CSE331: Introduction to Networks and Security

Lecture 9

Fall 2006

Announcements

- Project 1 Due TODAY
- HW 1 Due on Friday

- Midterm I will be held next Friday, Oct. 6th.
  - Will cover all course material up to next Weds.

Today: Reliable Transmission

- Now we can detect errors…
  - CRC
  - Checksum
- What do we do when we find one?

- Corrupt frames/packets must be discarded.
- Need to recover them.

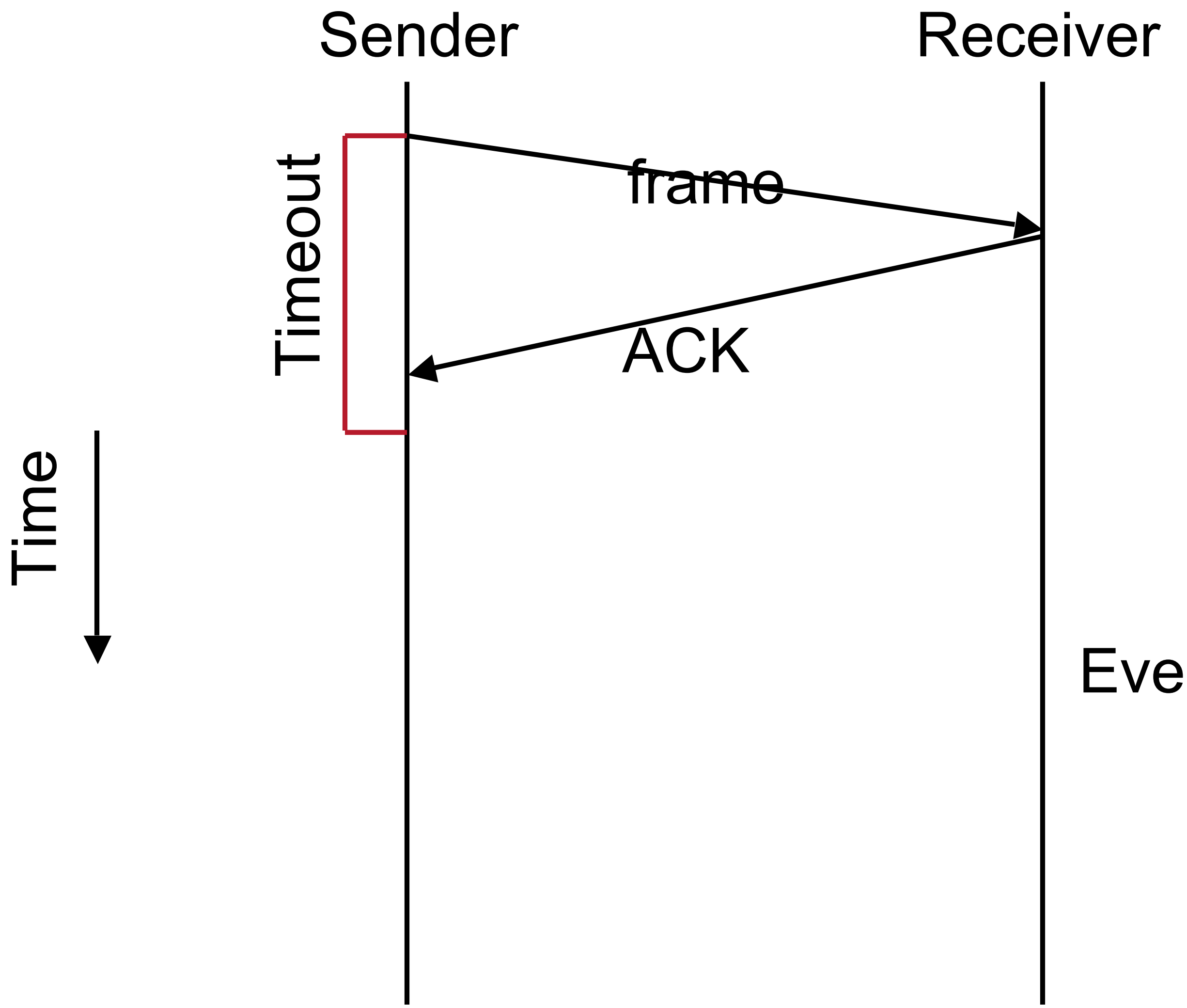
Fundamental mechanisms

- *Acknowledgments (ACK)*
  - Small control frame/packet (little data)
  - When sender gets an ACK, recipient has successfully gotten a frame
- *Timeouts*
  - If sender doesn't get an ACK after "reasonable" time it retransmits the original
- General strategy called Automatic Repeat Request (ARQ)

Stop-and-Wait

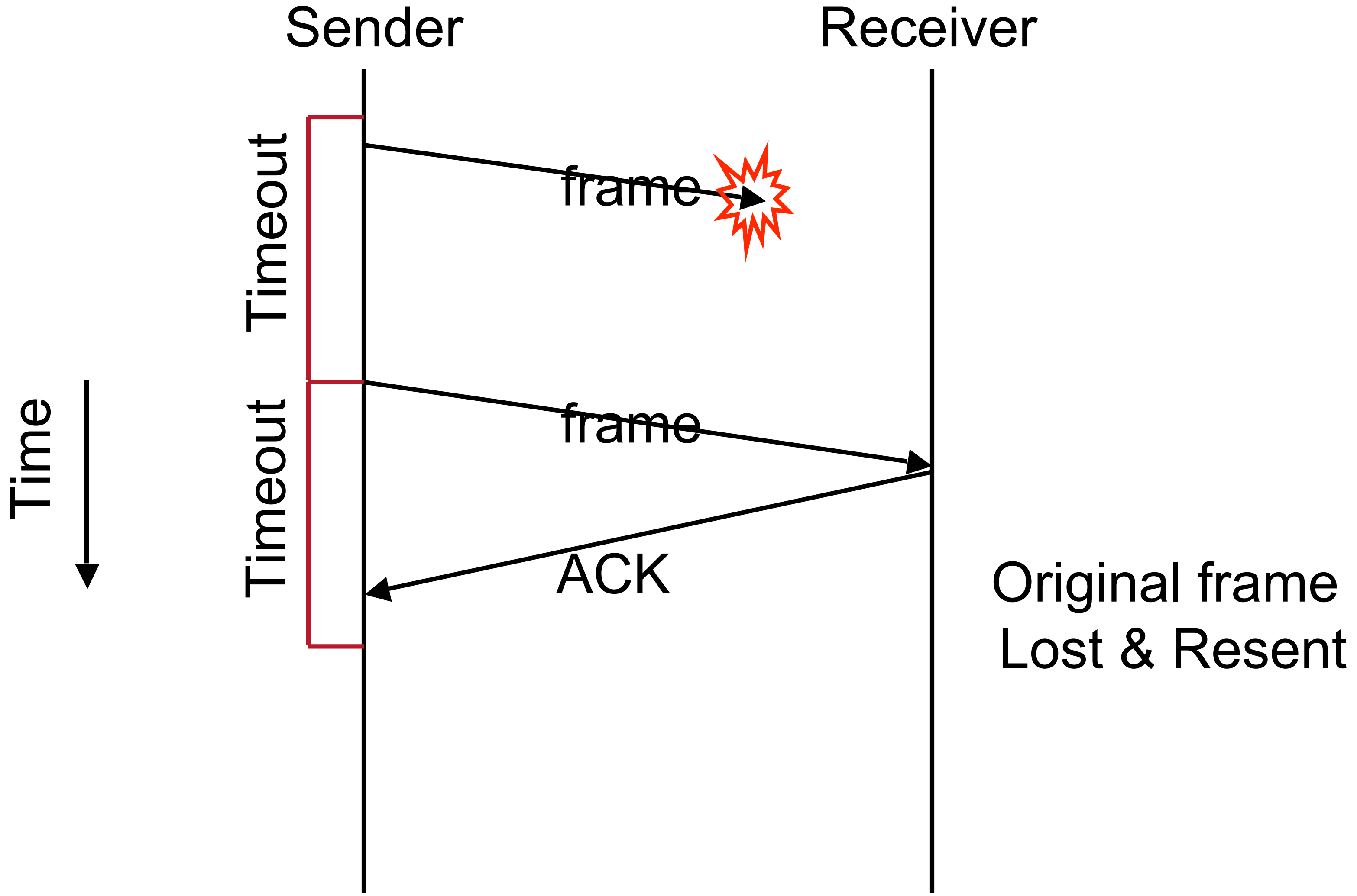
- Simplest scheme
  - After transmitting one frame, sender waits for an ACK
  - If the ACK doesn't arrive, sender retransmits

Stop-and-Wait scenario 1



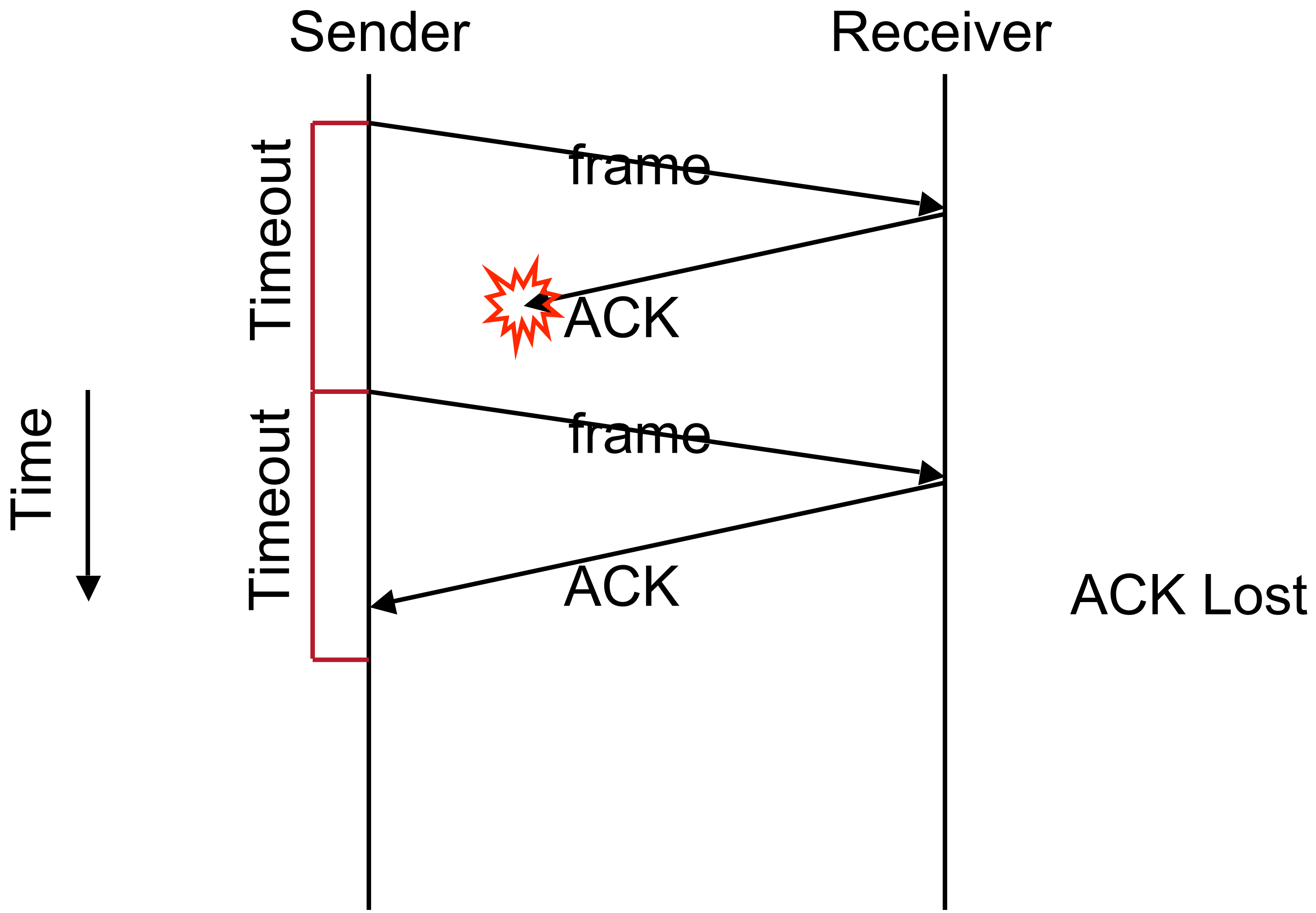
Everything works

Stop-and-Wait scenario 2

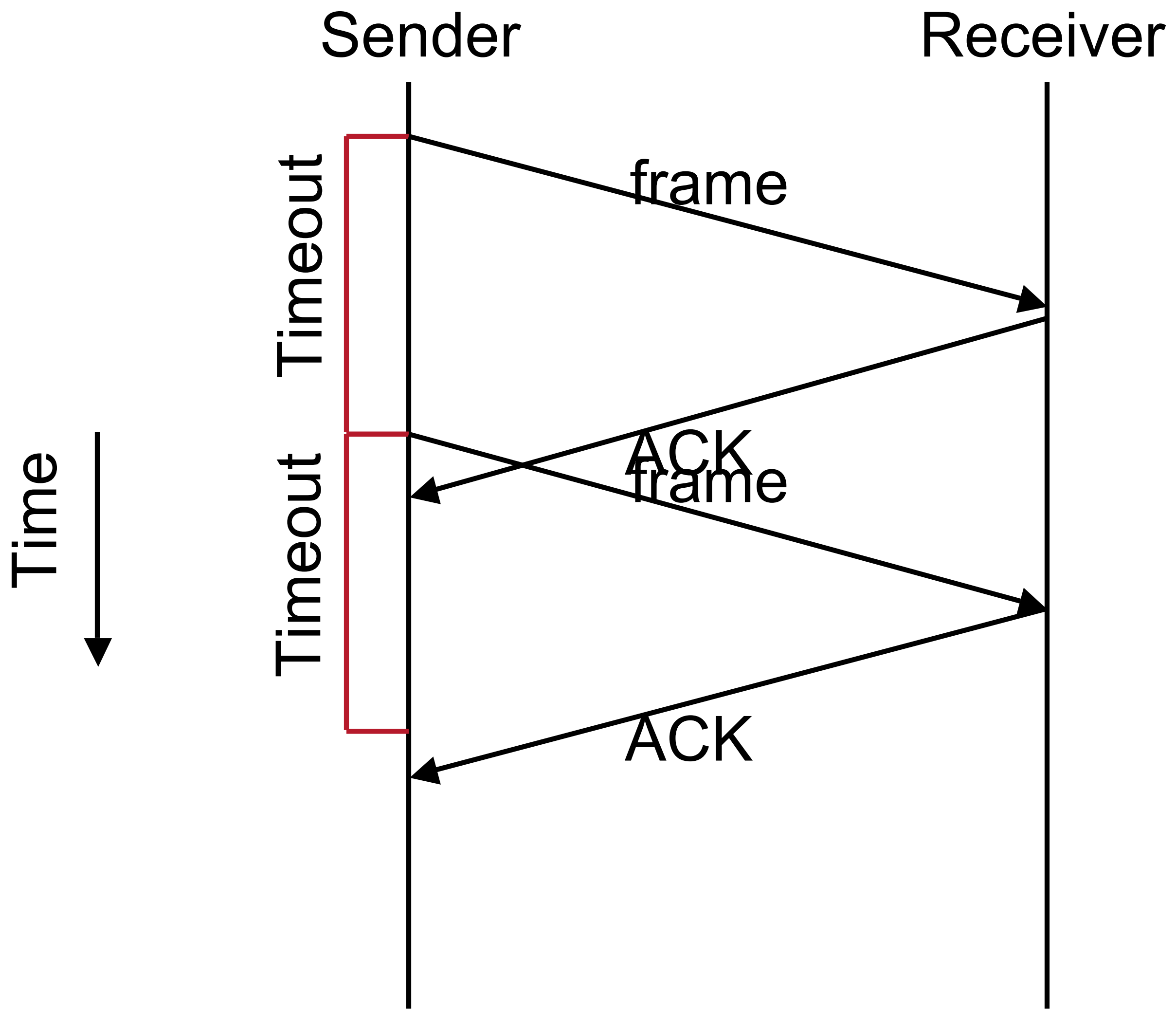


Original frame
Lost & Resent

Stop-and-Wait scenario 3

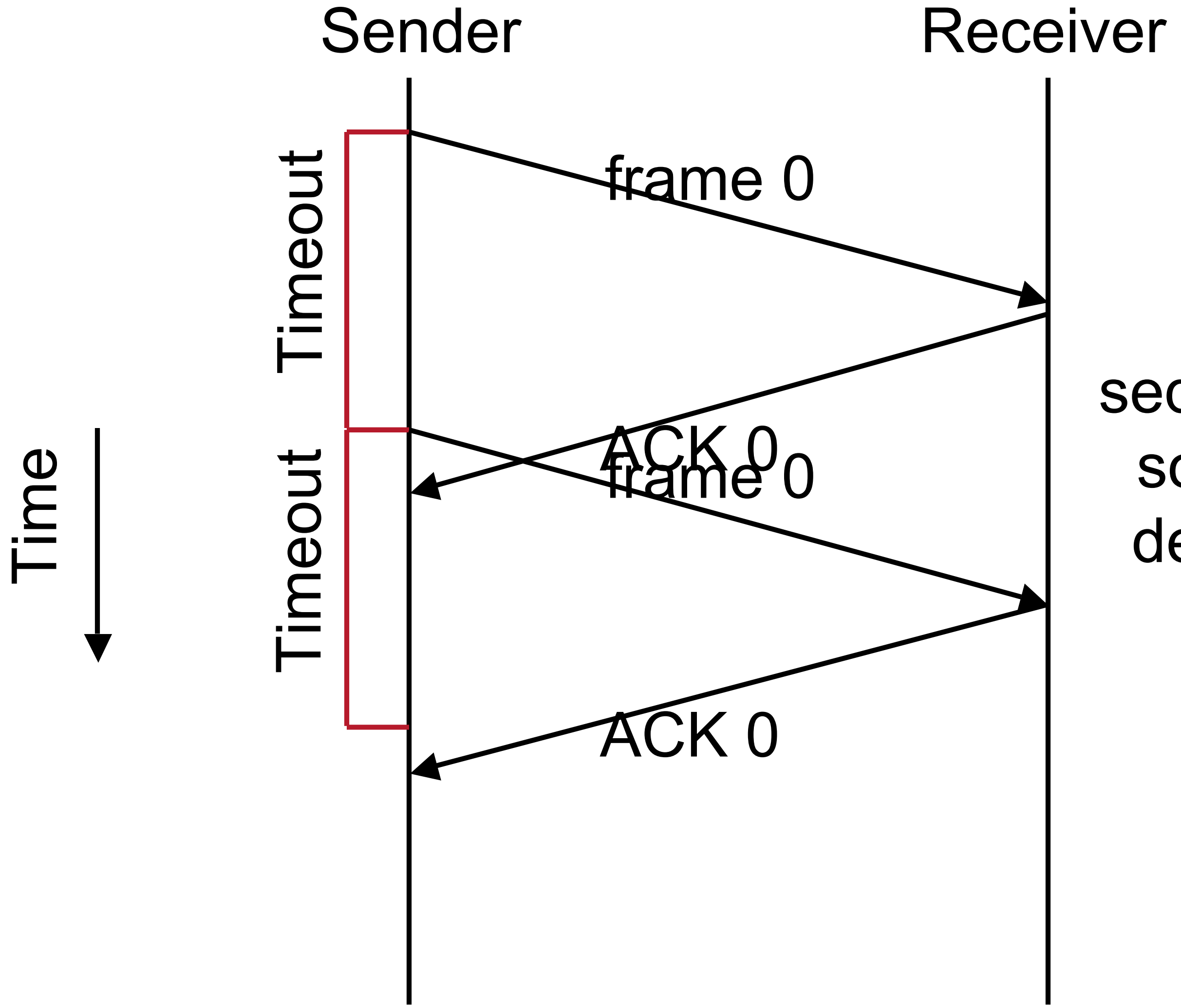


Stop-and-Wait scenario 4



Timeout too short

Sequence numbers



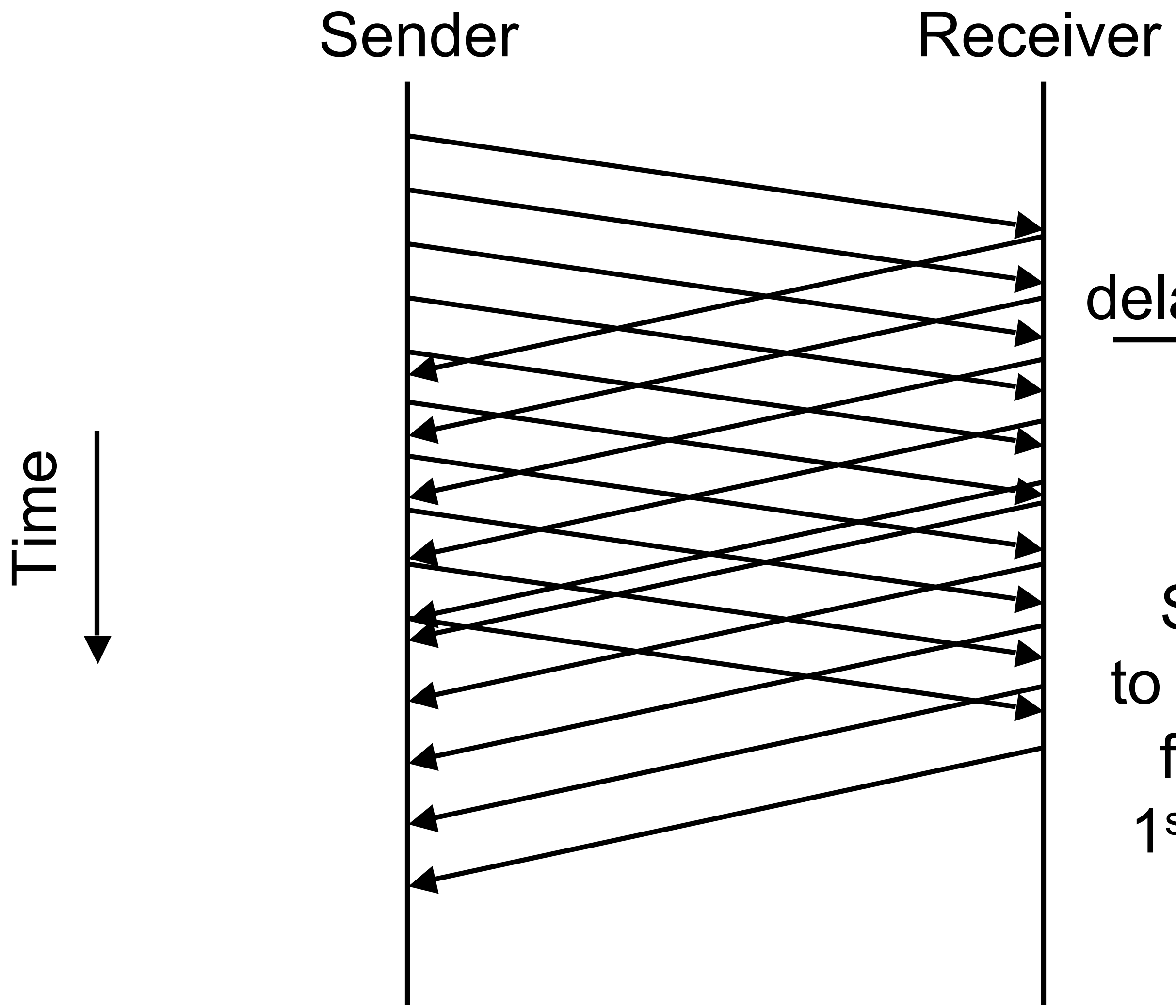
Add a 1-bit sequence number so receiver can detect repeated frames.

Stop-and-Wait

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

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

More efficient solution



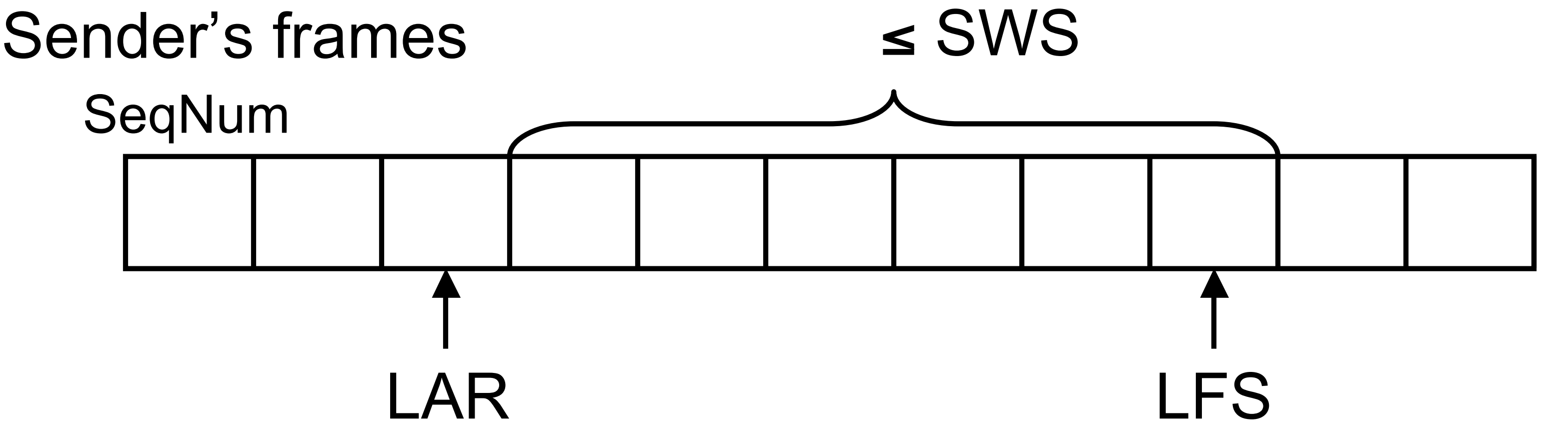
$$N = \frac{\text{delay x bandwidth}}{\text{frame size}}$$

Sender ready
to transmit N+1st
frame just as
1st ACK arrives.

Sliding Window Algorithm

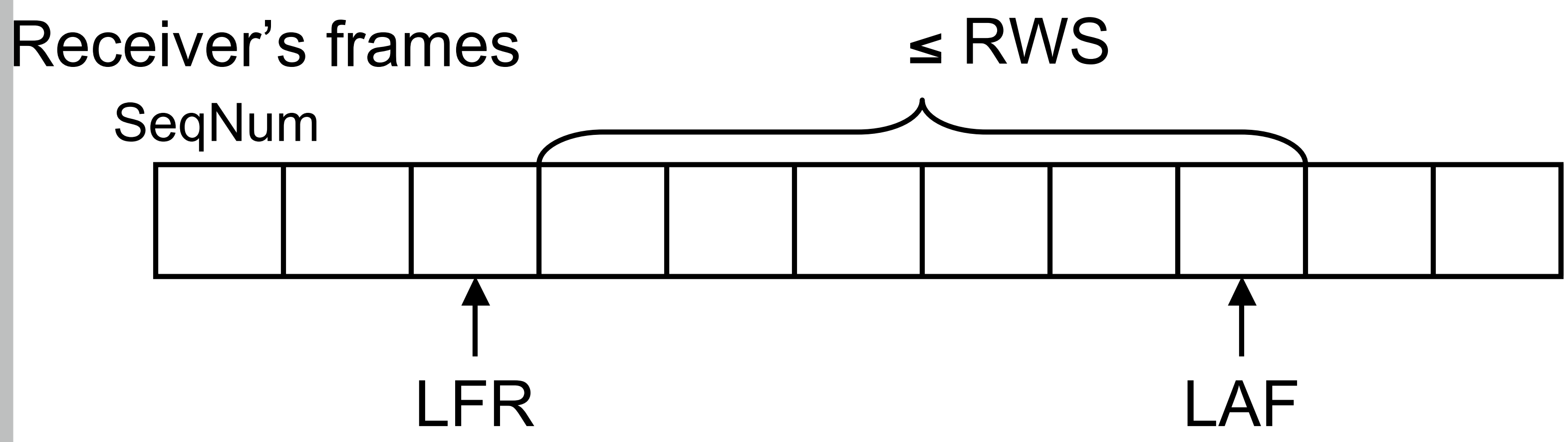
- Sender assigns a *sequence number* to each frame: SeqNum
  - For now, assume SeqNum can grow infinitely
- Send Window Size (SWS)
  - Upper bound on # of unacknowledged frames sender can transmit
- Last ACK Received (LAR)
  - Sequence number of last ACK
- Last Frame Sent (LFS)

Sender Invariant



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

Receiver

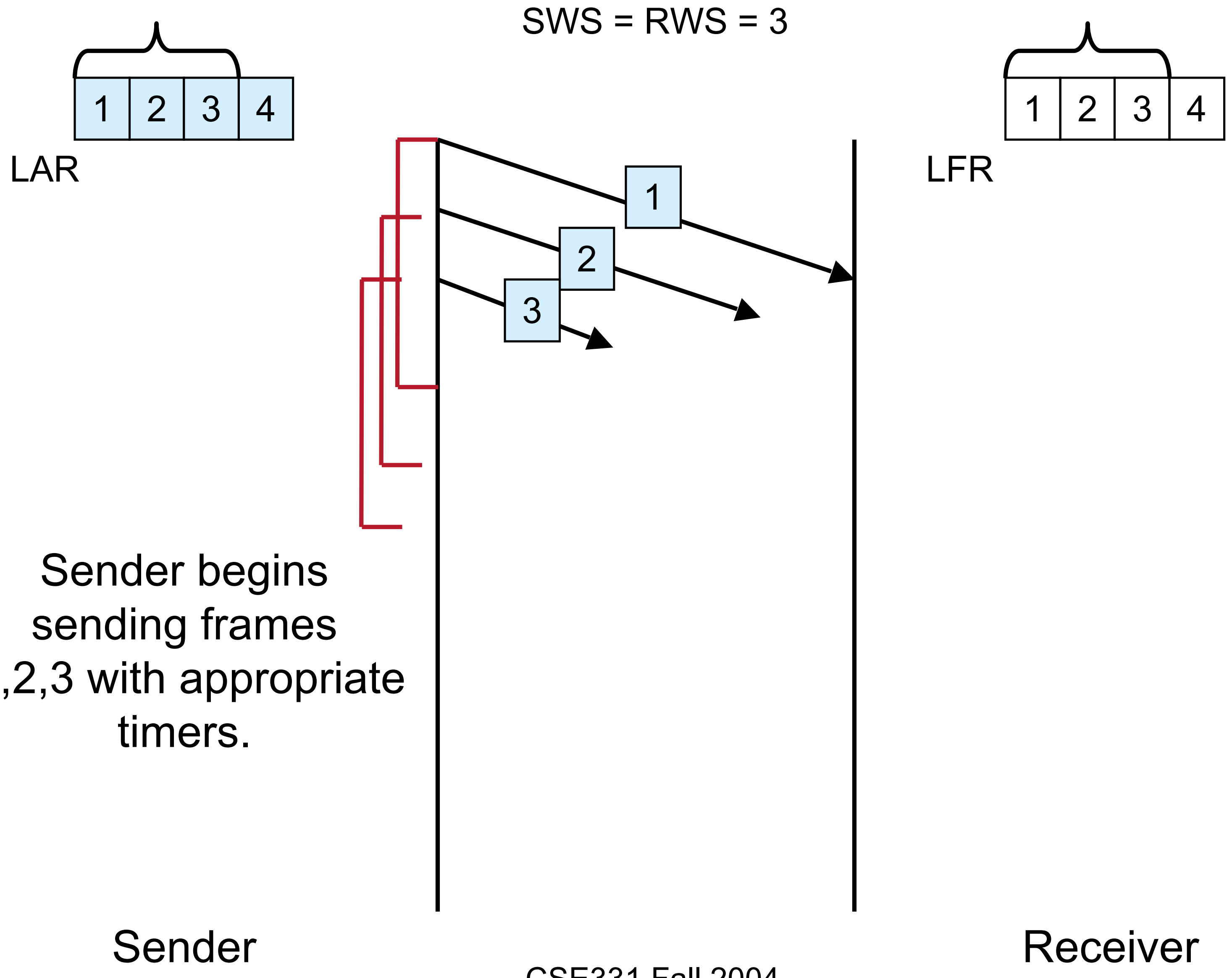


- Receive Window Size (RWS)
  - – Number of out-of-order frames it will accept
- Largest Acceptable Frame (LAF)
- Largest Frame Received (LFR)
- LAF – LFR ≤ RWS

Receiver Algorithm

- When packet numbered SeqNum arrives
  - If (SeqNum ≤ LFR) or (SeqNum > LAF) discard
  - Else accept the packet
- Define: SeqNumToAck
  - Largest unACK'ed sequence # s.t. all earlier frames have been accepted
- Receiver sends ACK(SeqNumToAck)
- LFR = SeqNumToAck
- Laf = LFR + RWS

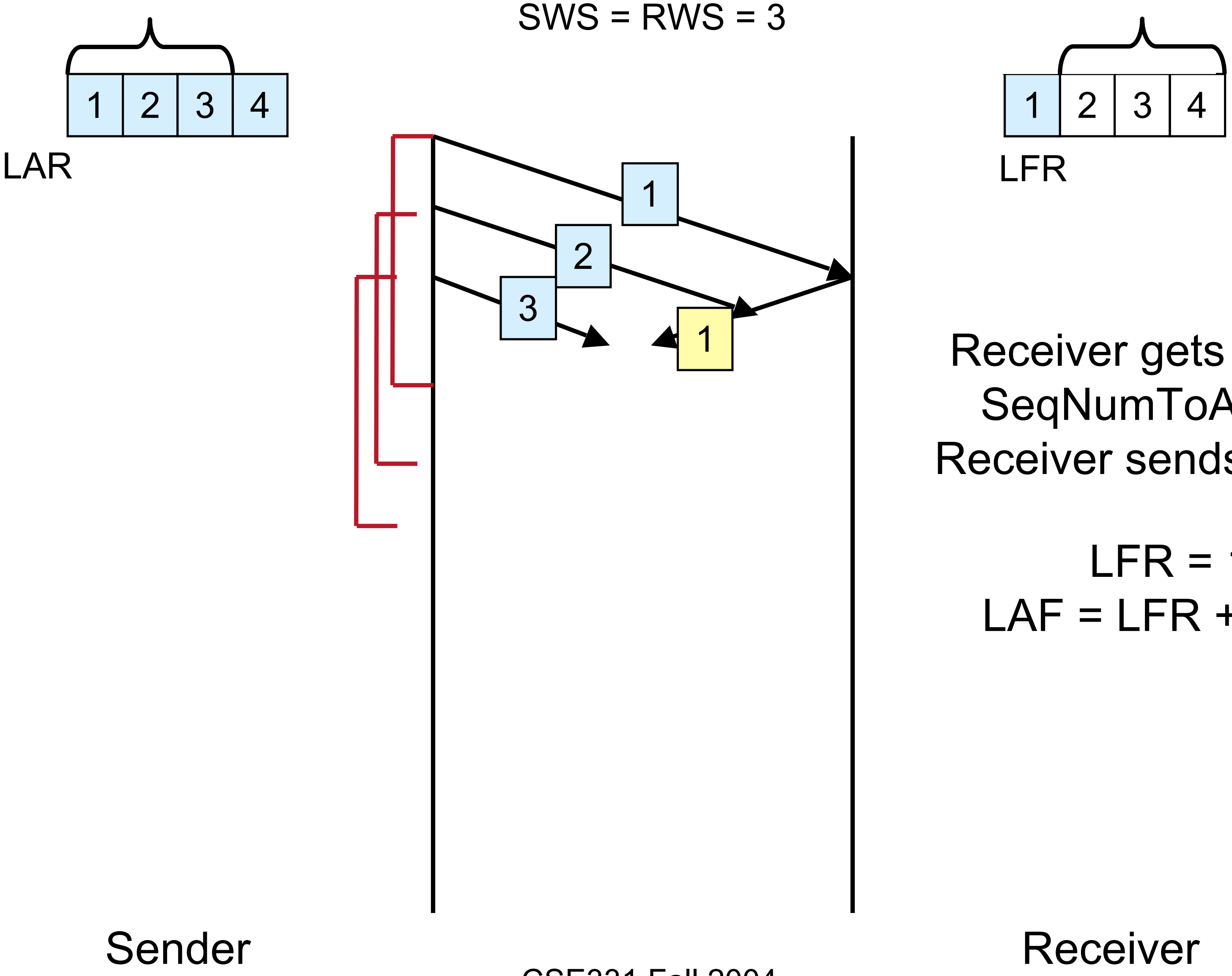
Example Sliding Window Protocol



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

Example Sliding Window Protocol

SWS = RWS = 3

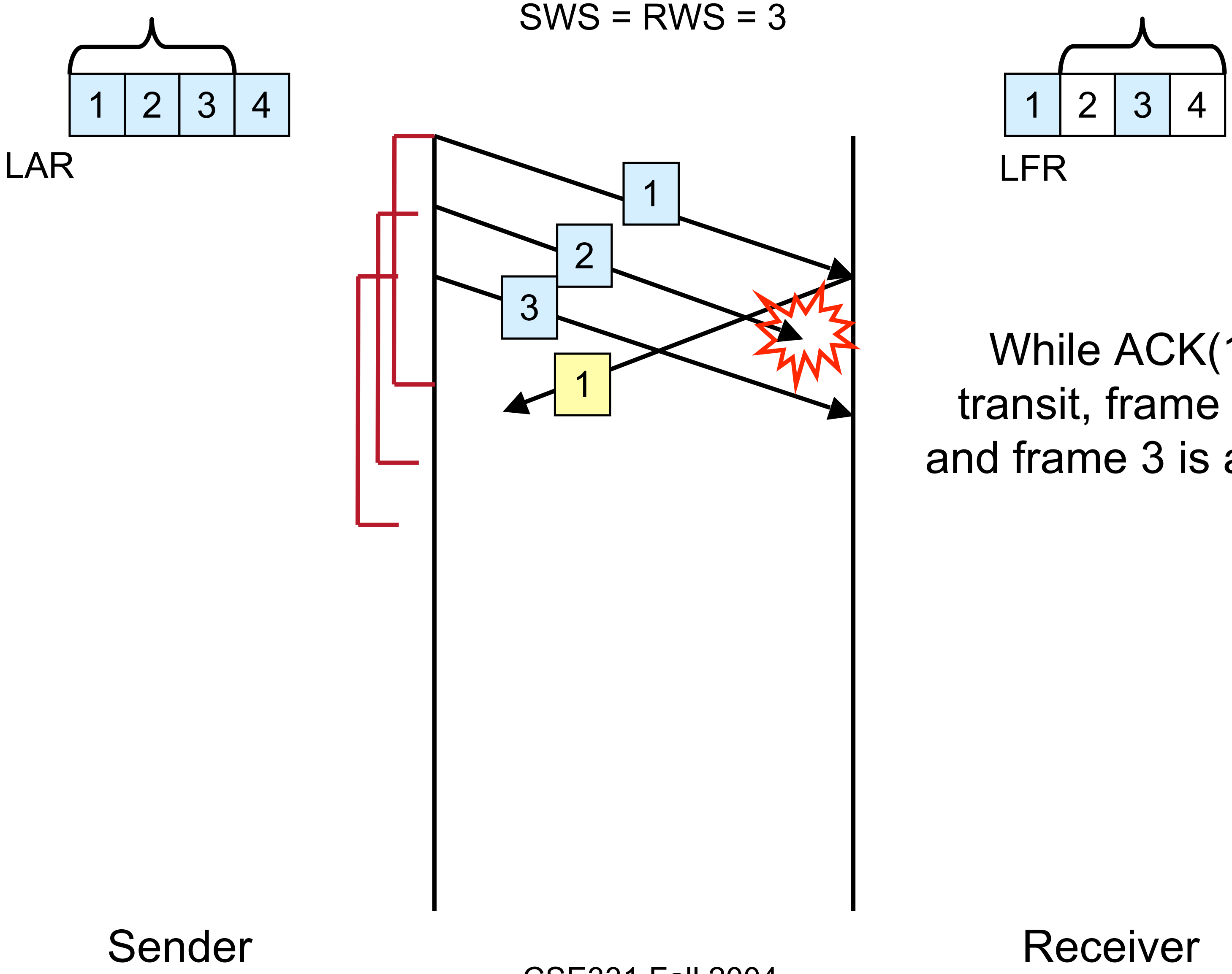


Receiver gets frame 1
SeqNumToAck = 1
Receiver sends ACK(1)

LFR = 1
LAF = LFR + RWS

Example Sliding Window Protocol

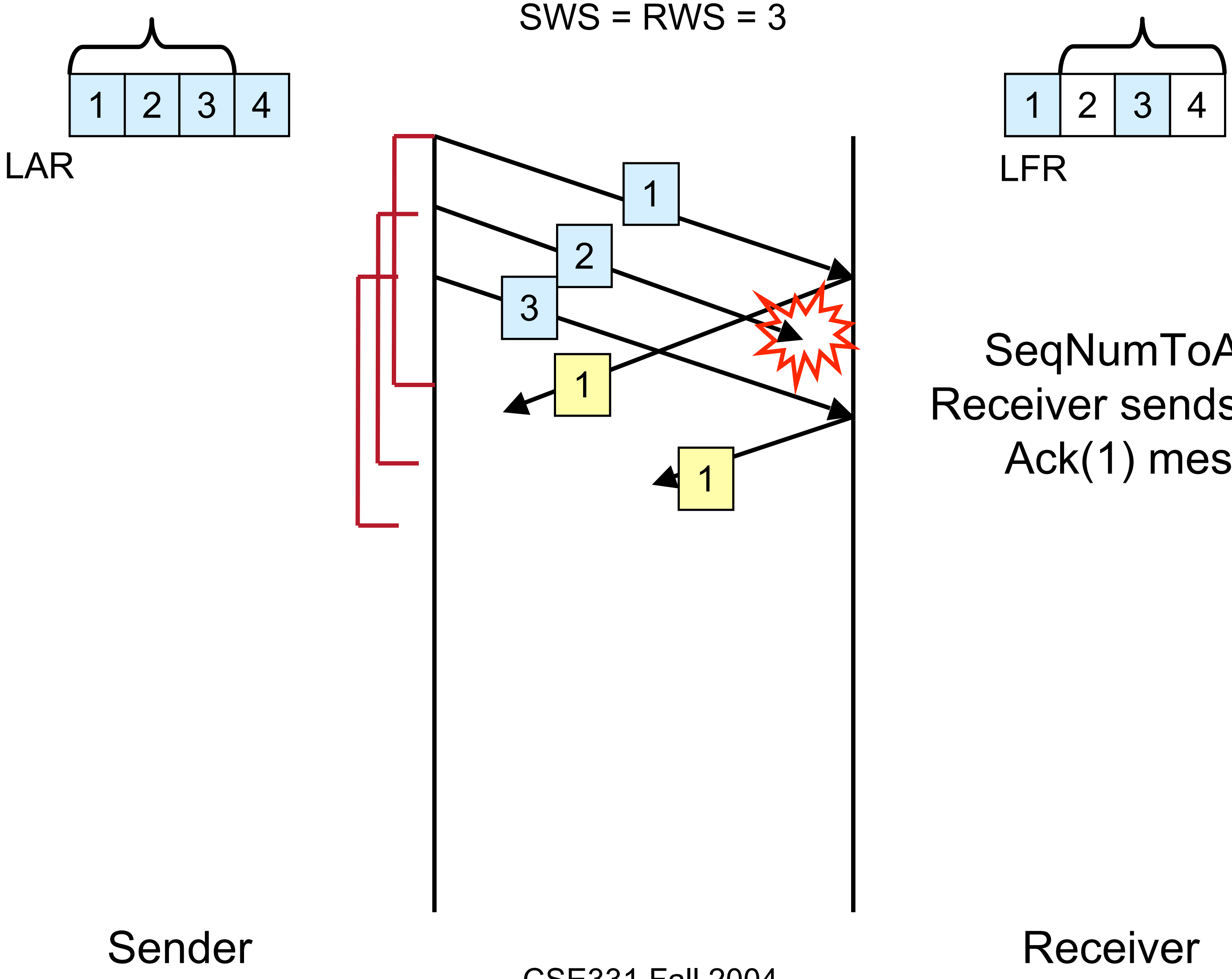
SWS = RWS = 3



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

Example Sliding Window Protocol

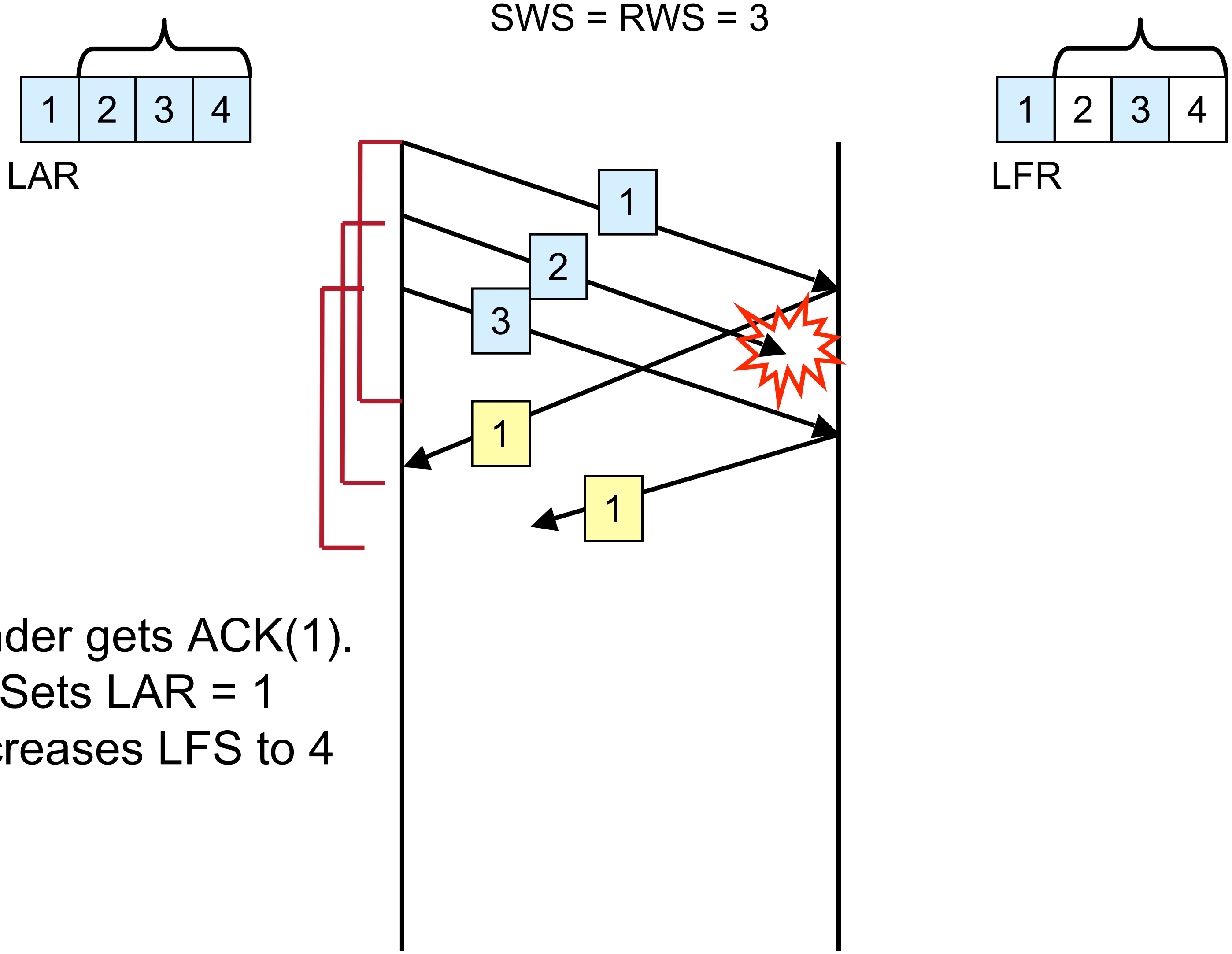
SWS = RWS = 3



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

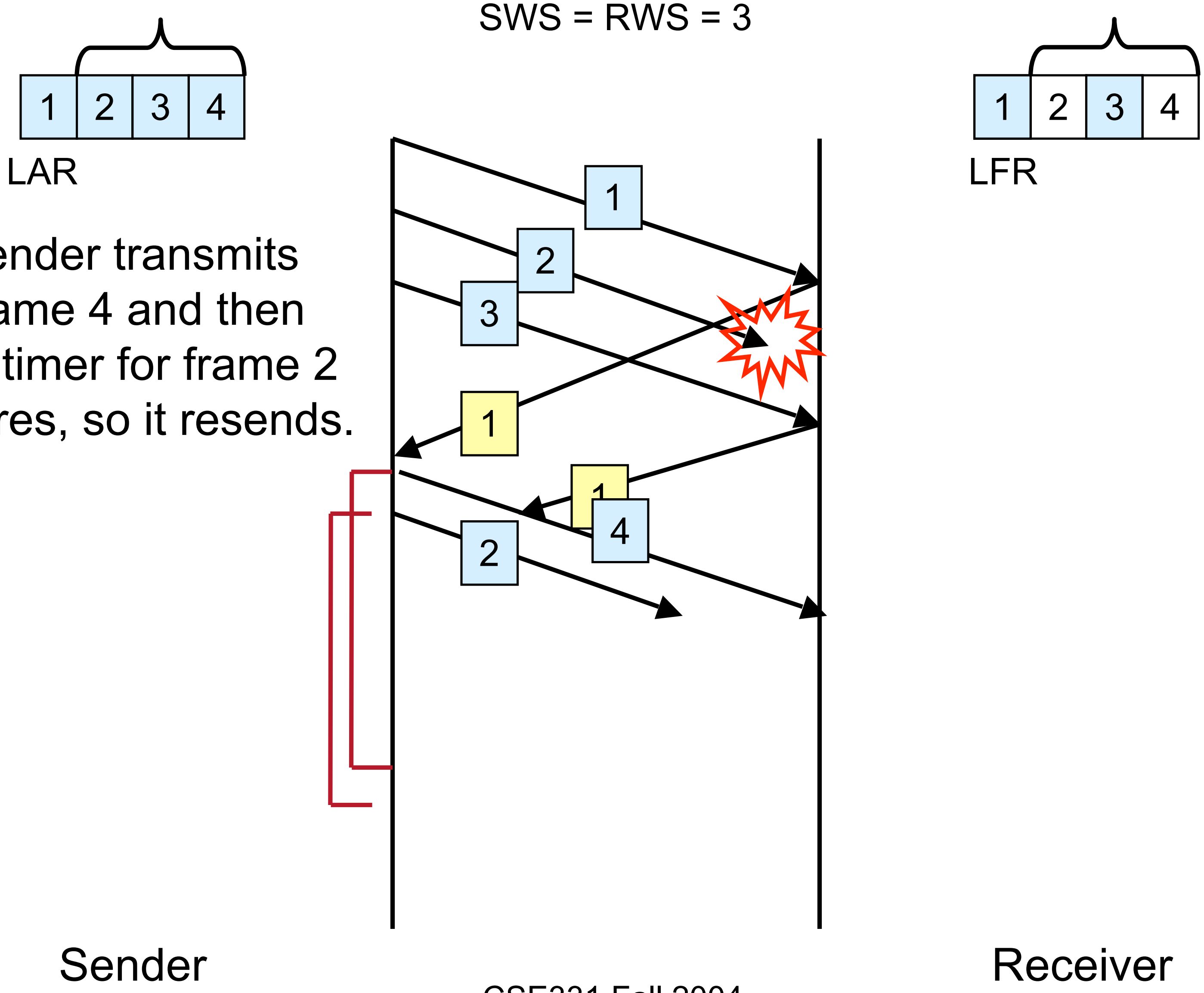
Sender

Receiver

Example Sliding Window Protocol

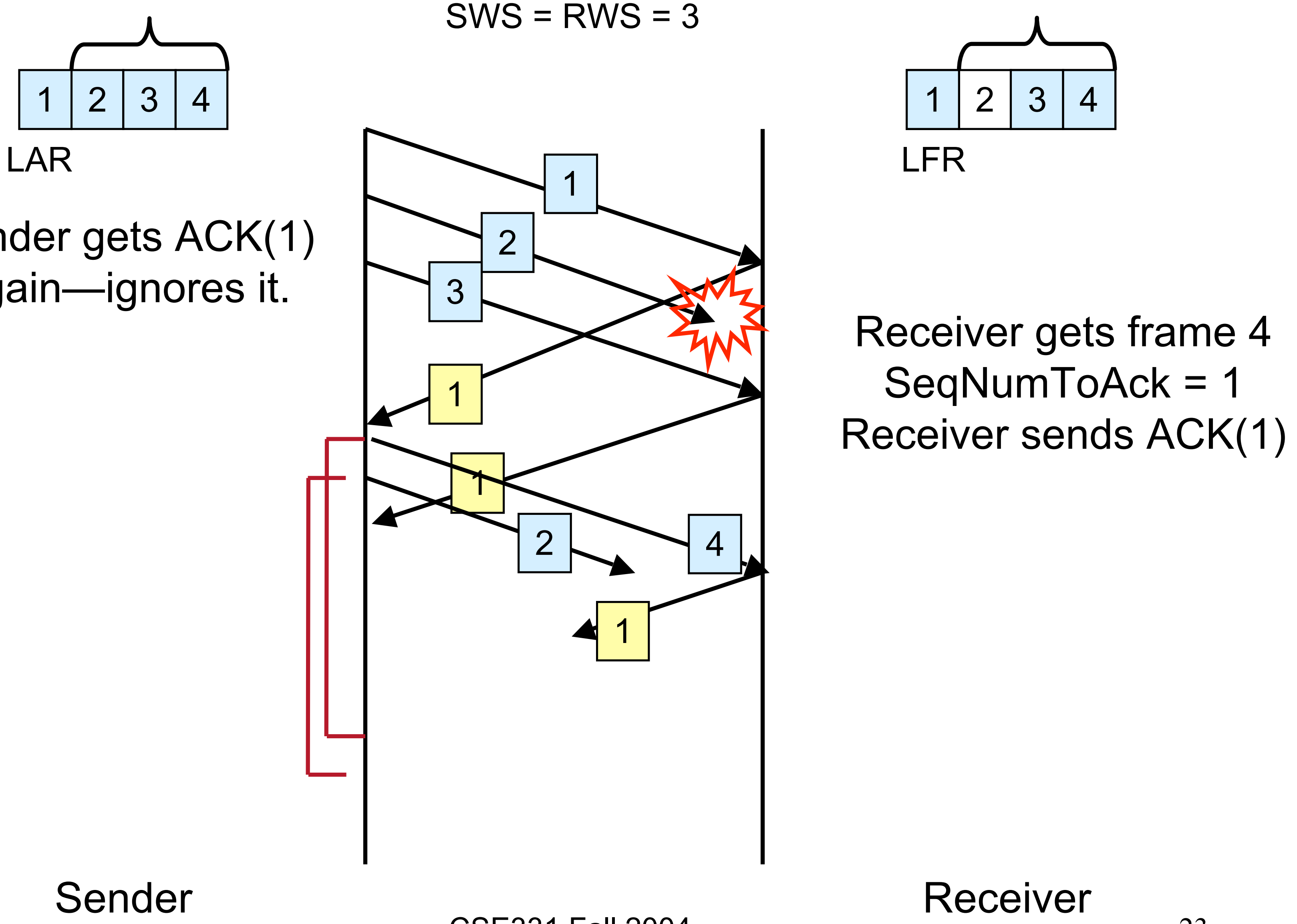
SWS = RWS = 3

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



Example Sliding Window Protocol

SWS = RWS = 3

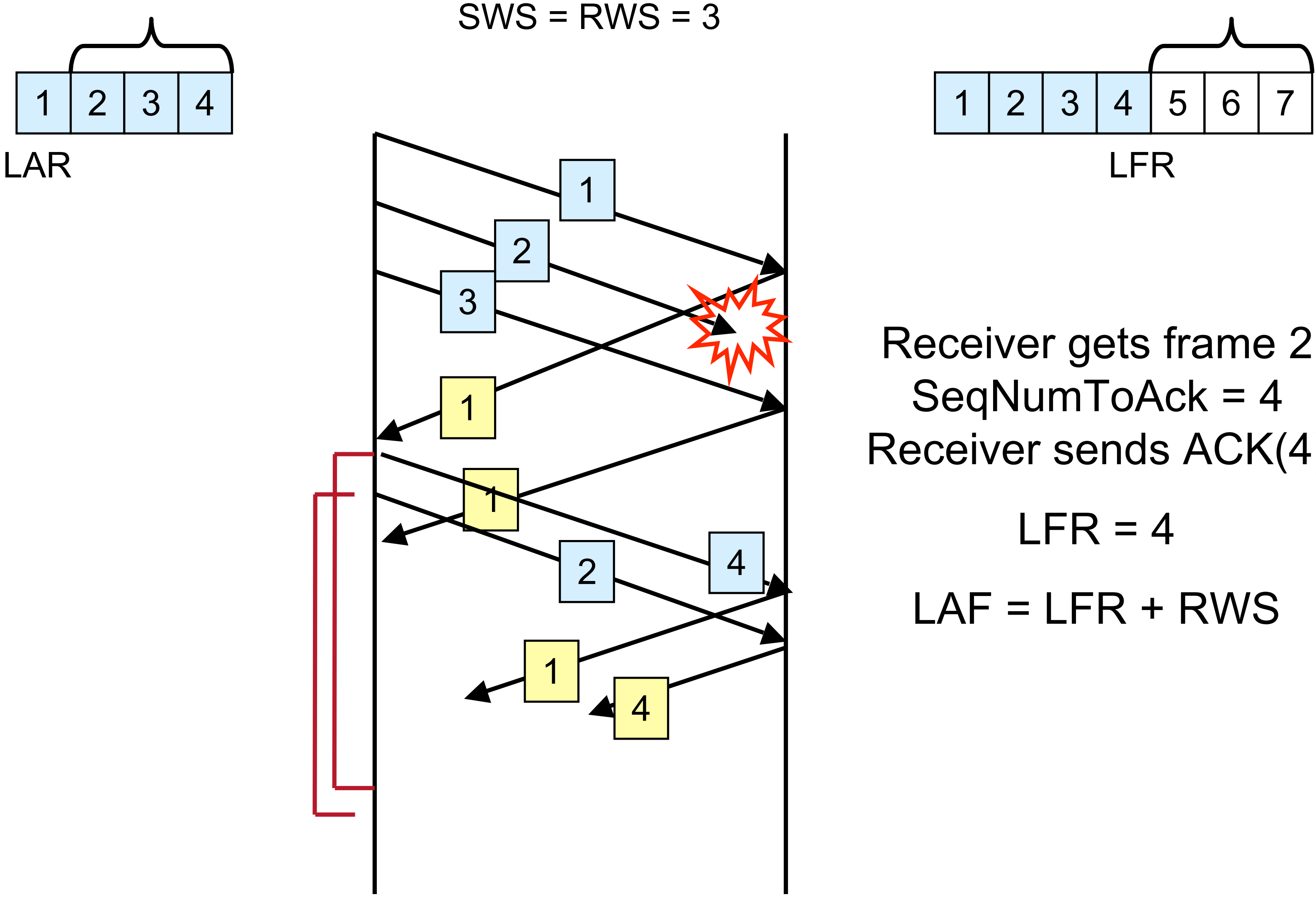


Sender gets ACK(1) again—ignores it.

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

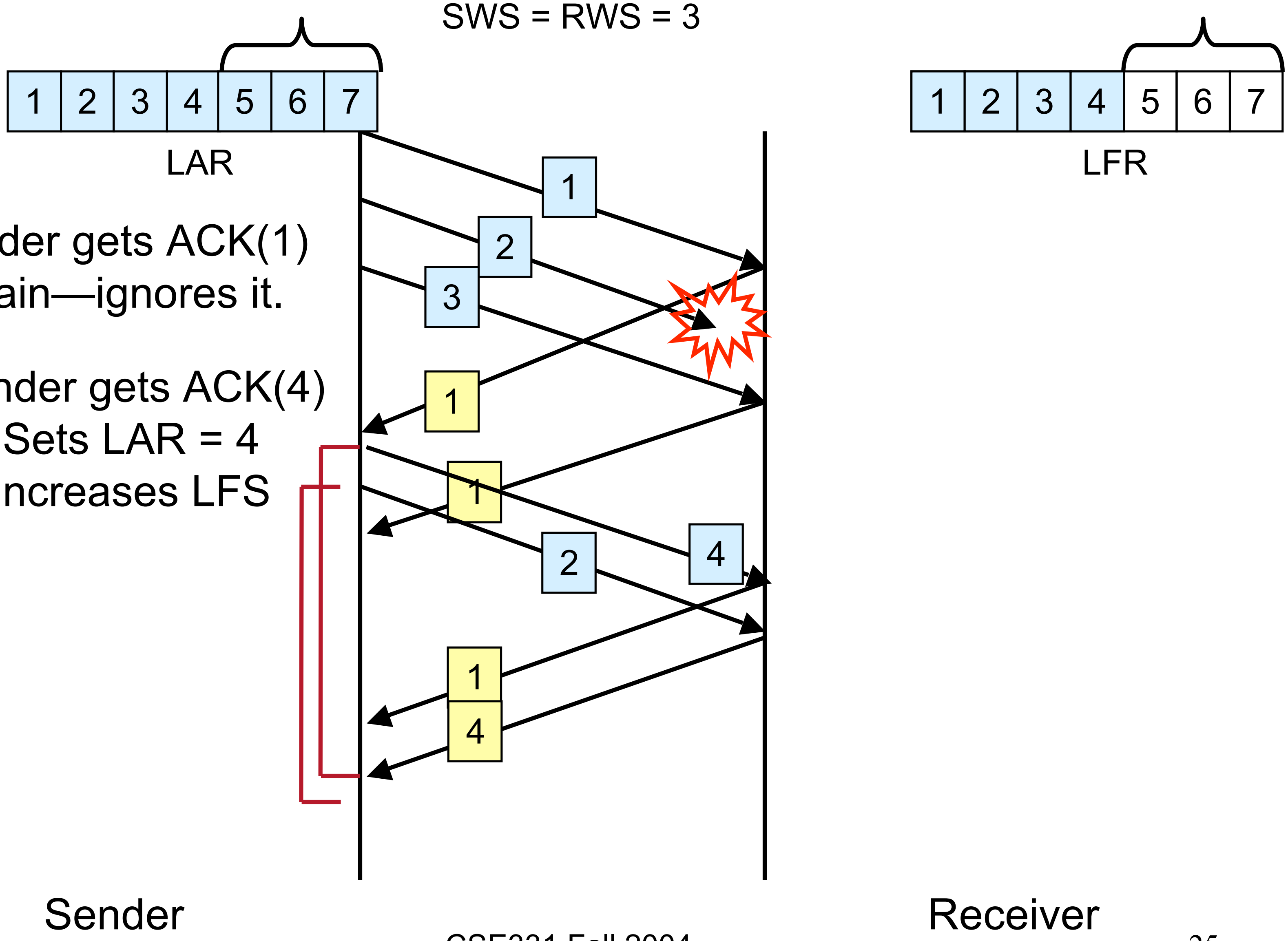
LAF = LFR + RWS

Sender

Receiver

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
- Receiver transmits *selective ACKS*
  - ACK indicates exactly which frames have been accepted

Window Sizes

- If RTT x Bandwidth product is known then SWS = RTT x Bandwidth / Framesize

- Receive window size:
  - 1 = no buffering of out-of-order frames
  - RWS = SWS buffers as many as can be in flight
  - Note that RWS > SWS is not sensible

Finite Sequence Numbers

- Recall that for Stop-and-Wait we needed two sequence numbers.
- How many do we need for Sliding Window?
- Suppose SWS=RWS
  - How many sequence numbers should there be?
  - Is SWS + 1 sufficient?

Sufficient MaxSeqNum

- Frame i's sequence num is i%MaxSeqNum

- Assuming SWS = RWS
- SWS < (MaxSeqNum + 1)/2

- Why?
  - Consider case where all the ACKS are lost.
  - Suppose SWS = RWS = 3
  - MaxSeqNum = 5 (sequence numbers = 0,1,2,3,4) is insufficient

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 sophisticate version could give # of frames the receiver has room for—throttles the sender.

Sliding window in practice

- TCP (Transmission Control Protocol)
  - Transportation layer protocol
  - Uses sliding window algorithm
  - More complex because it’s used in an Internetwork – not over a direct link

  - Bandwidth x delay not known
  - Dynamically changes timeouts
  - Larger buffers for in-order delivery