



Week 5

Intro to Transport Layer & UDP

CSCM603154 – **Computer Networks**

Faculty of Computer Science Universitas Indonesia

All material taken from “Computer Networking: A Top-Down Approach”
copyright 1996-2020, J.F Kurose and K.W. Ross, All Rights Reserved

Transport layer: overview

Our goal:

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

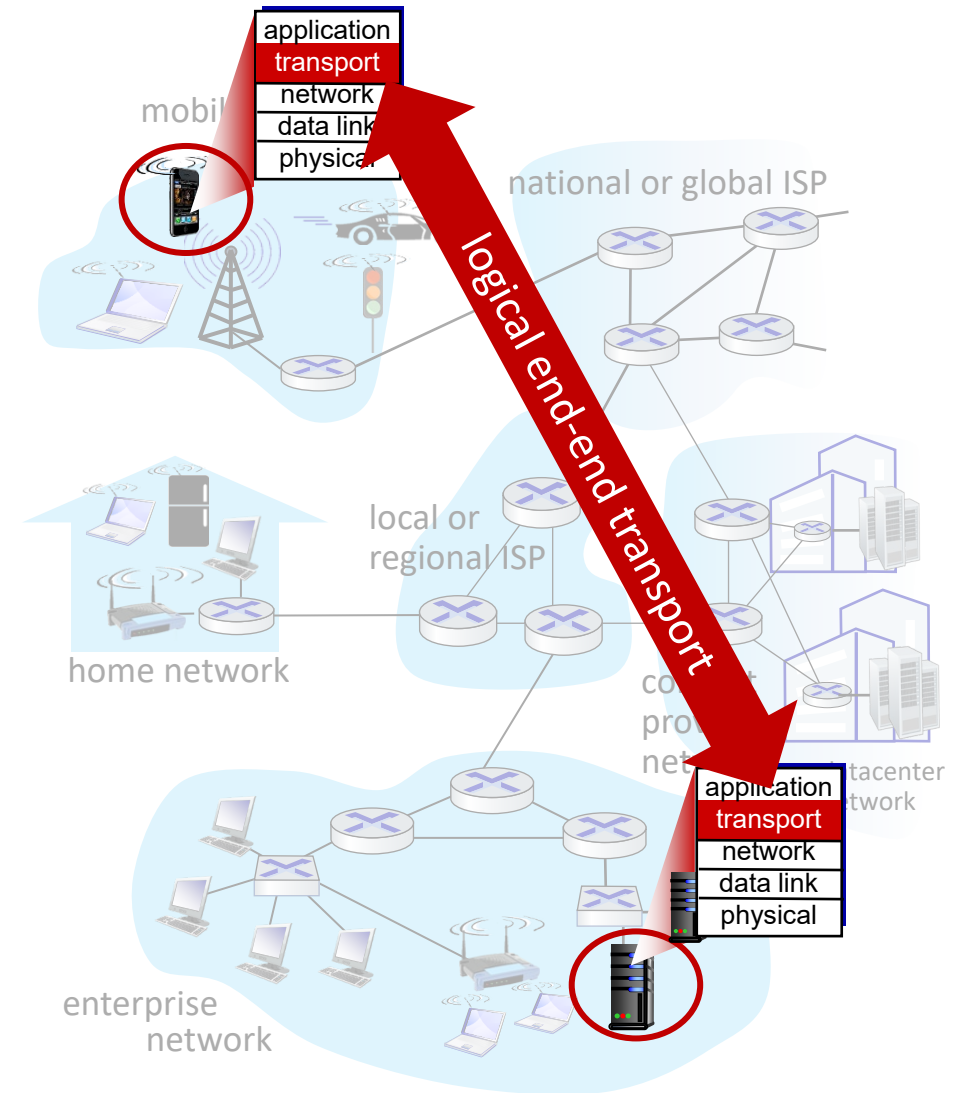
Transport layer: roadmap

- Transport-layer services
 - Multiplexing and demultiplexing
 - Connectionless transport: UDP
 - Connection-oriented transport: TCP
 - TCP congestion control



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 services and protocols



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 vs. network layer services and protocols

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

household analogy:

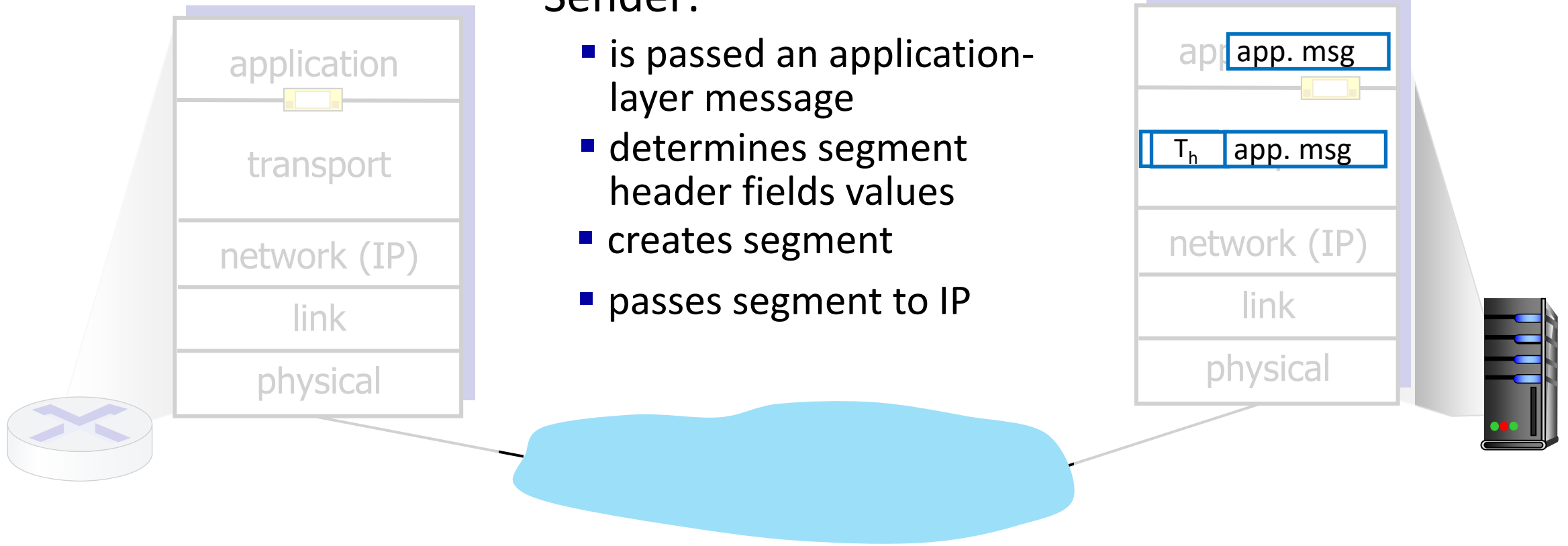
12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes

Transport Layer Actions

Sender:

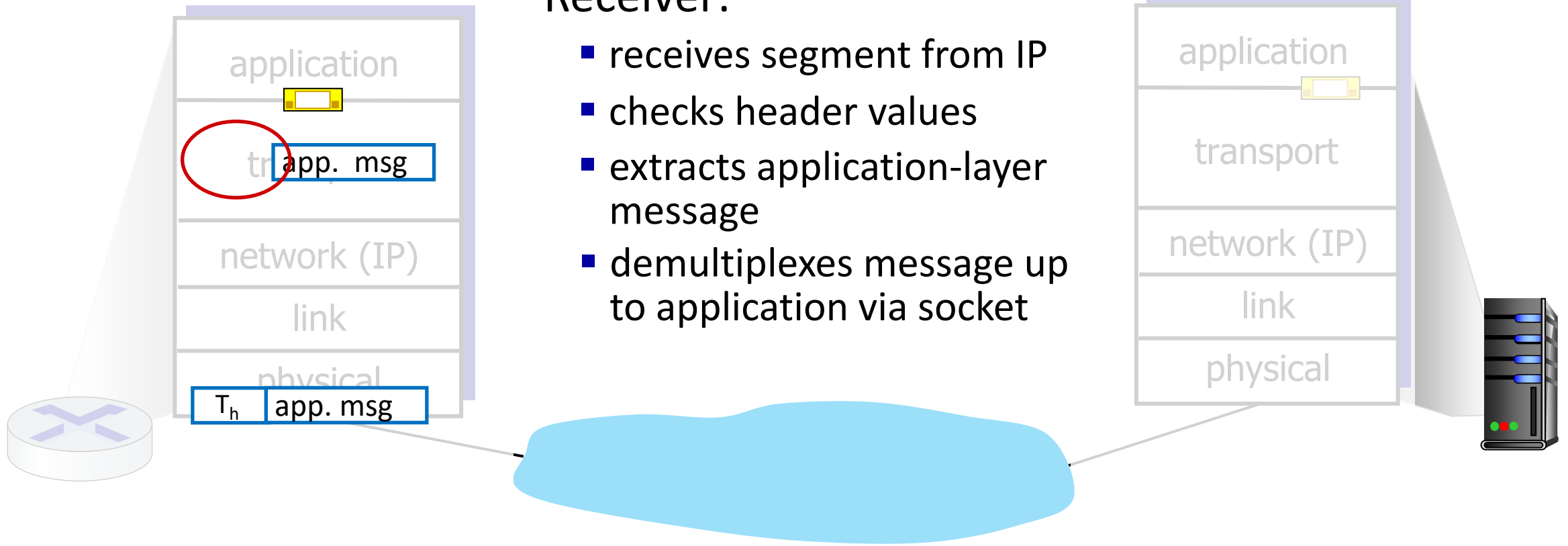
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



Transport Layer Actions

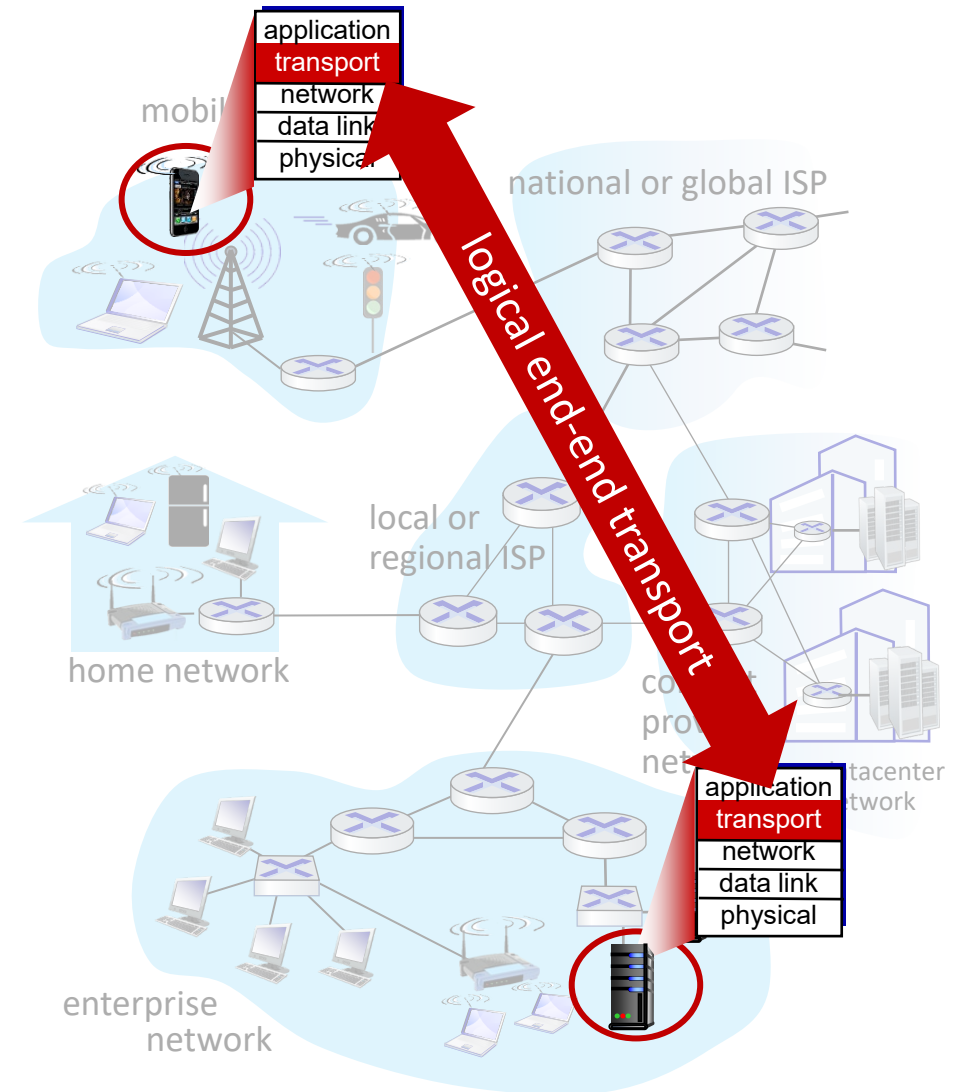
Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



Two principal Internet transport protocols

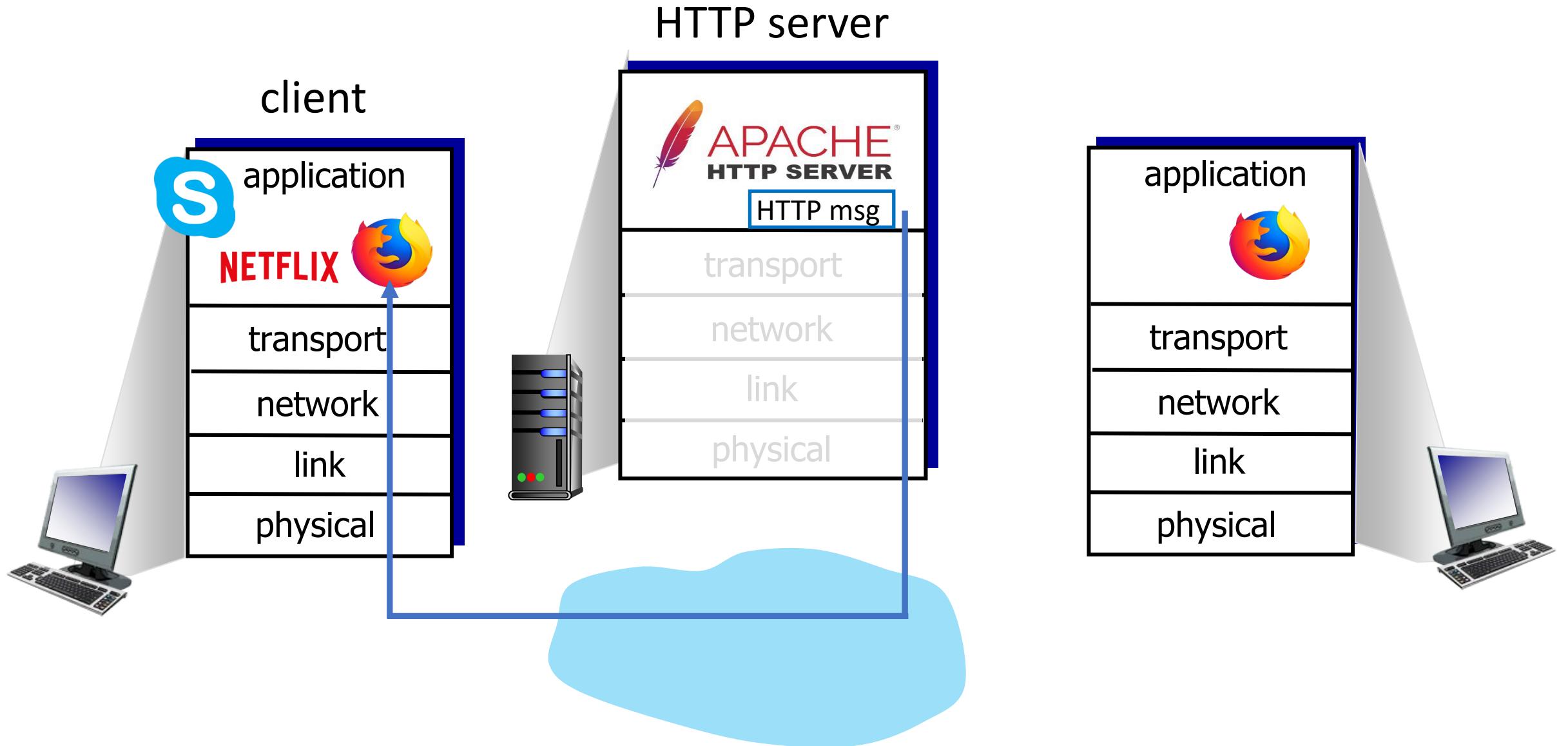
- **TCP:** Transmission Control Protocol
 - reliable, in-order delivery
 - congestion control
 - flow control
 - connection setup
- **UDP:** User Datagram Protocol
 - unreliable, unordered delivery
 - no-frills extension of “best-effort” IP
- services not available:
 - delay guarantees
 - bandwidth guarantees

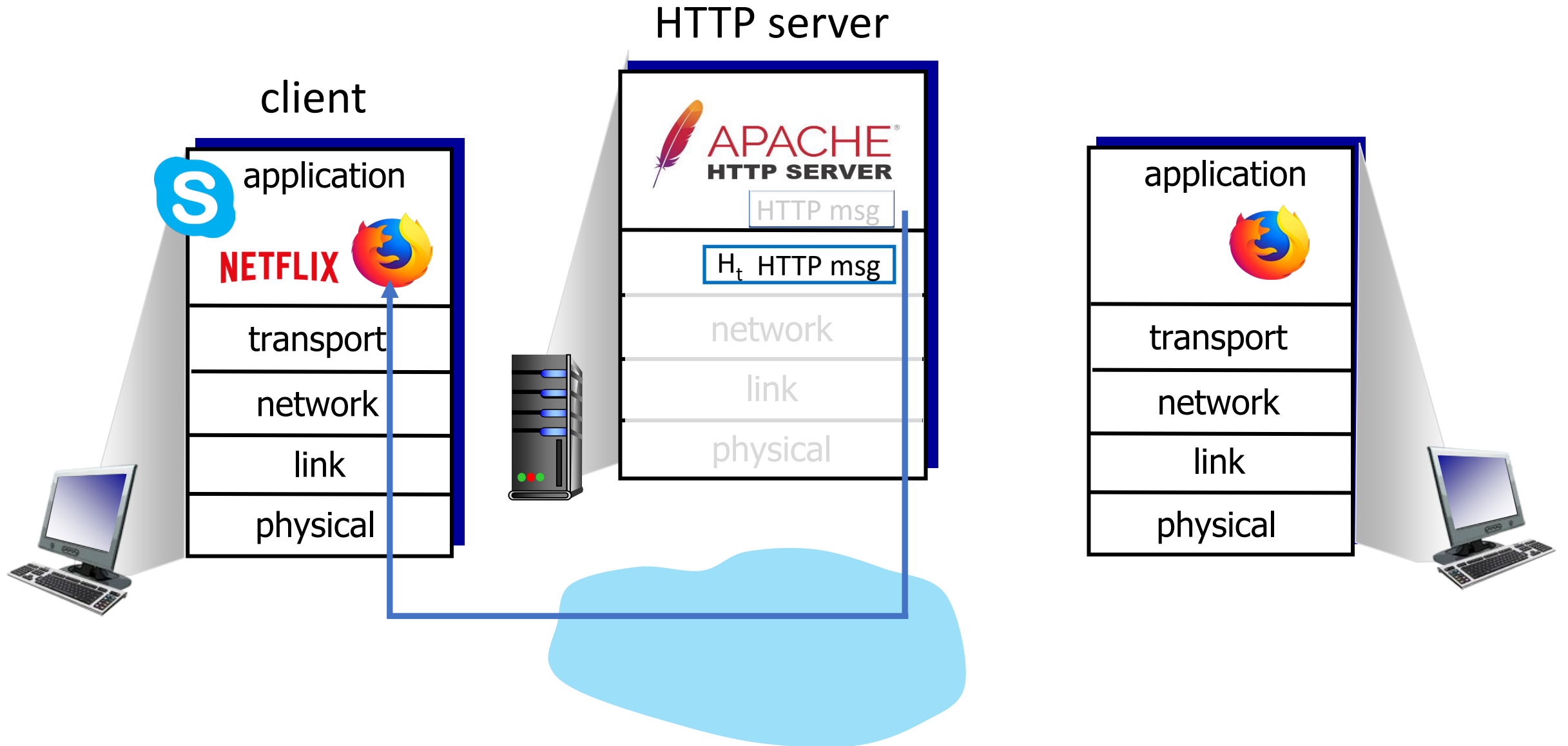


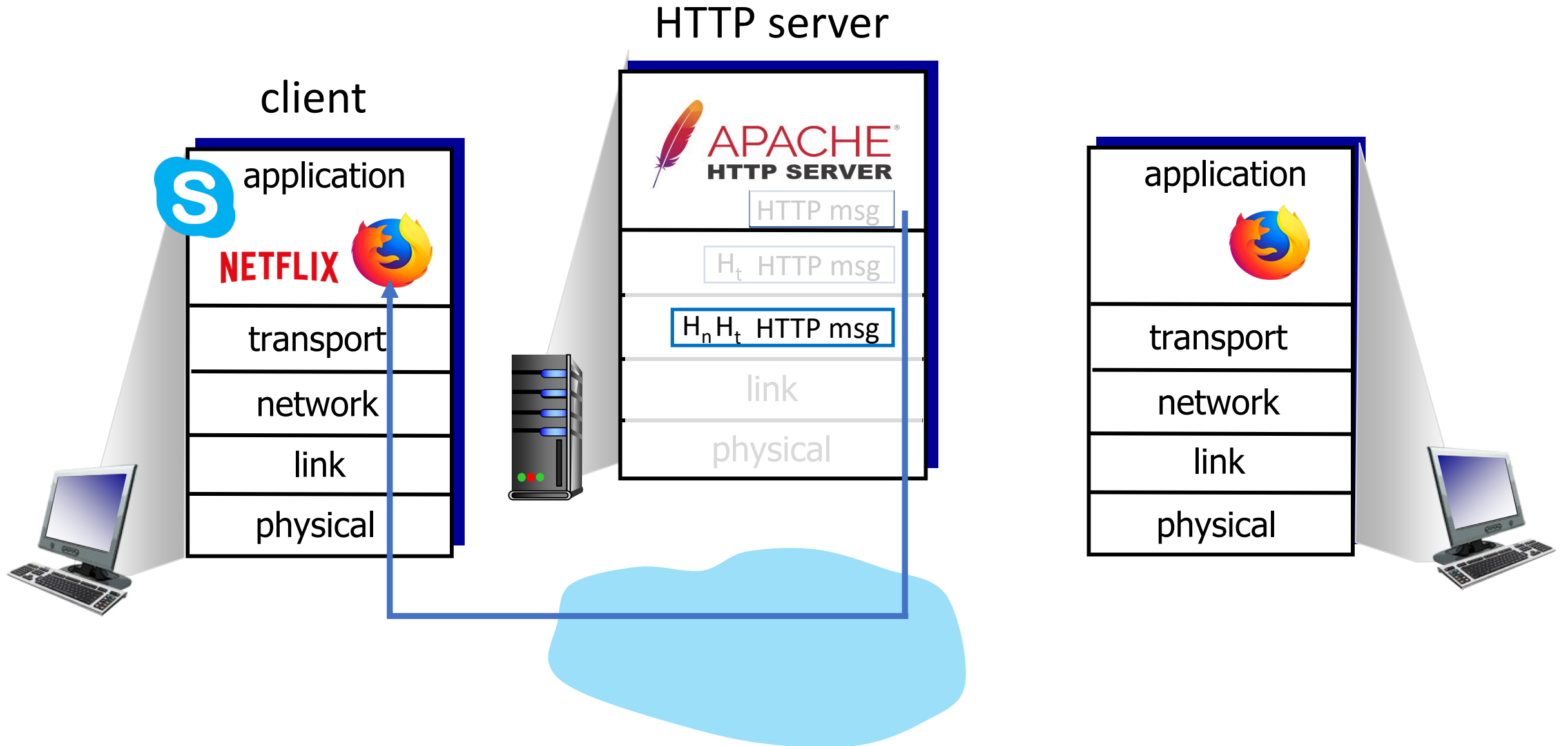
Chapter 3: roadmap

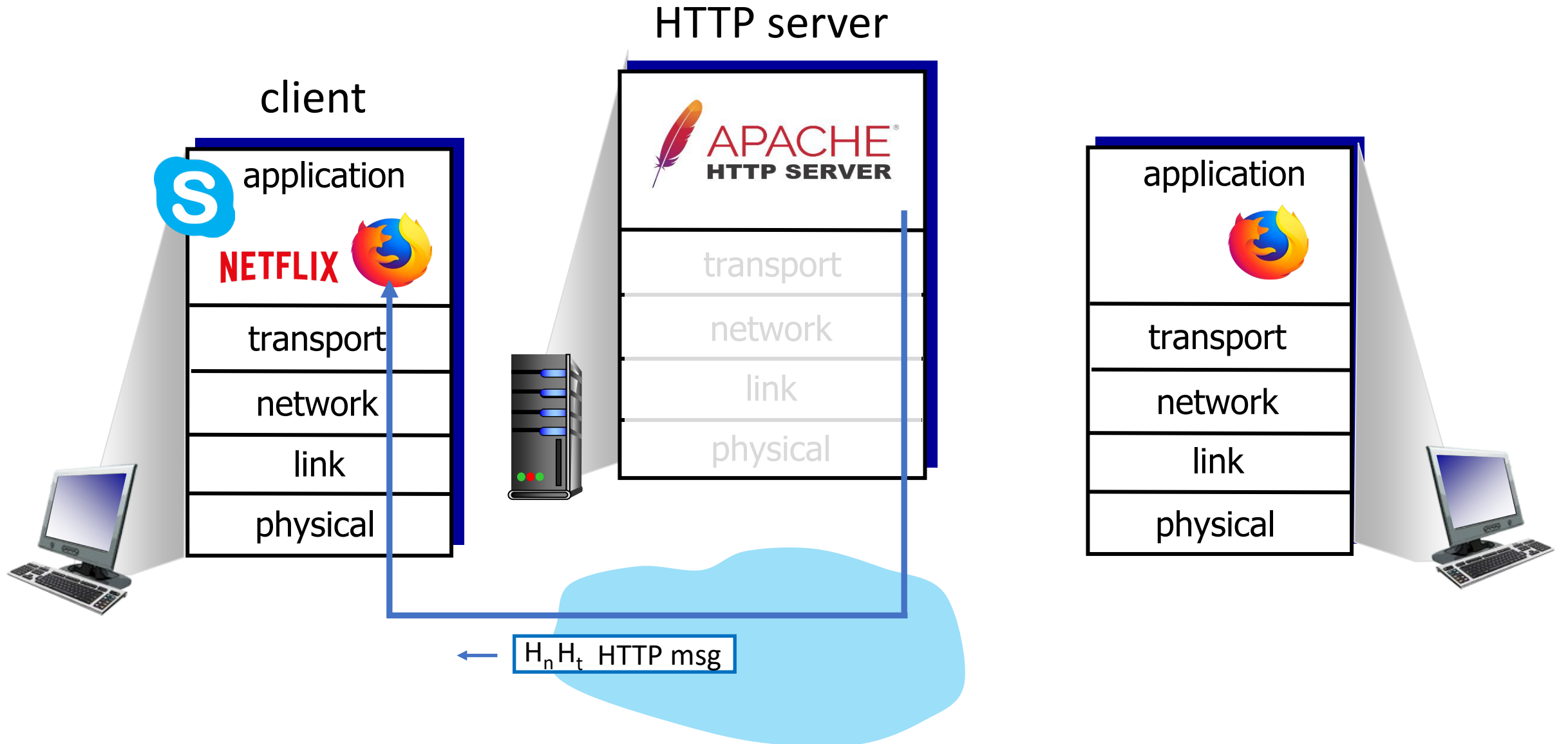
- Transport-layer services
- **Multiplexing and demultiplexing**
- Connectionless transport: UDP
- Connection-oriented transport: TCP
- TCP congestion control

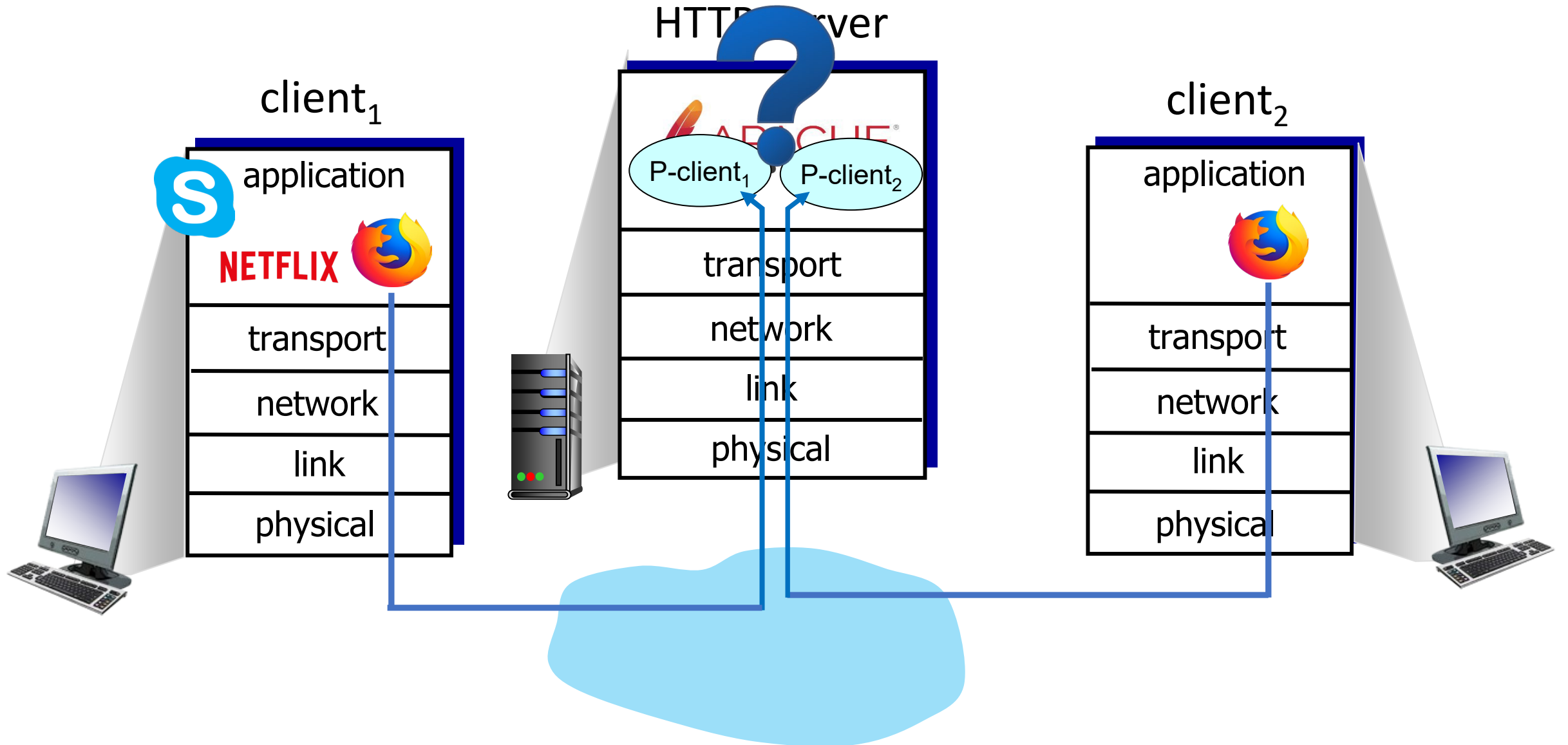












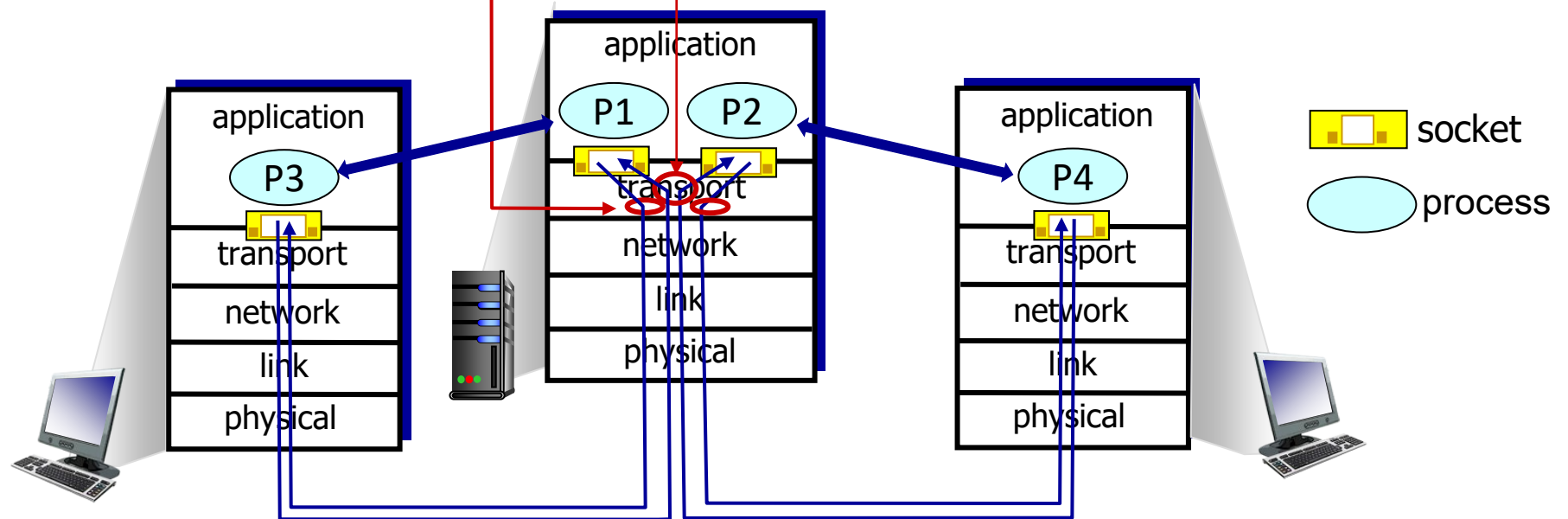
Multiplexing/demultiplexing

multiplexing at sender:

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

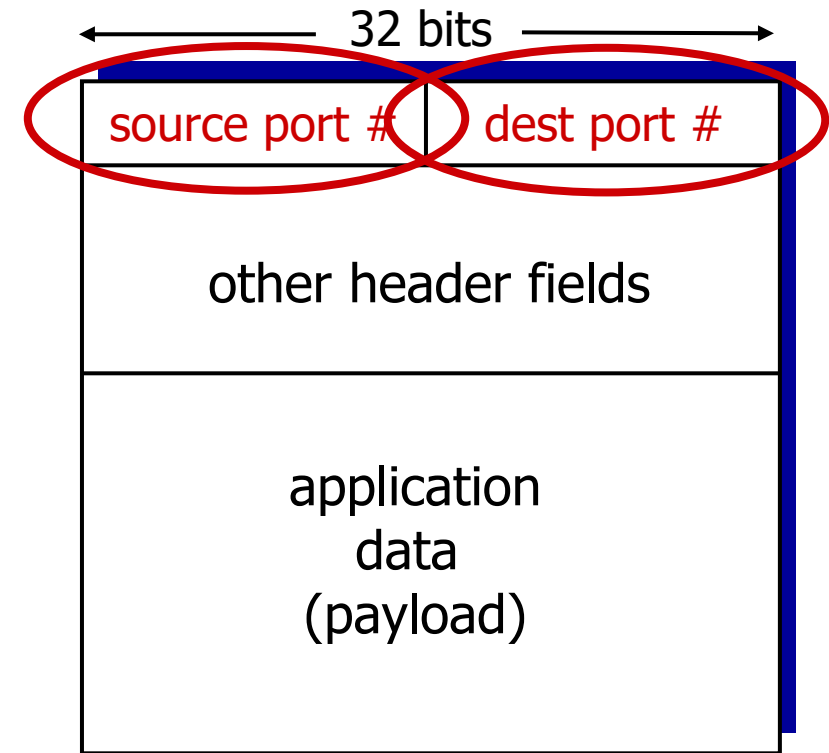
demultiplexing at receiver:

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

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1  
= new DatagramSocket(12534);
```

- 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 #



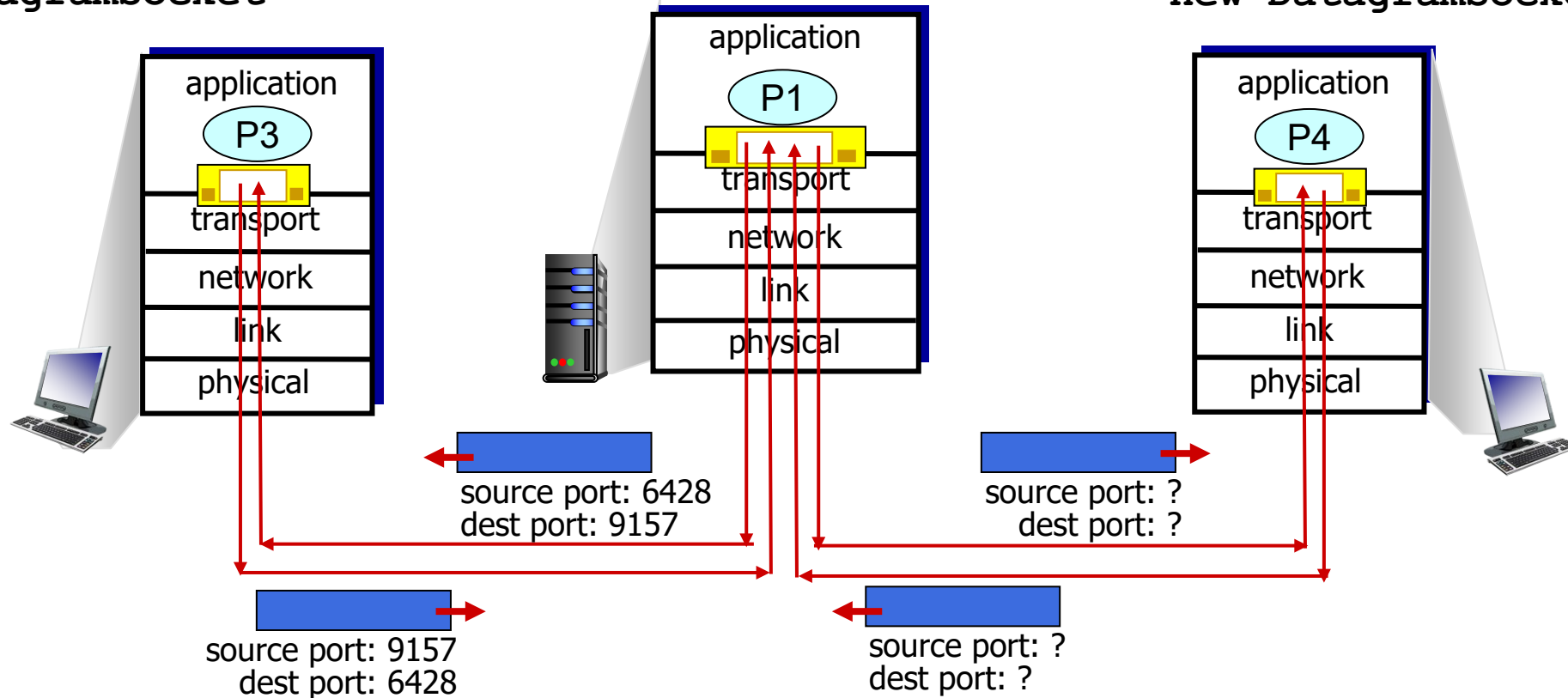
IP/UDP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

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

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

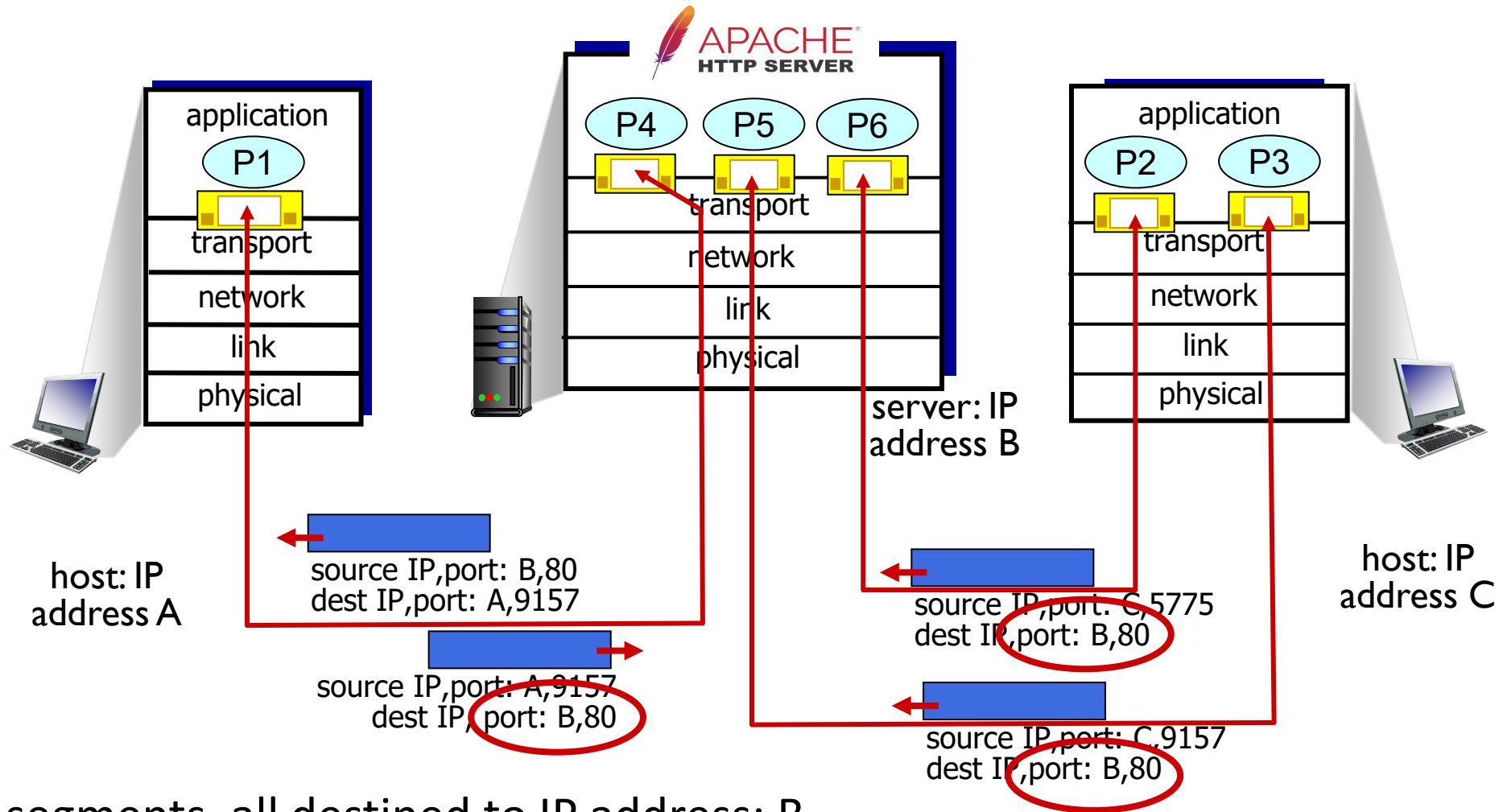
```
DatagramSocket mySocket1 =  
new DatagramSocket (5775) ;
```



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

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- **Connectionless transport: UDP**
- Connection-oriented transport: TCP
- TCP congestion control



UDP: User Datagram Protocol

- “no frills,” “bare bones”
Internet transport protocol
- “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

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
 - UDP can blast away as fast as desired!
 - can function in the face of congestion

UDP: User Datagram Protocol

- UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
 - SNMP
 - HTTP/3
- if reliable transfer needed over UDP (e.g., HTTP/3):
 - add needed reliability at application layer
 - add congestion control at application layer

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel

ISI

28 August 1980

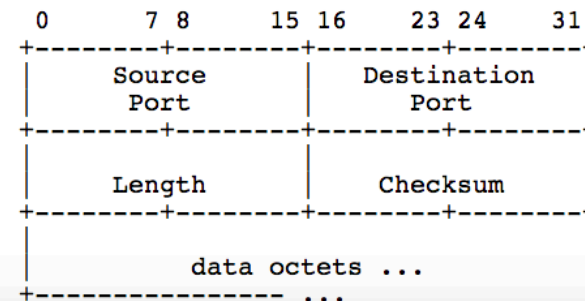
User Datagram Protocol

Introduction

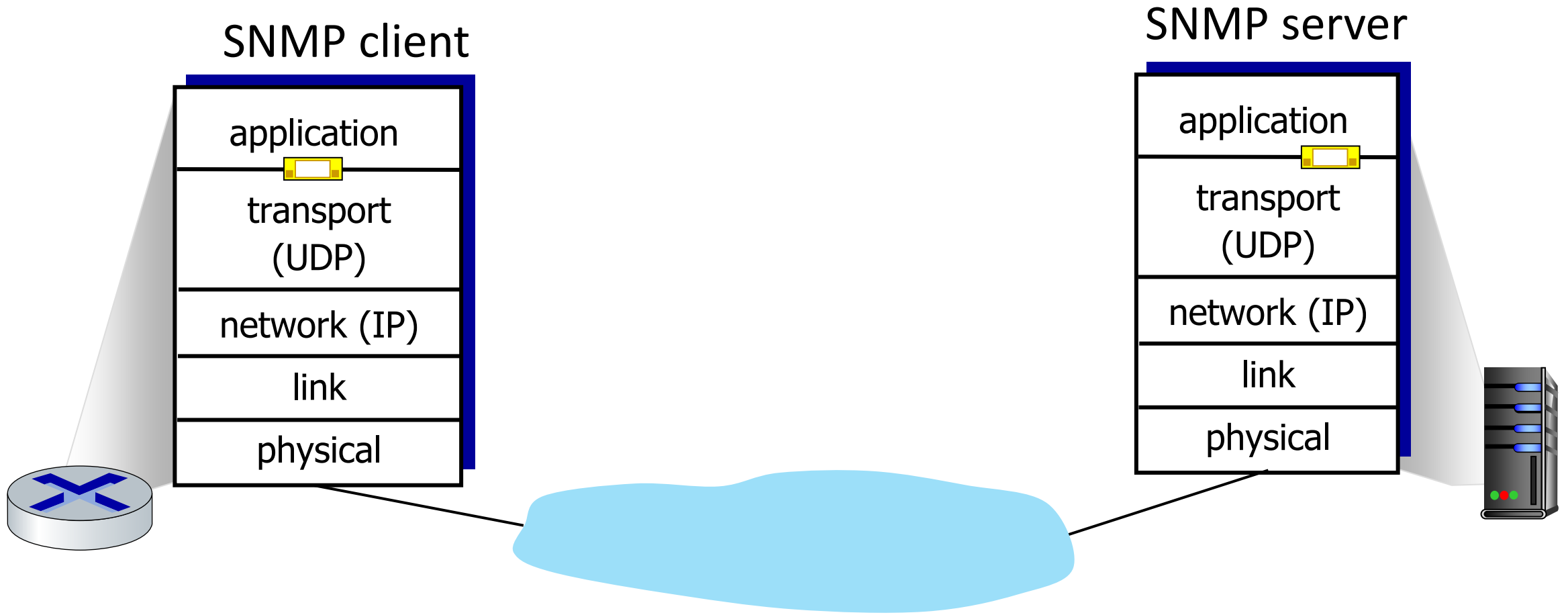
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

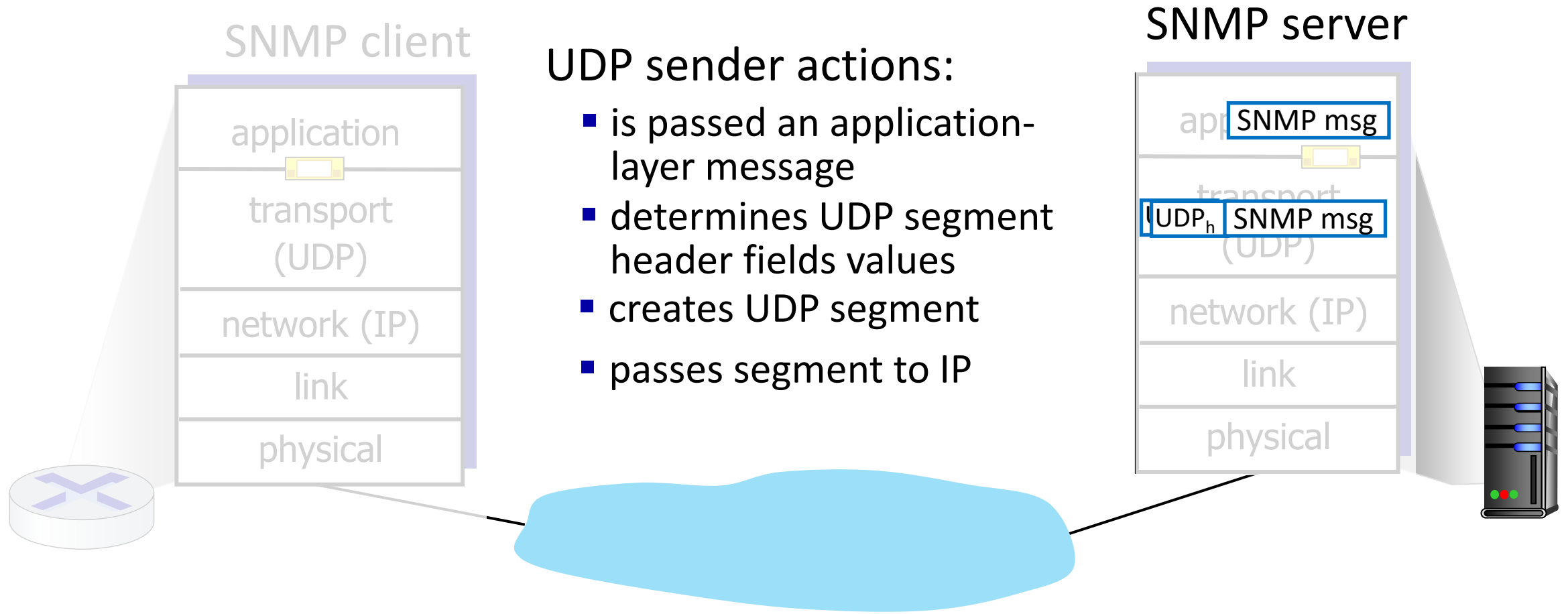
Format



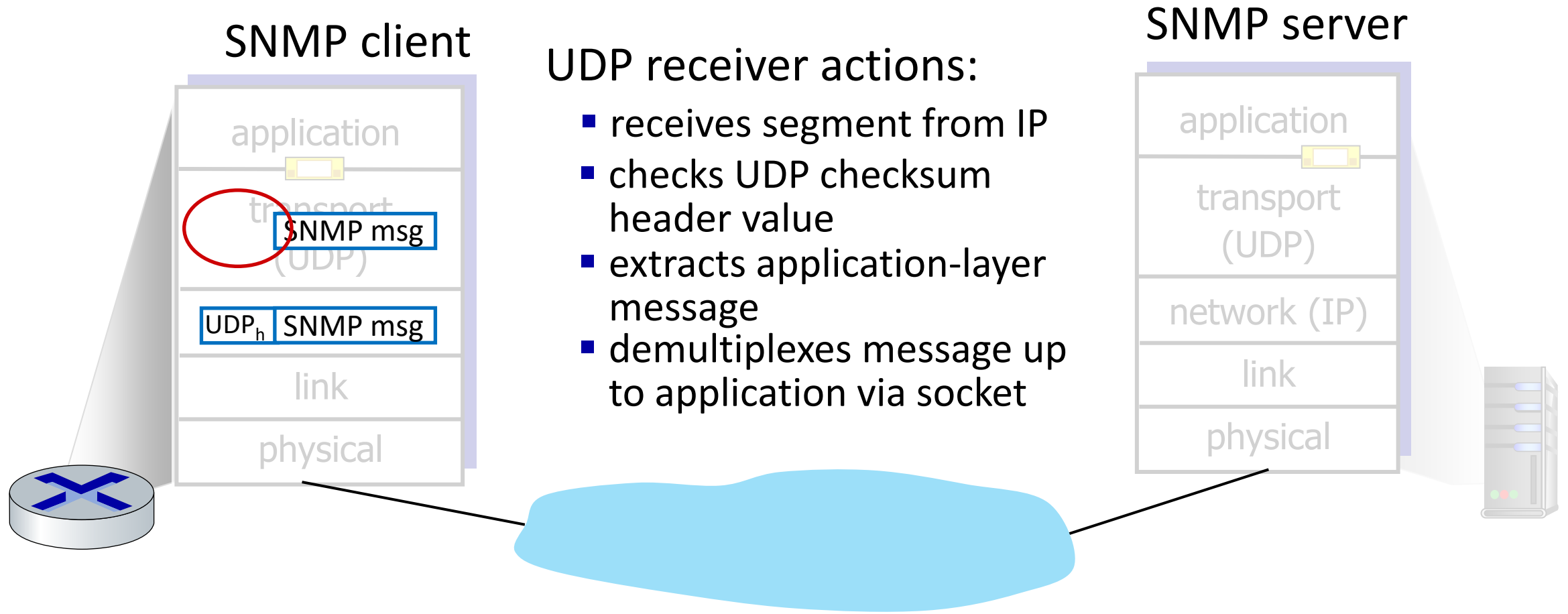
UDP: Transport Layer Actions



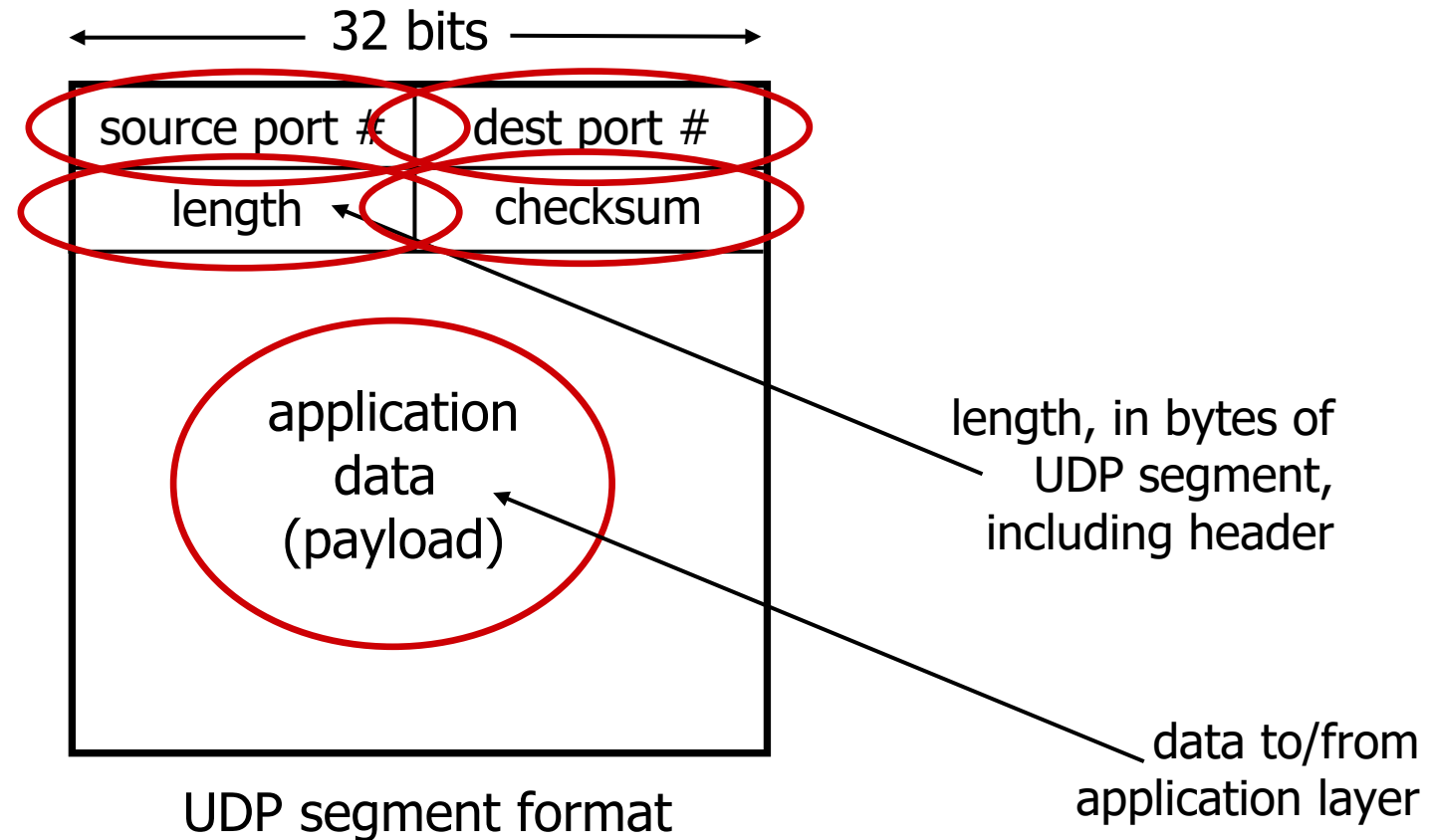
UDP: Transport Layer Actions



UDP: Transport Layer Actions

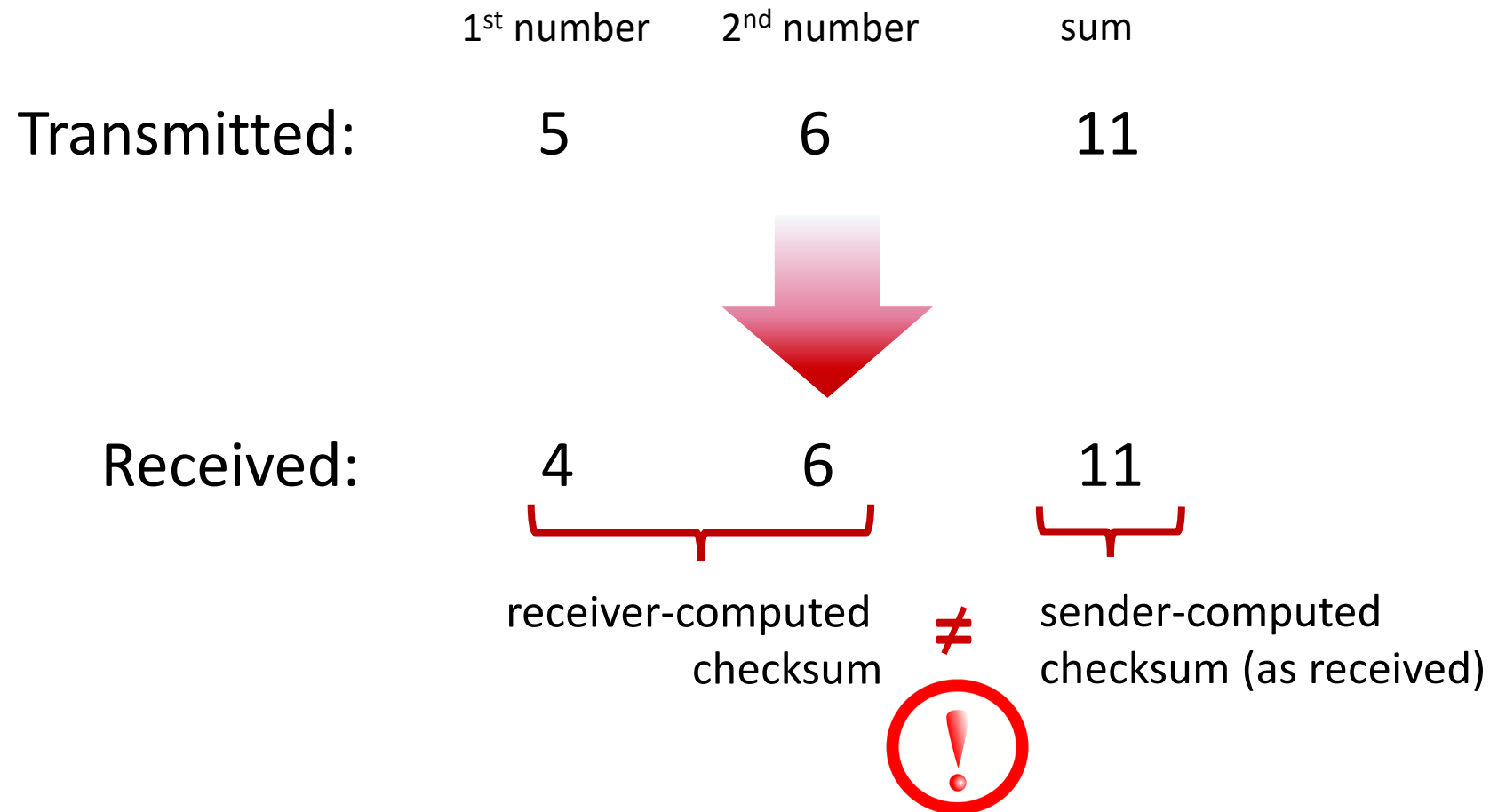


UDP segment header



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	0	1	1

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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!

Summary: UDP

- “no frills” protocol:
 - segments may be lost, delivered out of order
 - best effort service: “send and hope for the best”
- UDP has its plusses:
 - no setup/handshaking needed (no RTT incurred)
 - can function when network service is compromised
 - helps with reliability (checksum)