Error Detection and Correction

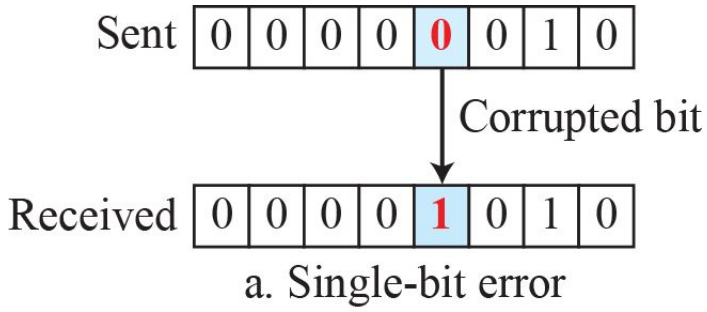
Session Objectives

After going through this session you will learn

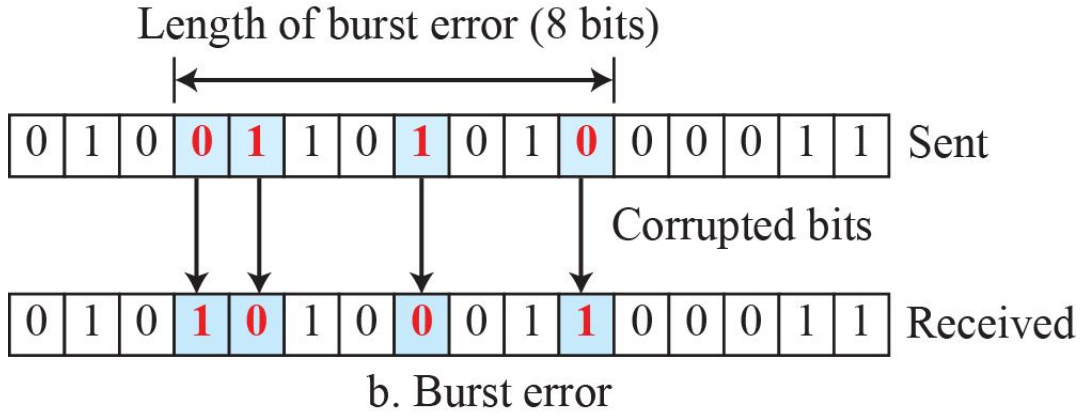
- Types of error
- Block Coding Technique
- Cyclic Codes (CRC)
- Checksum

Types of Errors

- Whenever bits flow from one point to another, they are subject to unpredictable changes because of **interference**. This interference can change the shape of the signal.
- The term **single-bit error** means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.
- The term **burst error** means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure below shows the effect of a single-bit and a burst error on a data unit.



a. Single-bit error



b. Burst error

Redundancy Bits

- The central concept in detecting or correcting errors is redundancy.
- To be able to detect or correct errors, we need to **send some extra bits** with our data.
-
- These redundant bits are added by the sender and removed by the receiver.
- Their presence allows the receiver to detect or correct corrupted bits.

Detection versus Correction

- The correction of errors is more difficult than the detection.
- In error detection, we are **only looking** to see if any error has occurred.
- The answer is a simple yes or no. We are not even interested in the number of corrupted bits.
- A single-bit error is the same for us as a burst error. In error correction, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message.

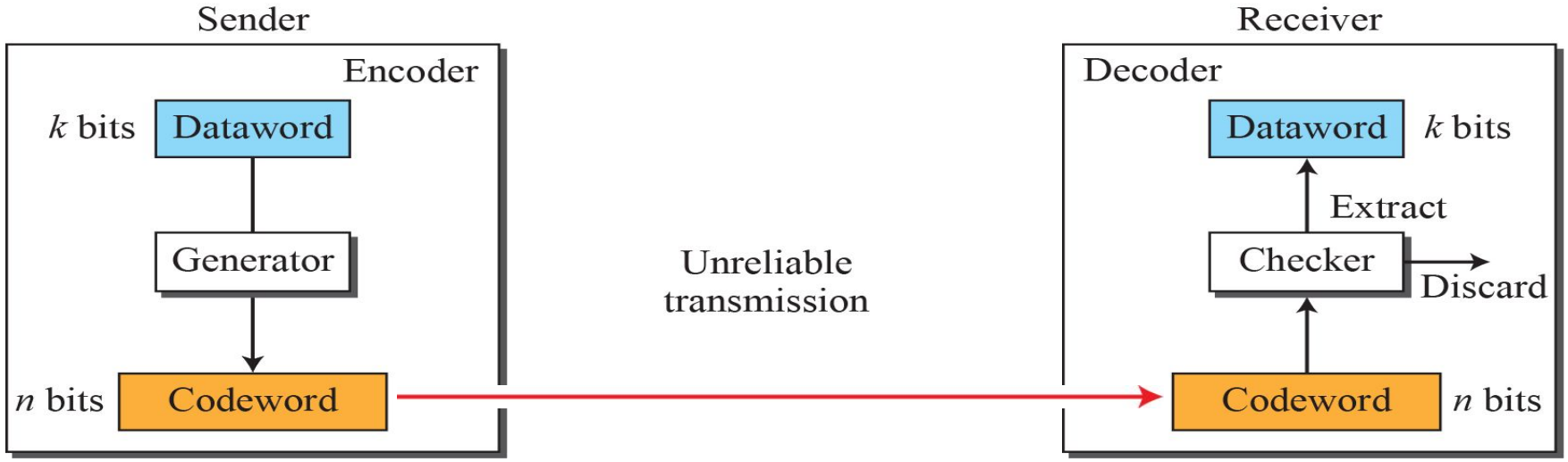
Block Coding Technique

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

Error detection using Block Coding

Applying some conditions, the receiver can detect a change in the codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.



Example 1

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

Table 1: A code for error detection

Datawords	*Codewords*	*Datawords*	*Codewords*
00	000	10	101
01	011	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.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Hamming Distance between two codes

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).

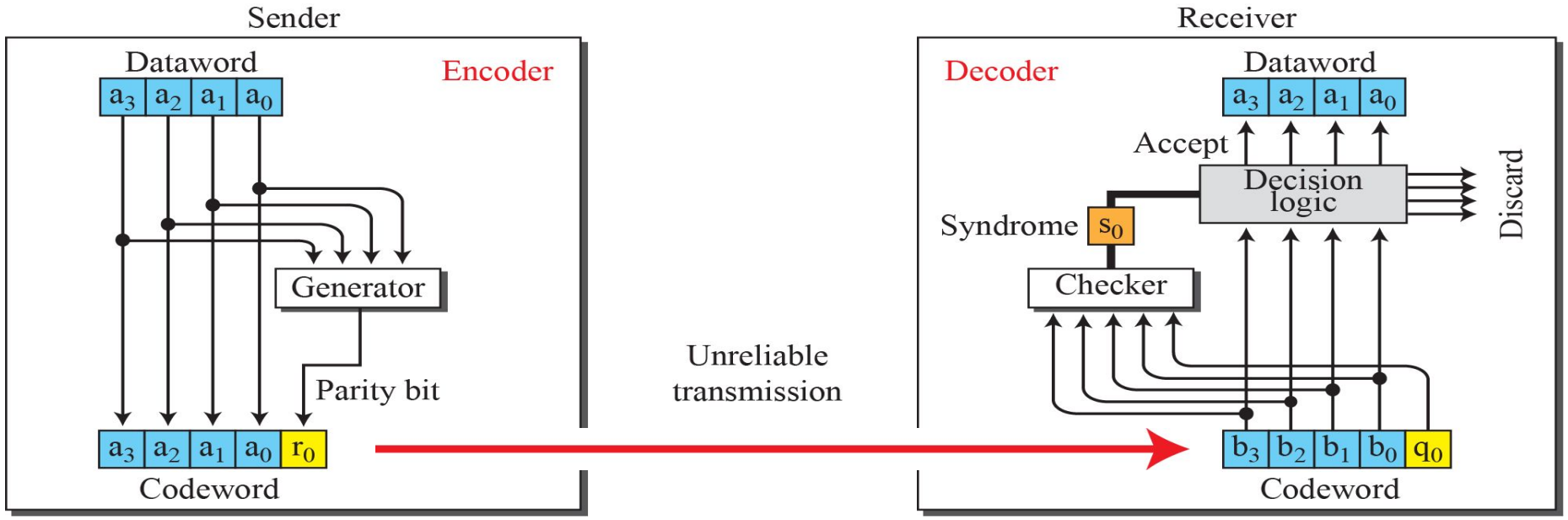
The minimum Hamming distance for our first code scheme is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Conclusion

- A code scheme has a Hamming distance $\mathbf{d}_{min} = 4$. This code guarantees the detection of up to three errors $\mathbf{d} = s + 1$ or $\mathbf{s} = 3$).
- The code in **Table 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.

Table 2: Parity Checking Method 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**



Example 2

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:

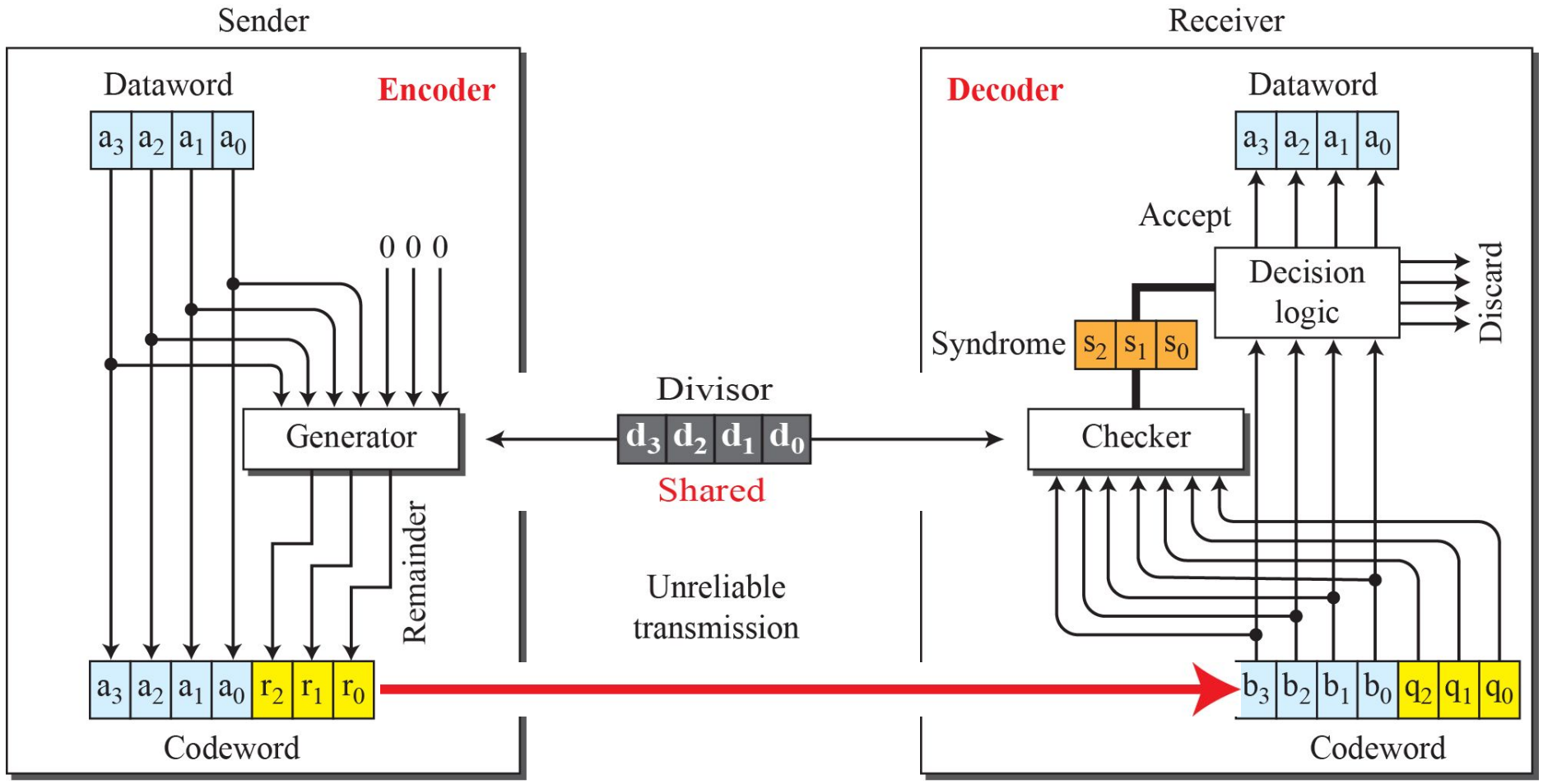
1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a_1. The received codeword is 10**0**11. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0. The received codeword is 1011**0**. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes r_0 and a second error changes a_3. The received codeword is **0**011**0**. 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. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is **010**11. 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.

Cyclic Codes (CRC) Method

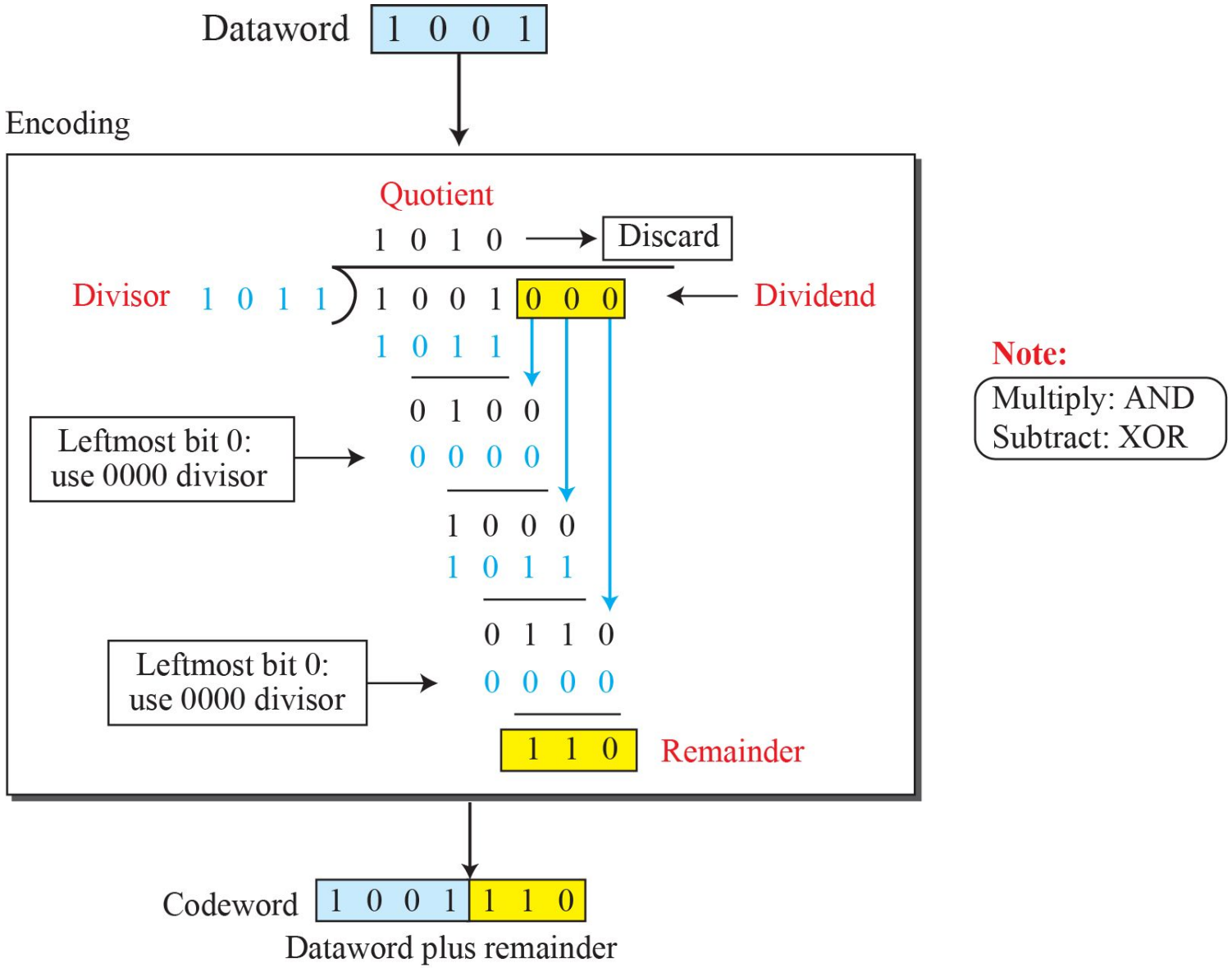
- Cyclic codes are special linear block codes with one extra property.
- In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.
- For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.
- ***Example:*** Cyclic code for C(7, 4) is shown below

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

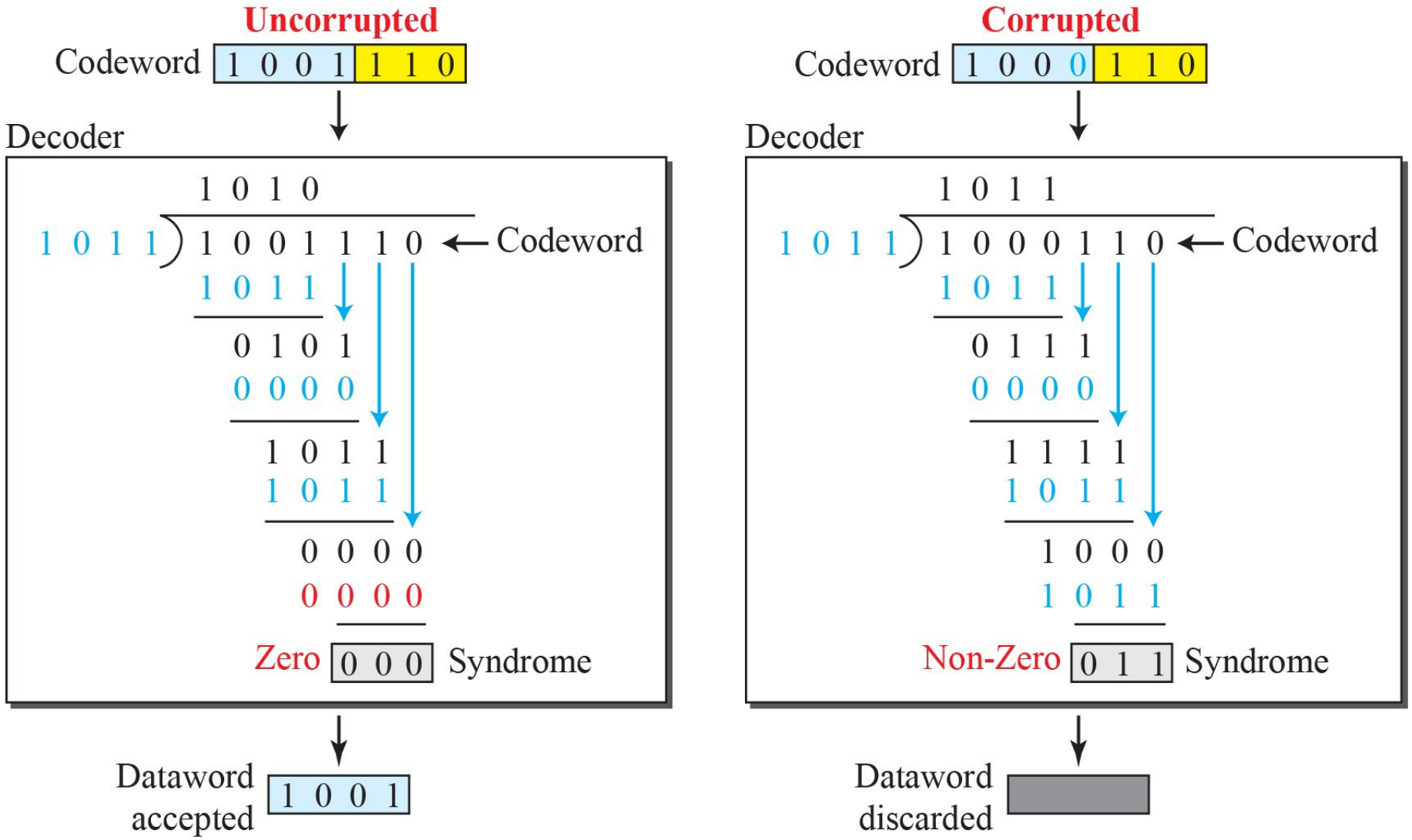
CRC Encoder and Decoder



Division in CRC Encoder



Division in CRC Decoder for Two cases

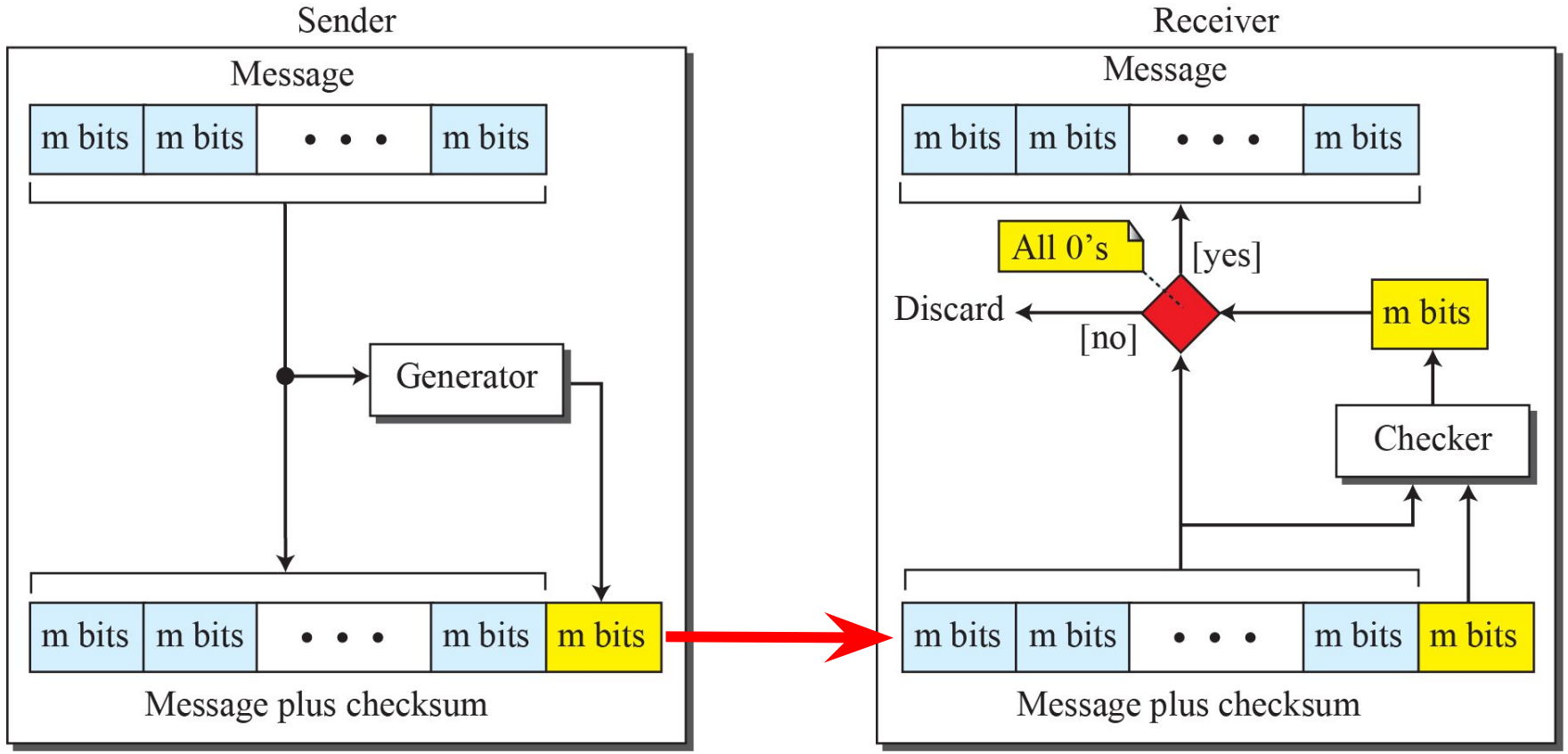


CRC Advantages

- 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.

Checksum Method

- Checksum is an error-detecting technique that can be applied to a message of any length.
- In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.



Checksum Illustration with an Example

Suppose the message 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 message not accepted.

Explanation:

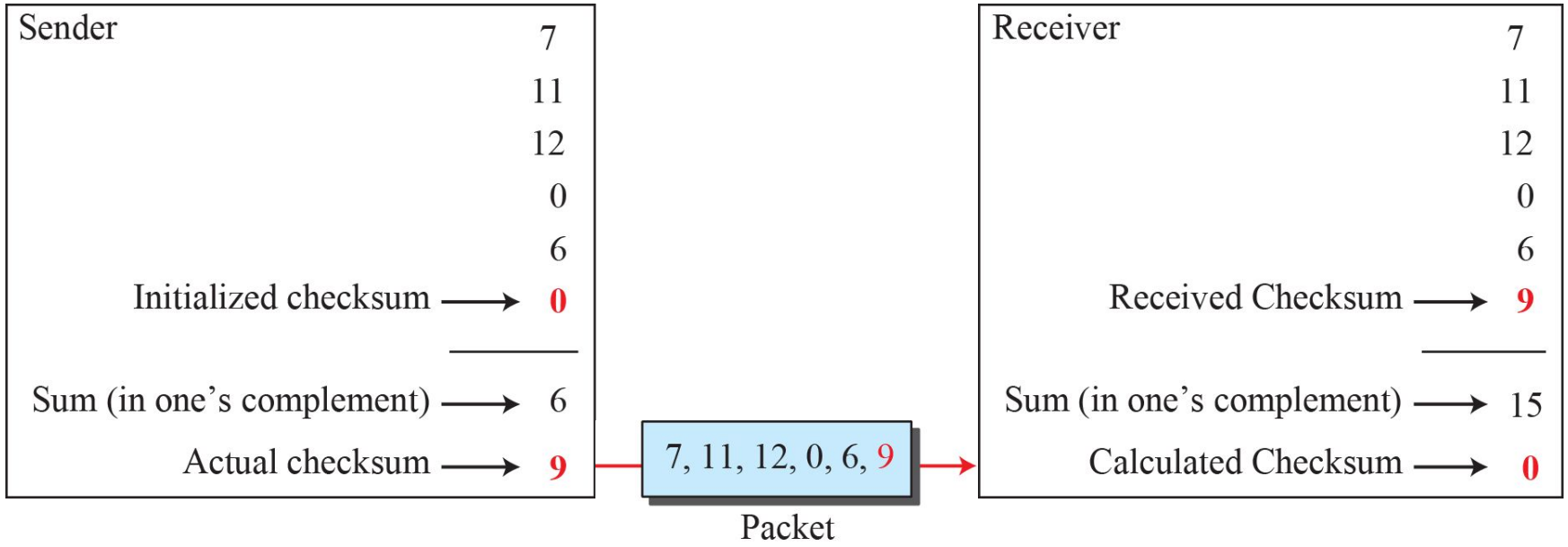
In the example, the decimal number 36 in binary is $(100100)_2$. To change it to a 4-bit number we add the extra leftmost bit to the right four bits as shown below.

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, 6). The receiver can add the first five numbers in one's complement arithmetic. If the result is 6, the numbers are accepted; otherwise, they are rejected.

Using idea of Checksum

- Let us use the idea of the checksum in previous example. The sender adds all five numbers in one's complement to get the sum = 6. The sender then complements the result to get the **checksum = 9**, which is 15 − 6. Note that 6 = (0110)2 and **9** = (1001)2; they are complements of each other.
- The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**). If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one's complement to get 15.



Procedure to find Checksum

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Summary

In this section we have discussed the following:

- ✔ Single bit and multi bit errors
- ✔ Finding Hamming Distance.
- ✔ Block coding method for error detection.
- ✔ CRC Method for error detection.
- ✔ Checksum method for error detection.


Thank
you!