Error Detection and Correction

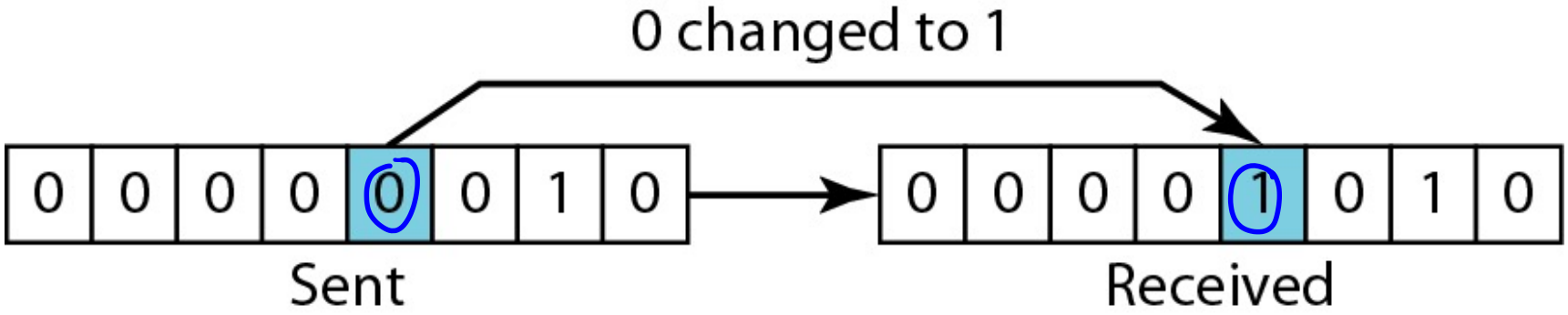
Introduction

- *Whenever bits flow from one point to another, they are subject to unpredictable changes*
- *For most application, a system must guarantee that the data received are identical to data transmitted*
- *Many factors can alter one or more bits of a message*
  - *Called errors*
- *Random errors in audio and video transmissions may be tolerable*
  - *But for text high accuracy is required*

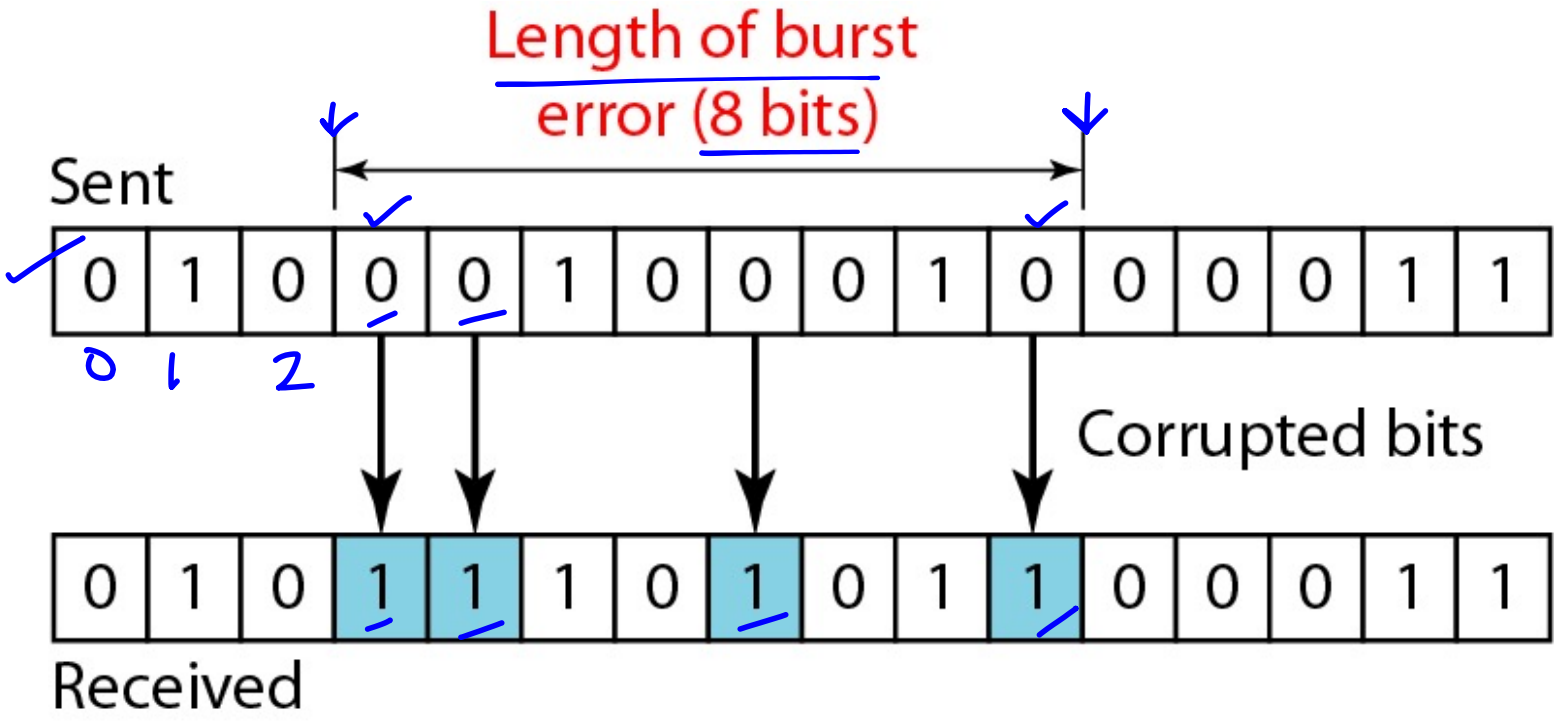
Types of errors

- In digital data we can consider two types of error
  - **Single bit error**
    - Only 1 bit of a given data unit (such as byte, character, or packet) is changed from 1 to 0 or 0 to 1
  - **Burst error:**
    - 2 or more bits in the data unit have changed from 1 to 0 or 0 to 1

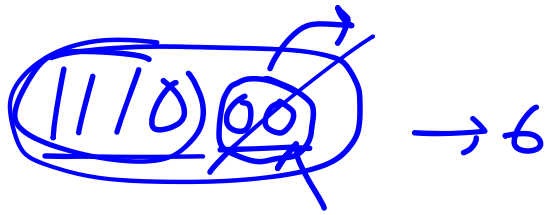
Single-bit error



Burst error of length 8



Redundancy

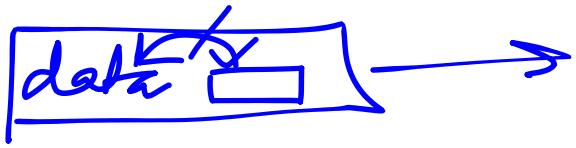


- To detect or correct errors, we need to send extra (redundant) bits with data.
- These redundant bits are added by the sender and removed by the receiver
- Their presence allows the receiver to detect or correct corrupted bits

Detection Vs Correction

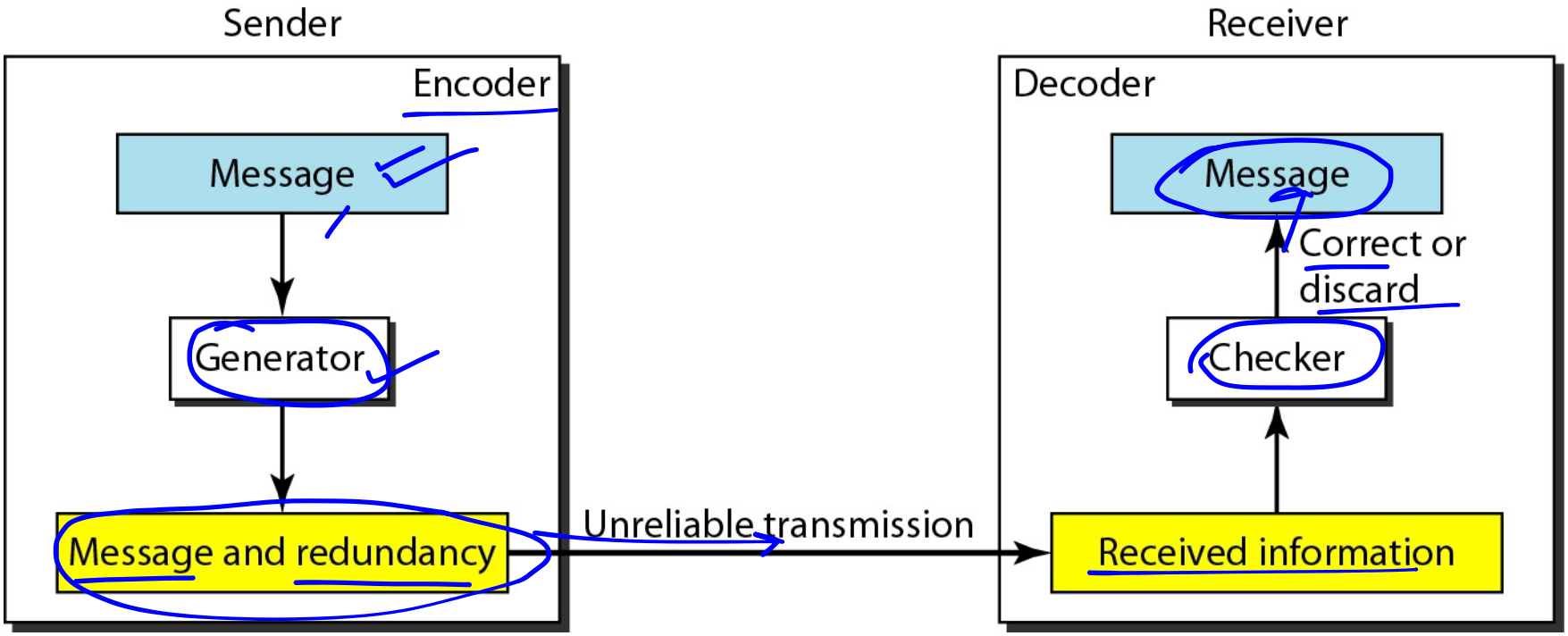
- The correction of error is more difficult than the detection
- In ***error detection***, we are only interested to see whether any error has occurred
  - Answer is yes or no
  - Single bit error is same as burst error
- In ***error correction***, we need to know
  - Exact number of bits corrupted
  - Location of those bits in data unit

Coding



- Redundancy is achieved through various coding schemes
- The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits
- The receiver check the relationship and detect errors
- The ratio of redundant bits to data bits and the robustness of the process is important factors in any coding scheme
- Two coding scheme: **Block coding** and **Convolution coding**

The structure of encoder and decoder



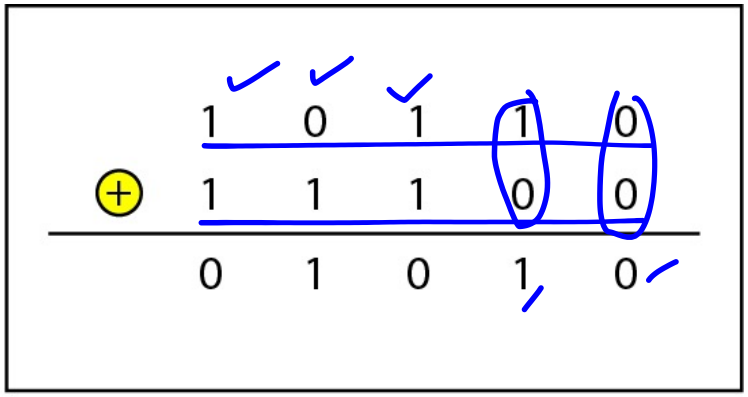
XORing of two single bits or two words

$0 \oplus 0 = 0$ $\quad$ $1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

$0 \oplus 1 = 1$ $\quad$ $1 \oplus 0 = 1$

b. Two bits are different, the result is 1.

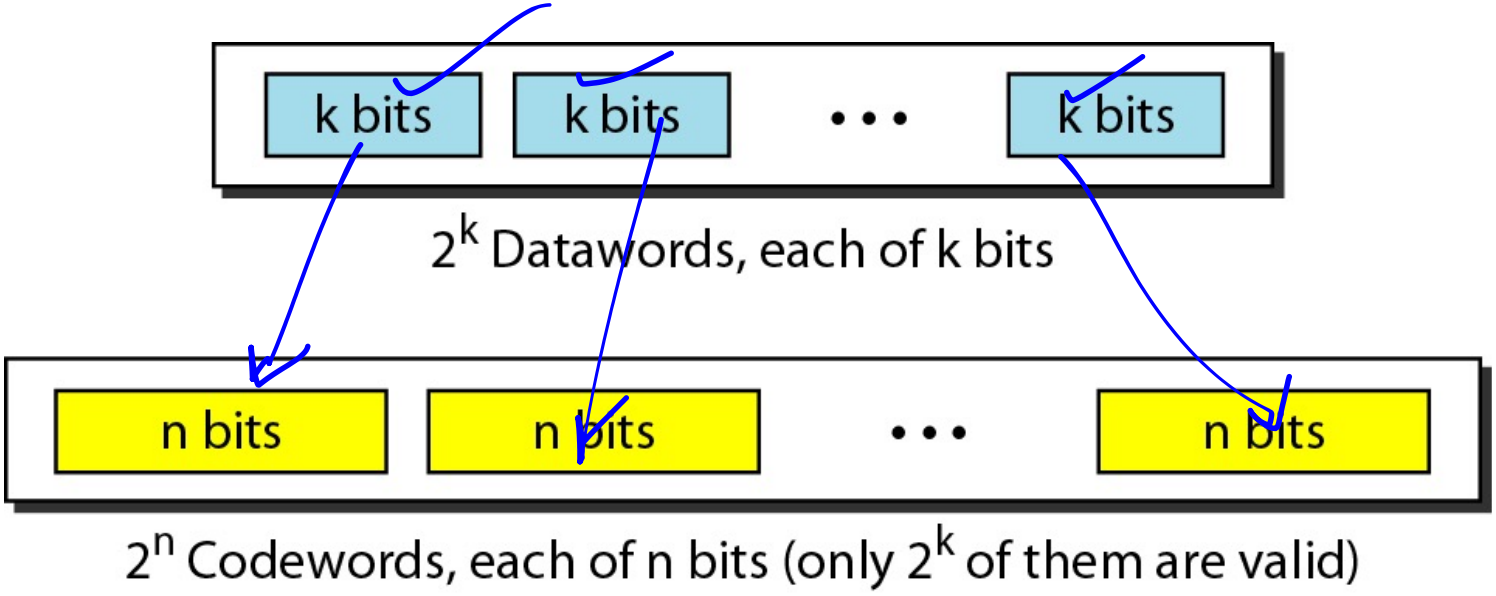


c. Result of XORing two patterns

Block Coding

- In block coding, we divide our message into blocks, each of k bits, called datawords.
- We add r redundant bits to each block to make the length n = k + r.
- The resulting n-bit blocks are called codewords.
- Since n>k, the number of possible codewords is larger than the number of possible datawords.
- The block coding process is one-to-one and hence $2^n - 2^k$ codewords are not used.
  - These are called invalid or illegal code word
- So if the receiver receives an invalid coderword, data was corrupted

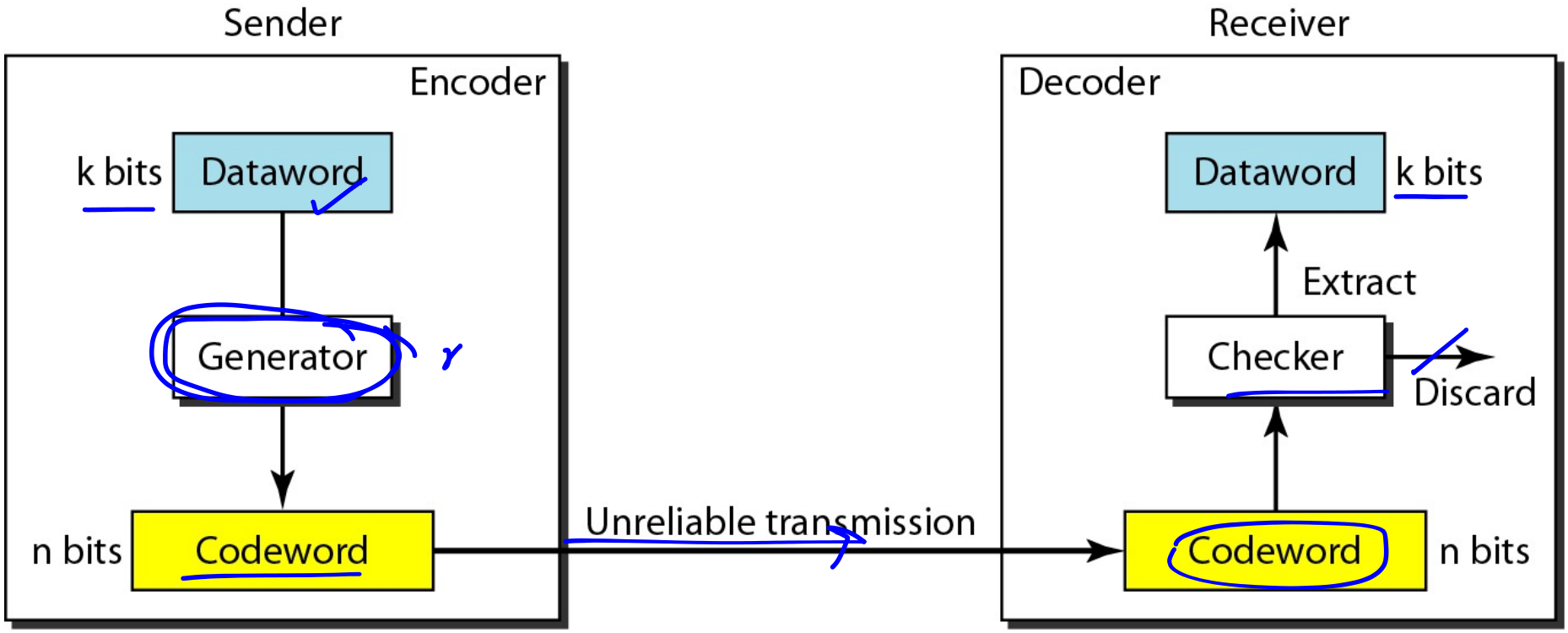
Datawords and codewords in block coding



Example

- *The 4B/5B block coding discussed is a good example of this type of coding.*
- *In this coding scheme, $k = 4$ and $n = 5$.*
- *As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords.*
- *We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.*

Process of error detection in block coding



Error Detection

- Enough redundancy is added to detect an error.
- The receiver knows an error occurred but does not know which bit(s) is(are) in error.
- Has less overhead than error correction.

Example

Let us assume that k = 2 and n = 3.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver.

Consider the following cases:

1. ***The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.***

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Table 1

Example (continued)

2. ***The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is discarded.***

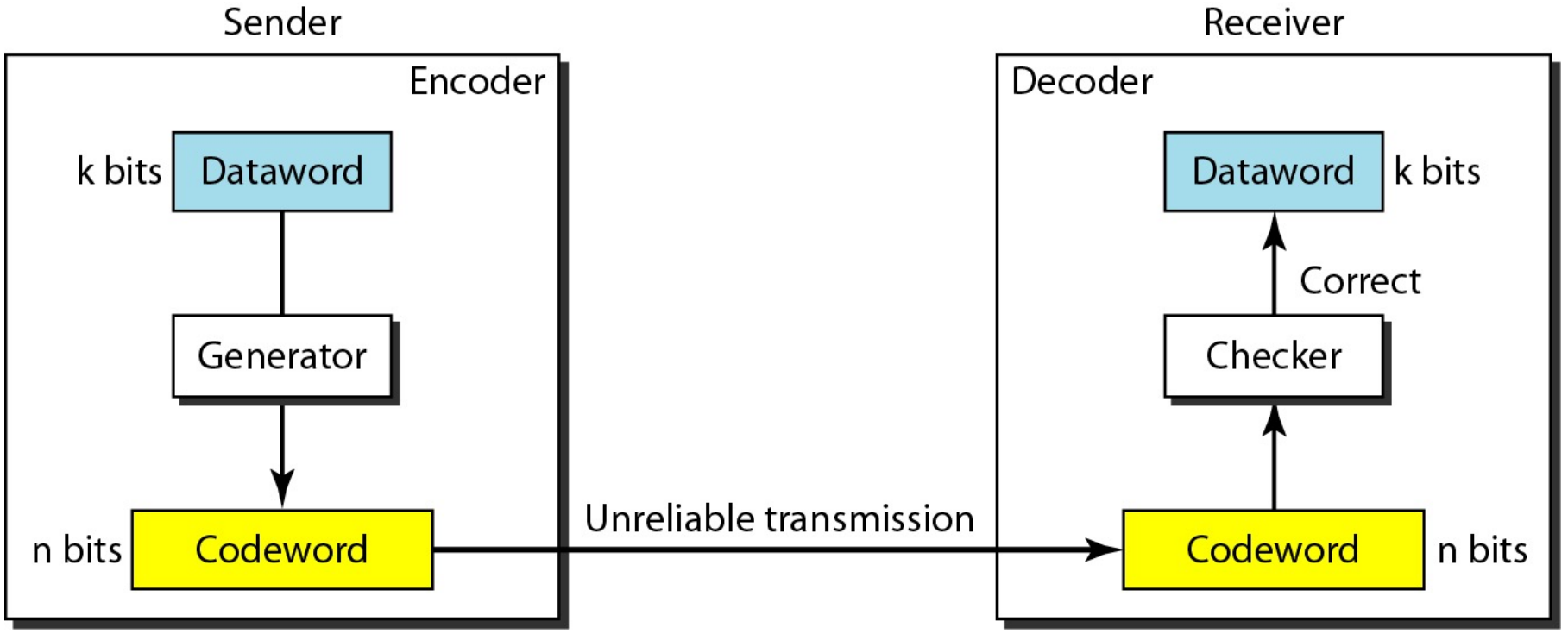
3. ***The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.***

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Note

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Structure of encoder and decoder in error correction



Example

- Let us add more redundant bits to the same Example to see if the receiver can correct an error without knowing what was actually sent.
- We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords.
- Table shows the datawords and codewords.
- Assume the dataword is 01. The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received.
  - First, the receiver finds that the received codeword is not in the table.
  - This means an error has occurred.
  - The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

Example (continued)

1. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.

2. By the same reasoning, the original codeword cannot be the third or fourth one in the table.

3. The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

A code for error correction

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

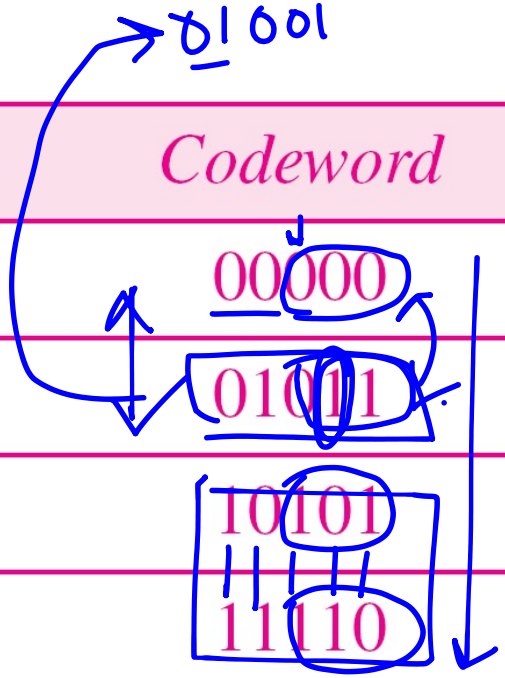


Table 2

Hamming distance

- Hamming distance between two words (of the same size) is the number of differences between the corresponding bits
- Hamming distance between two words *x* and y as *d(x,y)*
- This is important in error detection
  - This distance determines the distance between dataword and codeword as the number of bits that are corrupted during transmission

Example

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d(000, 011) is 2 because

$000 \oplus 011$ is 011 (two 1s)

2. The Hamming distance d(10101, 11110) is 3 because

$10101 \oplus 11110$ is 01011 (three 1s)

Note

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Example

Find the minimum Hamming distance of the coding scheme in Table.

Solution

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Example

Find the minimum Hamming distance of the coding scheme in Table.

Solution

We first find all the Hamming distances.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Note

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

Example

- The minimum Hamming distance for our first code scheme is 2.
- This code guarantees detection of only a single error.
- For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword.
- If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

$d(000, 011) = 2$	$d(000, 101) = 2$	$d(000, 110) = 2$	$d(011, 101) = 2$
$d(011, 110) = 2$	$d(101, 110) = 2$		

Example

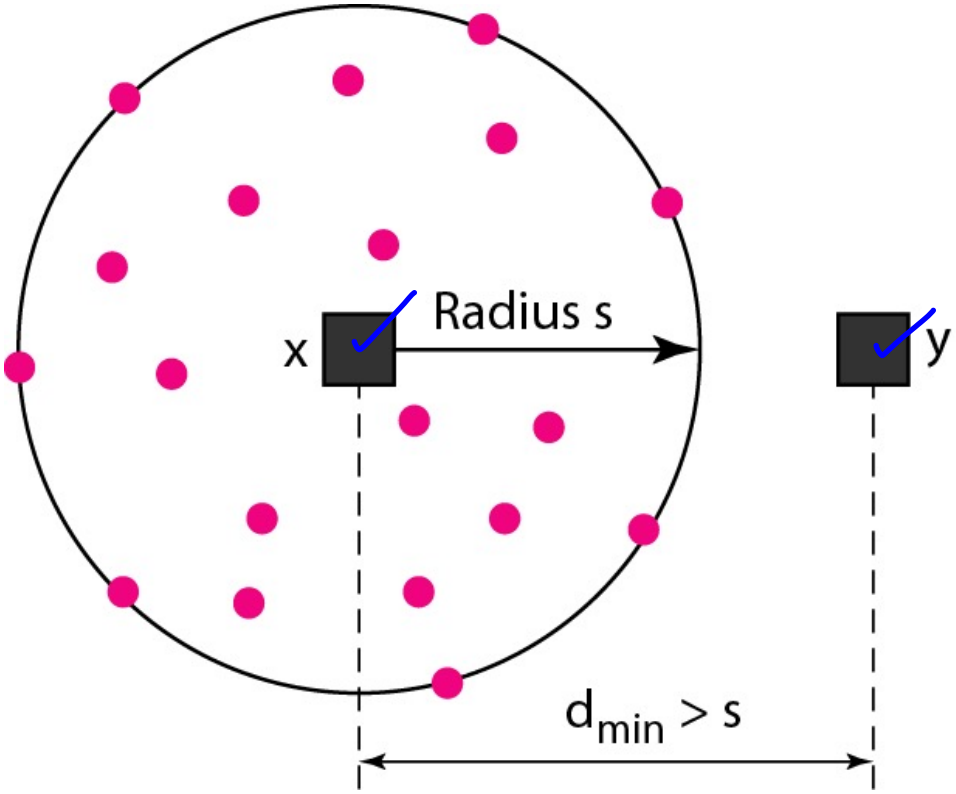
- Our second block code scheme has $d_{min} = 3$. This code can detect up to two errors.
- Again, we see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.

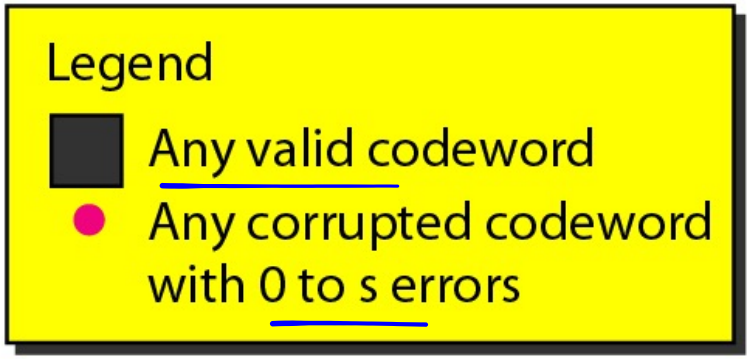
- However, some combinations of three errors change a valid codeword to another valid codeword.

- The receiver accepts the received codeword and the errors are undetected.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

Geometric concept for finding d_{min} in error detection





Linear Block Codes

- ***Almost all block codes used today belong to a subset called linear block codes.***
- ***A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.***

Note

In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

Example

Let us see if the two codes we defined in earlier Tables and Table belong to the class of linear block codes.

1. ***The scheme in Table is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword.***
   - ***For example, the XORing of the second and third codewords creates the fourth one.***

2. ***The scheme in second Table is also a linear block code. We can create all four codewords by XORing two other codewords.***

Example

- ***In our first code (Table 1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.***

- ***In our second code (Table 2), the numbers of 1s in the nonzero codewords are 3, 3, and 4. So in this code we have $d_{min} = 3$.***

Parity Check Code

- Most familiar error-detecting code
- This code is a linear block code
- In this code, a k-bit data word is changed to a n=k+1-bit codeword
- The extra bit is called the parity bit
  - Parity bit is selected to make the total number of 1s in the codeword even
- Minimum hamming distance for this category is $d_{min} = 2$
- So this is a single bit error detecting code

Note

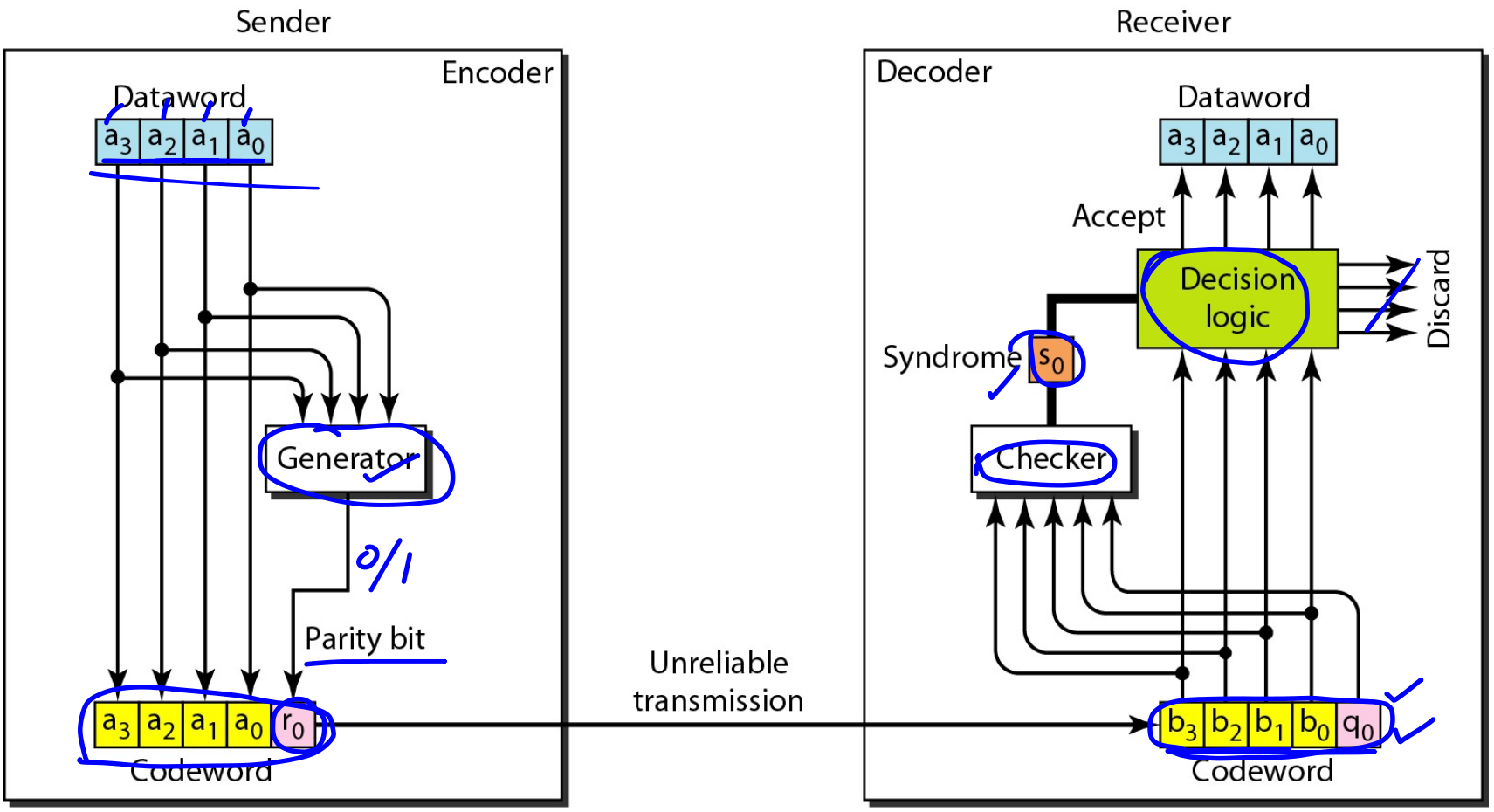
- **A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.**

- **Even parity (ensures that a codeword has an even number of 1's) and odd parity (ensures that there are an odd number of 1's in the codeword)**

Simple parity-check code C(5, 4)

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Encoder and decoder for simple parity-check code



- **Generator uses modulo 2 arithmetic: $r_0 = a_3 + a_2 + a_1 + a_0$**
- **Checker does the calculation: $s_0 = b_3 + b_2 + b_1 + b_0 + q_0$**

If syndrome is $s_0 = 0$ then no error

Example: *Some transmission scenarios*

- Assume the sender sends the dataword 1011.
- The codeword created from this dataword is 10111, which is sent to the receiver.

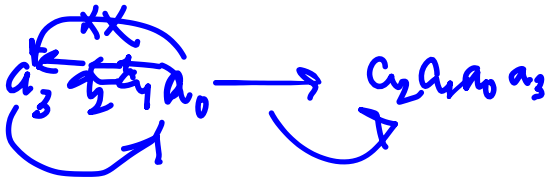
- We examine five cases:
  - No error occurs; the received codeword is 10111.
    - The syndrome is 0. The dataword 1011 is created.
  - One single-bit error changes a_1.
    - The received codeword is 10011.
    - The syndrome is 1.
    - No dataword is created.
  - One single-bit error changes r_0.
    - The received codeword is 10110. The syndrome is 1.
      - No dataword is created.

Example (continued)

- An error changes r_0 and a second error changes a_3 .
  - The received codeword is 00110. The syndrome is 0.
  - The dataword 0011 is created at the receiver.
  - Note that here the dataword is wrongly created due to the syndrome value.

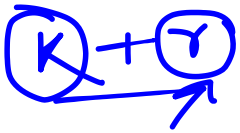
- Three bits—a_3, a_2, and a_1—are changed by errors.
  - The received codeword is 01011. The syndrome is 1.
  - The dataword is not created.
  - This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Cyclic Codes



- ***Cyclic codes** are special linear block codes with one extra property.*
- *In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.*
- *We can create cyclic codes to correct errors*
- *We will consider a subset of cyclic codes to called the **Cyclic Redundancy Check(CRC)***
  - *Used in LANs and WANs*

Cyclic Redundancy Check(CRC)



- A systematic error detecting code
  - a group of error control bits (which is the remainder of a polynomial division of a message polynomial by a generator polynomial) is appended to the end of the message block
  - with considerable burst-error detection capability
- The receiver generally has the ability to send retransmission requests back to the data source through a feedback channel.

CRC

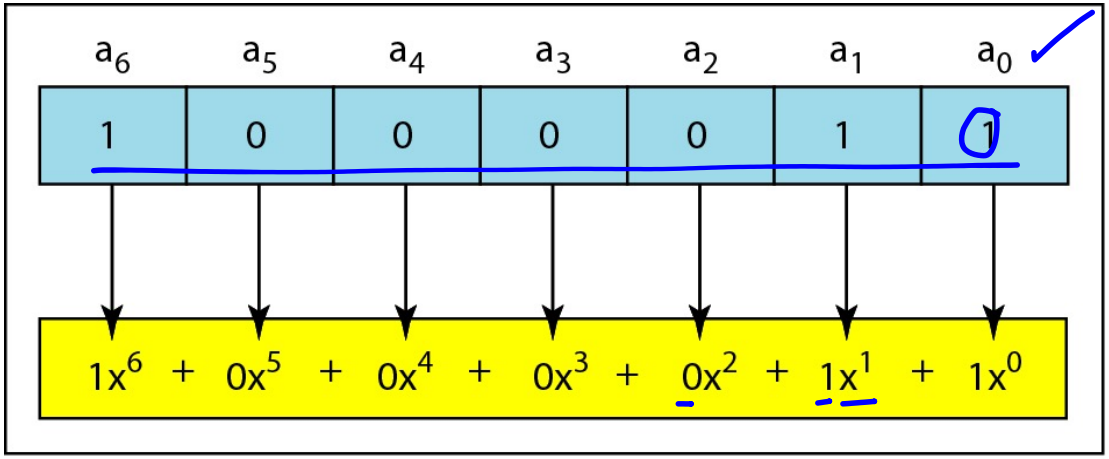
- Binary (N, k) CRC code
  - K-bit message or data bits are encoded into N code bits by appending to the message bits a sequence of n=N-k bits.
    - Message bits: $m = [m_{k-1}\ m_{k-2}\ \ldots\ m_1\ \ m_0]$
    - Appended bits: $R == [r_{n-1}\ r_{n-2}\ \ldots\ r_1\ \ r_0]$
    - CRC Code bits: $C = [c_{N-1}\ c_{N-2}\ \ldots\ c_1\ \ c_0]$

$$= [m_{k-1}\ m_{k-2}\ \ldots\ m_1\ \ m_0\ r_{n-1}\ r_{n-2}\ \ldots\ r_1\ \ r_0]$$

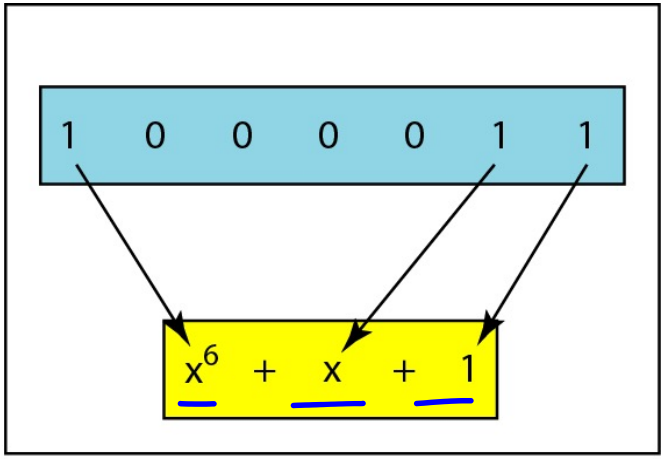
Using Polynomials

- We can use a polynomial to represent a binary word.
- Each bit from right to left is mapped onto a power term.
- The rightmost bit represents the “0” power term. The bit next to it the “1” power term, etc.
- If the bit is of value zero, the power term is deleted from the expression.

A polynomial to represent a binary word



a. Binary pattern and polynomial



b. Short form

Polynomial Representation

- Message bits: $m = [m_{k-1}\ m_{k-2}\ \ldots\ m_1\ \ m_0]$

$$m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \ldots\ m_1x^1 + \ m_0x^0$$

- Appended bits: $R == [r_{n-1}\ r_{n-2}\ \ldots\ r_1\ \ r_0]$

$$R(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \ldots\ r_1x^1 + \ r_0x^0$$

- CRC Code bits: $C = [c_{N-1}\ c_{N-2}\ \ldots\ c_1\ \ c_0]$

$$C(x) = c_{N-1}x^{N-1} + c_{k-2}x^{N-2} + \ldots\ c_1x^1 + \ c_0x^0$$

$$= x^n\, m(x) + R(x)$$

CRC polynomial

- **Example:** CRC code (k=10, N=13, n=N-k=3)
- m=[1010100101], R=[111]

$m(x) = x^9 + x^7 + x^5 + x^2 + 1$ $\quad R(x) = x^2 + x + 1$

- C=[1010100101111]

$= x^{12} + x^{10} + x^8 + x^5 + x^3 + x^2 + x + 1$

$= x^3(x^9 + x^7 + x^5 + x^2 + 1) + R(x)$

$= x^3 m(x) + R(x)$

CRC

- How to obtain the polynomial R(X) (the appended bits)
  - CRC codes are designated by a generator polynomial g(X) with degree of n

    $g=[g_n\ g_{n-1}\ g_{n-2}\ \cdots\ g_1\ \ g_0]$
  - Divide $x^n\ m(x) + R(x)$ by g(X) (modulo-2 division) and obtain the remainder, which is R(x)

    $$x^n\ m(x) = p(x)g(x) + R(x)$$
    - The remainder R(x) is always a polynomial of maximum order (n-1).

CRC

- **Example:** the polynomial R(x) (the appended bits)
- Message [11100110] 8 bits
- Given N-k=n=4,

generator polynomial g(x)= x^4+x^3+1=[11001]

$$\frac{x^n m(x)}{g(x)} = \frac{x^{11}+x^{10}+x^9+x^6+x^5}{x^4+x^3+1}$$

$$= \underbrace{x^7 + x^5 + x^4 + x^2 + x}_{P(x)} + \frac{x^2 + x}{x^4 + x^4 + 1}$$

$x^n m(x)$ is the polynomial corresponding to the message bit sequence to which a *n*-number of 0's is appended. [111001100000]

- The remainder is $R(x) = x^2 + 1$
- Therefore appended bits are: [0110] (since n=4)
- So the CRC code bits are: [11100110 0110]

Error detection

- **The polynomial for the received code word T(X) is divided by the generator polynomial g(X)**
  - ✓ ***Upon the reception without error***
    - T(X)=C(X)=X^nm(X)+R(X)
    - The remainder of T(X)/g(X) =R(X)+R(X)=***the all-zero row*** (modulo-2 addition).
    - Example:

$$\mathbf{g}(X) = X^4 + X^3 + 1$$

The transmitted CRC code bits are $[111001100110]$

$$\mathbf{T(X)} = \mathbf{C}(X) = X^{11} + X^{10} + X^9 + X^6 + X^5 + X^2 + X$$

$$= (X^7 + X^5 + X^4 + X^2 + X)\mathbf{g}(X)$$

The remainder of $[\mathbf{C}(X)/\mathbf{g}(X)] = \mathbf{0} \longrightarrow [0000]$

Error detection

- **The polynomial for the received code word *T(X)* is divided by the generator polynomial *g(X)***

✓ ***The remainder is not zero***

  - An indication that an error has occurred in transmission and the received codeword is not a valid code word.
  - Example:

$$\mathbf{g}(X) = X^4 + X^3 + 1$$

The transmitted CRC code bits are $[111001100110]$

The received CRC code bits are $[110011100110]$

$$\mathbf{T}(X) = X^{11} + X^{10} + X^7 + X^6 + X^5 + X^2 + X = \mathbf{C}(X) + X^9 + X^7$$

$$\frac{\mathbf{T}(X)}{\mathbf{g}(X)} = \frac{\mathbf{C}(X) + X^9 + X^7}{X^4 + X^3 + 1} = (X^7 + X^2) + \frac{X}{X^4 + X^3 + 1}$$

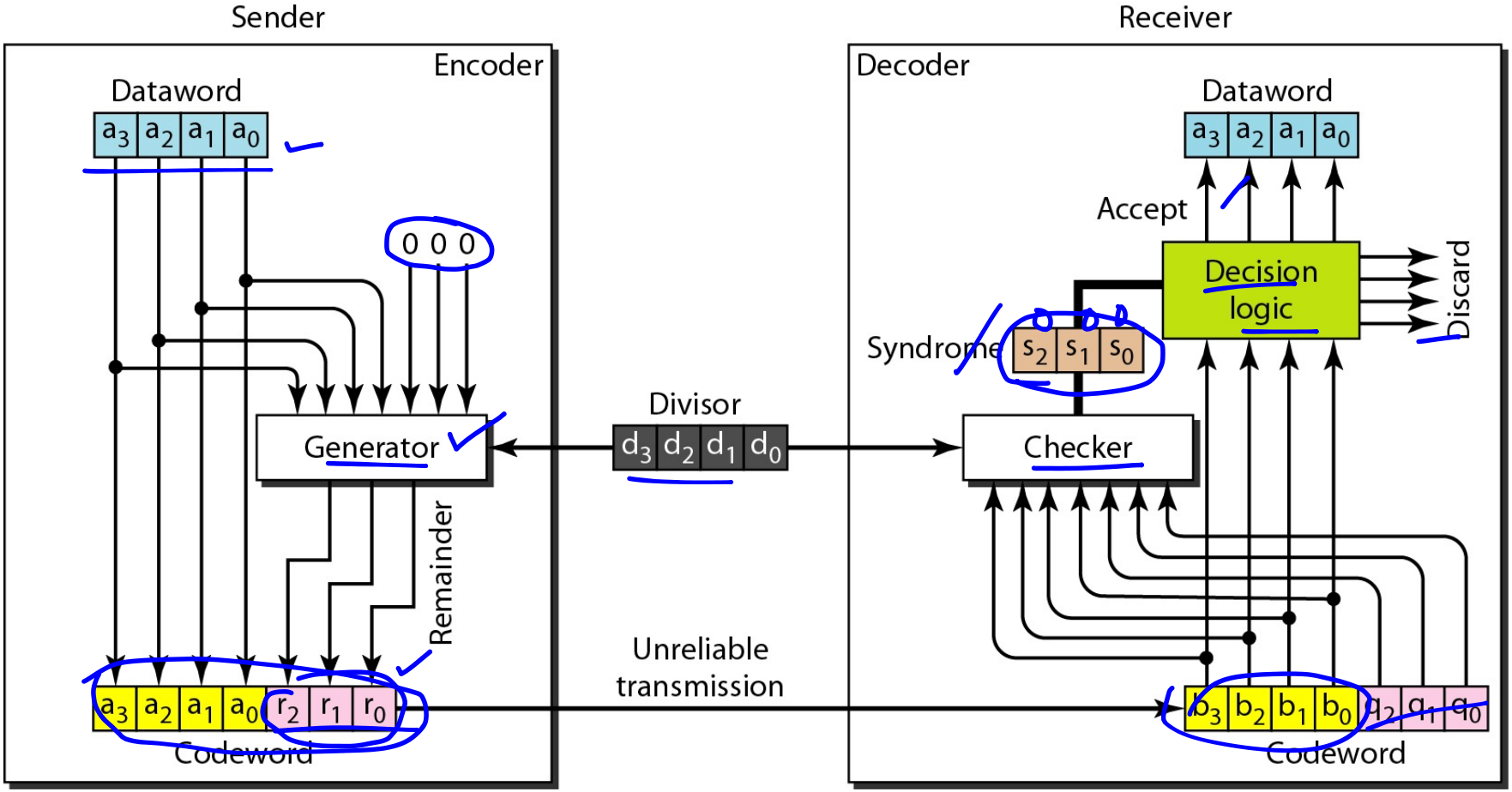
The remainder of $[\mathbf{T}(X)/\mathbf{g}(X)] = X \longrightarrow [0010]$

A CRC code with C(7, 4)

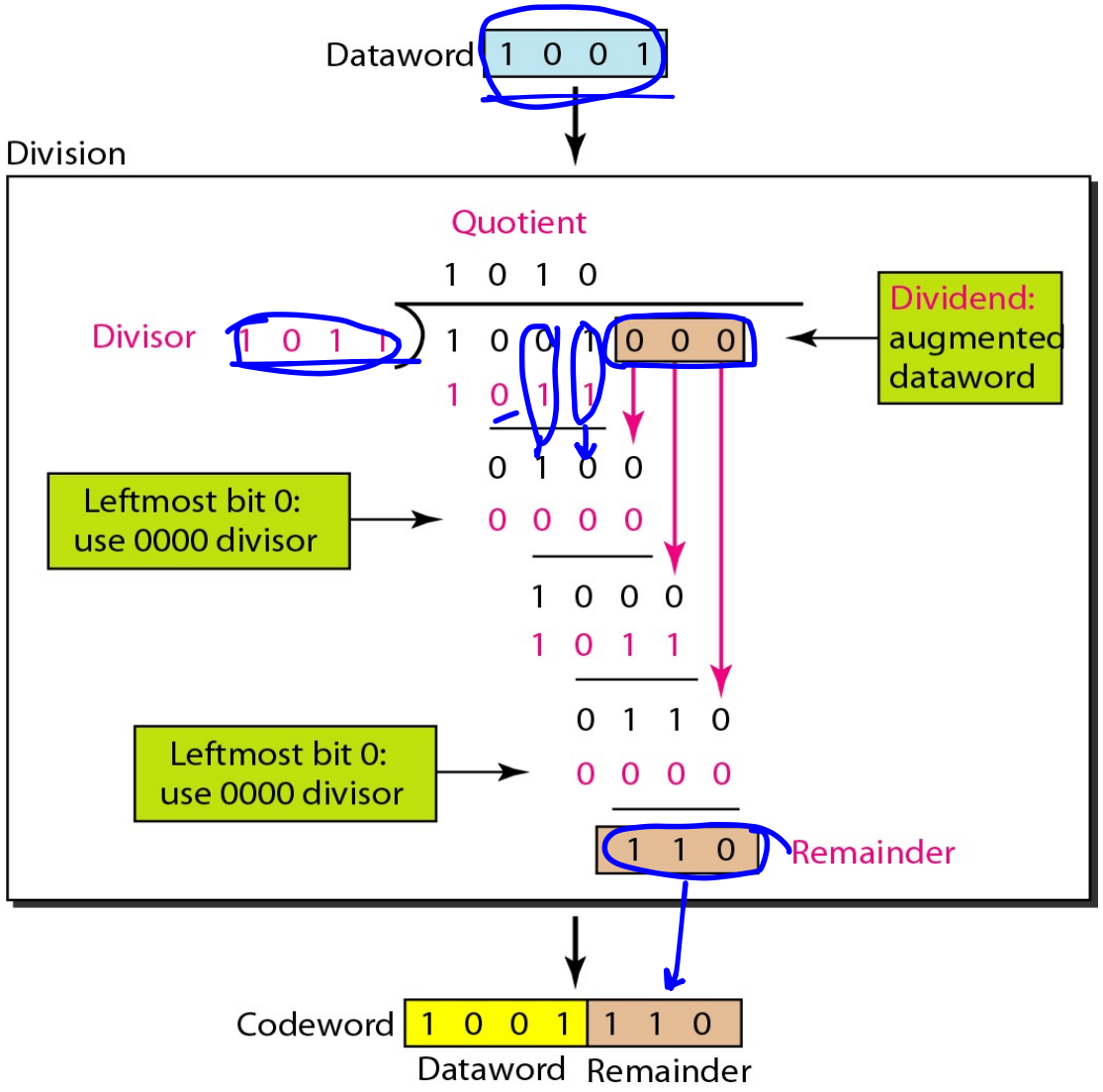
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Generator polynomial: g(x)= x^3+x+1 → [1011]

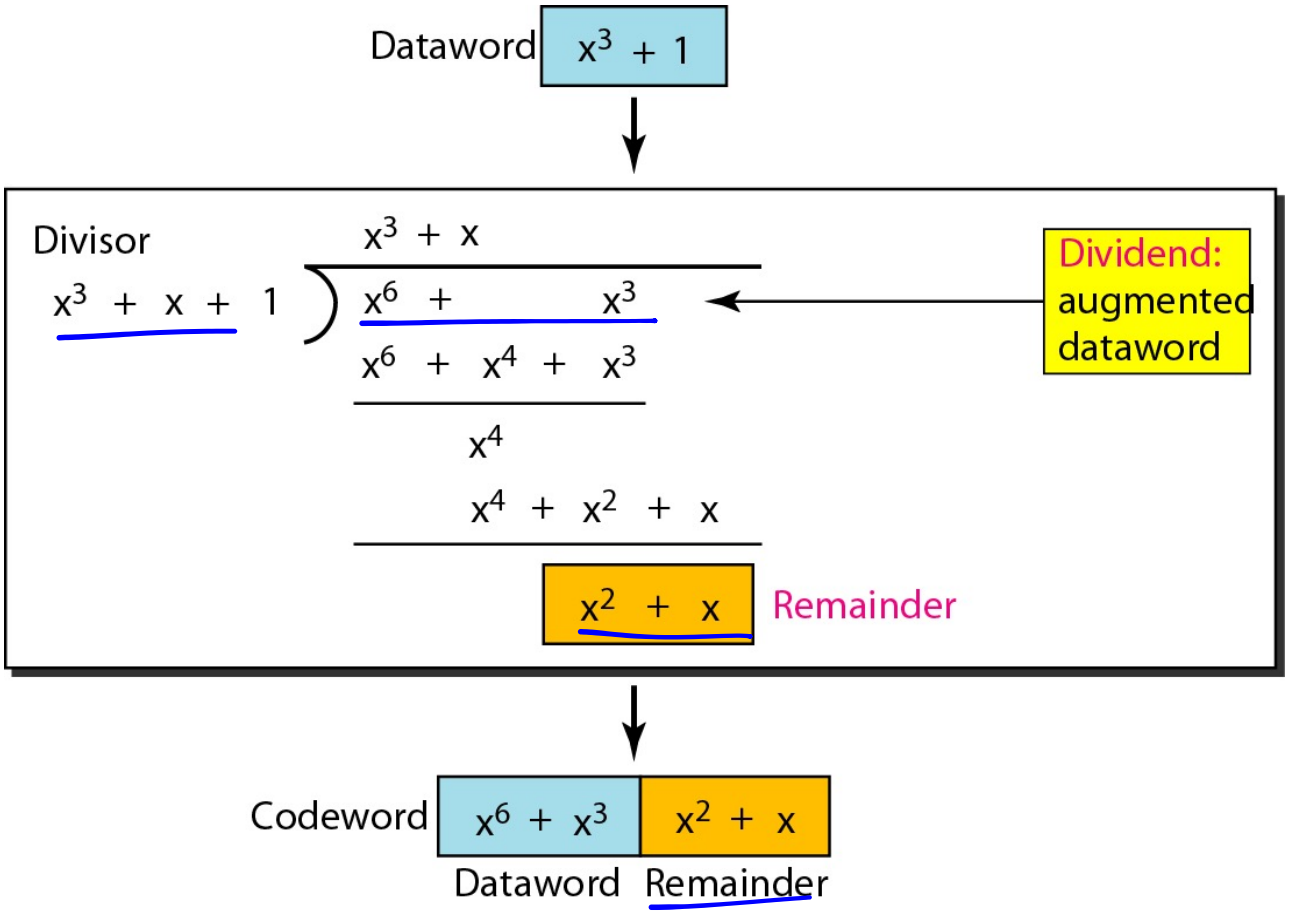
CRC encoder and decoder



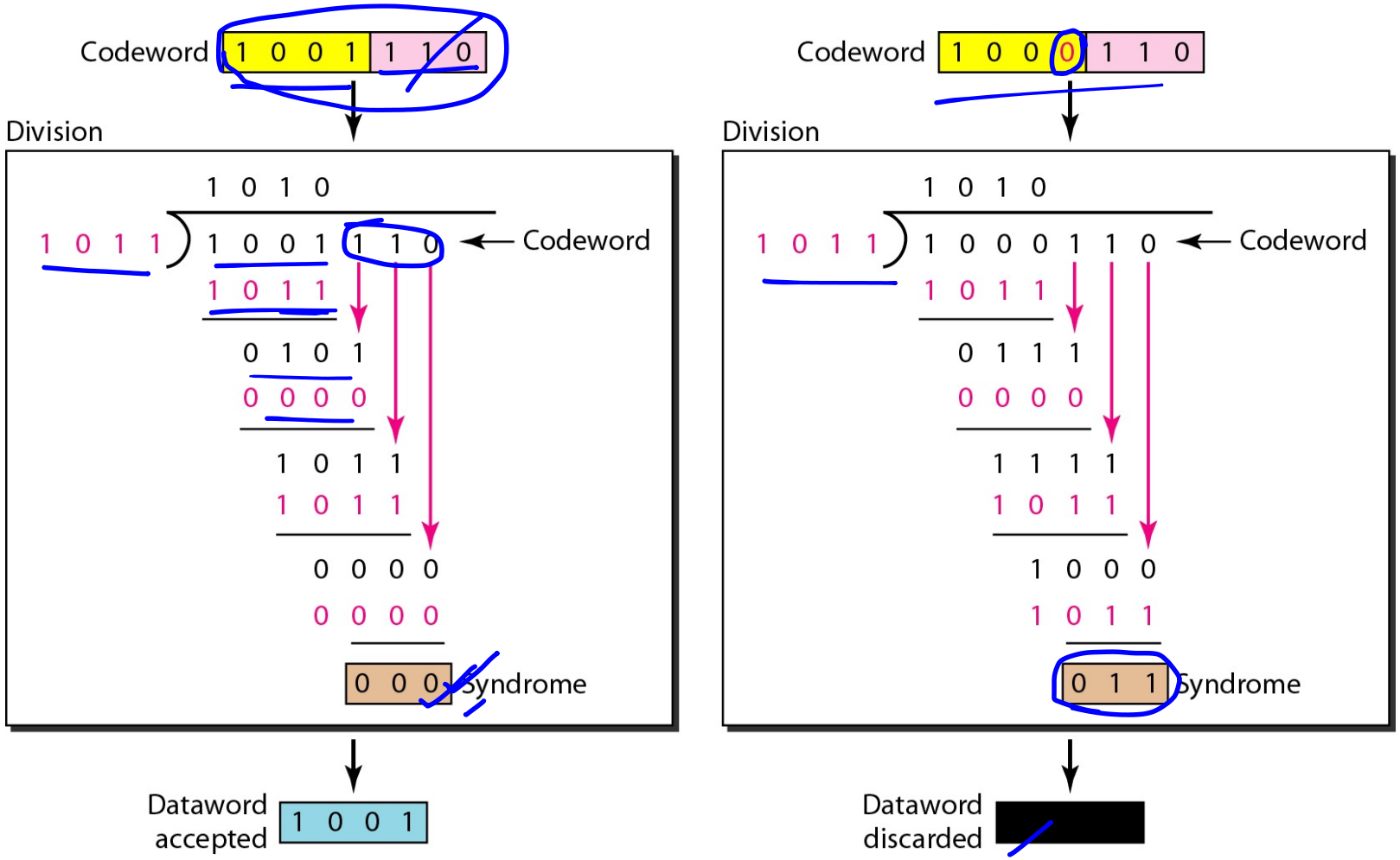
Division in CRC *encoder*



CRC division using polynomials



Division in the CRC decoder for two cases



Cyclic Code Analysis

- We can analyze a cyclic code to find its capabilities by using polynomials
- We define the following polynomials with binary coefficients
  - Dataword: d(x)
  - Codeword: c(x)
  - Generator: g(x)
  - Syndrome: s(x)
  - Error: e(x)

Error detection

- In a cyclic code
  - if $s(x) \neq 0$ then one or more bits is corrupted
  - If $s(x)=0$ either
    - No bit is corrupted or
    - Some bits are corrupted, but the decoder failed to detect them
- To analyze we want to find the criteria that must be imposed on the generator, $g(x)$ to detect type of error we want to detect

Relationships

- Let us try to find the relations among the functions involved:
  - Received codeword: c(x) + e(x)
  - Receiver divides the received codeword by g(x) to get syndrome

So, $\frac{Received\ \ codeword}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$

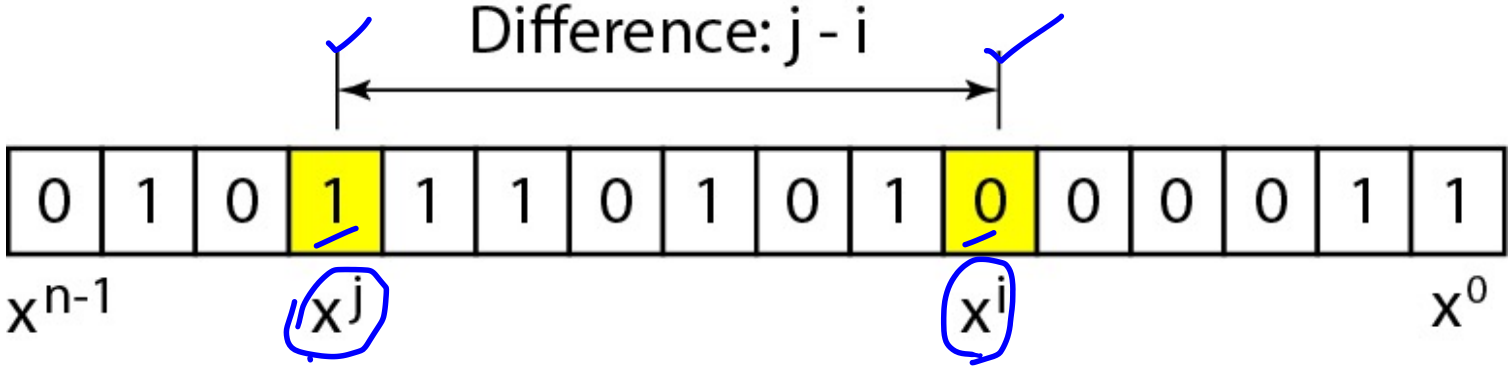
x^i → s(x)

  - First term has a remainder of zero
  - Syndrome is obtained from the second term
  - If this term does not have a *remainder* (s(x)=0), either e(x)=0 or e(x) is divisible by g(x)
  - Those errors that are divisible by g(x) are not caught

Analysis: Single-bit-error

- Single bit error is $e(x)=x^i$, i is the position of the bit.
- If a single bit error is caught, then x^i is not divisible by $g(x)$.
  - *This is possible when g(x) has at least two terms and the coefficient of x^0 is not zero*

Isolated single-bit errors



Analysis: Isolated Single-bit-error

- This type of error can be written as $e(x) = x^j + x^i$
  - The value of *i and j* define the position of the errors and the difference *j-i* defines the length of the error
  - Now $e(x) = x^j + x^i = x^i(x^{j-i}+1)$
  - If *g(x)* has more than one term and one term is x^0, it cannot divide x^i.
  - So if g(x) is to divide e(x), it must divide $(x^{j-1}+1)$
  - In other words g(x) must not divide (x^t+1), where t is between *0 and (n-1)*
    - But t=0 is meaningless and *t=1* is needed.
    - So t is between ***2 to (n-1)***

Analysis: Odd numbers of error

- A generator with a factor of (x+1) can detect *all odd numbers of errors*
  - It is not that g(x) = x+1 itself.
  - If it is only (x+1), it cannot catch the two adjacent isolated errors
- **Example:** $g(x)=x^4+x^2+x+1$ can catch all odd number of errors

Burst Errors

- A burst error is in the form

$$e(x) = x^j + \ldots\ldots + x^i = x^i(x^{j-i} + \ldots + 1)$$

- *If our generator cannot divide x^i it can detect a single bit error.*
- *Now, we should worry about the generators that divide $(x^{j-i}+\ldots+1)$*
  - *The remainder $(x^{j-i}+\ldots+1)/(x^r+\ldots+1)$ must not be zero.*
    - *Here, $(x^r+\ldots+1)$ is the generator polynomial.*

Note

- All burst errors with $L \leq r$ will be detected.
- All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$
- All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^{r}$.

Note

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

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

Checksum

- *It is an error-detecting technique*
- *The checksum is used in the Internet by several protocols although not at the data link layer.*

Example

- Suppose our data is a list of five 4-bit numbers that we want to send to a destination.
- In addition to sending these numbers, we send the sum of the numbers.
  - **For example:**, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers.
- The receiver adds the five numbers and compares the result with the sum.
- If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum.
- Otherwise, there is an error somewhere and the data are not accepted.

Example

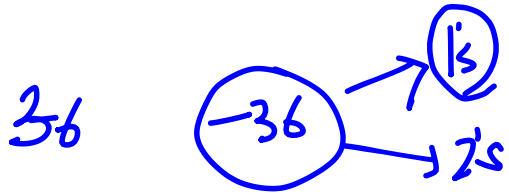
- We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum.

- In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum).

- If the result is 0, it assumes no error; otherwise, there is an error.

Example

How can we represent the number 21 in one's complement arithmetic using only four bits?



Solution

- ***The number 21 in binary is 10101 (it needs five bits).***
- ***We can wrap the leftmost bit and add it to the four rightmost bits.***
- ***We have (0101 + 1) = 0110 or 6.***

Example

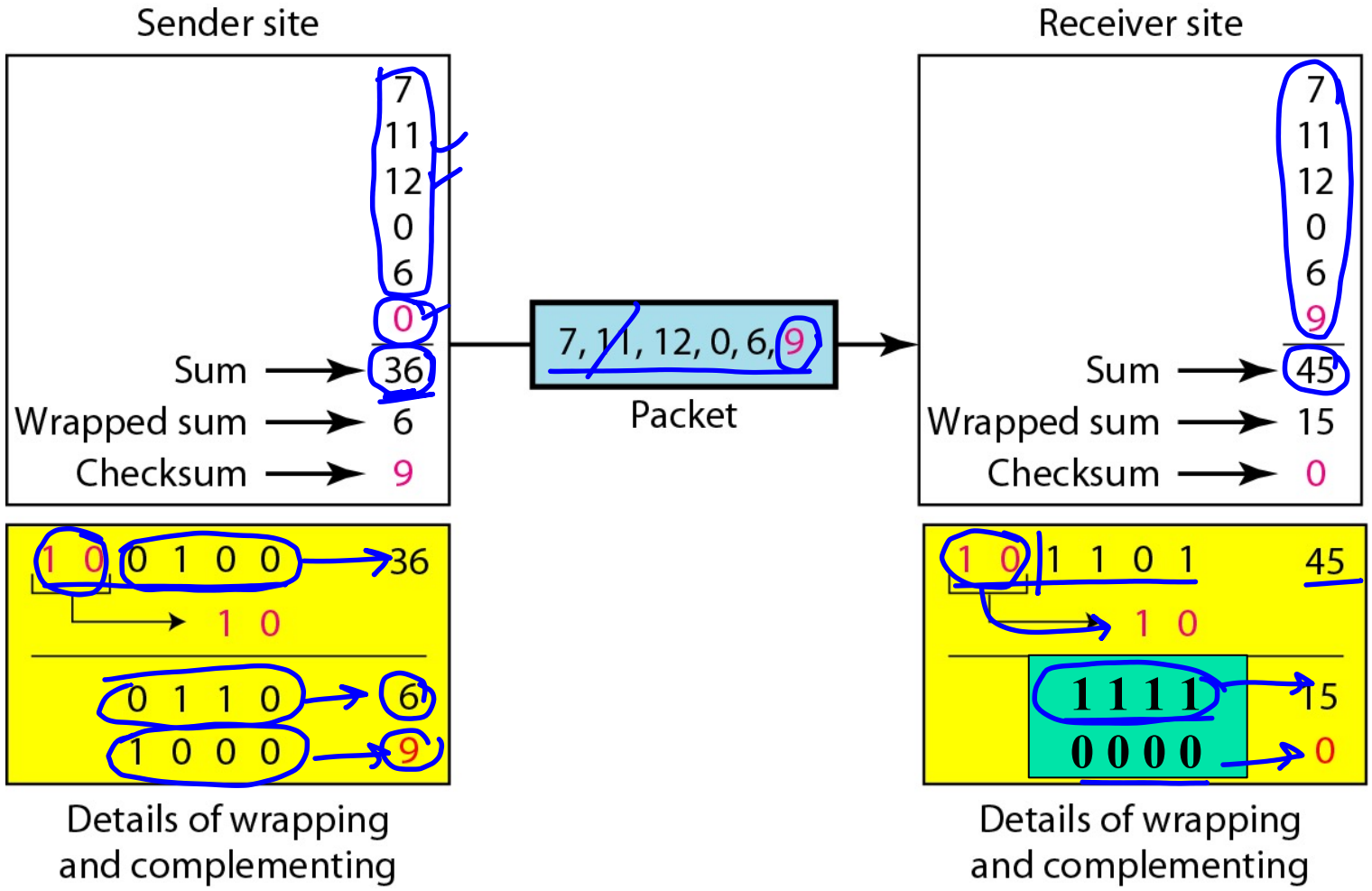
How can we represent the number −6 in one's complement arithmetic using only four bits?

Solution

- In one's complement arithmetic, the negative or complement of a number is found by *inverting all bits*.
- Positive 6 is 0110; negative 6 is 1001.
- If we consider only unsigned numbers, this is 9. In other words, the complement of 6 is 9.
- Another way to find the complement of a number in one's complement arithmetic is to subtract the number from $2^n - 1$ (16 − 1=15 in this case).

Example

Uses 4 bits checksum



Example

- The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color).

- The result is 36.
- However, 36 cannot be expressed in 4 bits.
  - *The extra two bits are wrapped and added with the sum to create the wrapped sum value 6.*

- The sum is then complemented,
  - Resulting in the checksum value 9 (15 − 6 = 9).

- The sender now sends six data items to the receiver including the checksum 9.

Example (continued)

- The receiver follows the same procedure as the sender.
- It adds all data items (including the checksum); the result is 45.
- The sum is wrapped and becomes 15.
- The wrapped sum is complemented and becomes 0.
- Since the value of the checksum is 0, this means that the data is not corrupted.
- The receiver drops the checksum and keeps the other data items.
- If the checksum is not zero, the entire packet is dropped.

Note

Traditionally the Internet has used a 16-bit checksum

Sender site:

1. The message is divided into 16-bit words.

2. The value of the checksum word is set to 0.

3. All words including the checksum are added using one's complement addition.

4. The sum is complemented and becomes the checksum.

5. The checksum is sent with the data.

Note

Receiver site:

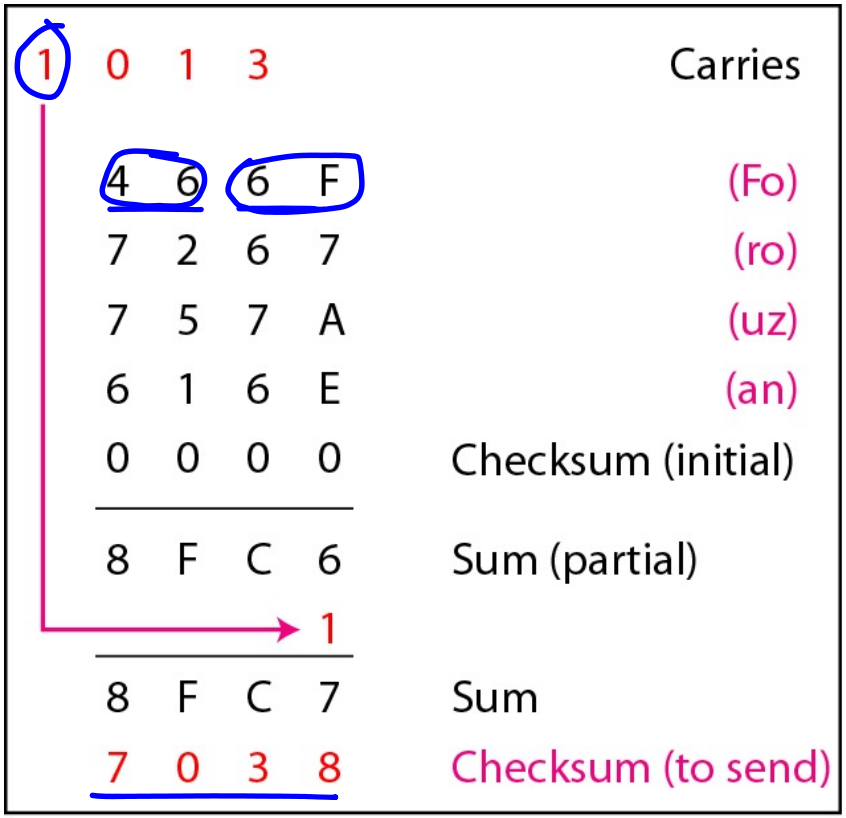
1. The message (including checksum) is divided into 16-bit words.

2. All words are added using one's complement addition.

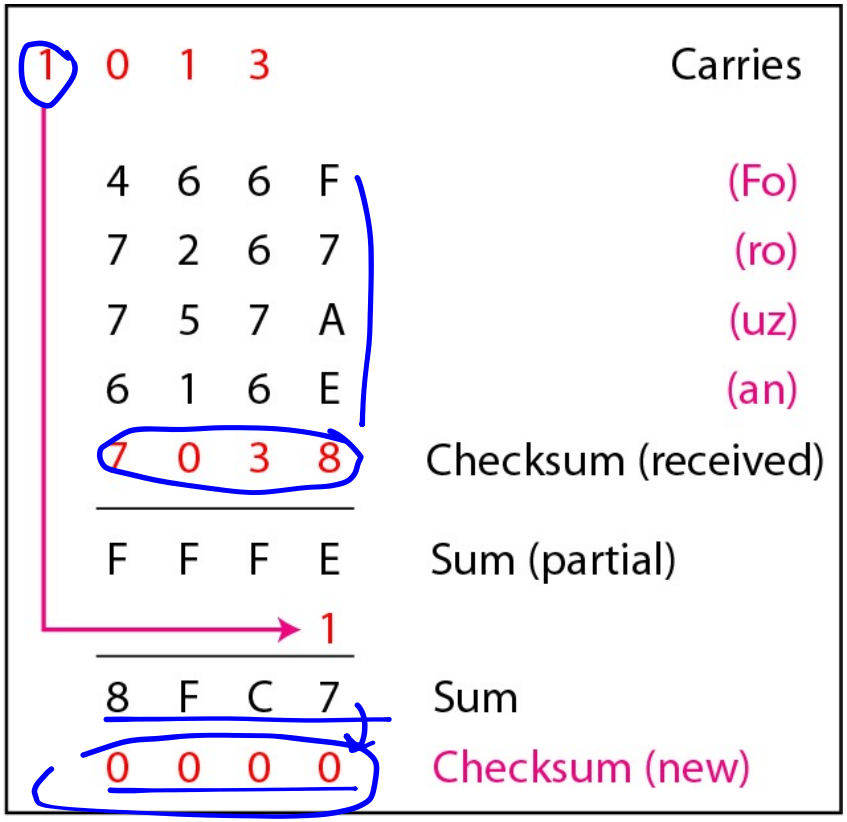
3. The sum is complemented and becomes the new checksum.

4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.

Example



a. Checksum at the sender site



a. Checksum at the receiver site