CAP275: Data Communication and Networking
Unit-3: Data Link Layer – Error Control

Dr. Manmohan Sharma

School of Computer Applications

Lovely Professional University

Error Control

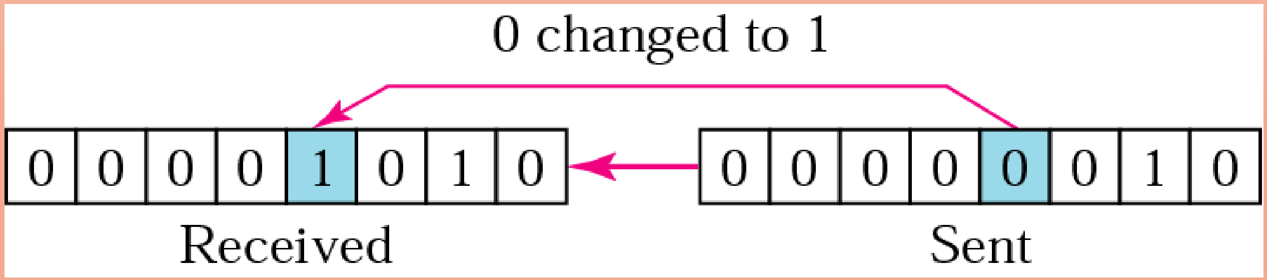
- Because of Attenuation, distortion, noise and interferences, errors during transmission are inevitable, leading to corruption transmitted bits.
- Longer the frame size and higher the probability of single bit error, lower is the probability receiving a frame without error.
- ***Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.***

Error

- When data is being transmitted from one machine to another, it may possible that data become corrupted on its way. Some of the bits may be altered, damaged or lost during transmission. Such a condition is known as **error**.
- **Types of Errors**
  - **Single bit error**: Only one bit gets corrupted. Common in Parallel transmission.
  - **Burst error:** More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.

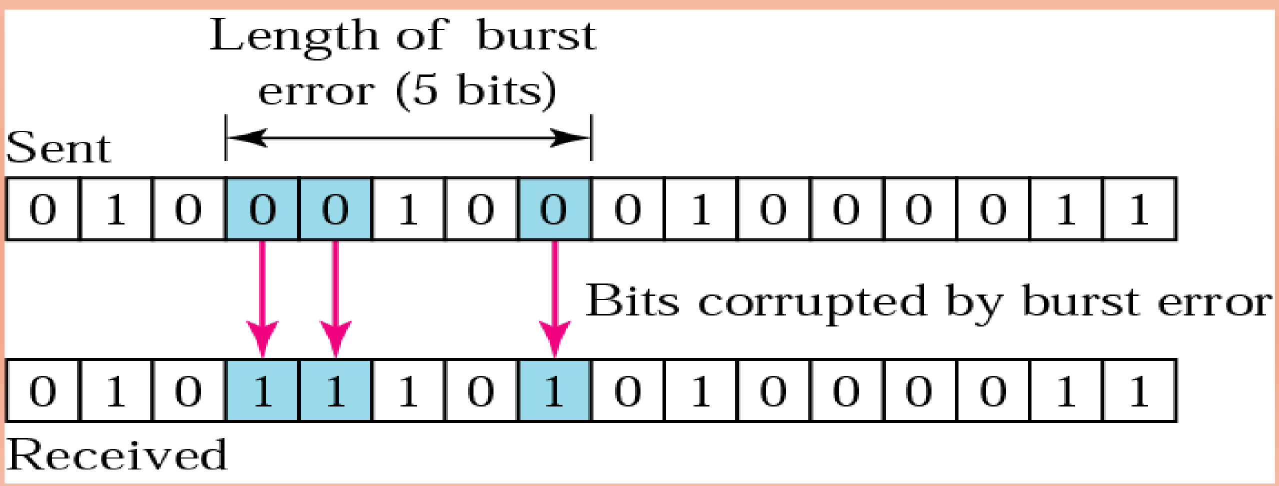
Single bit error

- The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1 as shown in Fig.
- Single bit errors are least likely type of errors in serial data transmission.
  - For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.



Burst error

- More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.
- The noise affects data; it affects a set of bits.
- The number of bits affected depends on the data rate and duration of noise.



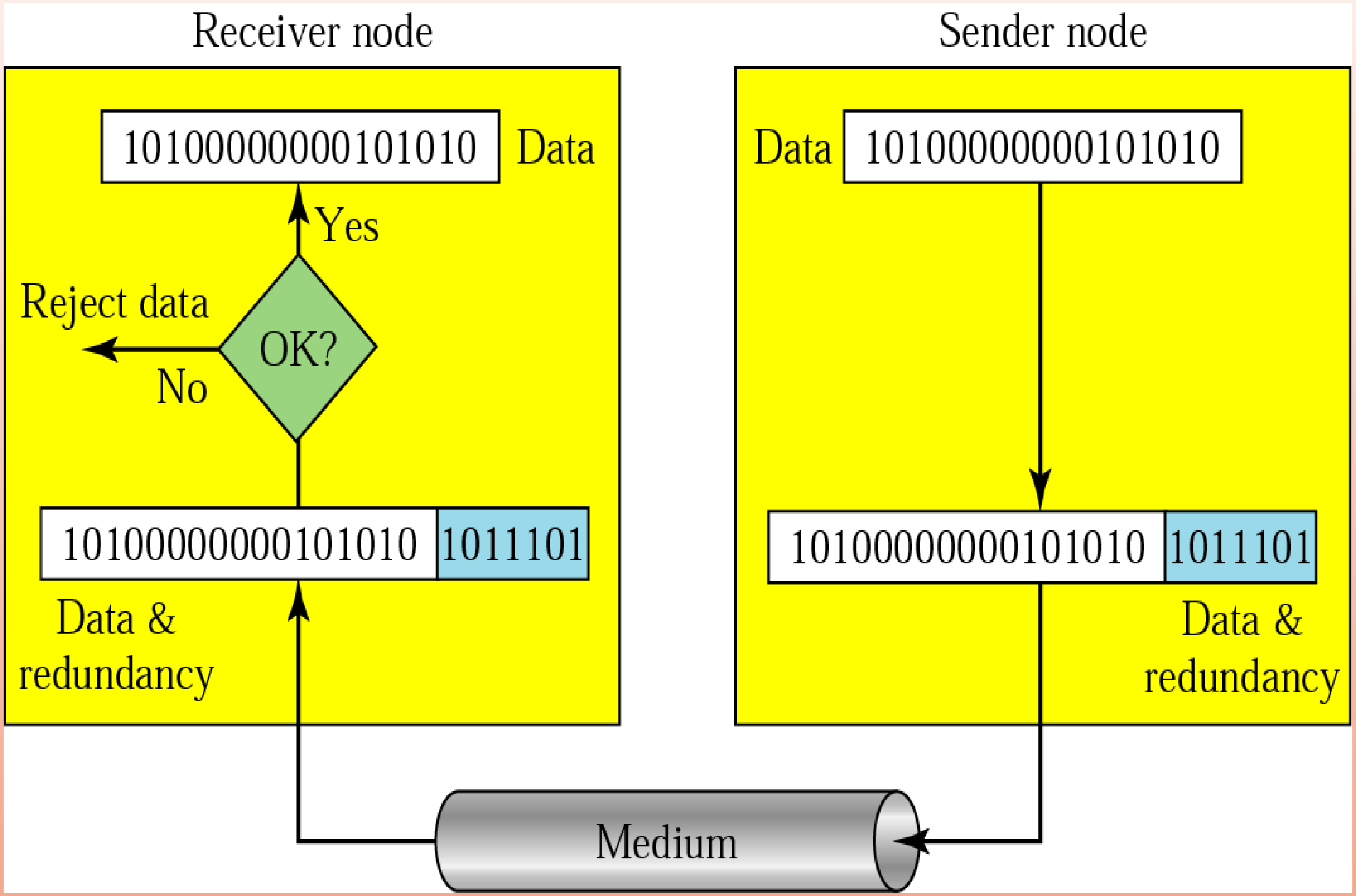
Error Detection Techniques

- Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors.
- **Redundancy** is the method in which some extra bits are added to the data so as to check whether the data contain error or not. Popular techniques are:
  - Simple Parity check
  - Two-dimensional Parity check
  - Checksum
  - Cyclic redundancy check
- An n-bit unit containing data and check-bits is often referred to as an n-bit codeword.

$$n = (m + r).$$

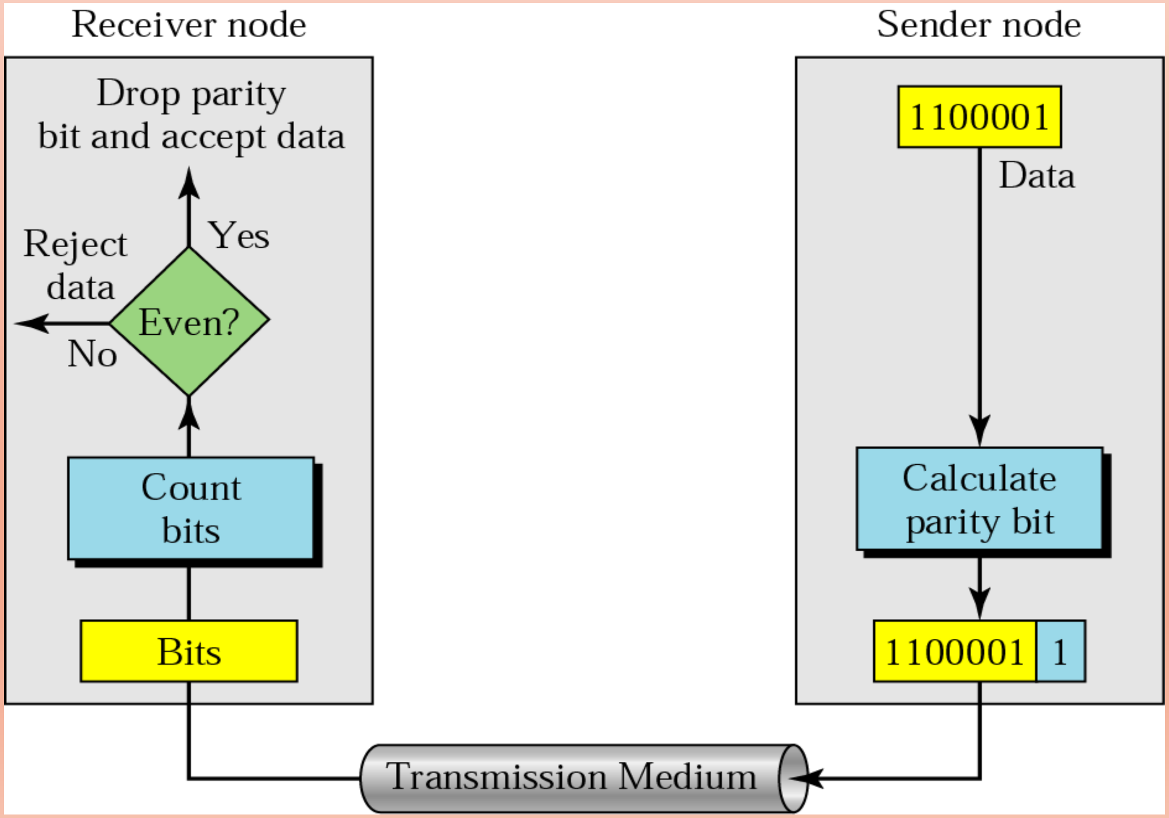
where

m	- data bits (i.e., message bits)
r	- redundant bits (or check bits).
n	- total number of bits


Receiver node
Sender node
10100000000101010
Data
Yes
OK?
Reject data
No
10100000000101010
1011101
Data & redundancy
Data
10100000000101010
10100000000101010
1011101
Data & redundancy
Medium

Simple Parity Check

- The simplest and most popular error detection scheme. Appends a Parity bit to the end of the data.
- In parity check, a parity bit is added to every data unit so that the total number of 1's is even (or odd for odd-parity).



- Suppose the sender wants to send the word *world*. In ASCII the five characters are coded as

1110111 1101111 1110010 1101100 1100100

- The following shows the actual bits sent

1110111<u>0</u> 1101111<u>0</u> 1110010<u>0</u> 1101100<u>0</u> 1100100<u>1</u>

6 6 4 4 4

(6,6,4,4,)

Received without error

- Now suppose the word in Example is received by the receiver without being corrupted in transmission.

11101110   11011110   11100100   11011000   11001001

- The receiver counts the 1s in each character and comes up with even numbers (6, 6, 4, 4, 4). The data are accepted.

Received with error

- Now suppose the word world in Example 1 is corrupted during transmission.

11111110   11011110   11101100   11011000   11001001

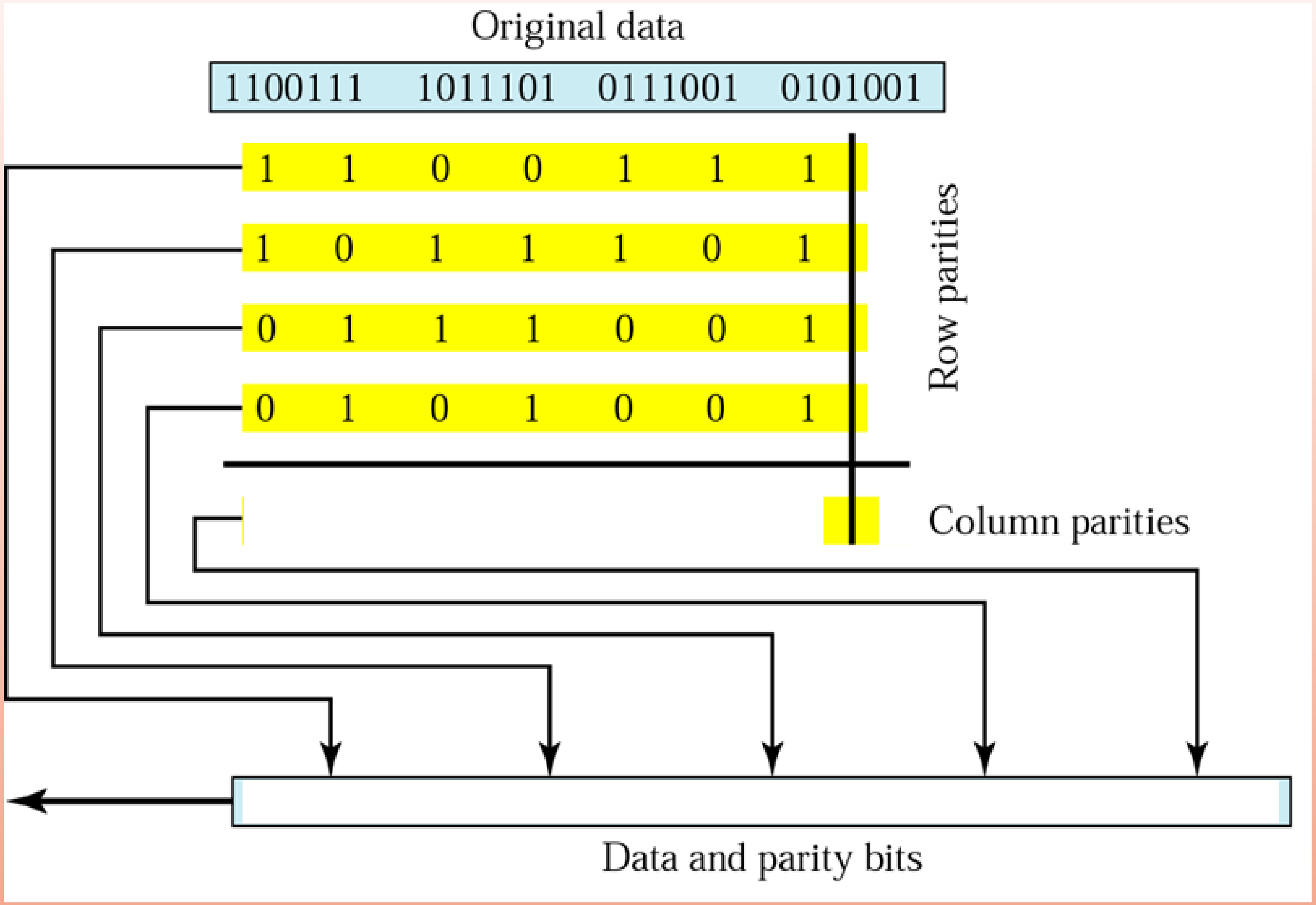
- The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, 4). The receiver knows that the data are corrupted, discards them, and asks for retransmission.

Performance of Simple Parity Check

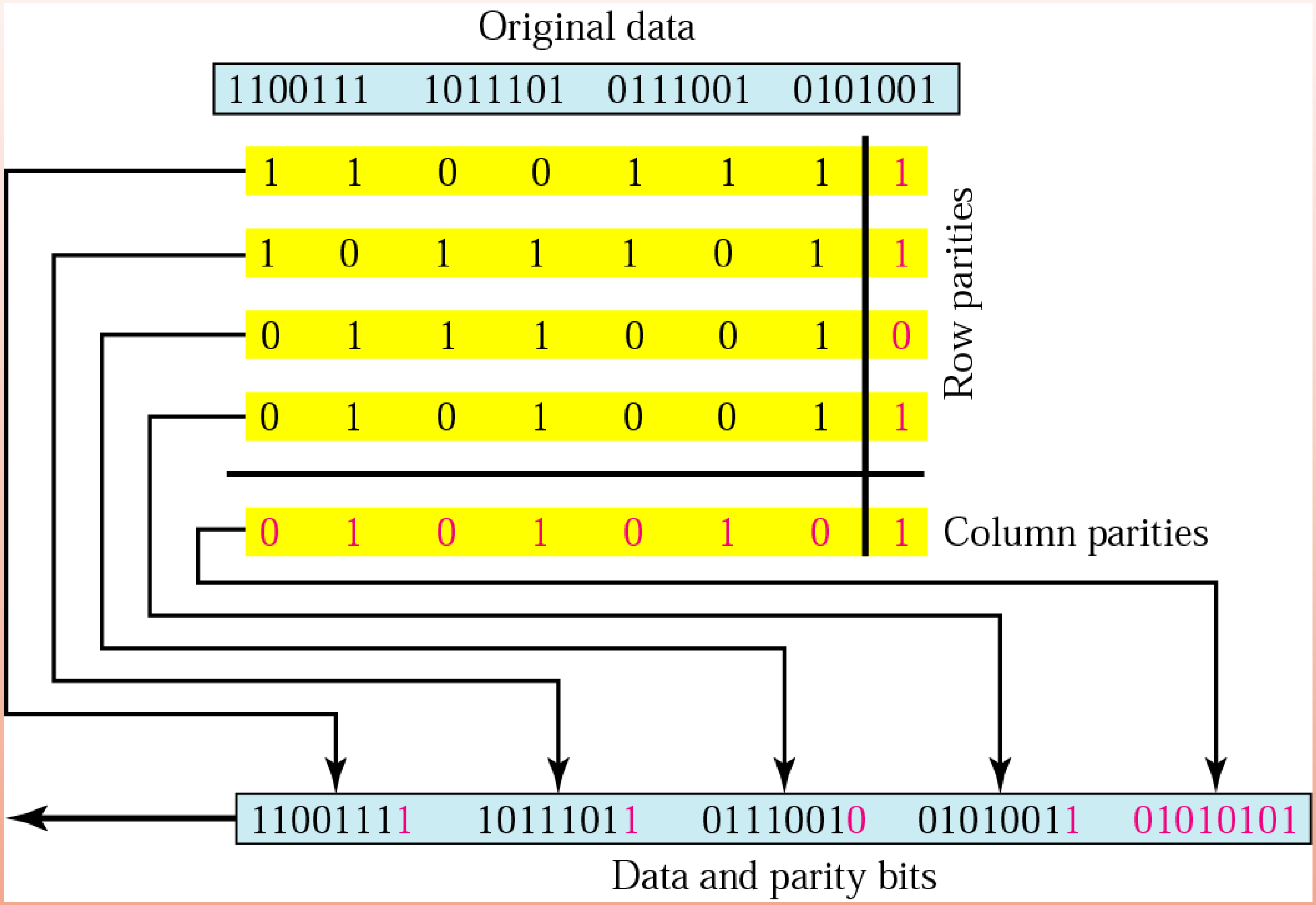
- Simple parity check can **detect all single-bit error**
- It can also detect burst error, if the number of bits in **even or odd.**
- The technique is not foolproof against burst errors that **invert more than one bit**. If an even number of bits is inverted due to error, the **error is not detected.**

Two-Dimensional Parity Checking

- Performance can be improved by using two dimensional parity check, which **organizes the block of bits in the form of table.**
- Parity check bits are **calculated from each row**, which is equivalent to a simple parity check.
- Parity check bits are also **calculated for all columns.**
- Both row parity and column parity are sent along with the data.
- At the receiving end these are compared with the parity bits calculated on the received data.



Original data
1100111 1011101 0111001 0101001
1 1 0 0 1 1 1
1 0 1 1 1 0 1
0 1 1 1 0 0 1
0 1 0 1 0 0 1
Row parities
Column parities
Data and parity bits


Original data
1100111 1011101 0111001 0101001
1 1 0 0 1 1 1 1
1 0 1 1 1 0 1 1
0 1 1 1 0 0 1 0
0 1 0 1 0 0 1 1
0 1 0 1 0 1 0 1
Row parities
Column parities
11001111 10111011 01110010 01010011 01010101
Data and parity bits

- Suppose the following block is sent:

  10101001   00111001   11011101   11100111 10101010

- However, it is hit by a burst noise of length 8, and some bits are corrupted.

  1010**0011**   **1000**1001   11011101   11100111 10101010

- When the receiver checks the parity bits, some of the bits do not follow the even-parity rule and the whole block is discarded.

  10100011   10001001   11011101   11100111   **1**0**101**0**1**0

Performance

- If two bits in one data unit are damaged and two bits in exactly same position in another data unit are also damaged, The 2-D Parity check **checker will not detect an error**.
- For example, if two data units: **11001100 and 10101100.** If first and second from last bits in each of them is changed, making the data units as **01001110 and 00101110**, the error cannot be detected by 2-D Parity check.

CHECKSUM

- In checksum error detection scheme, the **data is divided into k segments each of m bits.**
- At the sender's end the segments are added using **1's complement arithmetic to get the sum.**
- The sum is complemented to get the checksum. The **checksum** segment is sent **along with the data segments.**
- At the receiver's end the data unit is divided into k sections, each of m bits.
- All sections are added using one's complement to get the sum.
- The sum is complemented. If the result is zero, the data are accepted: otherwise, rejected.

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.

10101001   00111001

The numbers are added using one’s complement

```
               10101001
               00111001
               ------------
Sum            11100010
Checksum       00011101
```

The pattern sent is   10101001   00111001   **00011101**

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.

10101001 00111001

The numbers are added using one's complement

```
                10101001
                00111001
                ------------
Sum             11100010
Checksum        00011101
```

The pattern sent is 10101001 00111001 **00011101**

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

10101**111 11**111001 00011101

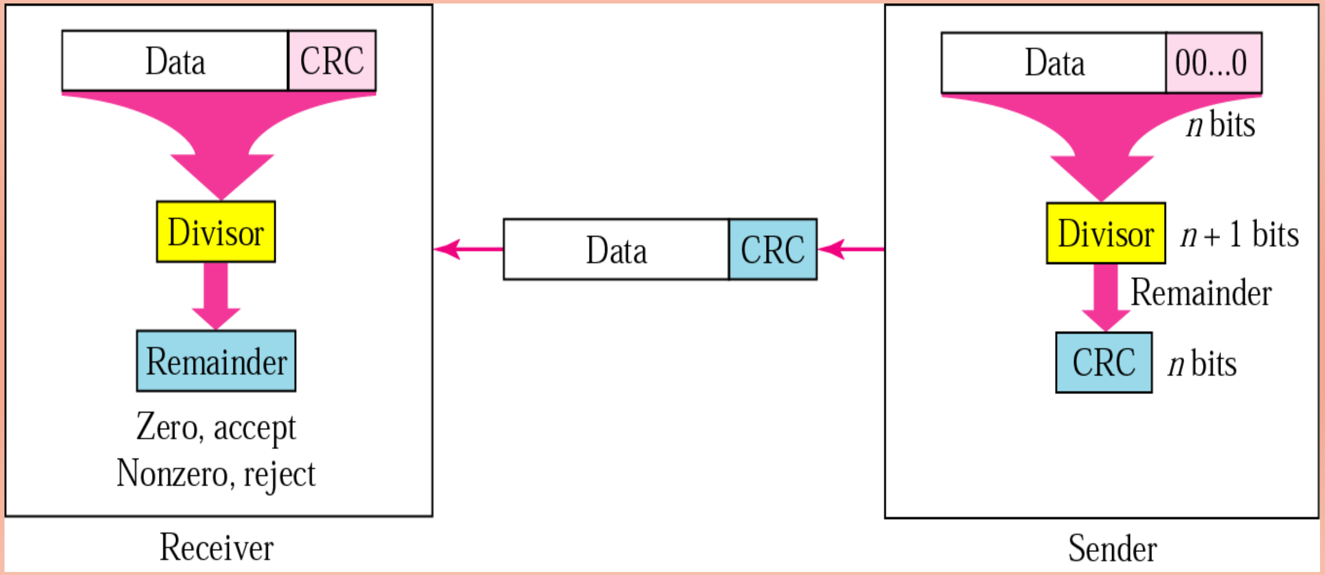
When the receiver adds the three sections, it gets

	10101111
	11111001
	00011101
Partial Sum	**1** 11000101
Carry	**1**
Sum	11000110
Complement	**00111001**

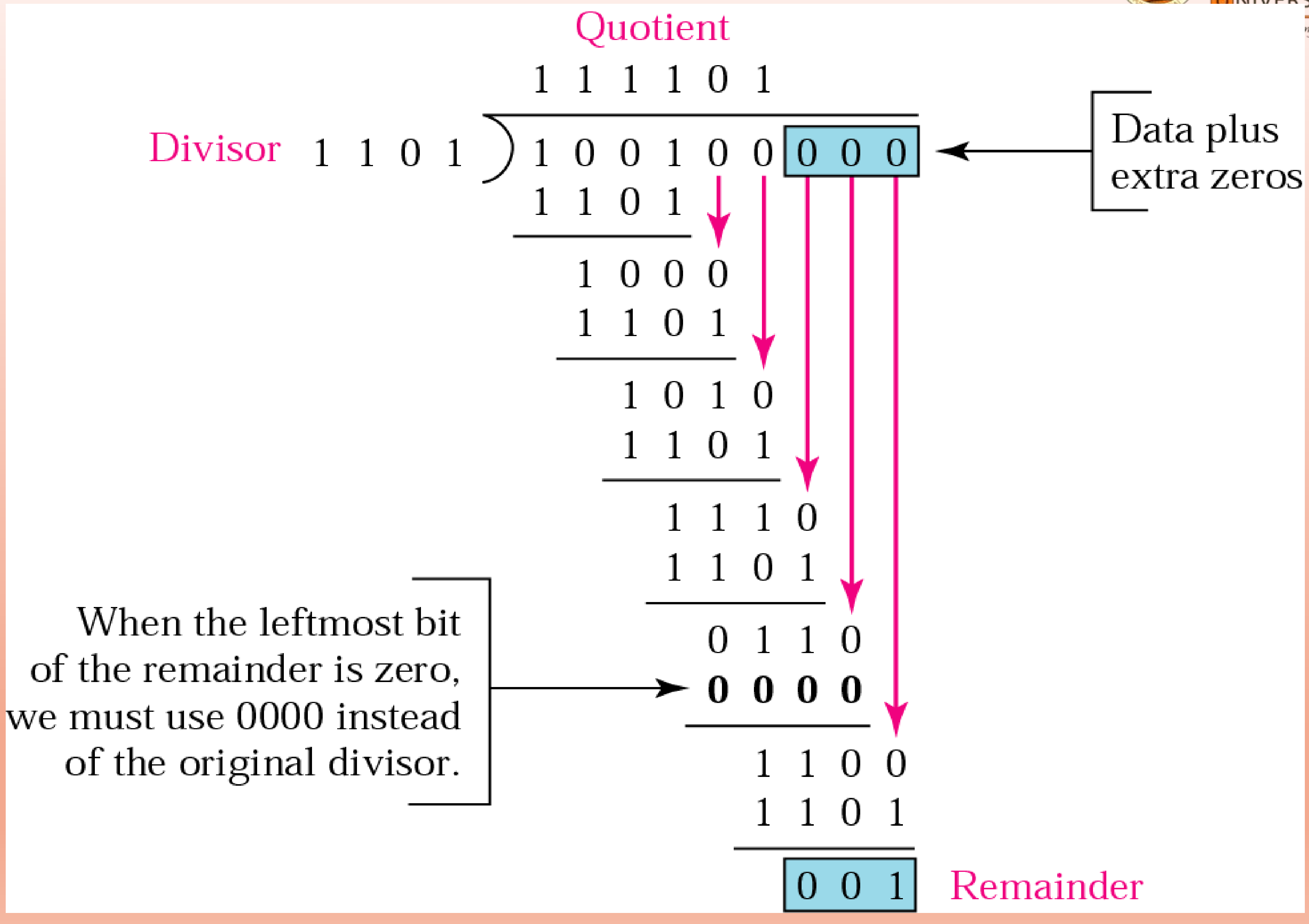
If the result is non zero then the pattern is corrupted.

Cyclic Redundancy Check

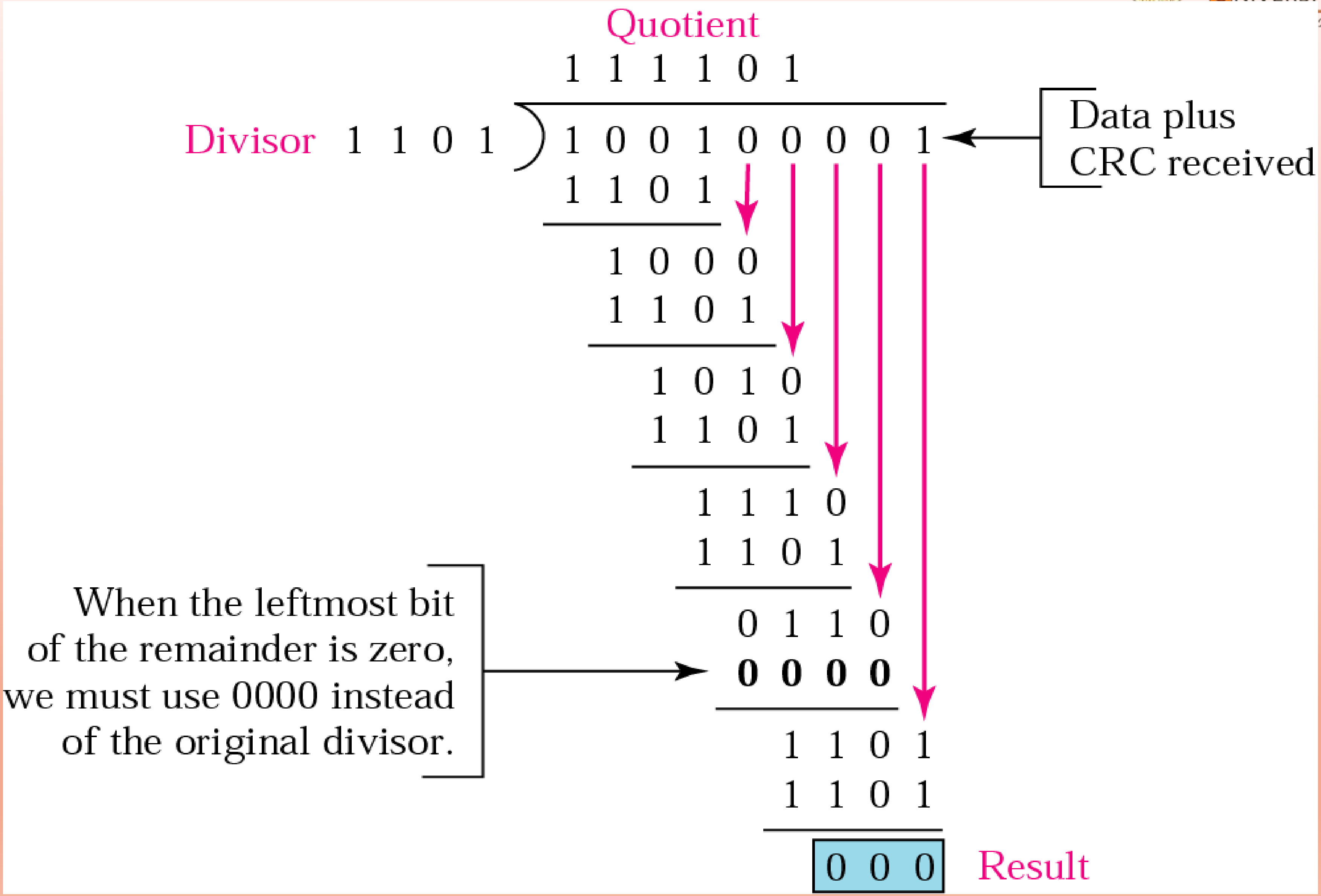
- One of the most powerful and commonly used error detecting codes.
- Given a m-bit block of bit sequence, the sender generates an n-bit sequence known as **frame sequence check (FCS),** so that the resulting frame, consisting of m+n bits exactly divisible by **same predetermined number.**
- The receiver divides the incoming frame by that number and, if there is **no reminder, assumes there was no error.**



At Sender's end



At receiver's end



Note: Remainder is zero, no error. Receiver can accept the data.

Performance of CRC

- CRC can detect all single-bit errors.
- CRC can detect all double-bit errors(three1's)
- CRC can detect any odd number of errors of less than the degree of the polynomial.
- CRC detects most of the larger burst errors with a high probability.

ERROR CORRECTING CODES

- A single additional bit can detect error, but it's not sufficient enough to correct that error too.
- For correcting an error one has to know the exact position of error, i.e. exactly which bit is in error (to locate the invalid bits).
  - For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in error. To this, we have to add some additional redundant bits.
- To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two. So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different values.
- Out of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations. So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2r states. Hence, 2r must be greater than d+r+1.

$$\mathbf{2^r >= d+r+1}$$

  - The value of r must be determined by putting in the value of d in the relation. For example, if d is 7, then the smallest value of r that satisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits (d+r = 7+4 =11).

Hamming Code

- A technique developed by R.W. Hamming provides a practical solution to manipulate these bits to discover which bit is in error. The solution or coding scheme he developed is commonly known as **Hamming Code**.
- Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits

Number of data bits m	Number of redundancy bits r	Total bits m + r
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Basic Approach

- To each group of m information bits k parity bits are added to form (m+k) bit code as shown in Fig. 3.2.8.
- Location of each of the (m+k) digits is assigned a decimal value.
- The k parity bits are placed in positions 1, 2, …, 2^{k-1} positions.
- K parity checks are performed on selected digits of each codeword.
- At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.

11	10	9	8	7	6	5	4	3	2	1
d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

Calculation of Redundant Bits

r_1 will take care of these bits.

11		9		7		5		3		1
d	d	d	r_8	d	d	d	r_4	d	r_2	r_1

r_2 will take care of these bits.

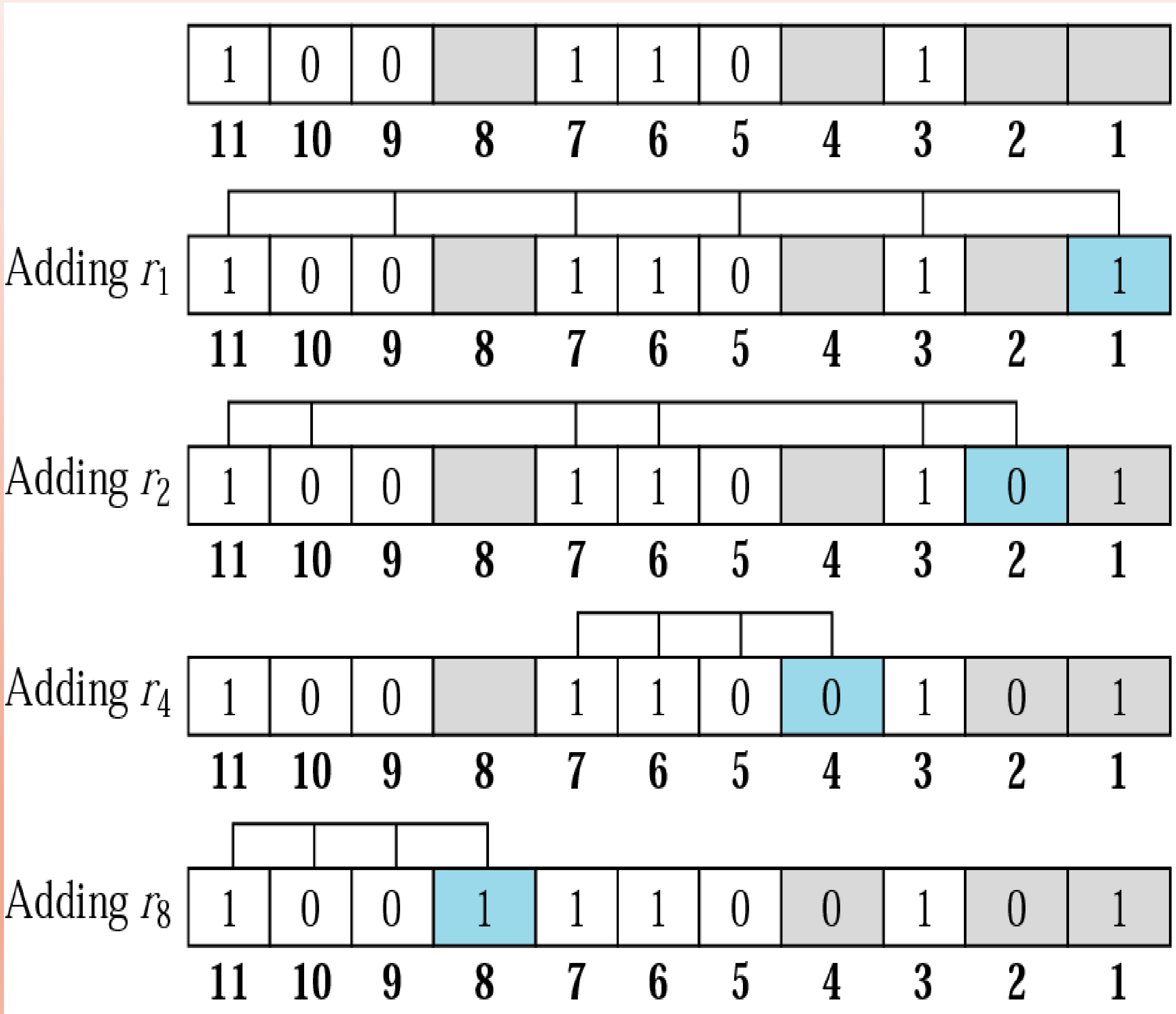
11	10			7	6			3	2	
d	d	d	r_8	d	d	d	r_4	d	r_2	r_1

r_4 will take care of these bits.

				7	6	5	4			
d	d	d	r_8	d	d	d	r_4	d	r_2	r_1

r_8 will take care of these bits.

11	10	9	8							
d	d	d	r_8	d	d	d	r_4	d	r_2	r_1

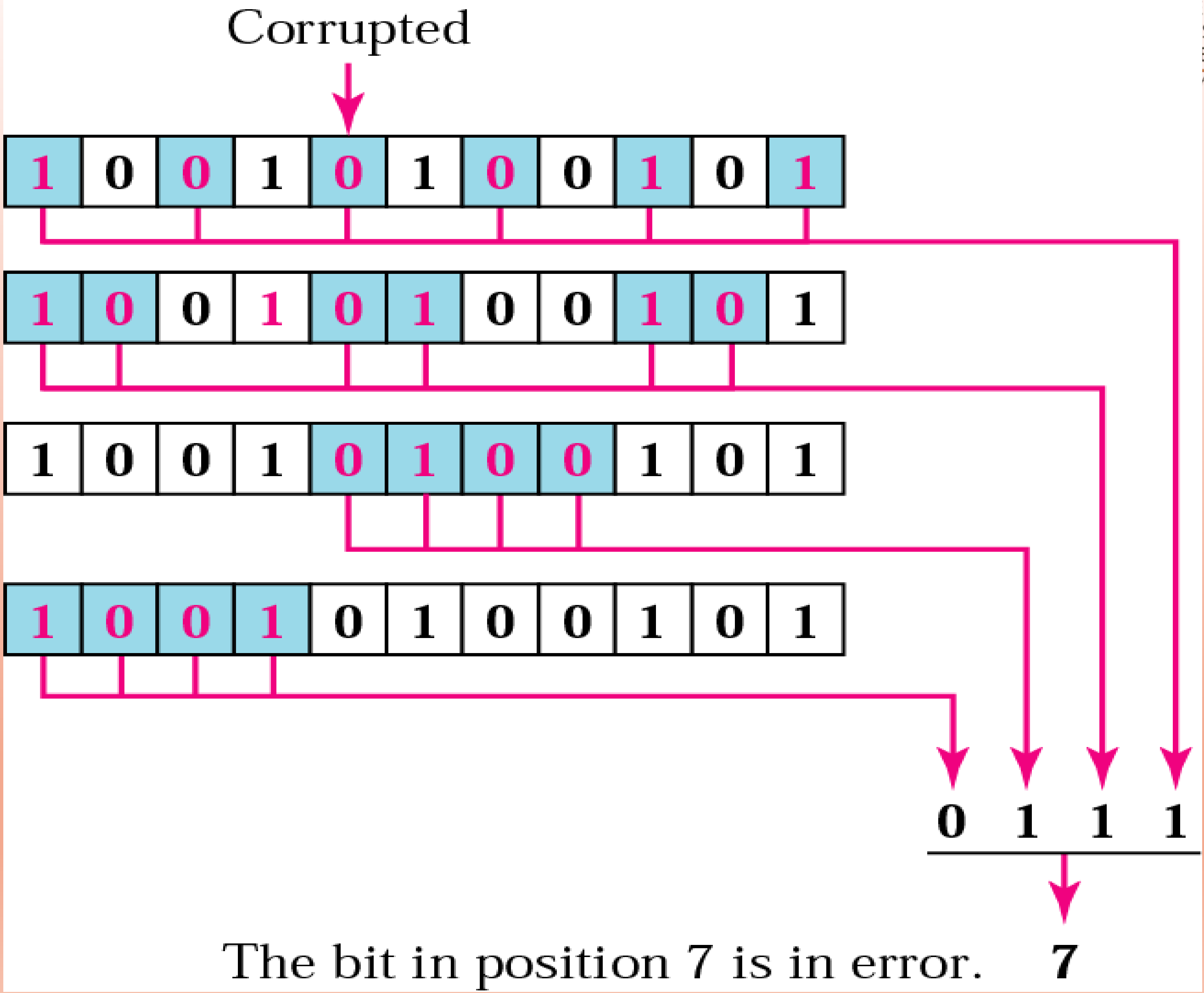


Data:

1 0 0 1 1 0 1

Code:

1 0 0 1 1 1 0 0 1 0 1



The bit in position 7 is in error.