

This lecture will be recorded!!

Welcome to

CS 3516:
Computer Networks

Prof. Yanhua Li

Time: 9:00am –9:50am M, T, R, and F

Zoom Link

Fall 2020 A-term

Lab 2 due 9/21 Monday.

Quiz 4: Grading done by today!

Quiz 5: 9/21 Monday on P2P

Project 2 started

We will do a demo next M or T

Application Layer 2-2

Proj 1: Extended to next M 9/21

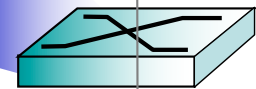
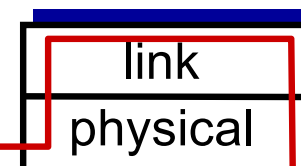
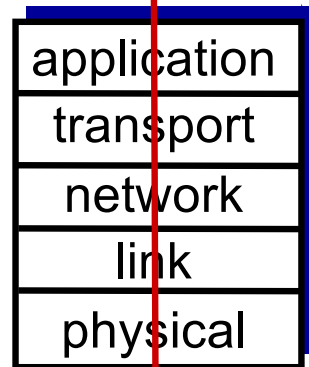
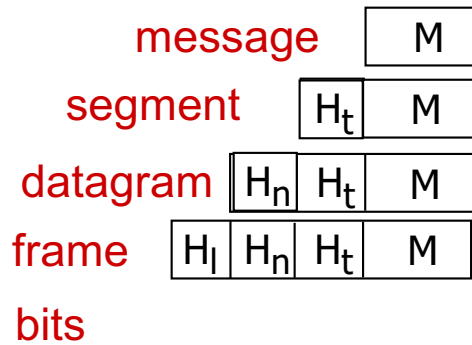
Mid-term exam

1. We will send you sample questions during the weekend
 2. Next Friday 9/25 in Canvas 8:55-9:50AM. Zoom for Questions
- ❖ We will have a review session for the sample questions.
 - ❖ In-person session on Thursday 9/24 in AK219.

Encapsulation

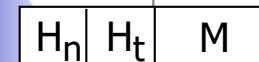
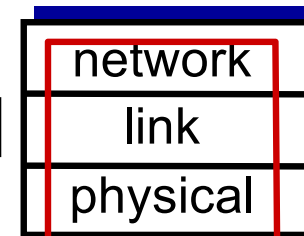
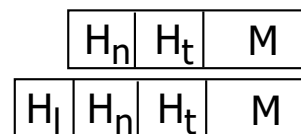
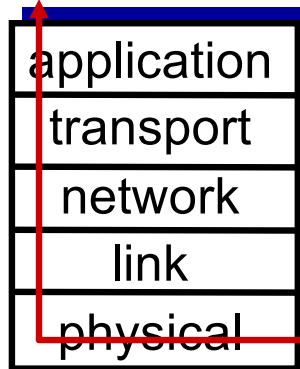
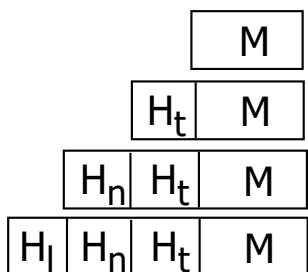


source



switch

destination



router

Chapter 3: Transport Layer

our goals:

- ❖ understand principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
- ❖ learn about Internet transport layer protocols:
 - UDP: connectionless transport
 - TCP: connection-oriented reliable transport

Chapter 3 outline

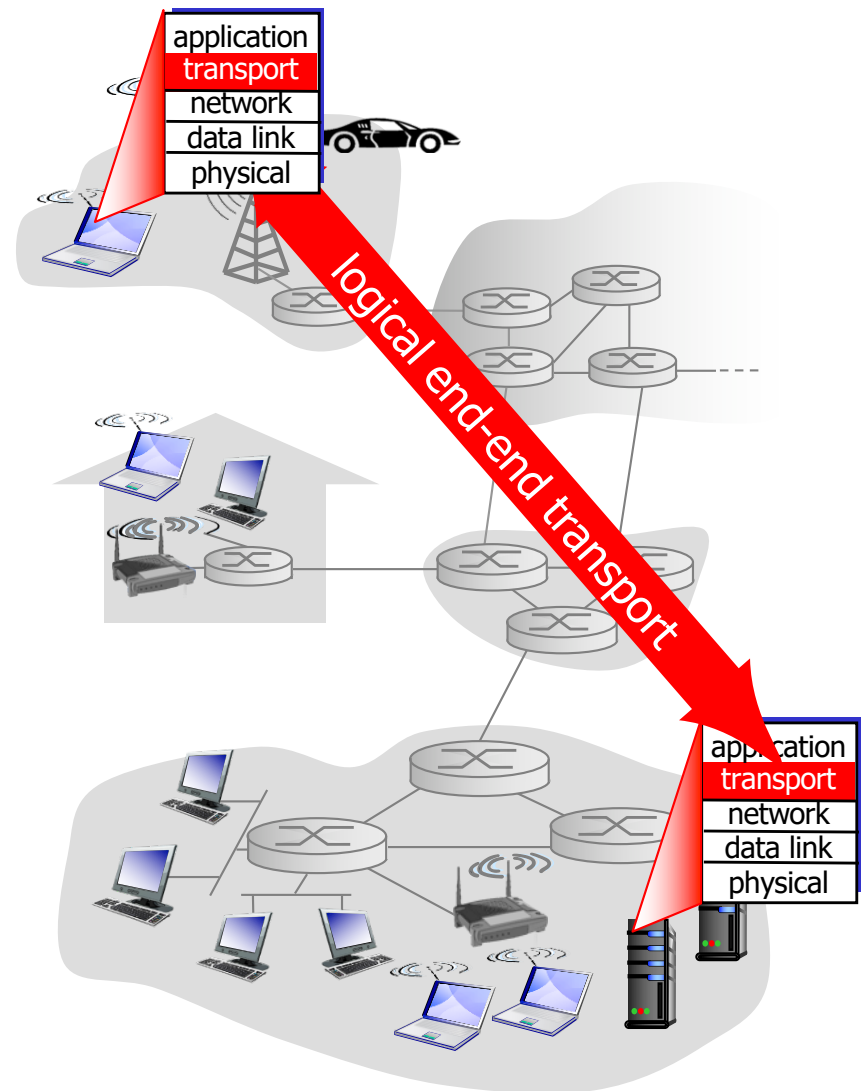
3.1 transport-layer
services

3.2 multiplexing and
demultiplexing

3.3 connectionless
transport: UDP

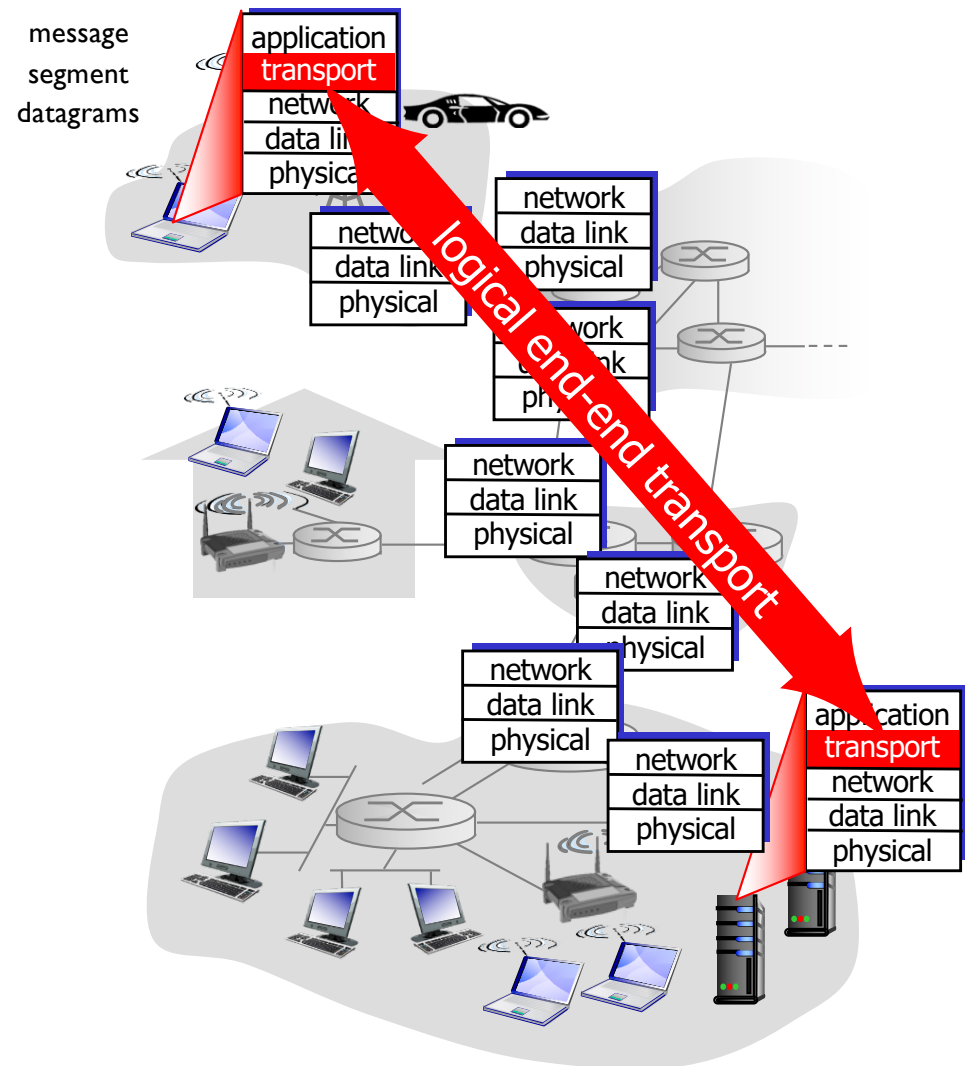
Transport services and protocols

- ❖ provide **logical communication** between app processes running on different hosts
- ❖ transport protocols run in end systems
 - **send side**: breaks app messages into **segments**, passes to network layer
 - **rcv side**: reassembles segments into messages, passes to app layer
- ❖ more than one transport protocol available to apps
 - Internet: TCP and UDP



Internet transport-layer protocols

- ❖ **reliable, in-order delivery (TCP)**
 - retransmission
 - congestion control
 - flow control
 - connection setup
- ❖ **unreliable, unordered delivery: UDP**
 - no-frills extension of “best-effort” IP



Chapter 3 outline

3.1 transport-layer
services

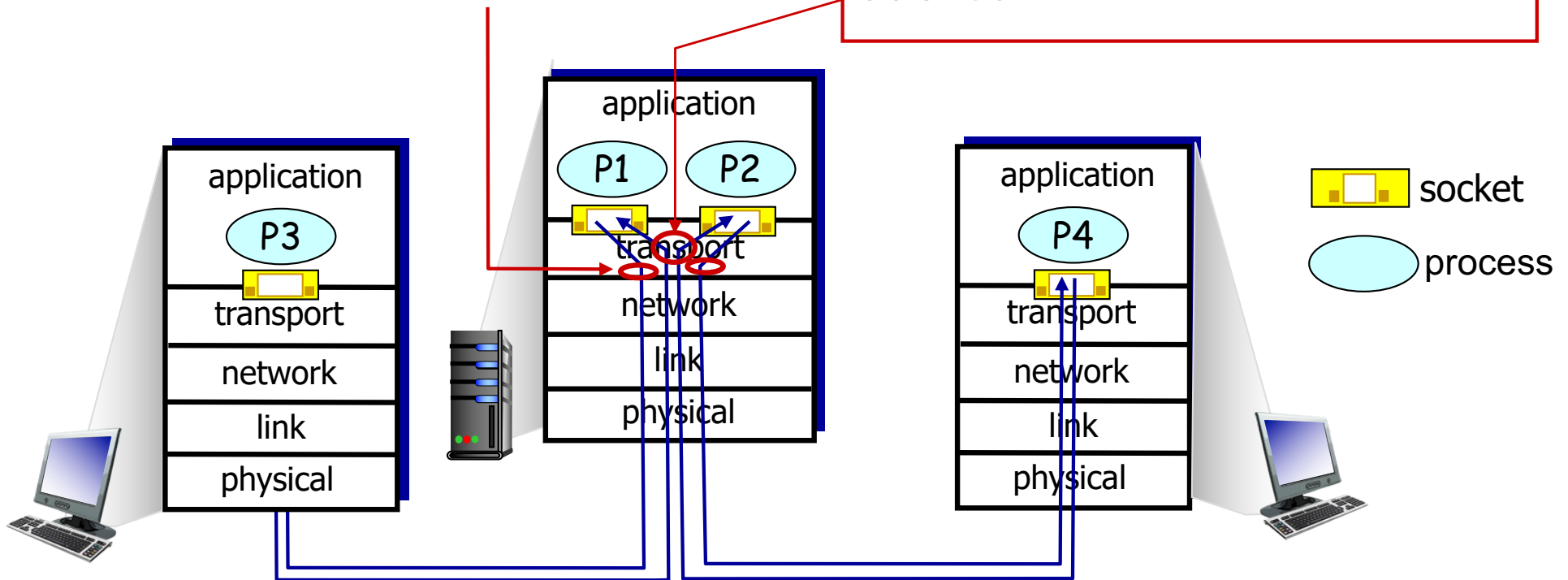
3.2 multiplexing and
demultiplexing

3.3 connectionless
transport: UDP

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



Connectionless demultiplexing

❖ when host receives UDP segment:

- checks **destination port #** in segment
- directs UDP segment to socket with that port #



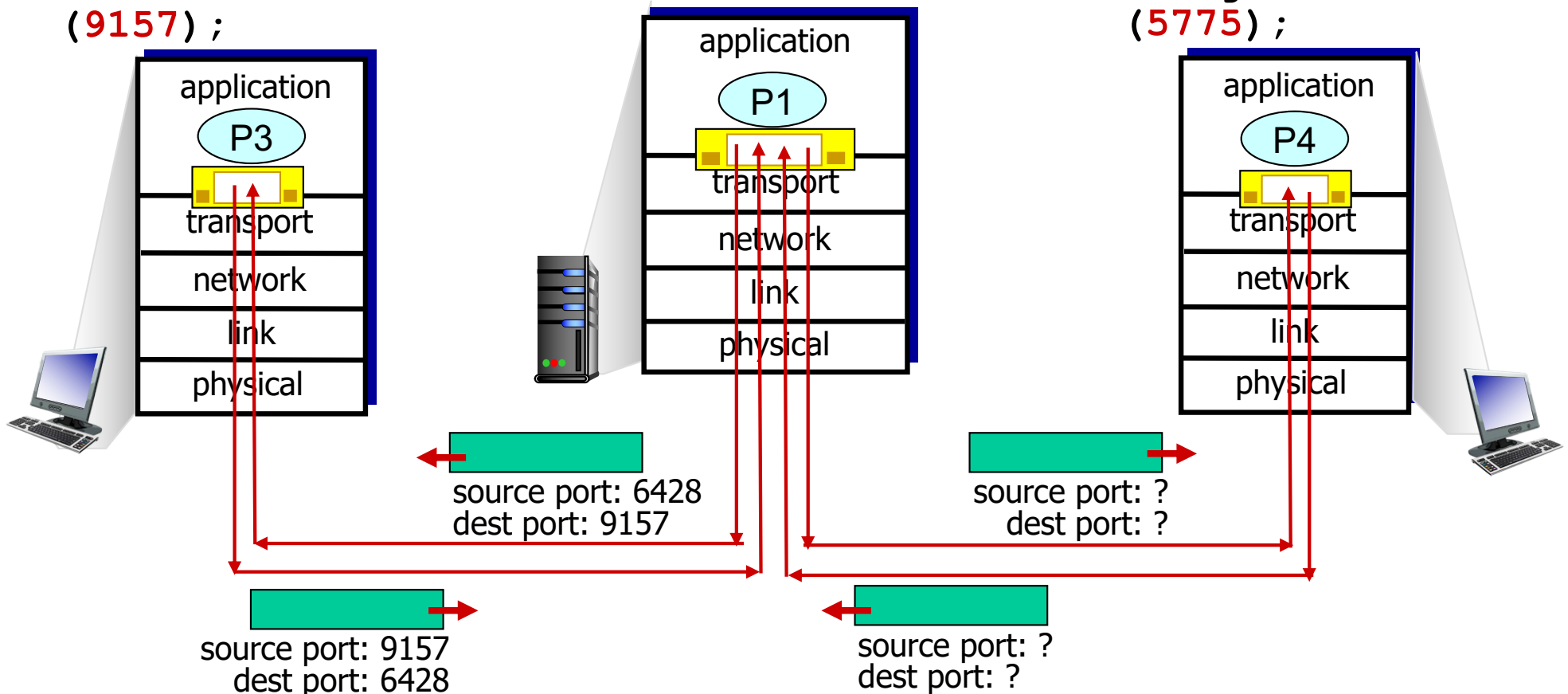
IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

Connectionless demux: example

```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```

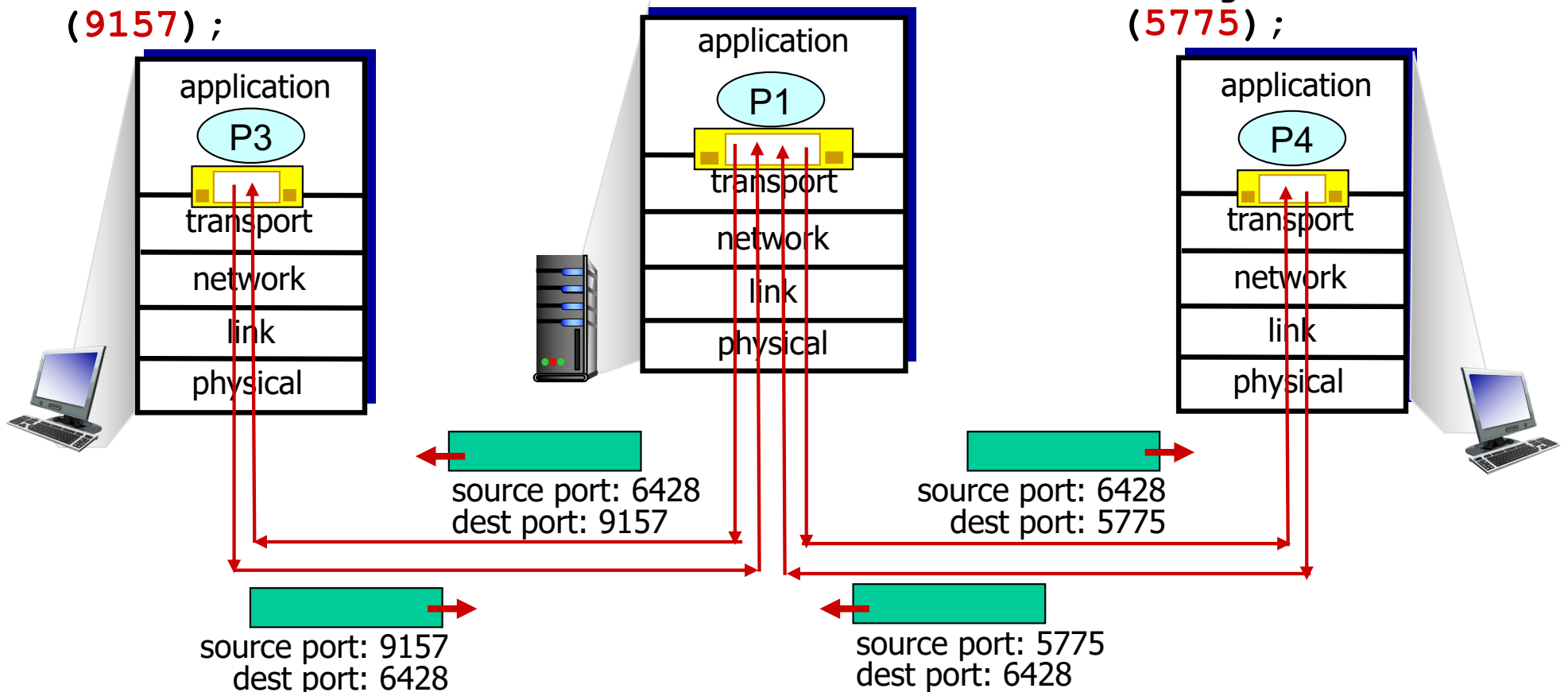


Connectionless demux: example

```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

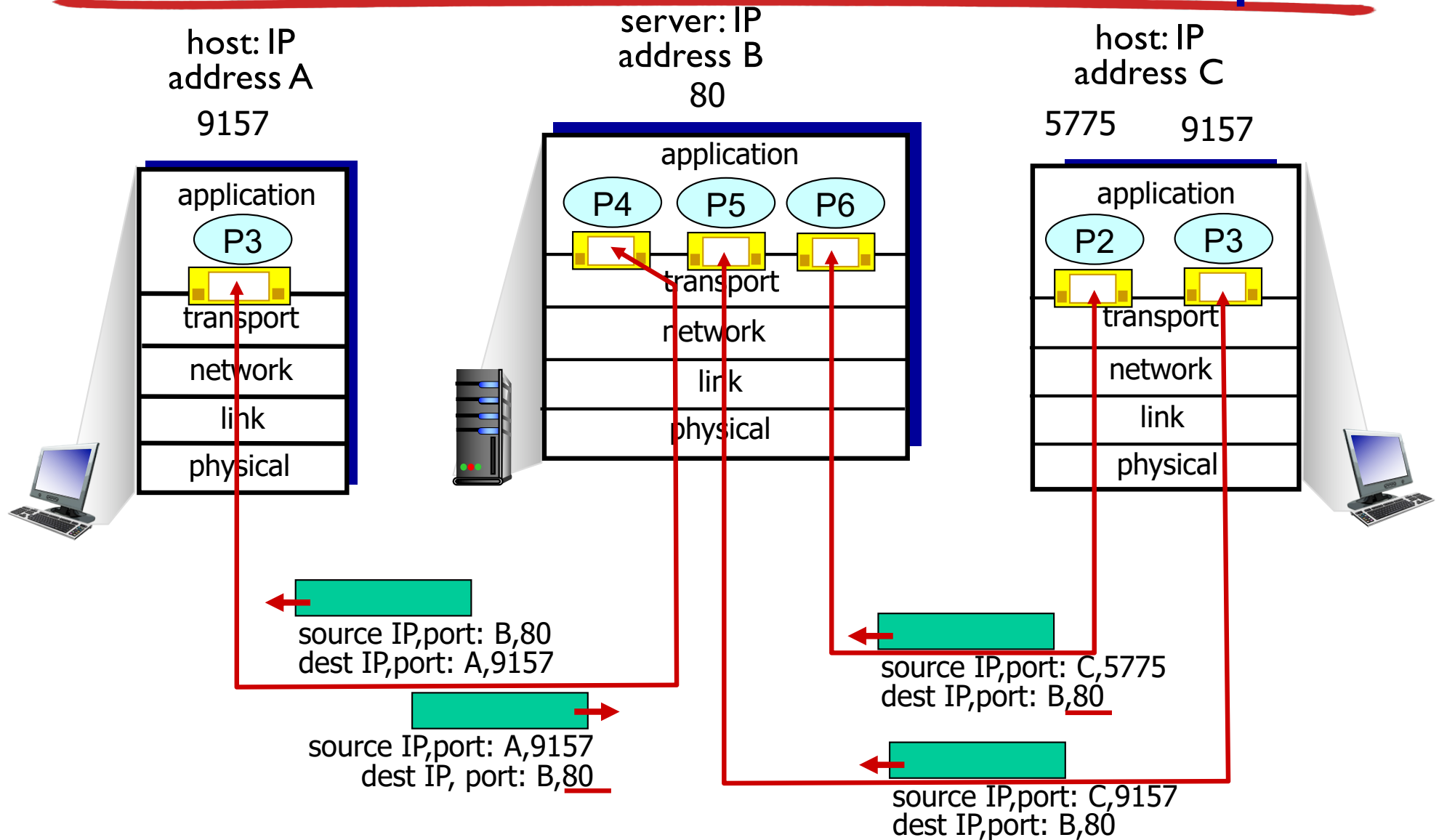
```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```



Connection-oriented demux

- ❖ TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❖ demux: receiver uses all four values to direct segment to appropriate socket
- ❖ server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple

Connection-oriented demux: example



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Chapter 3 outline

3.1 transport-layer
services

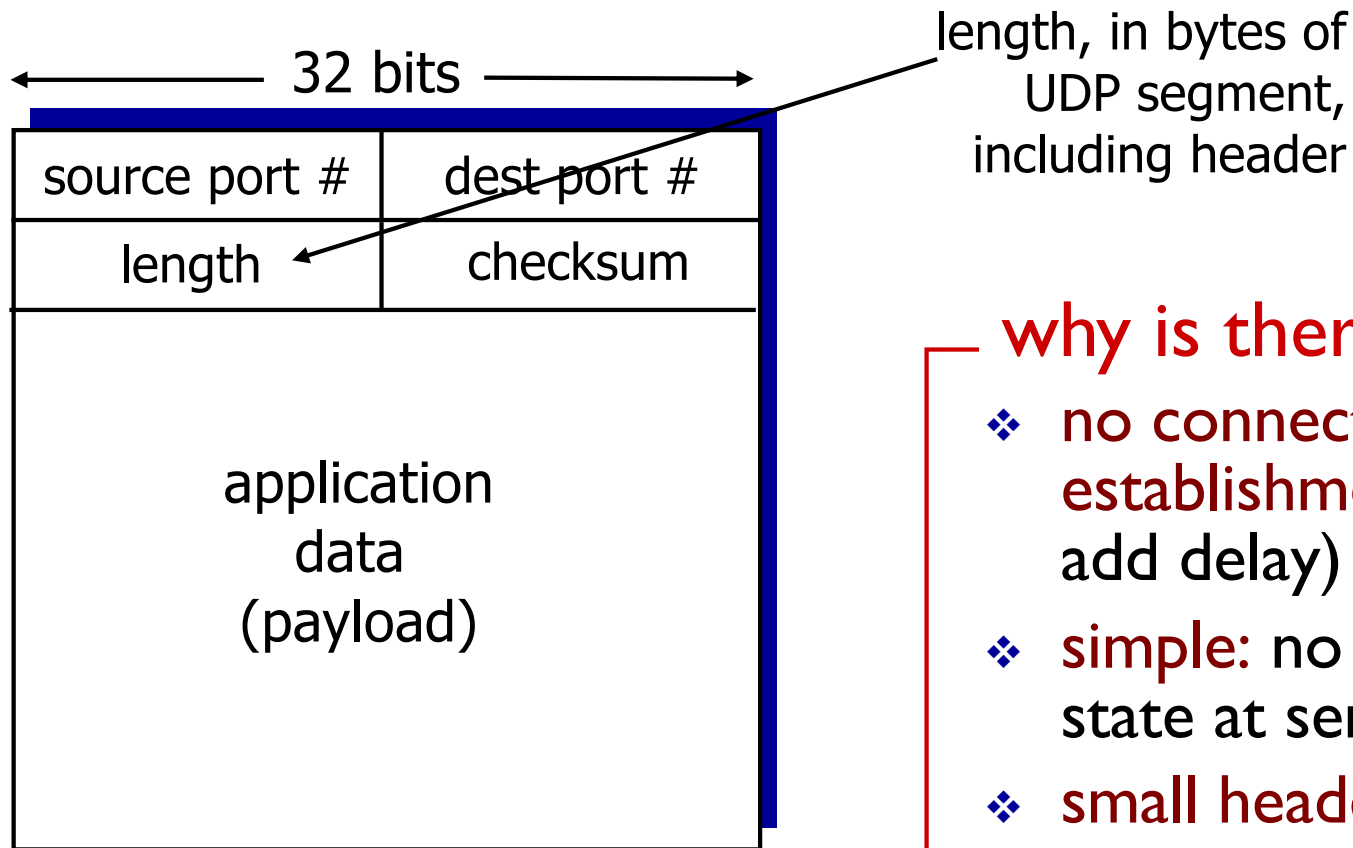
3.2 multiplexing and
demultiplexing

3.3 connectionless
transport: UDP

UDP: User Datagram Protocol [RFC 768]

- ❖ “best effort” service, UDP segments may be:
 - lost
 - delivered out-of-order to app
- ❖ *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others
- ❖ **UDP use:**
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
- ❖ **reliable transfer over UDP:**
 - add reliability at application layer
 - application-specific error recovery!

UDP: segment header (8 bytes header)



why is there a UDP?

- ❖ no connection establishment (which can add delay)
- ❖ **simple**: no connection state at sender, receiver
- ❖ **small header size**
- ❖ **no congestion control**: UDP can blast away as fast as desired

UDP segment format

UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

sender:

- ❖ treat segment contents, including header fields, as sequence of 16-bit integers
- ❖ **checksum:** addition (one's complement sum) of segment contents
- ❖ **sender puts checksum value into UDP checksum field**

receiver:

- ❖ compute checksum of received segment
- ❖ check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected.
But maybe errors nonetheless? More later
....

Internet 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
<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

Questions?