

ERROR DETECTION AND CORRECTION

Komunikasi Data

Objective

- Mahasiswa mampu mengenali eror yang terjadi pada transmisi asinkron, dan mendeteksinya dengan menggunakan bit parity.
- Mahasiswa mampu mengenali eror yang terjadi pada transmisi sinkron, dan eror dikoreksi dengan menggunakan LRC dan VRC.
- Mahasiswa dapat menjelaskan bagaimana mendeteksi eror pada transmisi sinkron dan dengan menggunakan checksum dan CRC
- Mahasiswa dapat menjelaskan bagaimana eror pada transmisi sinkron dapat dikoreksi dengan menggunakan kode Hamming
- Mahasiswa mampu menjelaskan bagaimana eror dapat dikoreksi menggunakan ARQ (Automatic Repeat Request).

Error Detection and Correction

1 Types of Errors

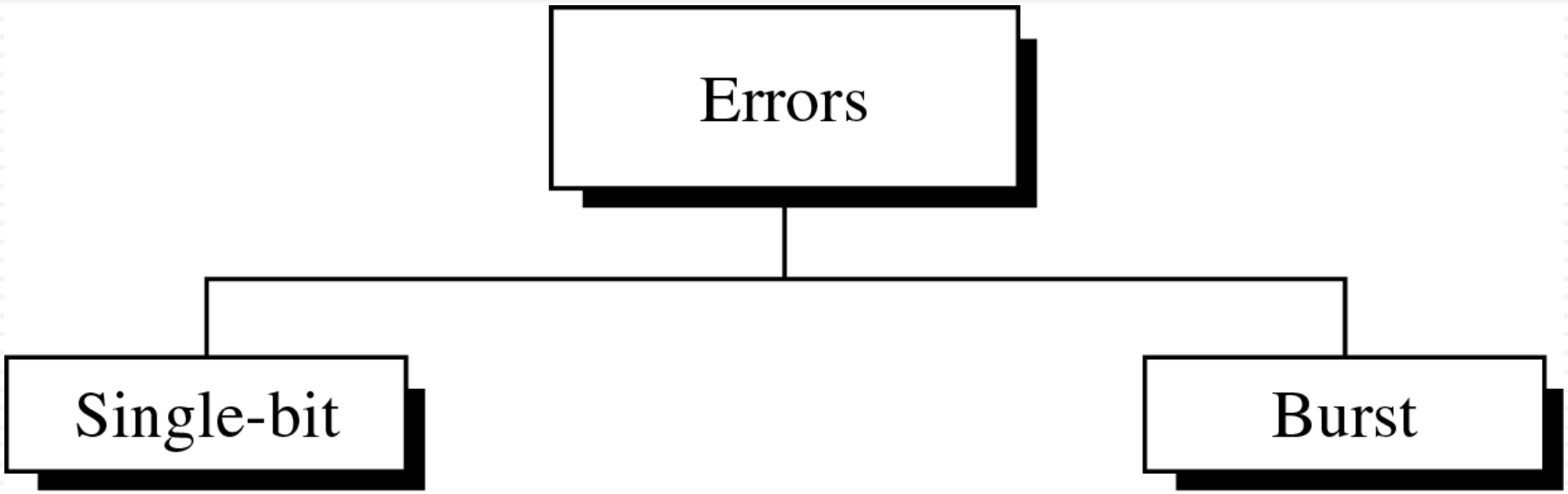
2 Detection

3 Error Correction

Error Detection and Correction

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

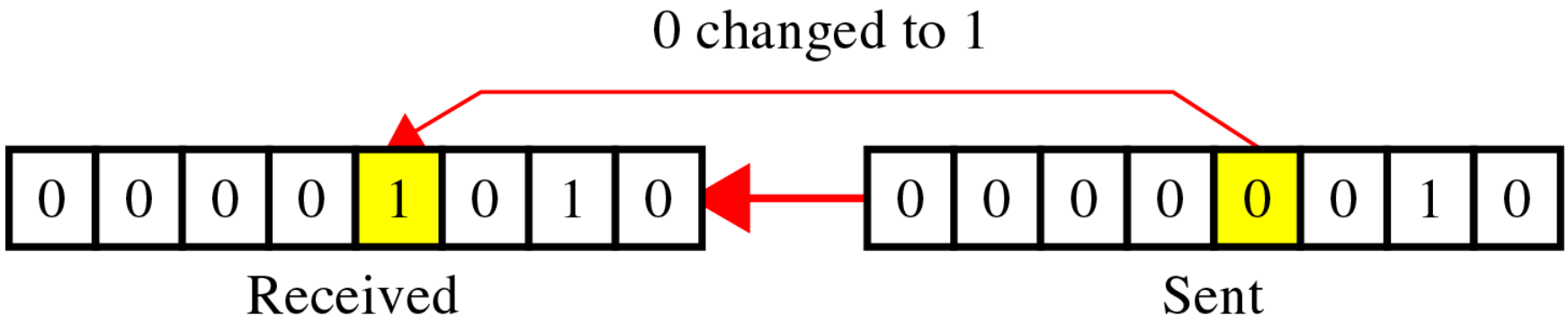
1. Type of Errors



Type of Errors(cont'd)

- Single-Bit Error

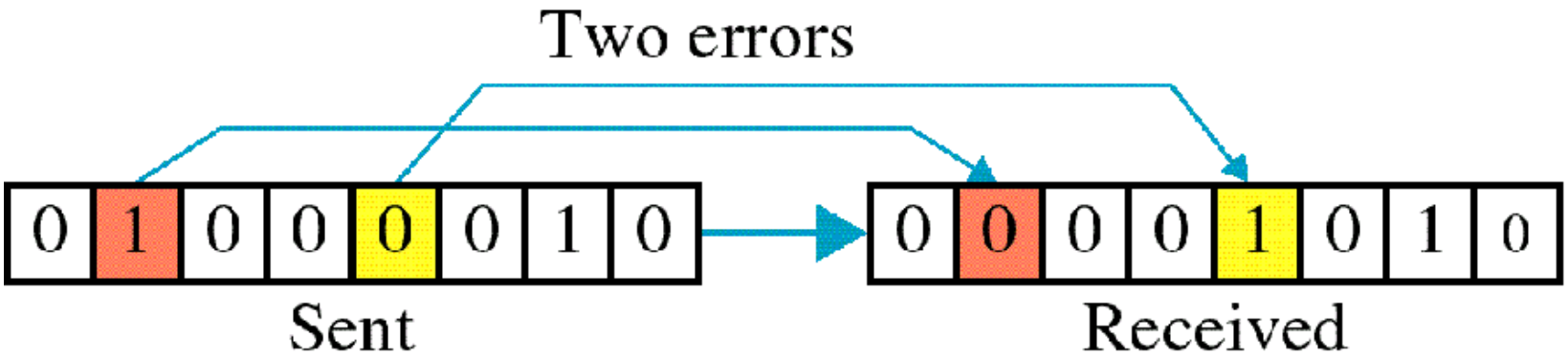
~ is when only one bit in the data unit has changed (ex : ASCII STX - ASCII LF)



Type of Errors(cont'd)

- Multiple-Bit Error

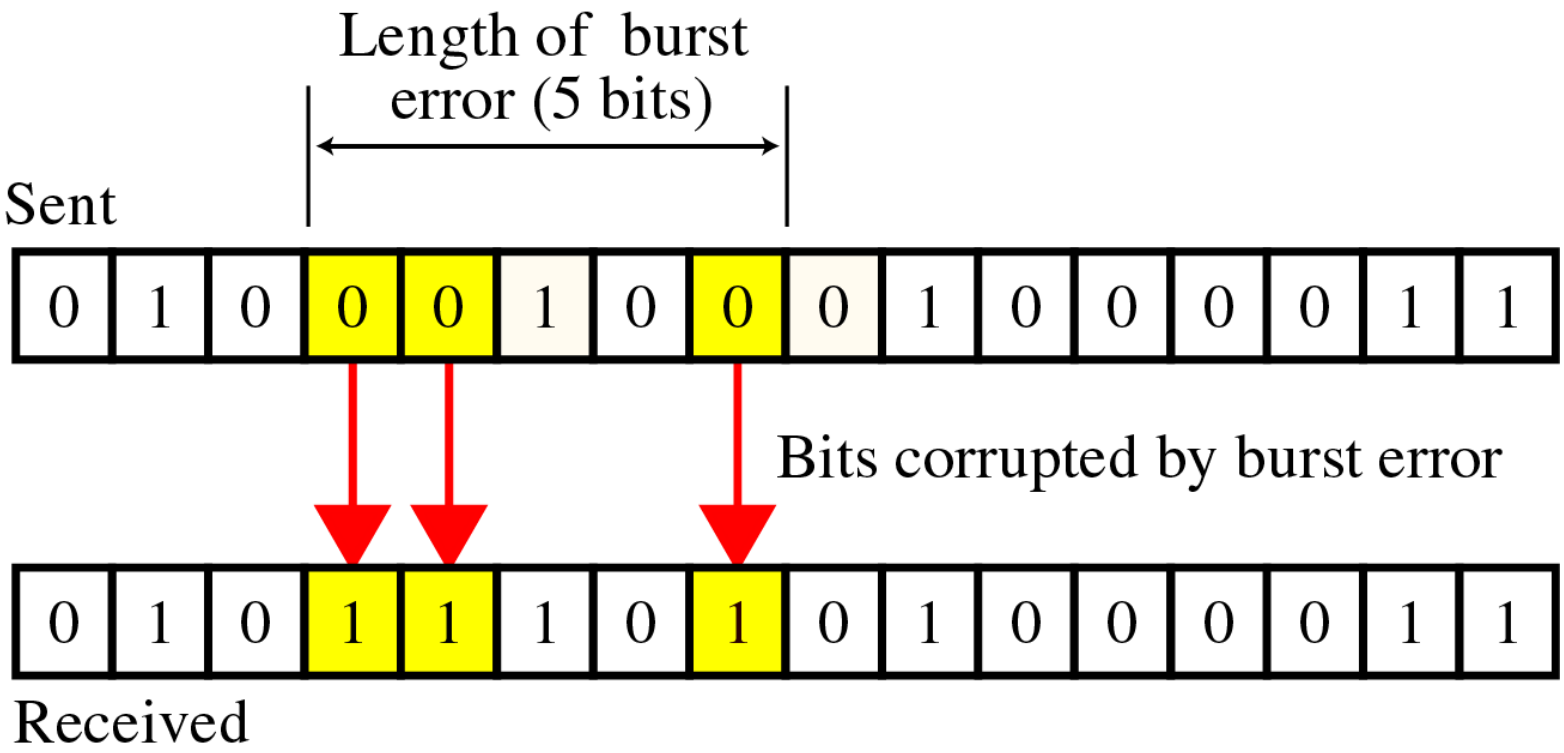
~ is when two or more nonconsecutive bits in the data unit have changed(ex : ASCII B - ASCII LF)



Type of Errors(cont'd)

- Burst Error

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

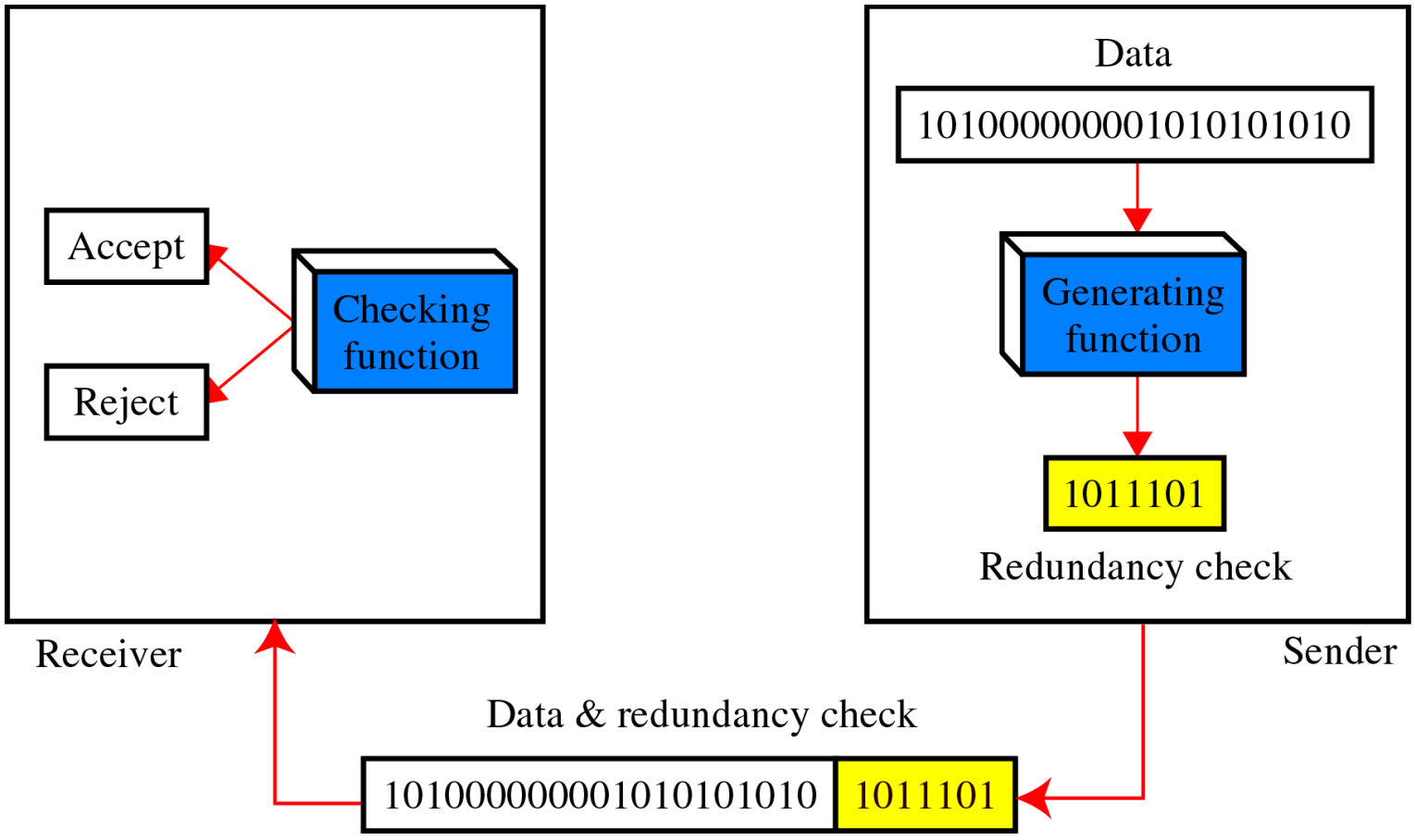


2. Detection

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

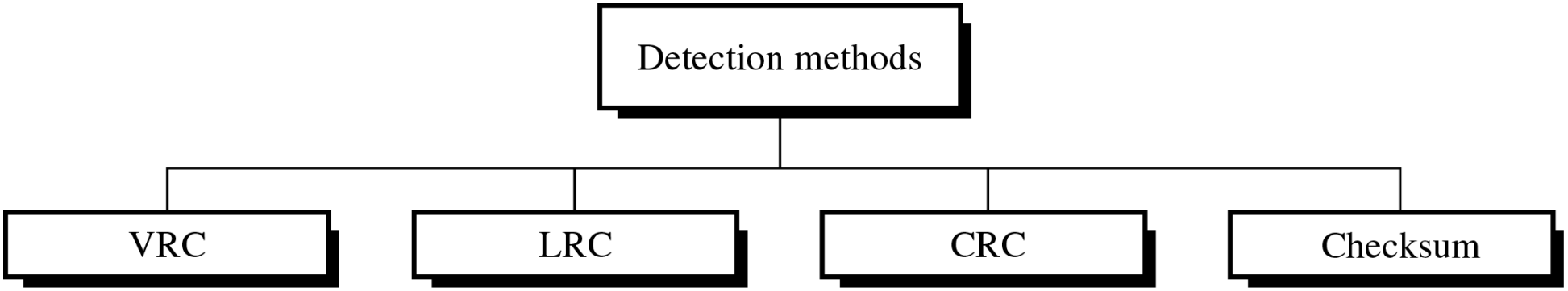
Detection(cont'd)

- Redundancy



Detection(cont'd)

- Detection methods
  - VRC (Vertical Redundancy Check)
  - LRC (Longitudinal Redundancy)
  - CRC (Cyclical redundancy Check)
  - Checksum

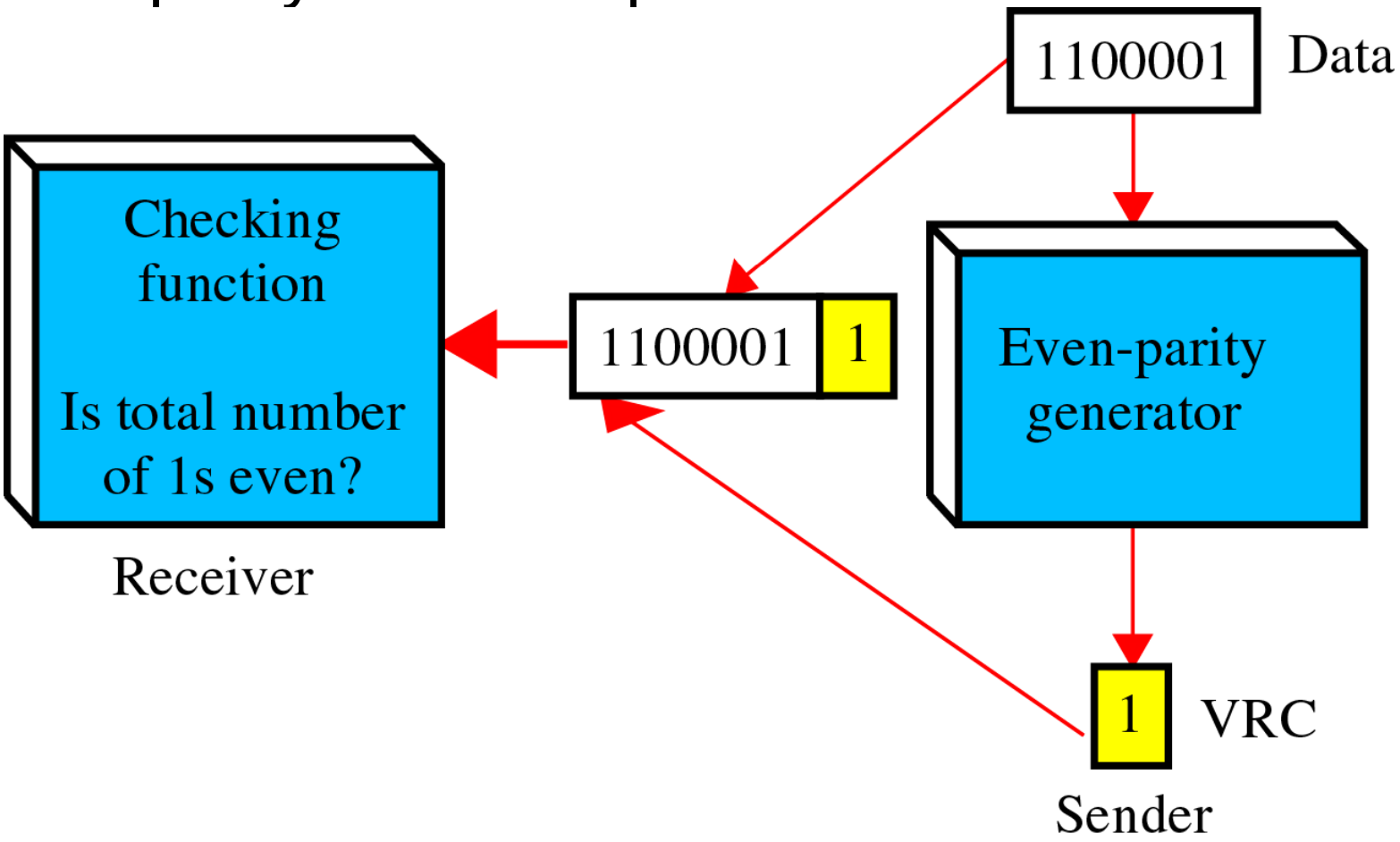


Detection(cont'd)

- VRC(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
  - VRC can detect all single-bit errors. It can detect multiple-bit or burst errors only the total number of errors is odd.

Detection(cont'd)

- Even parity VRC concept



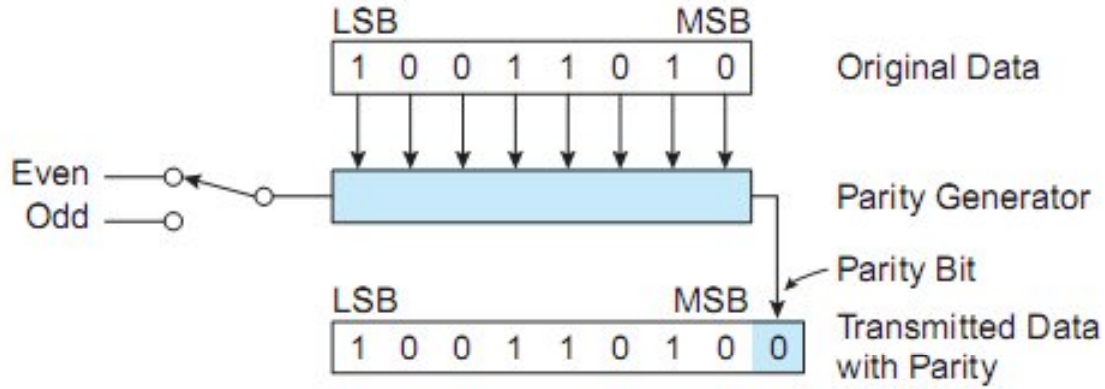
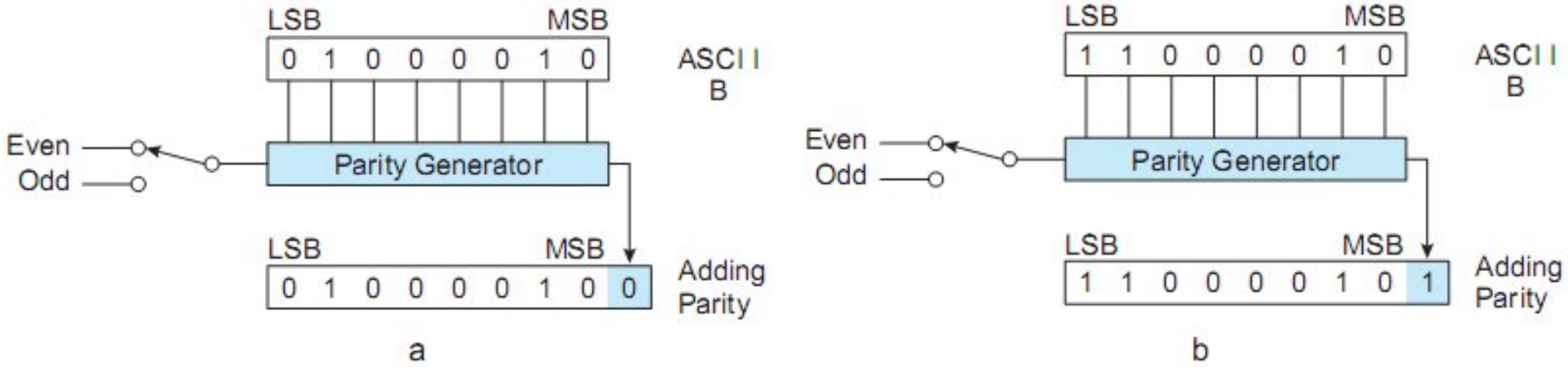
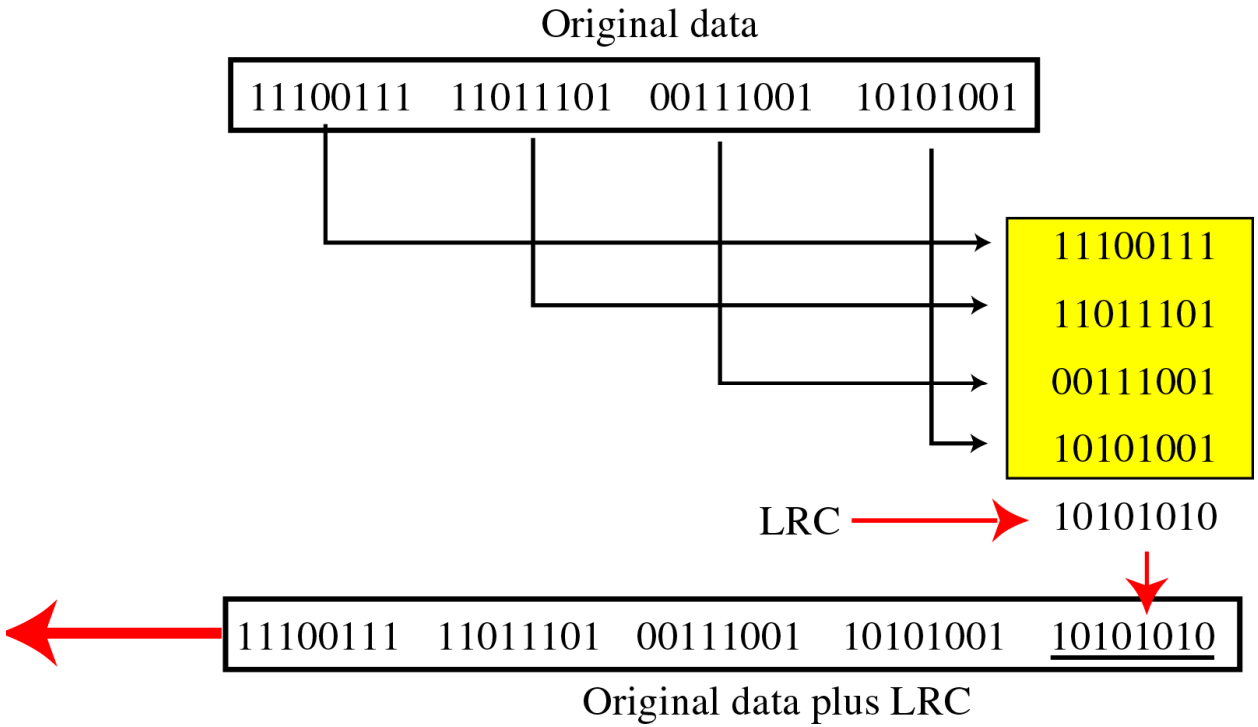


FIGURE 3-1 Appending Parity Bit



Detection(cont'd)

- LRC(Longitudinal Redundancy Check)
  - Parity bits of all the positions are assembled into a new data unit, which is added to the end of the data block



EXAMPLE 3-4

Determine the states of the LRC bits for the asynchronous ASCII message "Help!"

SOLUTION

The first step in understanding the process is to list each of the message's characters with their ASCII code and even VRC parity bit:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!

Next, for each vertical column, find the LRC bit by applying the exclusive OR function. To make this process easier, you can consider the results of the exclusive OR process as being low or zero (0), if the number of ones (1) are even, and one (1) if the count is odd. For instance, in the LSB column, there are two 1's, so the LRC bit for that column is a 0. And for the rest:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC

EXAMPLE 3-5

Show how a good message would produce an LRC of 0 at the receiver.

SOLUTION

Repeat the process as before, but include the LRC character this time. Note that the number of 1s in each column are always even if there are no errors present:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	0	0	0	0	0	0	Receiver LRC

EXAMPLE 3-6

Illustrate how LRC/VRC is used to correct a bad bit.

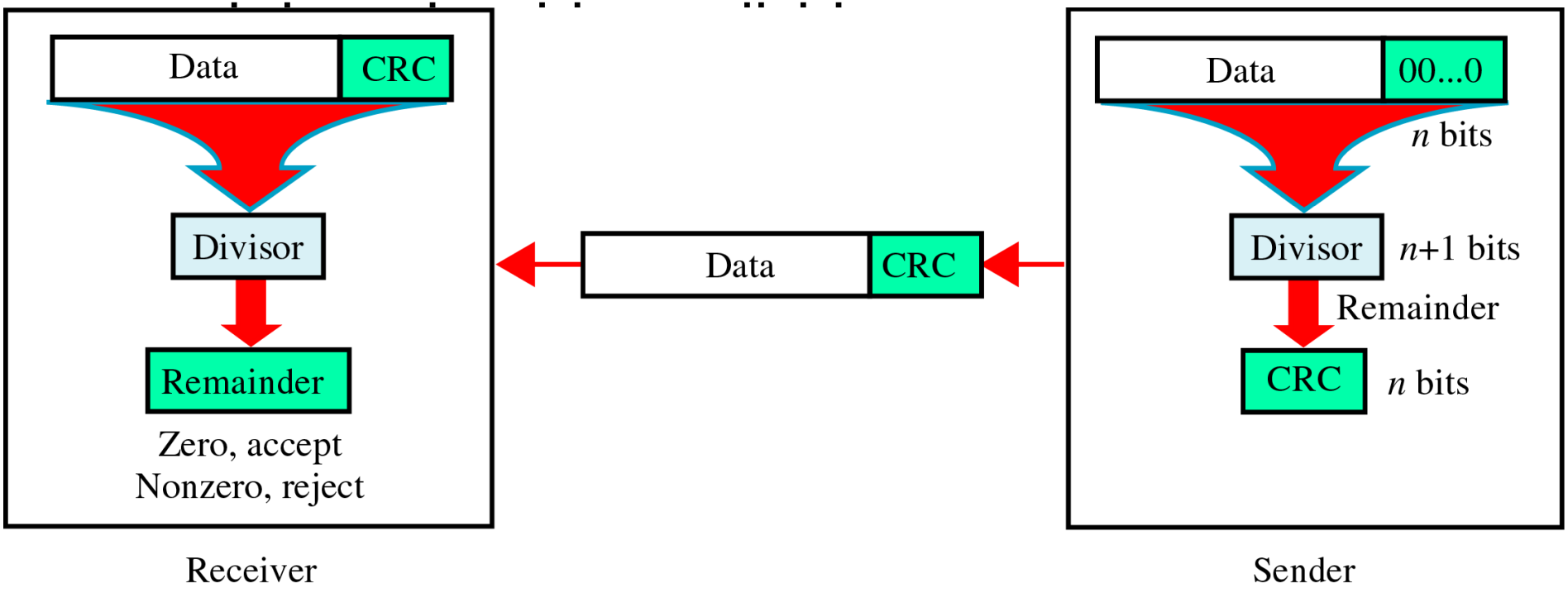
SOLUTION

We will use the same message, but by placing an error in the received data would cause the l character to print as an *h*. You can compare the data with the good example to satisfy yourself as to which bit is bad and confirm that the LRC process does indeed pick out the same bit.

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	0	1	0	1	1	0	h
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	1	0	0	0	0	1	Received LRC

Detection(cont'd)

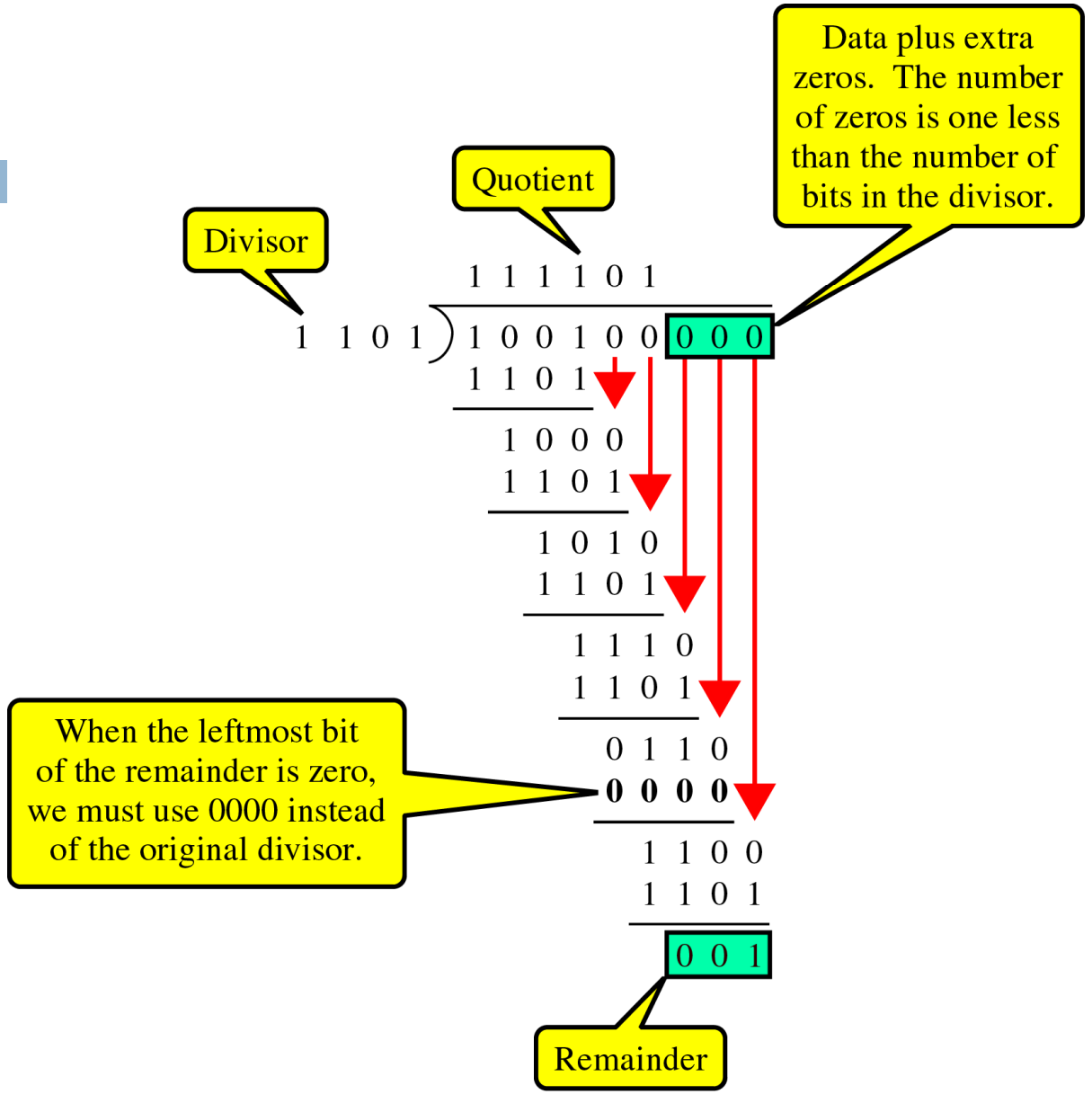
- CRC(Cyclic Redundancy Check)



Detection(cont'd)

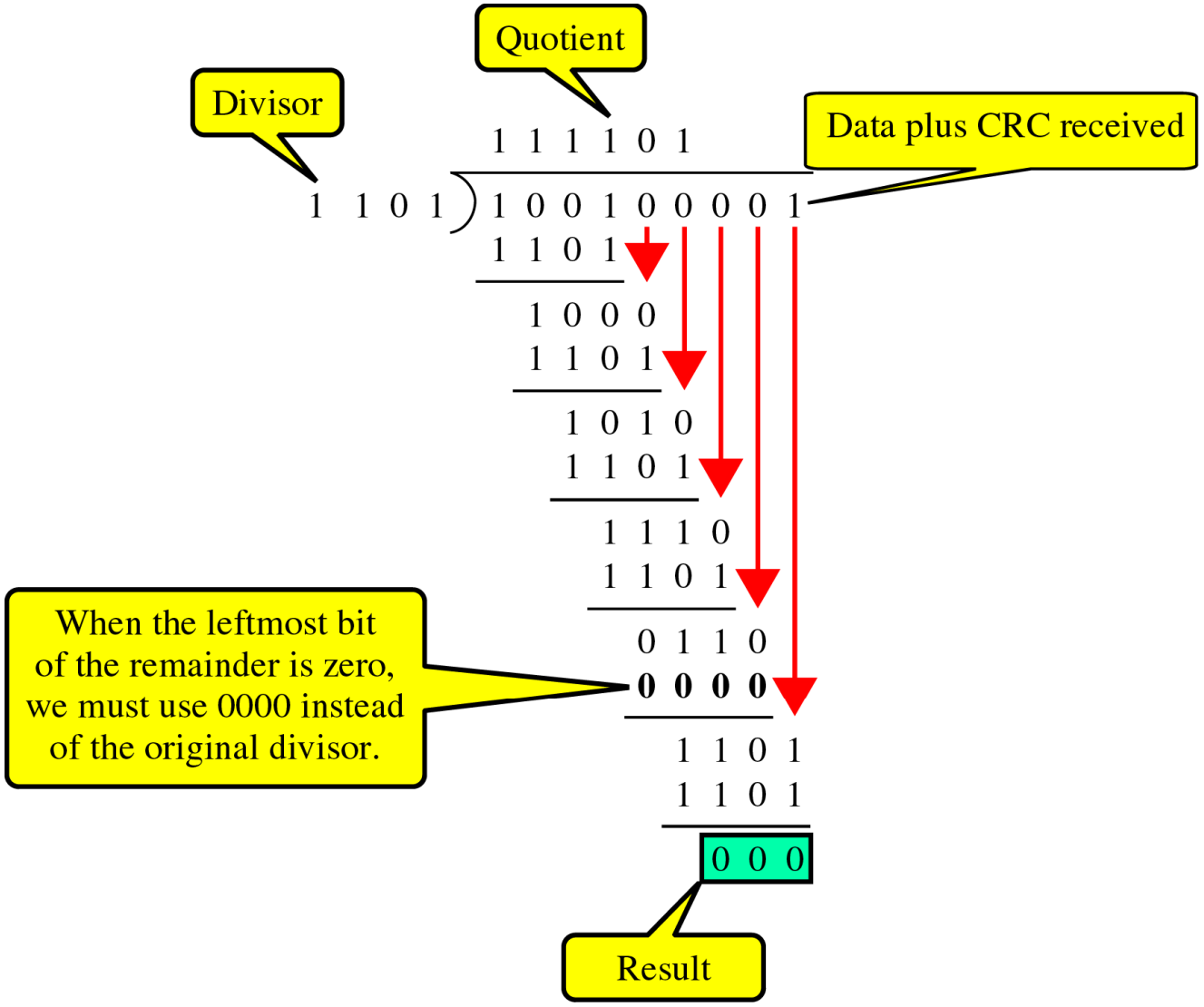
- CRC generator
  - ~ uses modular-2 division.





Detection(cont'd)

Binary Division
in a
CRC Checker



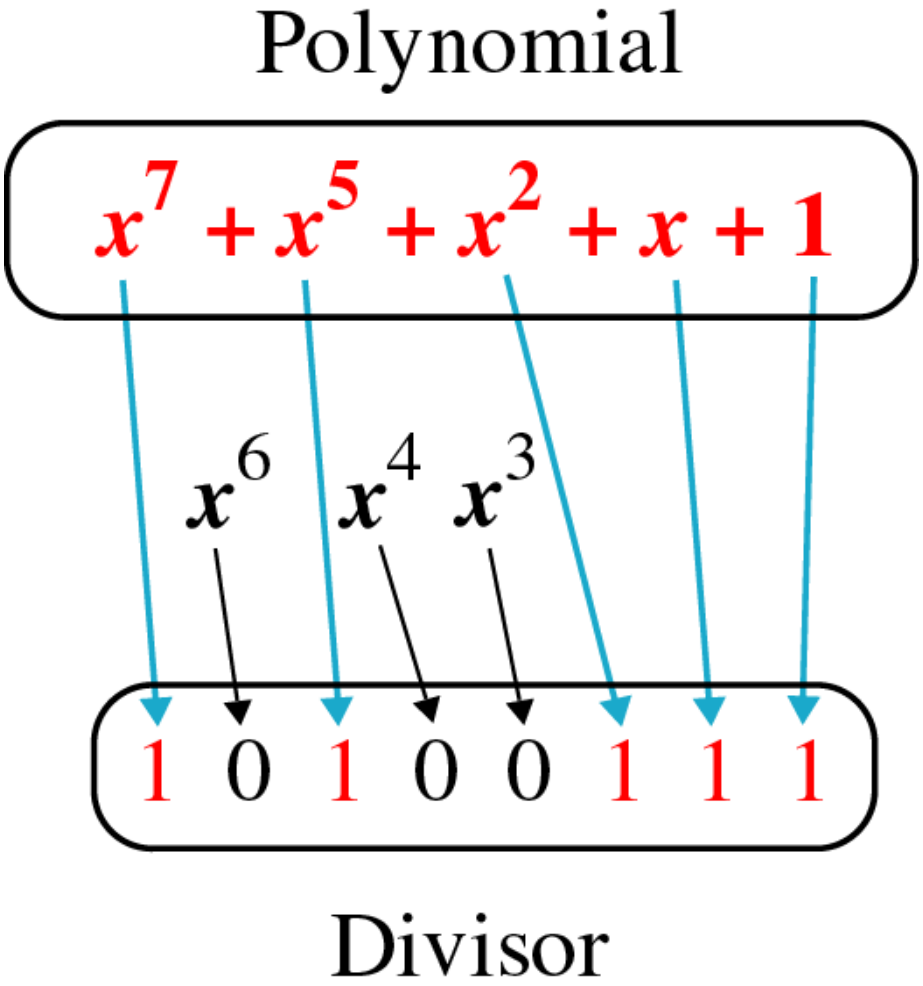
Detection(cont'd)

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

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

Detection(cont'd)

- A polynom



Detection(cont'd)

- Standard polynomials

CRC-12

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

CRC-16

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

CRC-ITU-T

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

CRC-32

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

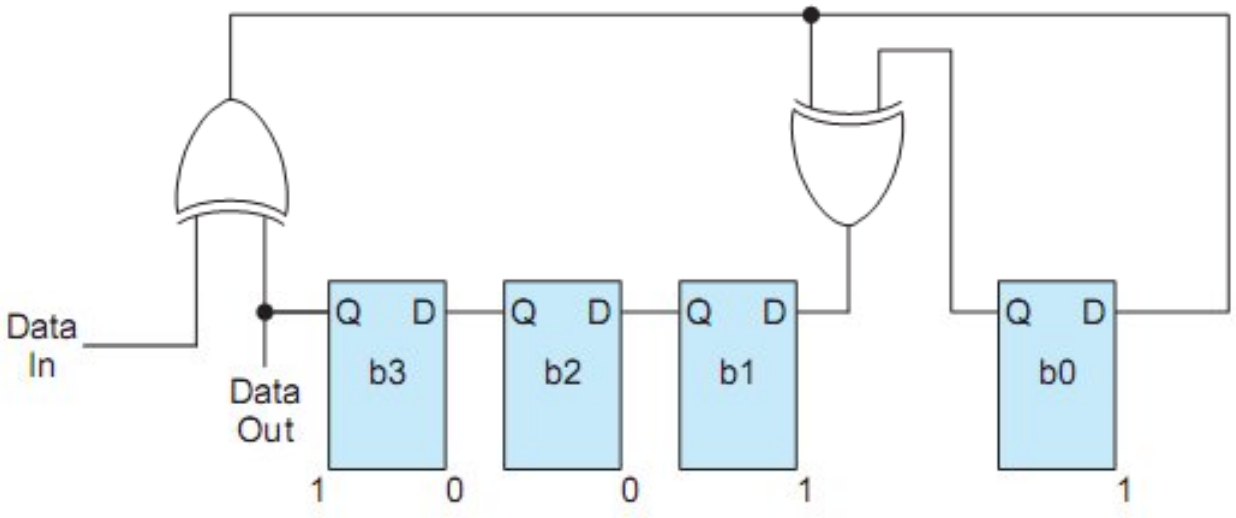


FIGURE 3-8 CRC-4 Circuit

Detection(cont'd)

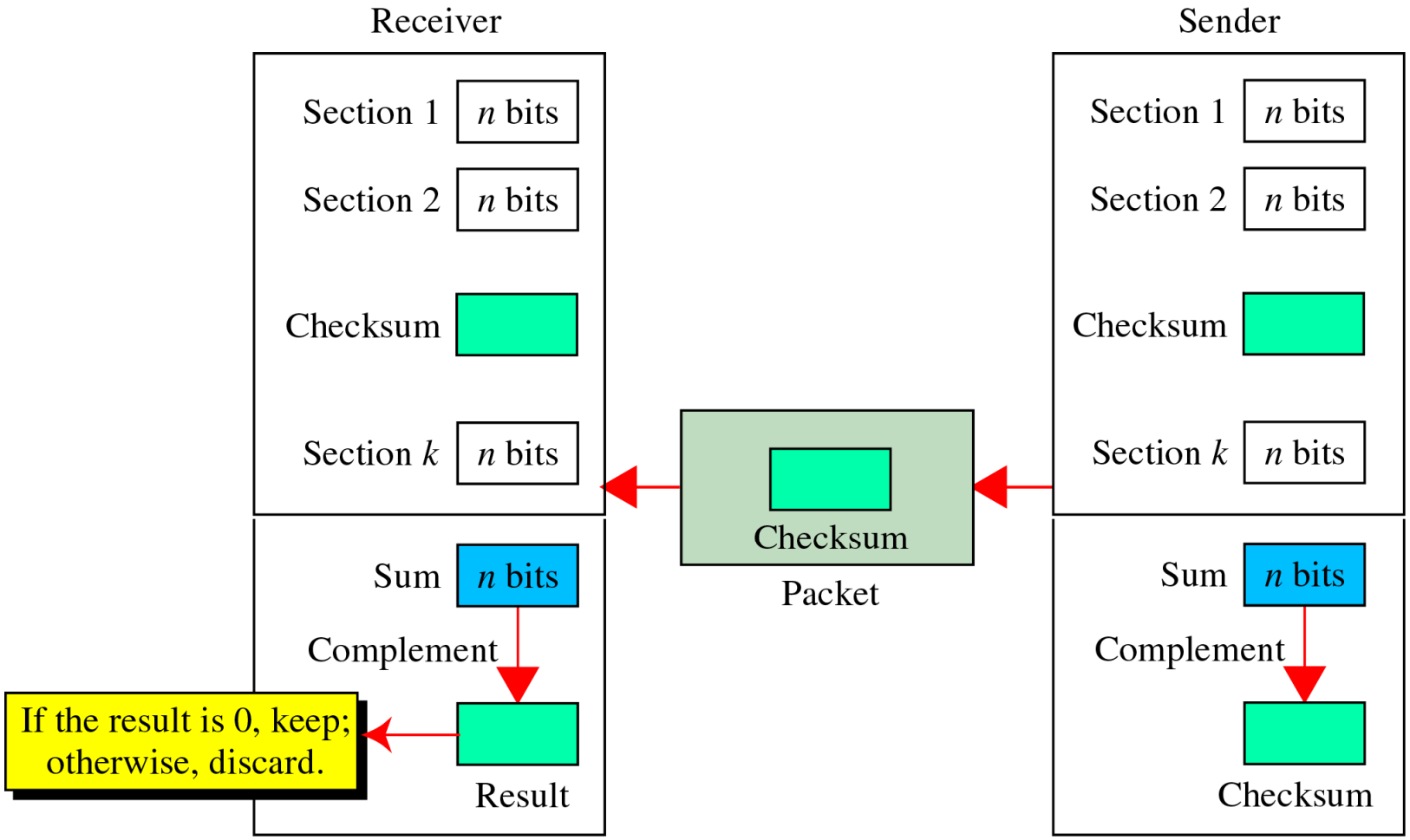
- Checksum

~ 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.

EXAMPLE 3-10

What is the checksum value for the extended ASCII message "Help!"?

SOLUTION

The checksum value is found by adding up the bytes representing the Help! characters:

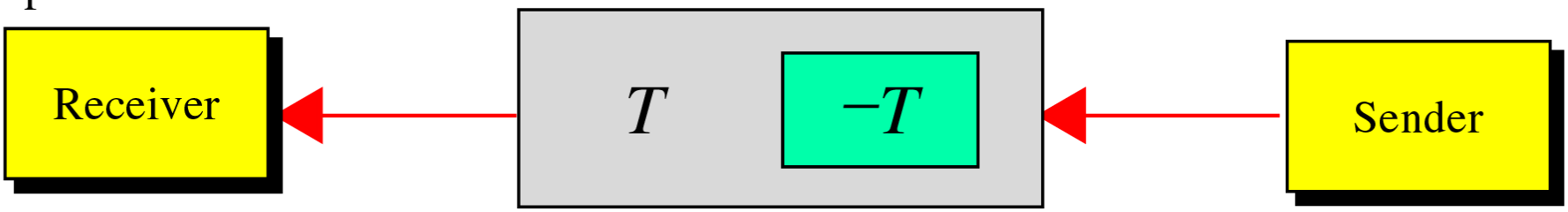
01001000	H
01100101	e
01101100	l
01110000	p
00100001	!
00010000	Checksum

Detection(cont'd)

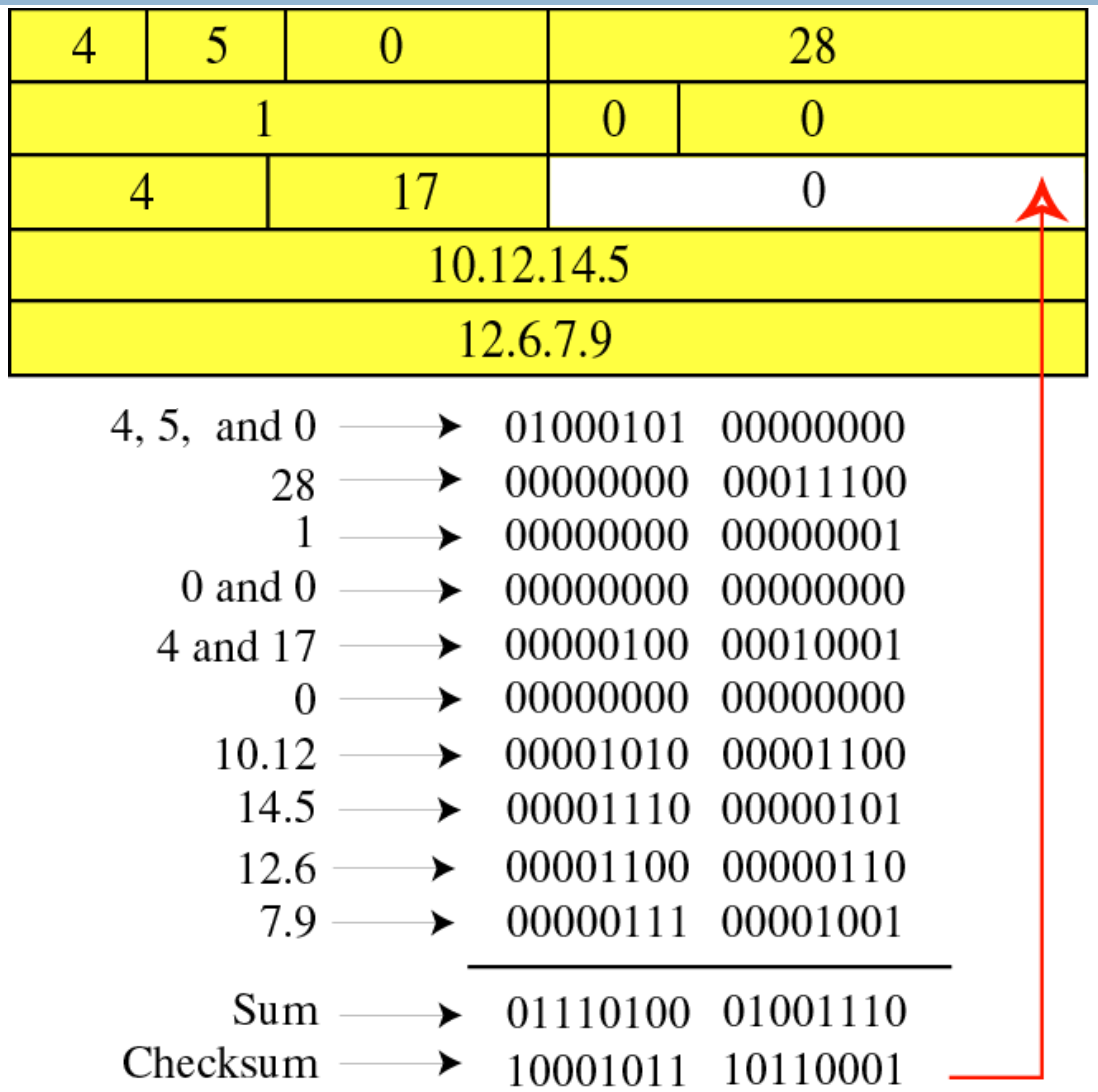
- data unit and checksum

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.

$$
\begin{array}{rr}
 & T \\
 & -T \\
\hline
 & -0 \\
\text{Sum} & \\
\text{Complement} & 0
\end{array}
$$



Detection(cont'd)



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.