Error Detection and Correction

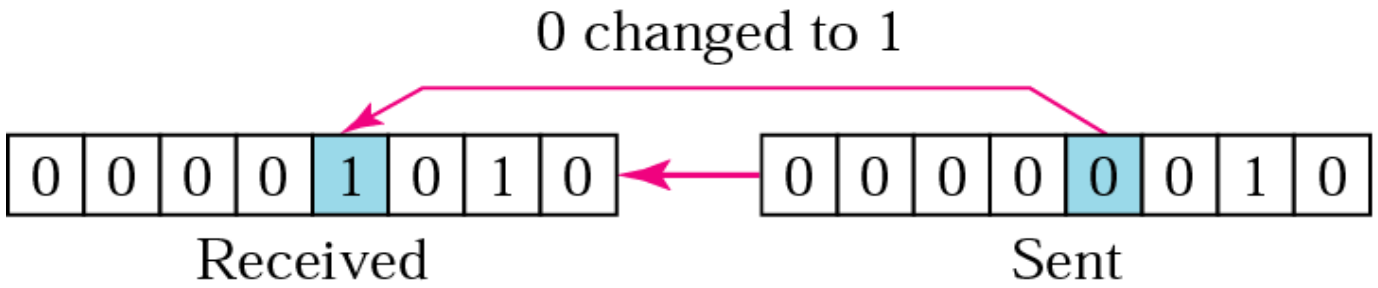
Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.

Types of Error

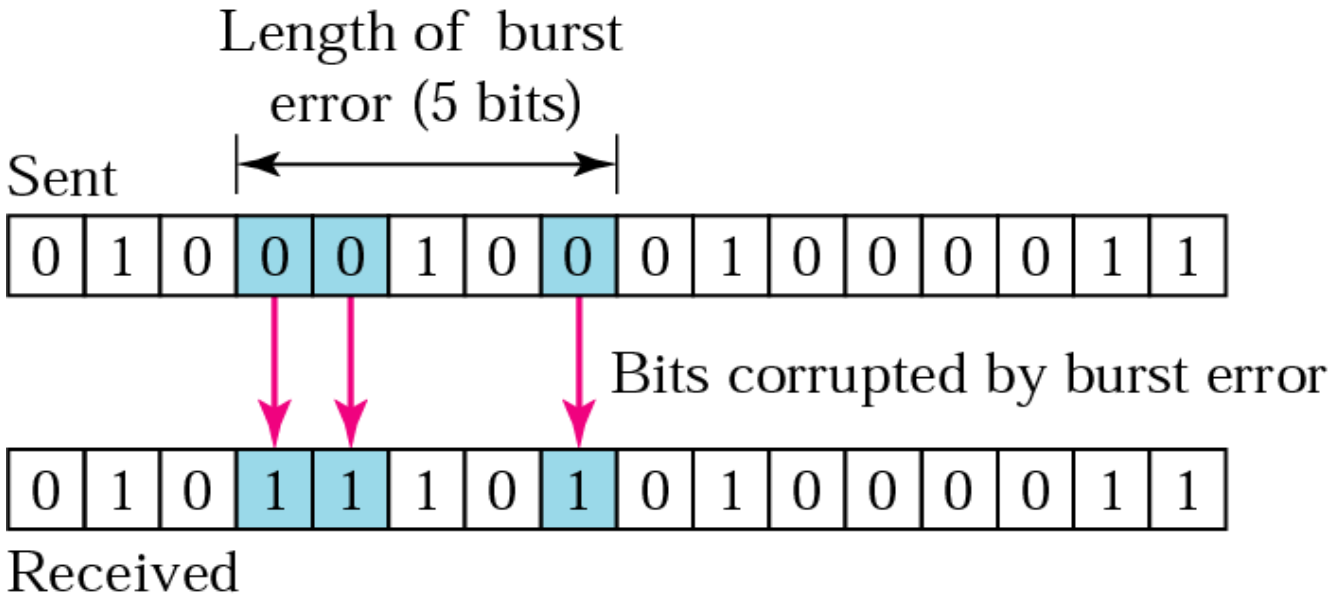
Single-Bit Error

Burst Error

In a single-bit error, only one bit in the data unit has changed.



A burst error means that 2 or more bits in the data unit have changed.



Detection

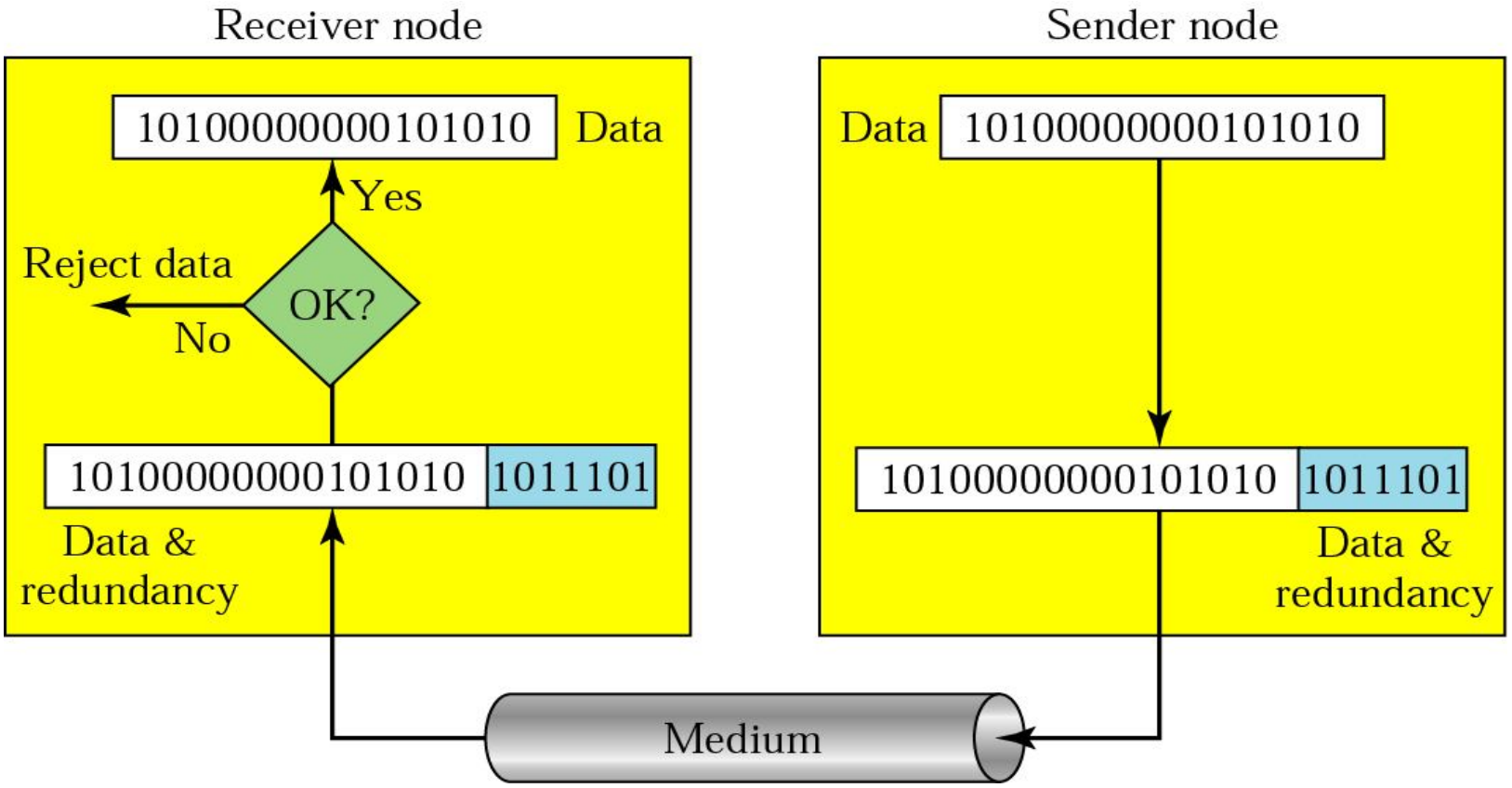
Redundancy

Parity Check

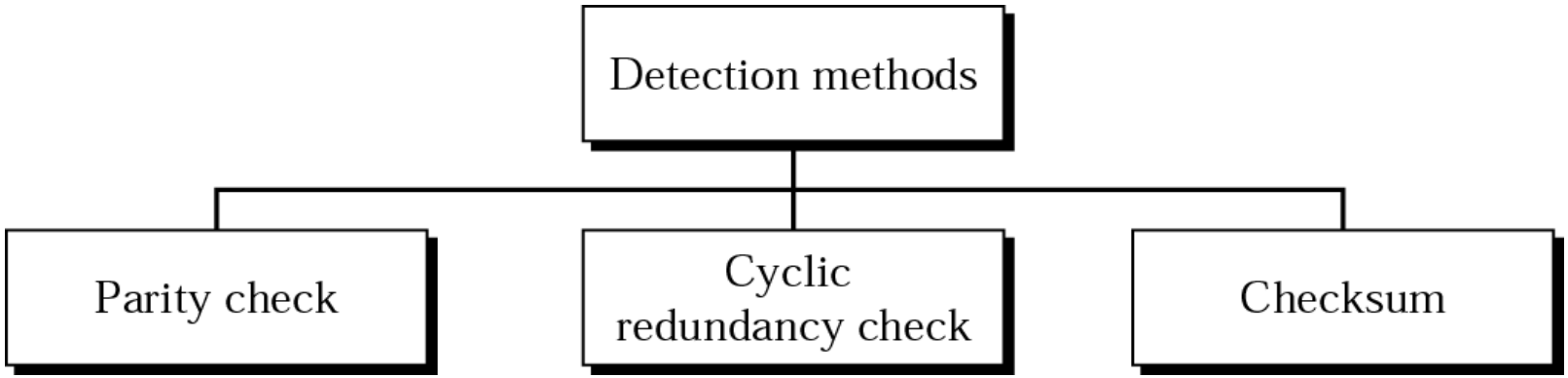
Cyclic Redundancy Check (CRC)

Checksum

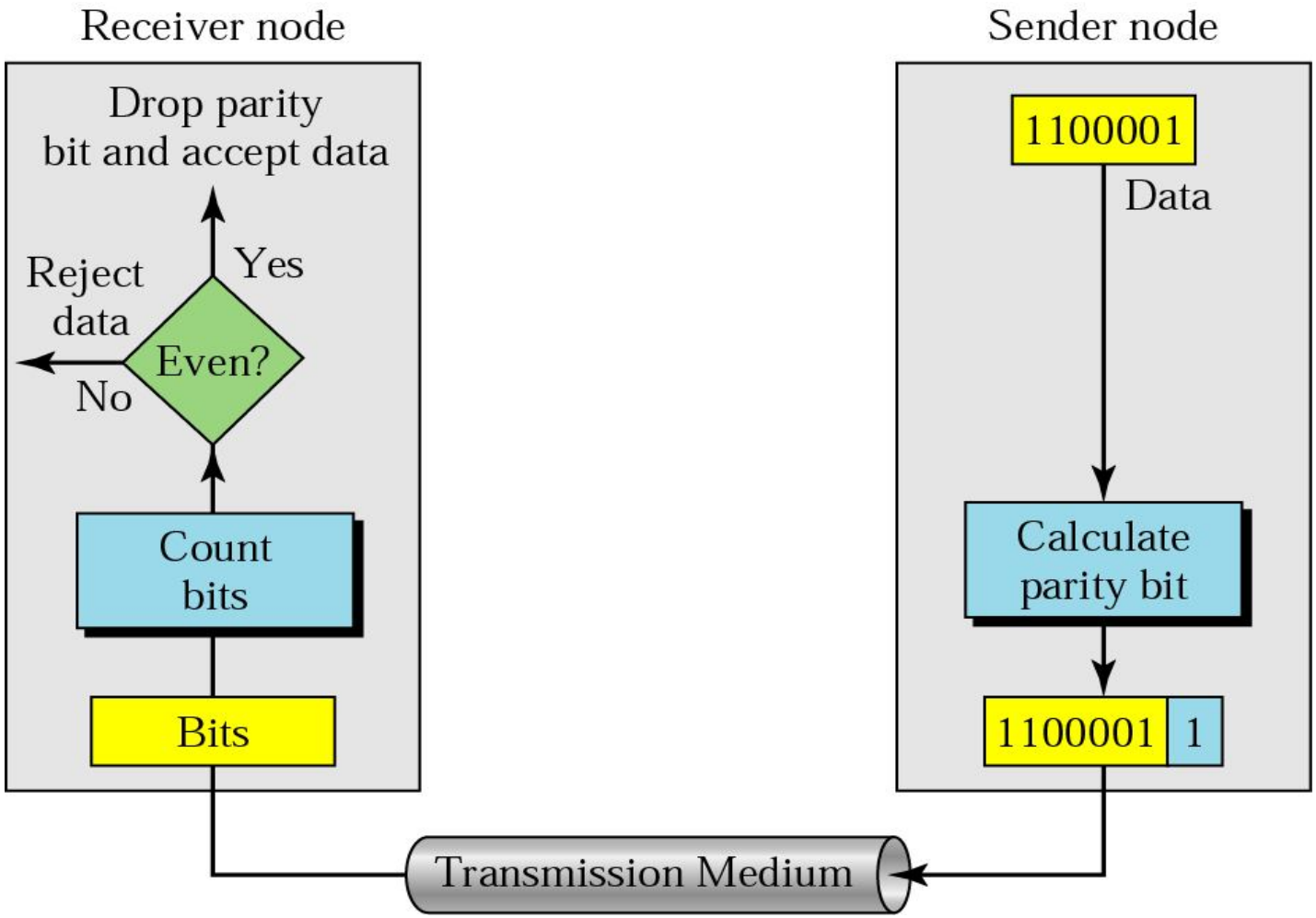
Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.



Detection methods



In 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).



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**

1. The data 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.

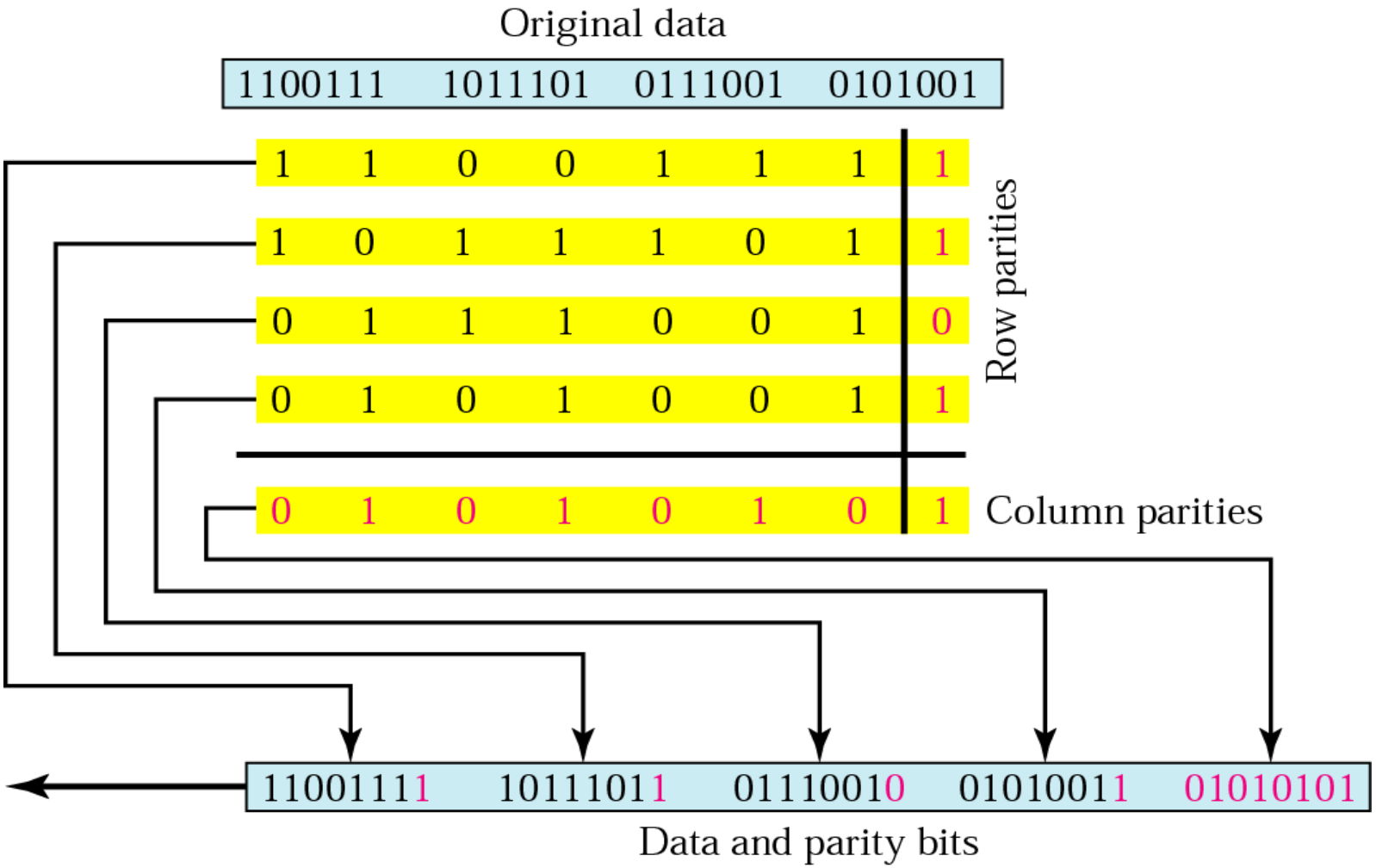
2. The data is corrupted during transmission.

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.

Simple parity check can detect all single-bit errors. It can detect burst errors only if the total number of errors in each data unit is odd.

Two-dimensional parity



Example

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.

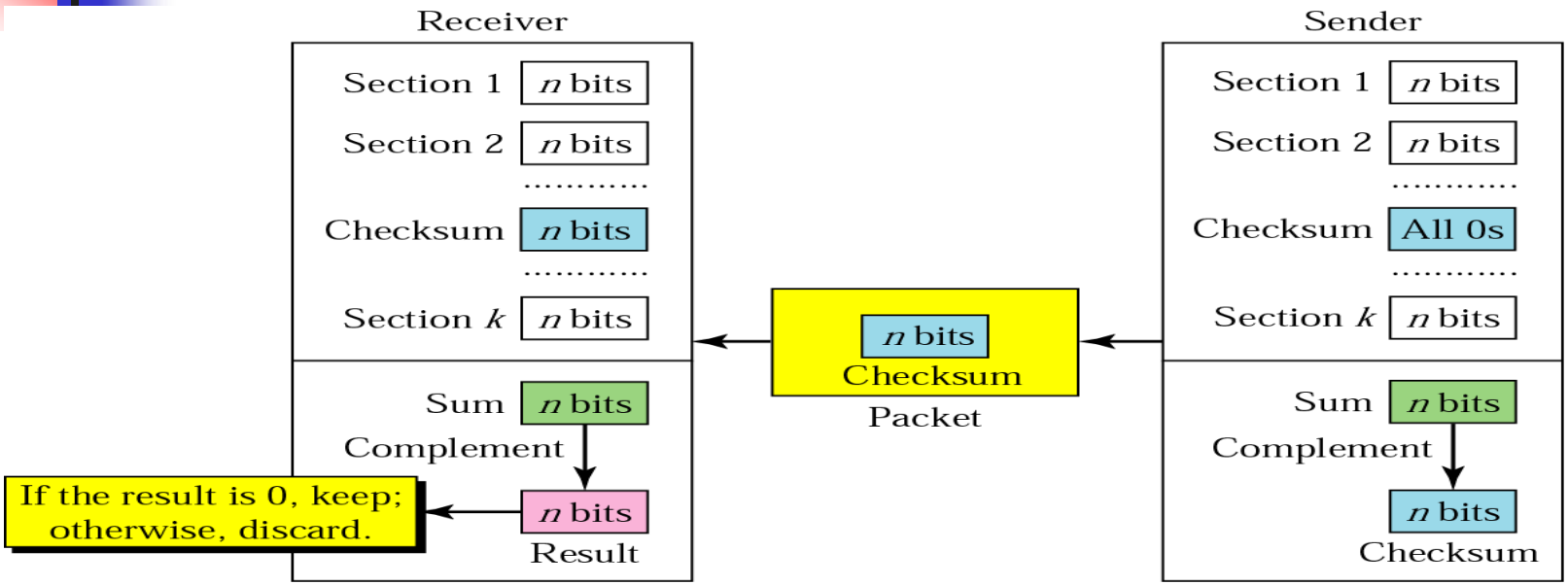
1010**0011** **1000**1001 11011101 11100111 10101010

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

In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Checksum



The receiver follows these steps:

The unit is divided into k sections, each of n bits.

All sections are added using one's complement to get the sum.

The sum is complemented.

If the result is zero, the data are accepted: otherwise, rejected.

The sender follows these steps:

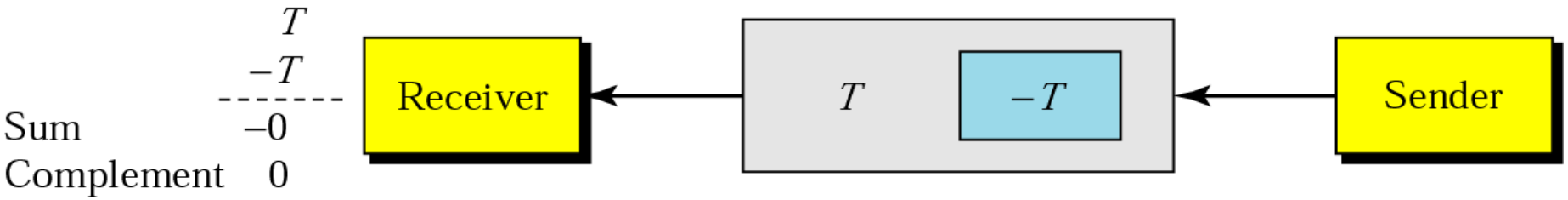
The unit is divided into k sections, each of n bits.

All sections are added using one's complement to get the sum.

The sum is complemented and becomes the checksum.

The checksum is sent with the data

Data unit and checksum



The sender follows these steps:

- *The unit is divided into k sections, each of n bits.*
- *All sections are added using one's complement to get the sum.*
- *The sum is complemented and becomes the checksum.*
- *The checksum is sent with the data.*

The receiver follows these steps:

- *The unit is divided into k sections, each of n bits.*
- *All sections are added using one's complement to get the sum.*
- *The sum is complemented.*
- *If the result is zero, the data are accepted: otherwise, rejected.*

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 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

Now suppose there is a burst error of length 5 that affects 4 bits.

10101**111 11**111001 00011101

When the receiver adds the three sections, it gets

	10101**111**
	11111001
	00011101
Partial Sum	**1** 11000101
Carry	**1**
Sum	11000110
Complement	**00111001** **the pattern is corrupted.**

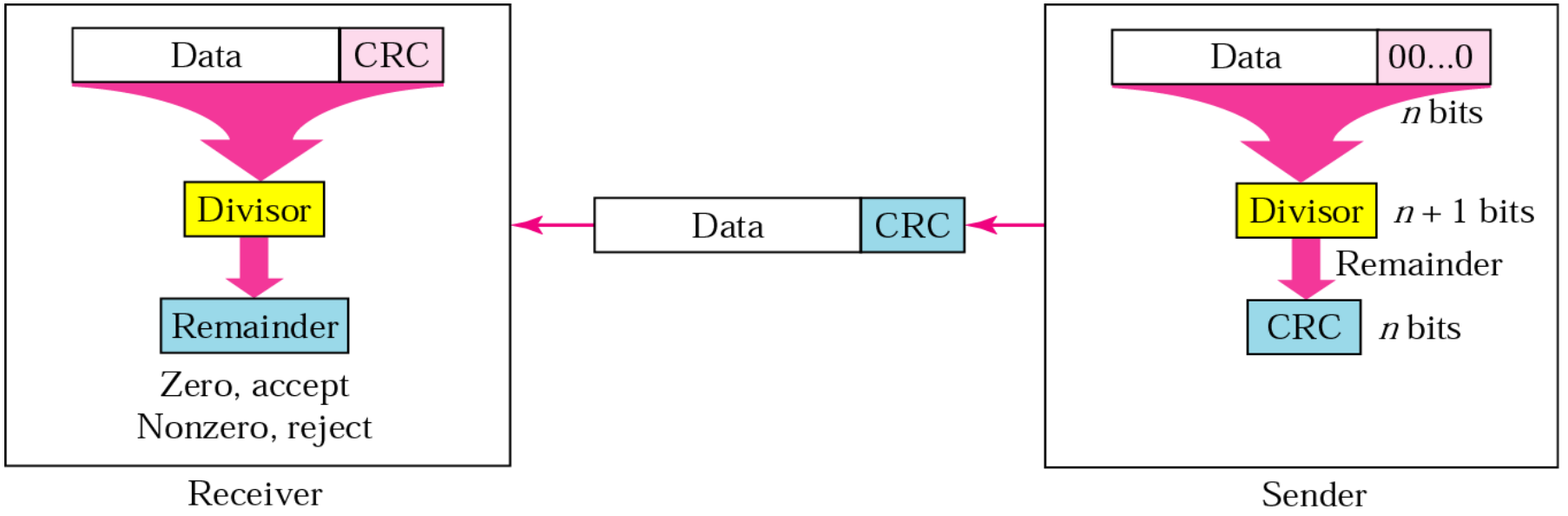
Cyclic redundancy checks for error detection (CRC)

- **CRC is a rather unusual but clever method that does error checking via polynomial division.**
- **The following steps outline the CRC method:**
  - **Given a bit String, append several 0s to the end of it and call it B.** ***let B(x) be polynomial.***
  - **Divide B(x) by some agreed-on polynomial G(x). And determine the remainder R(x).**
  - **Define T(x)=B(x)-R(x). Or T(x)=B(x)+R(x).**
  - **Transmit T, the bit string corresponding to T(x)**
  - **Let T' represent the bit stream the receiver gets, T'(x)/G(x), if there is a 0 remainder, then no error.**

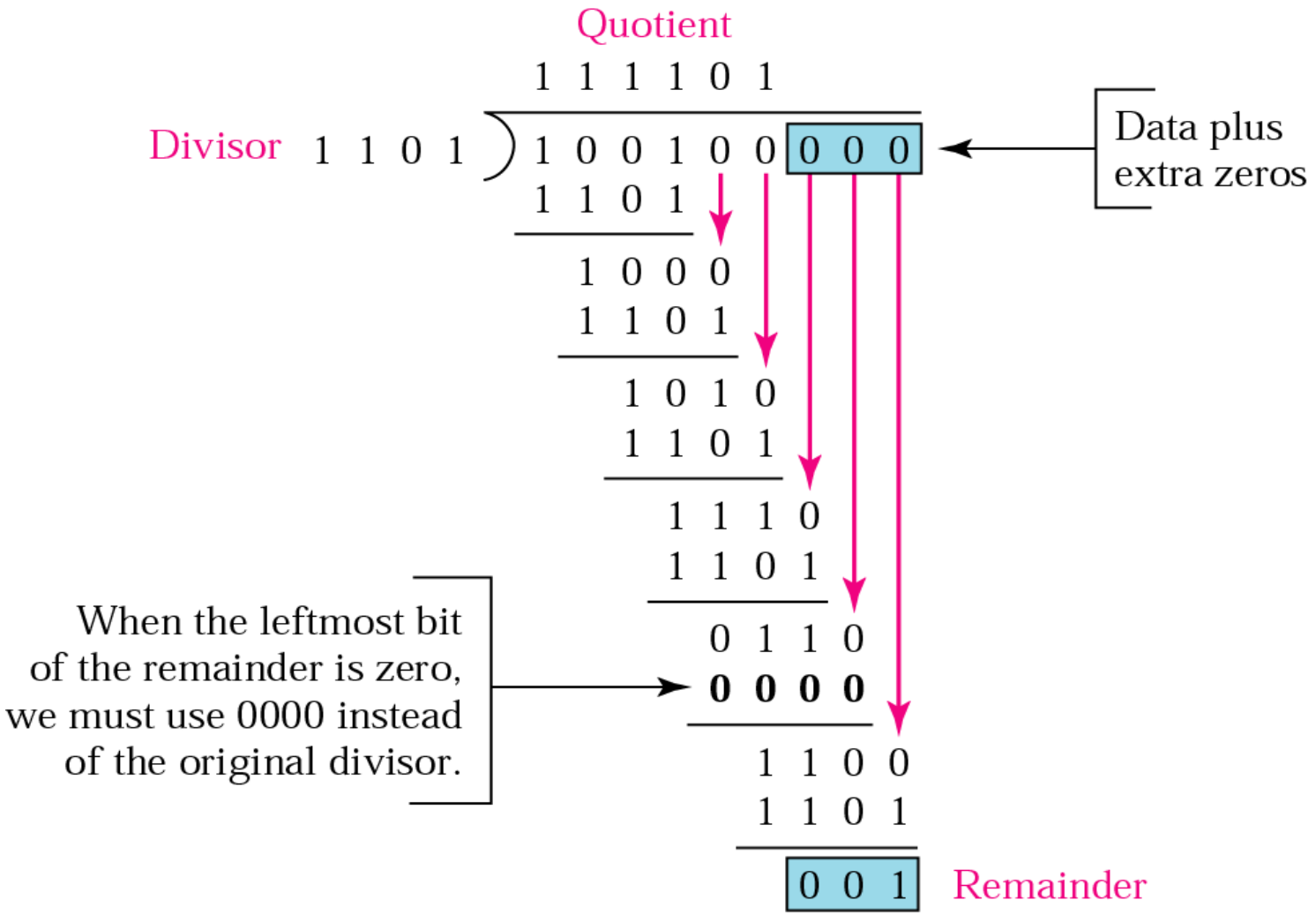
Caution: modulo 2 arithmetic:

0-0=0 .1-0=1, 0-1=1, 1-1=0

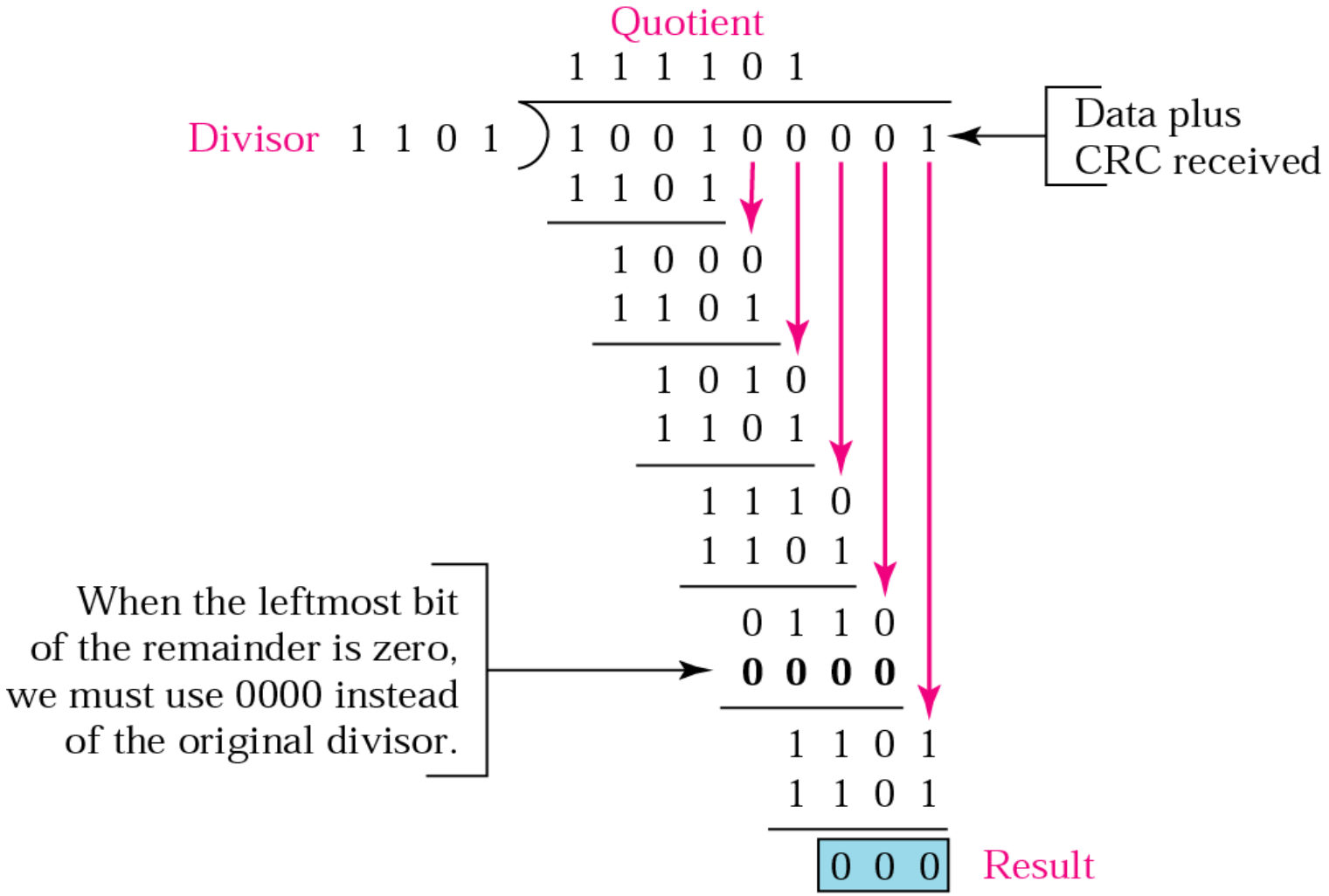
CRC generator and checker



Binary division in a CRC generator



Binary division in CRC checker



A polynomial

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

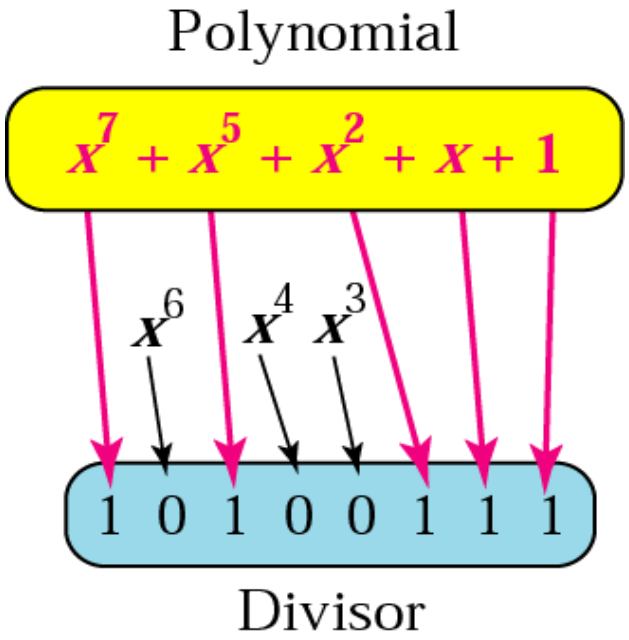


Table : Standard polynomials

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	**ATM header**
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	**ATM AAL**
ITU-16	$x^{16} + x^{12} + x^5 + 1$	**HDLC**
ITU-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	**LANs**

Example

It is obvious that we cannot choose **x** (binary 10) or **$x^2 + x$** (binary 110) as the polynomial because both are divisible by x.

However, we can choose **x + 1** (binary 11) because it is not divisible by x, but is divisible by x + 1.

We can also choose **$x^2 + 1$** (binary 101) because it is divisible by x + 1 (binary division).

Example

The CRC-12

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

which has a degree of 12, will detect all burst errors affecting an odd number of bits, will detect all burst errors with a length less than or equal to 12, and will detect, 99.97 percent of the time, burst errors with a length of 12 or more.

Analysis of CRC

- Whether the method is any good?
- Will the receiver always be able to detect a damaged frame?
- The answer: when will $(T(x)+E(x))/G(x)$ generate a zero remainder? The error cann't be detected.
- Because: $(T(x)+E(x))/G(x) = T(x)/G(x) + E(x)/G(x)$

  $E(x)/G(x) = 0$.

  So: undetected transmission errors correspond to errors for which G(x) is a factor of E(x).

Consider: under what conditions is G(x) a factor of E(x)?

Analysis of CRC

under what conditions is G(x) a factor of E(x)?

$$\frac{E(x)}{G(x)} = \frac{x^i \cdot (x^{k-1} + \ldots + 1)}{G(x)}$$

We assume that k-1 >= r = degree G(x), it is possible That G(x) is a factor of ($x^i \cdot (x^{k-1} + \ldots + 1)$).

Consider: what are the chances this will happen?

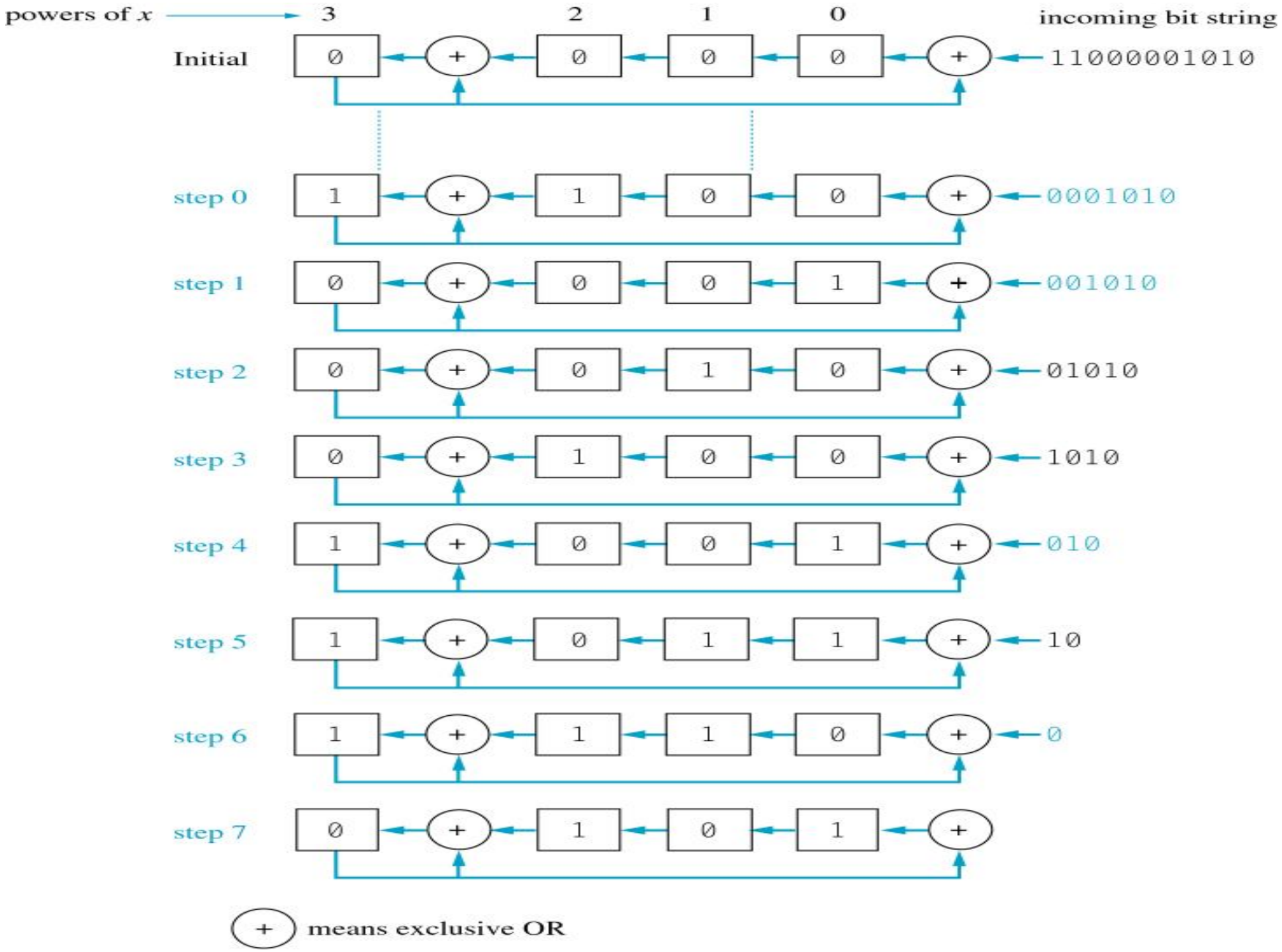
when r = k-1, the probability is $1/2^{r-1}$
when r < k-1, the probability is $1/2^{r}$

Analysis of CRC

- CRC is very effective if G(x) is chosen properly.
- CRC-12,CRC-16,CRC-ITU,CRC-32.
- G(x) should be chosen so that x is not a factor but x+1 is a factor.
- In This Case, CRC Dectects the following errors:
  - All burst errors of length r less than degree G(x)
  - All burst errors affecting an odd number of bits
  - All burst errors of length equal to r+1 with probability $(2^{r-1} - 1)/2^{r-1}$
  - All burst errors of length greater than r+1 with probability $(2^{r-1} - 1)/2^{r-1}$

CRC implementation using circular shifts

- Can we divide two polynomials and get the remainder quickly?
- Caution: we really need is the remainder. The quotient was never used.
- One widely used CRC implementation uses a circuit that is constructed depending on the generator polynomial G(x).


powers of x
3
2
1
0
incoming bit string
Initial
0 + 0 0 0 + 11000001010
step 0
1 + 1 0 0 + 0001010
step 1
0 + 0 0 1 + 001010
step 2
0 + 0 1 0 + 01010
step 3
0 + 1 0 0 + 1010
step 4
1 + 0 0 1 + 010
step 5
1 + 0 1 1 + 10
step 6
1 + 1 1 0 + 0
step 7
0 + 1 0 1 +
+ means exclusive OR

Correction

Retransmission

Forward Error Correction

Burst Error Correction

Hamming codes:error correction

- Errors are detected there are typically two choices:
  - resend or
  - fix the damaged frame.
- Single-bit error correction
- Multiple-bit error correction

Single-bit error correction

- Hamming code requires the insertion of multiple parity bits in the bit string before sending.
- How many parity bits for parity do we use?

  in general, if n parity checks are used, there are 2^n possible combinations.
- We must associate each bit position with a unique combination to allow the receiver to analyze the parity checks and conclude where an error occurred.

To calculate the no of redundancy bits r required to correct a given no of data bits m we must find a relationship between m and r

$$2^r >= m+r+1$$

Table : 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

Positions of redundancy bits in Hamming code

To associate a combination with a unique event, we insert four parity into the frame as shown.

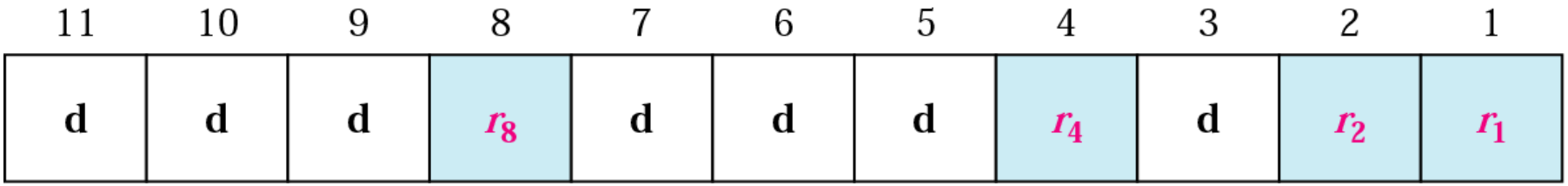
For a 7 bit ASCII code required 4 redundancy bits that can be added to the end of the data unit or interspersed with the original data bits.

R1= parity of the bits (1,3,5,7,9,11)

R2=parity of the bits (2,3,6,7,10,11)

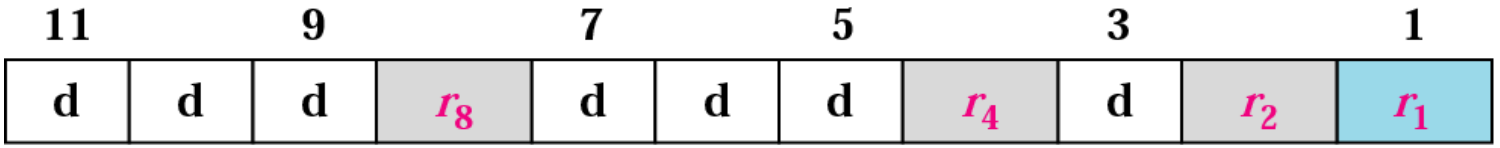
R3=parity of the bits (4,5,6,7)

R4=parity of the bits (8,9,10,11)

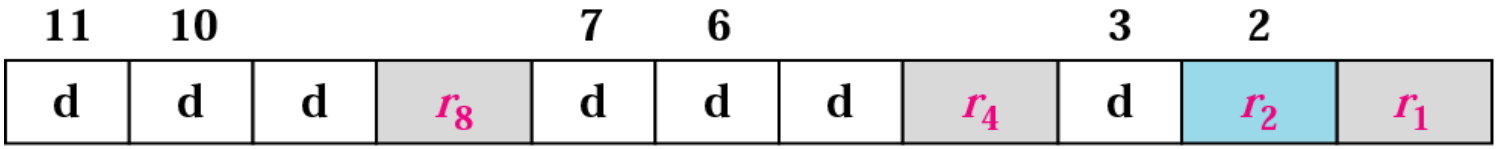


Redundancy bits calculation

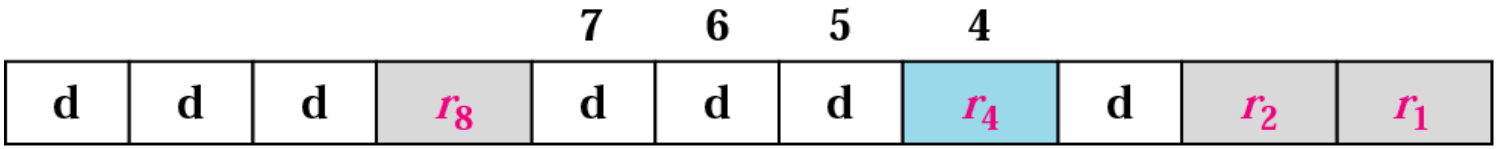
r_1 will take care of these bits.



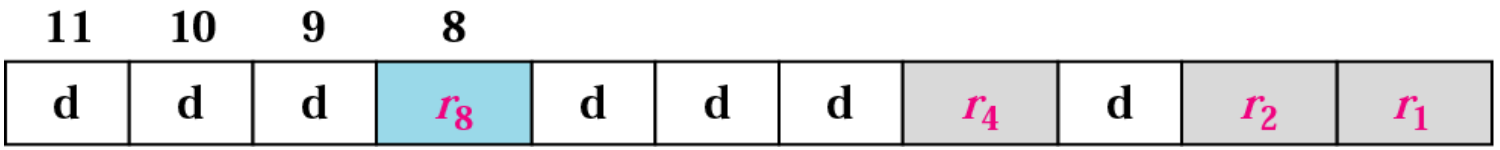
r_2 will take care of these bits.



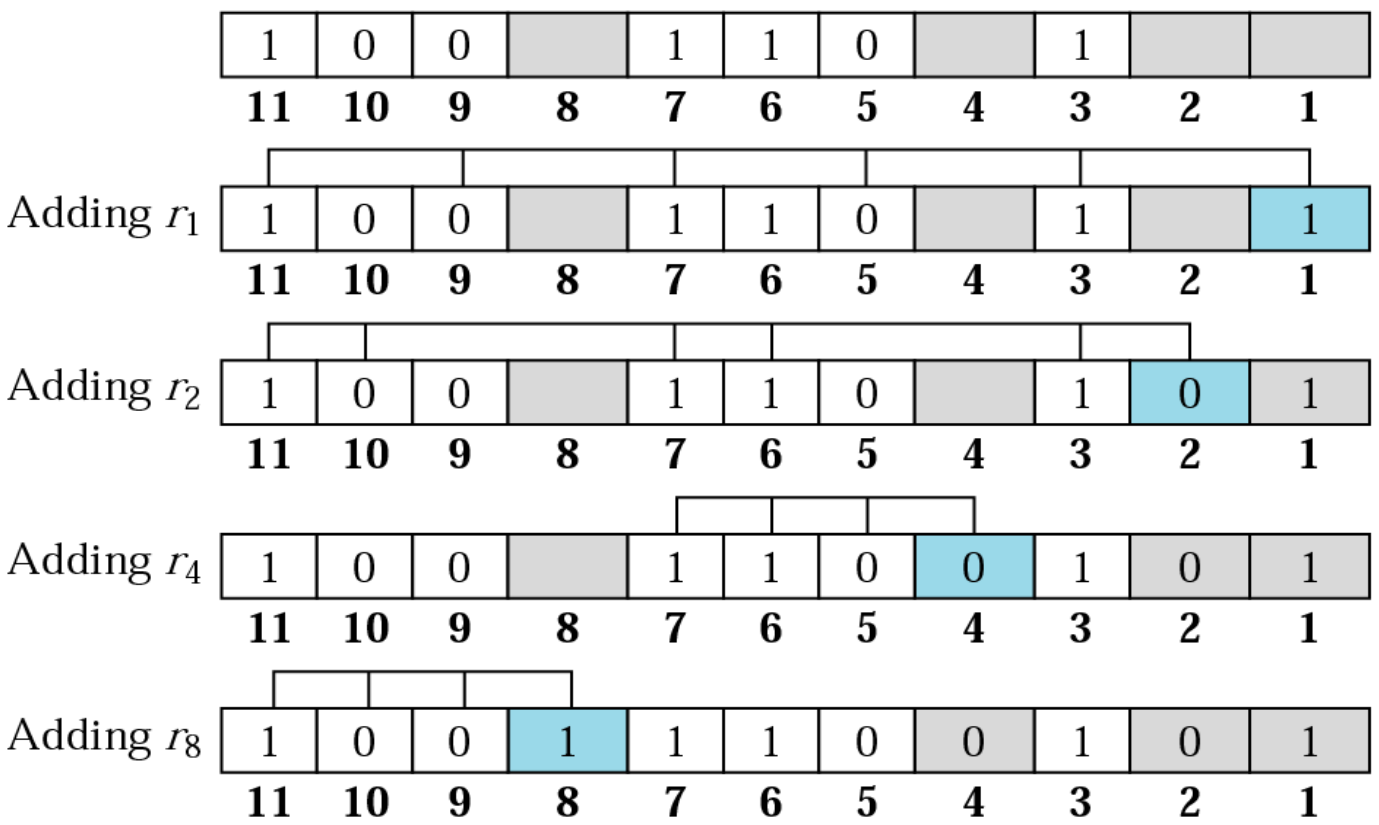
r_4 will take care of these bits.



r_8 will take care of these bits.



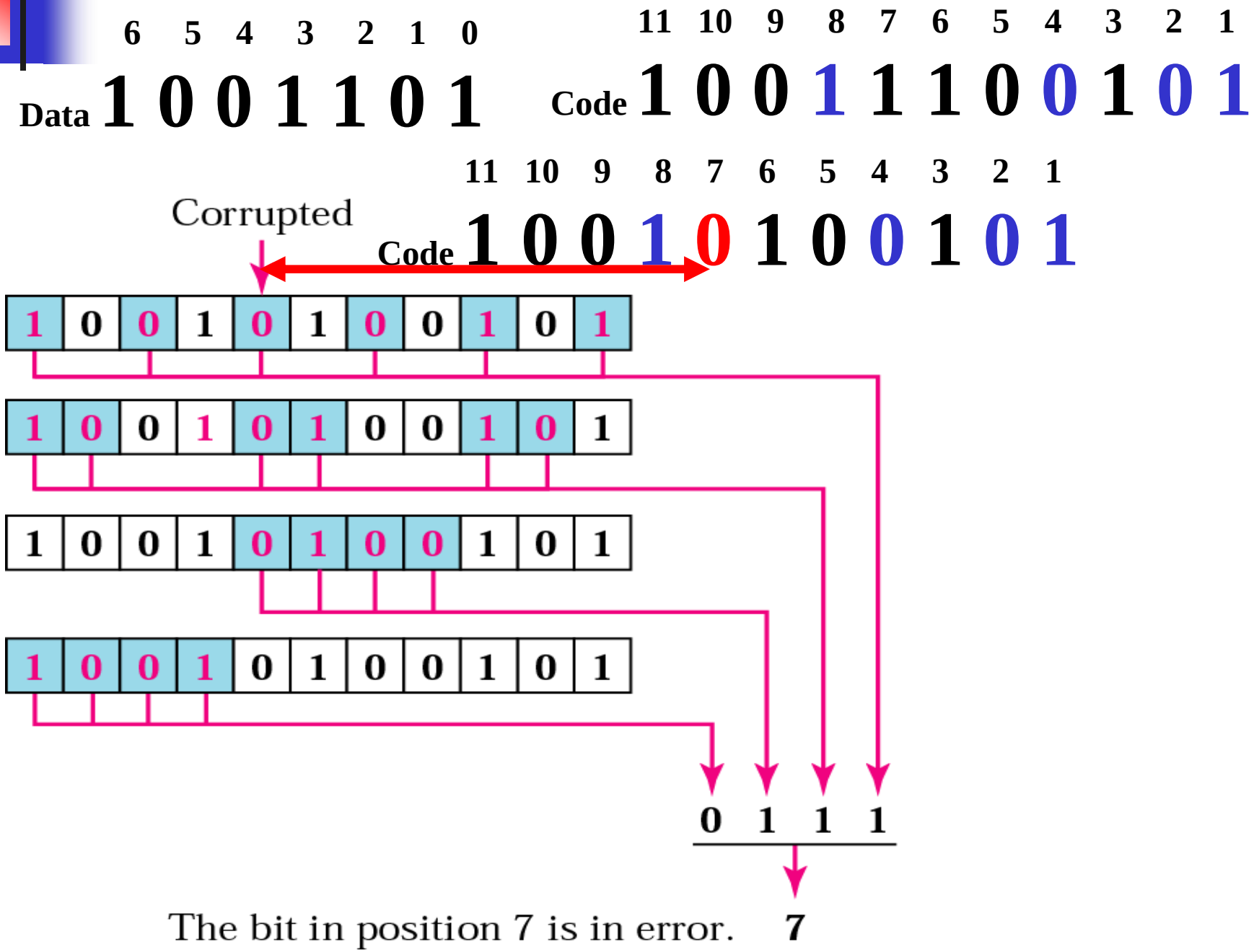
Example of redundancy bit calculation



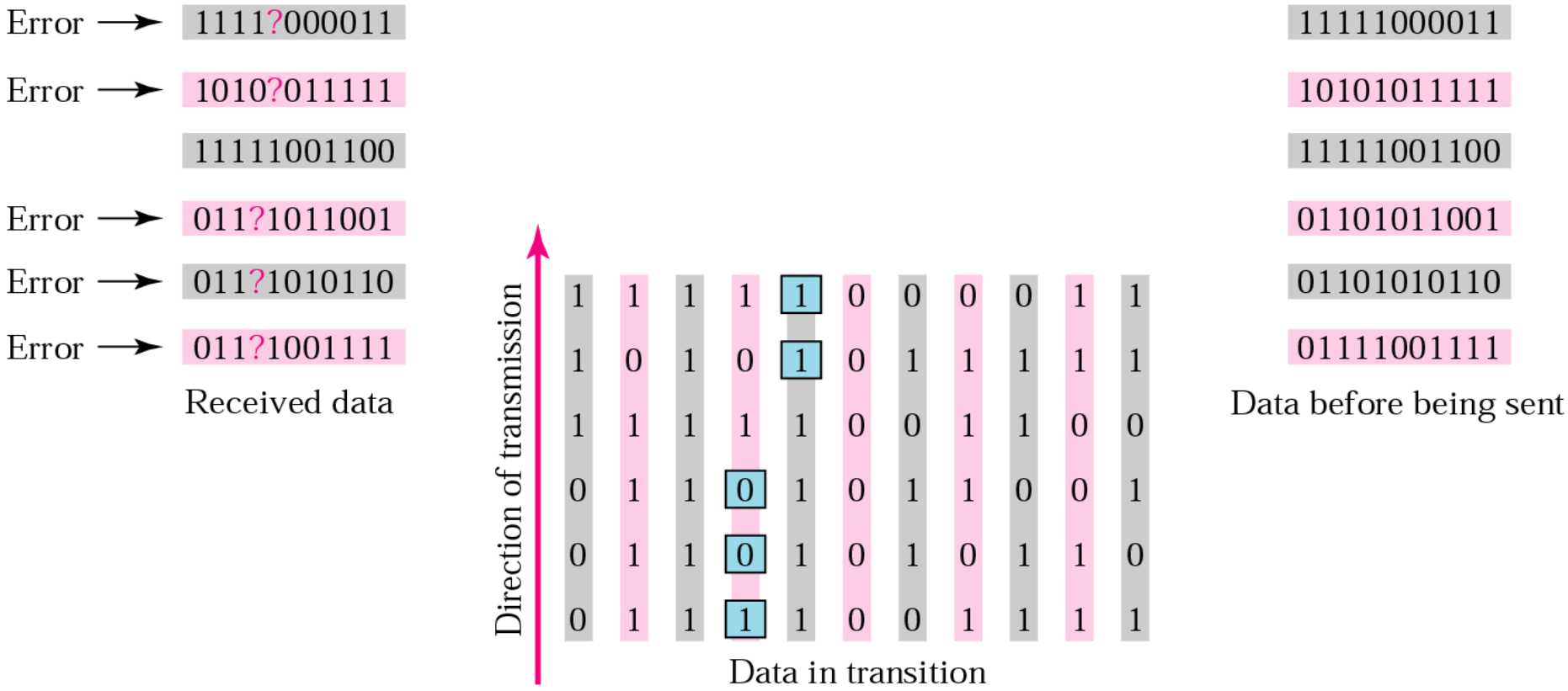
Data:
1 0 0 1 1 0 1

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

Error detection using Hamming code



Burst error correction example



Multiple-bit error correction

- **Single bit errors are not common in data communications. One response is to generalize hamming codes for double- or multiple-bit error correction.**
- **The number of extra bits becomes quite large and is used in very specialized cases.**