22AIE314 - COMPUTER SECURITY
UNIT-1
Error Detection & Correction

March 3, 2025

Dr.Remya S
Assistant Professor
Department of Computer Science

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

School of Computing
Amritapuri Campus, Clappana P.O., Kollam - 690525, Kerala, India. Ph: +91 (476) 280 2100 Email: csoffice@am.amrita.edu
www.amrita.edu/school/computing/amritapuri

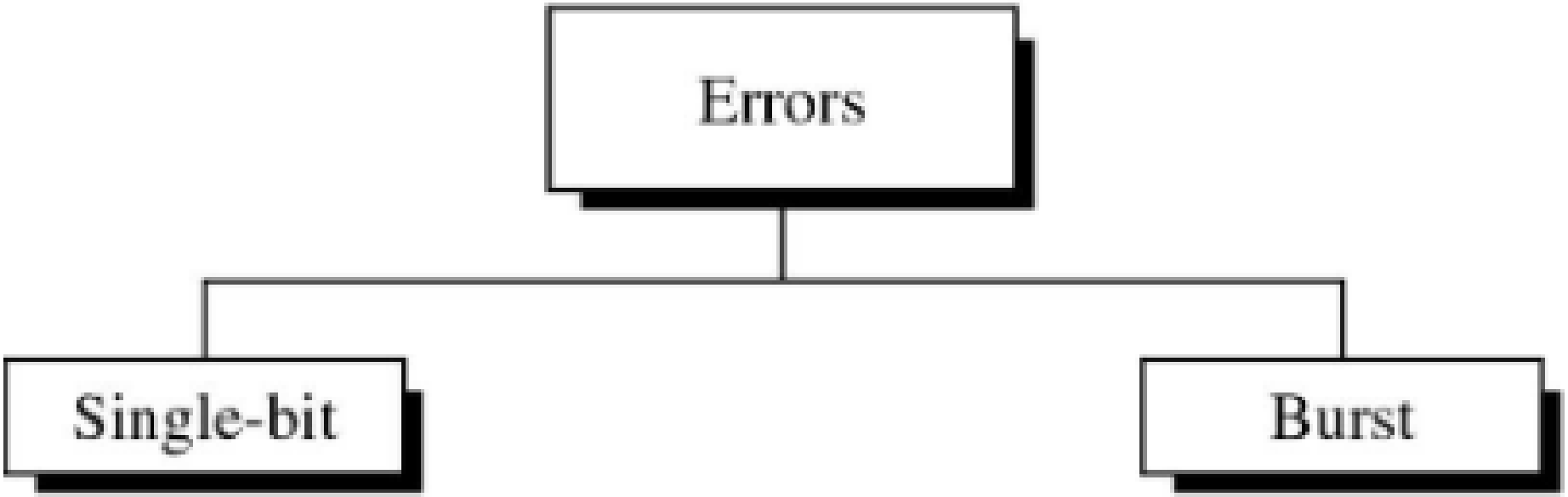
Agenda

- Types of Errors
- Error Detection
- Error Correction

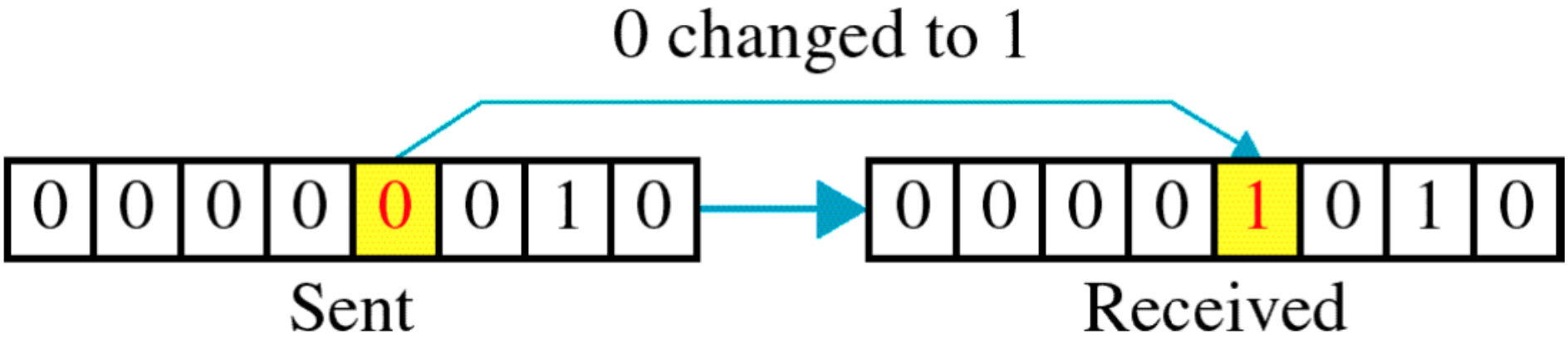
Basic Concepts

1. Networks must be able to transfer data from one device to another with complete accuracy.
2. Data can be corrupted during transmission.
3. For reliable communication, errors must be detected and corrected.
4. Error detection and correction are implemented either at the **data link layer** or the **transport layer** of the OSI model.

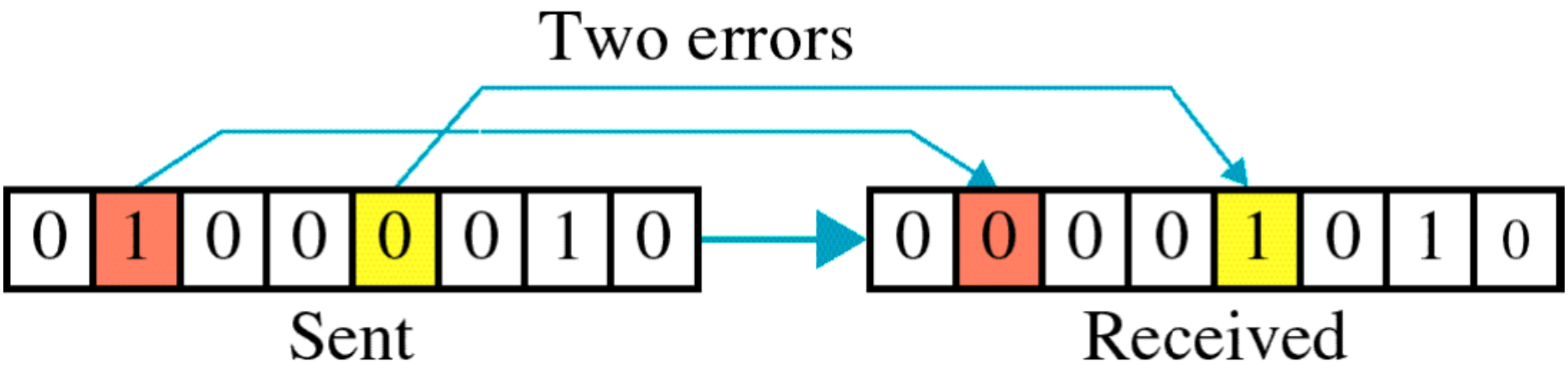
Types of Errors



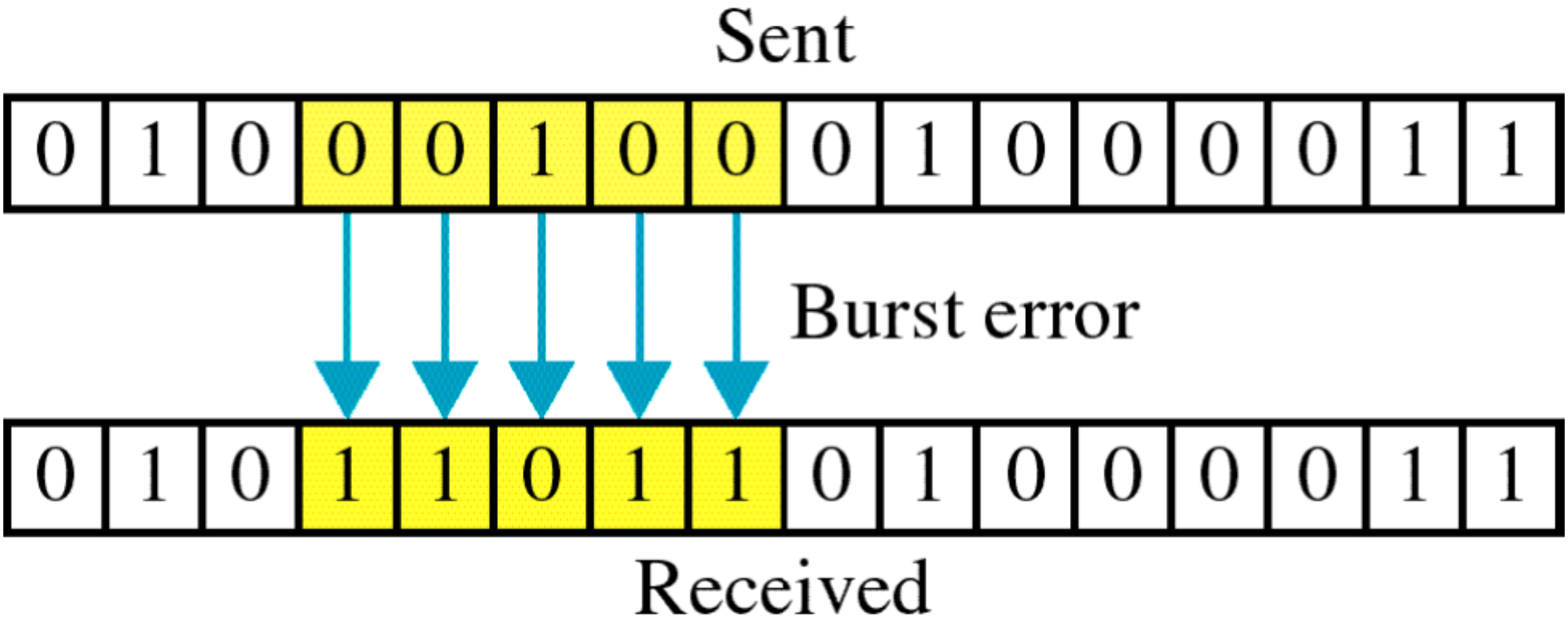
Single-bit error



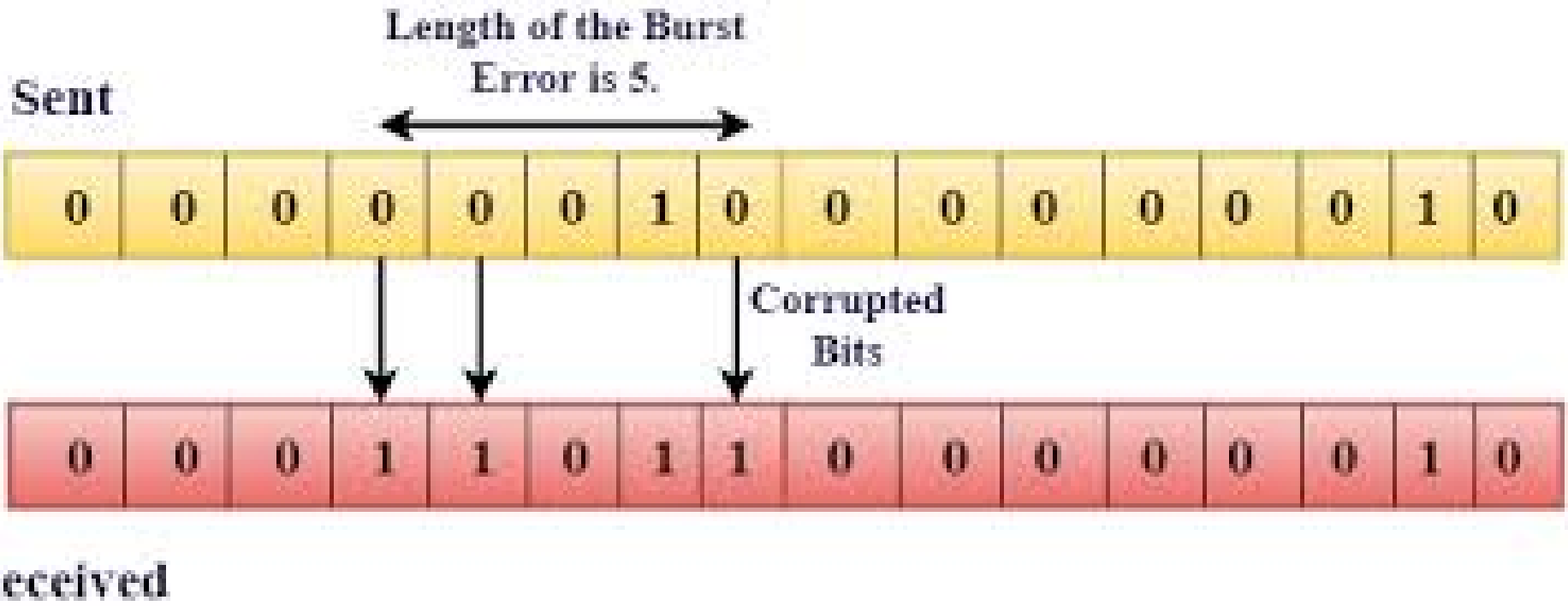
Burst error



Burst error


Sent
0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
Burst error
0 1 0 1 1 0 1 1 0 1 0 0 0 0 1 1
Received

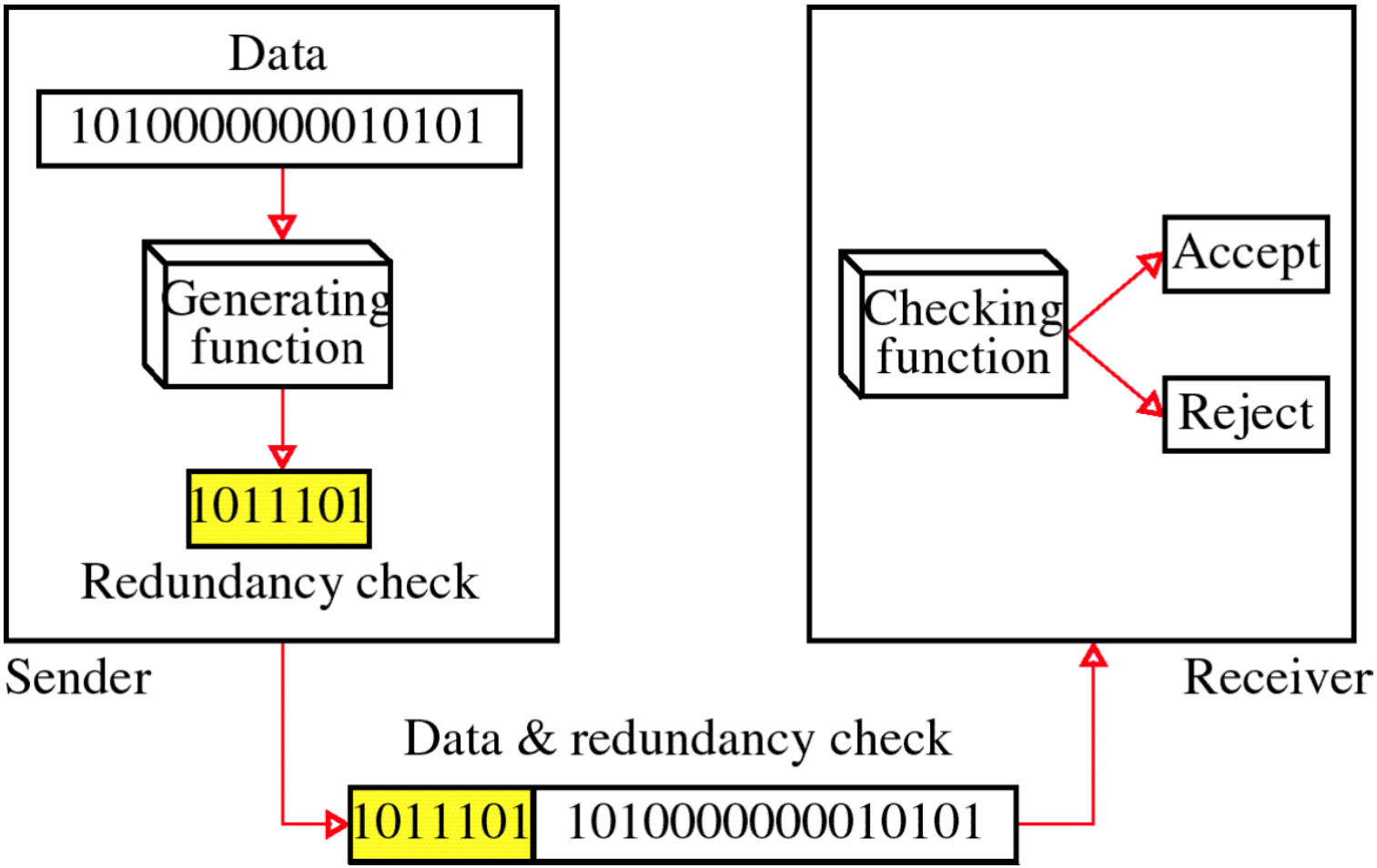
- The term burst error means that two or more bits in the data unit have changed from 1 to 0 or from 0 to 1.
- Burst errors does not necessarily mean that the errors occur in consecutive bits, the length of the burst is measured from the **first corrupted bit to the last corrupted bit**. Some bits in between may not have been corrupted.


Length of the Burst
Error is 5.
Sent
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
Corrupted
Bits
0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0
Received

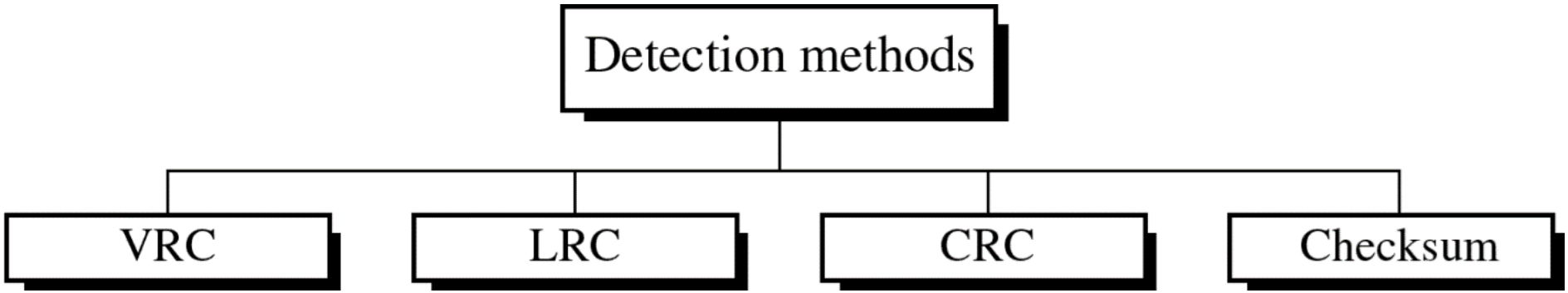
Error detection

- Error detection means to decide whether the received data is correct or not without having a copy of the original message.
- Error detection uses the concept of **redundancy**, which means adding extra bits for detecting errors at the destination

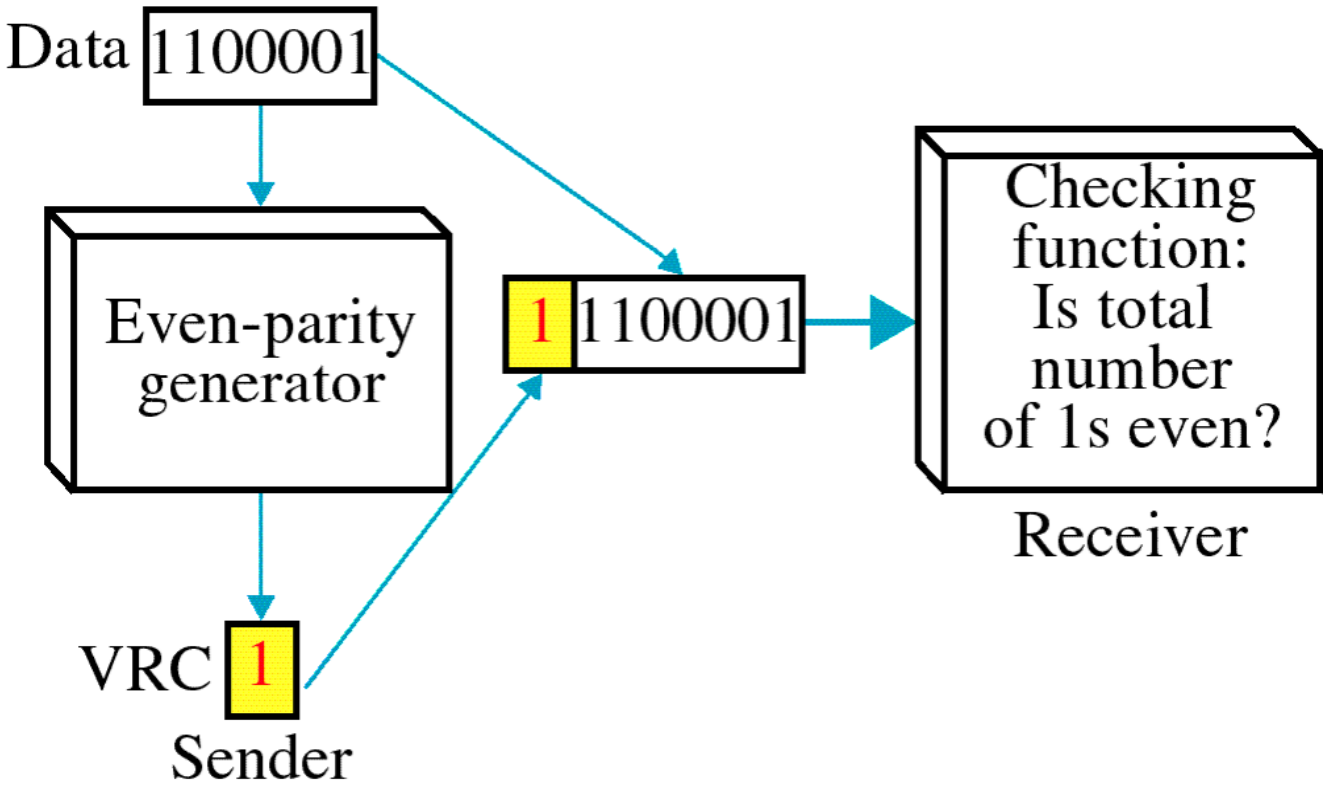
Redundancy


Data
1010000000010101
Generating function
1011101
Redundancy check
Sender
Checking function
Accept
Reject
Receiver
Data & redundancy check
1011101
1010000000010101

Four types of redundancy checks are used in data communications



Vertical Redundancy Check(VRC)-Simple Parity Check



- **1** is added to the block if it contains an odd number of 1's, and
- **0** is added if it contains an even number of 1's

Advantages of Simple Parity Check

- Simple parity check can detect all single bit error/odd number of errors.
- Implementation: Simple Parity Check is easy to implement in both hardware and software.
- Minimal Extra Data: Only one additional bit (the parity bit) is added per data unit (e.g., per byte).
- Fast Error Detection: The process of calculating and checking the parity bit is quick, which allows for rapid error detection without significant delay in data processing or communication.

Disadvantages of Simple Parity Check

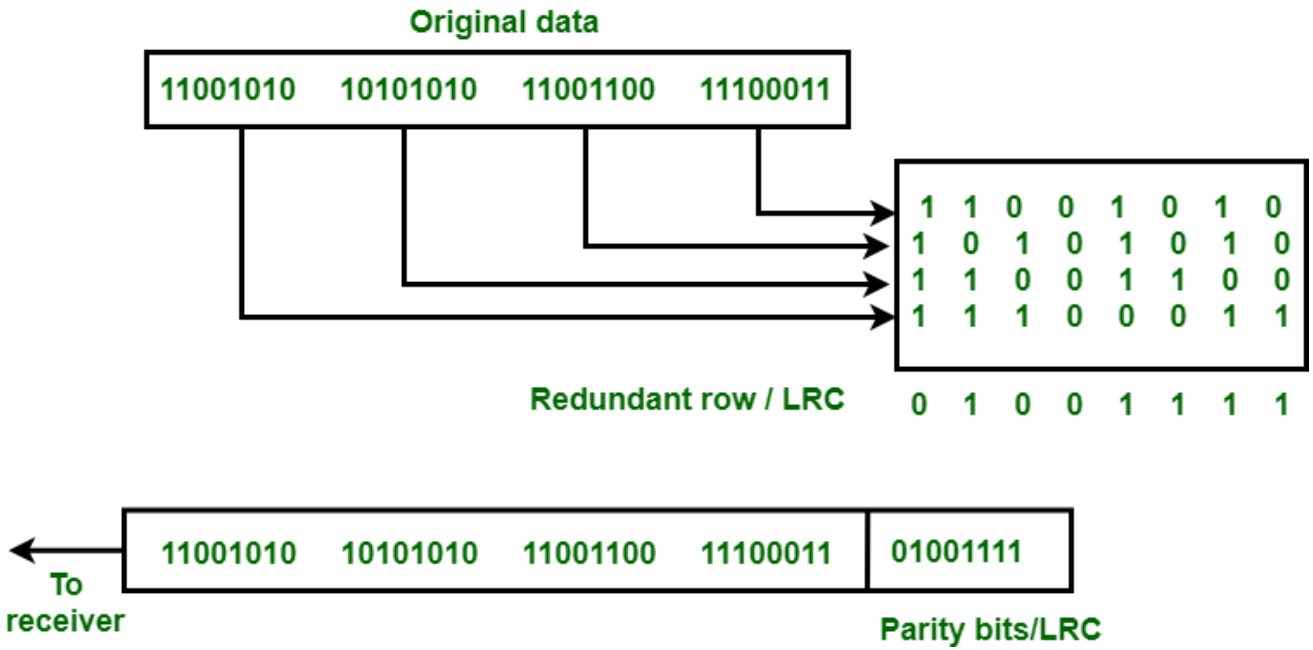
- Single Parity check is not able to detect even no. of bit error.
- For example, the Data to be transmitted is **101010**. Codeword transmitted to the receiver is 101010**1** (we have used even parity).
- Let's assume that during transmission, two of the bits of code word flipped to **1111101**.
  On receiving the code word, the receiver finds the no. of ones to be even and hence no error, which is a **wrong assumption**.

Longitudinal Redundancy Check(LRC)-Two-Dimensional Parity Check

- In this method, data which the user want to send is organised into **tables of rows and columns**.
- A block of bit is divided into table or matrix of rows and columns.
- In order to detect an error, a **redundant bit** is added to the whole block and this block is transmitted to receiver.
- The receiver uses this redundant row to detect error. After checking the data for errors, receiver accepts the data and discards the redundant row of bits.

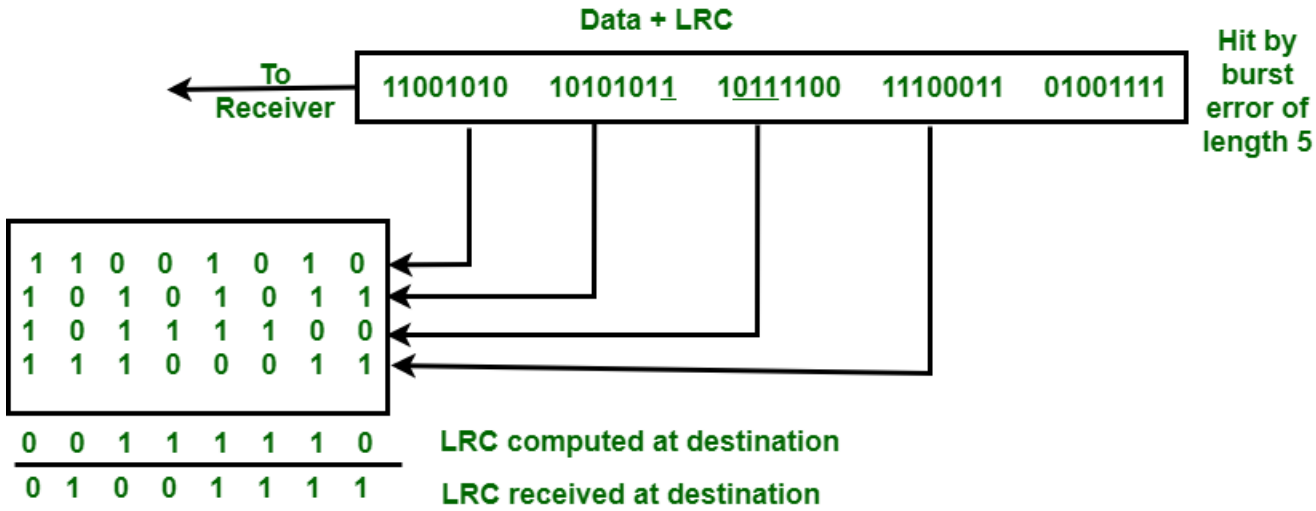
Example 1

- If a block of 32 bits is to be transmitted, it is divided into matrix of four rows and eight columns
- In this matrix of bits, a parity bit (odd or even) is calculated for each column. It means 32 bits data plus 8 redundant bits are transmitted to receiver. Whenever data reaches at the destination, receiver uses LRC to detect error in data

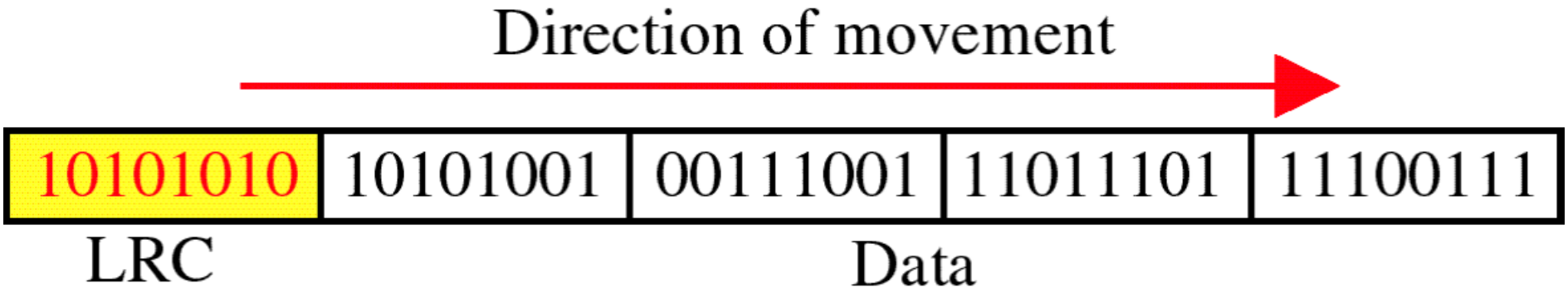


At Receiver side

- Suppose 32 bit data plus LRC that was being transmitted is hit by a burst error of length 5 and some bits are corrupted
- The LRC received by the destination does not match with newly corrupted LRC. The destination comes to know that the data is erroneous, so it discards the data.

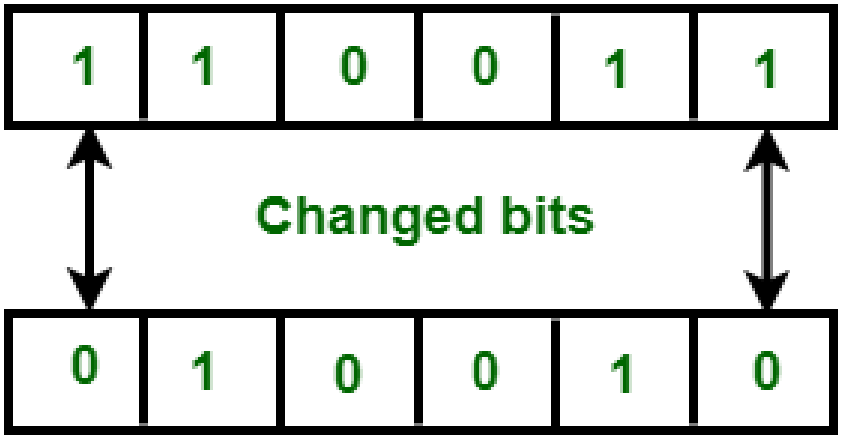


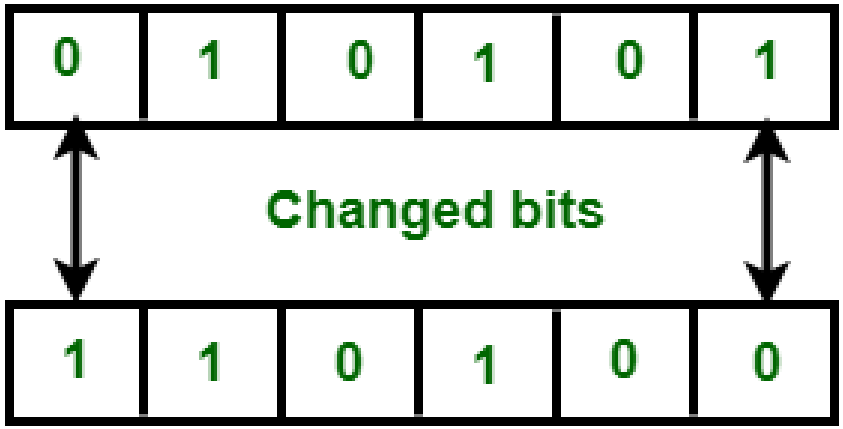
Example 2


Direction of movement
10101010
10101001
00111001
11011101
11100111
LRC
Data

Performance

- Advantage : LRC is used to detect burst errors.
- Disadvantage : The main problem with LRC is that, it is not able to detect error two bits in exactly the same position in other data unit is damaged.





Checksum

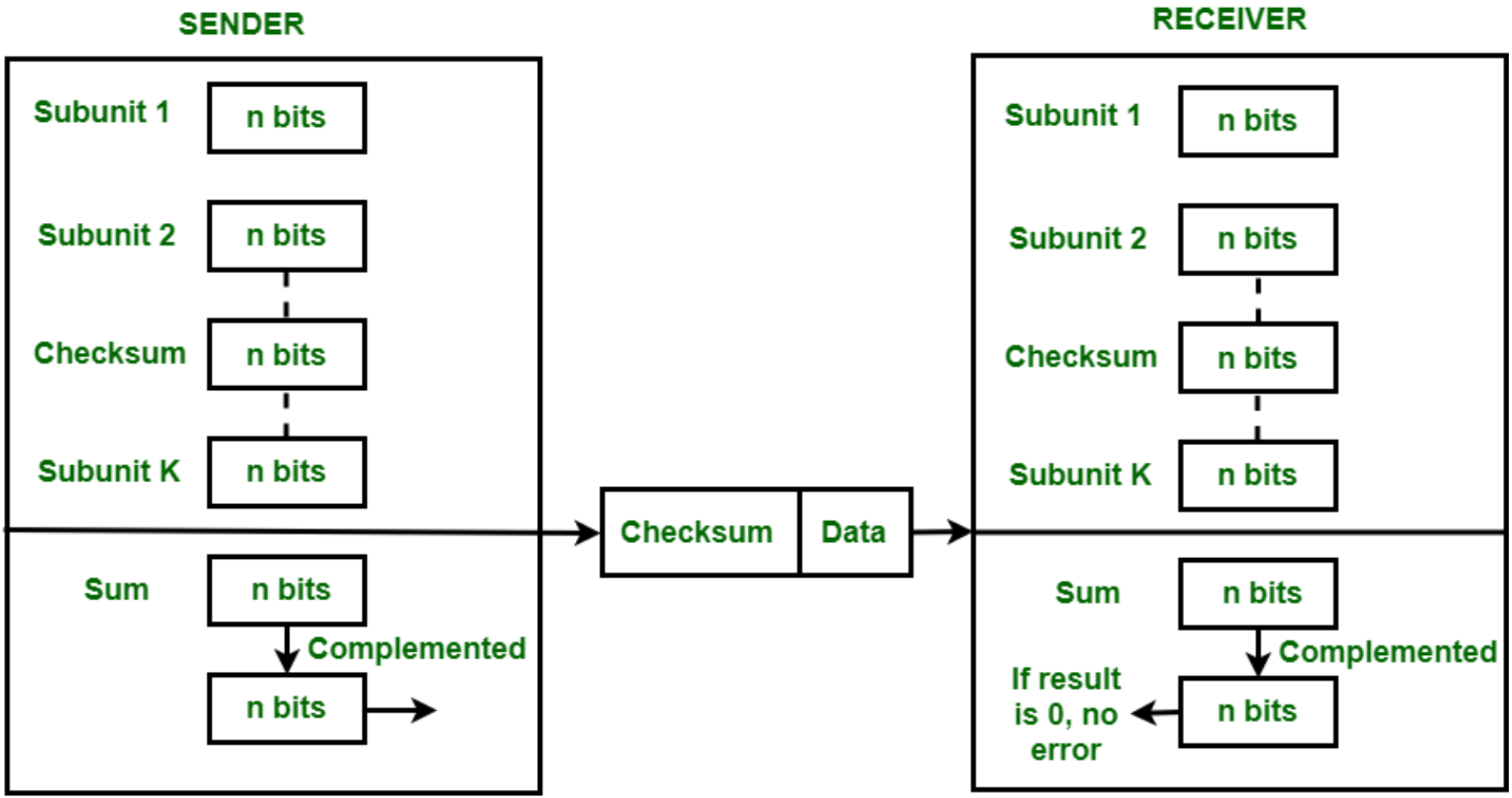
- Checksum error detection is a method used to identify errors in transmitted data.
- The process involves dividing the data into **equally sized segments** and using a **1's complement** to calculate the sum of these segments. The calculated sum is then sent along with the data to the receiver.
- At the receiver's end, the same process is repeated and if all zeroes are obtained in the sum, it means that the data is correct.
- uses a Checksum Generator on the sender side and a Checksum Checker on the receiver side.

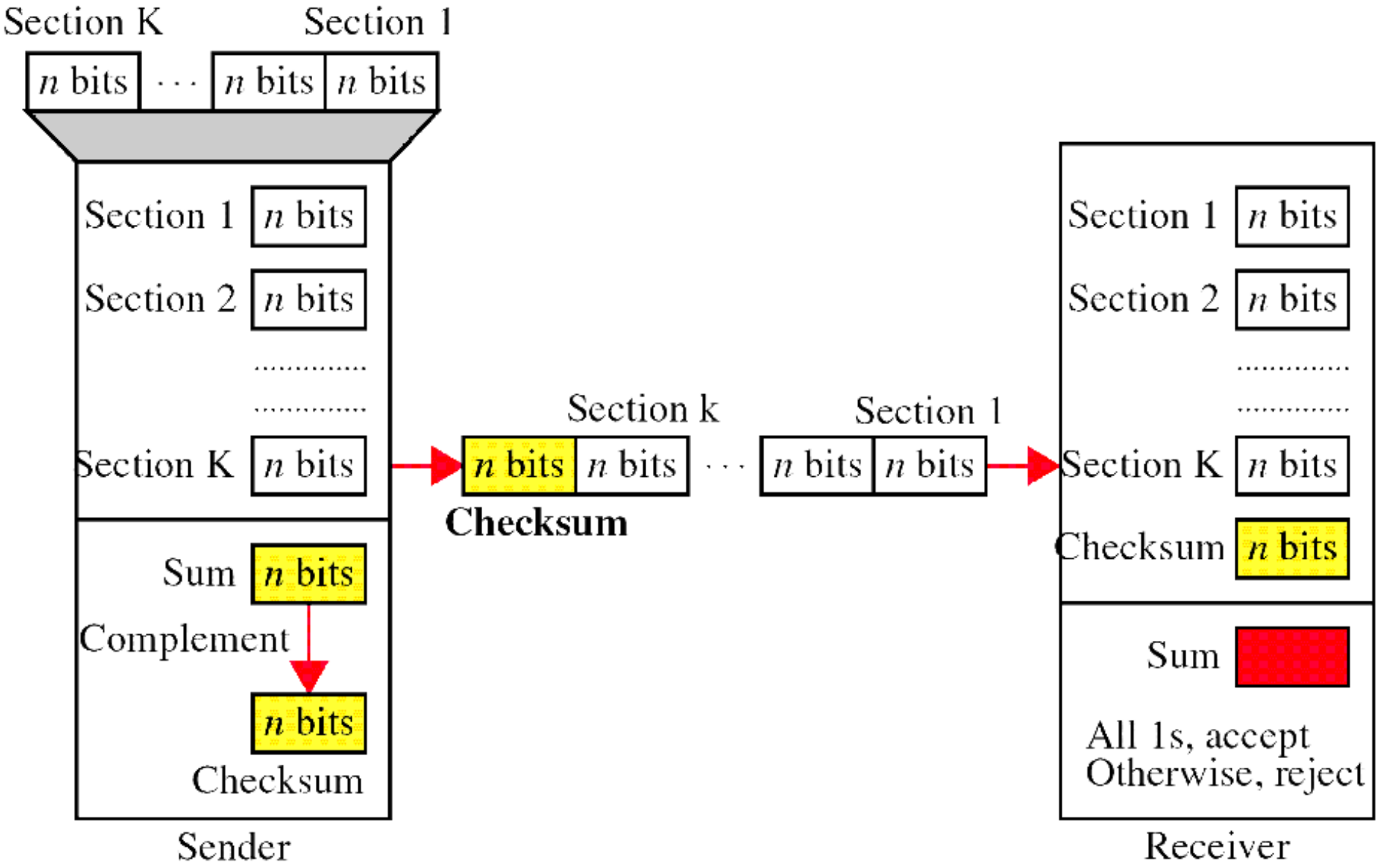
Checksum – Operation at Sender's Side

- Firstly, the data is divided into **k segments** each of **n bits**.
- On 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.

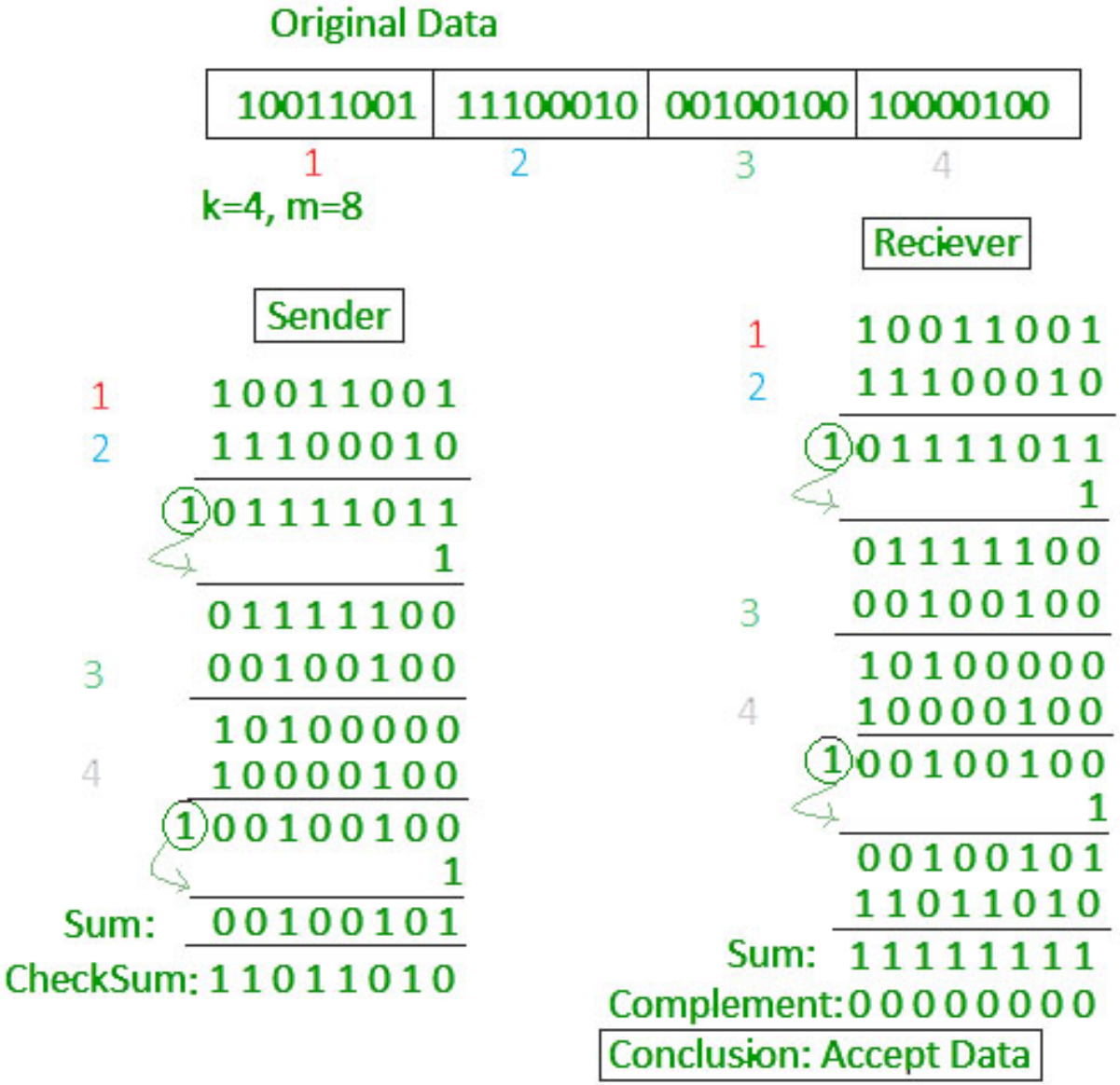
Checksum – Operation at Receiver's Side

- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum.
- The sum is complemented.
- **If the result is zero, the received data is accepted; otherwise discarded**.


SENDER
Subunit 1
n bits
Subunit 2
n bits
Checksum
n bits
Subunit K
n bits
Sum
n bits
Complemented
n bits
Checksum
Data
RECEIVER
Subunit 1
n bits
Subunit 2
n bits
Checksum
n bits
Subunit K
n bits
Sum
n bits
Complemented
If result is 0, no error
n bits


Section K
Section 1
n bits
n bits
n bits
Section 1
n bits
Section 2
n bits
Section K
n bits
Sum
n bits
Complement
n bits
Checksum
Sender
Section k
Section 1
n bits
n bits
n bits
n bits
Checksum
Section 1
n bits
Section 2
n bits
Section K
n bits
Checksum
n bits
Sum
All 1s, accept
Otherwise, reject
Receiver

Example

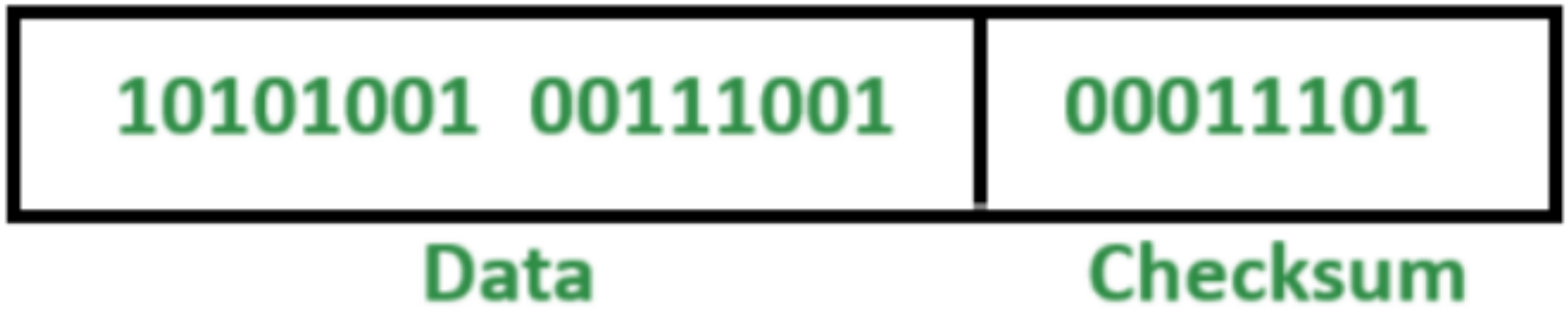

Original Data
10011001 11100010 00100100 10000100
1 2 3 4
k=4, m=8
Sender
1 10011001
2 11100010
01111011
1
01111100
3 00100100
10100000
4 10000100
00100100
1
Sum: 00100101
CheckSum: 11011010
Reciever
1 10011001
2 11100010
01111011
1
01111100
3 00100100
10100000
4 10000100
00100100
1
00100101
11011010
Sum: 11111111
Complement: 00000000
Conclusion: Accept Data

Example – If the data unit to be transmitted is 10101001 00111001, the following procedure is used at Sender site and Receiver site.

Sender Site:

```
10101001        subunit 1
00111001        subunit 2
11100010        sum (using 1s complement)
00011101        checksum (complement of sum)
```

Data transmitted to Receiver is:



Receiver Site:

```
10101001          subunit 1
00111001          subunit 2
00011101          checksum
11111111               sum
00000000     sum's complement

Result is zero, it means no error.
```

Example – If the data transmitted along with checksum is 10101001 00111001 00011101. But the data received at destination is 00101001 10111001 00011101.

Receiver Site:

```
00101001              1st bit of subunit 1 is damaged
10111001               1st bit of subunit 2 is damaged
00011101              checksum
11111111                  sum
00000000         Ok 1's complement
```

Although data is corrupted, the error is undetected.

Cyclic Redundancy Check(CRC)

- CRC is based on **Modulo-2 division(XOR operation)**.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of the data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

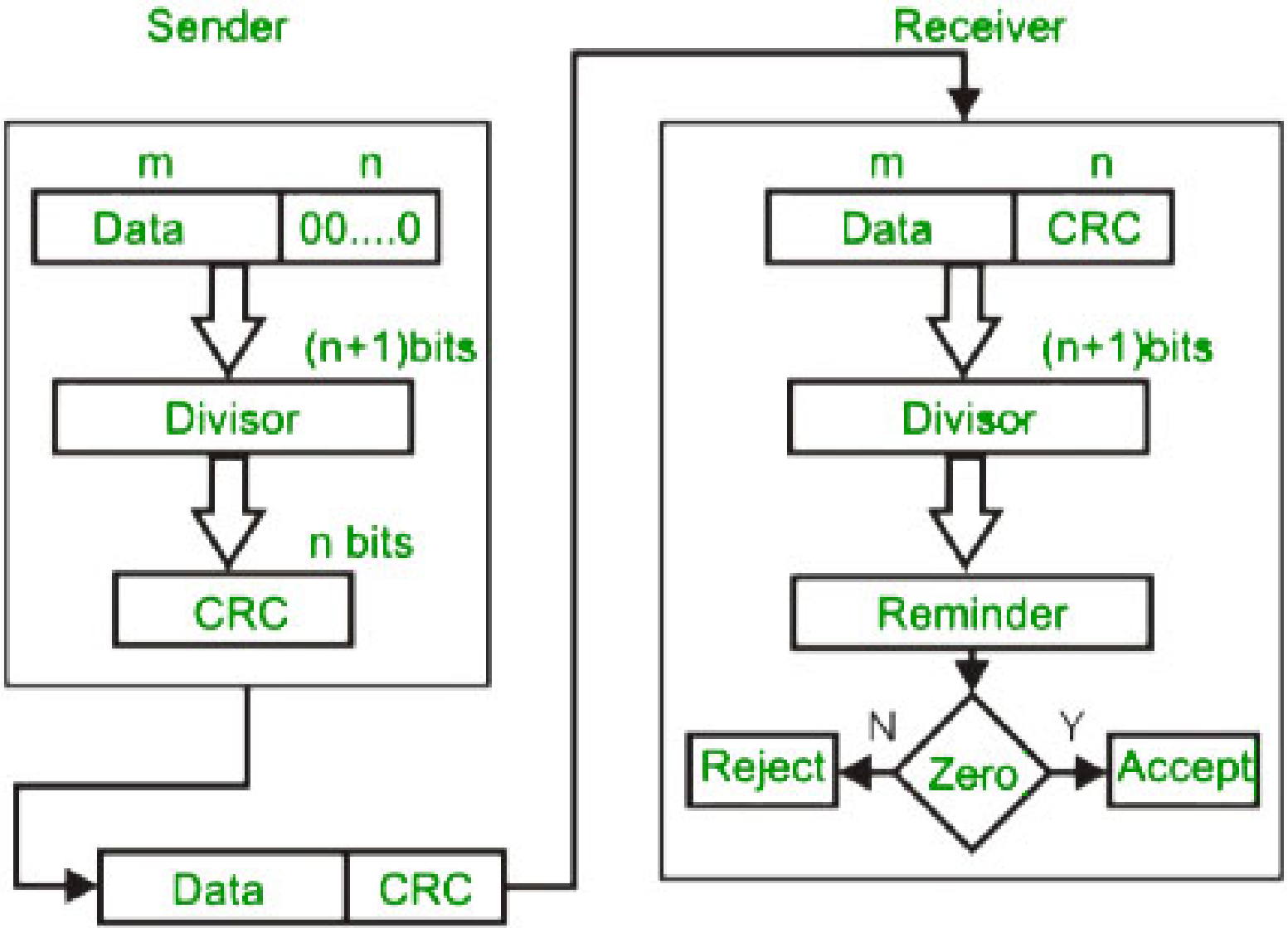
CRC Working

We have given dataword of length n and divisor of length k.

1. Append (k-1) zero's to the original message
2. Perform modulo 2 division
3. Remainder of division = CRC
4. Code word = Data with append k-1 zero's + CRC

Note:

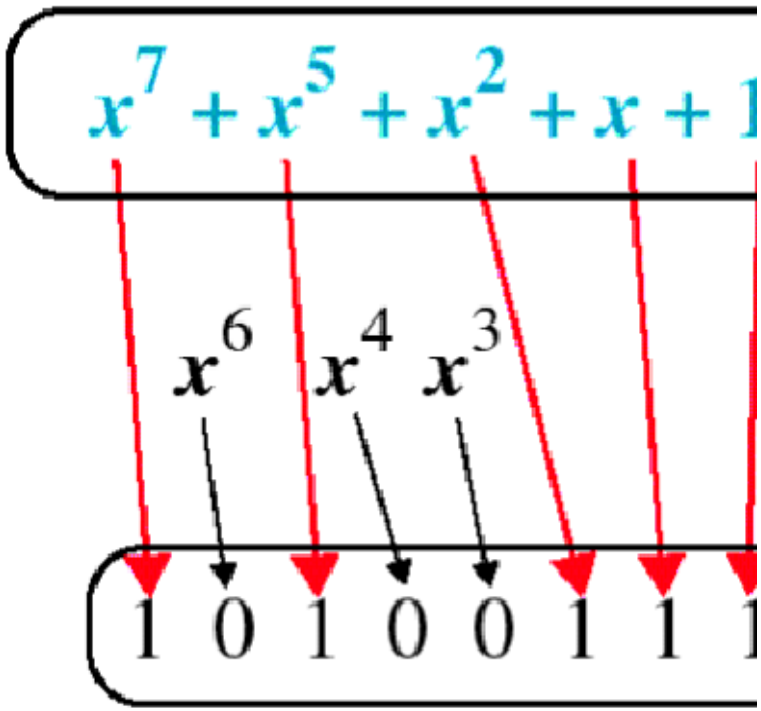
- CRC must be k-1 bits
- Length of Code word = n+k-1 bits


Sender
Receiver
m
n
Data
00....0
(n+1)bits
Divisor
n bits
CRC
Data
CRC
m
n
Data
CRC
(n+1)bits
Divisor
Reminder
N
Y
Reject
Zero
Accept

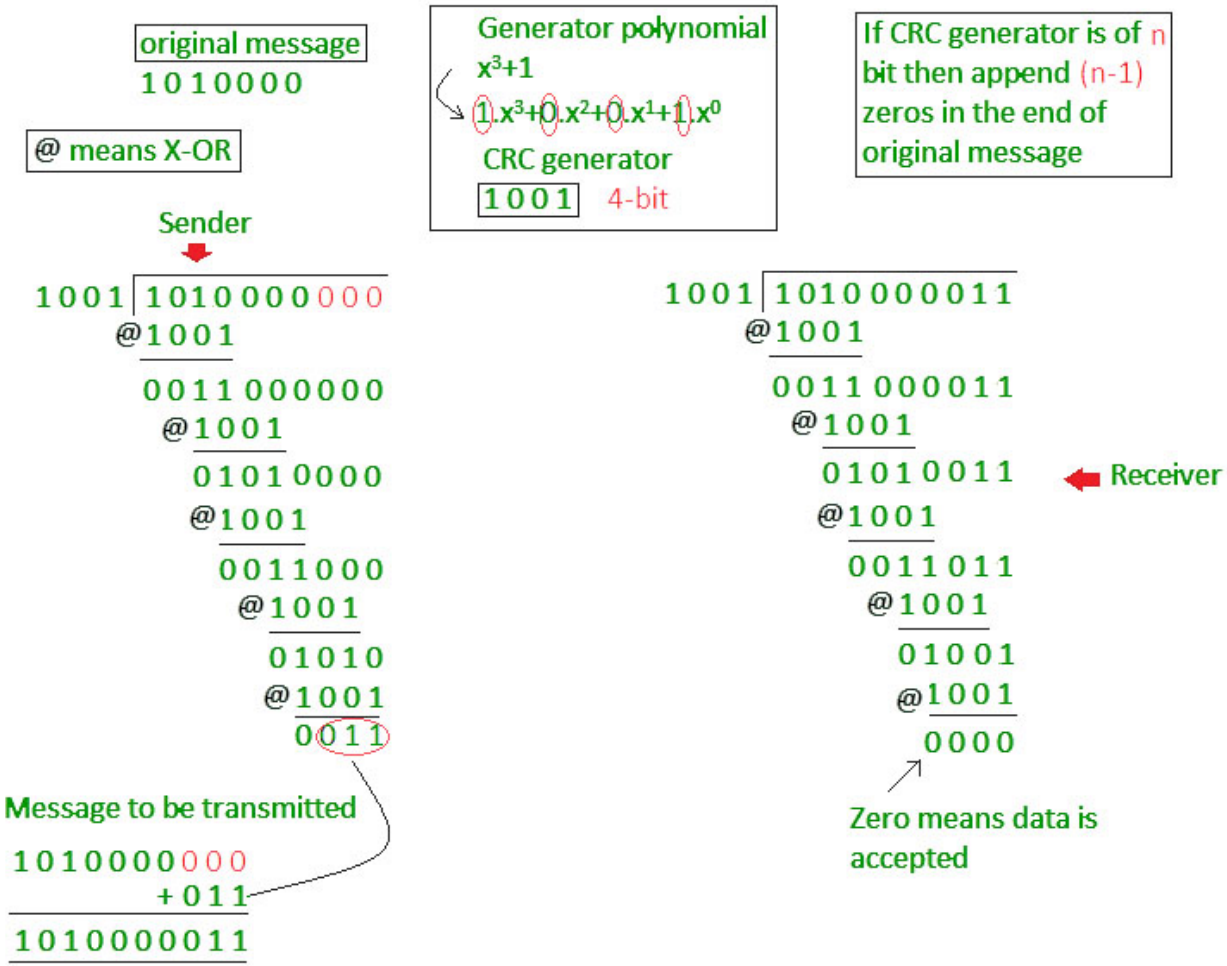
Polynomial Representaion

$$x^7 + x^5 + x^2 + x + 1$$

Polynomial



Example: Let's data to be send is 1010000 and divisor in the form of polynomial is x^3+1.



CRC	Polynomial	Notation	Application
CRC-4	X^4+X+1	0X3	ITU-T G.704
CRC-8	X^8+X^2+X+1	0X07	ATM HEC, ISDN HEC
CRC-12	$X^{12}+X^{11}+X^3+X^2+X+1$	0X80F	telecom systems
CRC-16	$X^{16}+X^{15}+X^2+1$	0X8005	Bisync, Modbus, USB, ANSI X3.28, SIA DC-07, many others
CRC-CCITT	$X^{16}+X^{12}+X^5+1$	0X1021	X.25, V.41, HDLC FCS, XMODEM, Bluetooth, PACTOR, many others
CRC-32	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$	0x04C11DB7	HDLC, IEEE 802.3 (Ethernet), SATA, ZIP, many others