Lecture 3

Error Detection and Correction

(Chapter 10 of “Data Communications and Networking” [B. A. Forouzan], 5th Edition)

Note

Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

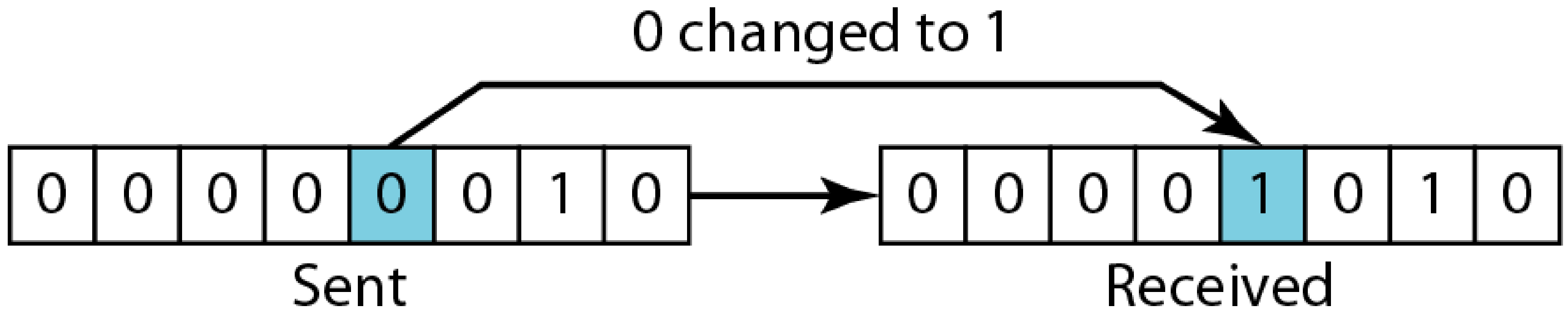
10-1 INTRODUCTION

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

Note

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

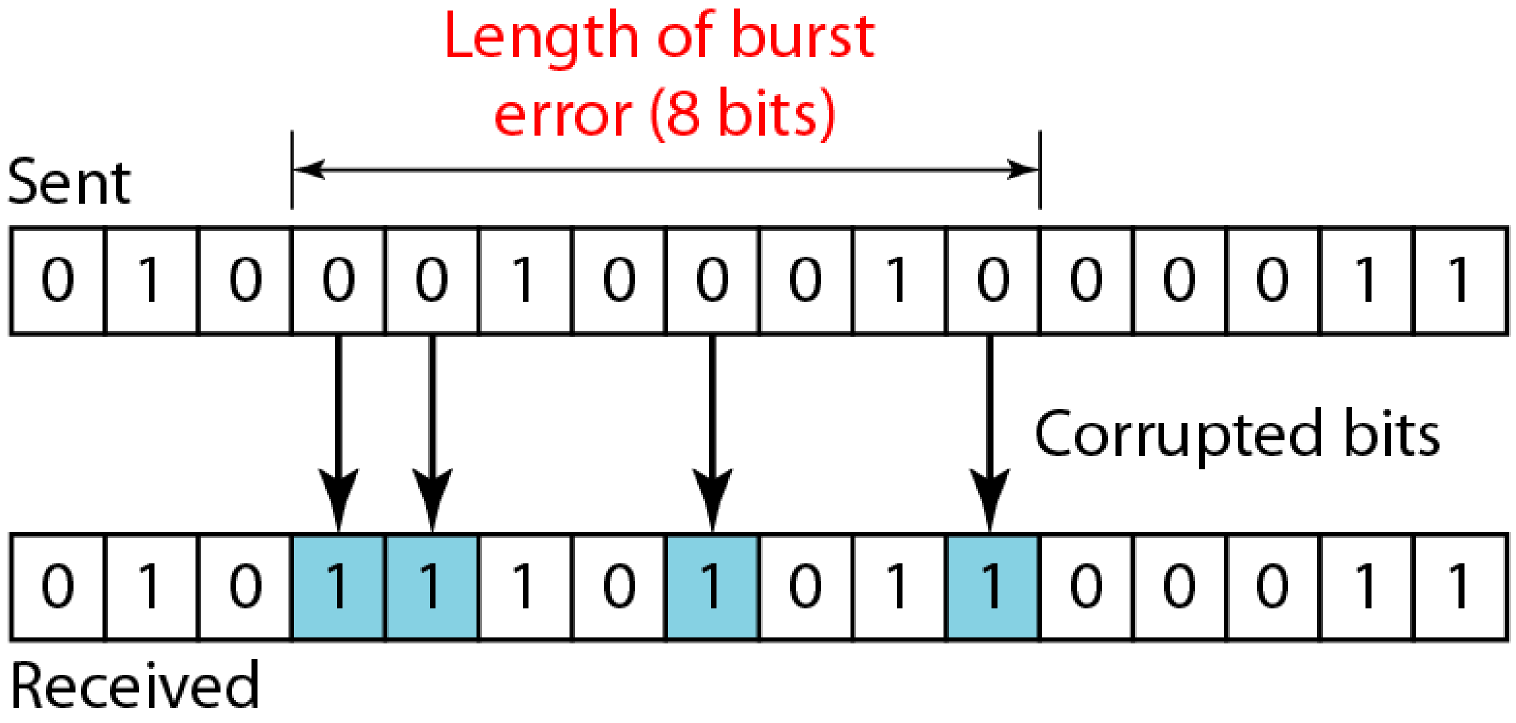
Figure 10.1 *Single-bit error*



Note

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

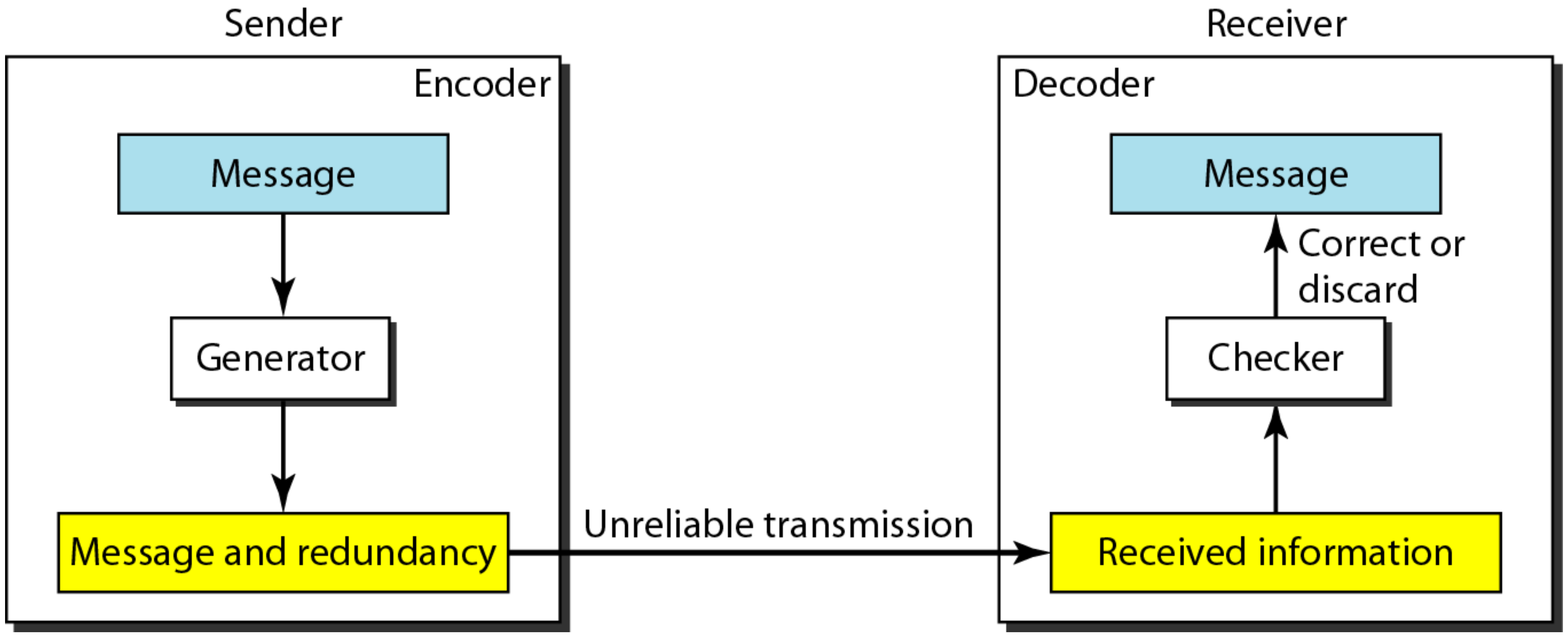
Figure 10.2 *Burst error of length 8*



Note

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

Figure 10.3 *The structure of encoder and decoder*



Note

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

Note

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

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

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

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

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

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

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

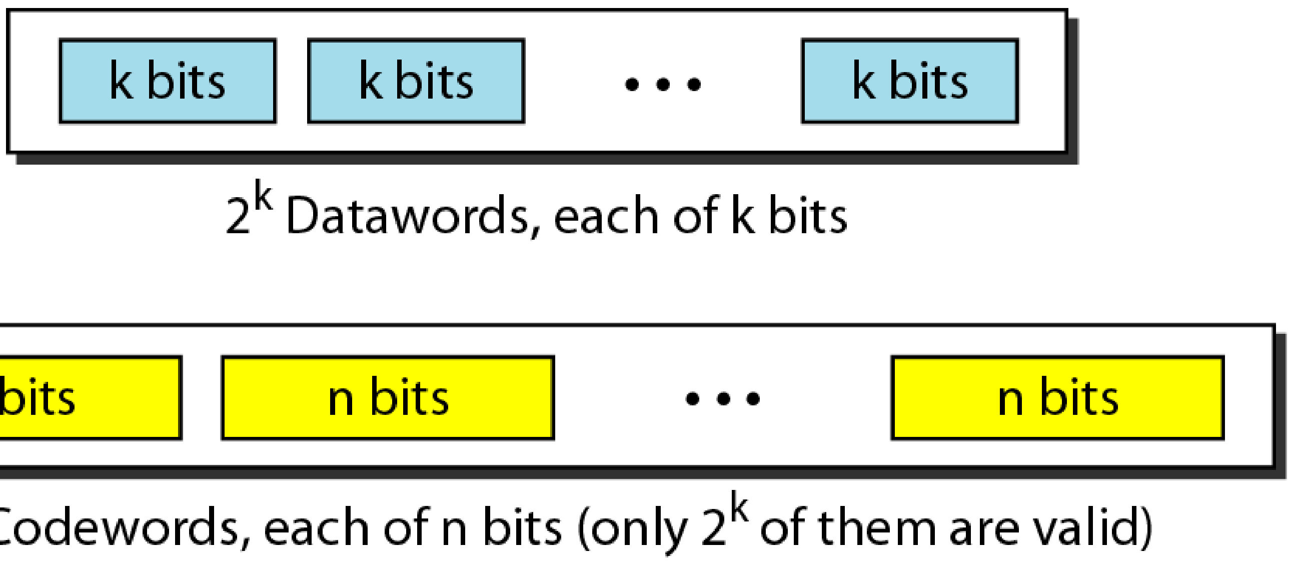
c. Result of XORing two patterns

XOR is actually binary addition without carry

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.

Figure 10.5 ***Datawords and codewords in block coding***


k bits
k bits
• • •
k bits
2^k Datawords, each of k bits
n bits
n bits
• • •
n bits
2^n Codewords, each of n bits (only 2^k of them are valid)

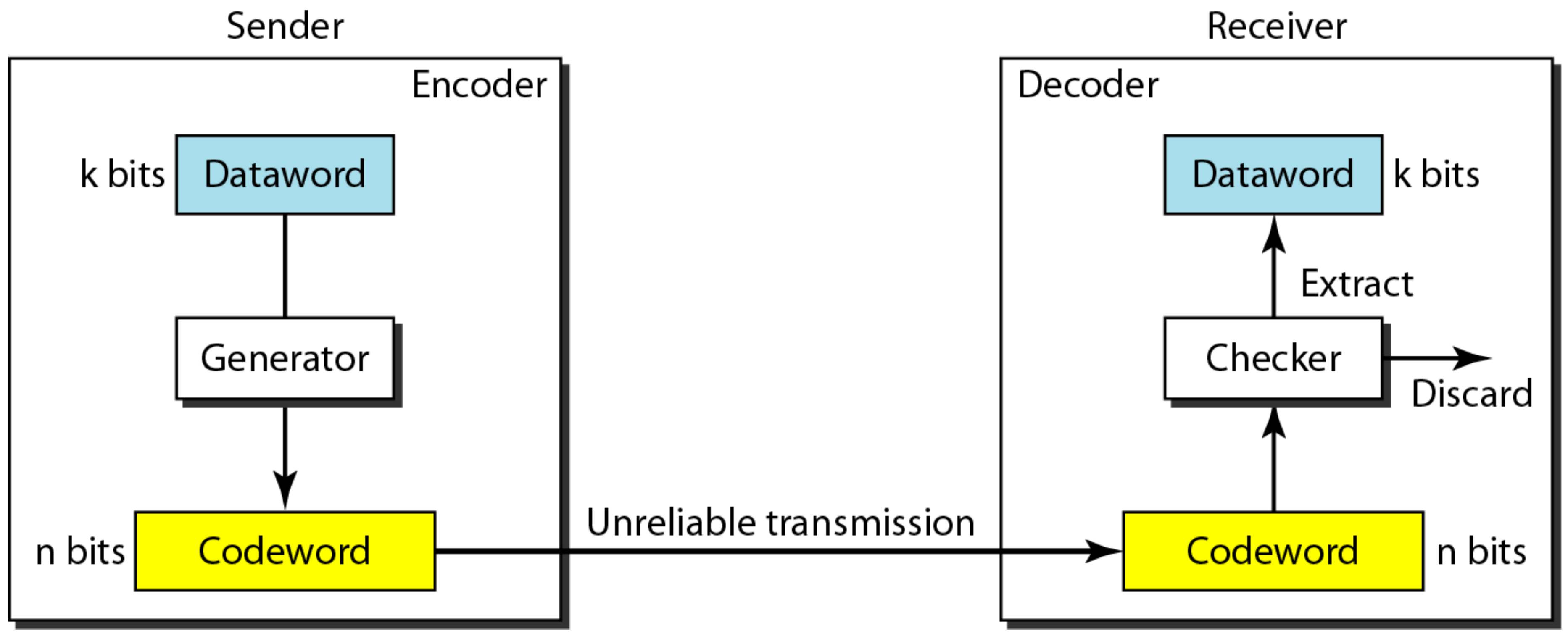
At the transmitter side, it only sends one valid codeword. Errors may happen in the transmission.

If we receive an invalid codeword, we know error(s) happened. (errors detected).

If we receive a valid codeword, we assume there is no error in transmission. In this case, it is possible that errors in the transmission make the original valid codeword become another valid codeword (which is received). This means the receiver is fooled. (errors undetected).

There is no coding scheme that can detect all possible errors. For example, if up to n bit errors can happen, for any two valid codewords, it is always possible that the sender sends one valid codeword but the receiver receivers the other valid codeword. Therefore, each coding method has a target error pattern, For example, up to a single bit error, up to two bit errors, up to three bit errors…

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. Later, we will see how to derive a codeword from a dataword.

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

Datawords	Codewords
00	000
01	011
10	101
11	110

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

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

Example 10.2 (continued)

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

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

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

A single error always makes a valid codeword invalid. So we can claim the coding scheme in Table 10.1 Can detect single bit error.

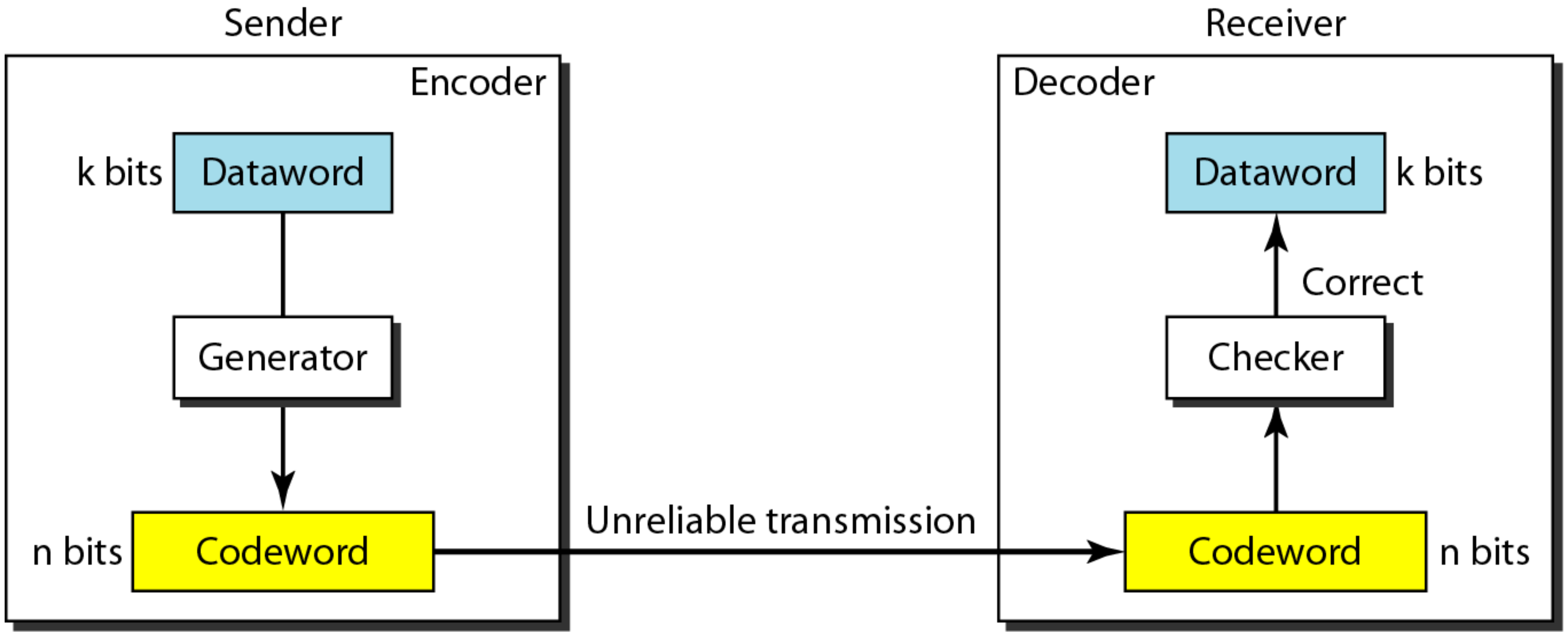
If a coding scheme can detect all possible cases in an error pattern (single-bit error, up to two bit errors, up to three bit errors, …), we claim the coding scheme can detect the error pattern. This means any possible cases of the error pattern always changes a valid codeword to an invalid codeword.

Note

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

The coding scheme in Table 10.1 is designed for single bit error. It cannot detect up to two bit errors.

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. Assume the dataword is 01. The sender creates the codeword 01011.

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

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

Example 10.3 (continued)

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

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

1. *The transmitted codeword 01011 is corrupted during transmission, and 01001 is received. First, the receiver finds that the received codeword is not in the table (error detected). This means error(s) occurred. For error correction, the receiver would select the valid codeword that is closest to the received codeword. Here "closest" means the least number of different bits.*

   Compare the received codeword (01001) with the four valid codewords in the Table 10.2 and find out the number of different bits:

   01001 vs. 00000: 2 different bits
   01001 vs. 01011: 1 different bit
   01001 vs. 10101: 3 different bits
   01001 vs. 11110: 4 different bits.

   So the receiver believes originally codeword 01011 was sent. It replaces the received codeword 01001 with 01011, and extract the dataword 01. The error is successfully corrected.

For the coding scheme in Table 10.2, any single bit error can be detected and corrected.

Example 10.3 (continued)

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

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

2. The transmitted codeword 01011 is corrupted during transmission, and 01000 is received. First, the receiver finds that the received codeword is not in the table (error detected). This means error(s) occurred. For error correction, the receiver would select the valid codeword that is closest to the received codeword.

Compare the received codeword (01000) with the four valid codewords in the Table 10.2 and find out the number of different bits:

01000 vs. 00000: 1 different bits
01000 vs. 01011: 2 different bit
01000 vs. 10101: 4 different bits
01000 vs. 11110: 3 different bits.

So the receiver believes originally codeword 00000 was sent. It replaces the received codeword 01000 with 00000, and extract the dataword 00. The errors are NOT successfully corrected.

For the coding scheme in Table 10.2, any two bit errors can be detected; but cannot be corrected.

Note

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

Assuming the Hamming distance of two valid codewords is d. If one codeword is sent, then d bits of error may make you receive the other codeword.

Example 10.4

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

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

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

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

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

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.

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

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.

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

Note

For a coding scheme with minimum Hamming distance d_{min}, up to (d_{min} -1) errors can be detected.

1. Undetected errors means the errors change a valid codeword to another valid codeword.
2. To change a valid codeword to another valid codeword, we need at least d_{min} errors.
3. So d_{min} -1 errors would not change a valid codeword to another valid codeword, and thus, can be detected.

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 will always match a valid codeword and the 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

Example 10.8

Our second block coding scheme (Table 10.2) has d_{min} = 3. This coding scheme 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.

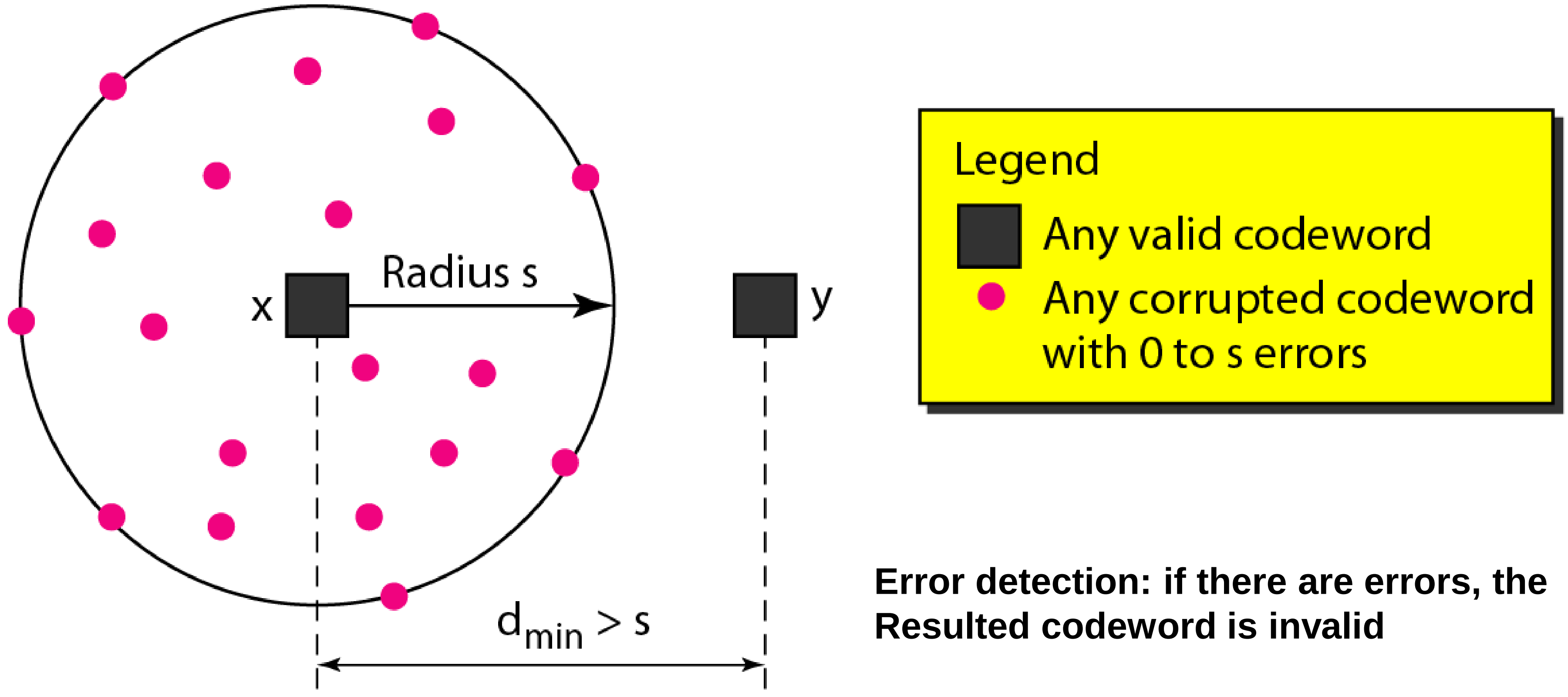
Some three bit errors can be detected. For example, if the sender sends codeword 00000 and the receiver receives 11100, which is invalid.
However, some combinations of three errors change a valid codeword to another valid codeword. For example, if the sender sends codeword 00000 and the receiver receives 10101(valid), three bit errors happen, and cannot be detected.

Therefore, since the coding scheme in Table 10.2 cannot detect all possible cases of three bit errors, we do NOT claim the coding scheme can detect three bit errors.

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

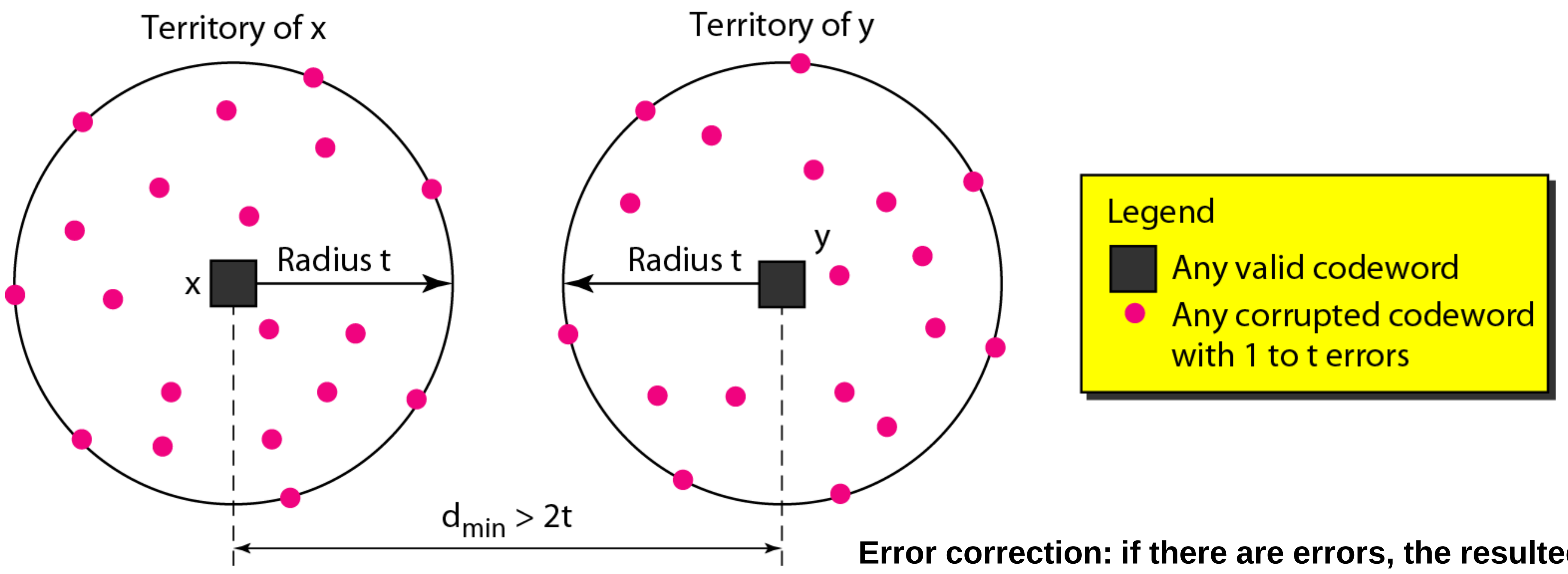
Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

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



Error detection: if there are errors, the Resulted codeword is invalid

Figure 10.9 *Geometric concept for finding d_{min} in error correction*



Error correction: if there are errors, the resulted (invalid) codeword is still closer to the original codeword (compared to distance to other valid codewords)

Note

For a coding scheme with minimum Hamming distance d_{min}, up to *t* errors can be corrected, with *t* being the maximal integer such that $2t < d_{min}$.

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 prefer 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.

Note

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

Datawords	Codewords
00	000
01	011
10	101
11	110

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

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Example 10.11

Recall that to calculate the Hamming distance between codewords A and B, we only need to count the number of bit 1s in the result of A XOR B

In a linear block code, if valid codeword A XOR valid codeword B gets valid codeword C, then the number of bit 1s in codeword C is actually the Hamming distance of codewords A and B.

Thus, for a linear block code, to get all Hamming distance values, we only need to count the number of bit 1s in each nonzero valid codeword. The minimum Hamming distance is the smallest number of 1s in a nonzero valid codeword.

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.

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

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

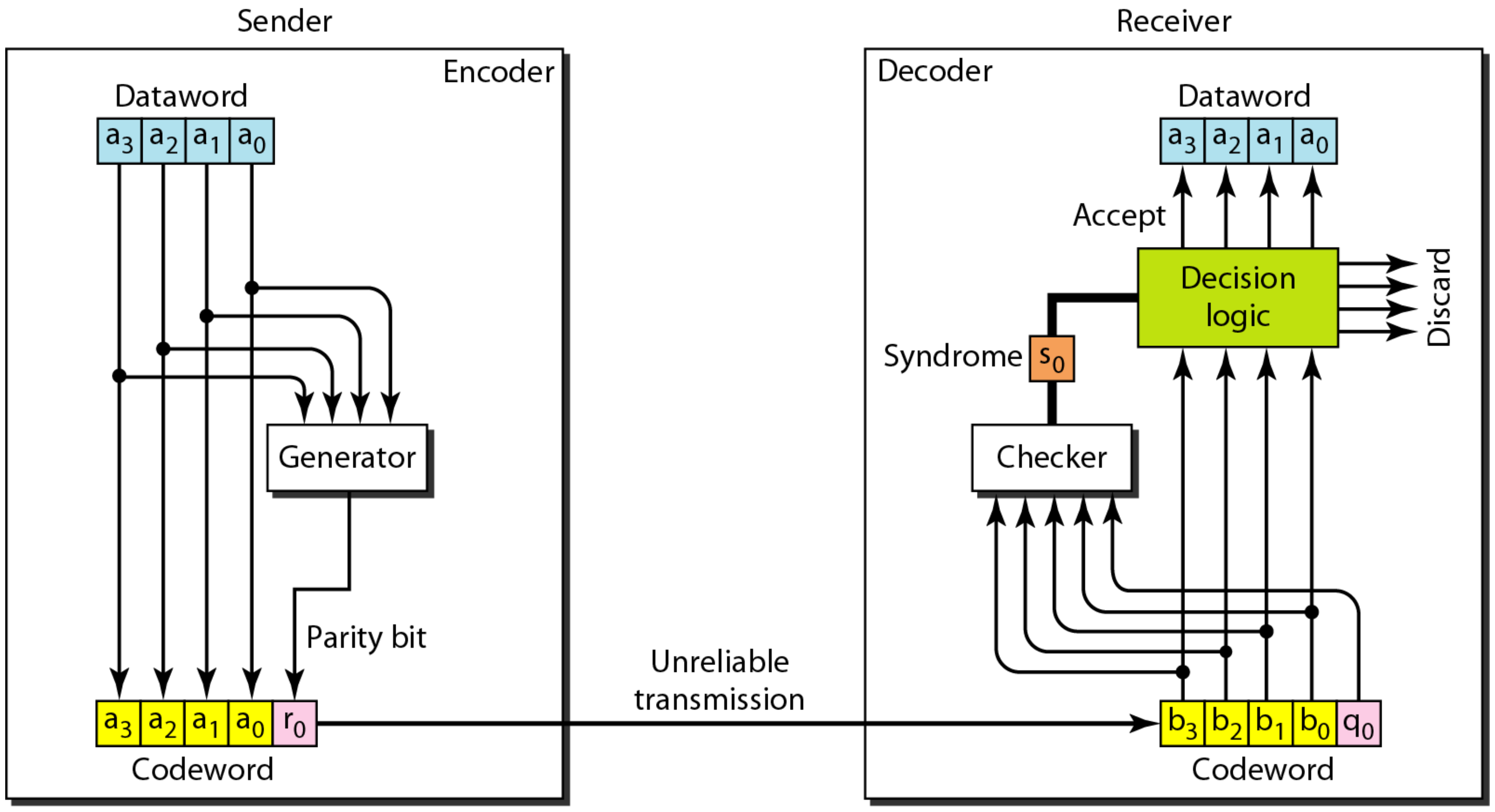
Note

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

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***



$r_0=a_3+a_2+a_1+a_0$ modulo-2

$s_0=b_3+b_2+b_1+b_0+q_0$ modulo-2

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. Note that here the dataword is wrongly created due to the syndrome value.*

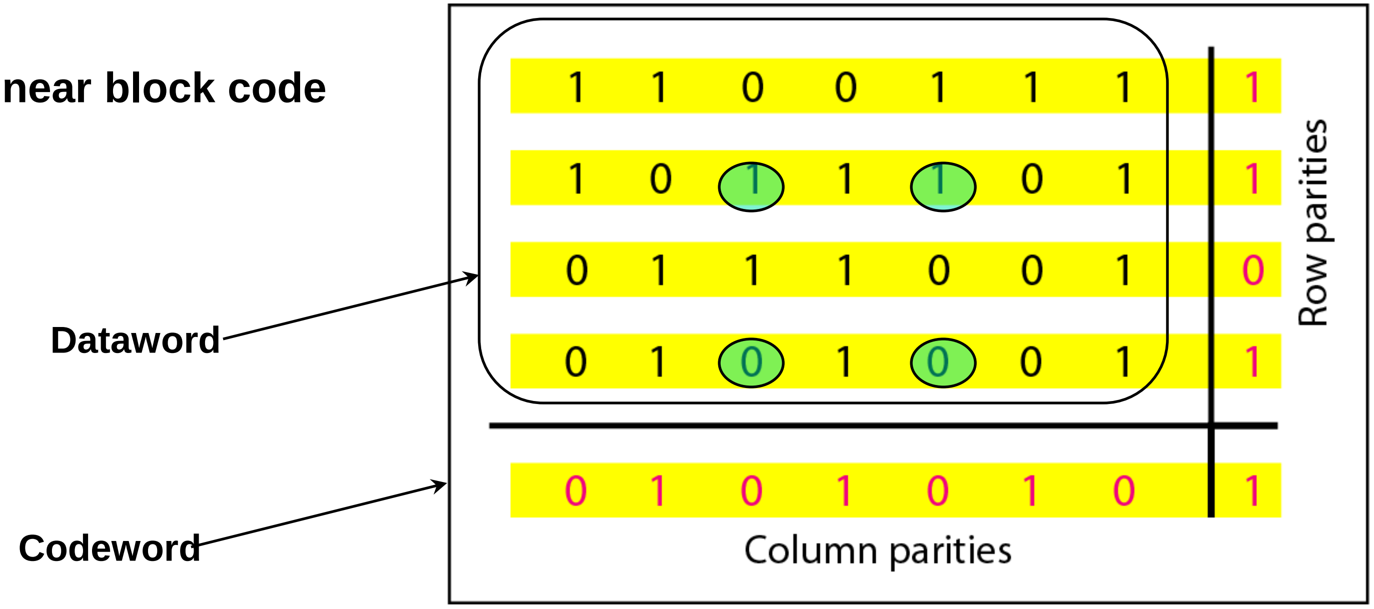
***5**. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.*

Note

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

Figure 10.11 *Two-dimensional parity-check code*

It's linear block code



a. Design of row and column parities

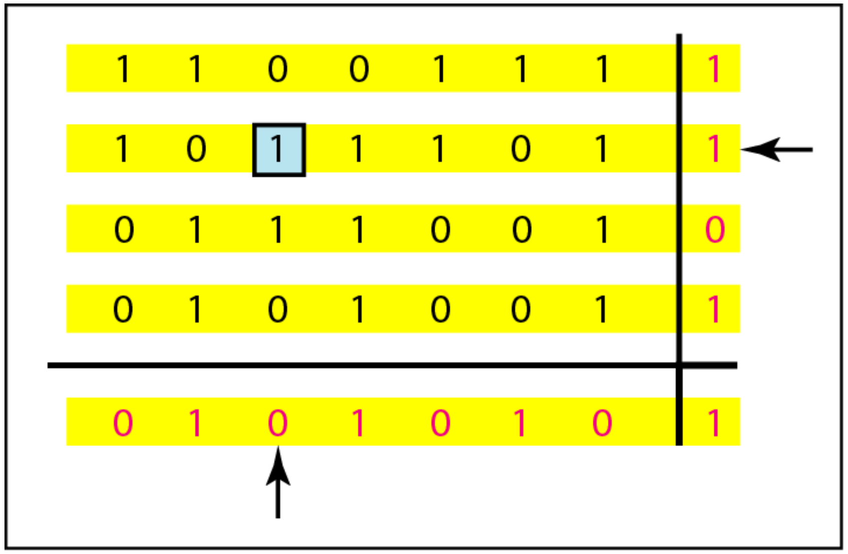
Totally 12 parities, 4 for the 4 rows, 7 for the 7 columns, and one for both the "parity row" and the "parity column"

At the receiver, we perform parity checks for the 5 rows and 8 columns. If all the 13 syndromes are zeros, then receiver thinks there is no error.

The minimum Hamming distance is 4 (as long as the four errors form a rectangle, they will change a valid codeword to another valid codeword).

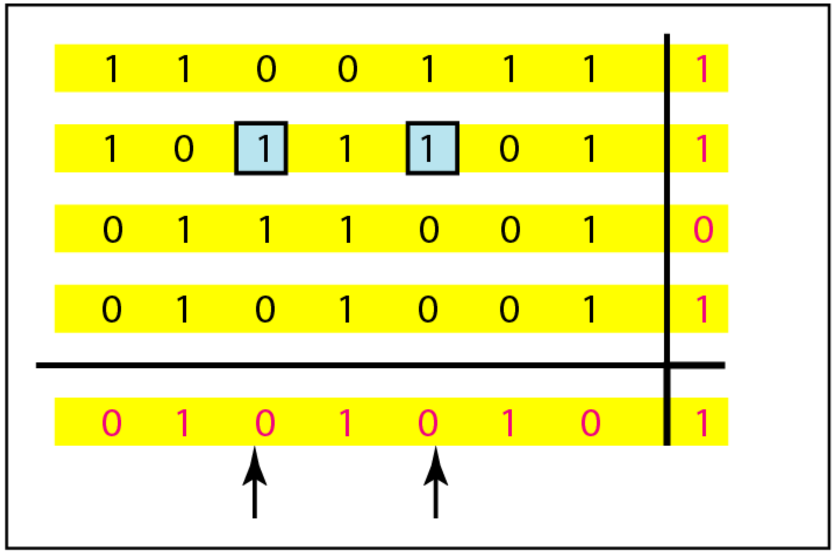
So it can detect up to three errors, and correct one error.

Figure 10.11 *Two-dimensional parity-check code*



b. One error affects two parities

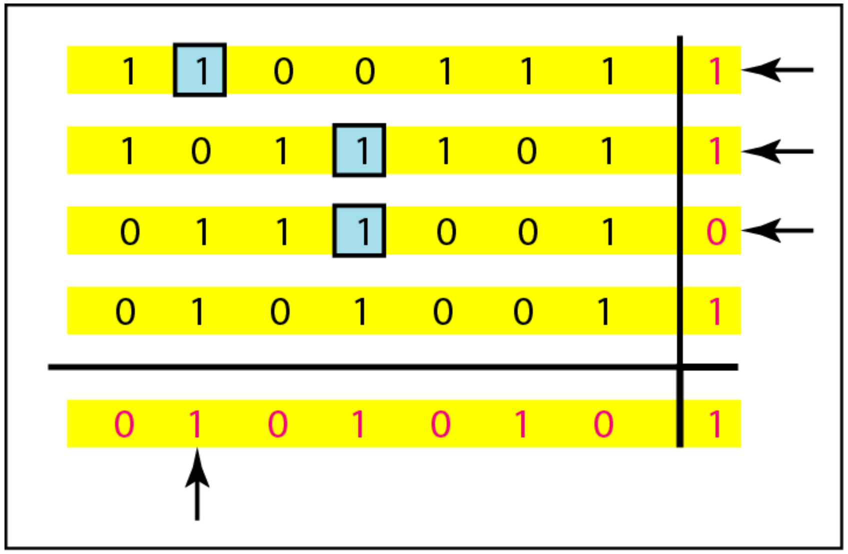
One error can be located and corrected



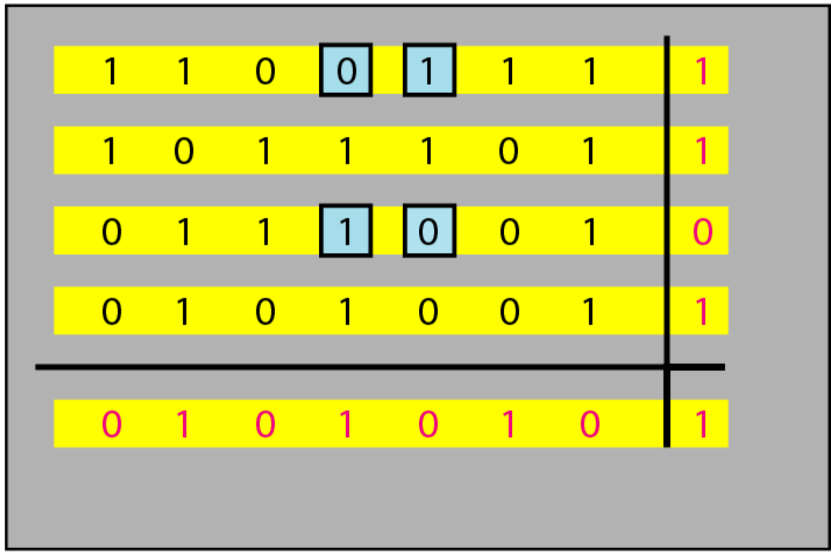
c. Two errors affect two parities

Two errors affect at least two parities

Figure 10.11 *Two-dimensional parity-check code*



d. Three errors affect four parities



e. Four errors cannot be detected

Three errors affect at least two parities.

Other patters of four errors will be detected

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

It’s linear block code

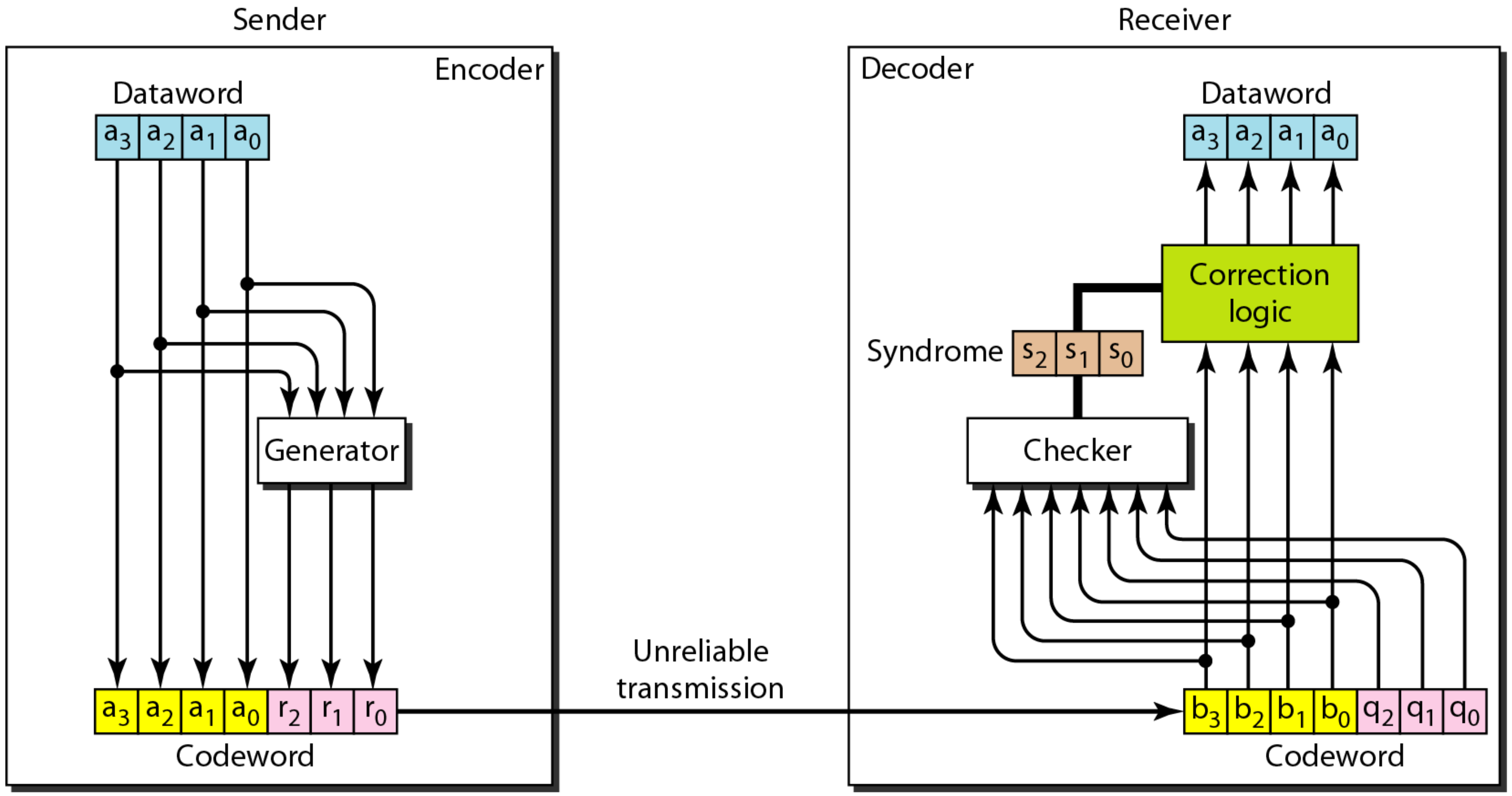
Note

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

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

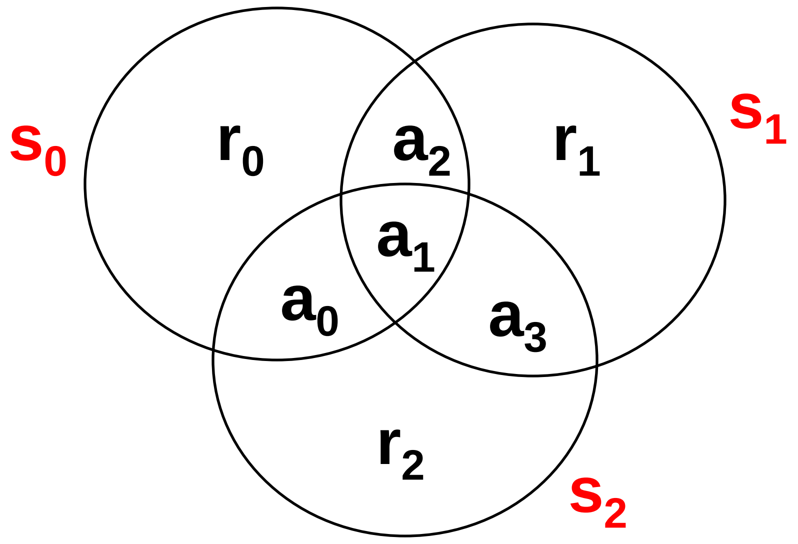
m: number of check bits
n: number of bits in a codeword

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

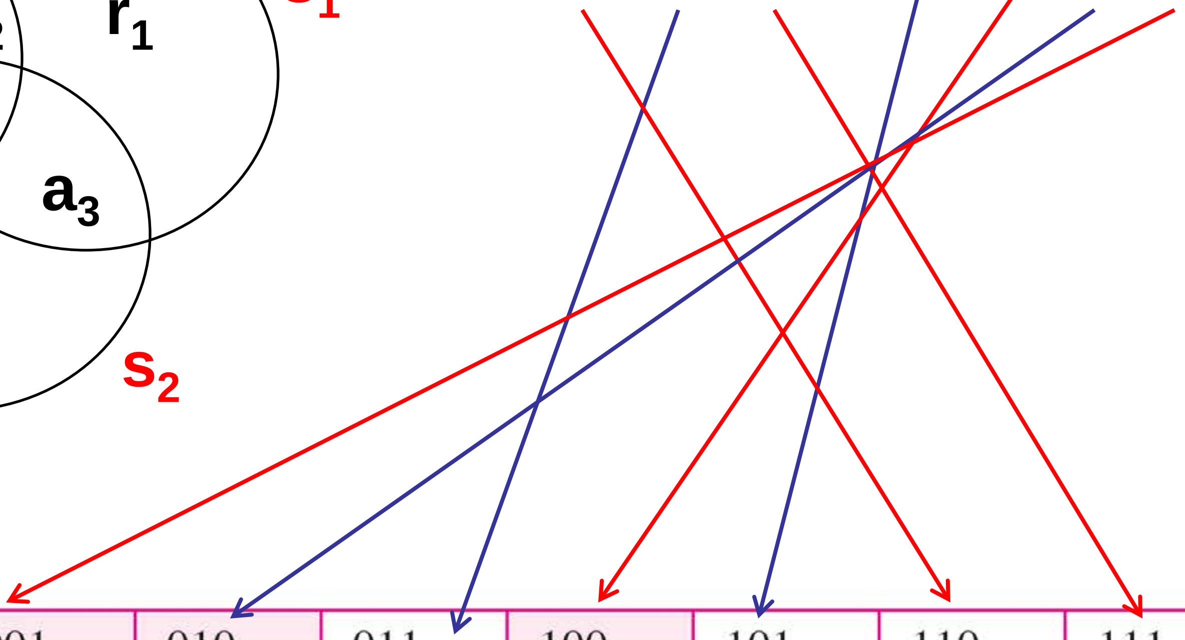


$r_2 = a_3 + a_1 + a_0$ modulo-2
$r_1 = a_3 + a_2 + a_1$ modulo-2
$r_0 = a_2 + a_1 + a_0$ modulo-2

$s_2 = b_3 + b_1 + b_0 + q_2$
$s_1 = b_3 + b_2 + b_1 + q_1$
$s_0 = b_2 + b_1 + b_0 + q_0$



s_2	1	0	1	1	1	0	0
s_1	1	1	1	0	0	1	0
s_0	0	1	1	1	0	0	1



Syndrome $s_2s_1s_0$	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

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

Example 10.13

For Hamming code's error detection/correction, we can use Table 10.4 on Slide 49: If a received codeword in not in the list of Table 10.4, we say it is invalid (error detection), and pick up the valid Codeword in Table 10.4 that has the minimum distance to the received codeword (error correction).

This method works, but takes time. Next we perform error detection and correction based on syndrome bits $S_2S_1S_0$, which is faster.

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 codeword 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.

	Simple parity-check Code (k=4, r=1, n=5)	Two-dimensional parity-check code (k=28, r=12, n=40)	Hamming code (k=4, r=3, n=7)
Min Hamming distance	2	4	3
Efficiency (k/n)	4/5	28/40	4/7
Detection	odd number of errors	odd number of errors; two errors (per 40 bits)	up to two errors (per 7 bits)
Correction	none	1 error (per 40 bits)	1 error (per 7 bits)

Further Discussions for Error Detection Capability of Linear Block Codes

We use example of k=4, n=7. We have $2^4=16$ valid codewords.

The sender transmits a valid codeword.

Define 7-bit error pattern as follows: for each bit in the error pattern, if the bit is '0', it means that the corresponding bit in the codeword is correctly received; if the bit in the error pattern is '1', it means that the corresponding bit in the codeword is corrupted.

For example, when the sender transmits valid codeword 0111001, if the error pattern Is 1000000, this means the receiver receives codeword1111001.

We can see: transmitted valid codeword XOR error pattern = received codeword.

Recall that in linear block codes, valid codeword XOR valid codeword = valid codeword.

Thus, if the error patter is in the form of a valid codeword, the receiver would receive a valid codeword. In other words, the error pattern cannot be detected.

Thus, when error(s) happen, the error pattern has $2^7-1=127$ combinations. $2^4-1=15$ combinations cannot be detected.

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.**

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

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

Figure 10.14 *CRC encoder and decoder*

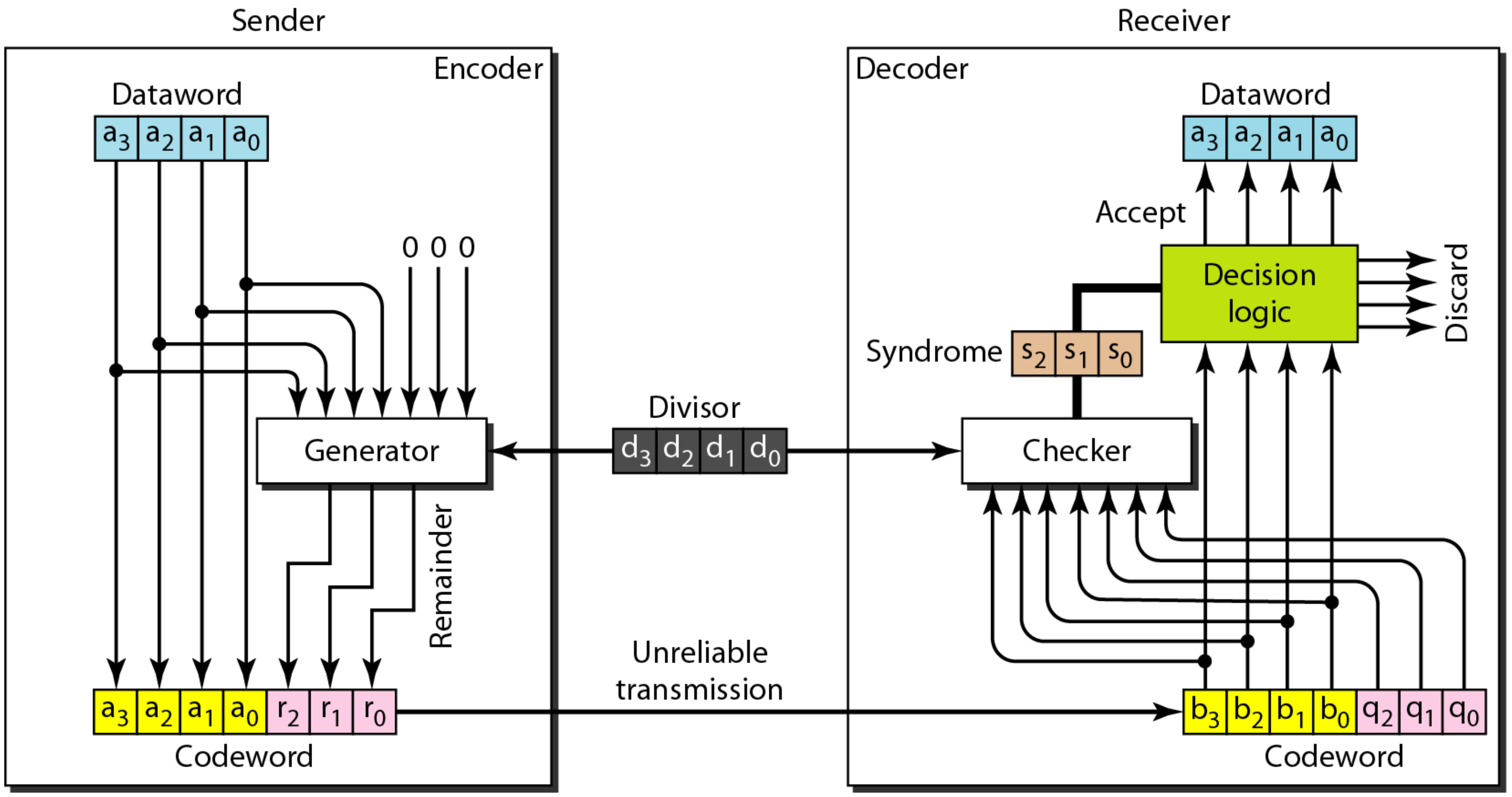
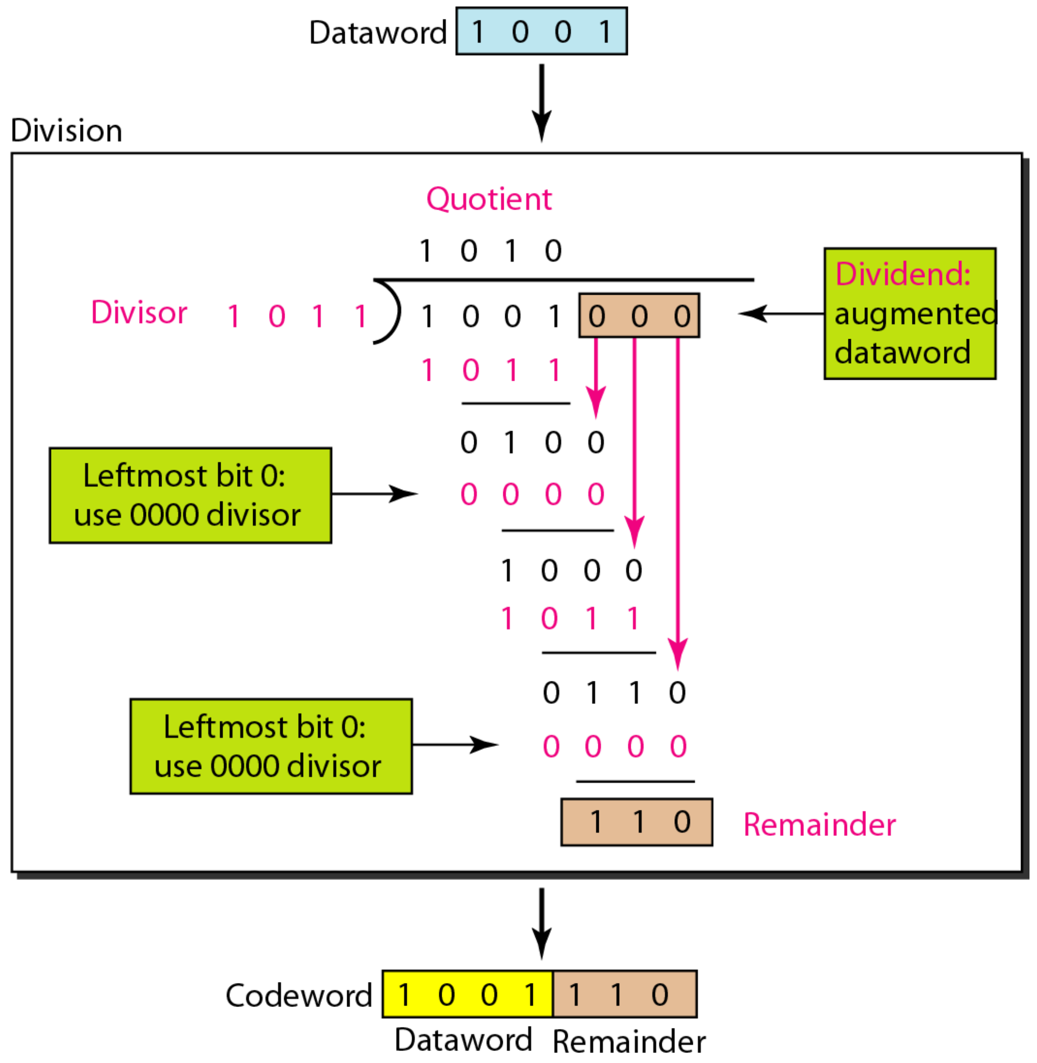
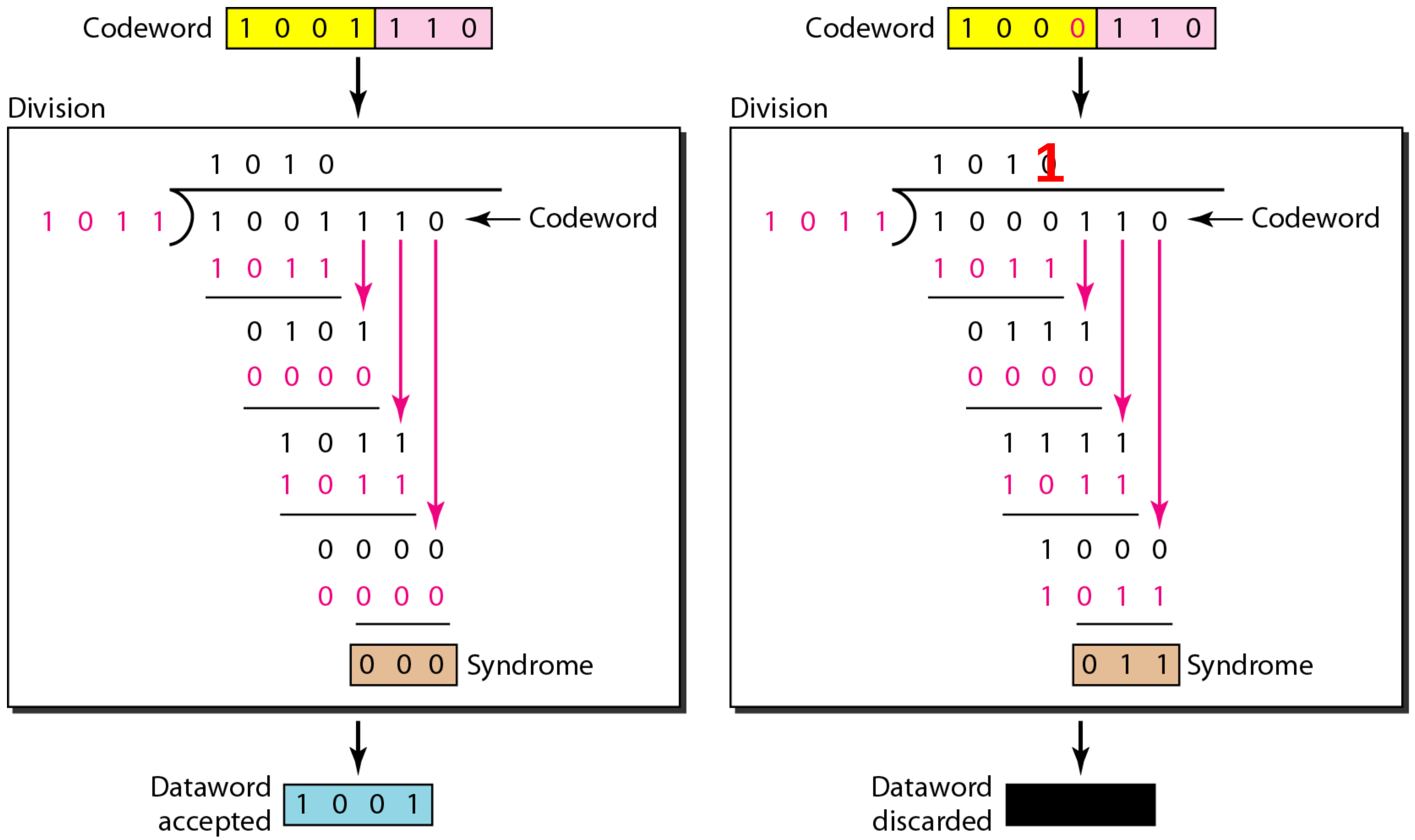


Figure 10.15 *Division in CRC encoder*



We need 3 redundant bits. So we add 3 zeros

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



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

Used in network and transport layers

Example 10.18

Suppose our data is a list of five 8-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.

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.