COMP 6461

Computer Networks & Protocols

Winter 2023

Dr. Abdelhak Bentaleb

Lecture 5a

Transport Layer (Part 1)

Chapter 3: Transport-layer

Overview/roadmap:

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer

- Connection-oriented transport: TCP
- TCP congestion control

Chapter 3: Transport-layer

After this chapter, you are expected to:

- 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

Layered Internet protocol stack

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *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, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"

application
Transport
network
link
physical
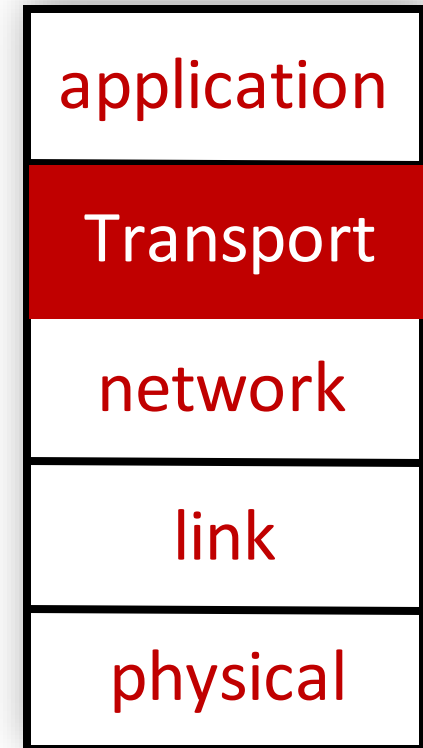
Transport services and protocols

▪ 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
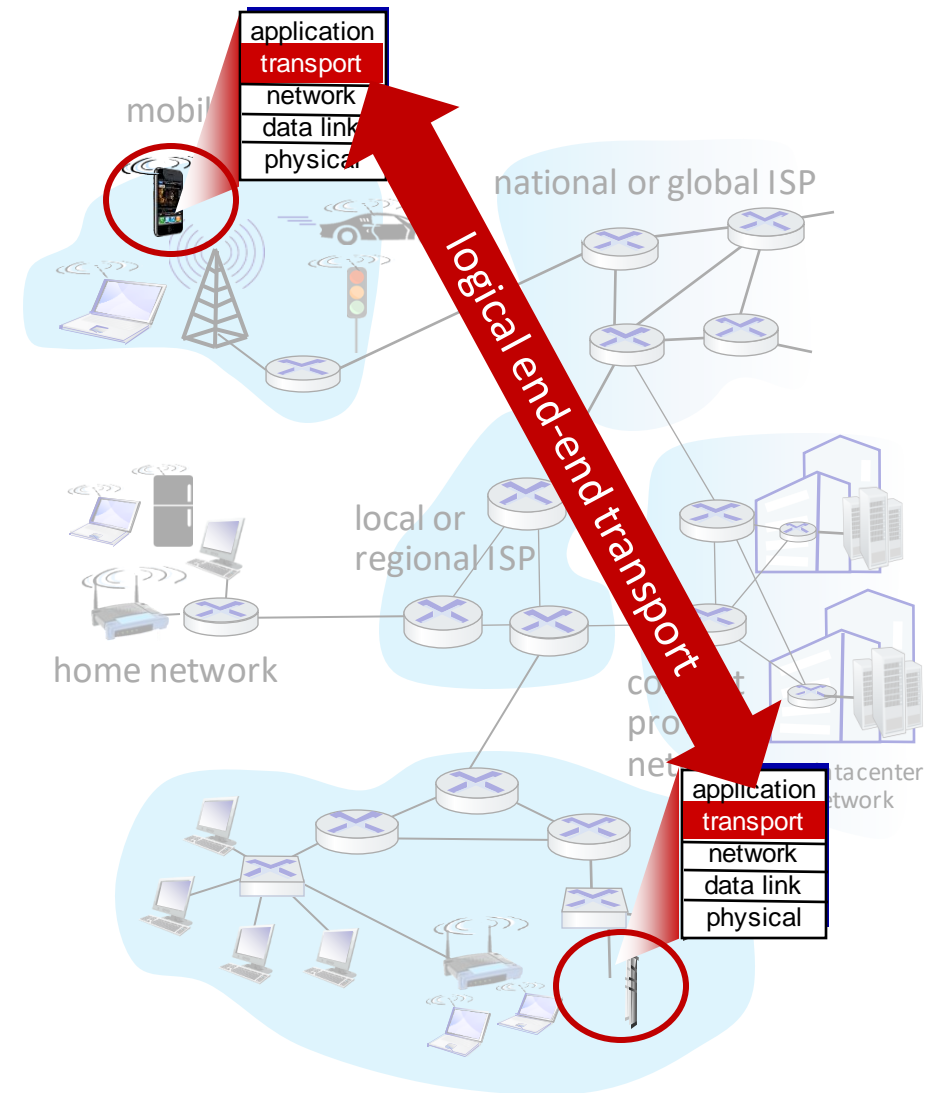
Transport vs. network layer services and protocols

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 protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Transport vs. network layer services and protocols

- **network layer:** logical communication between *hosts*

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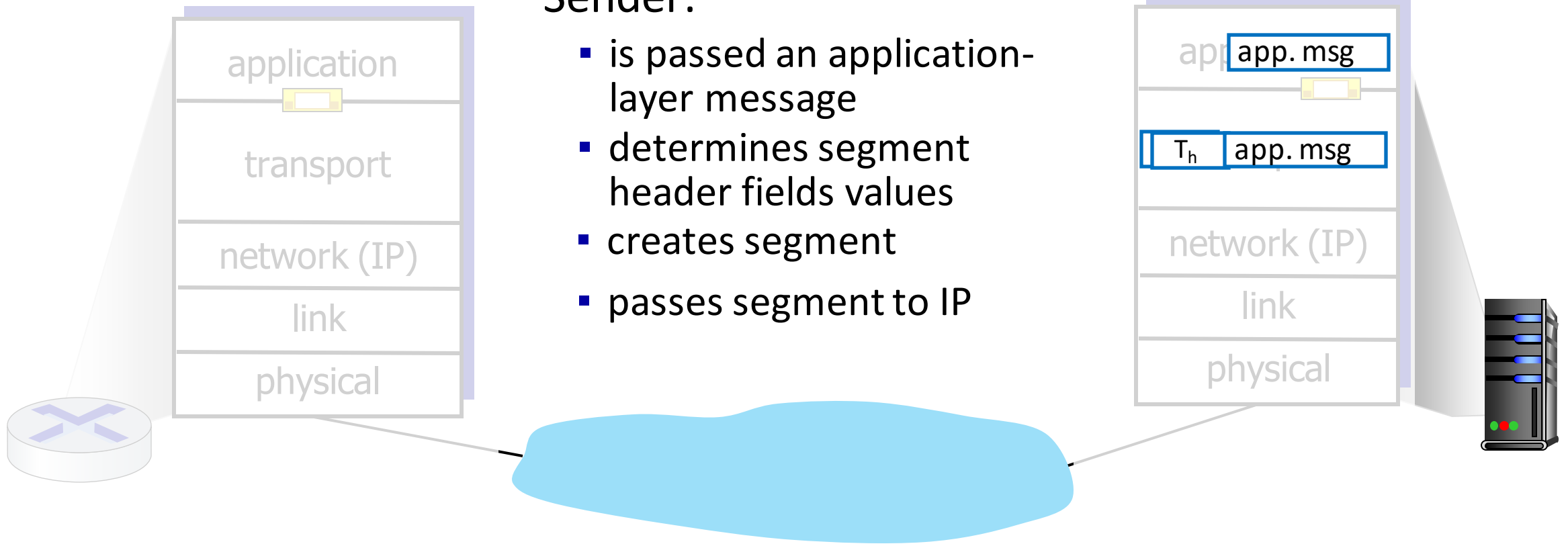 - relies on, enhances, network layer services

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 protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Transport Layer Actions

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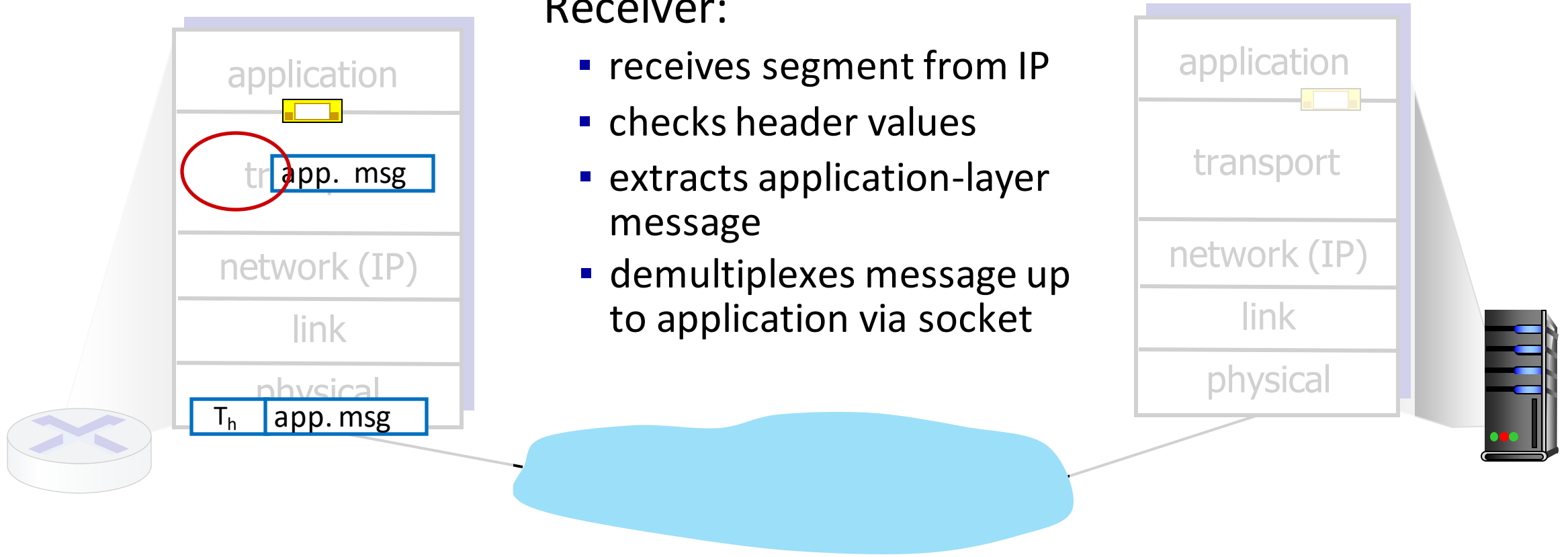
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP

app. msg

T_h | app. msg

application

transport

network (IP)

link

physical

app...

network (IP)

link

physical

Transport Layer Actions

Receiver:

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

application

transport

app. msg

network (IP)

link

physical

T_h app. msg

application

transport

network (IP)

link

physical
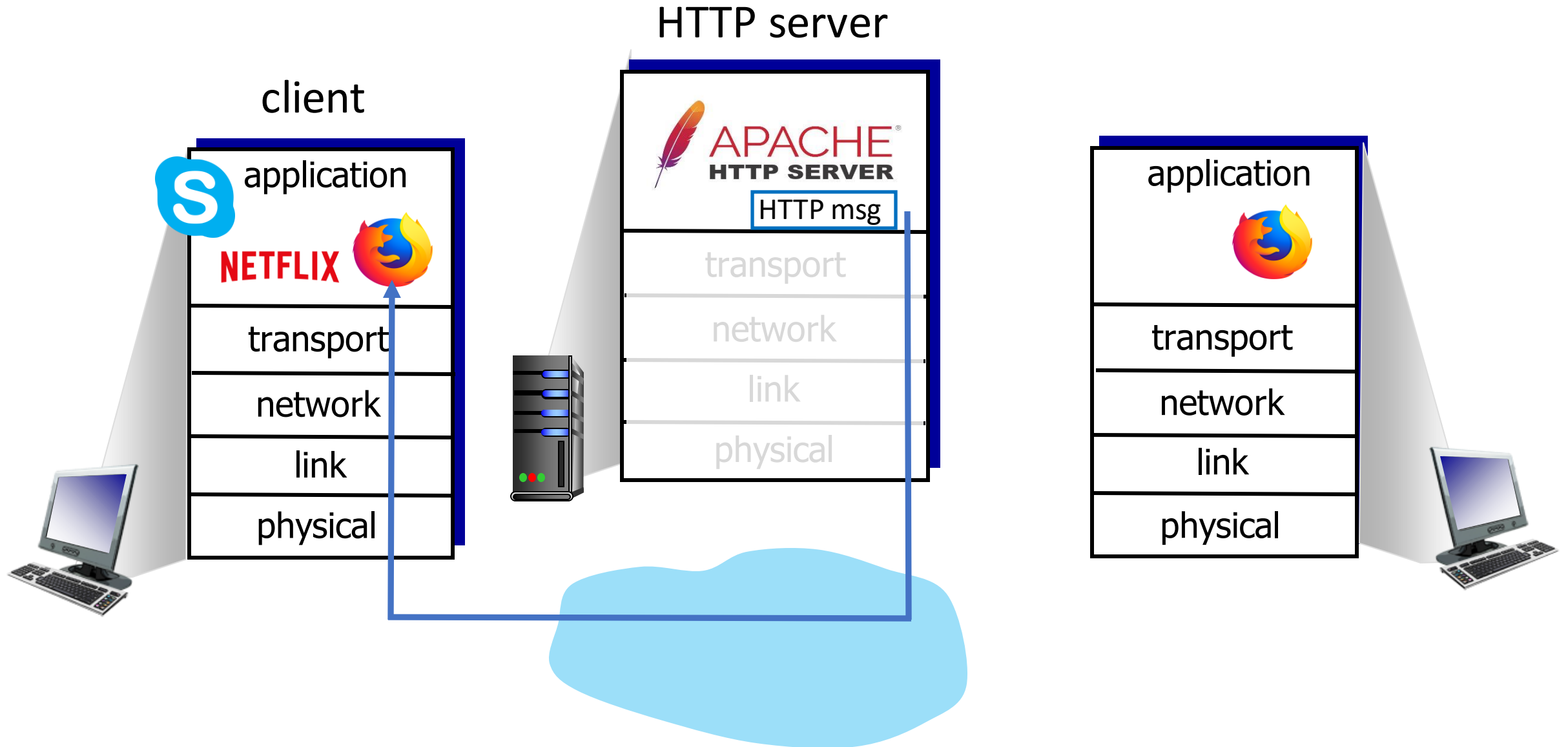
Two principal Internet transport protocols

- **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:
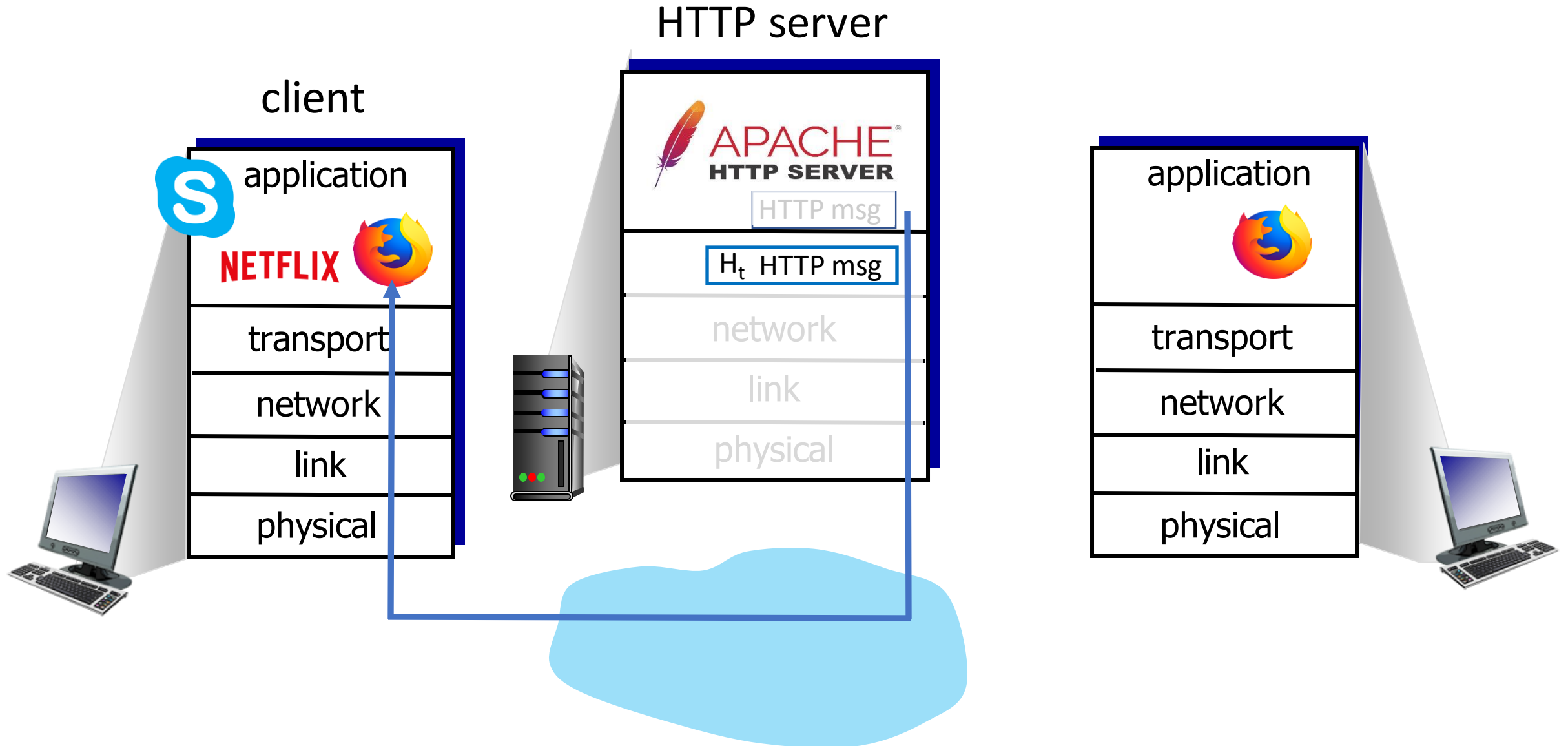 - delay guarantees
  - bandwidth guarantees
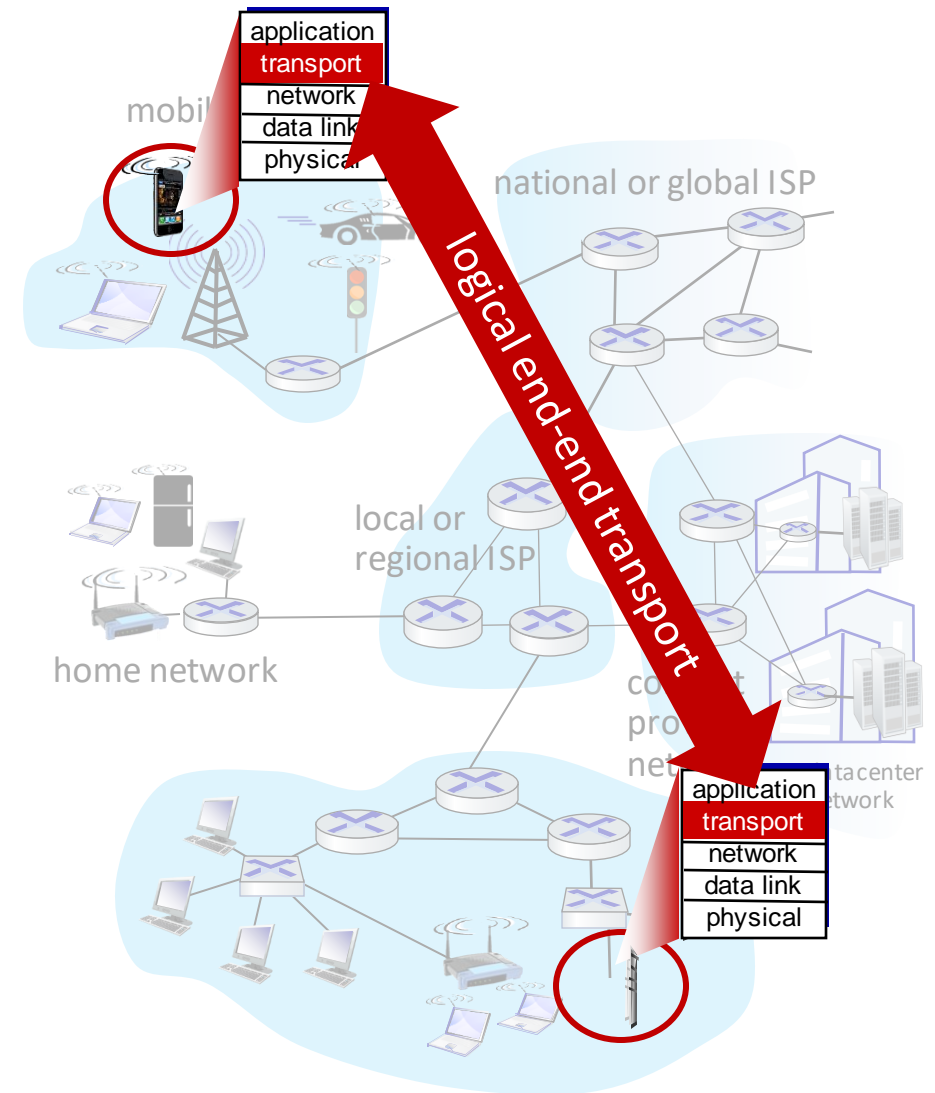
Chapter 3: roadmap

- Transport-layer services
- **Multiplexing and demultiplexing**
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Multiplexing/demultiplexing

client

HTTP server

application

NETFLIX

transport

network

link

physical

APACHE HTTP SERVER

HTTP msg

transport

network

link

physical

application

transport

network

link

physical

Multiplexing/demultiplexing

Multiplexing/demultiplexing

HTTP server

client

application

NETFLIX

transport

network

link

physical

HTTP msg

H_t HTTP msg

$H_n H_t$ HTTP msg

link

physical

application

transport

network

link

physical

Multiplexing/demultiplexing

HTTP server

client

application

transport

network

link

physical

transport

network

link

physical

application

transport

network

link

physical

H_n H_t HTTP msg
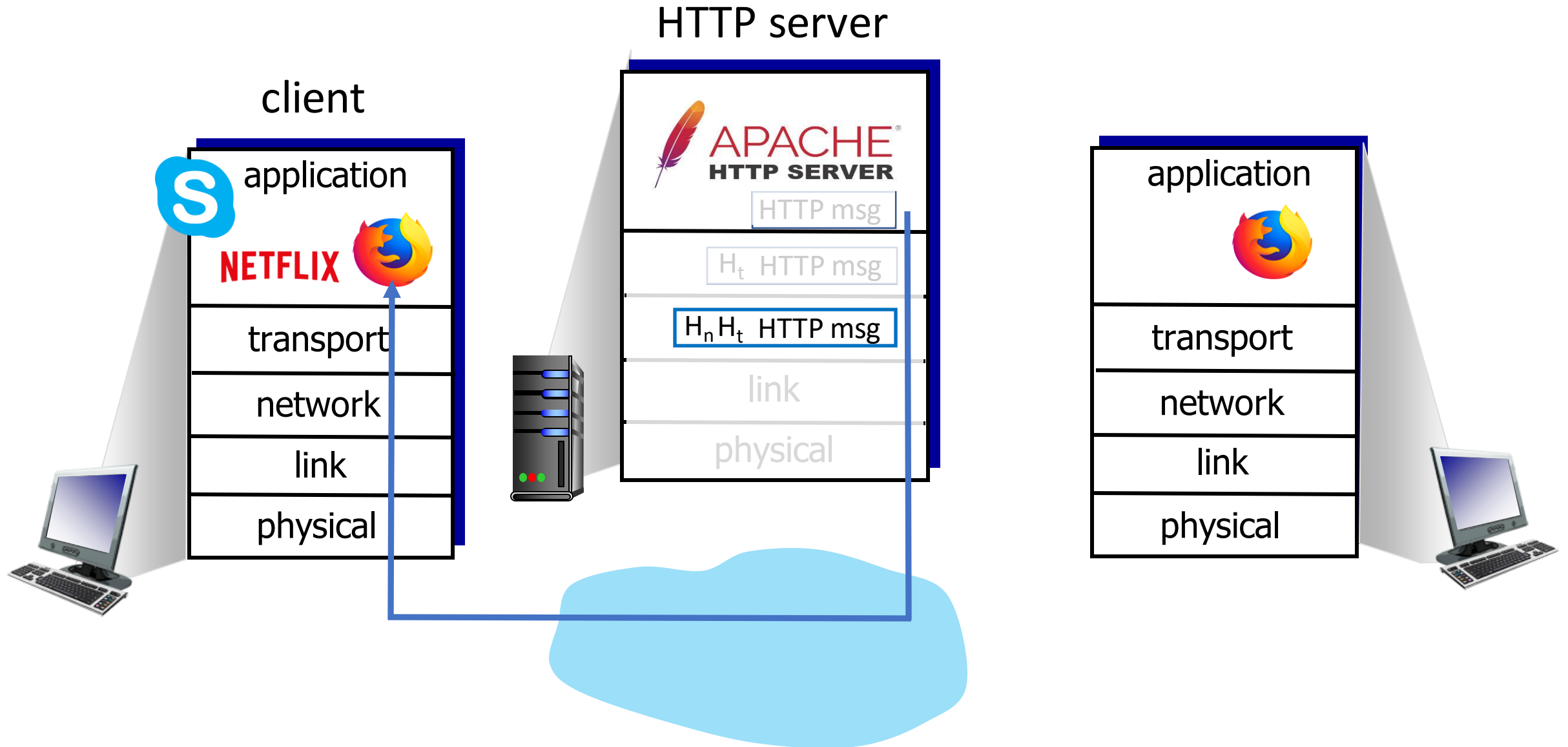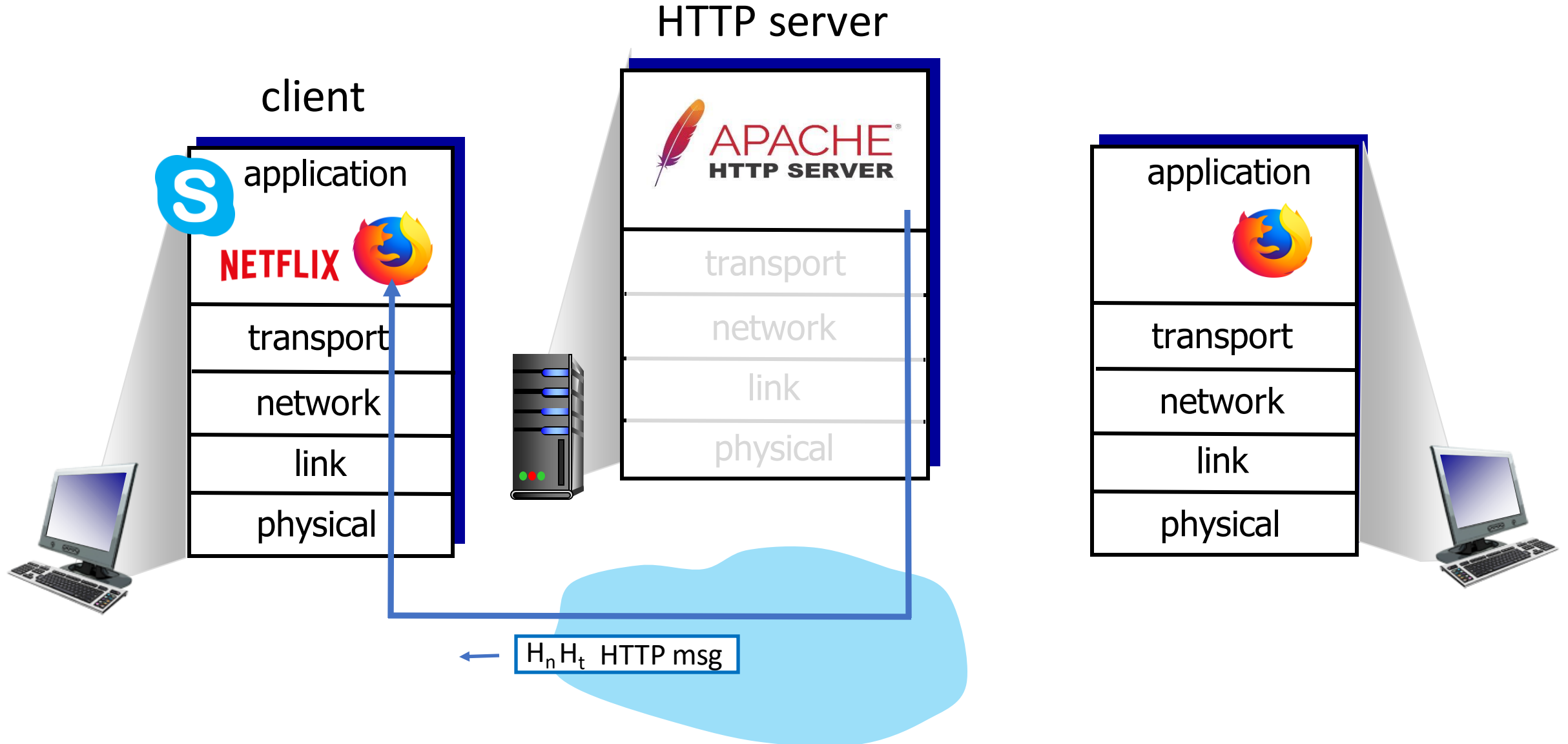
Multiplexing/demultiplexing
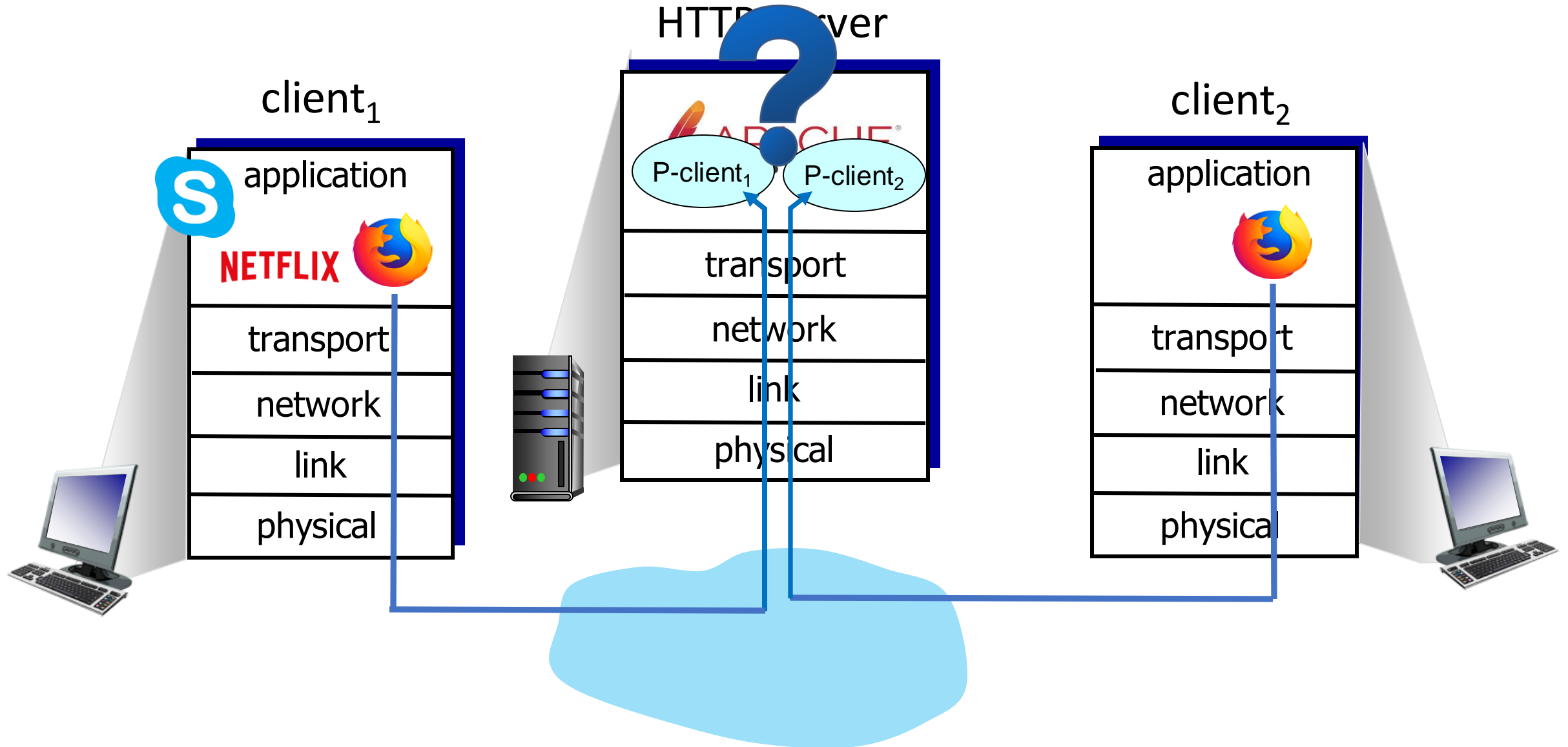
Multiplexing/demultiplexing

multiplexing at sender:

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

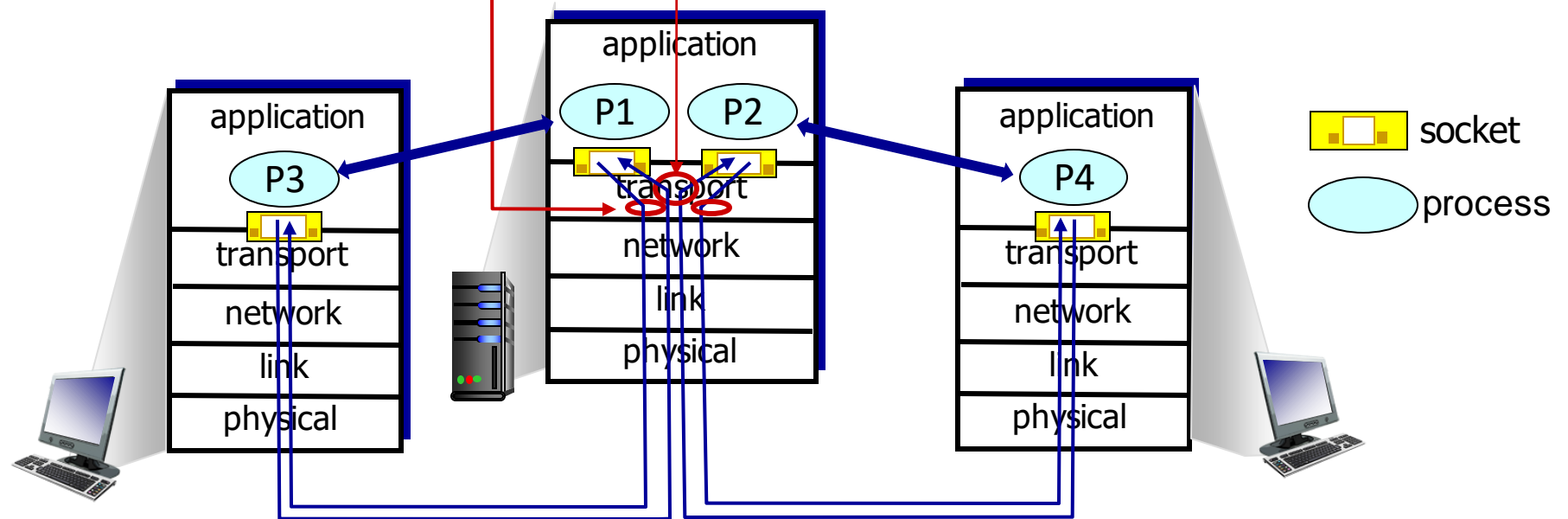*demultiplexing at receiver:*

use header info to deliver received segments to correct socket

18

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

32 bits

| source port # | dest port # |

other header fields

application
data
(payload)
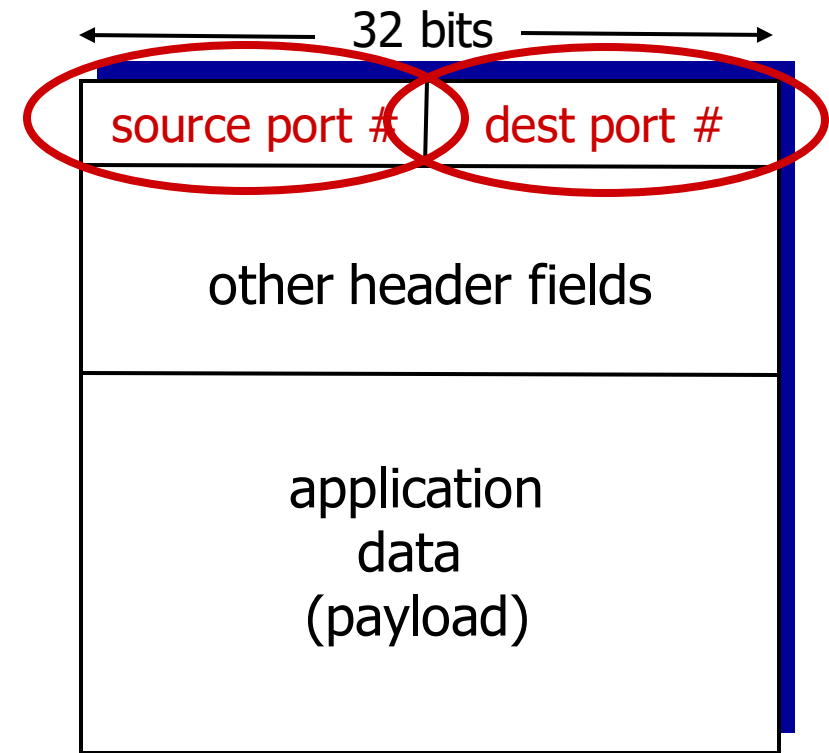
TCP/UDP segment format

19

Connectionless demultiplexing

Recall:

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

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

application

P3

transport

network

link

physical

application

P1

transport

network

link

physical

application

P4

transport

network

link

physical

source port: 6428
dest port: 9157

source port: ?
dest port: ?

source port: 9157
dest port: 6428

source port: ?
dest port: ?
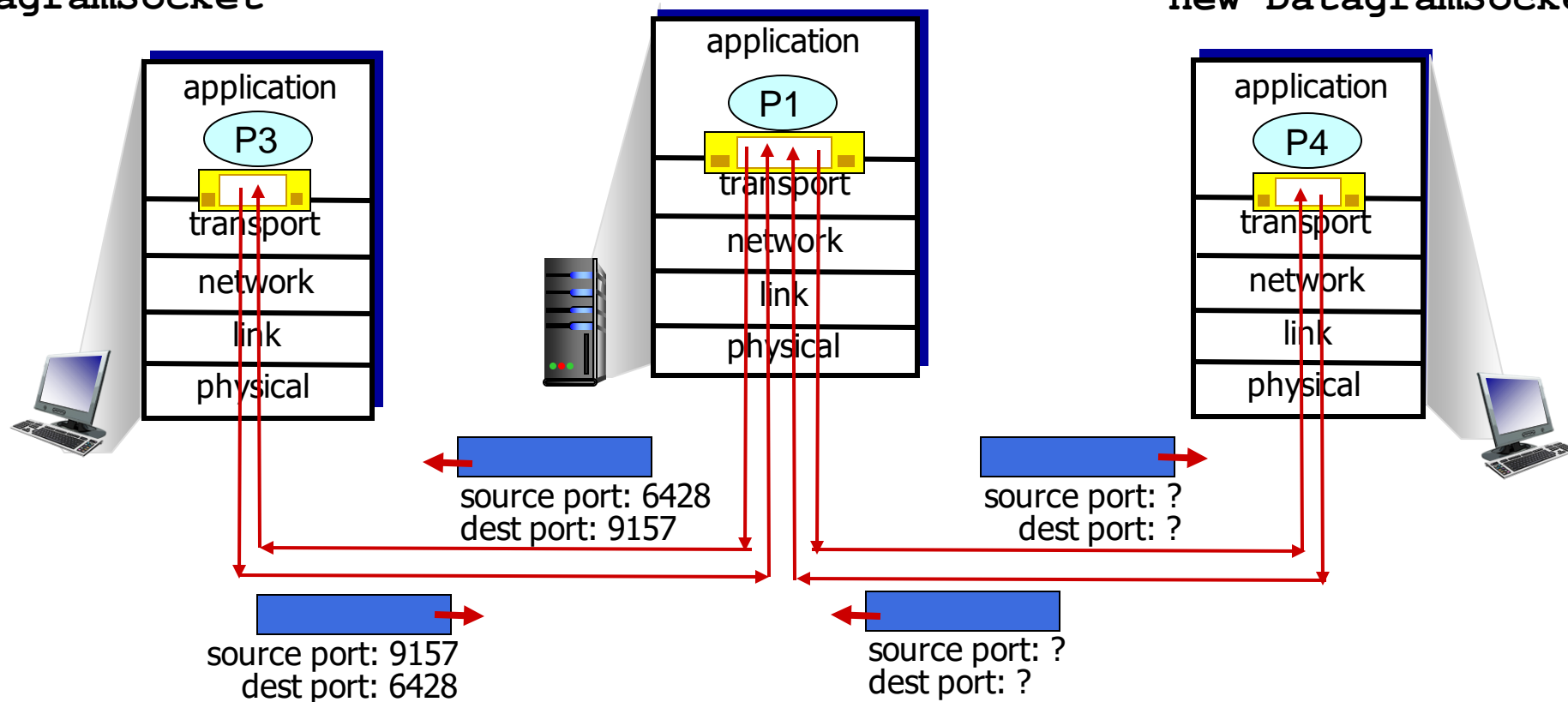
Connection-oriented demultiplexing

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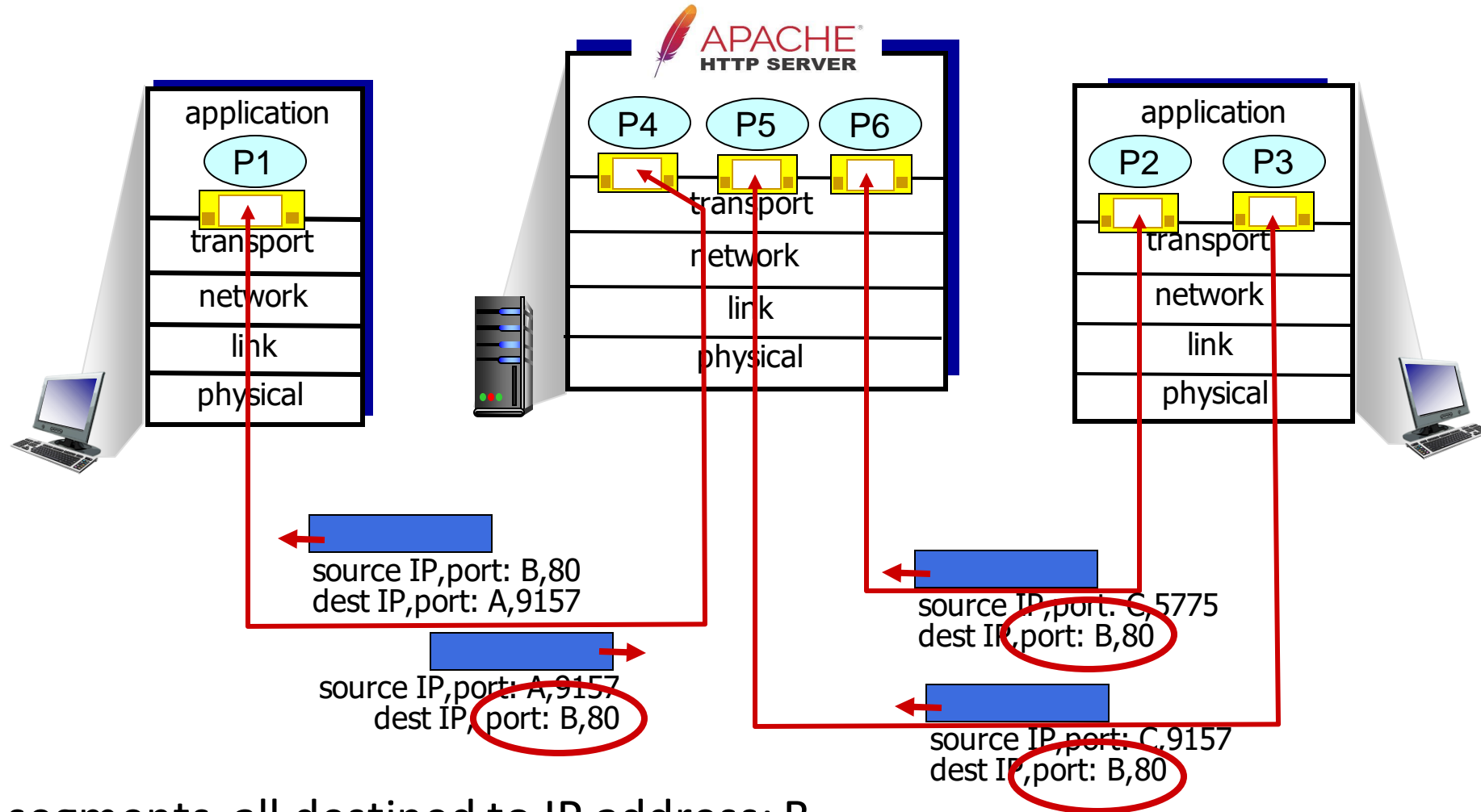
Three segments, all destined to IP address: B,
 dest port: 80 are demultiplexed to *different* sockets

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**
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- 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]

                        User Datagram Protocol
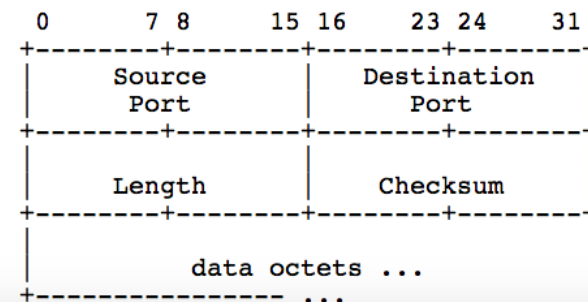   ----------------------

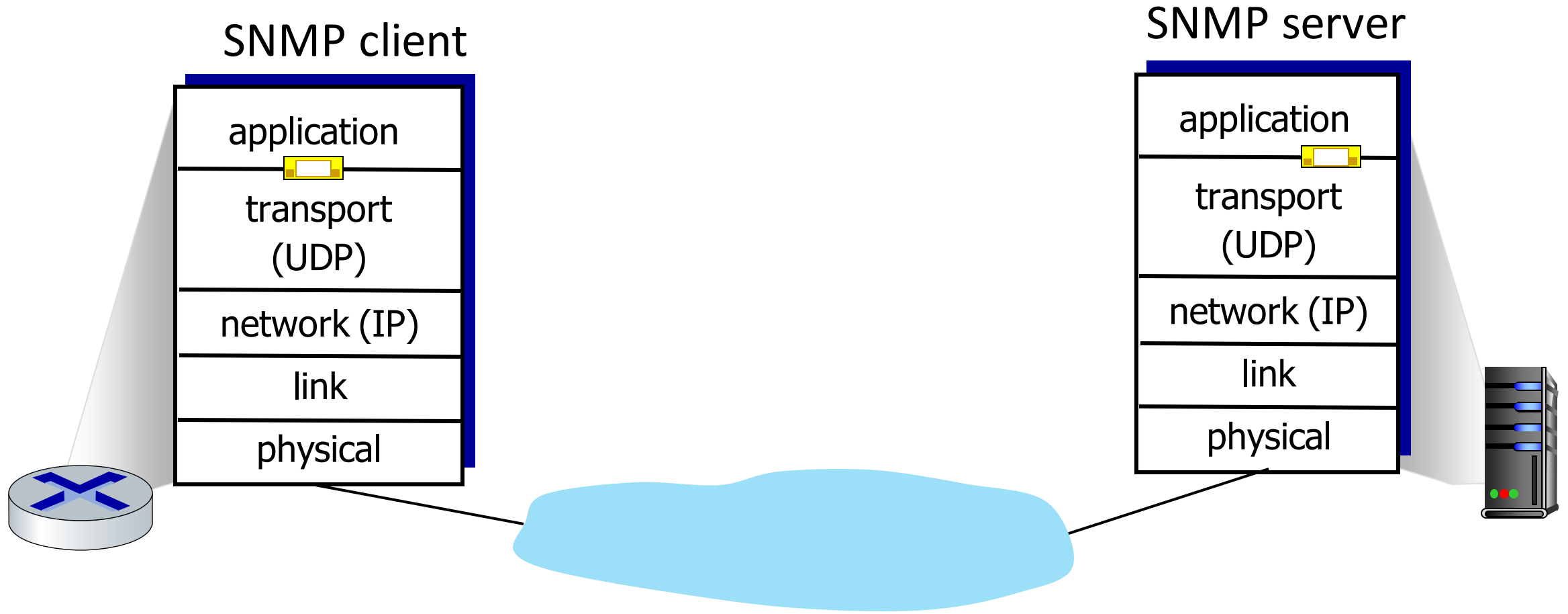Introduction
----

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

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+--------------- ...
```

UDP: Transport Layer Actions

SNMP client

SNMP server

application

transport (UDP)

network (IP)

link

physical

application

transport (UDP)

network (IP)

link

physical

UDP: Transport Layer Actions

SNMP client

SNMP server

UDP sender actions:

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

application

transport
(UDP)

network (IP)

link

physical

application

SNMP msg

transport
(UDP)

UDP_h | SNMP msg

network (IP)

link

physical

UDP: Transport Layer Actions

SNMP client

SNMP server

UDP receiver actions:

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

UDP segment header

UDP segment format

length, in bytes of UDP segment, including header

data to/from application layer

UDP checksum

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

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

receiver-computed checksum ≠ sender-computed checksum (as received)

UDP checksum

Goal: detect "errors" (e.g., 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
```
───

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

───

```
            0  0  0  0  0  0  0  0  0  0  0  0  0  0  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
```

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        0  1
1  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1        1  0
```

wraparound **1** ① 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

sum   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
           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 (congested)
  - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)