CMSC 332
Computer Networks
Transport Layer

Professor Szajda

Chapter 3: Transport Layer

Our goals:

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

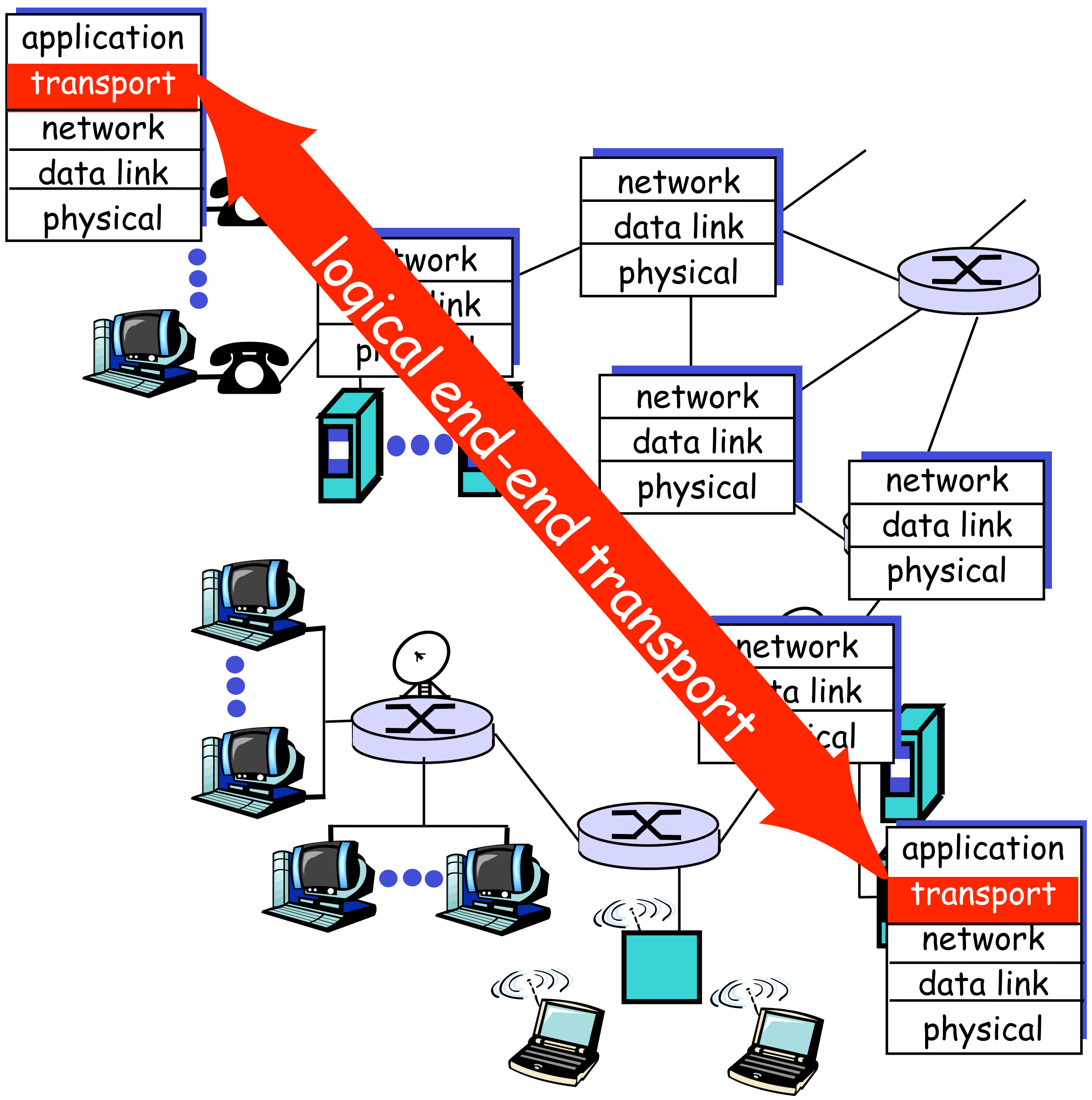
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

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

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

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

Household analogy:

12 kids sending letters to 12 kids

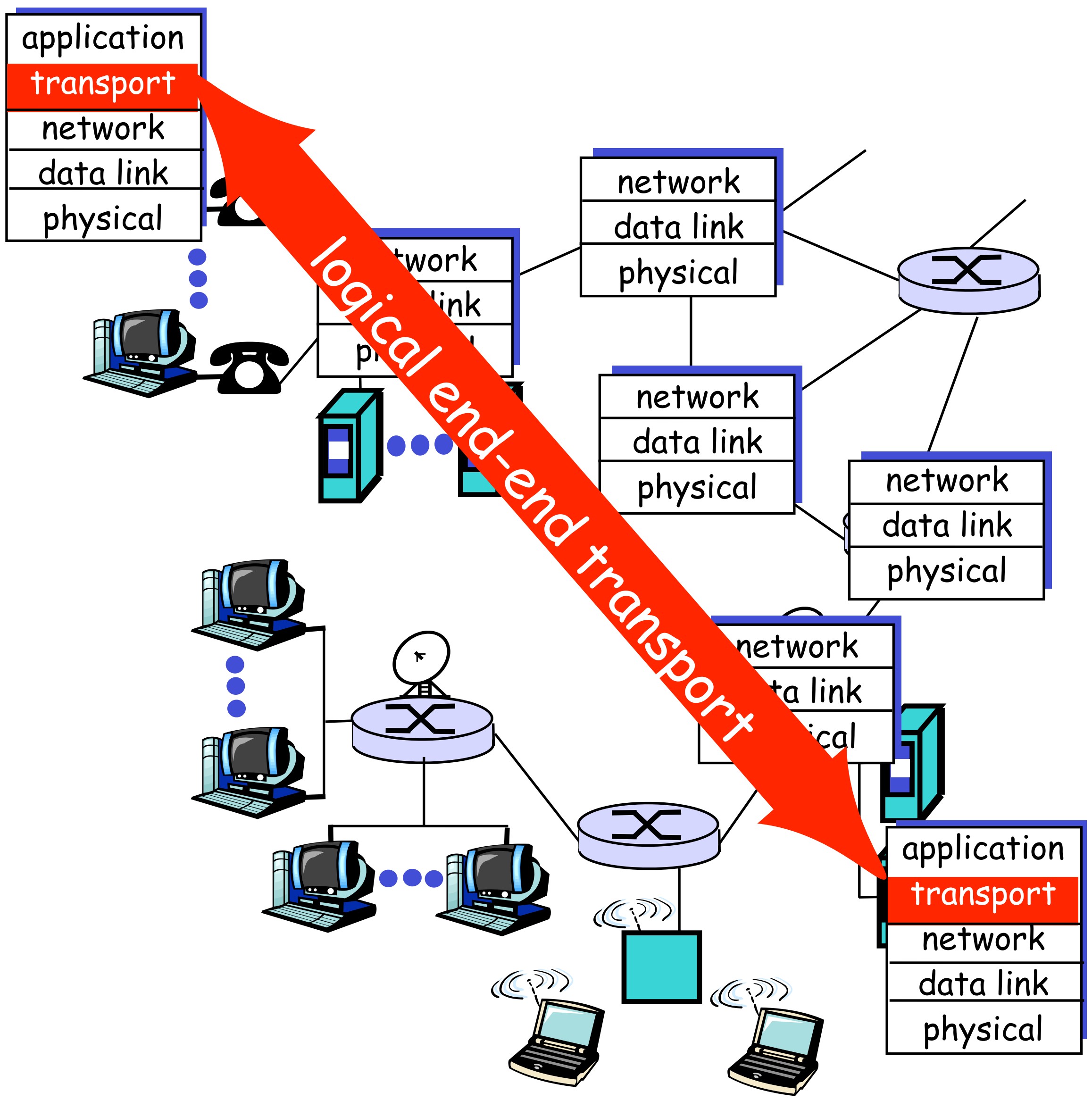
- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Layers of Networks?

- You can view each layer that we have discussed thus far as an abstract network:
  - Application Layer Networks: P2P, Social Networks, etc
  - Transport Layer Networks: Communicating processes
  - Network Layer Networks: Networks of Hosts
  - Link Layer Networks: One-Hop Networks
  - Physical Layer Networks: Wires

Internet transport-layer protocols

- reliable, in-order delivery, no duplicates (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

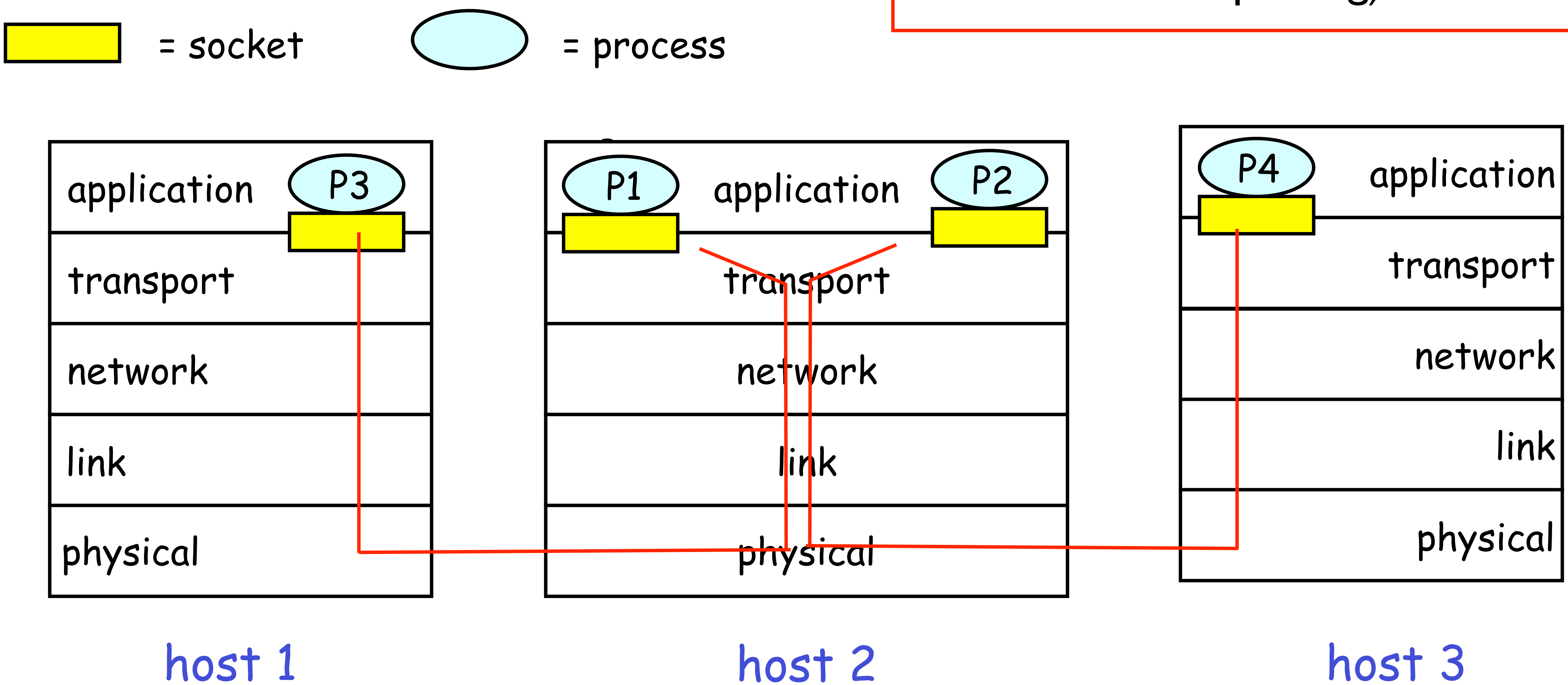
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

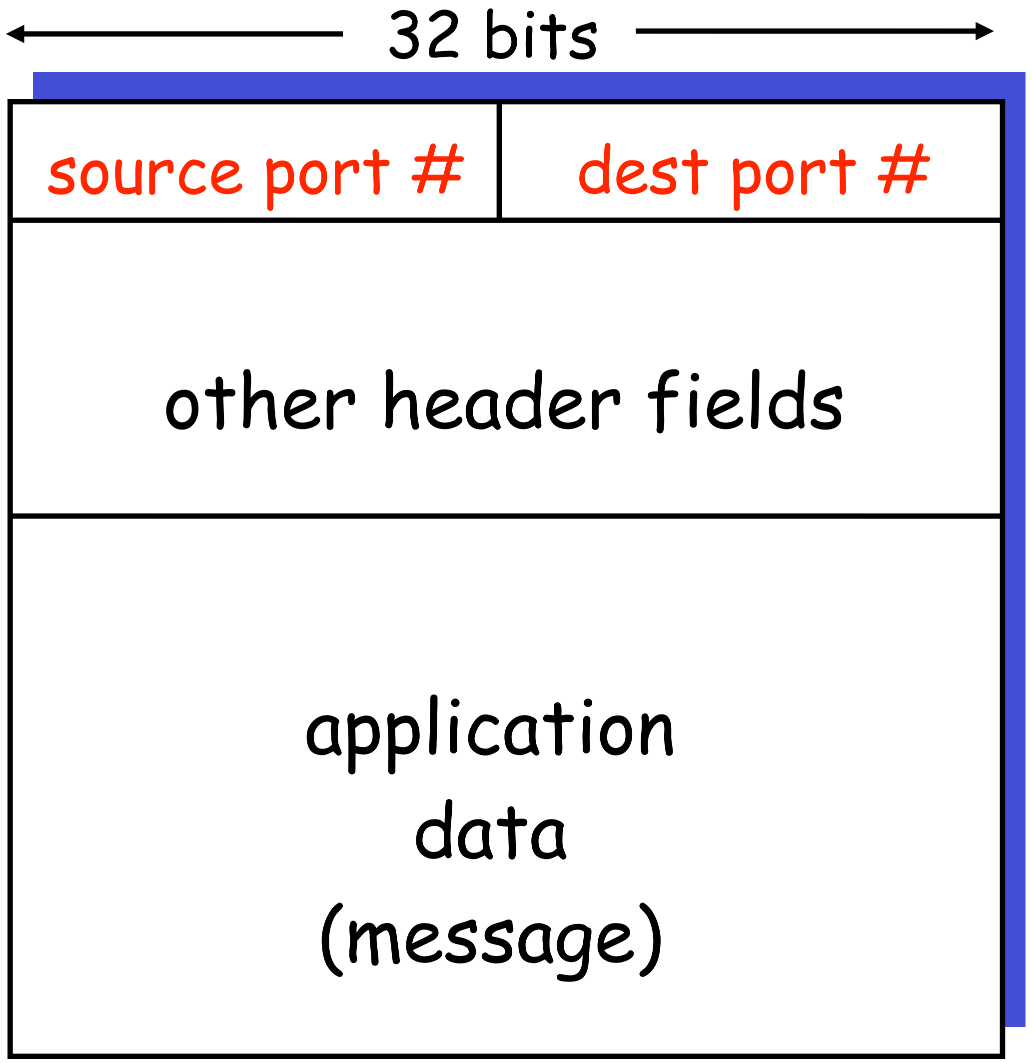
Multiplexing at send host:

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



TCP/UDP segment format

Connectionless demultiplexing

- Create sockets with port numbers:

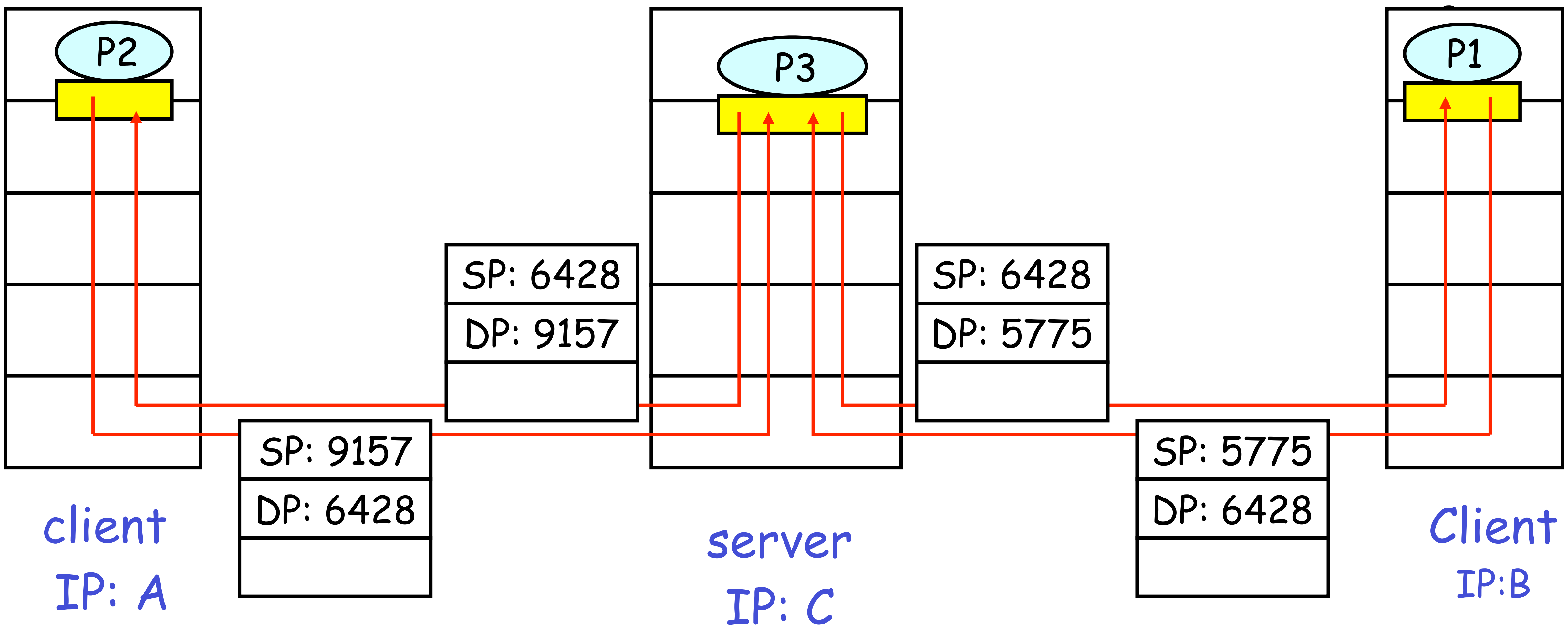
```
addr1.sin_port = htons(12534);

addr2.sin_port = htons(12535);
```

- 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)



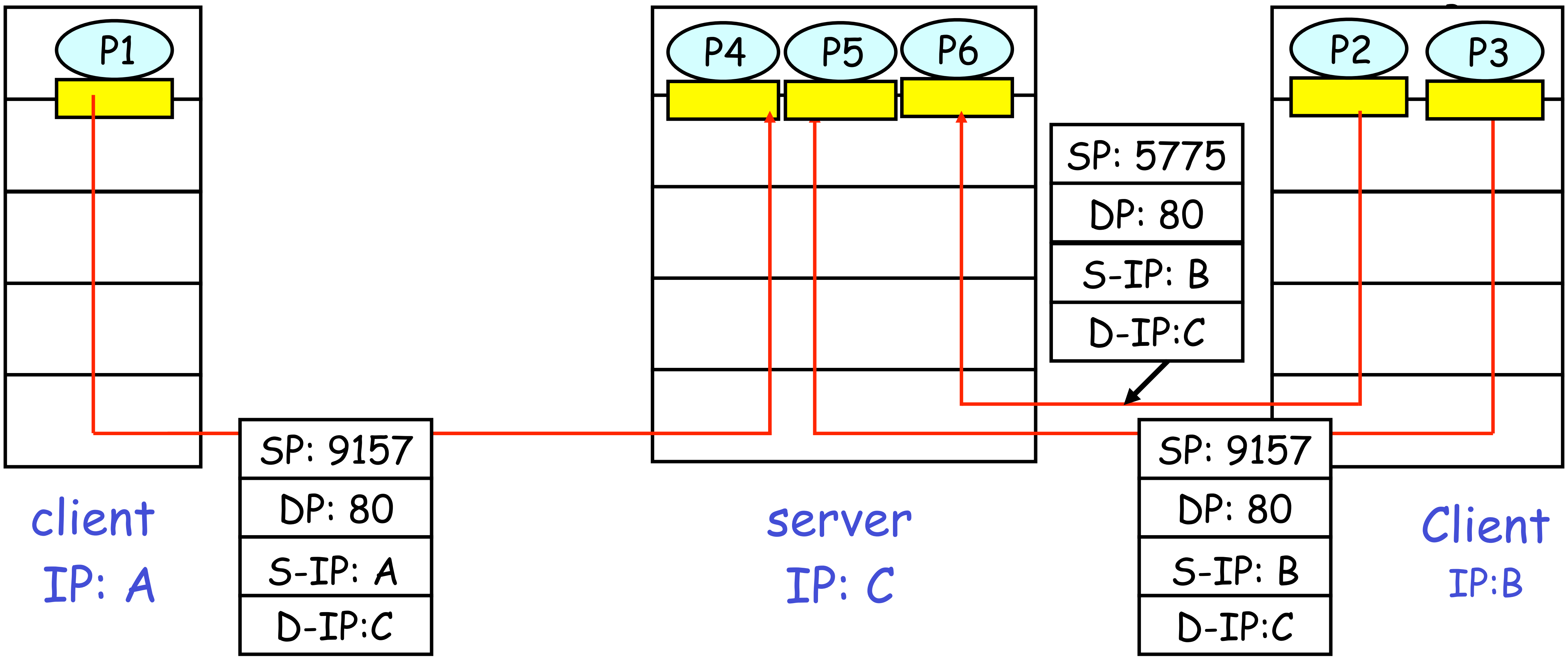
SP provides "return address"

Connection-oriented demux

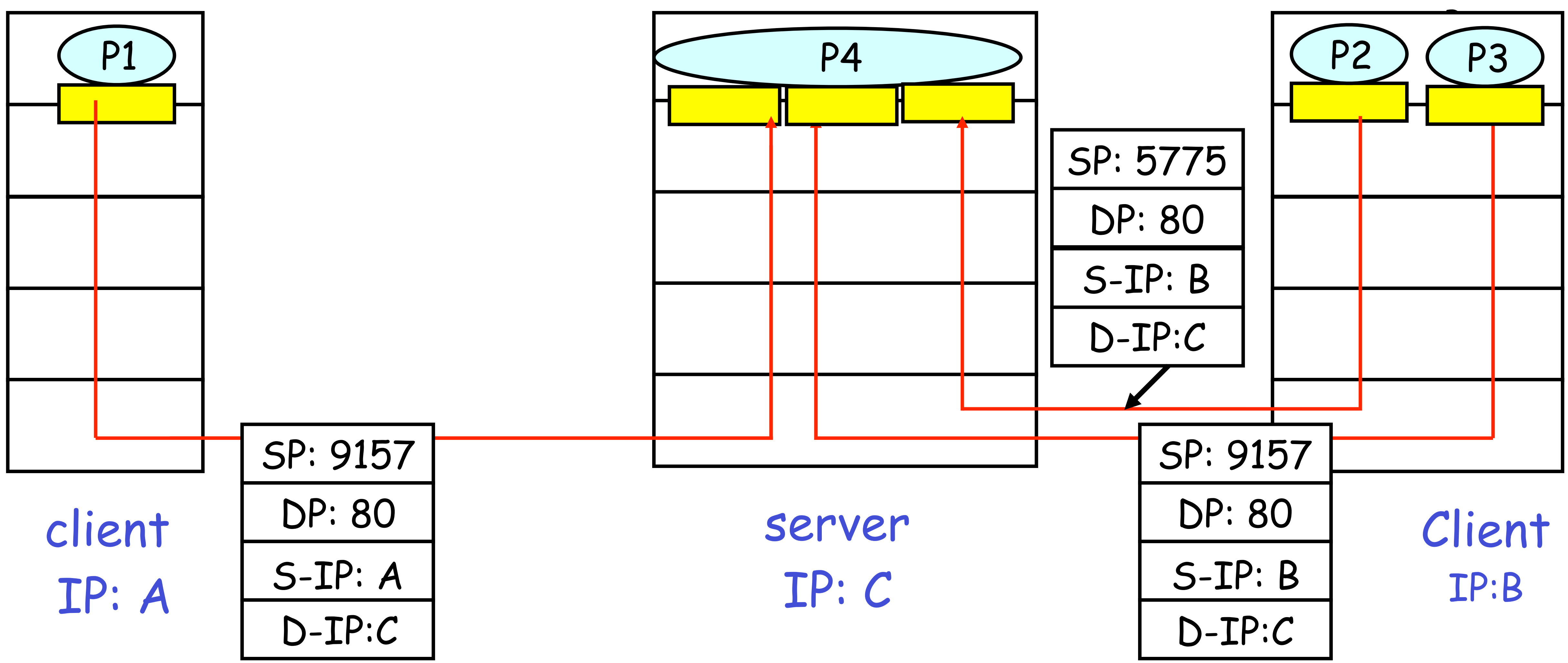
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket

- 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



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

UDP: User Datagram Protocol [RFC 768]

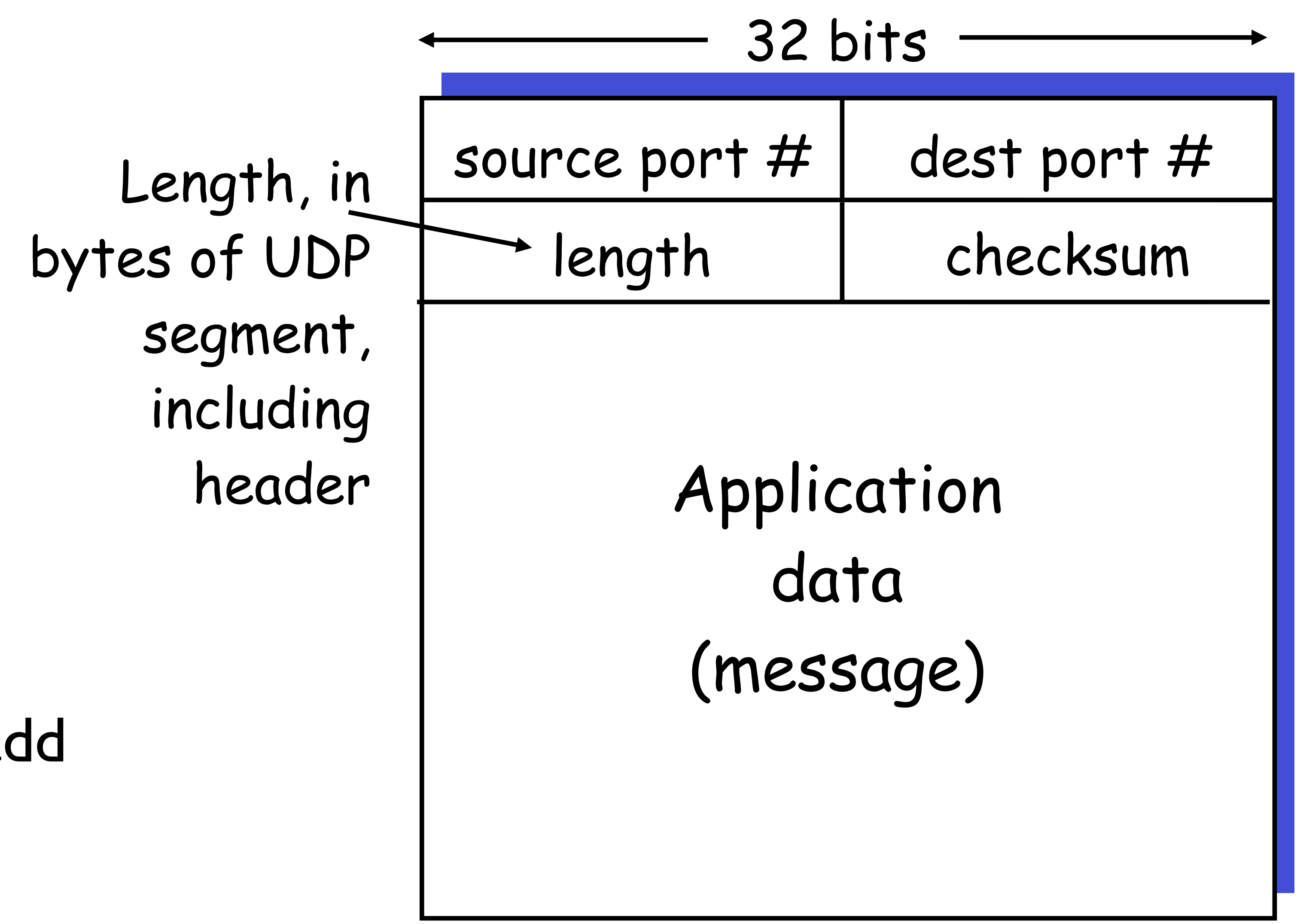
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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

UDP pseudo-header for checksum computation (IPv4)

<table>
<tr><th>Offset</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="32">Source Address</td></tr>
<tr><th>4</th><th>32</th><td colspan="32">Destination Address</td></tr>
<tr><th>8</th><th>64</th><td colspan="8">Zeroes</td><td colspan="8">Protocol</td><td colspan="16">UDP Length</td></tr>
<tr><th>12</th><th>96</th><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><th>16</th><th>128</th><td colspan="16">Length</td><td colspan="16">Checksum</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Data</td></tr>
<tr><th>24</th><th>192</th></tr>
<tr><th>⋮</th><th>⋮</th></tr>
</table>

Note: This is *not* the actual header that is sent!

UDP checksum

UDP pseudo-header for checksum computation (IPv6)

<table>
<tr><th>Offset</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="32" rowspan="4">Source address</td></tr>
<tr><th>4</th><th>32</th></tr>
<tr><th>8</th><th>64</th></tr>
<tr><th>12</th><th>96</th></tr>
<tr><th>16</th><th>128</th><td colspan="32" rowspan="4">Destination address</td></tr>
<tr><th>20</th><th>160</th></tr>
<tr><th>24</th><th>192</th></tr>
<tr><th>28</th><th>224</th></tr>
<tr><th>32</th><th>256</th><td colspan="32">UDP length</td></tr>
<tr><th>36</th><th>288</th><td colspan="24">Zeroes (0)</td><td colspan="8">Next Header (17)</td></tr>
<tr><th>40</th><th>320</th><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><th>44</th><th>352</th><td colspan="16">Length</td><td colspan="16">Checksum</td></tr>
<tr><th>48</th><th>384</th><td colspan="32" rowspan="3">Data</td></tr>
<tr><th>52</th><th>416</th></tr>
<tr><th>⋮</th><th>⋮</th></tr>
</table>

UDP checksum

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

Sender:

- treat segment contents as sequence of 16-bit integers
- What exactly is "contents"?
  - Pseudo header taken from selected fields in IP header (included in case packet is misrouted to wrong IP address due to bit error)
  - UDP header
  - UDP data

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 ….

UDP checksum

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

Sender:

- treat segment contents as sequence of 16-bit integers
- What exactly is "contents"?
  - Pseudo header taken from selected fields in IP header (included in case packet is misrouted to wrong IP address due to bit error)
  - UDP header
  - UDP data

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 ….

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
  - How is this different than 2's complement?
- 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

		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

Port Scanning

- Technique used by black- and white-hat communities alike.
- Attempts to connect to a large number (usually all) of ports on a machine.
  - Successful responses mean that a process is running.
  - If you know what processes are running, you may be able to select the right exploit to launch.
  - Most firewalls offer some protection against this.
- This is happening *all the time* on the Internet.
  - The bad guys are constantly looking for a way in...

Port Scanning Tools

- *nmap* is a popular tool for port scanning.
  - ...and it is free...
- By seeing which ports are active, nmap can tell a lot about your machine.
  - For instance, what OS you are running...
- Be careful to check with admins before running this!
  - Most admins will automatically shut you down if you run it...