ELEC / COMP 177 – Fall 2011

Computer Networking → Exam Review

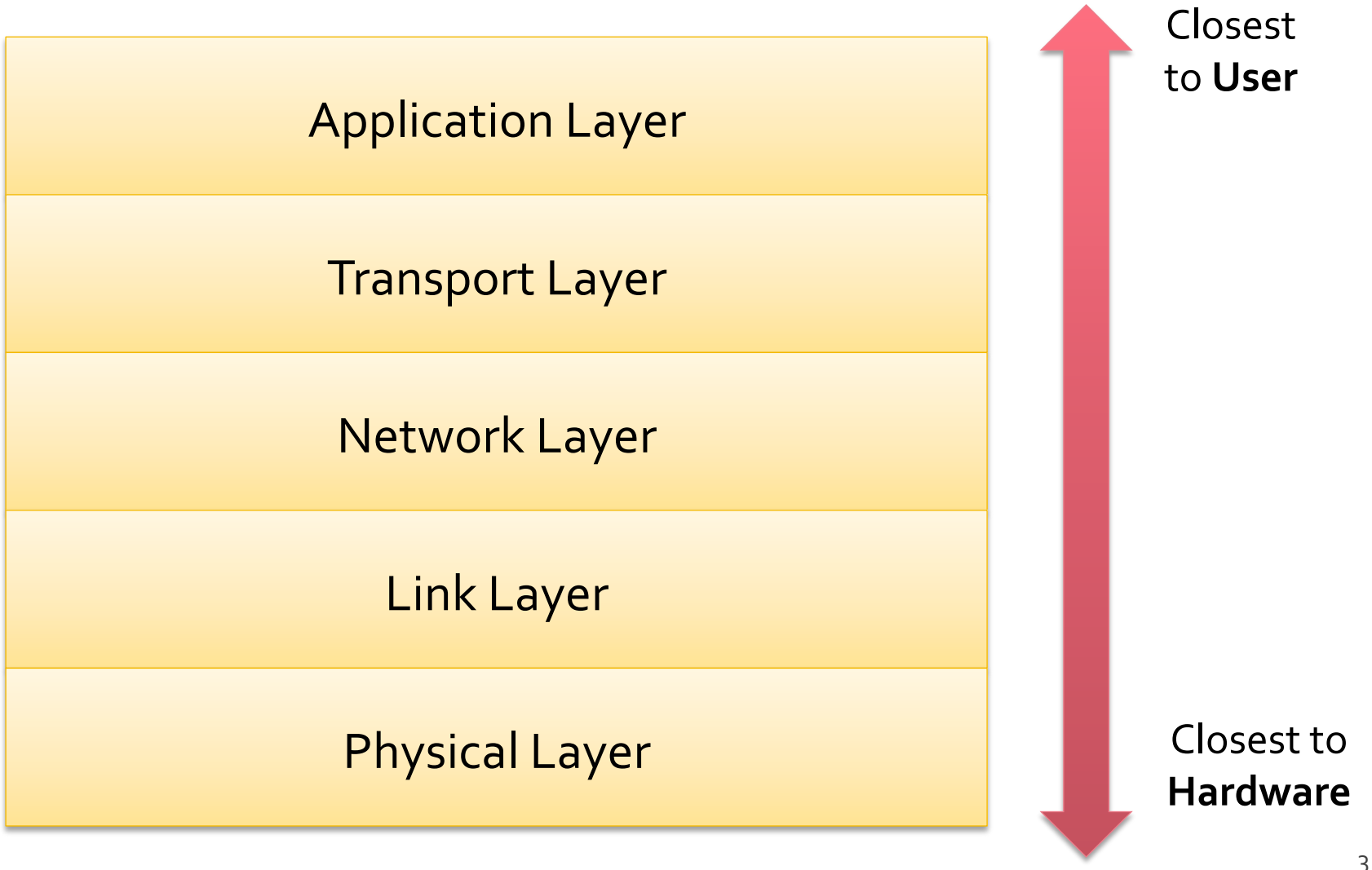
Some slides from Kurose and Ross, *Computer Networking*, 5th Edition

Midterm Exam

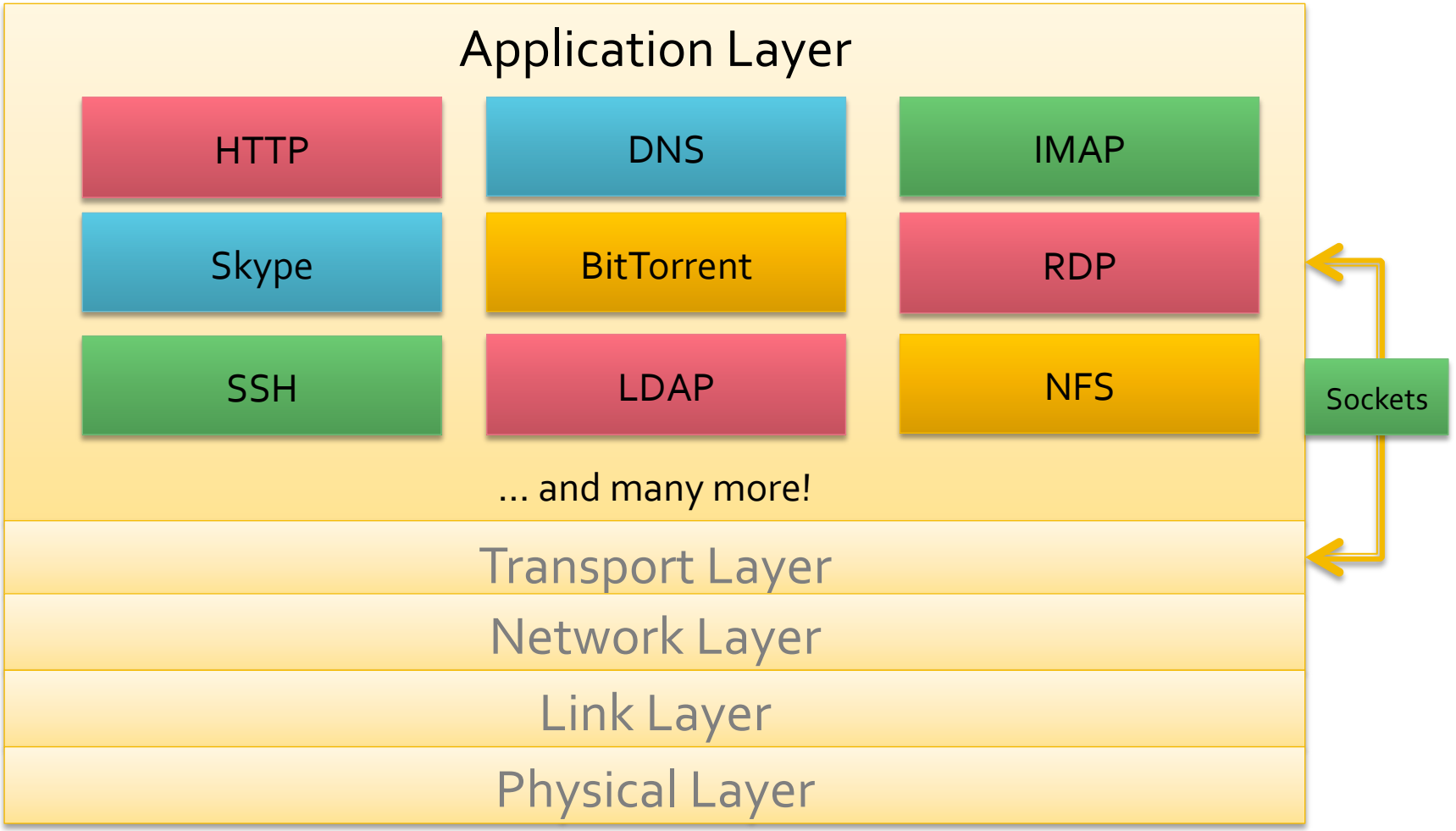
- **Midterm Exam – Tuesday, October 18th**
- **Format**
  - Short answer problems
  - No questions on programming

  - Closed notes
  - Closed book
  - Closed internet
  - Closed friends
  - Etc…

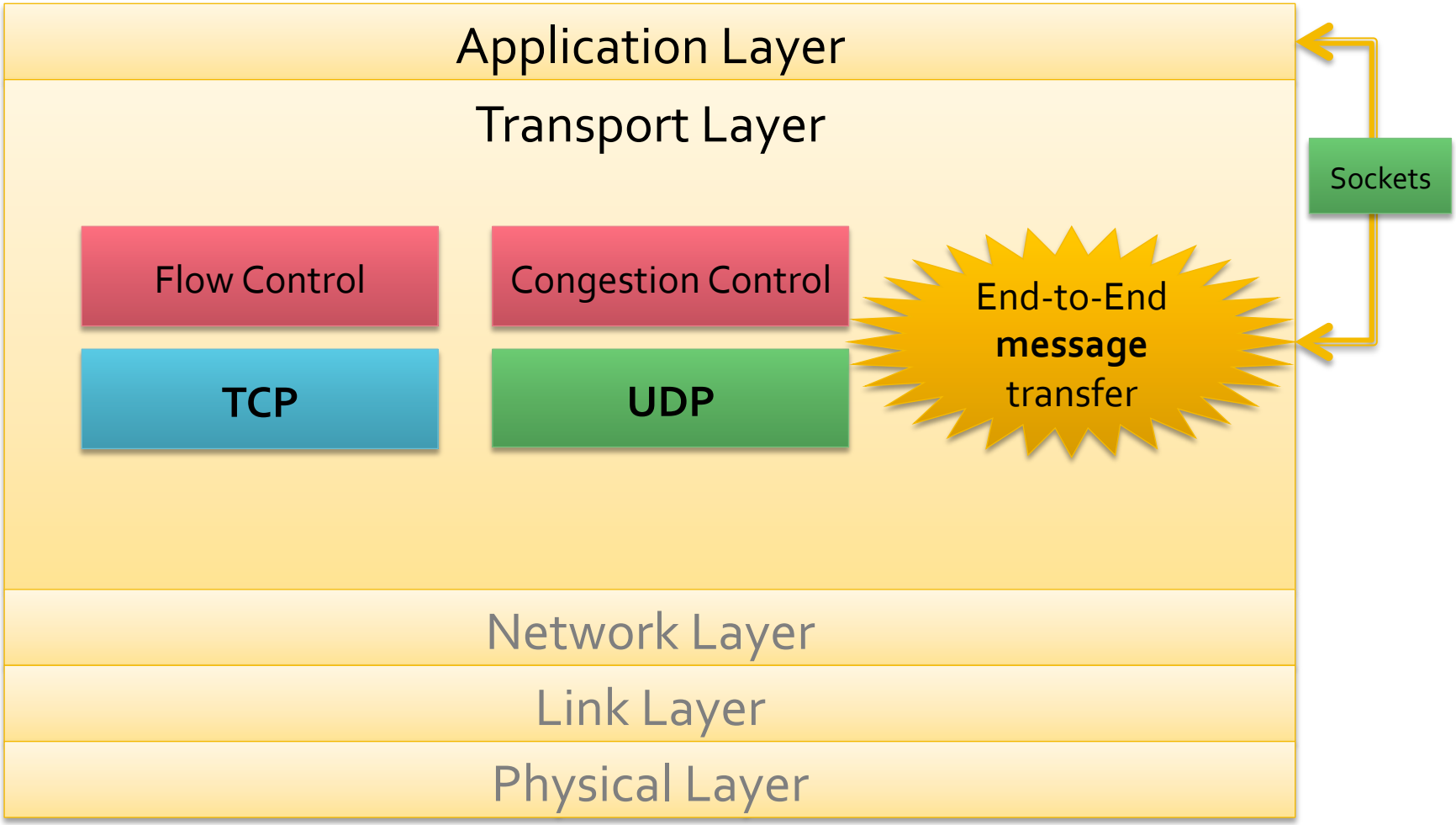
Network Model



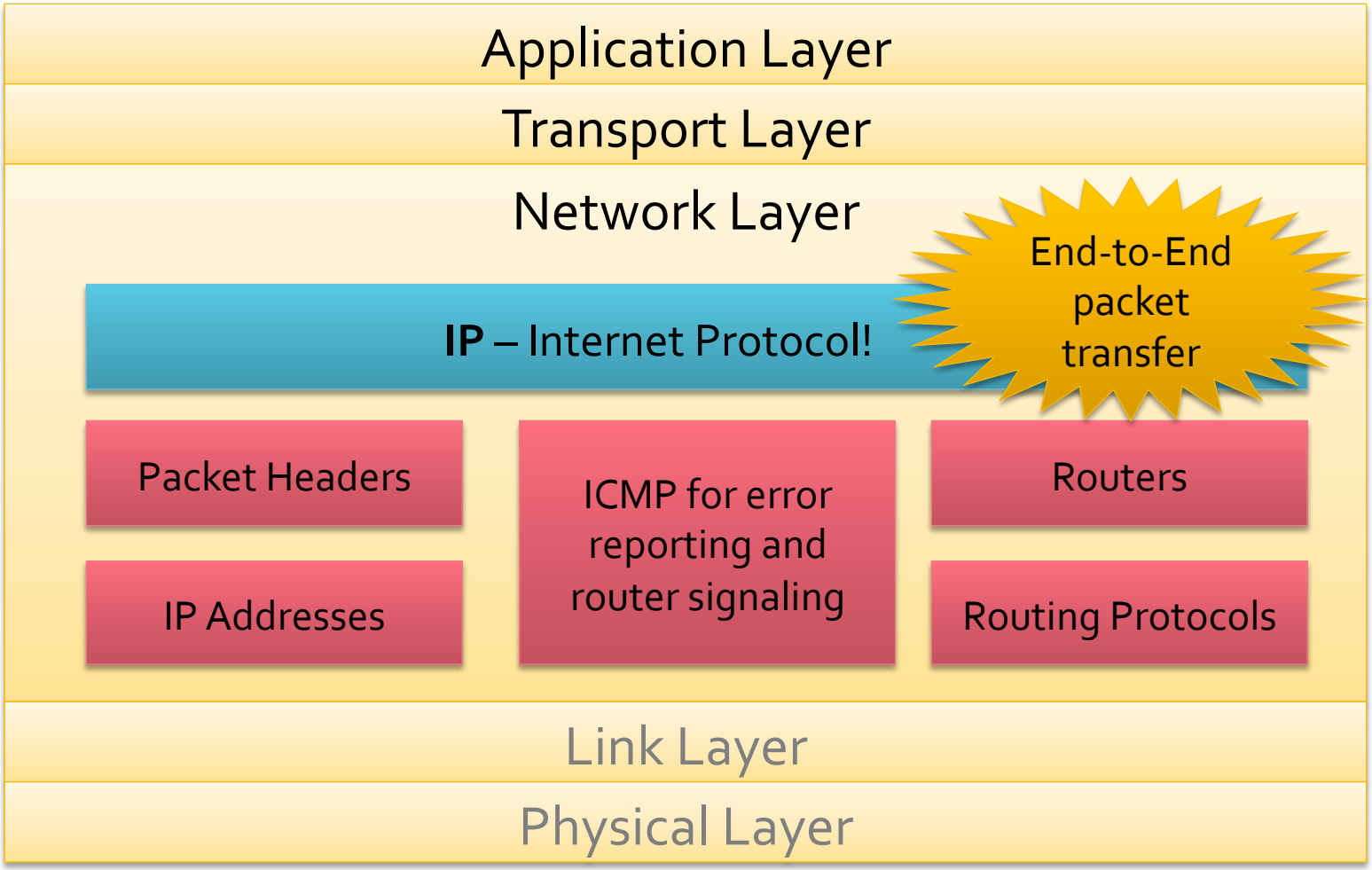
Application Layer



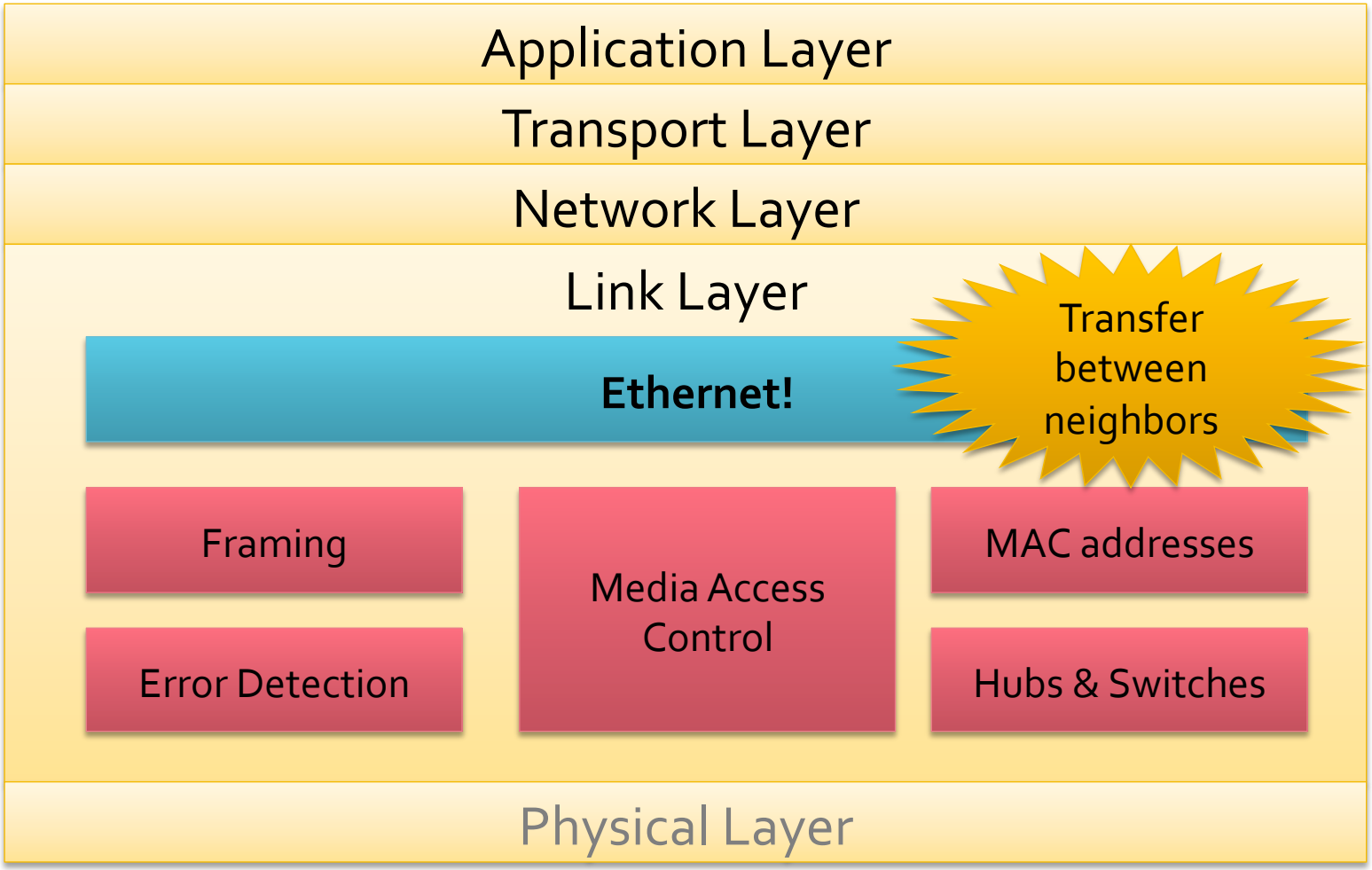
Transport Layer



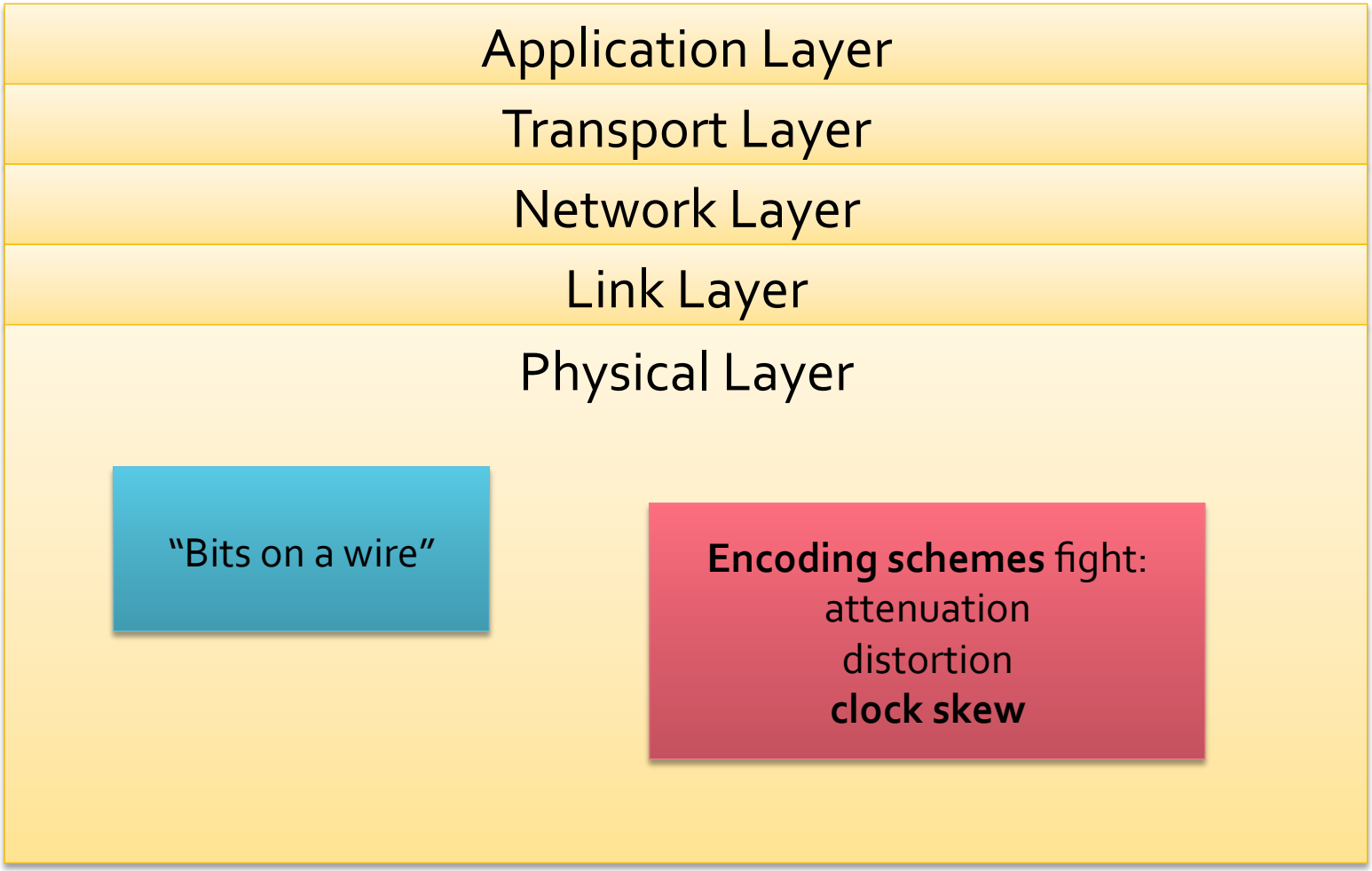
Network Layer



Link Layer


Application Layer
Transport Layer
Network Layer
Link Layer
Ethernet!
Transfer between neighbors
Framing
Error Detection
Media Access Control
MAC addresses
Hubs & Switches
Physical Layer

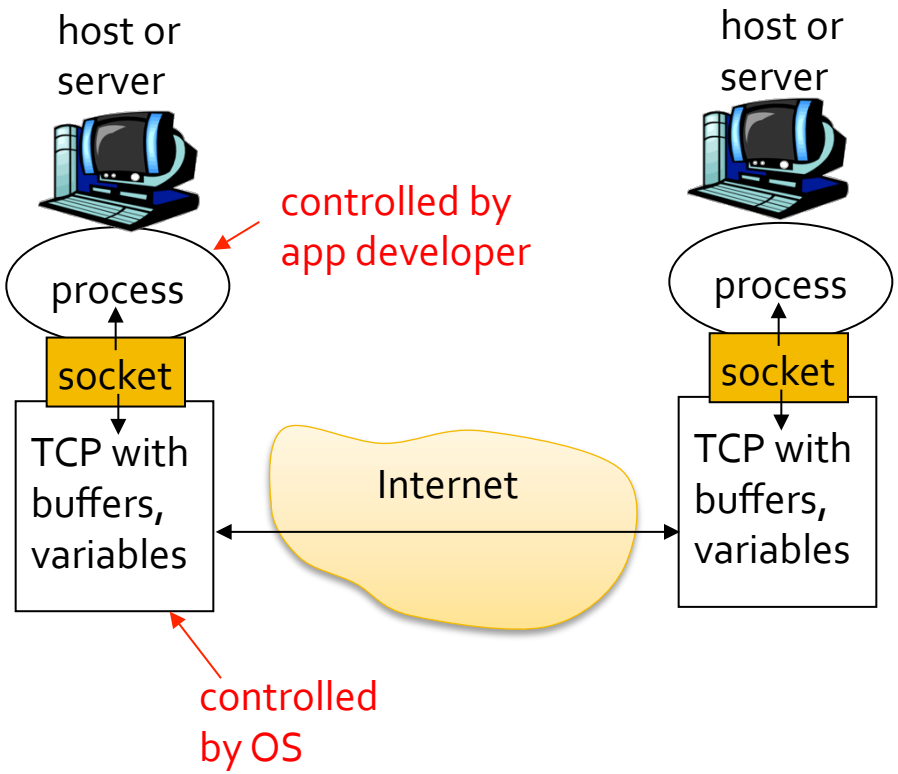
Physical Layer



Application Layer

Sockets

- **What is a Socket?**
- API between application and OS that allows for for network communication



What is a Socket?

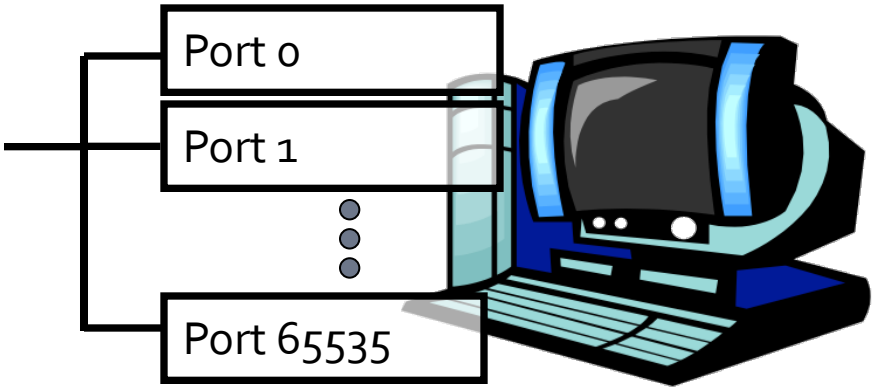
- An interface between process (application) and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - Reliable vs. best effort
    - Connection-oriented vs. connectionless
- Once configured the application can
  - Pass data to the socket for network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

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 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**



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

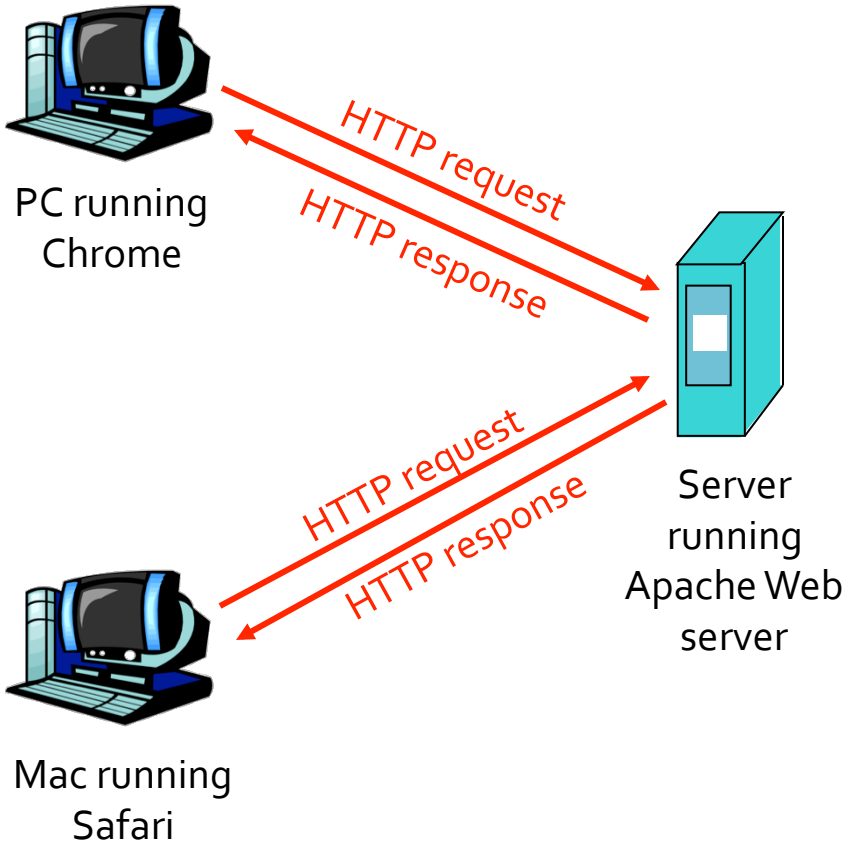
- Sockets just allow us to send raw messages between processes on different hosts
  - Transport service takes care of moving the data
- **What** exactly is sent is up to the application
  - An application-layer protocol
  - Examples: HTTP, IMAP, Skype, Pizza Consumers Union, etc…

Application-Layer Protocol

- **What do the client and server (or peers) speaking the protocol have to 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

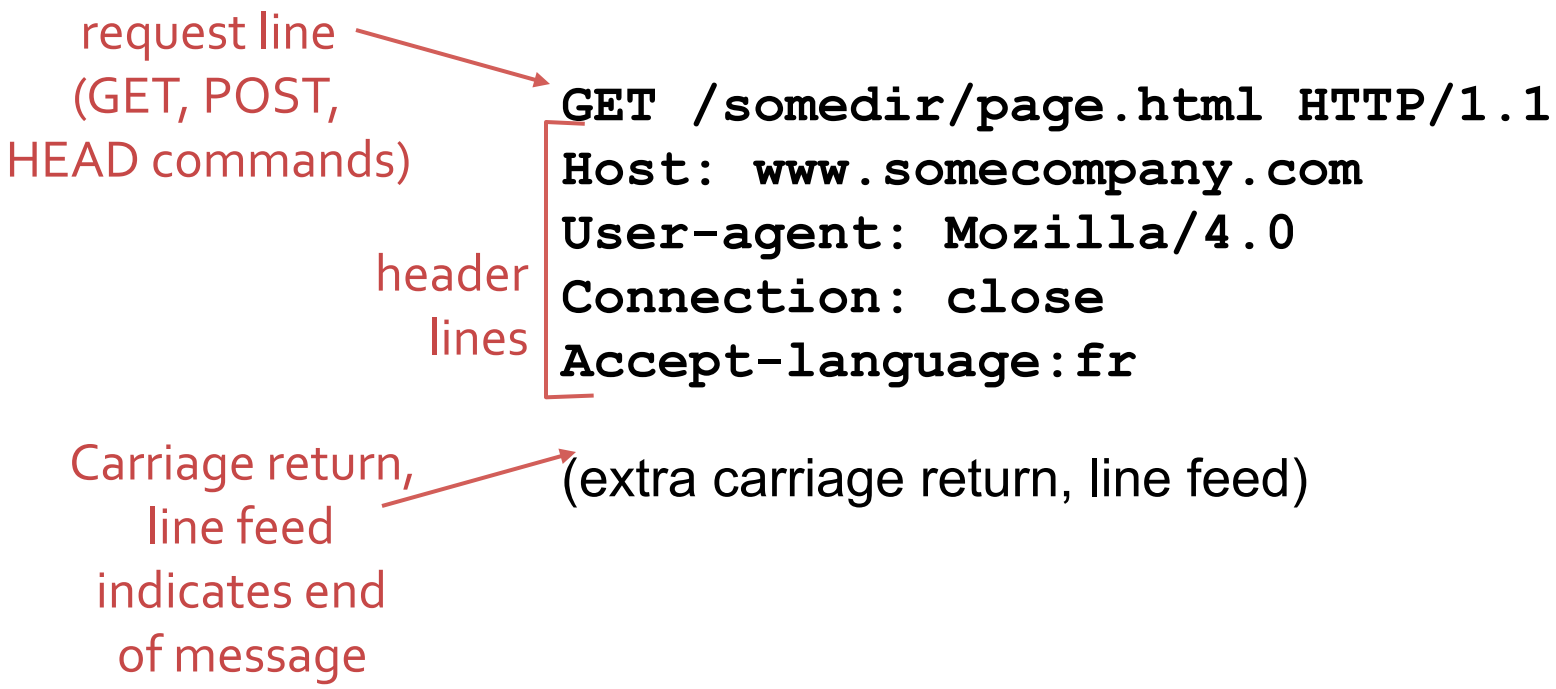
Hypertext Transfer Protocol Overview

- **HTTP** is the *application layer protocol* for the web
- Client/server model



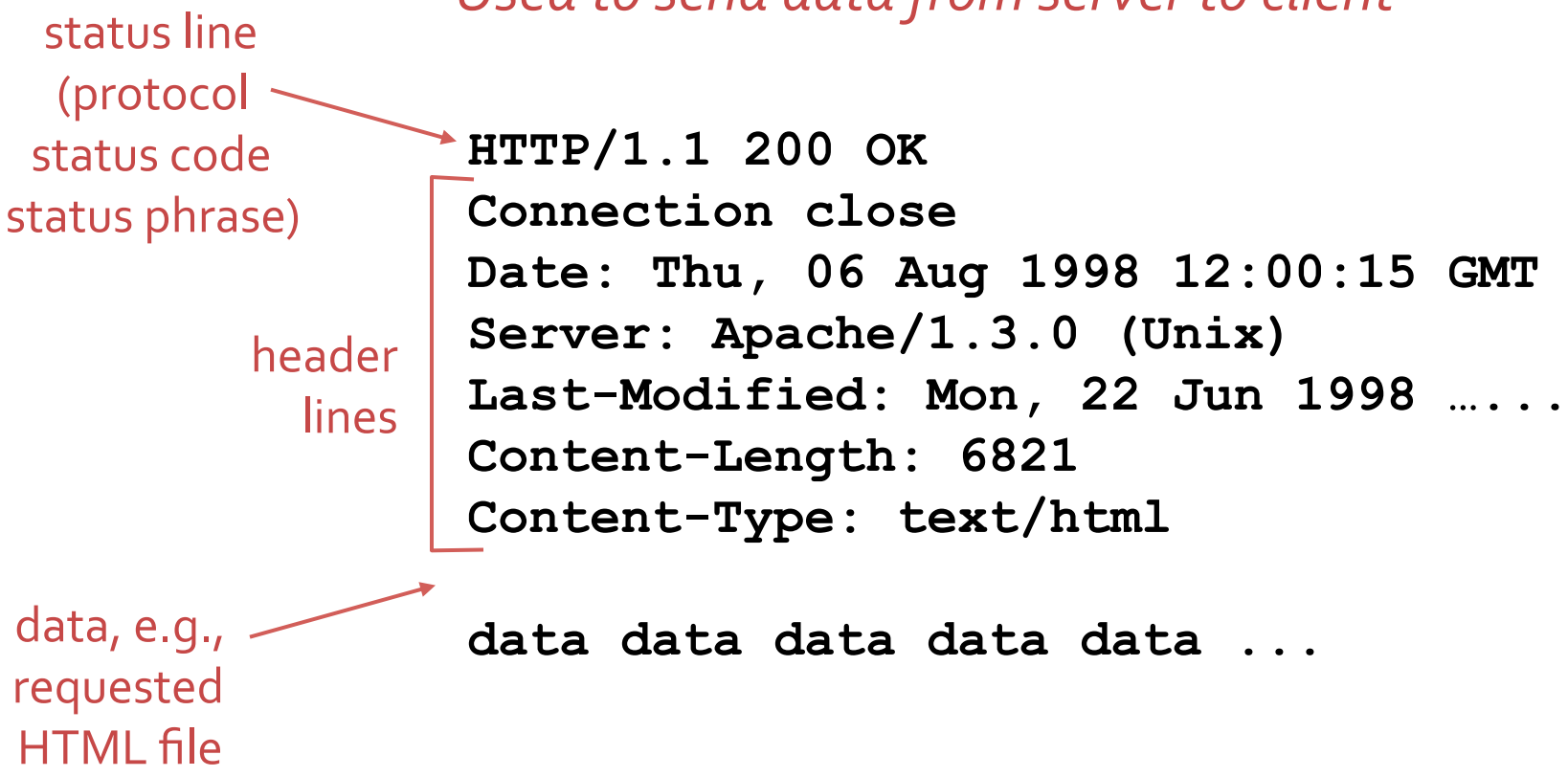
HTTP Request Message

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



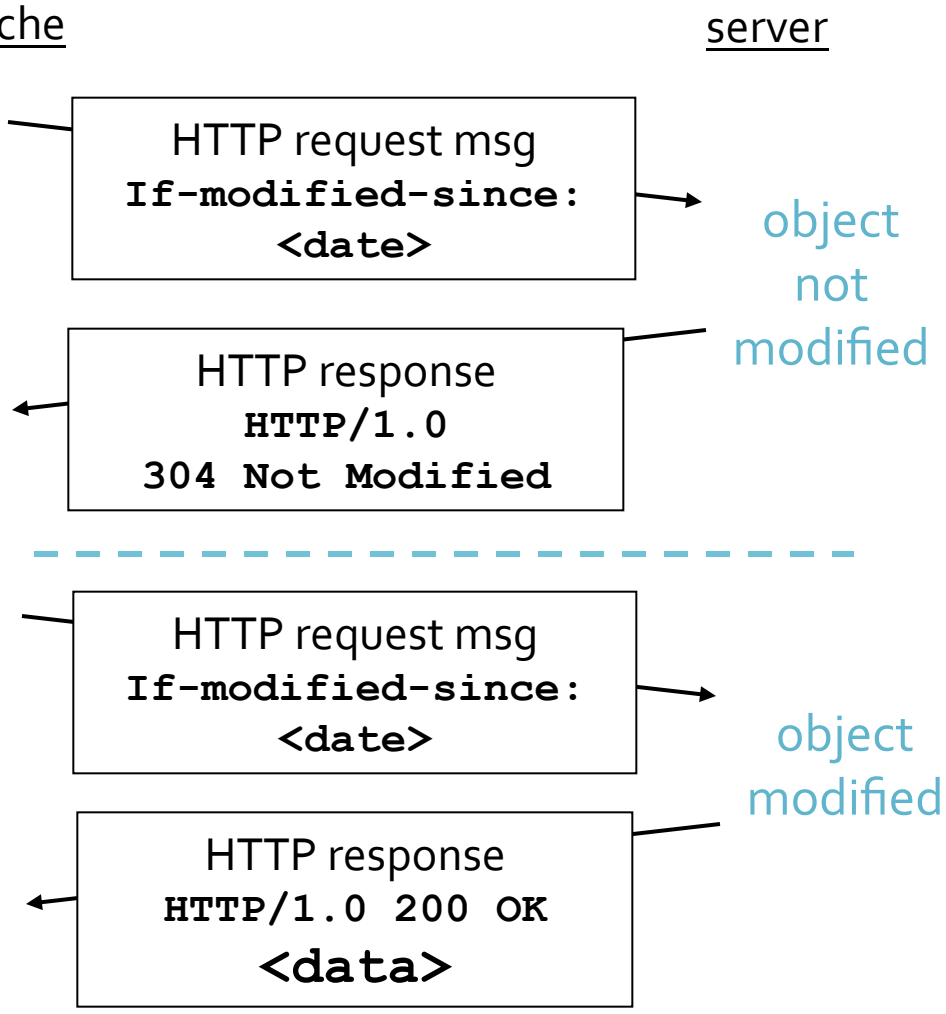
HTTP Response Message

Used to send data from server to client



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`



Application Models

- What is the difference between the client/server model and the P2P model?
- Why advantages does the P2P model have? Disadvantages?

Advantages of P2P

GREATER RESOURCES

- Typically client-server relationship has many clients and 1 server
  - Server can be overwhelmed!
  - As more clients join, each gets fewer resources
- Idea: Use the client's network bandwidth / CPU / disk to assist
  - As more clients join, more resources are available

GREATER RELIABILITY

- A single server can be a reliability problem
  - What if it crashes?
  - What if both of my servers crash?
  - What if my entire datacenter (1000's of servers) loses power?)
- Idea: Use clients all over the world to ensure resources are always accessible (somewhere)

Disadvantages of P2P

- **What about drawbacks of the P2P model?**
  - Complexity?
  - Trust? (Security?)

Application Protocols: Email

- **What is SMTP?**
  - Simple Mail Transport Protocol
  - Used to transfer message to server for storage/ delivery
- **What is POP?**
  - Post Office Protocol
  - Enables clients to download copies of email from server
- **What is IMAP?**
  - Internet Message Access Protocol
  - **How is it different from POP?**

Internet Message Access Protocol (IMAP)

- Keep all messages in one place: the server
  - Clients might have a temporary *cache* for offline access
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - Names of folders and mappings between message IDs and folder name
- Other features
  - Server-side searches (don't have to download mailbox!)
  - Multiple concurrent clients

Network Addressing

- **How does a host obtain its IP address?**
  - DHCP – Dynamic Host Configuration Protocol
- DHCP is an application
  - But it is interested in IP address information
  - That is part of the network layer! (two layers down!)

Dynamic Host Configuration Protocol (DHCP)

- Goals of DHCP
  - Plug and play!
    (Can't trust grandma to set her IP address, netmask, and default gateway correctly…)
  - Allow host to *dynamically* obtain its IP address from network server when it joins network
  - Allow host to renew its lease on in-use address
  - Allow reuse of addresses (if you disconnect your host, someone else can use that address)

DNS

- **What is the goal of DNS?**
- **What are the root name servers used for?**
  - Store the IP addresses of nameservers responsible for the top-level domains (.com, .org, .edu, etc…)
  - Serve as a pointer deeper into the DNS hierarchy
- **What are the local name servers used for?**
  - A cache!

DNS: Root name servers

13 root name "servers" worldwide!

(Each "server" is really a cluster)

(Some clusters are geographically distributed)

Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy
- **You typically know this server's IP address from DHCP**

DNS Name Resolution

- Two types
- **Recursive**
  - The server you contact provides the final answer
  - *Behind the scenes, it may make several consecutive requests*
- **Iterative**
  - The server you contact directs you to a different server to get (closer to) the final answer

DNS and UDP

- DNS uses UDP by default
  - It *can* use TCP, but it's rare
  - **Isn't this unreliable?**
- Why use UDP
  - Faster (in three ways!)
    - No need to establish a connection (RTT/latency overhead)
    - Lower per-packet byte overhead in UDP header
    - Less packet processing by hosts
  - Reliability not needed
    - DNS will just re-request if no response received (2-5 seconds)

Transport Layer

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
  - 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?

TCP Details

- How many packets does it take for TCP to do an initial handshake? (i.e. open connection)

TCP Connection Management

- TCP sender and receiver establish "connection" before exchanging data segments
  - Client initiates connection
    - Calls connect() to an IP/port
  - Server is contacted by client
    - Calls accept()
- TCP variables initialized while establishing connection
  - Sequence #s
  - Buffers and flow control info (e.g. RcvWindow)

- Three way handshake:
- **Step 1:** client host sends TCP **SYN** segment to server
  - Specifies initial seq #
  - No data
- **Step 2:** server host receives SYN, replies with **SYNACK** segment
  - Server allocates buffers
  - Specifies server initial seq. #
- **Step 3:** client receives SYNACK, replies with **ACK** segment, which may contain data

TCP Details

- How does TCP provide for reliable delivery?
- How does TCP provide for in-order delivery?

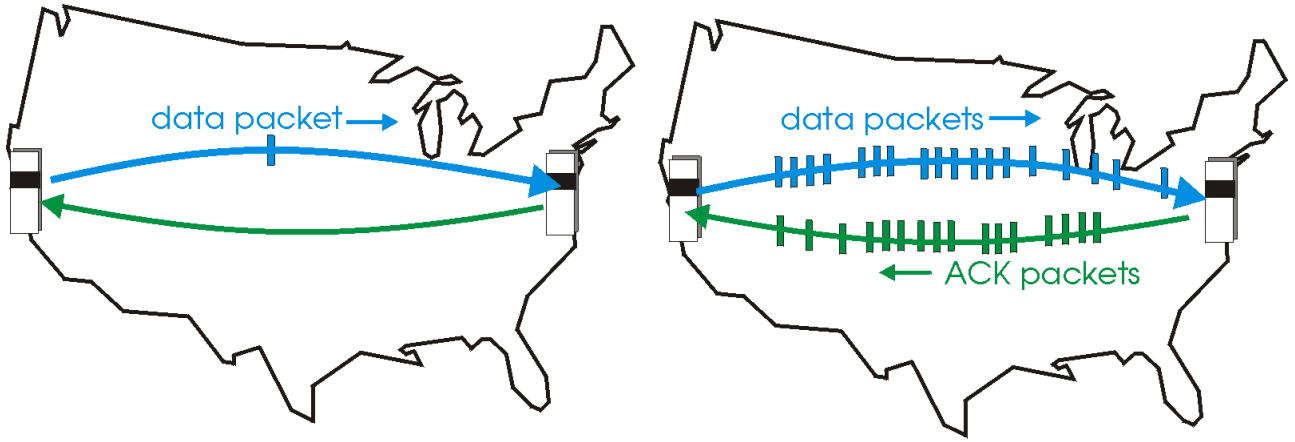
TCP Reliability

- Acknowledgements indicate delivery of data
- Checksums are used to detect corrupted data
- Sequence numbers detect missing, or mis-sequenced data
- Corrupted data is retransmitted after a timeout
- Mis-sequenced data is re-sequenced by the receiver

Pipelined Protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets

- Range of sequence numbers must be increased
- Buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

TCP Round Trip Time and Timeout

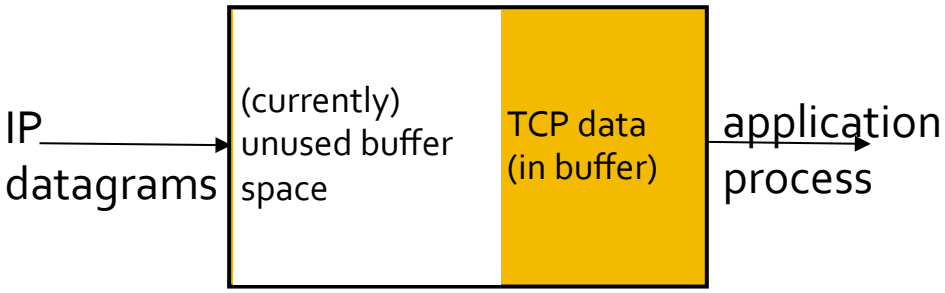
- **How to set TCP timeout value?**
- Should be longer than RTT (round-trip-time)
  - But RTT varies…
- If it is too short
  - Premature timeout
  - Unnecessary retransmissions…
- If it is too long
  - Slow reaction to segment loss

TCP Details

- What is flow control, and why is it important?
- What is congestion control, and why is it important?

TCP Flow Control

- Receive side of TCP connection has a receive buffer:



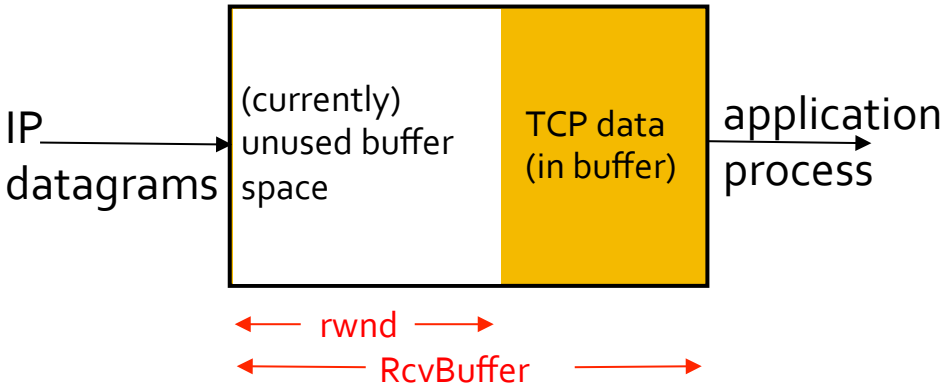
- Application process may be slow at reading from buffer
  - What if buffer fills up?

Flow Control:

Prevents **sender** from **overflowing receiver's buffer** by transmitting too much, too fast

Speed matching service: matching send rate to receiving application's drain rate

TCP Flow Control: How it Works



- Suppose TCP receiver discards out-of-order segments…
- Unused buffer space = rwnd

  = RcvBuffer-[LastByteRcvd - LastByteRead]

- **Receiver notifies sender of unused buffer space**
  - Segment header includes the `rwnd` value
- Sender limits # of unACKed bytes to `rwnd`
  - Guarantees receiver's buffer doesn't overflow

Principles of Congestion Control

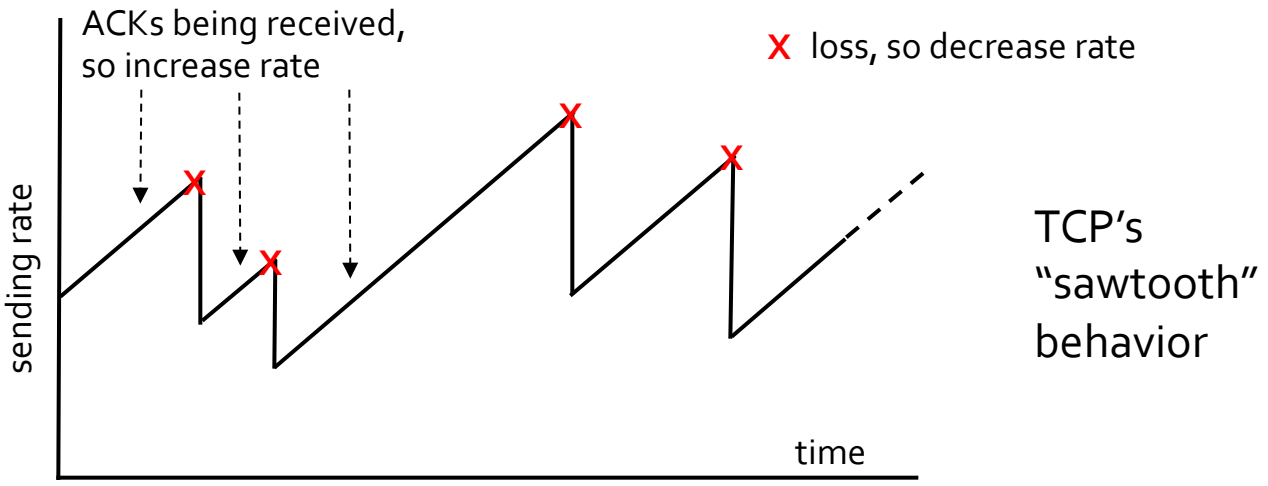
- What is congestion?
  - Informally: "too many sources sending too much data too fast for **network** to handle"
- Different from flow control!
- Manifestations
  - Lost packets (buffer overflow at routers)
  - Long delays (queueing in router buffers)

TCP Congestion Control

- Goal: TCP sender should transmit **as fast as possible**, but without congesting network
- **How do we find the rate just below congestion level?**
  - Decentralized approach – each TCP sender sets its own rate, based on *implicit* feedback:
  - ACK indicates segment received (a good thing!)
    - Network not congested, so increase sending rate
  - Lost segment – assume loss is due to congested network, so decrease sending rate
    - No device tells us congestion is occurring, we just guess!

TCP Congestion Control: Bandwidth Probing

- Probing for bandwidth
  - Increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate



Transport Protocols

- Are TCP and UDP the only possible transport layer protocols we could use?
- **What is SCTP?**
  - Stream Control Transmission Protocol
  - **What does it do differently than TCP and UDP?**

Lower Layers (Briefly!)

Host Configuration

- My computer has several key network settings:
  - My Ethernet / **MAC address**
  - My **IP address**
  - **Netmask** of network I'm connected to
  - **Next-hop gateway** IP address of network I'm connected to
- **What do these mean?**

Ethernet Switch

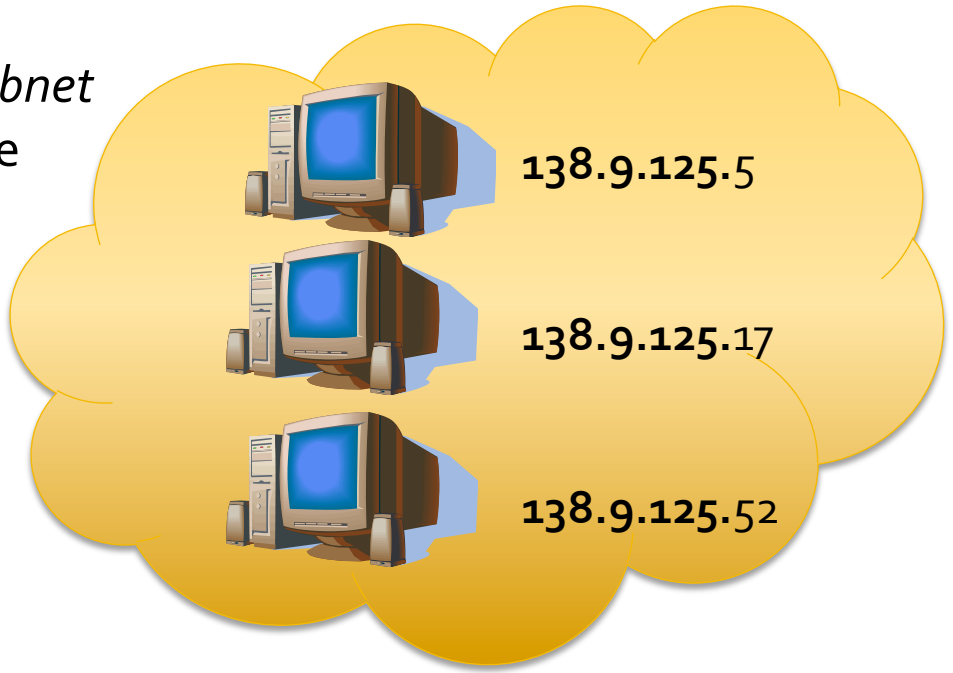
- **Learns location** of computers on Ethernet network
  - Examine header of each arriving frame
  - What is its source MAC address? (i.e. who sent it?)
    - Note the port it came in on!
    - Save this data in **forwarding table**
- Forwards data out correct port
  - Search forwarding table for **destination** MAC address

Subnet

- A small network that is part of a larger network
- A collection of computers *(probably in the same physical area)* that have similar IP addresses

All computers in this *subnet* have IP addresses of the form 138.9.125.x

Note: There is no rule that says subnet addresses have to be at 8-bit boundaries!



The Internet Protocol - Motivations

- Ethernet is sufficient for a local-area network
- IP is needed for a global network (the **Internet**!)

Routers

- "Similar" to switches, but only at a high level
  - Packet comes in
  - Switch/router looks up the destination address
  - Packet forwarded out correct port
- Key difference #1: Routers forward based on IP addresses!
  - Router works at *network* layer, switch works at *link* layer

Address Resolution Protocol (ARP)

- Find link layer address given a network layer address
  - i.e., what is the **Ethernet address** for a given **IP address**?
- Every IP node (hosts and routers) has an ARP table
  - Mapping from IP to Ethernet addresses on their LAN
  - May be incomplete
  - Can include both static and dynamic entries