15-441/641: Computer Networks
The Transport Layer, Part 1 of 3

15-441 Fall 2019

Profs Peter Steenkiste & **Justine Sherry**

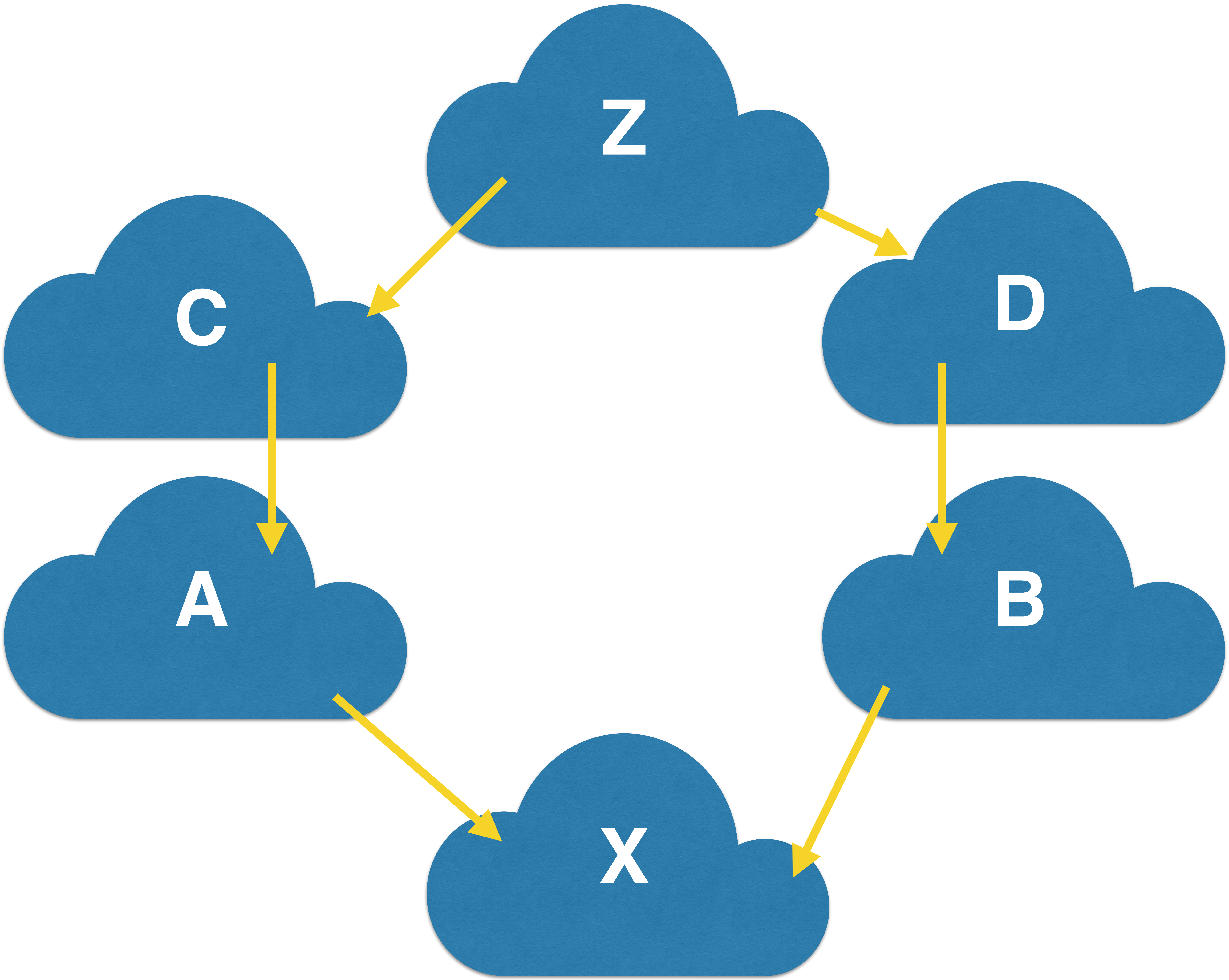
Carnegie Mellon University

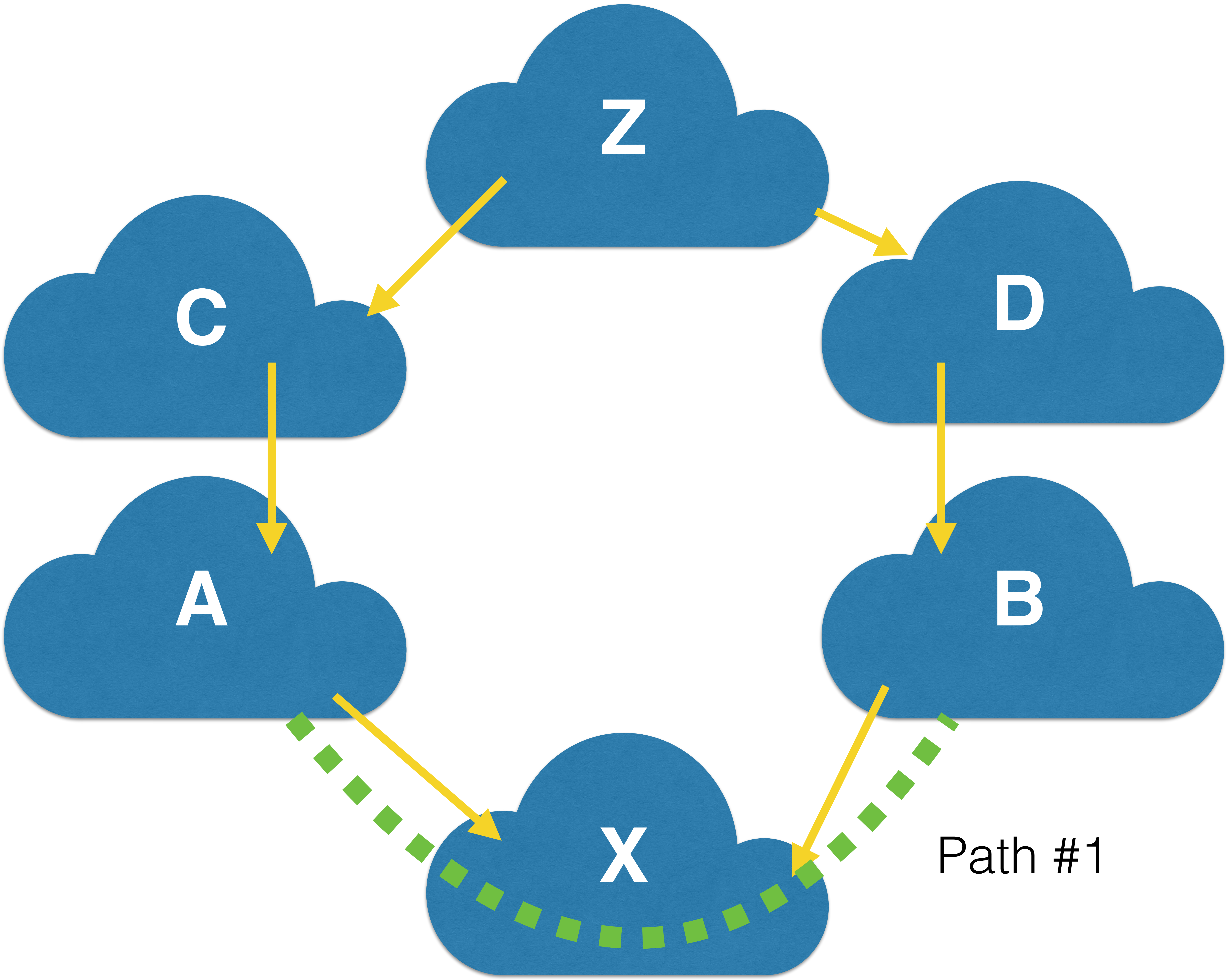
Warmup: BGP Refresh

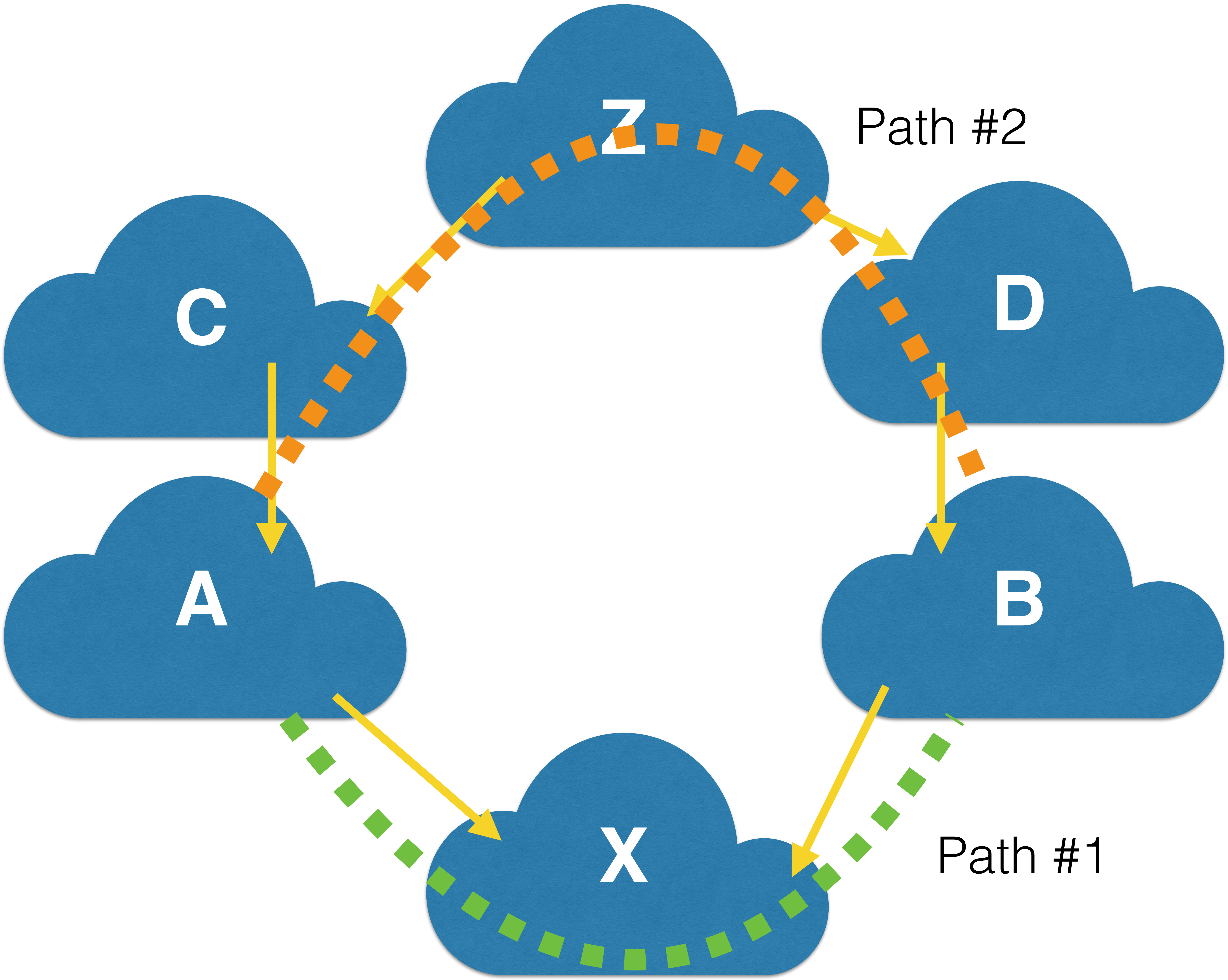
- X is a small university network with two providers, A and B.
- A’s provider is C.
- B’s provider is D.
- C’s provider is Z.
- D’s provider is Z.

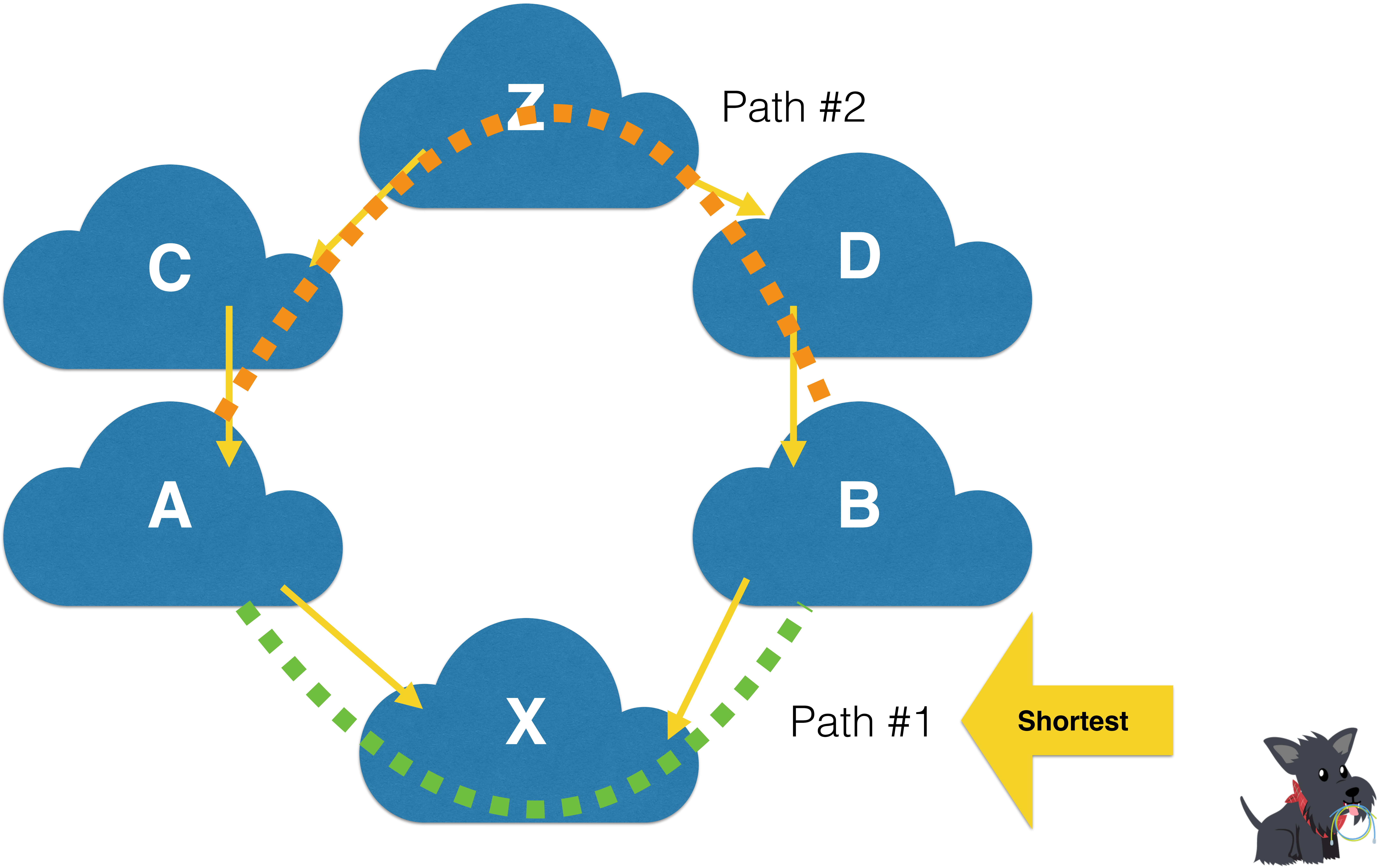
What AS path does traffic take from A to B?

- Why?


Z
C
D
A
B
X


Z
C
D
A
B
X
Path #1


Z
Path #2
C
D
A
B
X
Path #1


Z
Path #2
C
D
A
B
X
Path #1
Shortest

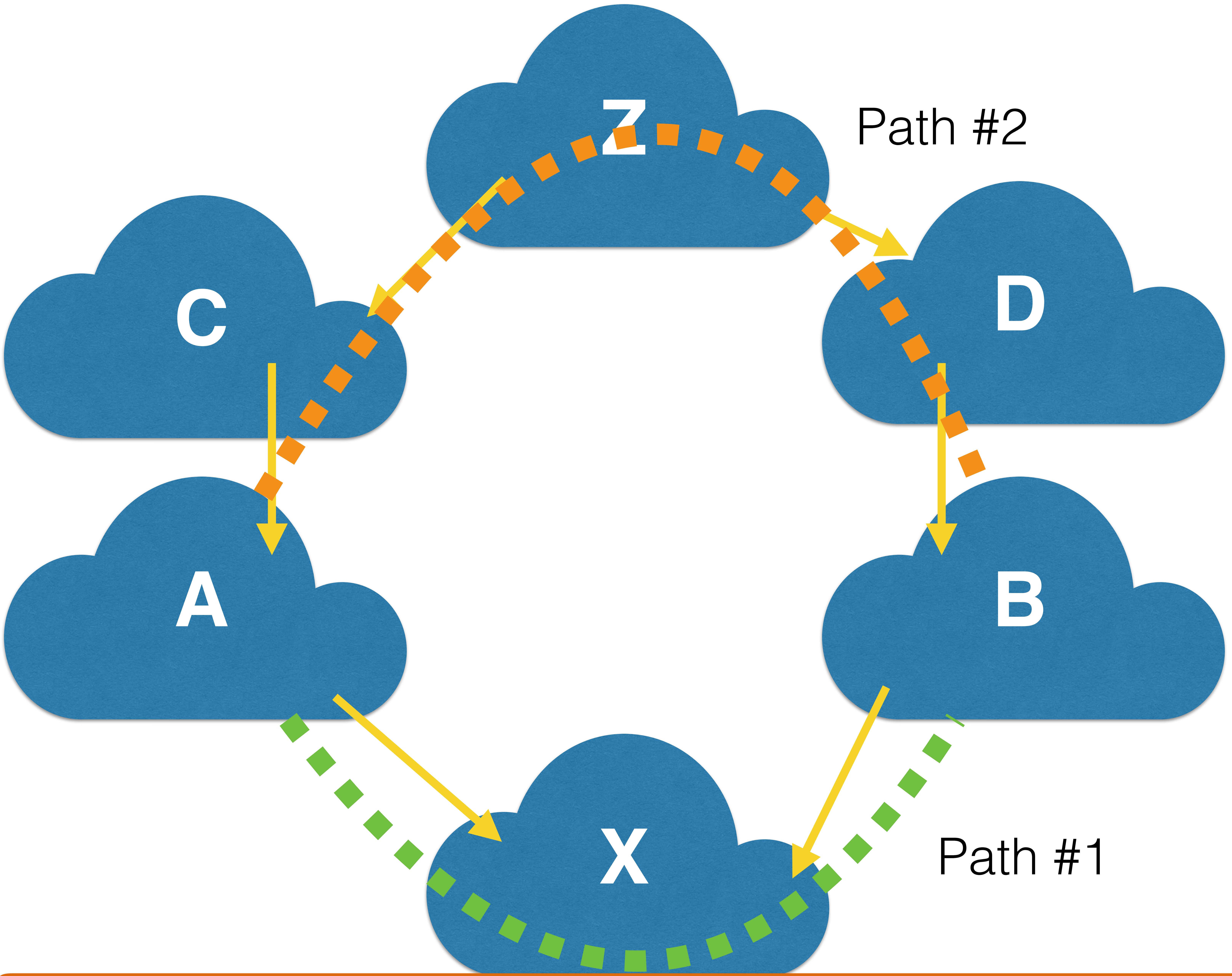

That's illegal.

Gao-Rexford Conditions

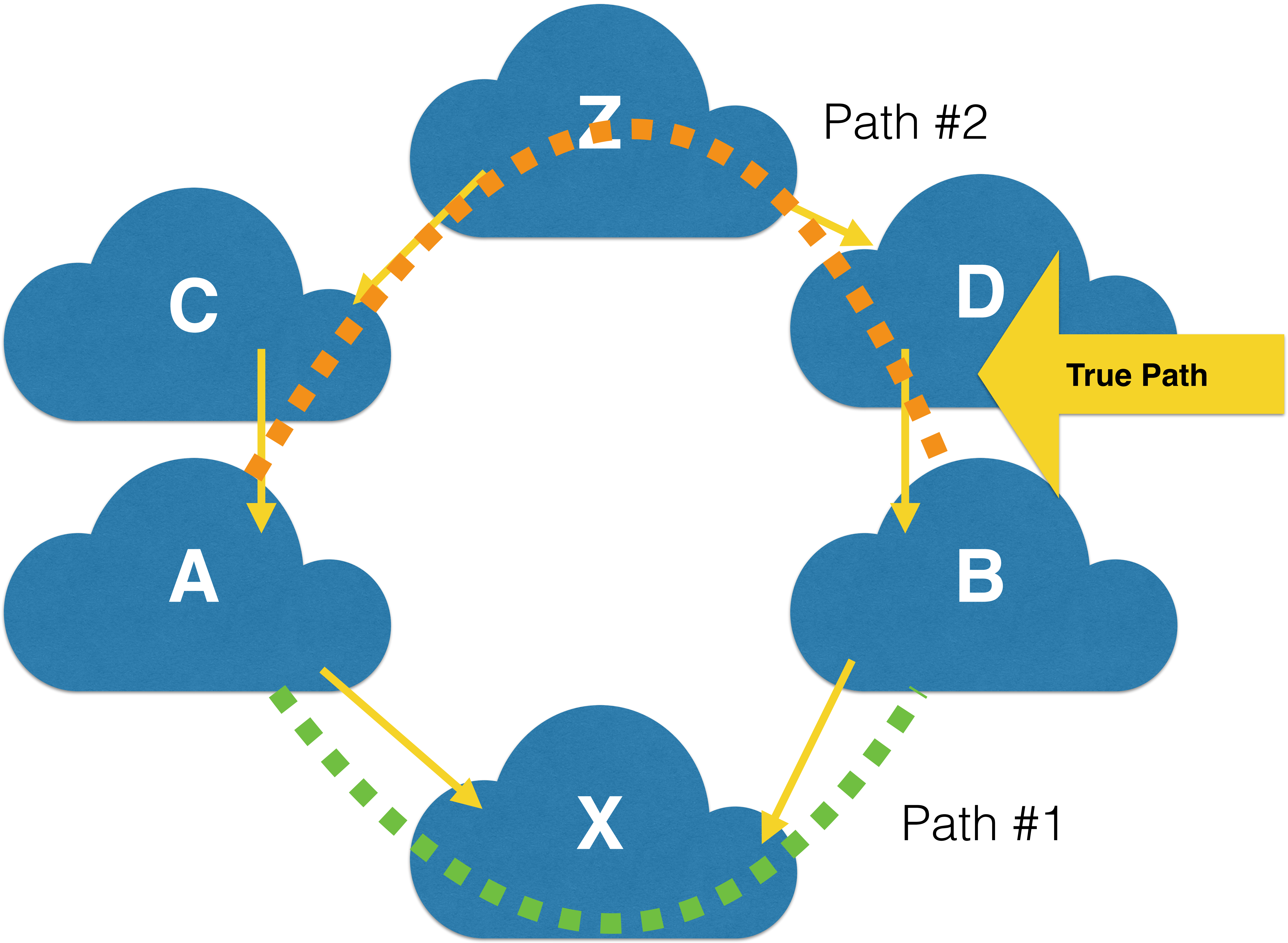
- If I receive a route announcement from my *customer*, I will announce that route to my customers, peers, and my providers.
- If I receive a route announcement from my *peer*, I will announce that route only to my customers.
- If I receive a route announcement from my *provider*, I will announce that route only to my customers.

I only want to carry traffic that will earn me a profit!

Gao-Rexford: “Scrooge McDuck Policy”



X would never announce a route for B to A

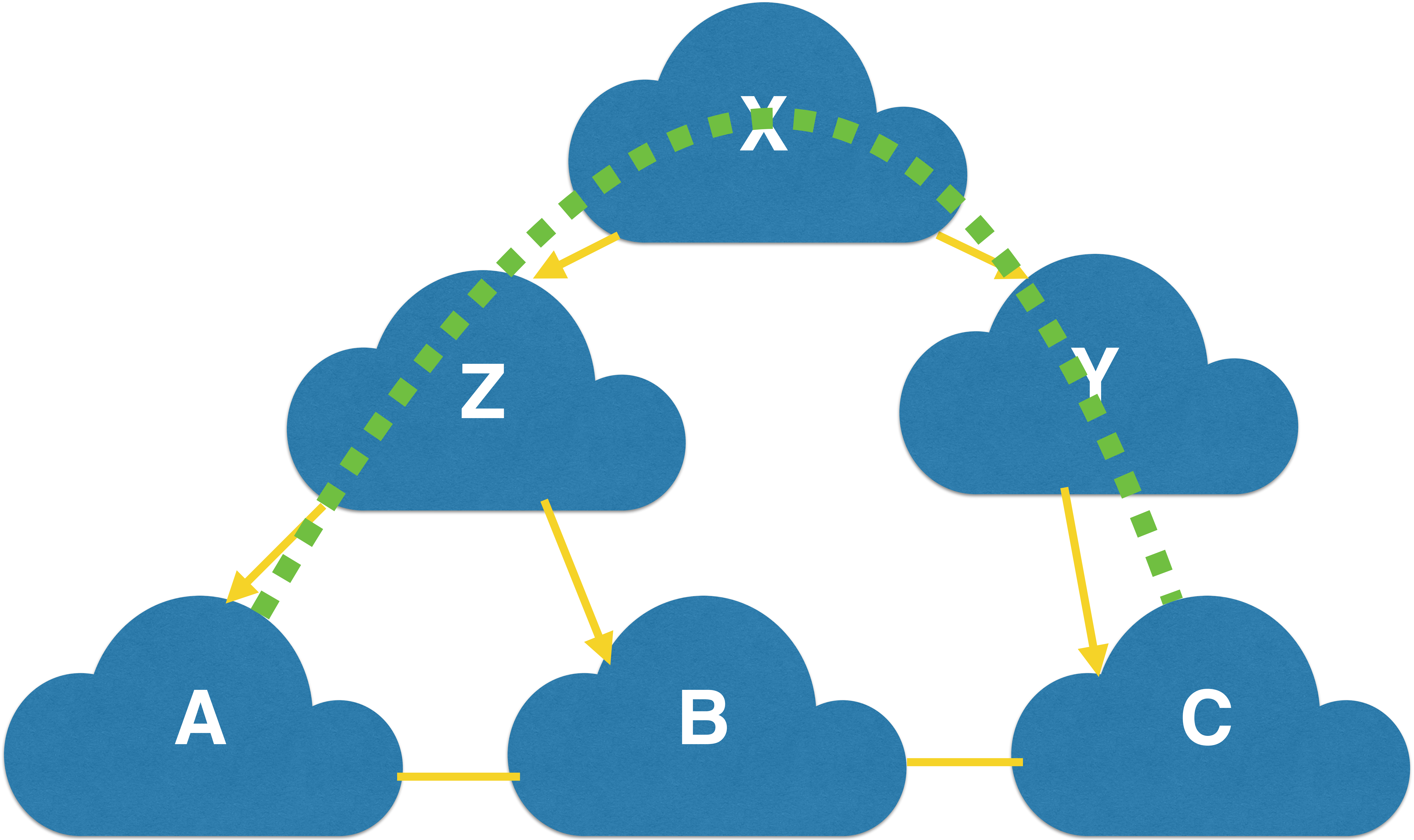

Z
Path #2
C
D
True Path
A
B
X
Path #1

Another BGP Warmup

- A's provider is Z. A peers with B.
- B's provider is Z. B peers with A and C.
- C's provider is Y. C peers with B.
- Z's provider is X.
- Y's provider is X.

What AS path does traffic take from A to C?

- Why?



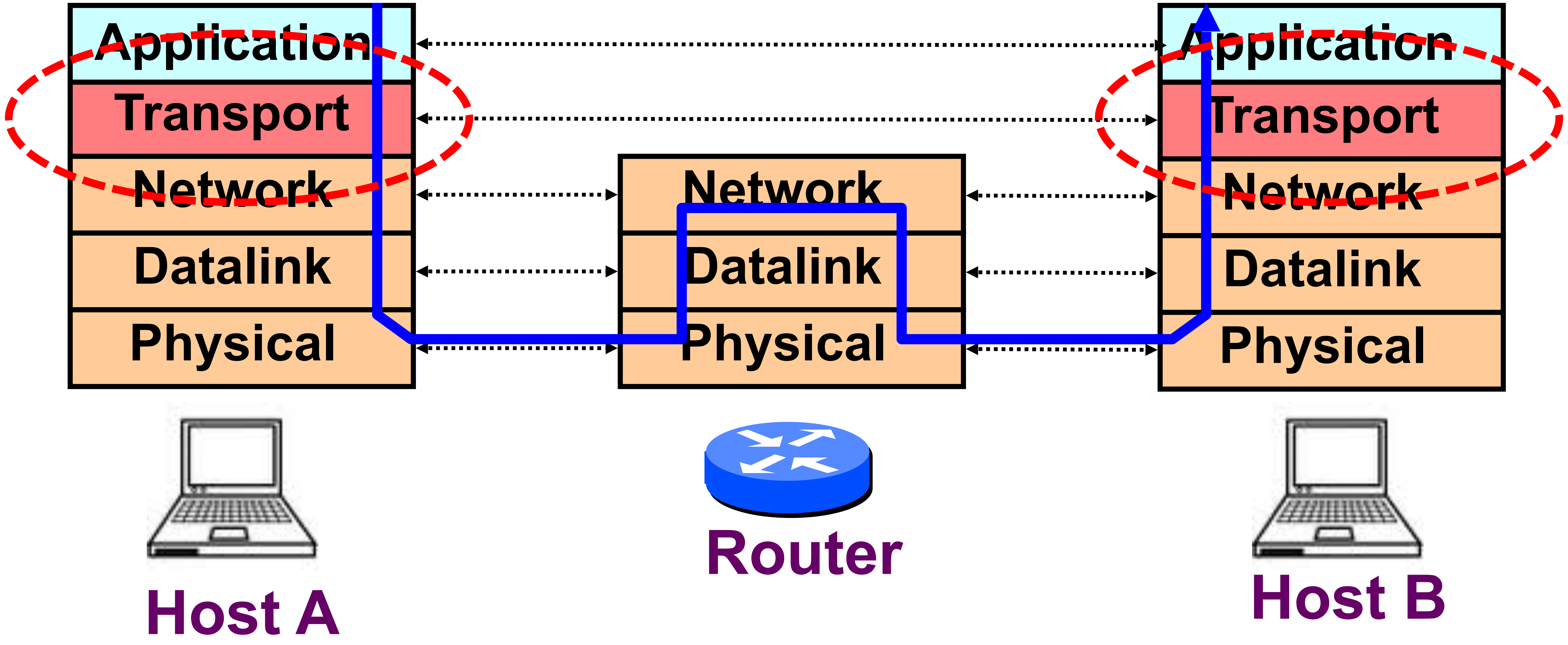
Follow the money!

Today

- Starting three lectures on the transport layer.
- The transport layer is currently one of my primary areas of research.
- I'll teach you the basics….
- For lecture #3, our TA Ranysha is going to tell you about her PhD research on modern transport on the Internet.
    - Including new protocols from companies like Google and Akamai

Quick Review

Transport Layer in the Internet Model

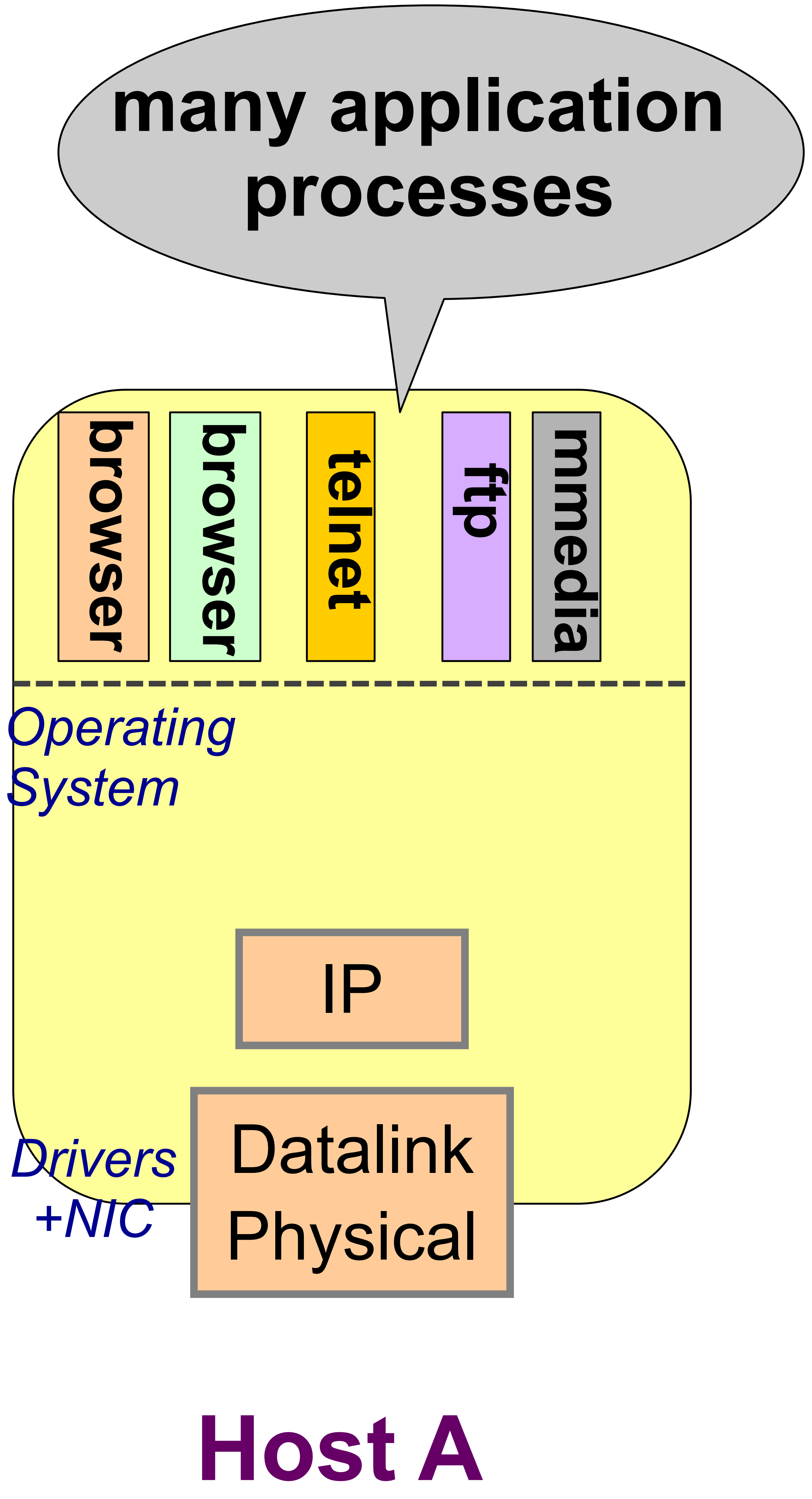


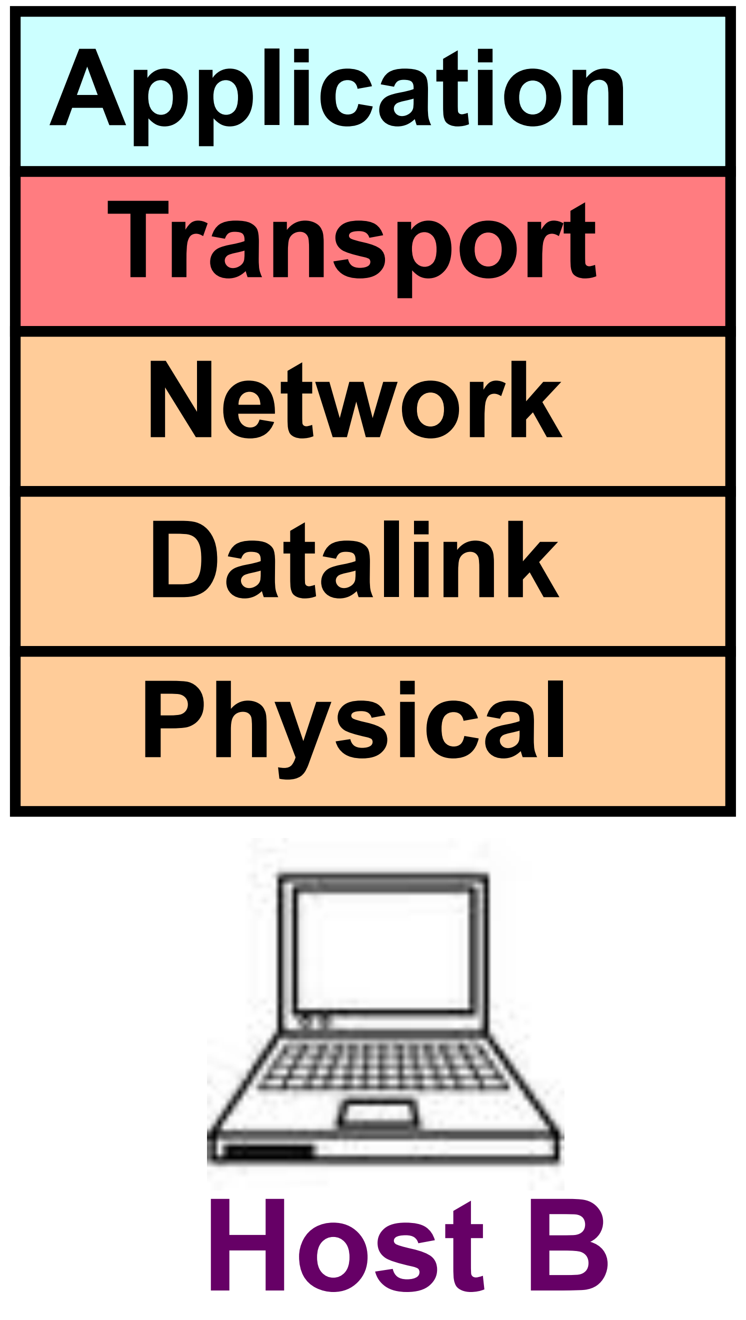
Why a transport layer?

- 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 (*multiplexing/demultiplexing*)

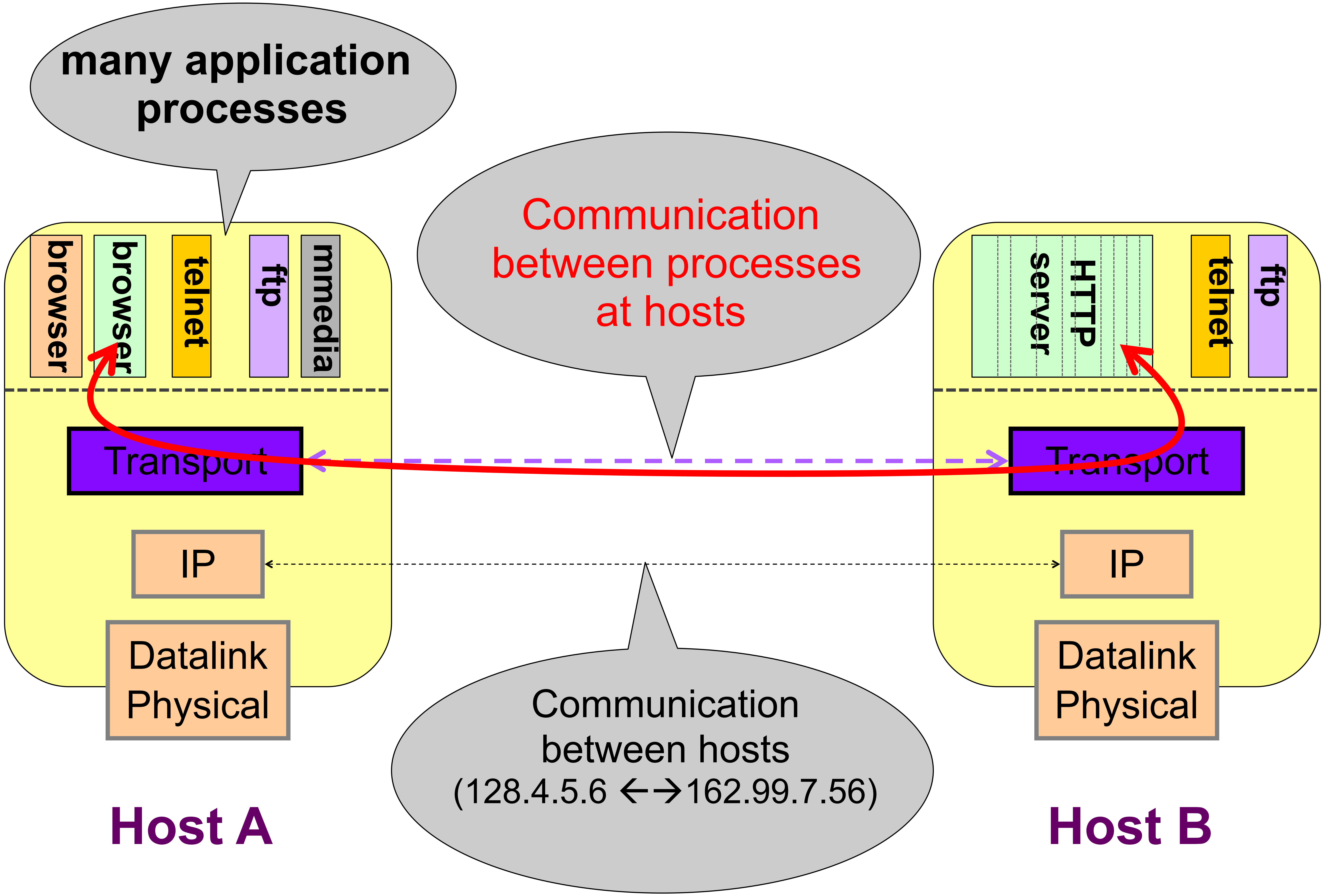
Why a transport layer?

Why a transport layer?





Why a transport layer?



Role of the Transport Layer

- Communication between application processes
  - Mux and demux from/to application processes
  - Implemented using *ports*
- *You know this from Liso project!*

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

Role of the Transport Layer

- Communication between application 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, MTCP, SST, RDP, DCCP, …

Context: Applications and Sockets

- Socket: software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system
  - socketID = socket(…, socket.TYPE)
  - socketID.sendto(message, …)
  - socketID.recvfrom(…)

- Two important types of sockets
  - UDP socket: TYPE is SOCK_DGRAM
  - TCP socket: TYPE is SOCK_STREAM

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, no-frills transport protocol
    - only provides mux/demux capabilities

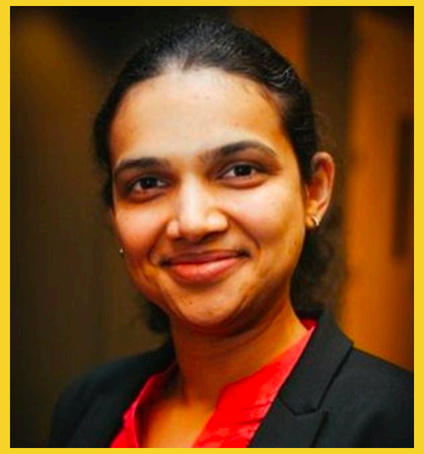
UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
- UDP described in RFC 768 – (1980!)
  - Destination IP address and port to support demultiplexing
  - Optional error checking on the packet contents
    - (**checksum** field = 0 means “don’t verify checksum”)

SRC port	DST port
checksum	length
DATA	

What is a checksum?

- Wikipedia: “A checksum is a small-sized c digital data for the purpose of detecting errors th during its transmission or storage.”



Take a class from Prof. Vinayak to learn more about information theory and how to use it to build systems!

- Simplest checksum:
  - Take every, say, 32-bit word and XOR them all ether
  - Append the result to the end of the packet (add overhead!)
  - At the receiver, re-compute the XOR. If it does n t match the appended checksum, you know some of the data has beer corrupted.
- There is a huge literature on “coding” checksumming schemes.

UDP

- That's literally the entire protocol.
- If a packet gets lost, it's up to the application developer to decide what to do about it.

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, no-frills transport protocol
- TCP is the whole-hog protocol
  - offers apps a reliable, in-order, bytestream abstraction
  - with congestion control
  - but no performance guarantees (delay, bw, etc.)

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

TCP, literally the next three lectures

Getting this right is *hard* and hence it is an active area of research.

Let’s get started understanding why this is challenging…

I need two volunteers.

Team Structure

- I have ten beanbags labeled 1 to 10.
- Your job is to transport them from one end of the classroom to the other.
- Like Professor Sherry, you must throw them — you can't simply carry them to the other side of the classroom.
  - Unlike Professor Sherry, you may have better aim.
  - Or they might fall to the ground. If they fall, you can't pick them back up!
- If you determine that a beanbag is lost, you can grab another beanbag, label it with the missing number, and re-transmit it.

Team Structure

- Two of you are the end points (sender/receiver) who decide what packets to transmit, and whether or not to re-transmit. The endpoints must face the wall — they can't see the network. But, they can talk!

- The other two represent the network in the middle. You can see everything, but you can't talk or signal in any way to the endpoints.

    - The sender will will hold up a bean bag in the air if they have a bag they want you deliver.

    - They receiver will hold up their hand so you can put beanbags into it.

    - But otherwise no talking! Just try not to let the beanbags fall!

PRIZES

- The winner is whoever successfully gets beanbags numbered 1…10 to their receiver first.
- Winning team gets t-shirts
- Losing team still gets candy!

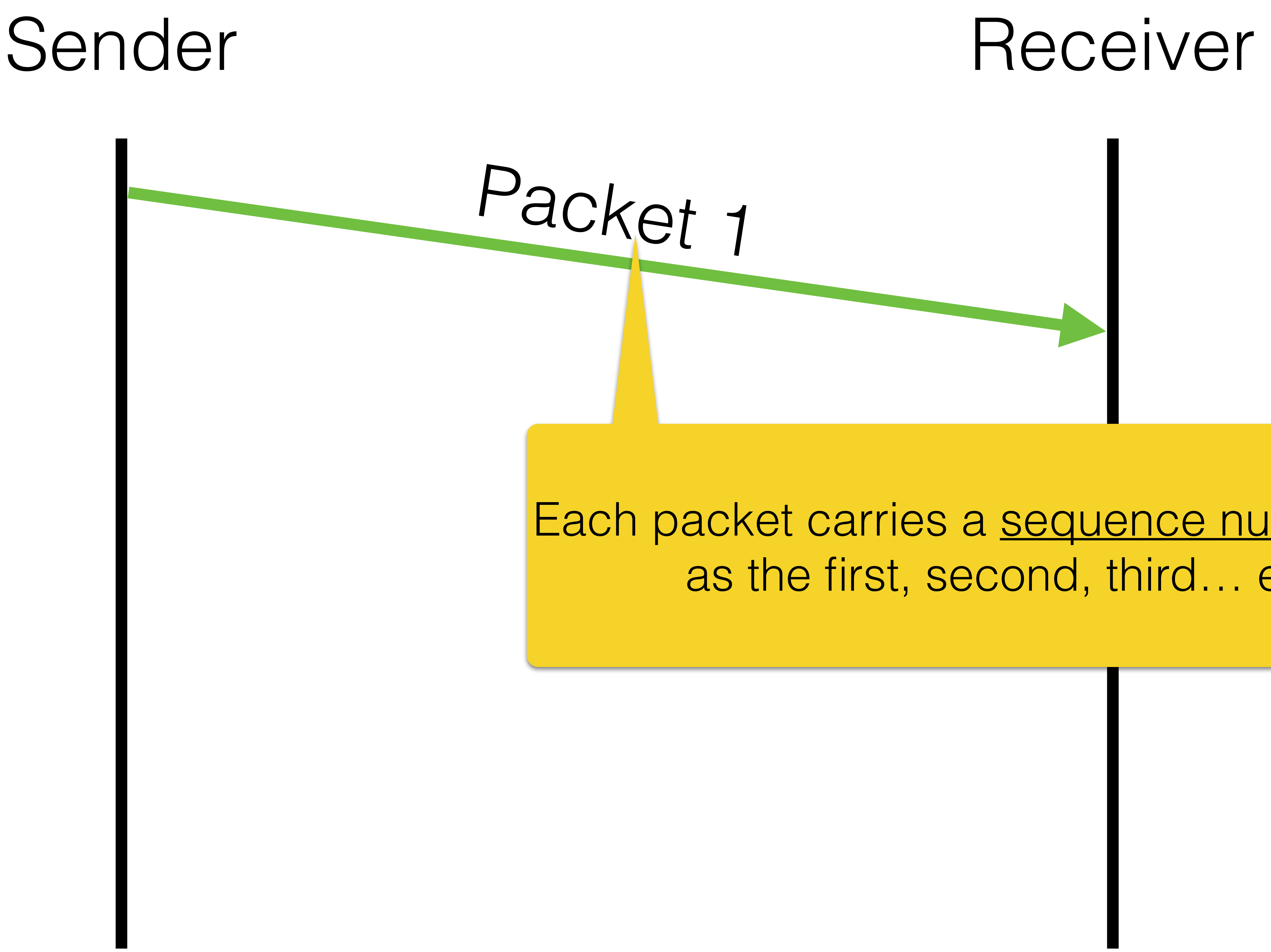
Back to the real Internet…

How do we tell that a packet has been lost?

- The packet was sent a long time ago, but still has not arrived
- Packet arrives at receiver, but data does not match its checksum

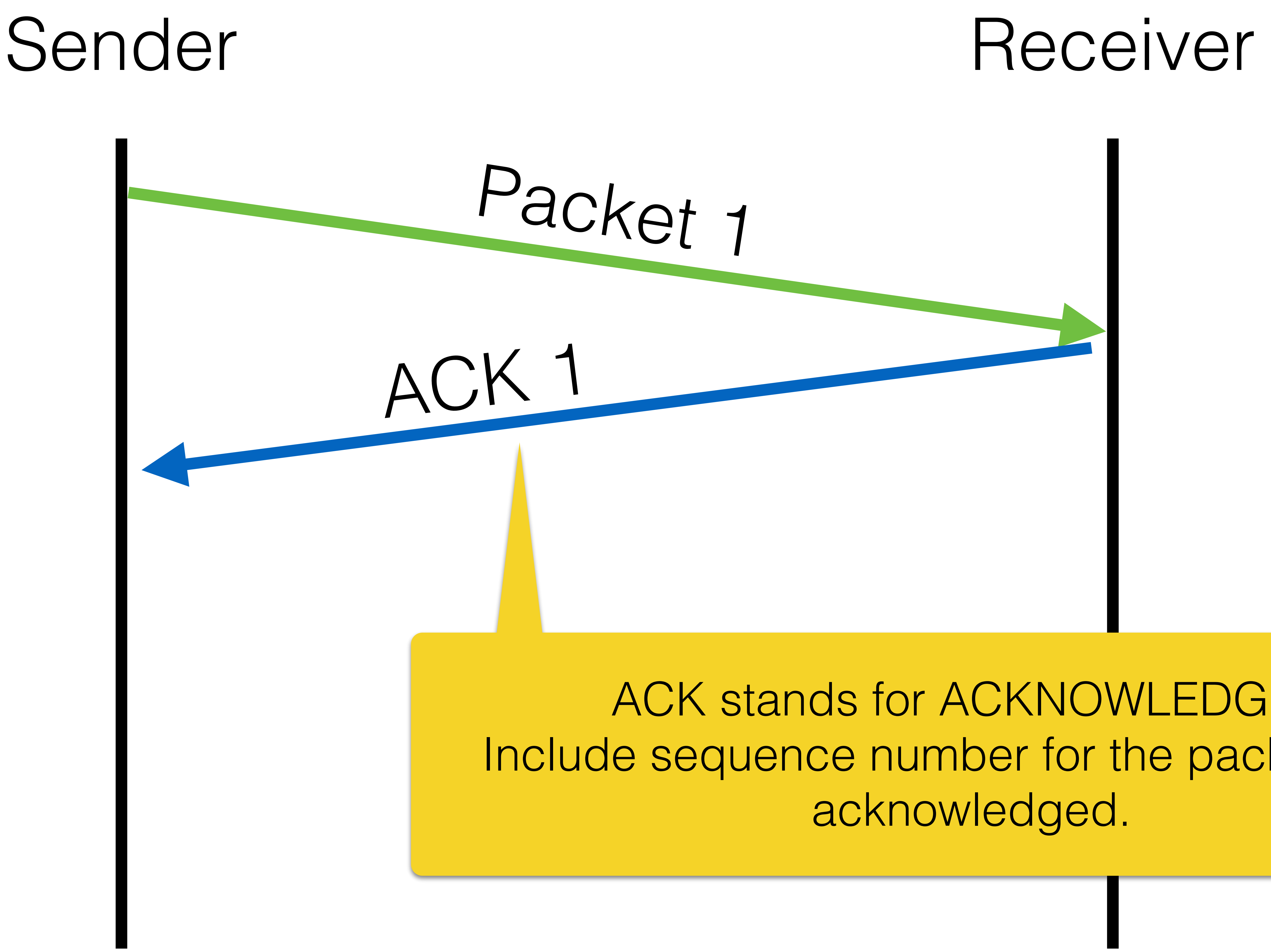
A basic protocol: Stop and Wait

A basic protocol: Stop and Wait



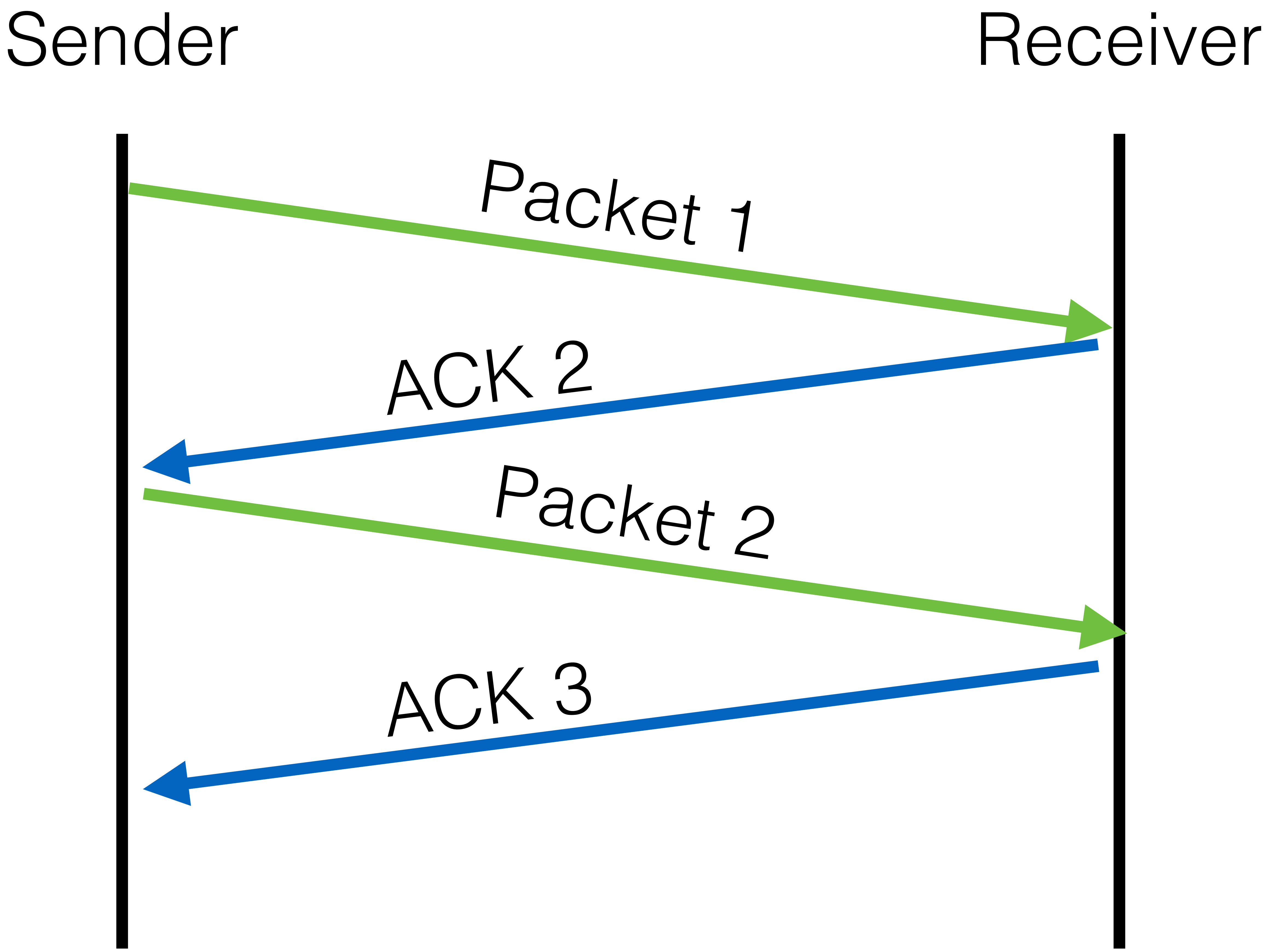
Each packet carries a sequence number identifying it as the first, second, third… etc packet.

A basic protocol: Stop and Wait



ACK stands for ACKNOWLEDGED
Include sequence number for the packet being acknowledged.

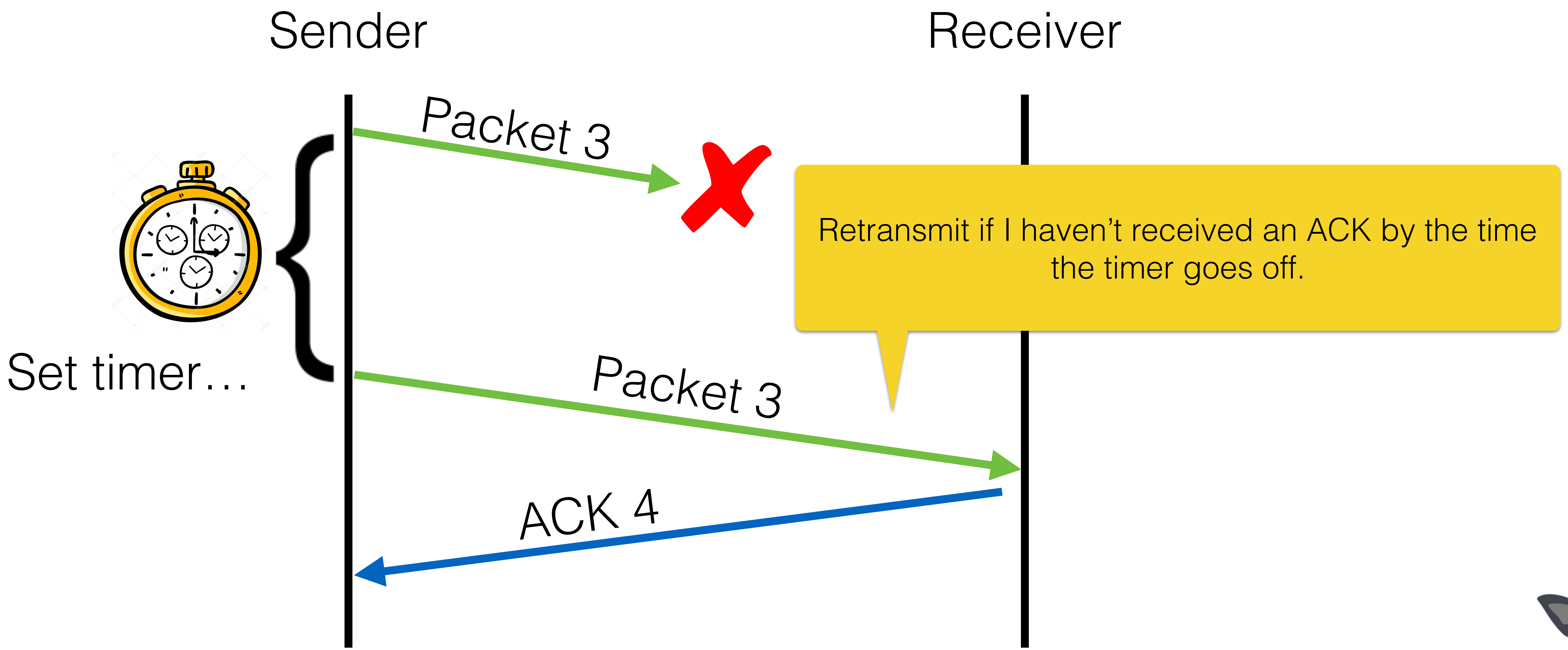
A basic protocol: Stop and Wait



How do we tell that a packet has been lost?

- The packet was sent a long time ago, but still has not arrived
- Packet arrives at receiver, but data does not match its checksum

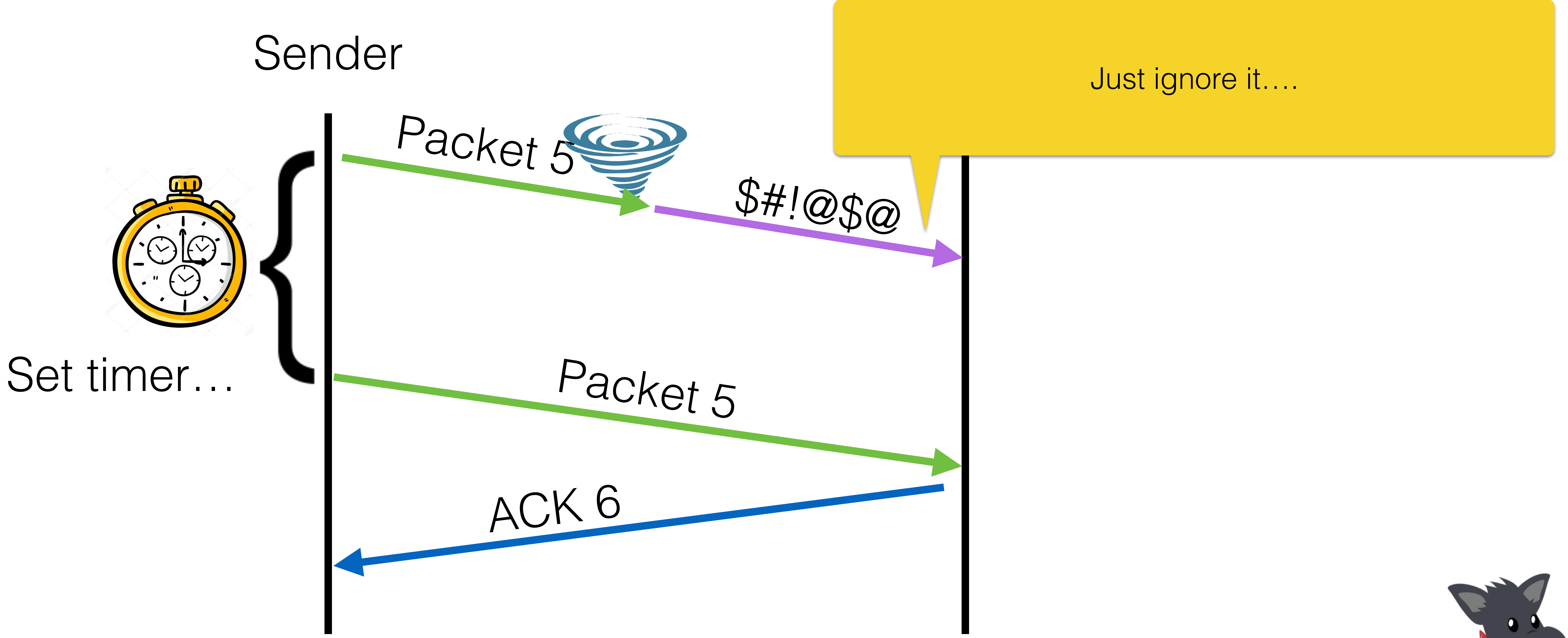
A basic protocol: Stop and Wait



How do we tell that a packet has been lost?

- The packet was sent a long time ago, but still has not arrived
- Packet arrives at receiver, but data does not match its checksum

A basic protocol: Stop and Wait



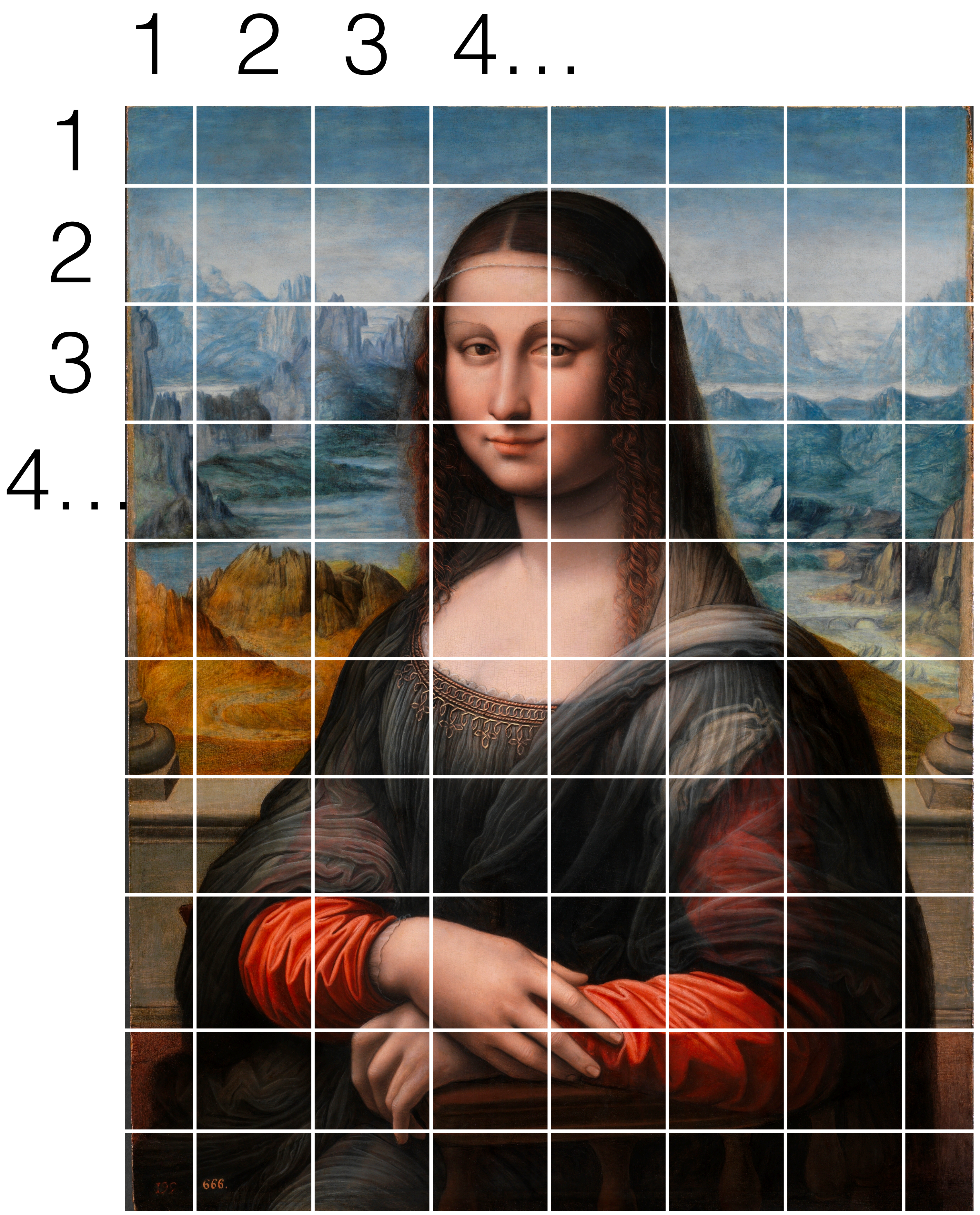
Stop-and-Wait: Summary

- **Sender:**
  - Transmit packets one by one. Label each with a sequence number. Set timer after transmitting.
  - If receive ACK, send the next packet.
  - If timer goes off, re-send the previous packet.
- **Receiver:**
  - When receive packet, send ACK.
  - If packet is corrupted, just ignore it — sender will eventually re-send.

Why do we need sequence numbers? Could we use Stop-and-Wait without them?

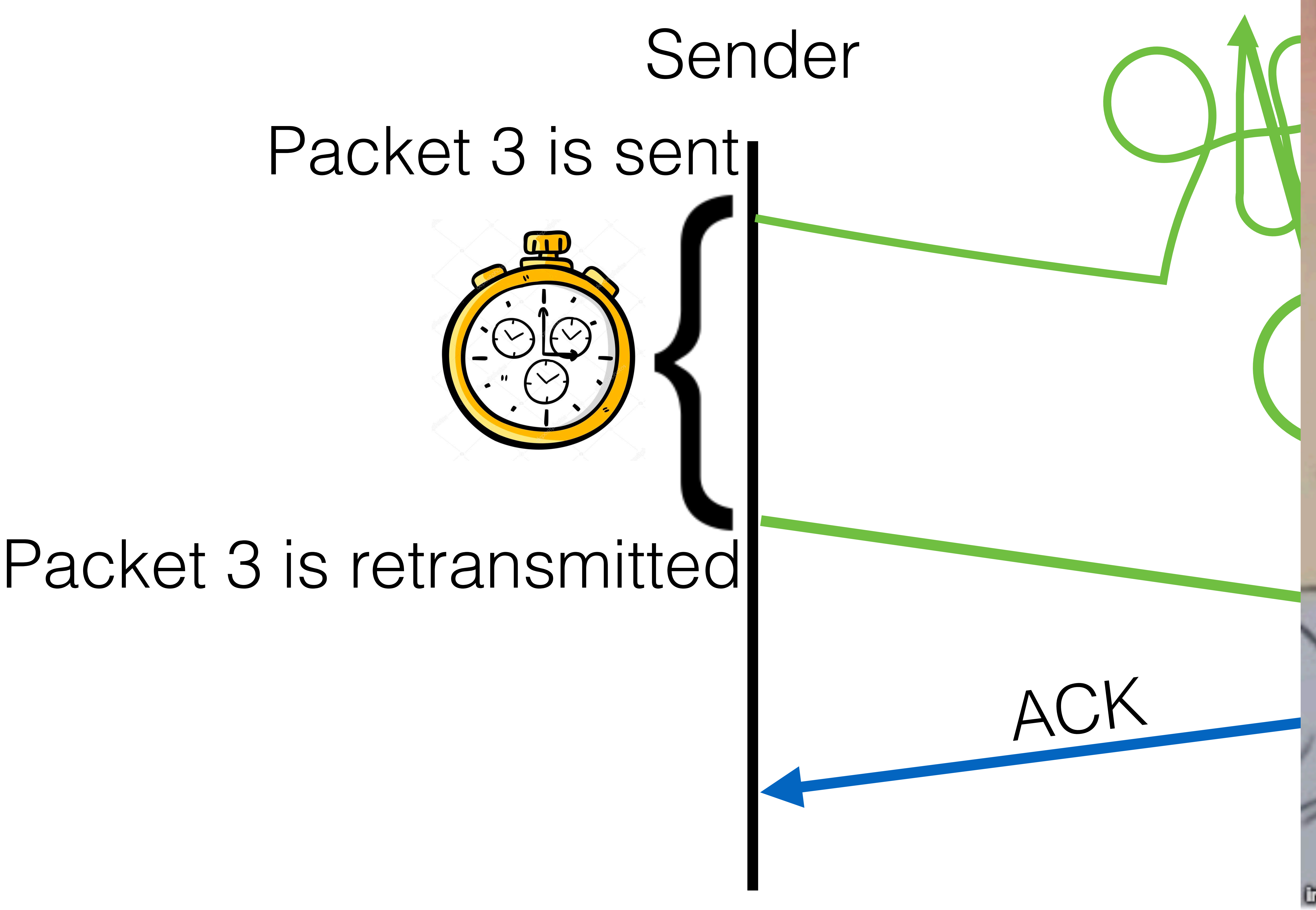
Intuitive Need for Sequence Numbers…

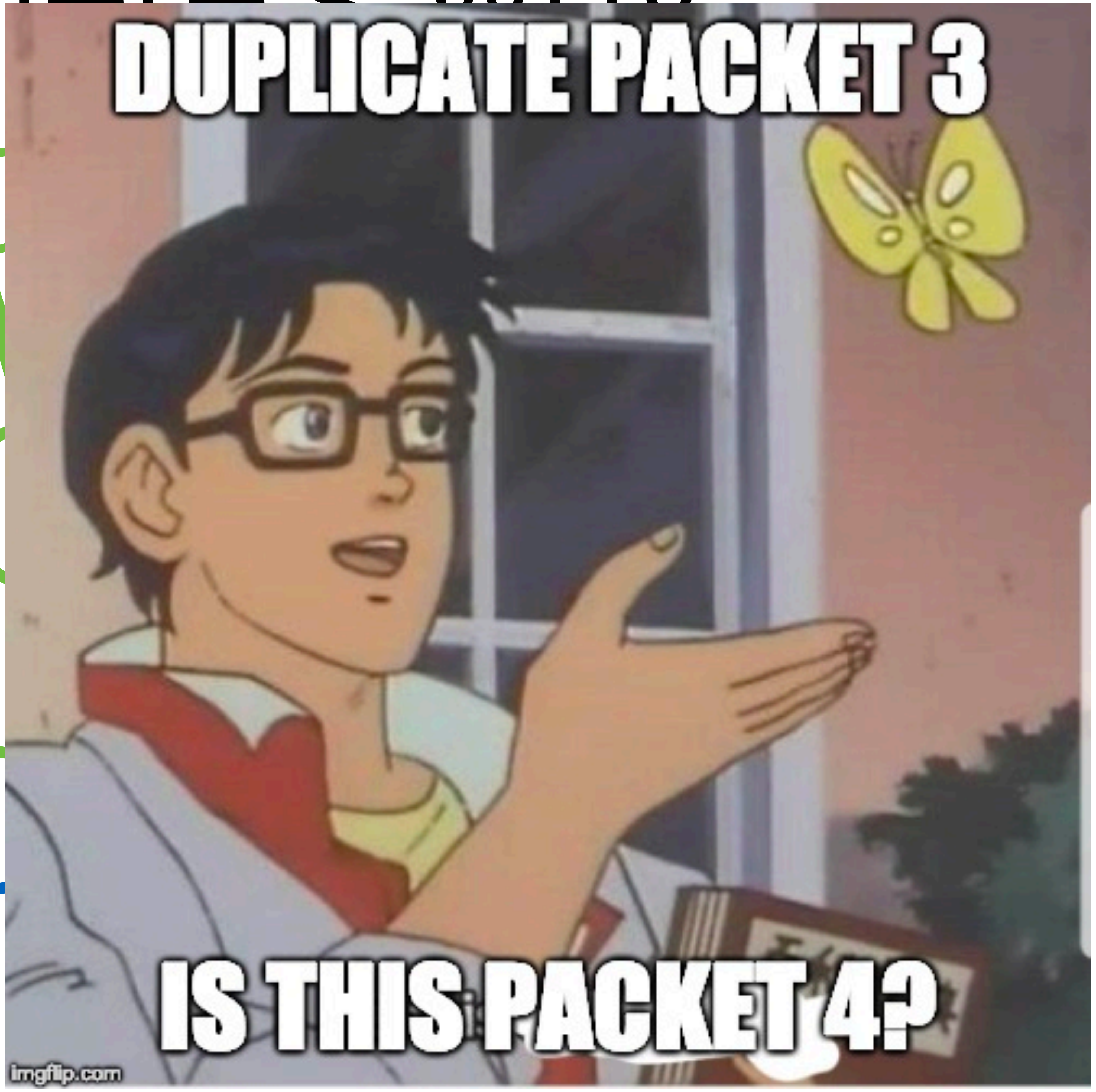
How do we put the file back together again after packetization?



But maybe we could just standardize this —- say each packet is in row-order starting from top left. Would we still need sequence numbers for the protocol?

We do, and here’s why





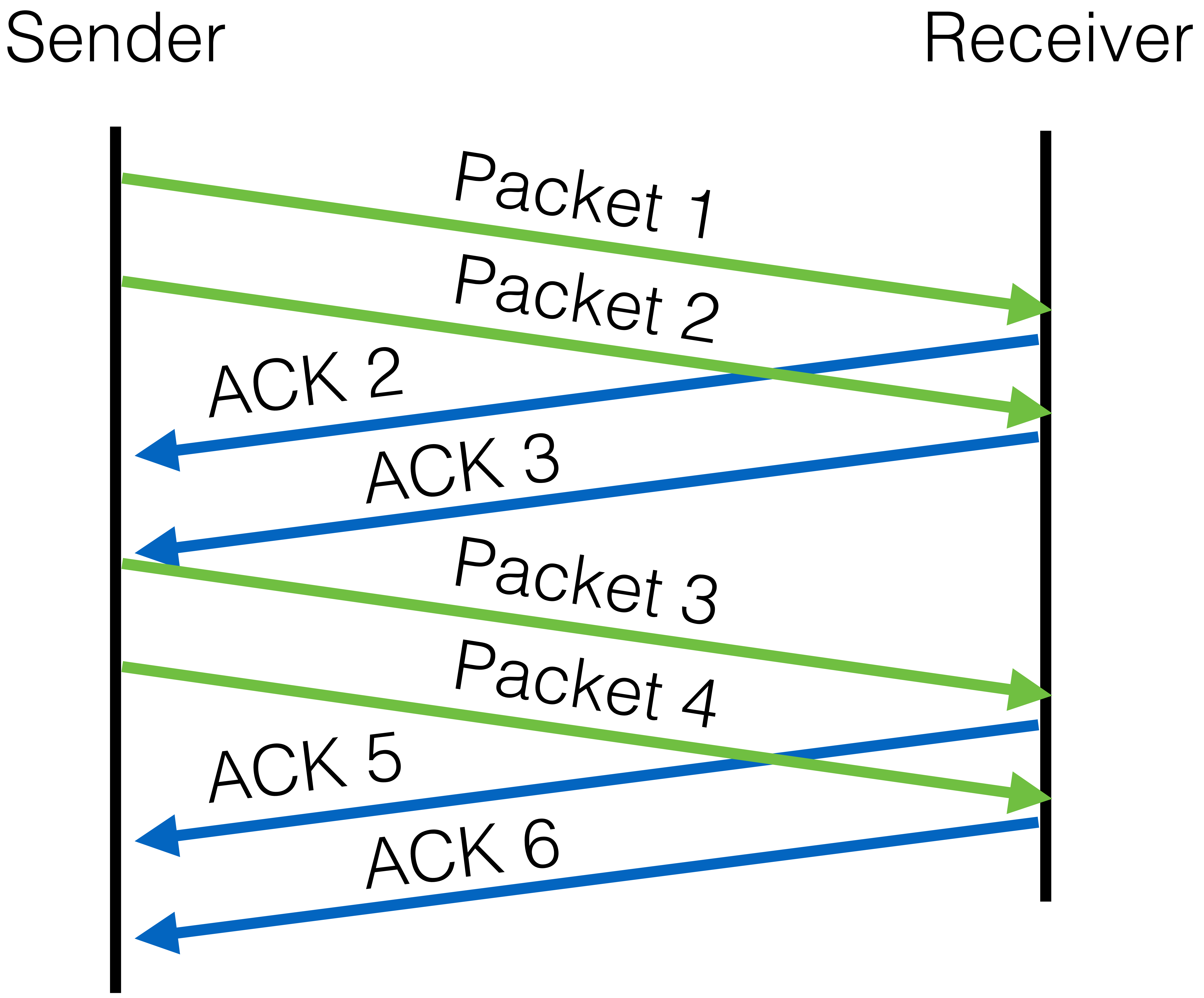
Sequence numbers are needed for reliability.

What's wrong with stop-and-wait?

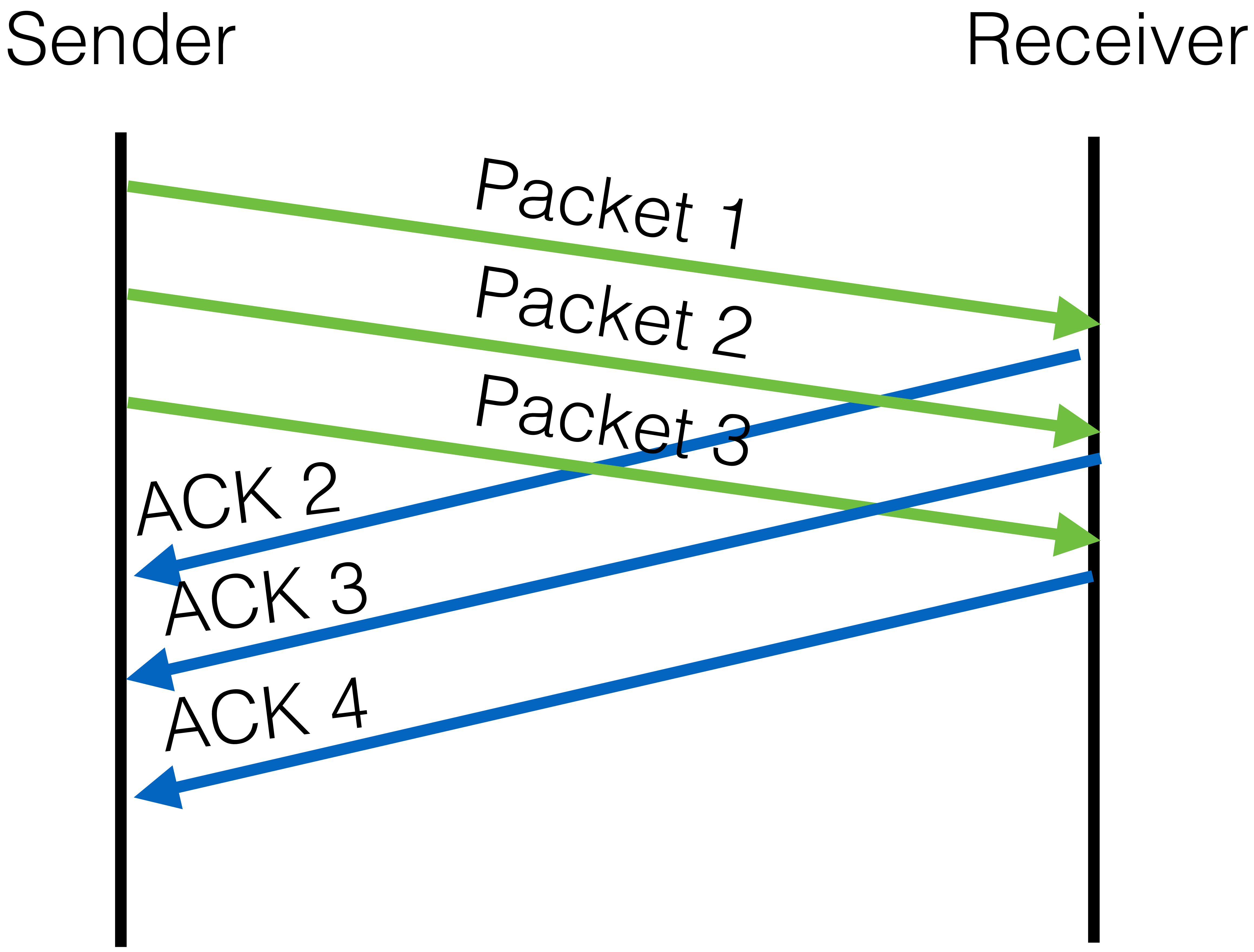
It’s slow!

How might we fix it?

Making Stop and Wait faster…

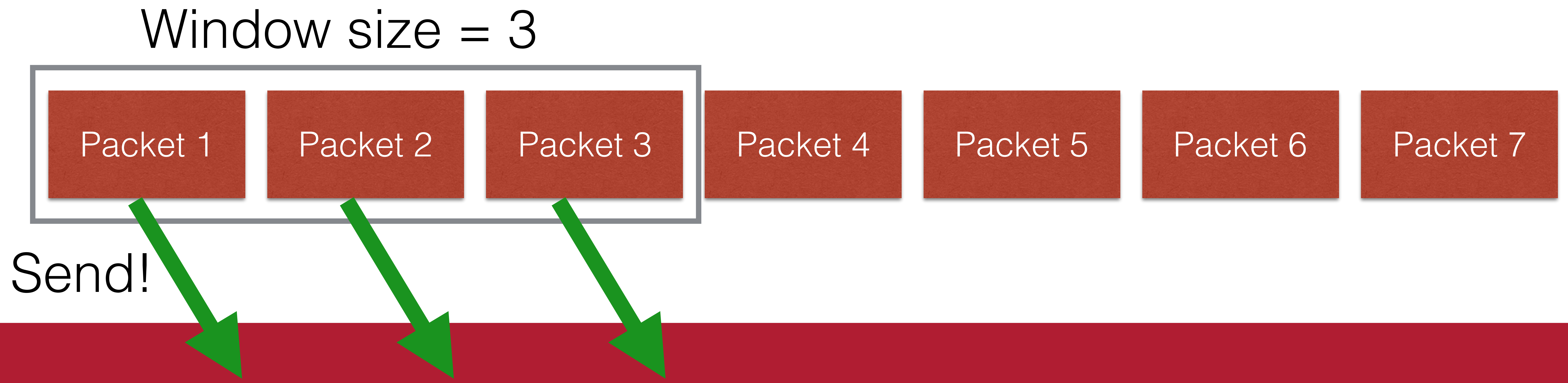


…and faster…



Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Windowing improves the efficiency of a transport protocol.
- We say the window "slides" when a packet is asked.

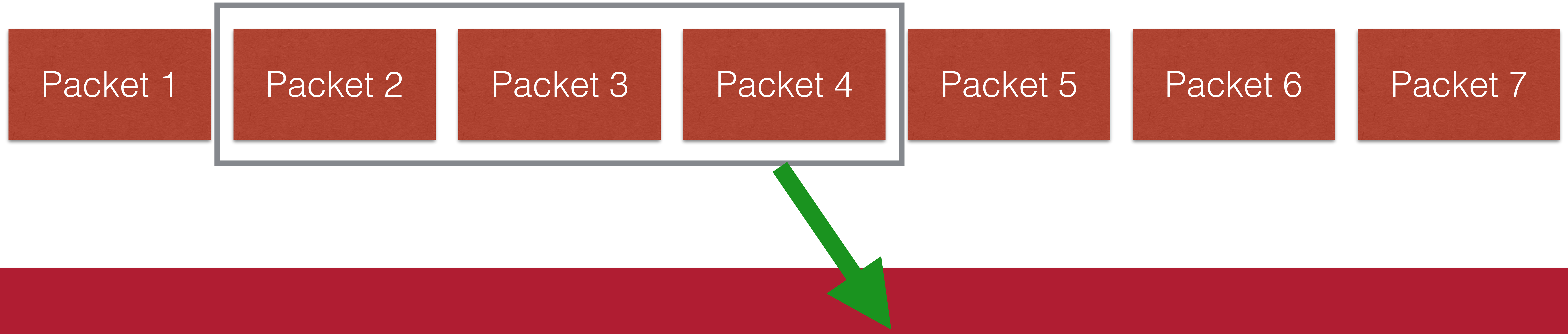


Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Windowing improves the efficiency of a transport protocol.
- We say the window "slides" when a packet is asked.

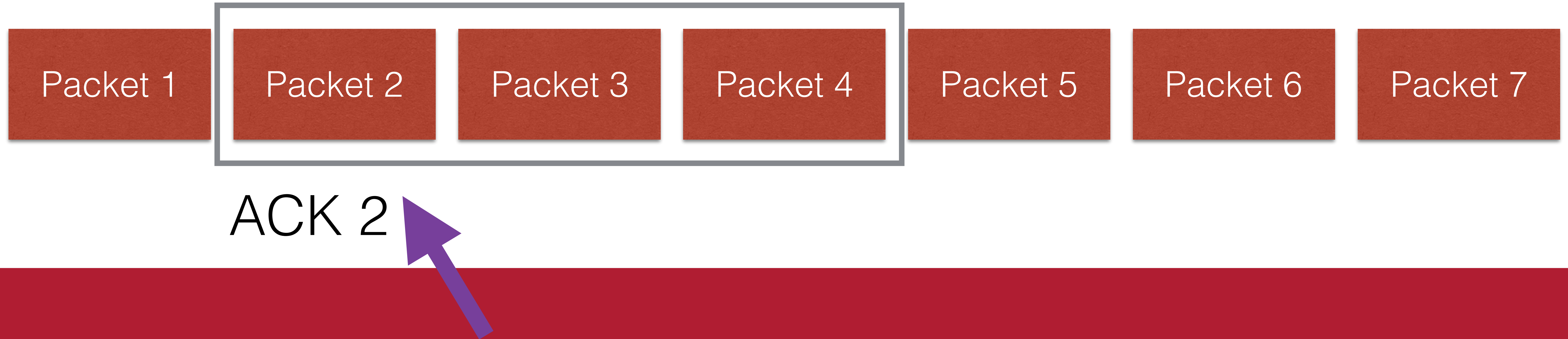
Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Windowing improves the efficiency of a transport protocol.
- We say the window "slides" when a packet is asked.



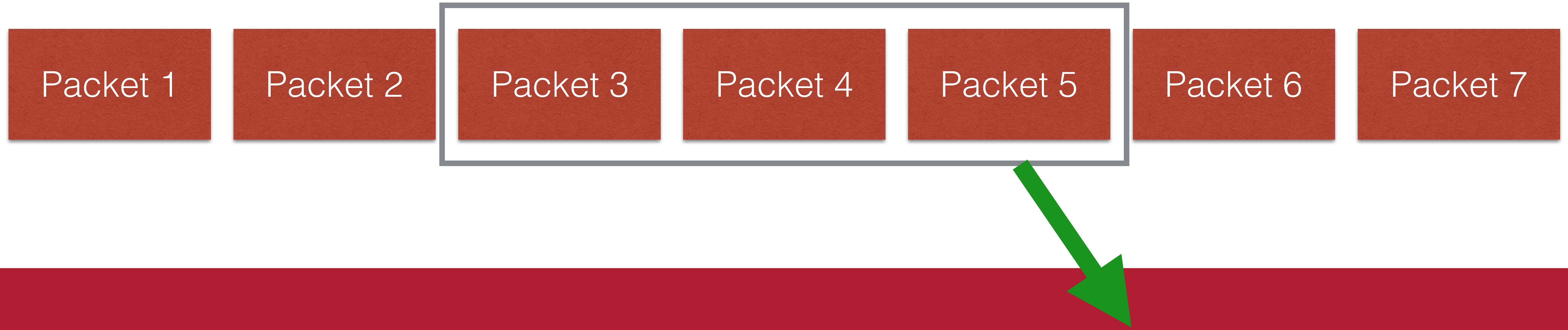
Sliding Windows

- A sender’s “window” contains a set of packets that have been transmitted but not yet acked.
- Windowing improves the efficiency of a transport protocol.
- We say the window “slides” when a packet is asked.



Sliding Windows

- A sender’s “window” contains a set of packets that have been transmitted but not yet acked.
- Windowing improves the efficiency of a transport protocol.
- We say the window “slides” when a packet is asked.



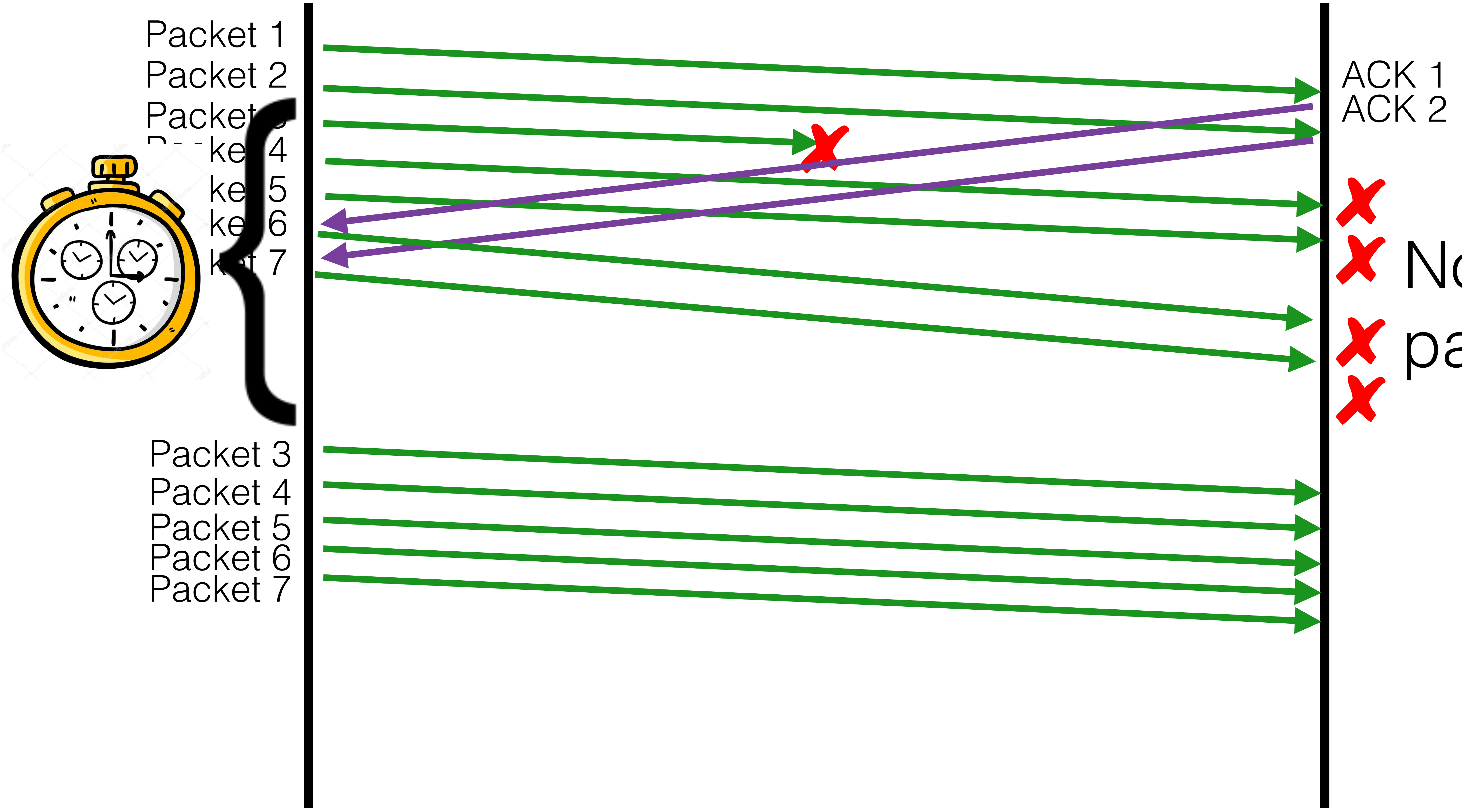
Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Sliding windows improve the efficiency of a transport protocol.
- Two questions we need to answer to use windows:
  - (1) How do we handle loss with a windowed approach?
  - (2) How big should we make the window?

Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Sliding windows improve the efficiency of a transport protocol.
- Two questions we need to answer to use windows:
    - (1) How do we handle loss with a windowed approach?
    - (2) How big should we make the window?

Approach #1: Go Back N



Not the expected packet — 3 — so ignore.

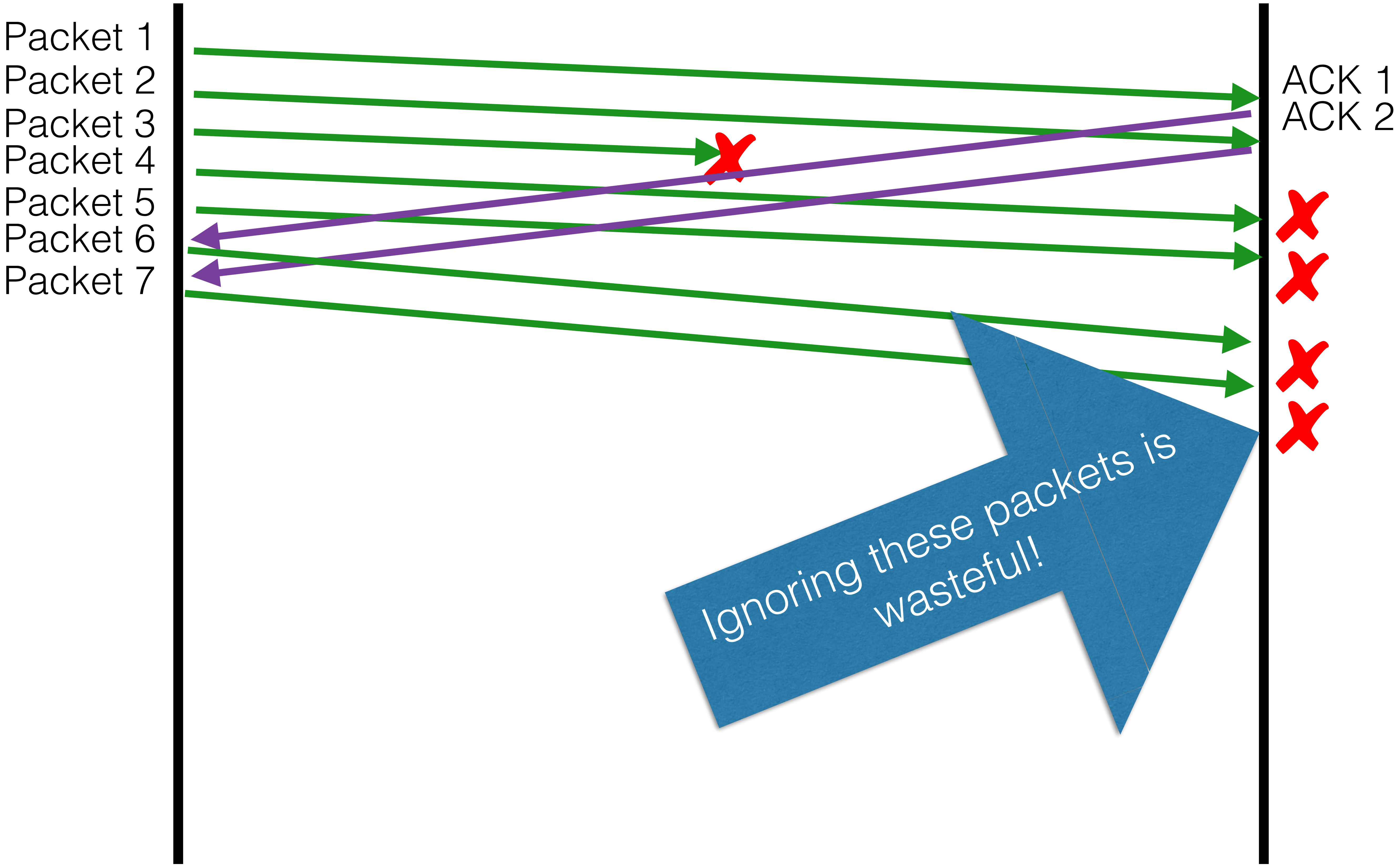
Go Back N

- **Sender:**
  - Send up to {n} packets at a time. Set a timeout timer for every packet.
  - On receiving an ACK, slide the window forward.
  - On timeout, retransmit the timeout packet, and everything after it in the window.
- **Receiver:**
  - On receive next expected sequence number, send an ACK
  - If packet is corrupted or has an unexpected sequence number, ignore it.

We don’t use Go Back N on the Internet… why not?

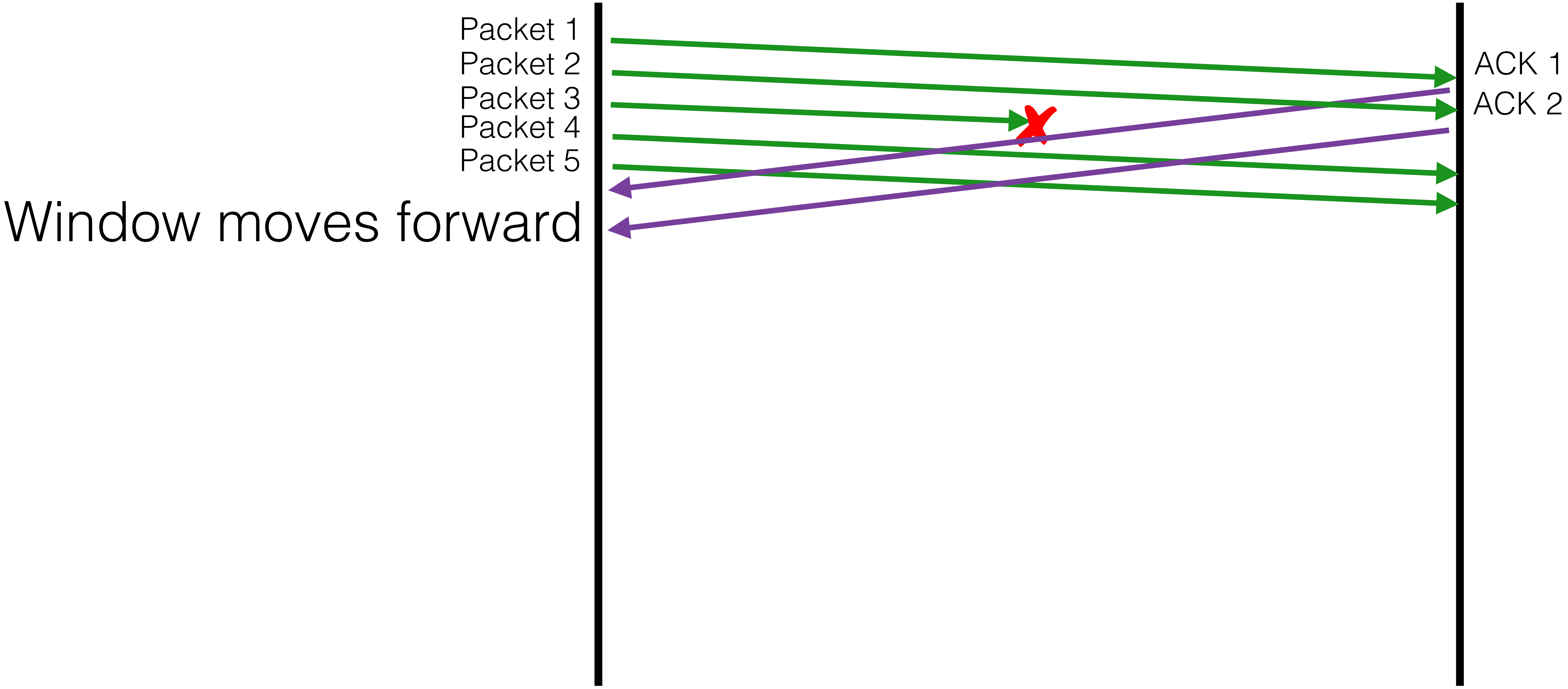
Loss recovery *works*… but it's not very efficient.

Approach #1: Go Back N



Approach #2: Selective Repeat

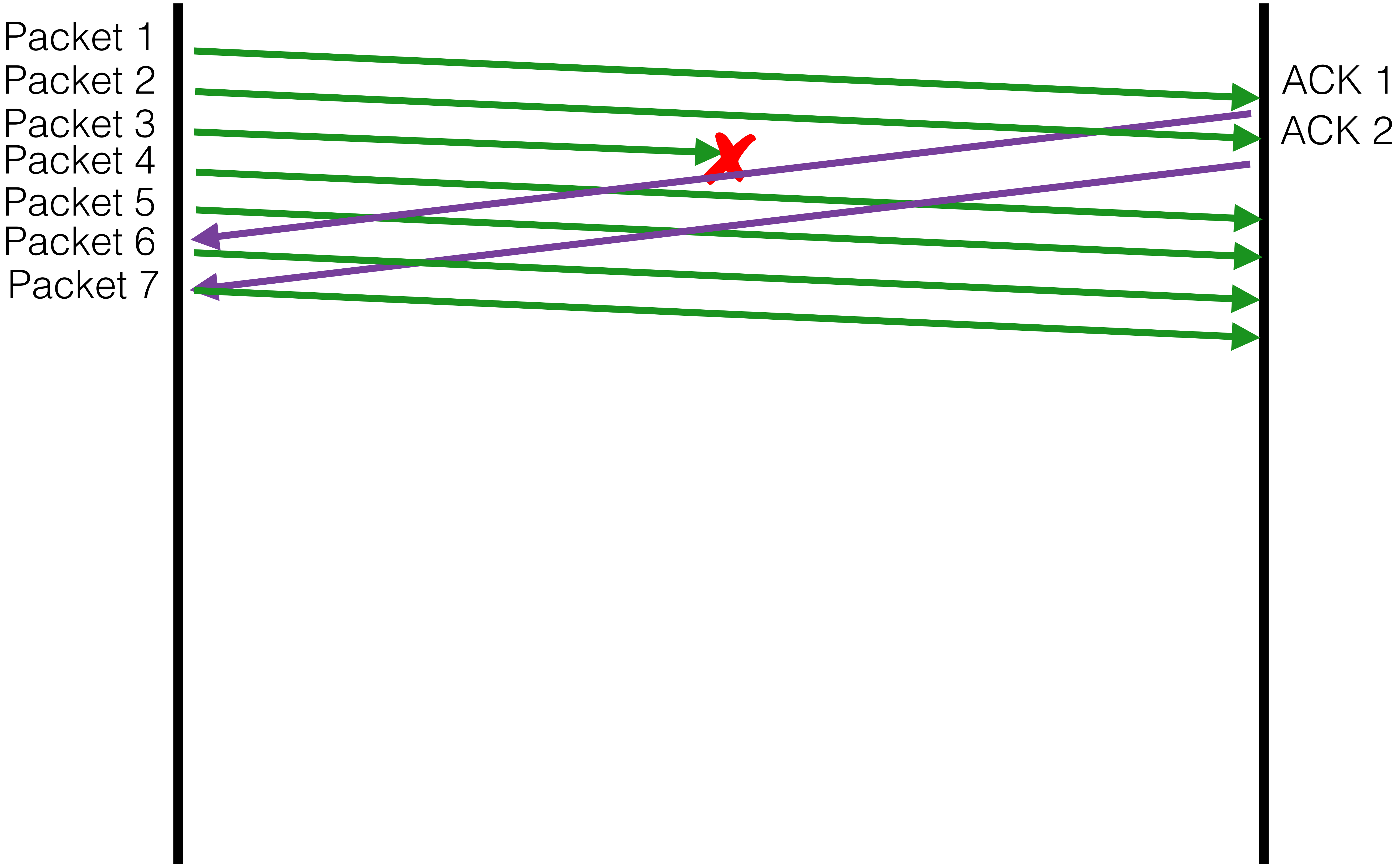
Approach #2: Selective Repeat



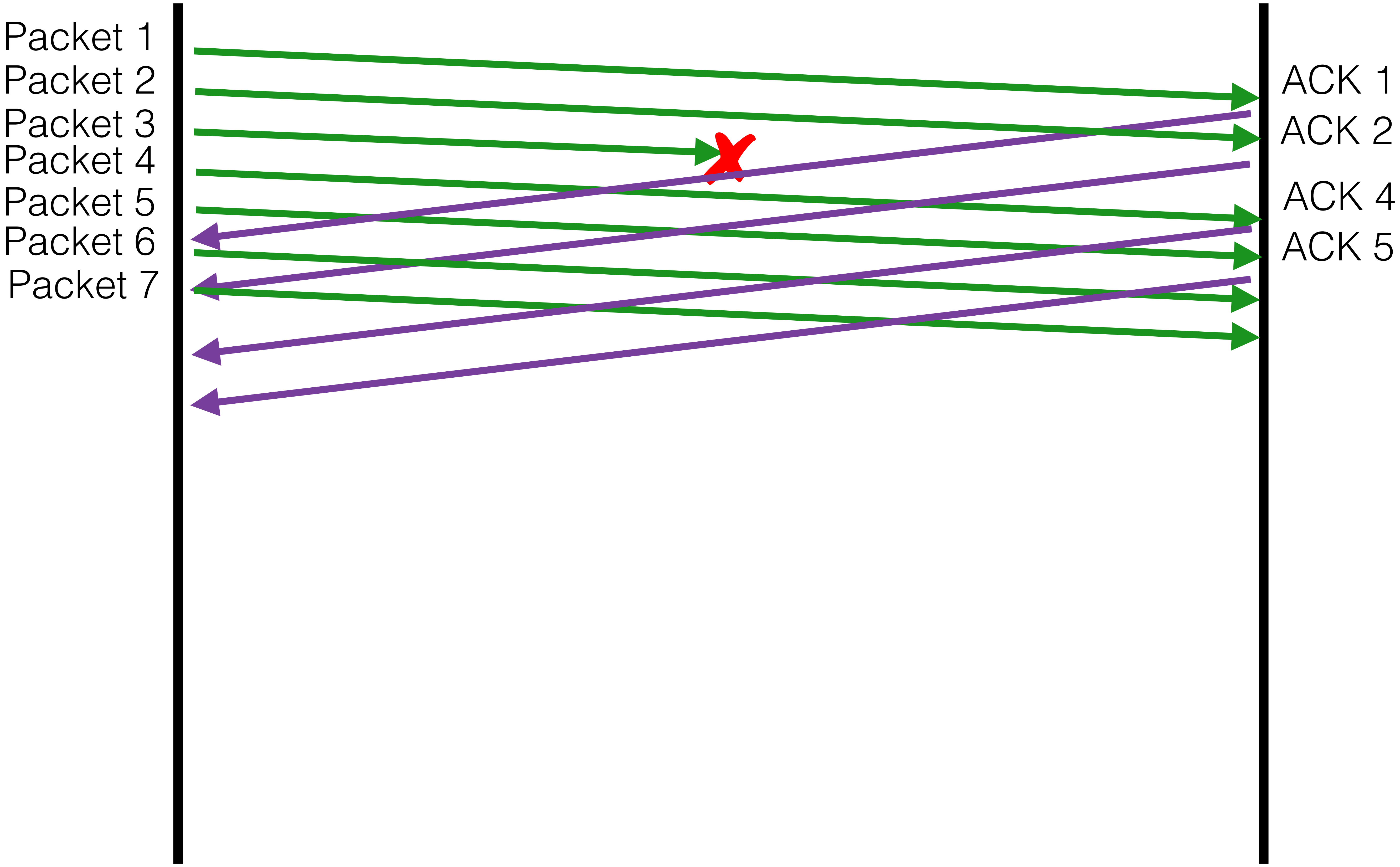
Approach #2: Selective Repeat

Approach #2: Selective Repeat

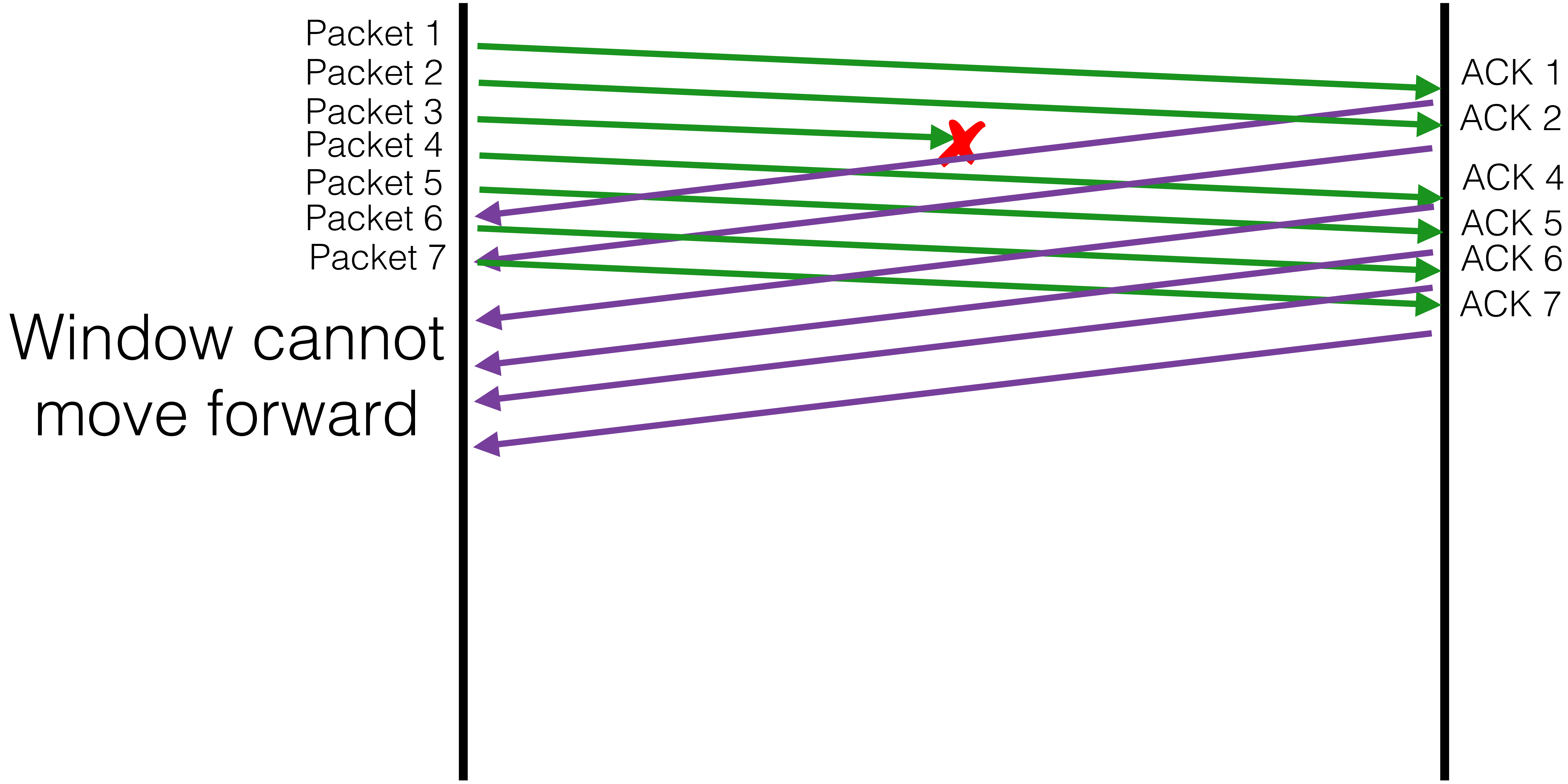
Approach #2: Selective Repeat



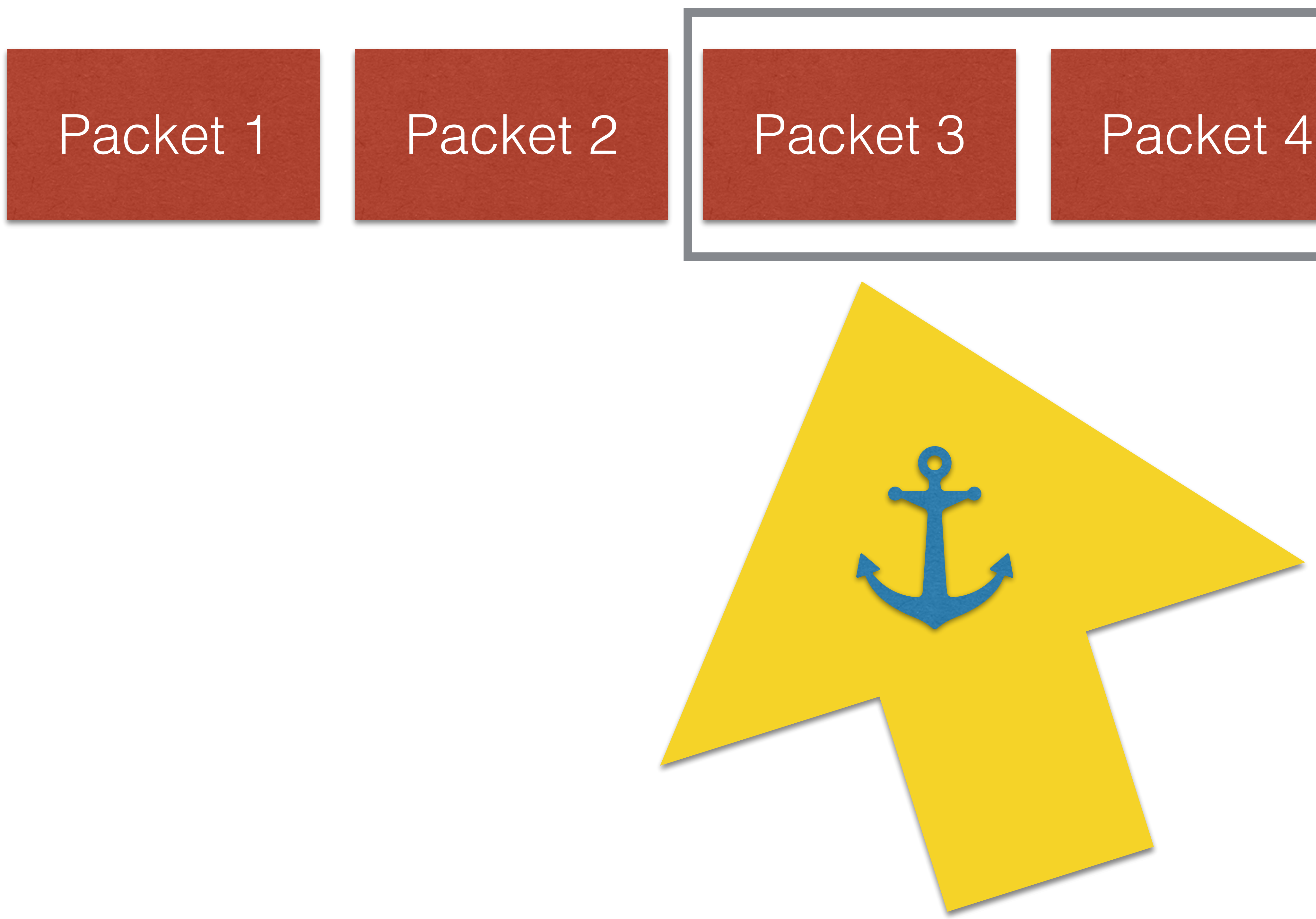
Approach #2: Selective Repeat



Approach #2: Selective Repeat

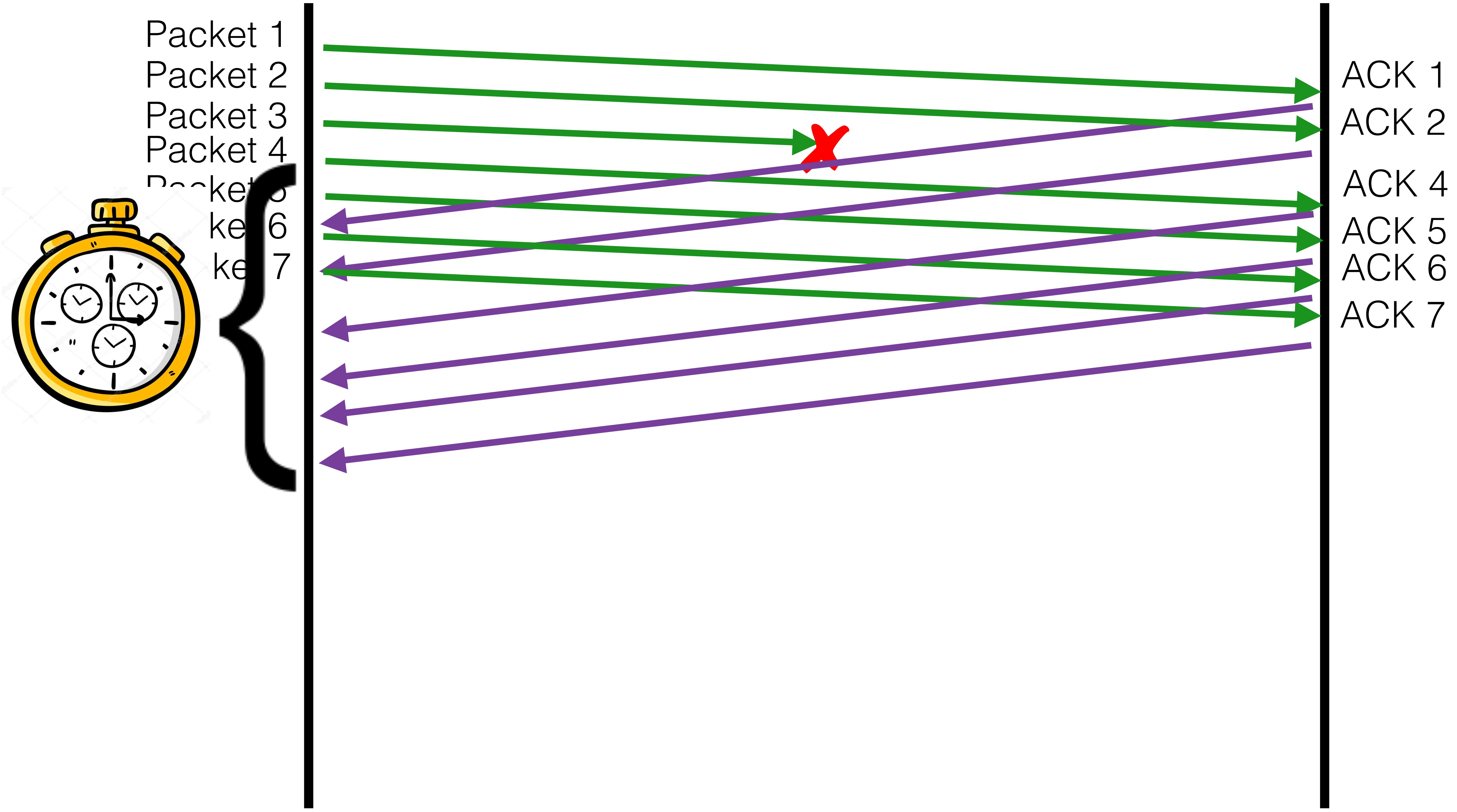


Approach #2: Selective Repeat

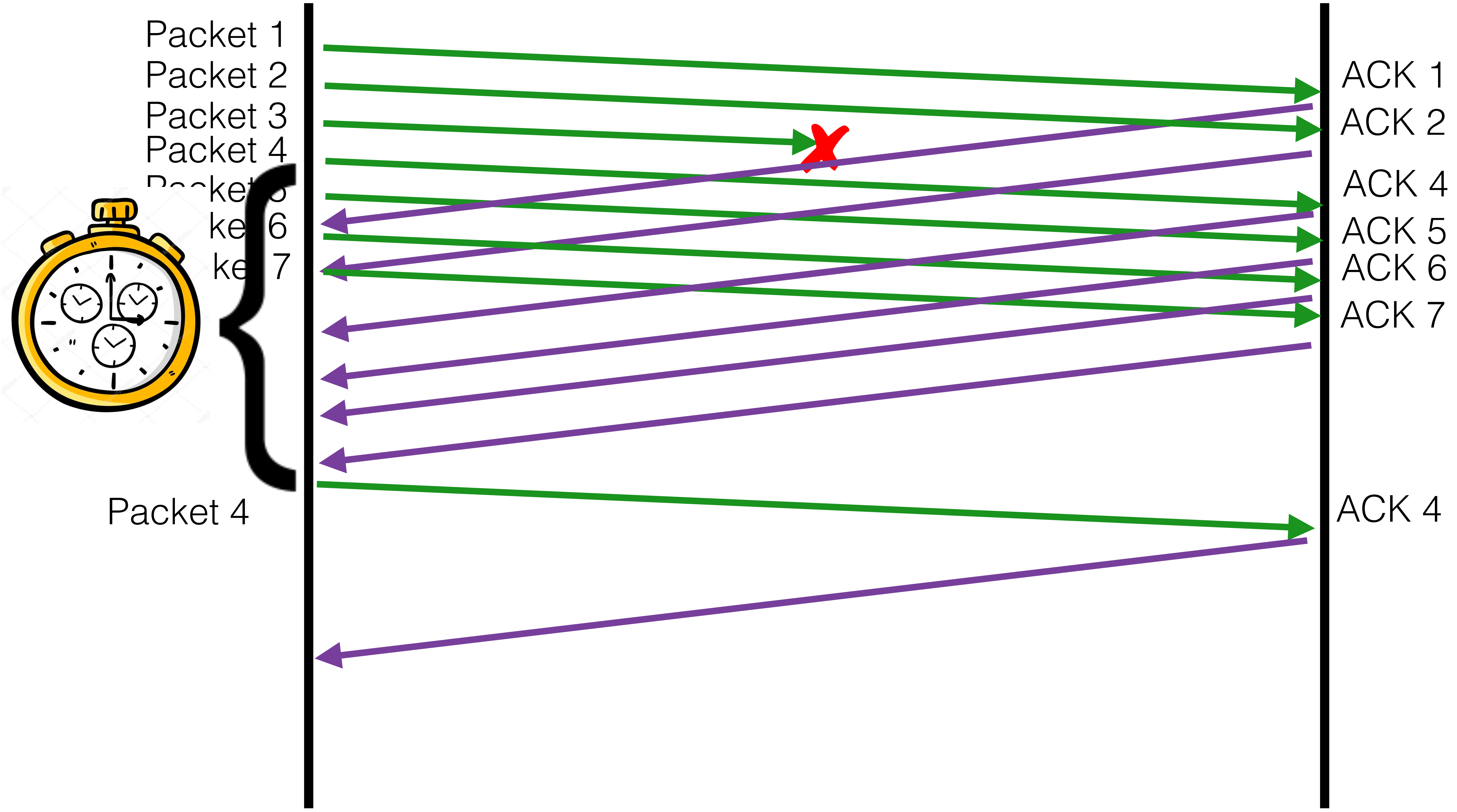


Missing packet 3 stops the window from moving forward

Approach #2: Selective Repeat



Approach #2: Selective Repeat

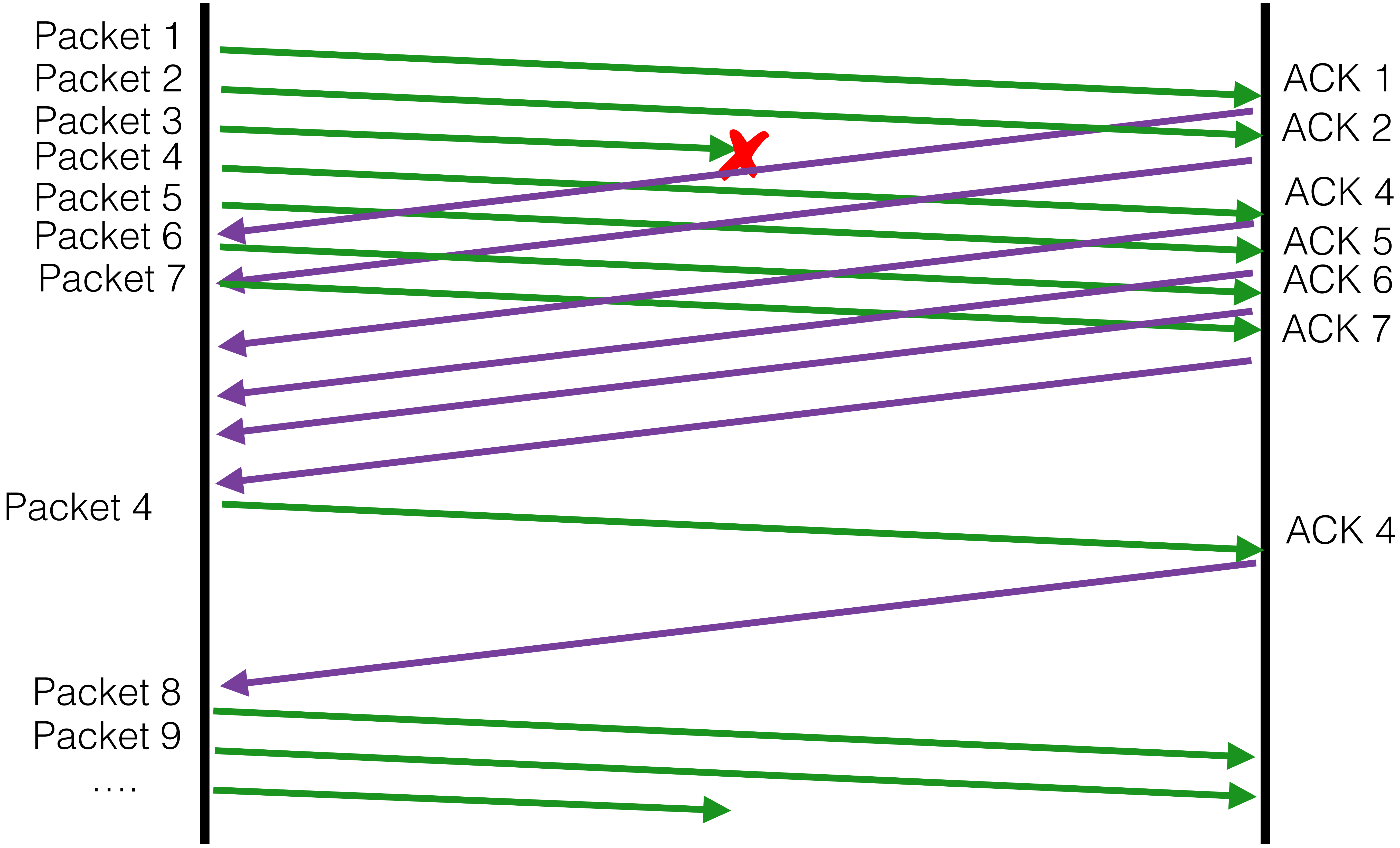


Approach #2: Selective Repeat

Approach #2: Selective Repeat

Packet 1 | Packet 2 | Packet 3 | Packet 4 | Packet 5 | Packet 6 | Packet 7 | Packet 8 | Packet 9

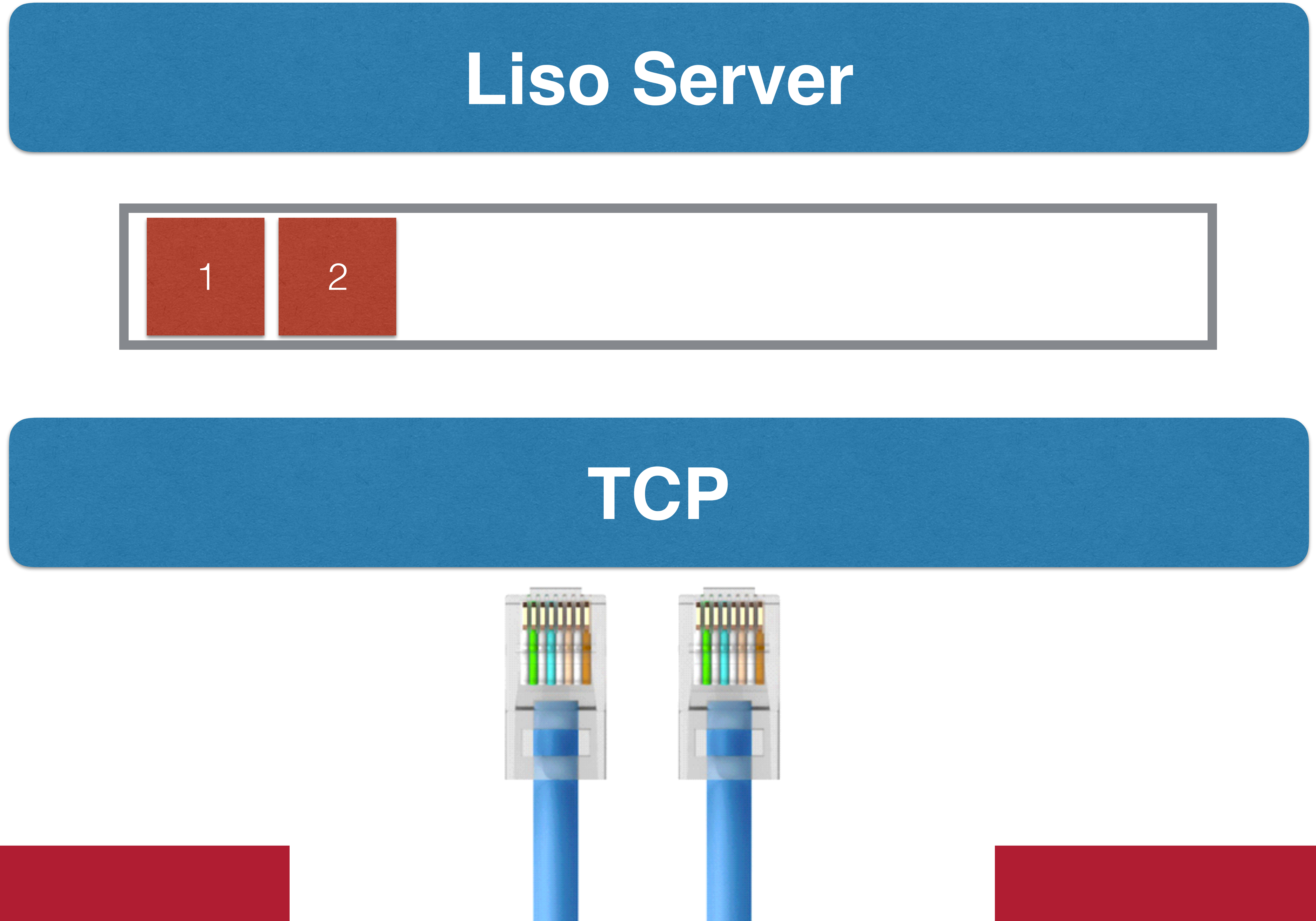
Approach #2: Selective Repeat



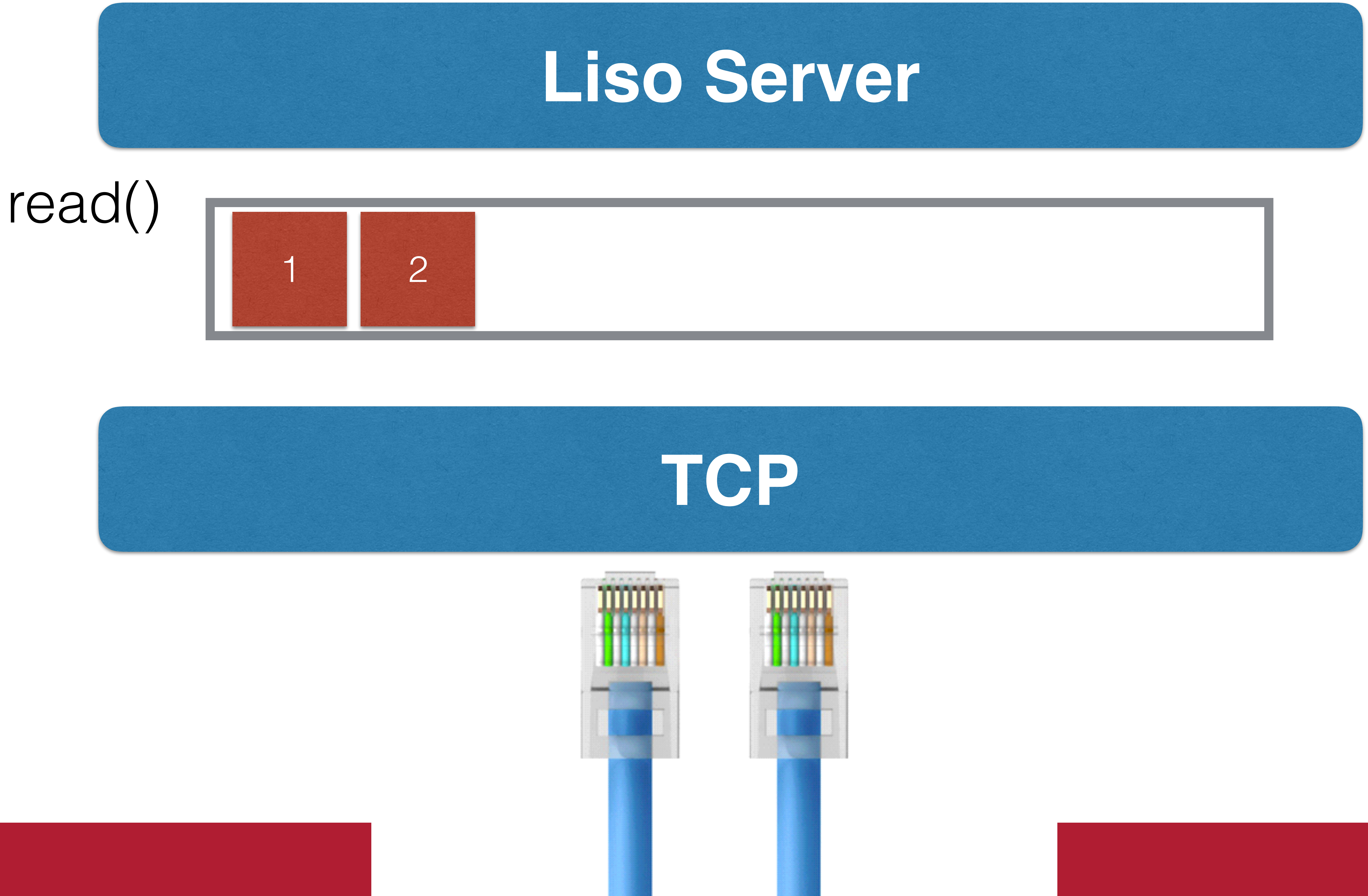
Selective Repeat

- **Sender:**
  - Send packets from the window. Set timeout for each packet.
  - On receiving ACKs for the “left side” of the window, slide forward.
    - Send packets that have now entered the window.
  - On timeout, retransmit only the timed out packet
- **Receiver**
  - Keep a buffer of size of the window.
  - On receiving packets, send ACKs for every packet.
  - If packets come in out of order, just store them in the buffer and send ACK anyway.

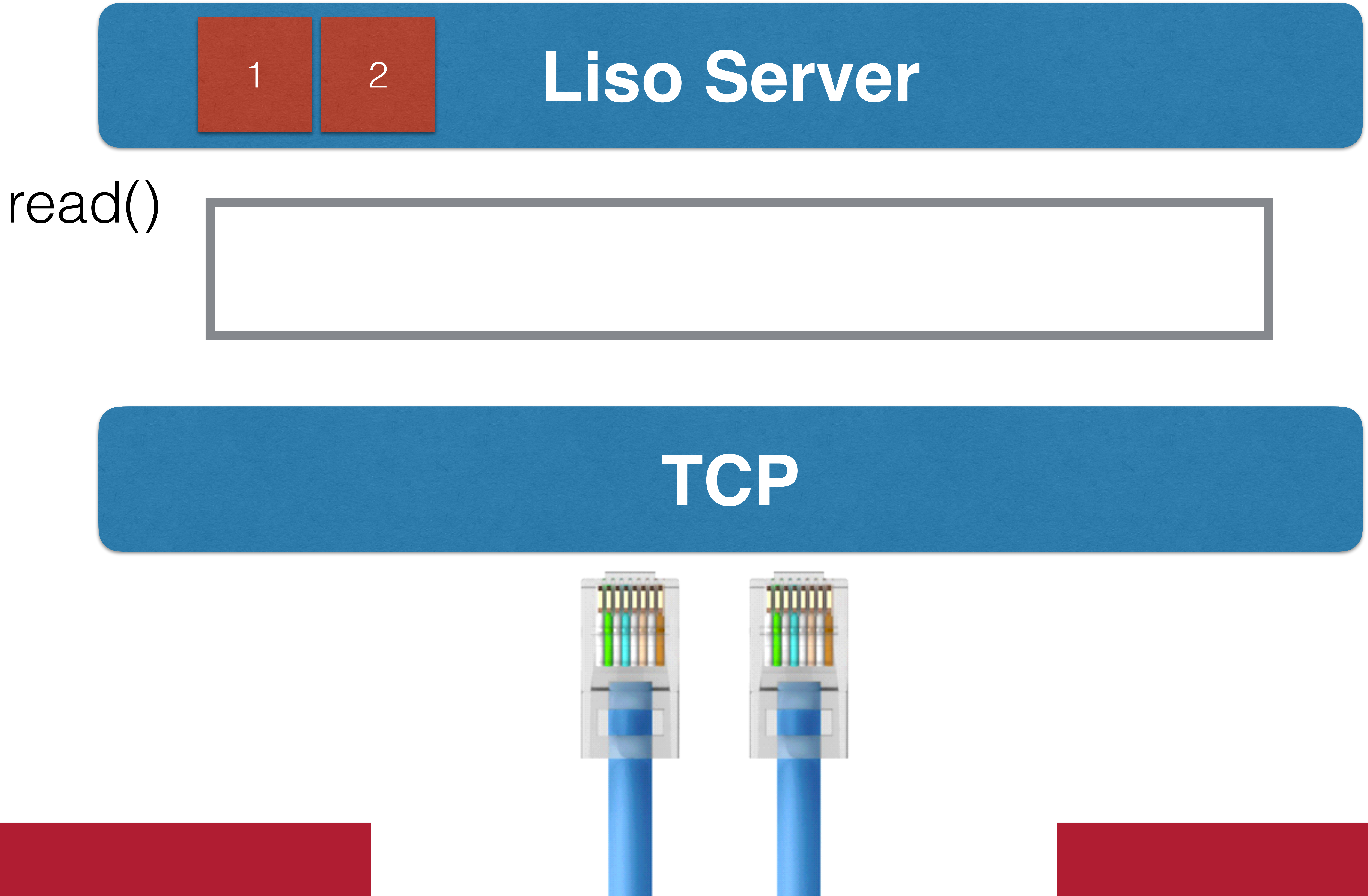
Receive Buffer



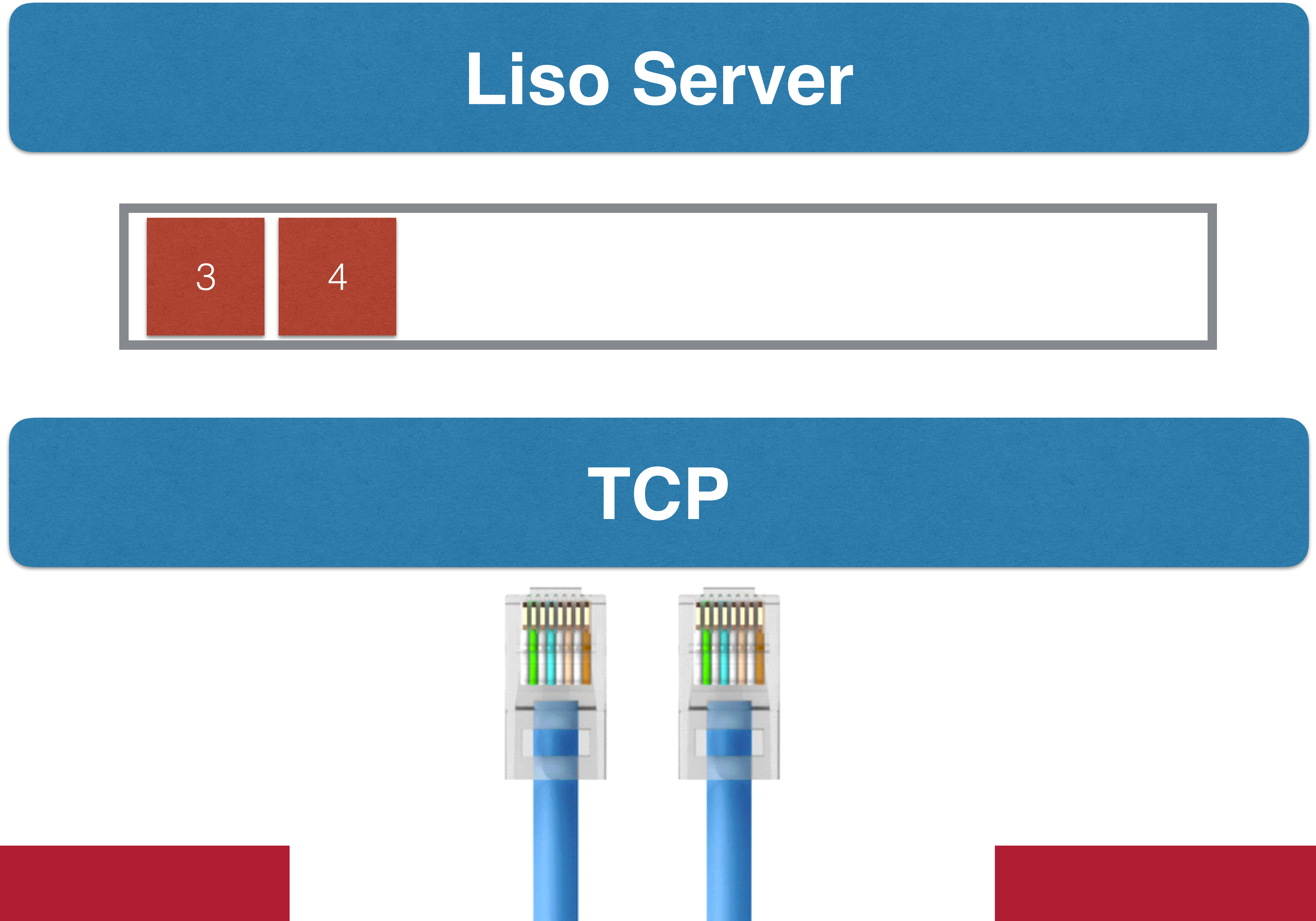
Receive Buffer



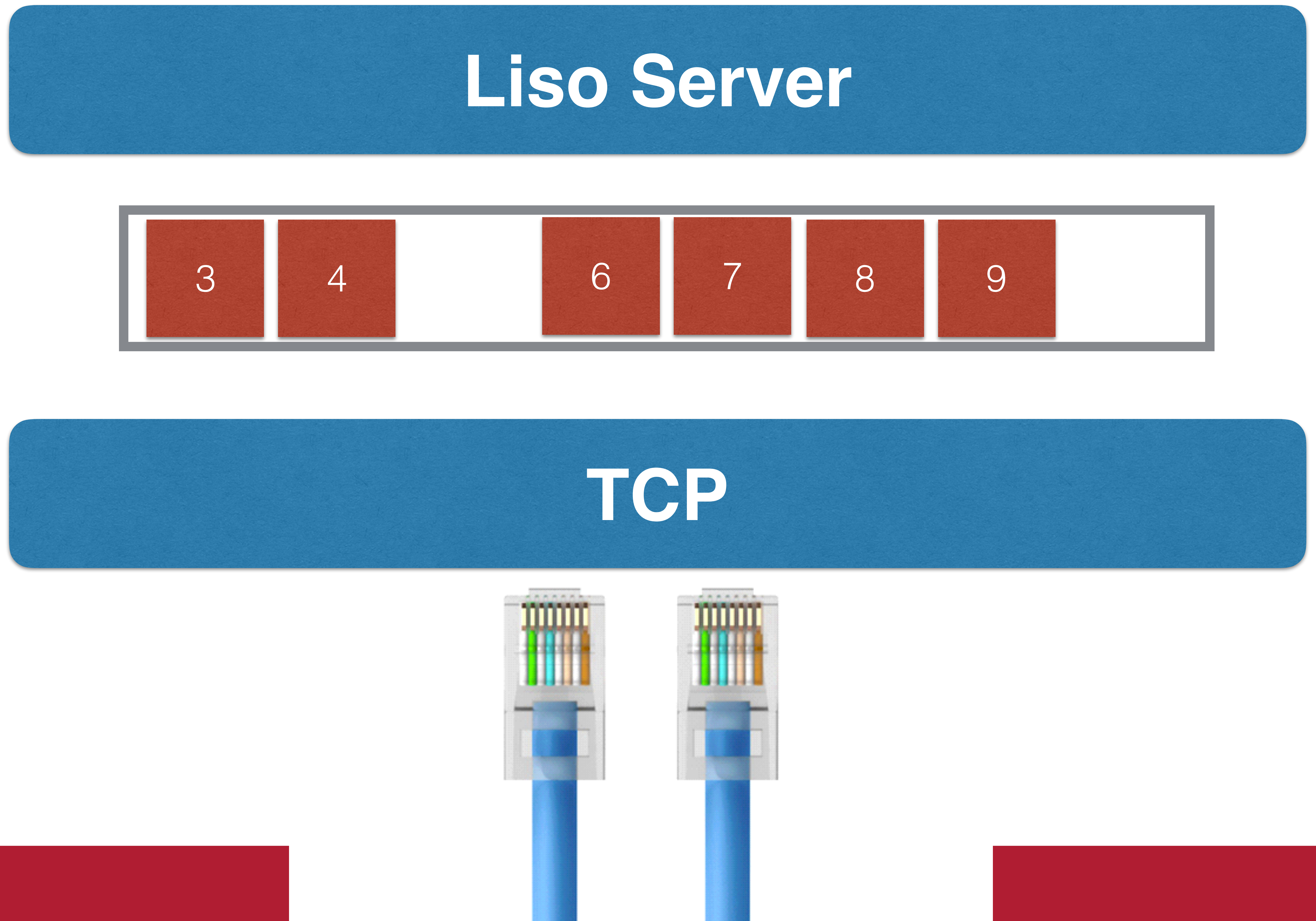
Receive Buffer



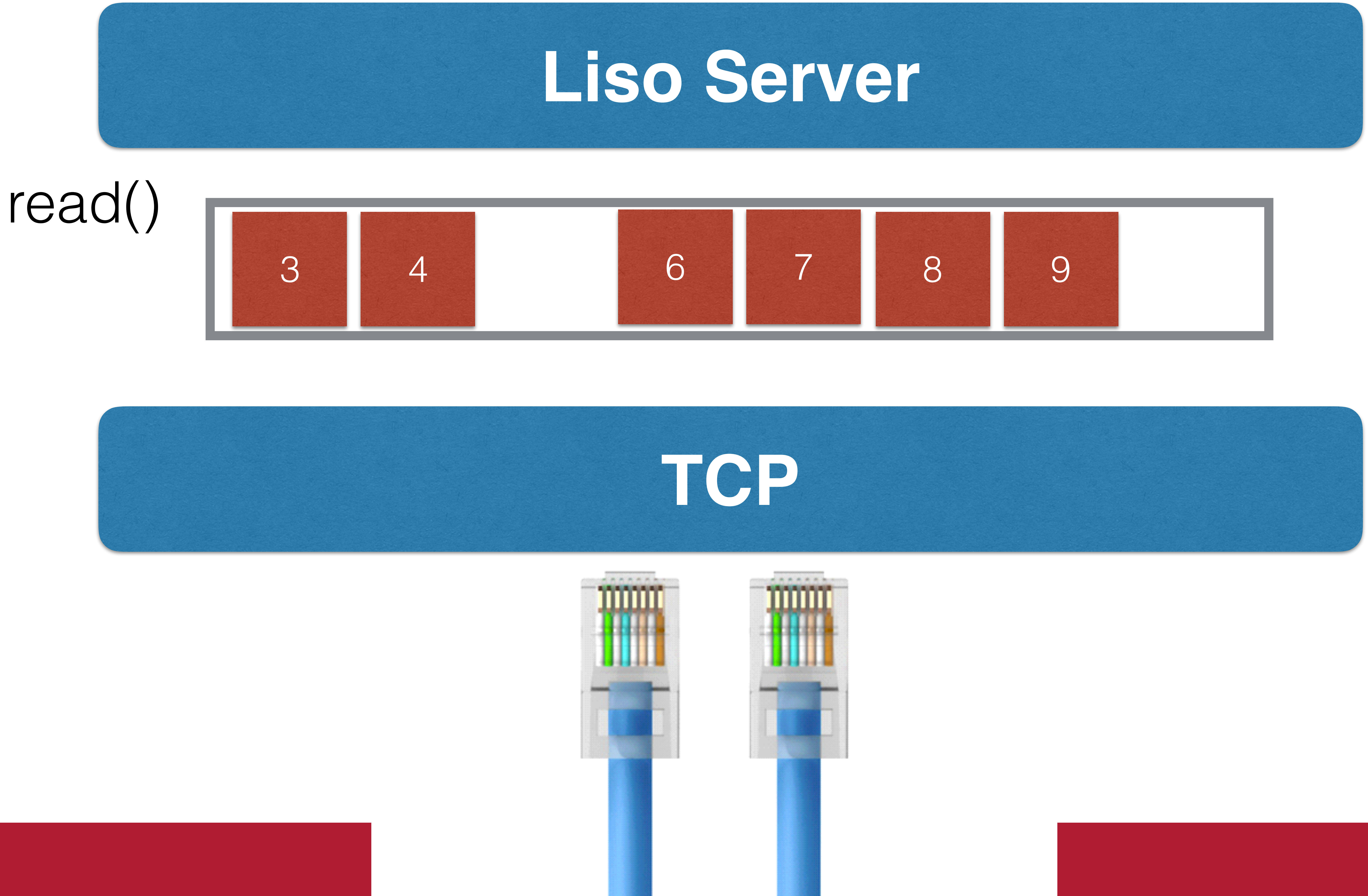
Receive Buffer



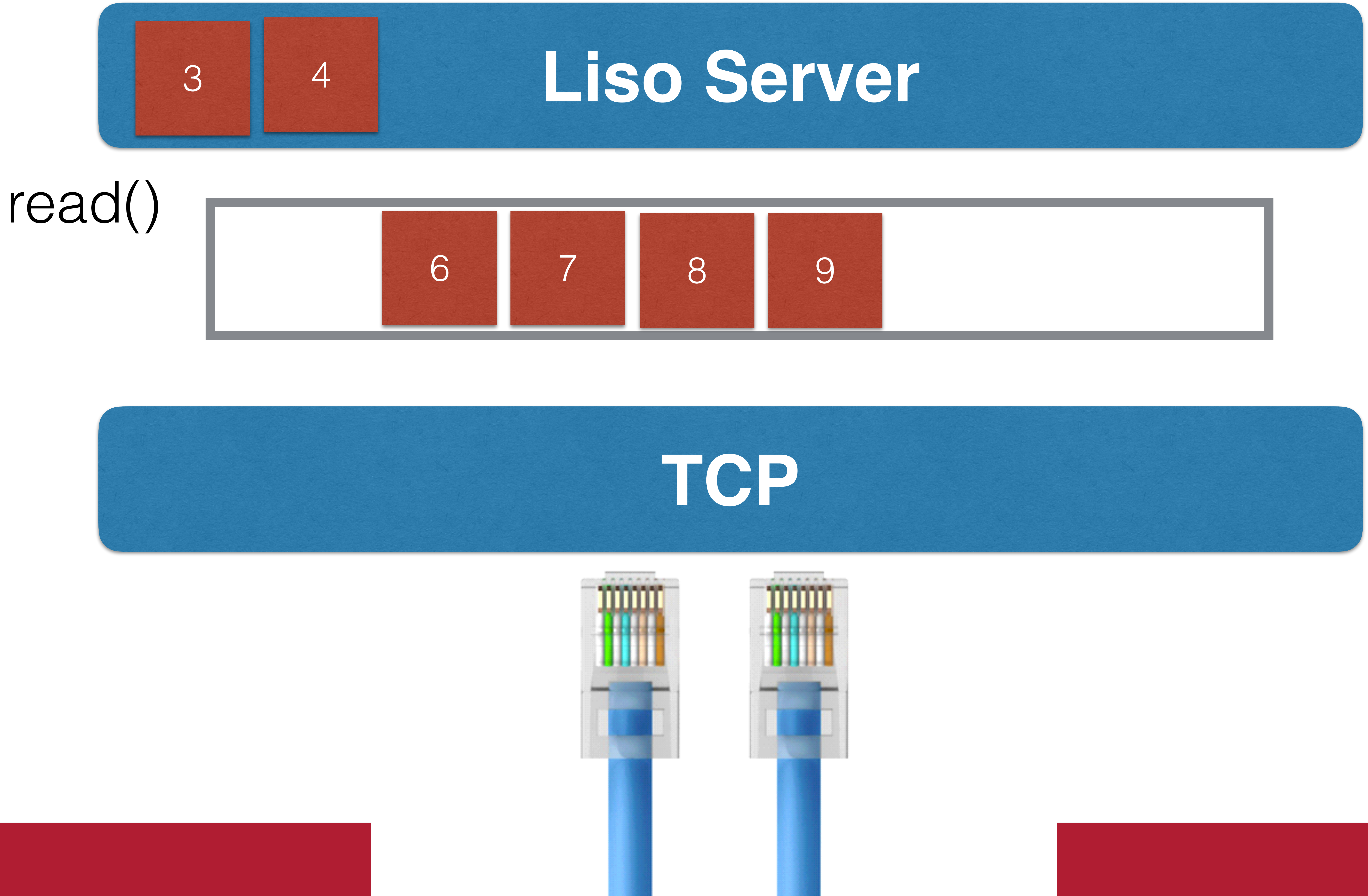
Receive Buffer



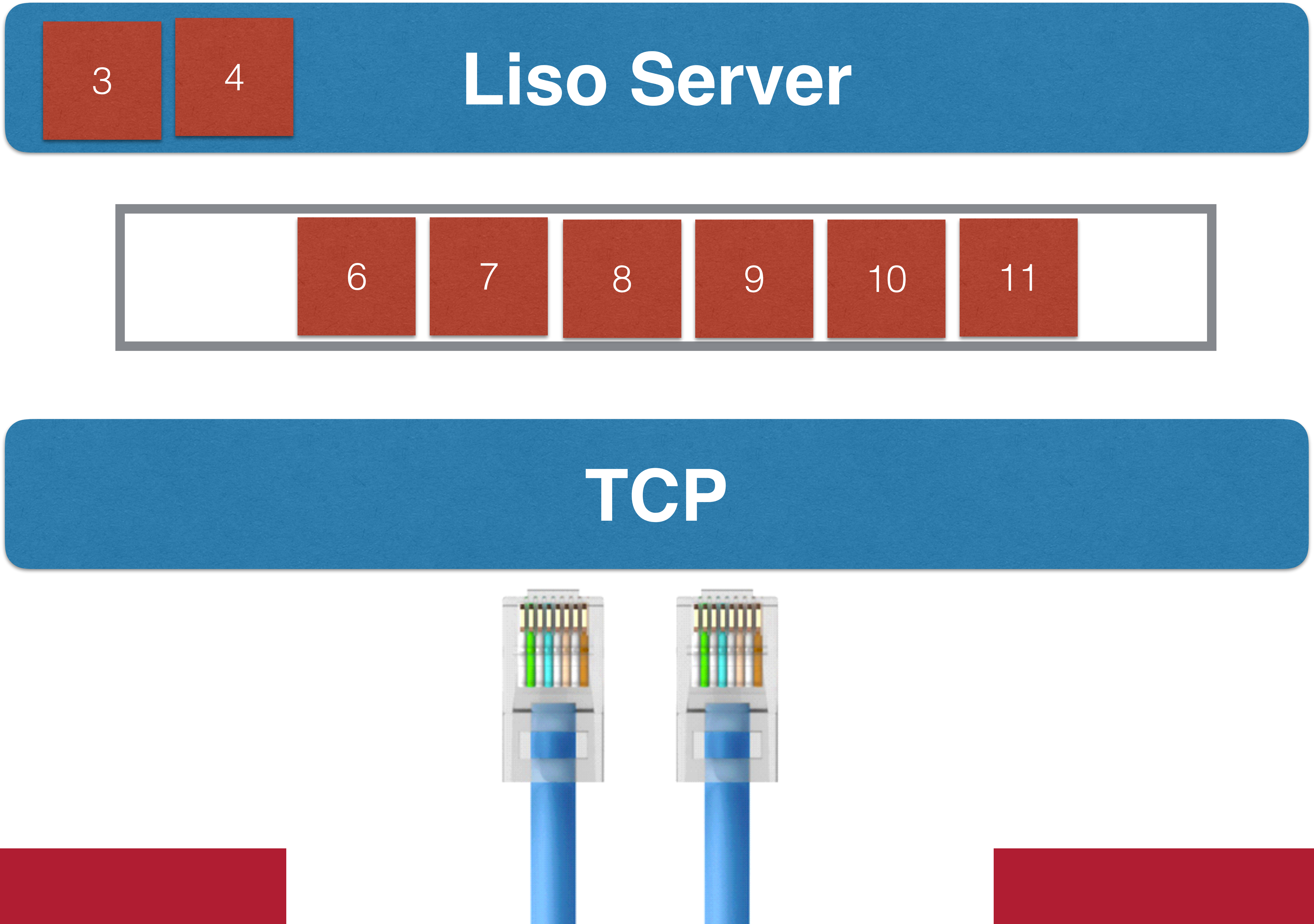
Receive Buffer



Receive Buffer

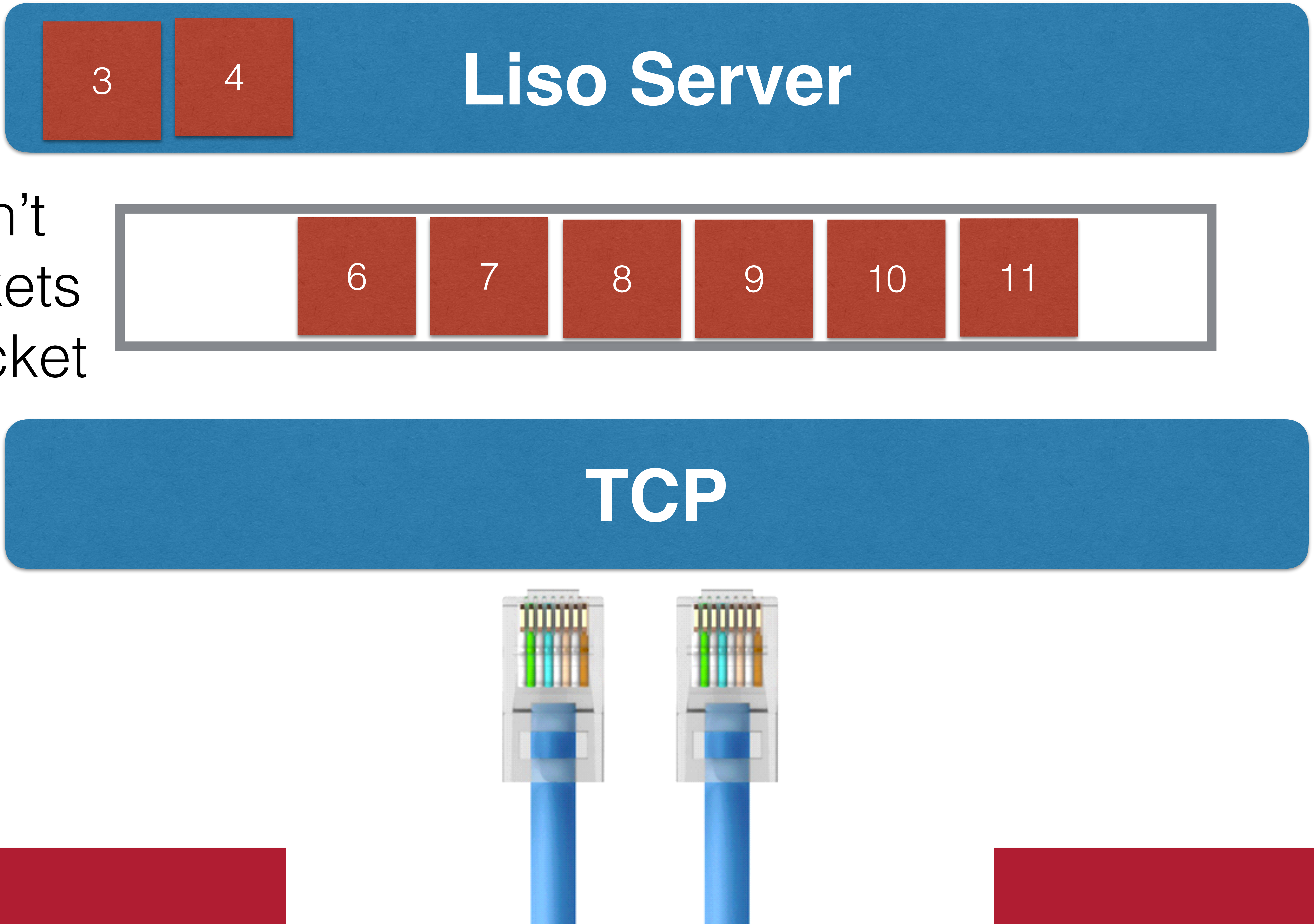


Receive Buffer



Receive Buffer

Application can’t take in any packets until missing packet comes in



Handling Loss

- Go-Back-N and Selective Repeat both handle loss, while allowing lots of packets in flight.
- Selective repeat is more efficient at recovering from failure.

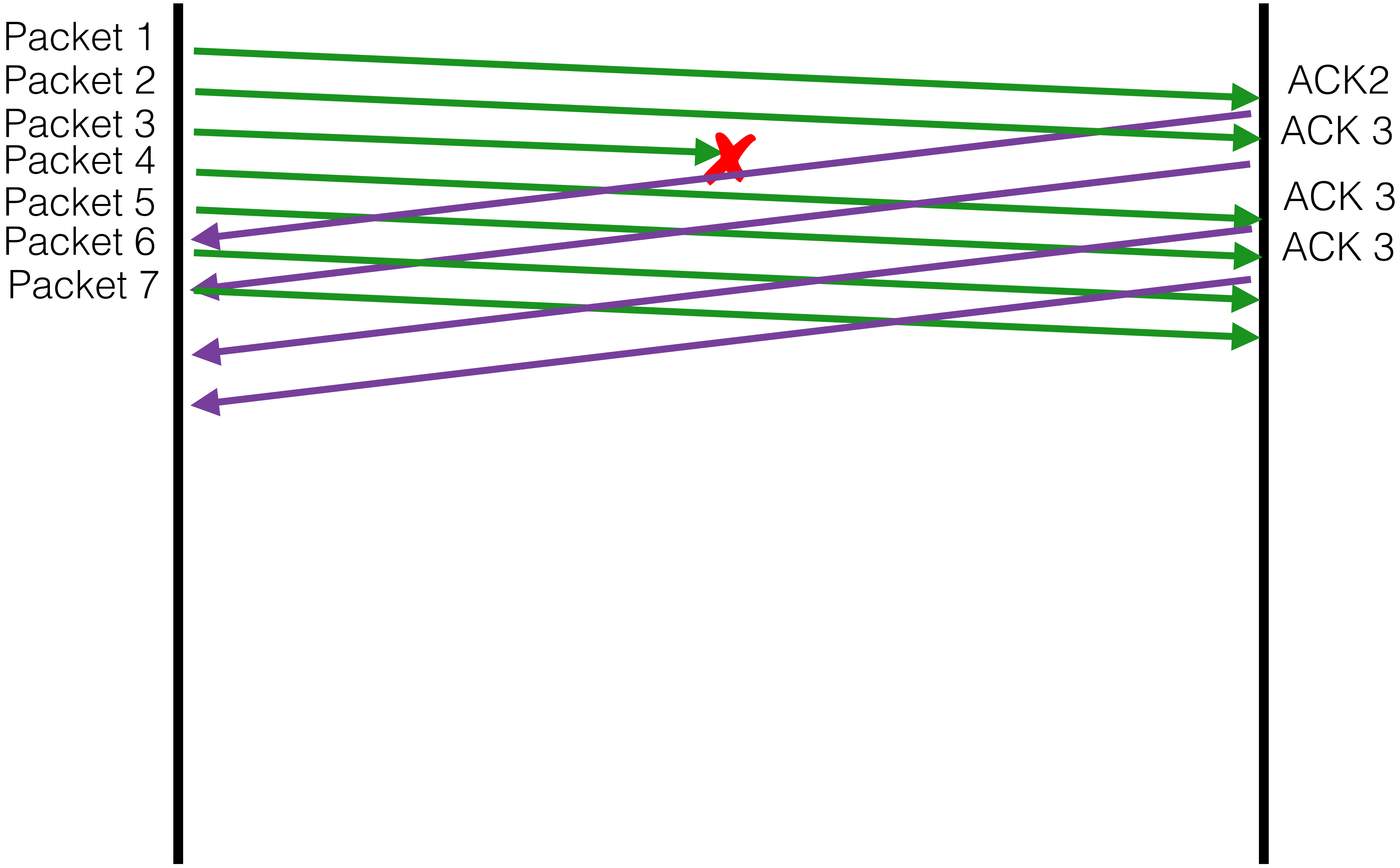
What does TCP Do?

- TCP is like Selective Repeat, but…
  - It uses *cumulative ACKs*
  - Instead of using per-packet sequence numbers, it uses per-byte sequence numbers.
    - e.g. if packet #1 has 1000 bytes of payload data, packet #2 will have the sequence number 1001
  - It implements *fast recovery (we'll discuss this on Tuesday)*

Basic ACKs vs Cumulative ACKs

- Basic ACKs: “ACK n” means “I just received packet n”
- Cumulative ACKs: “ACK n” means, “I have received all packets up to (but not including) n”

Cumulative ACK



What does TCP Do?

- TCP is like Selective Repeat, but…
  - It uses *cumulative ACKs*
  - Instead of using per-packet sequence numbers, it uses per-byte sequence numbers.
    - e.g. if packet #1 has 1000 bytes of payload data, packet #2 will have the sequence number 1001
  - It implements *fast recovery (we'll discuss this on Tuesday)*

Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.

You now know most of this

- Sliding windows improve the effic… a transport protocol.
- Two questions we need to answer … windows:
  - (1) How do we handle loss with a windowed approach?
  - (2) How big should we make the window?

Sliding Windows

- A sender's "window" contains a set of packets that have been transmitted but not yet acked.
- Sliding windows improve the efficiency of a transport protocol.
- Two questions we need to answer to use windows:
  - (1) How do we handle loss with a windowed approach?
  - (2) How big should we make the window?

We’ll figure this out on Thursday.

Have a great afternoon!