# ARQ, Sliding Window,

27 November 2024
Lecture 4

# Topics for Today

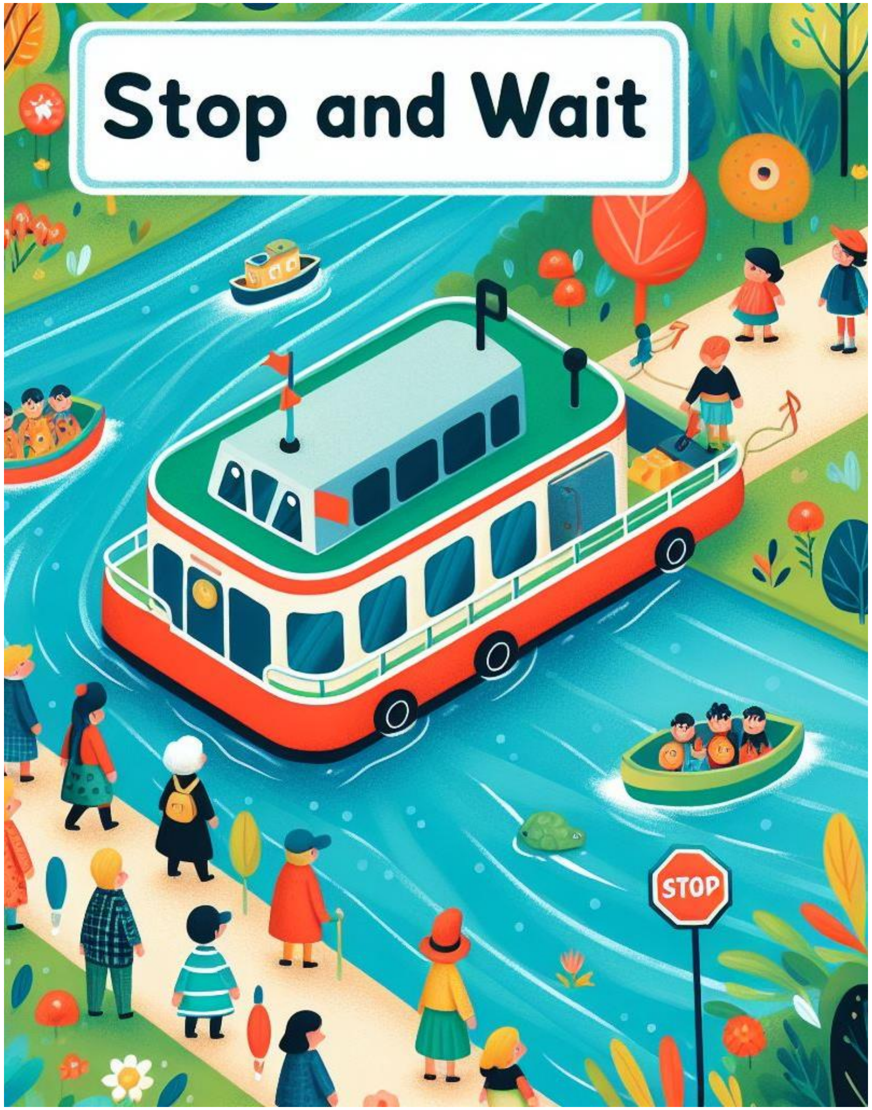
- ARQ
  - Stop and Wait
  - Sliding Window
- Ethernet
  - Topologies
  - Frame format
  - Media Access Control
  - Efficiency

- Source: Peterson and Davie 2.1-2.5, 2.6, Tanenbaum 4.3

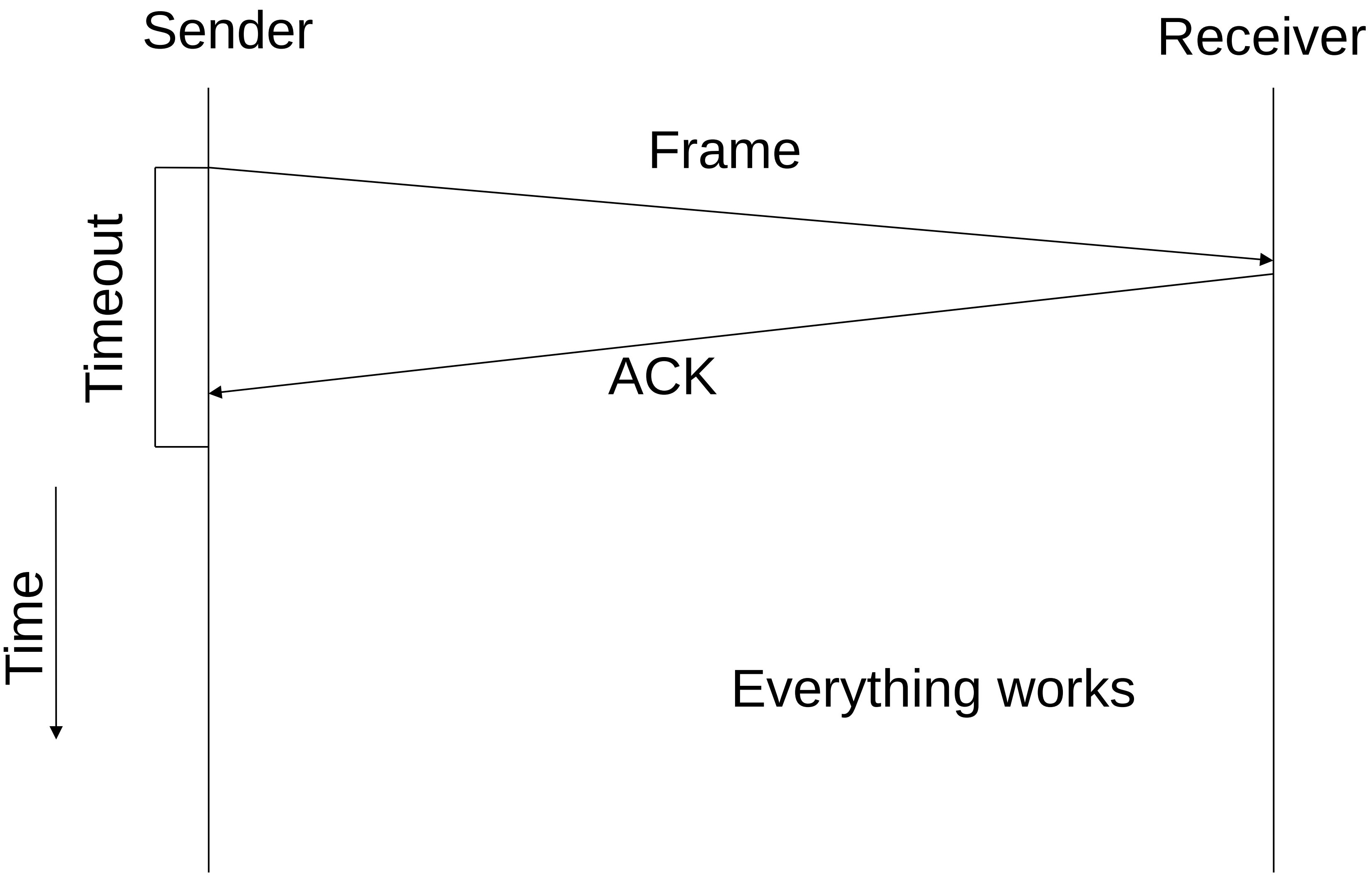
# Stop-and-Wait - Simplest scheme

1. After transmitting one frame, sender waits for an ACK

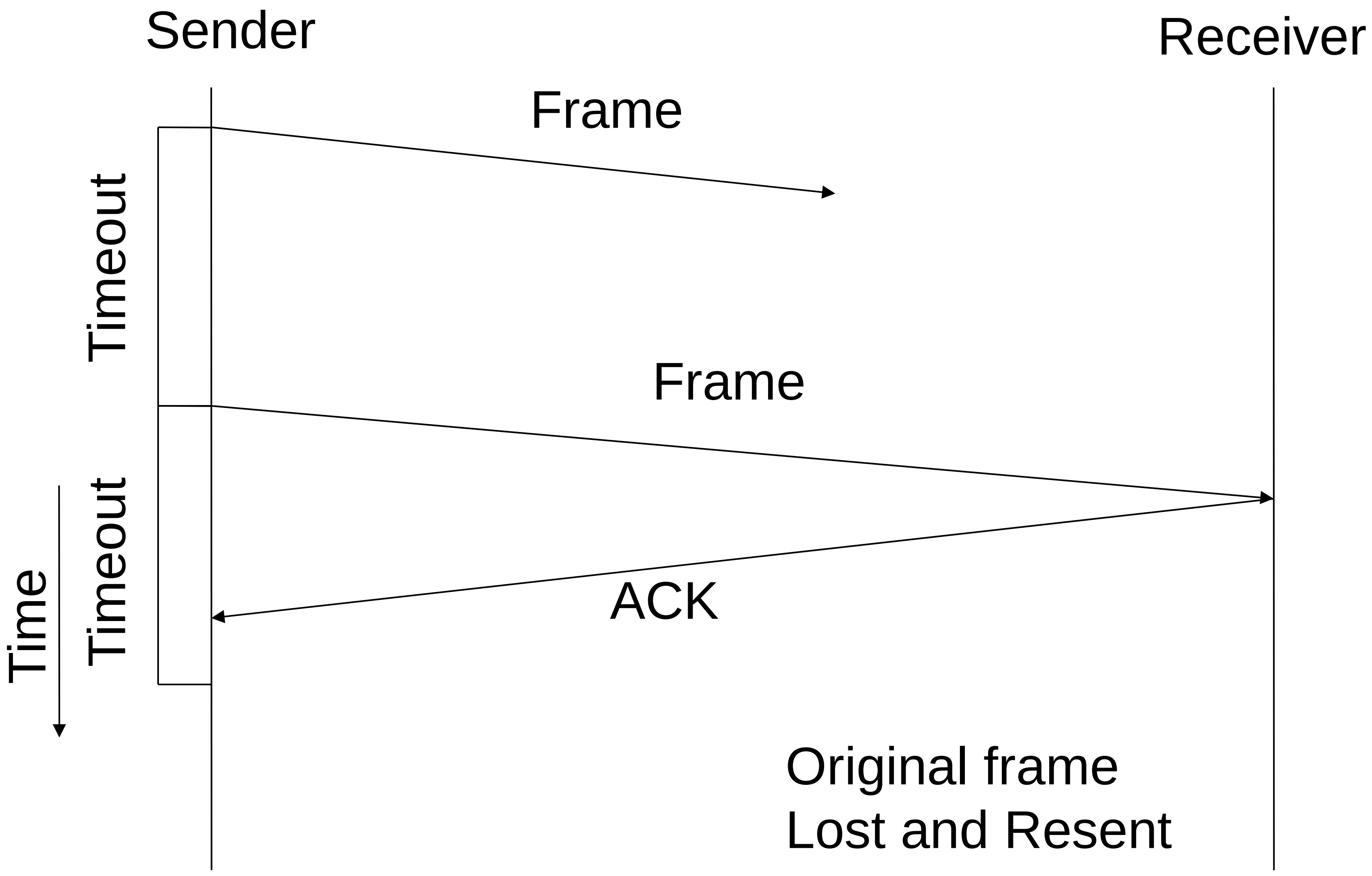
2. If the ACK doesn't arrive, sender retransmits



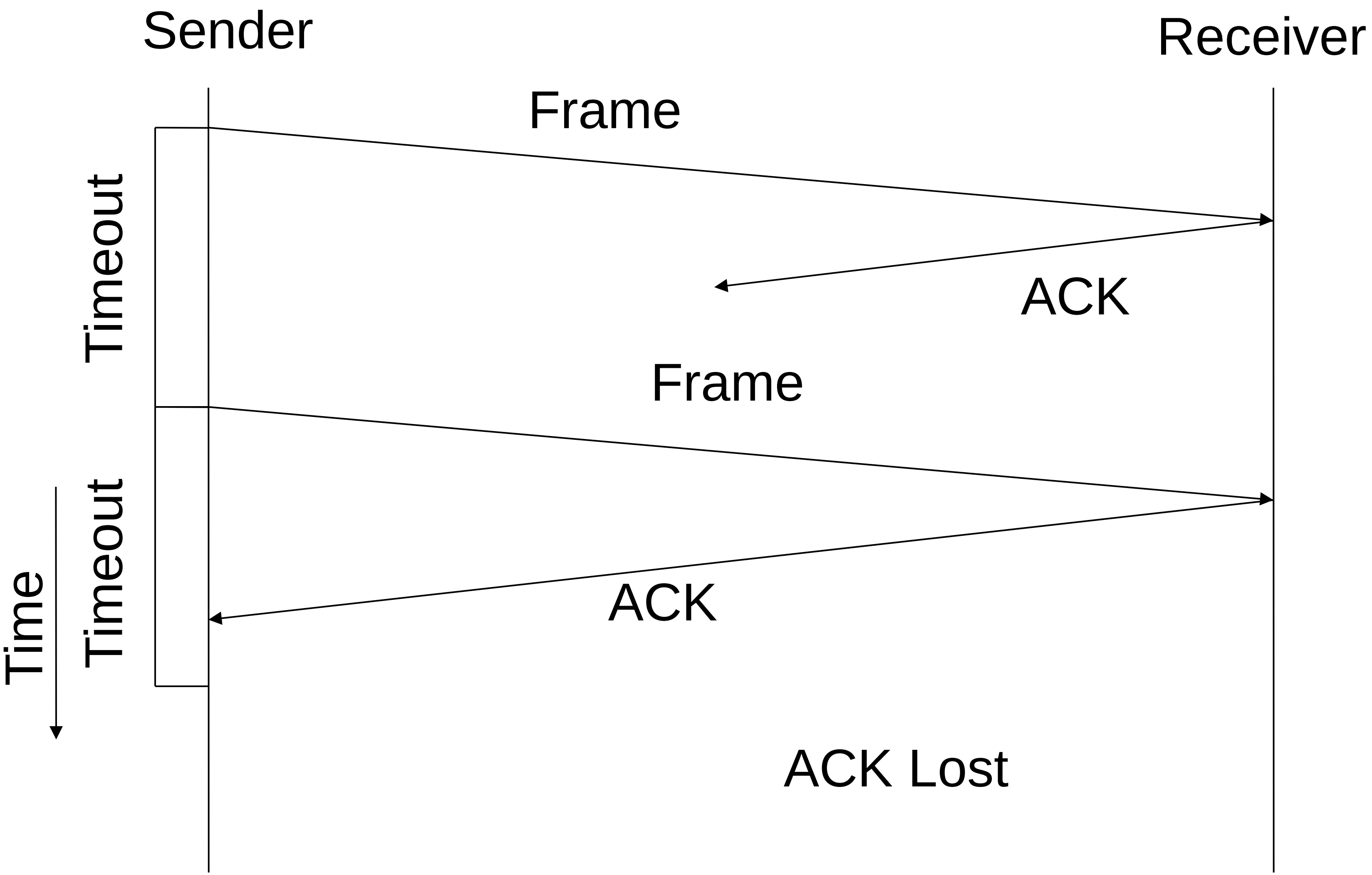
# Stop-and-Wait scenario 1



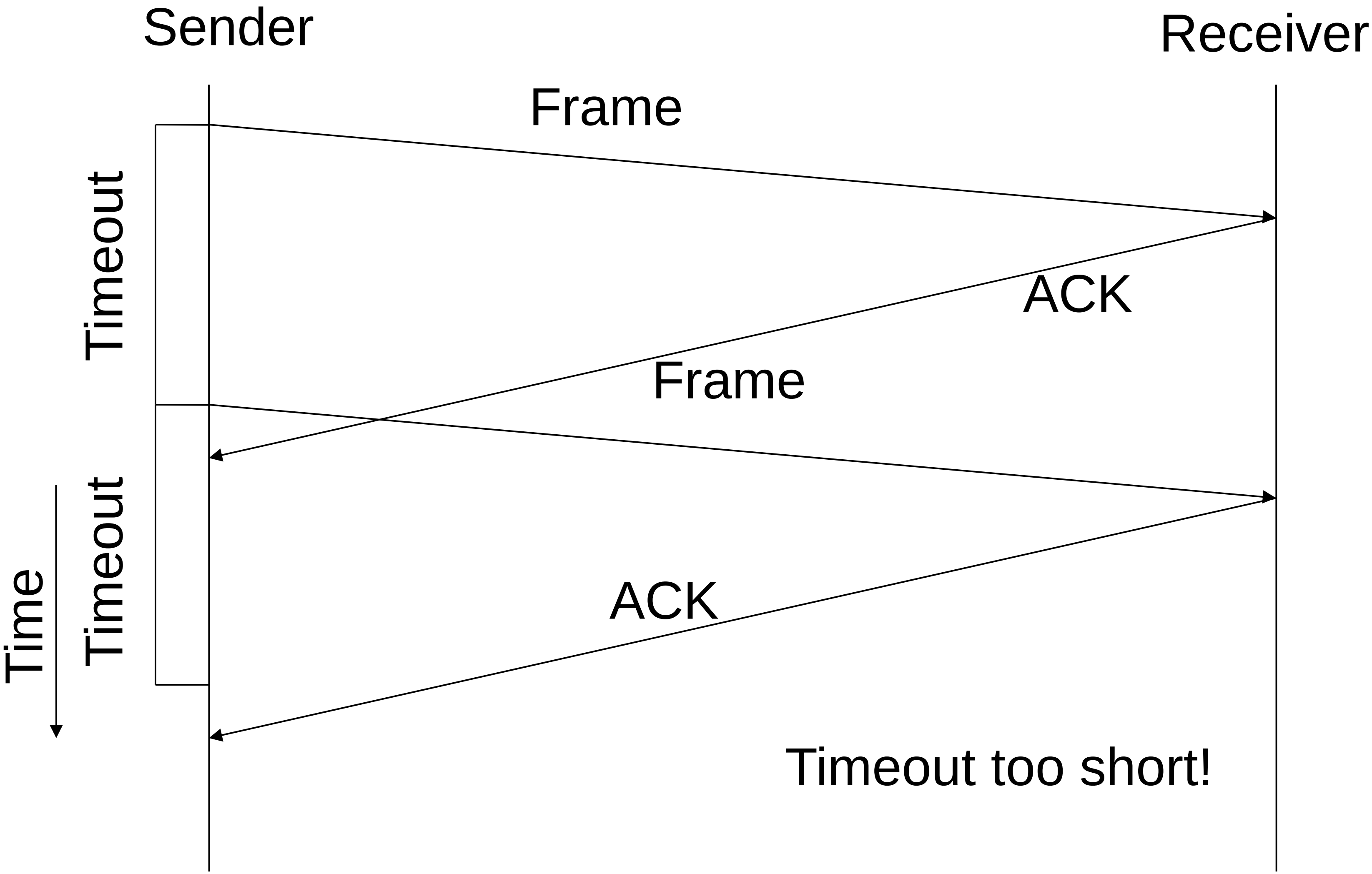
# Stop-and-Wait scenario 2



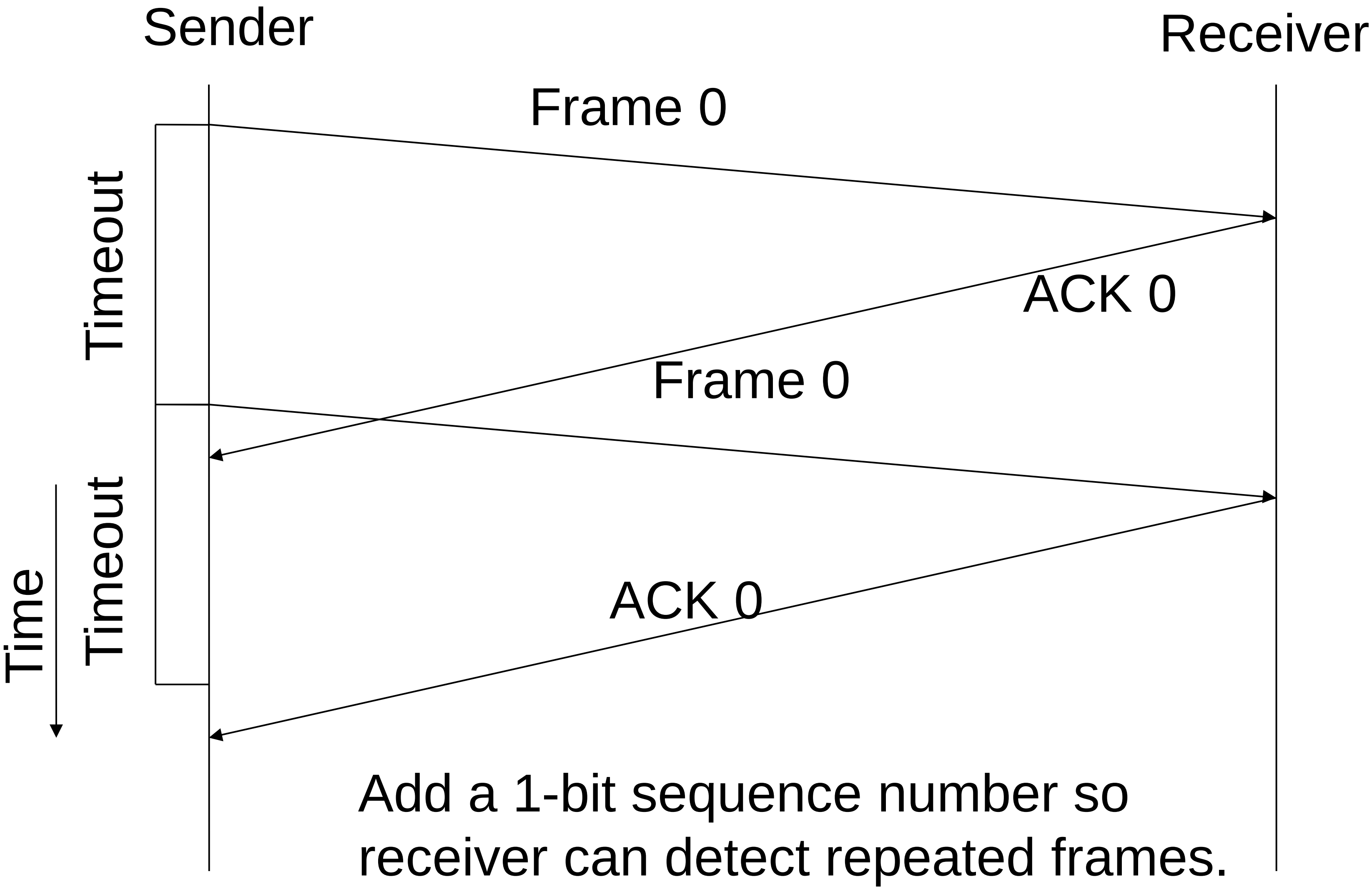
# Stop-and-Wait scenario 3



# Stop-and-Wait scenario 4



# Sequence numbers



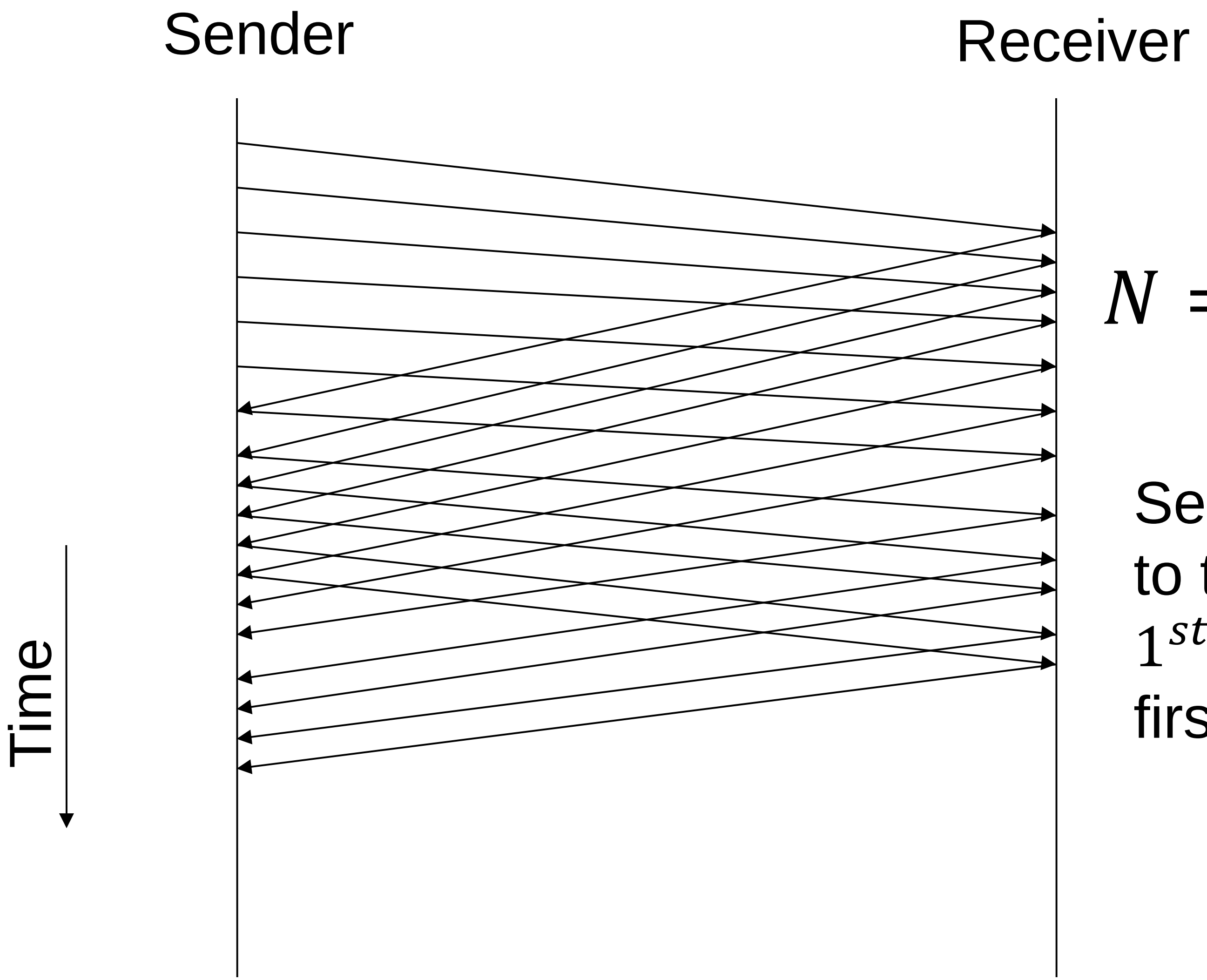

Add a 1-bit sequence number so receiver can detect repeated frames.

# Stop-and-Wait

- Inefficient use of link's capacity
- Sends 1 frame per RTT

- Example:
  - 10Mbps Link
  - 16ms RTT
  - Delay x Bandwidth product is about 20KB
  - Frame size of 1K yields about 5% link capacity

# More efficient solution




$$N = \frac{\frac{RTT}{2} \times BW}{frame\ size}$$

Sender ready to transmit $N+1^{st}$ frame just as first ACK arrives.

# So Far

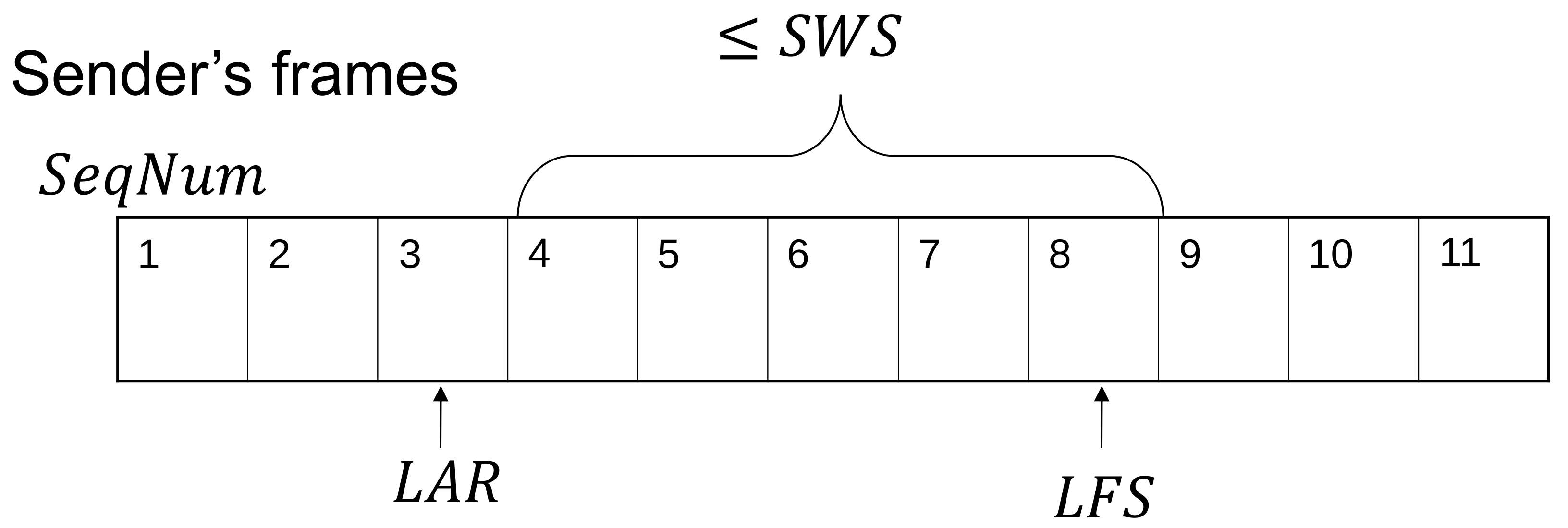
- ARQ
  - Stop and Wait
  - Sliding Window

# Sliding Window Algorithm

- Sender assigns a *sequence number* to each frame: $SeqNum$
  - For now, assume $SeqNum$ can grow infinitely

- Send Window Size ($SWS$)
  - Upper bound on # of unacknowledged frames sender will transmit
- Last ACK Received ($LAR$)
  - Sequence number of last ACK
- Last Frame Sent ($LFS$)

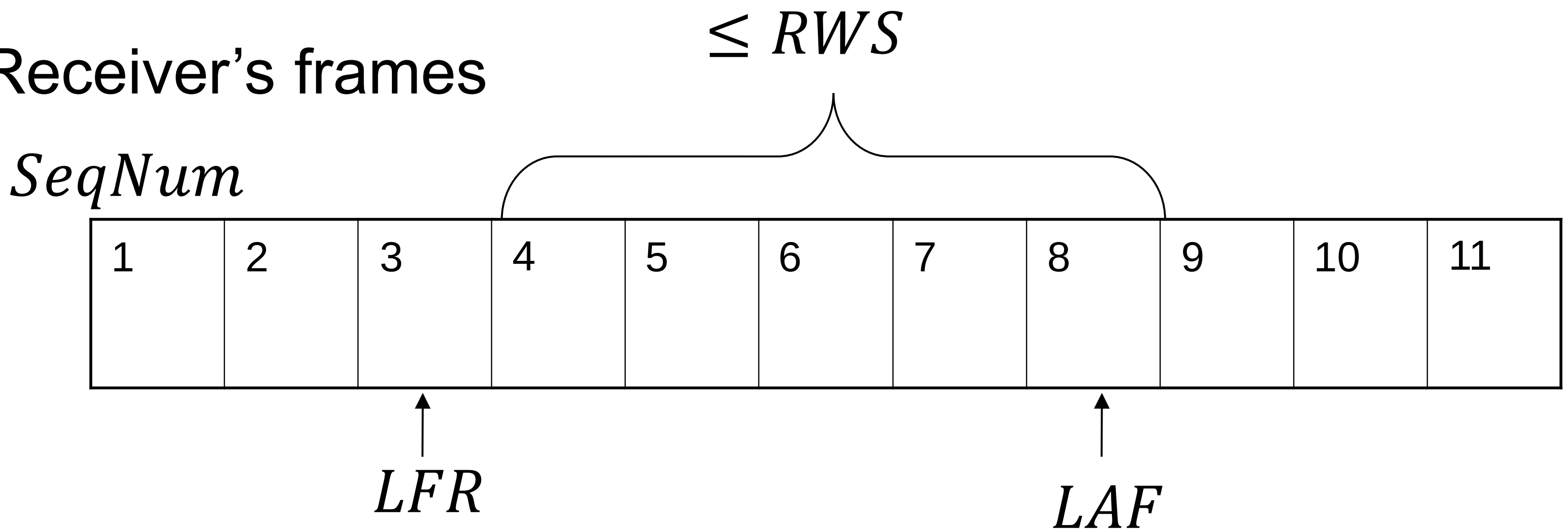
# Sender Invariant




- $LFS - LAR \leq SWS$
- Associates timeout with each frame sent
  - Retransmits if no ACK received before timeout
- When ACK arrives, increase $LAR$
  - Means another frame can be sent

# Receiver Invariant

Receiver's frames



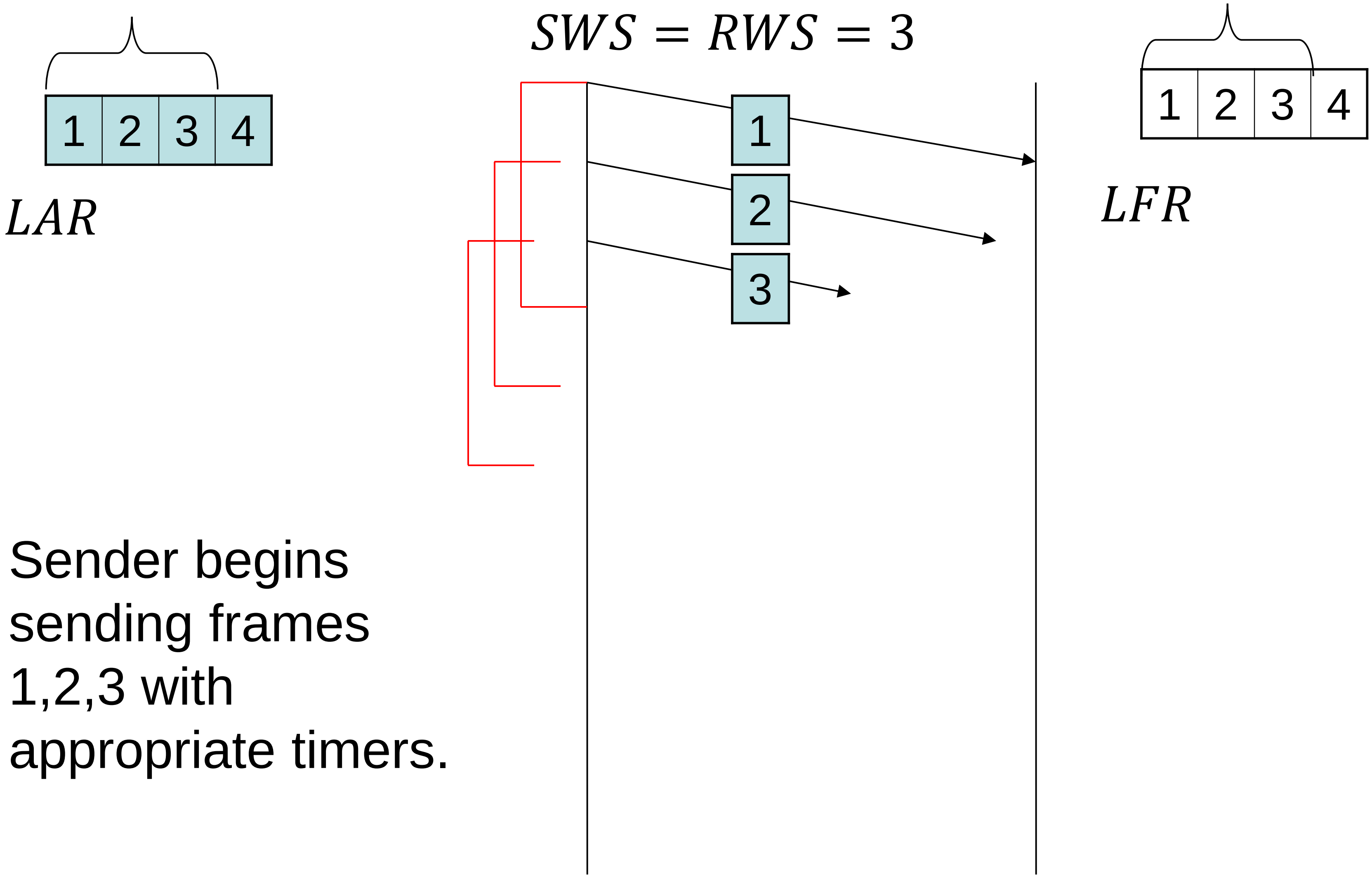

- Receive Window Size ($RWS$)
  - Number of out-of-order frames it will accept
- Largest Acceptable Frame ($LAF$)
  - Largest Frame Received ($LFR$)
- $LAF - LFR \leq RWS$

# Receiver Algorithm

- When packet numbered $SeqNum$ arrives
  - If ($SeqNum \leq LFR$) or ($SeqNum > LAF$) discard
  - Else accept the packet
- Define: $SeqNumToAck$
  - Largest unACK'ed sequence number such that all earlier frames have been accepted
- Receiver sends ACK($SeqNumToAck$)
- $LFR = SeqNumToAck$
- $LAF = LFR + RWS$

# Example Sliding Window Protocol

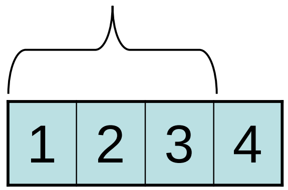
$SWS = RWS = 3$



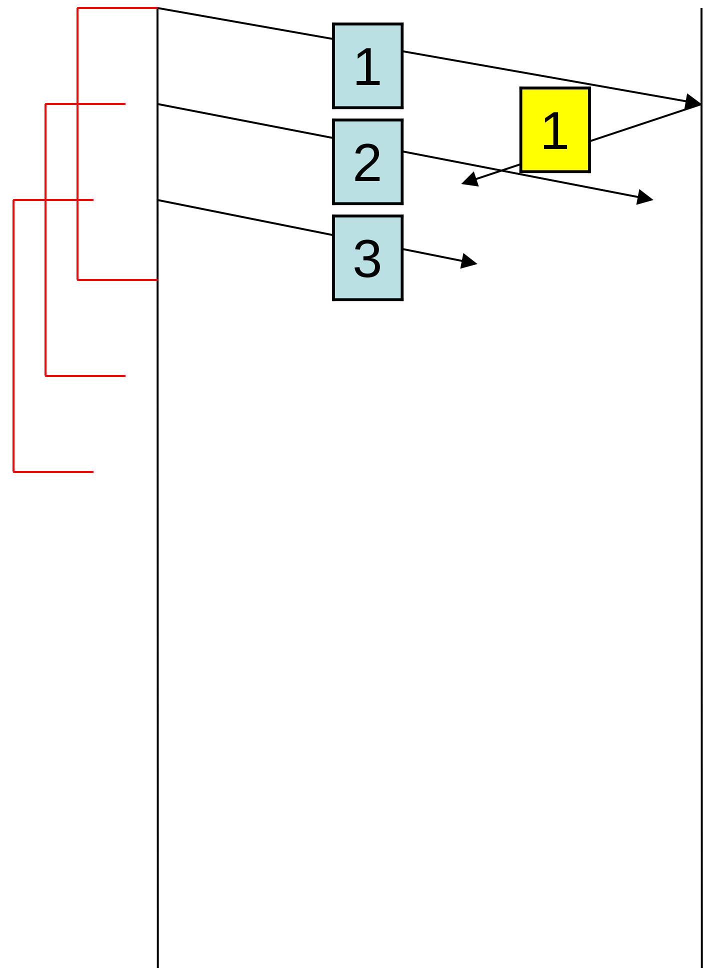

Sender begins sending frames 1,2,3 with appropriate timers.

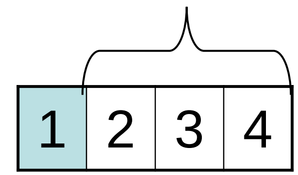
# Example Sliding Window Protocol

$SWS = RWS = 3$




$LAR$




$LFR$

Receiver gets frame 1

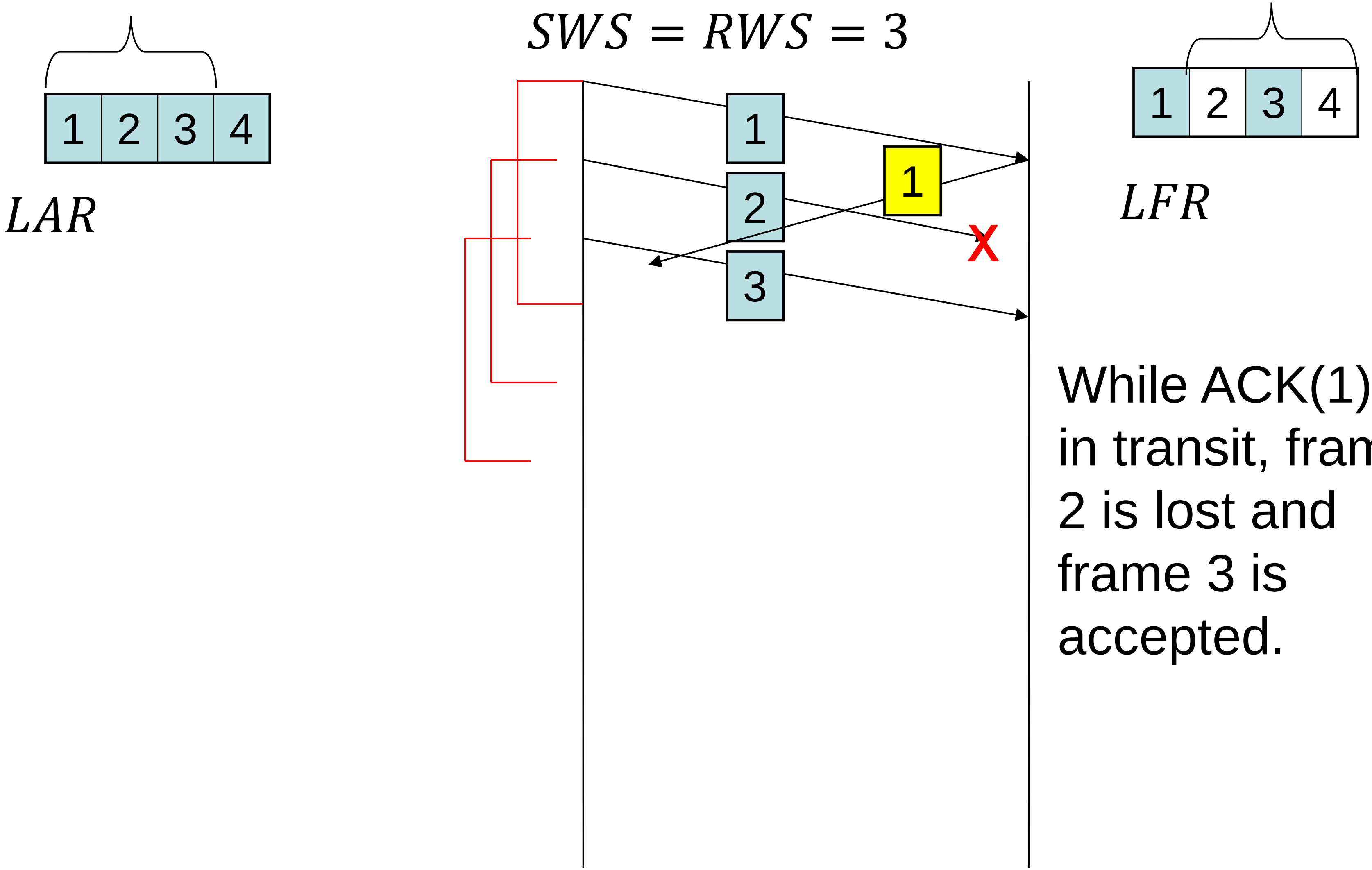
$SeqNumToAck = 1$

Receiver sends ACK(1)

$LFR = 1$

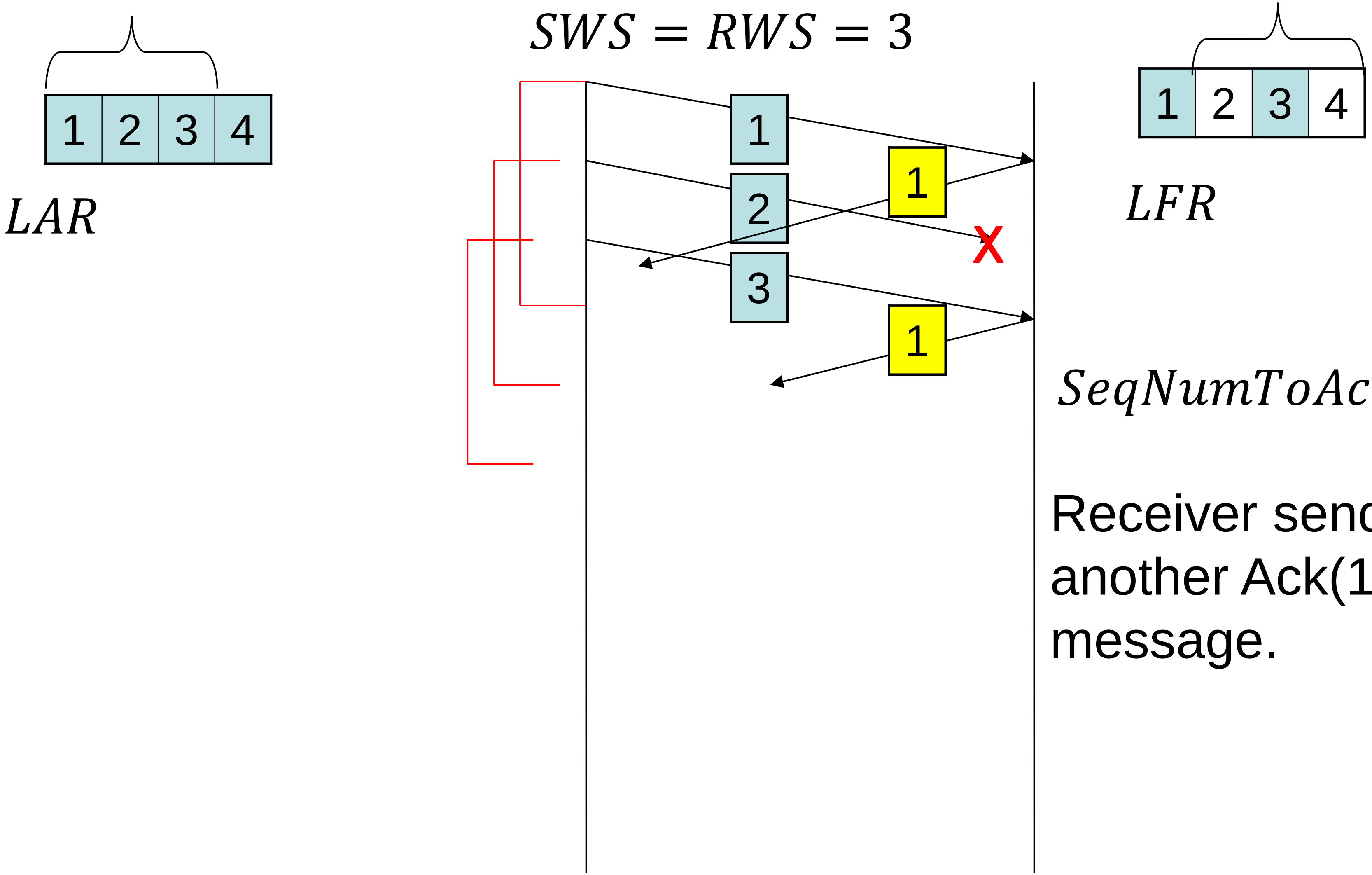
$LAF = LFR + RWS$

# Example Sliding Window Protocol




While ACK(1) is in transit, frame 2 is lost and frame 3 is accepted.

# Example Sliding Window Protocol

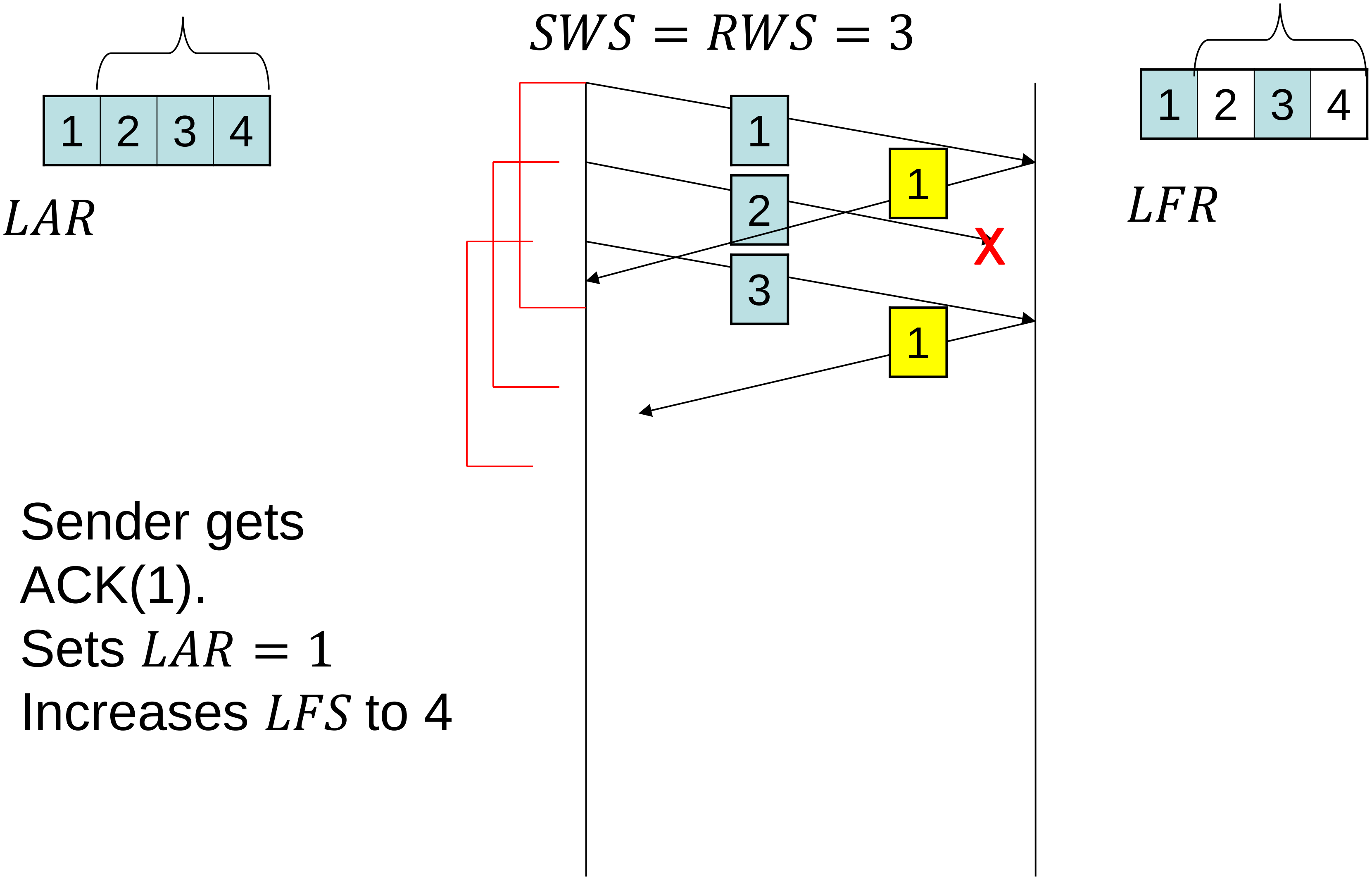



$SeqNumToAck = 1$

Receiver sends another Ack(1) message.

# Example Sliding Window Protocol

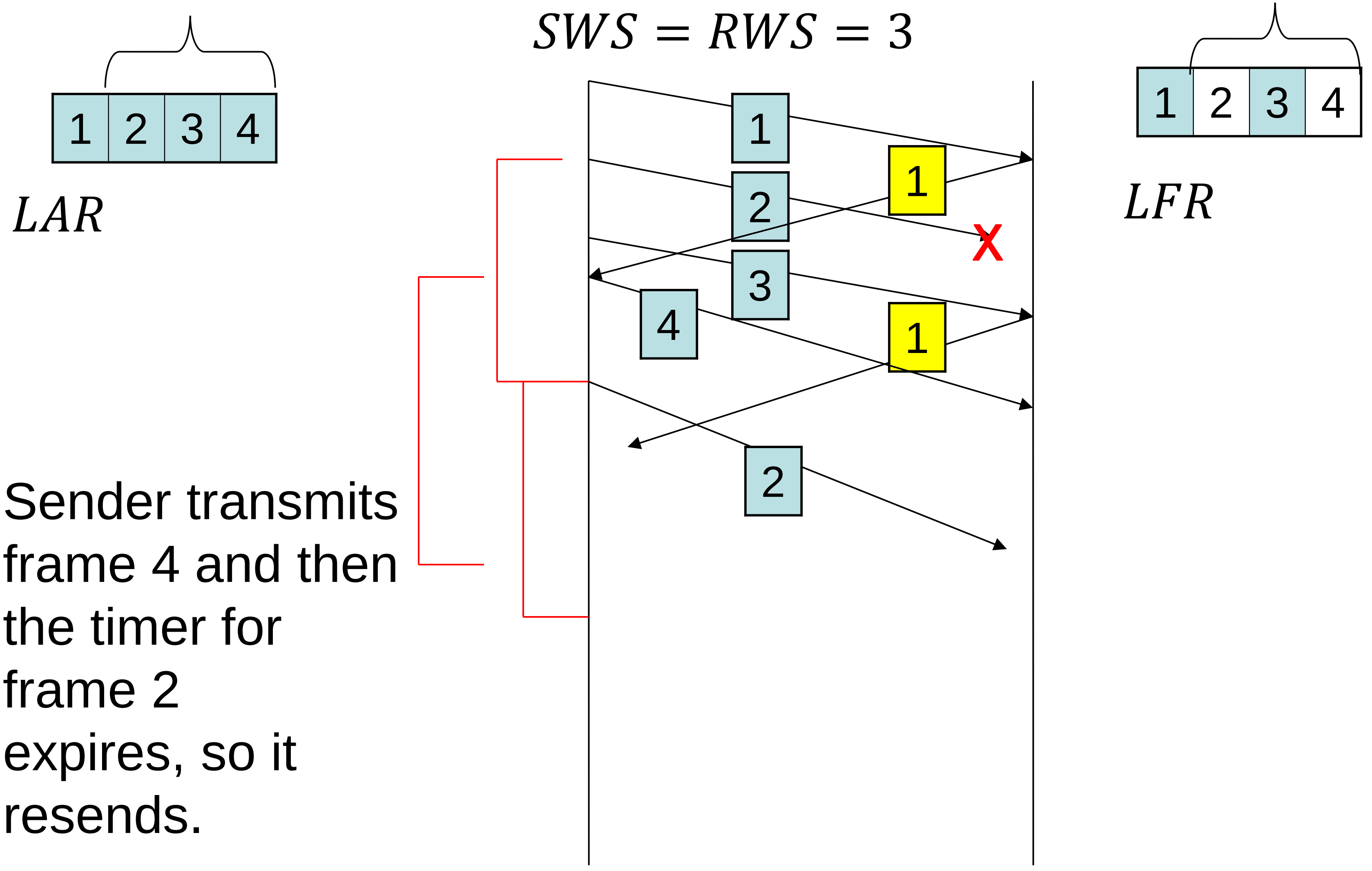
$SWS = RWS = 3$




Sender gets ACK(1).
Sets $LAR = 1$
Increases $LFS$ to 4

# Example Sliding Window Protocol

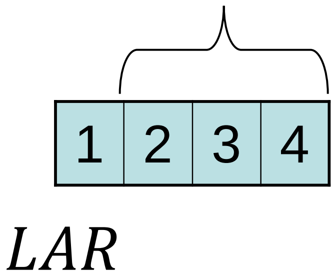
$SWS = RWS = 3$

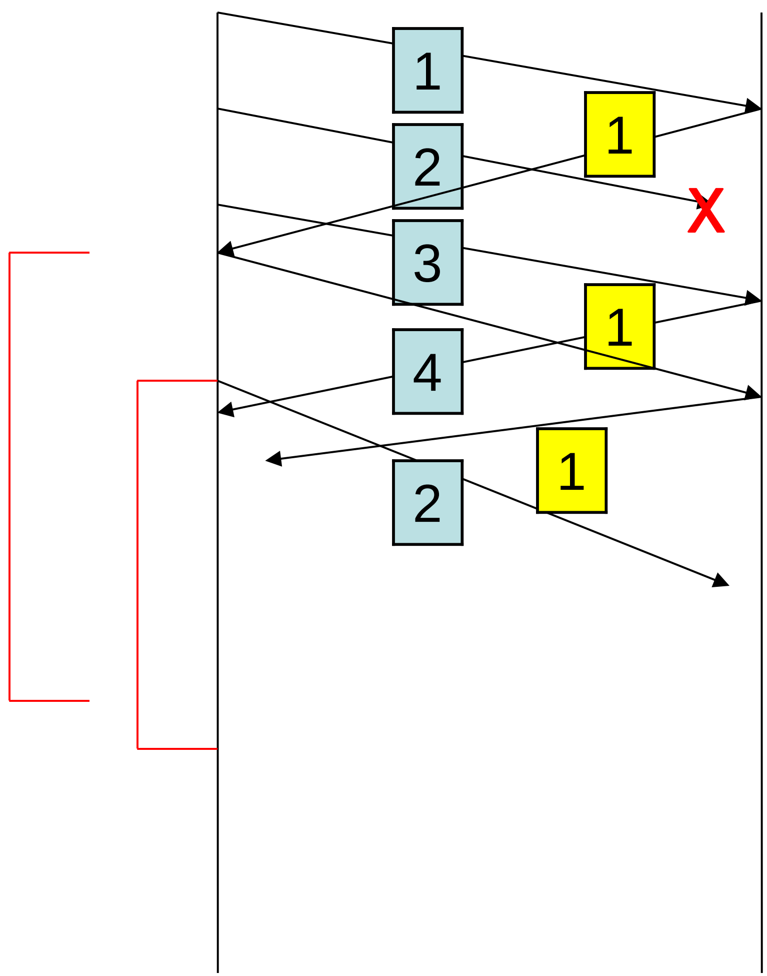


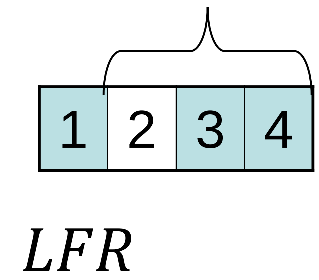

Sender transmits frame 4 and then the timer for frame 2 expires, so it resends.

# Example Sliding Window Protocol

SWS = RWS = 3



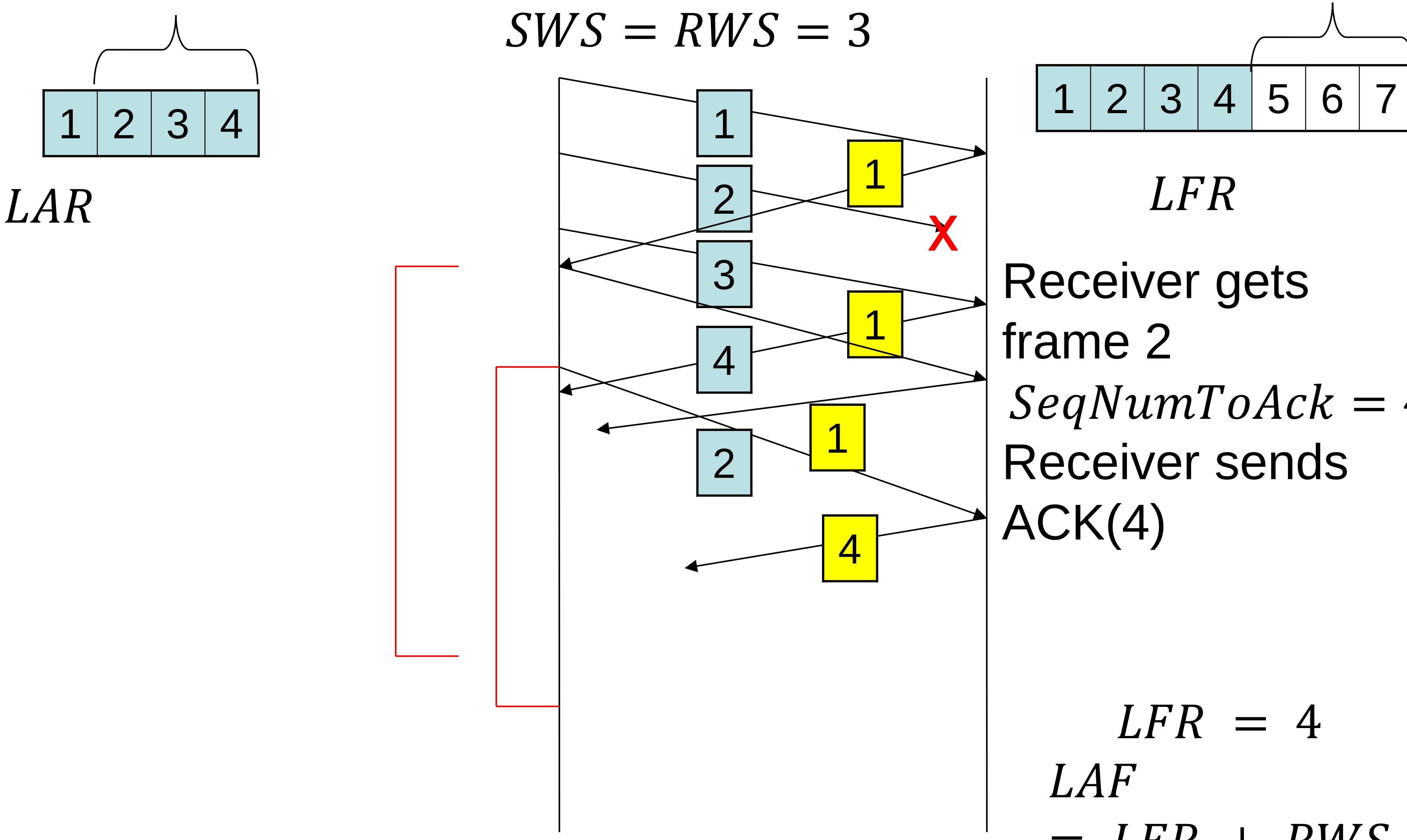




Sender gets ACK(1) again—ignores it.

Receiver gets frame 4
$SeqNumToAck = 1$
Receiver sends ACK(1)

# Example Sliding Window Protocol

$SWS = RWS = 3$




Receiver gets frame 2

$SeqNumToAck = 4$

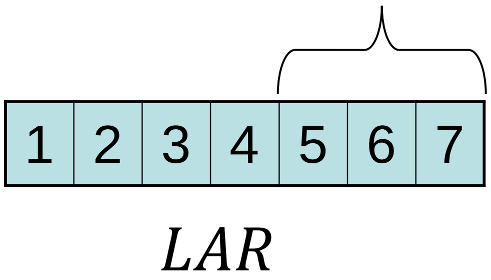
Receiver sends ACK(4)

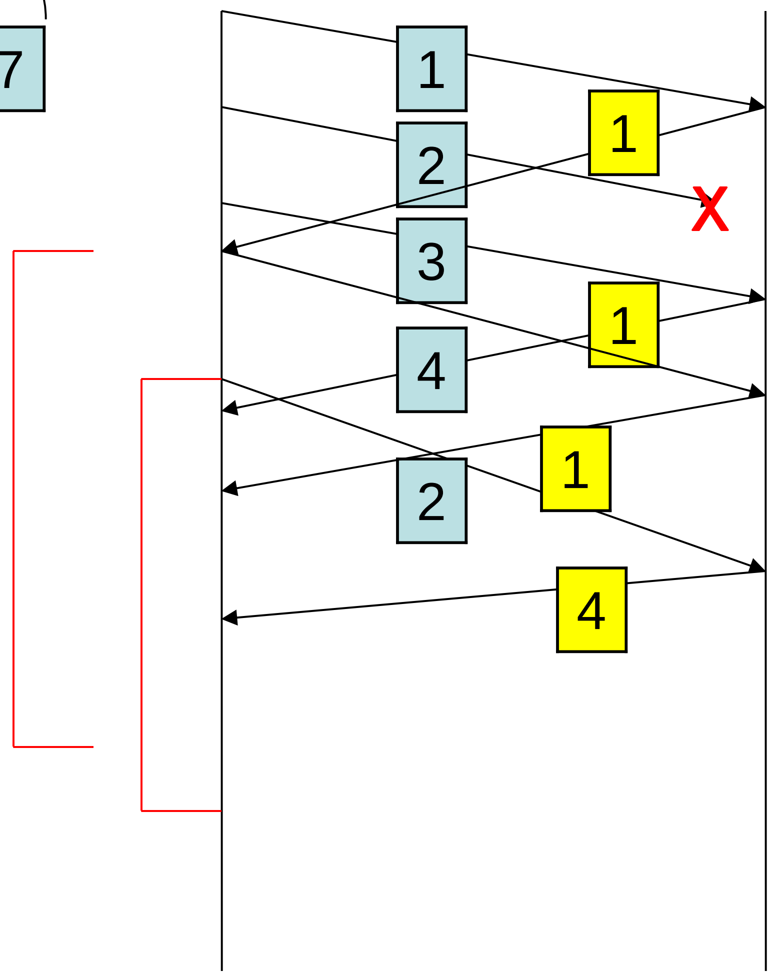
$LFR = 4$

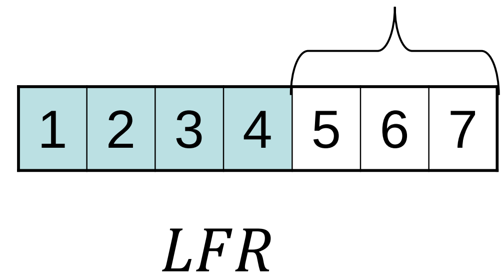
$LAF = LFR + RWS$

# Example Sliding Window Protocol

$SWS = RWS = 3$




Sender gets ACK(1) again—ignores it.

Sender gets ACK(4)
Sets $LAR = 4$
Increases $LFS$

# Variants on Sliding Window

## Receiver doesn't transmit redundant ACKs

- Just ignore out of order arrivals

## Receiver transmits *selective ACKS*

- ACK indicates exactly which frames have been accepted

# What size for the window?

- If $RTT \times Bandwidth$ product known, ideal is:

$$SWS = \frac{RTT}{2} \times \frac{Bandwidth}{Framesize}$$

- Common receive window size settings:

| $RWS = 1$ | $RWS = SWS$ | Note: $RWS > SWS$ is not sensible |
| --- | --- | --- |
| • No buffering of out-of-order frames | • Buffer as many as can be in flight | |

# Finite Sequence Numbers

We've assumed infinite sequence numbers so far.

Real packets have finite size "Sequence Number" field

What do we do?


Image source: reddit.com

# What's a sufficient SeqNum Field size?

## Principle: Re-use sequence numbers

- 8-bit example
- They wrap: 0,1, 2, …, 254, 255, 0,1, 2, …, 254, 255, 0,1, 2, …

## Recall:

- For Stop-and-Wait we need 2 sequence values (0/1)
- 1 bit of space

## What about for Sliding Window with X packets?

## Suppose $SWS = RWS$

- How many sequence numbers should there be?
- Is $SWS + 1$ sufficient?

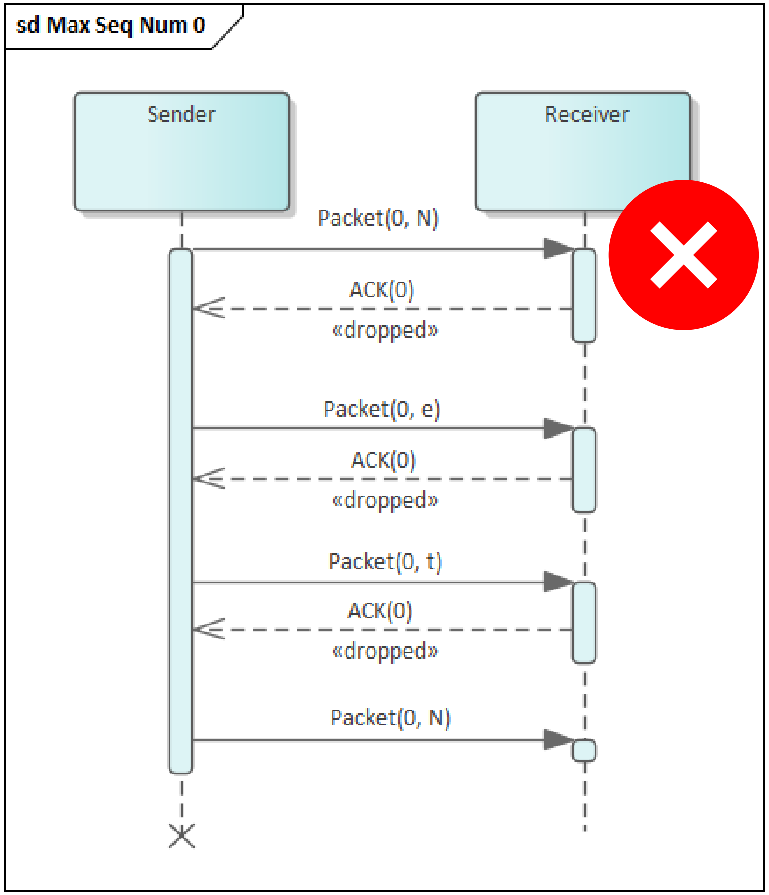
# Sufficient MaxSeqNum

- Frame $i$'s sequence num is $i \ mod \ MaxSeqNum$
- Assuming $SWS = RWS$
- $SWS < \frac{MaxSeqNum+1}{2}$
- $MaxSeqNum > (2 \times SWS) - 1$
- Why?
  - Consider case where all the ACKS are lost.
  - Suppose $SWS = RWS = 3$
  - $MaxSeqNum \geq 5$ since (0,1,2,3,4) are insufficient

# Sequence Number Worst Case

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | e | t | w | o | r | k | s |

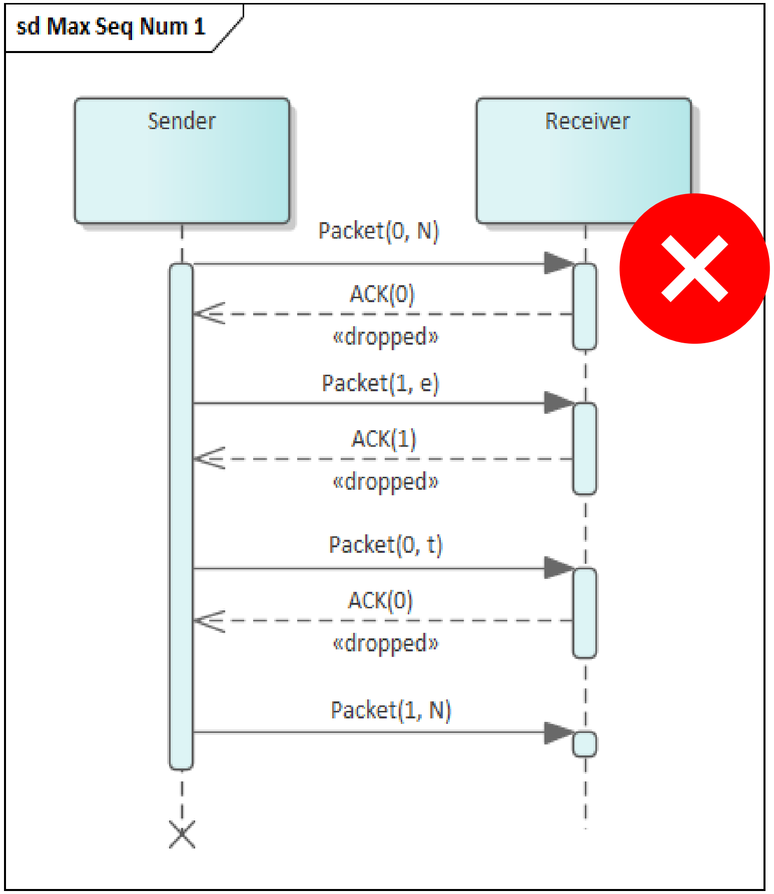
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |



# Sequence Number Worst Case

| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| N | e | t | w | o | r | k | s |

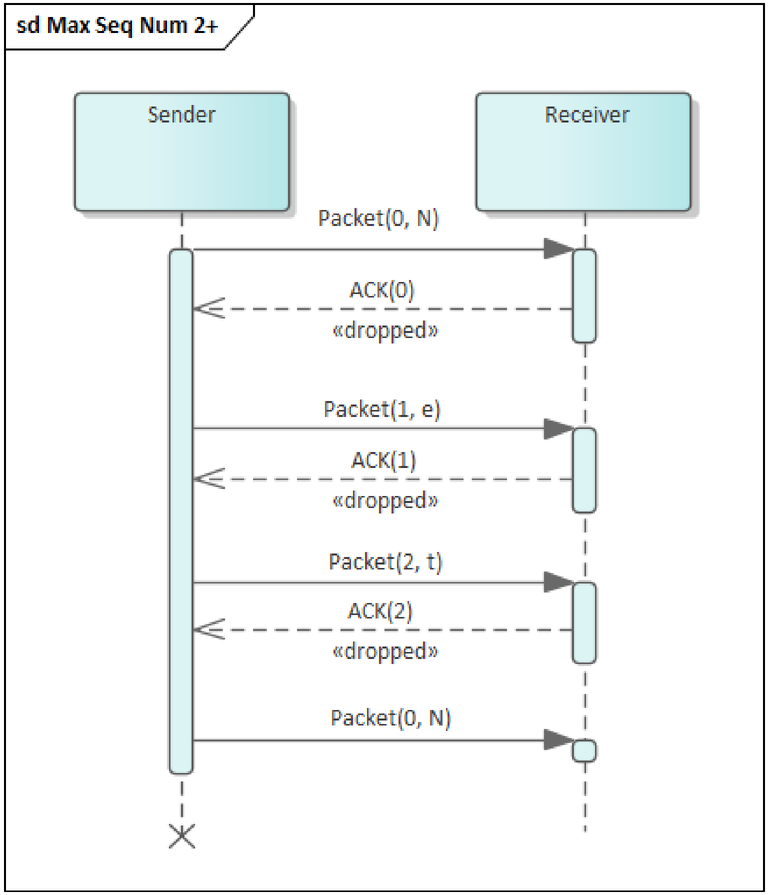
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |



# Sequence Number Worst Case

| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| N | e | t | w | o | r | k | s |

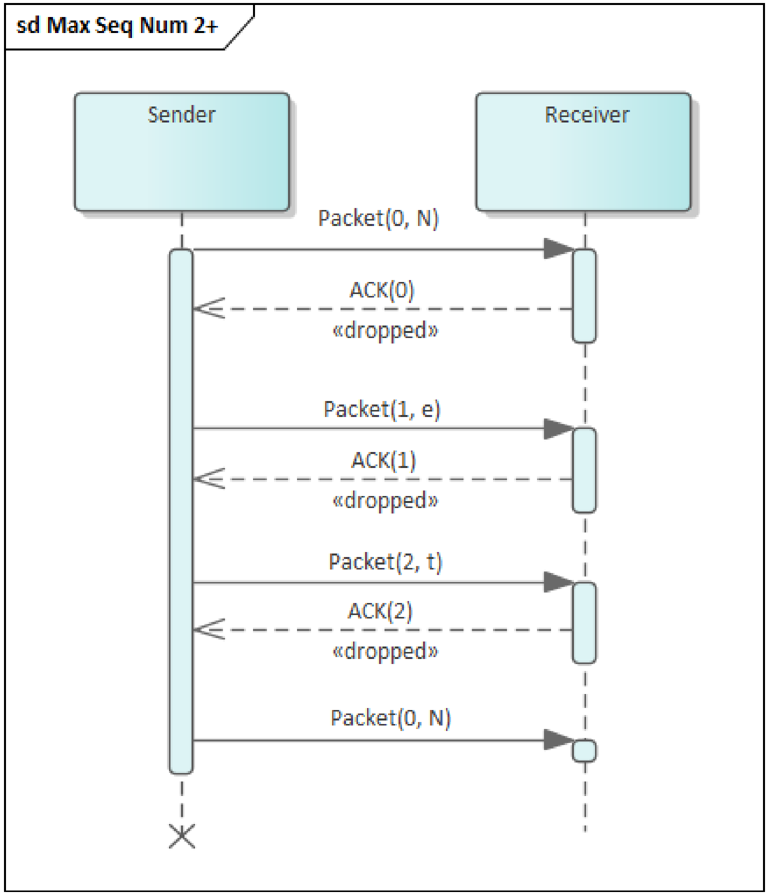
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |



# Sequence Number Worst Case

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | e | t | w | o | r | k | s |

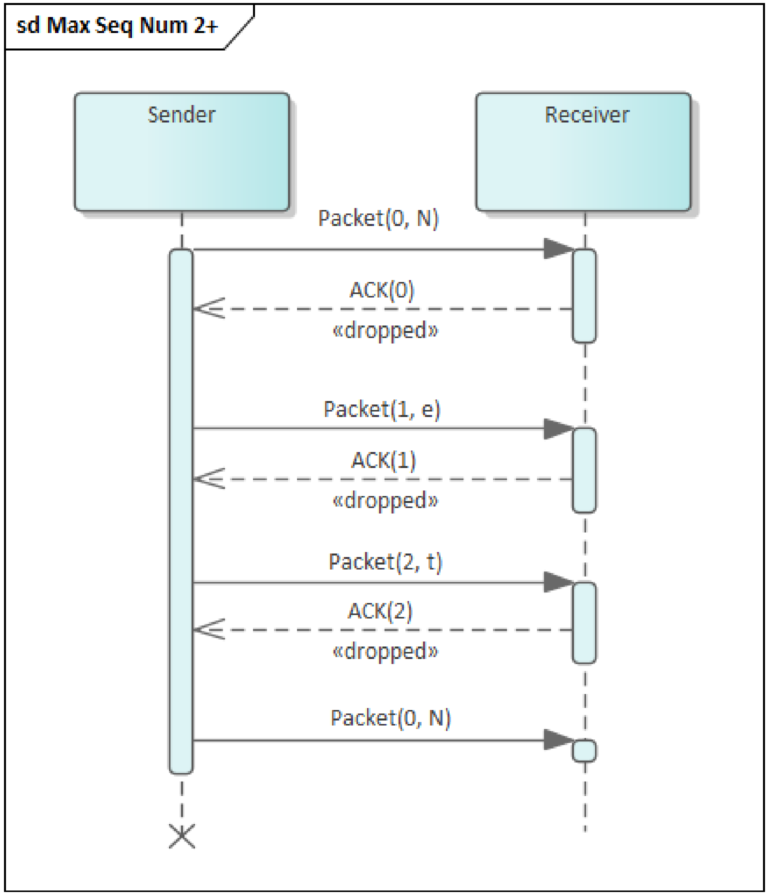
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | | | | | | | |



# Sequence Number Worst Case

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | e | t | w | o | r | k | s |

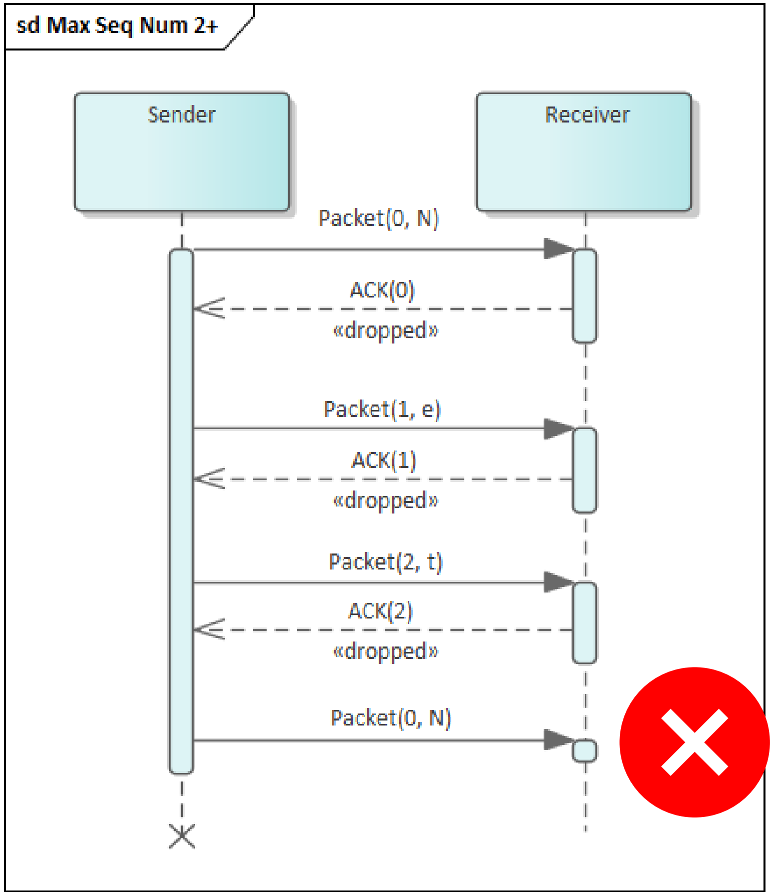
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | e | | | | | | |



# Sequence Number Worst Case

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | e | t | w | o | r | k | s |

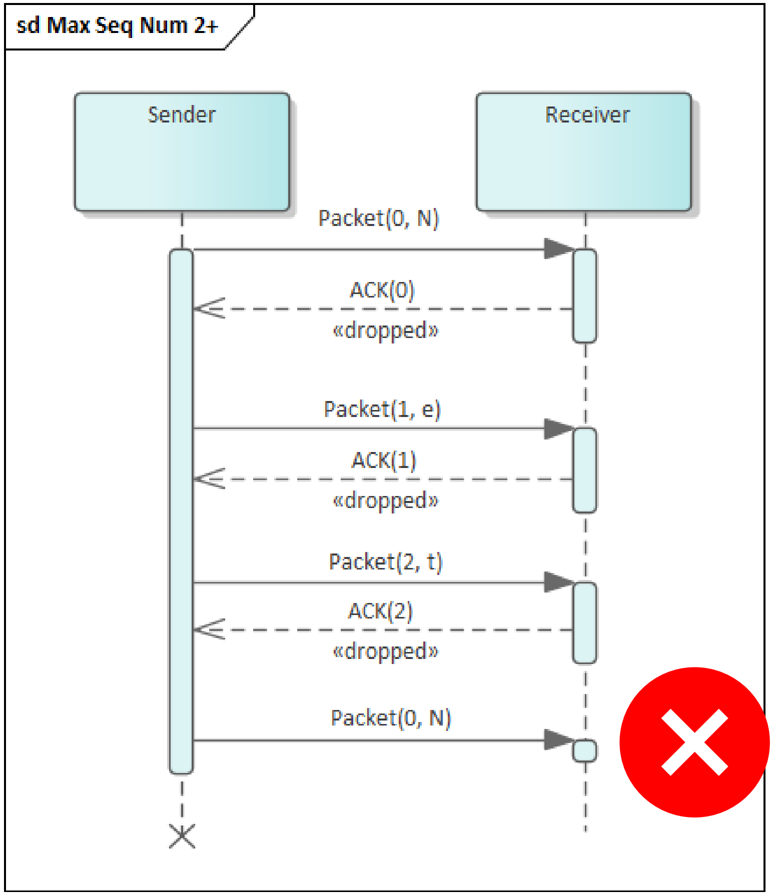
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 |
| N | e | t | | | | | |



# Sequence Number Worst Case

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| N | e | t | w | o | r | k | s |

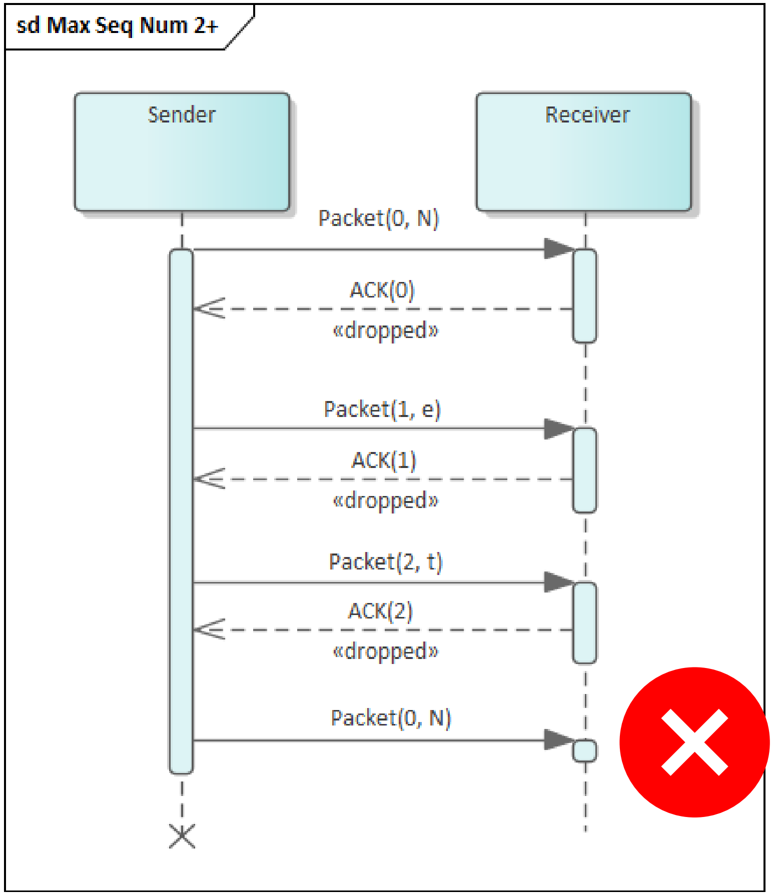
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| N | e | t | | | | | |



# Sequence Number Worst Case

| 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 |
|---|---|---|---|---|---|---|---|
| N | e | t | w | o | r | k | s |

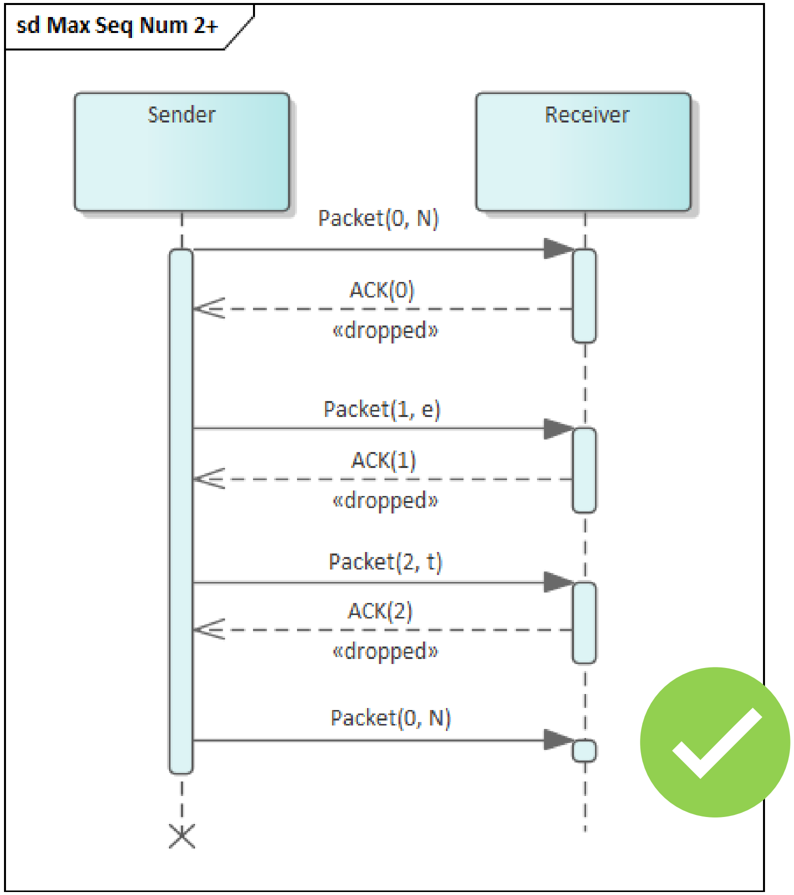
| 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 |
|---|---|---|---|---|---|---|---|
| N | e | t | | | | | |



# Sequence Number Worst Case

| 0 | 1 | 2 | 3 | 4 | 5 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| N | e | t | w | o | r | k | s |

| 0 | 1 | 2 | 3 | 4 | 5 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| N | e | t | | | | | |



# We're doing this backward

## Max Sequence Number determined by the protocol design

- Fixed field size

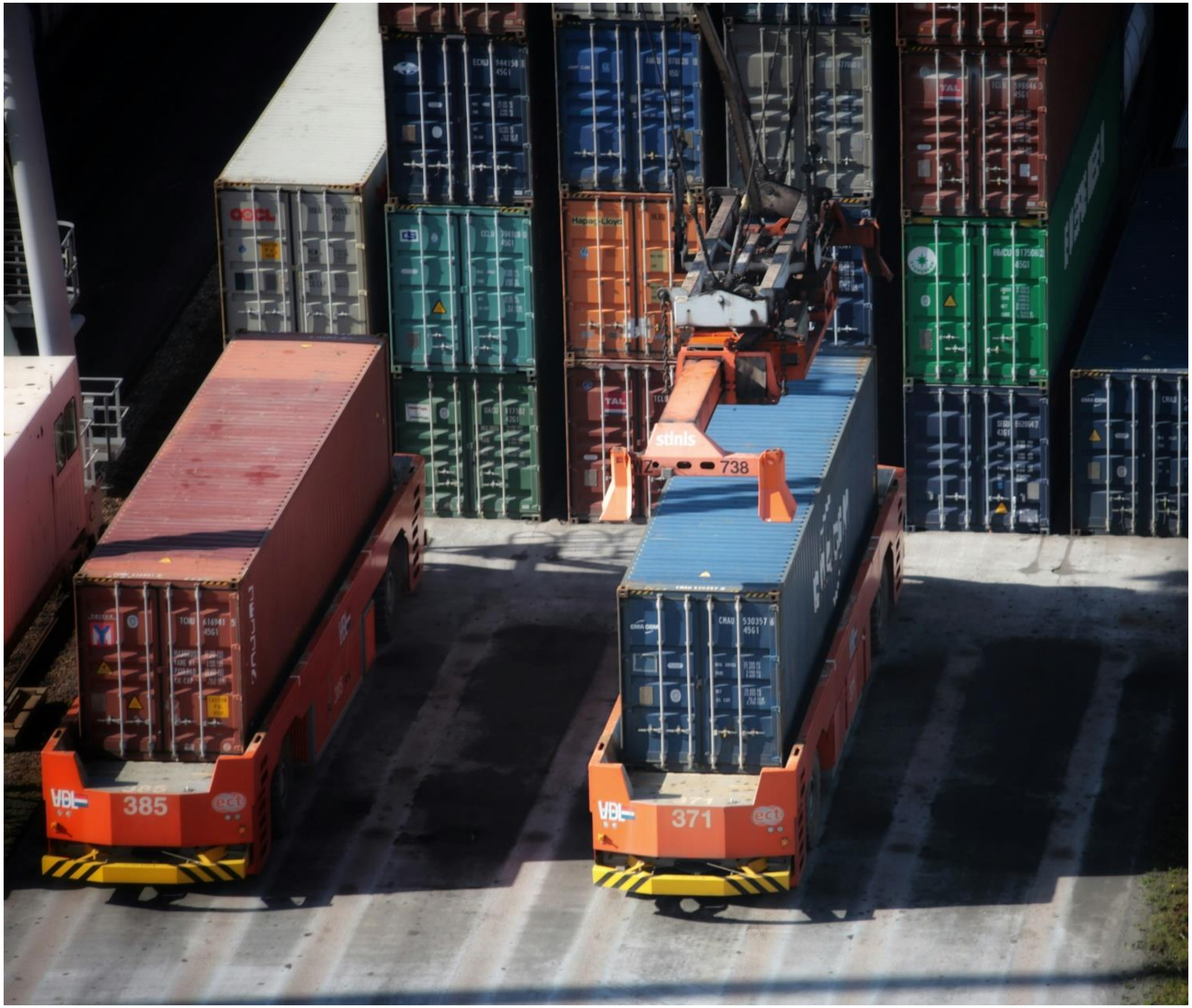
## Must select SWS and RWS accordingly

- $SWS \approx 0.5 * MaxSeqNum$
- $\mathrm{R}WS \approx 0.5 * MaxSeqNum$
- $\mathrm{R}WS \leq SWS$

## Sequence numbers wrap fast

- Imagine a 100Gbps connection
- 12.5 GB per second
- 32 bit sequence number field ~4 billion possibilities
- Number wraps ~3 times per second

Photo by Bernd 📷 Dittrich on Unsplash



# Roles of Sliding Window Algorithm

## Reliable delivery

- It provides an efficient retransmission protocol for dealing with errors

## In-order delivery

- The receiver buffers frames and delivers them in sequence number order

## Flow control

- It sends ACKs back to give hints to sender
- More sophisticated version could give # of frames the receiver has room for → throttles the sender

# Sliding window in practice

## TCP (Transmission Control Protocol)

Uses sliding window algorithm

More complex because it's used in internet – not over a direct link

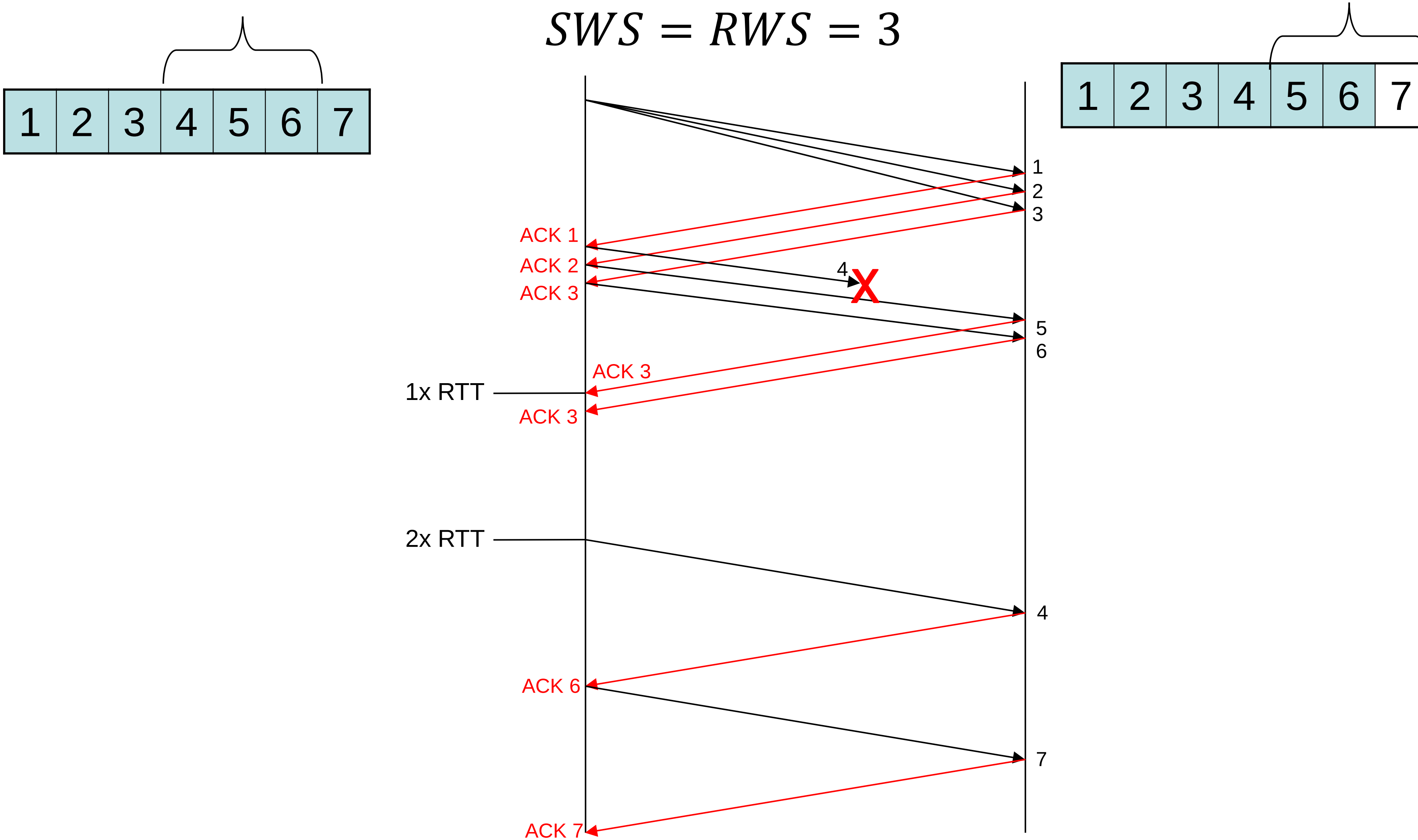
$Bandwidth \times delay$ not known

Dynamically changes timeouts

Larger buffers for in-order delivery

# Example: SWS=RWS=3, 4 drops

$$SWS = RWS = 3$$



# Conclusion

- ARQ
  - Stop and Wait
  - Sliding Window