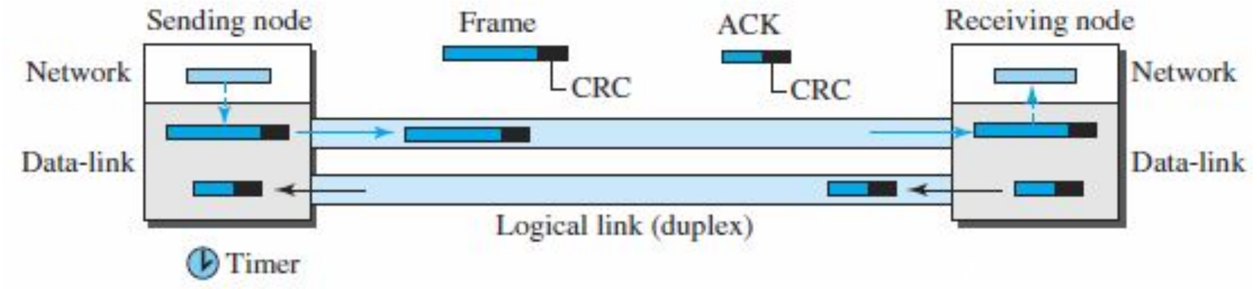
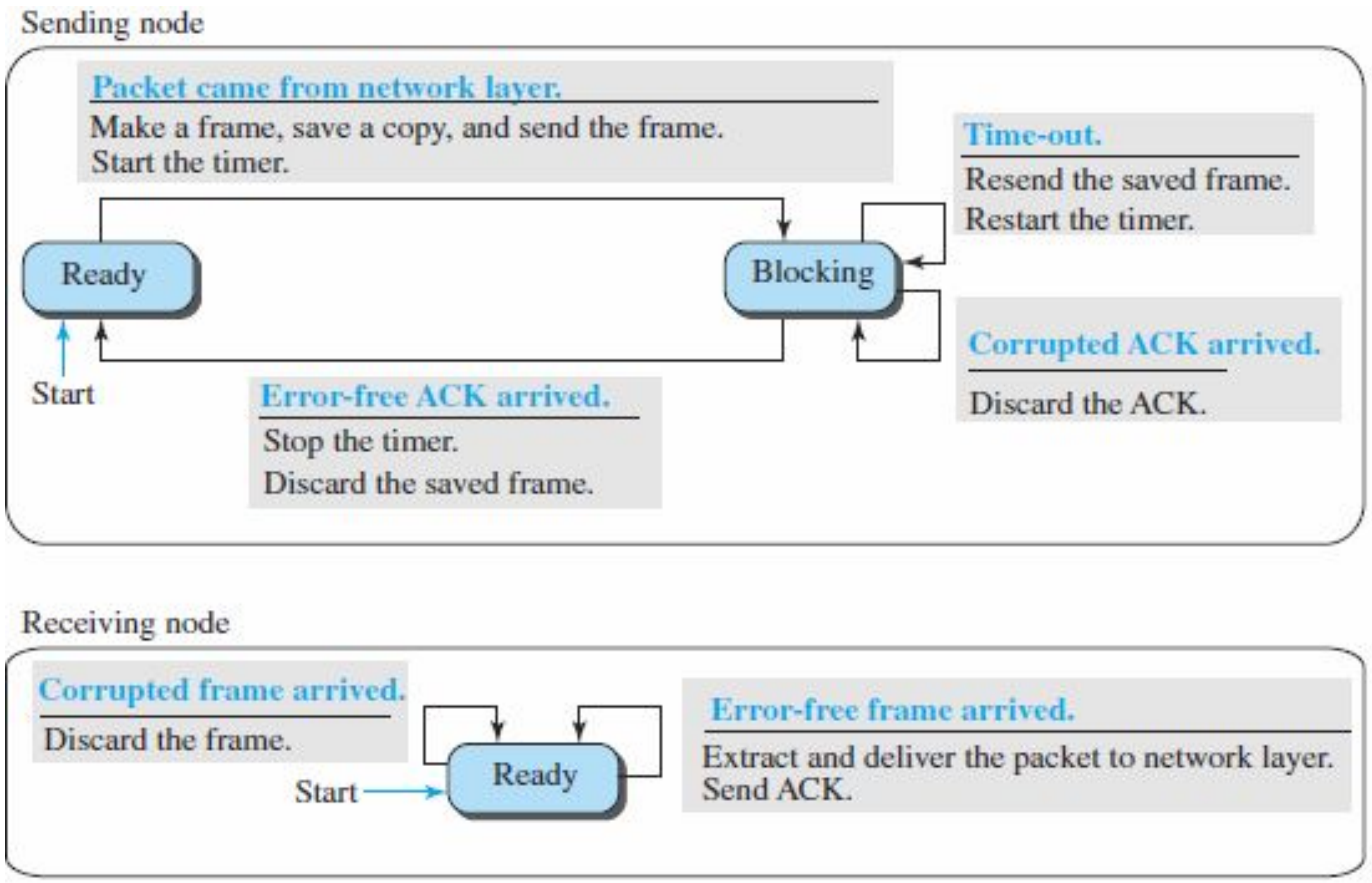
Stop and Wait
Go back N
Selective Repeat

Stop-and-Wait Protocol

- Uses both flow and error control
- Sender sends one frame at a time and waits for an acknowledgment before sending the next one
- To detect corrupted frames, CRC is added to each data frame
- When a frame arrives at the receiver site, if its **CRC is incorrect**, the frame is corrupted and silently **discarded**
- The silence of the receiver is a signal for the sender that a frame was either corrupted or lost.
  - Every time the sender sends a frame, it starts a timer
  - If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame (if it has one to send)
  - If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted
    - This means that the sender needs to keep a copy of the frame until its acknowledgment arrives
  - When the corresponding acknowledgment arrives, the sender discards the copy and sends the next frame if it is ready



Stop-and-Wait Protocol FSM



Stop-and-Wait Protocol FSM

Sender States

- The sender is initially in the ready state, but it can move between the ready and blocking state
- **Ready State:** only waiting for a packet from the network layer
  - If a packet comes from the network layer, the sender creates a frame, saves a copy of the frame, starts the only timer and sends the frame
  - The sender then moves to the blocking state.
- **Blocking State:** three events can occur:
  - If a time-out occurs, the sender resends the saved copy of the frame and restarts the timer
  - If a corrupted ACK arrives, it is discarded
  - If an error-free ACK arrives, the sender stops the timer and discards the saved copy of the frame
    - It then moves to the ready state.

Stop-and-Wait Protocol FSM

Receiver States

- The receiver is always in the ready state. Two events may occur:
  - If an error-free frame arrives, the message in the frame is delivered to the network layer and an ACK is sent
  - If a corrupted frame arrives, the frame is discarded

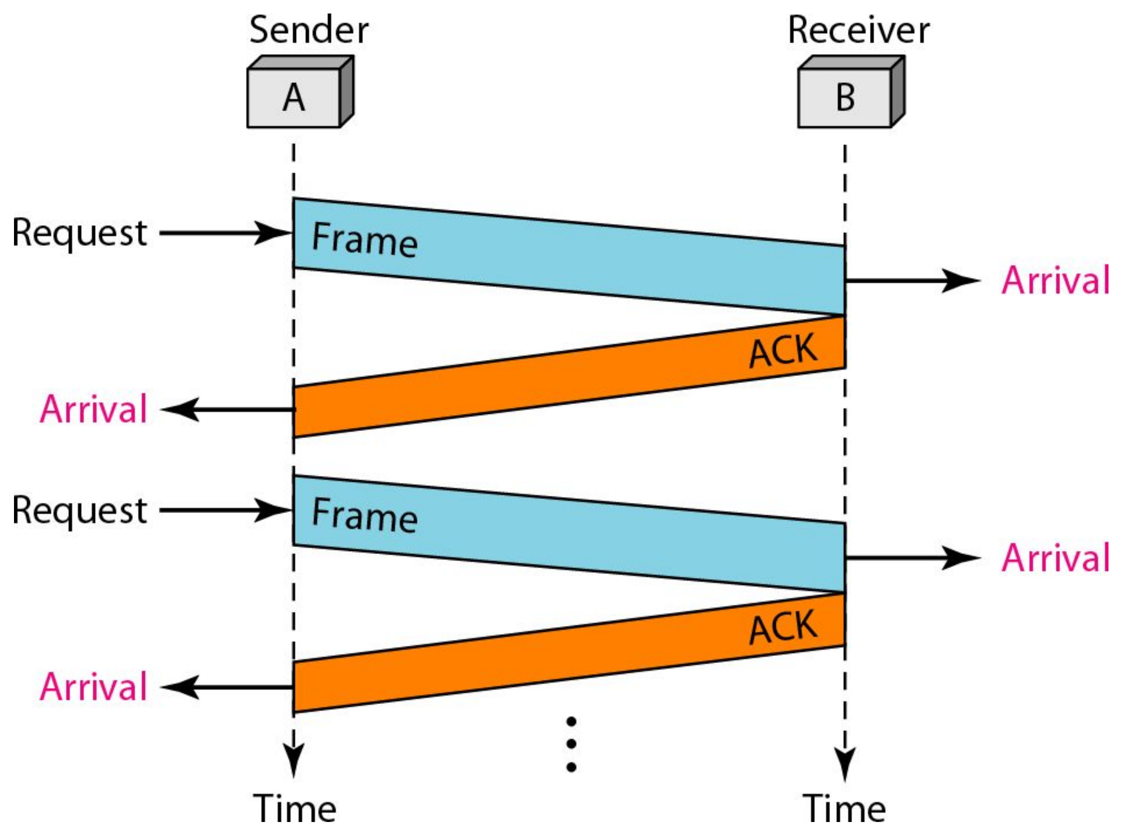
- **Cases of Operations:**

1. Normal operation
2. The frame is lost
3. The Acknowledgment (ACK) is lost
4. The ACK is delayed

Stop-and-Wait Protocol Example -1

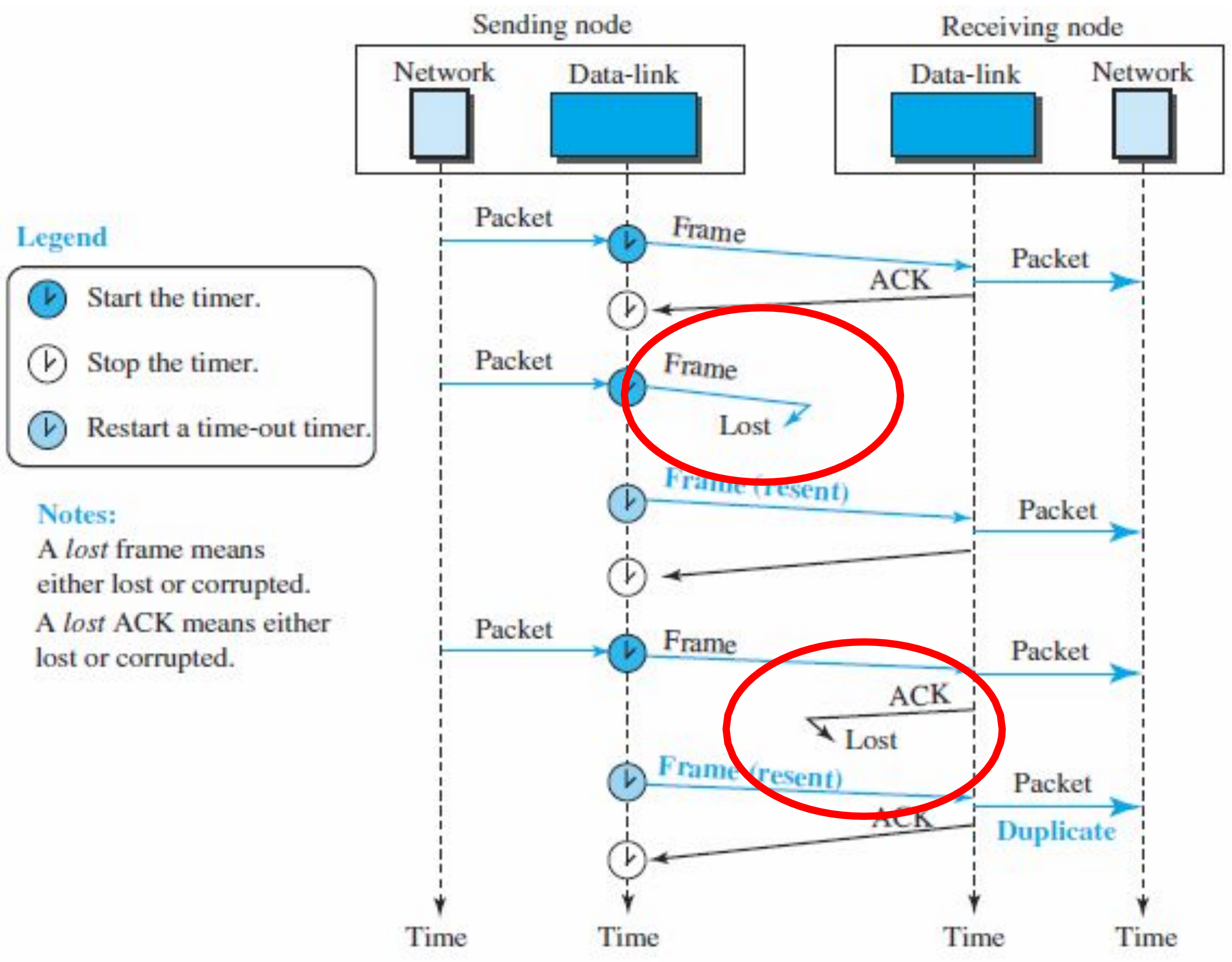
Stop-and-Wait Protocol Example -1

- The sender sends one frame and waits for feedback from the receiver
- When the ACK arrives, the sender sends the next frame



Stop-and-Wait Protocol Example -2

Stop-and-Wait Protocol Example -2



Stop-and-Wait Protocol Example -2

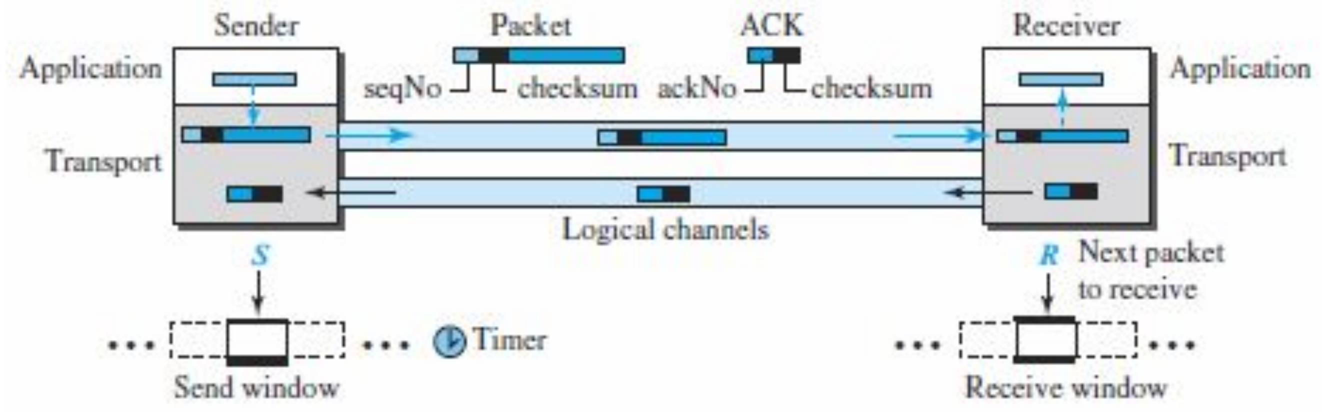
Stop-and-Wait Protocol Example -2

- The first frame is sent and acknowledged
- The second frame is sent, but lost
  - After time-out, it is resent
- The third frame is sent and acknowledged, but the acknowledgment is lost
  - The frame is resent
  - Problem: The network layer at the receiver site receives **two copies** of the third packet, which is not right
    - Solution: use **sequence numbers and acknowledgment numbers**

Stop-and-Wait Protocol

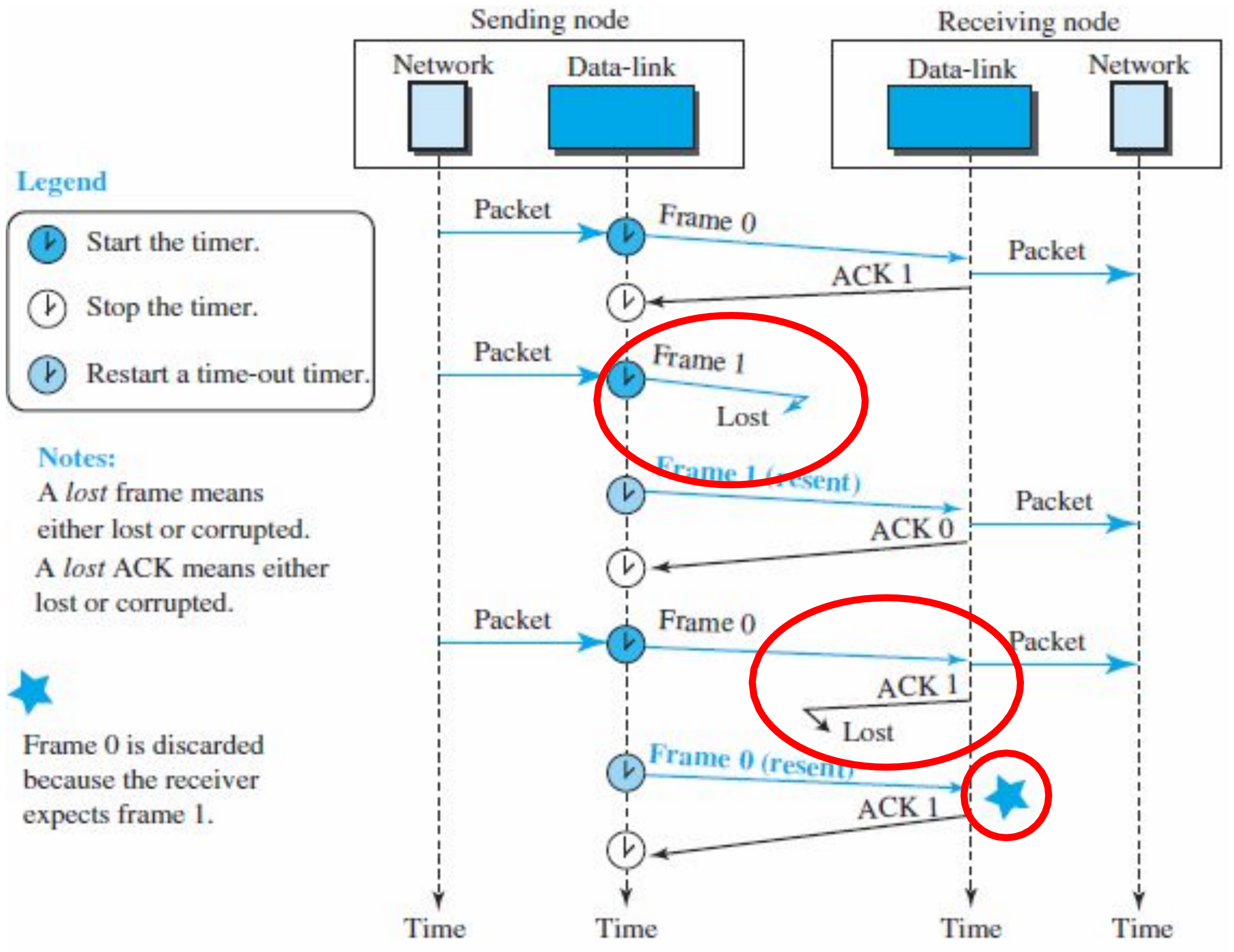
Sequence and Acknowledgment Numbers

- Duplicate packets need to be avoided
- Add sequence numbers to the data frames and acknowledgment numbers to the ACK frames
- The acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected
  - Sequence numbers are 0, 1, 0, 1, 0, 1, . . . ; the acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, …
    - the sequence numbers start with 0, the acknowledgment numbers start with 1.
    - An acknowledgment number always defines the sequence number of the next frame to receive



Stop-and-Wait Protocol Example -3

Stop-and-Wait Protocol Example -3



Stop-and-Wait Protocol Example -3

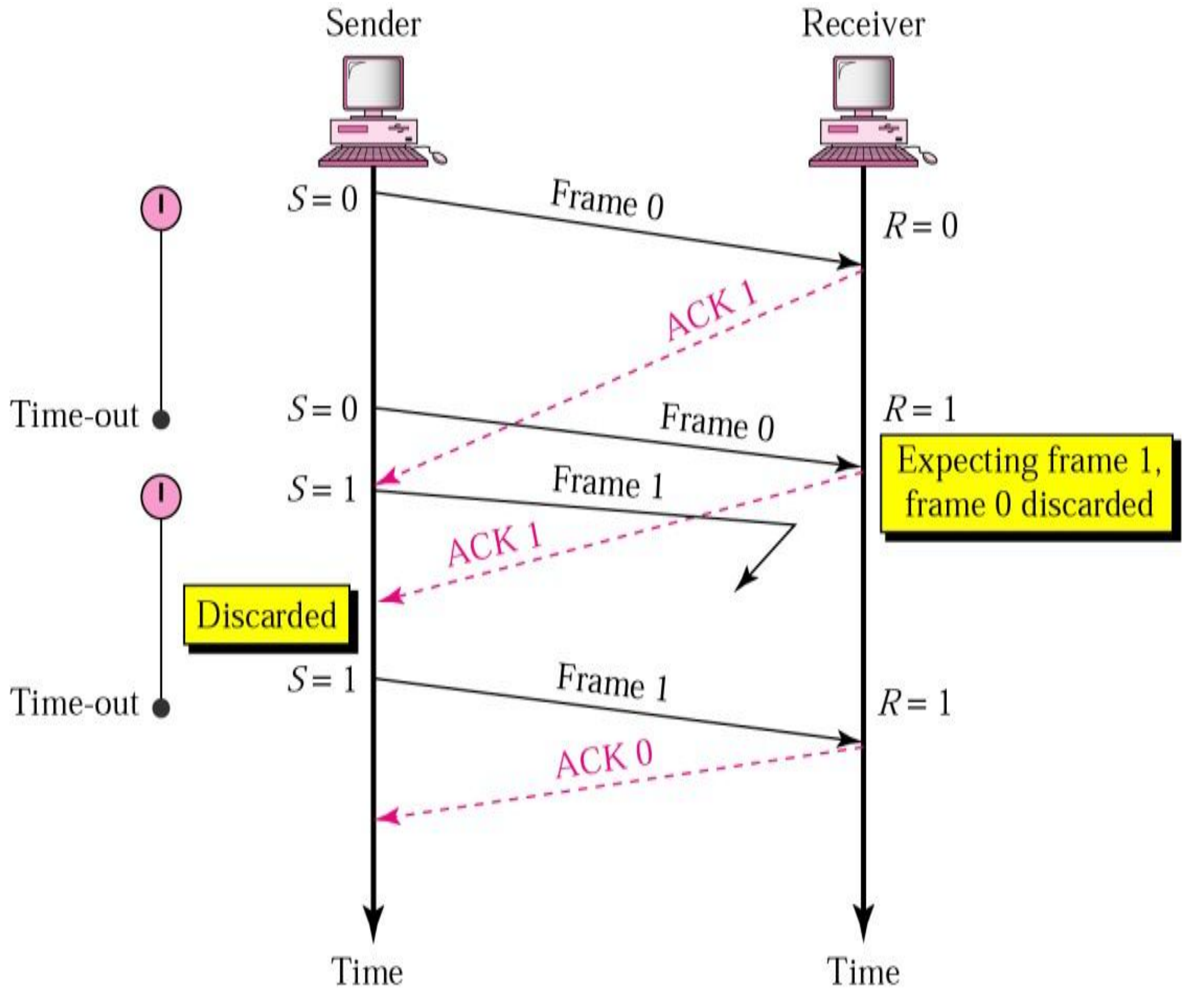
Stop-and-Wait Protocol Example -3

- Adding sequence numbers and acknowledgment numbers can prevent duplicates
  - The first frame is sent and acknowledged
  - The second frame is sent, but lost
    - After time-out, it is resent
  - The third frame is sent and acknowledged, but the acknowledgment is lost
    - The frame is resent

Stop-and-Wait Protocol Example -4

Stop-and-Wait Protocol Example -4

- **Delayed ACK and lost frame**



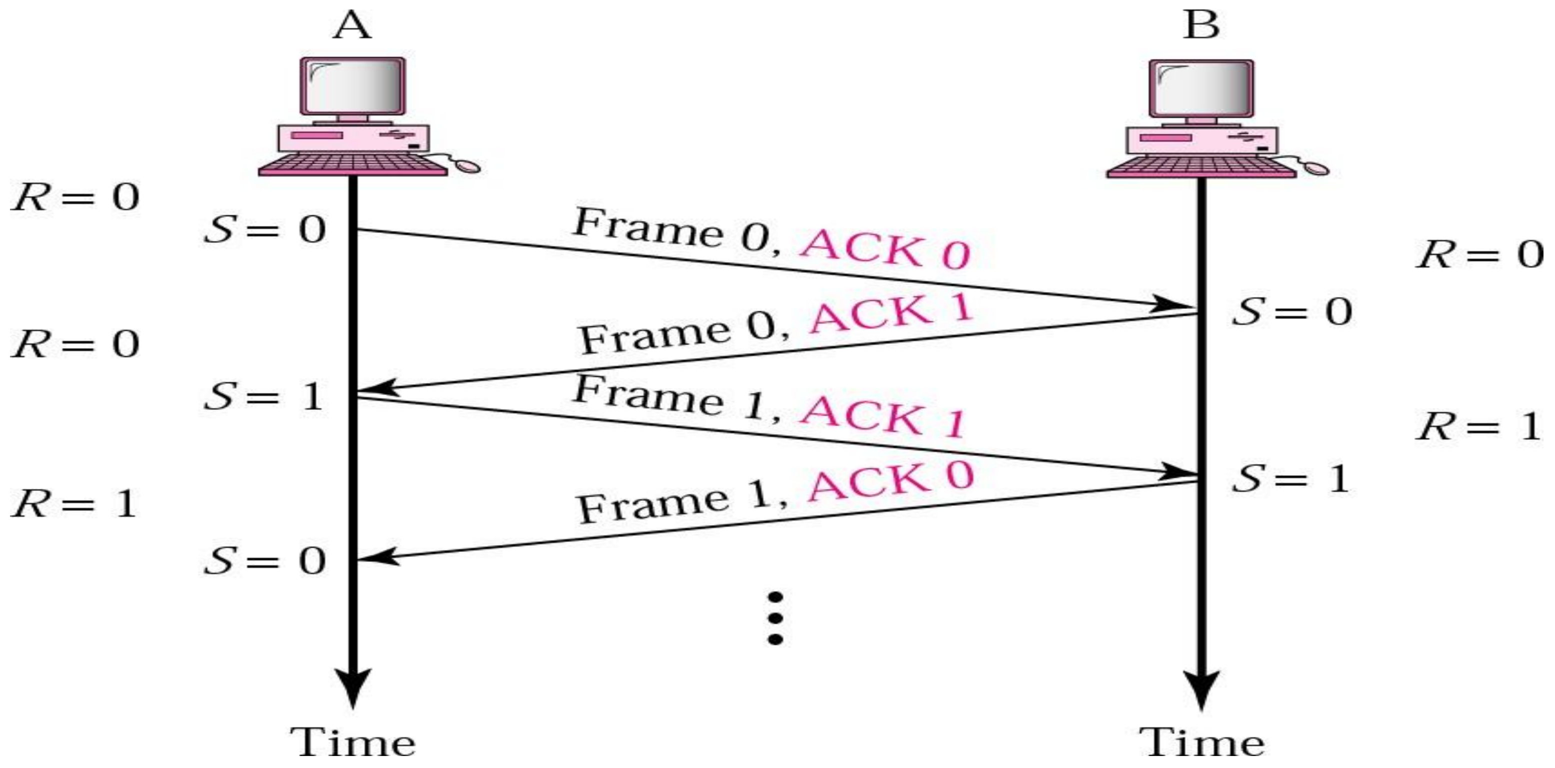
Piggybacking

Piggybacking

- Simple/stop-and-wait protocols are designed
  - unidirectional communication, in which data is flowing only in one direction although the acknowledgment may travel in the other direction
- To allow data to flow in both directions to make the communication more efficient, the data in one direction is piggybacked with the acknowledgment in the other direction
  - When node A is sending data to node B, Node A also acknowledges the data received from node B
- Because piggybacking makes communication at the datalink layer **more complicated**, it is not a common practice

Flow diagram using piggybacking

Flow diagram using piggybacking



Stop-and Wait - Limitations

Stop-and Wait - Limitations

- After each frame sent, the host must wait for an ACK
  - ❖ inefficient use of bandwidth
- To improve efficiency ACK should be sent after multiple frames
  - Pipelining: A task is begun before the previous task has ended

❖ There is no pipelining in stop and wait ARQ because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent

❖ Pipelining improves the efficiency of the transmission

- Alternatives: Sliding Window protocol

✓ Go-back-N ARQ

✓ Selective Repeat ARQ

Sliding window protocols

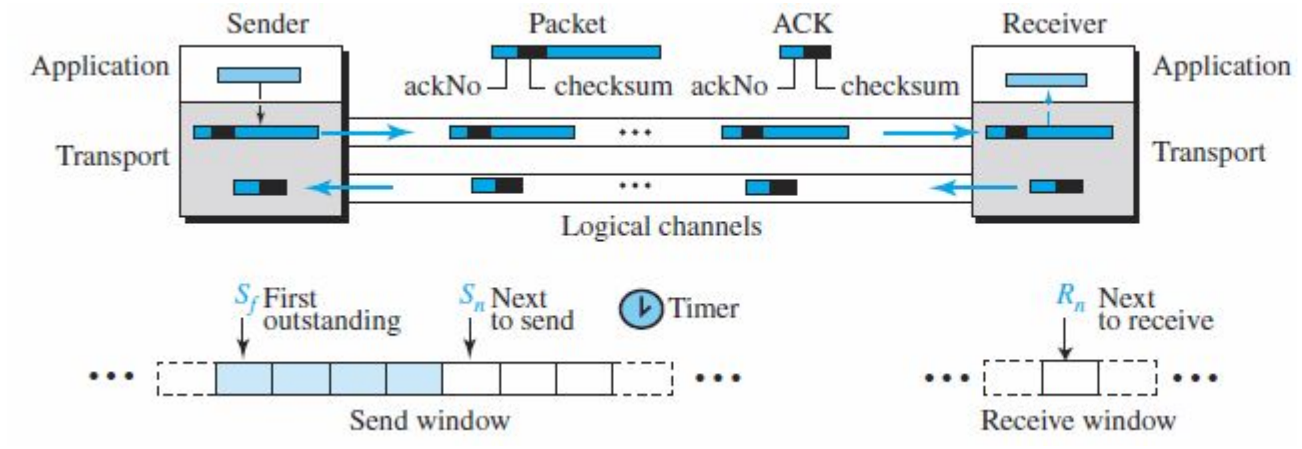
Sliding window protocols

- Sliding window protocols improve the efficiency
- Multiple frames should be in transition while waiting for ACK - Let more than one frame to be outstanding.
- Outstanding frames: frames sent but not acknowledged
- Send up to W frames and keep a copy of these frames (outstanding) until the ACKs arrive
- This procedures requires additional feature to be added : sliding window

Go-Back-*N Protocol (GBN)*

Go-Back-*N Protocol (GBN)*

- Idea of Stop-and-Wait Protocol- how to add flow control to its predecessor
  - Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires
- Go-Back-N send several packets before receiving ACKs but the receiver can only buffer one packet
- Sender keep a copy of sent packets until ACKs arrive



The **sequence numbers** are modulo 2^m, where m is the size of the sequence number field in bits

The **acknowledgment number** is **cumulative** and defines the sequence number of the next packet expected to arrive

GBN Windows

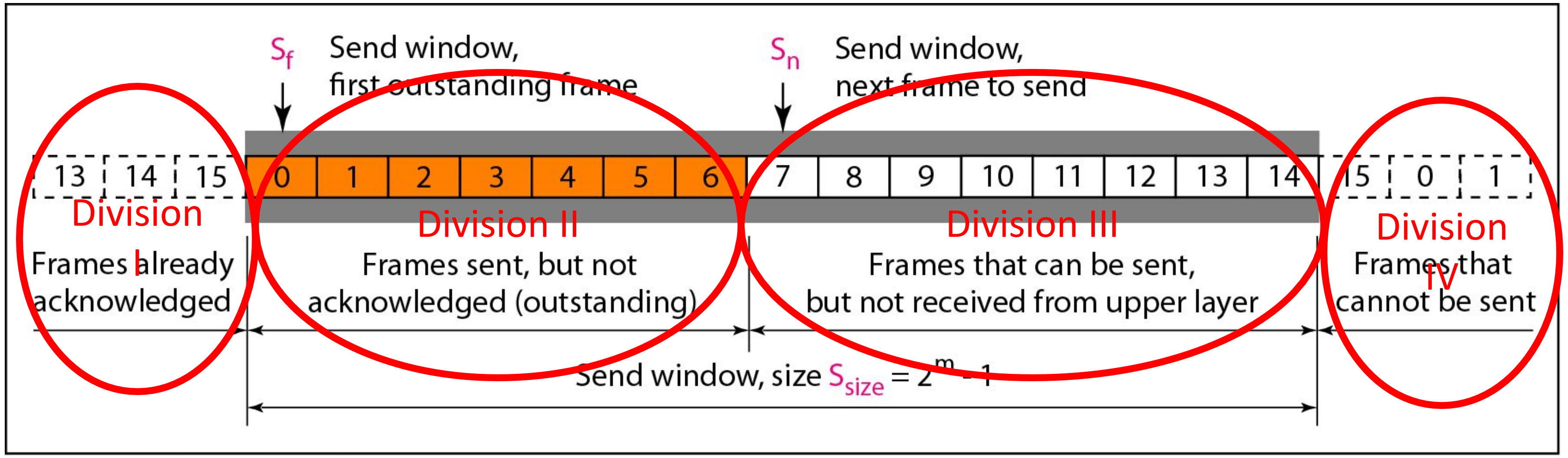
GBN Windows

- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}
  - Covers the sequence numbers of the data packets that can be in transit or can be sent
  - The send window can slide one or more slots when a valid acknowledgment arrives
    - Ack number ahould be greater than or equal to S_f and less than S_n arrives
- The receive window is an abstract concept defining an imaginary box   of size 1 with one single variable R_n
  - The window slides when a correct frame has arrived; sliding occurs one slot at a time $R_n = R_n + 1$
    - Out-of order packet (Region I and III in receiver window packet) is discarded need to be resend

Send window for Go-Back-N ARQ

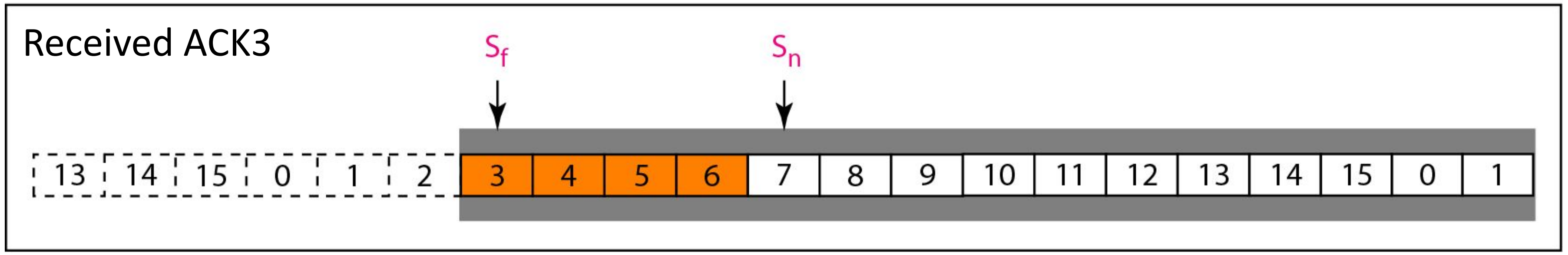
Send window for Go-Back-N ARQ

- Possible Sequence number (m=4) division – four and Sliding window size = 2^4-1 = 15



S_f Send window, first outstanding frame
S_n Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Division I
Frames already acknowledged
Division II
Frames sent, but not acknowledged (outstanding)
Division III
Frames that can be sent, but not received from upper layer
Division IV
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$

a. Send window before sliding

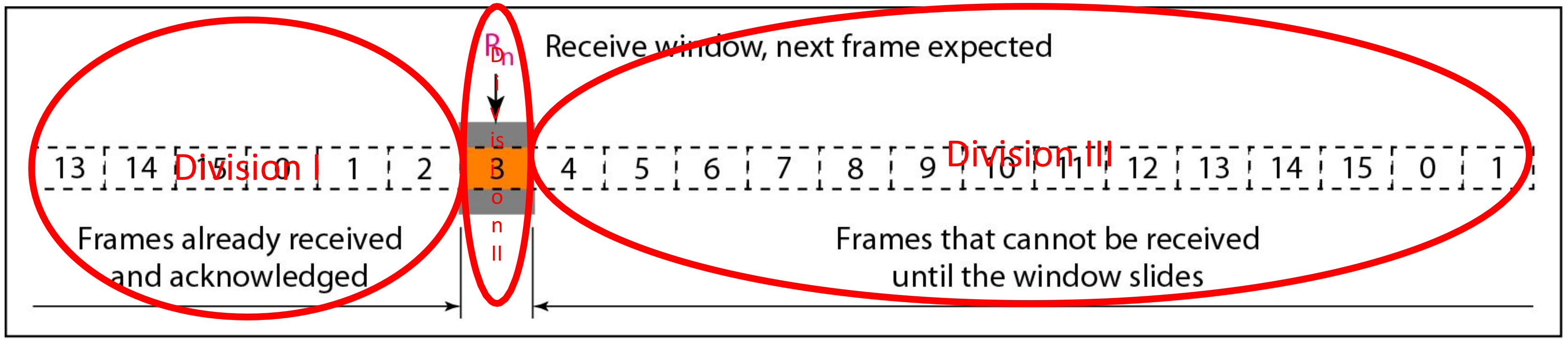


b. Send window after sliding

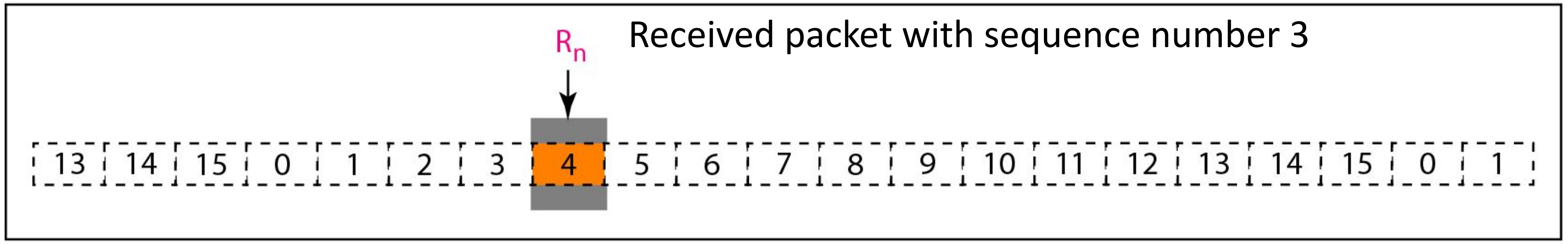
Receive window for Go-Back-N ARQ

Receive window for Go-Back-N ARQ

- Possible ACK number division – three and Sliding window size = 1


R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Division I
Division II
Division III
Frames already received and acknowledged
Frames that cannot be received until the window slides

a. Receive window



b. Window after sliding

Timer for sent frame

Timer for sent frame

- Multiple frame sent but only one timer (for S_f) used
- As first outstanding packet always expire first
- If S_f timer expires all outstanding packets are represent that is why GBN
- On a time-out the machine goes back N-locations and resents all packets

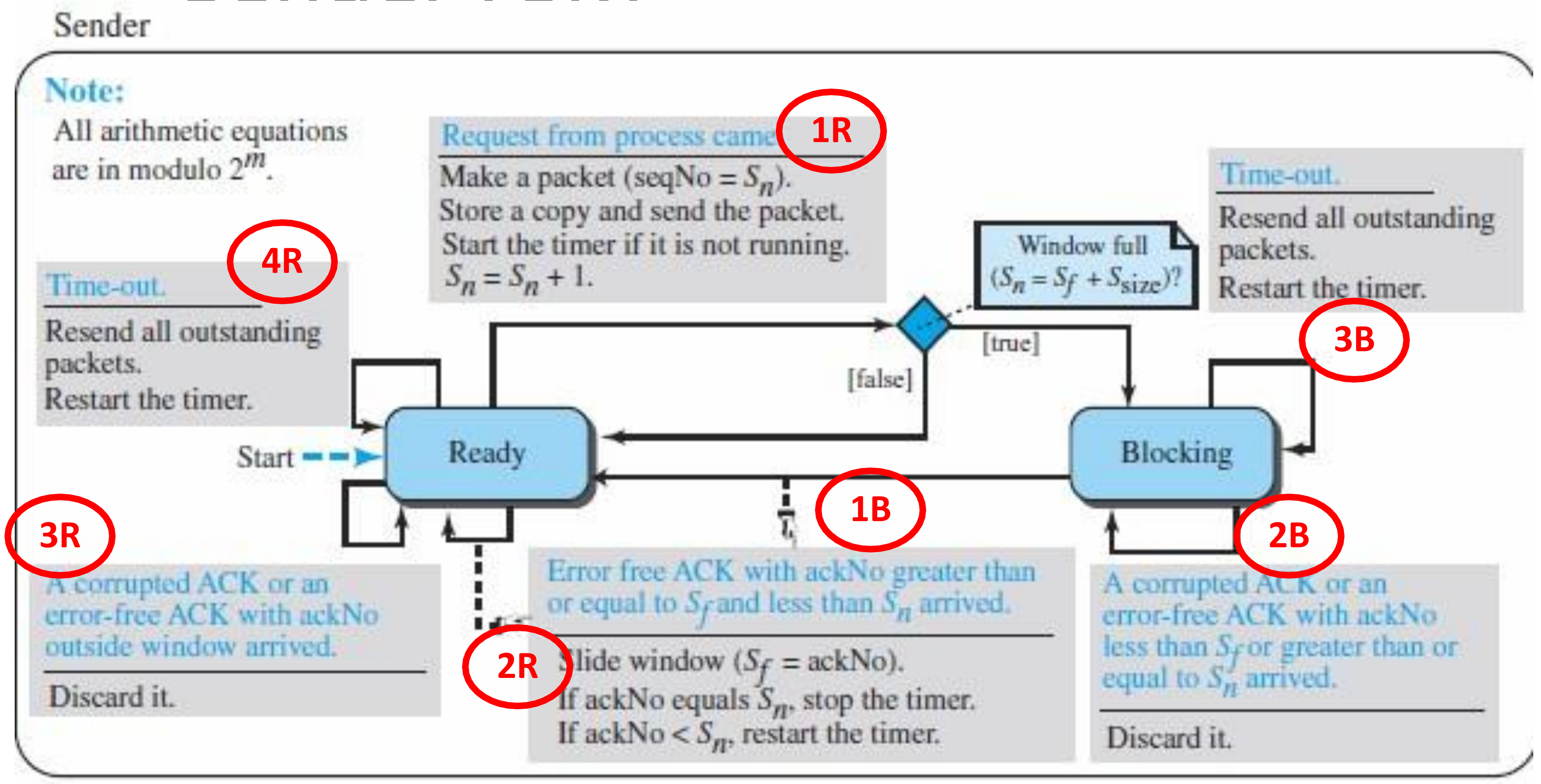
FSMs for the GBN protocol

FSMs for the GBN protocol

- Modulo 2^m arithmetic
- Sender FSM: two states
  - ready 4-events can occur numbered as 1R,2R, 3R, 4R
  - Blocking 3-events may occur numbered as 1B,2B, 3B
- The sender starts in the ready state
  - *The two variables are normally initialized to 0 (Sf = Sn = 0)*

Sender FSM

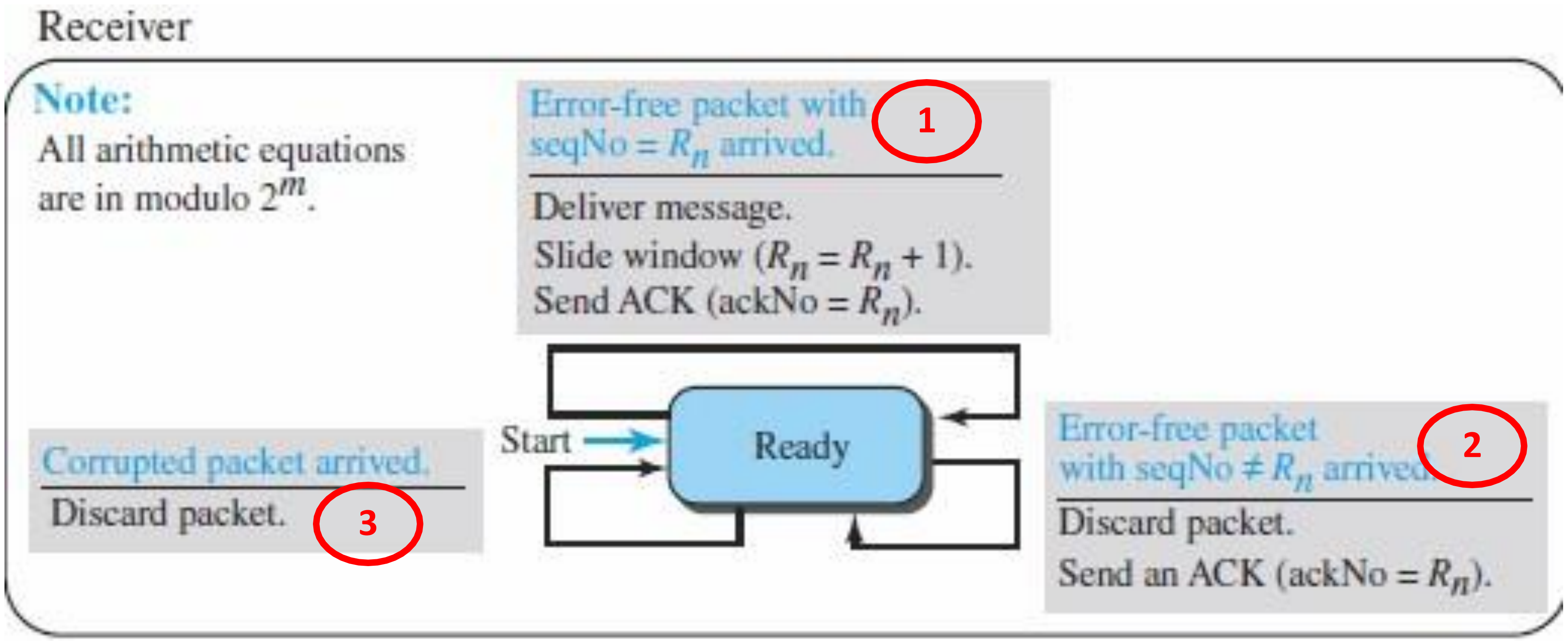
Sender FSM


Sender
Note:
All arithmetic equations are in modulo 2^m.
1R
Request from process came.
Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer if it is not running.
$S_n = S_n + 1$.
4R
Time-out.
Resend all outstanding packets.
Restart the timer.
Window full ($S_n = S_f + S_{size}$)?
[true]
[false]
Time-out.
Resend all outstanding packets.
Restart the timer.
3B
Start
Ready
Blocking
1B
3R
A corrupted ACK or an error-free ACK with ackNo outside window arrived.
Discard it.
2R
Error free ACK with ackNo greater than or equal to S_f and less than S_n arrived.
Slide window (S_f = ackNo).
If ackNo equals S_n, stop the timer.
If ackNo < S_n, restart the timer.
2B
A corrupted ACK or an error-free ACK with ackNo less than S_f or greater than or equal to S_n arrived.
Discard it.

Receiver FSM

Receiver FSM

- R_n = 0 initially
- Only one state 3 states


Receiver
Note:
All arithmetic equations are in modulo 2^m.
Error-free packet with seqNo = R_n arrived.
1
Deliver message.
Slide window ($R_n = R_n + 1$).
Send ACK (ackNo = R_n).
Start
Ready
Corrupted packet arrived.
Discard packet.
3
Error-free packet with seqNo ≠ R_n arrived.
2
Discard packet.
Send an ACK (ackNo = R_n).

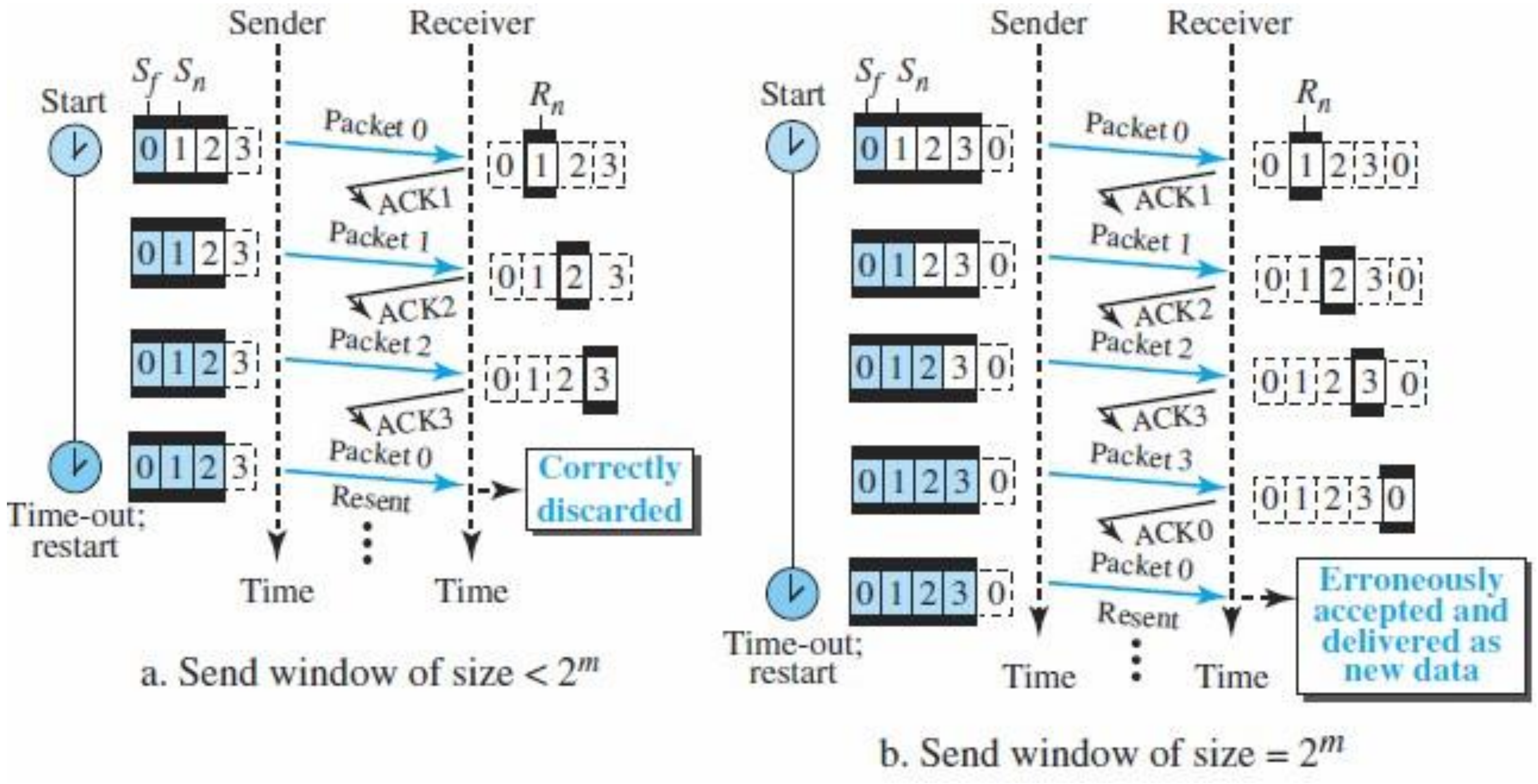
Size of send window

Size of send window

- Why the size of send window must be less than 2^m?
  - For m=2, window size < 2^m = 3
    - If all 3 - ACK lost, timer expires and all three packets are resend
  - For m=2, window size = 2^m = 4
    - If all 4 - ACK lost, timer expires and all four packets are resend which are accepted as new data erroneously even through duplicate
- **Hence *the size of the send window must be less than 2^m;* the size of the receive window is always 1**

Size of send window

Size of send window



a. Send window of size $< 2^m$

b. Send window of size $= 2^m$

GBN Analysis

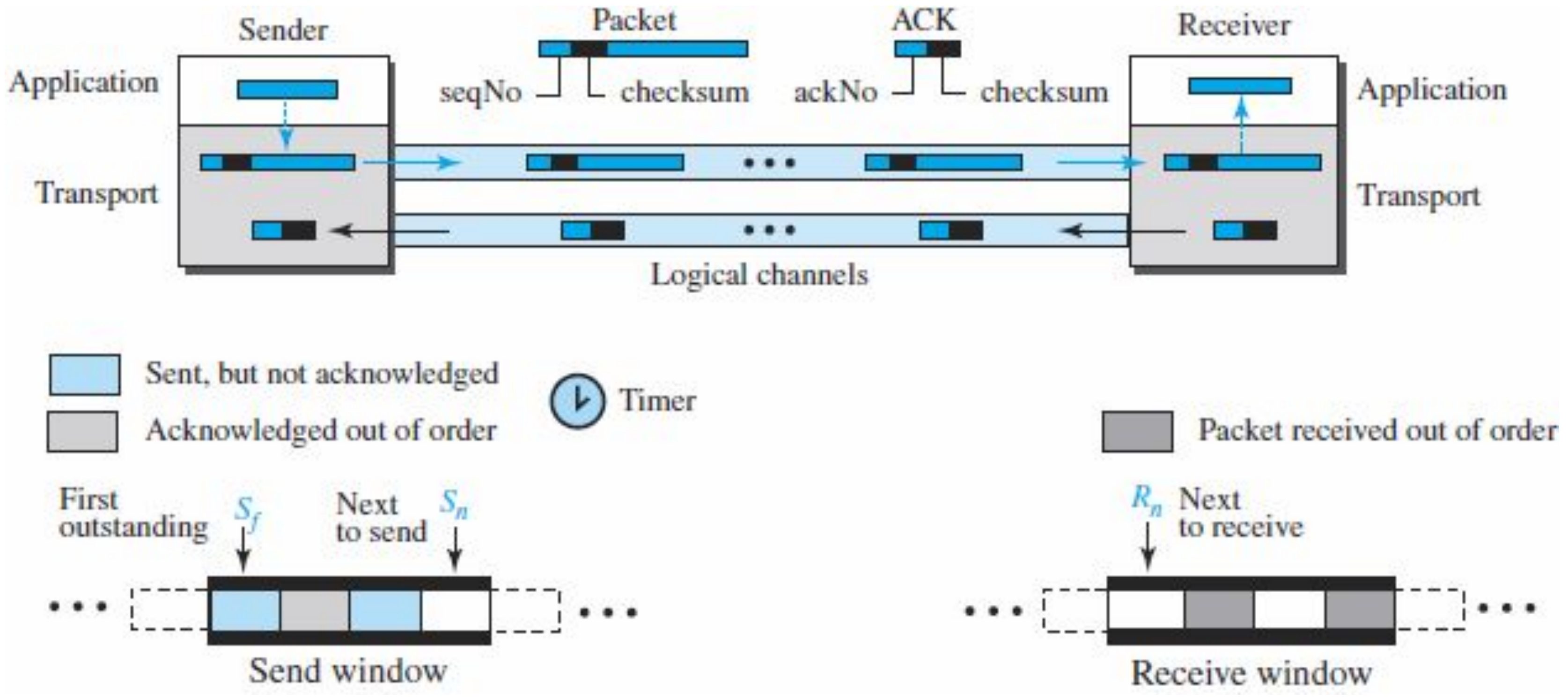
GBN Analysis

- Advantage: Simplifies the process at the receiver
  - Receiver keeps track of only one variable
  - No need to buffer out-of-order packets
    - simply discarded
- Disadvantage: Inefficient if the underlying network protocol loses a lot of packets
  - Each time a single packet is lost or corrupted, the sender resends all outstanding packets, even though some of these packets may have been received safe and sound but out of order
  - If the network layer is losing many packets because of congestion in the network, the resending of all of these outstanding packets makes the congestion worse, and eventually more packets are lost
  - This has an avalanche effect that may result in the total collapse of the network
    - Solution Selective Repeat

Selective-Repeat (SR) protocol

Selective-Repeat (SR) protocol

- Name implies, resends only selective packets, those that are actually lost



SR protocol windows

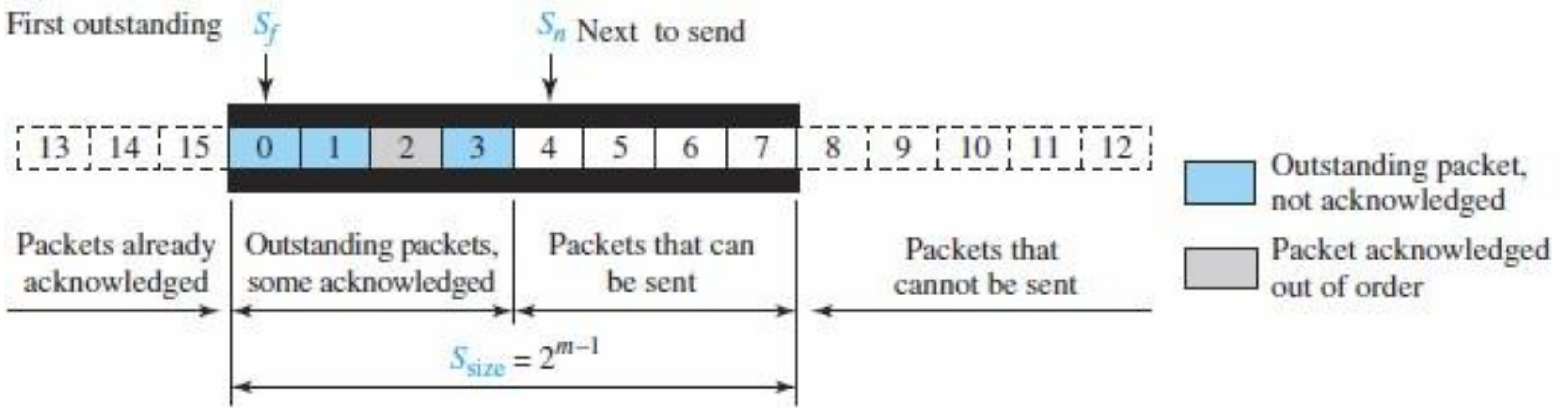
SR protocol windows

- two windows: a send window and a receive window

SR protocol windows	**GBN protocol windows**
Maximum size of the send window is much smaller (maximum 2^m-1)	*The send window is* 2^m-1
The receive window is the same size as the send window (maximum 2^m-1)	*The receive window is 1*
if m = 4, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8	*if m = 4, the sequence* numbers go from 0 to 15 the size of the window is 15

Send window for SR protocol

Send window for SR protocol

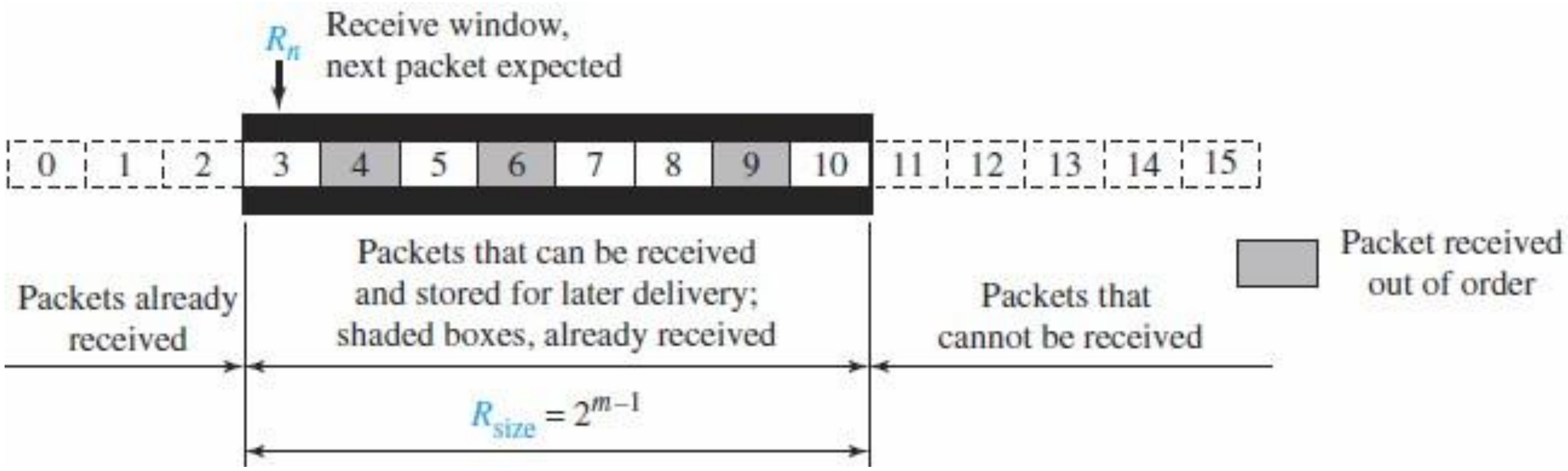

First outstanding
S_f
S_n Next to send
13
14
15
0
1
2
3
4
5
6
7
8
9
10
11
12
Packets already acknowledged
Outstanding packets, some acknowledged
Packets that can be sent
Packets that cannot be sent
$S_{size} = 2^{m-1}$
Outstanding packet, not acknowledged
Packet acknowledged out of order

- Allows as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer
- Because the sizes of the send window and receive window are the same, all the packets in the send packet can arrive out of order and be stored until they can be delivered
- A reliable protocol never delivers packets out of order to the application layer

Receive window for SR protocol

Receive window for SR protocol

- Shaded slots inside the window define packets that have arrived out of order and are waiting for the earlier transmitted packet to arrive before delivery to the application layer


R_n
Receive window, next packet expected
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Packets already received
Packets that can be received and stored for later delivery; shaded boxes, already received
Packets that cannot be received
Packet received out of order
$R_{size} = 2^{m-1}$

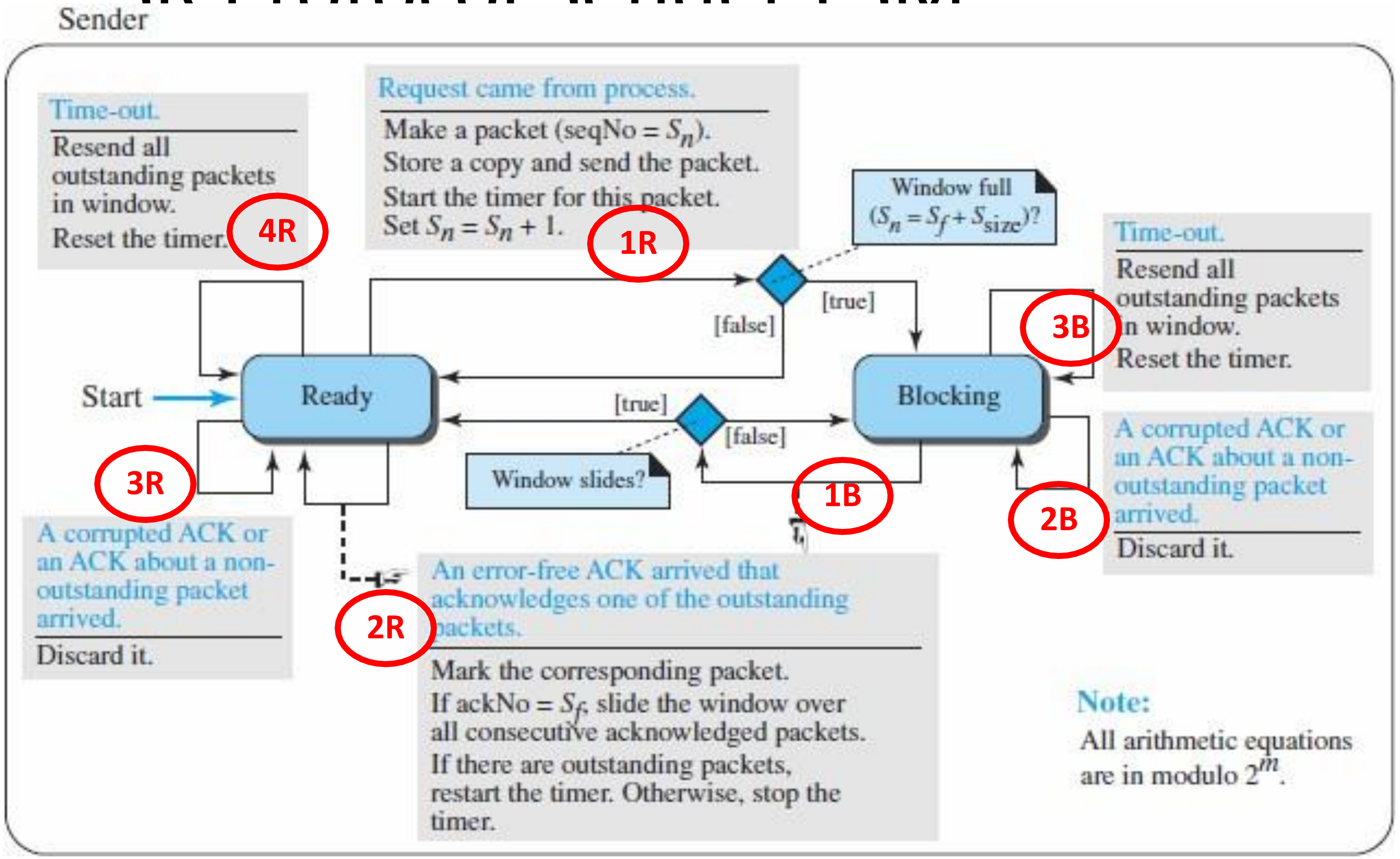
SR protocol *Timer and Acknowledgements*

- **Timers:**
- Theoretically, one timer for each outstanding packet
- When a timer expires, only the corresponding packet is resent
- GBN treats outstanding packets as a group; SR treats them individually
- However, most transport-layer protocols that implement SR use only a single timer

- **Acknowledgements:**
- GBN an ackNo is cumulative (defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound)
- The semantics of acknowledgment is different in SR
  - An ackNo defines the sequence number of a single packet that is received safe and sound; there is no feedback for any other

SR Protocol Sender FSM

SR Protocol Sender FSM



Time-out.

Resend all outstanding packets in window.
Reset the timer.

Request came from process.

Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer for this packet.
Set $S_n = S_n + 1$.

Time-out.

Resend all outstanding packets in window.
Reset the timer.

A corrupted ACK or an ACK about a non-outstanding packet arrived.

Discard it.

A corrupted ACK or an ACK about a non-outstanding packet arrived.

Discard it.

An error-free ACK arrived that acknowledges one of the outstanding packets.

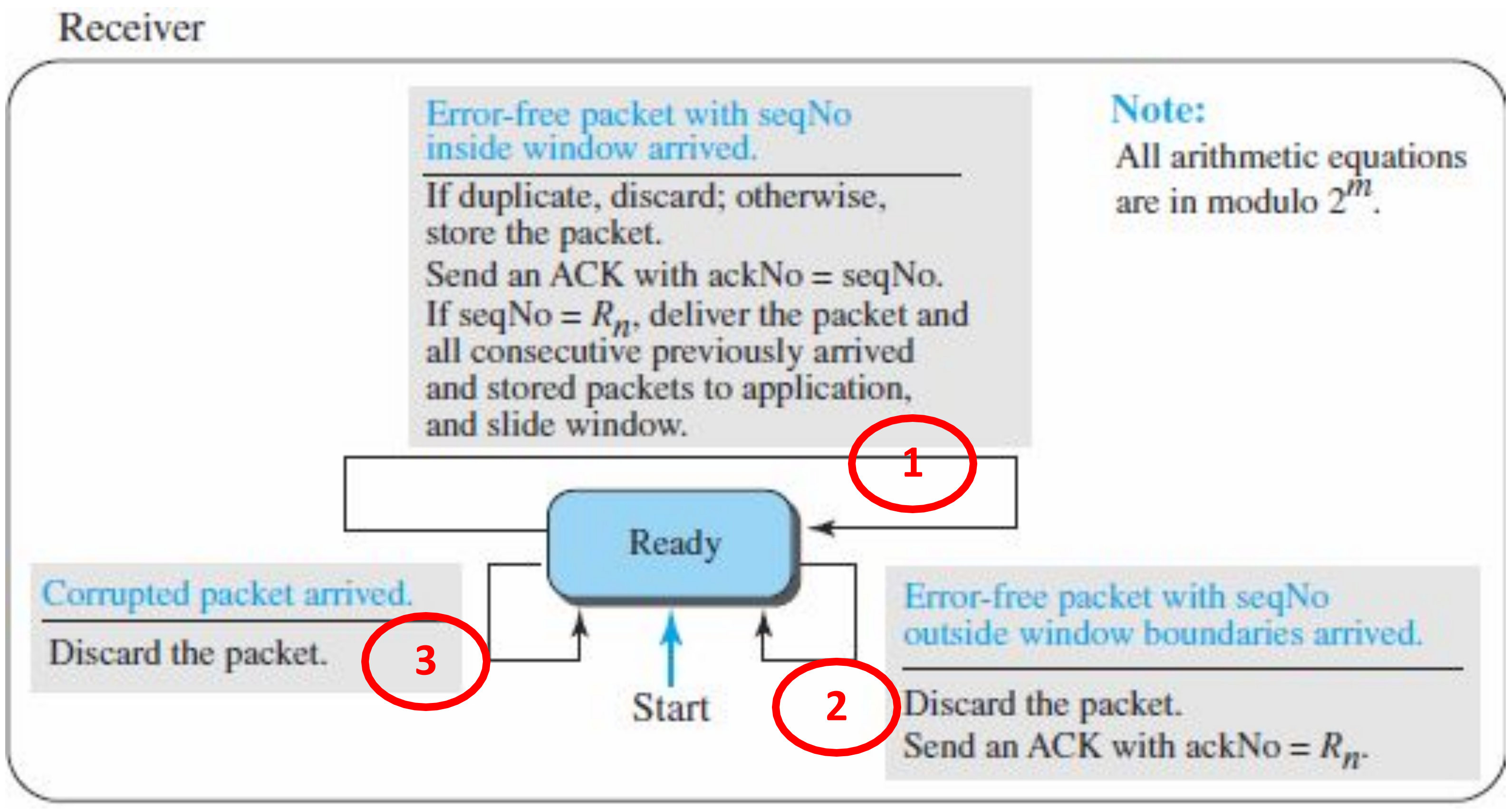
Mark the corresponding packet.
If ackNo = S_f, slide the window over all consecutive acknowledged packets.
If there are outstanding packets, restart the timer. Otherwise, stop the timer.

Note:
All arithmetic equations are in modulo 2^m.

SR Protocol Receiver FSM

SR Protocol Receiver FSM

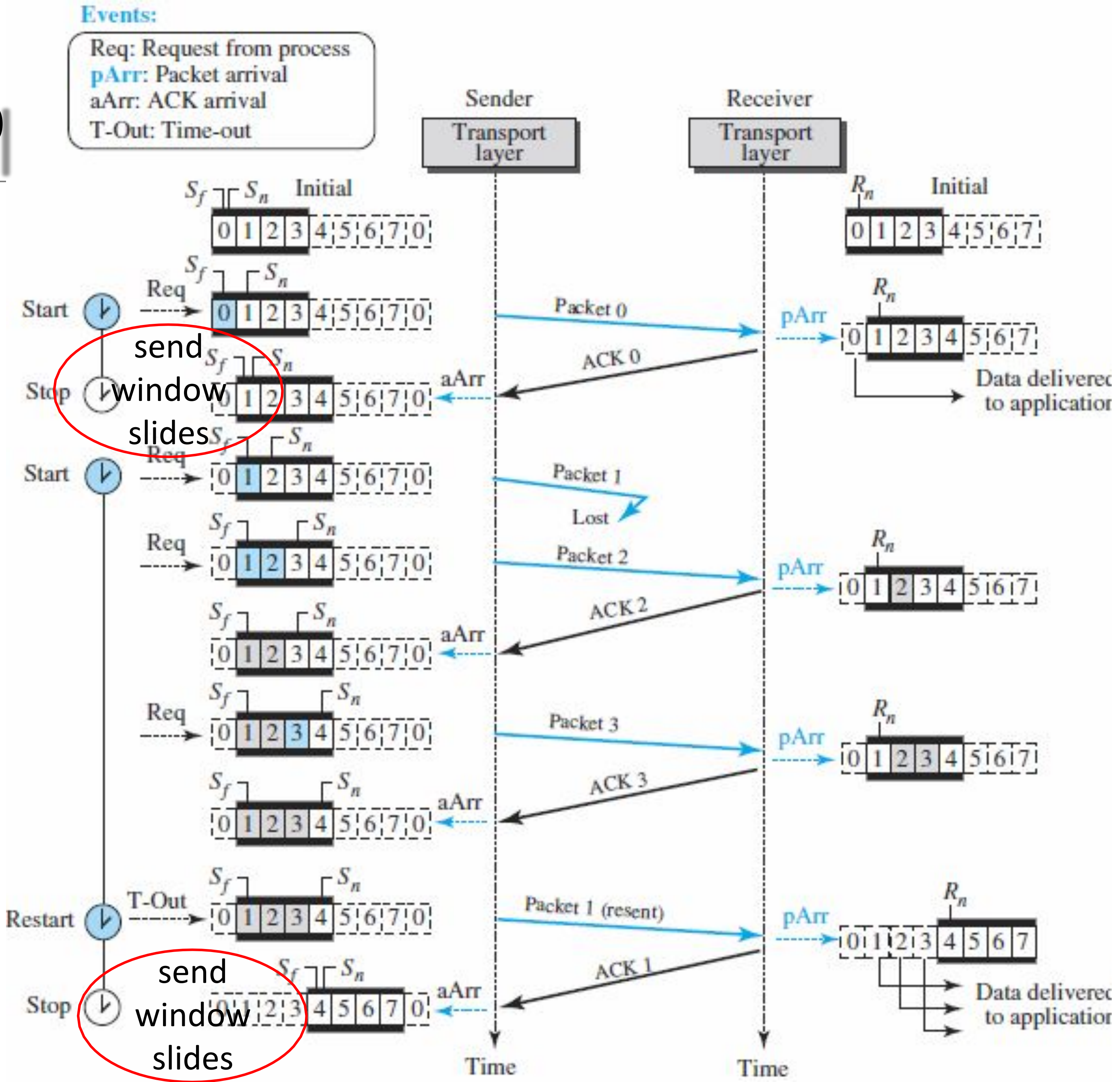
- always in the *ready state*


Receiver
Error-free packet with seqNo inside window arrived.
If duplicate, discard; otherwise, store the packet.
Send an ACK with ackNo = seqNo.
If seqNo = R_n, deliver the packet and all consecutive previously arrived and stored packets to application, and slide window.
Note:
All arithmetic equations are in modulo 2^m.
1
Ready
Corrupted packet arrived.
Discard the packet.
3
Start
2
Error-free packet with seqNo outside window boundaries arrived.
Discard the packet.
Send an ACK with ackNo = R_n.

SR Protocol

SR Protocol

- Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost.



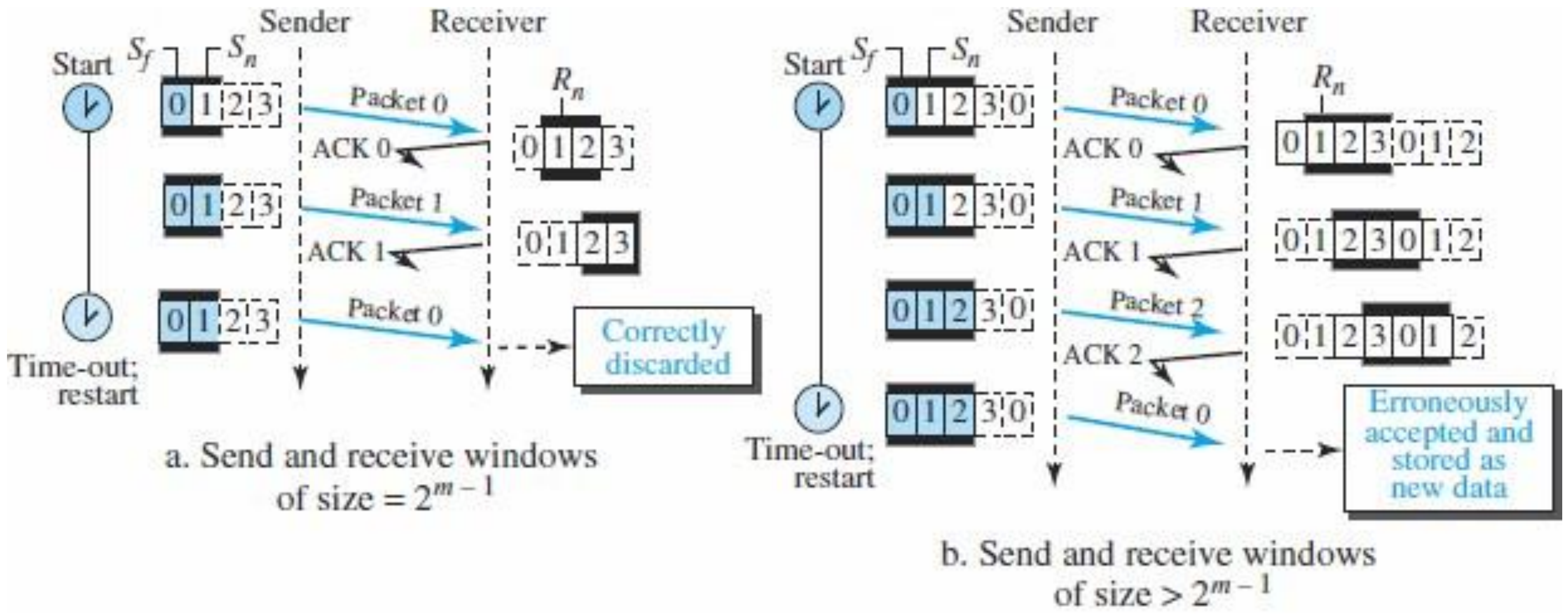
SR Protocol *Window Sizes*

SR Protocol *Window Sizes*

- Why the size of the sender and receiver windows can be at most one-half of 2^m?
- *For m = 2, size of the window is* $2^m/2$ *or* $2^{(m-1)} = 2$
- If the size of the window is 2 and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window)
- When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of packet 0
  - However, this time, the window of the receiver expects to receive packet 0 (0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle
  - This is clearly an error

SR Protocol *Window Sizes*

SR Protocol *Window Sizes*



a. Send and receive windows of size = 2^{m-1}

b. Send and receive windows of size > 2^{m-1}

- The size of the sender and receiver window must be at most one-half of 2^m