COMPUTER NETWORKS

TEAM NETWORKS

Department of Computer Science and Engineering

COMPUTER NETWORKS

Transport Layer

TEAM NETWORKS

Department of Computer Science and Engineering

Transport Layer - Roadmap

3.1 Transport-layer Services

3.2 Multiplexing and Demultiplexing

3.3 Connectionless Transport: UDP

Transport Layer - Roadmap

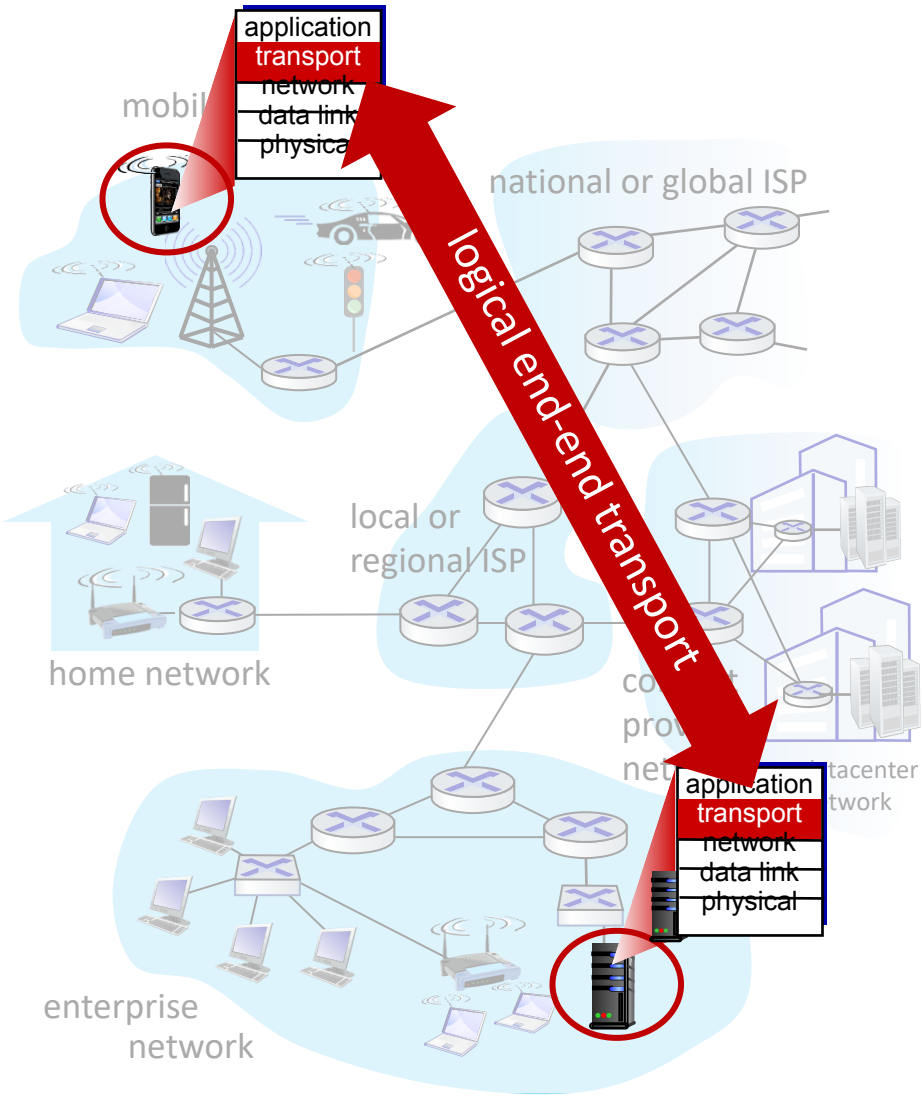
3.1 Transport-layer Services

3.2 Multiplexing and Demultiplexing

3.3 Connectionless Transport: UDP

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:

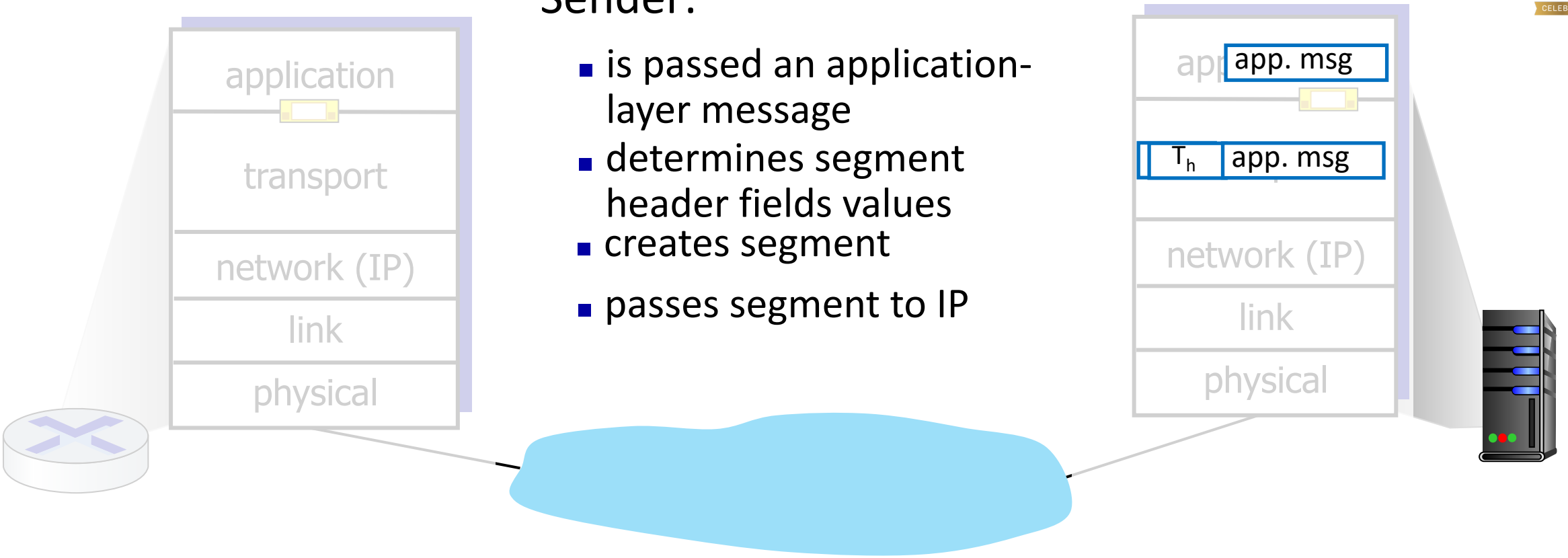
12 kids in Ann's house sending letters to 12 kids in Bill's house:

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

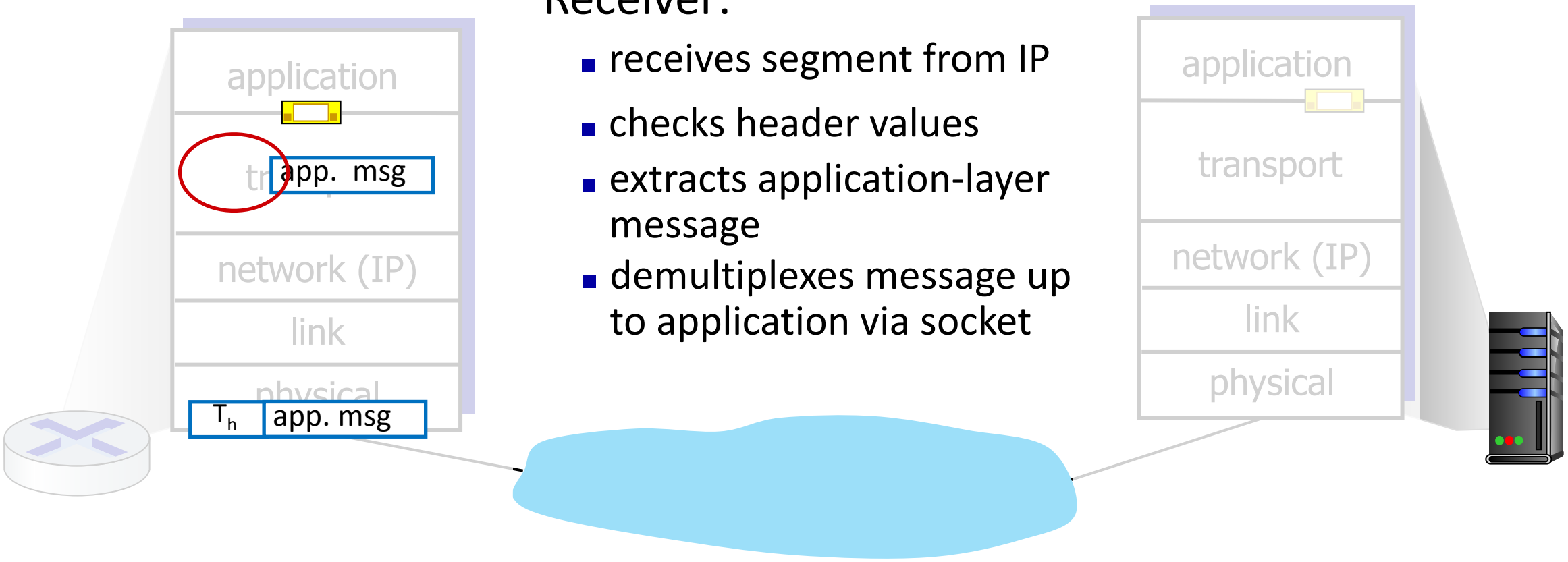
Transport Layer Actions

Sender:

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



Transport Layer Actions

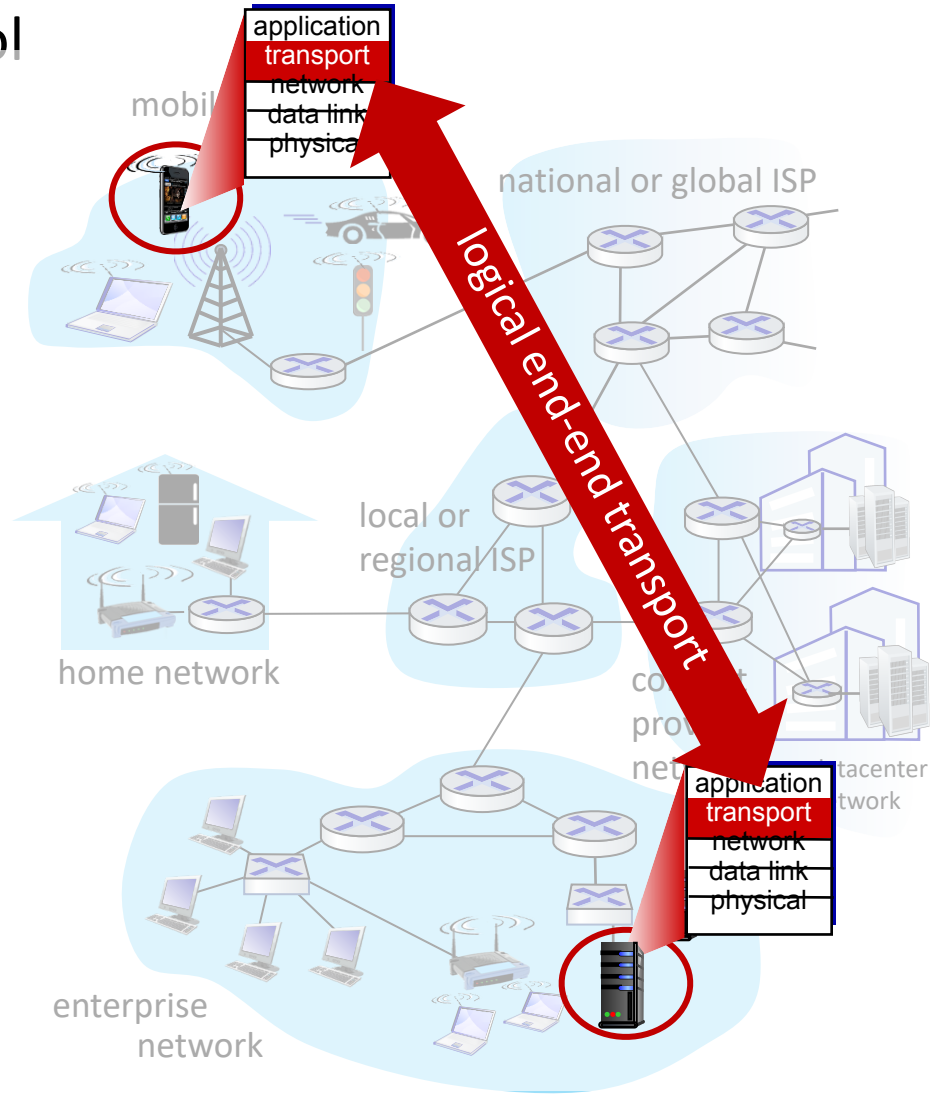


Receiver:

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

Principal Internet Transport Layer protocols

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



Suggested Readings

- Transport Layer – Explained – https://youtu.be/FxFJ1XlWtdI
- Transport Layer Services – IIT Kharagpur –https://youtu.be/8-3CSAkscYU
- Transport Layer – Process to Process Delivery – https://youtu.be/9e4vTcaEYCg

Thank You
For Your Attention

Transport Layer - Roadmap

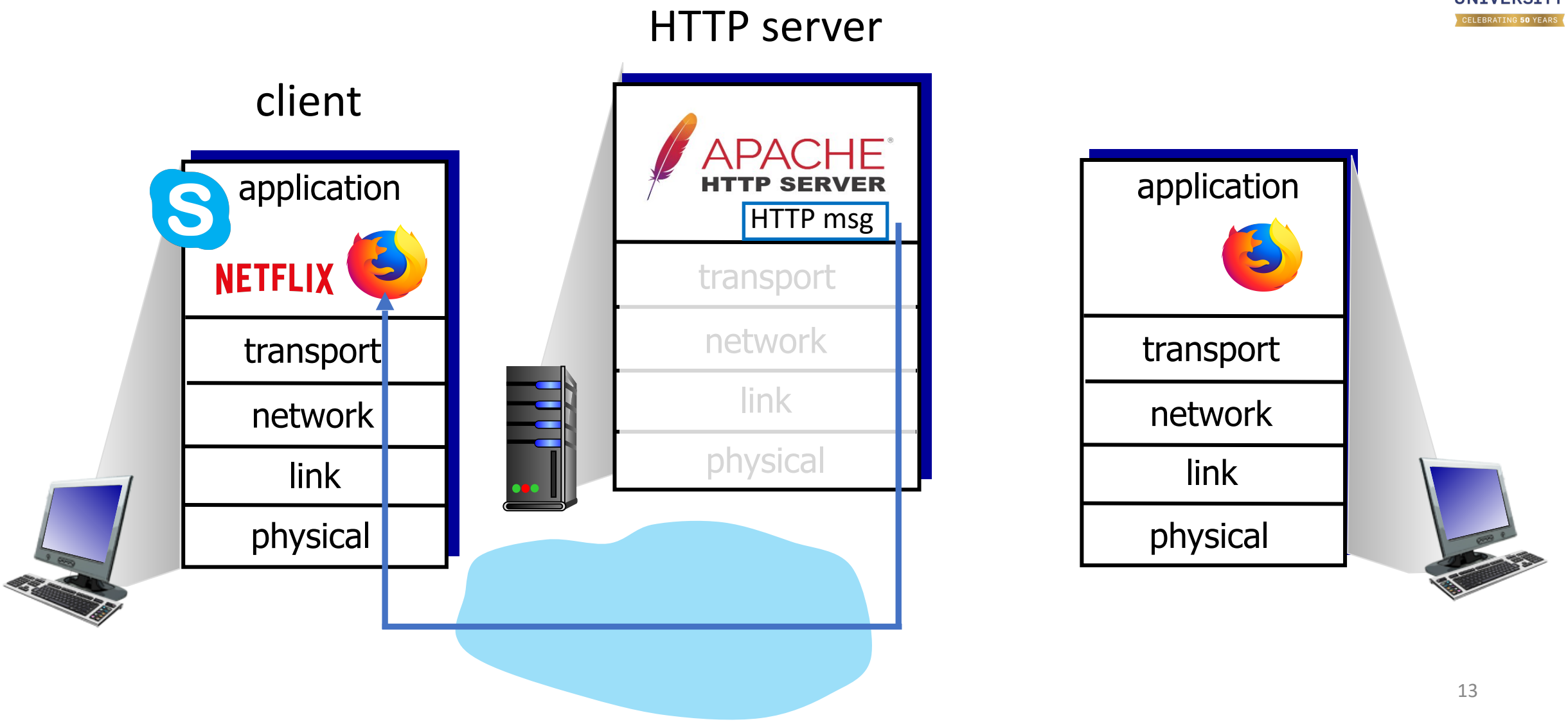
3.1 Transport-layer Services

3.2 Multiplexing and Demultiplexing

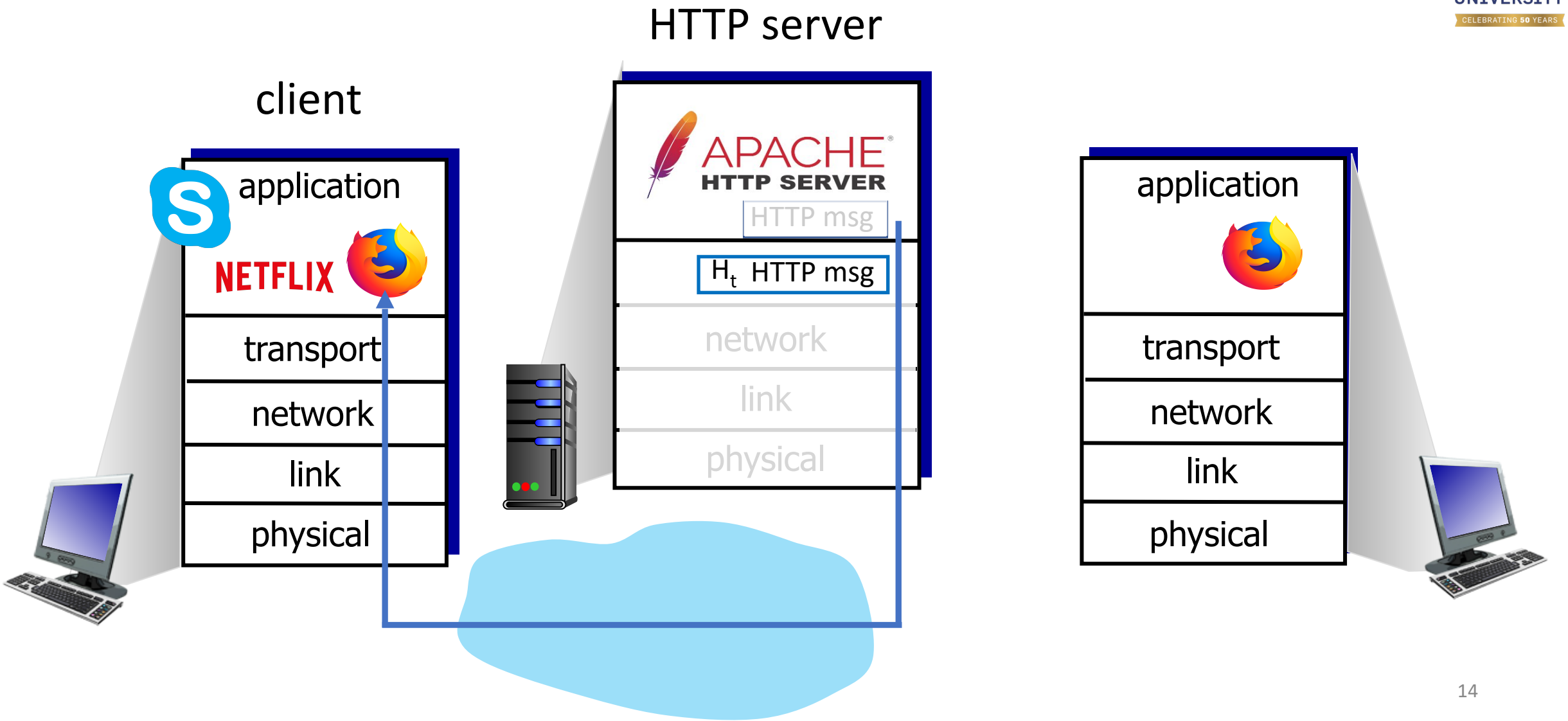
3.3 Connectionless Transport: UDP

Extending host-to-host delivery to process-to-process delivery

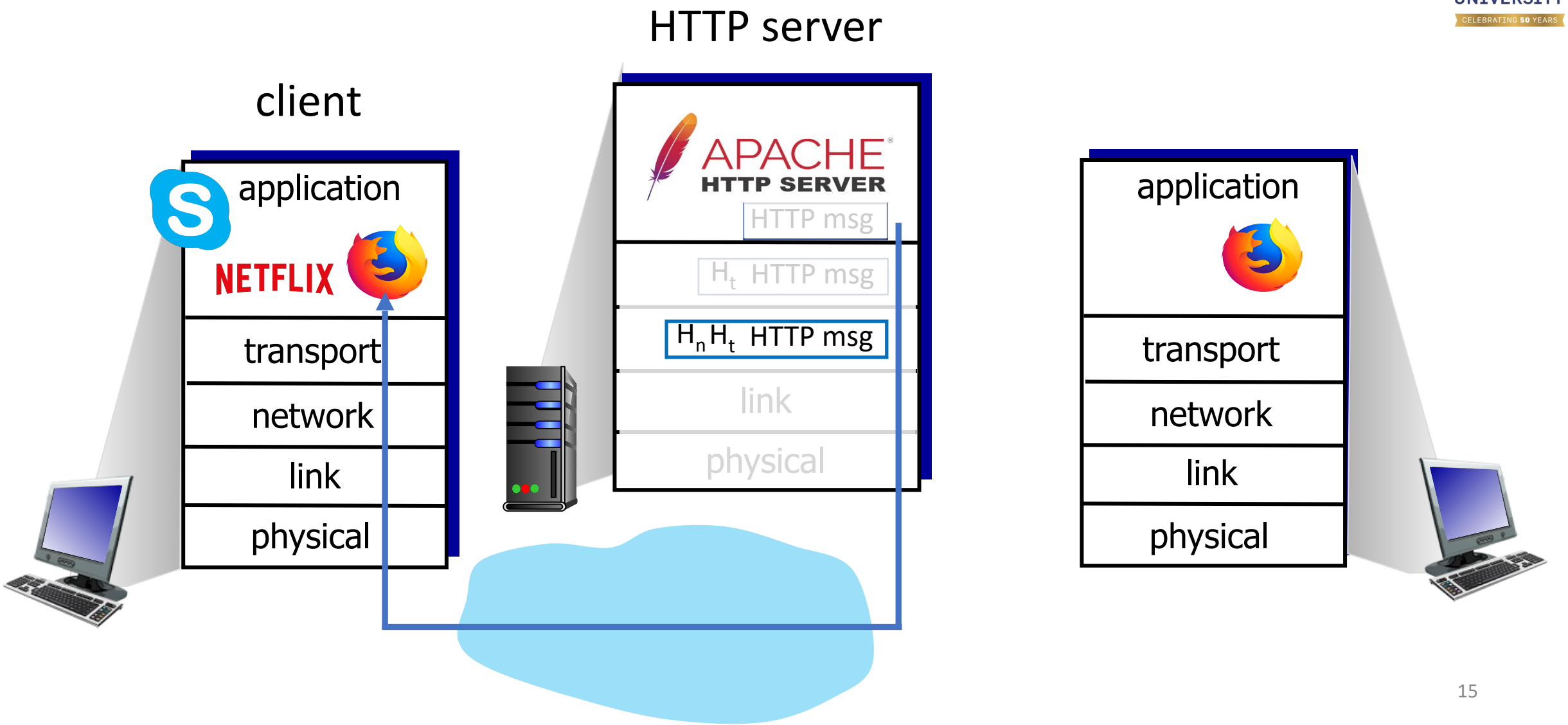
Multiplexing and Demultiplexing



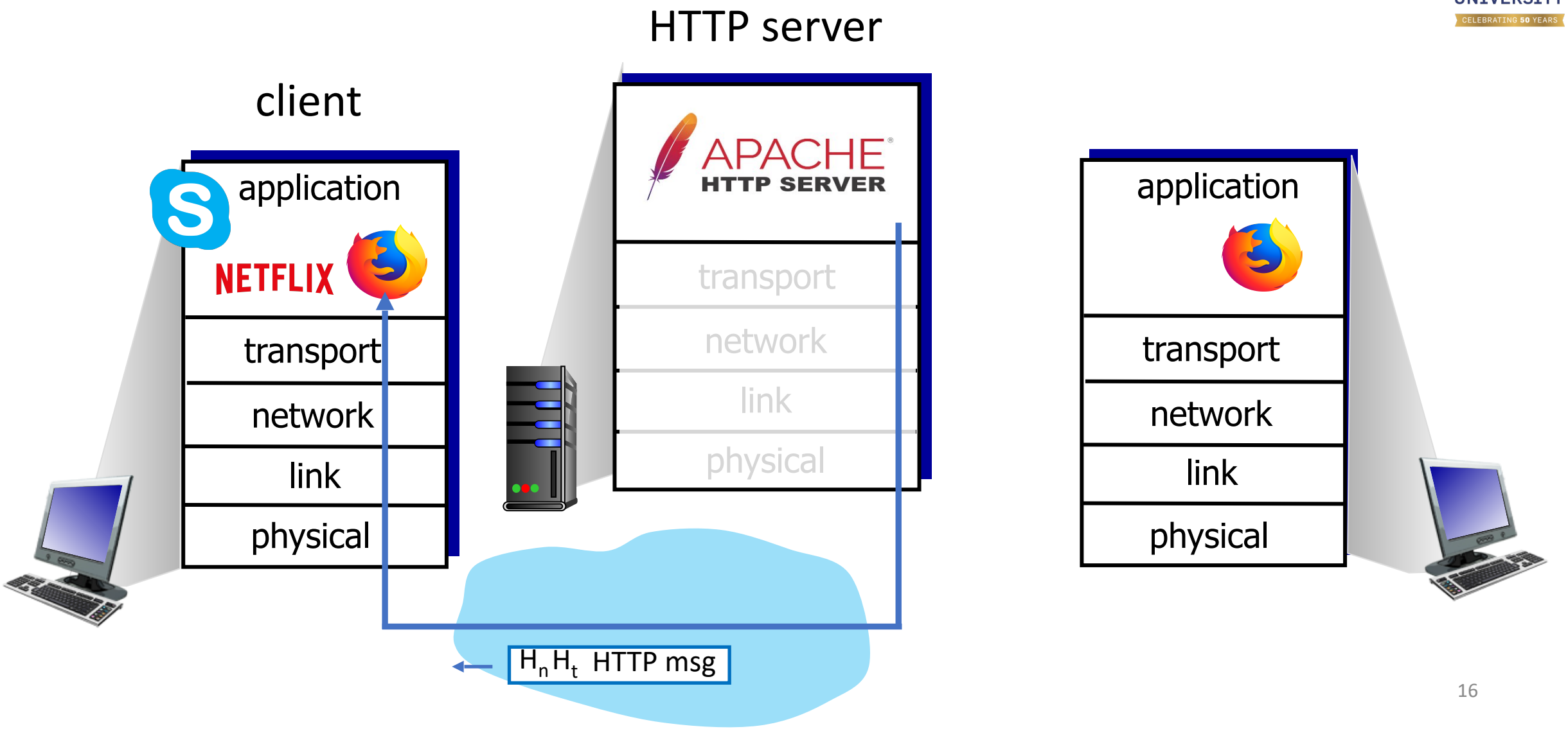
Multiplexing and Demultiplexing



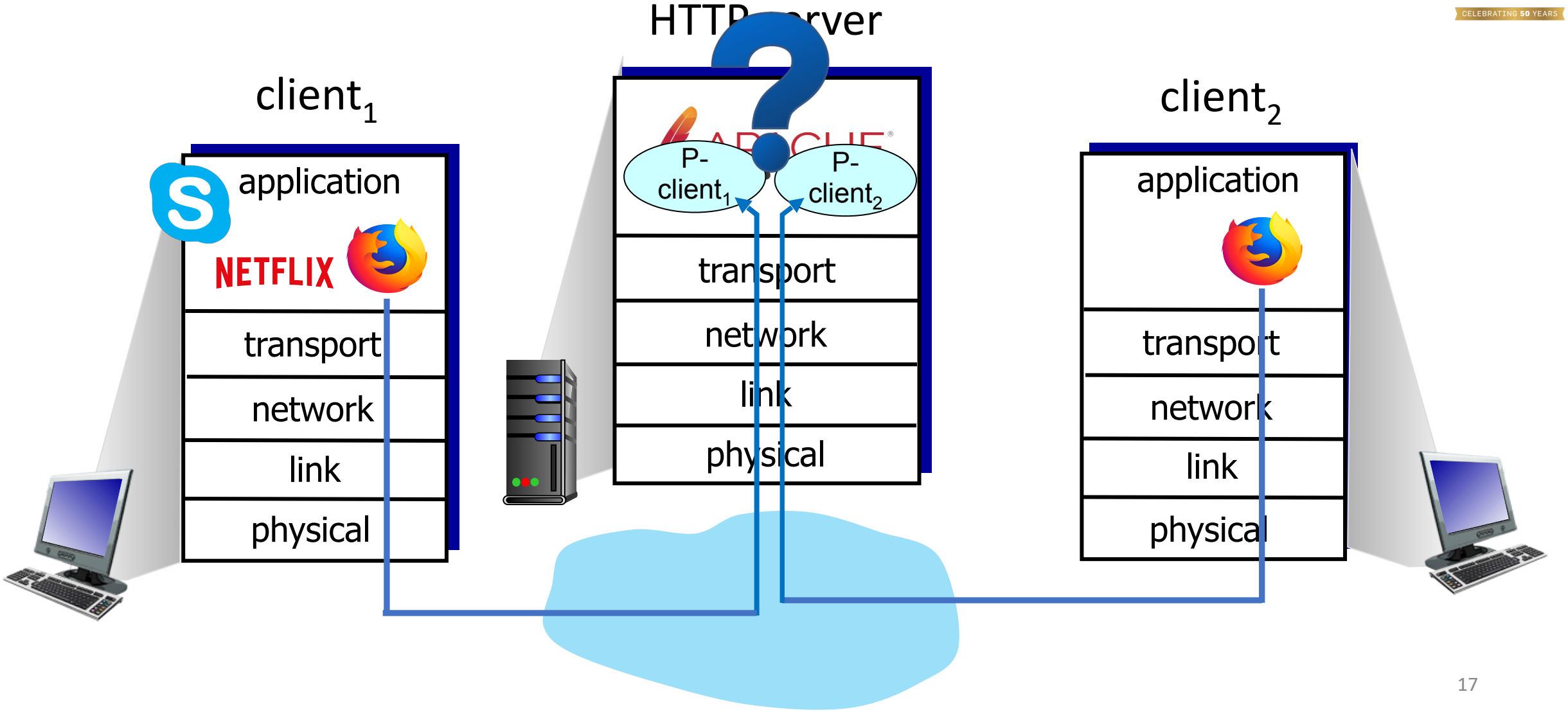
Multiplexing and Demultiplexing



Multiplexing and Demultiplexing



Multiplexing and Demultiplexing



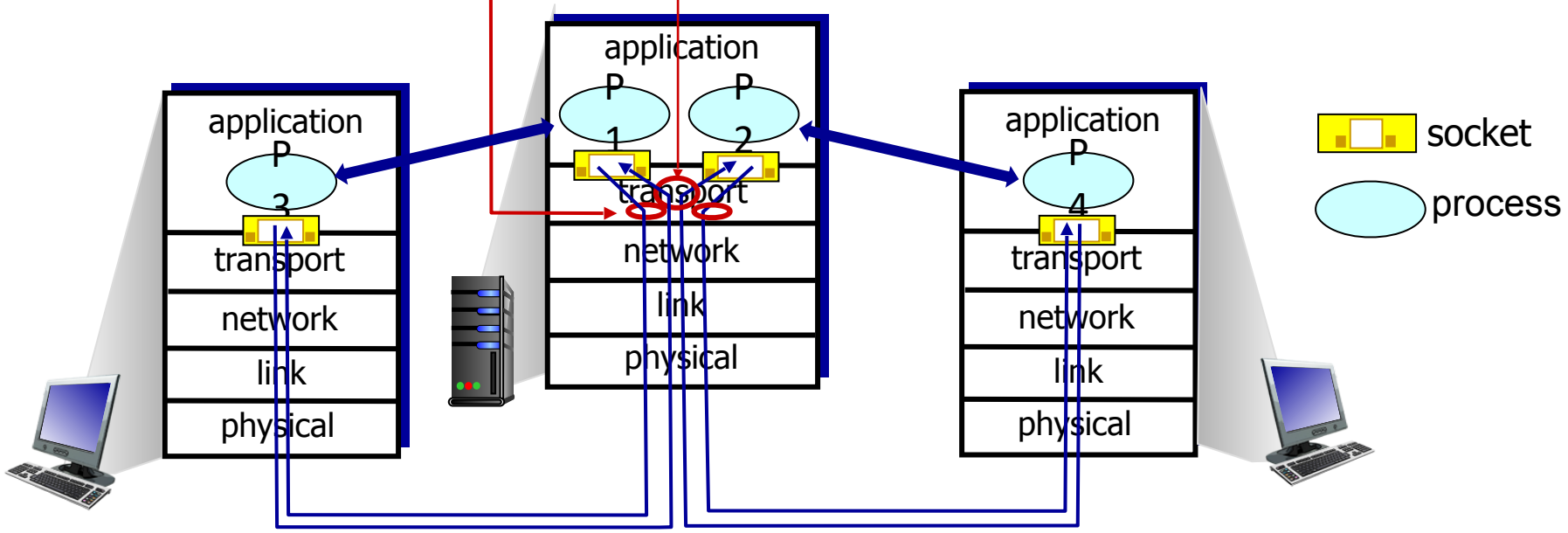
Multiplexing and Demultiplexing

multiplexing at sender:

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

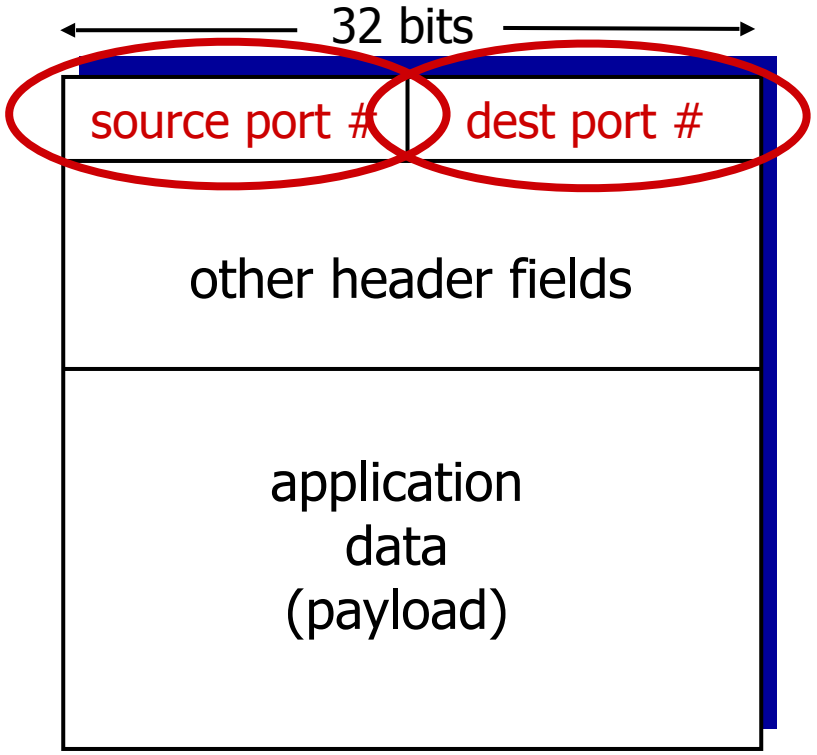
demultiplexing at receiver:

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

- Each port number - ranges from 0 to 65535.
- Port numbers ranging from 0 to 1023 are called **well-known port numbers** (restricted/reserved)

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

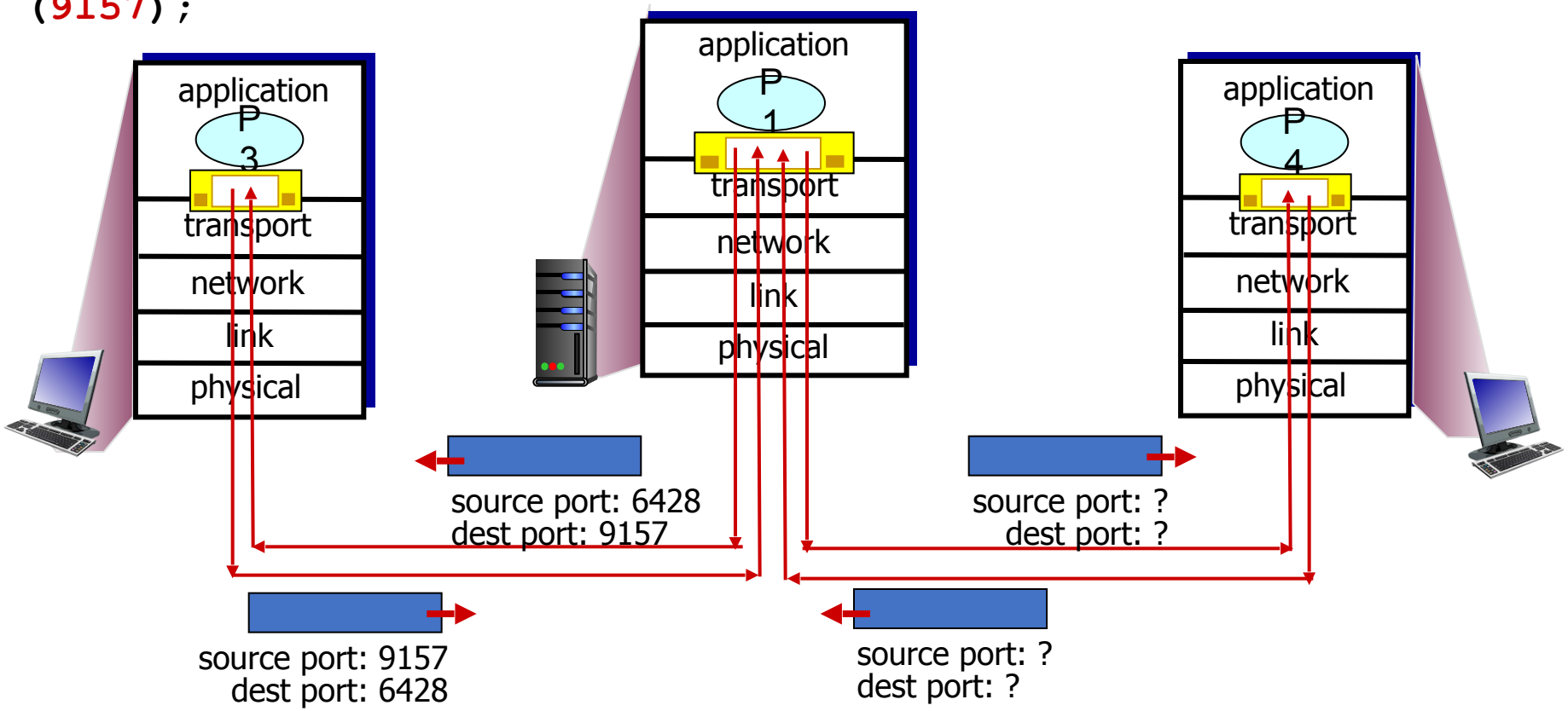
COMPUTER NETWORKS

Connectionless demultiplexing: example

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

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

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

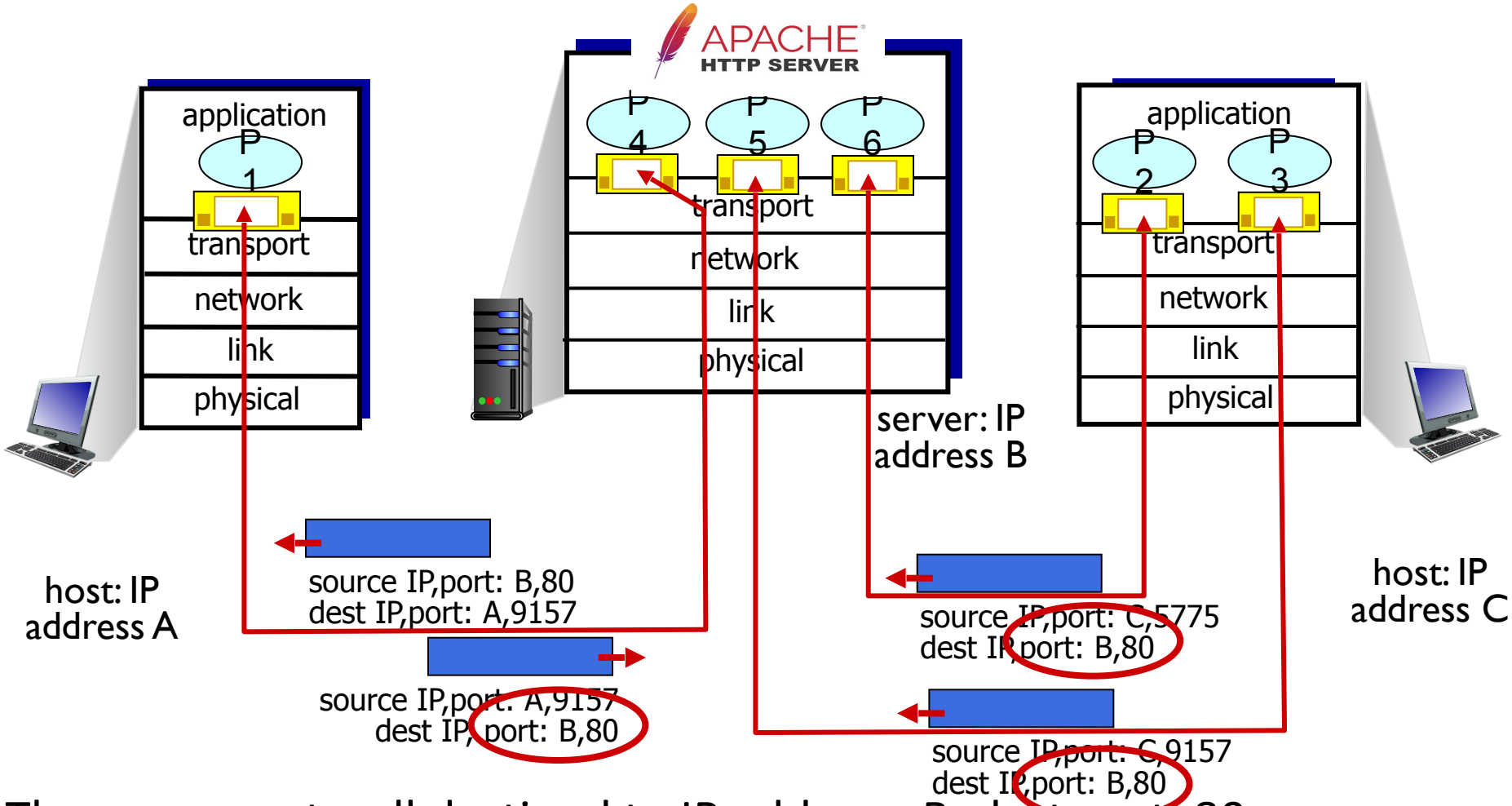


Connection-oriented demultiplexing

- 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 host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

COMPUTER NETWORKS

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

Quiz

How many TCP connections can a server handle?

Suggested Readings

- Transport Layer Multiplexing and Demultiplexing

https://youtu.be/hgWCMry9EYo

- Transport Layer – Process to Process Delivery – https://youtu.be/9e4vTcaEYCg

Thank You
For Your Attention

Transport Layer - Roadmap

3.1 Transport-layer Services

3.2 Multiplexing and Demultiplexing

3.3 Connectionless Transport: UDP

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)
- **no connection state** at sender, receiver (buffer, seq, ack, c-c parameters)
- **small header size** (8 vs 20 bytes)
- **no congestion control**
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

UDP: User Datagram Protocol [RFC 768]

- 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

Popular Internet Applications using TCP/UDP

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Name translation	DNS	Typically UDP

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel
ISI
28 August 1980

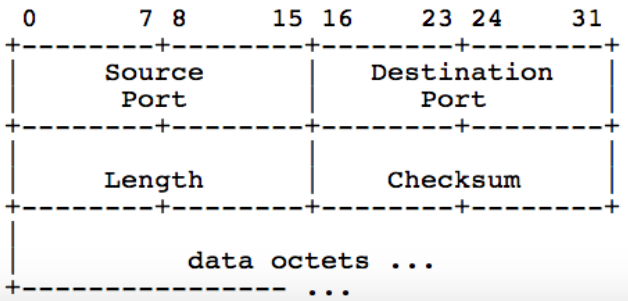
User Datagram Protocol

Introduction

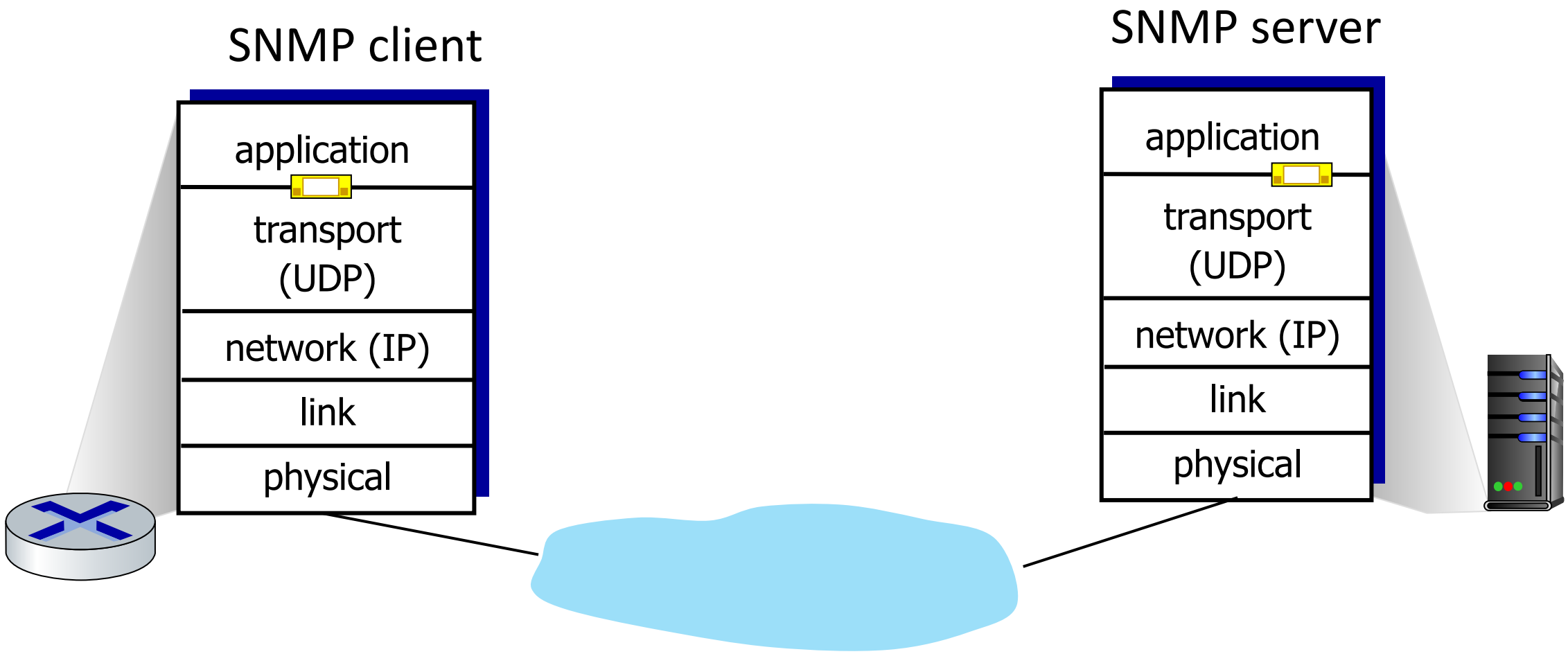
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

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].

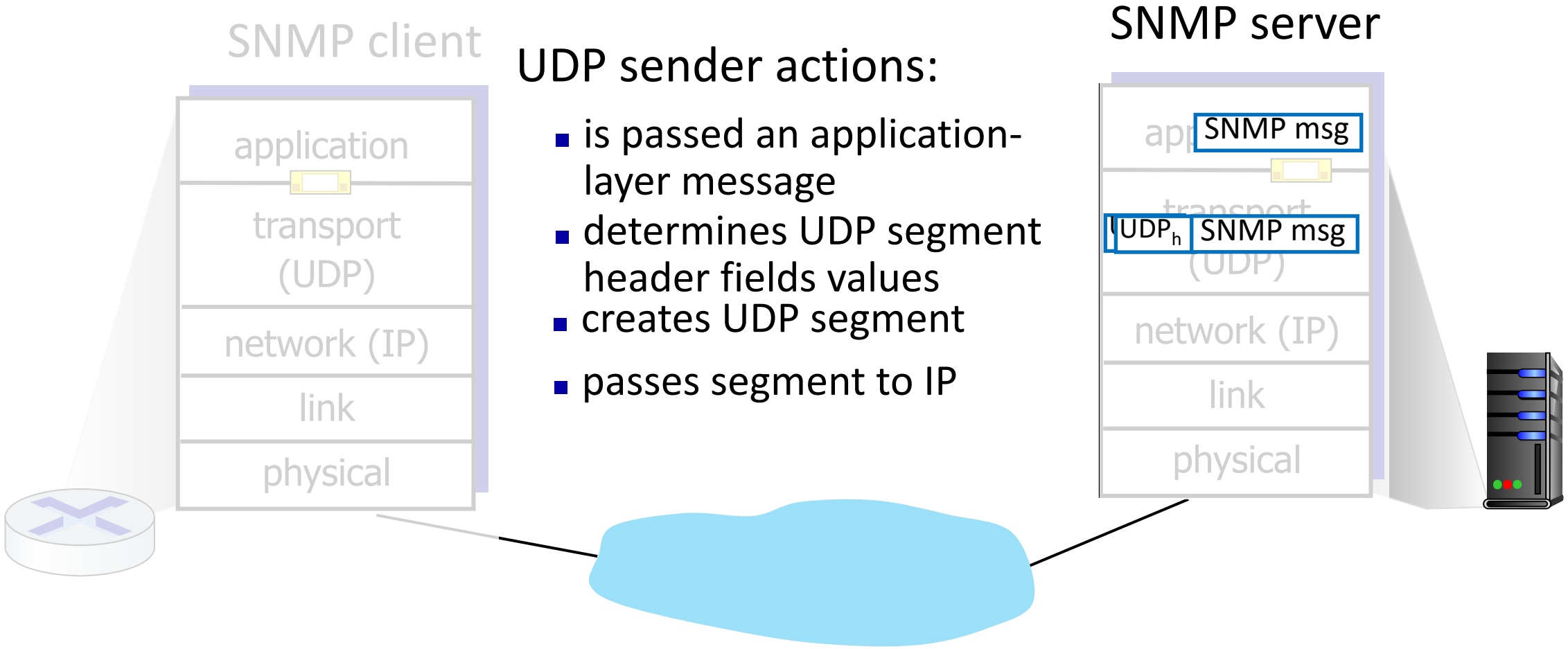
Format



UDP: Transport Layer Actions



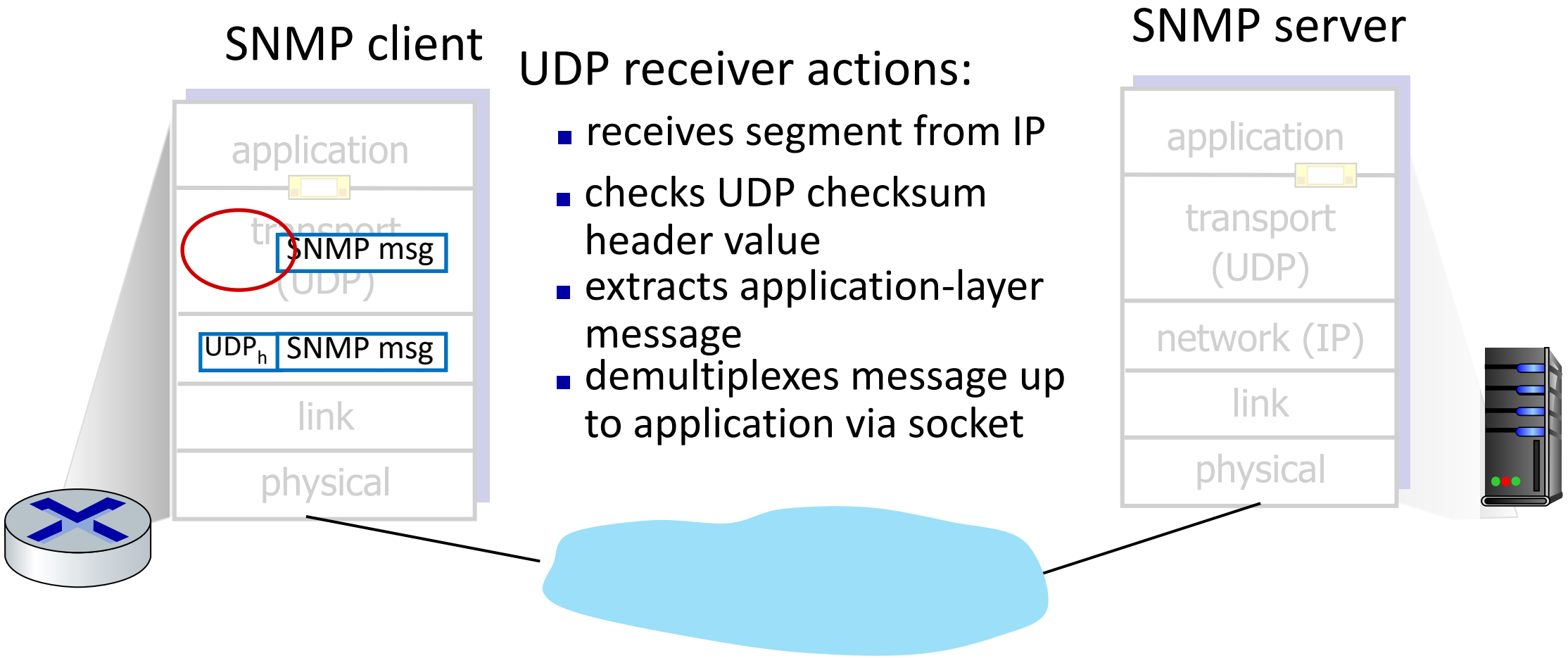
UDP: Transport Layer Actions



UDP sender actions:

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

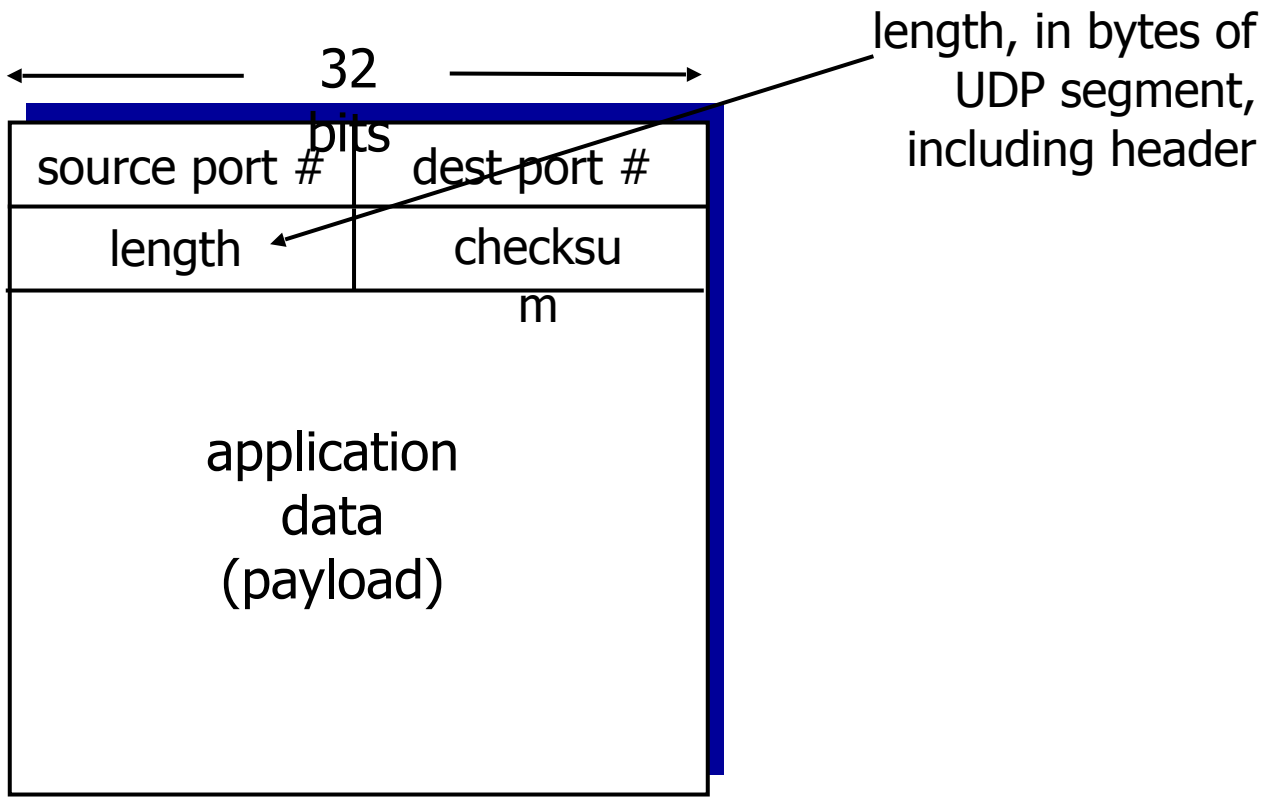
UDP: Transport Layer Actions



UDP receiver actions:

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

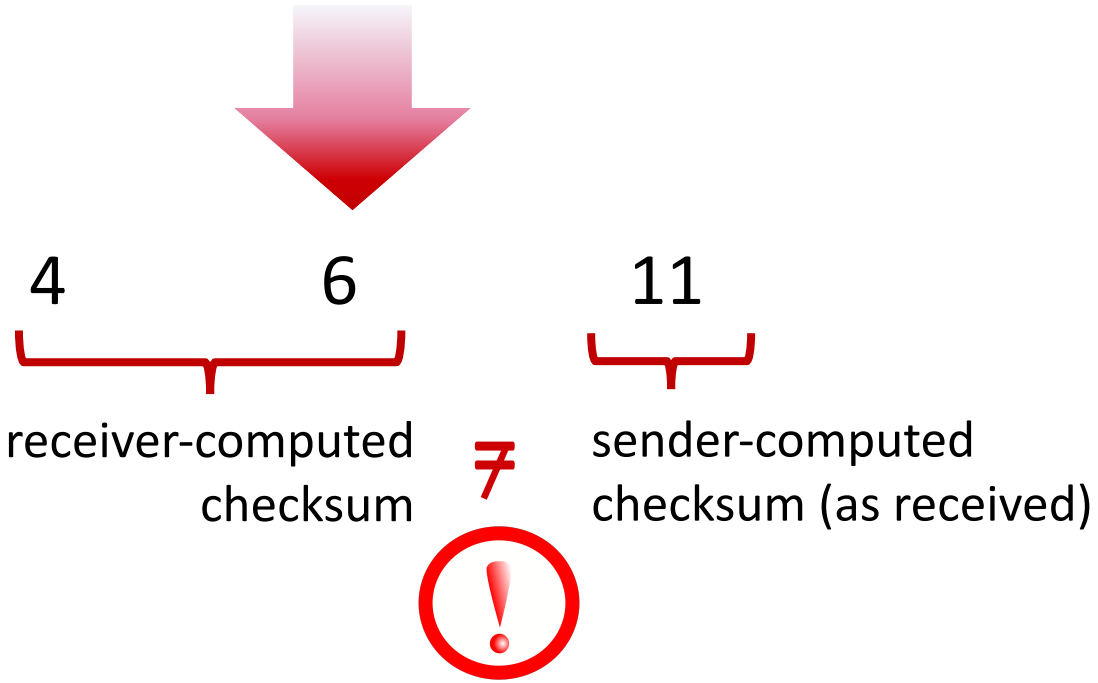
UDP: segment header



UDP segment format

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

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



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: example

example: add two 16-bit integers

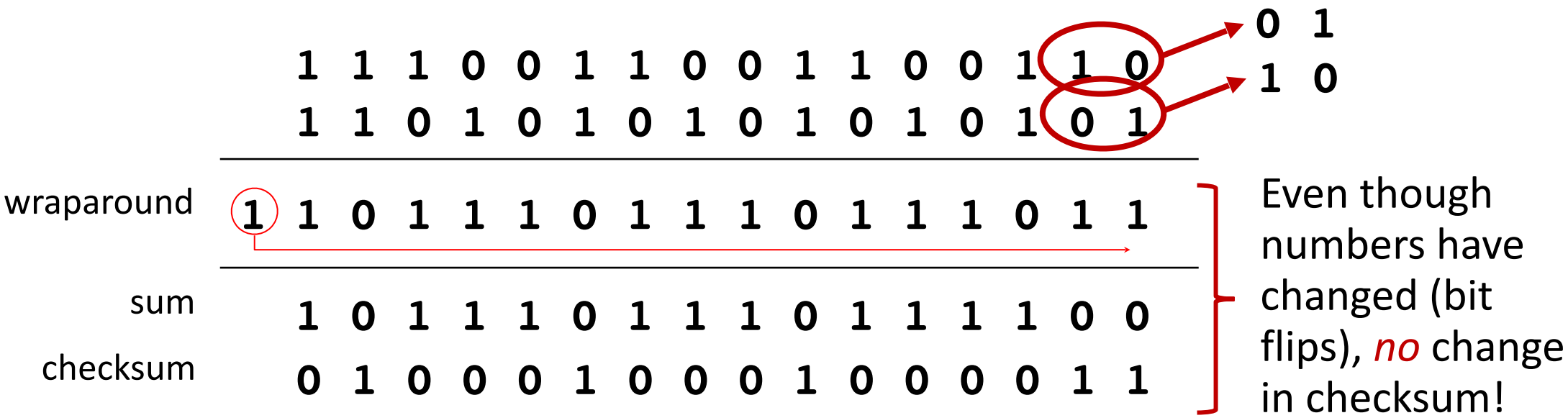
	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

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

* 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



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

Internet Checksum: example

example: add three 16-bit integers

	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	1	0	0	0	1	1	1	1	0	0	0	0	1	1	0	0
sum	0	1	0	0	1	0	1	0	1	1	0	0	0	0	1	0
checksum	1	0	1	1	0	1	0	1	0	0	1	1	1	1	0	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/

Suggested Readings

- UDP – RFC 768 https://tools.ietf.org/html/rfc768
- Networking - DNS and UDP https://youtu.be/vuyQ1PW6AwY

Thank You
For Your Attention

THANK YOU

Department of Computer Science and Engineering