Module 2: Data Link and MAC Layer

Part-I Error Control

Dr. Prasanna Shete
Dept. of Computer Engineering
K. J. Somaiya College of Engineering

Slide Source: B. A. Forauzan, Data Communications and Networking, McGraw-Hill Online Learning Centre
http://highered.mheducation.com/sites/0072967757/information_center_view0/index.html

Note

Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

10-1 INTRODUCTION

Topics discussed in this section:

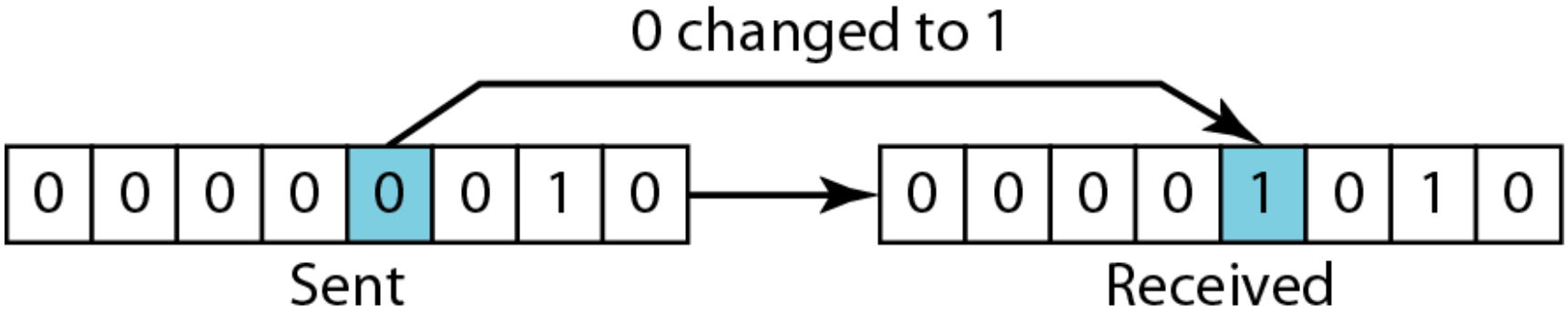
Types of Errors
Redundancy
Detection Versus Correction
Forward Error Correction Versus Retransmission
Coding
Modular Arithmetic

Note

Single-bit error:

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

Figure 10.1 *Single-bit error*

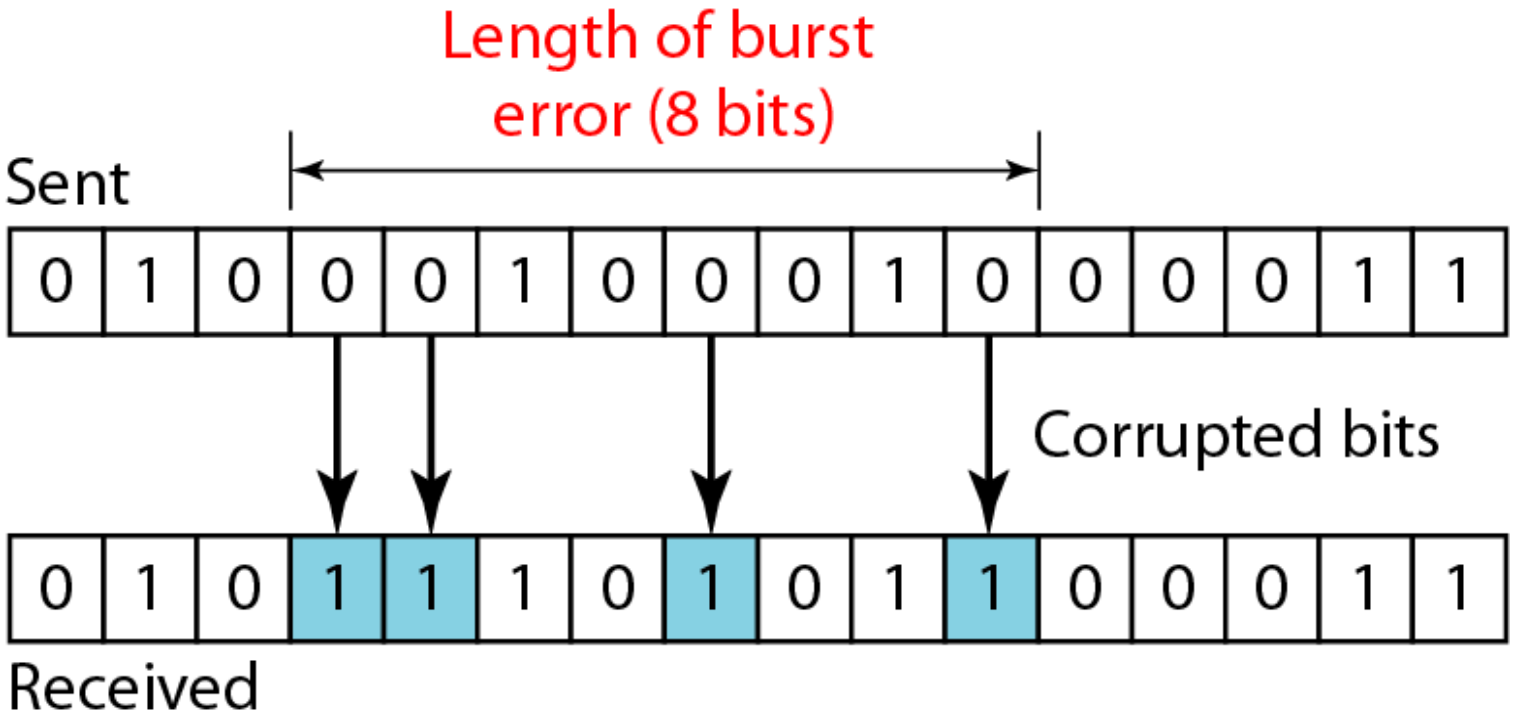


Note

Burst Error:

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

Figure 10.2 *Burst error of length 8*

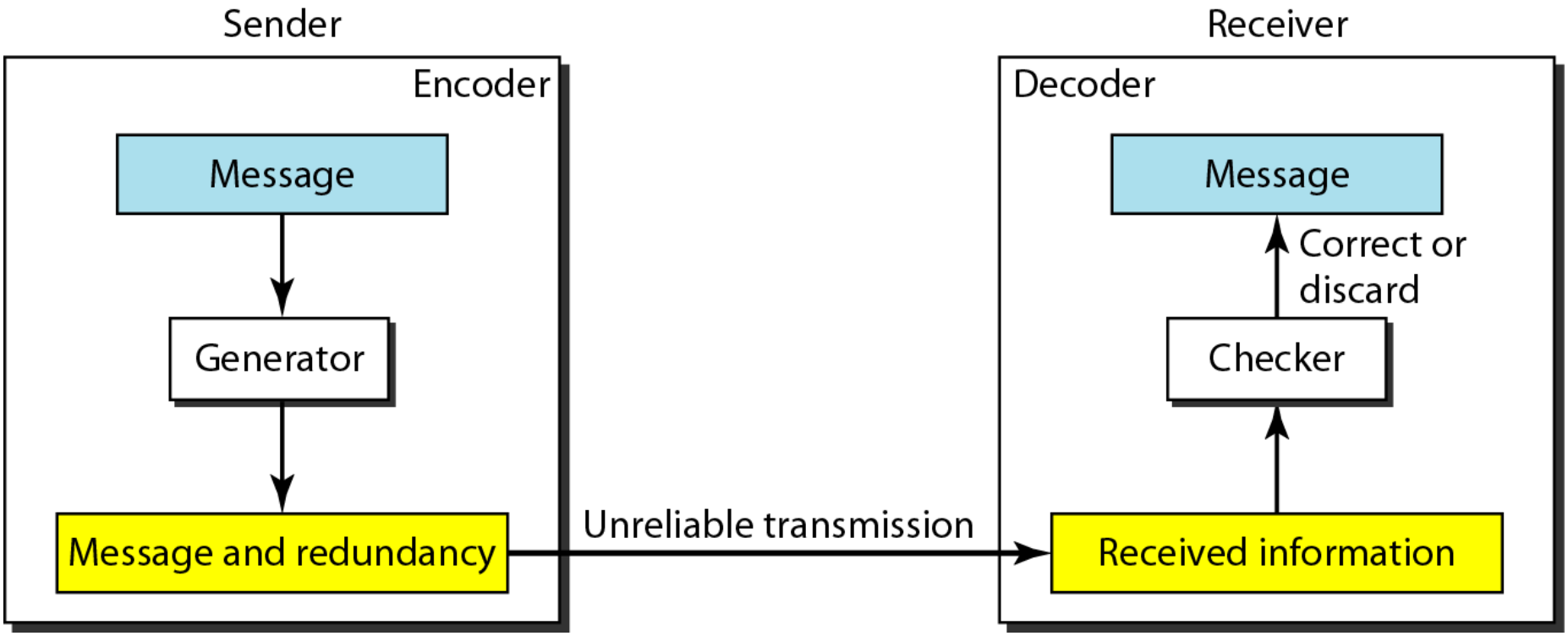


Note

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

Sender/generator creates Relationship between the redundant bits and the actual bits

Figure 10.3 *The structure of encoder and decoder*



Note

Codes→ Block codes and Convolution Codes

In this book, we concentrate on block codes; we leave convolution codes to advanced texts.

Note

Modular Arithmetic

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

Modulo-2 Arithmetic

Modulus N=2; 0 and 1

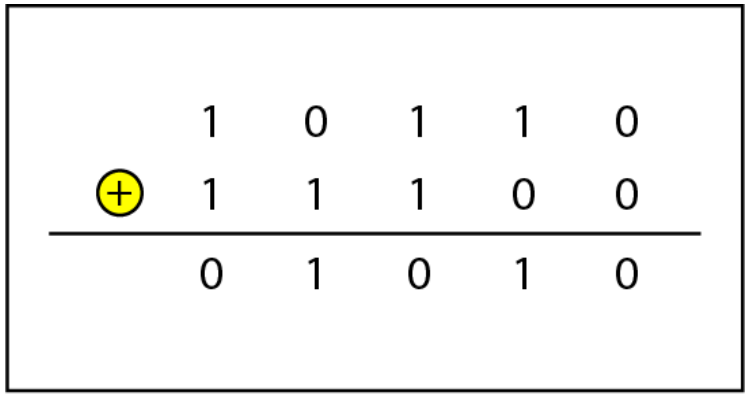
Addition : 0+0=0, 0+1=1, 1+0=1, 1+1=0

Subtraction: 0-0=0, 0-1=1, 1-0=1, 1-1=0

Figure 10.4 ***XORing of two single bits or two words***

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

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



c. Result of XORing two patterns

10-2 BLOCK CODING

In block coding, message is divided into blocks, each of k bits, called datawords.

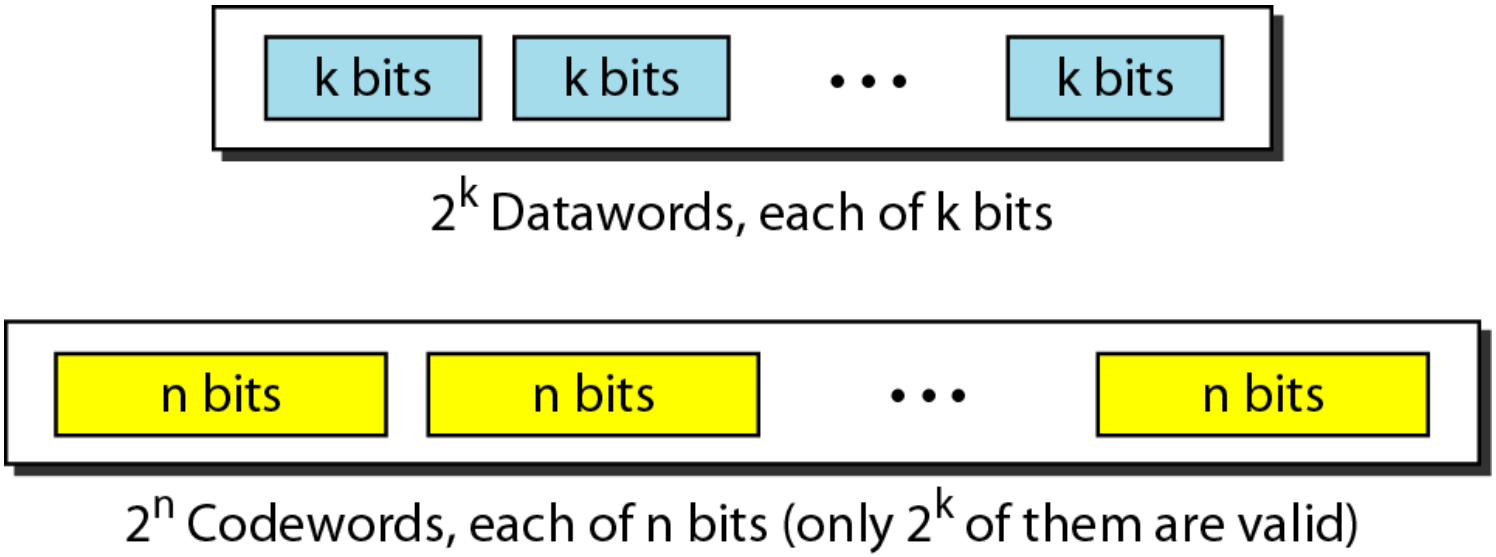
- r redundant bits are added to each block to make the length n = k + r

The resulting n-bit blocks are called codewords.

Topics discussed in this section:

Error Detection
Error Correction
Hamming Distance
Minimum Hamming Distance

Figure 10.5 *Datawords and codewords in block coding*



Example 10.1

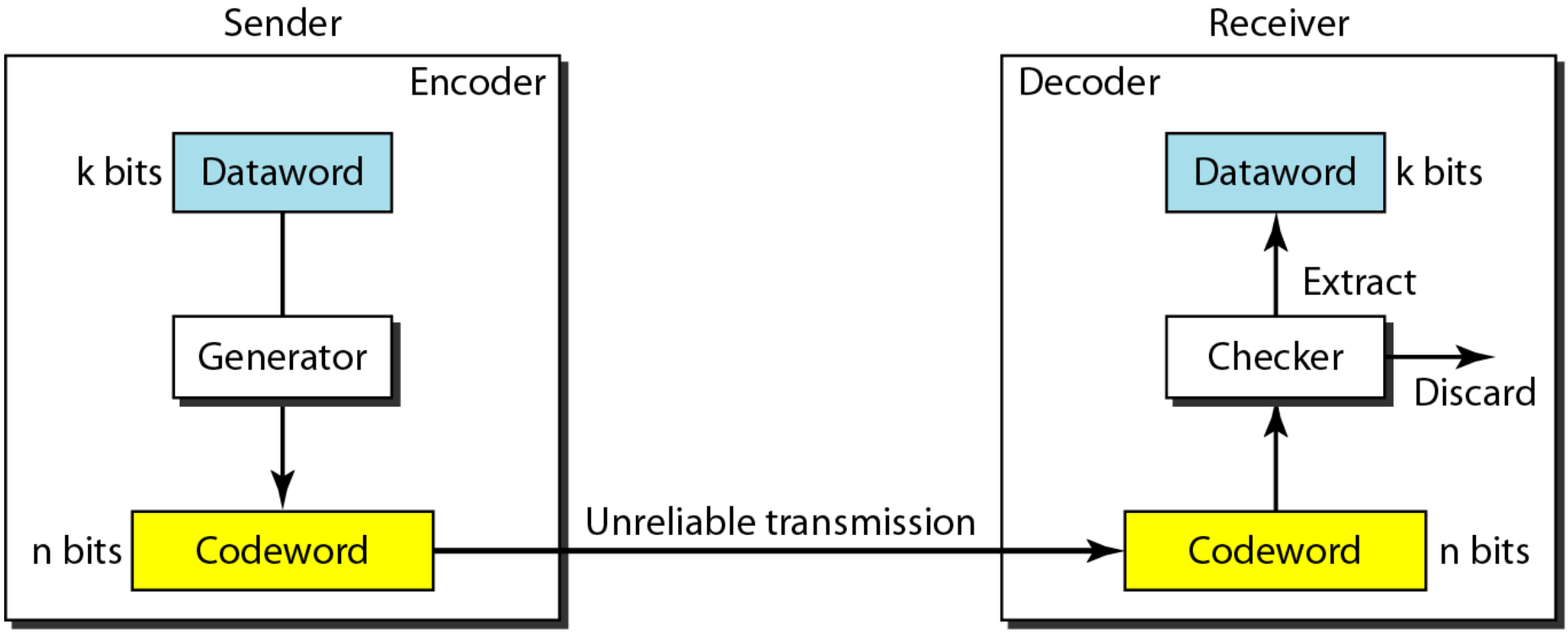
The 4B/5B block coding is a good example of this type of coding.

In this coding scheme, $k = 4$ and $n = 5$

- we have $2^k = 16$ datawords
- and $2^n = 32$ codewords

16 out of 32 codewords are used for message transfer and the rest are either unused or used for other purposes

Figure 10.6 *Process of error detection in block coding*



Example 10.2

Let us assume that $k = 2$ and $n = 3$. Table 10.1 shows the list of datawords and codewords.

Datawords	Codewords
00	000
01	011
10	101
11	110

Example 10.2

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.***

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.

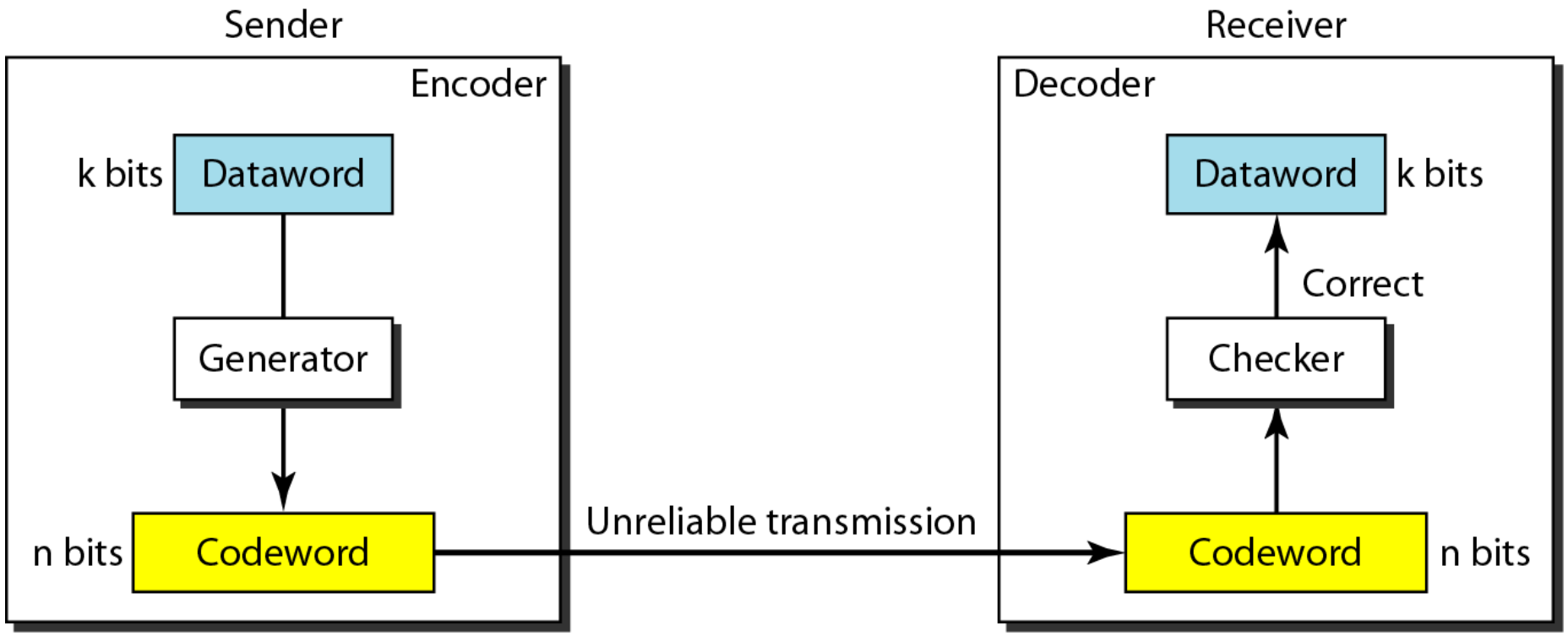
Table 10.1 ***A code for error detection (Example 10.2)***

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

Note

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

Figure 10.7 *Structure of encoder and decoder in error correction*



Example 10.3

Let us add more redundant bits to Example 10.2 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 10.2 shows the datawords and codewords.

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

- ***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 →means an error has occurred.***
- ***The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword:***

Example 10.3 (continued)

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.*

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

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

Hamming Distance

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

The Hamming Distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission

Example 10.4: finding hamming distance

Let us find the Hamming distance between two pairs of words.

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

$000 \oplus 011$ is 011 (two 1s)

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.

Example 10.5

Find the minimum Hamming distance of the coding scheme in Table 10.1.

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.

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

Example 10.6

Find the minimum Hamming distance of the coding scheme in Table 10.2.

Dataword	Codeword
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.

Note

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

Example 10.7

The minimum Hamming distance for our first code scheme (Table 10.1) is 2. This code guarantees detection of only a single error.

For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword.

If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

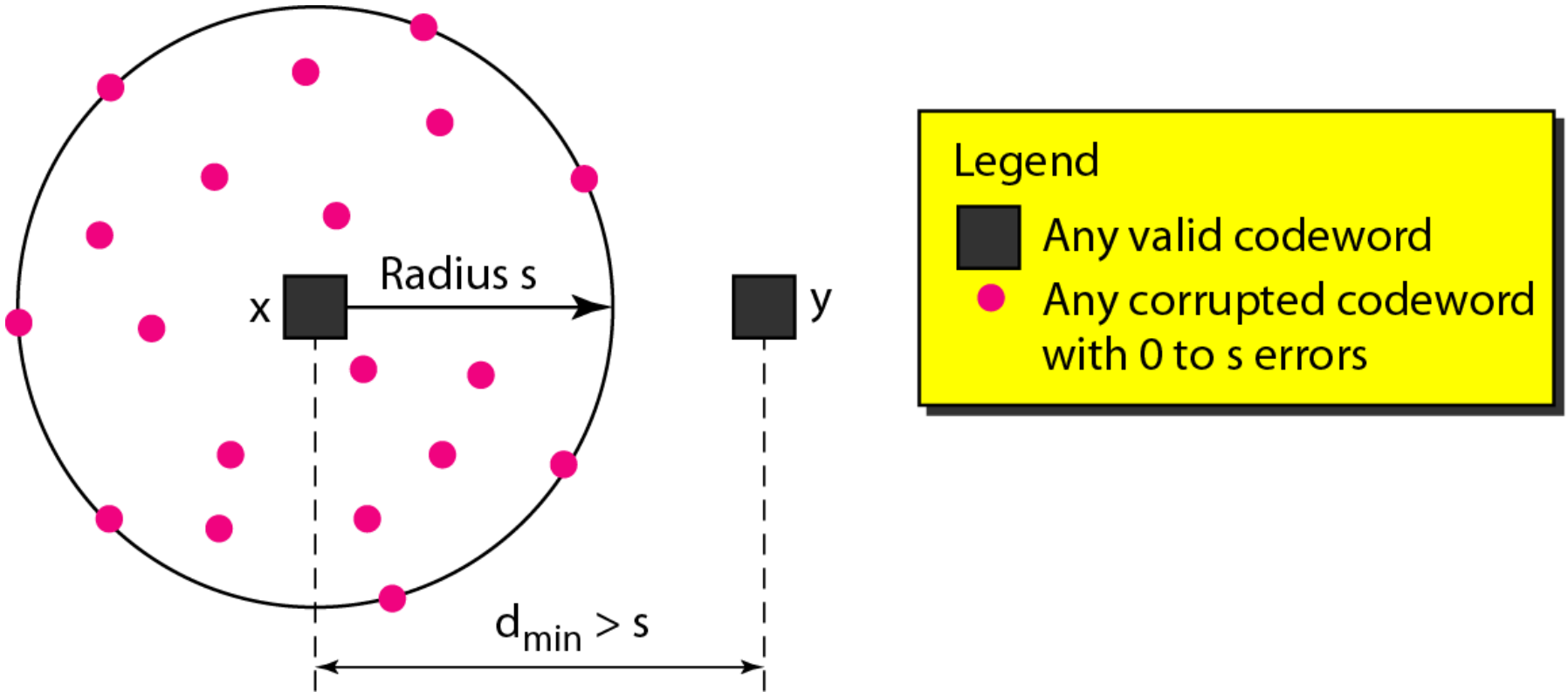
Example 10.8

Our second block code scheme (Table 10.2) has $d_{min} = 3$. This code can detect up to two errors.

We see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.

However, some combinations of three errors change a valid codeword to another valid codeword. The receiver accepts the received codeword and the errors are undetected.

Figure 10.8 *Geometric concept for finding d_{min} in error detection*



Note

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 10.9

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, . . .).

10-3 LINEAR BLOCK CODES

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

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

Topics discussed in this section:

Minimum Distance for Linear Block Codes
Some Linear Block Codes

Note

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

Example 10.10

Let us see if the two codes we defined in Table 10.1 and Table 10.2 belong to the class of linear block codes.

1. *The scheme in Table 10.1 is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword. For example, the XORing of the second and third codewords creates the fourth one.*

2. *The scheme in Table 10.2 is also a linear block code. We can create all four codewords by XORing two other codewords.*

Example 10.11

In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.

In the second code (Table 10.2), the numbers of 1s in the nonzero codewords are 3, 3, and 4. So we have $d_{min} = 3$.

Datawords	Codewords
00	000
01	011
10	101
11	110

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Simple Parity Check Code

- Here, the *k*-bit dataword is changed to an *n*-bit codeword where *n= k+1*
  - Extra bit is called the parity bit; selected to make the total number of 1s in the codeword even (or odd)

- Min. hamming distance $d_{min} = 2$ → it's a single bit-error detecting code
  - Cannot correct error

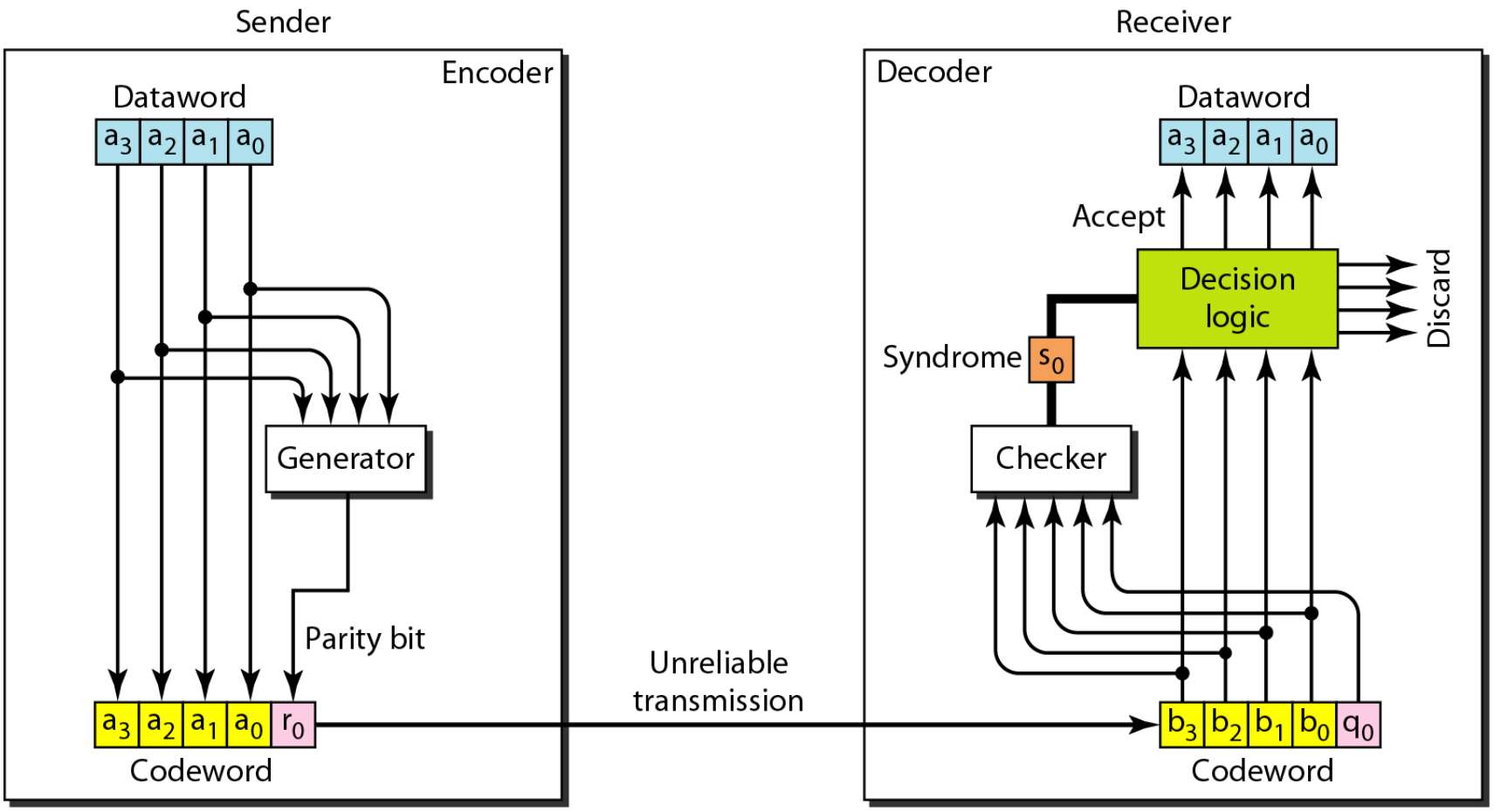
Note

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.

Table 10.3 ***Simple parity-check code C(5, 4)***

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

Figure 10.10 *Encoder and decoder for simple parity-check code*



Example 10.12

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 a_1. The received codeword is 10011. The syndrome is 1. No dataword is created.*
3. *One single-bit error changes r_0. The received codeword is 10110. The syndrome is 1. No dataword is created.*

Example 10.12 (continued)

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; here the dataword is wrongly created due to the syndrome value.

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.

→This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Note

A simple parity-check code can detect an odd number of errors.

Hamming Code

- Designed with $d_{min}=3$
  - Can detect 2 bit errors and correct one single error

- Relationship between n and k :
  - Choose integer $r >= 3$
  - the value of n and k are calculated as:

    $$n = 2^r - 1 \text{ and } k = n - r$$

  - $2^r >= r + k + 1$ (r-parity bits; for understanding you may take p instead of r)

  - → No. of redundant bits reqd. depends on no. of data bits and can be calculated as: $r >= \log_2(r + k + 1)$

Note

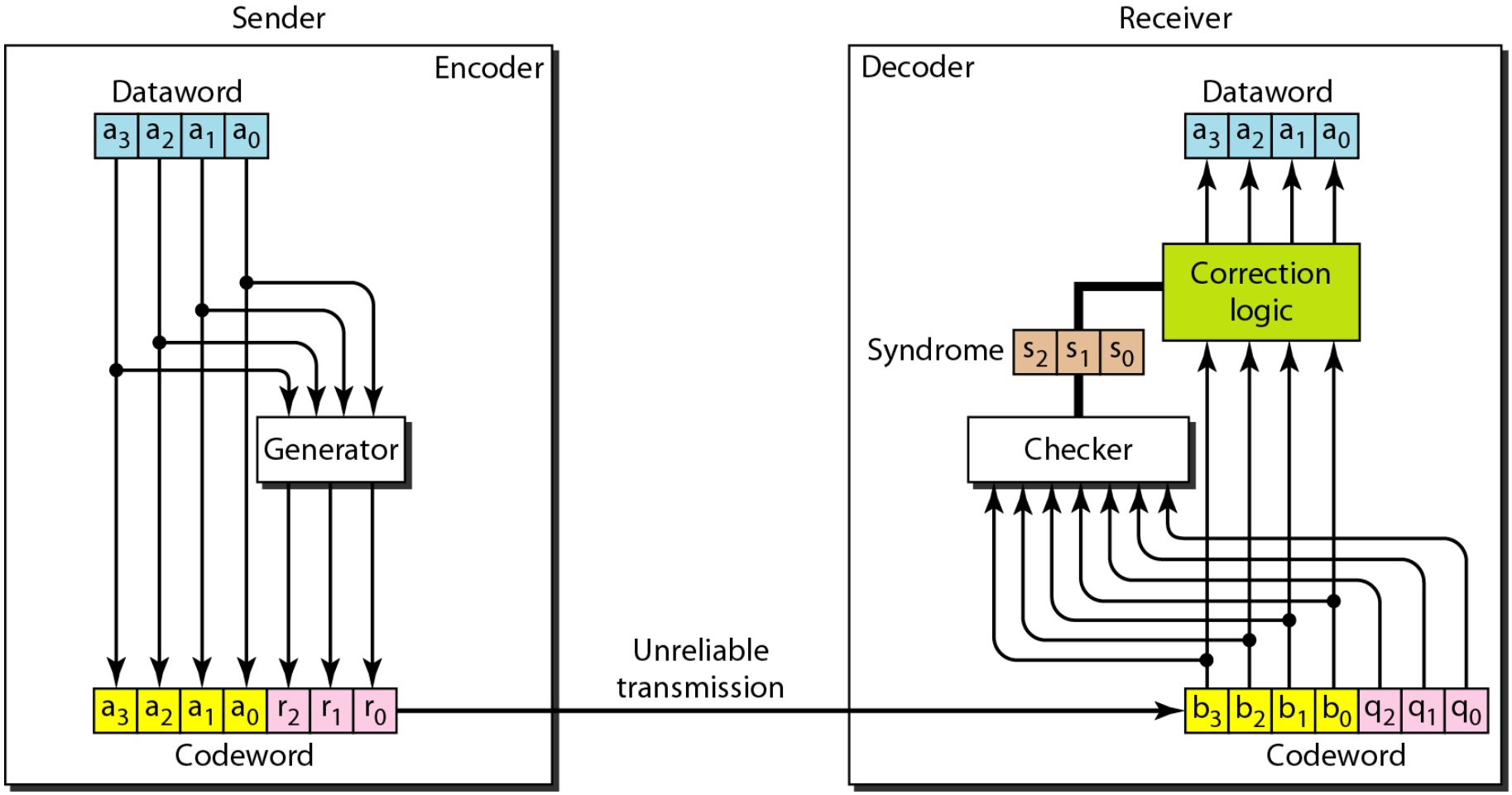
All Hamming codes discussed in this book have $d_{min} = 3$.

The relationship between r and n in these codes is $n = 2^r - 1$

Table 10.4 ***Hamming code C(7, 4)***

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

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



Hamming Code: Generator

- Parity check bits (r_0, r_1, r_2) calculation at Encoder
  - $r_0 = a_2 + a_1 + a_0$ (modulo 2)
  - $r_1 = a_3 + a_2 + a_1$
  - $r_2 = a_1 + a_0 + a_3$

- Syndrome bits calculation at Decoder
  - $s_0 = b_2 + b_1 + b_0 + q_0$ (modulo 2)
  - $s_1 = b_3 + b_2 + b_1 + q_1$
  - $s_2 = b_1 + b_0 + b_3 + q_2$

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

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

Example 10.13

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

1. *The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.*
2. *The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b_2 (changing the 1 to 0), the final dataword is 0111.*
3. *The dataword 1101 becomes the codeword 1101000. The syndrome is 101. After flipping b_0, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.*

Example 10.14

We need to send a dataword of atleast 7 bits. Calculate values of m and n .

Solution

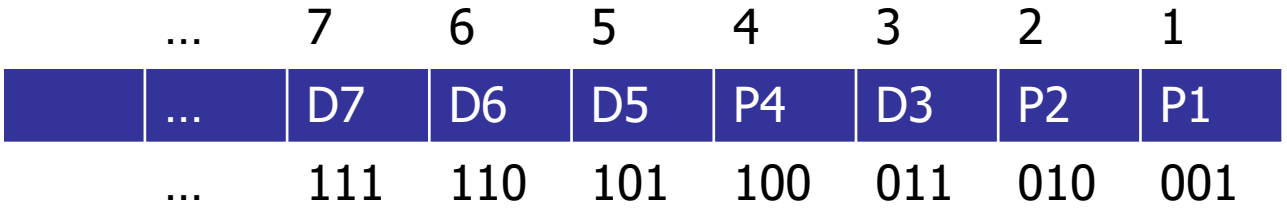
We need to make $k = n - m$ greater than or equal to 7, or $2^m - 1 - m \geq 7$. (since $2^m \geq m+k+1$)

1. *If we set $m = 3$, the result is $n = 2^3 - 1$ and $k = 7 - 3$, or 4, which is not acceptable.*
2. *If we set $m = 4$, then $n = 2^4 - 1 = 15$ and $k = 15 - 4 = 11$, which satisfies the condition. So the code is*

C(15, 11)

Structure of Hamming Code

- In Hamming code, position of parity check bits and data bits is predefined. The code structure is as follows:



- Parity check bits are placed at the bit position (index) that is power of 2 (have a single 1 bit in the binary form of their position)
- Data bits placed at other positions (two or more 1 bits in the binary form of their position)

Algorithm

- Steps to implement the algorithm at encoder:

1. Number the bits as1, 2, 3, 4, 5, 6, 7,…etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, 110, 111, etc.
3. All bit positions that are powers of two are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)
4. All other bit positions, are data bits.

1. Parity Bit: Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.
    1. Parity bit P1 covers all bit positions which have the **least** significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
    2. Parity bit P2 covers all bit positions which have the **second** least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
    3. Parity bit P4 covers all bit positions which have the **third** least significant bit set: bits 4–7, 12–15, 20–23, etc.
    4. Parity bit P8 covers all bit positions which have the **fourth** least significant bit set: bits 8–15, 24–31, 40–47, etc.

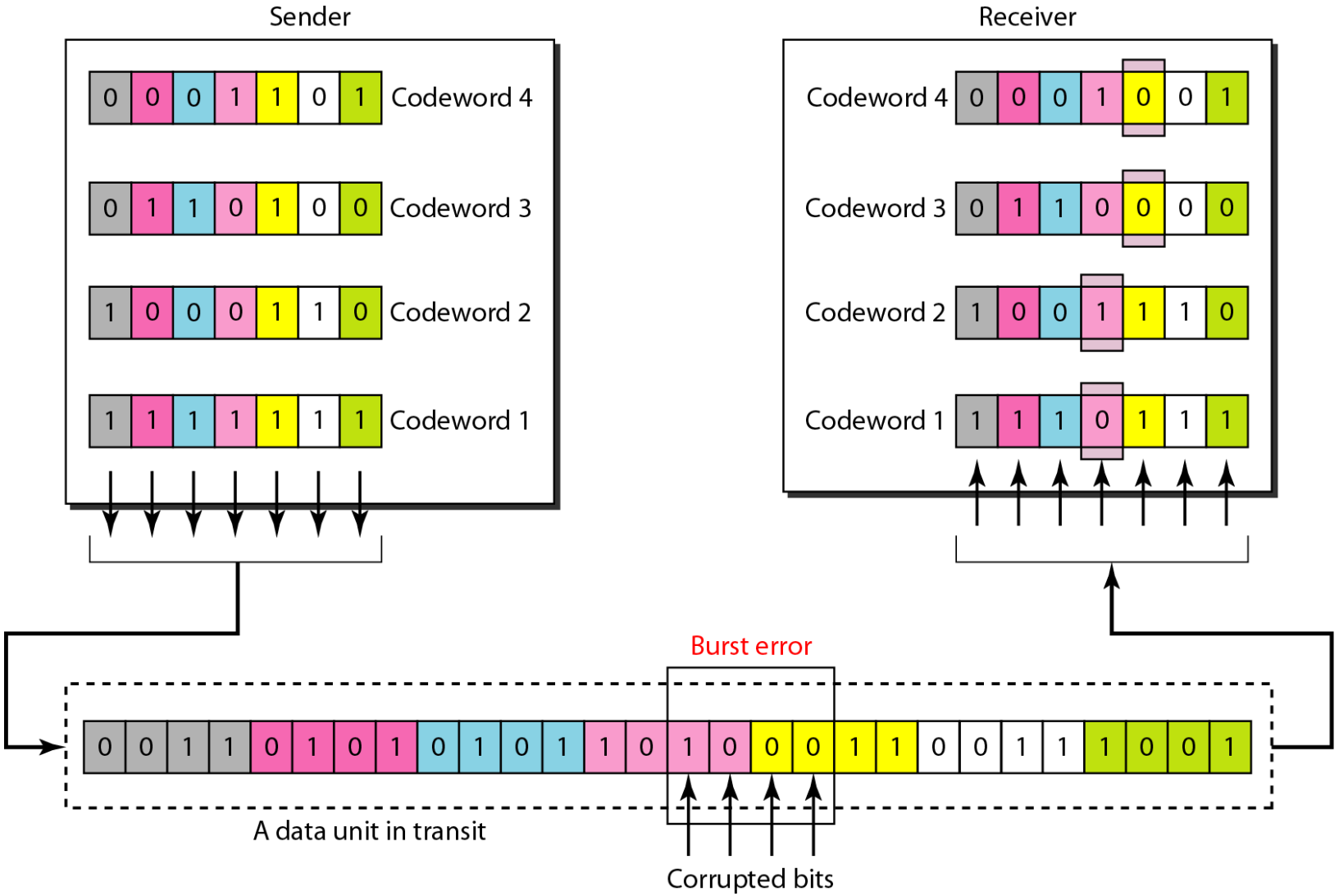
    In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

- **Error detection algorithm at decoder:**

- The receiver finds the syndrome by Ex-ORing data bits along with respective parity bits
- If the syndrome bit is all zeroes→ No error
- Else there is an error; the syndrome value shows which bit position is in an error

- [E.g in (7, 4) Hamming code, if s= 110→Sixth bit (D6) is in error]

Figure 10.13 *Burst error correction using Hamming code*



10-4 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.

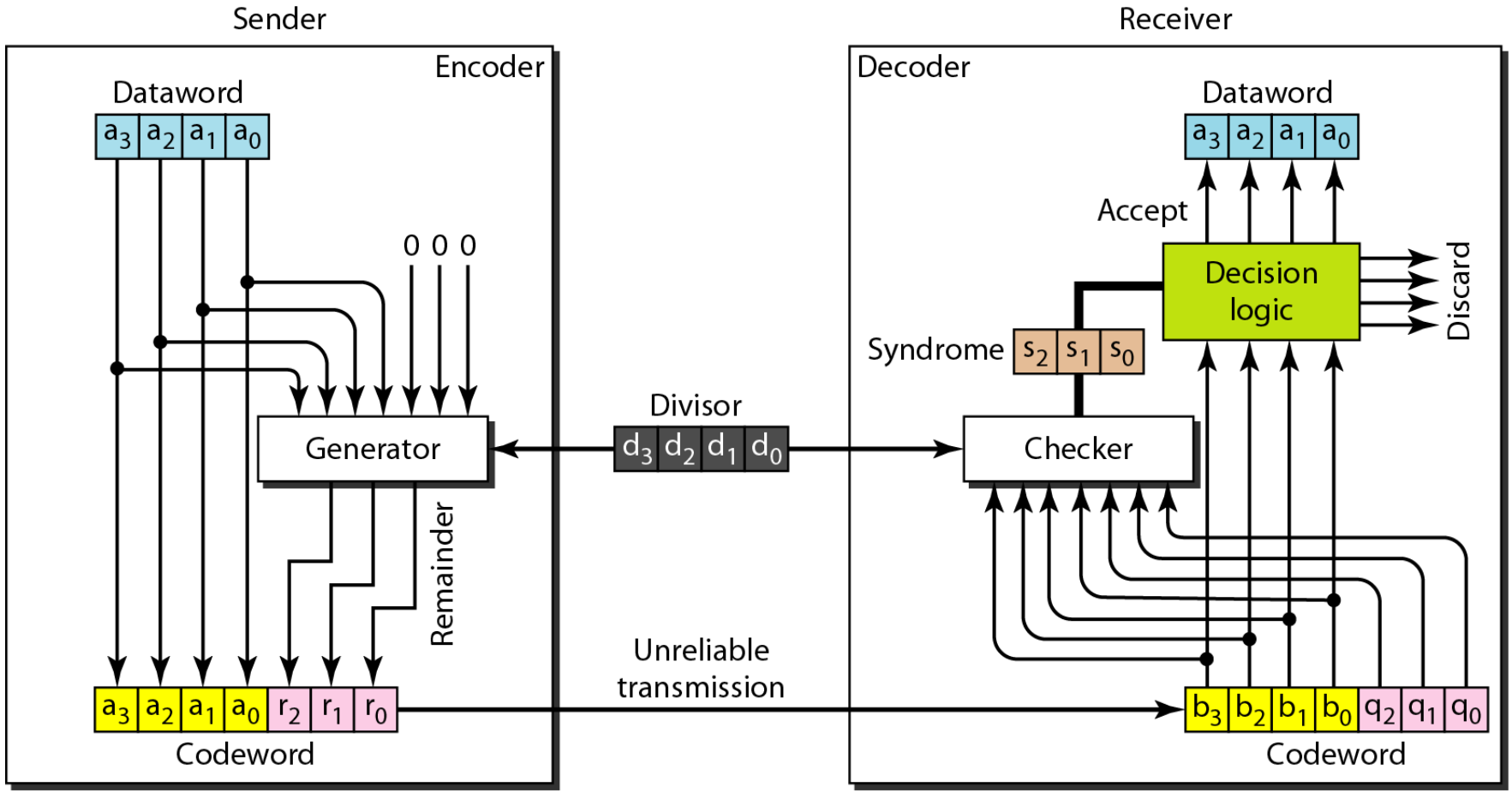
Topics discussed in this section:

Cyclic Redundancy Check
Hardware Implementation
Polynomials
Cyclic Code Analysis
Advantages of Cyclic Codes
Other Cyclic Codes

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

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

Figure 10.14 *CRC encoder and decoder*



Algorithm

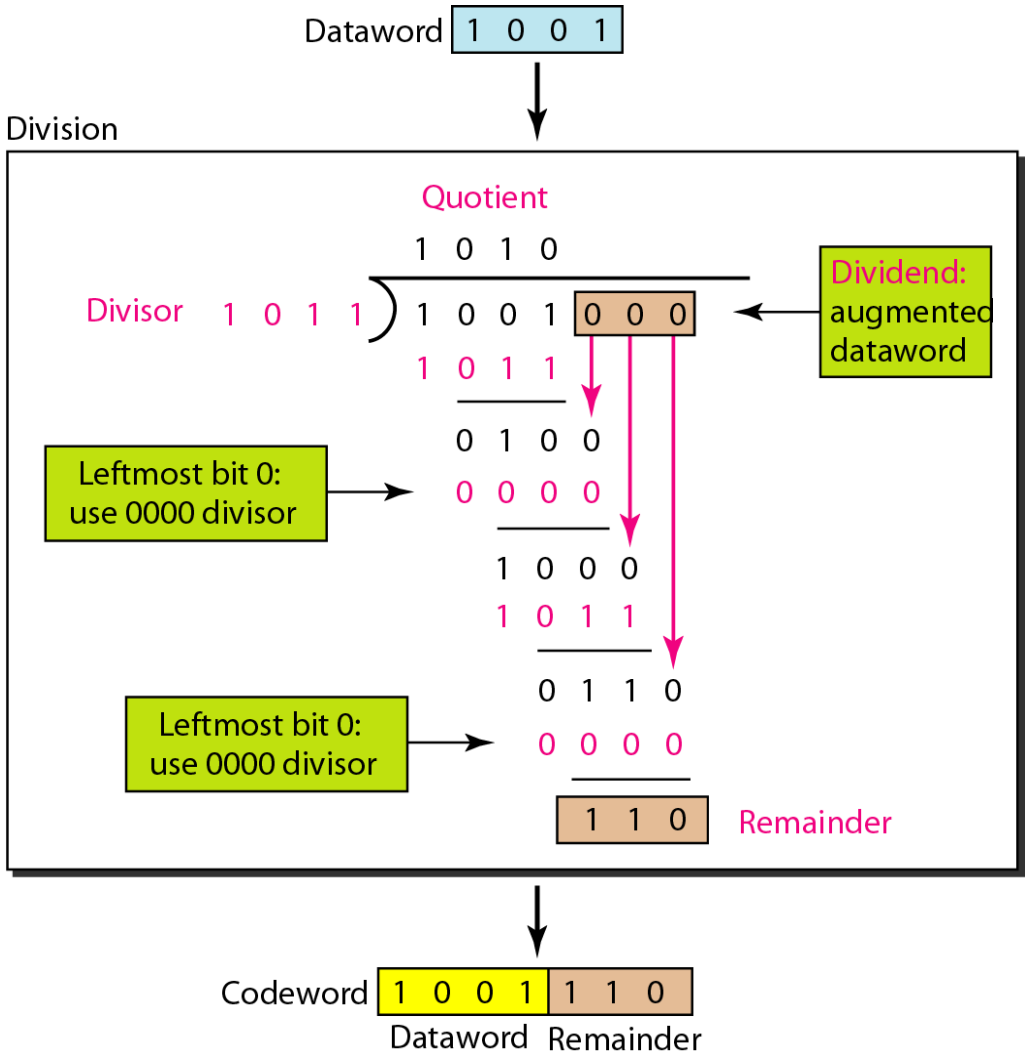
- Steps to implement the algorithm at encoder:

Dataword (k) = 4 bits; Codeword (n) = 7 bits

1. Augment Dataword *k* with *n-k* (= 3), 0's at the RHS
2. Feed n-bit result into the generator
3. Generator uses predefined and agreed upon divisor of size *n-k+1* (= 4) bits
4. Generator divides Augmented Dataword by the divisor→ modulo 2 division
5. Quotient is discarded and the remainder is appended to the Dataword to create the Codeword

Ex. Dataword: 1001, Divisor: 1011

Figure 10.15 *Division in CRC encoder*



Algorithm

Decoder performs the same division process

1. Divides received Codeword by the divisor→ modulo 2 division
2. Quotient is discarded and the remainder is taken as Syndrome
3. If Syndrome is all 0's→ No Error;
4. the dataword is separated from the received codeword and accepted; discarded otherwise

Ex. Transmitted codeword: 1001110

Received codeword/s: 1001110, 1000110

Calculate the Syndrome values

Figure 10.16 *Division in the CRC decoder for two cases*

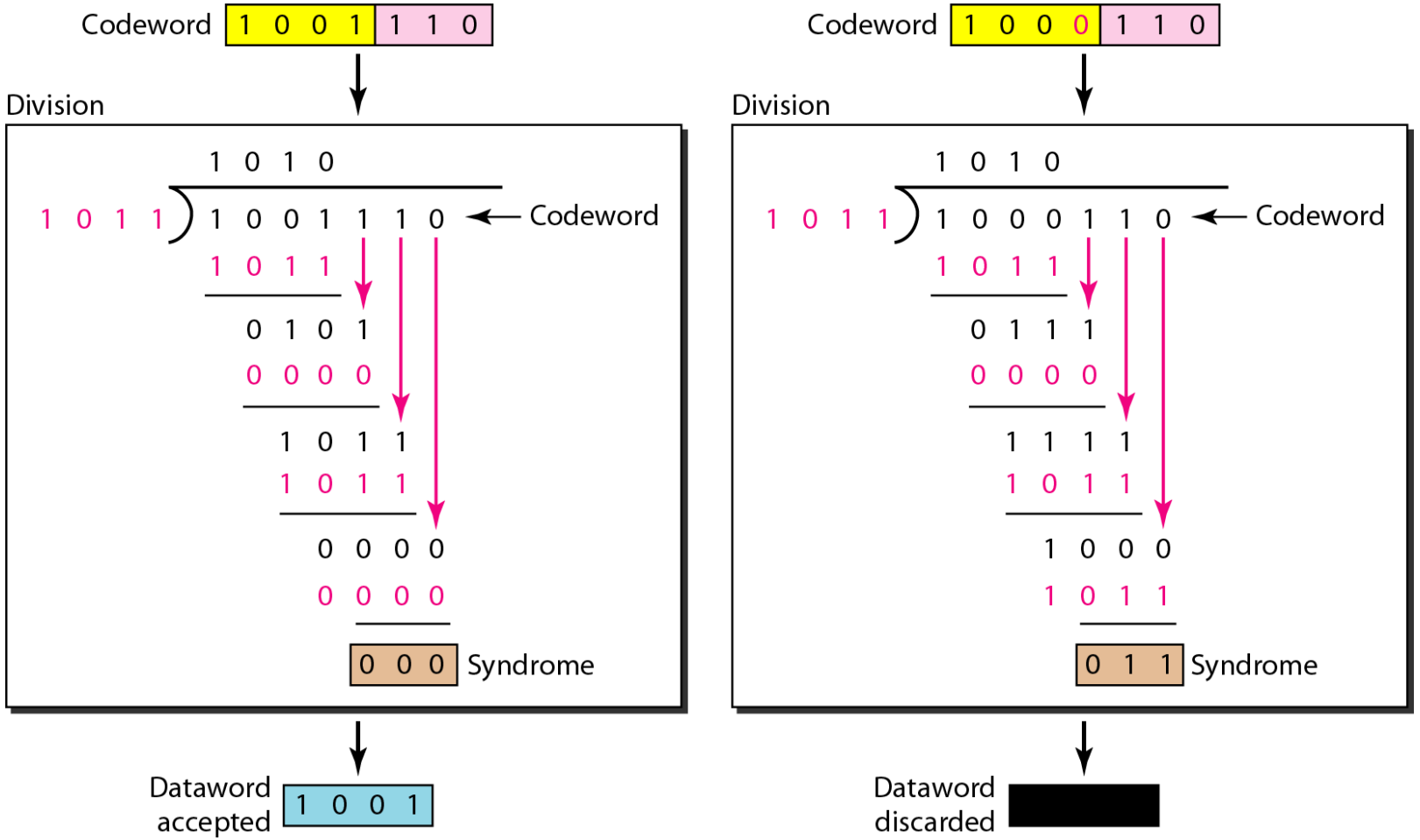
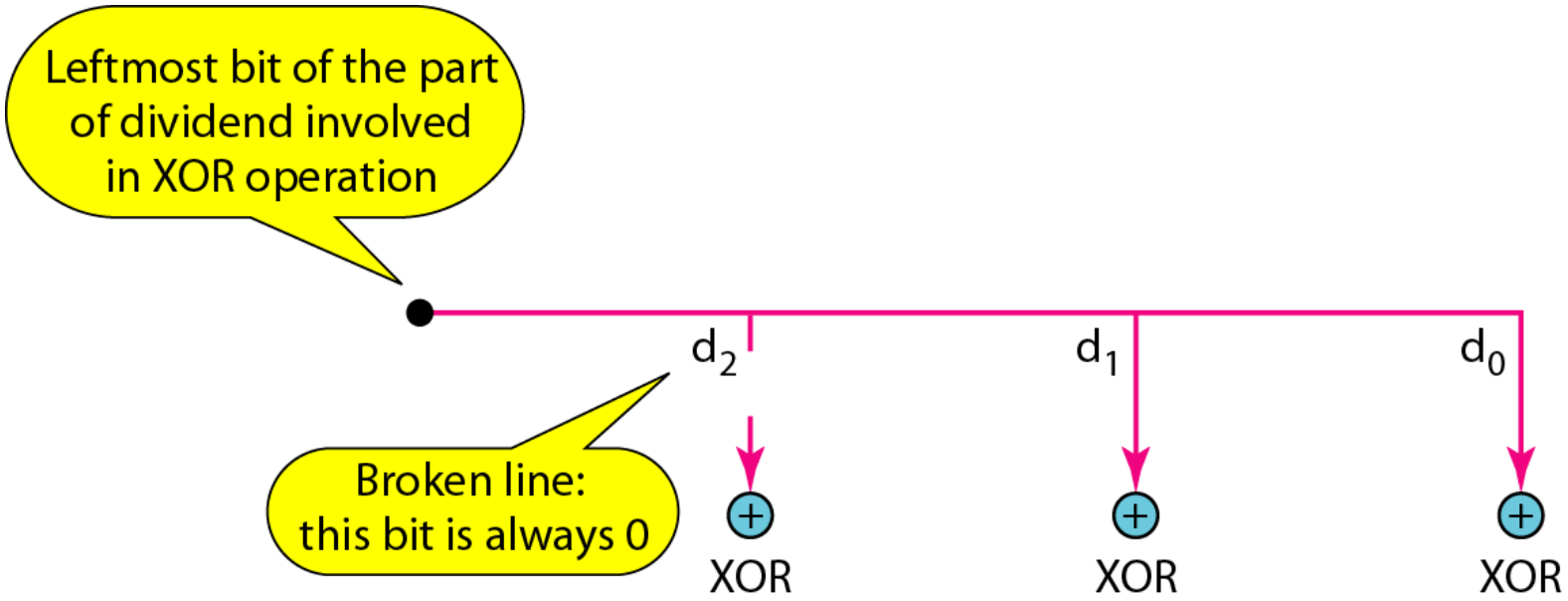


Figure 10.17 *Hardwired design of the divisor in CRC*



Step-by-step for Division in H/W-S/W

1. Assume the remainder is originally all 0s (000)
2. At each tick of the clock (arrival of 1 bit from augmented dataword), following two actions are repeated
   a. The leftmost bit is used to make a decision about the divisor (011 or 000)
   b. The other 2 bits of the remainder and the next bit from the augmented dataword (total of 3 bits) are XORed with the 3-bit divisor to create the next remainder

Figure 10.18 *Simulation of division in CRC encoder*

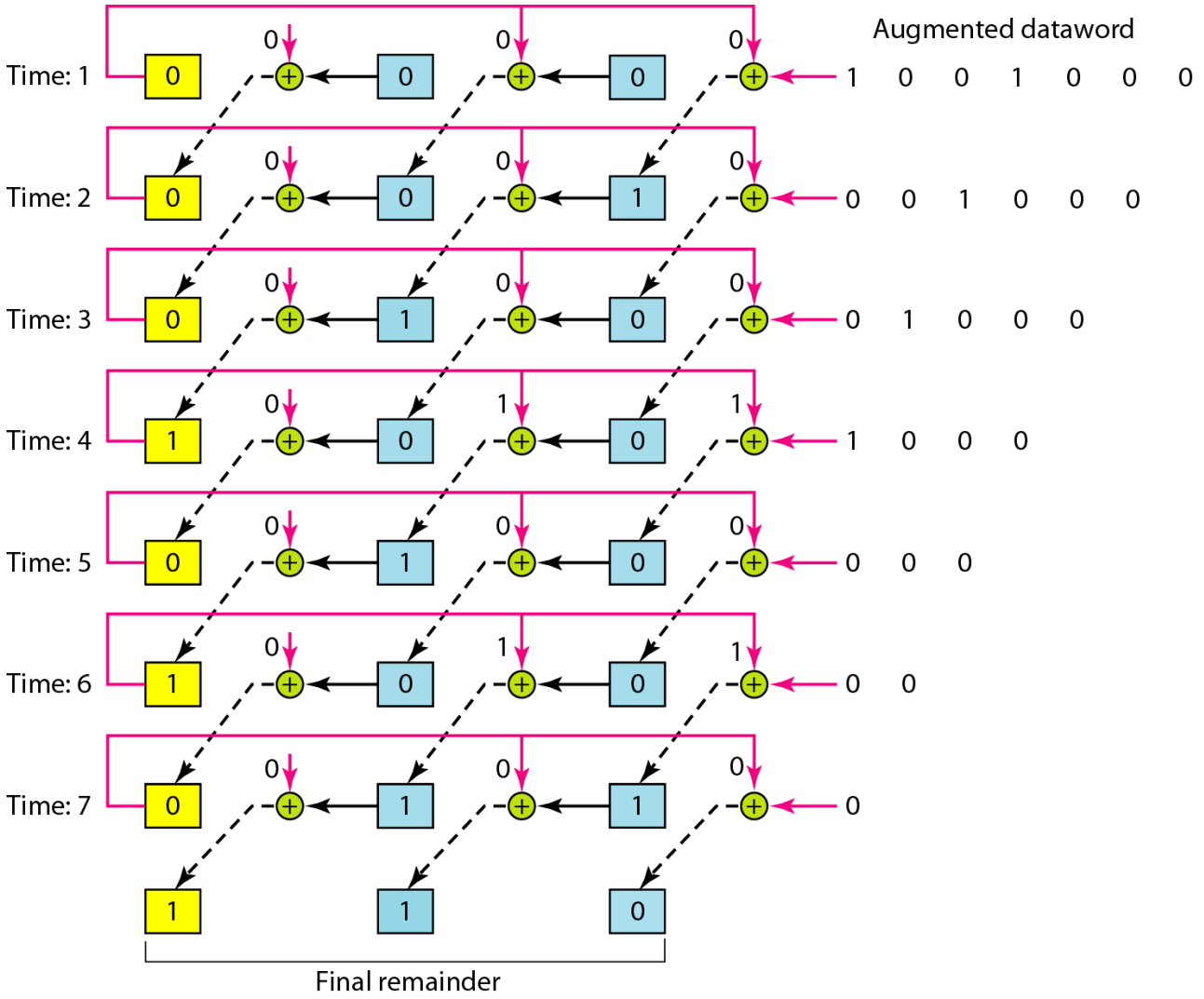


Figure 10.19 *The CRC encoder design using shift registers*

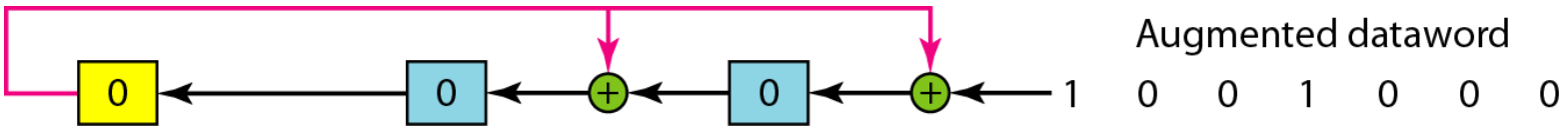


Figure 10.20 *General design of encoder and decoder of a CRC code*

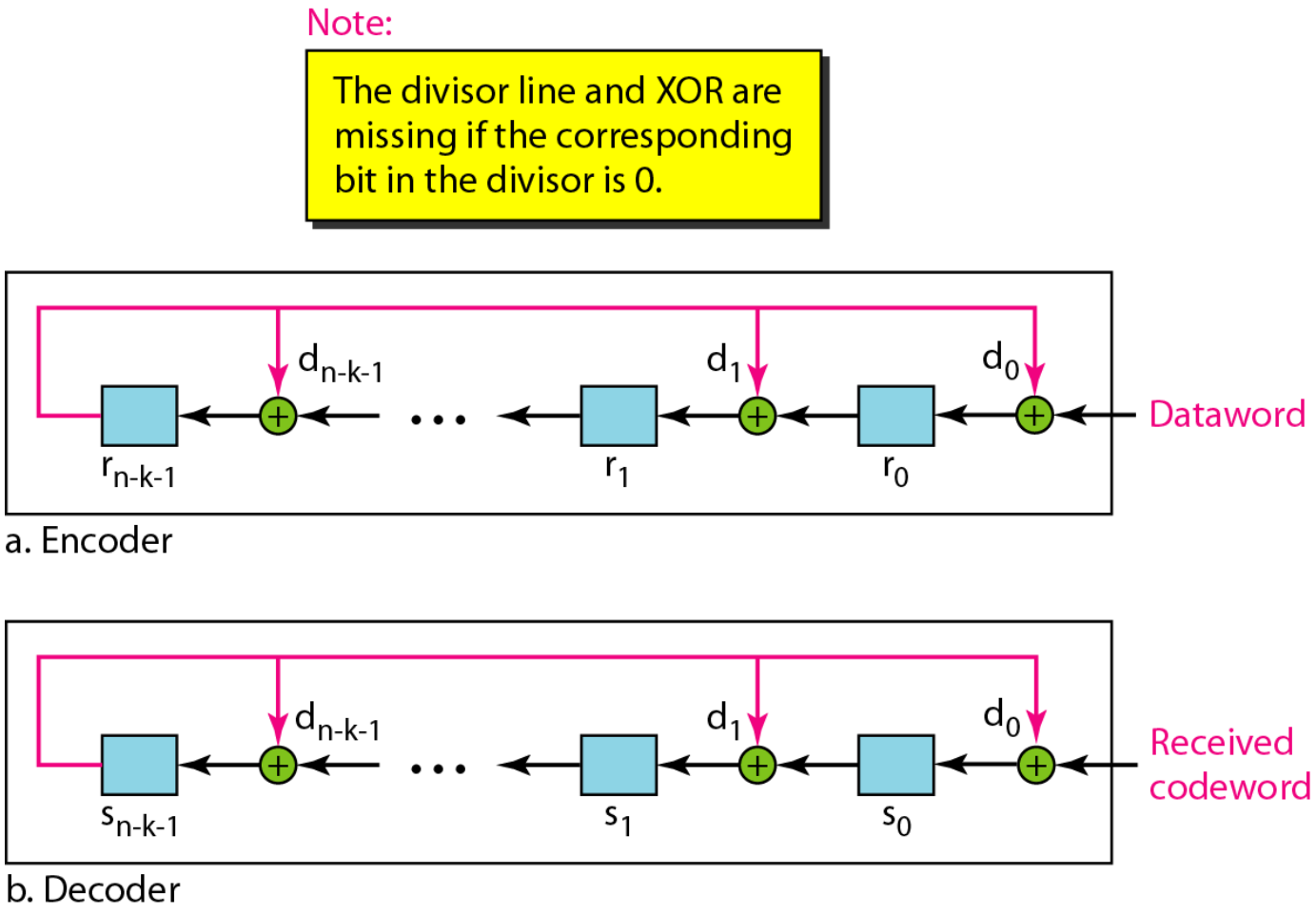
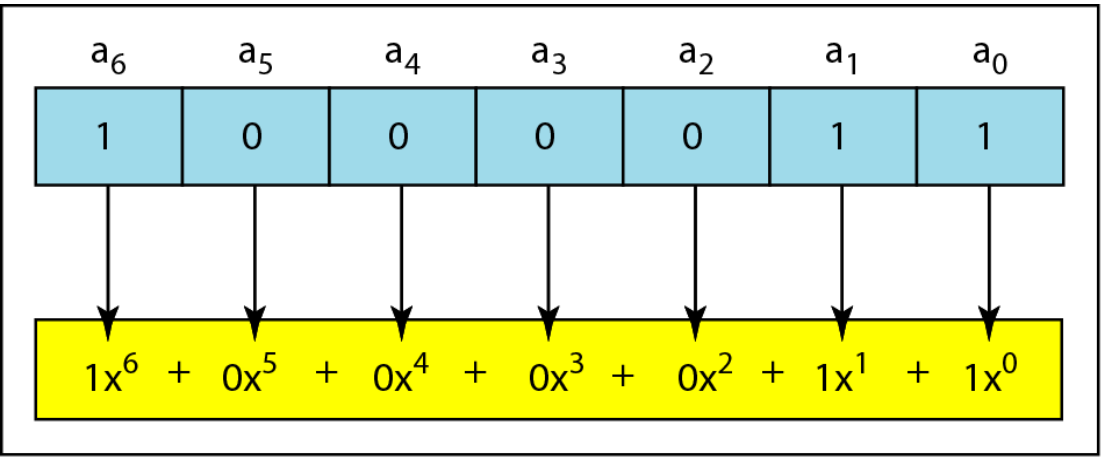
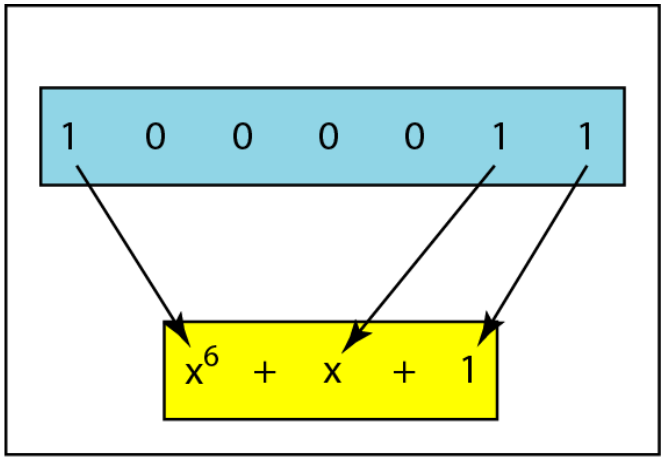


Figure 10.21 *A polynomial to represent a binary word*

- 0s and 1s can be represented as a polynomial with coefficients 0 and 1
- Power of each term shows the position of the bit; coefficient shows the value of the bit
- Degree of a polynomial is the highest power in it

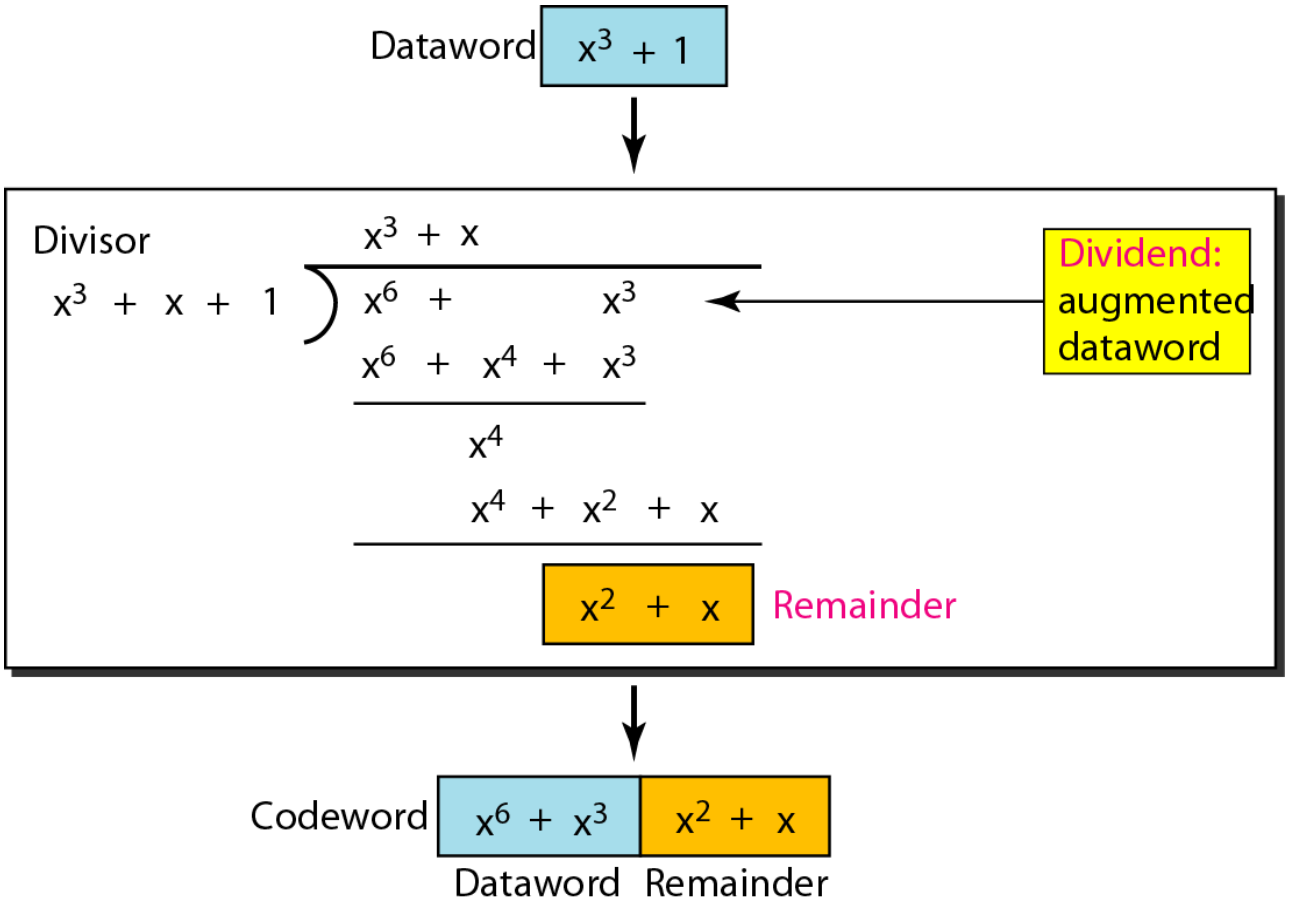


a. Binary pattern and polynomial



b. Short form

Figure 10.22 *CRC division using polynomials*



Note

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

Note

In a cyclic code,

If $s(x) \neq 0$, one or more bits is corrupted.

If $s(x) = 0$, either

a. No bit is corrupted. or

b. Some bits are corrupted, but the decoder failed to detect them.

Note

In a cyclic code, those errors [*e*(x)] that are divisible by *g*(x) are not caught.

Note

If the generator has more than one term and the coefficient of x^0 is 1, all single bit errors can be caught.

Example 10.15

Which of the following g(x) values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$ *b.* x^3 *c.* 1

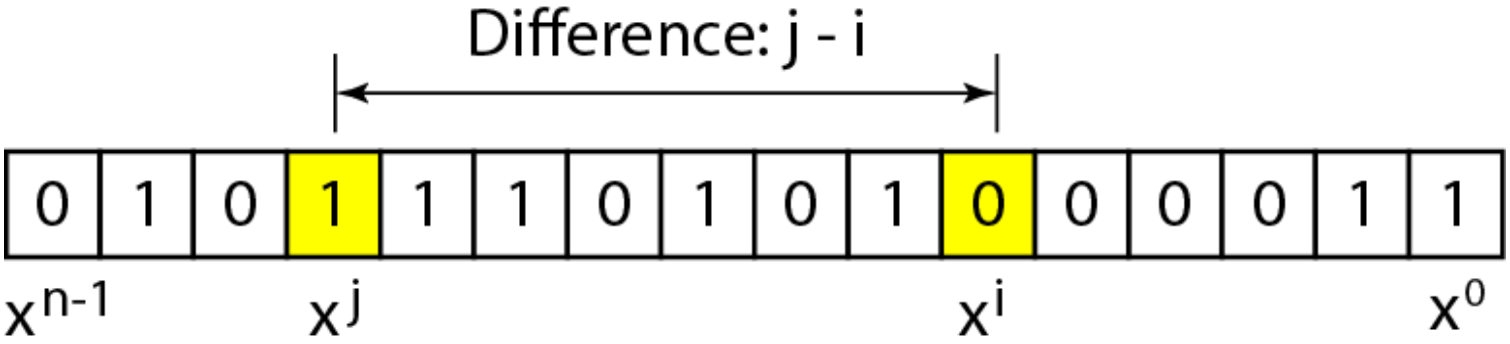
Solution

a. No x^i can be divisible by $x + 1$. Any single-bit error can be caught.

b. If i is equal to or greater than 3, x^i is divisible by g(x). All single-bit errors in positions 1 to 3 are caught.

c. All values of i make x^i divisible by g(x). No single-bit error can be caught. This g(x) is useless.

Figure 10.23 *Representation of two isolated single-bit errors using polynomials*



Note

If a generator cannot divide $x^t + 1$ (t between 0 and n – 1), then all isolated double errors can be detected.

Example 10.16

Find the status of the following generators related to two isolated, single-bit errors.

a. $x + 1$ *b.* $x^4 + 1$ *c.* $x^7 + x^6 + 1$ *d.* $x^{15} + x^{14} + 1$

Solution

a. This is a very poor choice for a generator. Any two errors next to each other cannot be detected.

b. This generator cannot detect two errors that are four positions apart.

c. This is a good choice for this purpose.

d. This polynomial cannot divide $x^t + 1$ if t is less than 32,768. A codeword with two isolated errors up to 32,768 bits apart can be detected by this generator.

Note

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

Note

- All burst errors with $L \leq r$ will be detected.
- All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$.
- All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^{r}$.

Example 10.17

Find the suitability of the following generators in relation to burst errors of different lengths.

a. $x^6 + 1$ *b.* $x^{18} + x^7 + x + 1$ *c.* $x^{32} + x^{23} + x^7 + 1$

Solution

a. This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.

Example 10.17 (continued)

b. ***This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of 1 million burst errors of length 20 or more will slip by.***

c. ***This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.***

Note

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 $x + 1$.**

Table 10.7 *Standard polynomials*

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

10-5 CHECKSUM

The last error detection method we discuss here is called the checksum. The checksum is used in the Internet by several protocols although not at the data link layer. However, we briefly discuss it here to complete our discussion on error checking

Topics discussed in this section:

Idea
One's Complement
Internet Checksum

Example 10.18

- *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 (1, 2, 3, 4, 5), we send (1, 2, 3, 4, 5 and 15), where 15 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.*

Example 10.19

- *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.*

- *We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum.*
- *In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum).*
- *If the result is 0, it assumes no error; otherwise, there is an error.*

Example 10.20

How can we represent the number 21 in one's complement arithmetic using only four bits?

Solution

The number 21 in binary is 10101 (it needs five bits).

We can wrap the leftmost bit and add it to the four rightmost bits.

We have (0101 + 1) = 0110 or 6.

Example 10.21

How can we represent the number −6 in one's complement arithmetic using only four bits?

Solution

In one's complement arithmetic, the negative or complement of a number is found by inverting all bits. Positive 6 is 0110; negative 6 is 1001.

If we consider only unsigned numbers, this is 9. In other words, the complement of 6 is 9.

Another way to find the complement of a number in one's complement arithmetic is to subtract the number from $2^n - 1$ (i.e 15 − 6 in this case).

Example 10.22

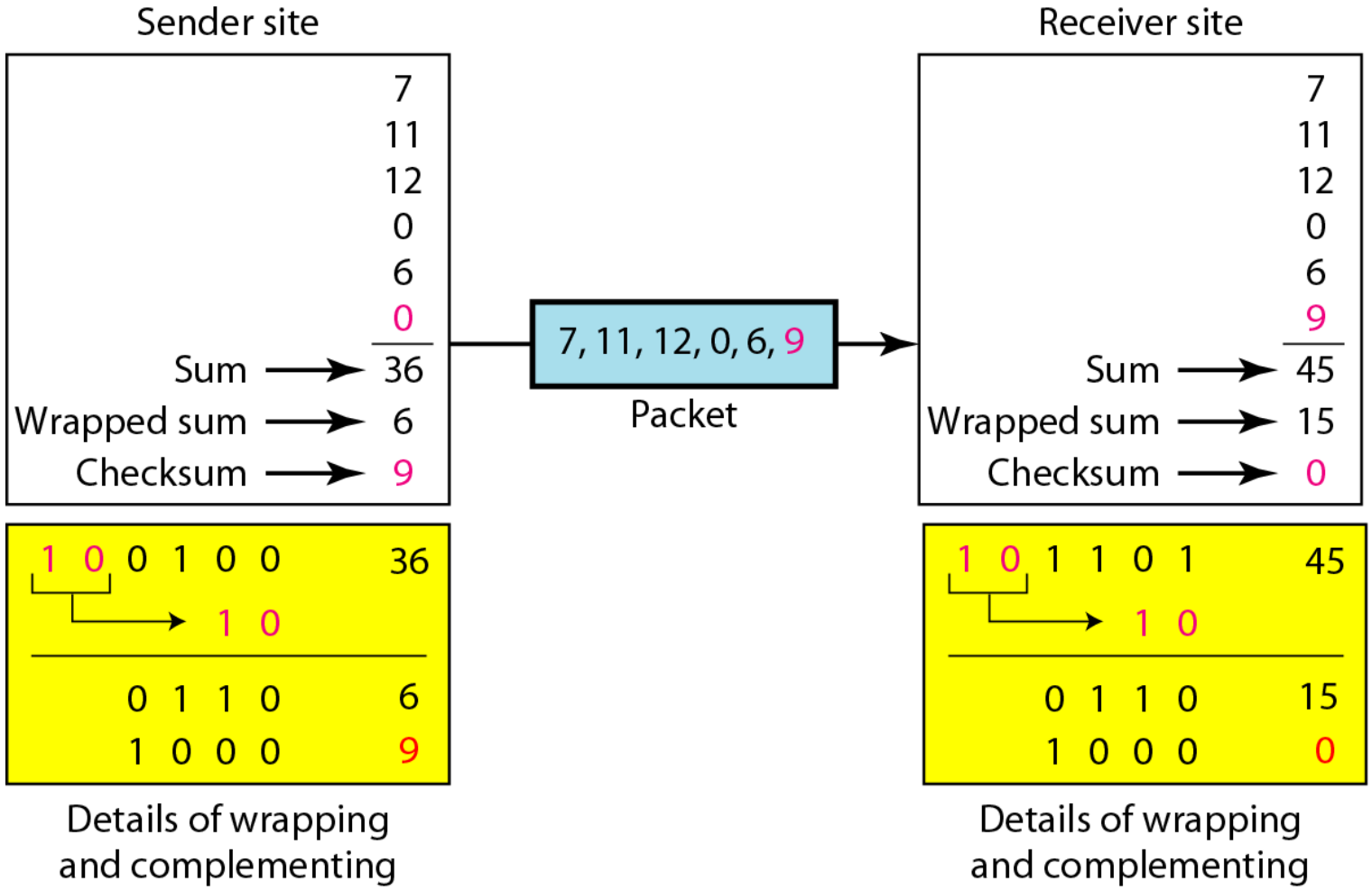
Let us redo Exercise 10.19 using one's complement arithmetic. Figure 10.24 shows the process at the sender and at the receiver.

- *Sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color) → The result is 36*
- *However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6*
- *The sum is then complemented, resulting in the checksum value 9 (15 − 6 = 9)*
- *The sender now sends six data items to the receiver including the checksum 9.*

Example 10.22 (continued)

- *The receiver follows the same procedure as the sender.*
- *It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15.*
- *The wrapped sum is complemented and becomes 0.*
- *Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items.*
- *If the checksum is not zero, the entire packet is dropped.*

Figure 10.24 *Example 10.22*



Internet Checksum

Note

Sender site:

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

Note

Receiver site:

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

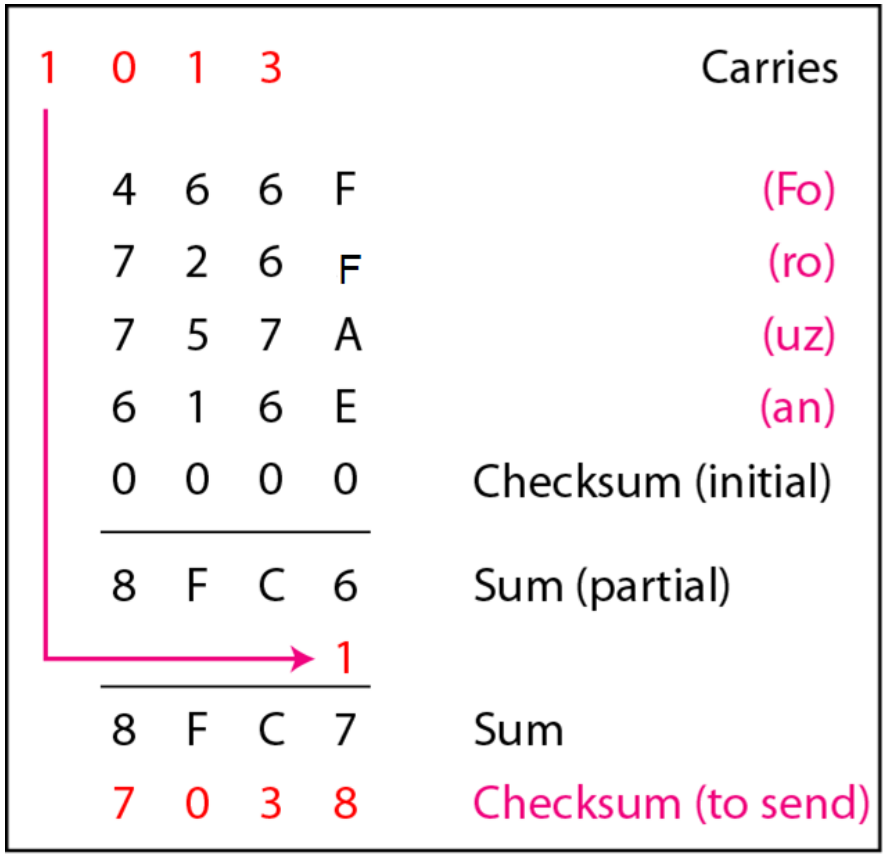
Example 10.23

Let us calculate the checksum for a text of 8 characters ("Forouzan"). The text needs to be divided into 2-byte (16-bit) words. We use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as 0x46 and o is represented as 0x6F. Figure 10.25 shows how the checksum is calculated at the sender and receiver sites. In part a of the figure, the value of partial sum for the first column is 0x36. We keep the rightmost digit (6) and insert the leftmost digit (3) as the carry in the second column. The process is repeated for each column. Note that if there is any corruption, the checksum recalculated by the receiver is not all 0s. We leave this an exercise.

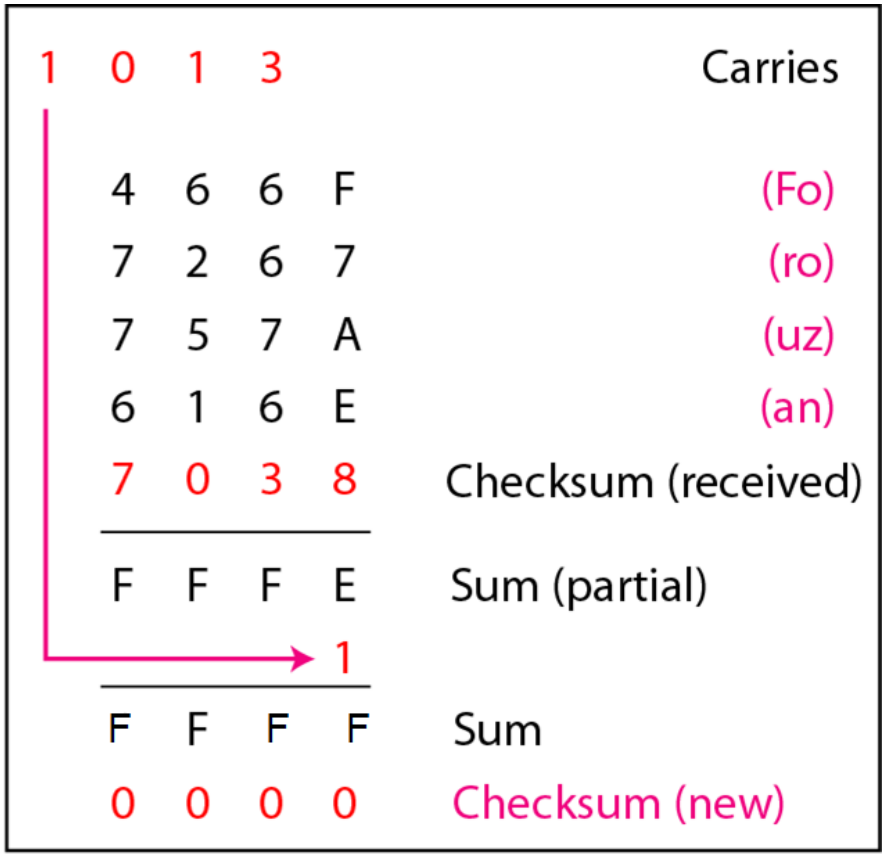
Internet Checksum Example

- Checksum for a text of 8 characters **“Forouzan”**
- ASCII Hex: **F- 46, o-6F, r-72, u-75, z-7A, a-61, n-6E**
- Arrange in 16-bit word as:
  - Fo: 46 6F
  - ro: 72 6F
  - uz: 75 7A
  - an: 61 6E
- Initialize checksum as 00 00
- Perform the sum
- Wrap to 16 bits
- Complement

Figure 10.25 *Example 10.23*



a. Checksum at the sender site



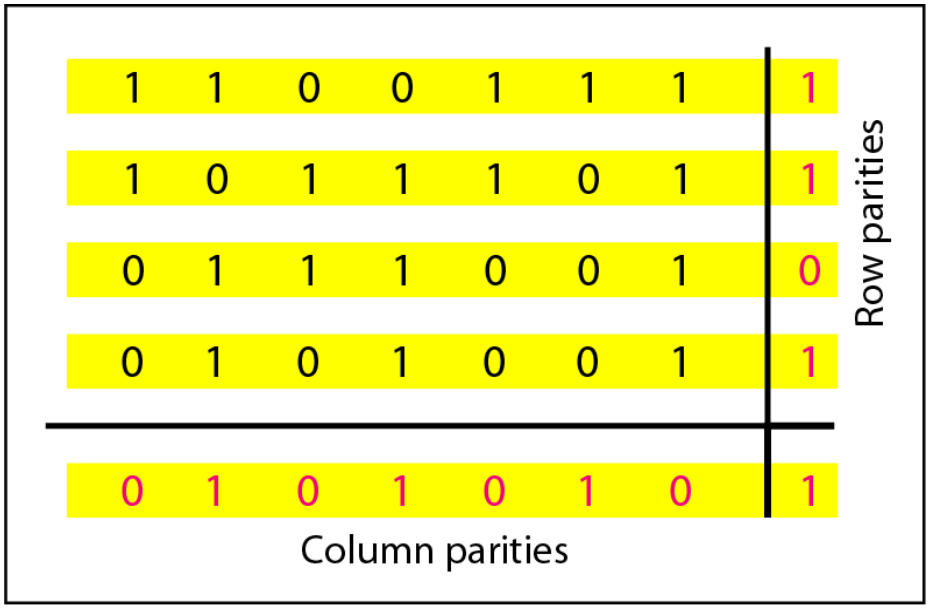
a. Checksum at the receiver site

Internet Checksum Example

- Checksum for a text **“KJSomaiya”**
- ASCII Hex: **K- 4B, J-4A, S-53, o-6F, m-6D, a-61, i-69, y-79**

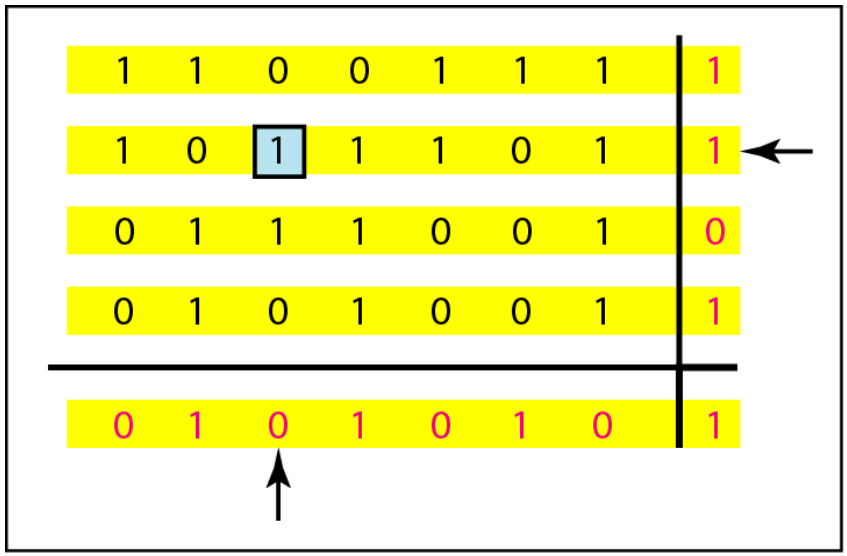
- Arrange in 16-bit word as:
- Initialize checksum as 00 00
- Perform the sum
- Wrap to 16 bits
- Complement

Figure 10.11 *Two-dimensional parity-check code*

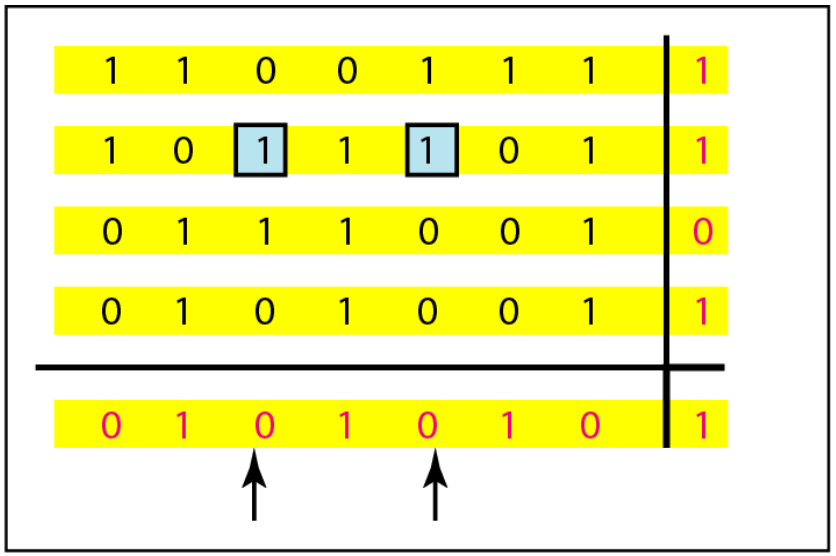


a. Design of row and column parities

Figure 10.11 *Two-dimensional parity-check code*

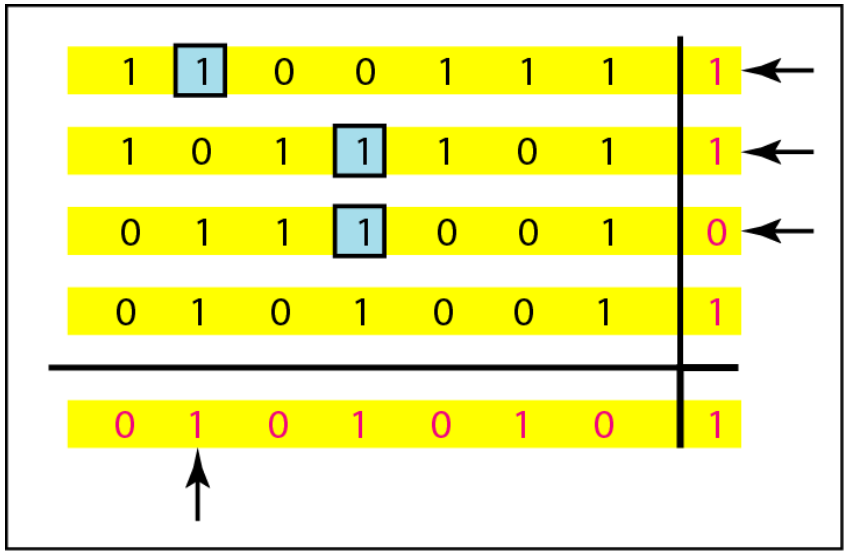


b. One error affects two parities

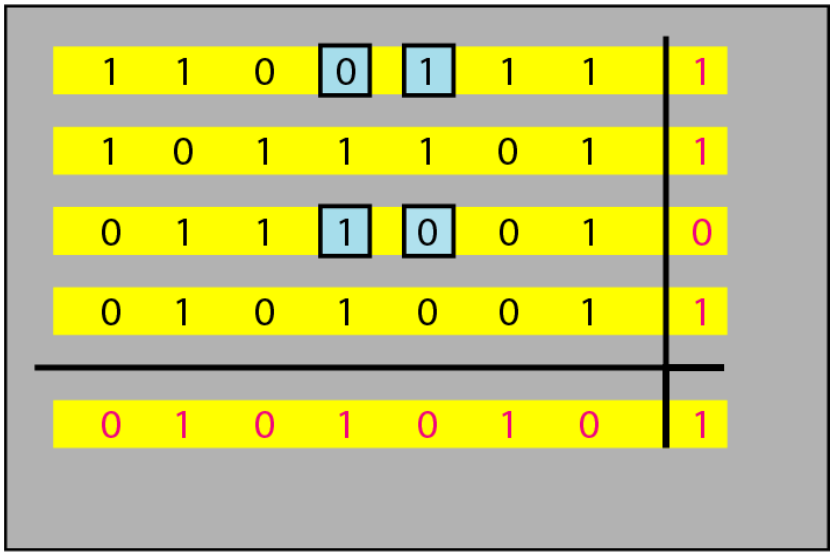


c. Two errors affect two parities

Figure 10.11 *Two-dimensional parity-check code*



d. Three errors affect four parities



e. Four errors cannot be detected

Module 2: Data Link and MAC Layer

Part-II Framing and Flow Control

Dr. Prasanna Shete
Dept. of Computer Engineering
K. J. Somaiya College of Engineering

Slide Source: B. A. Forauzan, Data Communications and Networking, McGraw-Hill Online Learning Centre
http://highered.mheducation.com/sites/0072967757/information_center_view0/index.html

11-1 FRAMING

The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.

Topics discussed in this section:

Fixed-Size Framing
Variable-Size Framing

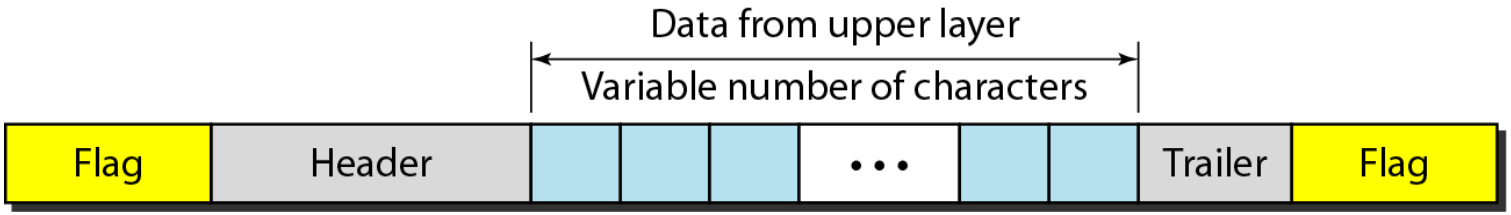
Framing in DLL

- Framing separates a message from one source to a destination, or from other message to other destinations
- Messages divided into smaller frames for efficient flow and error control
  - For large frame single-bit error would require retransmission of whole message
- Types: Fixed-size framing, Variable-size framing
- In Fixed-size framing no need for defining boundaries of the frame; size used as delimiter
- In variable-sized framing mechanism to define the end of the frame and the beginning of the next is required
  - Character oriented approach or bit-oriented approach

Character-Oriented Protocols

- Data to carried are 8-bit characters from a coding system e.g ASCII
  - Source and destination addresses and other control information carried in the Header or error detection /correction bits carried in the Trailer are multiples of 8-bits
- To separate one frame from the next, an 8-bit flag is added at the beginning and the end of a frame
- Flag composed of special character
- Any character not used for text-communication can be selected as flag

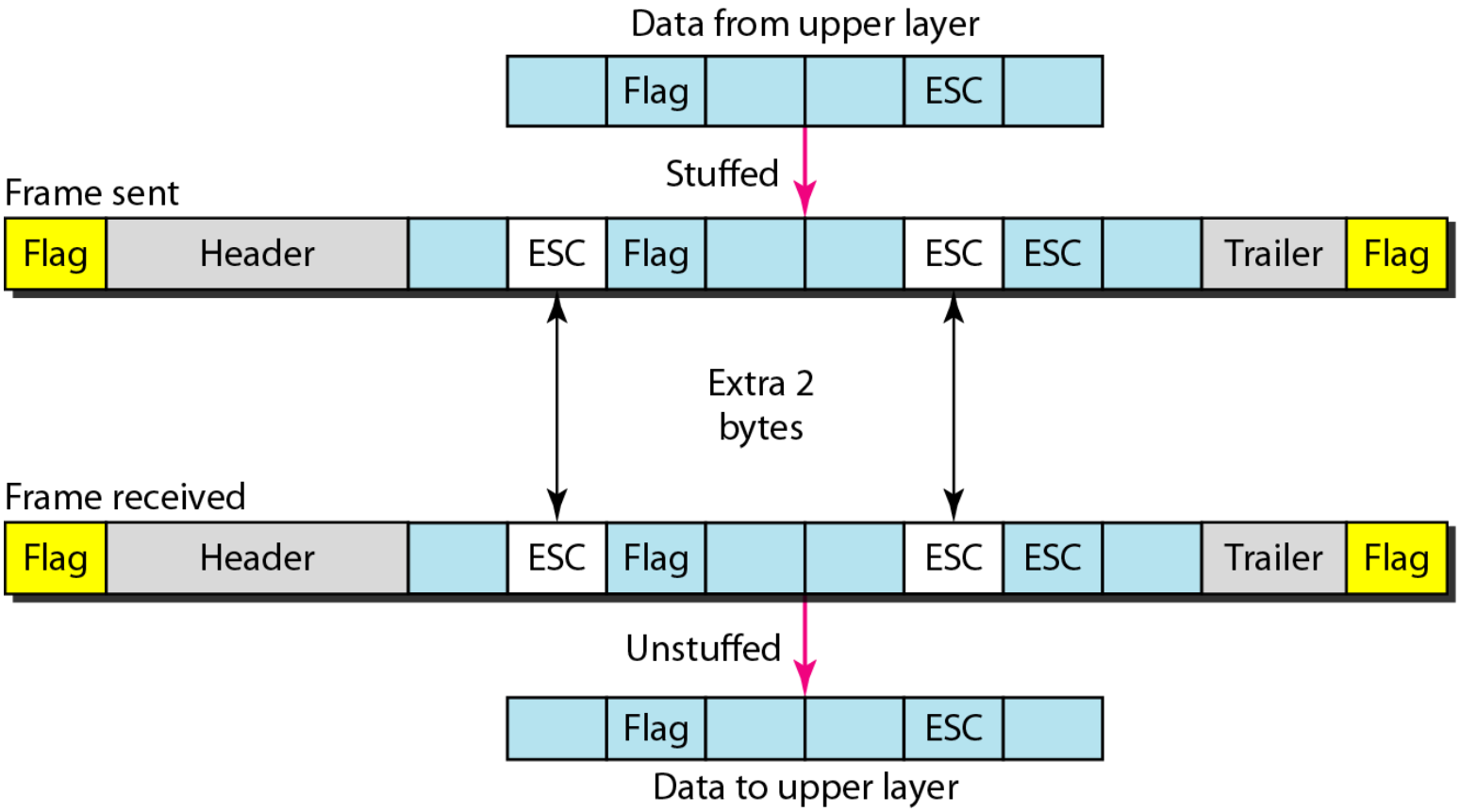
Figure 11.1 *A frame in a character-oriented protocol*



Character-Oriented Protocols contd…

- Problem: Pattern used for flag could be part of information (images, video or audio)
- On receiving such pattern in the middle of the data, Receiver thinks it has reached the end of the frame
- Solution: ***Byte-stuffing*** (or character stuffing)
- when there is a character with the same pattern as the flag, data section is stuffed with an extra byte- called escape character (**ESC),** having predefined bit pattern
- Receiver on encountering the ESC character, removes it from the data section and treats the next character as data, not as delimiting flag
- What if data consists bit pattern same as ESC?

Figure 11.2 *Byte stuffing and unstuffing*



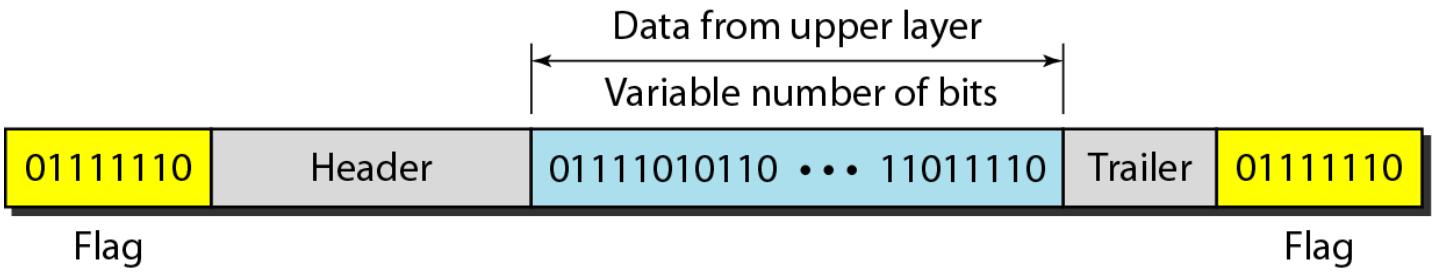
Note

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

Bit-Oriented Protocols

- Data section of a frame is a sequence of bits (representing text, graphic, audio, video etc.)
- To separate one frame from the other a delimiter is needed
- Special 8-bit pattern flag 01111110 used in most protocols
- If the flag pattern appears in the data, a mechanism is needed to inform the receiver that this is not the end of the frame
- Achieved by ***bit-stuffing***; a single bit is stuffed in the data
- If a 0 and five consecutive 1 bits are encountered, an extra 0 is added
  - Stuffed bit is removed from the data by the receiver

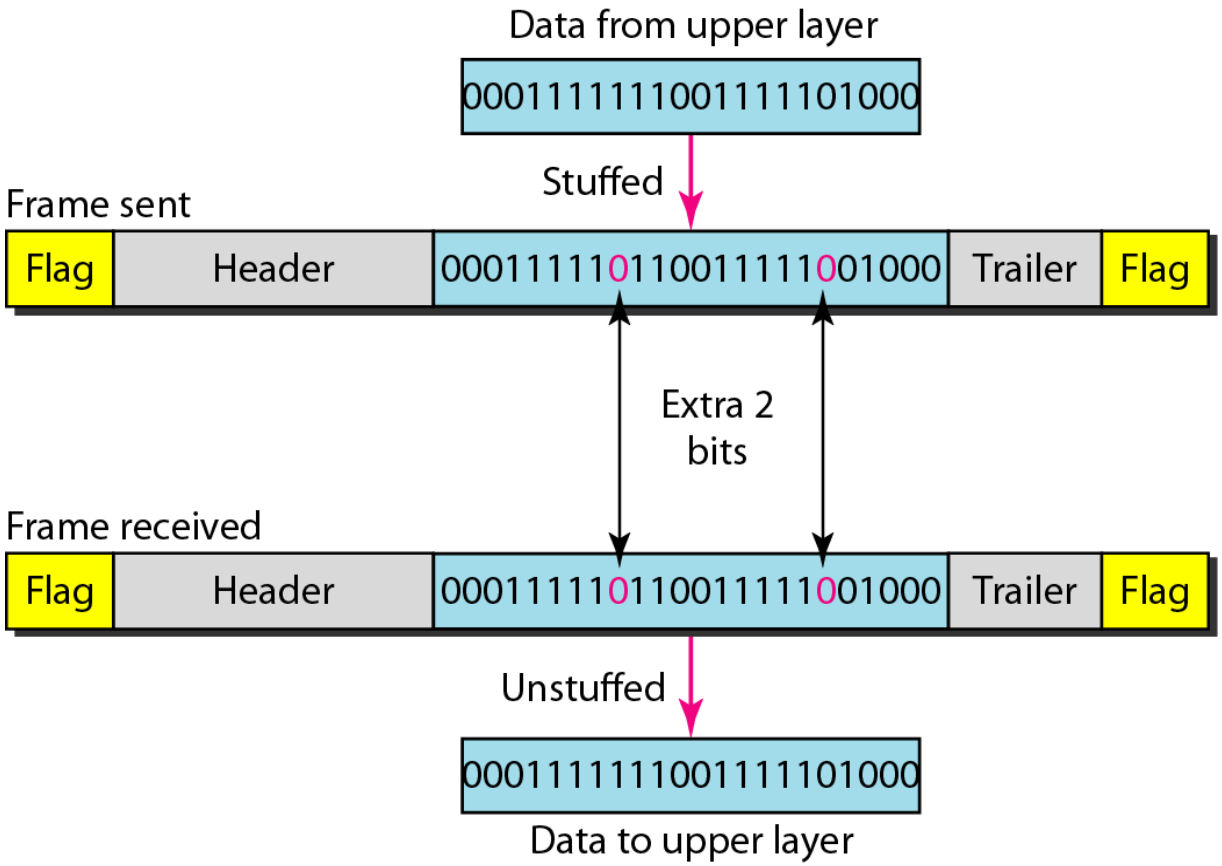
Figure 11.3 *A frame in a bit-oriented protocol*



Note

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

Figure 11.4 *Bit stuffing and unstuffing*



11-2 FLOW AND ERROR CONTROL

The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

Topics discussed in this section:

Flow Control
Error Control

Flow control ***coordinates the amount of data that can be sent before receiving an acknowledgement***

-Flow of data should not be allowed to overwhelm the receiver

Note

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* – *Error detection and correction

-When error is detected during exchange of data, specified frames are retransmitted

Note

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

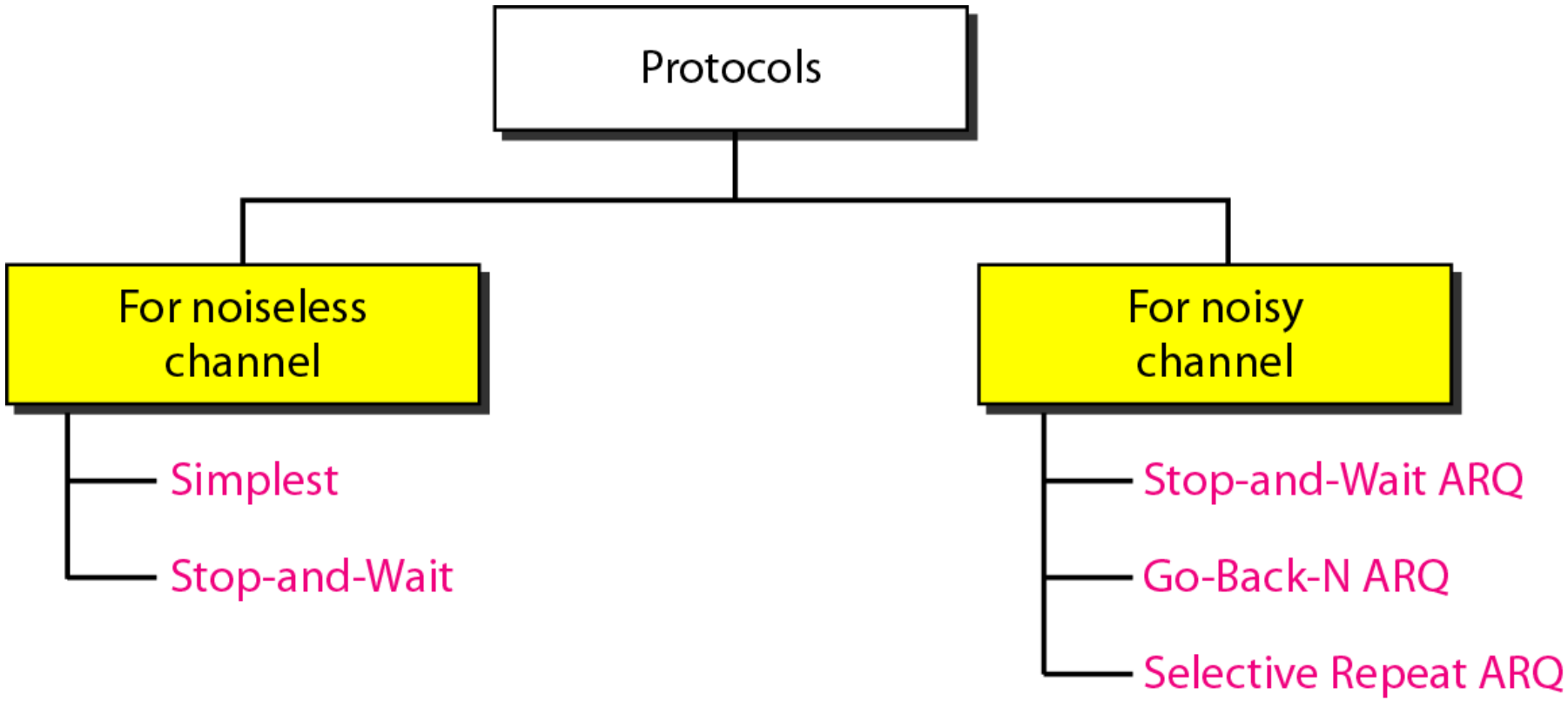
11-3 PROTOCOLS

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

The protocols are normally implemented in software by using one of the common programming languages.

To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.

Figure 11.5 *Taxonomy of protocols discussed in this chapter*



11-4 NOISELESS CHANNELS

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 and Stop-and-Wait Protocol

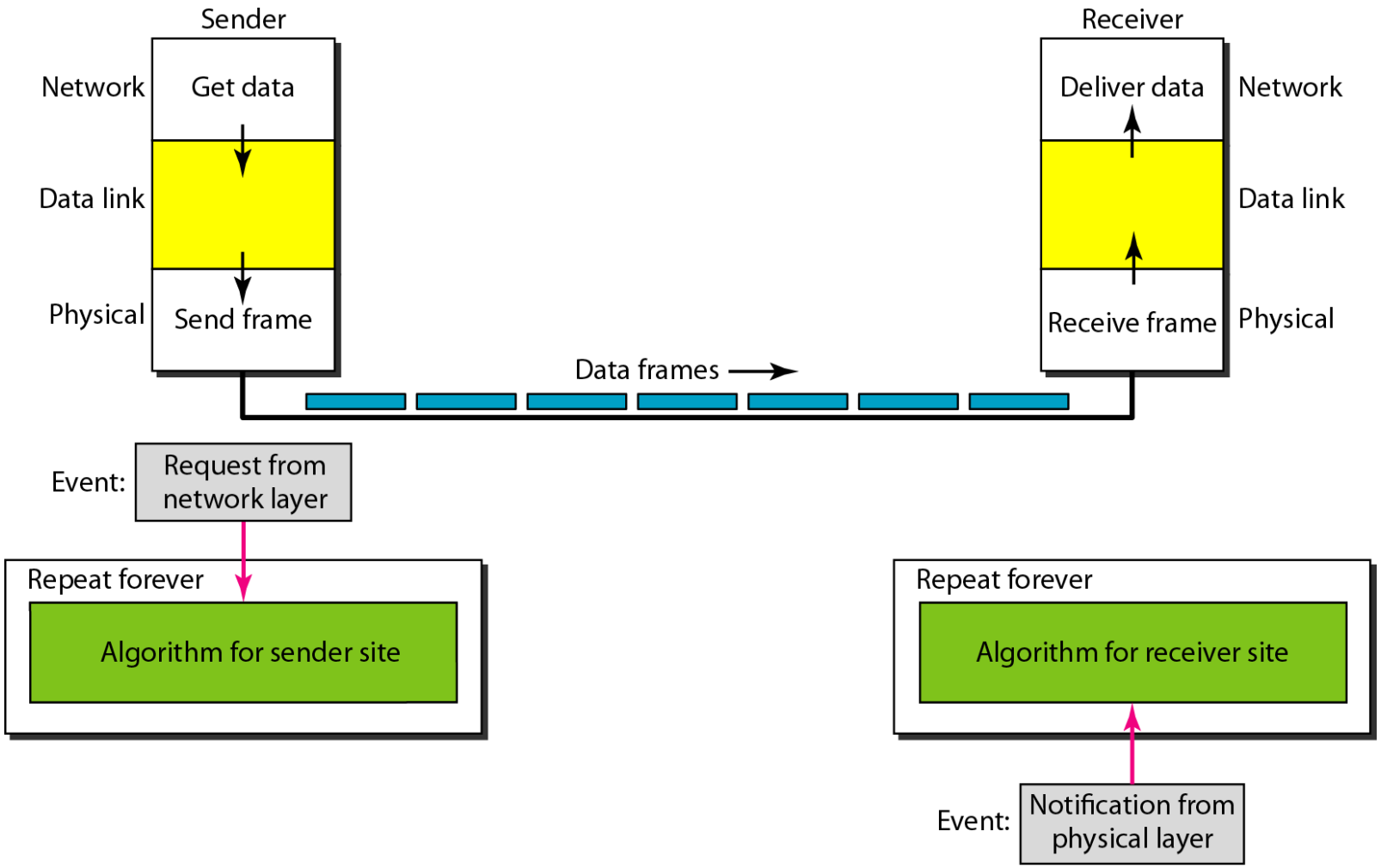
Topics discussed in this section:

Simplest Protocol
Stop-and-Wait Protocol

Simplest Protocol

- No flow control
- At Sender site: Data link layer gets data from Network layer, makes a frame out of the data, and sends it
- At Receiver : Data link layer receives a frame from its physical layer, extracts data, and delivers to the network layer.
- Sender cannot send a frame until its Network layer has data packet to send
- Receiver cannot deliver data packet to its Network layer until a frame arrives at PHY layer
  - Event driven protocol, implemented as procedures at sender and receiver (both running constantly, since they do not know when corresponding events will occur)

Figure 11.6 *The design of the simplest protocol with no flow or error control*



Algorithm 11.1 *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 }
```

Algorithm 11.2 *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 }
```

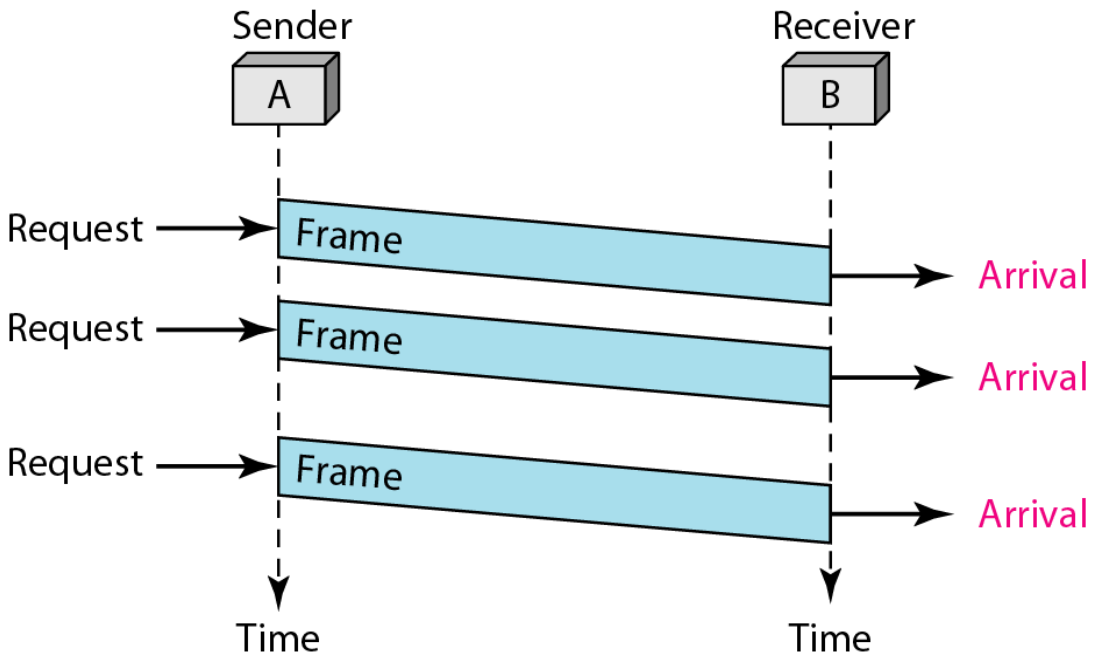
Example 11.1

Figure 11.7 shows an example of communication using this protocol.

It is very simple → The sender sends a sequence of frames without even thinking about the receiver.
To send three frames, three events occur at the sender site and three events at the receiver site.

Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.

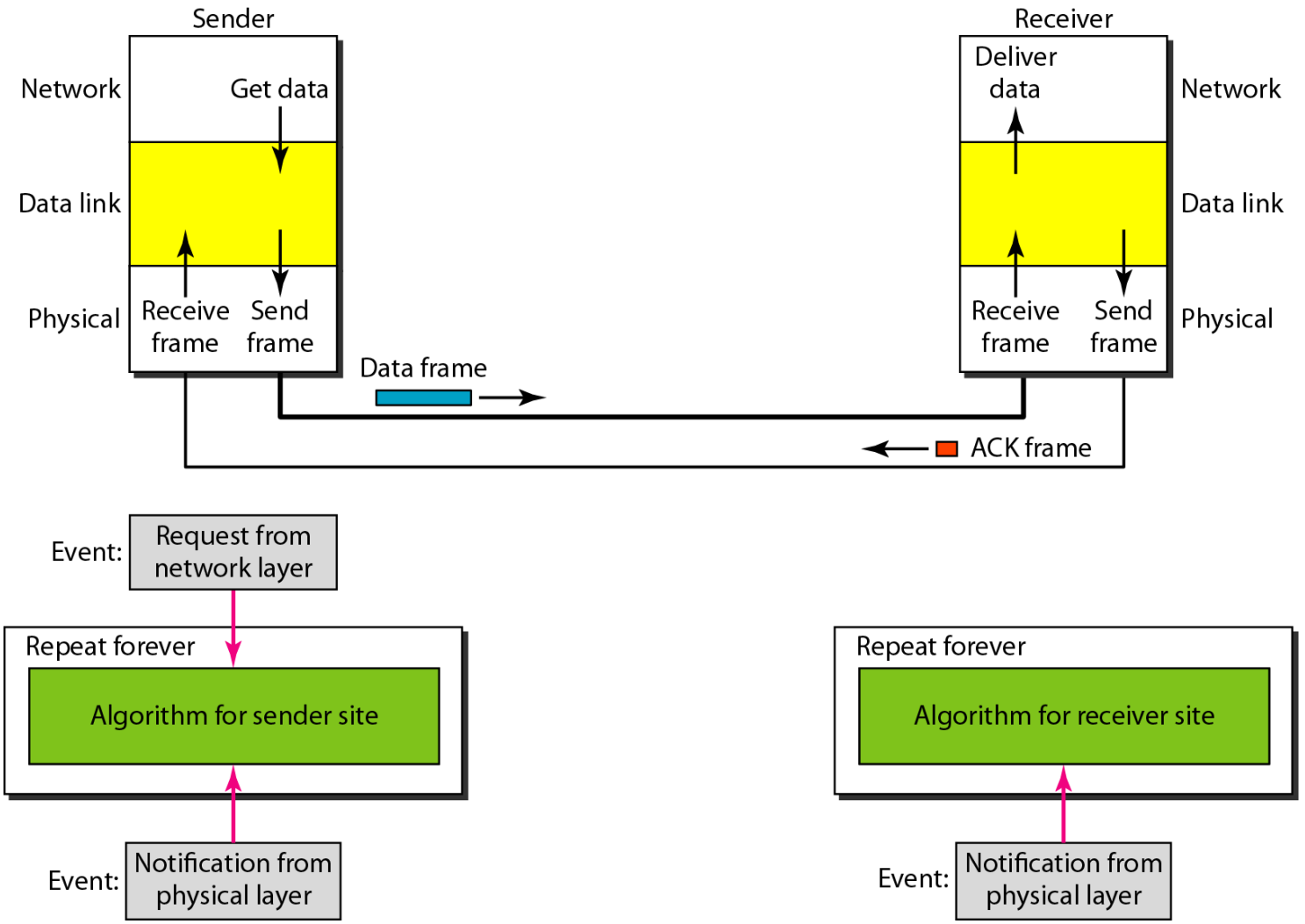
Figure 11.7 *Flow diagram for Example 11.1*



Stop-and-Wait Protocol

- If data frames arrive at the receiver faster than that can be processed, the receiver may discard (if no sufficient storage space available)
- To prevent overwhelming of the receiver, a mechanism needed to tell the sender to slow down→Stop-and-Wait:
- Sender sends one frame, stops until it receives confirmation from the Receiver to go ahead, and then sends the next frame
  - Thus flow control added to previous protocol
- Here, traffic in both directions; sender to receiver and vice-versa
  - either one data frame on forward channel or one ACK frame on the reverse channel → Half-Duplex links required

Figure 11.8 *Design of Stop-and-Wait Protocol*



Algorithm 11.3 *Sender-site algorithm for 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 }
```

Algorithm 11.4 *Receiver-site algorithm for 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 }
```

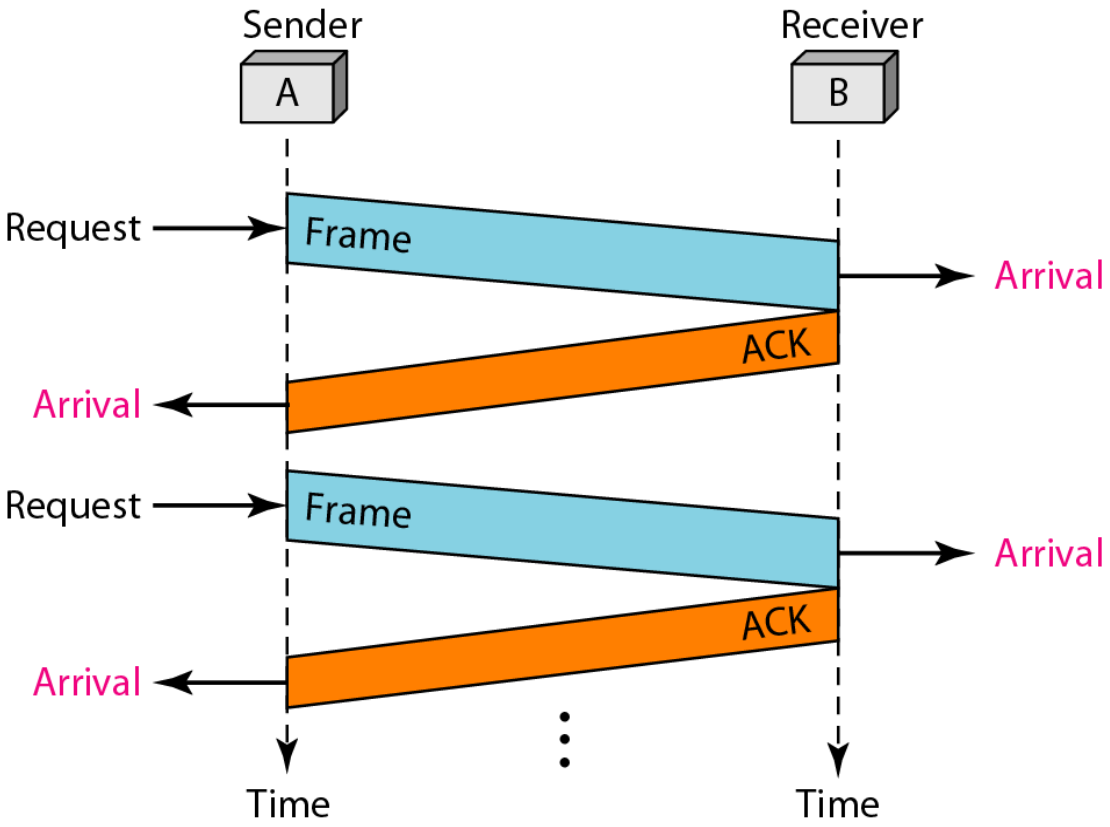
Example 11.2

Figure 11.9 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.

Figure 11.9 *Flow diagram for Example 11.2*



11-5 NOISY CHANNELS

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.

Topics discussed in this section:

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

Stop-and-Wait ARQ

- Stop-and-Wait ARQ adds a simple error control mechanism to Stop-and-Wait protocol
- When the frame arrives at the Receiver, it is checked and if it is corrupted, it is silently discarded →not acknowledged
  - detection of errors through silence of the Receiver
- Handling of lost frame: Lost frames are difficult to handle than corrupted frames
  - Received frame could be correct one, or a duplicate, or out of order
- Solution is to number the frames (sequence number)
  - when the receiver receives an out of order frame, this means the frames were either lost or duplicated

Stop-and-Wait ARQ contd..

- Corrupted and lost frames need to be resent
  - How will sender know which frame to resend?
- Sender keeps a copy of the sent frame and starts the Timer
- If the timer expires and there is no ACK, the frame is resent; the copy is held, and the timer is restarted

- ACK frame can also be corrupted → needs redundancy bits, and sequence numbers
  - Sender simply discards a corrupted ACK frame or out-of-order ACK frame

Note

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.

Stop-and-Wait ARQ: Sequence / ACK Numbers

- Sequence number field added to data frame
- To keep the frame size minimal, Range of sequence numbers should be as small as possible; can wrap around
  - E.g. for sequence number of *m* bits, Range→0 to 2^m-1, and then repeated
- Receiver only needs to know correct receipt of data frame & ACK, and loss of data or ACK
- →Sequence numbers needed are *x* (current frame) and *x*+1 (next frame)
  - then if *x*=0, *x*+1= 1; i.e SeqNos will be 0,1,0,1…
- ACK numbers announce the sequence number of next frame expected by the receiver
  - E.g. if receiver receives frame 0 safe and sound, it sends ACK frame with seqNo. 1, meaning next frame 1 is expected

Note

In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.

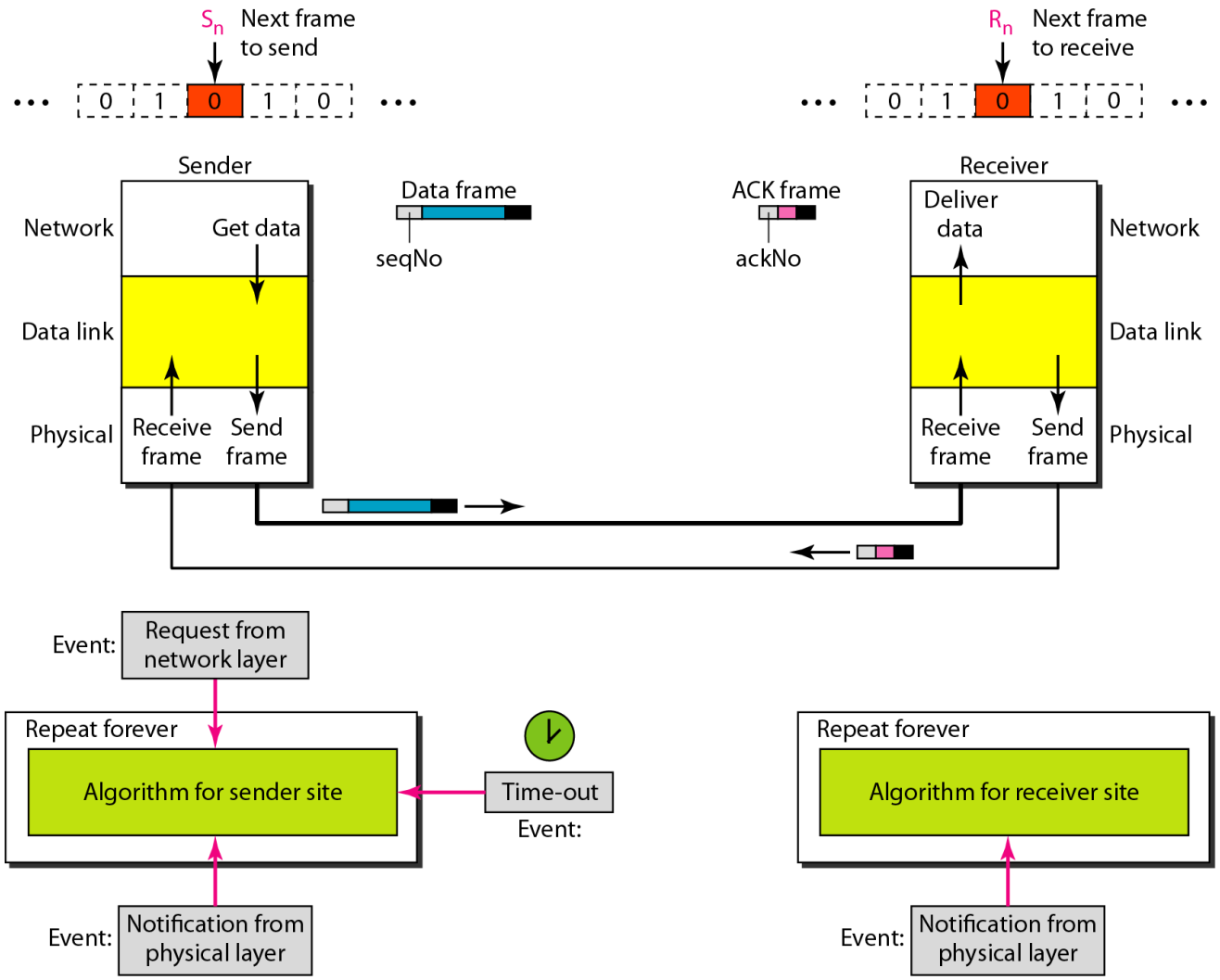
Note

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: Design

- Sender keeps a copy of the last frame transmitted until it receives an acknowledgement
- Data frame uses a seqNo; an ACK frame uses an ackNo.
- Two control variables S_n (sender next frame to send) and R_n (receiver next frame expected)
- Frame is sent- value of S_n *incremented* (modulo 2; if its 0, it will become 1 and vice-versa)
- Frame is received- value of R_n *incremented* (modulo2)
- 3 events can happen at sender site; 1 event can happen at receiver site
- S_n points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged;
- R_n points to the slot that matches the seq. no. of the expected frame

Figure 11.10 *Design of the Stop-and-Wait ARQ Protocol*



Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ*

```
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
```

(continued)

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ* (continued)

```
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 11.6 *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 }
```

Example 11.3

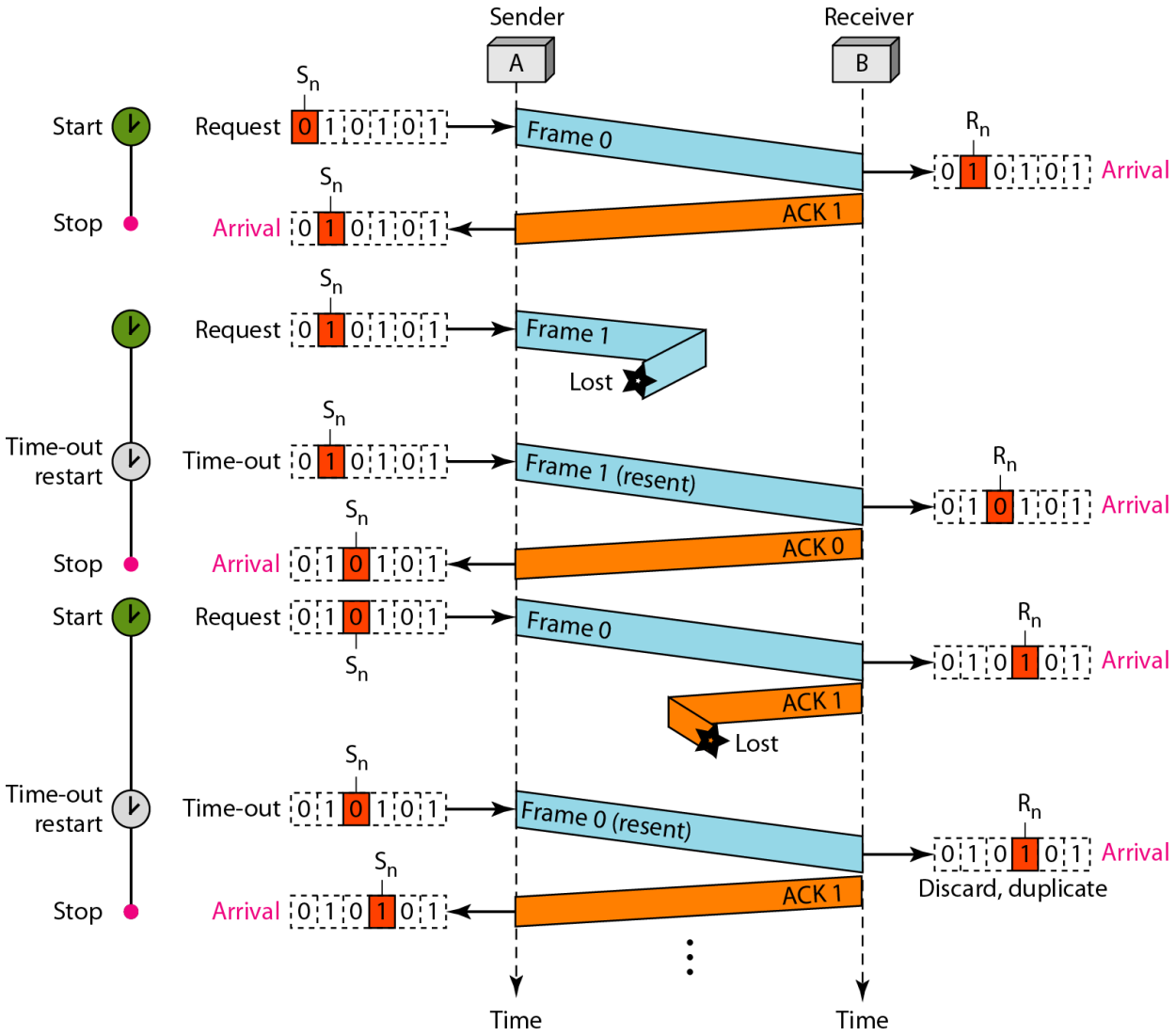
Figure 11.11 shows an example of Stop-and-Wait ARQ.

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.

Figure 11.11 *Flow diagram for Example 11.3*



Stop-and-Wait ARQ: *Efficiency*

- Stop-and-Wait ARQ is very in-efficient, when the channel has large Bandwidth and round-trip Delay (i.e ***Bandwidth-Delay*** product)

- *Bandwidth-Delay* product is the measure of the number of bits the sender can send out while waiting for feedback from the receiver

Example 11.4

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.

a) What is the bandwidth-delay product?

b) If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product= BW (bps) * Delay (s)

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

Example 11.4 (continued)

b) According to BW-Delay product, 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.

Example 11.5

What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits.
The system can send up to 15 frames or 15,000 bits during a round trip.
This means the utilization is 15,000/20,000, or 75 percent.

Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

Pipelining

- Pipelining means beginning a new task before the previous task has been completed
- No pipelining in Stop-and-Wait ARQ
  - its needed to wait for a frame to reach the destination and be acknowledged, before sending next frame
- With *pipelining,* several frames can be sent before receiving acknowledgement for previous frames
- →Pipelining improves the transmission efficiency, if the number of bits in transition is large w.r.t BW-delay product
  - Used in next schemes; Go-back-N ARQ and Selective Repeat ARQ

Go-Back-N ARQ

- To improve the efficiency of transmission, multiple frames must be in transition while waiting for acknowledgements

- Go-back-N can send multiple frames before receiving acknowledgements
- The sender keeps copy of sent frames until the ACKs arrive

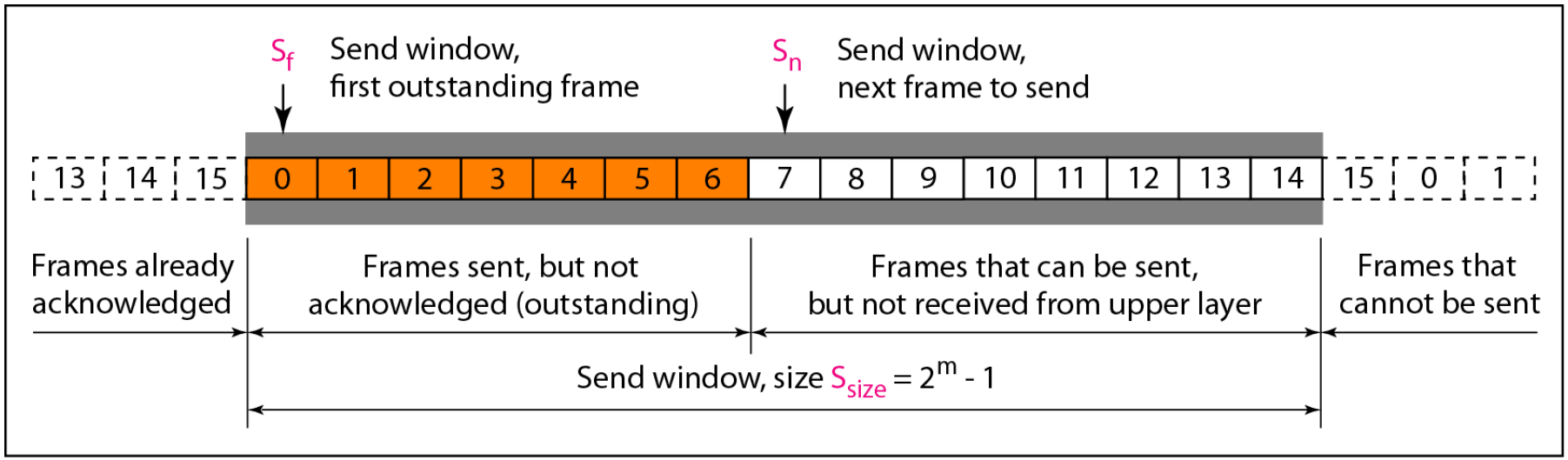
Go-Back-N ARQ: Sequence Numbers and Sliding window

- Frames from Sender are numbered sequentially
  - Limit set, to keep the frame header size minimal
- If *m* bit Sequence number allowed, the sequence numbers range from→0 to 2^m-1, and then repeated
  - For example if m=4, seqNos are: 0 to 15
  - i.e sequence numbers are modulo 2^m
- Sliding Window: Defines the range of sequence numbers that is the concern of the sender and receiver
  - send sliding window, and receive sliding window
- The send window is an imaginary box covering the sequence numbers of data frames which can be in transit
  - Maximum size of the window is 2^m-1
- Three variables define the size and location: S_f (first outstanding frame), S_n (next frame to be sent) and S_{size}

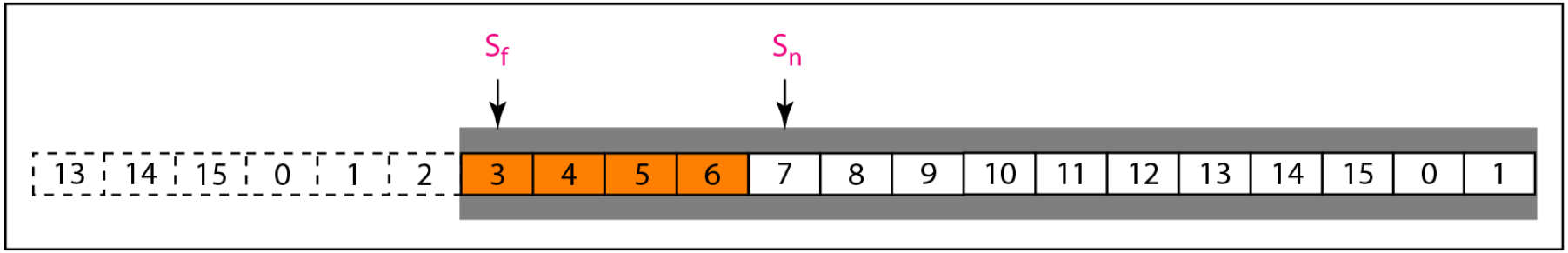
Note

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.

Figure 11.12 *Send window for Go-Back-N ARQ*



a. Send window before sliding



b. Send window after sliding

Note

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

Note

The send window can slide one or more slots when a valid acknowledgment arrives.

- The receive window ensures that correct frames are received and correct ACKs are sent
  - Any frame arriving out of order at the receiver, is discarded and needs to be resent

Note

The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.

Figure 11.13 *Receive window for Go-Back-N ARQ*

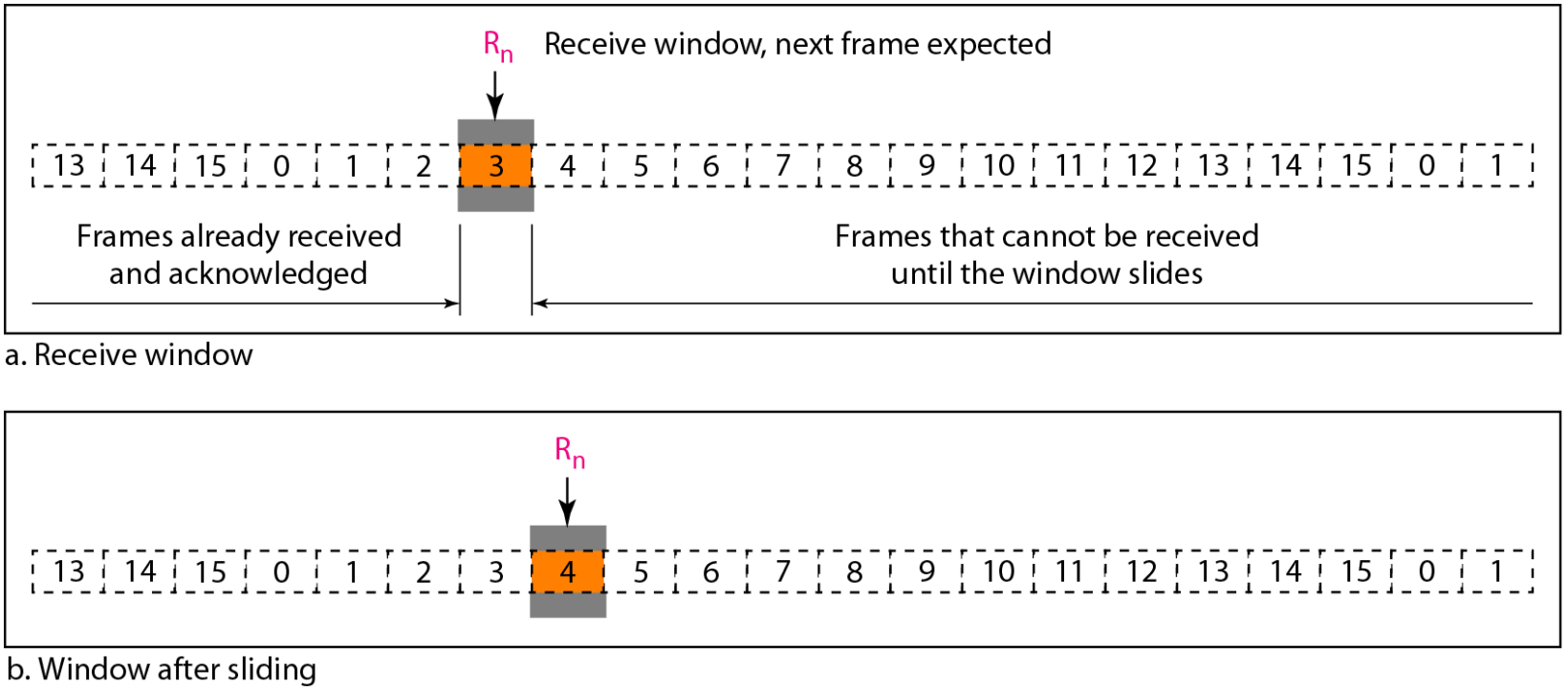
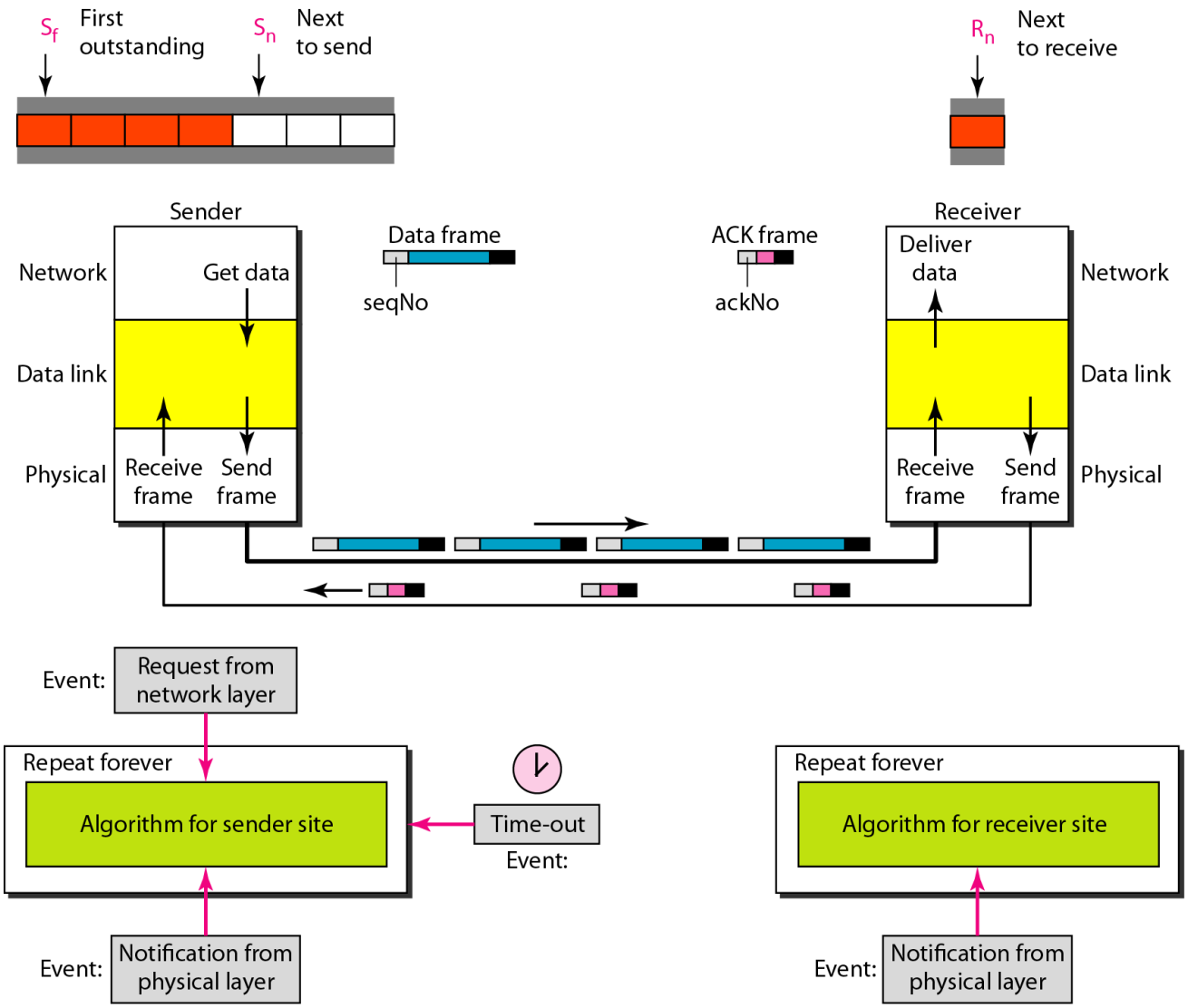


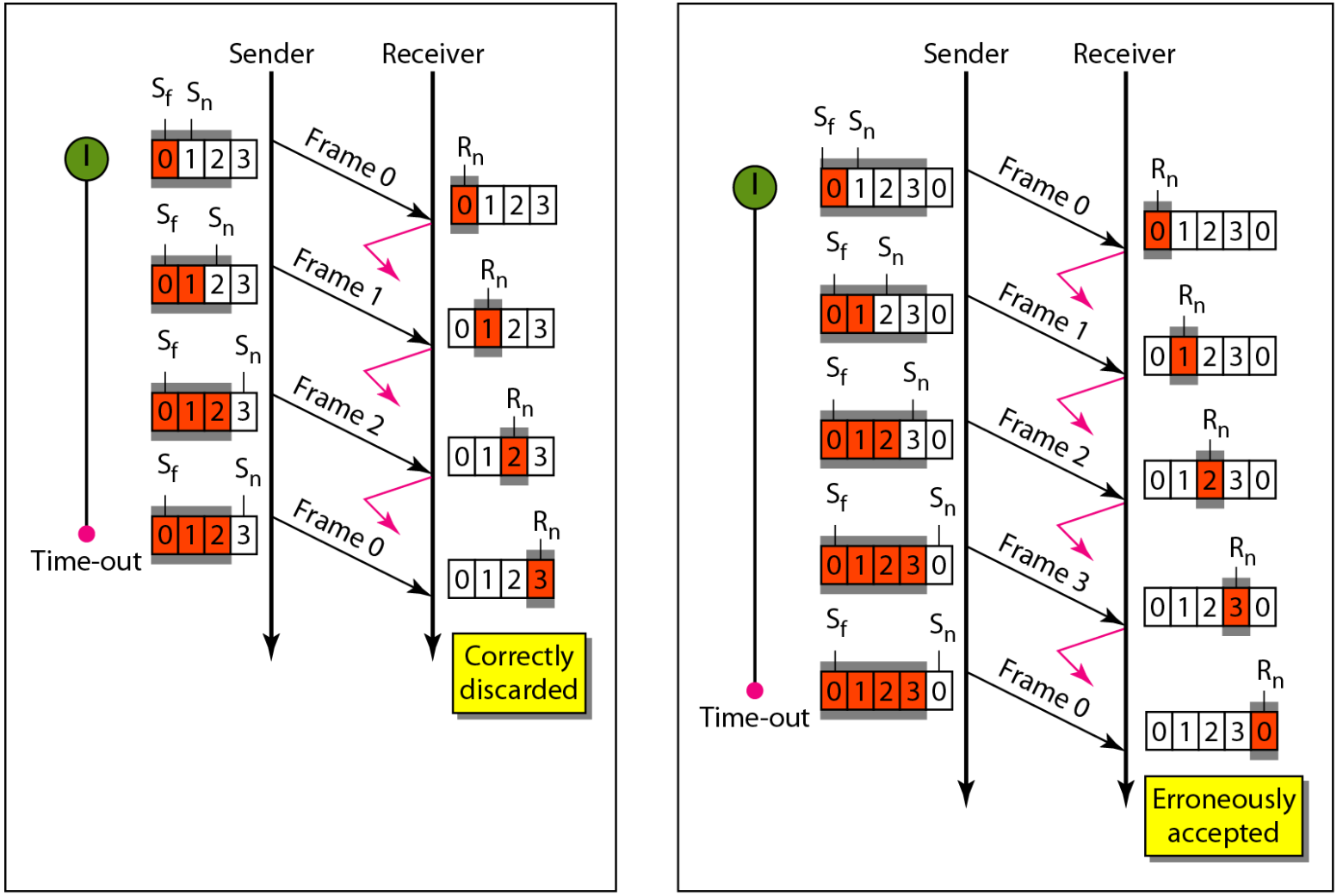
Figure 11.14 *Design of Go-Back-N ARQ*



Go-Back-N ARQ: Design

- Multiple frames can be in transit in the forward direction, and multiple acknowledgements in the reverse direction
  - Send window allows to have as many frames in transition as there are slots in it
- Timers: Only one timer used
- Acknowledgement: If a frame is damaged or received out of order, receiver discards all subsequent frames until it receives the one it is expecting
  - Need not have to acknowledge each frame received; can send one cumulative acknowledgement for several frames
- Resending a frame: When the timer expires, the sender resends all outstanding frames
  - E.g Sender sent 5 frames, but timer for frame 3 expires →sender goes back and sends frames 3, 4 and 5 again

Figure 11.15 *Window size for Go-Back-N ARQ*



a. Window size < 2^m

b. Window size = 2^m

Note

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

Algorithm 11.7 *Go-Back-N sender 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
```

(continued)

Algorithm 11.7 *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 }
```

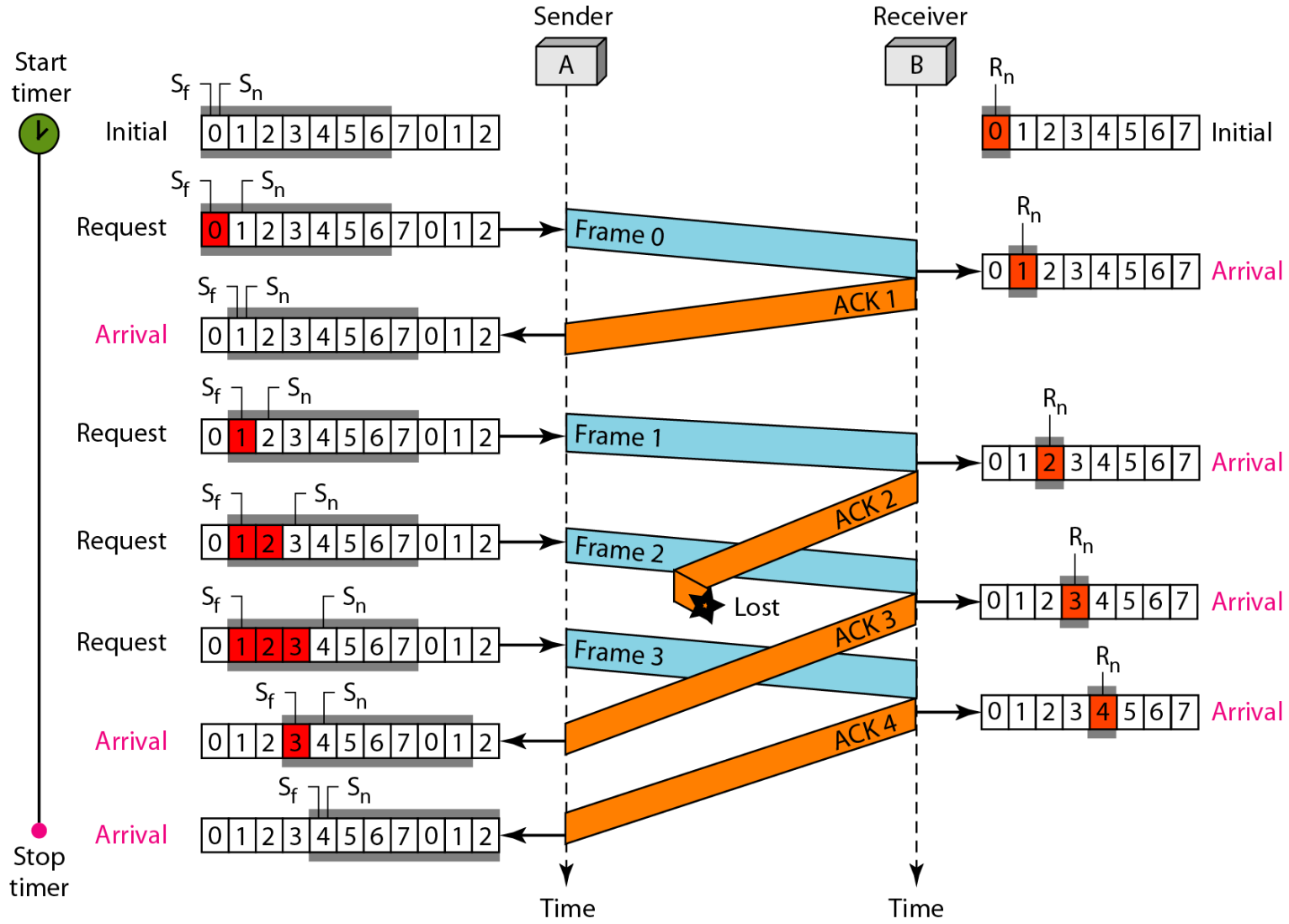
Algorithm 11.8 *Go-Back-N receiver algorithm*

```
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 }
```

Example 11.6

Figure 11.16 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.

Figure 11.16 *Flow diagram for Example 11.6*



Example 11.7

Figure 11.17 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.

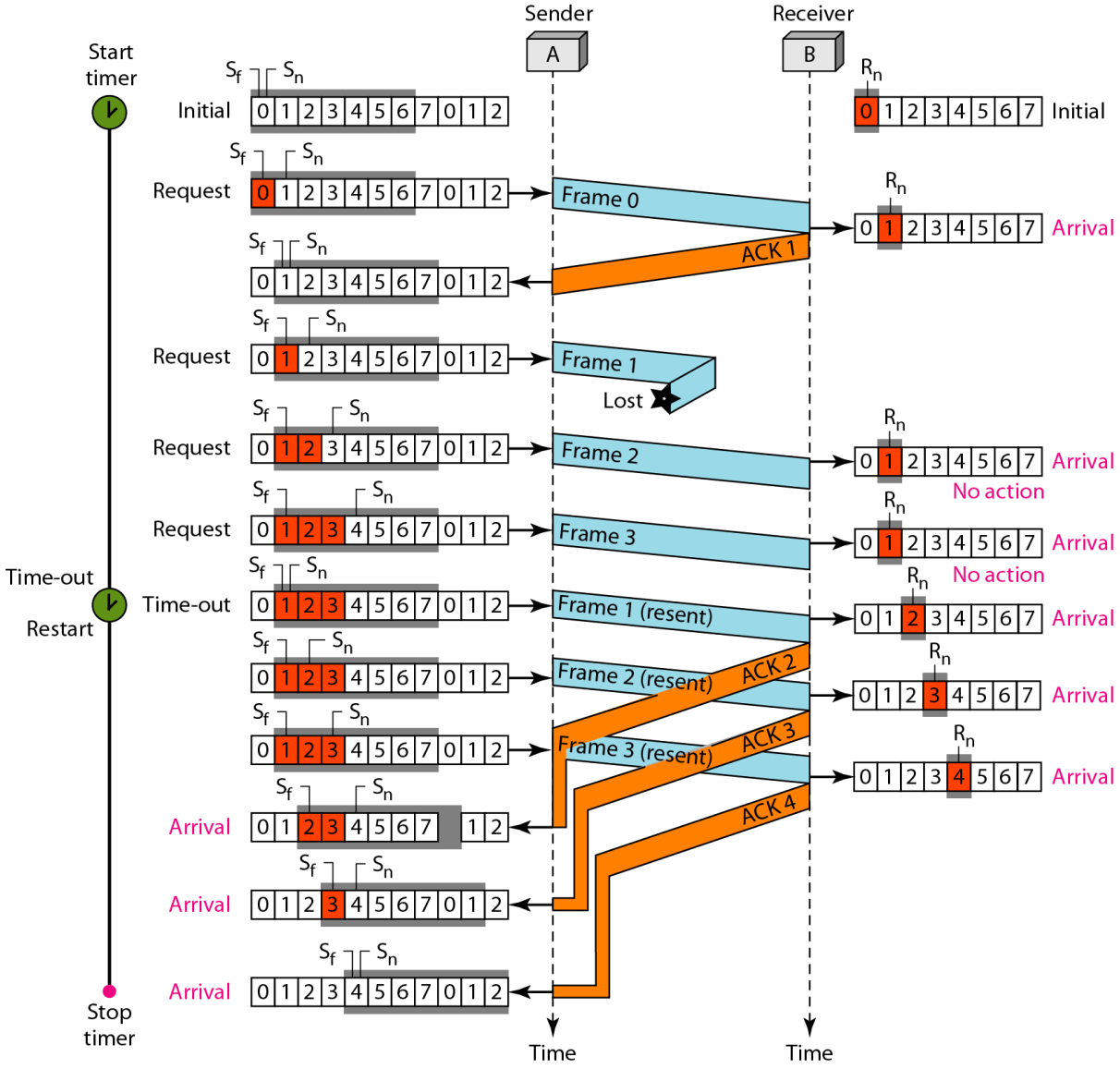
The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong.

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.

Example 11.7 (continued)

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.

Figure 11.17 *Flow diagram for Example 11.7*



Note

Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

Selective Repeat ARQ

- Go-back-N ARQ sends multiple frames when just one frame is damaged→ is inefficient
- The receiver keeps track of only one variable; out of order frames are simply discarded
  - There is no buffer to hold out of order frames
- In noisy channel frames have higher probability of damage; hence more frames need to be resent
  → consumes more BW and slows down transmission
- Selective Repeat ARQ: only damaged frame is resent; does not resend N frames
- Receiver operation is complex as compared to Go-back-N ARQ

Selective Repeat ARQ: Sequence Numbers and Sliding window

- Uses two windows; send window, and receive window
- Size of send window is 2^{m-1} , much smaller as compared to Go-back-N
  - If m=4, the seqNos are 0 to 15, but size of window is 8
  - Smaller window size means less (pipeline) efficiency; compensated by fewer duplicate frames
- Receive window size is same as send window
- Selective repeat allows as many frames as the size of receive window to arrive out of order
- Out of order frames are stored in a buffer, until there is a set of in-order frames to be delivered to the network layer
  - Receiver never delivers packets out of order to the network layer

Figure 11.18 *Send window for Selective Repeat ARQ*

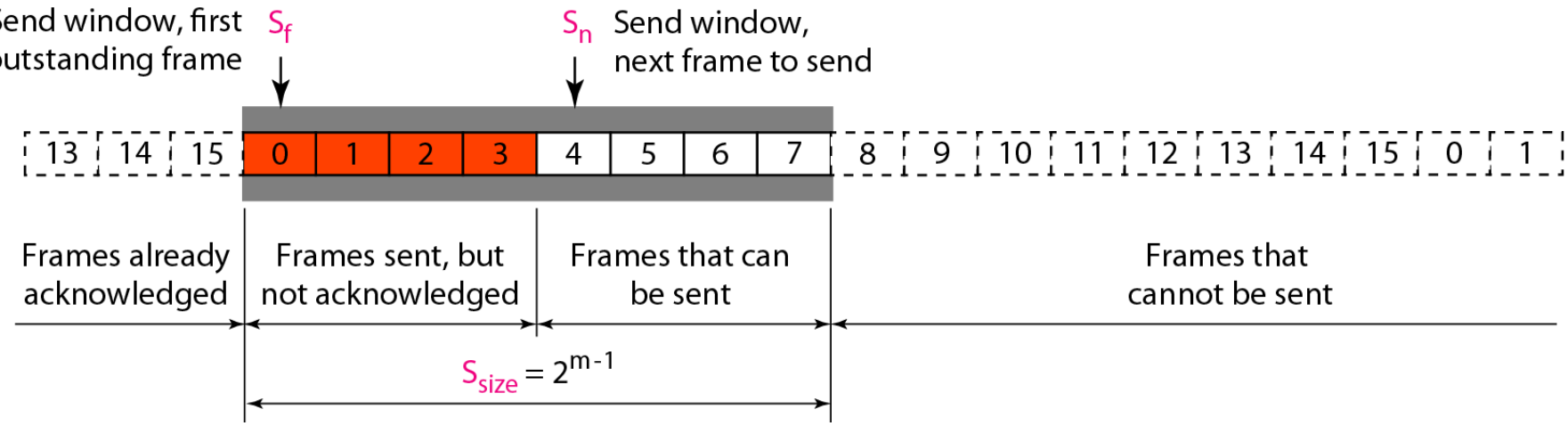
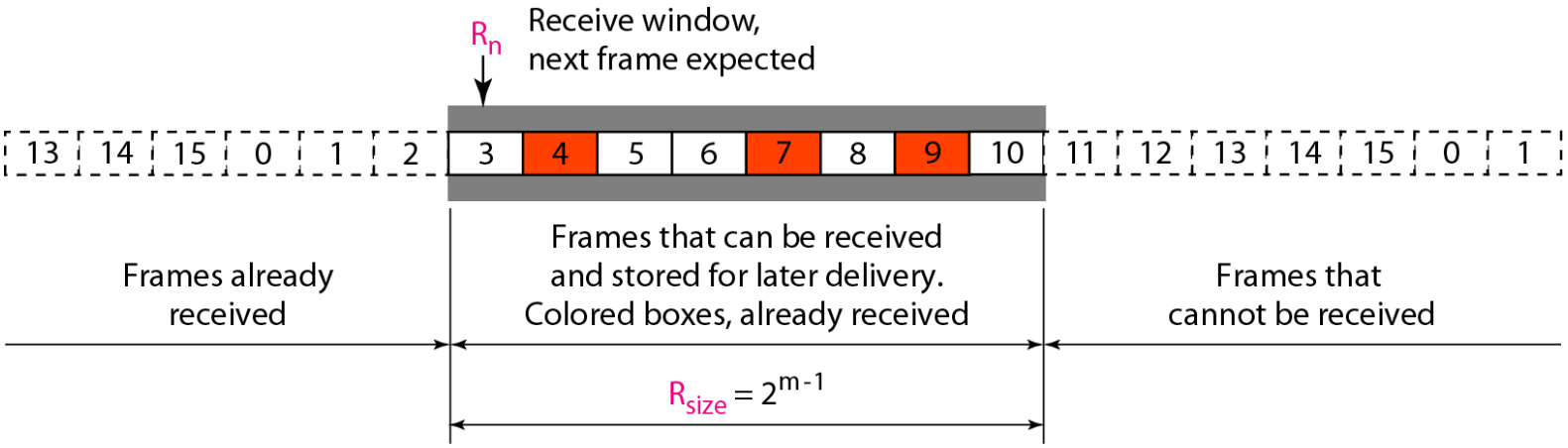


Figure 11.19 *Receive window for Selective Repeat ARQ*



Selective Repeat ARQ: Design

- Similar to Go-Back-N ARQ; Multiple frames can be in transit in the forward direction, and multiple acknowledgements in the reverse direction
- Timers: Multiple timers used; each per frame
- Acknowledgement: If a frame is received out of order, it is not discarded
- Resending a frame: When the timer for a frame expires, the sender resends that frame only
  - E.g Sender sent 5 frames, but timer for frame 3 expires →sender sends frame 3 again
  - Sender and Receiver window size must be at most one-half of 2^m

Figure 11.20 *Design of Selective Repeat ARQ*

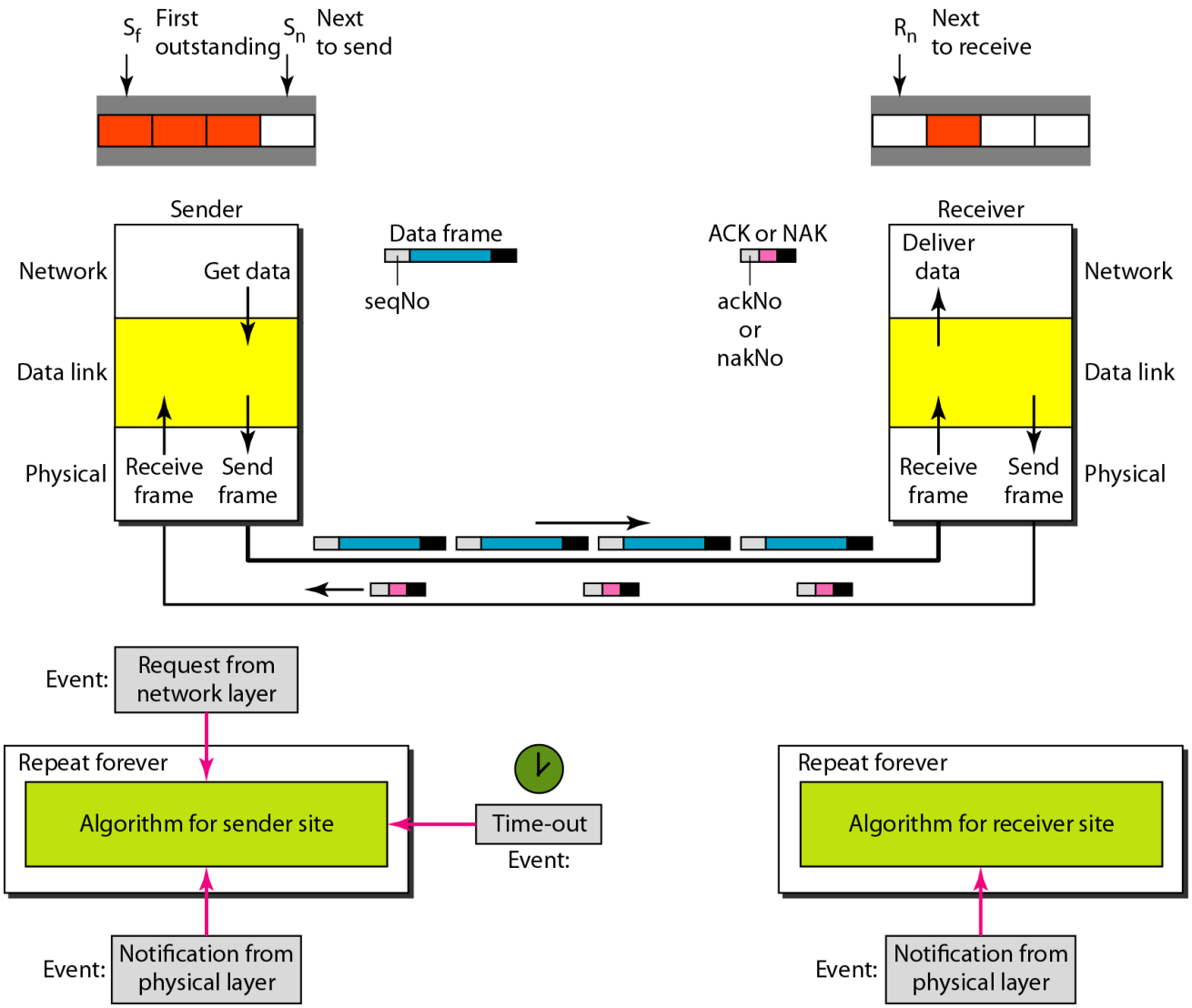
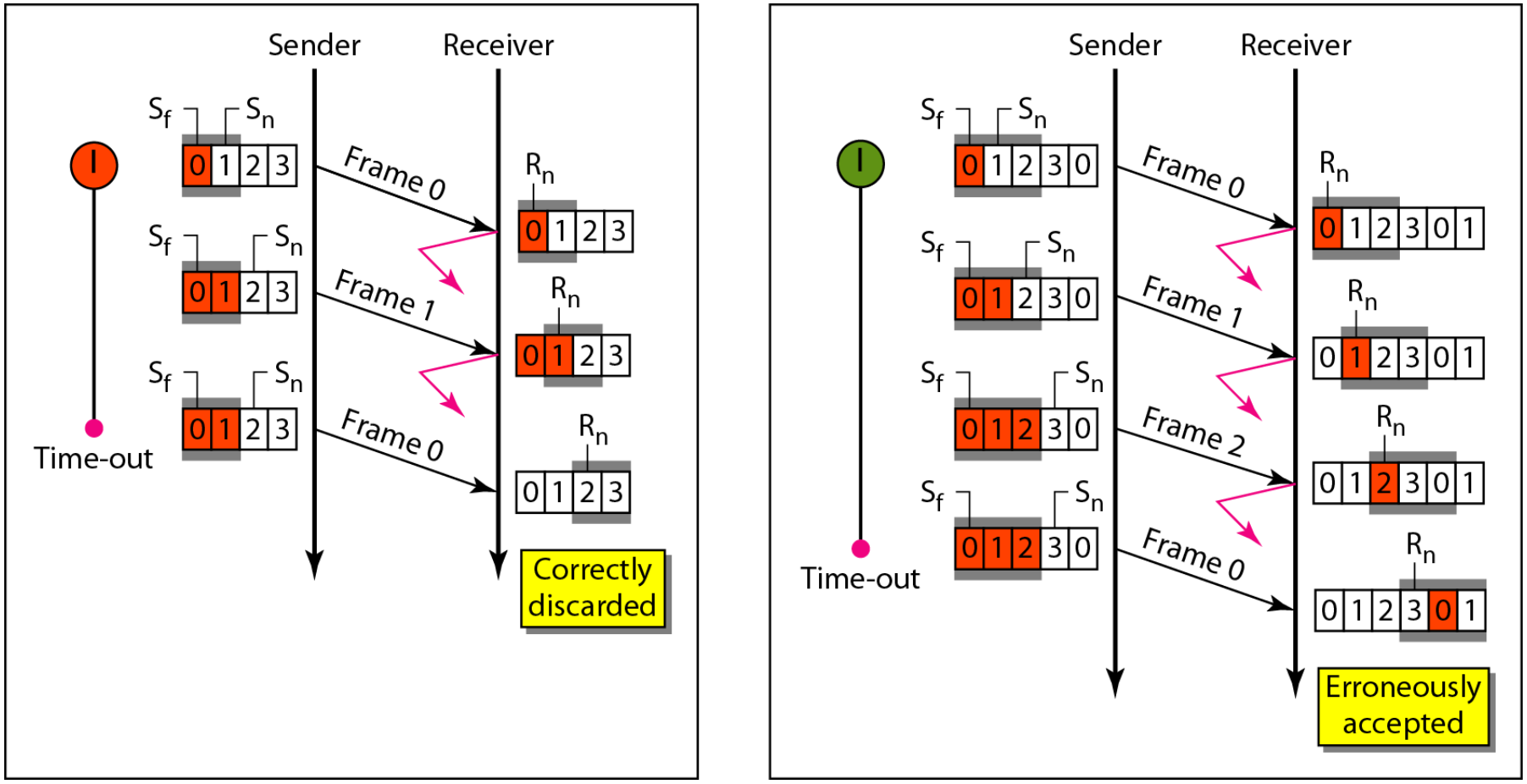


Figure 11.21 *Selective Repeat ARQ, window size*



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

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

Note

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

Algorithm 11.9 *Sender-site Selective Repeat 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
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
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  }
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

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

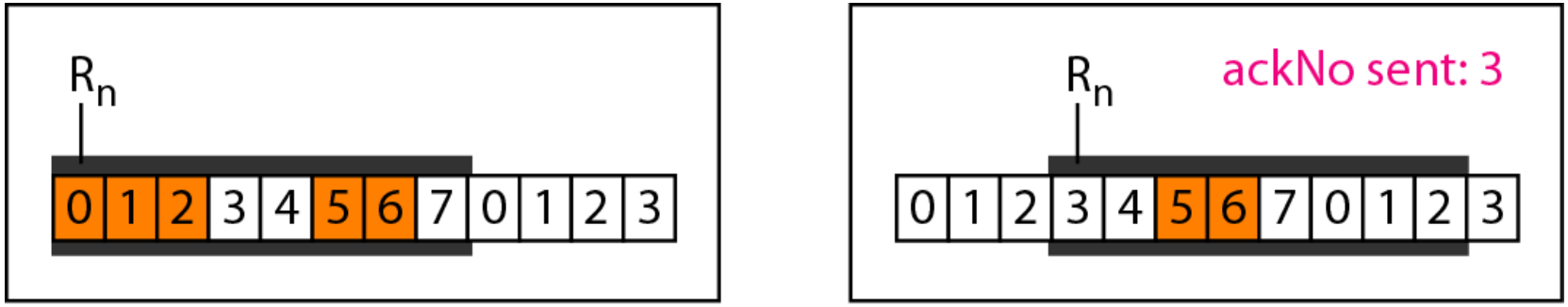
Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
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);
```

Algorithm 11.10 *Receiver-site Selective Repeat 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  }
```

Figure 11.22 *Delivery of data in Selective Repeat ARQ*



Example 11.8

This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 shows the situation.

One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3).

The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives.

The other two timers start when the corresponding frames are sent and stop at the last arrival event.

Example 11.8 (continued)

At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer.

At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer.

There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window.

Example 11.8 (continued)

Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same.

The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides.

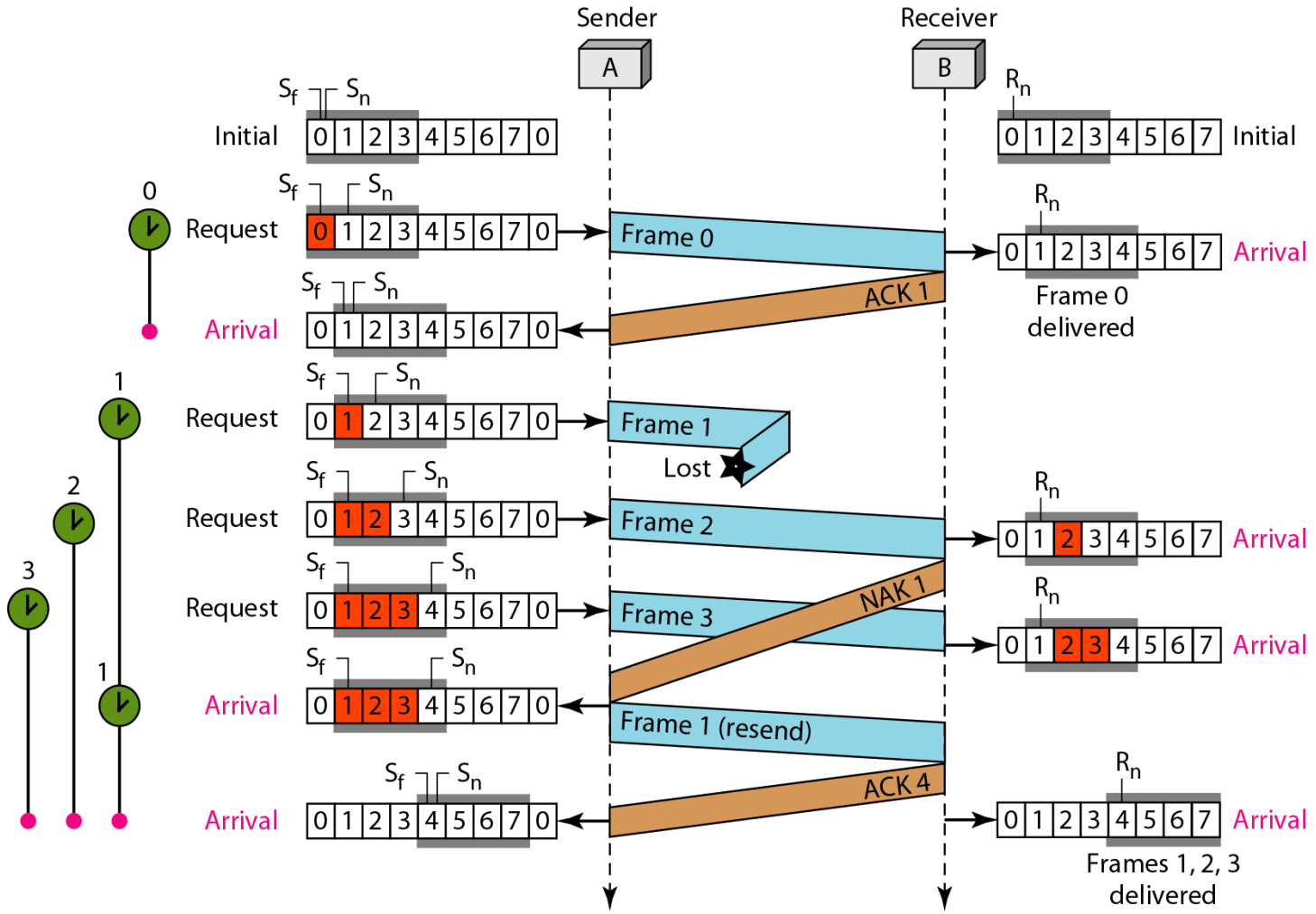
A NAK is sent once for each window position and defines the first slot in the window.

Example 11.8 (continued)

The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames.

In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.

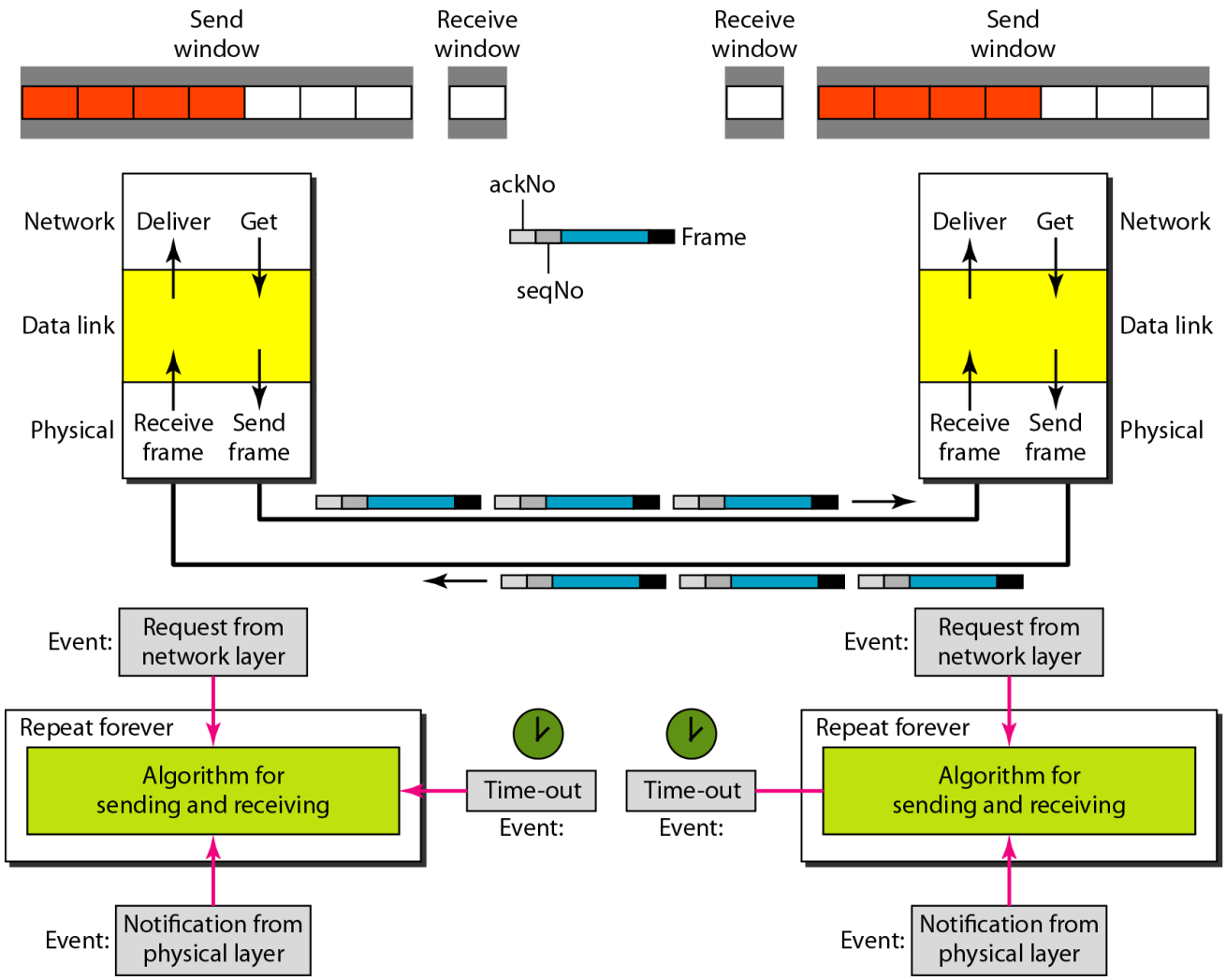
Figure 11.23 *Flow diagram for Example 11.8*



Piggybacking

- In real life the data frames flow in both directions; so the control information also needs to travel in both directions
- 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 to A (also other way round from B to A)
- ***Go-Back-N ARQ using piggybacking***
- Each node has 2 windows: 1 send and 1 receive window
- Both are involved in 3 types of events: request, arrival and timeout
- Request event uses only the send window at each site; arrival event needs to use both windows (since when a frame arrives, node needs to handle control info as well as the frame itself)

Figure 11.24 *Design of piggybacking in Go-Back-N ARQ*



Module 2: Data Link and MAC Layer

Part-III Multiple Access (Medium Access Control)

Dr. Prasanna Shete
Dept. of Computer Engineering
K. J. Somaiya College of Engineering

Slide Source: B. A. Forauzan, Data Communications and Networking, McGraw-Hill Online Learning Centre
http://highered.mheducation.com/sites/0072967757/information_center_view0/index.html

Figure 12.1 *Data link layer divided into two functionality-oriented sublayers*

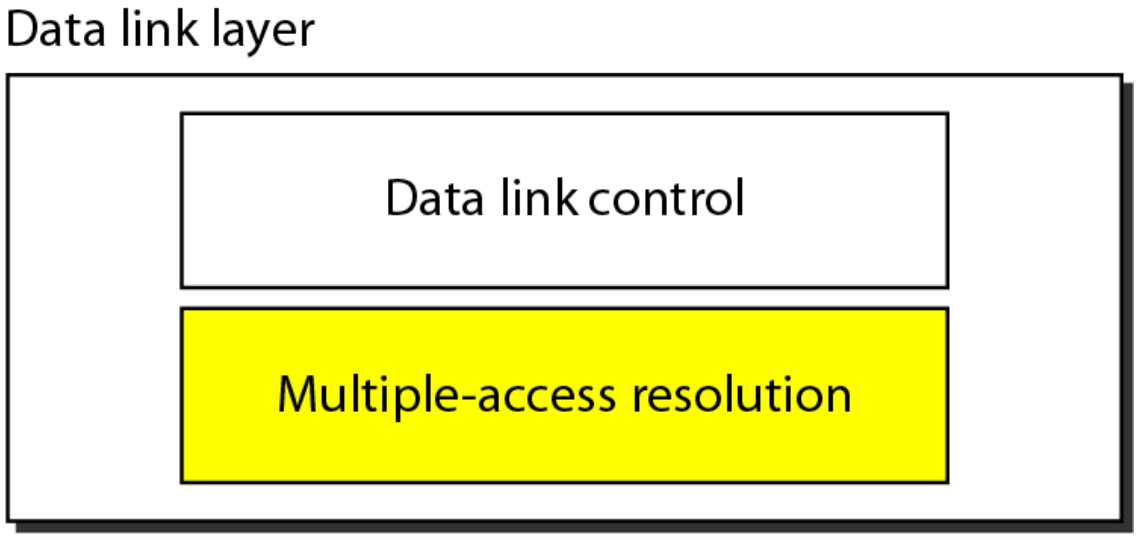
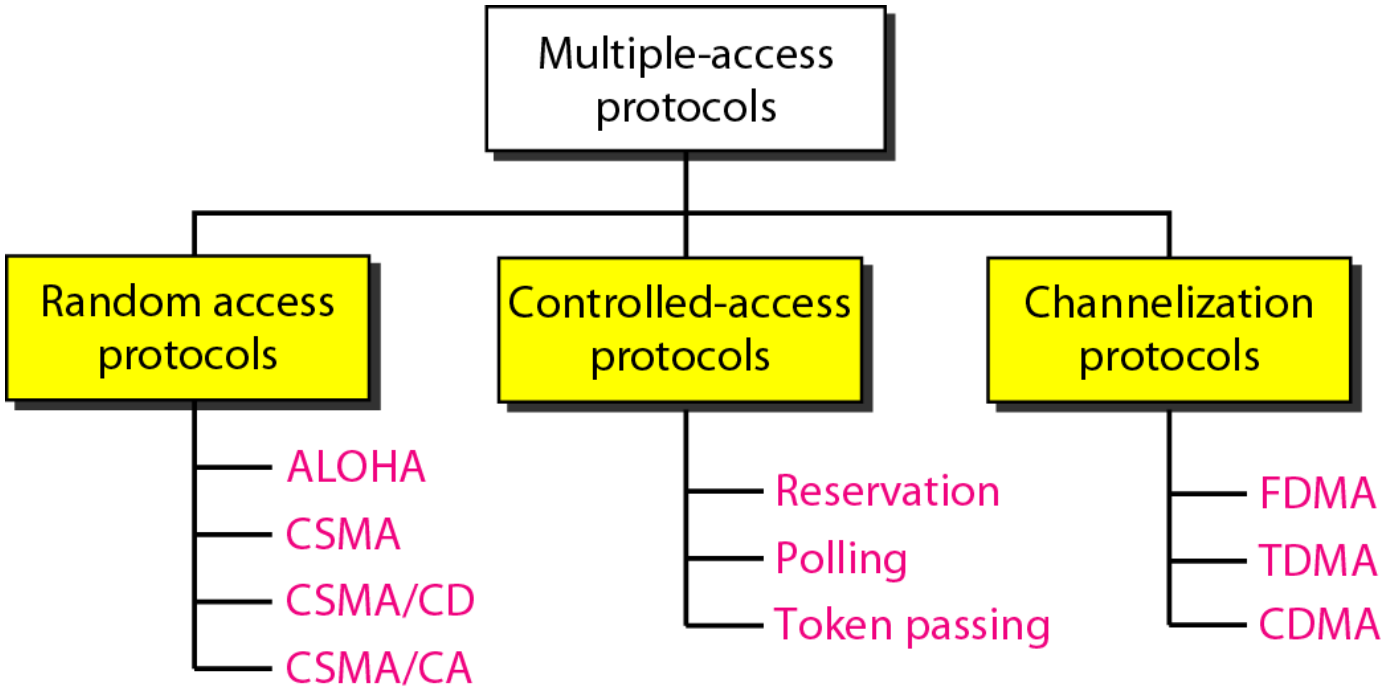


Figure 12.2 *Taxonomy of multiple-access protocols discussed in this chapter*



12-1 RANDOM ACCESS

In random access or contention methods, no station is superior to another station and none is assigned the control over another.

No station permits, or does not permit, another station to send.

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.

Topics discussed in this section:

ALOHA

Carrier Sense Multiple Access

Carrier Sense Multiple Access with Collision Detection

Carrier Sense Multiple Access with Collision Avoidance

ALOHA

- Earliest random access method developed at University of Hawaii → early 70's
- Simple protocol, Each station sends a frame whenever it has a frame to send
- Possibility of collision between frames from different stations (due to shared channel)

Figure 12.3 *Frames in a pure ALOHA network*

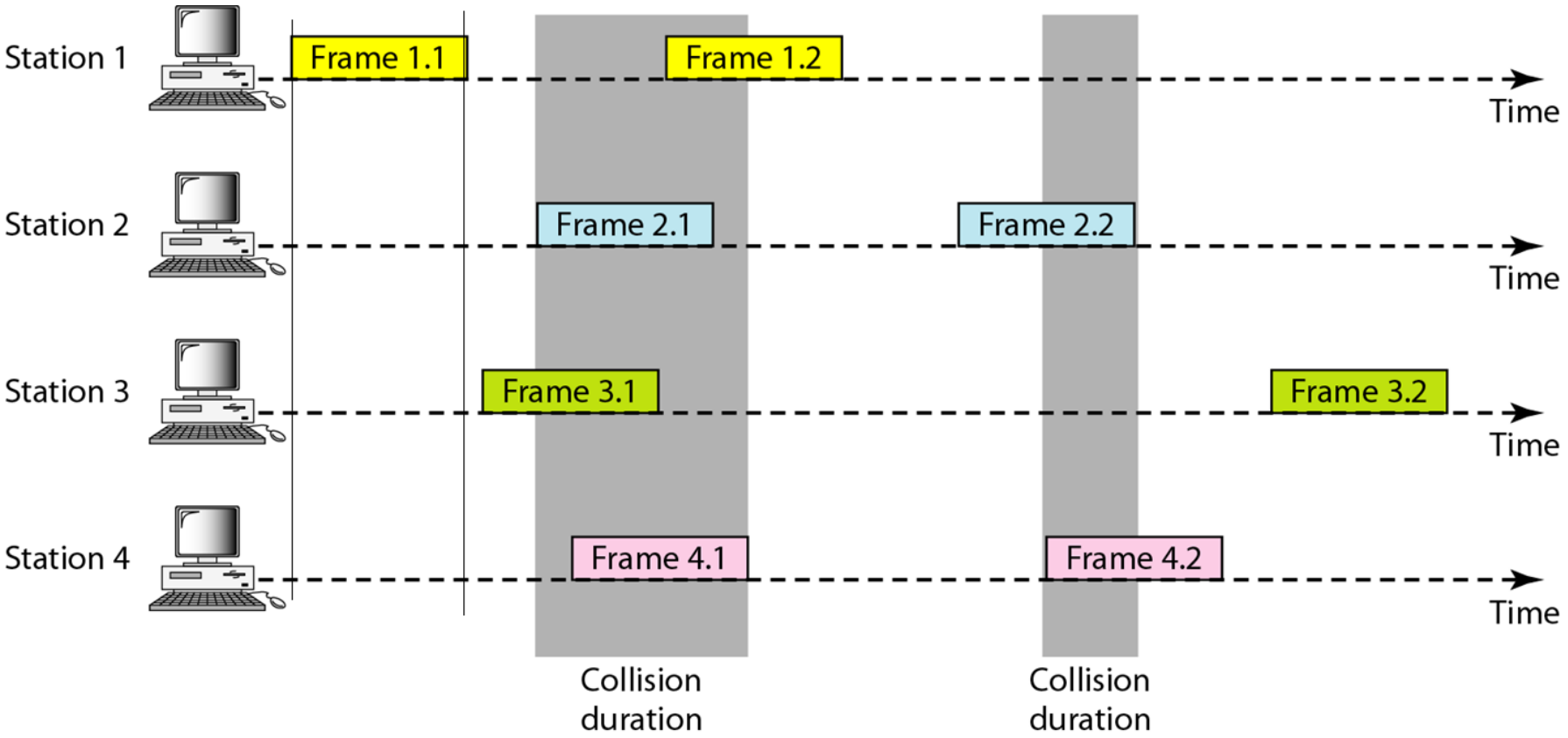
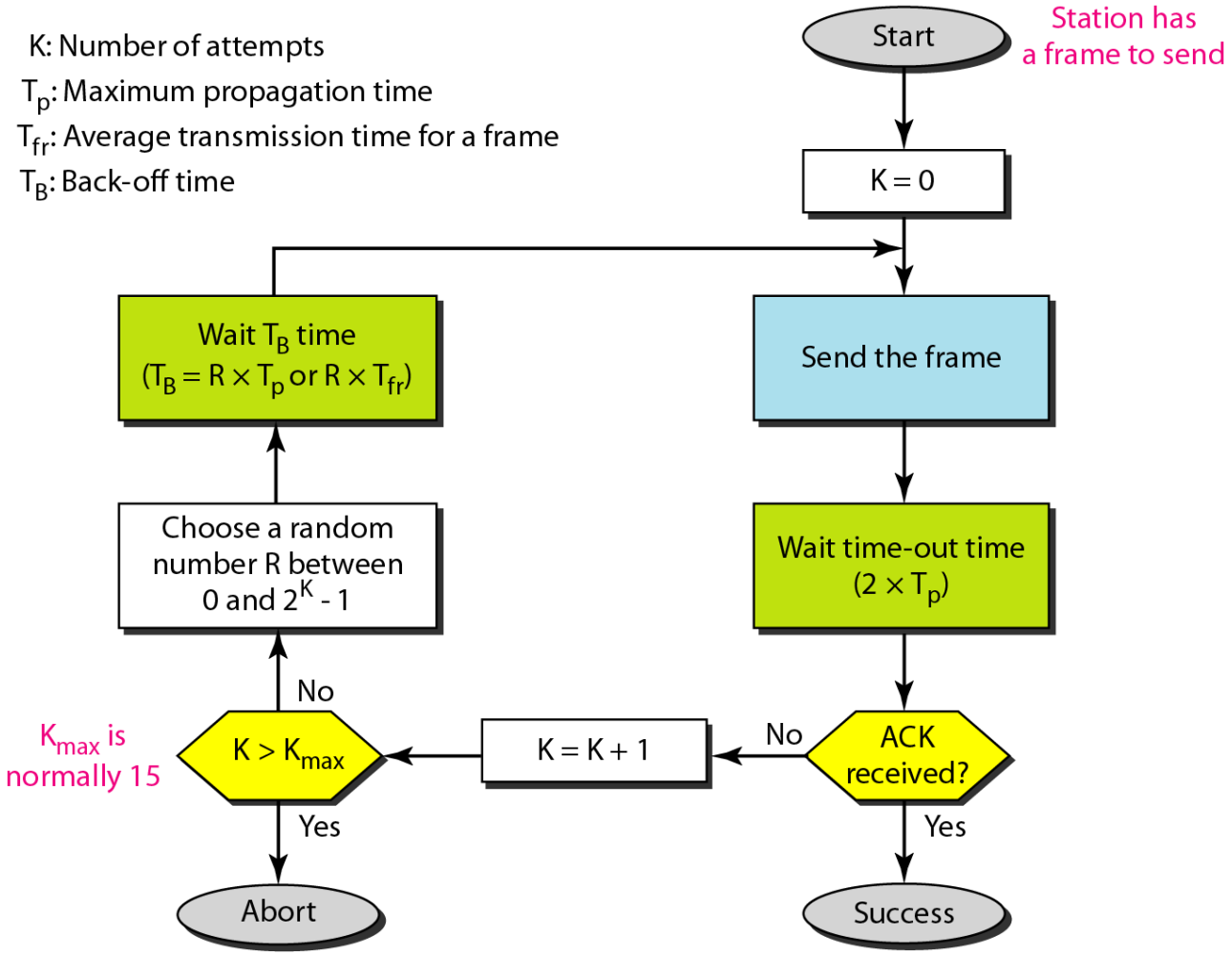


Figure 12.4 *Procedure for pure ALOHA protocol*



Example 12.1

The stations on a wireless ALOHA network are a maximum of 600 km apart. If we assume that signals propagate at 3×10^8 m/s, find the value of T_B for different values of K

→Propagation delay is

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

a. For $K = 1$, the range is {0, 1}. The station needs to generate a random number with a value of 0 or 1. This means that T_B is either 0 ms (0 × 2) or 2 ms (1 × 2), based on the outcome of the random variable.

Example 12.1 (continued)

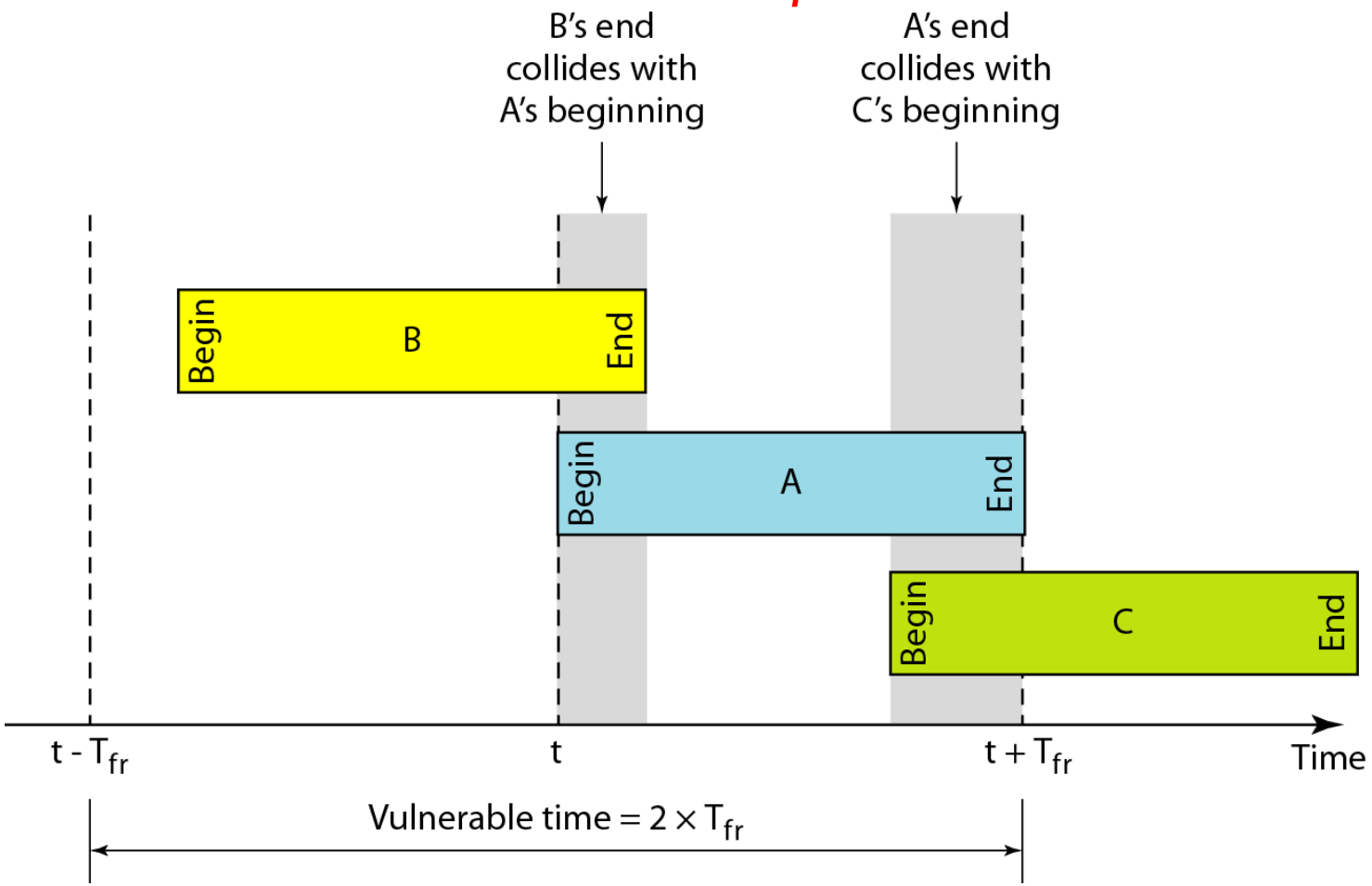
b. For $K = 2$, the range is $\{0, 1, 2, 3\}$. This means that T_B can be 0, 2, 4, or 6 ms, based on the outcome of the random variable.

c. For $K = 3$, the range is $\{0, 1, 2, 3, 4, 5, 6, 7\}$. This means that T_B can be 0, 2, 4, . . . , 14 ms, based on the outcome of the random variable.

We need to mention that if $K > 10$, it is normally set to 10.

Figure 12.5 *Vulnerable time for pure ALOHA protocol*

- *Vulnerable time*: Length of time in which there is a possibility of collision; for Pure ALOHA it $2*T_{fr}$



Example 12.2

A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the requirement to make this frame collision-free?

Solution

Average frame transmission time T_{fr} *=200 bits/200 kbps =1 ms.*

The vulnerable time = $2 \times T_{fr} = 2 \times 1$ *ms = 2 ms.*

This means no station should send later than 1 ms before this station starts transmission and no station should start sending during the one 1-ms period that this station is sending.

Throughput of ALOHA

- If '***G***' be the average number of frames generated by the system during one frame transmission time
- Then, average number of successful transmissions for ALOHA is given as: $\mathbf{S = G \times e^{-2G}}$
- The maximum throughput, $\mathbf{S_{max}}$ is 0.184 for $\mathbf{G = ½}$
- i.e. If one-half a frame is generated during 1 frame transmission time (or 1 frame during two frame transmission times), then 18.4% of these reach the destination successfully

Note

The throughput for pure ALOHA is

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

The maximum throughput

$$S_{max} = 0.184$$ **when G= (1/2).**

Example 12.3

A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the throughput if the system (all stations together) produces

a. 1000 frames per second b. 500 frames per second

c. 250 frames per second.

Solution

***Frame transmission time* = 200/200 kbps or 1 ms**

a. If system creates 1000 frames per second,

i.e.1 frame per millisecond → load is 1

$S = G \times e^{-2G} = 0.135$ (13.5 percent)

Throughput = $1000 \times 0.135 = 135$ frames/sec

This means only135 frames out of 1000 will probably survive.

Example 12.3 (continued)

b. If the system creates 500 frames per second, i.e. (1/2) frame per millisecond. The load is (1/2).

In this case $S = G \times e^{-2G}$ or $S = 0.184$ (18.4 percent)

This means that the throughput is $500 \times 0.184 = 92$

That is only 92 frames out of 500 will probably survive.

Note that this is the maximum throughput case, percentagewise

c. If the system creates 250 frames per second, this is (1/4) frame per millisecond. The load is (1/4). In this case $S = G \times e^{-2G}$ or $S = 0.152$ (15.2 percent)

This means that the throughput is $250 \times 0.152 = 38$

Only 38 frames out of 250 will probably survive.

Slotted ALOHA

- Time is divided into slots of T_{fr} *sec*
- Stations can only send at the beginning of the time slot
- If a station misses the beginning of a particular time slot, it must wait until the beginning of next time slot
- Possibility of collision, if 2 stations try to send at the beginning of the same slot

- Vulnerable time is reduced to one-half that of pure ALOHA = T_{fr}

Figure 12.6 *Frames in a slotted ALOHA network*

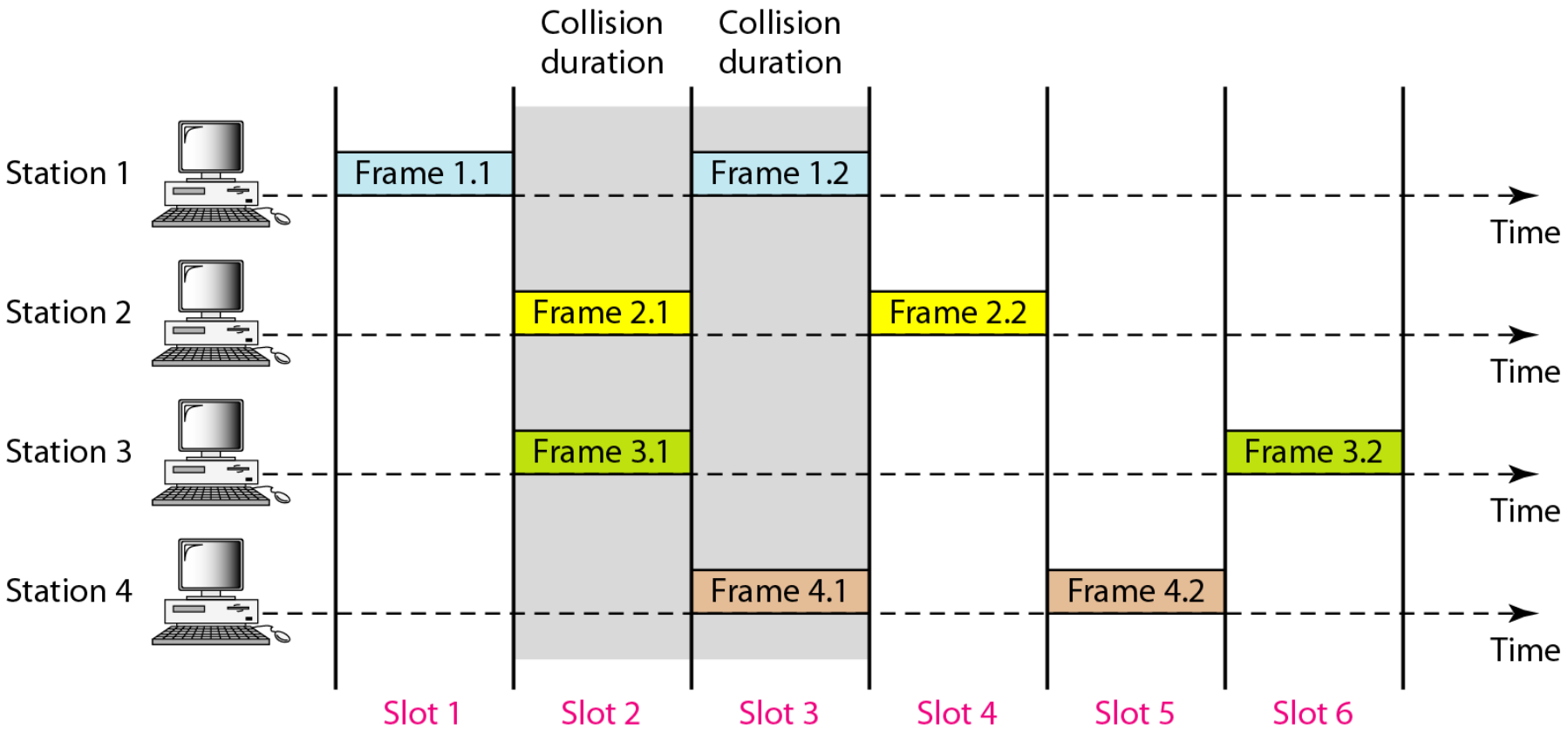
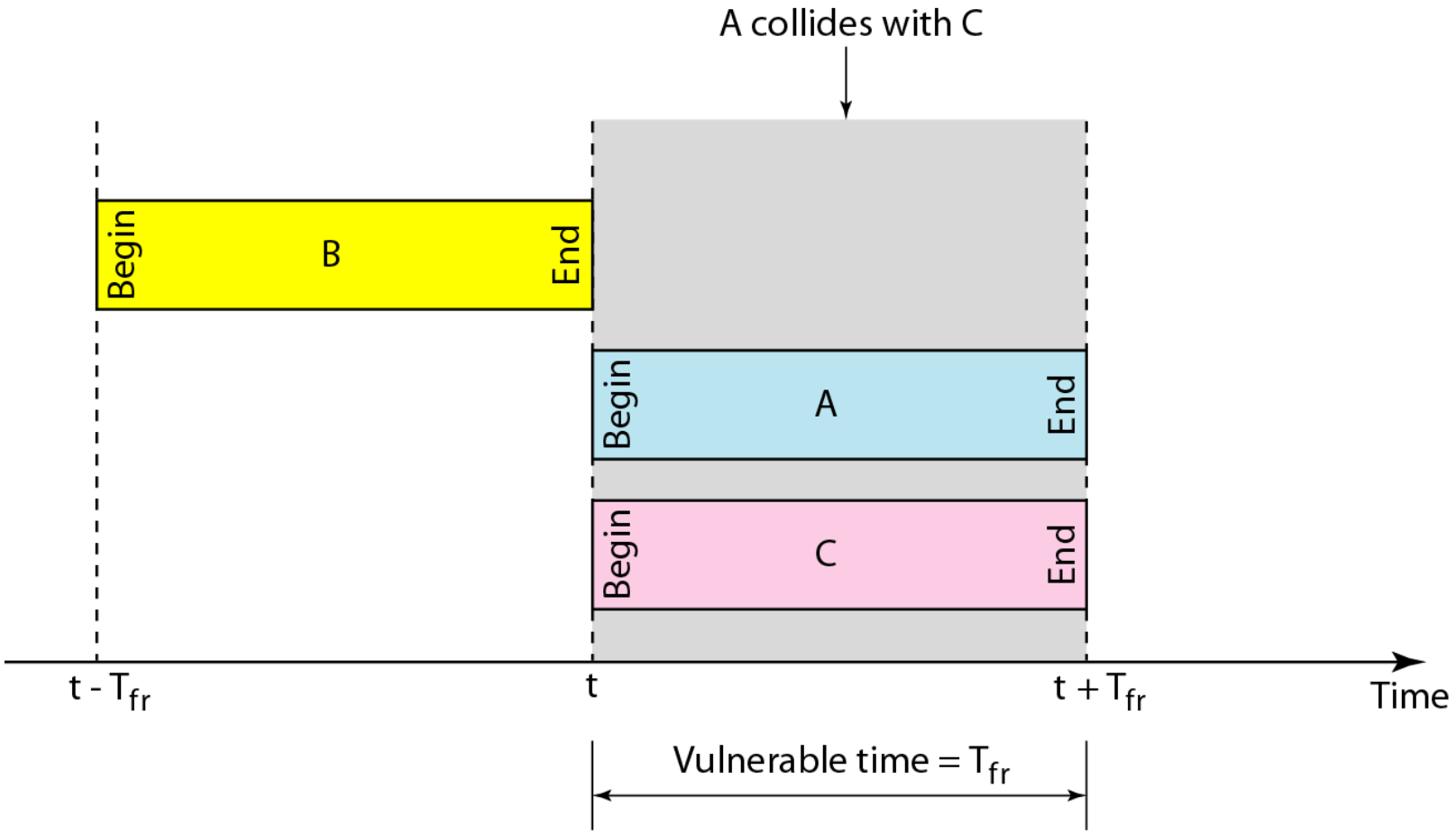


Figure 12.7 *Vulnerable time for slotted ALOHA protocol*



- Vulnerable time for slotted ALOHA is T_{fr}

Note

The throughput for slotted ALOHA is $S = G \times e^{-G}$.

The maximum throughput $S_{max} = 0.368$ when G = 1.

Example 12.4

A slotted ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the throughput if the system (all stations together) produces

a. 1000 frames per second b. 500 frames per second

c. 250 frames per second.

Solution

Frame transmission time = 200/200 kbps or 1 ms.

a. If the system creates 1000 frames per second, this is 1 frame per millisecond. The load is 1.

→ $S = G \times e^{-G}$ or S = 0.368 (36.8 percent)

Throughput = 1000 × 0.368 = 368 frames

i.e. Only 386 frames out of 1000 will probably survive.

Example 12.4 (continued)

b. If the system creates 500 frames per second, this is (1/2) frame per millisecond. The load is (1/2). In this case $S = G \times e^{-G}$ or $S = 0.303$ (30.3 percent). This means that the throughput is $500 \times 0.303 = 151$. Only 151 frames out of 500 will probably survive.

c. If the system creates 250 frames per second, this is (1/4) frame per millisecond. The load is (1/4). In this case $S = G \times e^{-G}$ or $S = 0.195$ (19.5 percent). This means that the throughput is $250 \times 0.195 = 49$.
Only 49 frames out of 250 will probably survive.

Carrier Sense Multiple Access (CSMA)

- Proposed to minimize the chances of collision and thus improve the performance
- Principle: Sense the medium before using it; "Listen before talk"
  - In CSMA, each station first checks the state of the medium before sending
  - If medium idle → transmit; busy medium → wait
- CSMA reduces the possibility of collisions, but cannot completely eliminate it
  - Possibility of collision exists due to propagation delay
  - *a sensing station may find medium to be idle only because the first bit sent by another station has not yet been received*

Figure 12.8 *Space/time model of the collision in CSMA*

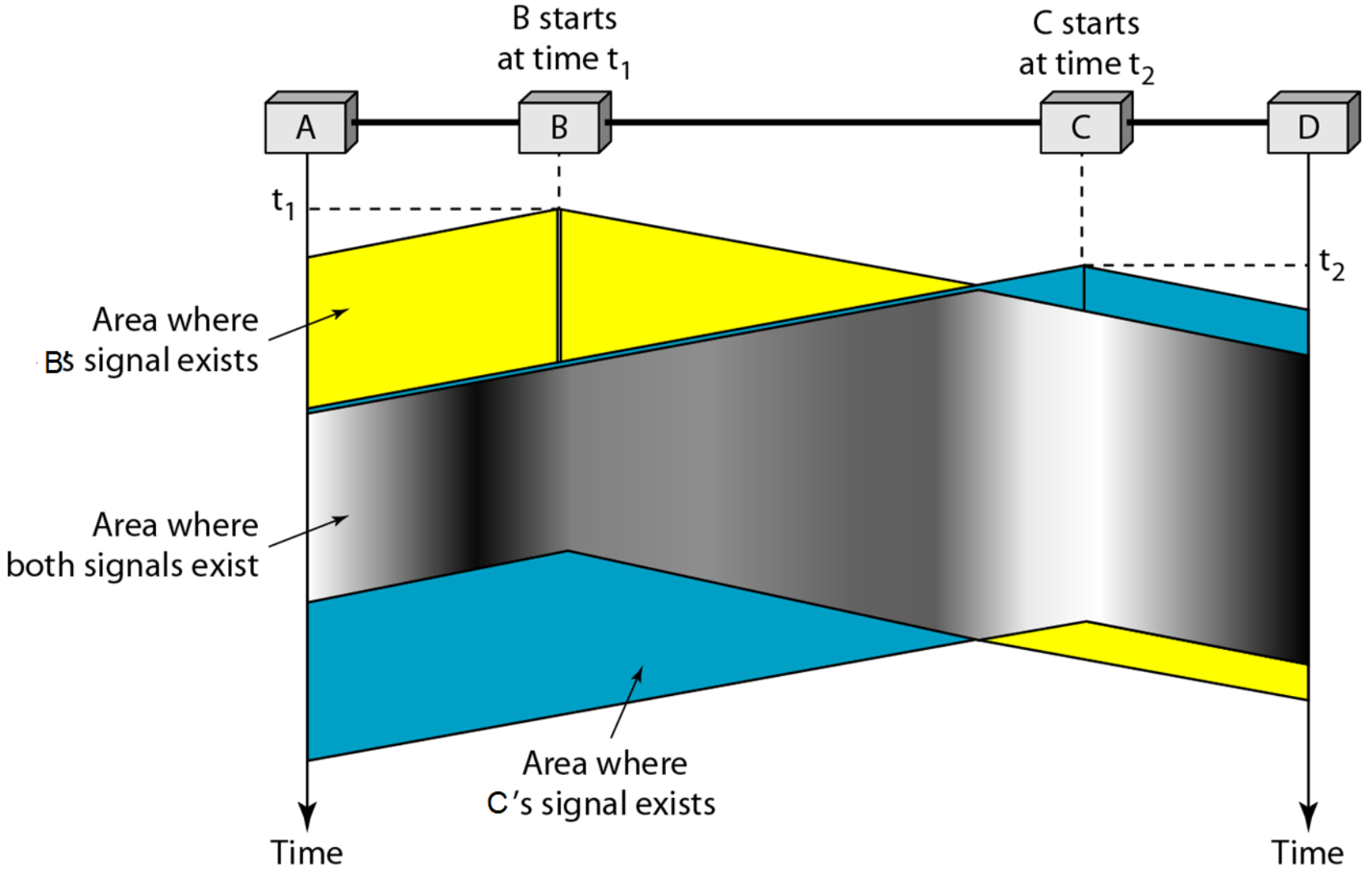
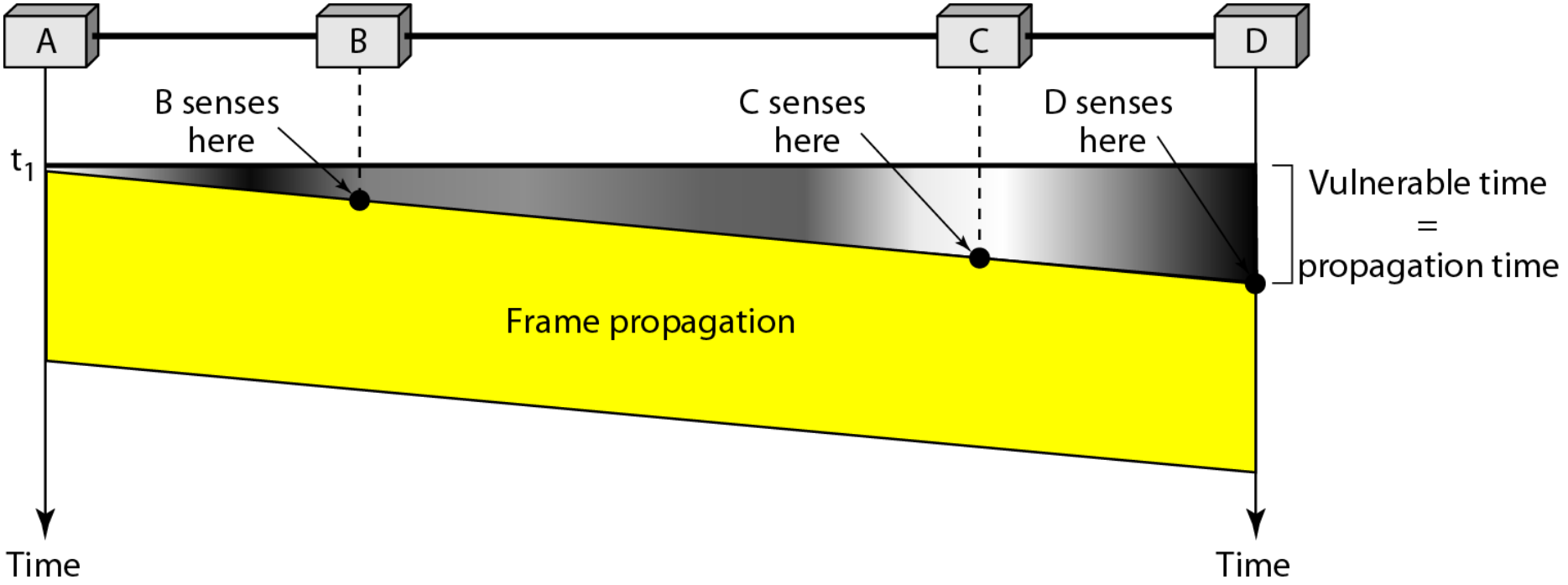


Figure 12.9 *Vulnerable time in CSMA*



- Vulnerable time for CSMA is the propagation time T_p

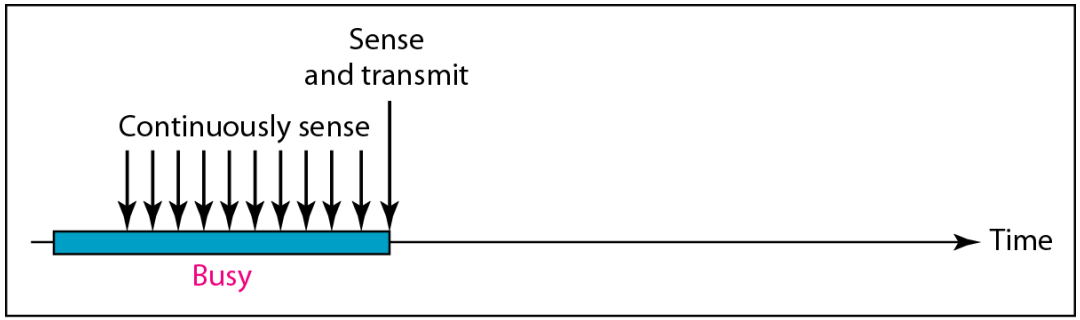
Carrier Sense: Persistence Methods

- These methods tell what should a station do if the channel is busy or idle
- ***1-persistent:***
- If a station senses channel is idle, it sends its frame immediately with probability '1'
  - Highest chance of collision, since 2 or more stations may find channel is idle and send their frames at once
- ***Nonpersistent:***
- If the channel is idle the station sends immediately; else waits random amount of time and senses the channel again
  - Reduces the chance of collision; its unlikely that 2 or more stations will wait same amount of time and retry to send

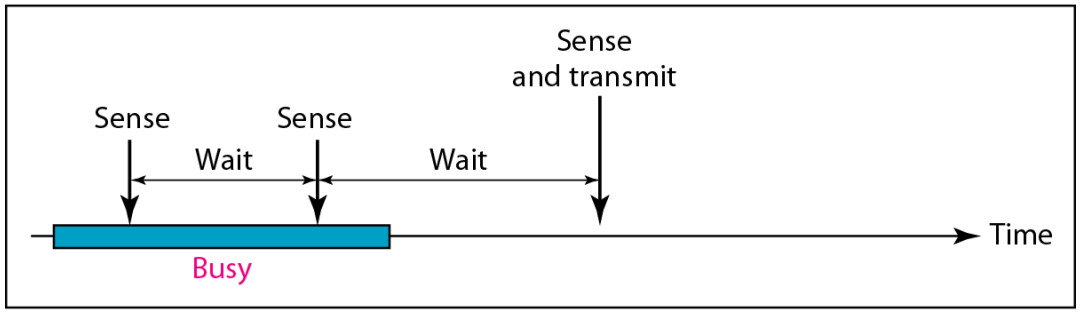
Carrier Sense: Persistence Methods contd…

- ***p-persistent***:
- Combines advantages of other 2 approaches to reduce the chance of collision and improves efficiency
- Used if the channel has time slots with slot duration => maximum propagation time
- After the station finds the channel is idle, it takes the following steps:
  1. **With probability *p*, the station sends its frame**
  2. **With probability *q* = (1 - *p*), the station waits for the beginning of next time slot and senses the channel again**
  3. **If line idle; goes to step 1**
  4. **If line busy, it acts as a collision has occurred, and uses the back-off procedure**

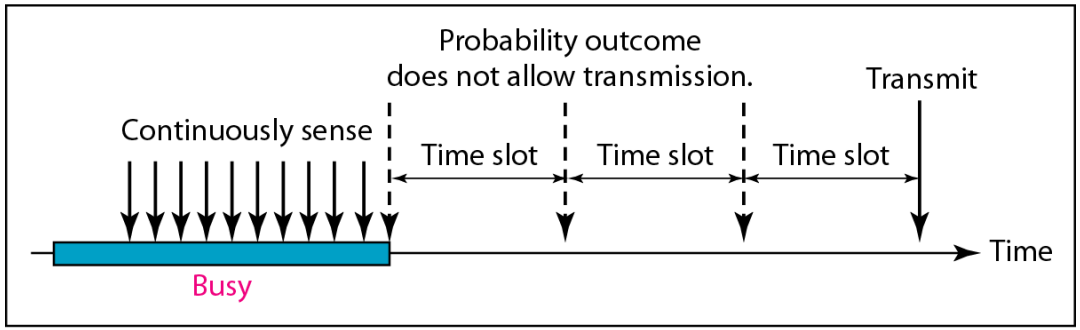
Persistence Methods: Figure 12.10 *Behavior of three persistence methods*



a. 1-persistent

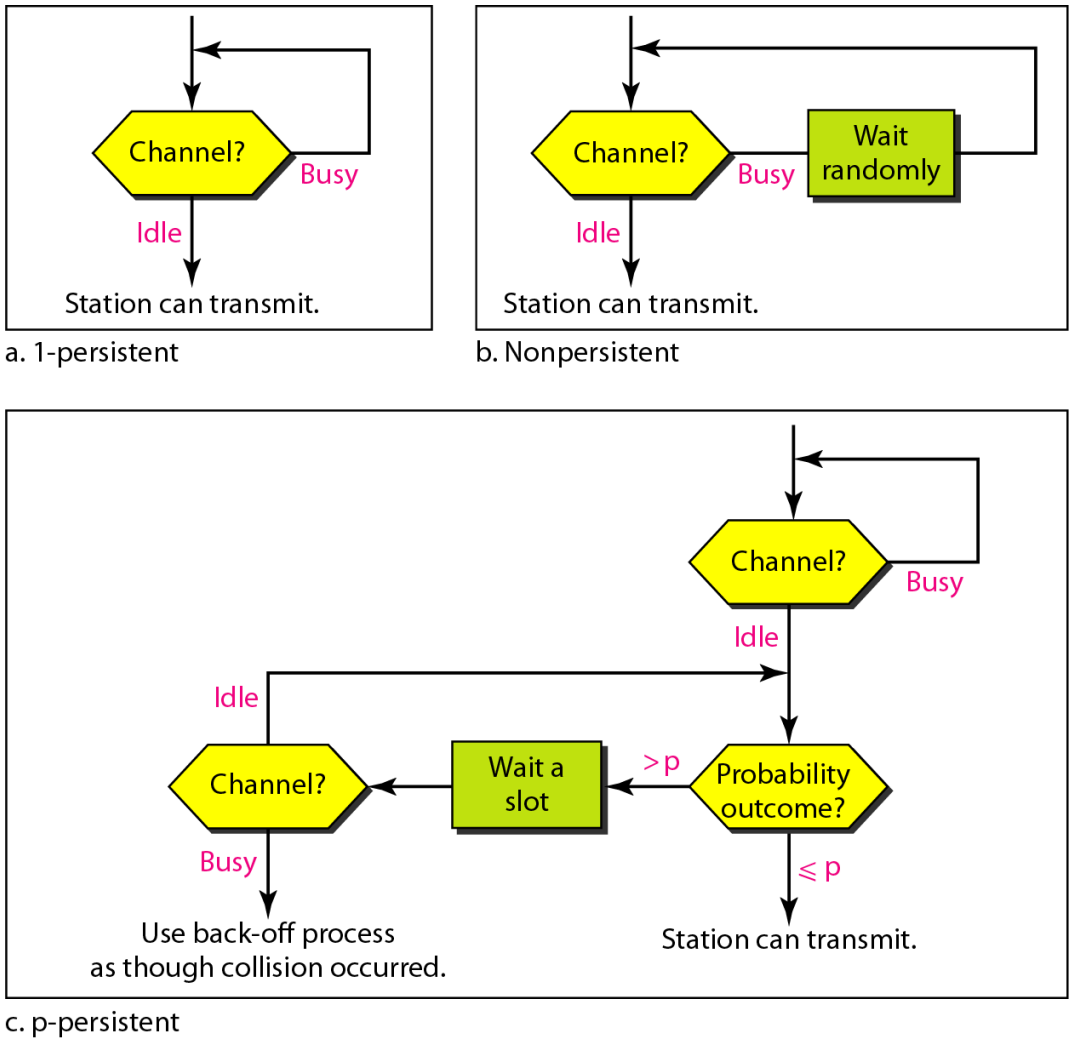


b. Nonpersistent



c. p-persistent

Figure 12.11 *Flow diagram for three persistence methods*

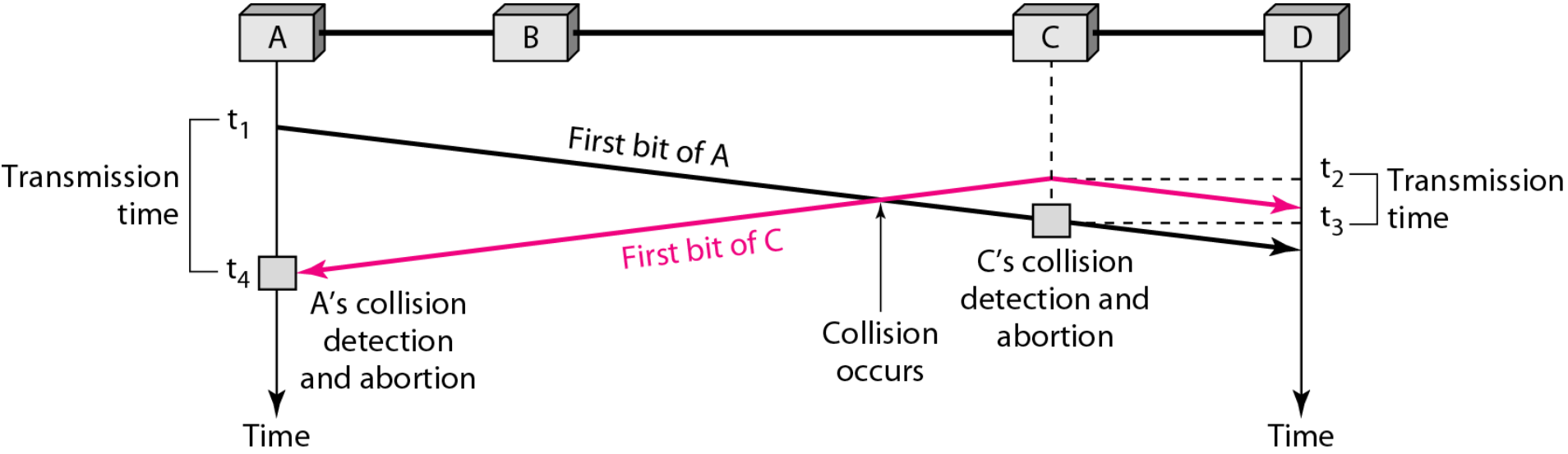


Carrier Sense Multiple Access with Collision Detection(CSMA/CD)

- CSMA does not specify what to do after collision
- CSMA/CD augments the CSMA algorithm to handle collision

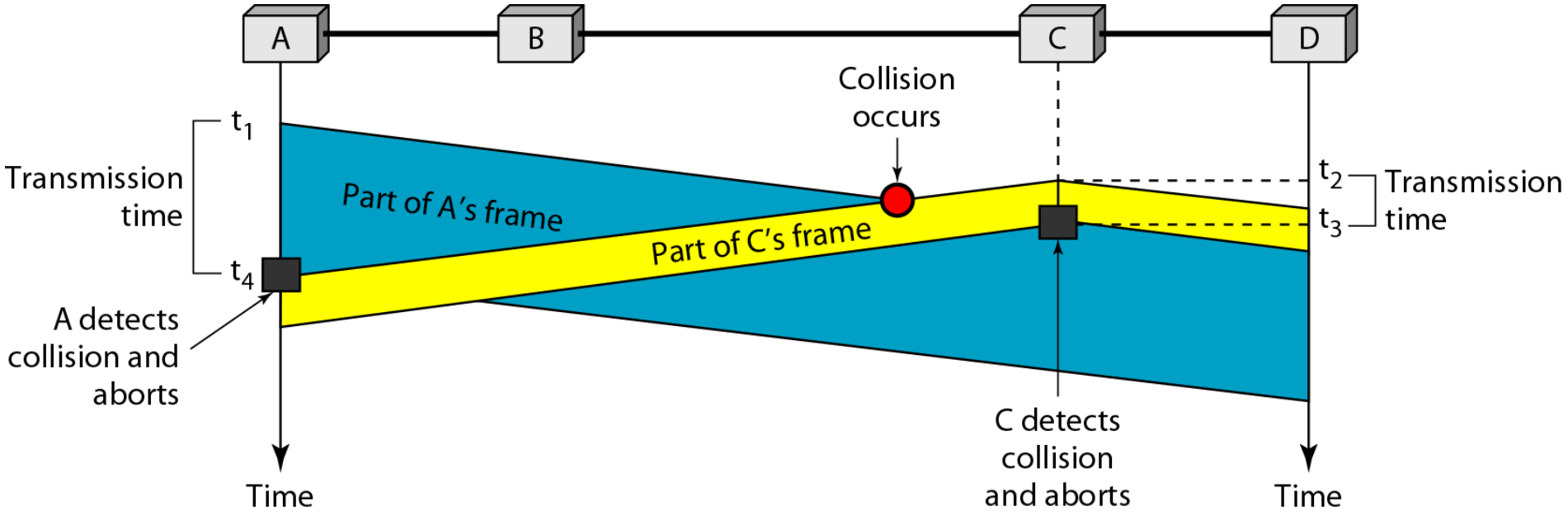
- Principle: Stations monitor the medium after sending a frame
    - If transmission successful → done;
      collision → resend the frame

Figure 12.12 *Collision of the first bit in CSMA/CD*



- At t_1, station ***A***, after executing its persistence procedure starts sending bits of its frame
- At t_2, station ***C*** has not yet sensed the first bit sent by ***A***; executes its persistence procedure and starts sending
- Collision occurs at some time after t_2
- ***C*** detects the collision at time t_3 and *A* detects the collision at time t_4 ; immediately abort transmission

Figure 12.13 *Collision and abortion in CSMA/CD*



Minimum frame size

- For CSMA/CD collision detection mechanism to work, the sending station must be able to hear a collision, before sending the last bit of the frame
- If two stations involved in the collision are maximum distance apart, the signal from the first station takes time T_p to reach the second, and the collision takes another time T_p to reach the first

- Therefore, the Frame transmission time $\boldsymbol{T_{fr}}$ must be at least two times the maximum propagation time T_p ($\boldsymbol{T_{fr} >= 2T_p}$)

- ***Throughput***
- Maximum throughput is based on the persistence method
- 1-*persistent*: Max throughput around 50% when G=1
- *Non-persistent*: can be upto 90%, when G is between 3 & 8

Example 12.5

A network using CSMA/CD has a bandwidth of 10 Mbps. If the maximum propagation time (including the delays in the devices) is 25.6 μs, what is the minimum size of the frame?

Solution

The frame transmission time, $T_{fr} = 2 \times T_p$ = 51.2 μs.

i.e. In the worst case, a station needs to transmit for a period of 51.2 μs to detect the collision

→ Minimum size of the frame = 10 Mbps × 51.2 μs = 512 bits or 64 bytes

(This is the minimum size of the frame for Standard Ethernet)

Figure 12.14 *Flow diagram for the CSMA/CD*

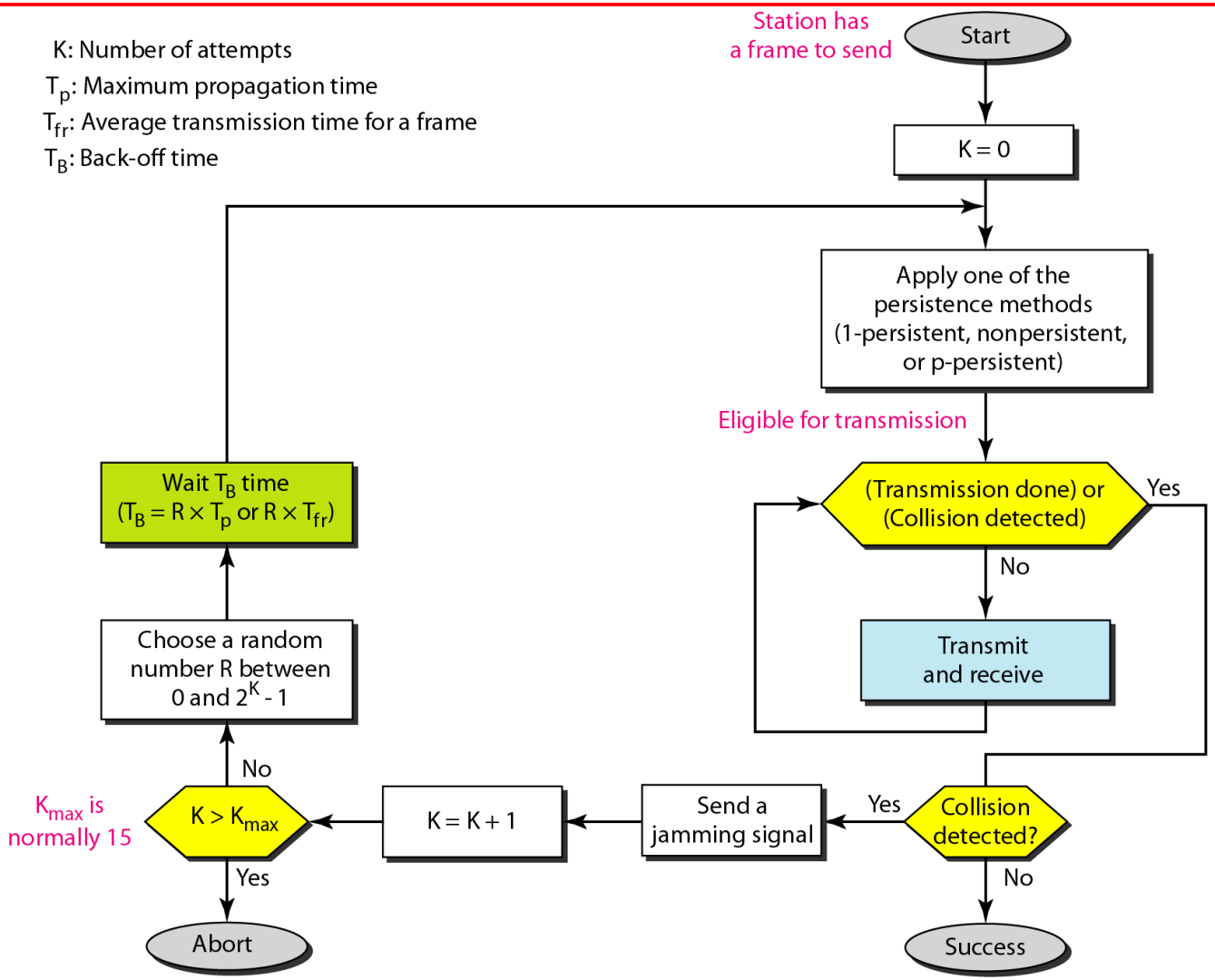
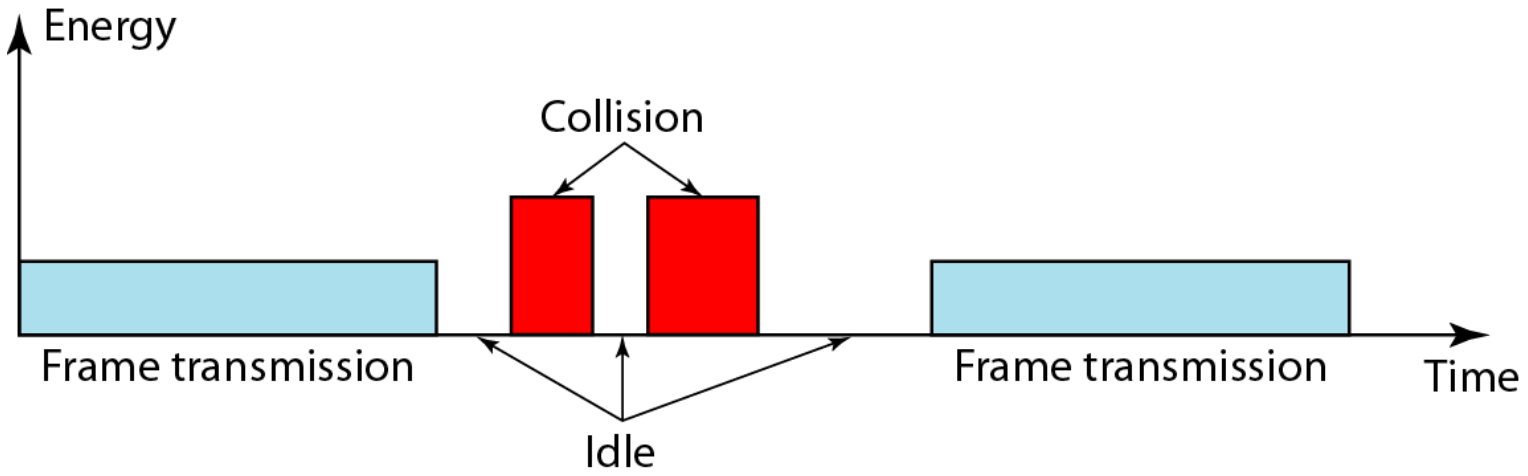


Figure 12.15 ***Energy level during transmission, idleness, or collision***

Energy Level on a channel: ***Zero*** *(channel idle),* ***Normal*** *(station successfully captured the channel and sending its frame), and* ***Abnormal*** *(collision ; level of energy is twice the normal level)*

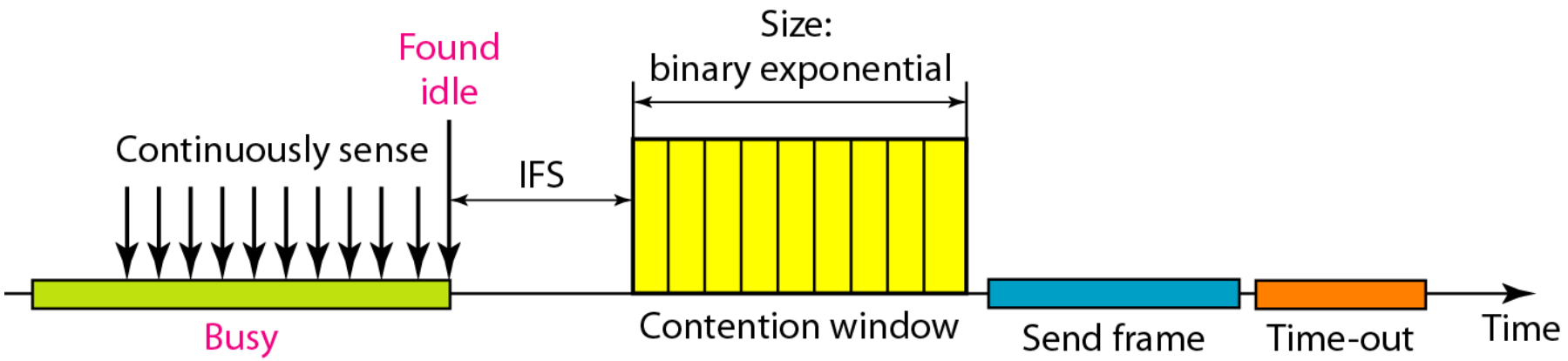


Carrier Sense Multiple Access with Collision Avoidance(CSMA/CA)

- For collision detection, significant energy needs to be added to the transmitted signal
  - CSMA/CD works well in wired networks, since the energy is almost double when collision occurs
- In wireless networks, much of the energy is lost in transmission (inverse square law)→ received signal has very little energy

  →Collision may add only 5 to 10% additional energy; thus collisions cannot be detected
- Need to avoid collisions→ CSMA/CA
- Collisions avoided through 3 strategies: the inter-frame space (IFS), contention window, and acknowledgements

Figure 12.16 *Timing in CSMA/CA*



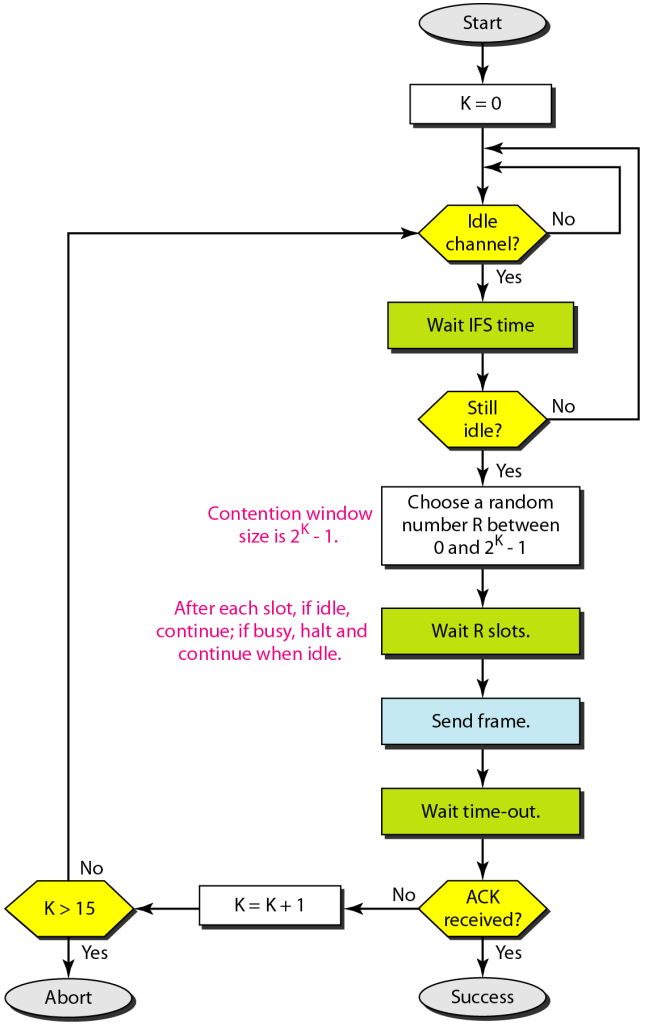
Note

In CSMA/CA, the IFS can also be used to define the priority of a station or a frame.

Note

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.

Figure 12.17 *Flow diagram for CSMA/CA*



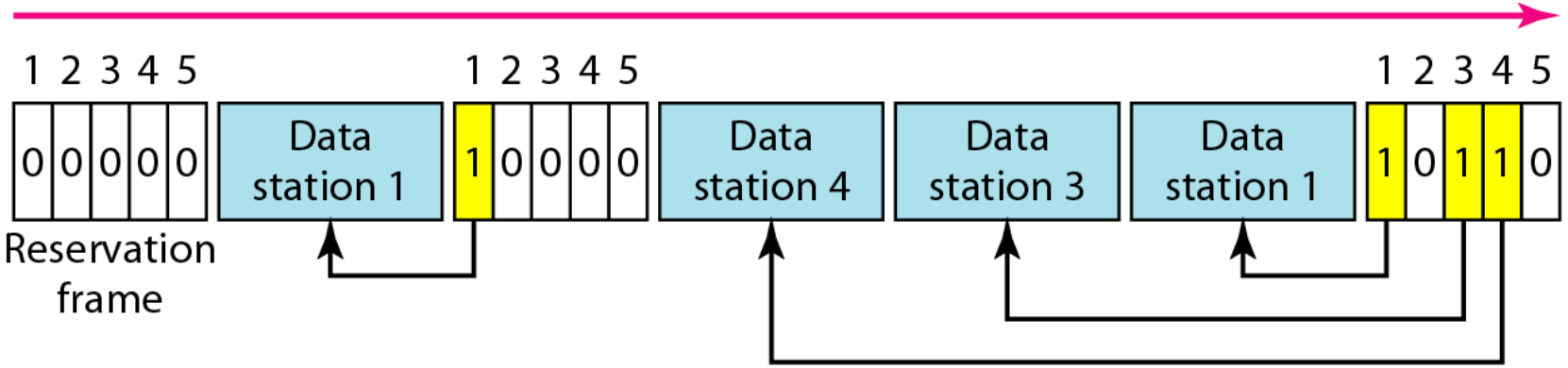
12-2 CONTROLLED ACCESS

In controlled access, the stations consult one another to find which station has the right to send. A station cannot send unless it has been authorized by other stations. We discuss three popular controlled-access methods.

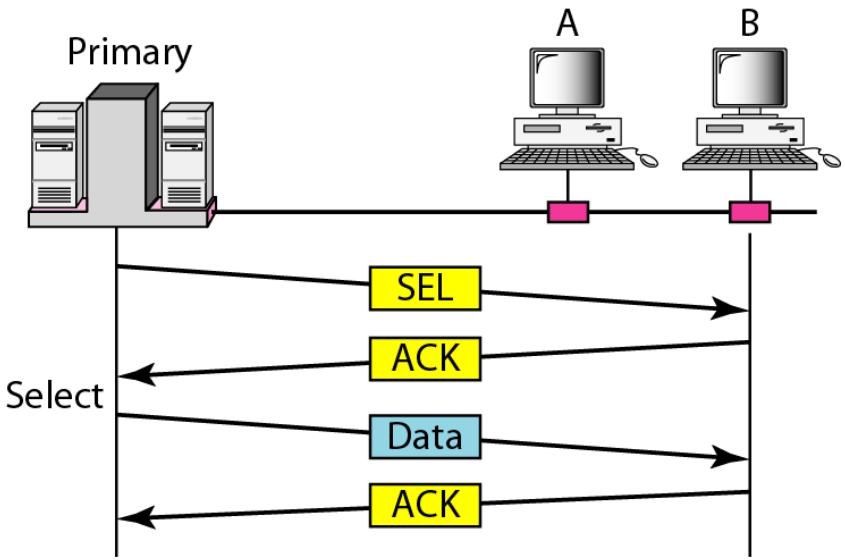
Topics discussed in this section:

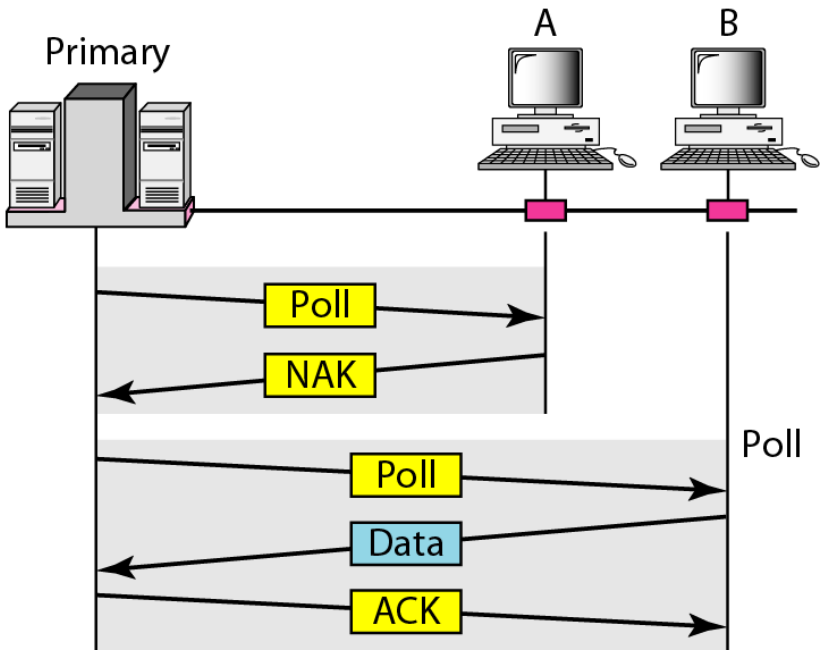
Reservation
Polling
Token Passing

Reservation: Figure 12.18 *Reservation access method*

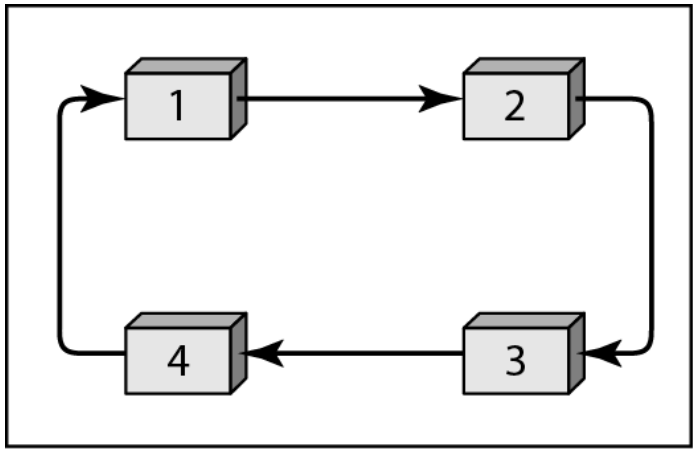


Polling: Figure 12.19 *Select and poll functions in polling access method*

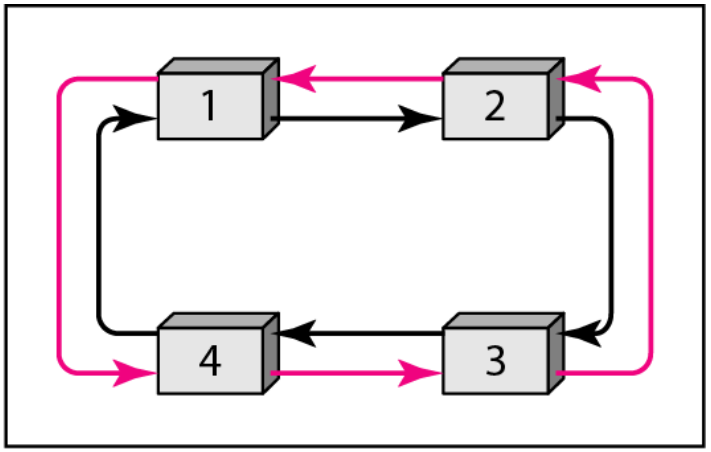




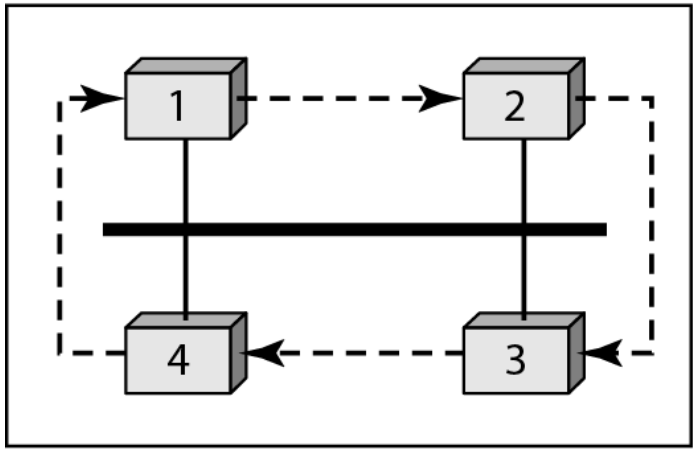
Token Passing: Fig 12.20 *Logical ring and physical topology in token-passing*



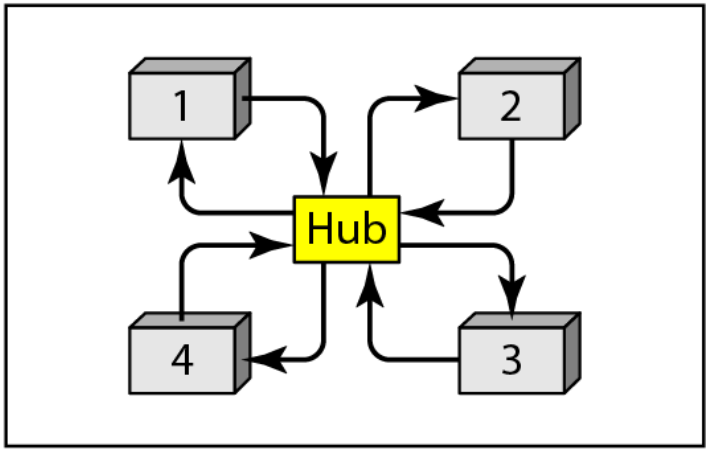
a. Physical ring



b. Dual ring



c. Bus ring



d. Star ring

12-3 CHANNELIZATION

***Channelization** is a multiple-access method in which the **available bandwidth** of a link is **shared in time, frequency, or through code**, between different stations.*
-Three channelization protocols.

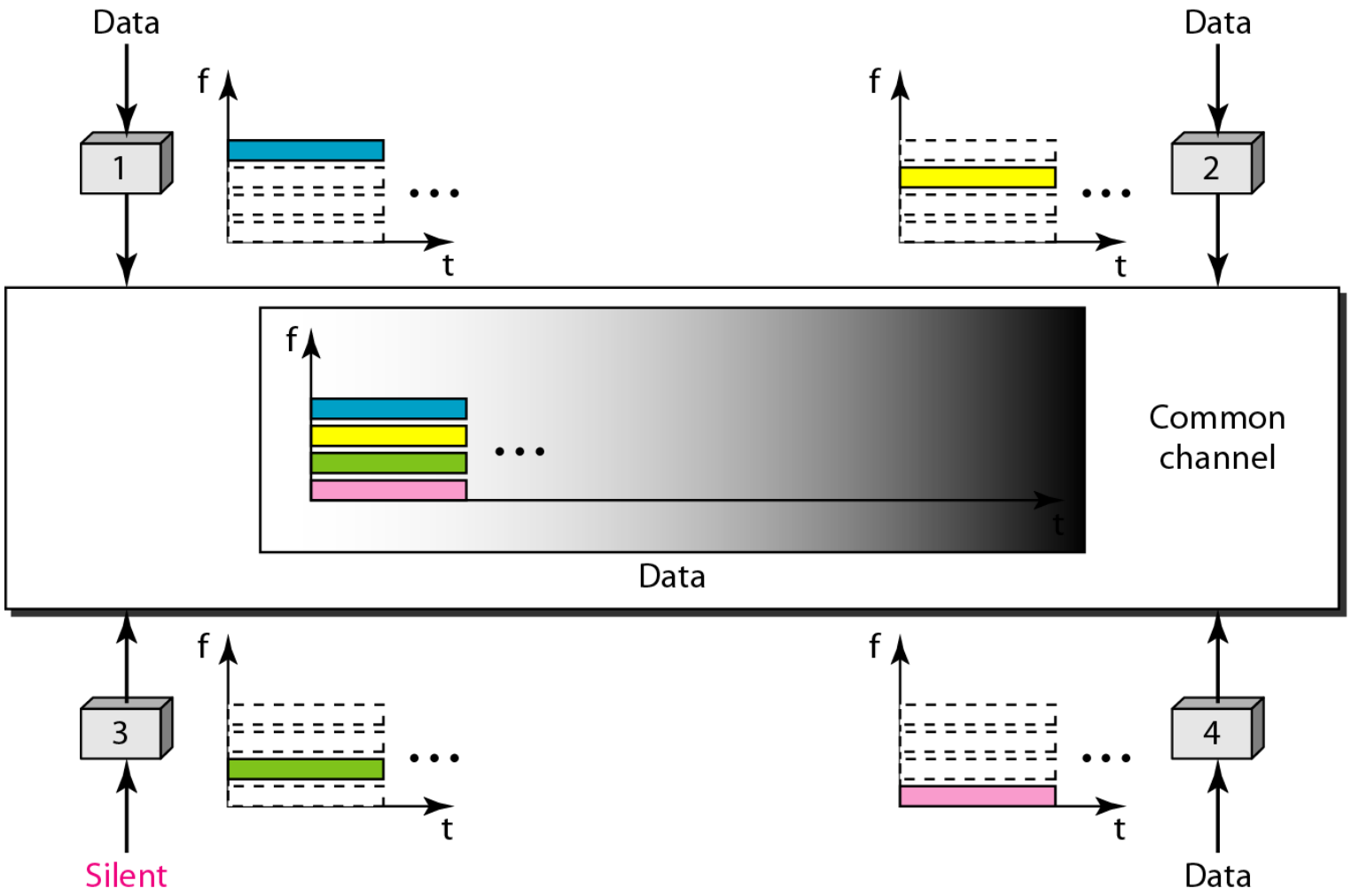
Topics discussed in this section:

Frequency-Division Multiple Access (FDMA)
Time-Division Multiple Access (TDMA)
Code-Division Multiple Access (CDMA)

Note

We see the application of all these methods in cellular phone systems

Figure 12.21 *Frequency-division multiple access (FDMA)*

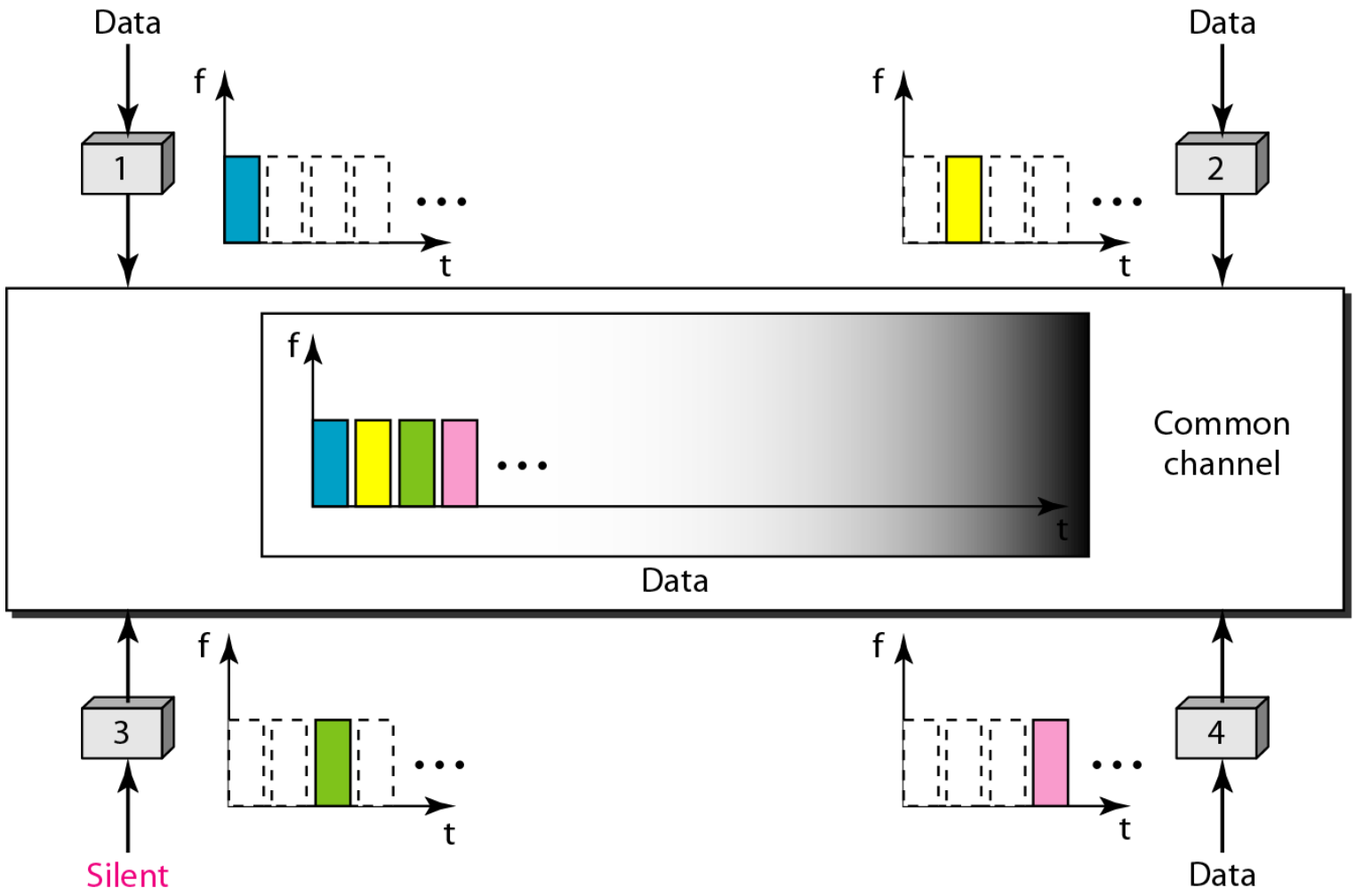


Note

In FDMA, the available bandwidth of the common channel is divided into bands that are separated by guard bands.

FDMA specifies a predetermined frequency band for the entire period of communication

Figure 12.22 *Time-division multiple access (TDMA)*



Note

In TDMA, the bandwidth is (~~just one channel that is~~) timeshared between different stations.

Note

In CDMA, one channel occupies the entire BW of the link, and carries all transmissions simultaneously.

Figure 12.23 *Simple idea of communication with code*

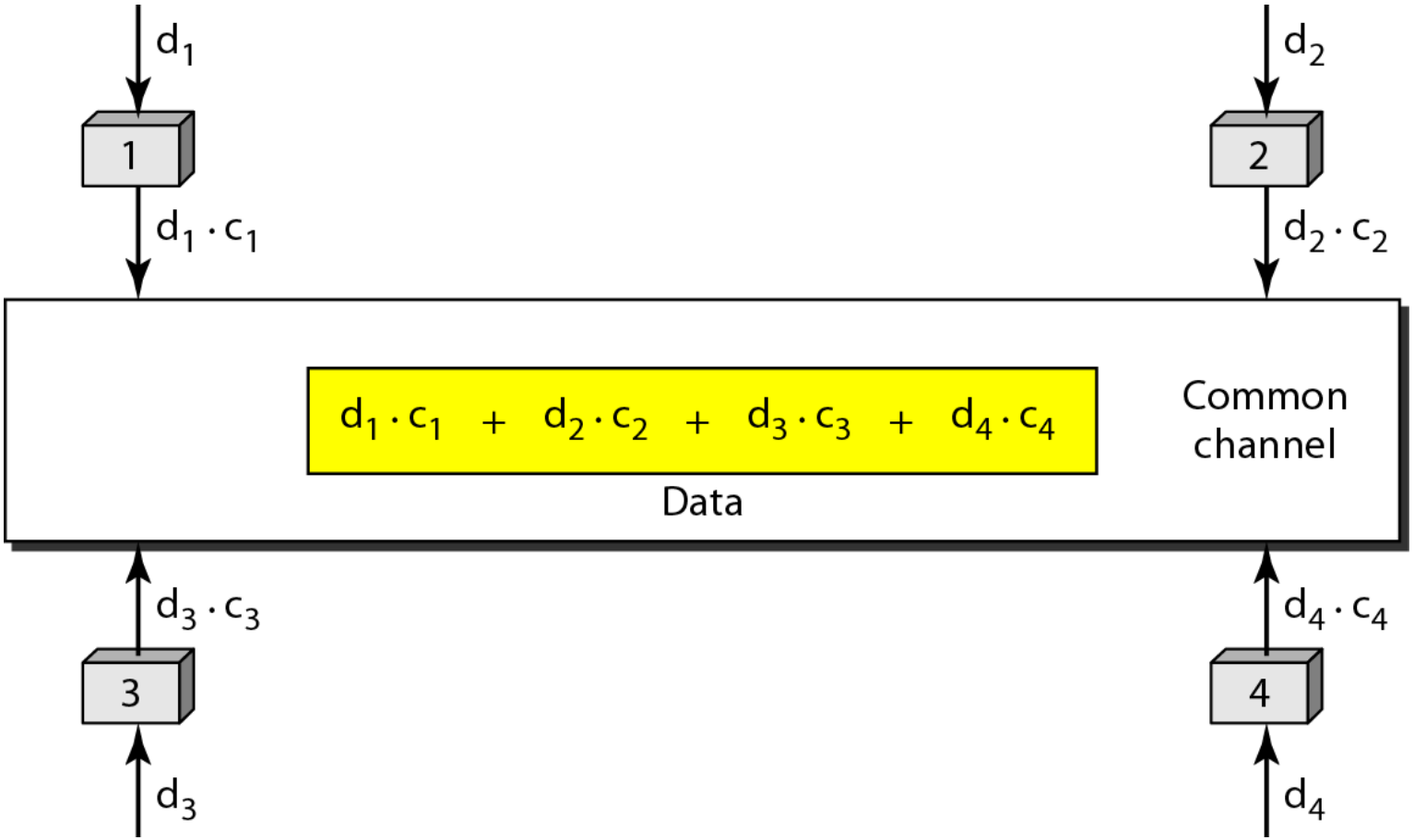


Figure 12.24 *Chip sequences*

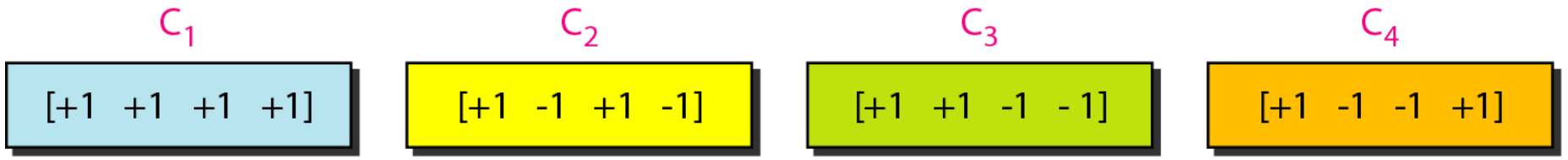


Figure 12.25 *Data representation in CDMA*

Figure 12.26 *Sharing channel in CDMA*

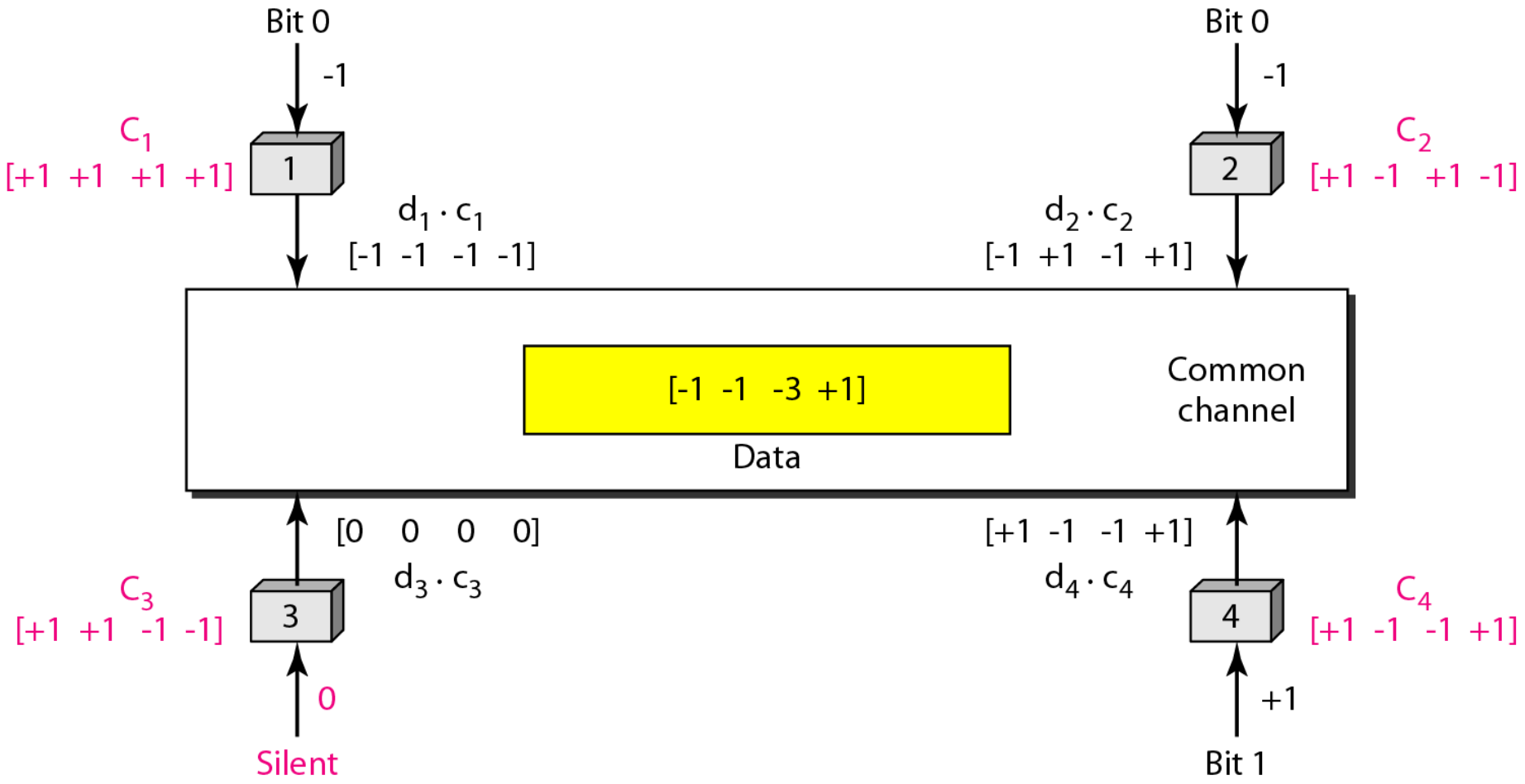


Figure 12.27 *Digital signal created by four stations in CDMA*

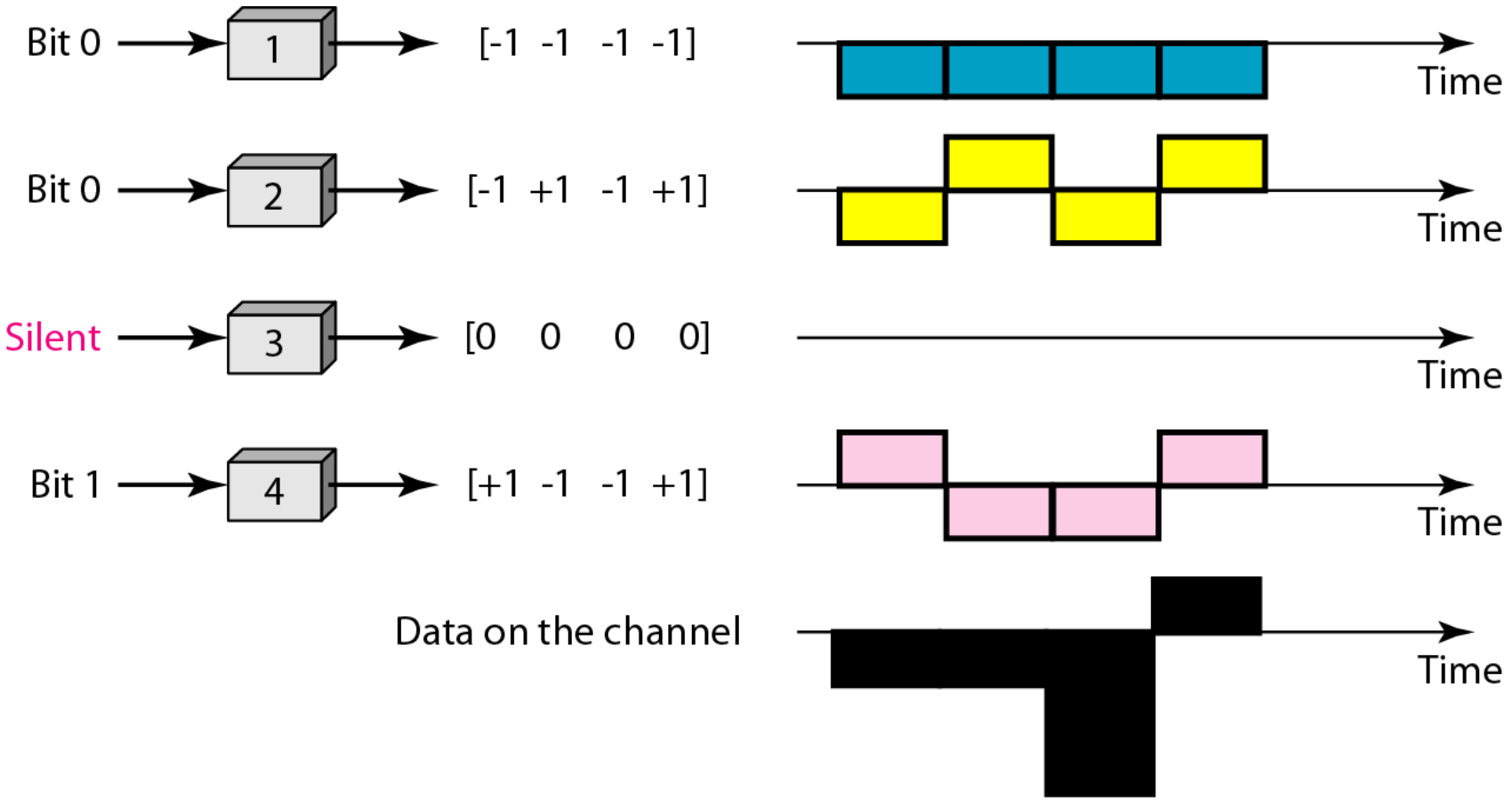
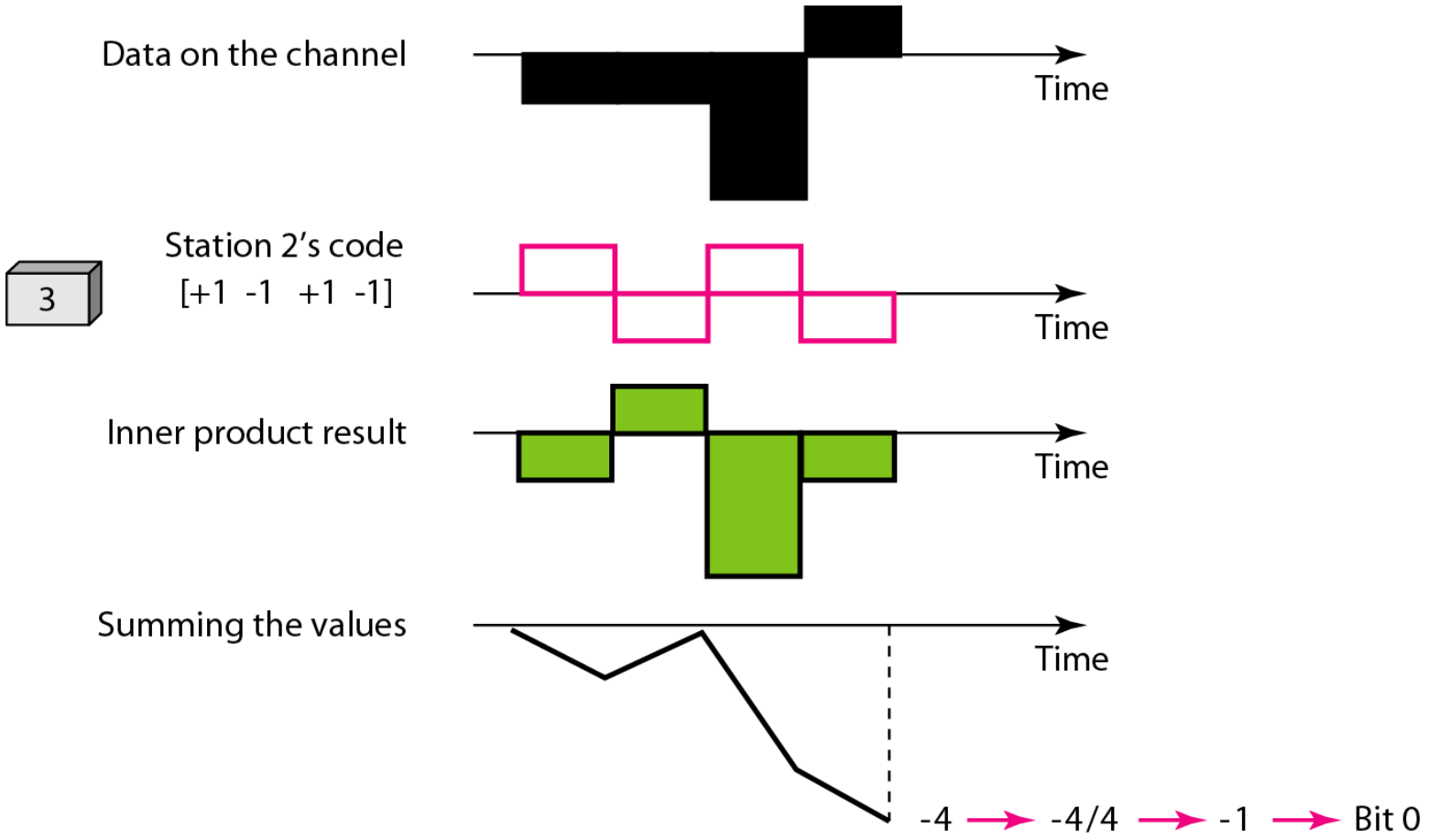


Figure 12.28 *Decoding of the composite signal for one in CDMA*



Sequence Generation:

Figure 12.29 *General rule and examples of creating Walsh tables*

$$W_1 = [+1] \qquad W_{2N} = \begin{bmatrix} W_N & W_N \\ W_N & \overline{W_N} \end{bmatrix}$$

a. Two basic rules

$$W_1 = [+1]$$

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

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

b. Generation of W_1, W_2, and W_4

Note

The number of sequences in a Walsh table needs to be $N = 2^m$.

Example 12.6

Find the chips for a network with

a. Two stations *b. Four stations*

Solution

We can use the rows of W_2 and W_4 in Figure 12.29:

a. For a two-station network, we have

$$[+1\ +1] \text{ and } [+1\ -1].$$

b. For a four-station network we have

$$[+1\ +1\ +1\ +1],\ [+1\ -1\ +1\ -1],$$
$$[+1\ +1\ -1\ -1],\ \textit{and}\ \ [+1\ -1\ -1\ +1].$$

Example 12.7

What is the number of sequences if we have 90 stations in our network?

Solution

The number of sequences needs to be 2^m. We need to choose $m = 7$ and $N = 2^7$ or 128. We can then use 90 of the sequences as the chips.

Example 12.8

Prove that a receiving station can get the data sent by a specific sender if it multiplies the entire data on the channel by the sender's chip code and then divides it by the number of stations.

Solution

Let us prove this for the first station, using our previous four-station example. We can say that the data on the channel

$$D = (d_1 \cdot c_1 + d_2 \cdot c_2 + d_3 \cdot c_3 + d_4 \cdot c_4)$$

Example 12.8 (continued)

The receiver which wants to get the data sent by station 1 multiplies these data by c_1.

$$
\begin{aligned}
D \cdot c_1 &= (d_1 \cdot c_1 + d_2 \cdot c_2 + d_3 \cdot c_3 + d_4 \cdot c_4) \cdot c_1 \\
&= d_1 \cdot c_1 \cdot c_1 + d_2 \cdot c_2 \cdot c_1 + d_3 \cdot c_3 \cdot c_1 + d_4 \cdot c_4 \cdot c_1 \\
&= d_1 \times N + d_2 \times 0 + d_3 \times 0 + d_4 \times 0 \\
&= d_1 \times N
\end{aligned}
$$

When we divide the result by N, we get d_1 .

Module 2

Wired LANs: Ethernet

13-1 IEEE STANDARDS

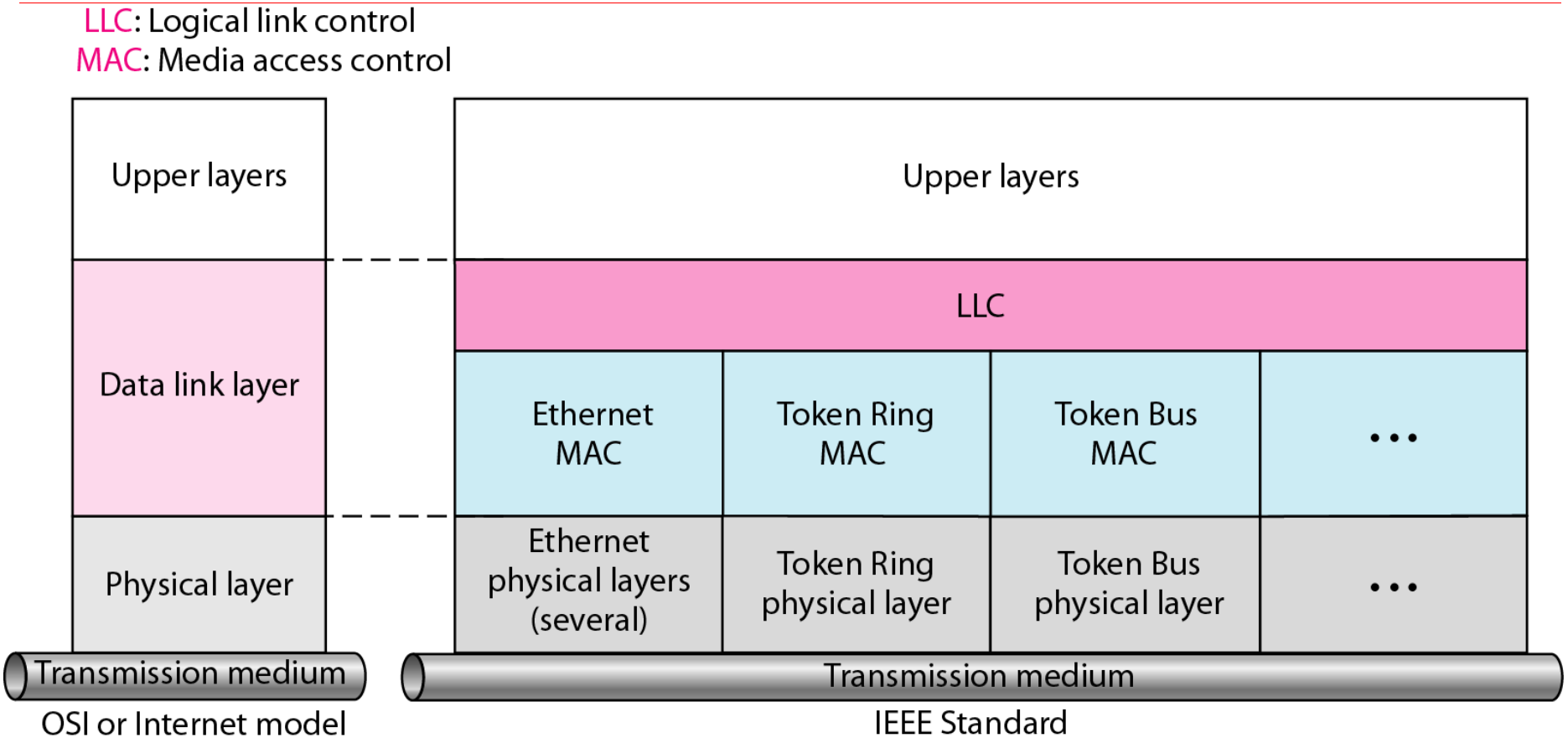
In 1985, the Computer Society of the IEEE started a project, called Project 802, to set standards to enable intercommunication among equipment from a variety of manufacturers.

Project 802 is a way of specifying functions of the Physical layer and the Data Link layer of major LAN protocols.

Topics discussed in this section:

Data Link Layer
Physical Layer

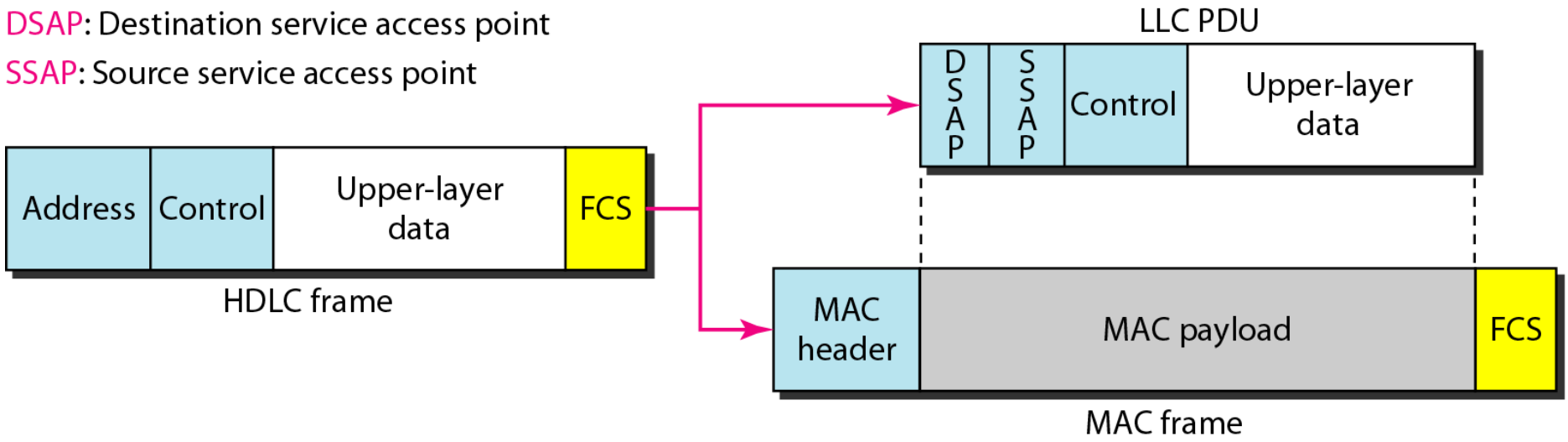
Figure 13.1 *IEEE standard for LANs*



- LLC provides single data link control protocol for all IEEE LANs; MAC provides different protocols for different LANs

Framing: LLC defines a protocol data unit (PDU) similar to HDLC

Figure 13.2 HDLC frame compared with LLC and MAC frames



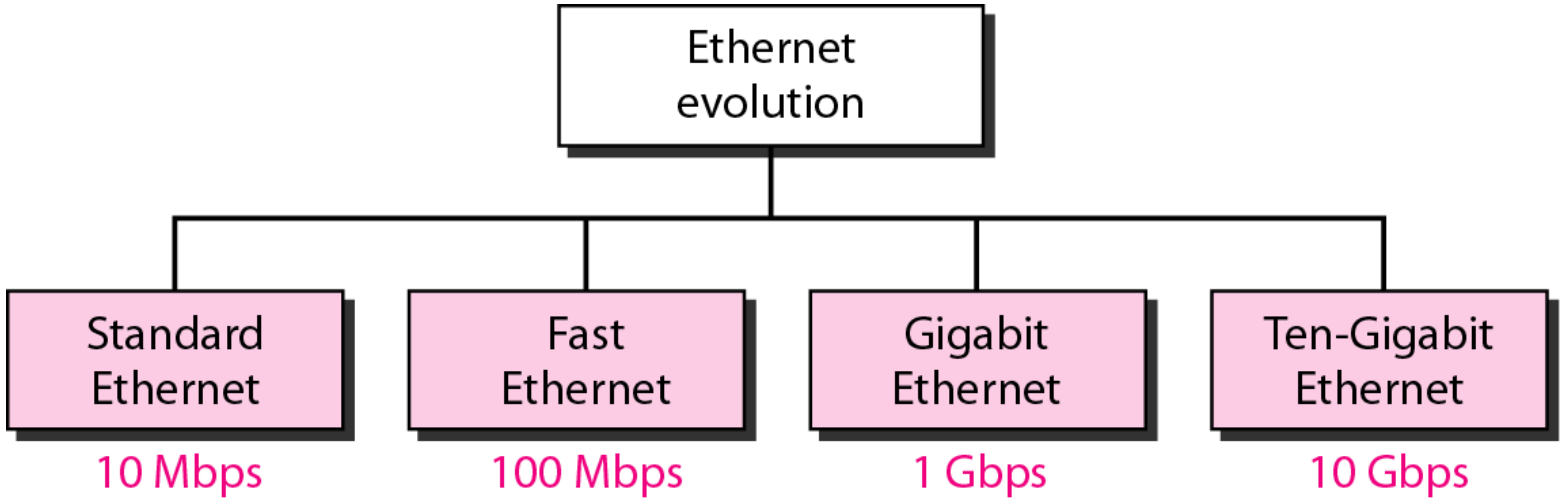
13-2 STANDARD ETHERNET

The original Ethernet was created in 1976 at Xerox's Palo Alto Research Center (PARC). Since then, it has gone through four generations. We briefly discuss the ***Standard (or traditional) Ethernet*** *in this section.*

Topics discussed in this section:

MAC Sublayer
Physical Layer

Figure 13.3 *Ethernet evolution through four generations*



MAC Sublayer: Figure 13.4 *IEEE 802.3 MAC frame*

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

SFD: Start frame delimiter, flag (10101011)

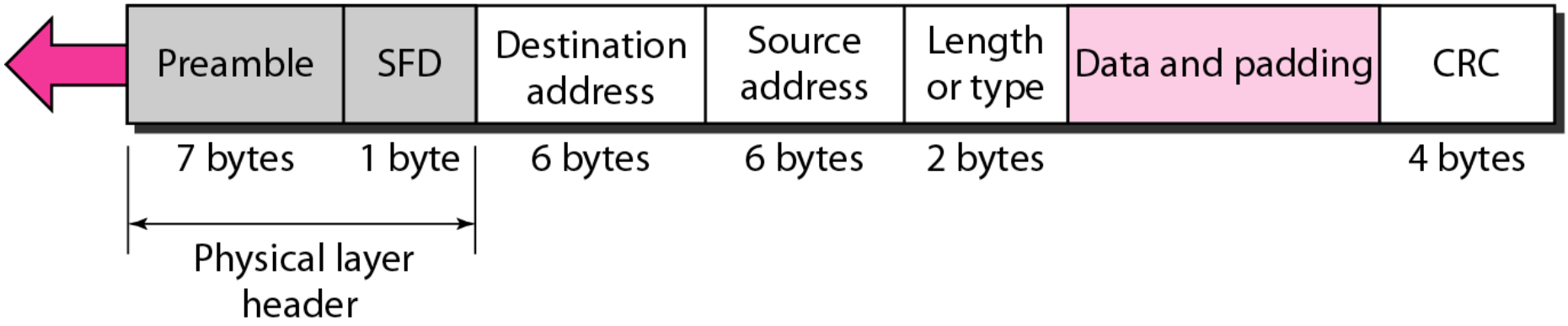
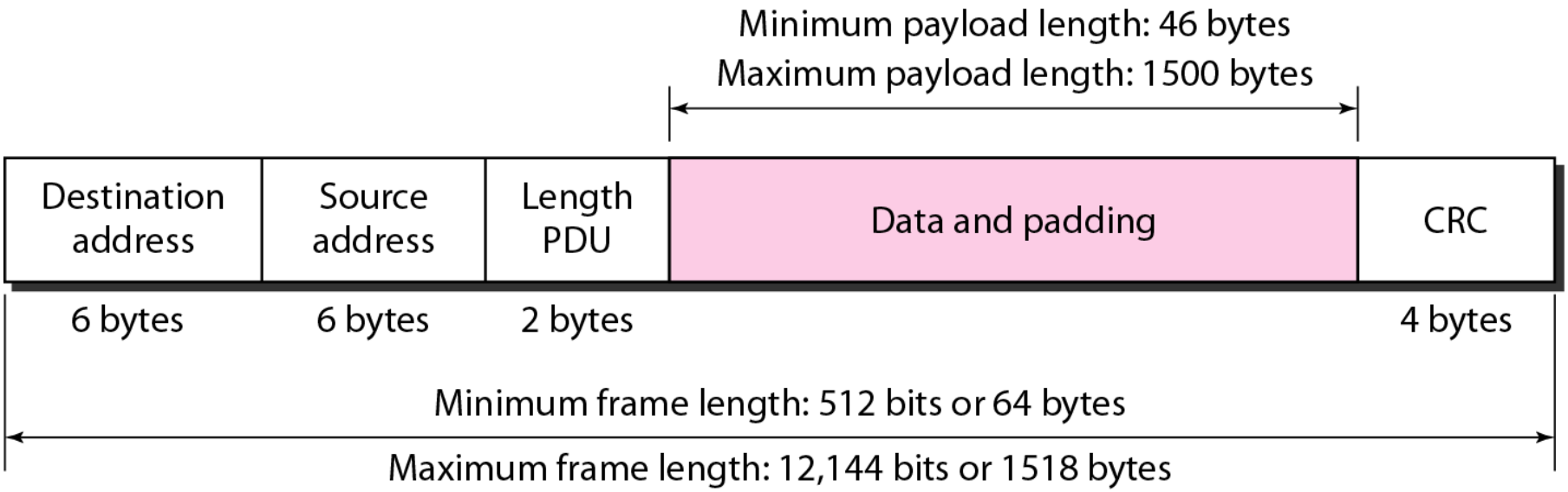


Figure 13.5 *Minimum and maximum frame lengths*



Note

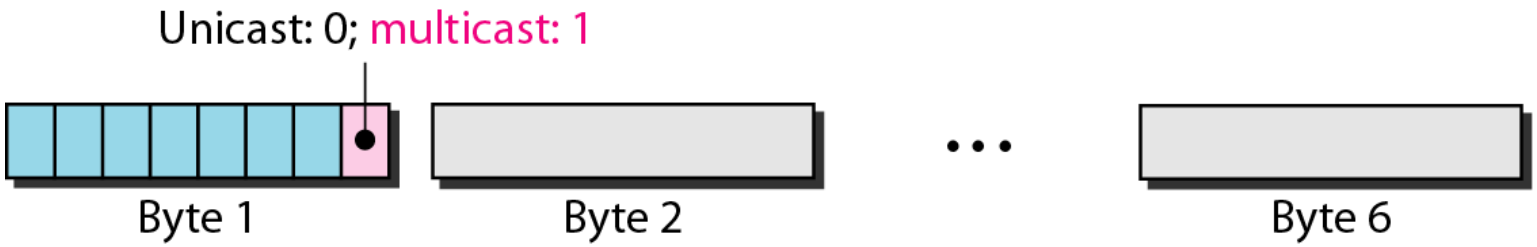
Frame length:

Minimum: 64 bytes (512 bits)

Maximum: 1518 bytes (12,144 bits)

Figure 13.6 *Example of an Ethernet address in hexadecimal notation*

Figure 13.7 *Unicast and multicast addresses*



Note

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.

Note

The broadcast destination address is a special case of the multicast address in which all bits are 1s.

Example 13.1

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.

Example 13.2

Show how the address 47:20:1B:2E:08:EE is sent out on line.

Solution

The address is sent left-to-right, byte by byte;
for each byte, it is sent right-to-left, bit by bit, as shown:

Figure 13.8 *Categories of Standard Ethernet*

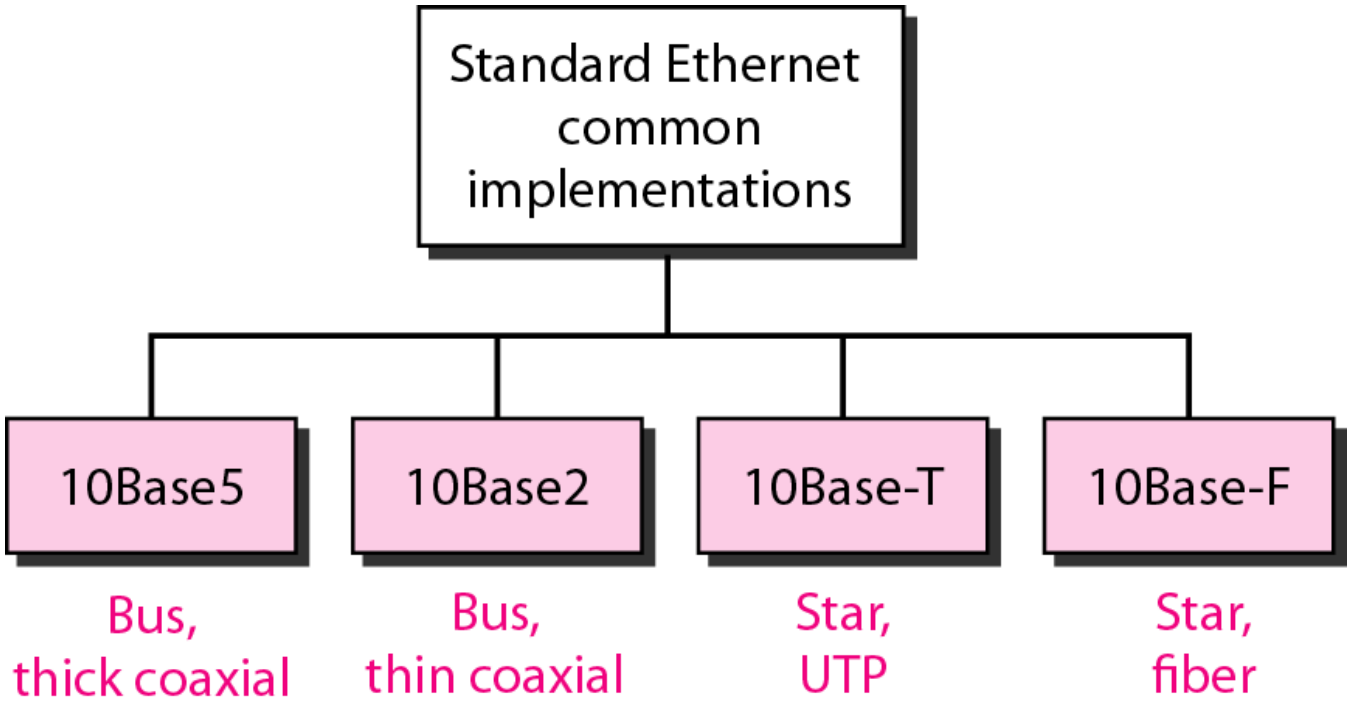


Figure 13.9 *Encoding in a Standard Ethernet implementation*

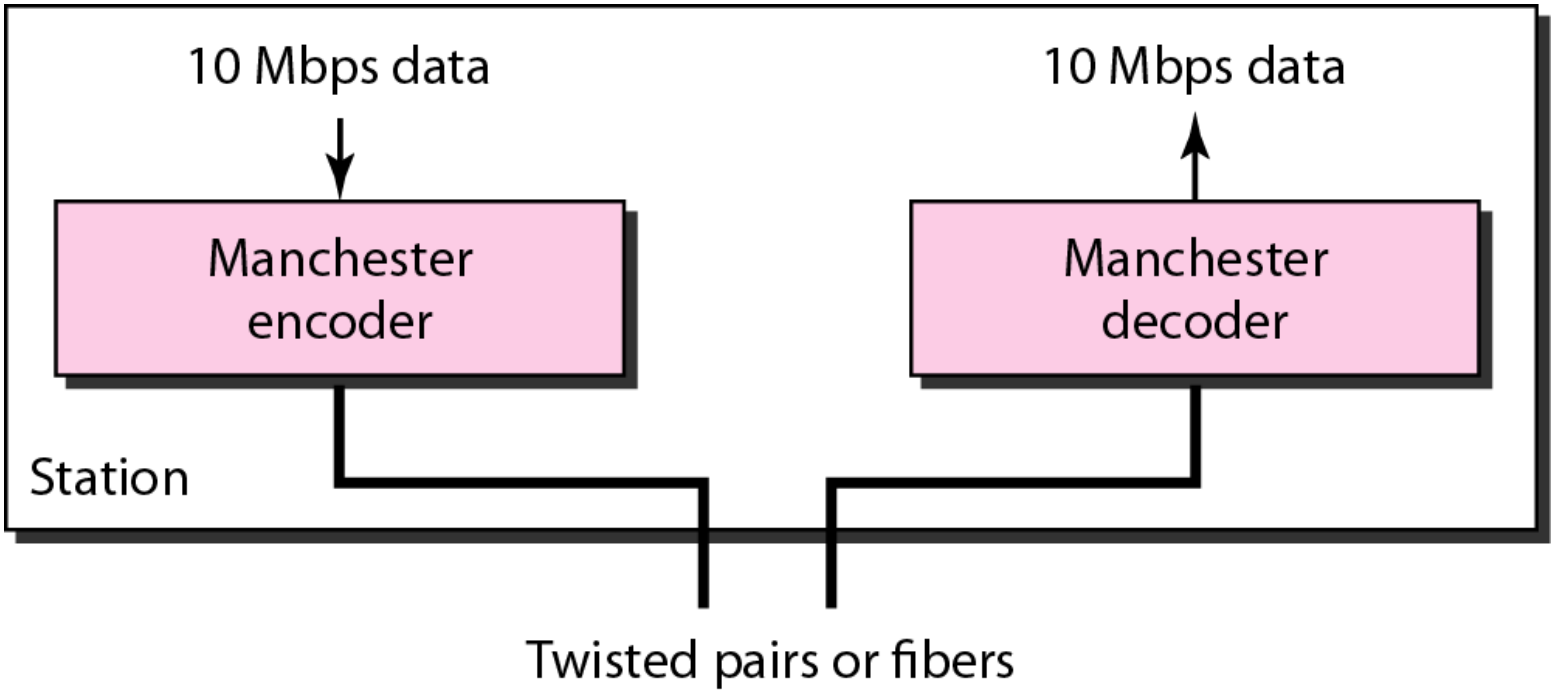


Figure 13.10 *10Base5 implementation*

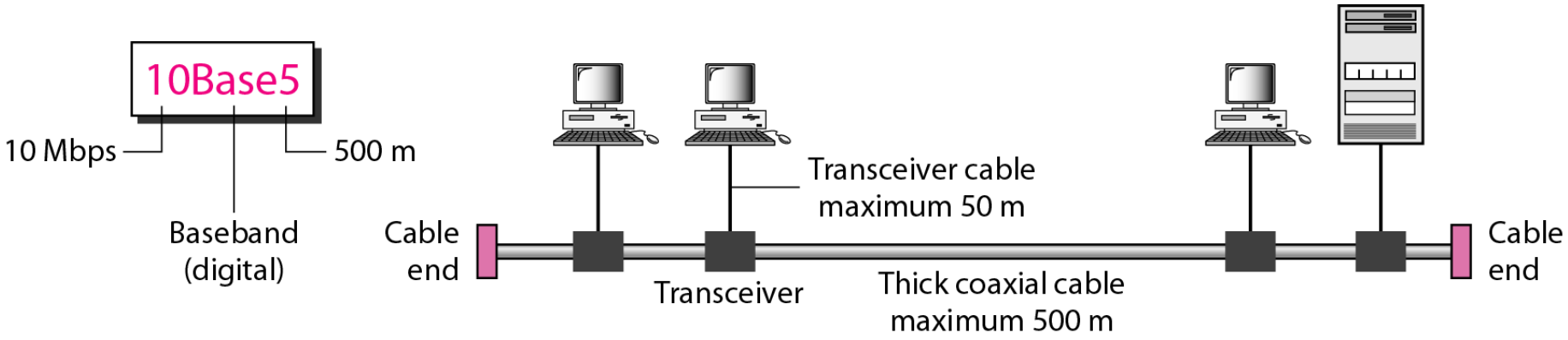


Figure 13.11 *10Base2 implementation*

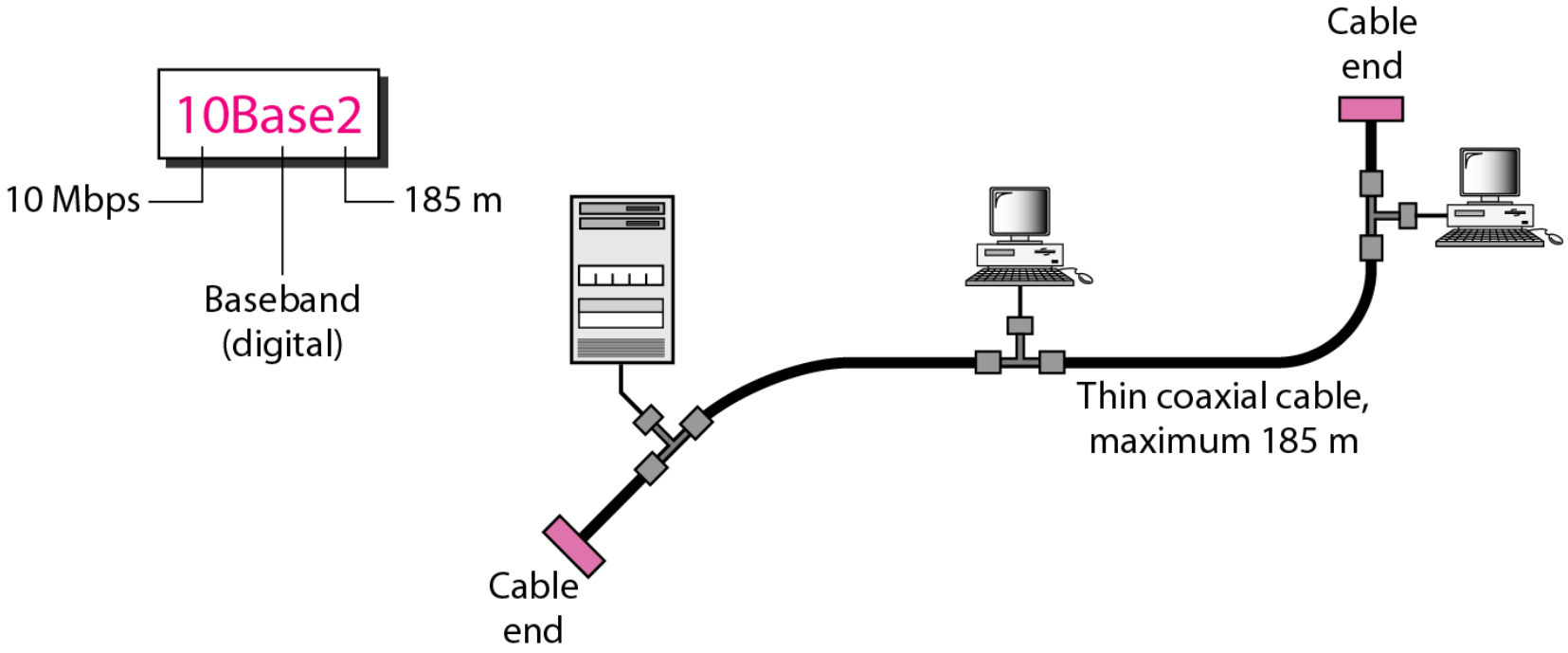


Figure 13.12 *10Base-T implementation*

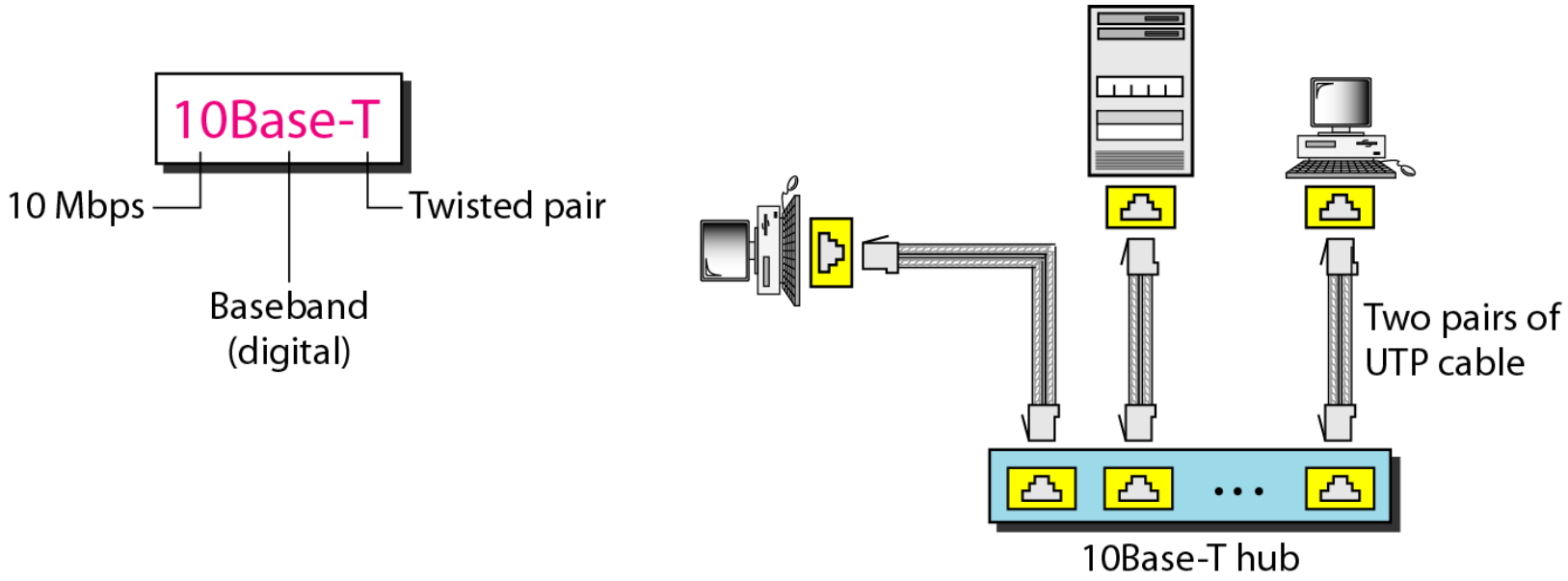


Figure 13.13 *10Base-F implementation*

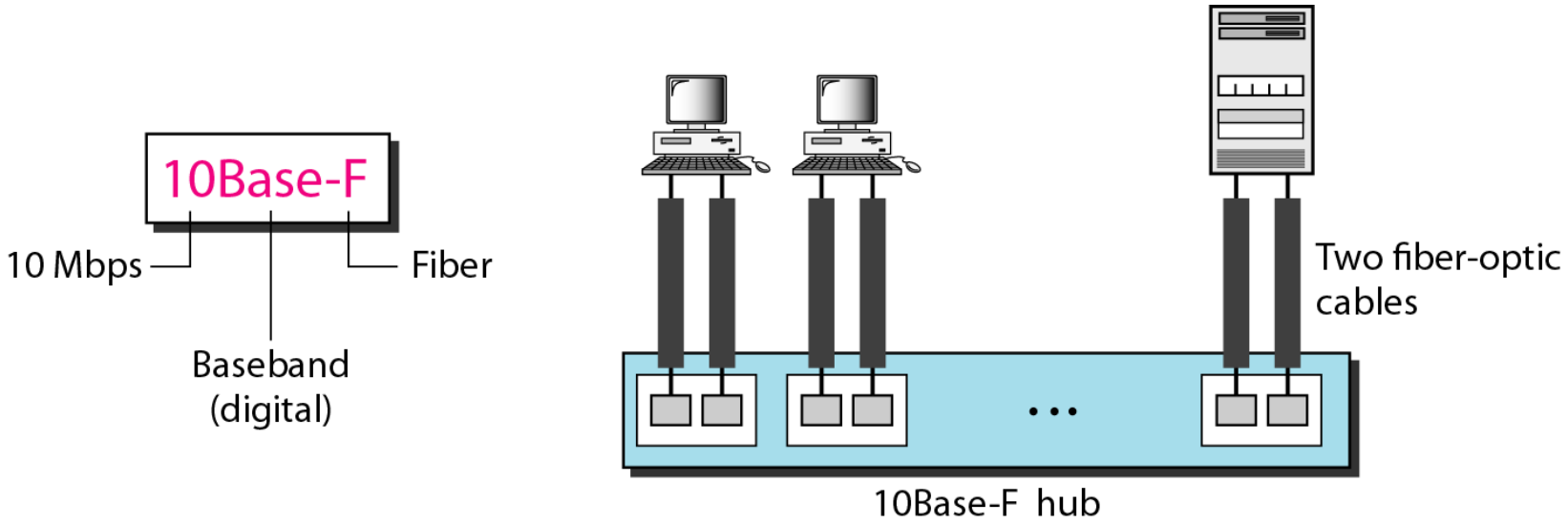


Table 13.1 *Summary of Standard Ethernet implementations*

Characteristics	*10Base5*	*10Base2*	*10Base-T*	*10Base-F*
Media	Thick coaxial cable	Thin coaxial cable	2 UTP	2 Fiber
Maximum length	500 m	185 m	100 m	2000 m
Line encoding	Manchester	Manchester	Manchester	Manchester

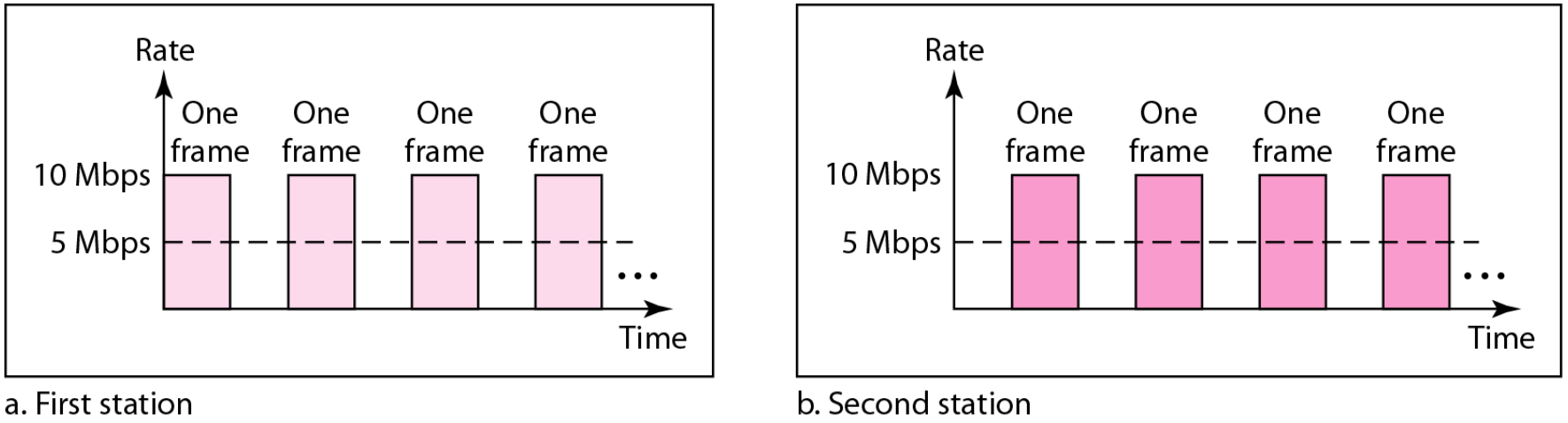
13-3 CHANGES IN THE STANDARD

The 10-Mbps Standard Ethernet has gone through several changes before moving to the higher data rates. These changes actually opened the road to the evolution of the Ethernet to become compatible with other high-data-rate LANs.

Topics discussed in this section:

Bridged Ethernet
Switched Ethernet
Full-Duplex Ethernet

Figure 13.14 *Sharing bandwidth*

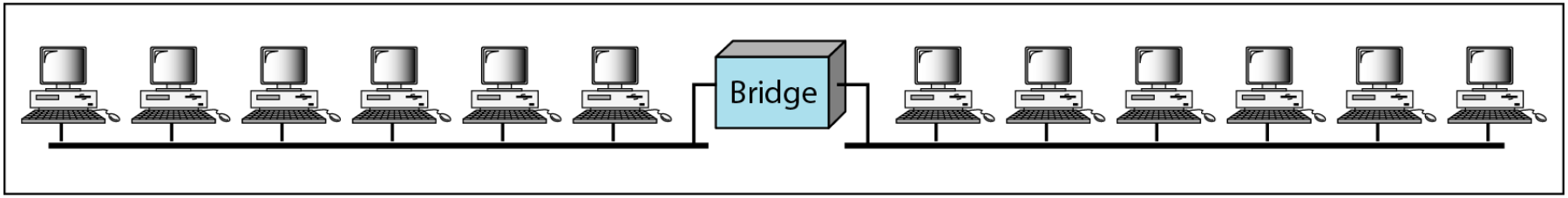


a. First station

b. Second station

Figure 13.15 *A network with and without a bridge*

a. Without bridging



b. With bridging

Figure 13.16 *Collision domains in an unbridged network and a bridged network*

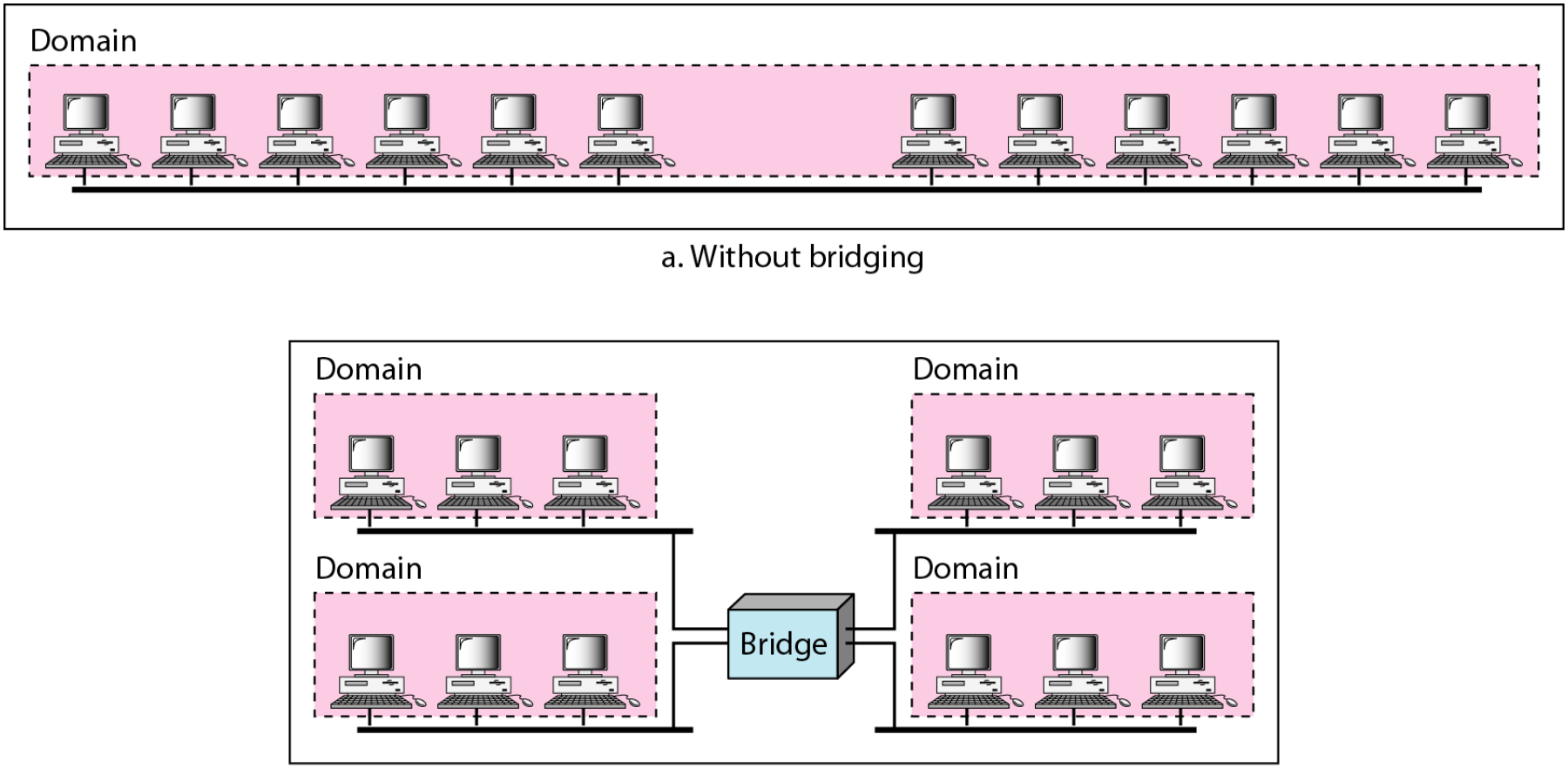


Figure 13.17 *Switched Ethernet*

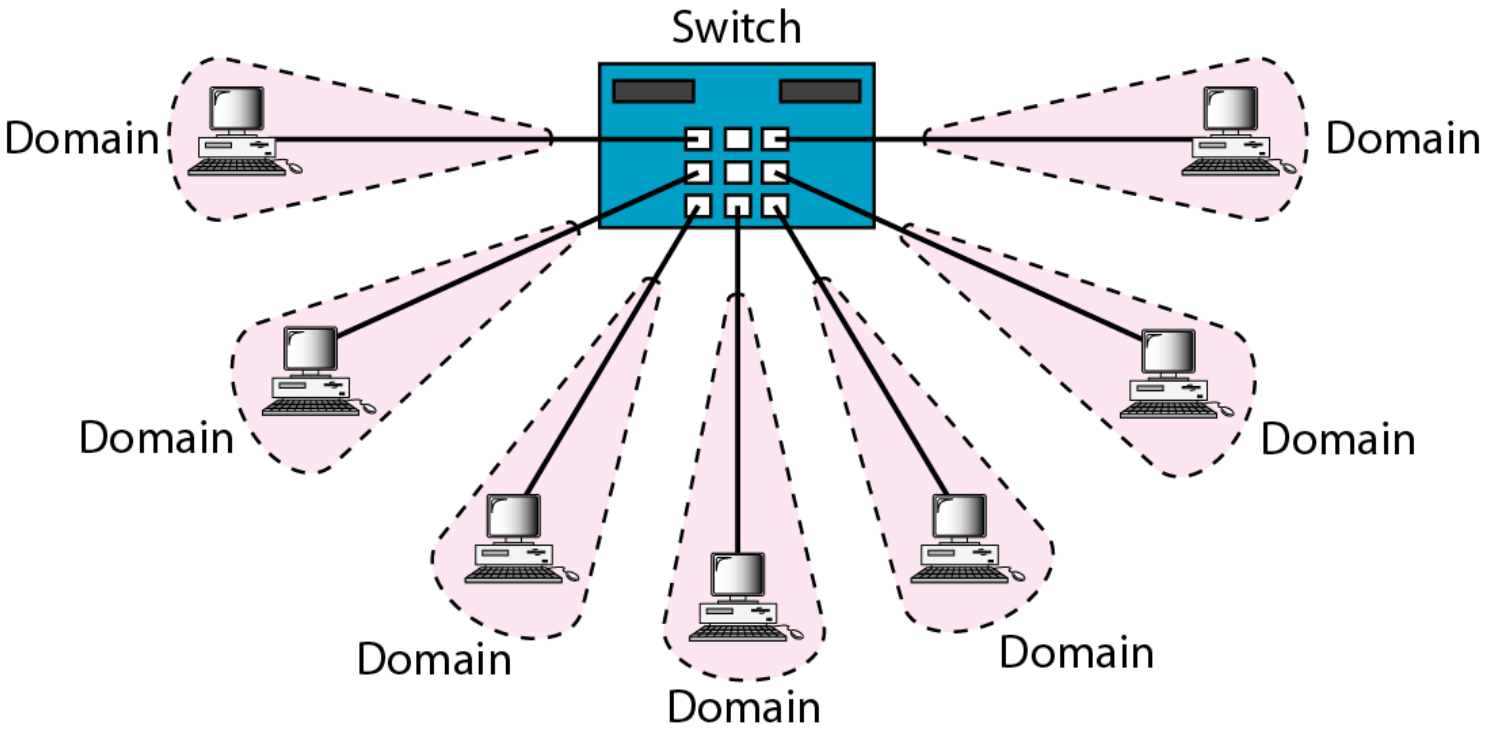
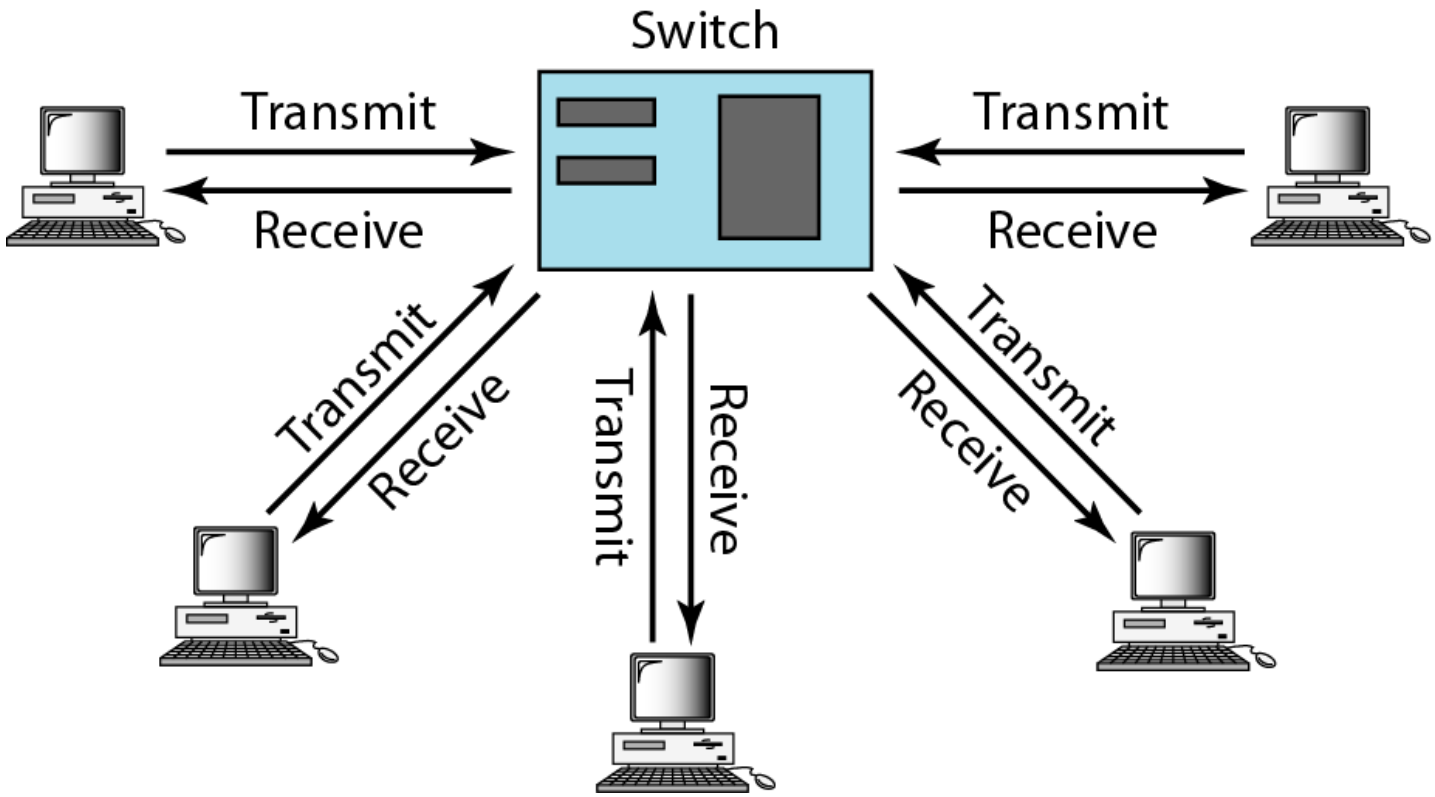


Figure 13.18 *Full-duplex switched Ethernet*



13-4 FAST ETHERNET

Fast Ethernet was designed to compete with LAN protocols such as FDDI or Fiber Channel. IEEE created Fast Ethernet under the name 802.3u. Fast Ethernet is backward-compatible with Standard Ethernet, but it can transmit data 10 times faster at a rate of 100 Mbps.

Topics discussed in this section:

MAC Sublayer

Physical Layer

Figure 13.19 *Fast Ethernet topology*

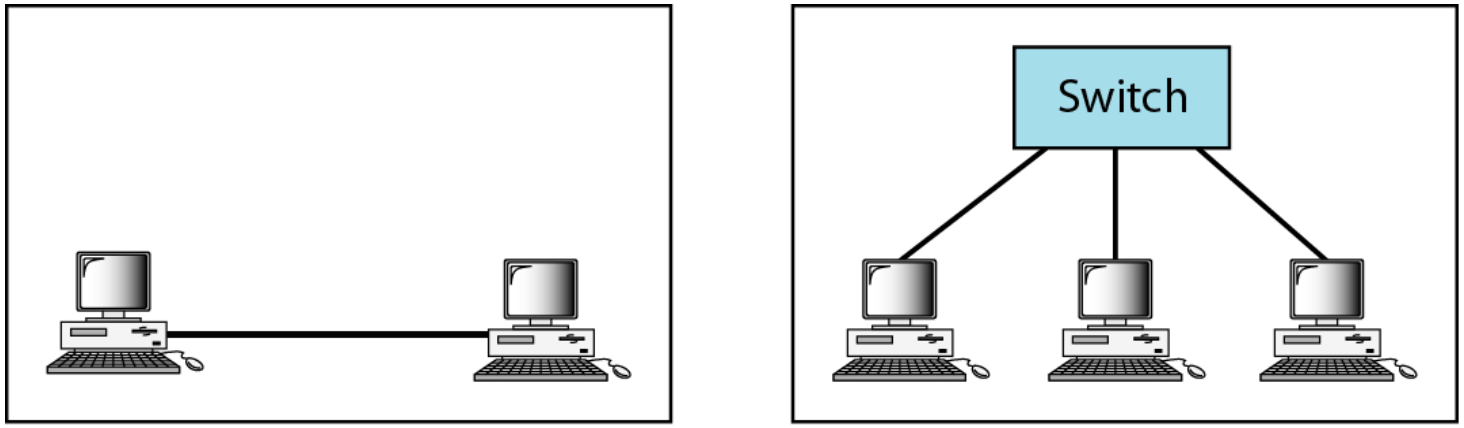


Figure 13.20 *Fast Ethernet implementations*

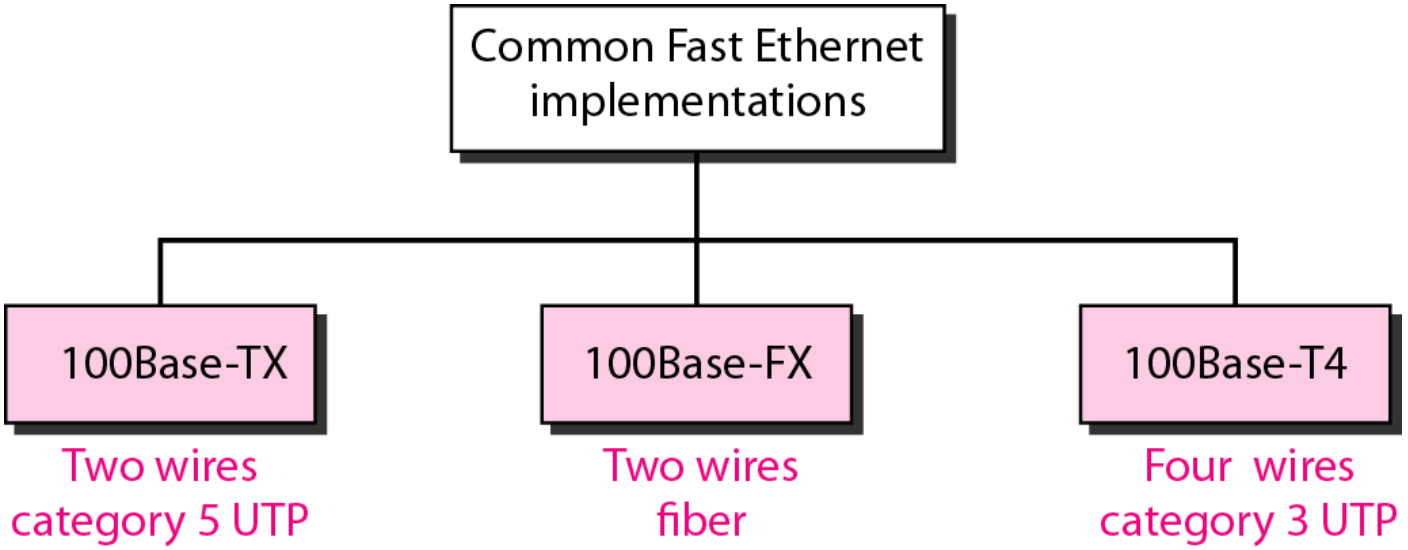


Figure 13.21 *Encoding for Fast Ethernet implementation*

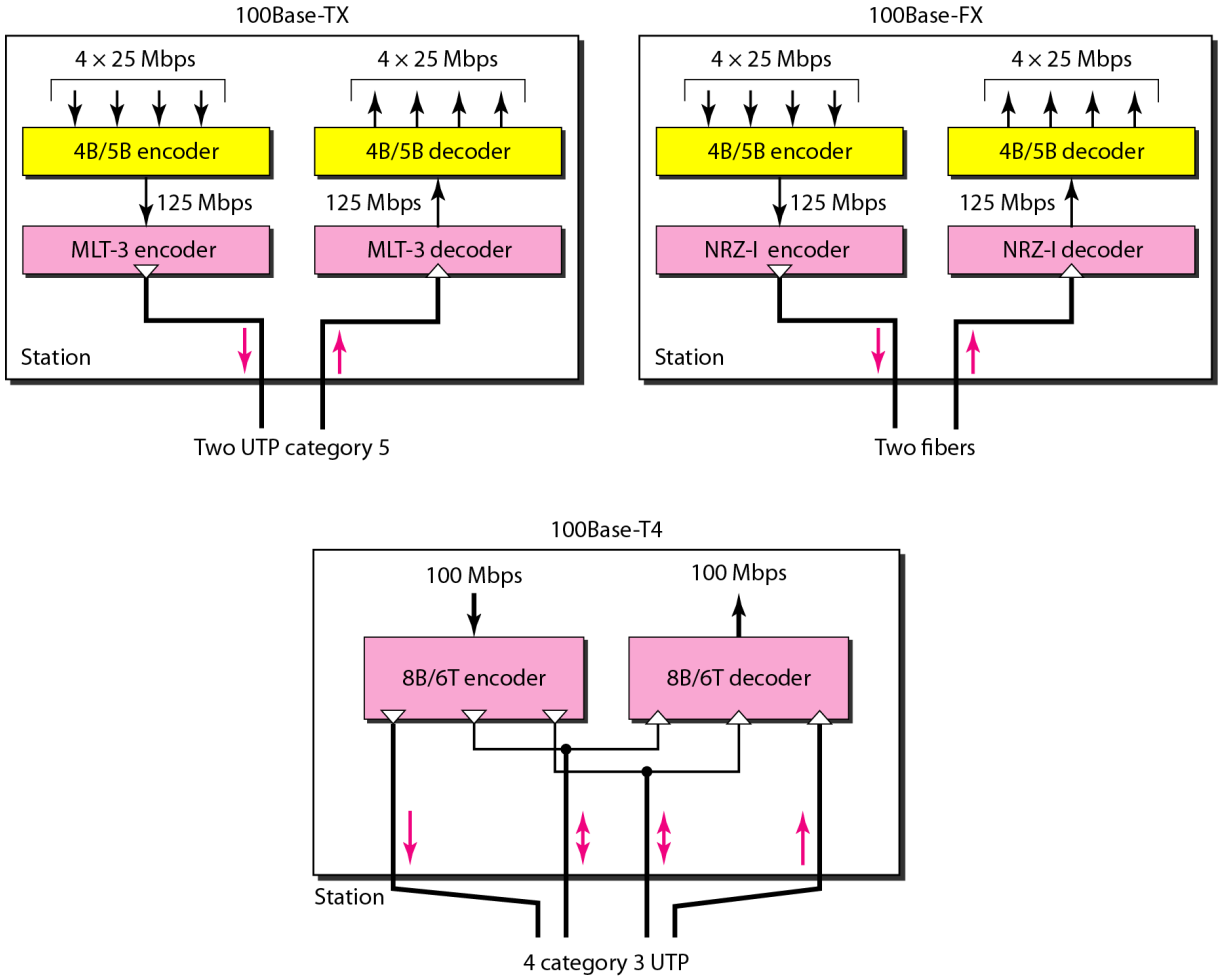


Table 13.2 *Summary of Fast Ethernet implementations*

Characteristics	*100Base-TX*	*100Base-FX*	*100Base-T4*
Media	Cat 5 UTP or STP	Fiber	Cat 4 UTP
Number of wires	2	2	4
Maximum length	100 m	100 m	100 m
Block encoding	4B/5B	4B/5B	
Line encoding	MLT-3	NRZ-I	8B/6T

13-5 GIGABIT ETHERNET

The need for an even higher data rate resulted in the design of the Gigabit Ethernet protocol (1000 Mbps). The IEEE committee calls the standard 802.3z.

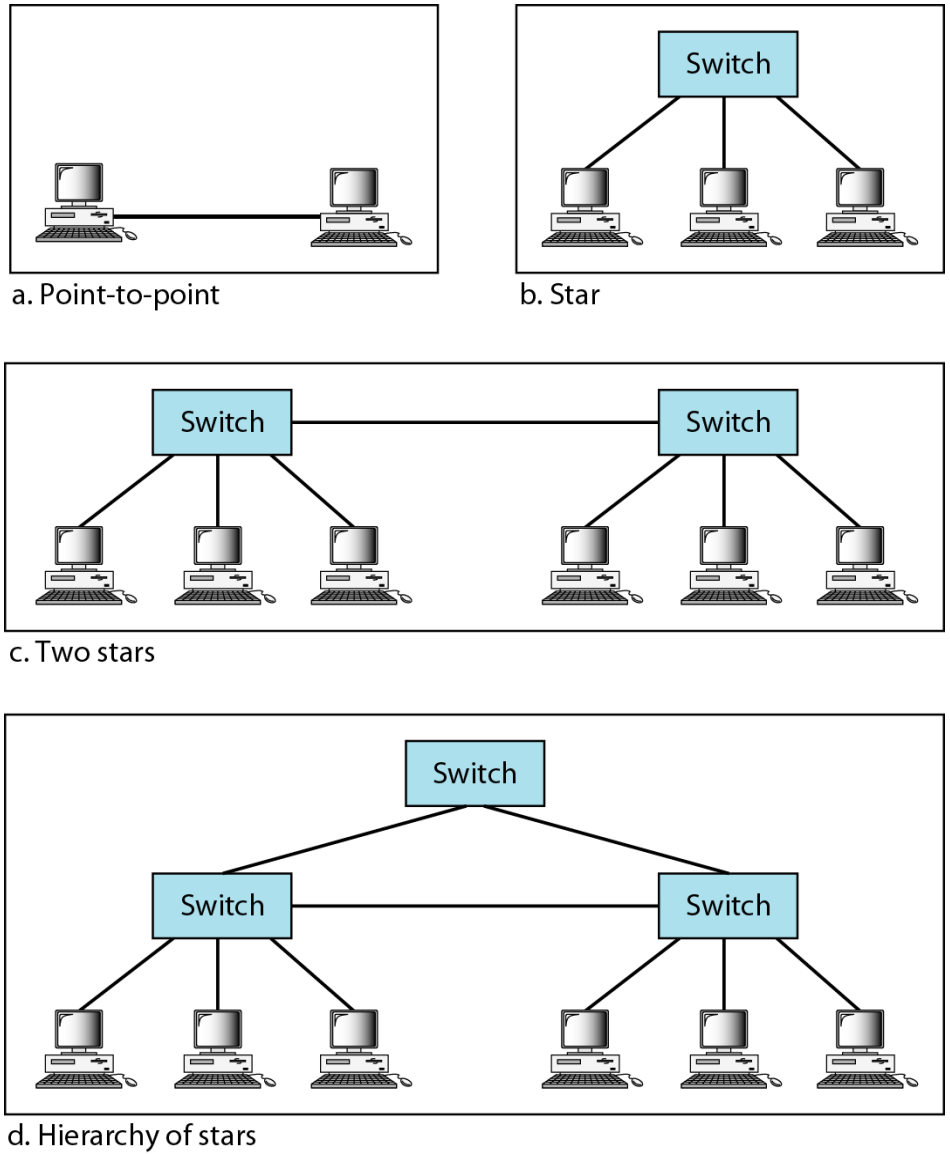
Topics discussed in this section:

MAC Sublayer
Physical Layer
Ten-Gigabit Ethernet

Note

In the full-duplex mode of Gigabit Ethernet, there is no collision; the maximum length of the cable is determined by the signal attenuation in the cable.

Figure 13.22 *Topologies of Gigabit Ethernet*



a. Point-to-point

b. Star

c. Two stars

d. Hierarchy of stars

Figure 13.23 *Gigabit Ethernet implementations*

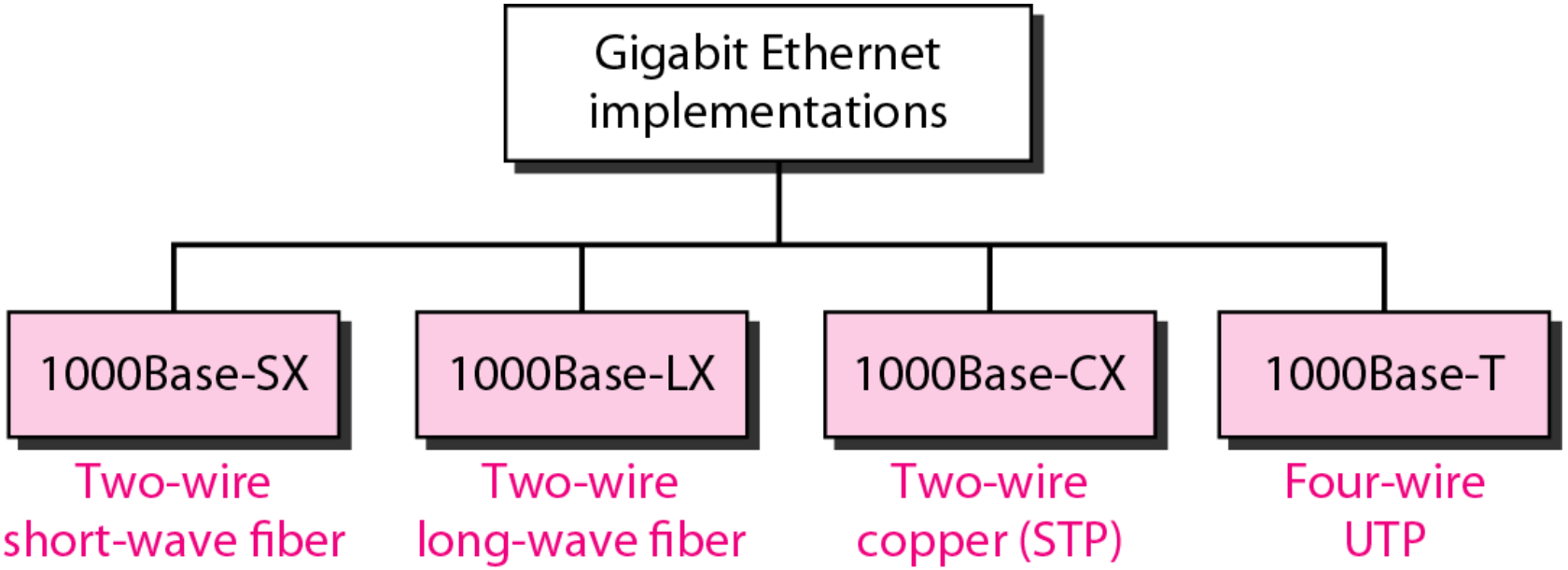


Figure 13.24 *Encoding in Gigabit Ethernet implementations*

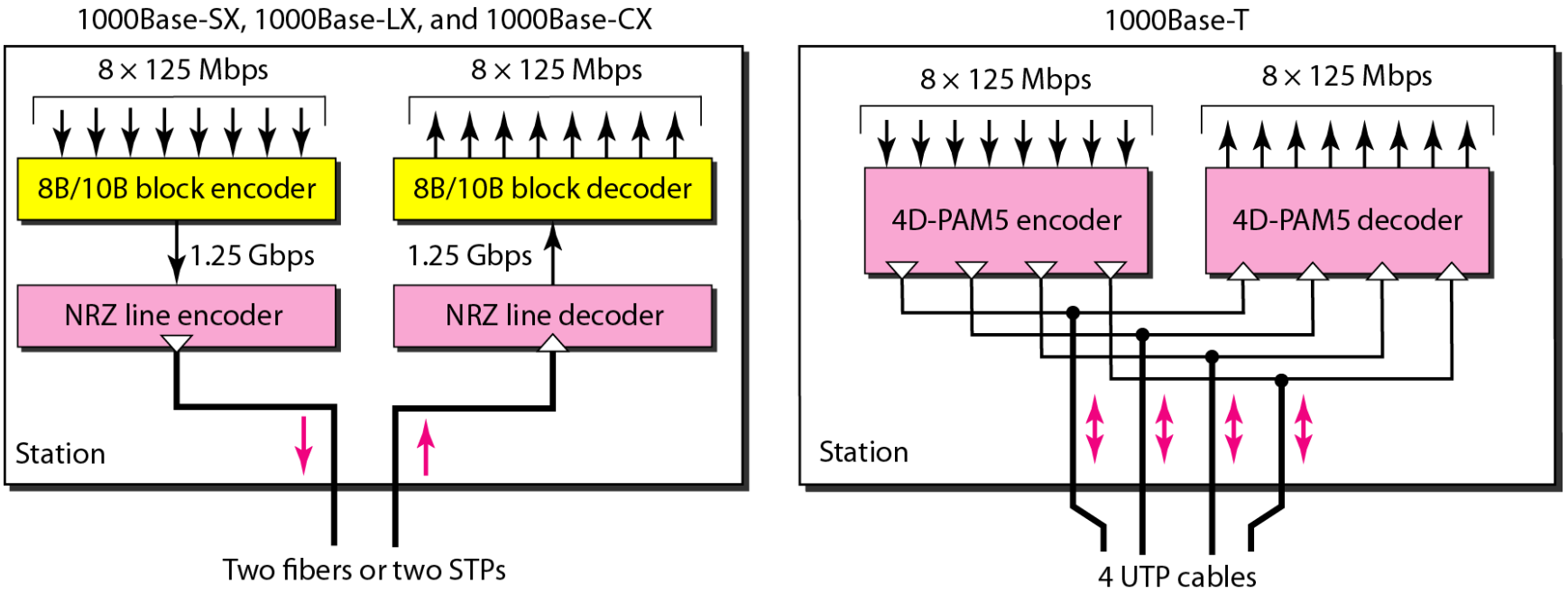


Table 13.3 *Summary of Gigabit Ethernet implementations*

Characteristics	*1000Base-SX*	*1000Base-LX*	*1000Base-CX*	*1000Base-T*
Media	Fiber short-wave	Fiber long-wave	STP	Cat 5 UTP
Number of wires	2	2	2	4
Maximum length	550 m	5000 m	25 m	100 m
Block encoding	8B/10B	8B/10B	8B/10B	
Line encoding	NRZ	NRZ	NRZ	4D-PAM5

Table 13.4 *Summary of Ten-Gigabit Ethernet implementations*

Characteristics	*10GBase-S*	*10GBase-L*	*10GBase-E*
Media	Short-wave 850-nm multimode	Long-wave 1310-nm single mode	Extended 1550-mm single mode
Maximum length	300 m	10 km	40 km