Lecture 5

The Transport Layer

UDP and TCP

SDU

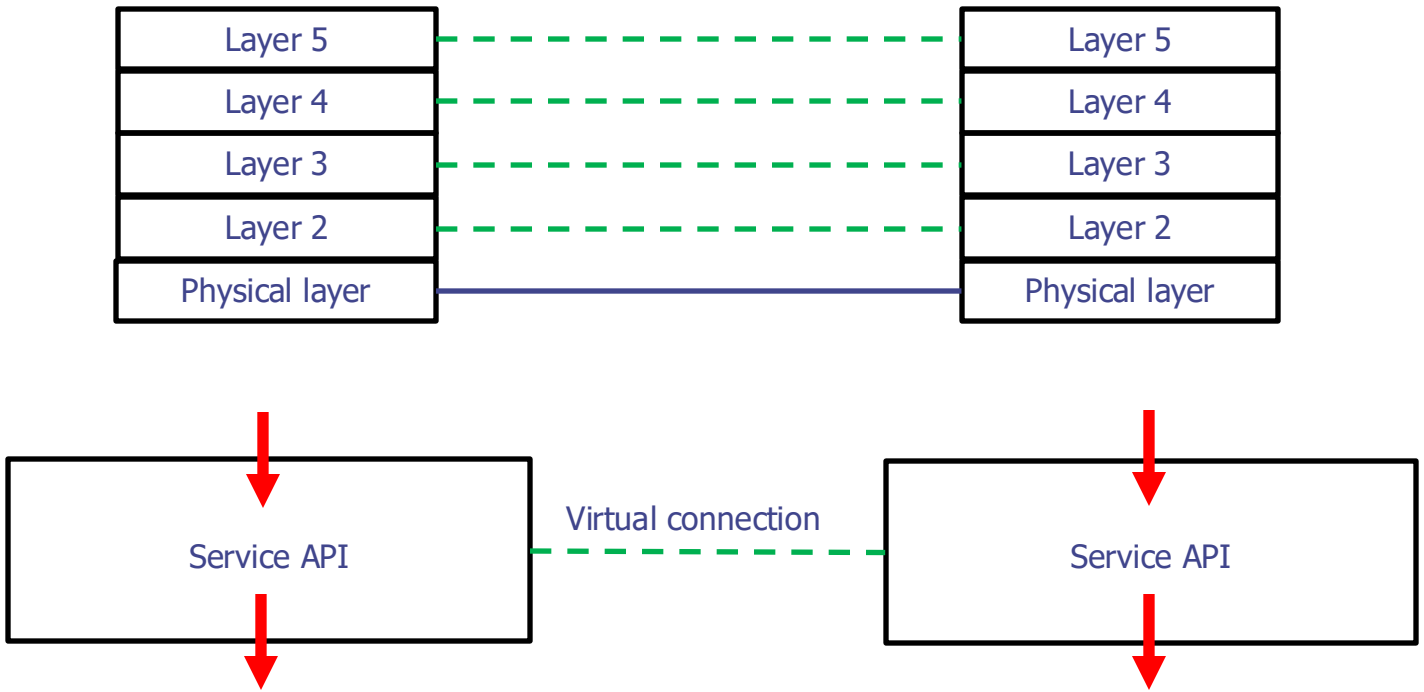
Subjects of today

- The Role of the Transport Layer
- Transport Layer Protocol 1: UDP
- Transport Layer Protocol 2: TCP

SDU

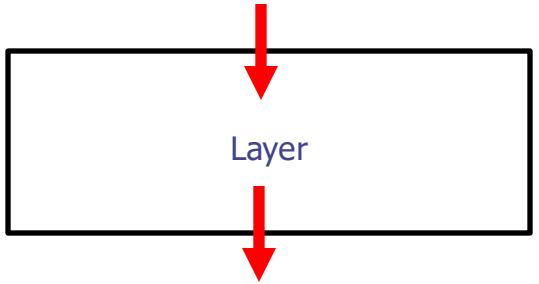
4.1 The Role of the Transport Layer

Layered protocol stack



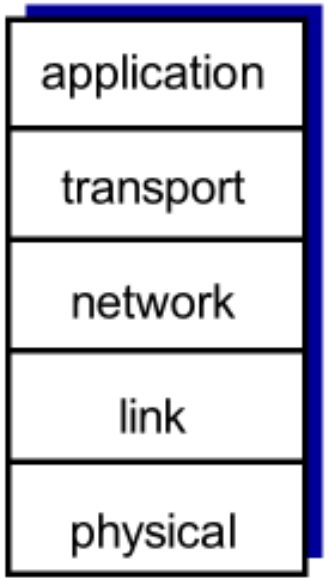
Every layer must…

- … offer services to an upper layer.
- … comply with agreed protocols.
- … utilize services from the underlying layer.



The Internet Protocol Stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"

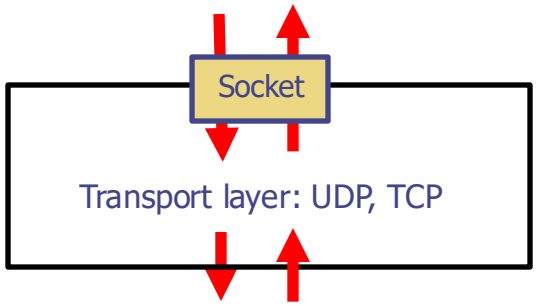


Introduction 1-60

The Transport Layer

- Upper: Retrieves and delivers message through socket.
- Lower: Sends and receives segments to and from remote host.

- Packets at this level is called **Segments**

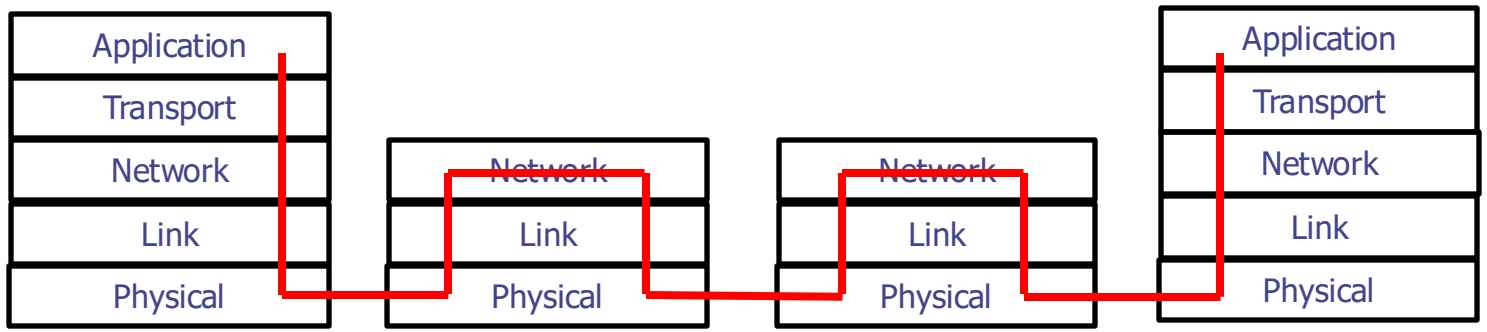


The Transport Layer

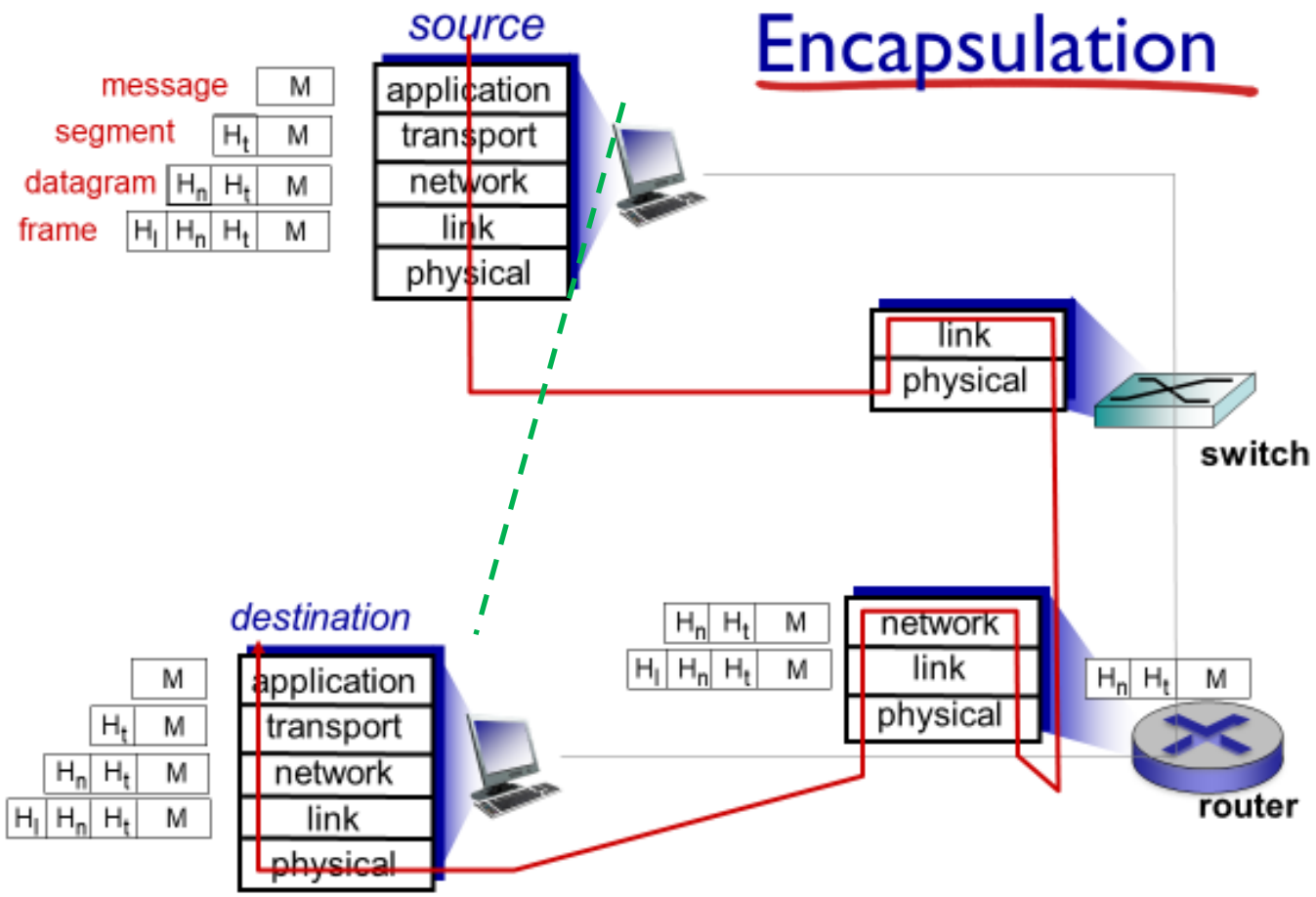
Application view

Transport view

Real view



The Transport Layer



What transport service does an app need?

Reliable data transfer

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

Throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

Security

- encryption, data integrity, …

What could we expect of the transport layer?

- Must have:
  - Breaking messages into segments
  - Multiplexing/demultiplexing
- Connection management
- Reliable data transfer
  - Error detection
  - Error recovery
  - In order delivery
- Timing
- Throughput
  - Flow control
  - Congestion Control
- Security

SDU

Transport vs. Network Layer

Network layer

- Logical communication between hosts

Transport layer

- Logical communication between processes
  - Relies on and enhances network layer services

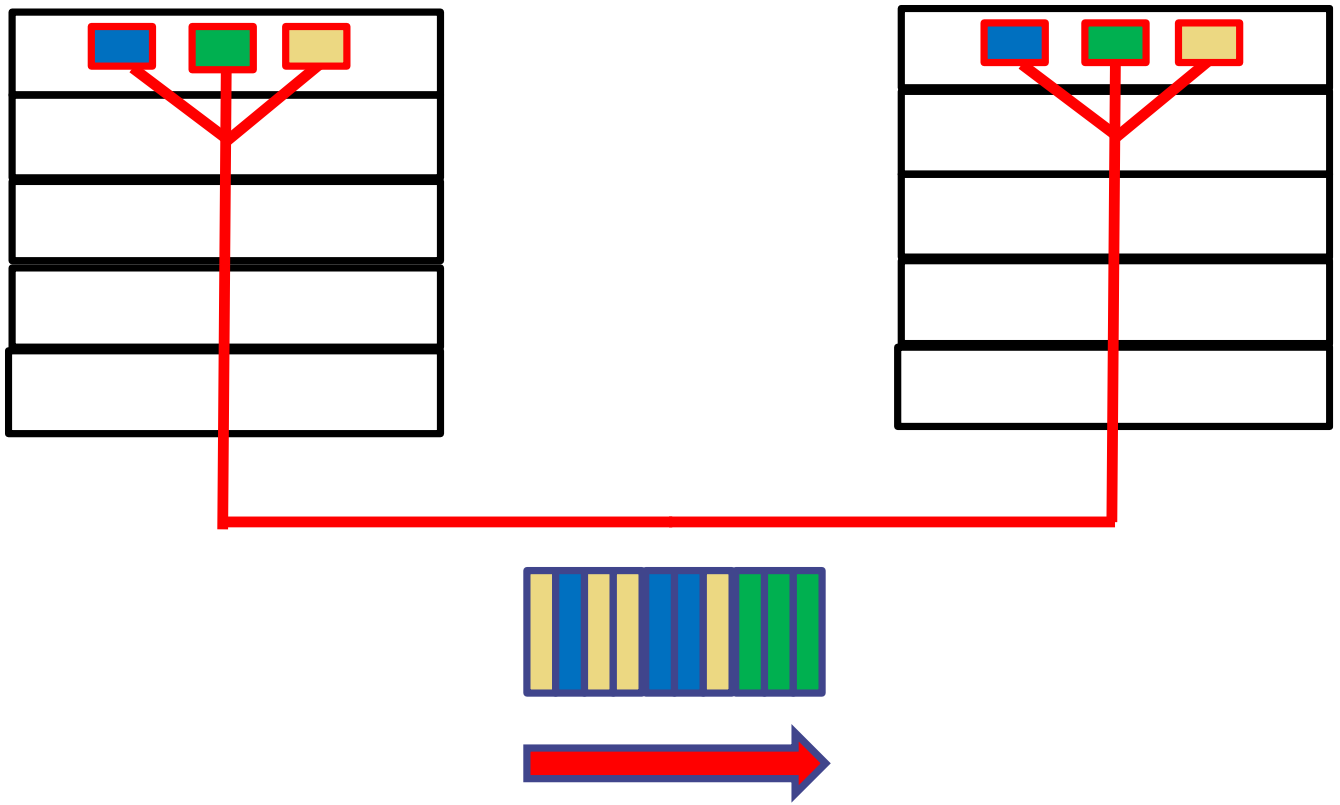
Analogy:

12 kids in Ann's house sending packages to 12 kids in Bill's house:

- Hosts = houses
- Processes = kids
- App messages = Packages in envelopes
- Transport protocol = Ann and Bill who **collects** and **distributes** packages from/to in-house siblings (*mux & demux*)
- Network-layer protocol = Package delivery service

SDU

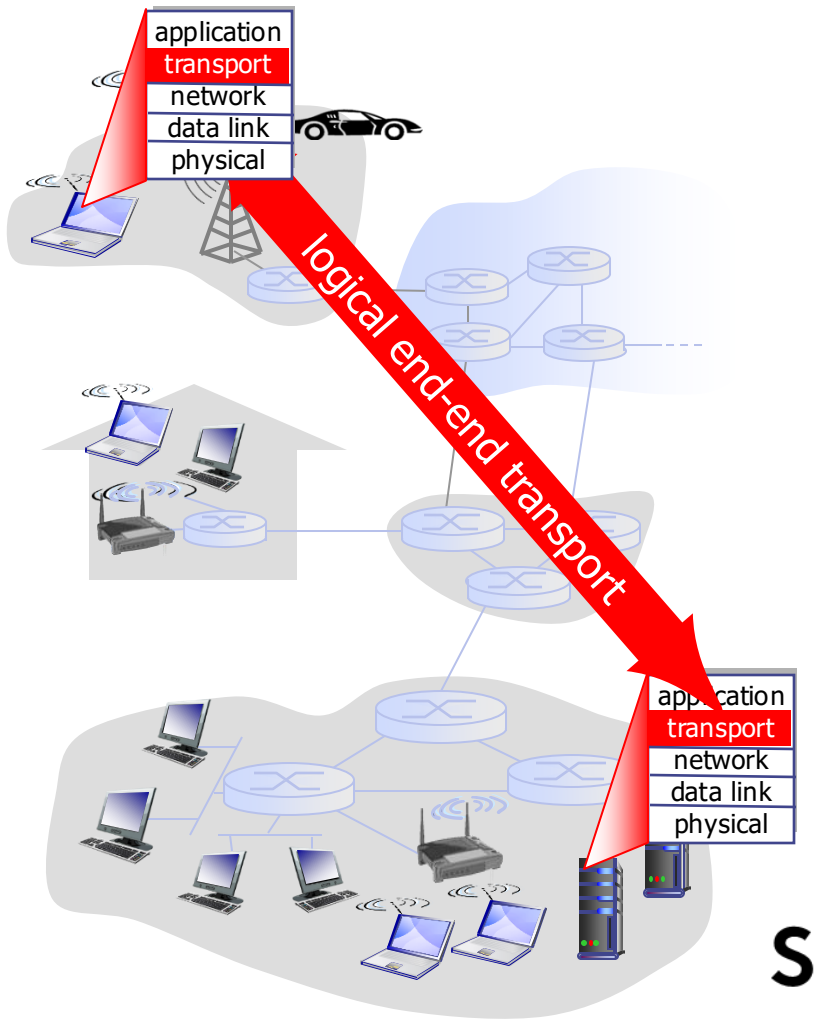
Multiplex/demultiplex



SDU

Summary of The Transport Layer's Role

- Provide *logical communication* between app processes running on different hosts by using:
  - IP addresses
  - Port numbers
- Transport protocols run in end systems
  - Send side: breaks app messages into *segments*, passes to network layer
  - Receiver side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps
  - Internet: TCP and UDP (most common)



4.2 Transport Layer Protocol 1: UDP

UDP - User Datagram Protocol

- Must have:
  - Breaking messages into segments: Yes
  - Multiplexing/demultiplexing: Yes
- Connection management: Connectionless communication
- Reliable data transfer: No (“best effort”)
  - Error detection: Checksum
  - Error recovery: None
  - In order delivery: No
- Timing: None
- Throughput: No
  - Flow control: None
  - Congestion Control: None
- Security: None

RFC 768

SDU

Connectionless demultiplexing

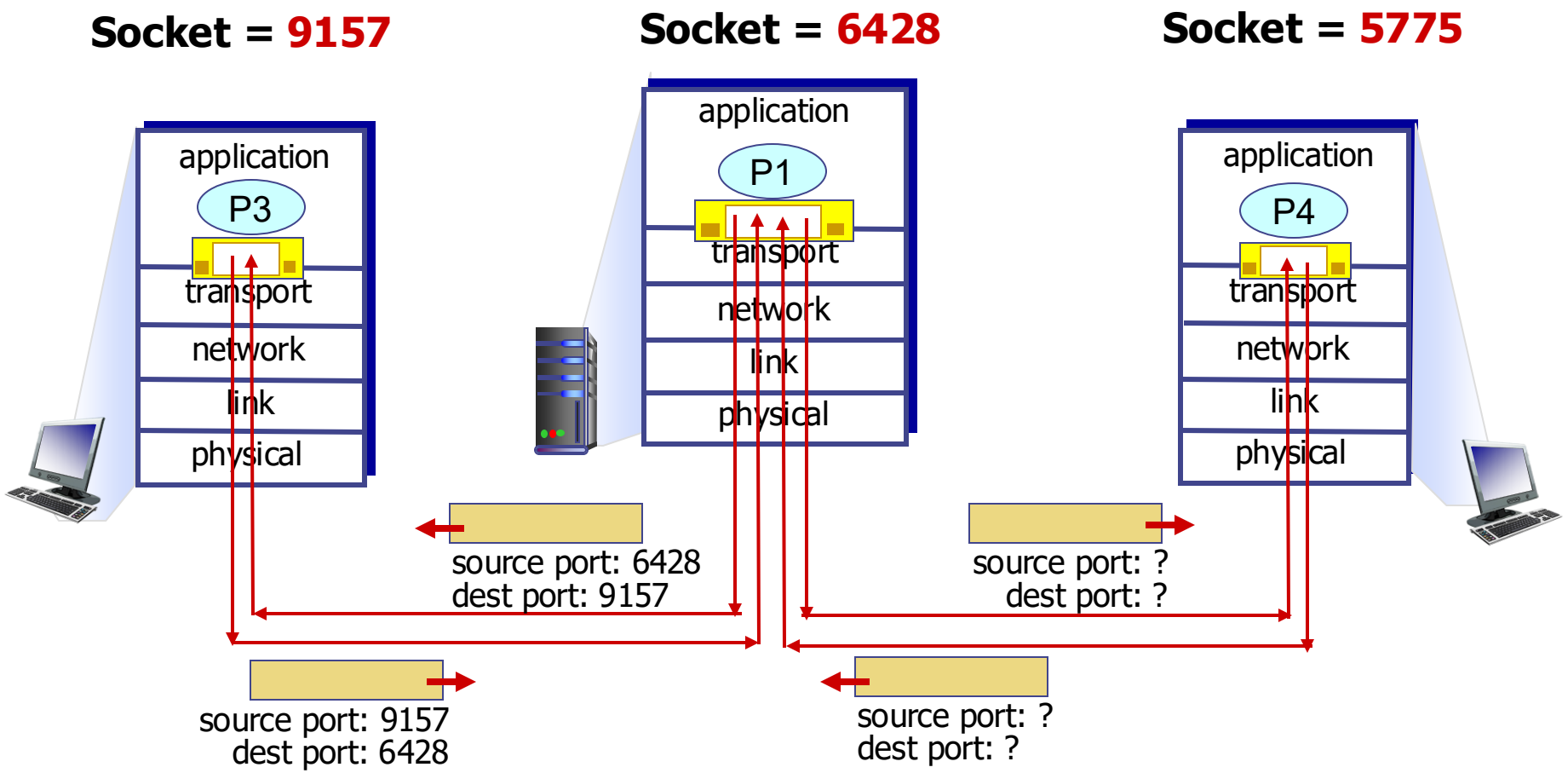
- Created socket has host-local port #
- When creating data packet to send into UDP socket, must specify
  - Destination IP address (host)
  - Destination port # (process)

- When host receives UDP segment:
  - Checks destination port # in segment
  - Directs UDP segment to socket with that port #

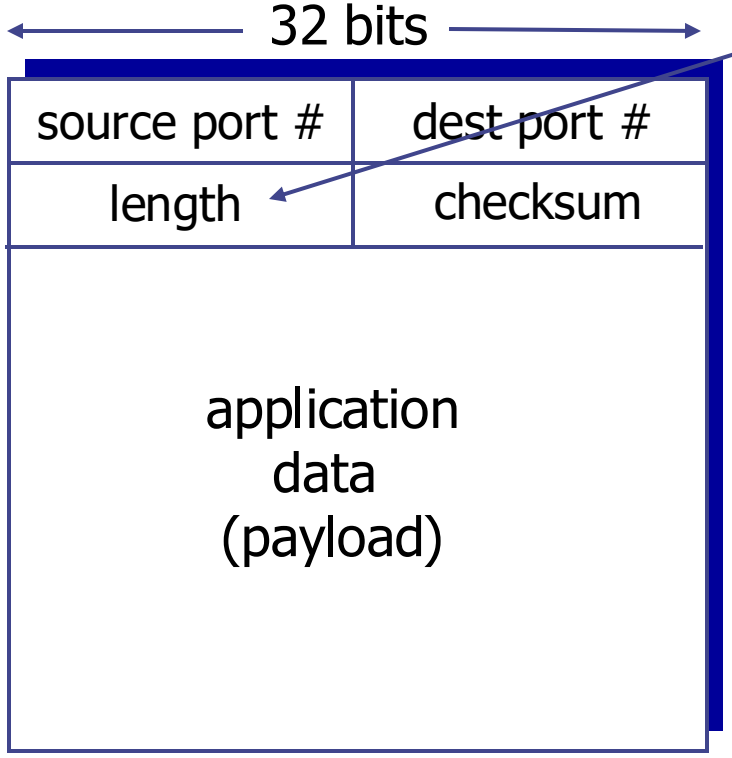
IP datagrams with **same dest. port #**, but different source IP addresses and/or source port numbers will be directed to **same socket** at destination

SDU

Connectionless Demux: Example



UDP: Segment header



UDP segment format

length, in bytes of UDP segment, including 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

Note: Datagram vs. segment

UDP: Checksum

Goal: Detect "errors" in transmitted segment (flipped bits)

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:

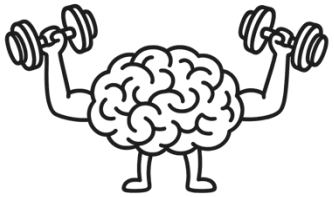
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.

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
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
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



Exercise: 1. Go to: http://gaia.cs.umass.edu/kurose_ross/interactive/
2. Complete the checksum exercise
3. Think about whether there can be errors despite this check

SDU

4.3 Transport Layer Protocol 2: TCP

SDU

Transmission Control Protocol

- Must have:
  - Breaking messages into segments: Yes
  - Multiplexing/demultiplexing: Yes
- Connection management: Connection-oriented communication
- Reliable data transfer: Yes
  - Error detection: Checksum
  - Error recovery: Yes
  - In order delivery: Yes
- Timing: None
- Throughput: No
  - Flow control: Yes
  - Congestion Control: Yes
- Security: None

SDU

TCP: Overview

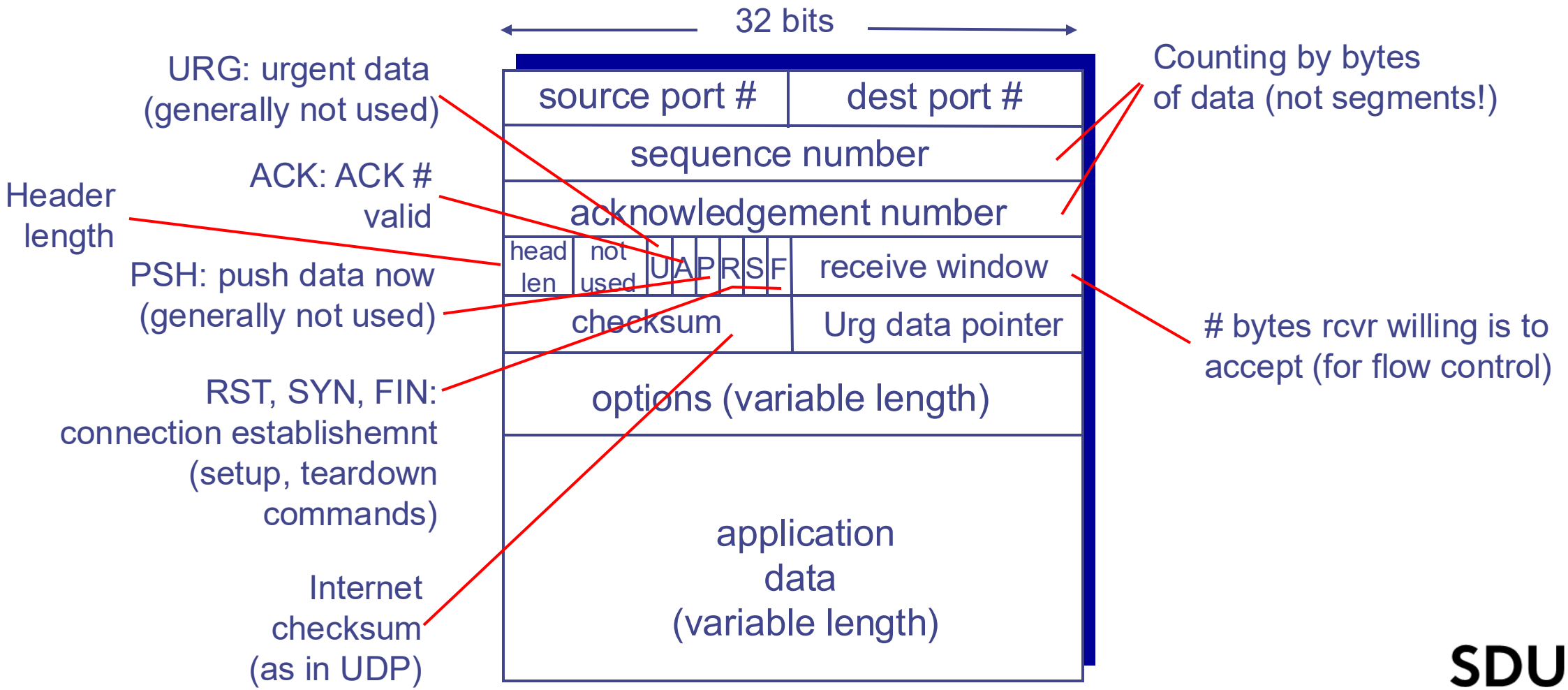
- Point-to-point:
  - one sender, one receiver
- Reliable, in-order *byte steam:*
  - No "message boundaries"
- Pipelined:
  - Does not have to wait to send next packet

- Full duplex data:
  - Bi-directional data flow in same connection
  - MSS: Maximum segment size
- Connection-oriented:
  - Handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- Flow controlled:
  - Sender will not overwhelm receiver

RFCs: 793 (main, but several others)

SDU

TCP segment structure



TCP Seq. numbers, ACKs

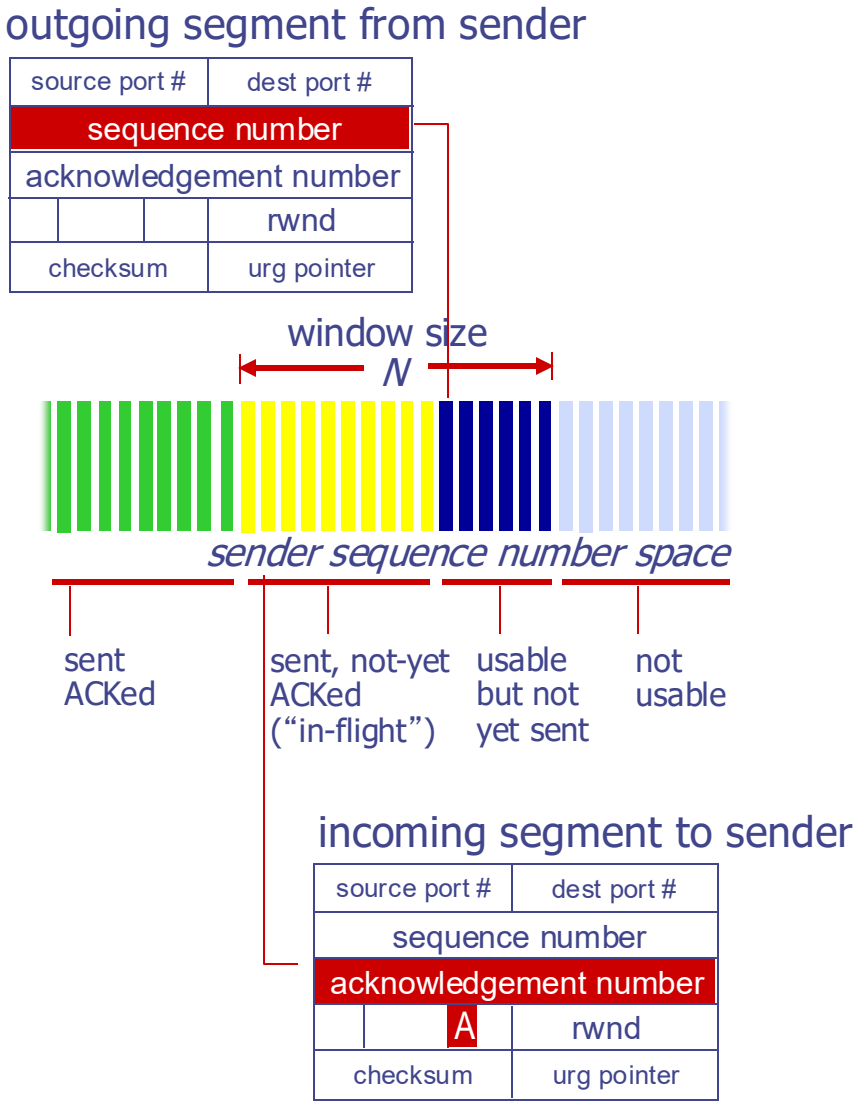
Sequence numbers:

- Byte stream "number" of first byte in segment's data

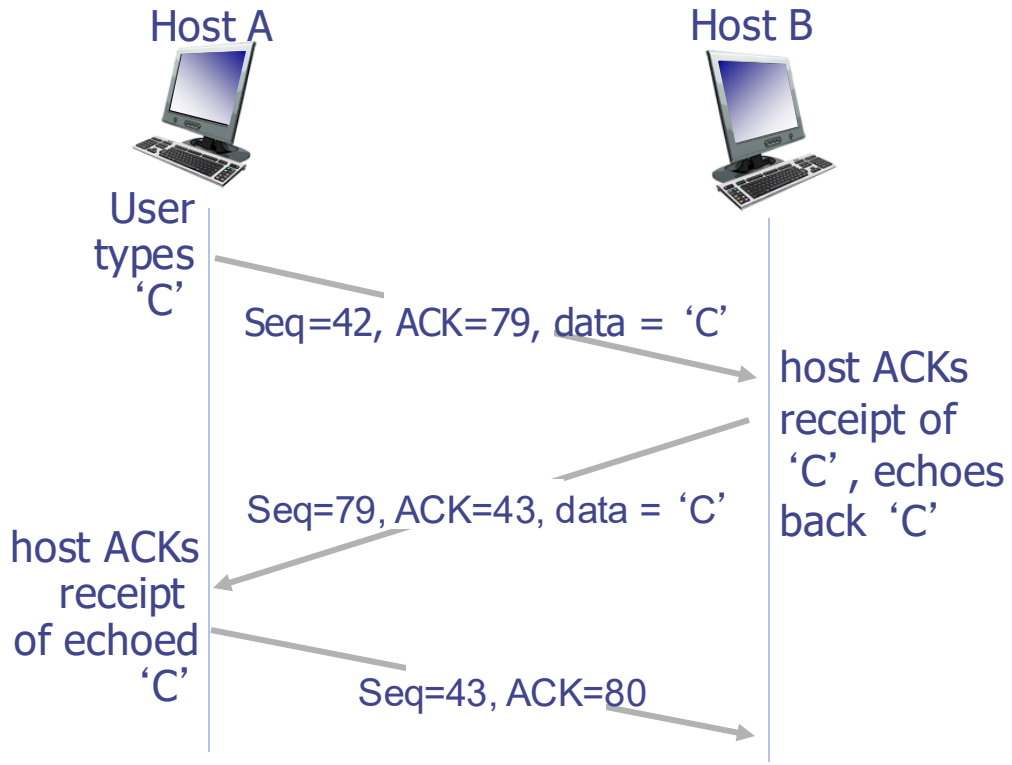
Acknowledgements:

- Seq # of next byte expected from other side
- Cumulative ACK

TCP is stateful!



TCP Seq. numbers, ACKs



simple telnet scenario

Connection-oriented Demux

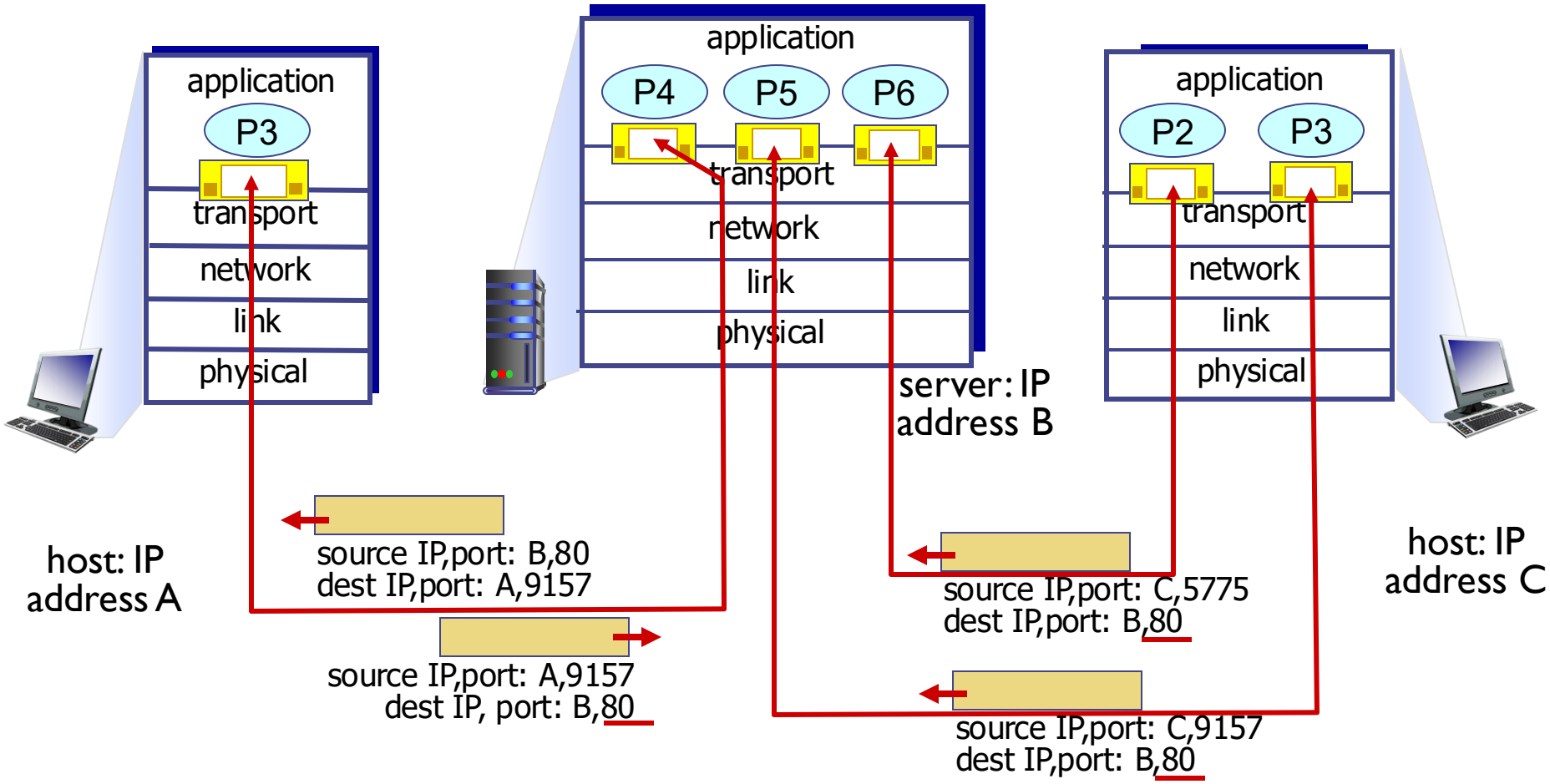
- TCP socket identified by 4-tuple:
  - **Source IP address**
  - **Source port number**
  - **Dest IP address**
  - **Dest port number**
- Demux: receiver uses all four values to direct segment to appropriate socket

Purpose:

- Server host may support many simultaneous TCP sockets:
  - Each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - **Note:** Non-persistent HTTP will have different socket for each request

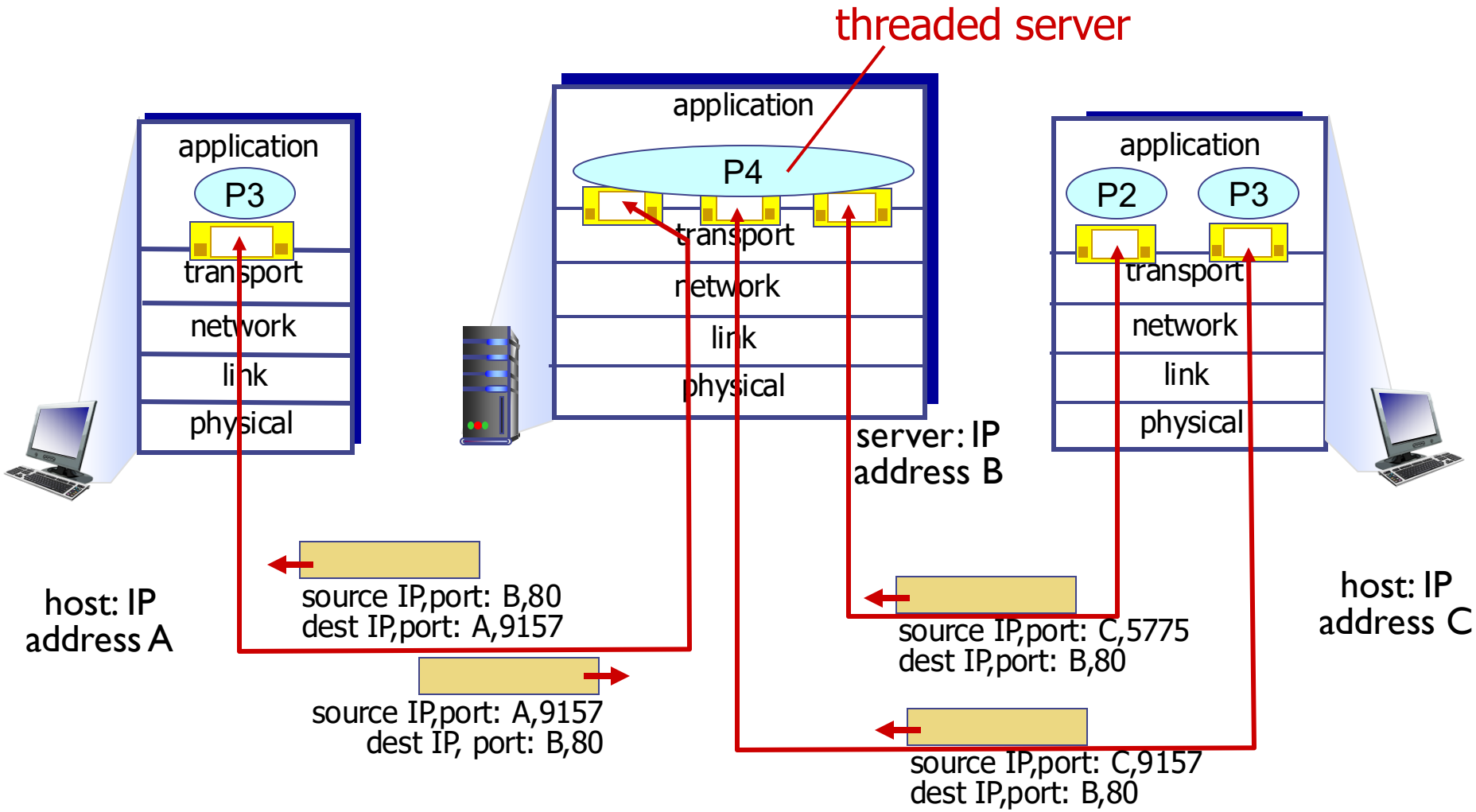
SDU

Connection-oriented demux: example



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

Connection-oriented demux: example



TCP Reliable Data Transfer

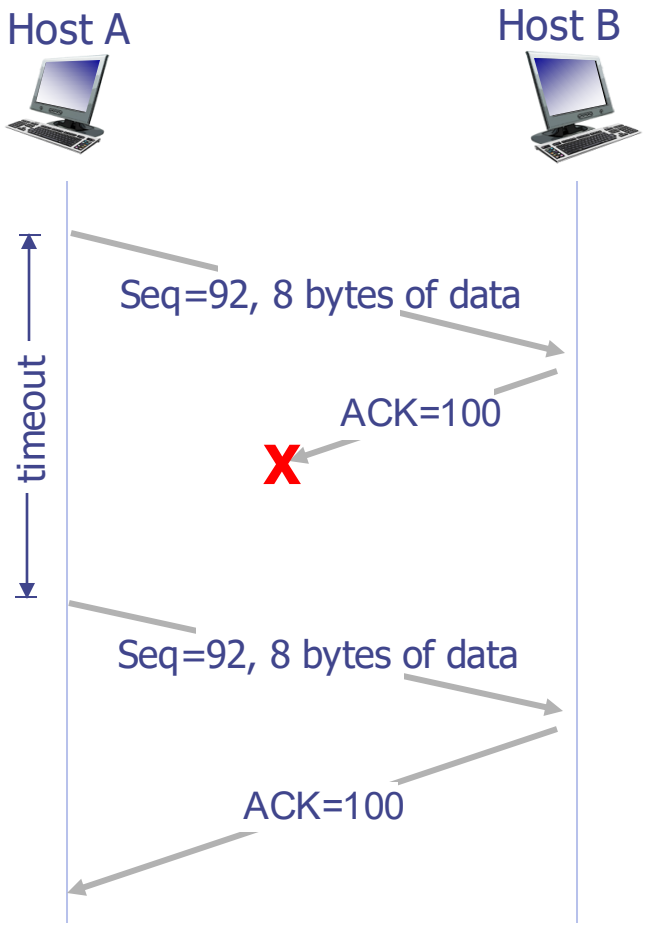
- TCP creates reliable data transfer service on top of IP's unreliable service
  - Pipelined segments
  - Cumulative acks
  - Single retransmission timer
- Retransmissions triggered by:
  - Timeout events
  - Duplicate acks

Consider simplified TCP sender:

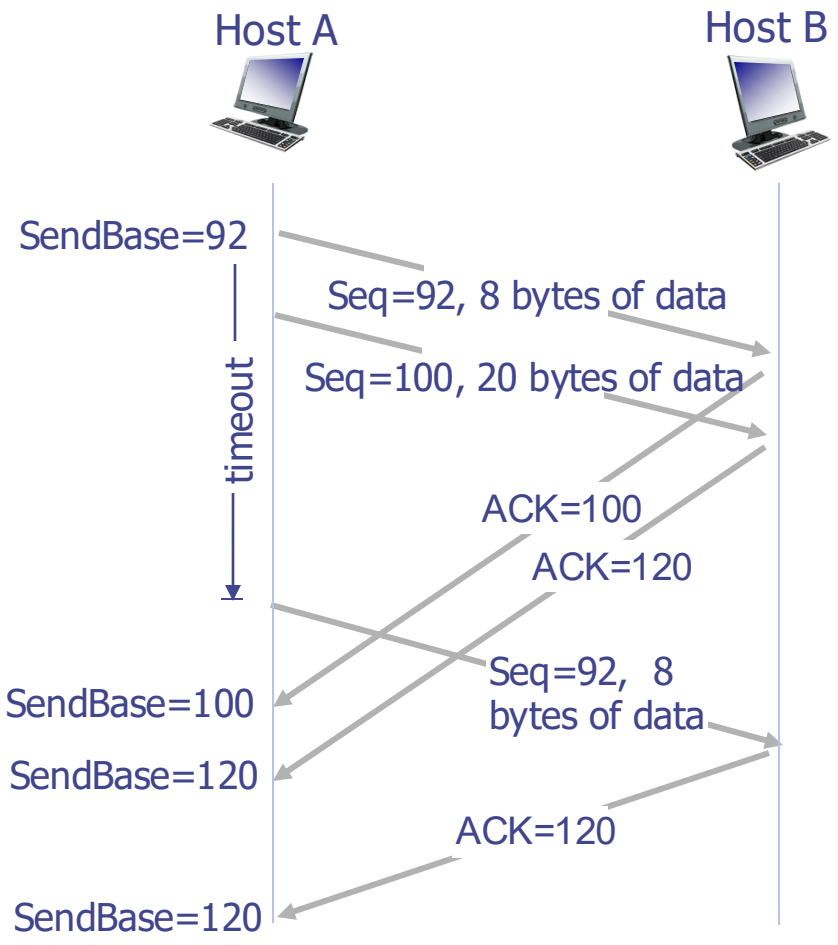
- Ignore duplicate acks
- Ignore flow control, congestion control
- We cover these next time

SDU

TCP: Retransmission Scenarios

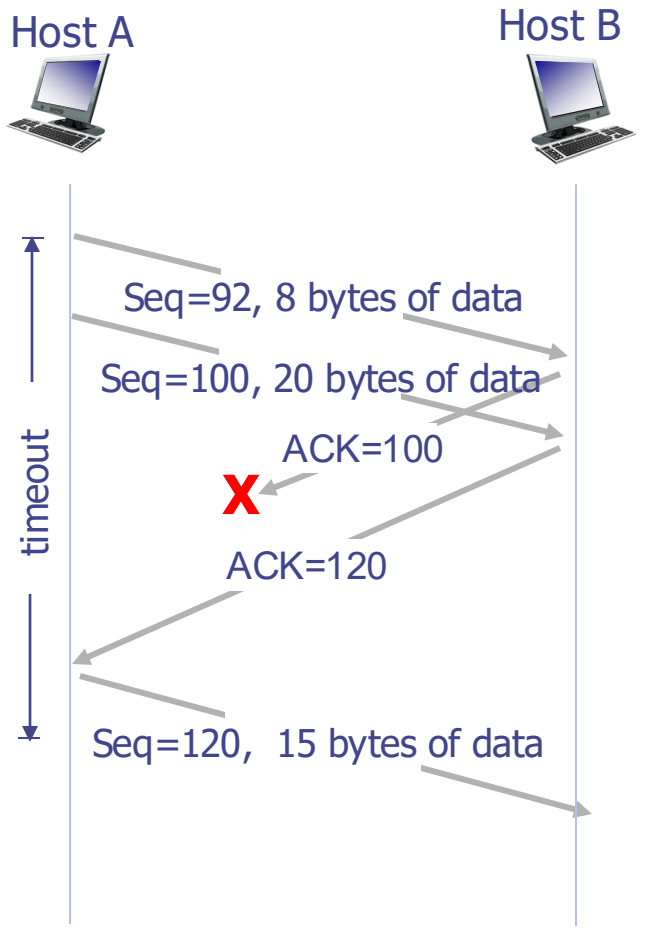


lost ACK scenario



premature timeout

TCP: Retransmission Scenarios



cumulative ACK

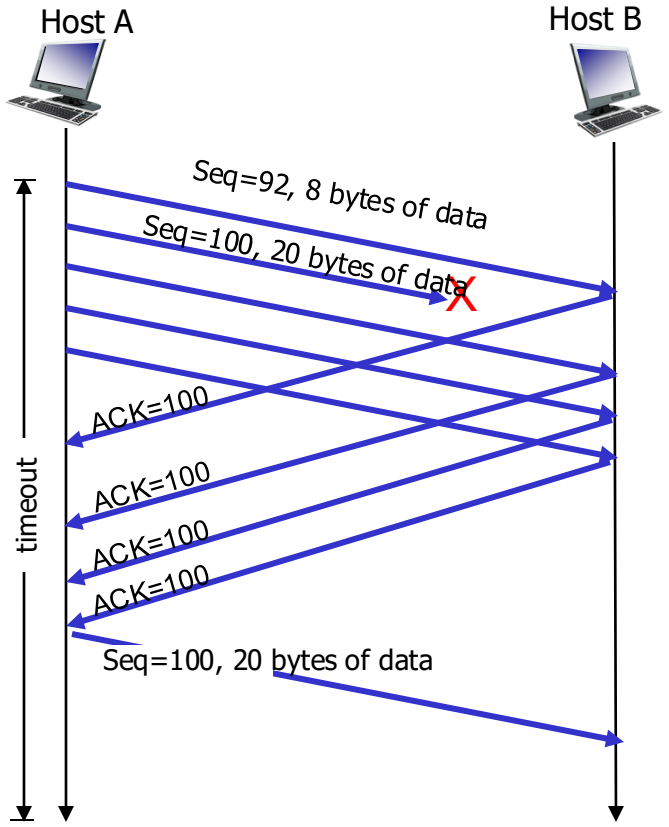
TCP Fast Retransmit

TCP fast retransmit

If sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

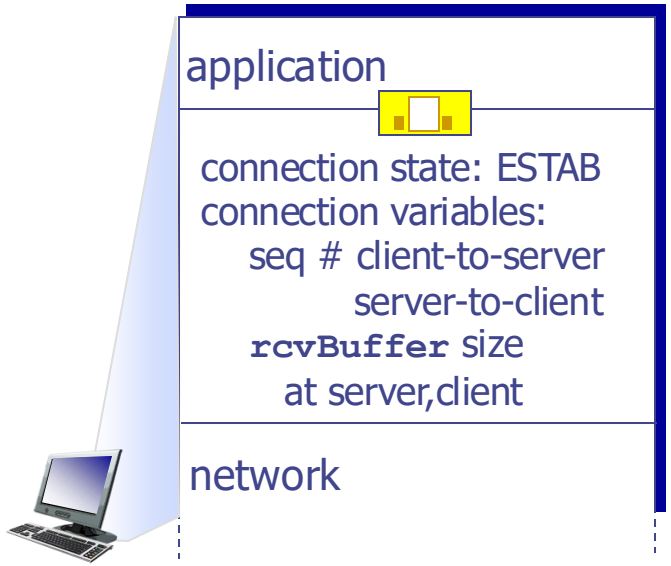
Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

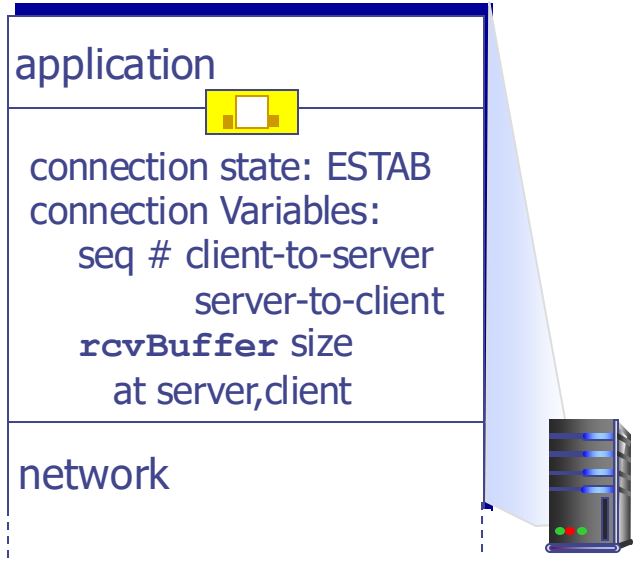


Connection Management

Before exchanging data, sender/receiver "handshake":

- Agree to establish connection (each knowing the other willing to establish connection)
- Agree on connection parameters





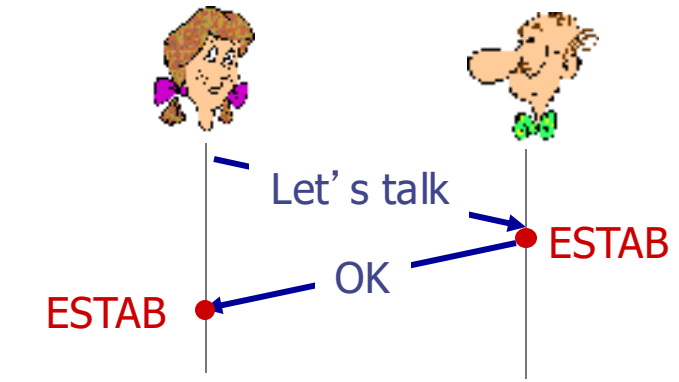

```
Socket clientSocket =
  newSocket("hostname","port number");
```

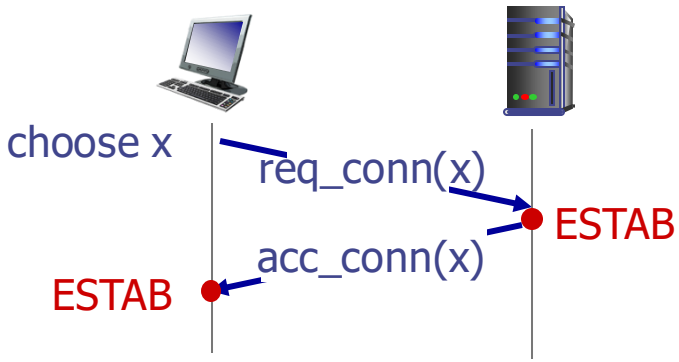
```
Socket connectionSocket =
  welcomeSocket.accept();
```


SDU

Agreeing to establish a connection

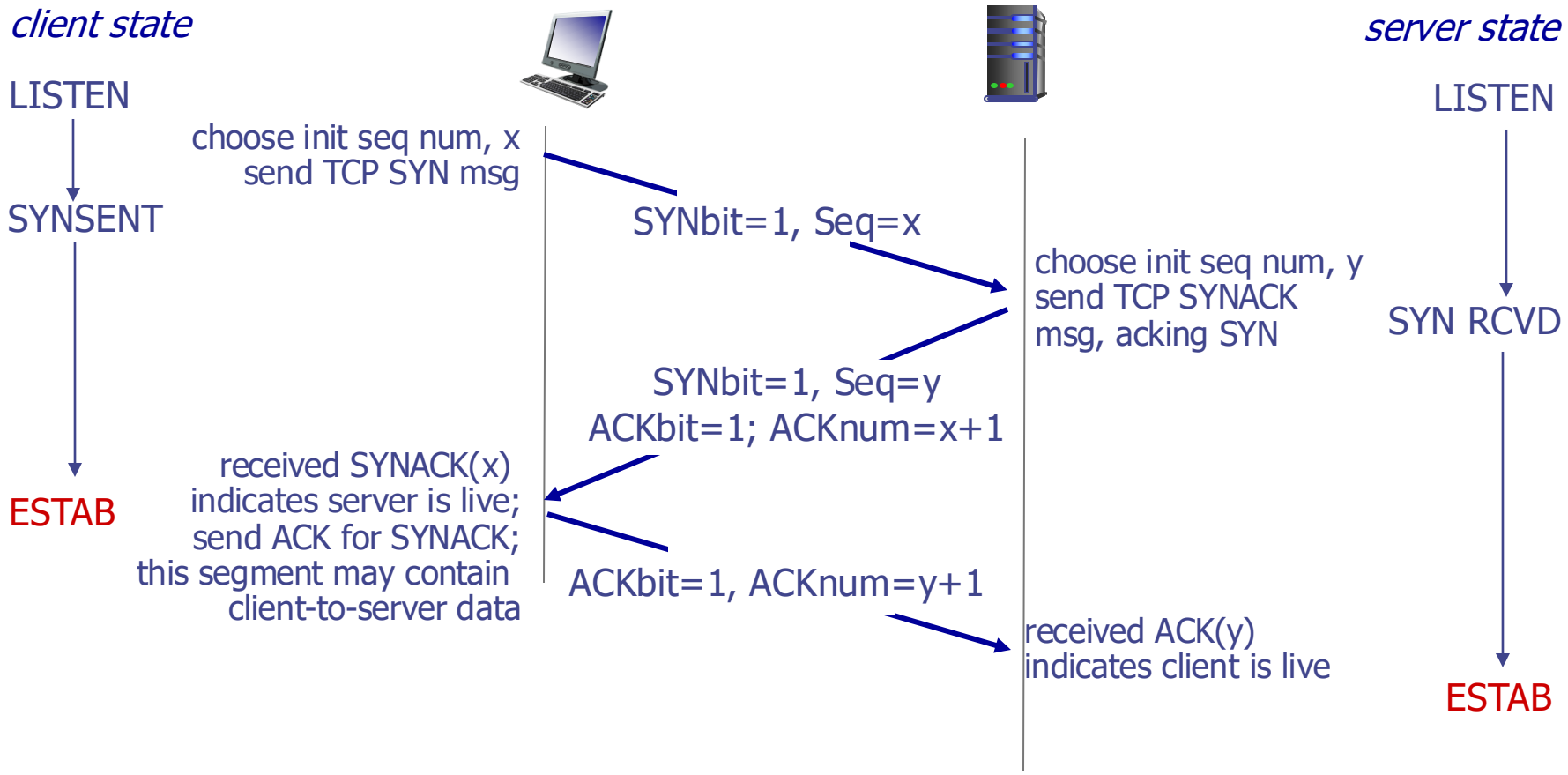
2-way handshake:





Why is this not enough for computer scenario?

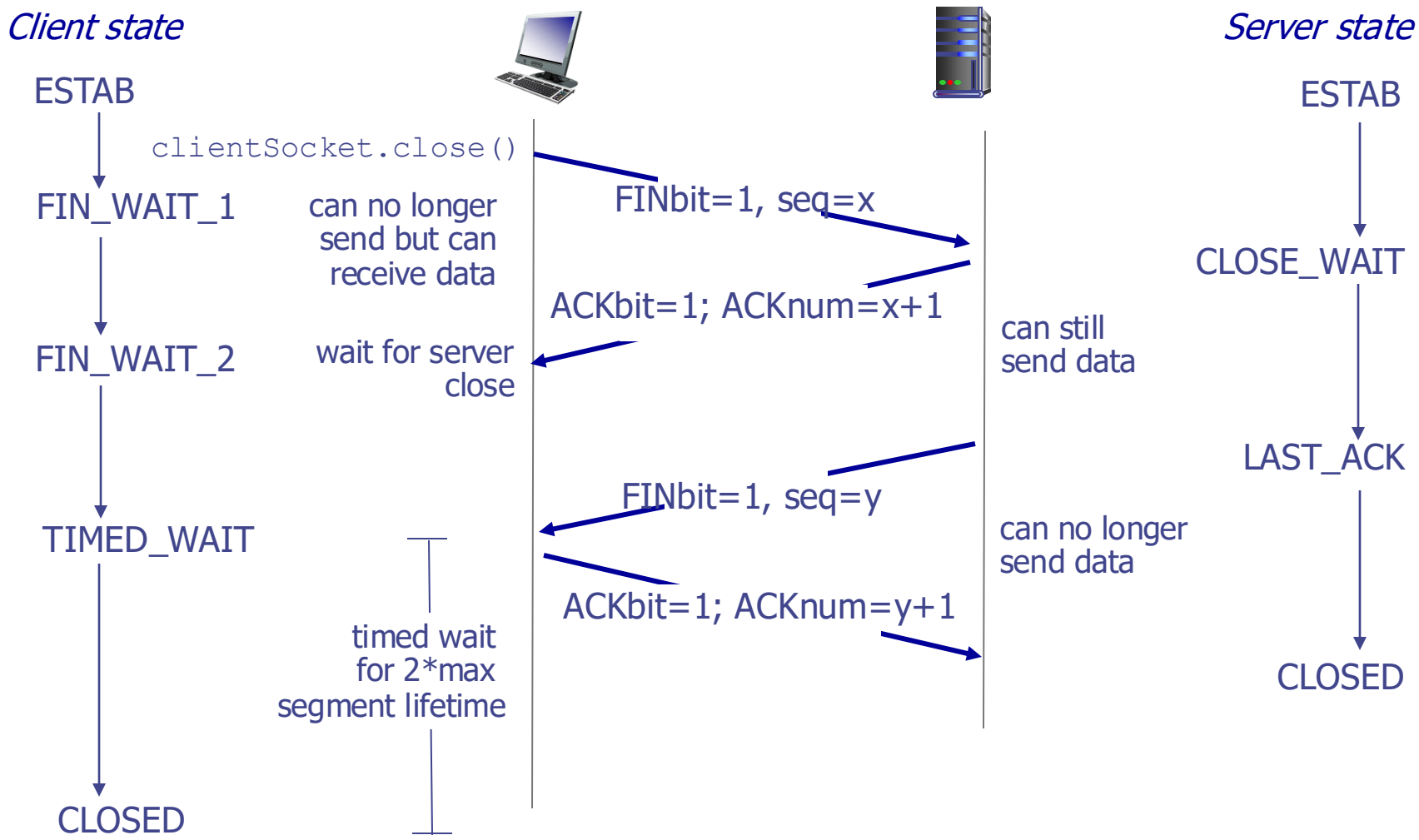
TCP 3-way handshake



TCP: Closing a Connection

- Client, server each close their side of connection
  - Send TCP segment with FIN bit = 1
- Respond to received FIN with ACK
  - On receiving FIN, ACK can be combined with own FIN
- Simultaneous FIN exchanges can be handled

TCP: Closing a Connection



For next time:

- Read Kurose & Ross. The rest of chapter 3 p286-314
  - Main topics will be flow and congestion control

- Carry the Wireshark Lab 4
- Finish the Python Lab 1
  - Last time to ask questions is next time!