CSCI3150 Introduction to Operating Systems

Lecture 17: Networking

Hong Xu

https://github.com/henryhxu/CSCI3150

Agenda

- Layering
- Transport layer

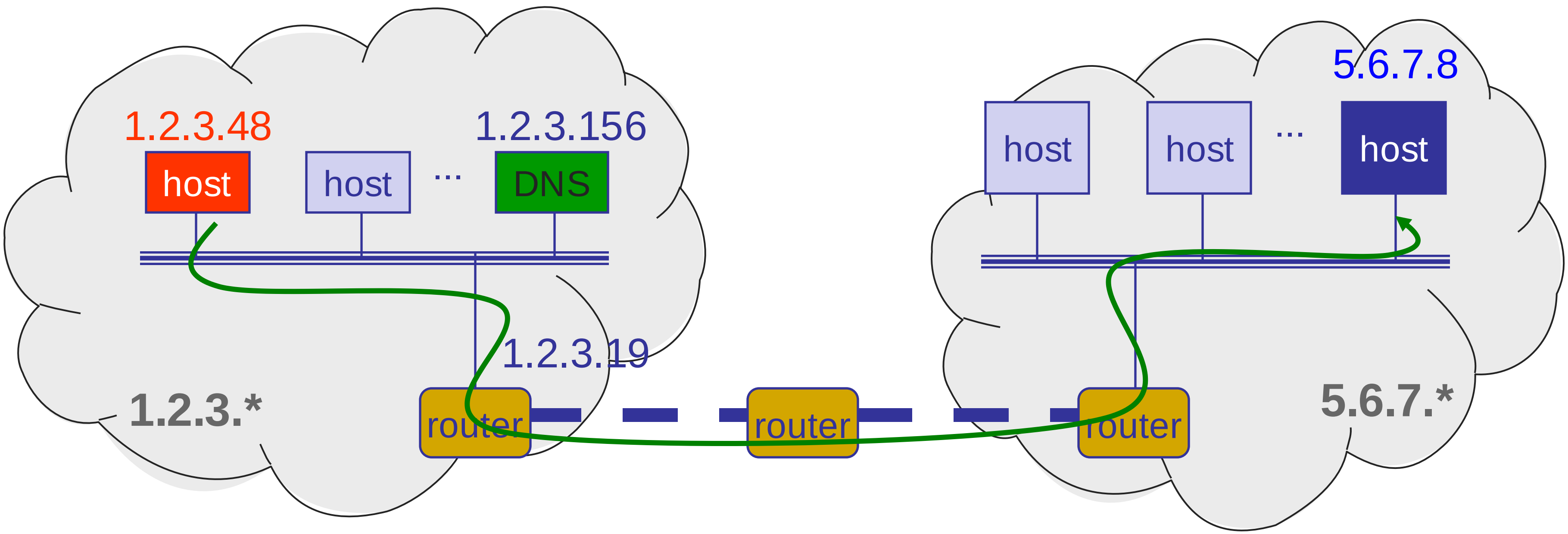
- This is just a rough introduction to the topic

Layering

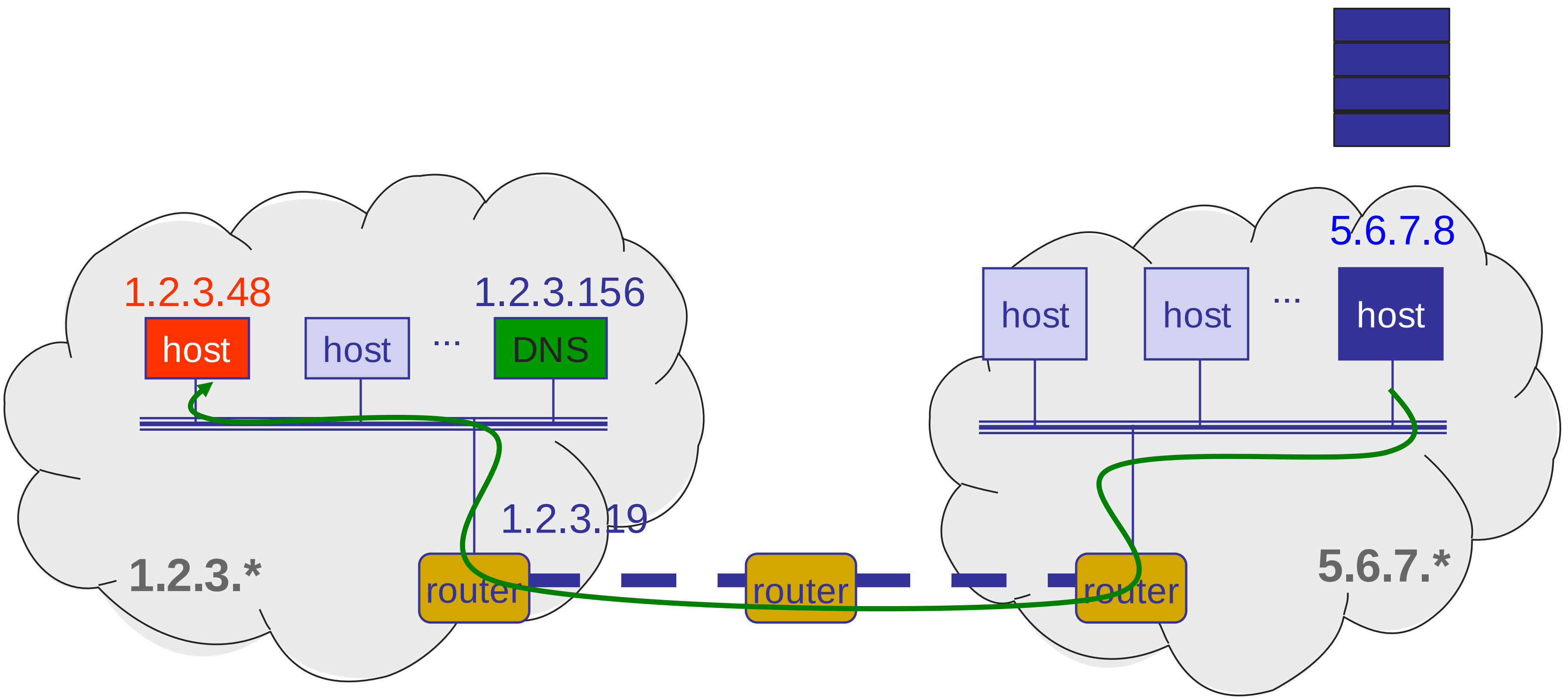
How should we organize communication?

What we want

(Some of) What happens…



(More of) What happens



What we get

Inspiration…

- CEO A writes letter to CEO B
  - Folds letter and hands it to administrative aide
- Aide:
  - Puts letter in envelope with CEO B’s full name
  - Takes to SF Express
- SF Express Office
  - Puts letter in a larger envelope
  - Puts name and street address on the envelope
  - Puts package on a delivery truck
- SF Express delivers to other company

The path of the letter

- “Peers” in same layer understand each other
- No one else needs to
- Lowest level has most packaging

CEO	Semantic Content	CEO
Aide	Identity	Aide
SF Express	Location	SF Express

Three steps

- Decompose the problem into tasks
- Organize these tasks
- Assign tasks to entities (who does what)

Back to the Internet: Decomposition

Applications

in built on

Reliable or unreliable transport

in built on

Best-effort global packet delivery

in built on

Best-effort local packet delivery

in built on

Physical transfer of bits

Communication organization

Applications

in built on

Reliable or unreliable transport

in built on

Best-effort global packet delivery

in built on

Best-effort local packet delivery

in built on

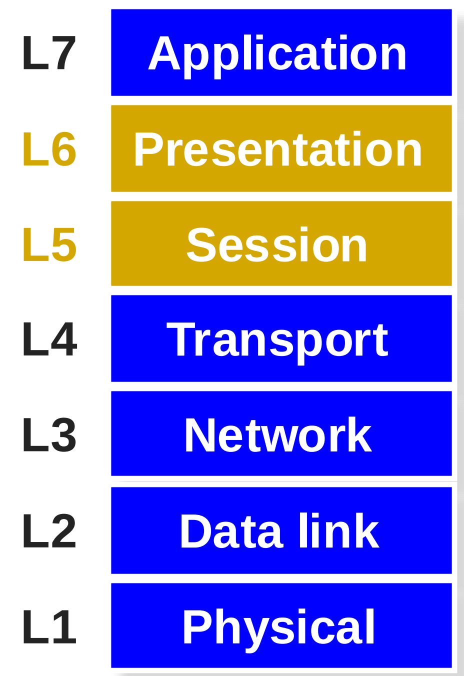
Physical transfer of bits

- L7 Application
- L4 Transport
- L3 Network
- L2 Data link
- L1 Physical

OSI layers

- OSI stands for Open Systems Interconnection model
  - Developed by the ISO

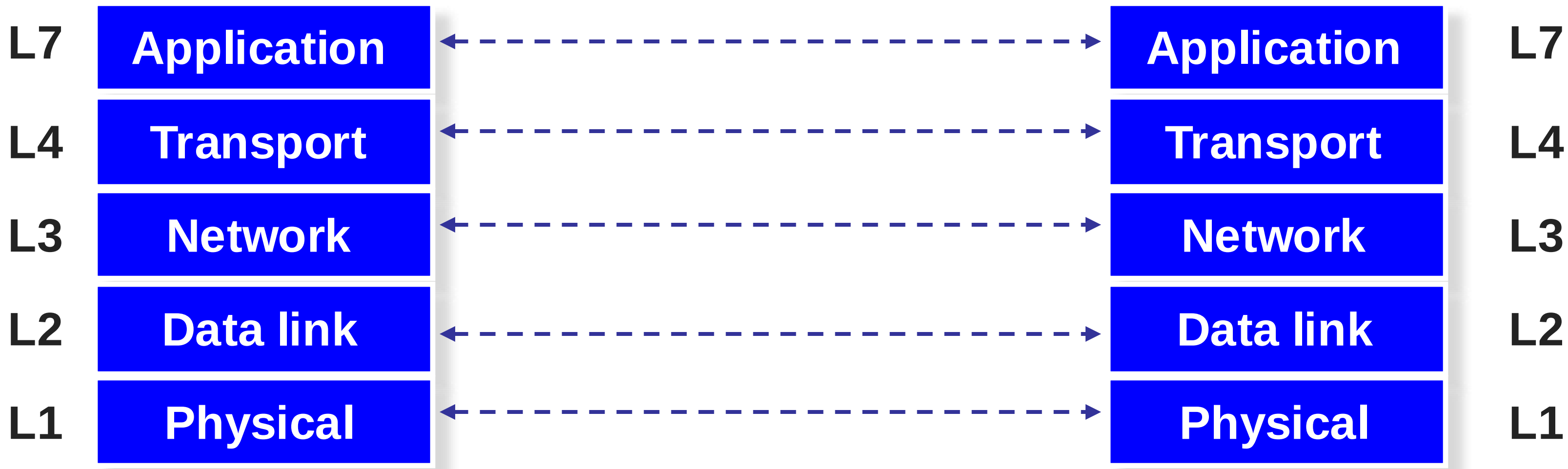
- Session and presentation layers are often implemented as part of the application layer



Layers

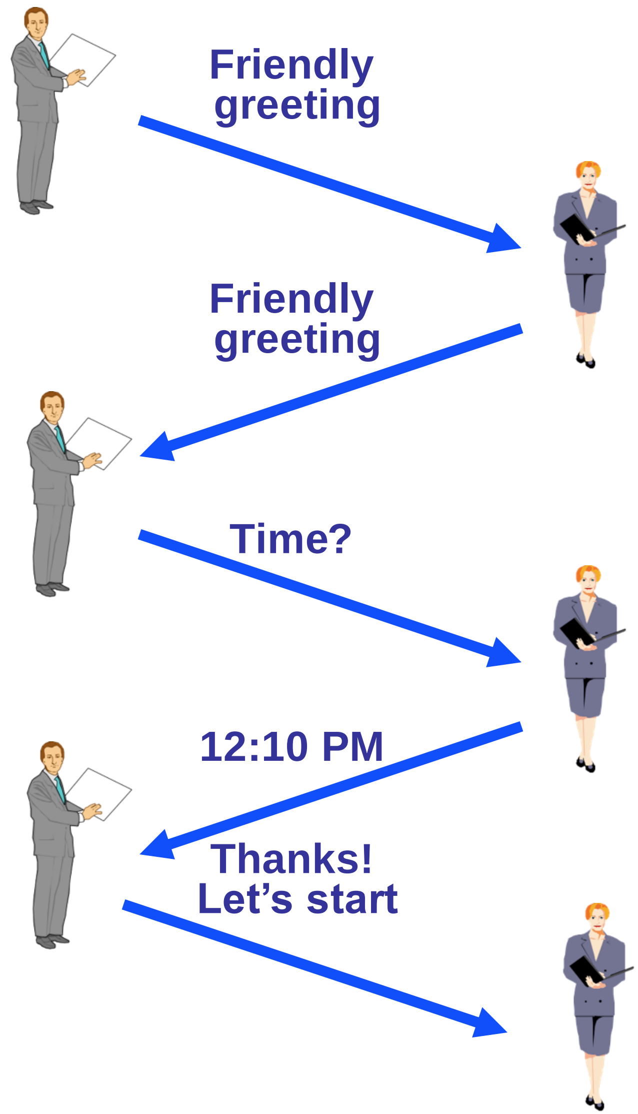
- Layer: a part of a system with well-defined interfaces to other parts
- One layer interacts only with layer above and layer below
- Two layers interact only through the interface between them

Layers and protocols



- Communication between peer layers on different systems is defined by protocols

What is a Protocol?



What is a Protocol?

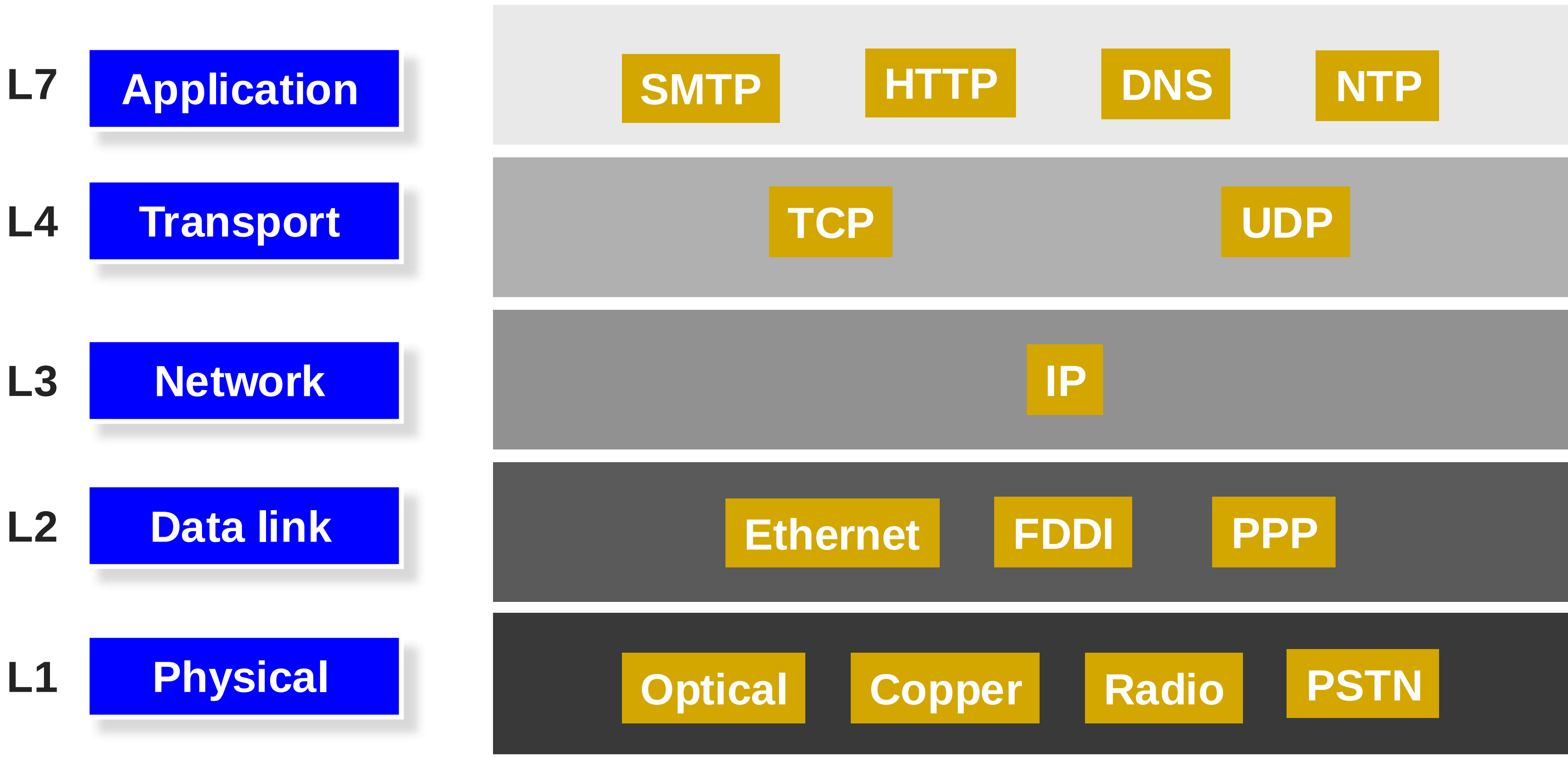
- An agreement between parties (in the same layer) on how to communicate
- Defines the syntax of communication
  - Header → instructions on how to process packet
  - Each protocol defines the format of its headers
    - e.g., “the first 32 bits carry the destination address”

Header	Payload

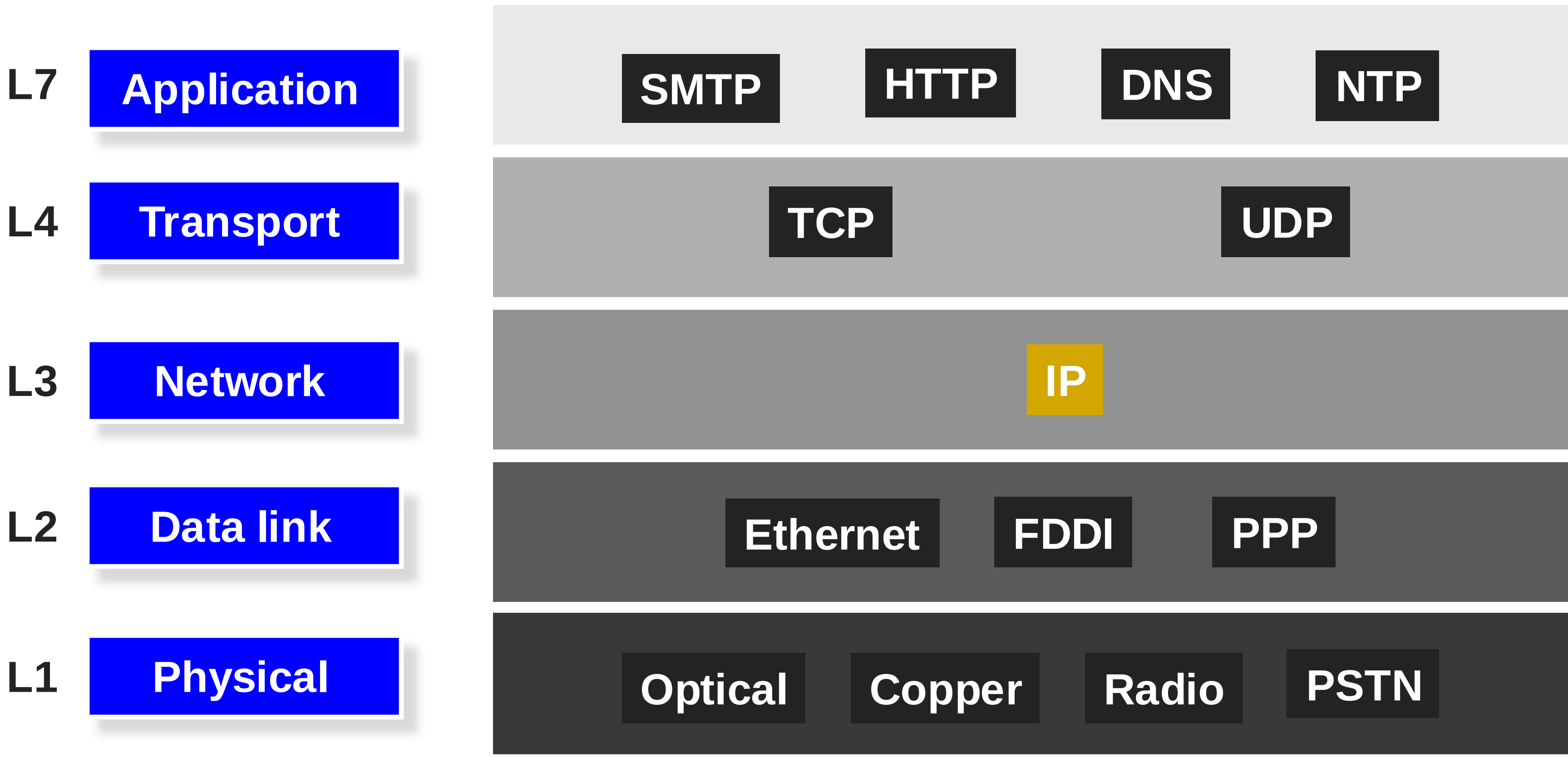
What is a Protocol?

- An agreement between parties on how to communicate
- Defines the syntax of communication
- And semantics
  - “First a hello, then a request…”
  - We will study many protocols later in the semester
- Protocols exist at many levels, hardware, and software
  - Defined by standards bodies like IETF, IEEE, ITU

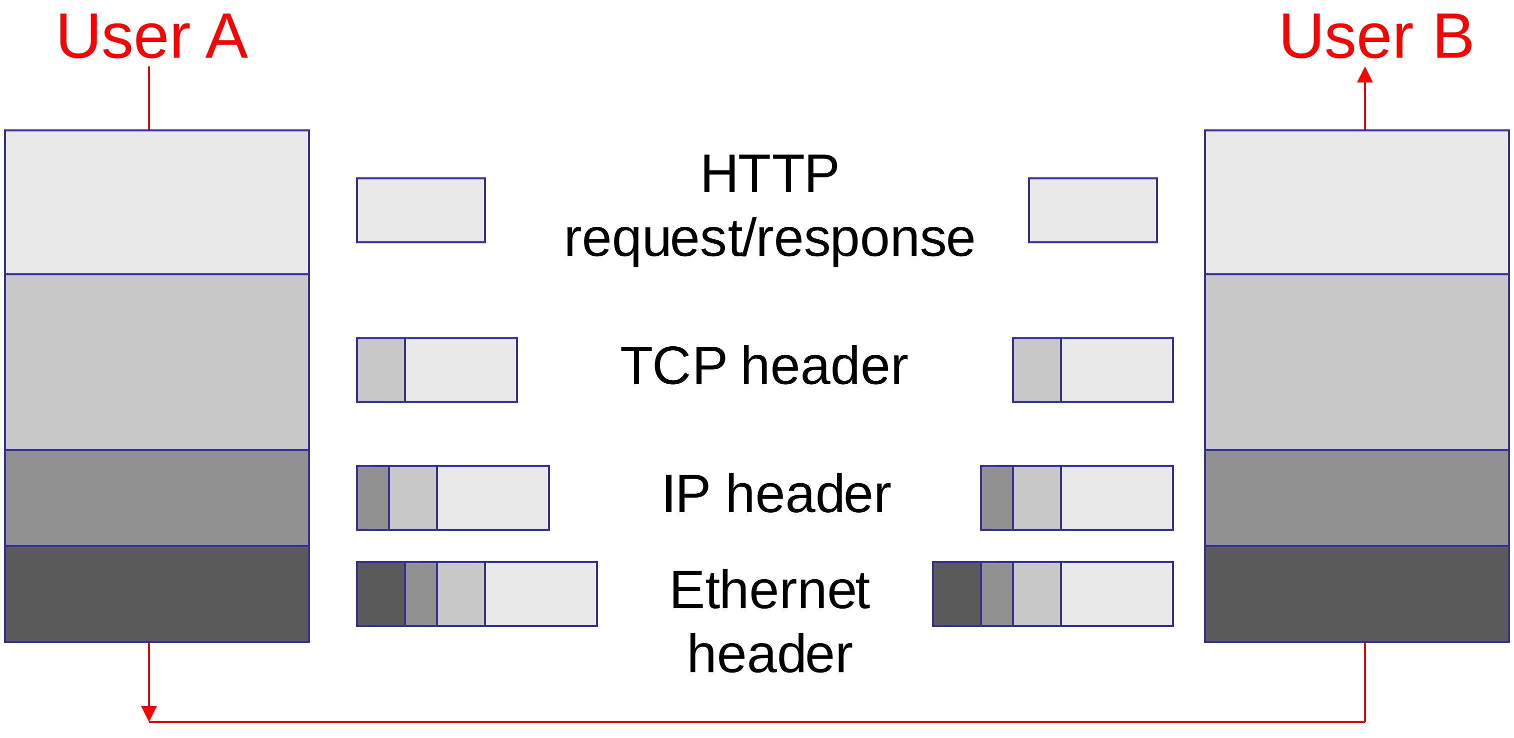
Protocols at different layers



ONE network layer protocol



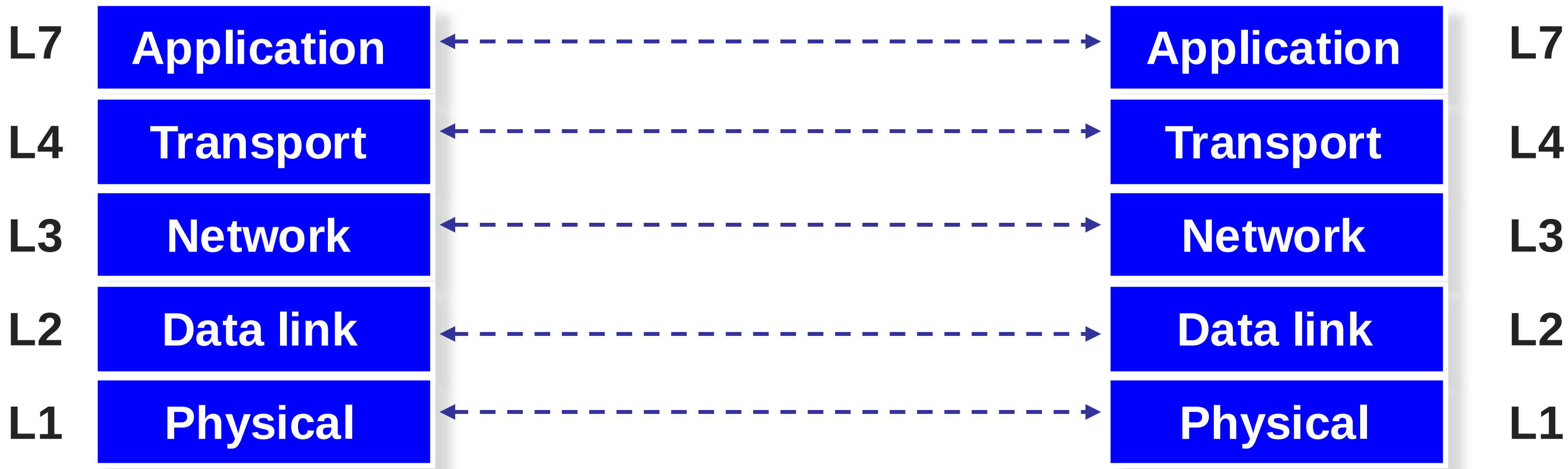
Layer encapsulation: Protocol headers



Three steps

- Decompose the problem into tasks
- Organize these tasks
- Assign tasks to entities (who does what)

What gets implemented where?



What gets implemented at the end systems?

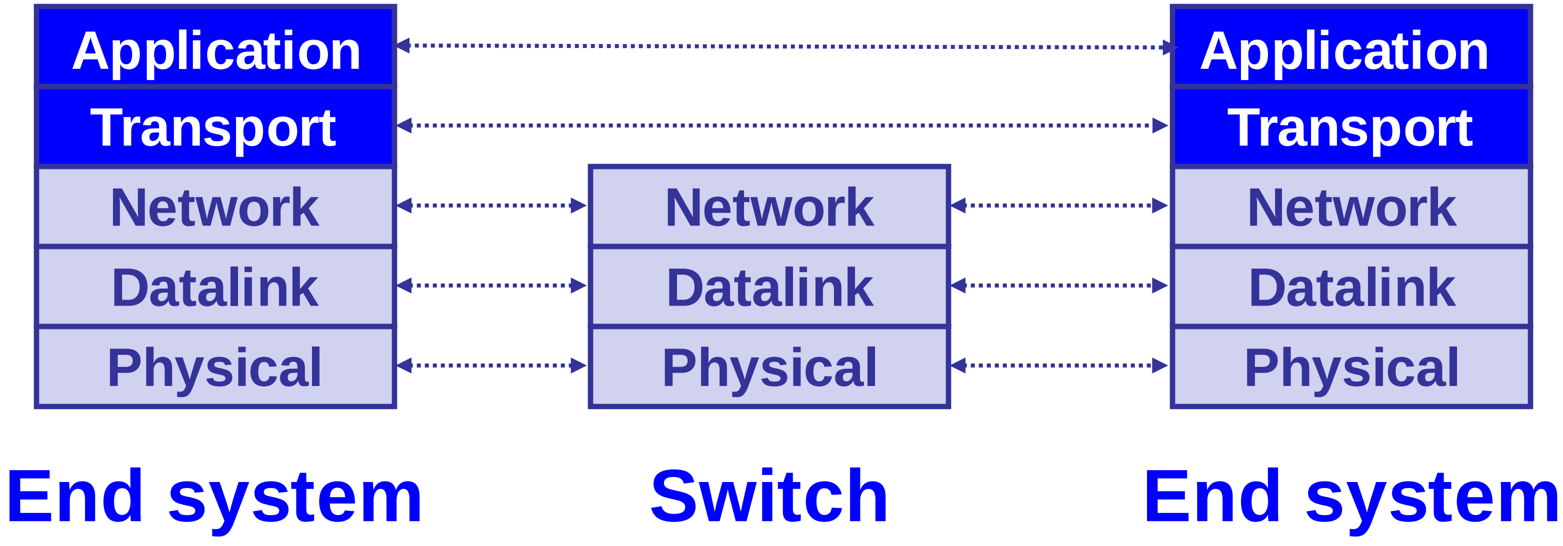
- Bits arrive on wire, must make it up to application
- Therefore, all layers must exist at host!

What gets implemented in the network?

- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)
- The network does not support reliable delivery
  - Transport layer (and above) not supported

Simple Diagram

- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



A closer look: End system

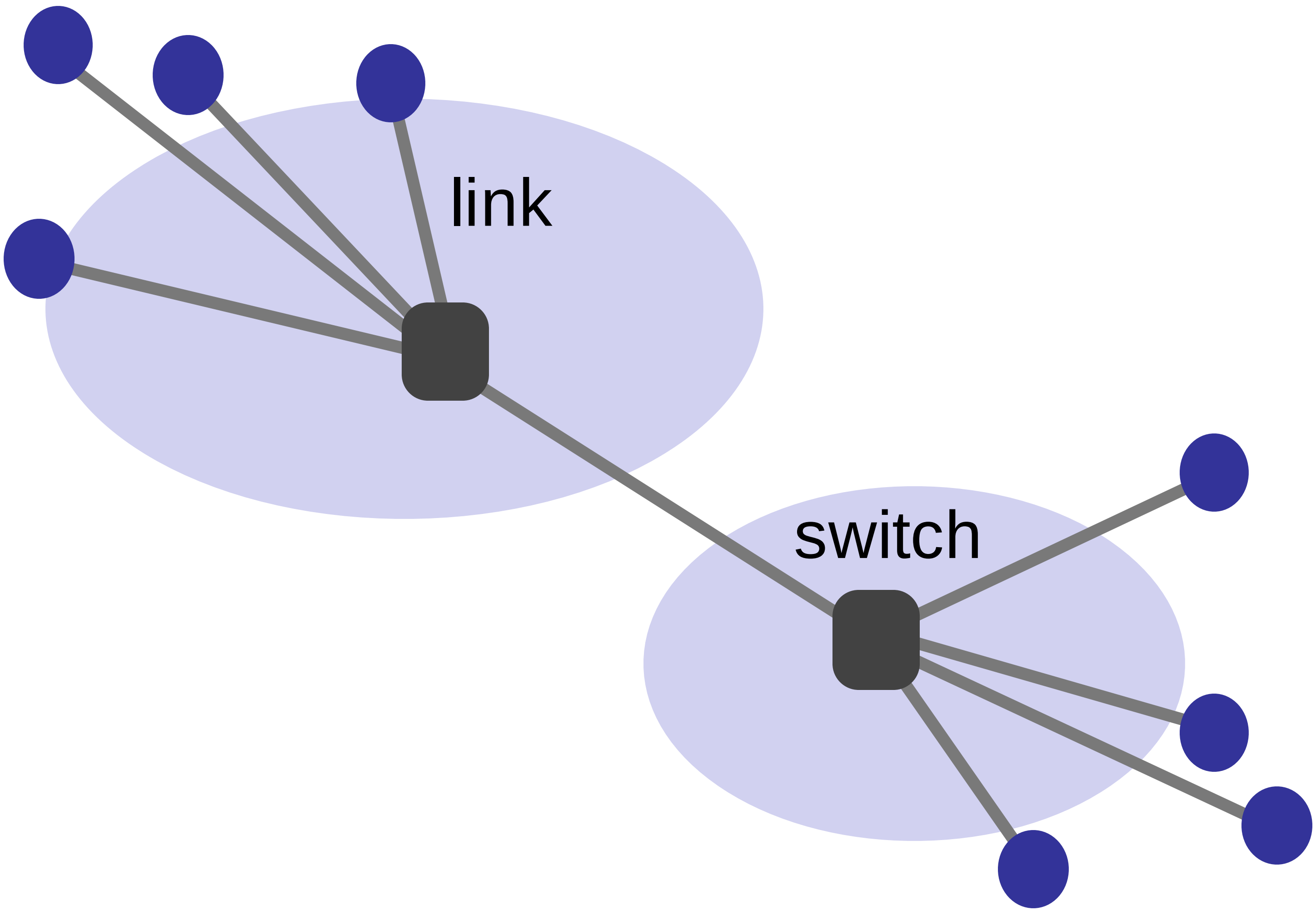
- Application
  - Web server, browser, mail, game
- Transport and network layer
  - Typically part of the operating system
- Datalink and physical layer
  - hardware/firmware/drivers

What gets implemented in the network?

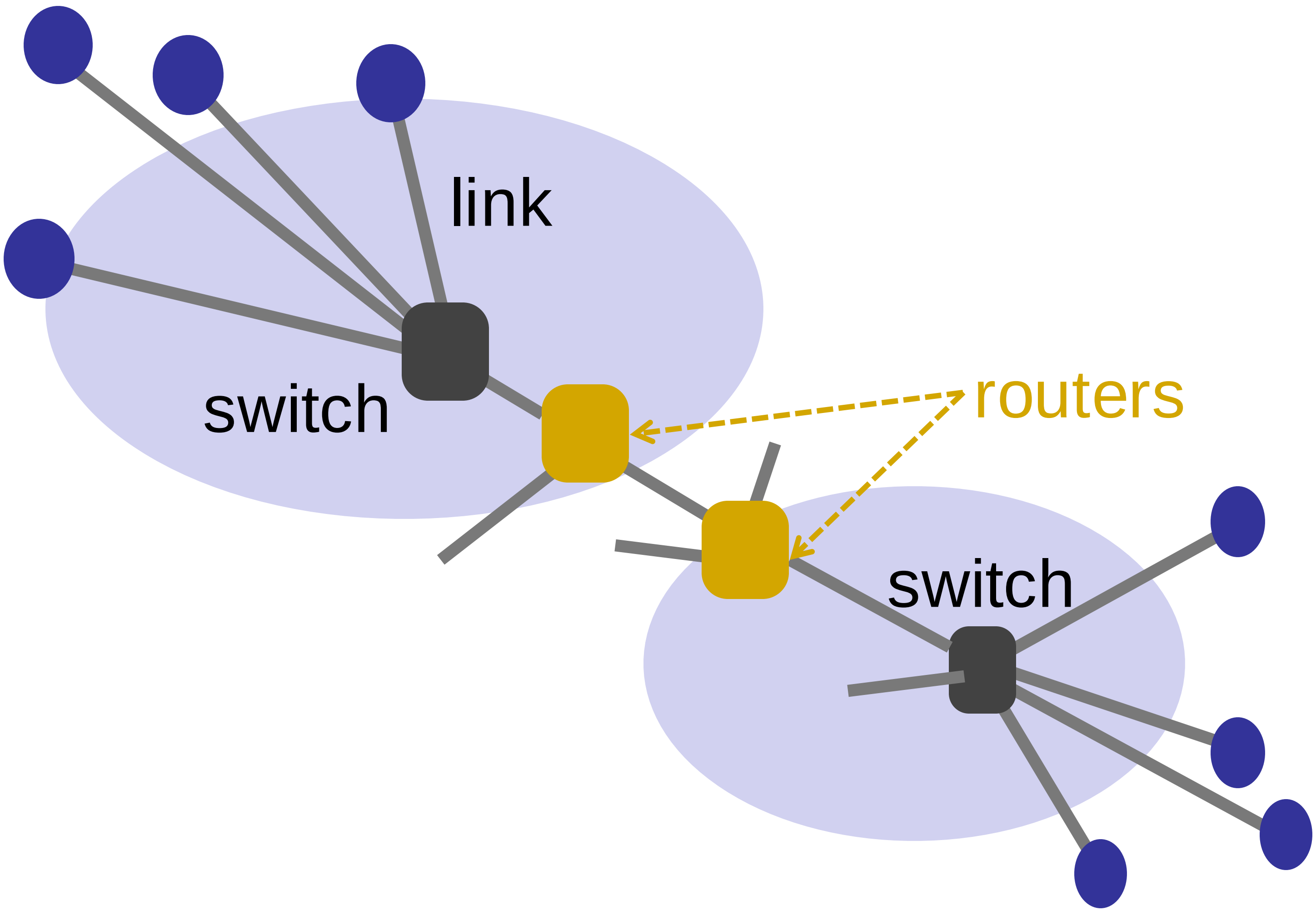
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)

- Switches implement only physical and datalink layers (L1, L2)
- Routers implement the network layer too (L1, L2, L3)

A closer look at the network



A closer look at the network

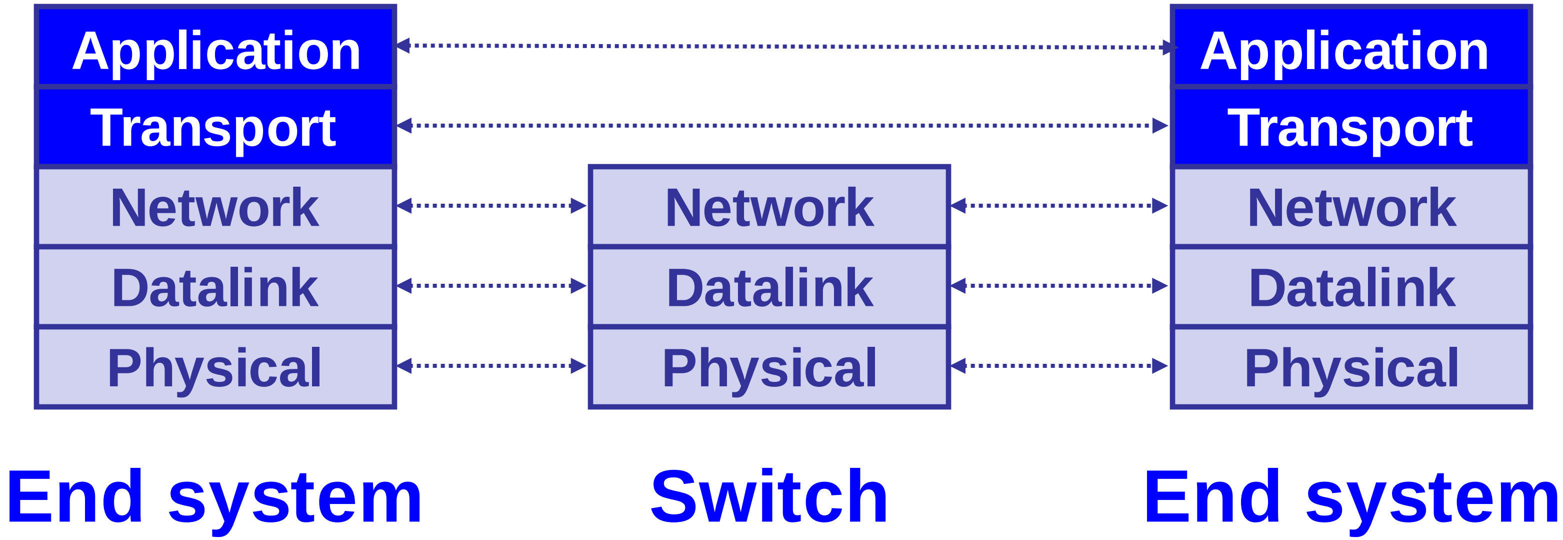


Switches vs. Routers

- Switches do what routers do but don’t participate in global delivery, just local delivery
  - Switches only need to support L1, L2
  - Routers support L1-L3
- Won’t focus on the router/switch distinction
  - Almost all boxes support network layer these days

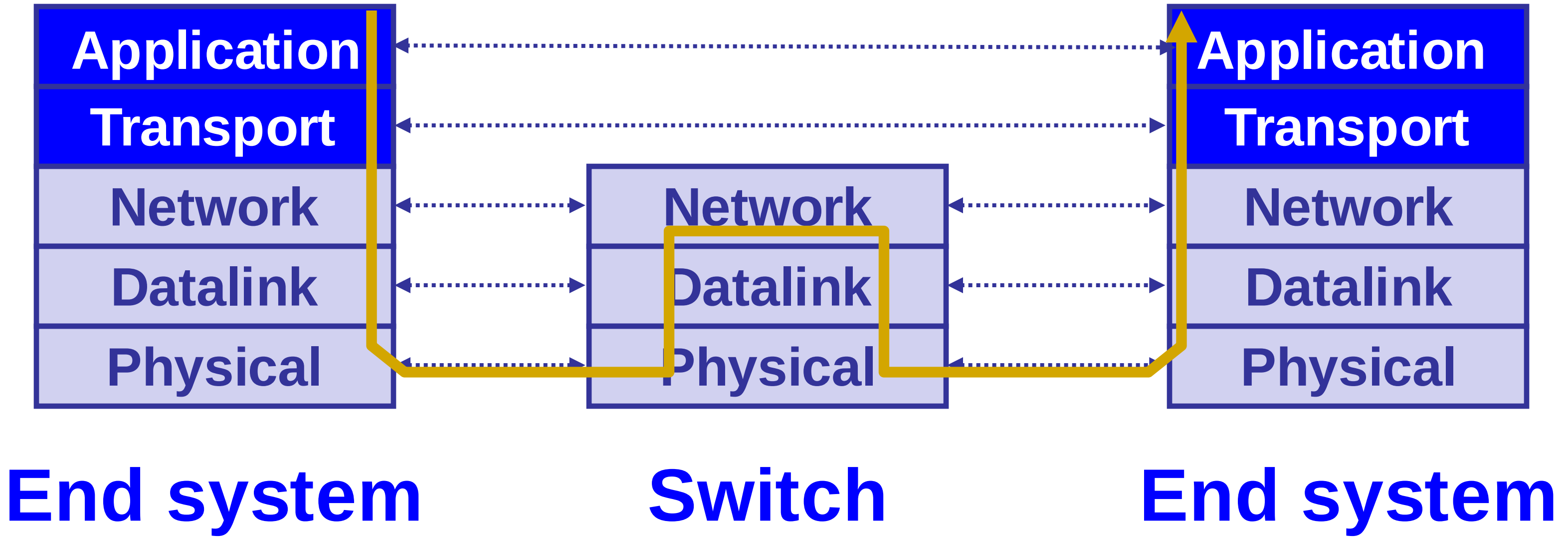
Logical communication

- A layer interacts with its peers in the corresponding layer

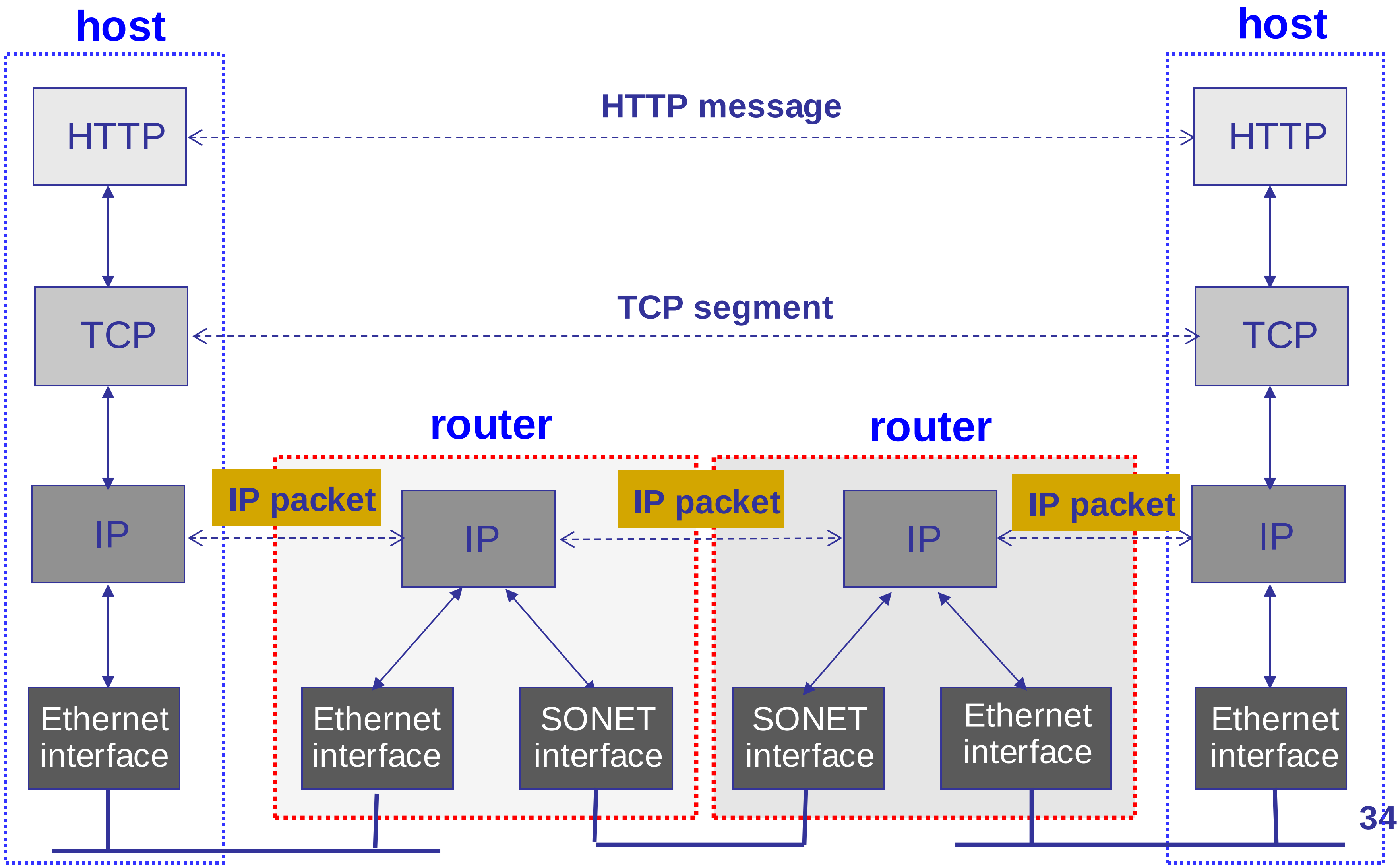


Physical communication

- Communication goes down to physical network
- Then up to relevant layer



A protocol-centric diagram



Pros and cons of layering

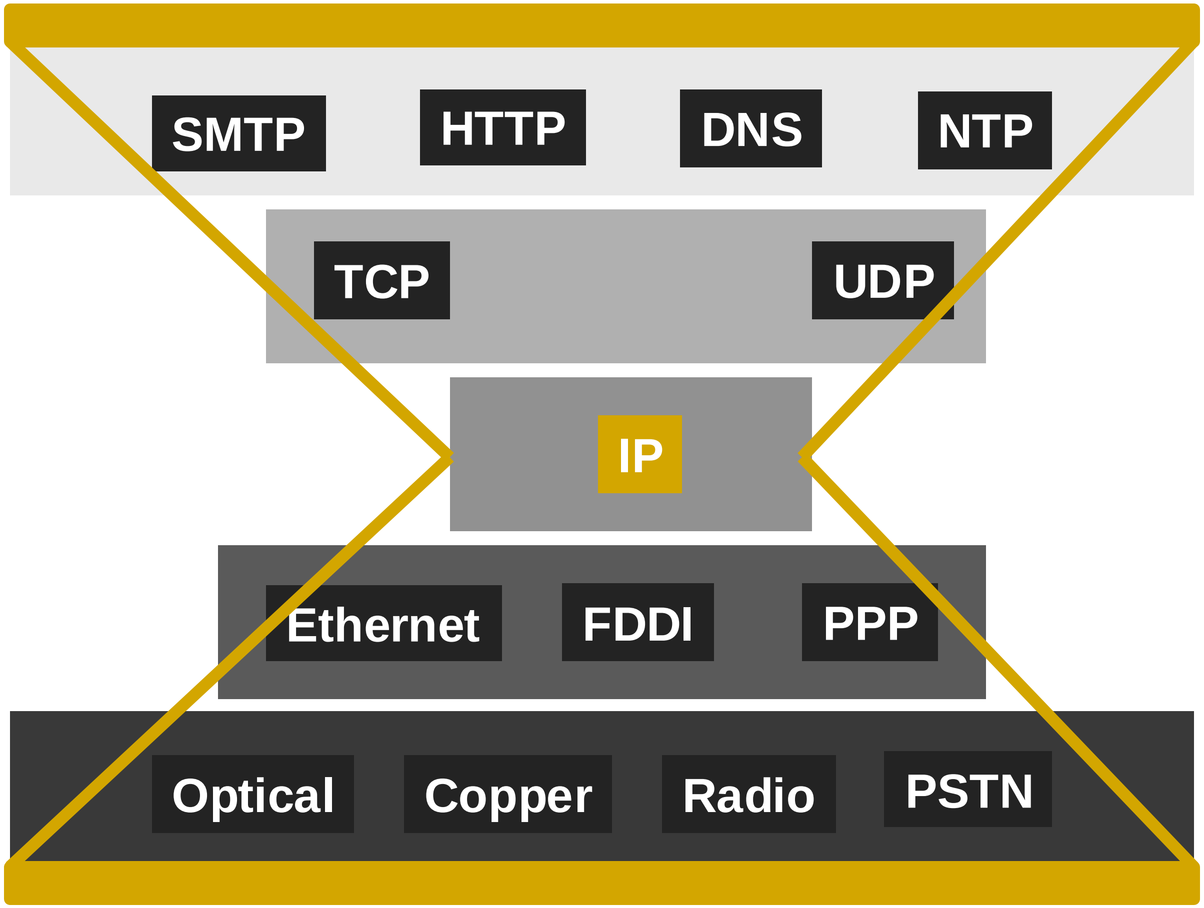
Why layers?

- Reduce complexity
- Improve flexibility

Why not?

- Higher overheads
- Cross-layer information often useful

IP is the narrow waist of the layering hourglass



Implications of hourglass

- Single network-layer protocol (IP)
- Allows arbitrary networks to interoperate
  - Any network that supports IP can exchange packets
- Decouples applications from low-level networking technologies
  - Applications function on all networks
- Supports simultaneous innovations above and below IP
- But changing IP itself is hard (e.g., IPv4 → IPv6)

Placing network functionality

- End-to-end arguments by Saltzer, Reed, and Clark
  - Dumb network and smart end systems
  - Functions that can be *completely* and *correctly* implemented *only* with the knowledge of application at end host, should not be pushed into the network
  - Sometimes necessary to break this for performance and policy optimizations
  - Fate sharing: fail together or don’t fail at all

Quick recap

- Layering is a good way to organize networks
- Unified Internet layer decouples applications from networks, allows technologies to evolve independently
- E2E argument encourages us to keep IP simple

TRANSPORT LAYER

Why a transport layer?

- IP addresses capture hosts, but end-to-end communication happens between applications
  - Need a way to decide which packets go to which applications (multiplexing/demultiplexing)
- IP provides a weak service model (best-effort)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Multiplexing & demultiplexing

- Multiplexing (Mux)
  - Gathering and combining data chunks at the source from different applications and delivering to the network layer
- Demultiplexing (Demux)
  - Delivering correct data to corresponding sockets from a multiplexed stream

Role of the transport layer

- Communication between processes
  - Mux and demux from/to application processes
  - Implemented using *ports*

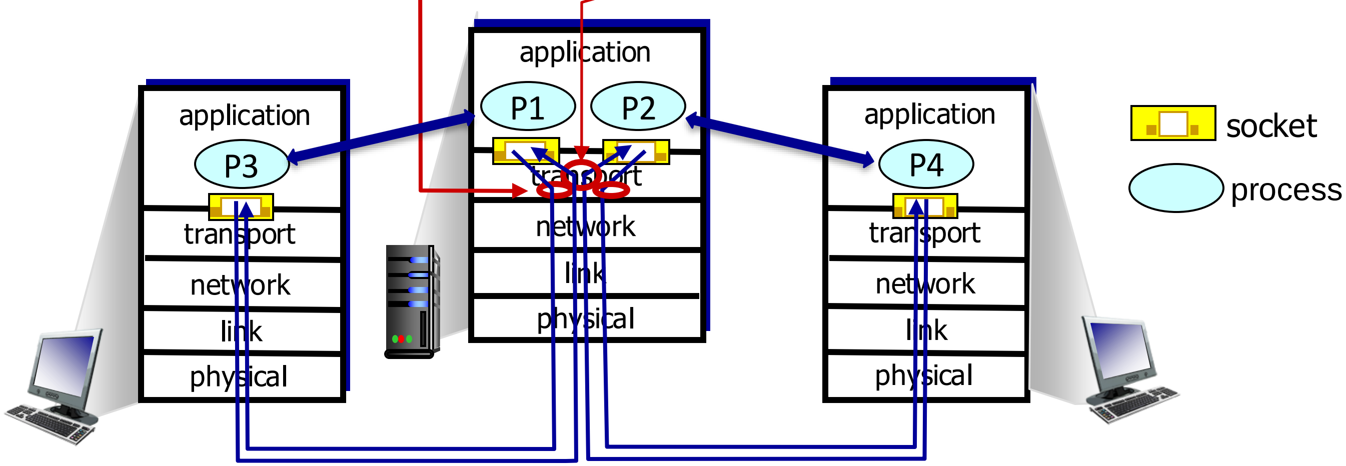
Multiplexing & demultiplexing

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



Role of the transport layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
  - Reliable, in-order data delivery
  - Well-paced data delivery
    - Too fast may overwhelm the network
    - Too slow is not efficient

Role of the transport layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
  - Also SCTP, MPTCP, SST, RDP, DCCP, …

Role of the transport layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist transport protocol
  - Only provides mux/demux capabilities

Role of the transport layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist transport protocol
- TCP offers a reliable, in-order, byte stream abstraction
  - With congestion control, but w/o performance guarantees (delay, b/w, etc.)

Applications and sockets

- Socket: software abstraction for an application process to exchange network messages with the (transport layer in the) operating system
- Two important types of sockets
  - UDP socket: TYPE is SOCK_DGRAM
  - TCP socket: TYPE is SOCK_STREAM

Ports

- 16-bit numbers that help distinguishing apps
  - Packets carry src/dst port no in transport header
  - Well-known (0-1023) and ephemeral ports
- OS stores mapping between sockets and ports
  - Port in packets and sockets in OS
  - For UDP ports (SOCK_DGRAM)
    - OS stores (local port, local IP address) ←→ socket
  - For TCP ports (SOCK_STREAM)
    - OS stores (local port, local IP, remote port, remote IP) ←→ socket

UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of order & reliability
- UDP described in RFC 768 – (1980!)
  - Destination IP address and port to support demultiplexing

SRC port	DST port
Length	Checksum
DATA	

UDP (cont'd)

- Optional error checking on the packet contents
  - (checksum field = 0 means "don't verify checksum")
- Source port is also optional
  - Useful to respond back to the sender in some cases

Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (mux/demux)
- IP provides a weak service model (best-effort)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Reliable transport

- In a perfect world, reliable transport is easy

@Sender

- Send packets

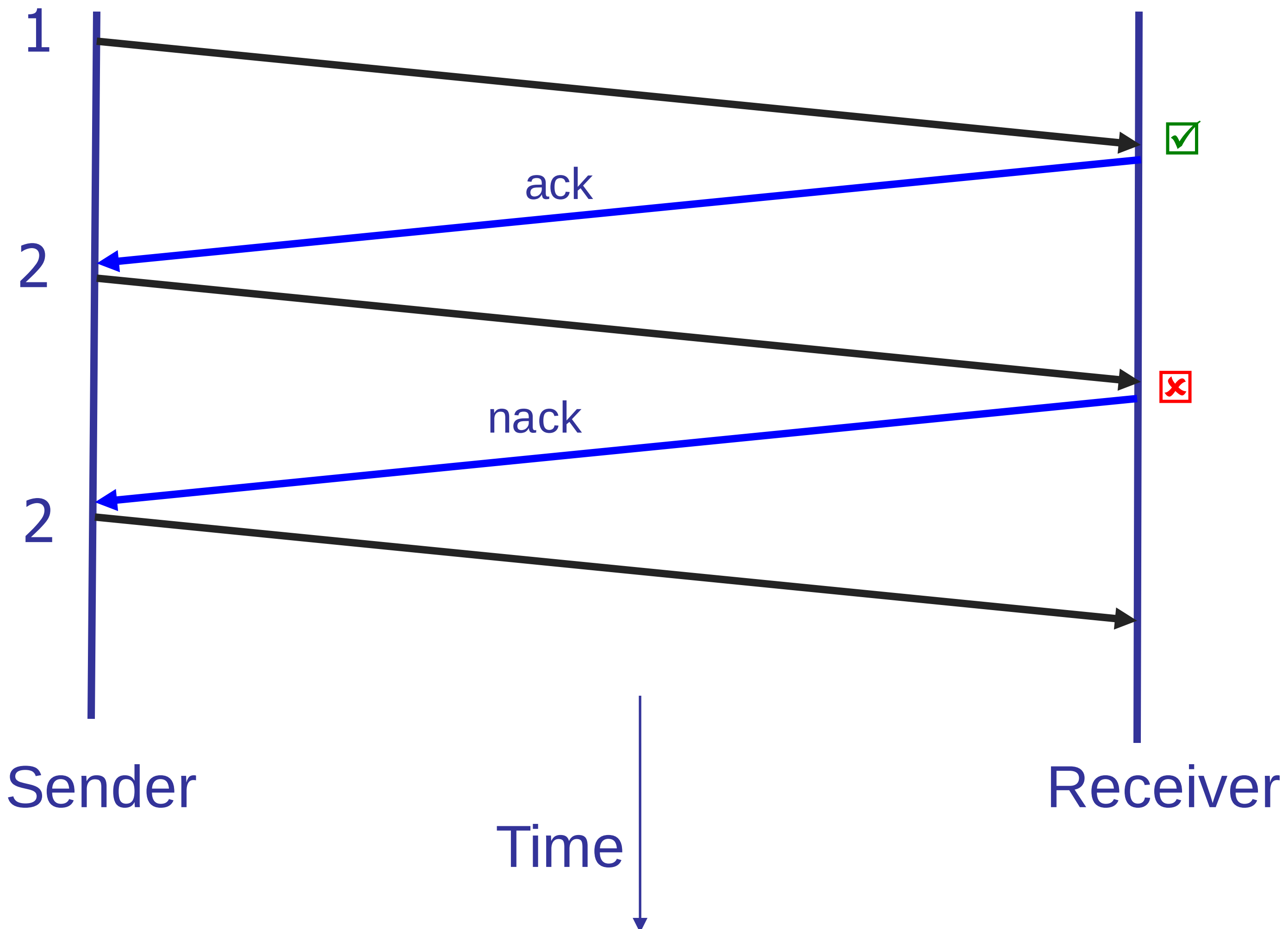
@Receiver

- Wait for packets

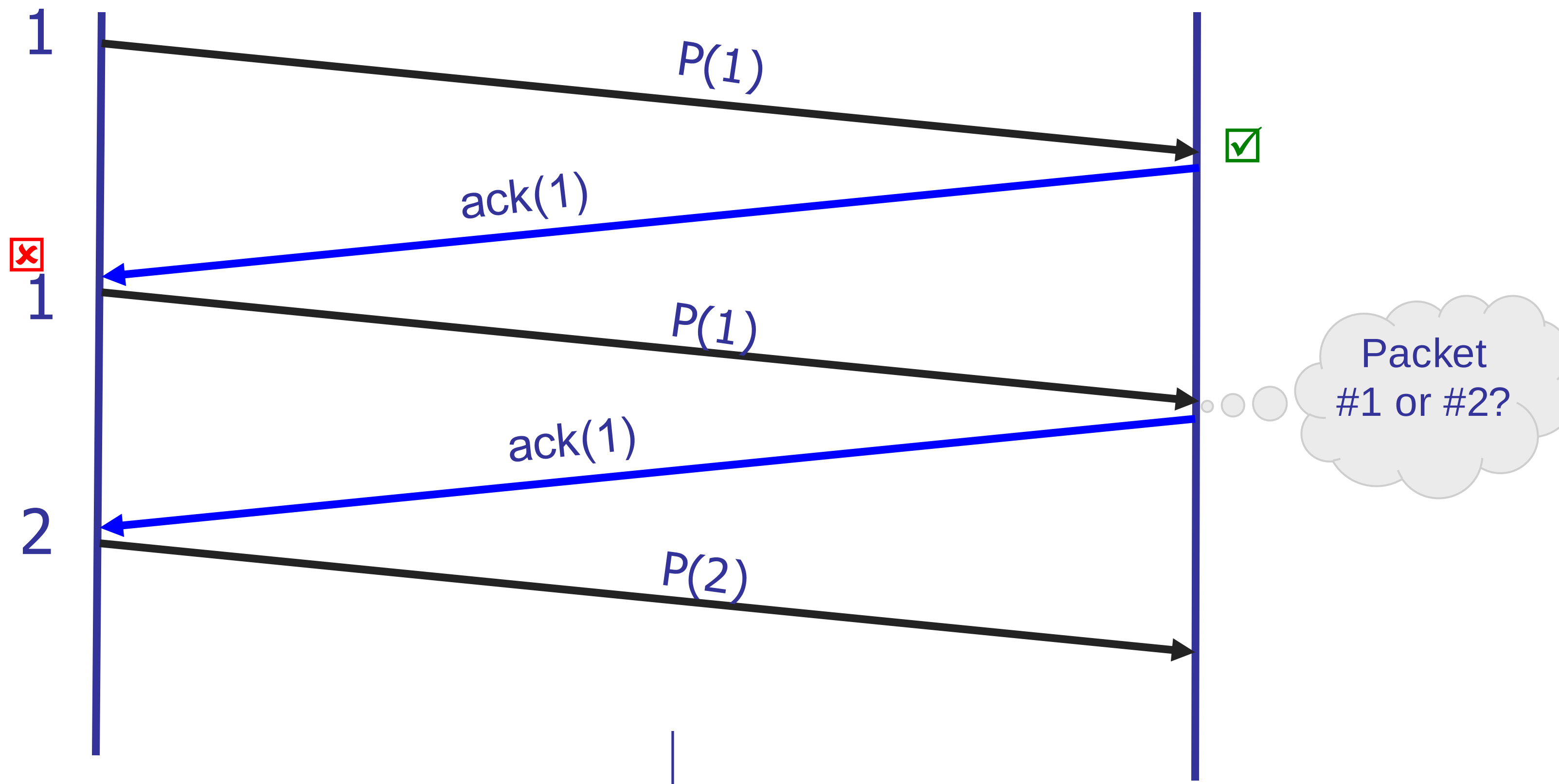
Reliable transport

- In a perfect world, reliable transport is easy
- All the bad things best-effort can do…
  - A packet is corrupted (bit errors)
  - A packet is lost (*why?*)
  - A packet is delayed (*why?*)
  - Packets are reordered (*why?*)
  - A packet is duplicated (*why?*)

Dealing with packet corruption

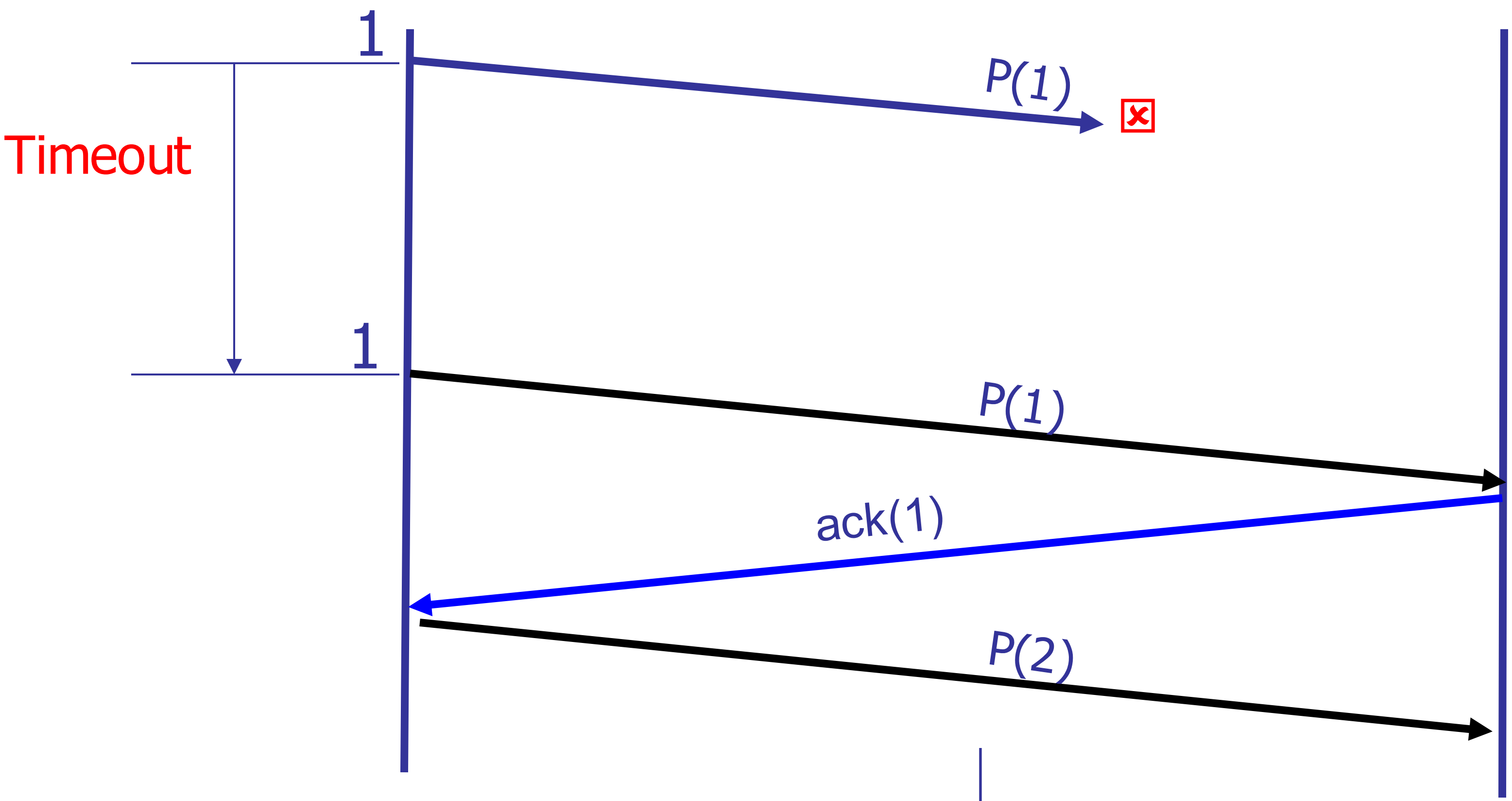


Dealing with packet corruption



Data and ACK packets carry sequence numbers

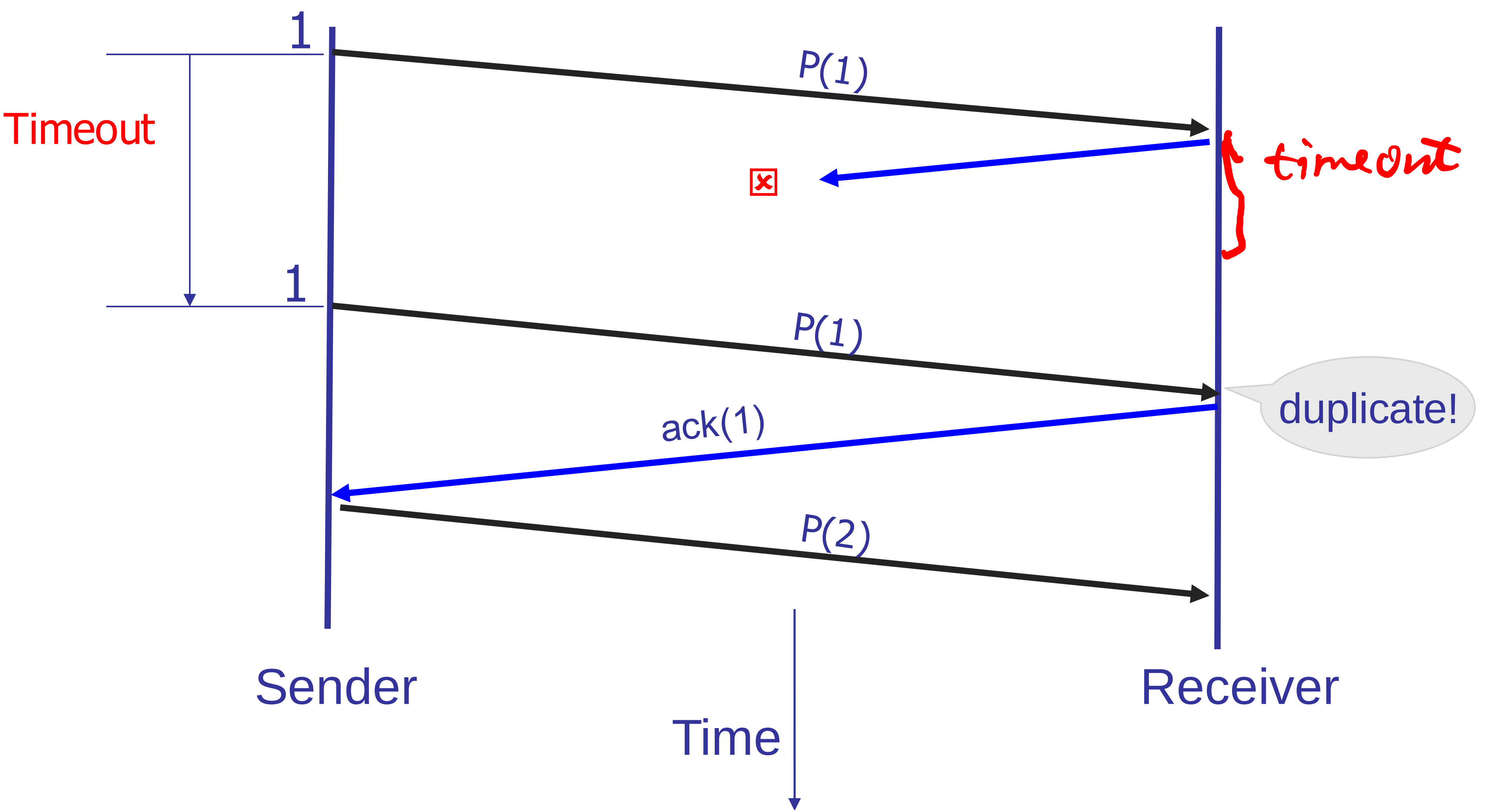
Dealing with packet loss



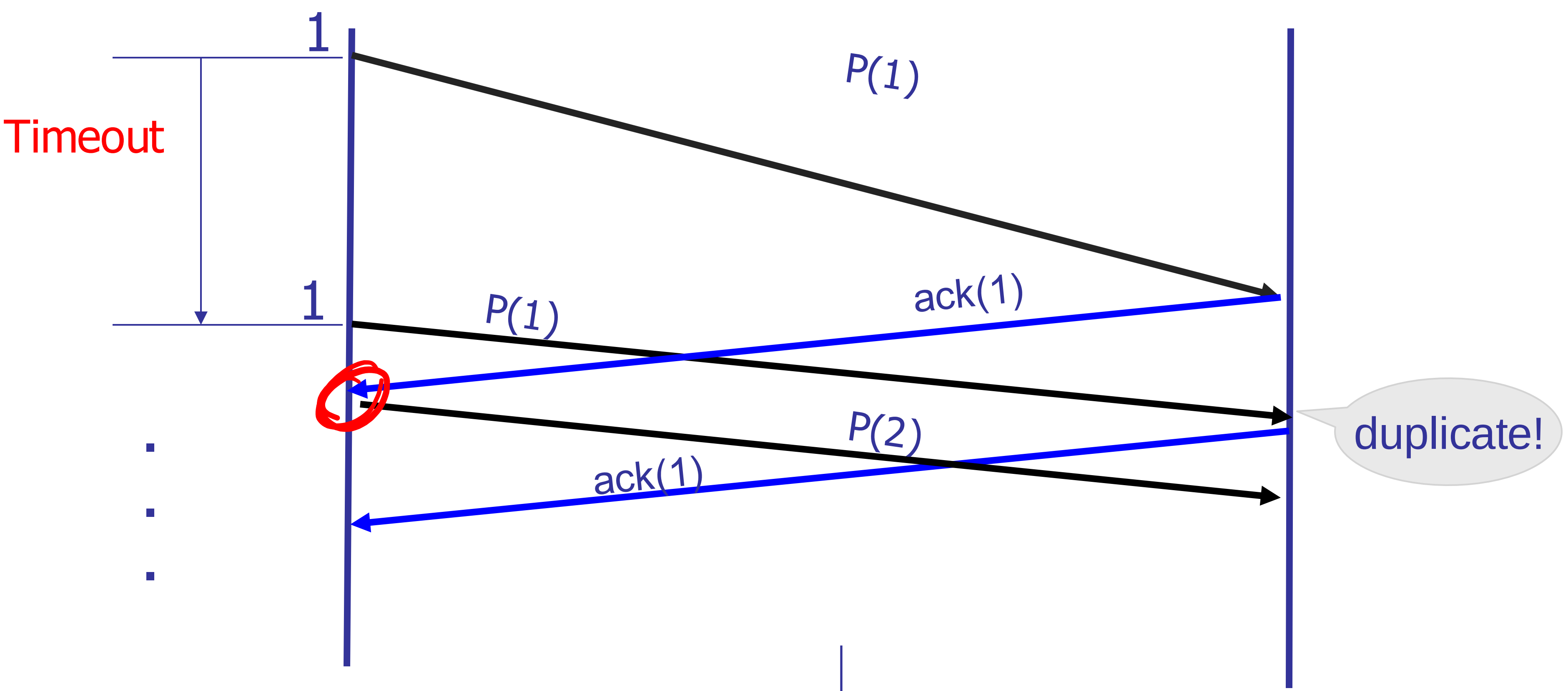
Timer-driven loss detection

Set timer when packet is sent; retransmit on timeout

Dealing with packet loss (of ack)



Dealing with delay



Timer-driven retransmission can lead to <u>duplicates</u>

Components of a solution

- Checksums (to detect bit errors)
- Timers (to detect loss)
- Acknowledgements (positive or negative)
- Sequence numbers (to deal with duplicates)

DESIGNING A RELIABLE TRANSPORT

A Solution: “Stop and Wait”

@Sender

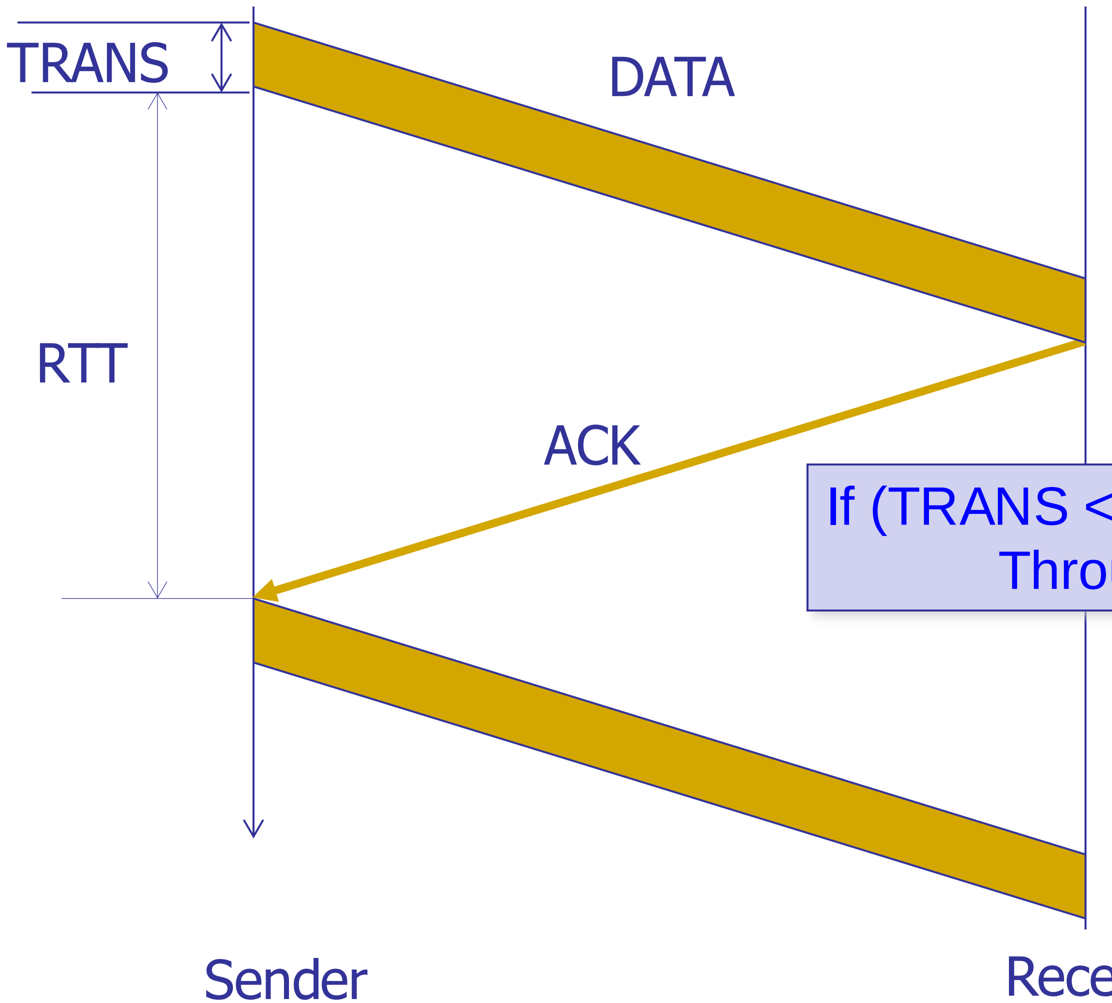
- Send packet(i); (re)set timer; wait for ack
- If (ACK)
  - i++; repeat
- If (NACK or TIMEOUT)
  - repeat

@Receiver

- Wait for packet
- If packet is OK, send ACK
- Else, send NACK
- Repeat

- A correct reliable transport protocol, but an extremely inefficient one

Stop & Wait is inefficient



throughput =

If (TRANS << RTT) then
Throughput ~ DATA/RTT

Orders of magnitude

- Transmission time for 10Gbps link:
  - ~ microsecond for 1500 byte packet
- RTT:
  - 1,000 kilometers ~ O(10) milliseconds

Three design decisions

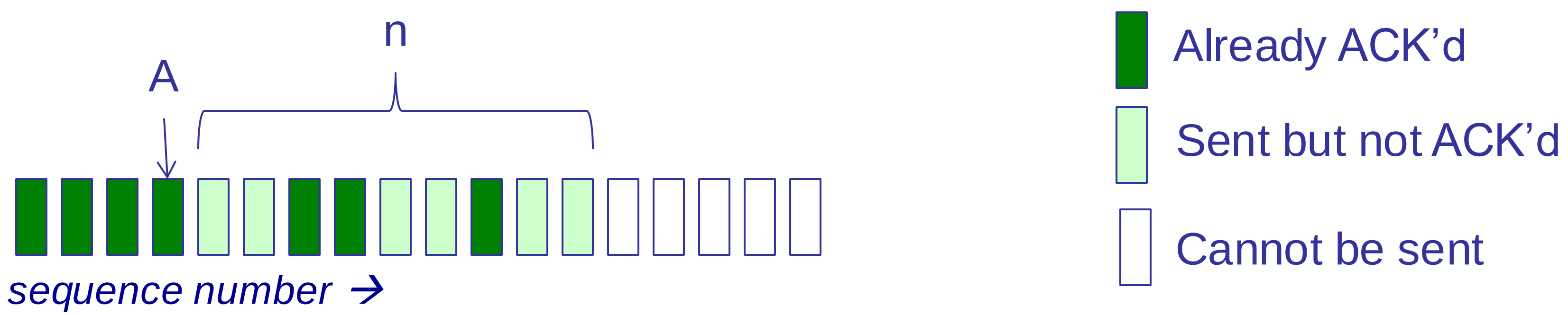
- Which packets can sender send?
- How does receiver ack packets?
- Which packets does sender resend?

Sliding window

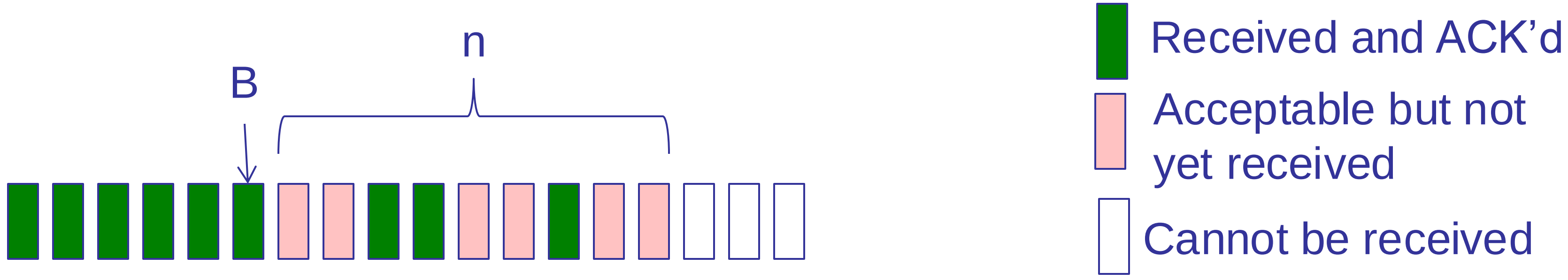
- Window = set of adjacent sequence numbers
  - The size of the set is the window size; assume window size is n
- General idea: send up to n packets at a time
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets "slides" on successful reception/acknowledgement
  - Window contains all packets that might still be in transit
- Sliding window often called "packets in flight" or "in-flight packets"

Sliding window

- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}



- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}



Acknowledgements w/ sliding window

- Two common options
  - Cumulative ACKs: ACK carries next in-order sequence number the receiver expects
  - Selective ACKs: ACK individually acknowledges correctly received packets

- Selective ACKs offer more precise information but require more complicated book-keeping

Sliding window protocols

- Resending packets: two canonical approaches
  - Go-Back-N
  - Selective Repeat
- Many variants that differ in implementation details

Go-Back-N

- Sender transmits up to n unacknowledged packets
- Receiver only accepts packets in order
  - Discards out-of-order packets (i.e., packets other than B+1)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#
- Sender sets timer for 1st outstanding ack (A+1)
- If timeout, retransmit A+1, … , A+n

Selective Repeat (SR)

- Sender: transmit up to n unacknowledged packets
- Assume packet k is lost, k+1 is not
  - Receiver: indicates packet k+1 correctly received
  - Sender: retransmit only packet k on timeout
- Efficient in retransmissions but complex book-keeping
  - Need a timer per packet

GBN vs. Selective Repeat

- When would GBN be better?
  - When error rate is low; wastes bandwidth otherwise

- When would SR be better?
  - When error rate is high; otherwise, too complex

Observations

- For a large-enough window, it is possible to fully utilize a link with sliding windows
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits
- Implementation complexity depends on protocol details (GBN vs. SR)

Components of a solution

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - Cumulative
  - Selective
- Sequence numbers (duplicates, windows)
- Sliding windows (for efficiency)
- Reliability protocols use the above to decide when and what to retransmit or acknowledge

Quick recap

- Transport layer allows applications to communicate with each other
- Provides unreliable and reliable mechanisms
- Possible to build reliable transport over unreliable medium

Summary

- We’ve skipped many important materials
  - TCP: flow control, congestion control
  - L3 routing
  - User-space network stacks
  - L7 applications
  - Network management
- Networking is lots of fun!
  - Also lots of mess
- Take CSCI4430 next term if this excites you