COMPUTER NETWORKS

TEAM NETWORKS

Department of Computer Science and Engineering

PES UNIVERSITY
CELEBRATING 50 YEARS

COMPUTER NETWORKS

Transport Layer

TEAM NETWORKS

Department of Computer Science and Engineering

Transport Layer - Roadmap

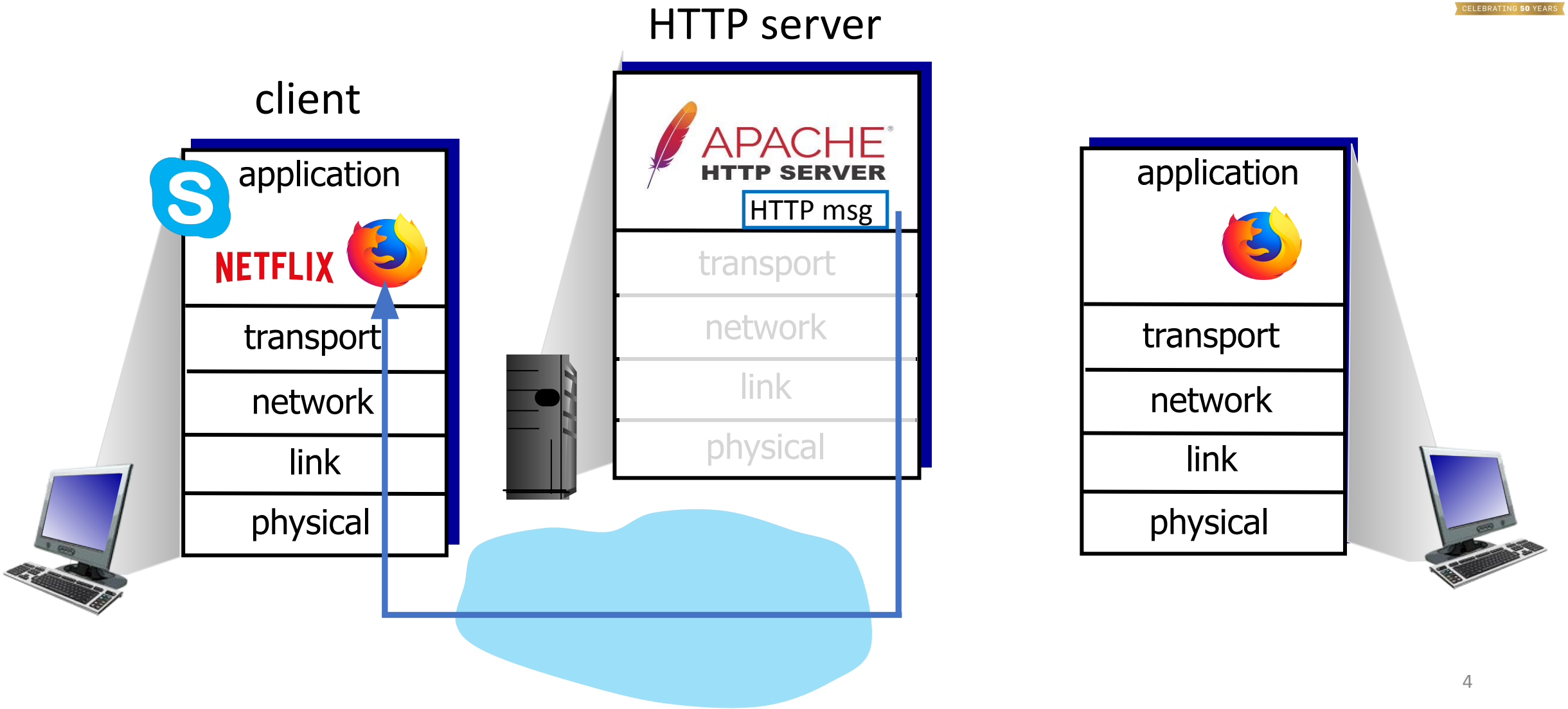
3.1 Transport-layer Services

3.2 Multiplexing and Demultiplexing

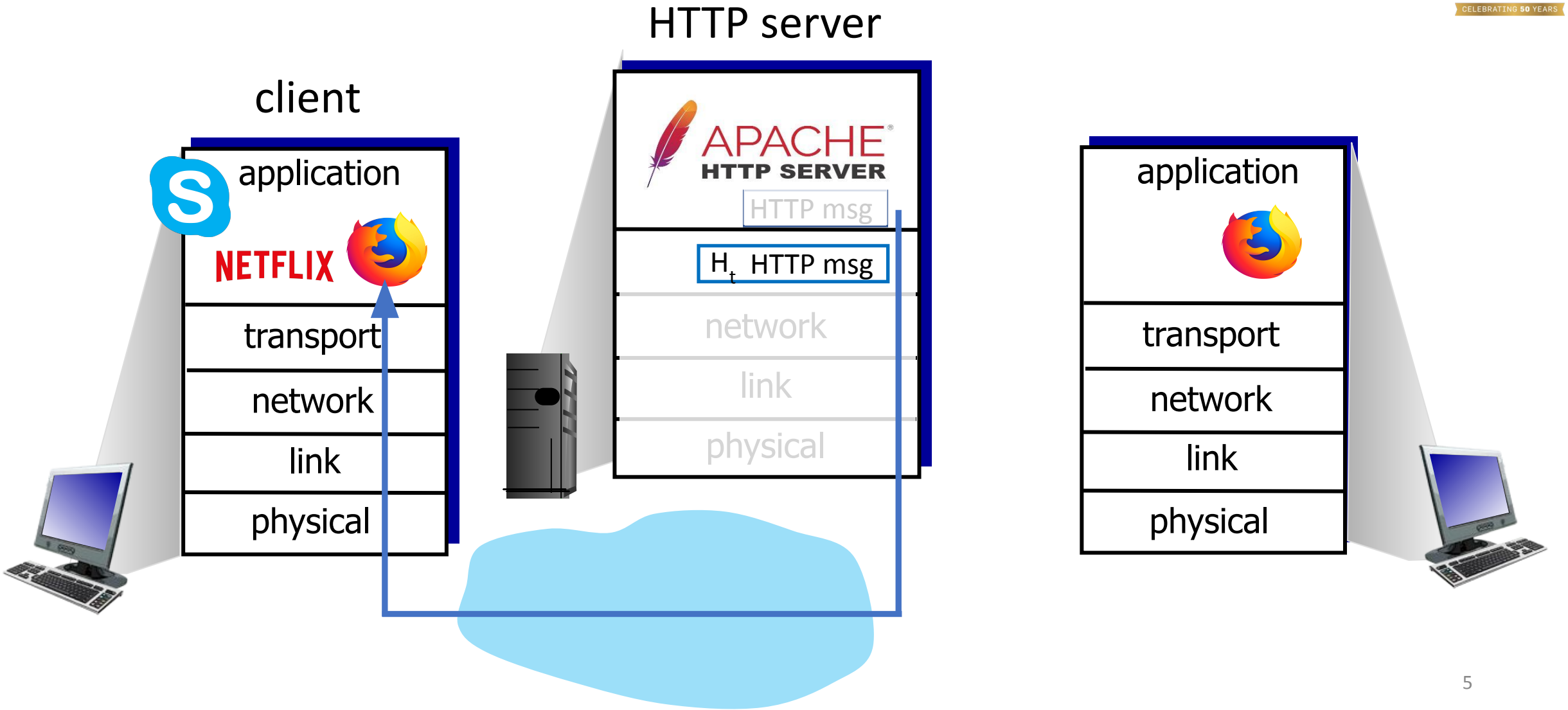
3.3 Connectionless Transport: UDP

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

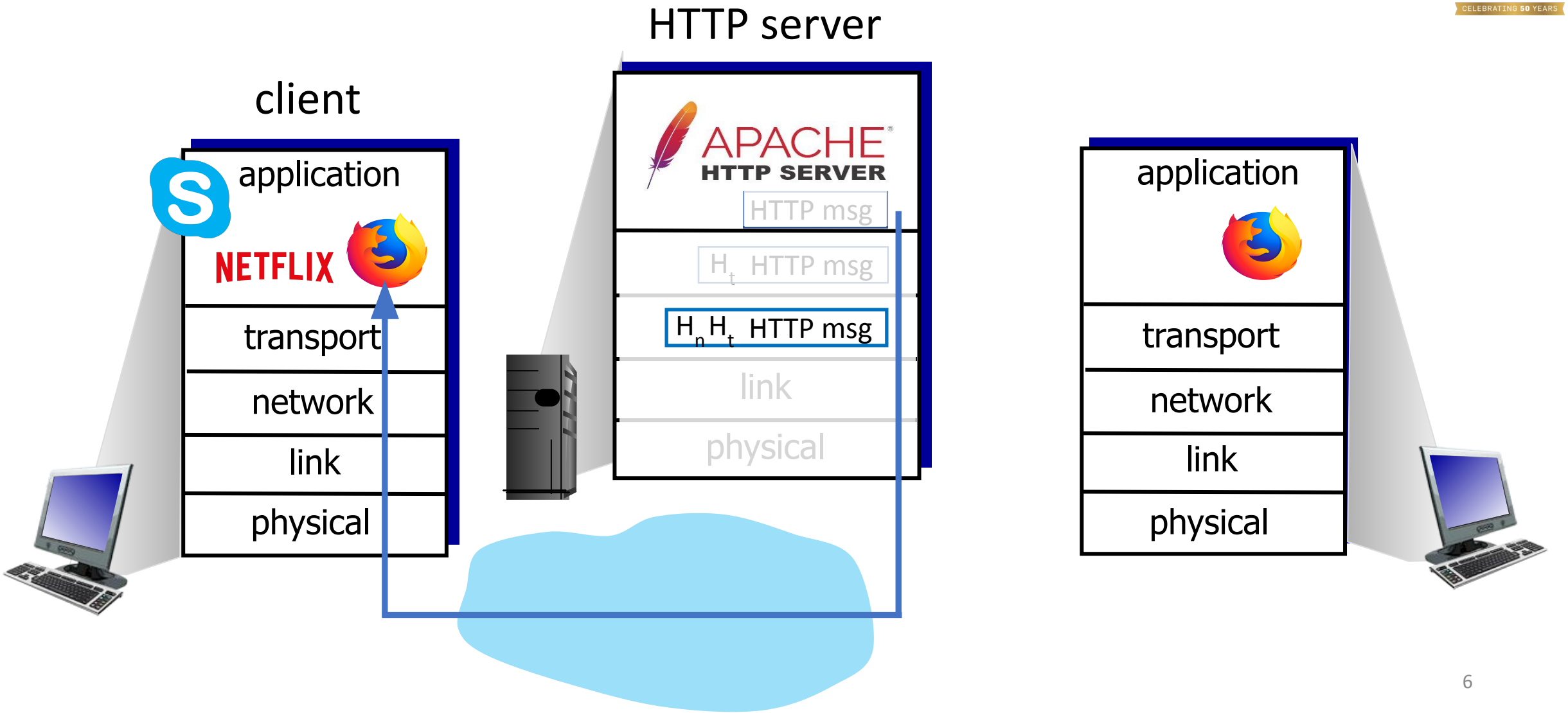
Multiplexing and Demultiplexing



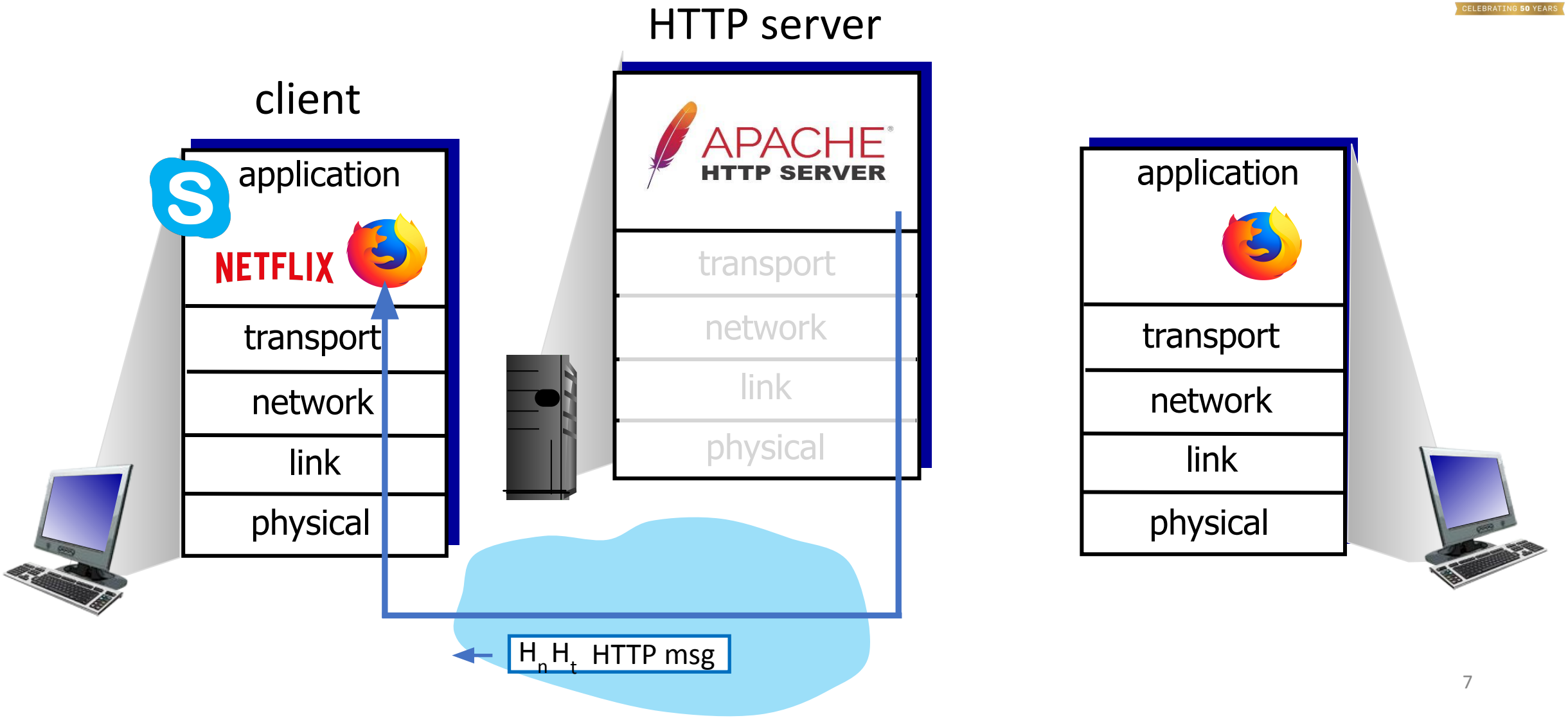
Multiplexing and Demultiplexing



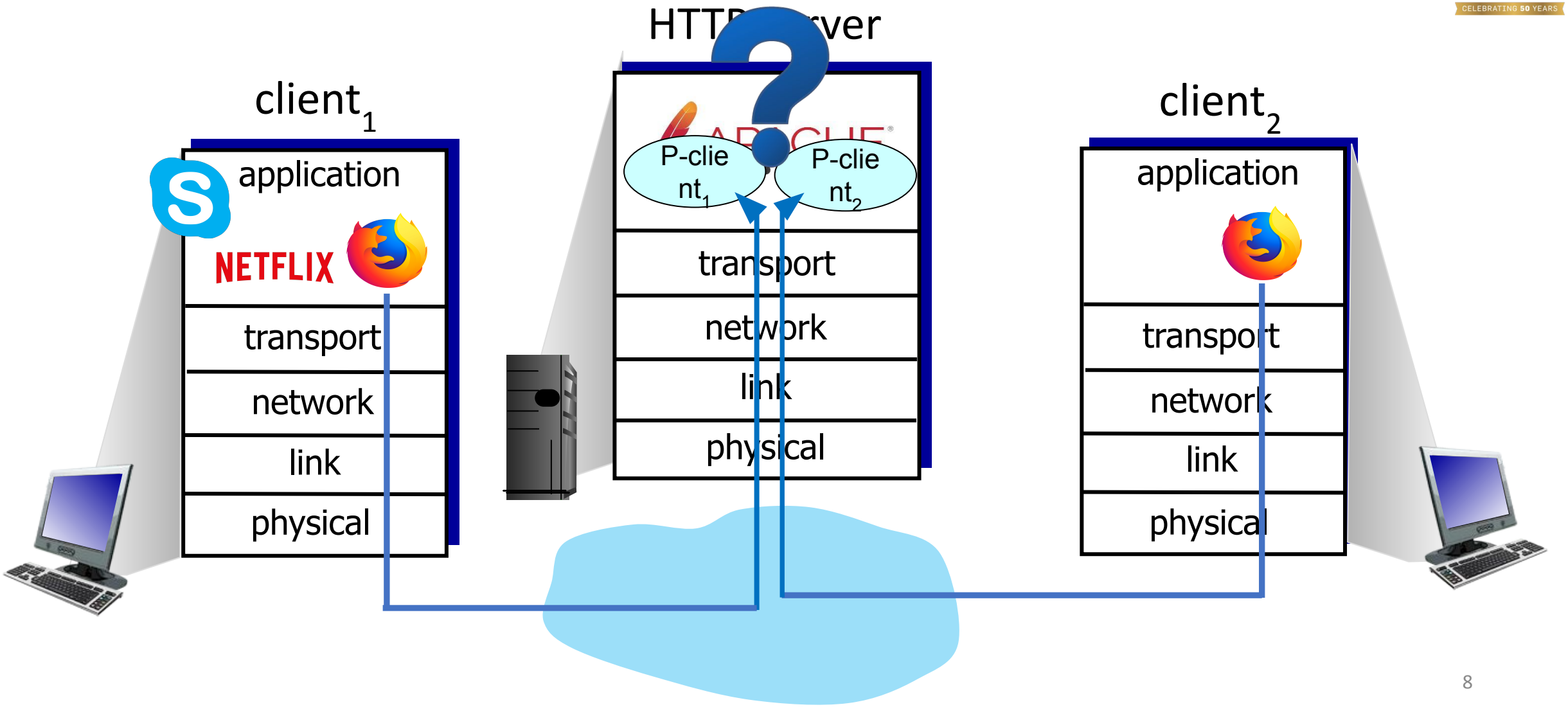
Multiplexing and Demultiplexing



Multiplexing and Demultiplexing



Multiplexing and Demultiplexing



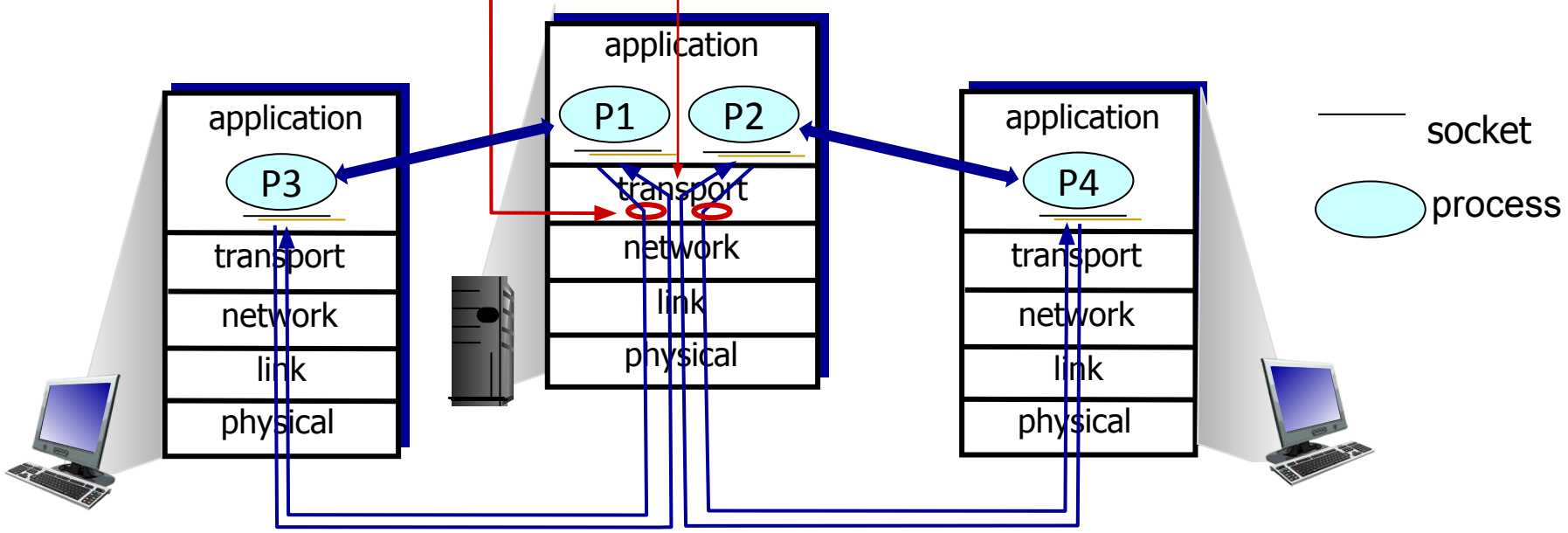
Multiplexing and Demultiplexing

multiplexing at sender:

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

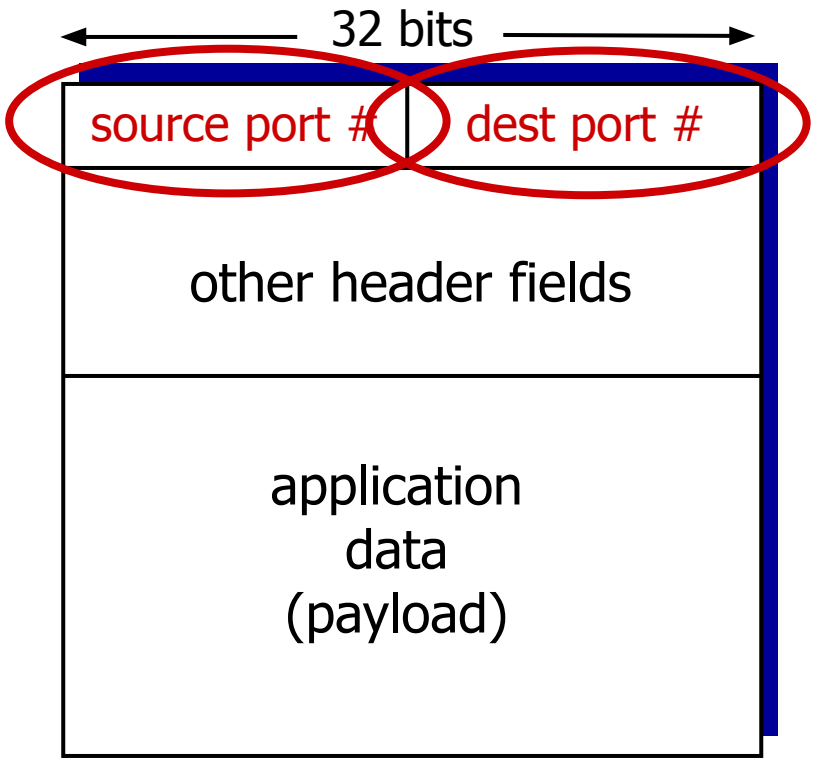
demultiplexing at receiver:

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

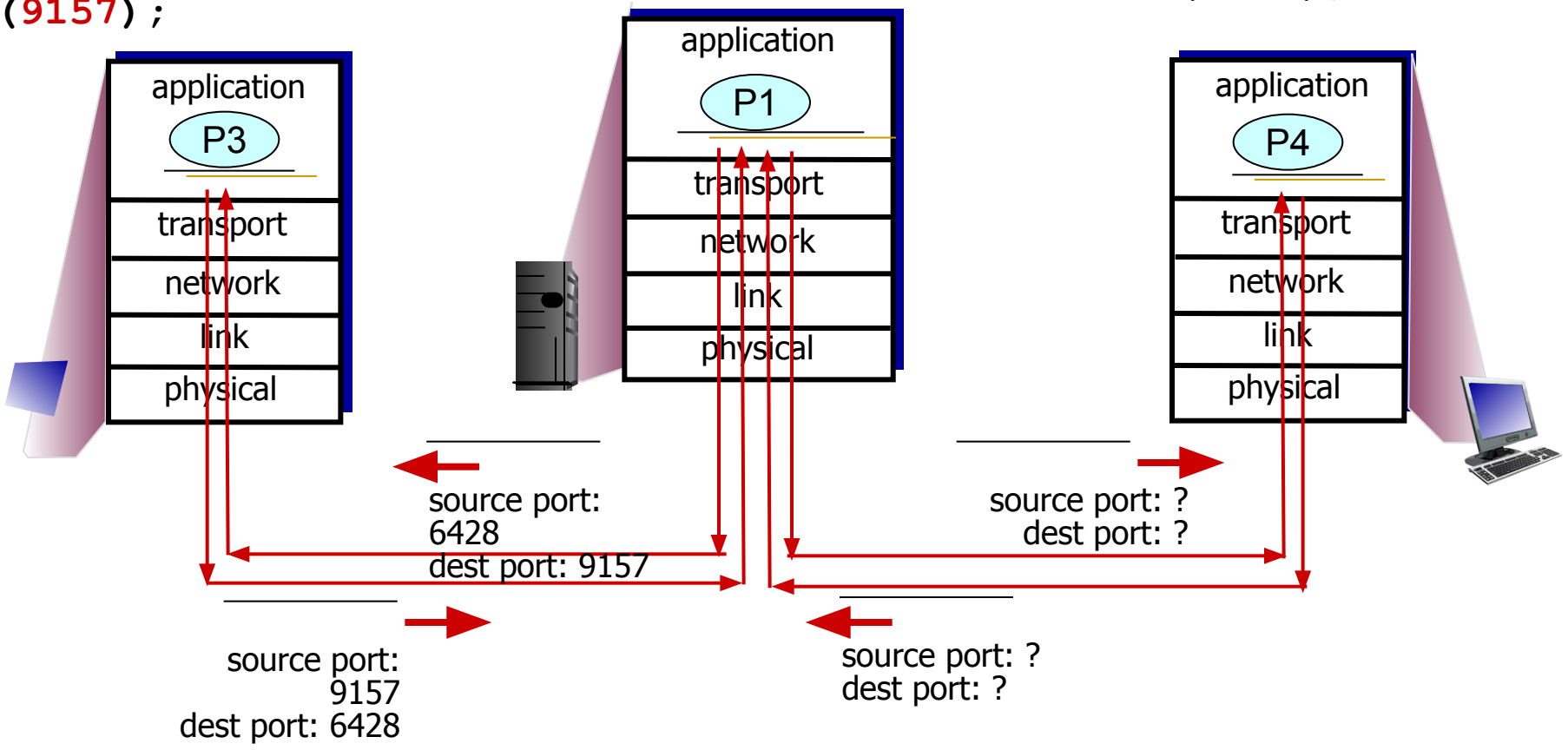
COMPUTER NETWORKS

Connectionless demultiplexing: example

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

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

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

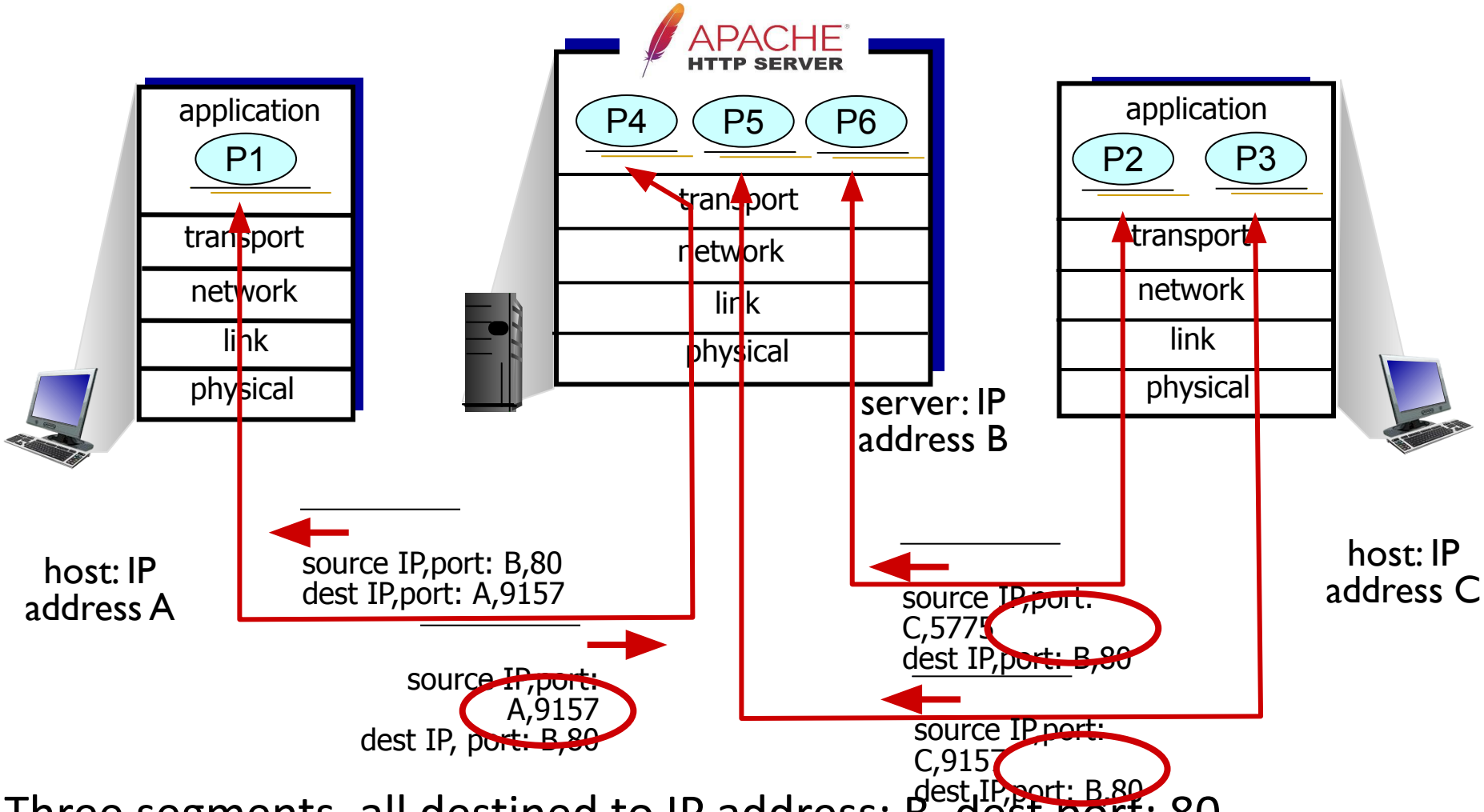


Connection-oriented demultiplexing

- 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

THANK YOU

Department of Computer Science and Engineering