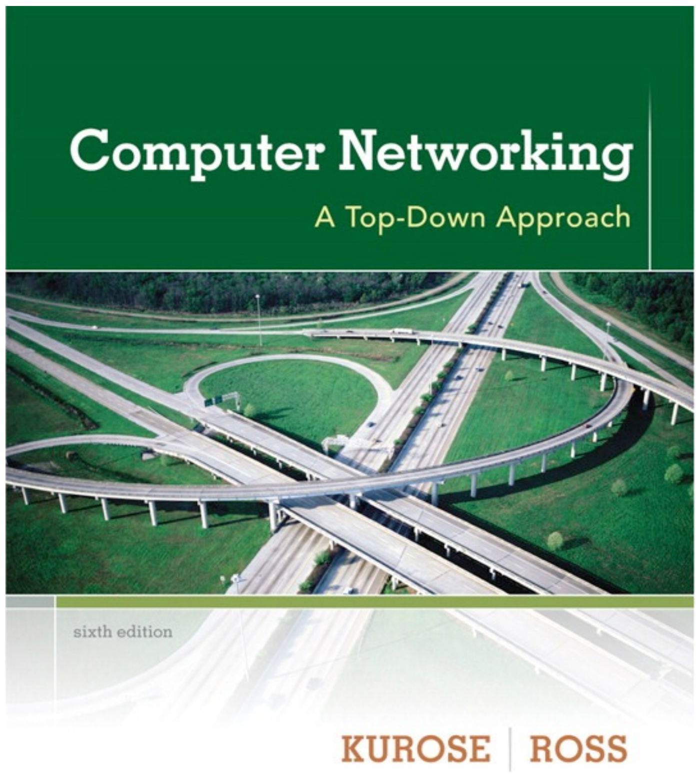
CN-Basic
L19

User Datagram Protocol

Dr. Ram P Rustagi
rprustagi@ksit.edu.in
http://www.rprustagi.com
https://www.youtube.com/rprustagi

Chapter 3
Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

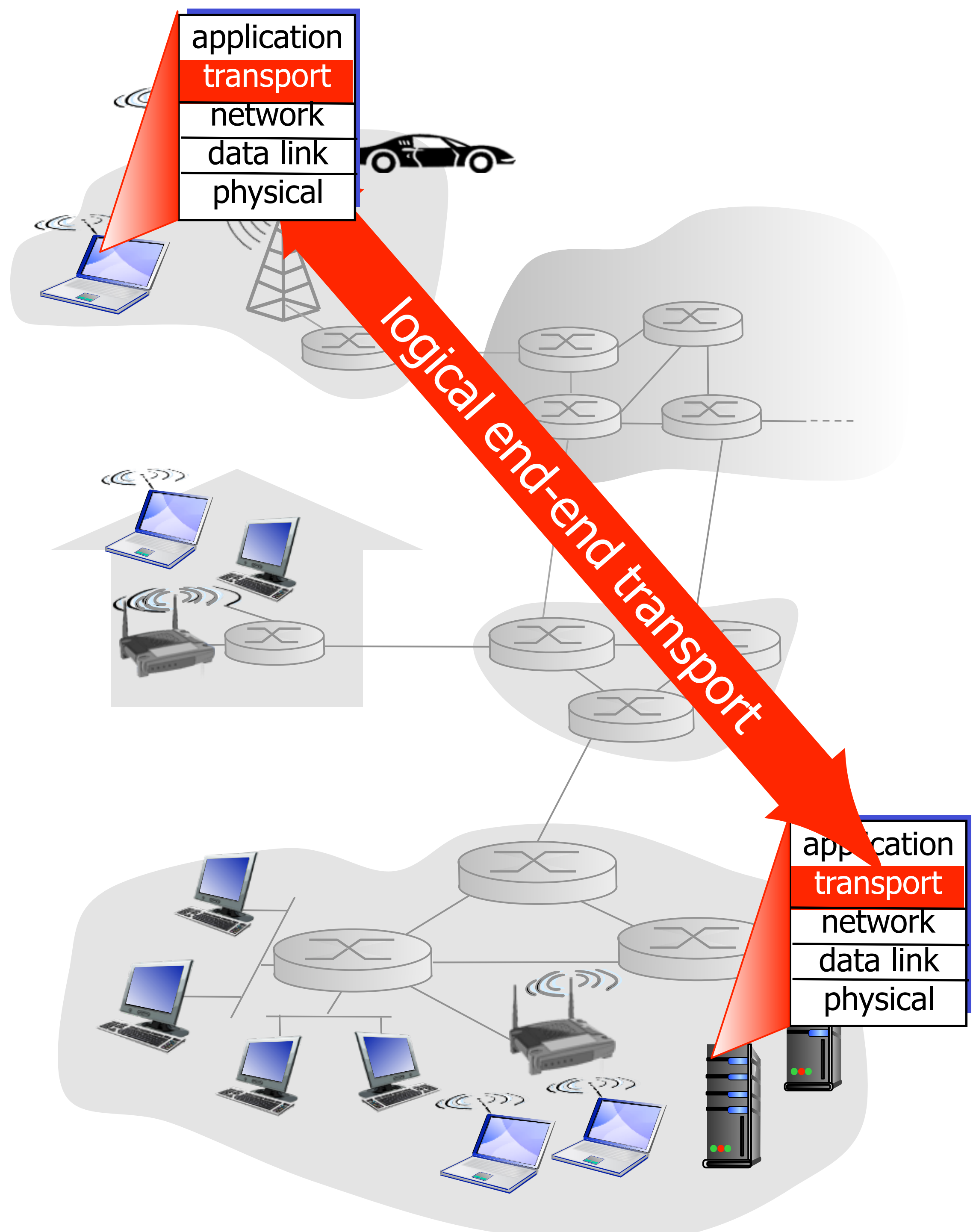
Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 3: Transport Layer

- Goals:
- 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

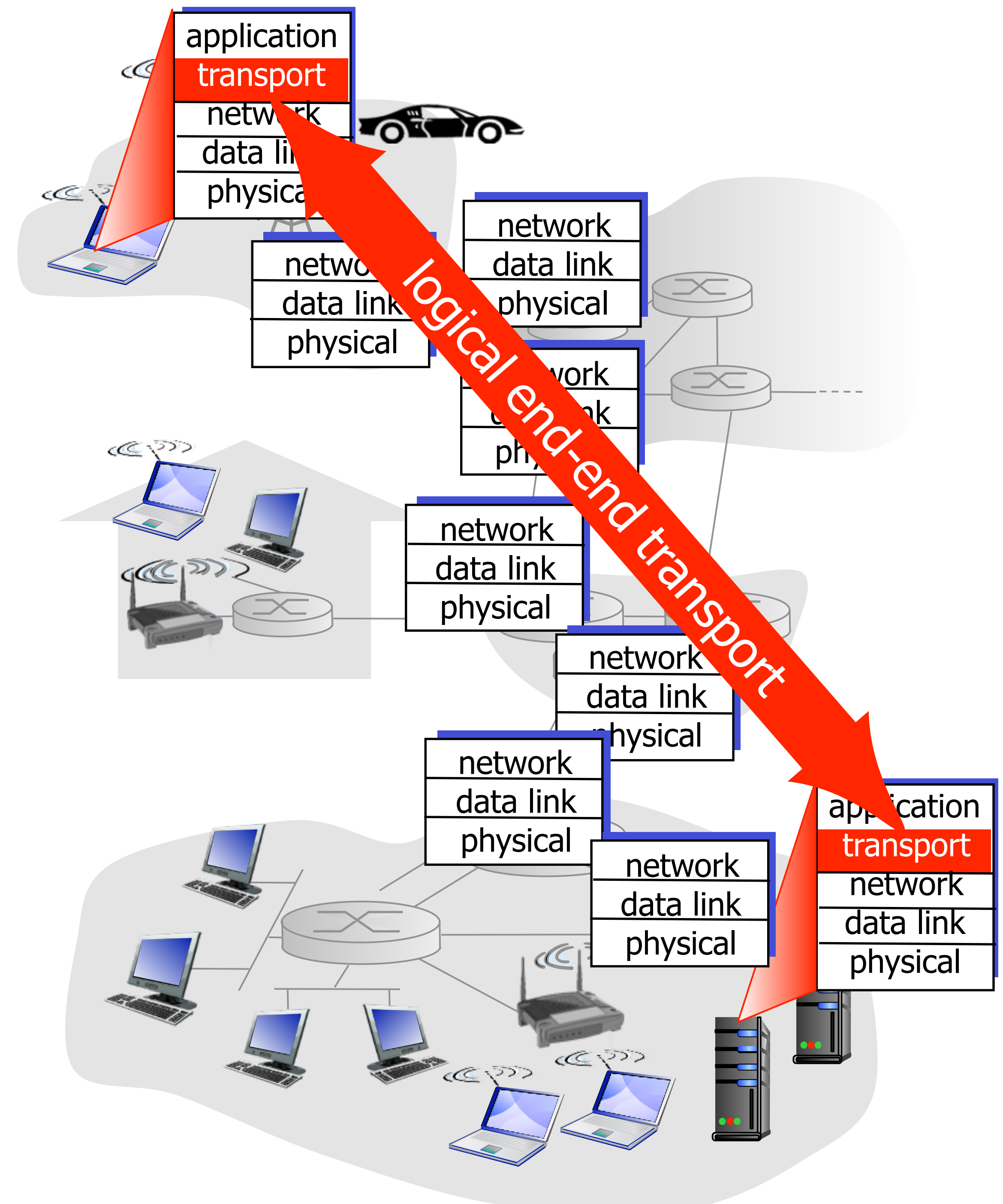
Transport services and protocols

- Provides *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



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 protocol

- How would you design it
- What would you like to achieve
  - At simplest level
    - Multiplex/de-multiplex
  - At advanced level
    - Reliable delivery i.e. Data integrity
      - Include error detection and retransmissions
    - Sequential delivery
      - Would need buffer
    - Detecting packet loss or duplicate delivery
    - Message boundaries
    - Security

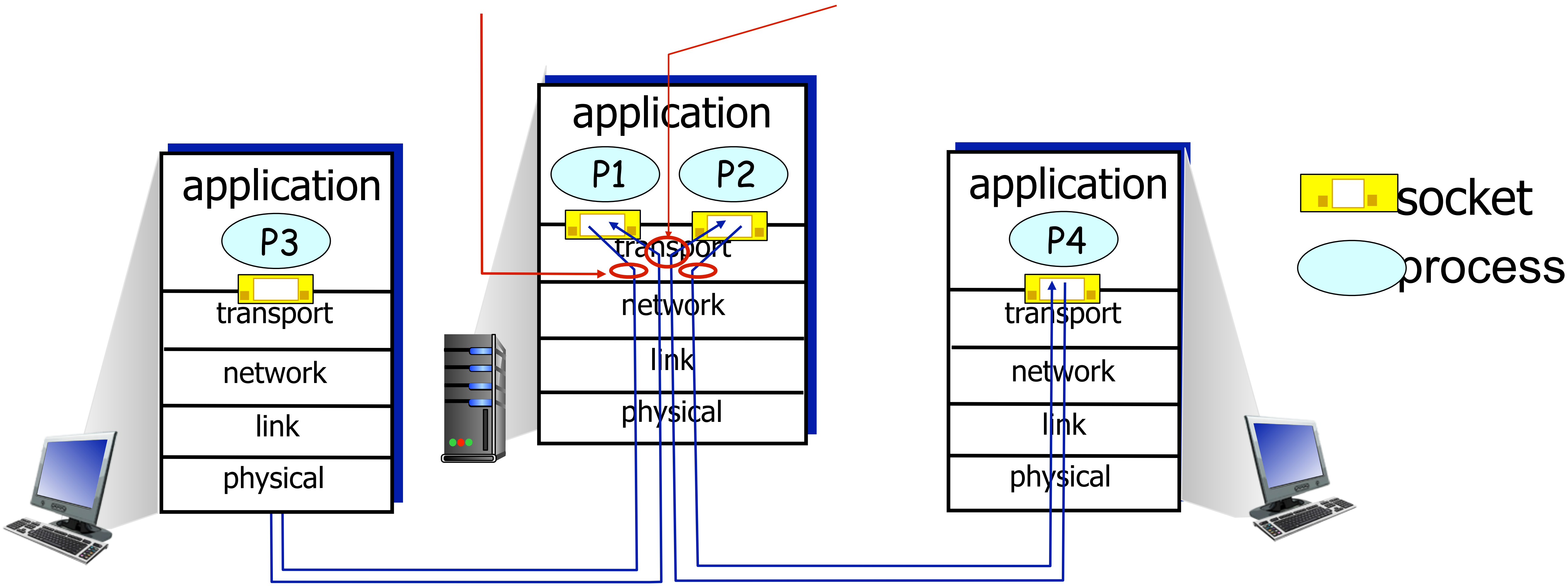
Multiplexing/demultiplexing

multiplexing at sender:

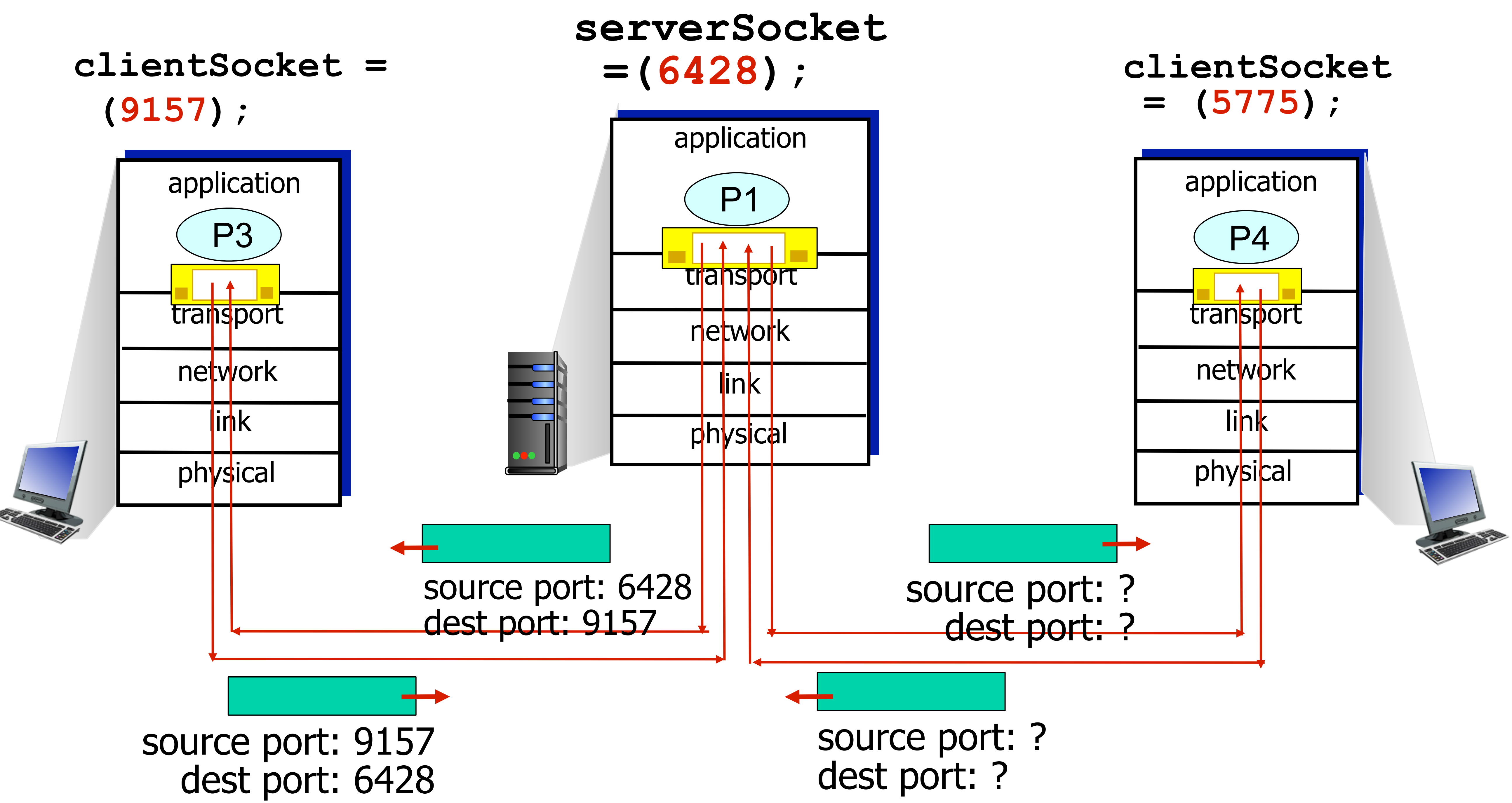
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket



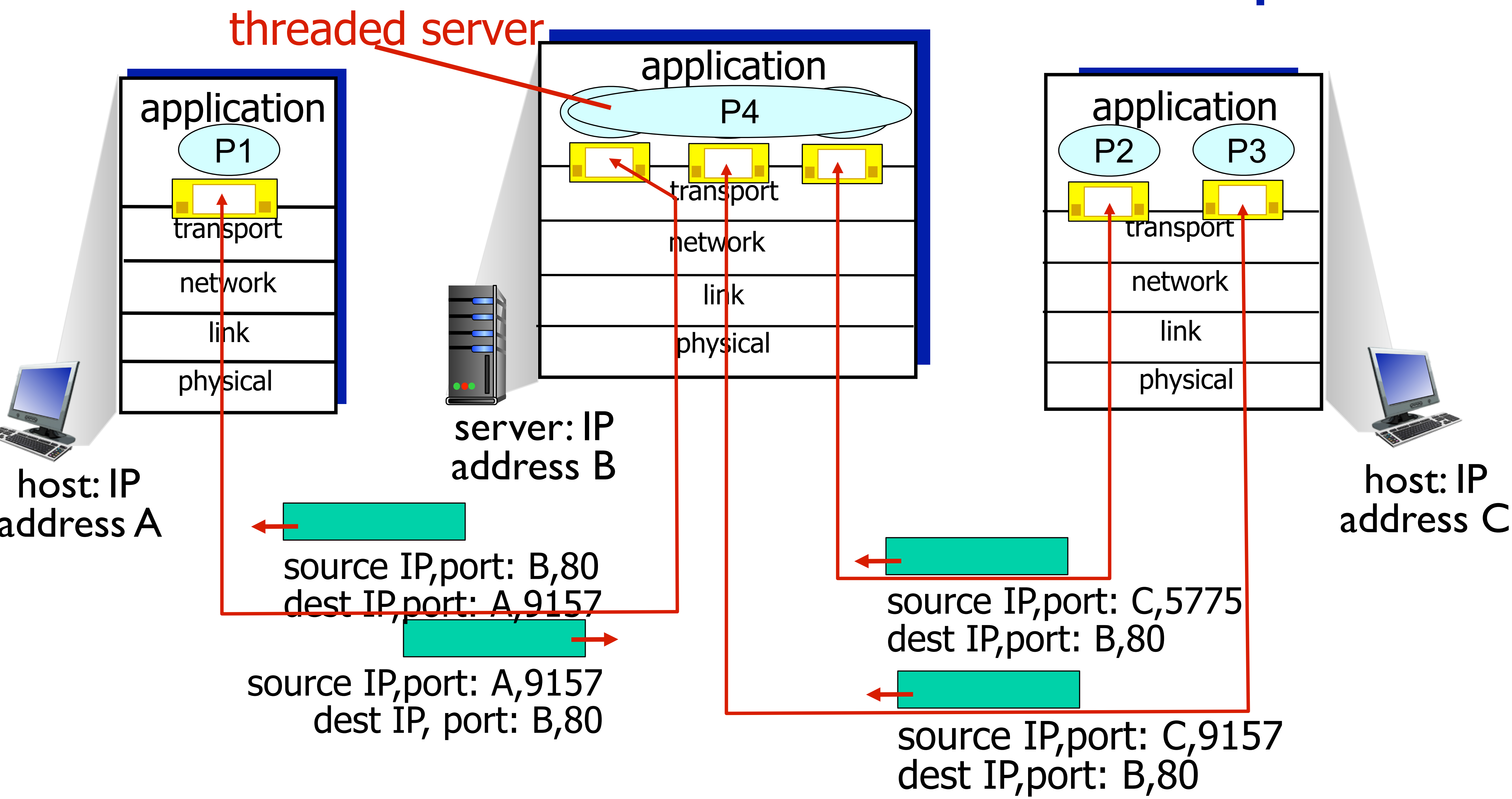
Connectionless demux: example



Connection-oriented demux

- Transport layer socket identified by 4(or 5)-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
  - (Protocol (TCP))
- 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



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to different sockets

Connectionless vs Connection oriented

- Connection less
  - Packets are not numbered
    - Packets may arrive out of order
  - No acknowledgement
    - Packets may be lost
  - No prior handshake
- Connection oriented
  - Setup, data xfer, and teardown phase
  - Provides Reliability
    - Ordered Delivery
  - Handles Error Control better

Reliability

- Reliable protocol
  - Needs error and flow control
    - Compels slower service
    - Require extra overheads
- Unreliable protocol
  - No extra overheads
- Reliability at data link layer
  - Provides error and flow control
  - Why do we need it at Transport layer?

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 used by:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP

- When to prefer UDP over TCP
  - Real time apps do not want congestion control
    - Some packet loss is okay
  - Connection handshake not required
    - No overhead and quick response e.g. DNS
    - analogy: SMS vs phone call, Alerts?
  - No connection state maintenance
    - OS has less resources overhead for TCP
    - Can support more UDP clients than TCP
  - Better utilisation efficiency
    - UPD overhead is 8 bytes vs 20 (min.) bytes of TCP

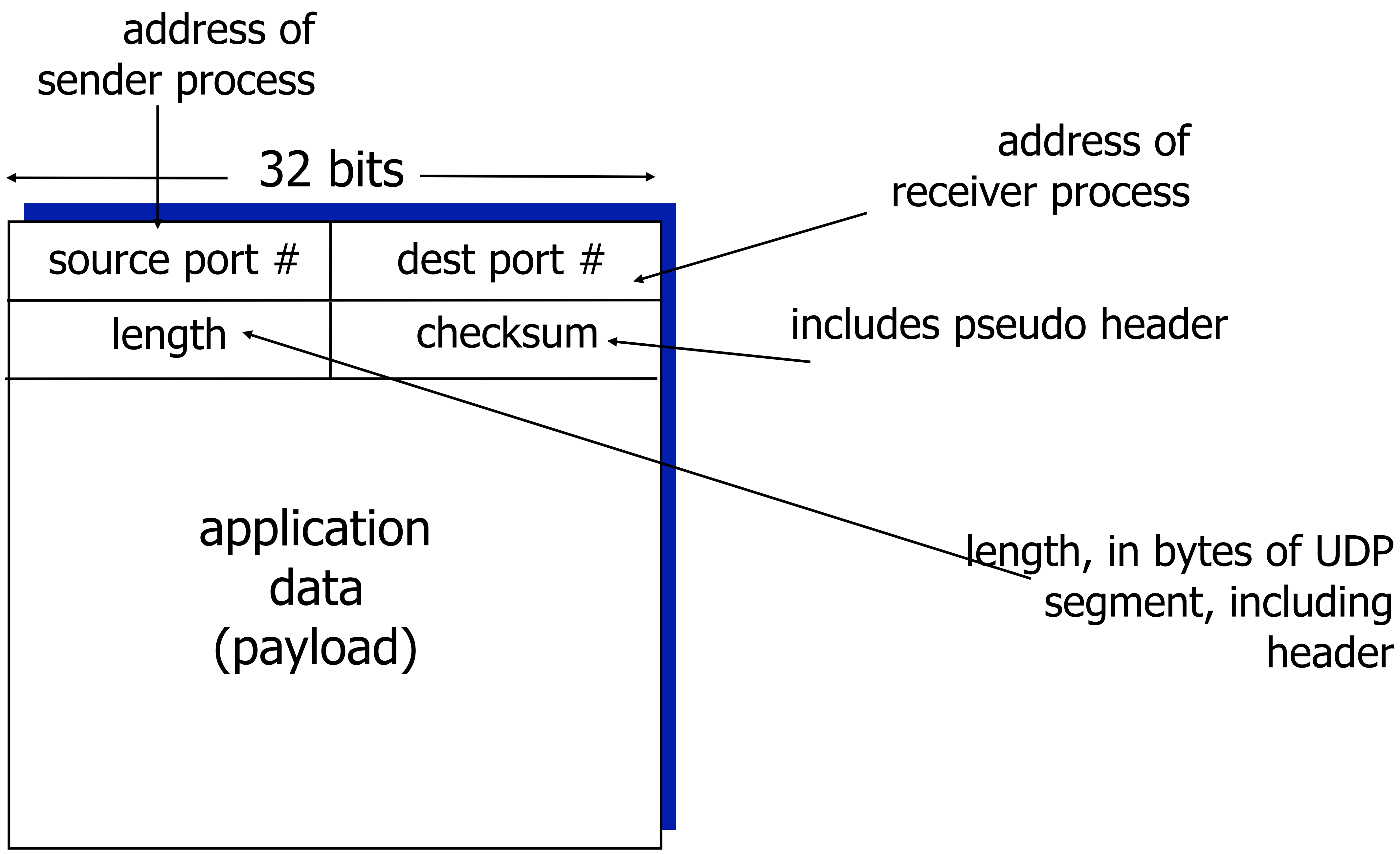
UDP

- How will you design a simple transport layer ?
- Just provides transport on top of IP
  - Multiplexing and demultiplexing
  - Little bit of error checking
  - No handshake
- Rest all has to be managed by application
  - Application practically talks to IP
- DNS uses UDP
  - What happens when query/response is lost?

UDP Headers

- What it should contain
  - Destination port number
    - Delivery to destination application process
  - Source port number. Why?
    - In case response needs to be returned back
    - Receiver identifies the sender's receiving point
  - Length
    - Each message can be of different length
  - Checksum
    - To detect if packet is corrupted

UDP: segment header



UDP segment format

Internet checksum: example

RFC 1071

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

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

Internet checksum: example

- Consider 3 words
  - `0110 0110 0110 0000   - 0x6660`
  - `0101 0101 0101 0101   - 0x5555`
  - `1000 1111 0000 1100   - 0x8F0C`
- --
- `10100 1010 1100 0001   -0x14AC1`
- Wrapping around the overflow bit makes it
  - `0100 1010 1100 0010   - 0x4AC2`
- 1's complement will be
  - `1011 0101 0011 1101   - 0xB53D`

UDP checksum

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

- sender:
  - treat segment contents, including header fields, as sequence of 16-bit integers
  - checksum: addition (one'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 ….

Pseudo header for checksum

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|          source address           |
+--------+--------+--------+--------+
|        destination address        |
+--------+--------+--------+--------+
|  zero  |protocol|   UDP length    |
+--------+--------+--------+--------+

+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|          data octets ...
+---------------- ...
```

Src: RFC 768

UDP headers

- Why pseudo headers ?
  - Protection against misrouted datagrams
- When computed checksum is zero
  - Transmitted as all ones
  - (equivalent to 0 in 1's complement)
  - All zero checksum implies no checksum generated
- Checksum
  - Uses pseudo header, UDP header and data
- Length: min value is 8 (why?)
- Data: padded if needed
  - to make a multiple of 16 bits octets

UDP Checksum Exercise

- Compute Checksum for the following case
  - Src IP: `10.30.26.1`, Dest IP: `10.30.26.11`
  - Src Port: `16384`, Dest port : `53`
  - Application Data: "`TESTING`"
    - `0x 54 45 53 54 49 4E 47`
  - Hint: Do you need padding?
    - Define Pseudo-header, UDP Protocol value is `0x11`
- Answer
  - `0A14` + `1001` + `0A14` + `110B` + `0011` + `000F` +
  - `4000` + `0035` + `000F` + `5445` + `5354` + `494E` +
  - `4700` = `??`
- What is checksum when data is
  - "UQUQUQTESTING" or
  - "INSTTEG"

Summary

- Transport Protocol
- Multiplexing and Demultiplexing
- Connectionless and Connection oriented
- UDP protocol
- UDP Checksum