# Content Outline (Unit 3)

- **Data Link Layer**
  - Introduction
  - Error Detection and Correction
    - Block Coding and Hamming Distance
    - Error Detection
      - Parity checking
      - Cyclic Redundancy Check (CRC)
      - Check Sum
    - Forward Error Correction
      - Hamming Code

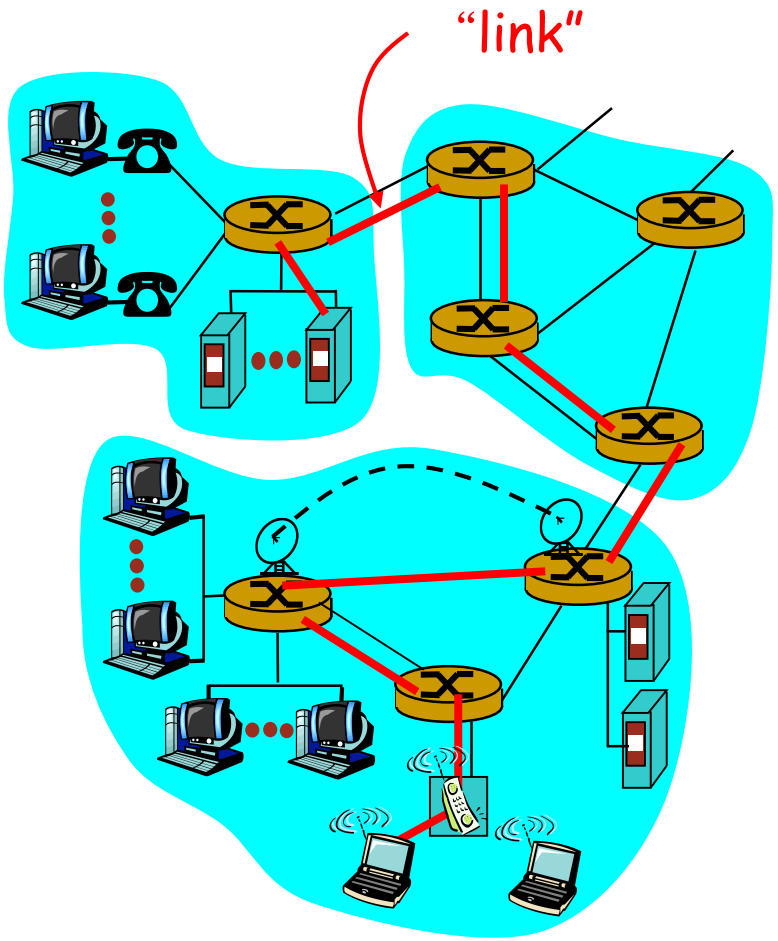
# Objectives of this Unit

- Describe the major functions at link layer
- Understand the idea of block coding and Hamming distance for error detection and correction.
- Describe the three commonly used error detection methods (parity checking, Cyclic Redundancy Check (CRC) and Check Sum) and their limitations
- Introduce the forward error correction and describe Hamming codes for error correction.

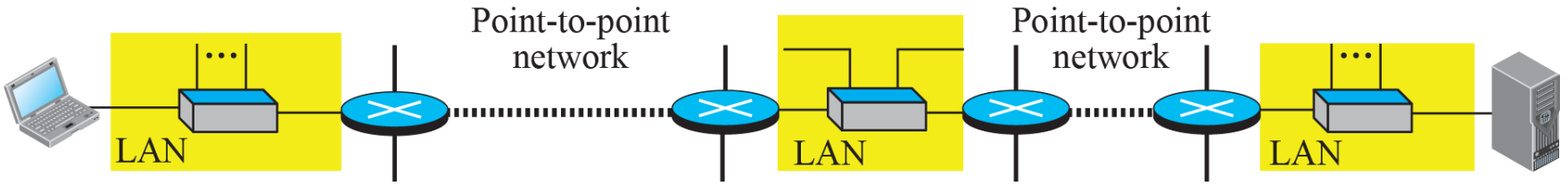
# Data Link Layer

- Hosts and routers are **nodes**
- Communication channels that connect adjacent nodes along communication path are **links**
  - wired links
  - wireless links
  - LANs
- Layer-2 packet is a **frame**, encapsulates packet (from network layer)

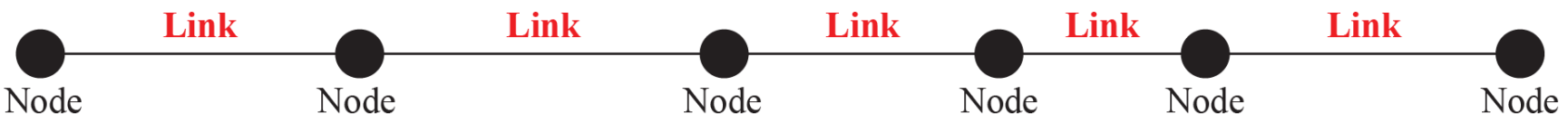
Data link layer has responsibility of transferring data (a frame) from one node to adjacent node over a link.



# Nodes and Links

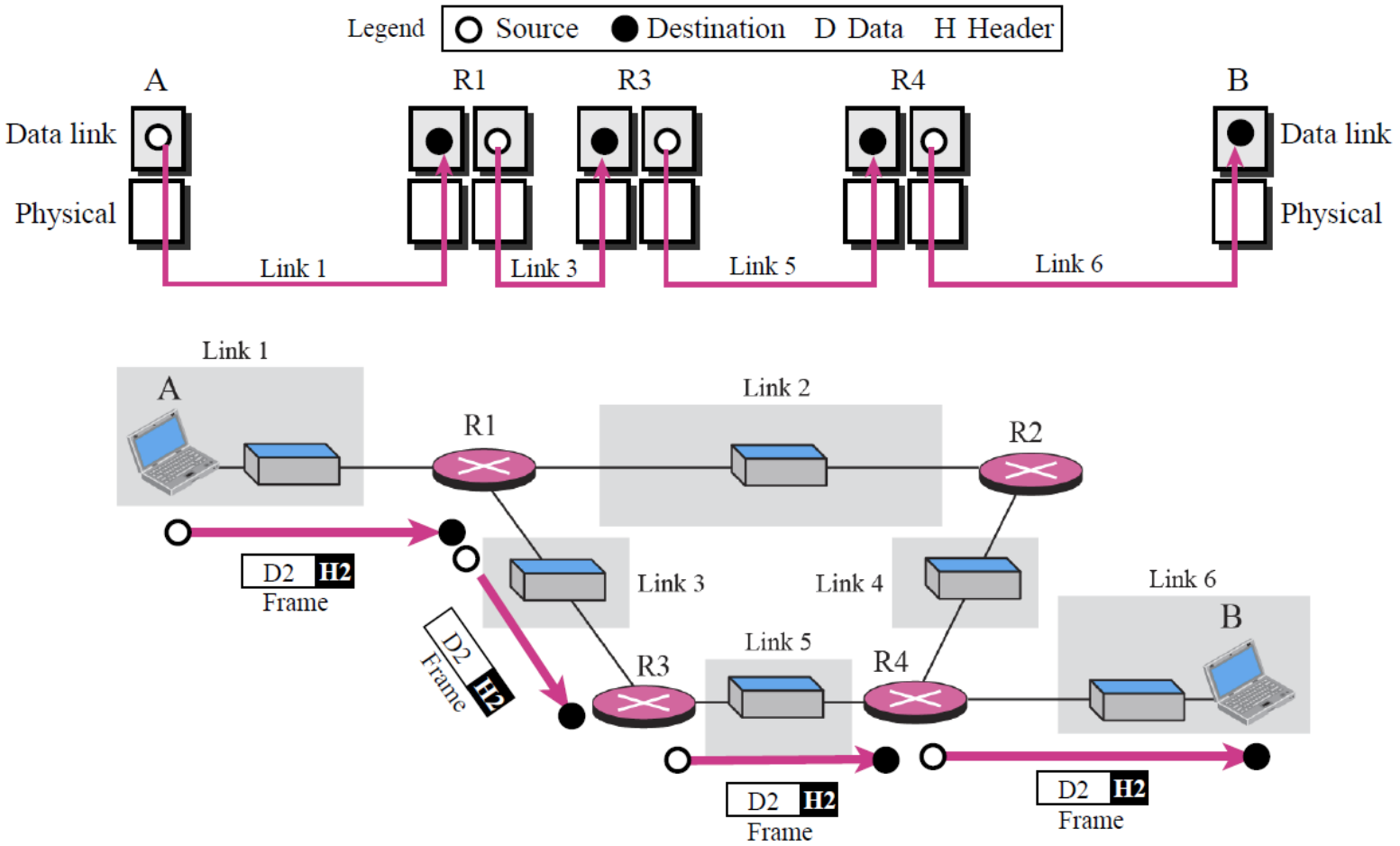



a. A small part of the Internet



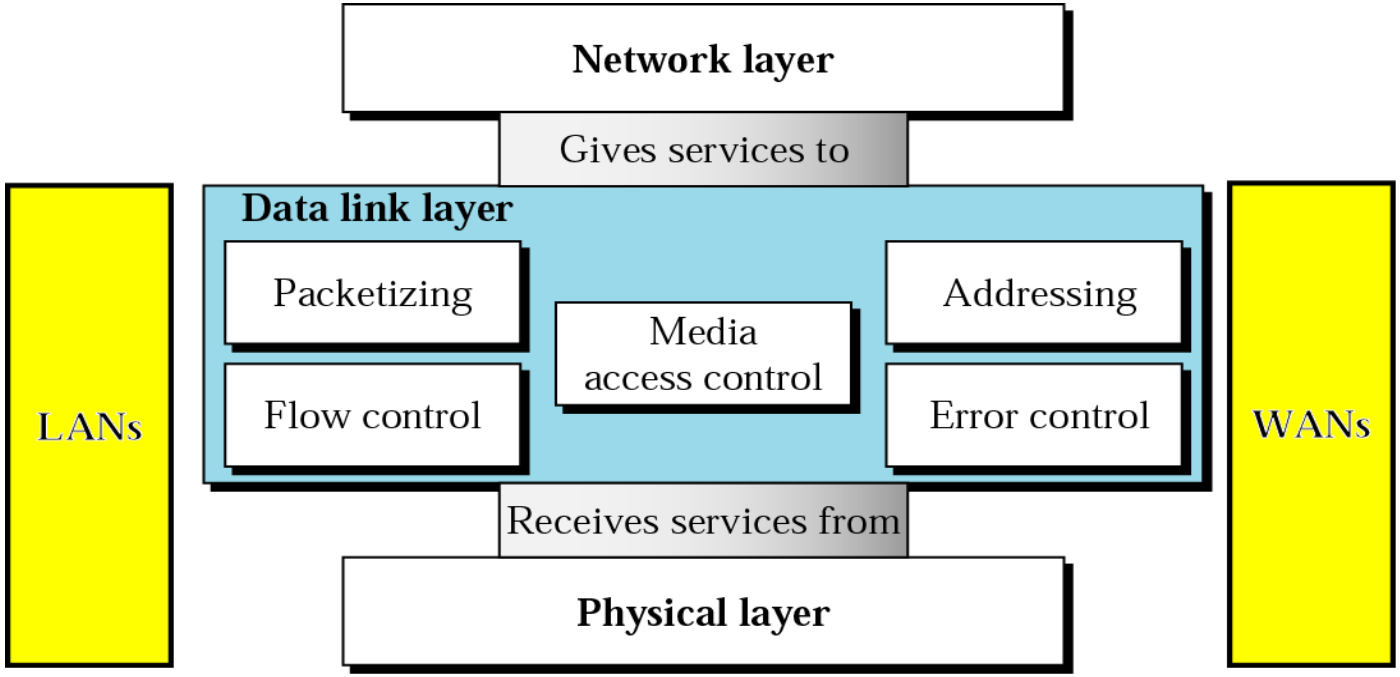

b. Nodes and links

# Communications at Link Layer



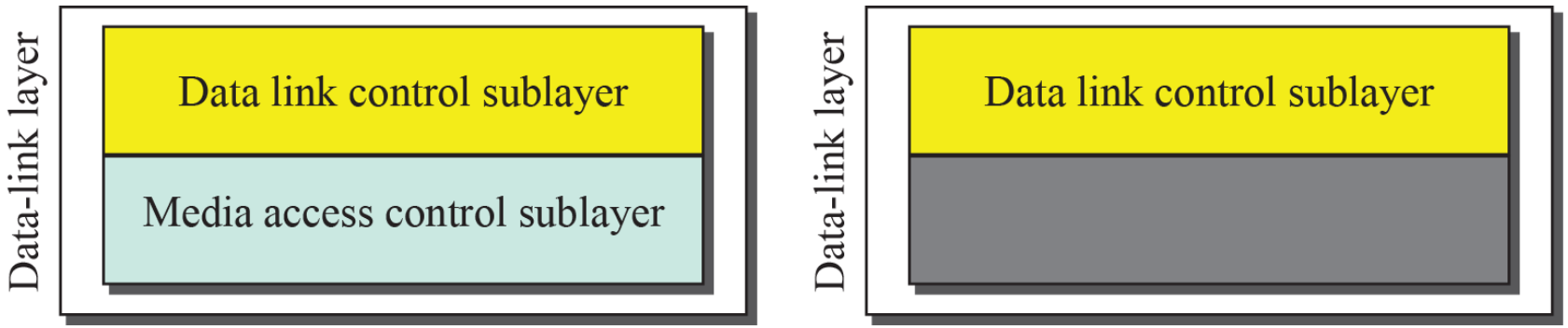
# Functions of Data Link Layer

- Framing
- Flow Control
- Error Control
- Access Control
- Addressing



# LLC and MAC Sublayers

- Data Link Layer: carrying data (frame) from one hop to the next hop

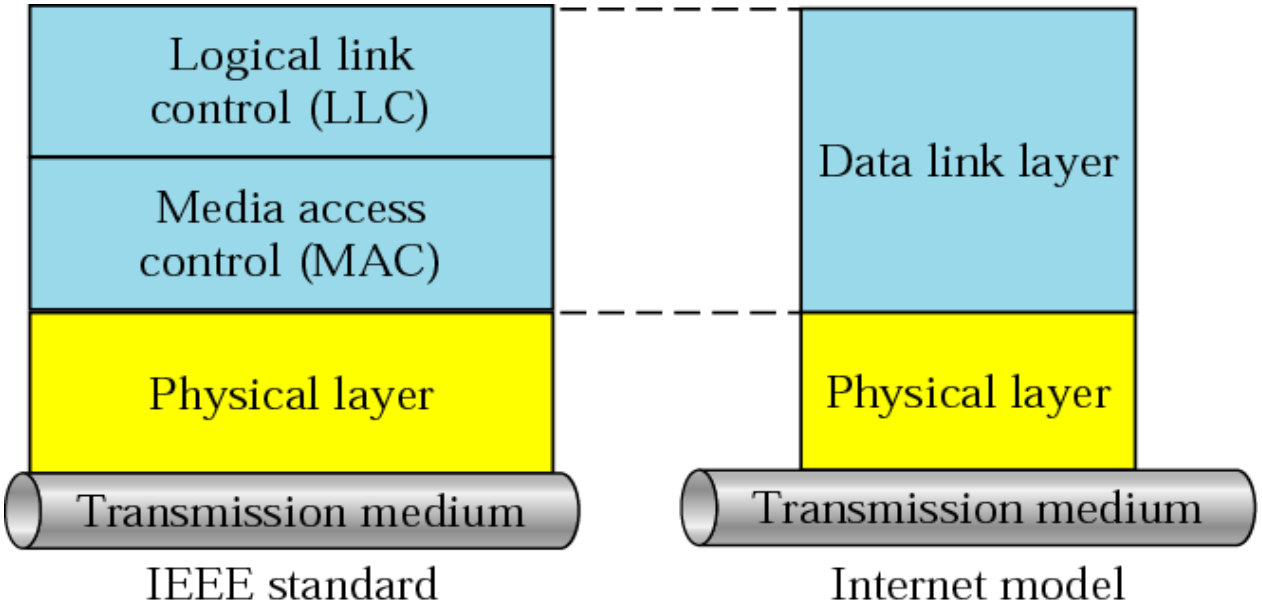



a. Data-link layer of a broadcast link

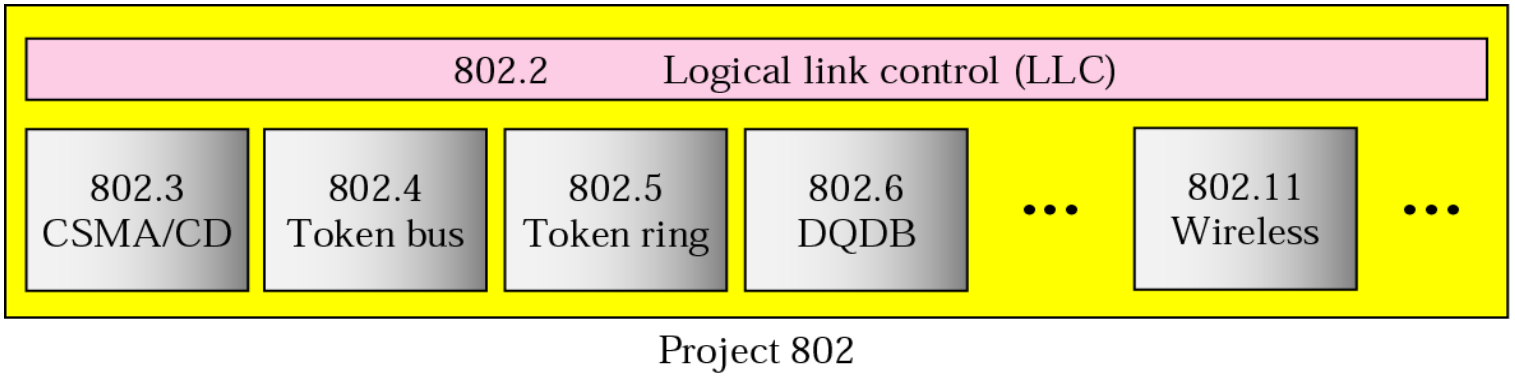
b. Data-link layer of a point-to-point link

# LLC and MAC Sublayers

- Logical Link Control (LLC): Interoperability for different LANs
- Media Access Control (MAC): Operation of the access methods



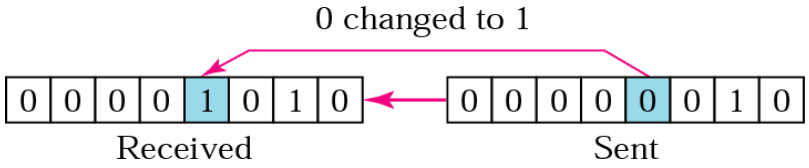
# IEEE Model on Data Link Layer

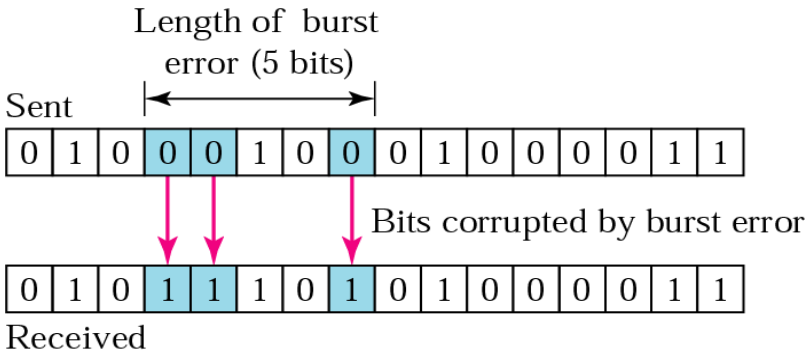



Project 802

# Types of Errors

- Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.
- Types of Error
  - Single-Bit Error: only one bit in the data unit has changed
  - Burst Error: 2 or more bits in the data unit have changed
- A single-bit error can occur in the presence of white noise.
- Burst errors can be caused by impulse noise. The effects of burst errors are greater at higher data rate.



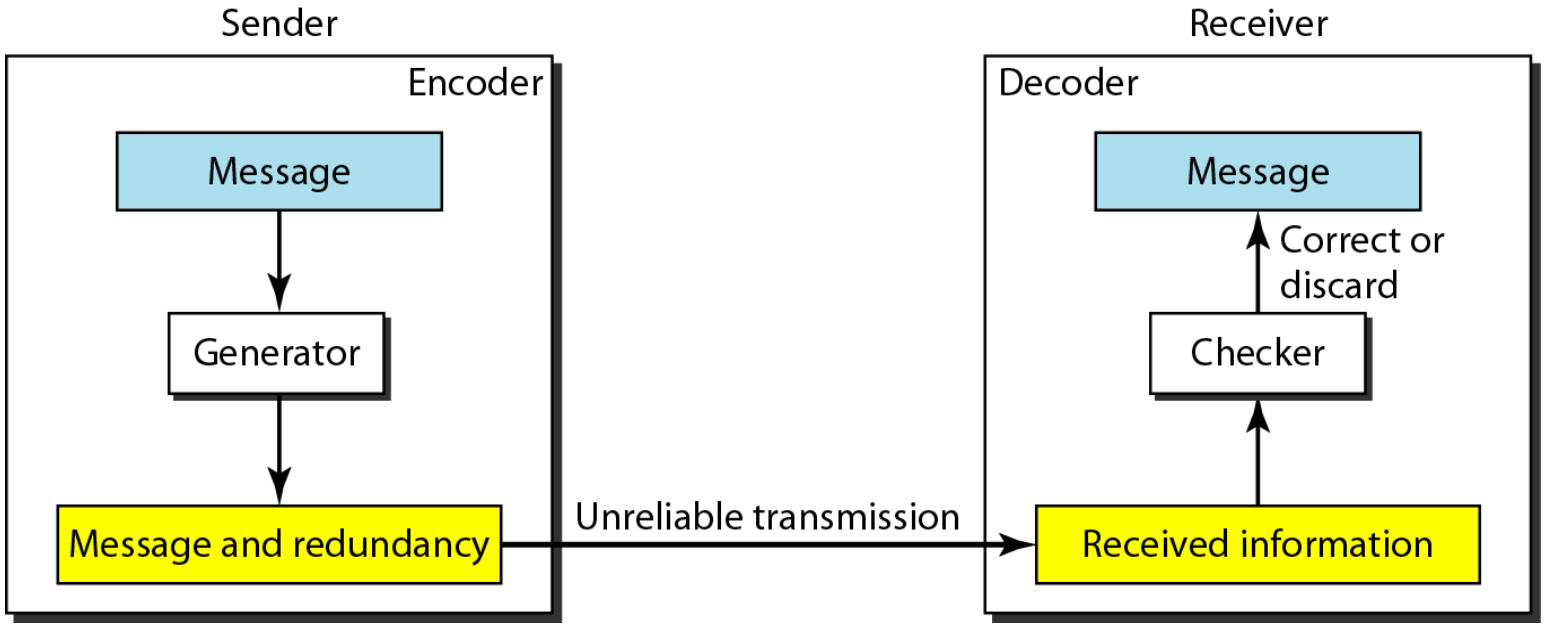
# Error Detection and Correction

- **Error detection**
  - 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 as a burst error.

- **Error correction**
  - To know the exact number of bits that are corrupted.
  - To locate the error location in the message.

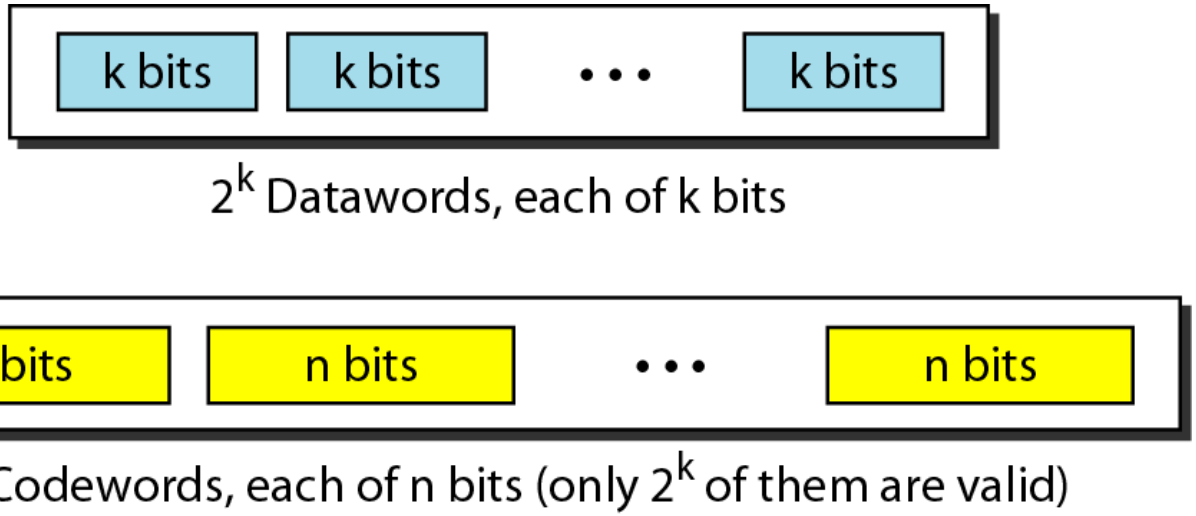
# Redundancy

- To detect or correct errors, redundant bits of data must be added



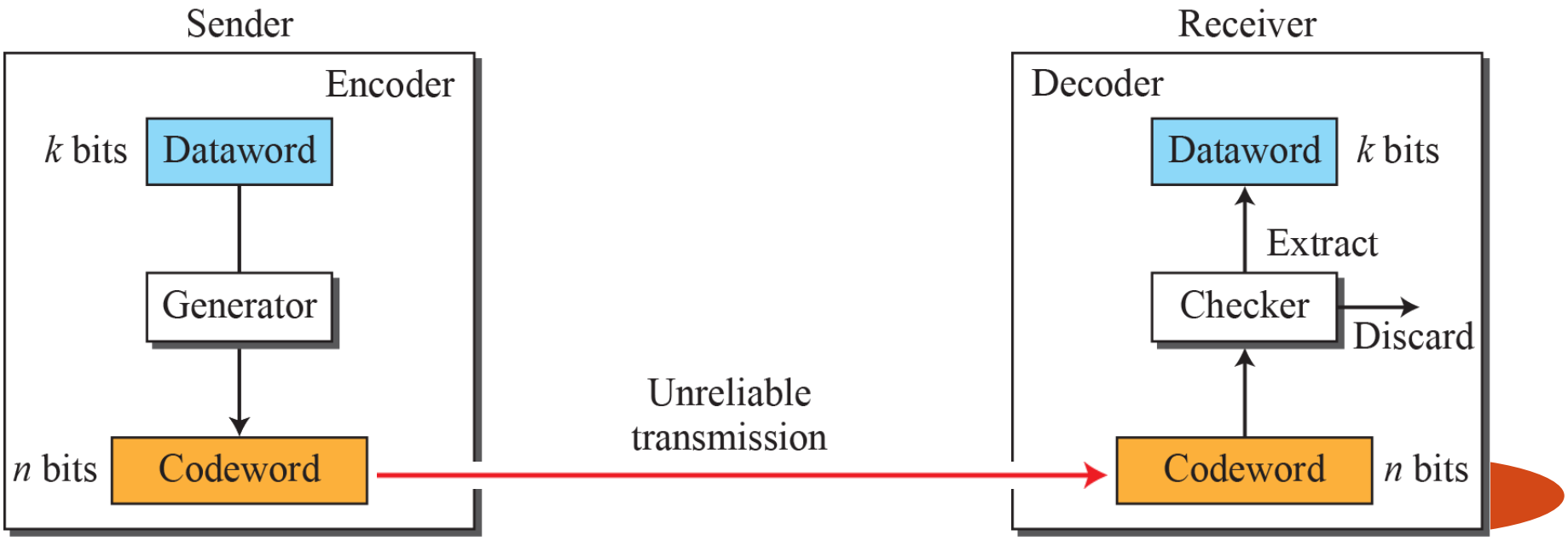
# Block Coding

- Message is divided into *k*-bit blocks
  - Known as ***datawords***
- *r* redundant bits are added
  - Blocks become $n=k+r$ bits
  - Known as ***codewords***



# Error Detection in Block Coding

- The receiver has (or can find) a list of valid codewords.
- The original codeword has changed to an invalid one ⇨ an error is declared.
- There is no way to detect every possible error!



# Example: Parity Check

Parity check for k=2 and n=3

| *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

***The Hamming Distance between two words is the number of differences between corresponding bits.***

- $d(01, 00) = 1$
- $d(11, 00) = 2$
- $d(010, 100) = 2$
- $d(0011, 1000) = 3$

# Minimum Hamming Distance

***The minimum Hamming distance is the smallest Hamming distance between all possible pairs of codewords in a codebook***

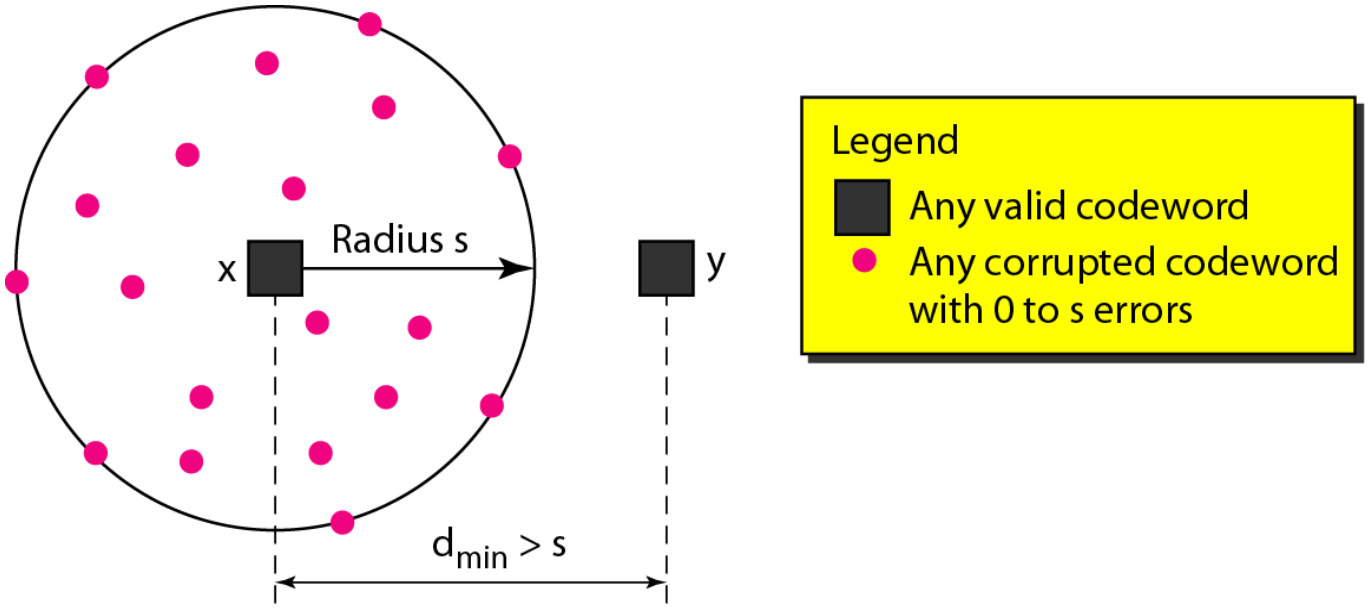
- Find the minimum Hamming Distance of the following codebook

| |
|---|
| 00000 |
| 01011 |
| 10101 |
| 11110 |

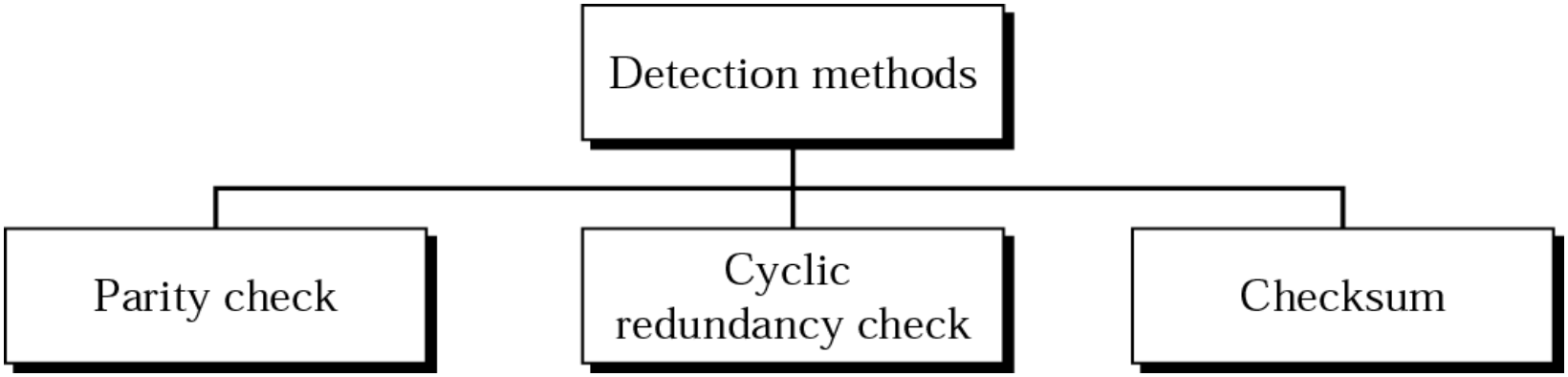
# Detection Capability of Code

- To guarantee the detection of up to *s*-bit errors, the minimum Hamming distance in a block code must be

$$d_{min} = s + 1$$



# Error Detection Methods



# Parity Check

- Most common, least complex
- Single bit is added to a block
- Two schemes:
  - Even parity – Maintain even number of 1s
    - E.g., 1011 → 10111
  - Odd parity – Maintain odd number of 1s
    - E.g., 1011 → 10110
- What is the minimum Hamming distance of the codewords of single parity check?

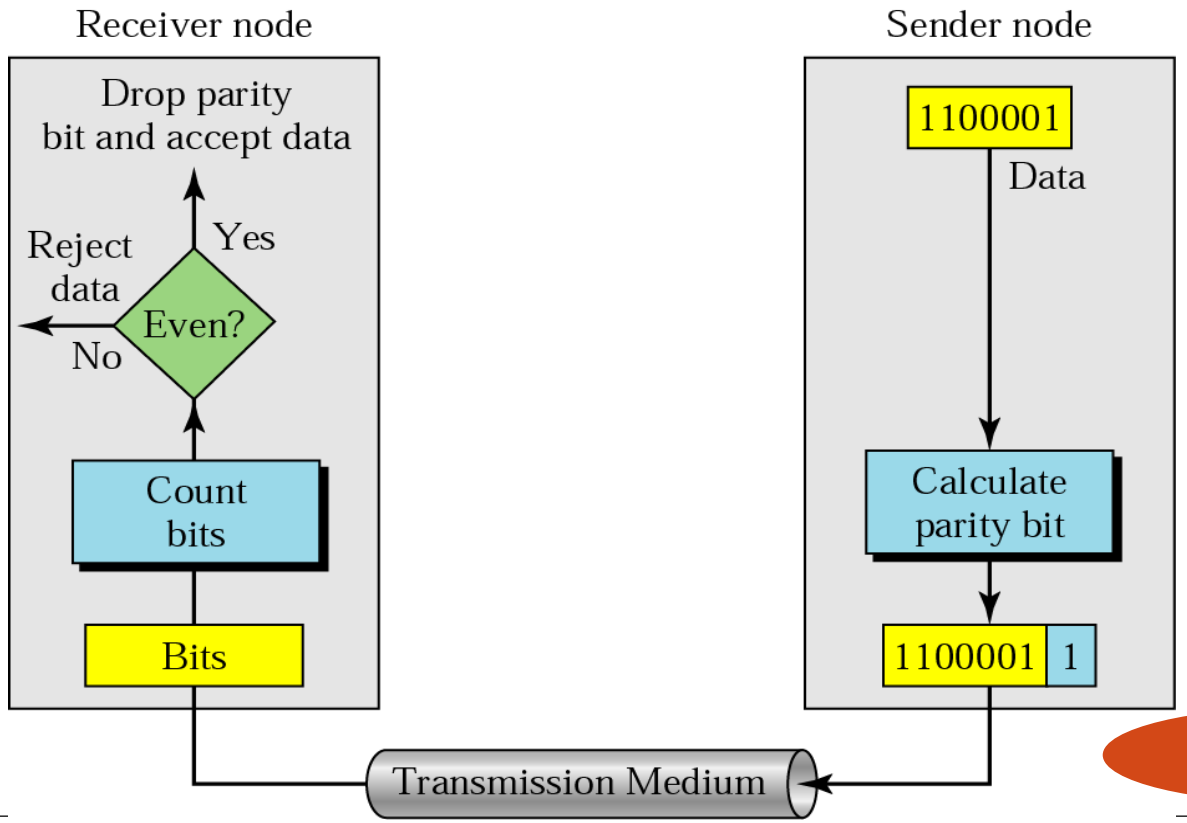
# Parity Check

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

- A parity bit is added to every data unit so that the total number of 1s is even (or odd for odd-parity)



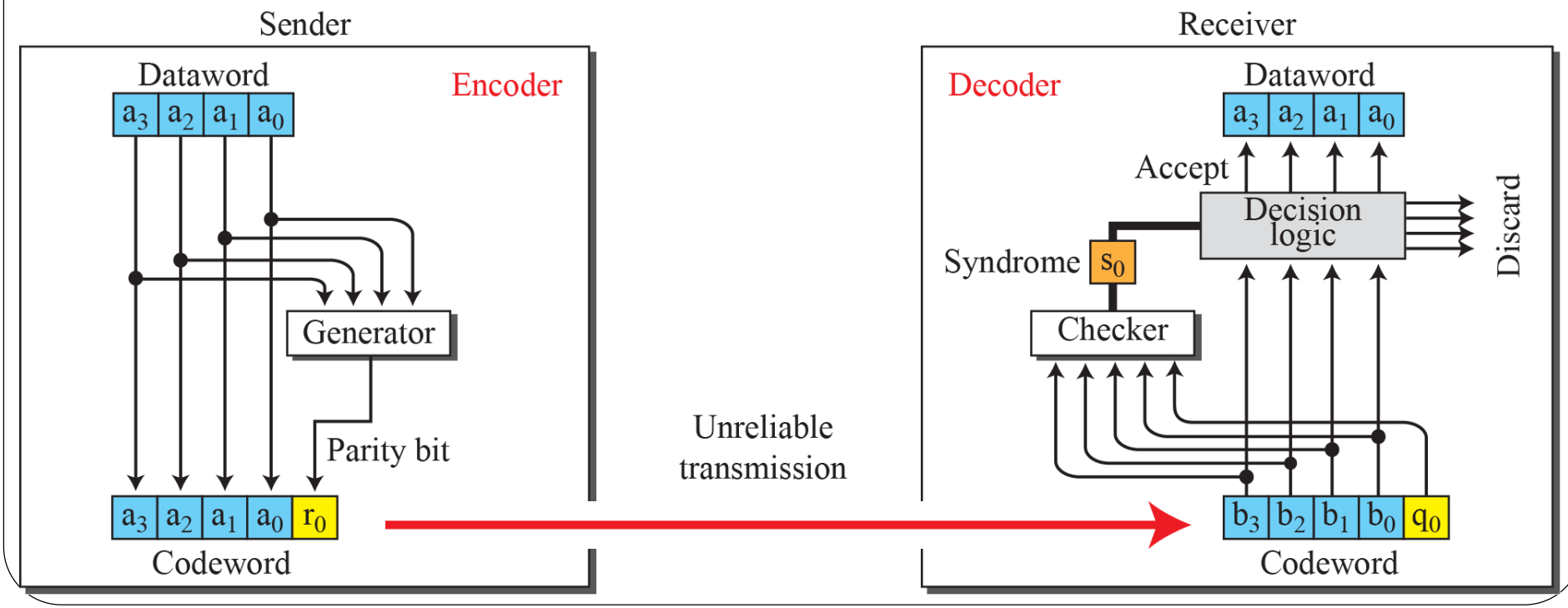
# Parity Check

- Single Parity Check
  - Problems: Only detects an odd number of errors
  - When the data block is badly garbled by a long burst error, the probability of detecting the error is 0.5 and therefore not acceptable

# Parity Check -Implementation

- The parity bit is obtained by adding the k bits of the codewords (modulo-2), for example, for a 4-bit block

$$r_0 = a_3 \oplus a_2 \oplus a_1 \oplus a_0 \quad \text{(modulo-2)}$$



# Example

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**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

Now suppose the word world is received by the receiver without being corrupted in transmission

11101110 11011110 11100100 11011000 11001001

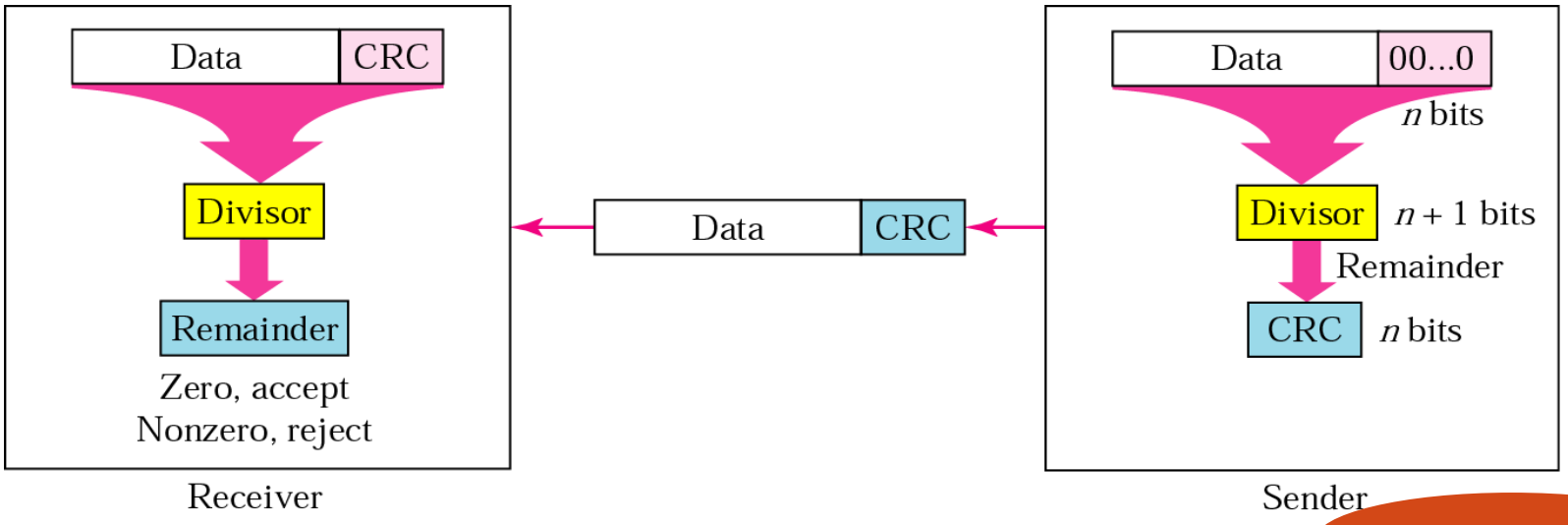
The receiver counts the 1s in each character and comes up with even numbers (6, 6, 4, 4, 4). The data are accepted

# Parity Check

- Simple parity check can detect all single-bit errors
- It *cannot* detect burst errors only if the total number of errors in each data unit is even (even parity check!)
- E.g. 110111010 => 111101010

# Cyclic Redundancy Check

- Unlike the parity check which is based on addition, cyclic redundancy check (CRC) is based on binary division
- Instead of adding bits, *a sequence of redundant bits* is appended to the end of a data unit so that the resulting data unit becomes exactly divisible by a second predetermined binary number

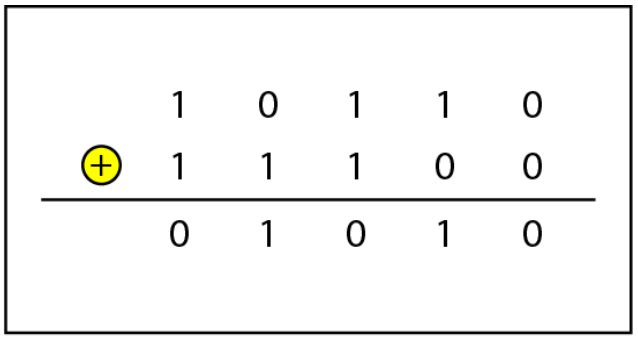


# XOR Operation

- Main operation for computing error detection/correction codes
- Similar to modulo-2 addition


a. Two bits are the same, the result is 0.


b. Two bits are different, the result is 1.




c. Result of XORing two patterns

# Modulo-2 Arithmetic

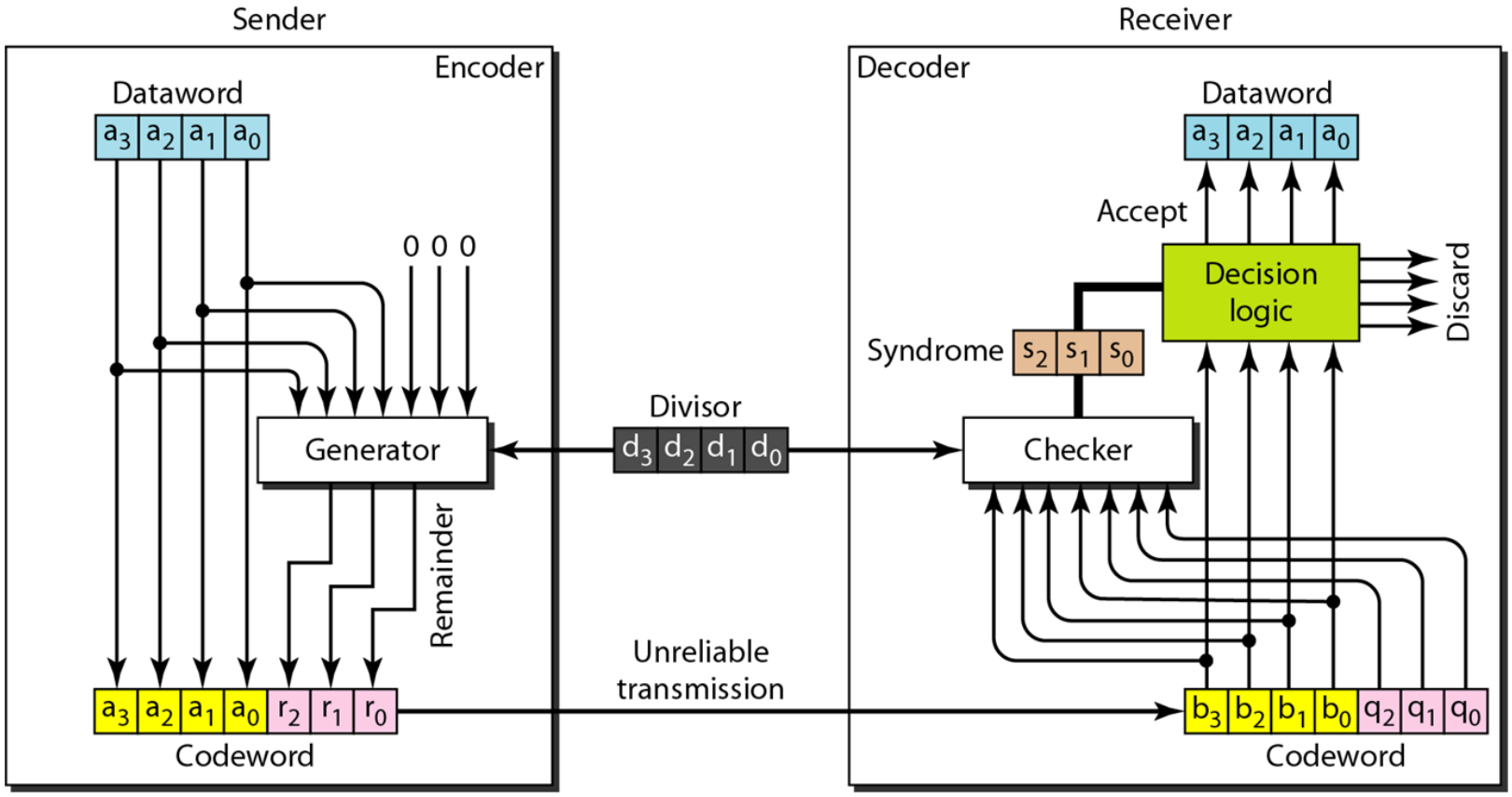
- Modulo-2 arithmetic uses binary addition with no carries, which is just the exclusive-OR (XOR) operation.
- Binary subtraction with no carries is also interpreted as the XOR operation.

```
  1111
+ 1010
------
  0101
```

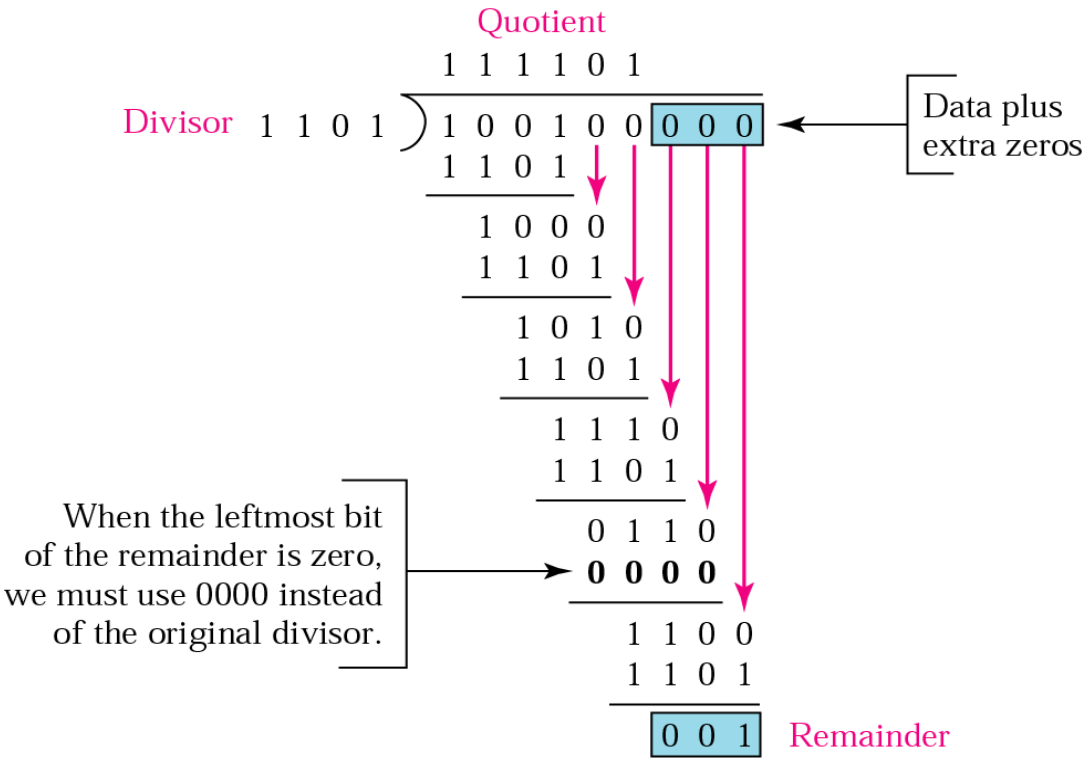
```
  1111
-0101
------
  1010
```

```
  11001
x   11
------
  11001
 11001
------
101011
```

# CRC Encoder and Decoder



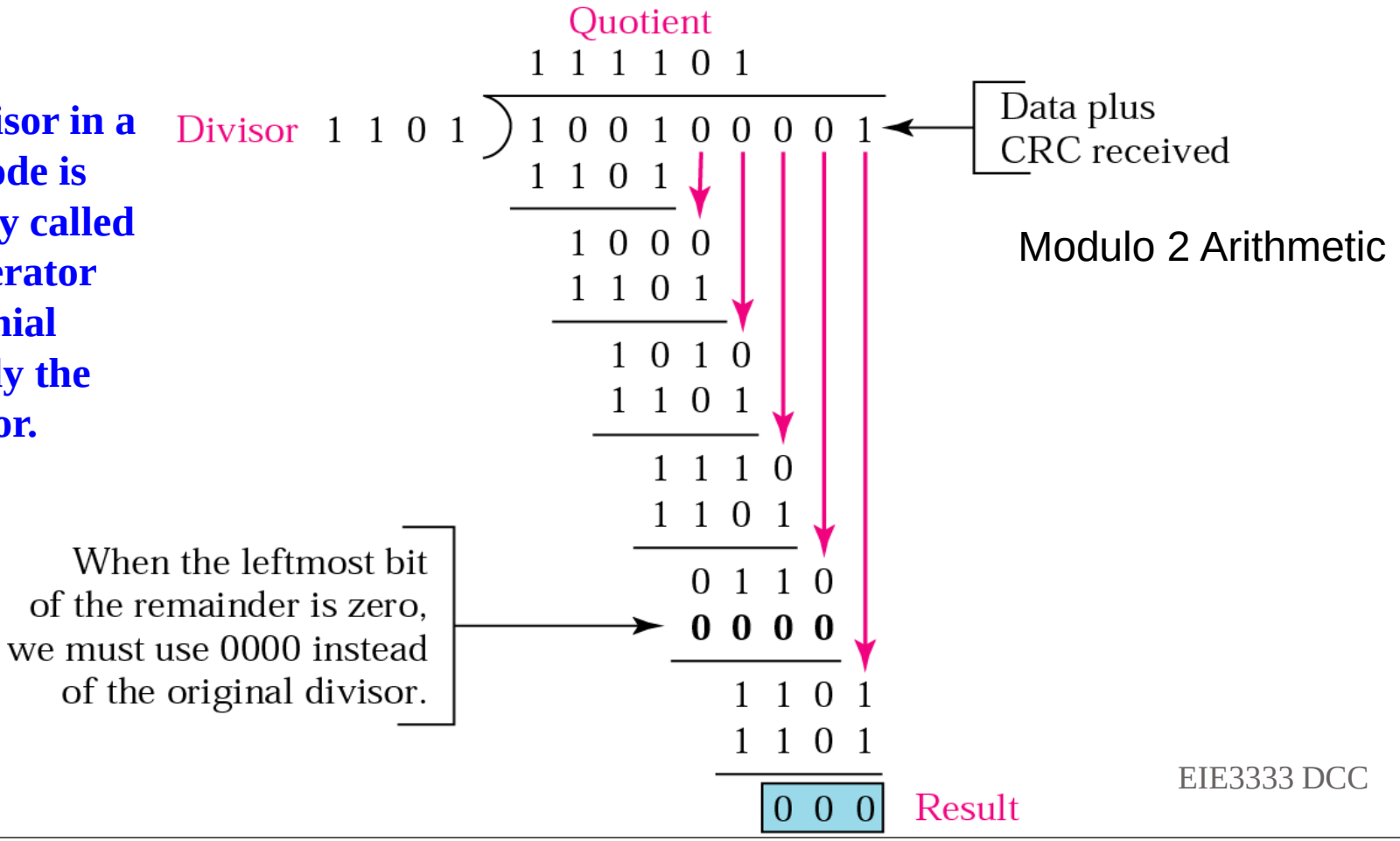
# CRC Generator




Modulo 2 Arithmetic

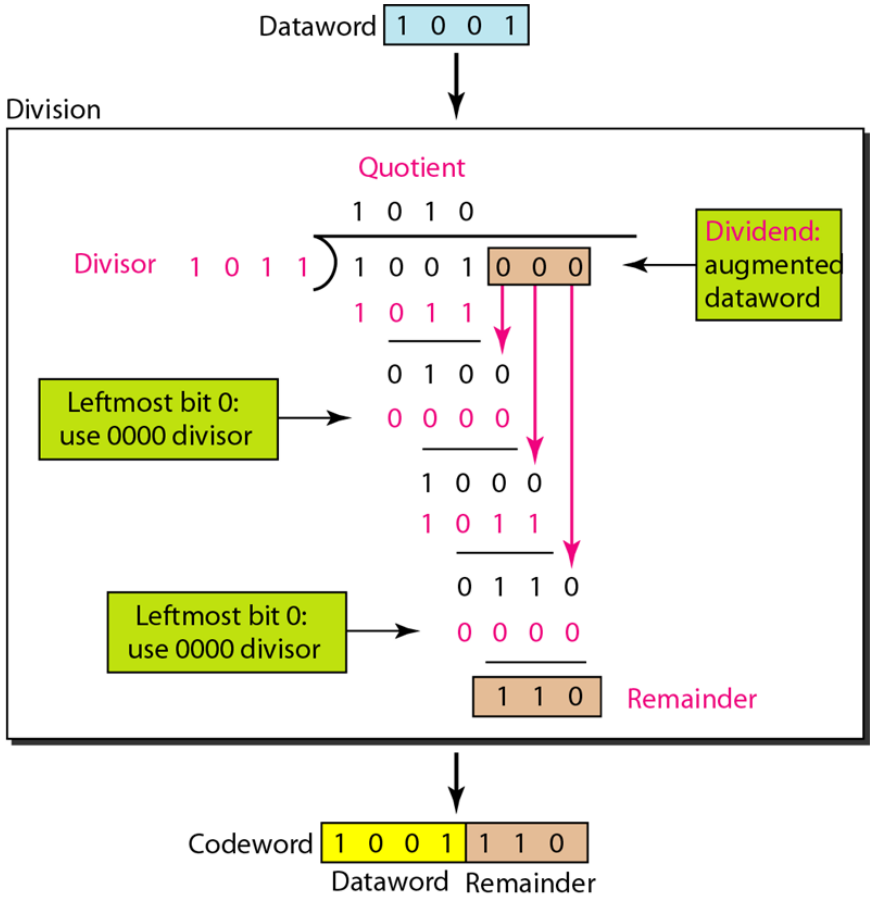
# CRC Checker

**The divisor in a cyclic code is normally called the generator polynomial or simply the generator.**

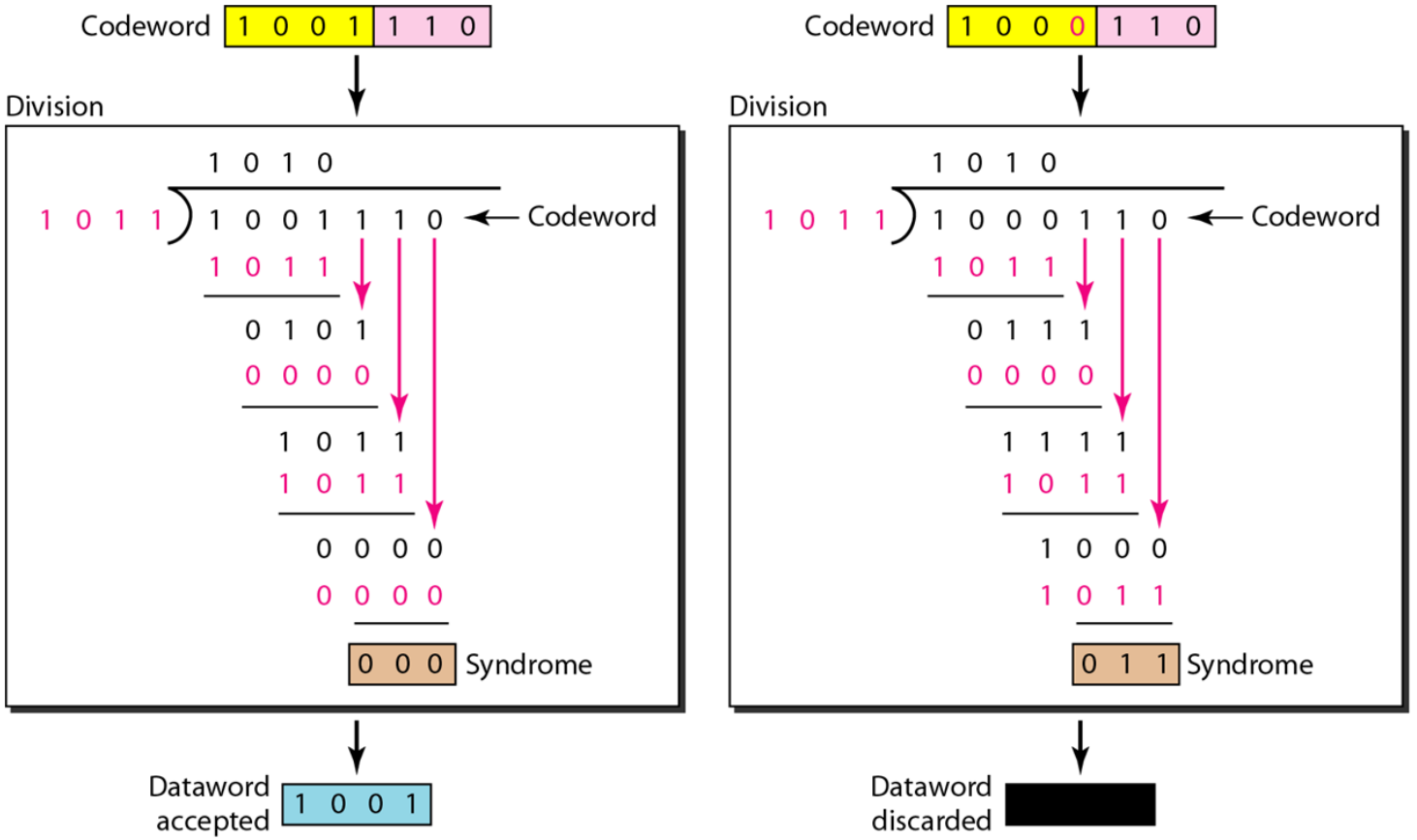



Modulo 2 Arithmetic

# CRC Example


Dataword 1 0 0 1
Division
Quotient
1 0 1 0
Divisor 1 0 1 1
1 0 0 1 0 0 0
Dividend: augmented dataword
1 0 1 1
0 1 0 0
0 0 0 0
Leftmost bit 0: use 0000 divisor
1 0 0 0
1 0 1 1
0 1 1 0
0 0 0 0
Leftmost bit 0: use 0000 divisor
1 1 0
Remainder
Codeword 1 0 0 1 1 1 0
Dataword Remainder

# CRC Example

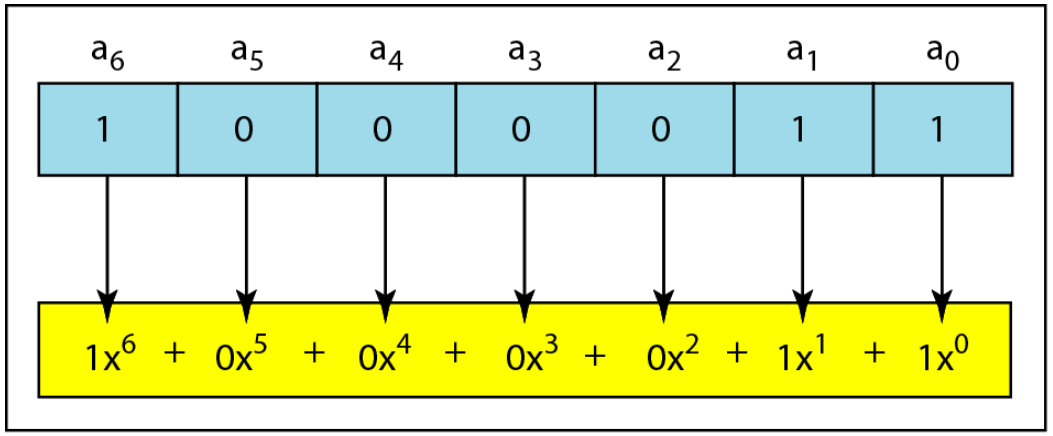

Codeword 1 0 0 1 1 1 0
Division
1 0 1 0
1 0 1 1 | 1 0 0 1 1 1 0 ← Codeword
1 0 1 1
0 1 0 1
0 0 0 0
1 0 1 1
1 0 1 1
0 0 0 0
0 0 0 0
0 0 0 Syndrome
Dataword accepted 1 0 0 1
Codeword 1 0 0 0 1 1 0
Division
1 0 1 0
1 0 1 1 | 1 0 0 0 1 1 0 ← Codeword
1 0 1 1
0 1 1 1
0 0 0 0
1 1 1 1
1 0 1 1
1 0 0 0
1 0 1 1
0 1 1 Syndrome
Dataword discarded

# CRC Codes

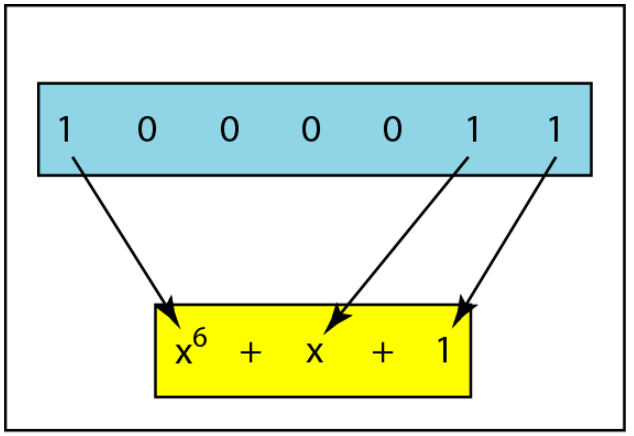
- In a cyclic code,
  - If $s(x) \neq 0$, one or more bits is corrupted.
  - If $s(x) = 0$, either
    - No bit is corrupted. or
    - Some bits are corrupted, but the decoder failed to detect them. (In a cyclic code, those errors that are divisible by $g(x)$ are not caught.)

# Polynomial Representation

- More common representation than binary form
- Easy to analyze
- Divisor is commonly called ***generator polynomial***

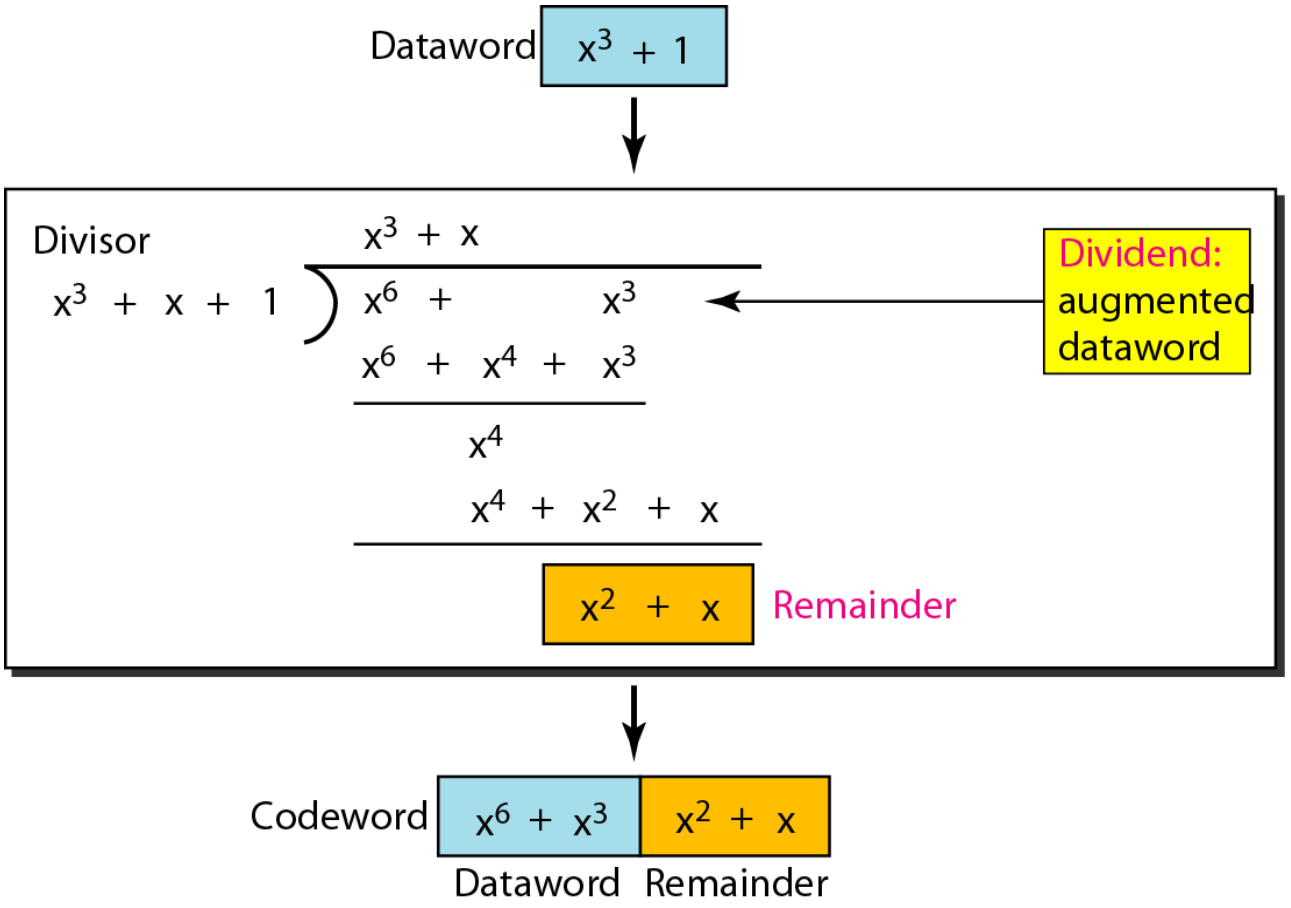



a. Binary pattern and polynomial




b. Short form

# Division Using Polynomial



# Strength of CRC

- Can be analyzed using polynomial
  - $M(x)$ – Original message
  - $G(x)$ – Generator polynomial of degree **n**
  - $R(x)$ – Generated CRC

$$M(x)\cdot x^n = Q(x)\cdot G(x) + R(x)$$

- Transmitted message is

$$M(x)\cdot x^n - R(x)$$

  which is divisible by $G(x)$

# Strength of CRC

- Received message is

$$M(x) \cdot x^n - R(x) + E(x)$$

where $E(x)$ represents bit errors

- Receiver does not detect any error when $E(x)$ is divisible by $G(x)$, which means either:
  - $E(x) = 0 \rightarrow$ No error
  - $E(x) \neq 0 \rightarrow$ Undetectable error

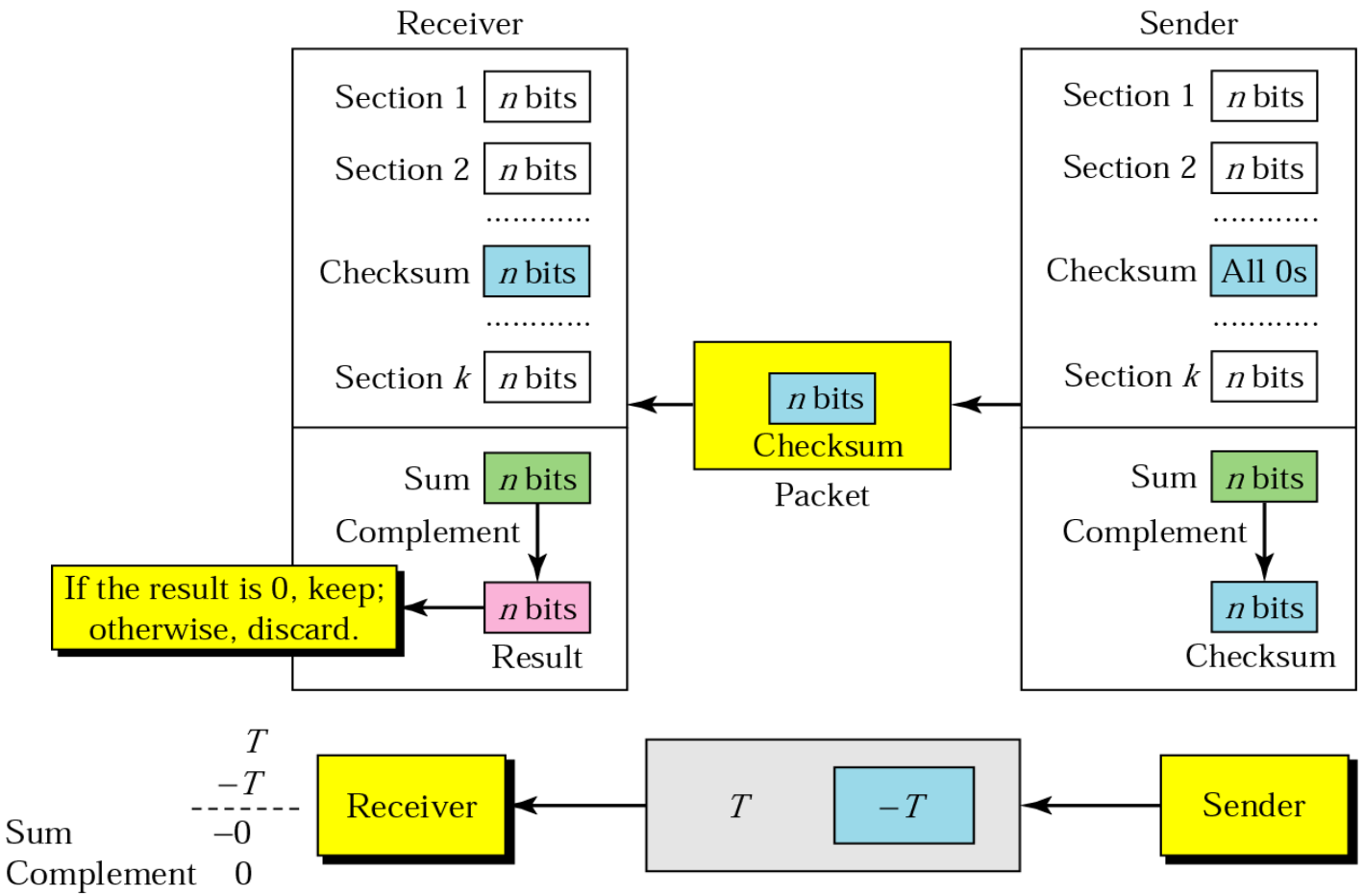
# Standard Polynomials

| *Name* | *Polynomial* | *Used in* |
|---|---|---|
| CRC-8 | $x^8 + x^2 + x + 1$<br>**100000111** | ATM header |
| CRC-10 | $x^{10} + x^9 + x^5 + x^4 + x^2 + 1$<br>**11000110101** | ATM AAL |
| CRC-16 | $x^{16} + x^{12} + x^5 + 1$<br>**1000100000100001** | HDLC |
| CRC-32 | $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$<br>**10000010011000001000111011011 0111** | LANs |

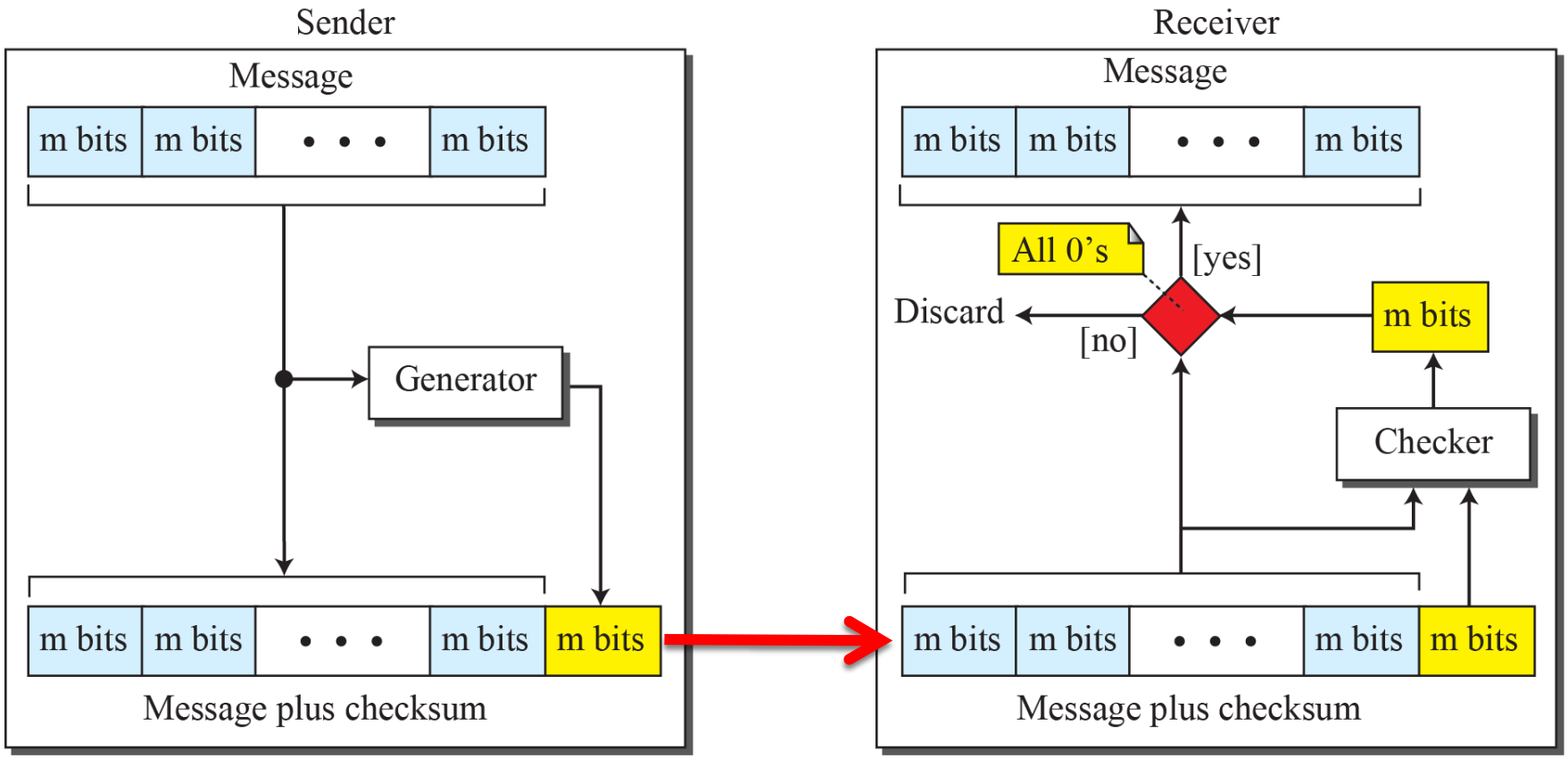
# Checksum

- **Main ideas:**
  - Checksum generator subdivides the data unit into equal segments of n bits
  - These segment are added using ones complement in such a way that the total is n bits long
  - That total is then complemented and appended to the end of the original data unit as redundancy bits
  - The sum of data segment is T, the checksum will be -T

# Checksum


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

# Checksum


Sender
Message
m bits
m bits
m bits
Generator
m bits
m bits
m bits
m bits
Message plus checksum
Receiver
Message
m bits
m bits
m bits
All 0's
[yes]
Discard
[no]
m bits
Checker
m bits
m bits
m bits
m bits
m bits
Message plus 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**

# Example

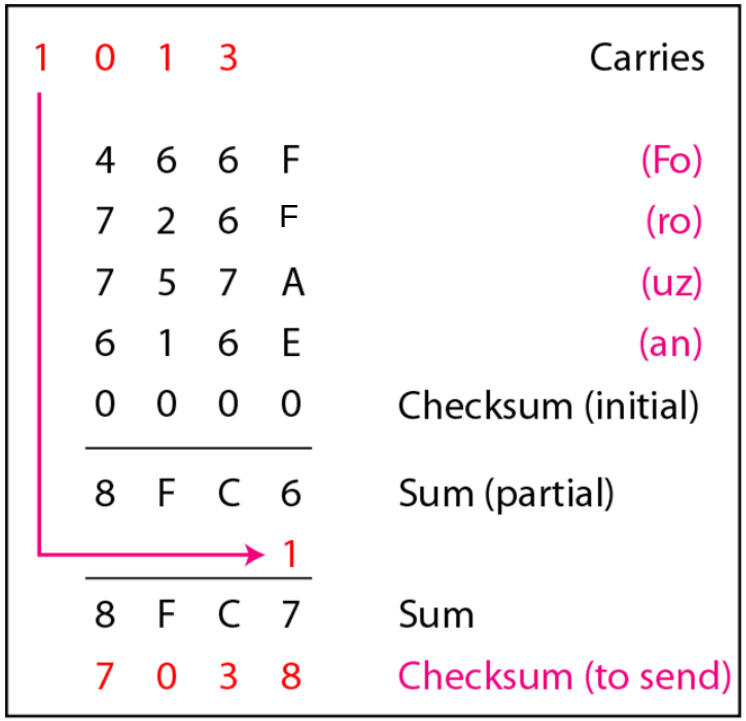
Now suppose the receiver receives the pattern sent in Example 3 and there is no error.

10101001 00111001 00011101

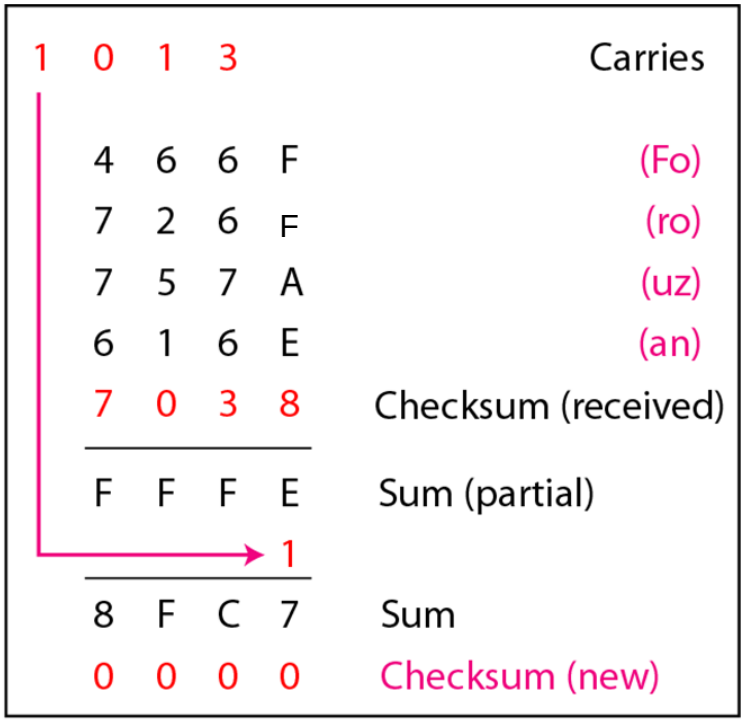
When the receiver adds the three sections, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

| | |
|---|---|
| | 10101001 |
| | 00111001 |
| | 00011101 |
| Sum | 11111111 |
| Complement | **00000000** means that the pattern is OK. |

# Example



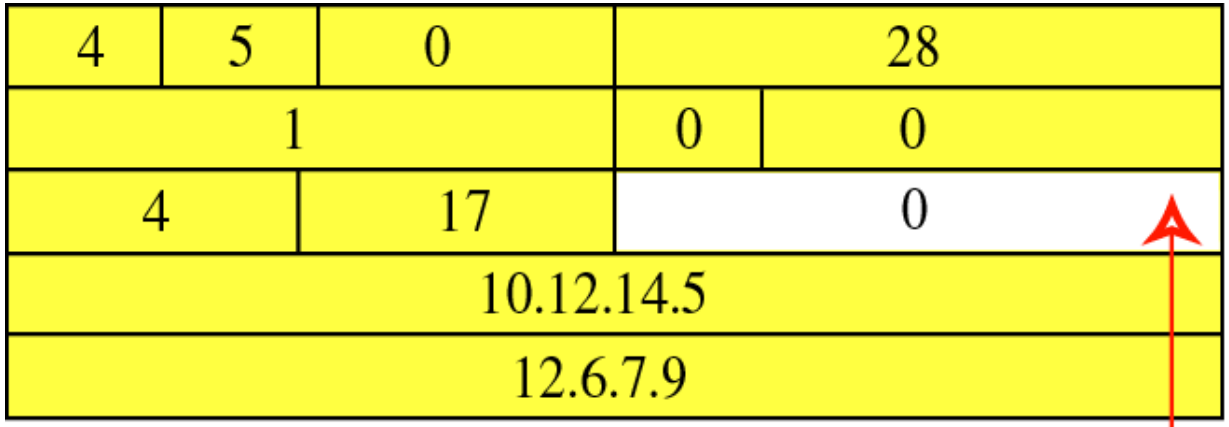

a. Checksum at the sender site




a. Checksum at the receiver site

# Example:

# Checksum of IP Header




| | | |
|---|---|---|
| 4, 5, and 0 | → | 01000101 00000000 |
| 28 | → | 00000000 00011100 |
| 1 | → | 00000000 00000001 |
| 0 and 0 | → | 00000000 00000000 |
| 4 and 17 | → | 00000100 00010001 |
| 0 | → | 00000000 00000000 |
| 10.12 | → | 00001010 00001100 |
| 14.5 | → | 00001110 00000101 |
| 12.6 | → | 00001100 00000110 |
| 7.9 | → | 00000111 00001001 |
| Sum | → | 01110100 01001110 |
| Checksum | → | 10001011 10110001 |

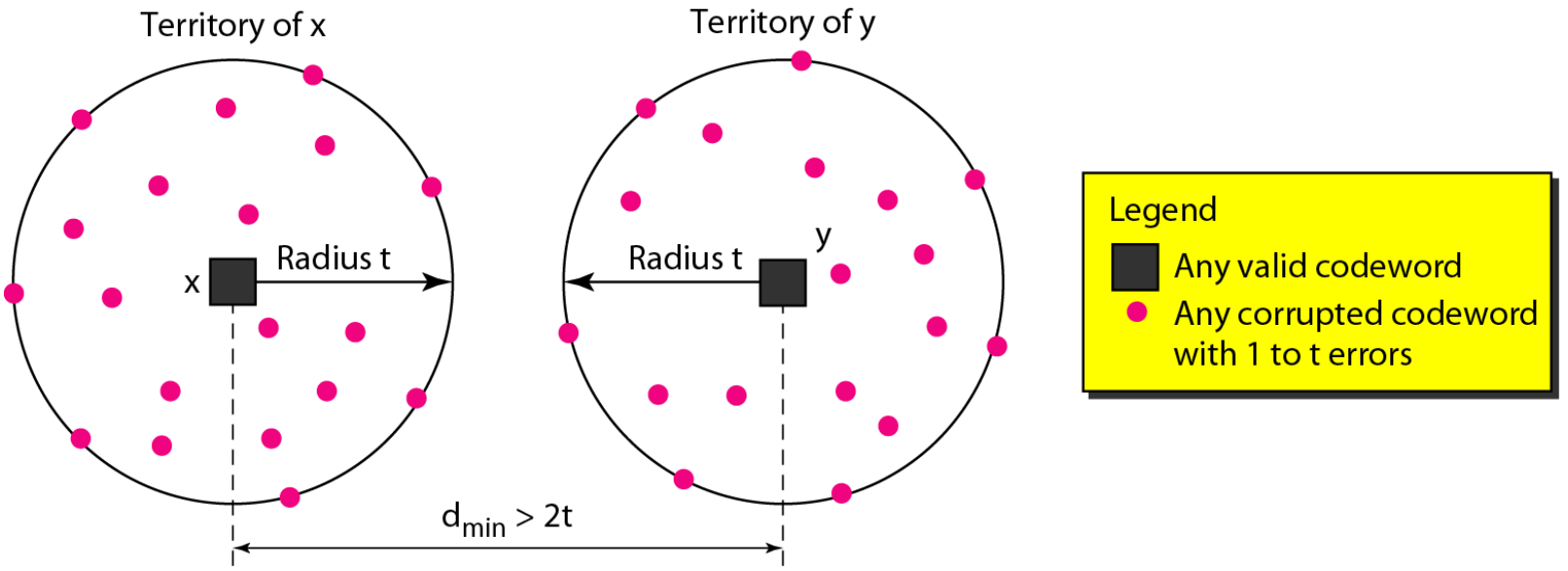
# Error Correction

- Once error is detected, correction can be handled
- Retransmission
  - The receiver asks the sender to retransmit the entire data unit (will be discussed)
- Forward Error Correction
  - The receiver can use an error-correcting code, which automatically corrects certain errors

# Correction Capability of Code

- To guarantee the correction of up to *t*-bit errors, the minimum Hamming distance in a block code must be

$$d_{min} = 2t + 1$$



# Forward Error Correction

- Consider only a single-bit error in $k$ bits of data
  - $k$ possibilities for an error
  - One possibility for no error
  - No. of possibilities = $k + 1$

- Add r redundant bits to distinguish these possibilities; we need

$$2^r \geq k+1$$

- But the r bits are also transmitted along with data; hence

$$2^r \geq k+r+1$$

# Number of Redundant Bits

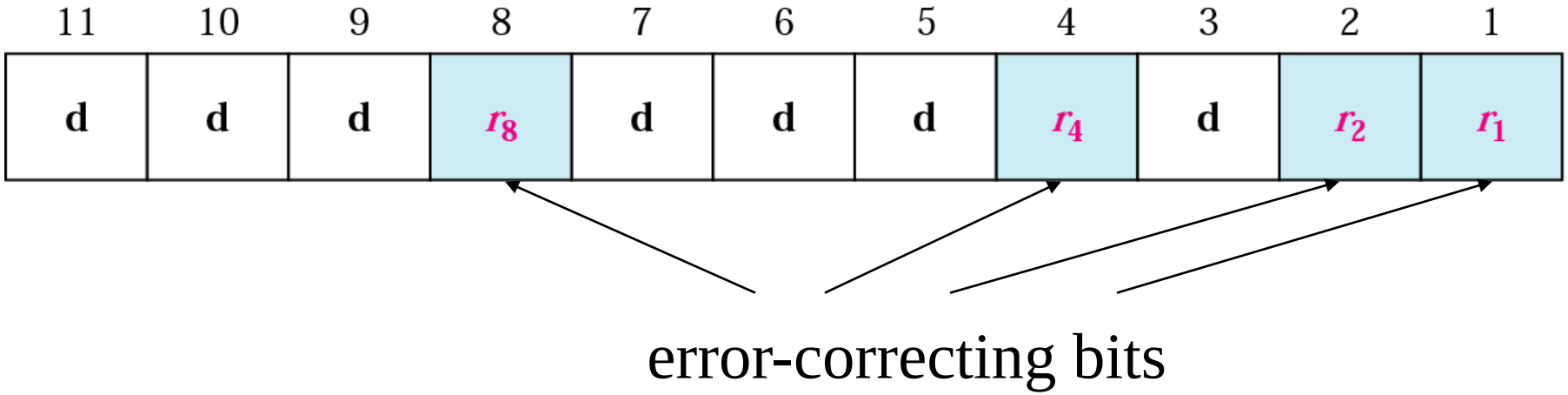
| Number of data bits $k$ | Number of redundancy bits $r$ | Total bits $k + r$ |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 5 |
| 3 | 3 | 6 |
| 4 | 3 | 7 |
| 5 | 4 | 9 |
| 6 | 4 | 10 |
| 7 | 4 | 11 |

# Hamming Code

- Hamming Code is type of Error Correcting Code (ECC)
- Provides error detection and correction mechanism
- Adopt parity concept, but have more than one parity bit
- Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors
- Simple, powerful FEC
- Widely used in computer memory
  - Known as ECC memory

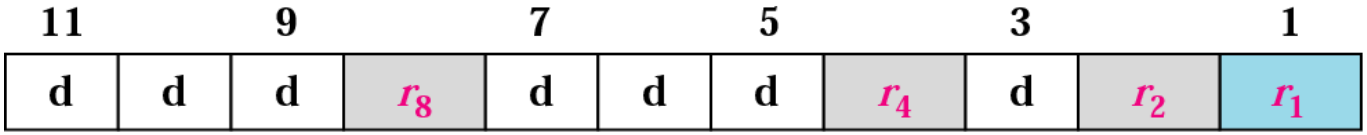
# Hamming Code

Hamming (11,7)

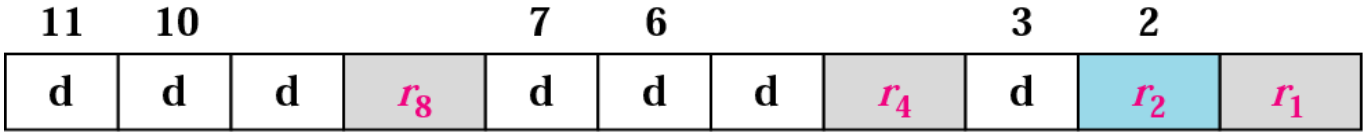


# Redundant Bit Calculation

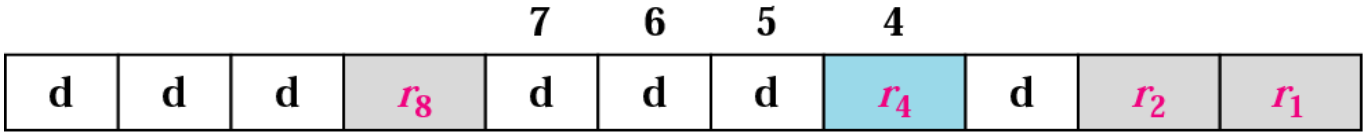
$r_1$ will take care of these bits.



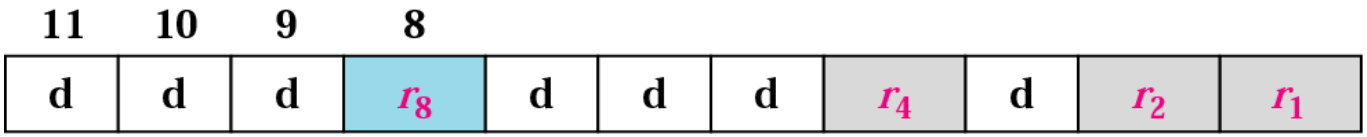

$r_2$ will take care of these bits.



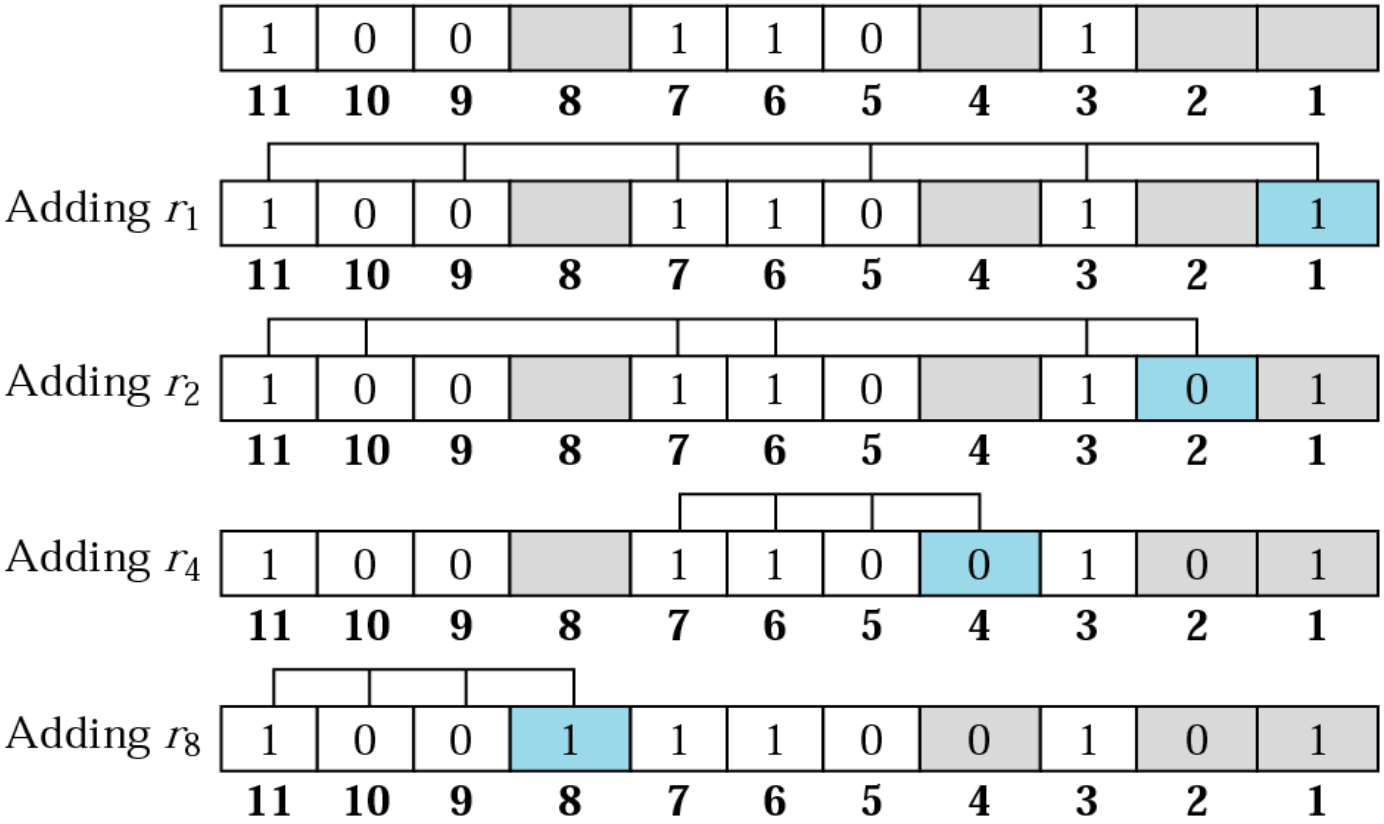

$r_4$ will take care of these bits.




$r_8$ will take care of these bits.



# Example: Hamming Code

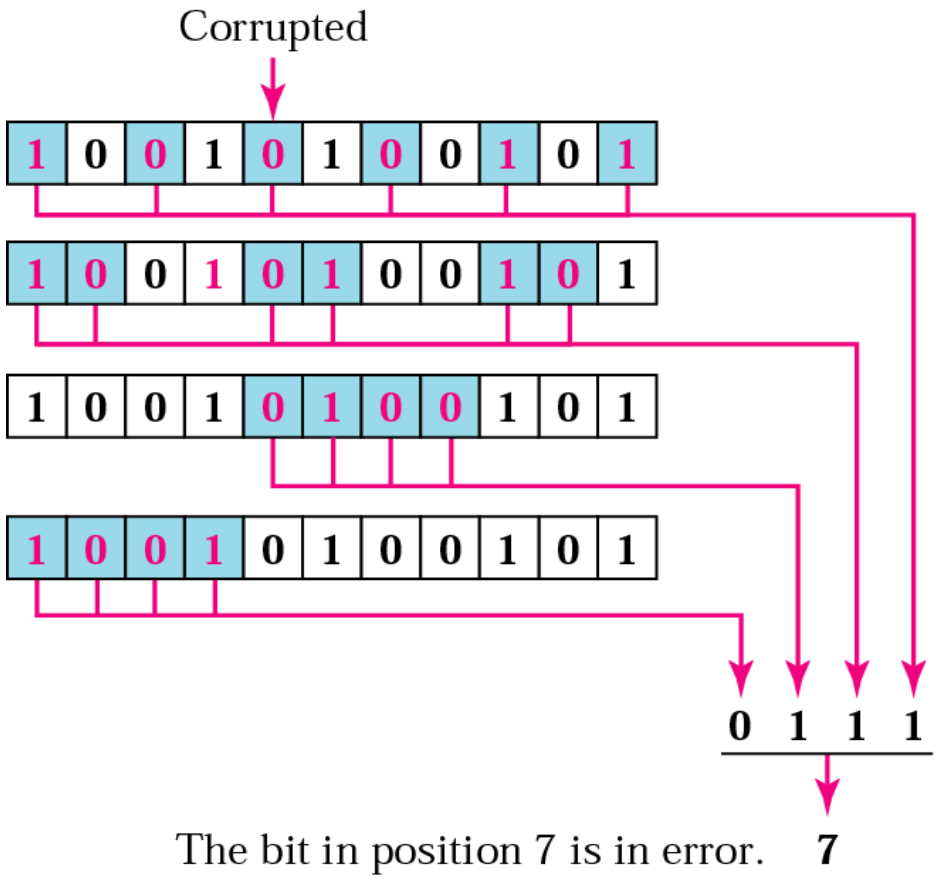



**Data:**
**1 0 0 1 1 0 1**

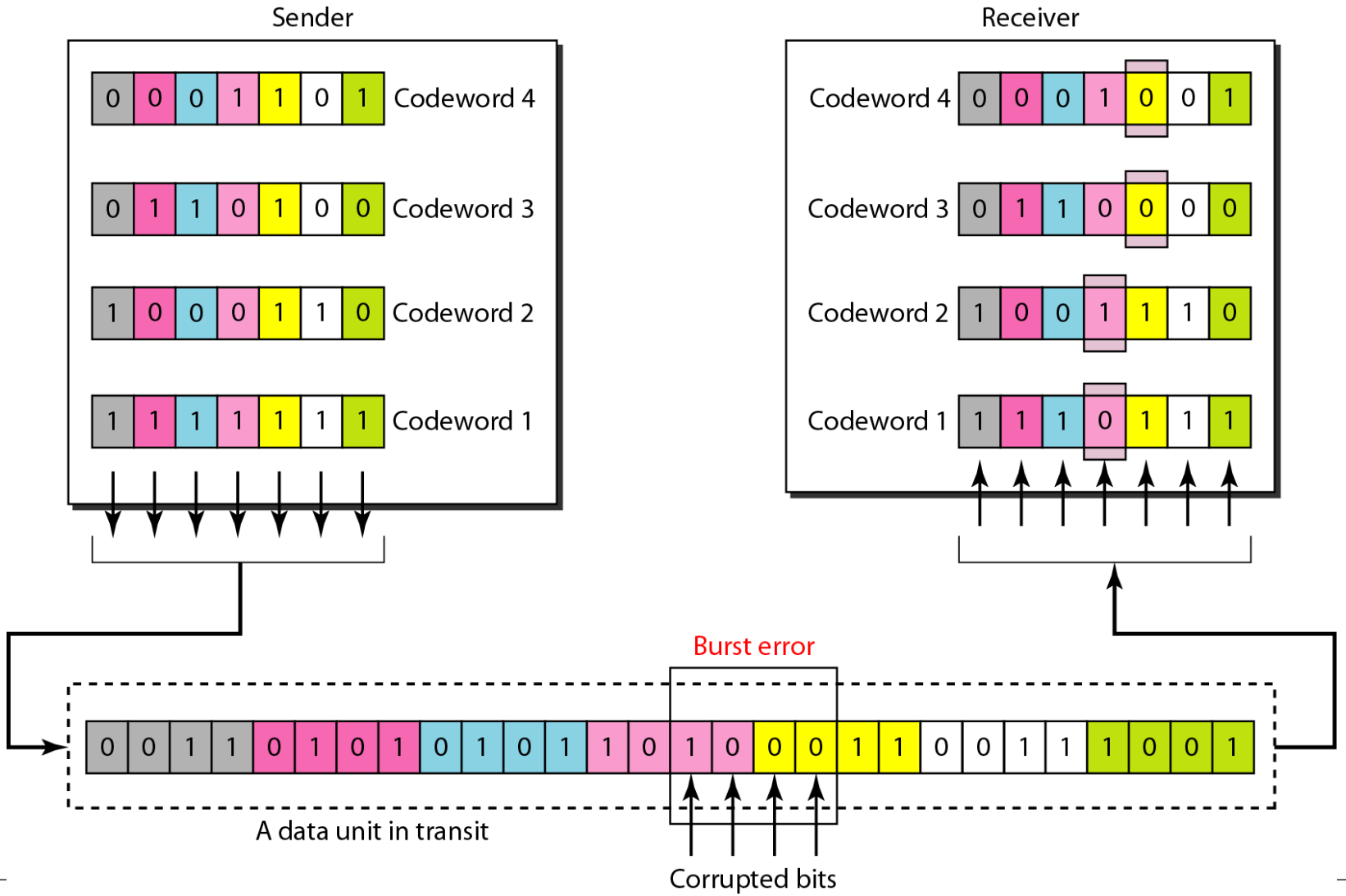
**Code:**
**1 0 0 1 1 1 0 0 1 0 1**

# Example: *Correcting Error*

- Receiver receives 10010100101



# Burst Error Correction - Interleaving



# Reading

- B. A. Forouzan, "Data Communications and Networking," 5th Edition, McGraw-Hill 2013 (Chapter 10)

- William Stallings, "Data and Computer Communications," 10th Edition, Pearson 2015 (Chapter 6)