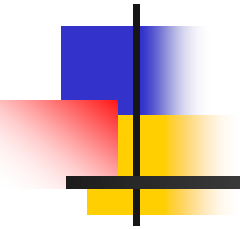
Application Layer



Instructor: C. Pu (Ph.D., Assistant Professor)

Lecture 05

puc@marshall.edu

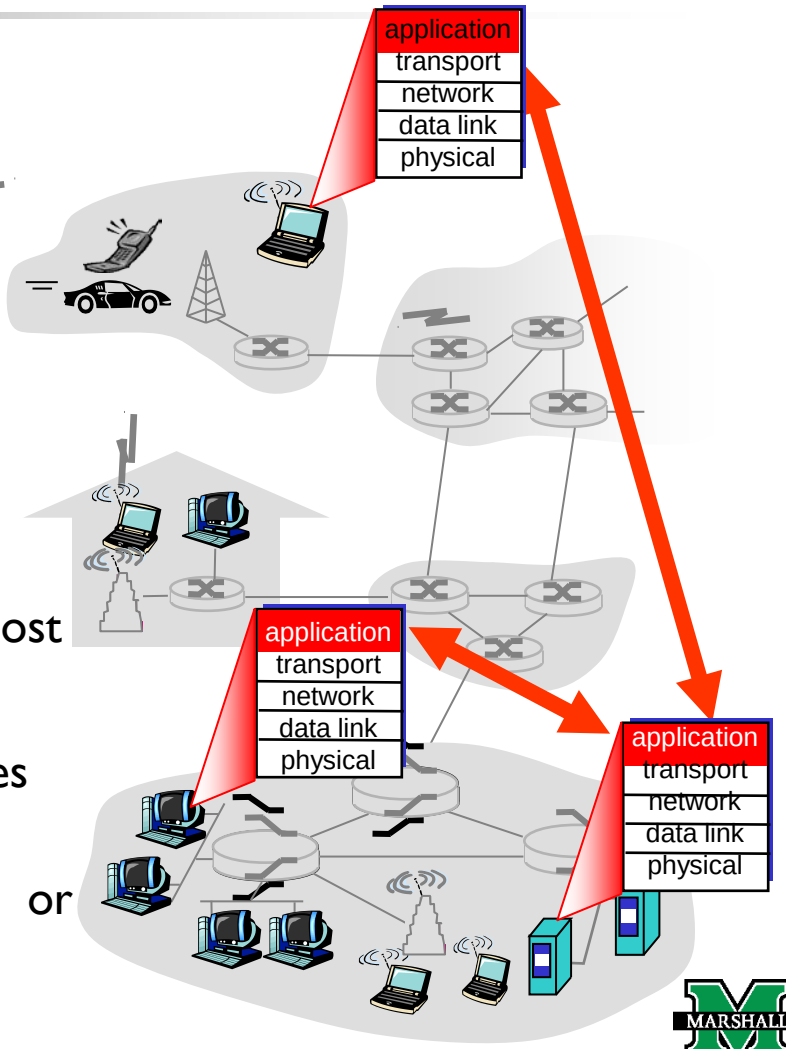
Creating a Network App

- Suppose you have an idea for a new network app.
  - app. might be a great service to humanity
  - app. might please your professor
  - app. might bring you great wealth
  - app. might be fun to develop

- How do you transform the idea into a real-world network app.?

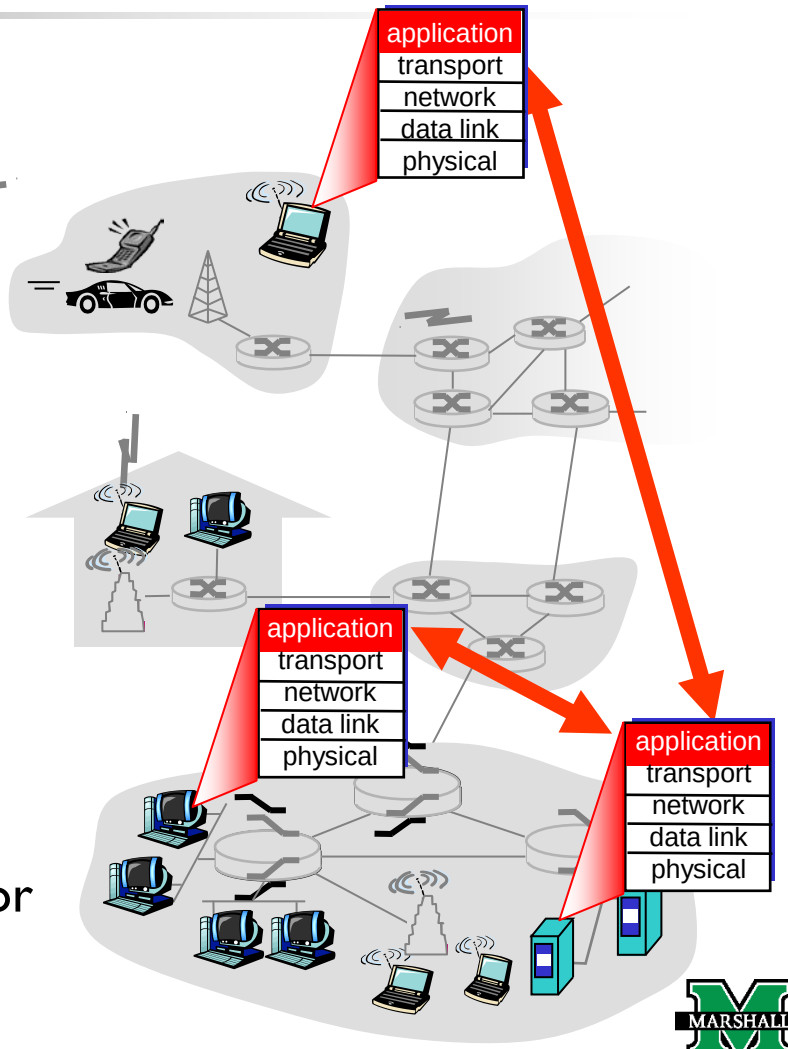
Creating a Network App

- the core of netw. app. dev. is writing programs that
  - run on *different end systems*
  - communicate with each other over network
  - e.g., web app.
    - browser program in user's host
    - web server program in web server host
  - e.g., p2p file-sharing
    - program in each host that participates in the file-sharing community
      - program might be similar or identical



Creating a Network App

- when developing netw. app., write software that runs on multiple end systems
  - program in C, Java, or Python
- **no** need to write software for network-core devices
  - network-core devices *__do not__* run user applications
    - network-core devices function at lower layers – network layer and below
  - applications on end systems allows for rapid app development, propagation

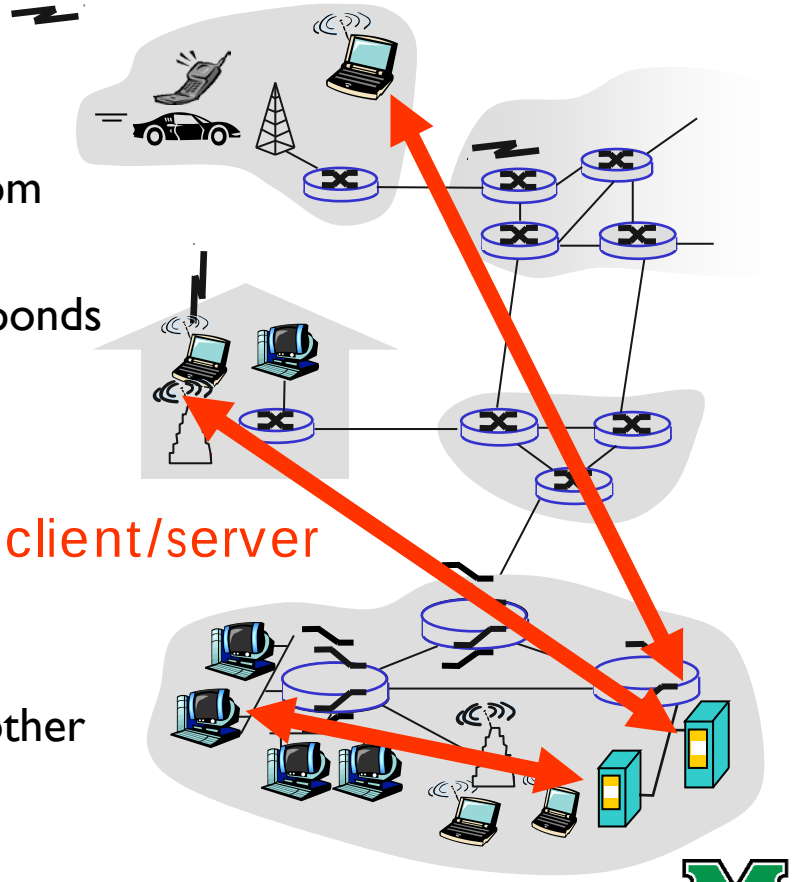


Network Application Architectures

- Before diving into coding, what is the architecture plan for your app.?
- application architecture vs. network architecture
  - network architecture: the five-layer Internet architecture discussed before
    - for app. dev., it is fixed and provides a set of service to app.
- **application architecture**
  - designed by the application developer
  - dictates how the application is structured over the various end systems
- choosing application architecture
  - **client-server**
  - **peer-to-peer (P2P)**

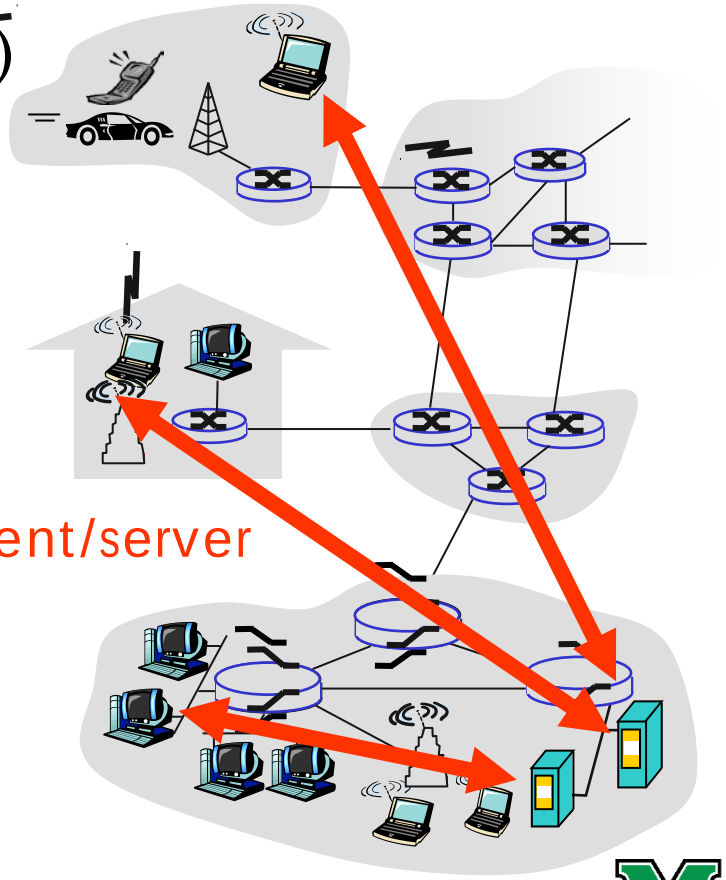
Client-Server Architecture

- **server**:
  - always-on host
    - services requests from other hosts
    - e.g., web server services requests from browsers running on client hosts
      - when receiving a request, it responds with requested object
  - fixed, well-known, permanent IP address
    - client can always contact server
- **clients**:
  - communicate with server
  - do not communicate directly with each other
  - may be intermittently connected
  - may have dynamic IP addresses



Client-Server Architecture

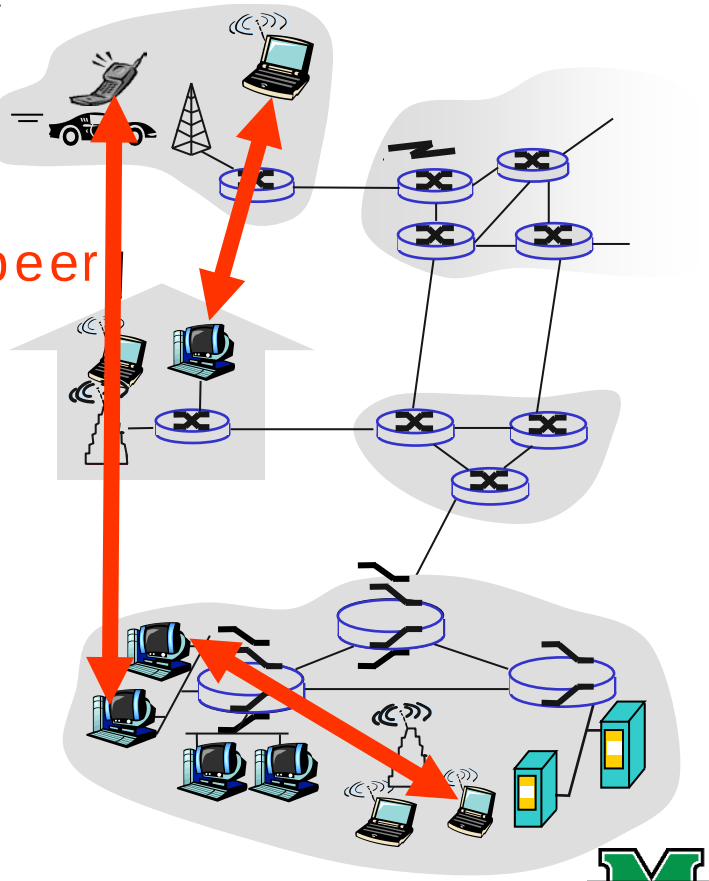
- a single-server is not enough (become overwhelmed with large network traffic)
  - **data center**
    - host a large number of servers
  - TikTok, WeChat, Google, …
  - Google has 30 to 50 data centers
    - handle search, YouTube, Gmail, and other services



Google Data Centers

Pure P2P Architecture

- minimal (or no) reliance on dedicated server
- application exploits direct communications between pairs of intermittently connected hosts, called **peers**
  - peers are not owned by the service provider
  - peers are users' desktop and laptop
  - end hosts in home, univ., and office
- communicating without passing through a dedicated server, the architecture is called **peer-to-peer**
  - e.g., file sharing application (BitTorrent)

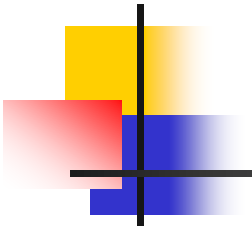


Processes Communicating

- before building netw. app., you need to understand how the programs running in multiple end systems communicate with each other
- it is not actually programs but **processes** that communicate
- **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 across the computer network
    - a sending process creates and sends messages into the network
    - a receiving process receives these messages and responds by sending messages back

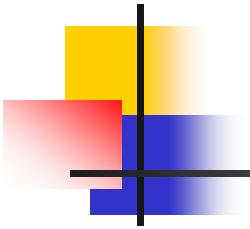
Client and Server Processes

- a netw. app. consists of pairs of processes that send msgs to each other over a network
  - Web application: client browser process vs. Web server process
  - P2P file-sharing: a file is transferred from a process in one peer to a process in another peer
- **client process** vs. **server process**
  - who is client and server in Web and P2P?
  - client process: process that initiates communication
  - server process: process that waits to be contacted
- in some applications, a process can be both a client and a server
  - p2p file sharing
- in any given communication session, one client process and one server process



Client and Server Processes

In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the **client**. The process that waits to be contacted to begin the session is the **server**.

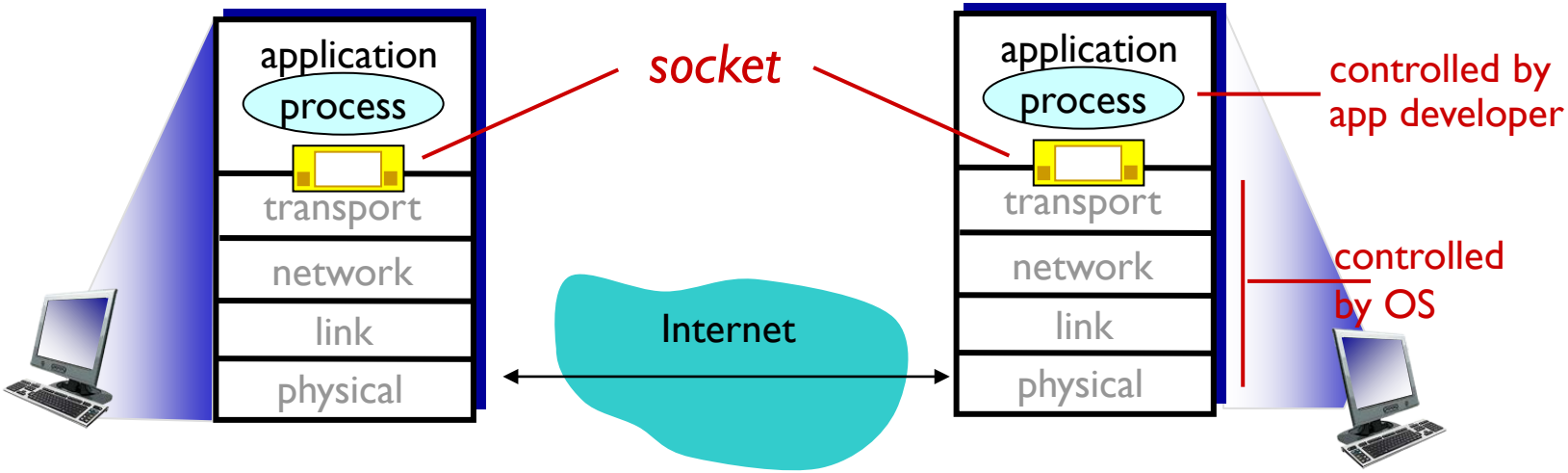


Sockets

- Most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other
- Any message sent from one process to another must go through the underlying network
- A process sends messages into, and receives messages from, the network through a ***software interface*** called a ***socket***
- Analogy,
  - house ~ process; door ~ socket

Sockets

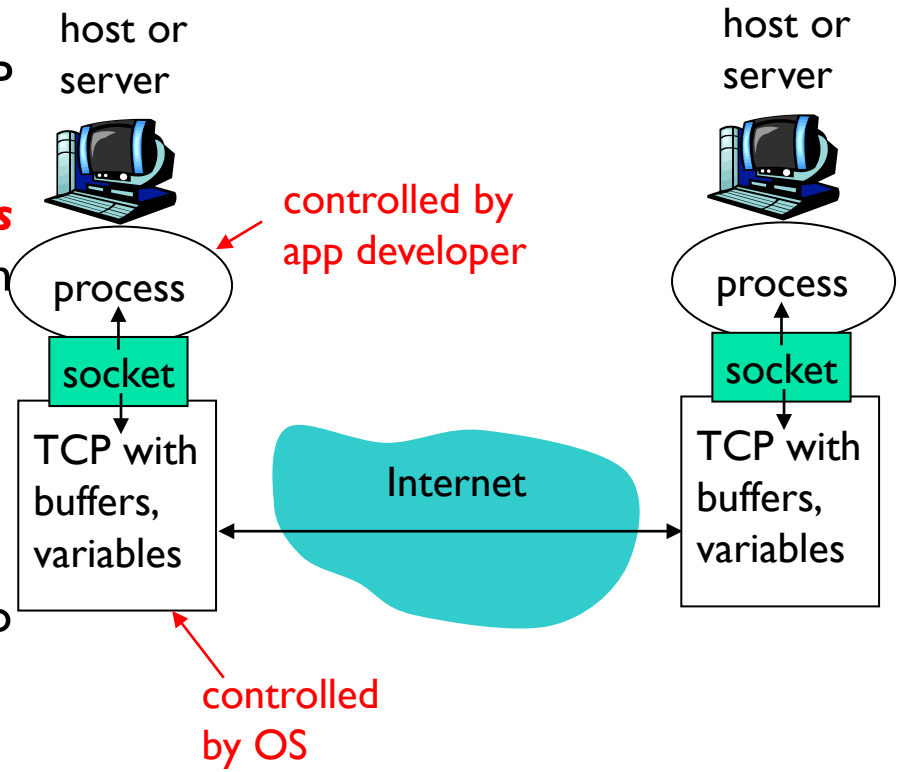
socket communication between two process that communicate over the Internet



socket: the interface between application layer and transport layer

Addressing Processes

- to receive messages, process must have ***identifier***
- host device has unique 32-bit IP address
- ***Identifier*** includes both ***IP address*** and ***port numbers*** associated with process on host.
  - e.g.,
    - HTTP server: 80
    - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80



Transport Services Available to Applications

- socket: the interface between the application process and the transport-layer protocol
  - Application pushes message through the socket
  - Transport-layer protocol gets messages to the socket of the receiving process
- many networks provide more than one transport-layer protocols
- for application, choose one of the available protocols
  - study the services provided by the protocols
  - pick the protocol with the services that best match application's needs

Transport Services Available to Applications

- reliable data transfer
  - packet get lost
    - overflow buffer
    - discarded by end host
  - for some app., packet lost can be serious
    - Email, file transfer, …
  - guarantee: the data sent by one host is delivered correctly and completely to the other host
    - **reliable data transfer**
  - if transport-layer protocol does not provide **reliable data transfer**
    - **loss-tolerant application**

Transport Services Available to Applications

- timing
  - some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"
  - guarantee: every bit arrives at the receiver's socket no more than 100 msec later
- throughput
  - some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
  - other apps ("elastic apps") make use of whatever throughput they get
  - guarantee: available throughput at some specified rate
- security
  - Encryption, data integrity, …

Transport Services Provided by the Internet

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 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

Internet Transport Protocols Services

- **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:* throttle sender when network overloaded
  - *does not provide:* timing, minimum throughput guarantees, security
- **UDP service:**
  - *connectionless:* between sending and receiving process
  - *unreliable data transfer:* between sending and receiving process
  - *does not provide:* connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

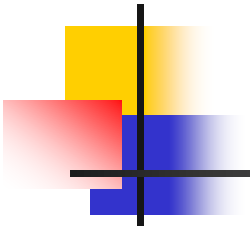
Internet apps: Application, Transport Protocols

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

Application-Layer Protocols

- an application-layer protocol defines
  - The types of messages exchanged, for example, request messages and response messages
  - The syntax of the various message types, such as the fields in the message and how the fields are delineated
  - The semantics of the fields, that is, the meaning of the information in the fields
  - Rules for determining when and how a process sends messages and responds to messages

An application-layer protocol defines how an application's process, running on different end systems, pass messages to each other.



HTTP

- HyperText Transfer Protocol (HTTP)
  - Web's application-layer protocol
  - at the heart of the Web
- HTTP is implemented in two programs:
  - Client program
  - Server program
- client program and server program
  - executing on different end systems
  - talking to each other by exchanging HTTP messages
- HTTP defines
  - the structure of message
  - how the client and server exchange the messages

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
  - example URL:

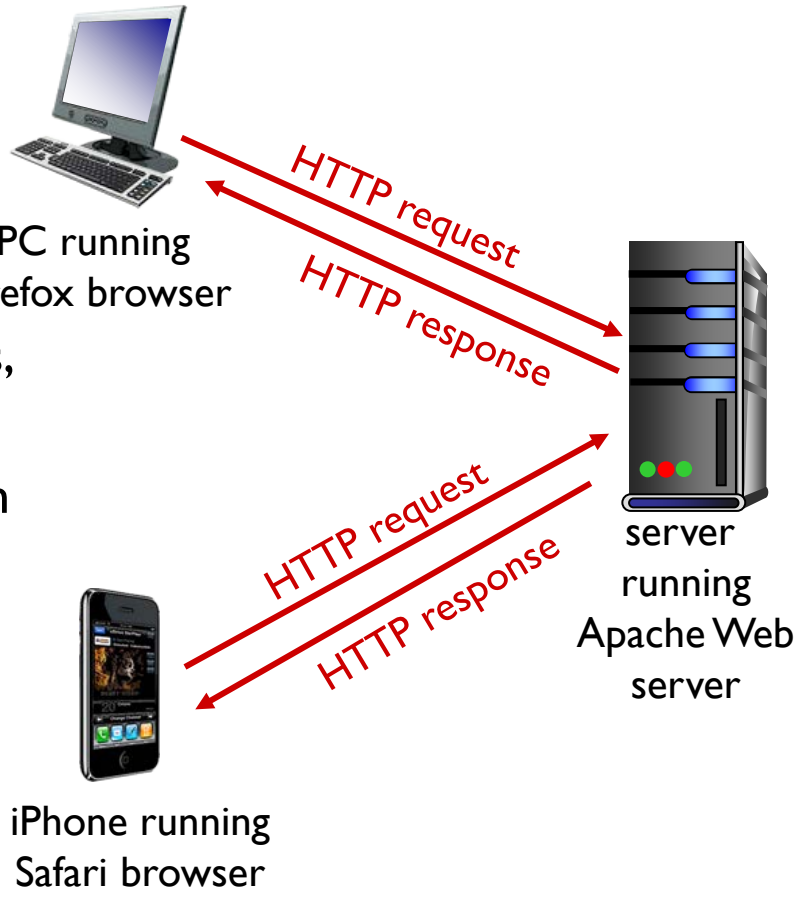
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP Overview

HTTP: hypertext transfer protocol

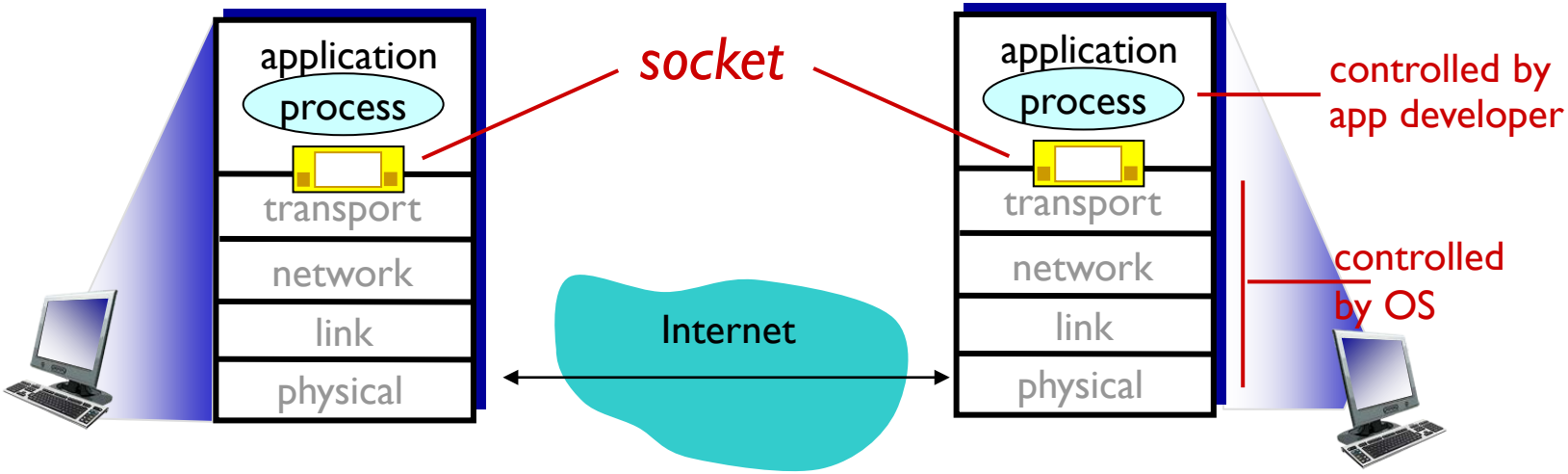
- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests



HTTP Overview (cont.)

HTTP 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 Overview (cont.)

HTTP 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 is "stateless"

- server maintains no information about past client requests
- if a client asks for the same object in a while, the server will server the object to client

Protocols that maintain "state" are complex!

- ❒ past history (state) must be maintained
- ❒ if server/client crashes, their views of "state" may be inconsistent, must be reconciled