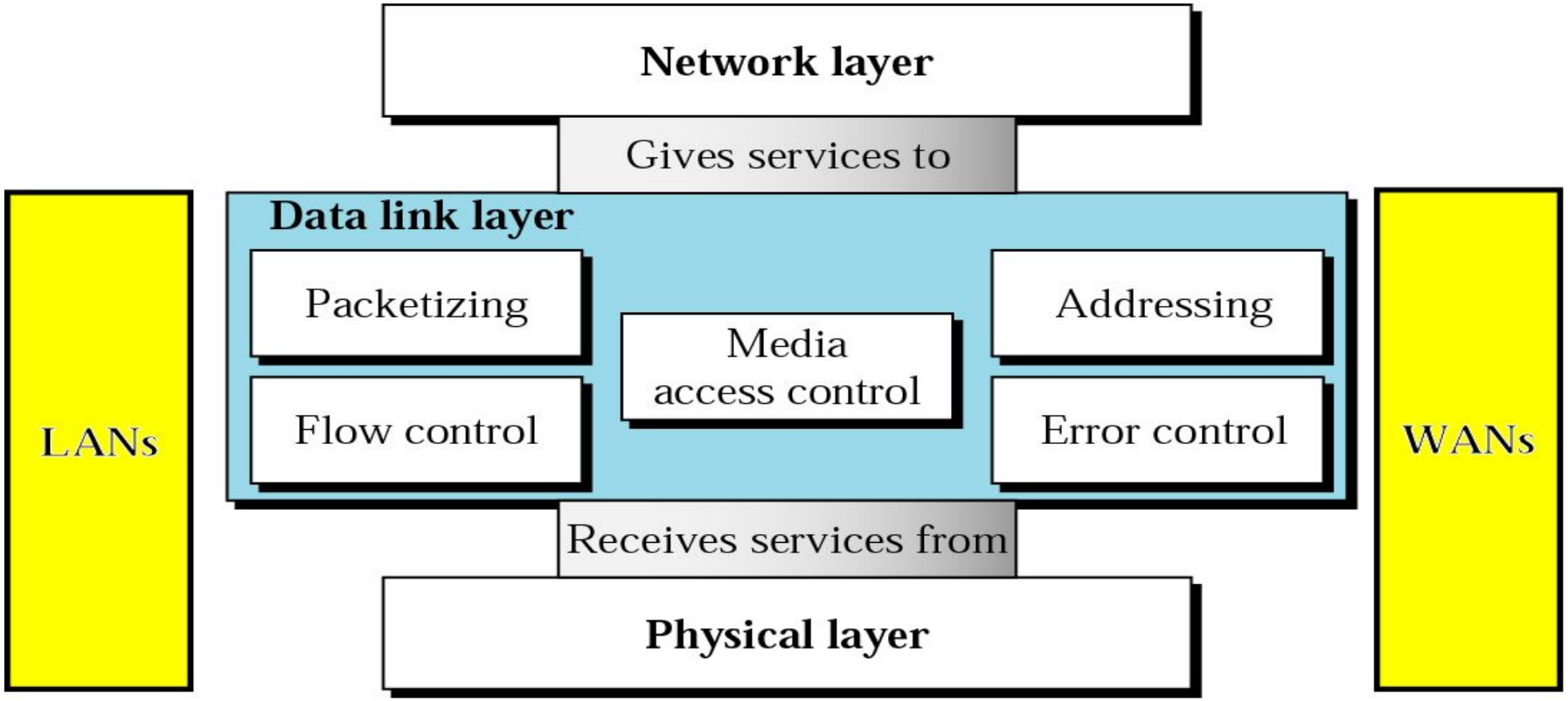
Position of the data-link layer



Chapter 10

Error Detection and Correction

Error Detection and Correction

Note

- Data can be corrupted during transmission.
  - Some applications require that errors be detected and corrected.
- Error Detection and Correction are implemented

  at the data link layer and the transport layer of the

  internet model

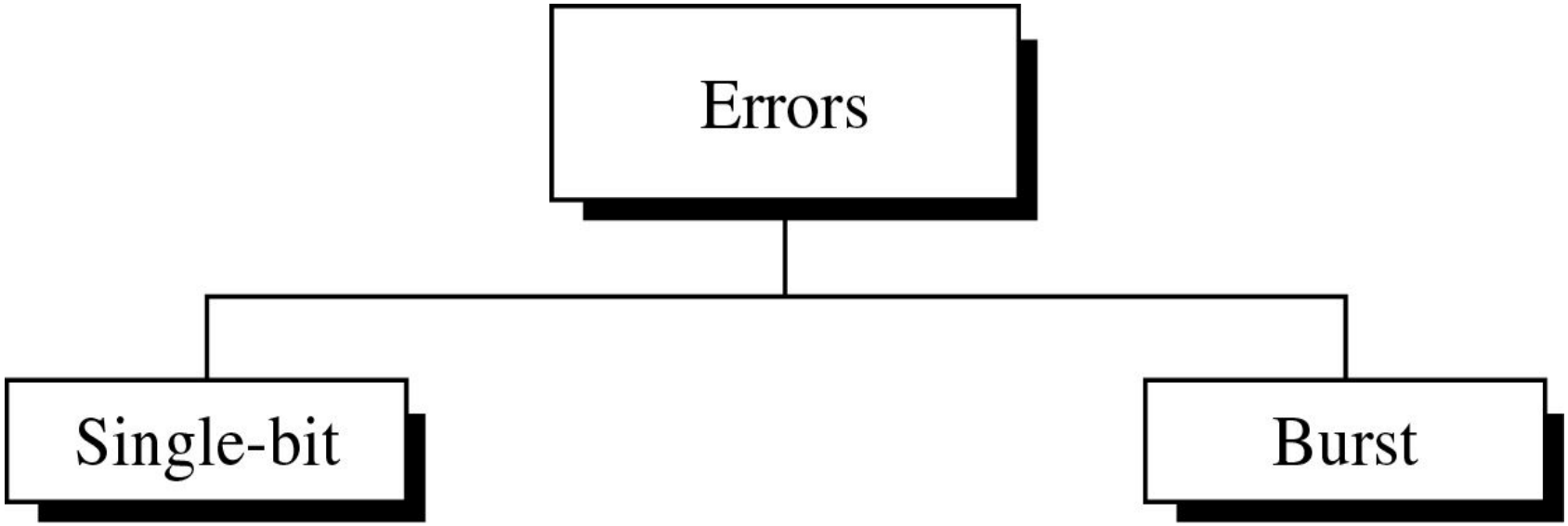
10.1 INTRODUCTION

Let us first discuss some issues related, directly or indirectly, to error detection and correction.

Topics discussed in this section:

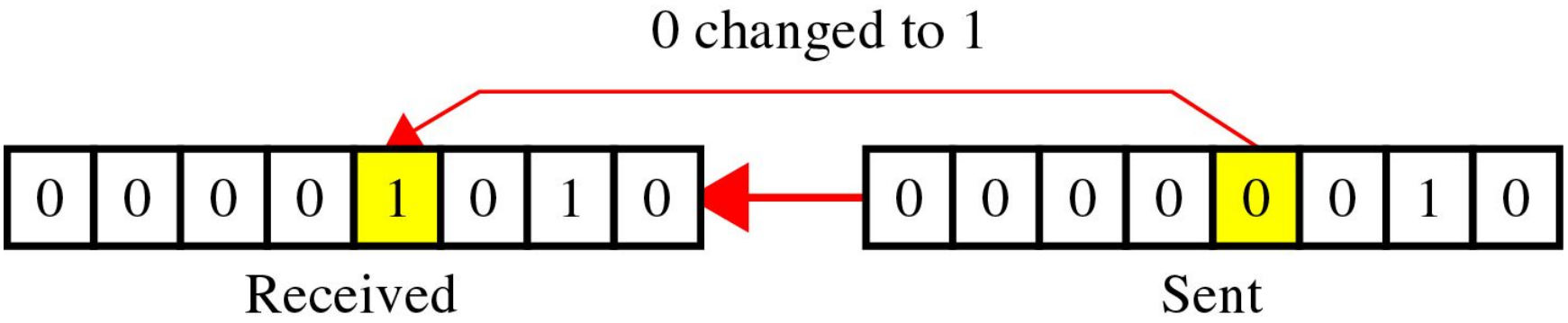
Types of Errors
Redundancy
Detection Versus Correction
Forward Error Correction Versus Retransmission
Coding
Modular Arithmetic

10.1 Type of Errors



Type of Errors(cont'd)

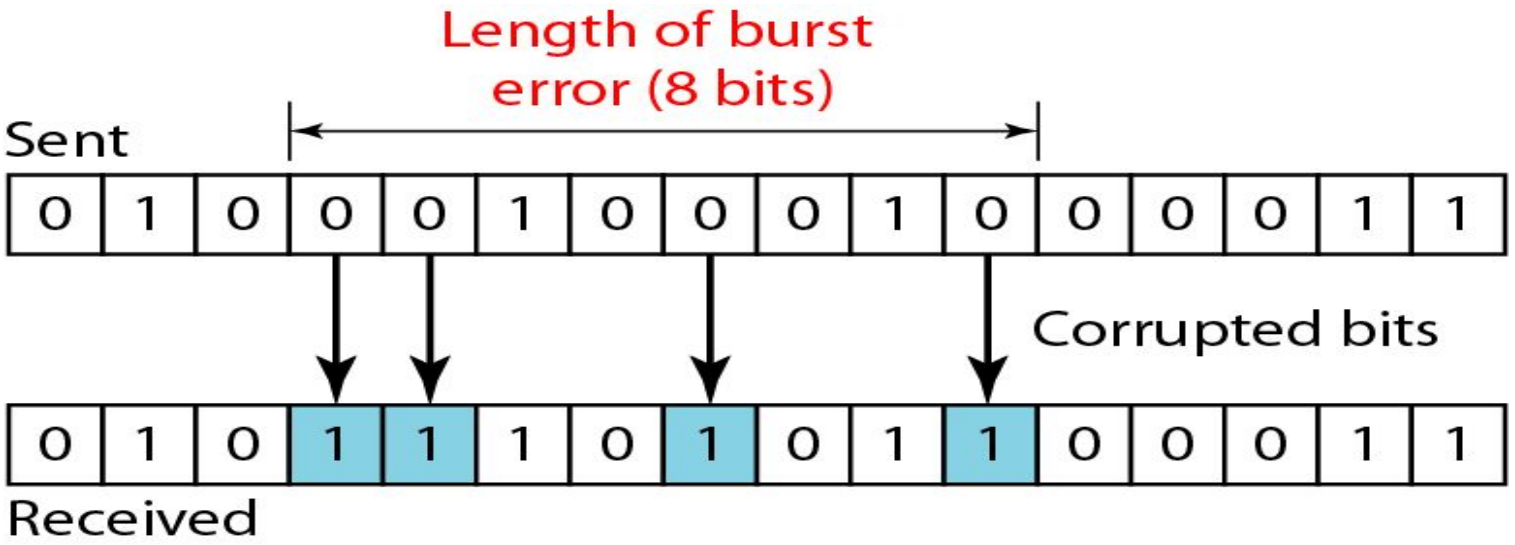
- **Single-Bit Error**



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

Type of Errors(cont'd)

Burst-Bit Error



- A burst error means that 2 or more bits in the data unit have changed.
- The length of the burst is measured from the 1st corrupted bit to the last corrupted bit.
- Some bits in between may not have been corrupted.

Redundancy

- The central concept in detecting or correcting errors is Redundancy.
- Instead of repeating the entire data stream, a shorter group of bits may be appended to the end of each unit.
  → This technique is called Redundancy.
  - These extra bits are discarded as soon as the accuracy of the transmission has been determined.

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

Error Detection Vs Correction

- Error detection
  - looking only to see if any error has occurred.
- Error correction
  - Need to know the exact number of bits that are corrupted and more importantly, their location in the message.
- The correction of errors is more difficult than the detection.

Forward Error Correction Vs Retransmission

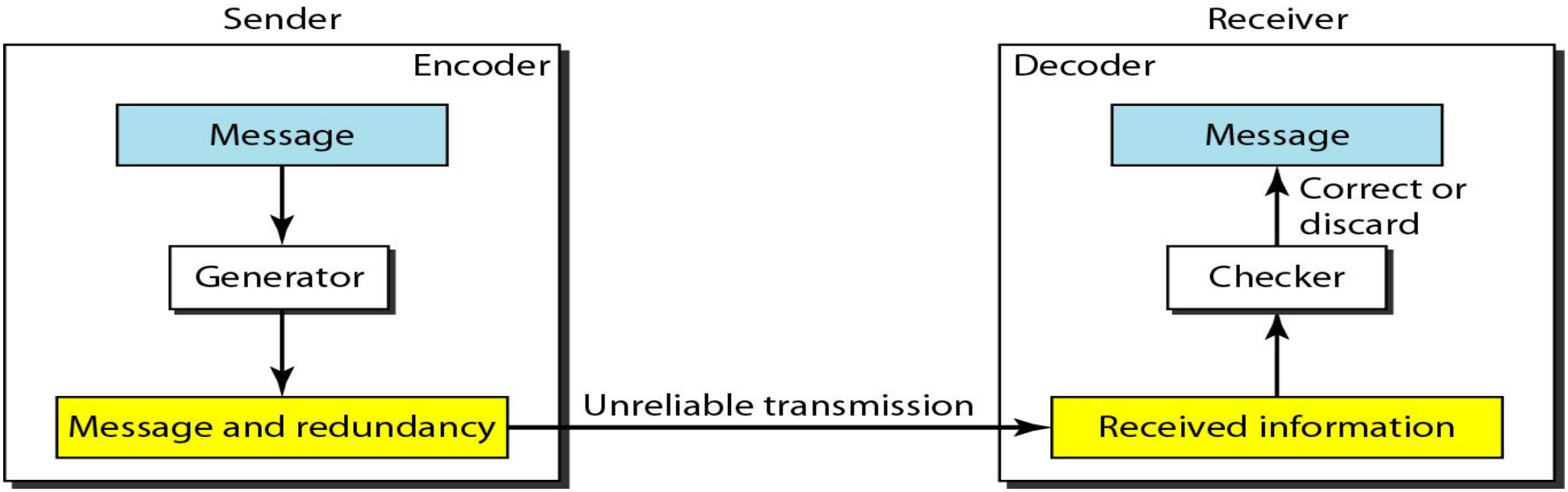
- **Forward Error Correction**
  - **is the process in which the receiver tries to guess the message by using redundant bits.**
- **Retransmission**
  - **is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message.**

Coding

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

Coding

Figure 10.3 *The structure of encoder and decoder*



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

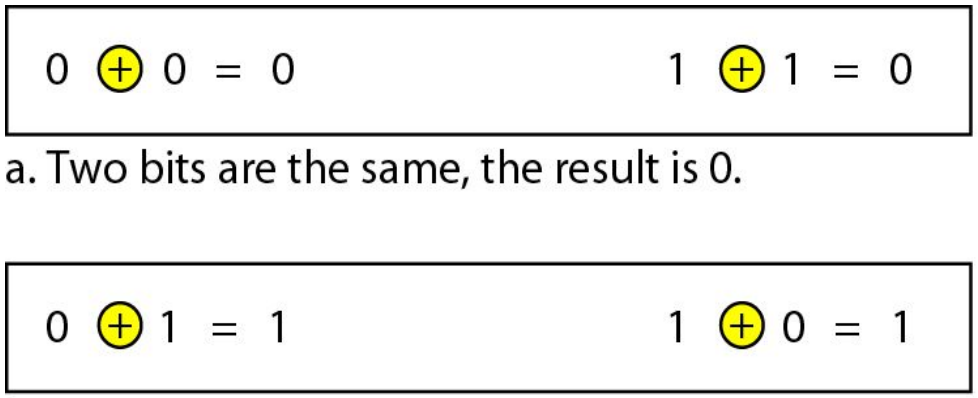
Modular Arithmetic

- In modulo-N arithmetic, we use only the integers in the range 0 to N −1, inclusive.
  - We define an upper limit, called a Modulus N.
  - We then use only the integers 0 to N −1, inclusive.

Modulo-2 Arithmetic

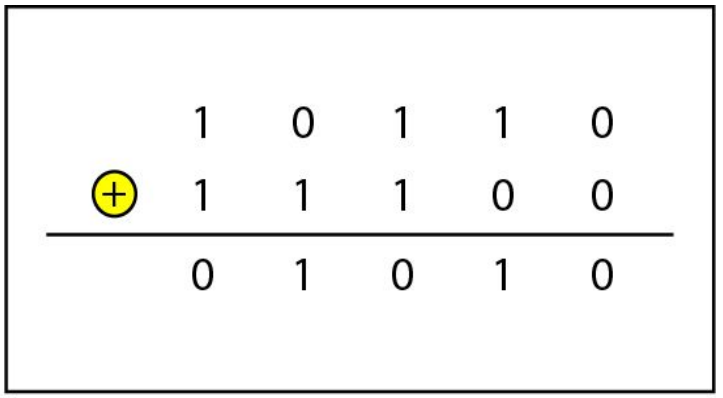
- Adding 0+0=0 0+1=1 1+0=1 1+1=0
- Subtracting 0-0=0 0-1=1 1-0=1 1-1=0

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

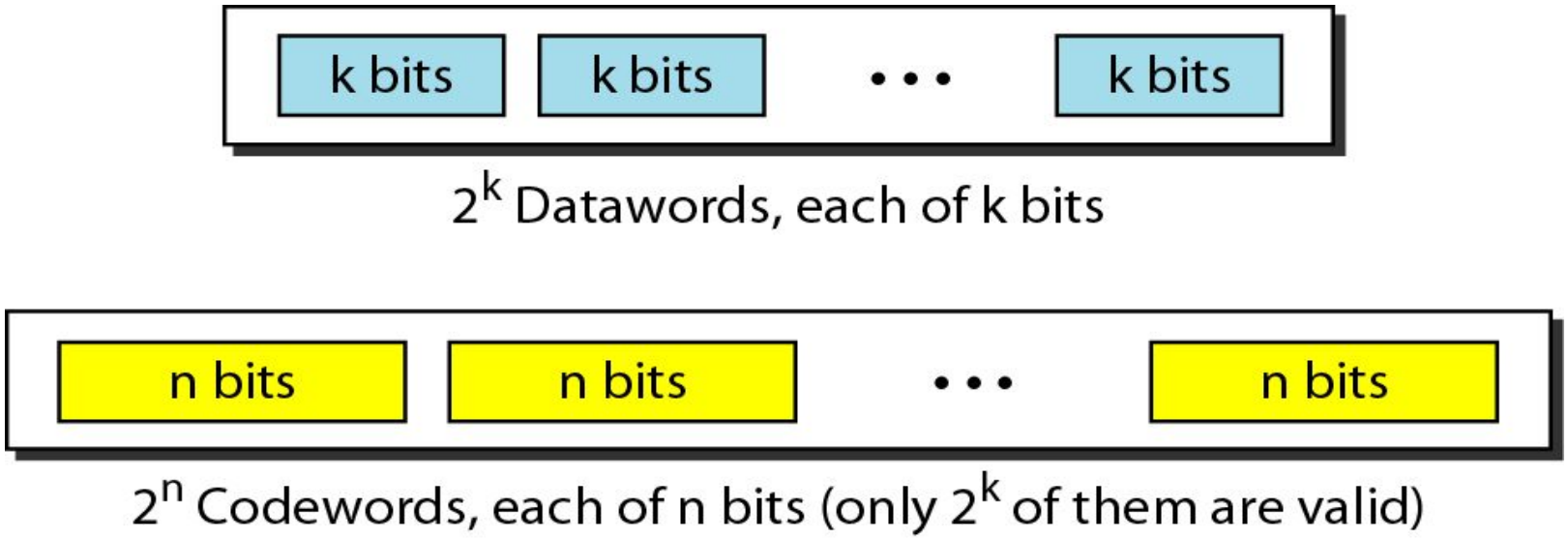
In block coding, we divide our message into blocks, each of k bits, called datawords We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords

Topics discussed in this section:

Error Detection
Error Correction
Hamming Distance
Minimum Hamming Distance

Block Coding

Figure 10.5 *Datawords and codewords in block coding*



Block Coding

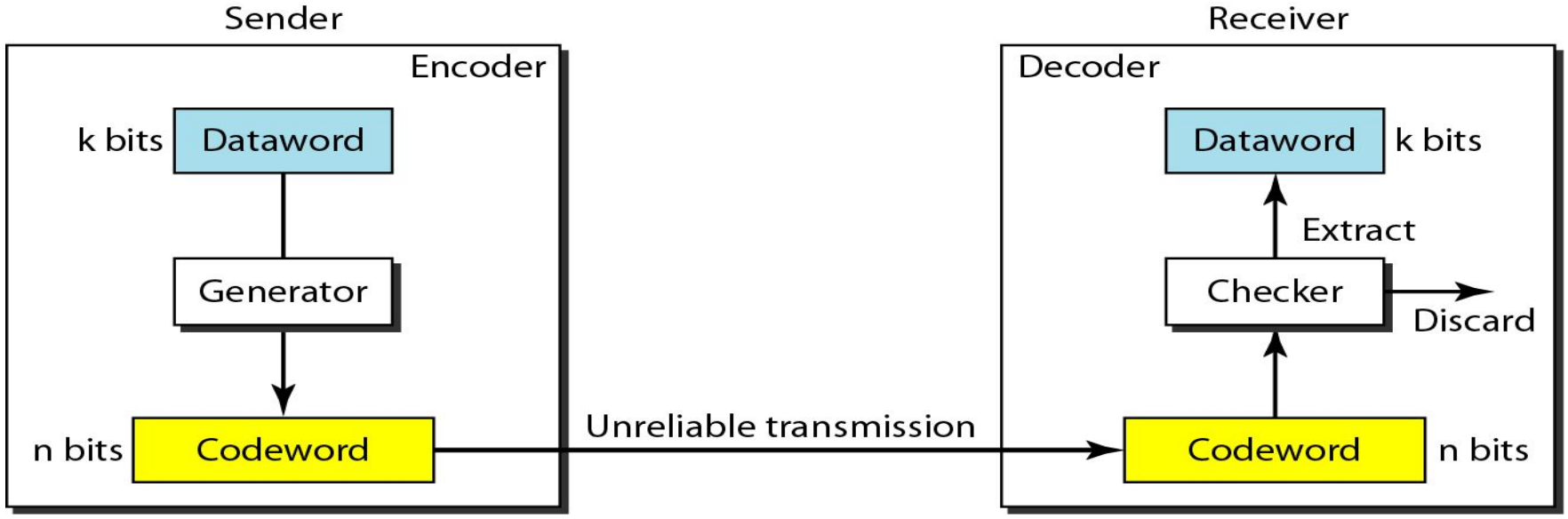
Example 10.1

The 4B/5B block coding discussed in Chapter 4 is a good example of this type of coding. In this coding scheme,
k = 4 and n = 5. As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.

BLOCK CODING - Error detection

- If the following two conditions are met, the receiver can detect a change in the original codeword.
  - The receiver has (or can find) a list of valid codewords.
  - The original codeword has changed to an invalid one.

Figure 10.6 *Process of error detection in block coding*



BLOCK CODING - Error detection

Example 10.2

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

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

Let us assume that k = 2 and n = 3. Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

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

BLOCK CODING - Error detection

Example 10.2 (continued)

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.

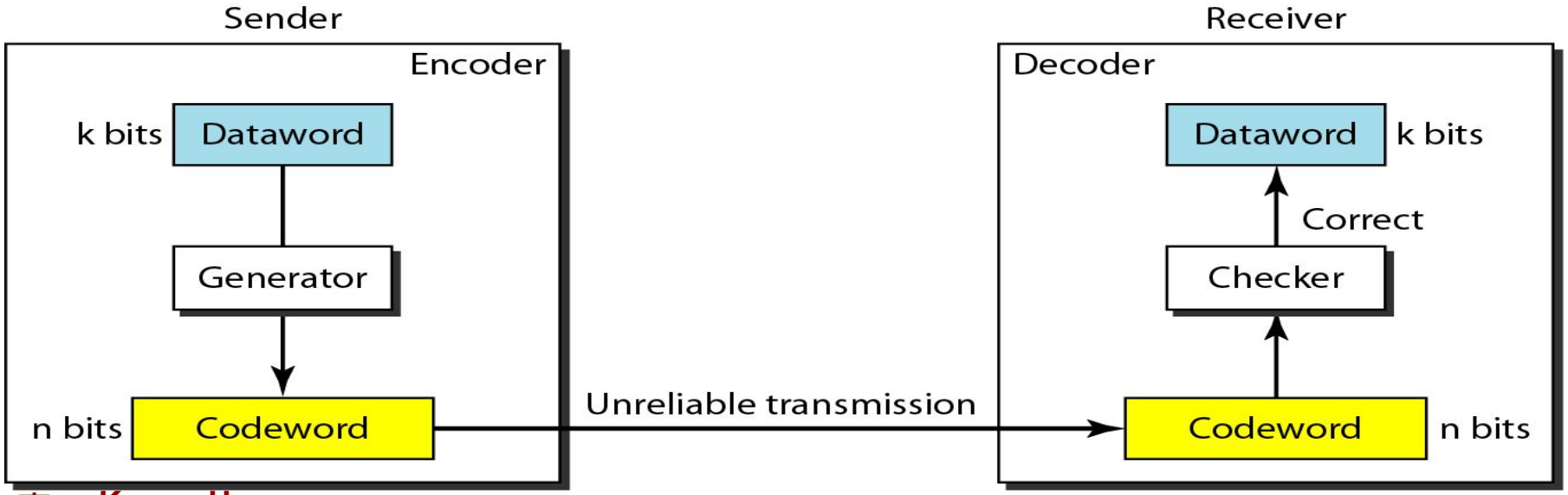
Table 10.1 *A code for error detection (Example 10.2)*

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

BLOCK CODING - Error Correction

- In error correction, the receiver needs to find (or guess) the original codeword sent.
- We need more redundant bits for error correction than for error detection.

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



BLOCK CODING - 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. Assume the dataword is 01. The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received. First, the receiver finds that the received codeword is not in the table. This means an error has occurred. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

BLOCK CODING - Error Correction

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

BLOCK CODING - Hamming Distance

- **Hamming distance between two words (of the same size) is the number of difference between the corresponding bits.**
  - **We show the Hamming distance between two words x and y as d(x,y)**
  - **The Hamming distance can easily be found if we apply the XOR operation ($\oplus$)on the two words and count the number of 1_s in the result.**

BLOCK CODING - Error Correction

Example 10.4

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

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

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

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

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

BLOCK CODING - Minimum Hamming Distance

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

  - **We use d_{min} to define the minimum Hamming distance in a coding scheme.**

BLOCK CODING - Minimum Hamming Distance

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.

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

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

BLOCK CODING - Minimum Hamming Distance

Example 10.6

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

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.

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

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

BLOCK CODING - Hamming Distance

- **Three parameters**
  - A coding scheme C is written as C(n,k) with a separate expression for d_{min} .
  - For example, we can call our first coding scheme c(3,2) with d_{min} =2 and our second coding scheme c(5,2) with d_{min} =3.
- **Hamming distance and Error**
  - The Hamming distance between the received cordword and the sent codeword is the number of bits that are corrupted during transmission.
  - For example : 3 bits are in error and the hamming distance between the two is d(00000, 01101) = 3

BLOCK CODING - Minimum Distance for error detection

- To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.
  - so that the received codeword does not match a valid codeword.

BLOCK CODING - Minimum Distance for error detection

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.

- one error : 101 □ 100 (does not match any valid code)
- Two errors : 101 □ 000 (errors are not detected)

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

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

BLOCK CODING - Minimum Distance for error detection

Example 10.8

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

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

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

10.3 LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called linear block codes. A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Topics discussed in this section:

Minimum Distance for Linear Block Codes
Some Linear Block Codes

Linear Block Codes

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.**

Table10.1

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

Table10.2

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

Minimum Distance for Linear Block Codes

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

Minimum Distance for Linear Block Codes

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

Table10.1

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

Table10.2

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

Linear Block Codes-Simple parity-Check Code

- In Simple parity-check code, a k-bit dataword is changed to an n-bit codeword where n=k+1.
- The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even or odd.
- This code is a single-bit error-detecting code; it cannot correct any error.

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

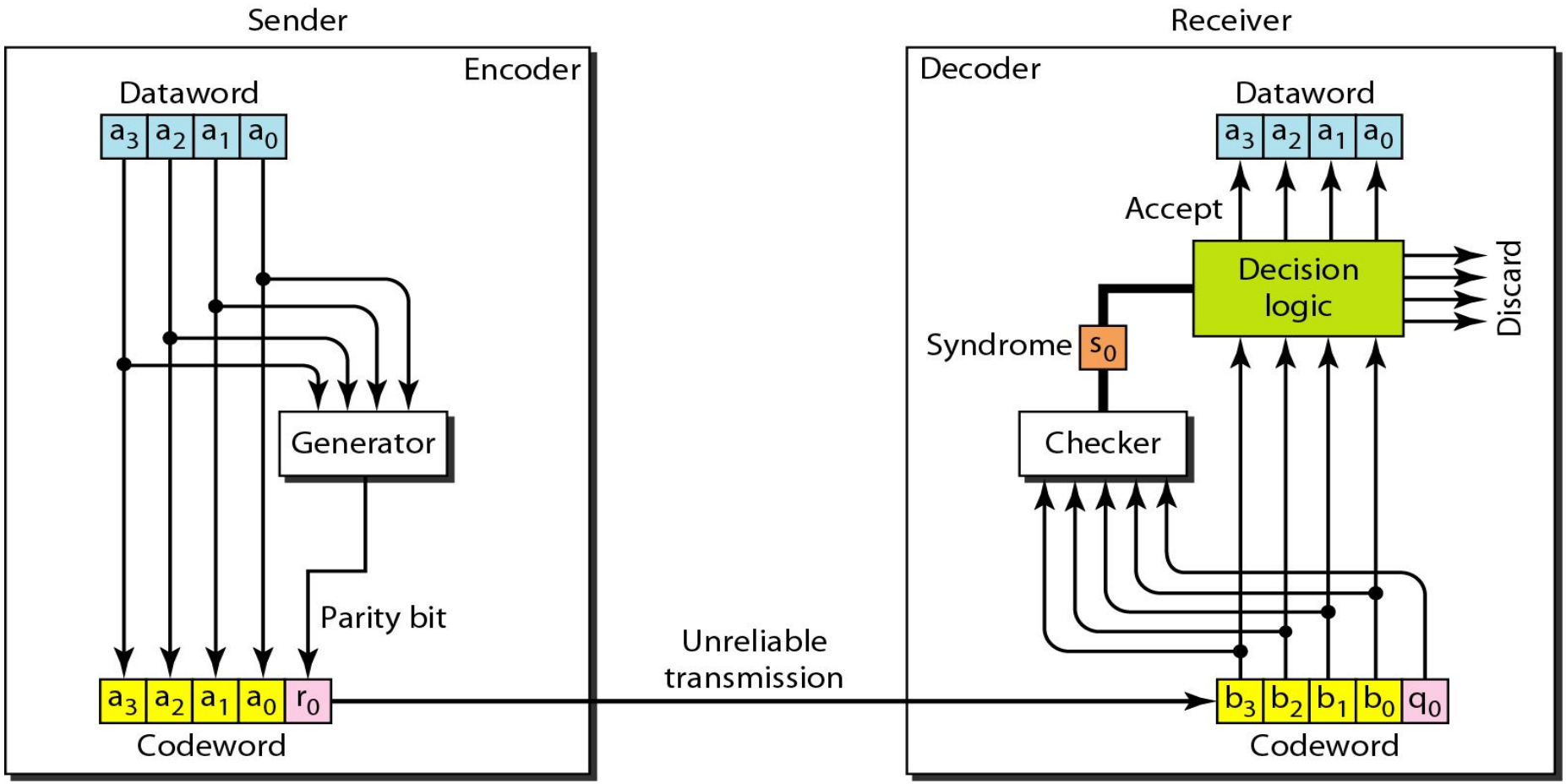
Minimum Distance for Linear Block Codes

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

Minimum Distance for Linear Block Codes

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



Minimum Distance for Linear Block Codes

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.

Minimum Distance for Linear Block Codes

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. Note that 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

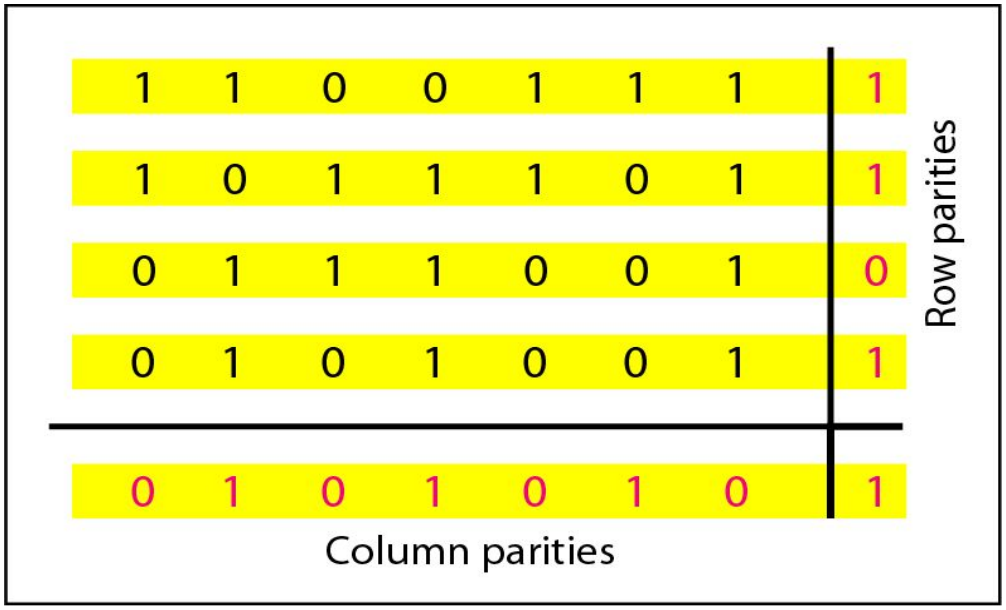
A simple parity-check code can detect an odd number of errors.

LINEAR BLOCK CODES - Two-dimensional parity Check Code

- In Two-dimensional parity-check code, the data code is organized in a table (rows and columns).
  - In fig.10.11, the data to be sent, five 7bit bytes, are put in separate rows.
  - For each row and each column, 1 parity-check bit is calculated.
  - The whole table is then sent to the receiver, which finds the syndrome for each row and each column.
  - As Fig.10.11 shows, the two-dimensional parity check can detect up to three errors that occur anywhere in the table.

LINEAR BLOCK CODES - Two-dimensional parity Check Code

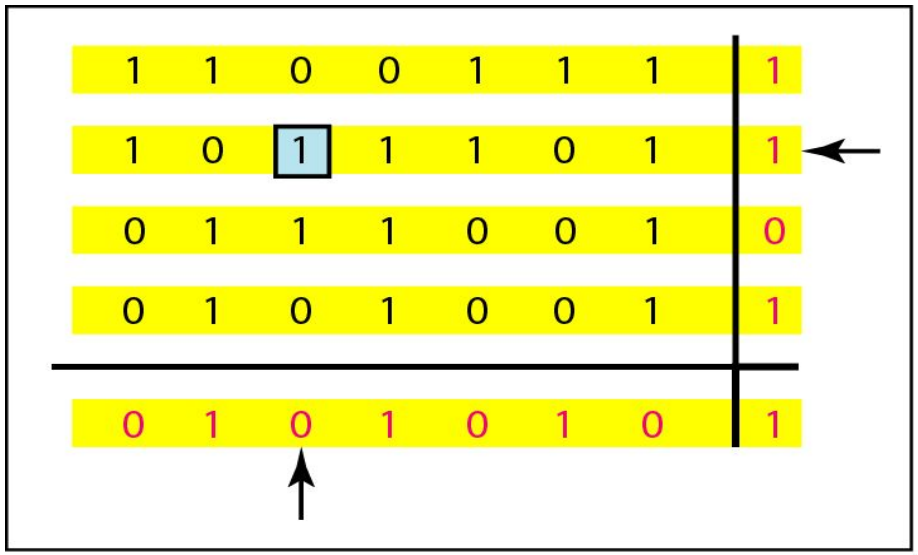
Figure 10.11 *Two-dimensional parity-check code*



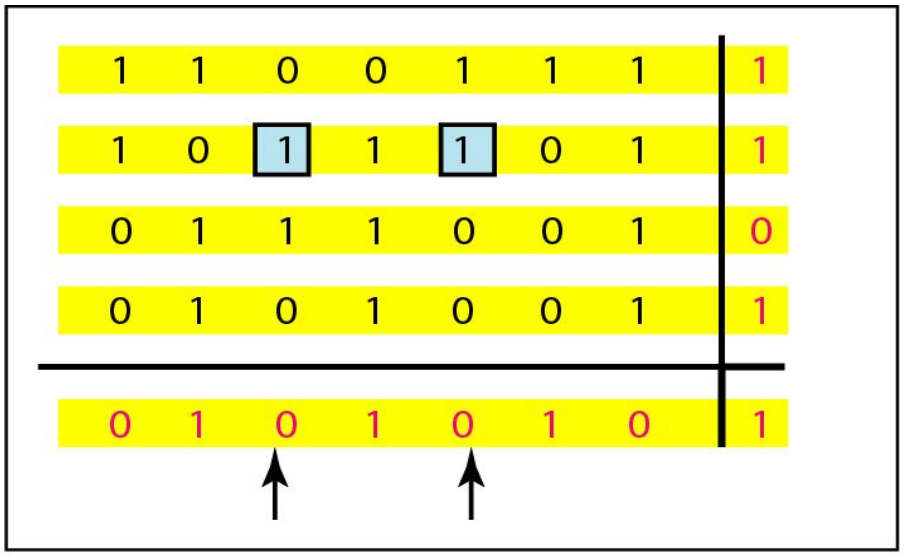
a. Design of row and column parities

LINEAR BLOCK CODES - Two-dimensional parity Check Code

Figure 10.11 *Two-dimensional parity-check code*



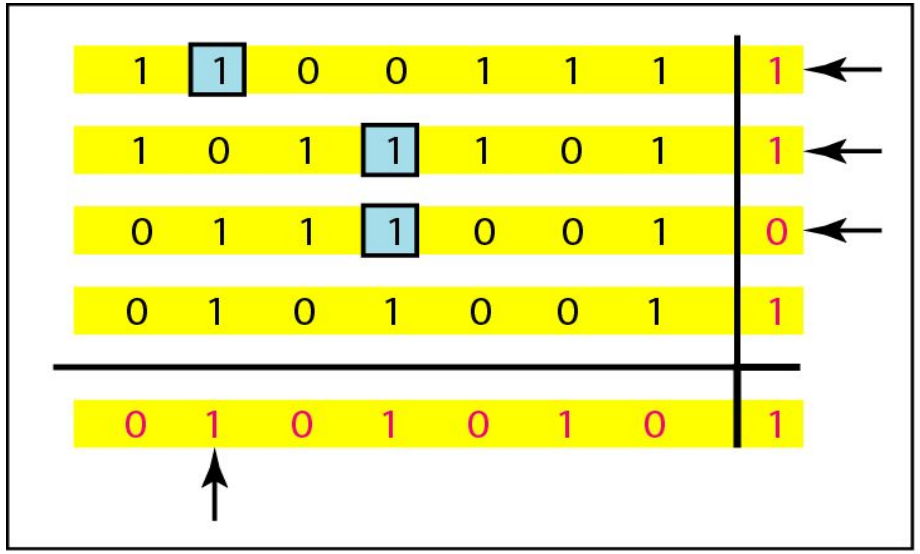
b. One error affects two parities



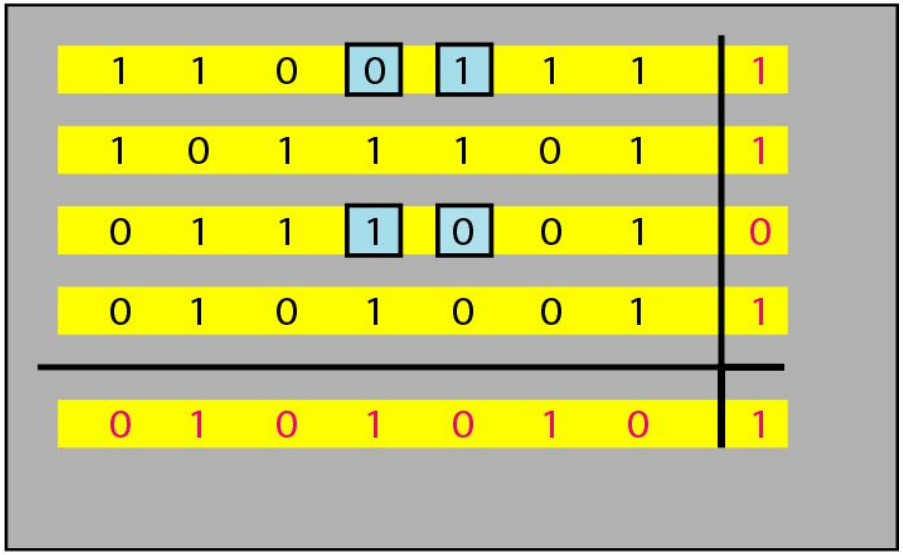
c. Two errors affect two parities

LINEAR BLOCK CODES - Two-dimensional parity Check Code

Figure 10.11 *Two-dimensional parity-check code*



d. Three errors affect four parities



e. Four errors cannot be detected

LINEAR BLOCK CODES - Hamming Codes

- Hamming Codes were originally designed with d_{min} =3, which means that can detect up to two errors or correct one single error.
  - We need to choose an integer m≥3.
  - The value of n and k are then calculated from m as $n=2^m -1$ and k = n - m.
  - The number of check bits r = m.

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

The relationship between *m* and *n* in these codes is $n = 2^m - 1$.

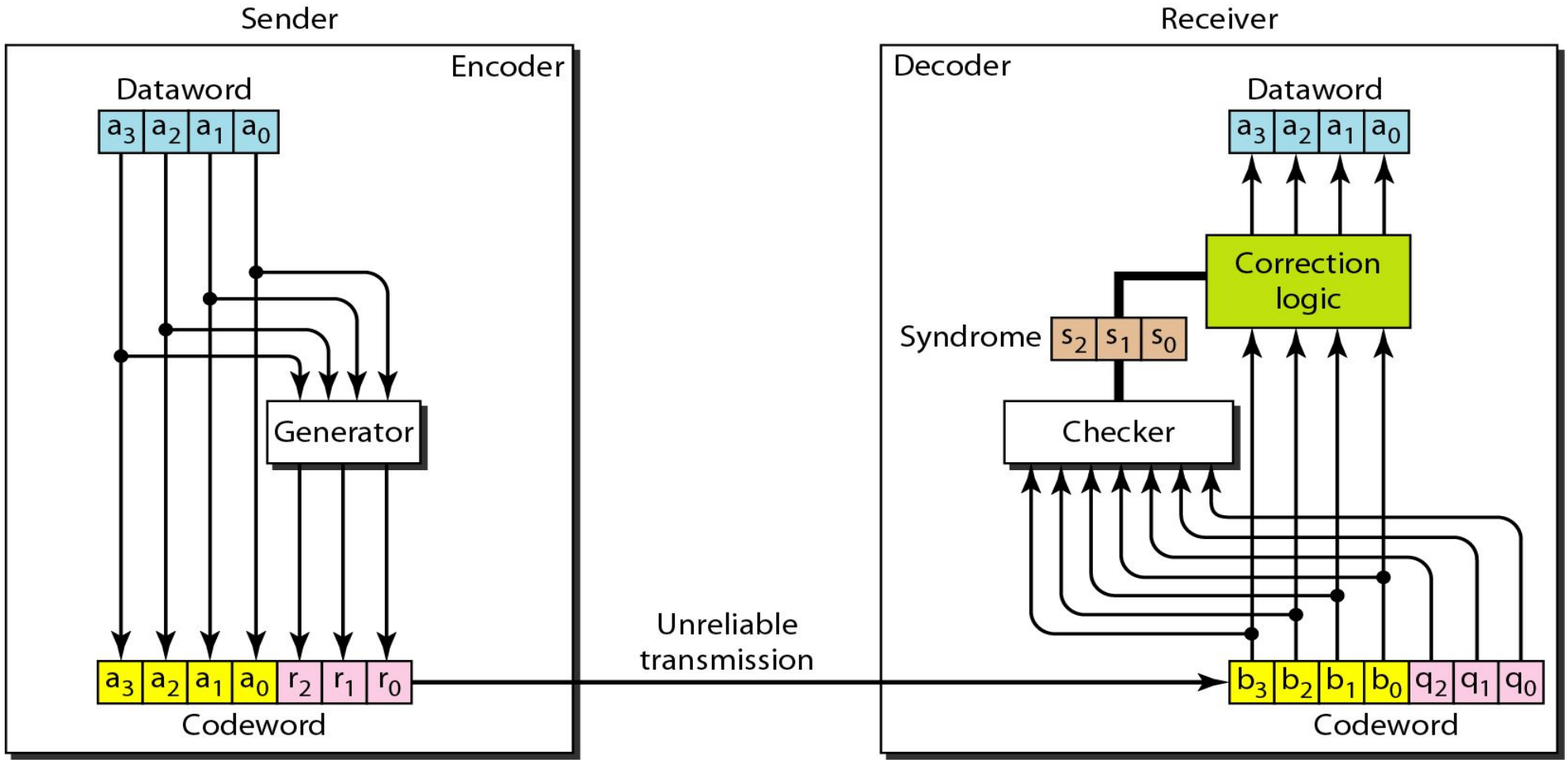
LINEAR BLOCK CODES - Hamming Codes

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

LINEAR BLOCK CODES - Hamming Codes

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



The calculation for a Hamming code

- Generator creates 3 parity-check bit (r_0 ,r_1 , and r_2)
  - $r_0 = a_2 + a_1 + a_0$ modulo-2
  - $r_1 = a_3 + a_2 + a_1$ modulo-2
  - $r_2 = a_1 + a_0 + a_3$ modulo-2

- The checker in the decoder creates a 3bit syndrome bit
  - $s_0 = b_2 + b_1 + b_0 + q_0$ modulo-2
  - $s_1 = b_3 + b_2 + b_1 + q_1$ modulo-2
  - $s_2 = b_1 + b_0 + b_3 + q_2$ modulo-2

LINEAR BLOCK CODES - Hamming Codes

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

- **In the cases unshaded in Table 10.5 ;**

  1 of the bits must be flipped (changed 0 to 1, or 1 to 0) to find the correct dataword.

  - **If syndrome value is 011 : b_2 bit must be changed 1 to 0,**

    or 0 to 1

  - **if b_1 is in error, all 3 syndrome bits affected.**
  - **If syndrome value is 000 : the generator is not concerned, because there is either no error or an error in the parity bit.**

LINEAR BLOCK CODES - Hamming Codes

Example 10.13

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

1. **The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.**
2. **The dataword 0111 becomes the codeword 0111001. The codeword 0011001 is received. 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 code 0001000 is received. 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 a dataword of at least 7 bits. Calculate values of k and n that satisfy this requirement.

Solution

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

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**

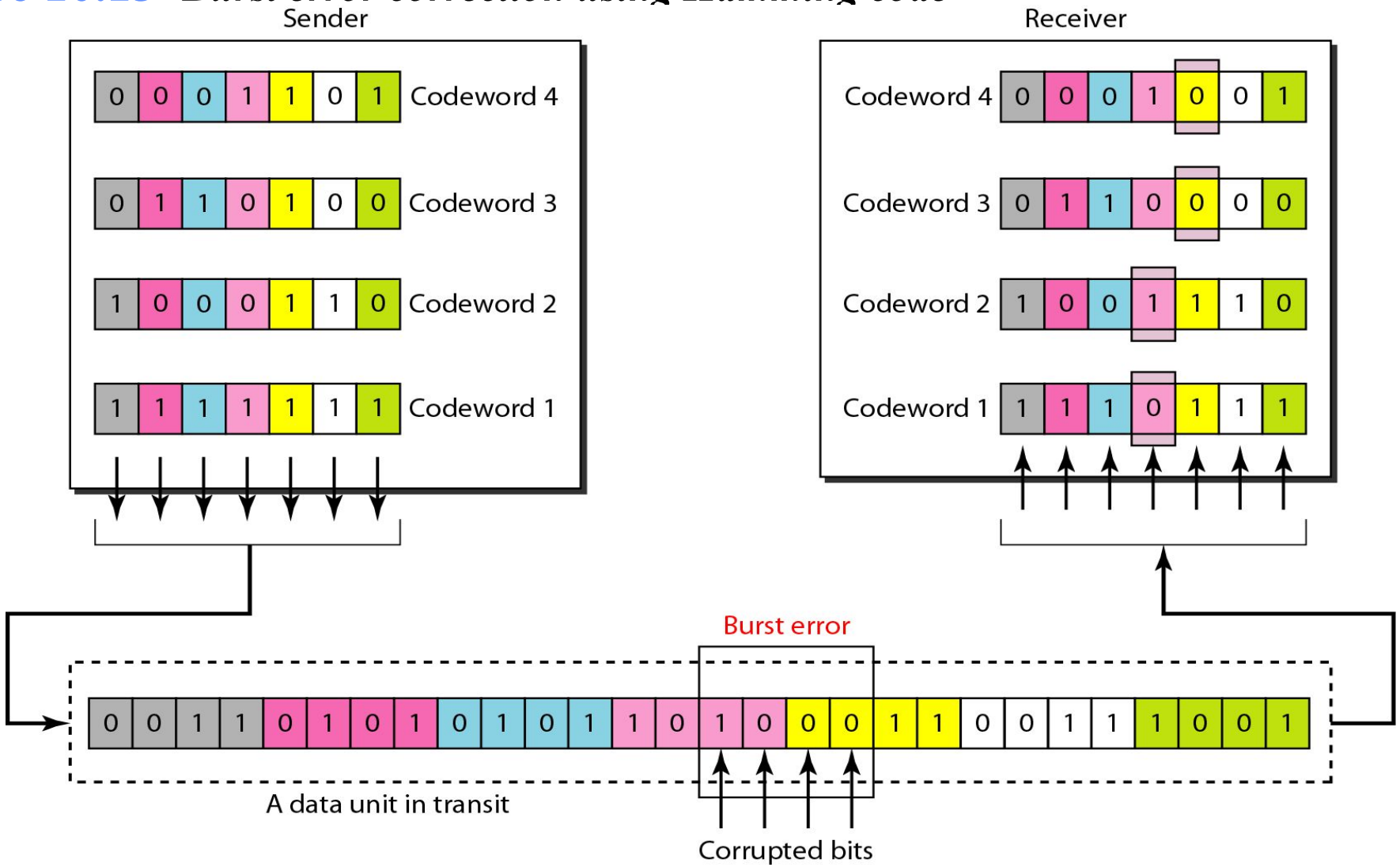
C(15, 11)

Burst error correction using LINEAR BLOCK Hamming code

- A hamming code can only correct a single error or detect a double error.
- However, there is a way to make it correct a burst error.
  - The key is to split a burst error between several codewords, one error for each code.

Burst error correction using LINEAR BLOCK Hamming code

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

Cyclic Redundancy Check

- In a Cyclic Code, if a codeword is cyclically shifted , the result is another codeword.
  - For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.

$$b_1 = a_0,\ b_2 = a_1,\ b_3 = a_2,\ b_4 = a_3,\ b_5 = a_4,\ b_6 = a_5,\ b_0 = a_6$$

  - In the right most equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.
- Cyclic Redundancy Check (CRC) is used in networks such as LANs and WANs.

Cyclic Redundancy Check

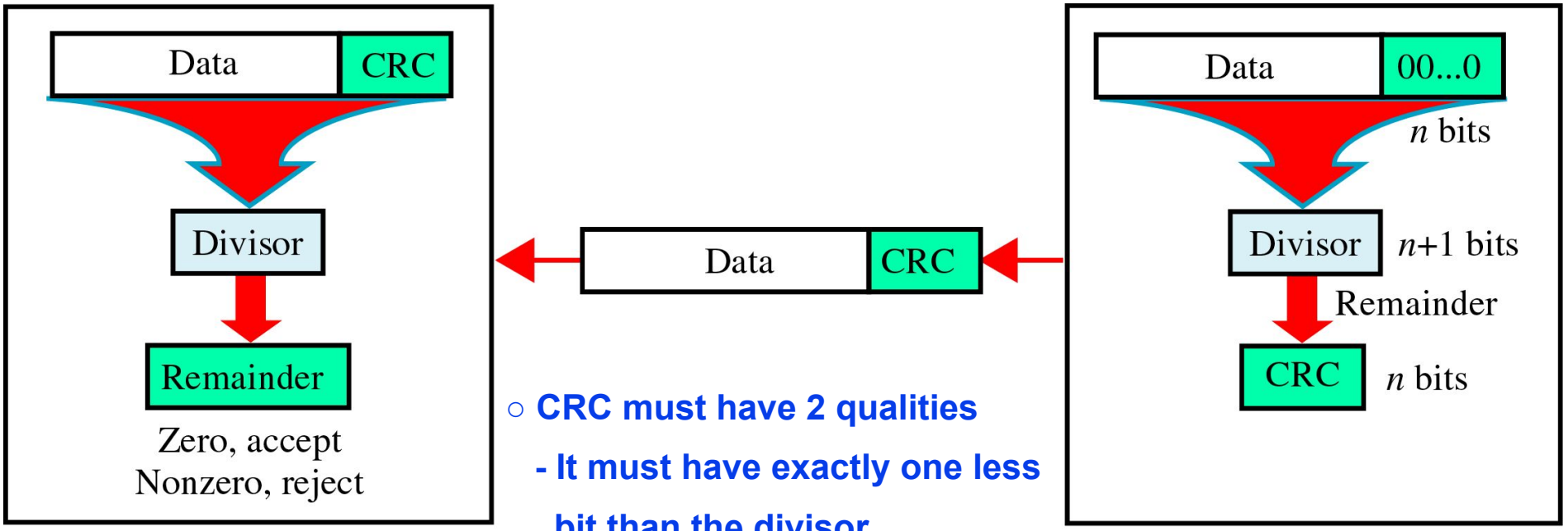
Table 10.6 *A CRC code with C(7, 4)*

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

Cyclic Redundancy Check

- **CRC(Cyclic Redundancy Check)**

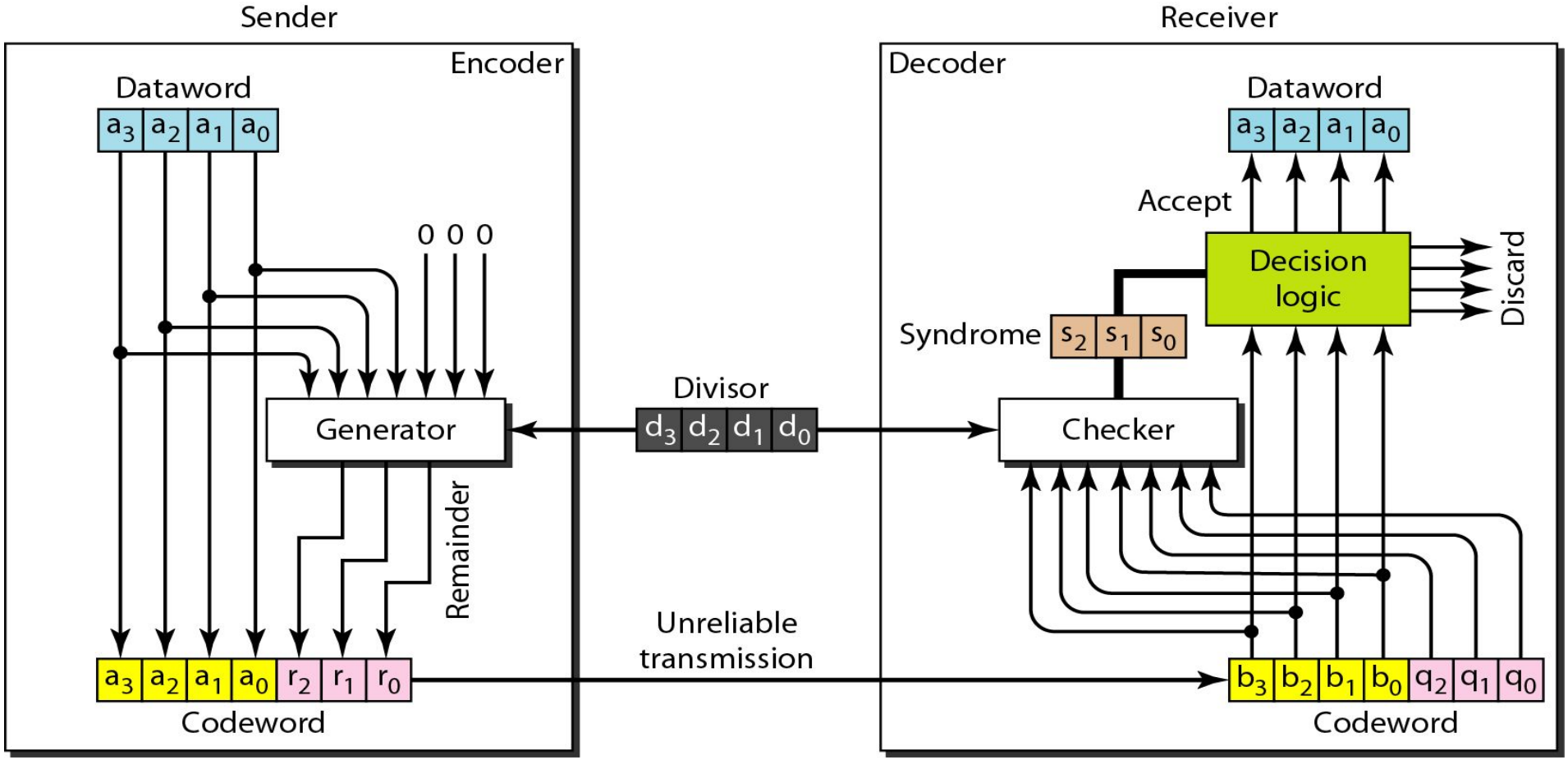
~ is based on binary division.



- **CRC must have 2 qualities**
  - **It must have exactly one less bit than the divisor**
  - **Resulting bit sequence (Data+CRC) exactly divisible by the divisor.**

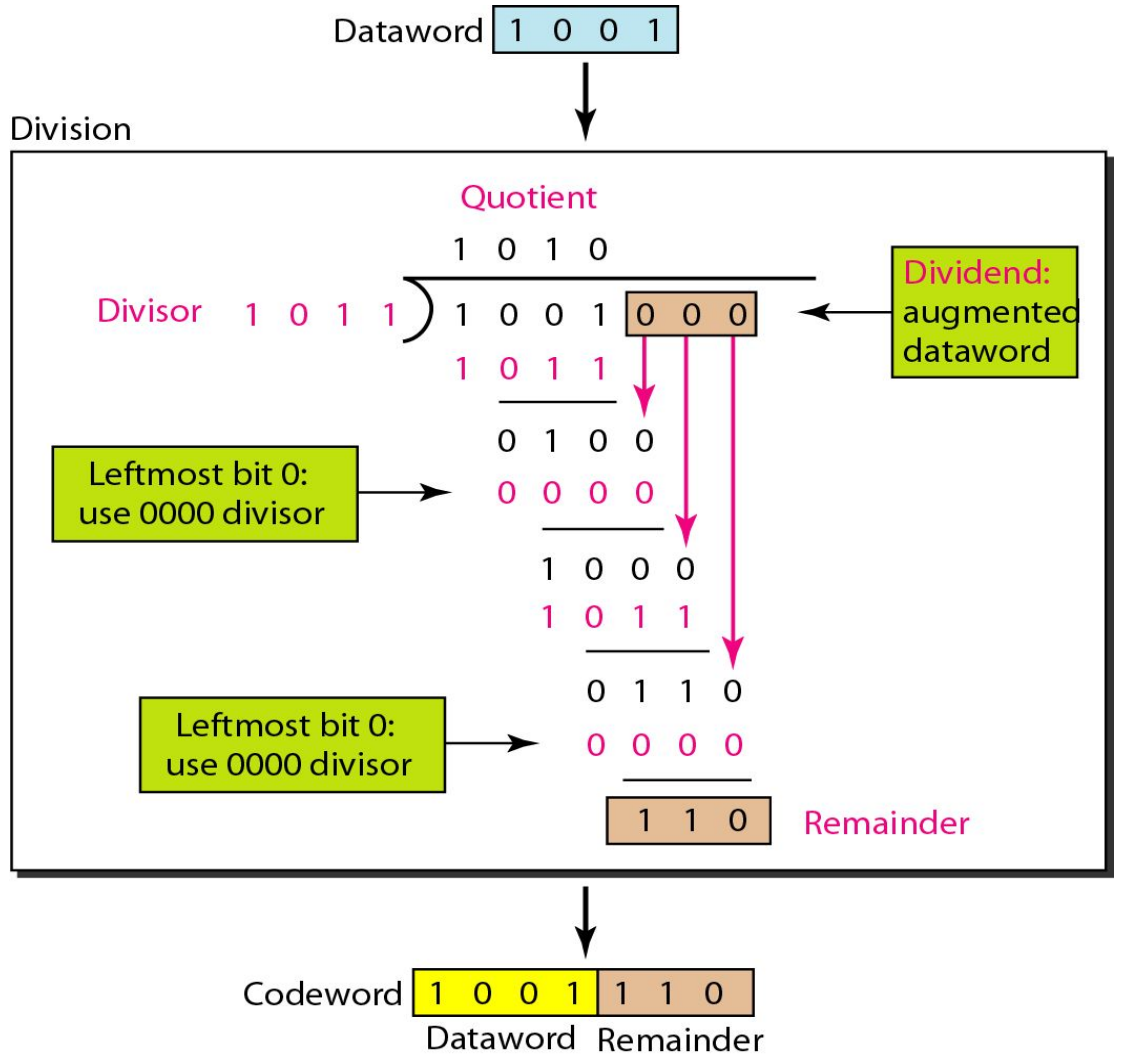
Cyclic Redundancy Check

Figure 10.14 *CRC encoder and decoder*



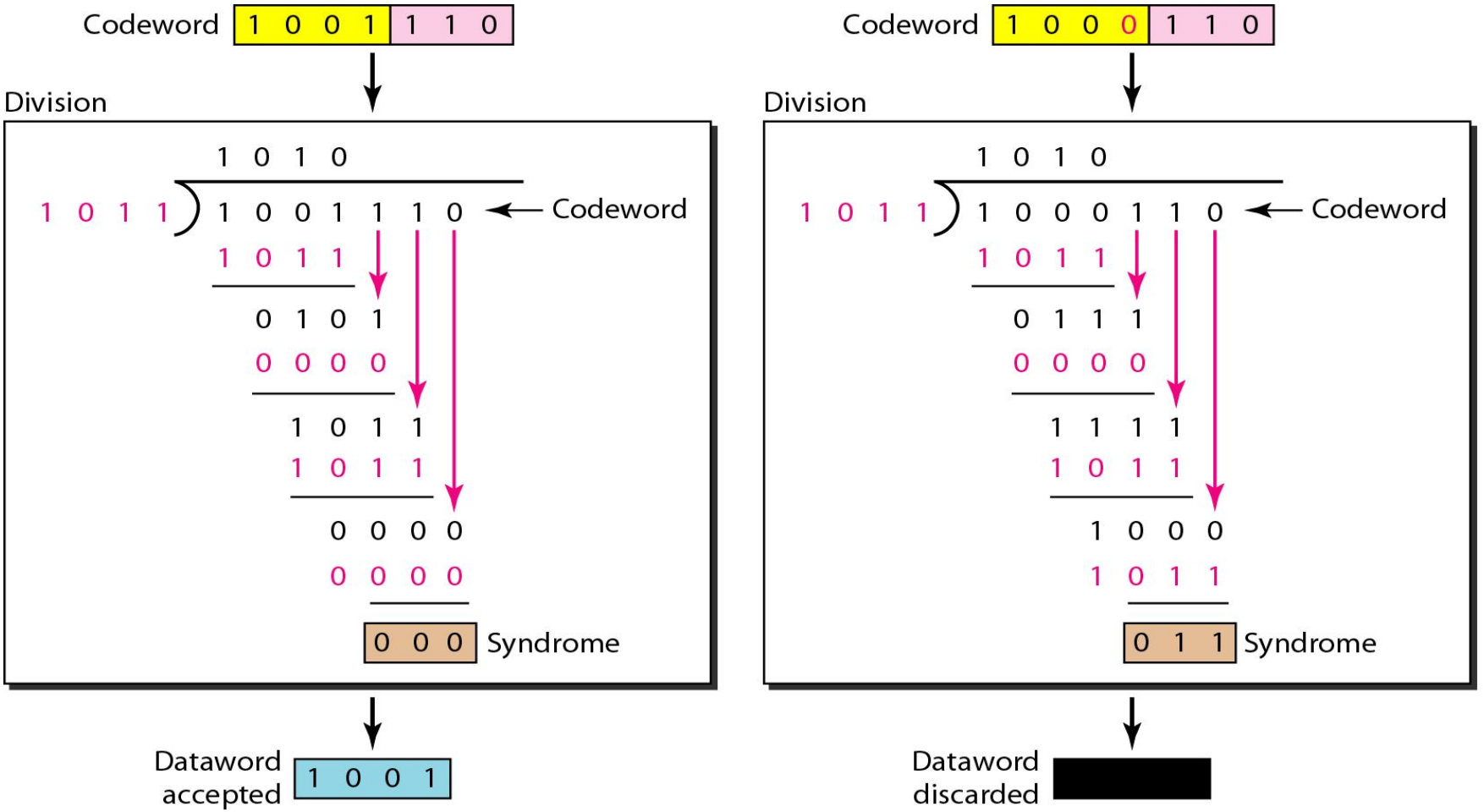
Cyclic Redundancy Check

Figure 10.15 *Division in CRC encoder*



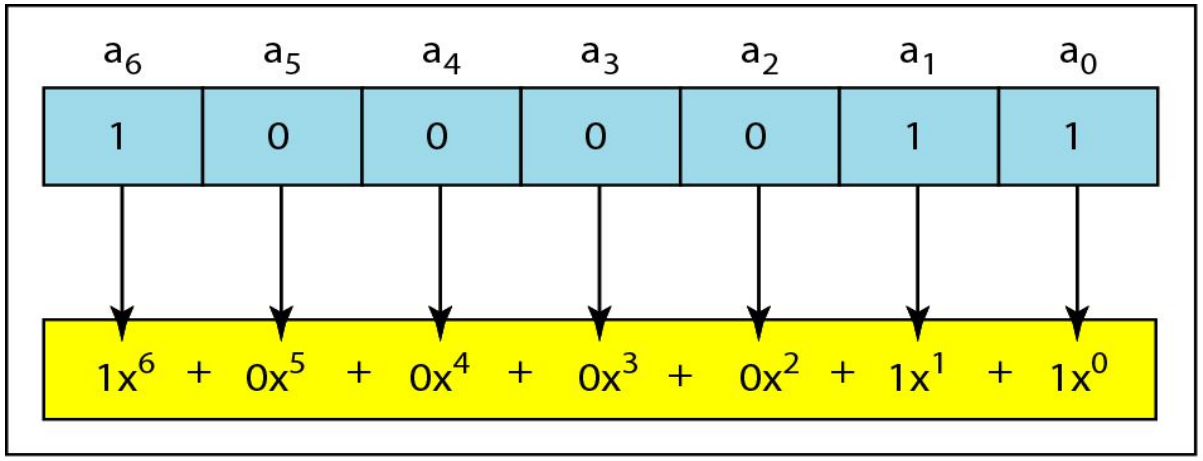
Cyclic Redundancy Check

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

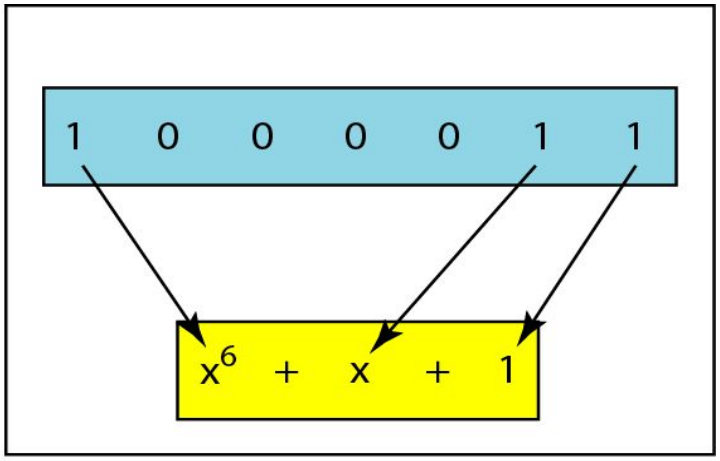


Cyclic Redundancy Check

Figure 10.21 *A polynomial to represent a binary word*



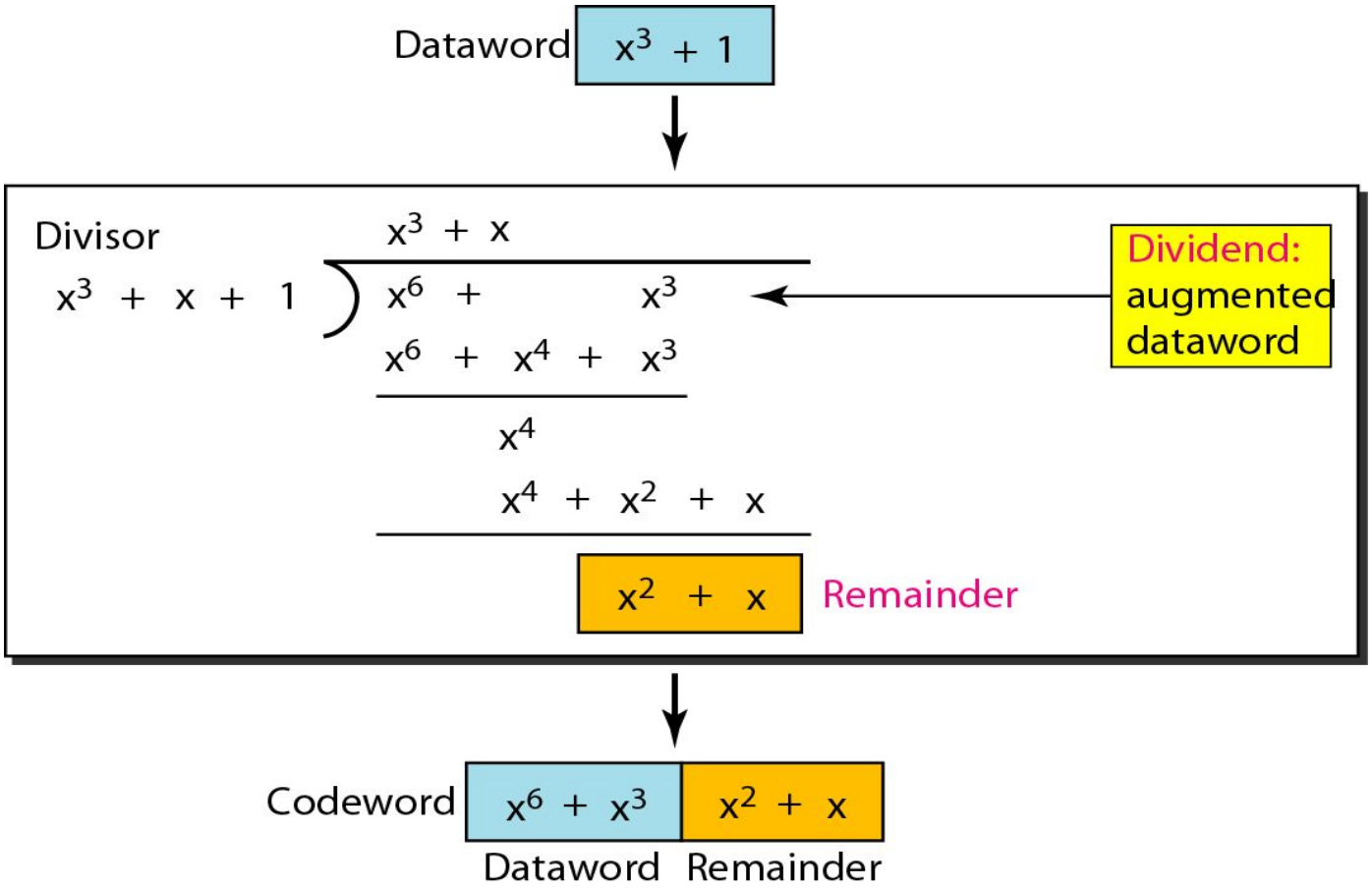
a. Binary pattern and polynomial



b. Short form

Cyclic Redundancy Check

Figure 10.22 *CRC division using polynomials*



Cyclic Redundancy Check

Note

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

Cyclic Redundancy Check

Note

In a cyclic code,

If s(x) ≠ 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.

*** S(X) = Syndrome**

Cyclic Redundancy Check

Note

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

- **Received Code word = transmitted Code word + Error**
  - **{Received Code word = C(X) + e(X)} / g(X)**

Cyclic Redundancy Check

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

Cyclic Code

Advantages of Cyclic Codes

- Cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors.
- They can easily be implemented in hardware and software.
- They are especially fast when implemented in hardware.
- This has cyclic codes a good candidate for many networks.

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

Checksum

- Like linear and cyclic codes, the checksum is based on the concept of redundancy.
- Several protocols still use the checksum for error correction, although the tendency is to replace it with a CRC.

Idea - 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 (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Idea - Checksum

Example 10.19

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.

One's Complement - Checksum

- In one's Complement arithmetic, We can represent unsigned numbers between 0 and 2^n -1 using only n bits.
  - If the number has more than n bits, the extra left most bits need to be added to the n rightmost bits (wrapping).
  - A negative number can be represented by inverting all bits (changing a 0 to a 1 and a 1 to a 0).
    - This is the same as subtracting the number from 2^n -1.

One's Complement - Checksum

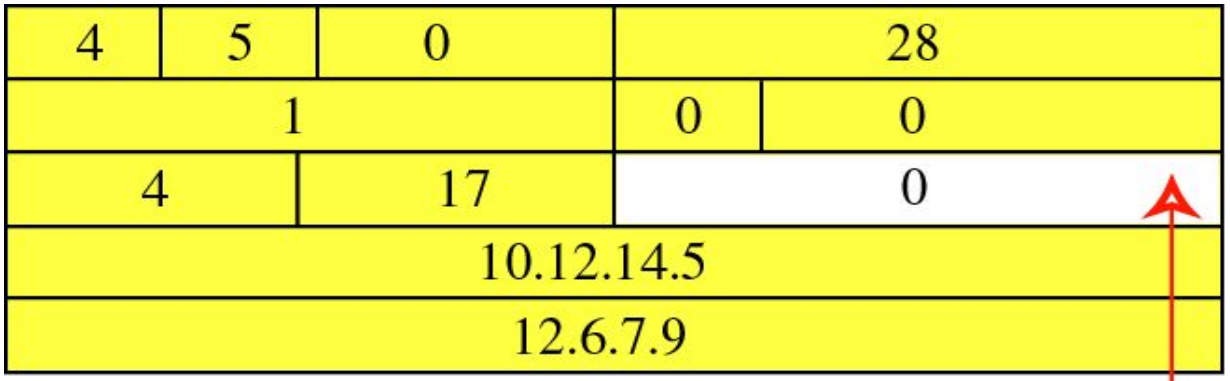
Example 10.20

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.

Detection



4, 5, and 0	→	01000101 00000000
28	→	00000000 00011100
1	→	00000000 00000001
0 and 0	→	00000000 00000000
4 and 17	→	00000100 00010001
0	→	00000000 00000000
10.12	→	00001010 00001100
14.5	→	00001110 00000101
12.6	→	00001100 00000110
7.9	→	00000111 00001001
Sum	→	01110100 01001110
Checksum	→	10001011 10110001

Detection(cont'd)

Example 7 (at a sender)

- 16 bits block □ 8 bits checksum

  Original data : 10101001 00111001

- 1's complement addition

  10101001

  00111001

  11100010 Sum

  00011101 Checksum

- Sending bits

  10101001 00111001 00011101 □ transmitting

Detection(cont'd)

Example 8 (at a receiver)

- No errors:

  Received data : 10101001 00111001 00011101

- 8 bits sum

  10101001

  00111001

  00011101

  11111111 □ Sum

- Complement of Sum

  00000000 □ Complement : It means No error

Example

Now suppose there is a burst error of length 5 that affects 4 bits.

10101111 11111001 00011101

When the receiver adds the three sections, it gets

1 ← carry from 8th column

10 ← carry from 6th column

1 ← carry from 5th column

10 ← carry from 4th column

1 ← carry from 3rd column

1 ← carry from 2nd column

1 ← carry from 1st column

10101111

11111001

00011101

Partial Sum **1 11000101**

Carry **1**

~~**Sum** **11000110**~~

Complement **00111001** **the pattern is corrupted.**

One's Complement

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 $2^n - 1$ (16 − 1 in this case).

One's Complement

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. The 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. In the figure, we have shown the details in binary. 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.

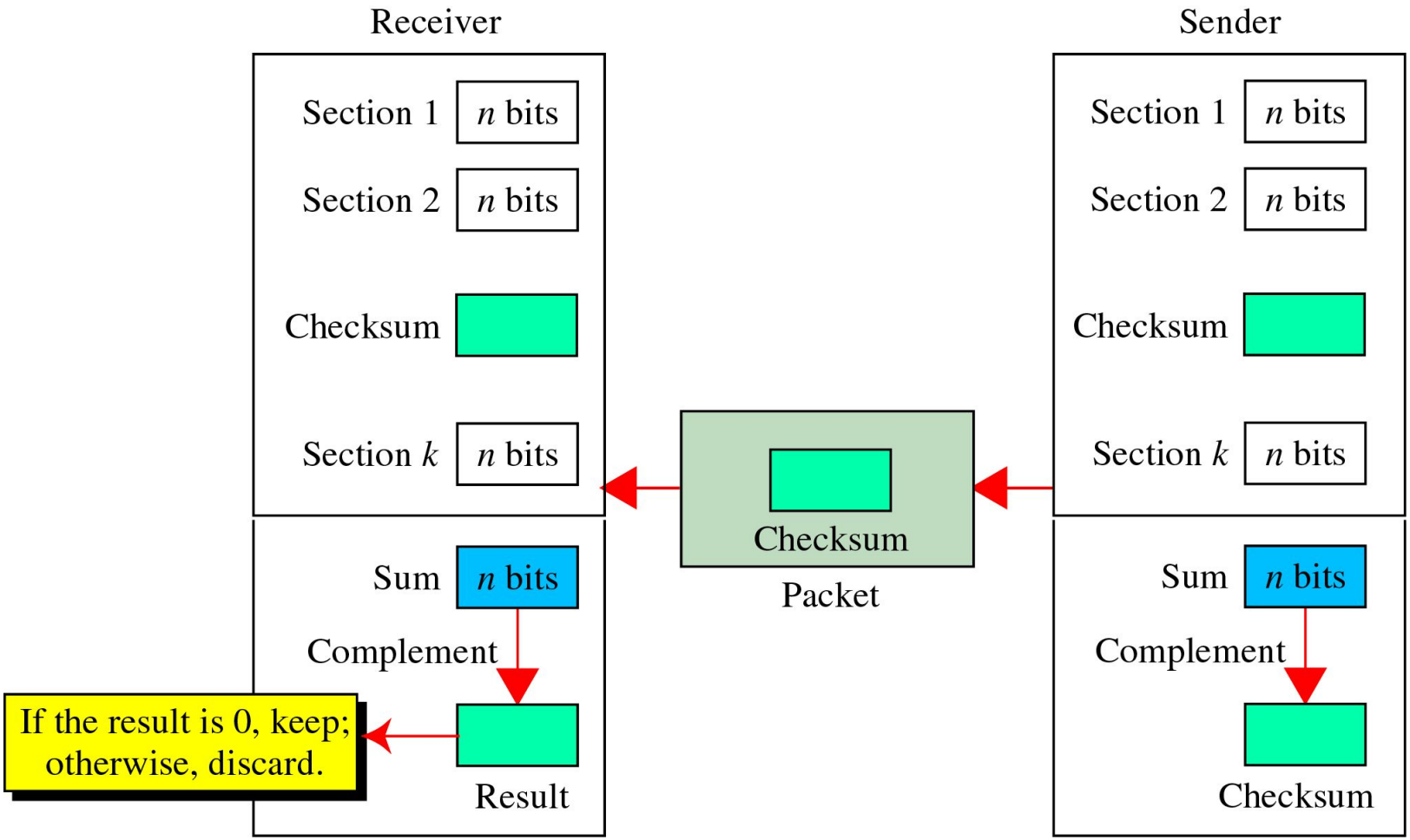
One's Complement

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.

Internet Checksum

Checksum Generator



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.

Internet Checksum

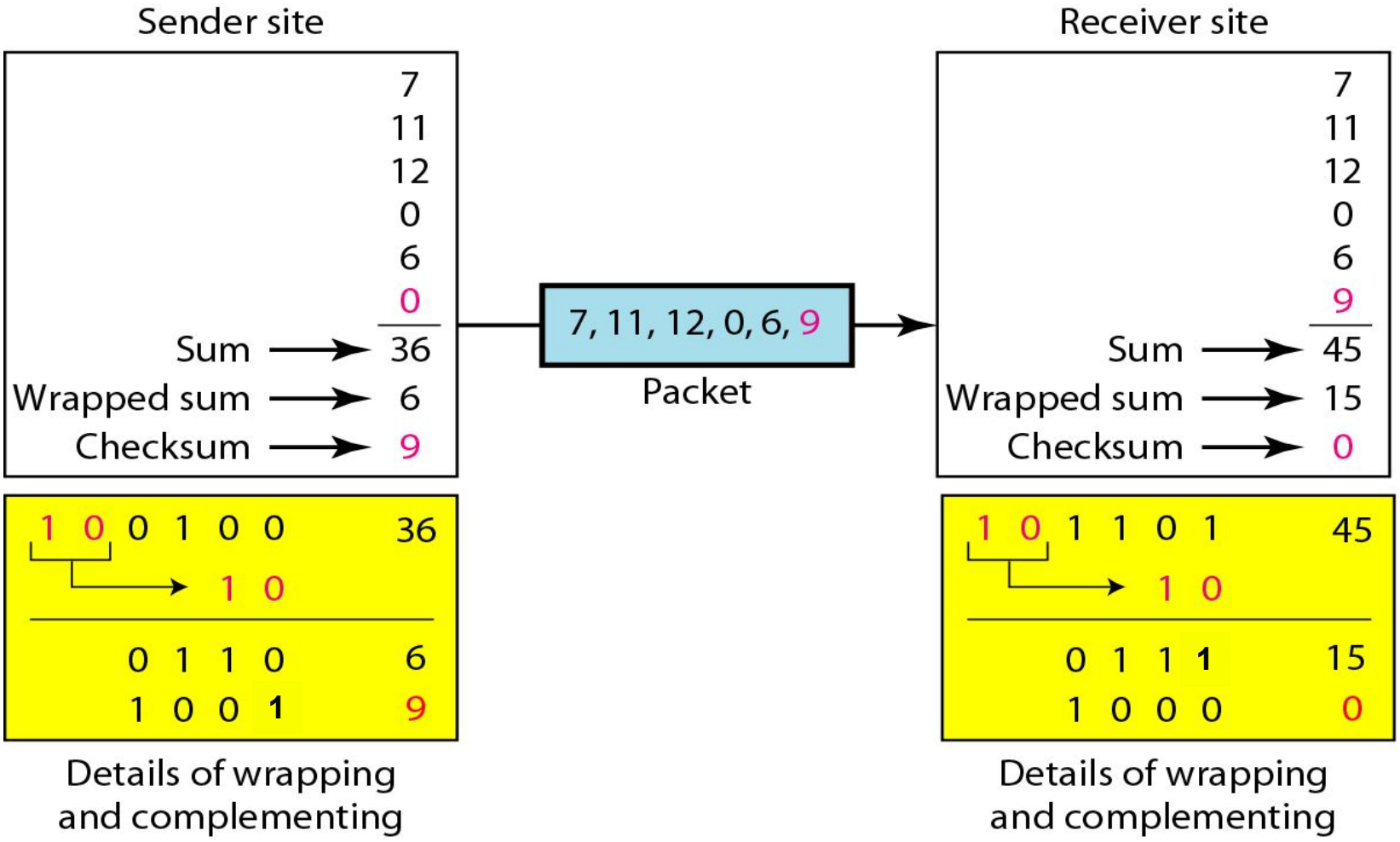
Note

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.**

One's Complement

Figure 10.24 *Example 10.22*



Summary (1)

- Data can be corrupted during transmission. Some applications require that errors be detected and corrected.
- In a single-bit error, only one bit in the data unit has changed. A burst error means that two or more bit in the data unit have changed.
- To detect or correct errors, we need to send extra (redundant) bits with data.
- There are two main methods of error correction: forward error correction and correction by retransmission.
- In coding, we need to use modulo-2 arithmetic. Operations in this arithmetic are very simple; addition and subtraction give the same results, we use the XOR (exclusive OR) operation for both addition and subtraction.

Summary (2)

- The Hamming distance between two words is the number of differences between corresponding bits. The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.
- 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.
- A category of cyclic codes called the cyclic redundancy check (CRC) is used in networks such as LANs and WANs.
- A pattern of 0s and 1s can be represented as a polynomial with coefficient of 0 and 1.
- Traditionally, the Internet has been using a 16-bit checksum, which uses *one's complement* arithmetic. In this arithmetic, we can represent unsigned numbers beteen 0 and $2^n - 1$ using only n bits.

Q & A