CSC2B10/CSC02B2

Data Communications

Chapter 3.1

UNIVERSITY OF JOHANNESBURG

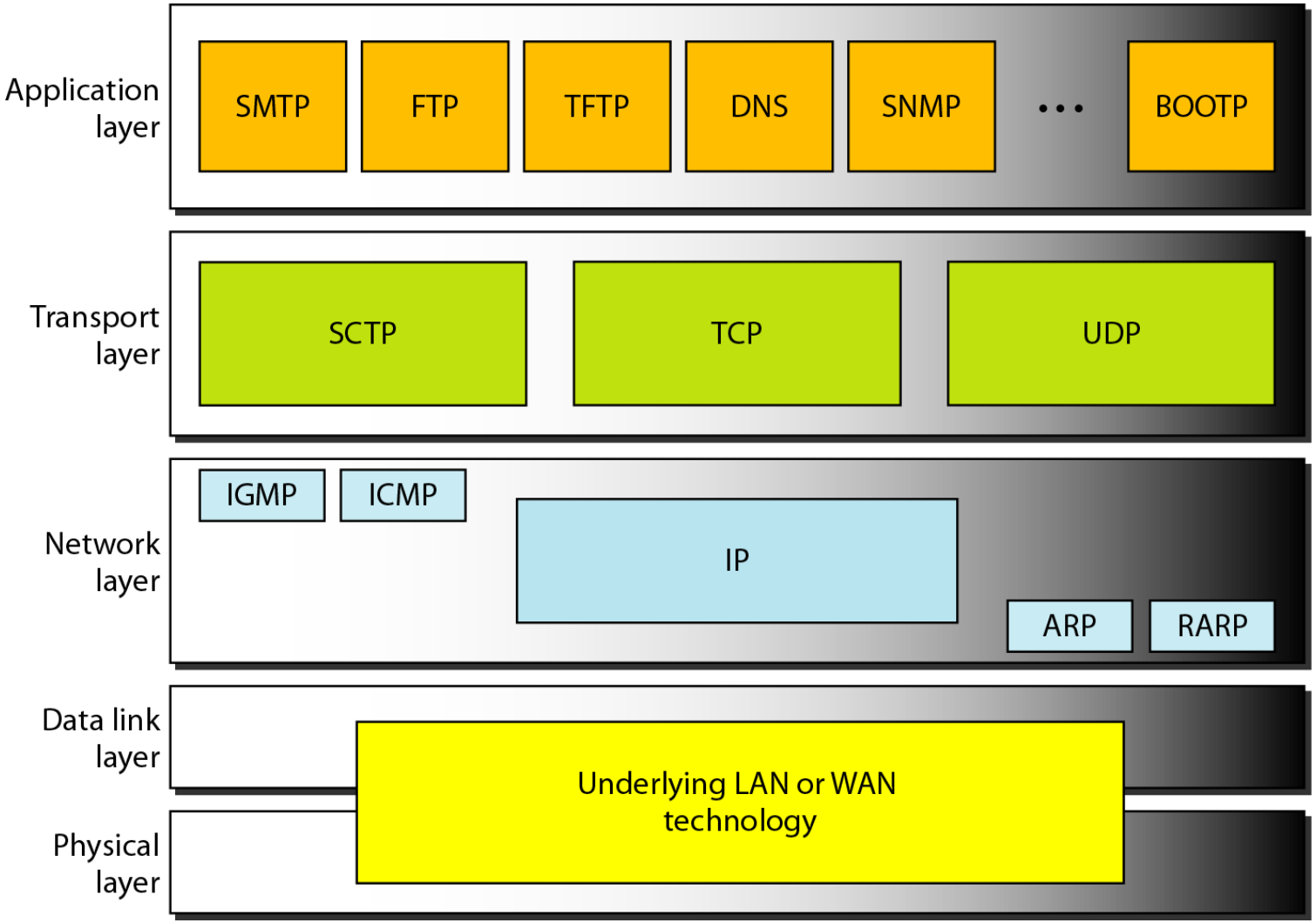
Our goals:

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

Transport layer: 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

Position of UDP, TCP, and SCTP in TCP/IP suite

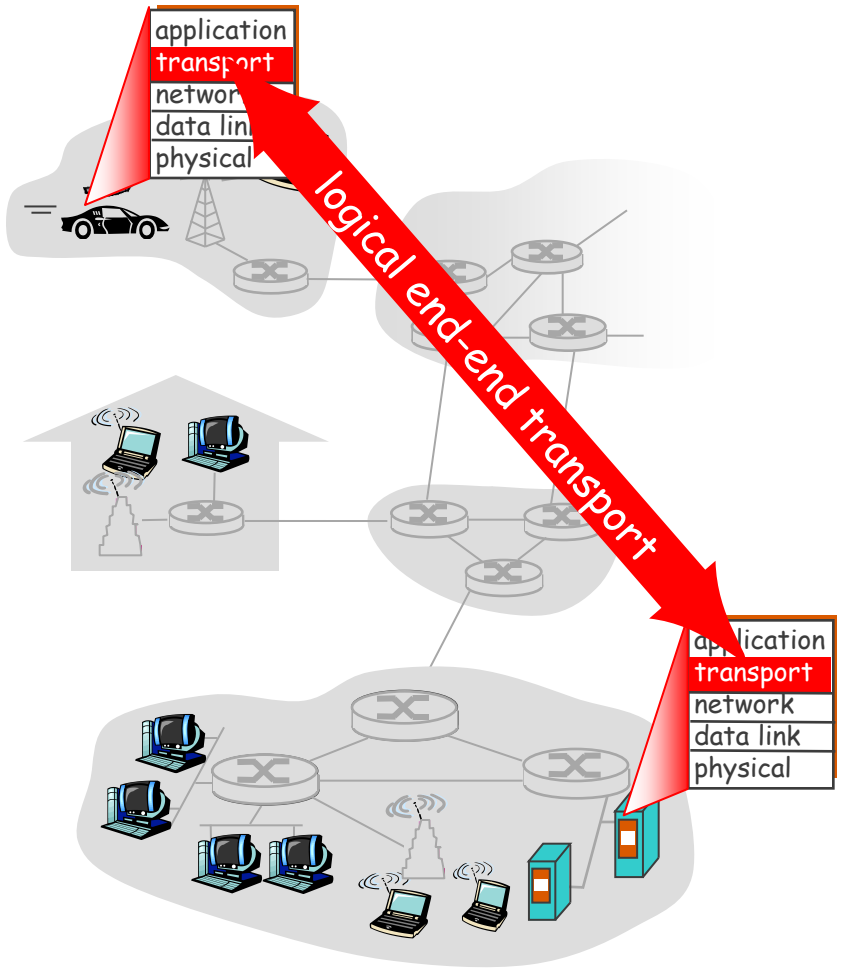


Transport services and protocols

- provide ***logical communication*** between app processes running on different hosts

- transport protocols run in end systems
  - send side: breaks app messages into **segments,** passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer

- more than one transport protocol available to apps
  - Internet: **TCP and UDP**



Transport vs. network layer services and protocols

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

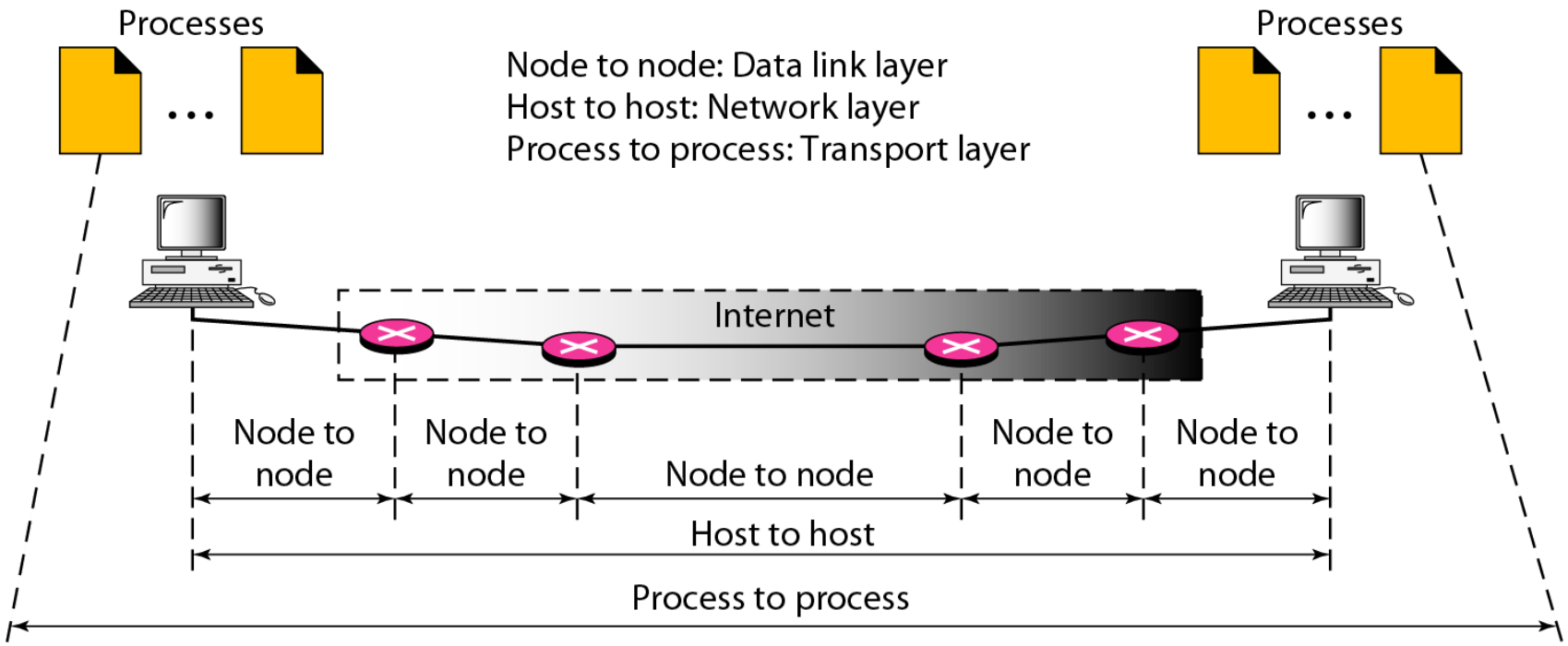
- ***transport layer:*** logical communication between processes
  - relies on, enhances, network layer services

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

Process to process delivery



Transport vs. network layer

Addressing:

Data link – MAC address

Network – IP address

Transport – port number

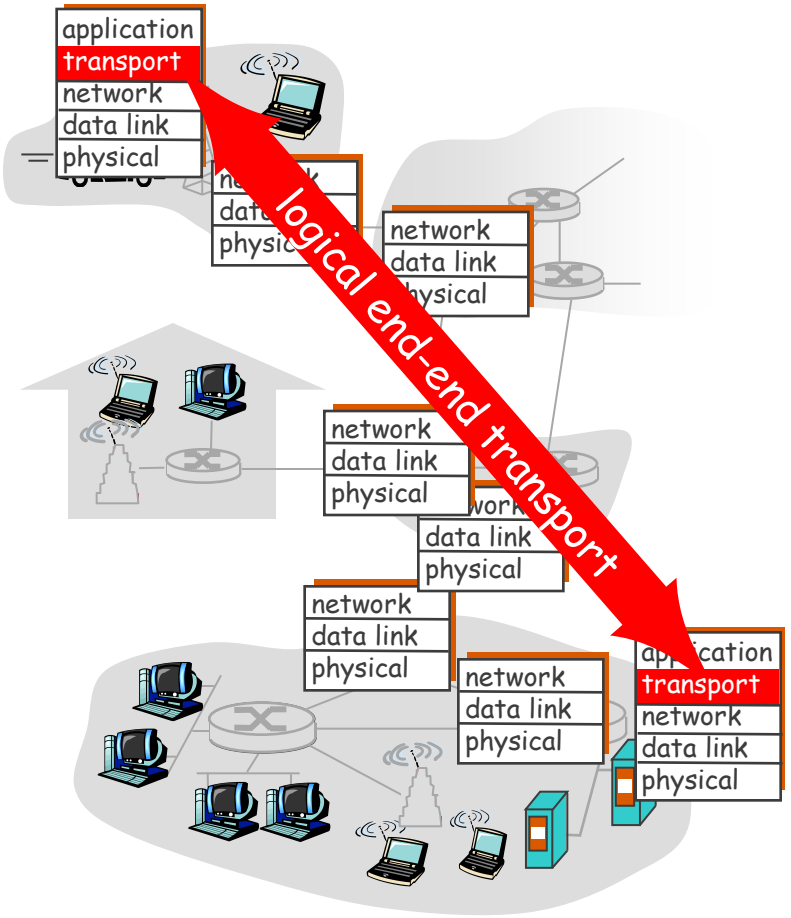
16 bit integer – 0-16 535
Client port – ephemeral port number
Server port – not random

Internet transport-layer protocols

- reliable, in-order delivery (**TCP**)
  - congestion control
  - flow control
  - connection setup

- unreliable, unordered delivery: **UDP**
  - no-frills extension of “best-effort” IP

- services not available:
  - delay guarantees
  - bandwidth guarantees



Chapter 3 outline

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

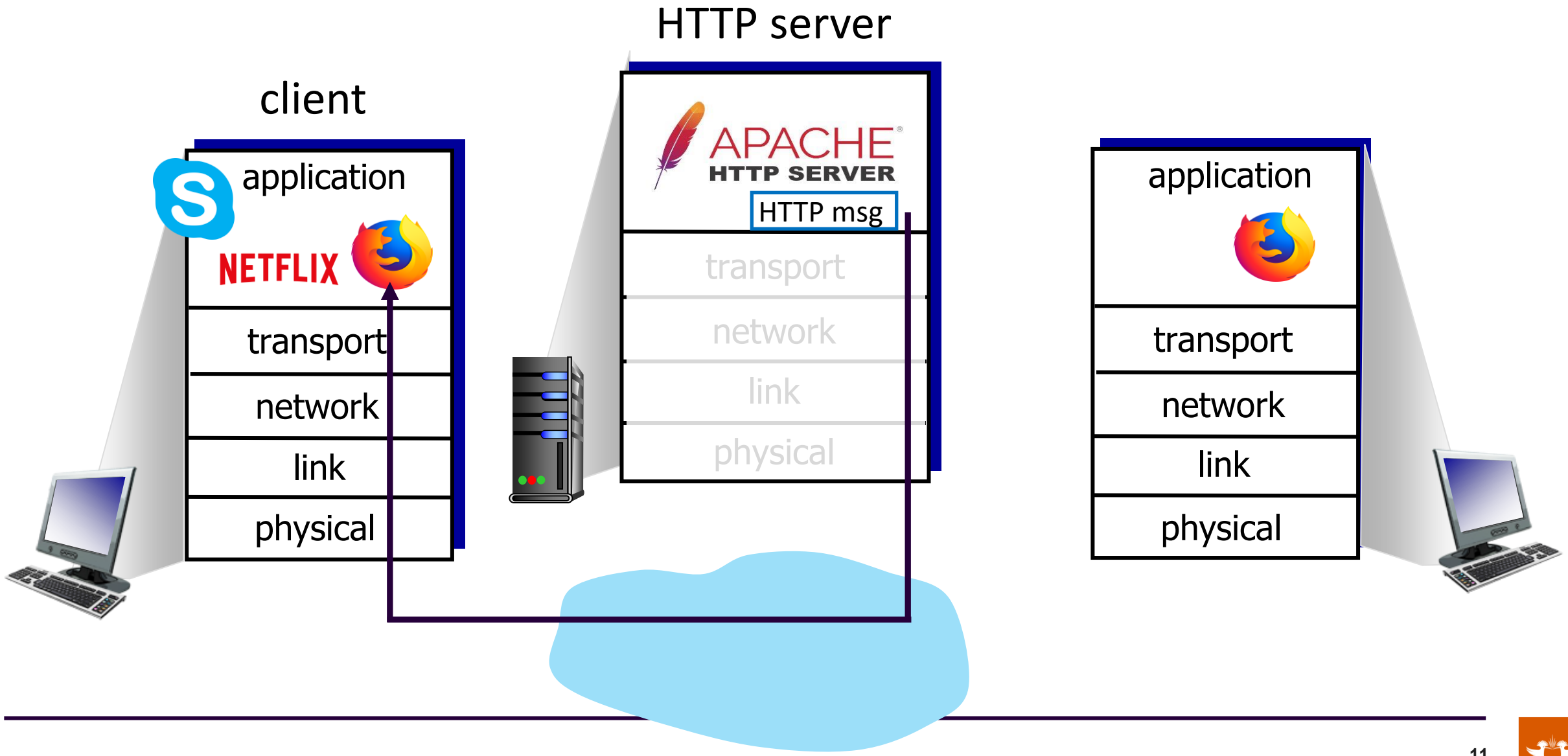
3.4 Principles of reliable data transfer

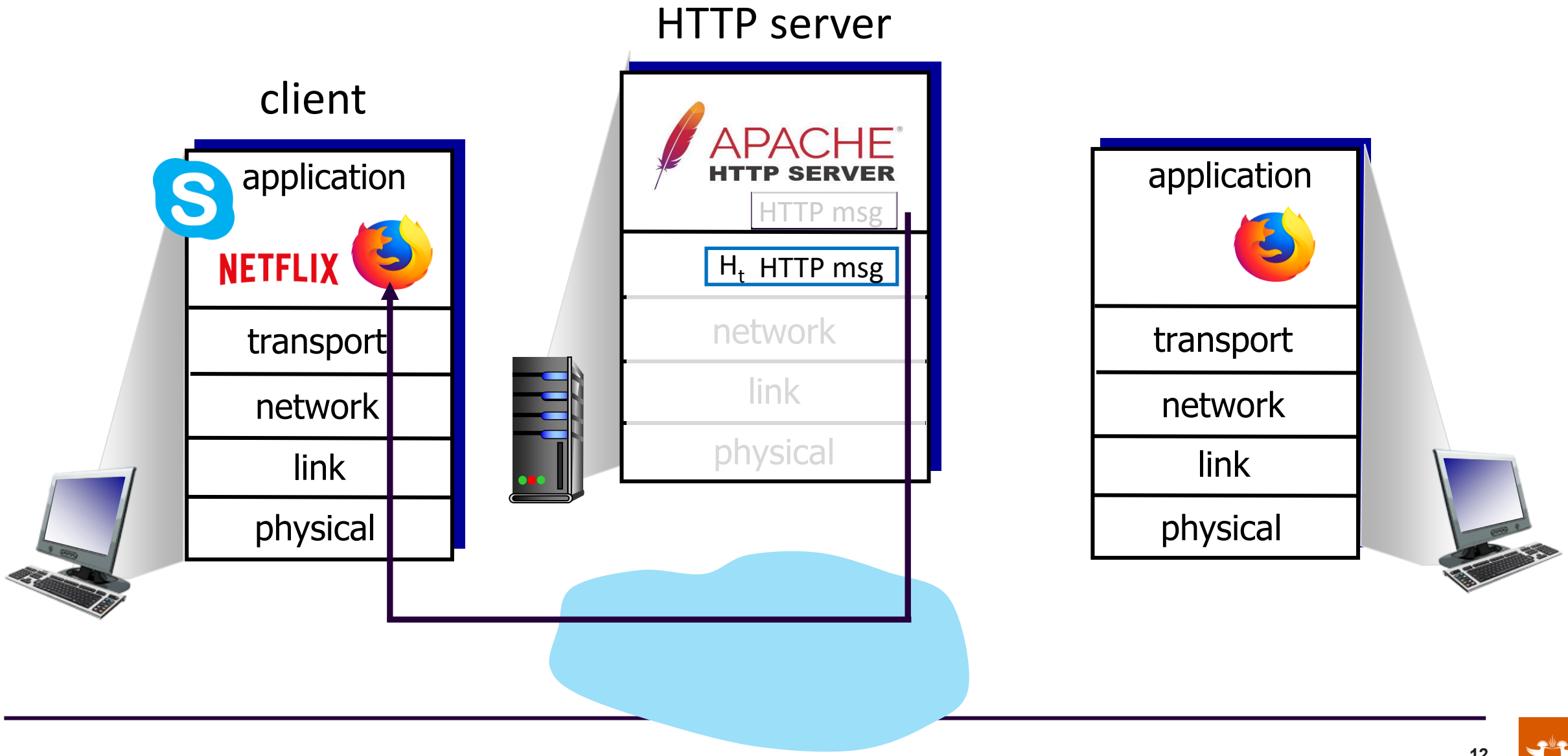
3.5 Connection-oriented transport: TCP

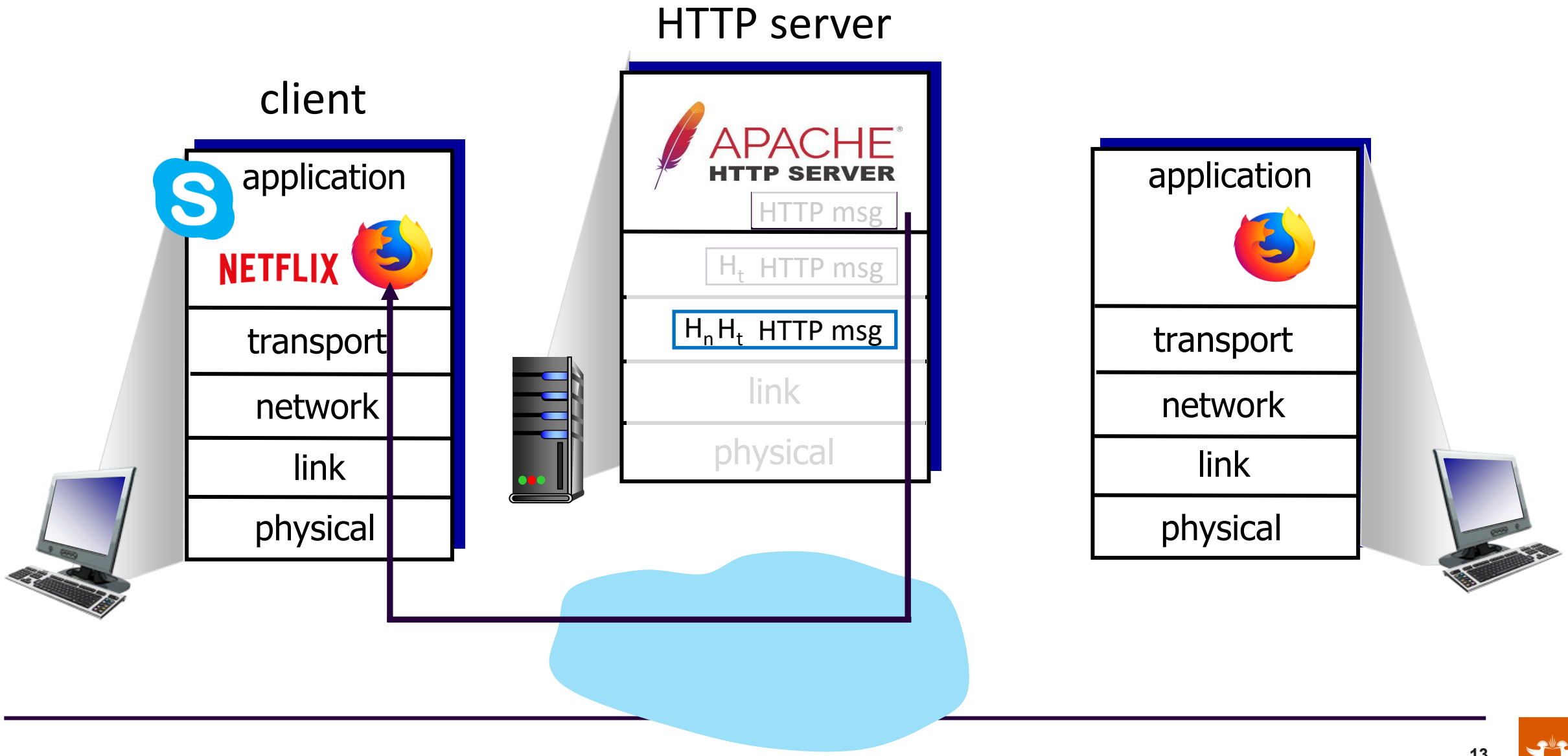
- segment structure
- reliable data transfer
- flow control
- connection management

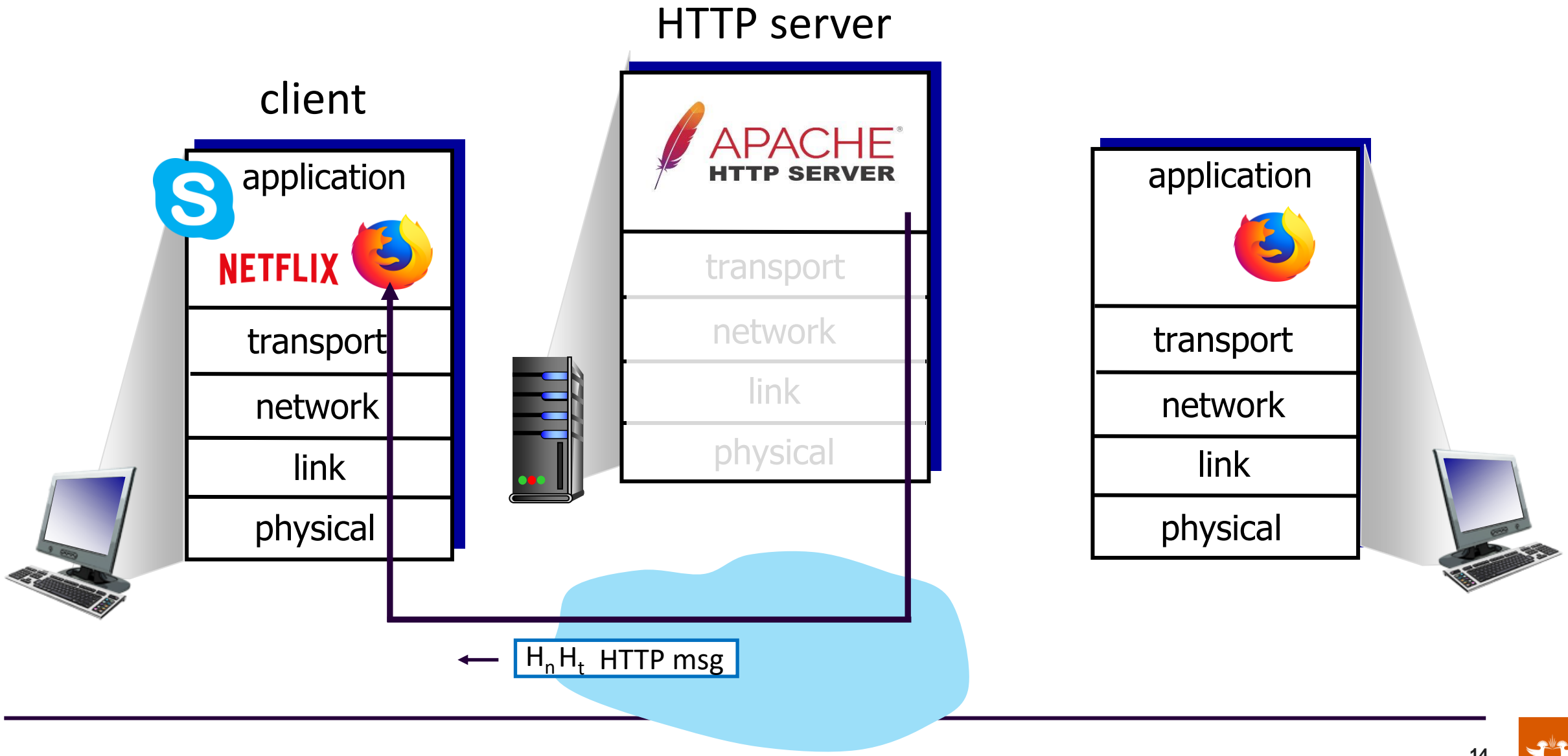
3.6 Principles of congestion control

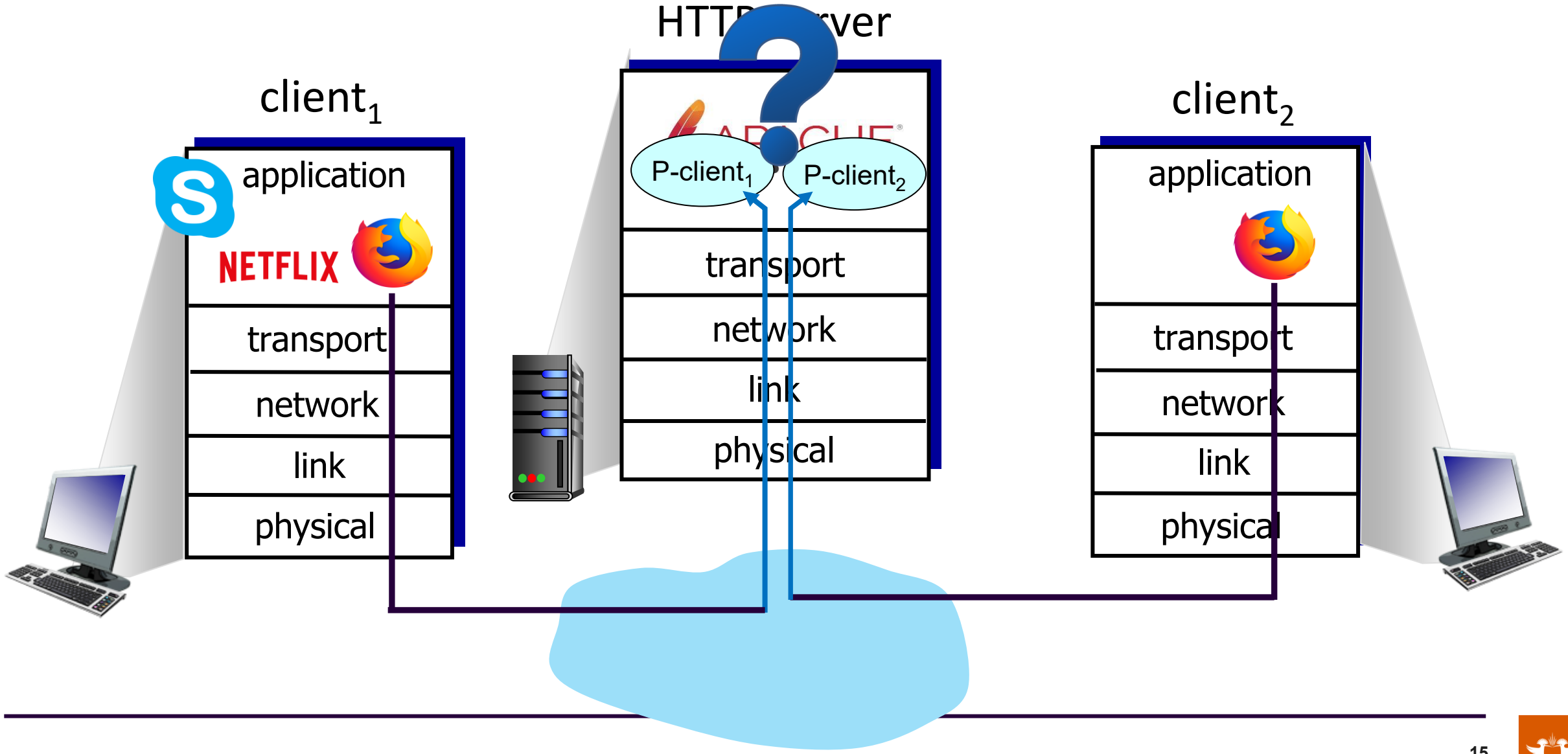
3.7 TCP congestion control


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
transport
network
link
physical
application
transport
network
link
physical


client
application
NETFLIX
transport
network
link
physical
HTTP server
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
$H_n H_t$ HTTP msg
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical
$H_n H_t$ HTTP msg


client1
application
NETFLIX
transport
network
link
physical
HTTP server
P-client1
P-client2
transport
network
link
physical
client2
application
transport
network
link
physical

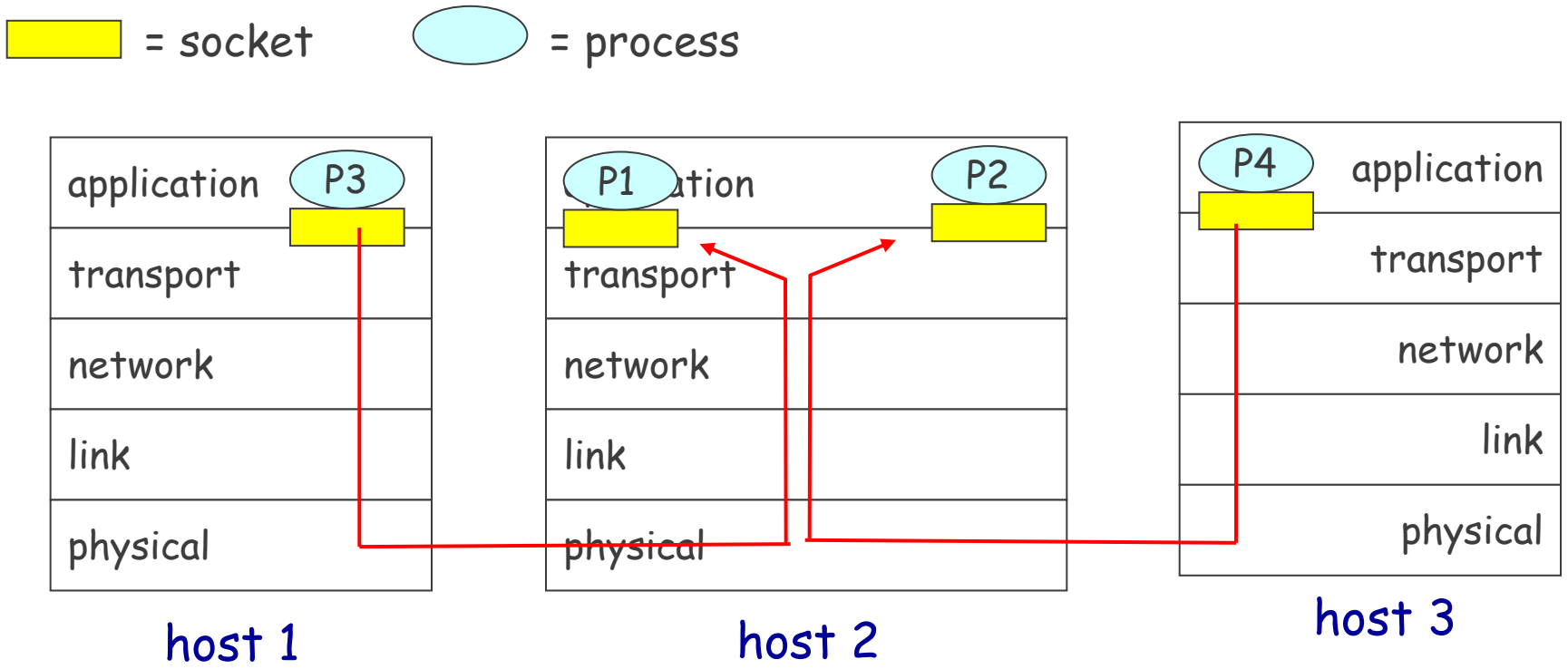
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

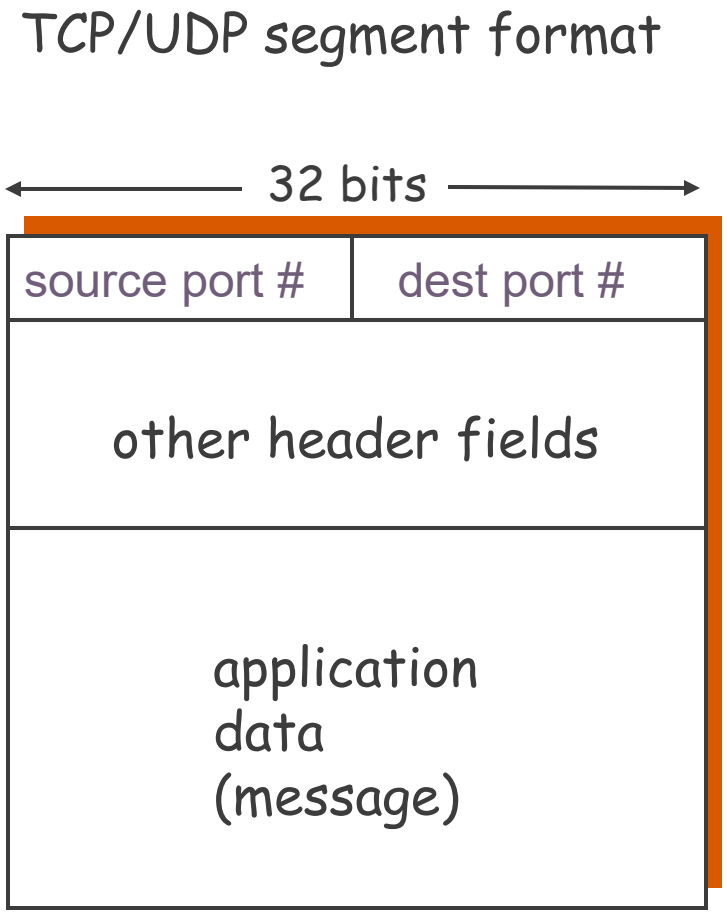
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number

- host uses IP addresses & port numbers to direct segment to appropriate socket



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 #

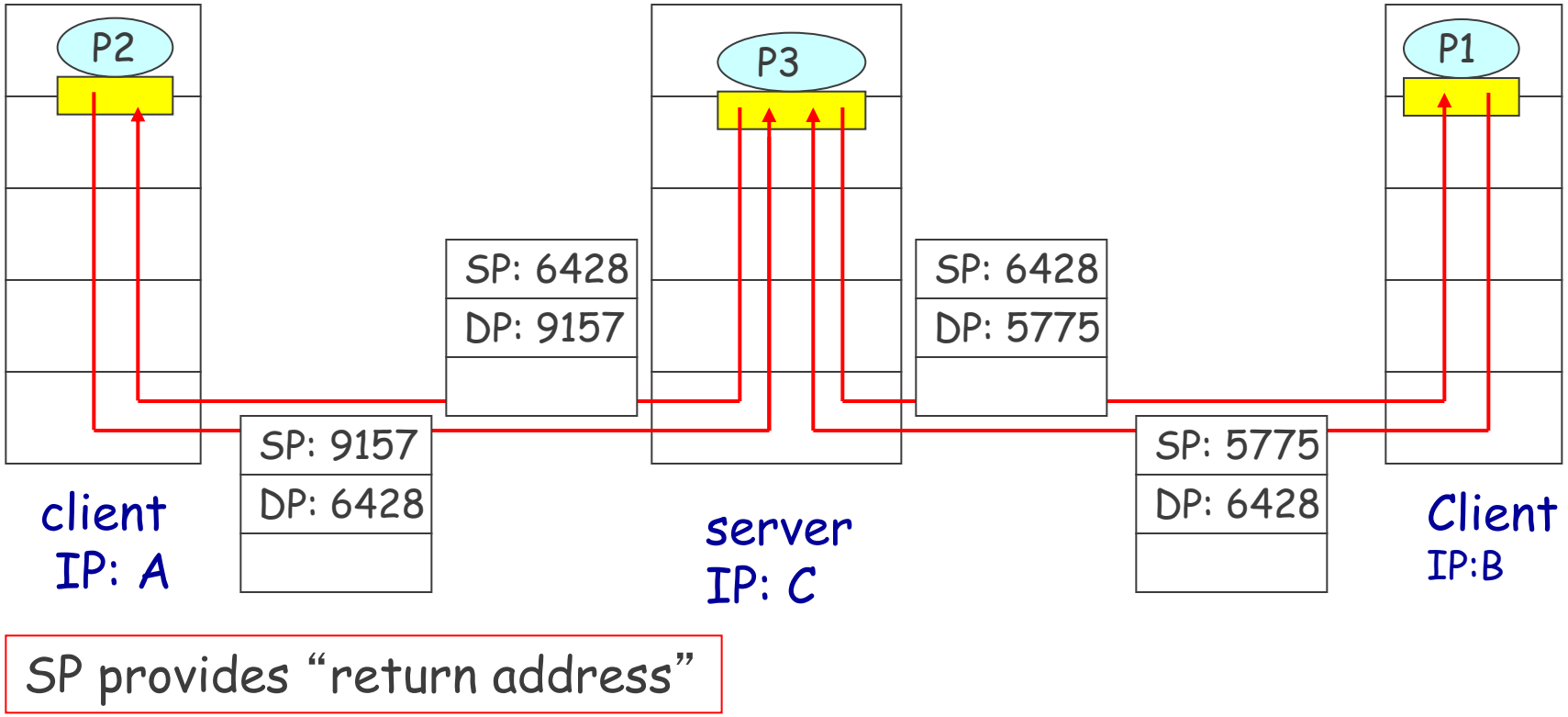
when receiving host receives *UDP* segment:

- 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 demux (cont)

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



SP provides "return address"

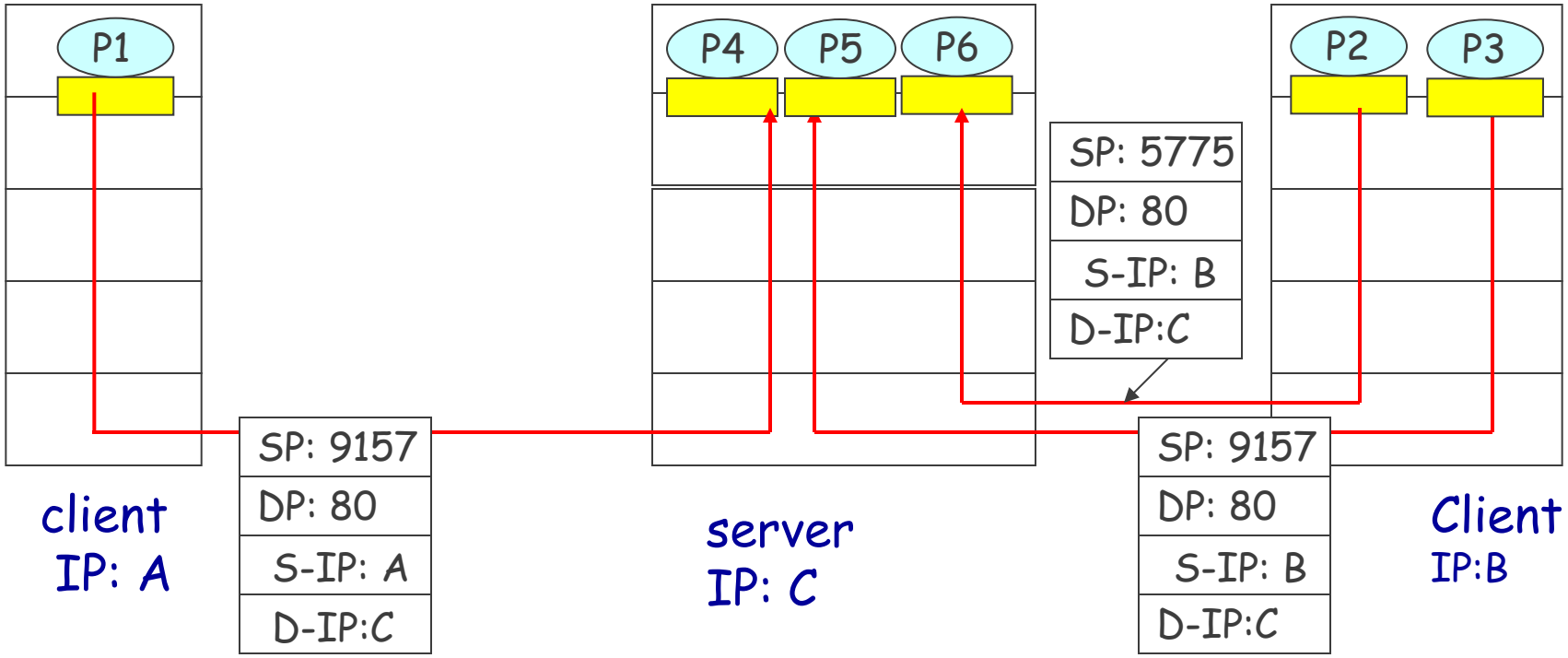
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - **source IP address**
  - **source port number**
  - **dest IP address**
  - **dest port number**

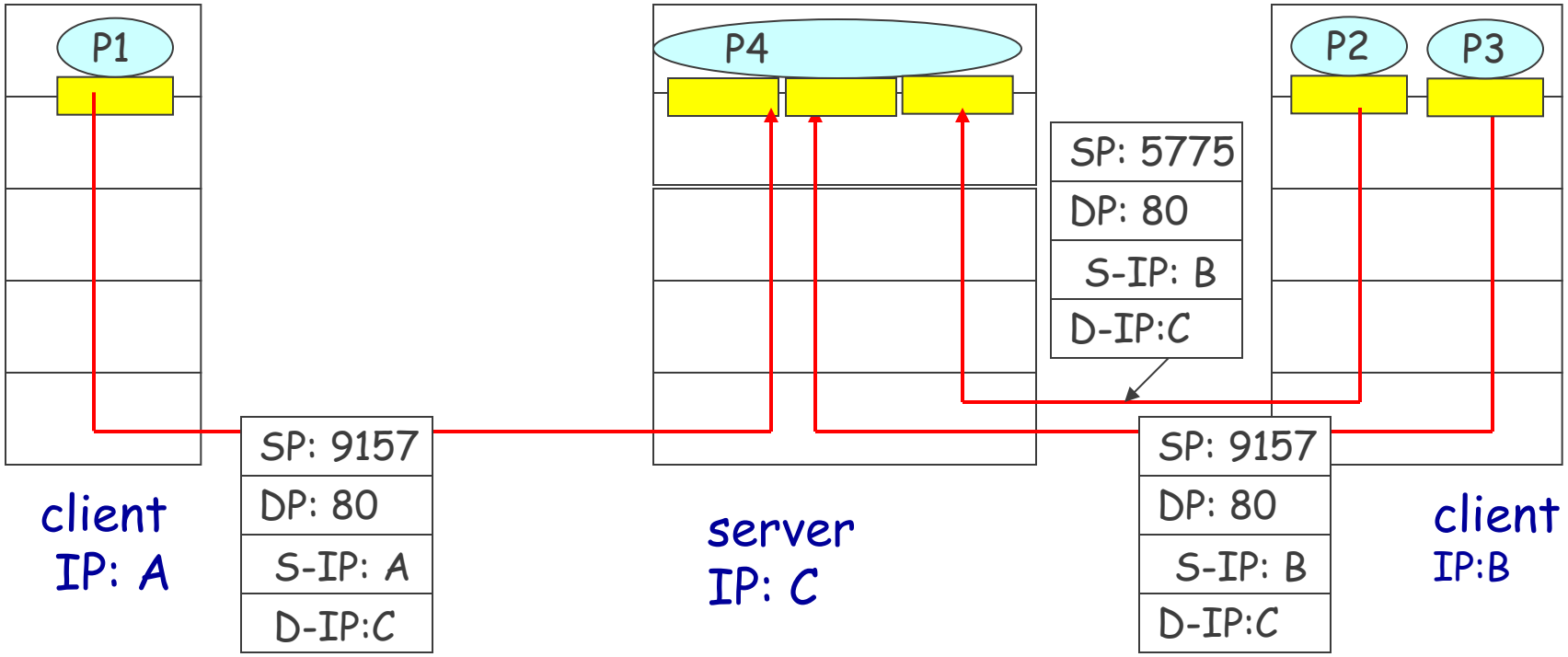
- recv host uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)



Connection-oriented demux: Threaded Web Server



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
- Evolution of transport-layer functionality

UDP: User Datagram Protocol [RFC 768]

- "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?

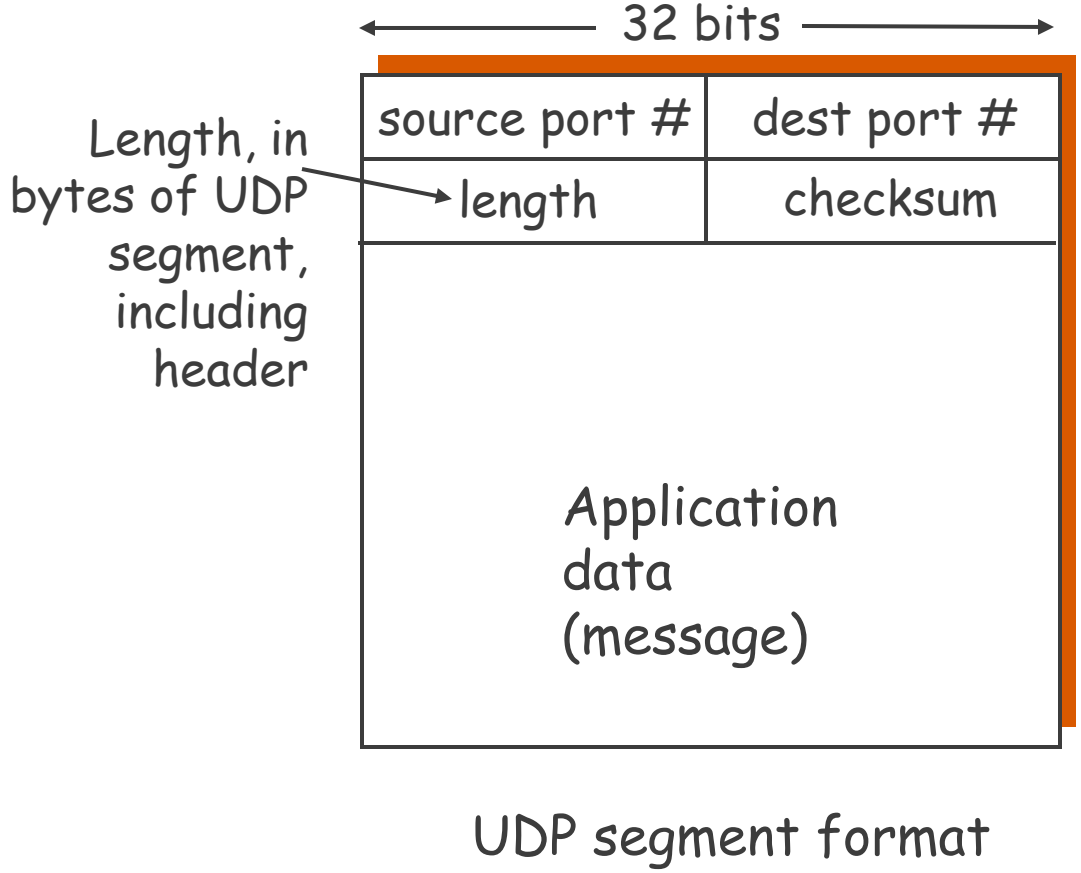
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive

- other UDP uses
  - DNS
  - SNMP

- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP checksum

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

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

Internet Checksum Example

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

Example: add two 16-bit integers