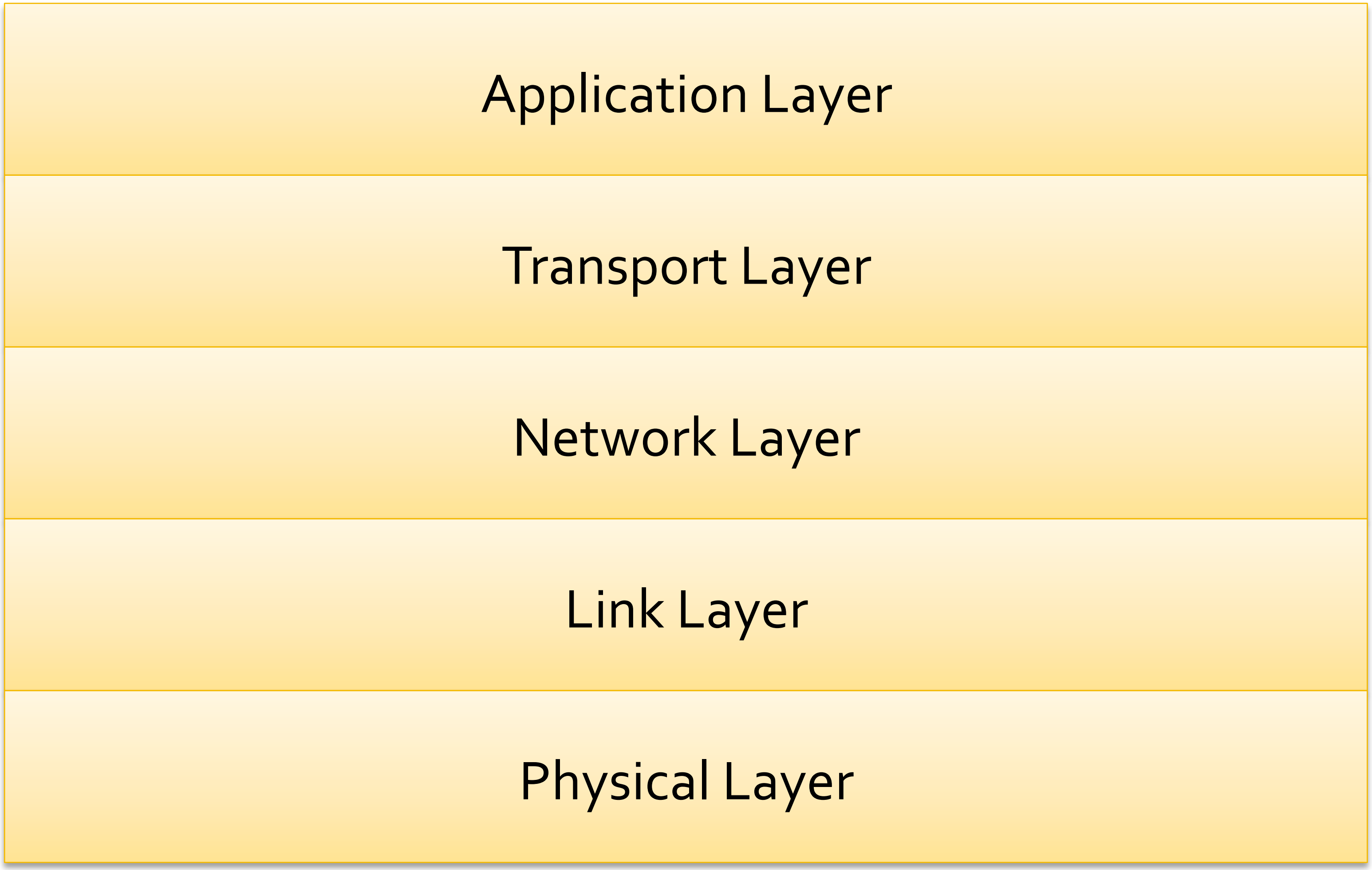
ELEC / COMP 177 – Fall 2011

Computer Networking → Sockets

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

Recap – Network Model



Application Layer

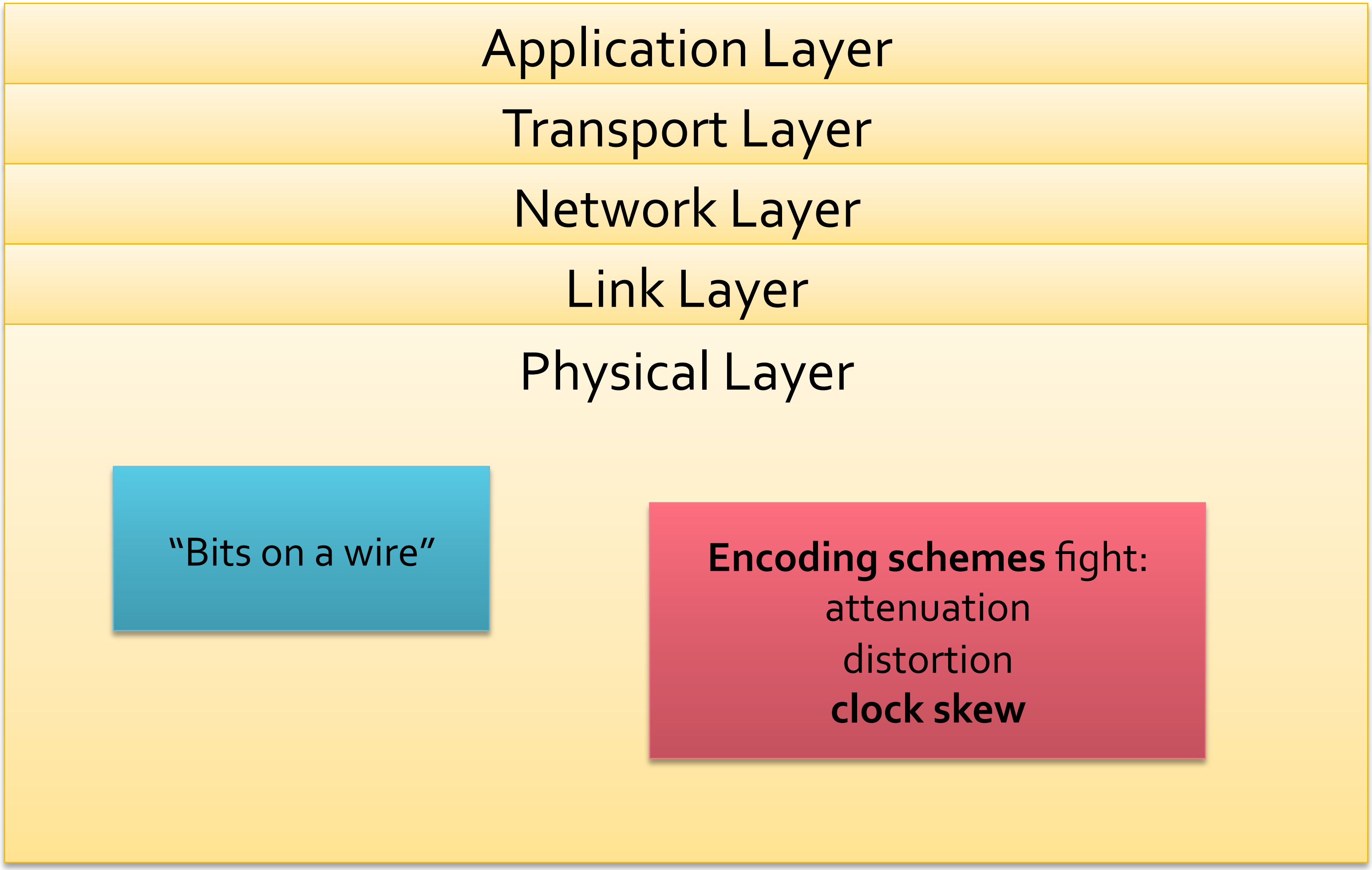
Transport Layer

Network Layer

Link Layer

Physical Layer

Recap – Physical Layer



Application Layer

Transport Layer

Network Layer

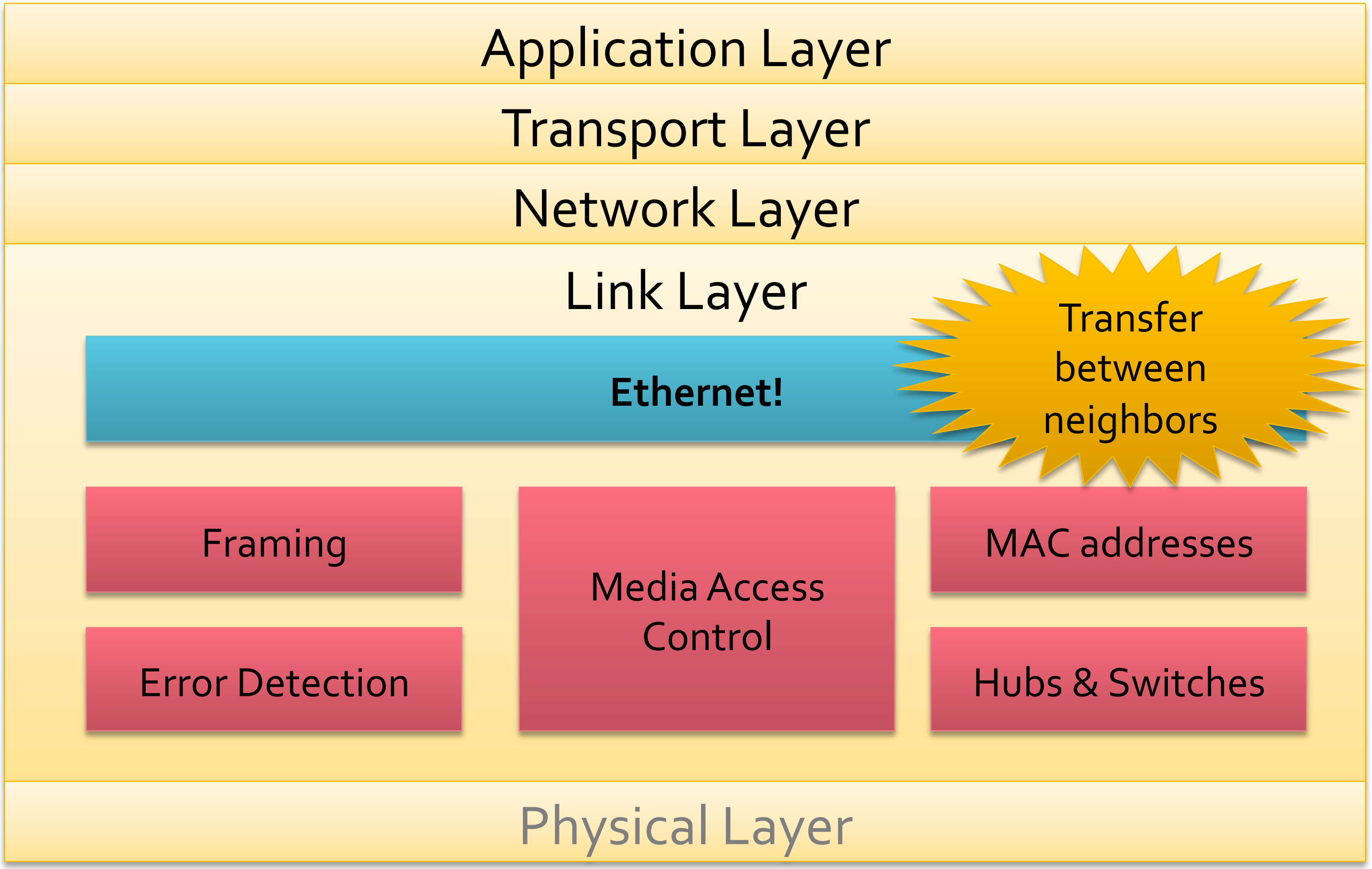
Link Layer

Physical Layer

"Bits on a wire"

Encoding schemes fight:
attenuation
distortion
clock skew

Recap – Link Layer



Application Layer

Transport Layer

Network Layer

Link Layer

Ethernet!

Transfer between neighbors

Framing

Media Access Control

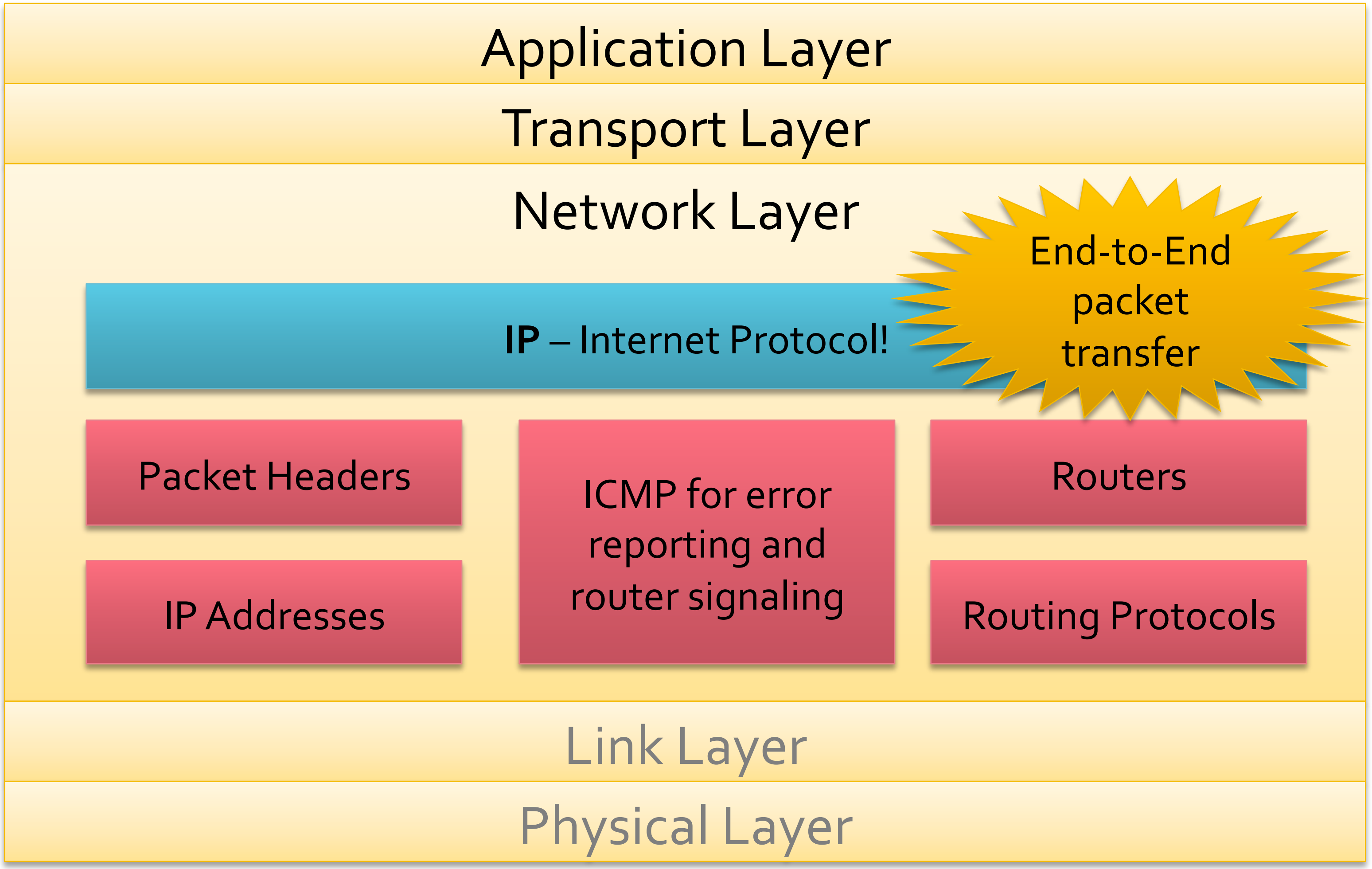
MAC addresses

Error Detection

Hubs & Switches

Physical Layer

Recap – Network Layer



Application Layer

Transport Layer

Network Layer

End-to-End packet transfer

IP – Internet Protocol!

Packet Headers

IP Addresses

ICMP for error reporting and router signaling

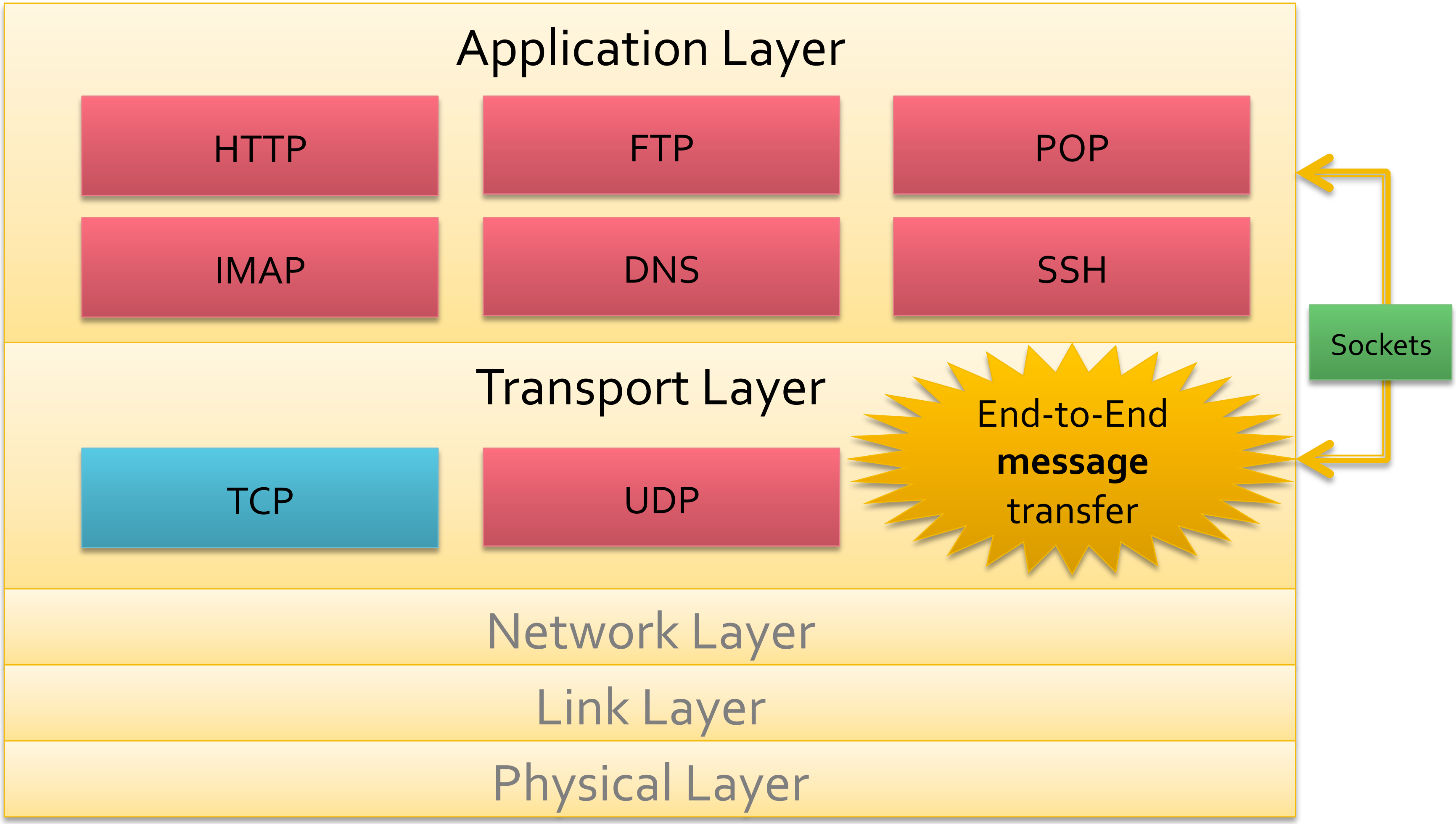
Routers

Routing Protocols

Link Layer

Physical Layer

Recap – Application and Transport Layers



Sockets

Why Do We Have Sockets?

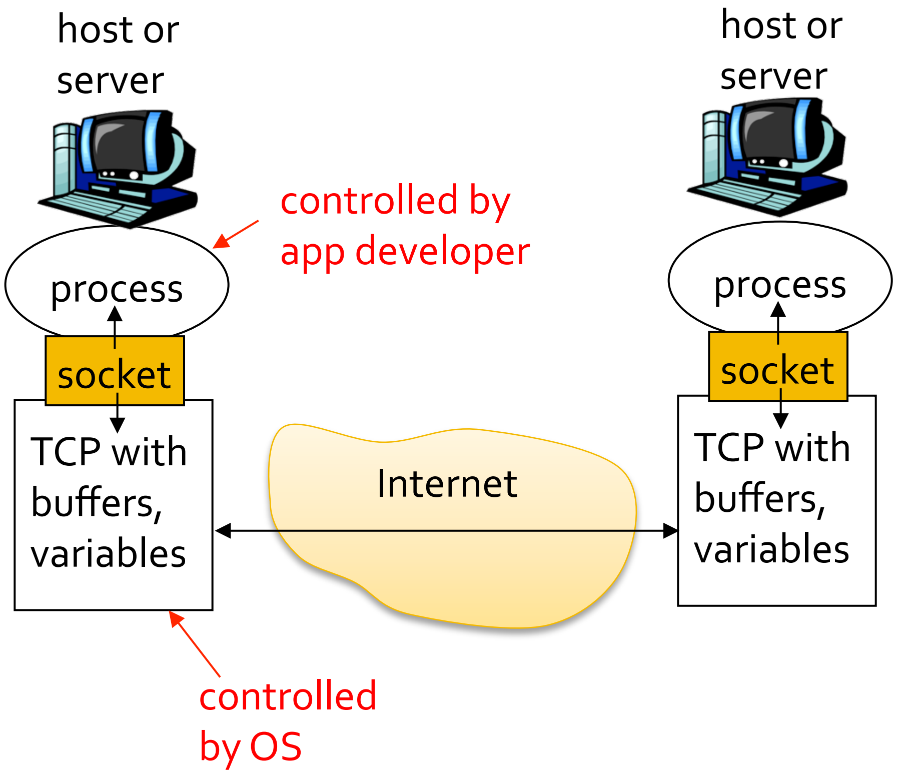
- Challenge – **Inter-process communication**
- A **process** is an independent program running on a host
  - Separate memory space
- How to processes communicate with other processes
  - On the same host?
  - On different hosts?
- Send **messages** between each other

What is a Socket?

- An interface between process (application) and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - Reliable vs. best effort
    - Connection-oriented vs. connectionless
- Once configured the application can
  - Pass data to the socket for network transmission
  - Receive data from the socket (transmitted through the network by some other host)

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
- API allow customization of socket
  - Choose transport protocol
  - Choose parameters of protocol

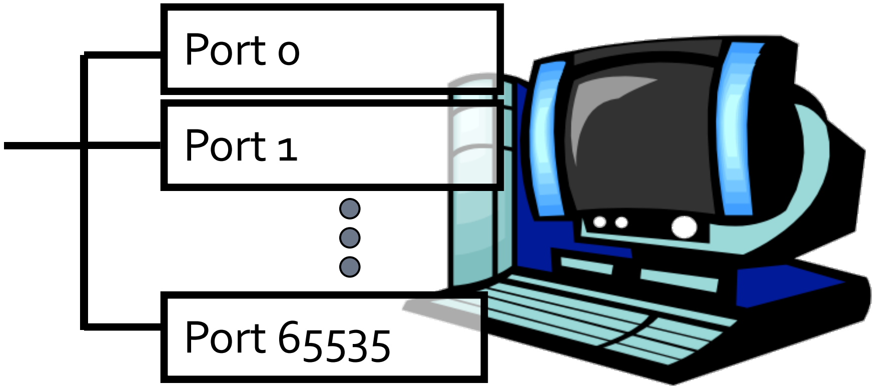

host or server
process
socket
TCP with buffers, variables
controlled by app developer
Internet
controlled by OS
host or server
process
socket
TCP with buffers, variables

Addressing Processes

- To receive messages, each process on a host must have an **identifier**
  - IP addresses are unique
  - **Is this sufficient?**
- No, there can thousands of processes running on a single machine (with 1 IP address)
- Identifier must include
  - IP address
  - **and** port number (example: 80 for web)

Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
  - FTP (20, 21), Telnet (23), HTTP (80), etc…
- Outgoing ports (on clients) can be dynamically assigned by OS in upper region (above 49,152) – called **ephemeral ports**
- See http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers



Client versus Server Processes

- **Client** process
  - Process that initiates communication
- **Server** process
  - Process that waits to be contacted
- How does this change in P2P (peer-to-peer) applications?
  - Those applications contain both client and server processes

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

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

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
  - Throttle sender when network overloaded
- 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?

Socket Programming

- Goal: learn how to build client/server application that communicate using sockets
- Socket API
  - Introduced in BSD4.1 UNIX, 1981
  - Client/server paradigm
  - Two types of transport services via socket API:
    - UDP
    - TCP

socket

A *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket Programming Basics

- Server must be running before client can send anything to it
- Server must have a socket (door) through which it receives and sends messages
- Similarly client needs a socket
- Socket is locally identified with a port number
  - *Analogous to the apt # in a building*
- Client needs to know server IP address and socket port number
  - **How do we find this?**

Socket Programming with UDP

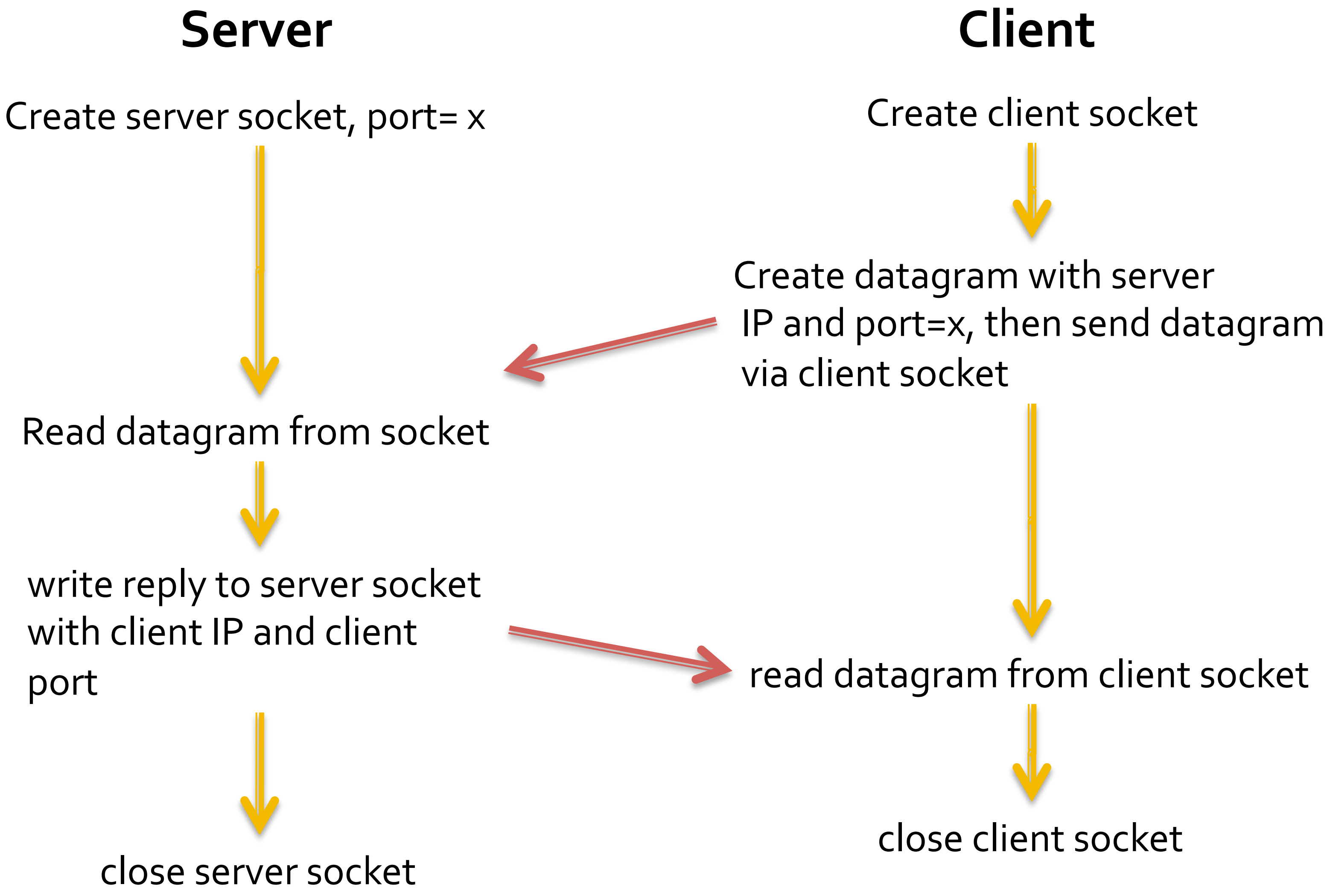
- UDP: no "connection" between client and server
  - No handshaking
  - Sender explicitly attaches IP address and port of destination to each message

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

  - OS attaches IP address and port of sending socket to each segment
  - Server can extract IP address, port of sender from received segment

Client/Server Socket Interaction with UDP



Server

Create server socket, port= x

Read datagram from socket

write reply to server socket with client IP and client port

close server socket

Client

Create client socket

Create datagram with server IP and port=x, then send datagram via client socket

read datagram from client socket

close client socket

UDP Question

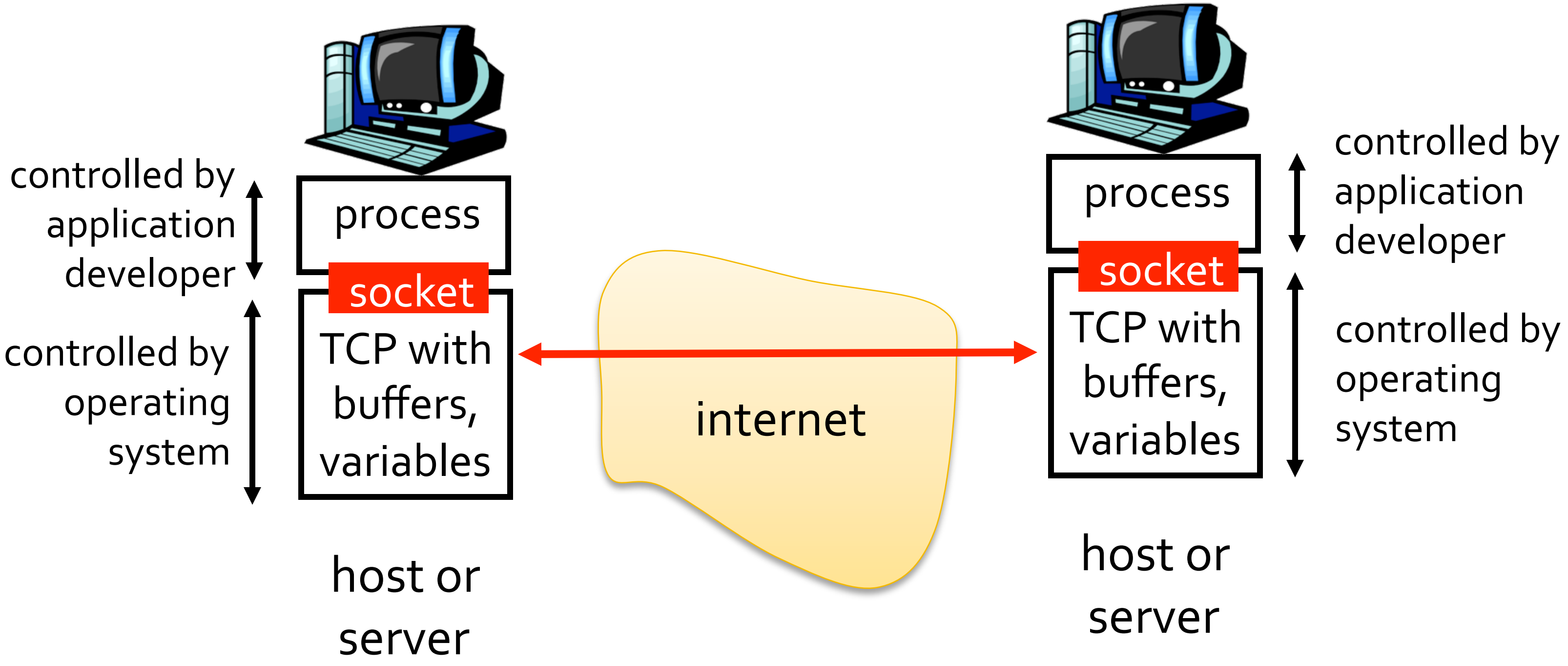
- Can the client send a segment to server without knowing the server's IP address and port number?
- Could use **broadcast IP address** of the subnet to get around lack of IP address knowledge…
- No way to avoid knowing port number…

UDP Observation

- Each UDP message is self-contained and complete
- Each time you read from a UDP socket, you get a complete message as sent by the sender
  - *That is, assuming it wasn't lost in transit!*
- Think of UDP sockets as putting a stamp on a letter and sticking it in the mail

Socket Programming using TCP

TCP service: reliable transfer of bytes from one process to another


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
controlled by application developer
controlled by operating system
host or server

Socket Programming *with TCP*

Client must contact server

- Server process must first be running
- Server must have created socket (door) that welcomes client's contact

Client contacts server by:

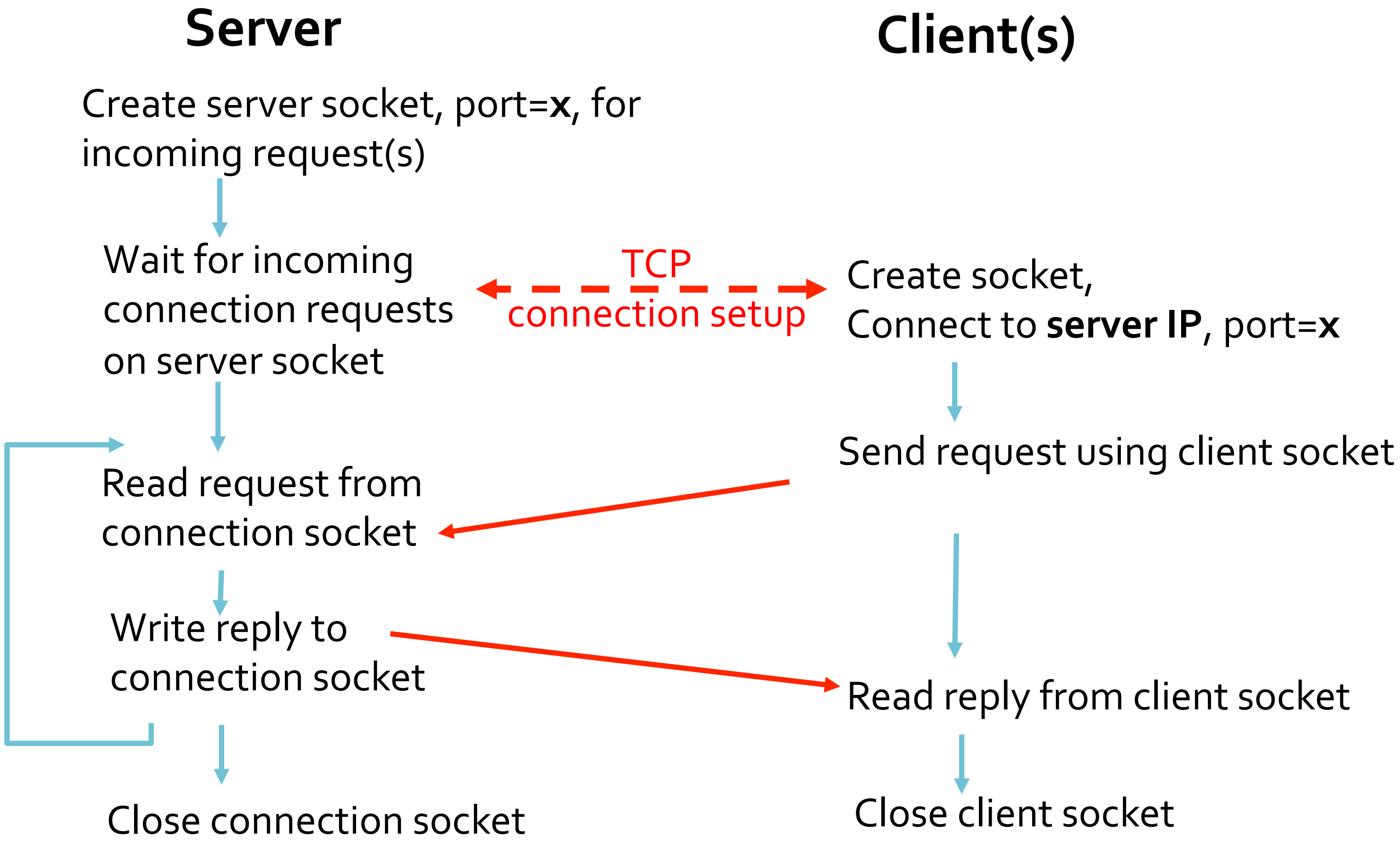
- Creating client-local TCP socket
- Specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/Server Socket Interaction with TCP



Server

Create server socket, port=**x**, for incoming request(s)

Wait for incoming connection requests on server socket

Read request from connection socket

Write reply to connection socket

Close connection socket

TCP connection setup

Client(s)

Create socket,
Connect to **server IP**, port=**x**

Send request using client socket

Read reply from client socket

Close client socket

What is a Stream?

- A **stream** is a sequence of characters that flow into or out of a process.
- An **input stream** is attached to some input source for the process, e.g., keyboard or socket.
- An **output stream** is attached to an output source, e.g., monitor or socket.

TCP Observations

- TCP sockets are **stream** based
  - At the receiver, each read on a UDP socket is **not** guaranteed to produce the same number of bytes as were sent by the transmitter
  - All you know is that you'll get the **next** set of bytes
    - Keep reading, and eventually you'll get them all
  - Your **application** has to have some way to separate a stream of bytes into discrete messages
- Server has two types of sockets
  - One that listens for incoming connections
  - One on a per-client basis after a connection is opened

Upcoming Events

- **Class**
  - Topics for next week: Transport Layer operation (TCP and UDP in detail)
- **Lab**
  - This afternoon @ 2pm
- **Homework 2**
  - Assigned **next Tuesday**
  - Due in one week
  - *Haven't posted it yet!*