

# Data Link Layer

## Module 2

Error Detection and Correction

## Computer Networks

# Introduction

- Data transfer with accuracy.
- Data can be corrupted during transmission.
- Some applications require that errors be detected and corrected.



**SOMAIYA**  
VIDYAVIHAR UNIVERSITY



# *Important Topics*

- Types of Errors
- Redundancy
- Détection Versus Correction
- Forward Error Correction Versus Retransmission
- Coding



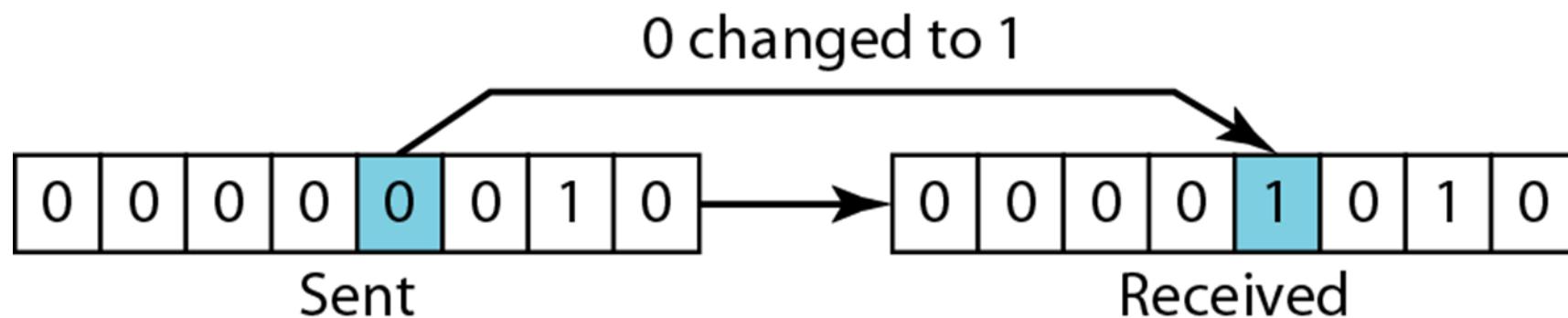
**SOMAIYA**  
VIDYAVIHAR UNIVERSITY



# Types of Errors

- Interference can change the shape of the signal
- In Single Bit error, a 0 is changed to a 1 or a 1 to a 0.
- The term ***single-bit error*** means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.

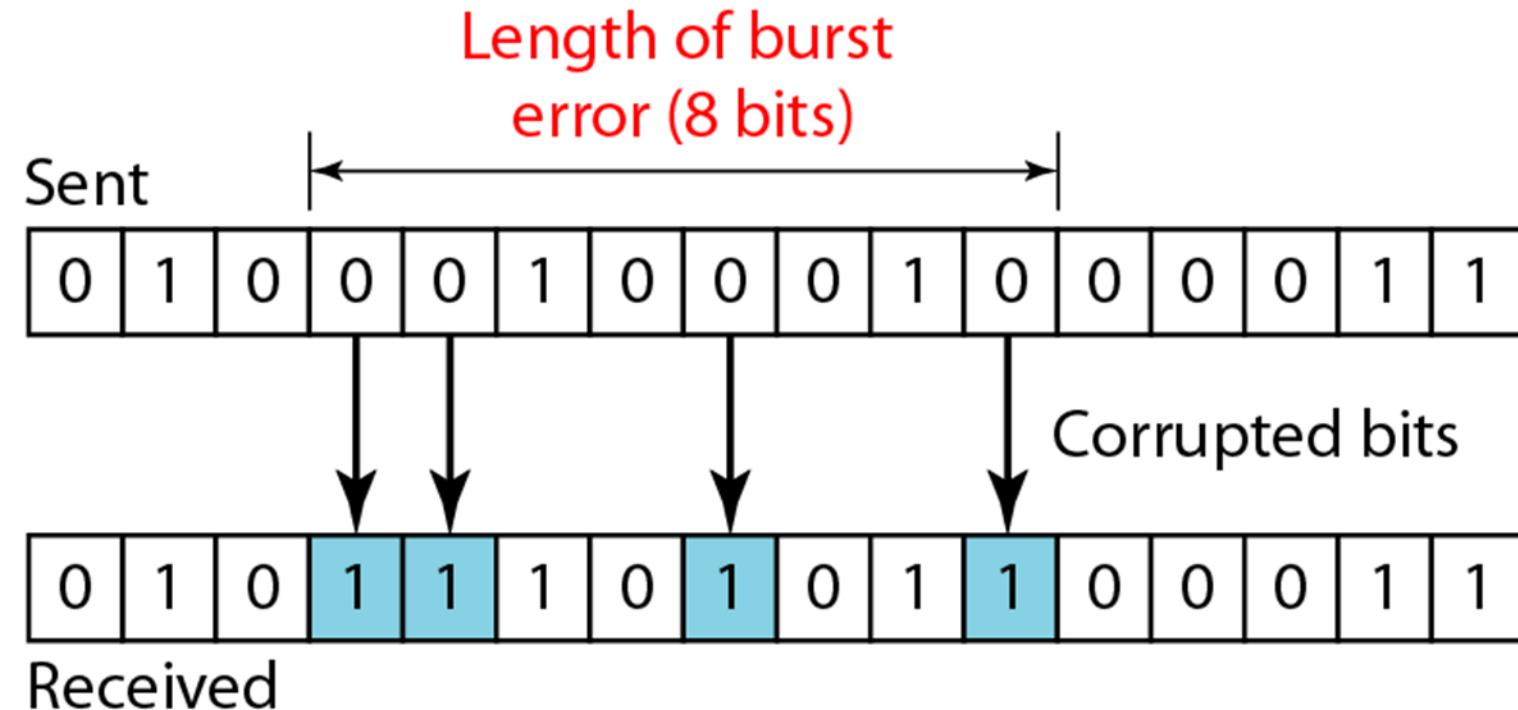
## *Single-Bit Error*



# Types of Errors

- The term ***burst error*** means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.
- ***Burst error*** does not necessarily mean that the errors occur in consecutive bits.
- The length of the burst is measured from the first corrupted bit to the last corrupted bit.

## Burst Error of length 8



## Redundancy

- Central concept in error detection and correction is Redundancy.
- To detect or correct errors, we need to send extra (redundant) bits with data.
- Redundant bits are added by the sender and removed by the receiver.

# Detection Versus Correction

- Correction of errors is more difficult than the detection.
- In ***error detection***, we are looking only to see if any error has occurred. The answer is a simple yes or no.
- In ***error correction***, we need to know the exact number of bits that are corrupted and their location in the message.

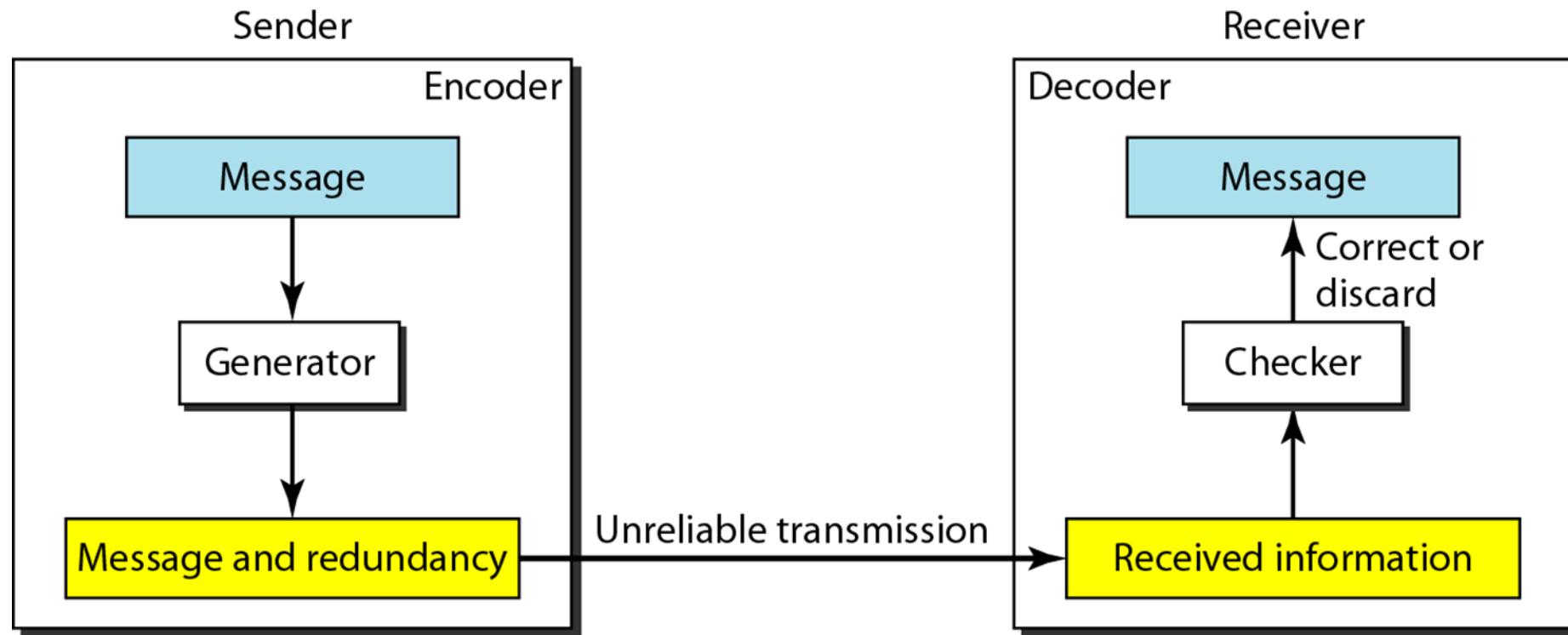
# Forward Error Correction Versus Retransmission

- Two main methods of error correction:
  - Forward Error Correction
  - Retransmission
- ***Forward error correction*** is the process in which the receiver tries to guess the message by using redundant bits.
- ***Retransmission*** is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message.

# Coding

- Redundancy is achieved through various coding schemes
- The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits.
- The receiver checks the relationships between the two sets of bits to detect or correct the errors.
- Coding schemes can be divided into two broad categories:  
***Block coding*** and convolution coding

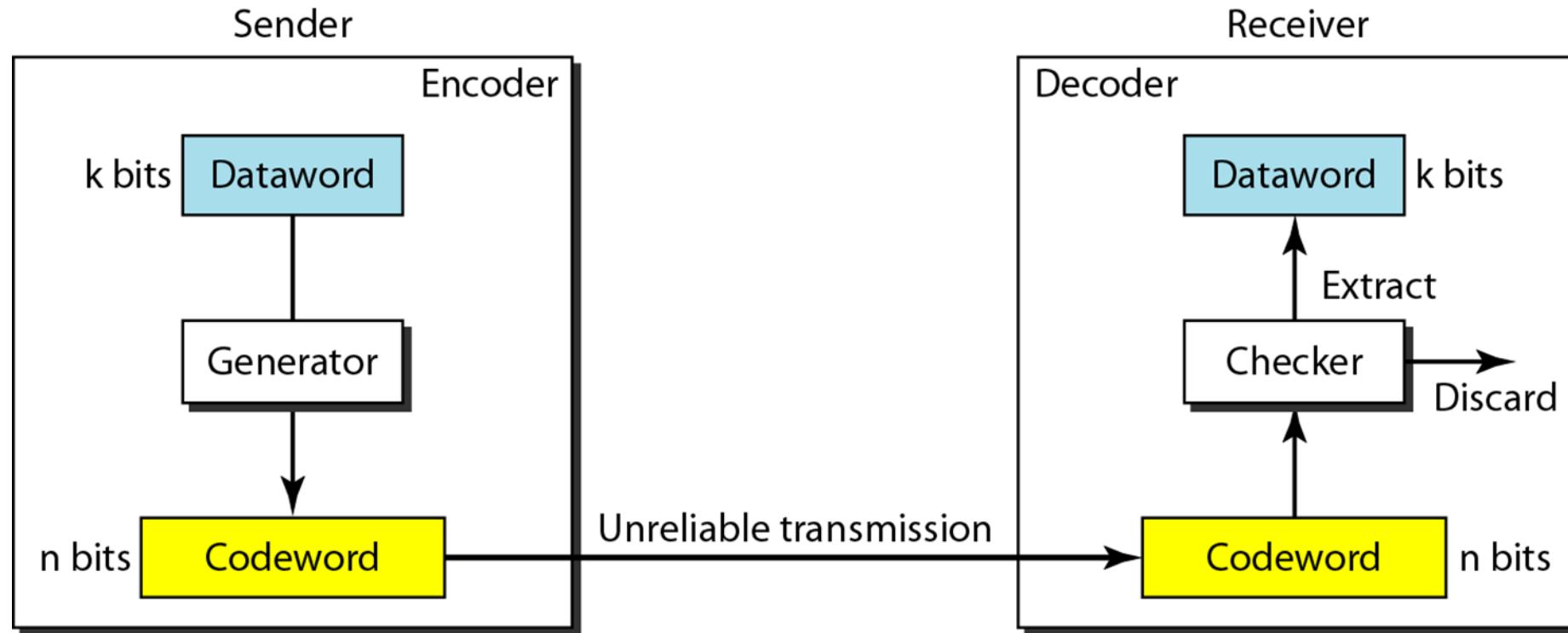
# Coding



*Fig: The structure of 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**.



## Example 1

- Let us assume that  $k = 2$  and  $n = 3$ . Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.
- Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:
  1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.

## Example 1(Cont.)

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.*

## Example 1(Cont.)

<i>Datawords</i>	<i>Codewords</i>
00	000
01	011
10	101
11	110

**Table 10.1** A code for error detection (Example 1)

# Error Correction

- Error correction is much more difficult than error detection
- We need more redundant bits for error correction than for error detection.
- Here, the idea is the same as error detection, but the checker functions are much more complex.

<i>Dataword</i>	<i>Codeword</i>
00	00000
01	01011
10	10101
11	11110

**Table 10.1** A code for error correction

# Hamming Distance

- One of the central concepts in coding for error control is Hamming Distance.
- The Hamming distance between two words(of same size) is the number of differences (XOR operation) between corresponding bits.
- Distance between two words  $x, y$  is represented as  $d(x,y)$

# Hamming Distance

- Let us find the Hamming distance between two pairs of words (number of differences between 2 words)
- 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
$$10101 \oplus 11110 \text{ is } 01011 \text{ (three 1s)}$$

# Minimum Hamming Distance

- Measurement used for designing a code is the minimum Hamming distance.
- The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of codewords.
- $d_{min}$  is used to define the minimum Hamming distance in a coding scheme.

## Example 2

- Find the minimum Hamming distance of the coding scheme in Table.

<i>Datawords</i>	<i>Codewords</i>
00	000
01	011
10	101
11	110

**Solution:**

We 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.

## Example 3

- Find the minimum Hamming distance of the coding scheme in Table.

<i>Dataword</i>	<i>Codeword</i>
00	00000
01	01011
10	10101
11	11110

### Solution:

We first find all the 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.

## 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

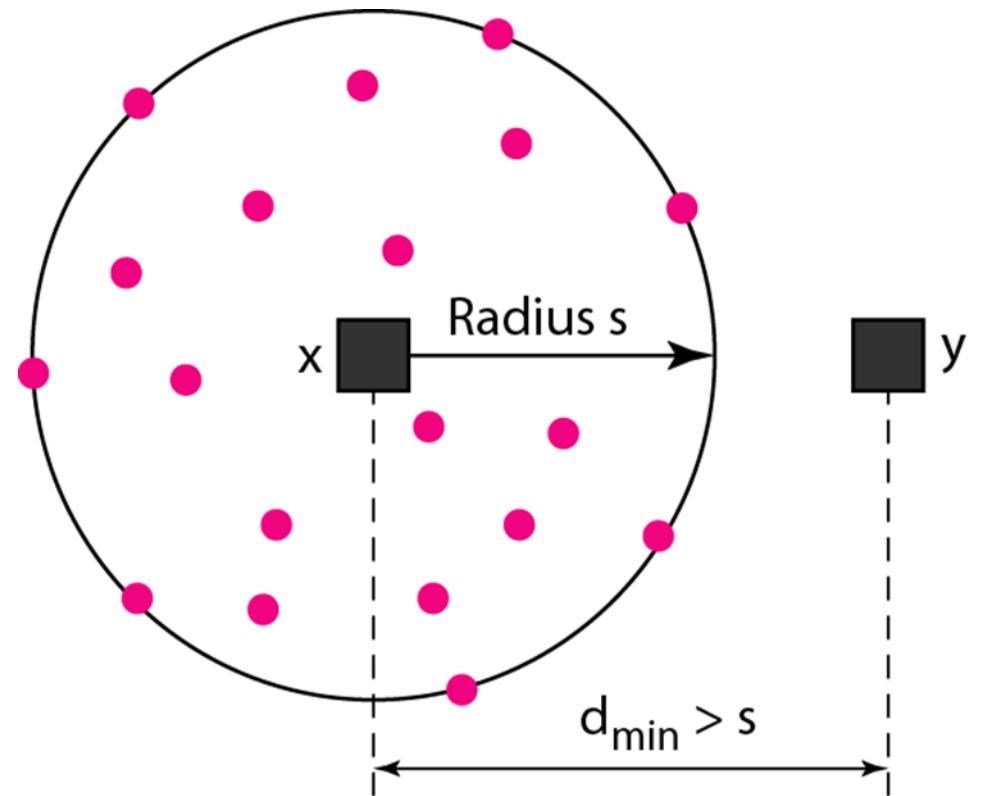
$$d_{min} = s + 1.$$

## Minimum Hamming Distance for Error Correction

To guarantee **correction** of up to  $t$  errors in all cases, the minimum Hamming distance in a block code must be

$$d_{min} = 2t + 1.$$

**Fig: Geometric concept for finding  $d_{\min}$  in error detection**



**Legend**

- Any valid codeword
- Any corrupted codeword with 0 to  $s$  errors

## Example 4

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.

## Minimum Distance for Linear Block Codes:

The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Example: [refer previous examples](#)

# Some Linear Block Codes

1. Simple Parity-Check Code
2. Hamming Codes

## Simple Parity-Check Code:

- a k-bit dataword is changed to an n-bit codeword where  $n = k + 1$ .
- The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even/odd.
- The minimum Hamming distance for this category is  $d_{min} = 2$ , which means that the code is a single-bit error-detecting code; it cannot correct any error.

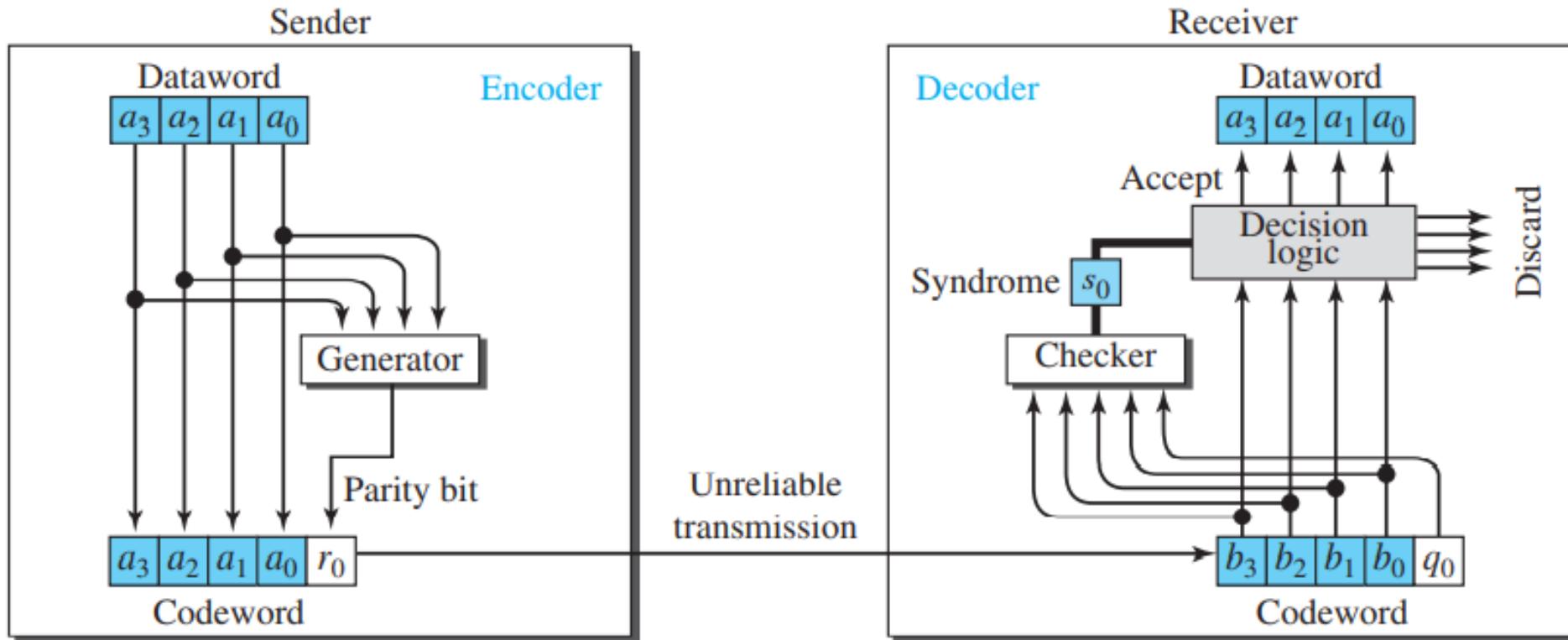
# Example

**Table 10.2** Simple parity-check code  $C(5, 4)$

Dataword	Codeword	Dataword	Codeword
0000	<b>00000</b>	1000	<b>10001</b>
0001	<b>00011</b>	1001	<b>10010</b>
0010	<b>00101</b>	1010	<b>10100</b>
0011	<b>00110</b>	1011	<b>10111</b>
0100	<b>01001</b>	1100	<b>11000</b>
0101	<b>01010</b>	1101	<b>11011</b>
0110	<b>01100</b>	1110	<b>11101</b>
0111	<b>01111</b>	1111	<b>11110</b>

# Parity Check Code

**Figure 10.4** Encoder and decoder for simple parity-check code



## Parity Check Code

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.

## Parity Check Code

4. An error changes  $r_0$  and a second error changes  $a_3$ . The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
  
5. Three bits— $a_3$ ,  $a_2$ , and  $a_1$ —are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created.

**Note:** A parity-check code can detect an odd number of errors.

# Hamming Code

- Designed with  $d_{min} = 3$  i.e. detect up to 2 errors and correct up to 1-bit error.
- For example, if  $m = 3$ , then  $n=7$  and  $k=4$ . This is a Hamming code  $C(7, 4)$  with  $d_{min} = 3$ .

# Hamming Code C(7,4)

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
Parity bit coverage	p1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
p2		x	x		x	x			x	x			x	x			x	x		
p4				x	x	x	x				x	x	x	x						x
p8								x	x	x	x	x	x	x	x					
p16																x	x	x	x	x

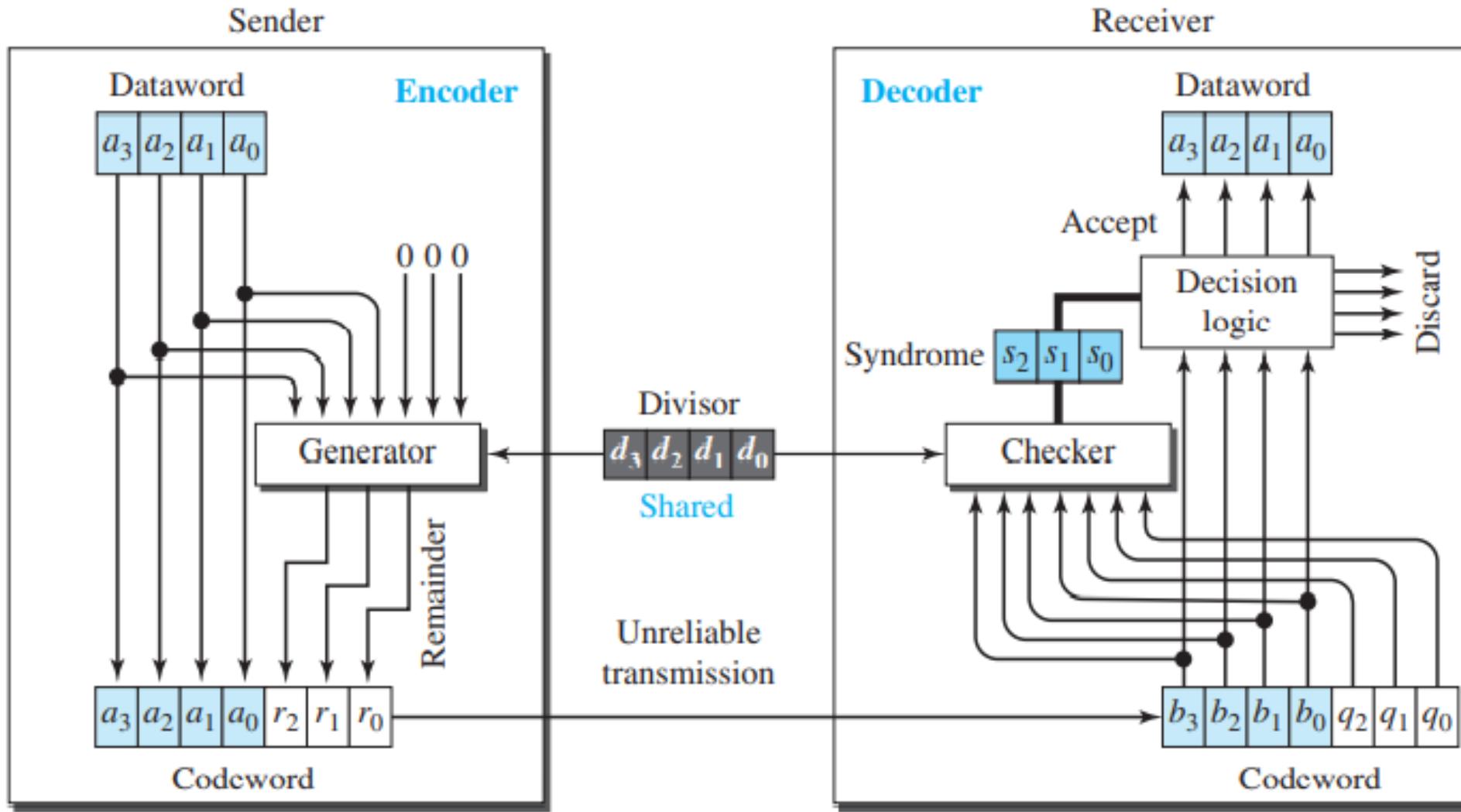
# Cyclic Codes

- Cyclic codes are special linear block codes.
- If a codeword is cyclically shifted (rotated), the result is another codeword.
- **Cyclic Redundancy Check:**
  - A subset of cyclic codes, used for error detection.
  - Used in networks such as LANs and WANs.

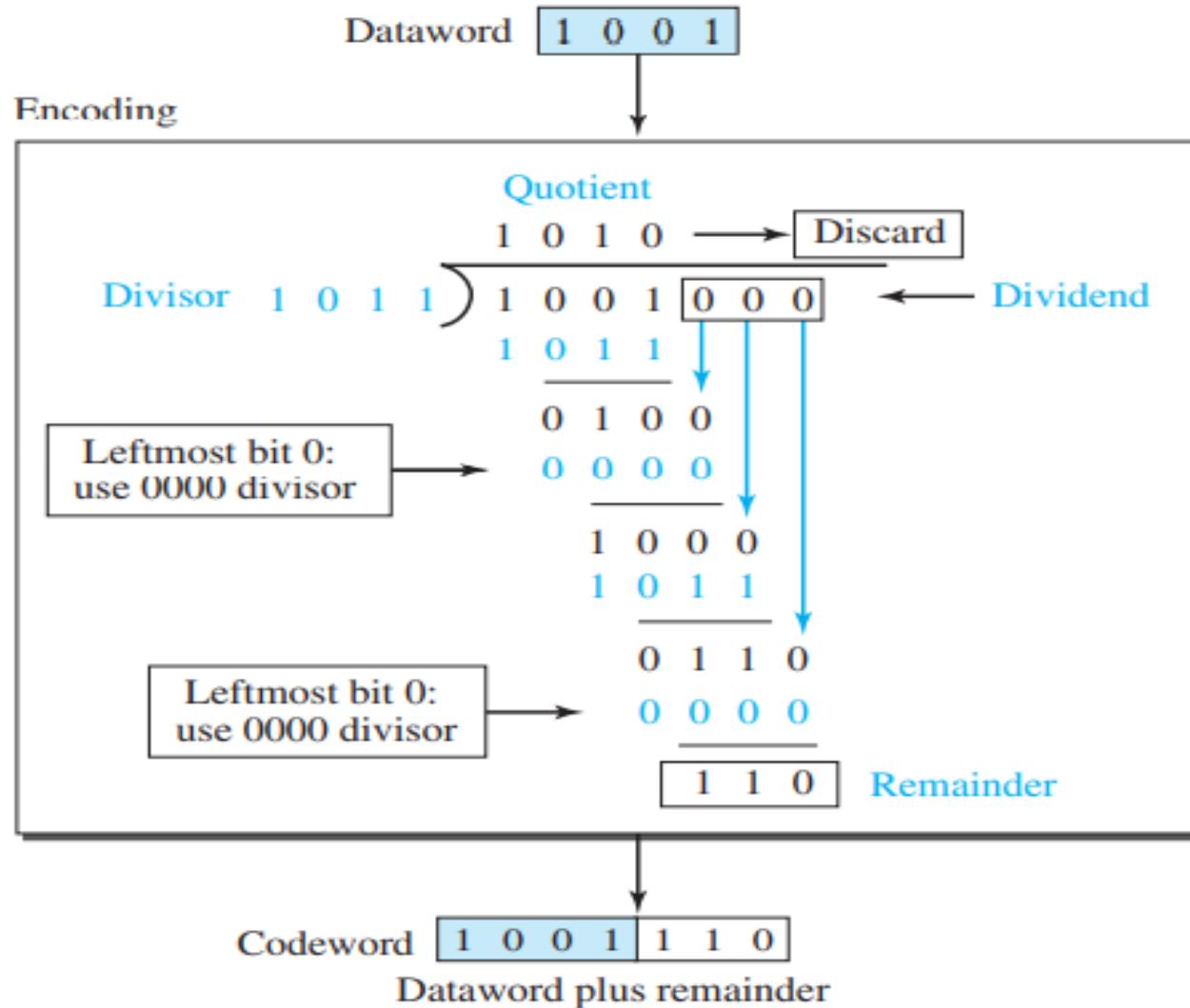
# CRC Code with C(7,4)

<i>Dataword</i>	<i>Codeword</i>	<i>Dataword</i>	<i>Codeword</i>
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

# CRC Code encoder and decoder



# Division in CRC encoder



Note:

Multiply: AND  
Subtract: XOR

# CRC Division using polynomial

Dataword  $x^3 + 1$



Divisor

$$\begin{array}{r}
 x^3 + x \\
 \hline
 x^3 + x + 1 ) \overline{x^6 + \quad x^3} \\
 \quad \quad \quad x^6 + x^4 + x^3 \\
 \hline
 \quad \quad \quad x^4 \\
 \quad \quad \quad x^4 + x^2 + x \\
 \hline
 \quad \quad \quad x^2 + x
 \end{array}$$

**Dividend:**  
augmented dataword

Remainder

Codeword  $x^6 + x^3 \quad | \quad x^2 + x$

Dataword      Remainder

The divisor in a cyclic code is normally called the generator polynomial or simply the generator

# Standard polynomials

Name	Polynomial	Used in
CRC-8	$x^8 + x^2 + x + 1$ <b>100000111</b>	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ <b>11000110101</b>	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ <b>10001000000100001</b>	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ <b>100000100110000010001110110110110111</b>	LANs

The divisor in a cyclic code is normally called the generator polynomial or simply the generator

# Checksum

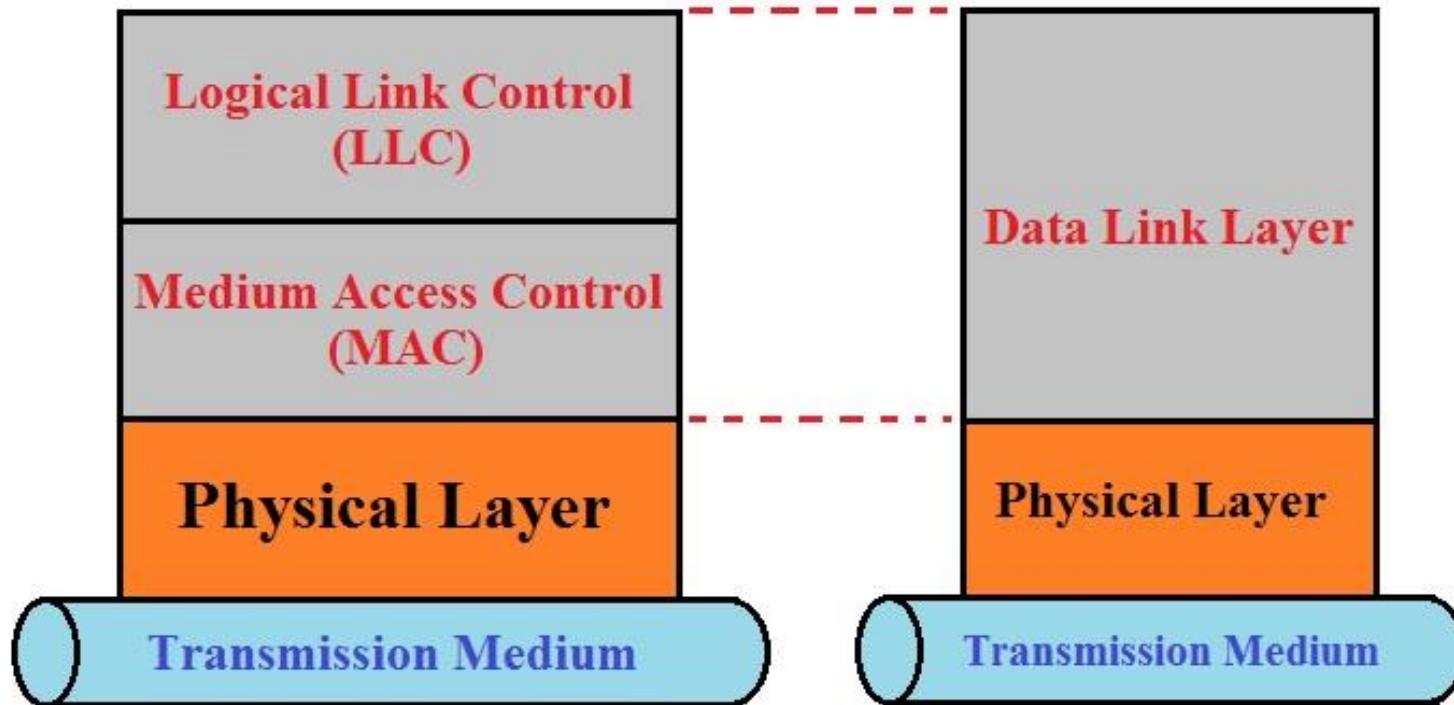
## Internet Checksum:

- Traditionally, the Internet has used a 16-bit checksum
- The sender and the receiver follow the steps depicted in Table

<i>Sender</i>	<i>Receiver</i>
<ol style="list-style-type: none"> <li>1. The message is divided into 16-bit words.</li> <li>2. The value of the checksum word is initially set to zero.</li> <li>3. All words including the checksum are added using one's complement addition.</li> <li>4. The sum is complemented and becomes the checksum.</li> <li>5. The checksum is sent with the data.</li> </ol>	<ol style="list-style-type: none"> <li>1. The message and the checksum are received.</li> <li>2. The message is divided into 16-bit words.</li> <li>3. All words are added using one's complement addition.</li> <li>4. The sum is complemented and becomes the new checksum.</li> <li>5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.</li> </ol>

# Data Link Layer

Data Link Layer is Divided into two sub layers:



# Data Link Layer

## Logical Link Control (LLC Sublayer):

- Controls frame synchronization, flow control and error Checking.

## Media Access Control (MAC Sublayer):

- Responsible for moving data packets from one interface to another across the shared channel.

# Framing

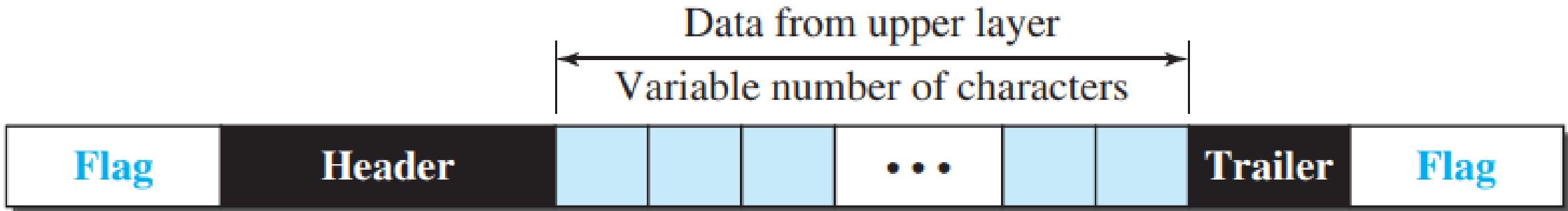
The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.

Example: Postal system: letter and envelope(delimiter)

- **Fixed Size Framing:** Size itself act as a delimiter
- **Variable Size Framing:** need a way to define end of one frame and beginning of next.
  - Character (Byte) Oriented Framing
  - Bit Oriented Framing

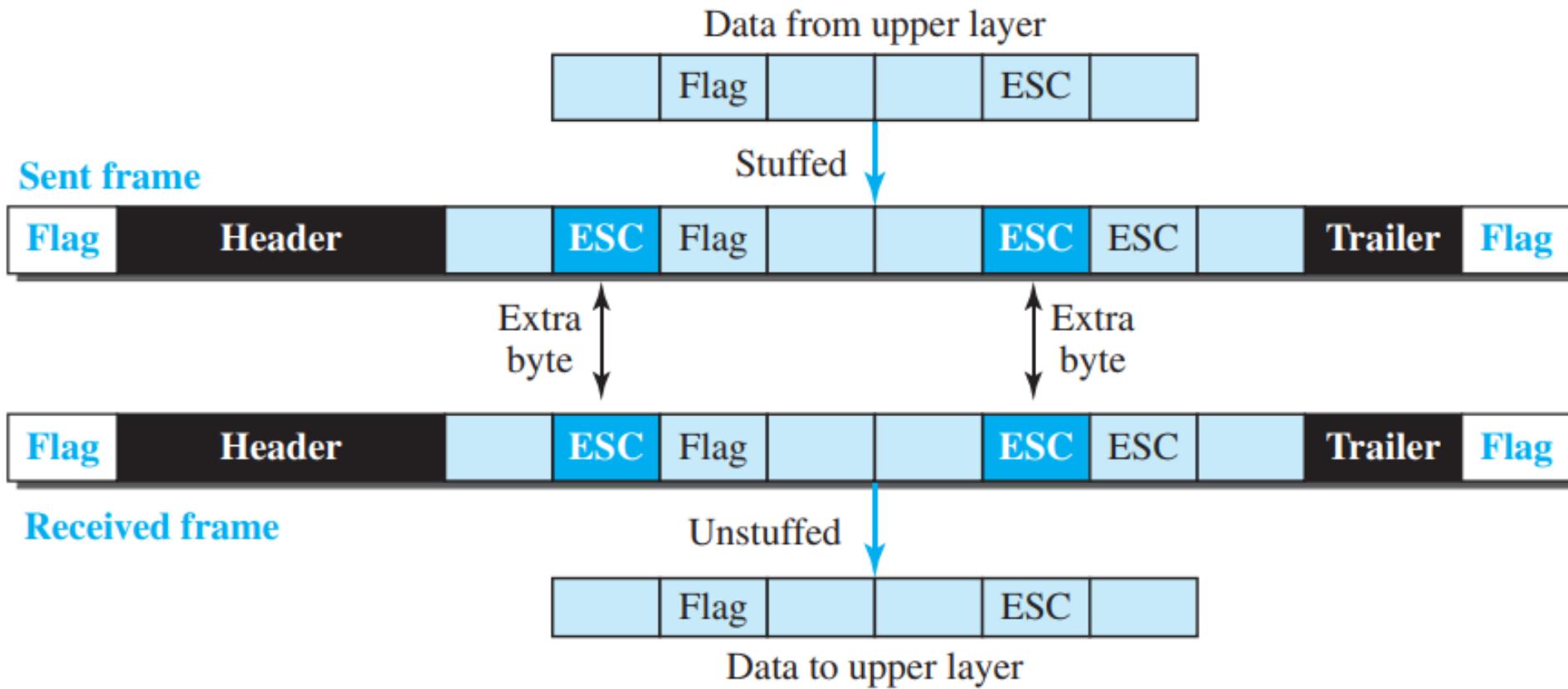
# Character Oriented Framing

- In Character (Byte) oriented Framing, data to be carried are 8-bit characters.
- Header and Trailer are also multiples of 8-bit.
- To separate one from the next, flag is added at the beginning and end of the frame.



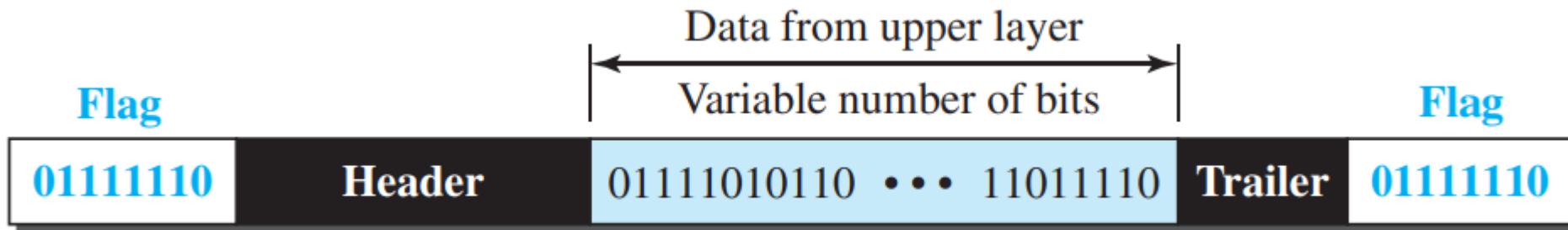
# Character Oriented Framing

**Byte Stuffing and Unstuffing:** Byte stuffing is the process of adding one extra byte whenever there is a flag or escape character in the text.



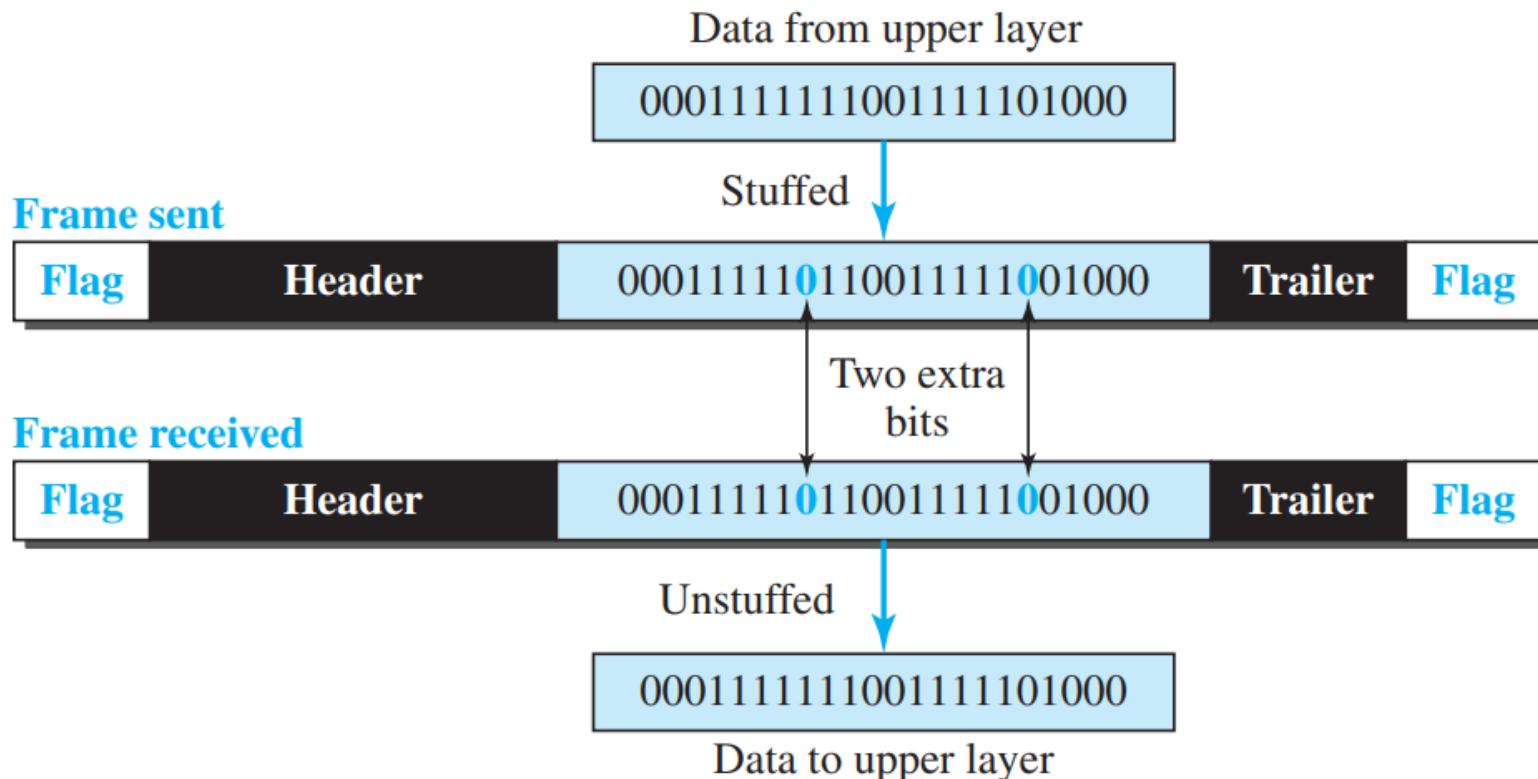
# Bit-Oriented Framing

- In Bit-oriented Framing, a special 8-bit pattern flag, **01111110**, is used as the delimiter to define the beginning and the end of the frame.



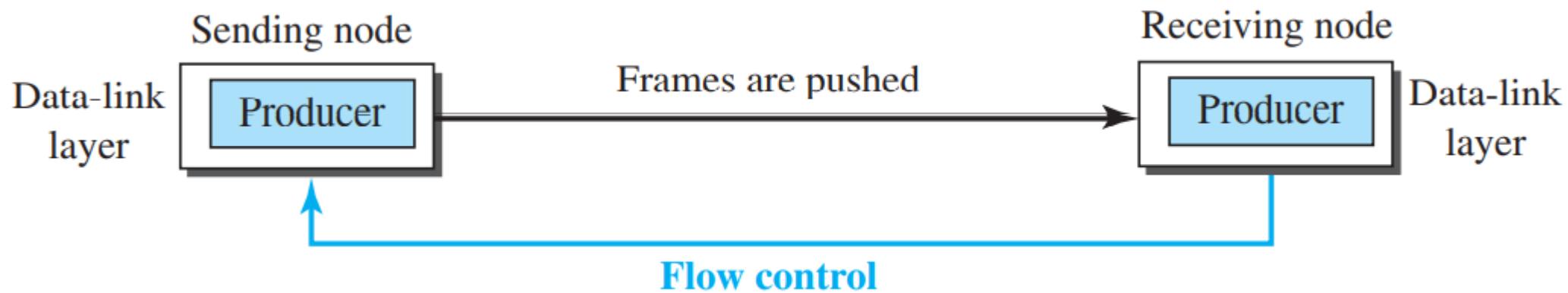
# Bit Stuffing and unstuffing

- In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added.
- This extra stuffed bit is eventually removed from the data by the receiver.



# Flow Control and Error Control

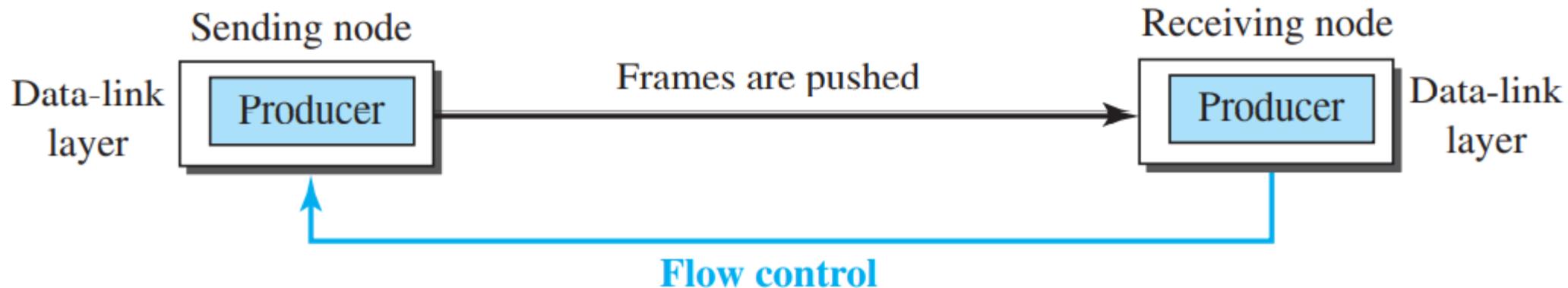
One of the responsibilities of the data link control sublayer is flow and error control at the data-link layer. Collectively, these functions are known as **data link control**. waiting for acknowledgment.



# Flow Control and Error Control

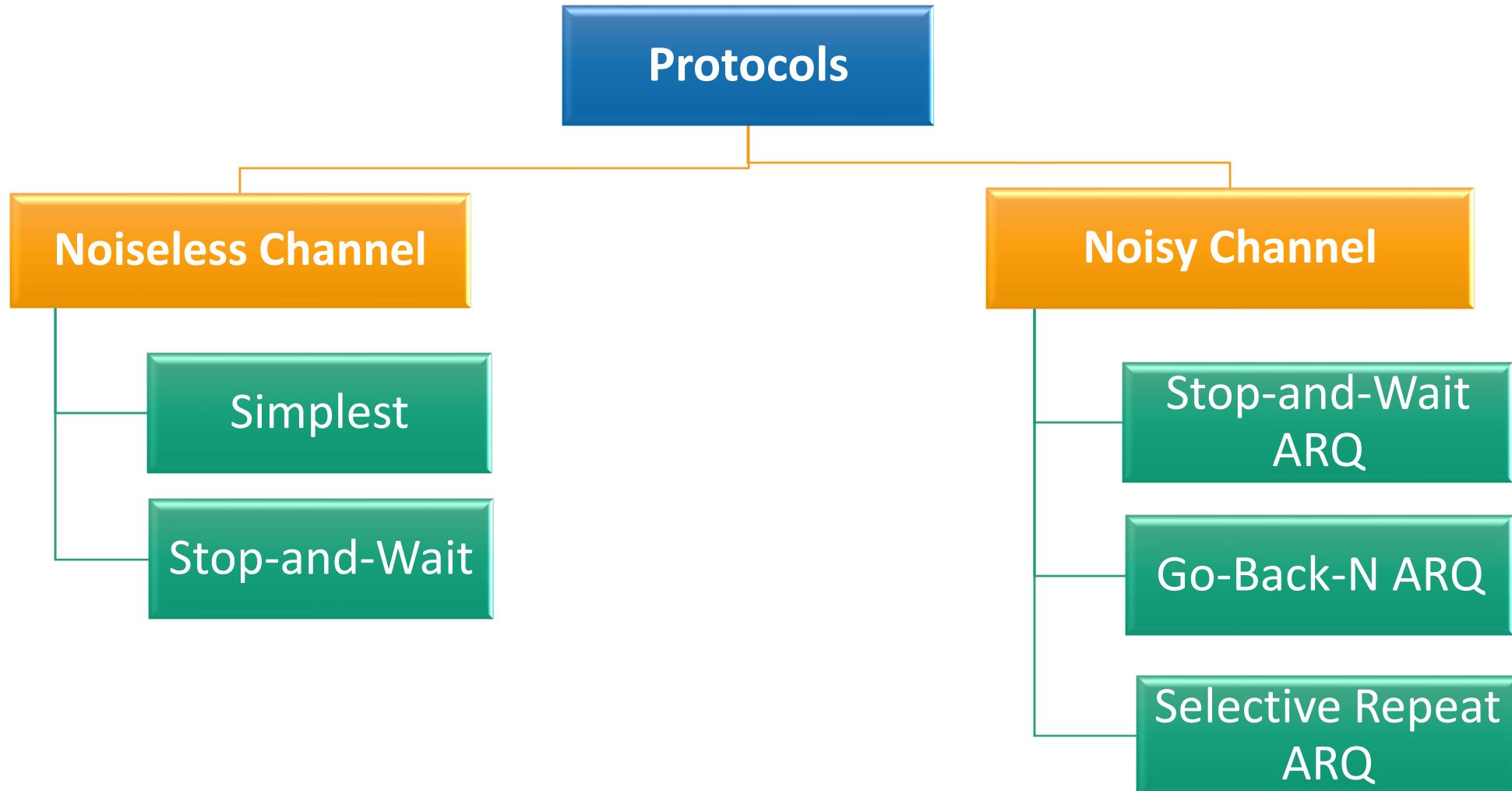
**Flow Control at Data Link Layer:** Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before.

**Error control** in the data link layer is based on automatic repeat request(ARQ), which is the retransmission of data.



## Flow Control and Error 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.
- Flow of data must not allow to overwhelm the receiver.
- Receiving device have limited speed at which it can process incoming data and limited amount of memory to store incoming data.
- Error control is both error detection and correction.
- Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.



## Noiseless Channel

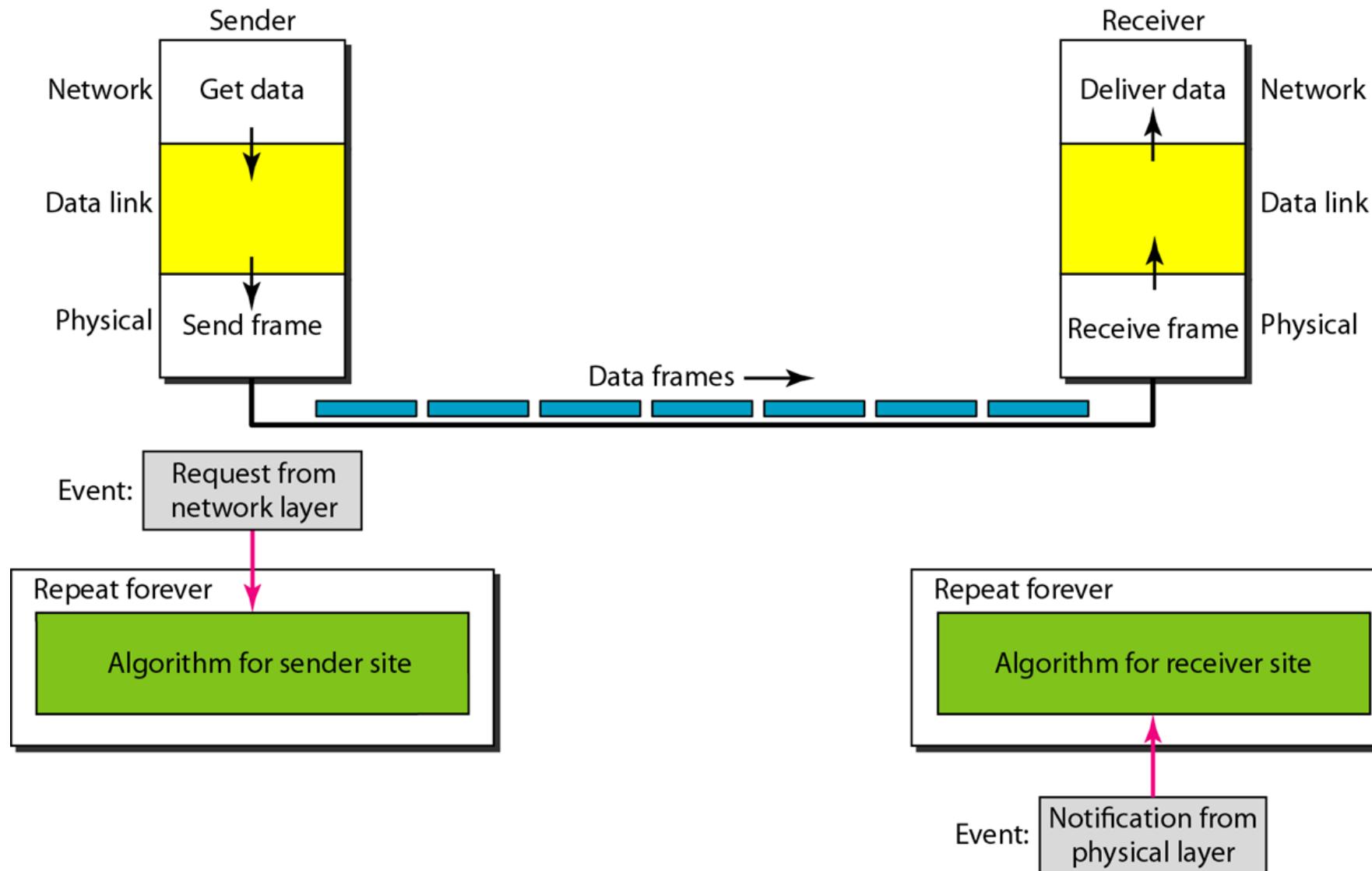
Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.

- Simplest Protocol
- Stop-and-Wait Protocol

# Simplest Protocol

- Cannot be used in real life
- No flow and error control
- Receiver can handle any frame with negligible processing time.
- Receiver can never be overwhelmed with incoming frames.

# Simplest Protocol



## Sender-site algorithm for the simplest protocol

```
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 }
```

## Receiver-site algorithm for the simplest protocol

```
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 }
```

## Simplest Protocol (*Example*)

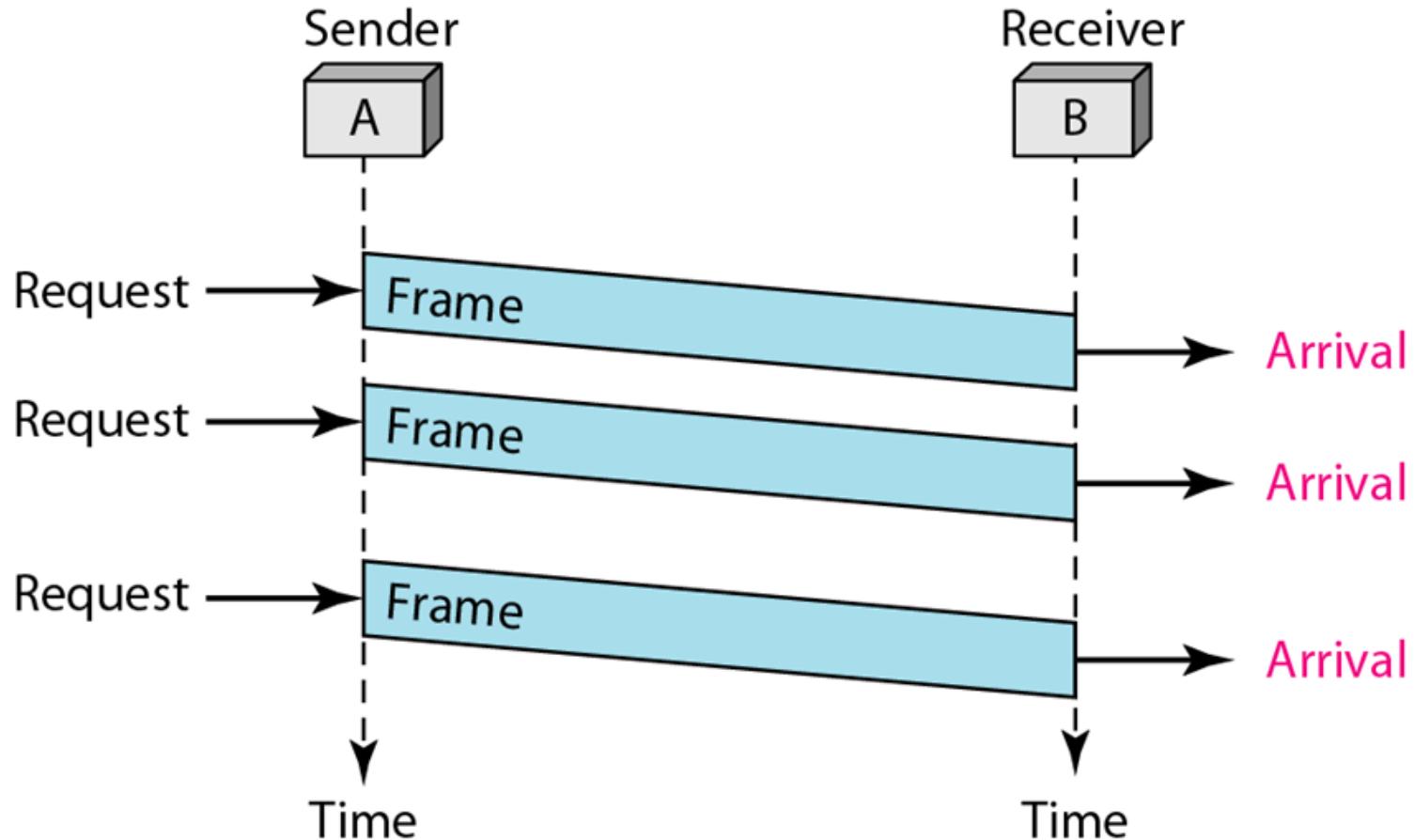
***Figure shows an example of communication using this protocol.***

- *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.*

# Simplest Protocol (*Example*)

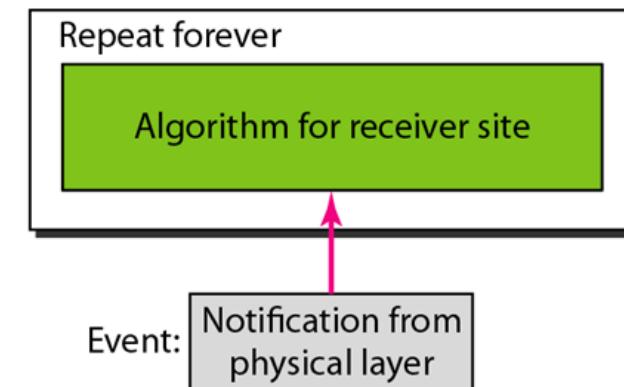
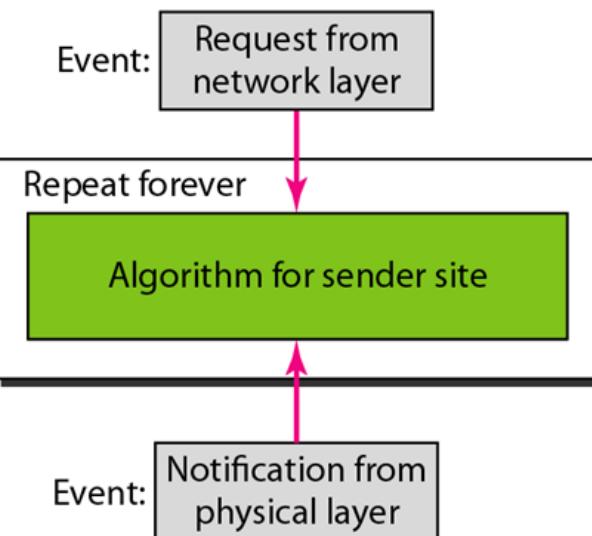
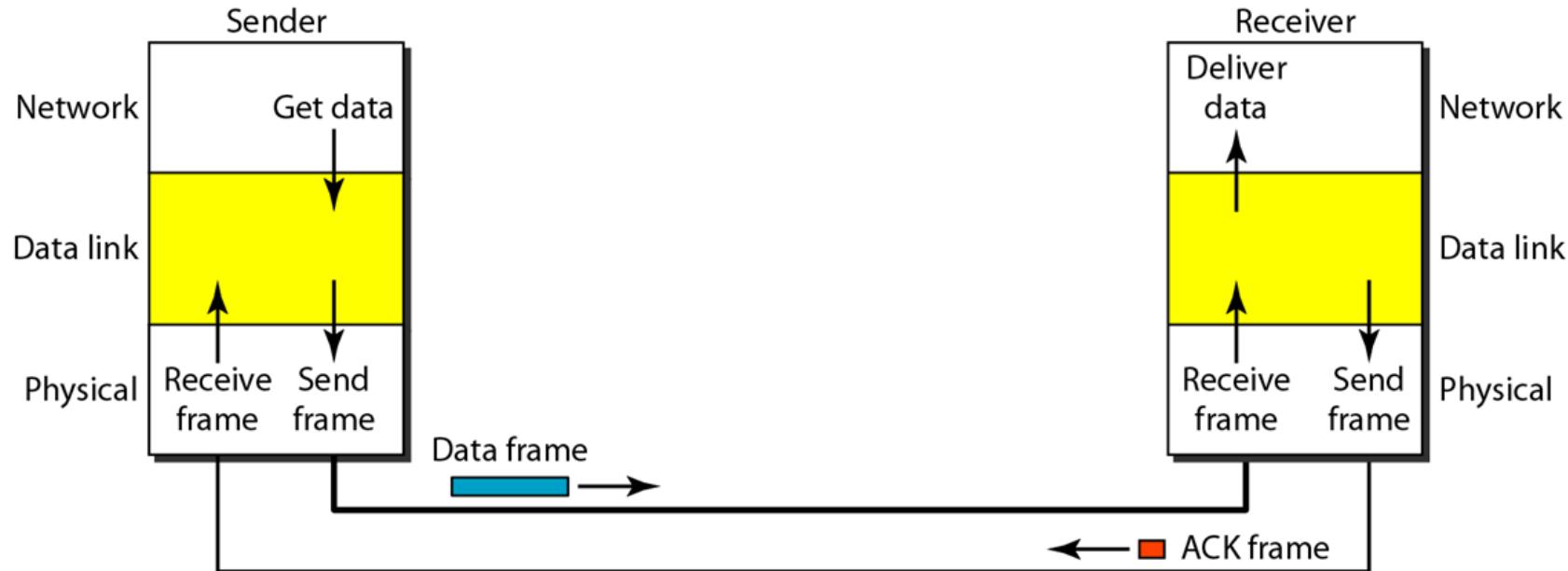
**Figure:** *Flow Diagram*



## Stop and Wait Protocol

- If frames arrive faster than they can be processed, frames must be stored until their use.
- To prevent receiver from becoming overwhelmed, receiver needs to tell sender to slow down (ACK).
- Sender sends one frame, stops until it receives confirmation from receiver and then send next frame.

# Stop and Wait Protocol



# Stop and Wait Protocol

```
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 }
```

# Stop and Wait Protocol

```
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 }
```

## Stop and Wait Protocol (*Example*)

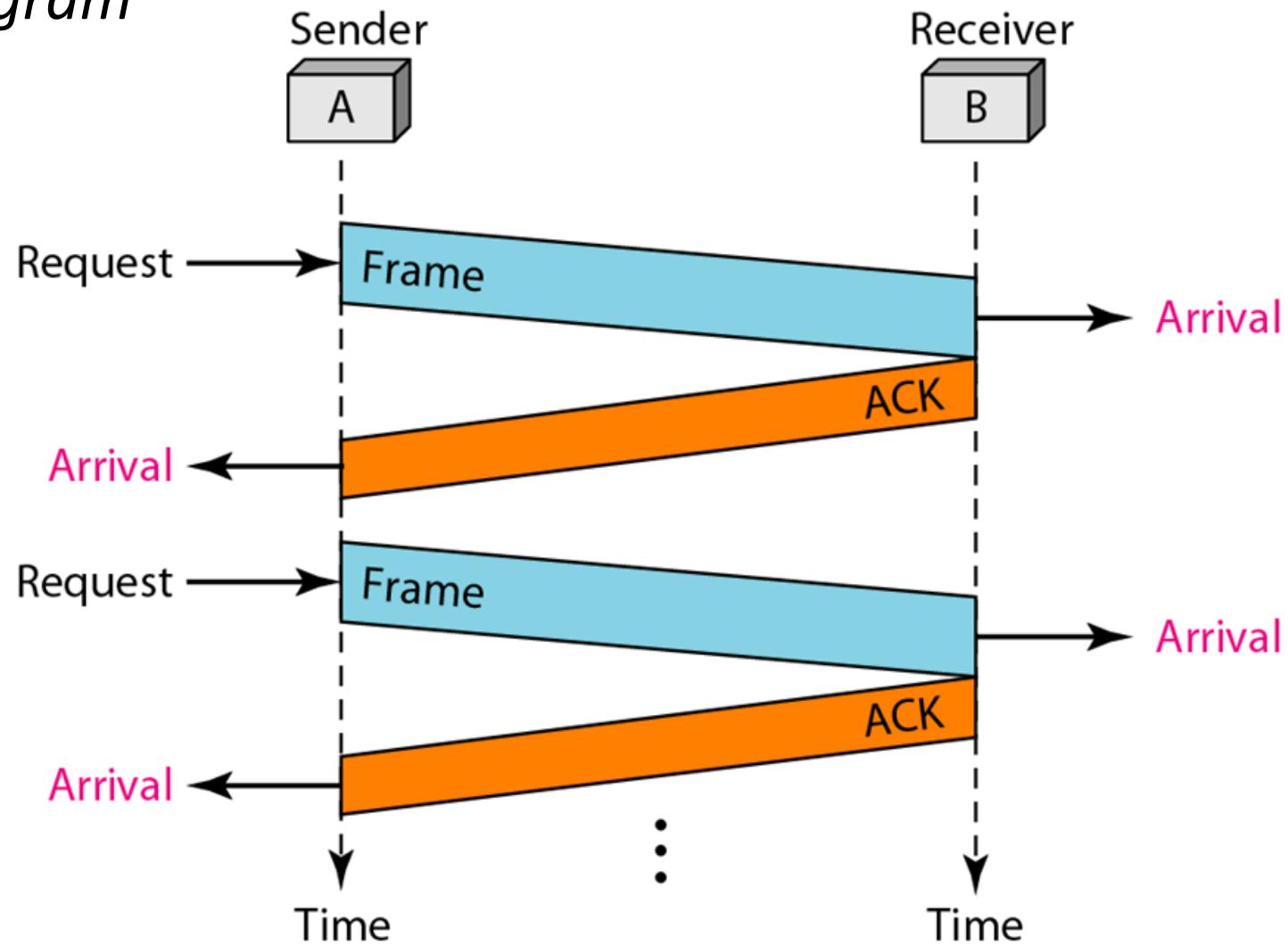
*Figure shows an example of communication using this protocol. It is still very simple.*

- *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.*

# Stop and Wait Protocol (*Example*)

Figure: Flow Diagram



## Noisy Channel

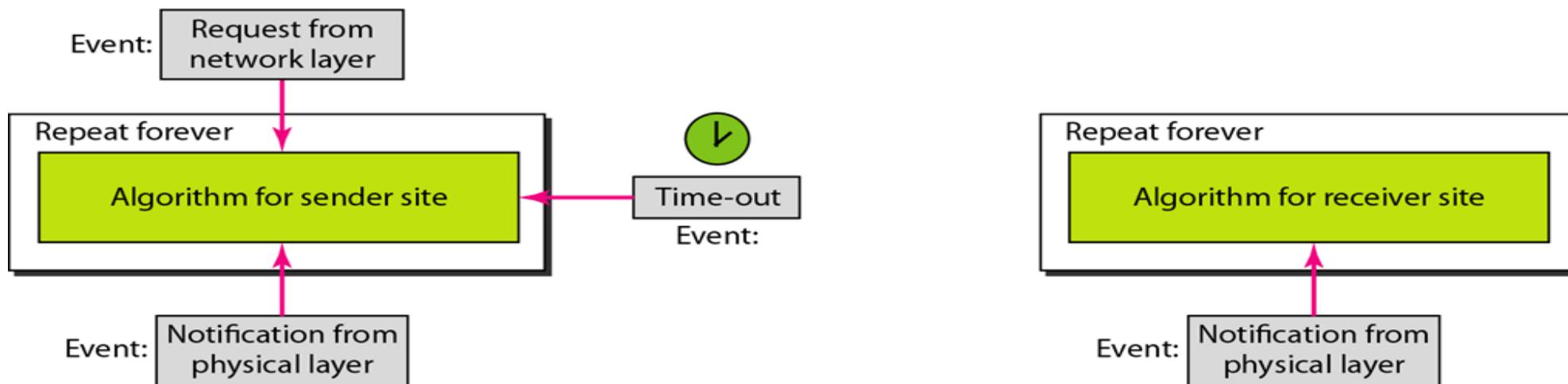
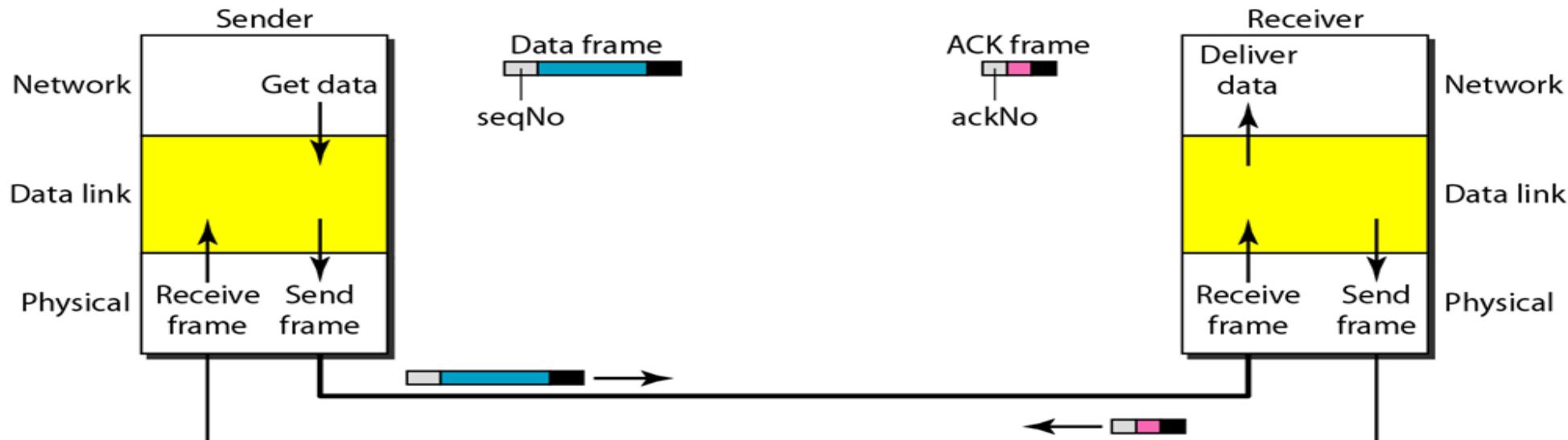
Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.

- Stop-and-Wait Automatic Repeat Request
- Go-Back-N Automatic Repeat Request
- Selective Repeat Automatic Repeat Request

# 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



## Algorithm: Sender site Algorithm for Stop And Wait ARQ Protocol

```
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
```

(Continues...)

## Algorithm: Sender site Algorithm for Stop And Wait ARQ Protocol

```
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 }
```

## Algorithm: Receiver site Algorithm for Stop And Wait ARQ Protocol

```

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 }
```

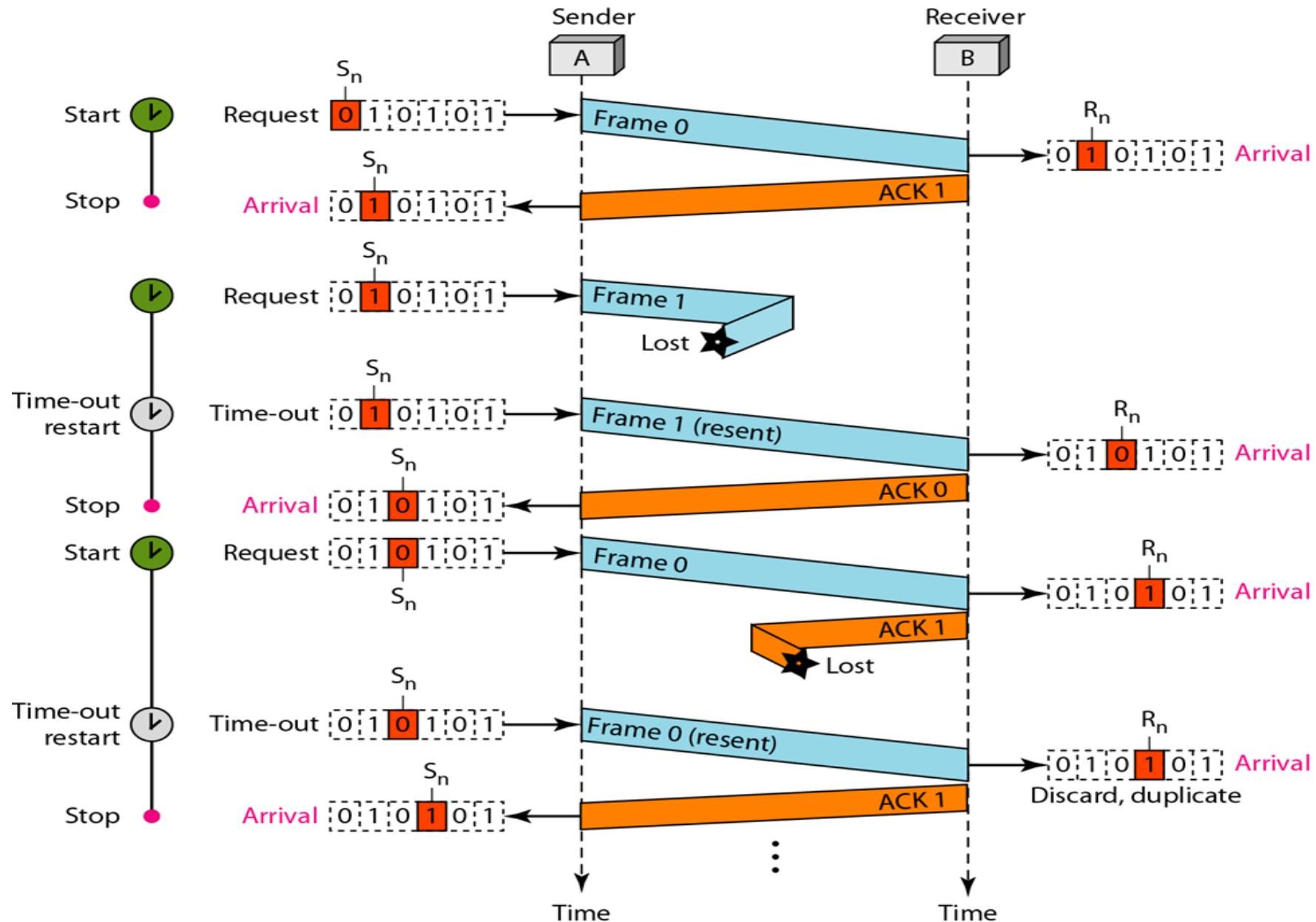
## Stop And Wait ARQ (Example)

**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.*

# Stop And Wait ARQ (Example)



# Stop And Wait ARQ

## **Efficiency:**

- *Very inefficient if channel is **thick** (large bandwidth) and **long** (round trip delay is long).*
- *The product of these two is called the **bandwidth-delay product** (volume of the pipe in bits).*
- *The **bandwidth-delay product** is a measure of the number of bits we can send out of our system while waiting for ACK from the receiver.*

# Stop And Wait ARQ

## **Example 1:**

*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:*

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

## Stop And Wait ARQ

*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.*

# Stop And Wait ARQ

## **Pipelining:**

- *No pipelining in Stop-And-Wait ARQ.*
- *We need to wait for a frame to reach the destination and be acknowledged before next frame can be sent.*
- *Pipelining is applicable to next two protocols, wherein several frames can be sent before receiving acknowledgement for previous frames.*
- *Pipelining improves efficiency.*

# Go-Back-N ARQ

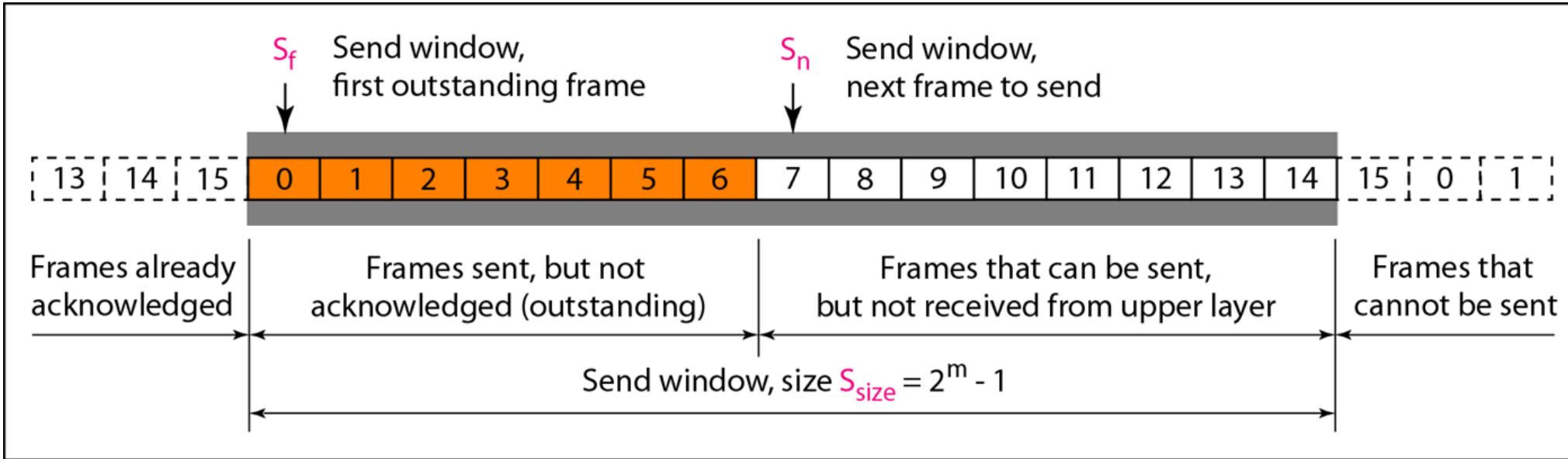
- To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment.
- This protocol can send several frames before receiving acknowledgments it keep a copy of these frames until the acknowledgments arrive.
- In the Go-Back-N Protocol, the **sequence numbers** are modulo  $2m$  i.e. the sequence numbers range from **0 to  $2^m - 1$** , where  $m$  is the size of the sequence number field in bits.

# Go-Back-N ARQ

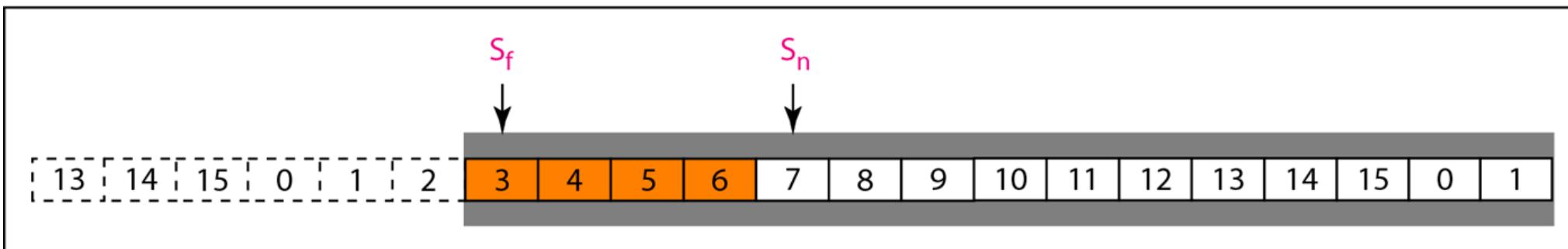
## Sliding Window:

- Sliding window is an abstract concept that defines the range of sequence numbers.
- The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.
- The window at any time divides the possible sequence numbers into four regions.

## Send Window for Go-Back-N ARQ:



a. Send window before sliding



b. Send window after sliding

# Go-Back-N ARQ

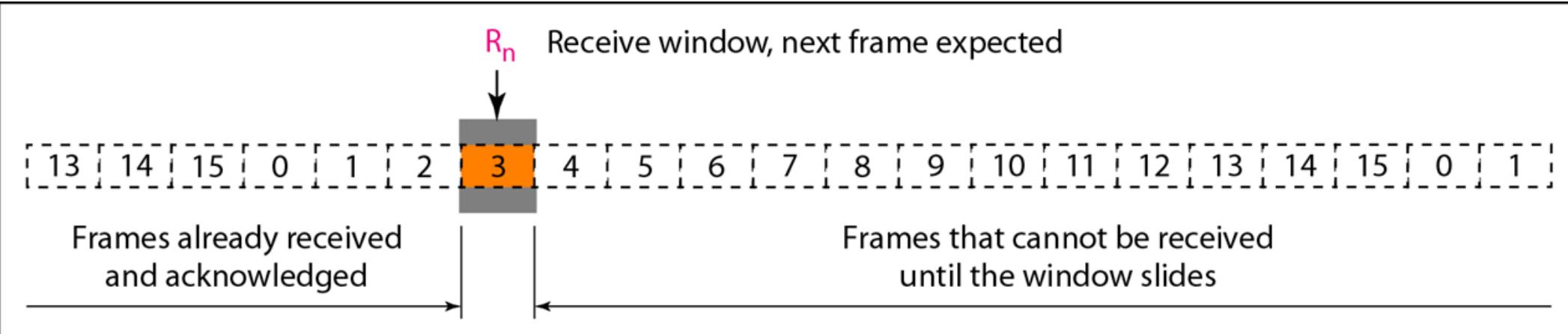
## Send Window :

- Sliding window is an abstract concept that defines the range of sequence numbers.
- The send window can slide one or more slots when a valid acknowledgment arrives.

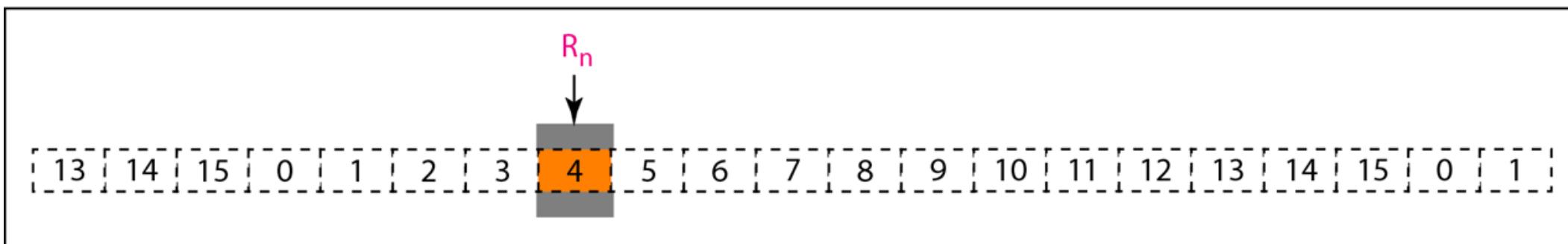
## Receive Window :

- 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.

## Receive Window for Go-Back-N ARQ:

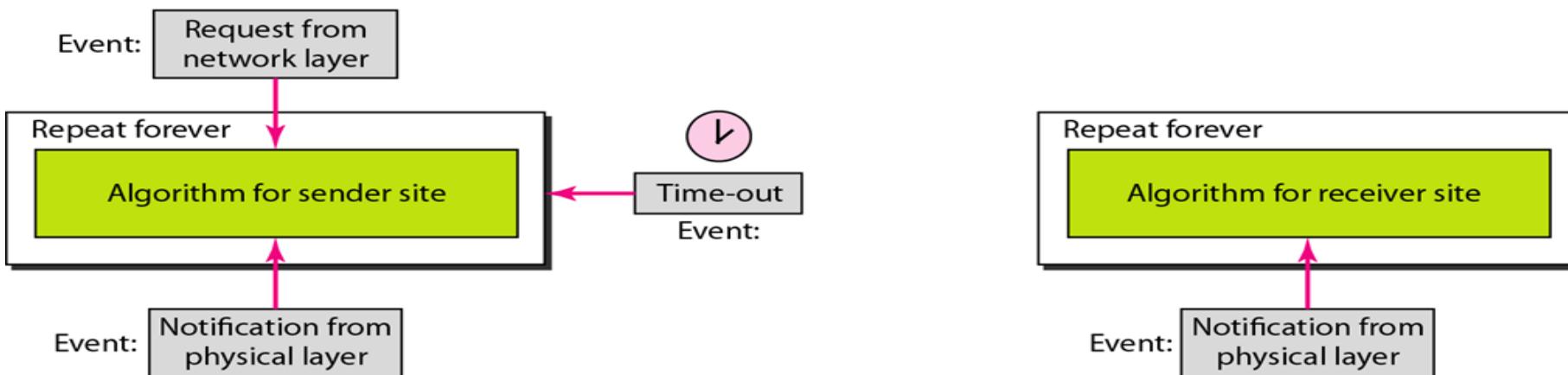
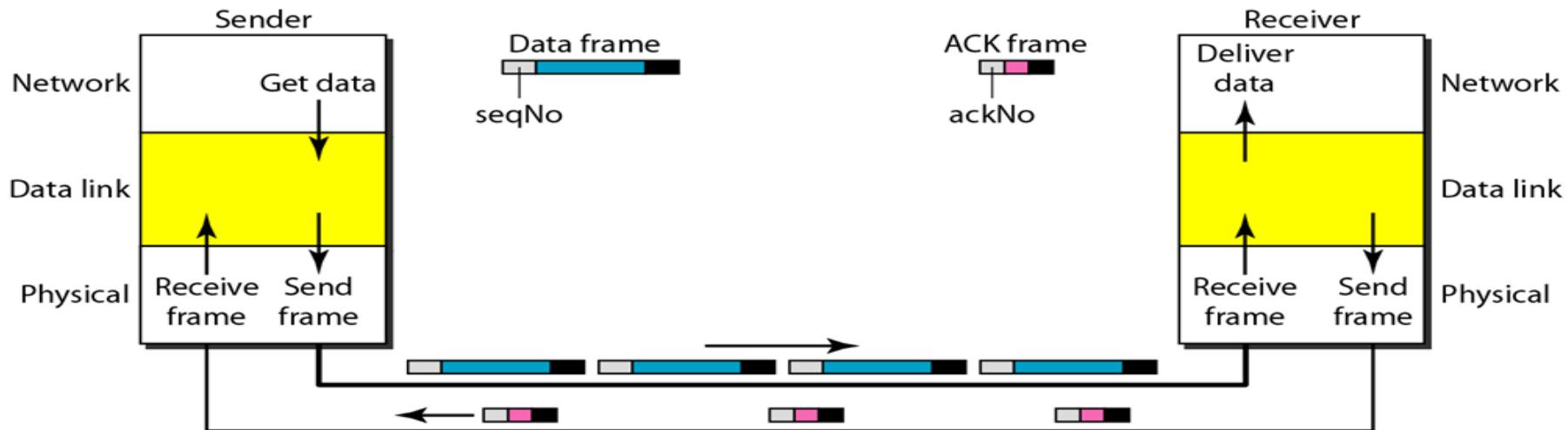
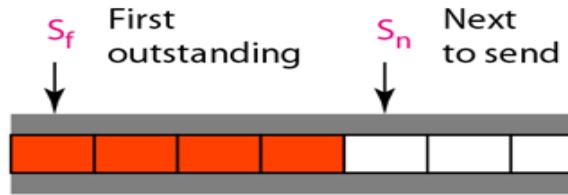


a. Receive window



b. Window after sliding

# Go-Back-N ARQ

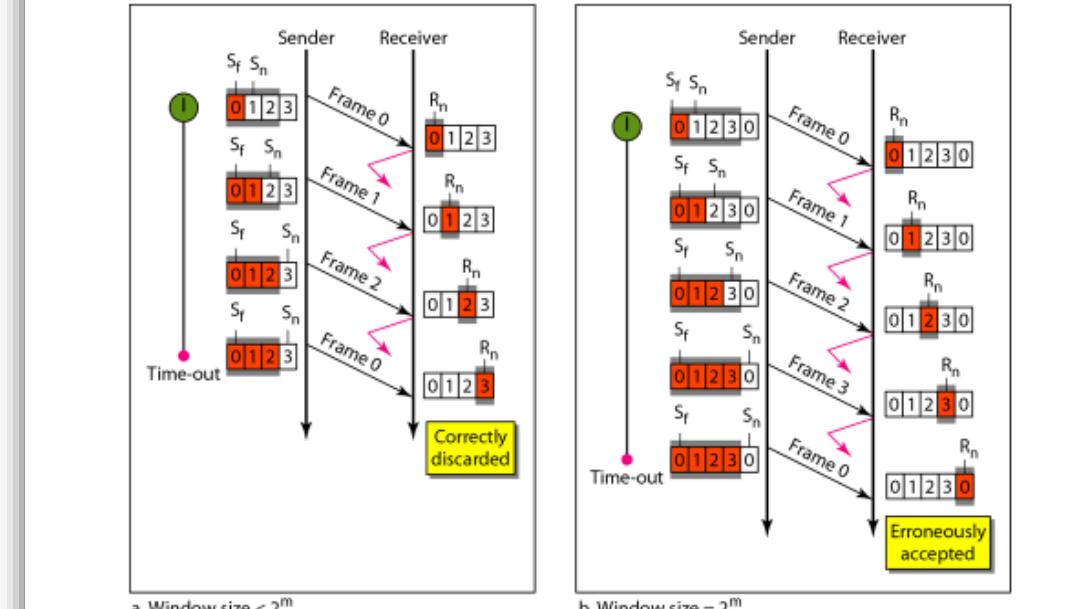


# Go-Back-N ARQ

## NOTE:

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

Figure 11.15 Window size for Go-Back-N ARQ



## Algorithm: Go-Back-N sender algorithm

```
1 Sw = 2m - 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 }
```

## Algorithm: Go-Back-N sender algorithm

(continued)

```
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 }
```

## Algorithm: Go-Back-N receiver algorithm

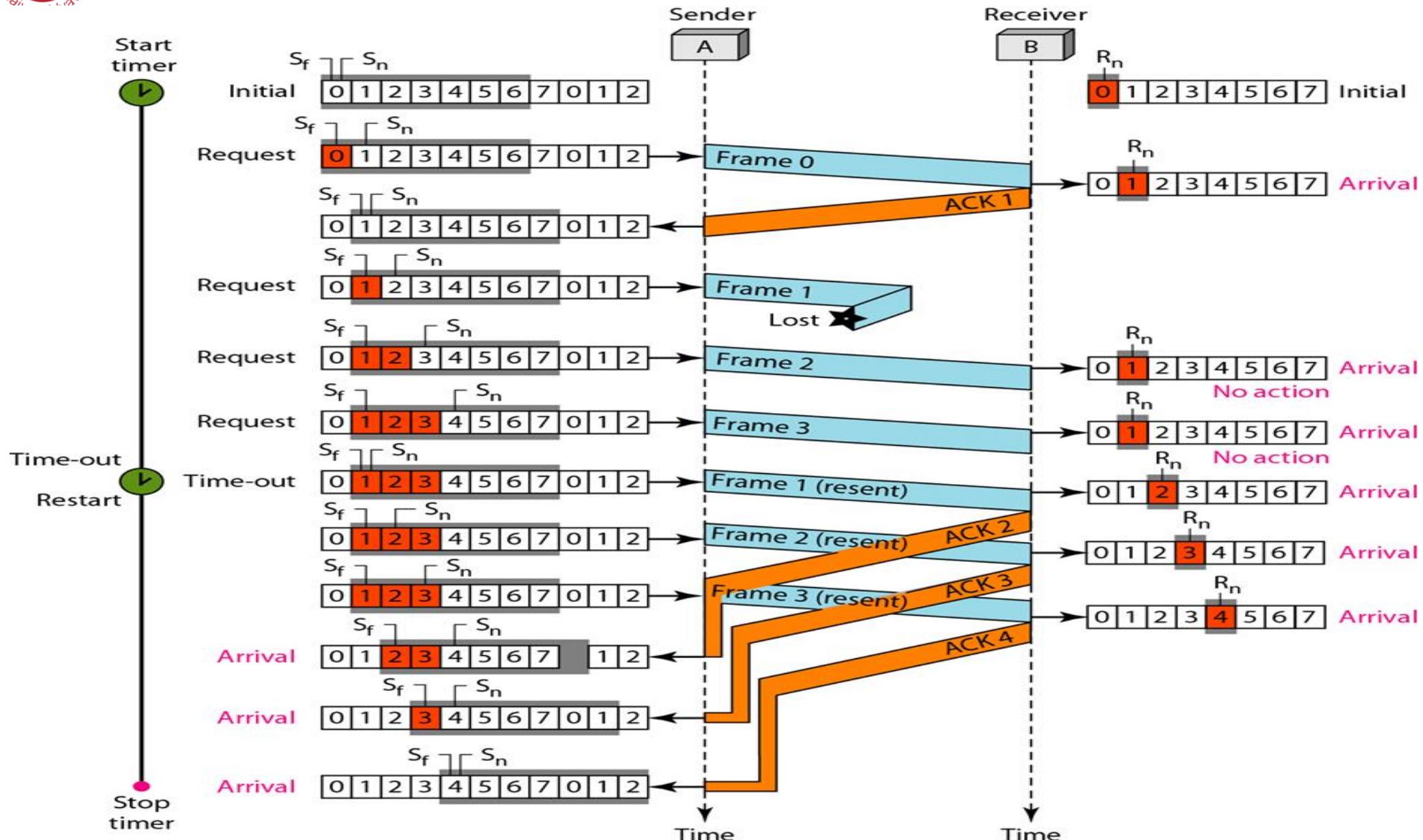
```
1 Rn = 0;  
2  
3 while (true) //Repeat forever  
4 {  
5   WaitForEvent();  
6  
7   if(Event(ArrivalNotification)) /Data frame arrives  
8   {  
9     Receive(Frame);  
10    if(corrupted(Frame))  
11      Sleep();  
12    if(seqNo == Rn) //If expected frame  
13    {  
14      DeliverData(); //Deliver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

# Go-Back-N ARQ

## *Example:*

- *The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state.*
- *We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives.*
- *Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.*

# Go-Back-N ARQ



# Selective Repeat ARQ

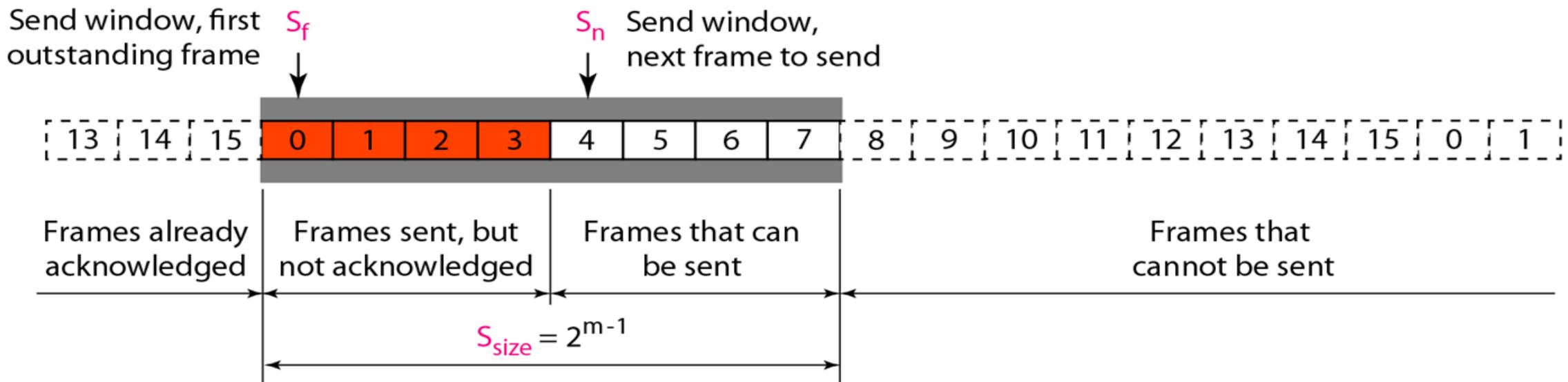
- Go-Back-N ARQ simplifies the process at the receiver site.
- The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded.
- However, this protocol is very inefficient for a noisy link.
- In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission.

# Selective Repeat ARQ

- Mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent.
- More efficient for noisy links, but the processing at the receiver is more complex.
- More efficient for noisy links, but the processing at the receiver is more complex.
- the size of the **send window** is much smaller; it is  $2^{m-1}$ .
- sizes of the **send window and receive window are the same**.

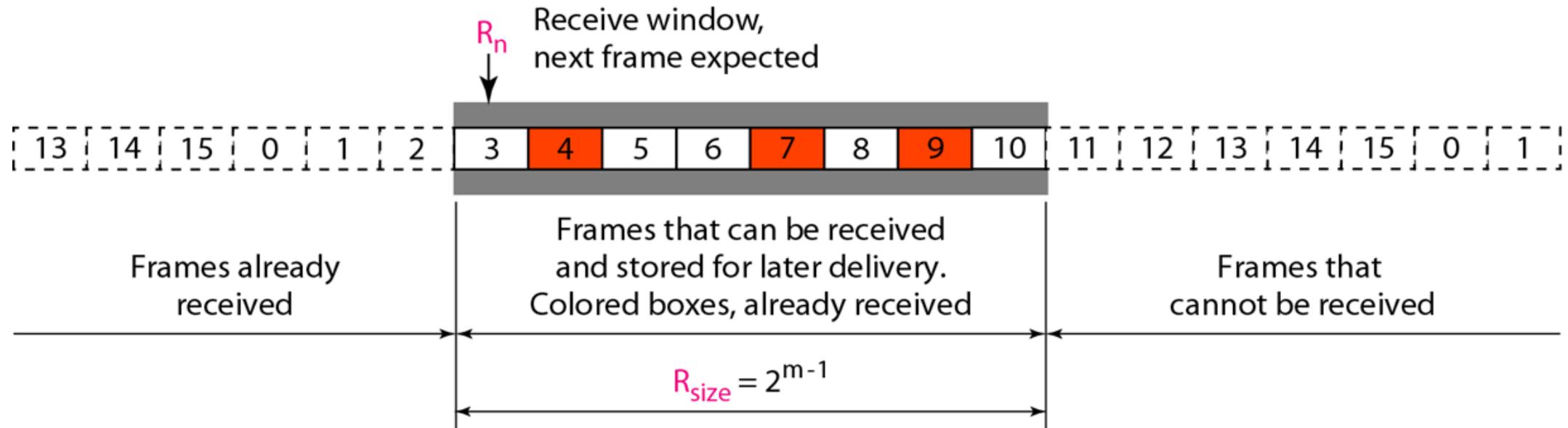
# Selective Repeat ARQ

Send window for Selective Repeat ARQ:

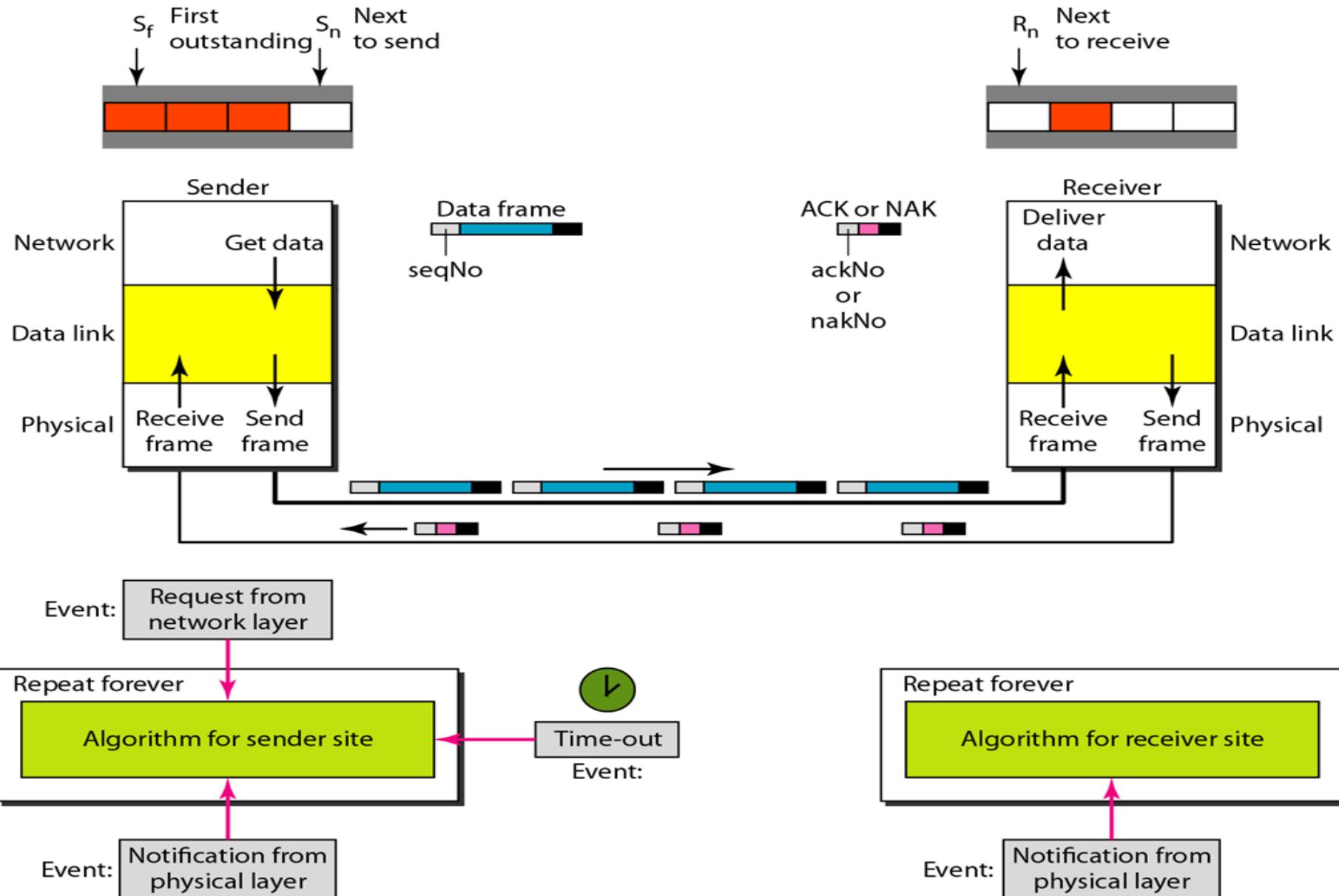


# Selective Repeat ARQ

Receive window for Selective Repeat ARQ:

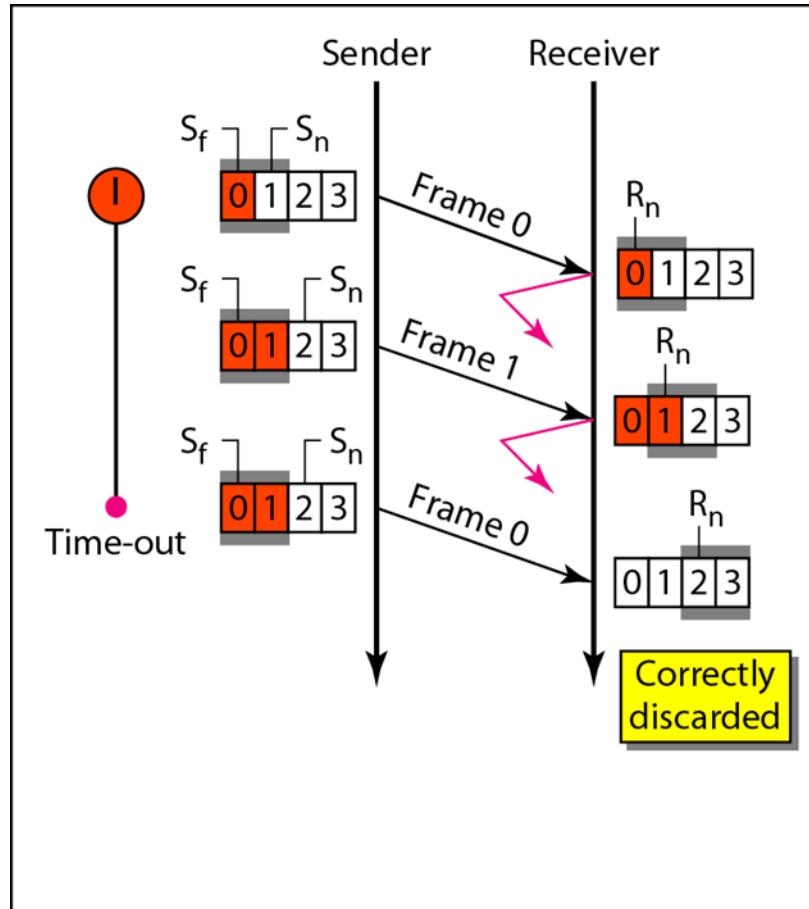


# Selective Repeat ARQ

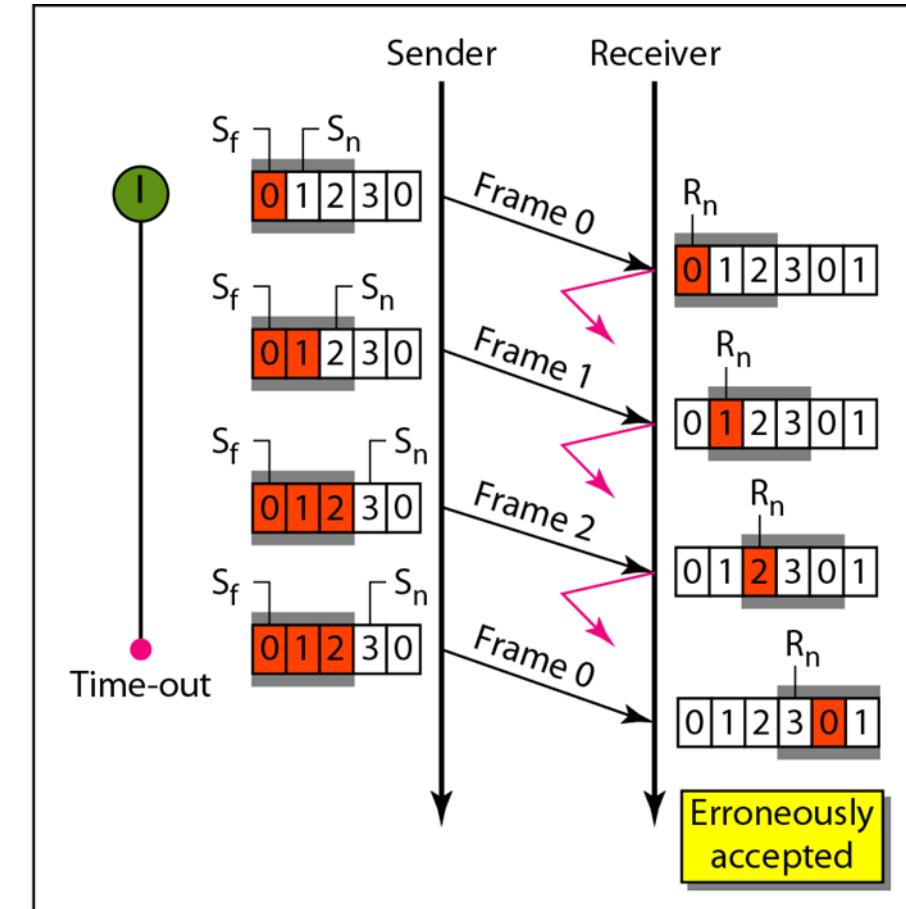


# Selective Repeat ARQ

Selective Repeat ARQ, window size:



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



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

## Sender Site SR ARQ Algorithm:

# Selective Repeat ARQ

```
1 Sw = 2m-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 SR ARQ Algorithm:

# Selective Repeat ARQ

```
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 }
```

# Selective Repeat ARQ

## Sender Site SR ARQ Algorithm:

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

## Receiver Site SR ARQ Algorithm:

# Selective Repeat ARQ

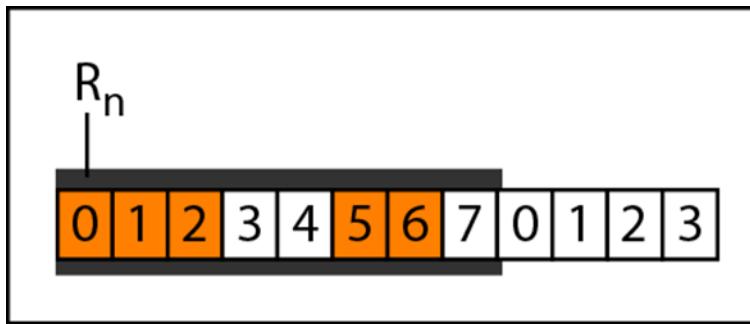
```
1 Rn = 0;  
2 NakSent = false;  
3 AckNeeded = false;  
4 Repeat(for all slots)  
    Marked(slot) = false;  
6  
7 while (true)                                //Repeat forever  
8 {  
9     WaitForEvent();  
10  
11    if(Event(ArrivalNotification))           /Data frame arrives  
12    {  
13        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 SR AR Algorithm

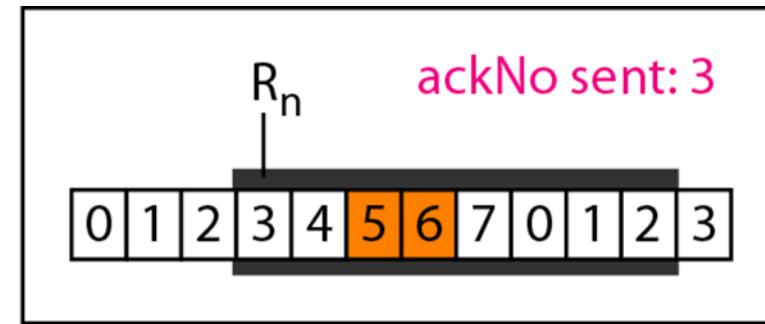
```
23     NakSent = true;
24     if ((seqNo in window)&&(!Marked(seqNo))
25     {
26         StoreFrame(seqNo)
27         Marked(seqNo)= true;
28         while(Marked(Rn))
29         {
30             DeliverData(Rn);
31             Purge(Rn);
32             Rn = Rn + 1;
33             AckNeeded = true;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

# Selective Repeat ARQ

Delivery of data in Selective Repeat ARQ:

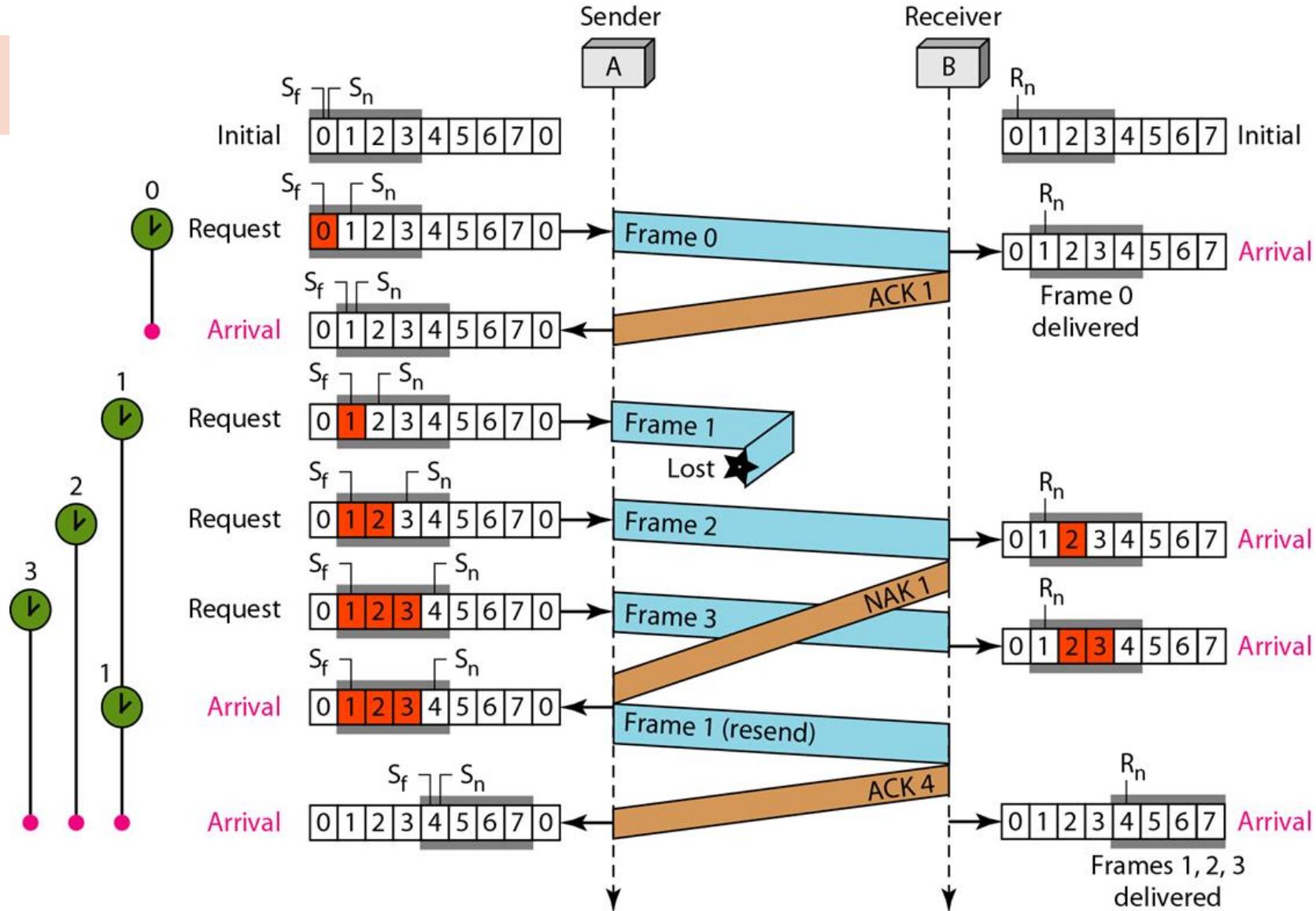


a. Before delivery

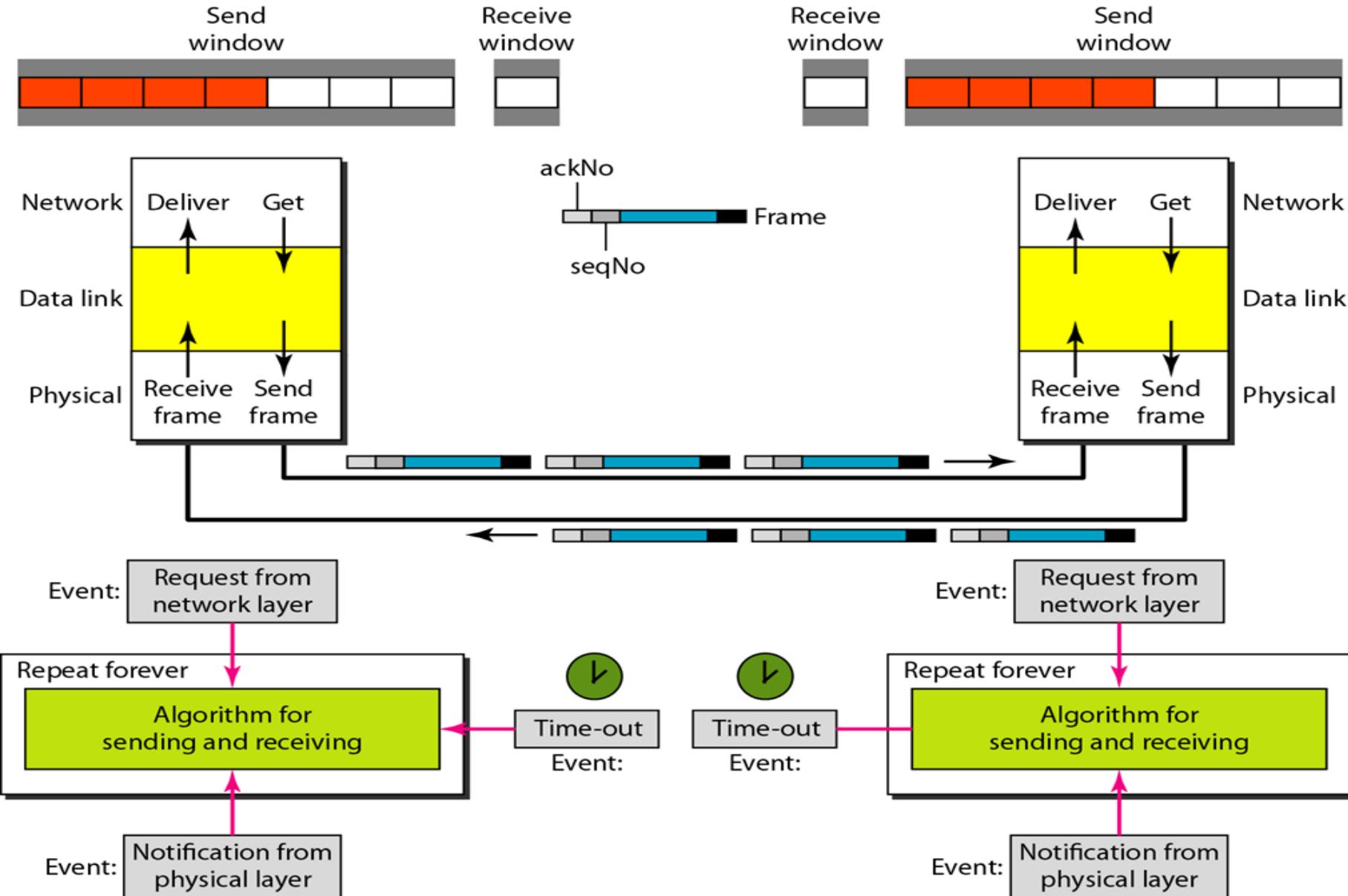


b. After delivery

## Flow Diagram:



# Piggybacking in Go Back N ARQ



- **High-level Data Link Control (HDLC)** is a bit-oriented protocol for communication over point-to-point and multipoint links.
- It implements the ARQ mechanisms.

### ***Configurations and Transfer Modes:***

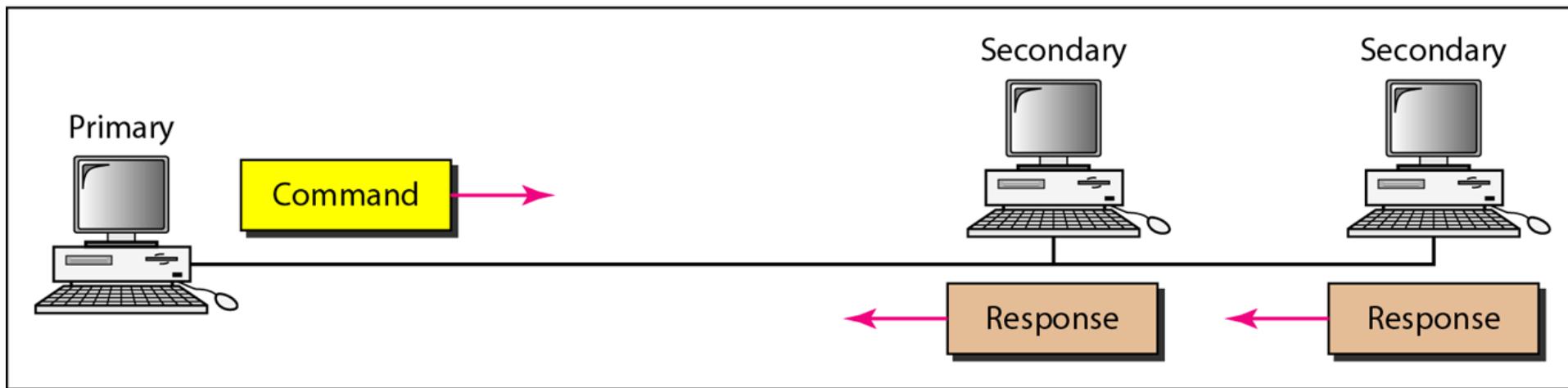
*HDLC provides two common transfer modes that can be used in different configurations:*

- *Normal response Mode(NRM)*
- *Asynchronous Balanced Mode(ABM)*

# Normal Response Mode

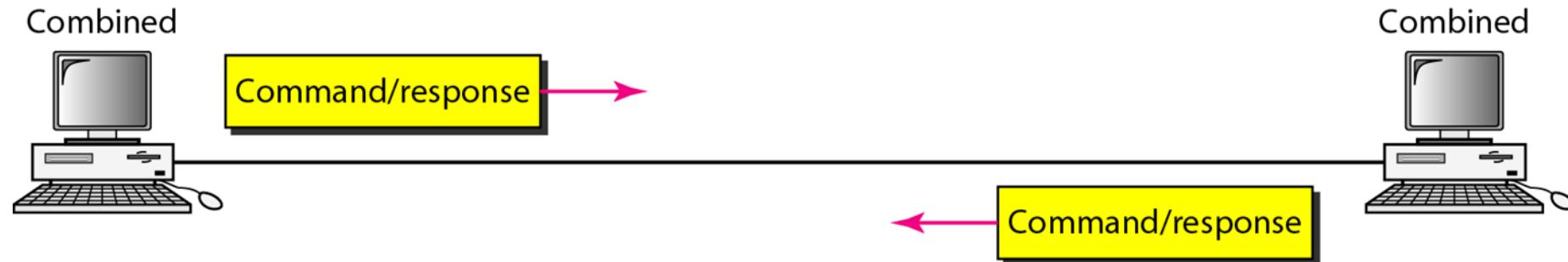


a. Point-to-point

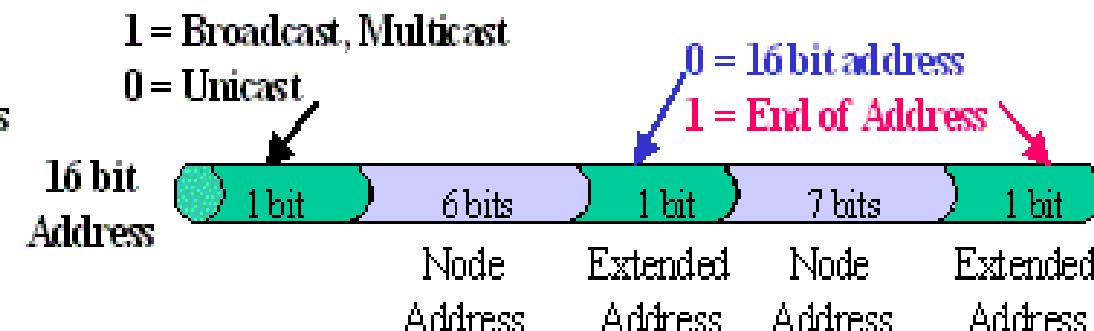
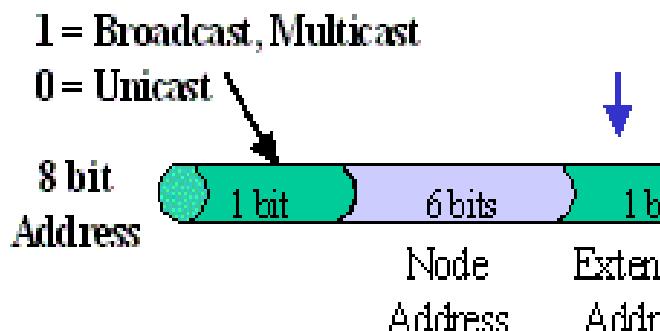
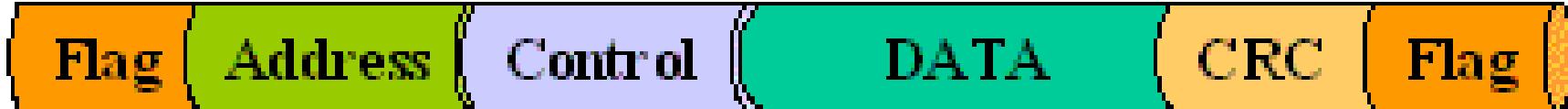


b. Multipoint

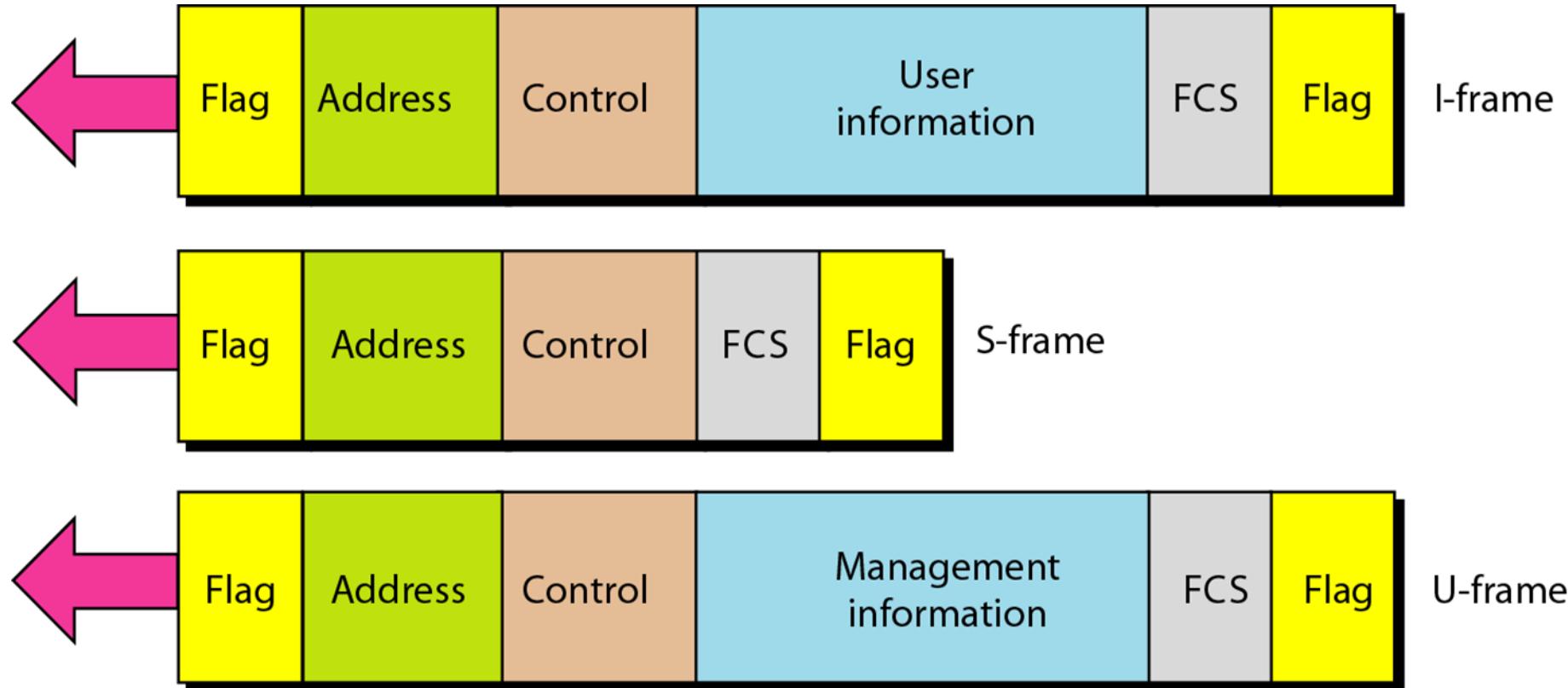
# Asynchronous Balanced Mode



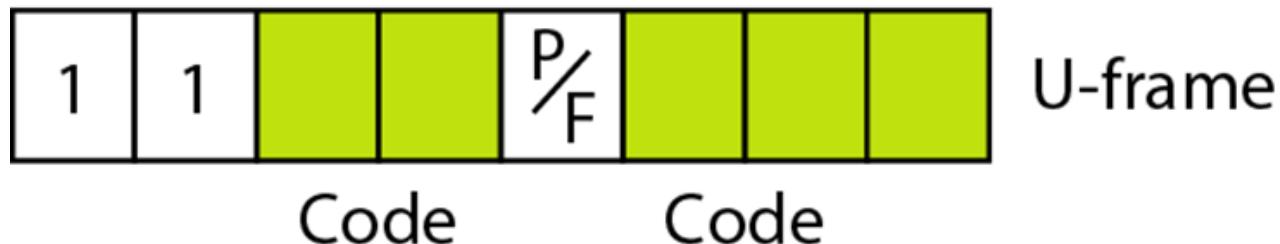
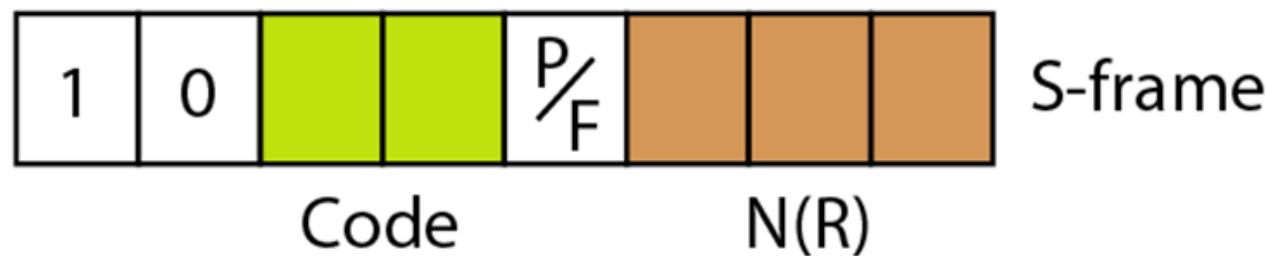
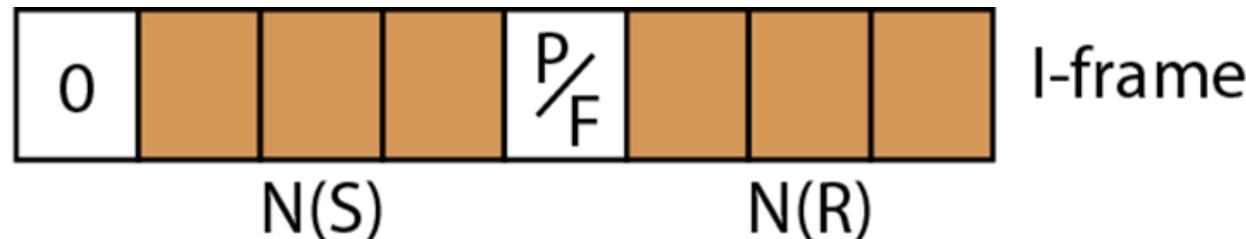
# HDLC frames



# HDLC frames



## Control field format for the different frame types



# U-frame control command and response

<i>Code</i>	<i>Command</i>	<i>Response</i>	<i>Meaning</i>
<b>00 001</b>	SNRM		Set normal response mode
<b>11 011</b>	SNRME		Set normal response mode, extended
<b>11 100</b>	SABM	<b>DM</b>	Set asynchronous balanced mode or <b>disconnect mode</b>
<b>11 110</b>	SABME		Set asynchronous balanced mode, extended
<b>00 000</b>	UI	<b>UI</b>	Unnumbered information
<b>00 110</b>		<b>UA</b>	<b>Unnumbered acknowledgment</b>
<b>00 010</b>	DISC	<b>RD</b>	Disconnect or <b>request disconnect</b>
<b>10 000</b>	SIM	<b>RIM</b>	Set initialization mode or <b>request information mode</b>
<b>00 100</b>	UP		Unnumbered poll
<b>11 001</b>	RSET		Reset
<b>11 101</b>	XID	<b>XID</b>	Exchange ID
<b>10 001</b>	FRMR	<b>FRMR</b>	Frame reject

# Multiple Access

Module 2

## Data Link Layer

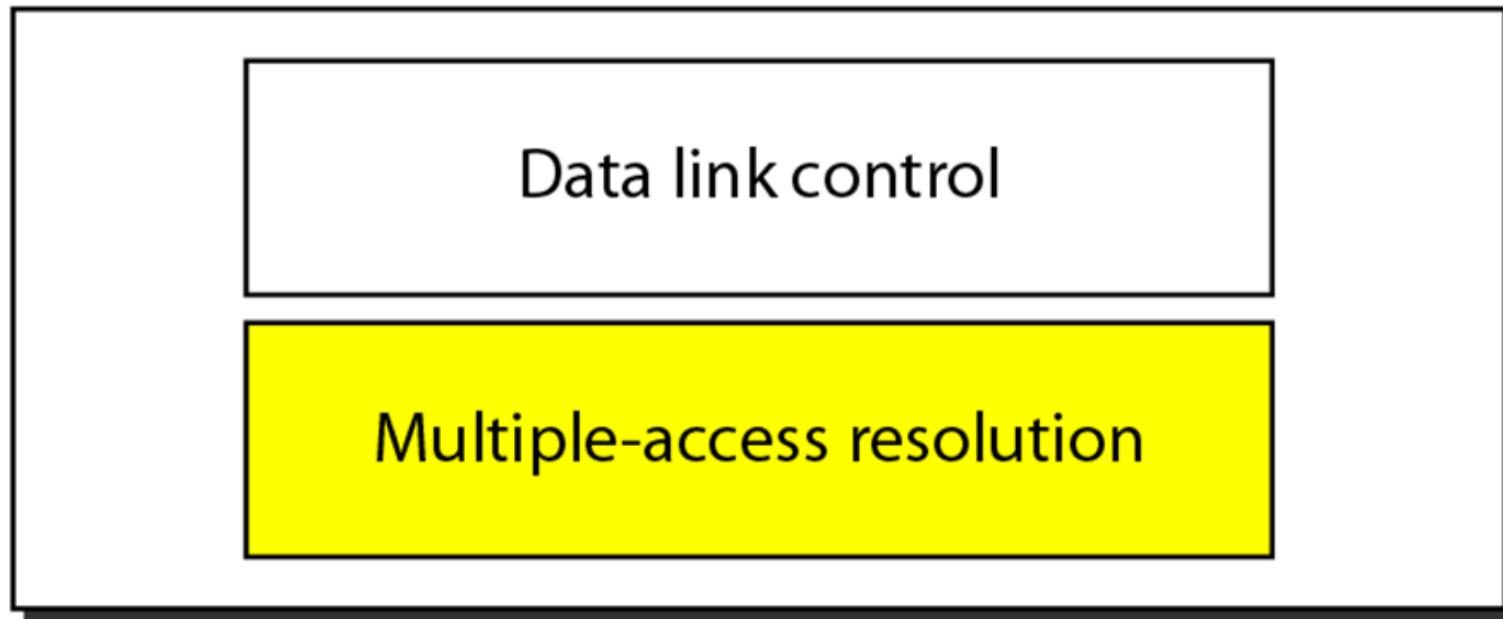
# Multiple Access

## Outline:

- Multiple access mechanisms
- Random access
- Controlled access
- Channelization

# Sublayers of Data Link Layer

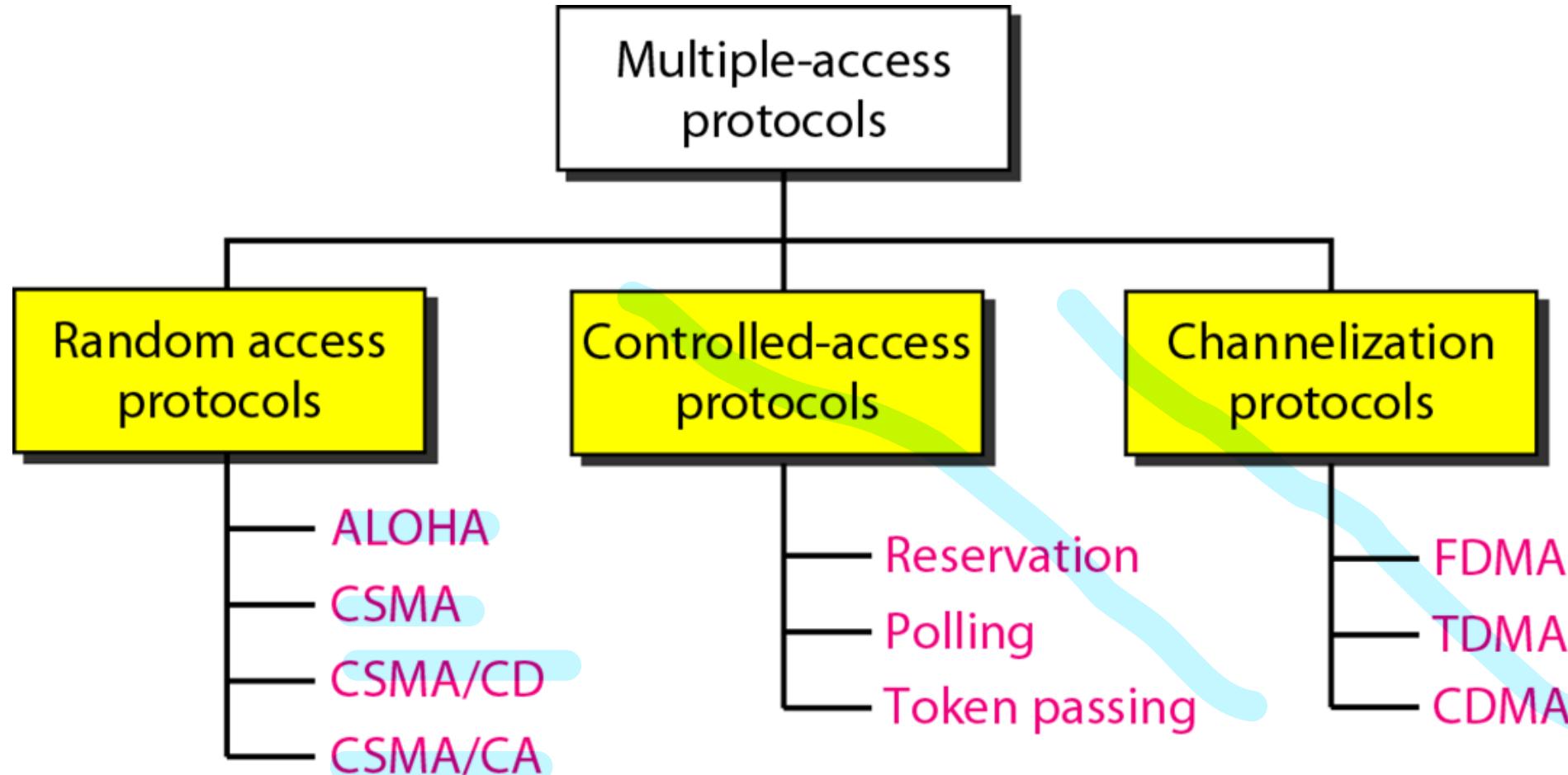
Data link layer



# Sublayers of Data Link Layer

- The upper sub layer that is responsible for flow and error control is called the **logical link control (LLC) layer**.
- Lower sub layer that is mostly responsible for multiple access resolution is called the **media access control (MAC) layer**.

# Multiple Access Mechanisms



## Random Access

- Also called *contention-based access*
- No station is assigned to control another
- At each instance, a station that has data to send uses a procedure defined by the protocol to make a decision on whether or not to send.

# Random Access

- Transmission is random among the stations - *random access*.
- Stations compete with one another to access the medium - *contention methods*.
- each station has the right to the medium without being controlled by any other station
  - more than one station tries to send
  - access conflict-collision - and the frames destroyed or modified.

## ALOHA Network

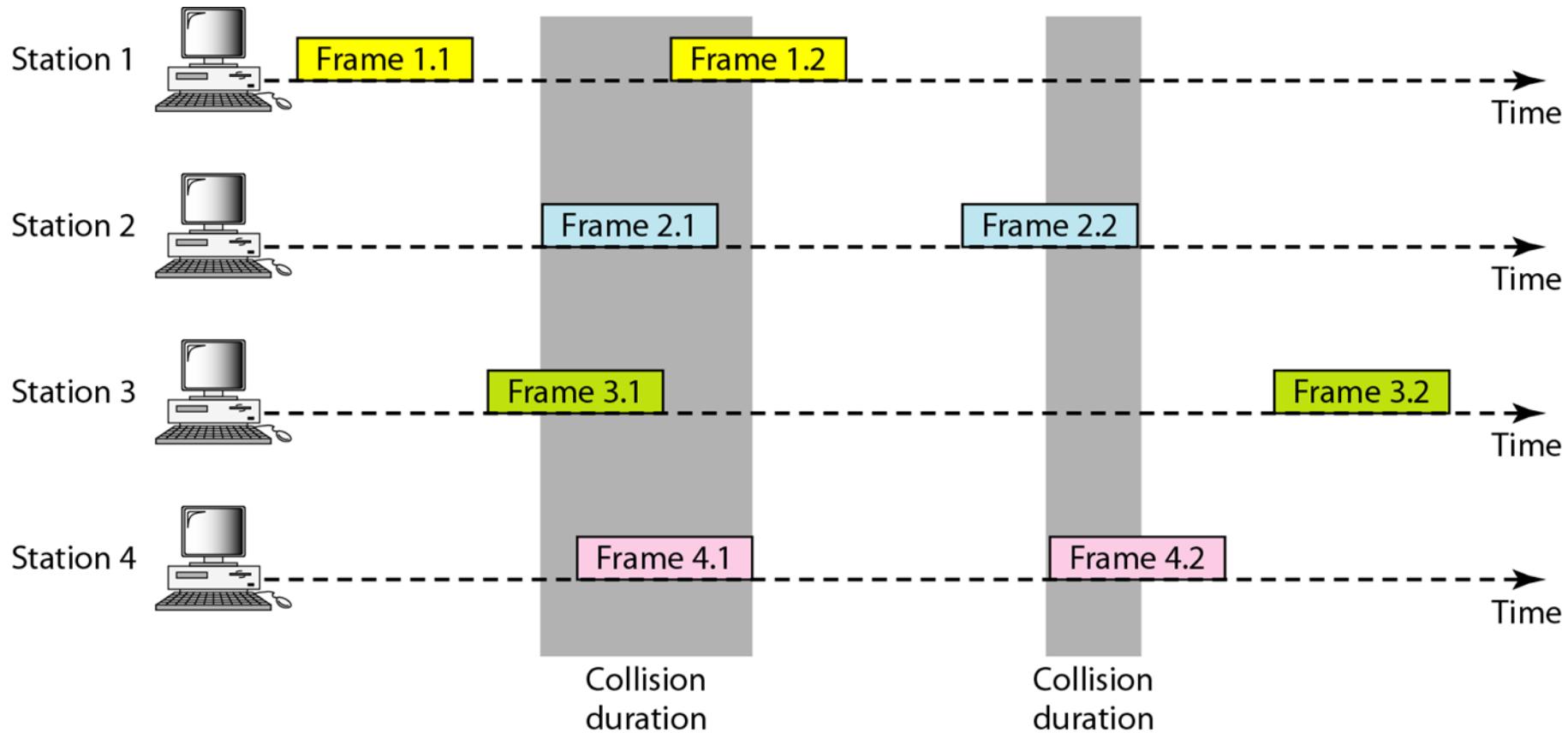
When a station sends data, another station may attempt to do so at the same time. The data from the two stations collide and become garbled.

- ✓ Pure Aloha
- ✓ Slotted Aloha

## Pure ALOHA

- The idea is that each station sends a frame whenever it has a frame to send.
- Since there is only one channel to share, there is the possibility of collision between frames from different stations.
- Even if one bit of a frame coexists on the channel with one bit from another frame, there is a collision, and both will be destroyed.

# Frames in Pure ALOHA



## Pure ALOHA

- Resend the frames that have been destroyed during transmission - acknowledgments from the receiver.
- Timeout – resend the frame
- Pure ALOHA - when the time-out period passes, each station waits a random amount of time before resending its frame.

## Pure ALOHA

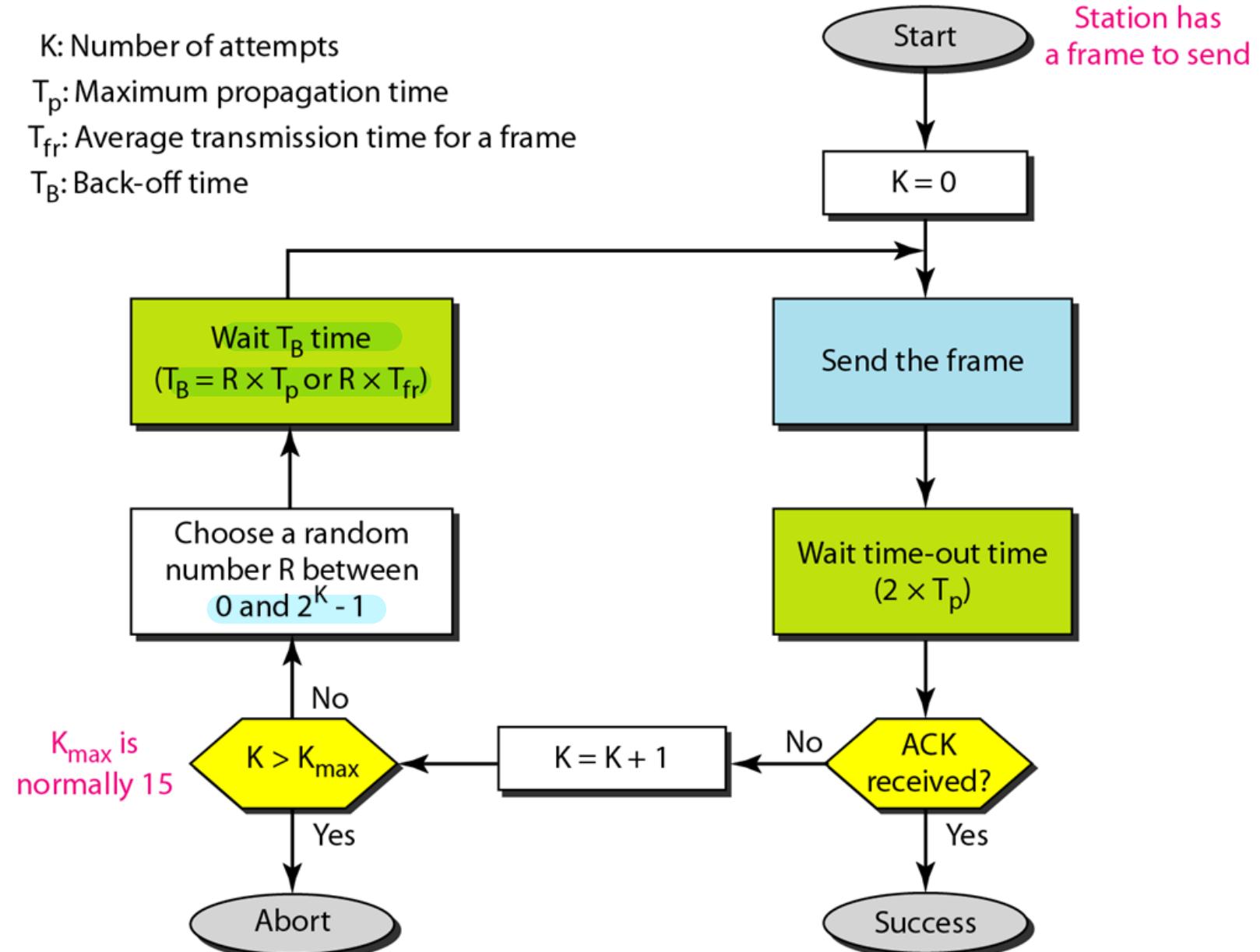
- The randomness will help avoid more collisions.
- *Random time – Back off time*  $T_B$
- Second method to prevent congesting the channel with retransmitted frames.
- Maximum number of retransmission,  $K_{\max}$ , after that give up and try later.

K: Number of attempts

$T_p$ : Maximum propagation time

$T_{fr}$ : Average transmission time for a frame

$T_B$ : Back-off time



## Example

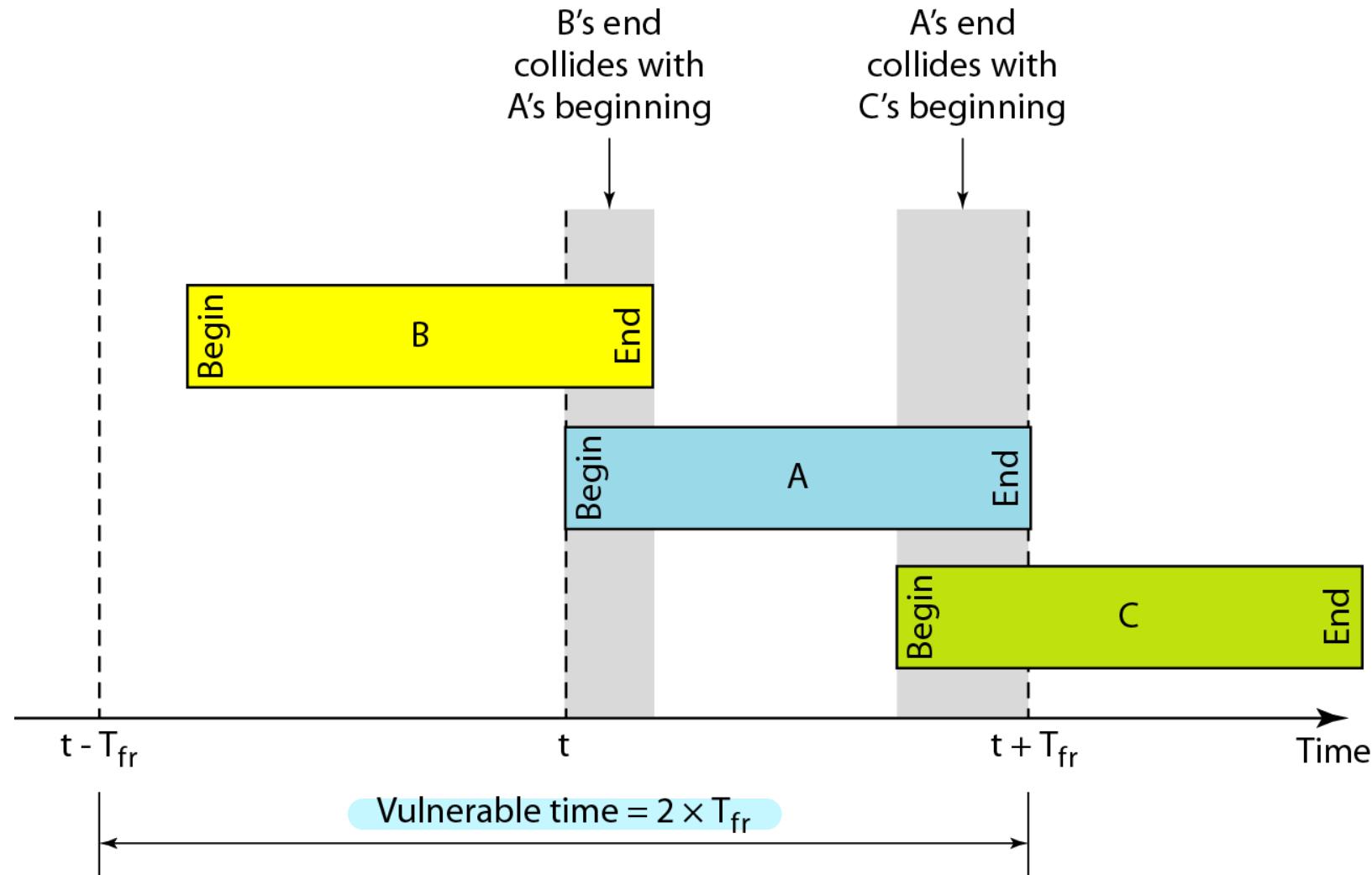
*Calculate possible values of  $T_B$  when stations on an ALOHA network are a maximum of 600 km apart; signal propagates at  $3 \times 10^8$  m/s.*

$$T_p = (600 \times 10^3) / (3 \times 10^8) = 2 \text{ ms}$$

When  $K=1$ ,  $T_B \in \{0 \text{ ms}, 2 \text{ ms}\}$

When  $K=2$ ,  $T_B \in \{0 \text{ ms}, 2 \text{ ms}, 4 \text{ ms}, 6\text{ms}\}$

# ALOHA: Vulnerable Time



# ALOHA: Throughput

- Assume number of stations trying to transmit follow *Poisson Distribution*

- The throughput for pure ALOHA is:

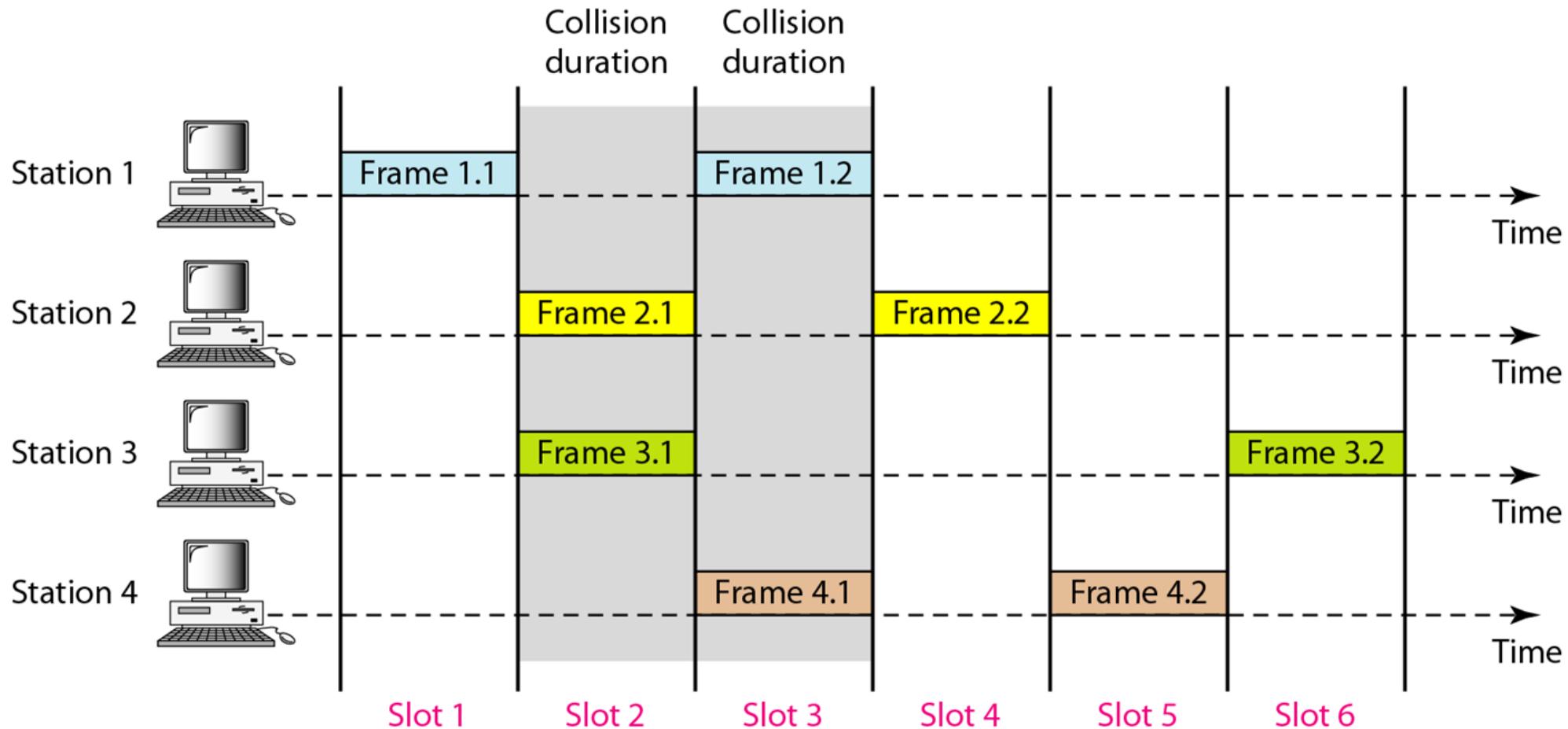
$$S = G \times e^{-2G}$$

where  $G$  is the *average number of frames requested/generated by the system per frame-time*

- The maximum throughput

$$S_{\max} = 0.184 \text{ when } G = 1/2$$

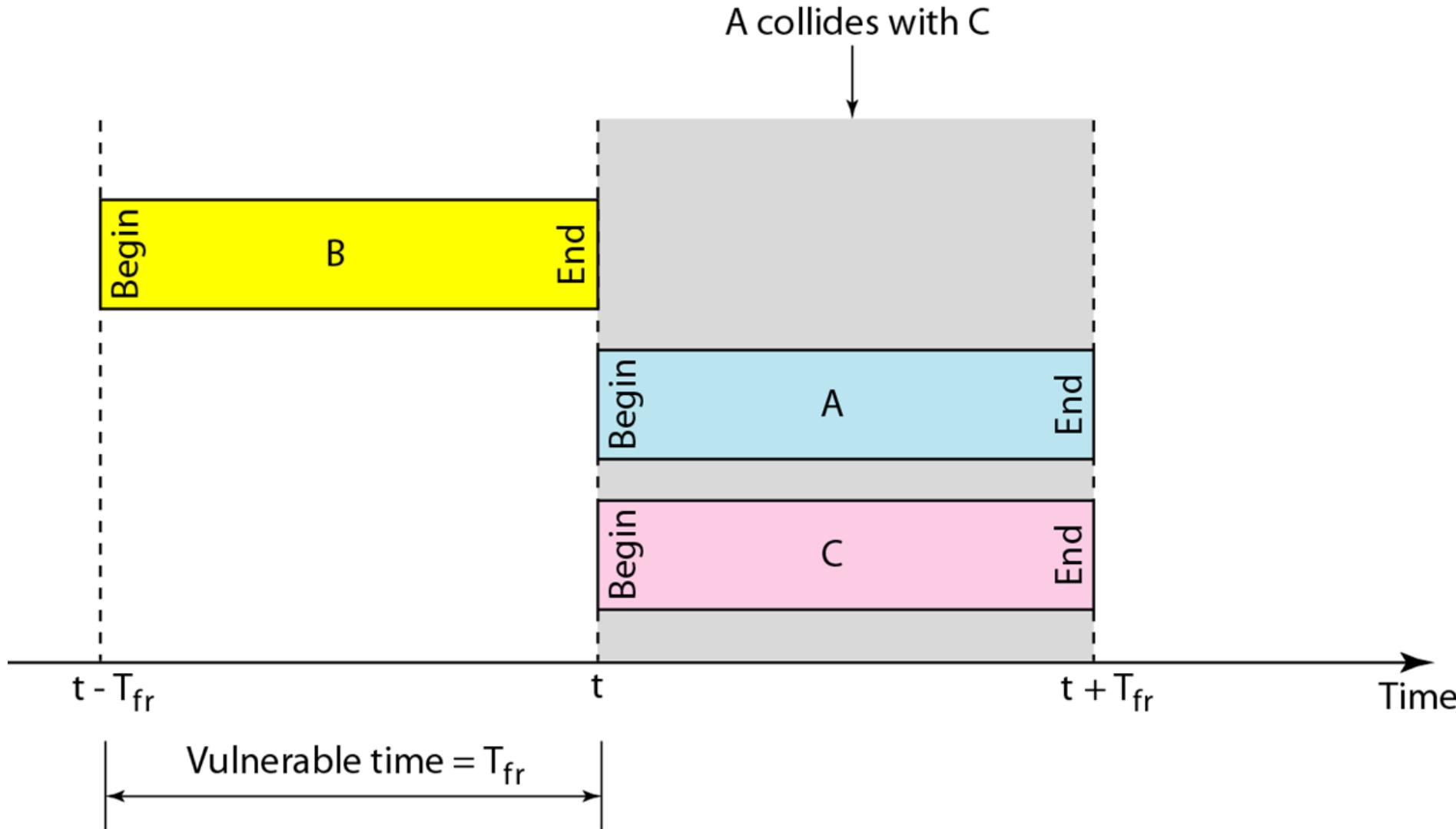
# Slotted ALOHA



## Slotted ALOHA

- Pure ALOHA has a vulnerable time of  $2 \times T_{fr}$  - no rule that defines when the station can send.
- Slotted ALOHA was invented to *improve the efficiency* of pure ALOHA.
- Slotted ALOHA divide time into slots of  $T_{fr}$  and force the station to send only at the beginning of the time slot.

# Slotted ALOHA: Vulnerable Time



## Slotted ALOHA: Throughput

- The throughput for Slotted ALOHA is

$$S = G \times e^{-G}$$

where  $G$  is the average number of frames requested per frame-time

- The maximum throughput

$$S_{\max} = 0.368 \text{ when } G= 1$$

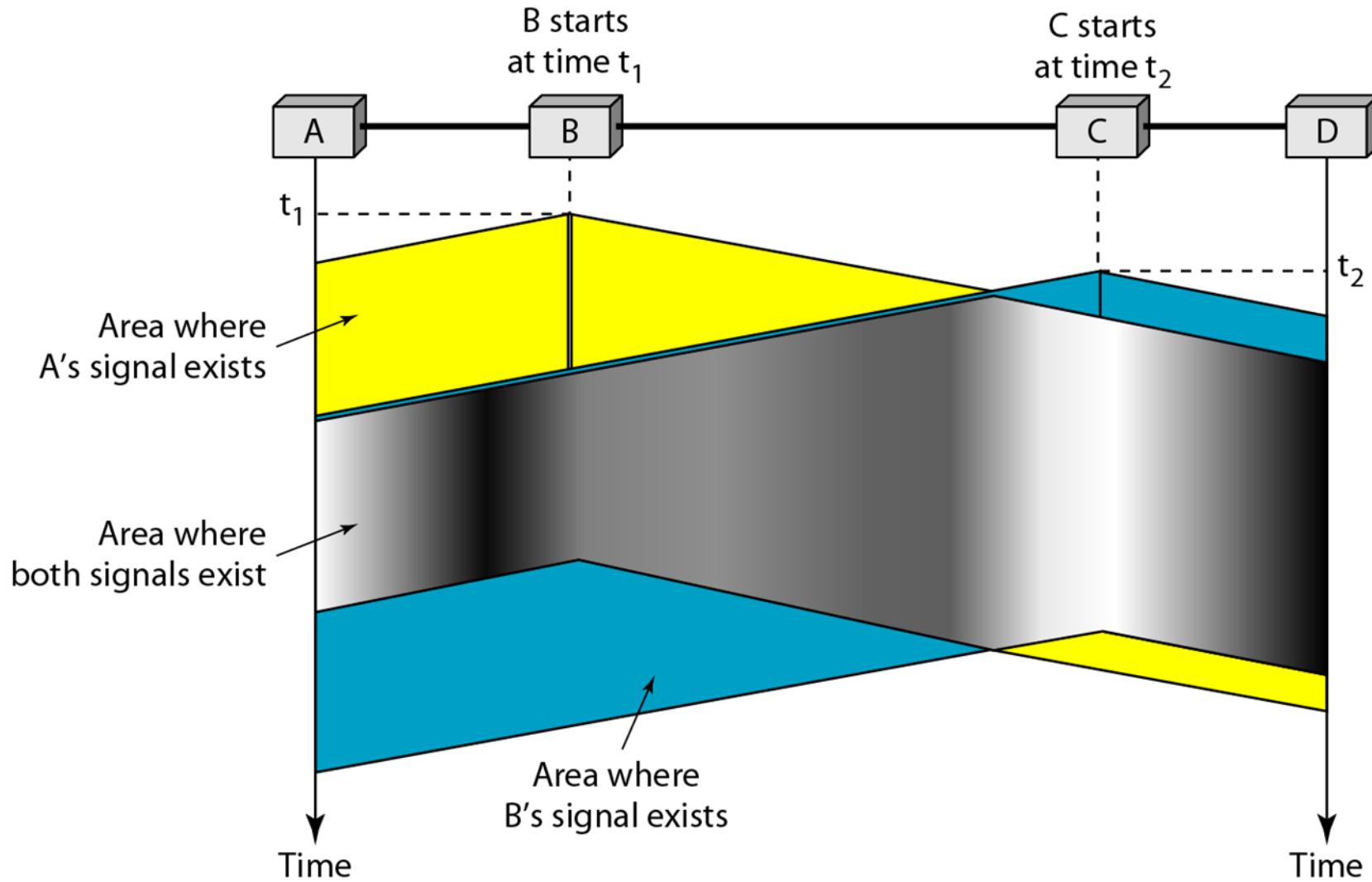
# CSMA

- **Carrier Sense Multiple Access**  
"Listen before talk"
- Reduce the possibility of collision, but *cannot completely eliminate* it
- Carrier sense multiple access (CSMA) requires that each station first check the state of the medium before sending.

## CSMA

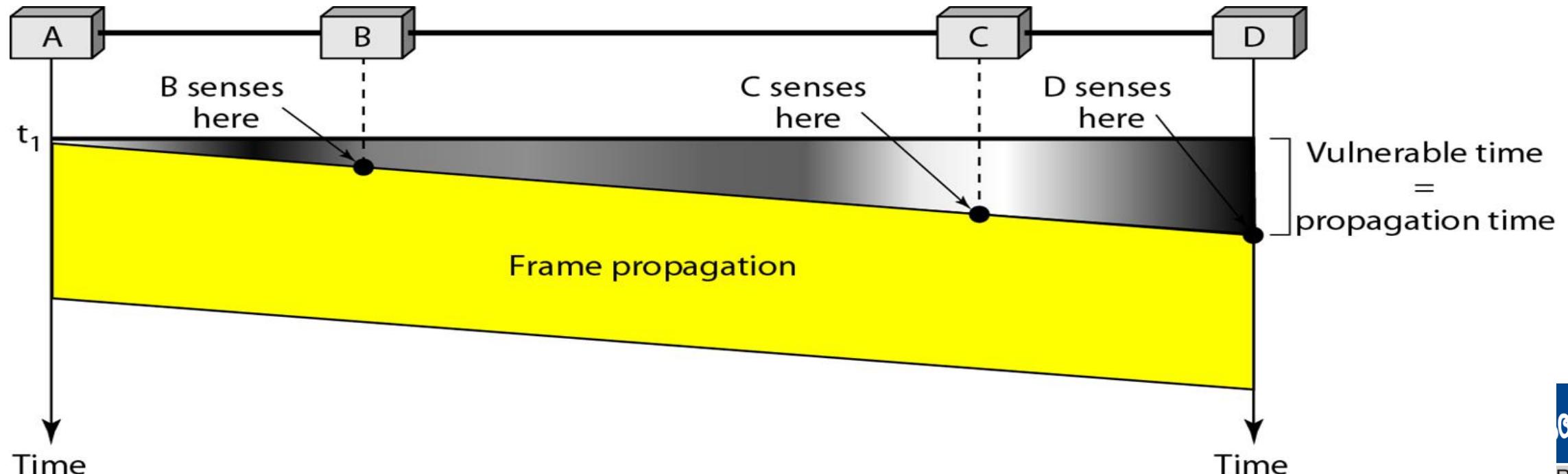
- The possibility of collision still exists because of propagation delay when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and sense it.

# Collision in CSMA

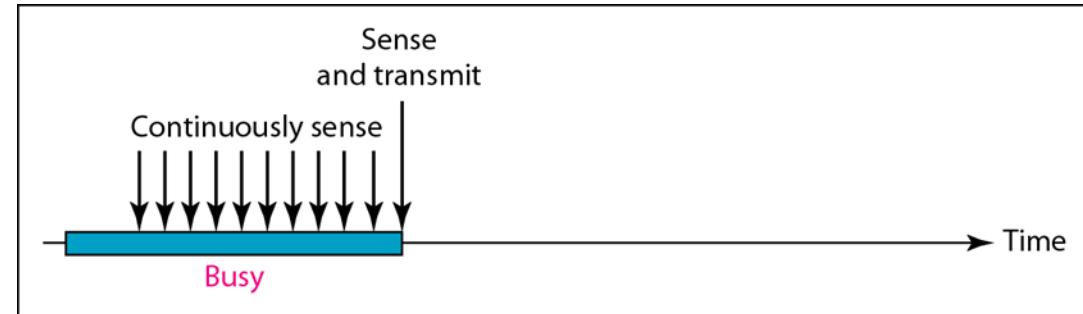


# Collision in CSMA

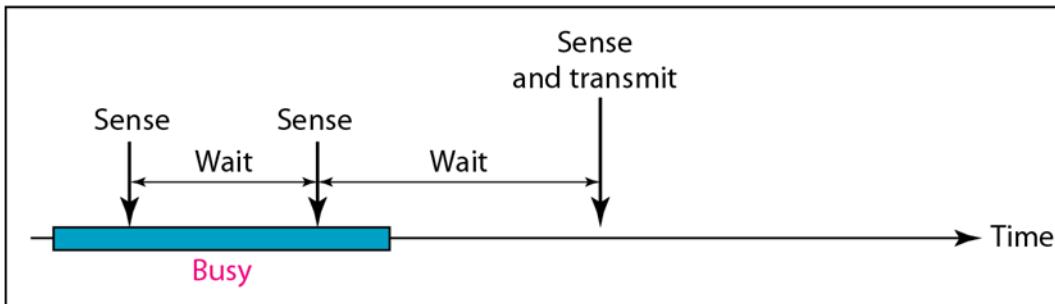
*The vulnerable time for CSMA is the propagation time  $T_p$*



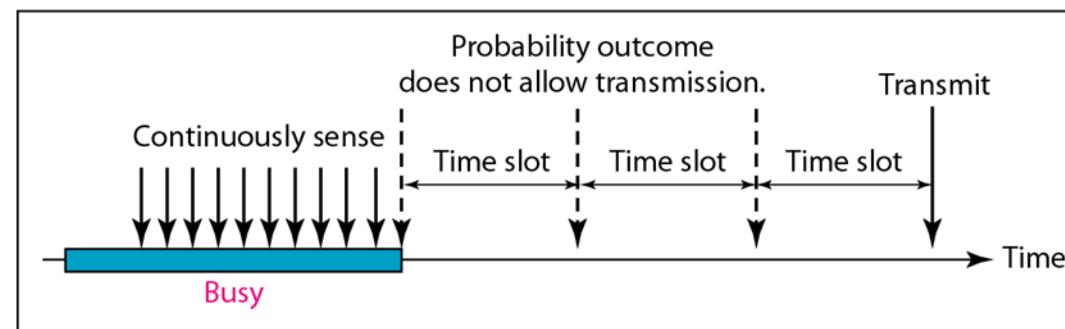
# Persistence Methods



a. 1-persistent



b. Nonpersistent

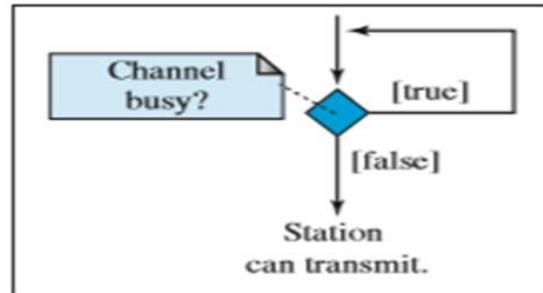


c. p-persistent

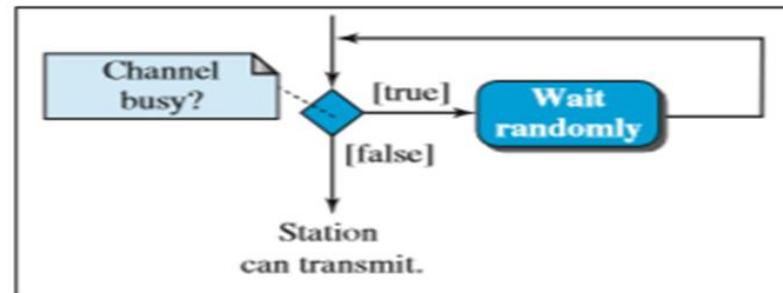
*What a station does  
when channel is idle  
or busy*

# Persistence Methods

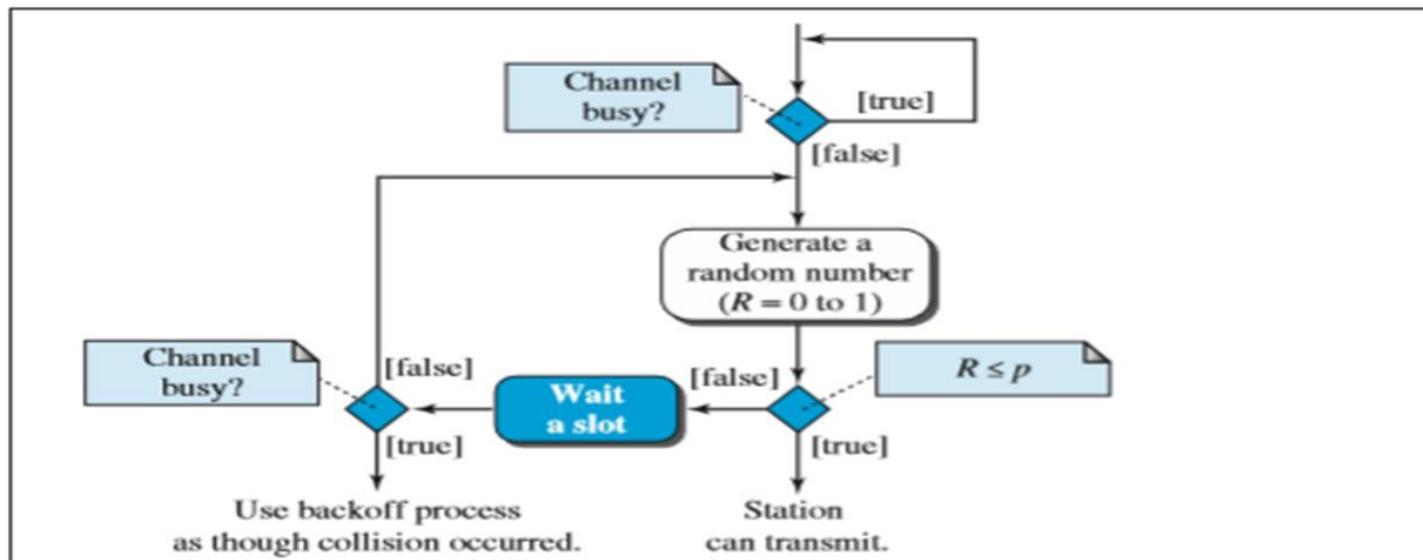
**Figure 12.10** Flow diagram for three persistence methods



a. 1-Persistent



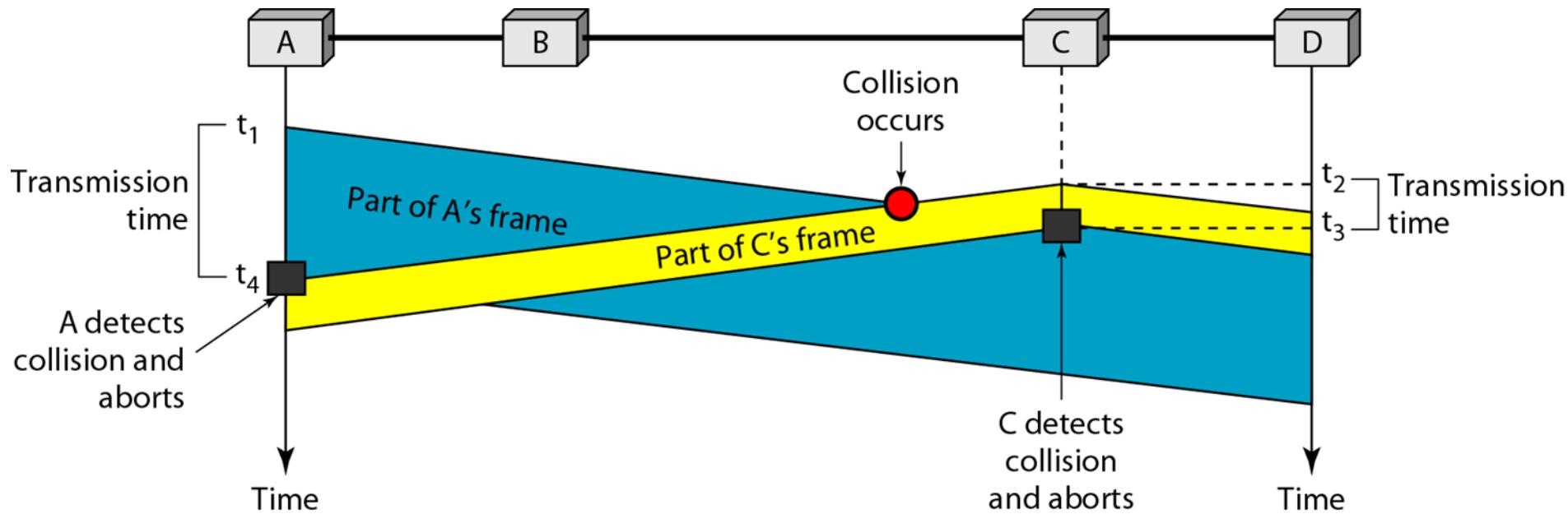
b. Nonpersistent



c.  $p$ -Persistent

# CSMA/CD

- Carrier Sense Multiple Access with Collision Detection
- Station monitors channel while sending a frame.



## CSMA/CD

- (CSMA/CD) augments the algorithm to handle the collision.
- a station monitors the medium after it sends a frame to see if the transmission was successful.
- For CSMA/CD to work, we need a restriction on the frame size.

## CSMA/CD

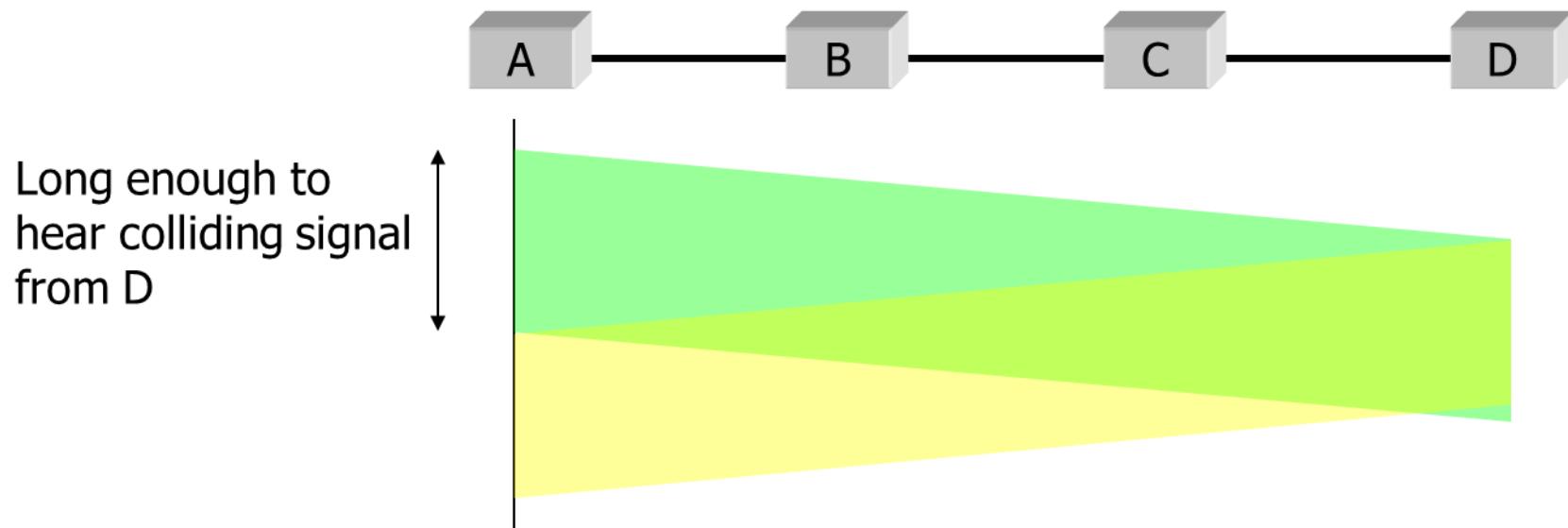
- Before sending last bit, station must detect a collision because once entire frame is sent, station does not keep copy of frame.
- Frame Transmission time  $T_{fr}$  must be atleast

**2\* TP**

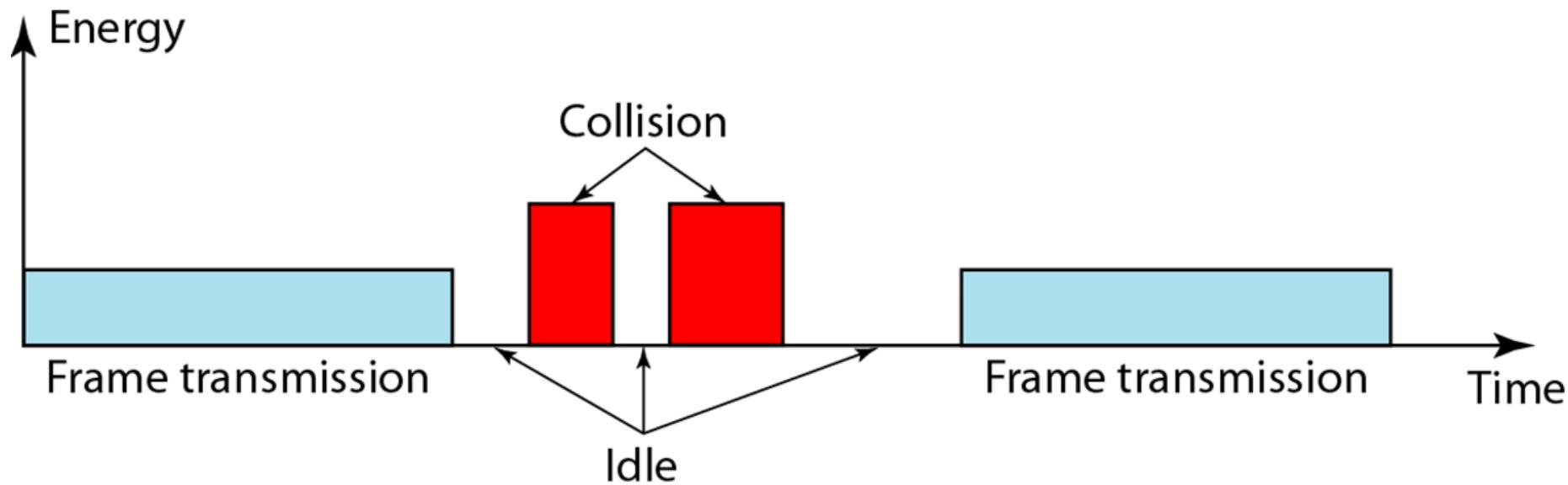
i.e. twice the maximum Propagation time.

# CSMA/CD: Minimum Frame Size

- Each frame must be large enough for a sender to detect a collision.
- Worst case scenario:
  - "A" is transmitting
  - "D" starts transmitting just before A's signal arrives



# CSMA/CD: Energy Levels

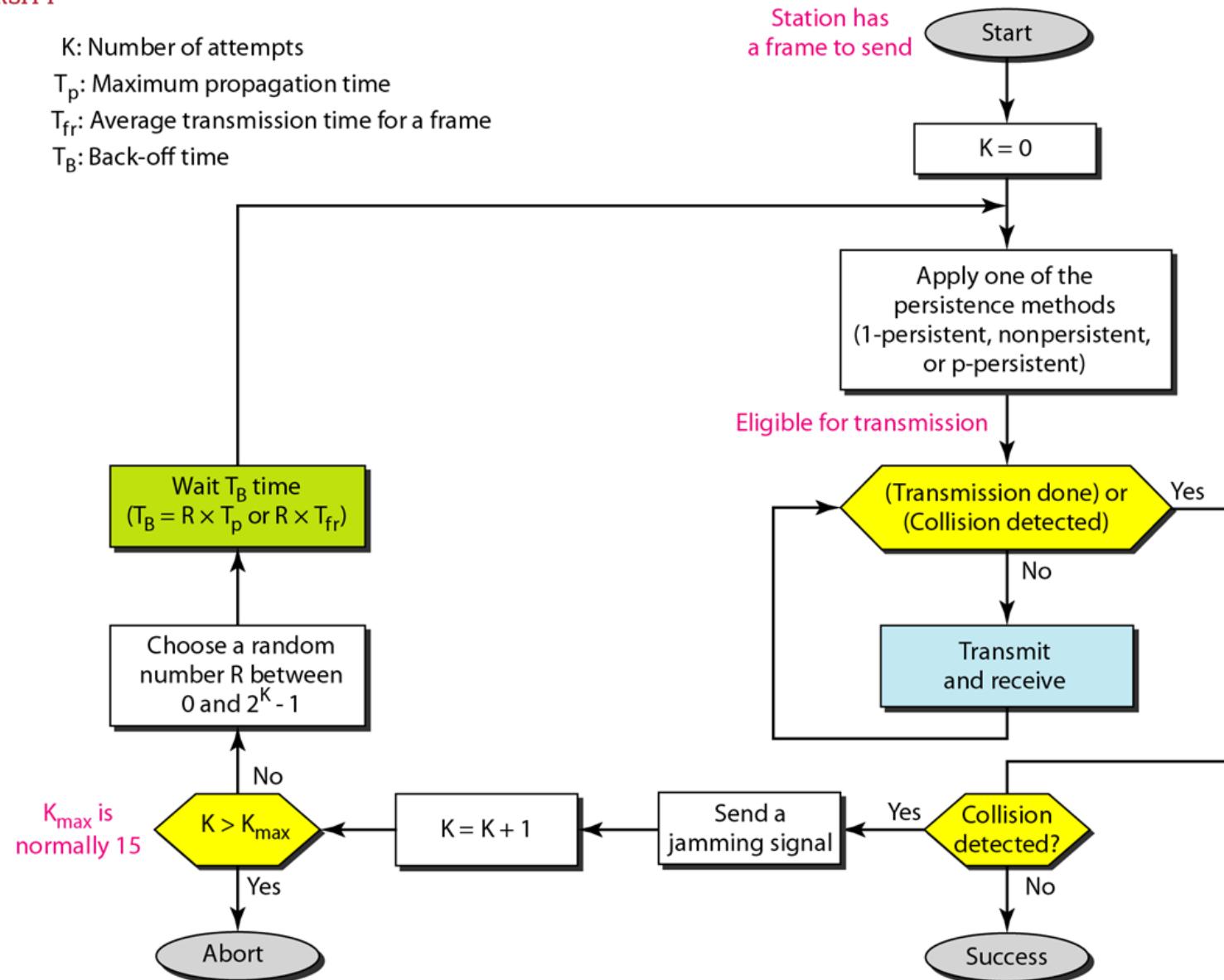


K: Number of attempts

$T_p$ : Maximum propagation time

$T_{fr}$ : Average transmission time for a frame

$T_B$ : Back-off time



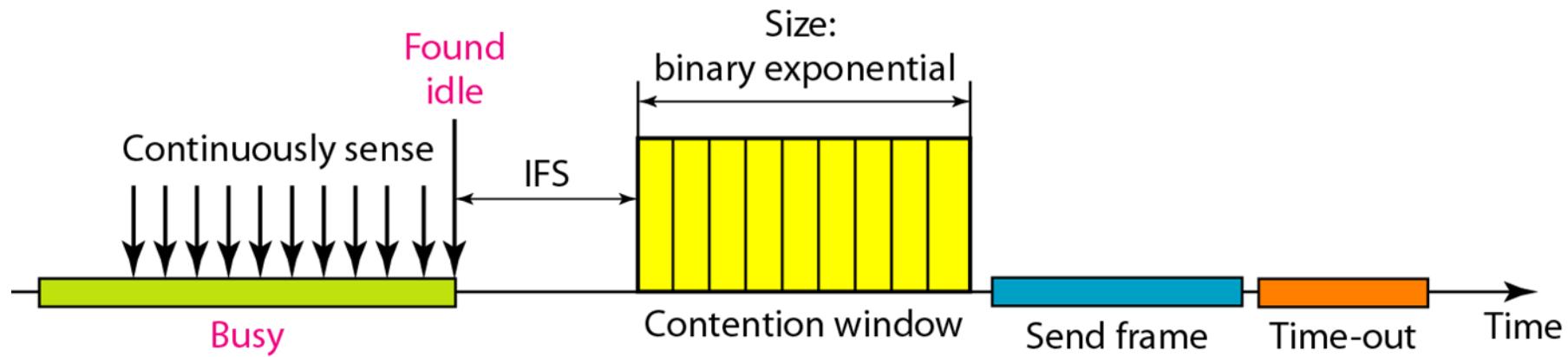


## Difference ALOHA and CSMA/CD

- first difference is the addition of the persistence process (**for sensing the medium**).
- second difference is the frame transmission.
  - In ALOHA, *first transmit the entire frame and then wait for an acknowledgment.*
  - In CSMA/CD, *transmission and collision detection is a continuous process.*
- The third difference is the sending of a **short jamming signal** that enforces the collision in case other stations have not yet sensed the collision.

# CSMA/CA

- Carrier Sense Multiple Access with Collision Avoidance
- Used in a network where collision cannot be detected.
  - E.g., wireless LAN



IFS – Interframe Space

# CSMA/CA

- need to avoid collisions on wireless networks because they cannot be detected.
- Collisions are avoided through the use of CSMA/CA's three strategies:
  - Inter-frame space
  - contention window
  - acknowledgments

## Interframe Space (IFS)

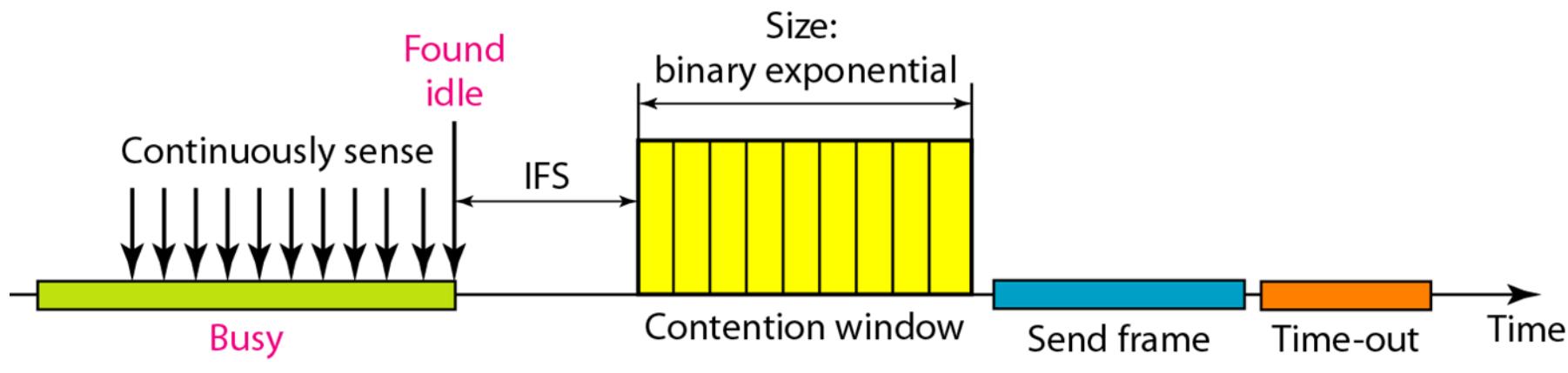
- collisions are avoided by deferring transmission even if the channel is found idle.
- idle channel is found - does not send immediately – waits for period of time called IFS.
- The IFS variable can also be **used to prioritize stations or frame types**. For example, a station that is assigned a shorter IFS has a higher priority.

## **Contention Window**

- The contention window is an **amount of time divided into slots**.
- A station that is ready to send chooses a **random number of slots** as its wait time.
- No. of slots in window changes according to **binary exponential back-off strategy**.
- Channel set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time.

## Contention Window

- station needs to **sense the channel** after each time slot.
- In CSMA/CA, if the station finds the channel busy, it does not restart the timer of the contention window; it stops the timer and restarts it when the channel becomes idle.



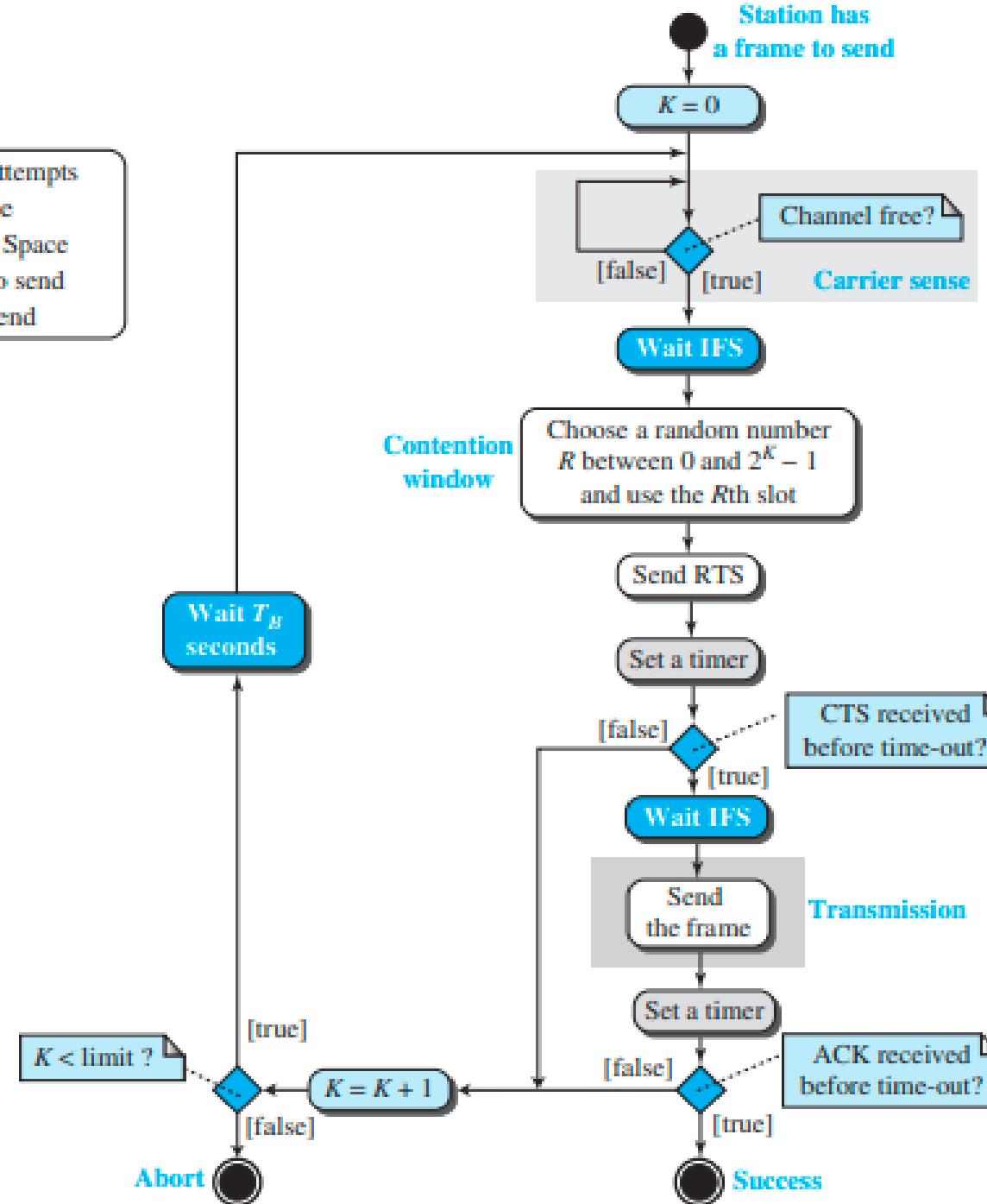
IFS – Interframe Space

## Acknowledgment

- there still may be a collision resulting in destroyed data.
- the data may be corrupted during the transmission.
- The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame.

## Legend

- $K$ : Number of attempts
- $T_B$ : Backoff time
- IFS: Interframe Space
- RTS: Request to send
- CTS: Clear to send



## Controlled Access

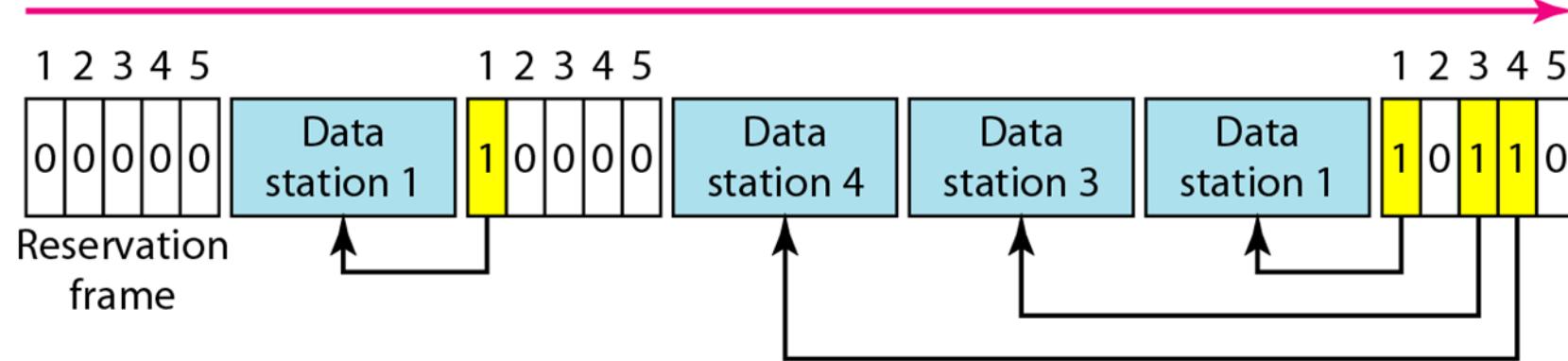
In controlled access, the stations consult one another to find which station has the right to send.

- A station must be authorized by someone (e.g., other stations) before transmitting
- Three common methods:
  - Reservation
  - Polling
  - Token passing

## Reservation

- a station needs to make a reservation before sending data.
- Time is divided into intervals.
- a reservation frame precedes the data frames sent in that interval.
- N stations in the system, there are exactly N reservation mini-slots

# Reservation Method



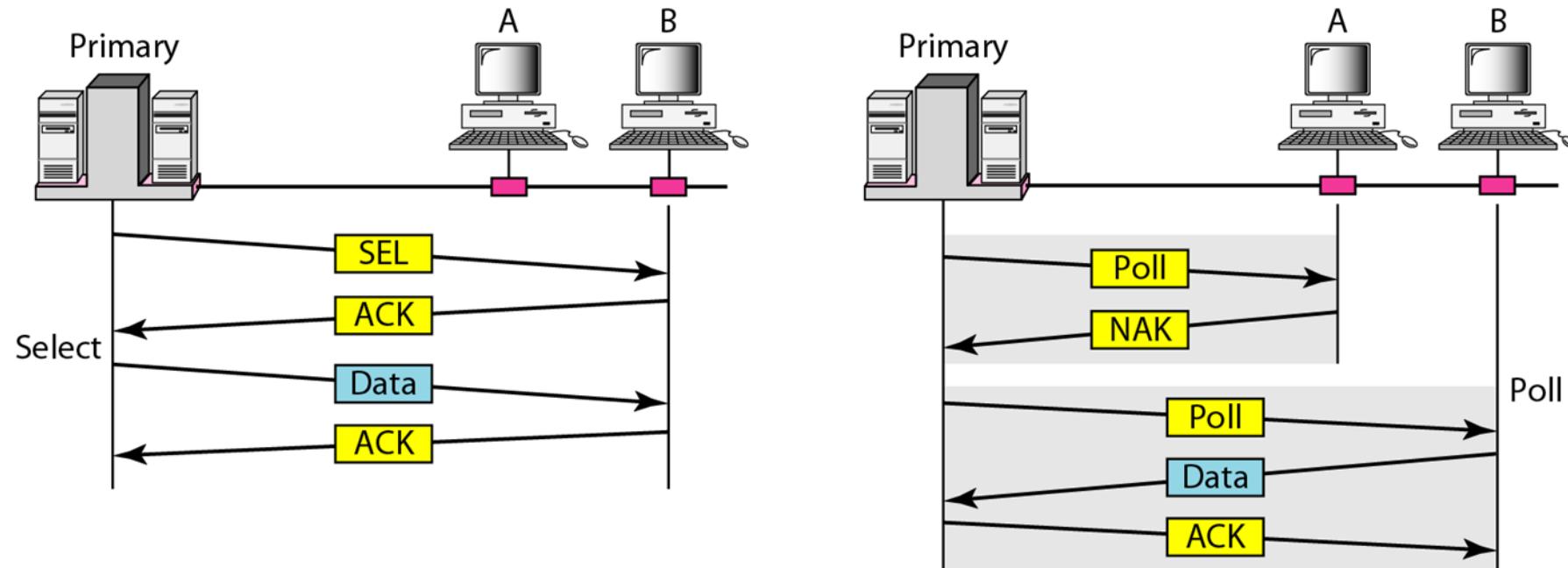
## Polling Method

- Polling works with topologies in which one device is designated as a primary station and the other devices are secondary stations.
- All data exchanges must be made through the primary device.
- It is up to the primary device to determine which device is allowed to use the channel

## Polling Method

- primary device - always the initiator of a session
- The **select** function is used whenever the primary device has something to send.
- The **poll** function is used by the primary device to solicit transmissions from the secondary devices.

# Polling Method



## Token Passing

- the stations in a network are organized in a logical ring.
- for each station, there is a predecessor and a successor.
- current station is the one that is accessing the channel now.
- right to this access has been passed from the predecessor to the current station.

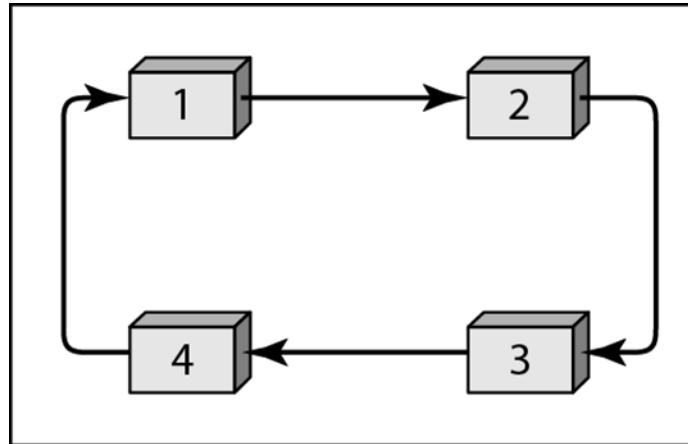
# Token Passing

- right will be passed to the successor when the current station has no more data to send.
- a special packet called a token circulates through the ring.
- Token management is needed for this access method.
- Stations must be limited in the time they can have possession of the token.

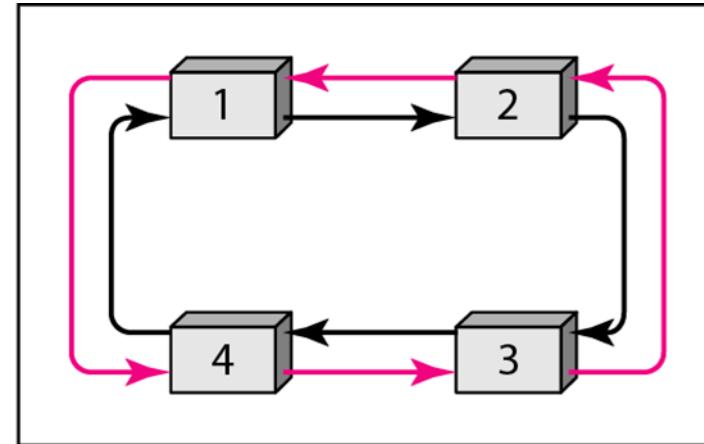
# Token Passing

- token must be monitored to ensure it has not been lost or destroyed.
- assign priorities to the stations and to the types of data being transmitted.
- make low-priority stations release the token to high priority stations.

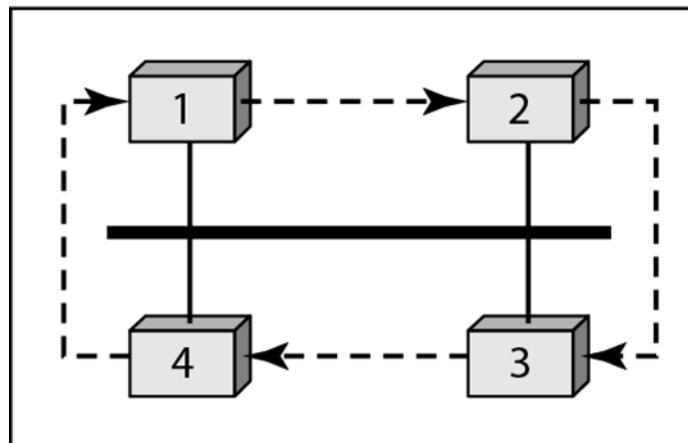
# Token Passing



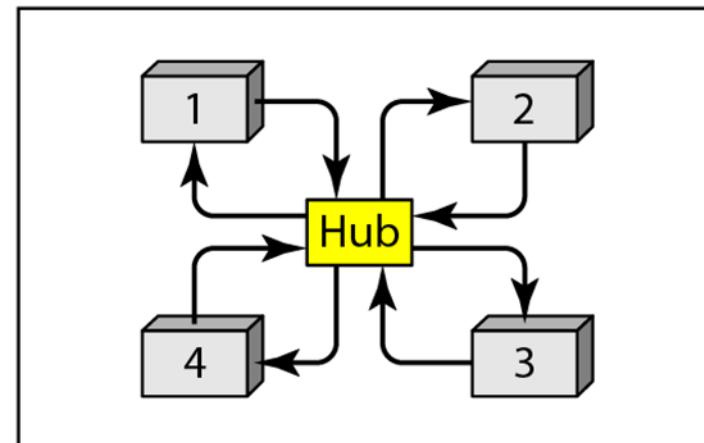
a. Physical ring



b. Dual ring



c. Bus ring

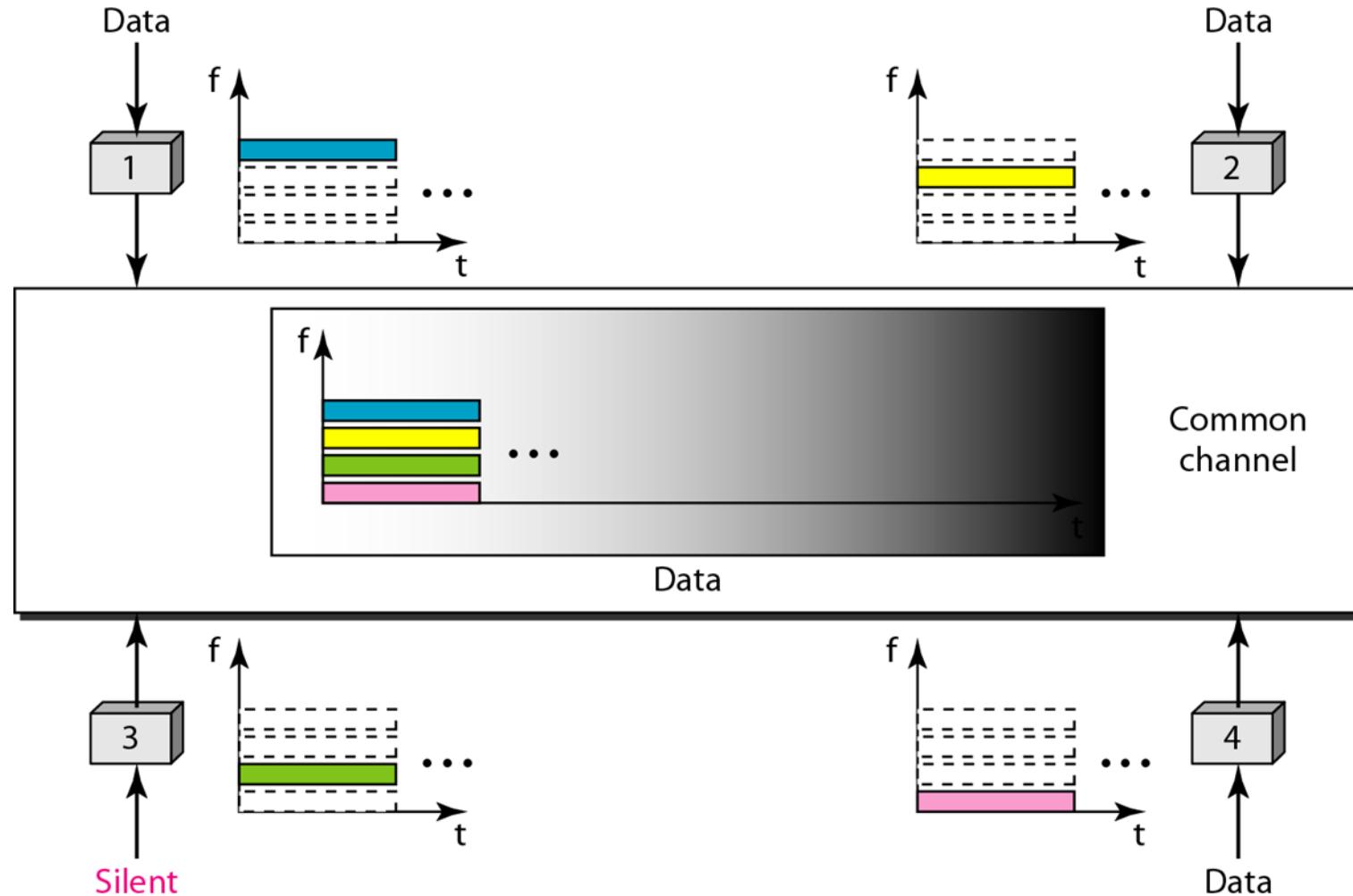


d. Star ring

# Channelization

- Similar to **multiplexing**
- Three schemes
  - Frequency-Division Multiple Access (FDMA)
  - Time-Division Multiple Access (TDMA)
  - Code-Division Multiple Access (CDMA)

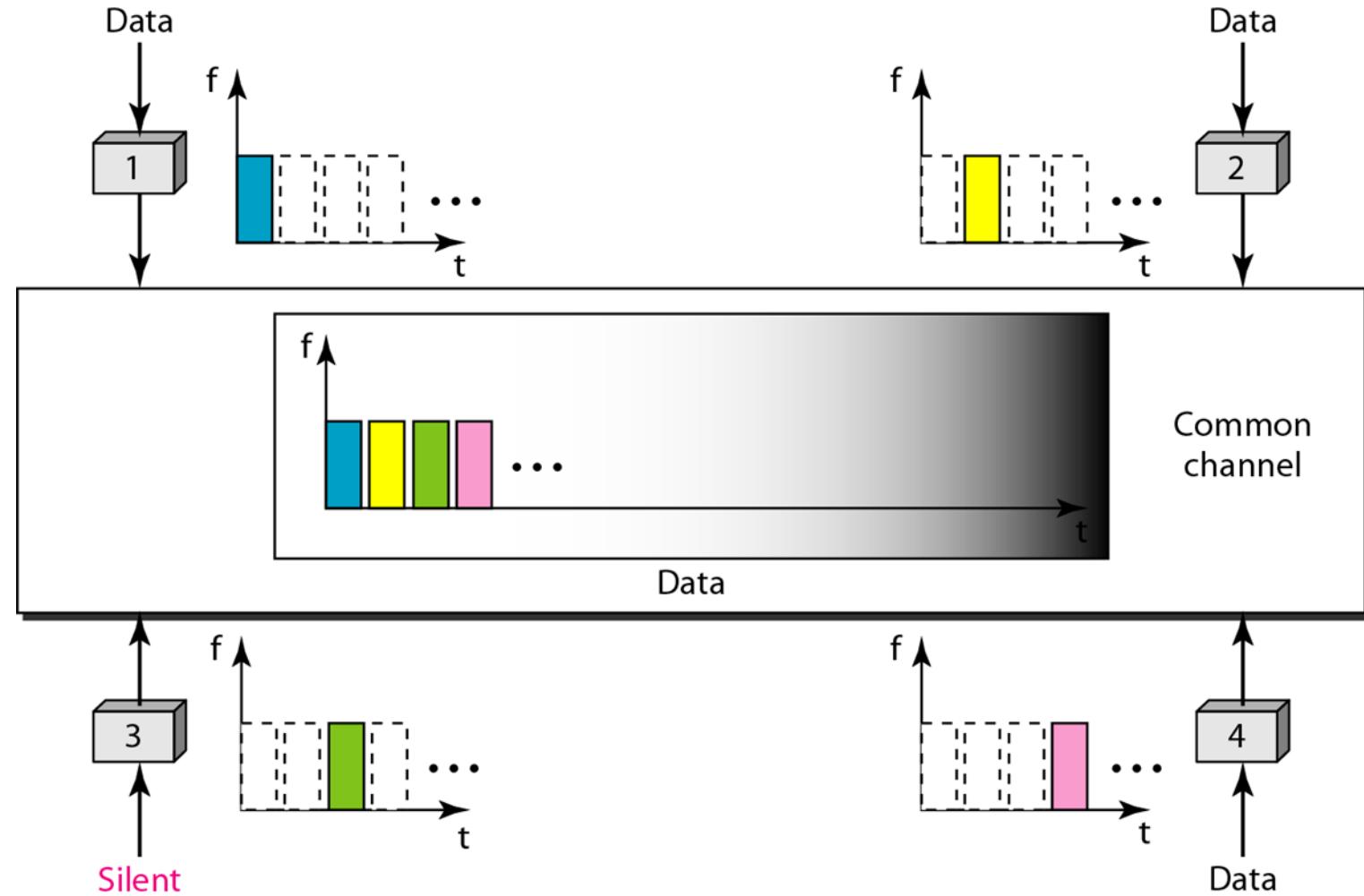
# FDMA



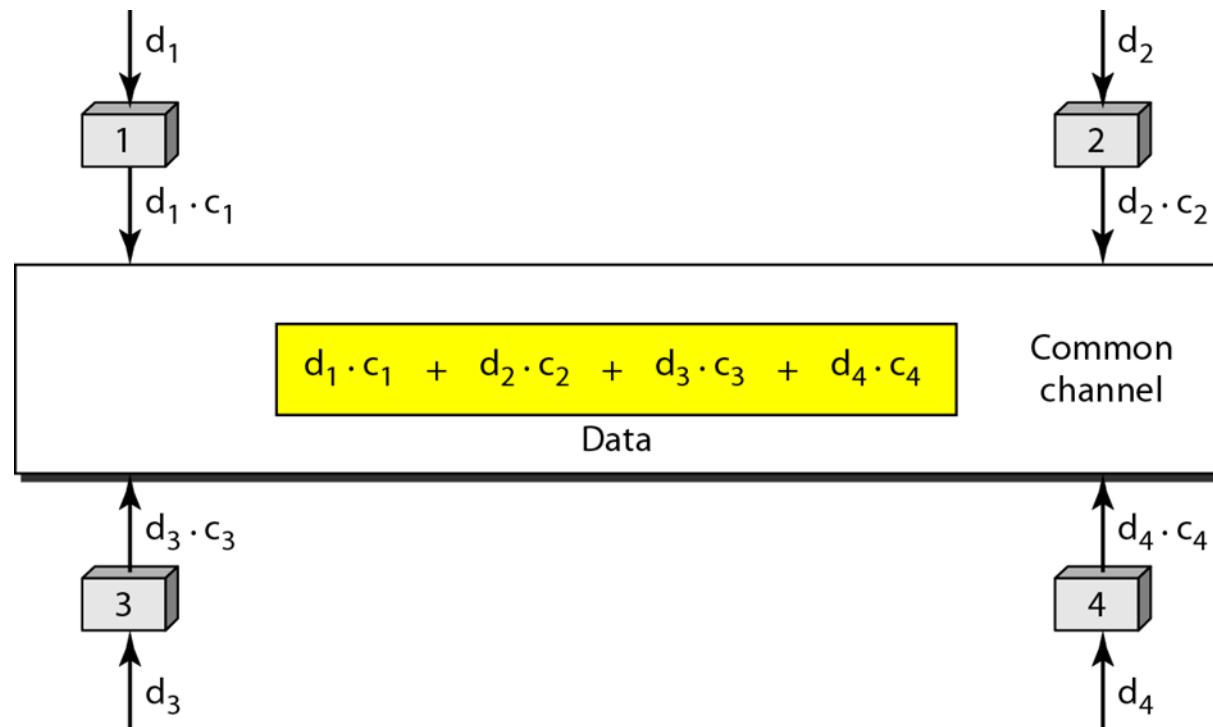
# FDMA and FDM

- FDM is a physical layer multiplexing technique, while FDMA is a data link layer access method.
- Using FDM to allow multiple users to utilize the same bandwidth is called FDMA.
- FDM uses a physical multiplexer, while FDMA does not.

# TDMA

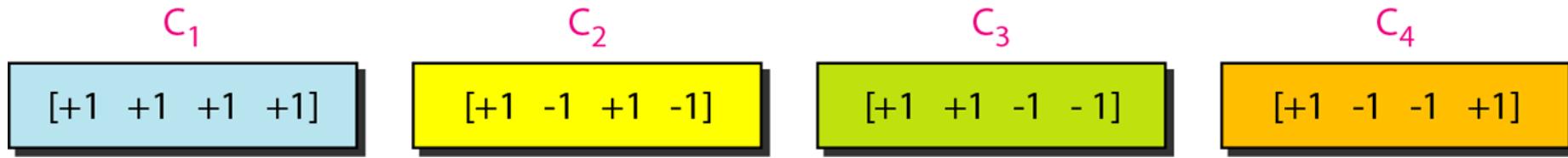


- One channel carries all transmissions at the same time
- Each channel is separated by code.



# CDMA: Chip Sequences

- Each station is assigned a **unique chip sequence**



- Chip sequences are *orthogonal vectors*
  - Inner product of any pair must be zero
- With N stations, sequences must have the following properties:
  - They are of length N
  - Their self inner product is always N

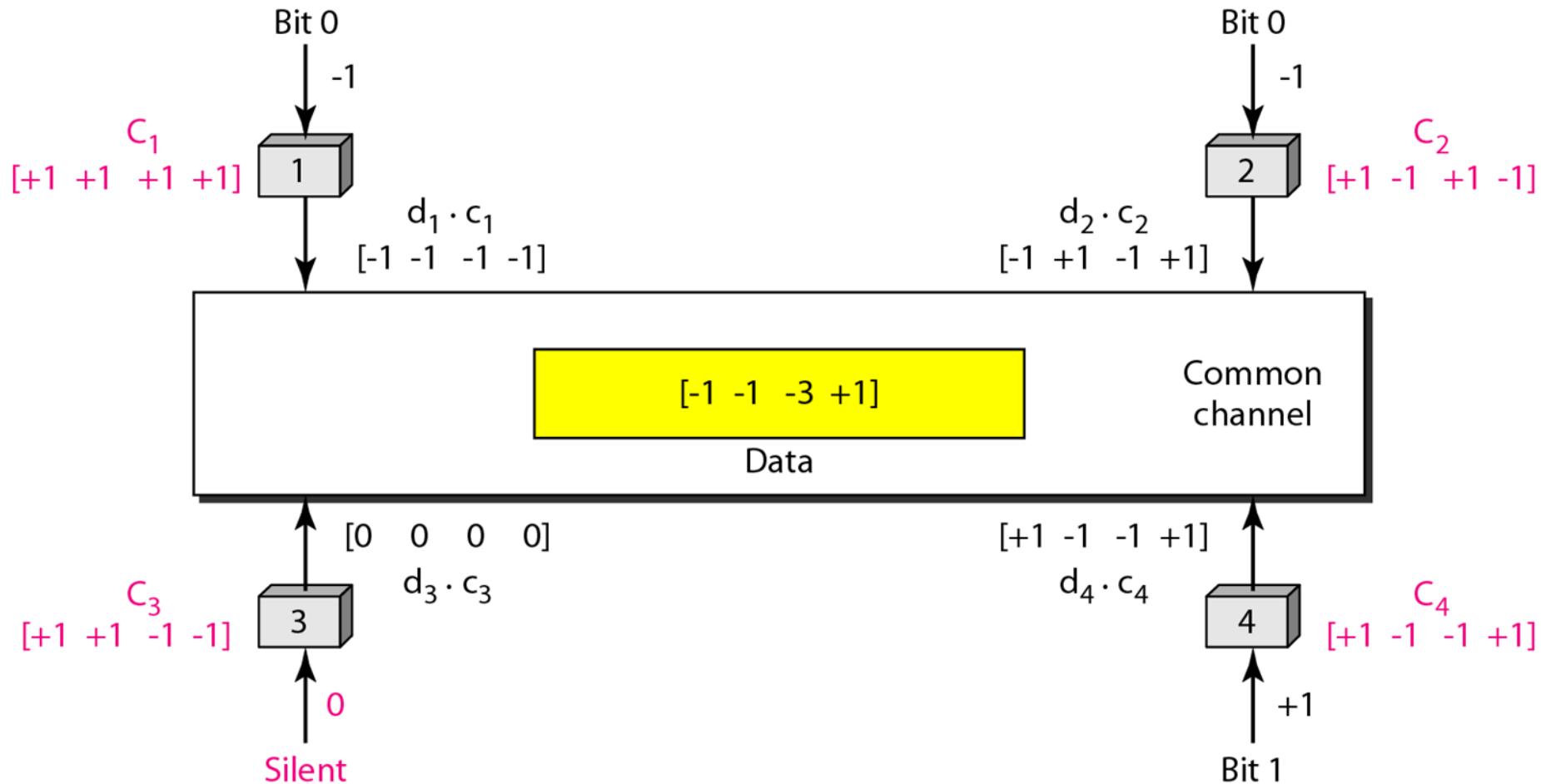
# CDMA: Bit Representation

Data bit 0 → -1

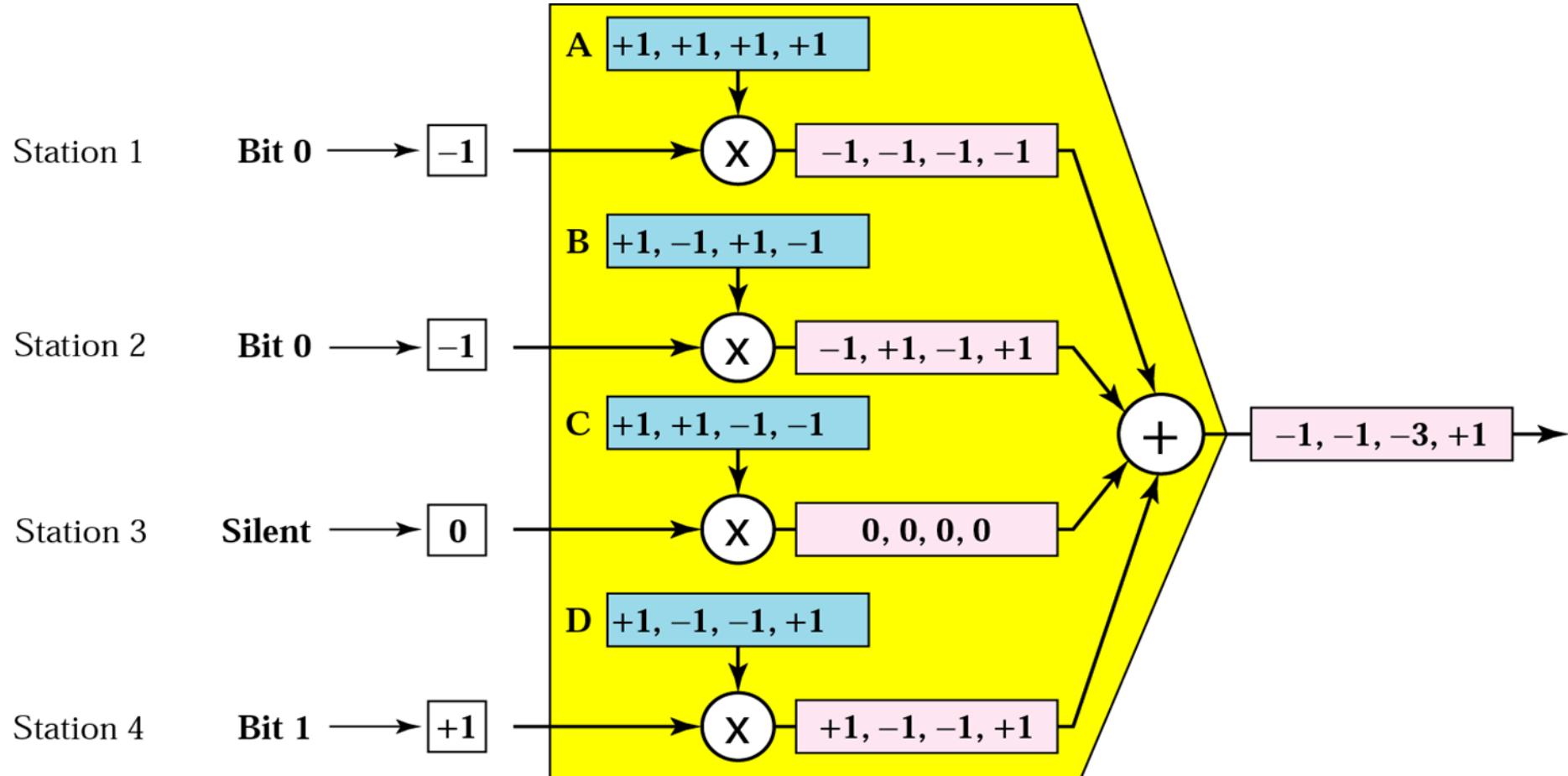
Data bit 1 → +1

Silence → 0

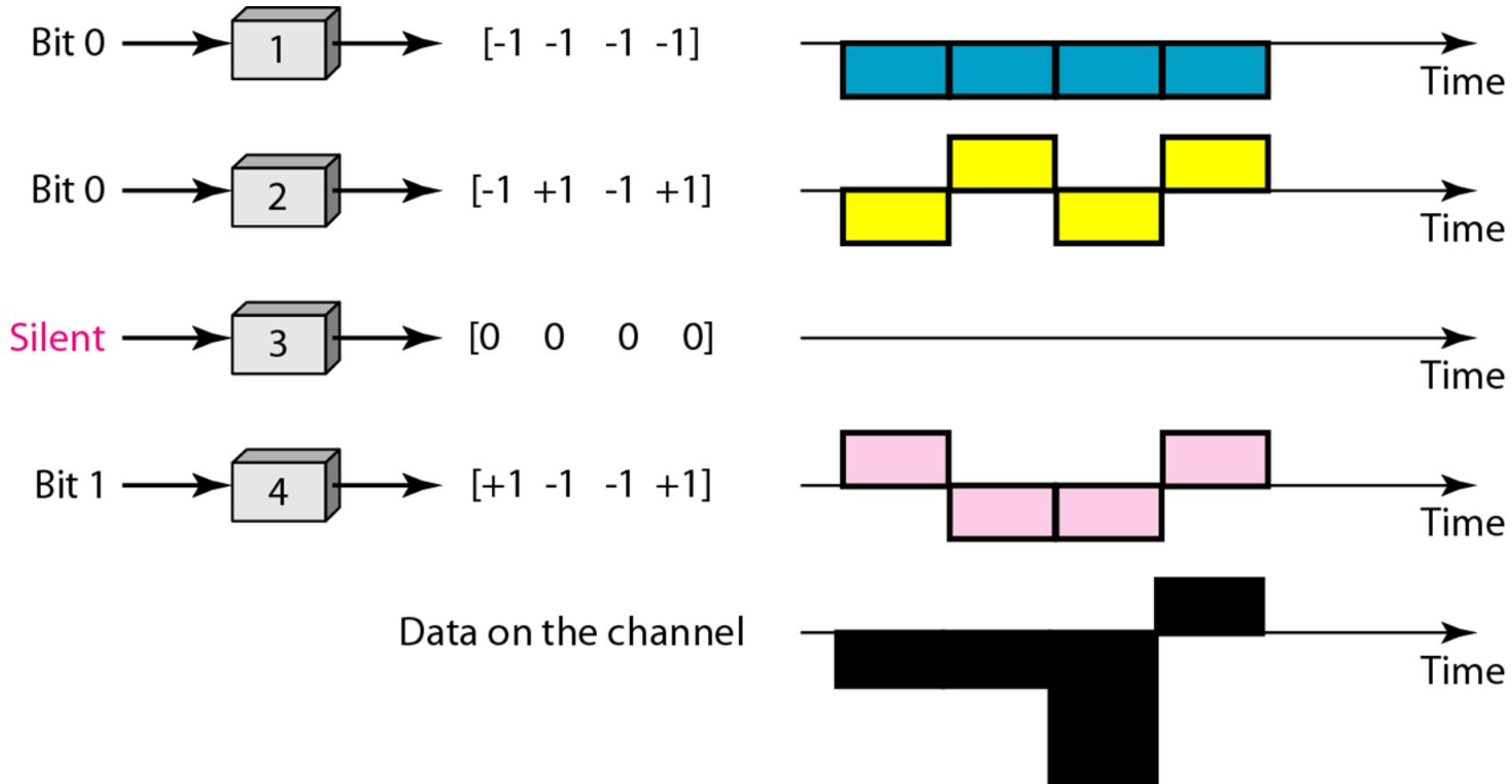
# Transmission in CDMA



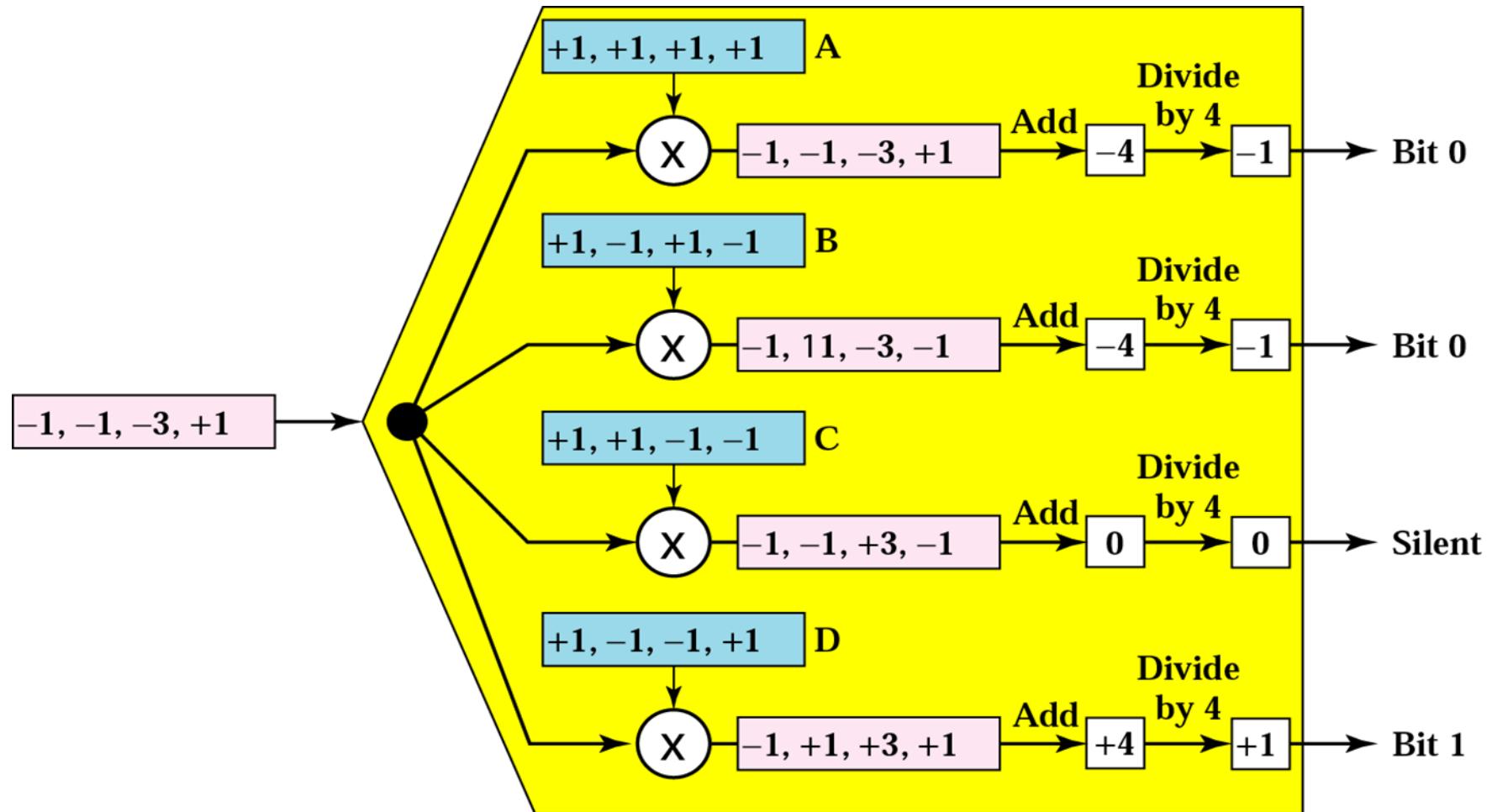
# CDMA Encoding



# Signal Created by CDMA



# CDMA Decoding



# Sequence Generation

- Common method: *Walsh Table*
  - Number of sequences is always a power of two

$$W_1 = \begin{bmatrix} +1 \end{bmatrix}$$

$$W_{2N} = \begin{bmatrix} W_N & W_N \\ W_N & \overline{W_N} \end{bmatrix}$$

a. Two basic rules

$$W_1 = \begin{bmatrix} +1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

b. Generation of  $W_1$ ,  $W_2$ , and  $W_4$

## Example: Walsh Table

1. Find chip sequences for eight stations
  
  
  
  
  
  
  
  
2. There are 80 stations in a CDMA network. What is the length of the sequences generated by Walsh Table?

# Wired LAN: Ethernet

Module 2

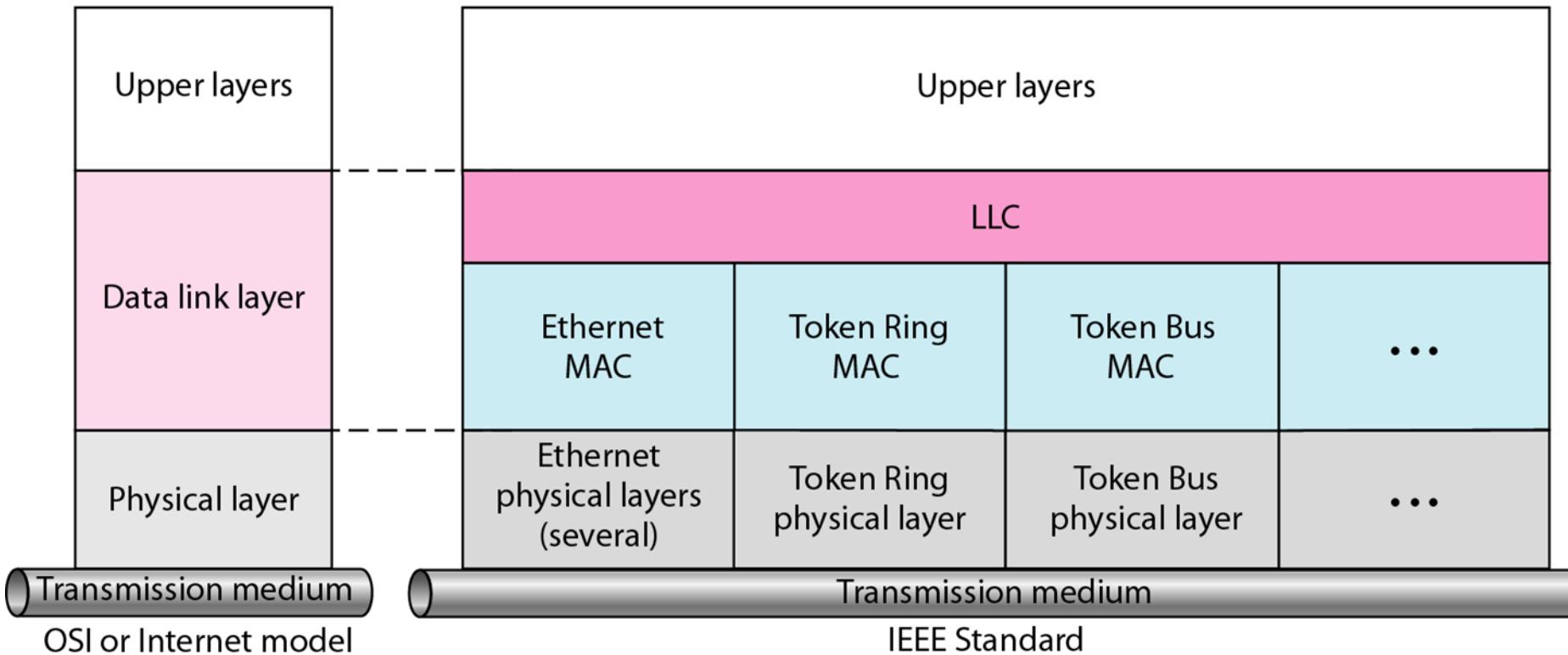
# IEEE Standards

- In 1985, the Computer Society of the IEEE started a project, called *Project 802*, to set standards to enable intercommunication among equipment from a variety of manufacturers.
- *Project 802* is a way of specifying functions of the physical layer and the data link layer of major LAN protocols.

# IEEE standard for LANs

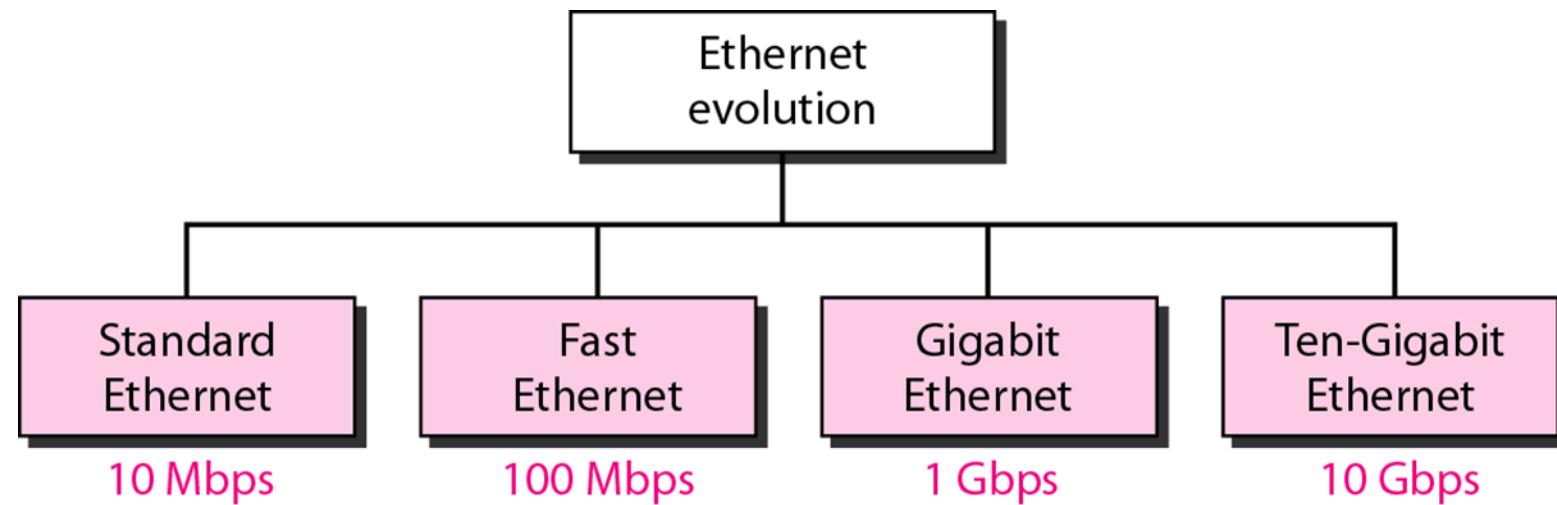
LLC: Logical link control

MAC: Media access control



# STANDARD ETHERNET

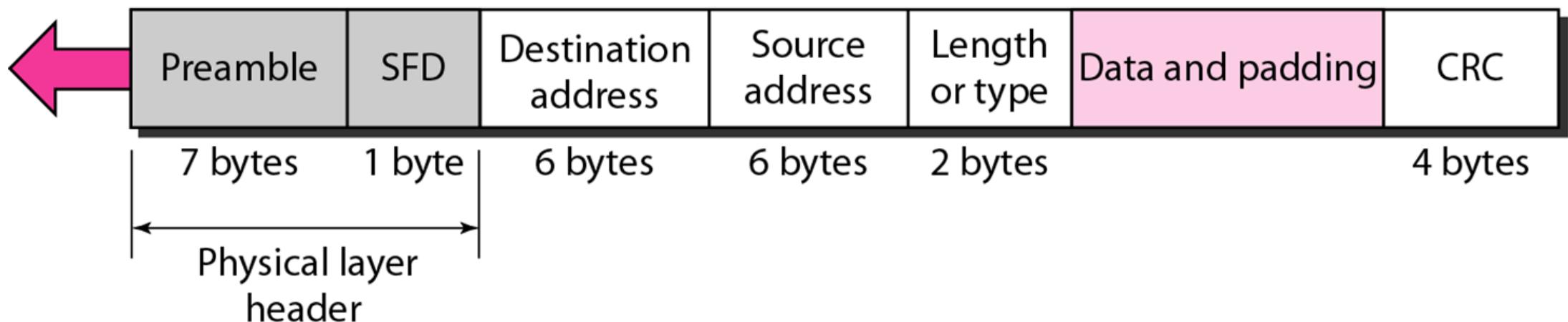
*Ethernet evolution through four generations*



# 802.3 MAC frame

**Preamble:** 56 bits of alternating 1s and 0s.

**SFD:** Start frame delimiter, flag (10101011)



## 802.3 MAC frame

- The **Preamble**
  - consists of seven bytes all of the form 10101010,
  - used by the receiver to allow it to establish bit synchronization.
- The **Start frame delimiter**
  - is a single byte, 10101011, which is a frame flag, indicating the start of a frame.
  - It also signifies that the Destination MAC Address field begins with the next byte.
- **Destination MAC Address**
- **Source MAC Address**

## 802.3 MAC frame

- ***Length or Type***
  - defines the type of protocol inside the frame, for example IPv4 or IPv6, ARP, OSPF etc.
  - it indicates the number of bytes of data in the frame's payload and can be anything from 0 to 1500 bytes.
  - Frames must be at least 64 bytes long, not including the preamble, so, if the data field is shorter than 46 bytes, it must be compensated by the ***Padding*** field.

## 802.3 MAC frame

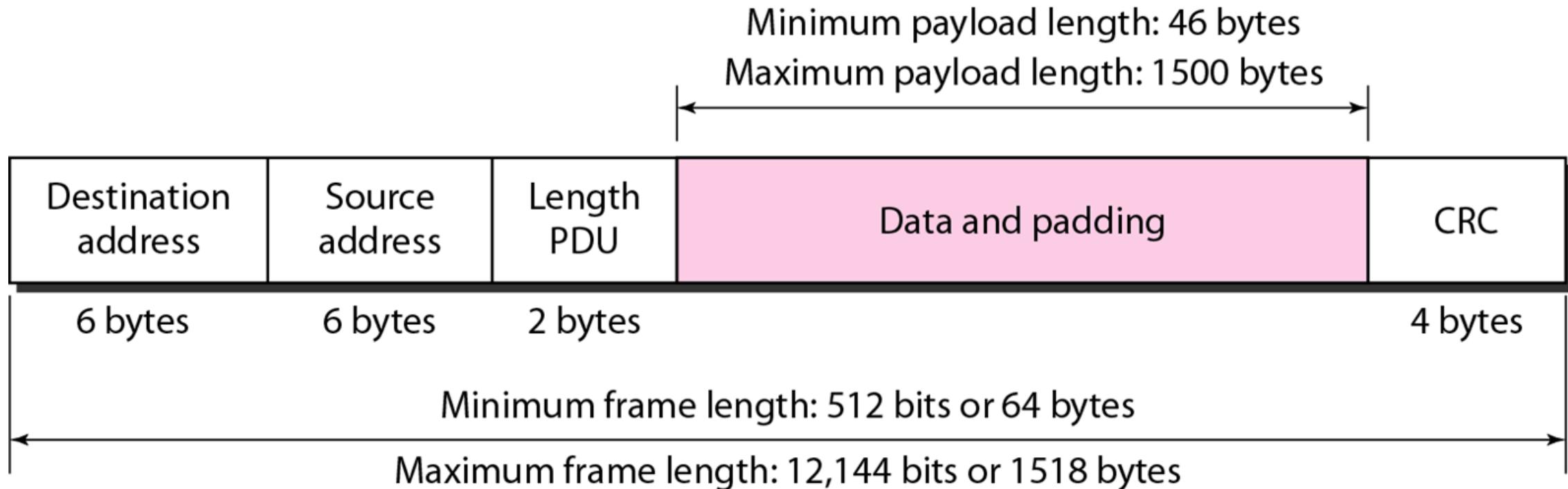
- ***Data***
  - Field carries data encapsulated from upper-layer protocols.
  - it is minimum of 46 bytes and maximum of 1500 bytes.
  - If data is more than 1500 bytes, then it should be fragmented and encapsulated in more than one frames.
  - If less than 46 bytes, it need to be padded with extra 0's

## 802.3 MAC frame

- **CRC**
  - Contains error detection information.
  - CRC is calculated over addresses, types and data field.
  - Receiver calculates CRC and finds that it is not zero, it discards the frame.

# 802.3 MAC frame

## *Minimum and maximum lengths*



### *Frame length:*

**Minimum: 64 bytes (512 bits)**

**Maximum: 1518 bytes (12,144 bits)**

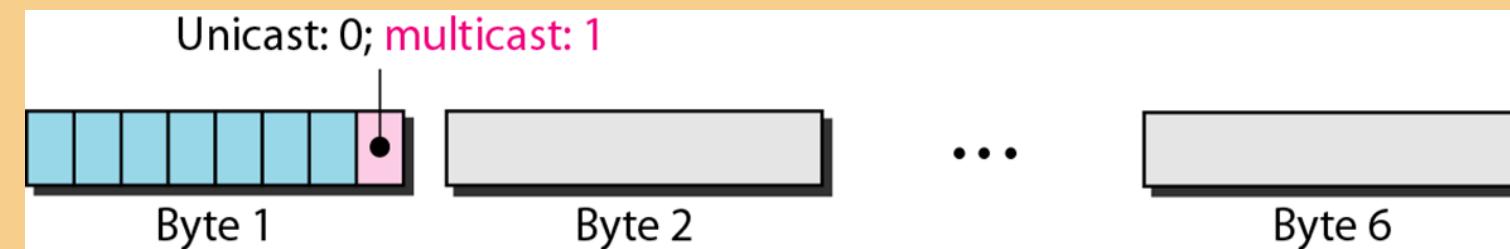
# Example of an Ethernet Address

*hexadecimal notation*

06 : 01 : 02 : 01 : 2C : 4B

6 bytes = 12 hex digits = 48 bits

## *Unicast and multicast addresses*



# Ethernet Address

- The least significant bit of the first byte defines the type of address. If the bit is 0, the address is unicast; otherwise, it is multicast.
- The broadcast destination address is a special case of the multicast address in which all bits are 1s.

## Example: Ethernet Address

Define the type of the following destination addresses:

- a. 4A:30:10:21:10:1A
- b. 47:20:1B:2E:08:EE
- c. FF:FF:FF:FF:FF:FF

### *Solution:*

To find the type of the address, we need to look at the second hexadecimal digit from the left. If it is even, the address is unicast. If it is odd, the address is multicast. If all digits are F's, the address is broadcast. Therefore, we have the following:

- a. This is a unicast address because A in binary is 1010.
- b. This is a multicast address because 7 in binary is 0111.
- c. This is a broadcast address because all digits are F's.

# Categories of Standard Ethernet

