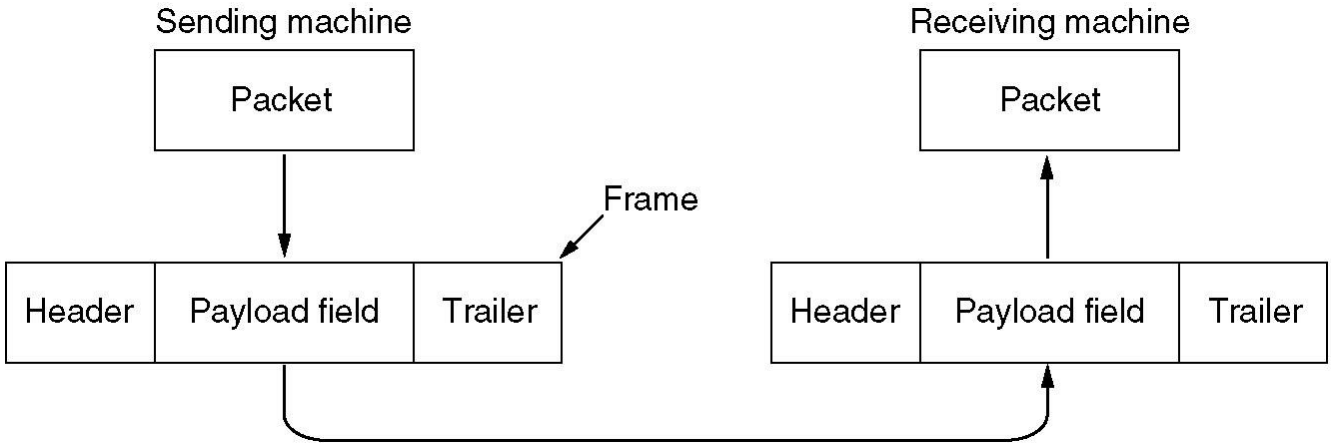
Data Link Layer Design Issues

Data link layer functions includes;

1. Providing a well defined service interface to the network layer.
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast sender.

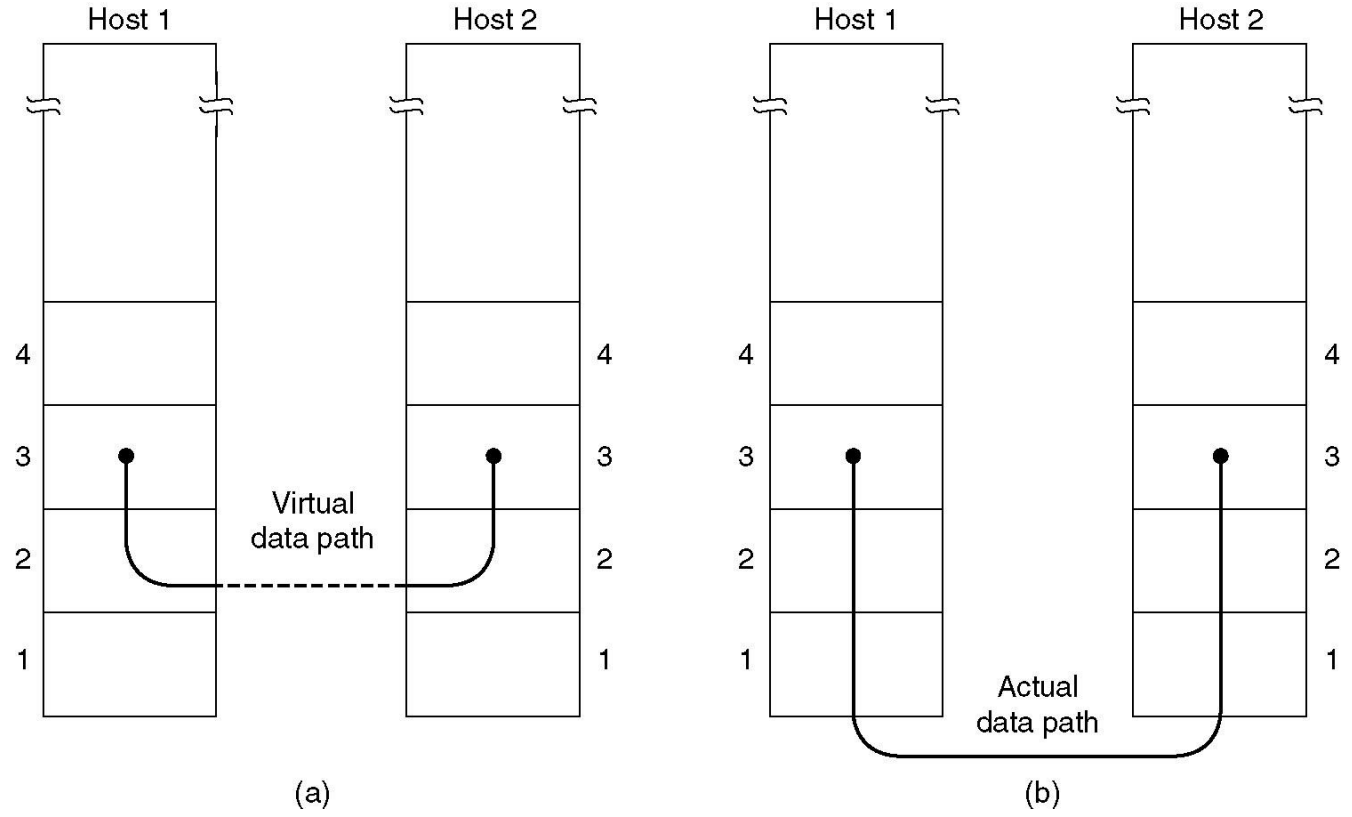


Relationship between packets and frames

Data Link Layer Design Issues

1. Services provided to Network Layers:

- From network layer on source to network layer on receiver.



(a) Virtual communication. (b) Actual communication.

Data Link Layer Design Issues

1. Services provided to Network Layers:

- Different services provided:
    i. **Unacknowledged Connectionless Service.**
    ii. **Acknowledged Connectionless Service**
    iii. **Acknowledged Connection-Oriented Service**

Data Link Layer Design Issues

i. Unacknowledged Connectionless Service.

- Independent Frames without acknowledgement.
- No Logical connection Established.
- No detection of loss and recovery from loss.
- Appropriate when error rate is very low.
- Appropriate for real time traffic, where late data are worse than bad data.
- Ex. LAN

Data Link Layer Design Issues

ii. Acknowledged Connectionless Service

- Add Reliability to previous approach.
- No logical connection defined.
- Individual Acknowledgement.
- Retransmission after specific time interval.
- Ex. Wireless Systems.

Data Link Layer Design Issues

iii. Acknowledged Connection-Oriented Service

- Most sophisticated service.
- Connection establishment before transmission.
- Numbered frame.
- Provide reliable communication.
- Provides ordered delivery.

Framing

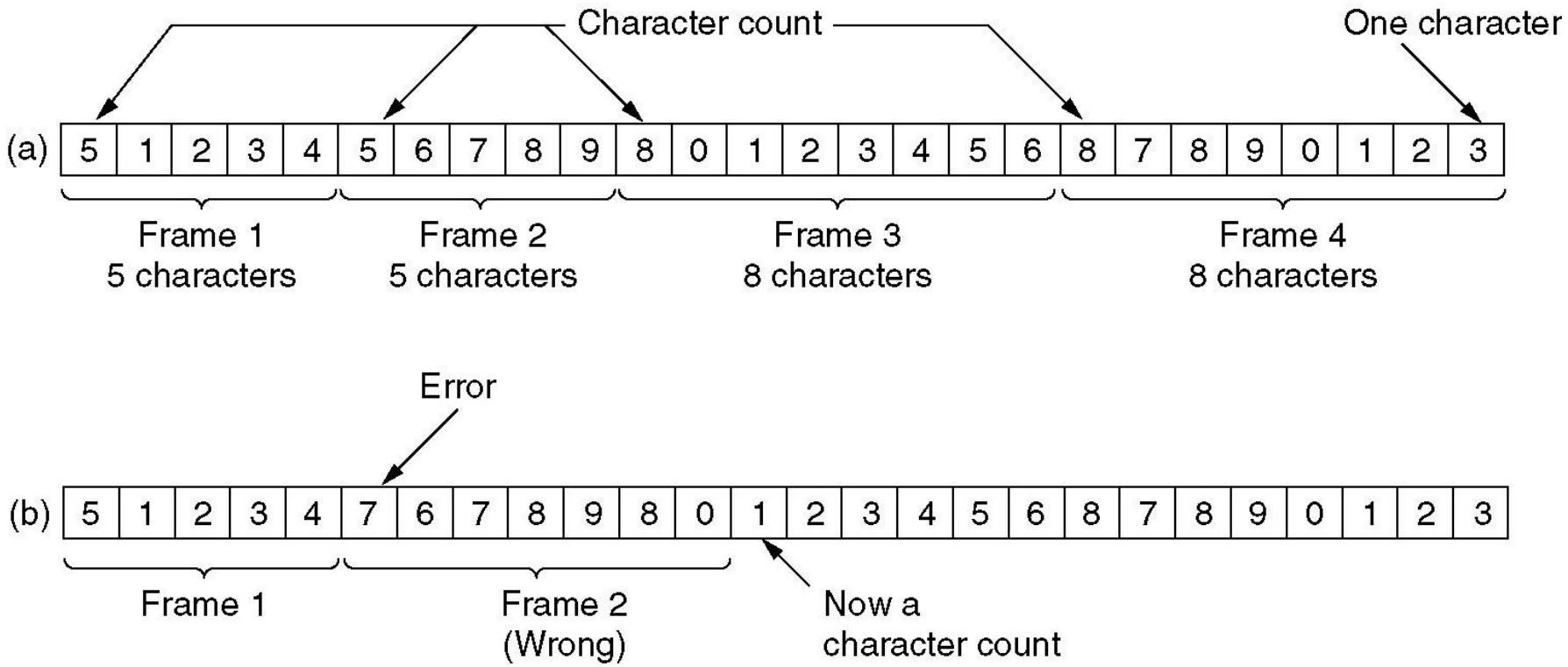
Framing

Methods to mark the starting and ending of frame.

1. Character Count.
2. Flag Bytes with **Byte Stuffing**.
3. Starting and ending flags, with **Bit Stuffing**.

Framing

1. Character Count



A character stream. (a) Without errors. (b) With one error.

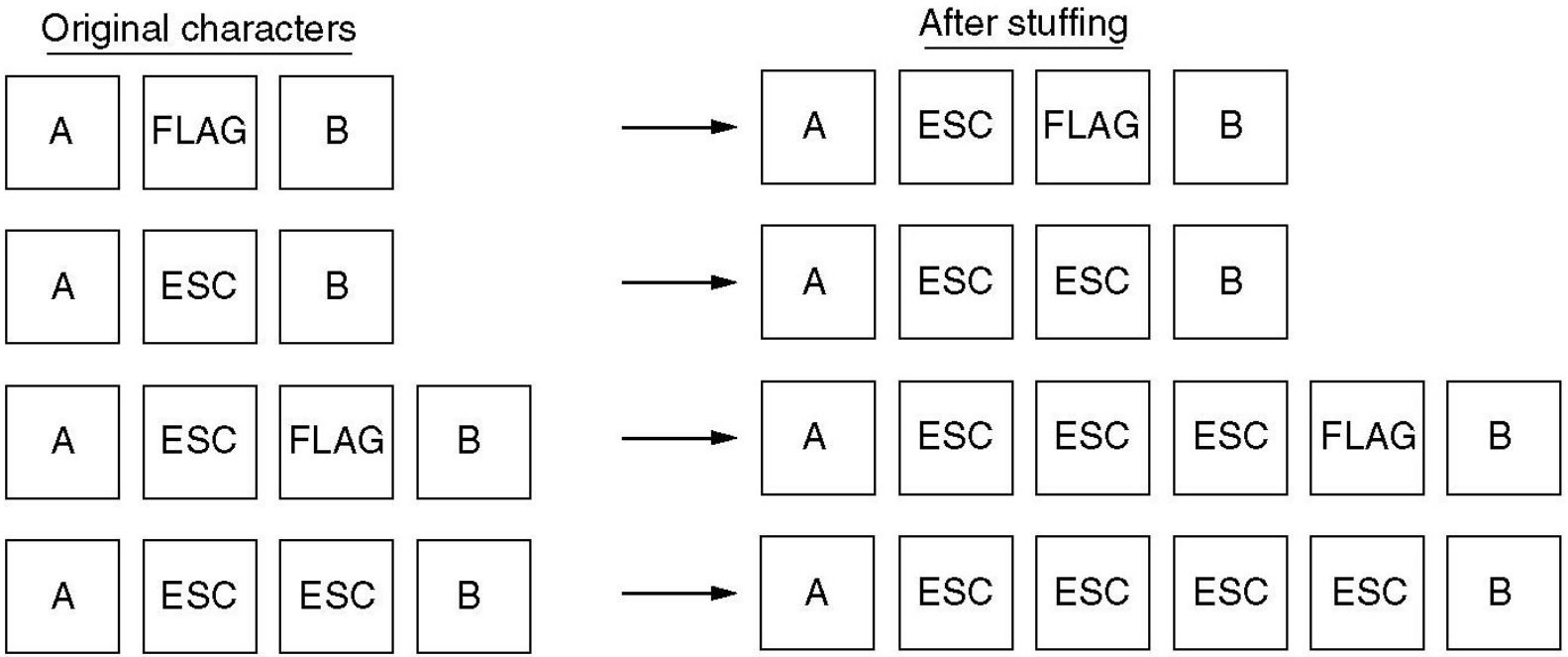
Difficult synchronization after transmission errors.

Framing

2. Flag Bytes with Byte Stuffing.

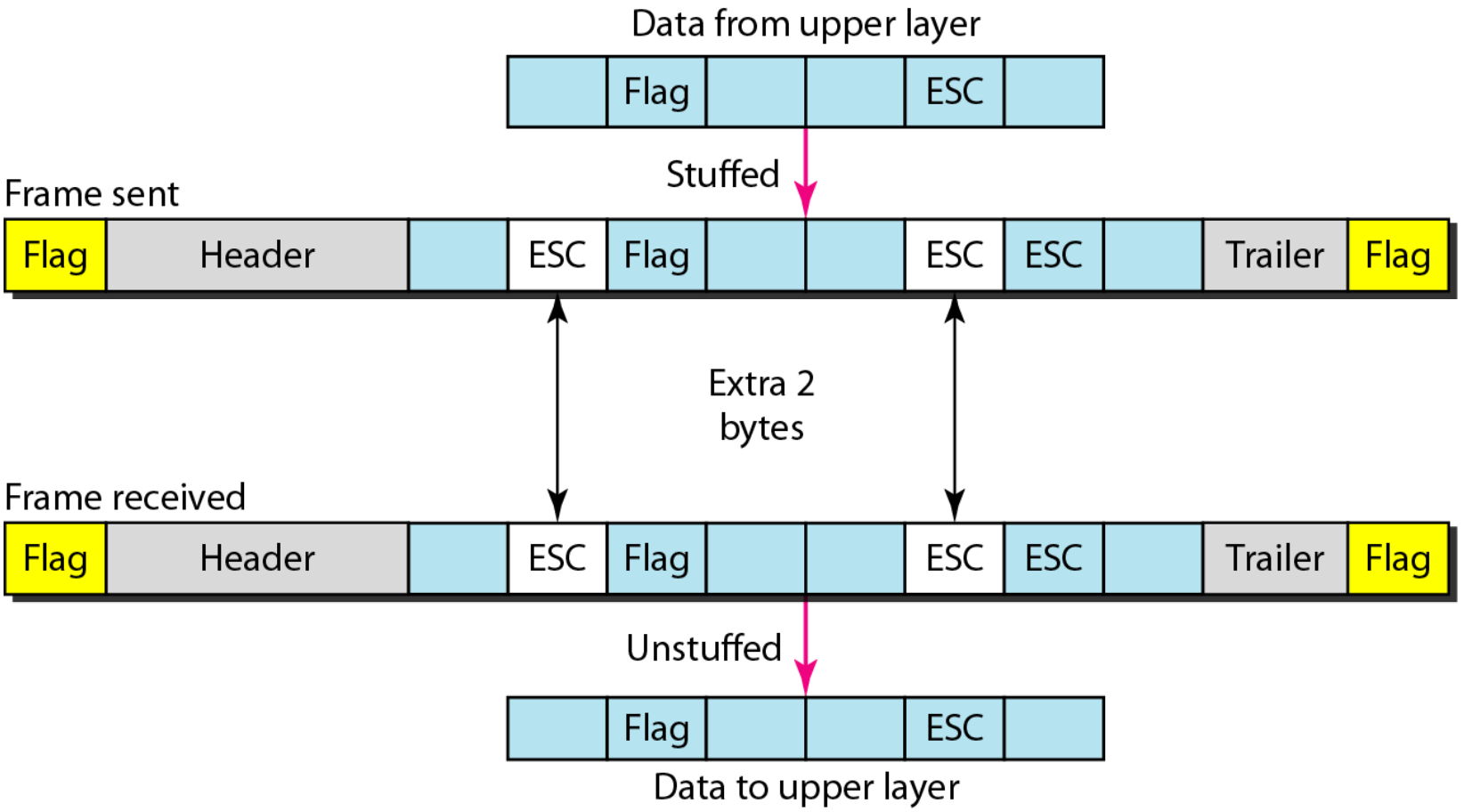
(a)

A frame delimited by flag bytes.



Framing

2. Flag Bytes with Byte Stuffing.



Framing

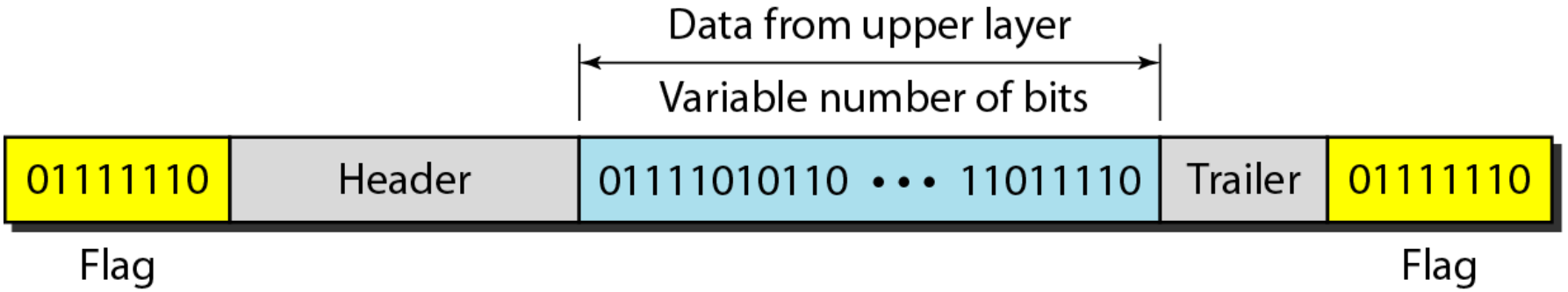
2. Flag Bytes with Byte Stuffing.

Note

Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

Framing

3. Starting and ending flags, with Bit Stuffing



Framing

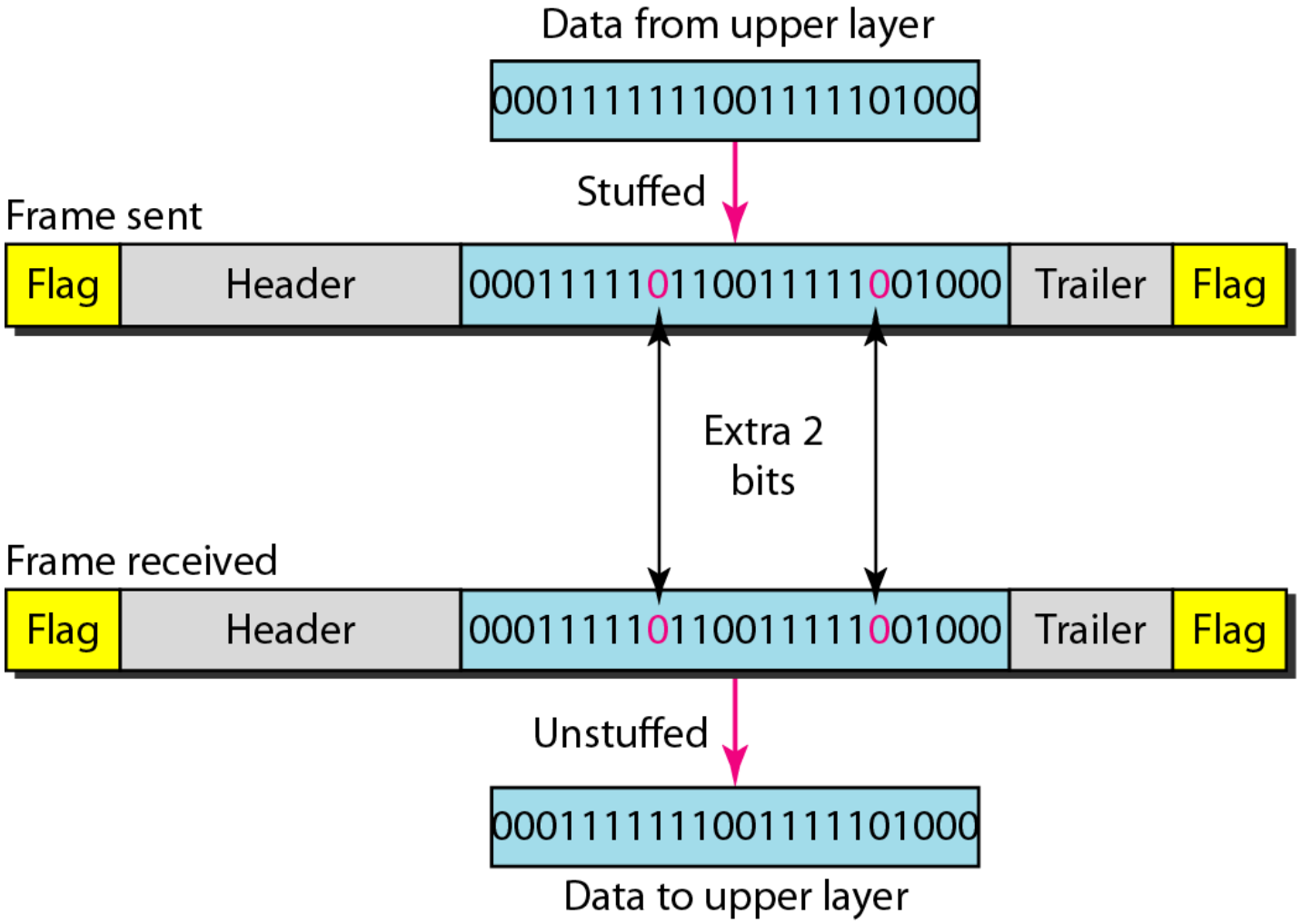
3. Starting and ending flags, with Bit Stuffing

Note

Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.

Framing

3. Starting and ending flags, with Bit Stuffing



Framing

3. Starting and ending flags, with Bit Stuffing

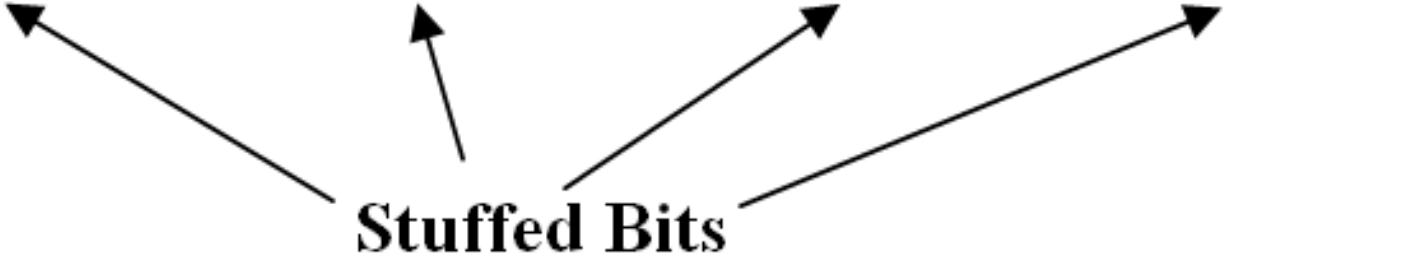
Bit Stuffing Example

Perform bit stuffing for the following bit pattern.
0001111111100111111000101111101101111110110

Solution

0001111111100111111000101111101101111110110

00011111**0**1110011111**0**0001011111**0**0110111111**0**0110



Framing

Example

The following character encoding is used in a data link layer protocol.

A: 01000111 B: 11100011 Flag: 01111110 Esc: 11100000

Show the bit sequence transmitted (in binary) for the following character frame:

A B ESC Flag

When the following protocols is being used,

1) **Flag byte with byte stuffing.**
2) **Starting and ending flag bytes, with bit stuffing**

Error Control

Error Detection and Correction

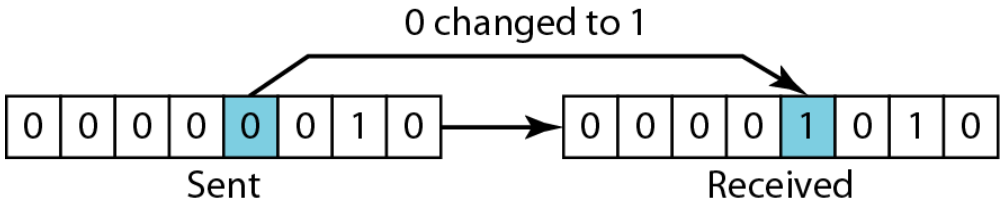
- Data can be corrupted during transmission.
- Errors can be detected and corrected.
- Some applications can tolerate the errors while some needs accurate transmission without errors.
- Let us first discuss some issues related, directly or indirectly, to error detection and correction.

    1. Types of Errors
    2. Redundancy
    3. Detection Versus Correction
    4. Forward Error Correction Versus Retransmission
    5. Coding

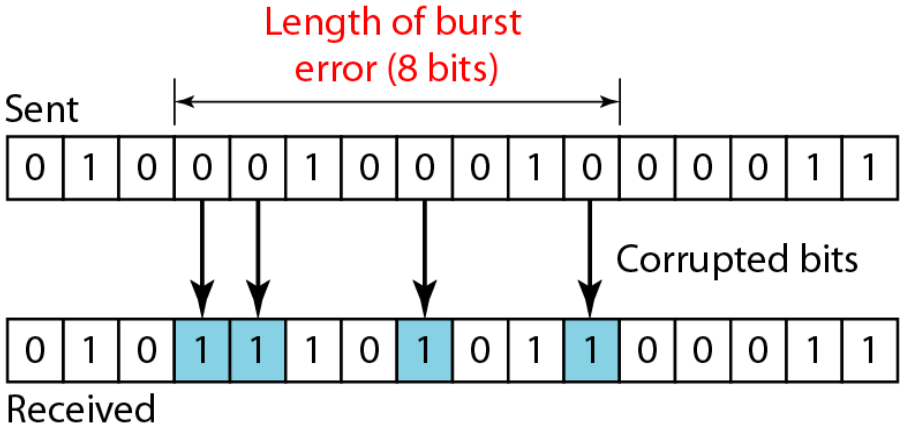
Error Detection and Correction

1. Types of Errors

- In a **single-bit error**, only 1 bit in the data unit has changed.



- A **burst error** means that 2 or more bits in the data unit have changed.



Error Detection and Correction

2. Redundancy

- To detect and correct the errors, some redundant bits will be send.
- Added by transmitter and removed by receiver.

3. Detection Versus Correction

- Correction more difficult than detection.
- In detection, Answer is yes/no.
- Correction extracts the number of bits corrupted plus their location.

4. Forward Error Correction Versus Retransmission

5. Coding

Note

In modulo-N arithmetic, we use only the integers in the range 0 to N −1, inclusive.

XORing of two single bits or two words

$0 \oplus 0 = 0$ $\qquad$ $1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

$0 \oplus 1 = 1$ $\qquad$ $1 \oplus 0 = 1$

b. Two bits are different, the result is 1.

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & 0 \\ \oplus & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1 & 0 \end{array}$$

c. Result of XORing two patterns

BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called datawords. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.

Figure 10.5 *Datawords and codewords in block coding*

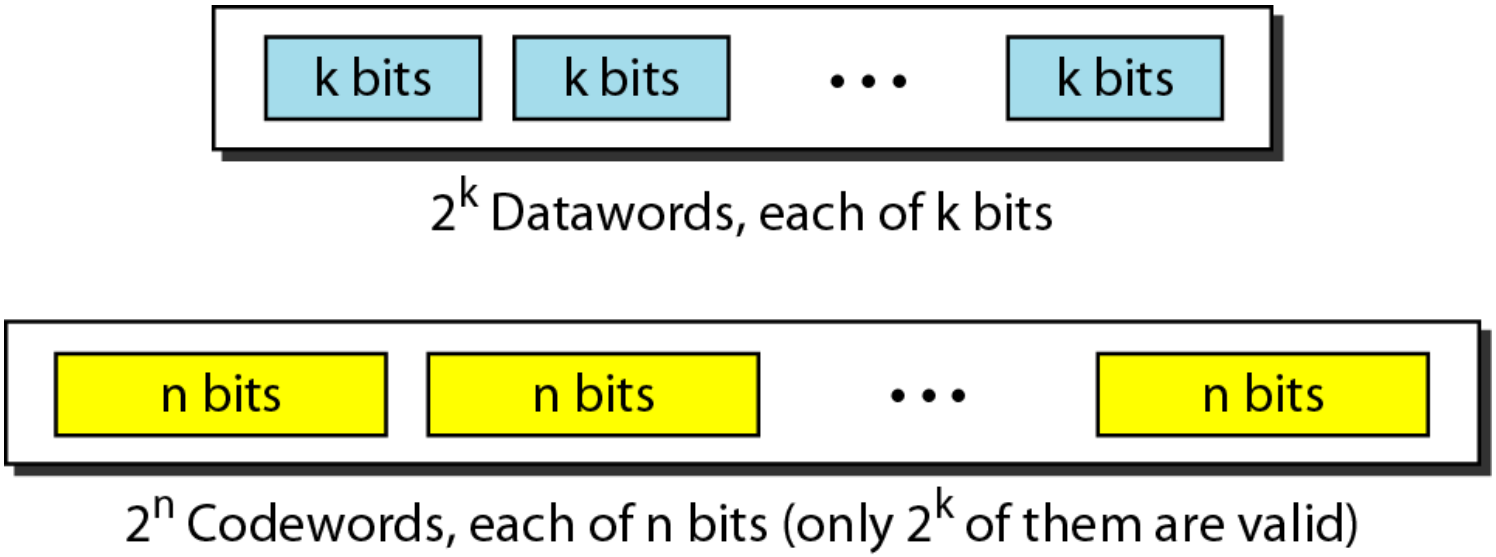


Figure 10.6 Process of error detection in block coding

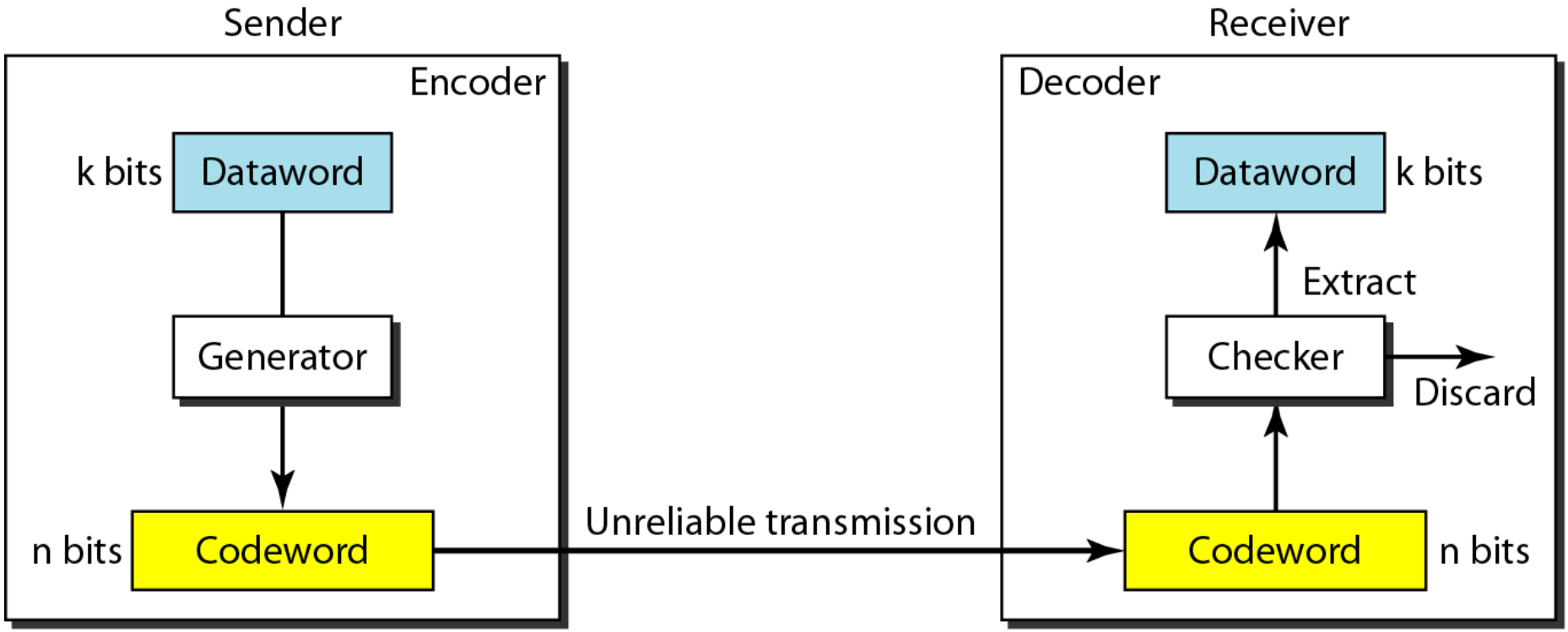


Table 10.1 ***A code for error detection (Example 10.2)***

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Table 10.2 ***A code for error correction (Example 10.3)***

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

10-3 LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called linear block codes.

A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Note

In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

Let us see if the two codes we defined in Table 10.1 belong to the class of linear block codes.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Error Correction Code

Hamming Code

- *The **Hamming distance** between two words is the number of differences between corresponding bits.*

- The Hamming distance d(000, 011) is 2 because,

$$000 \oplus 011 \text{ is } 011 \text{ (two 1s)}$$

- The Hamming distance d(10101, 11110) is 3 because,

$$10101 \oplus 11110 \text{ is } 01011 \text{ (three 1s)}$$

- *The **minimum Hamming distance** is the smallest Hamming distance between all possible pairs in a set of words.*

Error Correction Code

Hamming Code

- Find the minimum hamming distance for;

Scheme 1 :

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Solution

first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Error Correction Code

Hamming Code

- Find the minimum hamming distance for;

Scheme 2 :

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Solution

first find all Hamming distances.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Error Correction Code

Hamming Code:

Minimum hamming Distance for Error Detection:

- ***To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.***
- In Scheme 2, $d_{min} = 3$, So it can detect up to 2 errors.

Examples:

- **S (01011) -> R(11001)**

  {No such valid codeword, So, Detected}

- **S (01011) -> R(11110)**

  {Match with the valid codeword, So, Can't Detected}

Error Correction Code

Hamming Code:

Minimum hamming Distance for Error Correction:

- ***To guarantee the correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.***
- In Scheme 2, $d_{min} = 3$, So it can detect up to 1 errors.

Examples:

A code scheme has a Hamming distance $d_{min} = 4$. What is the error detection and correction capability of this scheme?

{ This code guarantees the detection of up to three errors, but it can correct up to one error. }

Figure 10.12 *The structure of the encoder and decoder for a Hamming code*

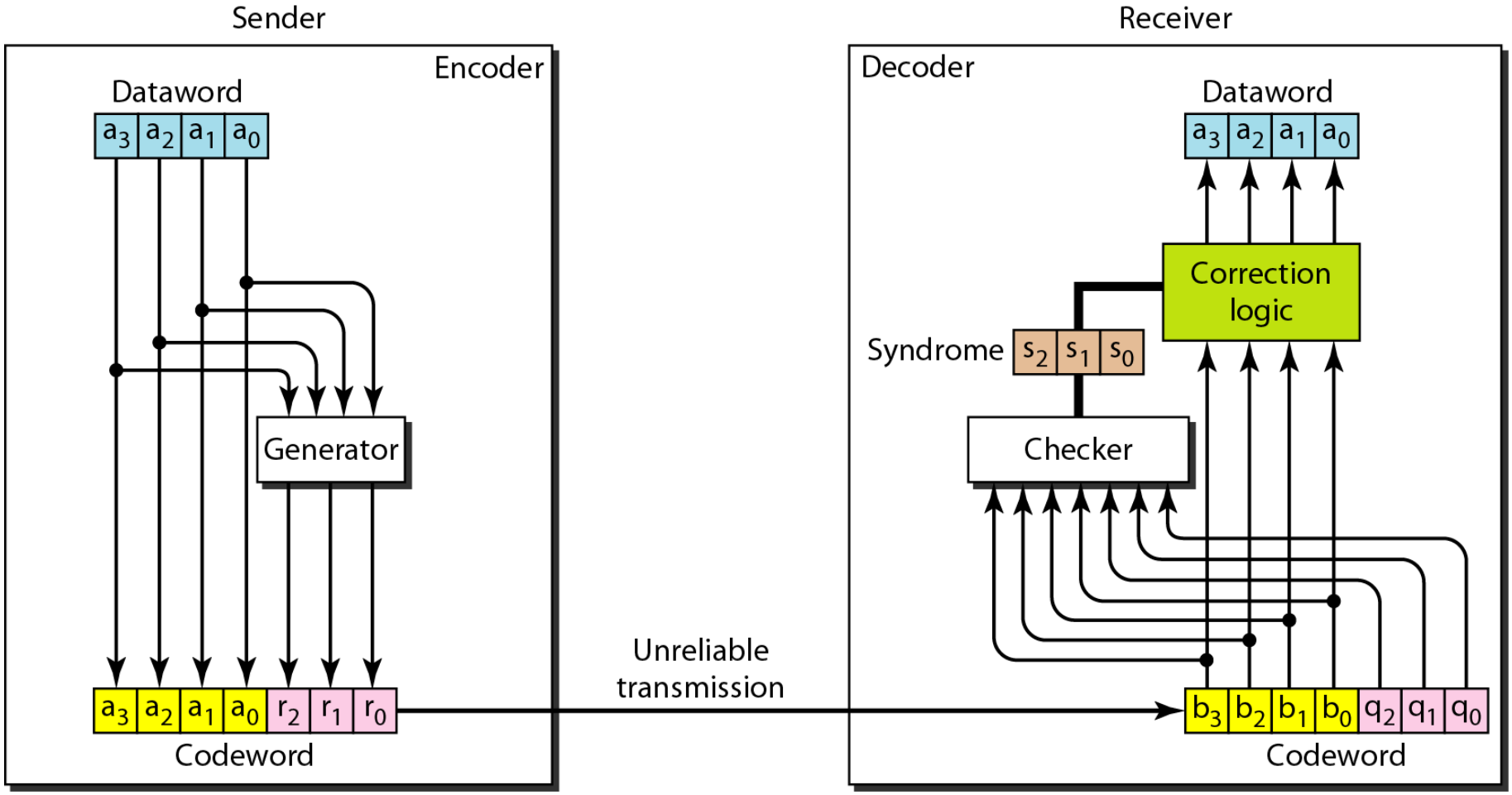


Table 10.4 Hamming code C(7, 4)

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Table 10.5 ***Logical decision made by the correction logic analyzer***

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received.***

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.***

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.***

2. ***The dataword 0111 becomes the codeword 0111001.The codeword 0011001 is received.***

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.***

2. ***The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b2 (changing the 1 to 0), the final dataword is 0111.***

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.***

2. ***The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b2 (changing the 1 to 0), the final dataword is 0111.***

3. ***The dataword 1101 becomes the codeword 1101000. The codeword 00010000 is received.***

Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. *The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.*

2. *The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b2 (changing the 1 to 0), the final dataword is 0111.*

3. *The dataword 1101 becomes the codeword 1101000. The syndrome is 101. After flipping b0, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.*

CYCLIC CODES

***Cyclic codes** are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.*

Table 10.6 A CRC code with C(7, 4)

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 10.14 *CRC encoder and decoder*

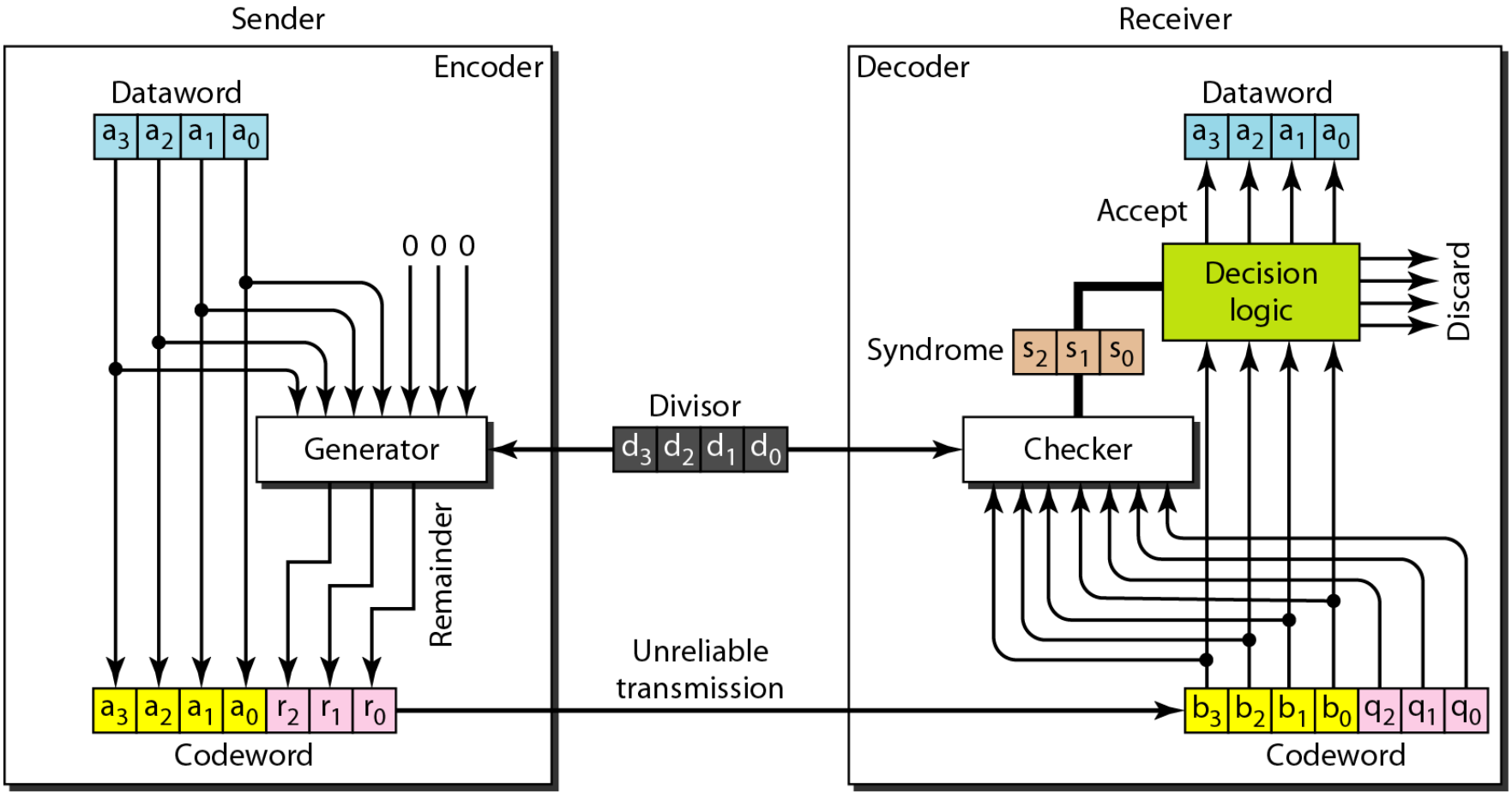


Figure 10.15 Division in CRC encoder

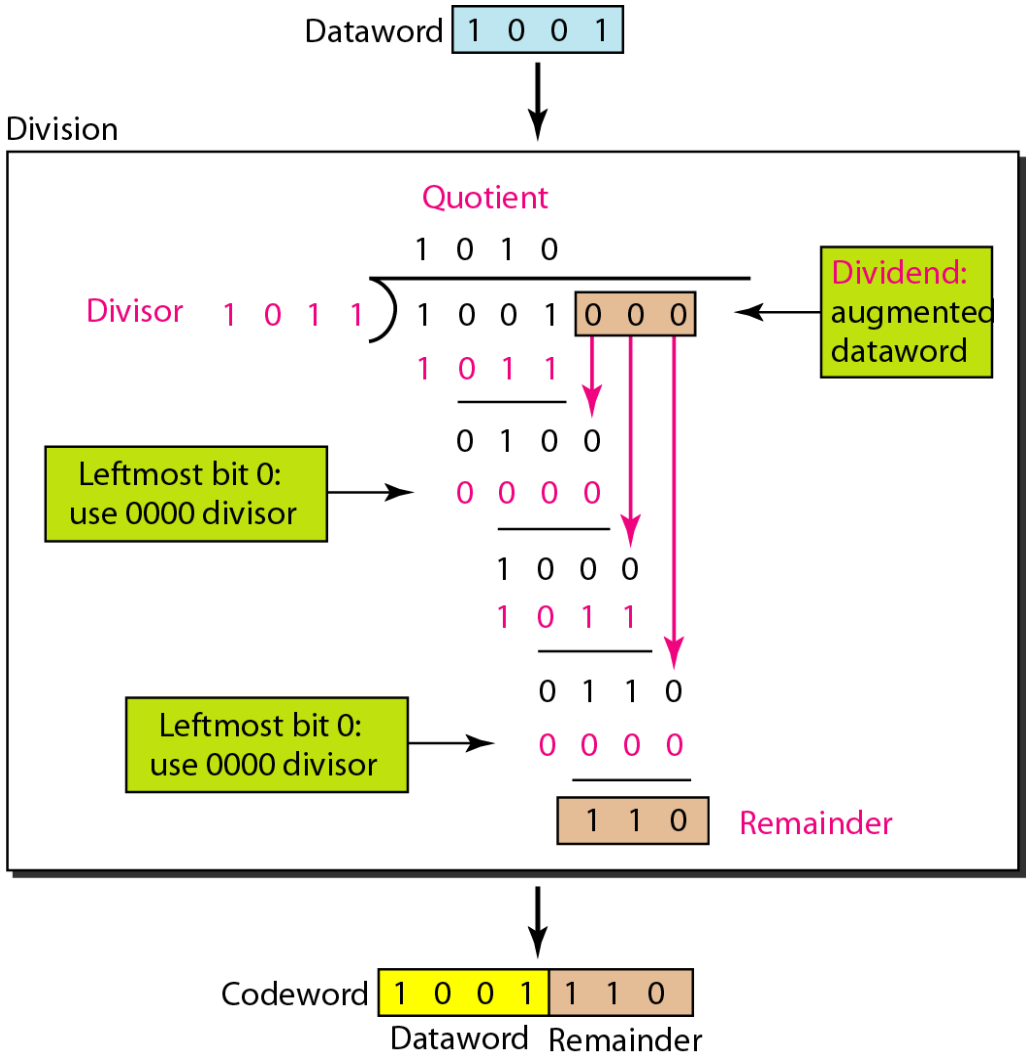
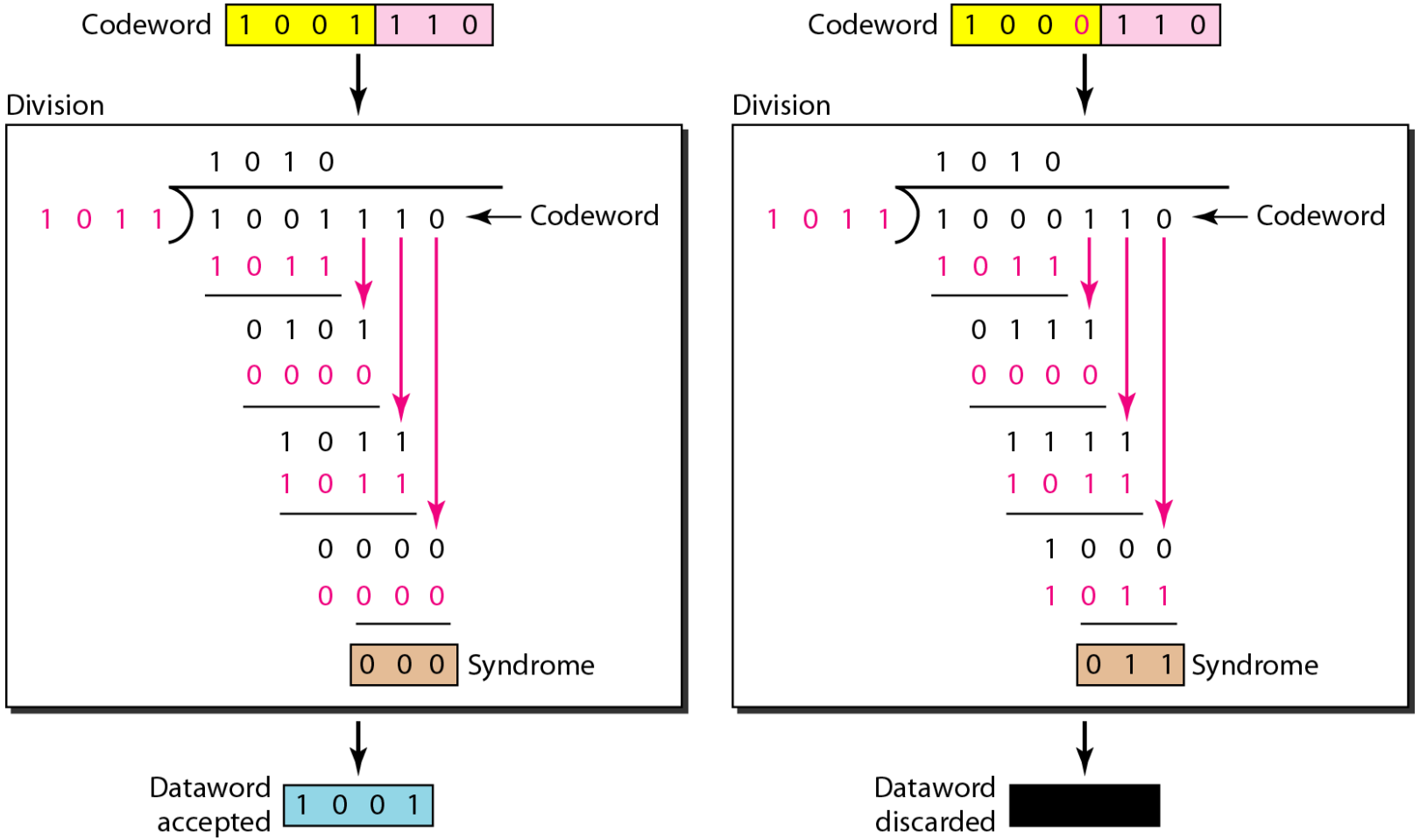


Figure 10.16 Division in the CRC decoder for two cases



CHECKSUM

The last error detection method we discuss here is called the checksum.

The checksum is used in the Internet by several protocols although not at the data link layer.

Note

Sender site:

1. ***The message is divided into 16-bit words.***
2. ***The value of the checksum word is set to 0.***
3. ***All words including the checksum are added using one's complement addition.***
4. ***The sum is complemented and becomes the checksum.***
5. ***The checksum is sent with the data.***

Note

Receiver site:

1. ***The message (including checksum) is divided into 16-bit words.***
2. ***All words are added using one's complement addition.***
3. ***The sum is complemented and becomes the new checksum.***
4. ***If the value of checksum is 0, the message is accepted; otherwise, it is rejected.***

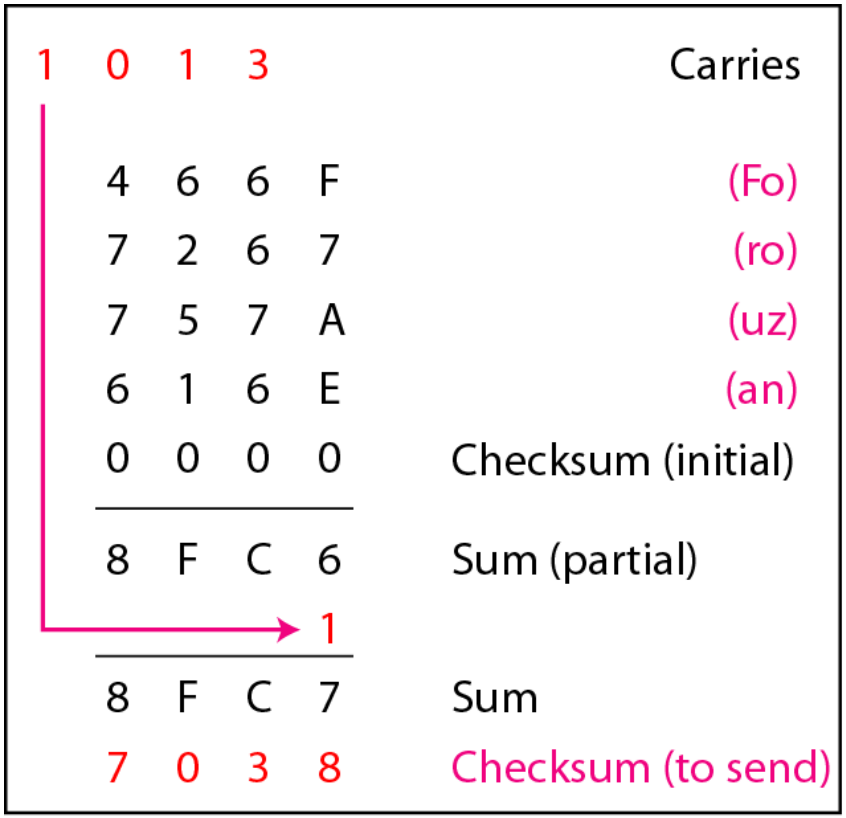
Example 10.23

Let us calculate the checksum for a text of 8 characters ("Forouzan"). The text needs to be divided into 2-byte (16-bit) words.

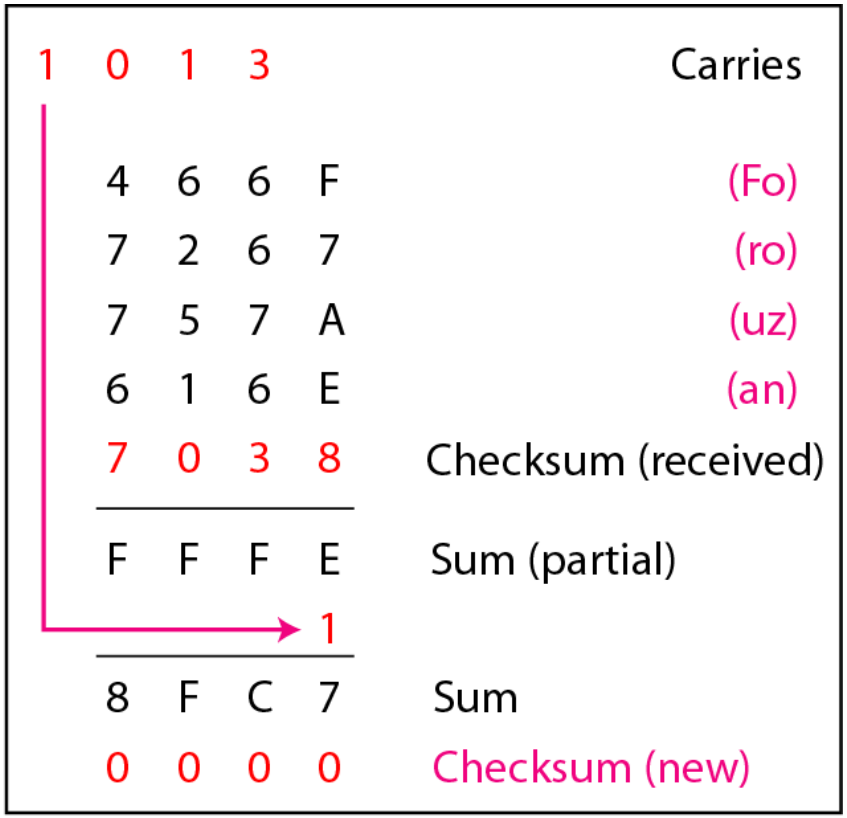
We use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as 0x46 and o is represented as 0x6F.

Figure 10.25 shows how the checksum is calculated at the sender and receiver.

Figure 10.25 *Example 10.23*



a. Checksum at the sender site



a. Checksum at the receiver site

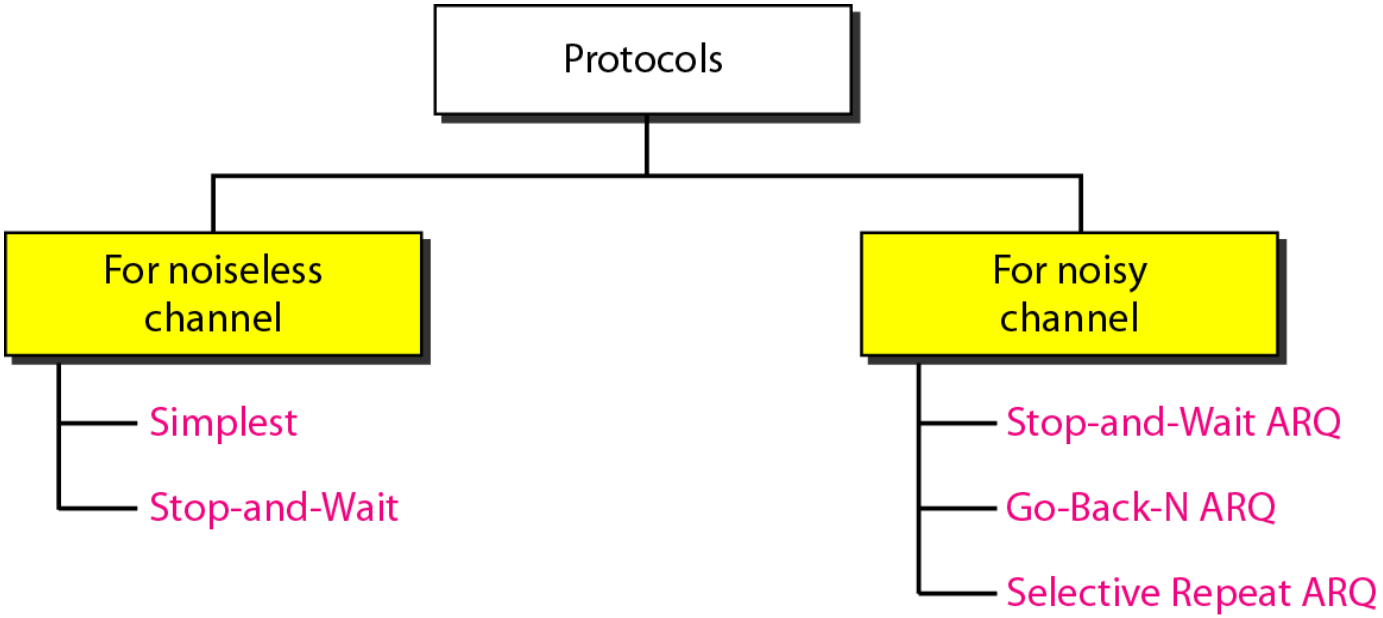
Elementary data link protocols

Elementary Data Link Protocol

- The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.
- ***Media Access Control***
- ***Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.***
- ***Error control in the data link layer is based on automatic repeat request, which is the retransmission of data***

Protocols

- Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another.



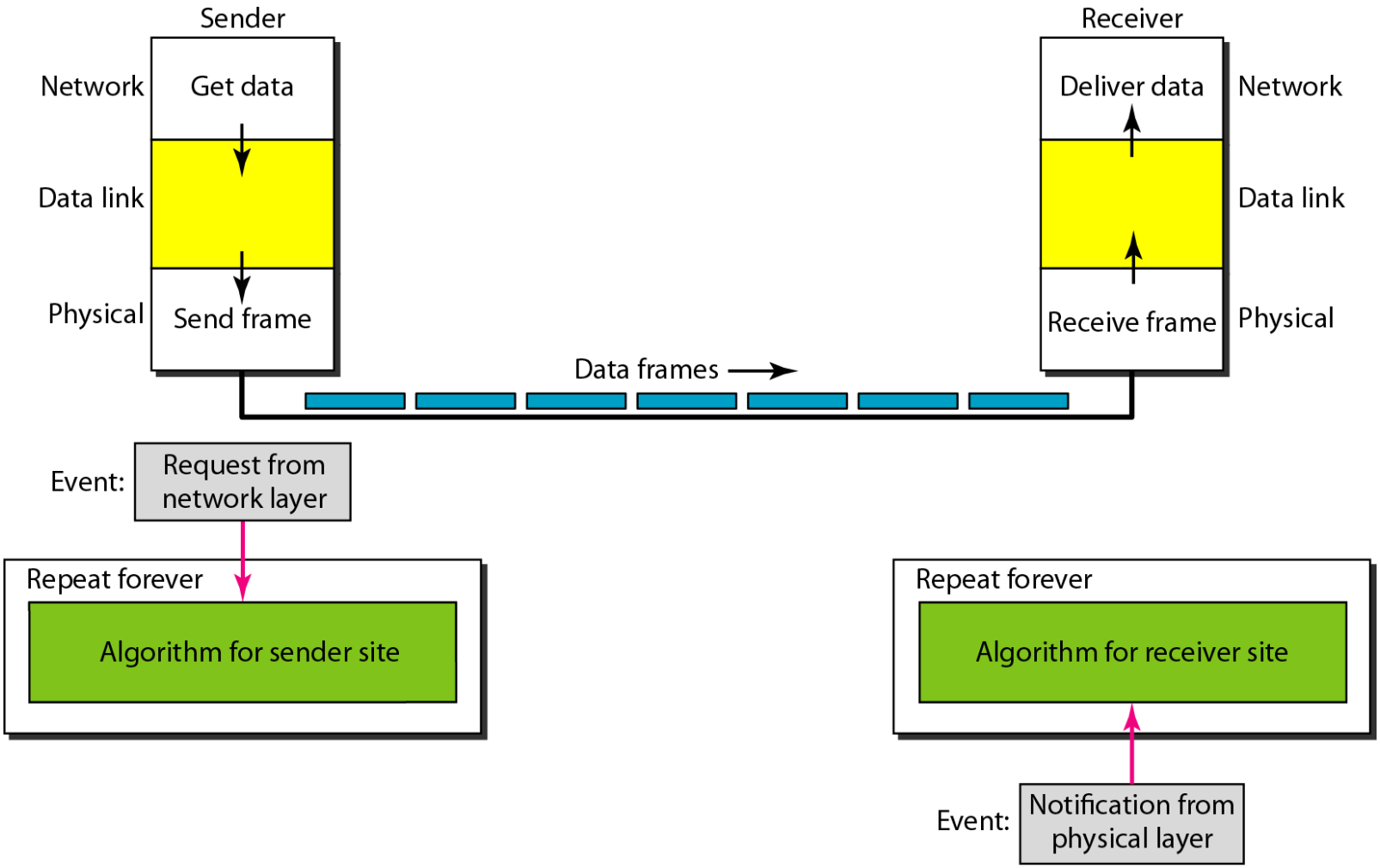
- Unidirectional Communication
- Piggybacking

Noiseless Channel

- No frames are lost, duplicated or corrupted.
- **Two Protocols:**
  1. Simplest Protocol
  2. Stop and Wait Protocol
- First protocol doesn’t use flow control.
- Both don’t require error control.

Noiseless Channel

1. Simplest Protocol



Noiseless Channel

1. Simplest Protocol

Example

Figure shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver.

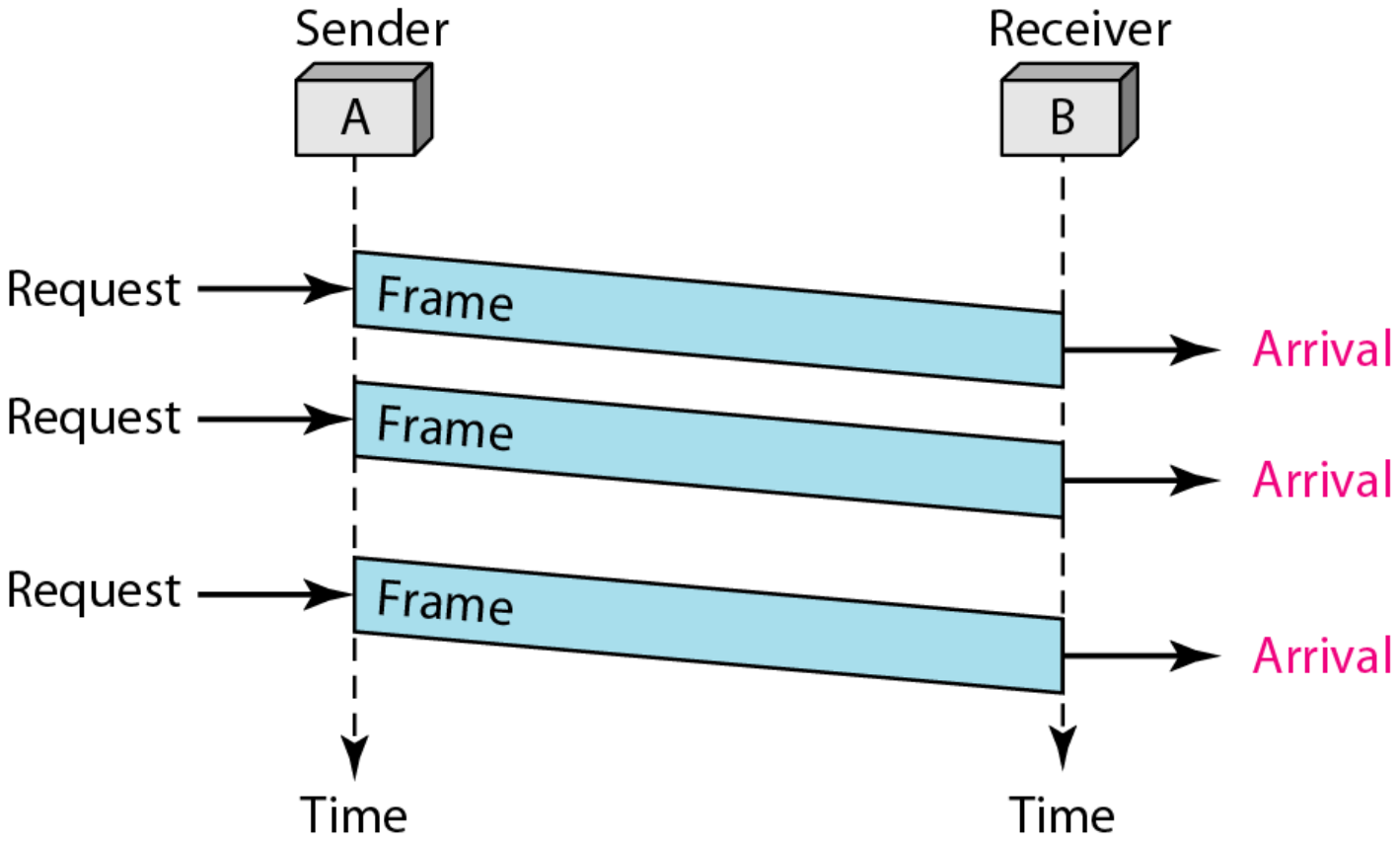
To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes;

The height of the box defines the transmission time difference between the first bit and the last bit in the frame.

Noiseless Channel

1. Simplest Protocol

Example



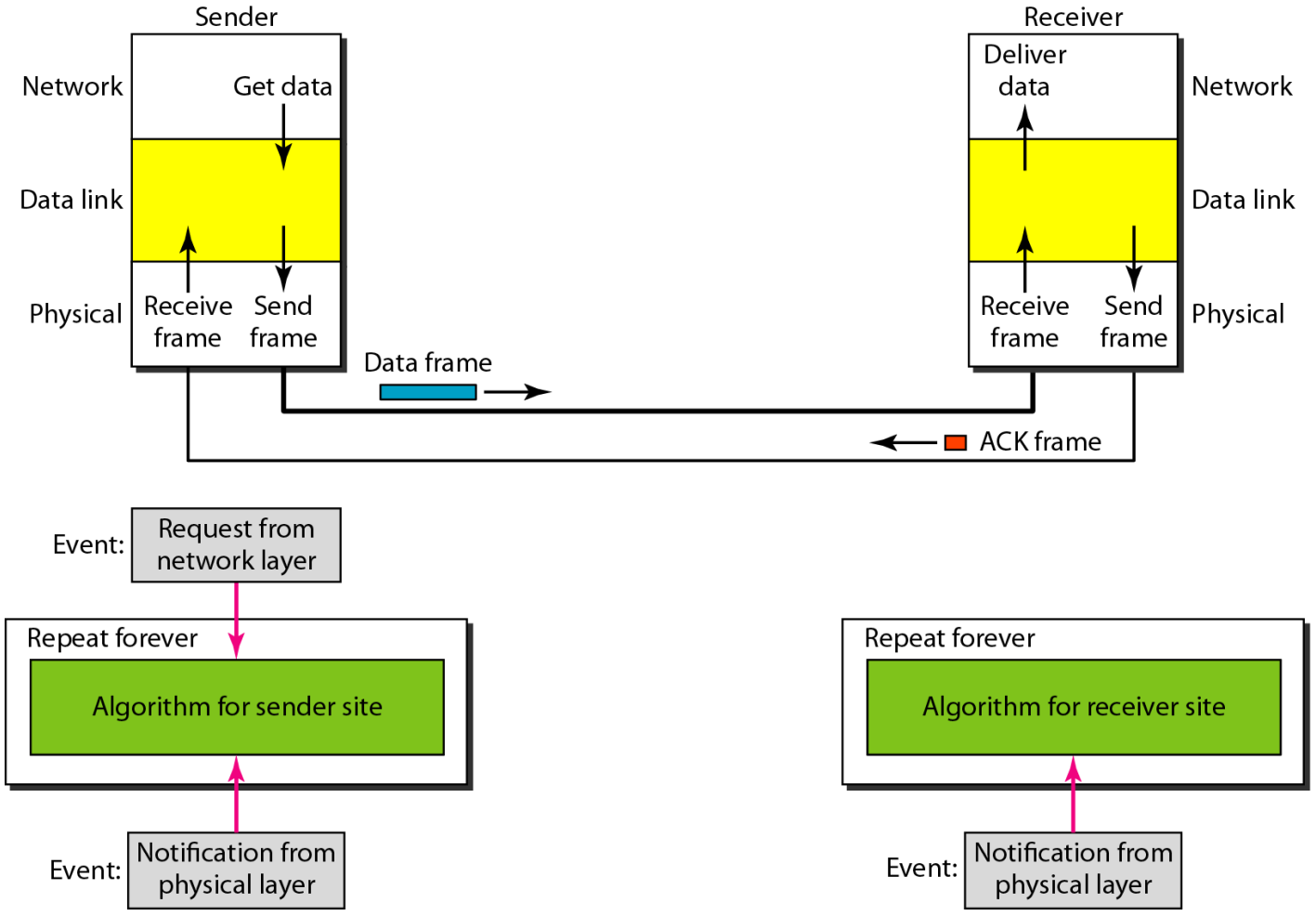
Noiseless Channel

2. Stop and Wait Protocol

- There must a feedback from Receiver to Sender.
- In this protocol, Sender sends one frame, stops until it receives confirmation from Receiver.
- Add Flow control to previous protocol.

Noiseless Channel

2. Stop and Wait Protocol



Noiseless Channel

2. Stop and Wait Protocol

Example

Figure shows an example of communication using this protocol. It is still very simple.

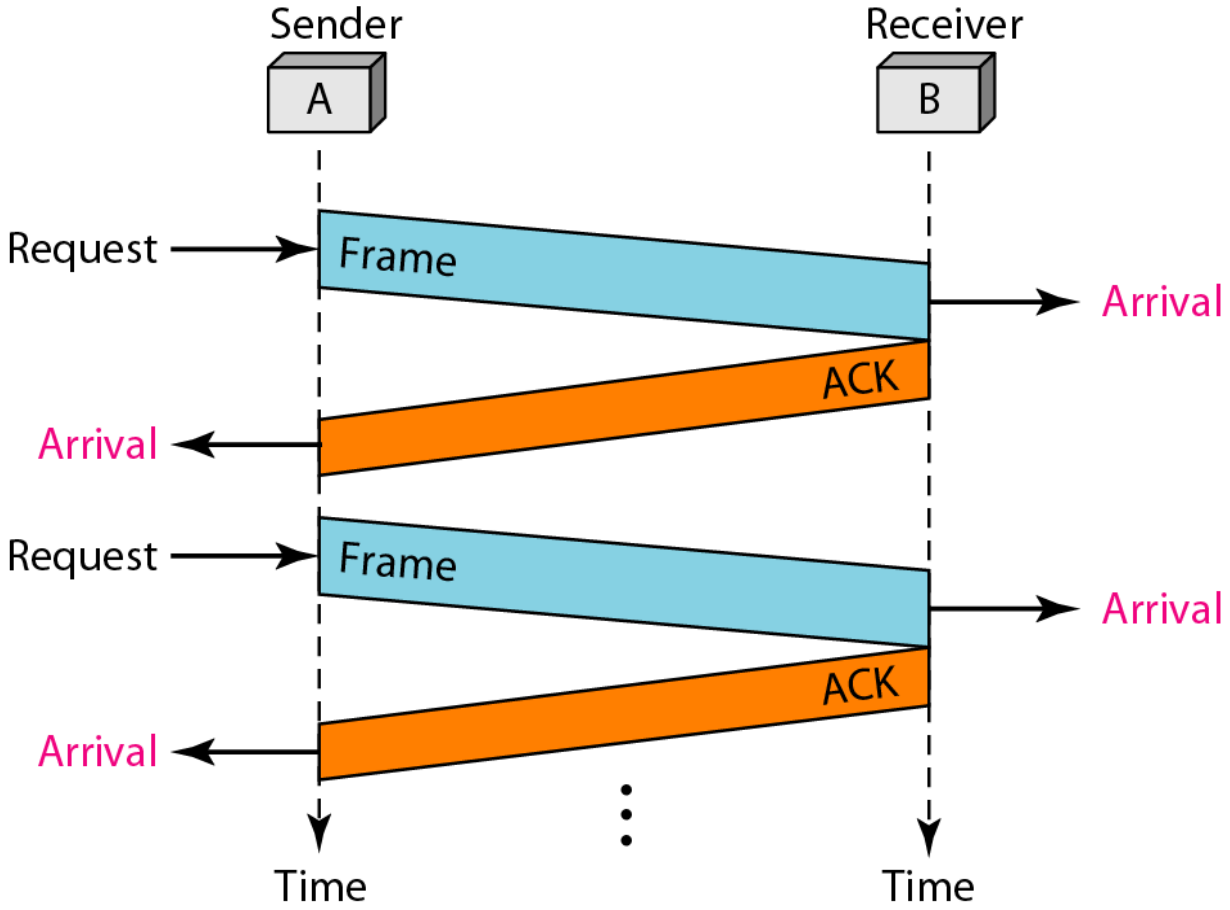
The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame.

Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

Noiseless Channel

2. Stop and Wait Protocol

Example



Noisy Channel

- Noiseless channels are nonexistent.
- **Protocols for Noisy Channel**
  1. Stop-and-Wait Automatic Repeat Request
  2. Go-back-N Automatic Repeat Request
  3. Selective Repeat Automatic Repeat Request

Noisy Channel Protocols

1. Stop-and-Wait Automatic Repeat Request

- Adds a simple error control mechanism to Stop-and-Wait Protocol.
- Lost frames are more difficult to handle than corrupted frames.
- No way to identify the frame in the last protocol.
- The received frame could be a correct one, or a duplicate, or a frame out of order.
- This protocols numbers each frame, to deal with above problems.

Noisy Channel Protocols

1. Stop-and-Wait Automatic Repeat Request

- The Corrupted or lost frame will be retransmitted in this protocol.
- Sender store a copy of each frame.
- 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.
- In Stop-and-Wait ARQ, we use sequence numbers to number the frames.

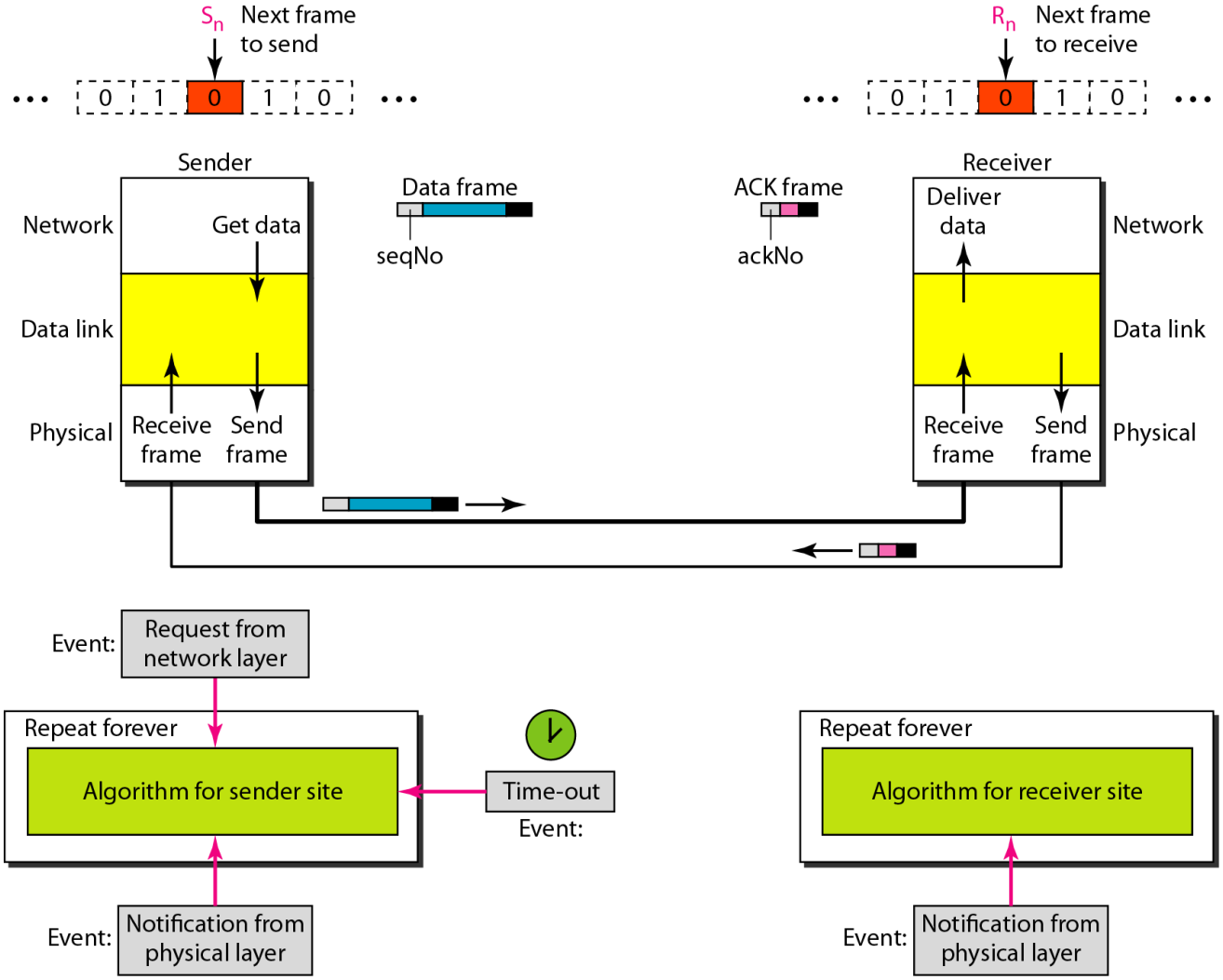
Noisy Channel Protocols

1. Stop-and-Wait Automatic Repeat Request

- Range of Sequence number. The sequence numbers are based on modulo-2 arithmetic.
- the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Noisy Channel Protocols

1. Stop-and-Wait Automatic Repeat Request



Noisy Channel

1. Stop-and-Wait Automatic Repeat Request

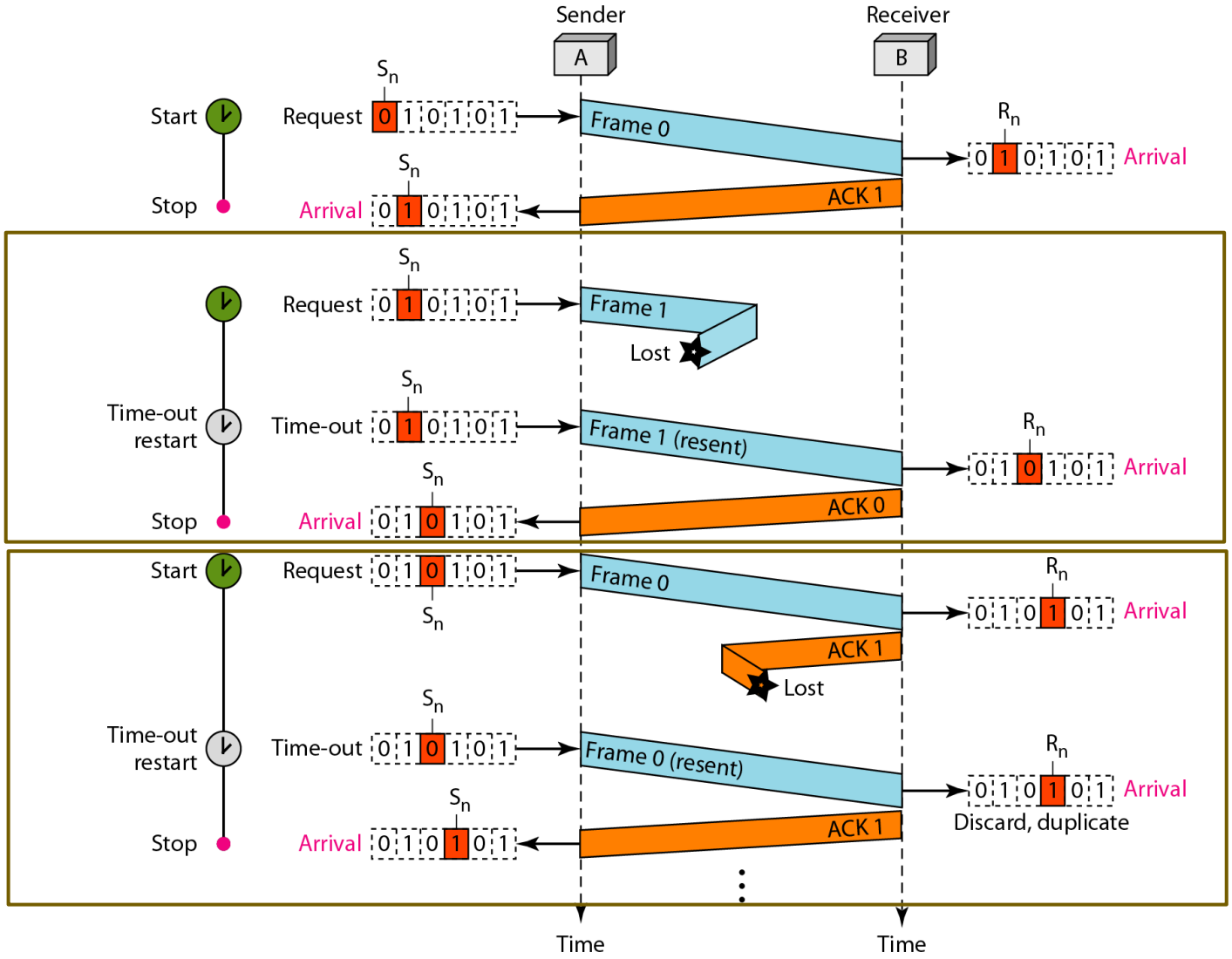
Example

Figure shows an example of Stop-and-Wait ARQ. Frame 0 is sent and acknowledged.

Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops.

Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.

Example



Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.

Noisy Channel

1. **Stop-and-Wait Automatic Repeat Request**

   Efficiency:

   1. **Bandwidth Delay Product**
   2. **Pipelining**

The Bandwidth-Delay product is the measure of the number of bits, we can send out of the system while waiting for news from receiver.

Noisy Channel

1. Stop-and-Wait Automatic Repeat Request

Example

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

Noisy Channel

1. Stop-and-Wait Automatic Repeat Request

Example

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again.

However, the system sends only 1000 bits.

We can say that the link utilization is only 1000/20,000, or 5 percent.

For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Noisy Channel

2. Go-Back-N Automatic Repeat Request

- If we can send Several frames before we receive any news about the previous frames, than pipelining is possible which can improves the efficiency.
- **Goal for next two protocols.**
- Keep channel busy while the sender is waiting for acknowledgement.
- Keep a copy of all these frames.
- **Sequence number**: header supports m bit, than $2^m - 1$. if m=4, than sequence numbers are,

  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,**0,1,2,3,4,5** . .

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Sliding Window:

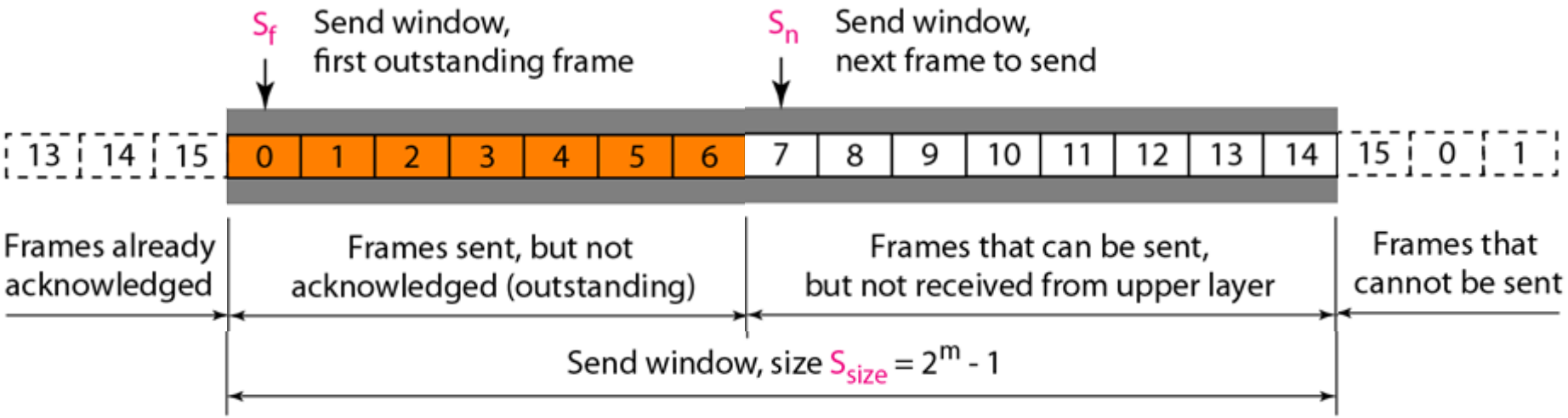
- Defines the range of sequence numbers, that is the concern of the sender and receiver.
- Sender and Receiver deal with only a part of possible numbers.
- Range concern with Sender: **Send Sliding Window.**
- Range concern with Receiver: **Receive Sliding Window.**
- **Window** make sure that the right frame is send and received.

Noisy Channel

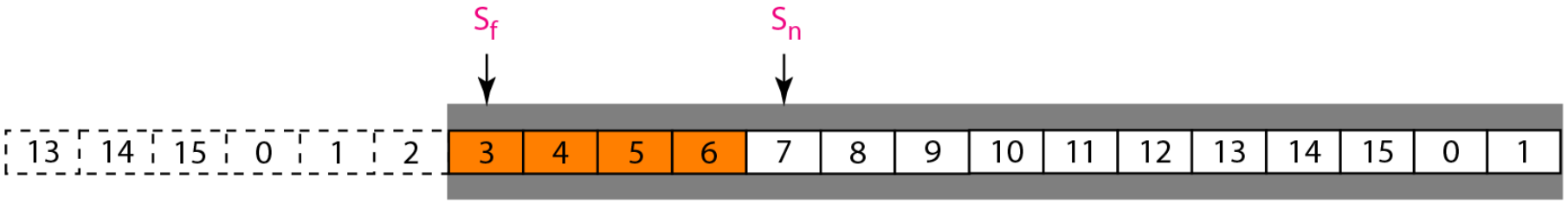
2. Go-Back-N Automatic Repeat Request

Sliding Window:

Send Sliding Window before Sliding



Send Sliding Window after Sliding

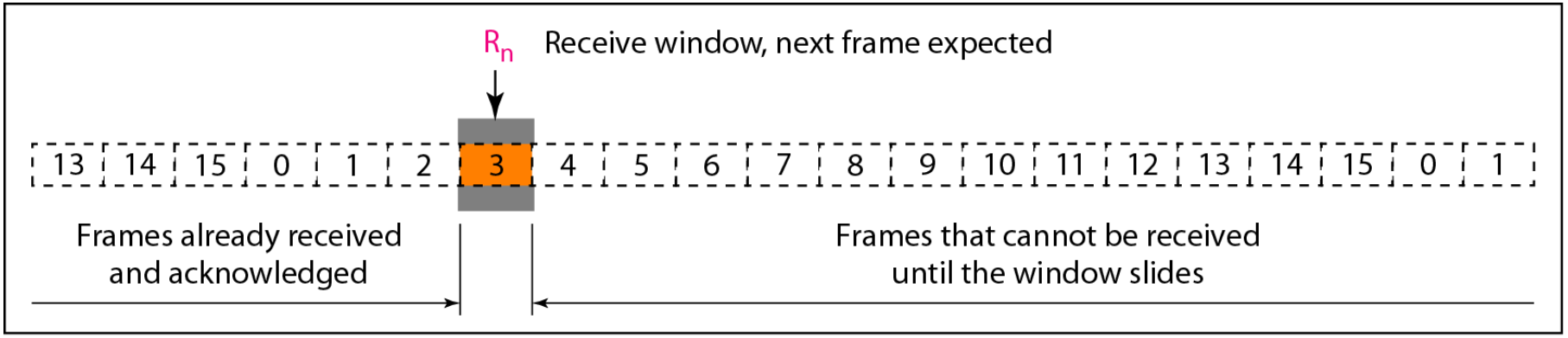


Noisy Channel

2. Go-Back-N Go-Back-N Automatic Repeat Request

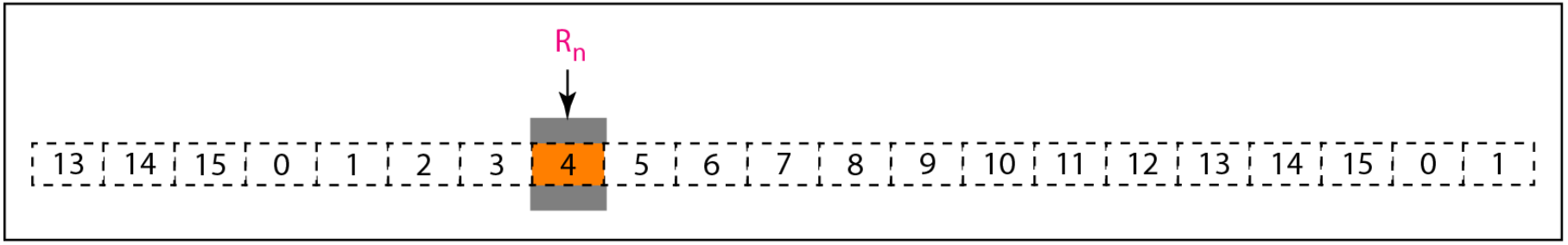
Sliding Window:

Receive Sliding Window before Sliding



a. Receive window

Receive Sliding Window after Sliding



b. Window after sliding

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Sliding Window:

Note

Size of the sender window is 2^m-1 while of receiving window is always 1.

Any Frames received out of order will be discarded. One timer will be used.

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Sliding Window:

Note

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}.

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Sliding Window:

Note

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.

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Timers:

- One timer in our implementation.
- All outstanding frames retransmitted, once timer get expired.

Noisy Channel

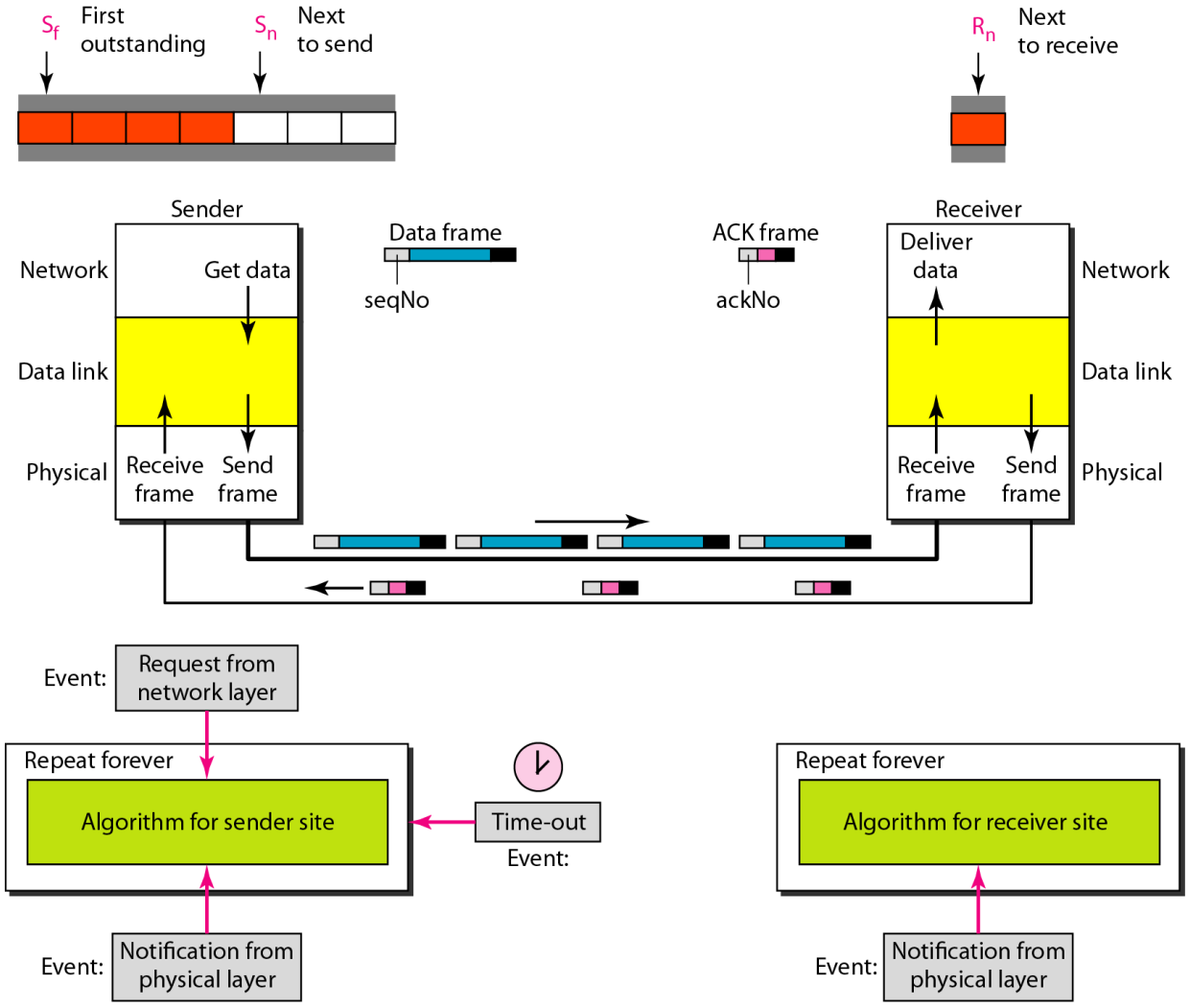
2. Go-Back-N Automatic Repeat Request

Acknowledgment:

- Receiver sends a positive acknowledgement, if a frame has arrived safe and in order.
- If received frame is damaged or out of order, receiver will silent and discard all the subsequent frames until it receives the one is expecting.
- Silent receiver causes the timer of the unacknowledged frame at the sender site to expire.
- Example: *if frame 6 sent, but the timer for 3 expires, Sender goes back and send frame 3, 4, 5 and 6 again.*

Noisy Channel

2. Go-Back-N Automatic Repeat Request



Noisy Channel

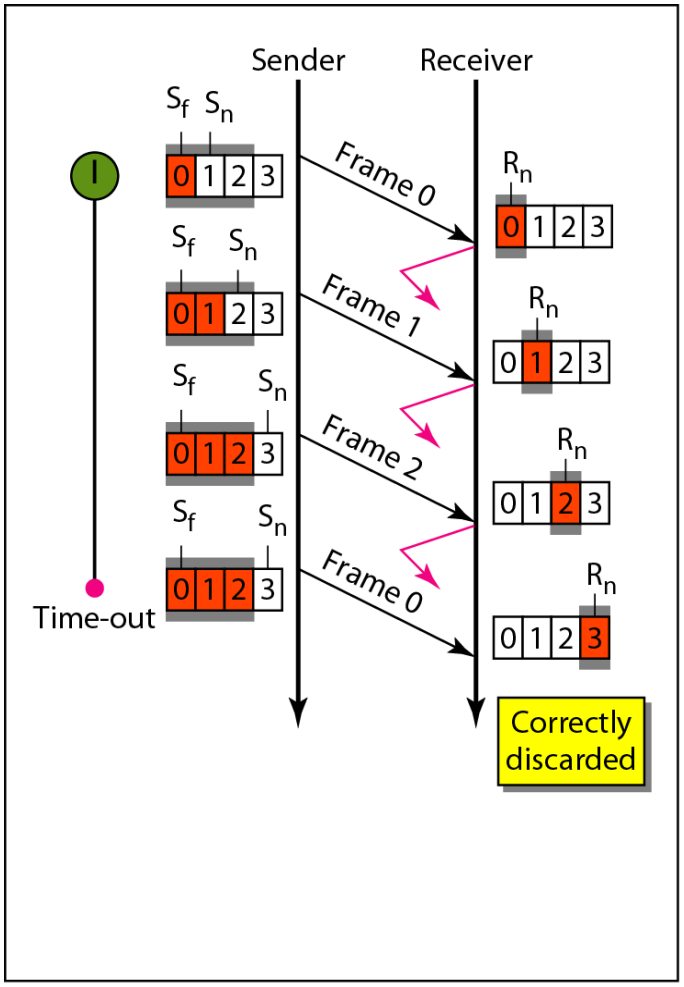
2. Go-Back-N Automatic Repeat Request

Send Window Size:

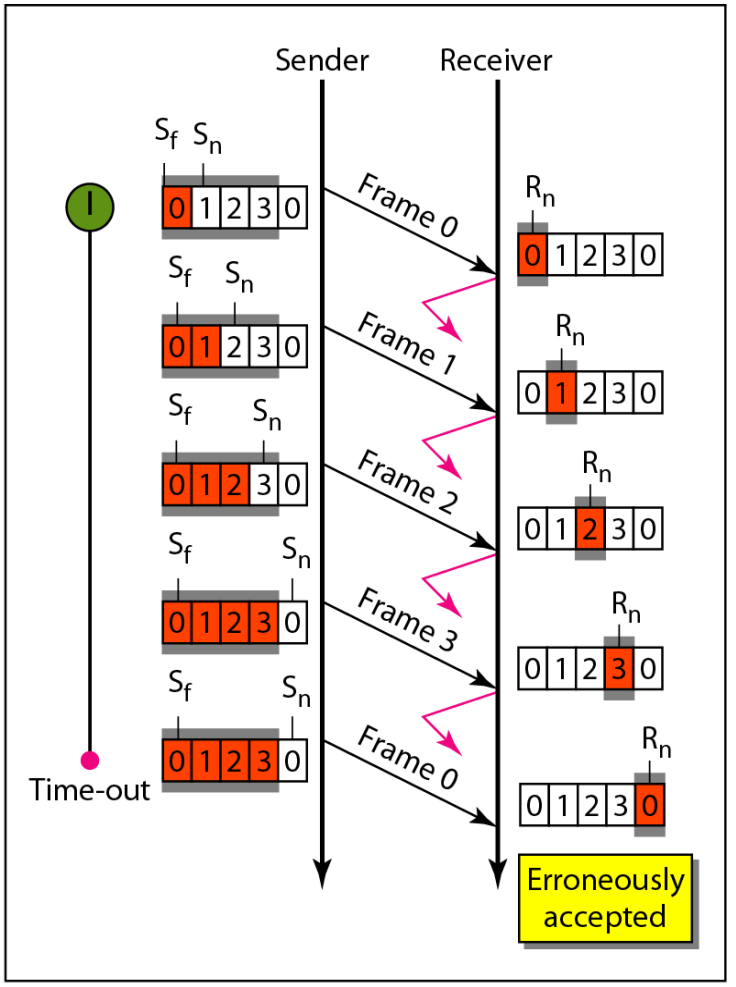
- **Why the size of the send window must be less than 2^m?**
- As an example, if m = 2, than window size can be 3, not 4.

Noisy Channel

2. Go-Back-N Automatic Repeat Request



a. Window size < 2^m



b. Window size = 2^m

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Note

In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Noisy Channel

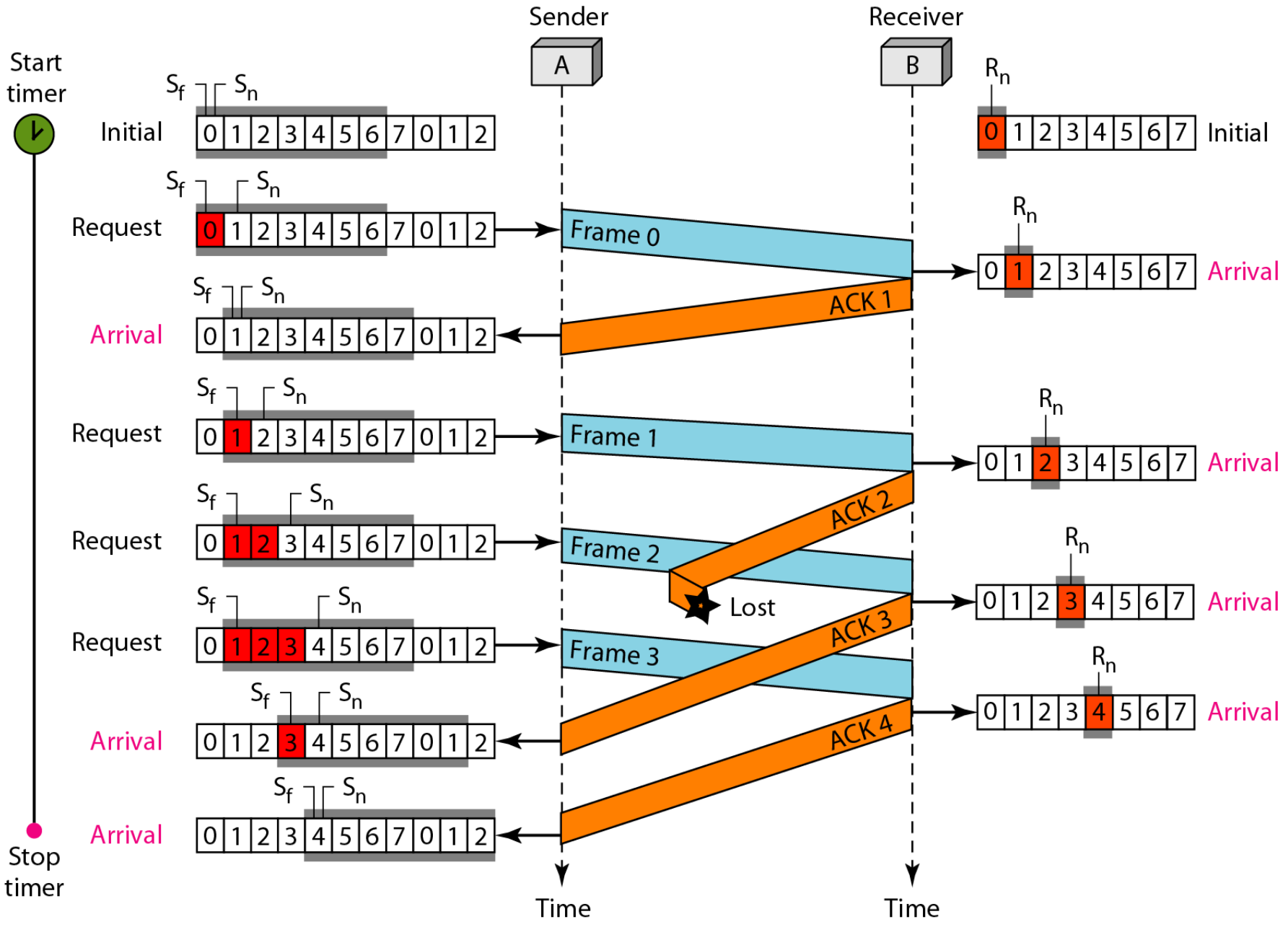
2. Go-Back-N Automatic Repeat Request

Example 1:

This is an example of a case where the forward channel is reliable, but the reverse is not.

No data frames are lost, but some ACKs are delayed and one is lost.

The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.


Sender
Receiver
A
B
Start timer
Stop timer
Initial
Request
Arrival
Frame 0
Frame 1
Frame 2
Frame 3
ACK 1
ACK 2
ACK 3
ACK 4
Lost
Time
Time

Noisy Channel

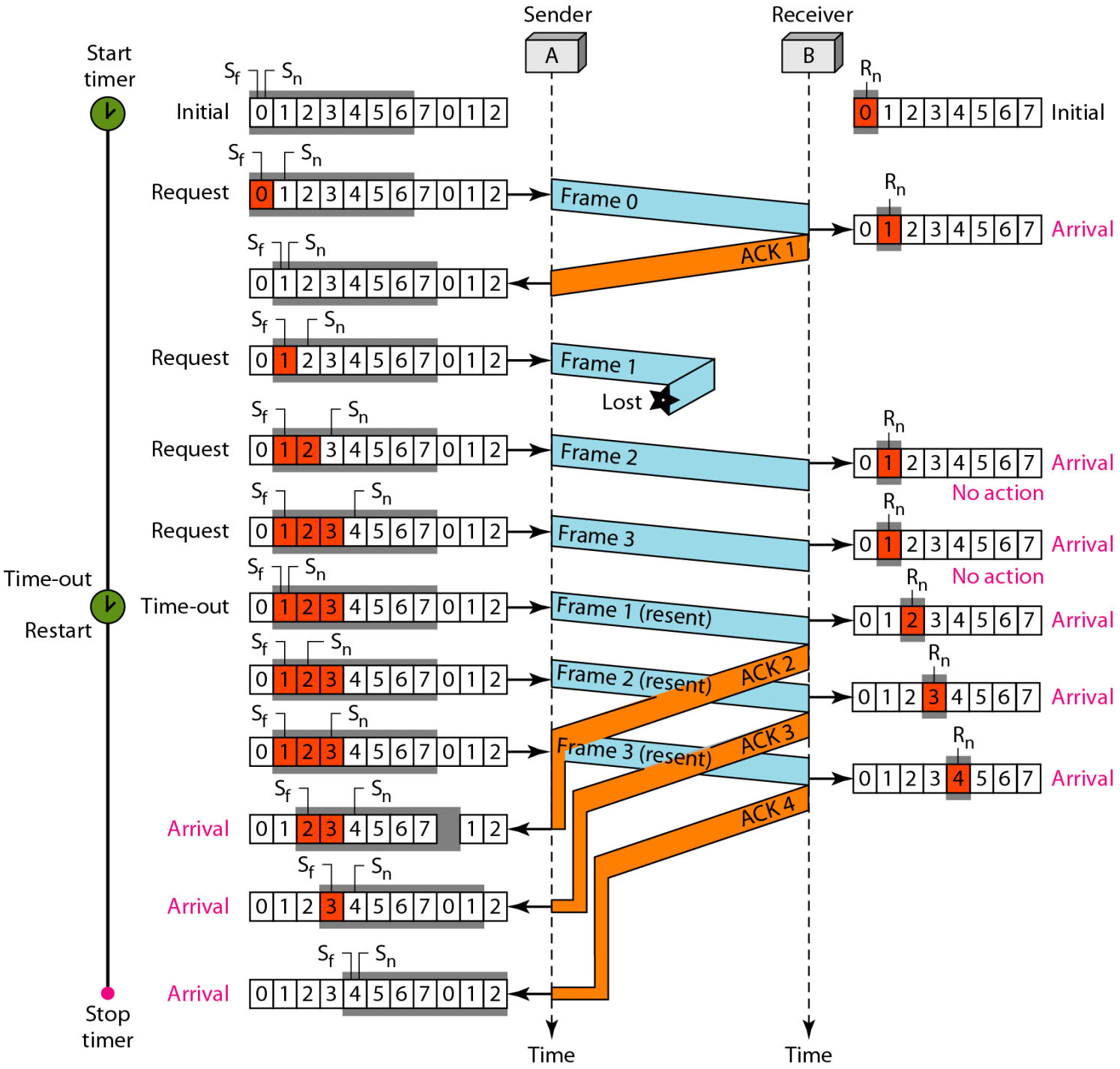
2. Go-Back-N Automatic Repeat Request

Example 2:

This is an example shows what happen when a frame is lost.

Frame 0, 1, 2 and 3 are sent. However frame 1 is lost. Receiver receives frame 2 and 3 out of order and discarded.

When timer gets expire at sender site, for frame 1, all the outstanding frame retransmitted.


Sender
Receiver
A
B
Start timer
Stop timer
Time-out
Restart
Initial
Request
Frame 0
ACK 1
Frame 1
Lost
Frame 2
No action
Frame 3
Time-out
Frame 1 (resent)
Frame 2 (resent)
ACK 2
Frame 3 (resent)
ACK 3
ACK 4
Arrival
Time
Time

Noisy Channel

2. Go-Back-N Automatic Repeat Request

Example 3:

Draw the sender and receiver side sliding window for the system using Go-Back-N ARQ:

1. Frame 0, 1 and 2 sent and acknowledged.
2. Frame 3, 4, 5 and 6 sent, but Frame 4 lost during the transmission.

Also find the sender and receiver side window size for same, if M = 3.

Noisy Channel

3. Selective Repeat Automatic Repeat Request

- Go-Back-N simplifies the processing at receiver site.
- But, inefficient for Noisy link.
- Retransmission of all outstanding frames uses up the bandwidth and slow down the transmission.

- **Selective Repeat ARQ does not resend all the N frames, when just one frame is damaged.**
- **Only the damaged frame is retransmitted.**
- **More Efficient for Noisy link.**
- **Processing at receiver site is more complex.**

Noisy Channel

3. Selective Repeat Automatic Repeat Request

Windows:

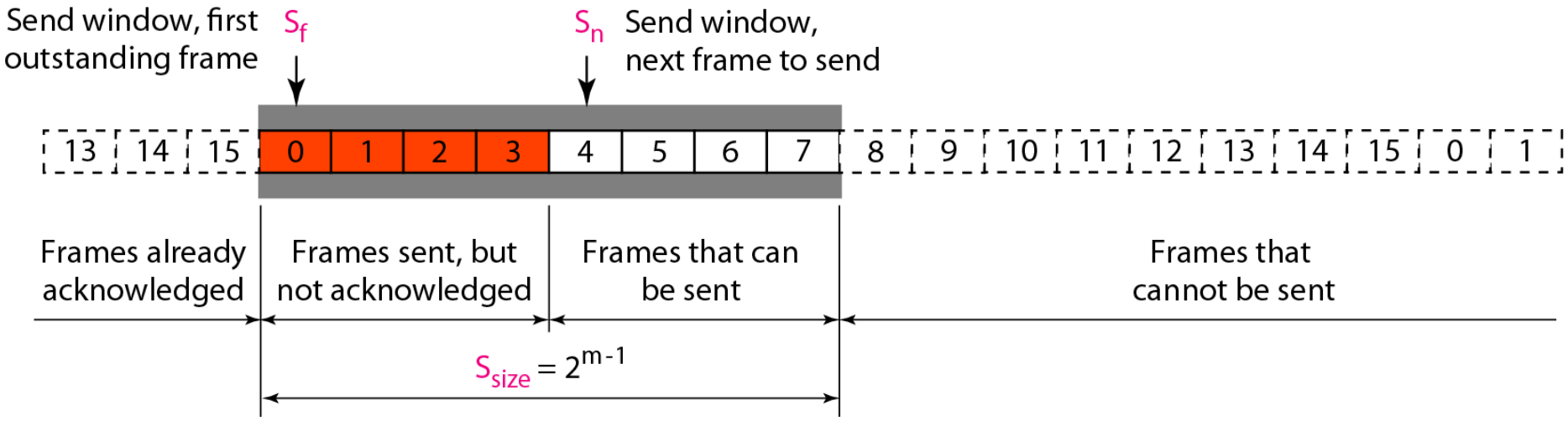
- Uses two window; **send window** and **receive window.**
- The size of the send window is much smaller, it is 2^{m-1}.
- Same window size at sender and receiver site.
- If m=4, sequence number range may be 0 to 15, but window size is 8.

Noisy Channel

3. Selective Repeat Automatic Repeat Request

Windows:

Send Window

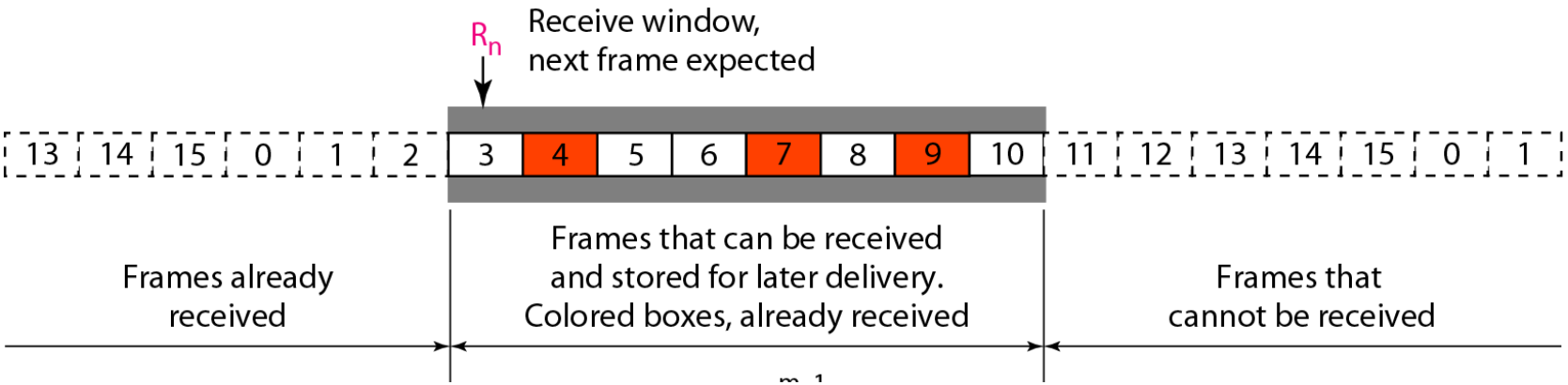


Noisy Channel

3. Selective Repeat Automatic Repeat Request

Windows:

Receive Window

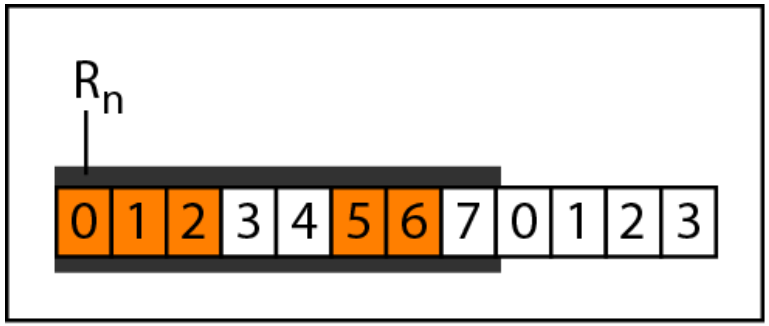


- Same Size: 2^{m-1}
- All frames inside window are accepted.
- Out of order frames are accepted and stored by receiver.
- No packet is delivered out of order to Network Layer.

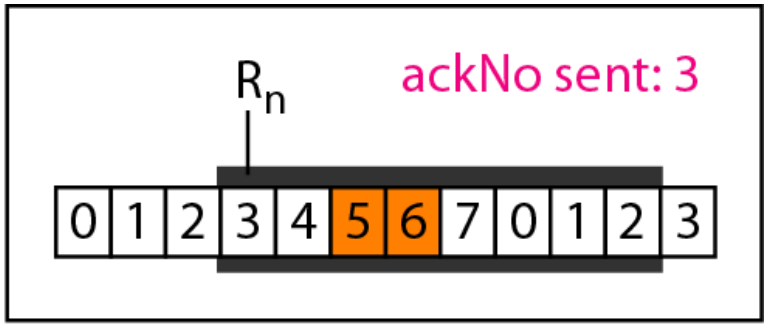
Noisy Channel

3. Selective Repeat Automatic Repeat Request

Delivery of Date



a. Before delivery



b. After delivery

Noisy Channel

3. Selective Repeat Automatic Repeat Request

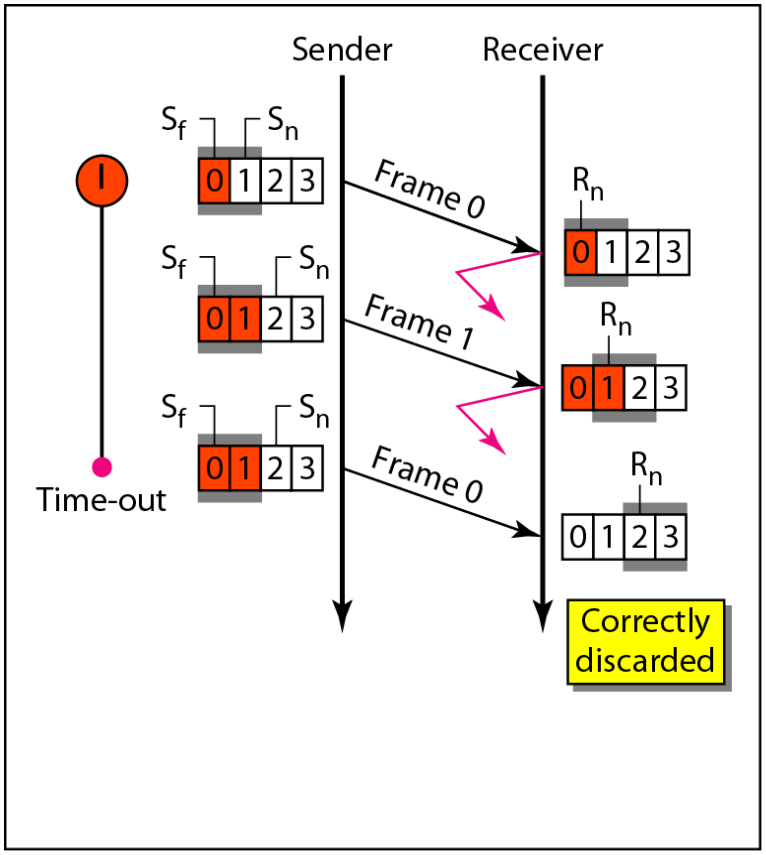
Send Window Size:

- **Why the size of the send window must be at most one-half of 2^m?**
- As an example, if m = 2, than window size can be 2, not 3.

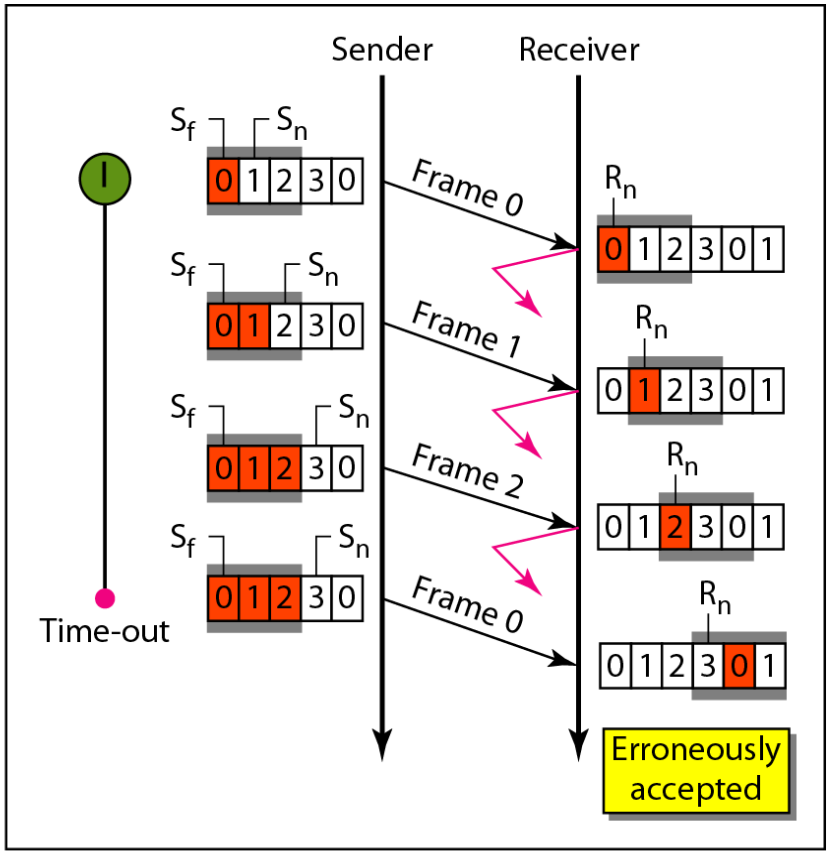
Noisy Channel

3. Selective Repeat Automatic Repeat Request

Send Window Size:



a. Window size = 2^{m-1}



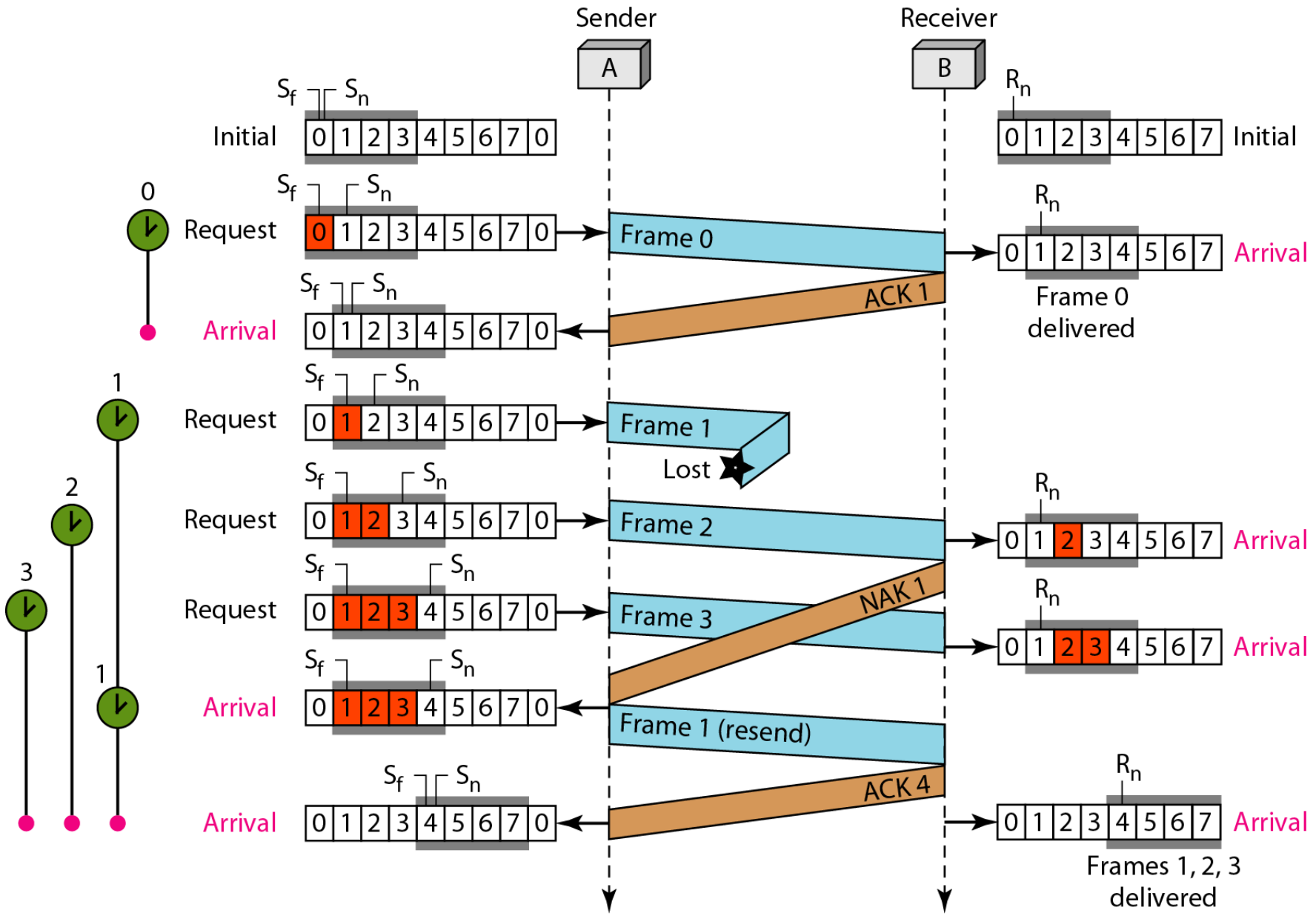
b. Window size > 2^{m-1}

Noisy Channel

3. Selective Repeat Automatic Repeat Request

Example:

- Examples shows the behavior of Selective Repeat ARQ at the time of frame loss.
- Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3).
- Frame 0 send and acknowledged properly.
- Frame 1 lost during the transmission, and NAK 1 arrives as a negative acknowledgement.


Sender
Receiver
A
B
Initial
Request
Arrival
Frame 0
ACK 1
Frame 0 delivered
Frame 1
Lost
Frame 2
Frame 3
NAK 1
Frame 1 (resend)
ACK 4
Frames 1, 2, 3 delivered

Noisy Channel

3. Selective Repeat Automatic Repeat Request

Example:

Draw the Sender and Receiver side-sliding window for the system using selective repeat ARQ for the following cases.

1. Frame 0, 1 sent and acknowledged.
2. Frame 2, 3 and 4 sent and NACK for Frame 2 arrived.

Find the sender and receiver window size for Go-Back-N ARQ and Selective Repeat ARQ if m=4.

Noisy Channel

Piggybacking

- All protocols discussed here are unidirectional.
- In real life, data frames are normally flowing in both the direction.
- **Piggybacking is a technique, can be use to improve the channel efficiency for bidirectional protocol.**
- When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B.