Data Link Layer

- The **data link layer** transforms the **physical layer**, a raw transmission facility, to a link responsible for **node-to-node communication**.
- Data link layer include **framing**, **addressing**, **flow control**, **error control**, and **media access control**.
- The data link layer **divides the stream of bits** received from the network layer into manageable data units called **frames**.
- The data link layer **adds a header to the frame** to define the addresses of the sender and receiver of the frame.
- The data link layer also **adds reliability to the physical layer** by adding mechanisms to detect and retransmit damaged, duplicate, or lost frames.
- Hence, it **makes the physical layer error-free** to the upper layer (Network layer).

Data Link Layer

- **Framing:** The data link layer **divides the stream of bits** received from the network layer into manageable data units called frames.

- **Physical addressing:** If frames are to be distributed to different systems on the network, it **adds a header** to the frame to define the sender and/or receiver of the frame.

- **Flow control:** If the rate at which the data are absorbed by the receiver is less than the rate at which data are produced in the sender, the data link layer **imposes a flow control mechanism** to avoid overwhelming the receiver.

- **Error control:** The data link layer **adds reliability to the physical layer** by adding mechanisms to detect and retransmit damaged or lost frames. It also uses a mechanism to recognize duplicate frames. Error control is normally achieved through a **trailer added to the end of the frame**.

- **Access control:** When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

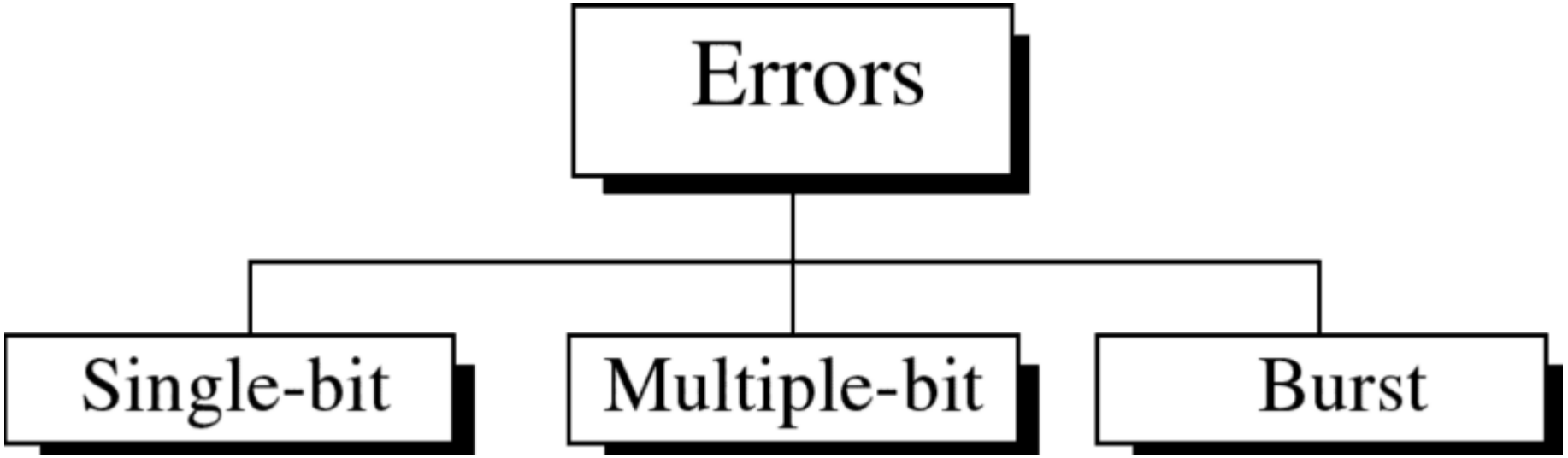
Outline of the Lecture

- Why error detection and correction?
- Types of Error
- Error Detection Techniques
  - Parity Check
  - Two-Dimensional Parity Check
  - Checksum
  - Cyclic Redundancy Check
- Error Correcting Codes

Why Error Detection and Correction

- Because of **attenuation**, **distortion**, **noise**, and **interfaces**, error during transmission are inevitable, leading to **corruption of transmission bits**
- **Longer the frame size** and higher the probability of single bit error, lower is the probability of receiving the frame without error
- **This clearly emphasizes the need for error detection and error correction**

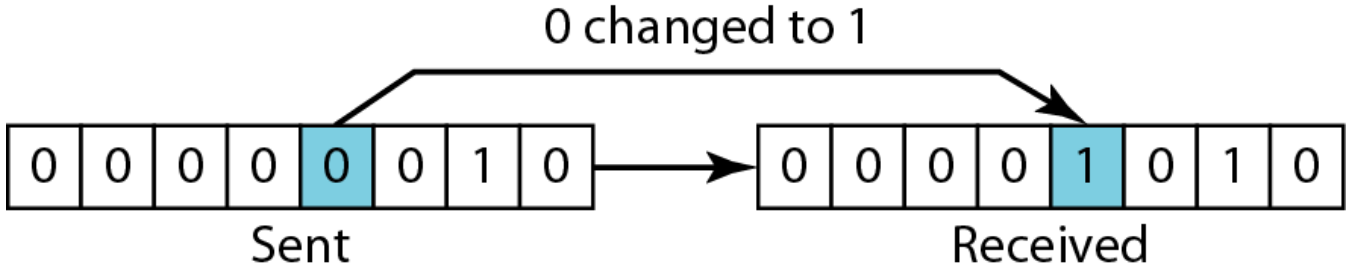
Types of Error



Types of Error

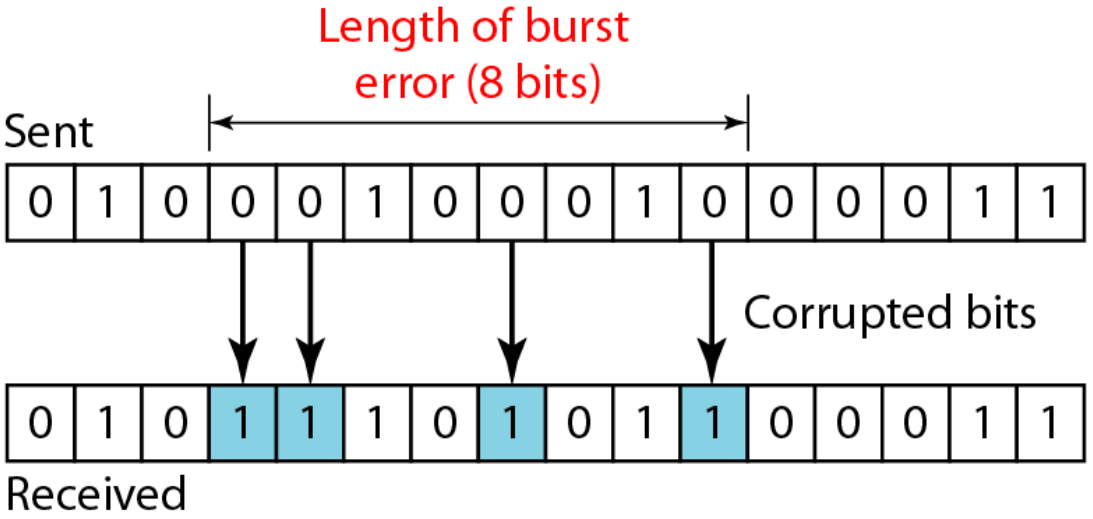
In a single-bit error, only 1 bit in the data unit has changed. Common in **parallel transmission**.

Single-bit error



- A burst error means that 2 or more bits in the data unit have changed or corrupted.
- Very common in **serial transmission of data.**
- Occurs when **the duration of the noise is longer** than the duration of one bit.

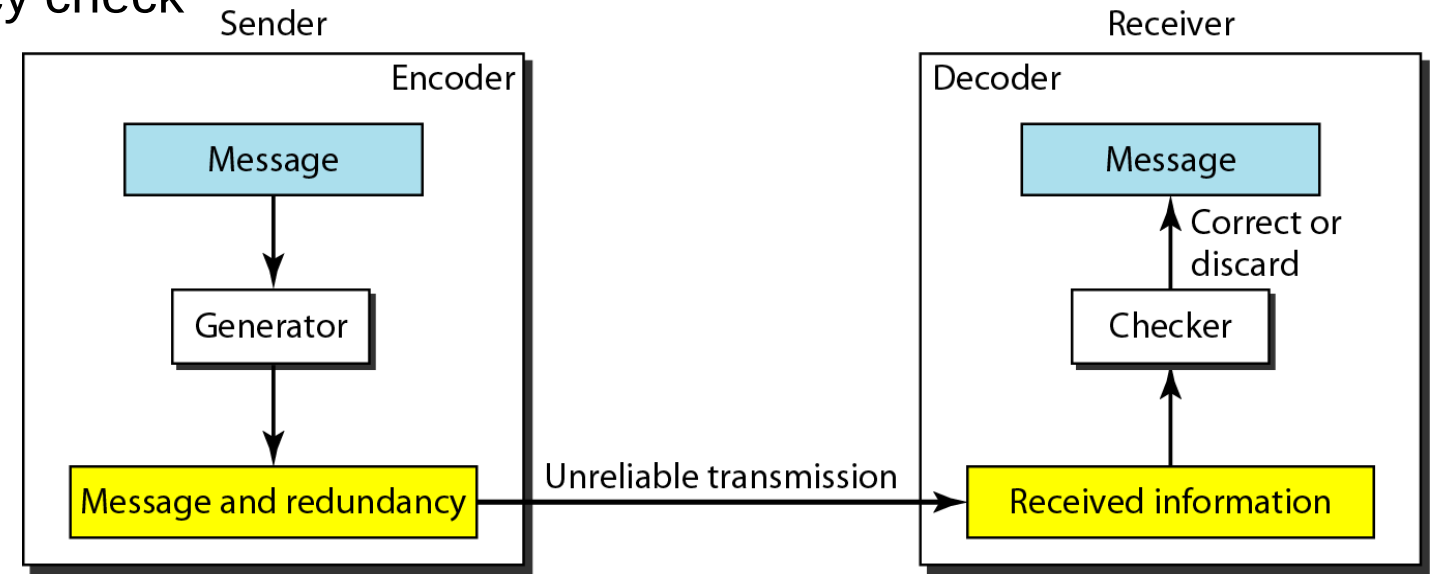
Burst error of length 8



Error Detection Techniques

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

- **Use of Redundancy:** Additional bits are added to facilitate detection and correction of errors.
- **Popular Techniques:**
  - Simple Parity Check
  - Two-dimensional Parity check
  - Checksum
  - Cyclic Redundancy check



The structure of encoder and decoder

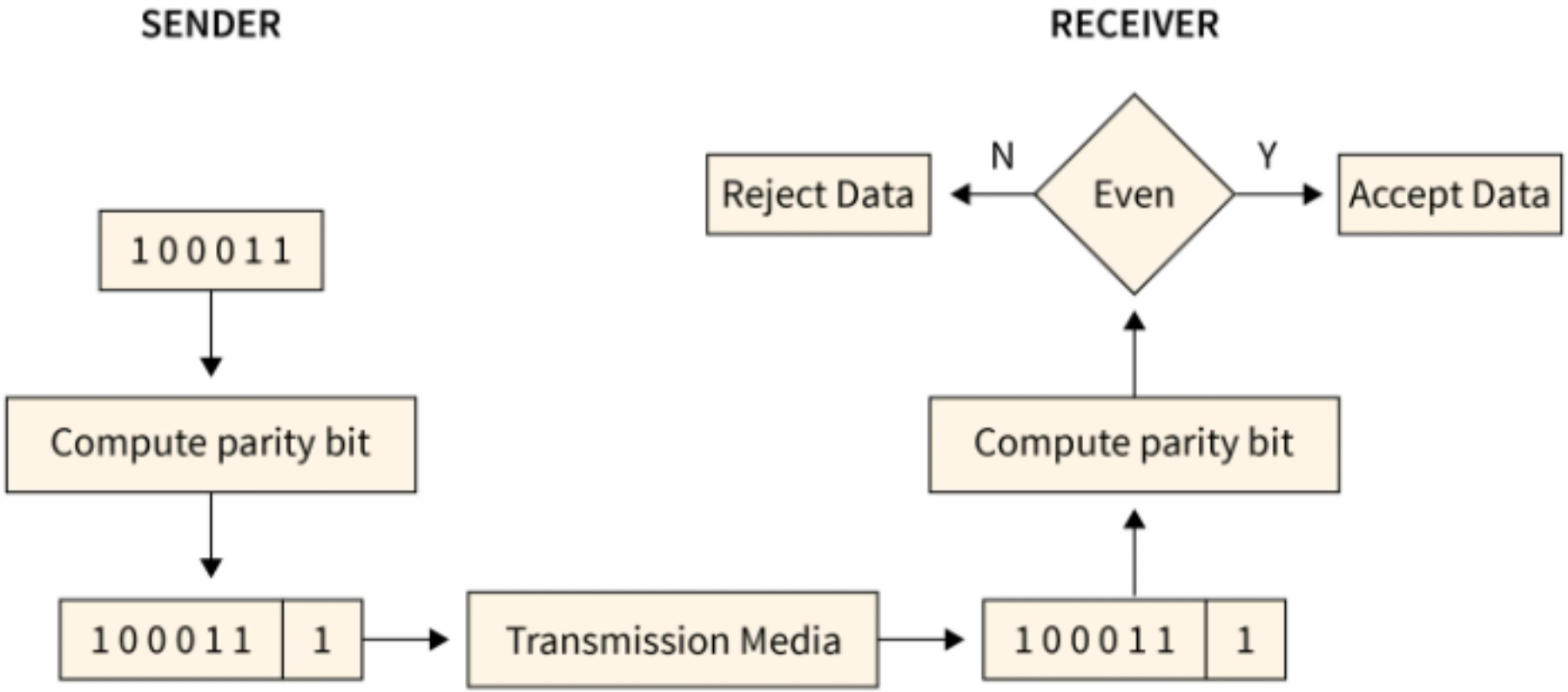
Simple Parity Check

- The simplest and the most popular error detection scheme.
- Appends a parity bit to the end of the data.
- A **simple parity-check code** is a **single-bit error-detecting code** in which $n = k + 1$.
- **Even parity** (ensures that a codeword has an even number of 1's) and **odd parity** (ensures that there are an odd number of 1's in the codeword)

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

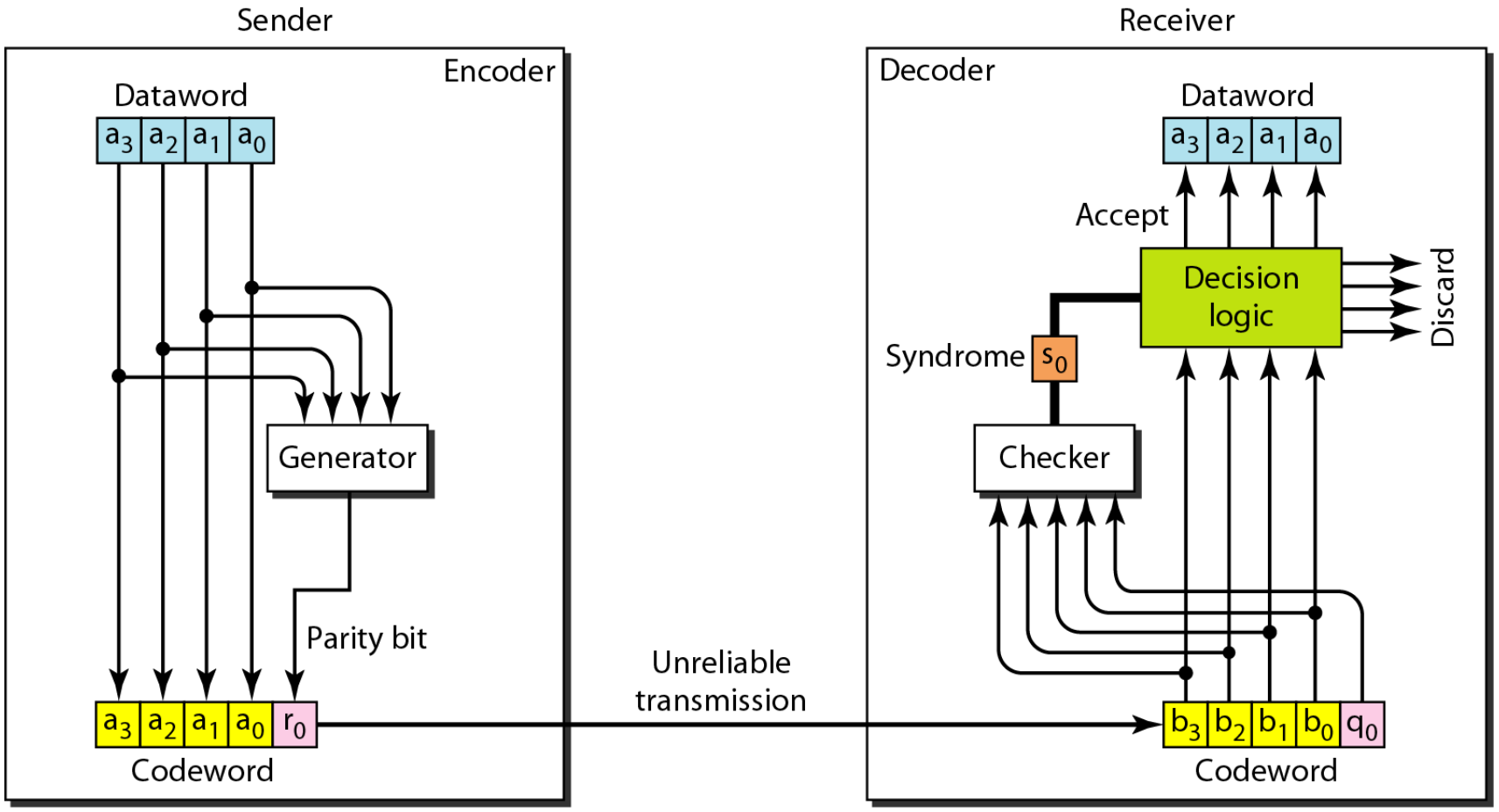
Parity Check



Disadvantage:

- **Only single-bit error is detected** by this method, it fails in multi-bit error detection.
- It can not detect an error in case of an error in two bits.

Simple Parity Check



Encoder and decoder for simple parity-check code

Simple Parity Check

- If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1.
- In both cases, the total number of 1s in the codeword is even.
- The sender sends the codeword which may be corrupted during transmission.
- The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator.
- The only difference is that the addition is done over all 5 bits.
- The result, which is called the syndrome, is just 1 bit.
- **The syndrome is 0** when the **number of 1s in the received codeword is even**; otherwise, it is 1.
- **If the syndrome is 0**, there is **no error** in the received codeword; the data portion of the received codeword is accepted as the dataword.

Simple Parity Check

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 a1 . The received codeword is 10011. The syndrome is 1. No dataword is created.*
3) *One single-bit error changes r0 . The received codeword is 10110. The syndrome is 1. No dataword is created.*
4) *An error changes r0 and a second error changes a3 .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—a3, a2, and a1—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.*

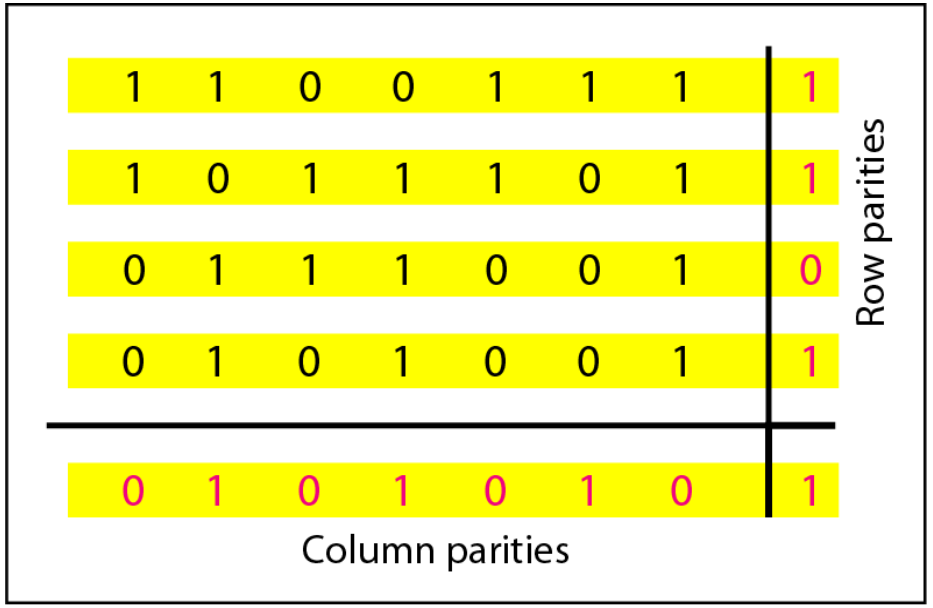
Two-Dimensional Parity Check

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

- A **better approach** is the two-dimensional parity check.
- In this method, the **dataword is organized in a table** (rows and columns).
- The data to be sent, four 7-bit codes are put in separate rows.
- For each row and each column, 1 parity-check bit is calculated.
- The whole table is then sent to the receiver, which finds the syndrome for each row and each column.
- The two-dimensional parity check can detect up to three errors that occur anywhere in the table (arrows point to the locations of the created nonzero syndromes). However, errors affecting 4 bits may not be detected.

Two-Dimensional Parity-Check

Original Data				
	1100111	1011101	0111001	0101001



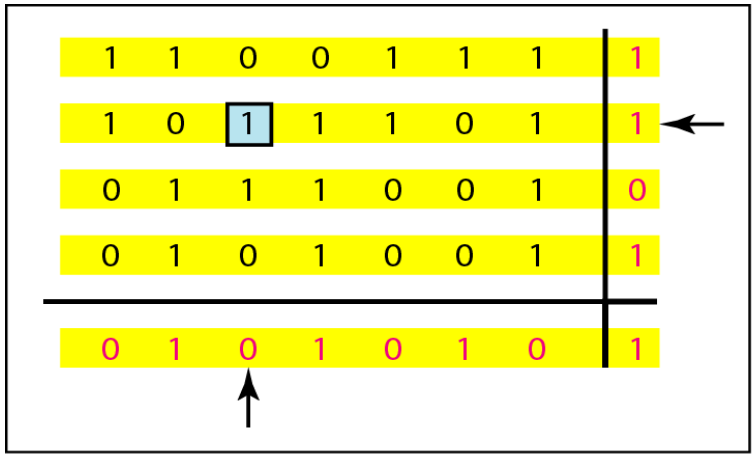
a. Design of row and column parities

Two-dimensional parity-check code

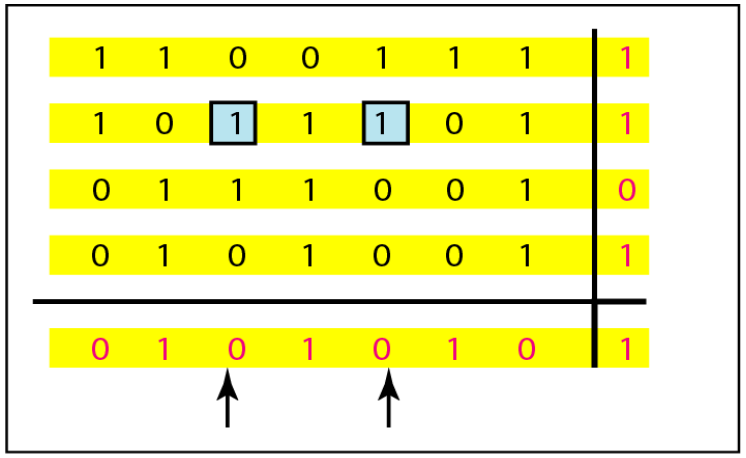
1100111	1011101	0111001	0101001	01010101

Data to be sent

Virtual-Circuit Networks

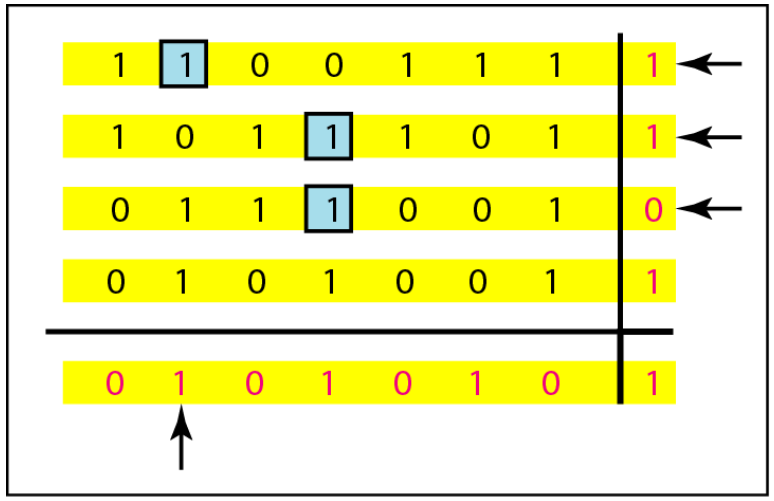


b. One error affects two parities

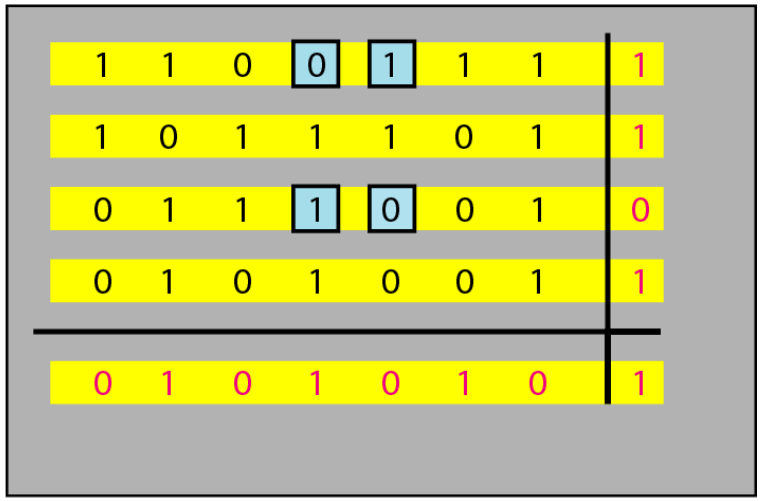


c. Two errors affect two parities

Two-dimensional parity-check code



d. Three errors affect four parities



e. Four errors cannot be detected

Two-dimensional parity-check code

Two-Dimensional Parity Check

Disadvantages:

- **Extra overhead** is traded for better error detection capability
- Two-dimensional parity check significantly improves error detection capability compared to simple parity check.
- **It can detect many burst errors, but not all.**
- If 2 bits are corrupted in 1 data unit and another data unit exactly at the same position is corrupted then this method is not able to detect the error.
- Sometimes this method is not used for **detecting 4-bit **errors or more than 4-bit errors.

Checksum

The Sender's End:

- The data is divided into K segments each of m bits.
- The segments are added using ones complement arithmetic to get the sum.
- The sun is complemented to get the checksum.
- The checksum segment is sent along with the data segment.

The Receiver's End:

- All received data segments are added using ones complement arithmetic to get the sum
- The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded

Checksum

Original Data

10011001	11100010	00100100	10000100
1	*2*	*3*	*4*

k=4, m=8

Sender

1	10011001	
2	11100010	
	101111011	
	↳	1
	01111100	
3	00100100	
	10100000	
4	10000100	
	100100100	
	↳	1
Sum:	00100101	
CheckSum:	11011010	

Receiver

1	10011001	
2	11100010	
	101111011	
	↳	1
	01111100	
3	00100100	
	10100000	
4	10000100	
	100100100	
	↳	1
	00100101	
	11011010	
Sum:	11111111	
Complement:	00000000	

Conclusion : Accept Data

Checksum

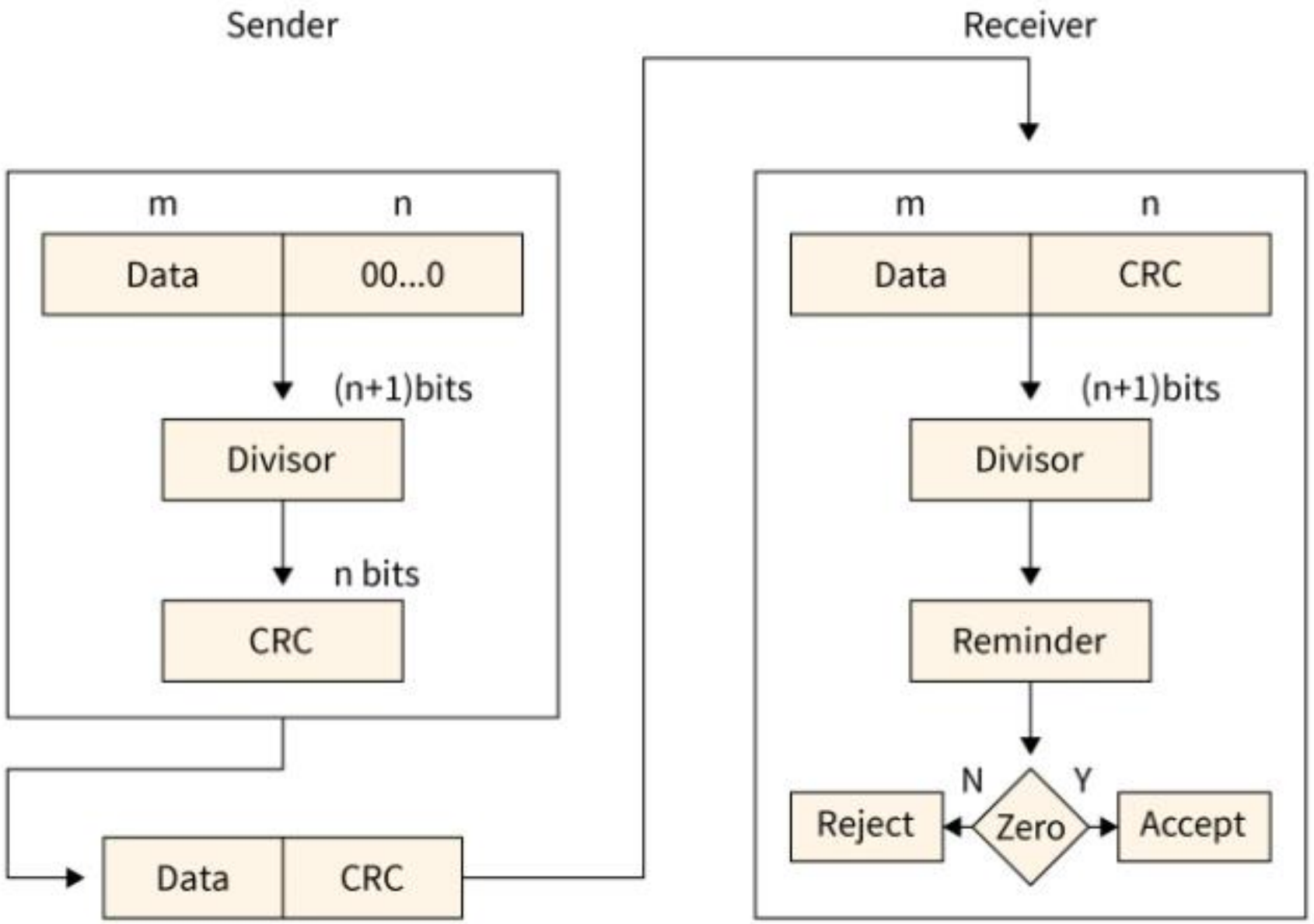
Disadvantages:

- In checksum error is not detected, if one sub-unit of the data has one or more corrupted bits and corresponding bits of the opposite value are also corrupted in another sub-unit.
- Error is not detected in this situation because in this case the sum of columns is not affected by corrupted bits.

Cyclic Redundancy Check (CRC)

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

Cyclic Redundancy Check (CRC)



Cyclic Redundancy Check (CRC)

- The checksum scheme uses the addition method but CRC uses binary division.
- A bit sequence commonly known as cyclic redundancy check is added to the end of the bits in CRC. This is done so that the resulting data unit will be divisible by the second binary number that is predetermined.
- The receiving data units on the receiver's side need to be divided by the same number. These data units are accepted and found to be correct only on the condition that the remainder of this division is zero. The remainder shows that the data is not correct. So, they need to be discarded.

Cyclic Redundancy Check (CRC)

original message

1 0 1 0 0 0 0

@ means XOR

Generator polynomial

x^3+1

$1.x^3+0.x^2+0.x^1+1.x^0$

CRC generator

1 0 0 1 4-bit

If CRC generator is of n bit then append (n-1) zeros in the end of original message

Sender

```
1001 | 1 0 1 0 0 0 0 0 0 0
     @1 0 0 1
       0 0 1 1 0 0 0 0 0 0
       @ 1 0 0 1
         0 1 0 1 0 0 0 0
         @1 0 0 1
           0 0 1 1 0 0 0
           @ 1 0 0 1
             0 1 0 1 0
             @1 0 0 1
               0 0 1 1
```

Message to be transmitted

```
1 0 1 0 0 0 0 0 0 0
          + 0 1 1
-----------------
1 0 1 0 0 0 0 0 1 1
-----------------
```

```
1001 | 1 0 1 0 0 0 0 0 1 1
     1 0 0 1
       0 0 1 1 0 0 0 0 1 1
       @ 1 0 0 1
         0 1 0 1 0 0 1 1   <- Receiver
         @1 0 0 1
           0 0 1 1 0 1 1
           @ 1 0 0 1
             0 1 0 0 1
             @1 0 0 1
               0 0 0 0
```

Zero means data is accepted