CS 330: Network Applications & Protocols

Transport Layer

Galin Zhelezov
Department of Physical Sciences
York College of Pennsylvania

Overview of Transport Layer

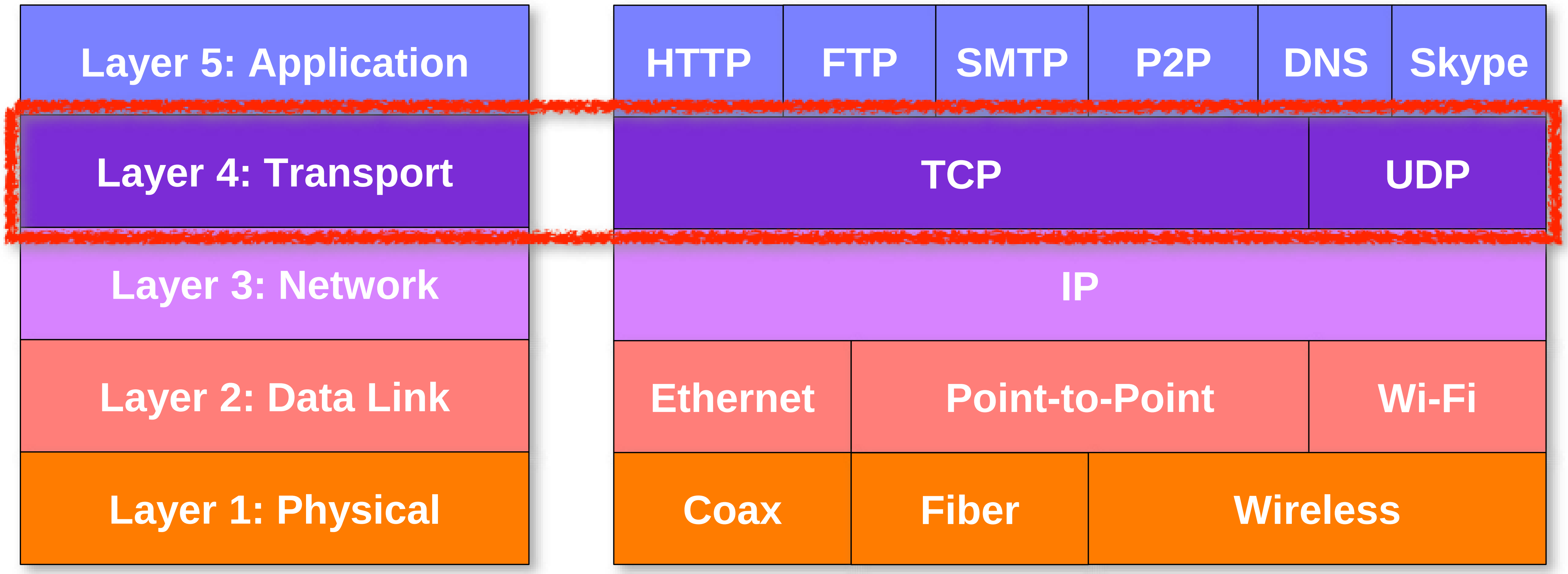
- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
- **Principles of Congestion Control**
- **TCP Congestion Control**

Overview of Transport Layer

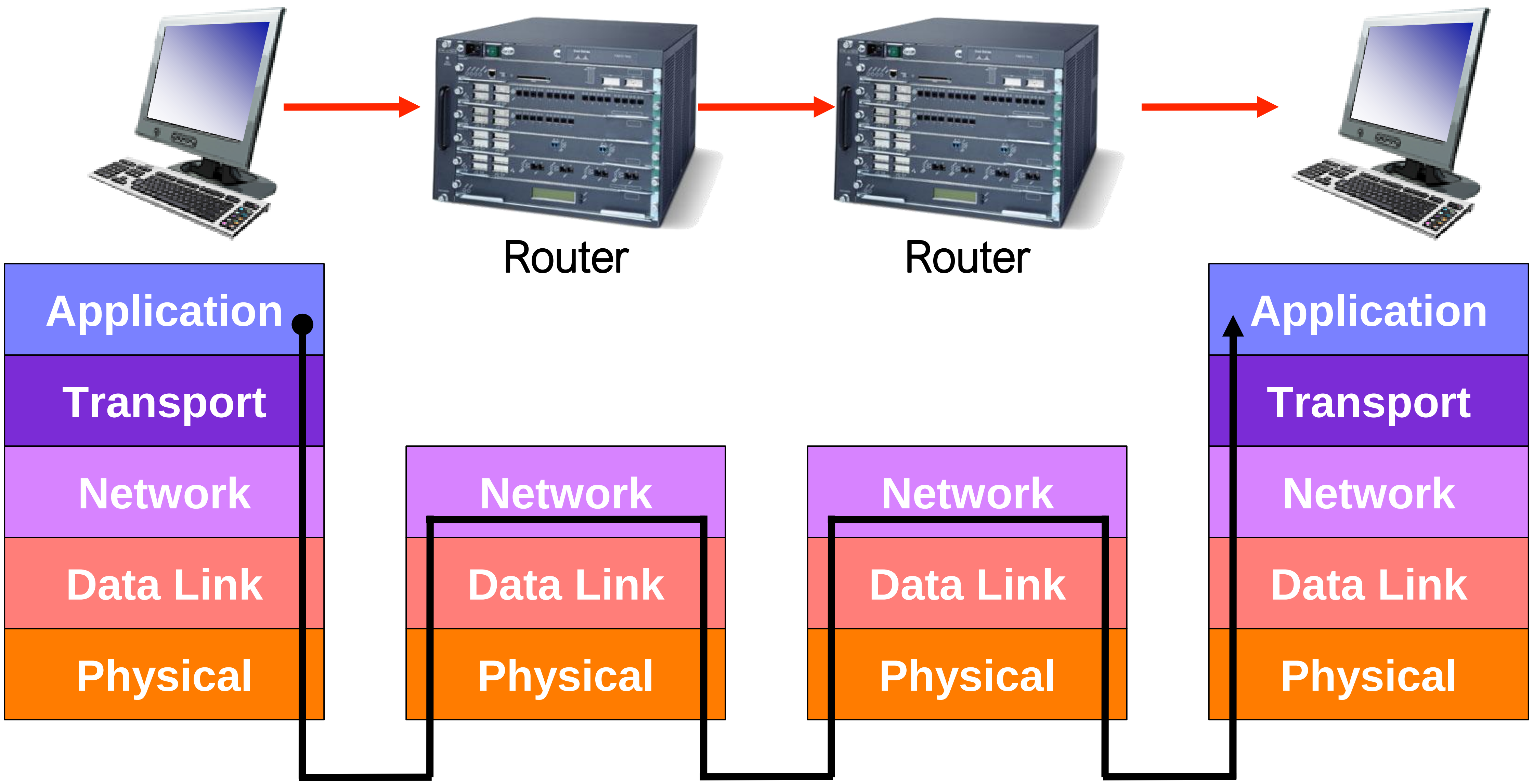
- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
- **Principles of Congestion Control**
- **TCP Congestion Control**

Protocol Layers

- **Top-Down Approach**



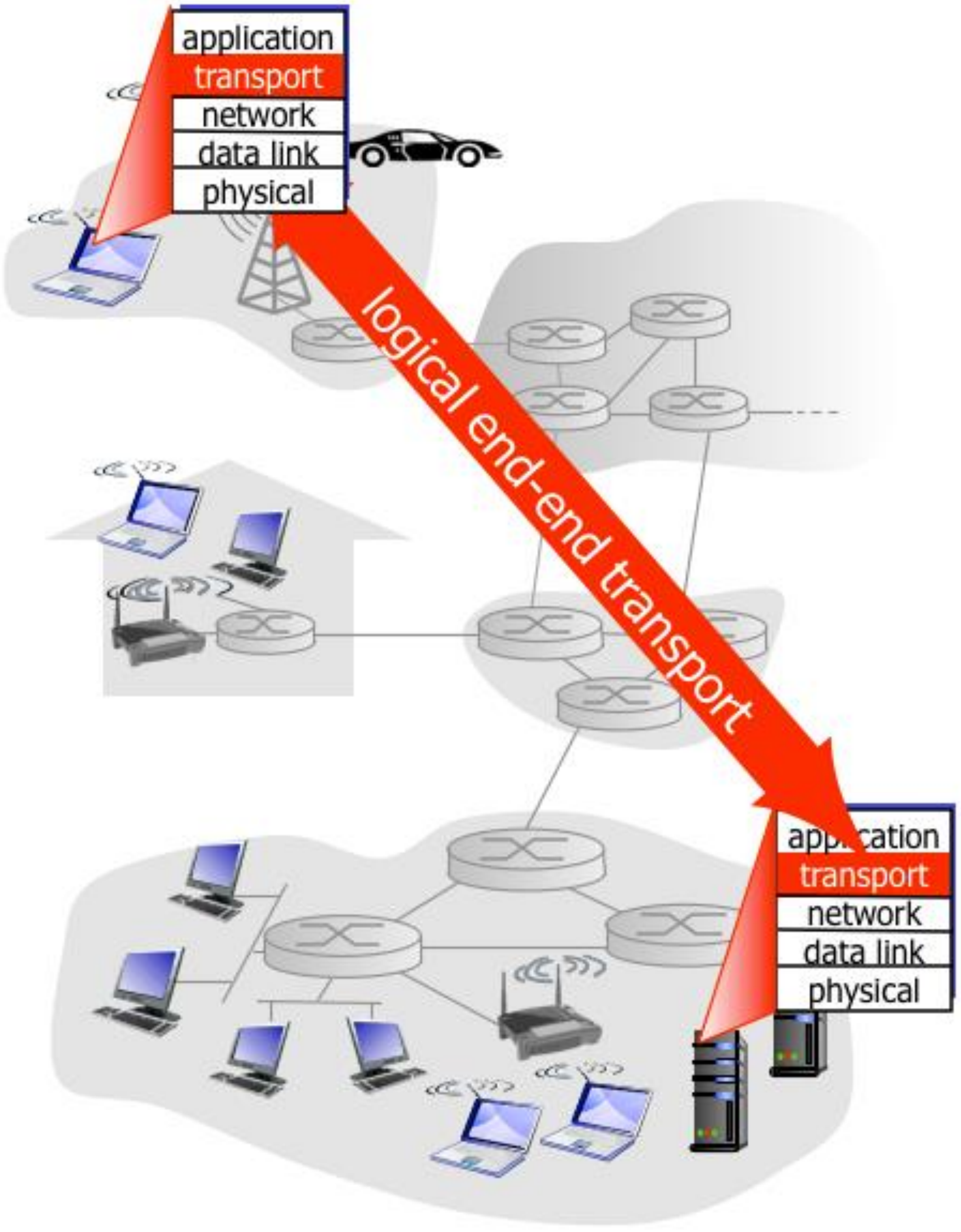
Transport Layer



- **Transport layer provides End-to-End Services**
  - Required at source and destination
  - Not required at intermediate hops

Transport Services and Protocols

- **Provide logical communication between application processes running on different hosts**
- **Transport protocols run in end systems**
  - Sending side: breaks application messages into segments, passes segments down to network layer
  - Receiving side: reassembles segments into messages, passes to application 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

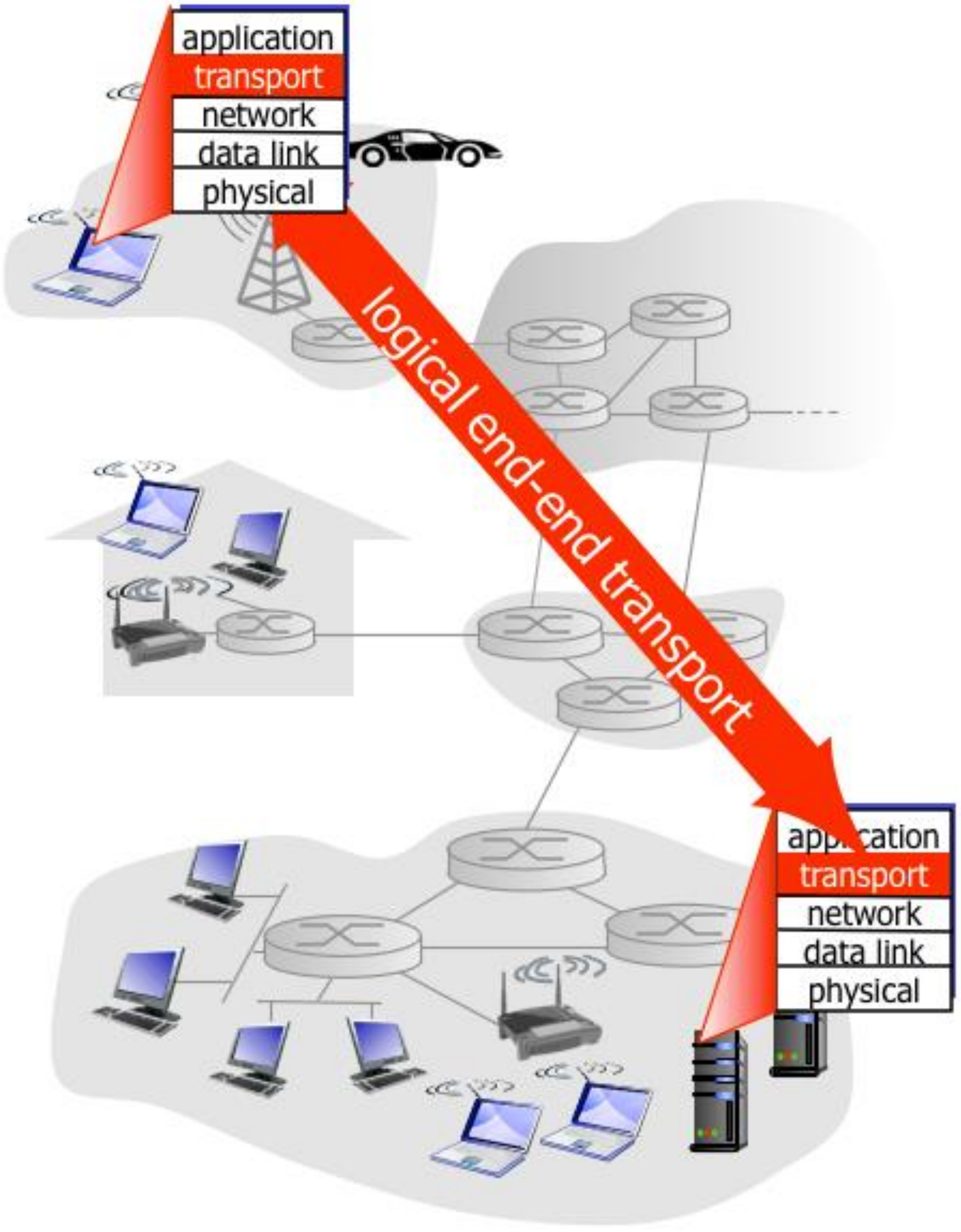
household analogy:

12 kids 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

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



Transport Layer Functions

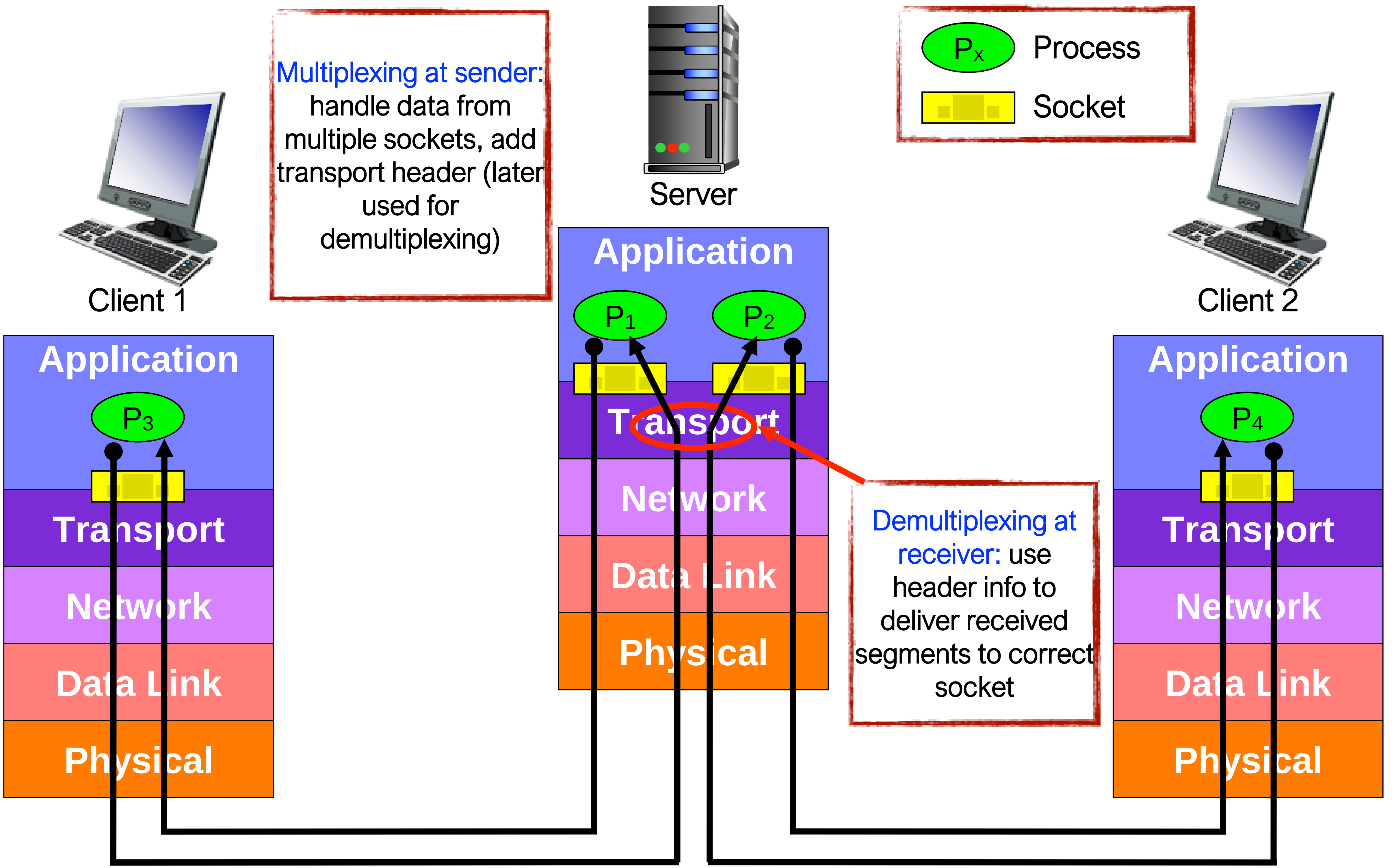
- **Multiplexing and demultiplexing among applications and processes at end systems**
- **Error detection of bit errors**
- **Loss detection - lost packets due to buffer overflow at intermediate systems**
- **Error/loss Recovery - retransmissions**
- **Flow Control - ensures receiver has buffer capacity to receive message**
- **Congestion Control - ensure the network has the capacity to transmit data**

Not all transport protocols provide all of the above functionality

Overview of Transport Layer

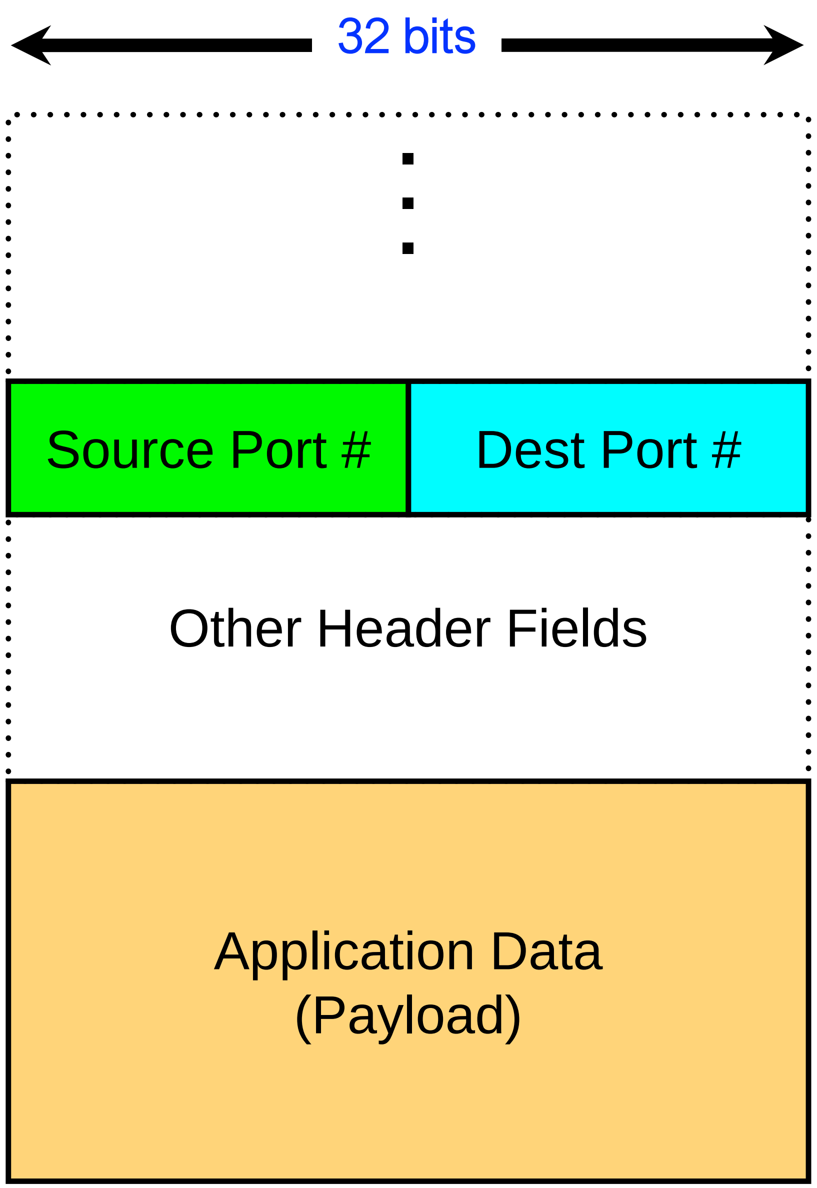
- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
  - Connectionless Demultiplexing
  - Connection-Oriented Demultiplexing
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
- **Principles of Congestion Control**
- **TCP Congestion Control**

Multiplexing/Demultiplexing



How Demultiplexing Works

- **Host receives IP datagrams**
  - Each datagram has source IP address and destination IP address
  - Each datagram carries one transport-layer segment
  - Each segment has source and destination port number
- **Host uses IP addresses & port numbers to direct segment to appropriate socket**



TCP/UDP Segment Format

Connectionless demultiplexing

- ***recall:* created socket has host-local port #:**

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

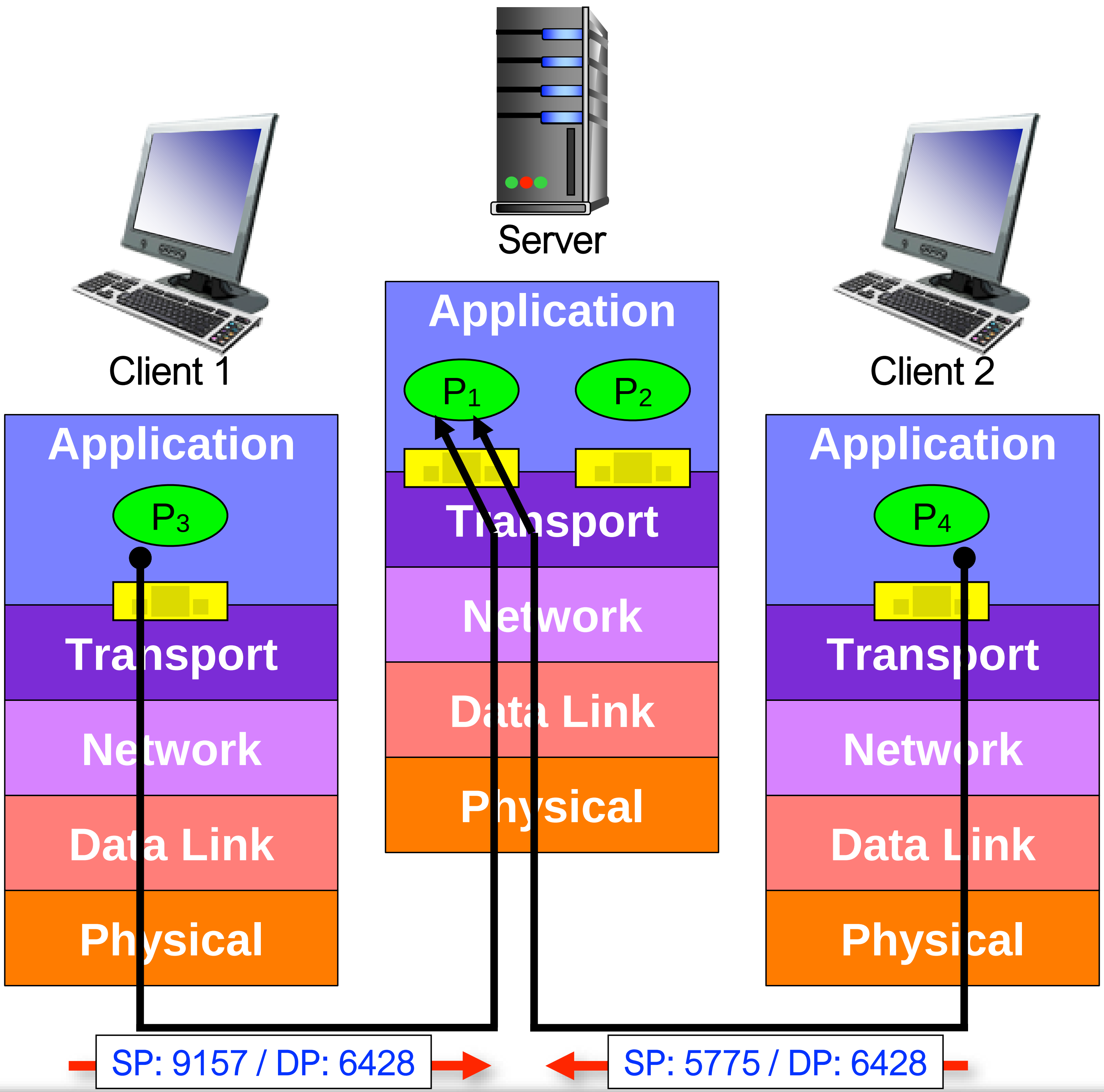
- *recall:* when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

- **when host receives UDP segment:**
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

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

Connectionless Demux: Example (UDP)

- **Server UDP socket has a local port #**
  - Same socket is shared for incoming connections destined for that port #
- **Each client:**
  - Creates own local socket with own local port #
  - When sending UDP datagram to server, client must specify IP address & port # of server's UDP socket

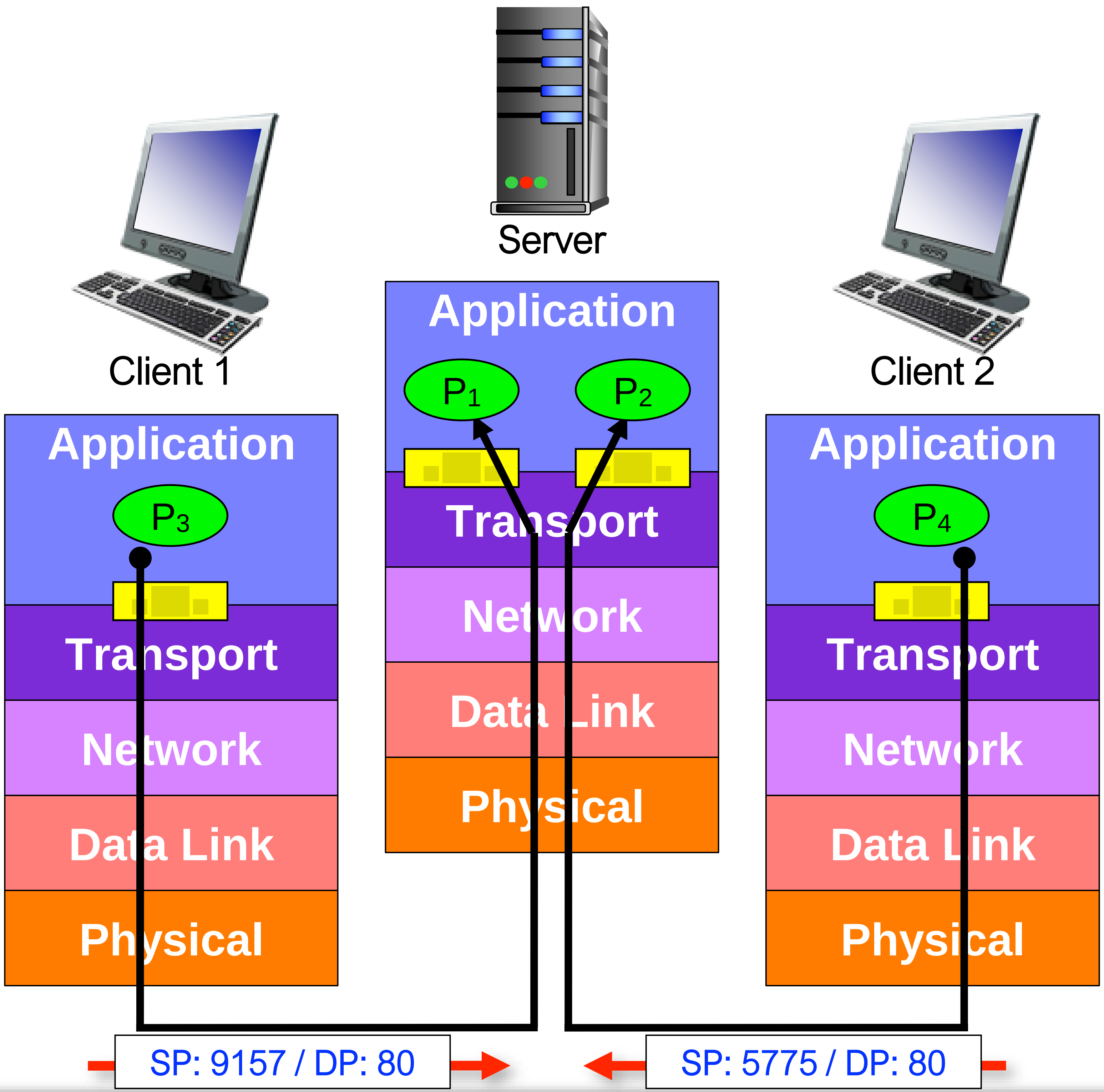


Connection-oriented demux

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

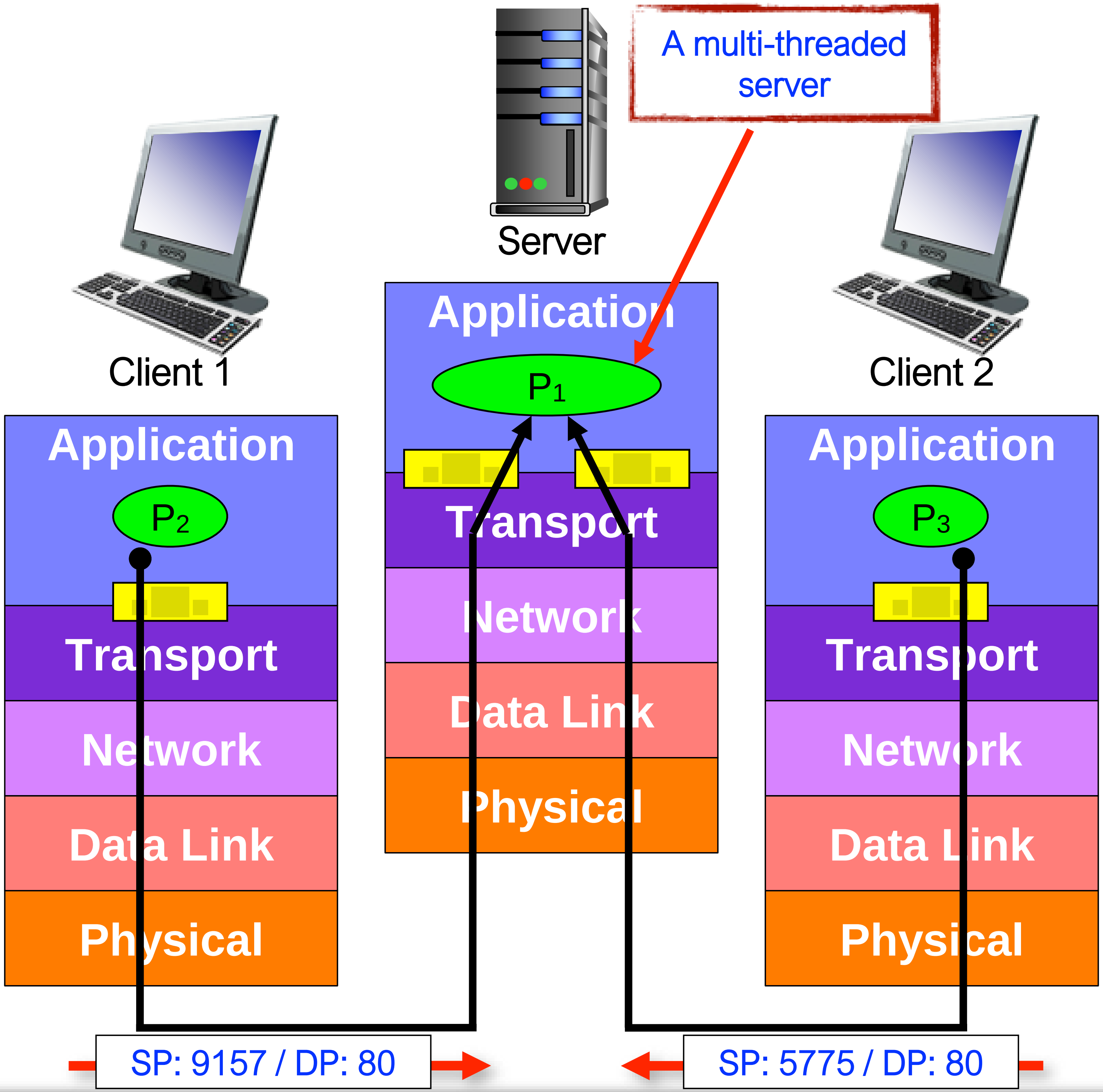
Connection-Oriented Demux: Example (TCP)

- **TCP socket is identified by a 4-tuple**
  - Source IP address, Source port number, Dest IP address, Dest port number
- **A new TCP socket is created for each unique 4-tuple**
  - Server host may support many simultaneous TCP sockets (even multiple from same client)



Connection-Oriented Demux: Example (TCP)

- **TCP socket is identified by a 4-tuple**
  - Source IP address, Source port number, Dest IP address, Dest port number
- **A new TCP socket is created for each unique 4-tuple**
  - Server host may support many simultaneous TCP sockets (even multiple from same client)



Overview of Transport Layer

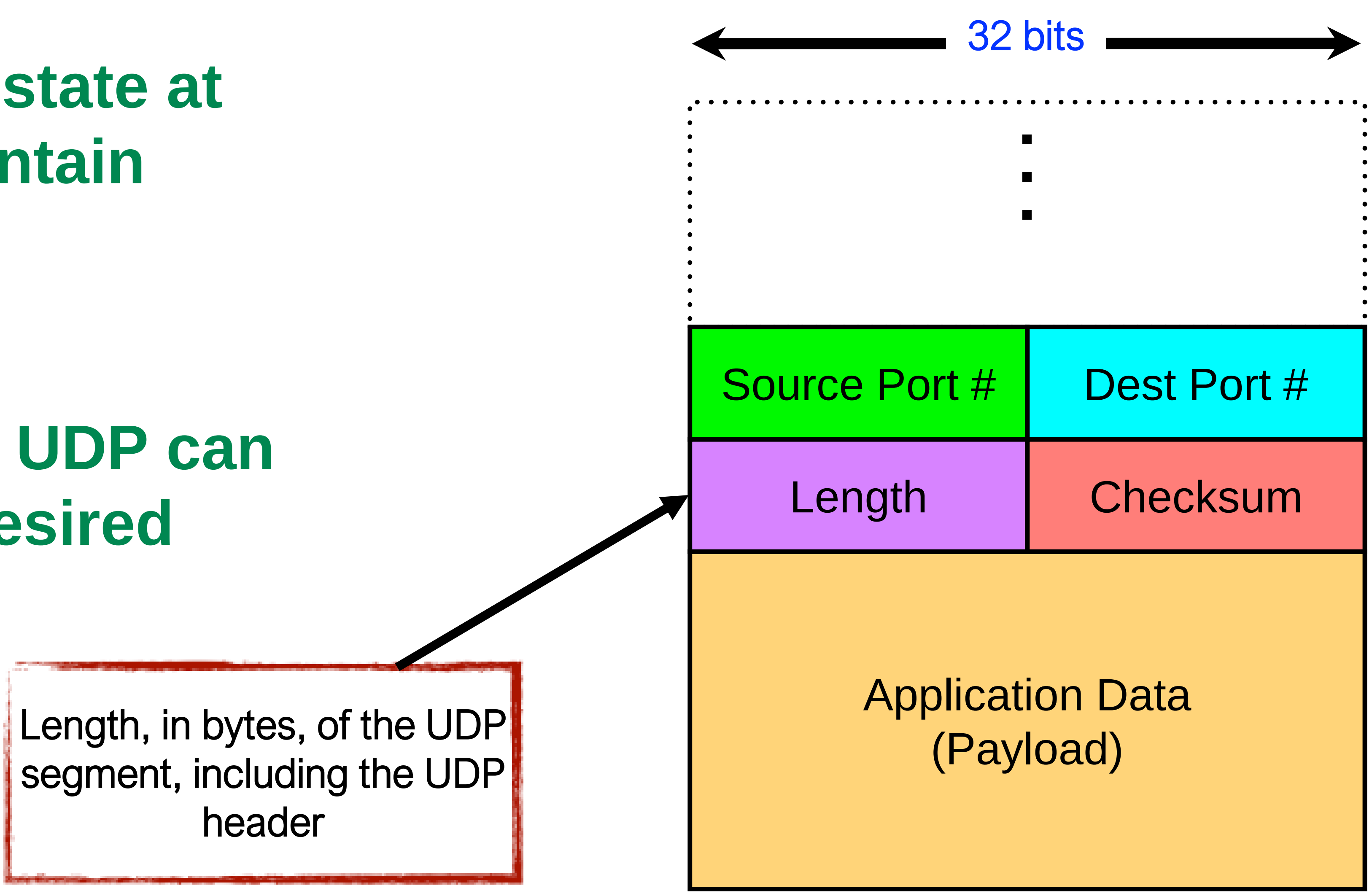
- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
  - Overview
  - Checksum
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
- **Principles of Congestion Control**
- **TCP Congestion Control**

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 application
- **Connectionless:**
  - No handshaking between UDP sender and receiver
  - Each UDP segment is handled independently of others
- **UDP use:**
  - Streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS (Domain Name System)
  - SNMP (Simple Network Management Protocol)
- **Reliable transfer over UDP:**
  - Must add reliability at application layer
  - Application-specific error recovery!

UDP: User Datagram Protocol (Cont.)

- **No connection establishment**
  - Eliminates a source of delay
- **Simple: no connection state at sender, receiver to maintain**
- **Small header size**
- **No congestion control: UDP can blast away as fast as desired**



UDP Segment Format

UDP Checksum

- **Used to detect errors (e.g. flipped bits) in transmitted data segment**
- **Sender:**
  - Treat segment contents, including the header fields, as sequence of 16-bit integers
  - Perform one's complement sum of segment contents, then take one's complement of that sum
  - Insert checksum value into UDP checksum field
- **Receiver:**
  - Compute one's complement sum of received segment (including checksum field)
  - Check if computed sum equals `0xFFFF`
    - YES - no error detected. But may have errors nonetheless? More later ….
    - NO - error detected

Checksum: Example

Example: add two 16-bit integers (a UDP checksum would add many more 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
Partial Sum	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	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

- **In one's complement addition, add overflow back into the partial sum to get the sum**
- **Take one's complement (invert) of sum to get the checksum**