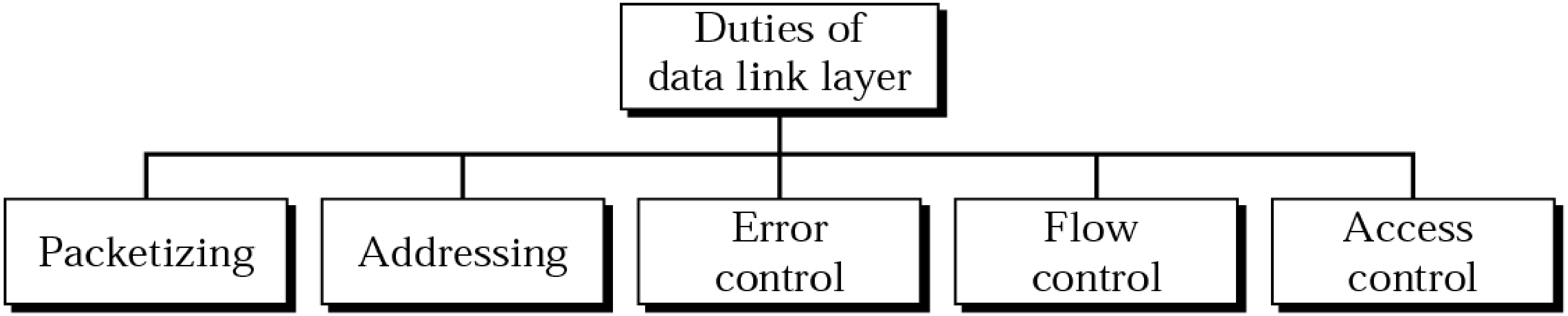
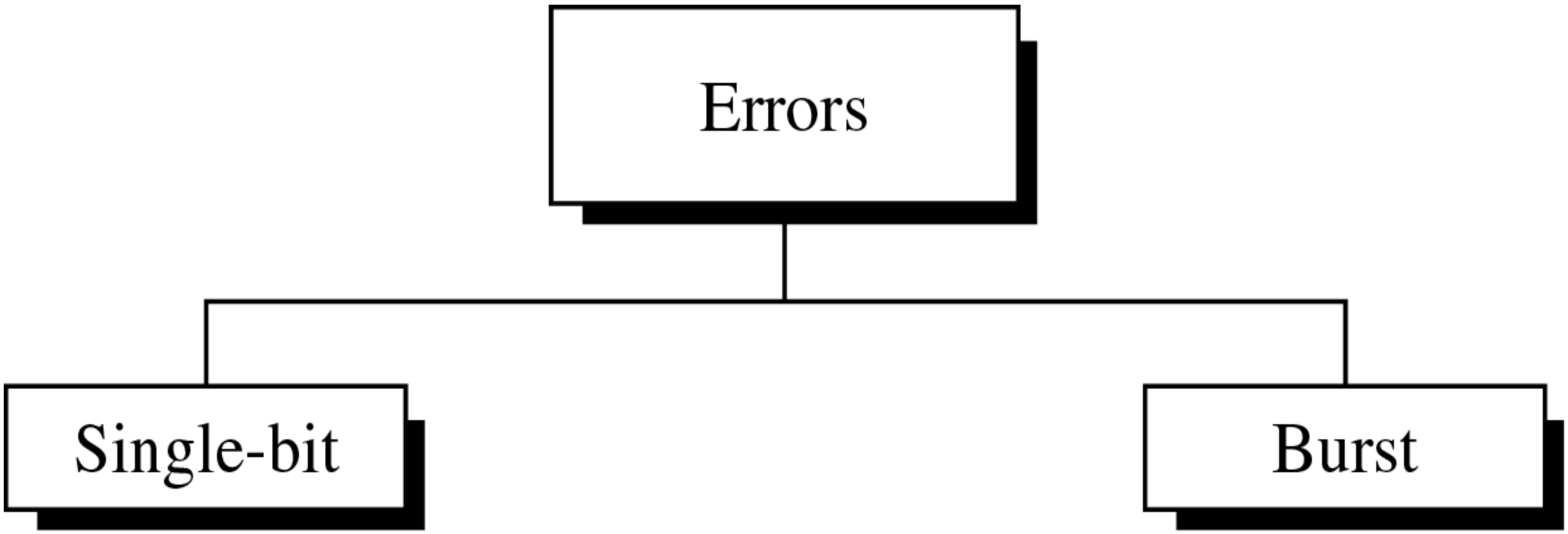
Data Link Layer


Duties of data link layer
Packetizing
Addressing
Error control
Flow control
Access control

Error Detection and Correction

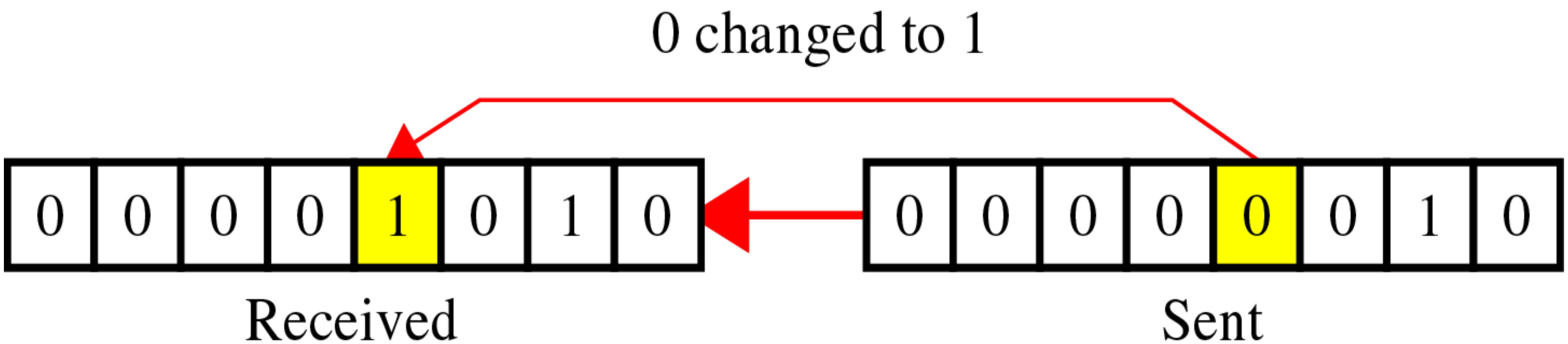
- Data can be corrupted during transmission.
- For reliable communication, error must be detected and corrected
- Error Detection and Correction are implemented either at the data link layer or the transport layer of the OSI model

Type of Errors



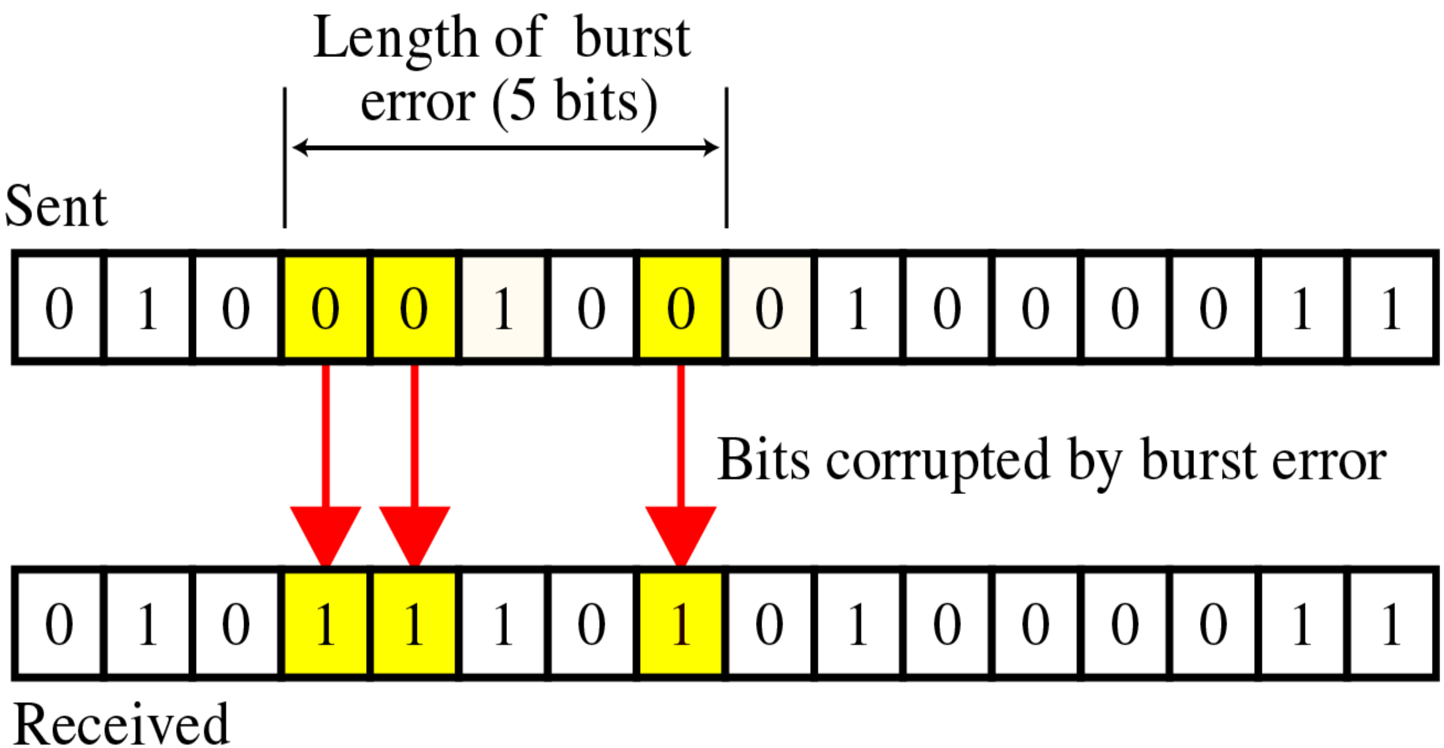
Single-Bit

~ is when only one bit in the data unit has changed



Burst Error

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

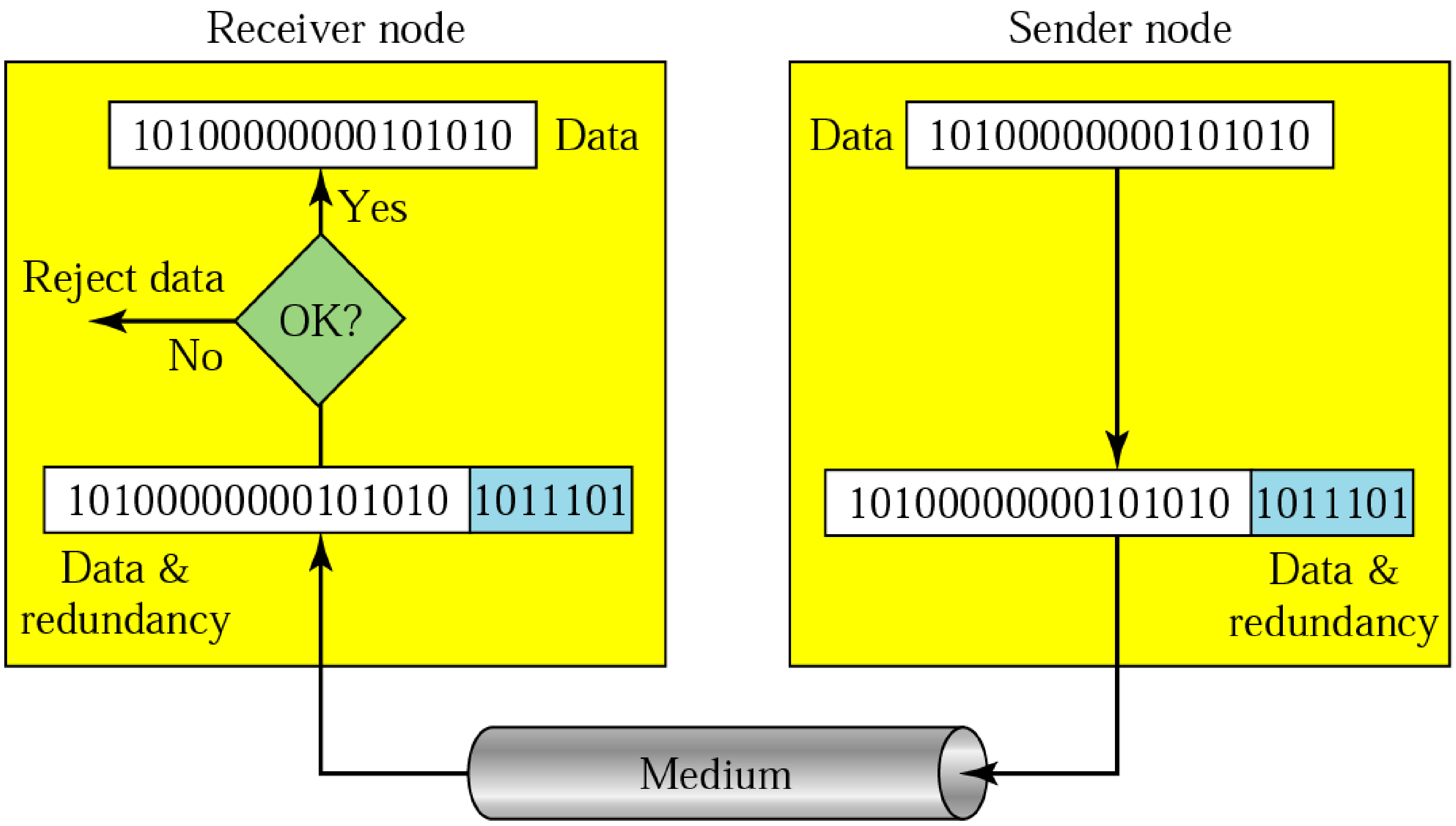


Detection

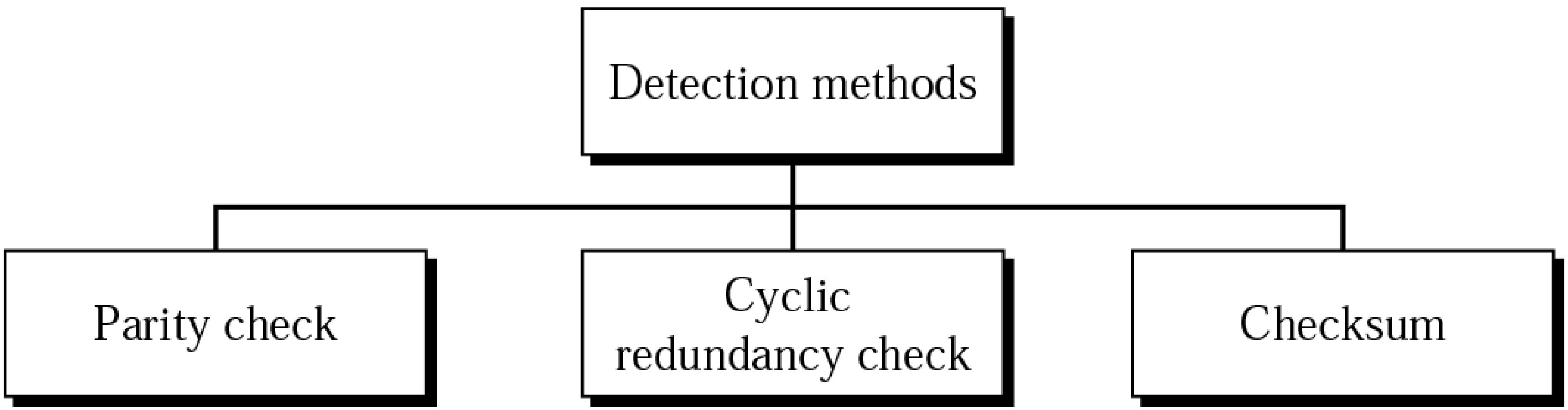
- Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination

Detection(cont'd)

- Redundancy

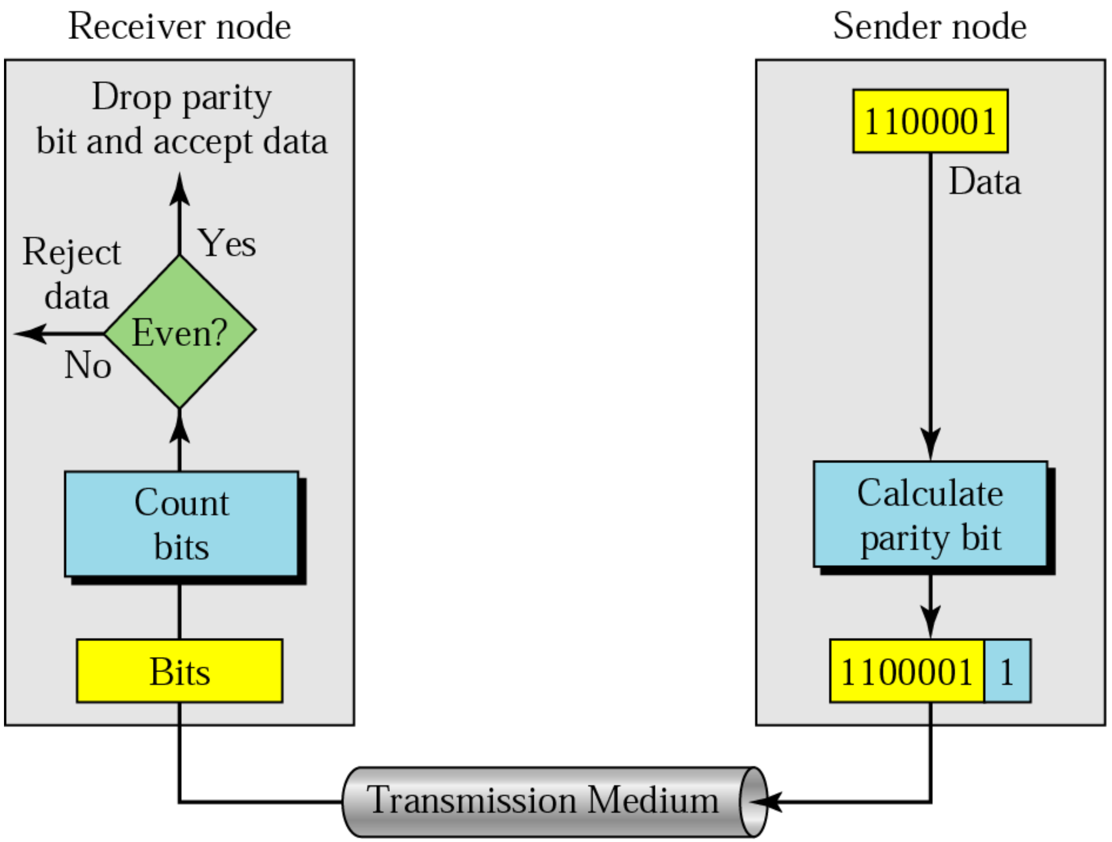


Error Detection Methods



Parity Check(Vertical Redundancy 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- 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 – 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.

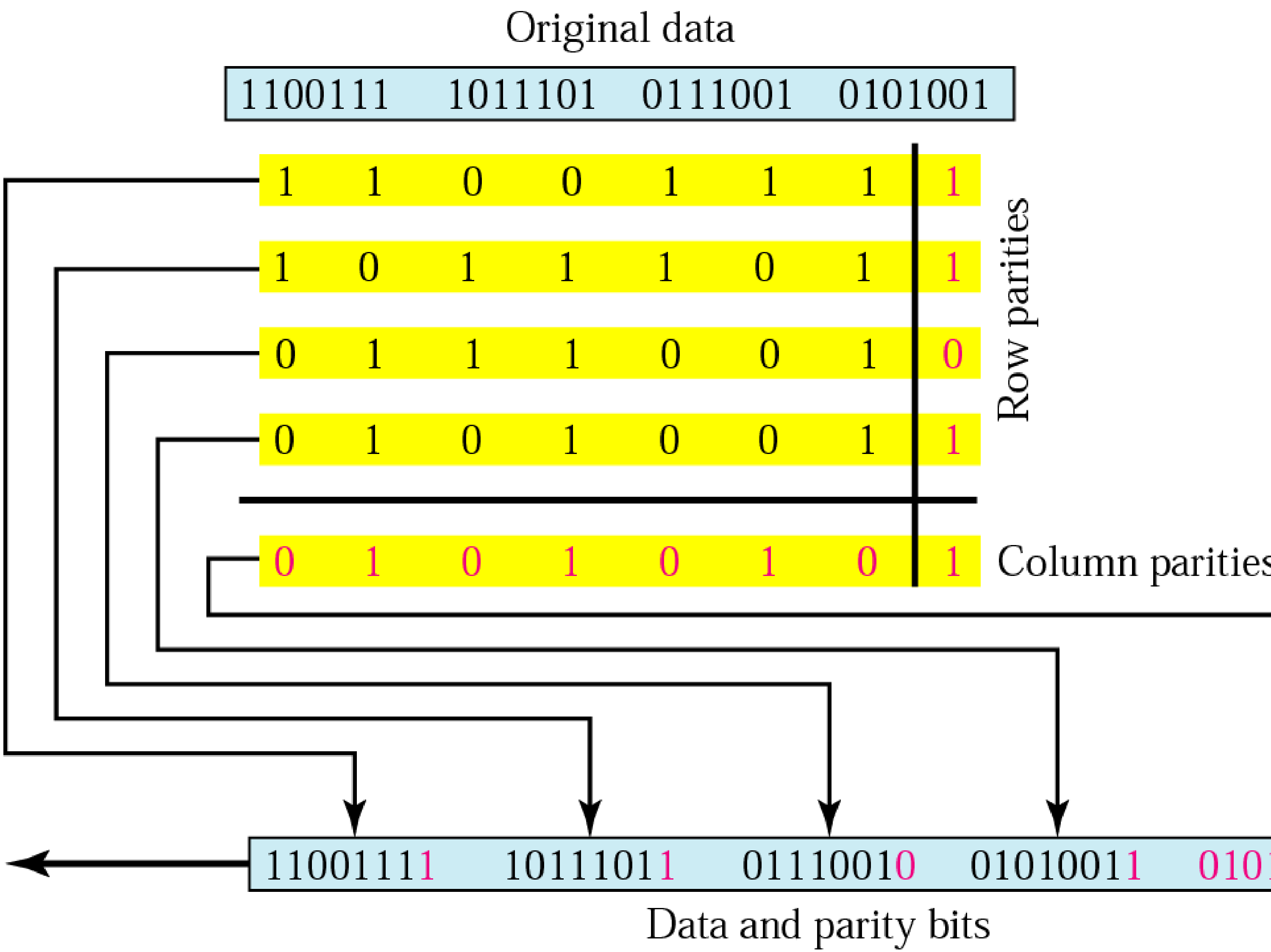
Detection – Example-3

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 (LRC) **Longitudinal Redundancy Check**



Detection – 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.

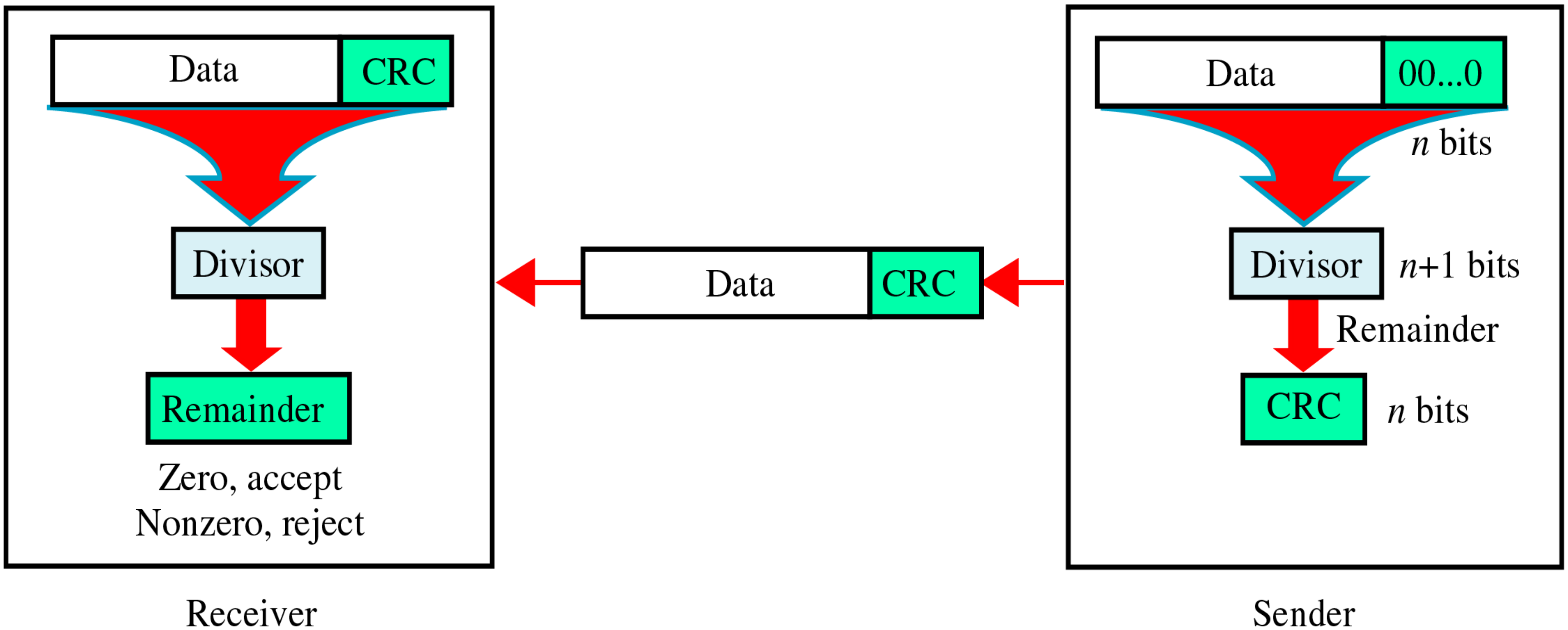
1010**0011** **1000**1001 11011101 11100111 10101010

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

CRC(Cyclic Redundancy Check)

~ is based on binary division.



CRC generation at the sender

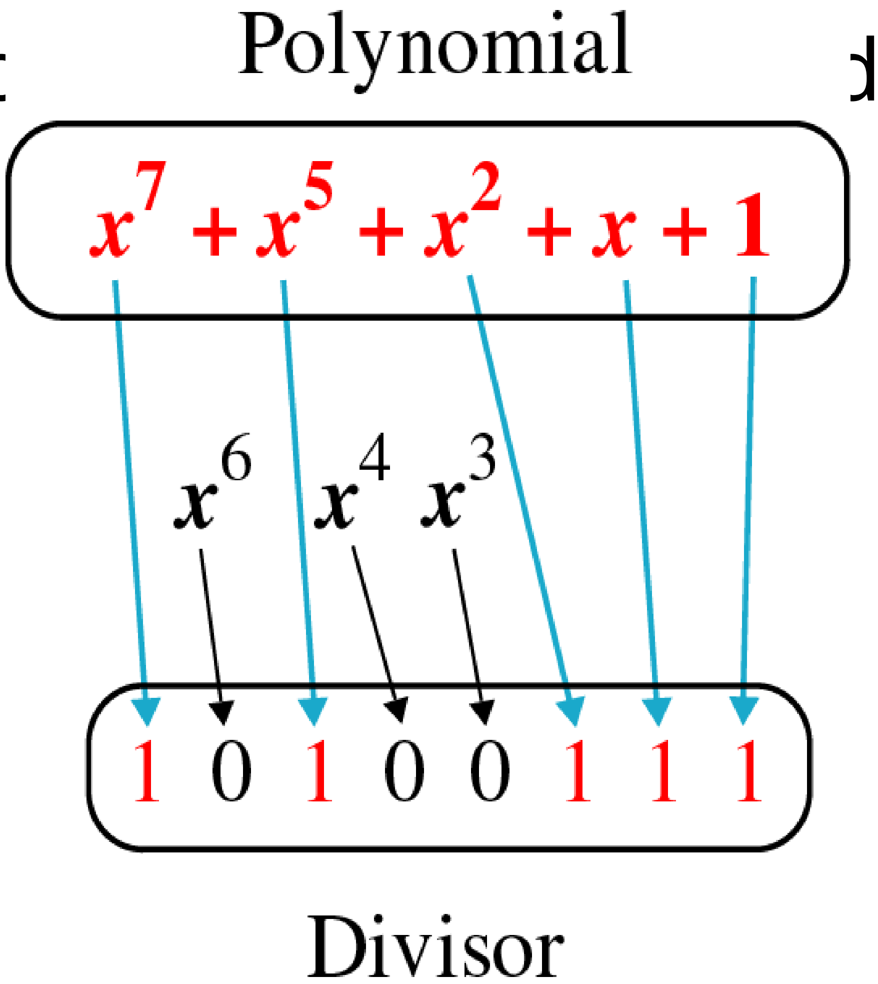
- Length of divisor ‘L’
- Append L-1 to original message
- Binary division operation
- Remainder is CRC, append to the data for transmission.
- CRC= L-1 bits.

Detection(cont'd)

- Polynomials
  - CRC generator(divisor) is most often represented not as a string of 1s and 0s,

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

Detection(cont'd)

- A polync


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

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

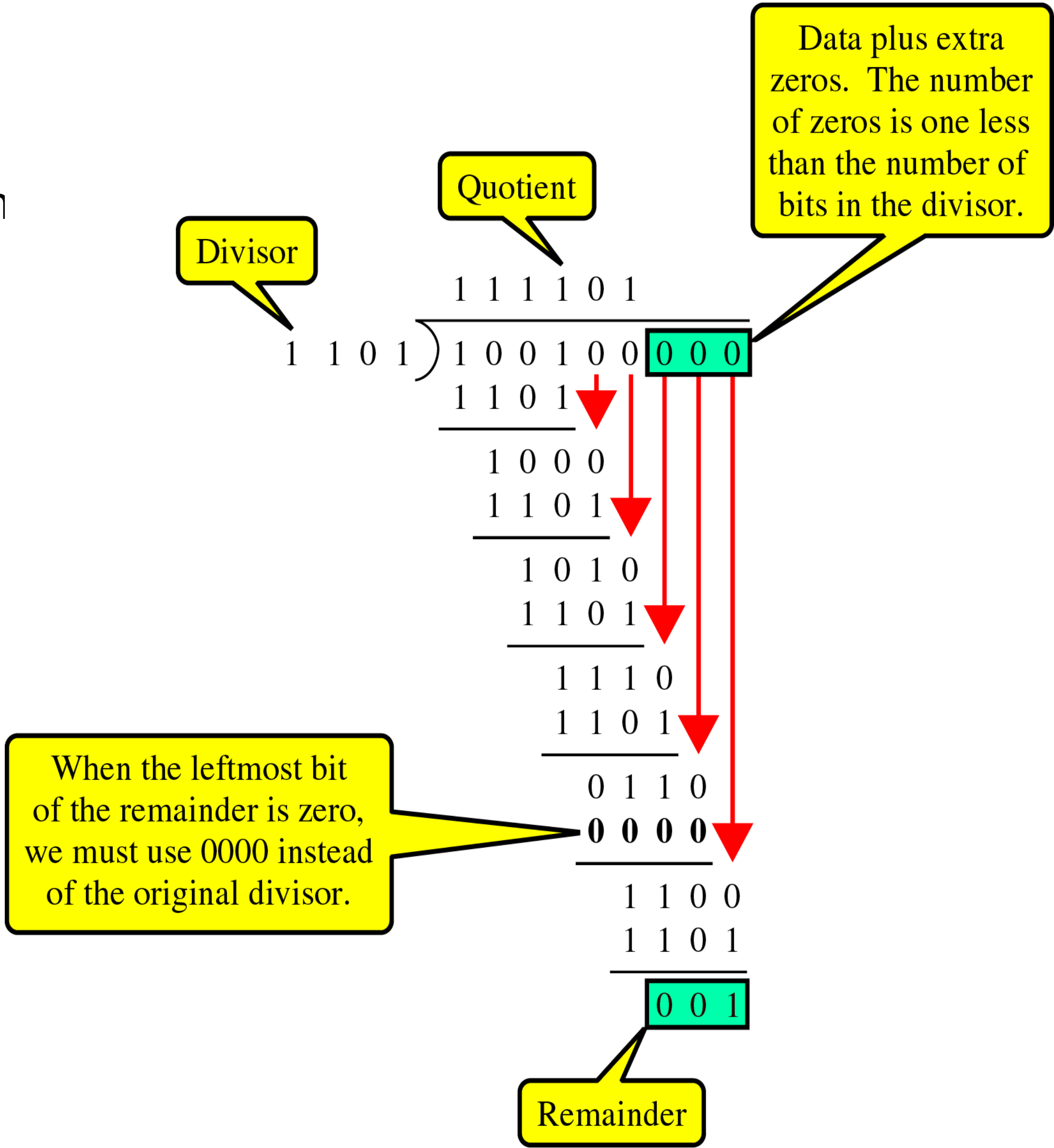
CRC-32

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

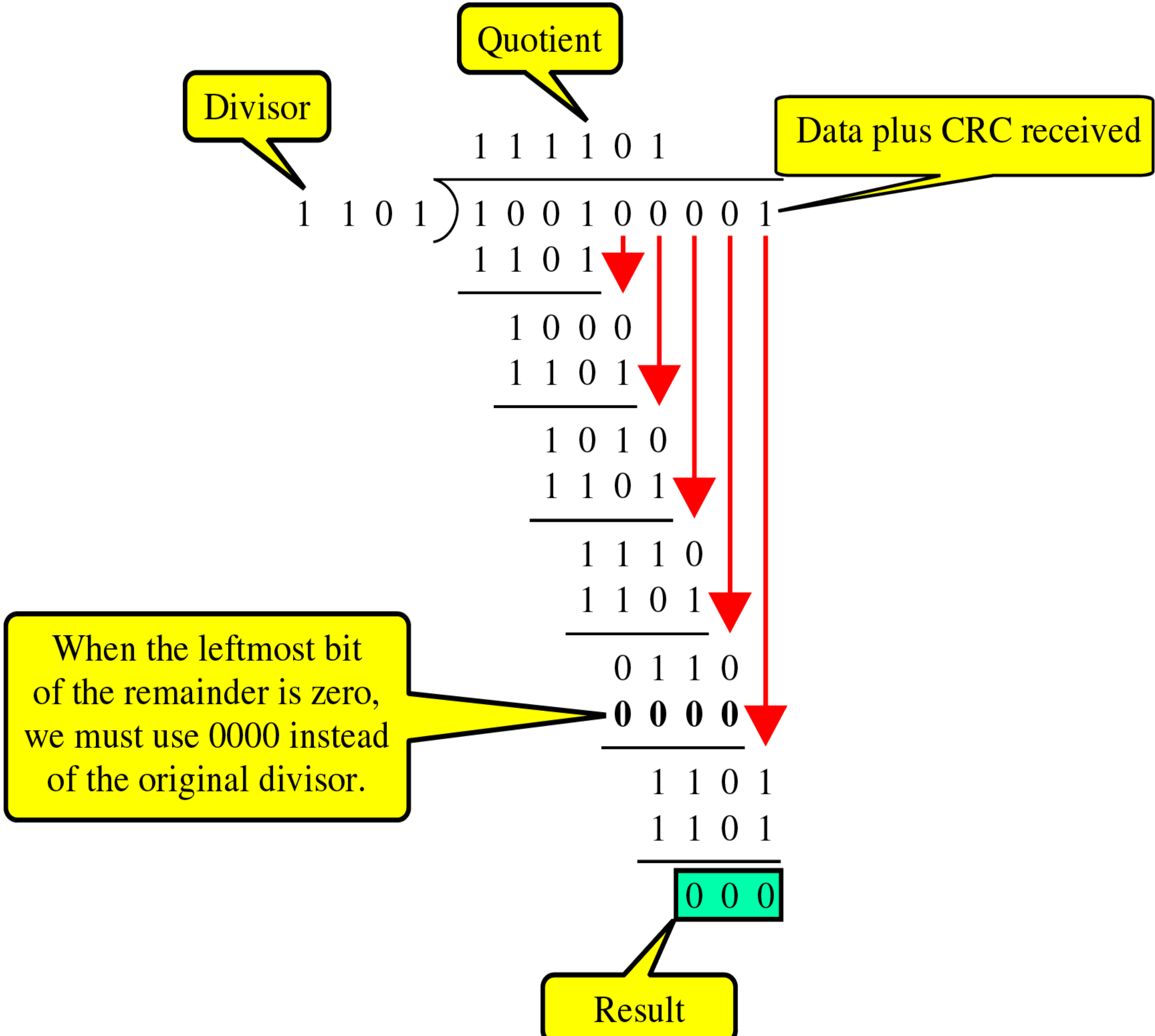
Eg

- Polynomial : 100100
- Divisor : 1101

- Sender
  - uses modular-2 division



Receiver

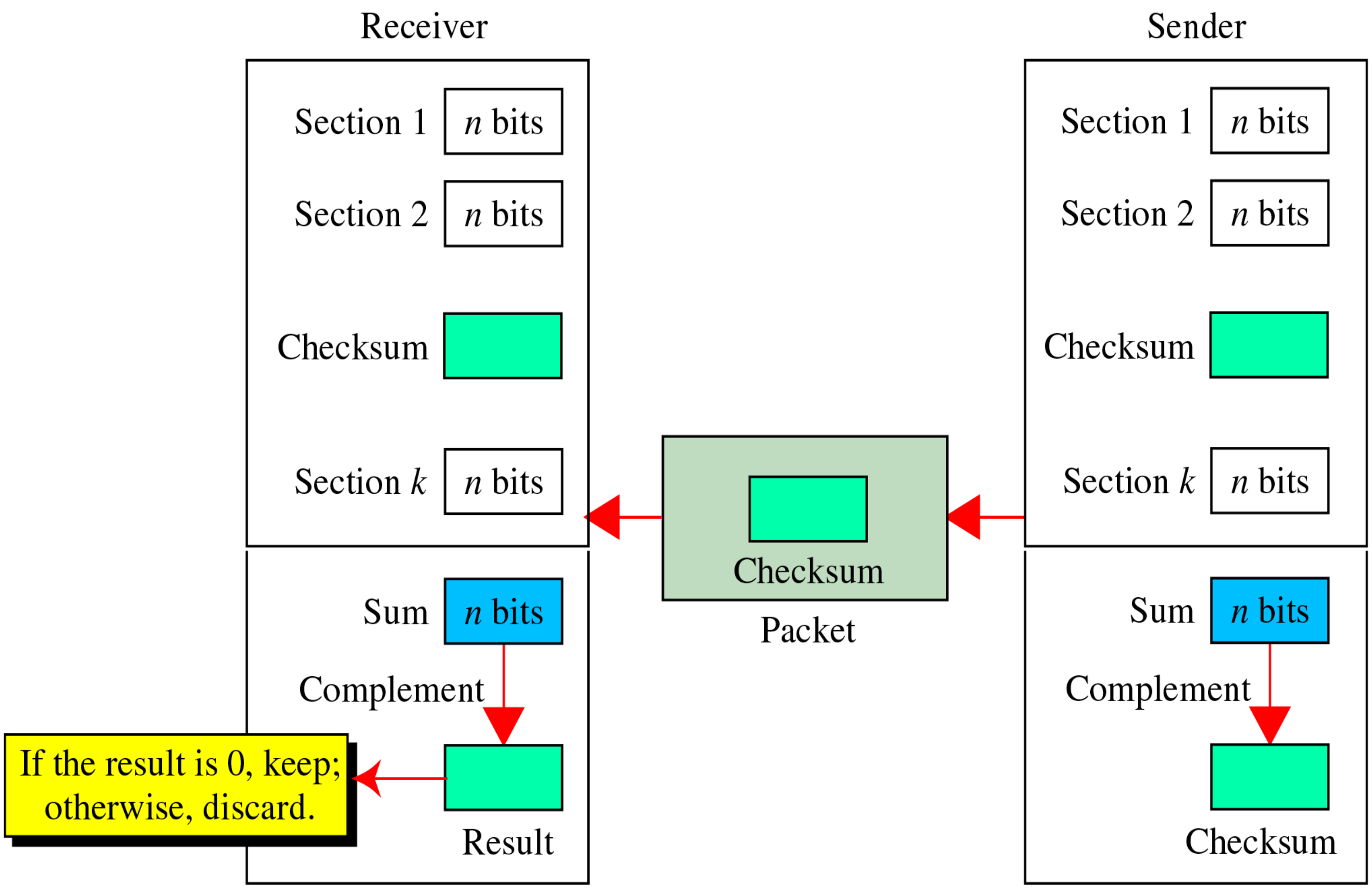

Quotient
Divisor
Data plus CRC received
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 1
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 1
1 1 0 1
0 0 0
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Result

Checksum

~ used by the higher layer protocols

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

- 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.
  - Sum all the bits in k blocks.
  - If carry exists add it to the sum.
  - The final result is complemented to make the checksum.

Detection(cont'd)

- data unit and checksum

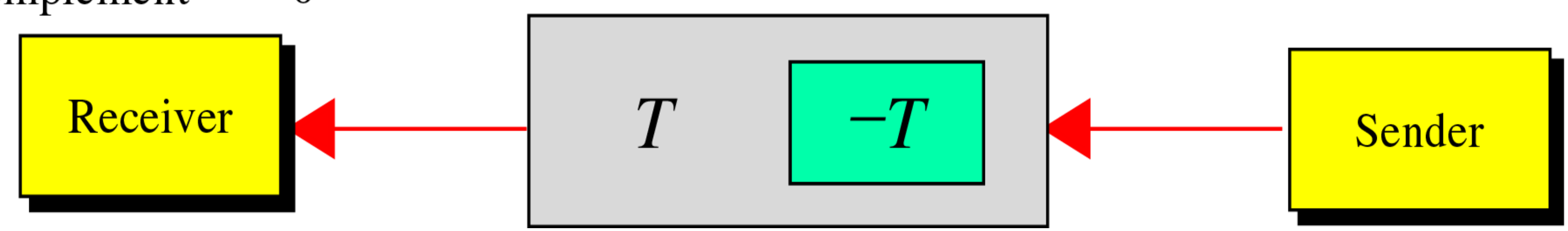
The receiver adds the data unit and the checksum field. If the result is all 1s, the data unit is accepted; otherwise it is discarded.

T

$-T$

Sum -0

Complement 0



Detection(cont'd)

- Sender

Original data : 10101001 00111001

10101001

00111001

11100010   Sum

00011101   Checksum

10101001 00111001 00011101 ← Checksum

Detection(cont'd)

- Receiver

Received data : 10101001 00111001 00011101

10101001
 00111001
 00011101

11111111 ← Sum
00000000 ← Complement

- Suppose that the sender wants to send 4 frames each of 8 bits, where the frames are 11001100, 10101010, 11110000 and 11000011.
- Calculate checksum for the same

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.

b) Given the data word 1111011111 and the divisor 11011,

i. Show the generation of the code word at the sender site (using binary division). (4 Marks)

ii. Show the checking of the code word at the receiver site (assume no error). (3 Marks)