



Inspiring Excellence

Introduction to Transport Layer

Lecture 6 | Part 1 | CSE421 – Computer Networks

Department of Computer Science and Engineering
School of Data & Science

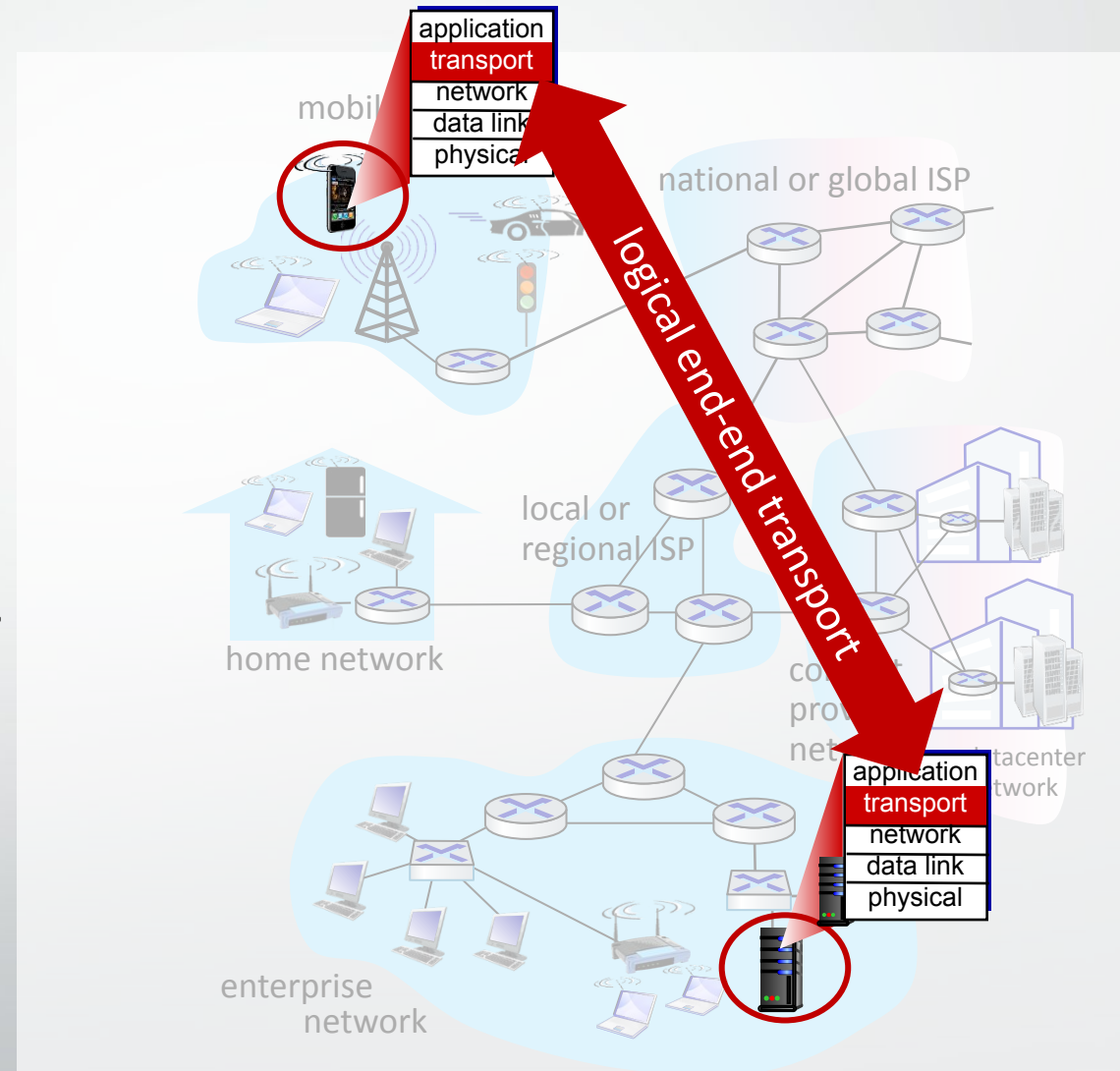
Objectives

our goals:

- understand principles behind transport layer services:
 - reliable data transfer, segmentation, flow control etc..
- learn about Internet transport layer protocols:
 - UDP: connectionless transport
 - TCP: connection-oriented reliable transport

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
 - sender: breaks application messages into *segments*, passes to network layer
 - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
 - TCP, UDP



Transport vs. Network layer

- **network layer:** logical communication between hosts
- **transport layer:** logical communication between processes
 - relies on, enhances, network layer services

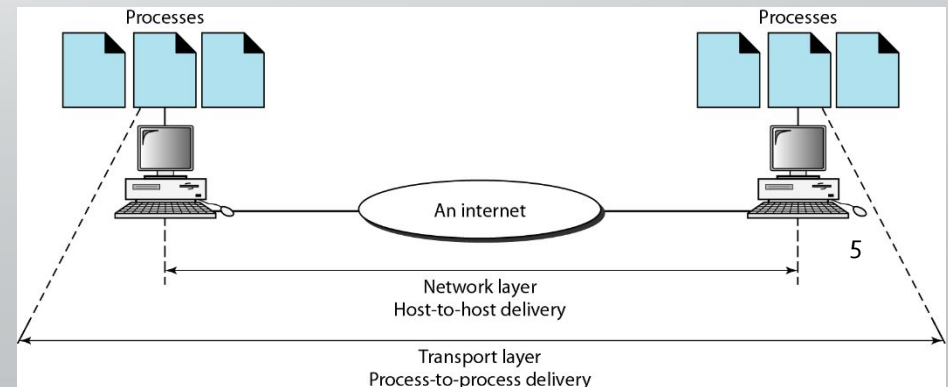
household analogy:

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

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Transport Layer

- The transport layer is responsible for the delivery of a message from one process (sender) to another (receiver).
- Transport Layer PDU is called **Segments**
- Functions:
 - Segmentation and Reassembly
 - Adds Port Address to identify the application
 - Multiplexing
 - Connection establishment and termination
 - Flow and Error Control



Purpose of the Transport Layer

- Primary responsibilities:
 - Segmenting the data and managing each piece.
 - Reassembling the segments into streams of application data.
 - Identifying the different applications.
 - Multiplexing
 - Initiating **Reliability** on.
 - Performing flow control between end users.
 - Enabling error recovery.

Reliability

- Three **basic operations** of reliability are:
 - Initiating sessions and tracking transmitted data
 - Acknowledging received data
 - Retransmitting any unacknowledged data
- To support these reliability operations, more control data is added in the Layer 4 header.

Different Applications

Different Requirements



- Some applications need their data to be complete with no errors or gaps and they can accept a slight delay to ensure this.
- Some applications can accept occasional errors or gaps in the data but they cannot accept any delay.

Solution : Two transport protocols?

Reliable

TCP

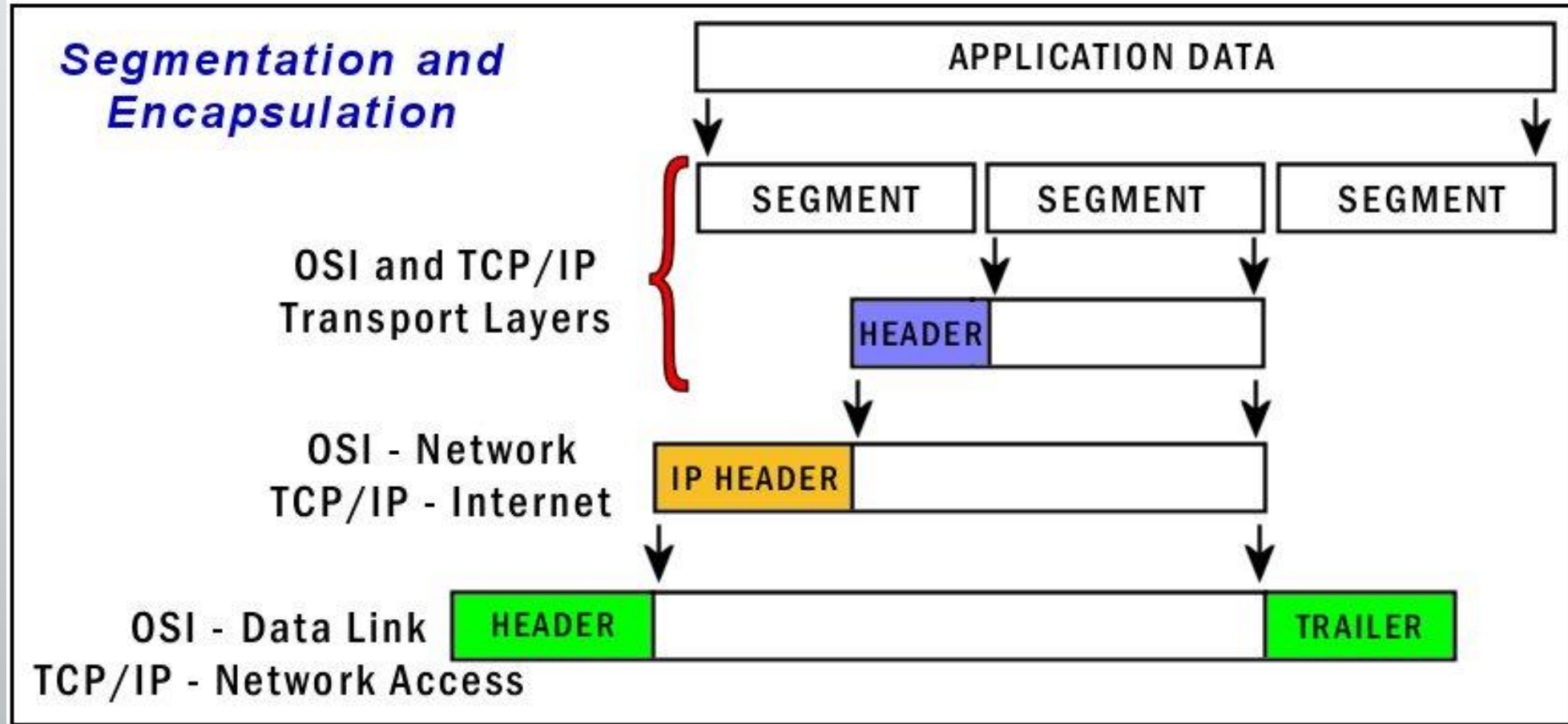
- SMTP/POP (Email)
- HTTP

Fast

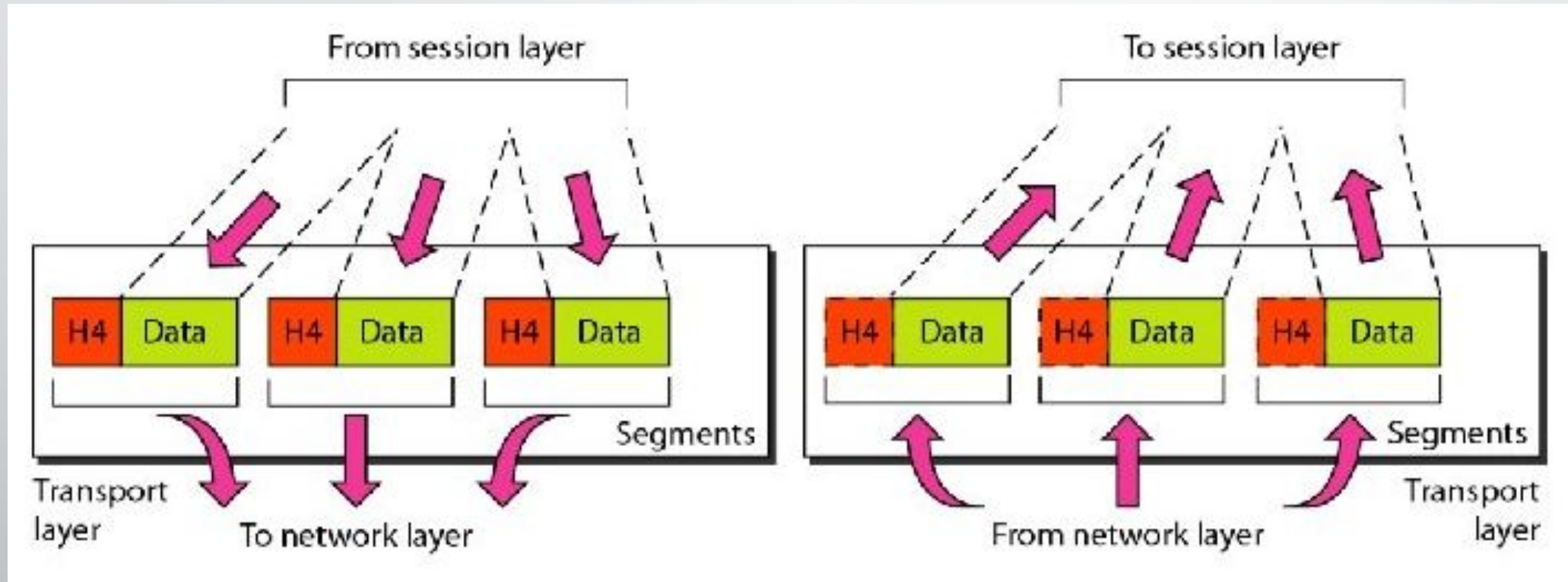
UDP

- IP Telephony
- Streaming Video

Function 1 – Segmentation and Reassembly

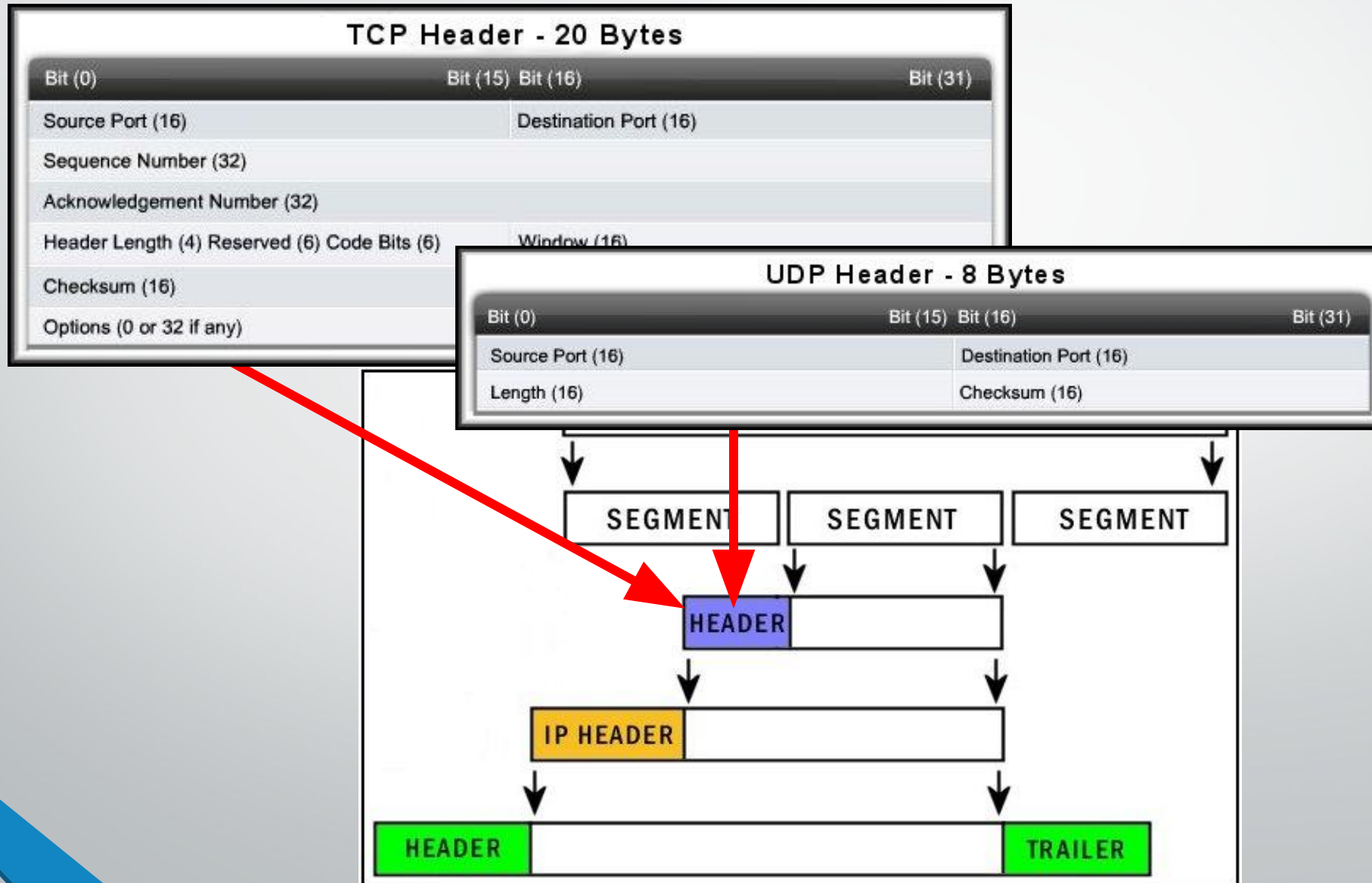


Function 1 – Segmentation and Reassembly

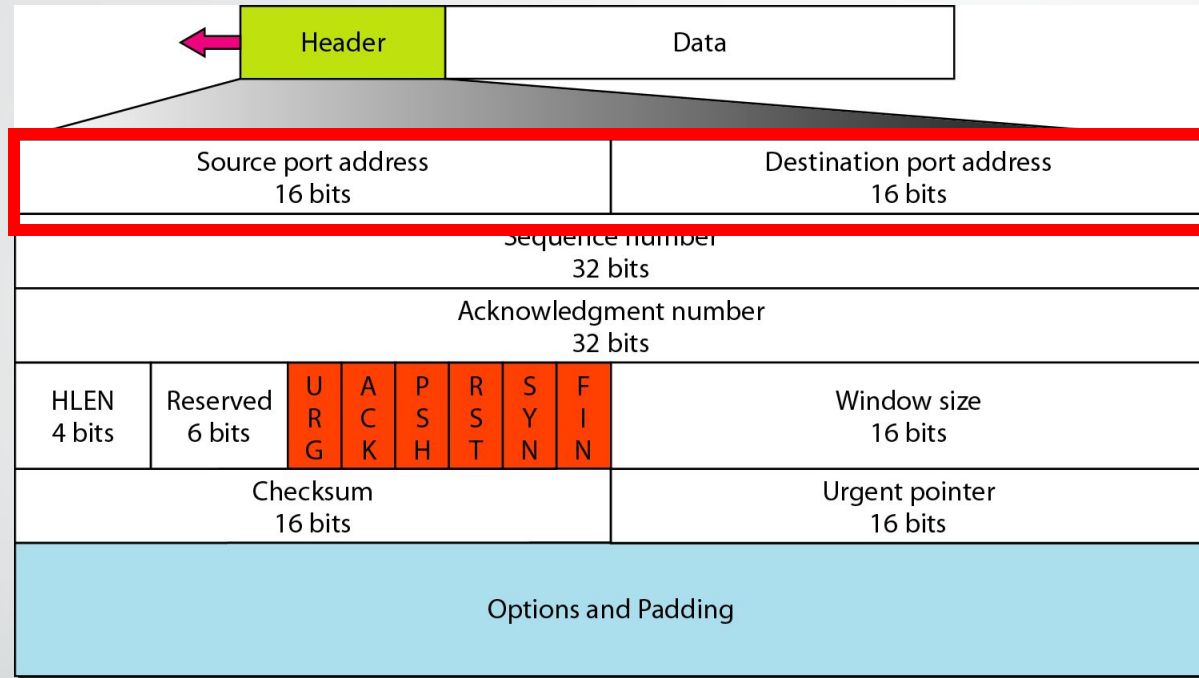


- Also known as encapsulation and de-capsulation.

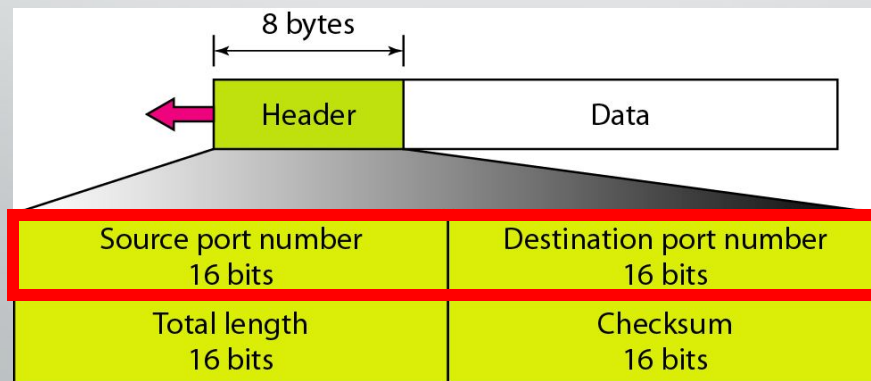
TCP and UDP Headers



TCP and UDP Headers



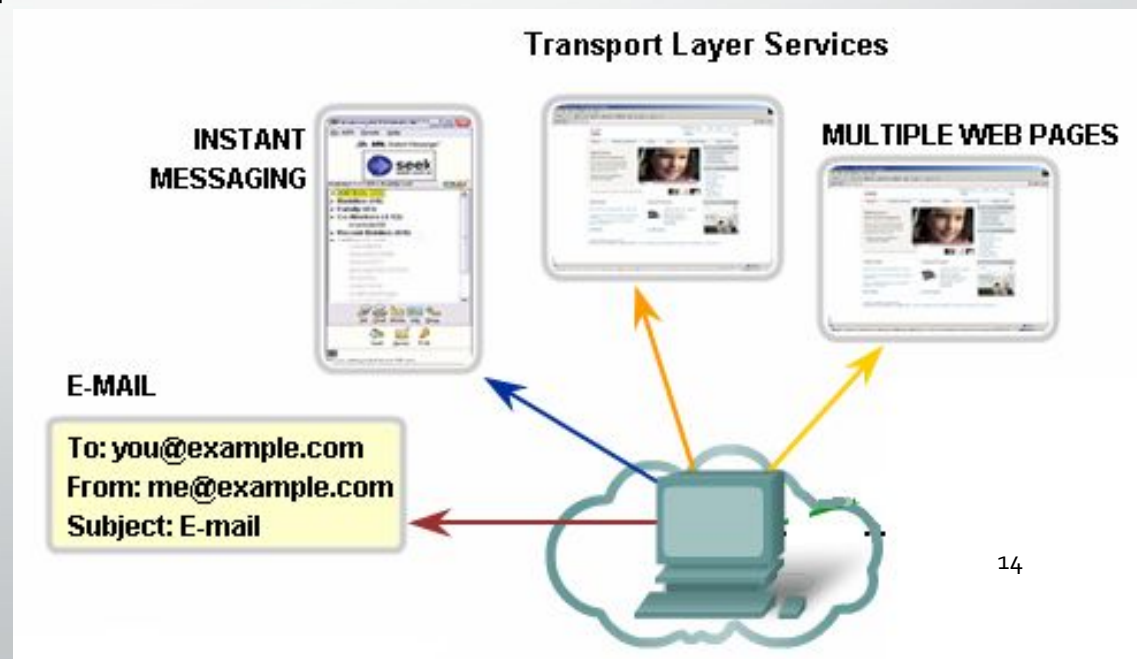
□ TCP Header



□ UDP Header

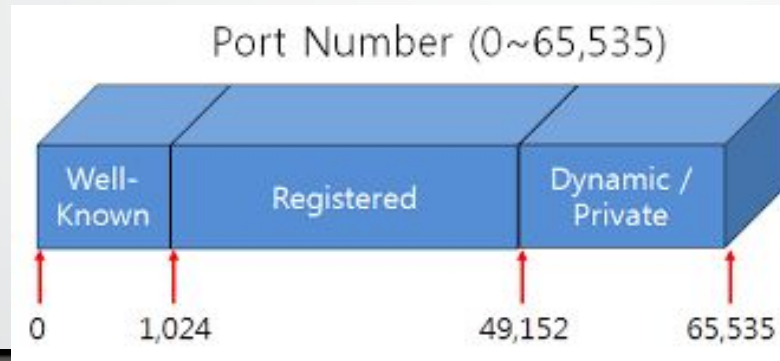
Function2 – Identification Applications Using Port Address

- Port Numbers/Addresses are used to identify different applications/processes running in a computer
- 16-bit in length
 - Represented as one single decimal number
 - Range **0 - 65535**
 - e.g. **80 – Web**; **25 – SMTP**
 - **110 – POP3**, **531 – Instant Messaging**



Port Numbers

- Internet Corporation for Assigned Names and Numbers (ICANN) assigns port numbers.
- **Three** categories:



Port Number Range	Port Group
0 to 1023	Well Known (Contact) Ports
1024 to 49151	Registered Ports
49152 to 65535	Private and/or Dynamic Ports

Port Addressing Types

- Well-Known Ports:
 - Reserved for common services and applications.
 - Pre-assigned by ICANN

Port Number Range	Port Group
0 to 1023	Well Known (Contact) Ports
1024 to 49151	Registered Ports
49152 to 65535	Private and/or Dynamic Ports

20 – FTP Data

25 – SMTP

443 – HTTPS

21 – FTP Control

69 – TFTP

23 – Telnet

110 – POP3

520 – RIP

Port Addressing Types

- Registered Ports:

- Port numbers that companies and other users register with ICANN for use by the applications that communicate using any one of the transport layer protocols.

Port Number Range	Port Group
0 to 1023	Well Known (Contact) Ports
1024 to 49151	Registered Ports
49152 to 65535	Private and/or Dynamic Ports

8008 – Alternate HTTP

1863 – MSN Messenger

8080 – Alternate HTTP

5004 – RTP

5060 – SIP (VoIP)

Port Addressing Types

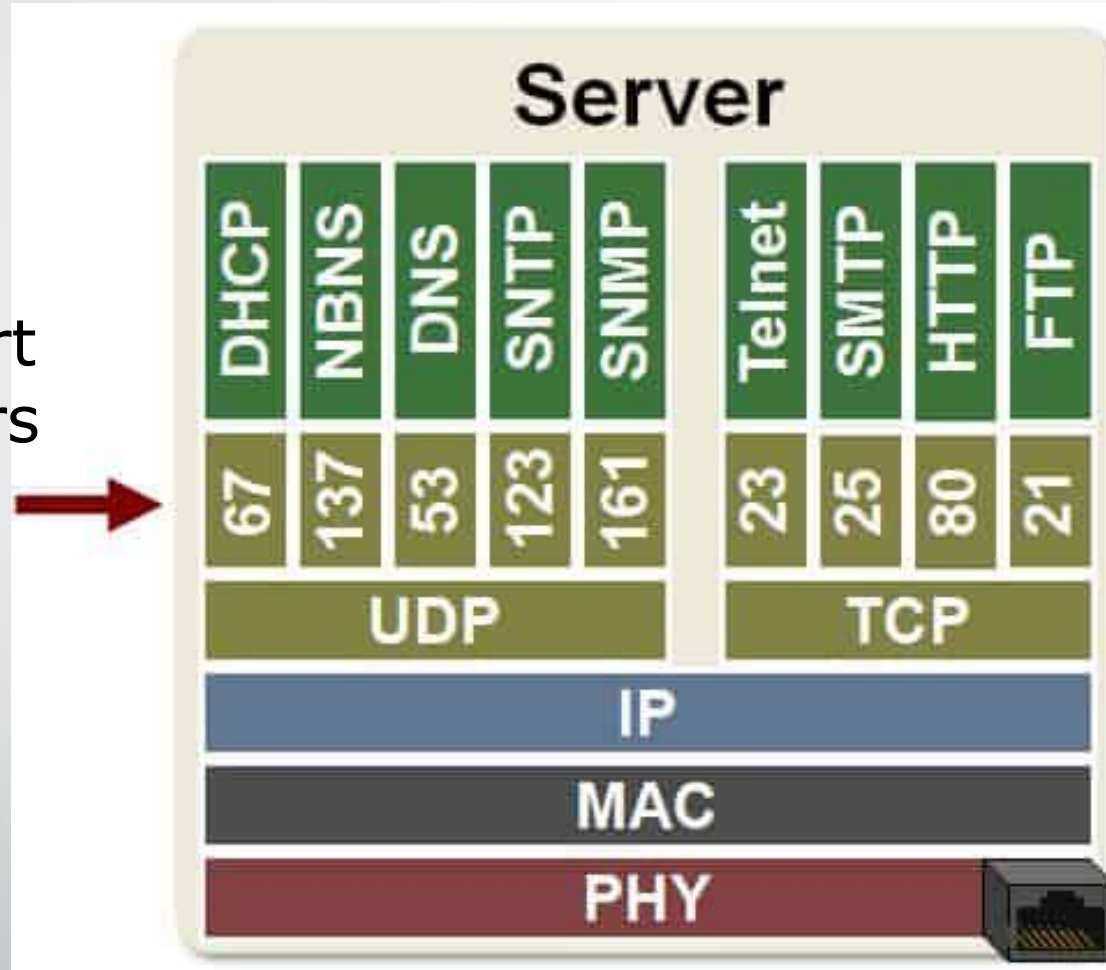
- Dynamic Ports:
 - Assigned to a user application at connect time.

Port Number Range	Port Group
0 to 1023	Well Known (Contact) Ports
1024 to 49151	Registered Ports
49152 to 65535	Private and/or Dynamic Ports

Dynamic port usage will become clearer as we move through the material.

Port Numbers

- UDP Port Numbers

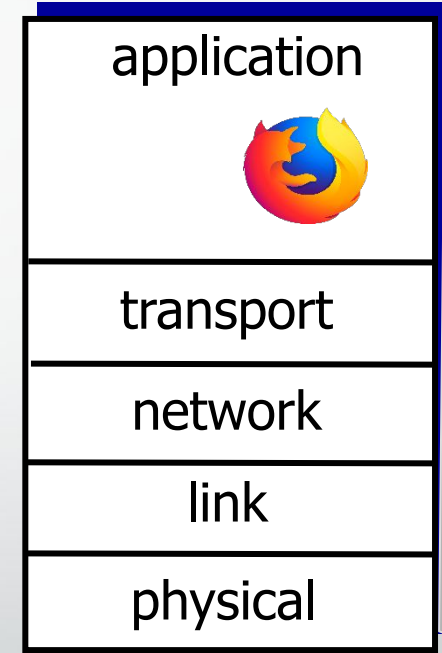
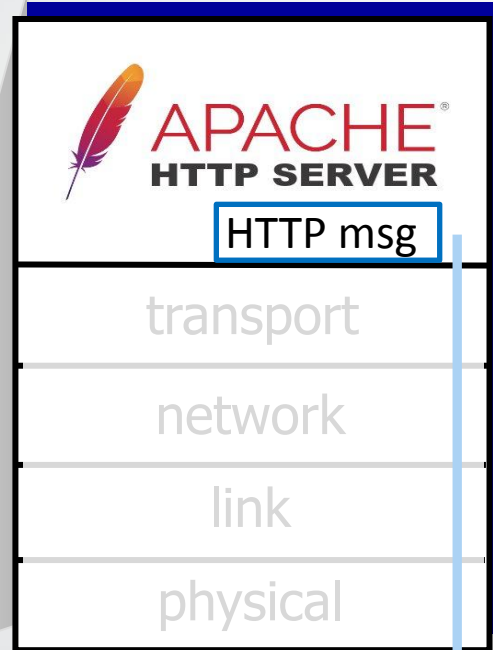
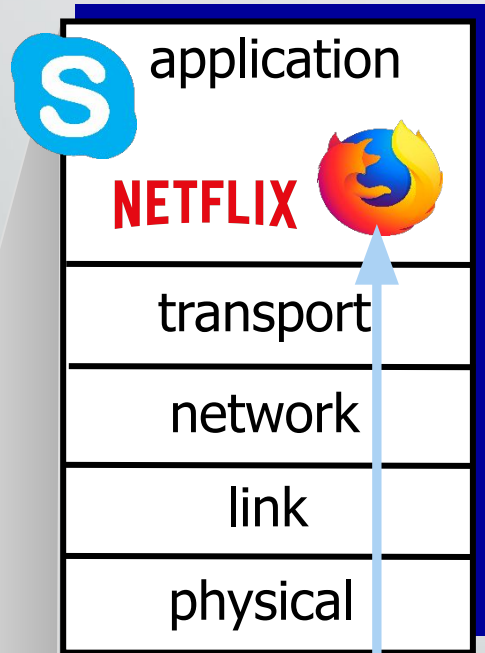


TCP Port Numbers

Function 3 - Multiplexing

HTTP server

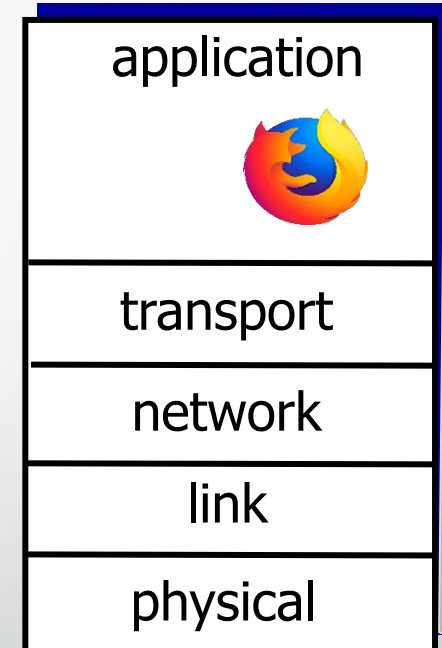
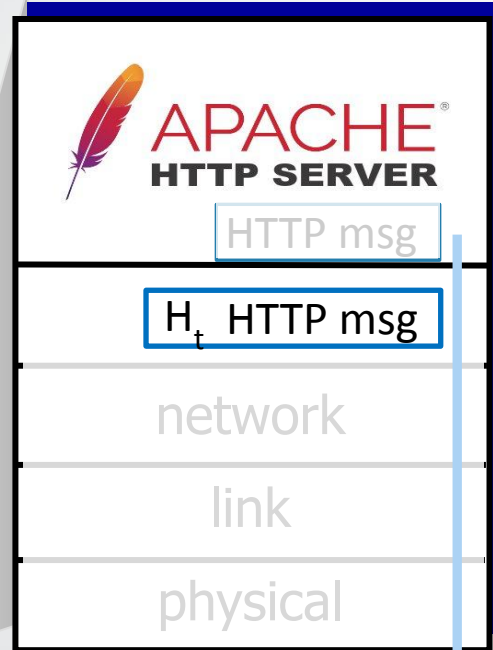
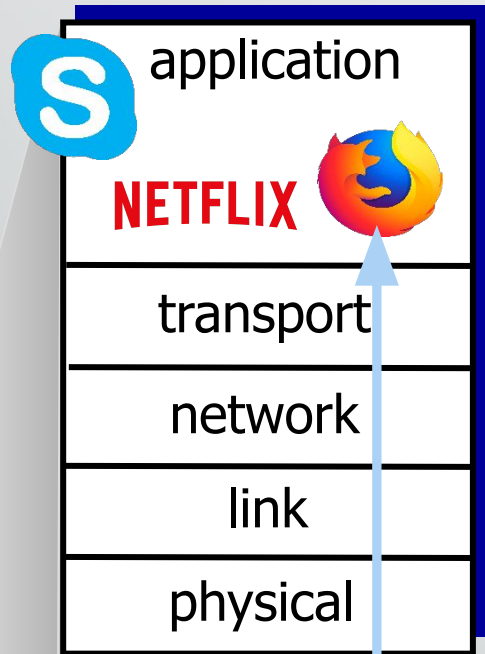
client



Multiplexing

HTTP server

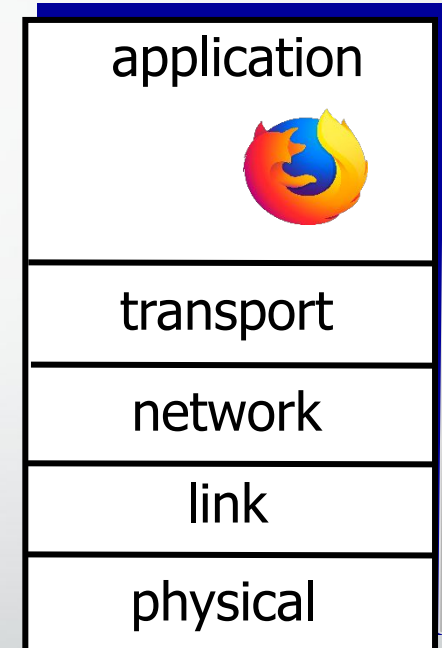
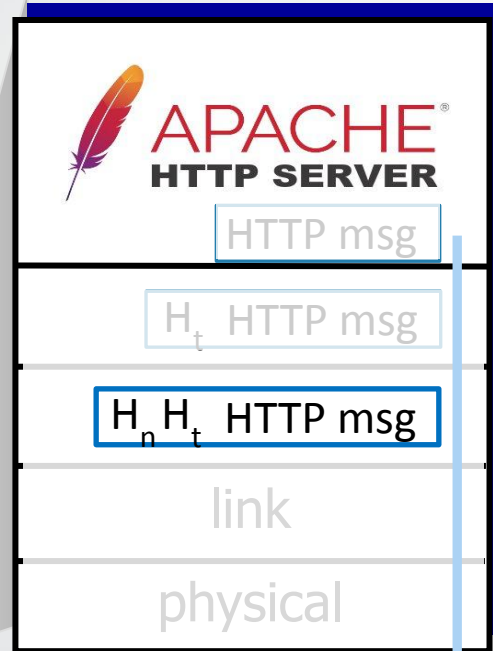
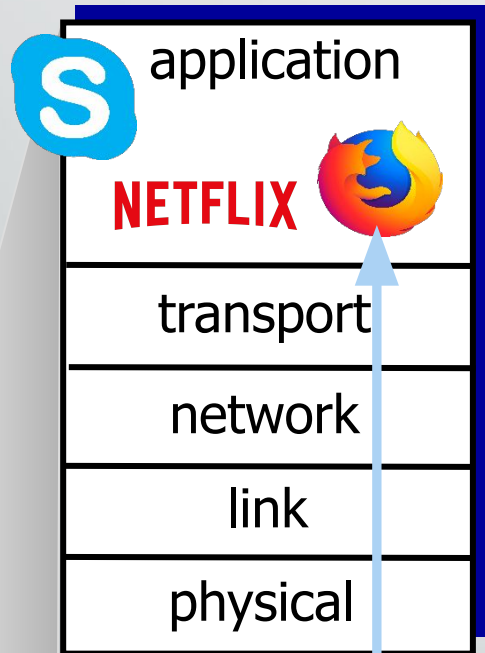
client



Multiplexing

HTTP server

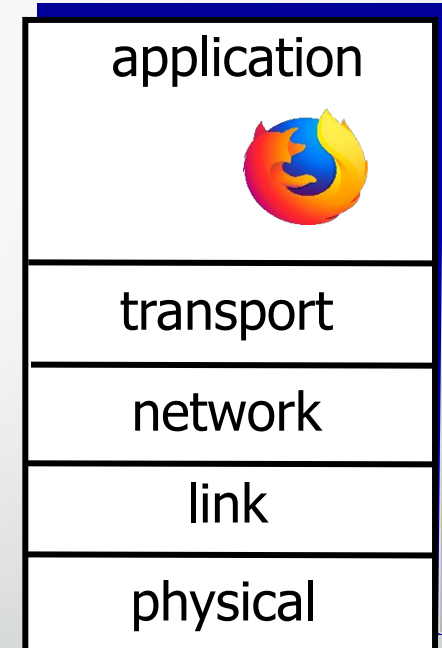
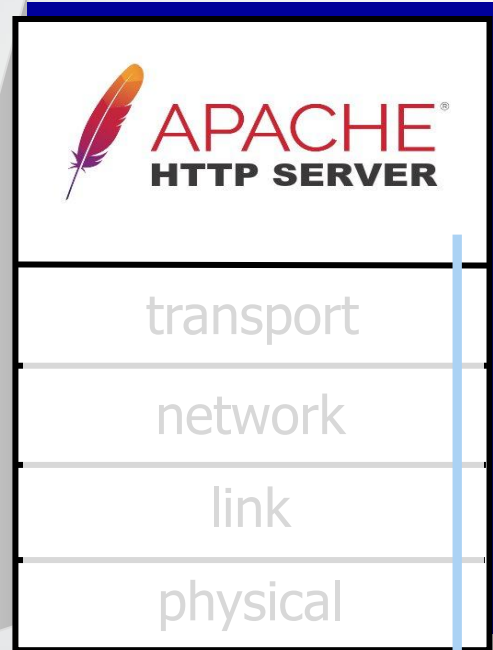
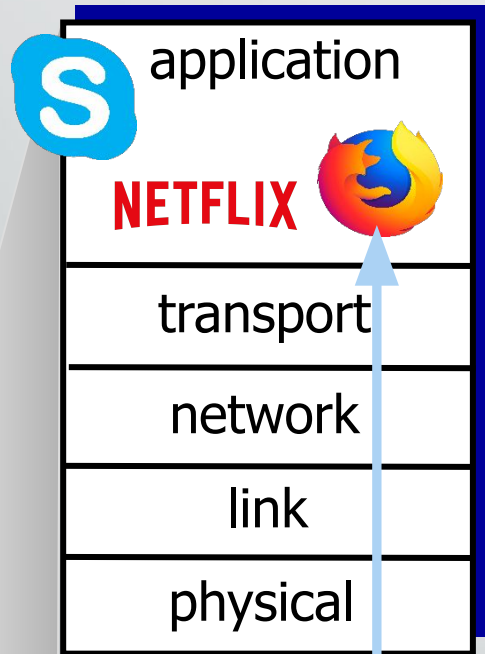
client



Multiplexing

HTTP server

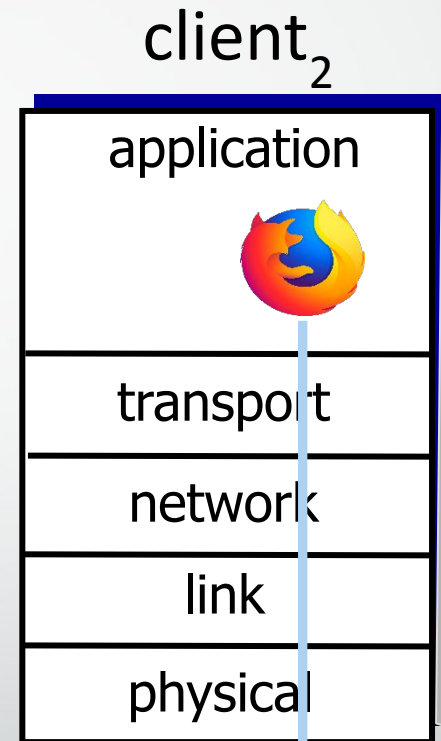
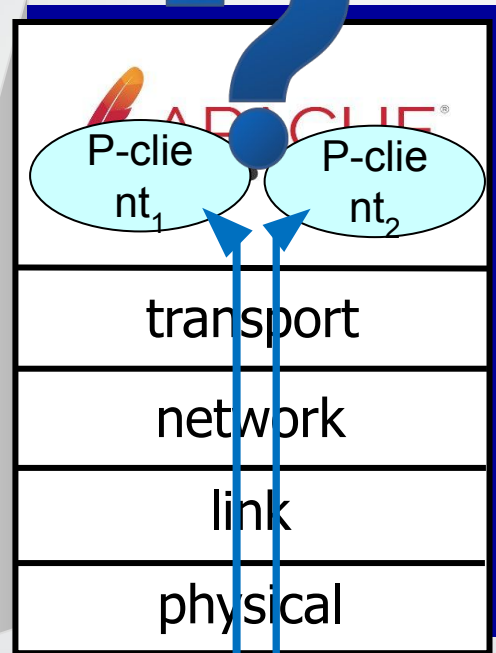
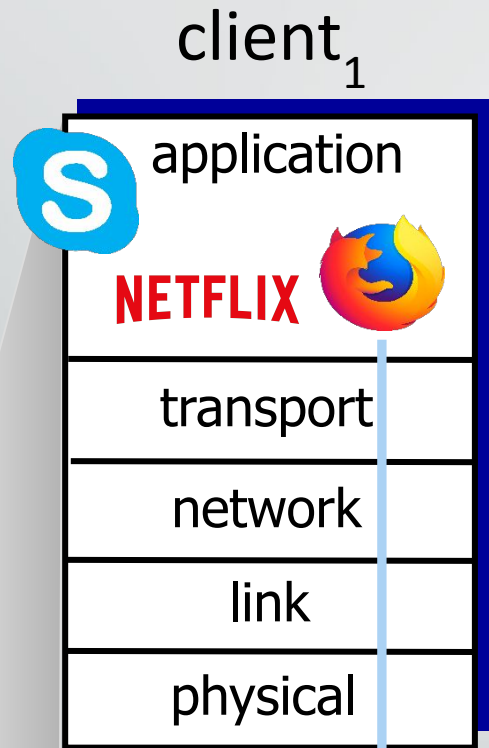
client



$H_n H_t$ HTTP msg

Multiplexing

HTTP server



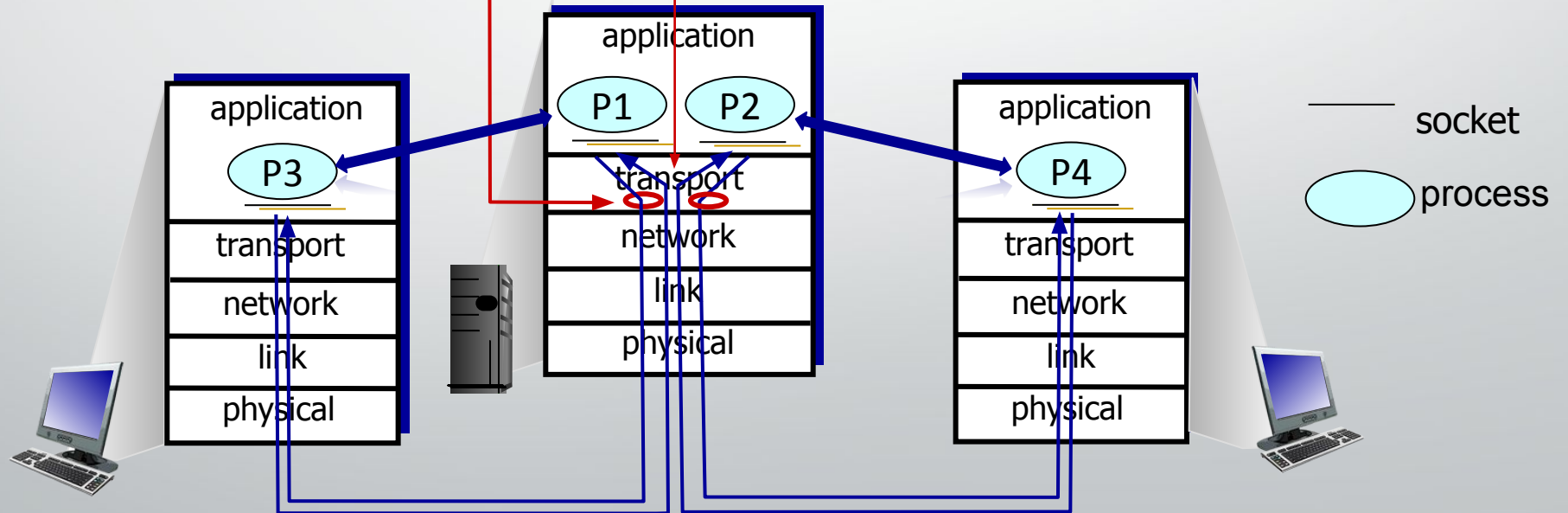
Multiplexing/demultiplexing

multiplexing at sender:

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

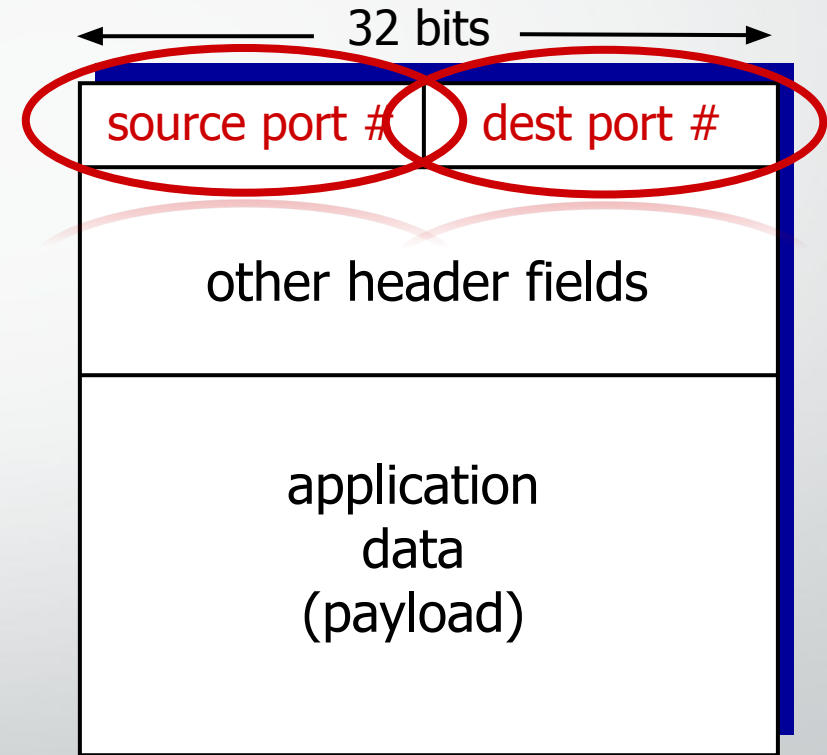
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

- host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries one transport-layer segment
 - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- when creating datagram to send into UDP socket, must specify
 - destination IP address
 - destination port #

when receiving host receives *UDP* segment:

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



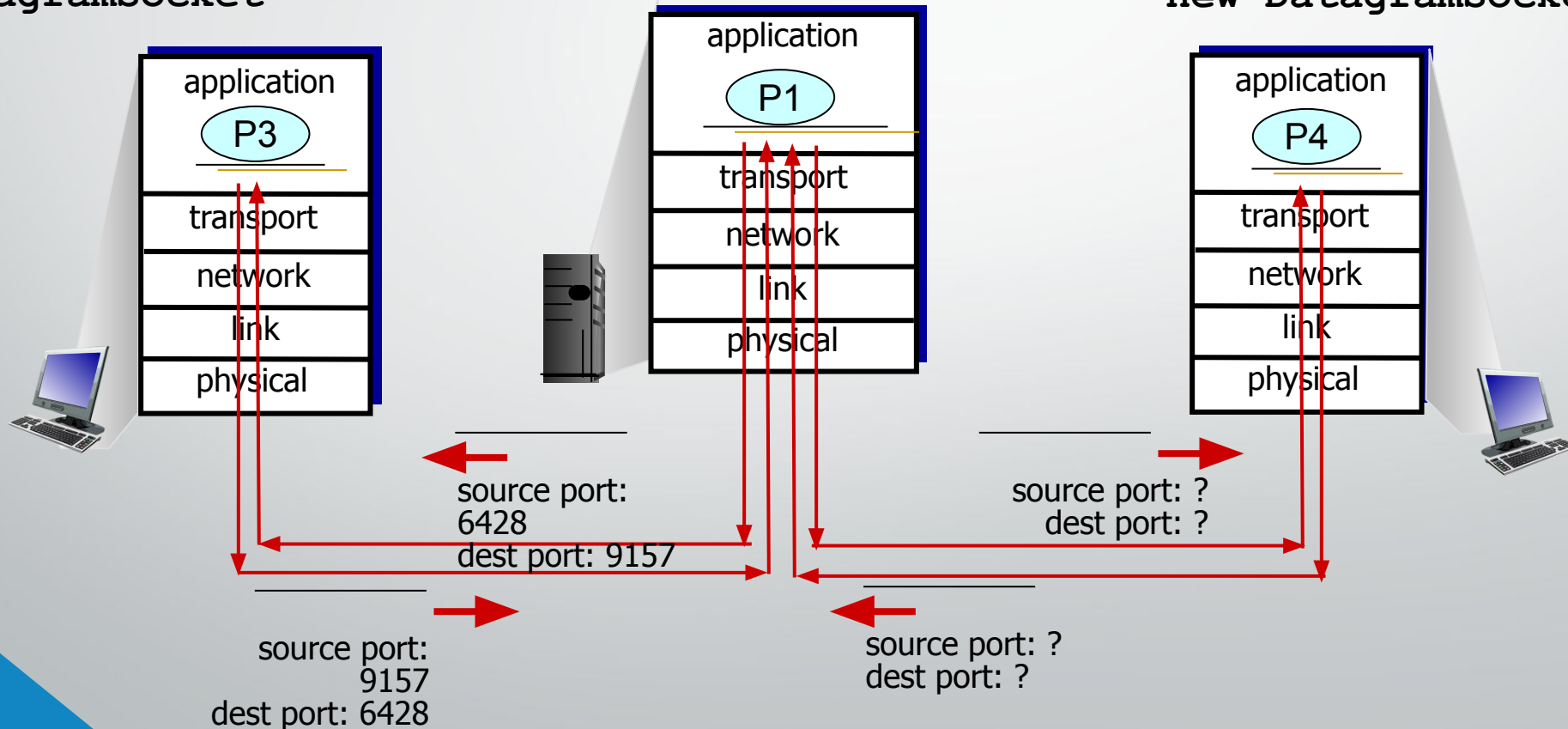
IP/UDP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

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

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

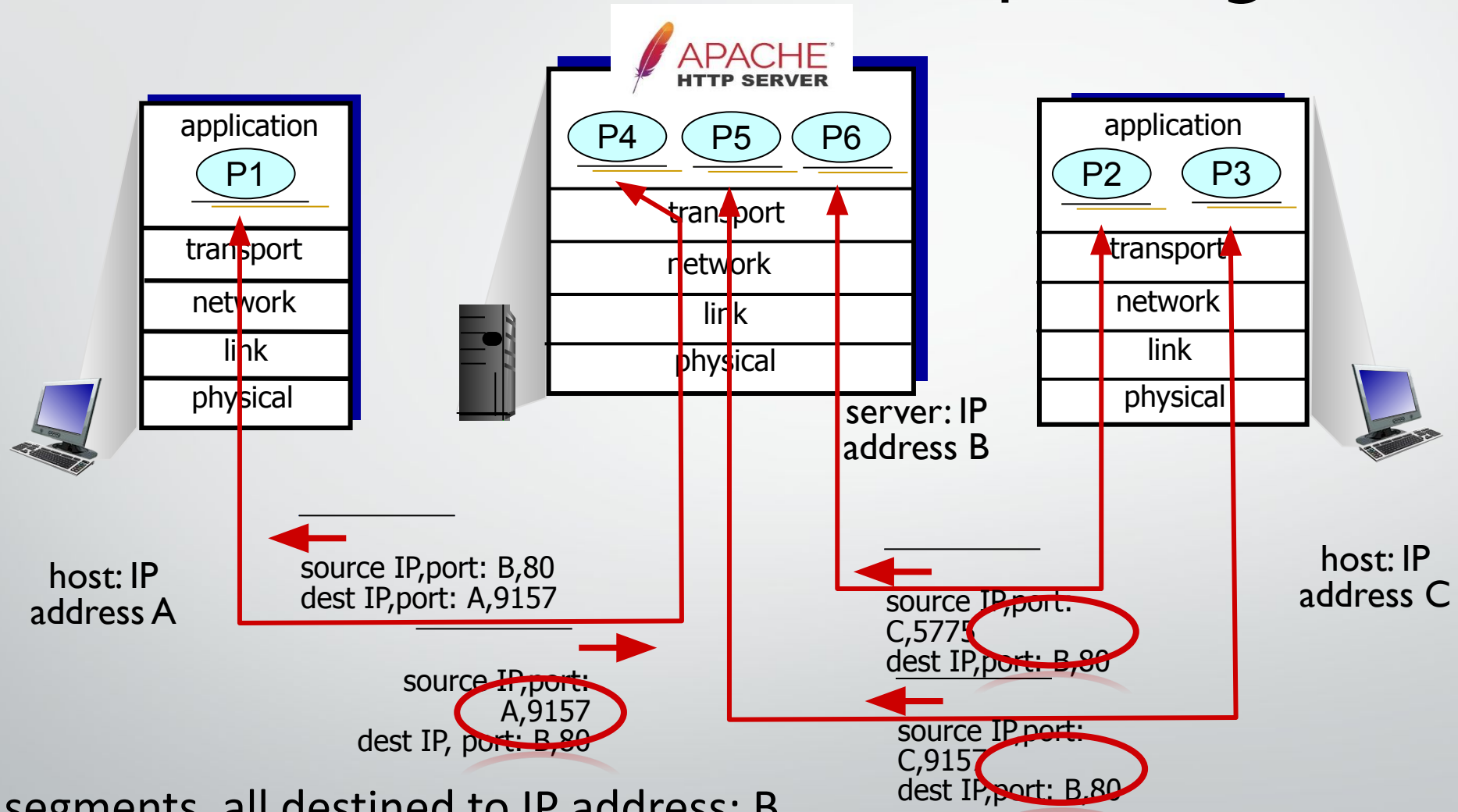
```
DatagramSocket mySocket1 =  
new DatagramSocket (5775);
```



Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
 - each socket associated with a different connecting client

Connection-oriented demultiplexing: example

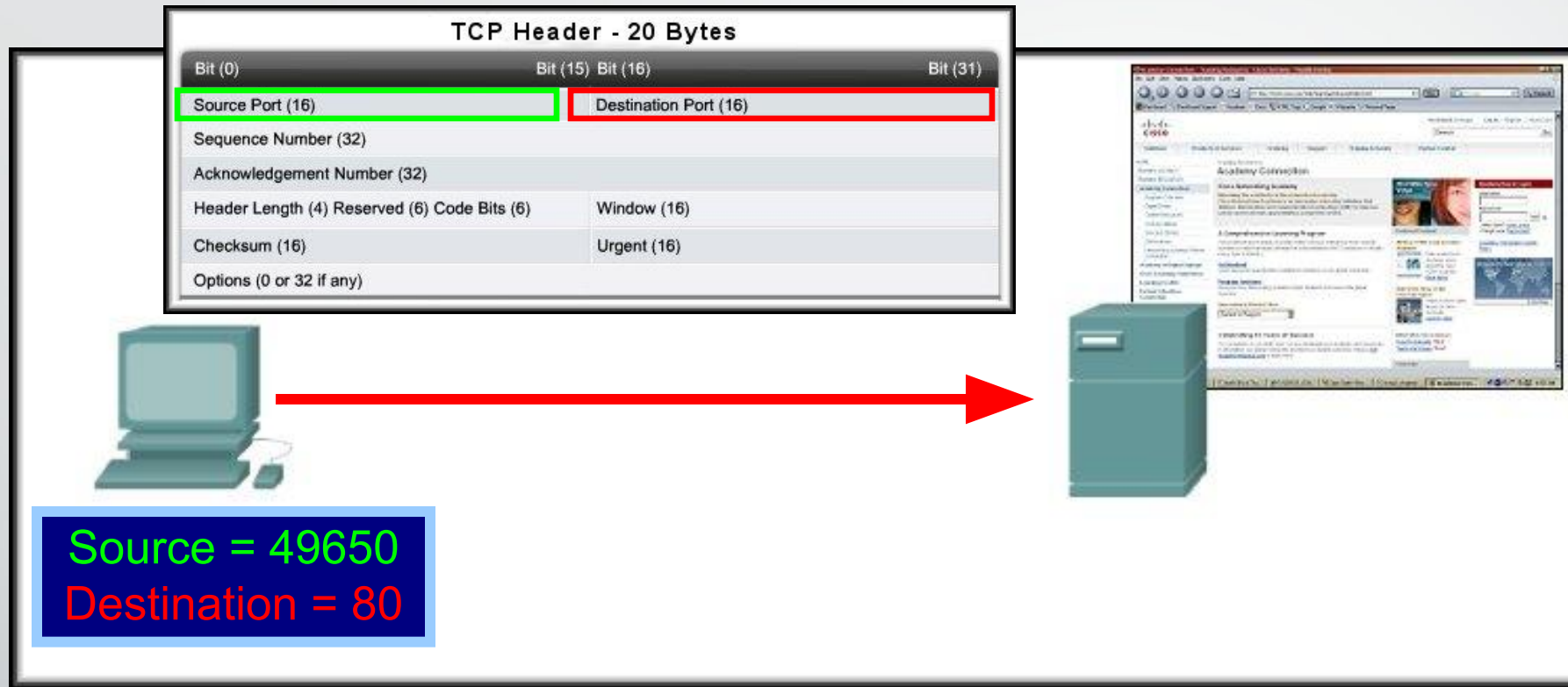


Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Summary

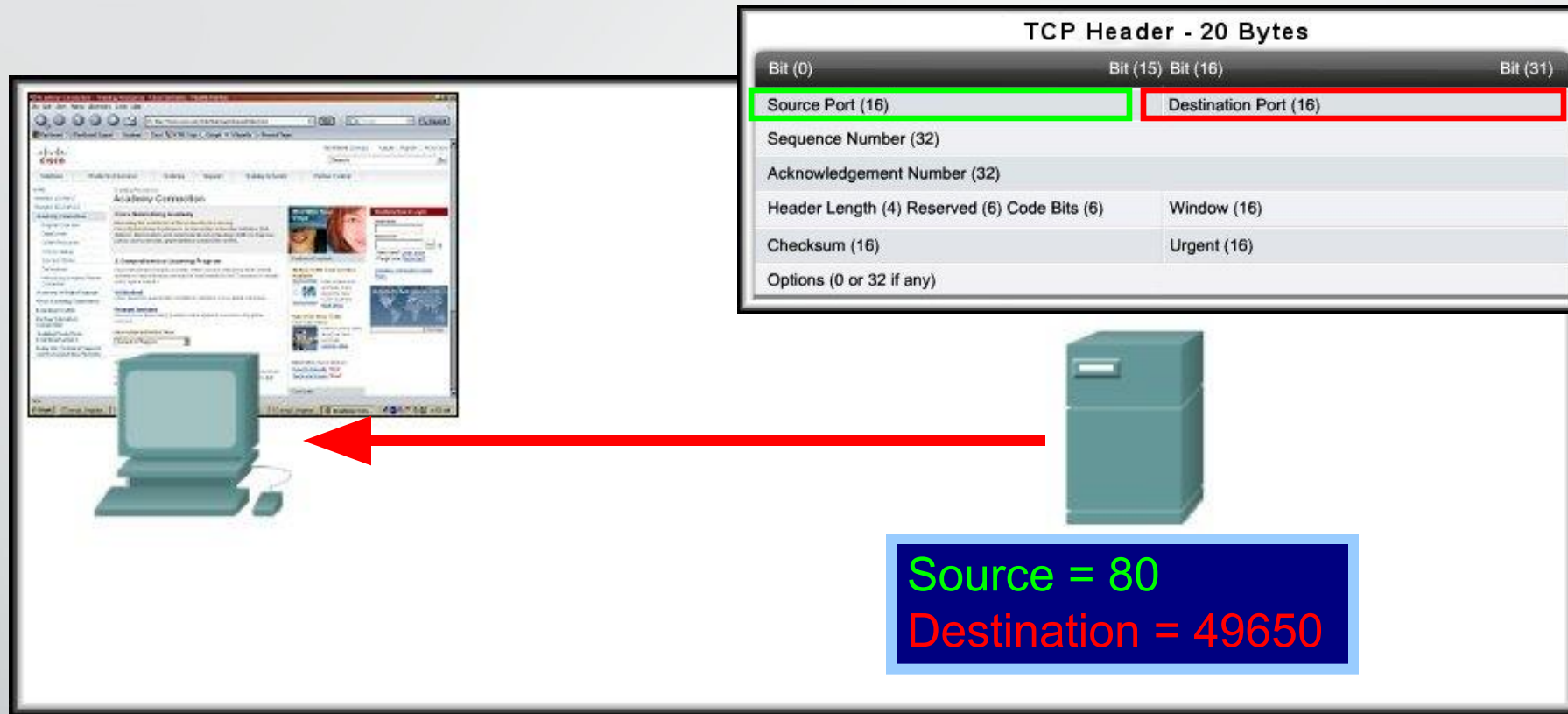
- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

More on Port Numbers



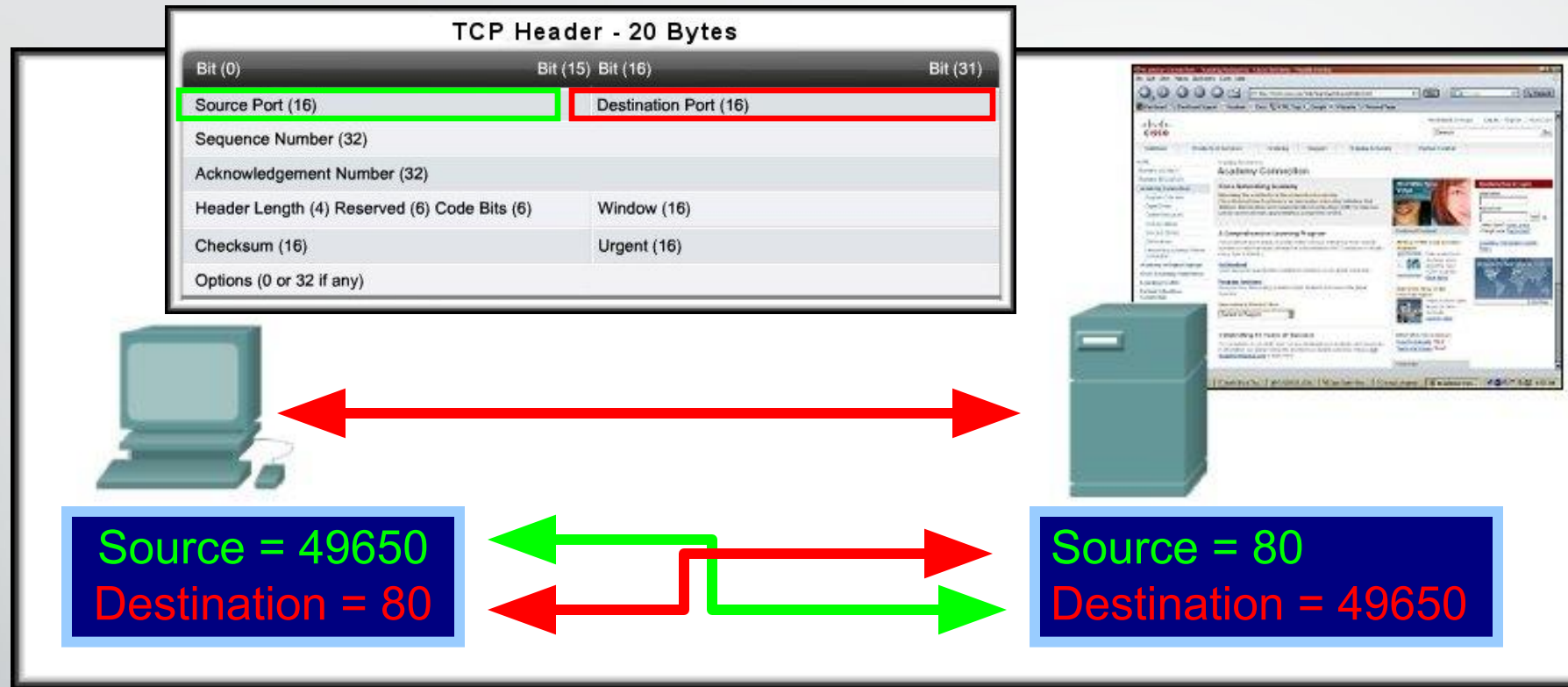
- Server is listening on Port 80 for HTTP connections.
- The client sets the destination port to 80 and uses a dynamic port as its source.

Port Numbers in Action



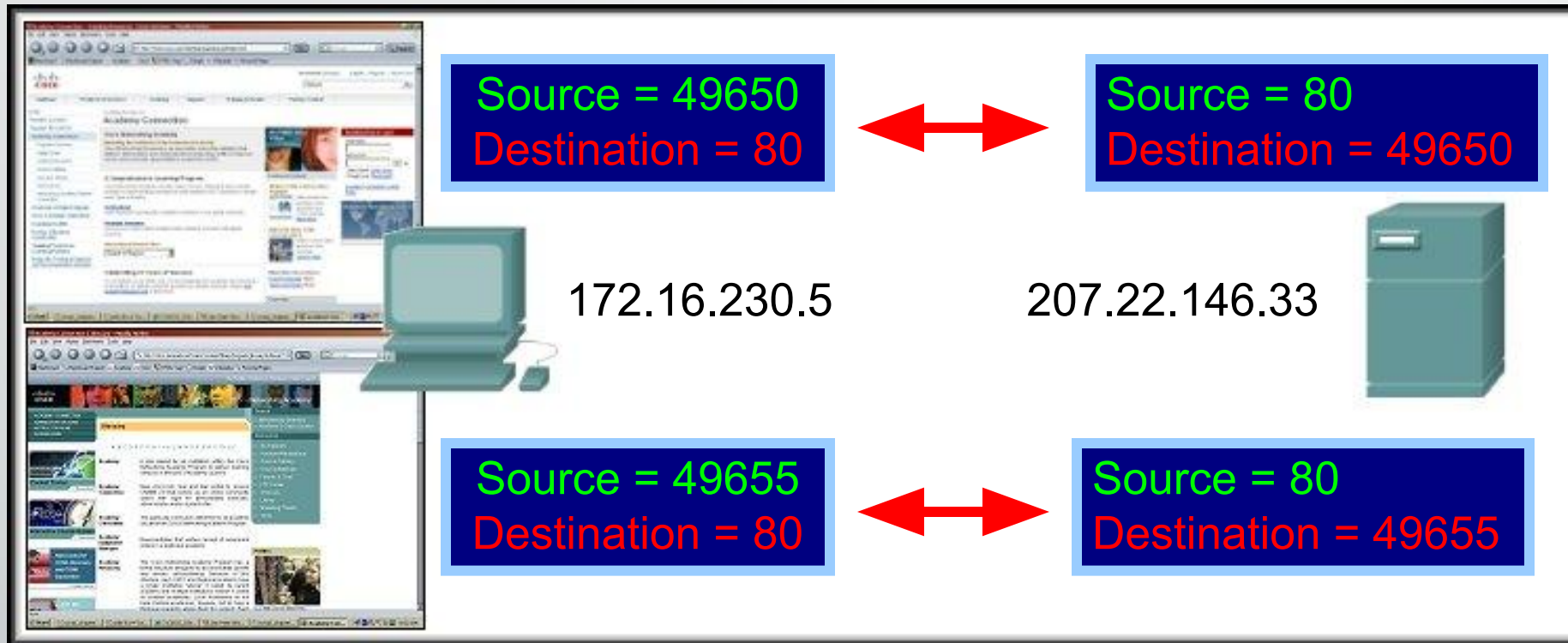
- Server replies with the web page.
 - Sets the source port to 80 and uses the client's source port as the destination.

Port Numbers in Action



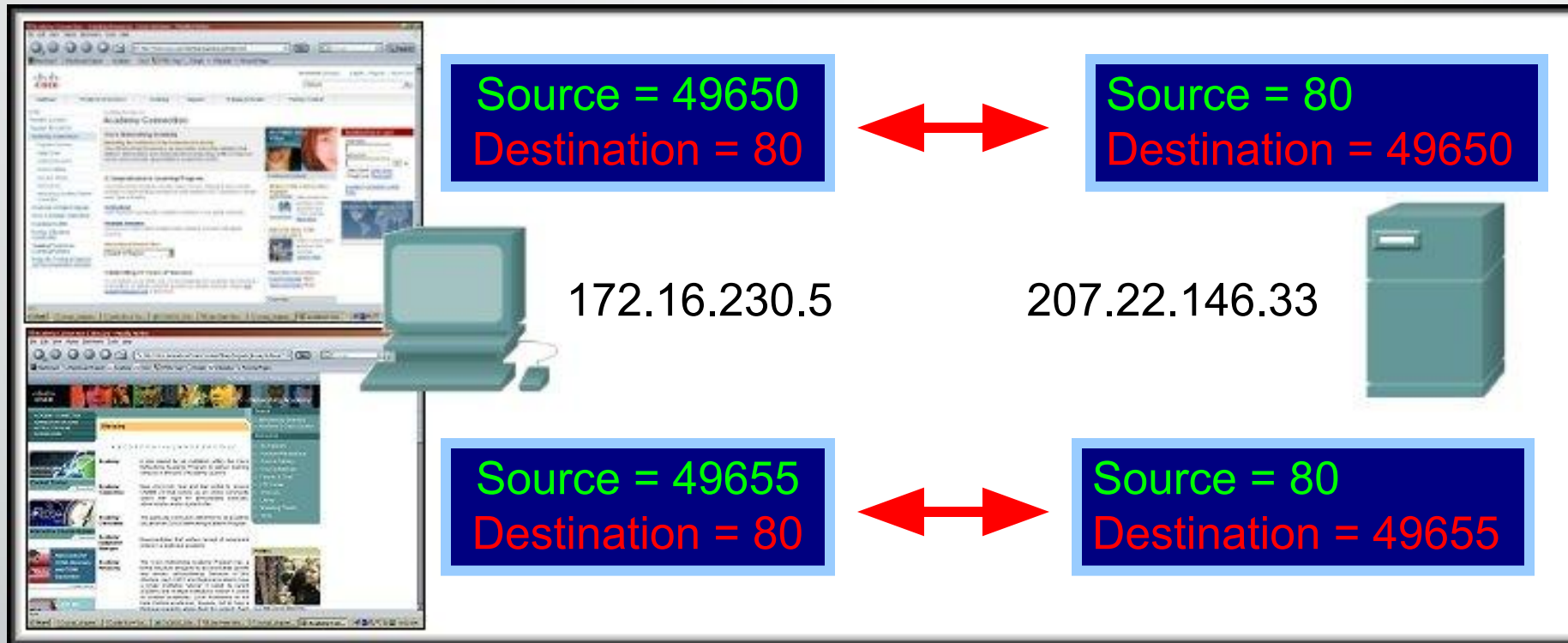
- Notice how the source and destination ports are used.
- Clients can use any random port number, servers can't.
 - Because clients won't be able to identify server process otherwise
- Servers, however, cannot use any random port number
 - Use of well-known port numbers!

Port Numbers in Action



- What if there are two sessions to the same server?
 - The client uses **another dynamic port** as its source and the destination is **still port 80**.
 - **Different source ports** keep the sessions unique on the server.

Port Numbers in Action



- How does the Transport Layer keep them separate?
 - The socket (IP Address:Port)

172.16.230.5:49650 ↔ 207.22.146.33:49650

172.16.230.5:49655 ↔ 207.22.146.33:49655

Source IP

Source Port

Destination IP

Destination Port

TCP/UDP

Source Socket

Destination Socket

Connection State

netstat -a -n command

Proto	Local Address	Foreign Address	State
CP	0.0.0.0:135	0.0.0.0:0	
CP	0.0.0.0:445	0.0.0.0:0	
TCP	0.0.0.0:3389	0.0.0.0:0	LISTENING
TCP	0.0.0.0:48698	0.0.0.0:0	LISTENING
TCP	127.0.0.1:3582	127.0.0.1:3583	ESTABLISHED
TCP	127.0.0.1:3583	127.0.0.1:3582	ESTABLISHED
TCP	192.168.1.103:135	0.0.0.0:0	LISTENING
TCP	192.168.1.103:3586	204.225.7.4:80	TIME_WAIT
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:5000	*:*	
UDP	0.0.0.0:1025	*:*	
UDP	0.0.0.0:1026	*:*	
UDP	0.0.0.0:1030	*:*	
UDP	0.0.0.0:1038	*:*	
UDP	0.0.0.0:1346	*:*	
UDP	0.0.0.0:4500	*:*	
UDP	0.0.0.1:123	*:*	
UDP	127.0.0.1:1900	*:*	
UDP	127.0.0.1:3492	*:*	
UDP	192.168.1.103:123	*:*	
UDP	192.168.1.103:137	*:*	
UDP	192.168.1.103:138	*:*	
UDP	192.168.1.103:139	*:*	

Netstat - Network Utility Tool

- Actually, when you open up a single web page, there are usually several TCP sessions created, not just one.

UDP: User Datagram Protocol [RFC 768]

UDP is a connectionless, unreliable protocol that has no flow and error control. It uses port numbers to track and multiplex data received from the application layer.

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

- ❖ UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
 - SNMP
 - HTTP/3
- ❖ reliable transfer over UDP:
 - add reliability at application layer
 - application-specific error recovery!

User Datagram Protocol (UDP)

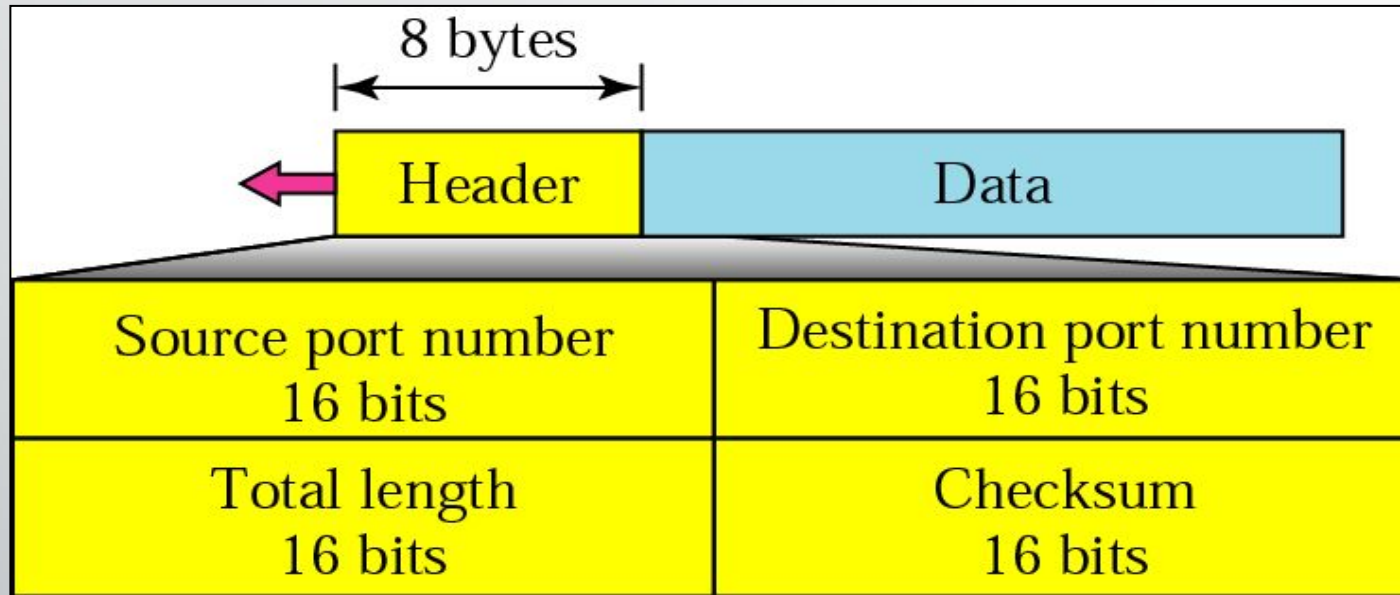
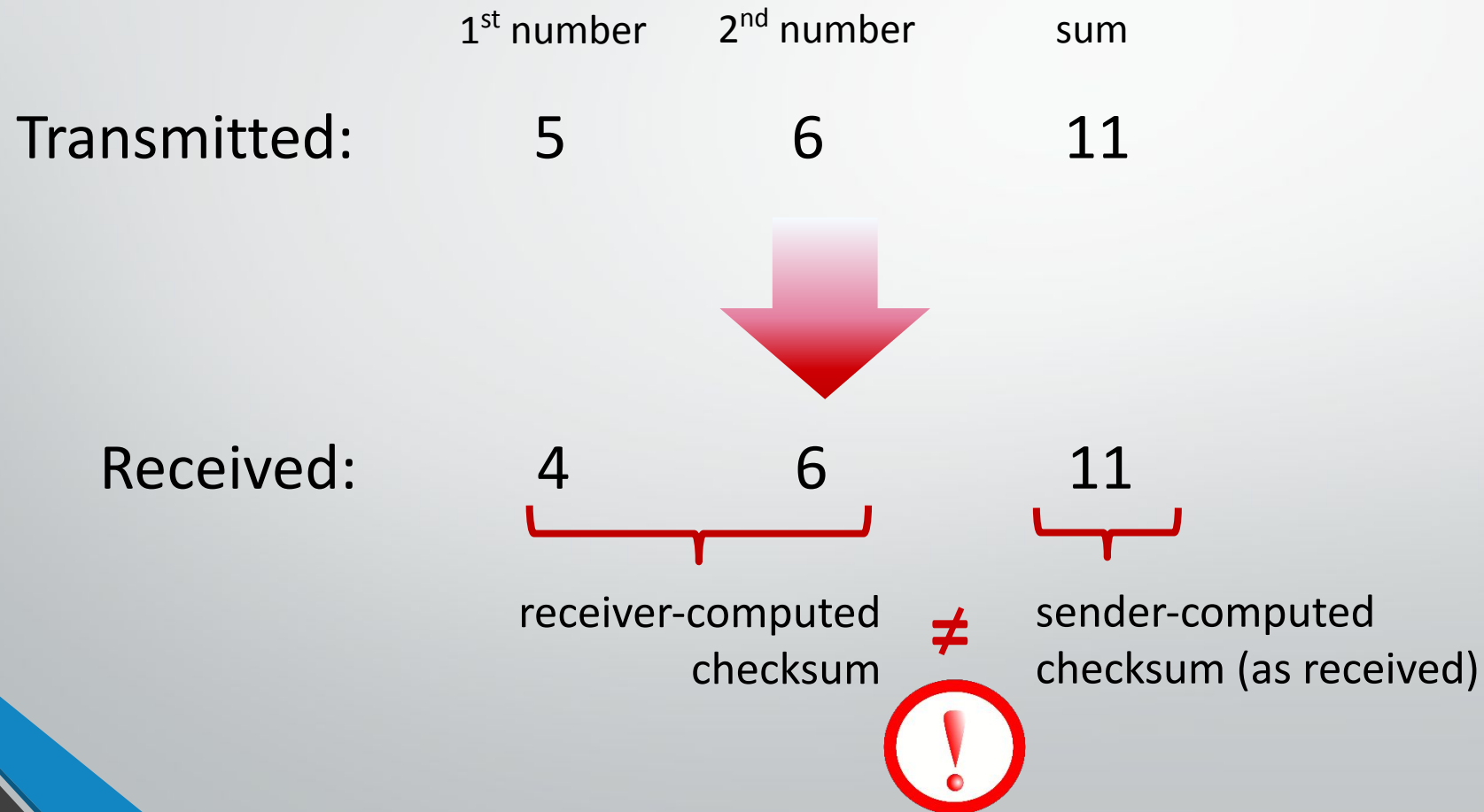


Figure 14: User Datagram Header format

- Connectionless
- No reassembly to order.
- No Error checking
- No Flow control

UDP checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment



UDP checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- **checksum:** addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
 - Not equal - error detected
 - Equal - no error detected. *But maybe errors nonetheless? More later*

Internet checksum: an 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

Internet checksum: weak protection!

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
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

Even though numbers have changed (bit flips), *no* change in checksum!



THE END