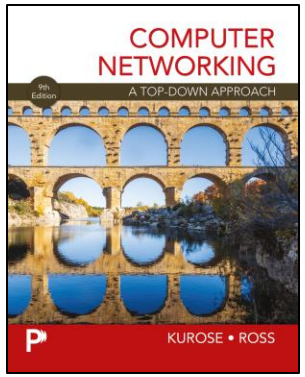


CSC 361
Computer Networks
Transport Layer
(Chapter 3 – Intro)

Wenjun Yang

Summer 2026

Transport Layer: Overview

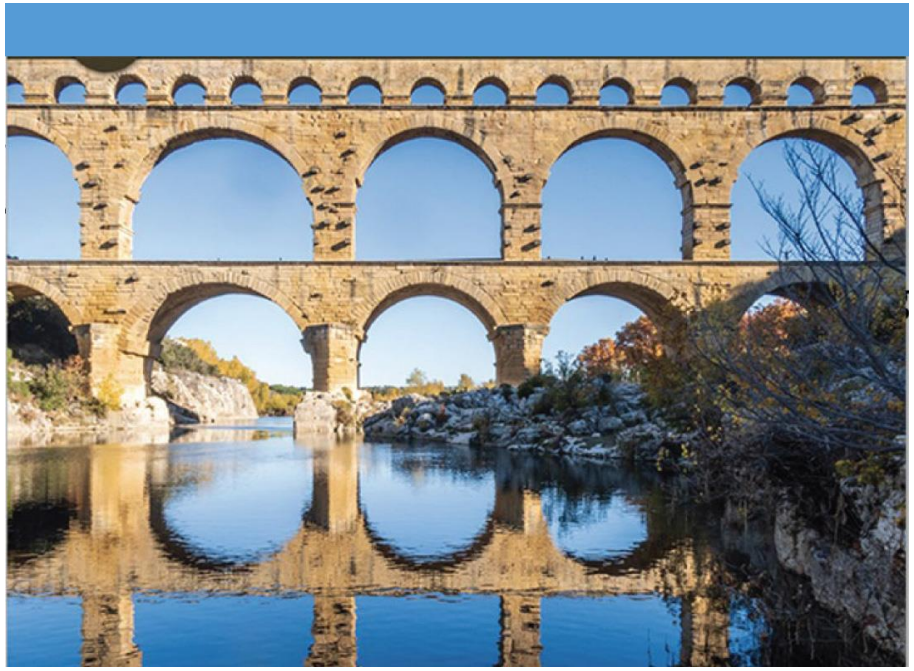
Our goal:

- 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

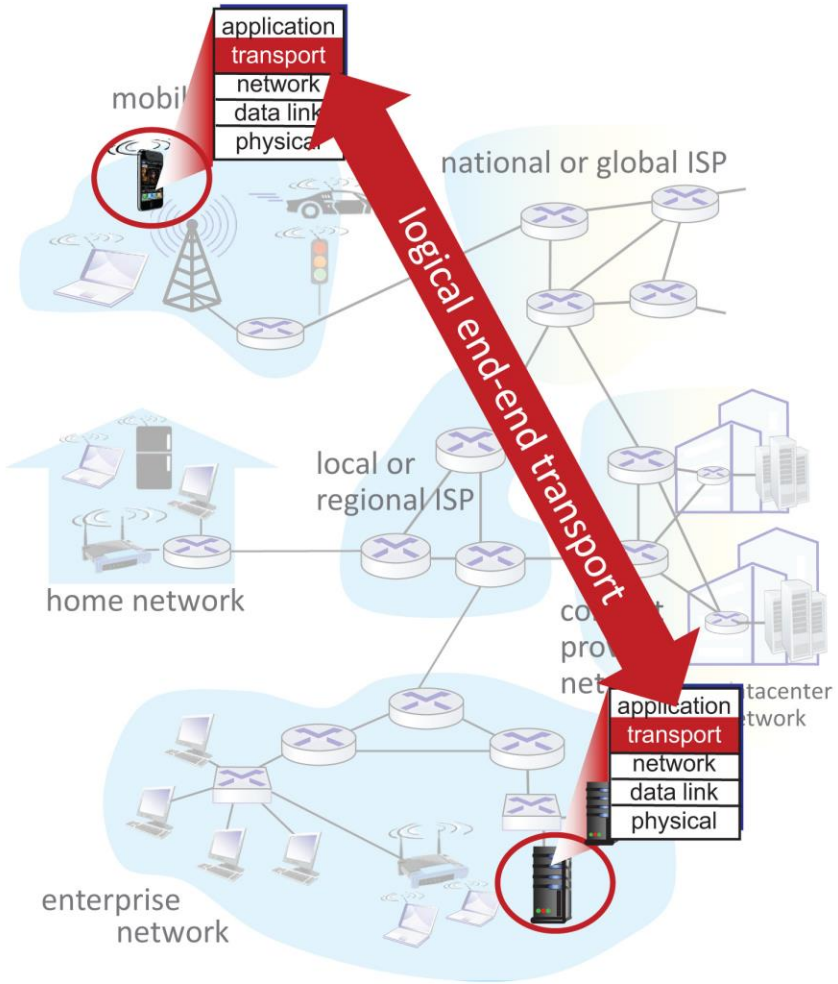
Transport Layer: Roadmap (1 of 2)

- 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

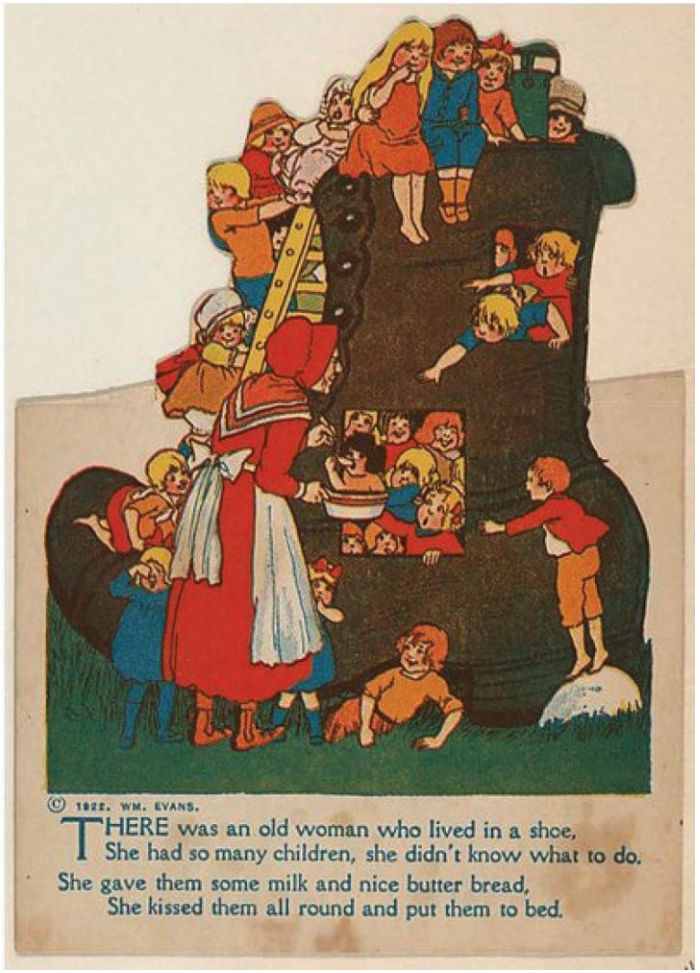


Transport Services and Protocols

- 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 (1 of 2)



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 (2 of 2)

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

household analogy:

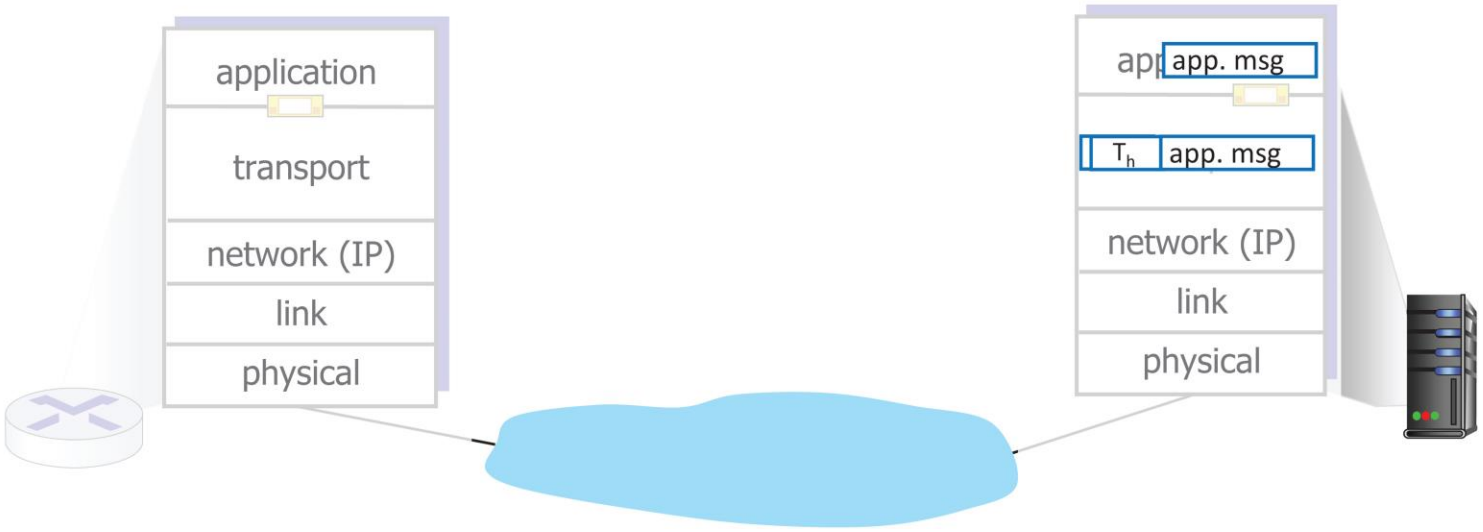
12 kids in Ann's house sending letters to 12 kids in Bill's house:

- 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 (1 of 2)

Sender:

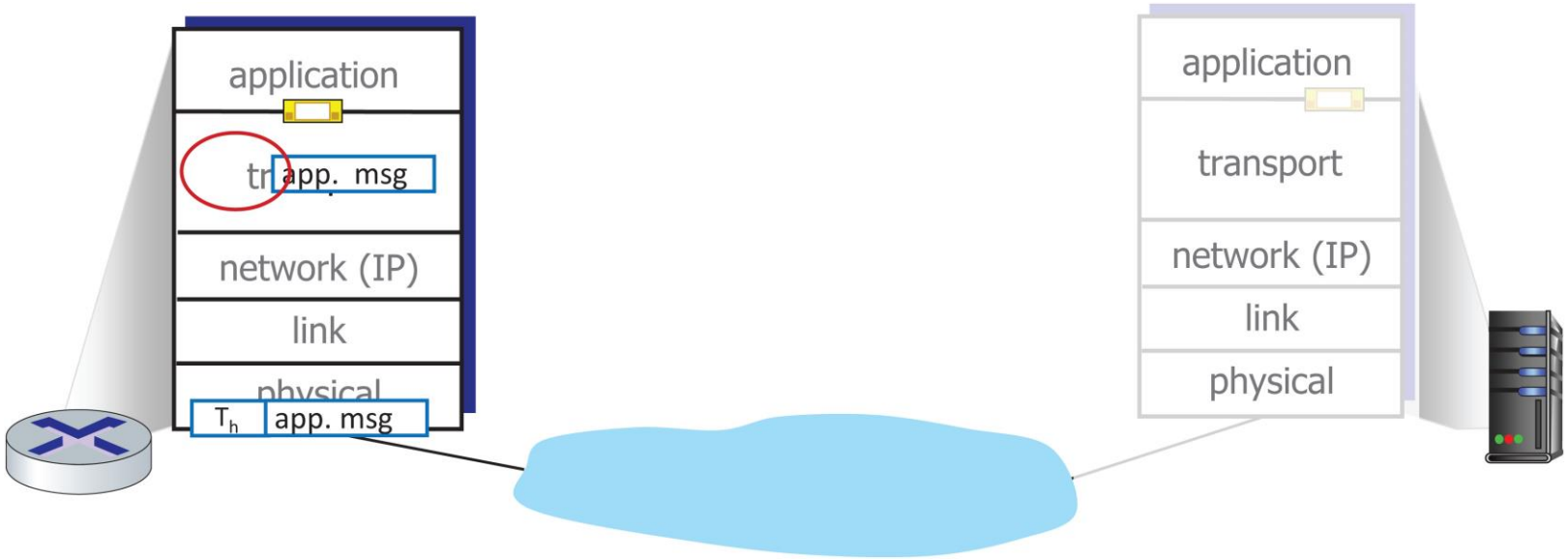
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



Transport Layer Actions (2 of 2)

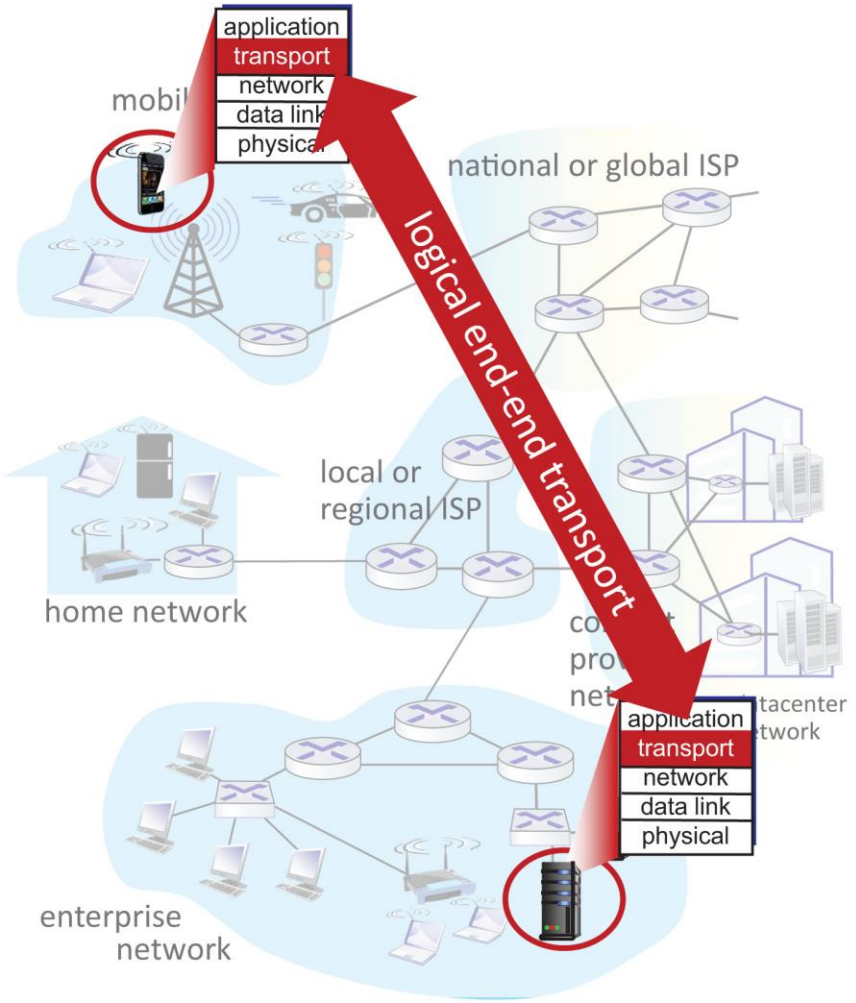
Receiver:

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



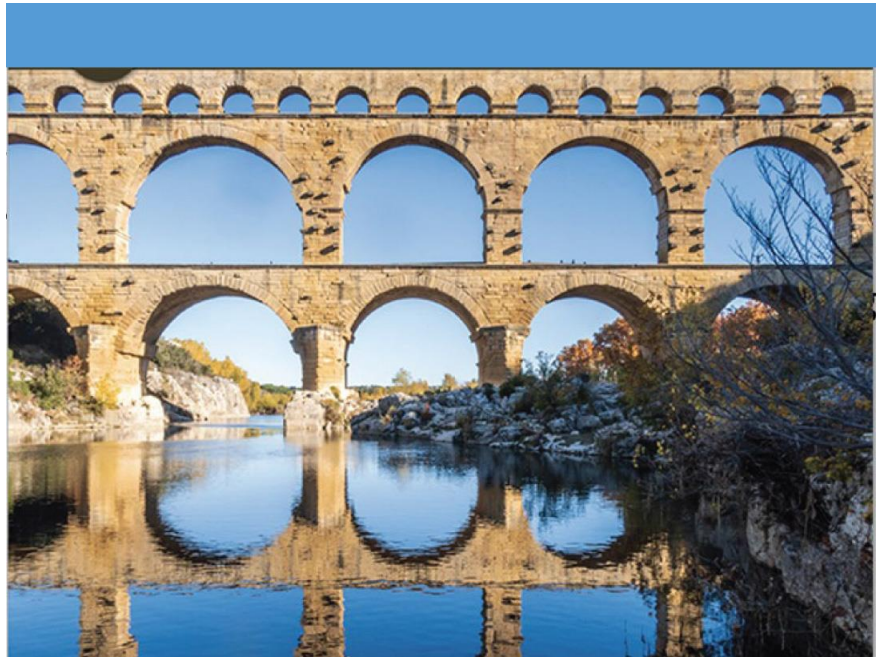
Two Principal Internet Transport Protocols

- **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 (1 of 7)

- 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



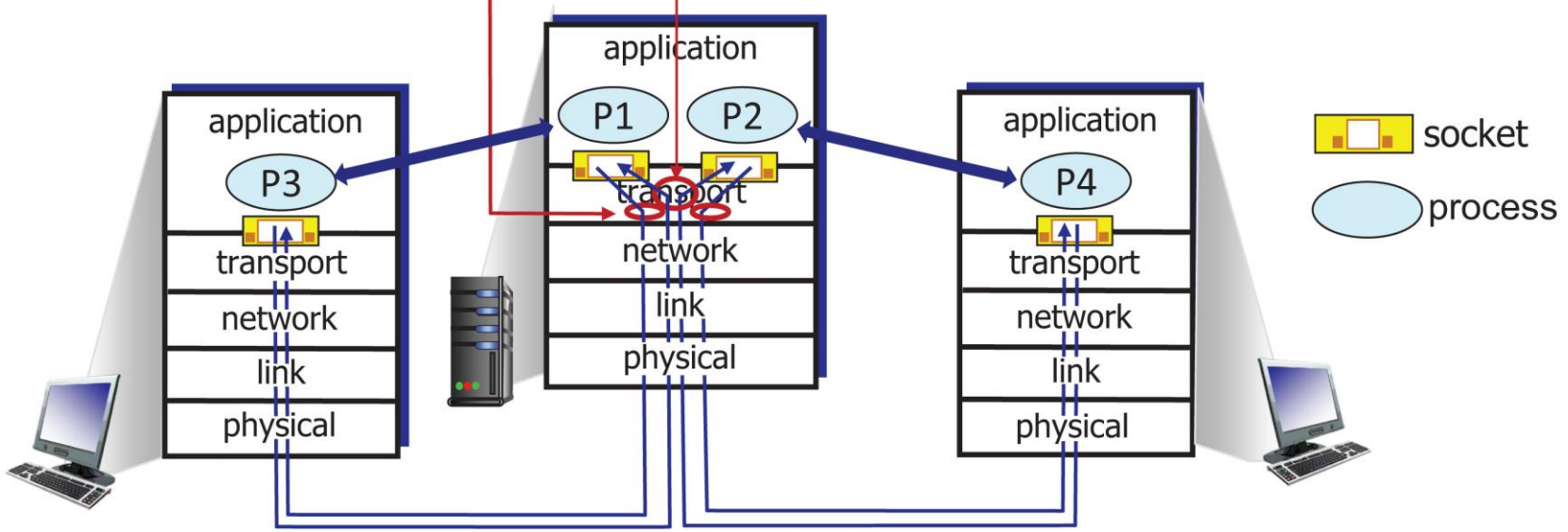
Multiplexing/Demultiplexing (1 of 7)

multiplexing as sender:

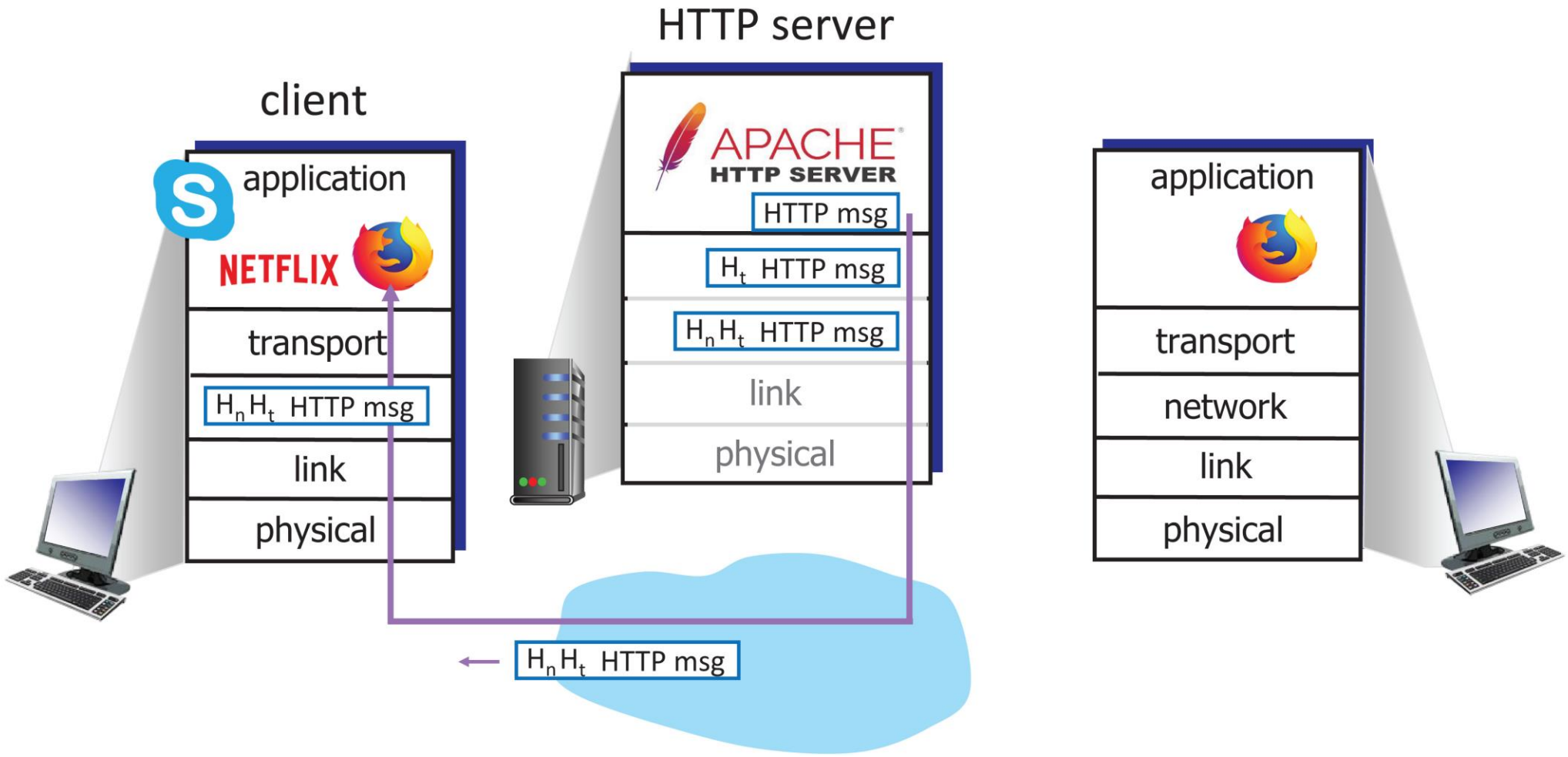
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing as receiver:

use header info to deliver received segments to correct socket

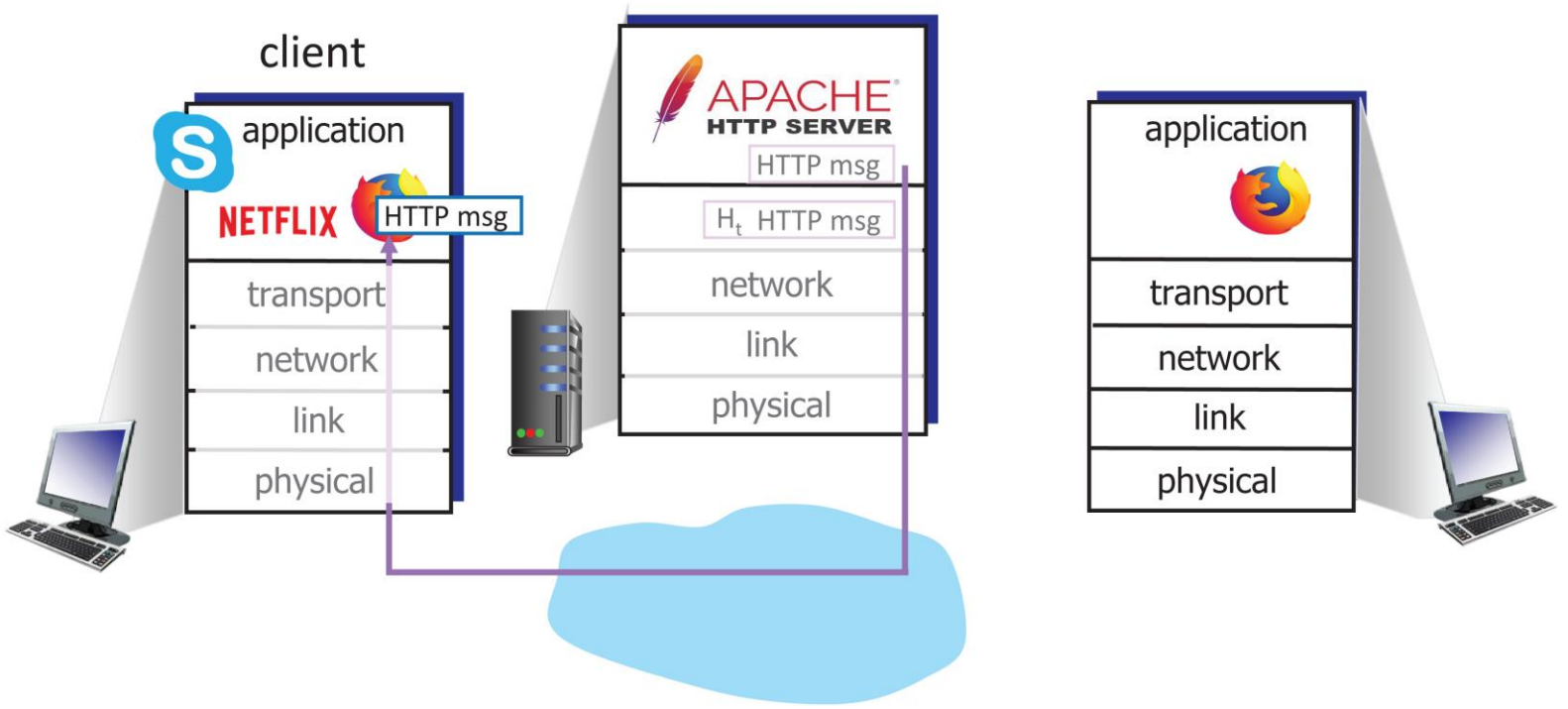


Multiplexing/Demultiplexing (2 of 7)

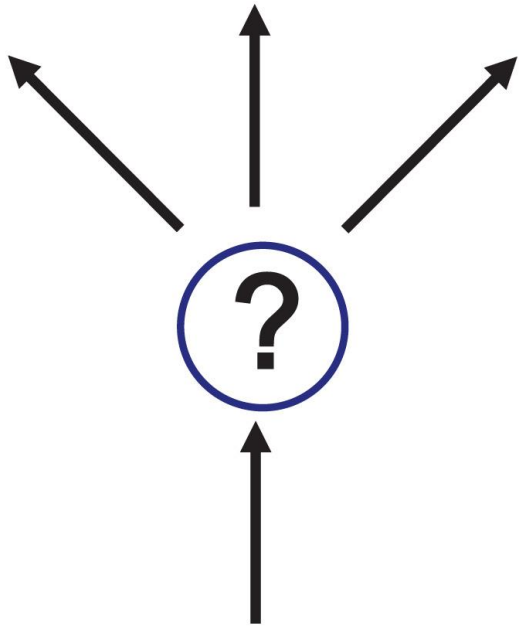


Multiplexing/Demultiplexing (3 of 7)

Q: how did transport layer know to deliver message to Firefox browser process rather then Netflix process or Skype process?

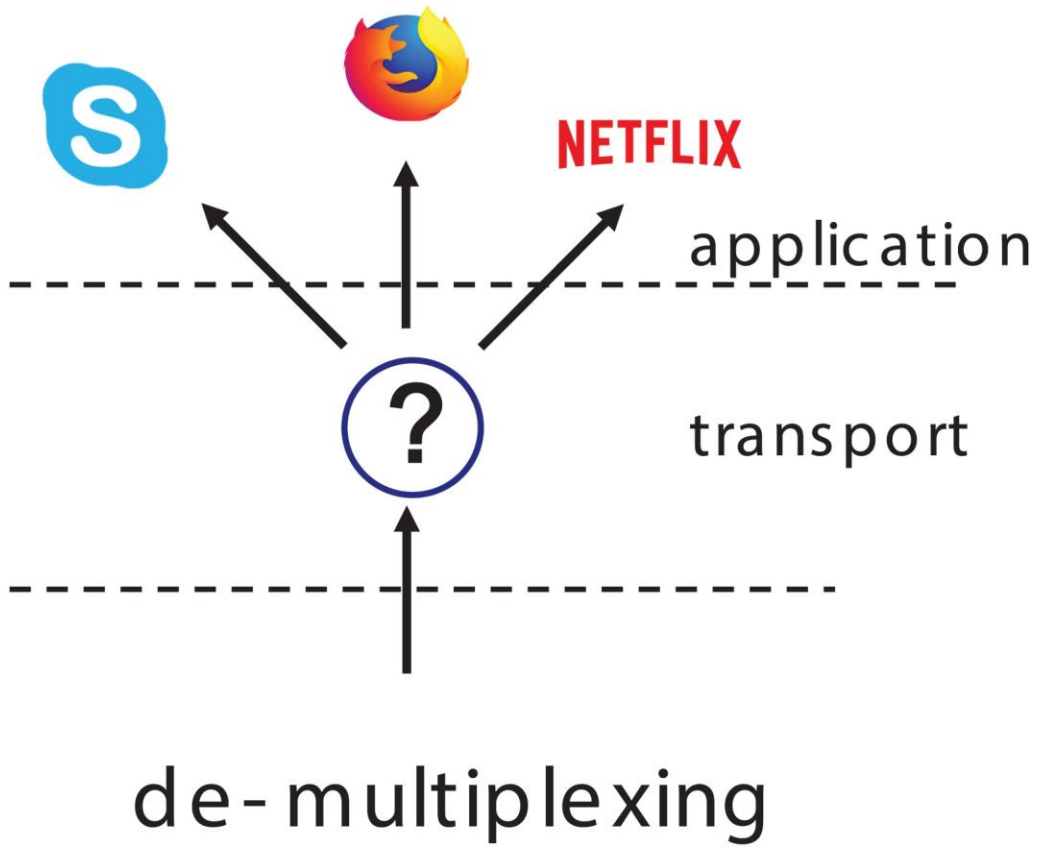


Multiplexing/Demultiplexing (4 of 7)



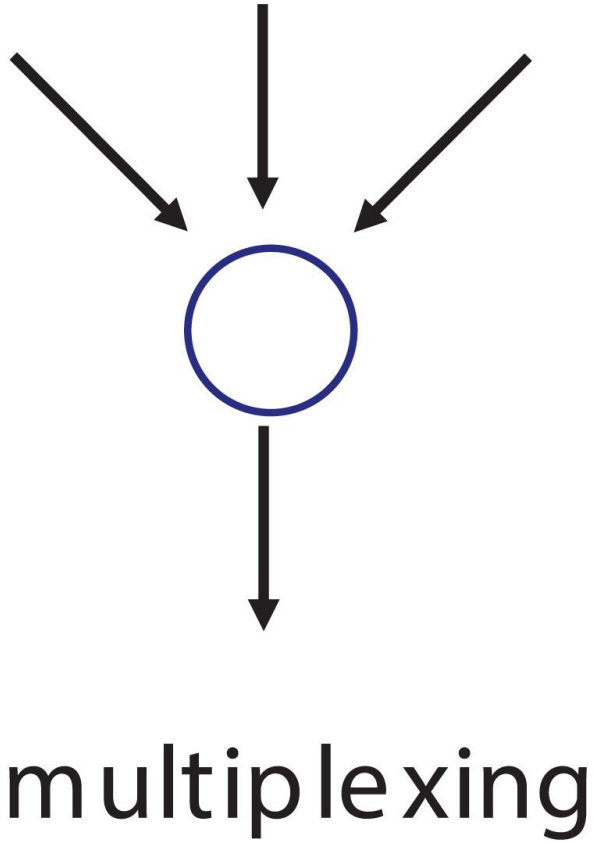
de-multiplexing

Multiplexing/Demultiplexing (5 of 7)

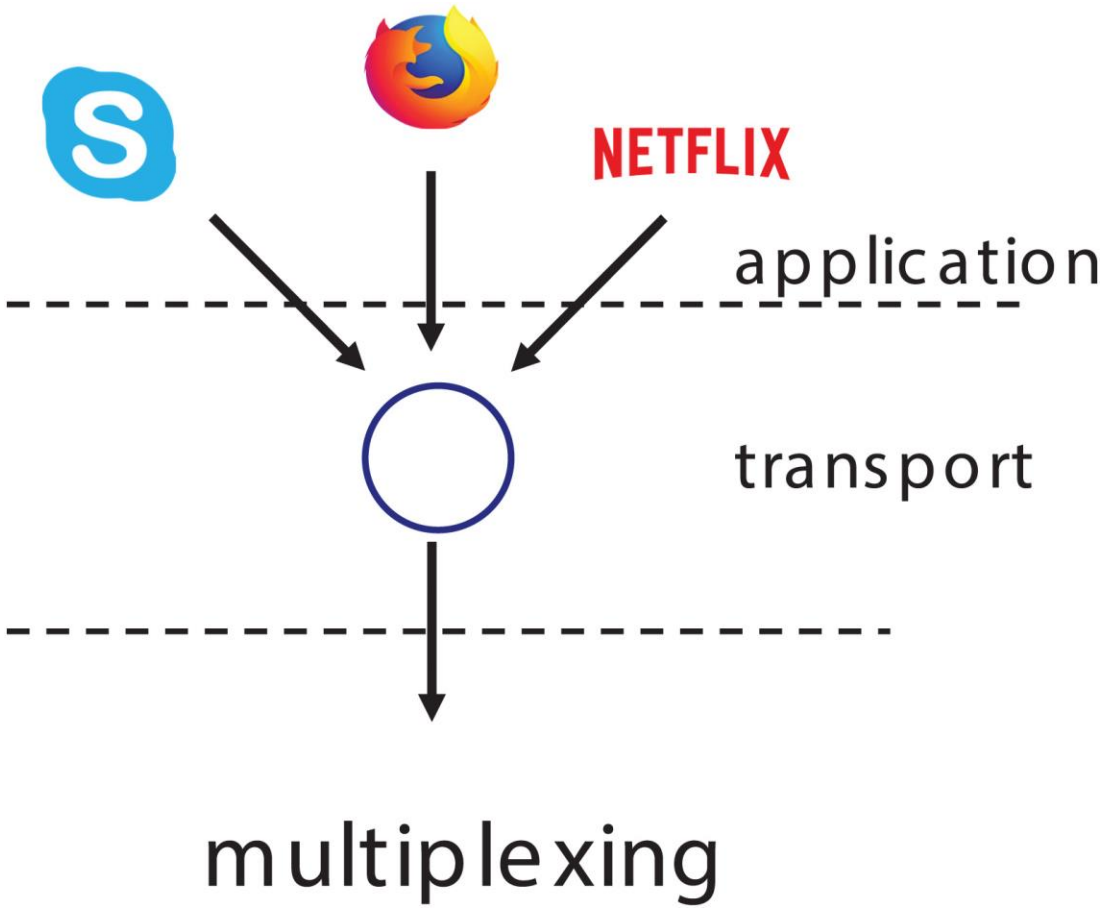



NORTH TO
95
NORTH
EXIT 127
NORTH TO
9
440
287
Woodbridge
Staten Island
Demultiplexing

Multiplexing/Demultiplexing (6 of 7)



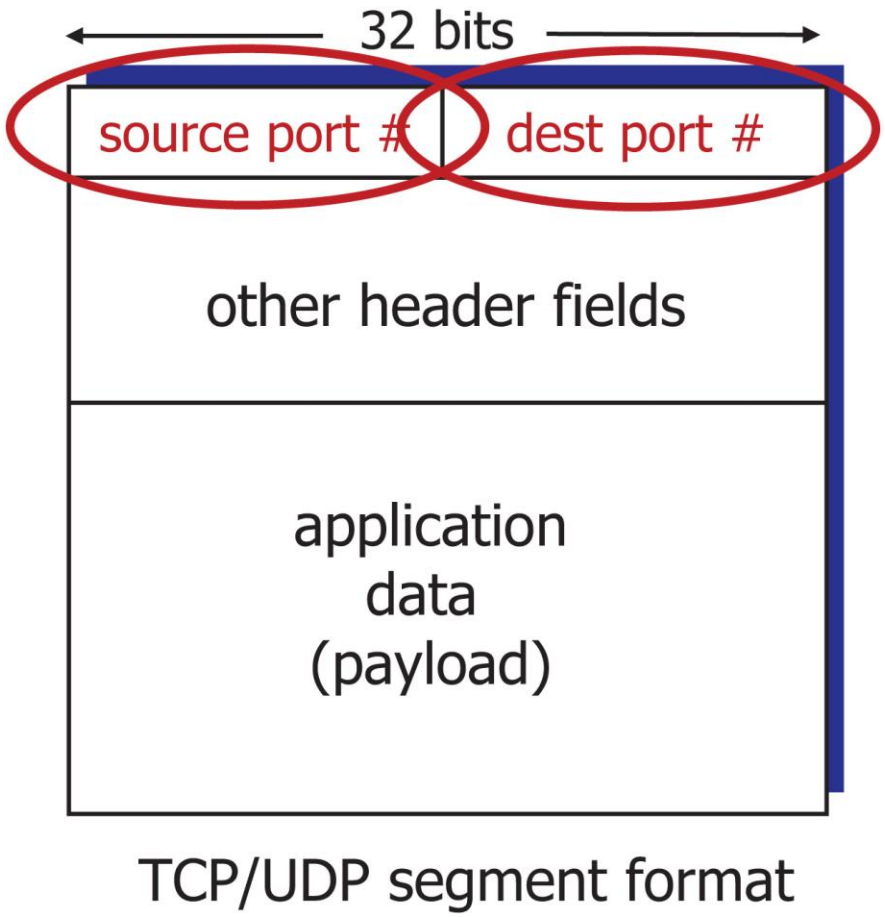
Multiplexing/Demultiplexing (7 of 7)




Multiplexing

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 #

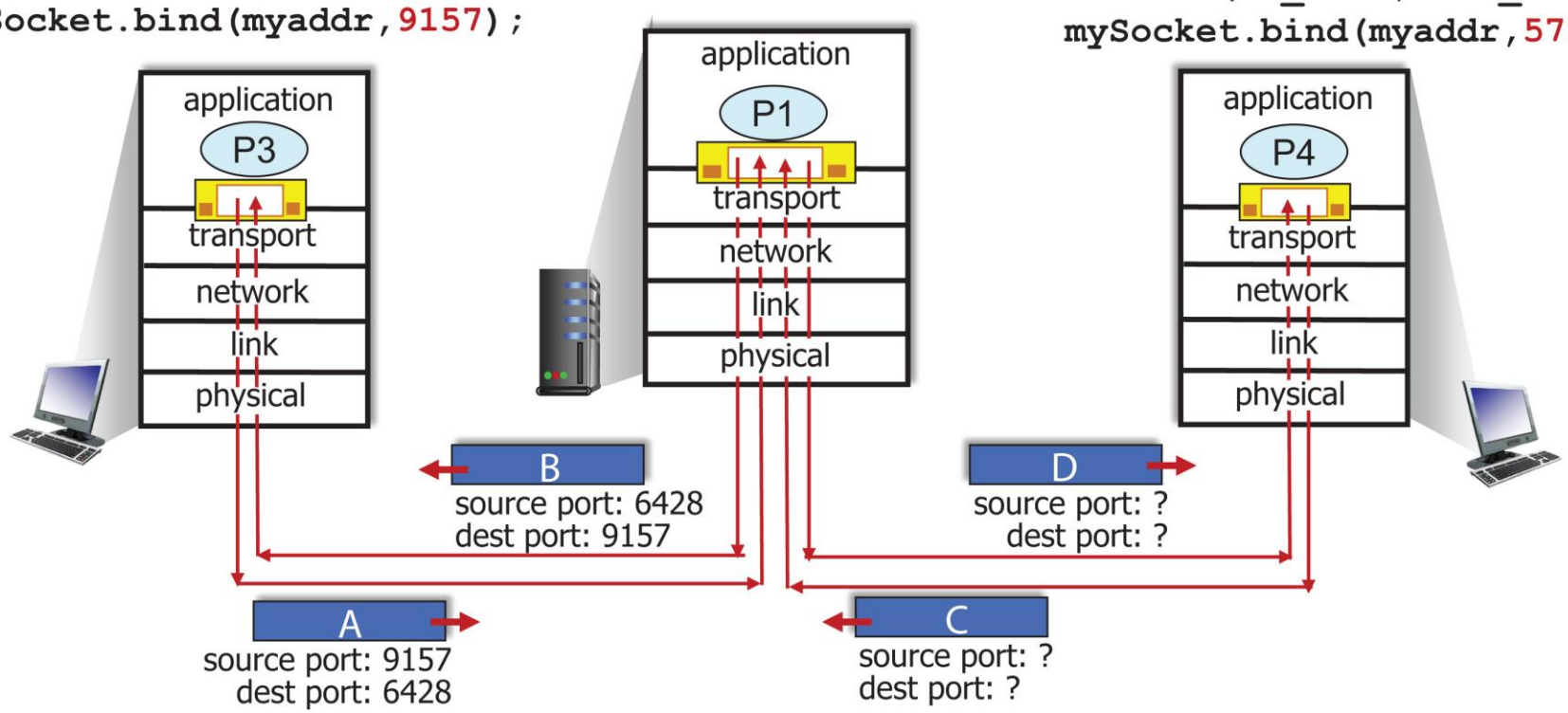
IP/UDP datagrams with **same dest. port #,** but different source IP addresses and/or source port numbers will be directed to **same socket** at receiving host

Connectionless Demultiplexing: an Example

```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,6428);
```

```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,9157);
```

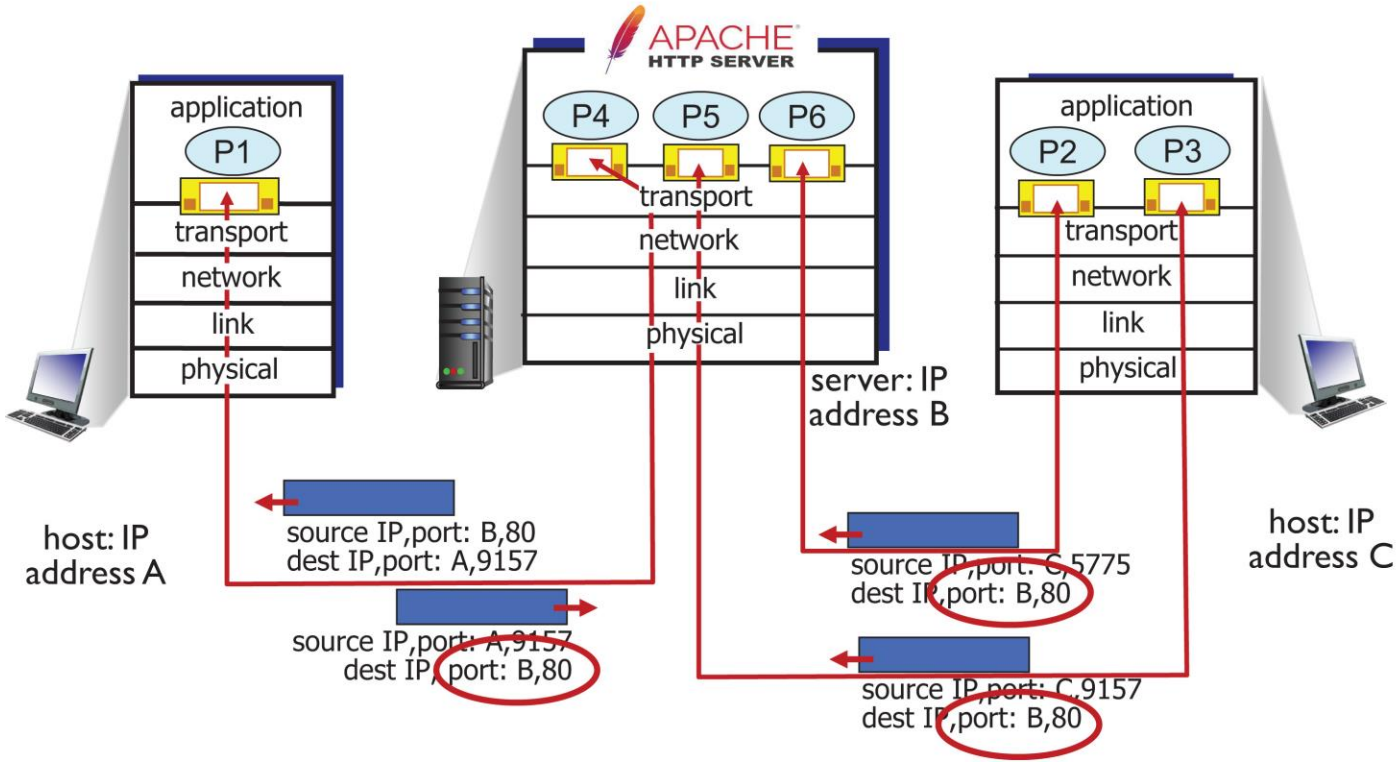
```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,5775);
```



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