Chapter 5
Transport Layer Introduction

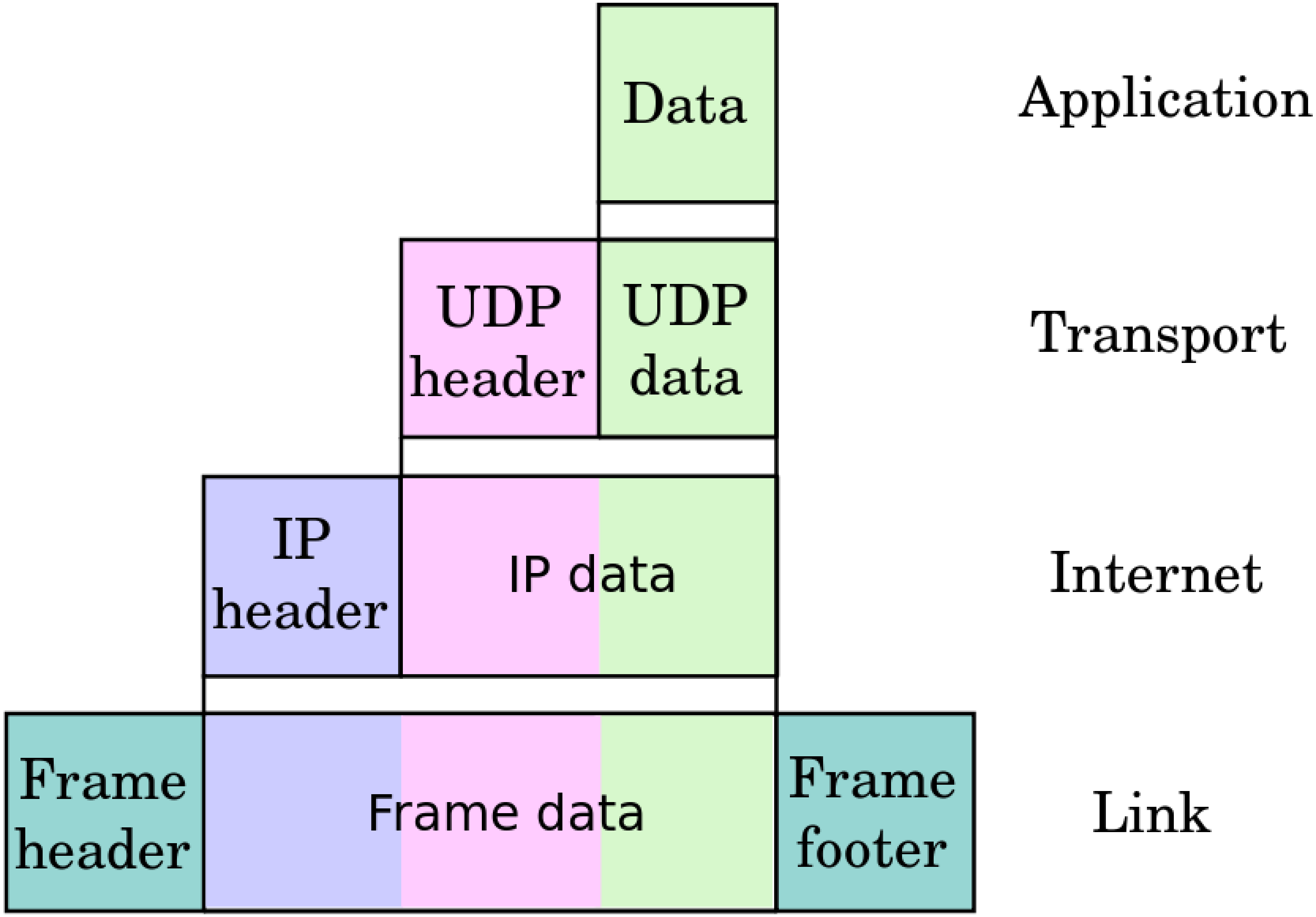
Networking

CS 3470, Section 1

Chapter 5: Transport Layer

- 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

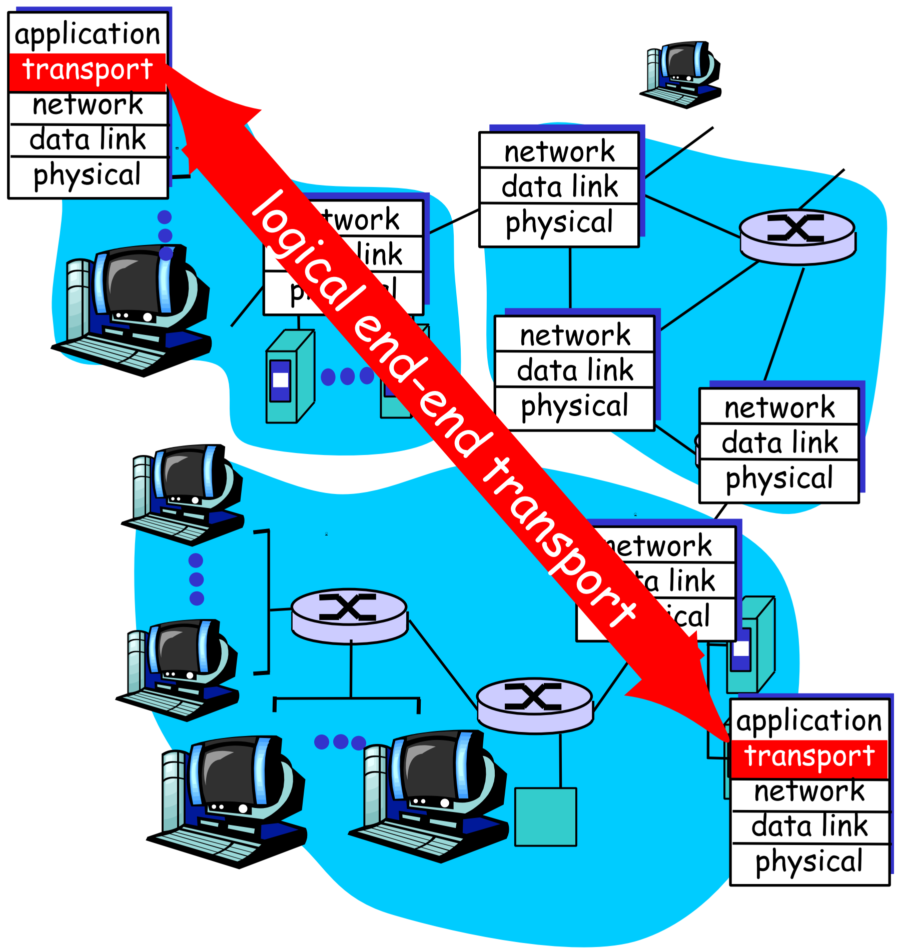
Packet Encapsulation



** Creative Commons: http://en.wikipedia.org/wiki/File:UDP_encapsulation.svg

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

- Why can’t we just use the network layer to send messages from host to host?

End-to-end Protocols

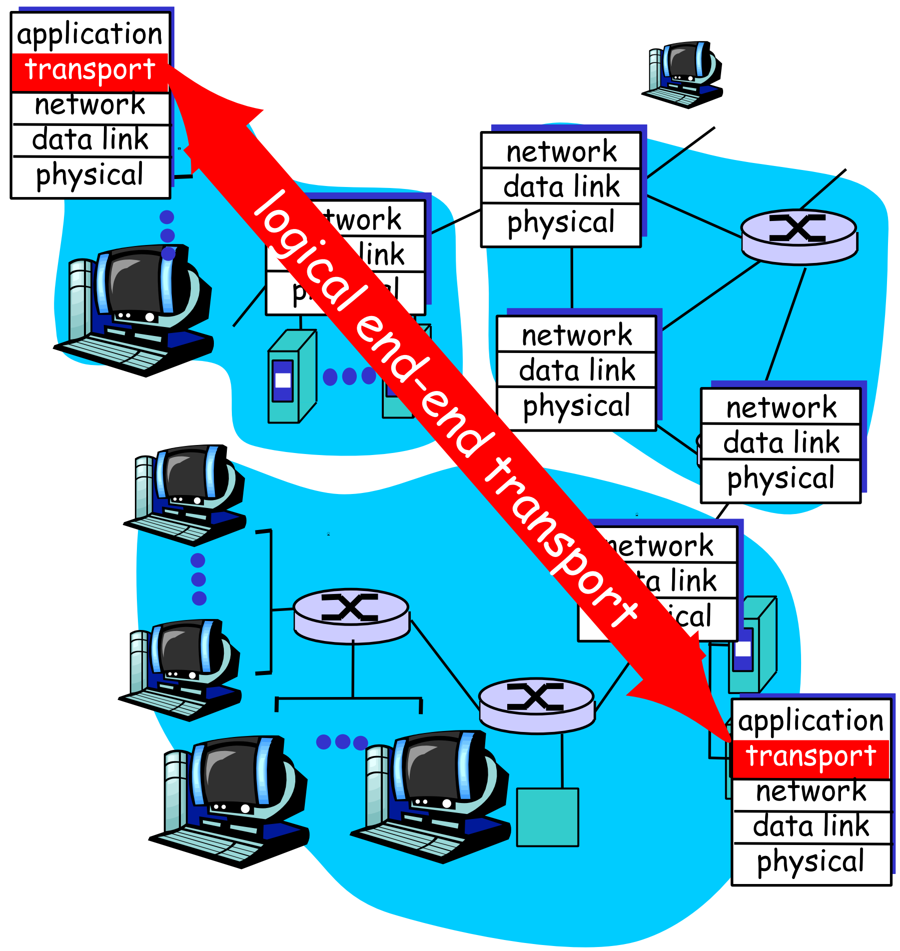
- Common properties that a transport protocol can be expected to provide
  - Guarantees message delivery
  - Delivers messages in the same order they were sent
  - Delivers at most one copy of each message
  - Supports arbitrarily large messages
  - Supports synchronization between the sender and the receiver
  - Allows the receiver to apply flow control to the sender
  - Supports multiple application processes on each host

End-to-end Protocols

- Typical limitations of the network on which transport protocol will operate
  - Drop messages
  - Reorder messages
  - Deliver duplicate copies of a given message
  - Limit messages to some finite size
  - Deliver messages after an arbitrarily long delay

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



UDP: User Datagram Protocol [RFC 768]

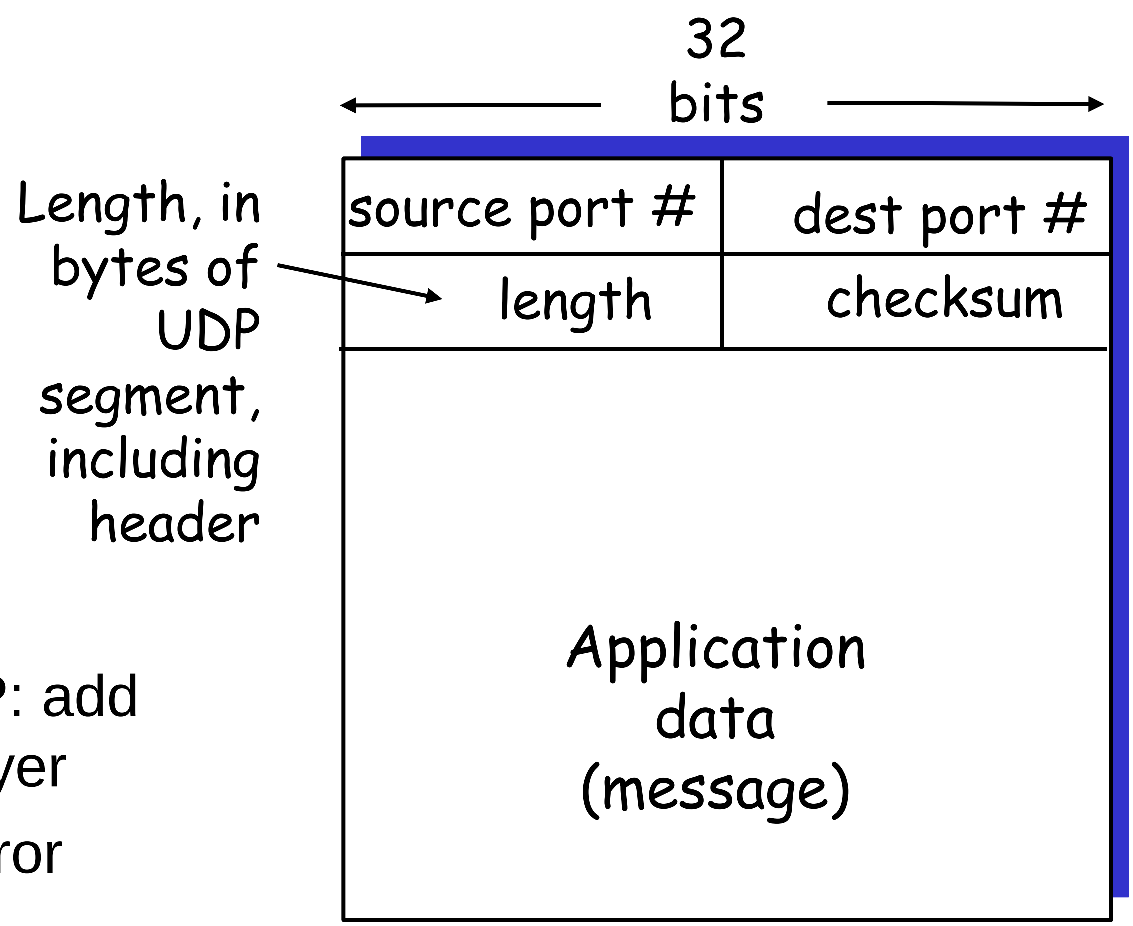
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

UDP: User Datagram Protocol [RFC 768]

- 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. This is the IP checksum from earlier in the course!

- **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?

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

		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

Internet Checksum Example

- What if we are adding three numbers?
  - Add first two numbers
    - Deal with carryout from most significant bit (if necessary)
  - Add sum of first two numbers with third number
    - Deal with carryout from most significant bit (if necessary)
  - Obtain the 1's compliment by converting all 0s to 1s and all 1s to 0s (flip the bits)
- Same general algorithm for adding > 3 numbers

Internet Checksum Example

- At the receiver, all 16-bit words are added (including the checksum)
- If no errors introduced, the sum at the receiver will be all 1's
  - If any of the bits are zero, then we know error(s) have been introduced into the packet

Internet Checksum Example

- UDP checksum takes as input:
  - UDP header
  - Contents of message body
  - ***Pseudoheader***
    - 3 fields from the IP packet: protocol number, src IP address, dst IP address
    - UDP length field
- Motivation of the pseudoheader is to verify this message has been delivered between correct two endpoints

Reliable Byte Stream (TCP)

- In contrast to UDP, Transmission Control Protocol (TCP) offers the following services
  - Reliable
  - Connection oriented
  - Byte-stream service

Flow control VS Congestion control

- ***Flow control*** involves preventing senders from overrunning the capacity of the receivers
- ***Congestion control*** involves preventing too much data from being injected into the network, thereby causing switches or links to become overloaded

End-to-end Issues

- At the heart of TCP is the ***sliding window algorithm*** (discussed in Chapter 2)
- As TCP runs over the Internet rather than a point-to-point link, the following issues need to be addressed by the sliding window algorithm
  - TCP supports logical connections between processes that are running on two different computers in the Internet
  - TCP connections are likely to have widely different RTT times
  - Packets may get reordered in the Internet

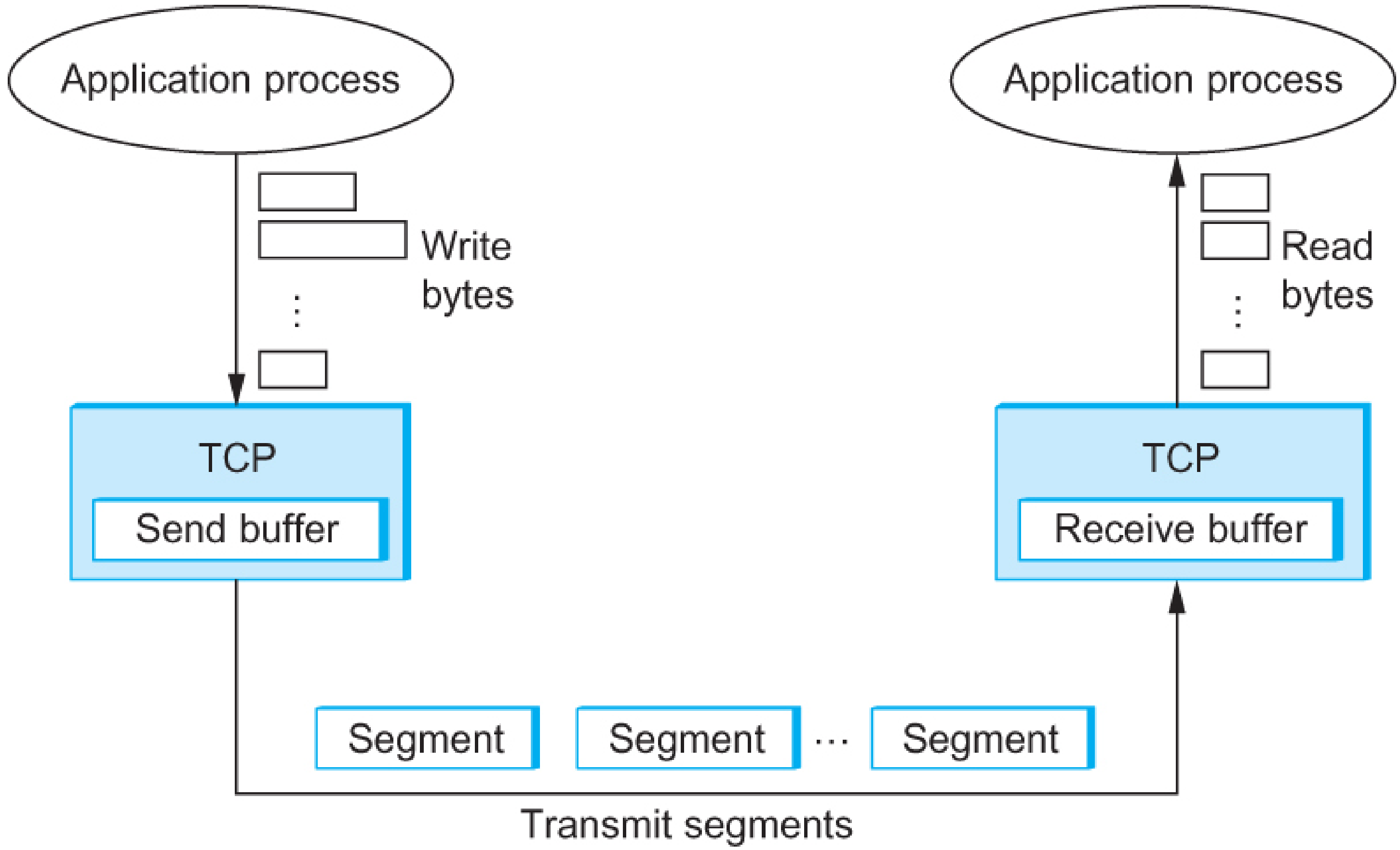
End-to-end Issues

- TCP needs a mechanism using which each side of a connection will learn what resources the other side is able to apply to the connection
- TCP needs a mechanism using which the sending side will learn the capacity of the network

TCP Segment

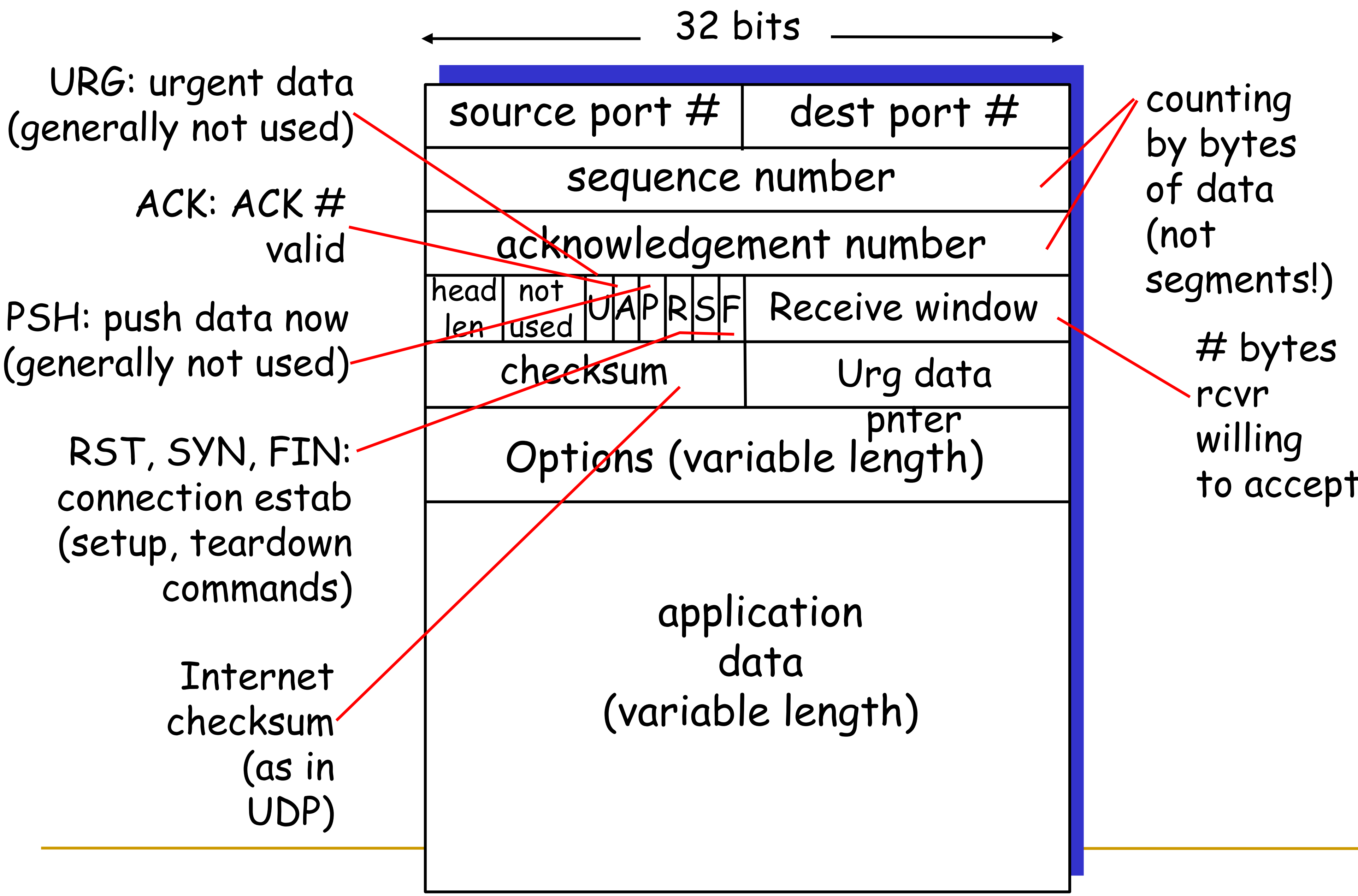
- TCP on the source host buffers enough bytes from the sending process to fill a reasonably sized packet and then sends this packet to its peer on the destination host.
- TCP on the destination host then empties the contents of the packet into a receive buffer, and the receiving process reads from this buffer at its leisure.
- The packets exchanged between TCP peers are called *segments*.

TCP Segment



How TCP manages a byte stream.

TCP segment structure



TCP Header

- The ***SrcPort*** and ***DstPort*** fields identify the source and destination ports, respectively.
- The ***Acknowledgment***, ***SequenceNum***, and ***AdvertisedWindow*** fields are all involved in TCP's sliding window algorithm.
- Because TCP is a byte-oriented protocol, each byte of data has a sequence number; the ***SequenceNum*** field contains the sequence number for the first byte of data carried in that segment.
- The ***Acknowledgment*** and ***AdvertisedWindow*** fields carry information about the flow of data going in the other direction.

TCP Header

- The 6-bit ***Flags*** field is used to relay control information between TCP peers.
- The ***SYN*** and ***FIN*** flags are used when establishing and terminating a TCP connection, respectively.
- The ***ACK*** flag is set any time the Acknowledgment field is valid, implying that the receiver should pay attention to it.

TCP Header

- The ***URG*** flag signifies that this segment contains urgent data.
- The ***PUSH*** flag signifies that the sender invoked the push operation, which indicates to the receiving side of TCP that it should notify the receiving process of this fact.
- Finally, the ***RESET*** flag signifies that the receiver has become confused

TCP Header

- Finally, the ***Checksum*** field is used in exactly the same way as for UDP
  - Computed over the TCP header, the TCP data, and the pseudoheader, which is made up of the source address, destination address, and length fields from the IP header.

Multiplexing/demultiplexing

- What do these words mean, again?

Multiplexing/demultiplexing

- What do these words mean, again?
- ***Multiplexing*** – combines multiple streams of information for transmission over a shared medium
- ***Demultiplexing*** – takes combined streams of information and separates the streams
  - Often abbreviated as ***demux***

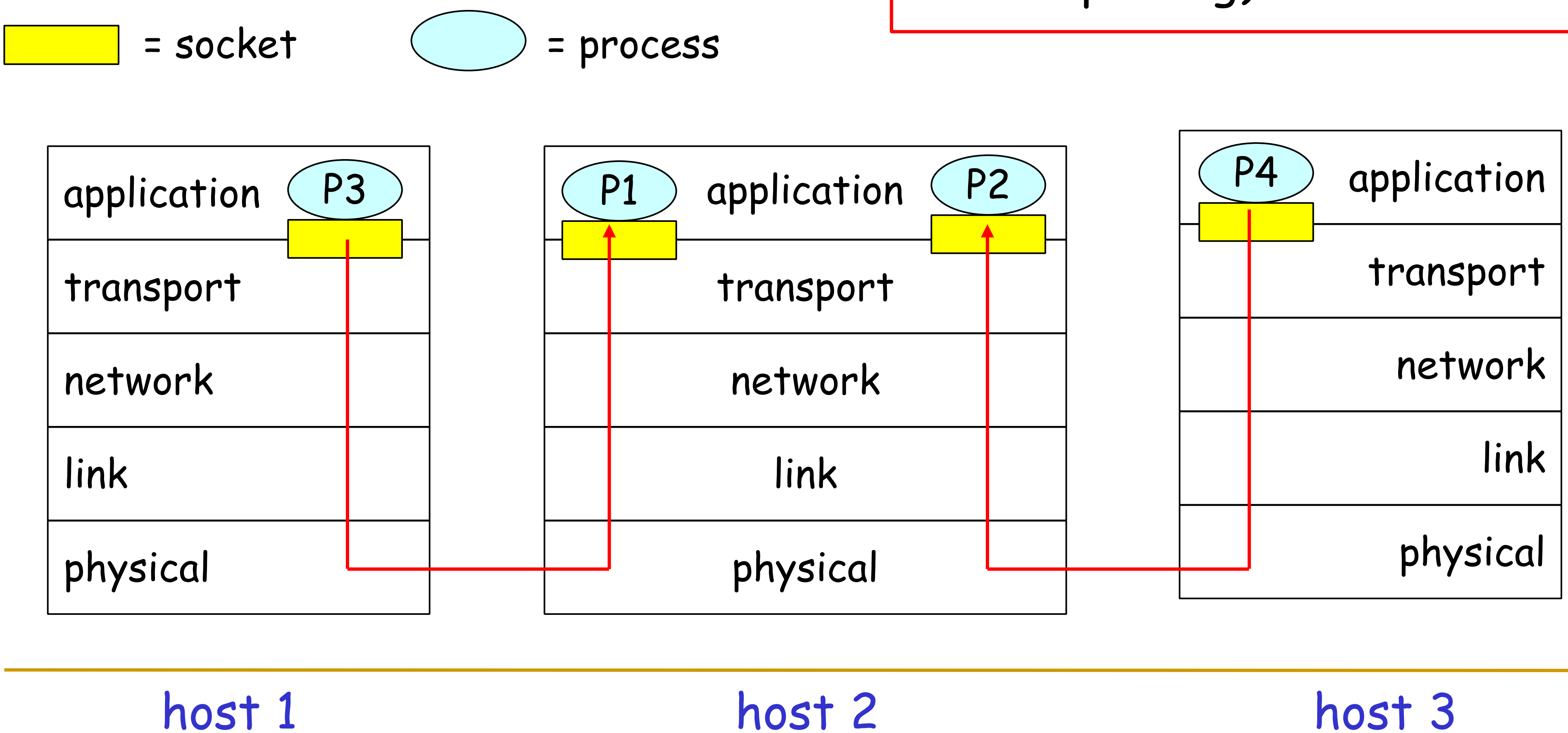
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

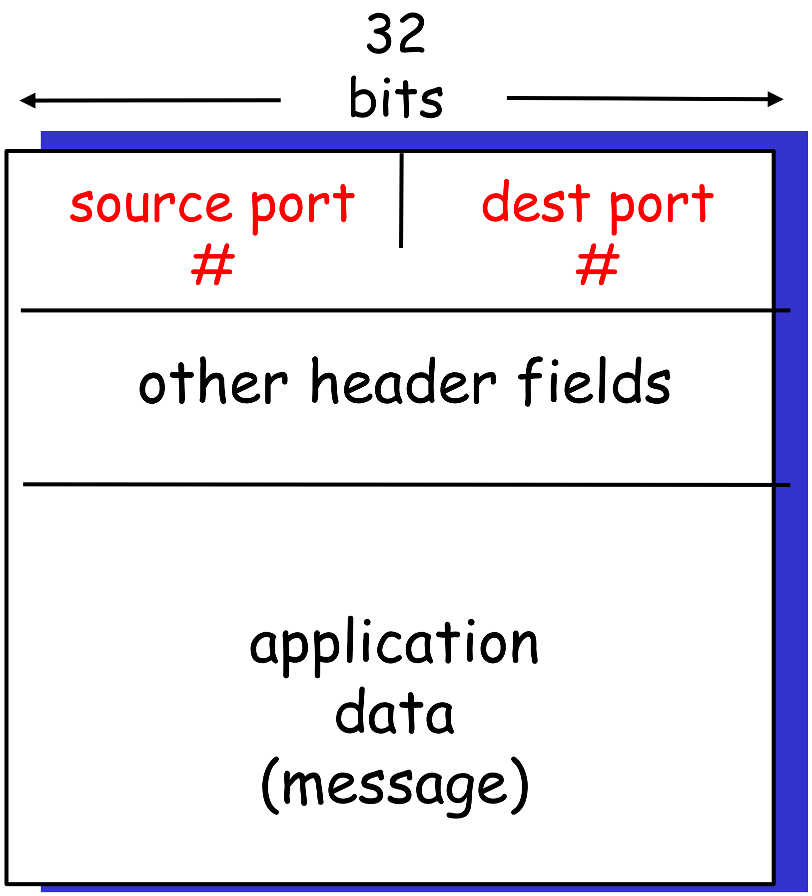
Multiplexing at send host:

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 (recall: well-known port numbers for specific applications)
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
   DatagramSocket(99111);
DatagramSocket mySocket2 = new
   DatagramSocket(99222);
```

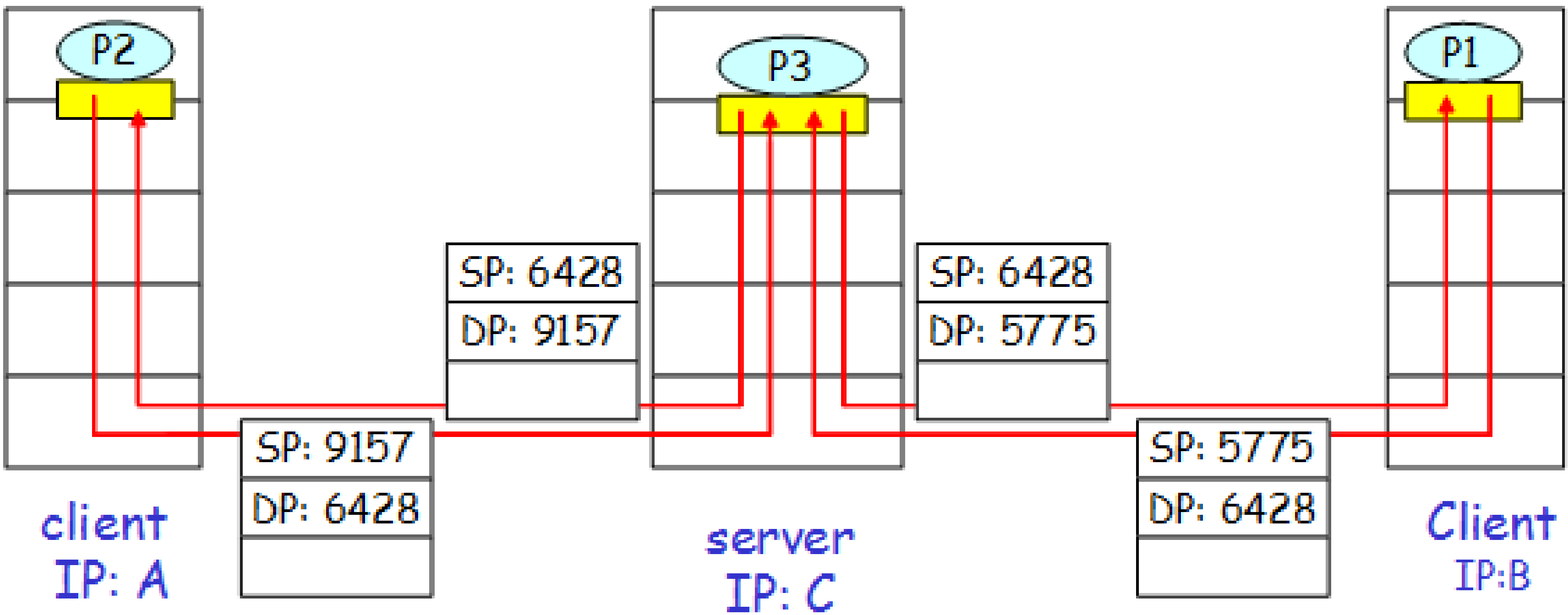
- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

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



SP provides "return address"

Source Ports and Destination Ports

- How does UDP (or TCP) know which source and destination ports to use?

Source Ports and Destination Ports

- How does UDP (or TCP) know which source and destination ports to use?
- Servers tend to listen on “well-known ports”
- We can utilize this information
    - Client uses well-known destination port number (e.g. port 80 to connect to web server)
    - Server uses well-known source port number to listen for incoming connections (e.g. port 80 if web server)

Source Ports and Destination Ports

- What about the client's source port?
- Operating systems pick a random, temporary port
  - Guaranteed to be > 1024
  - IANA suggests between 49,152-65,535
- What about the server's destination port?

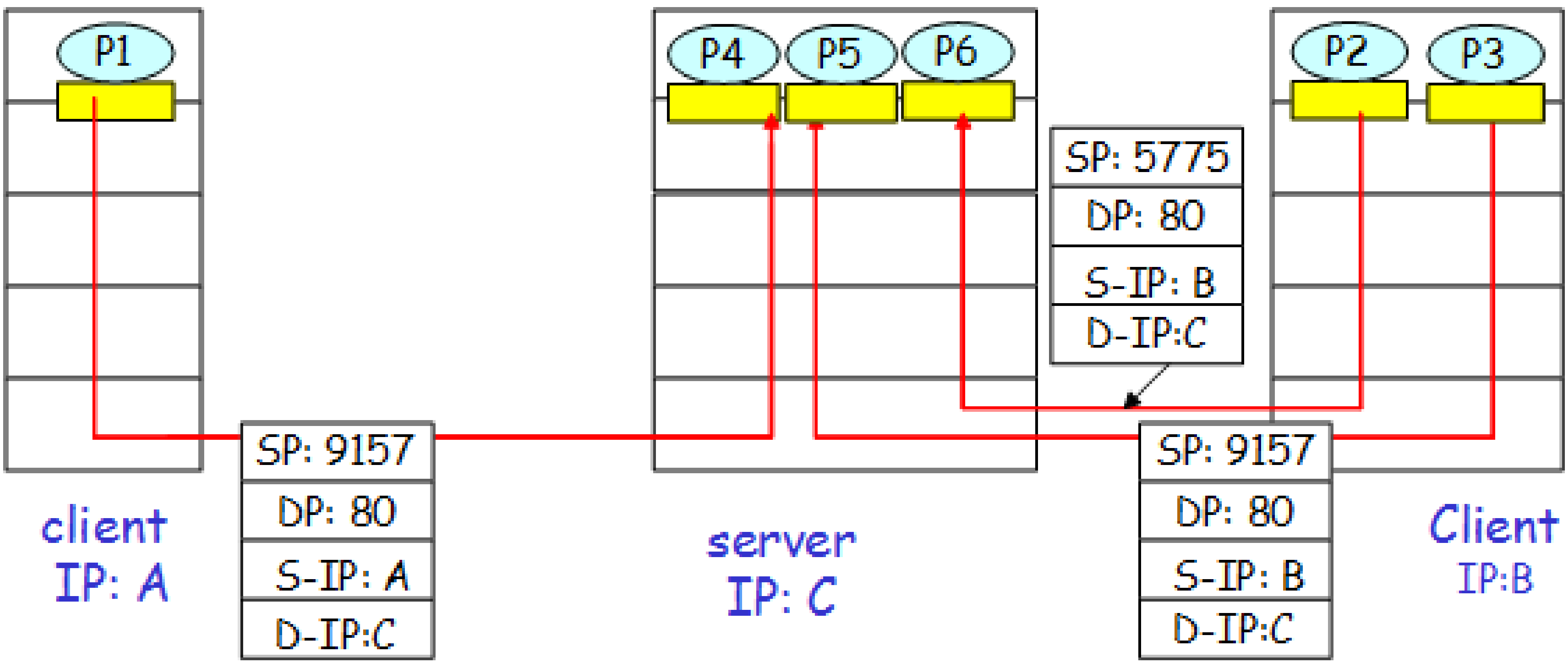
Connection-oriented demux

- 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

Connection-oriented demux

- 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