# Transport Layer
# Part I

Mark Allman
*mallman@case.edu*

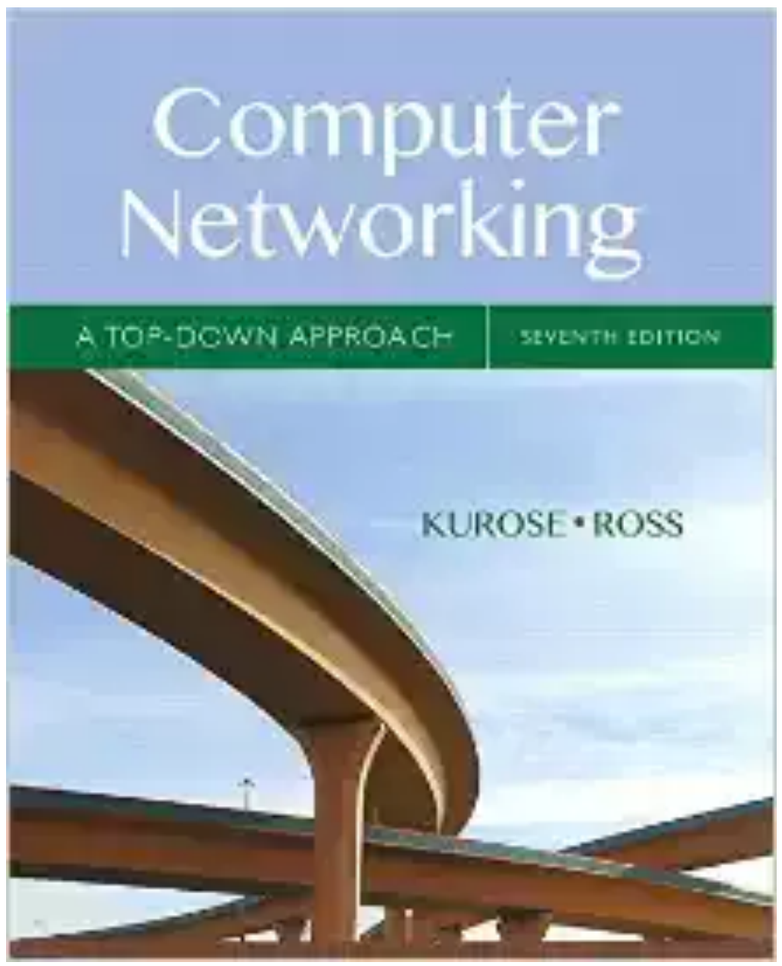
EECS 325/425
Fall 2018

*"Monday when the foreman calls time,*
*I already got Friday on my mind …"*

These slides are more-or-less directly from the slide set developed by Jim Kurose and Keith Ross for their book "Computer Networking: A Top Down Approach, 5th edition".

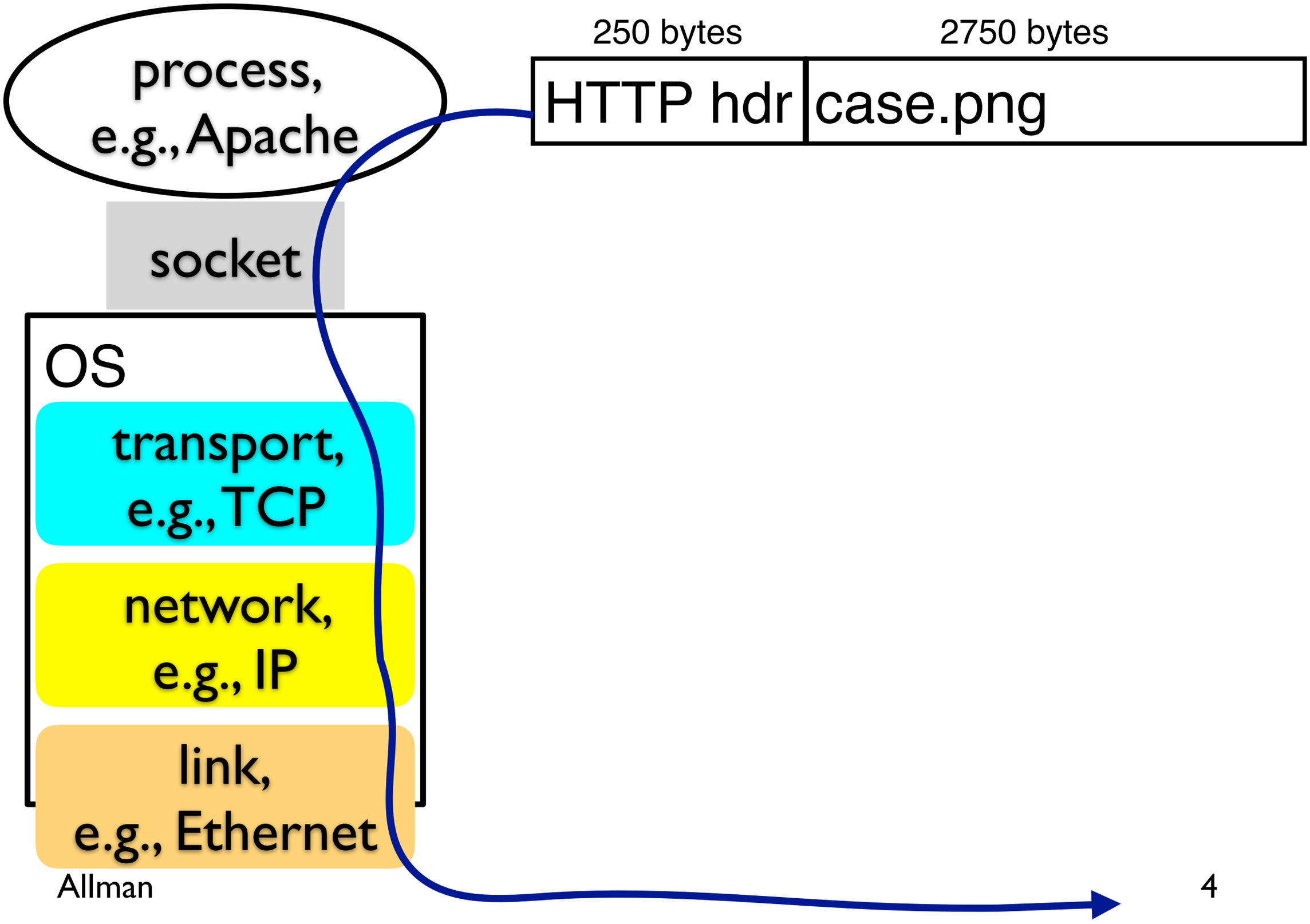
The slides have been lightly adapted for Mark Allman's EECS 325/425 Computer Networks class at Case Western Reserve University.

# Reading Along ...

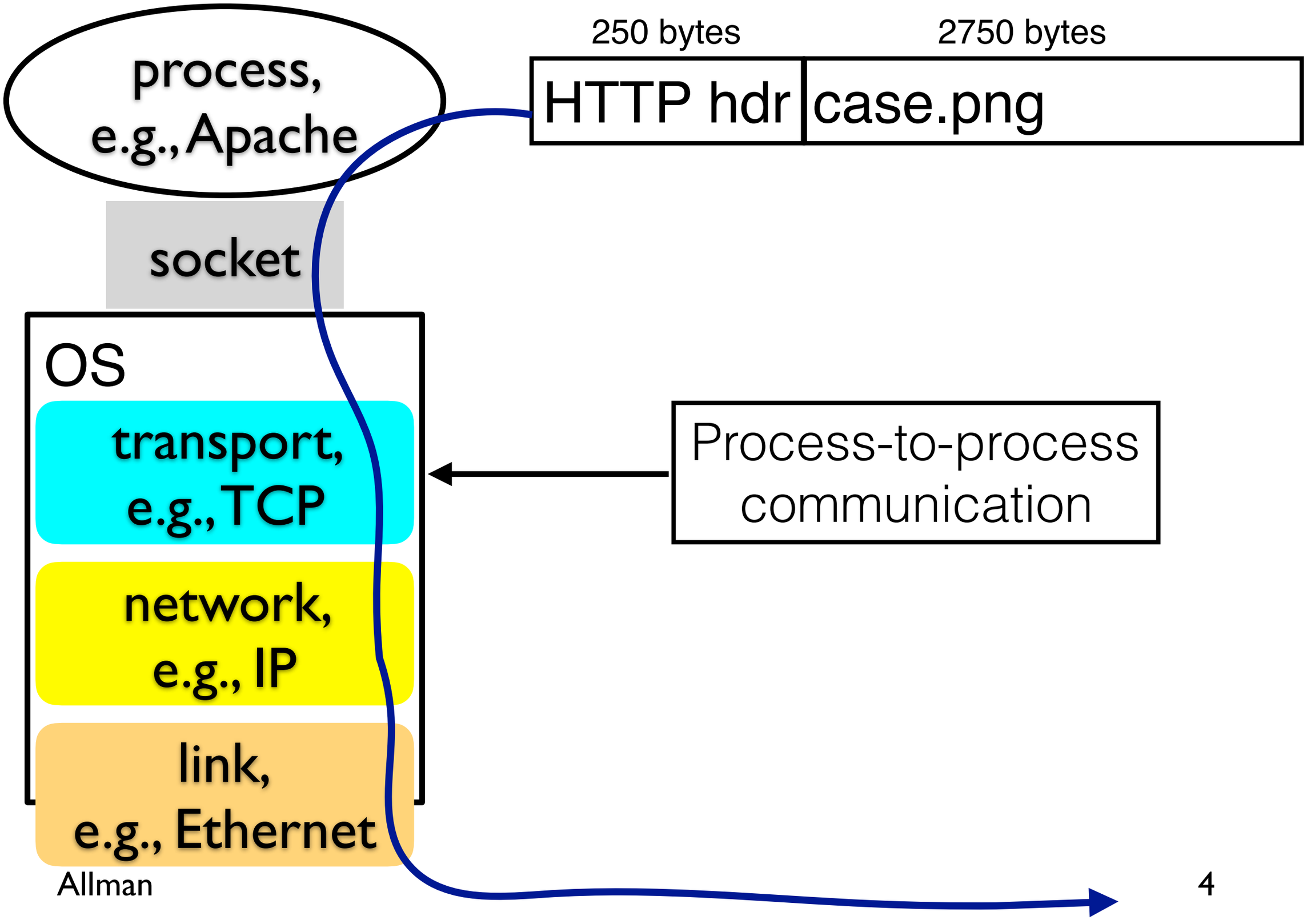



- Transport layer is chapter 3

# Data Flow


process,
e.g., Apache
250 bytes
2750 bytes
HTTP hdr
case.png
socket
OS
transport,
e.g., TCP
network,
e.g., IP
link,
e.g., Ethernet

# Data Flow

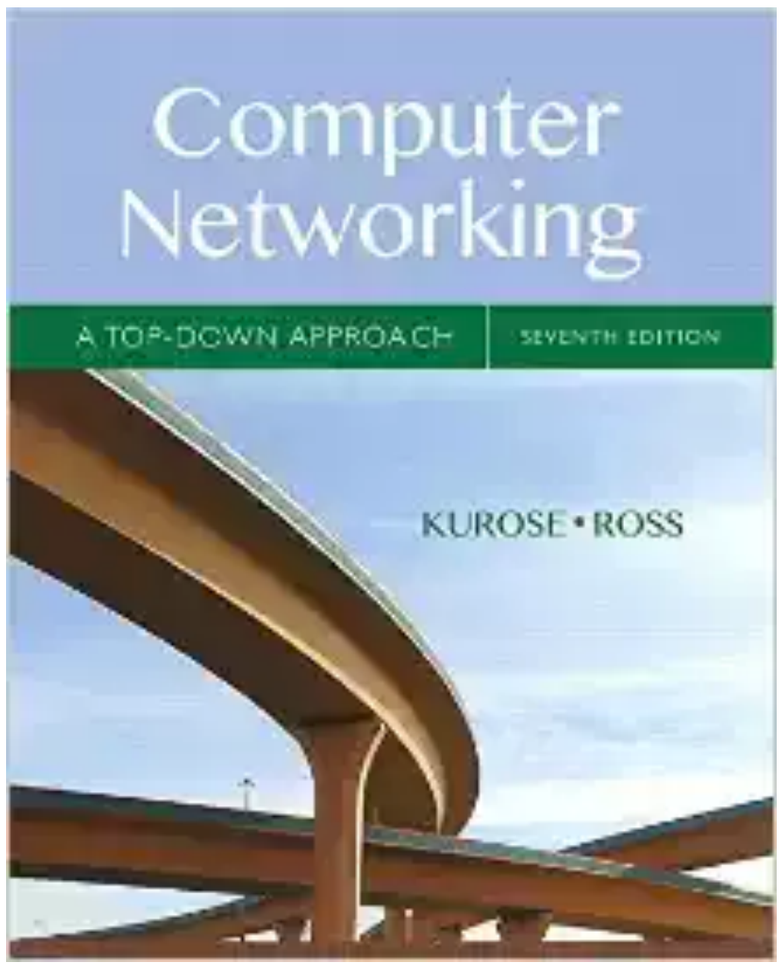

process,
e.g., Apache
250 bytes
2750 bytes
HTTP hdr
case.png
socket
OS
transport,
e.g., TCP
network,
e.g., IP
link,
e.g., Ethernet
Process-to-process
communication

# Transport Layer

Our goals:

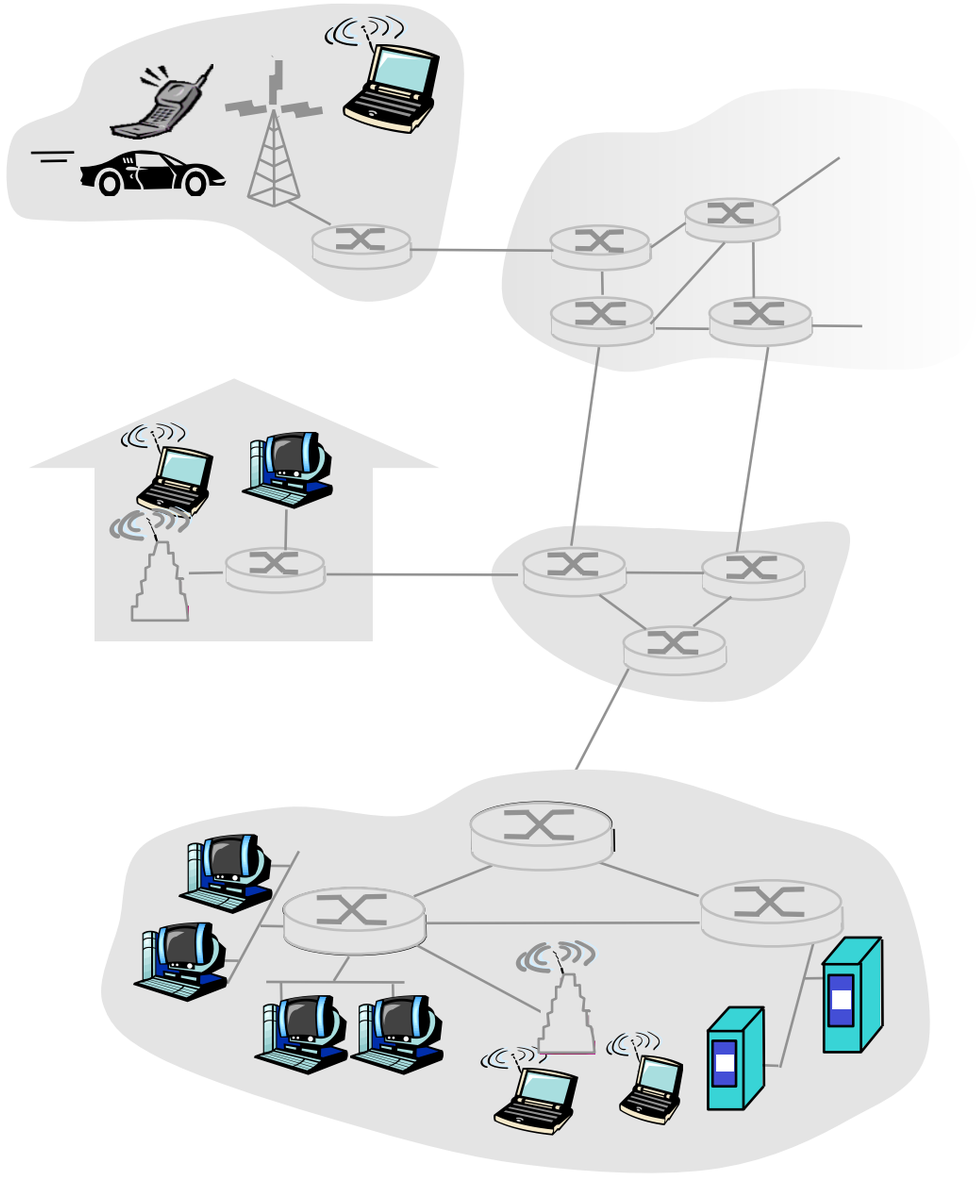
- understand principles behind transport layer services:
  - multiplexing/ demultiplexing
  - reliable data transfer
  - flow control
  - congestion control (later)
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport

# Reading Along ...



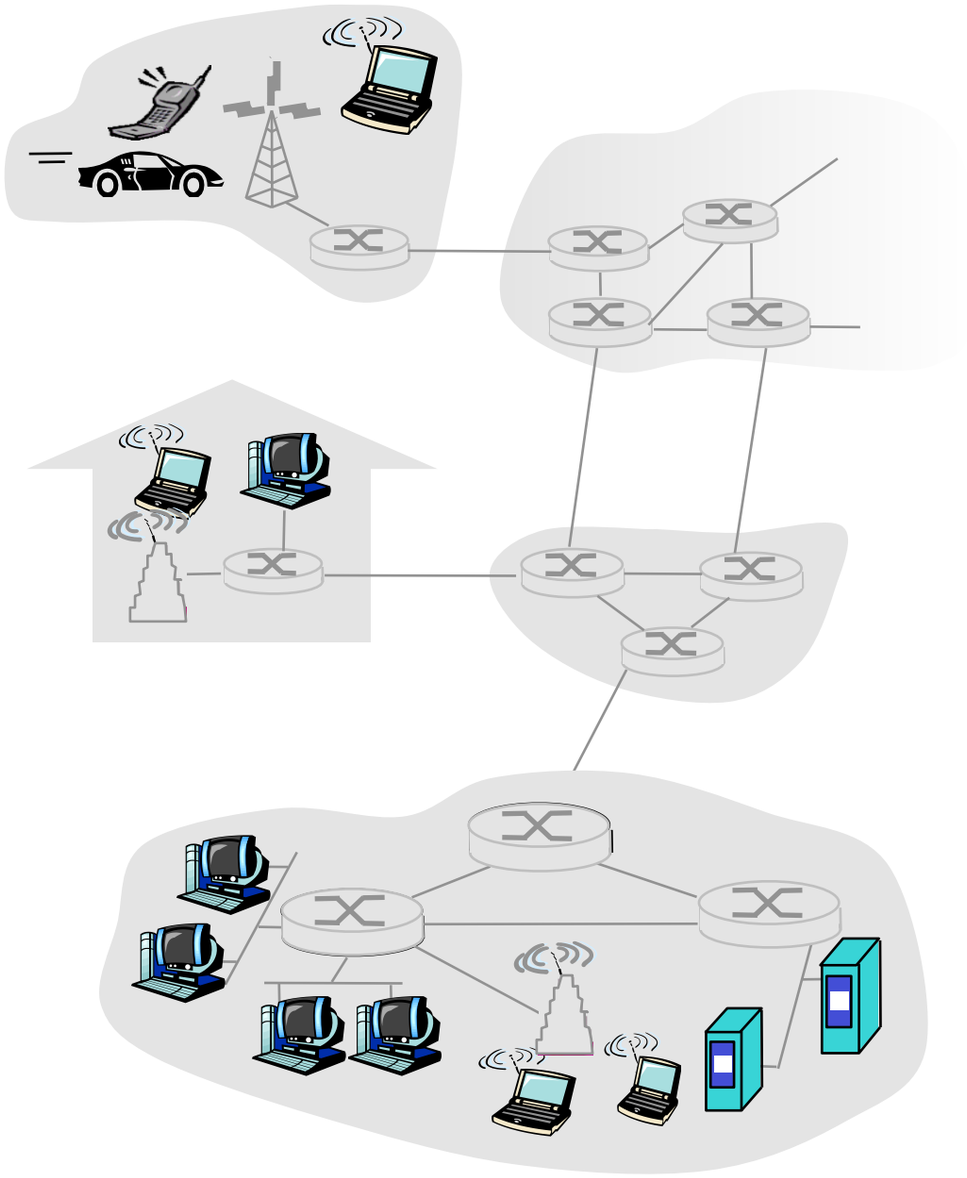

- 3.1: Transport Layer Services

# Transport services and protocols



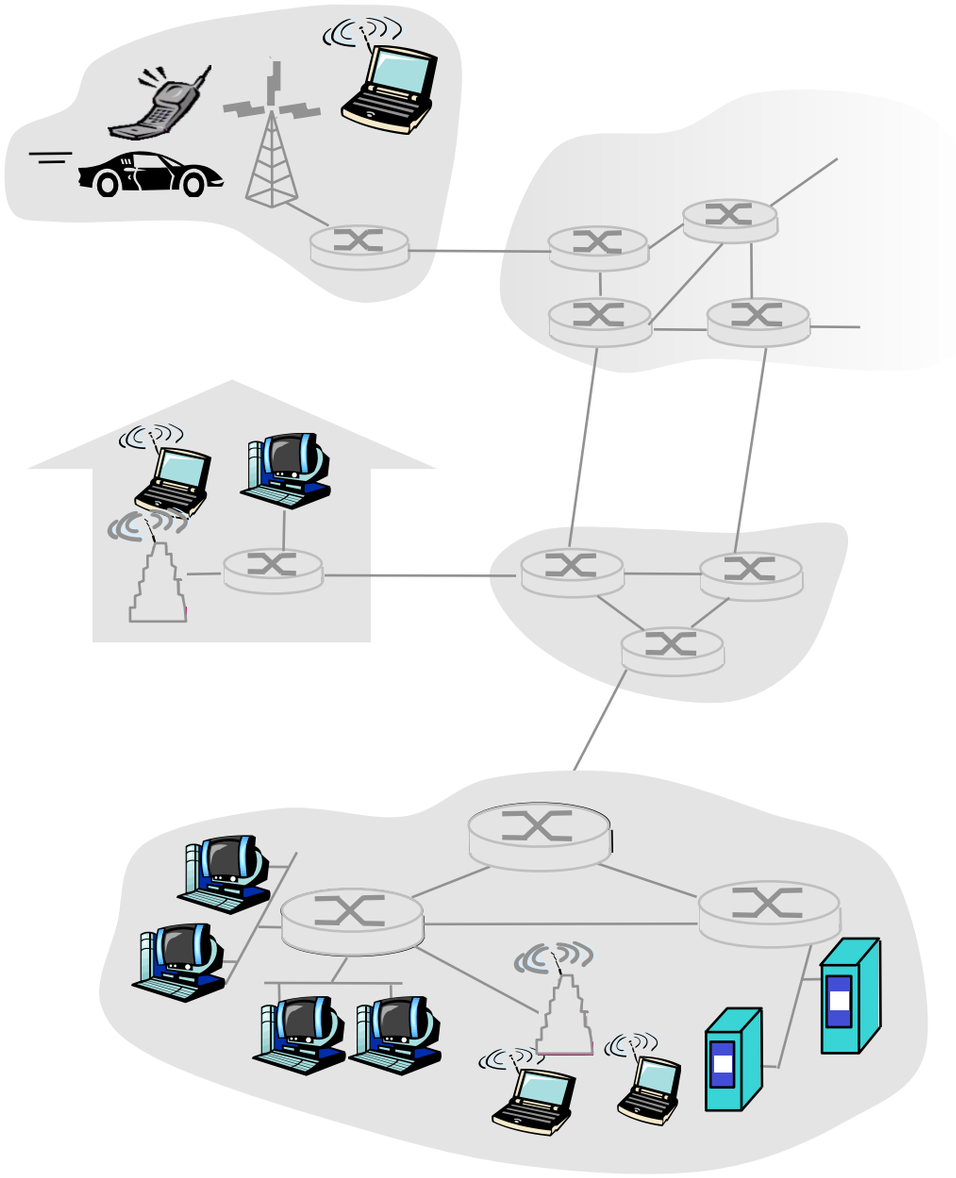
# Transport services and protocols

- provide logical communication between app processes running on different hosts



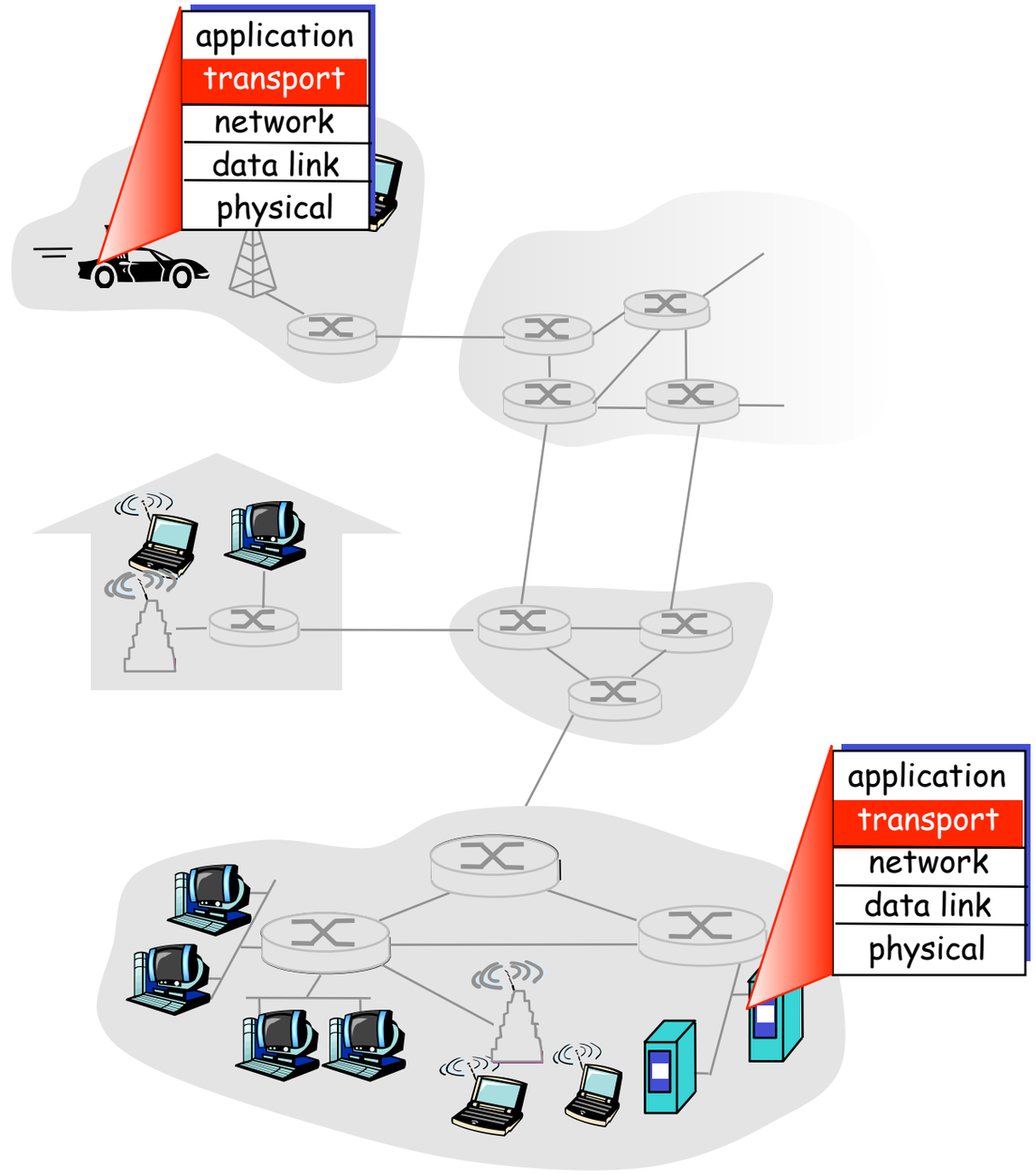
# Transport services and protocols

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems



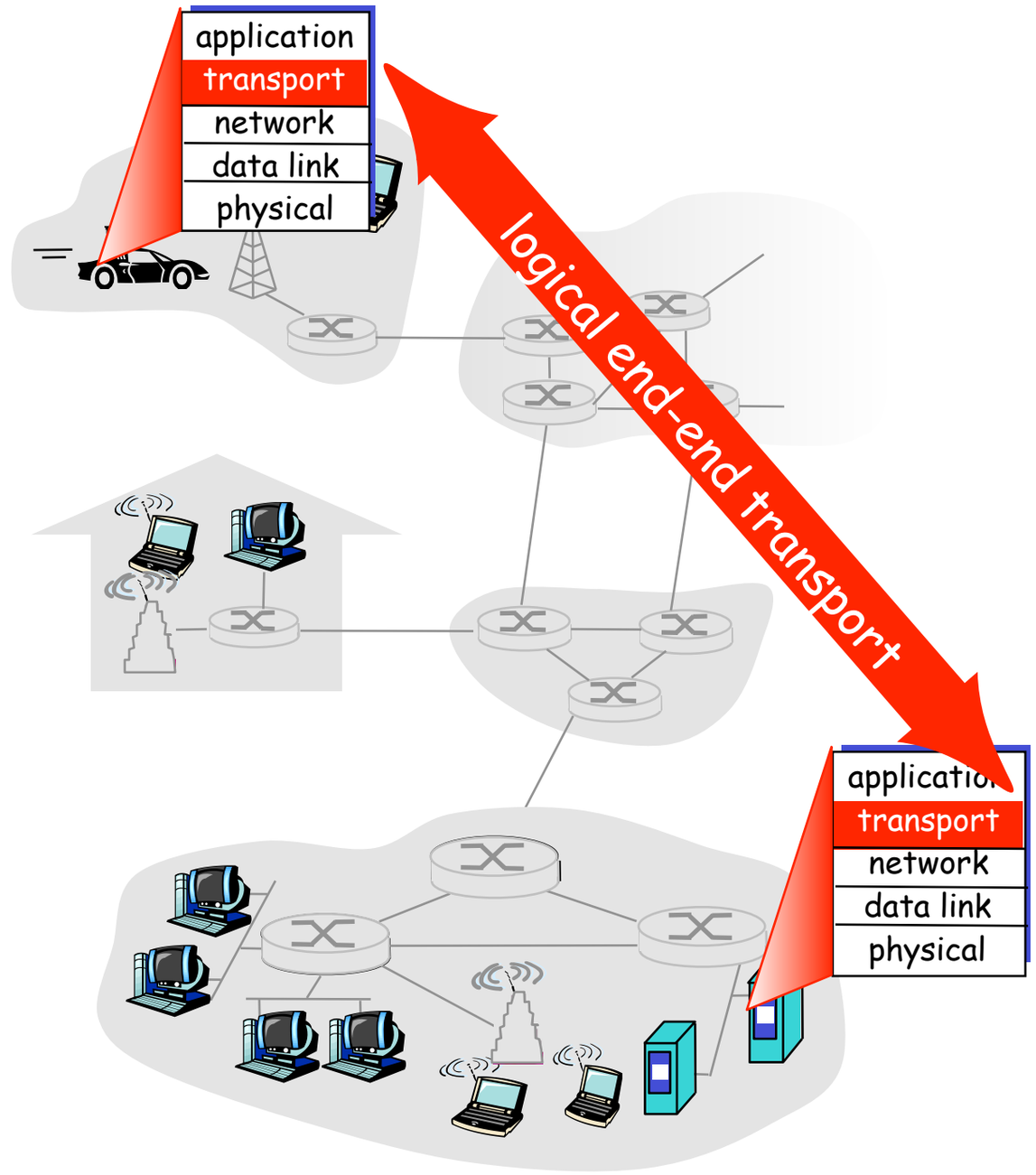
# Transport services and protocols

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems



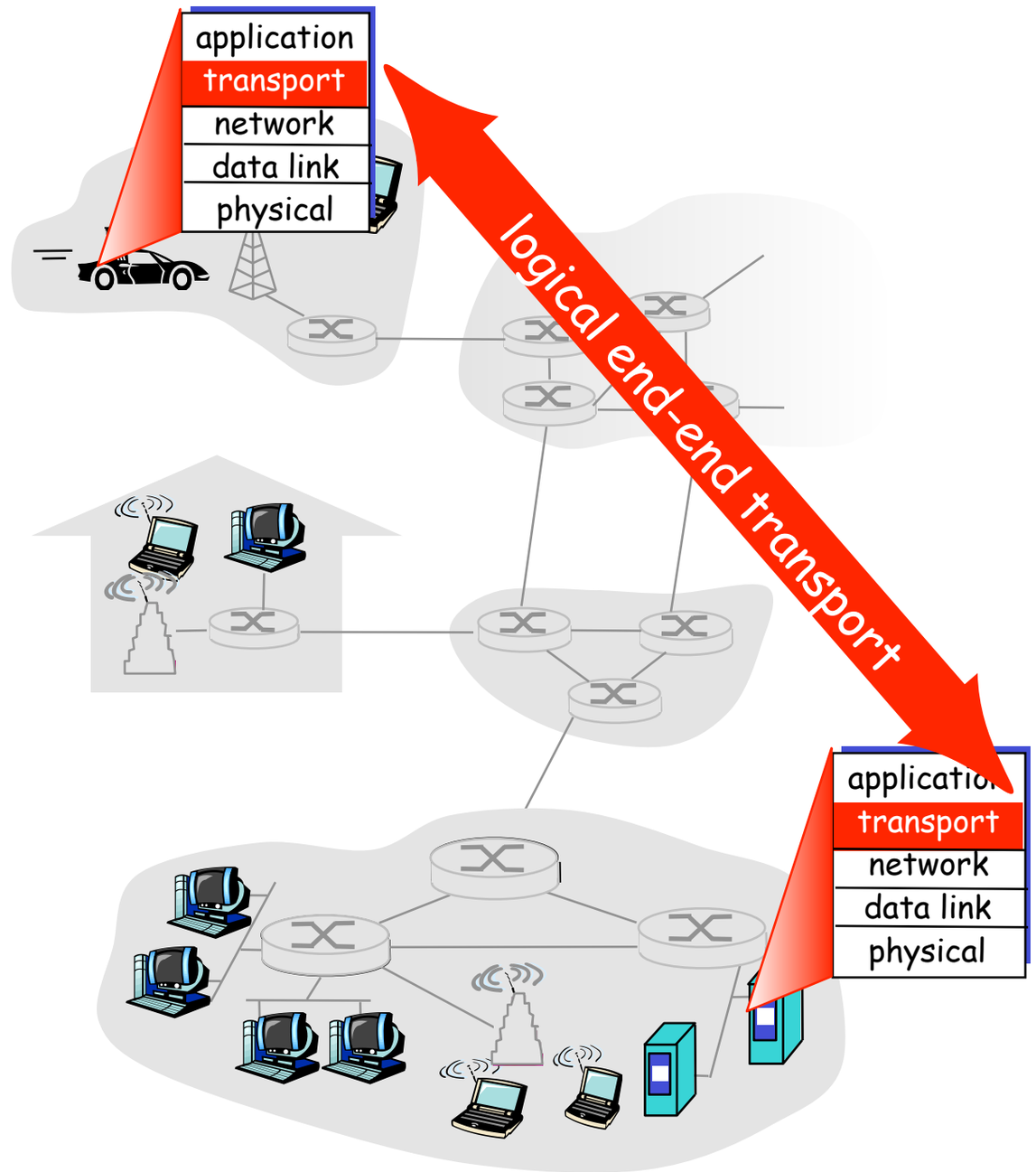
# Transport services and protocols

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems



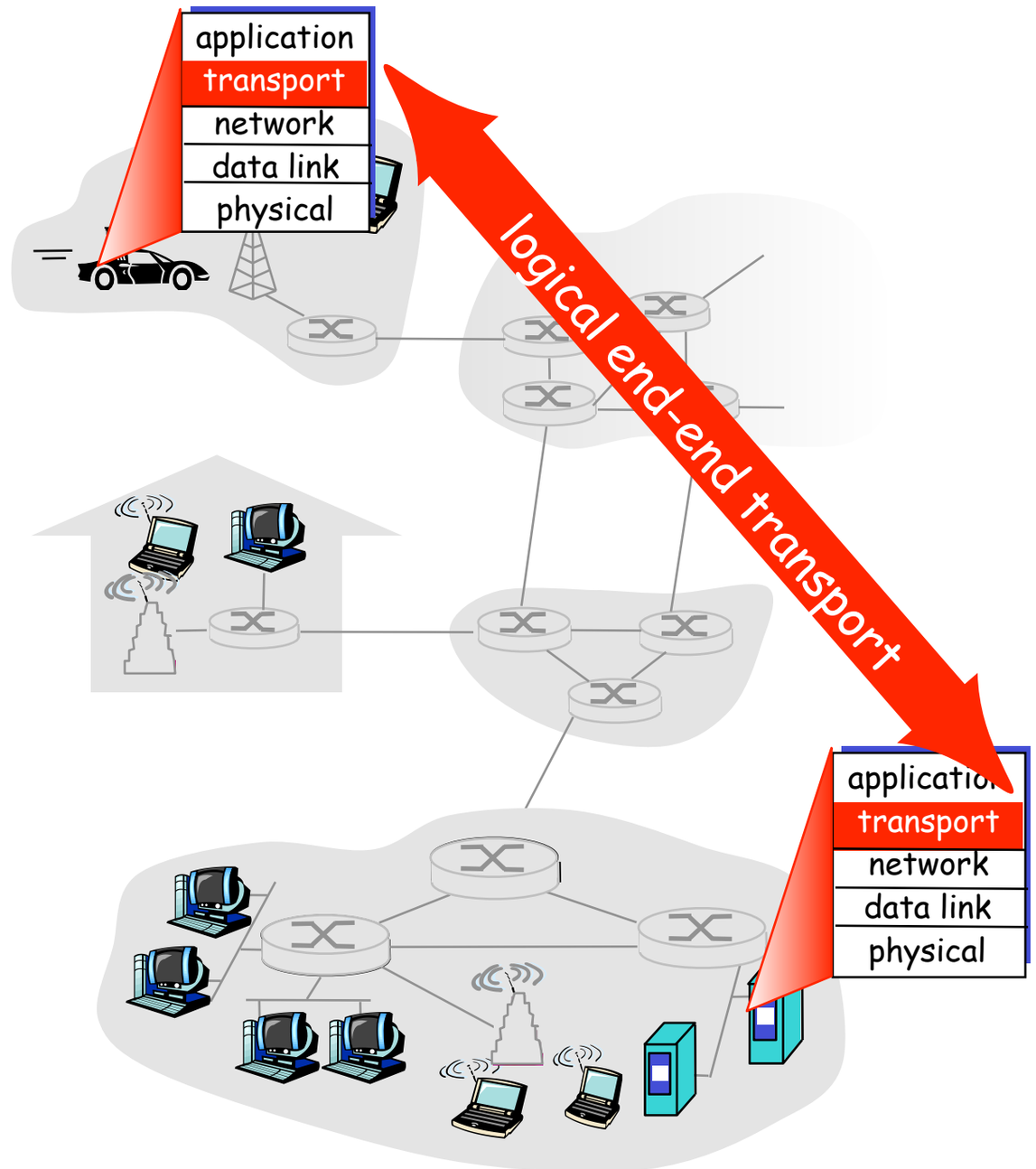
# Transport services and protocols

- 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



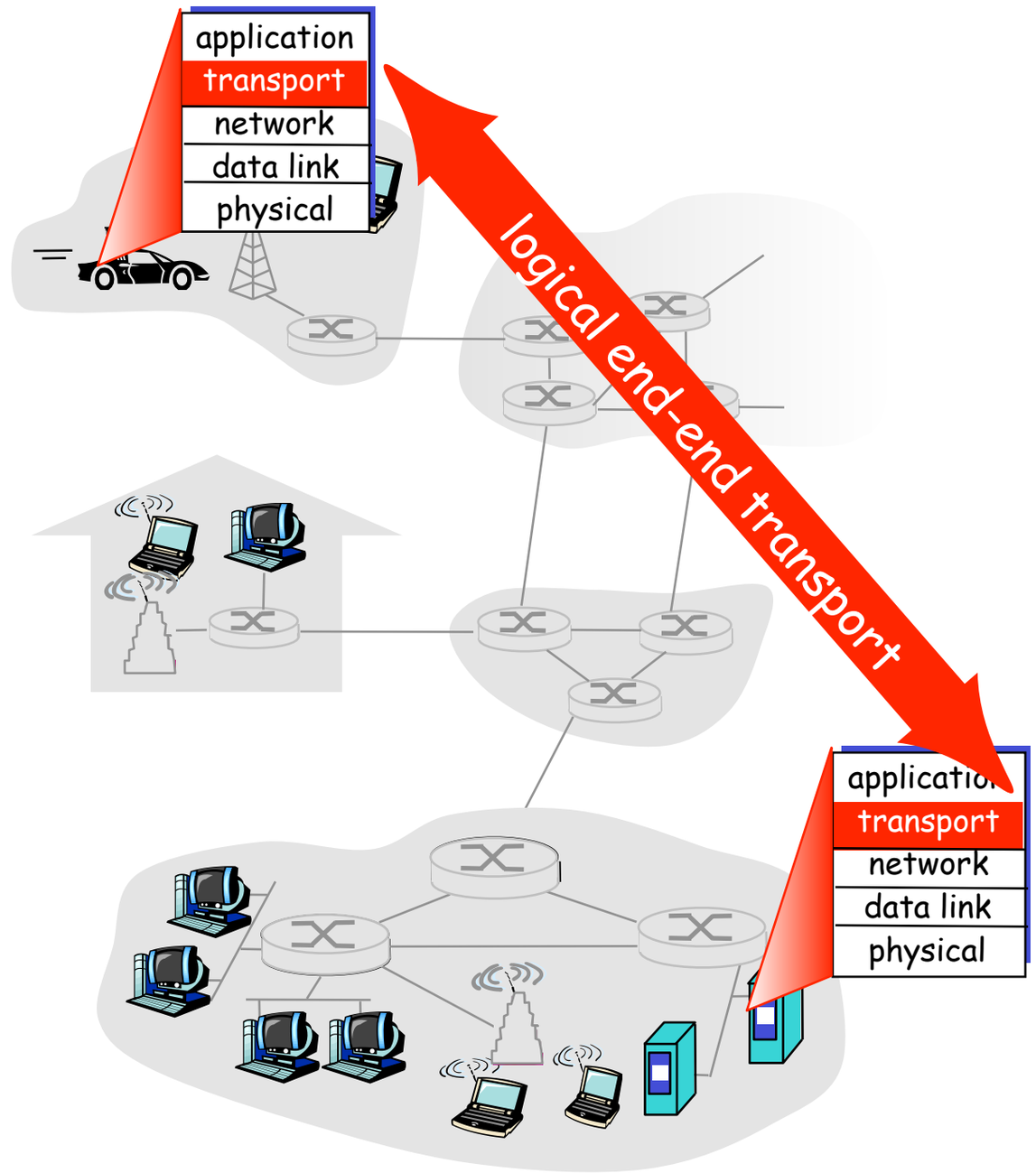
# Transport services and protocols

- 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



# 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



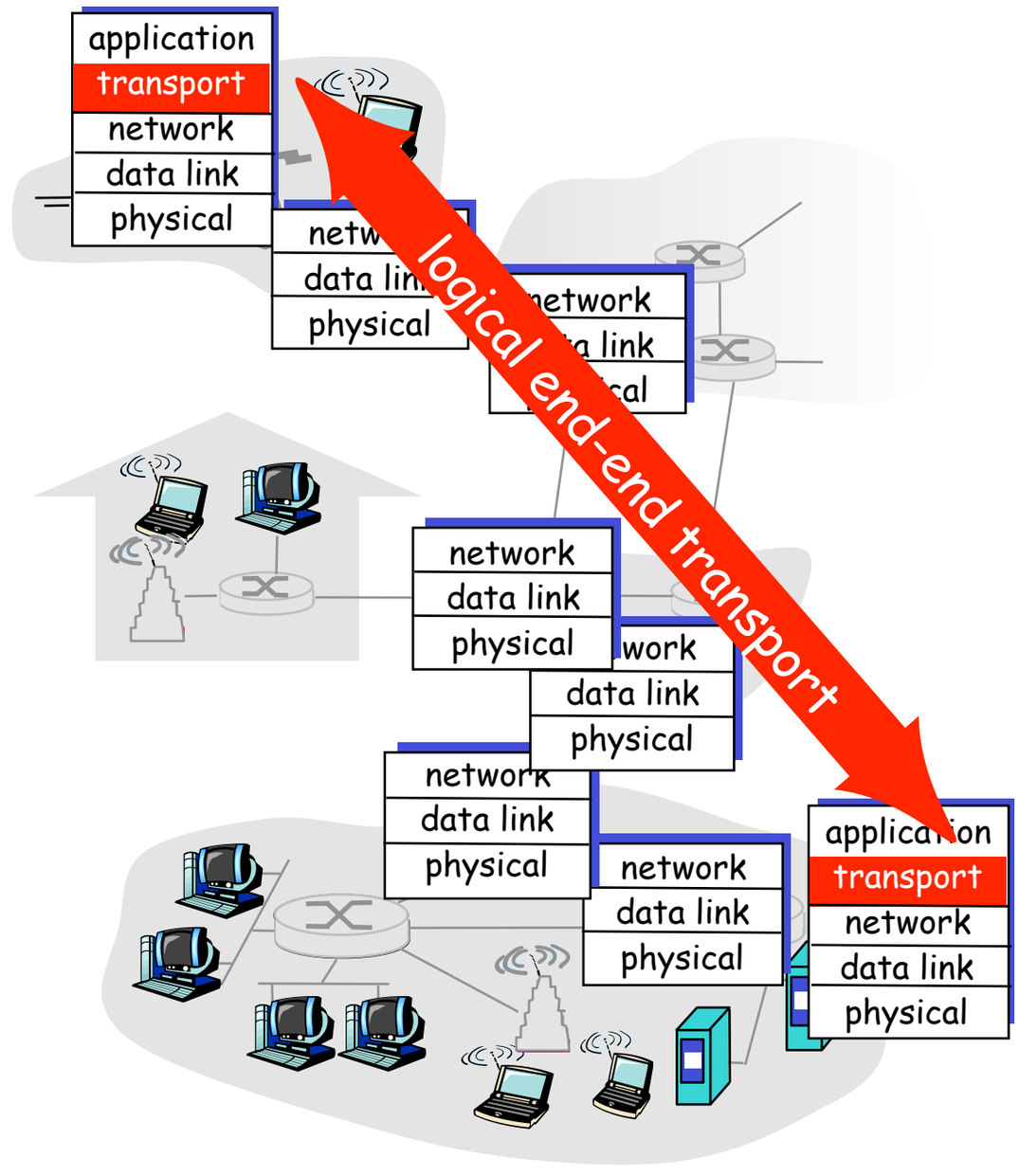
# Transport vs. network layer

# Transport vs. network layer

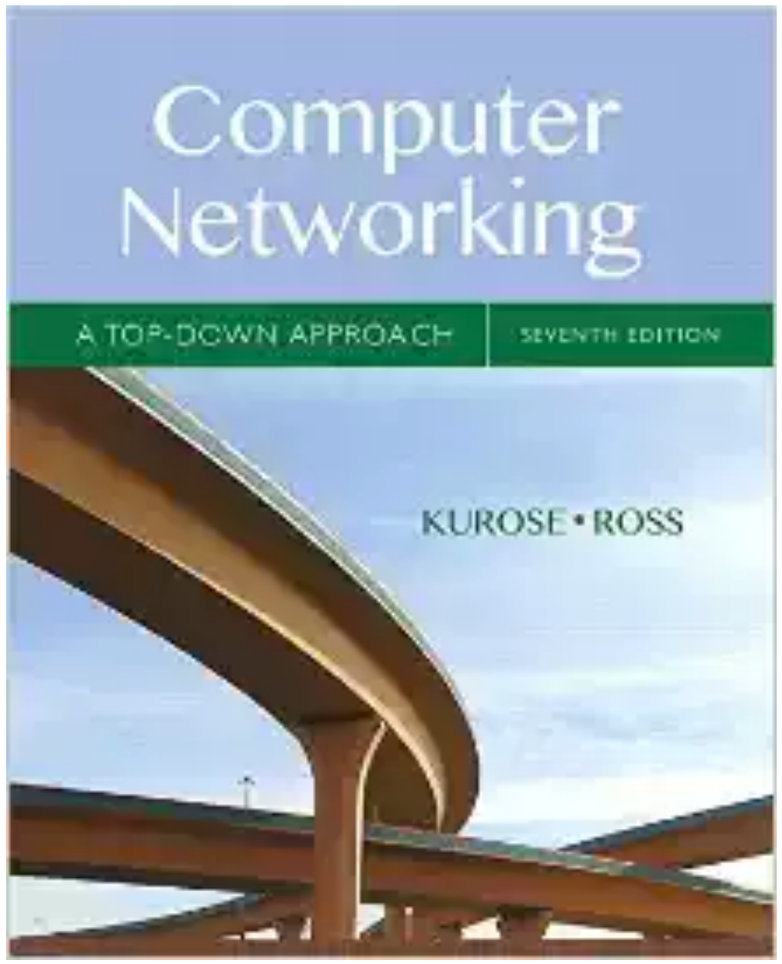
- network layer: logical communication between hosts
- transport layer: logical communication between processes
  - relies on, enhances, network layer services

# 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



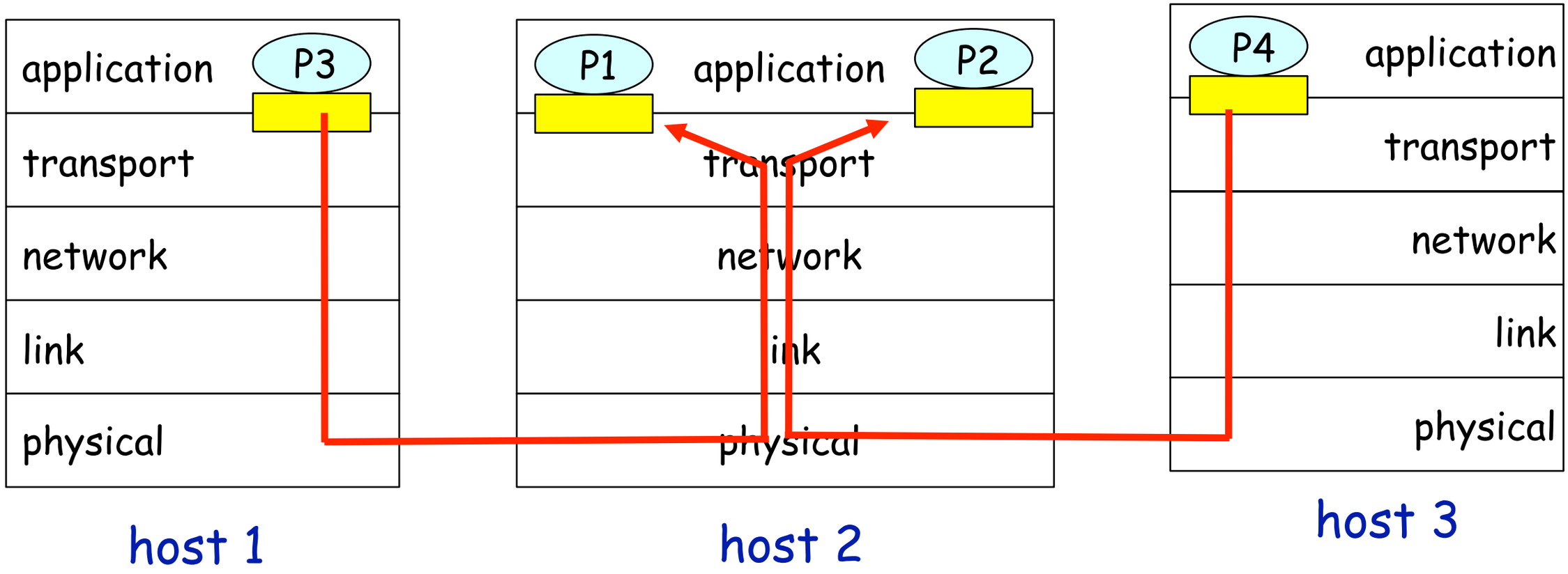
# Reading Along ...




- 3.2: Multiplexing and Demultiplexing

# Multiplexing/demultiplexing

= socket = process

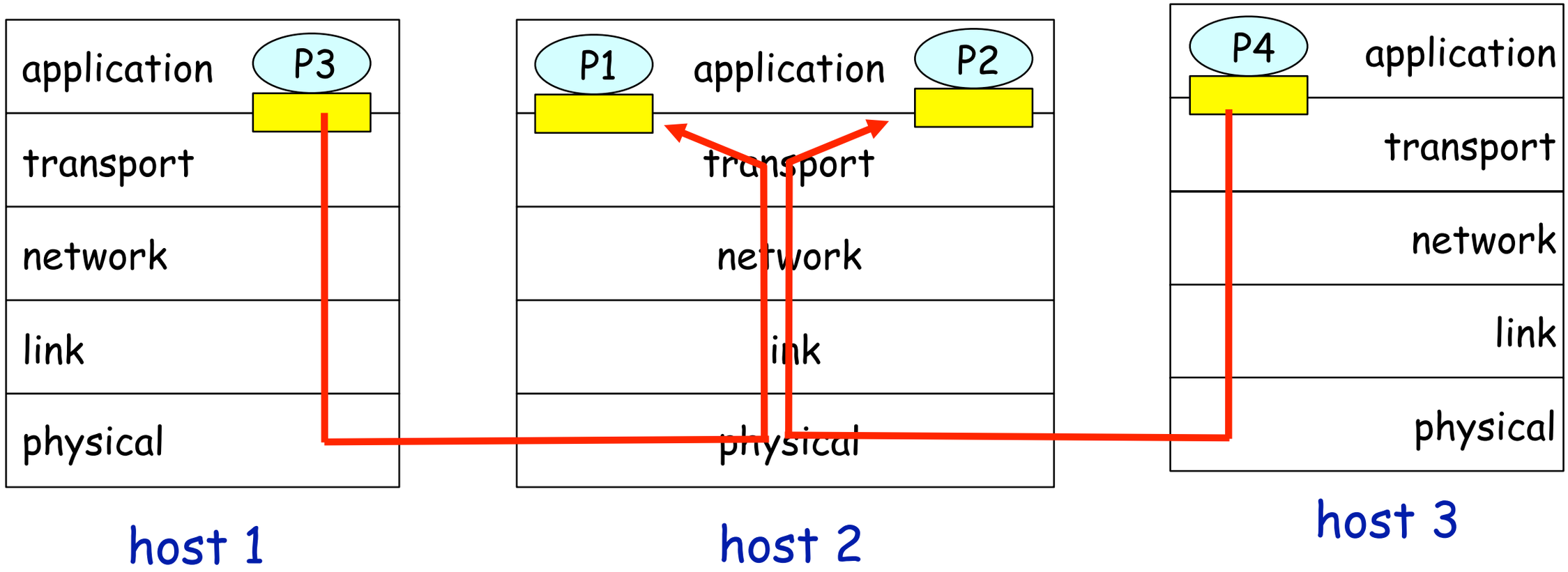


# Multiplexing/demultiplexing

**Multiplexing at send host:**

collecting data from sockets, enveloping data with header (later used for demultiplexing)

= socket = process



# Multiplexing/demultiplexing

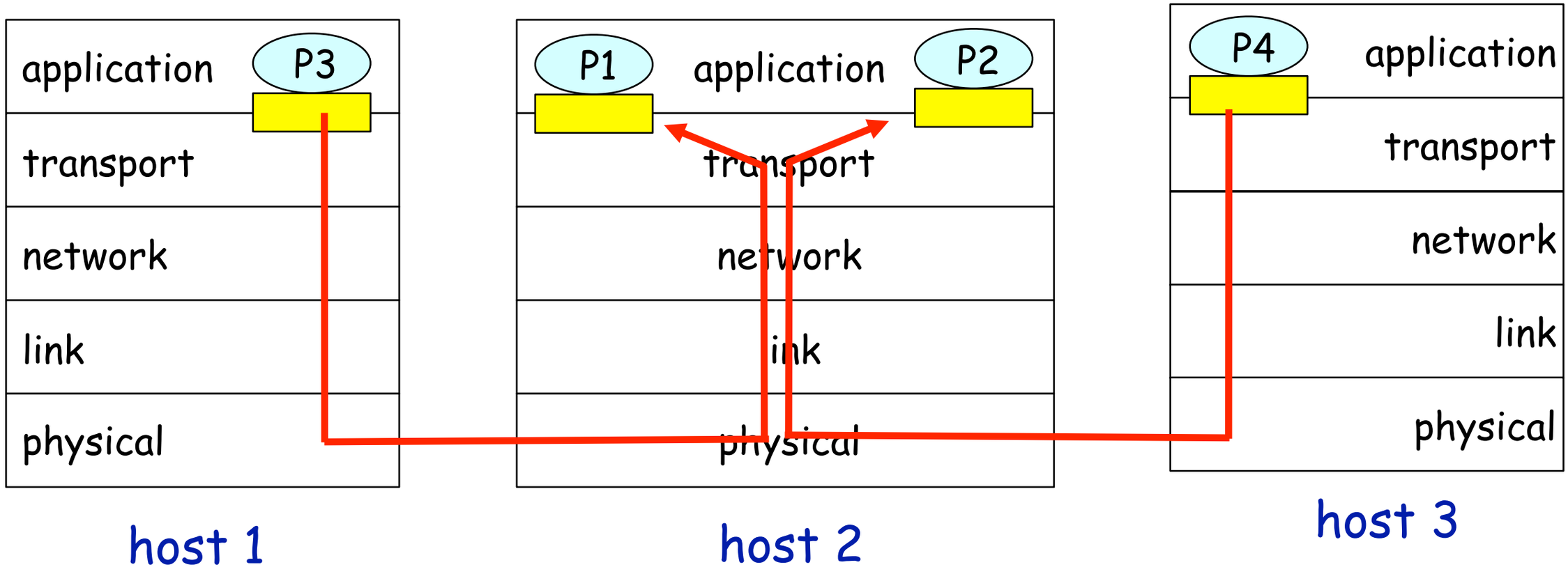
**Demultiplexing at rcv host:**

delivering received segments to correct socket

**Multiplexing at send host:**

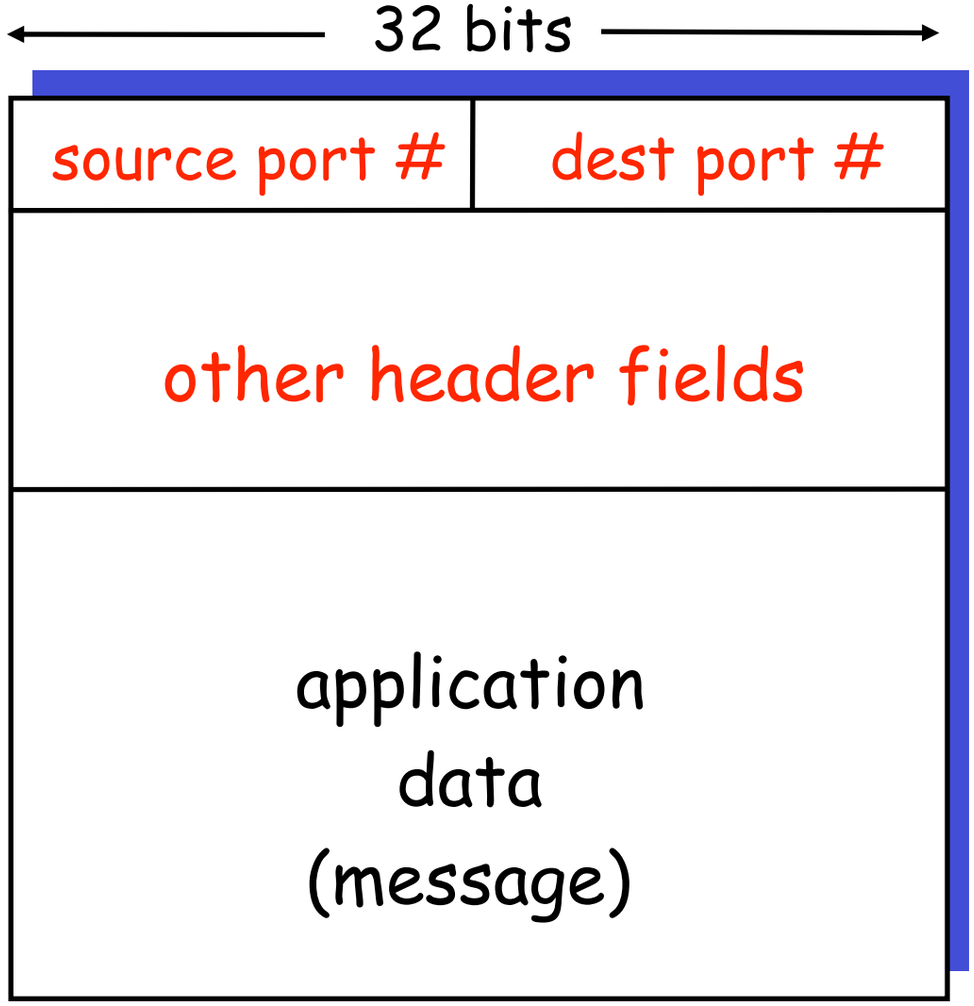
collecting data from sockets, enveloping data with header (later used for demultiplexing)

= socket = process



# 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, IP protocol number & port numbers to direct segment to appropriate socket




TCP/UDP segment format
(red = transport header)