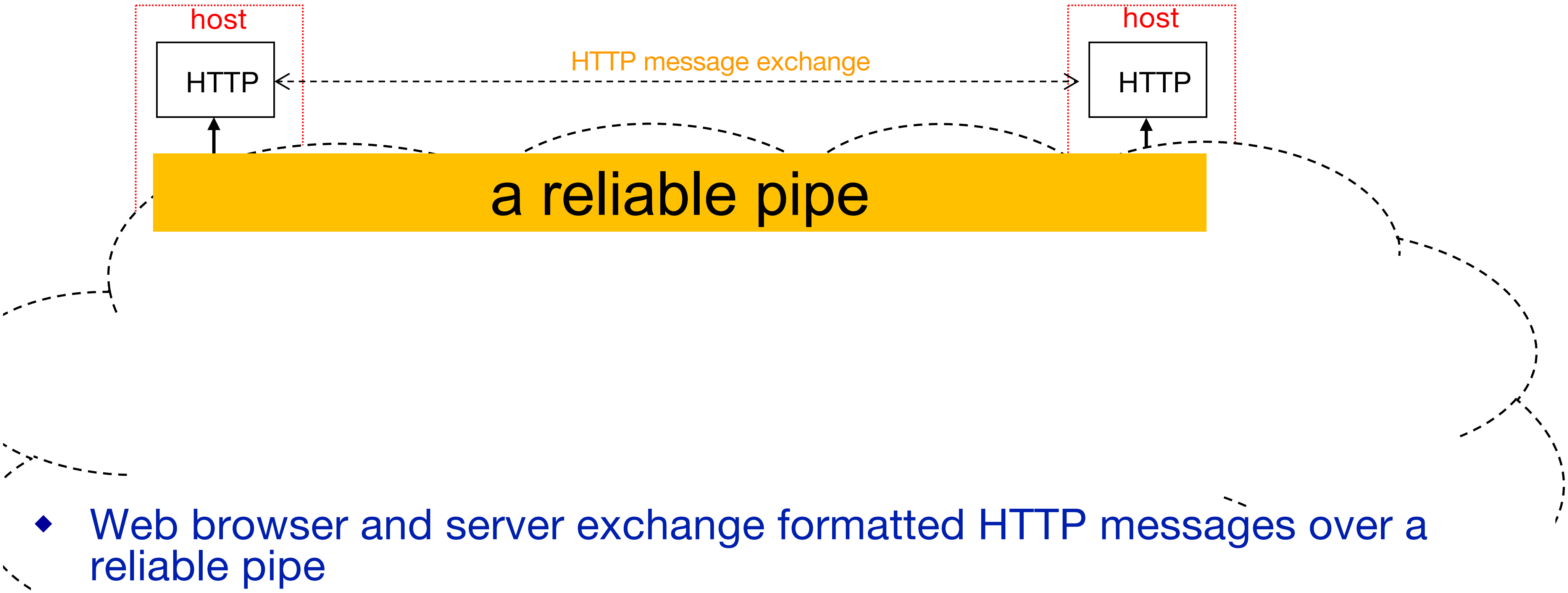
What we covered in lecture-1

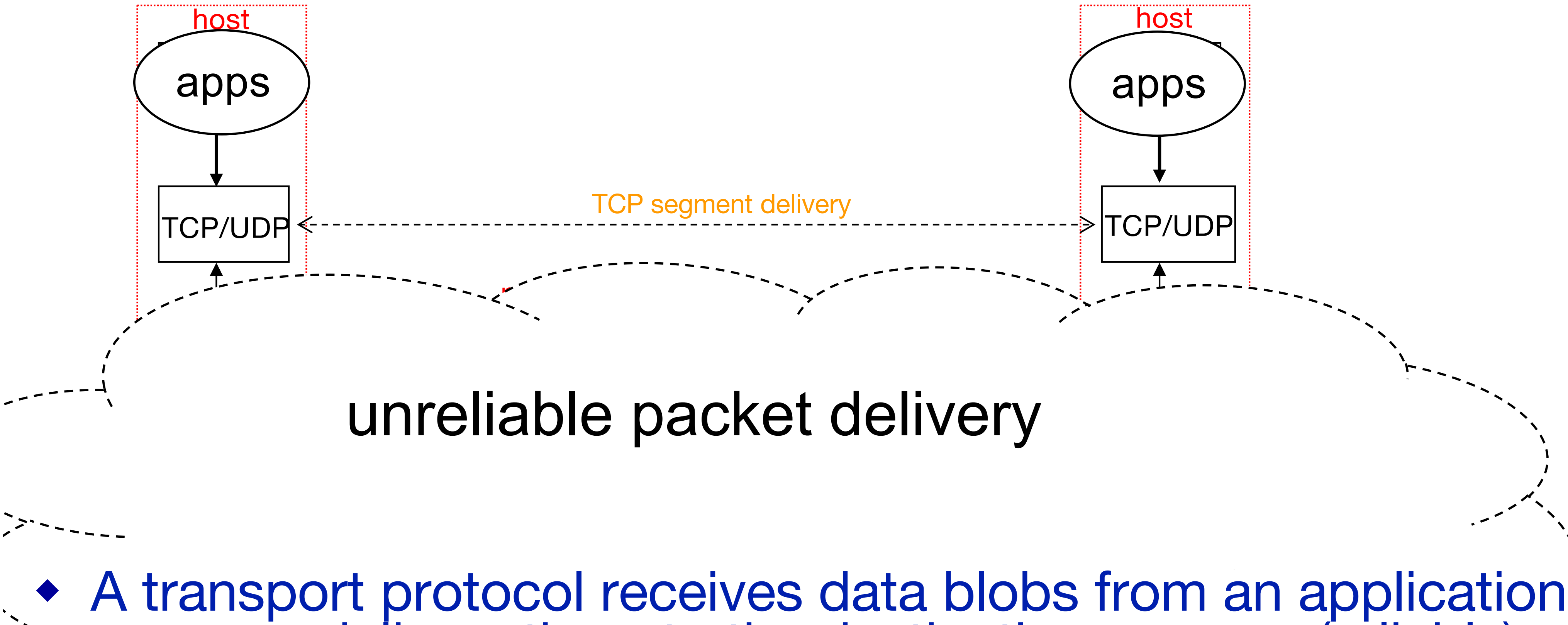
- Concepts:
  - **Internet**: made of a huge number of hosts and routers, interconnected by physical and wireless links
  - **Host**: a computer running applications and bunch of protocols to let apps exchange data with each other
  - **Router**: a packet switch running bunch of protocols to move packets toward their destinations
- Protocols are organized in layers:
  - Application protocols
  - Transport protocols
  - Network protocols
  - Link layer protocols
  - Physical layer
- How to calculate packet delays as they move across one hop

Application protocol's view of the world



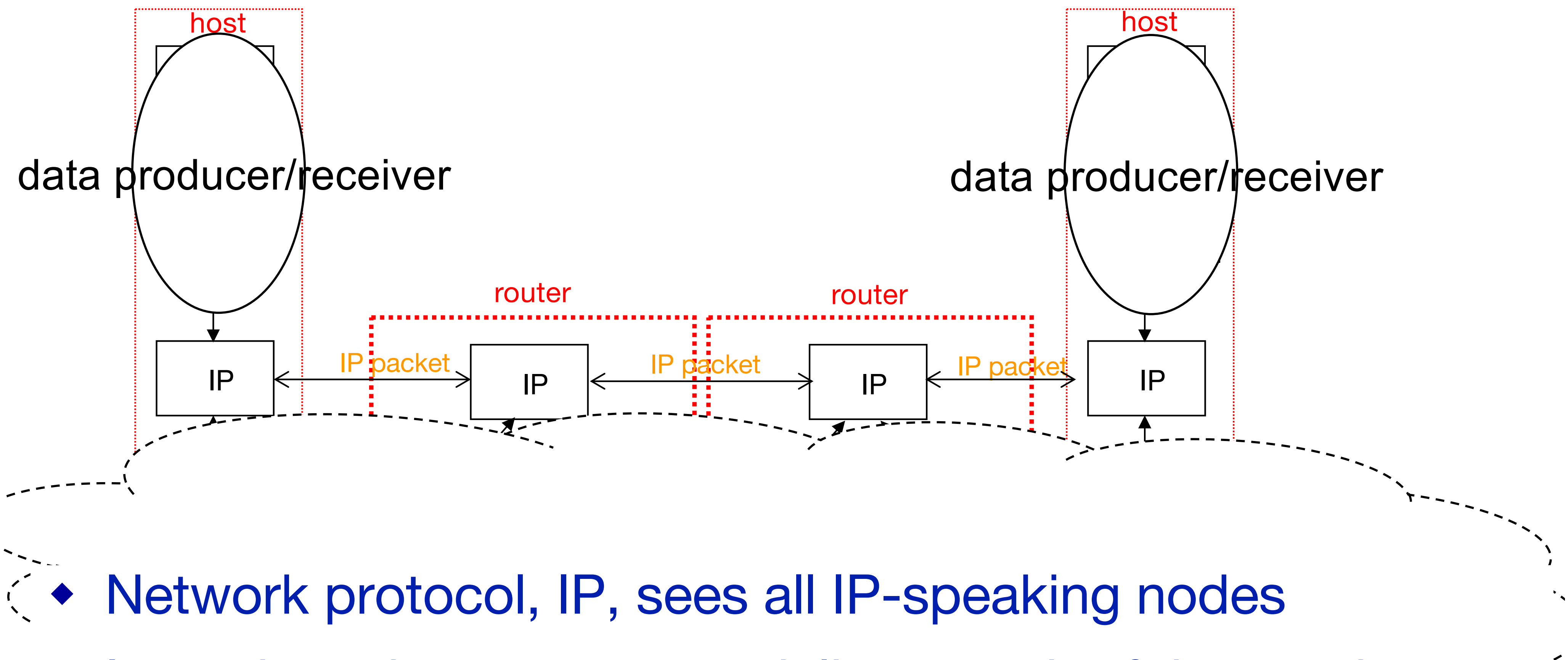
- Web browser and server exchange formatted HTTP messages over a reliable pipe
- As an application protocol, HTTP only concerns with the message's presentation format
- Application decides where msgs should be delivered to
  - The receiving end is identified by its name, which gets translated to IP address

Transport protocol's view of the world



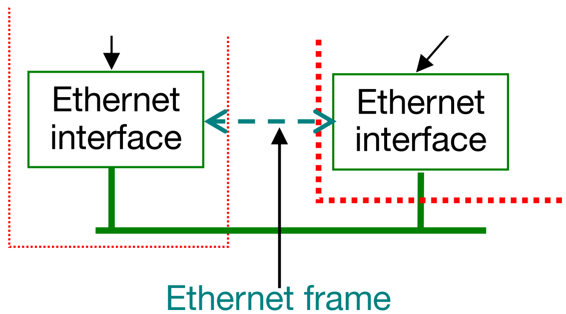
- A transport protocol receives data blobs from an application process, delivers them to the destination process (reliably)
  - Dest. Process is identified by IP address + (trans)port number
- It runs between two processes over an unreliable network (where packets can be garbled, lost, or reordered)

Network protocol's view of the world



- Network protocol, IP, sees all IP-speaking nodes
- It receives data segments, delivers each of them to its destination IP address (with its best effort)
  - A router forwards packets, without looking inside IP envelope

Link layer protocol's view of the world

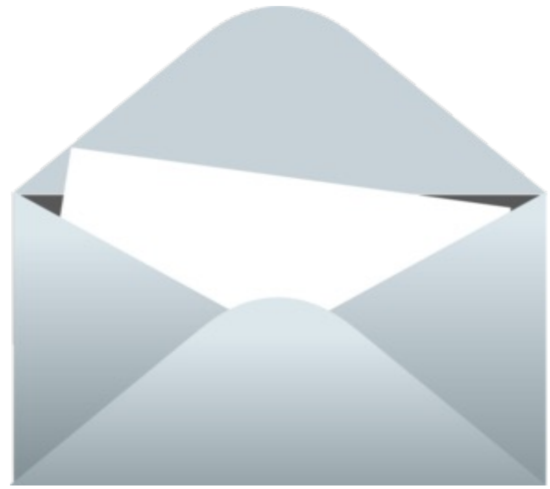


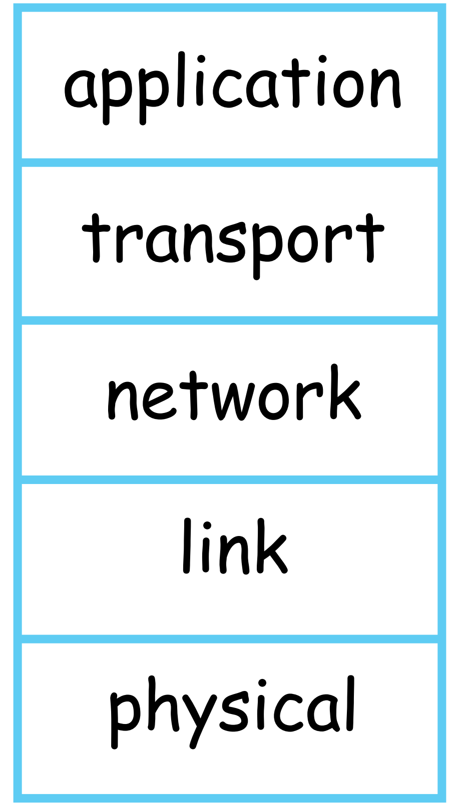
- A link layer protocol delivers data frames between two physically connected nodes
  - A link-layer header is added at sending node, removed by the receiving node
  - When a packet moves through the network across multiple hops: link-layer header is added and removed multiple times

Layered protocol implementation

- protocol header: contains the information one writes on the "envelope"
- all the information, and *only* the information, that's needed to carry out the protocol's functionality

What protocol “layer” really means

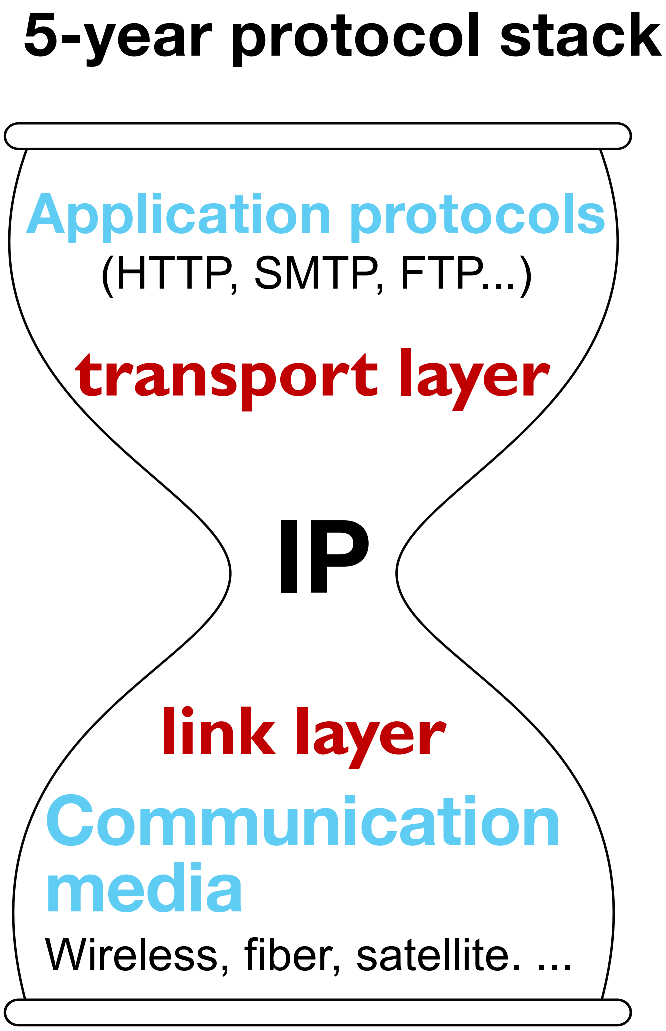




Link layer protocol

One more question: why 5 layers?

- Two layers are taken as given
  - Multiple different **application protocols**
  - Multiple different **physical communication media types**
- **IP**: the span layer
  - Connecting up all nodes
- **Link layer**: adaptation between IP and physical media
- **Transport**: adaptation between what apps want and what IP offers



CS118

Lecture-2: a few basic concepts in networked applications

Transmission vs. propagation delay

Transmission delay: L / R

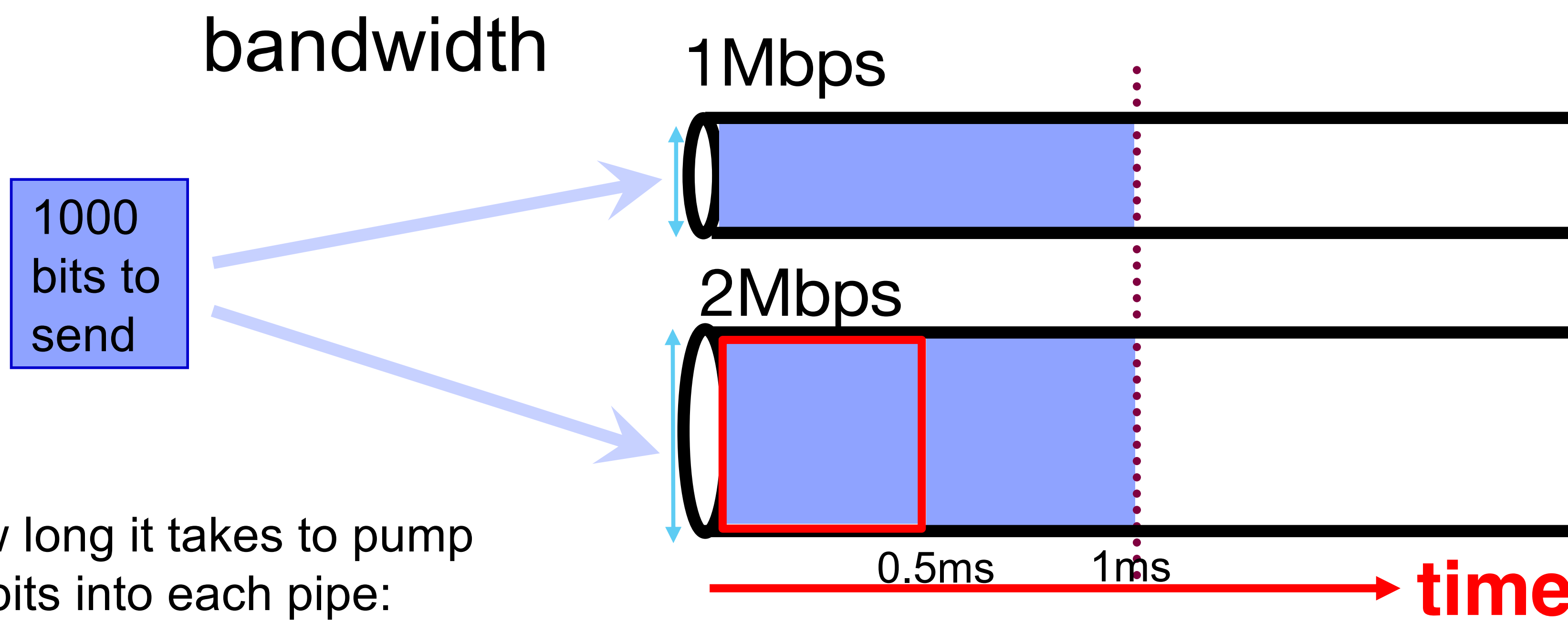
R = link bandwidth (bit-per-second, bps)

L = packet length (bits)

Propagation: d / s

d = length of a physical link

s = signal's propagation speed in the medium (~2×10^8 meter/sec)

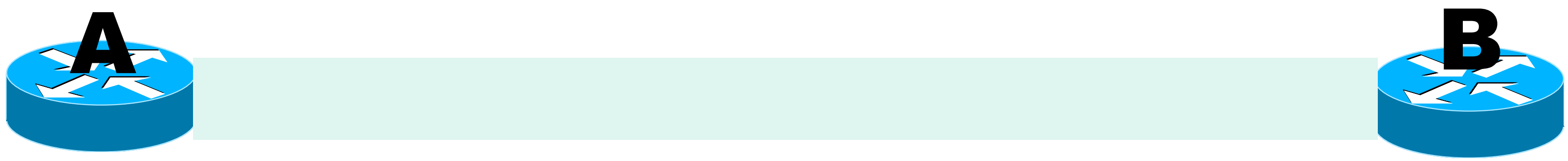


How long it takes to pump 1K bits into each pipe:

(1) $\frac{10^3 bits}{10^6\ bps}$ =1msec

(2) $\frac{10^3 bits}{2\times 10^6\ bps}$ =0.5msec

Where can a packet be?



Bandwidth = W, link length =200 km, Packet size: 10,000 bits (1250 bytes)

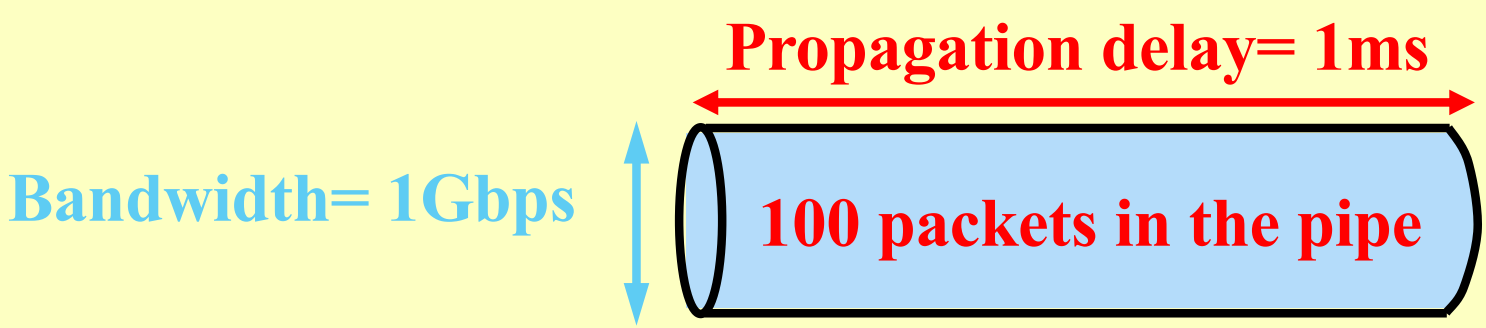
Q: When the first bit of the packet reaches B, where is the last bit?

P = propagation delay = $\frac{200{,}000m}{2 \times 10^8 m/\sec}$ = 1msec

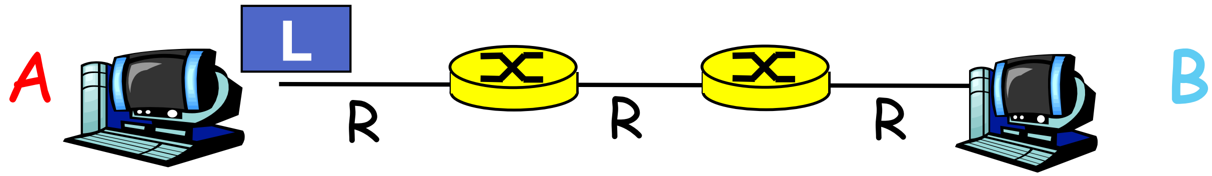
If W=1Mbps: D_{trans}=10,000/10^6 = 0.01sec = 10msec

If W=1Gbps: D_{trans}=10,000/10^9 = 0.01msec

bandwidth × **p-delay =** pipe size (amount of data “in-the-pipe”)



Packet-switching: store-and-forward



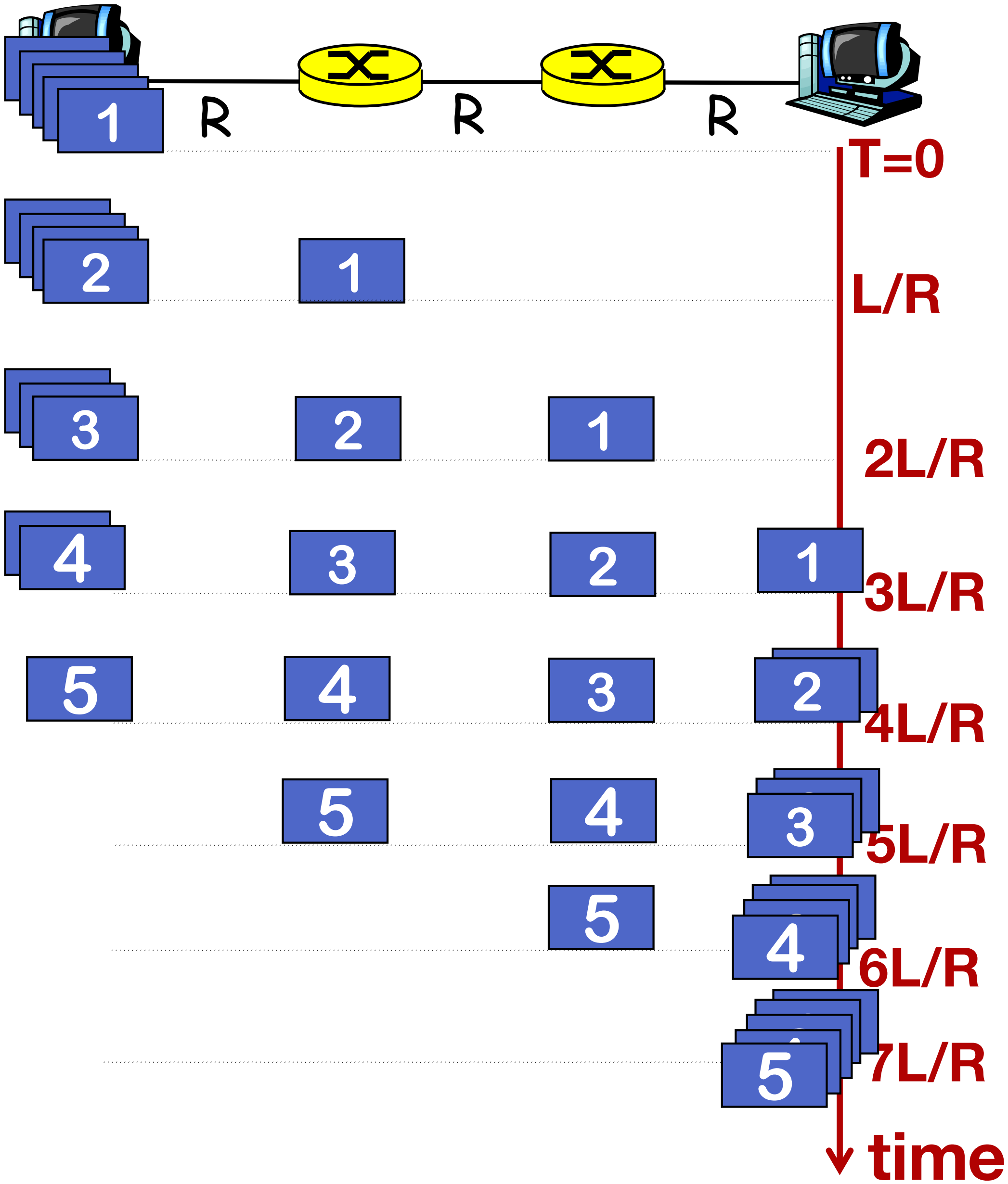
- Takes L/R seconds to transmit (push out) packet of L bits on to link of R bps
- Entire packet must arrive at router before it can be transmitted on next link: *store and forward*

Example 1: send L A → B

- L = 8000 bits (1000 bytes)
- Bandwidth R = 2 Mbps
- *Ignore propagation delay:*

  delay = 3xL/R = 12 msec

Packet-switching: store-and-forward



Example 2:

- A sends 5 packets to B
- L = 8000 bits, R = 2 Mbps
  - *Ignore propagation delay*
- How long does it take starting from A sending the first bit of first packet till B receives the last bit of the last packet?

What if one takes into account the propagation delay?

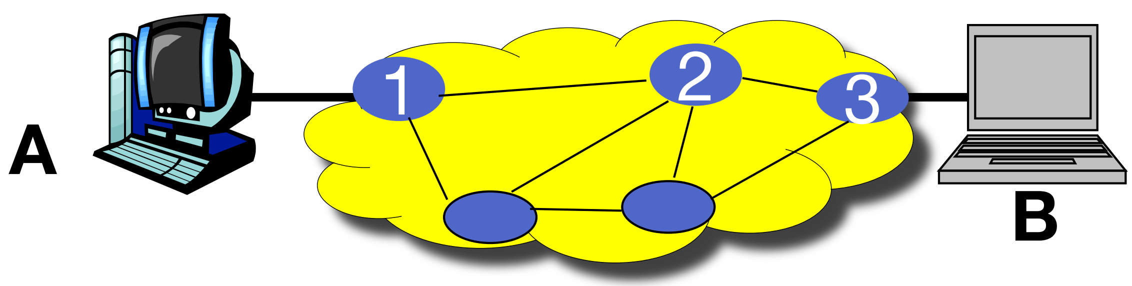
Network latency

- The time to send 1 packet from host A to B
  - sum of delays across each hop along the path

$$Delay_{A-B} = Delay_{A-1} + Delay_{1-2} + Delay_{2-3} + Delay_{3-B}$$

- **RTT**: round-trip-time

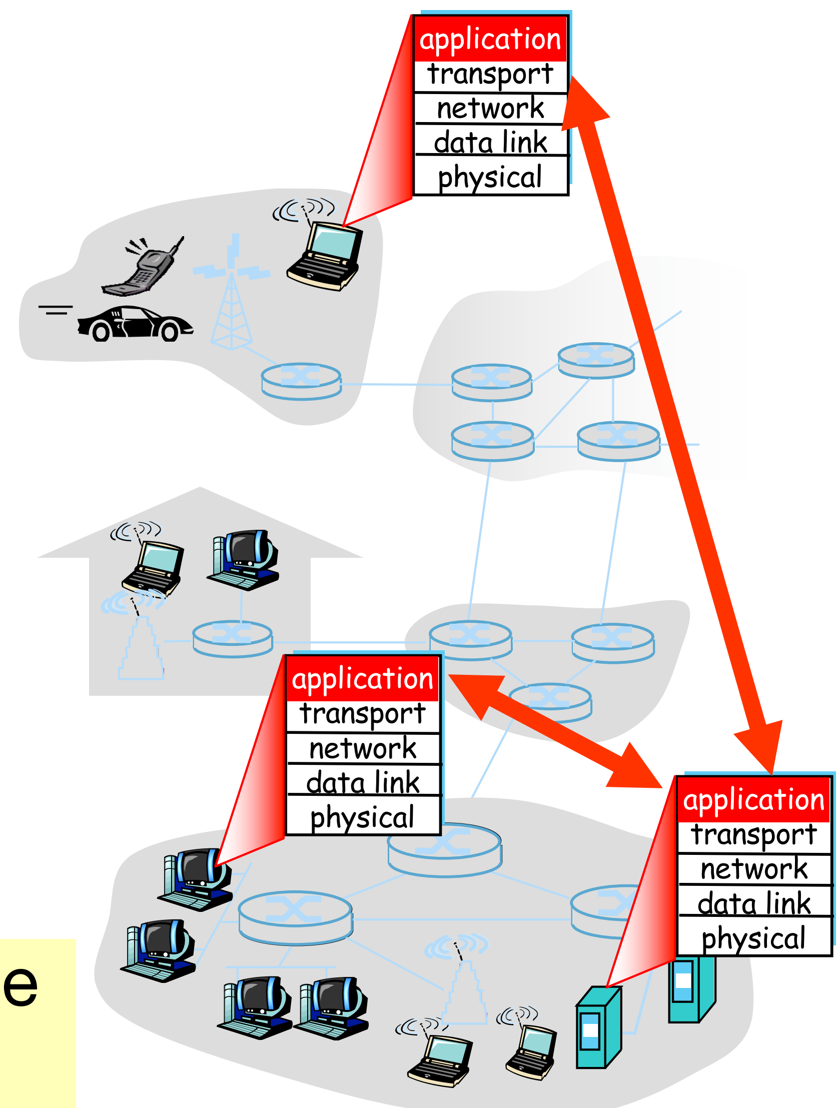
$$RTT_{AB} = Delay_{A-B} + Delay_{B-A}$$



Network applications:
how different parties reach each other

Some popular network applications

- e-mail
- web
- instant messaging
- P2P file sharing
- multi-user network games
- Streaming video (e.g. YouTube)
- voice over IP (e.g. skype)



Application processes communicate with each other using application protocols

Client-server application communication model

servers:

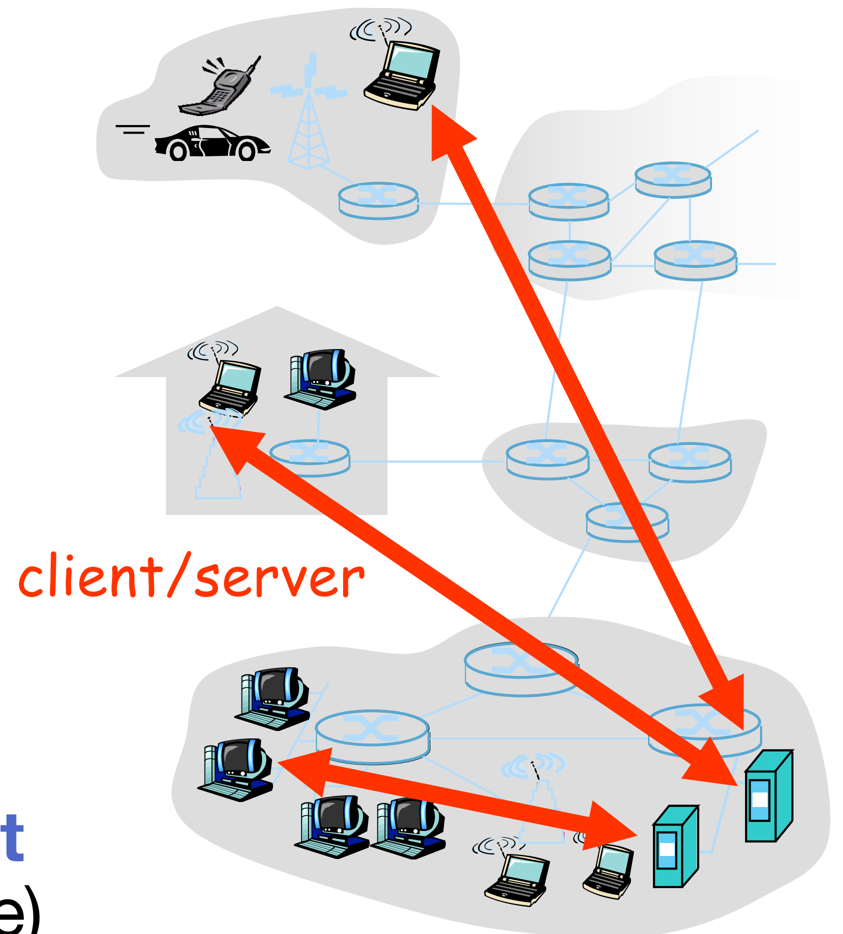
- Reachable by IP address
- **always-on**, waiting for incoming requests from clients

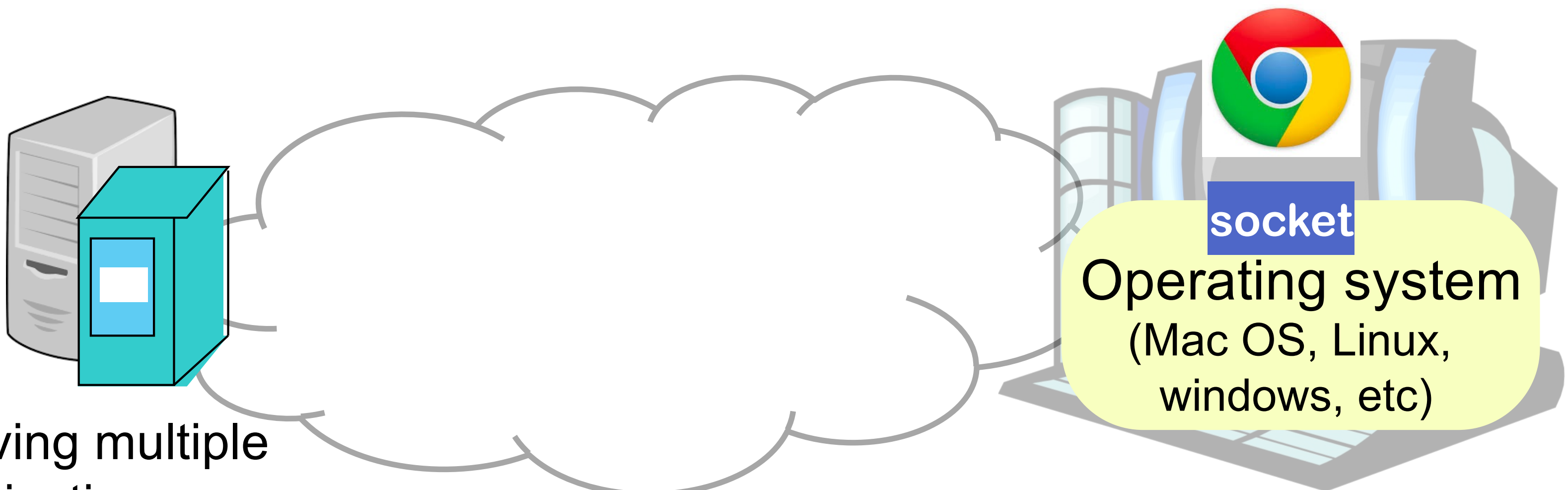
clients:

- Initiate communication with server

Q: How does a client process ***identify*** the server process with which it wants to communicate?

A: Using port numbers via the **socket API** (**A**pplication **P**rogram **I**nterface)

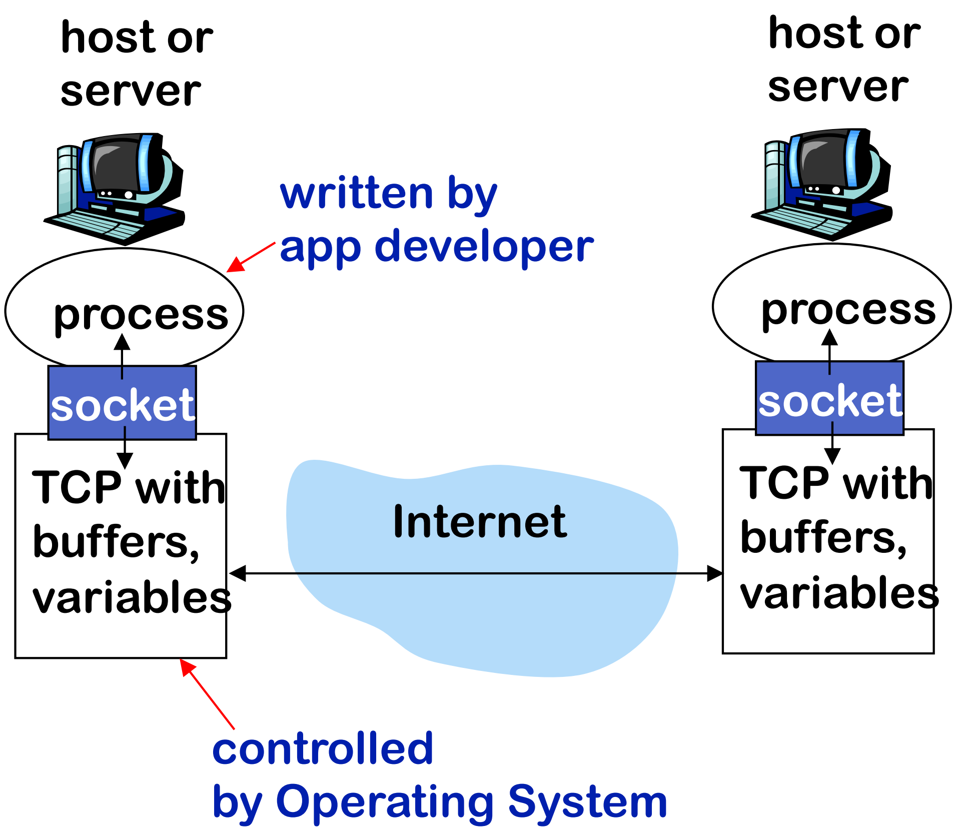




Serving multiple applications

Socket

- Process: program running on a host
- Between different hosts: Processes communicate through an application-layer protocol
- A process sends/ receives messages to/from its socket
- A socket analogous to a door:
  - sending process shoves message out of the door
  - transport protocol brings message up to the socket at receiving process



What is "socket"

- A set of system function calls

socket ( ): Create a socket

bind( ): bind a socket to a local IP address and port #

connect( ): initiating connection to another socket

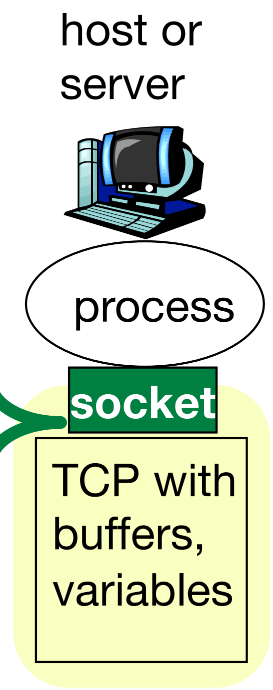
listen( ): passively waiting for connections

accept( ): accept a new connection

Write( ): write data to a socket

Read( ): read data from a socket

Close( )



What is "socket"

socket ( ): Create a socket

bind( ): bind a socket to a local IP address + port #

connect( ): initiating connection to another socke

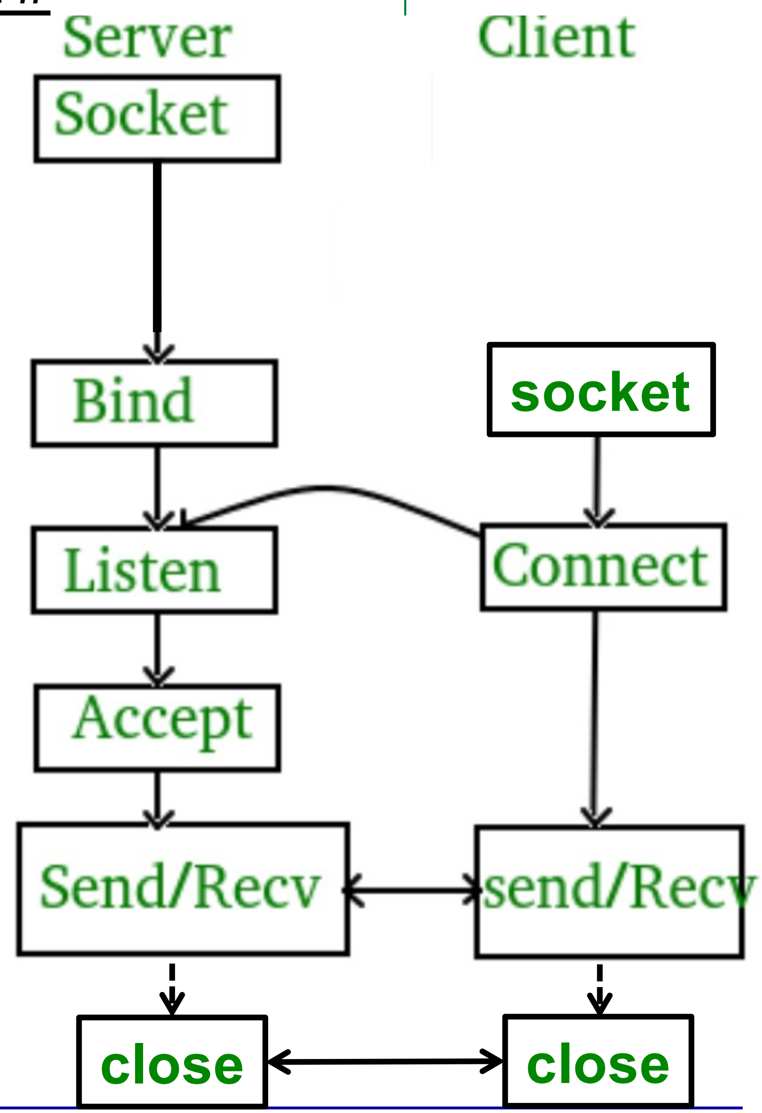
listen( ): passively waiting for connections

accept( ): accept a new connection

Write( ): write data to a socket

Read( ): read data from a socket

Close( )



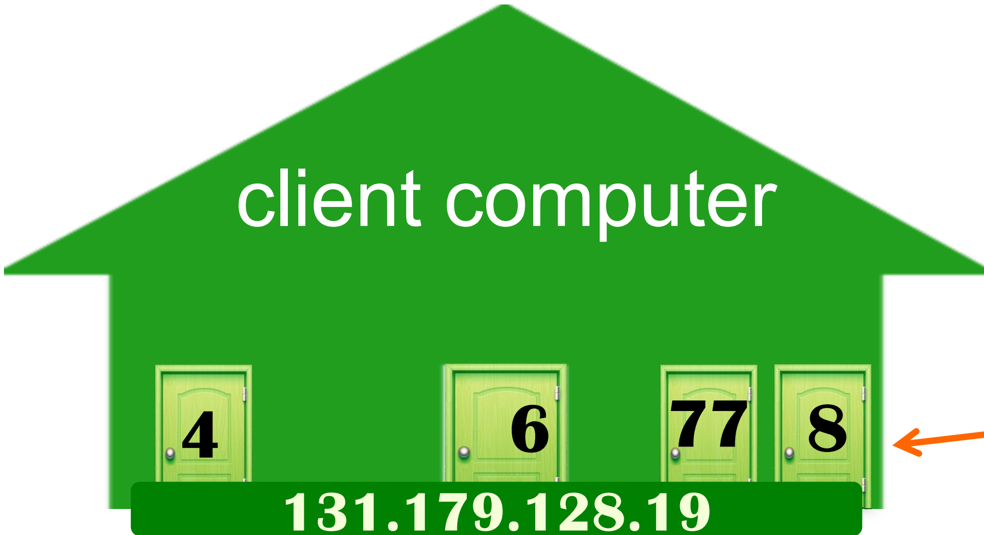
Establishing a socket on the *client* side:

- Create a socket with the socket( ) system call
- Connect the socket to the server using the connect( ) system call
- Send and receive data.
  - There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

Establishing a socket on the *server* side:

- Create a socket with the socket( ) system call
- Bind the socket to [address, port#] using the bind() system call.
- Listen for connections with the listen( ) system call
- Accept a connection with the accept( ) system call.
- Send and receive data

Socket: analogous to a door



socket( ): create a "door"
bind( ): tie the door to a [local IP addr, port#] pair

connect()

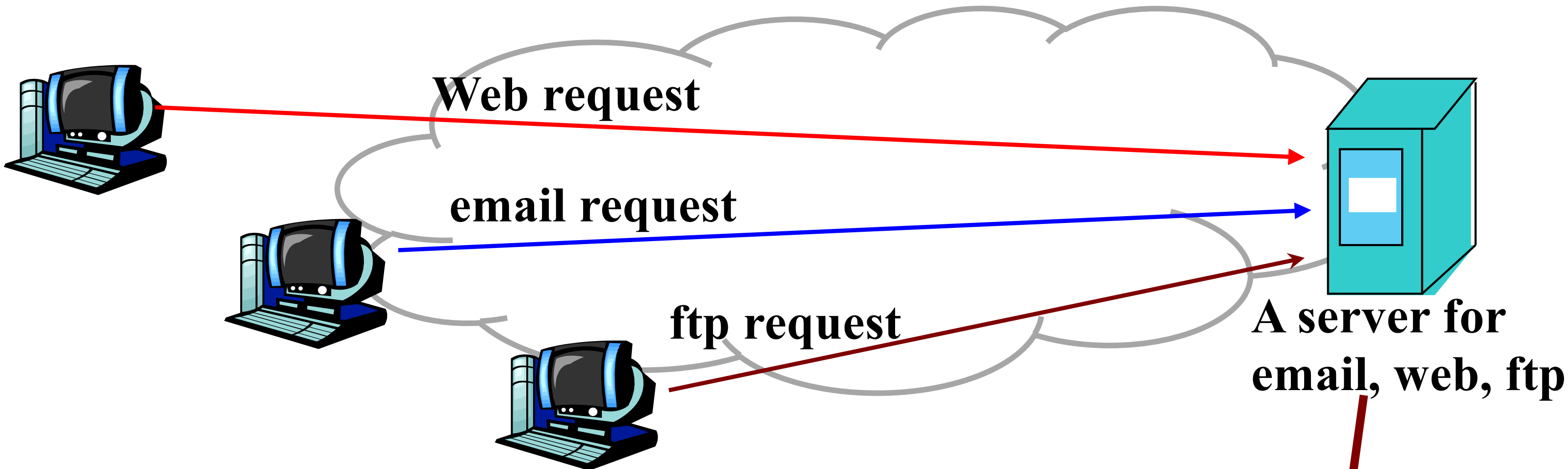
close(): delete/remove a door

A couple other questions:

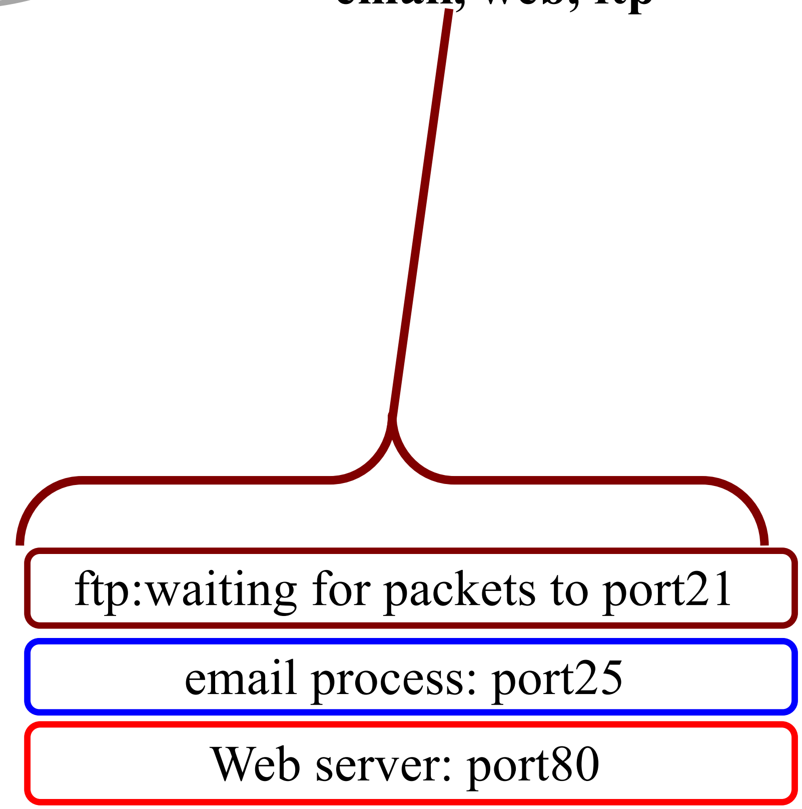
- What is the server's IP address?
  - Website name → IP address
- What port number to use?
  - Client: assigned by OS
  - Server: defined standards

listen( ): start waiting for incoming packet with matching destination port#

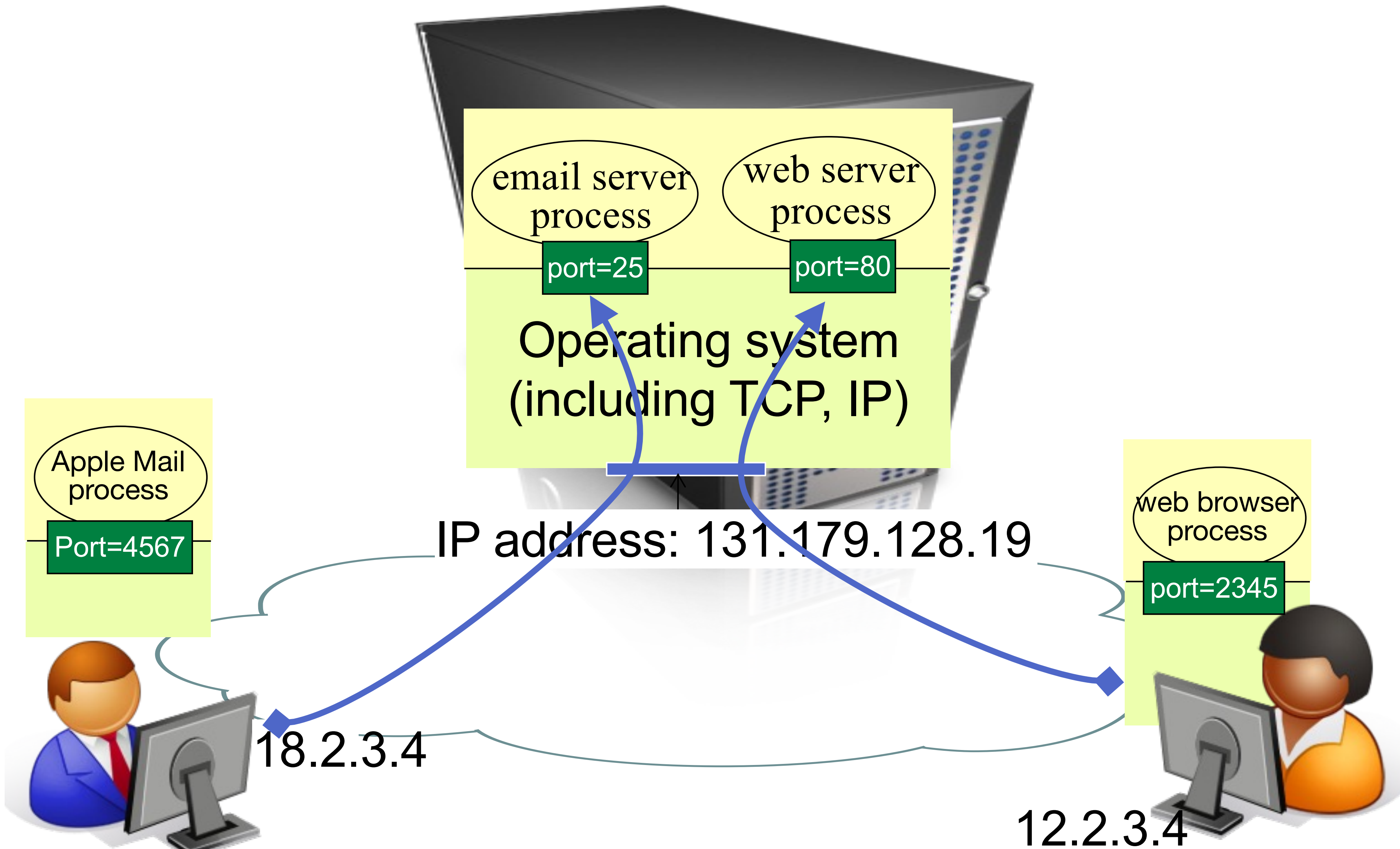
A quick comment about "port"



- Web, email, ftp all use TCP
- How does the server tell who wants what?
  - By port number: web using port 80, ftp 21, mail 25

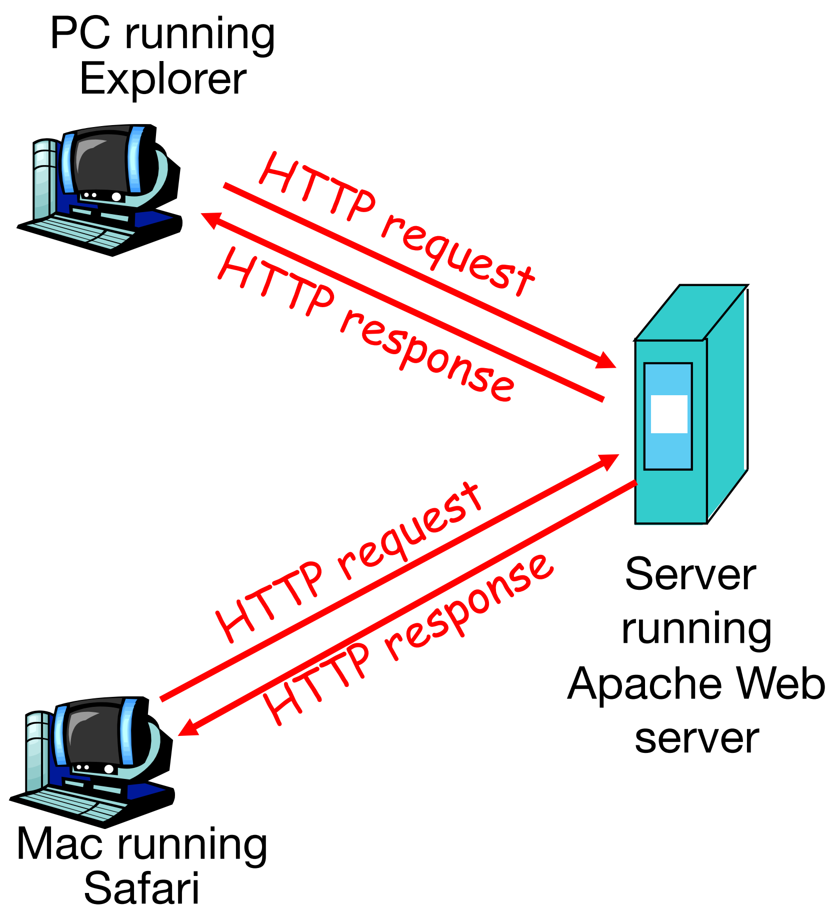


IP address, TCP connection, port number, processes, and sockets

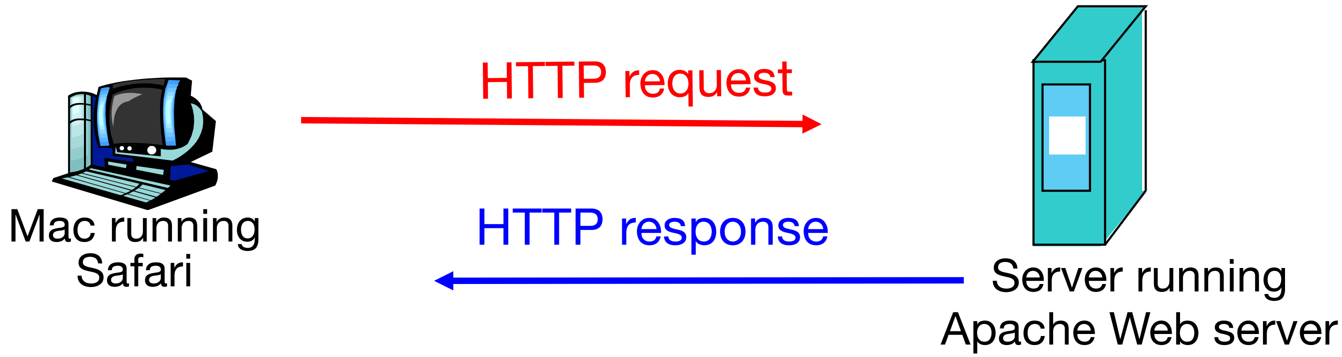


HTTP: HyperText Transfer Protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, and displays Web objects
  - *server:* Web server that sends objects in response to requests
- HTTP/1.0: non-persistent connection
- HTTP/1.1: persistent connection
  - May also do pipelining



Now we got the big picture



- Client (browser) speaks first
  - Setup a TCP connection, destination port 80 (details later)
  - Send HTTP request over the connection
- Server:
  - Accept TCP connection request
  - answers the HTTP request
  - HTTP is "stateless": server maintains no information about past requests

Exactly how HTTP request & reply messages look like?

HTTP request message example

FYI

http://www-net.cs.umass.edu:port#/some-dir/pic.gif

host name — optional, default value: 80 — path name

Written in ASCII (human-readable)

method — URL — version — carriage return character — line-feed character

request line

header lines

A blank line Indicates the end of HTTP header

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Optional message body

Method types

FYI

HTTP/1.0

- GET
- POST
- HEAD
  - Requesting the header only (i.e. response does not include the requested object)

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 from the server

and a few other types

- See the protocol specification RFC2616

https://www.ietf.org/rfc/rfc2616.txt

HTTP response message

FYI

status line (status code, status phrase)

```
HTTP/1.1 200 OK \r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
```

header lines

A blank line

Optional message body

```
data data data data data ...
```

Data: e.g., requested HTML file

HTTP response status codes

important

- Appears in the first line in the server → client response message:
- A few sample status codes:

  `200 OK`
  - request succeeded, requested object carried 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`