

CS 43: Computer Networks

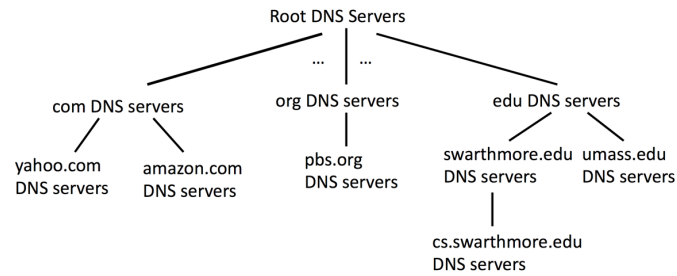
12: Transport Layer & UDP

October 24, 2024



Application Layer

Does whatever an application does!



DNS



Chrome

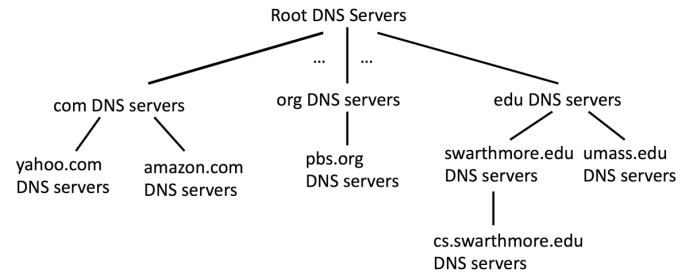


Thunderbird



Skype

Application Layer



DNS



Chrome

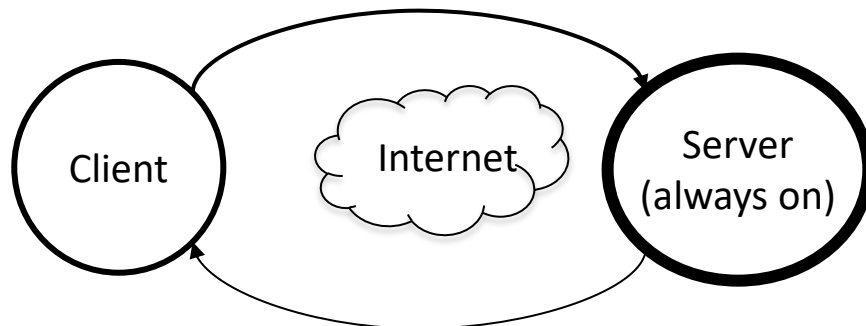


Thunderbird

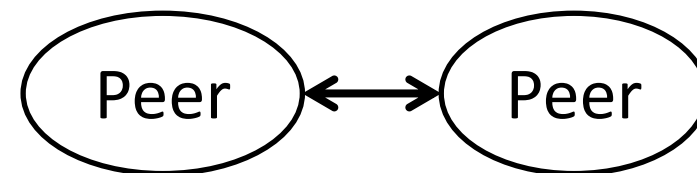


Skype

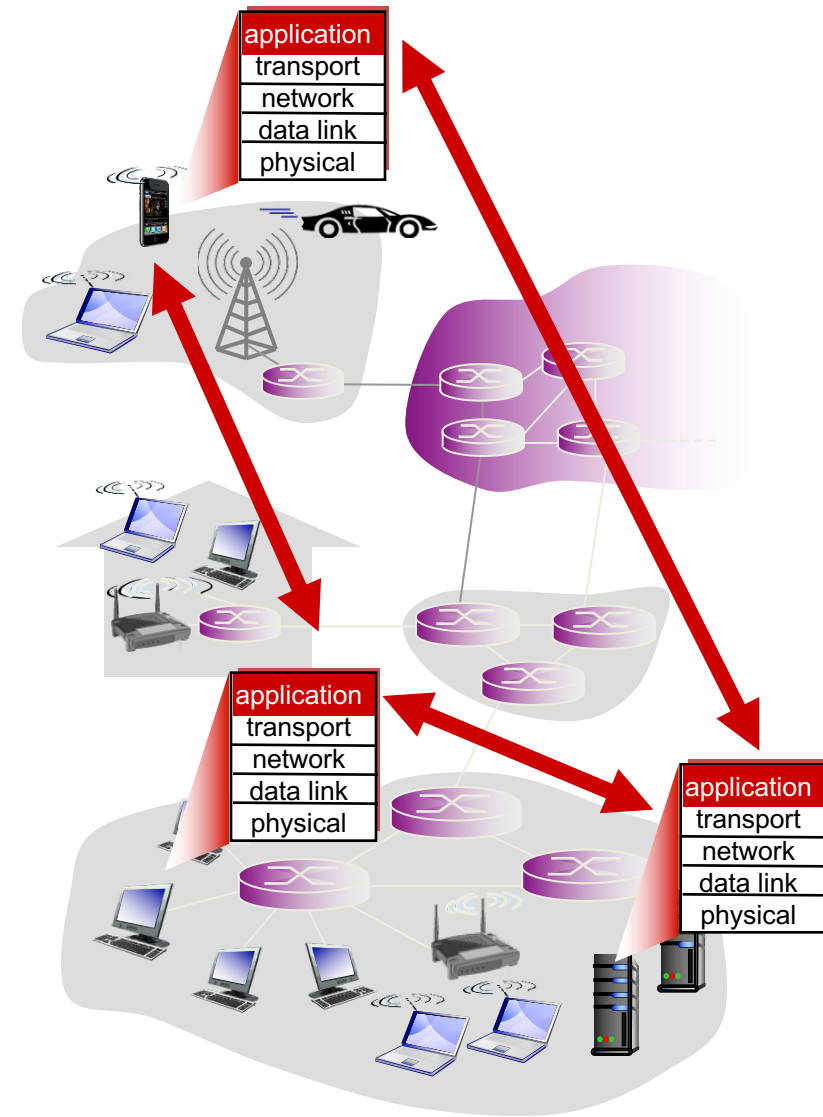
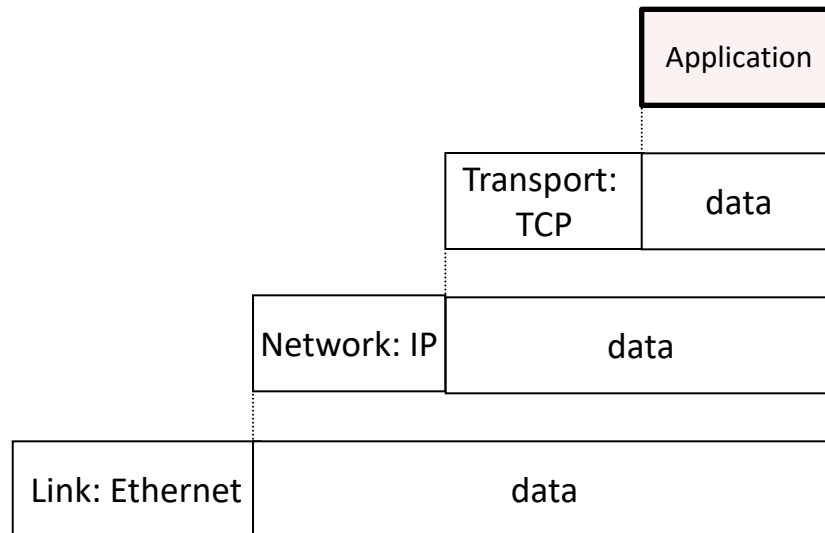
Client-server architecture



Peer-to-peer architecture



Application Layer



Encapsulation:
Higher layer within lower layer

Transport Layer!

Moving down a layer!

Application Layer

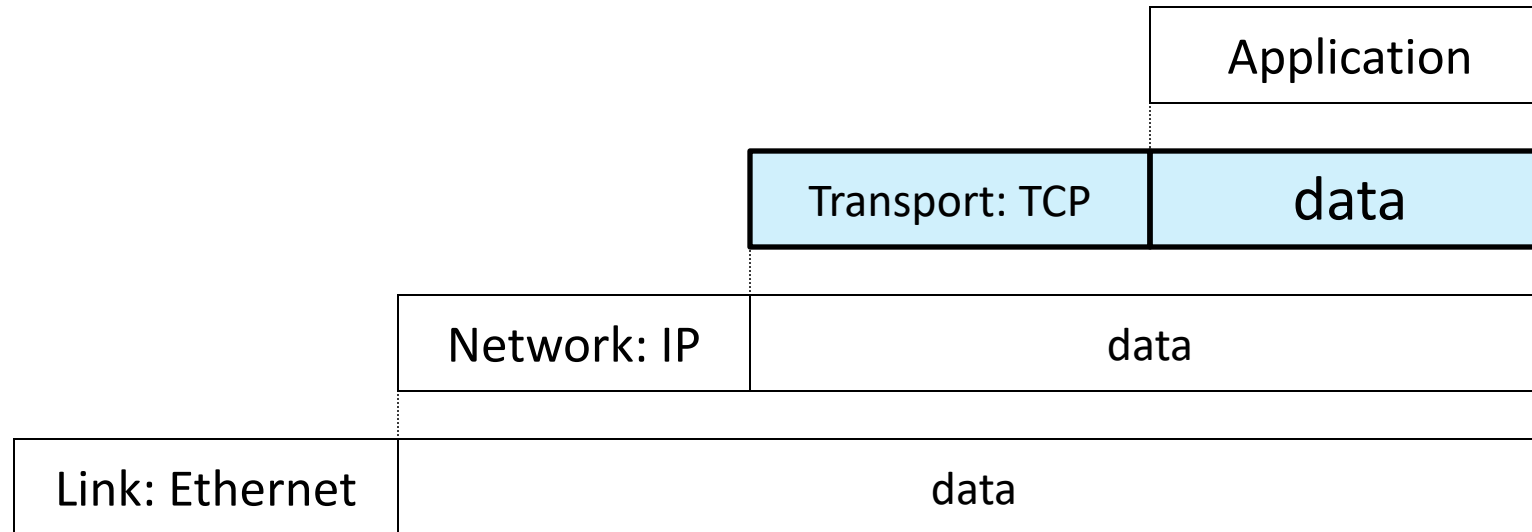
Transport: end-to-end connections, reliability

Network: routing

Link (data-link): framing, error detection

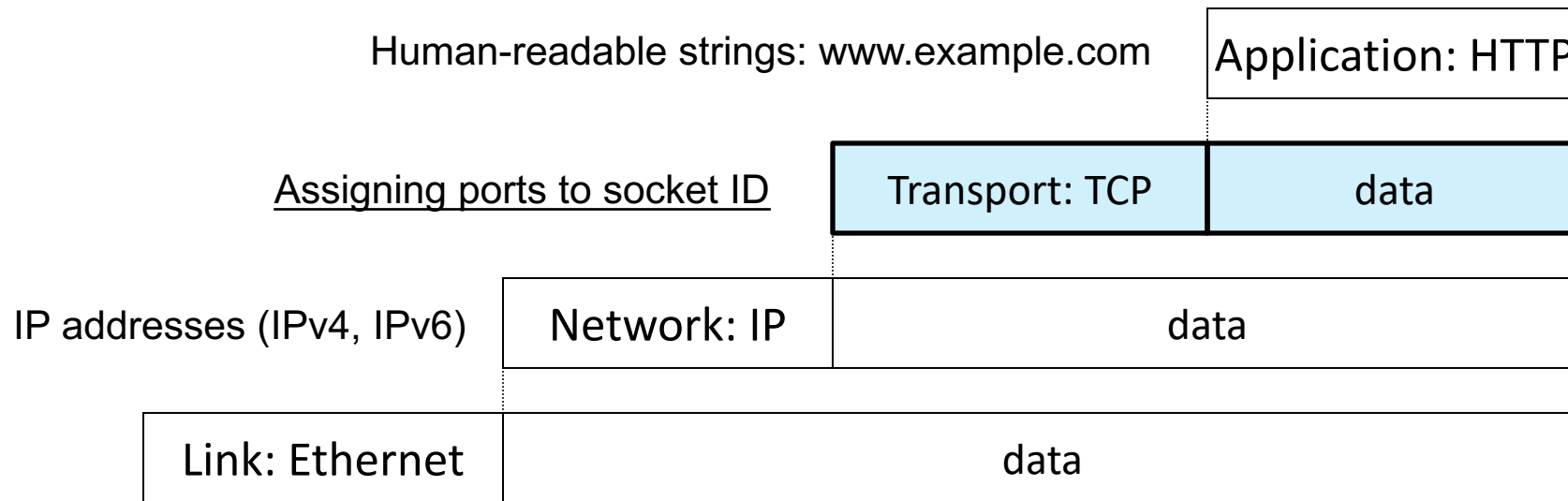
Physical: 1's and 0's/bits across a medium
(copper, the air, fiber)

Message Encapsulation



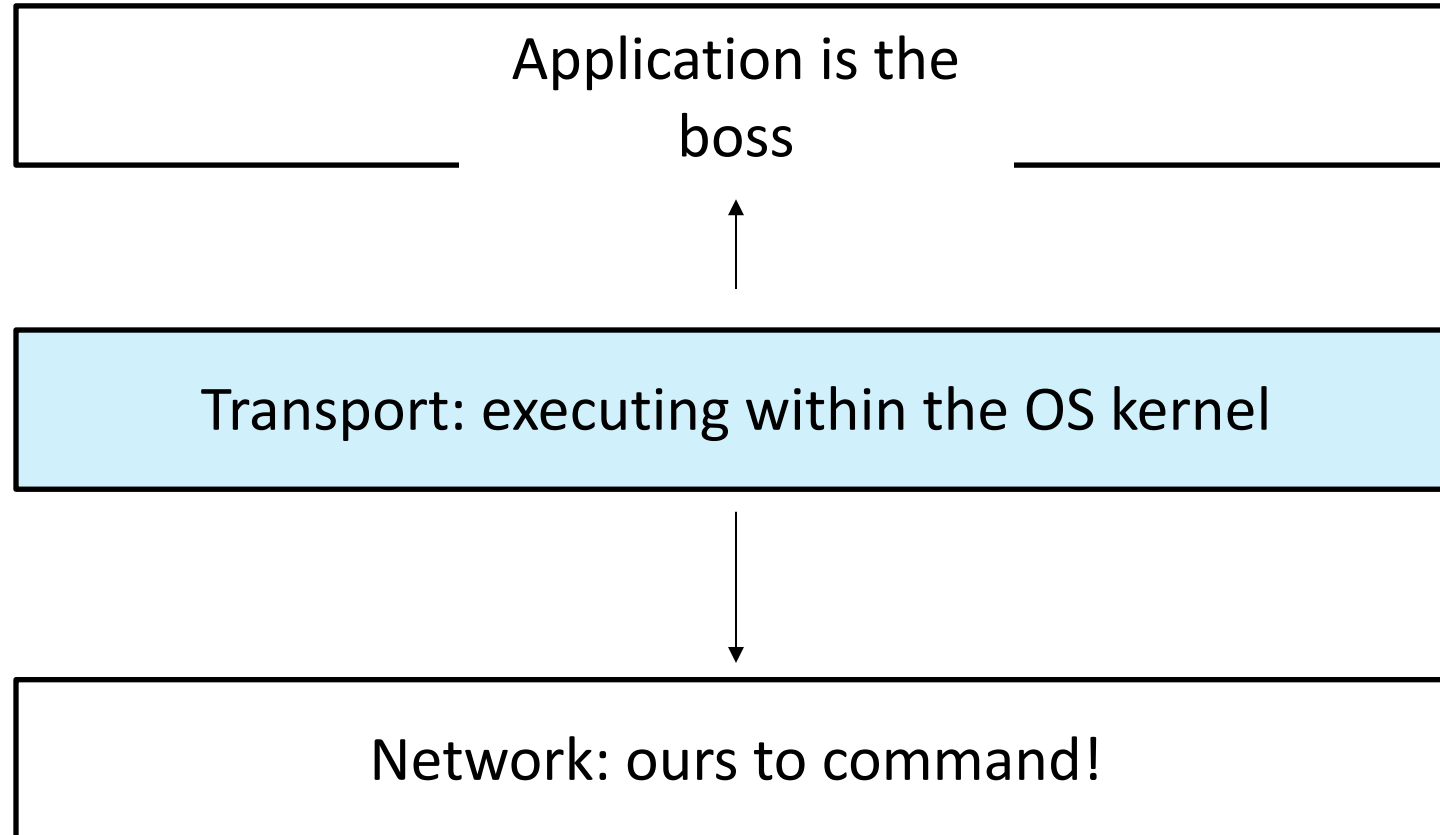
- Higher layer within lower layer
- Each layer has different concerns, provides abstract services to those above

Recall: Addressing and Encapsulation

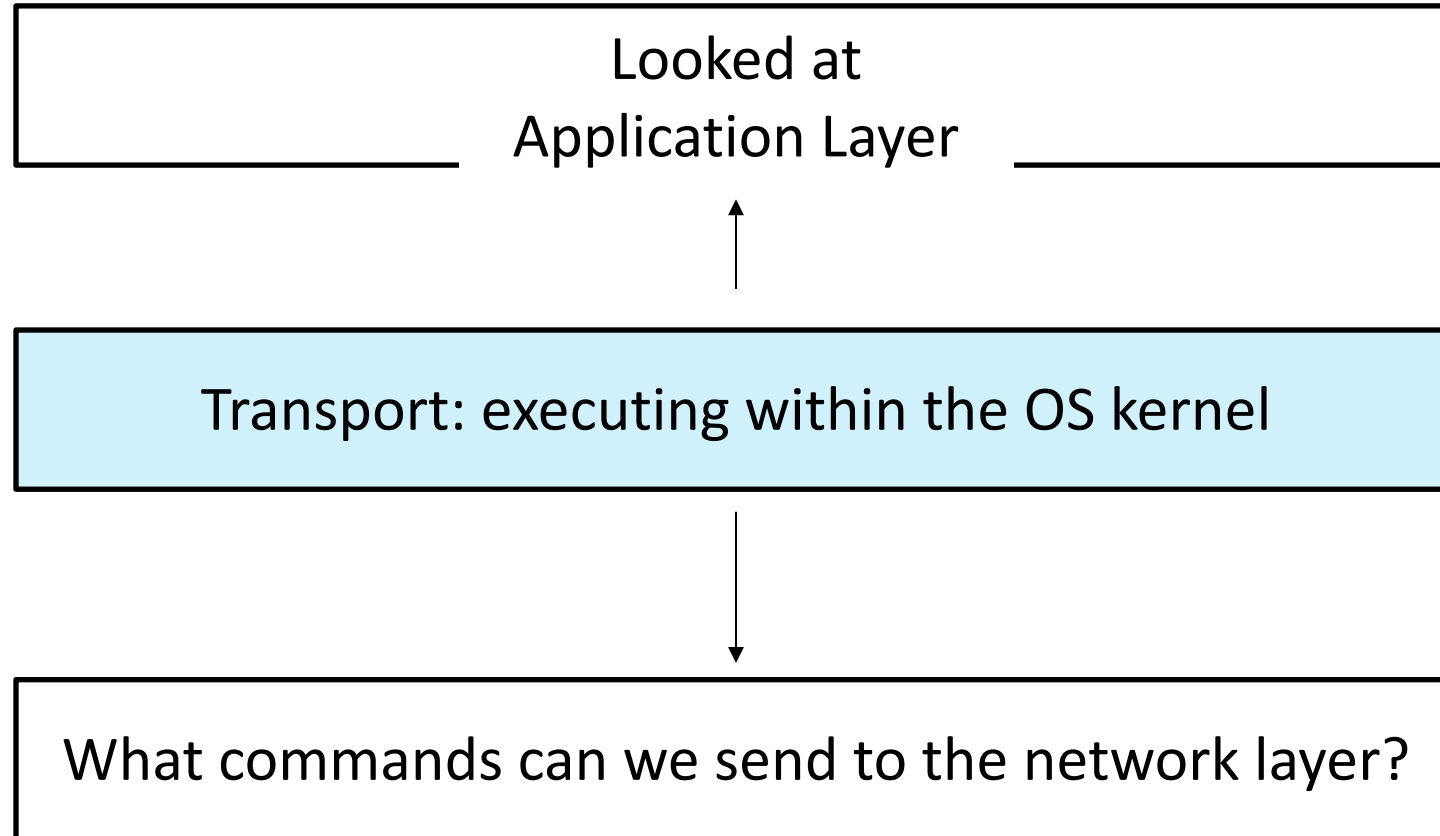


(Network dependent) Ethernet:
48-bit MAC address

Transport Layer perspective

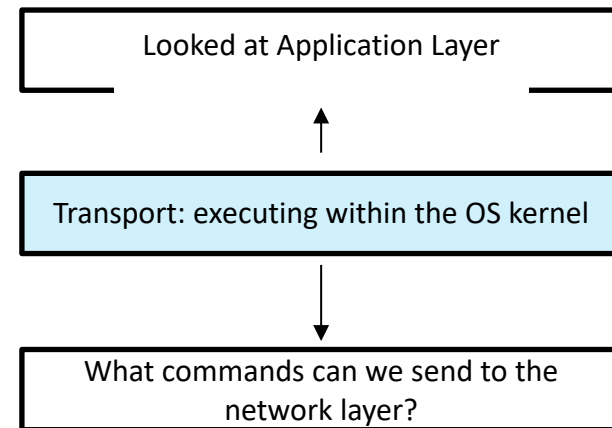


Transport Layer perspective



What services does the network layer provide to the transport layer?

- A. Find paths through the network
- B. Guaranteed delivery rates
- C. Best-effort delivery
- D. Reliable Data Transfer



Network Layer API

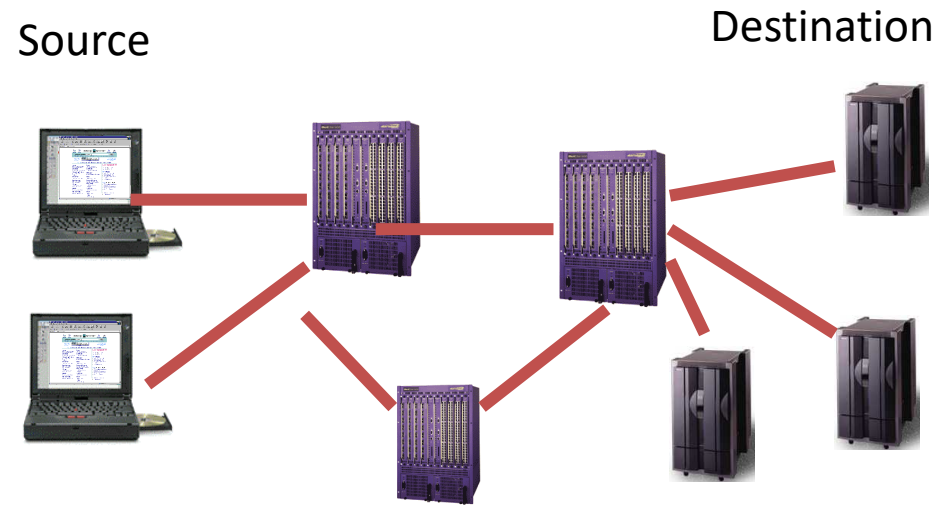
`send_to_host (data, host_IP)` : logical communication between end-hosts

✓ Find paths through the network

✗ reliable data transfer

✓ Best-effort delivery!

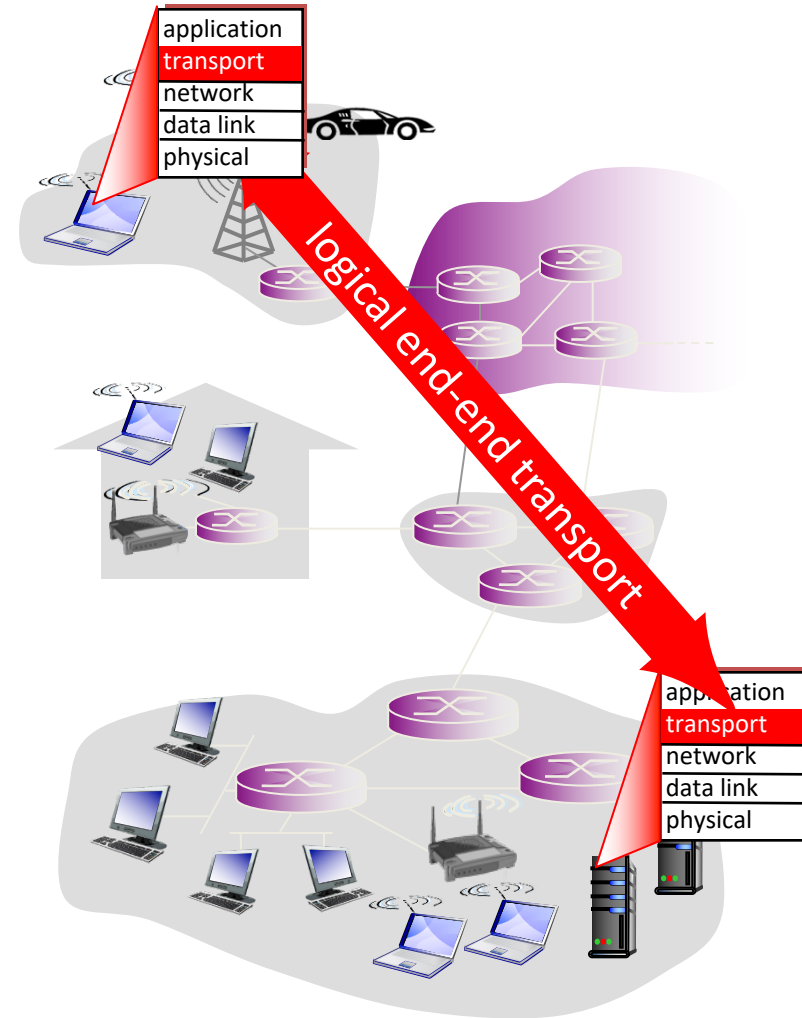
✗ guaranteed delivery (or rate!)



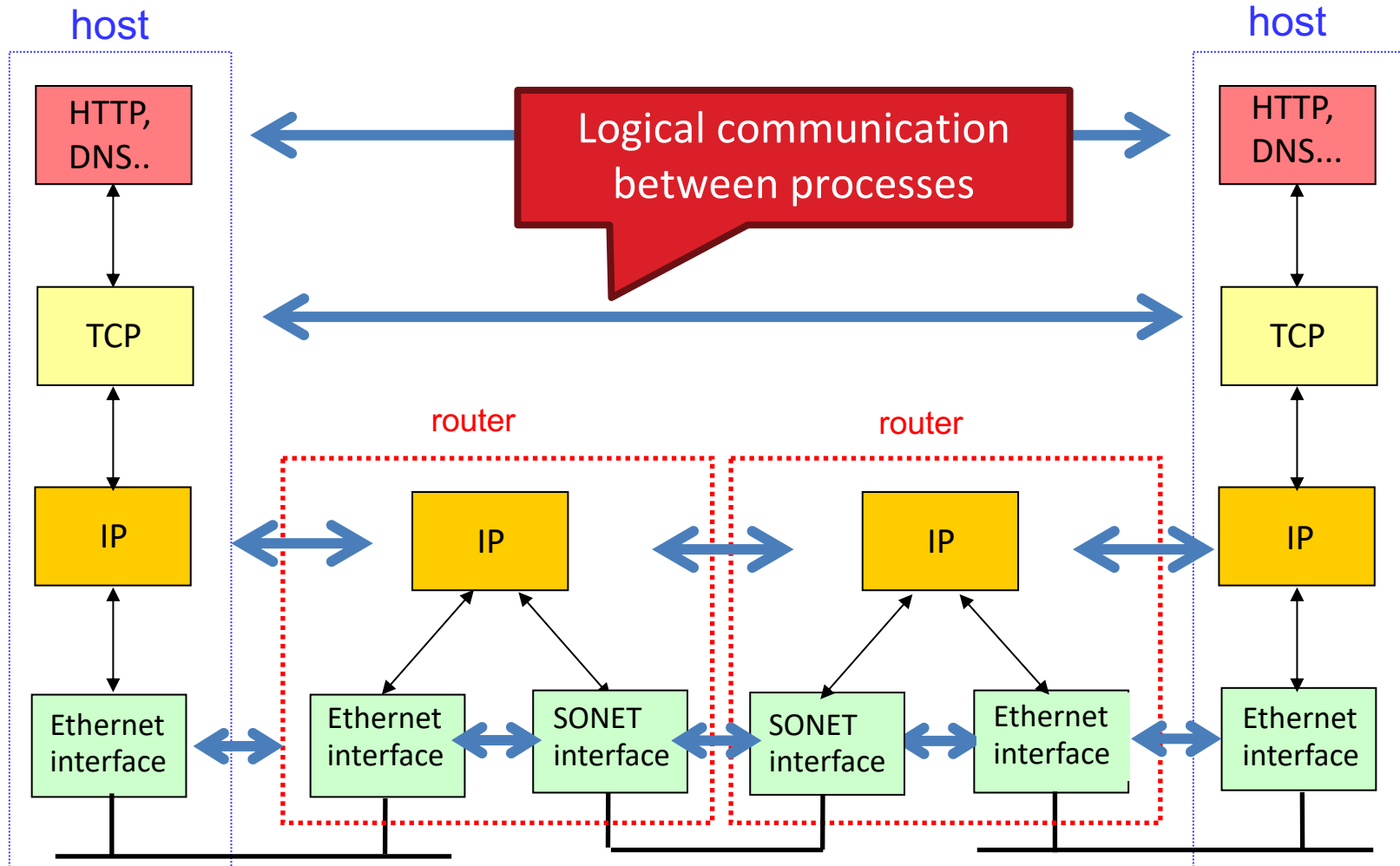
Transport Layer API

Provides logical communication between processes.

`send_data_to_application (data, port, socket)`



Transport Layer: Runs on end systems



How many of these services might we provide at the transport layer? Which?

- Reliable transfers
- Error detection
- Error correction
- Bandwidth guarantees
- Latency guarantees
- Encryption
- Message ordering
- Link sharing fairness with other end hosts

A. 4 or fewer

B. 5

C. 6

D. 7

E. All 8



How many of these services might we provide at the transport layer? Which?

- Reliable transfers (T)
- **Error detection** (U, T)
- Error correction (T)
- Bandwidth guarantees
- Latency guarantees
- Encryption
- Message ordering (T)
- Link sharing fairness (T)

Critical question: Can it be done at the end host?

A. 4 or fewer

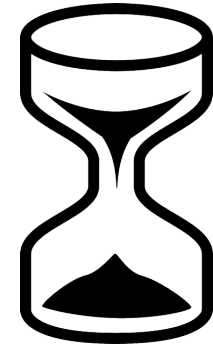
B. 5

C. 6

D. 7

E. All 8

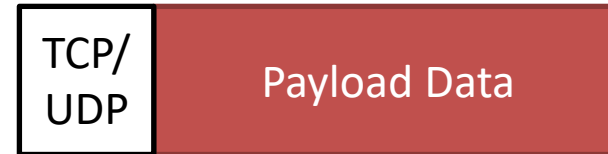
TCP sounds great! UDP...meh. Why do we need it?



- A. It has good performance characteristics.
- B. Sometimes all we need is error detection.
- C. We still need to distinguish between applications.
- D. It basically just fills a gap in our layering model.

Adding Features

- Nothing comes for free
- Data given by application
- Apply header
 - Keeps transport state
 - Attached by sender
 - Decoded by receiver



Moving down a layer!

Application Layer

Transport: end-to-end connections, reliability

Network: routing

Link (data-link): framing, error detection

Physical: 1's and 0's/bits across a medium
(copper, the air, fiber)

Network mnemonics

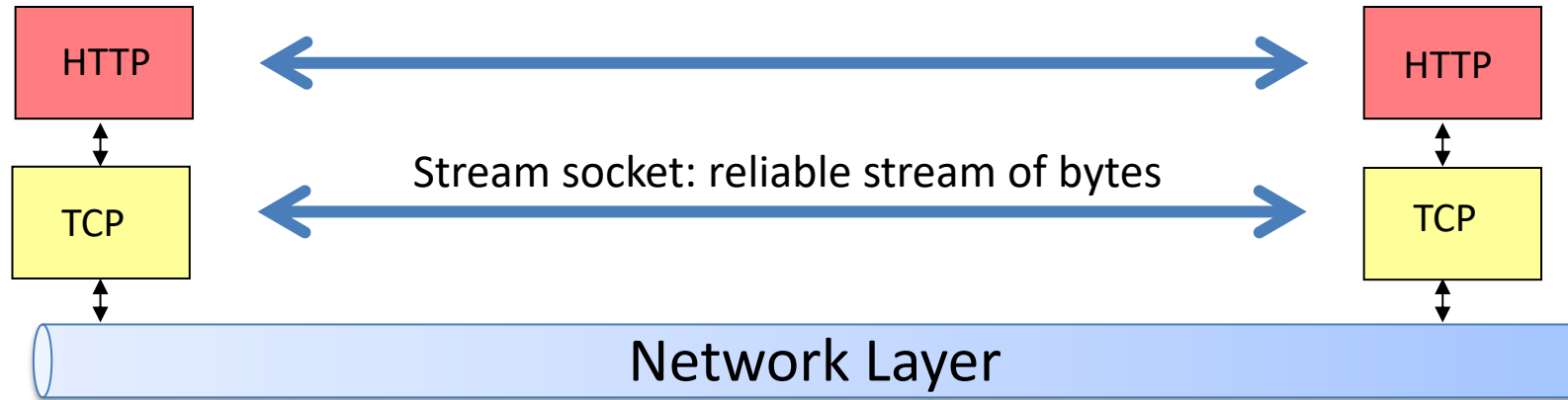
“Big Freaking Deal, Sherlock!”

- Data pieces:
 - Transport: Segments
 - Network: Datagrams (or packets)
 - Link: Frames
 - Physical: Bits

Two Main Transport Layer Protocols

- User Datagram Protocol (UDP)
 - Unreliable, unordered delivery
- Transmission Control Protocol (TCP)
 - Reliable in-order delivery

TCP: Transport Control Protocol



GET http://www.google.com HTTP/1.1
Host: www.google.com
...



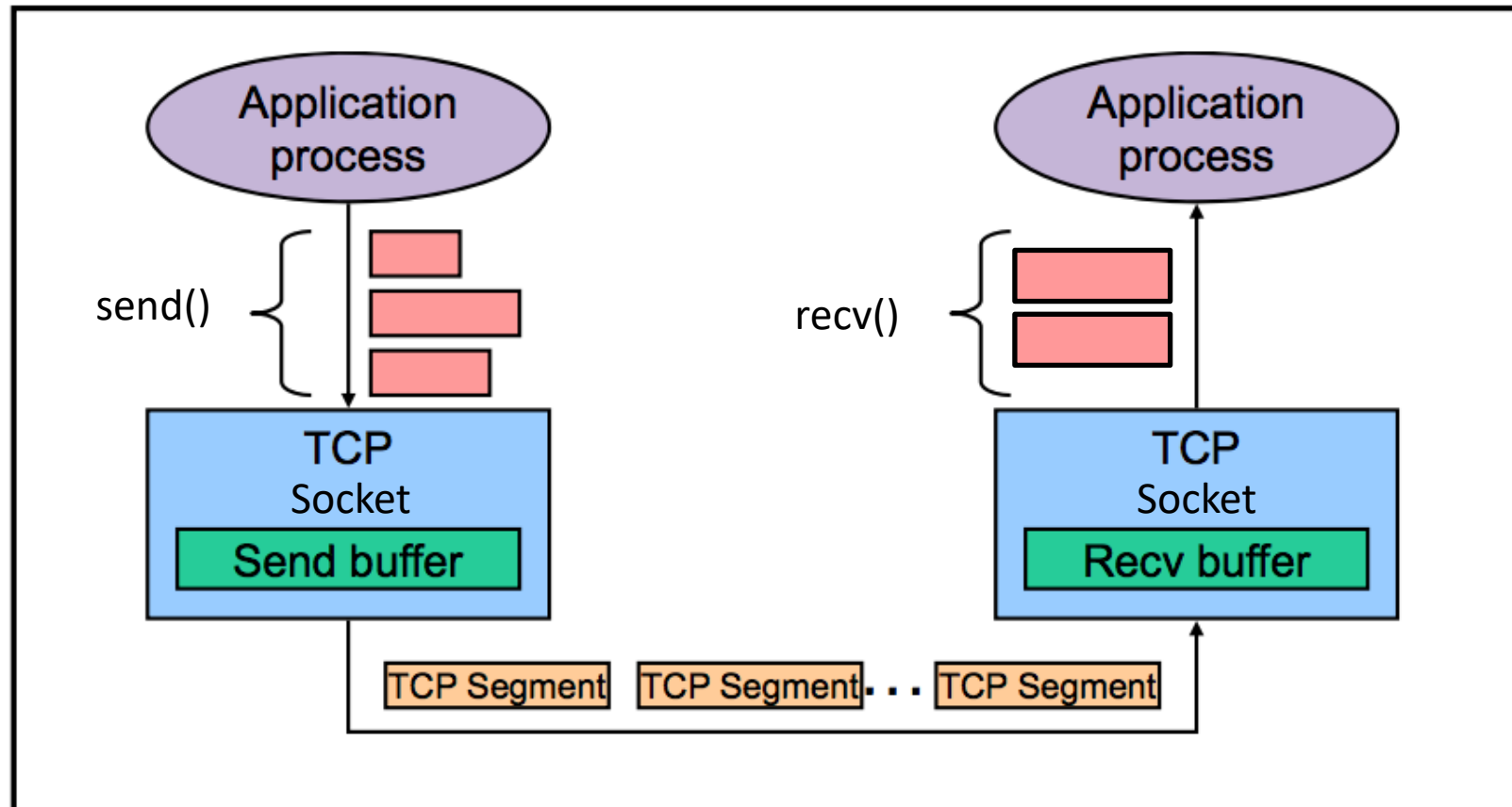
3 google.c 2 p://www. 1 GET htt



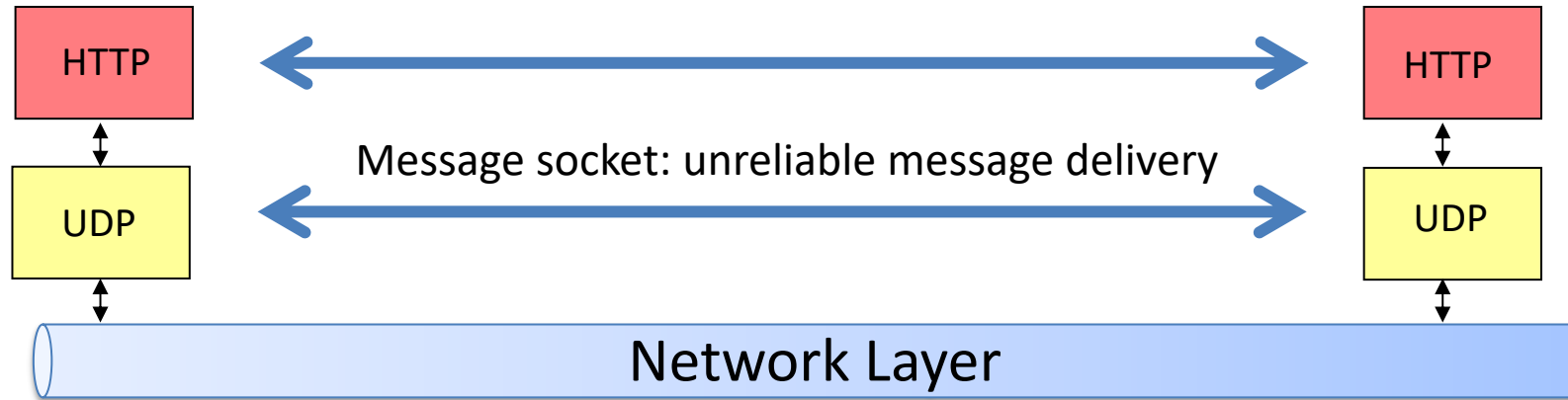
- application layer packet boundaries are not preserved
- multiple send() -> one recv()
 - 1 send() -> multiple recv()

TCP: Stream abstraction

send() and recv() need not have a 1-1 correspondence.



UDP: User Datagram Protocol



dig demo.cs.swarthmore.edu



Header
Question
Answer
Authority
Additional

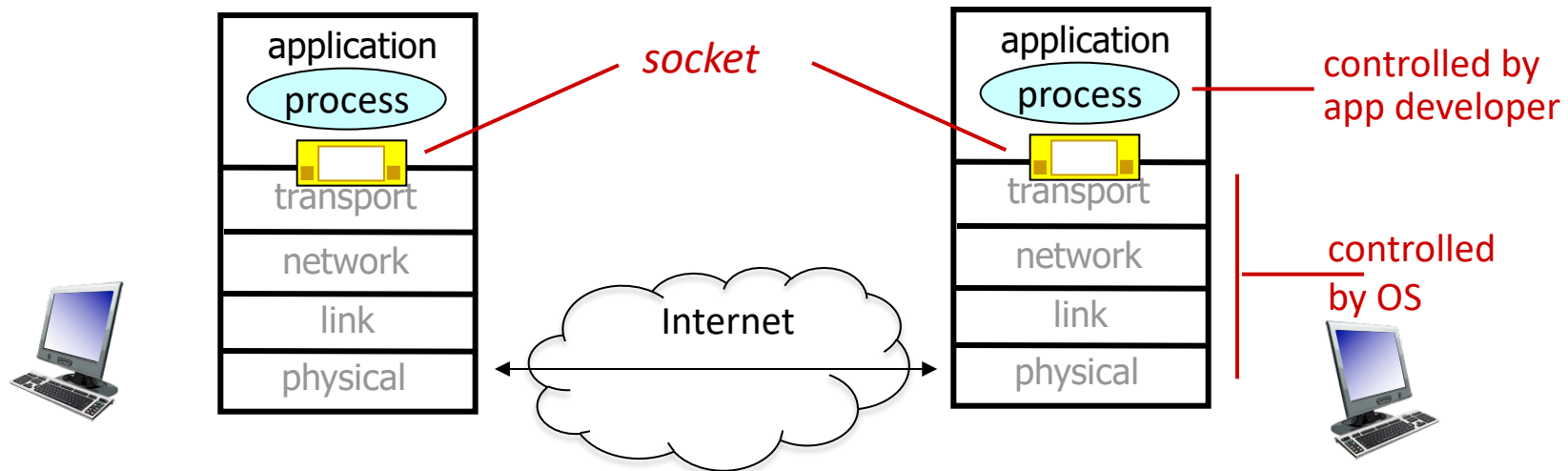


application layer packet boundaries are preserved

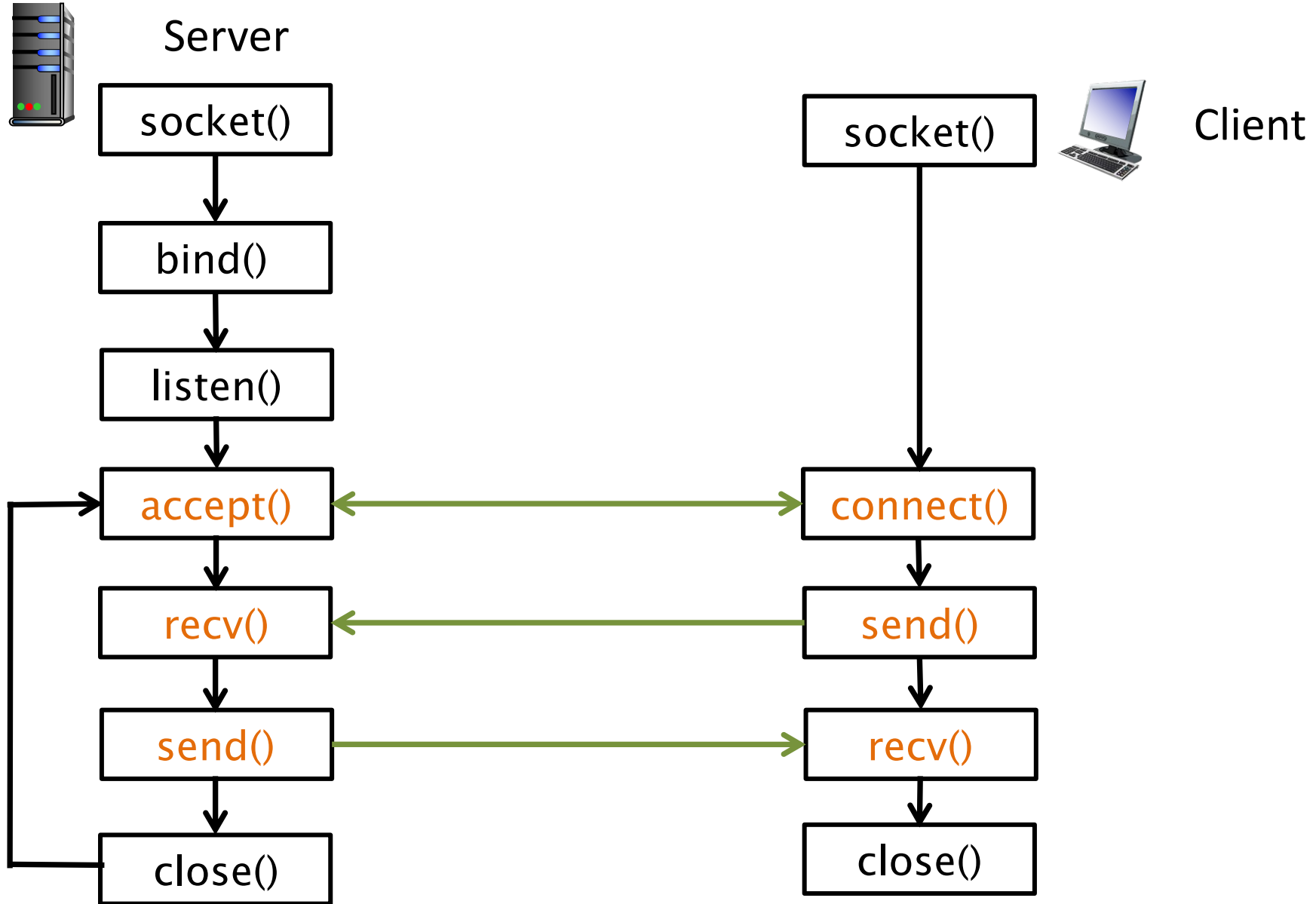
- 1 send() -> 1 recv()

Sockets

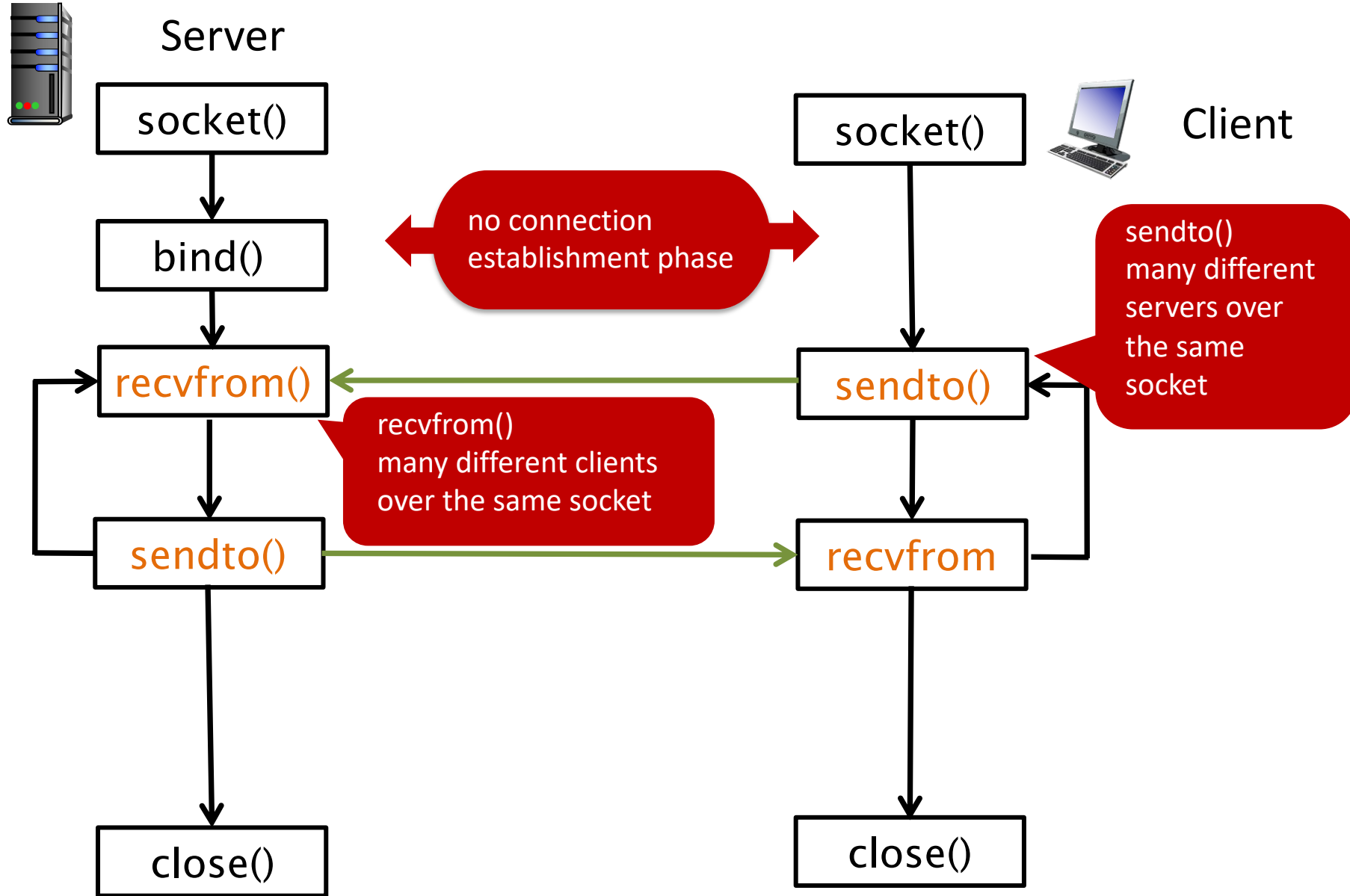
- Application processes communicate using “sockets” /mailboxes
 - Abstraction: sends/receives data to/from its **socket**



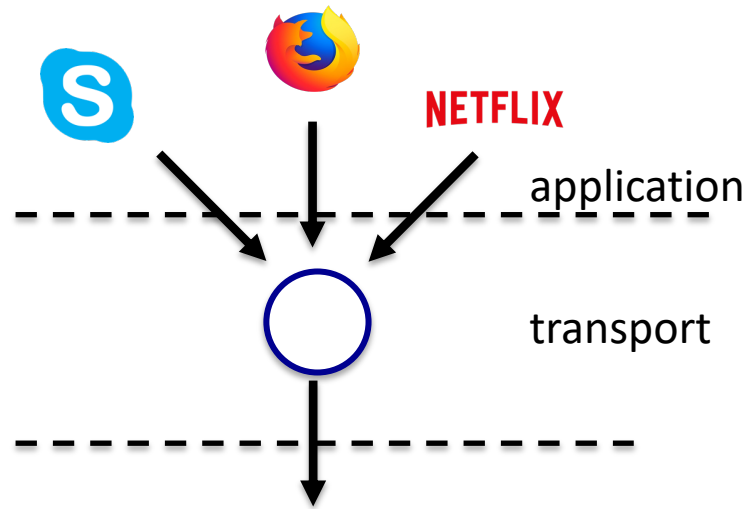
Recall TCP Sockets



UDP Sockets

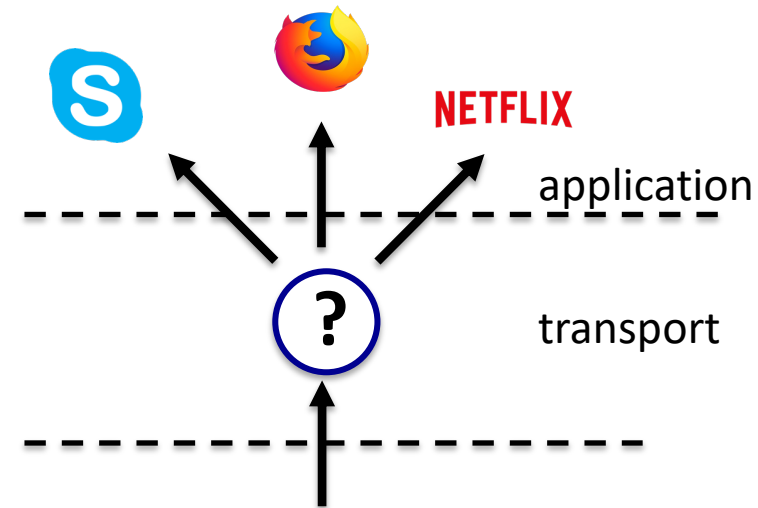


Multiplexing/demultiplexing: Transport Layer



multiplexing

assign port # to distinguish between applications on the same end hosts



de-multiplexing

use port # to direct packets to the correct application layer processes

Multiplexing



Multiplexing/
demultiplexing:

Happens at every layer!

De-multiplexing



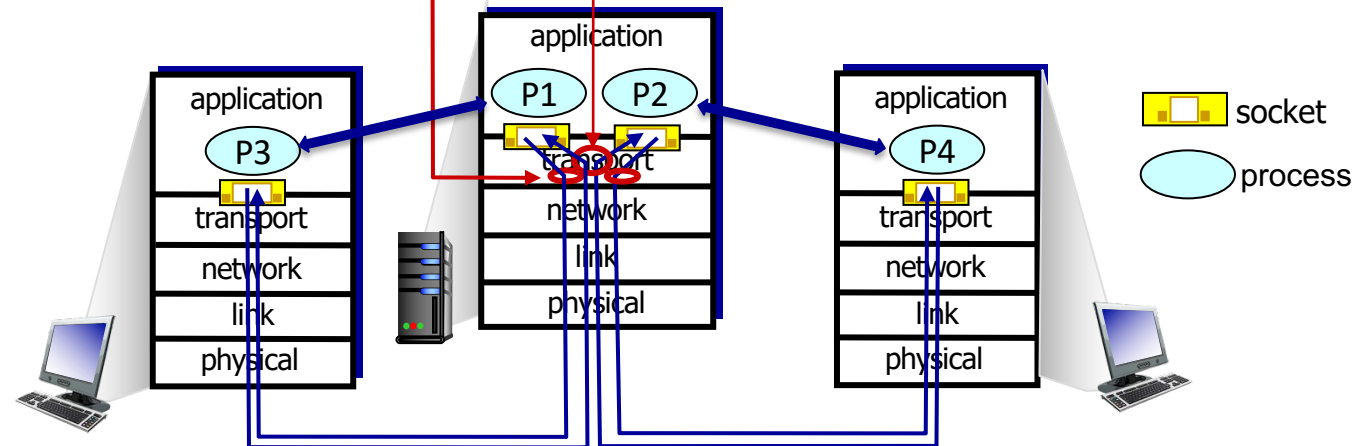
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket

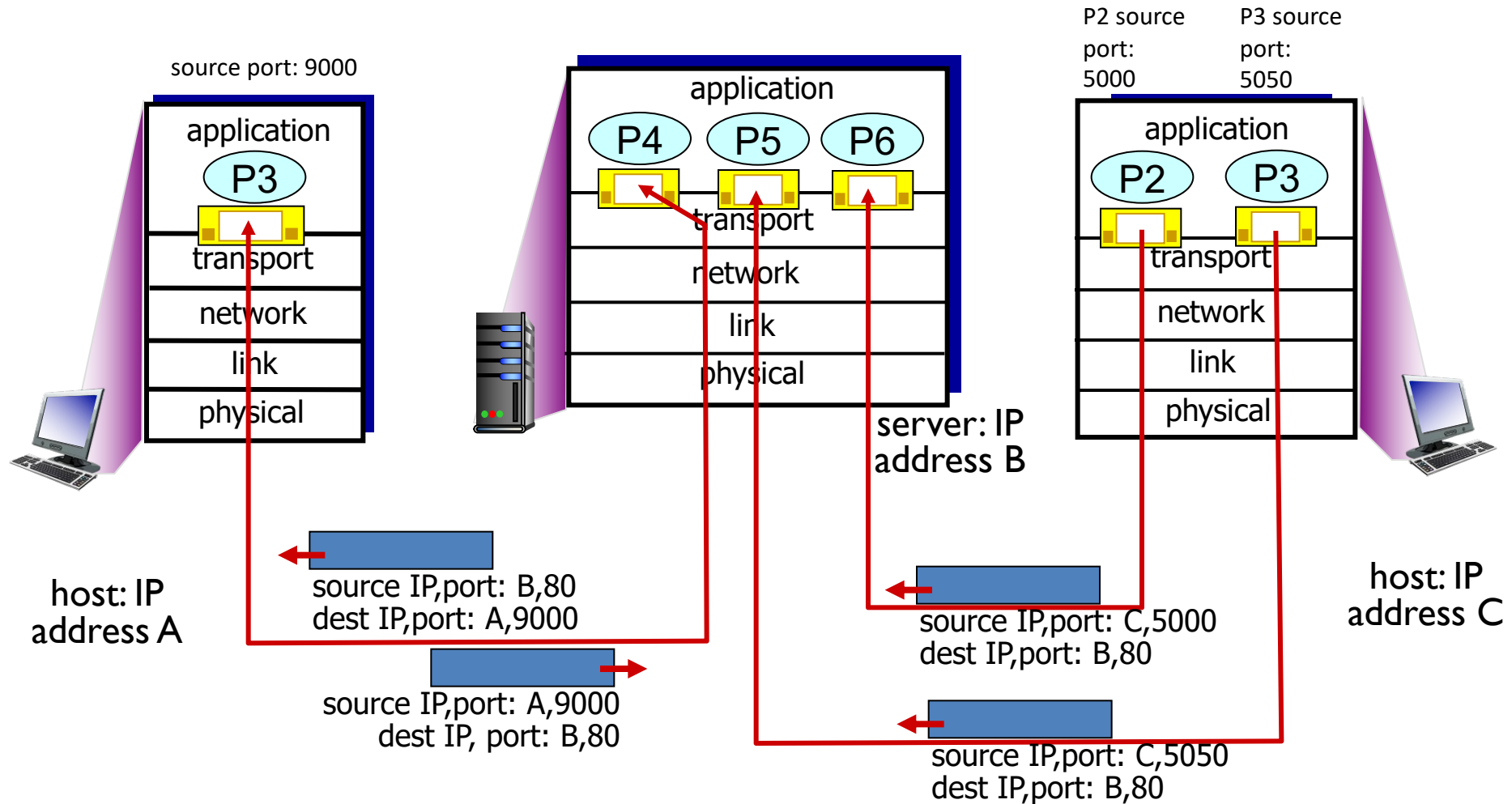


Recall: Connection-oriented: example

- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- Receiver uses all four values to direct segment to appropriate socket

Recall: Connection-oriented: HTTP example

A TCP socket is uniquely identified by (source IP, source port, dest IP, dest port)



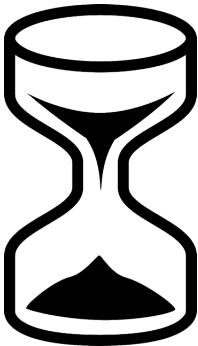
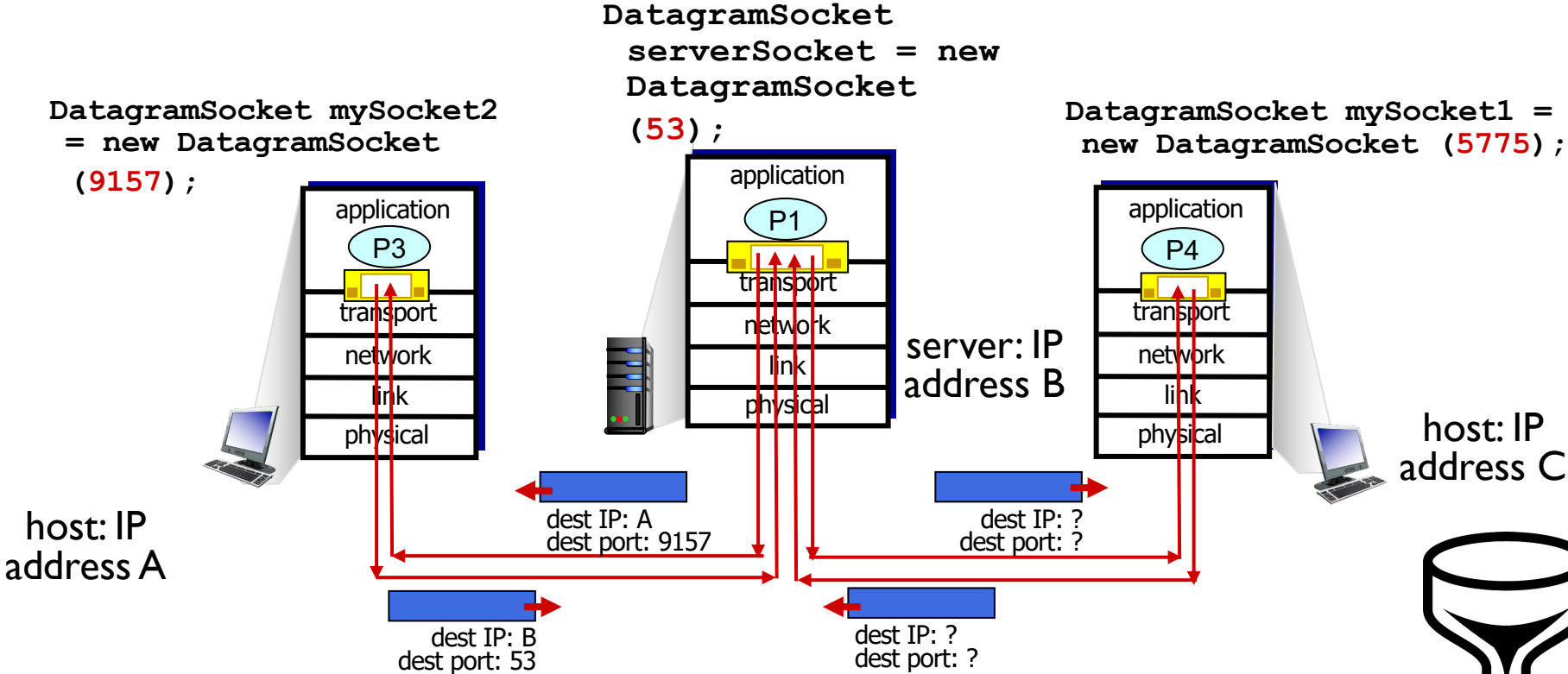
Connectionless: example

- UDP socket identified by 2-tuple:
 - dest IP address
 - dest port number
- when receiving host receives UDP segment:
 - checks destination port # in segment
 - directs UDP segment to socket with that port #

UDP datagrams with same dest. port #, but different source (IP/port #) will be directed to same socket at receiving host

Connectionless demultiplexing: an example

A UDP socket is uniquely identified by (dest IP, dest port)



UDP – User Datagram Protocol

- Unreliable, unordered service
- Adds:
 - multiplexing,
 - checksum (error detection)

UDP: User Datagram Protocol [RFC 768]

“No frills,” “Bare bones” Internet transport protocol

- RFC 768 (1980)
- Length of the document?

UDP: User Datagram Protocol [RFC 768]

“Best effort” service,
UDP segments may be:

- Lost
- Delivered out of order (same as underlying network layer)



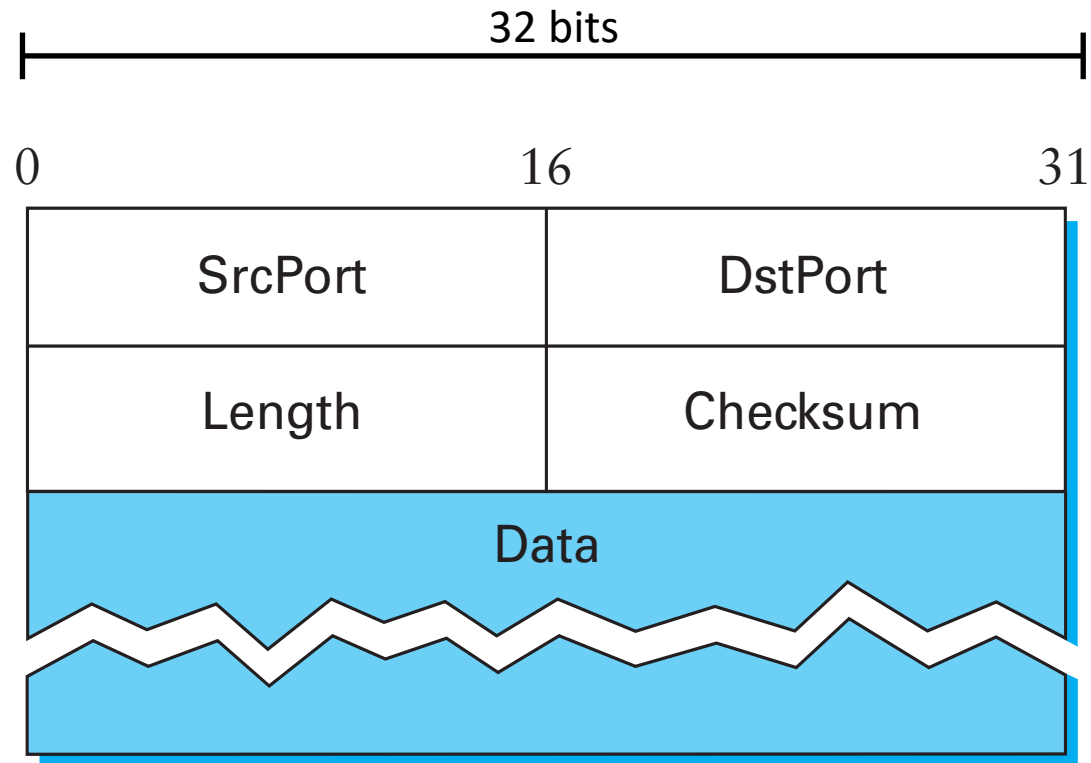
How many of the following steps does UDP implement? (which ones?)

1. exchange an initiate handshake (connection setup)
2. break up packet into segments at the source and number them
3. place segments in order at the destination
4. error-checking with checksum

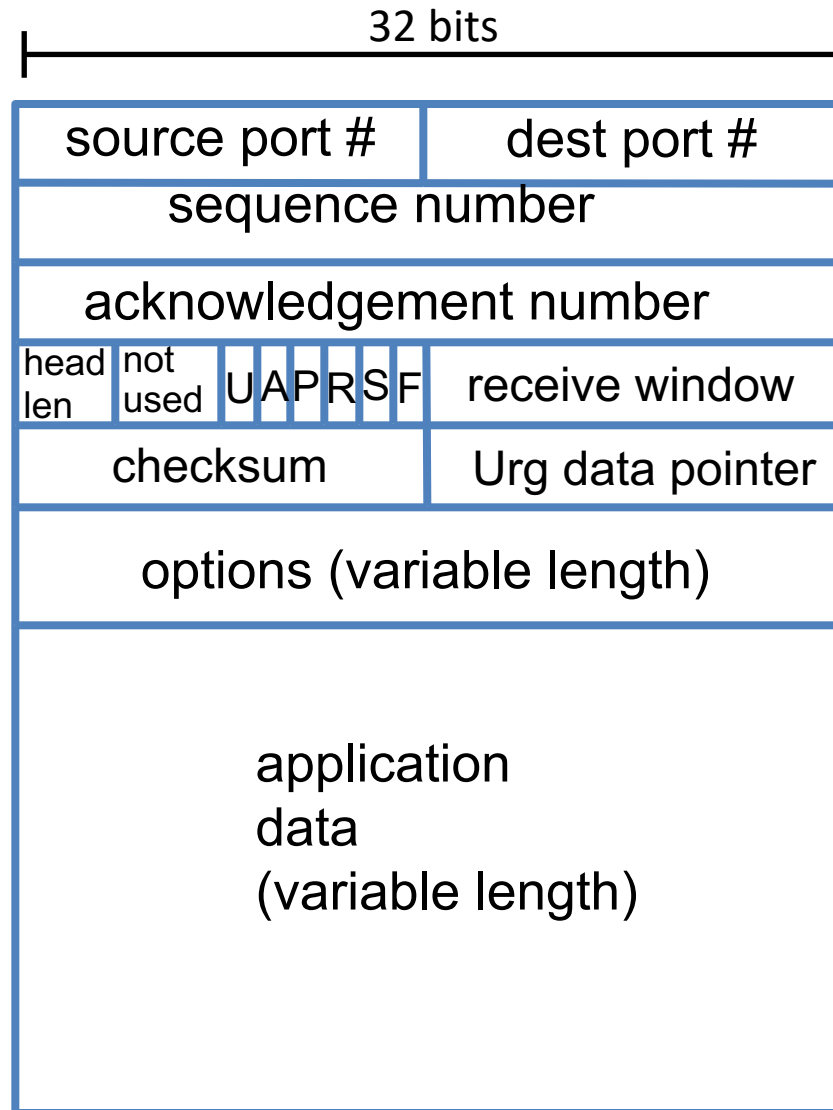
- A. 1
- B. 2
- C. 3
- D. 4



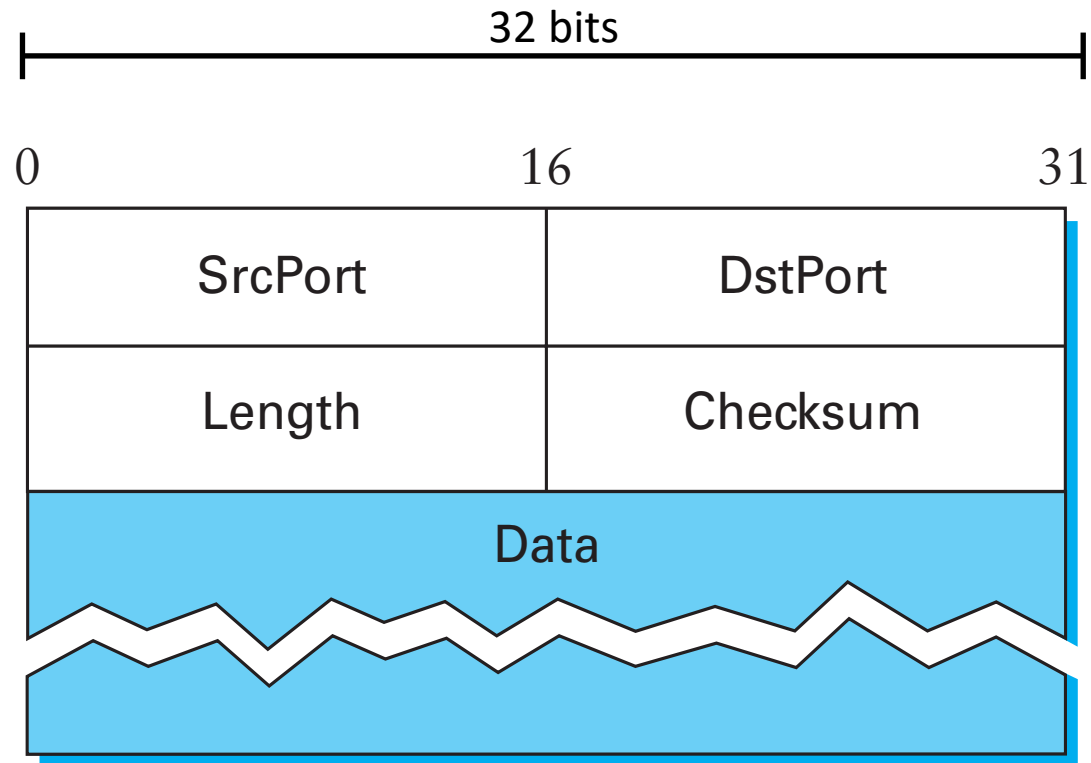
UDP Segment



TCP Segment!



UDP Segment



UDP Checksum

- Goal: Detect transmission errors (e.g. flipped bits)
 - Router memory errors
 - Driver bugs
 - Electromagnetic interference

UDP Checksum

RFC: “Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.”

(? _ ?)

UDP Checksum at the Sender

- Treat the entire segment as 16-bit integer values
- Add them all together (sum)
- Put the **1's complement** in the checksum header field

One's Complement

- In bitwise compliment, all of the bits in a binary number are flipped.
- So 1111000011110000 -> 0000111100001111

Checksum example

example: add two 16-bit integers

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
<hr/>	
wraparound	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
<hr/>	
sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Receiver

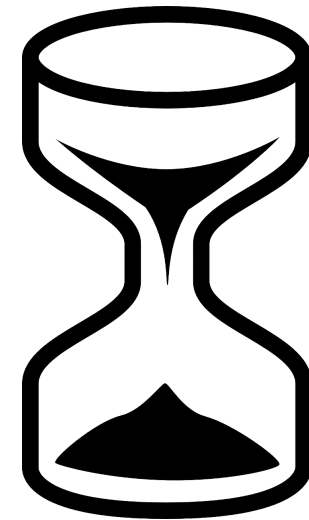
- Add all the received data together as 16-bit integers
- Add that to the checksum
- If result is not 1111 1111 1111 1111, there are errors!
- If there are errors chuck the packet.



If our checksum addition yields all ones, are we guaranteed to be error-free?

A. Yes

B. No



Checksum example

example: add two 16-bit integers

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
<hr/>	
wraparound	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
<hr/>	
sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

UDP Applications

- Latency sensitive
 - Quick request/response (DNS)
 - Network management (SNMP, DHCP)
 - Voice/video chat
- Communicating with *lots* of others

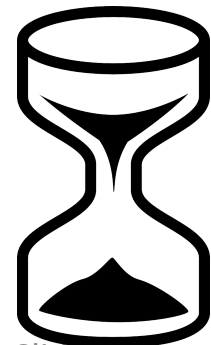
Recall: TCP send() blocking

With TCP, send() blocks if buffer full.

UDP sendto() blocking

With TCP, send() blocks if buffer full.

- Does UDP need to block? Should it?
 - A. Yes, if buffers are full, it should.
 - B. It doesn't need to, but it might be useful.
 - C. No, it does not need to and shouldn't do so.



UDP sendto() blocking

With TCP, send() blocks if buffer full.

- Does UDP need to block? Should it?
 - A. Yes, if buffers are full, it should.
 - B. It doesn't need to, but it might be useful.
 - C. No, it does not need to and shouldn't do so.

Summary

Transport Layer:

- Provides a logical communication between processes/ applications
- packets are called segments at the transport layer
- Transport layer protocol: responsible for adding port numbers (mux/demux segments)

Summary

UDP:

- No “frills” protocol, No state maintained about the packet
- Checksum (1’s complement) over IP + UDP + payload.
 - can only correct for 1 bit errors.
- adds port numbers over unreliable network (best effort)
- applications:
 - latency sensitive applications: real-time audio, video
 - communicating with a lot of end-hosts (like DNS)
- UDP Sockets:
 - do not need to be implemented as blocking system calls for correctness since the only guarantee UDP makes is best-effort delivery.
 - however send/rcv can be implemented as blocking system calls depending on the application