Data Link Layer

Outline

- Goal
  - Understanding the functions of DL

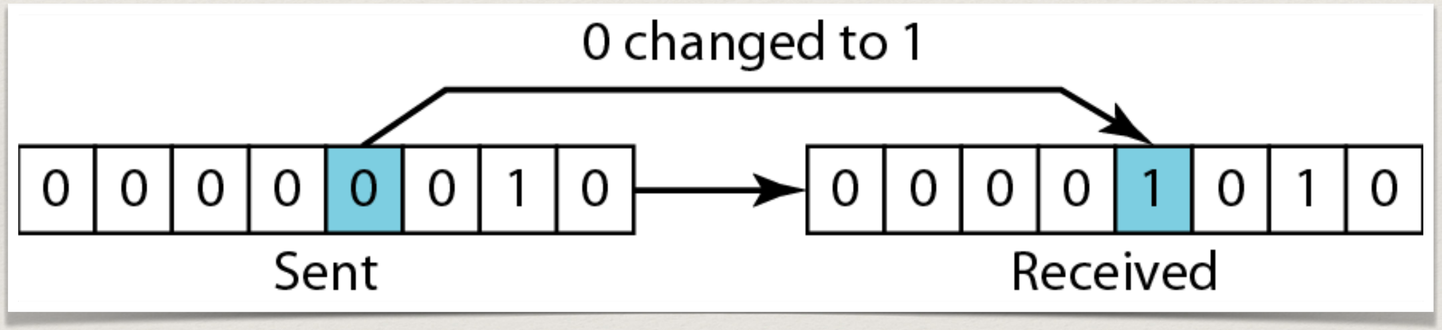
- Data Link Layer
- Error Control Techniques
- Flow Control Techniques
- Summary

Data Link layer Functions

- **Framing**
- **Physical addressing**
- **Flow control**
- **Error control**
  - Mechanisms to detect and retransmit damaged or lost frames
  - Uses a mechanism to recognize duplicate frames.
- **Access control:**

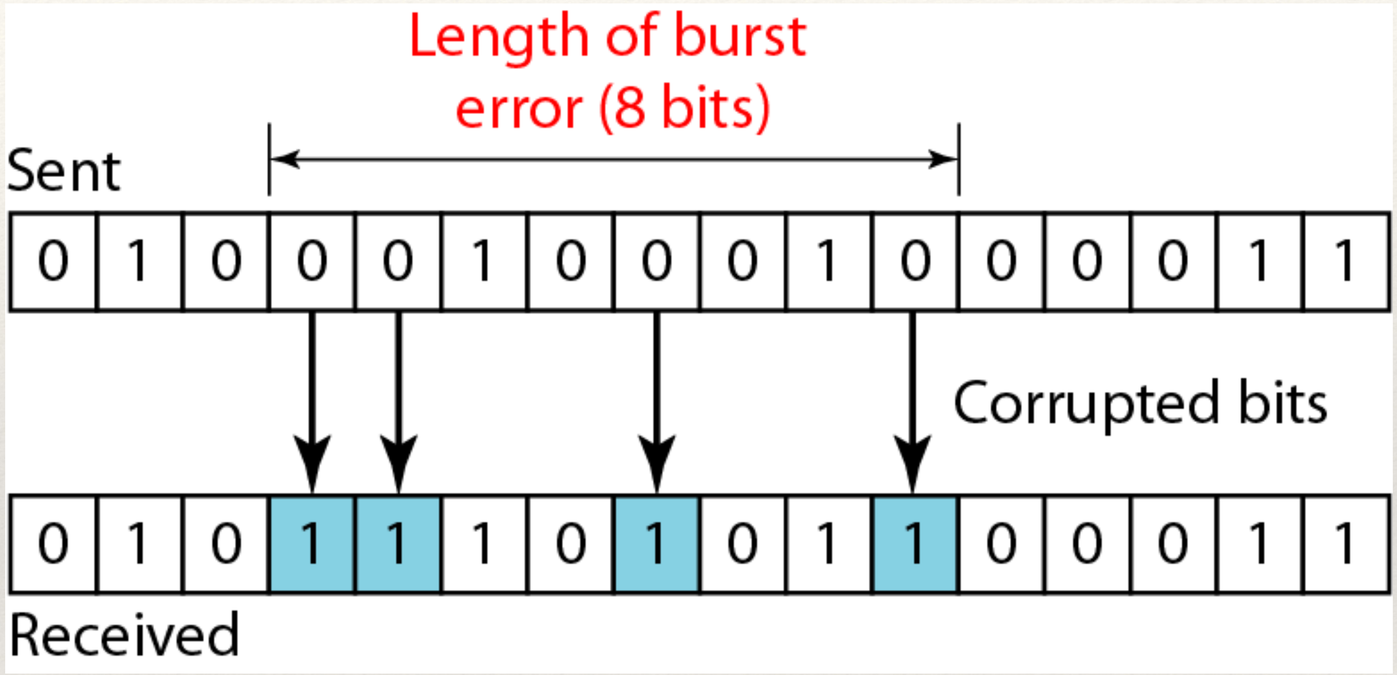
Types of Errors

- Single Bit Error: In a single-bit error, only 1 bit in the data unit has changed.



Types of Errors

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



Length: first corrupted bit to last corrupted

Basic Terminologies

Redundancy: To detect or correct errors, we need to send/add the extra redundant bits with the data.

Detection Vs Correction:

Detection: Only look for error occurred or not.

Correction: Need to know the number of bits and the locations of the corrupted bits in the message.

Forward Error Correction Vs Retransmission:

FEC: Receiver tries to guess the message by using the redundant bit

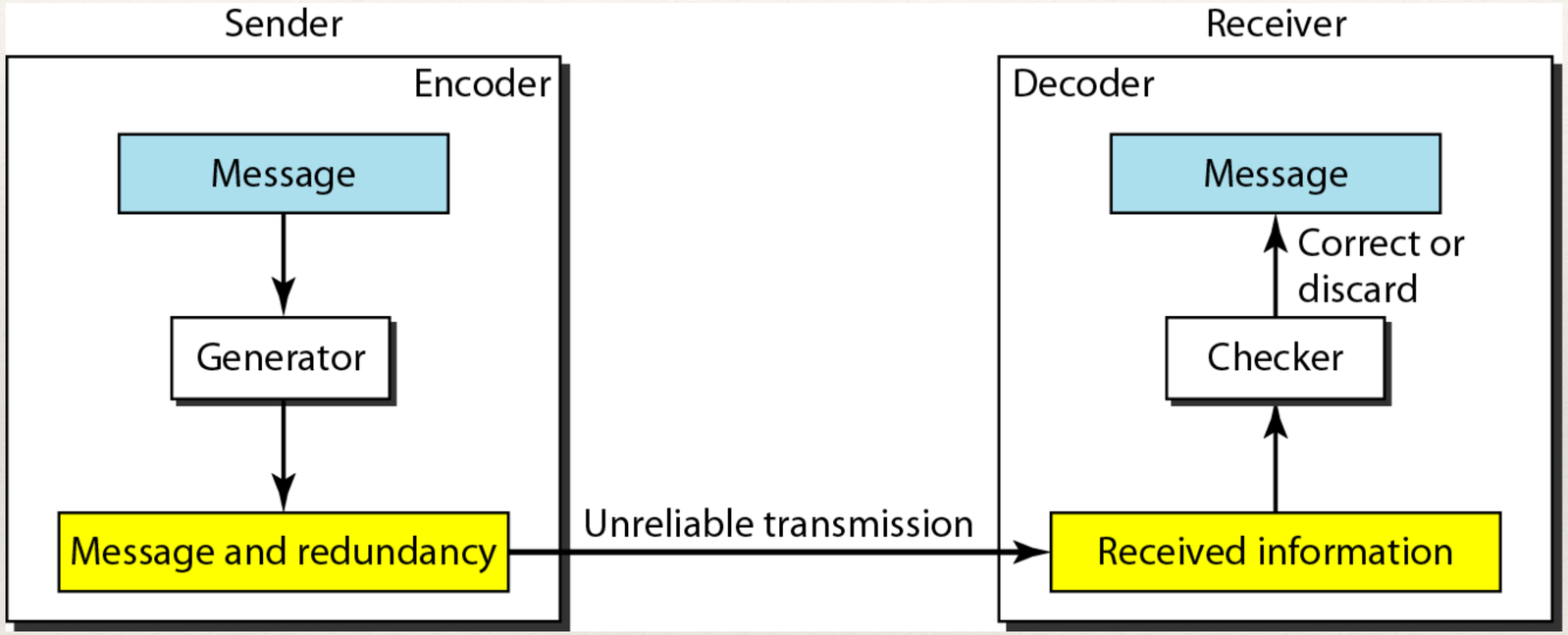
Retransmission: Receiver detects the occurrence of error and asks the sender to retransmit/resend the message.

Coding

- Redundancy is achieved via various coding schemes.
- Sender adds redundant bits that creates a relationship between the redundant bits and the actually data bits.
- The ratio of the redundant bits and the actual message and the robustness of the process are important factors of coding schemes.
- Two types of Coding Schemes are
  - Block Coding
  - Convolution Coding

The Structure of Encoder and Decoder

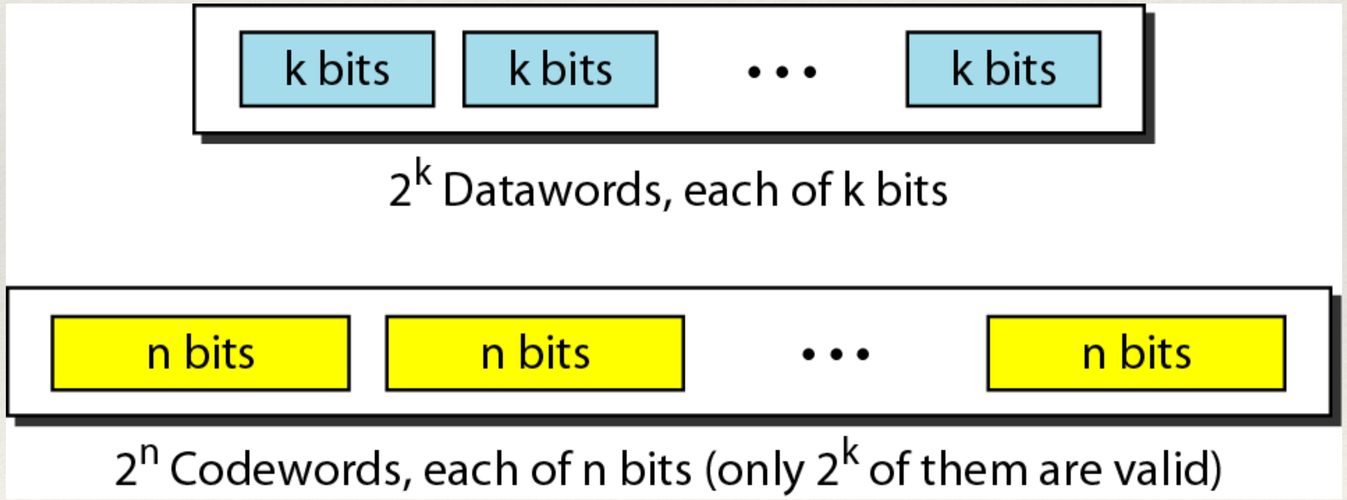
- To detect or correct errors, we need to send extra (redundant) bits with data



Encoder and Decoder

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.



Error Detection

- Enough redundancy is added to detect an error.
- The receiver knows an error occurred but does not know which bit(s) is(are) in error.
- Has less overhead than error correction.

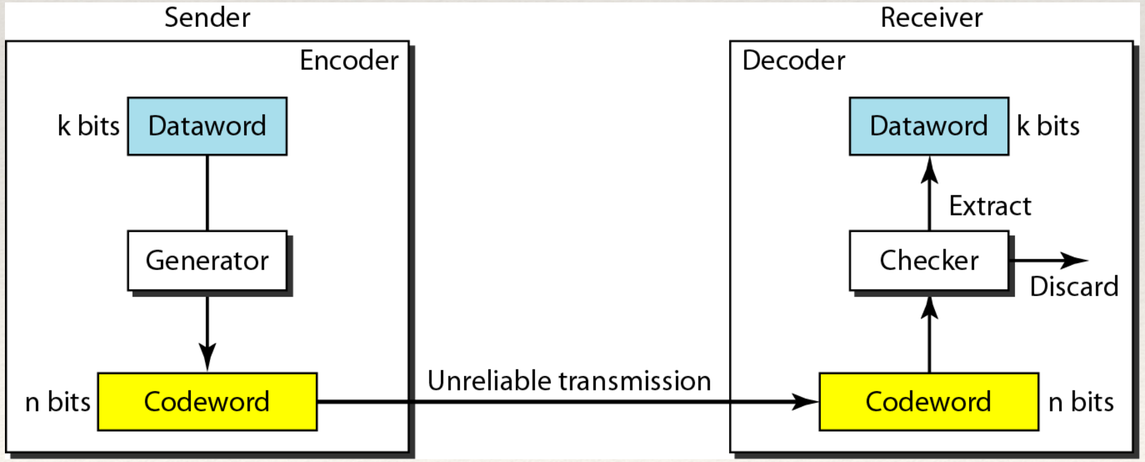


Fig. Process of Error Detection in Block Coding

Example

Let us assume that k = 2 and n = 3.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver.

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

Table 1

Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.

Example

2. The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is discarded.

3. The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Note: An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Error Correction

- Let us add more redundant bits to Example 1 to see if the receiver can correct an error without knowing what was actually sent.
- We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords.
- Table shows the datawords and codewords.
- Assume the dataword is 01. The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received. First, the receiver finds that the received codeword is not in the table.
- This means an error has occurred. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword

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

Table: 2

Example -2

1. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.

2. By the same reasoning, the original codeword cannot be the third or fourth one in the table.

3. The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

Hamming Distance

- The Hamming distance between two words is the number of differences between corresponding bits.
- The hamming distance between the two words x and y is represented by d(x,y).
- Hamming distance between two pairs of words is found using the XOR operation between the two words x and y.
- The Hamming distance d(000, 011) is 2 because

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

Hamming Distance

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

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

Note:

- The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.
- To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

Exercise: Find the minimum hamming distance for the above Tables 1 and 2.

Error Correction

- To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t+ 1$

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

Solution: This code guarantees the detection of up to three errors (s = 3), but it can correct up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

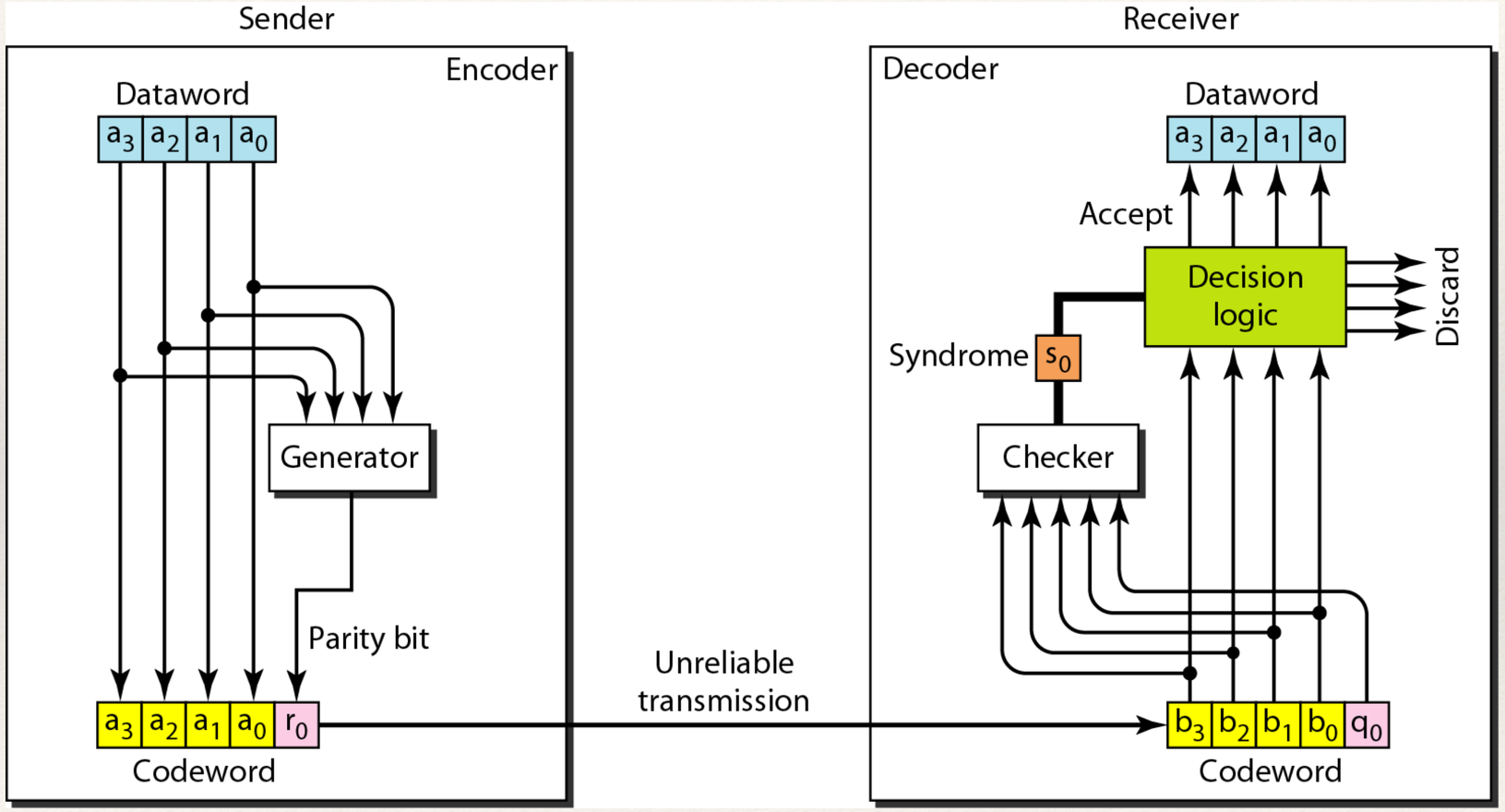
Linear Block Codes

- A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.
- A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.
- Even parity (ensures that a codeword has an even number of 1's) and odd parity (ensures that there are an odd number of 1's in the codeword)

Linear Block Codes/Parity Check

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Linear Block Codes/Parity Check



Example

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver. We examine five cases:

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.

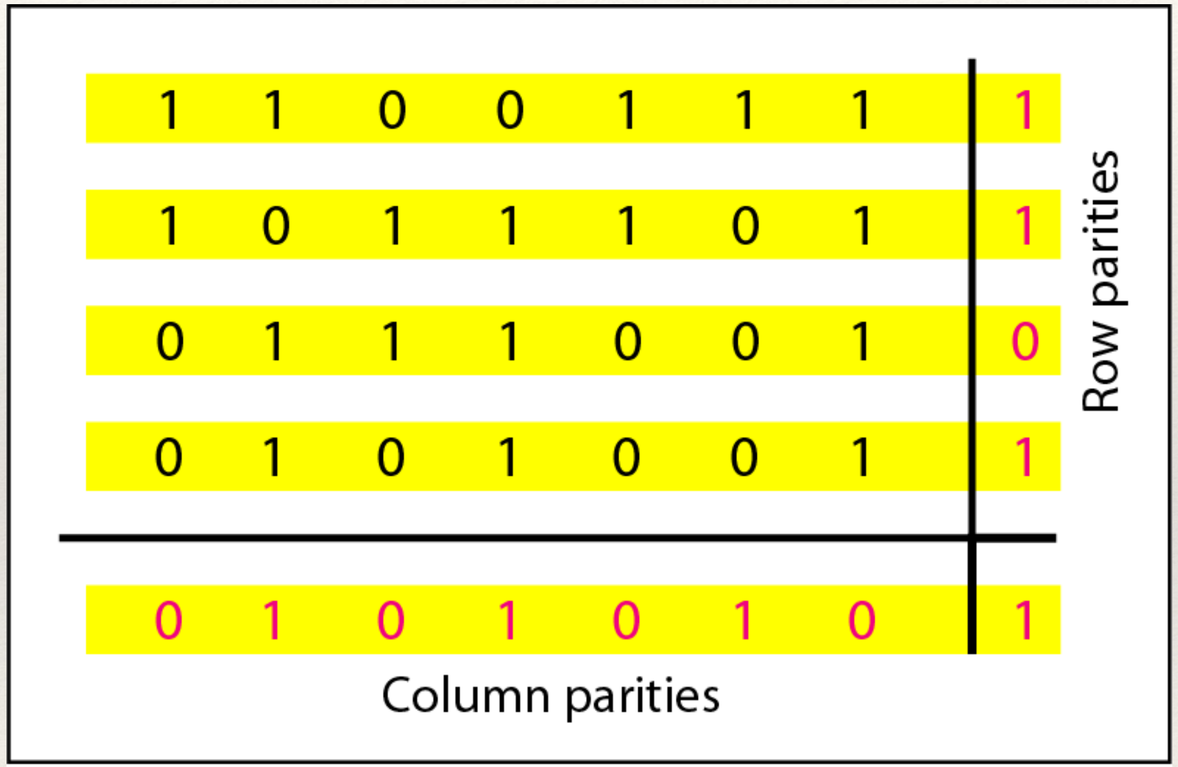
2. One single-bit error changes a1 . The received codeword is 10011. The syndrome is 1. No dataword is created.

3. One single-bit error changes r0 . The received codeword is 10110. The syndrome is 1. No dataword is created. 4. An error changes r0 and a second error changes a3 .The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value.

5. Three bits—a3, a2, and a1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

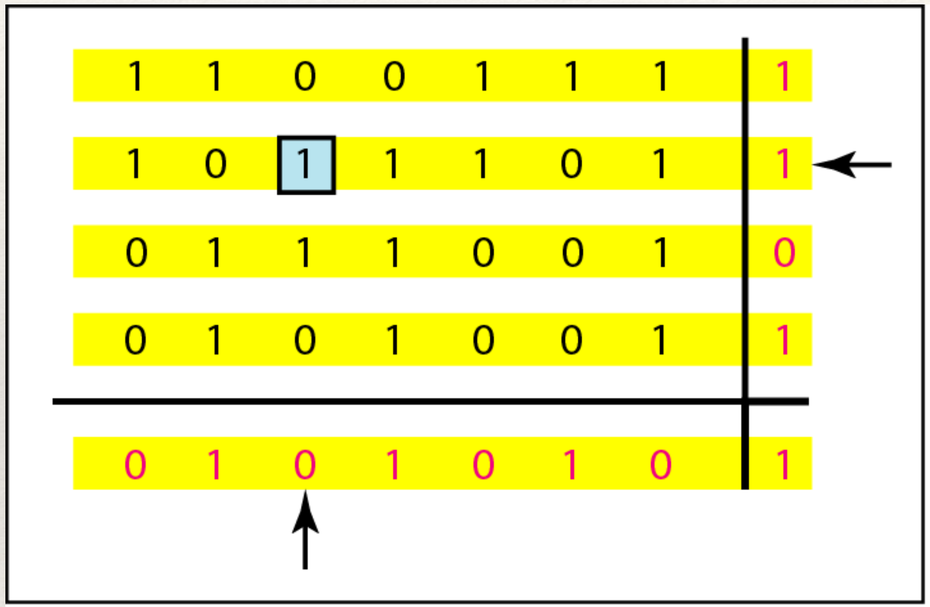
A simple Parity Check code can detect an odd number of errors

2-d Parity Check

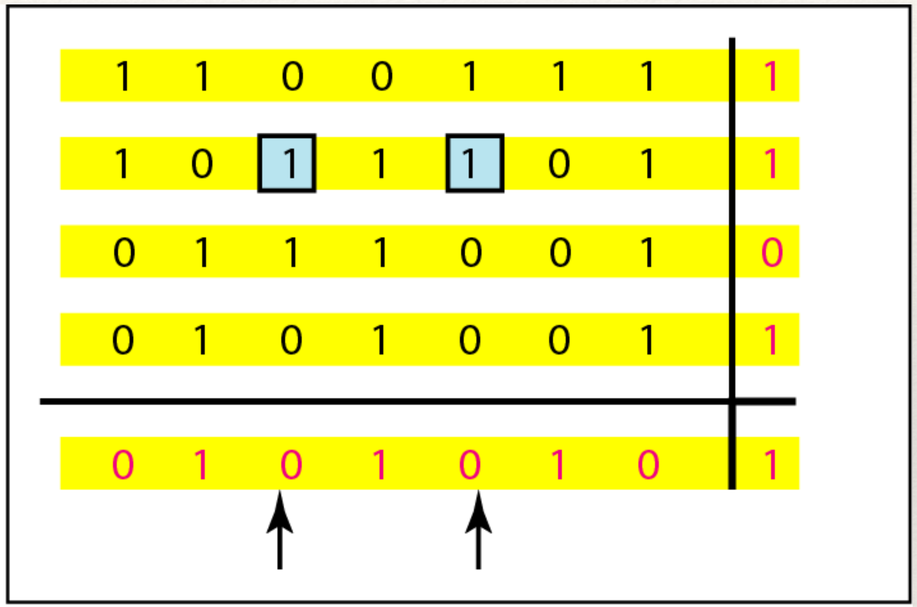


a. Design of row and column parities

2-d Parity Check

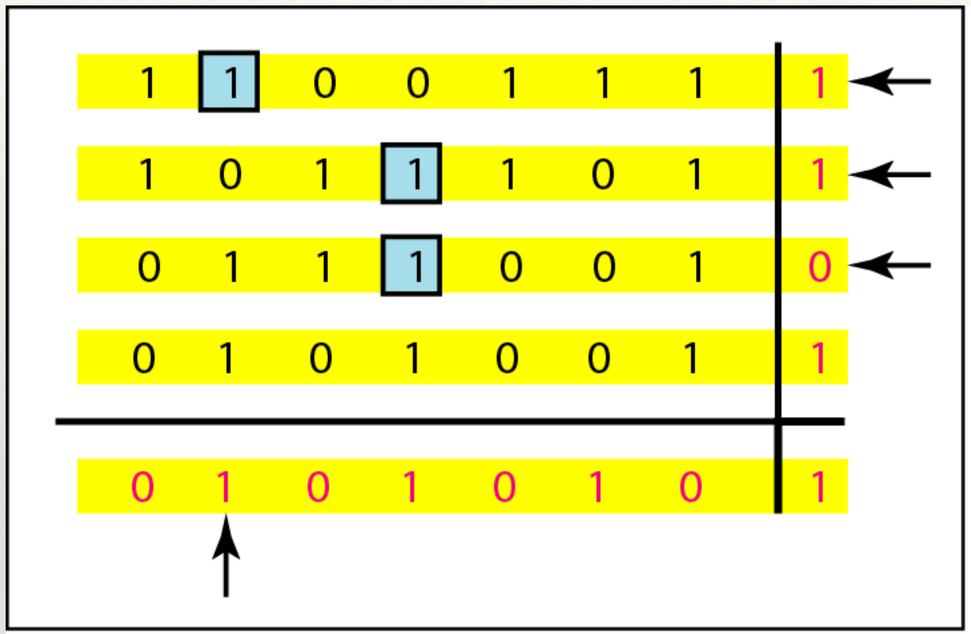


b. One error affects two parities

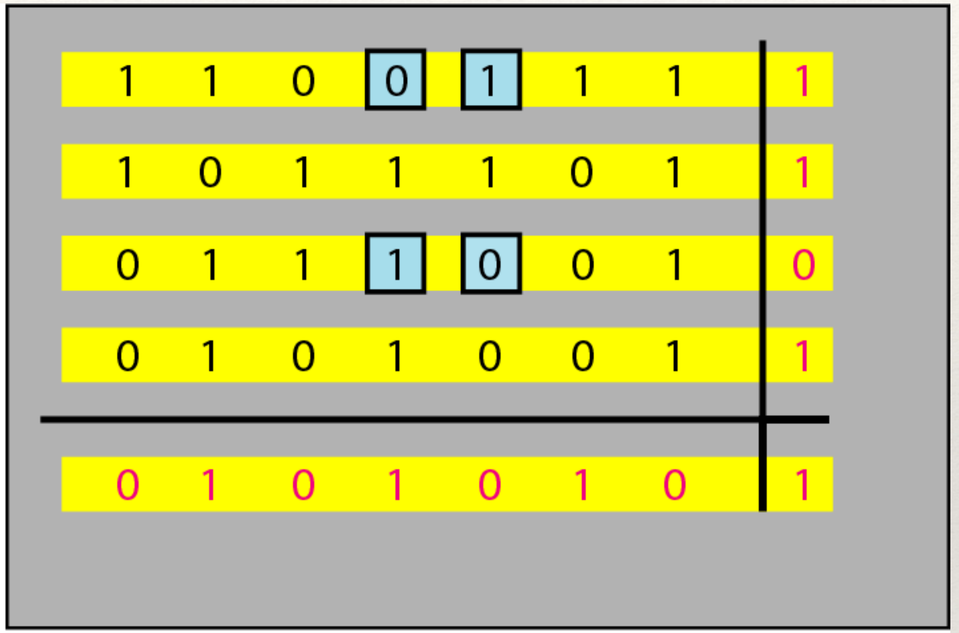


c. Two errors affect two parities

2-d Parity Check



d. Three errors affect four parities



e. Four errors cannot be detected

Using 2D parity check can detect upto 3bit errors anywhere in the table

Hamming Code

- Hamming code C(7, 4) ——> n=7, k = 4

These codes designed with d_{min}=3
Detect upto 2 errors and correct 1bit error

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

Hamming Code

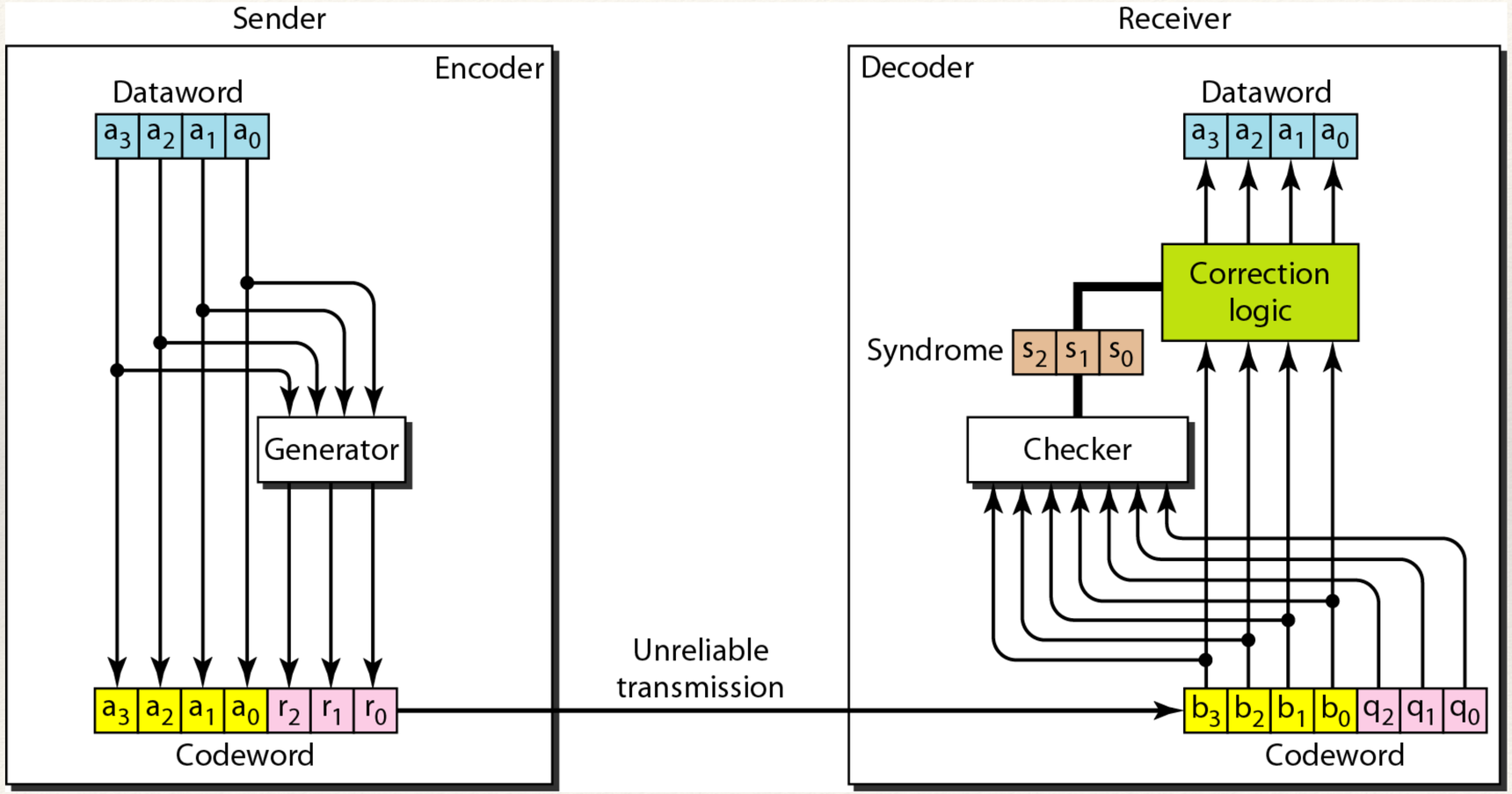


Fig. Structure of the Encoder and Decoder for the Hamming Code

Hamming Code

Calculating the parity bits at the transmitter

Modulo 2 arithmetic:

$r_0 = a_2 + a_1 + a_0$ modulo-2

$r_1 = a_3 + a_2 + a_1$ modulo-2

$r_2 = a_1 + a_0 + a_3$ modulo-2

Calculating the syndrome at the receiver:

$s_0 = b_2 + b_1 + b_0 + q_0$ modulo-2

$s_1 = b_3 + b_2 + b_1 + q_1$ modulo-2

$s_2 = b_1 + b_0 + b_3 + q_2$ modulo-2

Hamming Code

- Highlighted might suggest no error or errors the parity bit.
- First, if two errors occur during transmission, the created dataword might not be the right one.
- Second, if we want to use the above code for error detection, we need a different design

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

Fig. Logical decision made by correction logic Analyser of the Decoder

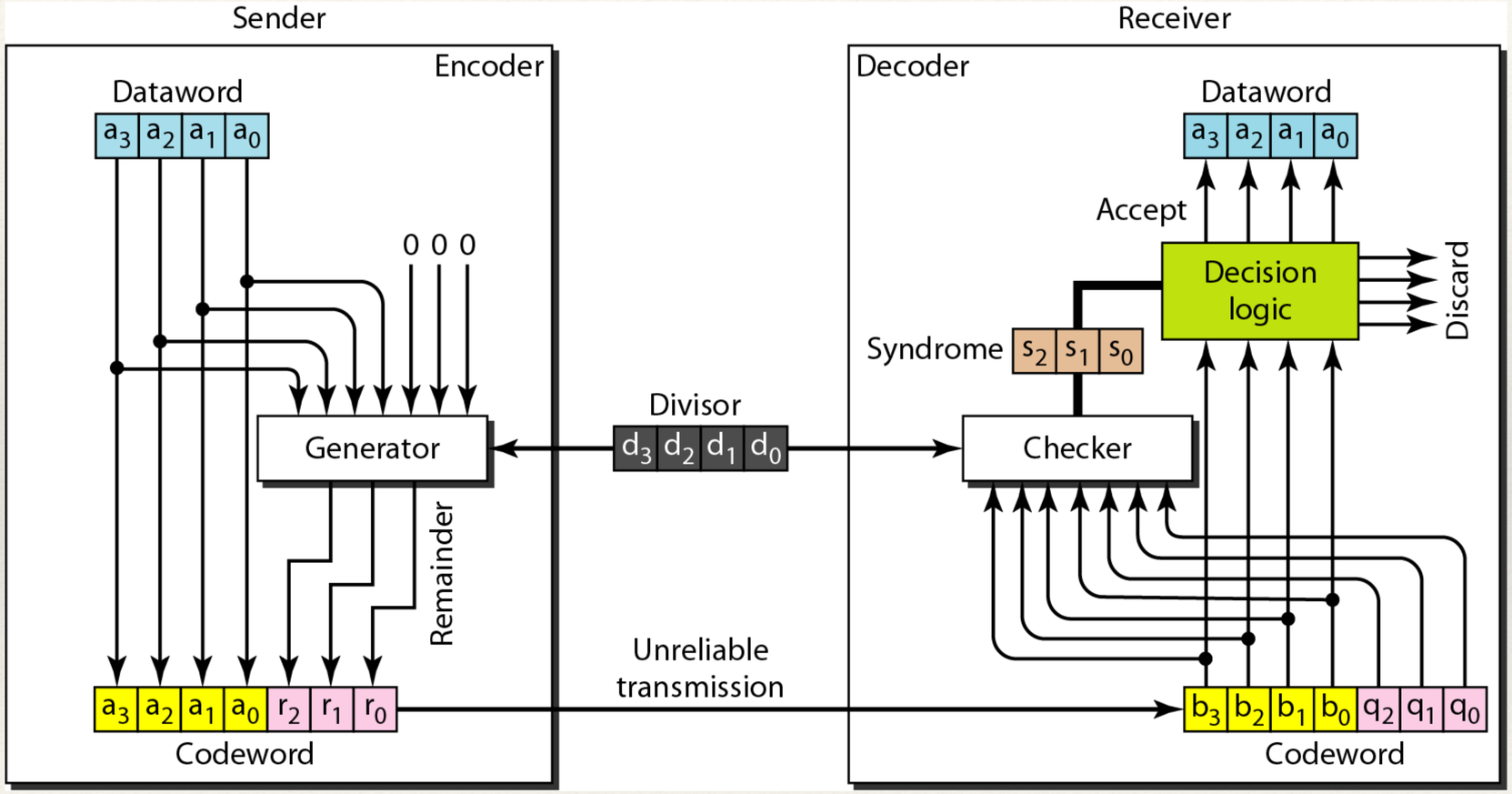
Burst-Errors

- Burst errors are very common, in particular in wireless environments where a fade will affect a group of bits in transit. The length of the burst is dependent on the duration of the fade.
- One way to counter burst errors, is to break up a transmission into shorter words and create a block (one word per row), then have a parity check per word.
- The words are then sent column by column. When a burst error occurs, it will affect 1 bit in several words as the transmission is read back into the block format and each word is checked individually.

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.

Cyclic-Code (CRC)

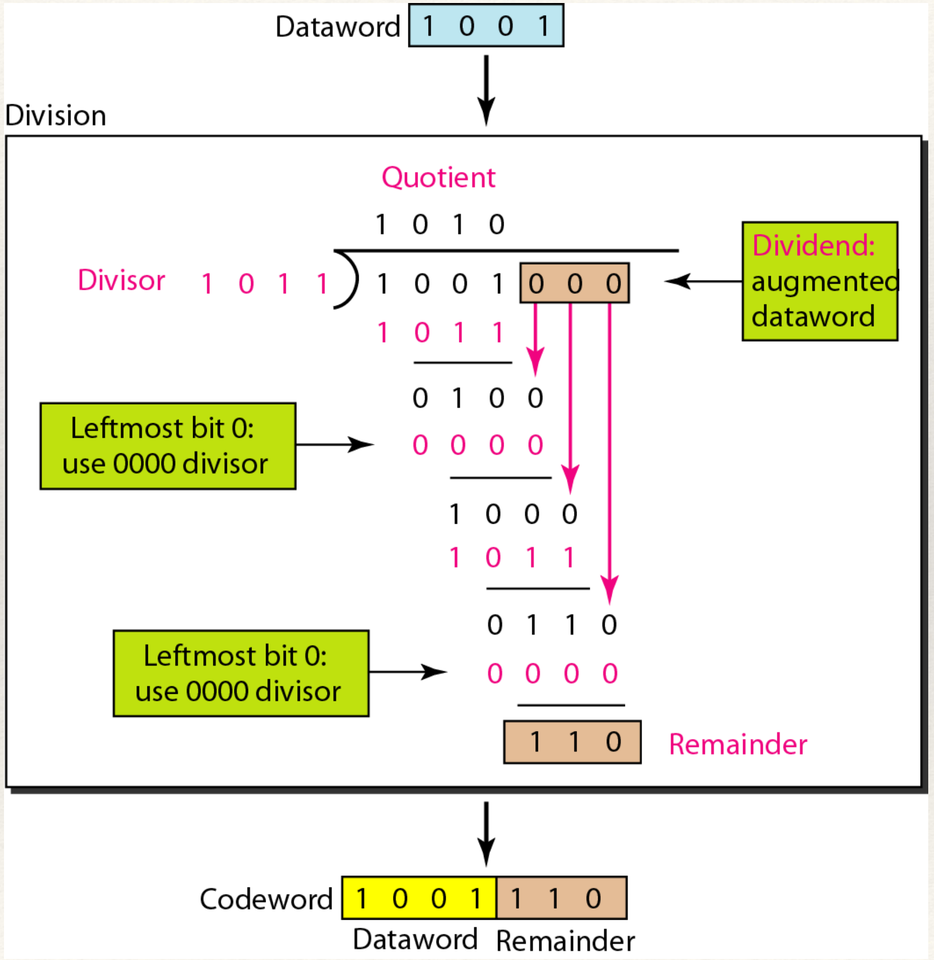


CRC -Example:

- Data=1001
- Divisor: 1011
- Perform and Check the CRC?

Note:
CRC using Polynomials can also de done.

CRC-Example


Dataword 1 0 0 1
Division
Quotient
1 0 1 0
Divisor 1 0 1 1
1 0 0 1 0 0 0
1 0 1 1
0 1 0 0
0 0 0 0
1 0 0 0
1 0 1 1
0 1 1 0
0 0 0 0
1 1 0
Remainder
Dividend: augmented dataword
Leftmost bit 0: use 0000 divisor
Leftmost bit 0: use 0000 divisor
Codeword 1 0 0 1 1 1 0
Dataword
Remainder

CRC Using Polynomials

- We can use a polynomial to represent a binary word.
- Each bit from right to left is mapped onto a power term.
- The rightmost bit represents the “0” power term. The bit next to it the “1” power term, etc.
- If the bit is of value zero, the power term is deleted from the expression.

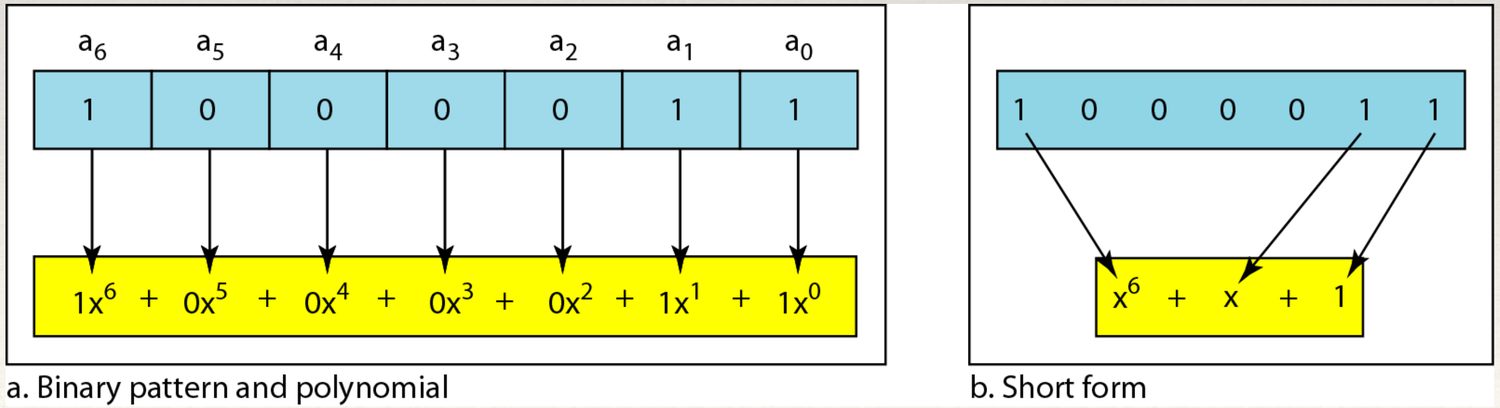
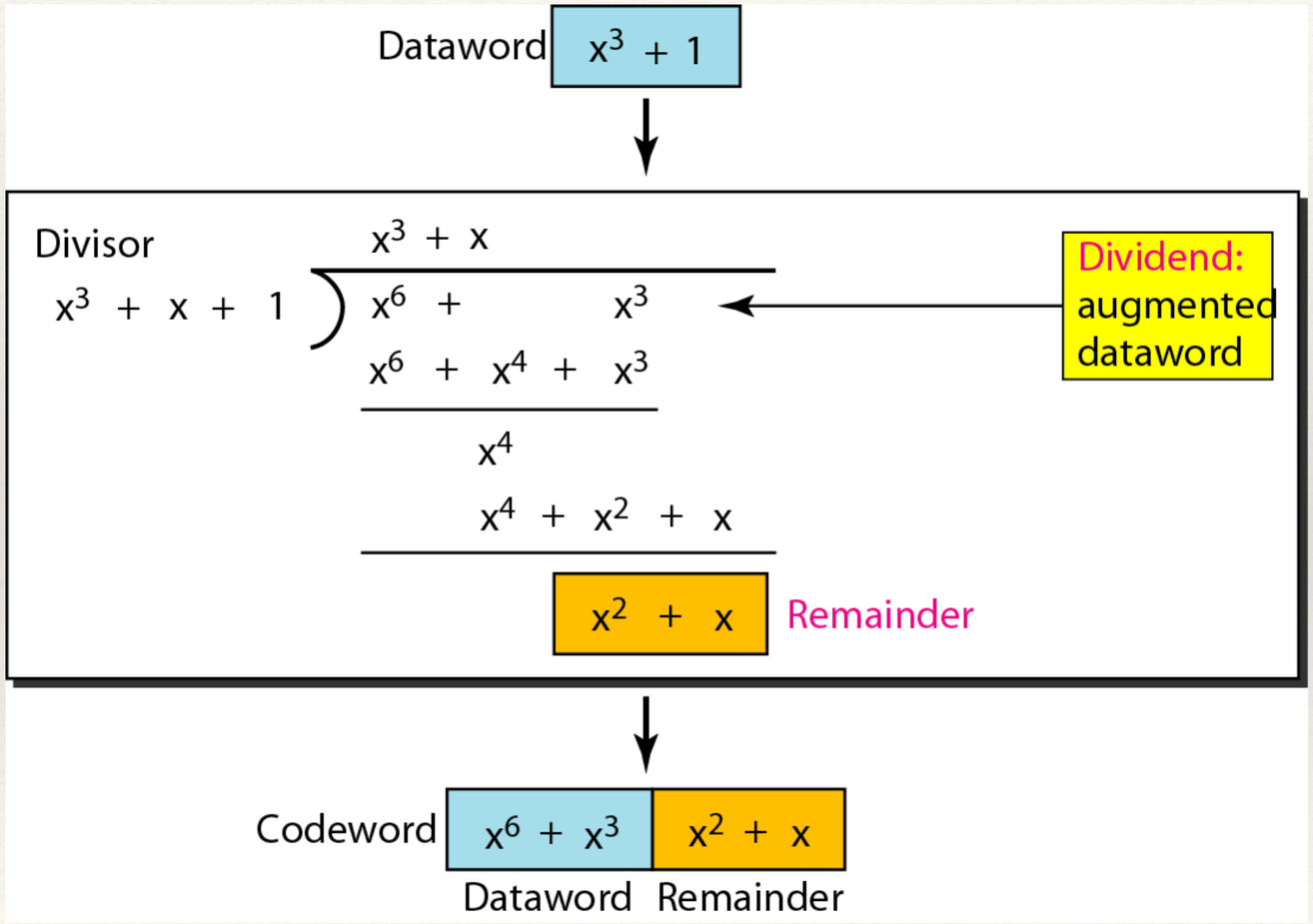


Fig. Polynomial Representation

Polynomials



- The divisor in a cyclic code is normally called the generator polynomial or simply the generator.
- In a cyclic code,
  - If $s(x) \neq 0$, one or more bits is corrupted.
  - If $s(x) = 0$, either
    - No bit is corrupted. or
    - Some bits are corrupted, but the decoder failed to detect them.

CRC-Polynomials

- In a cyclic code, those $e(x)$ errors that are divisible by $g(x)$ are not caught.
- Received codeword $(c(x) + e(x))/g(x) = c(x)/g(x) + e(x)/g(x)$
- The first part is by definition divisible the second part will determine the error. If "0" conclusion -> no error occurred.
- Note: that could mean that an error went undetected.
- Note: If the generator has more than one term and the coefficient of x^0 is 1, all single errors can be caught.
- Note: If a generator cannot divide $x^t + 1$ (t between 0 and n – 1), then all isolated double errors can be detected.

CRC-Polynomials

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

Name	*Polynomial*	*Application*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Table: Standard Polynomials

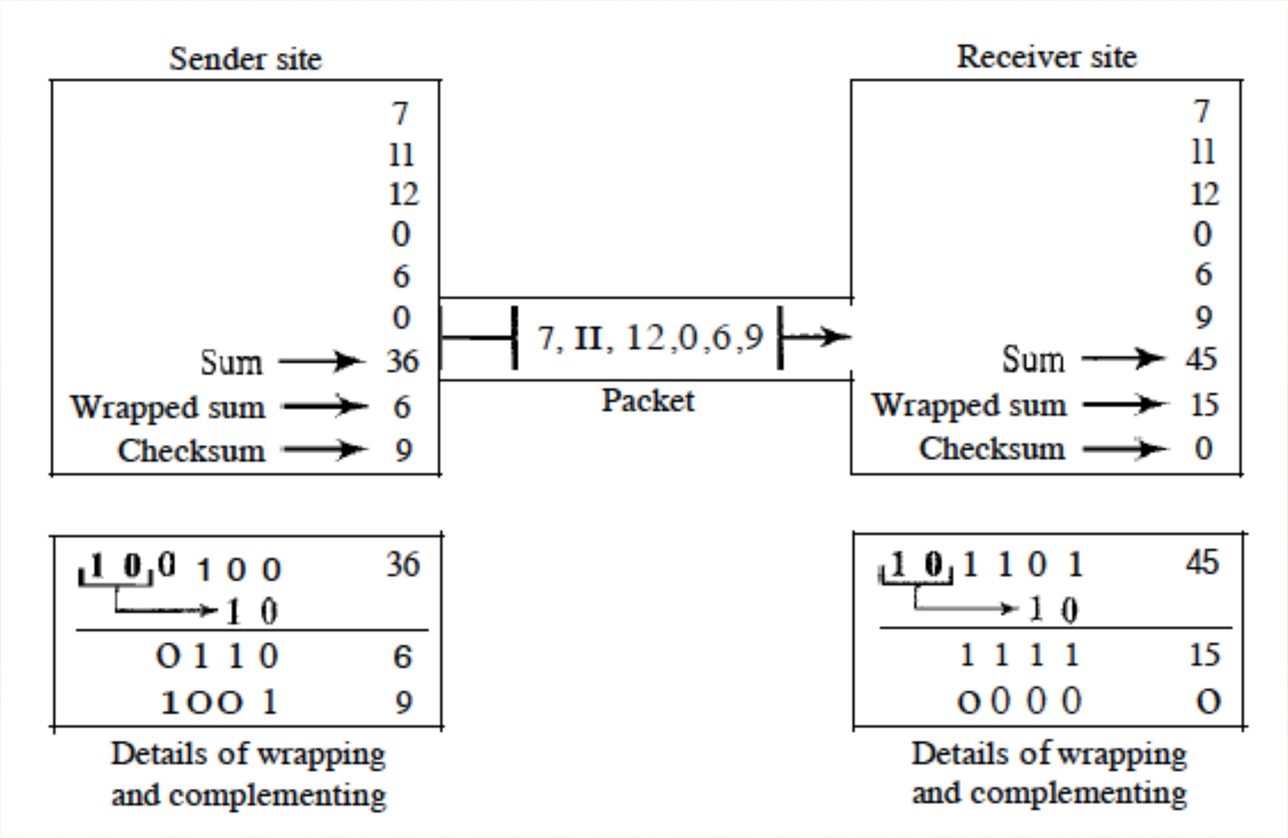
Checksum

- The last error detection method we discuss here is called the checksum/IP Checksum.
- The checksum is used in the Internet by several protocols although not at the data link layer.

Checksum Example

- Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers.
- For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers.
- The receiver adds the five numbers and compares the result with the sum.
- If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum.
- Otherwise, there is an error somewhere and the data are not accepted.

Checksum Example



Internet Checksum

Internet has been using a 16-bit checksum.

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.

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.

Internet Checksum Example

1. Let us calculate the checksum for a text of 8 characters (“Forouzan”).

Note: ASCII Values for A-Z: 65-90

ASCII Value for a-z is: 97 to 122

Internet Checksum Example

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	luz)
6	1	6	E	(an)
0	0	0	0	Checksum (initial)
8	F	C	6	Sum (partial)
			1	
8	F	C	7	Sum
7	0	3	8	Checksum (to send)

a. Checksum at the sender site

1	0	1	3	Carries
4	6	6	F	IFo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
7	0	3	8	Checksum (received)
F	F	F	E	Sum (partial)
			1	
F	F	F	F	Sum
0	0	0	0	Checksum (new}

b. Checksum at the receiver site

Fig: Internet Checksum Example

Flow Control

- Framing: The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
- The two types of framing are
  - Fixed Size Framing: No need for defining the boundaries
  - Variable Size Framing: Need a way to define the end of the frame and the start of the next.
- Frame Structure

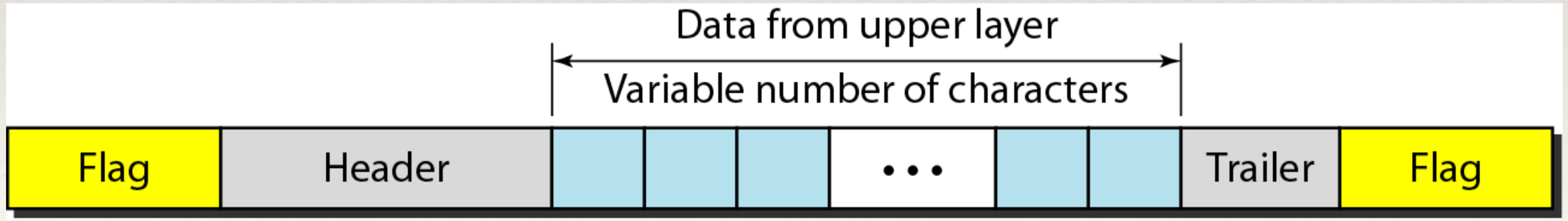


Fig. A frame in a character-oriented protocol

Character Oriented Protocol

- In Character oriented protocol data to be carried out are 8-bit characters.
- To Separate one frame from the next an 8-bit flag is added at the beginning and end of the frame.
- Frame Structure

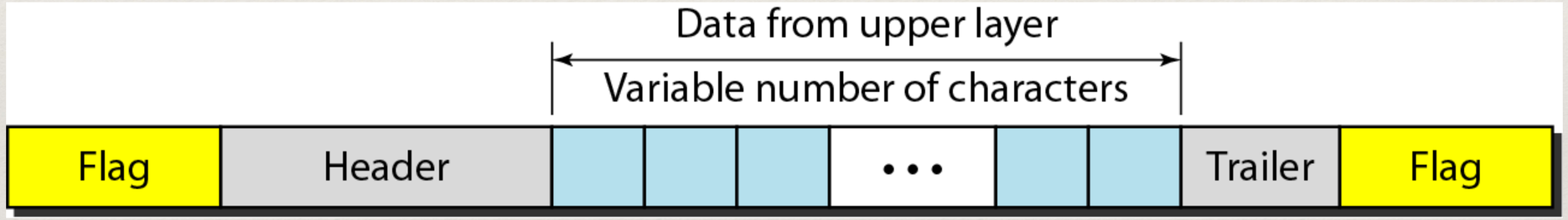


Fig. Frame Structure Character/Byte Oriented Protocol

- What happens if data is similar to the flag type???
  - To handle this Byte stuffing strategy is used in the character oriented protocol.

Byte Stuffing

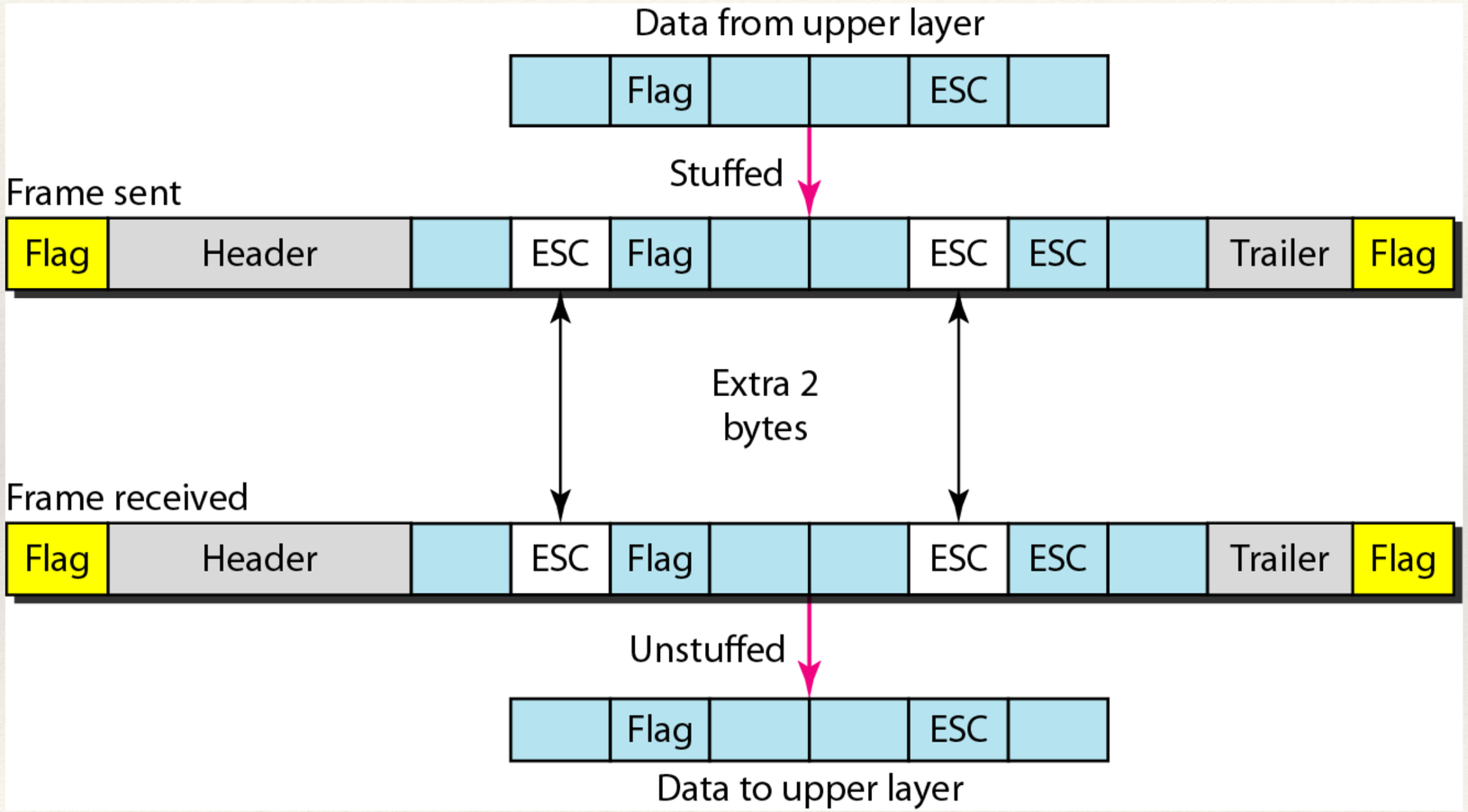


Fig: Byte Stuffing

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

The ESC character has a predefined bit pattern.

Bit-Oriented Protocols

- In bit-oriented protocol the data section of a frame is a sequence of bits to be interpreted by the upper layers.
- Most Protocol uses the a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and end of the frame.
- What if the data has a sequence of bits similar to the flag???

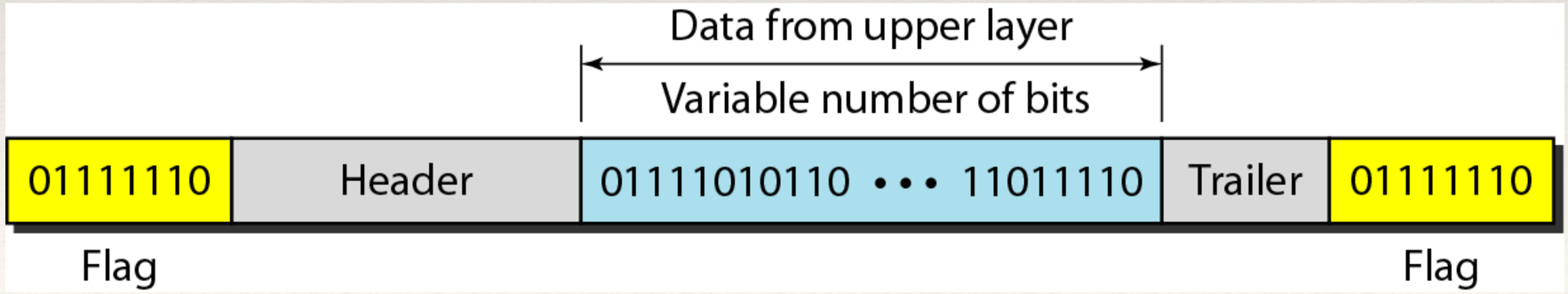


Fig. Frame in a Bit-oriented Protocol

Bit Stuffing

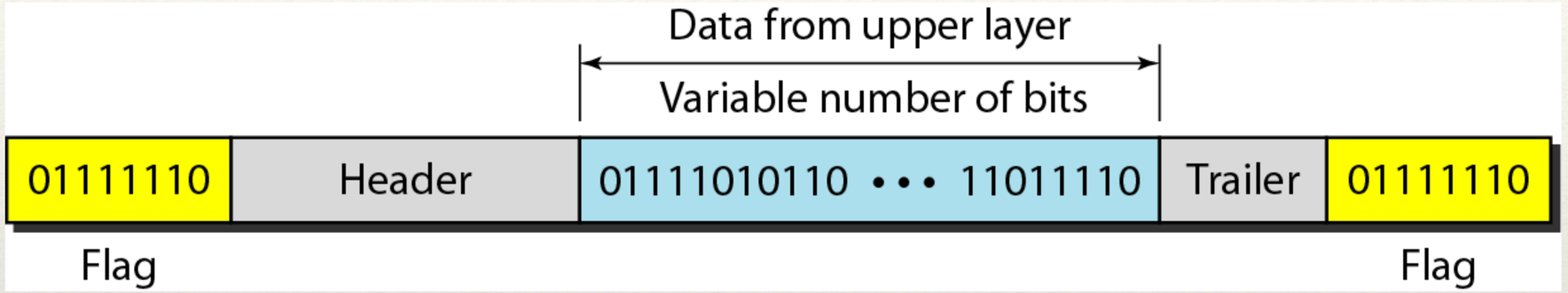


Fig. Bit Stuffing

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.

Example

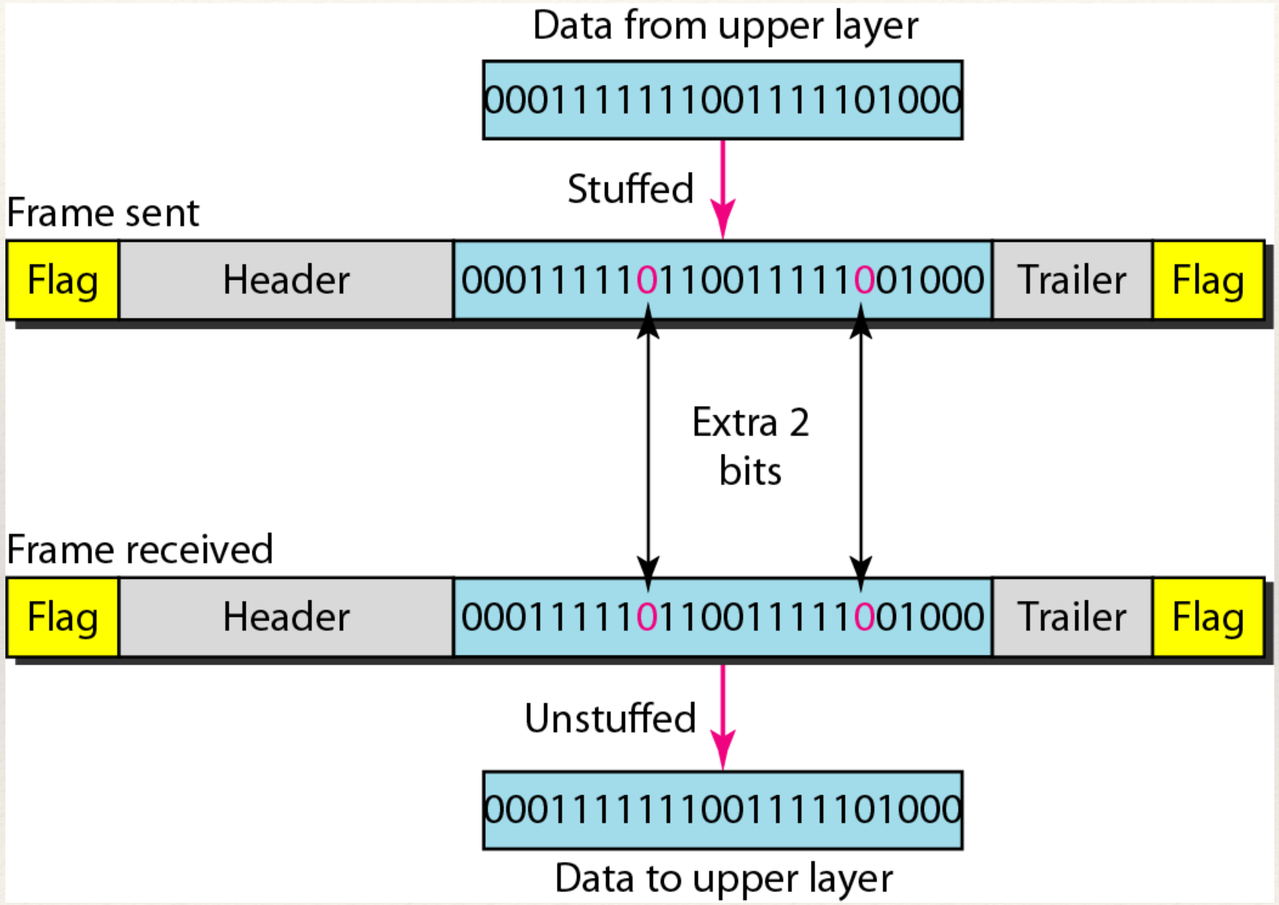


Fig. Bit Stuffing Example

Flow Control Techniques

- 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 (ARQ), which is the retransmission of data.
- Protocols used for handling the Flow Control are:

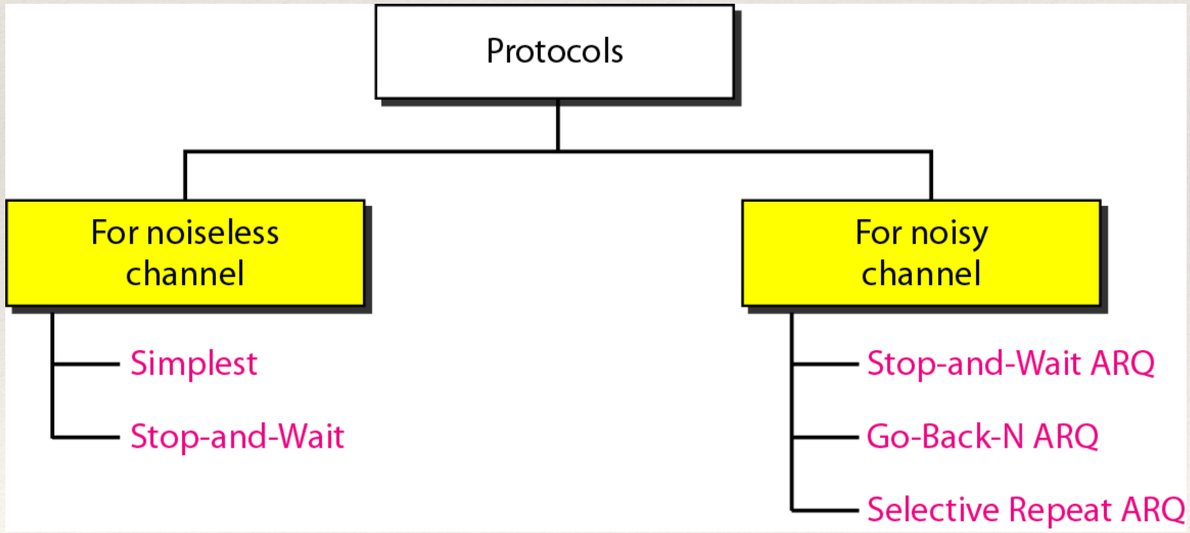


Fig. Taxonomy of protocols

Specified frames are retransmitted called as ARQ

Simplest Protocol

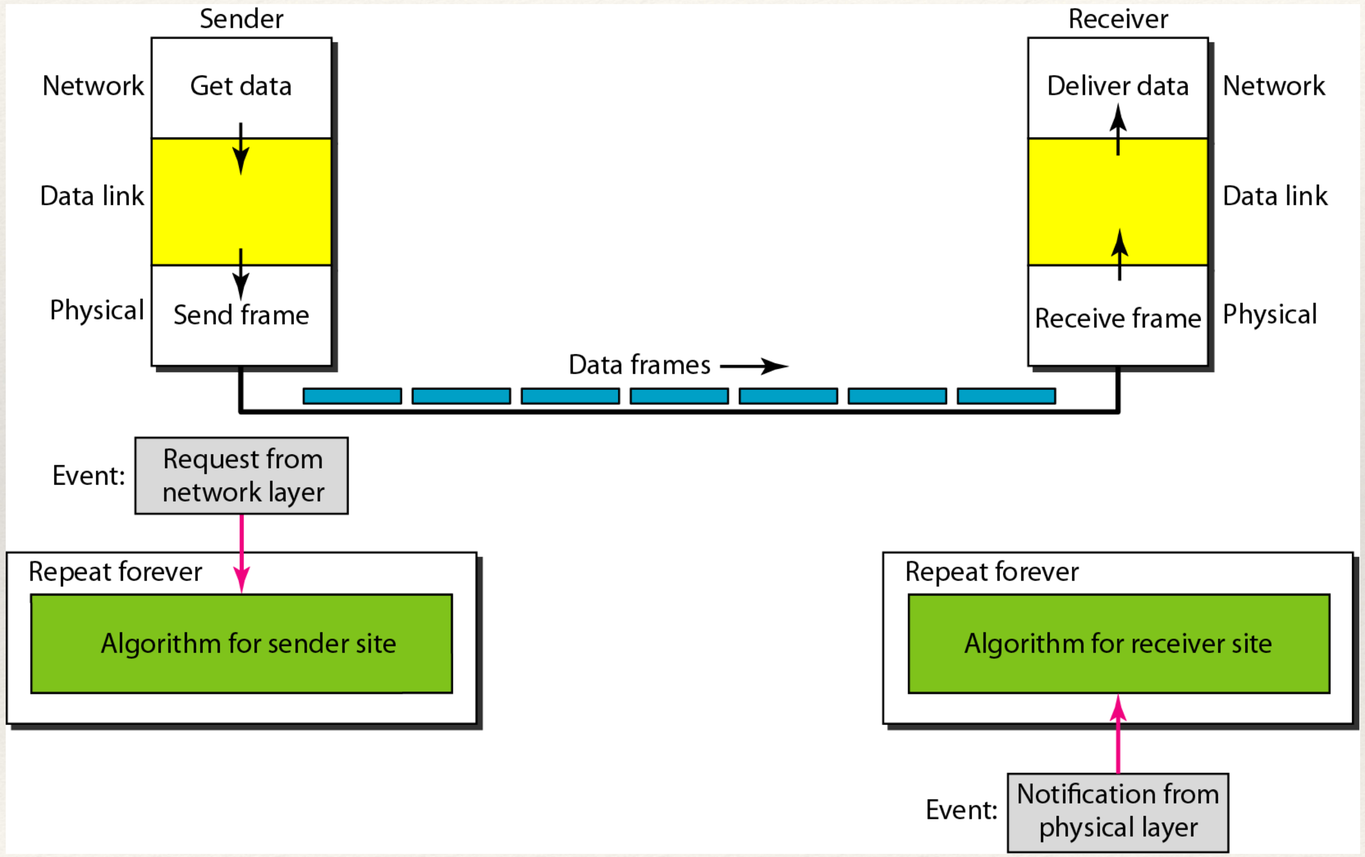


Fig. Simplest Protocol

- It is a unidirectional protocol in which data frames are traveling in only one direction.

Sender-site Algorithm

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6        GetData();
7        MakeFrame();
8        SendFrame();                   //Send the frame
9    }
10 }
```

Fig. Sender-site Algorithm

Receiver-site Algorithm

```
1  while(true)                                  // Repeat forever
2  {
3    WaitForEvent();                            // Sleep until an event occurs
4    if(Event(ArrivalNotification))             //Data frame arrived
5    {
6        ReceiveFrame();
7        ExtractData();
8        DeliverData();                         //Deliver data to network layer
9    }
10 }
```

Fig. Receiver-site Algorithm

Example: Simplest Protocol

Example: 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.

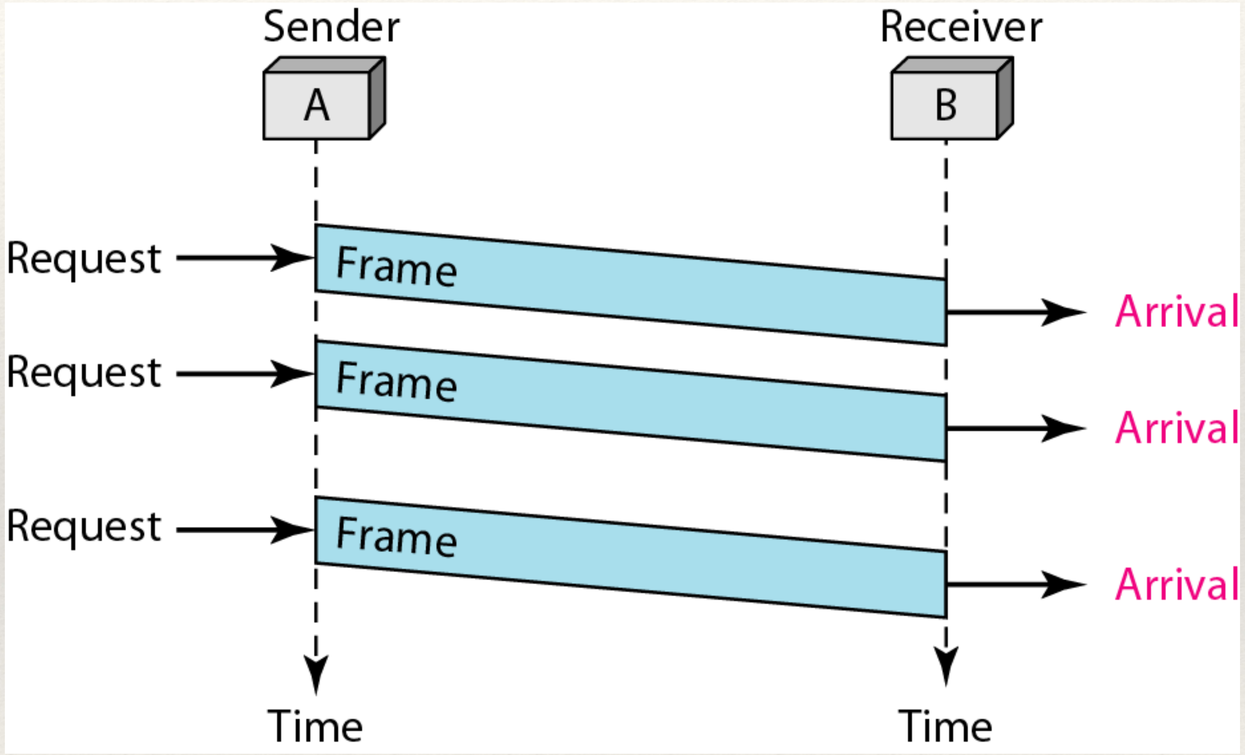


Fig. Flow Diagram

Stop-and-Wait Protocol

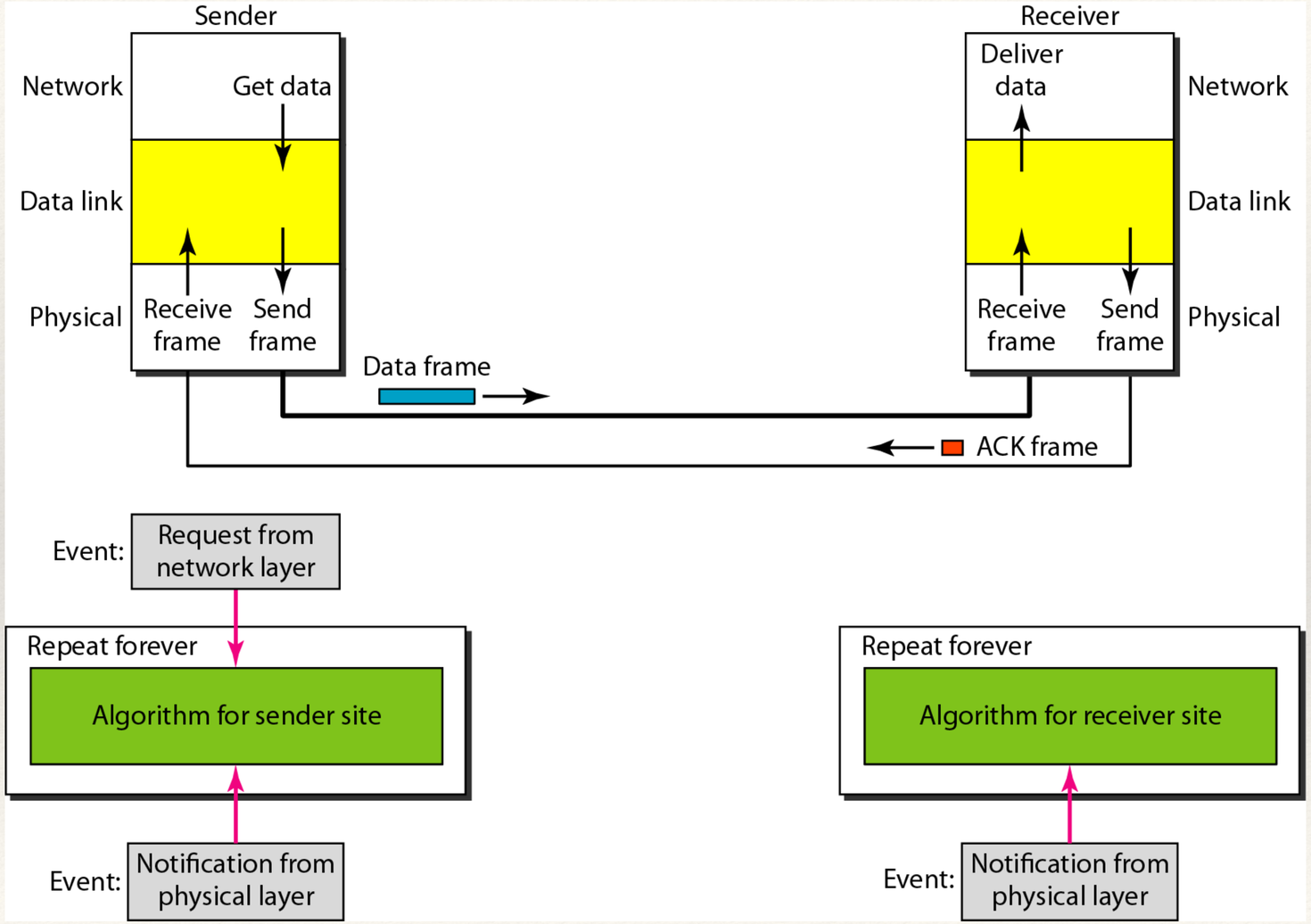


Fig. Design of Stop-and-Wait Protocol

Algorithm: Sender-site

```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Fig. Sender-site Algorithm

Algorithm: Receiver-site

```
 1  while(true)                              //Repeat forever
 2  {
 3    WaitForEvent();                        // Sleep until an event occurs
 4    if(Event(ArrivalNotification))         //Data frame arrives
 5    {
 6       ReceiveFrame();
 7       ExtractData();
 8       Deliver(data);                      //Deliver data to network layer
 9       SendFrame();                        //Send an ACK frame
10    }
11  }
```

Fig. Receiver-site Algorithm

Example: Stop-and-Wait Protocol

Example: 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

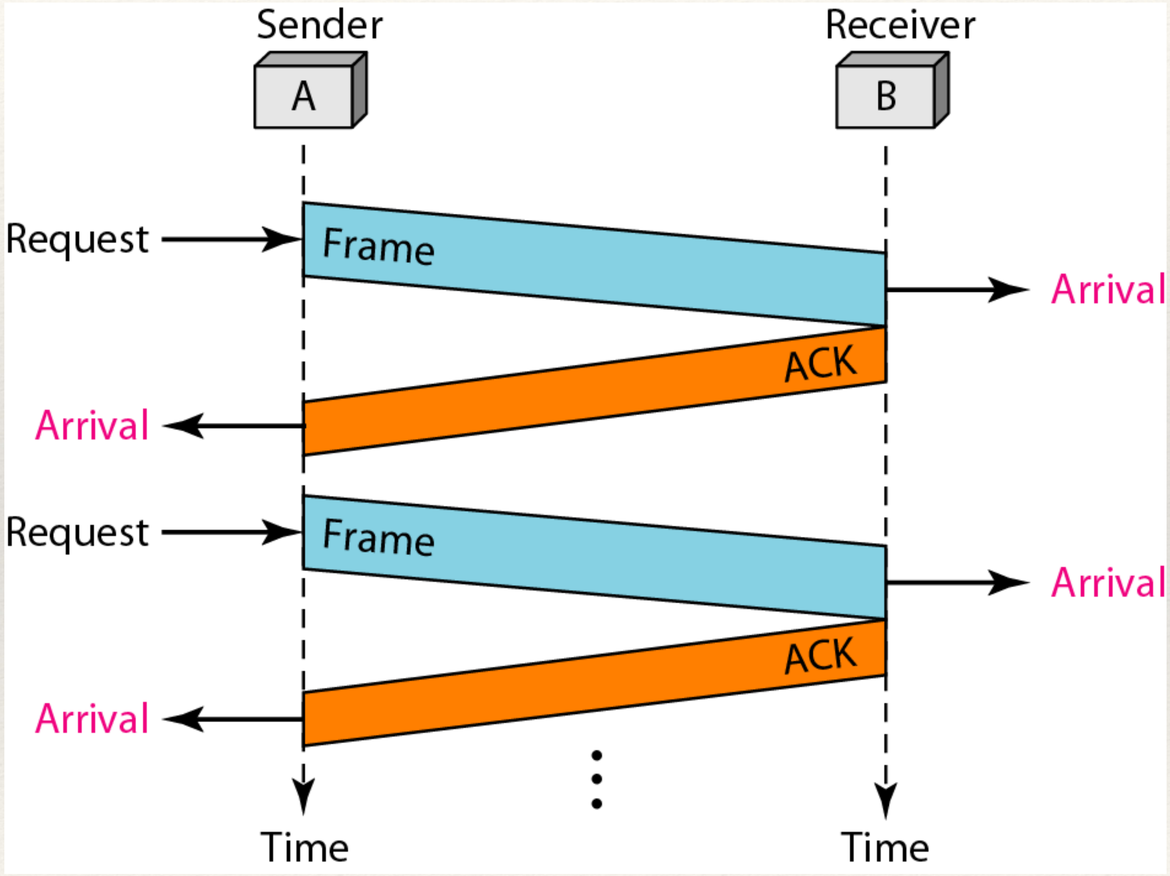


Fig. Flow-Diagram

Noisy-Channel Protocol: Stop-and-Wait ARQ

- 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.
- The sequence numbers are based on modulo-2 arithmetic.
- In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Stop-and-Wait ARQ

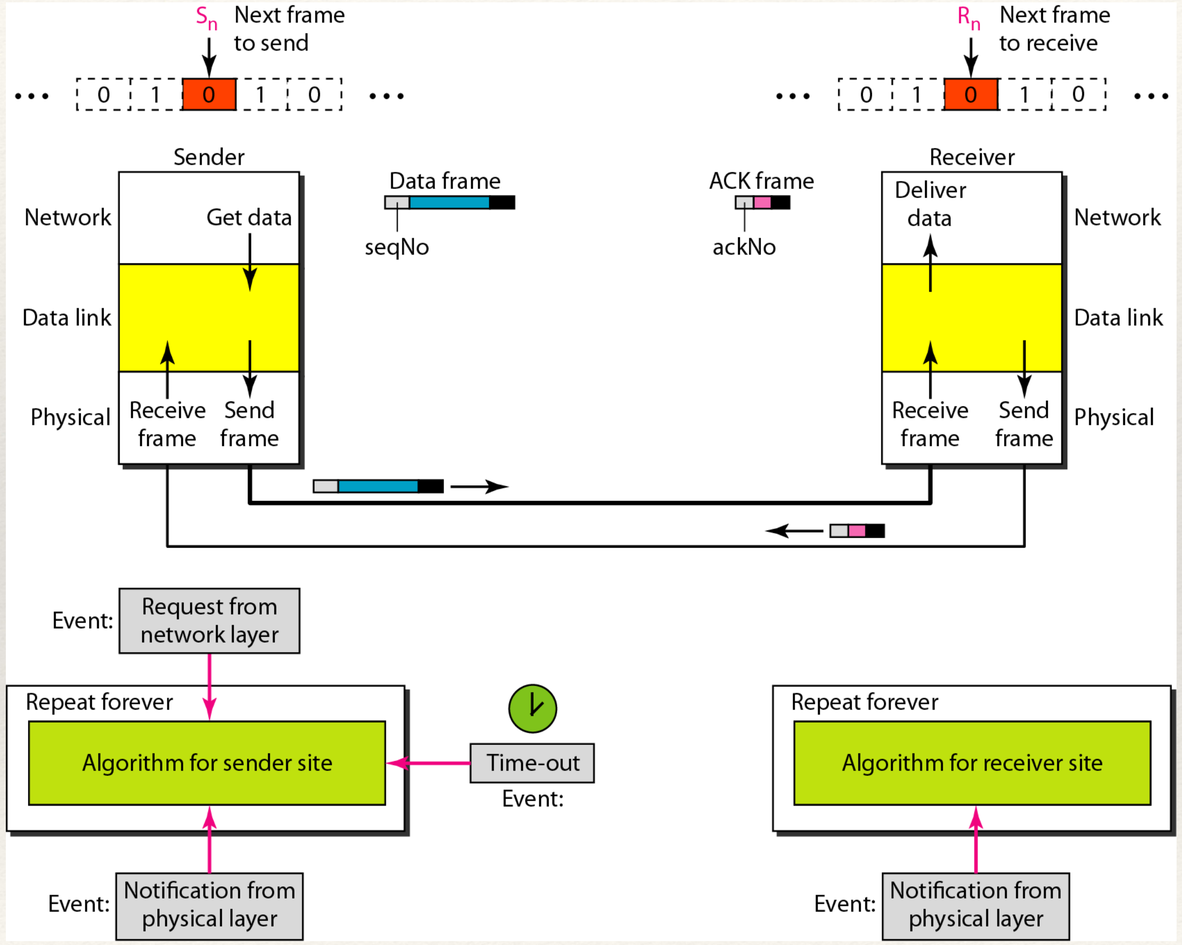


Fig. Design of Stop-and-wait ARQ Protocol

Sender-site Algorithm

```
1  Sn = 0;                          // Frame 0 should be sent first
2  canSend = true;                  // Allow the first request to go
3  while(true)                      // Repeat forever
4  {
5    WaitForEvent();                // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                       //The seqNo is Sn
10      StoreFrame(Sn);                      //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                         // Sleep
```

```
17      if(Event(ArrivalNotification)        // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);               //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)  //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);              //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                    // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                 //Resend a copy check
32      }
33  }
```

Receiver-site Algorithm

```
 1  Rn = 0;                          // Frame 0 expected to arrive first
 2  while(true)
 3  {
 4    WaitForEvent();                // Sleep until an event occurs
 5    if(Event(ArrivalNotification))  //Data frame arrives
 6    {
 7       ReceiveFrame();
 8       if(corrupted(frame));
 9          sleep();
10       if(seqNo == Rn)              //Valid data frame
11       {
12        ExtractData();
13         DeliverData();             //Deliver data
14         Rn = Rn + 1;
15       }
16        SendFrame(Rn);             //Send an ACK
17    }
18  }
```

Fig. Receiver-site Algorithm

Example: Stop-and-Wait ARQ

Figure 11.11 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.

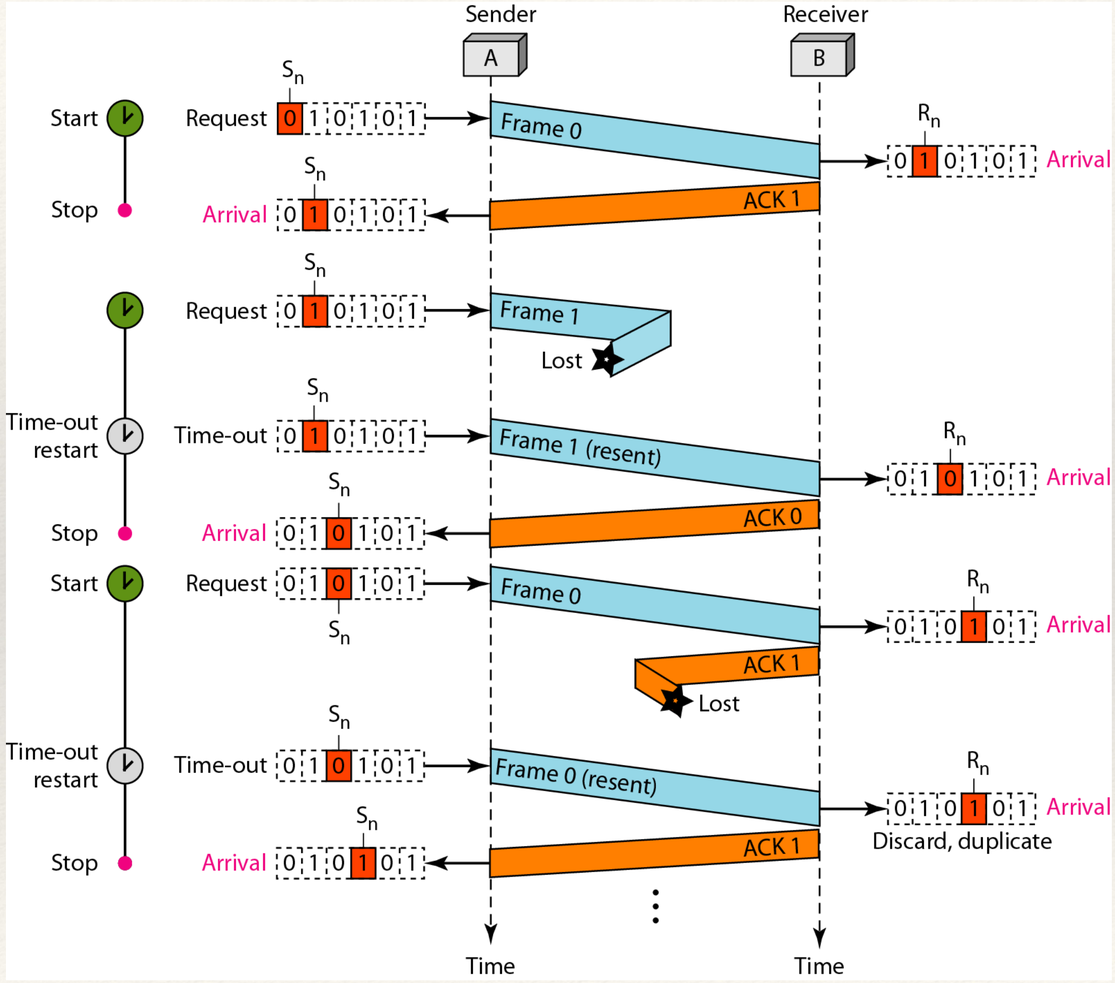


Fig. Flow diagram for the Example

Sender Window: Go-Back-N Protocol

- In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- 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}.
- The send window can slide one or more slots when a valid acknowledgment arrives.

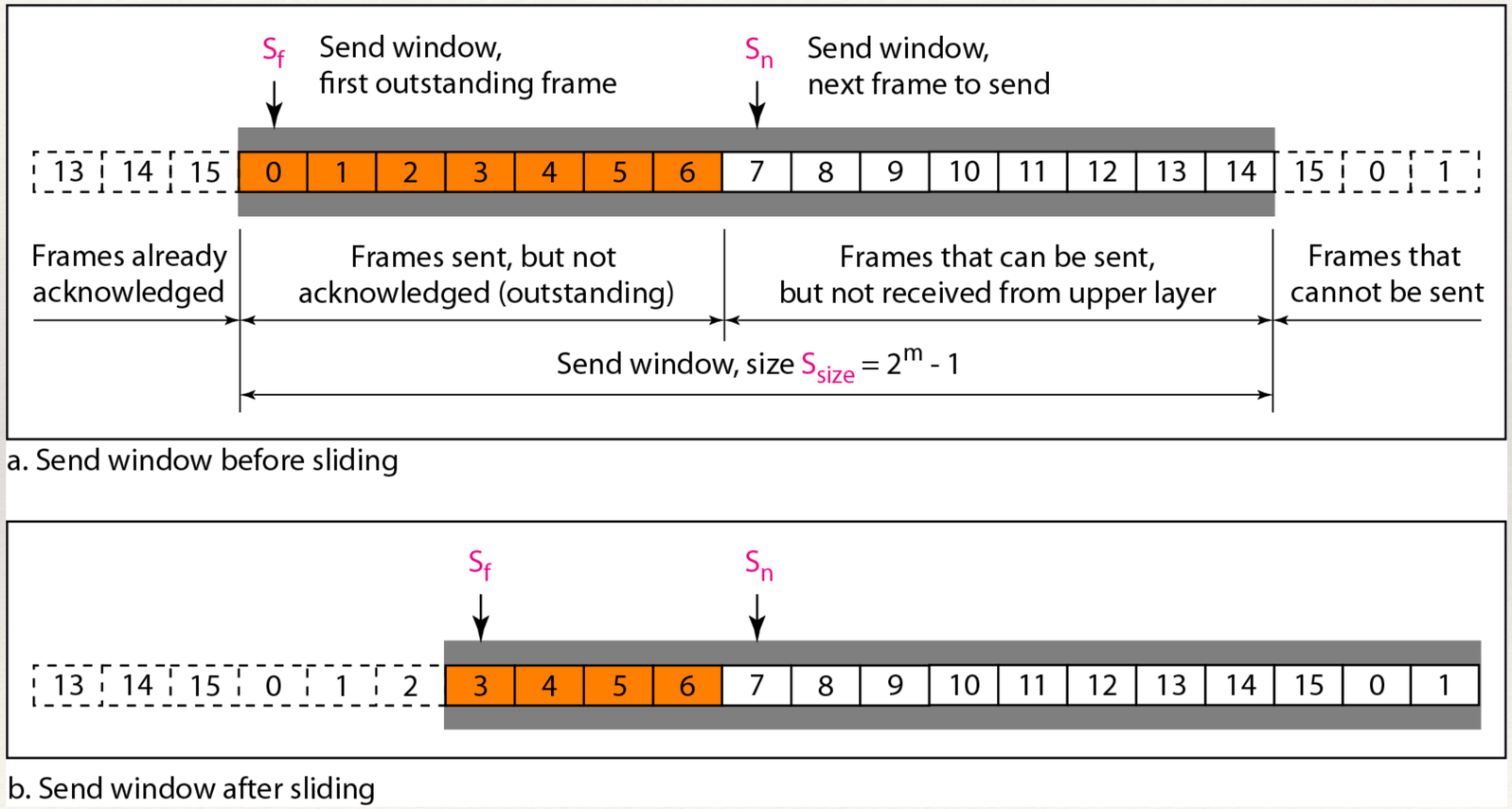


Fig. Sender Window in Go-back-N ARQ

Receiver Window: Go-back-N ARQ

- 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.
- If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting.

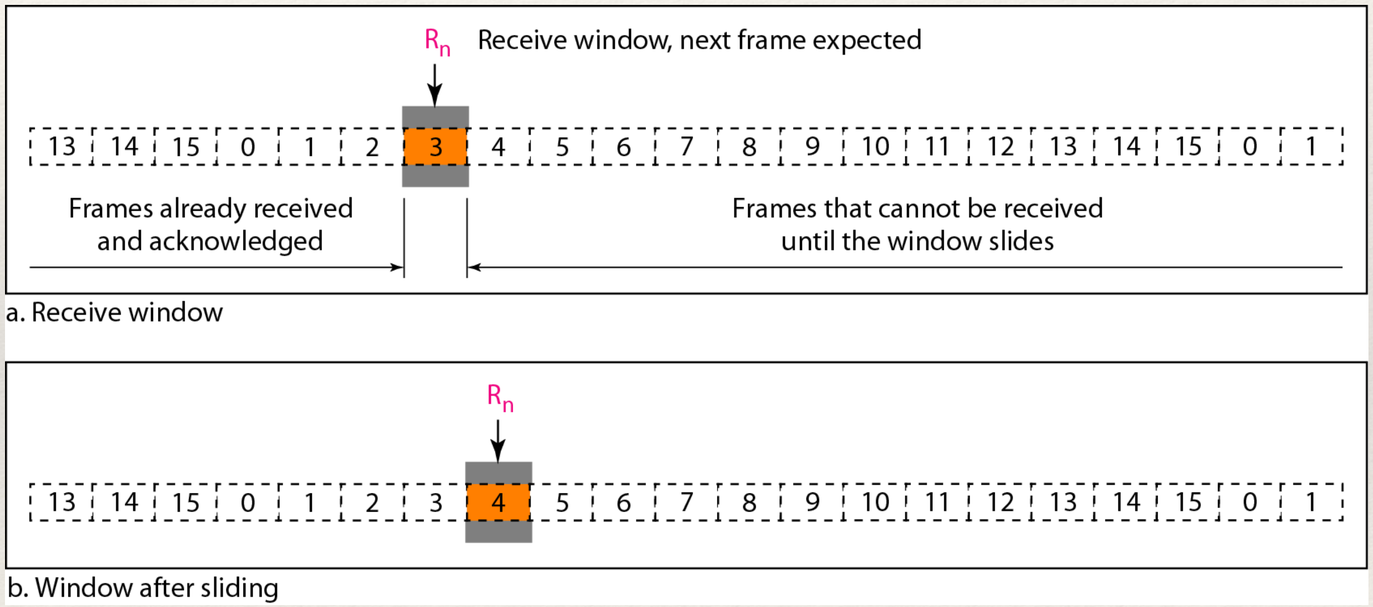


Fig. Receiver Window in Go-back-N ARQ

Window Size for Go-back-N ARQ

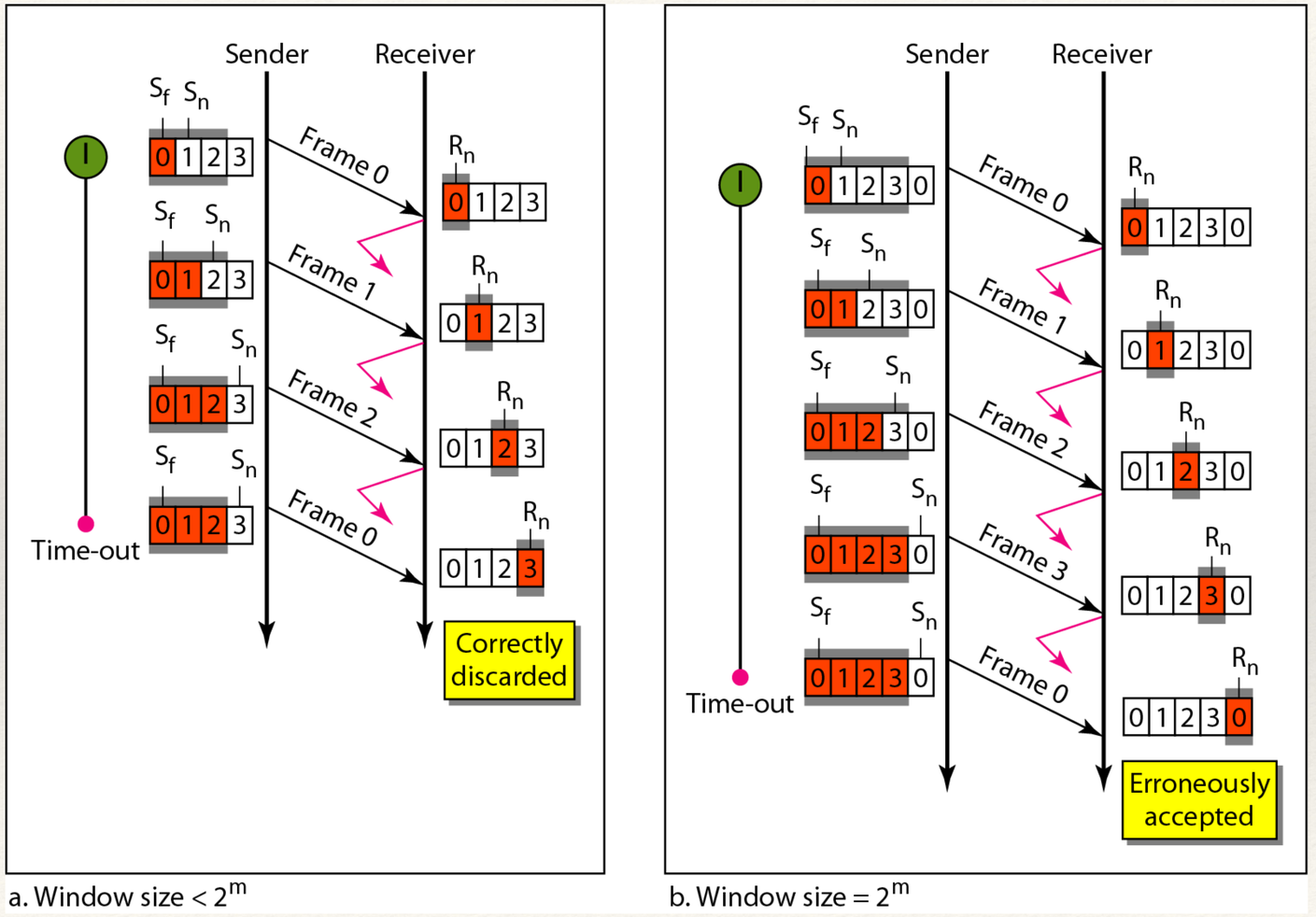


Fig. Window Size

Go-back-N ARQ: Design

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.

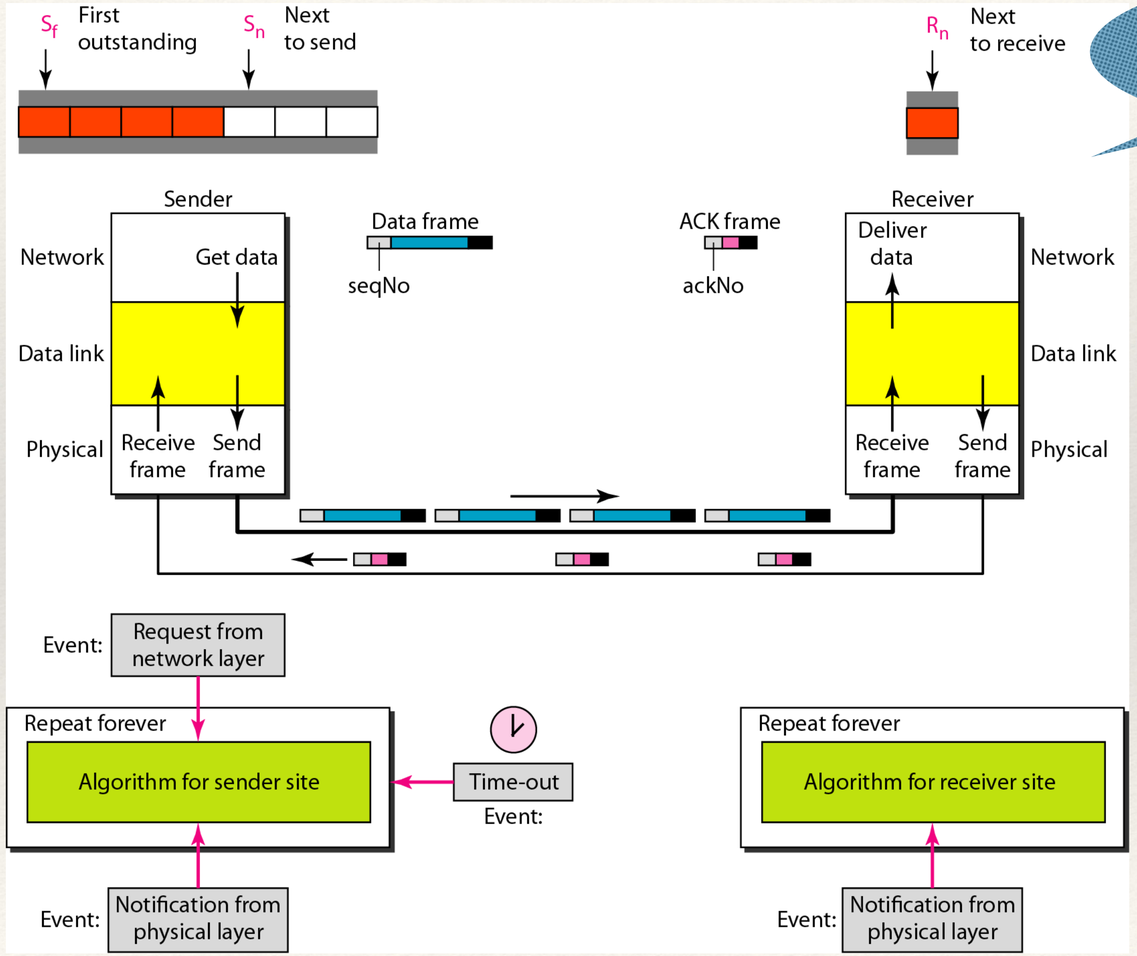


Fig. Go-back-N ARQ Design

Go-back-N ARQ: Sender-site Algorithm

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8     if(Event(RequestToSend))           //A packet to send
9     {
10       if(Sn-Sf >= Sw)                //If window is full
11             Sleep();
12       GetData();
13       MakeFrame(Sn);
14       StoreFrame(Sn);
15       SendFrame(Sn);
16       Sn = Sn + 1;
17       if(timer not running)
18            StartTimer();
19    }
20
```

Sender-site Algorithm (contd..)

```
21   if(Event(ArrivalNotification))  //ACK arrives
22   {
23      Receive(ACK);
24      if(corrupted(ACK))
25           Sleep();
26      if((ackNo>Sf)&&(ackNo<=Sn))  //If a valid ACK
27      While(Sf <= ackNo)
28       {
29        PurgeFrame(Sf);
30        Sf = Sf + 1;
31       }
32       StopTimer();
33   }
34
35   if(Event(TimeOut))               //The timer expires
36   {
37    StartTimer();
38    Temp = Sf;
39    while(Temp < Sn);
40     {
41      SendFrame(Sf);
42      Sf = Sf + 1;
43     }
44   }
45 }
```

Fig. Sender Site Algorithm for Go-back-N ARQ

Receiver-site Algorithm

```
 1  Rn = 0;
 2
 3  while (true)                          //Repeat forever
 4  {
 5    WaitForEvent();
 6
 7    if(Event(ArrivalNotification)) /Data frame arrives
 8    {
 9       Receive(Frame);
10       if(corrupted(Frame))
11            Sleep();
12       if(seqNo == Rn)                  //If expected frame
13       {
14         DeliverData();                 //Deliver data
15         Rn = Rn + 1;                   //Slide window
16         SendACK(Rn);
17       }
18    }
19  }
```

Fig. Receiver-site Algorithm

Example-1

- Figure shows an example of Go-Back-N.
- 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.
- After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer.
- There is no time-out event here because all outstanding frames are acknowledged before the timer expires.
- Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.

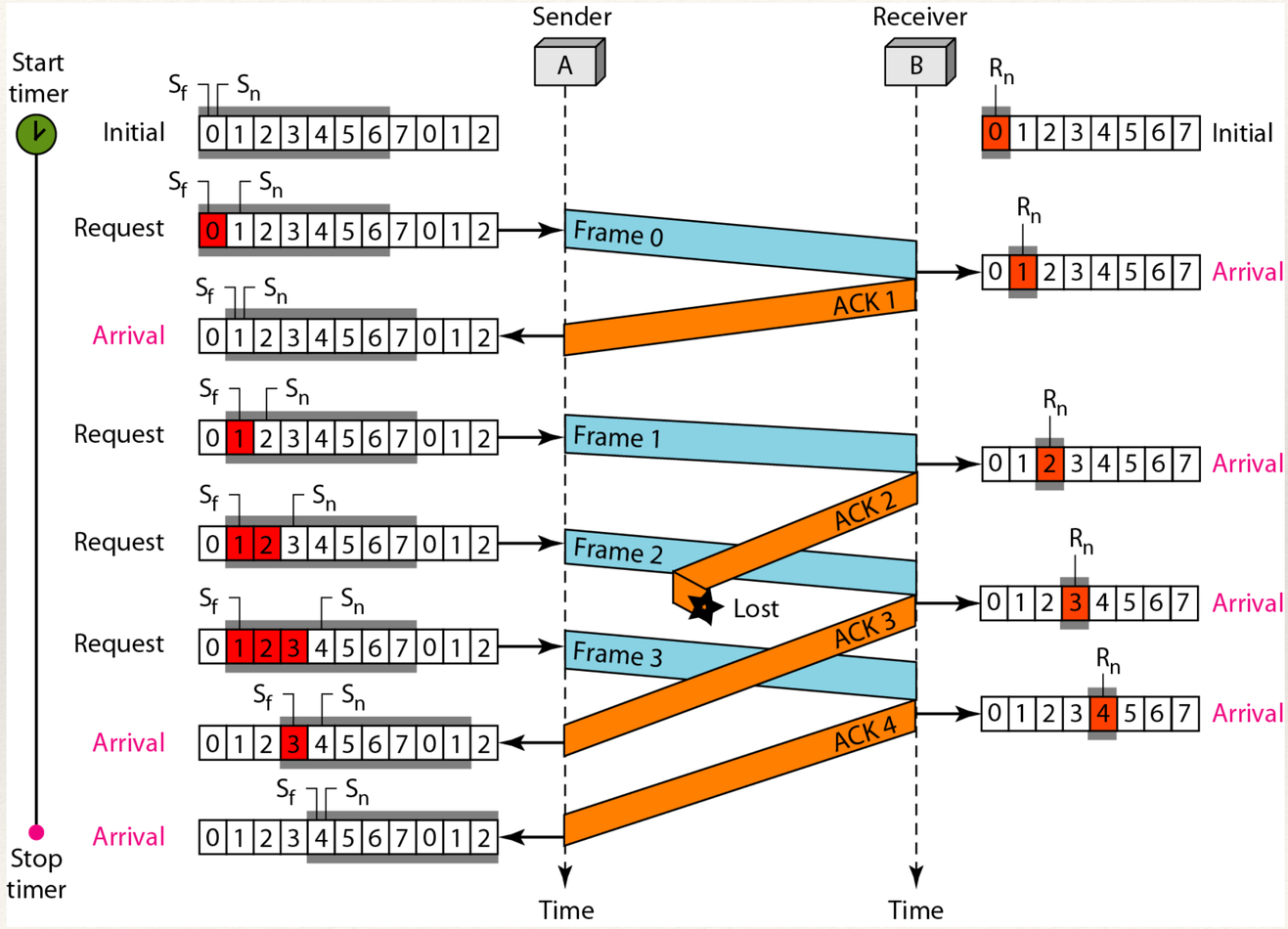


Fig. Flow Diagram -Go-back-N ARQ

Example-2

- Figure shows what happens when a frame is lost.
- Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost.
- The receiver receives frames 2 and 3, but they are discarded because they are received out of order.
- The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires.
- The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong.
- Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events.
- This means that when ACK 2 arrives, the sender is still busy with sending frame 3.

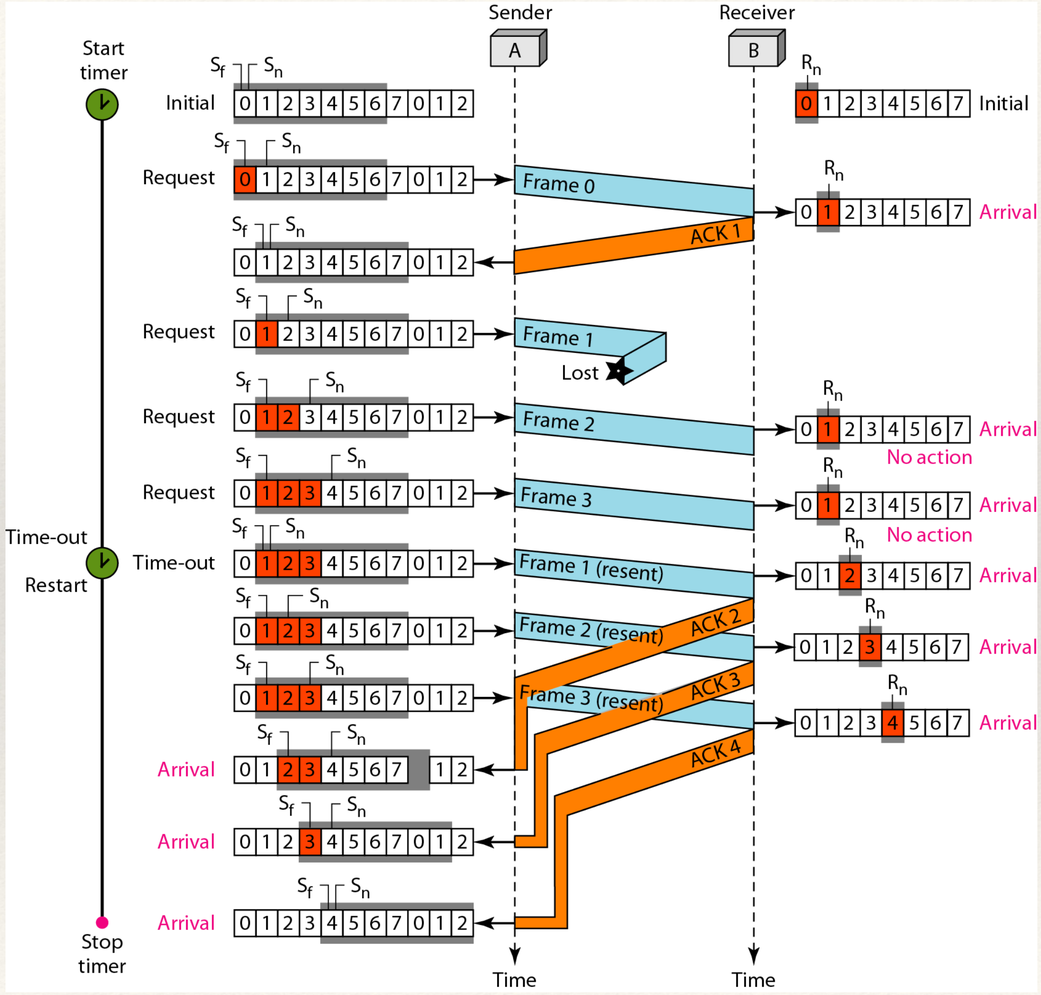


Fig. Flow Diagram Example 2

Sender Window: Selective Repeat ARQ

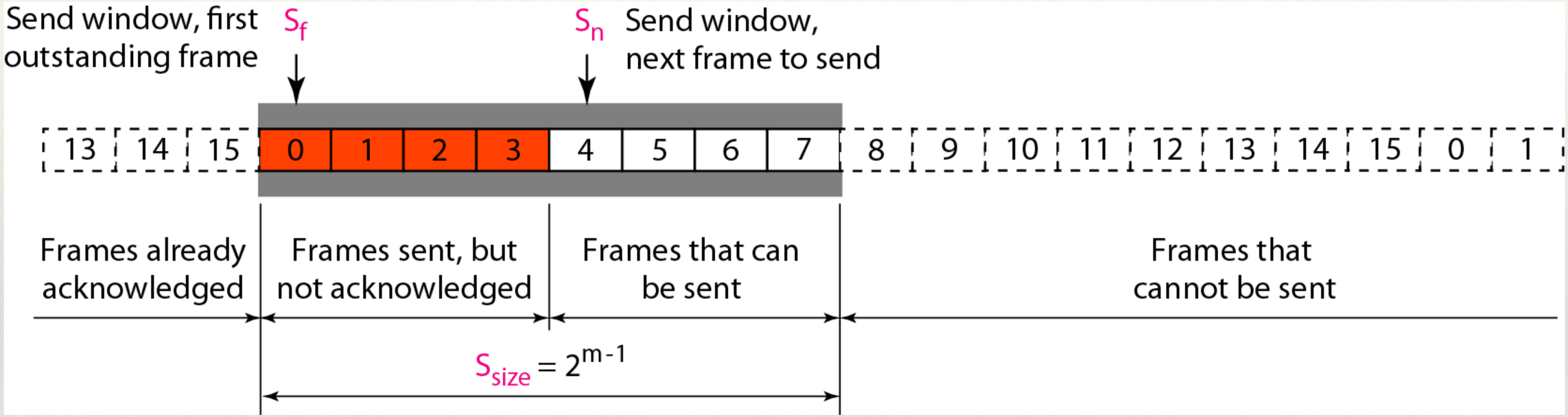


Fig. Sender Window for Selective Repeat ARQ

Receiver Window: Selective Repeat ARQ

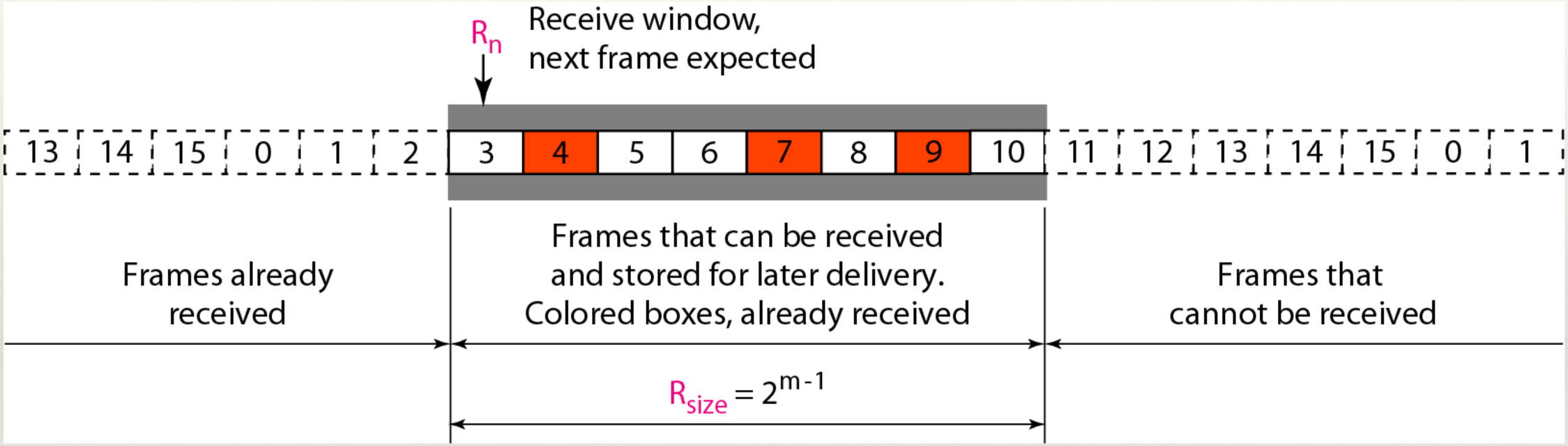
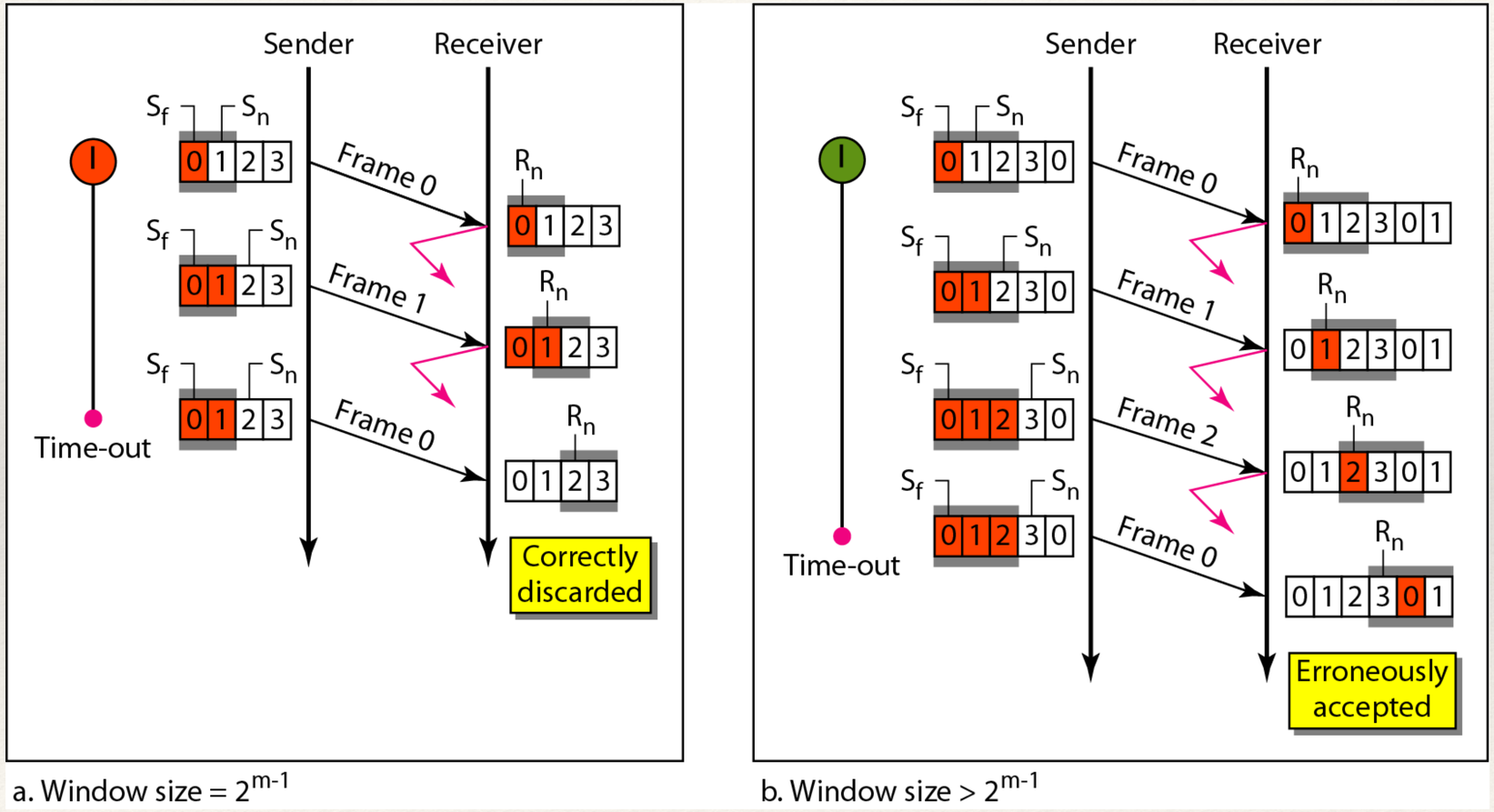


Fig. Receiver Window for Selective Repeat ARQ

Selective Repeat ARQ Window Size



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

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

In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m. (2^{m-1})

Fig. Selective Repeat ARQ Window Size

Selective Repeat ARQ: Design

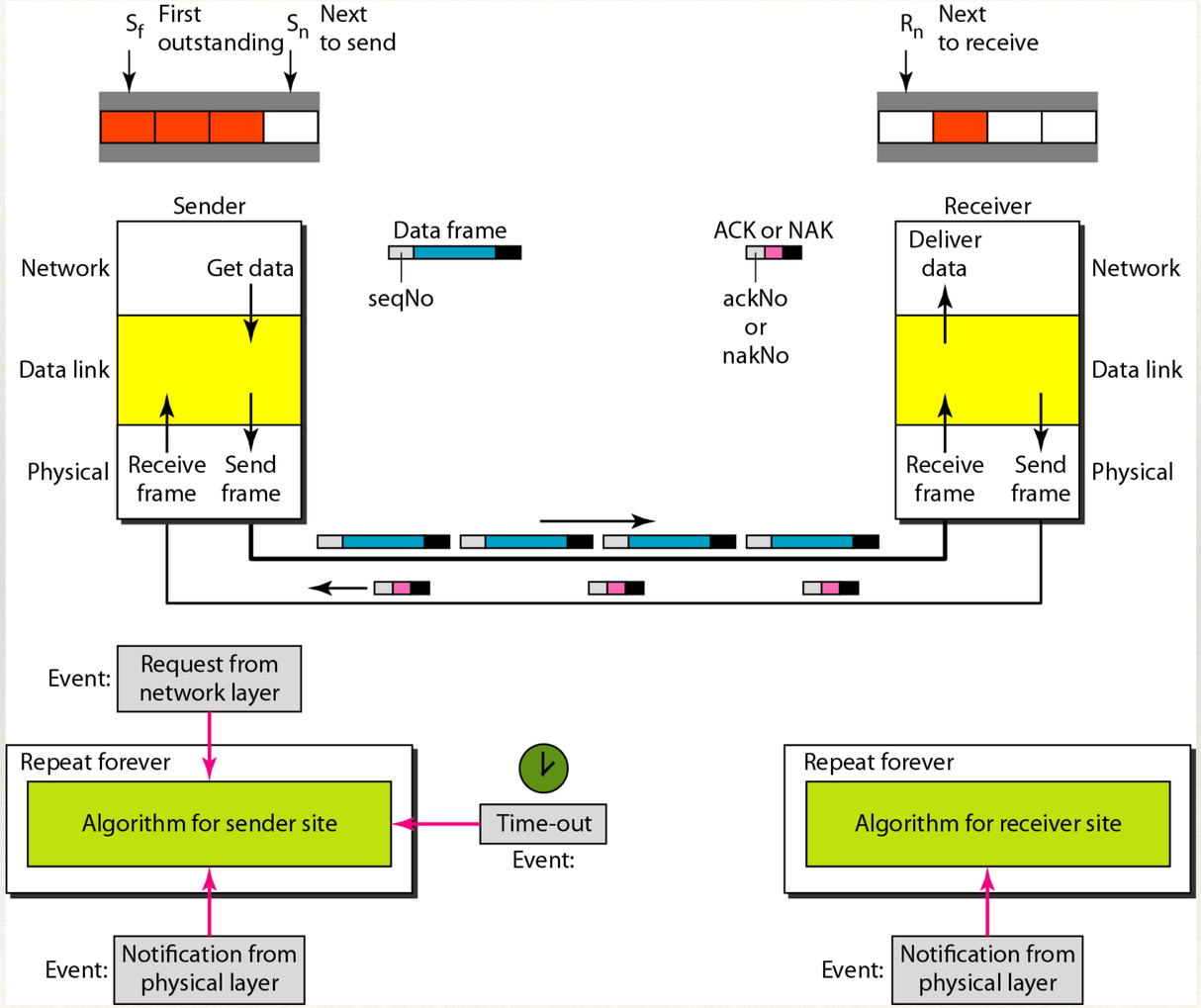


Fig. Selective Repeat ARQ Design

Sender-site Algorithm

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
```

Sender-site Algorithm

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);              //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```

Sender-site Algorithm

```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

Receiver-site Algorithm

```
 1  Rn = 0;
 2  NakSent = false;
 3  AckNeeded = false;
 4  Repeat(for all slots)
 5      Marked(slot) = false;
 6
 7  while (true)                                  //Repeat forever
 8  {
 9    WaitForEvent();
10
11    if(Event(ArrivalNotification))              /Data frame arrives
12    {
13       Receive(Frame);
14       if(corrupted(Frame))&& (NOT NakSent)
15       {
16        SendNAK(Rn);
17        NakSent = true;
18        Sleep();
19       }
20       if(seqNo <> Rn)&& (NOT NakSent)
21       {
22        SendNAK(Rn);
```

Receiver-site Algorithm

```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42        }
43    }
44  }
```

Example-1

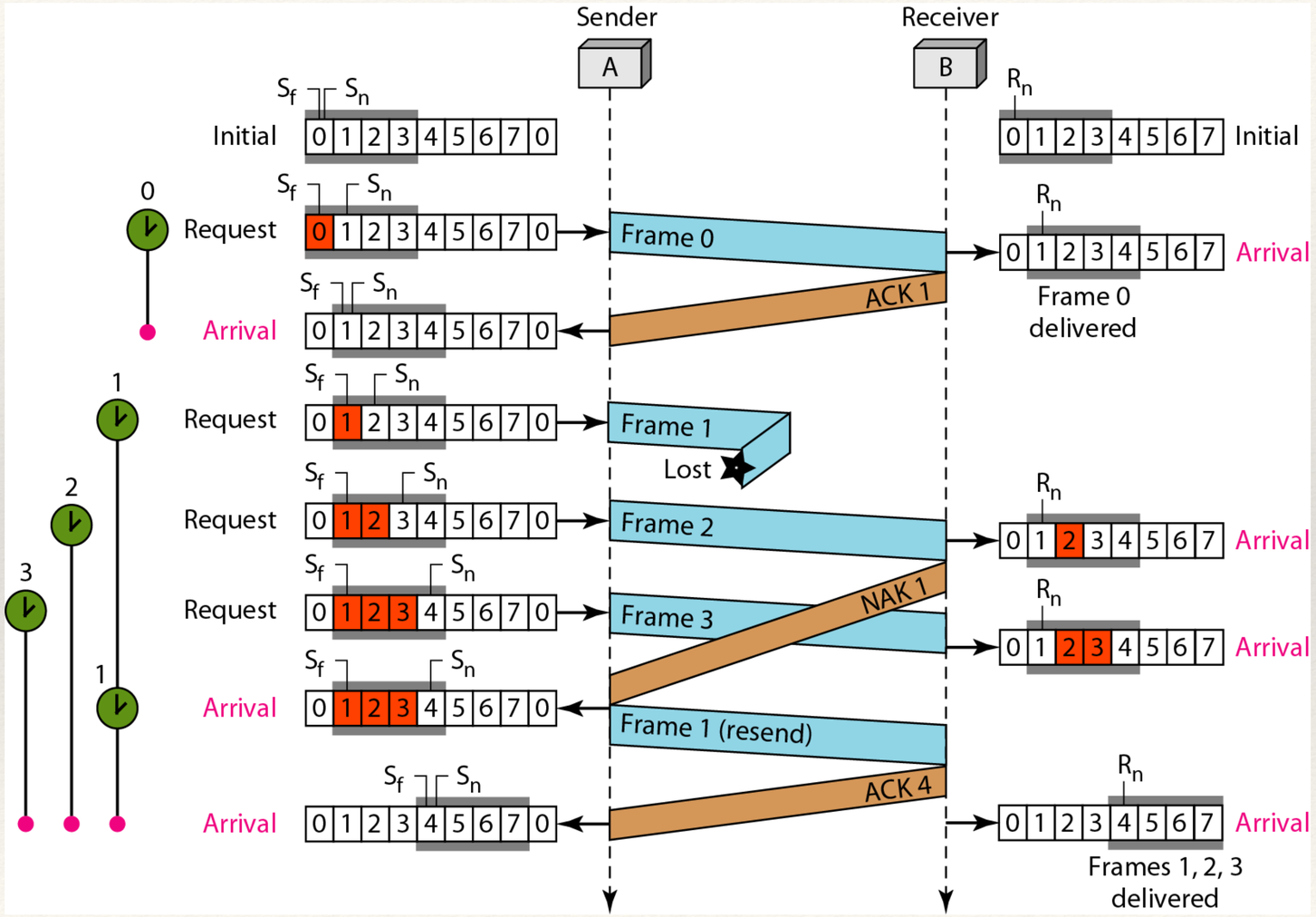
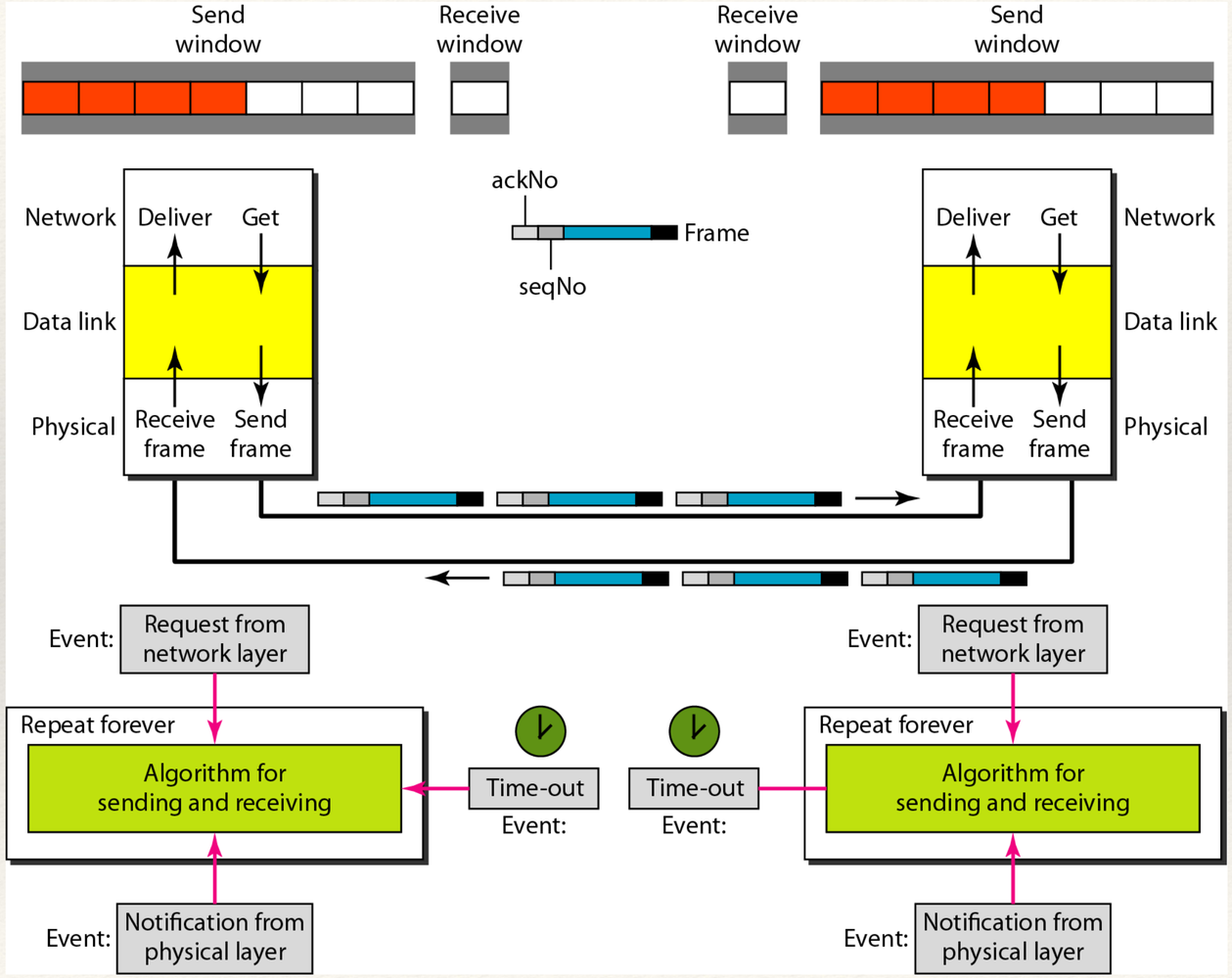


Fig: Selective Repeat ARQ Example Scenario

Piggybacking: Go-back-N ARQ

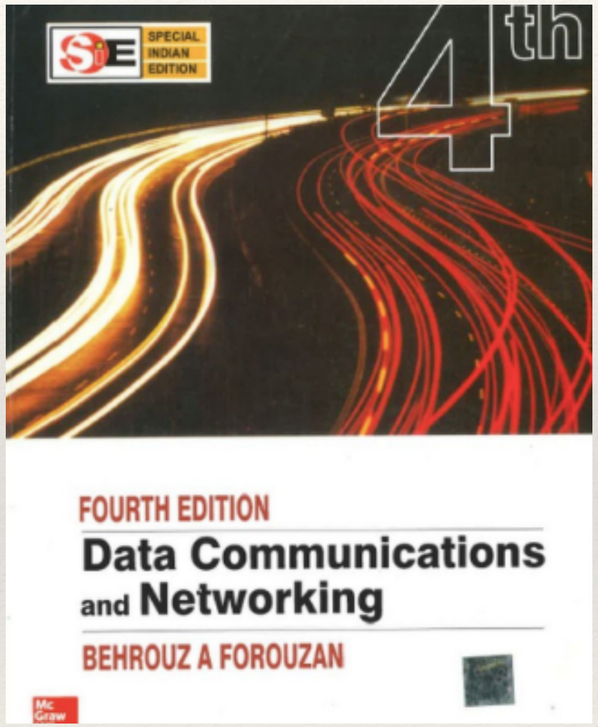
A technique called Piggybacking

- Is used to improve the efficiency of the bidirectional protocols.
- When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B;
- When a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A



Summary

- Types of Error
- Error Detection Techniques
- Framing
- Flow Control Protocols



Note: Refer Chapter 10 and 11 (upto 11.5)