CS2105 Introduction to Computer Networks

Lecture 4
Reliable Protocols

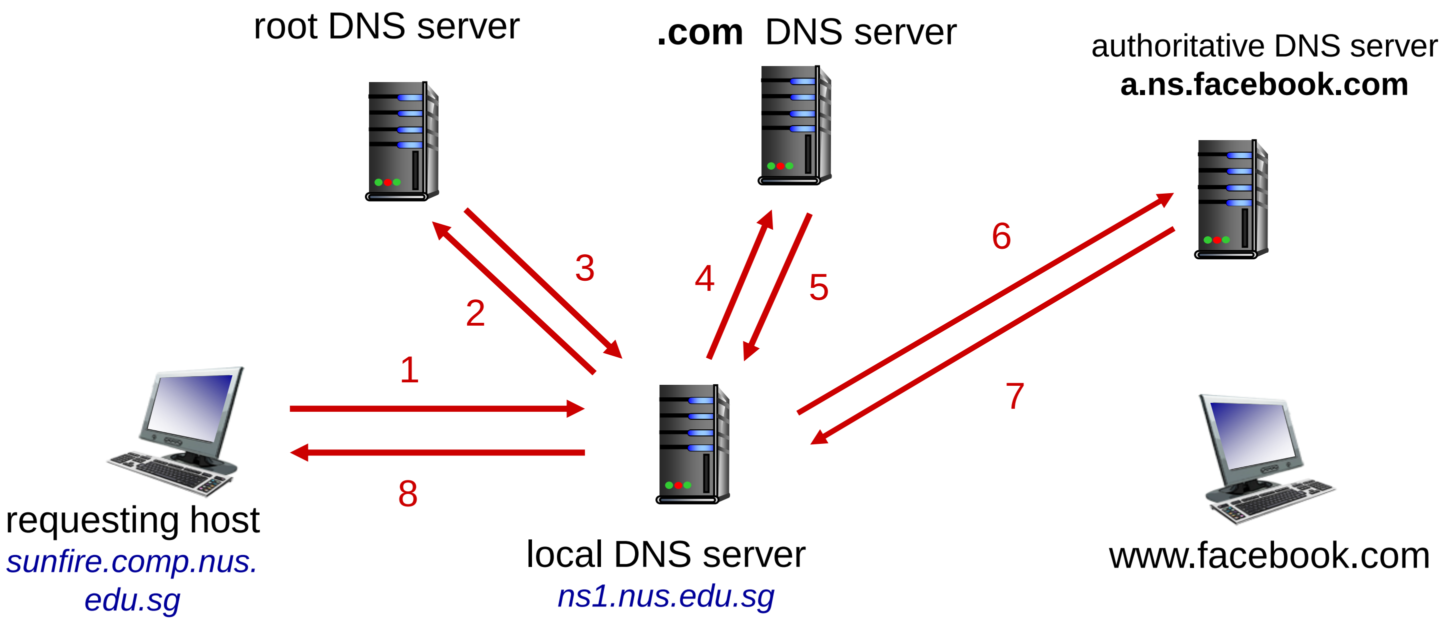
3 September 2018

PREVIOUS LECTURE

Domain Name System

DNS is the Internet's primary directory service.

- It translates host names, which can be easily memorized by humans, to numerical IP addresses used by hosts for the purpose of communication.



Socket

PREVIOUS LECTURE

Applications (processes) send messages over the network through sockets.

- Conceptually, socket = IP address + port number
- Programming wise, socket = a set of APIs

TCP and UDP sockets

- TCP socket (stream socket) uses TCP as transport layer protocol.
  - Connection-oriented, reliable
- UDP socket (datagram socket) uses UDP.
  - Connection-less, unreliable (transmitted data may be lost, corrupted or received out-of-order)

Learning Outcomes

After this class, you are expected to:

- know how the transport and network layer interface.
- be able to design your own reliable protocols with ACK, NAK, sequence number, timeout and retransmission.
- know how to calculate the utilization of a channel.
- know the workings of Go-Back-N and Selective Repeat protocols.

Chapter 3: Roadmap

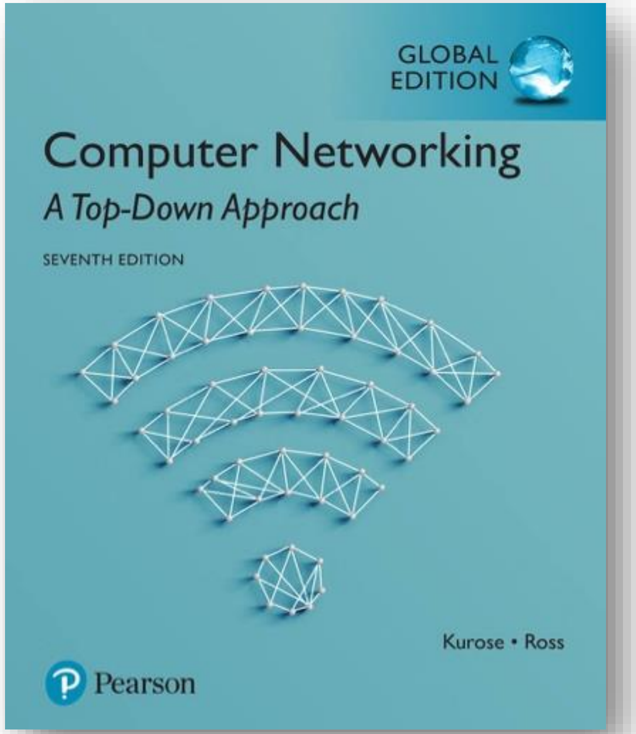
3.1 Transport-layer services

3.2 Multiplexing and de-multiplexing

3.3 Connectionless transport: UDP

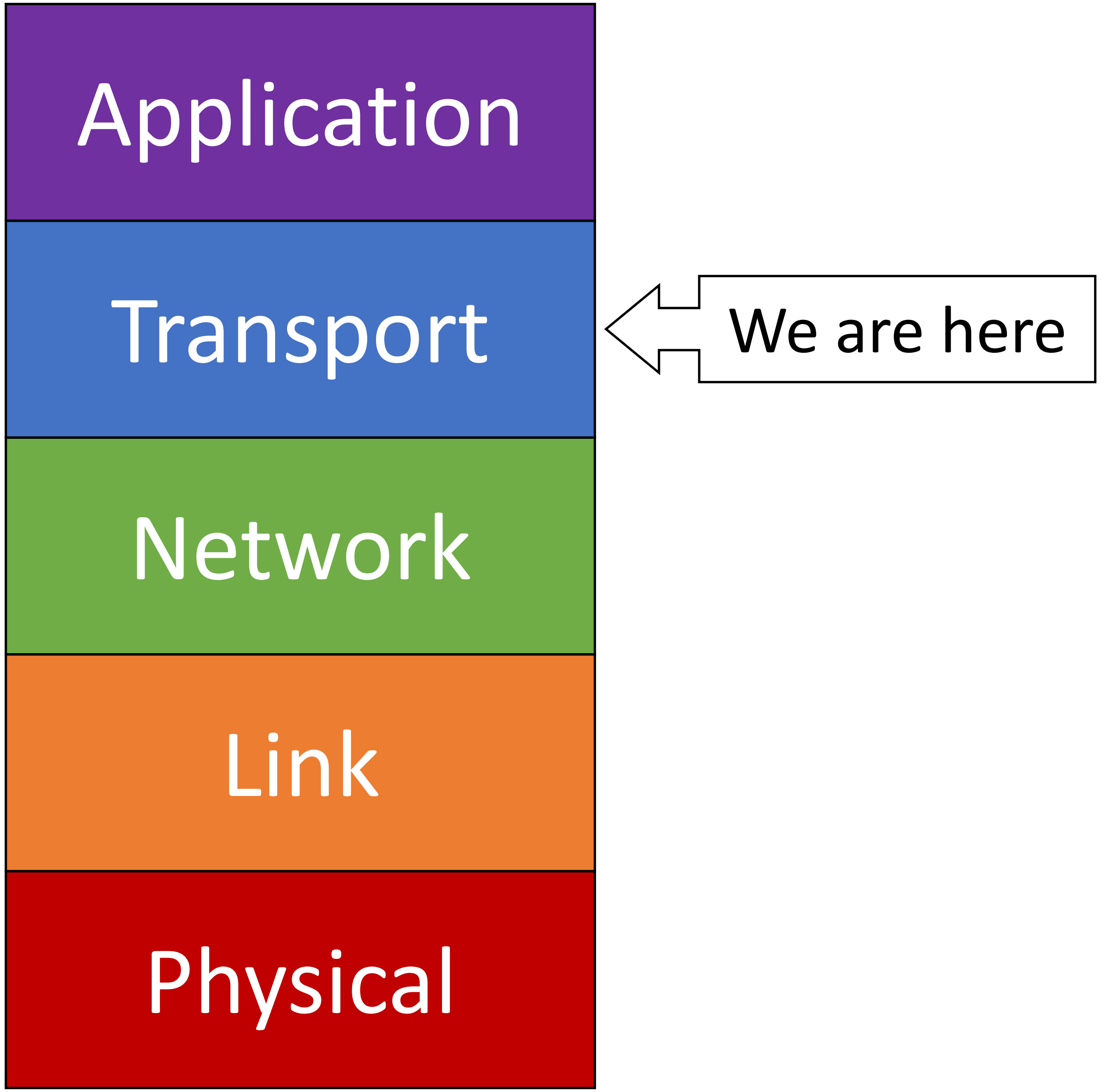
3.4 Principles of reliable data transfer

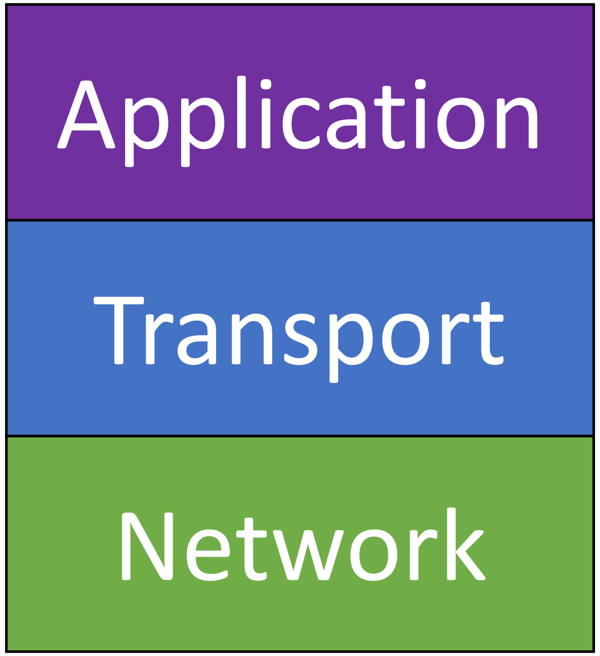
3.5 Connection-oriented transport: TCP



*some slides taken from the publisher

Assignment 1


Application
Transport
Network
Link
Physical
We are here



Transport layer

- resides on end hosts
- process-to-process communication

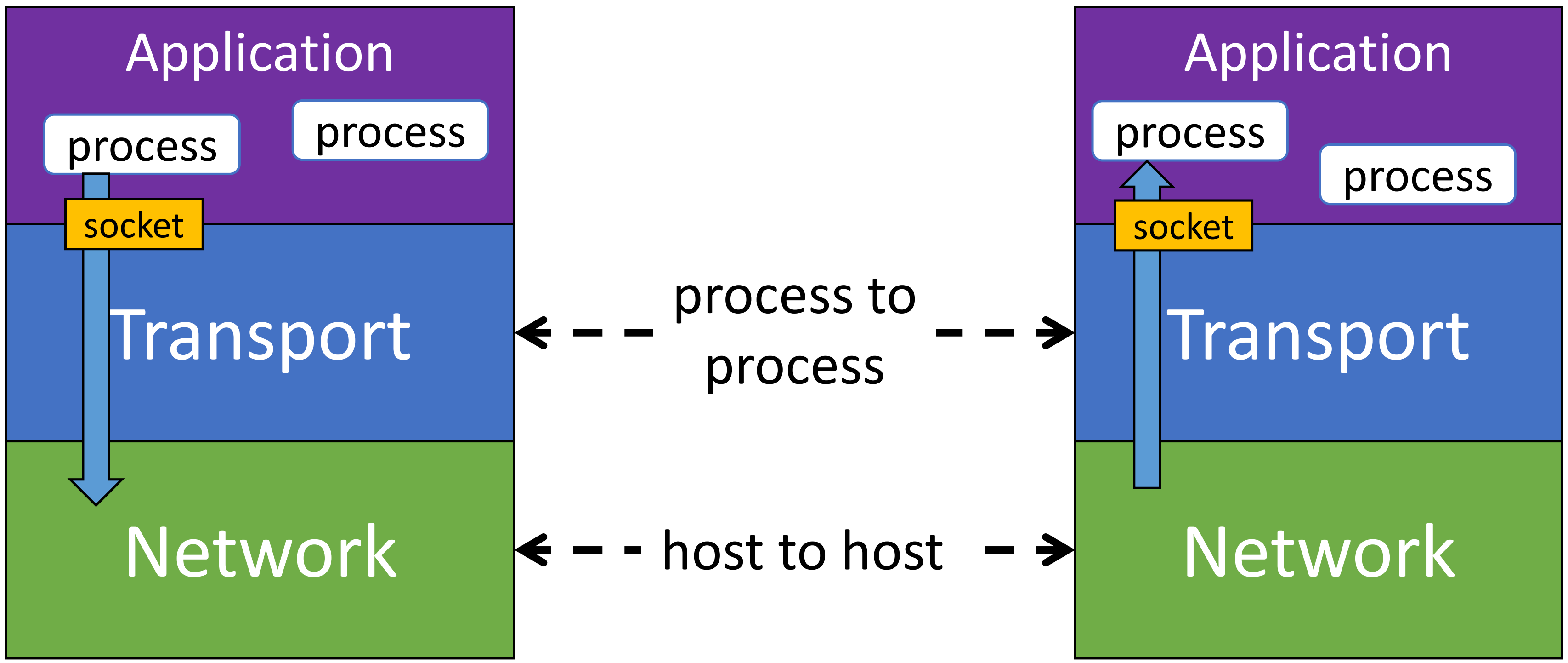
Transport Layer Services

Internet transport layer protocols:

- TCP: connection-oriented and reliable
- UDP: connection-less and unreliable

Transport layer protocols run in hosts.

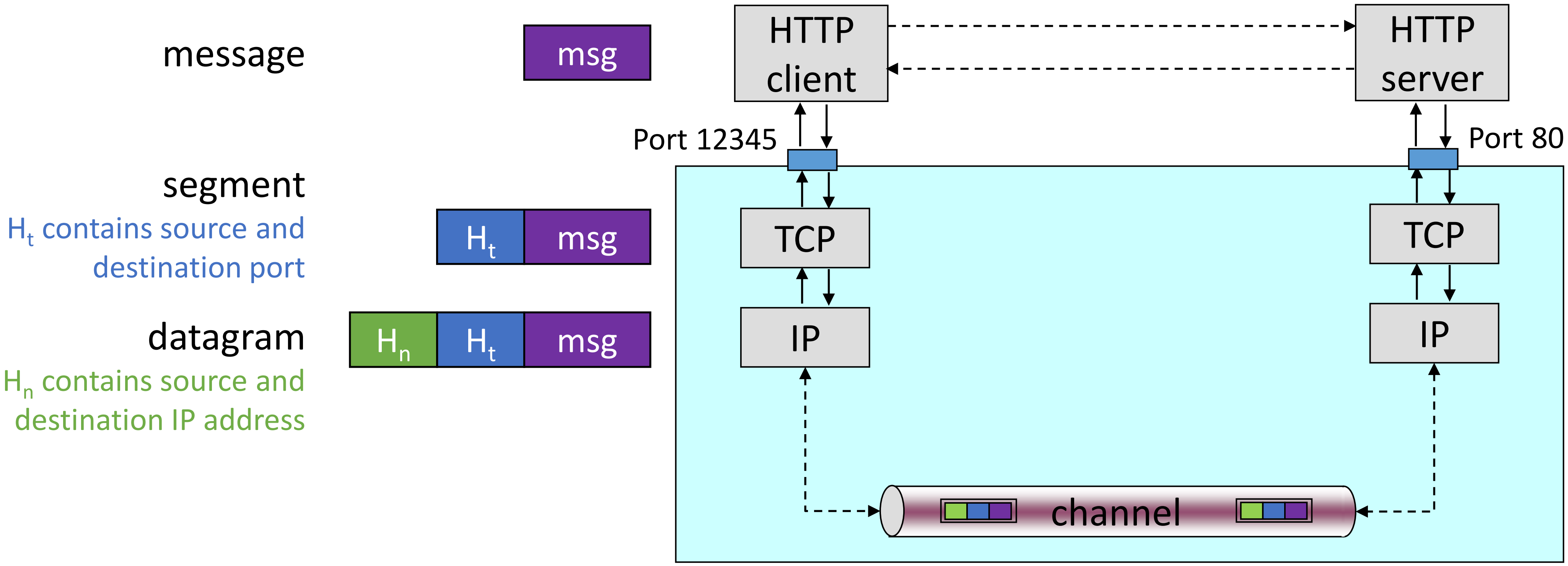
- Sender side: breaks app message into segments (as needed), passes them to network layer (aka IP layer).
- Receiver side: reassembles segments into message, passes it to app layer.
- Packet switches (routers) in between: only check destination IP address to decide routing.


Application
process
process
socket
Transport
Network
process to
process
host to host
Application
process
process
socket
Transport
Network

Transport / Network Layers

Each IP datagram contains source and dest IP addresses.

- Receiving host is identified by dest IP address.
- Each IP datagram carries one transport-layer segment.
- Each segment contains source and dest port numbers.



Chapter 3: Roadmap

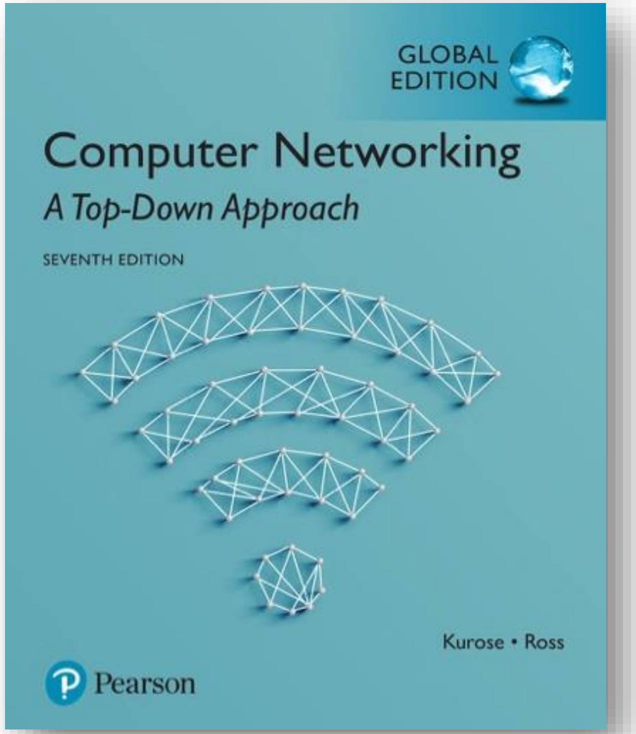
3.1 Transport-layer services

3.2 Multiplexing and de-multiplexing

3.3 Connectionless transport: UDP

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

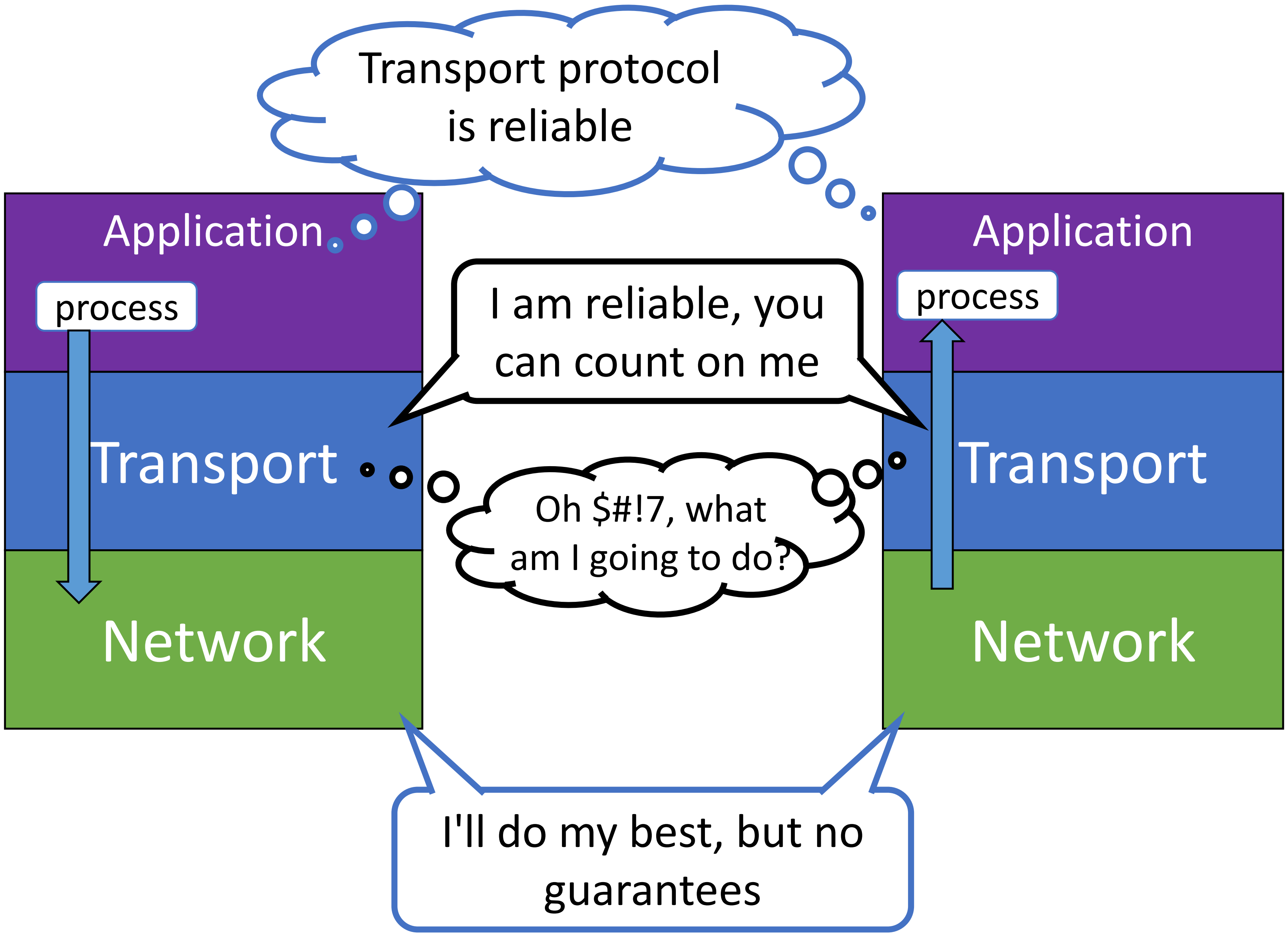


*some slides taken from the publisher

What assumptions should we make about the network layer?

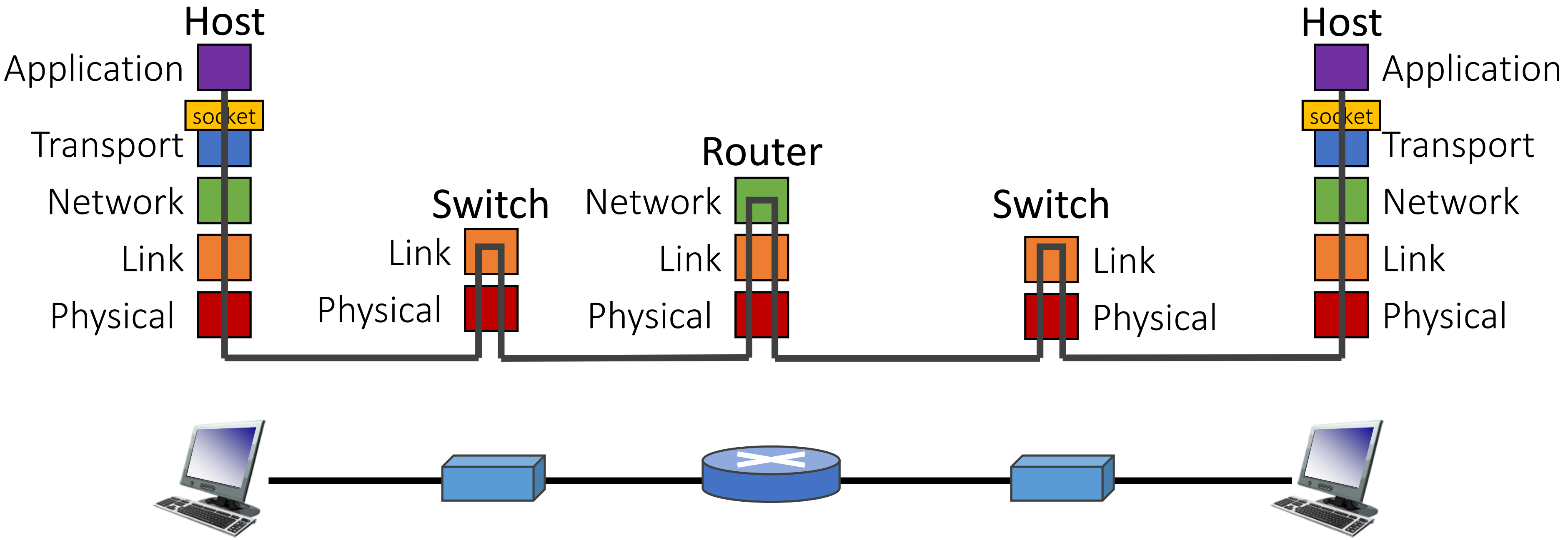
As little as possible

The network layer is "best-effort" and unreliable


Transport protocol
is reliable
Application
process
Transport
Network
I am reliable, you
can count on me
Oh $#!7, what
am I going to do?
Application
process
Transport
Network
I'll do my best, but no
guarantees

Transport vs. Network Layer

- Transport layer resides on end hosts and provides process-to-process communication.
- Network layer provides host-to-host, best-effort and unreliable communication.



Question:

How do you build a reliable transport protocol over an unreliable channel?

Sending Data Reliably is a lot harder than you think


YOUR
HEAD
ASPLODE

What can happen in an unreliable channel?

Reliable Transfer over Unreliable Channel

Underlying network may

- corrupt packets
- drop packets
- re-order packets
- deliver packets after an arbitrarily long delay

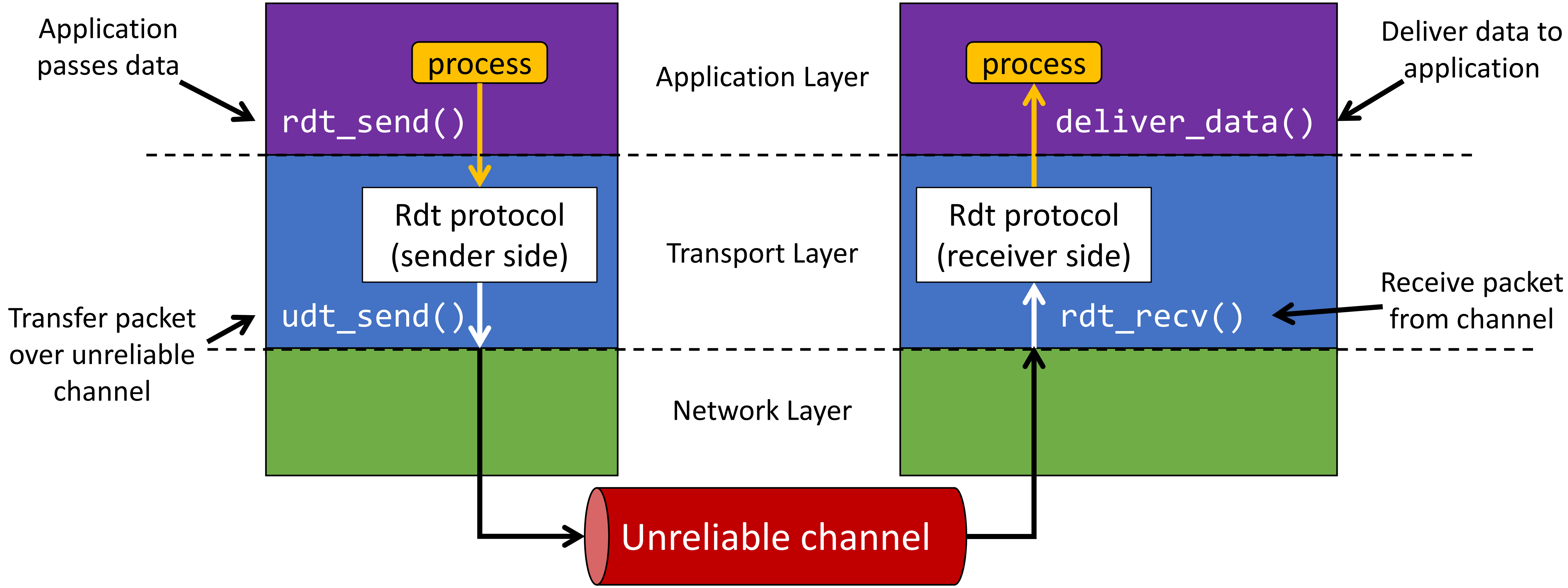
End-to-end reliable transport service should

- guarantee packets delivery and correctness
- deliver packets (to application) in the same order they are sent

Reliable Delivery Transfer

(rdt)

RDT Model



Application passes data
rdt_send()
process
Application Layer
process
deliver_data()
Deliver data to application
Rdt protocol (sender side)
Transport Layer
Rdt protocol (receiver side)
Receive packet from channel
Transfer packet over unreliable channel
udt_send()
rdt_recv()
Network Layer
Unreliable channel

How complex the rdt protocol is, is determined by how unreliable the channel is

We will incrementally develop rdt protocols, with increasing unreliability of the channel

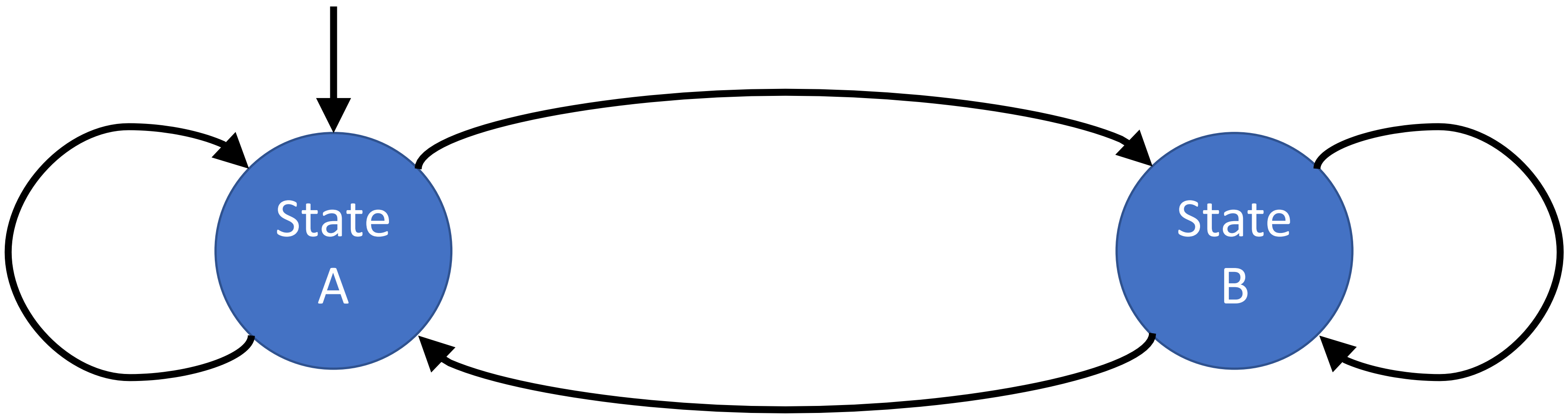
We will only consider one direction of data transfer

But control information may flow both ways

Finite State Machine

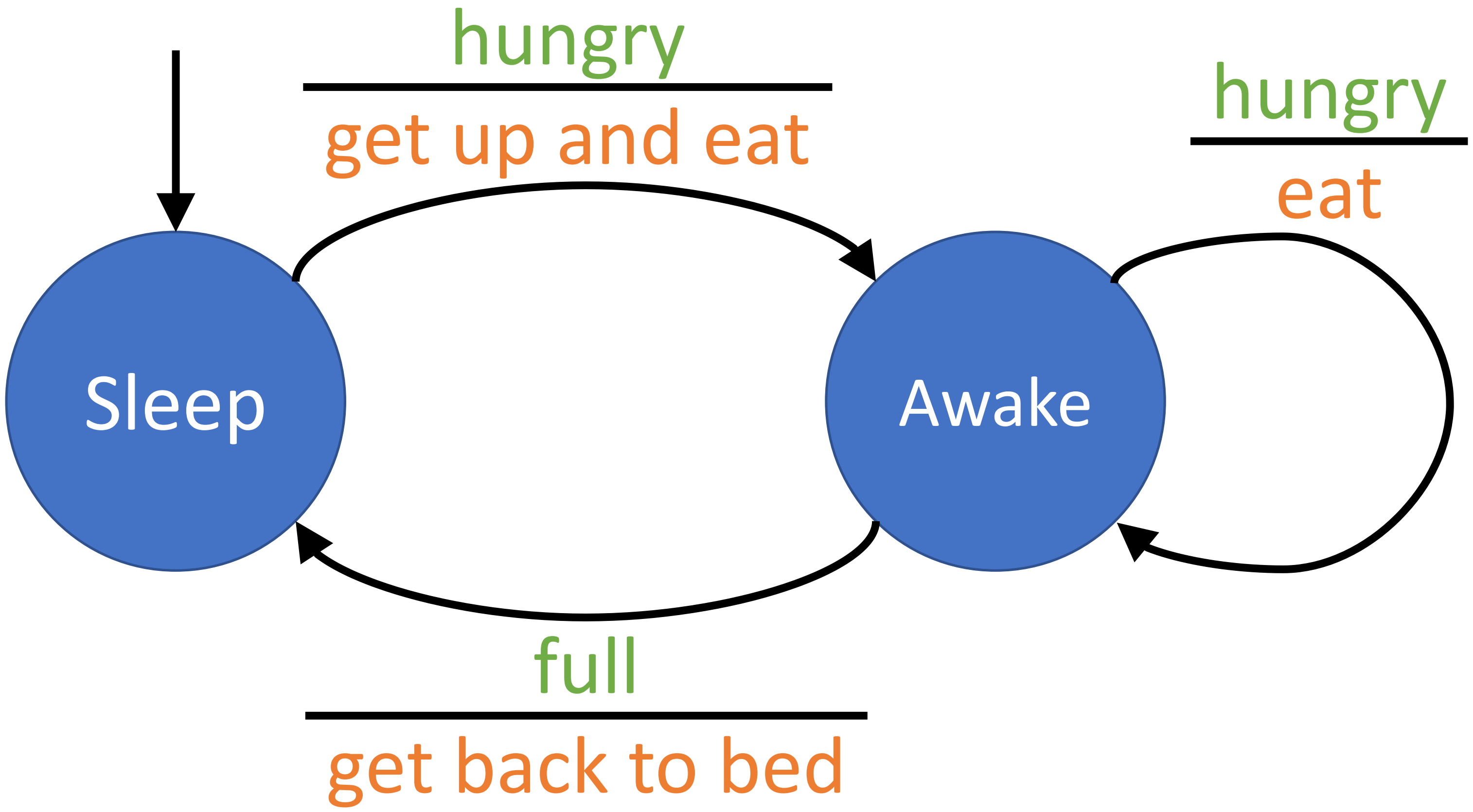
Used to describe sender and receiver of a protocol

- We will learn a protocol by examples, but FSM provides you the complete picture to refer to as necessary.



Event causing transition
___
Action taken on transition

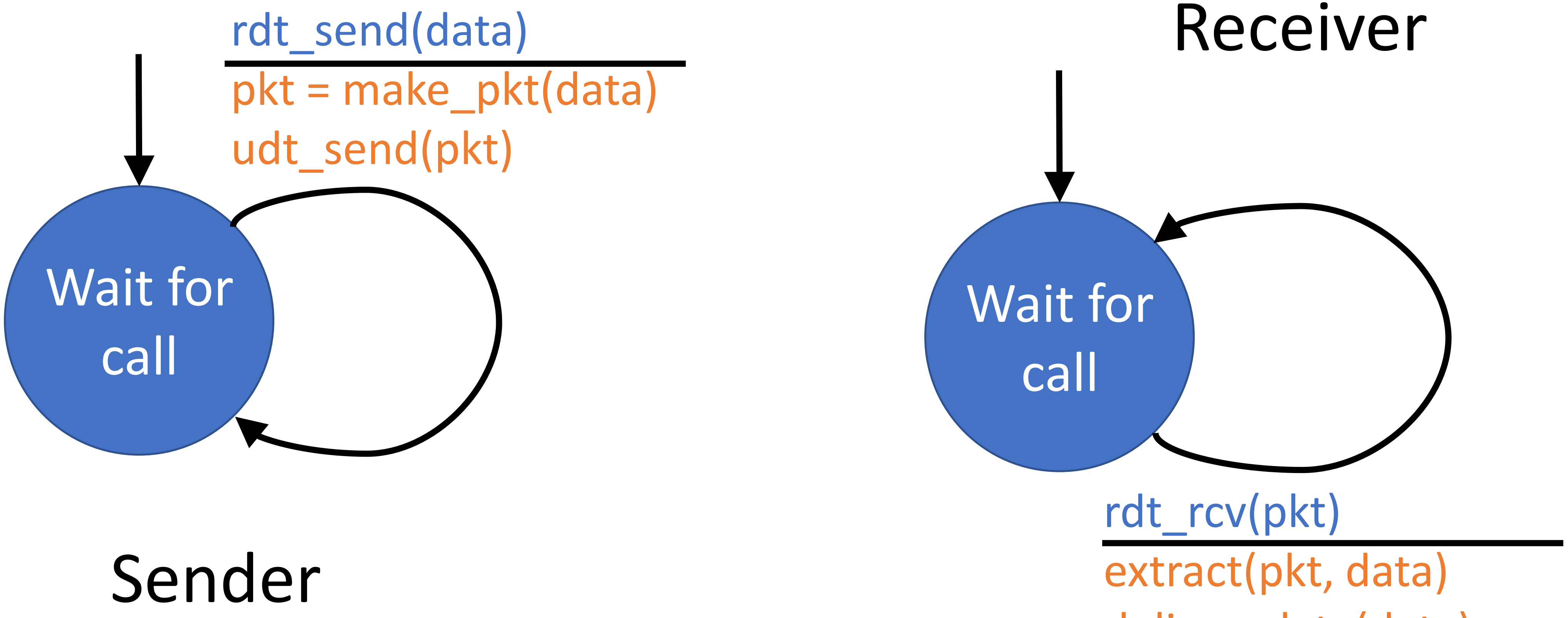
Example FSM


hungry
get up and eat
hungry
eat
Sleep
Awake
full
get back to bed

rdt 1.0

Assume underlying channel is reliable

rdt 1.0


rdt_send(data)
pkt = make_pkt(data)
udt_send(pkt)
Wait for call
Sender
Receiver
Wait for call
rdt_rcv(pkt)
extract(pkt, data)
deliver_data(data)

rdt 2.0: Channel with Bit Errors

Assumption:

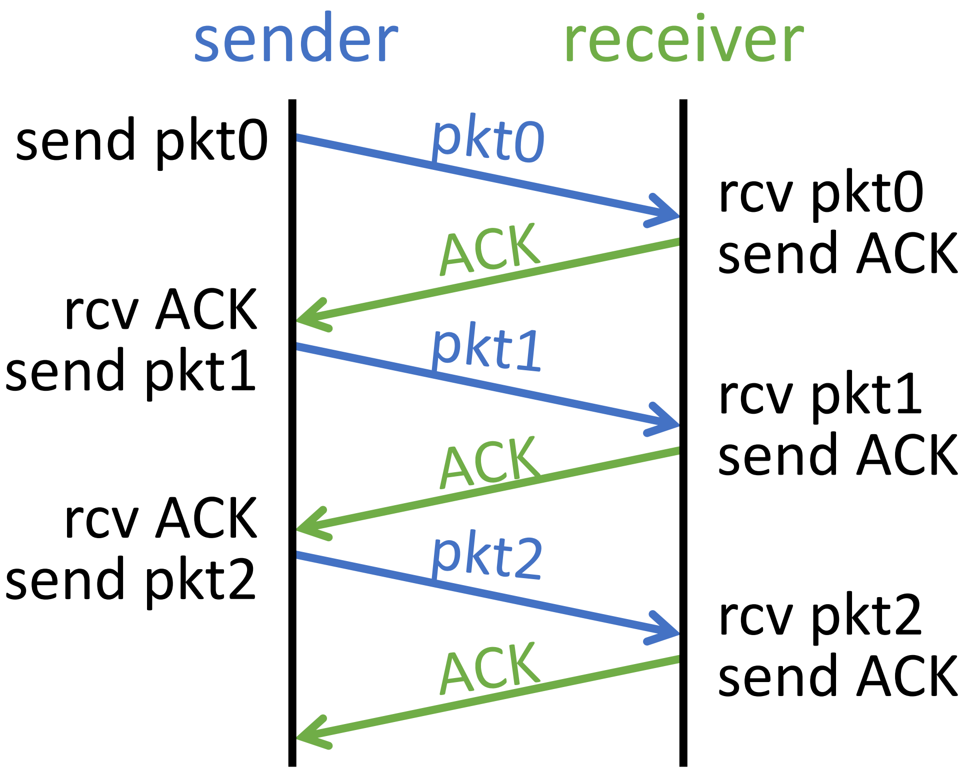
- underlying channel may flip bits in packets
- other than that, the channel is perfect

Receiver may use checksum to detect bit errors

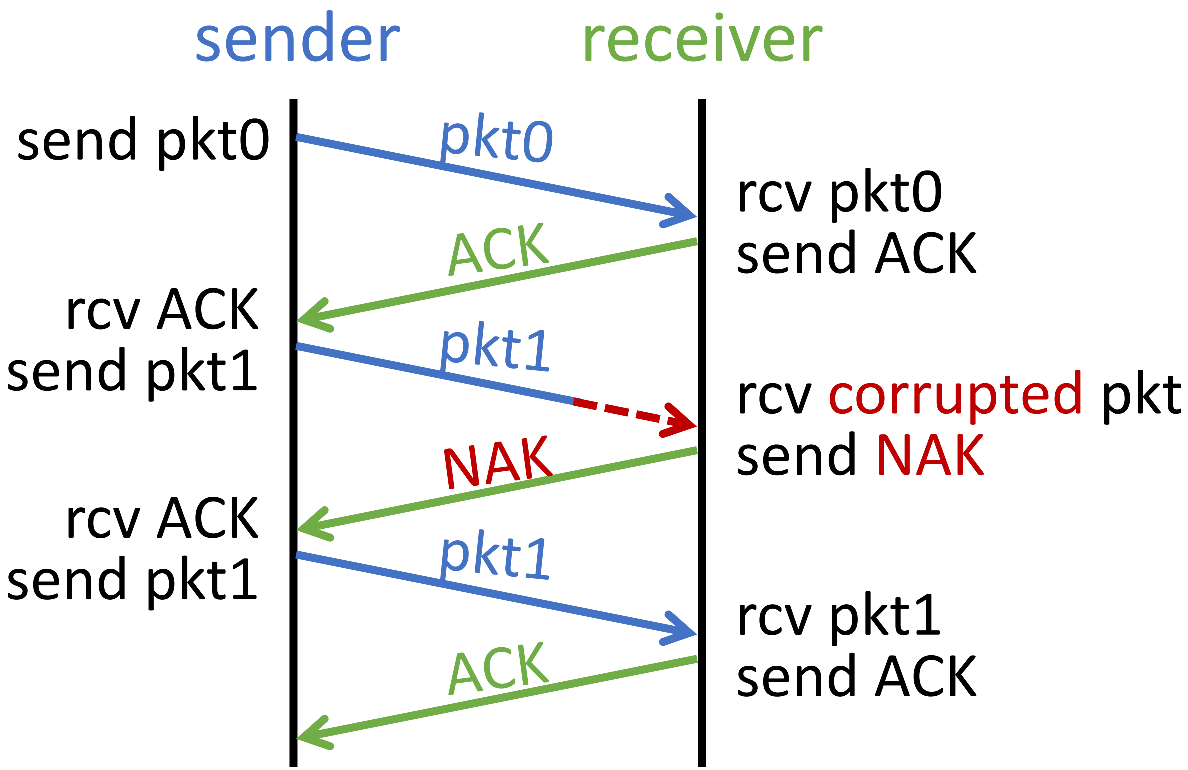
Question: how to recover from bit errors?

- Acknowledgements (ACKs): receiver explicitly tells sender that packet received is OK.
- Negative acknowledgements (NAKs): receiver explicitly tells sender that packet has errors.
- Sender retransmits packet on receipt of NAK.

rdt 2.0 in action



no bit error

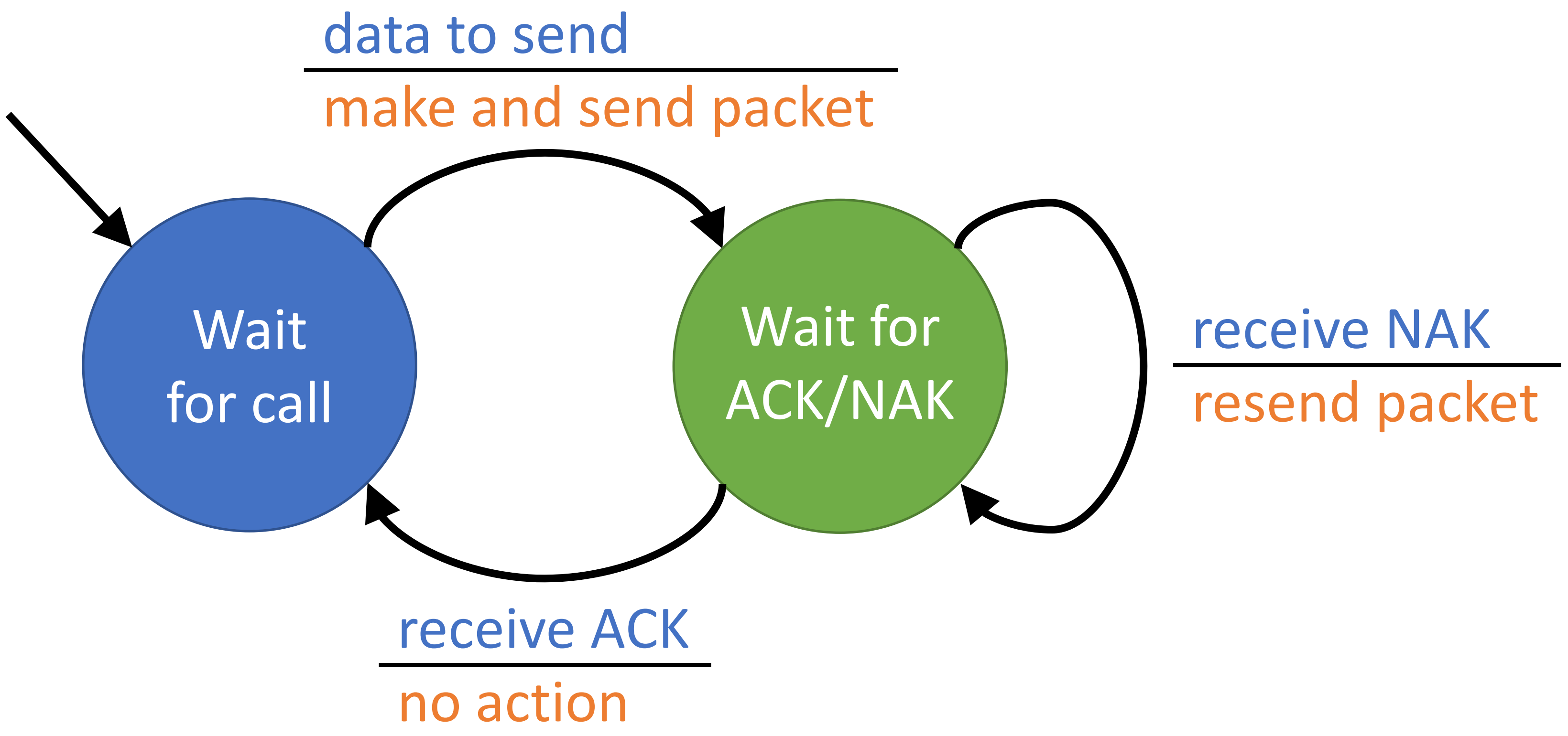


with bit error

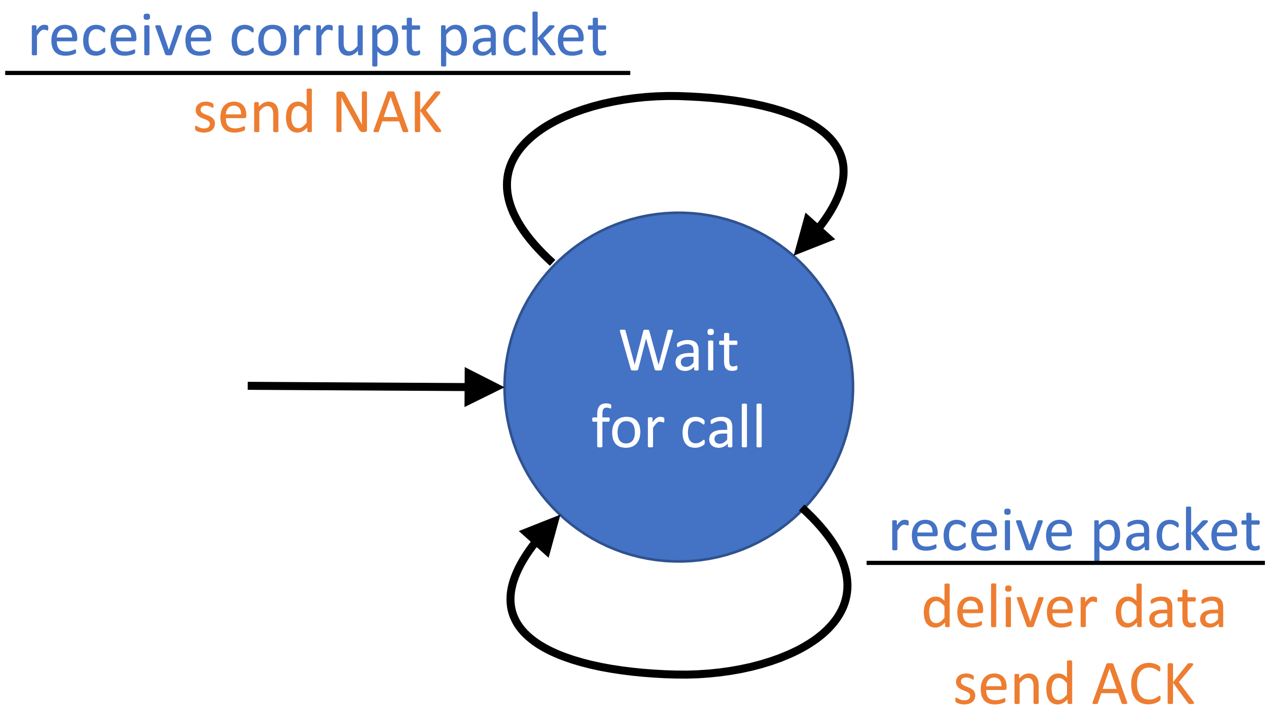
Stop-and-wait protocol

Sender sends one packet at a time, then waits for receiver response

rdt 2.0 sender



rdt 2.0 receiver



Fatal Bug:

What if ACK/NAK is corrupted?

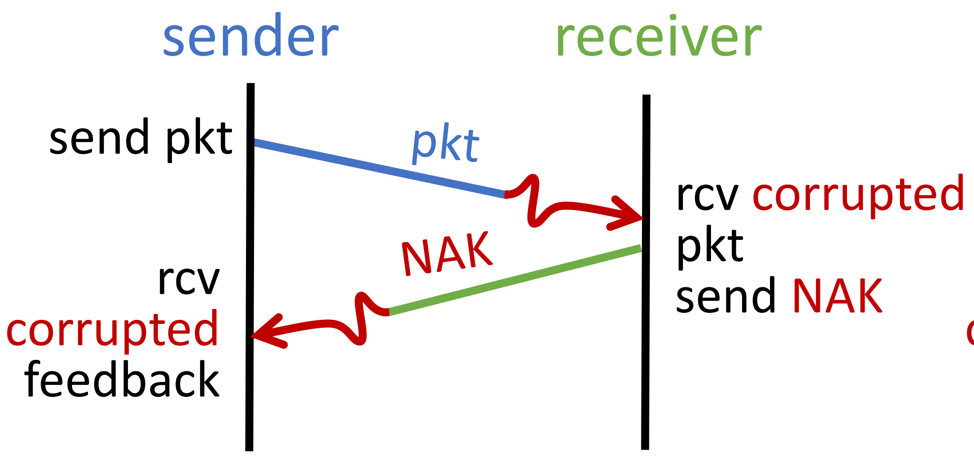
rdt 2.0 has a Fatal Flaw!

What happens if ACK/NAK is corrupted?

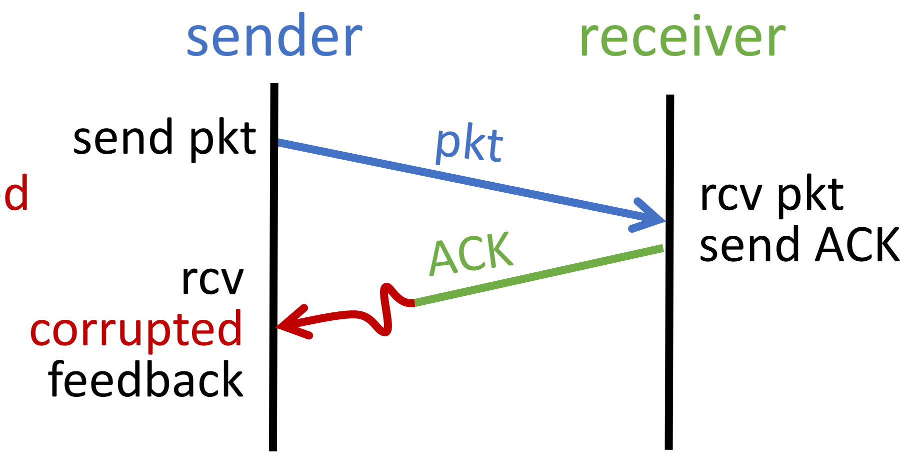
- Sender doesn’t know what happened at receiver!

So what should the sender do?

- Sender just retransmits when receives garbled ACK or NAK.
- Question: does this work?



a) corrupted NAK

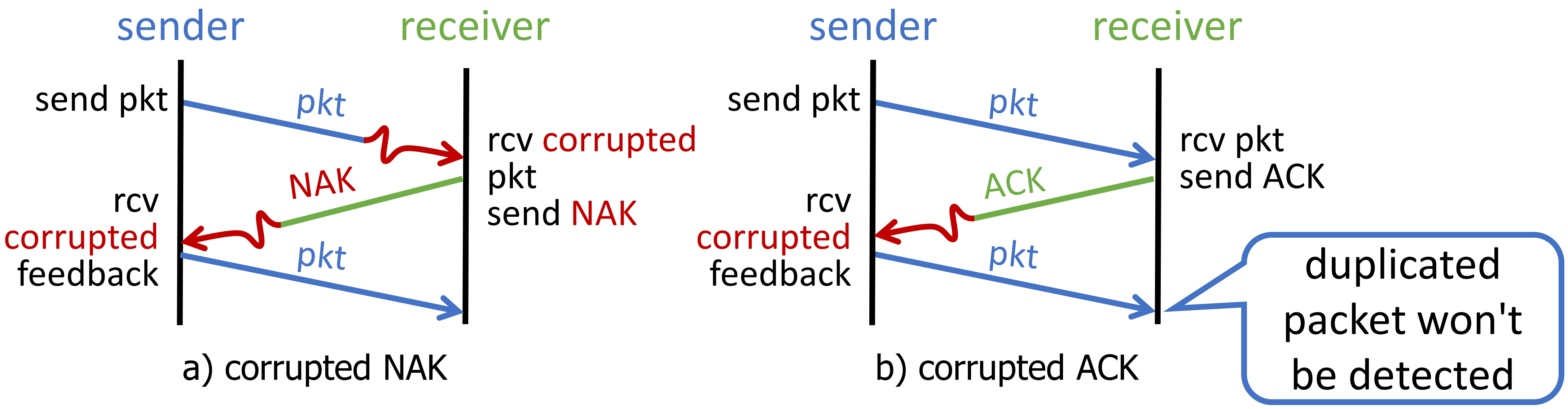


b) corrupted ACK

rdt 2.0 has a Fatal Flaw!

Sender just retransmits when it receives garbled feedback.

- This may cause retransmission of correctly received packet!
- Question: how can receiver identify duplicate packet?

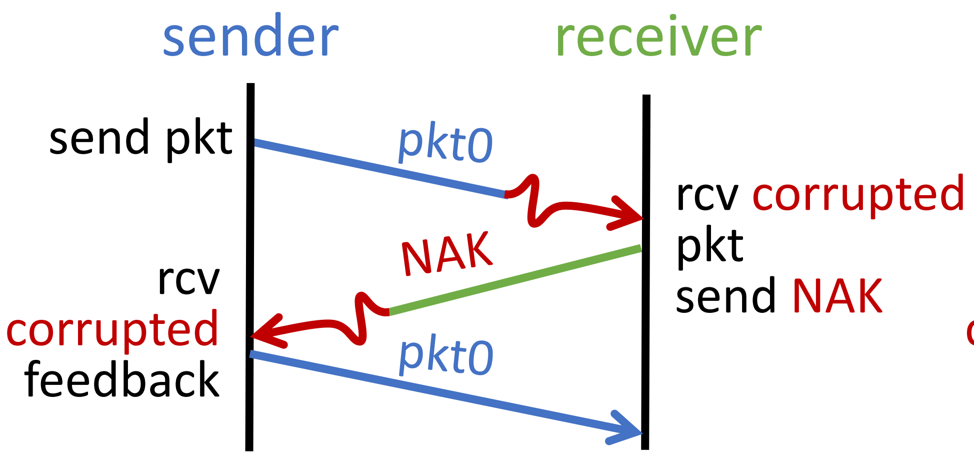


rdt 2.1: Add a sequence number

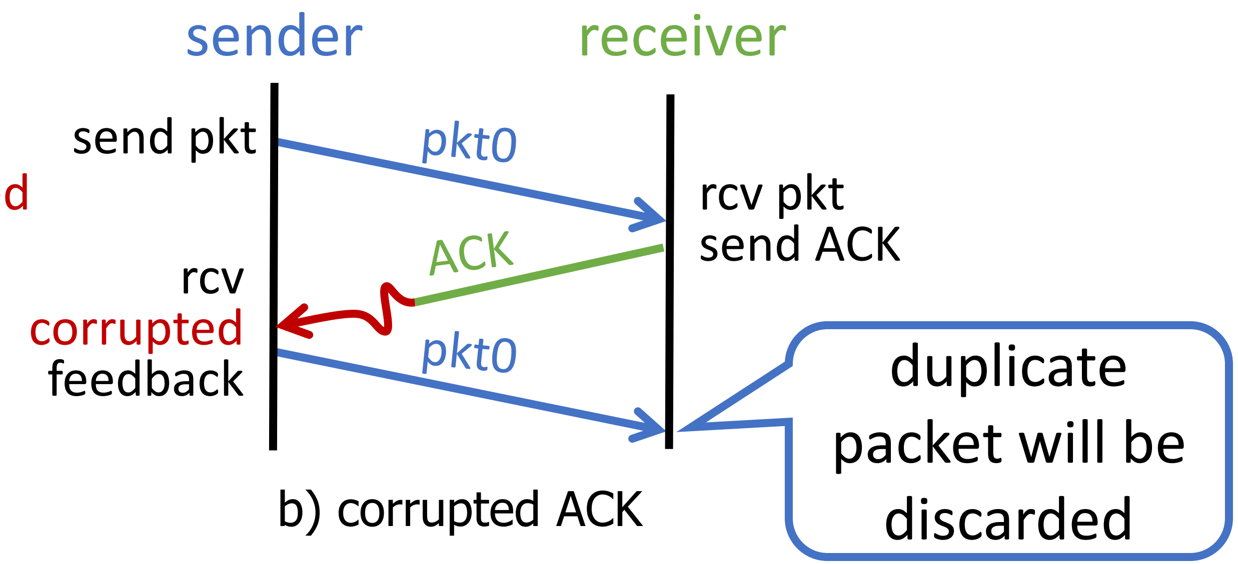
To handle duplicates:

- Sender retransmits current packet if ACK/NAK is garbled.
- Sender adds sequence number to each packet.
- Receiver discards (doesn't deliver up) duplicate packet.

This gives rise to protocol rdt 2.1.

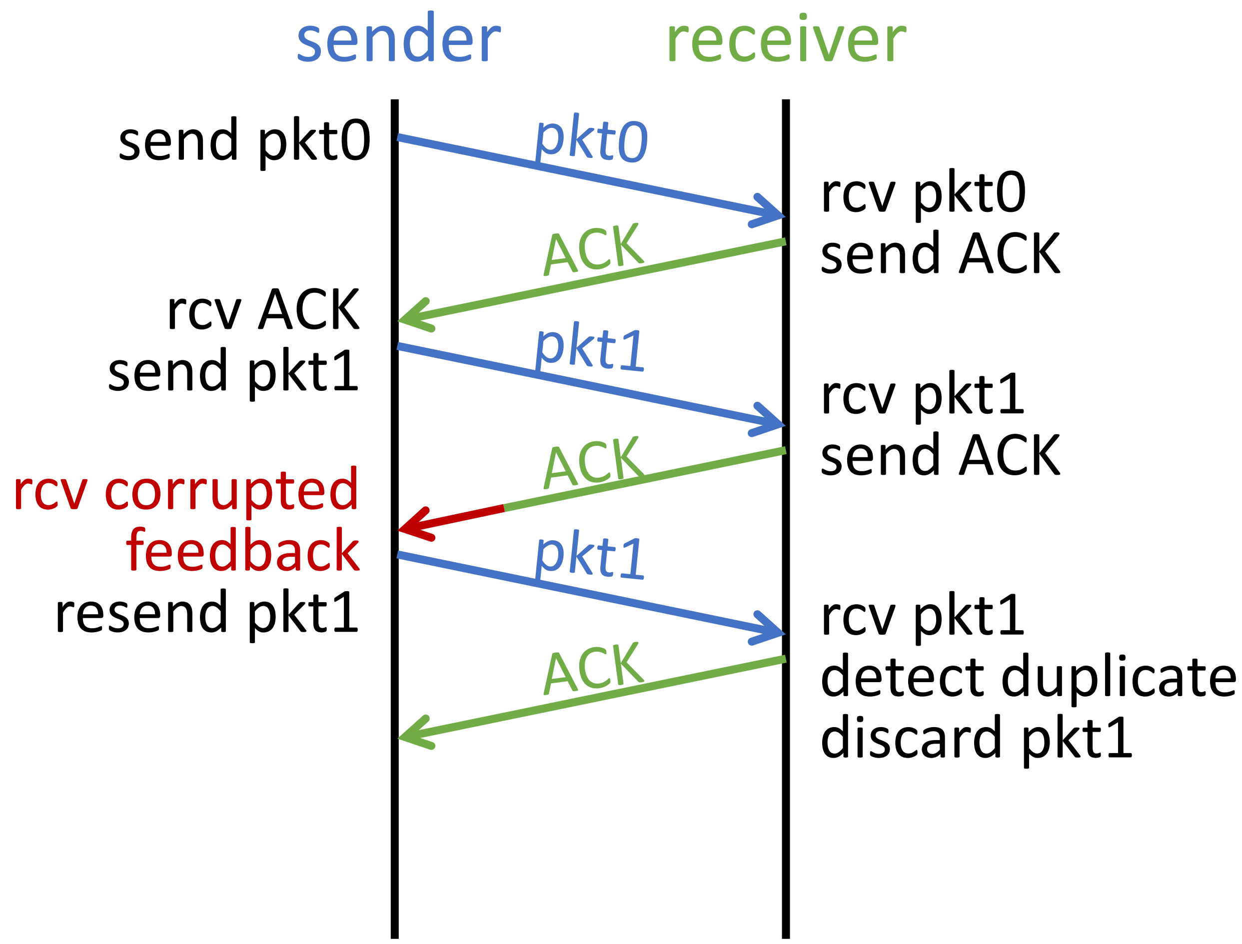


a) corrupted NAK

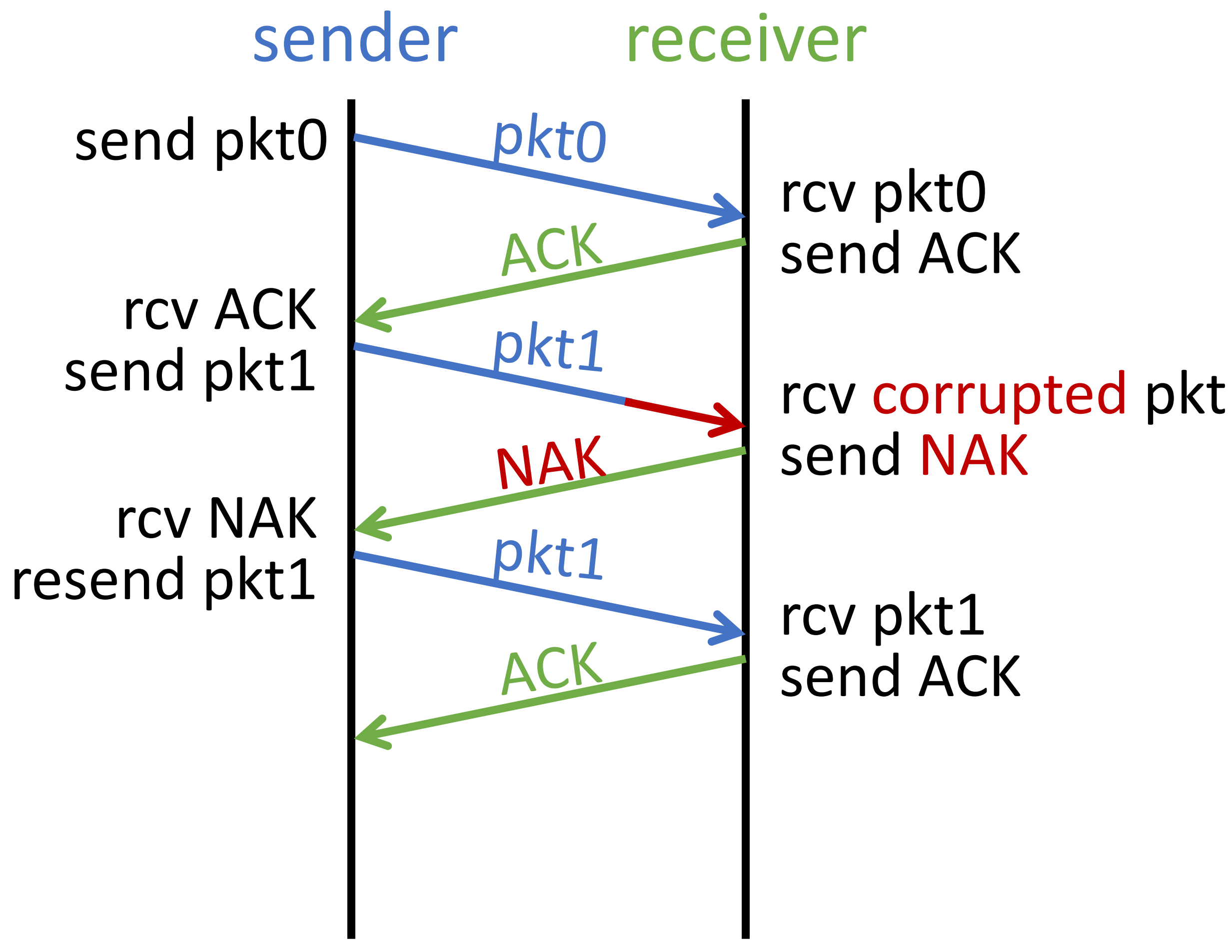


b) corrupted ACK

rdt 2.1 in action

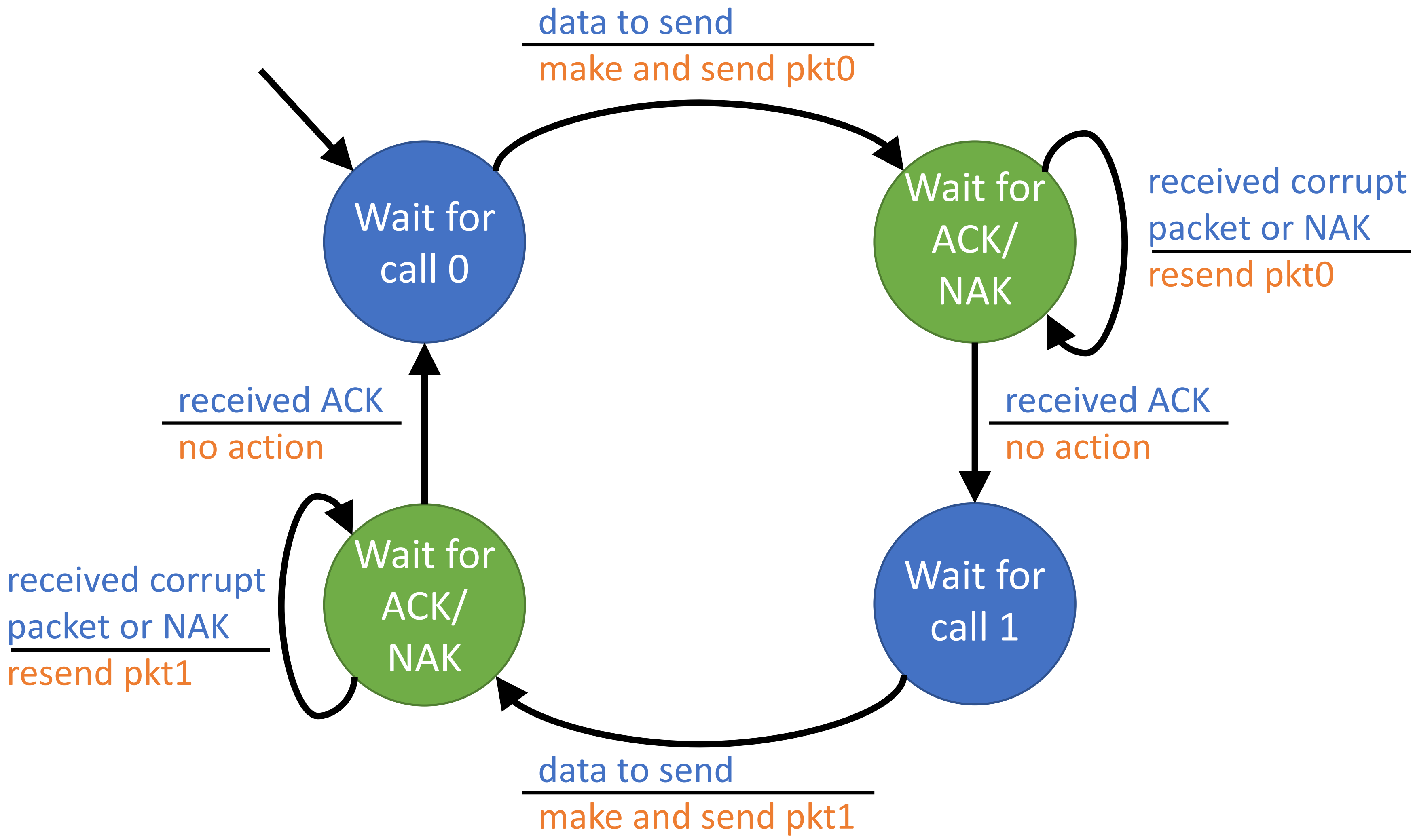


resend due to corrupted ACK

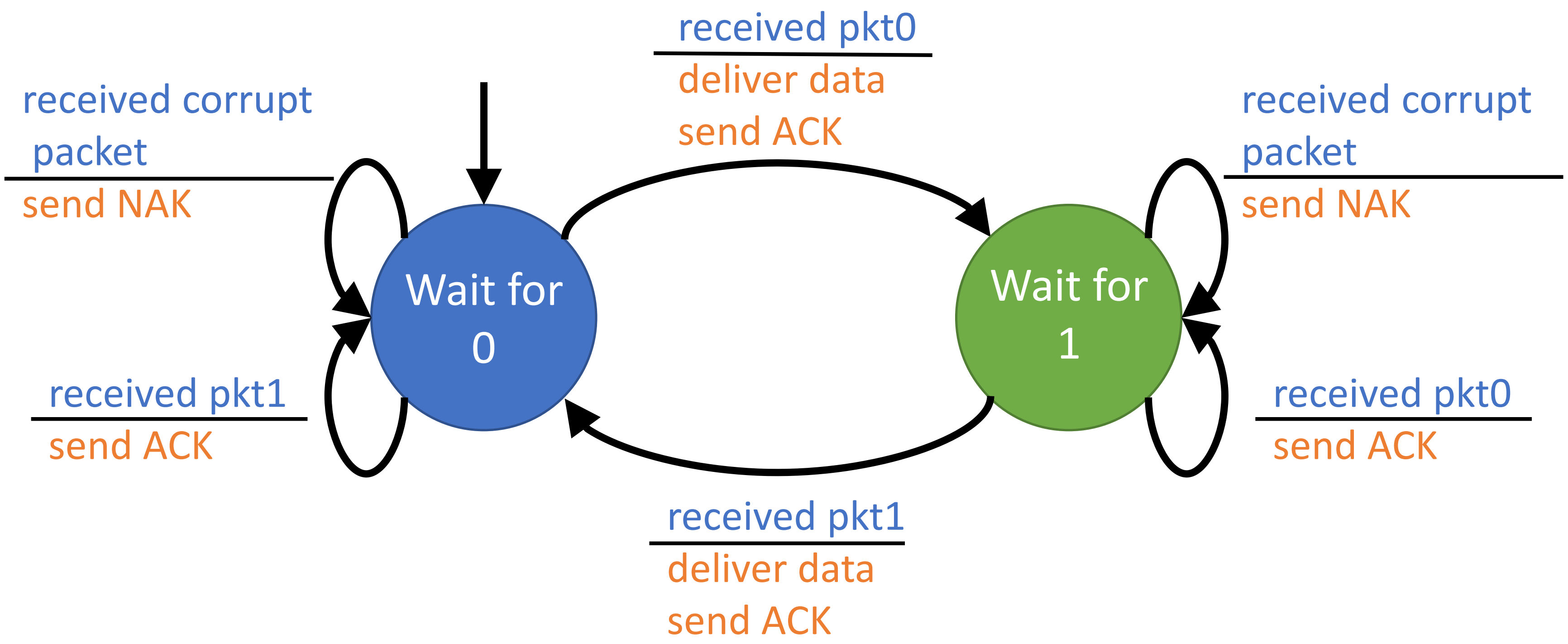


resend due to NAK

rdt 2.1 sender



rdt 2.1 receiver



rdt 2.2

Replace NAK with ACK of last correctly received packet

rdt 2.2: a NAK-free Protocol

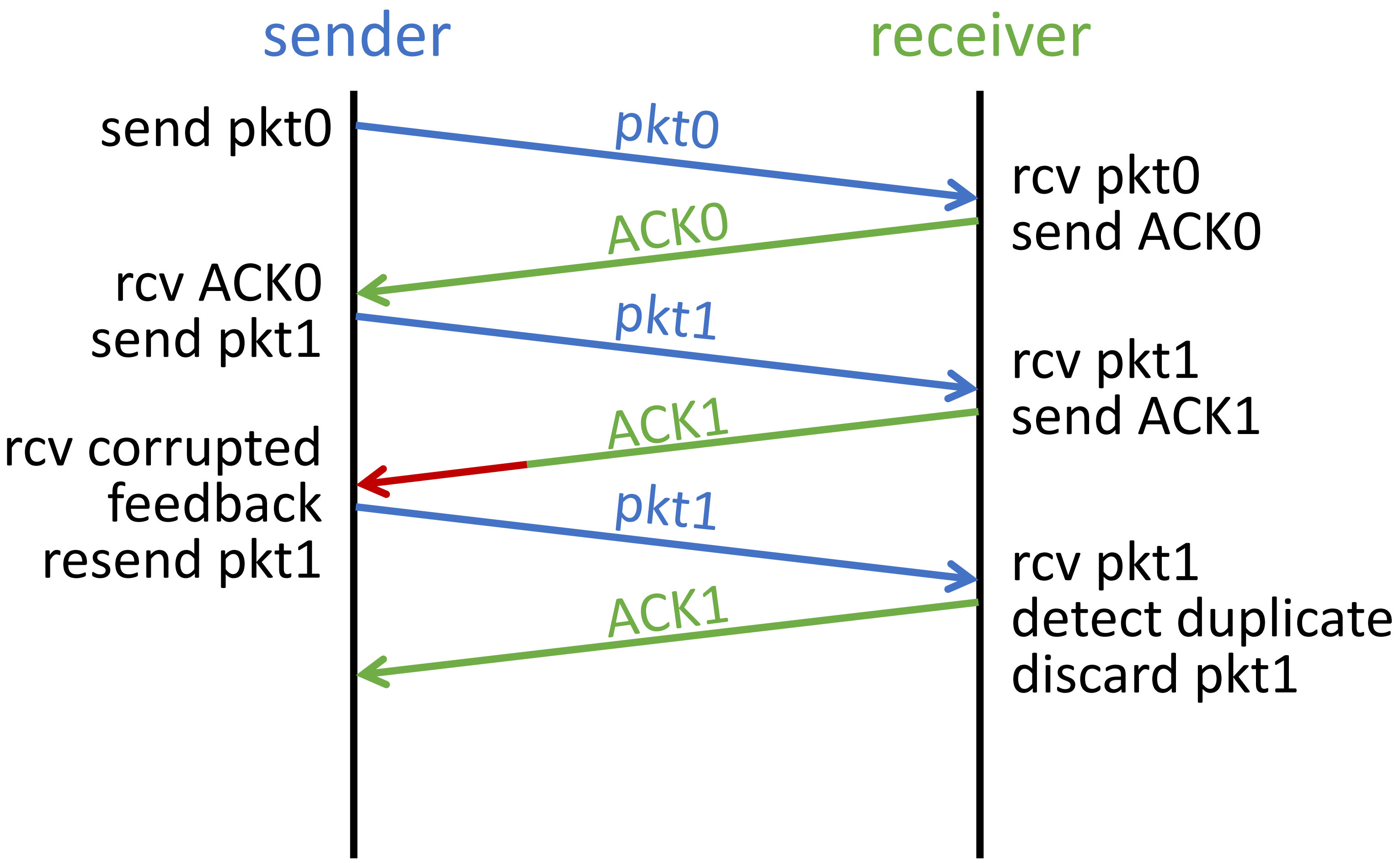
Same assumption and functionality as rdt 2.1, but use ACKs only.

Instead of sending NAK, receiver sends ACK for the last packet received correctly.

- Now receiver must explicitly include seq. # of the packet being ACKed.

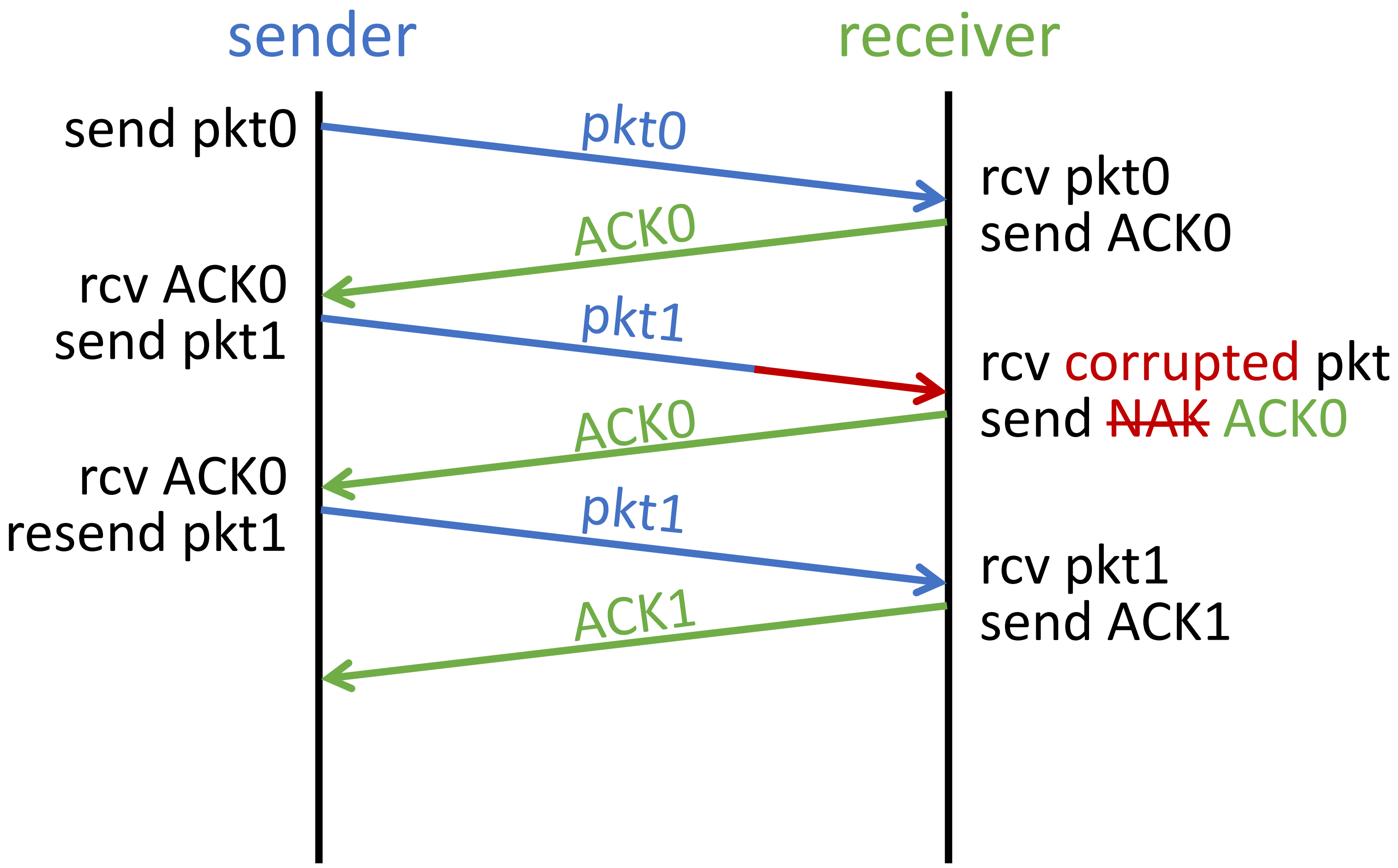
Duplicate ACKs at sender results in same action as NAK: retransmit current pkt

rdt 2.2 in action



resend due to corrupted ACK

rdt 2.2 in action



resend due to duplicated ACK

Homework

FSM for rdt 2.2

rdt 3.0

Packet can be lost or corrupted

rdt 3.0: Channel with Errors and Loss

Assumption: underlying channel

- may flip bits in packets
- may lose packets
- may incur arbitrarily long packet delay
- but will not re-order packets

Question: how to detect packet loss?

- checksum, ACKs, seq. #, retransmissions will be of help… but not enough

What if the ACK is lost?

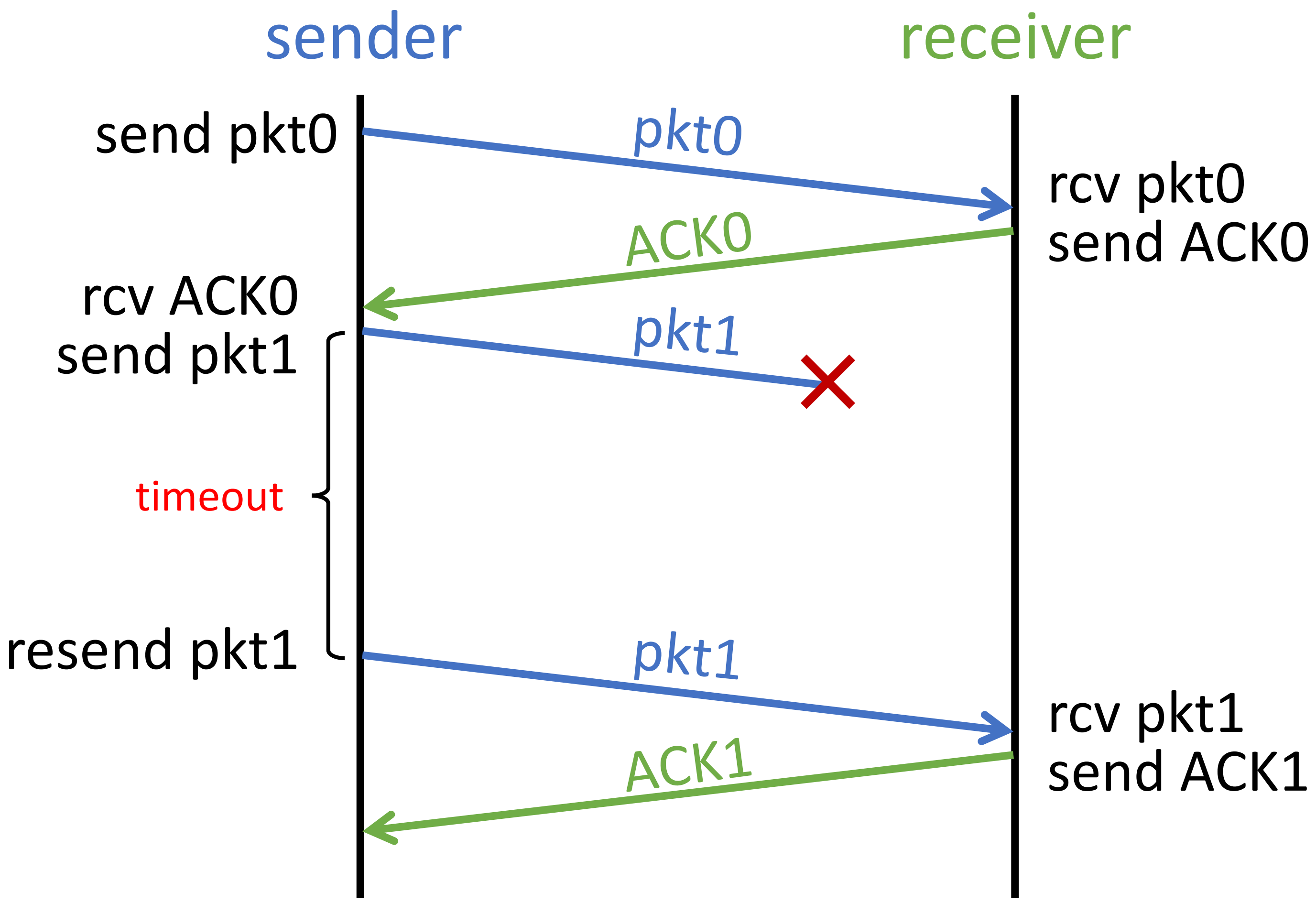
Re-send after waiting some time

rdt 3.0: Channel with Errors and Loss

To handle packet loss:

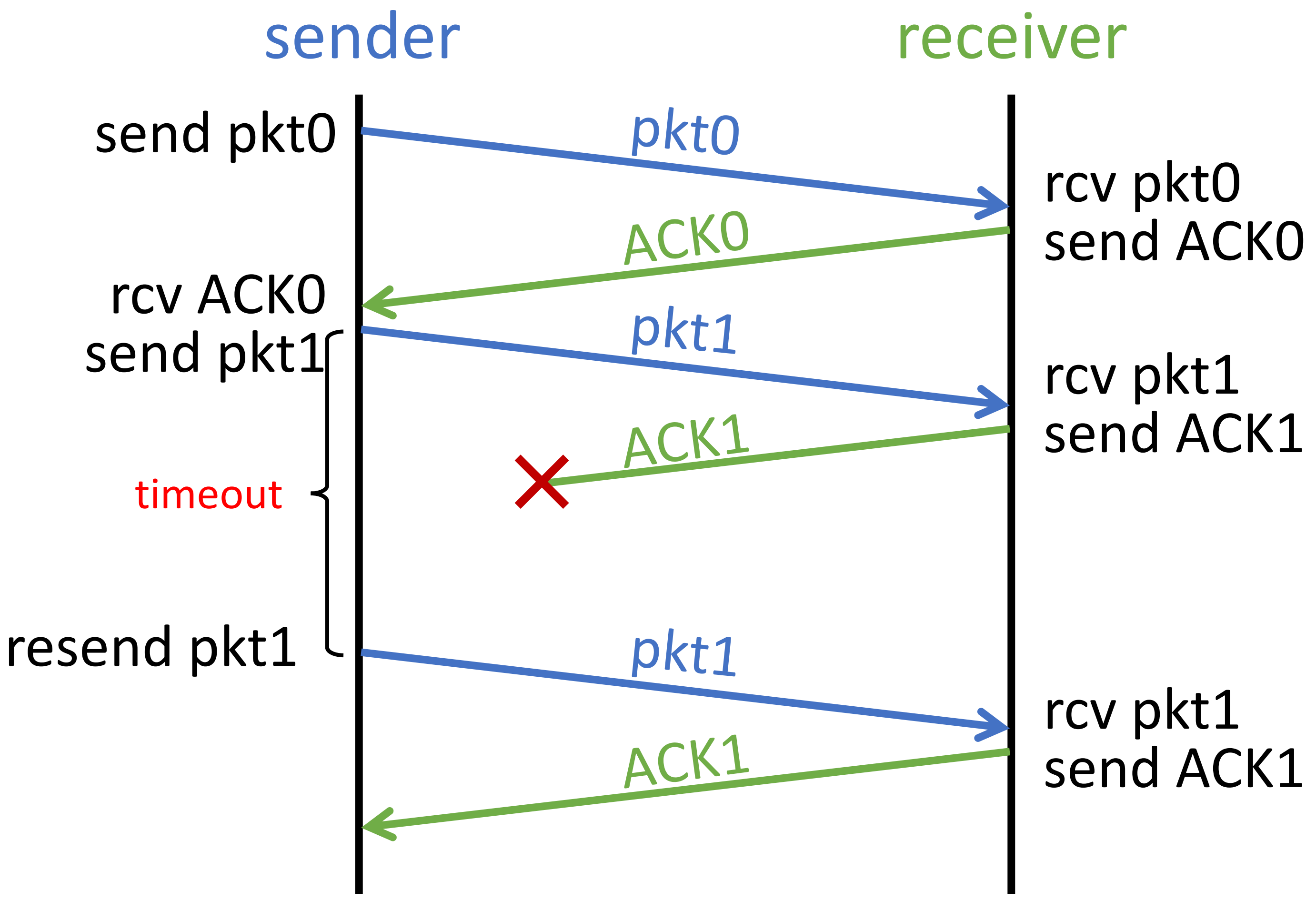
- Sender waits “reasonable” amount of time for ACK.
- Sender retransmits if no ACK is received till timeout.

rdt 3.0 in action



with packet loss

rdt 3.0 in action



with lost ACK

rdt 3.0: Channel with Errors and Loss

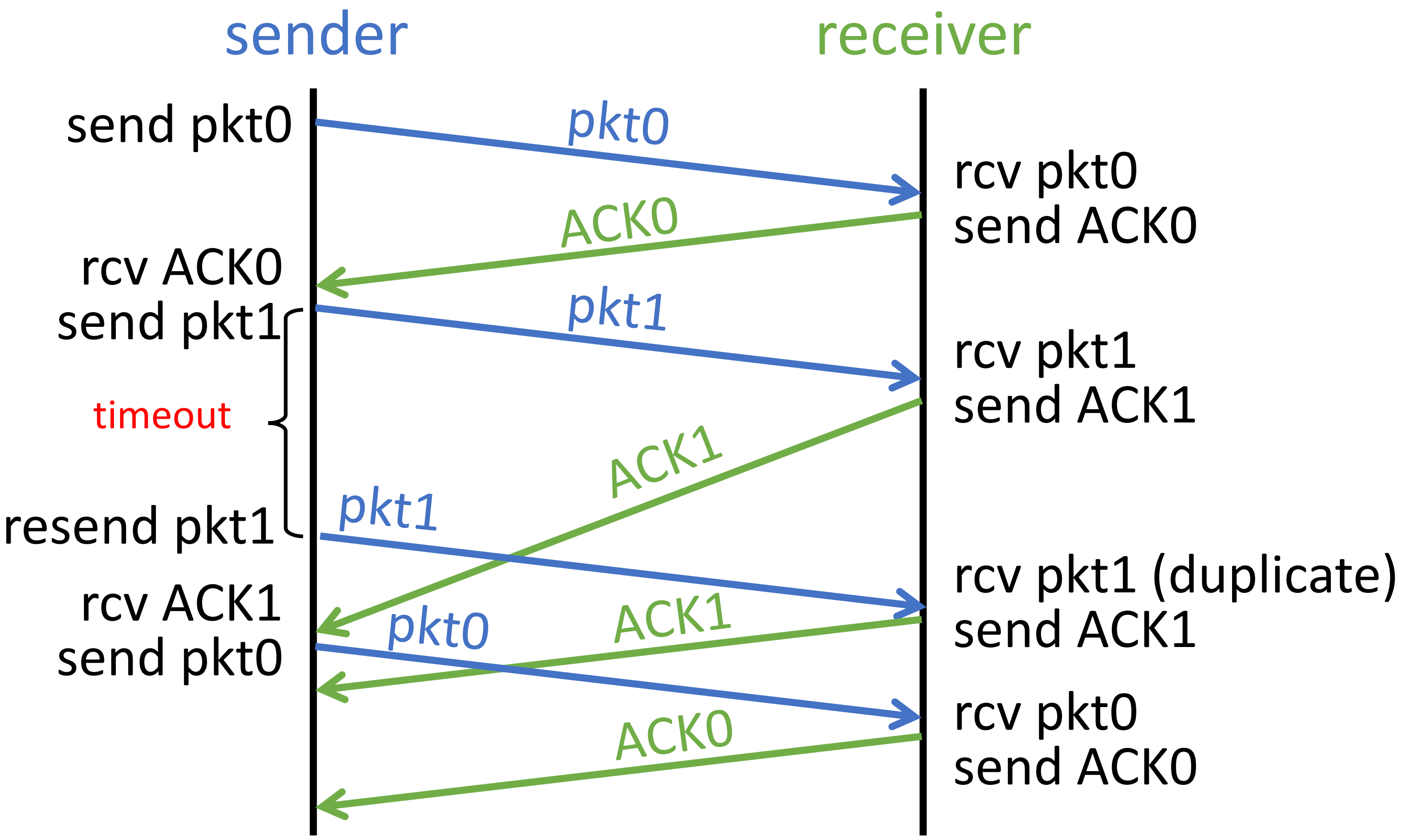
To handle packet loss:

- Sender waits “reasonable” amount of time for ACK.
- Sender retransmits if no ACK is received till timeout.

Question: what if packet (or ACK) is just delayed, but not lost?

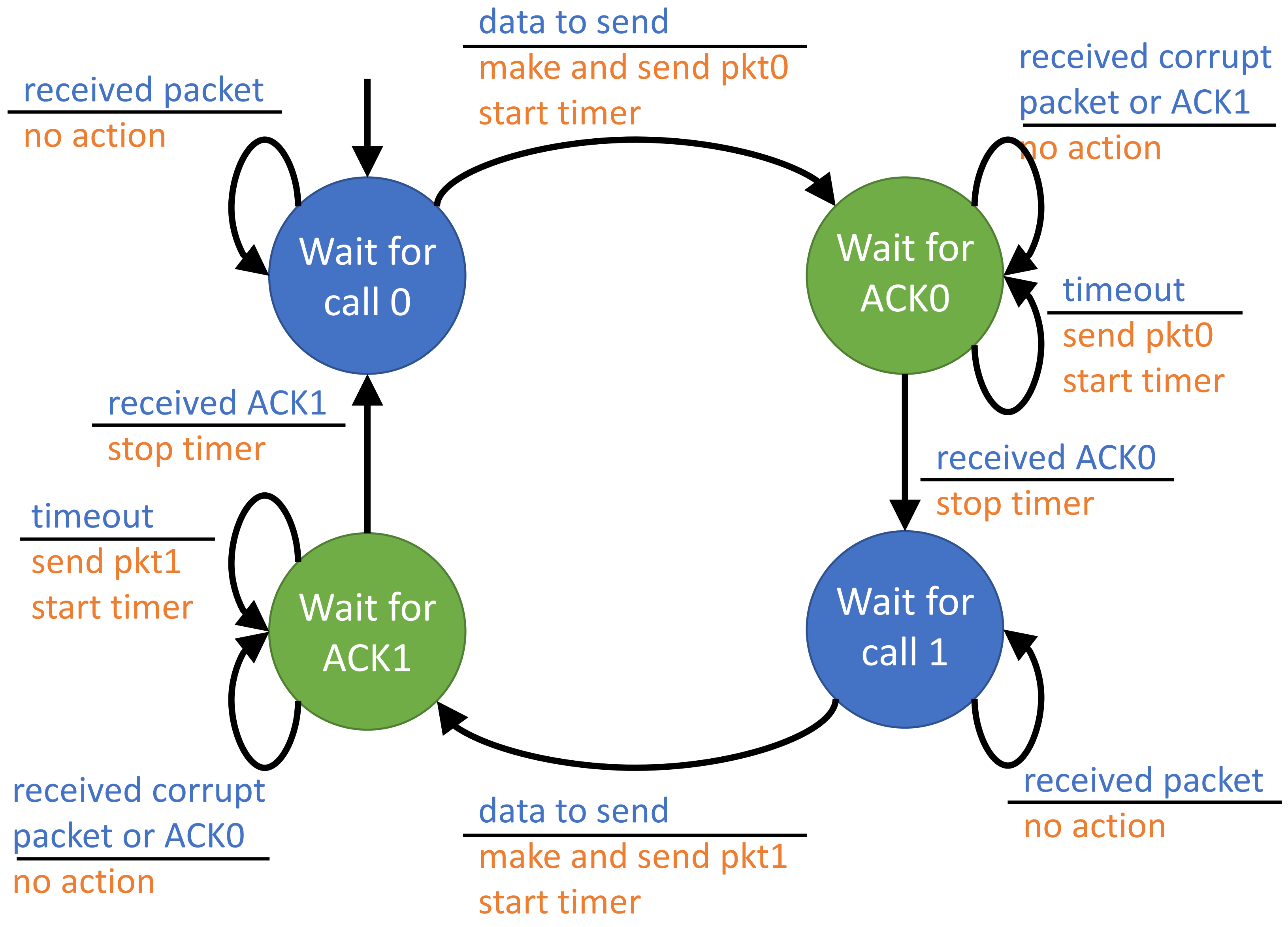
- Timeout will trigger retransmission.
- Retransmission will generate duplicates in this case, but receiver may use seq. # to detect it.
- Receiver must specify seq. # of the packet being ACKed (check scenario (d) two pages later).

rdt 3.0 in action



with premature timeout or delayed ACK

rdt 3.0 sender



Homework

FSM for rdt 3.0 receiver

Alternating-bit protocol

rdt 3.0 is still not perfect

<break>

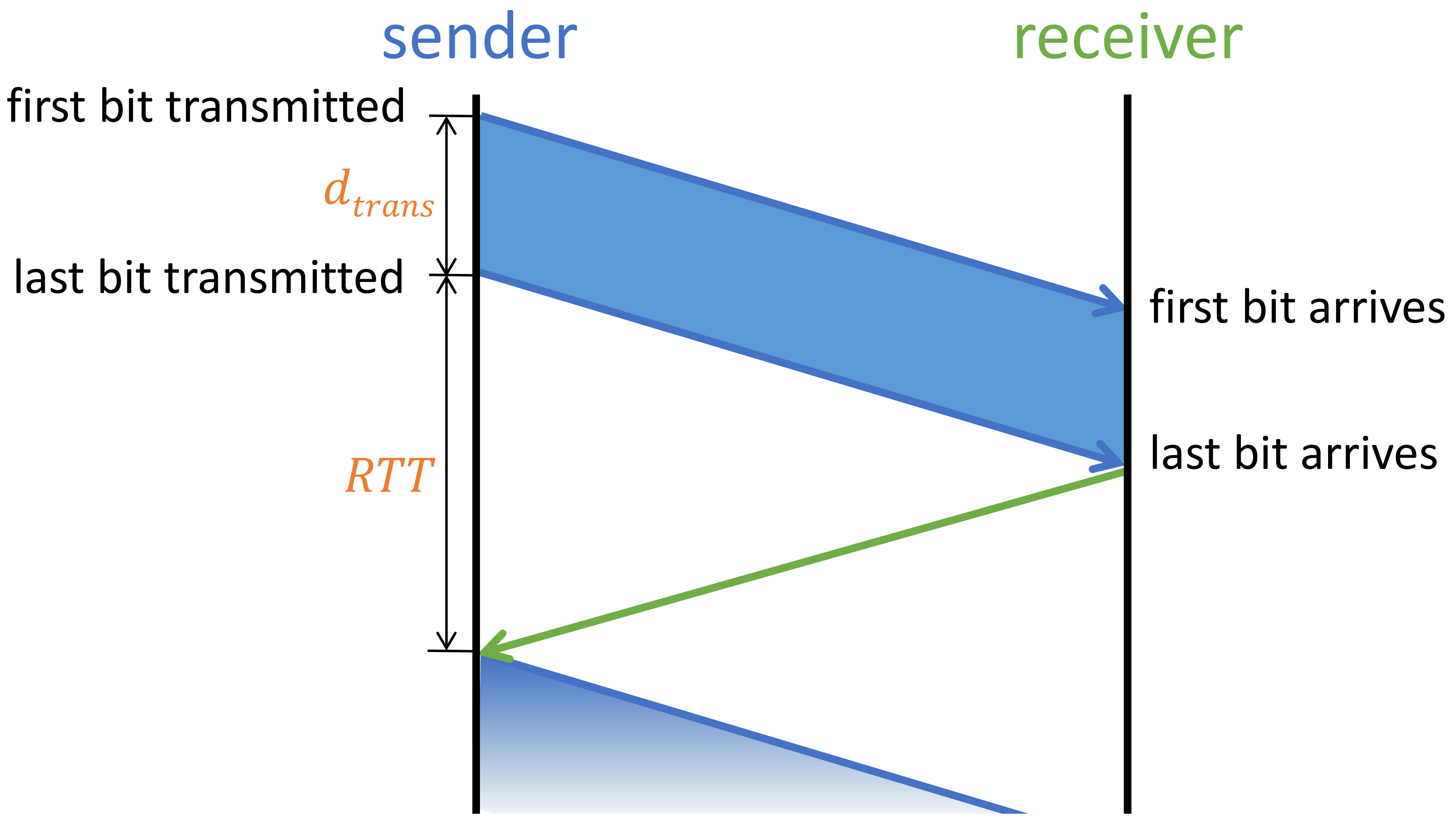
Our rdt protocols have bad performance

Utilization

The fraction of time the link is actually being used

$$\frac{Time\ spent\ sending}{Total\ time}$$

Problem with Stop-and-wait



$$U_{sender} = \frac{Time\ sending}{Total\ time} = \frac{d_{trans}}{d_{trans} + RTT}$$

Example

$$RTT = 30\, ms$$

$$R = 1\, Gb/s$$

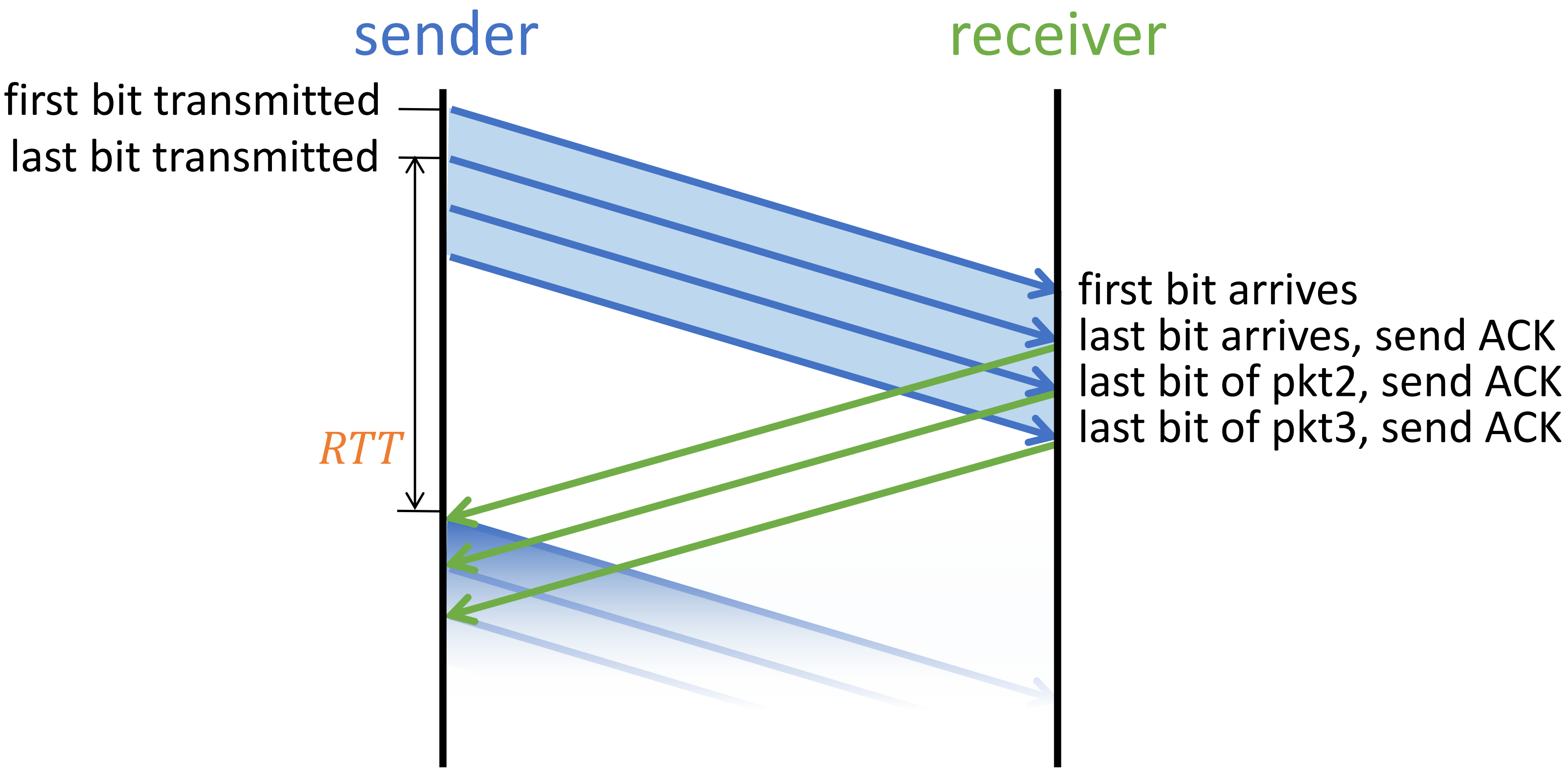
$$L = 8000\, bits$$

$$d_{trans} = {}^{L}/_{R} = 0.008\, ms$$

$$\text{throughput} = \frac{L}{RTT + d_{trans}} = \frac{8000}{30.008} = 267 kbps$$

$$U_{sender} = \frac{d_{trans}}{RTT + d_{trans}} = \frac{0.008}{30+0.008} = 0.00027$$

Pipelining

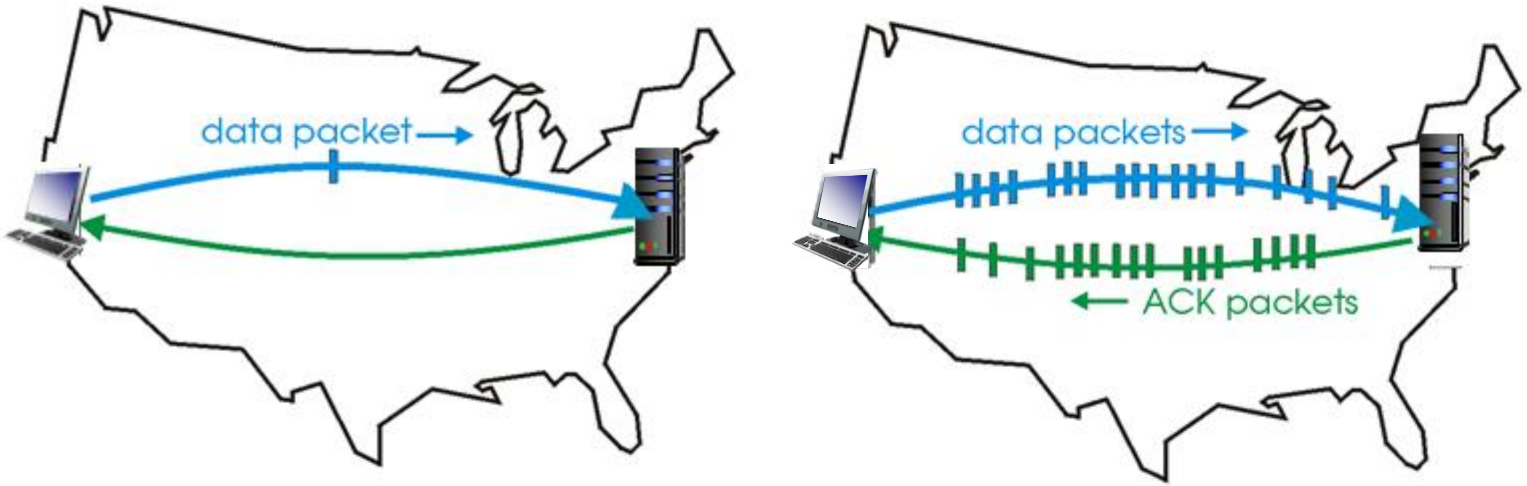


$$U_{sender} = \frac{3 \times d_{trans}}{d_{trans} + RTT} = \frac{3 \times 0.008}{30 + 0.008} = 0.00081$$

Pipelined Protocols

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets.

- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

Benchmark Pipelined Protocols

Two generic forms of pipelined protocols:

1. Go-Back-N
2. Selective repeat

Assumption (same as rdt 3.0): underlying channel

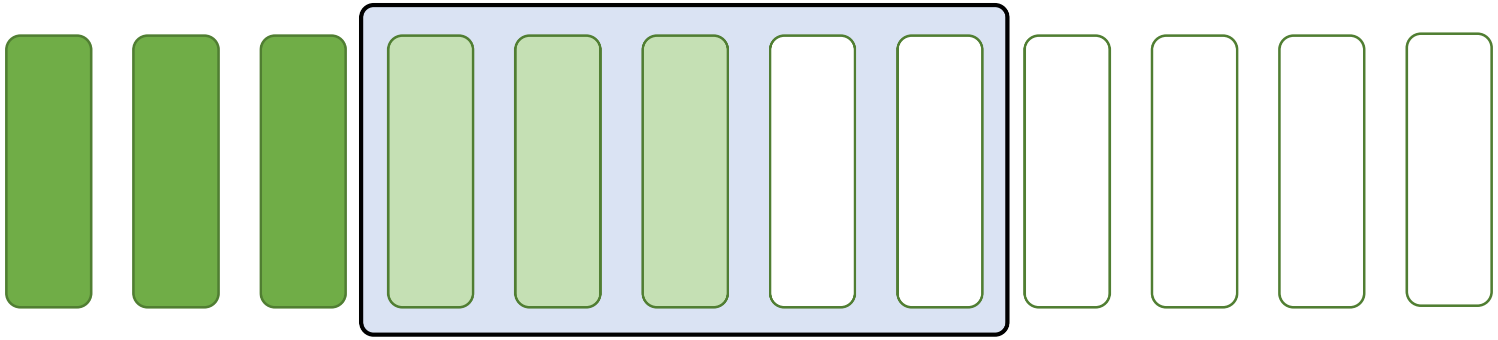
- may flip bits in packets
- may lose packets
- may incur arbitrarily long packet delay
- but will not re-order packets

Go-Back-N

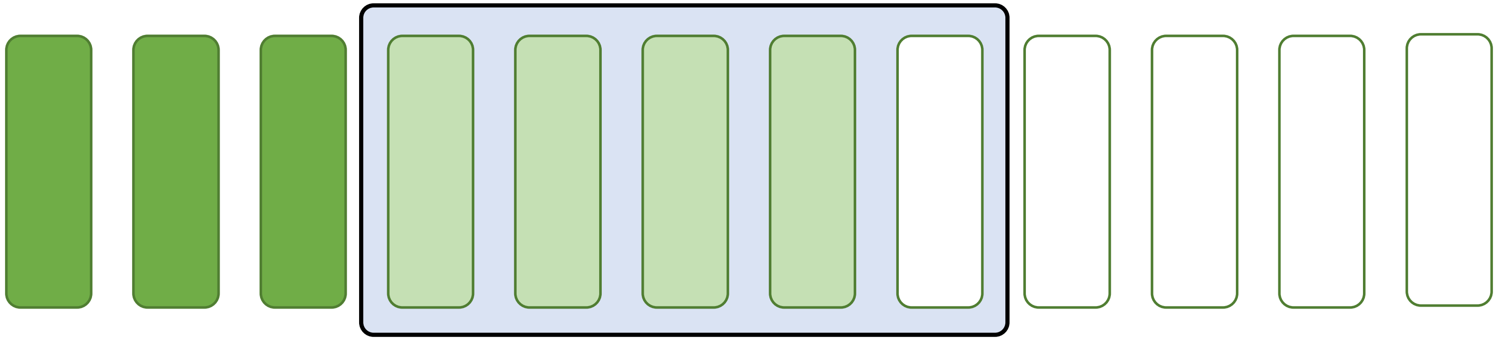
Cumulative ACK

ACK n means all packets $\leq n$ have been received

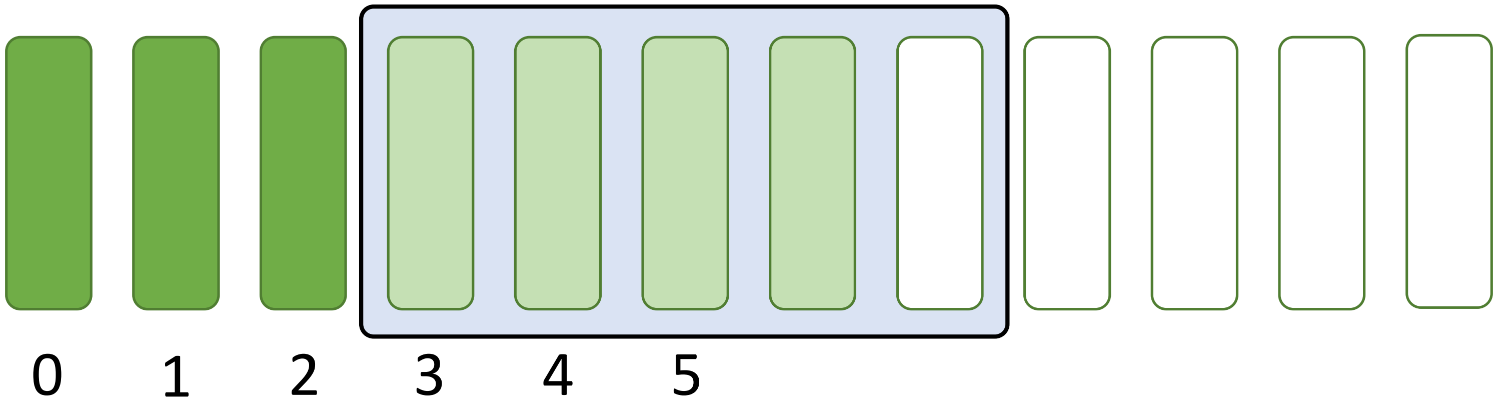
Go-Back-N sender



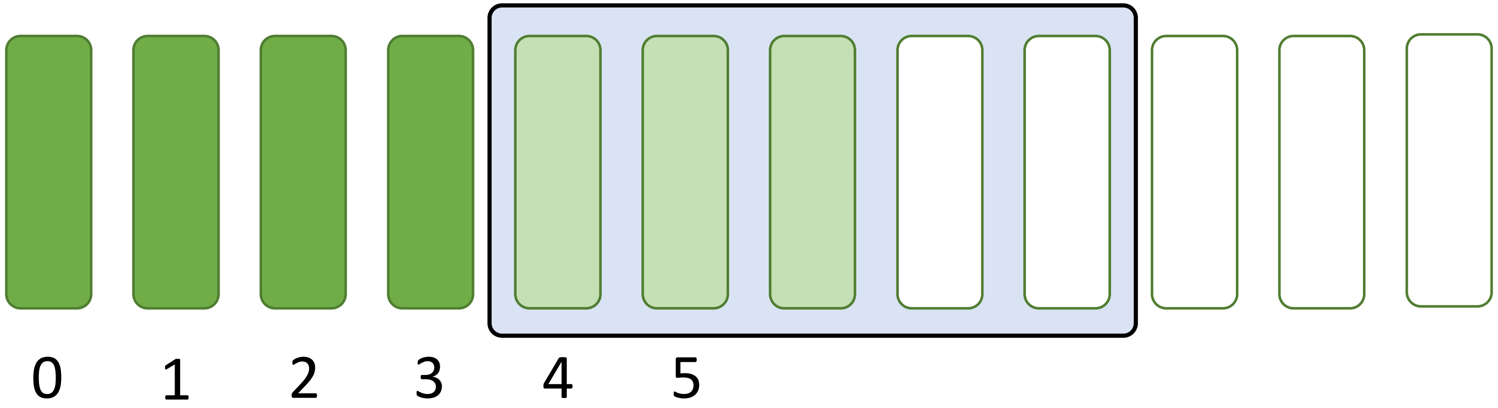
sends a packet



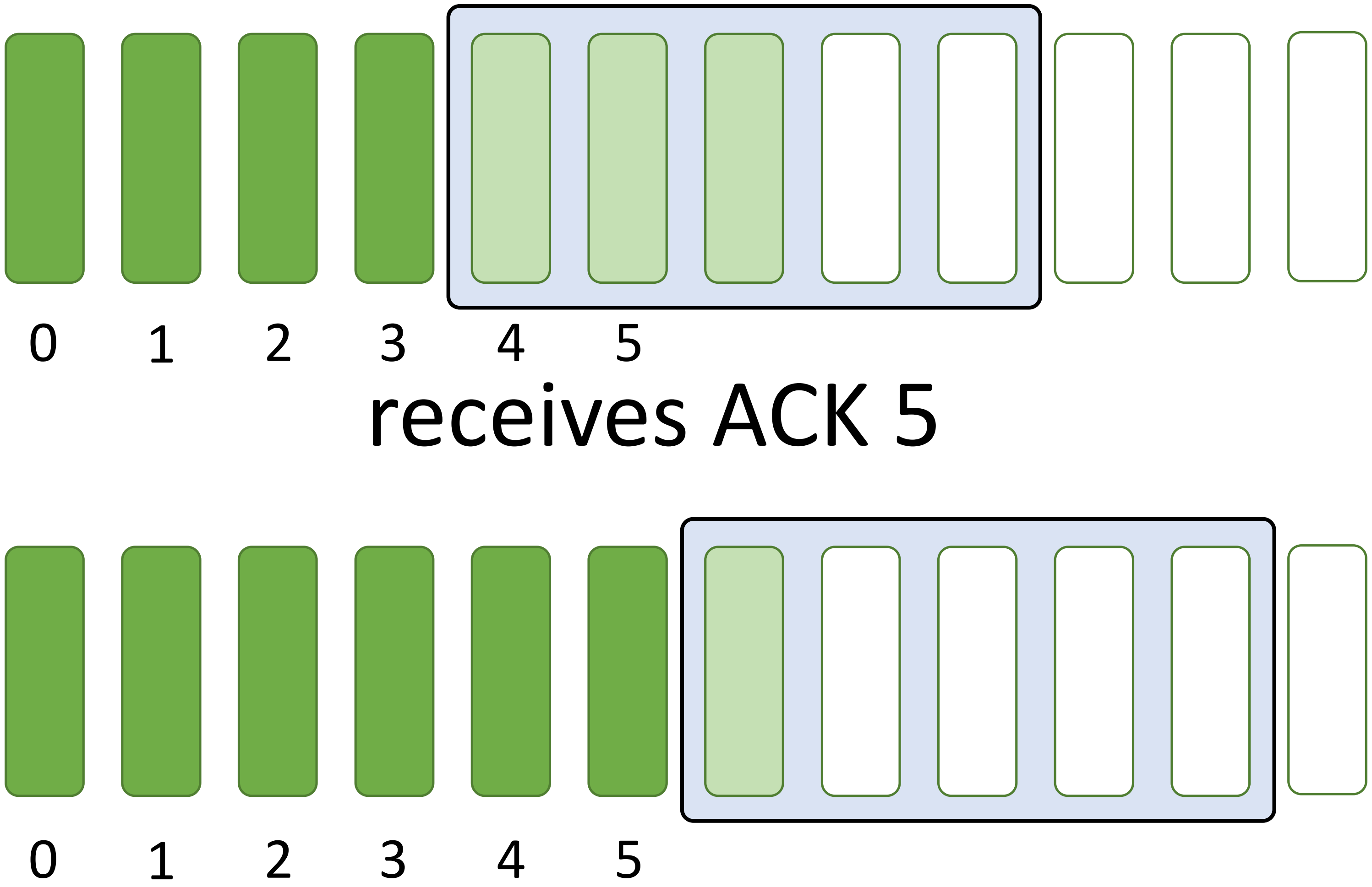
Go-Back-N sender



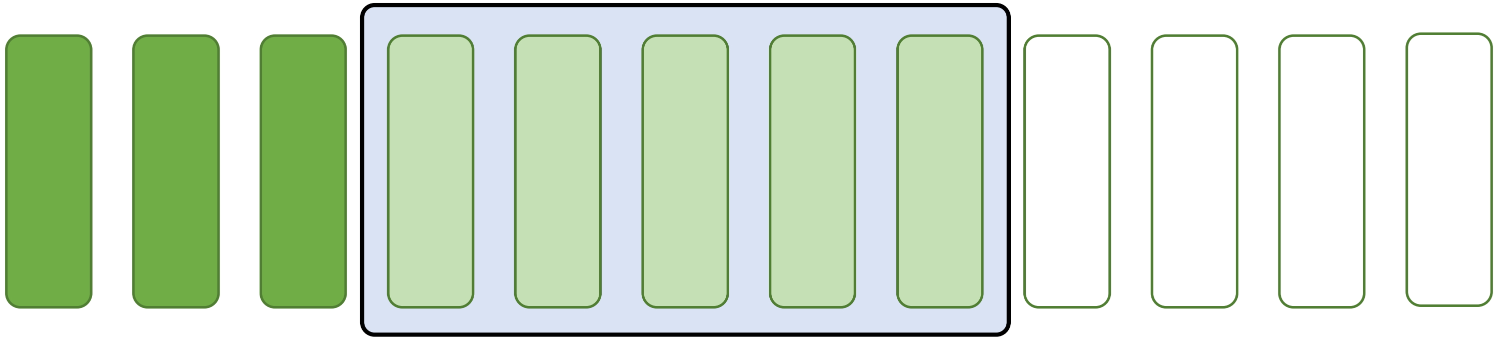
receives ACK 3



Go-Back-N sender

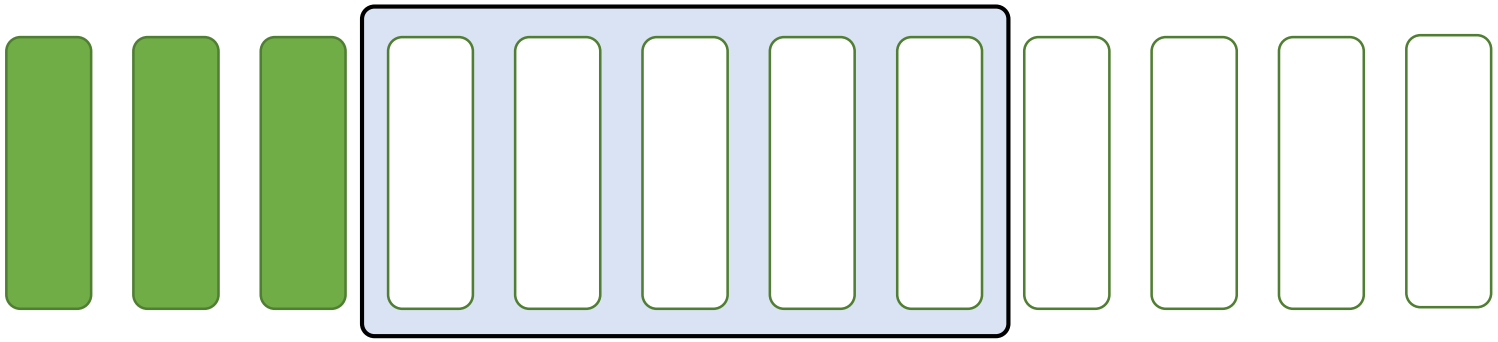


Go-Back-N sender



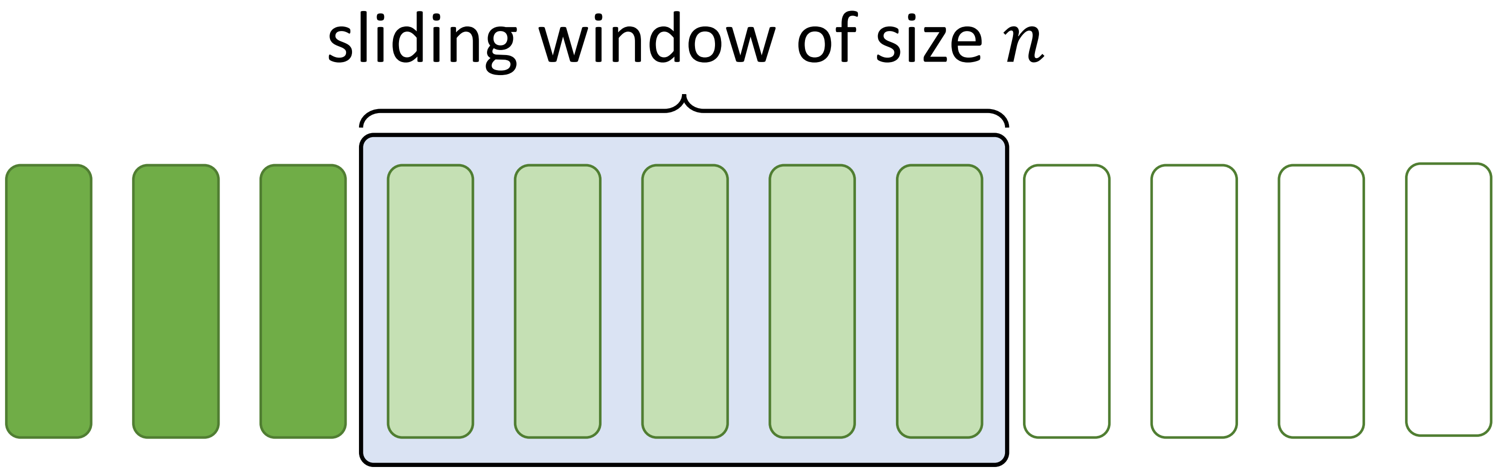
window is full

Go-Back-N sender



window is empty

Go-Back-N sender

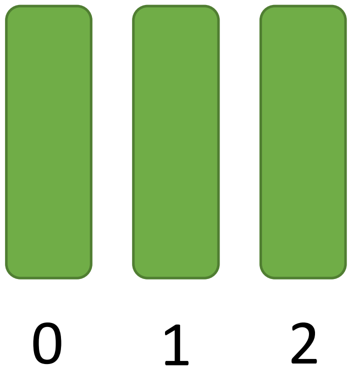


Keep track of n unACKed packets

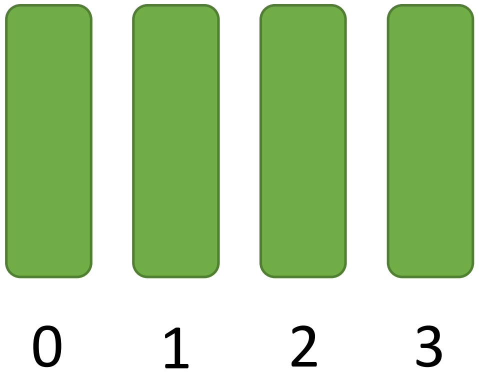
Timer for oldest unACKed packet

On timeout, retransmit all packets

Go-Back-N receiver

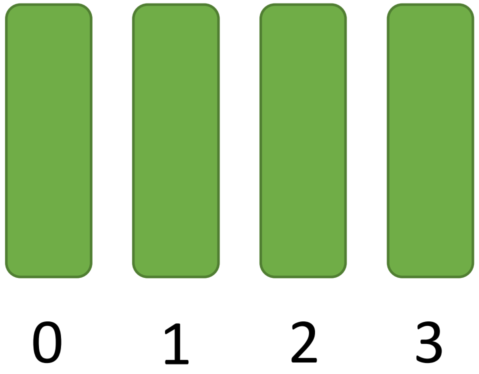


receives pkt 3

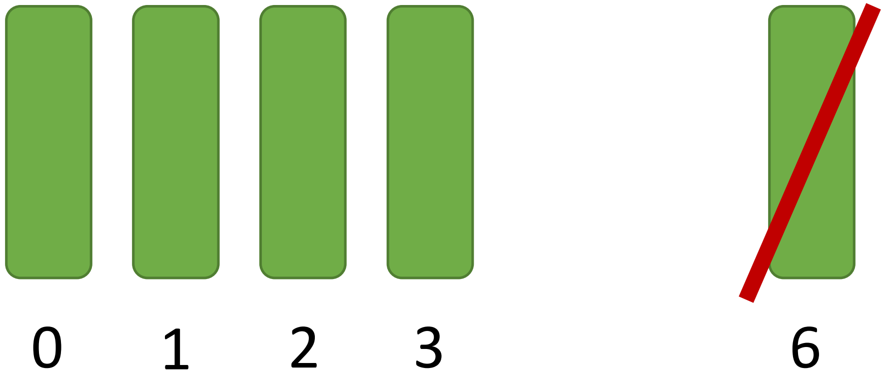


send ACK 3

Go-Back-N receiver



receives pkt 6



discards, (re)sends ACK 3

Go-back-N: Key Features

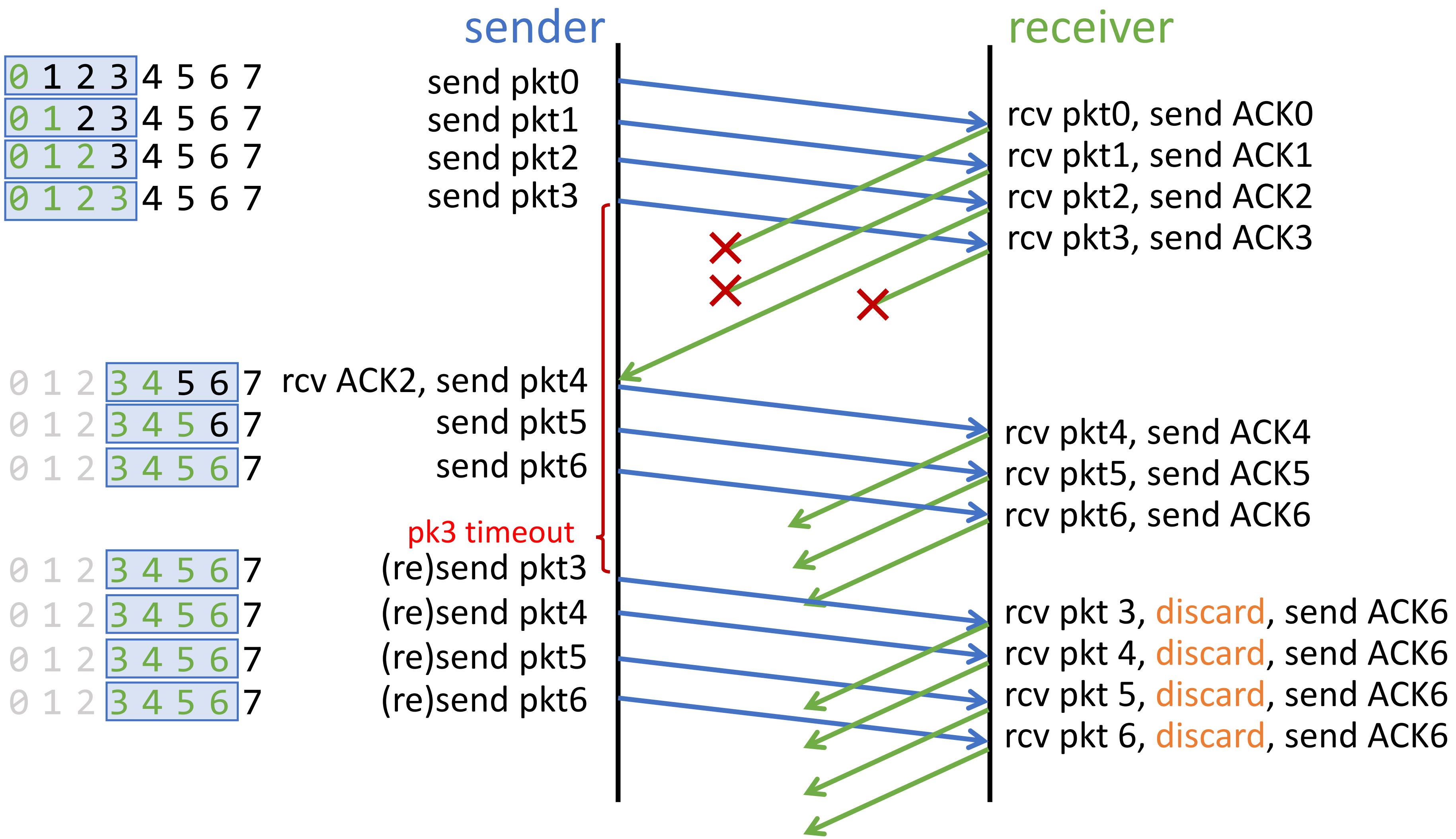
GBN Sender

- can have up to N unACKed packets in pipeline.
- insert k-bits sequence number in packet header.
- use a “sliding window” to keep track of unACKed packets.
- keep a timer for the oldest unACKed packet.
- timeout(n): retransmit packet n and all subsequent packets in the window.

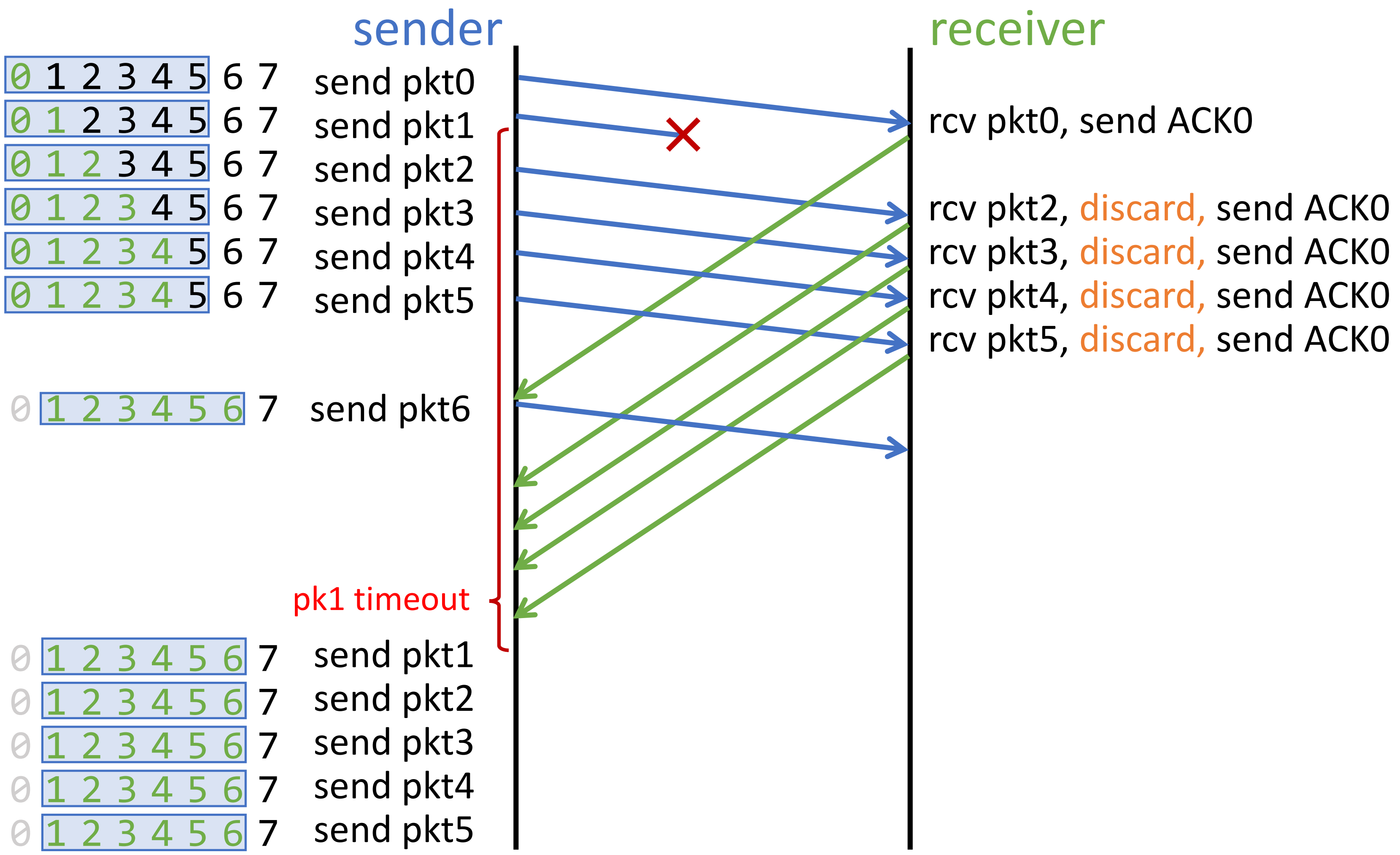
GBN Receiver

- only ACK packets that arrive in order.
    - simple receiver: need only remember expectedSeqNum
- discard out-of-order packets and ACK the last in-order seq. #.
    - Cumulative ACK: “ACK m” means all packets up to m are received.

Go-back-N in action



Go-back-N in action

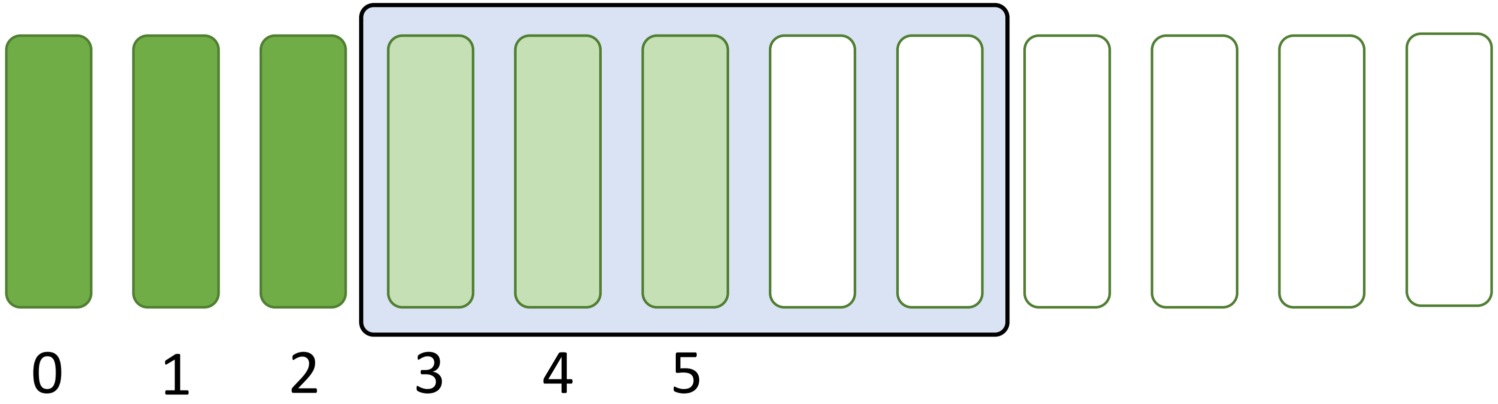


Selective Repeat

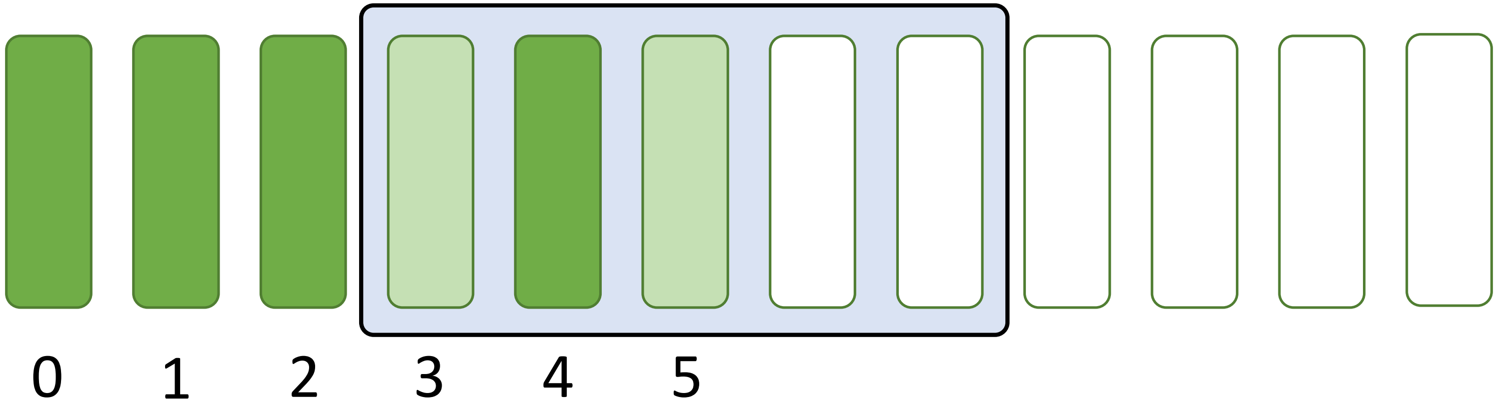
one timer per packet

receiver needs a buffer

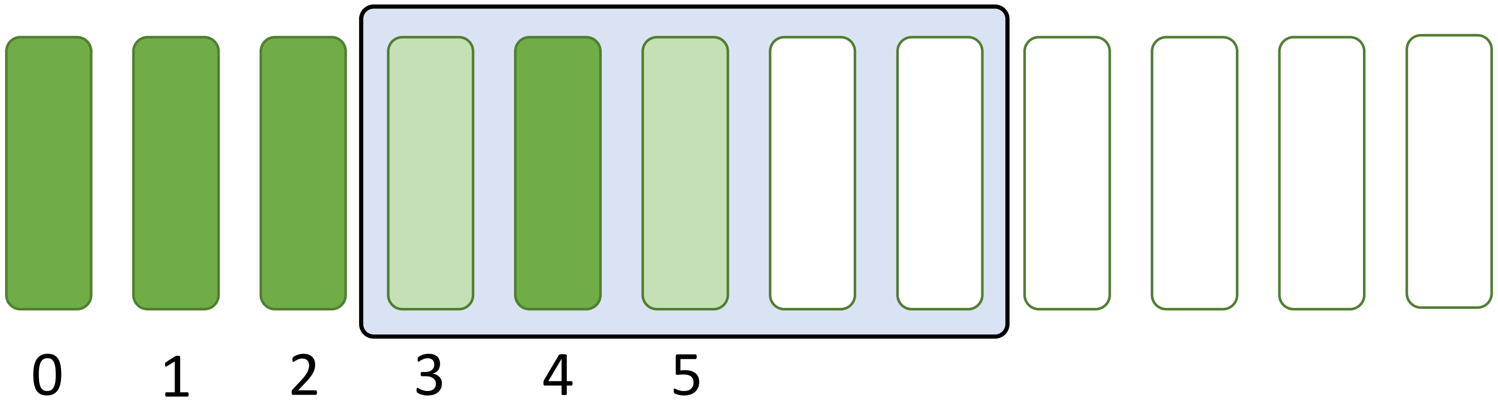
Selective Repeat sender



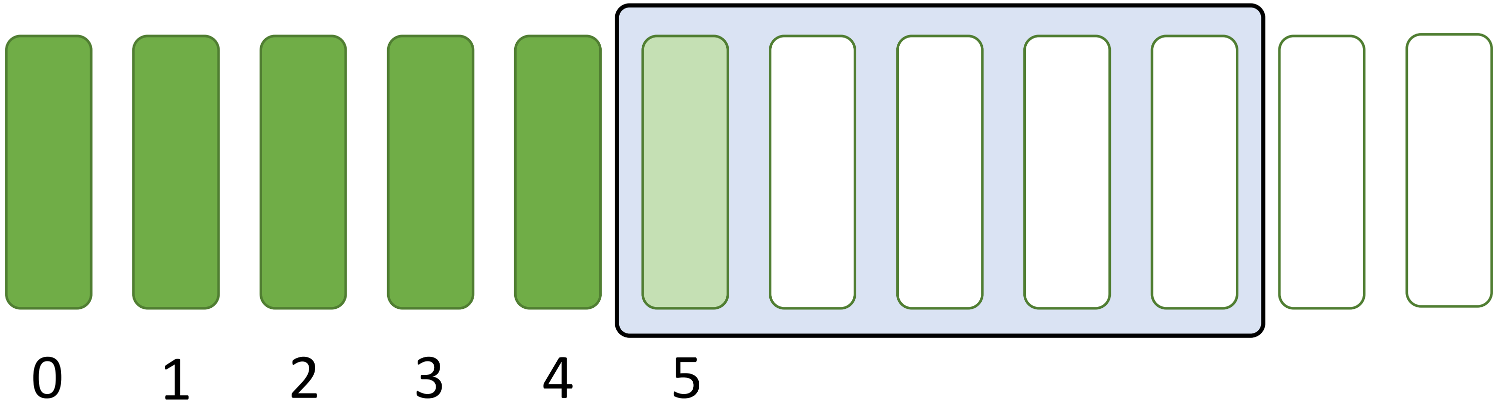
receives ACK 4



Selective Repeat sender



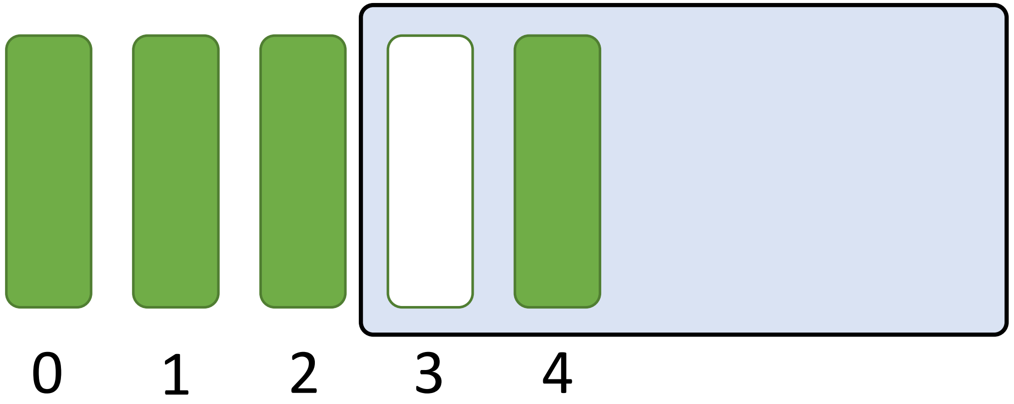
receives ACK 3



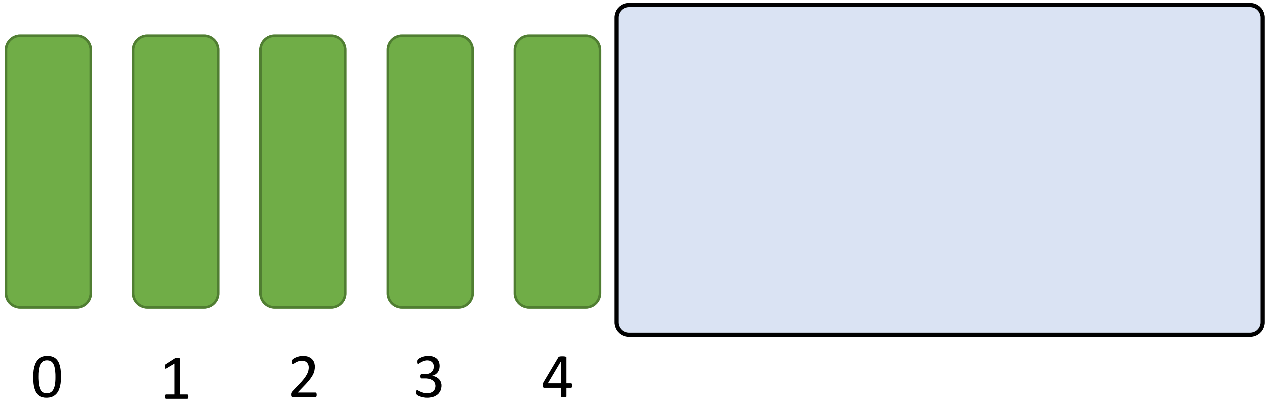
Selective Repeat receiver

Buffers pkt 4, send ACK 4

Selective Repeat receiver



receives pkt 3



send ACK 3, deliver pkts 3 & 4

Selective Repeat: Key Features

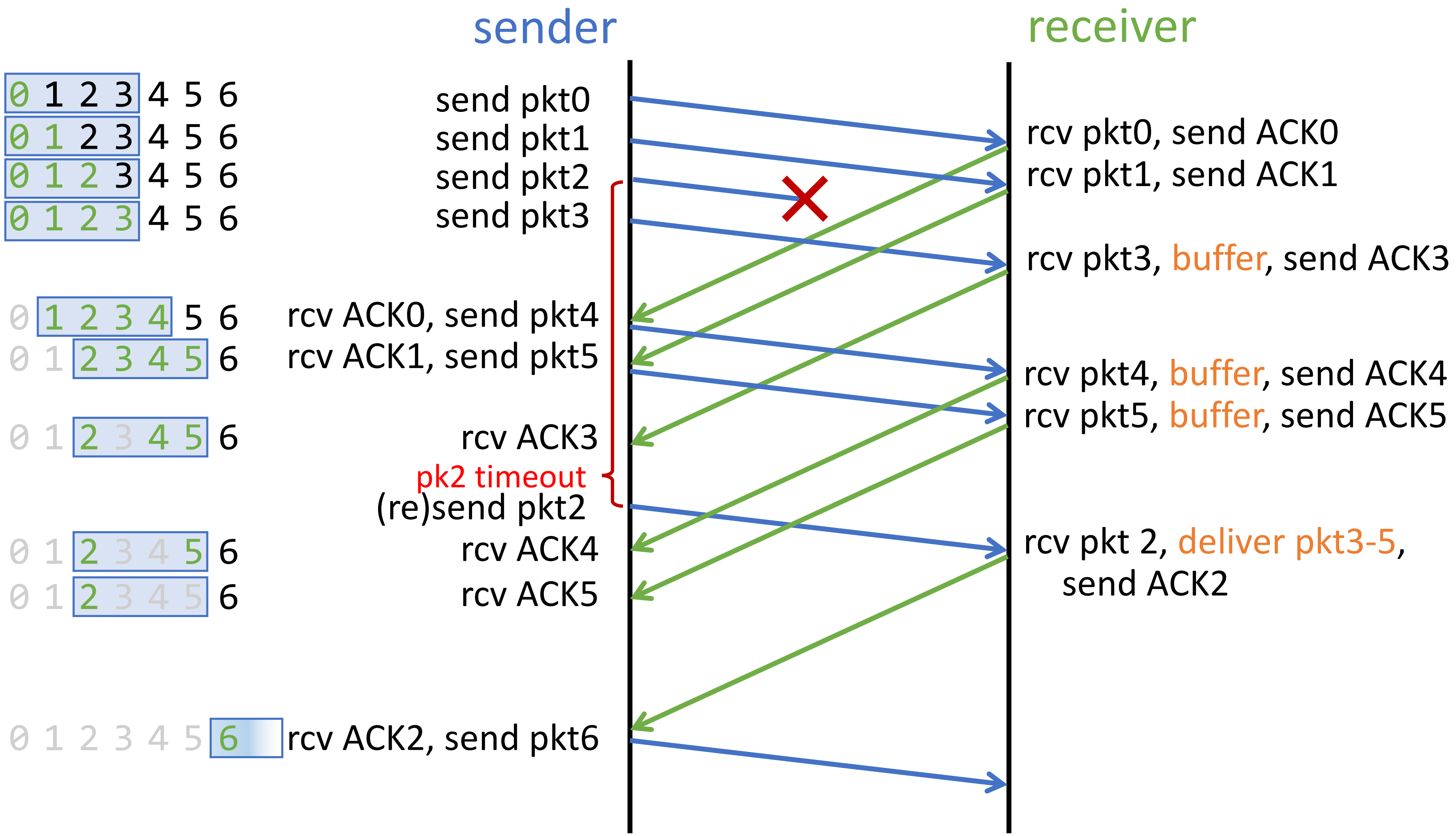
Receiver individually acknowledges all correctly received packets.

- Buffers out-of-order packets, as needed, for eventual in-order delivery to upper layer.

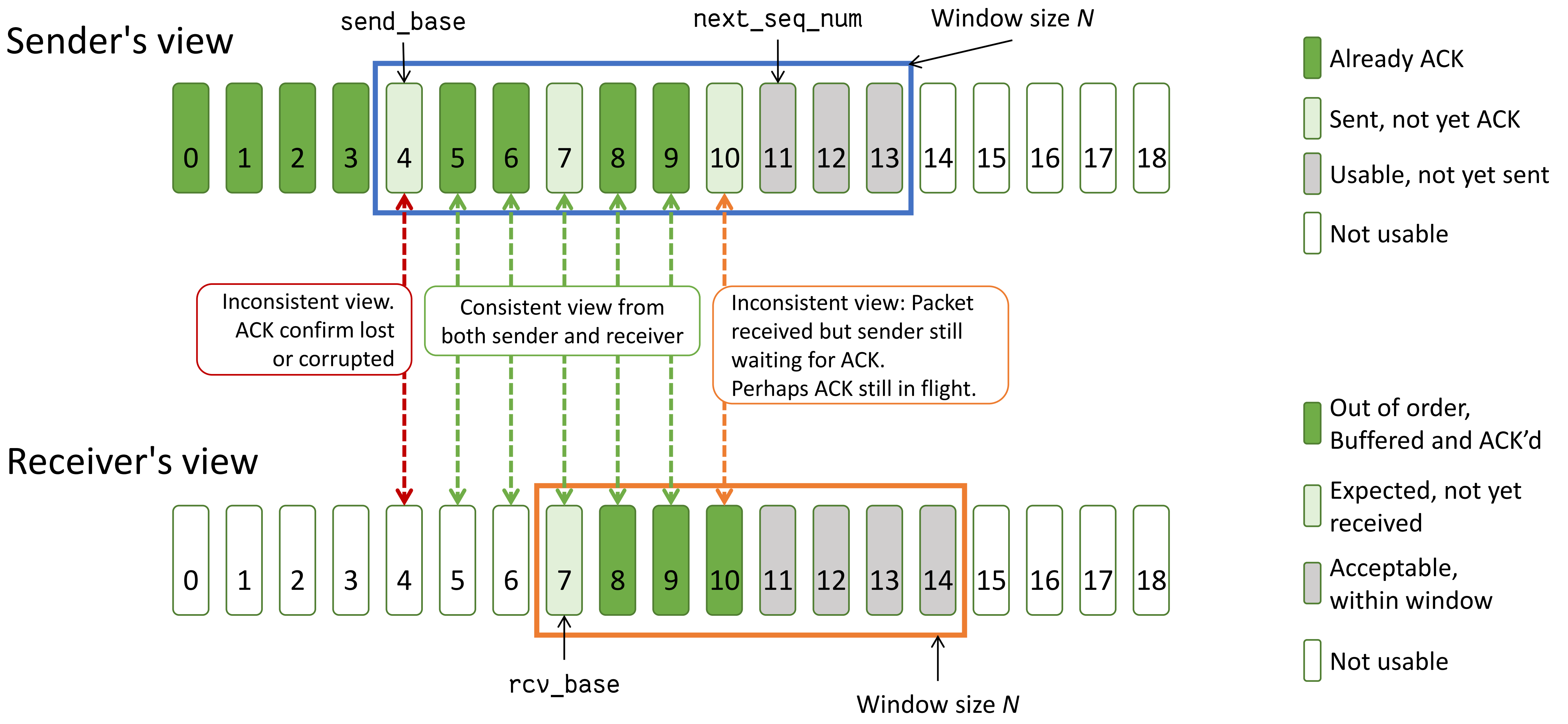
Sender maintains timer for each unACKed packet.

- When timer expires, retransmit only that unACKed packet.

Selective Repeat in action



Sender and Receiver Windows



Selective Repeat: Behaviors

Sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N]

- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq. #

Receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ACK(n)

otherwise:

- ignore

Summary

	Go-Back-N	Selective Repeat
#unACK packets	N packets in pipeline	N packets in pipeline
ACK style	cumulative	selective
out-of-order	discarded	buffered
timer	oldest unACK	each unACK
retransmit	all unACK	one unACK

Summary

rdt	Scenario	Features
1.0	no error	nothing
2.0	data corruption	checksum, ACK/NAK
2.1	data + ACK/NAK corruption	checksum, ACK/NAK, seq num
2.2	Same as 2.1	NAK free
3.0	data + ACK/NAK corruption, packet loss	checksum, ACK/NAK, seq num, timeout/re-transmit

What is scheme TCP?

Next lecture :)