NETWORK PROTOCOLS

Asst. Prof. DR. MUHANED TH. M. AL-HASHIMI

Tikrit University

Collage Of Computer And Mathematical Science

2024 - 2025

TRANSPORT LAYER
AND
TRANSPORT LAYER PROTOCOLS

LECTURE (4) PART A

2204 - 2025

14 October

Our goal

Our goal In this lecture is to:

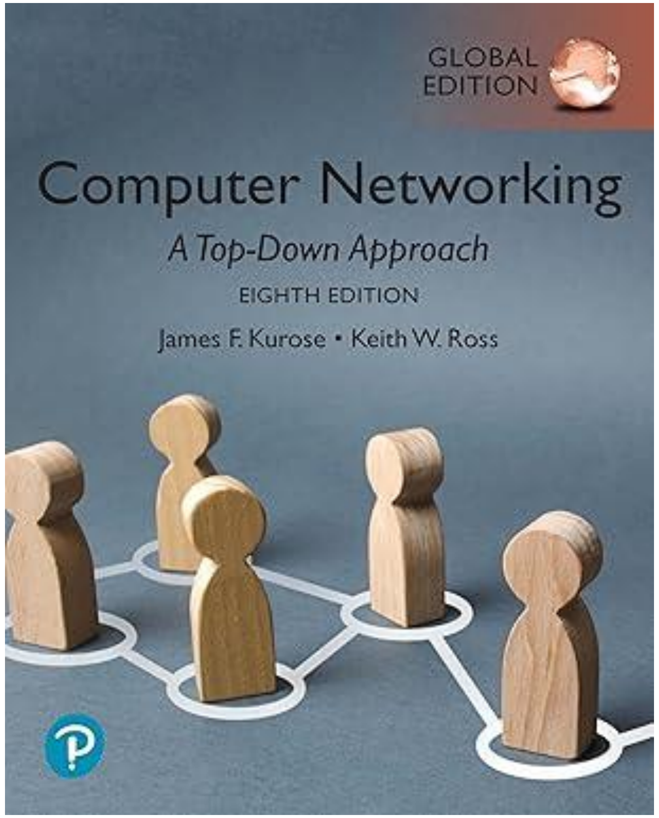
- **understand principles behind transport layer services:**
  - **multiplexing, demultiplexing**
  - reliable data transfer
  - flow control
  - congestion control

- **learn about Internet transport layer protocols:**
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Transport layer: roadmap

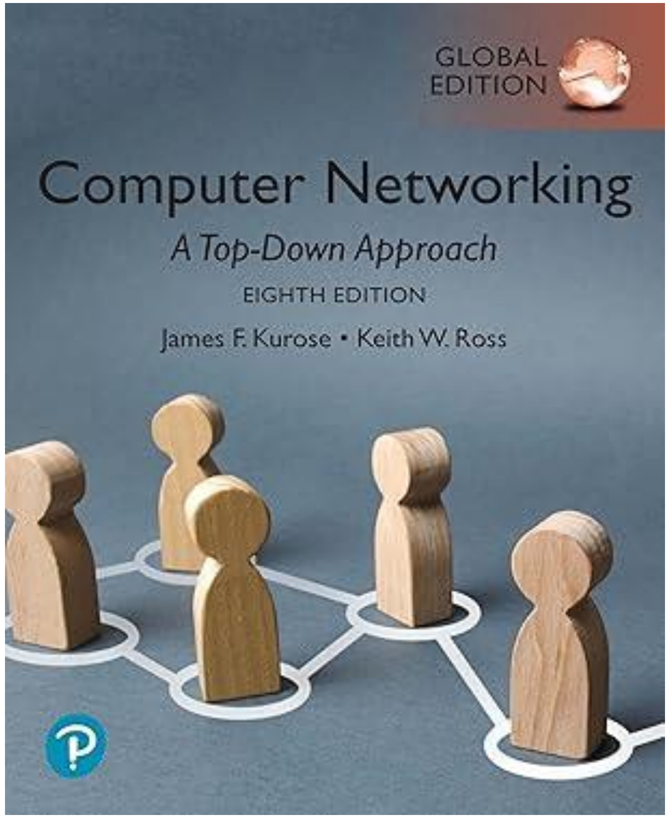
In this lecture part A will talk about the following:

- **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



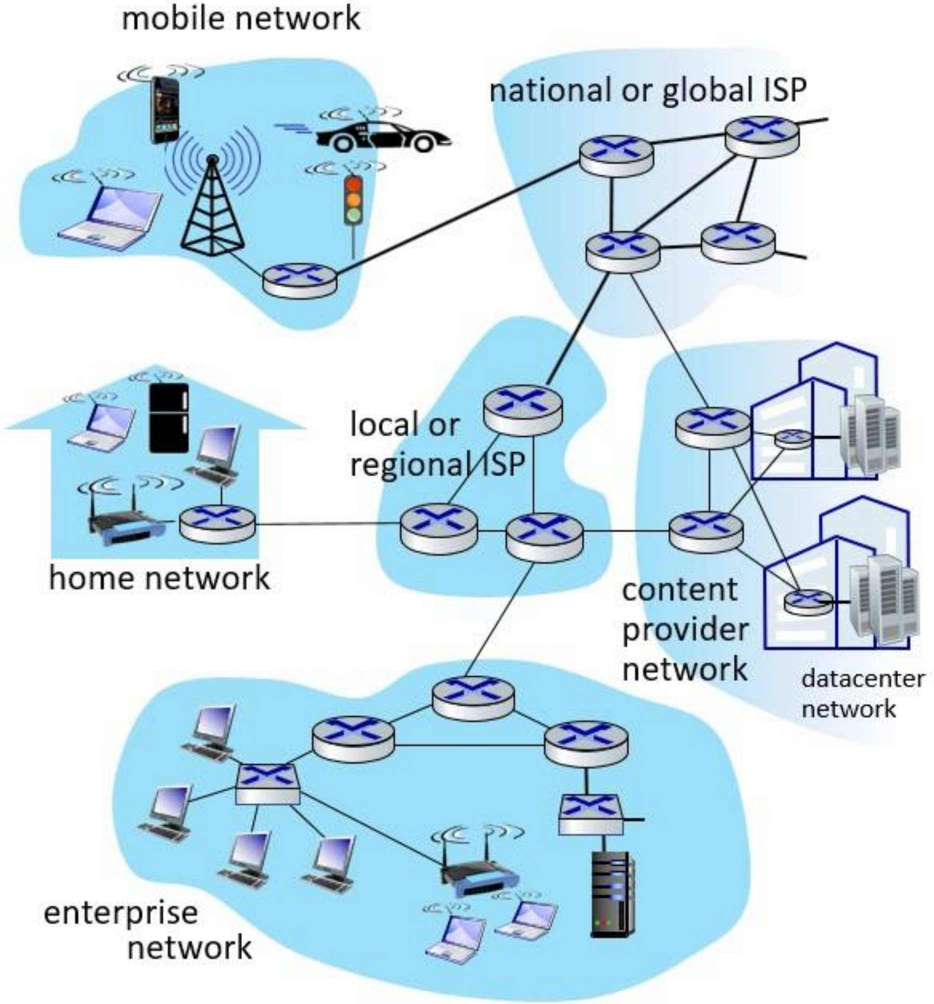
Transport layer: roadmap

- 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



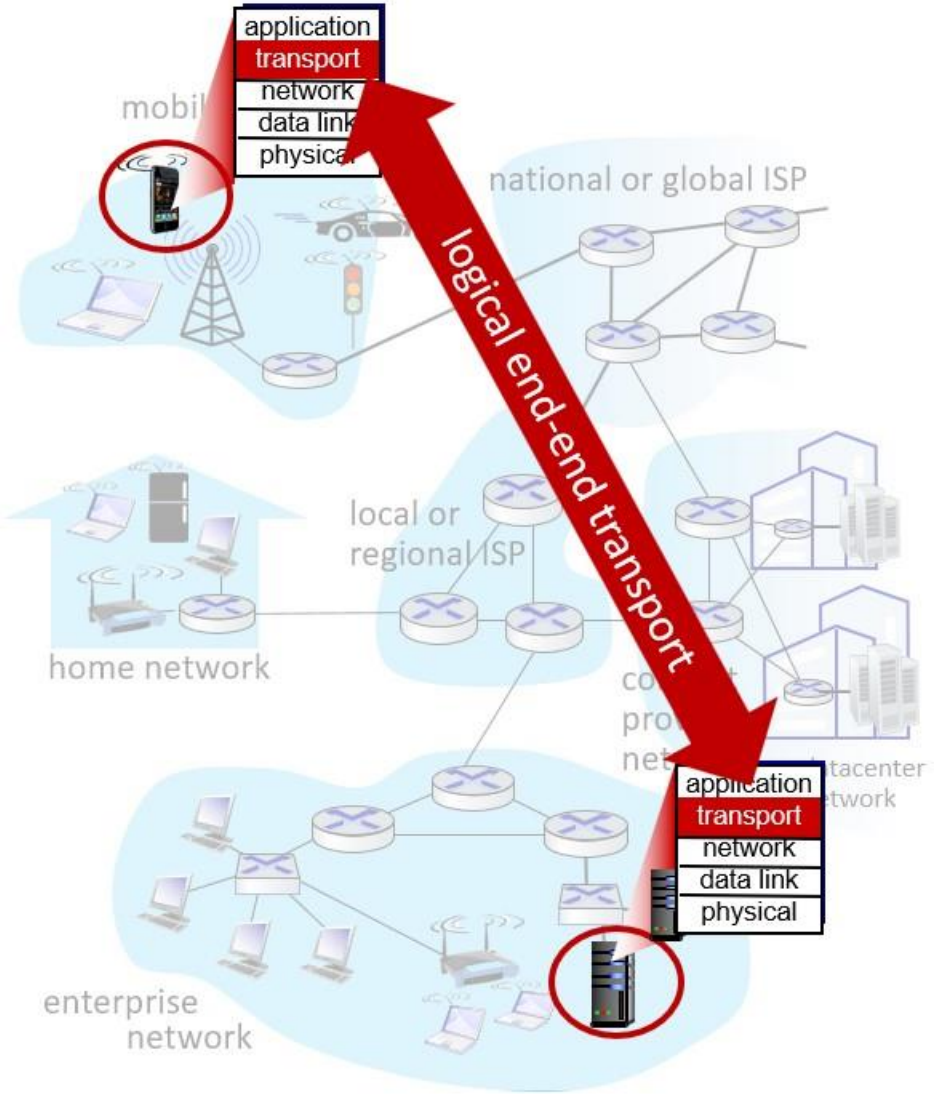
Transport services and protocols

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



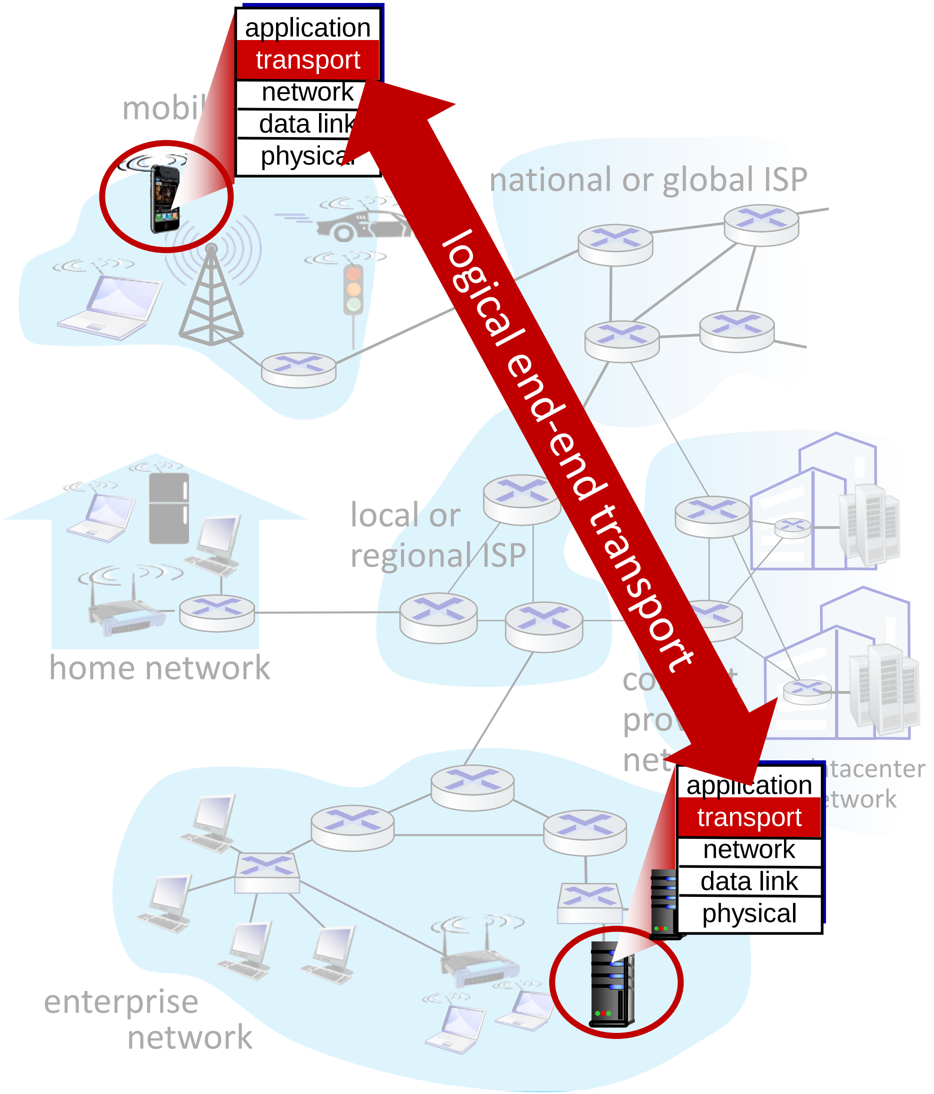
Transport services and protocols

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



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

- transport layer*:* communication between *processes*
  - relies on يعتمد على, 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 vs. network layer services and protocols

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

- network layer: communication between *hosts*

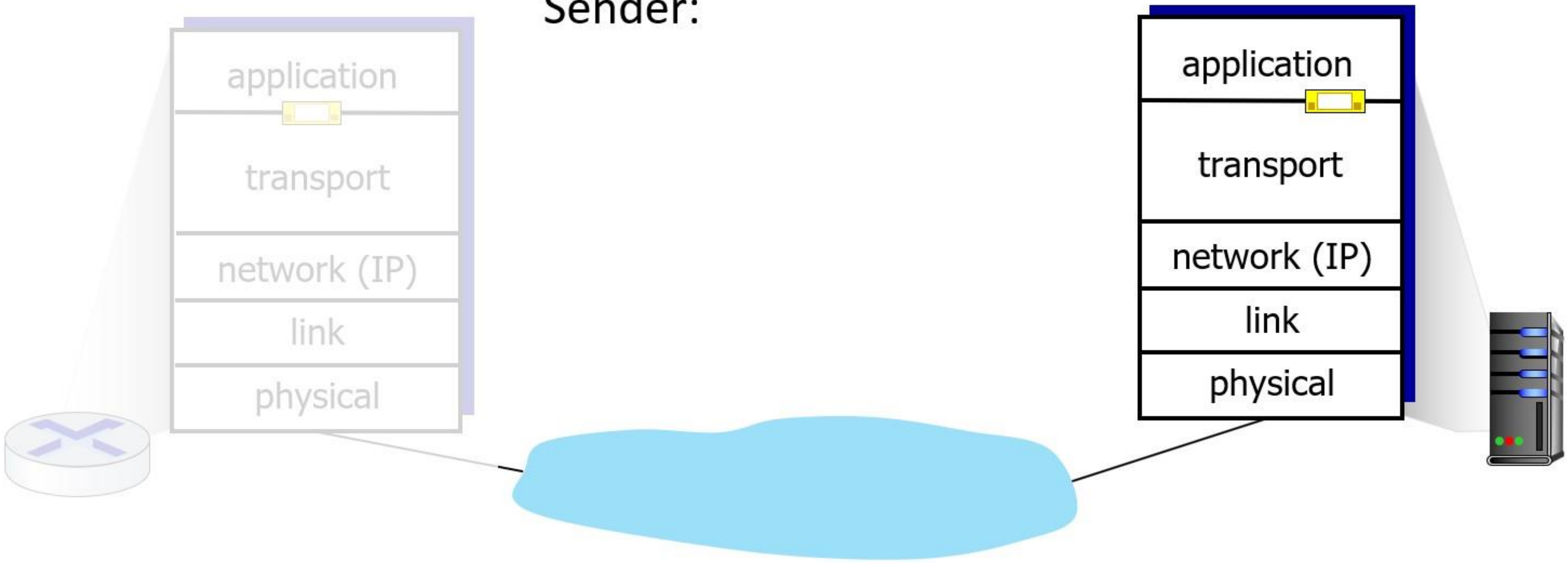
household analogy:

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

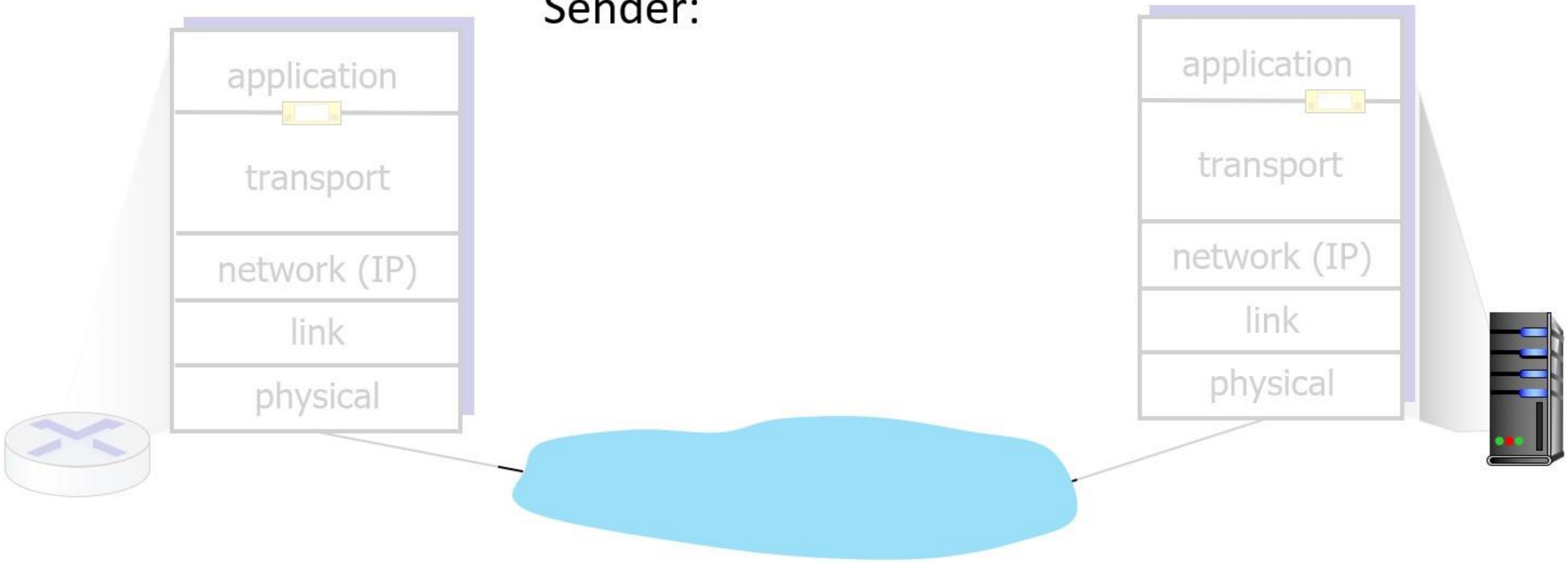
Transport Layer Actions

Sender:



Transport Layer Actions

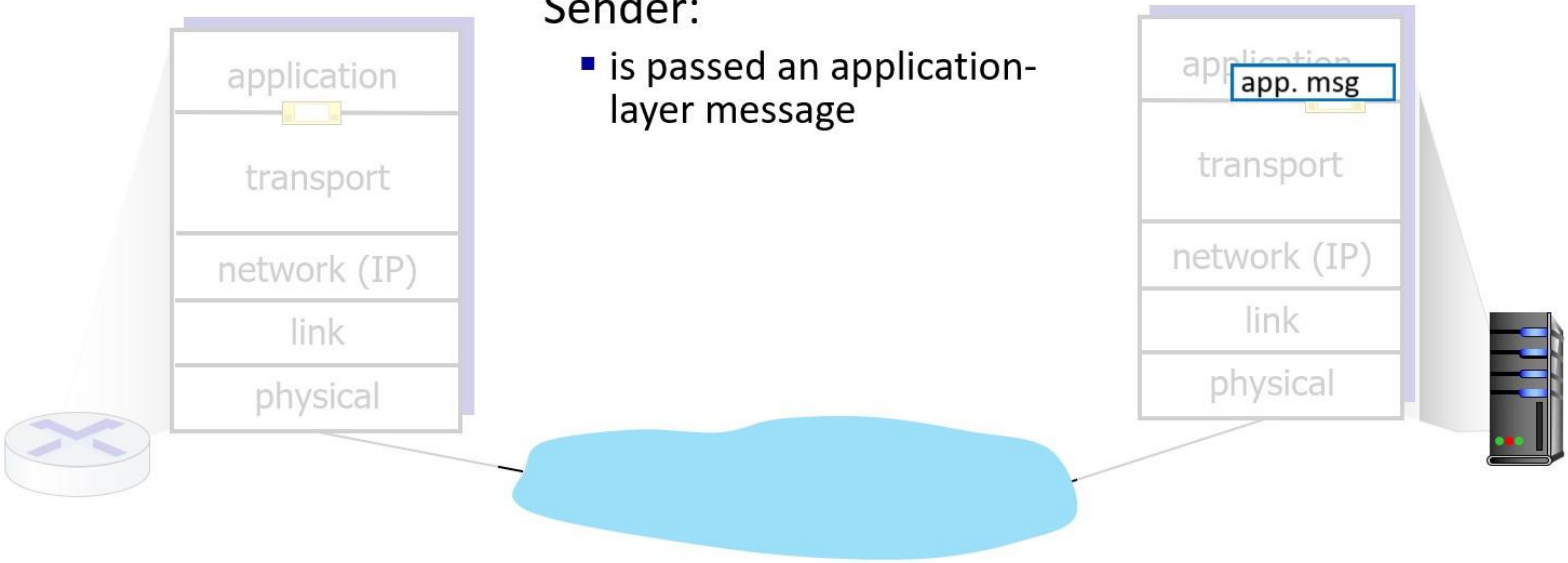
Sender:



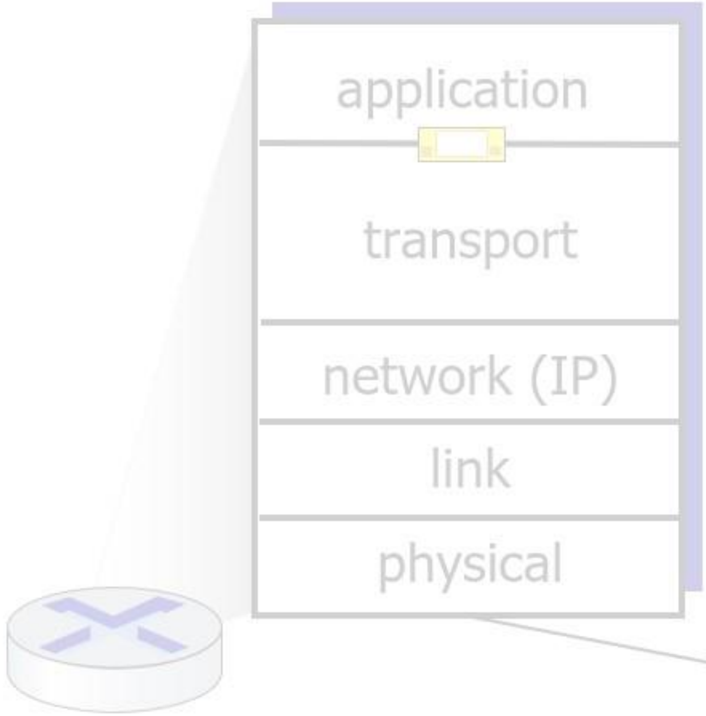
Transport Layer Actions

Sender:

- is passed an application-layer message

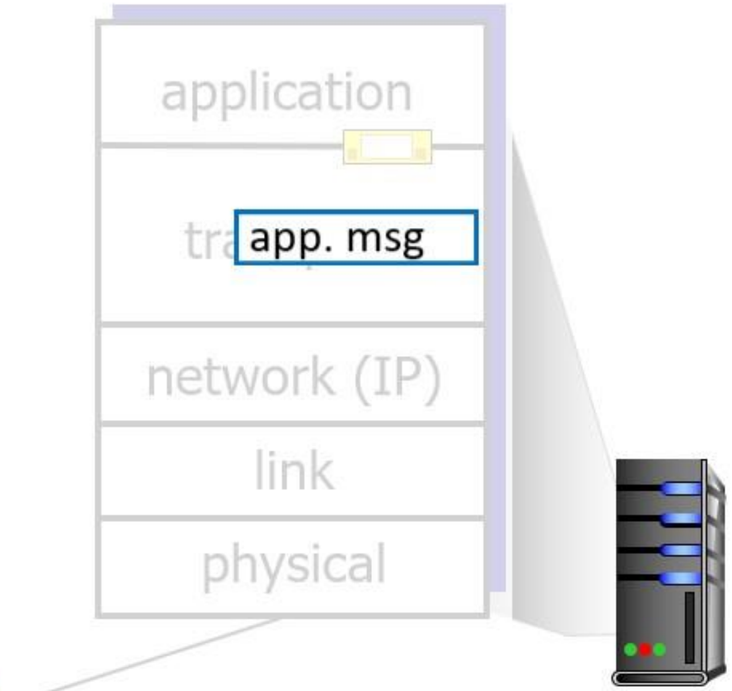


Transport Layer Actions



Sender:

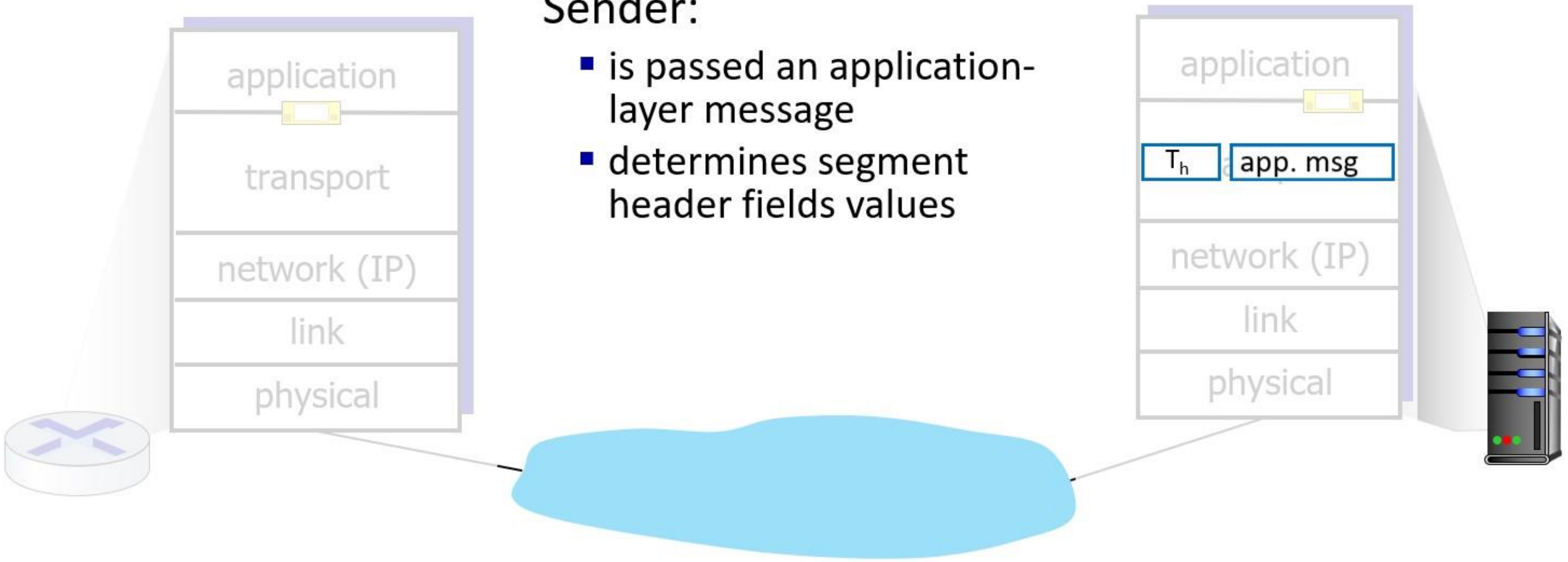
- is passed an application-layer message



Transport Layer Actions

Sender:

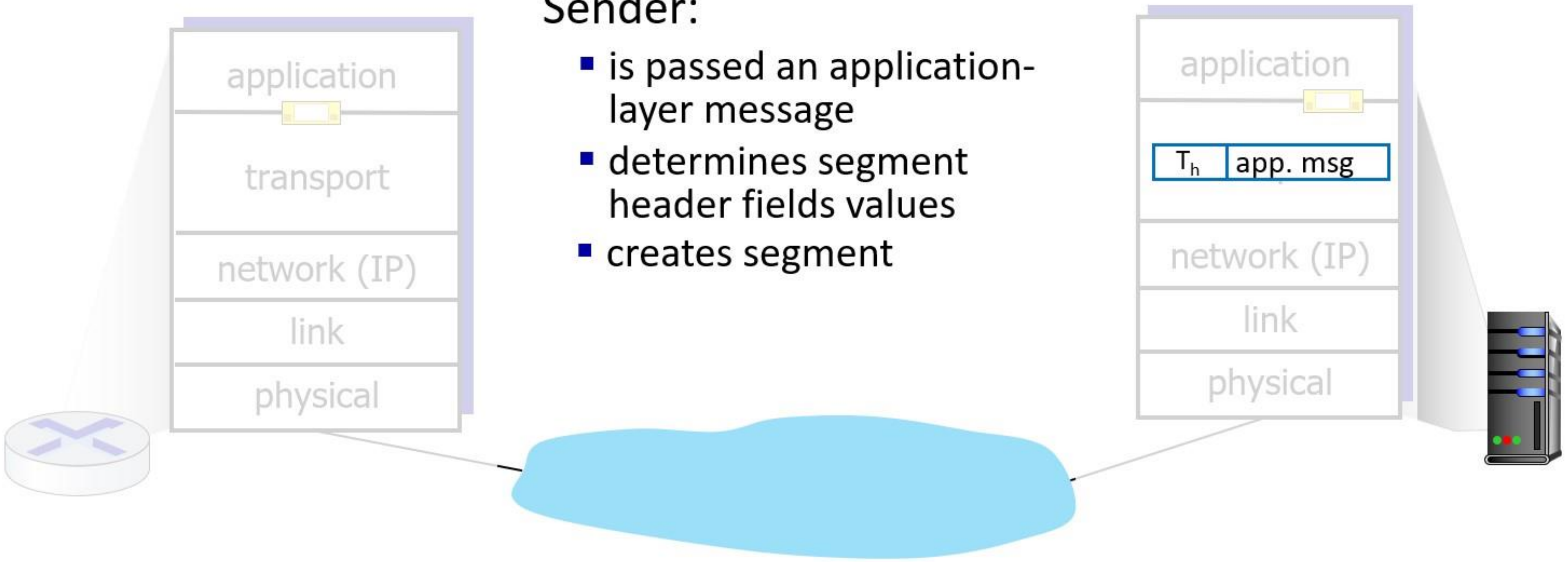
- is passed an application-layer message
- determines segment header fields values



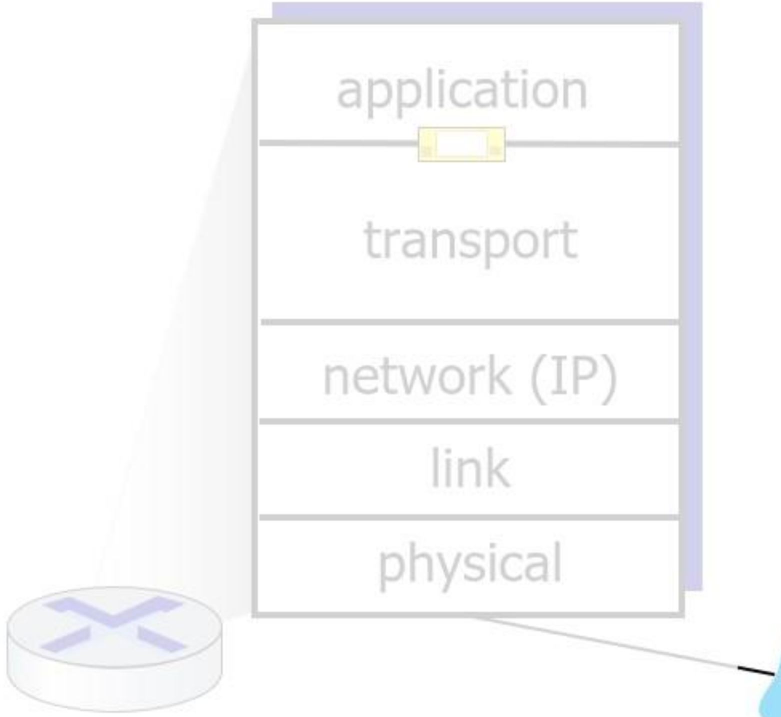
Transport Layer Actions

Sender:

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

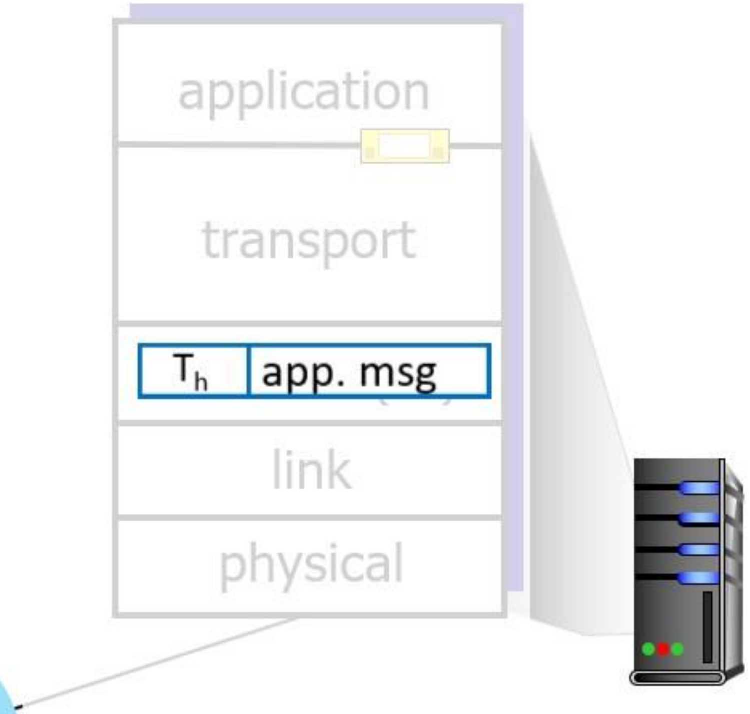


Transport Layer Actions



Sender:

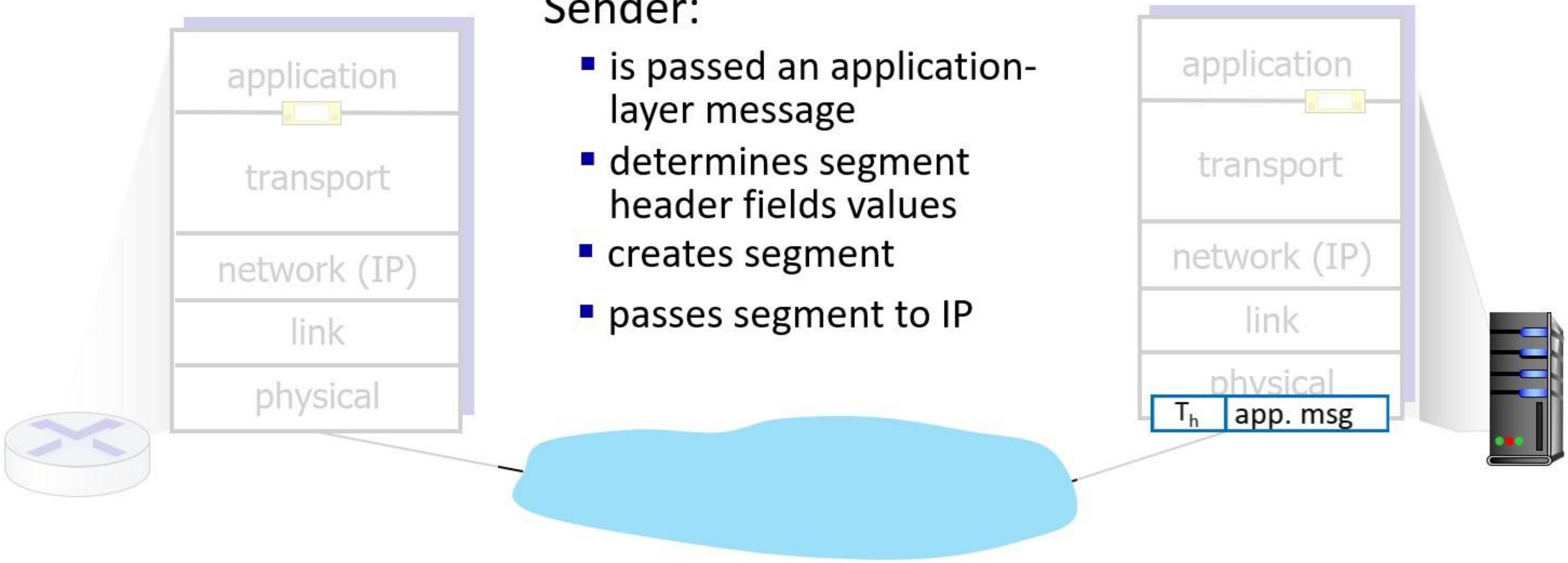
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



Transport Layer Actions

Sender:

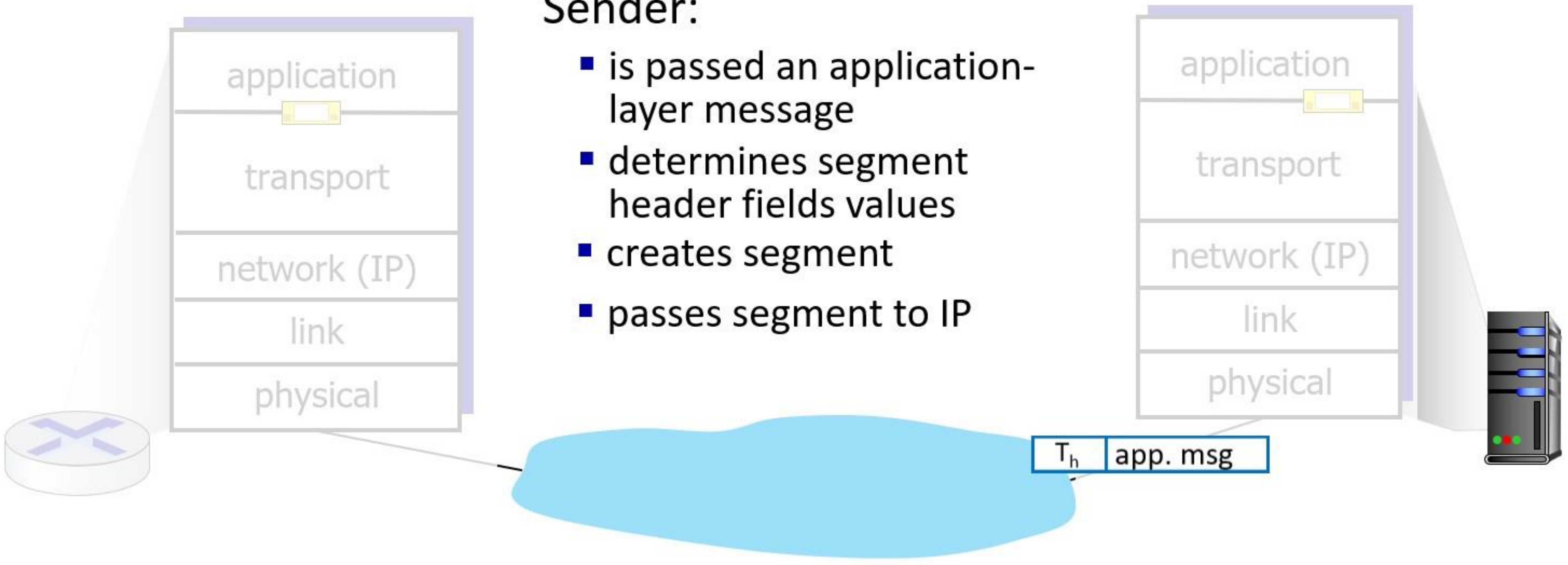
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



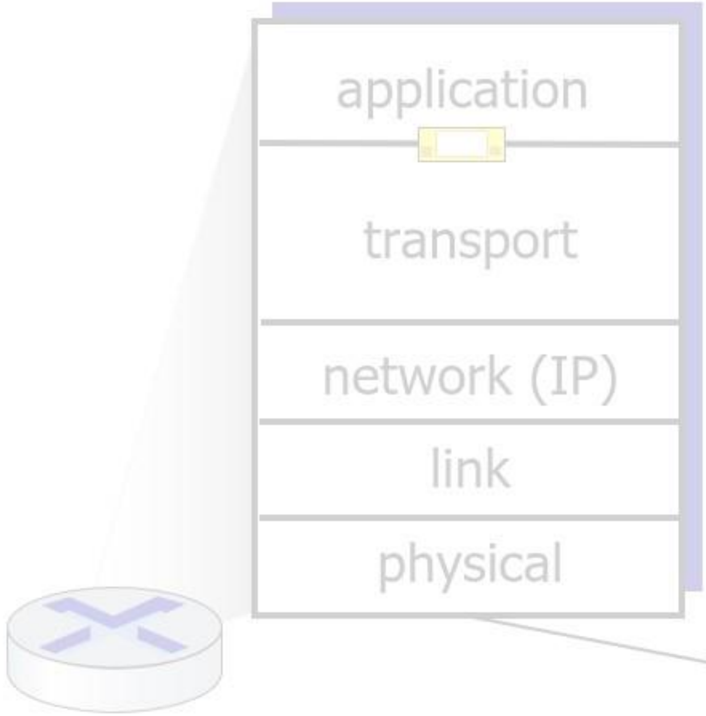
Transport Layer Actions

Sender:

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

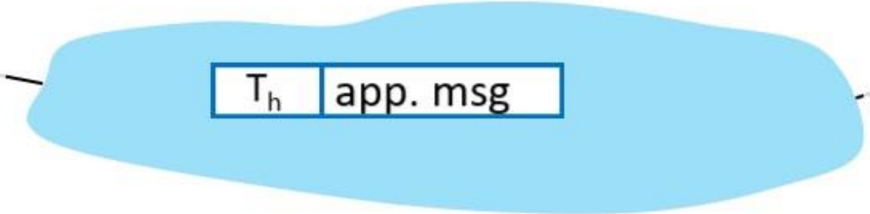


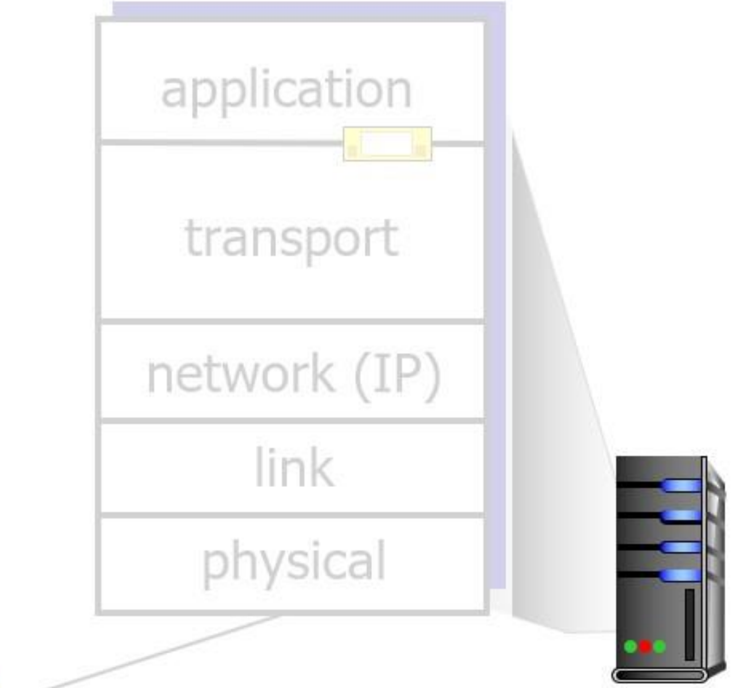
Transport Layer Actions



Sender:

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

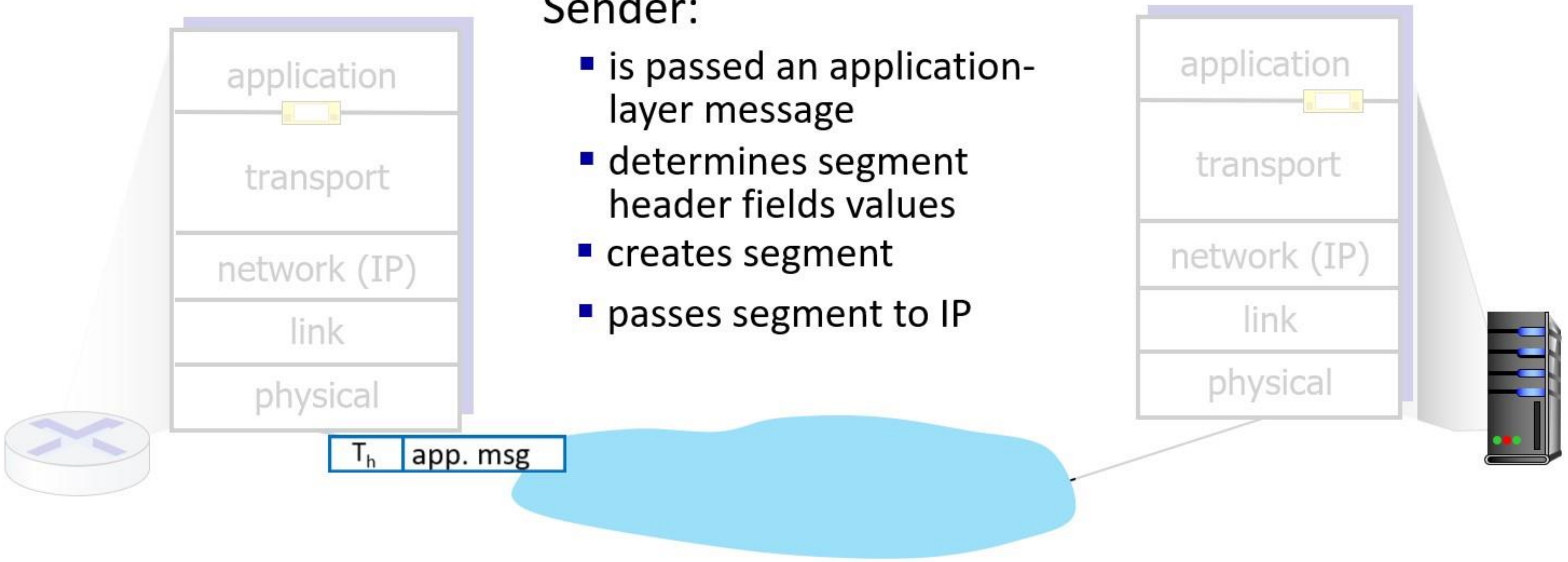




Transport Layer Actions

Sender:

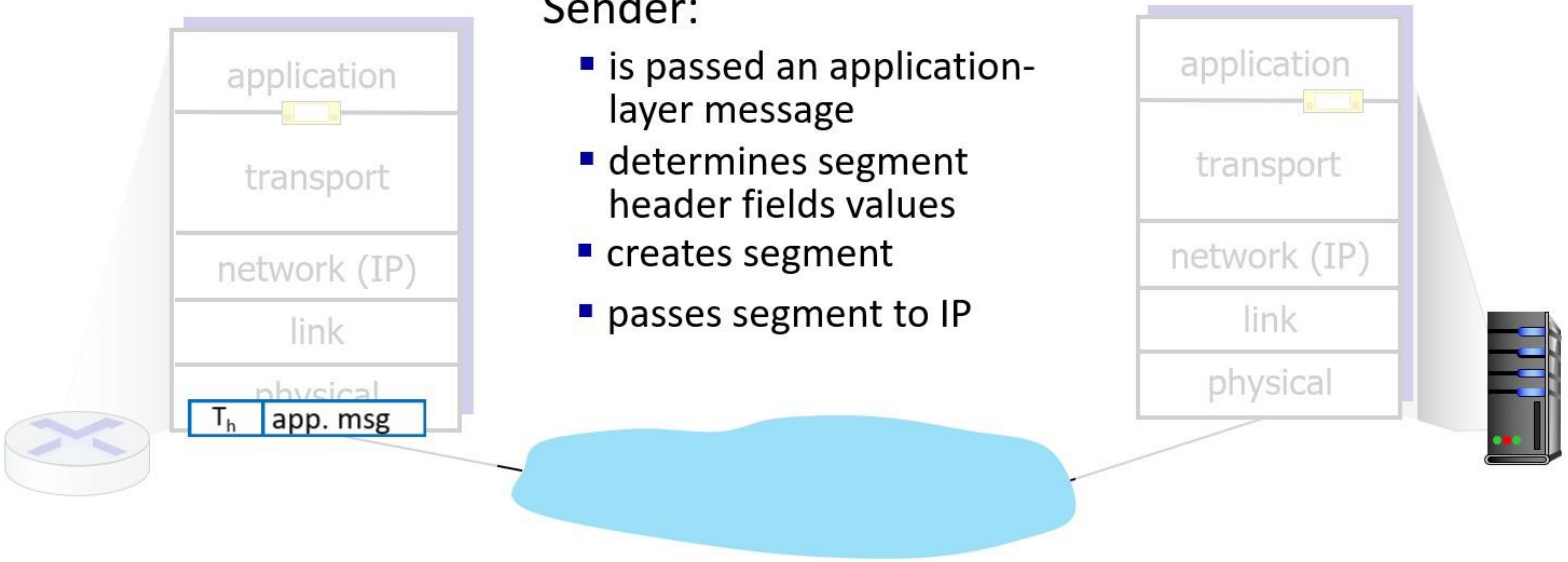
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



Transport Layer Actions

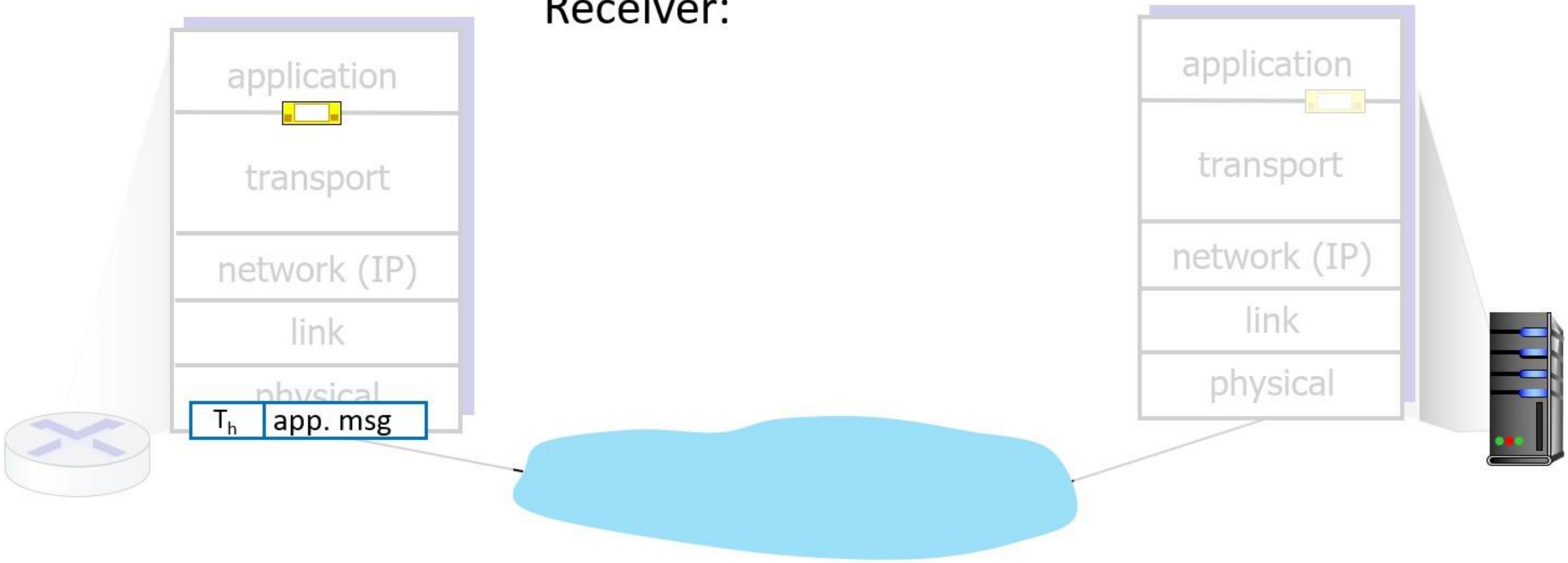
Sender:

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



Transport Layer Actions

Receiver:



Transport Layer Actions

Receiver:

- receives segment from IP

Transport Layer Actions

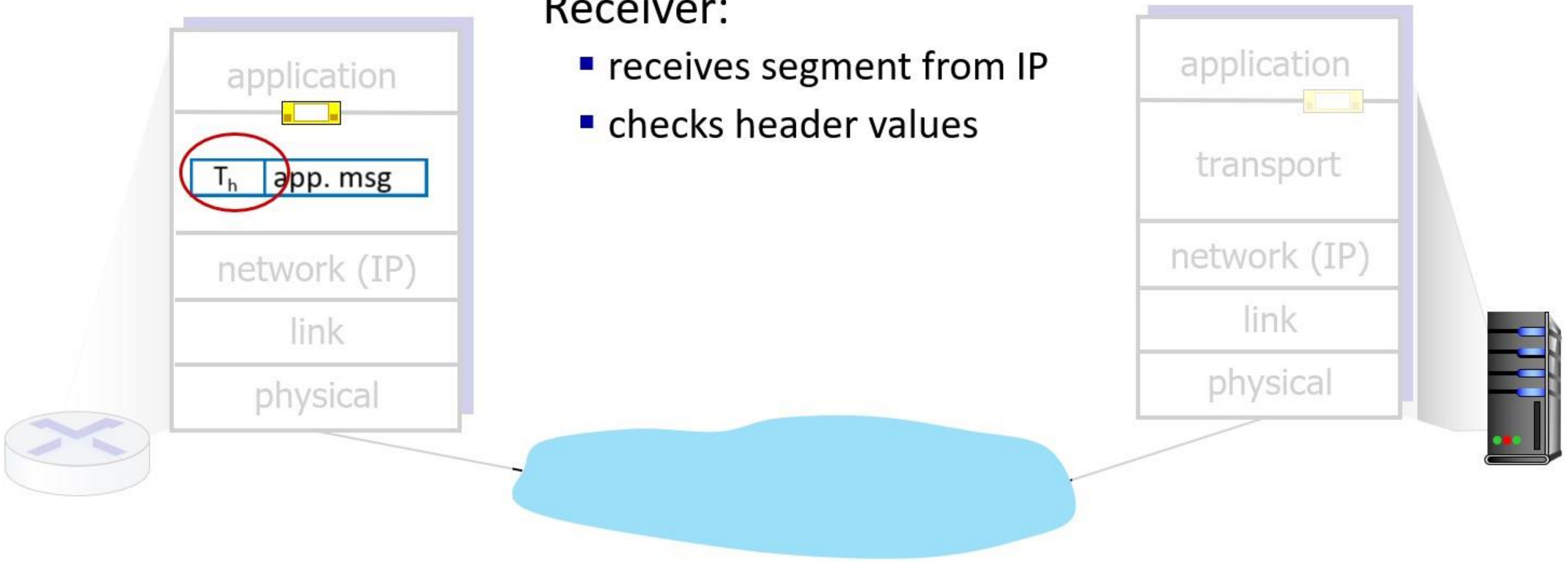
Receiver:

- receives segment from IP

Transport Layer Actions

Receiver:

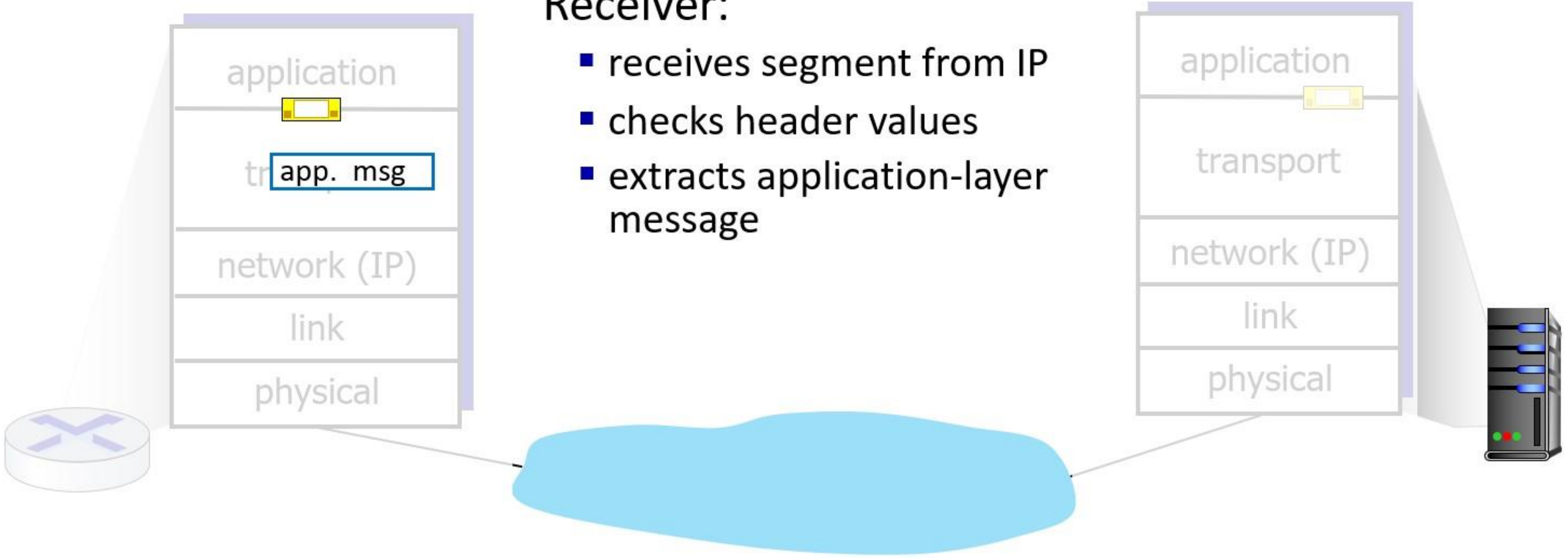
- receives segment from IP
- checks header values



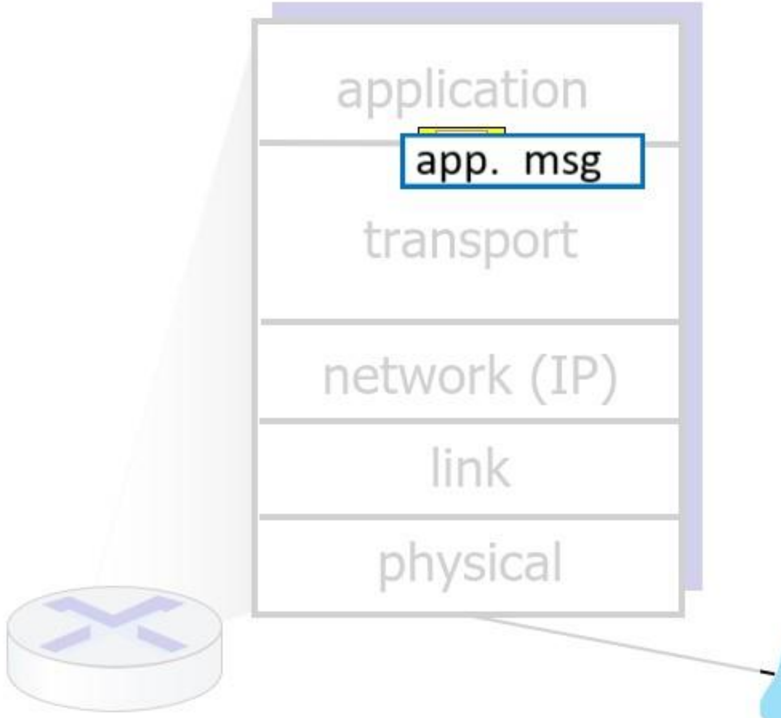
Transport Layer Actions

Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message

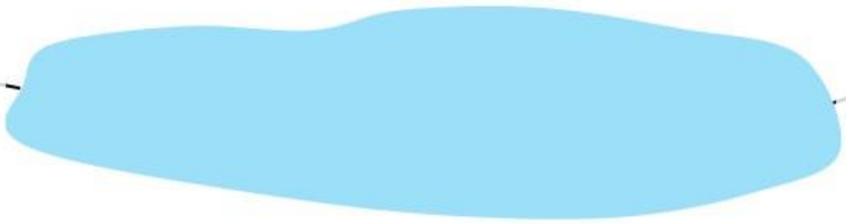


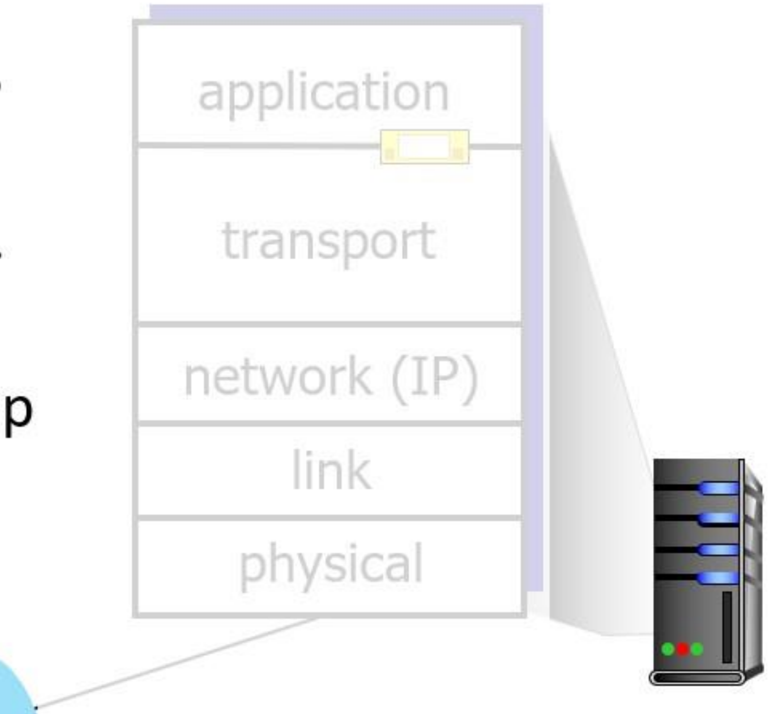
Transport Layer Actions



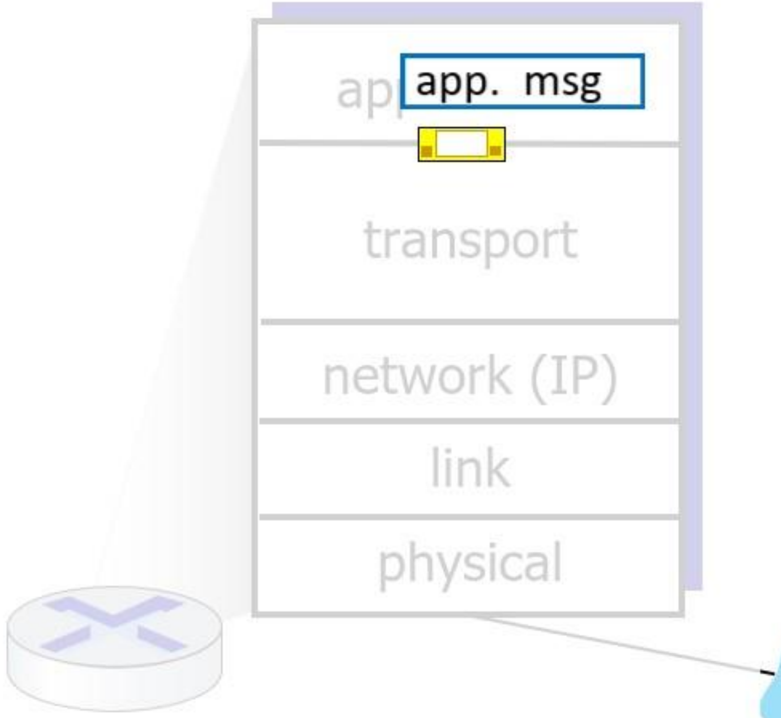
Receiver:

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



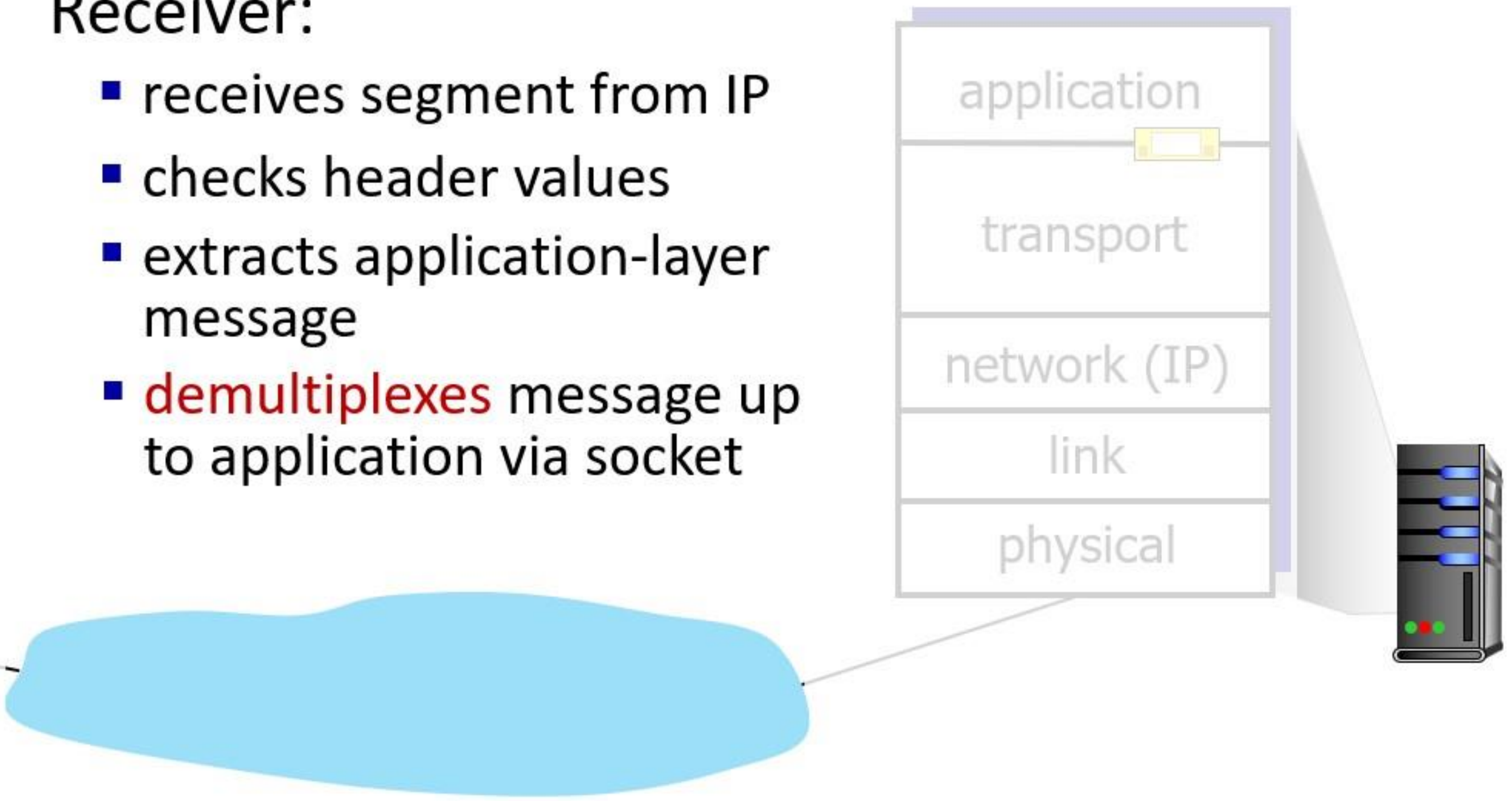


Transport Layer Actions



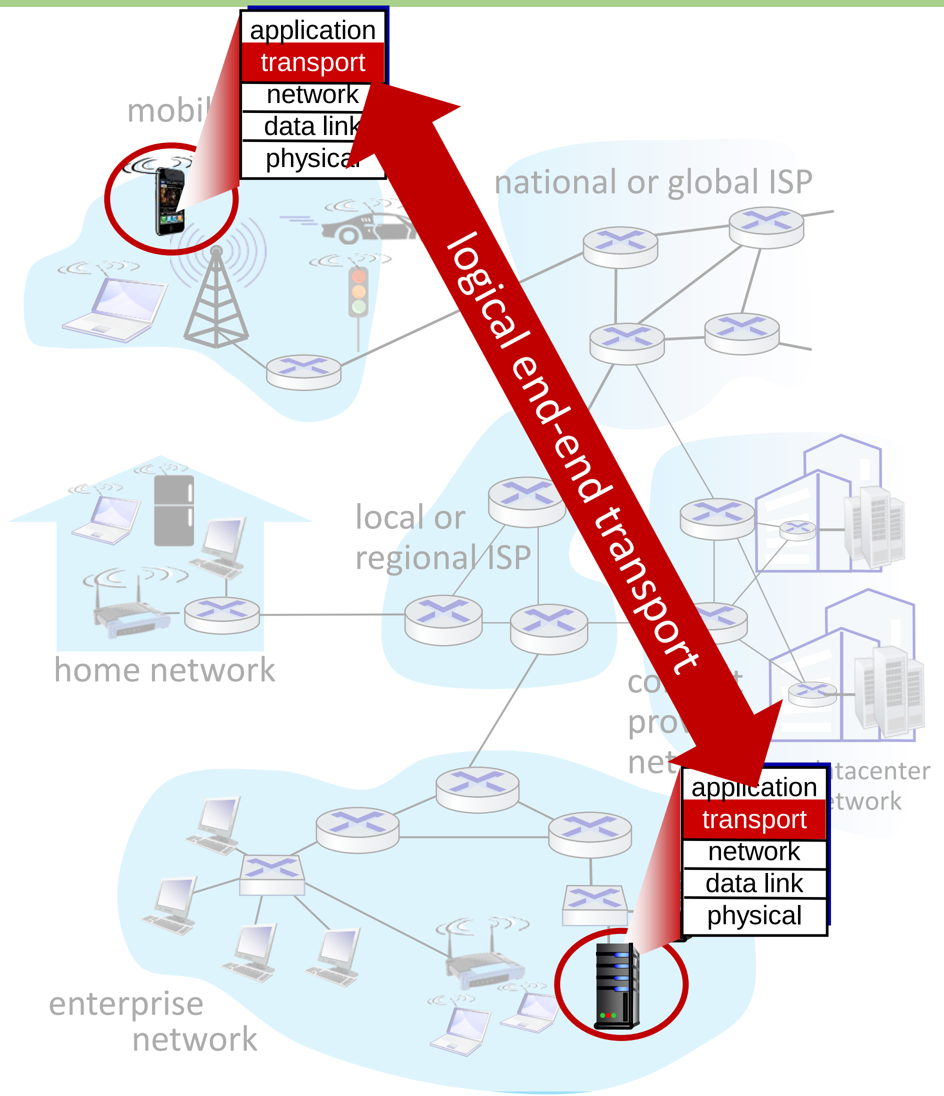
Receiver:

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



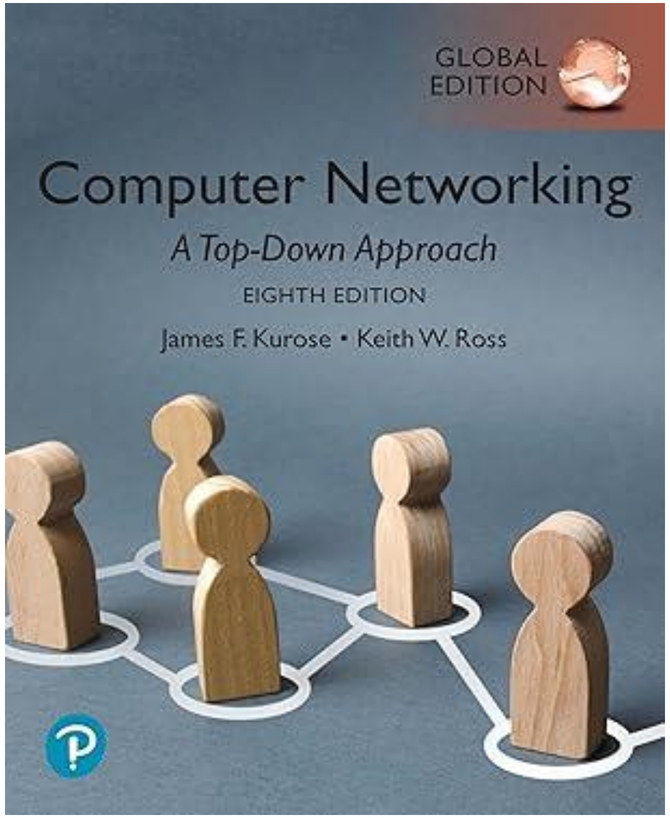
Two principal Internet transport protocols

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

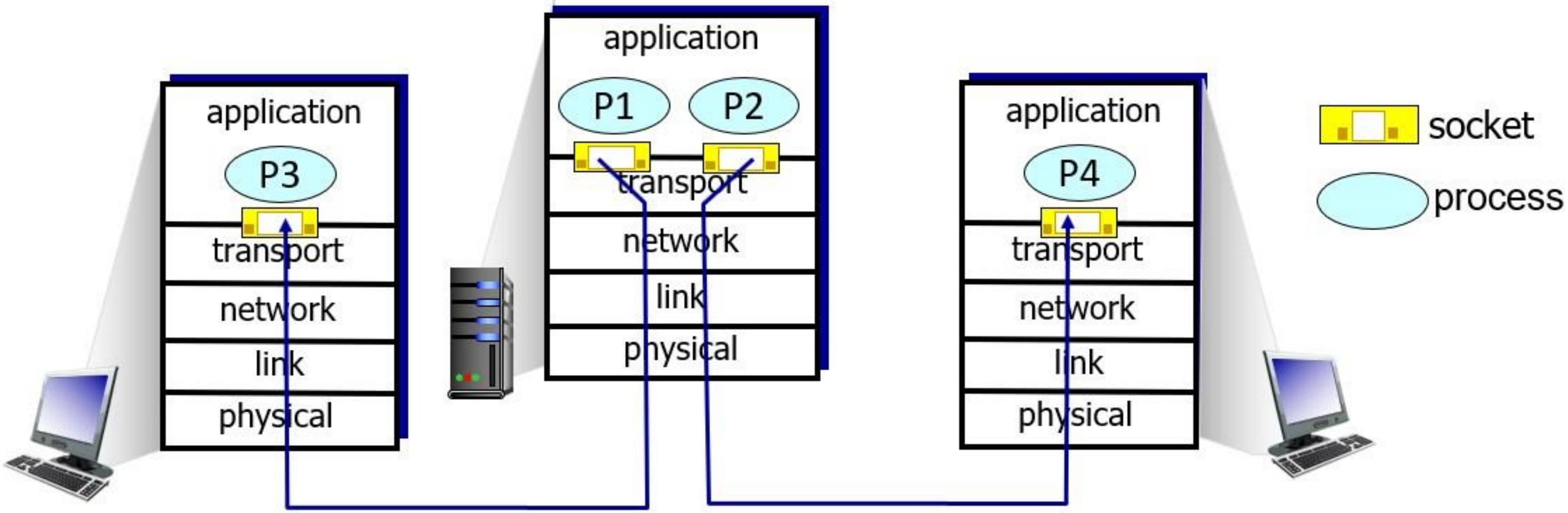


Transport layer: roadmap

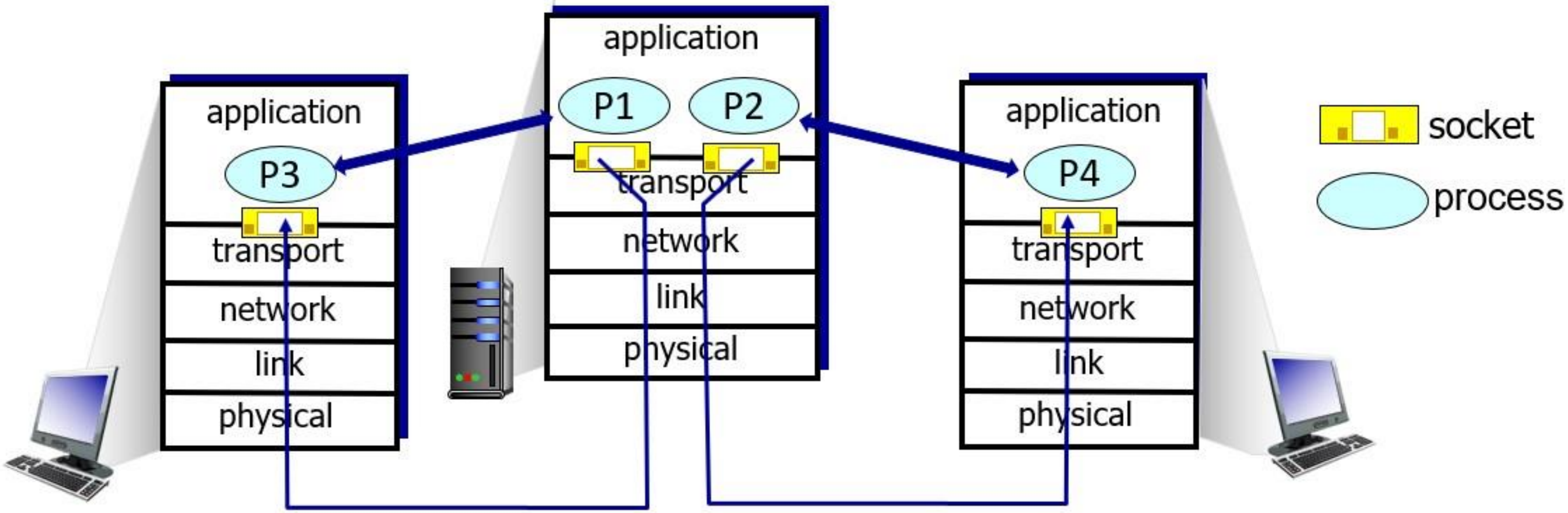
- Transport-layer services
- **Multiplexing and demultiplexing**
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality



Multiplexing/demultiplexing



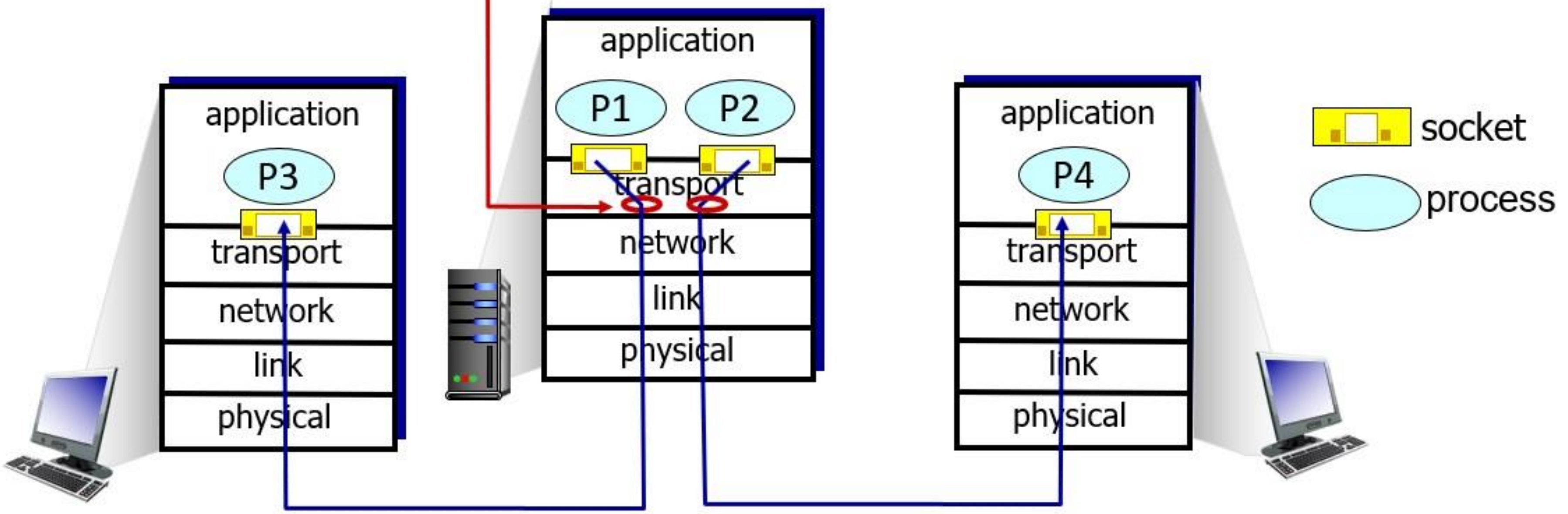
Multiplexing/demultiplexing



Multiplexing/demultiplexing

multiplexing as sender:

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



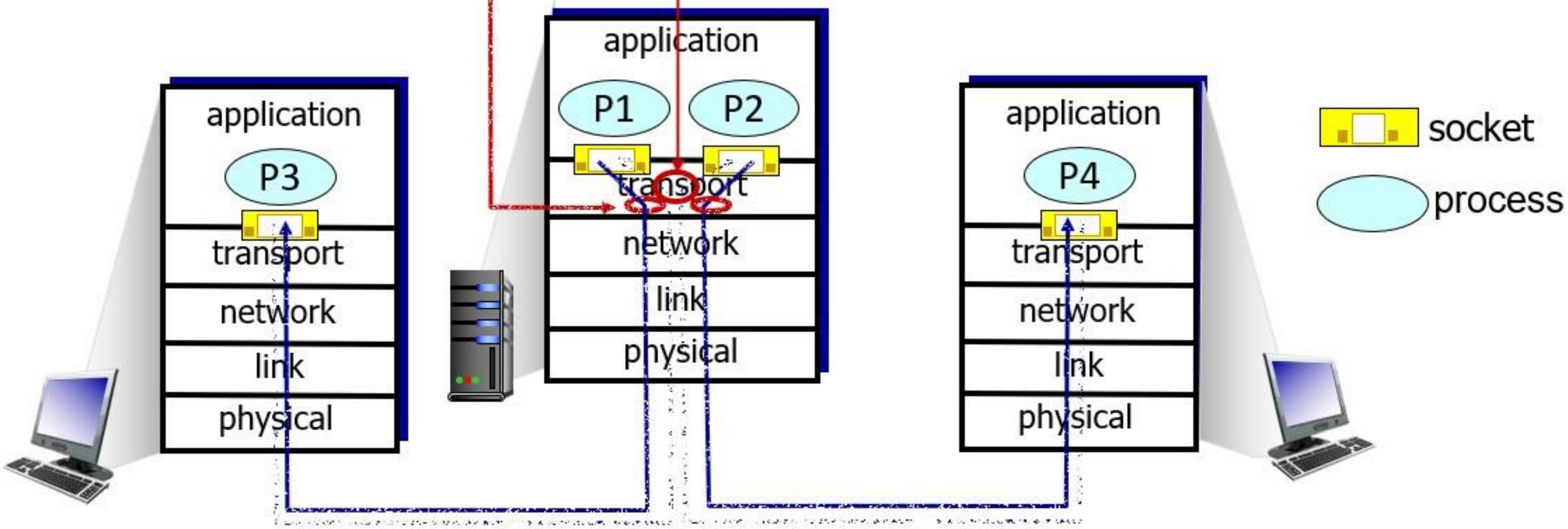
Multiplexing/demultiplexing

multiplexing as sender:

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

demultiplexing as receiver:

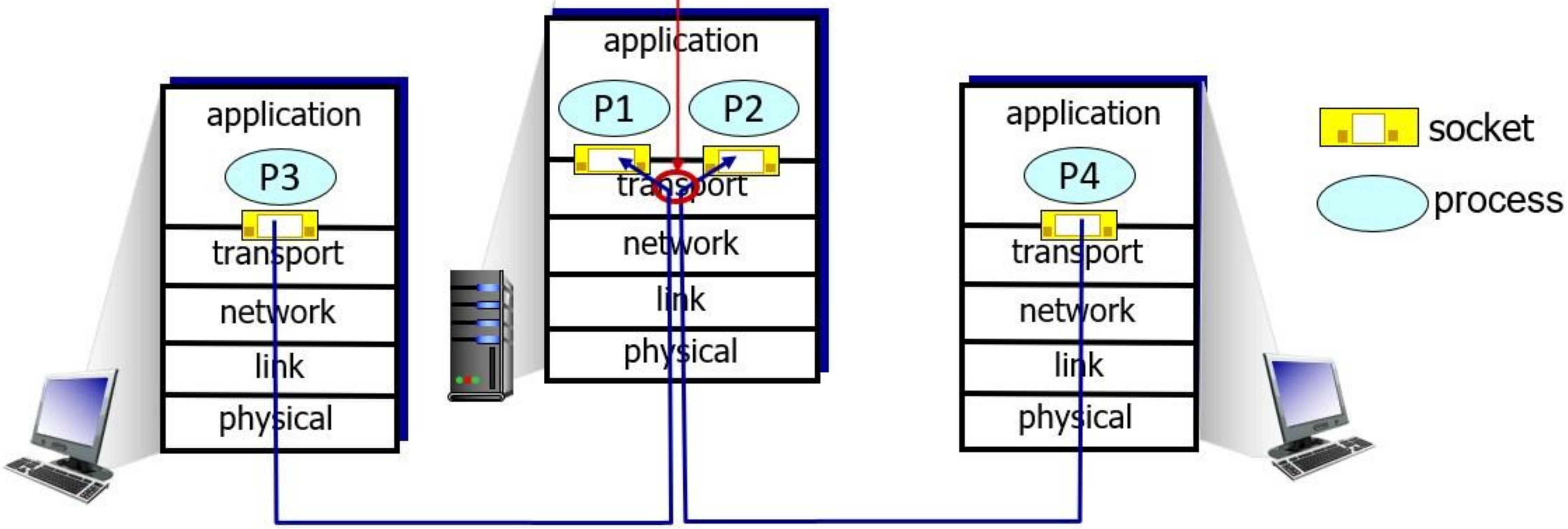
use header info to deliver received segments to correct socket



Multiplexing/demultiplexing

demultiplexing as receiver:

use header info to deliver received segments to correct socket



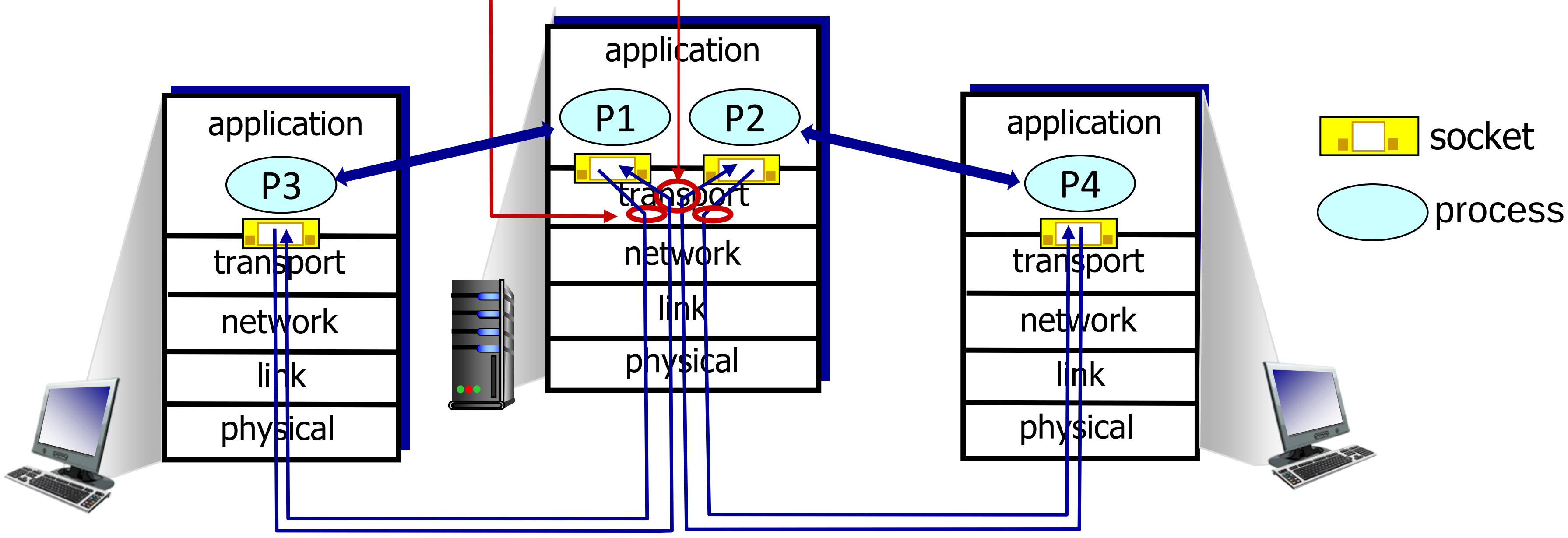
Multiplexing/demultiplexing

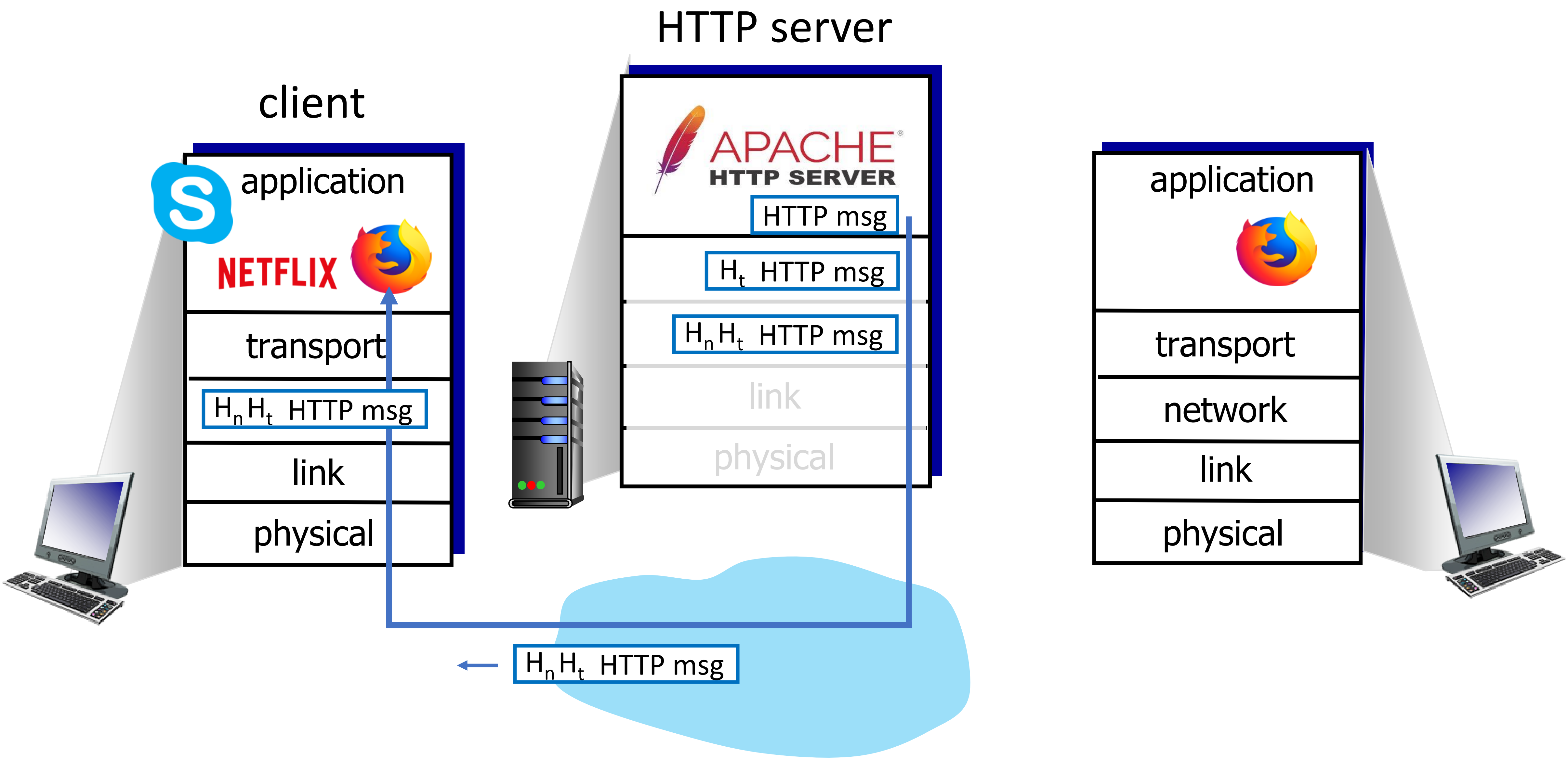
multiplexing as sender:

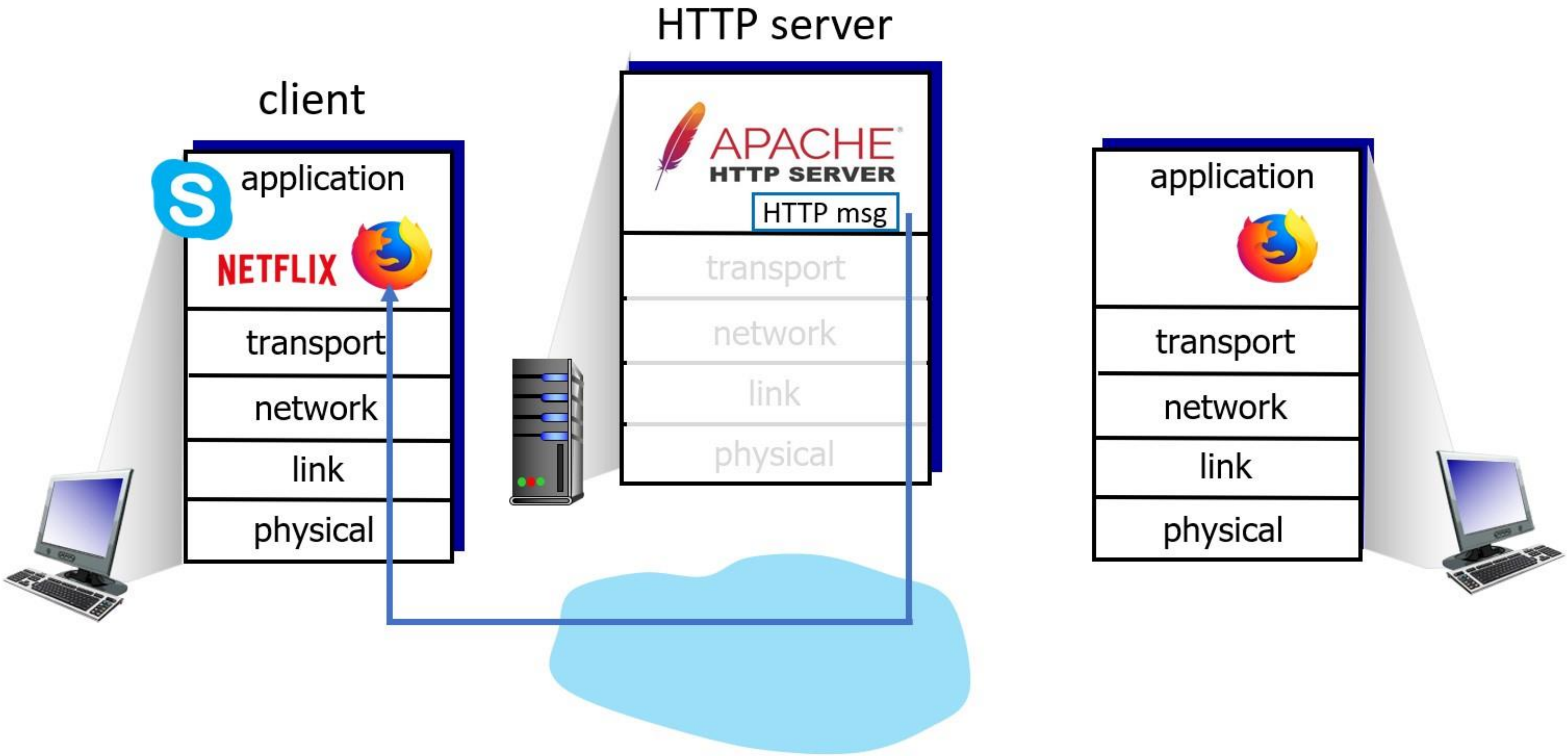
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing as receiver:

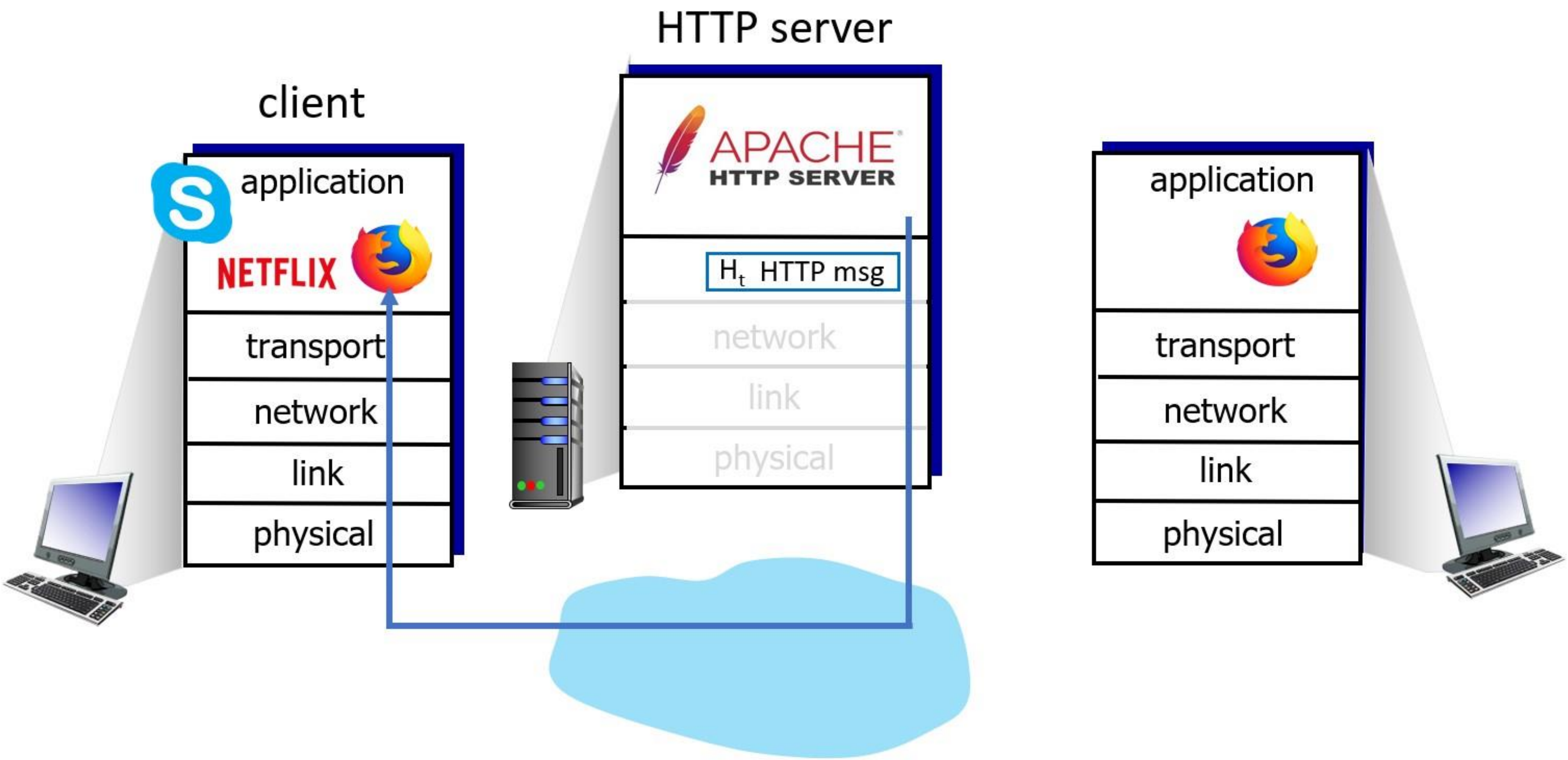
use header info to deliver received segments to correct socket

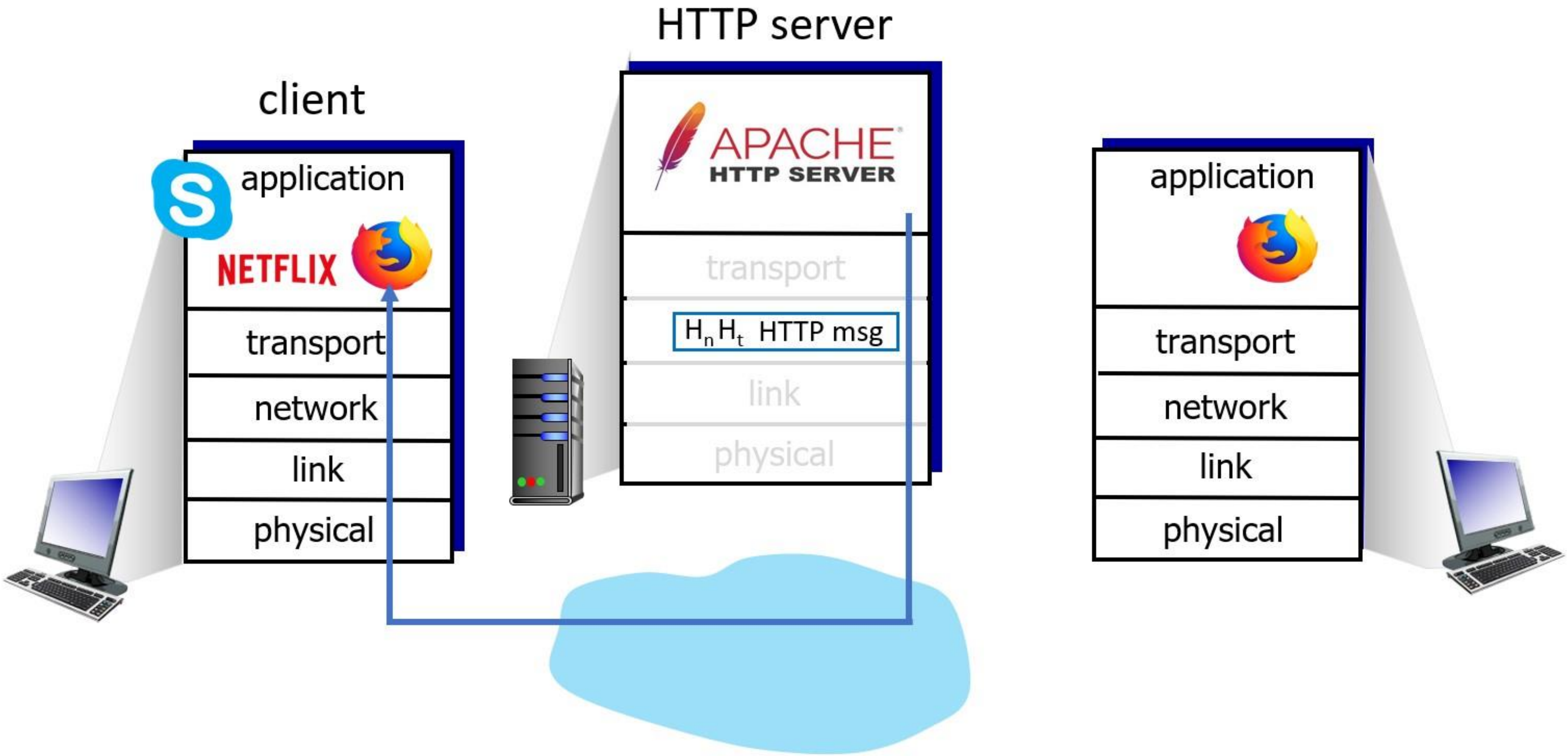



HTTP server
client
application
NETFLIX
transport
Hn Ht HTTP msg
link
physical
APACHE
HTTP SERVER
HTTP msg
Ht HTTP msg
Hn Ht HTTP msg
link
physical
Hn Ht HTTP msg
application
transport
network
link
physical

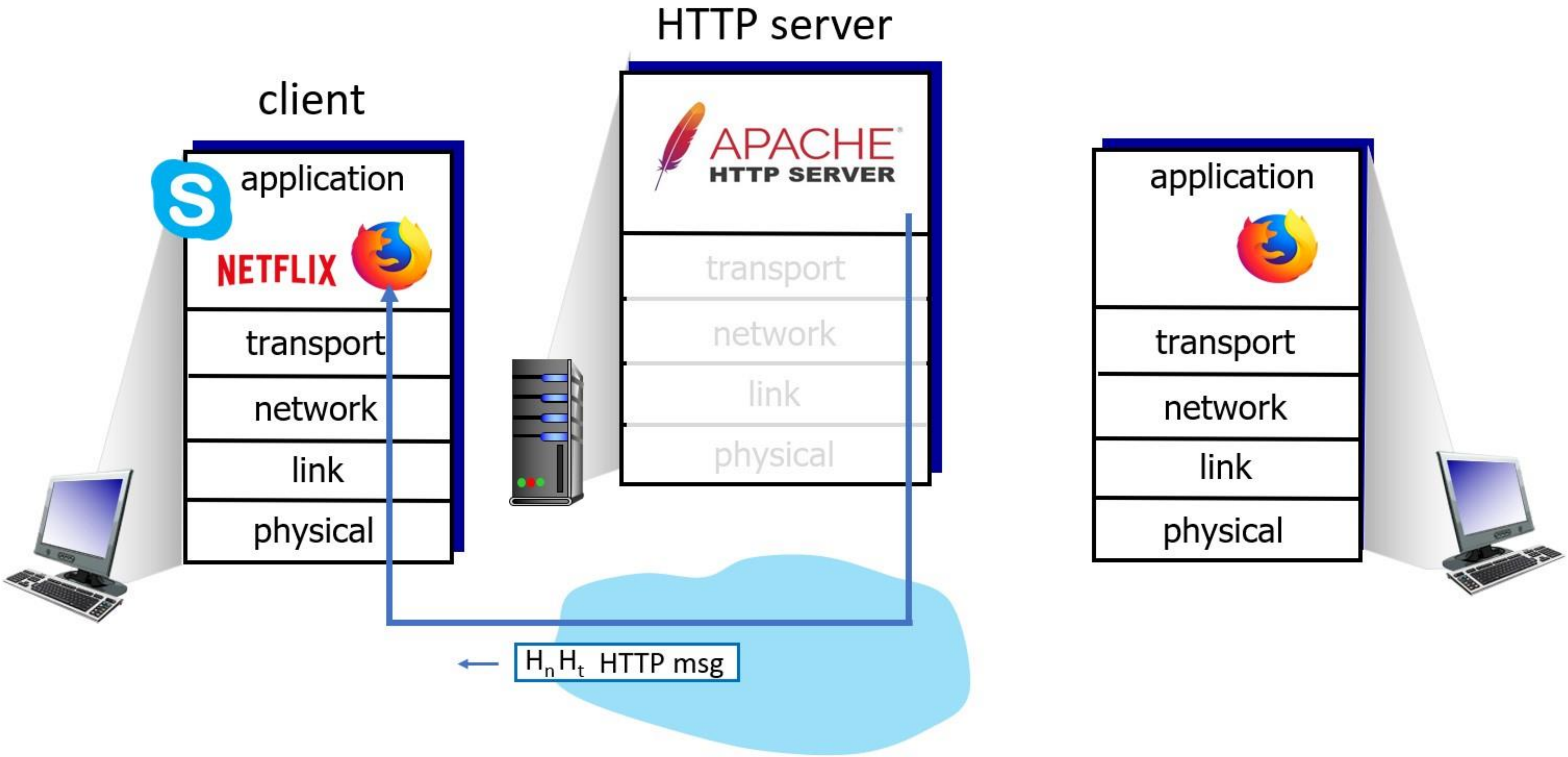


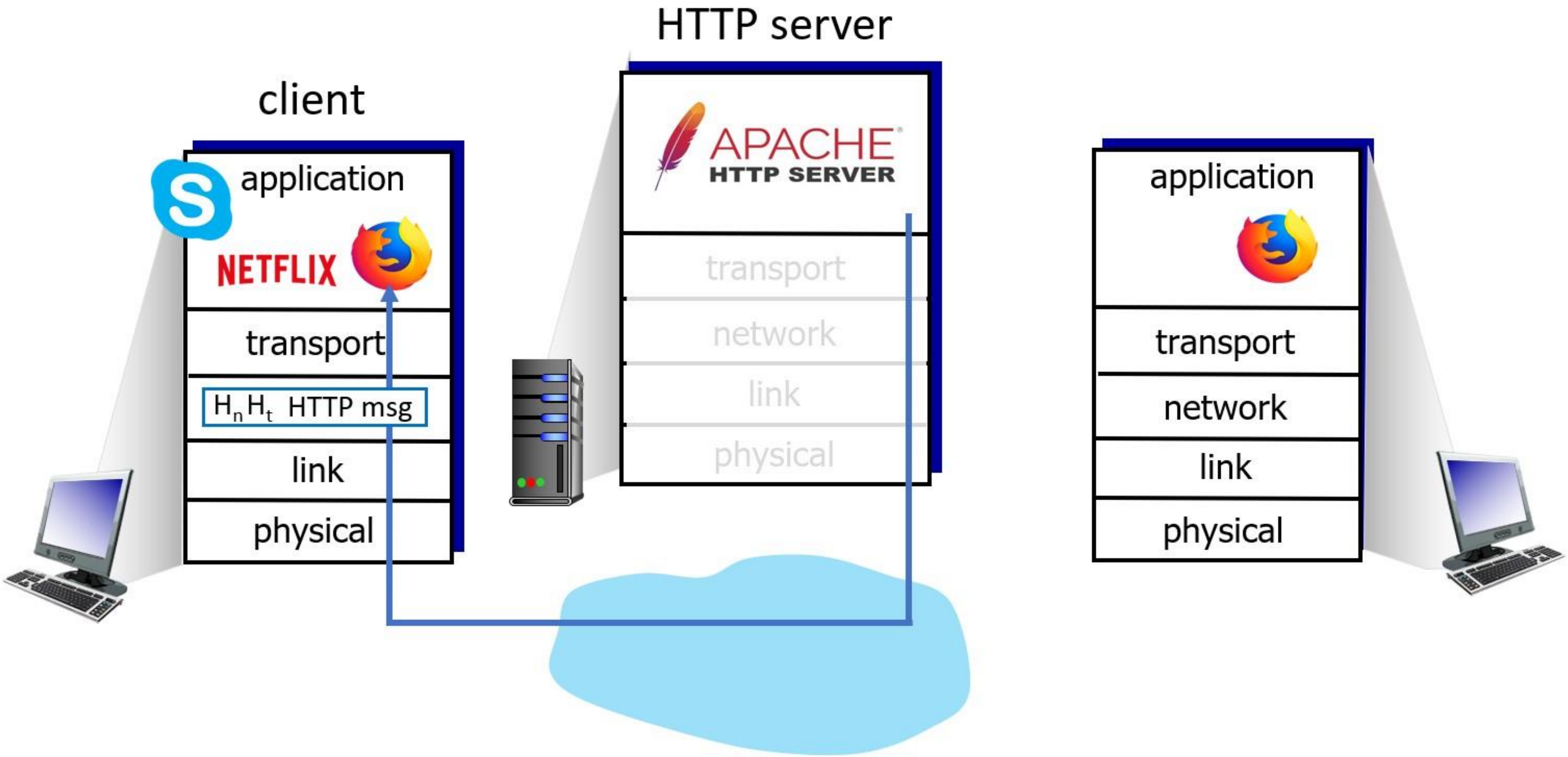
HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
transport
network
link
physical
application
transport
network
link
physical


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

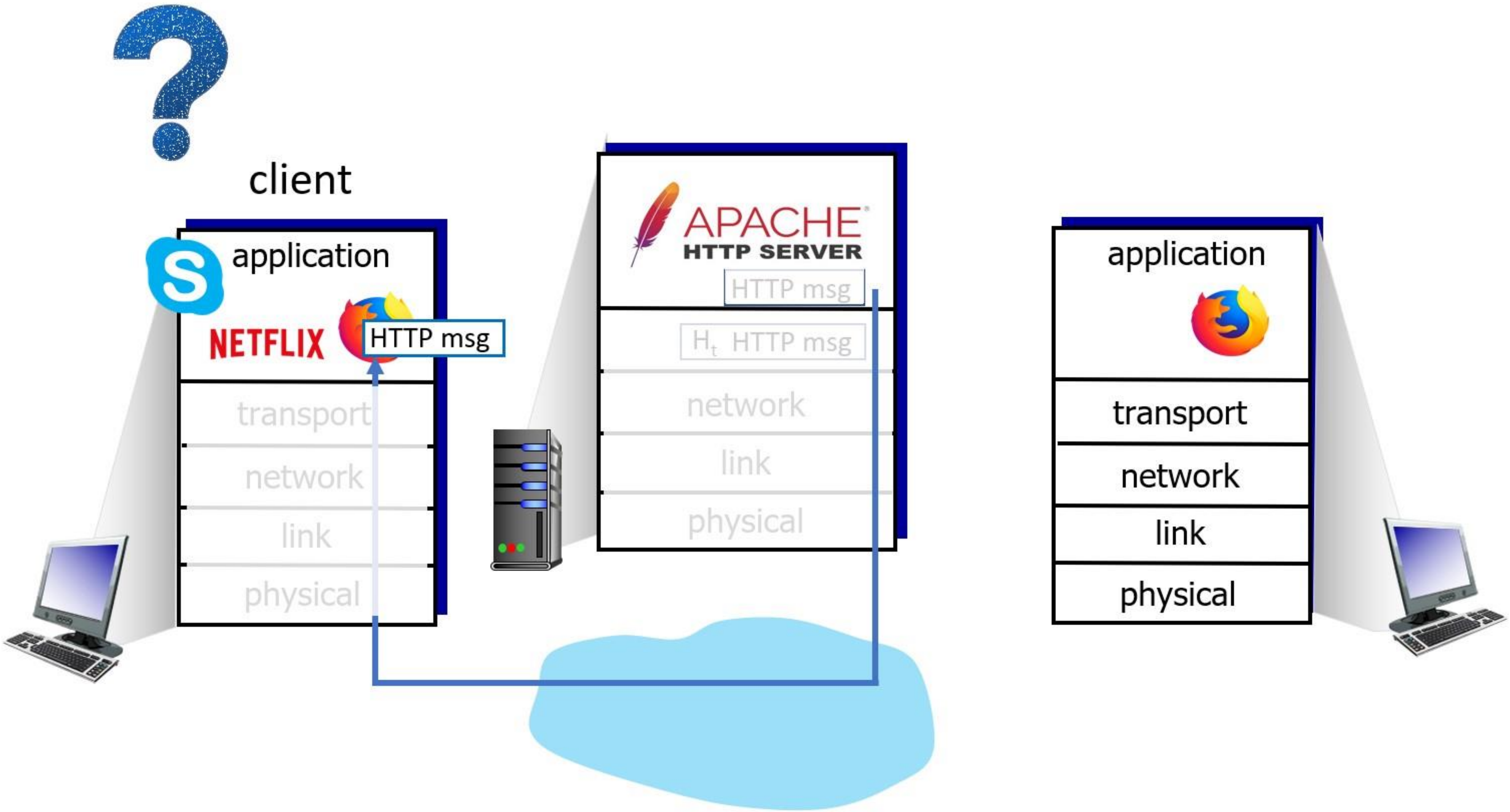


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
transport
$H_n H_t$ HTTP msg
link
physical
application
transport
network
link
physical


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

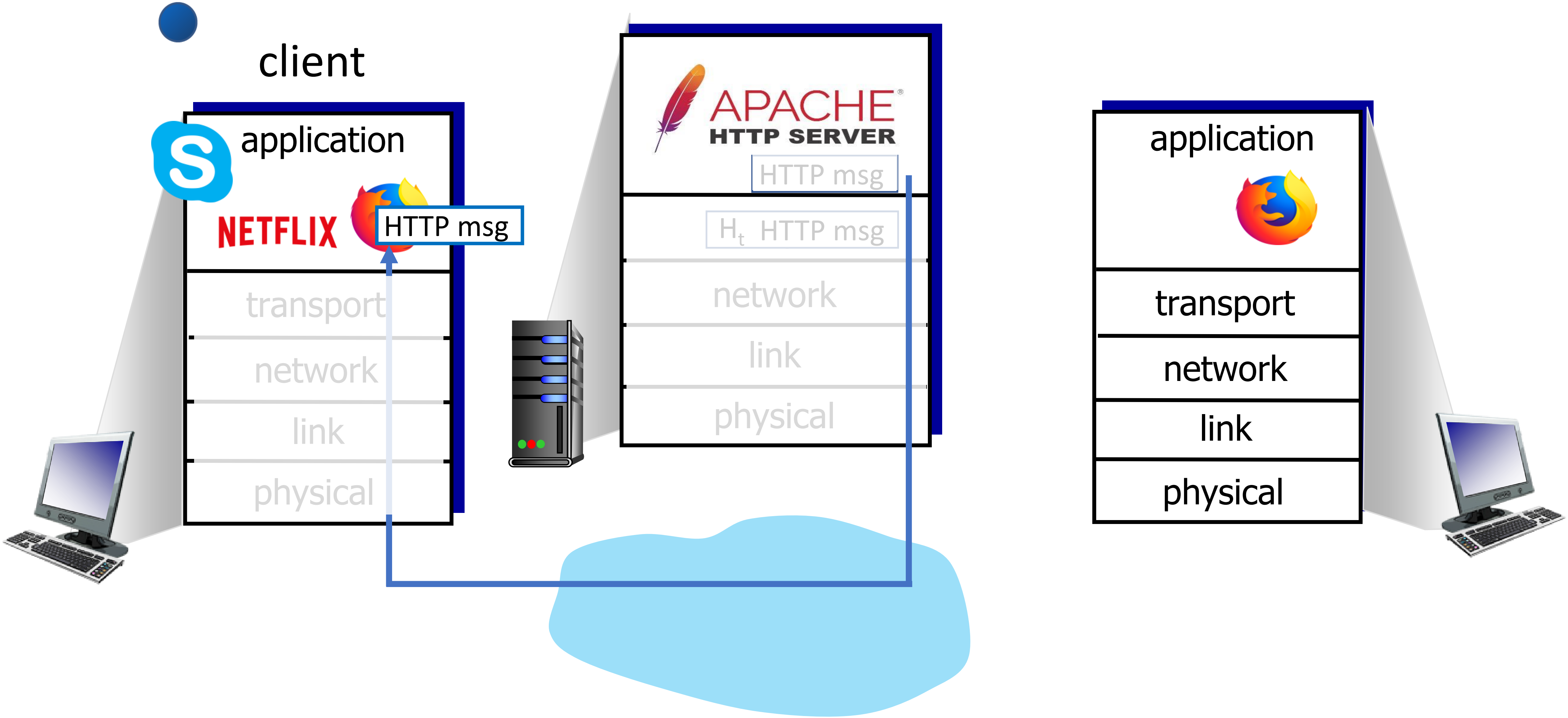


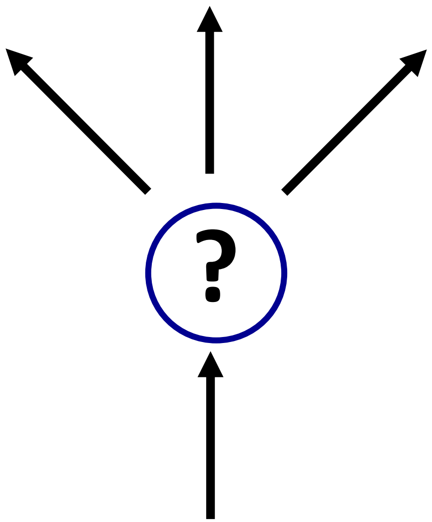
HTTP server
client
application
NETFLIX
transport
H_n H_t HTTP msg
link
physical
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical



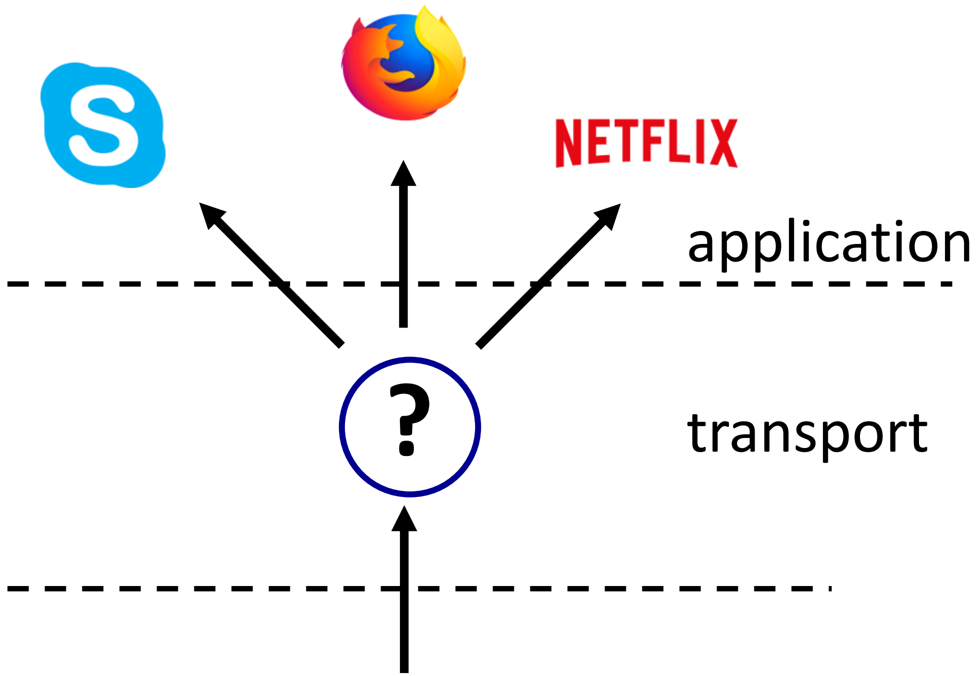
client
application
NETFLIX
HTTP msg
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
network
link
physical
application
transport
network
link
physical

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

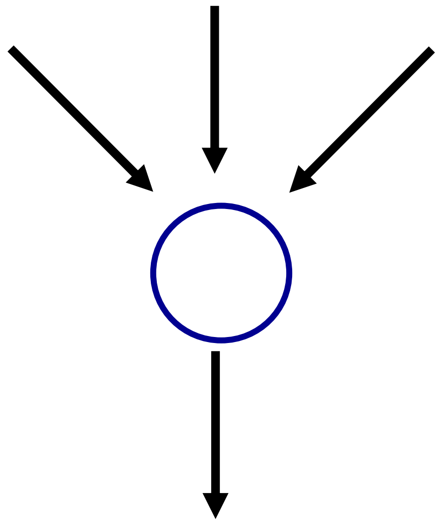




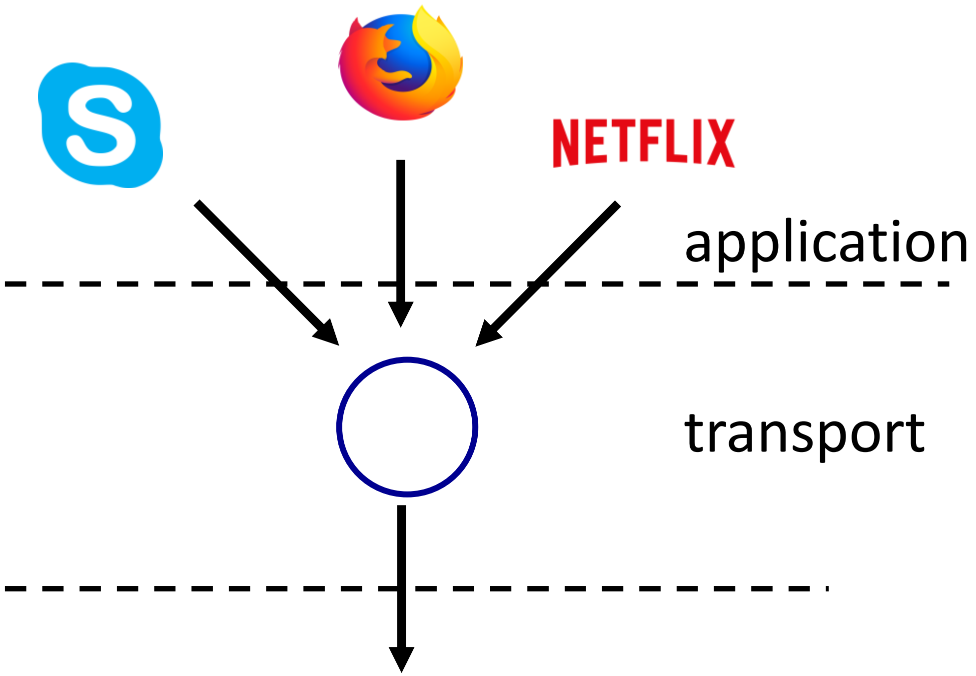
de-multiplexing



de-multiplexing



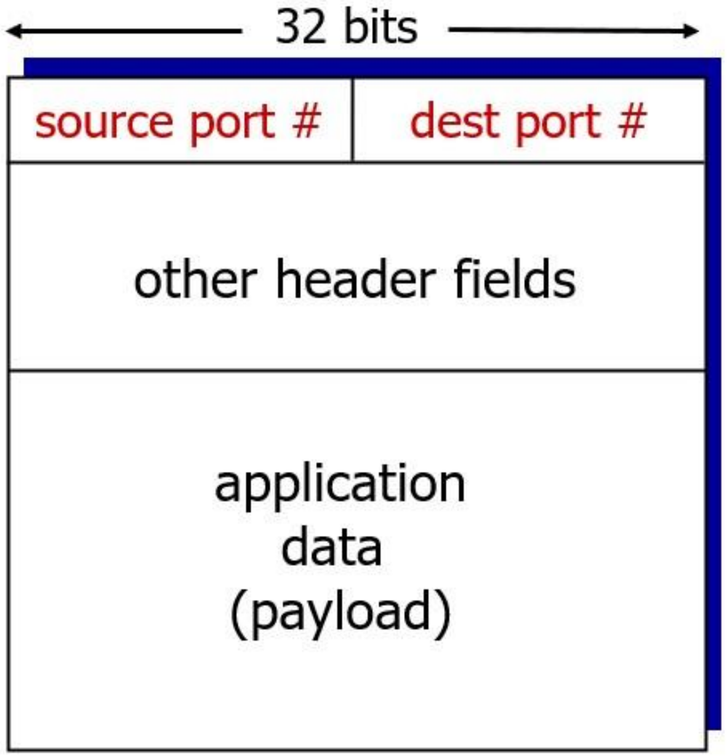
multiplexing



multiplexing

How demultiplexing works

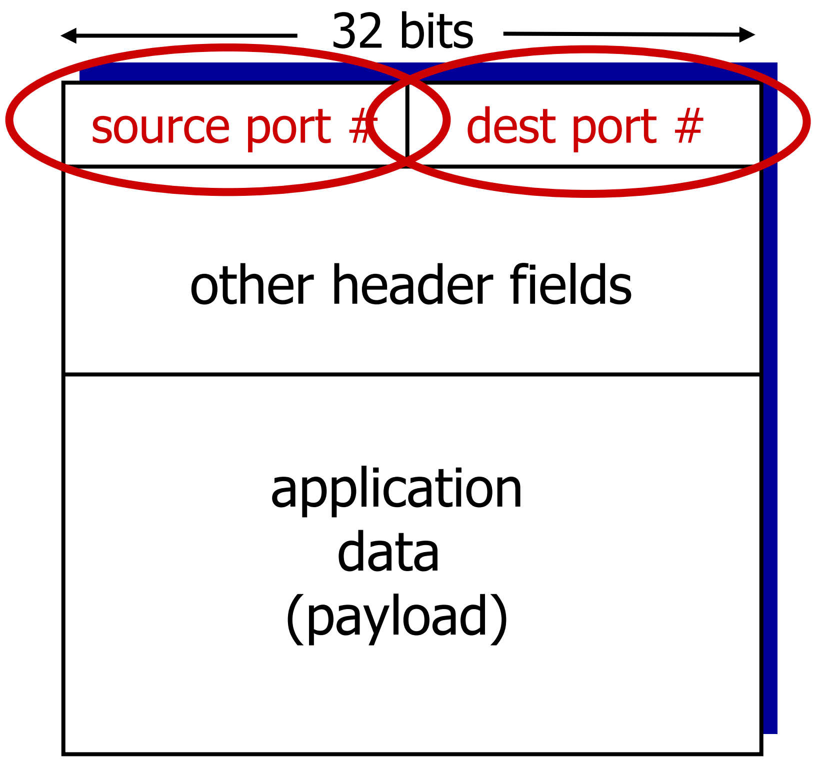
- host <u>receives</u> IP datagrams
  - **each datagram** has **source IP** address, **destination IP** address
  - each datagram carries one transport-layer segment
  - each segment has **source, destination port number**
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

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

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

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

Connectionless demultiplexing

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

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

Connectionless demultiplexing

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

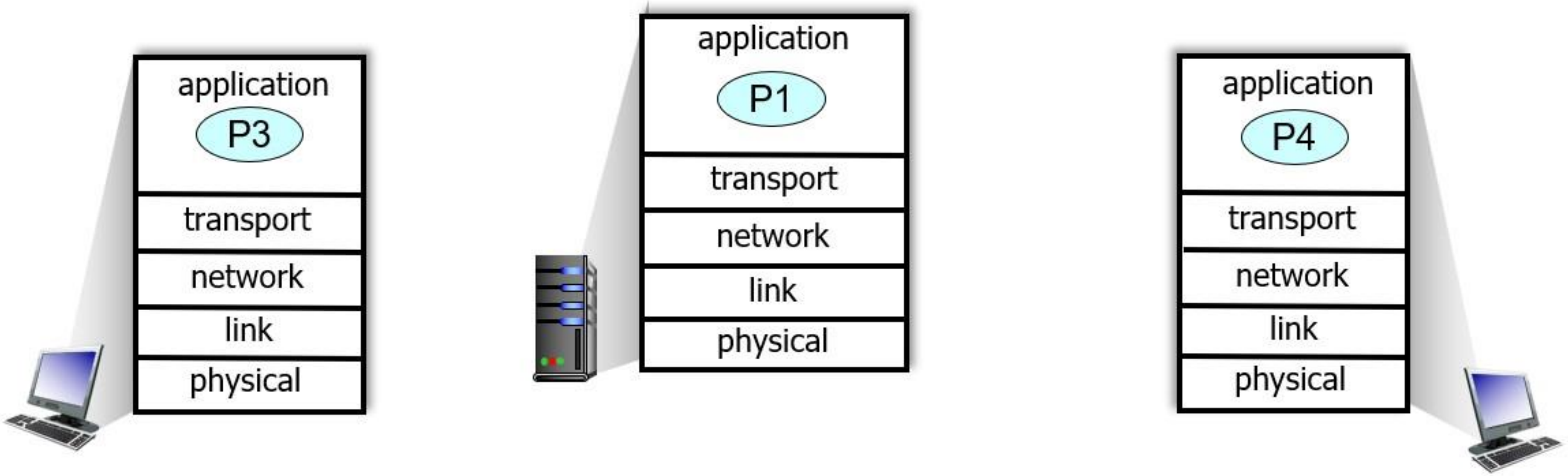
```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

when receiving host receives *UDP* segment:

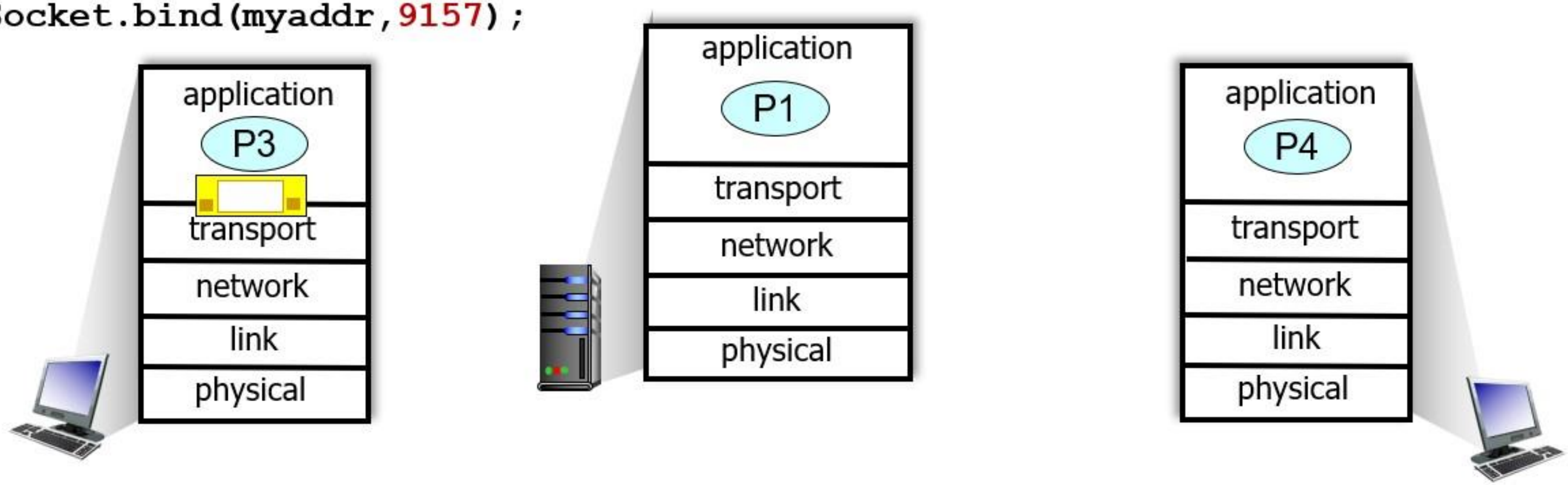
- checks destination port # in segment
- directs UDP segment to socket with that port #

Connectionless demultiplexing: an example



Connectionless demultiplexing: an example

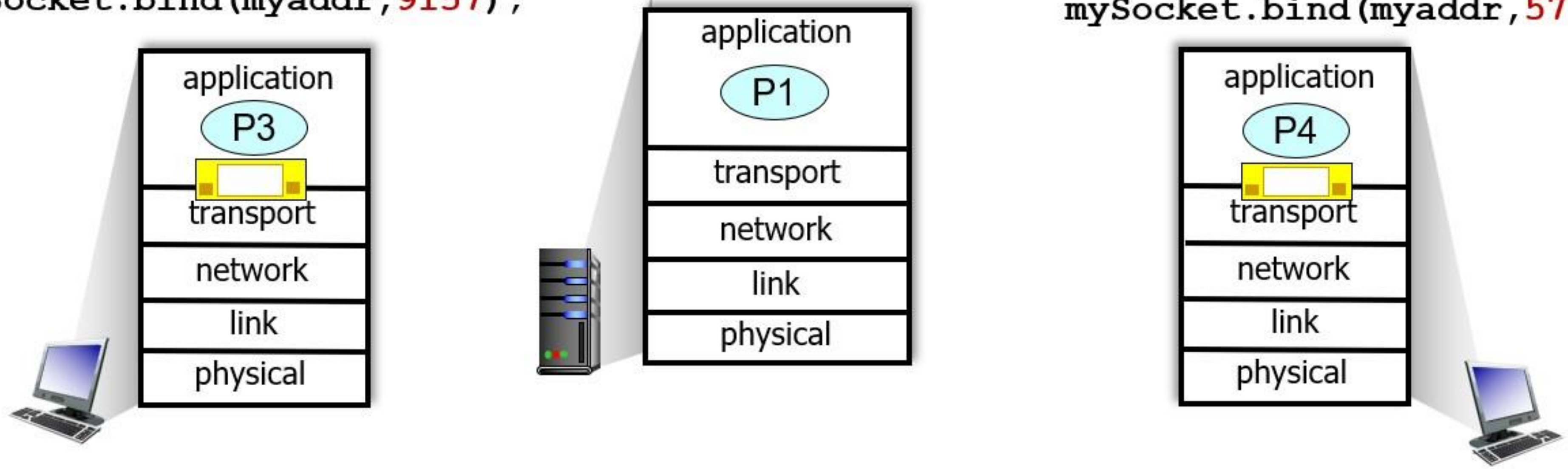
```
mySocket =
 socket(AF_INET,SOCK_STREAM)
mySocket.bind(myaddr,9157);
```



Connectionless demultiplexing: an example

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

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

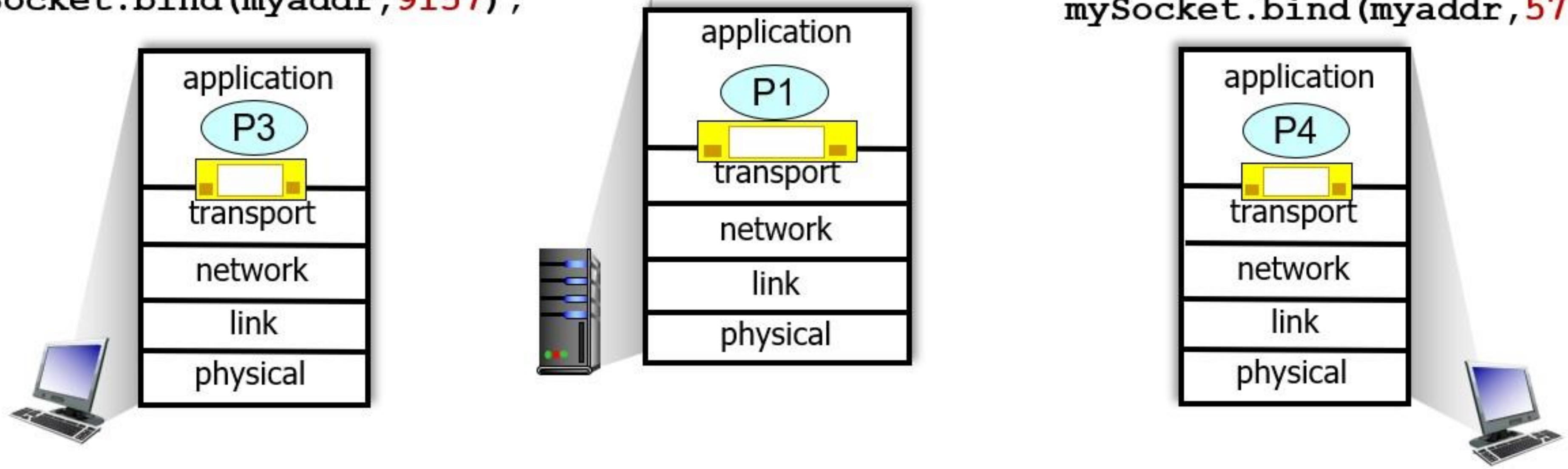


Connectionless demultiplexing: an example

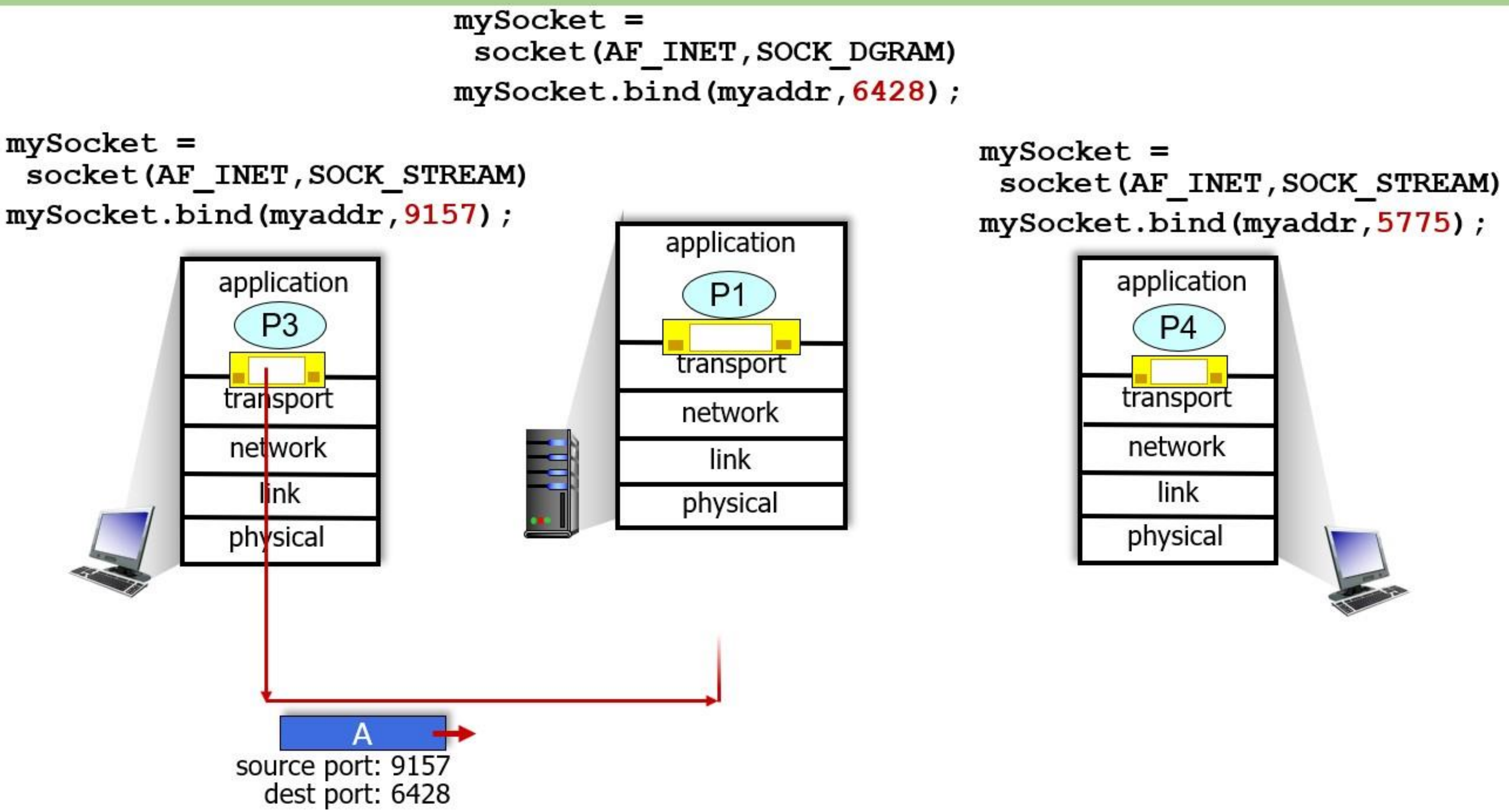
```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,6428);
```

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

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



Connectionless demultiplexing: an example

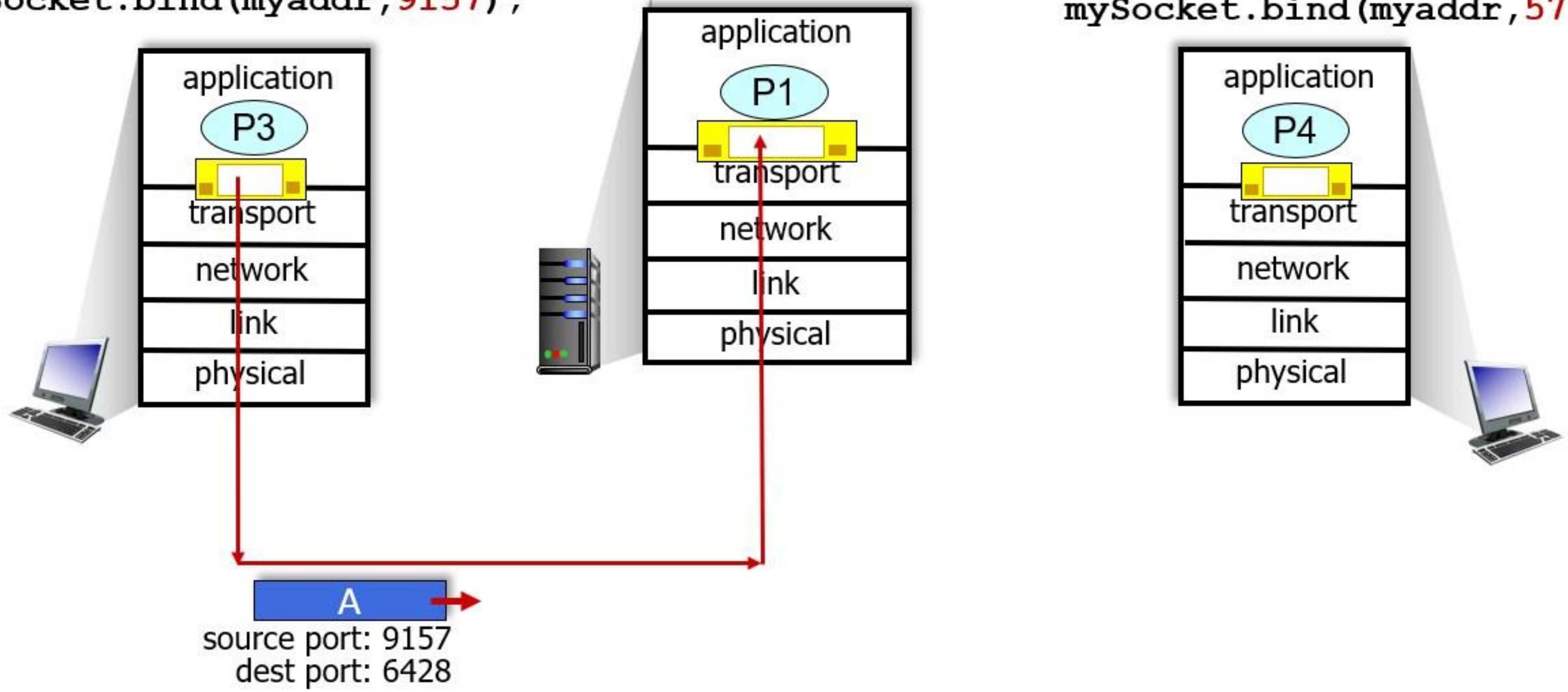


Connectionless demultiplexing: an example

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

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

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

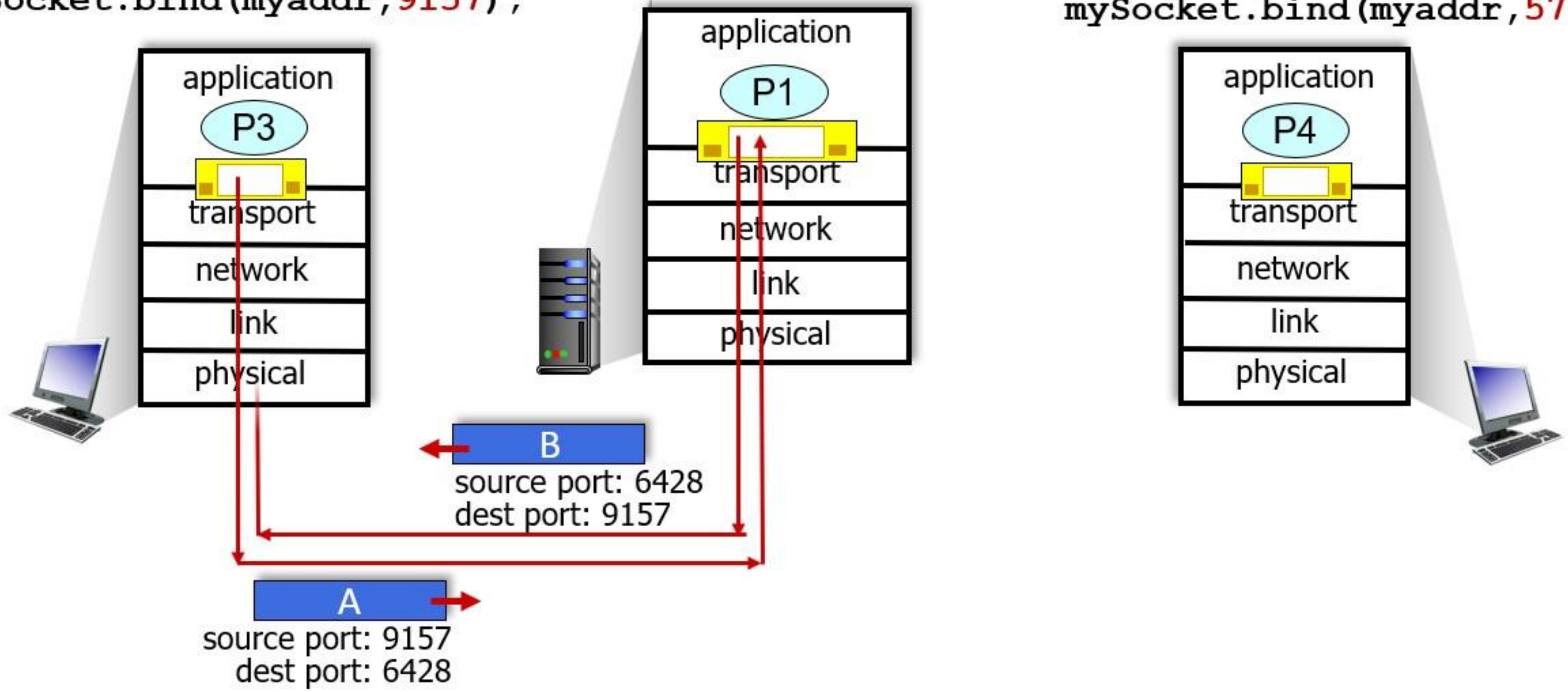


Connectionless demultiplexing: an example

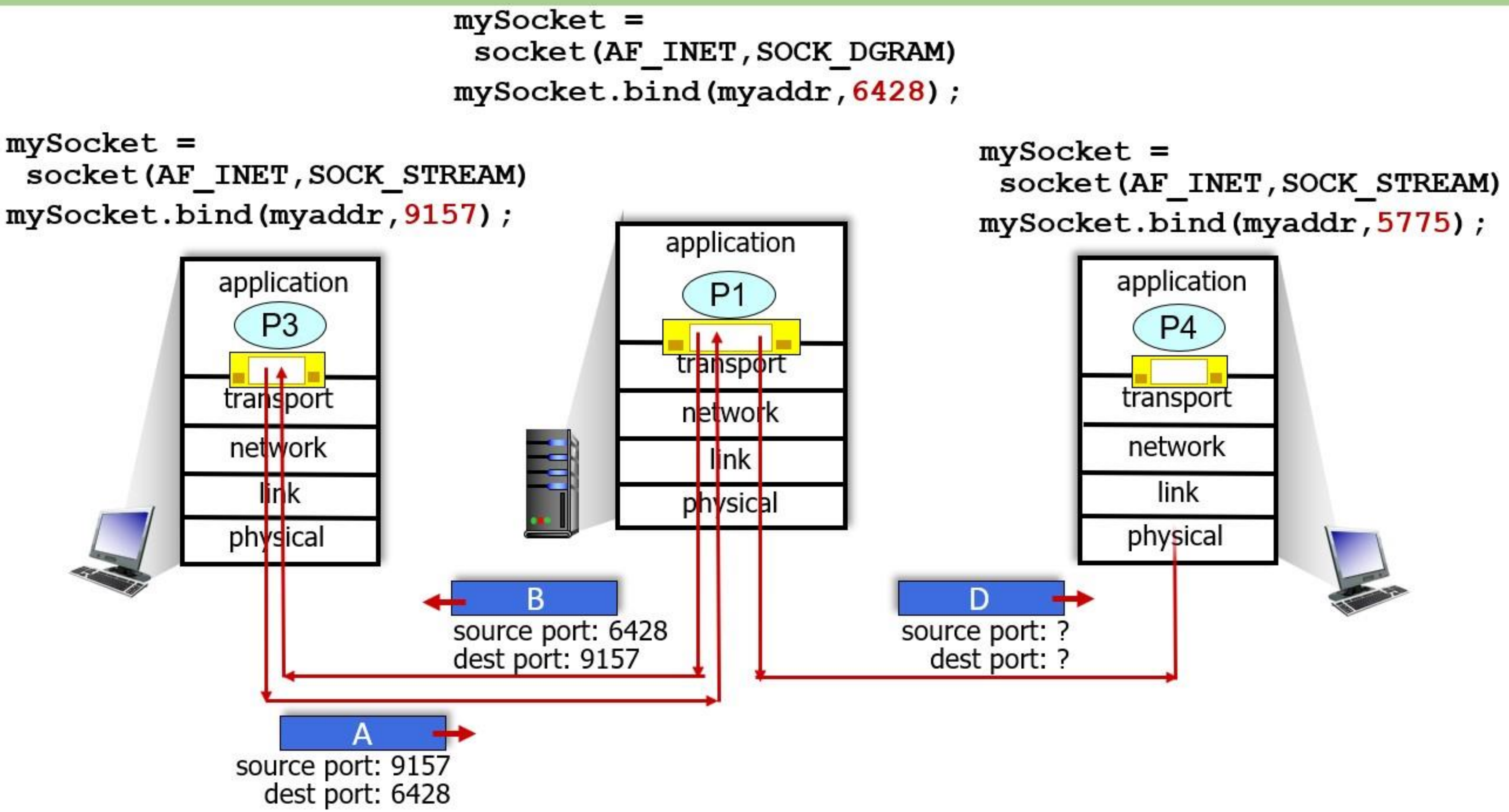
```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr,6428);
```

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

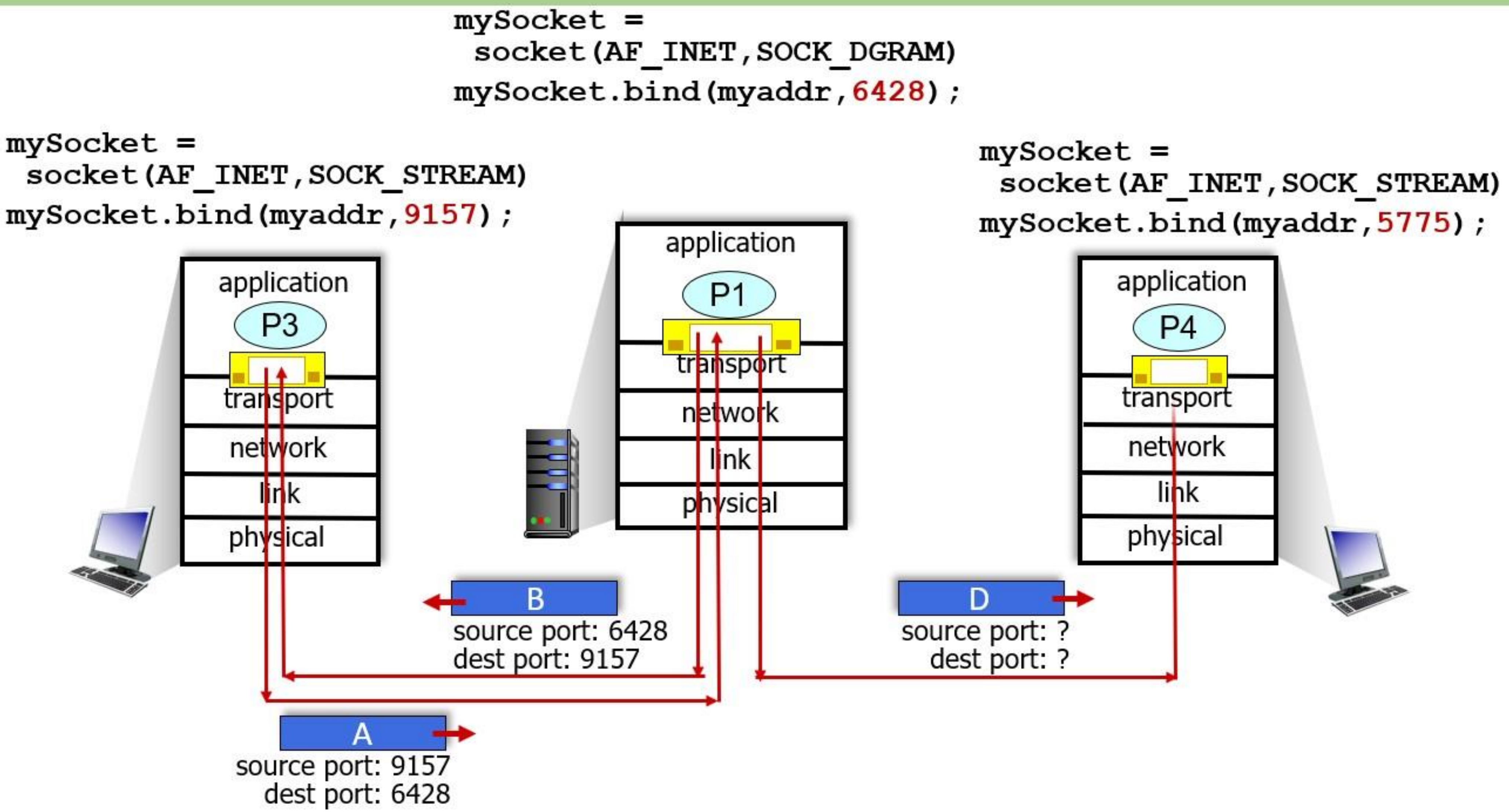
```
mySocket =
 socket(AF_INET,SOCK_STREAM)
mySocket.bind(myaddr,5775);
```



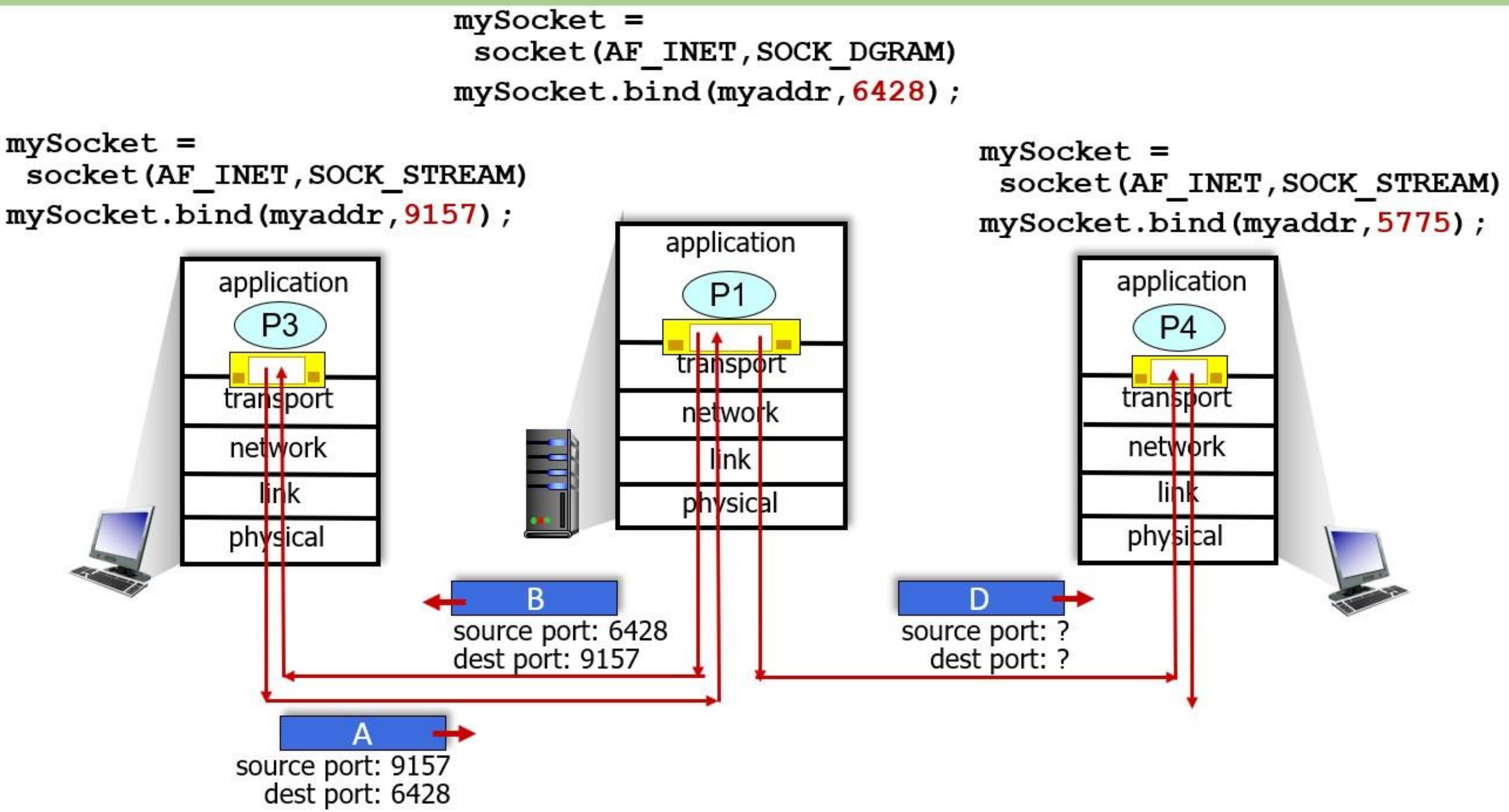
Connectionless demultiplexing: an example



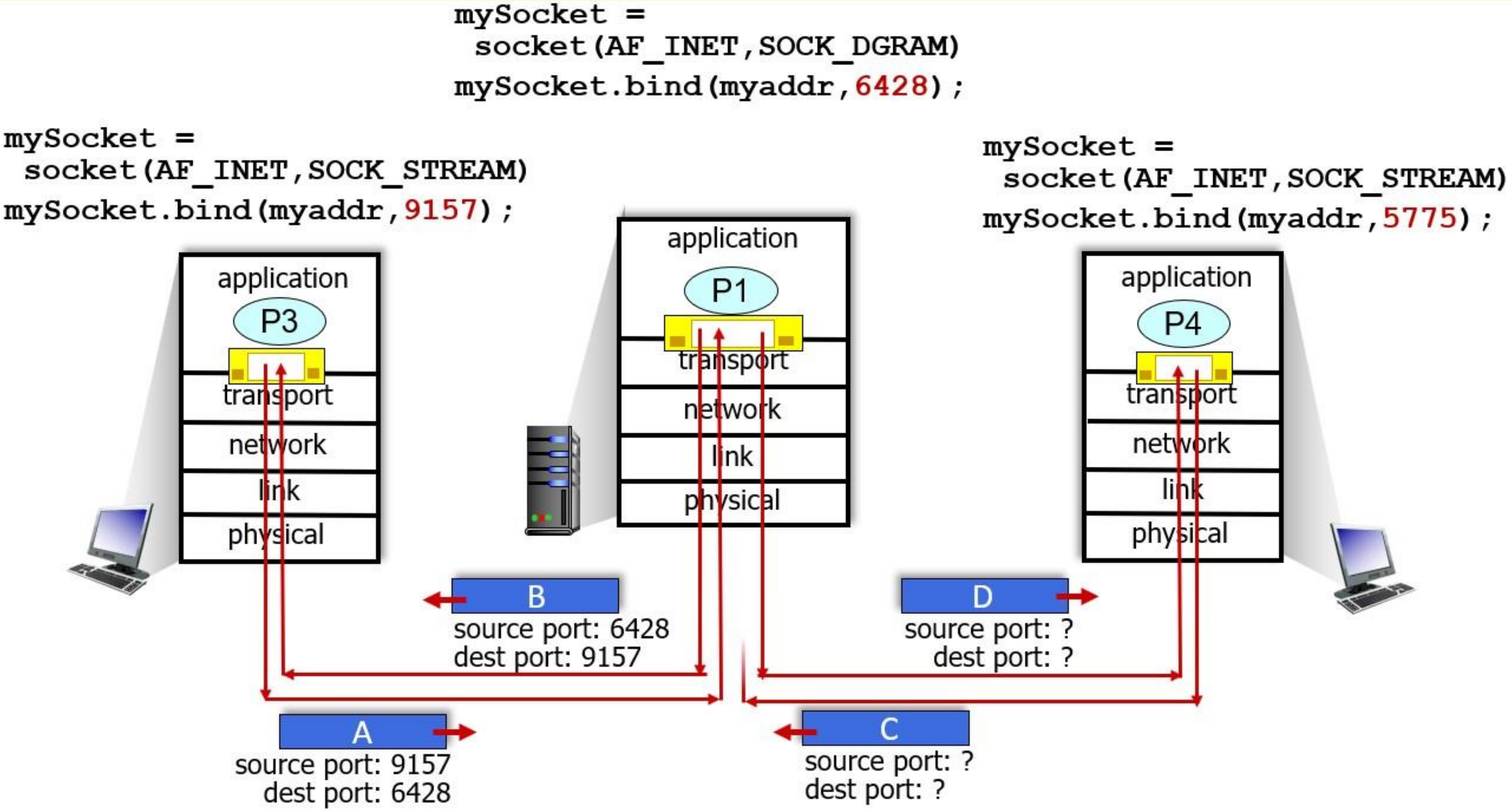
Connectionless demultiplexing: an example



Connectionless demultiplexing: an example



Connectionless demultiplexing: an example

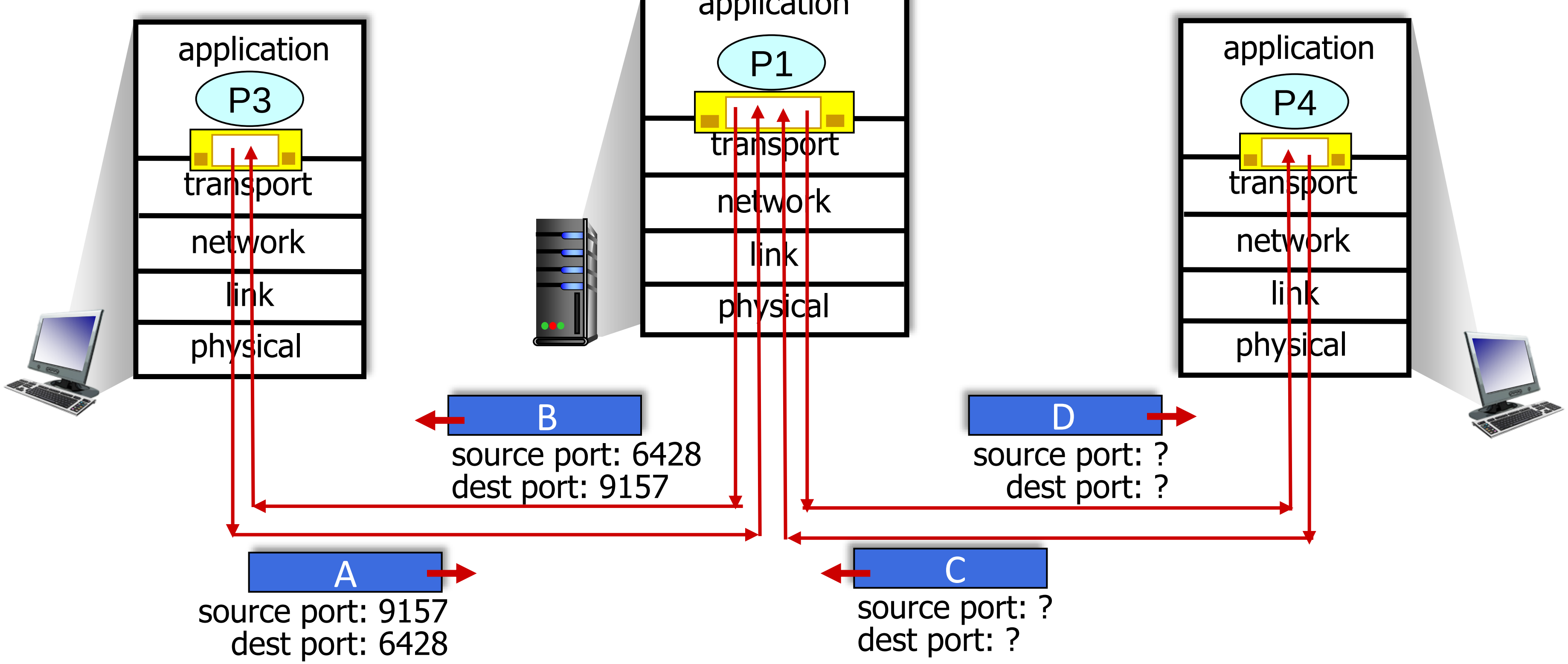


Connectionless demultiplexing: an example

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

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

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

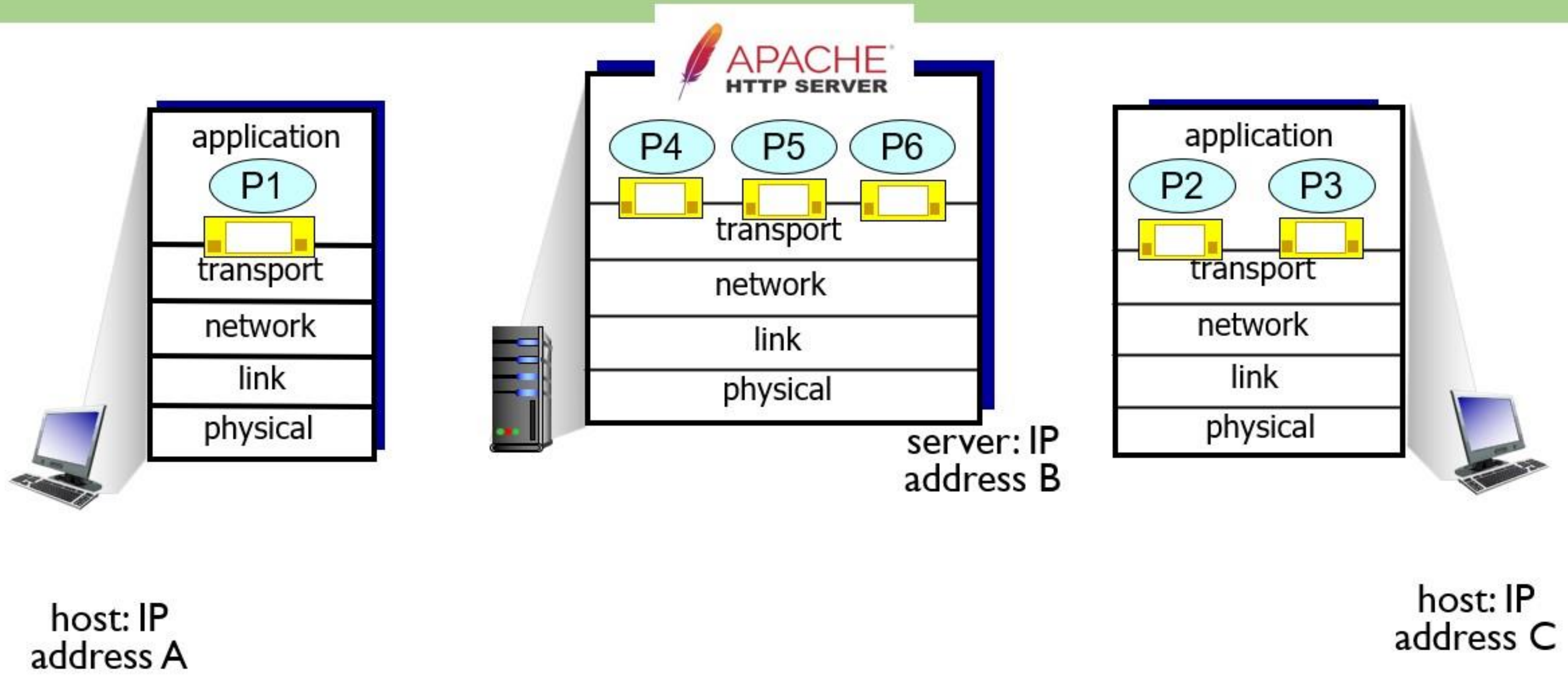


Connection-oriented demultiplexing

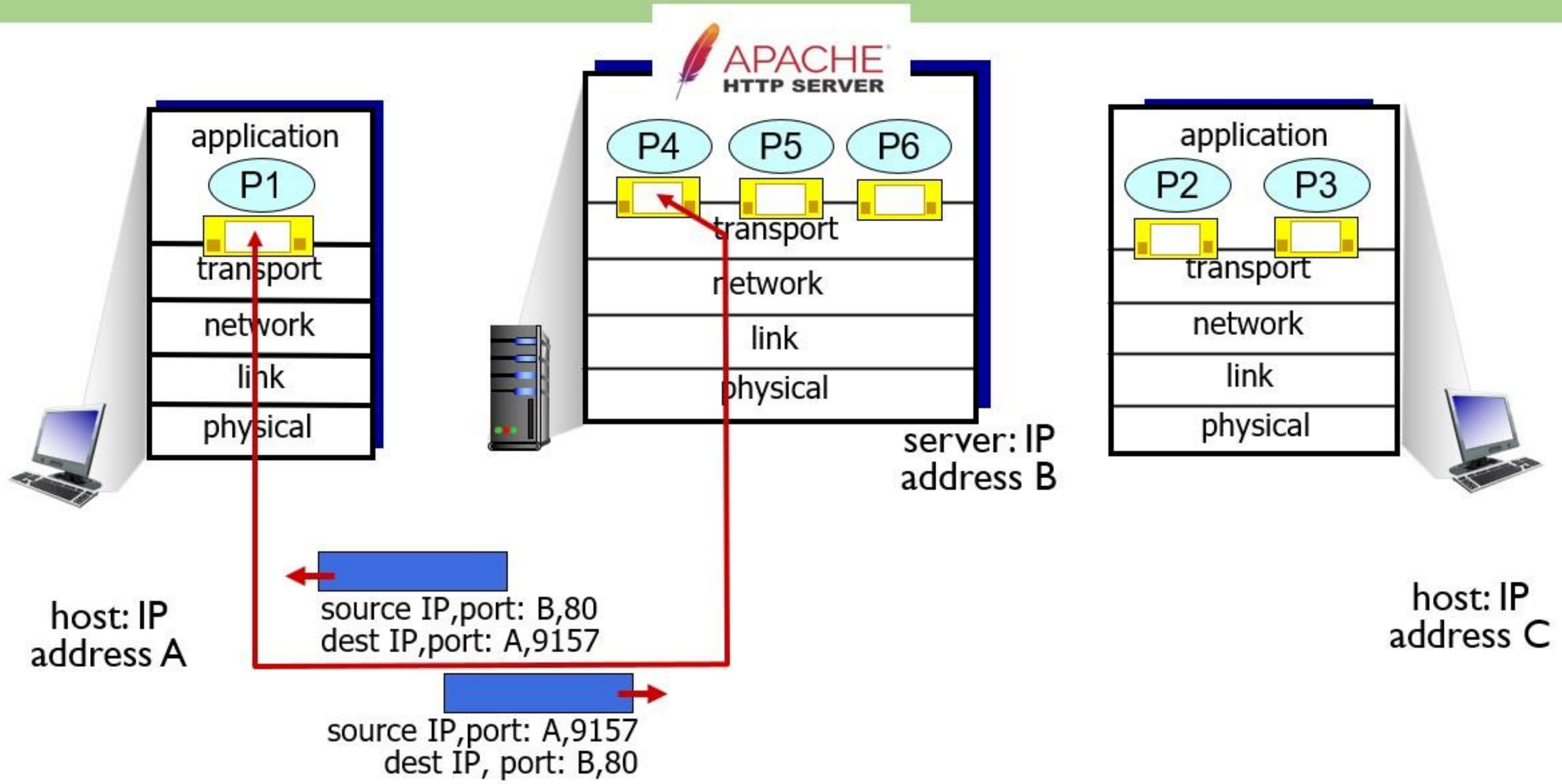
- TCP socket identified by 4-tuple:
  - **source IP address**
  - **source port number**
  - **dest IP address**
  - **dest port number**
- demux: receiver uses *all four values* *(4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

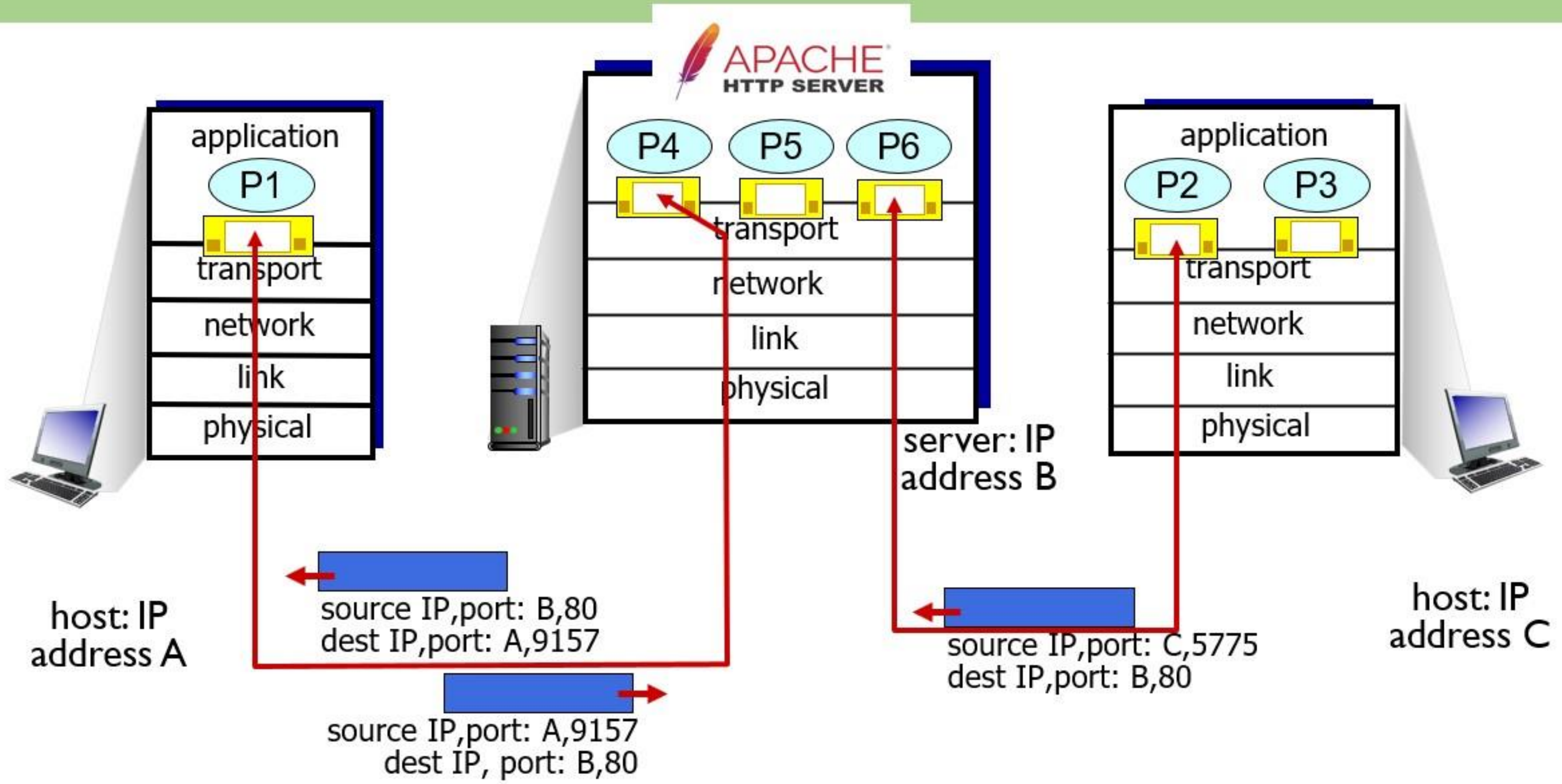
Connection-oriented demultiplexing: example



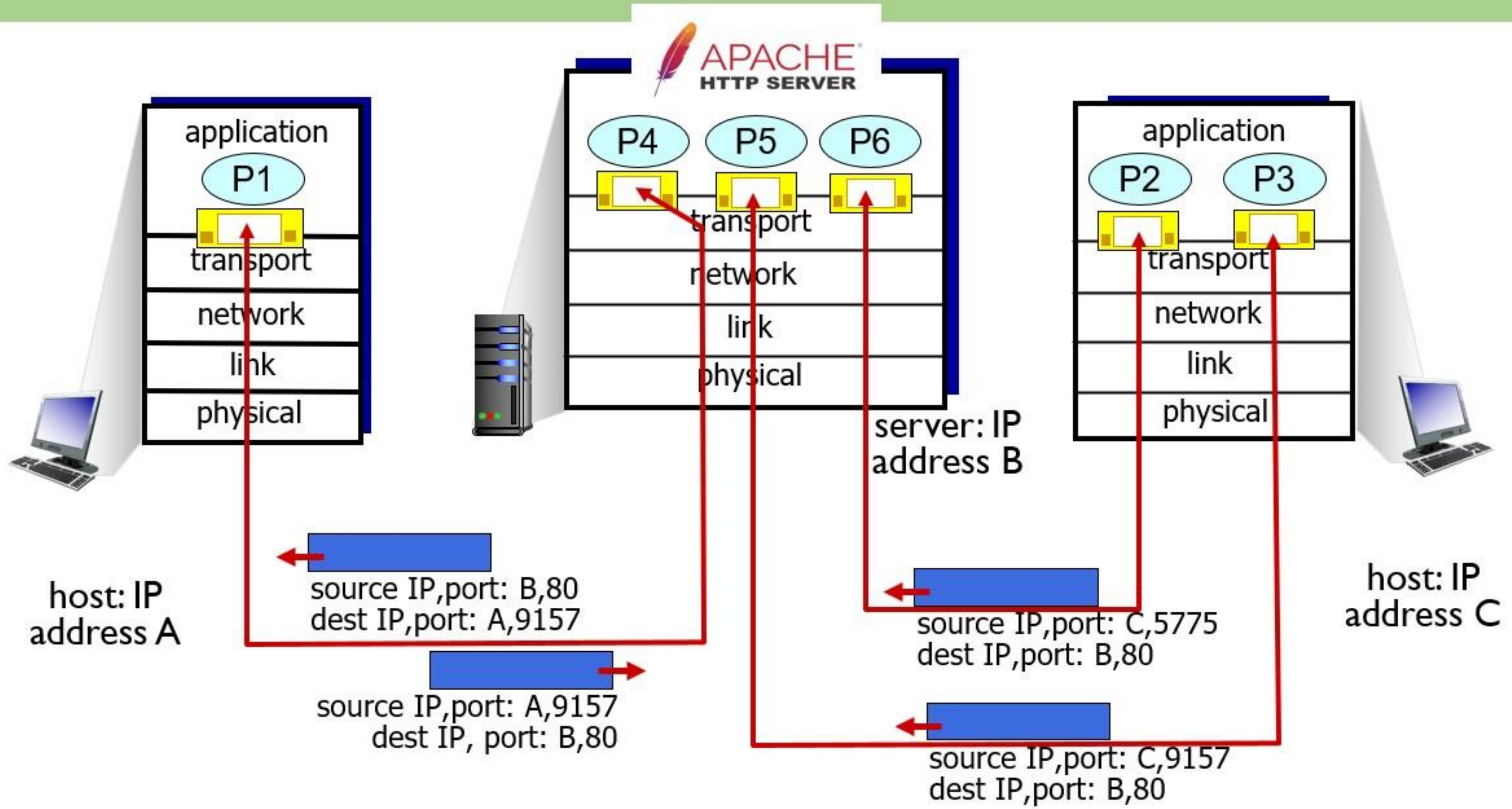
Connection-oriented demultiplexing: example



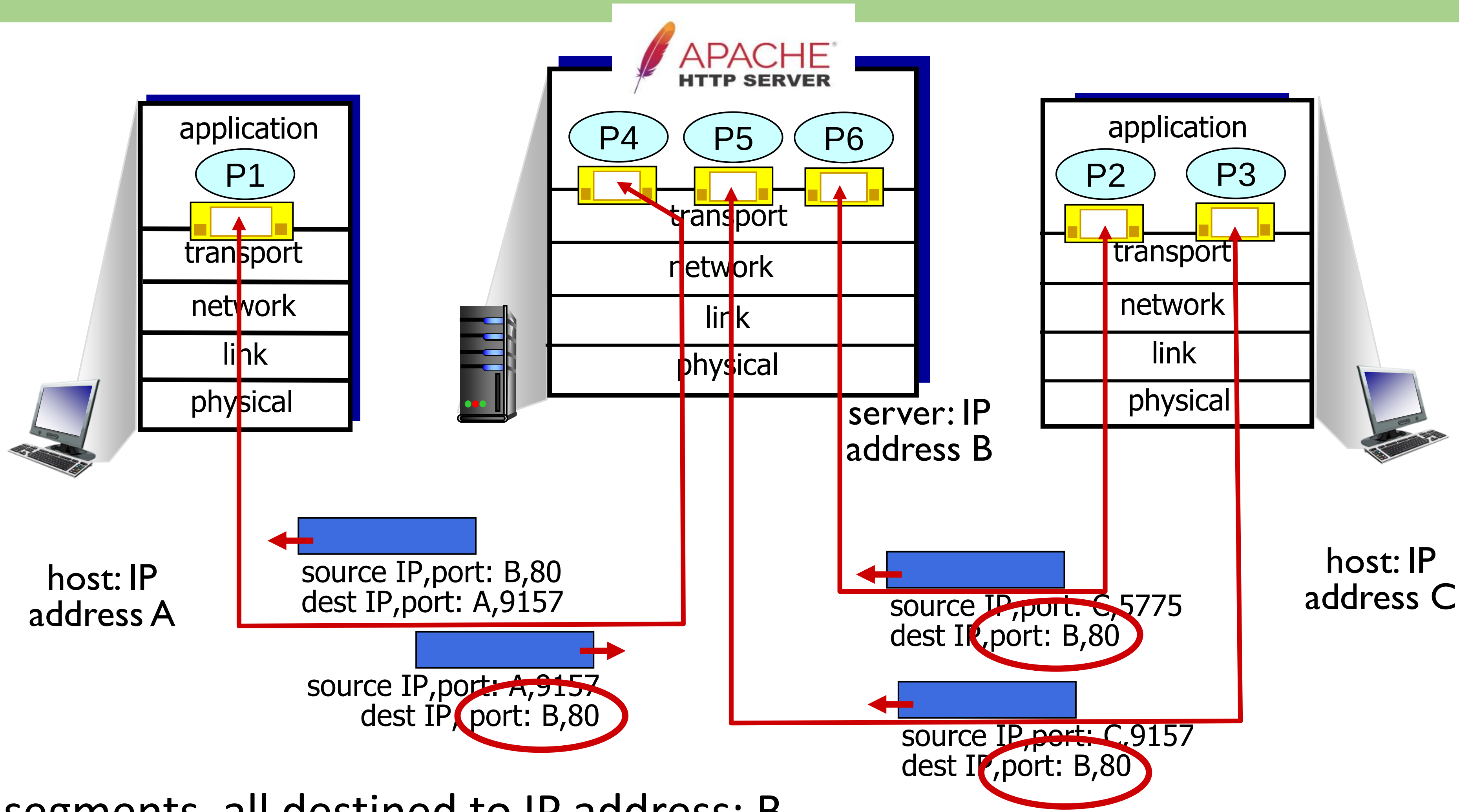
Connection-oriented demultiplexing: example



Connection-oriented demultiplexing: example



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

Acknowledgment

- **These lecture slides are based on:**

1) **Chapter 3 (P 211-224)** from the book "Computer Networking: A Top-Down Approach, Eighth Edition, Global Edition" by (James F. Kurose and Keith W. Ross's).

END OF LECTURE (4) Part A

Keep connected with the classroom

lmzcbsf

THANK YOU FOR YOUR ATTENTION