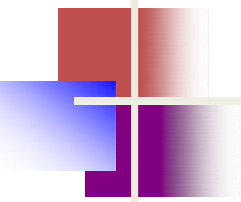

Data Link Layer



Duties of data link layer

Packetizing

Addressing

Error
control

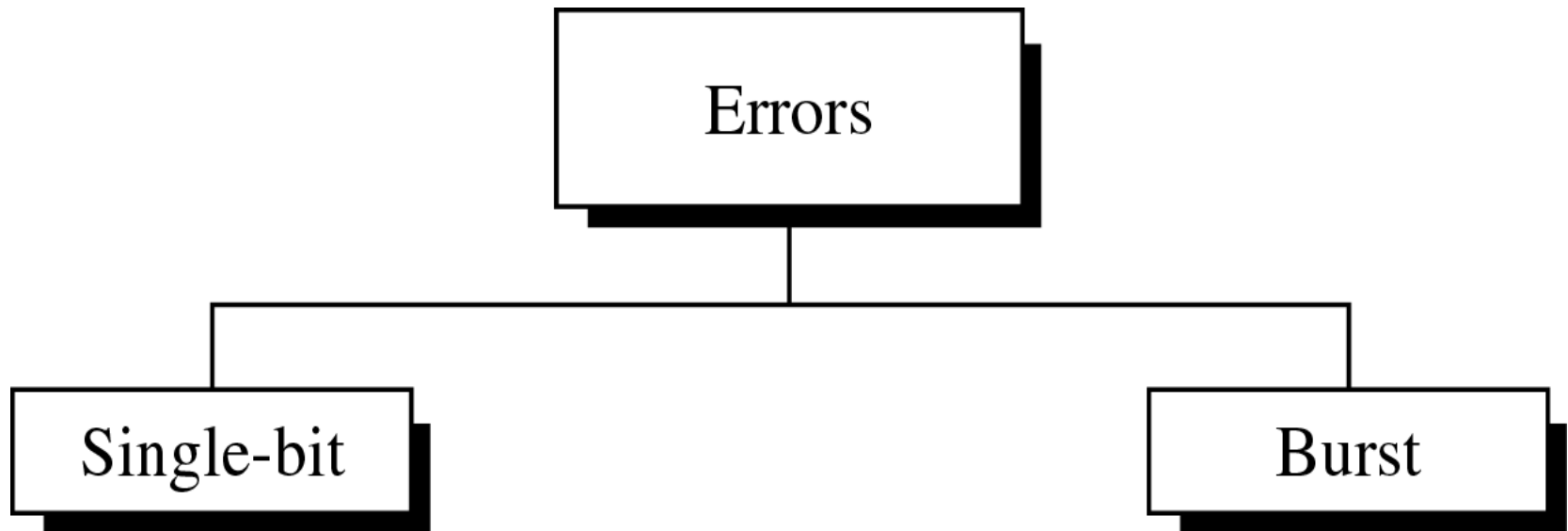
Flow
control

Access
control

Error Detection and Correction

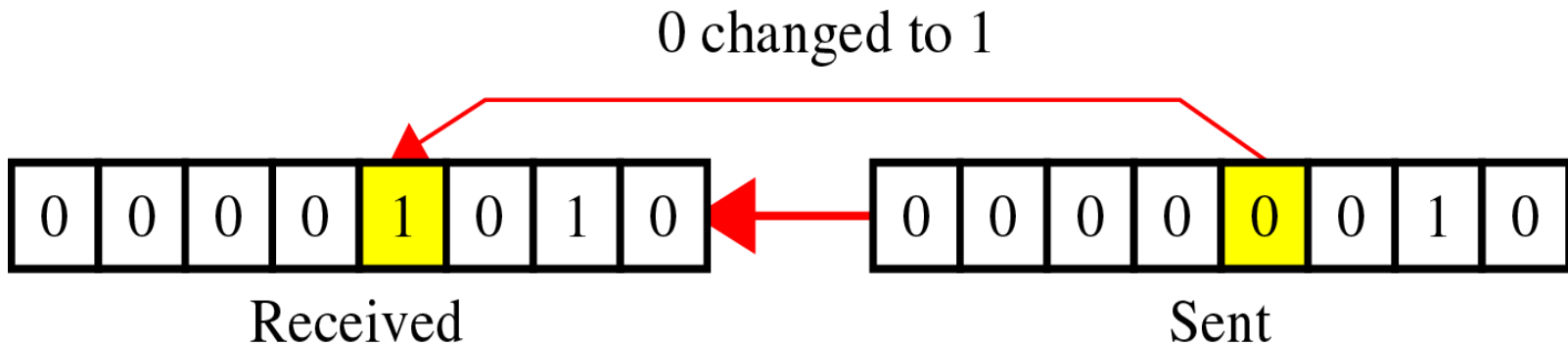
- Data can be corrupted during transmission.
- For reliable communication, error must be detected and corrected.
- Error Detection and Correction are implemented either at the data link layer or the transport layer of the OSI model.

Type of Errors



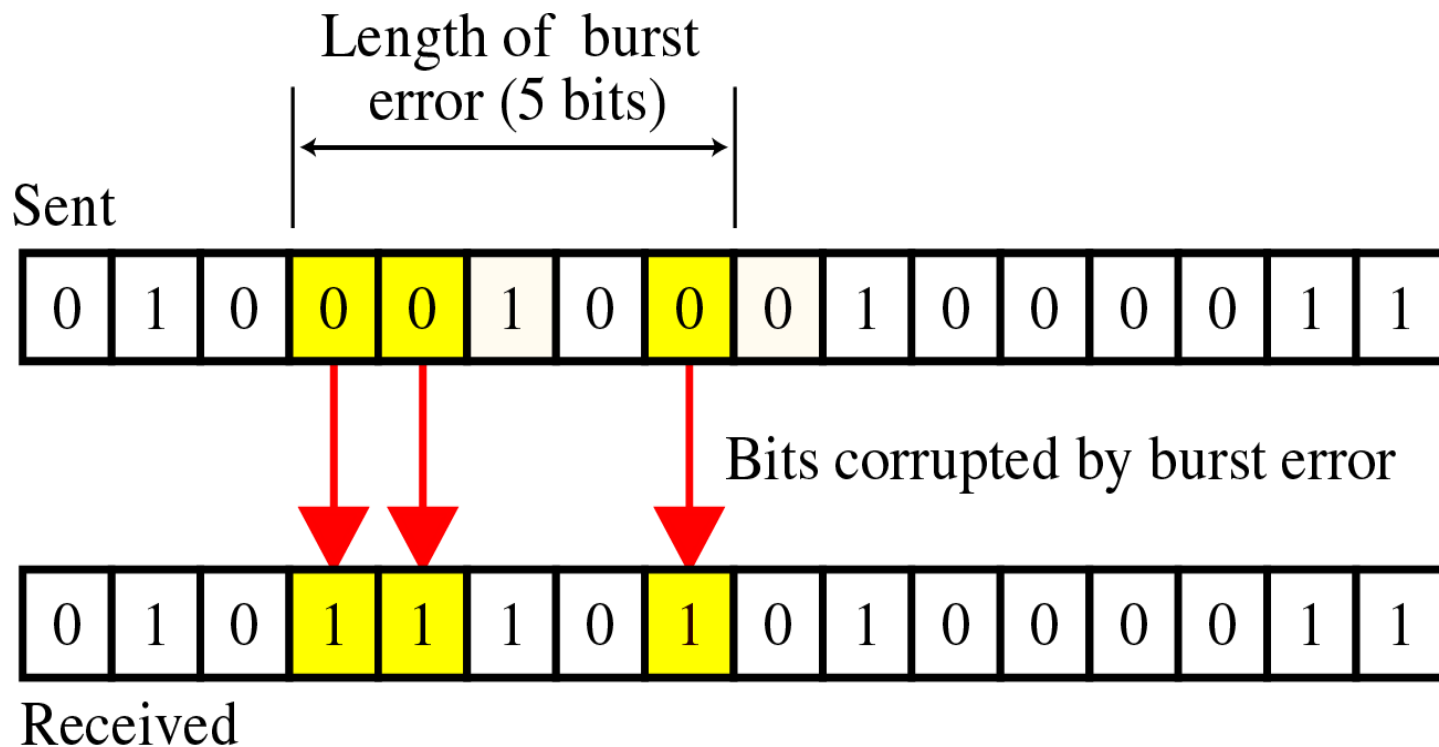
Single-Bit

~ 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

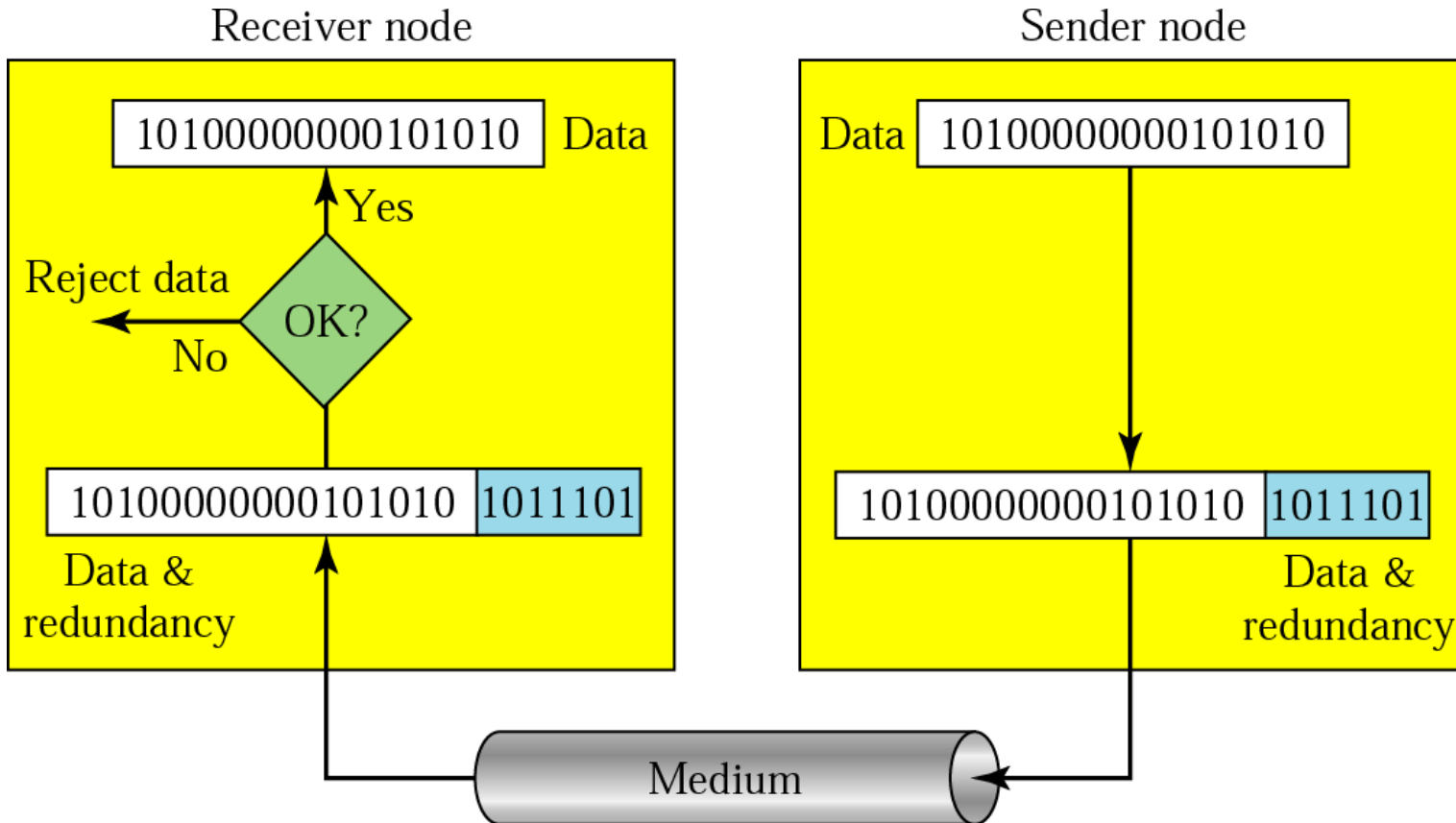


Detection

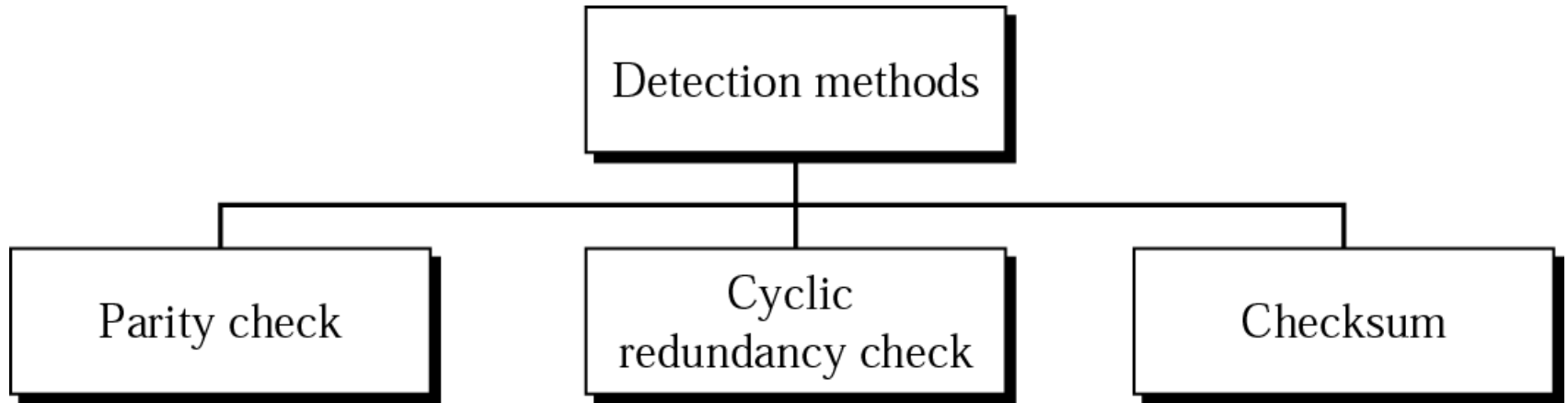
- Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

Detection(cont'd)

- Redundancy

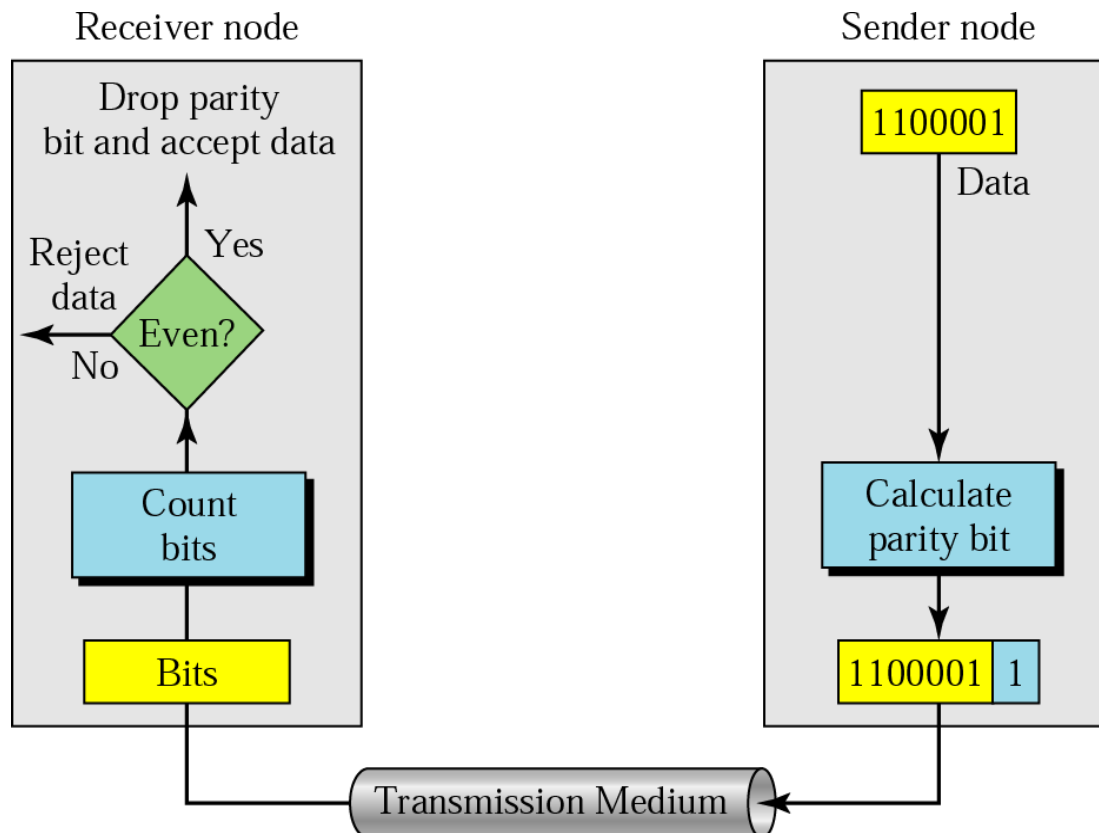


Error Detection Methods



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



Parity Calculation

Assuming even parity, find the parity bit for each of the following data units.

- a. 1001011
- b. 0001100
- c. 1000000
- d. 1110111

Assuming odd parity, find the parity bit for each of the following data units.

- a. 1001011
- b. 0001100
- c. 1000000
- d. 1110111

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

11101110 11011110 11100100 11011000 11001001

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

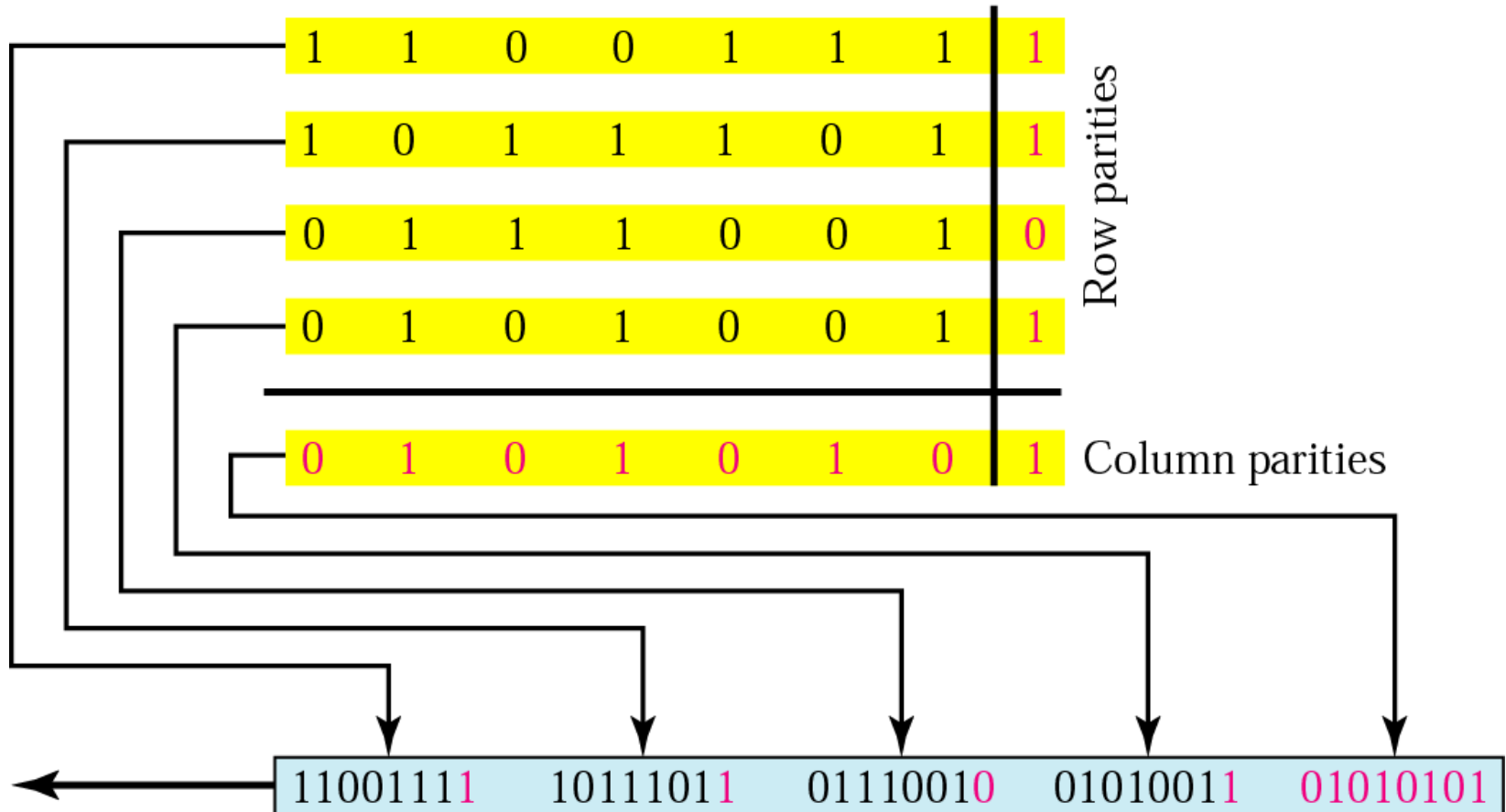
The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, 4). The receiver knows that the data are corrupted, discards them, and asks for retransmission.

Two-Dimensional Parity Check (LRC)

Longitudinal Redundancy Check

Original data

1100111	1011101	0111001	0101001
---------	---------	---------	---------



Data and parity bits

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.

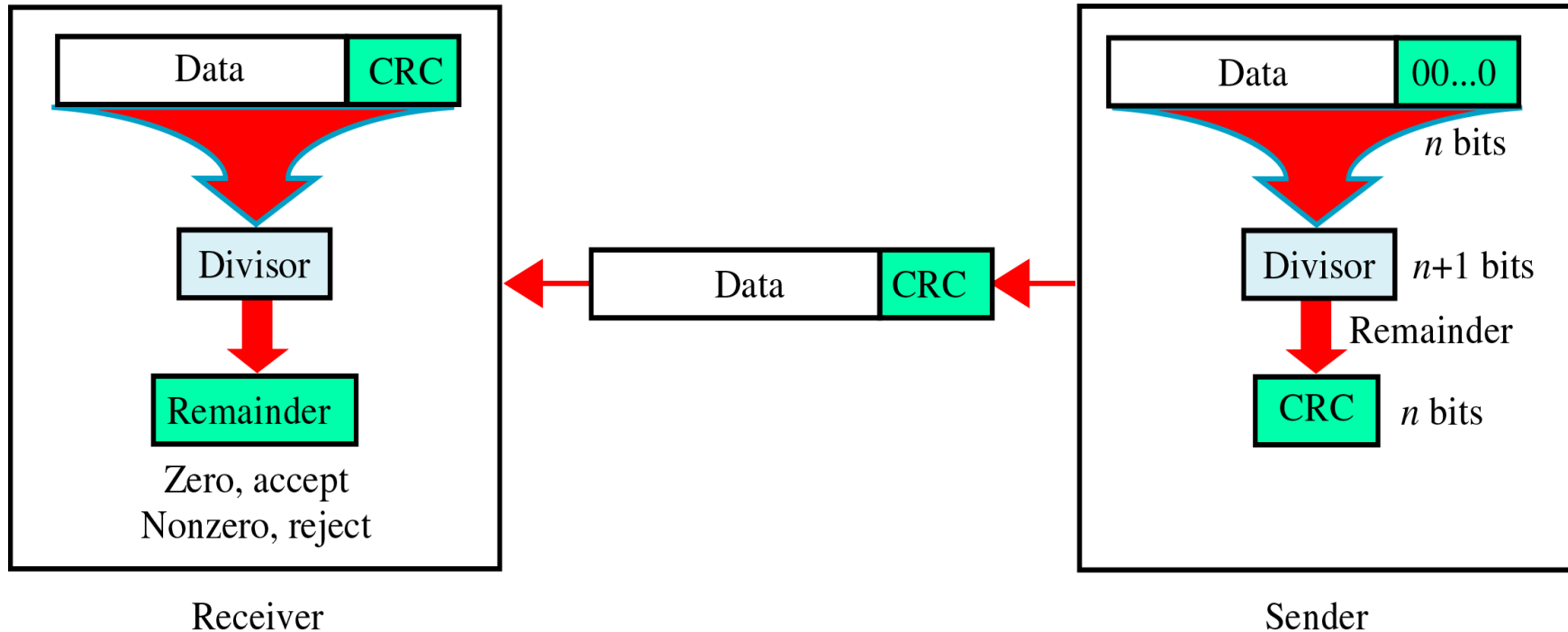
10100011 10001001 11011101 11100111 10101010

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 10101010

CRC(Cyclic Redundancy Check)

~ is based on binary division.



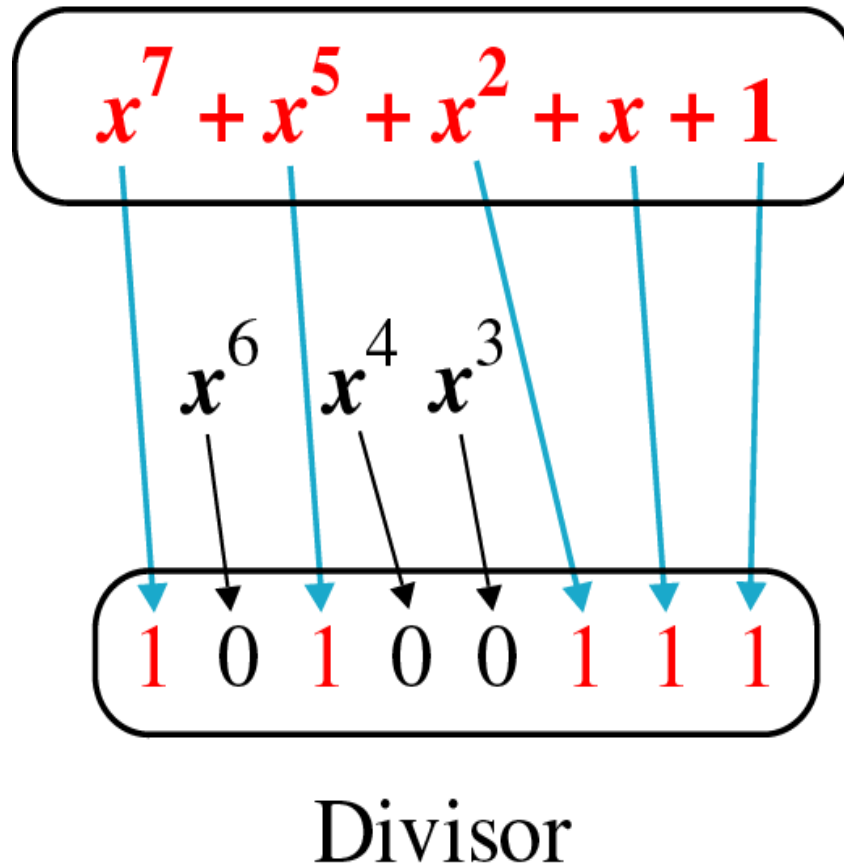
Detection(cont'd)

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

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

Detection(cont'd)

- A polynomial or



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

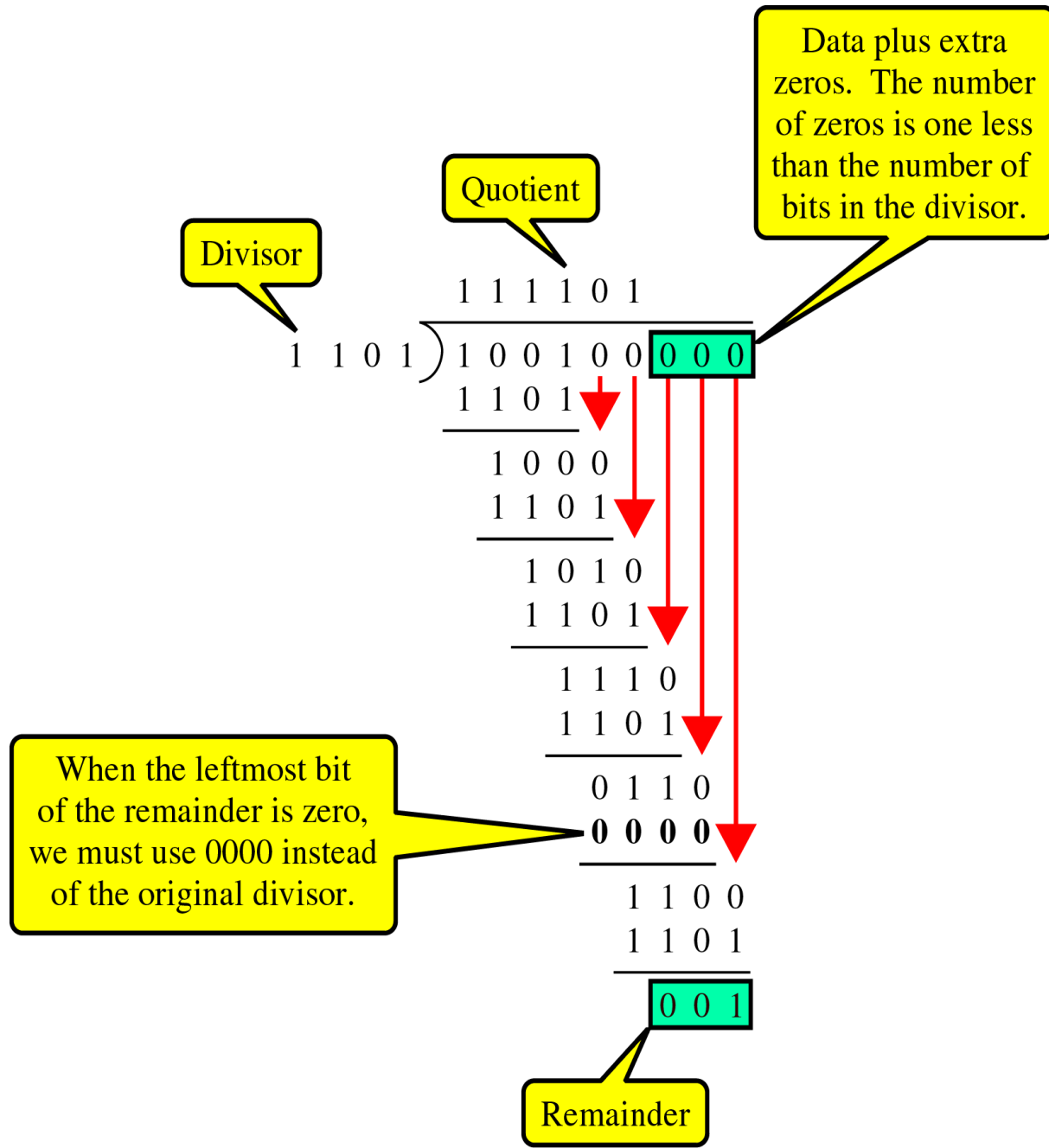
$$x^{16} + x^{12} + x^5 + 1$$

CRC-32

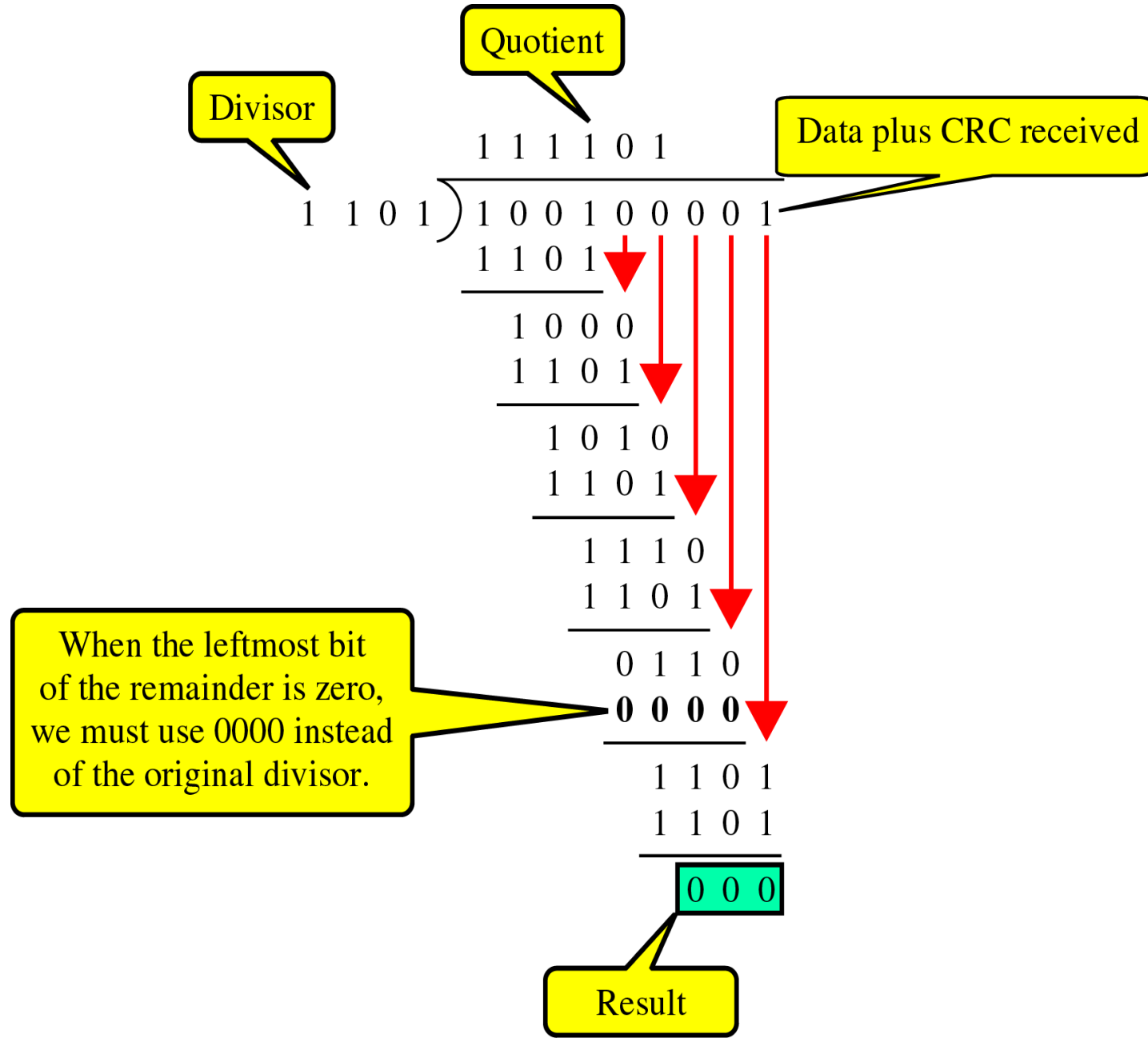
$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

- Sender

~ uses modular-2 division.



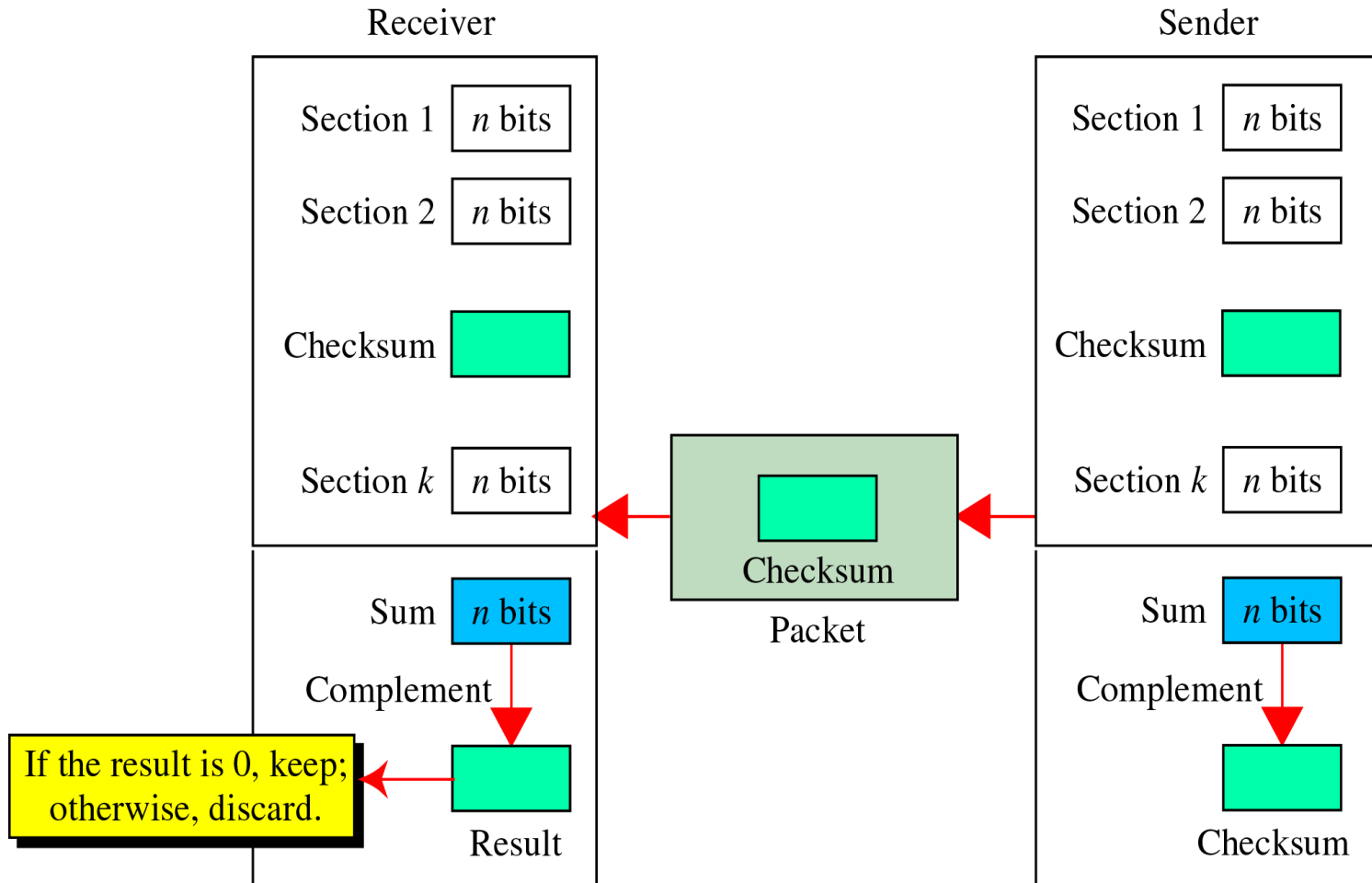
Receiver



Checksum

- ~ used by the higher layer protocols
- ~ is based on the concept of redundancy(VRC, LRC, CRC)

- Checksum Generator



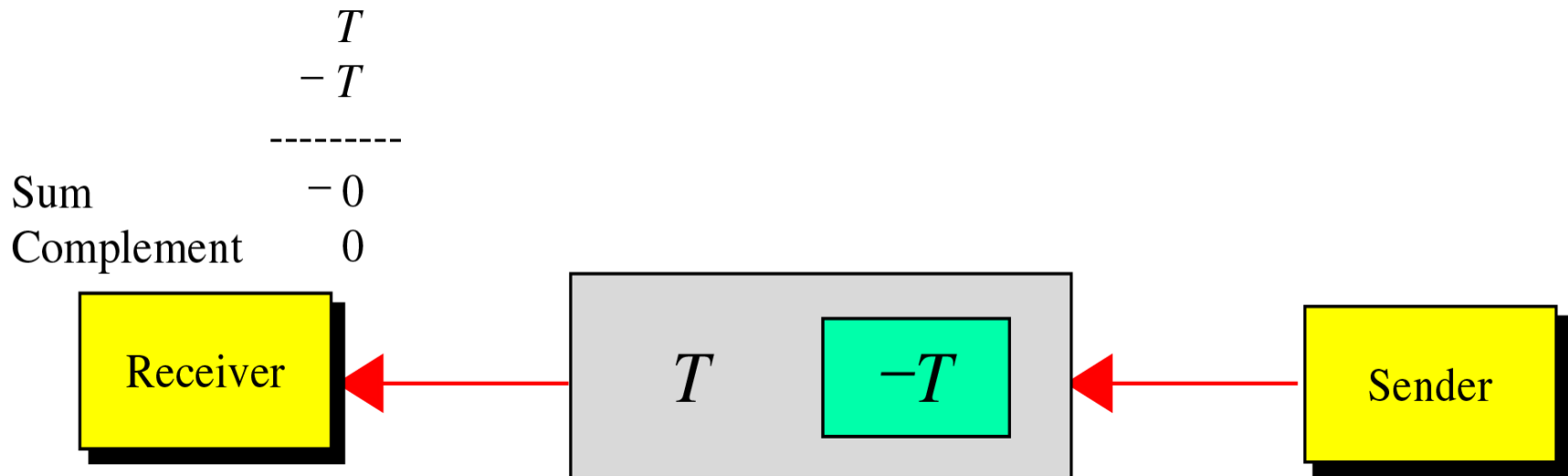
Detection(cont'd)

- To create the checksum the sender does the following:
 - The unit is divided into K sections, each of n bits.
 - 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)

- 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

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.

Hamming Code Generation

Single-bit error correction

To correct an error, the receiver reverses the value of the altered bit. To do so, it must know which bit is in error.

Number of redundancy bits needed

Let data bits = m

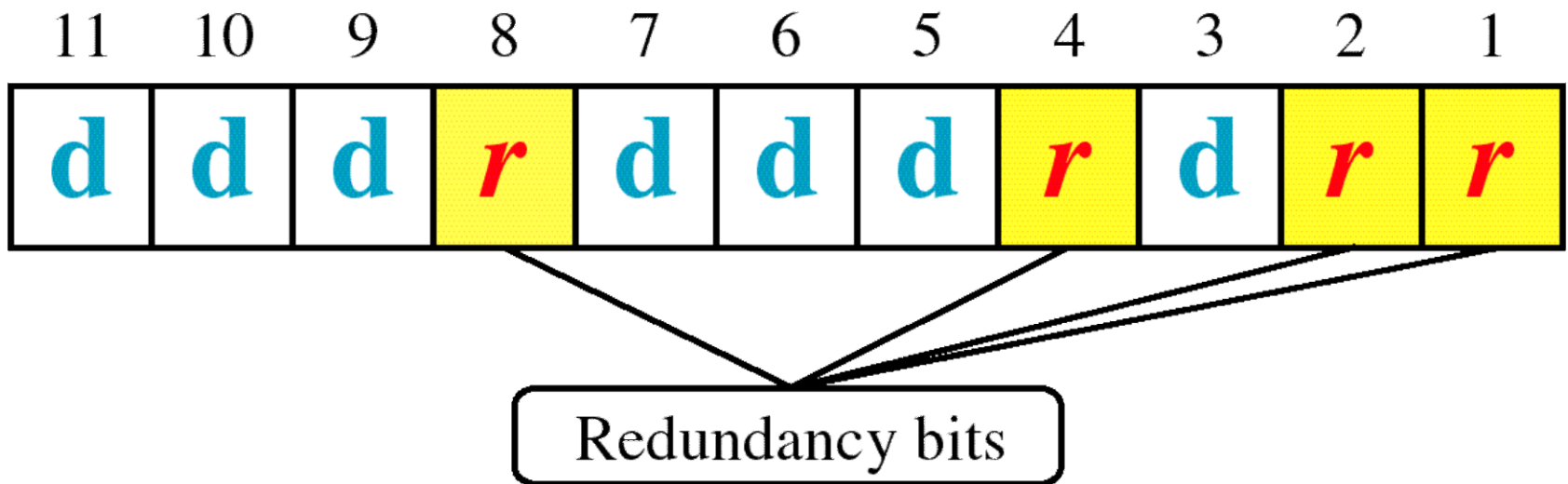
Redundancy bits = r

\therefore Total message sent = $m+r$

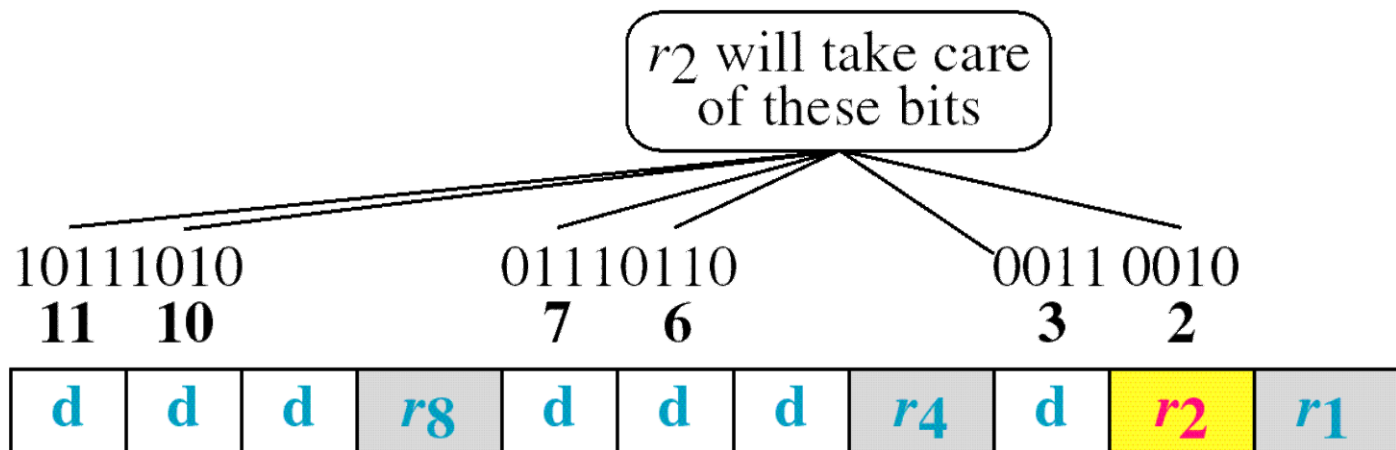
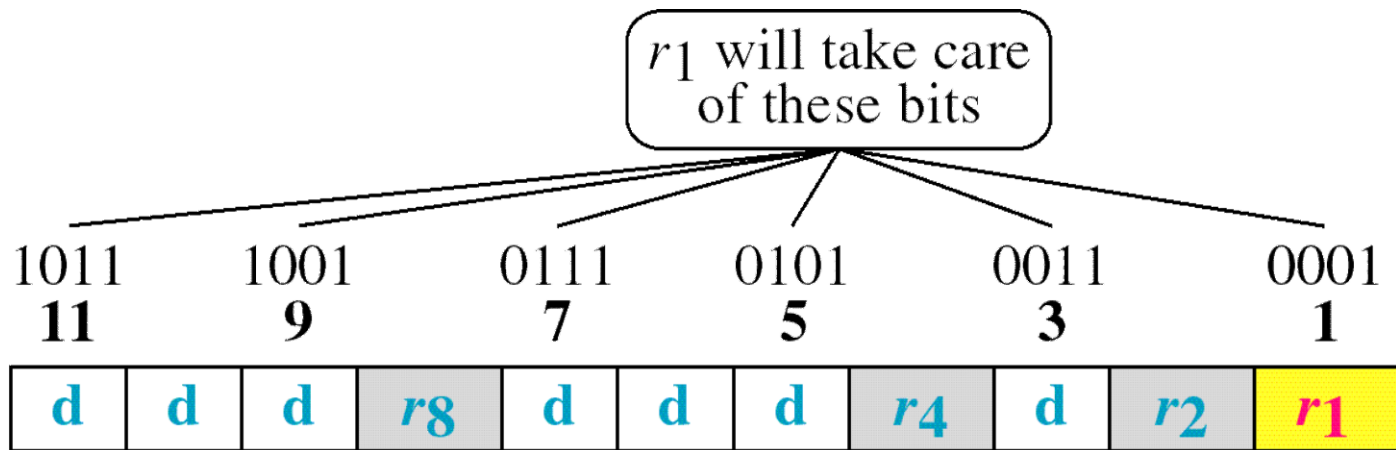
The value of r must satisfy the following relation:

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

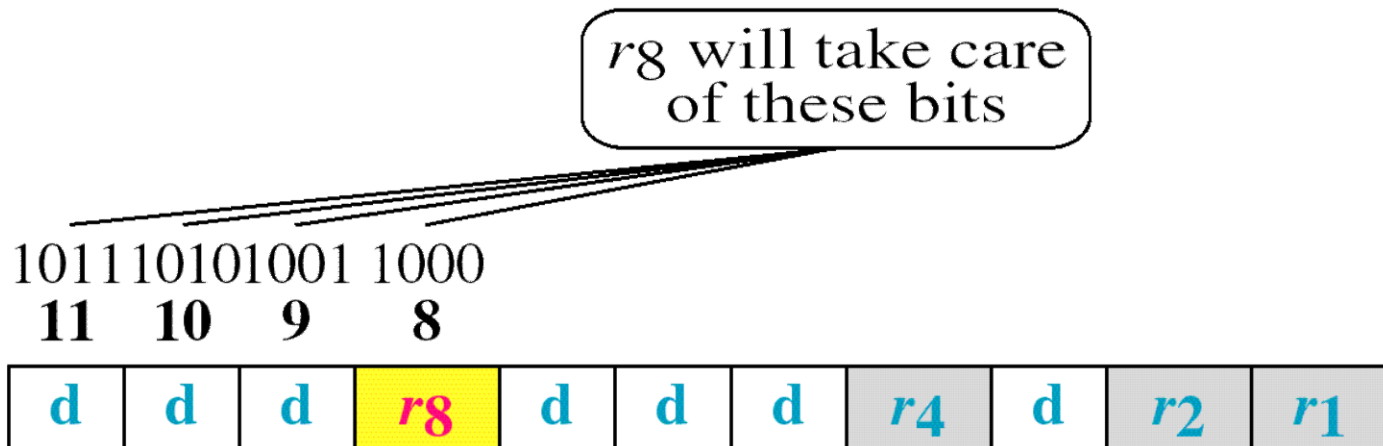
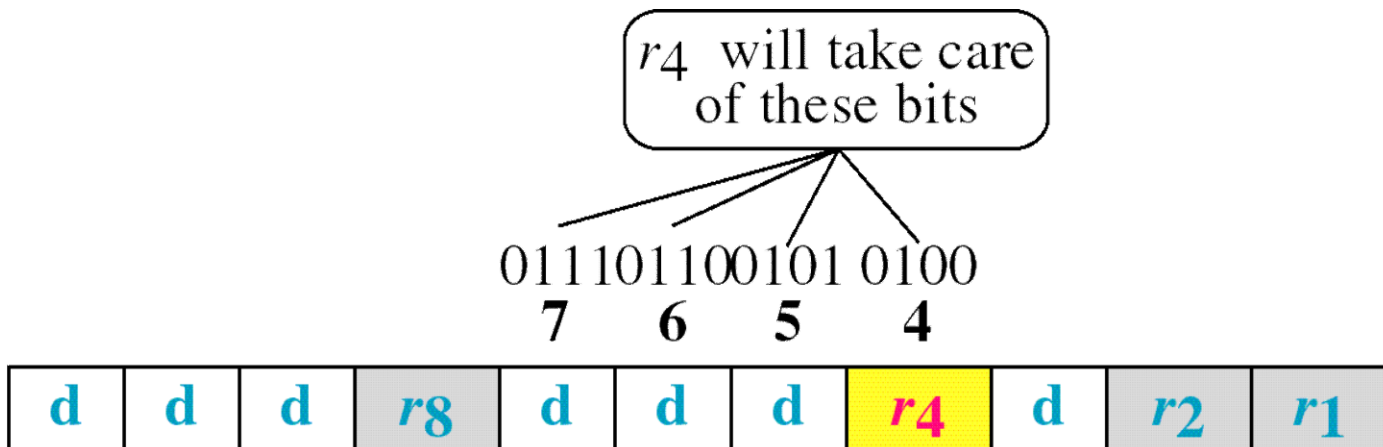
Hamming Code



Hamming Code



Hamming Code



Example of Hamming Code

Data: 1 0 0 1 1 0 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
---	---	---	--	---	---	---	---	---	---	---

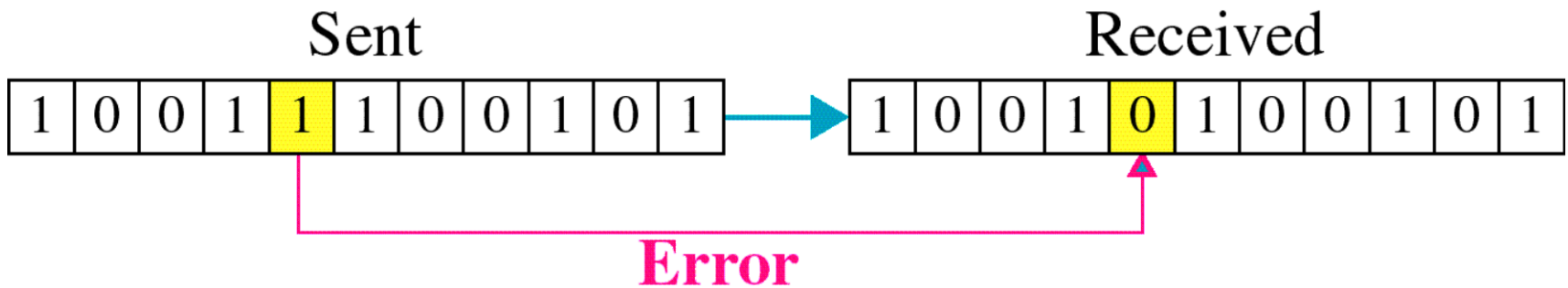
Adding r_8

1	0	0	1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---

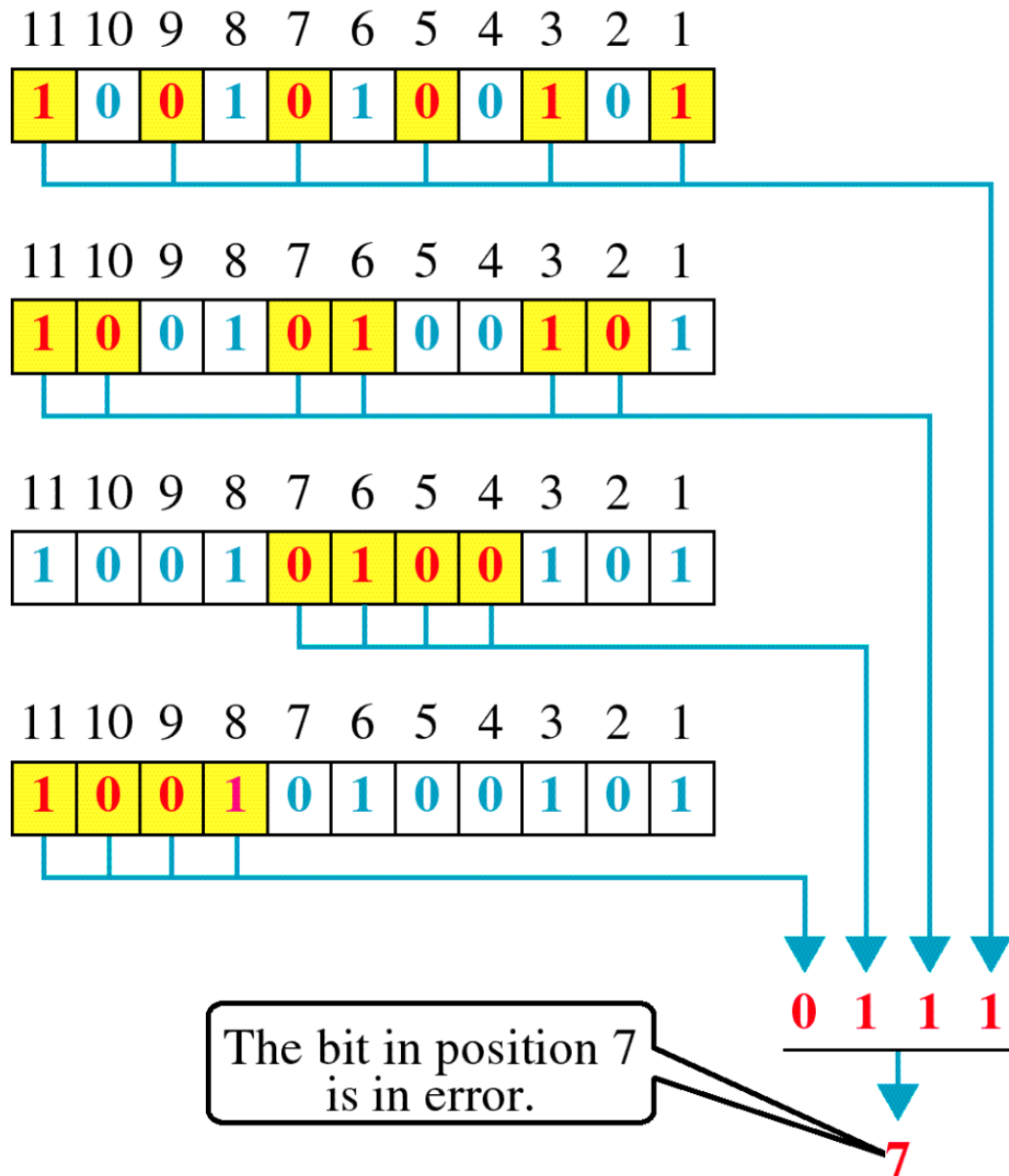


Code: 1 0 0 1 1 1 0 0 1 0 1

Single-bit error



Error Detection

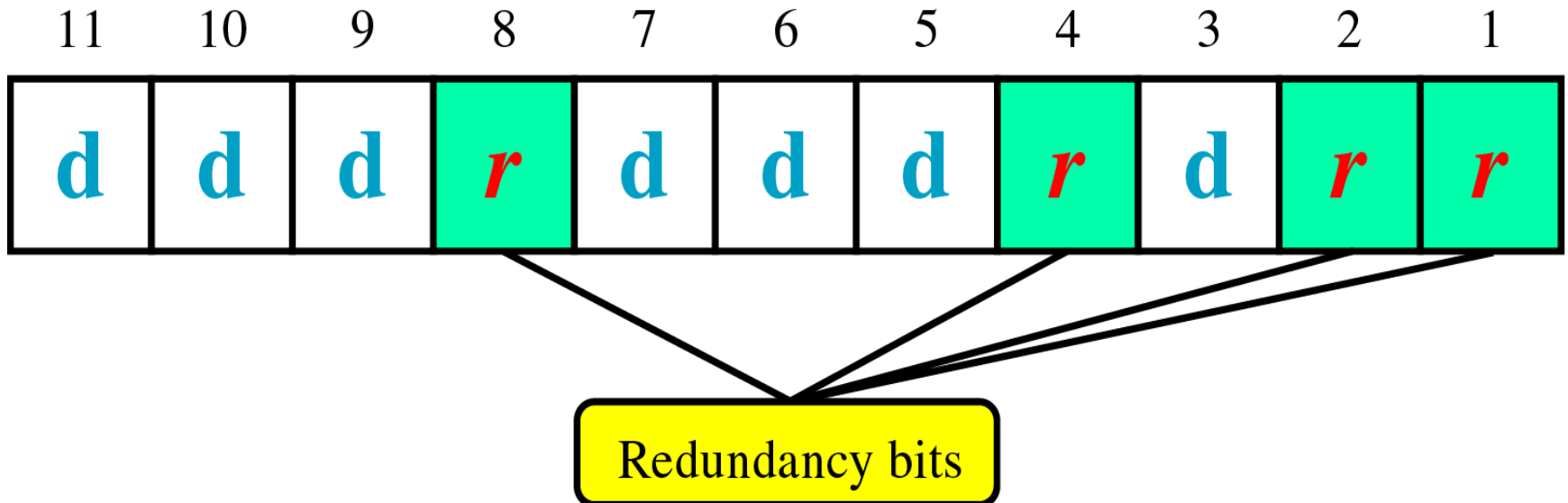


Error Correction(cont'd)

- Single-Bit Error Correction
 - parity bit
 - The secret of error correction is to locate the invalid bit or bits

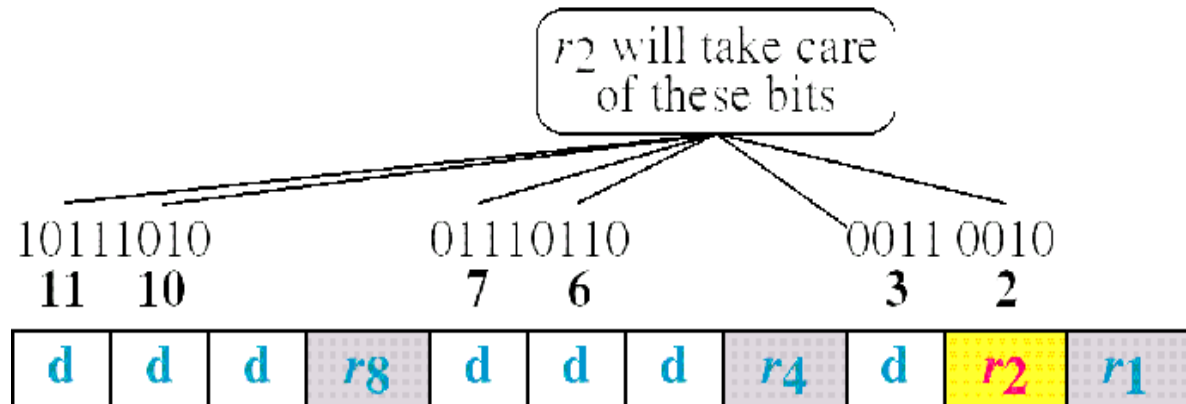
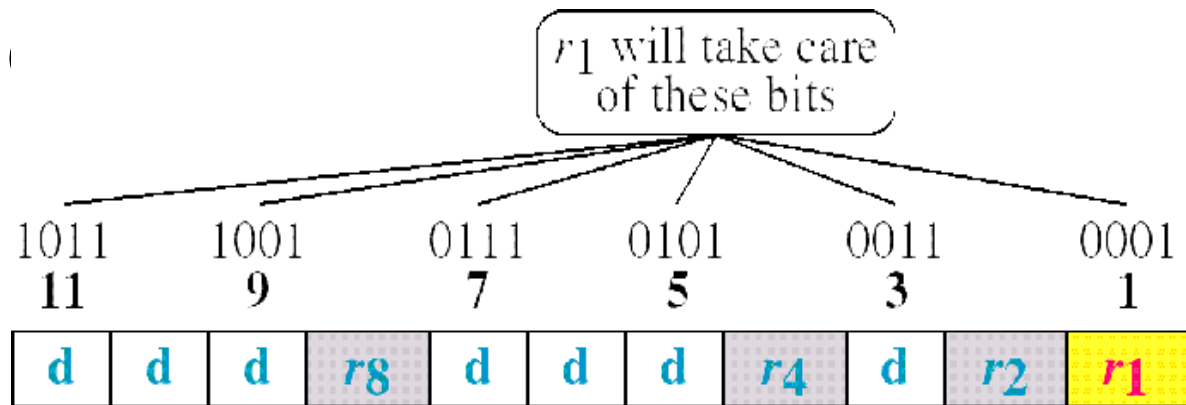
Error Correction(cont'd)

- Hamming Code
 - ~ developed by R.W.Hamming
- positions of redundancy bits in Hamming code



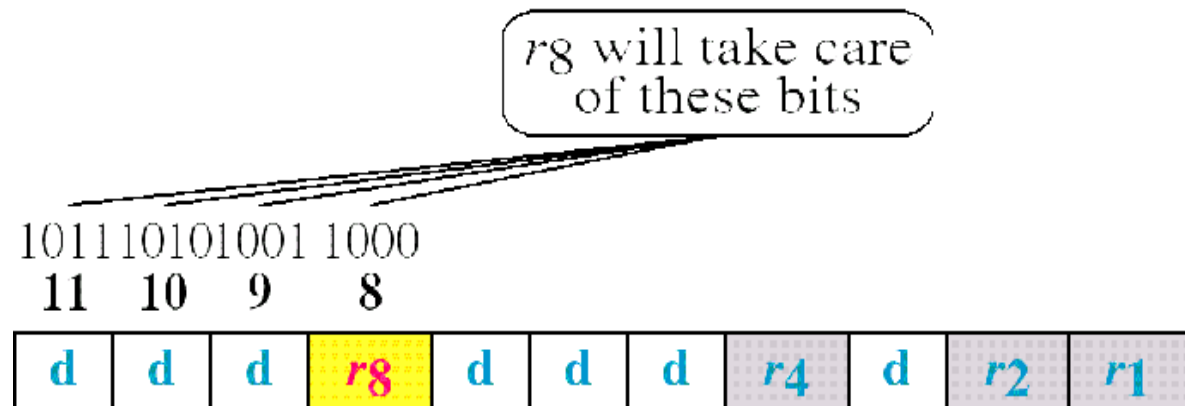
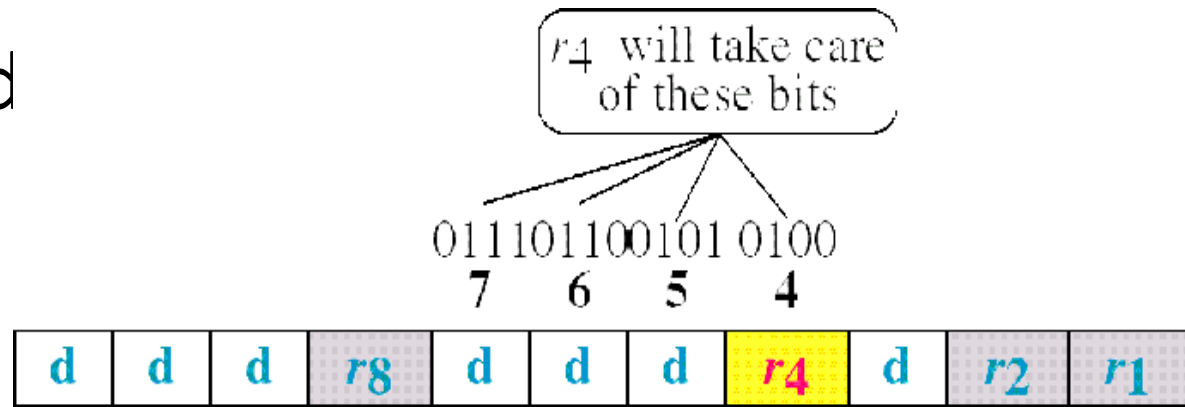
Error Correction(cont'd)

- Redundancy



Error Correction(cont'd)

- Redund



Error Correction(cont'd)

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

$$r_1 = \text{bits } 1, 3, 5, 7, 9, 11$$

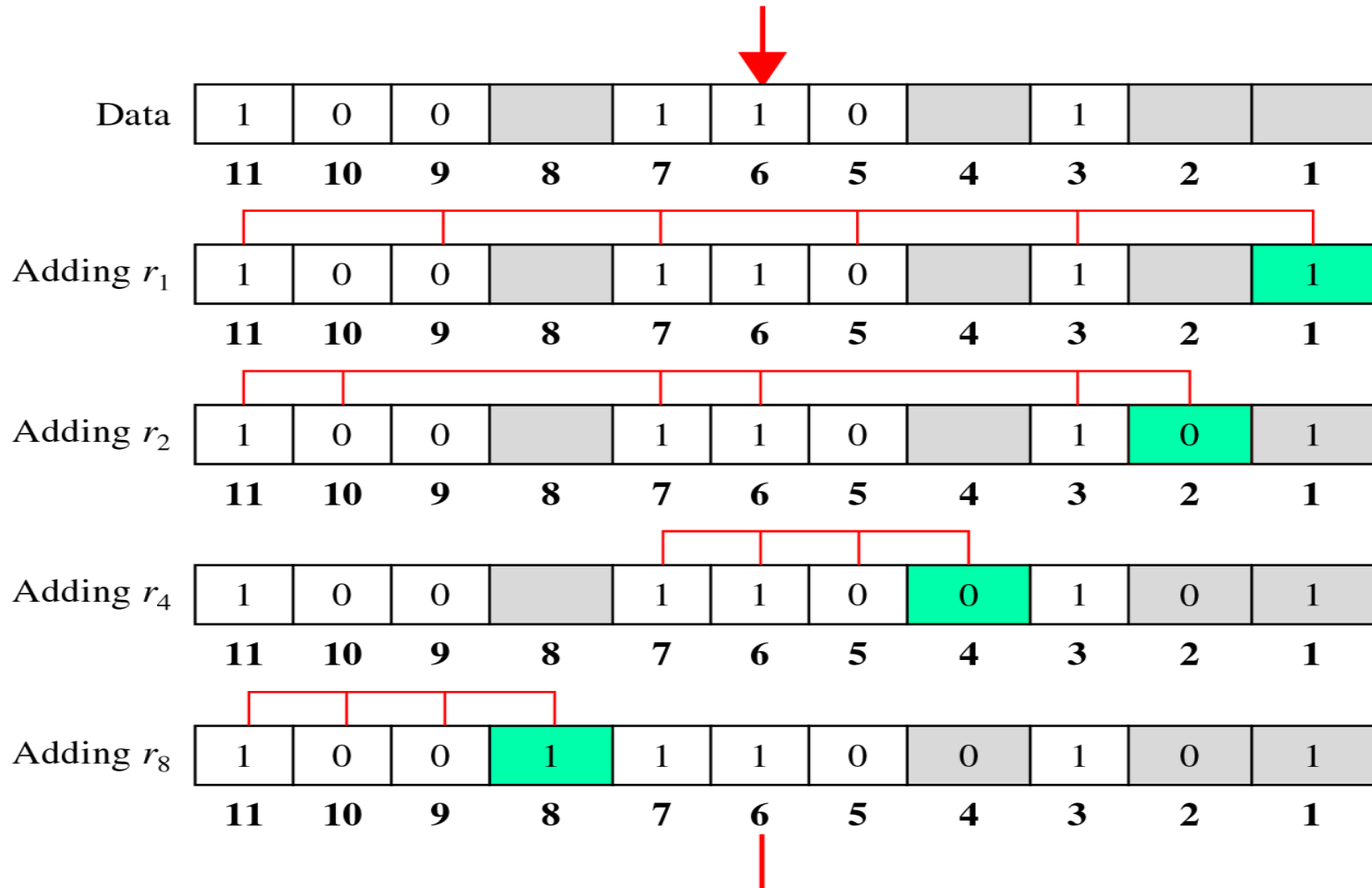
$$r_2 = \text{bits } 2, 3, 6, 7, 10, 11$$

$$r_4 = \text{bits } 4, 5, 6, 7$$

$$r_8 = \text{bits } 8, 9, 10, 11$$

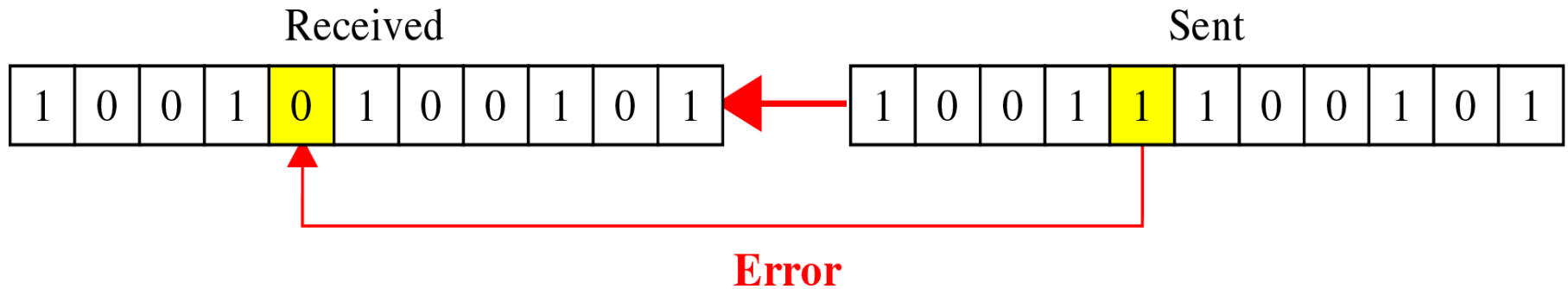
Sender Side Calculation

Data: 1 0 0 1 1 0 1

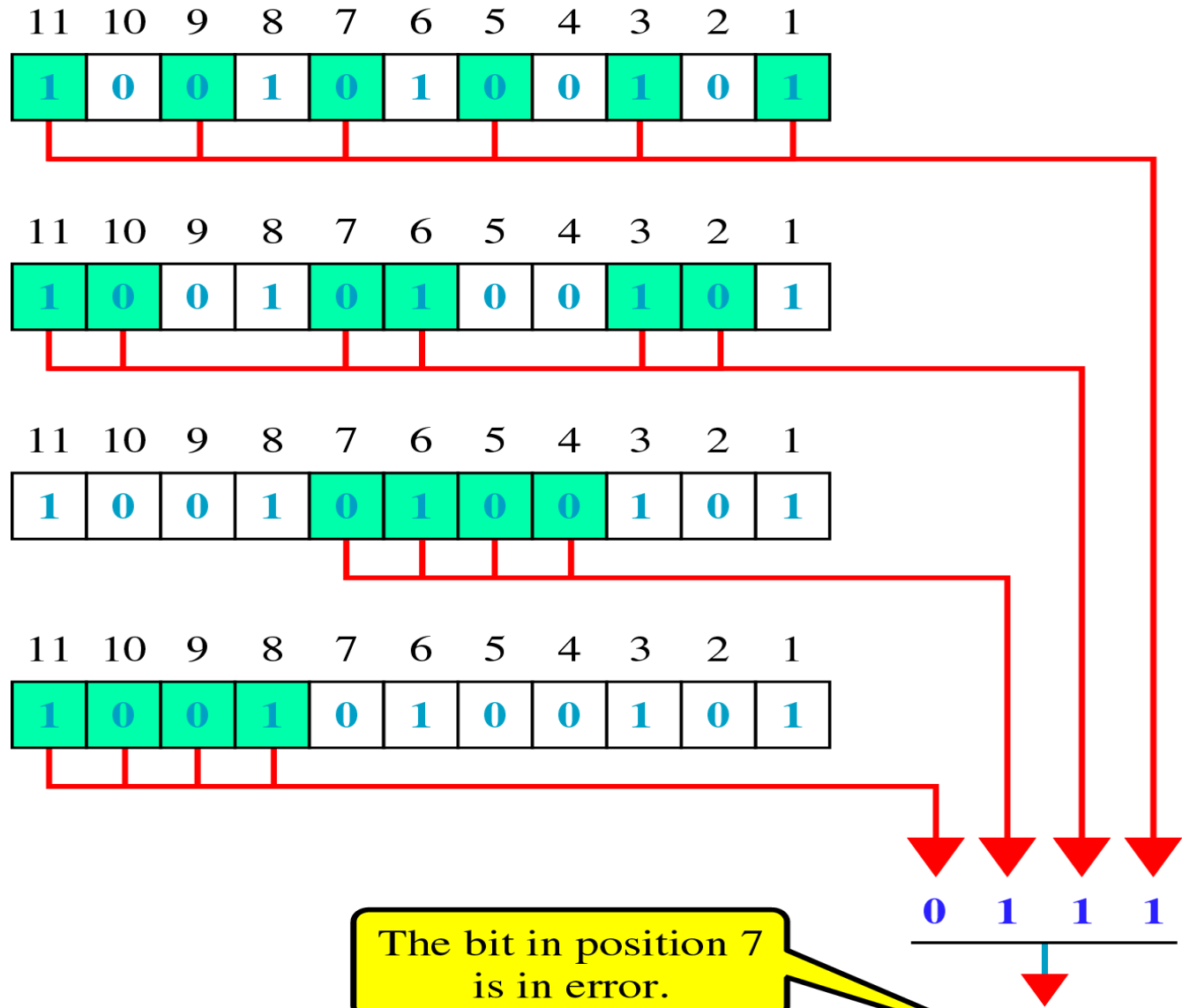


Error Correction(cont'd)

- Error Detection and Correction



Receiver Side Calculation



Practice Problems

- 1) Suppose the data to be transmitted is 1011001.
Generate the hamming code using the data word and assume 6th bit is corrupted during transmission.
Perform the receiver side calculation on the received message.
- 2) Differentiate between backward and forward error correction techniques. The source is transmitting a 32 bit of data as follows:

00110010 11101100 00101010 0001001

Engrave the steps involved at the sender side for finding the checksum. Ensure that the same data is received at the receiver side.

Practice Problems

3) Given the dataword 1010011110 and the divisor 10111, Show the generation of the codeword at the sender site (using binary division).

Suppose the leftmost bit of the message is inverted due to the noise on the transmission link.

What is the result of the receiver's CRC calculation? How does the receiver know that an error has occurred?