architecture

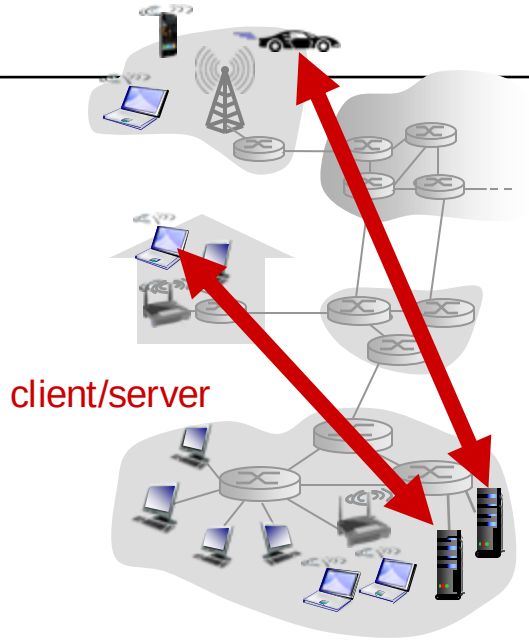
- Client – Server
  - Server:
  - Clients:
- Peer-to-peer (P2P)
  - Peers

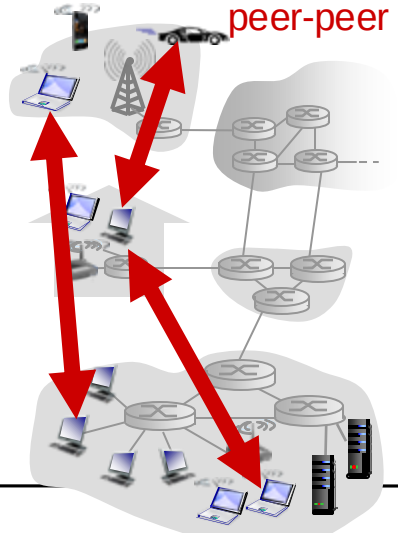




Application architecture

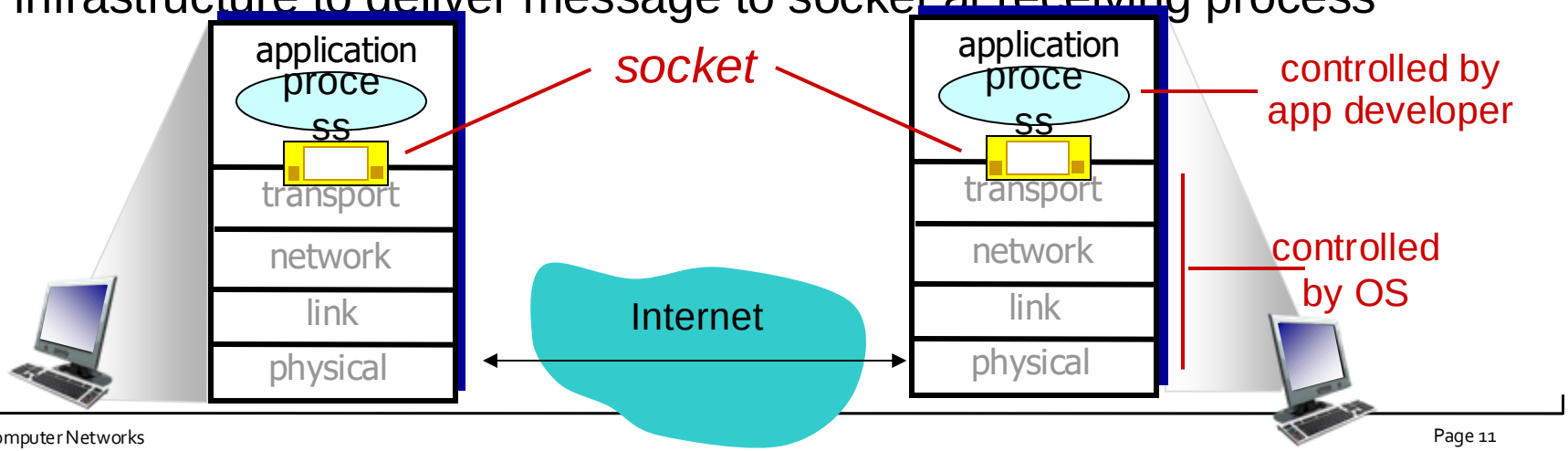
- Client – Server
  - Server: always-on host permanent IP address data centers for scaling
  - Clients: communicate with server may be intermittently connected may have dynamic IP addresses
- Peer-to-peer (P2P)
  - *no* always-on server arbitrary end systems directly communicate peers request service from other peers, provide service in return
  - peers are intermittently connected and change IP addresses





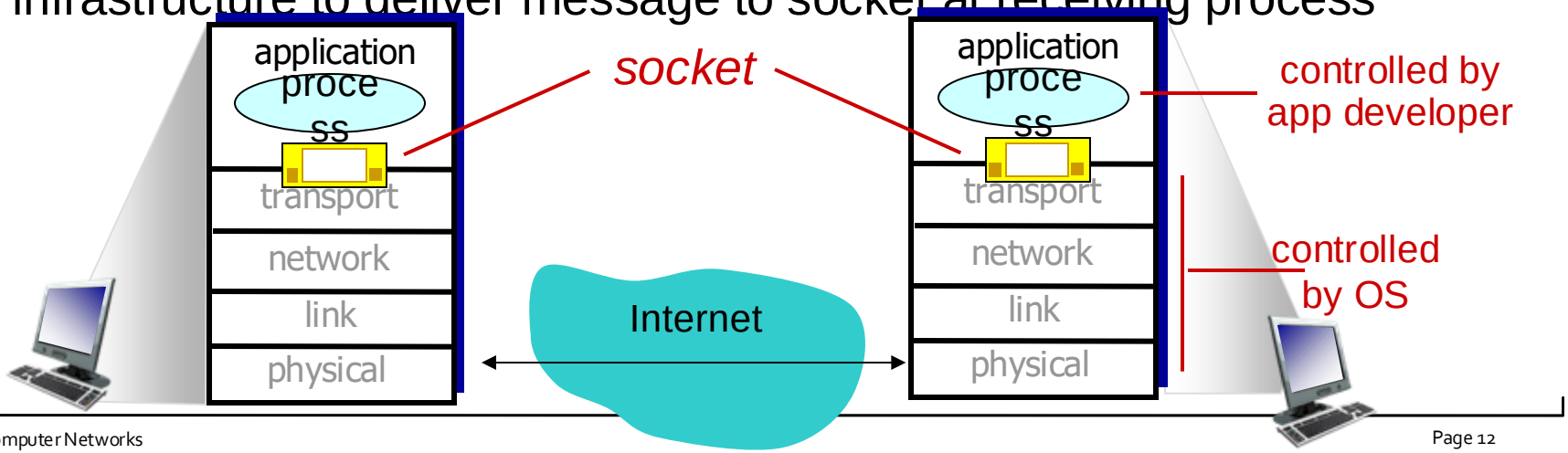
Inter-process communication

- Process: Program in Execution
  - Same hosts: processes communicate using
  - Different hosts: processes communicate by
- Socket: Process sends/ receives messages via socket (.....)
  - Sending process pushes message outdoor and relies on transport infrastructure to deliver message to socket at receiving process



Inter-process communication

- Process: Program in Execution
  - Same hosts: processes communicate using IPC defined by OS. e.g. Pipes, Shared Memory, Message Queues
  - Different hosts: processes communicate by exchanging messages. e.g. Client-Server, P2P
- Socket: Process sends/ receives messages via socket (IP + Port)
  - Sending process pushes message outdoor and relies on transport infrastructure to deliver message to socket at receiving process



Transport protocols

- Transmission Control Protocol (TCP)
  - *-oriented:* setup required between sending and receiving processes
  - *transport* between sending and receiving processes
  - *control:* sender won't overwhelm receiver
  - *control:* adjust when network overloaded
  - *does not provide:* timing, minimum throughput guarantee, or security
- User Datagram Protocol (UDP)
  - *-oriented:* no setup connection is required
  - data transfer between sending and receiving process
  - does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

Transport protocols

- Transmission Control Protocol (TCP)
  - *connection-oriented:* setup required between sending and receiving processes
  - *reliable transport* between sending and receiving processes
  - *flow control:* sender won't overwhelm receiver
  - *congestion control:* adjust when network overloaded
  - *does not provide:* timing, minimum throughput guarantee, or security
- User Datagram Protocol (UDP)
  - *connectionless-oriented:* no setup connection is required
  - unreliable data transfer between sending and receiving process
  - does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

TCP/ UDP applications

- Examples of applications with underlying
  - *transport* protocol
  - *application* protocol
- Criteria: Timing, throughput, security, data integrity

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

Securing TCP

- TCP and UDP
  - no encryption
  - cleartext passwords sent into socket traverse Internet in cleartext
- SSL (secure socket layer)
  - provides encrypted TCP connection (TLS (transport layer security) has succeeded SSL)
    - Note: SSL is by far most common on the Internet, so SSL will continue to be default acronym of choice when making non-application specific references.
  - data integrity
  - end-point authentication
- SSL is at app layer
  - some apps (e.g., multimedia) require minimum amount of throughput to be "effective". Apps use SSL libraries, that "talk" to TCP
- SSL socket API
  - encryption, data integrity, … cleartext passwords sent into socket traverse Internet encrypted, (To cover in Week 10)

The Web

World Wide Web (WWW): A platform for deploying applications and sharing information, portably and securely

WWW

- Distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990 at CERN (European Organization for Nuclear Research)
  - HTTP/0.9 – 1991 with simple GET command for the Web
  - HTTP/1.0 – 1992 with Client/Server information, simple caching
  - HTTP/1.1 – 1996, 2014 revision as in RFC 7230, 7231, 7232, 7233, 7234, 7235
  - HTTP/2.0 - 2015 ? HTTPS (HTTP *de facto* encryption)
- Web components
  - Infrastructure:
    - Clients, Servers
  - Content:
    - URL: naming content
    - HTML: formatting content
  - Protocol for exchanging information: HTTP, HTTPS

Web and HTTP

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,* e.g.,

https://www.gju.edu.jo/content/school-electrical…

host name | path name

URL: Uniform Record Locator

- Global identifiers of network retrievable resources

`protocol://host-name[:port]/directory-path/resource`

- *protocol*: http, ftp, https, smtp, rtsp, *etc*.
- *hostname*: DNS name, IP address
- *port:* defaults to protocol's standard port; *e.g.* http: 80, ftp 21, smtp 25, https: 443,
- *directory path*: hierarchical, reflecting file system
- *resource*: Identifies the desired resource

`e.g.: http://speedtest.tele2.net`
`ftp://speedtest.tele2.net`

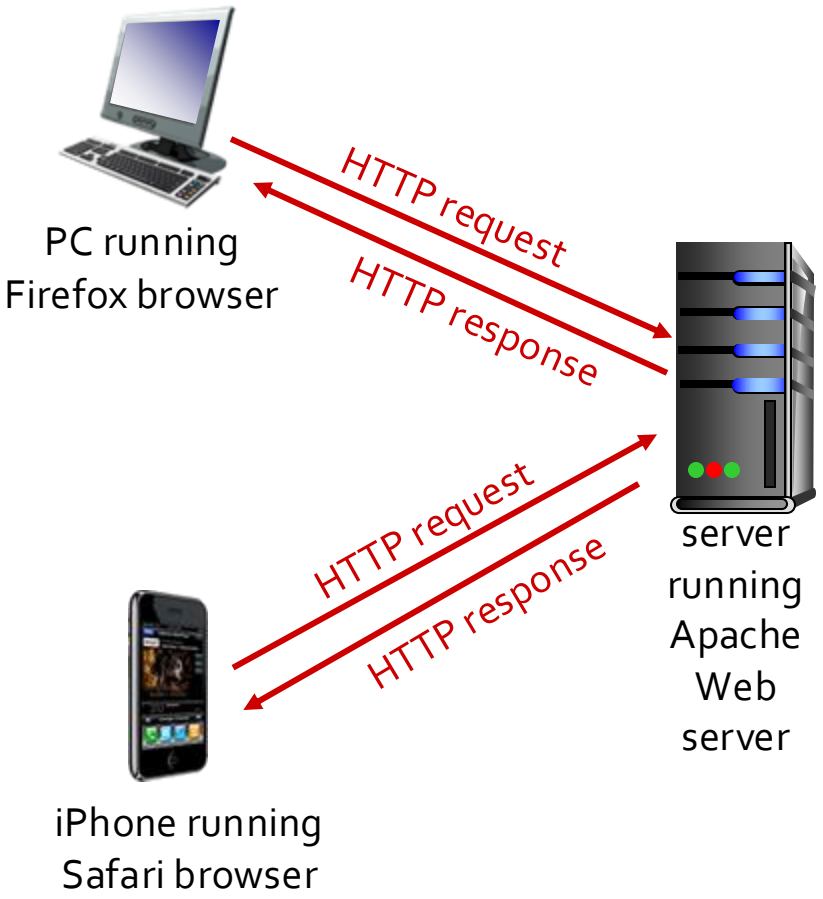
HTTP: Hypertext Transfer Protocol

A common data communication protocol based on client-server architecture

- server is "always on" and "well known" and clients initiate contact to server
- Synchronous request/reply protocol that runs over TCP, Port 80
- Stateless and with ASCII format

HTTP overview

- 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
- uses TCP:
  - 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



HTTP State and Connections

- HTTP is "stateless"
  - Each request-response treated independently
  - server maintains no information about past client requests
- protocols that maintain "state" are complex!
  - past history (state) must be maintained (client side: Cookies)

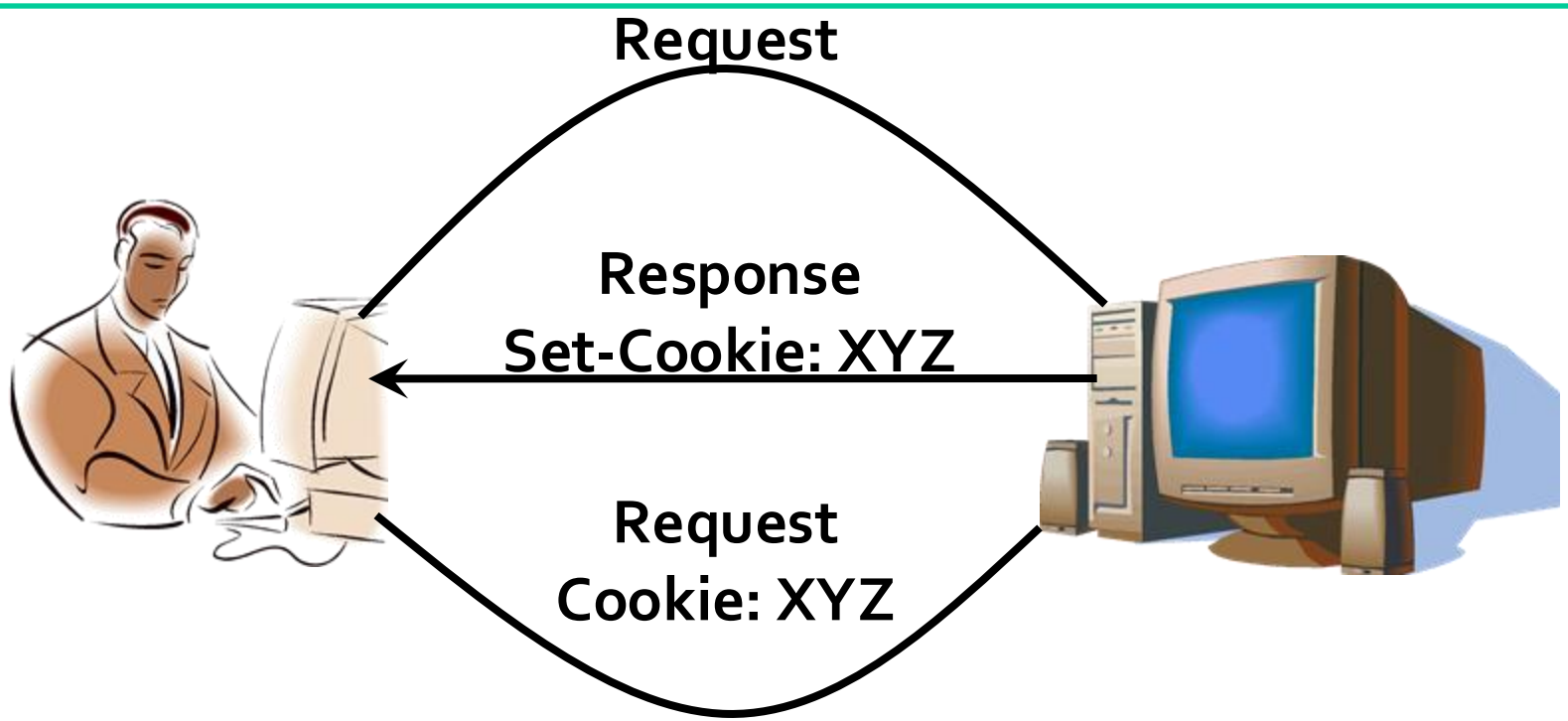
- *non-persistent HTTP connection*
  - at most one object sent over TCP connection
    - connection then closed
  - downloading multiple objects required multiple connections
- *persistent HTTP connection*
  - multiple objects can be sent over single TCP connection between client, server

Cookies: State in a Stateless Protocol

Client-side state maintenance

- Client stores small state on behalf of server
- Client sends state in future requests to the server

Can provide authentication



Non-persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

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

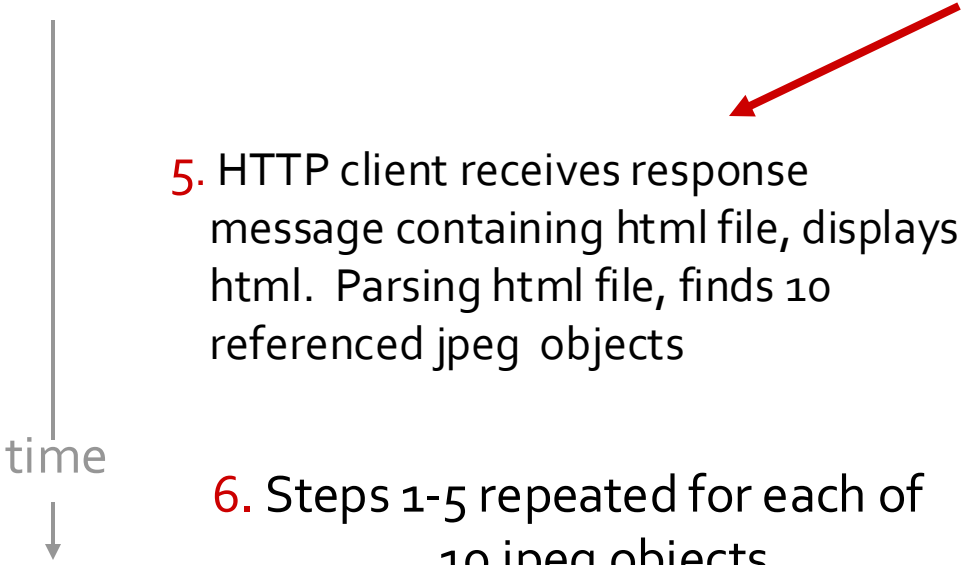
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

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

time

Non-persistent HTTP (cont.)



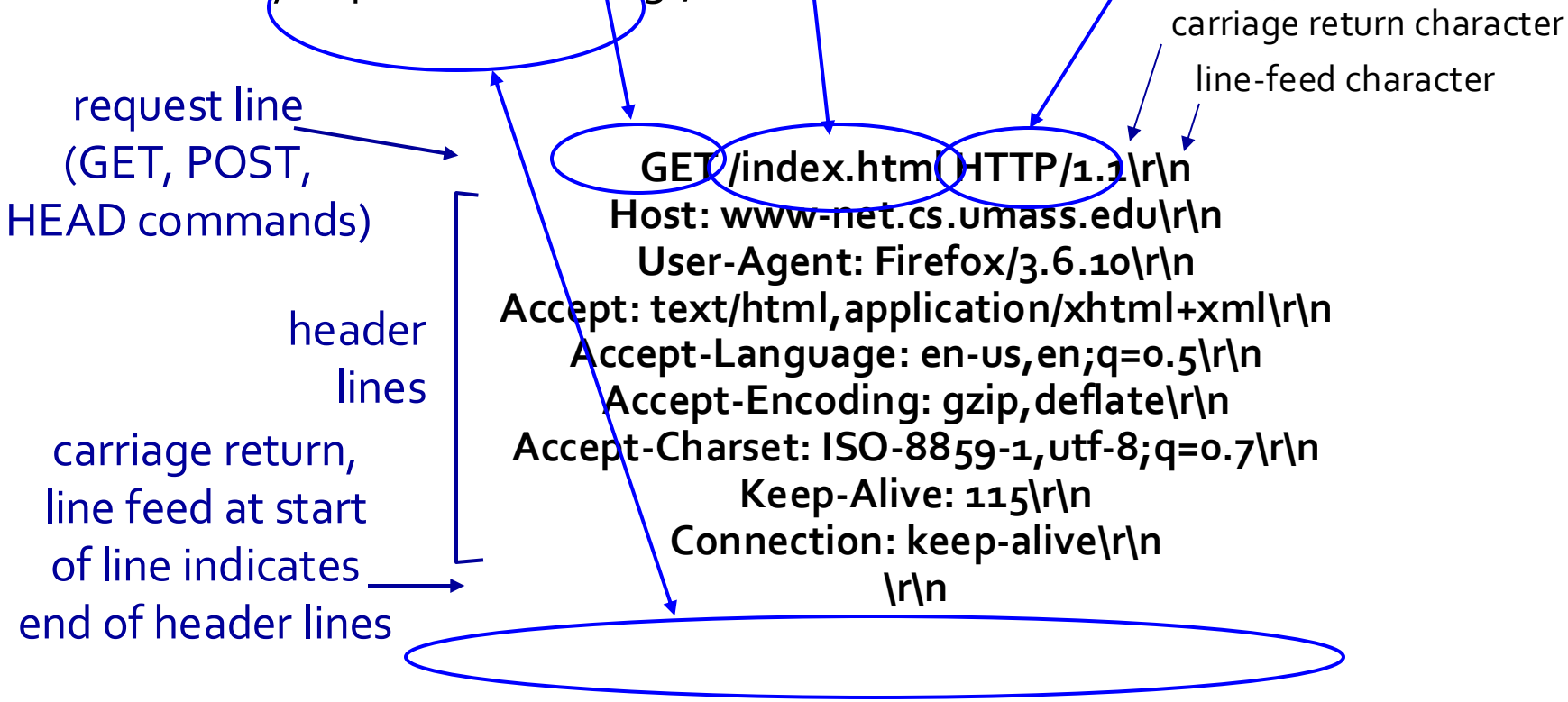
4. HTTP server closes TCP connection.

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 each of 10 jpeg objects

HTTP request message

- Request line: method, resource, and protocol version
- Request headers: provide information or modify request
- Body: optional data (*e.g.*, to "POST" data to the server)



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Method types

HTTP/1.0:

GET

POST

HEAD

- asks server to leave requested object out of response

HTTP/1.1:

GET, POST, HEAD

PUT

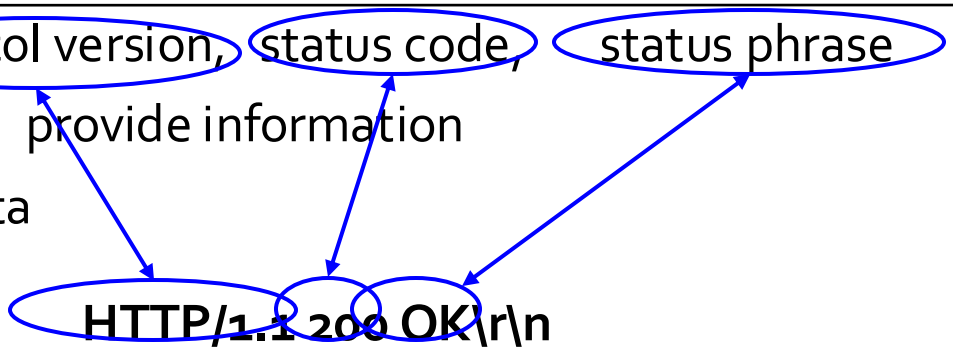
- uploads file in entity body to path specified in URL field

DELETE

- deletes file specified in the URL field

HTTP response message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional 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

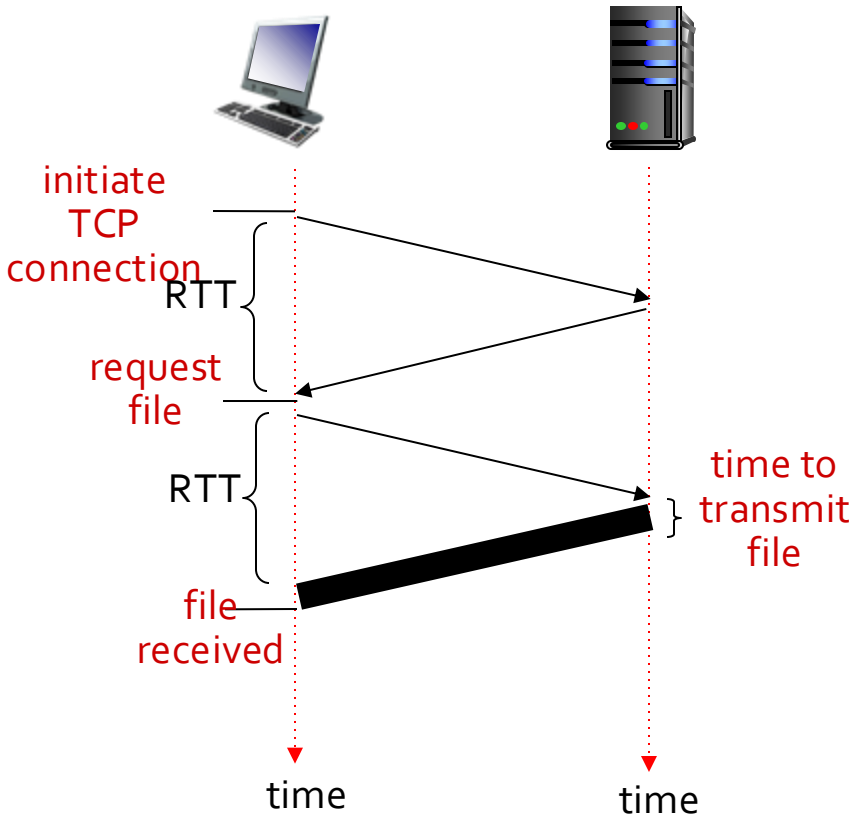
404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Non-persistent HTTP: response time

- RTT: time for a small packet to travel from client to server and back
- HTTP 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
- non-persistent HTTP response time = 2RTT+ file transmission



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

persistent HTTP:

server leaves connection open after sending response

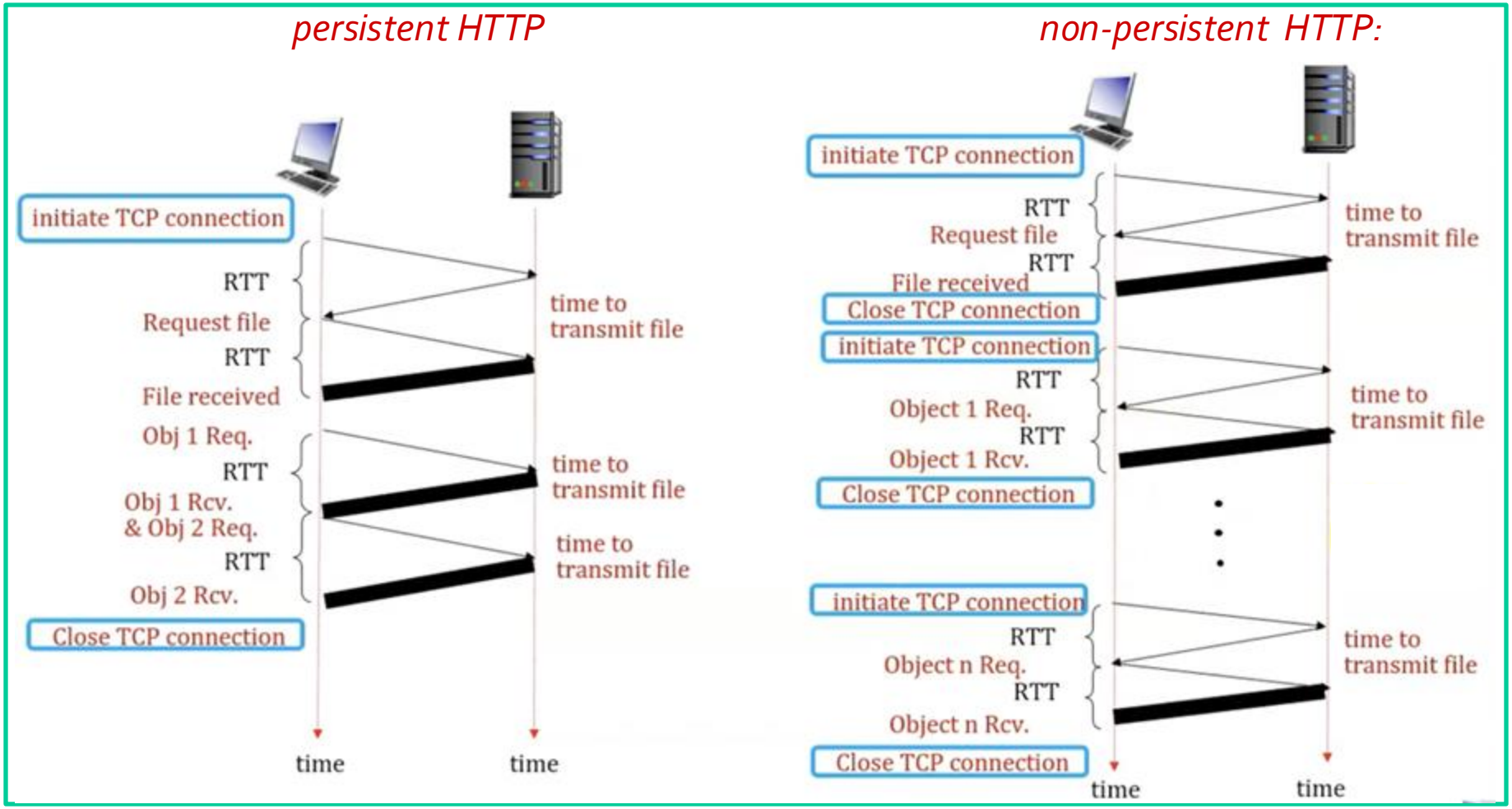
subsequent HTTP messages between same client/server sent over open connection

client sends requests as soon as it encounters a referenced object

as little as one RTT for all the referenced objects

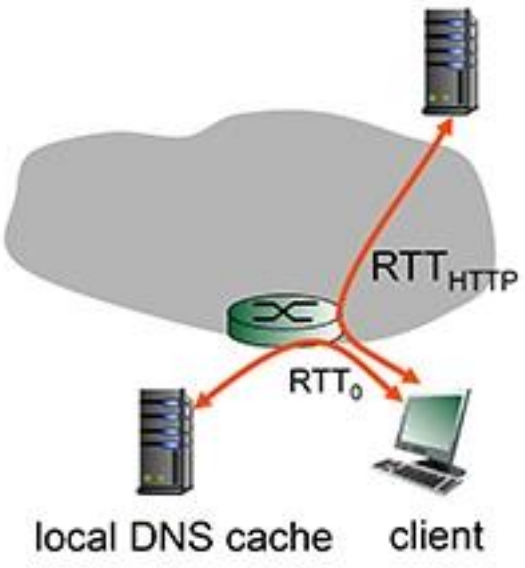
Parameters	Persistent HTTP	Non-Persistent HTTP
HTTP Version	HTTP Version 1.1	HTTP Version 1.0
Mode	It is default mode	It is not default mode
No. of RTT use	It uses one RTT for each object	It uses 2 RTT for each object
TCP Connection	TCP connection is not closed	closed after every request response
No. of request on TCP Connection	Multiple request over the single TCP connection.	Multiple request over the multiple TCP connection
Request Method	Request method are GET, HEAD, POST, PUT, DELETE, etc....	Request methods are used GET, POST and HEAD

Non- Persistent vs. Persistent HTTP



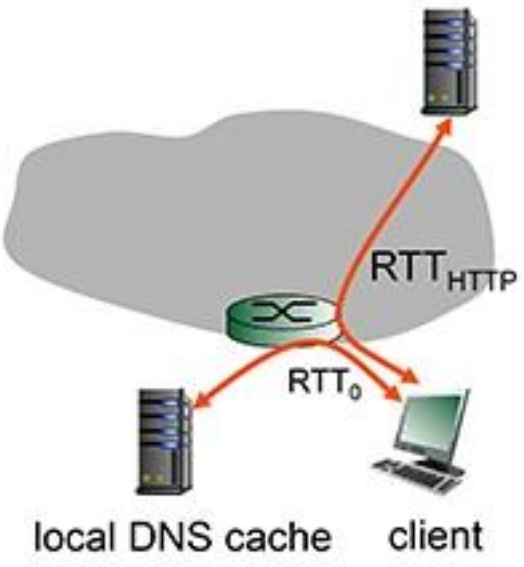
DNS and HTTP delays

- Web request: ? (DNS) + ? (TCP Connection) + ?(GET response)
  =
- Non-persistent retrieval of 8 objects = ?
- Non-persistent 5 parallel TCP connection = ?
- Persistent 5 parallel TCP connection = ?



DNS and HTTP delays

- Web request: $RTT_0 + RTT_{HTTP}$ (TCP Connection) + RTT_{HTTP} (GET response) = $RTT_0 + 2* RTT_{HTTP}$
- Reference 8 objects = $RTT_0 + 2* RTT_{HTTP} + 2* 8*RTT_{HTTP}$
- Non-persistent 5 parallel TCP connection = $RTT_0 + 2*RTT_{HTTP} + 2*RTT_{HTTP} + 2*RTT_{HTTP}$
- Persistent 5 parallel TCP connection = $RTT_0 + 2*RTT_{HTTP} + RTT_{HTTP} + RTT_{HTTP}$



Trying HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet gaia.cs.umass.edu 80

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.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!

HTML

A language to create structured documents - https://www.w3schools.com/

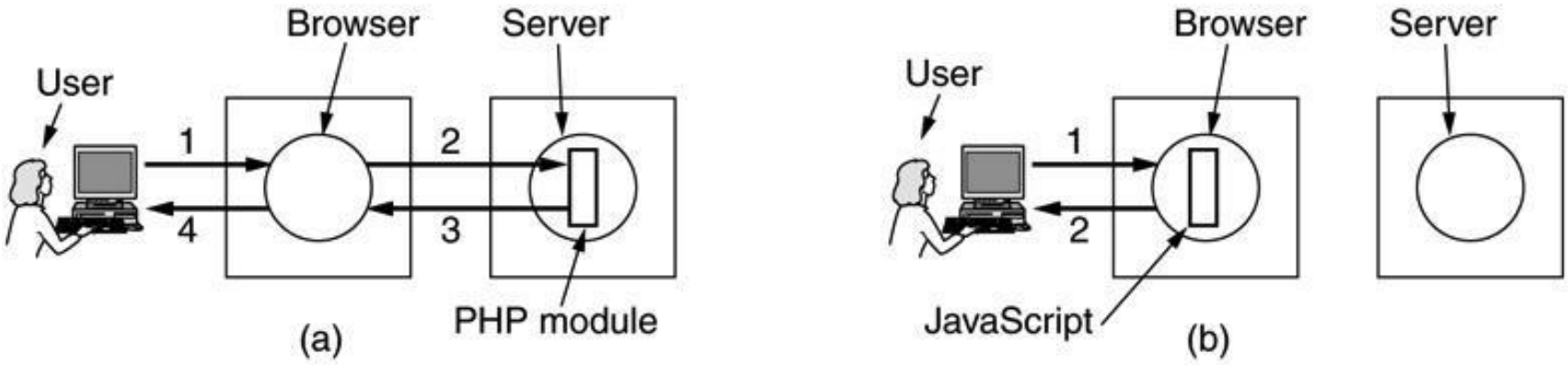
index.html

```
<html>
    <body>
        <div>
            foo
            <a href="http://google.com">Go to Google!</a>
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
    </body>
</html>
```

Tag	Description
<html> ... </html>	Declares the Web page to be written in H
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the pag
<body> ... </body>	Delimits the page's body
<h*n*> ... </h*n*>	Delimits a level *n* heading
...	Set ... in boldface
...	Set ... in italics
...	Center ... on the page horizontally
...	Brackets an unordered (bulleted) list
...	Brackets a numbered list
	Starts a list item (there is no)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
<img src="...">	Displays an image here
 ... 	Defines a hyperlink

Dynamic web pages

- Server – side scripting (Java, PHP, Python, Ruby)
  - Runs on server when a webpage is called up
  - Designed to interact with back-end databases
- Client – side scripting (JavaScript, ASP)
  - Runs on browser, embedded within HTML
  - Processes requests without call-back to the server



Summary

Today:

- Application Layer Protocols
- Application architecture
- Transport protocol and service
- The Web: URL and HTTP
- HTML, Dynamic web pages

Camino discussion:

- Reflection
- Exit ticket

Next time:

- read 2.7 of K&R (socket programming)
- follow on Canvas! material and announcements

Any questions?