COMPUTER NETWORKS

Lecture Notes

Module II

R.K.Dalei

rdalei@silicon.ac.in

Department of Computer Science Engineering & Application

Out Line of Module II

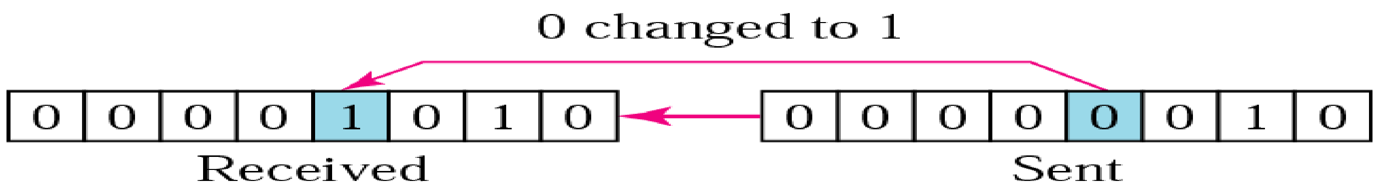
- Data-Link Layer
  - Error detection and correction
  - Data link control and protocols
  - Point-to-Point access (PPP)
  - Multiple Access
  - Local Area Networks: Ethernet
  - Wireless LANS
  - Virtual Circuit Switching: Frame Relay and ATM

Text: *"Data Communications and Networking"* Third Edition, Behrouz A Forcuzan, Tata Mc Graw-Hill. Chapter 10 - Chapter 15 and Chapter 18

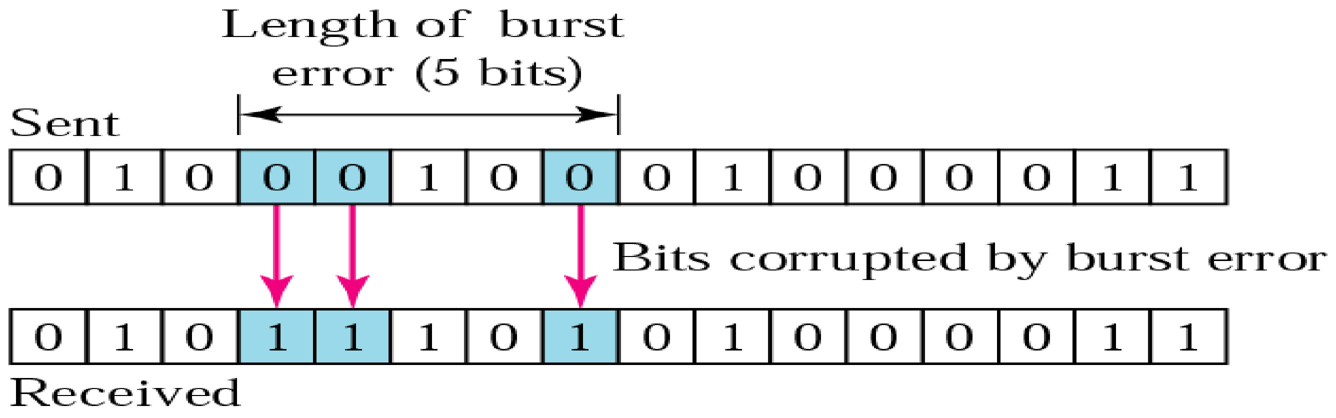
- *Error Detection and Correction*
  - *Types of Errors*
  - *Detection*
  - *Error Correction*
- *Data Link Control and Protocols*
  - *Stop and Wait ARQ*
  - *Go-Back-N ARQ*
  - *Selective Repeat ARQ*
  - *HDLC*
  - *PPP*

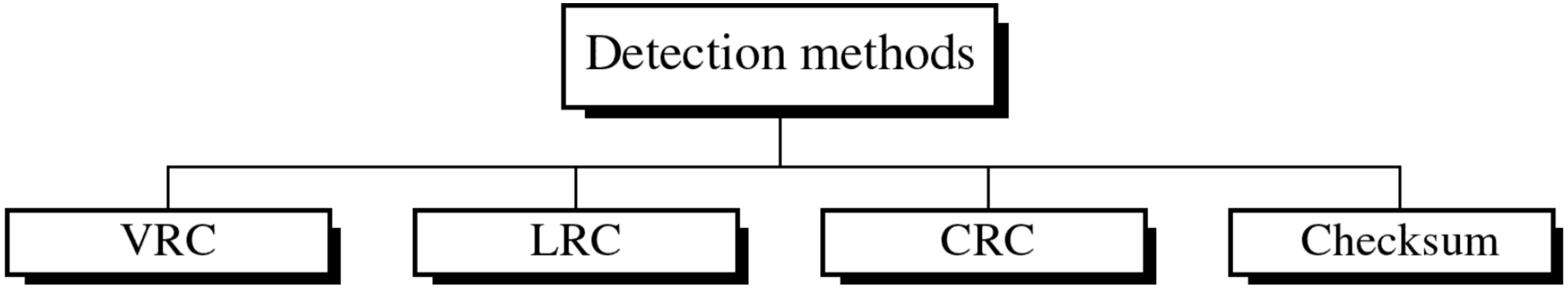
Errors in Transmission

- Data can be corrupted during transmission, for reliable communication, errors must be detected and corrected
- Two types of transmission errors
  - Single bit error: only one bit of the data unit is changed



  - Burst error: two or more bits in the data unit have changed



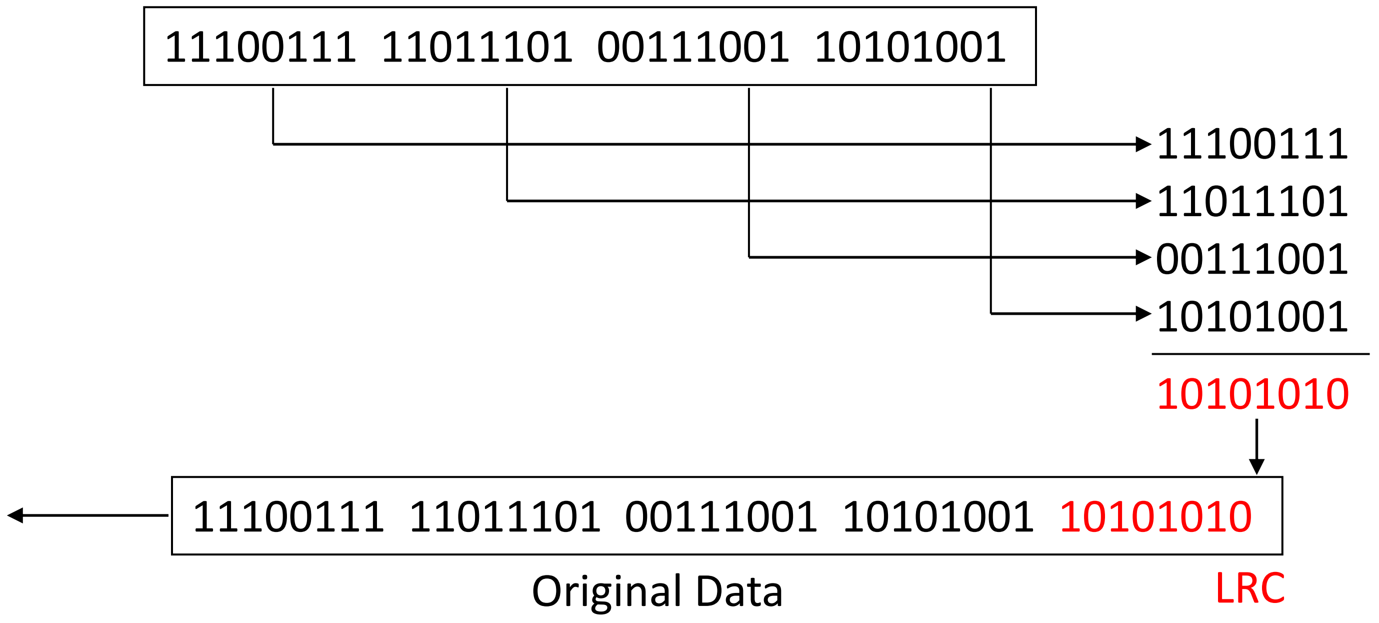

Detection methods
VRC
LRC
CRC
Checksum

Error Detection Methods

- **Vertical Redundancy Check (VRC)**
  - Append a single bit at the end of data block such that the number of ones is even
    → Even Parity (odd parity is similar)
    0110011 → 01100110
    0110001 → 01100011
  - VRC is also known as **Parity Check**
  - Performance:
    - Detects all odd-number errors in a data block

Error Detection Methods

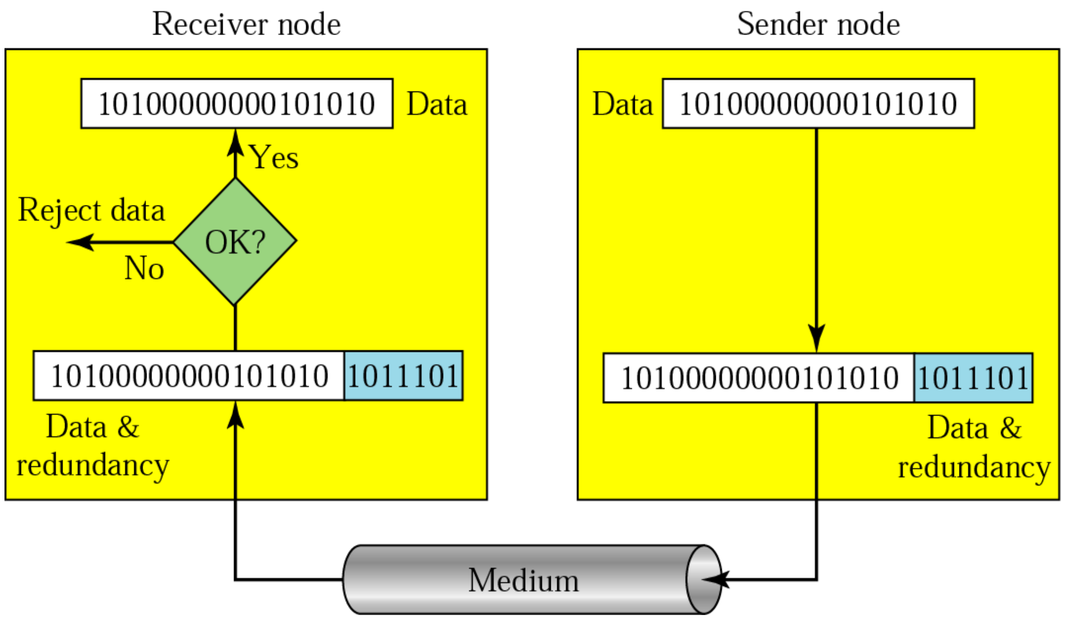
- **Longitudinal Redundancy Check (LRC)**
  - Organize data into a table and create a parity for each column



Detection mechanisms

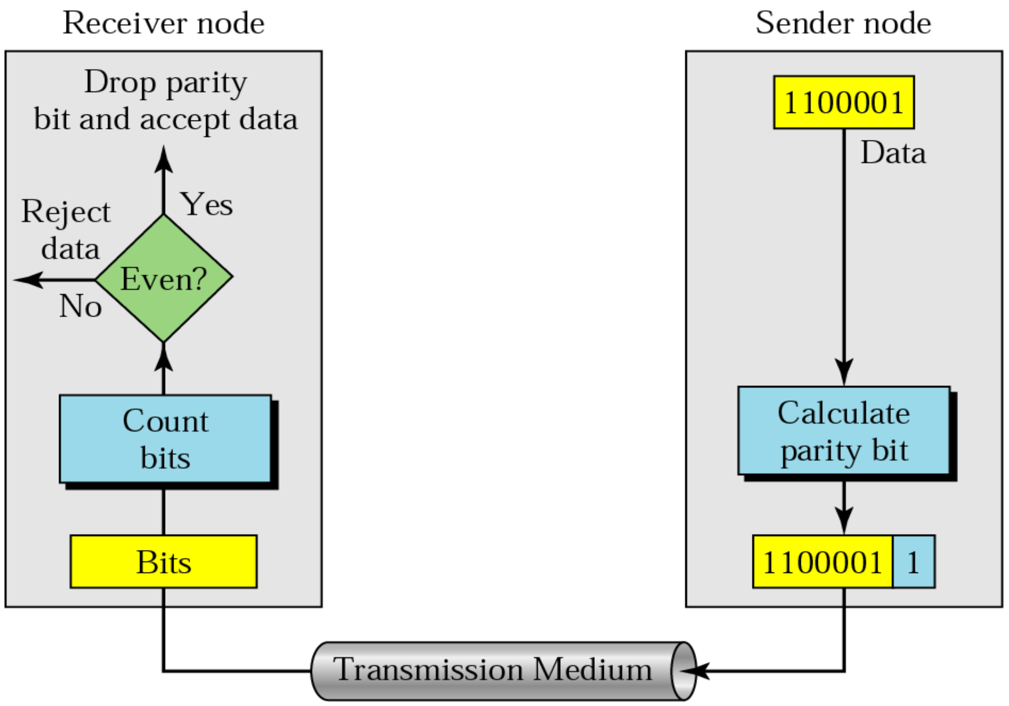
- To detect errors in transmission the concept of redundancy is used i.e. adding extra bits along with data and transmitted to other end.

- Detection Methods
  - Parity Check
  - Cyclic Redundancy Check (CRC)
  - Checksum



Parity check

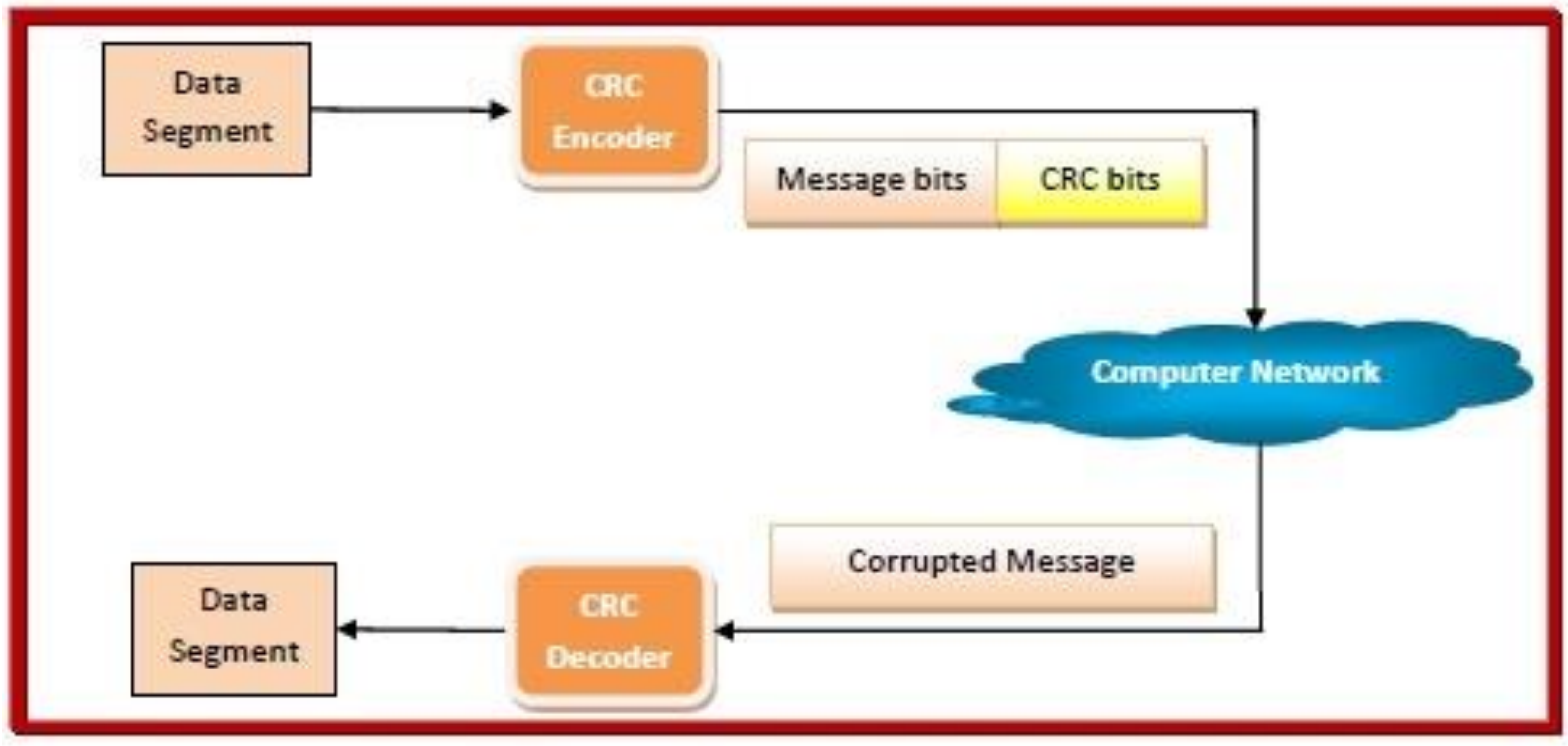
- It can be simple or two dimensional
- Simple Parity Check
  - A redundant bit called parity bit is added to every data unit so that the total no of 1s (including parity bit) in the unit becomes even or odd
  - Performance:
    - it can detect all single bit errors,
    - also it can detect burst errors of odd size i.e. 1, 3, 5 etc.
    - However it can not detect burst errors of even size, i.e. 2, 4, 6 etc



Cyclic Redundancy Check (CRC)

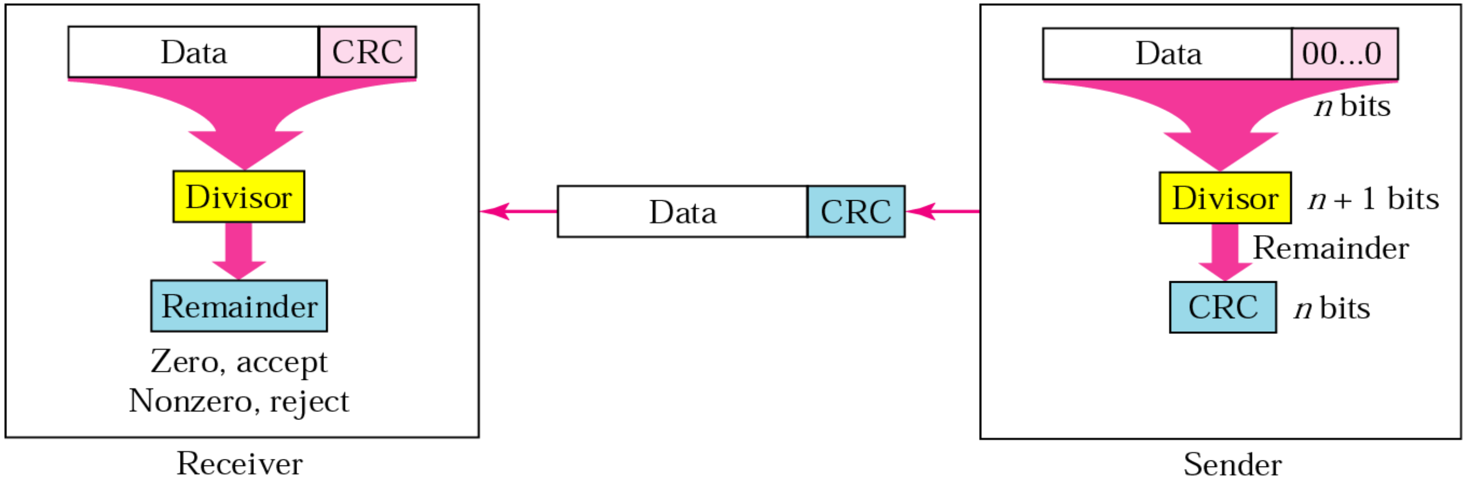
Cyclic Redundancy Check (CRC) is a block code invented by W. Wesley Peterson in 1961. It is commonly used to detect accidental changes to data transmitted via telecommunications networks and storage devices.

CRC involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials.



Cyclic Redundancy Check (CRC)

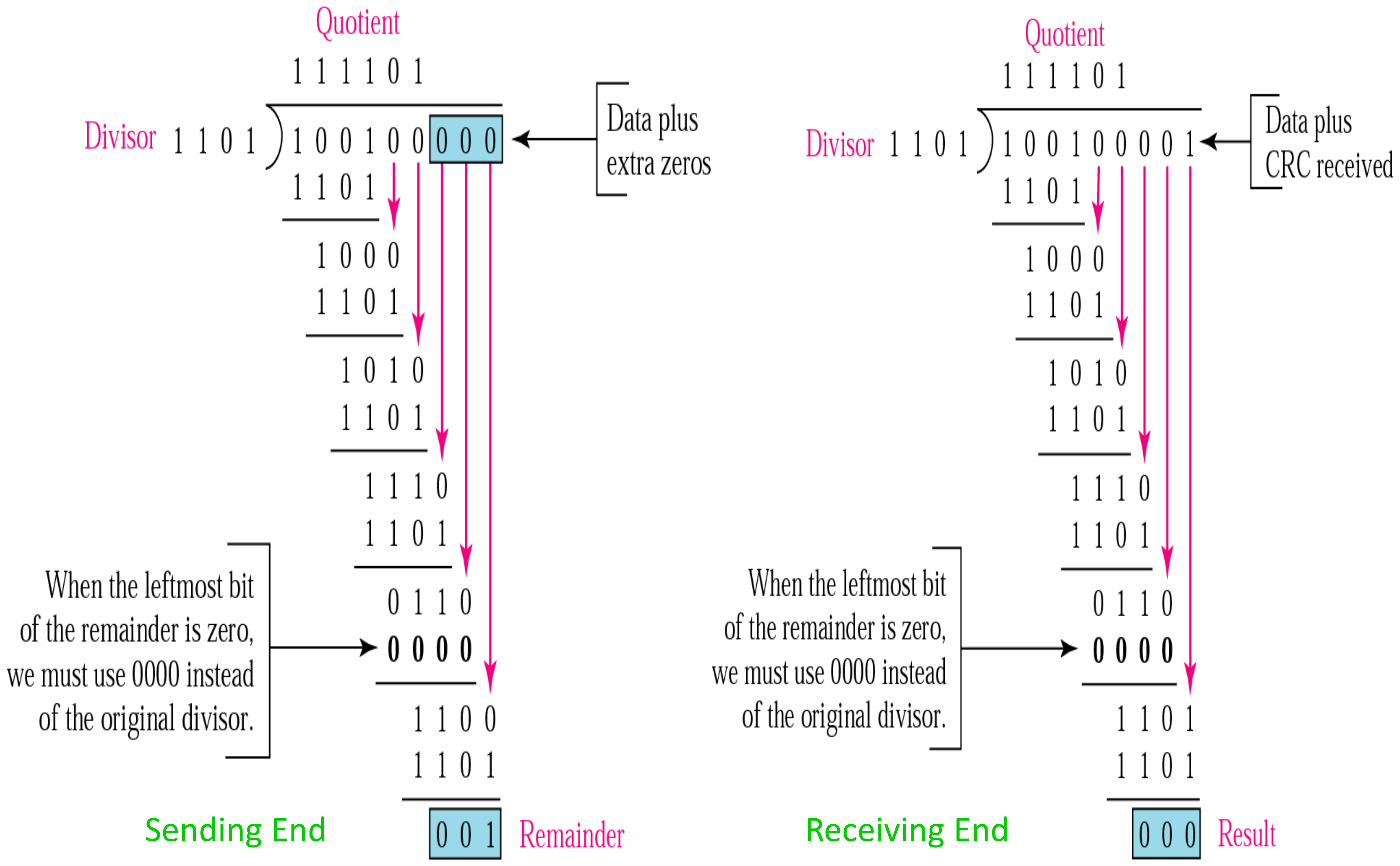
- It is the Most powerful method based on binary division
- The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor; the remainder is the CRC



▪Method

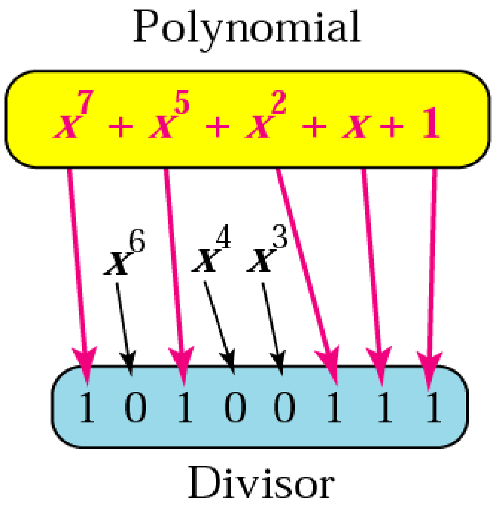
- A string of n 0s appended to data unit, n is one less then the number of bits present in the predetermined divisor
- Data unit appended with 0s is divided by the divisor using modulo-2 division and the remainder is collected
- Remainder replaces n no of 0s appended to data unit and transmitted
- At the receiving end the data along with CRC is divided by the divisor and if remainder is Zero than no error

Binary Division in CRC: An Example



CRC contd.

- Modulo-2 division
  - If left most bit of dividend is 1 then quotient will be 1 else 0
  - 0-0 = 0, 1-1=0, 1-0=1, 0-1=1
  - Note: we are dealing with bit-patterens not with quantitative values
- Divisor
  - It is calculated from an algebraic polynomial
  - Properties
    - It should not be divisible by x, guarantees that all burst errors of a length equal to the degree of polynomial are detected
    - It should be divisible by x+1, guarantees that all burst errors affecting an odd number of bits are detected



Checksums

This is a block code method where a checksum is created based on the data values in the data blocks to be transmitted using some algorithm and appended to the data.
When the receiver gets this data, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error.

Error Detection by Checksums

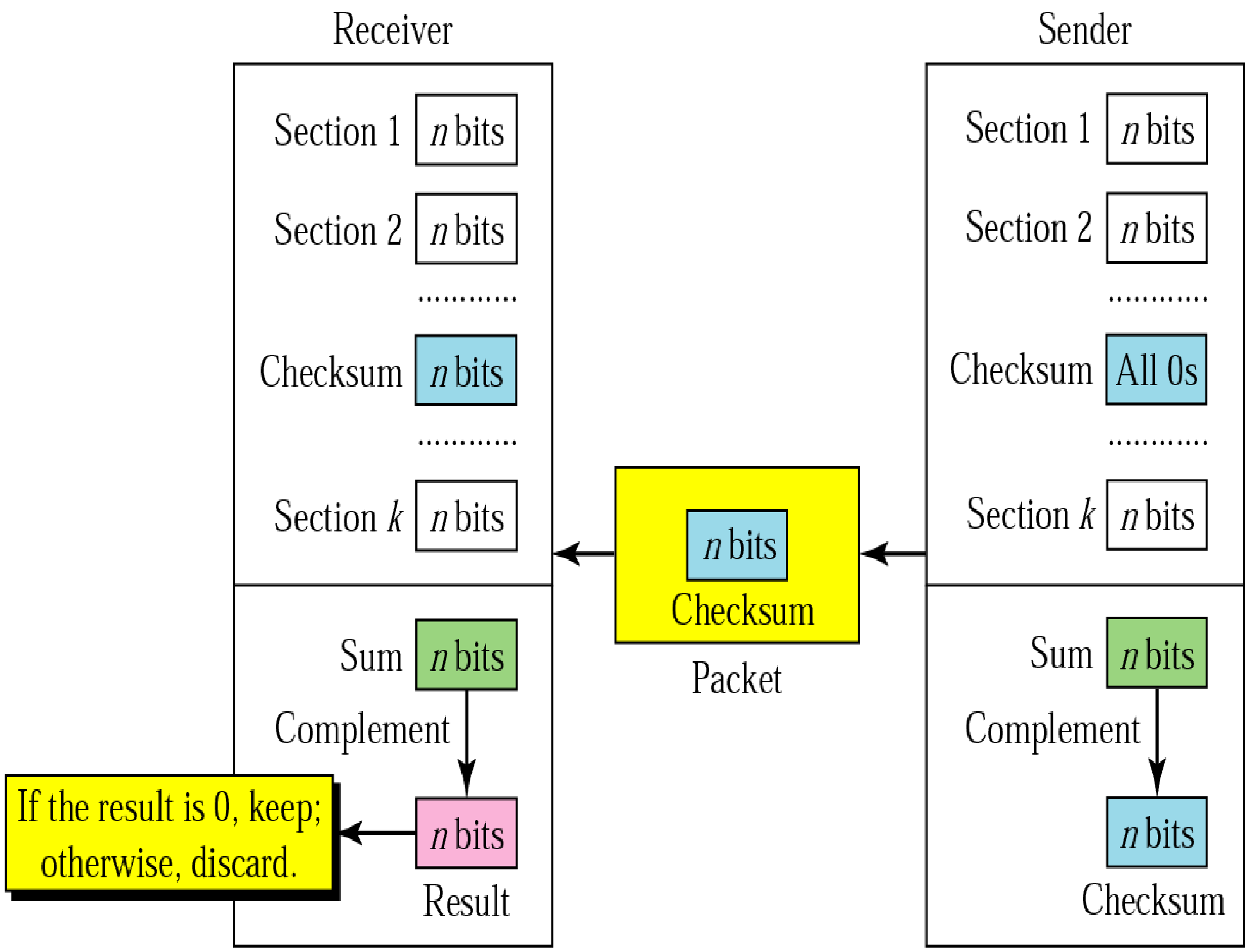
For error detection by checksums, data is divided into fixed sized frames or segments.
Sender's End − The sender adds the segments using 1's complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.

Receiver's End − The receiver adds the incoming segments along with the checksum using 1's complement arithmetic to get the sum and then complements it.
If the result is zero, the received frames are accepted; otherwise they are discarded.

Checksum

- A simple but effective method based on redundancy

- Method
  - Subdivide the data unit into equal segment of n bits
  - These segments are added using ones complement arithmetic so that result is also n bits
  - The sum is complemented and appended at the end of original data unit as redundant bits
  - At the receiving end all the groups are added again and if the result is zero then there is no error
- Performance
  - Detects all errors involving an odd number of bits as well as most errors involving an even number of bits


Receiver
Section 1
n bits
Section 2
n bits
Checksum
n bits
Section k
n bits
Sum
n bits
Complement
n bits
Result
If the result is 0, keep; otherwise, discard.
n bits
Checksum
Packet
Sender
Section 1
n bits
Section 2
n bits
Checksum
All 0s
Section k
n bits
Sum
n bits
Complement
n bits
Checksum

Example

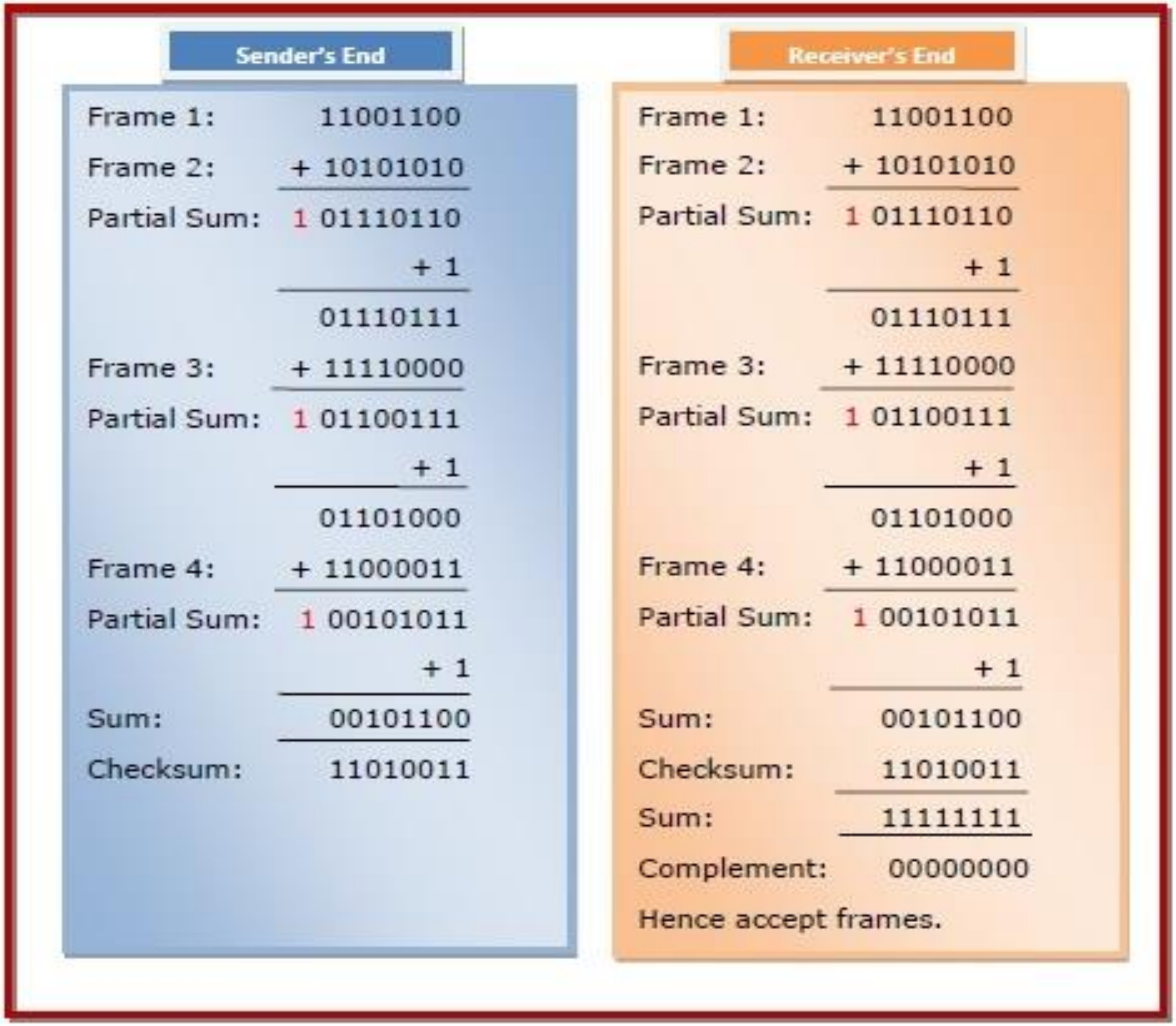
- Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.
- 10101001   00111001
- The numbers are added using one's complement
- 10101001
- 00111001
- ------------
- Sum   11100010 , Checksum   **00011101**
- The pattern sent is   10101001   00111001   **00011101**
- Now suppose the receiver receives the pattern sent without any error.
- Receiver adds all sections using same method

```
              10101001

              00111001

              00011101
             ------------
Sum           11111111 , Complement    00000000
```

means that the pattern is OK.



Sender's End

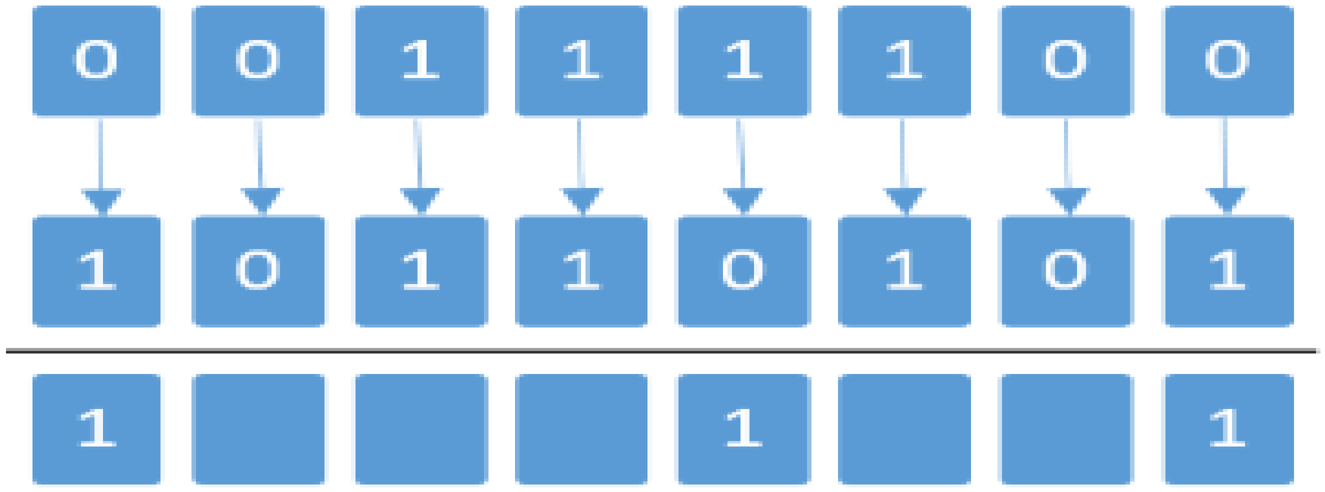
```
Frame 1:          11001100
Frame 2:        + 10101010
                ----------
Partial Sum:    1 01110110
                       + 1
                ----------
                  01110111
Frame 3:        + 11110000
                ----------
Partial Sum:    1 01100111
                       + 1
                ----------
                  01101000
Frame 4:        + 11000011
                ----------
Partial Sum:    1 00101011
                       + 1
                ----------
Sum:              00101100
                ----------
Checksum:         11010011
```

Receiver's End

```
Frame 1:          11001100
Frame 2:        + 10101010
                ----------
Partial Sum:    1 01110110
                       + 1
                ----------
                  01110111
Frame 3:        + 11110000
                ----------
Partial Sum:    1 01100111
                       + 1
                ----------
                  01101000
Frame 4:        + 11000011
                ----------
Partial Sum:    1 00101011
                       + 1
                ----------
Sum:              00101100
Checksum:         11010011
                ----------
Sum:              11111111
                ----------
Complement:       00000000
Hence accept frames.
```

Hamming Distance

The Hamming distance between two code words is simply the number of bits that are disparate between two bit strings as demonstrated in figure 1. Typically, hamming distance is denoted by the function d(x, y) where x and y are code words.



Hamming Distance = 3

Figure 1

We can count the number of 1s in the Xoring of two words

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

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

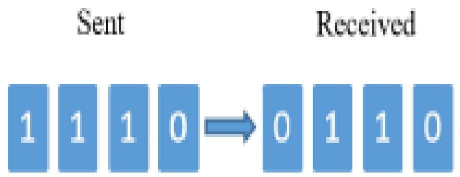
2. *The Hamming distance d(10101, 11110) is 3 because*

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

Nearest neighbor error correction involves first defining code words, typically denoted as C, that are known to both the source and destination.

Any received codeword not contained in C is obviously the result of noise.

Upon identifying an erroneous codeword, nearest neighbor decoding calculates the Hamming distance between it and every codeword contained in C.
The codeword with the smallest Hamming distance has a high probability of being correct. See figure 2.



$$C = \{0000, 1110, 1011\}$$

$$d(0110, 0000) = 2, \quad d(0110, 1110) = 1, \quad d(0110, 1011) = 3$$

Figure 2

Example

Find the minimum Hamming distance of the coding scheme

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

Solution

We first find all Hamming distances.

The d_{min} =2.

$d(000, 011) = 2$	$d(000, 101) = 2$	$d(000, 110) = 2$	$d(011, 101) = 2$
$d(011, 110) = 2$	$d(101, 110) = 2$		

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.

Generation of Hamming Code

- Let number of data bits be *m*, and redundant bits be *r*
- Therefore total no of bits sent is *m+r*
- The number of redundant bits(r) can be computed from the equation

  $2^r \geq m+r+1$ must be satisfied, with minimum r value.
- Now given the value of m, r can be calculated
- e.g. if m=7 then r has to be 4, i.e.

  $2^4 \geq 7+4+1$ is satisfied, r= 4.

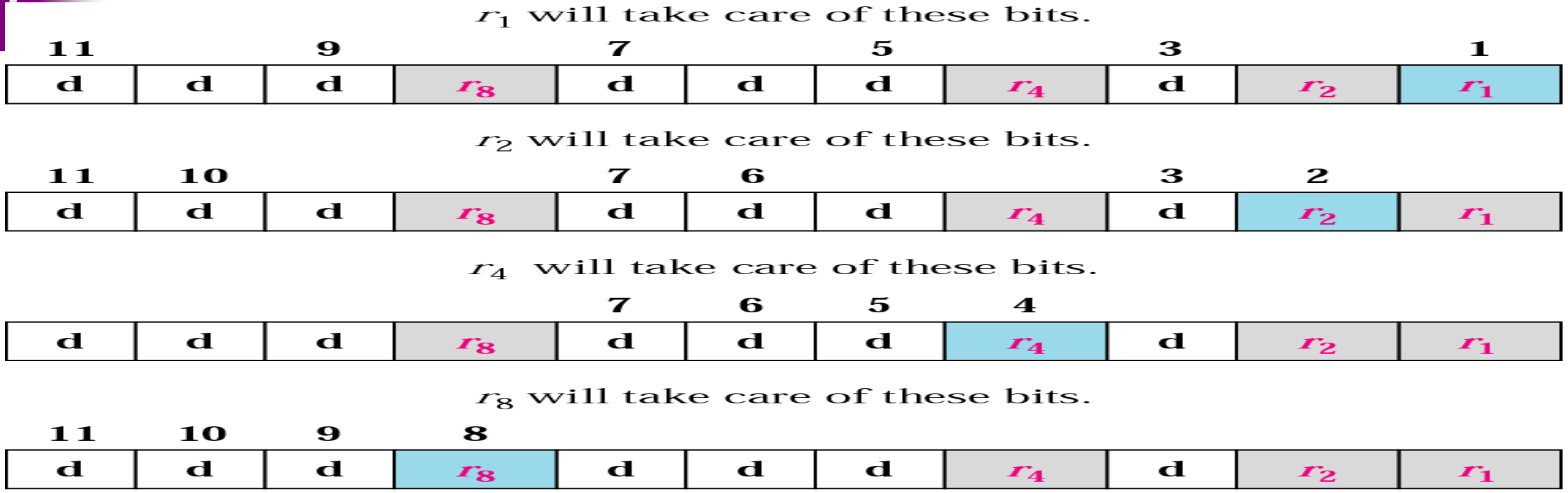
- Method:
  - Let number of data bits (m) be 7 => number of redundant bits (r) are 4
  - Position of redundant bits are defined as 1st, 2nd, 4th, and 8th, i.e. 2^x, x=0,1,2,...

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

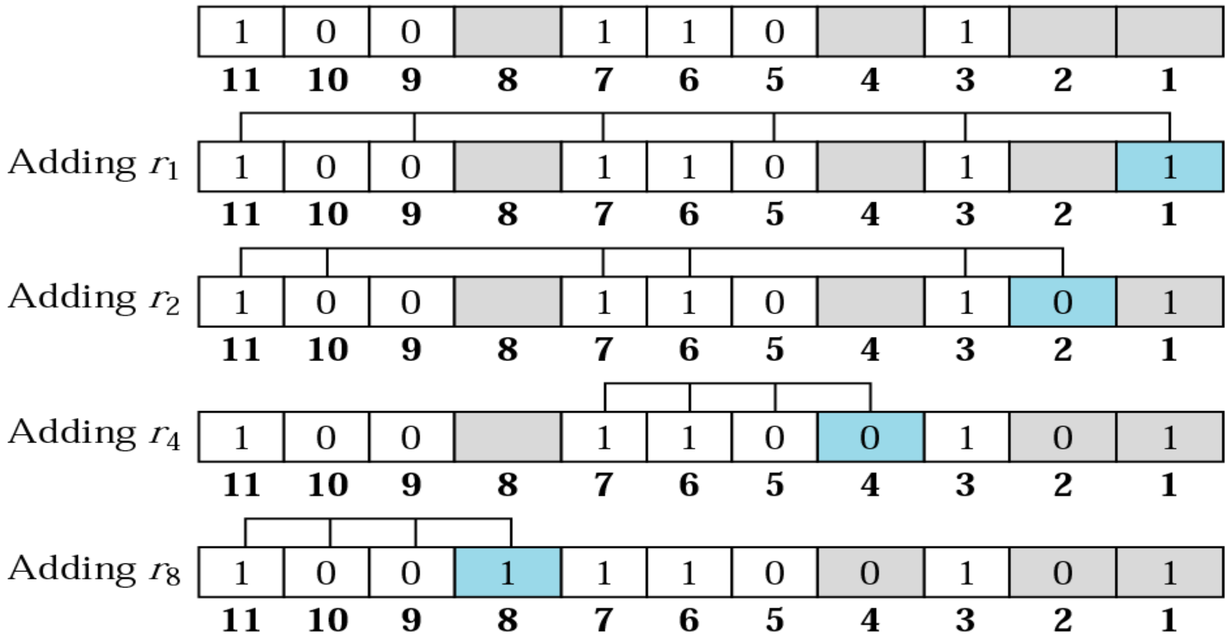
combination of data bits given as follows

- r_1: data bits 1, 3, 5, 7, 9, 11 (binary value containing 1 at 1st position)
- r_2: data bits 2, 3, 6, 7, 10, 11 (binary value containing 1 at 2st position)
- r_3: data bits 4, 5, 6, 7 (binary value containing 1 at 3rd position)
- r_4: data bits 8, 9, 10, 11 (binary value containing 1 at 4th position)

Redundancy bits calculation



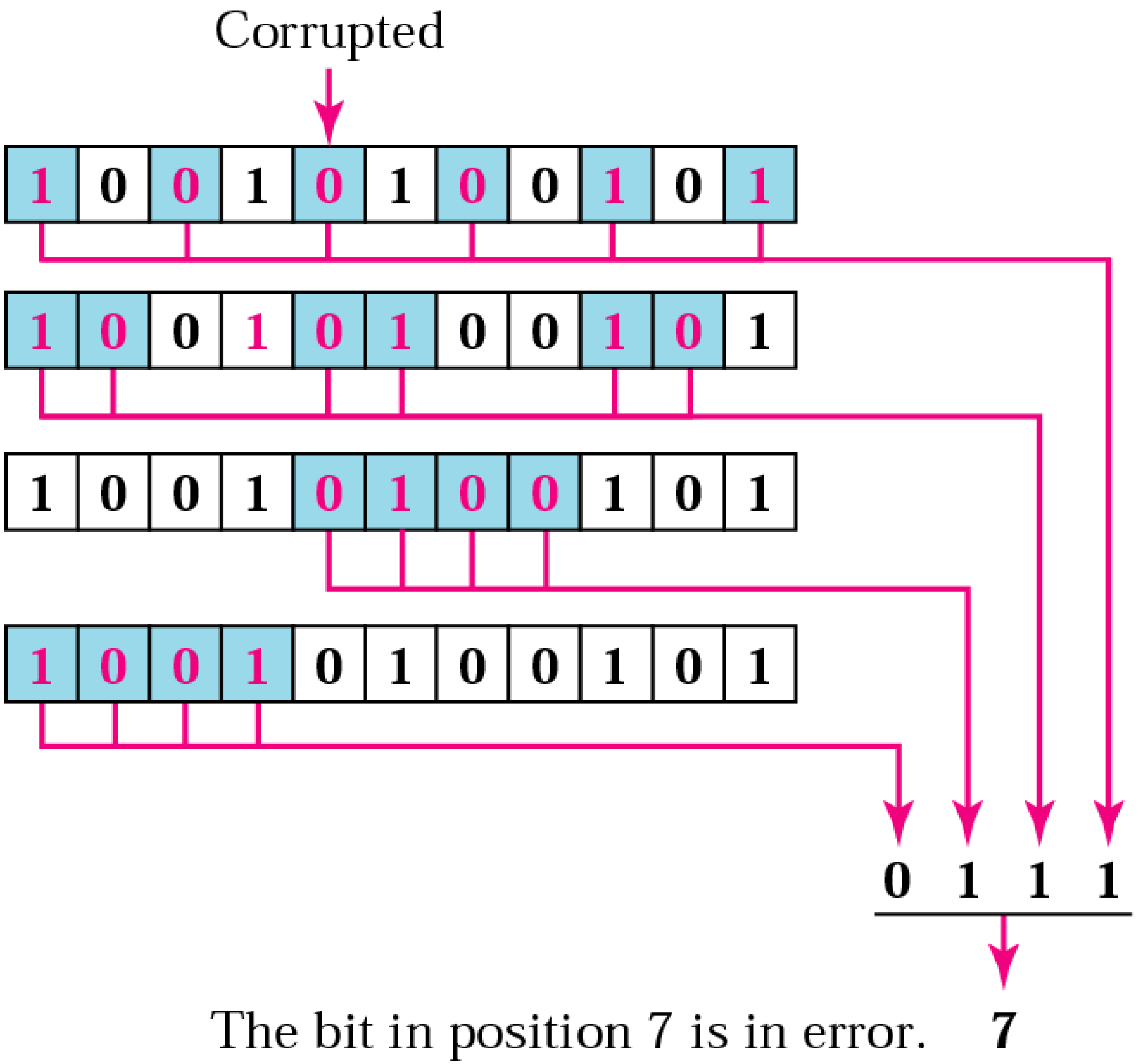
r_1 will take care of these bits.
11 9 7 5 3 1
d d d r_8 d d d r_4 d r_2 r_1
r_2 will take care of these bits.
11 10 7 6 3 2
d d d r_8 d d d r_4 d r_2 r_1
r_4 will take care of these bits.
7 6 5 4
d d d r_8 d d d r_4 d r_2 r_1
r_8 will take care of these bits.
11 10 9 8
d d d r_8 d d d r_4 d r_2 r_1



Data:
1 0 0 1 1 0 1

Code:
1 0 0 1 1 1 0 0 1 0 1

Error detection using Hamming code



Corrupted
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
1 0 0 1 0 1 0 0 1 0 1
0 1 1 1
The bit in position 7 is in error.
7