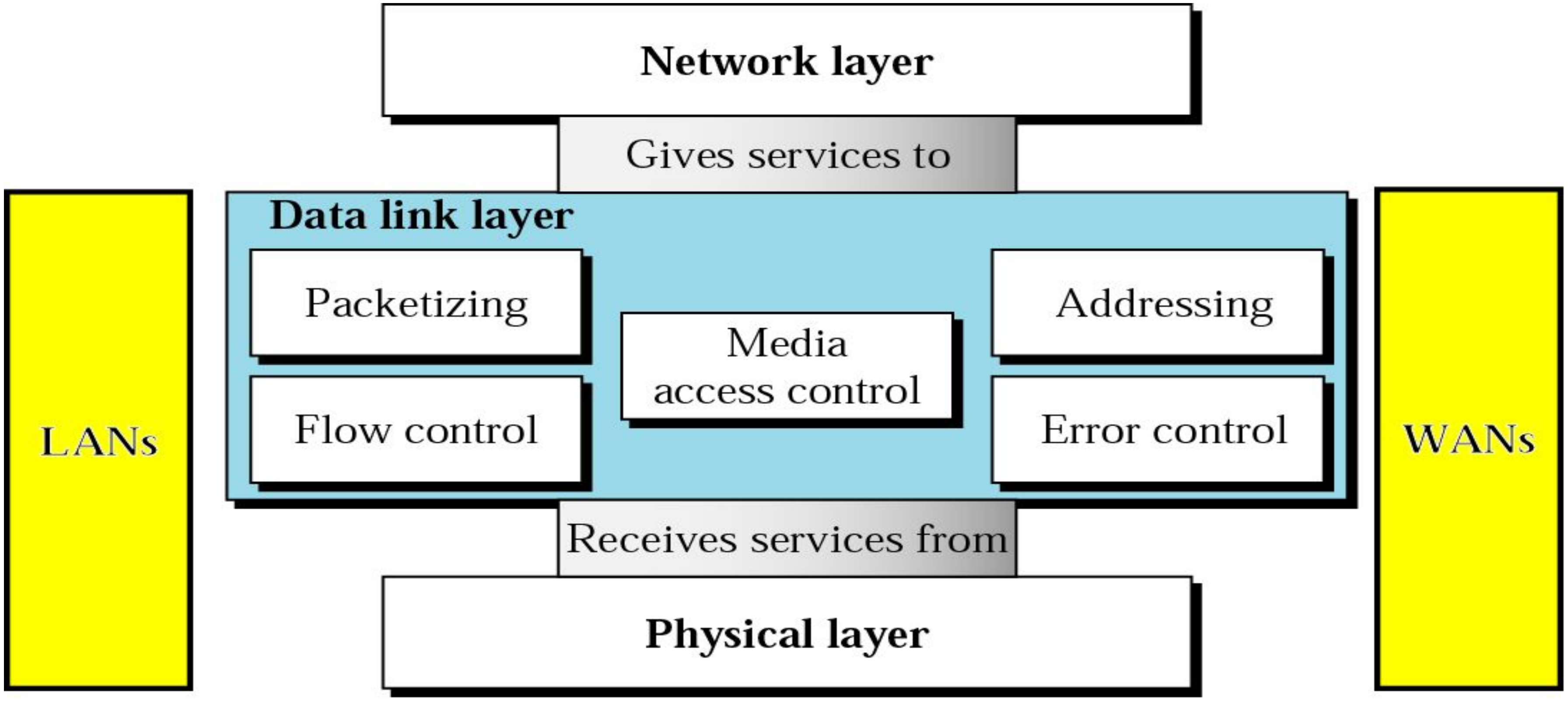
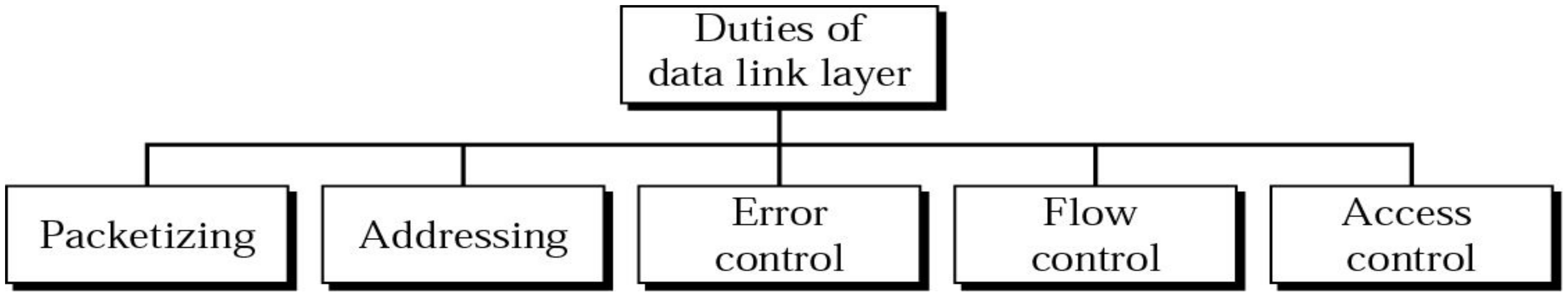
PART III

Data Link Layer

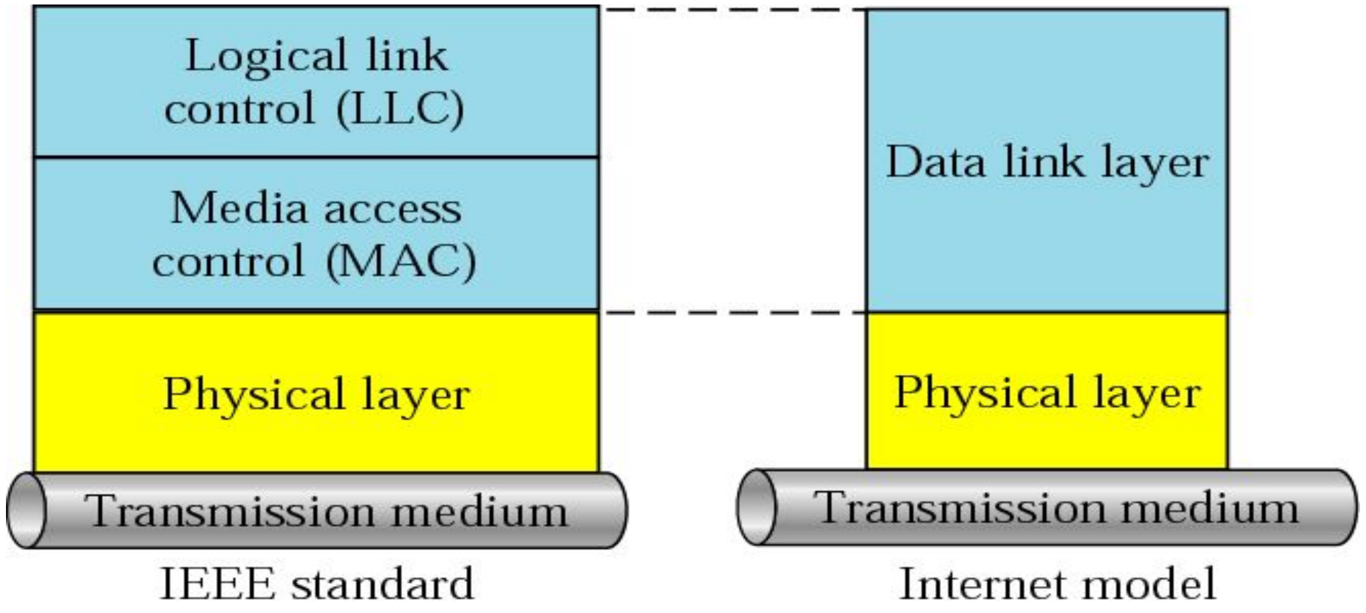
Position of the data-link layer



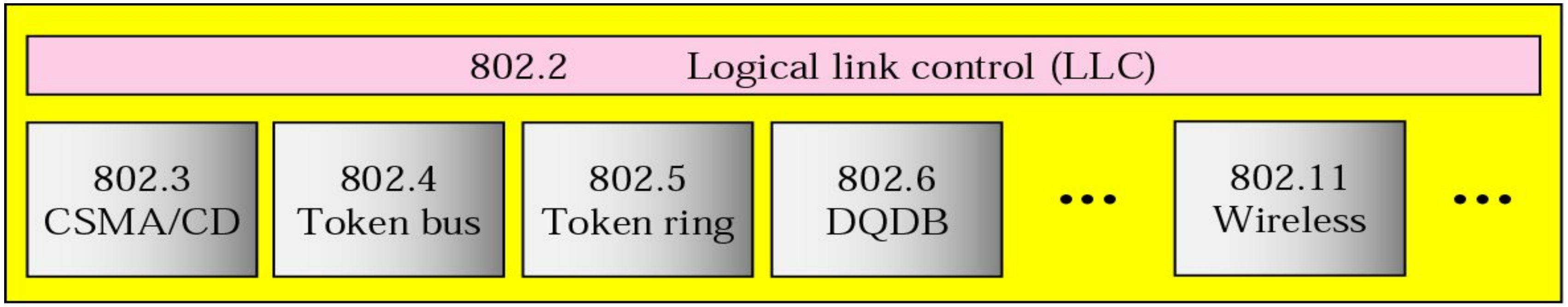
Data link layer duties



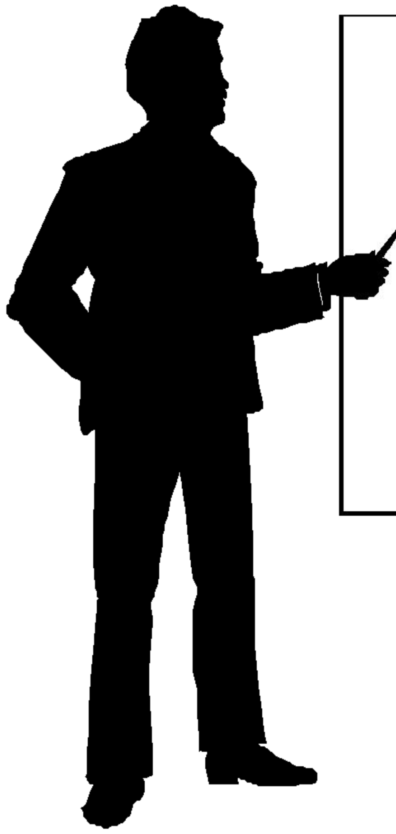
LLC and MAC sublayers



IEEE standards for LANs



Project 802



Error Detection and Correction

10 Error Detection and Correction

10.1 Types of Errors

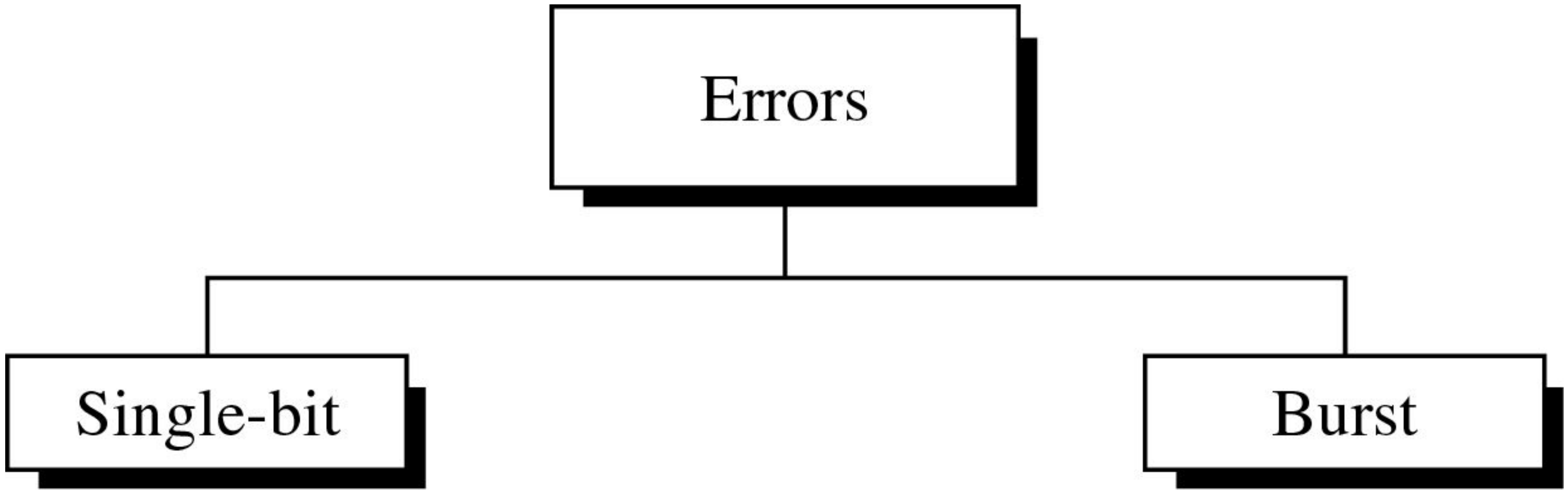
10.2 Detection

10.3 Error Correction

Error Detection and Correction

- Data can be corrupted during transmission. For reliable communication, error must be detected and corrected
- Error Detection and Correction are implemented either at the data link layer or the transport layer of the OSI model

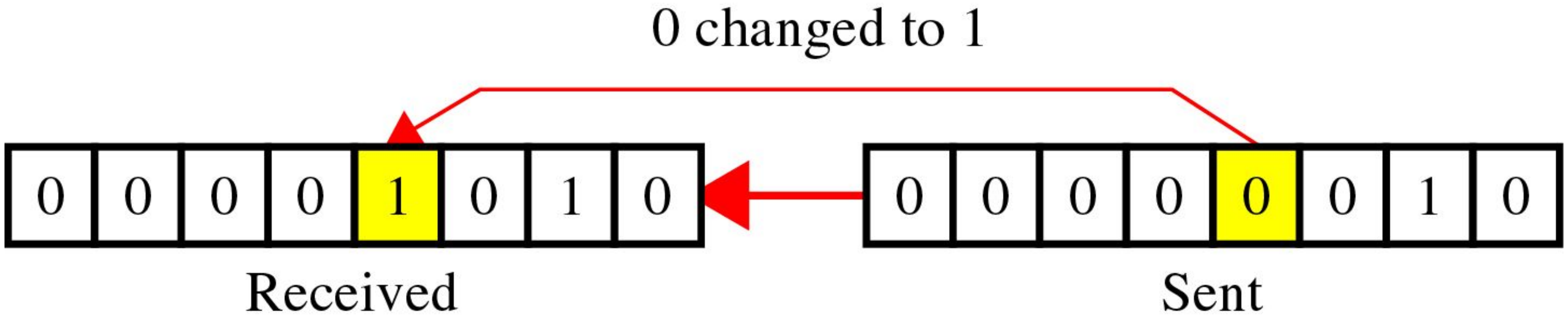
10.1 Type of Errors



Type of Errors(cont'd)

- Single-Bit Error

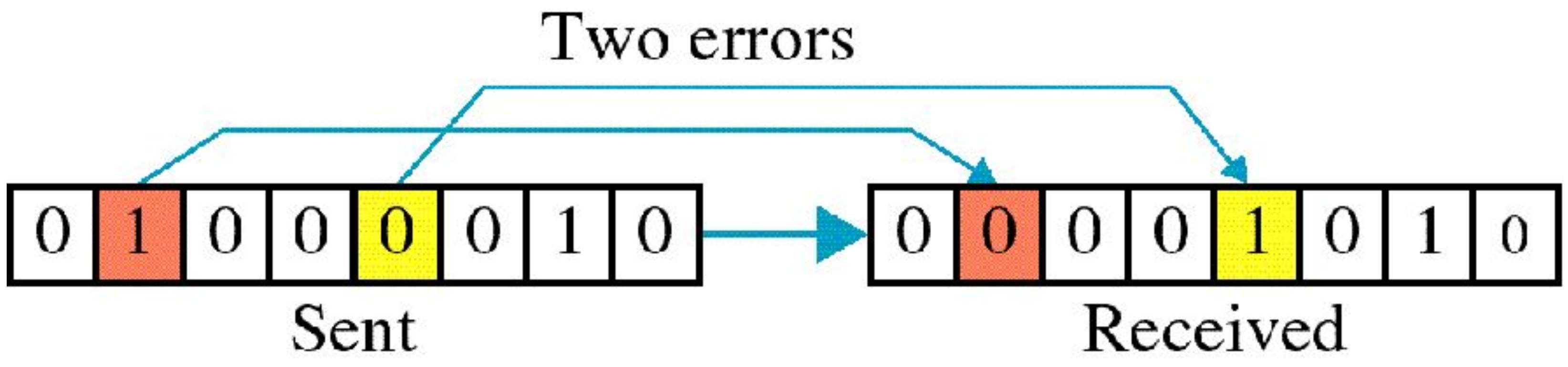
~ is when only one bit in the data unit has changed (ex : ASCII STX - ASCII LF)



Type of Errors(cont'd)

❑ **Multiple-Bit Error**

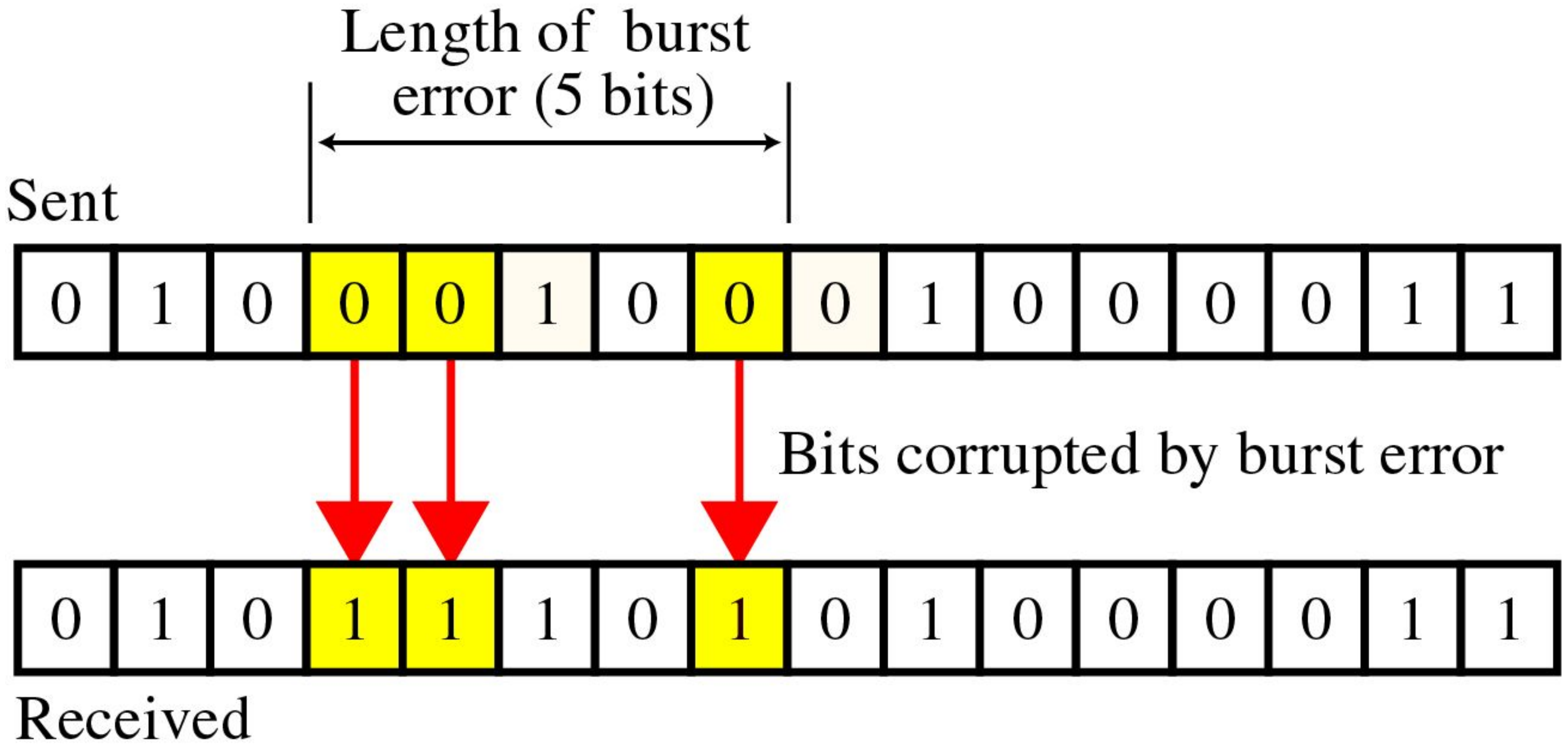
~ **is when two or more nonconsecutive bits in the data unit have changed(ex : ASCII B - ASCII LF)**



Type of Errors(cont'd)

- **Burst Error**

~ means that 2 or more consecutive bits in the data unit have changed

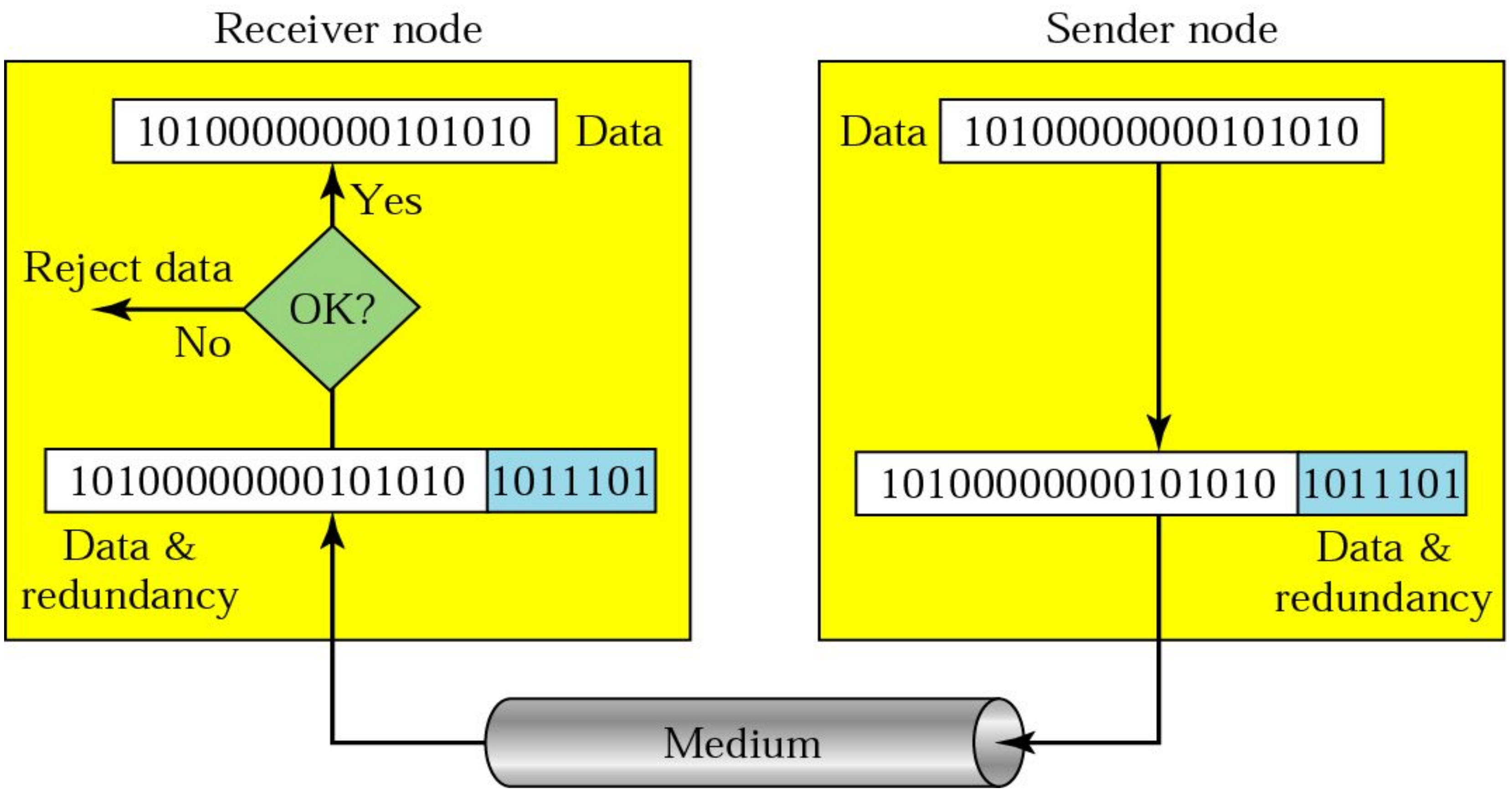


10.2 Detection

- Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination

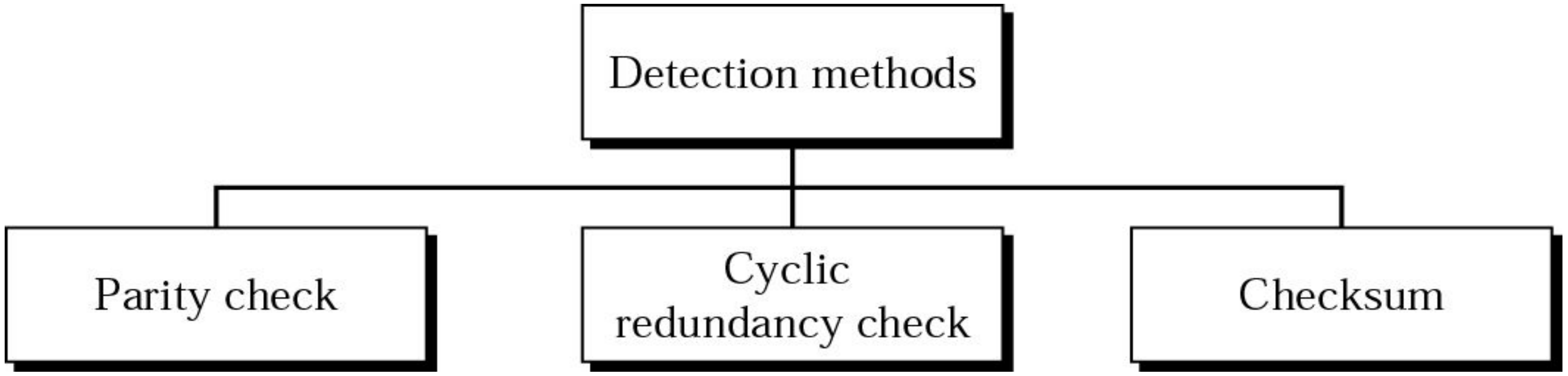
Detection(cont'd)

- Redundancy



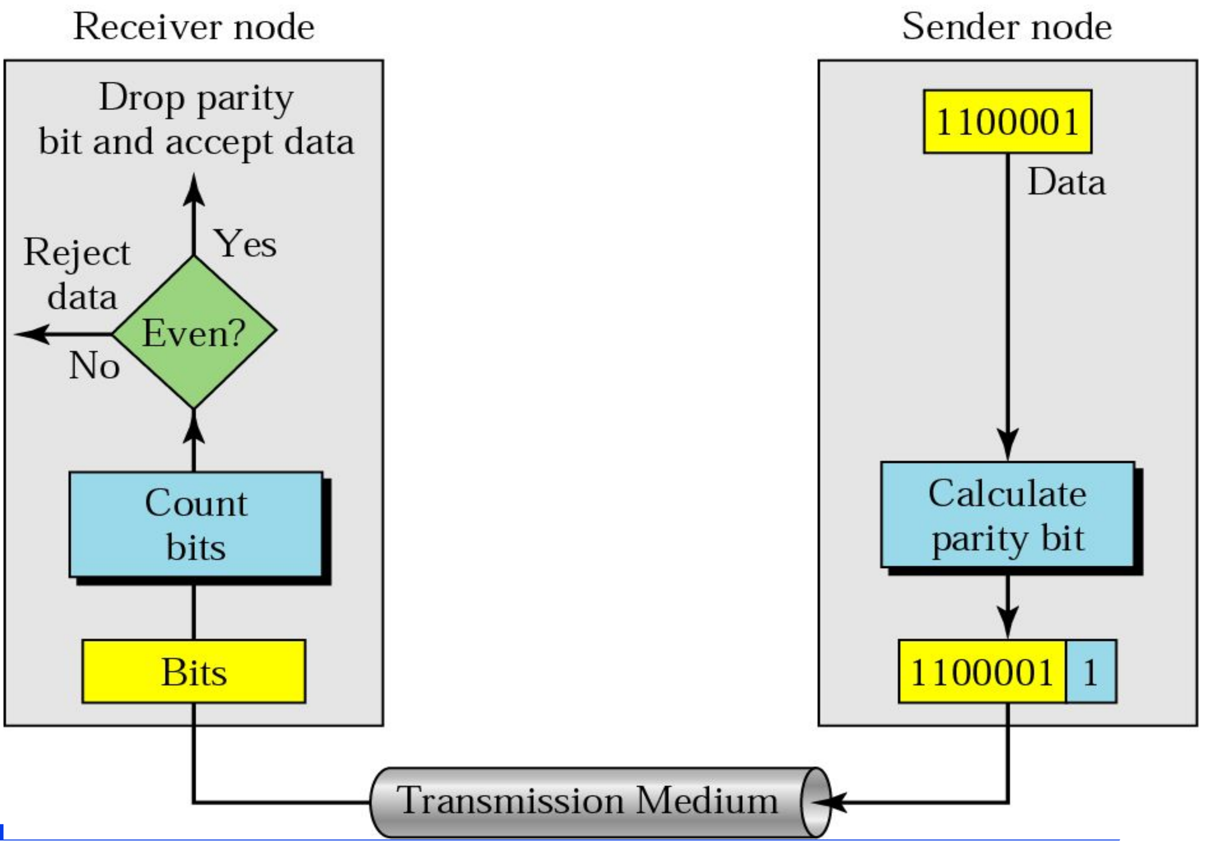
Detection(cont'd)

- Detection methods



Detection(cont'd)

- **Parity Check**
  - **A parity bit is added to every data unit so that the total number of 1s(including the parity bit) becomes even for even-parity check or odd for odd-parity check**
  - **Simple parity check**



Detection -examples

Example 1

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**

Detection – examples

Example 2

Now suppose the word world in Example 1 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.

Detection – examples

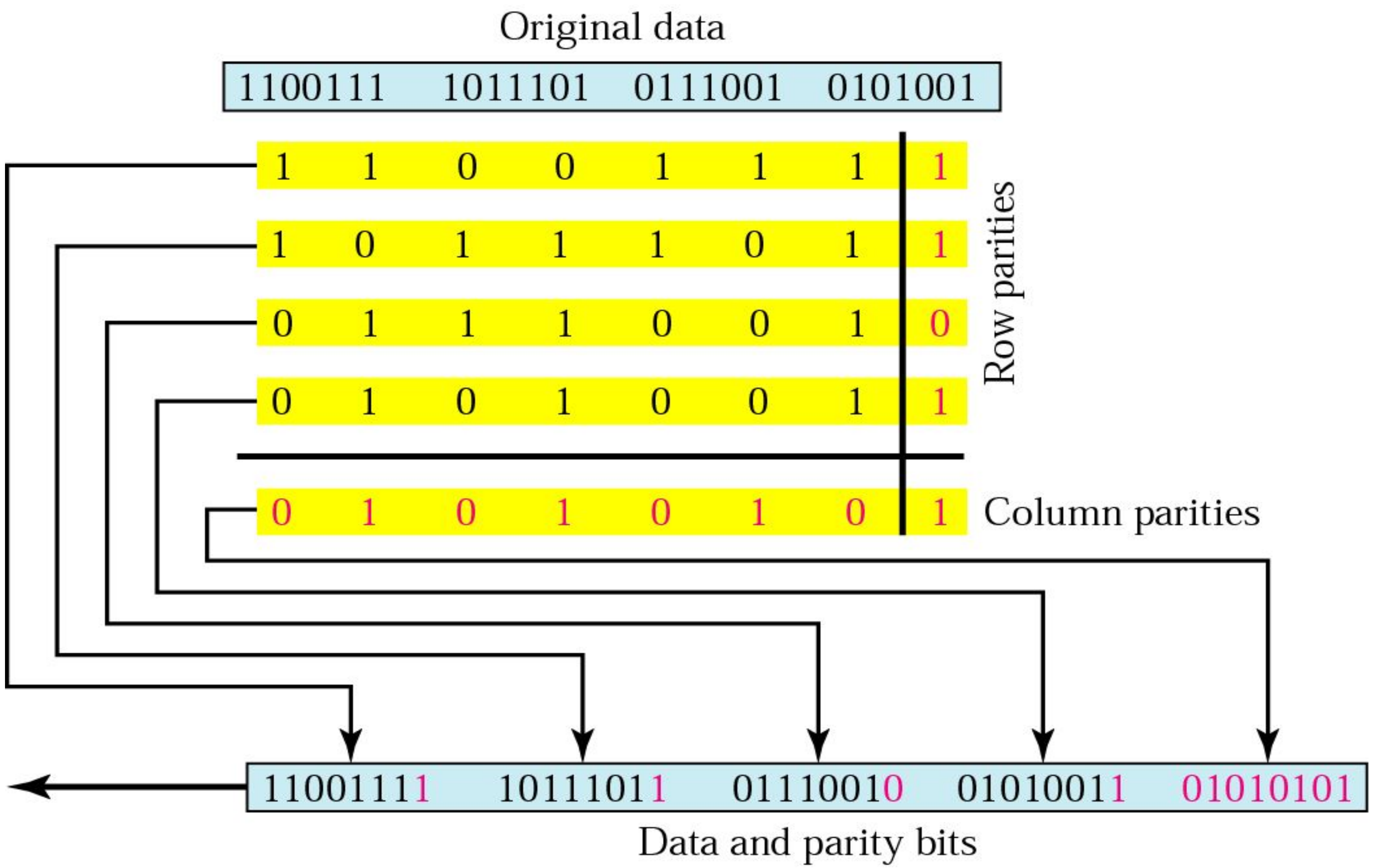
Example 3

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.

Two –Dimensional Parity Check



Detection - example

Example 4

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

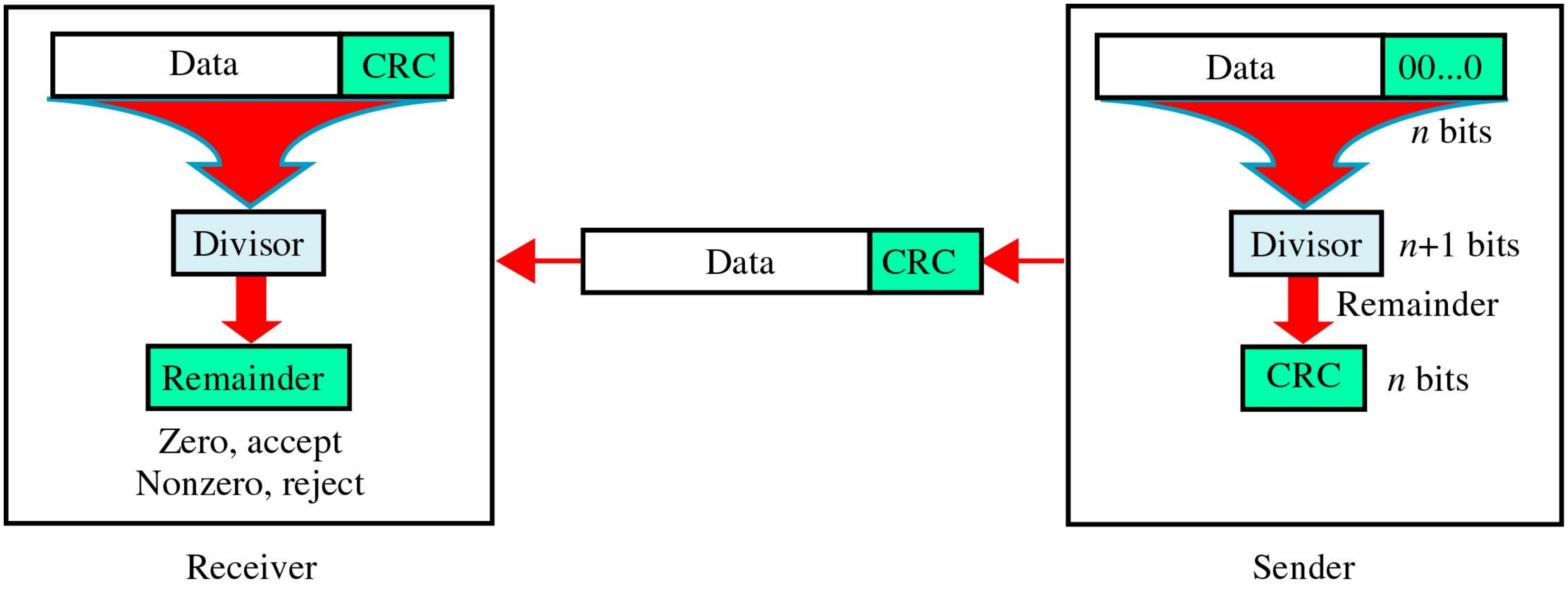
When the receiver checks the parity bits, some of the bits do not follow the even-parity rule and the whole block is discarded.

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

Detection(cont'd)

- **CRC(Cyclic Redundancy Check)**

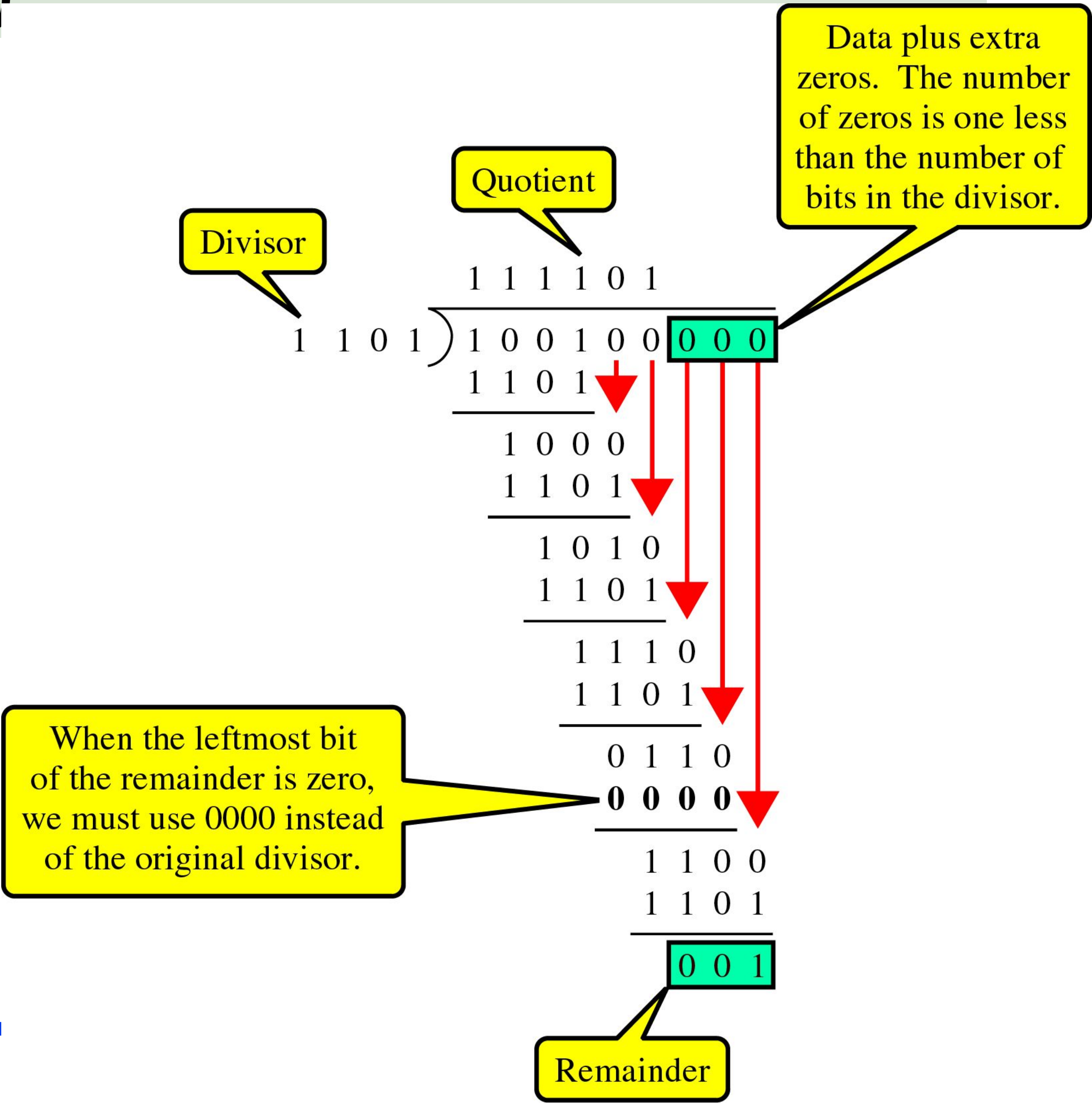
~ is based on binary division.



Detection(cont'd)

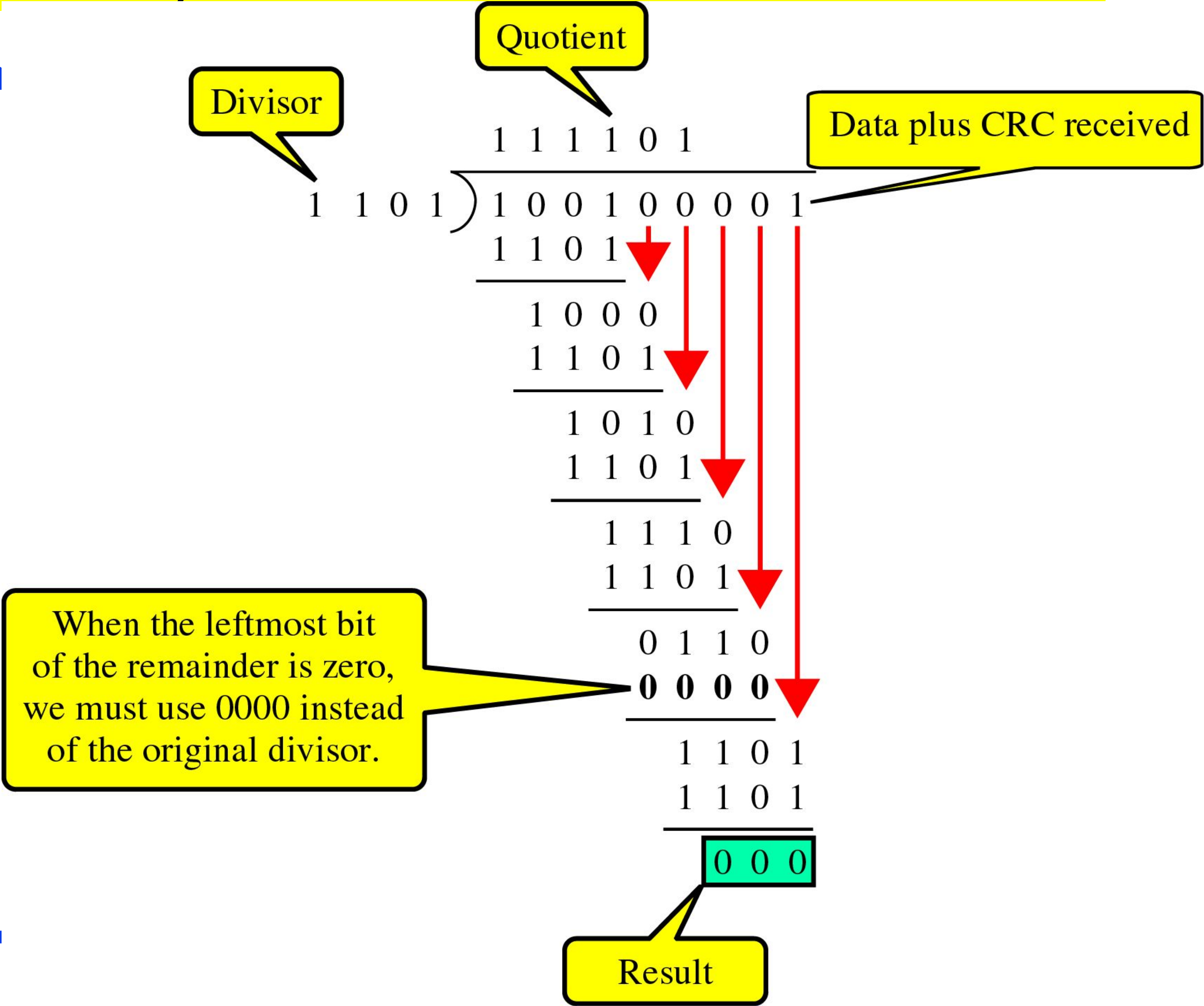
- CRC generator
  - ~ uses modular-2 division.

Binary Division in a CRC Generator


Data plus extra zeros. The number of zeros is one less than the number of bits in the divisor.
Quotient
Divisor
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 0
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 0
1 1 0 1
0 0 1
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Remainder

Detection(cont'd)

Binary Division in a CRC Checker



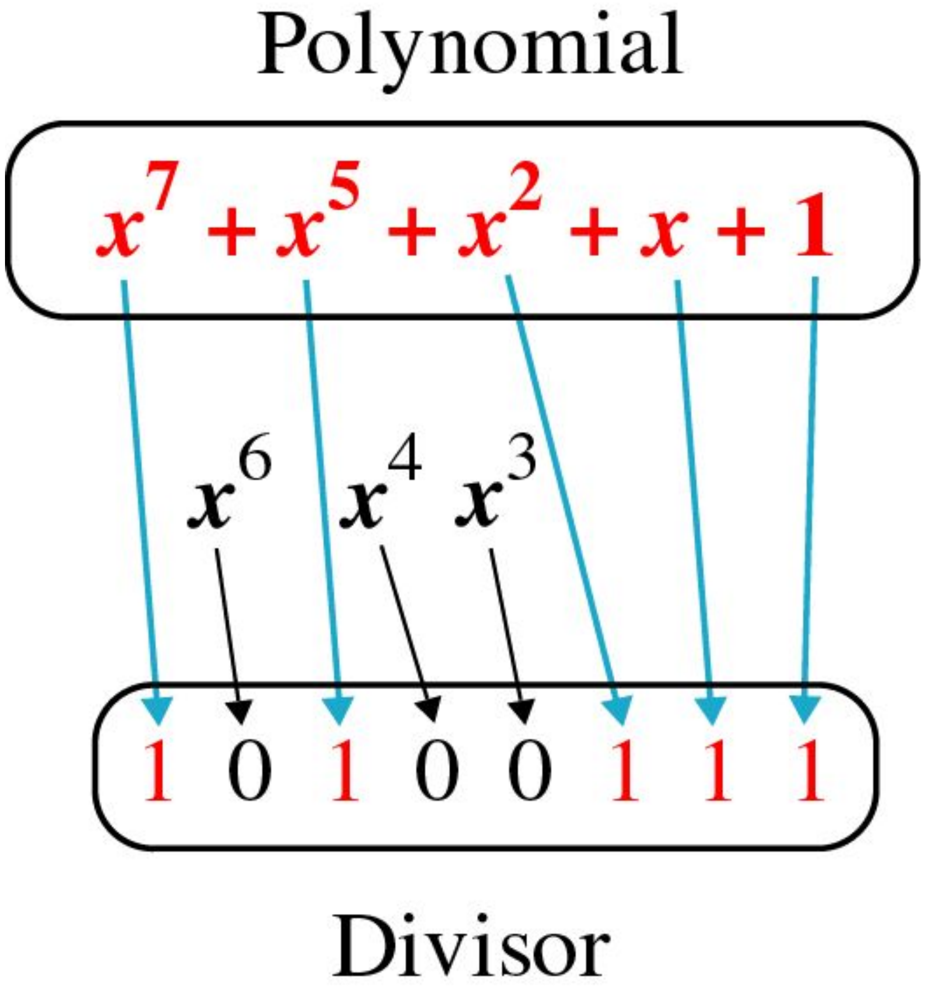
Detection(cont'd)

- **Polynomials**
  - **CRC generator(divisor) is most often represented not as a string of 1s and 0s, but as an algebraic polynomial.**

$$\boxed{x^7 + x^5 + x^2 + x + 1}$$

Detection(cont'd)

- A polynomial representing a divisor



Detection(cont'd)

- Standard polynomials

CRC-12

$$\boxed{x^{12} + x^{11} + x^{3} + x + 1}$$

CRC-16

$$\boxed{x^{16} + x^{15} + x^{2} + 1}$$

CRC-ITU-T

$$\boxed{x^{16} + x^{12} + x^{5} + 1}$$

CRC-32

$$\boxed{x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1}$$

Detection(cont'd)

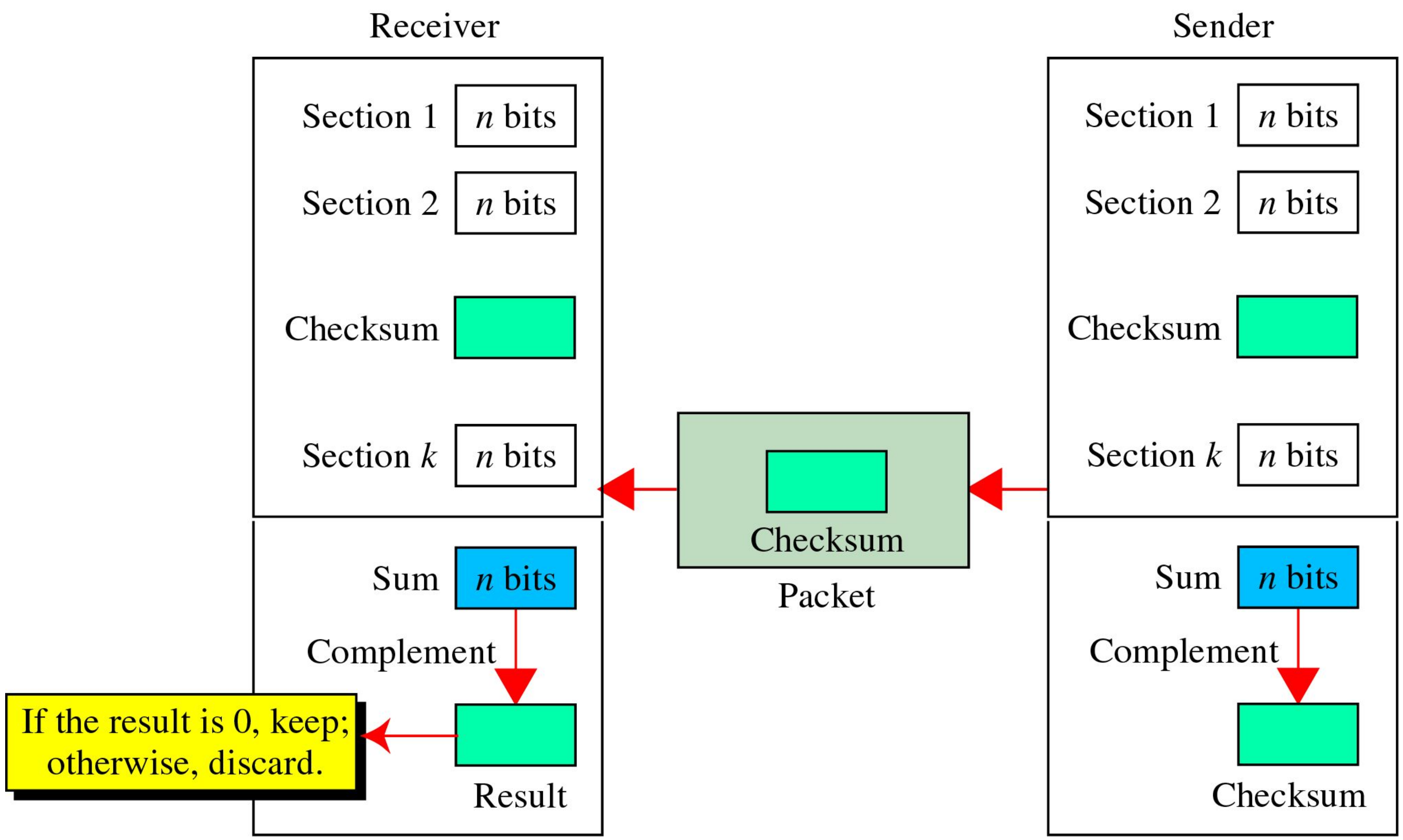
- Checksum

~ used by the higher layer protocols

~ is based on the concept of redundancy(VRC, LRC, CRC ….)

Detection(cont'd)

- Checksum Generator



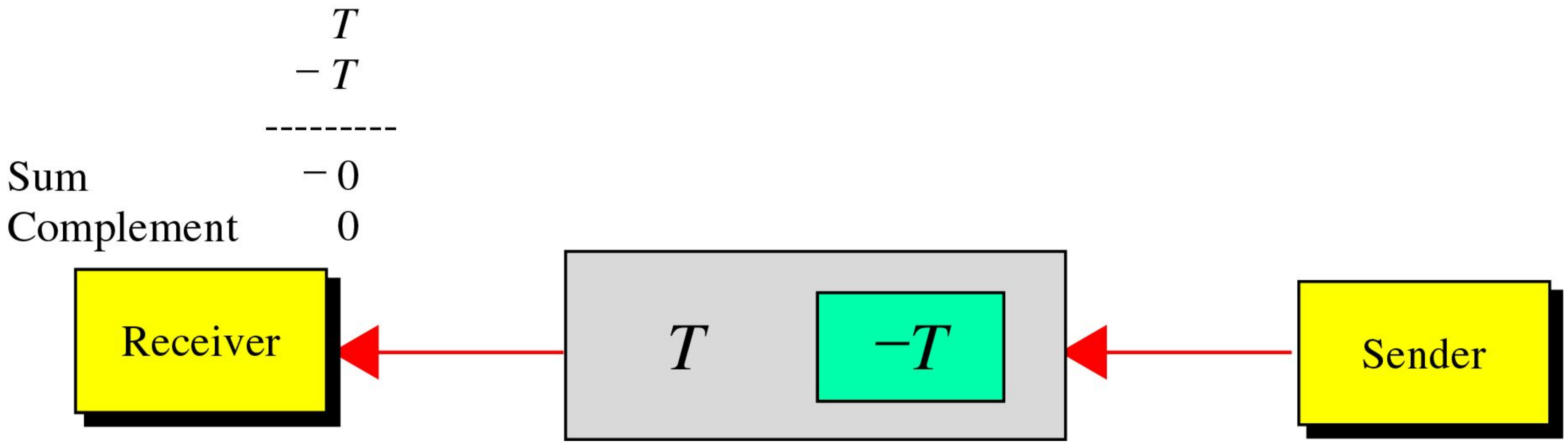
Detection(cont'd)

- To create the checksum the sender does the following:
  - The unit is divided into K sections, each of n bits.
  - Section 1 and 2 are added together using one's complement.
  - Section 3 is added to the result of the previous step.
  - Section 4 is added to the result of the previous step.
  - The process repeats until section k is added to the result of the previous step.
  - The final result is complemented to make the checksum.

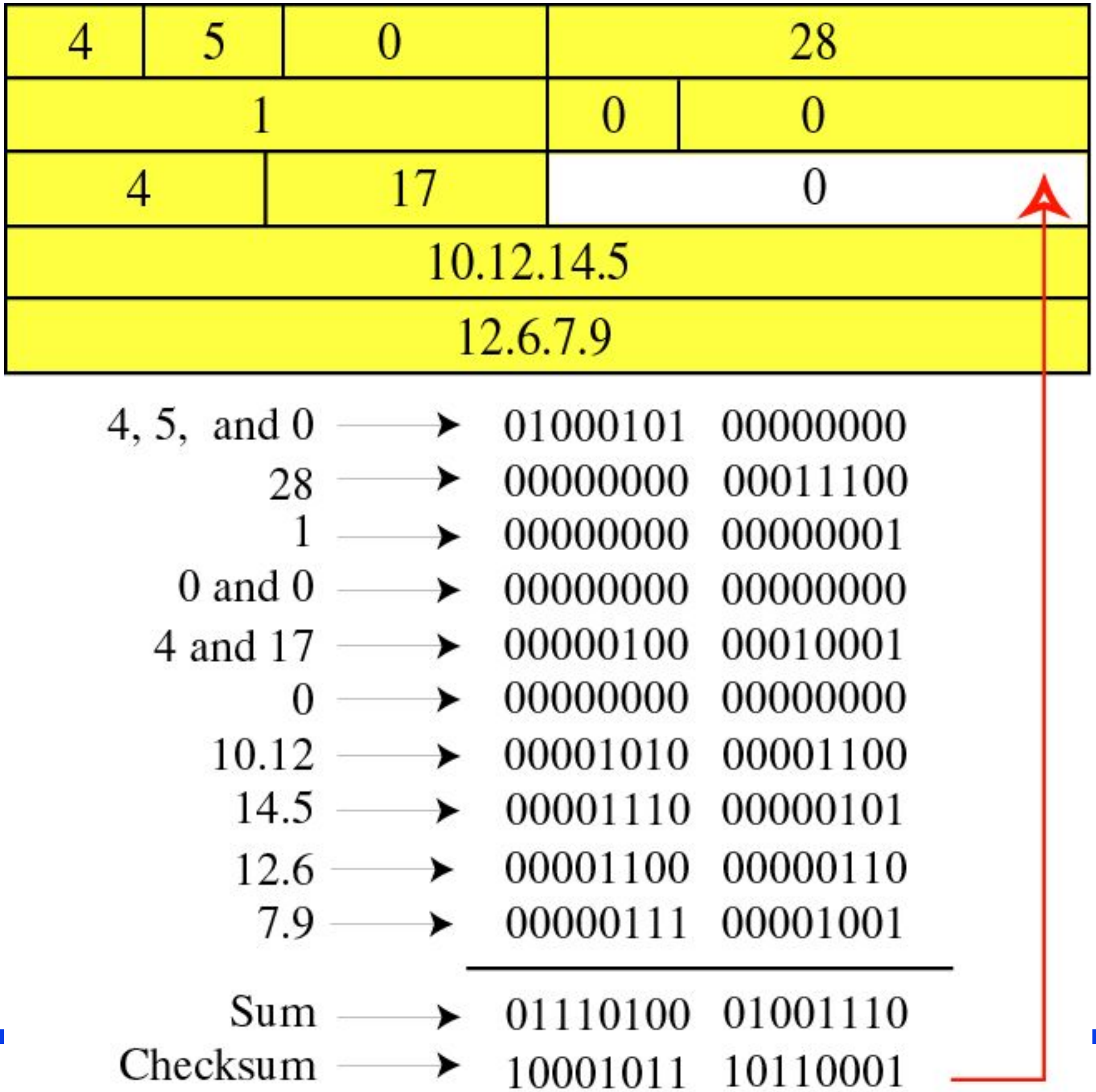
Detection(cont'd)

- data unit and checksum

The receiver adds the data unit and the checksum field. If the result is all 1s, the data unit is accepted; otherwise it is discarded.



Detection(cont'd)



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

Detection(cont'd)

(at a sender)

Original data : 10101001 00111001

10101001

00111001

11100010Sum

00011101Checksum

10101001 00111001 00011101

Detection(cont’d)

- Example (at a receiver)

Received data : 10101001 00111001 00011101

```
10101001
 00111001
 00011101
---------------
11111111 □ Sum
00000000 □ Complement
```

10.3 Error Correction

~ can be handled in two ways

- when an error is discovered, the receiver can have the sender retransmit the entire data unit.

- a receiver can use an error-correcting code, which automatically corrects certain errors.

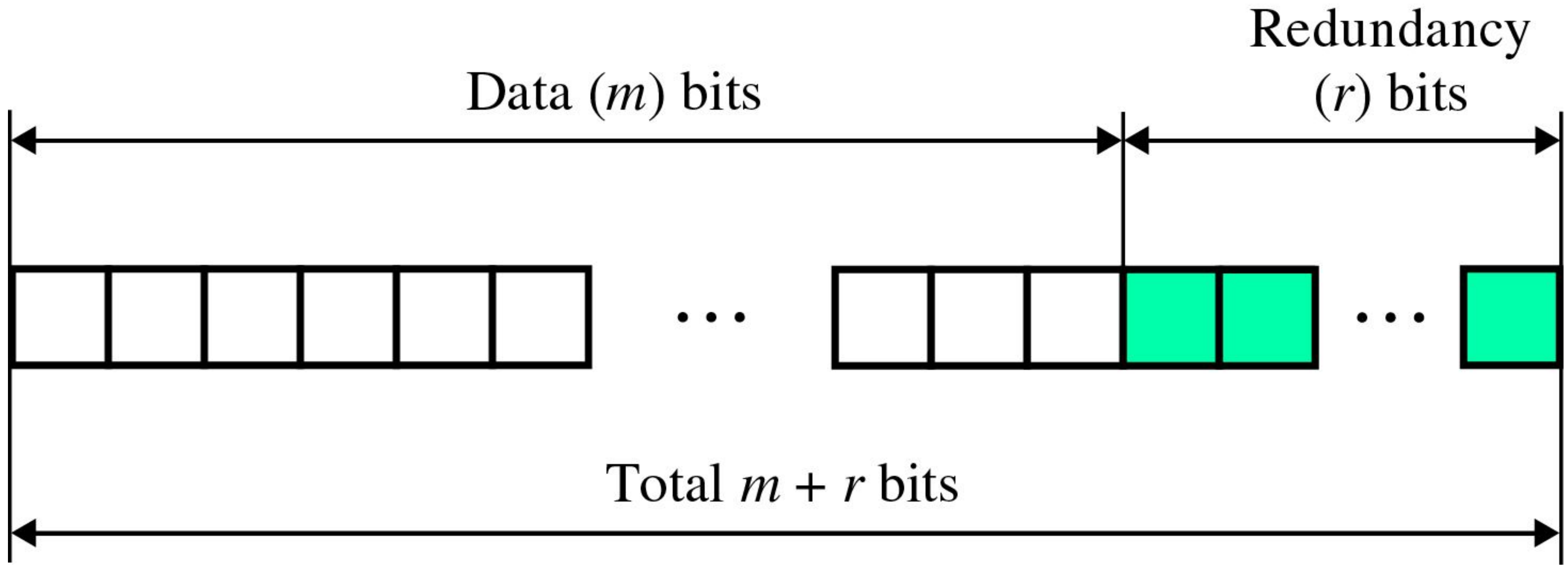
Error Correction(cont'd)

- **Single-Bit Error Correction**
  - **parity bit**
  - **The secret of error correction is to locate the invalid bit or bits**
  - **For ASCII code, it needs a three-bit redundancy code(000-111)**

Error Correction(cont'd)

- **Redundancy Bits**

~ to calculate the number of redundancy bits (R) required to correct a given number of data bit (M)



Error Correction(cont'd)

- If the total number of bits in a transmittable unit is m+r, then r must be able to indicate at least m+r+1 different states

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

ex) For value of m is 7(ASCII), the smallest r value that can satisfy this equation is 4

$$2^4 \geq 7 + 4 + 1$$

Error Correction(cont'd)

- Relationship between data and redundancy bits

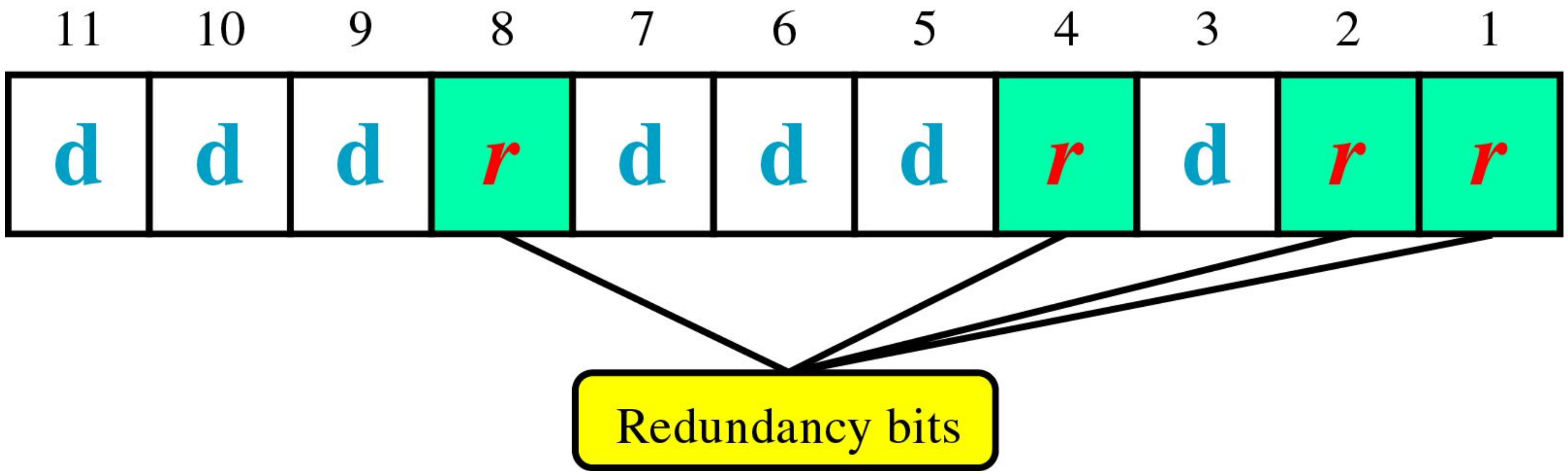
Number of Data Bits (m)	Number of Redundancy Bits (r)	Total Bits (m+r)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Error Correction(cont'd)

- Hamming Code

  ~ developed by R.W.Hamming

- positions of redundancy bits in Hamming code



Error Correction(cont'd)

- each r bit is the VRC bit for one combination of data bits

r_1 = bits 1, 3, 5, 7, 9, 11

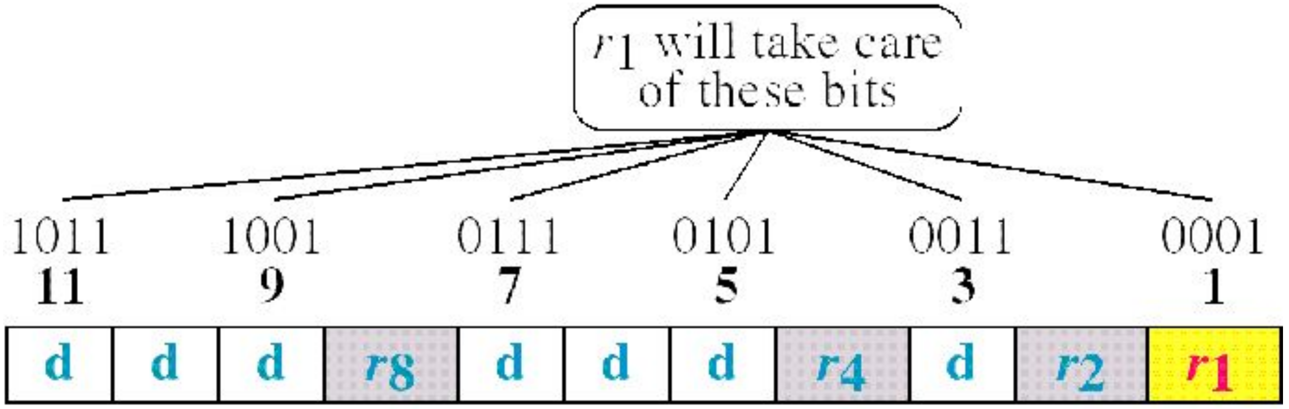
r_2 = bits 2, 3, 6, 7, 10, 11

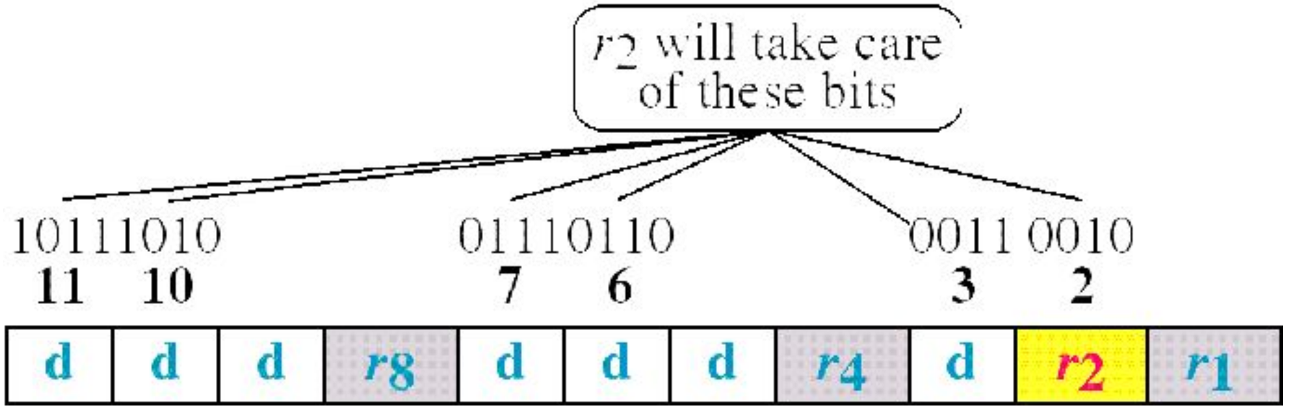
r_4 = bits 4, 5, 6, 7

r_8 = bits 8, 9, 10, 11

Error Correction(cont'd)

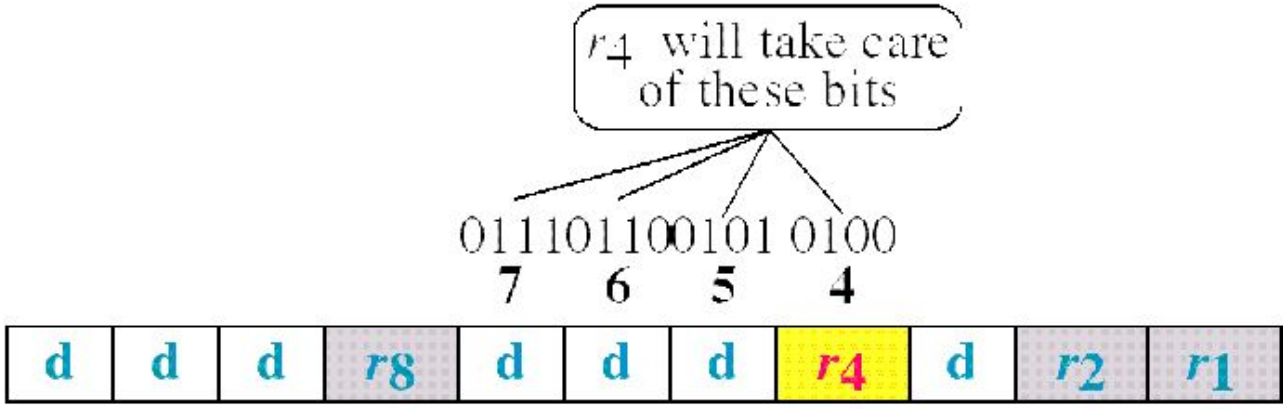
- Redundancy bits calculation(cont'd)

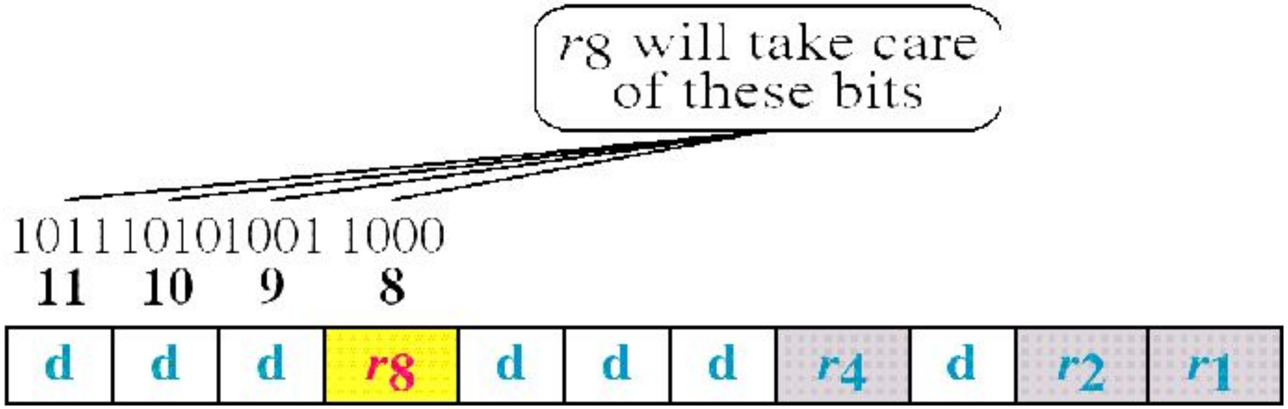




Error Correction(cont'd)

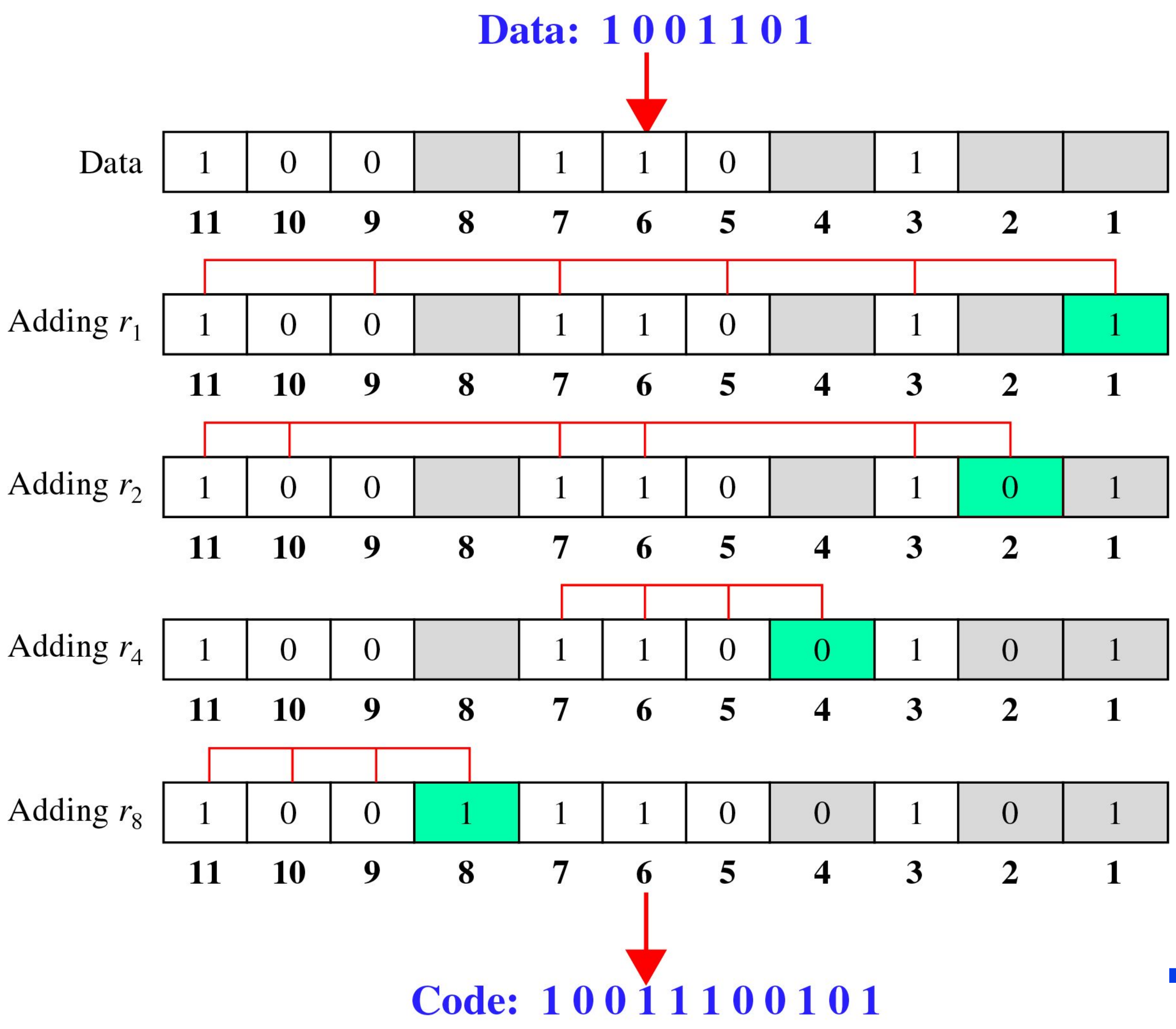
- Redundancy bits calculation





Error Correction(cont'd)

- Calculating the r values

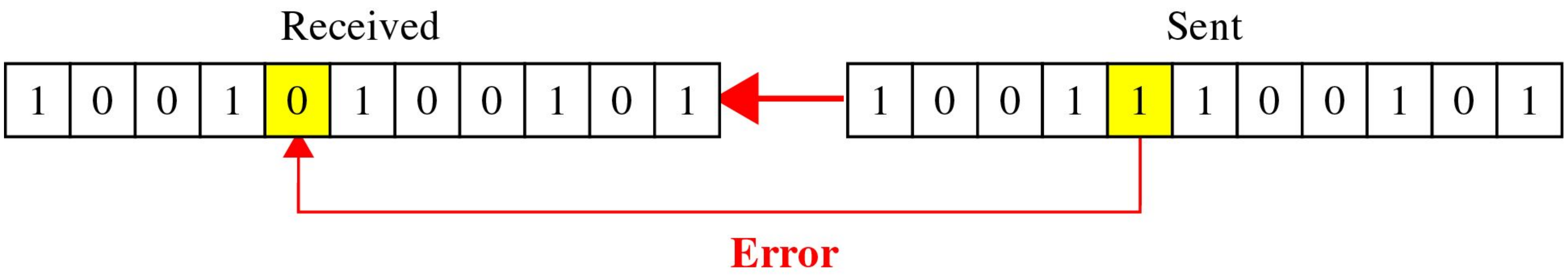


	11	10	9	8	7	6	5	4	3	2	1
Data	1	0	0		1	1	0		1		
Adding r_1	1	0	0		1	1	0		1		1
Adding r_2	1	0	0		1	1	0		1	0	1
Adding r_4	1	0	0		1	1	0	0	1	0	1
Adding r_8	1	0	0	1	1	1	0	0	1	0	1

Calculating Even Parity

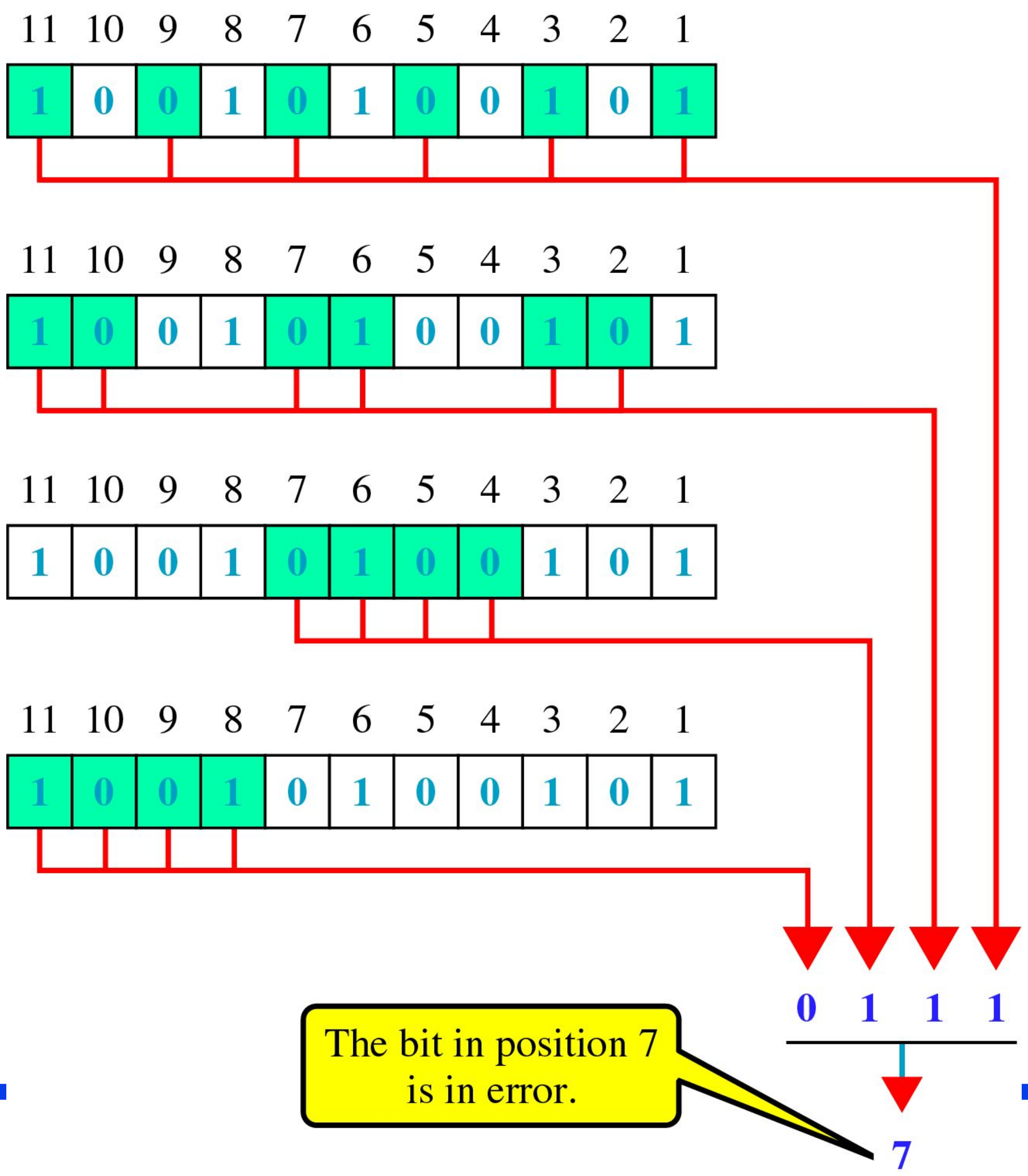
Error Correction(cont'd)

- Error Detection and Correction



Error Correction(cont'd)

- Error detection using Hamming Code



Error Correction(cont'd)

- **Multiple-Bit Error Correction**
  - **redundancy bits calculated on overlapping sets of data units can also be used to correct multiple-bit errors.**

  Ex) to correct double-bit errors, we must take into consideration that two bits can be a combination of any two bits in the entire sequence