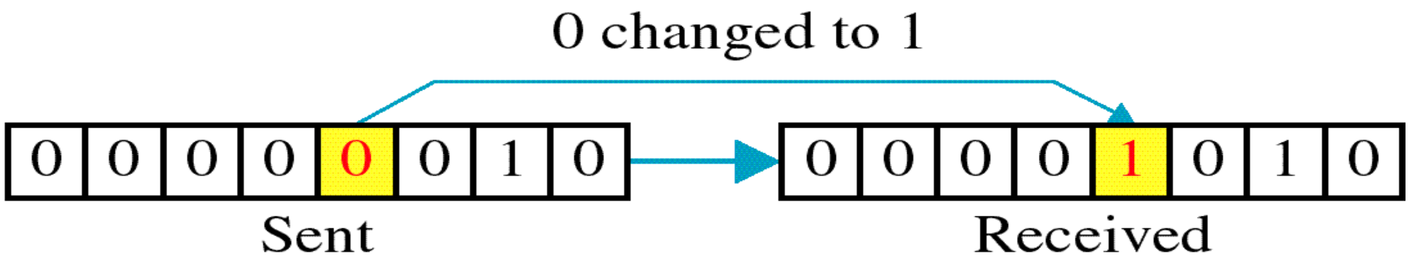
Error Detection & Correction

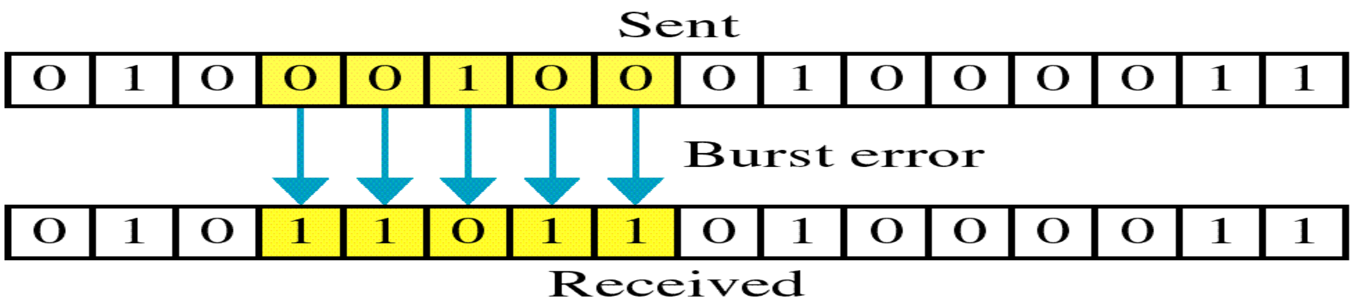
- Data can be corrupted during transmission.
- Some applications require that errors be detected and corrected.

Types of Errors



Single-bit error

In a single-bit error, only 1 bit in the data unit has changed.



Burst error

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

- Redundancy
  - To detect or correct errors, we need to send extra (redundant) bits with data.

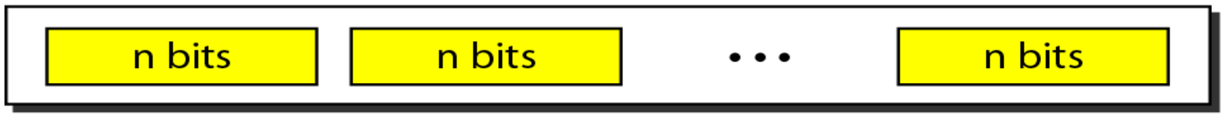
- Error detection vs. Correction:
  - In error detection, we are looking to see if any error has occurred. Simple YES or NO.
  - In error correction, we need to the exact number of bits that are corrupted and their locations in the message

- Forward Error Correction vs. Retransmission
  - Forward Error Correction is the process in which the receiver tries to guess the message by using the redundant bits.
  - In Retransmission, the receiver asks the sender to resend the message.

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.

2^k Datawords, each of k bits

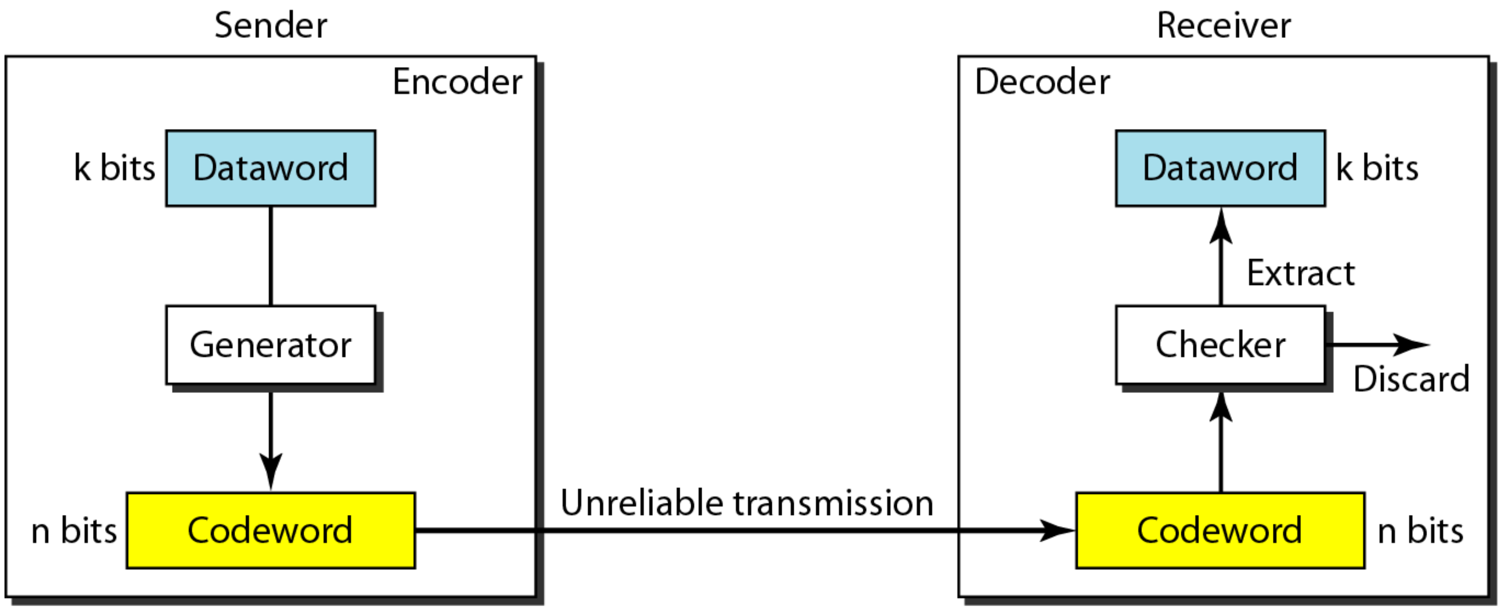


2^n Codewords, each of n bits (only 2^k of them are valid)

- With k bits, we can create a combination of 2^k datawords and with n bits we can create 2^n codewords.
- Since n > k, we have $2^n - 2^k$ codewords that are not used, which we call as invalid codewords or illegal.

Block Coding

- Error Detection:
  - An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.
  - The receiver can detect a change in the original codeword if:
    - The receiver has a list of valid codewords
    - The original codeword has changed to an invalid one.



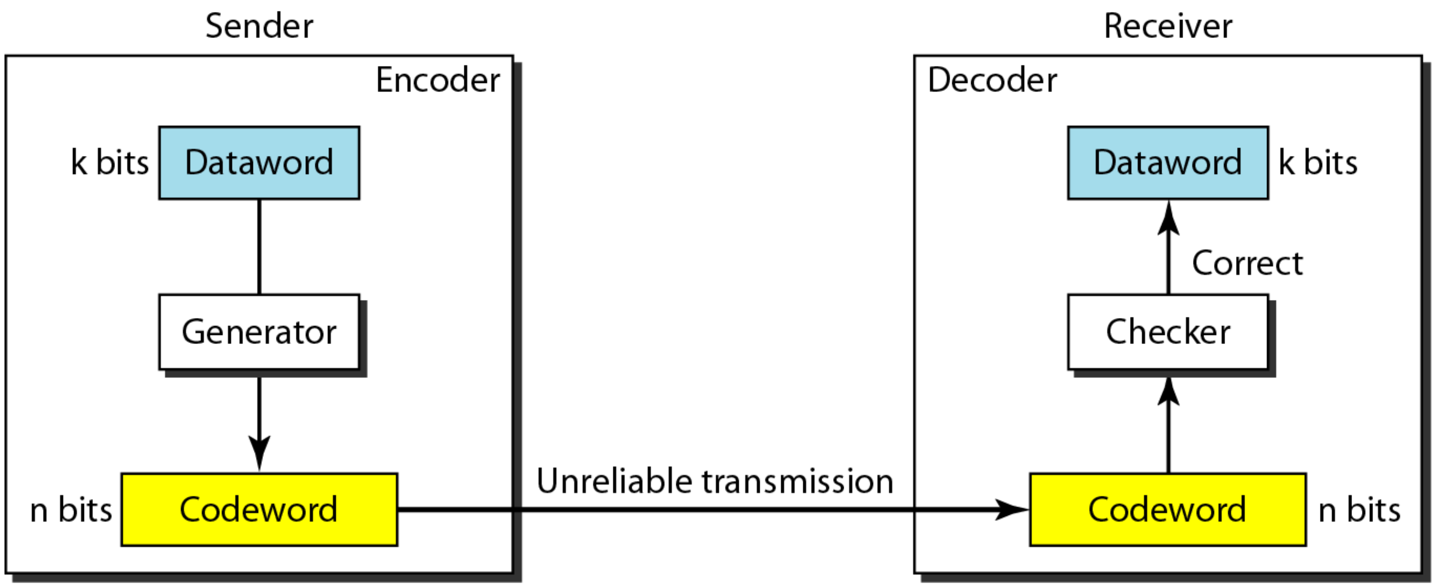
Error Detection

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

- Let us assume that k = 2 & n = 3. Table shows the datawords and codewords.
- Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:
  - The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
  - The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is discarded.
  - 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.

Block Coding

- Error Correction:
  - More complex then error detection
  - The receiver needs to find (or guess) the original codeword sent.
  - We need more redundant bits for error correction



Error Correction

- 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.

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

- The codeword is corrupted during transmission, and 01001 is received.
- First, the received codeword is not in the table. So, an error has occurred. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.
  - Comparing with the 1st codeword in the table (01001 vs. 00000), the receiver decides that its not the one that was sent as there are 2 different bits.
  - Similarly, the original codeword cannot be the third or fourth one in the table.
  - 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.

Hamming Distance

- The Hamming distance between two words is the number of differences between corresponding bits.
- The Hamming distance can be found by applying XOR operation on two words and count the number of 1s in the result.
- Let us find the Hamming distance between two pairs of words.
  - The Hamming distance d(000, 011) is 2 because

$$\boxed{000 \oplus 011 \text{ is } 011 \text{ (two 1s)}}$$

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

$$\boxed{10101 \oplus 11110 \text{ is } 01011 \text{ (three 1s)}}$$

Hamming Distance

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

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

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

- We first find all Hamming distances.

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

$d(011, 110) = 2$ $\quad d(101, 110) = 2$

- The d_{min} in this case is 2.

Hamming Distance

- Find the minimum Hamming distance of the coding scheme in the following table.

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

- We first find all the Hamming distances.

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

$d(01011, 10101) = 4 \quad d(01011, 11110) = 3 \quad d(10101, 11110) = 3$

- The d_{min} in this case is 3.

Hamming Distance and Error

- Notice that the Hamming distance between the sent and received codewords is the number of bits affected by the error.

- E.g. if the codeword sent is 00000 and received is 01101,
  - then 3 bits are in error and the Hamming distance between *d(00000, 01101) = 3*

Hamming Distance for Error Detection

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

- Consider the first example with minimum hamming distance $d_{min} = 2$.
  - This code guarantees detection of only a single error.
  - E.g., 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.

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

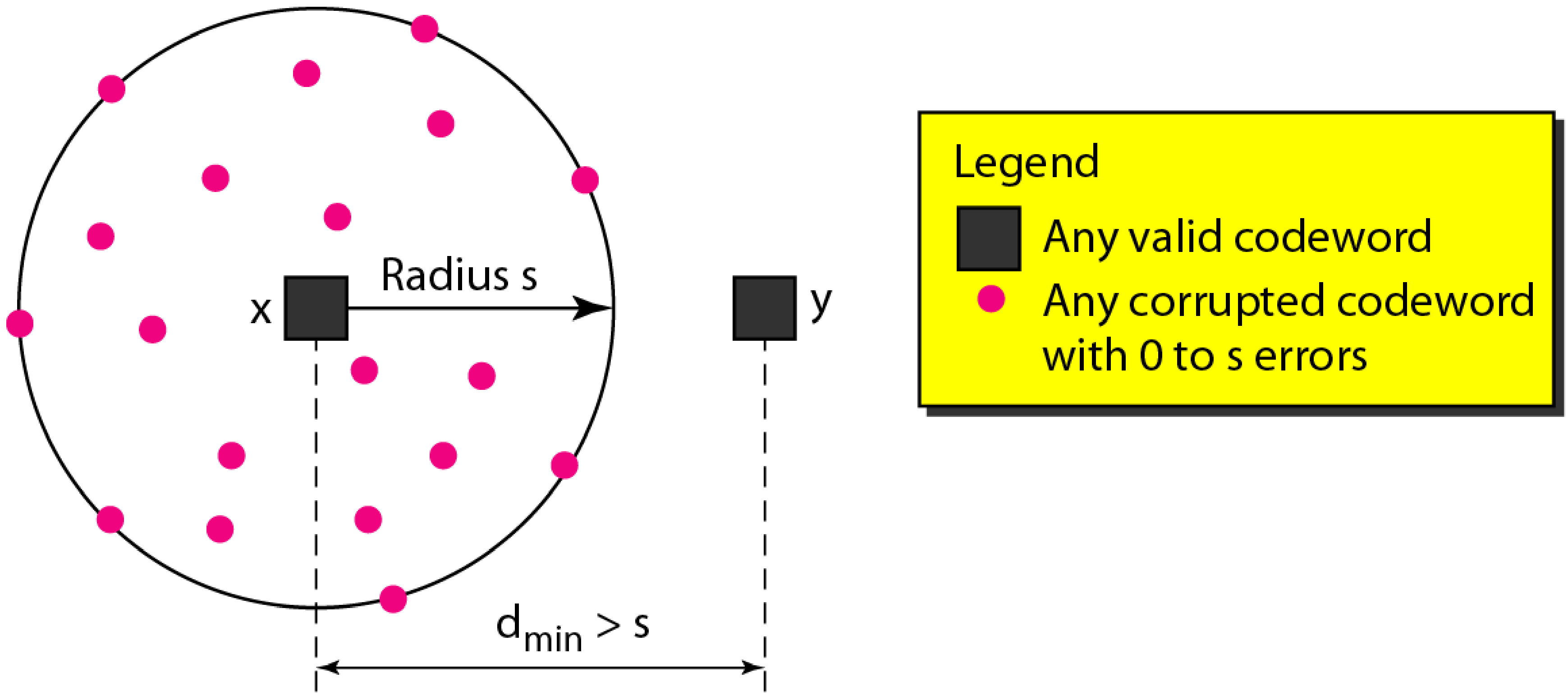
Hamming Distance for Error Detection

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

- In our second block code scheme with minimum hamming distance $d_{min} = 3$.
  - This code can detect up to two errors.
  - Again, when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver can detect the error.
  - 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.

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

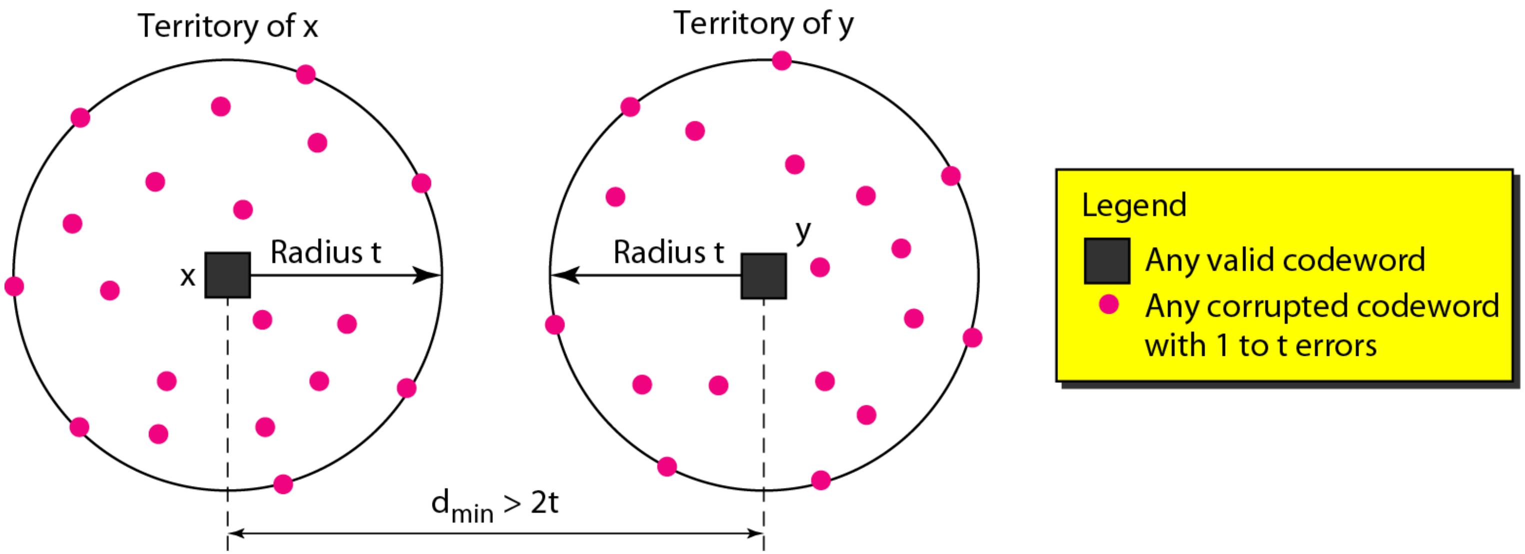
Geometric concept for finding d_{min} in error detection



Minimum Distance for Error Correction

- When a received codeword is not a valid, the receiver need to decide which valid codeword was actually sent.
- The decision is based on the concept of territory, an exclusive area surrounding the codeword.
- Suppose a codeword *x* is corrupted by *t* bits or less. Then this corrupted codeword is located either inside or on the perimeter of the circle.
- If the received code word belongs to this territory, we can decide that the original codeword is the one at the center.

Geometric concept for finding d_{min} in error correction



To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Simple parity-check code

- In a simple parity-check code, a k-bit dataword is changed to an n-bit codeword where n = k+1. The extra bit, called the parity bit
- **Even parity:** An extra bit (parity bit) is added to make the total number of 1's in the codeword even.
- **Odd parity:** An extra bit (parity bit) is added to make the total number of 1's in the codeword odd.
- A simple parity-check code, is a single-bit error-detecting code in which the minimum Hamming distance is $d_{min} = 2$. Thus, it can detect a single-bit error.

Simple parity-check code (Even parity)

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

- Assume a sender sends the dataword 1011. We examine five cases:
  - ✓ Received codeword is 10111 (No error). The dataword is 1011.
  - ✓ Received codeword is 10**0**11 (1 bit error). Error.
  - ✓ Received codeword is 1011**0** (1 bit error). Error.
  - ✓ Received codeword is **0**011**0** (2 bit error). The dataword 0011 is created (wrongly) at the receiver.
  - ✓ Received codeword is **010**11 (3 bit error). Error.

- A simple parity-check code can detect an odd number of errors.

Two-dimensional parity-check code

							Row parities
1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Column parities

a. Design of row and column parities

Two-dimensional parity-check code

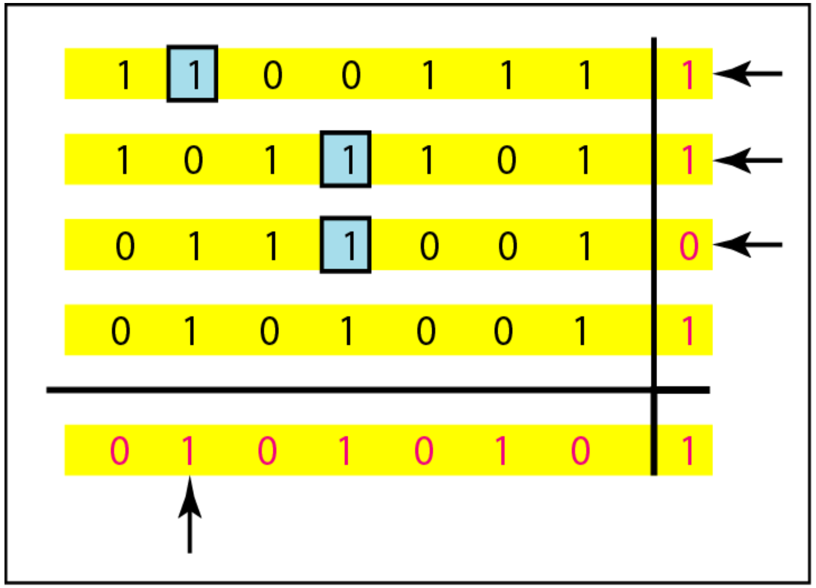
1	1	0	0	1	1	1	1
1	0	$\boxed{1}$	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

b. One error affects two parities

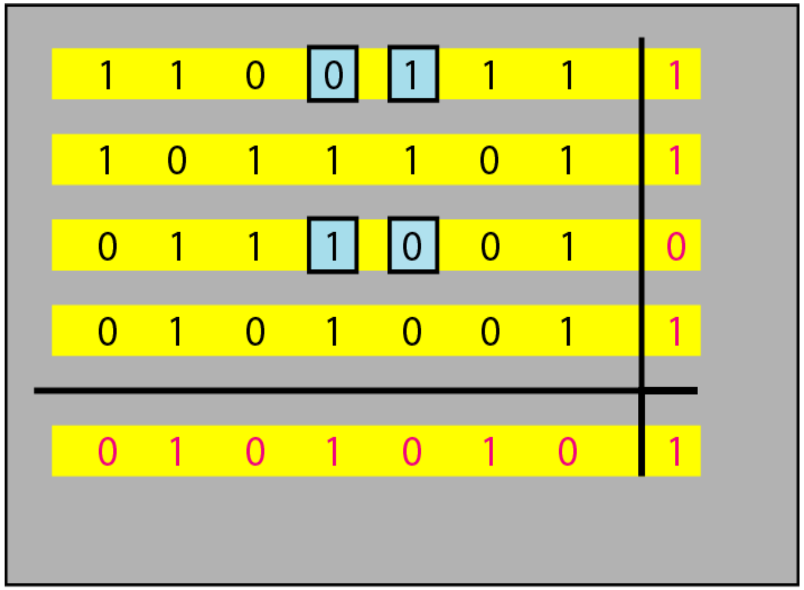
1	1	0	0	1	1	1	1
1	0	$\boxed{1}$	1	$\boxed{1}$	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

c. Two errors affect two parities

Two-dimensional parity-check code



d. Three errors affect four parities



e. Four errors cannot be detected

Hamming code

- Hamming codes were originally designed with $d_{min} = 3$, that is…
  - It can detect up to 2 errors.
  - Correct one single bit error.

Hamming code

Length of dataword = k	Add m bits

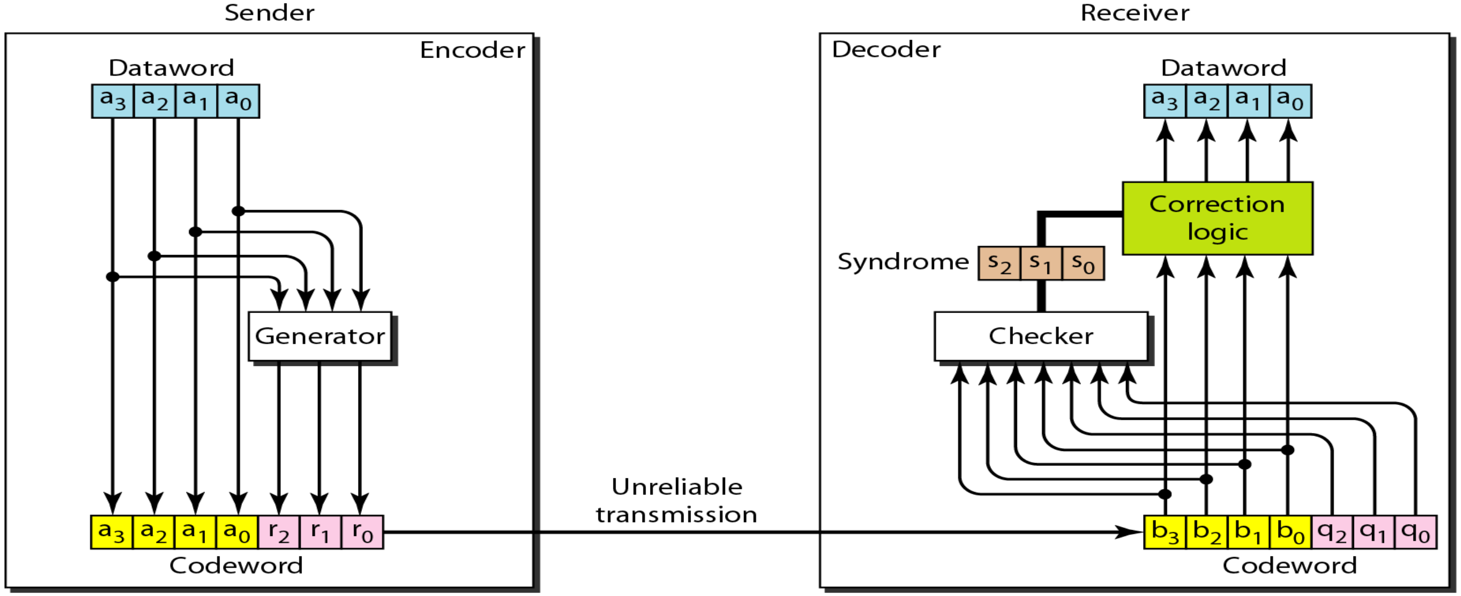
Length of codeword = n

- Relationship between n and k in Hamming code. Notice that,
  - $k = n - m$ and
  - $n = 2^m - 1$, that is m-bits should identify single bit error occurring in any n bits and no-error condition

- E.g. if $\mathrm{m} = 2$, then $\mathrm{n} = 3$ and $\mathrm{k} = 1$
- E.g. if $\mathrm{m} = 3$, then $\mathrm{n} = 7$ and $\mathrm{k} = 4$
- E.g. if $\mathrm{m} = 4$, then $\mathrm{n} = 15$ and $\mathrm{k} = 11$

- E.g. We need a dataword of at least 7 bits
  - We need to make $\mathrm{k} = \mathrm{n} - \mathrm{m}$ greater than or equal to 7
  - If we set $\mathrm{m} = 4$, then $\mathrm{n} = 2^4 - 1 = 15$ and $\mathrm{k} = 15 - 4 = 11$, which satisfies the condition.

The encoder and decoder for a Hamming code



$$r_0 = a_2 + a_1 + a_0 \quad modulo\ 2$$
$$r_1 = a_3 + a_2 + a_1 \quad modulo\ 2$$
$$r_2 = a_1 + a_0 + a_3 \quad modulo\ 2$$

$$s_0 = b_2 + b_1 + b_0 + q_0 \quad modulo\ 2$$
$$s_1 = b_3 + b_2 + b_1 + q_1 \quad modulo\ 2$$
$$s_2 = b_1 + b_0 + b_3 + q_2 \quad modulo\ 2$$

$$1101 \longrightarrow 1101\mathbf{000}$$
$$1010 \longrightarrow 1010\mathbf{001}$$

Hamming code

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

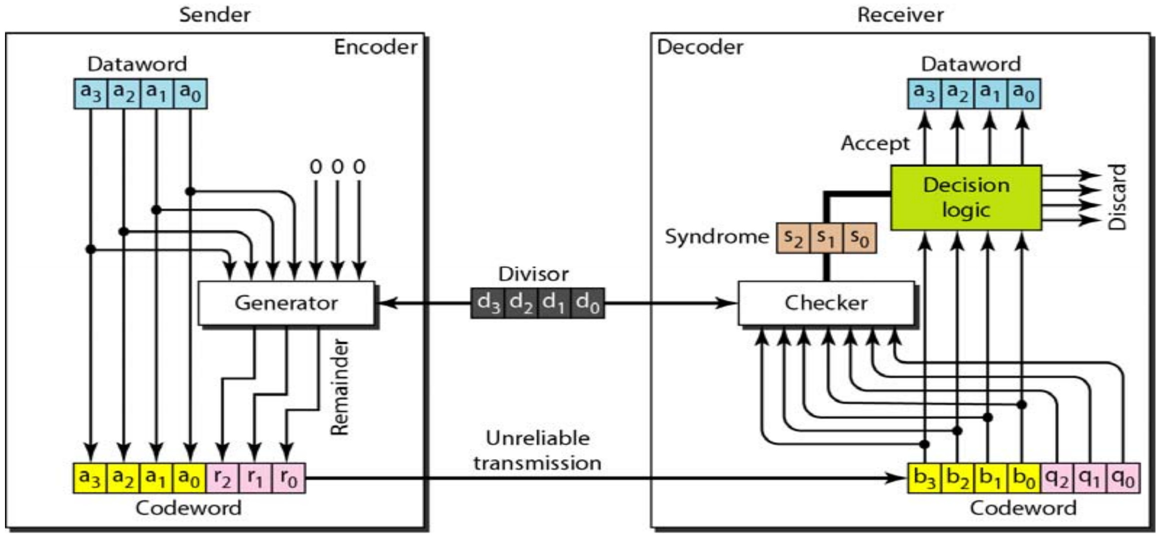
Hamming code

Logical decision made by the correction logic analyzer

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

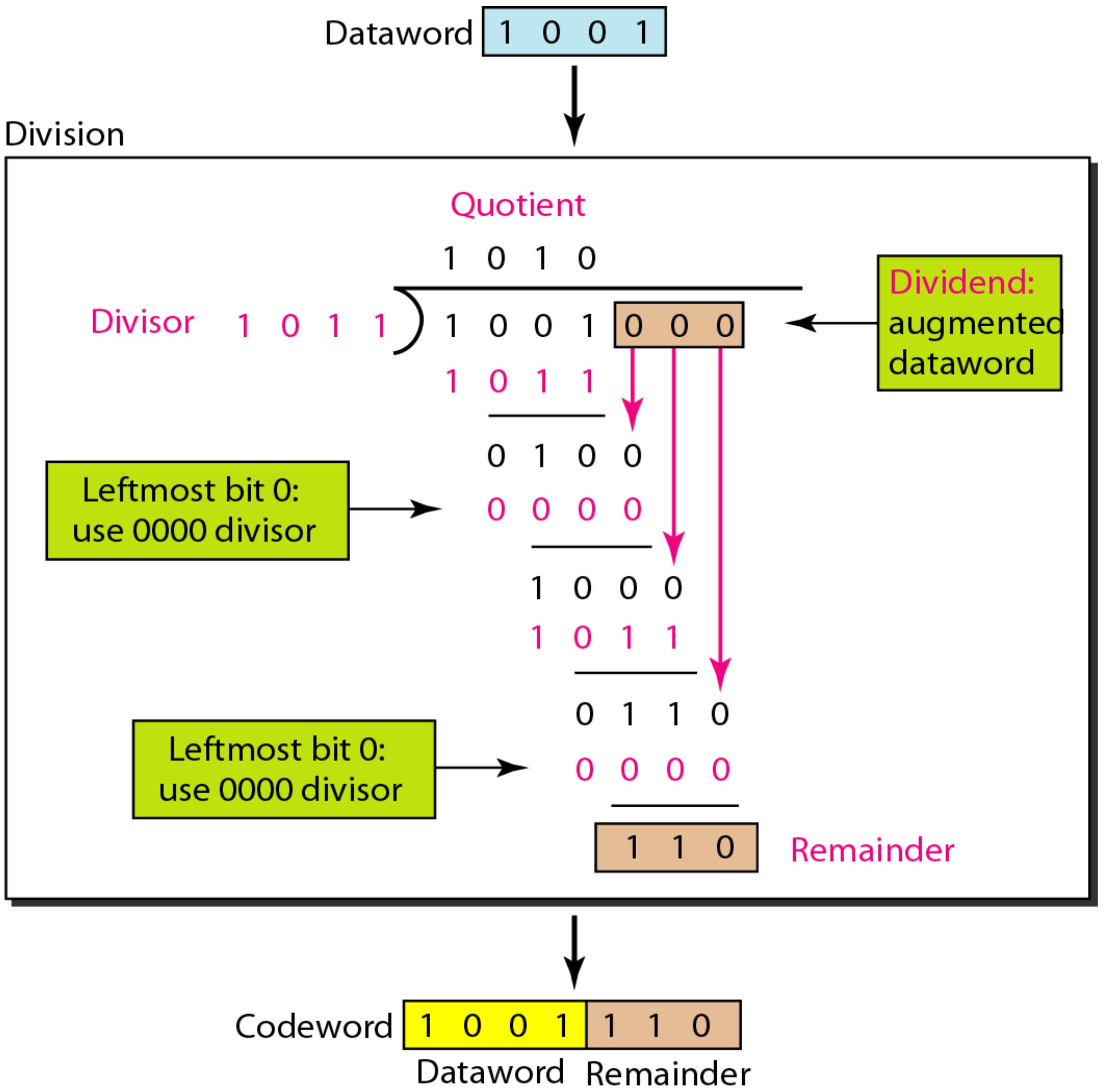
dataword	Codeword (sender)	Codeword (receiver)	syndrom
0100	0100011	0100011	000
0111	0111001	0**0**11001	011
1101	1101000	11010**0**0	010

Cyclic Redundancy Check (CRC)

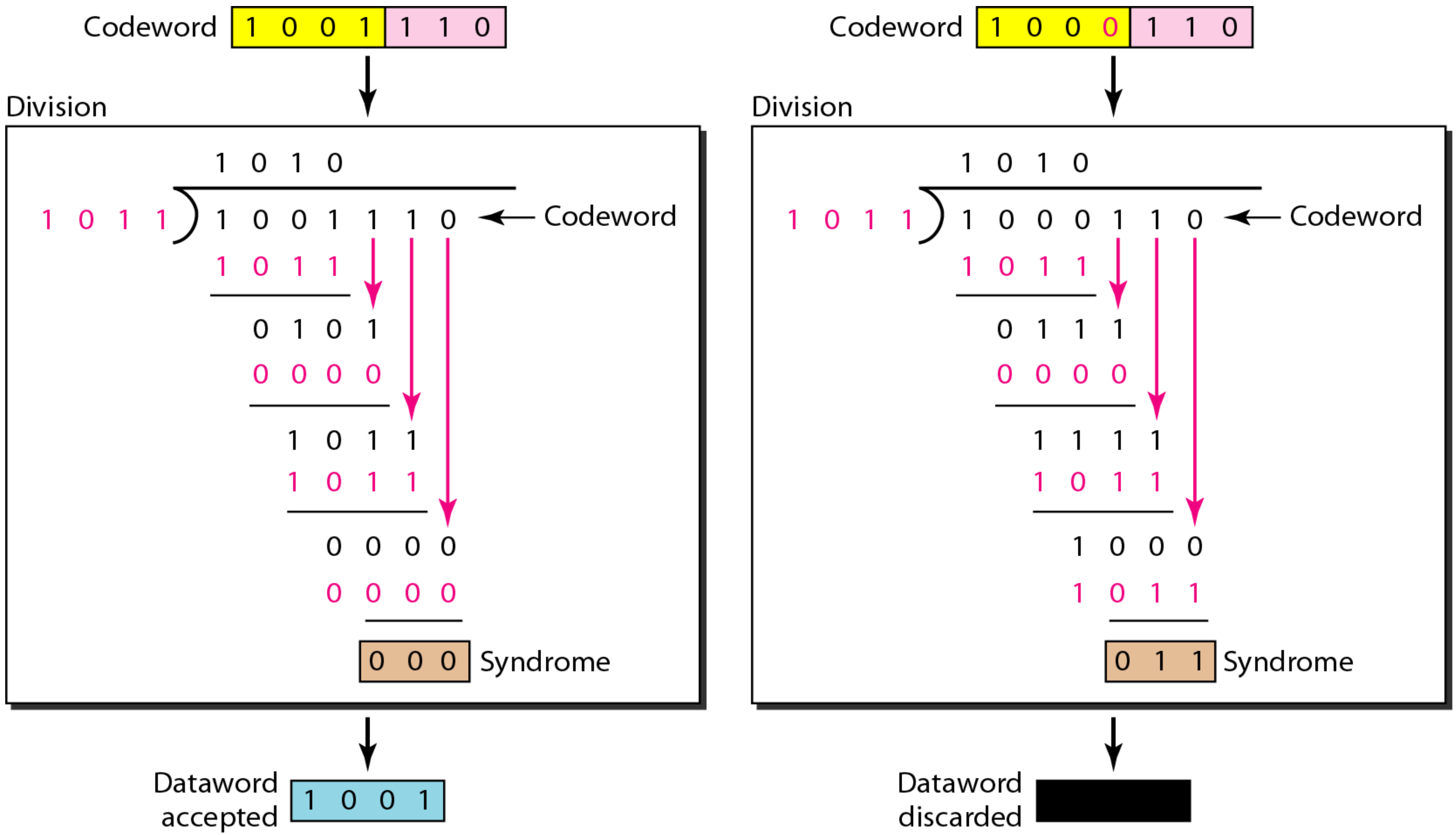


- Encoder
  - The dataword has k bits (4) & The codeword has n bits (7)
  - The size of the dataword is augmented by adding $n - k$ (3) 0s.
  - The n-bit result is fed to the generator.
  - The generator uses a divisor of size $n - k + 1$ (4) to divides the augmented dataword by the divisor.
  - The remainder is appended to create the codeword
- Decoder
  - Received N bits is fed to the checker
  - The remainder produced by the checker $(n - k)$ bits are all 0s the codeword is accepted, rejected otherwise

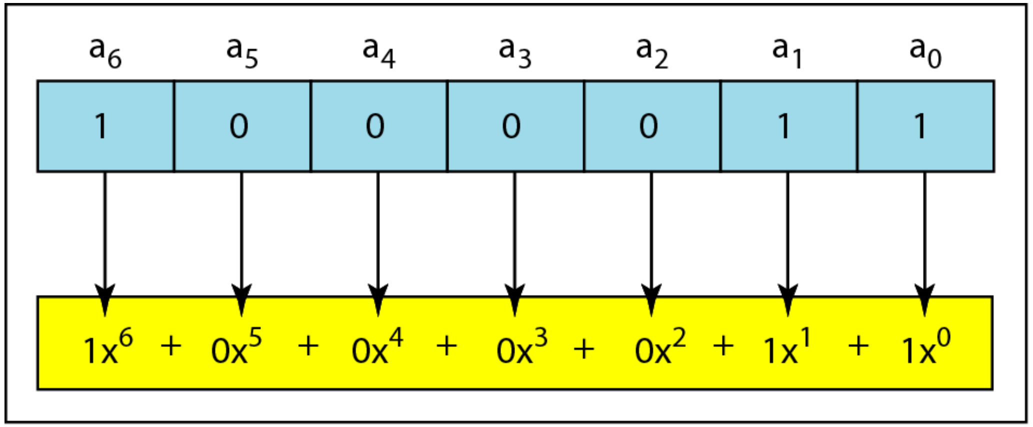
Division in CRC Encoder



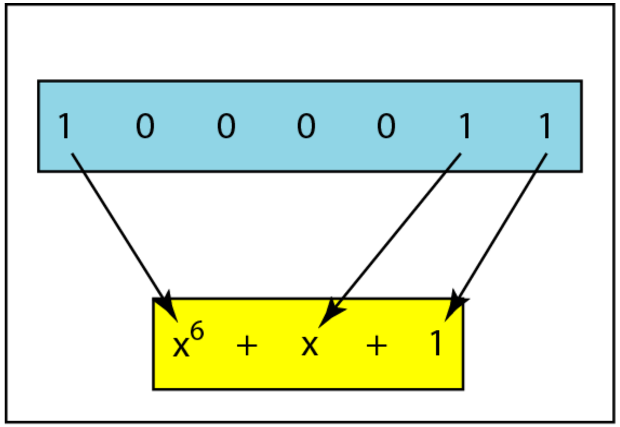
Division in CRC Decoder



A polynomial to represent a binary word

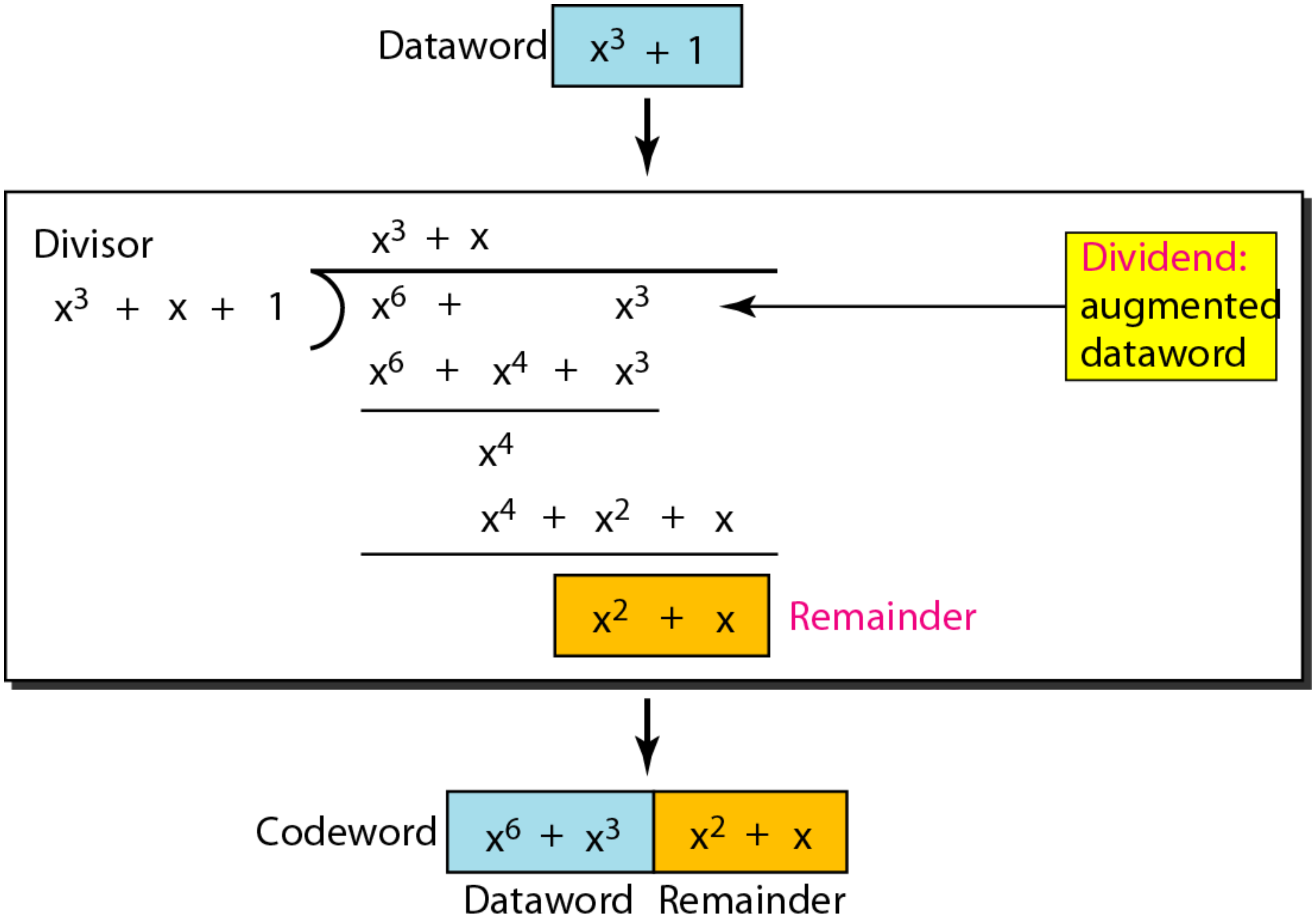


a. Binary pattern and polynomial



b. Short form

CRC division using polynomials



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