

# Data Link Layer I

## Introduction and Services

## Error Detection and Correction

### Agenda

- Data-Link Layer Introduction and Services
- Error Detection and Correction
  - Information Theory
  - Error Introduction
  - Block Coding
  - Hamming Distance
  - Linear Block Codes
  - Checksum
  - CRC - Cyclic Redundancy Check



# Data-Link Layer I

Our goals:

Understand principles behind link layer services:

- Introduction, Services
- Error detection, correction
- Multiple access protocols
- Sharing a broadcast channel: multiple access
- Data-Link Layer Addressing
- ARP/RARP
- LAN: Ethernet
- LAN: Switches
- LAN: VLANs
- Simple web request description

Implementation of various link layer technologies

# Introduction, Services

## Data Link Layer Context

Datagram transferred by different link protocols over different links:

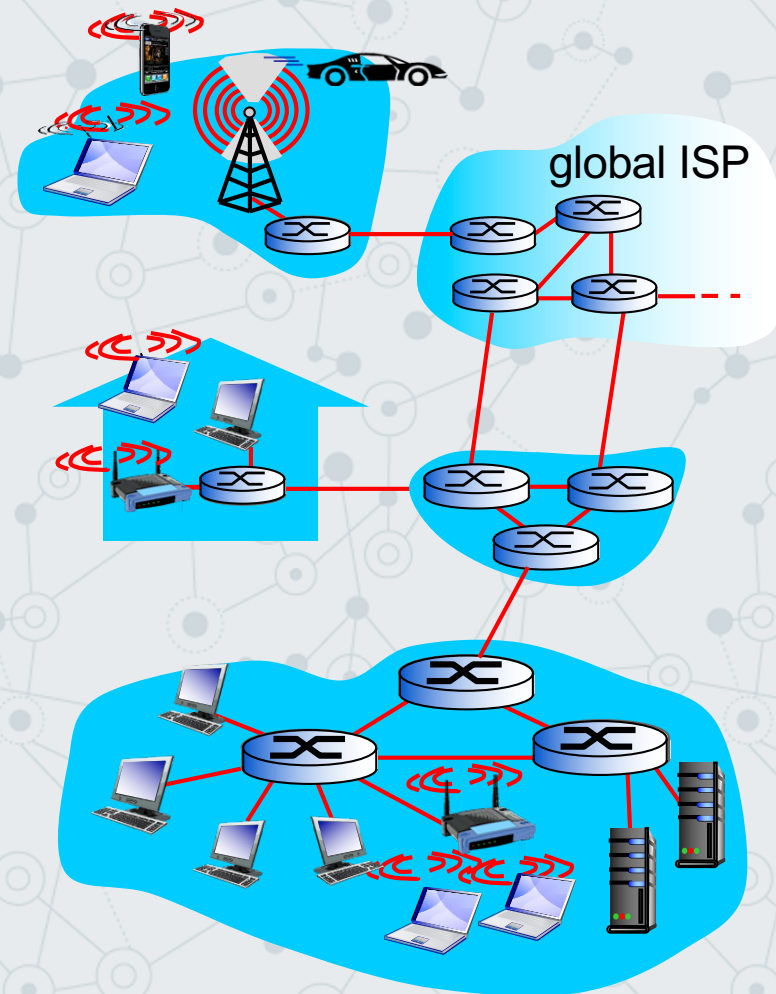
- e.g., Ethernet on first link, Frame Relay on intermediate links, 802.11 (WiFi) on last link

Each link protocol provides different services

- e.g., may or may not provide RDT - reliable data transfer (надежная) over link

### Transportation analogy:

- ❖ trip from Riga to Lausanne
- taxi: Riga to RIX
- plane: RIX to Geneva
- train: Geneva to Lausanne
- ❖ tourist = datagram
- ❖ transport segment = communication link
- ❖ transportation mode = link layer protocol
- ❖ travel agent = routing algorithm



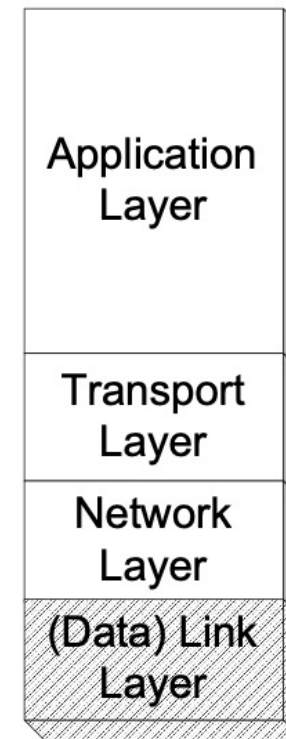
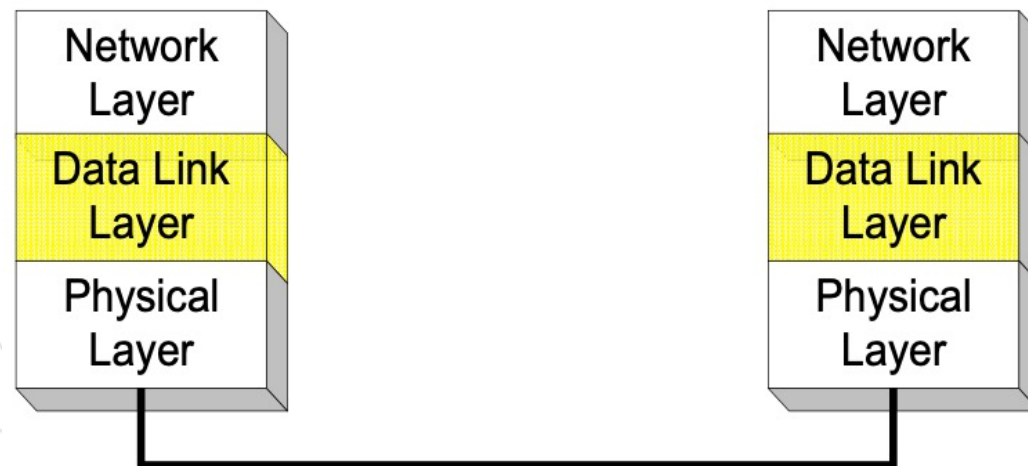
# Introduction, Services

## TCP/IP Suite and OSI Reference Model

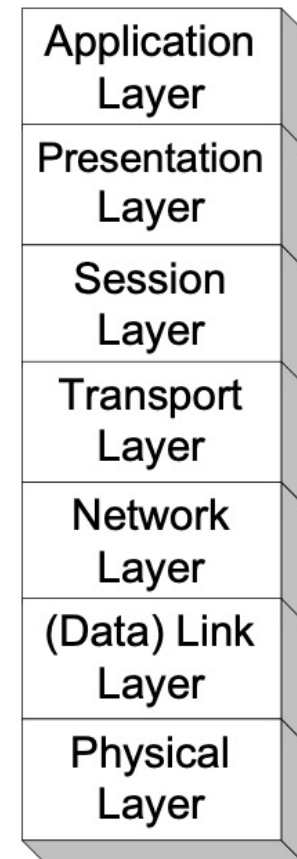
The TCP/IP protocol stack does not define the lower layers of a complete protocol stack →

**The main tasks** of the data link layer are:  
Transfer data from the network layer of one machine to the network layer of another machine.

Convert the raw bit stream of the physical layer into groups of bits (“**frames**”).



**TCP/IP Suite**



**OSI Reference Model**



# Introduction, Services

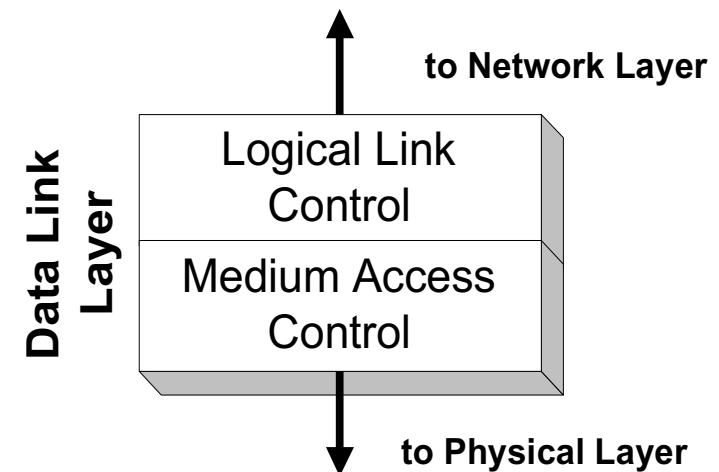
## MAC and LLC Sublayers

In any broadcast network, the stations must ensure that **only one station** transmits at a time on the shared communication channel

The protocol that determines who can transmit on a broadcast channel are called **Medium Access Control (MAC)** protocol

The MAC protocol are implemented in the **MAC sublayer** which is the lower sublayer of the data link layer

The higher portion of the data link layer is often called **Logical Link Control (LLC)**

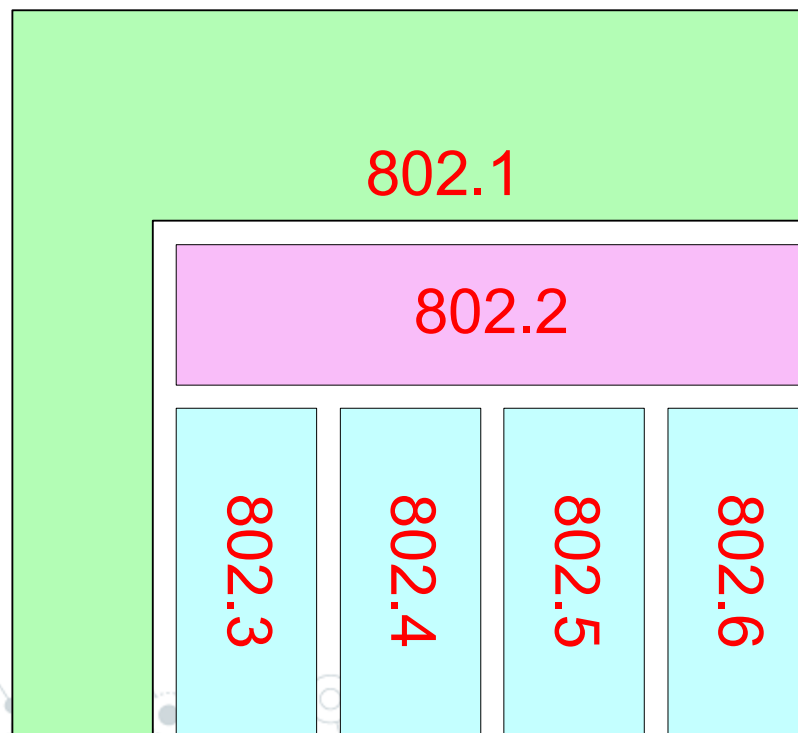


# Introduction, Services

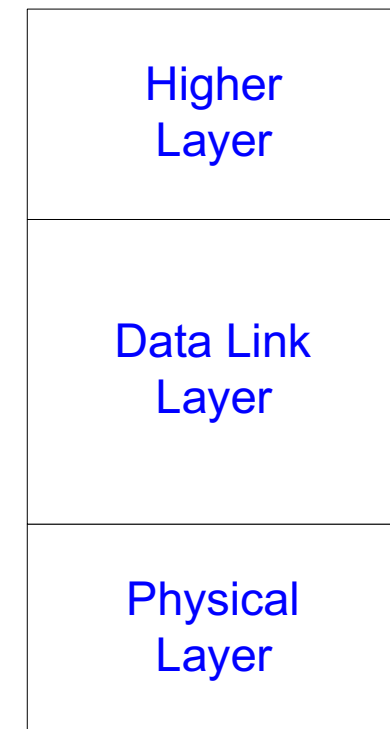
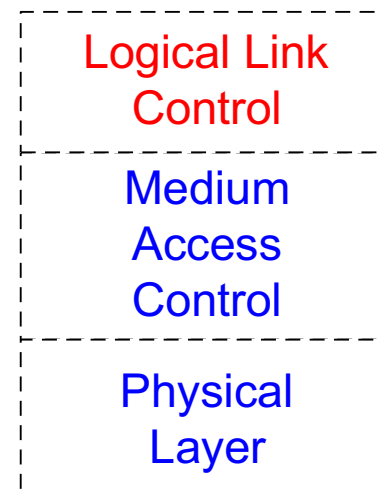
## IEEE 802 Standards

IEEE 802 is a family of standards for LANs, which defines an LLC and several MAC sublayers.

### IEEE 802 standard



### IEEE Reference Model



# Introduction, Services

## Data Link Layer Services

- Framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses used in frame headers to identify source, destination (different from IP address!)
- Reliable (надежная) delivery between LAN nodes
  - not used on low bit-error link (fiber, twisted pair)
  - used on wireless links: high error rates
- Flow control:
  - pacing between adjacent sending and receiving nodes
- Error detection:
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors: drops frame or signals sender for retransmission
- Error correction:
  - receiver identifies and corrects bit error(s) without resorting to retransmission
- Half-duplex and Full-duplex
  - with half duplex, nodes at both ends of link can transmit, but not at same time

# Introduction, Services

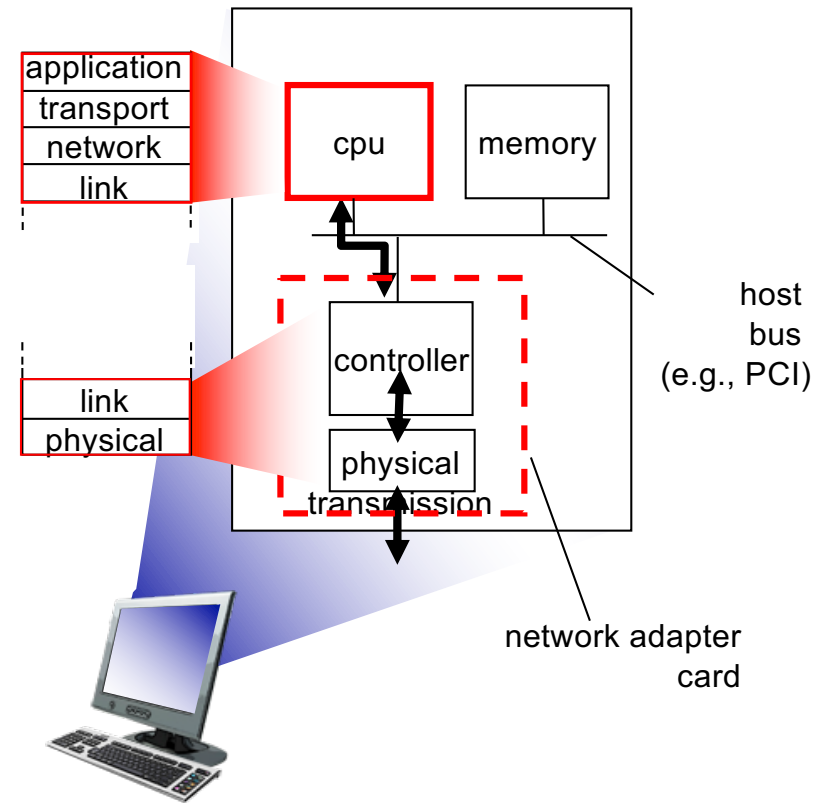
## Where is the Data Link Layer implemented?

There is in every host.

Link layer implemented in “adaptor” (aka **network interface card** NIC) or on a chip.

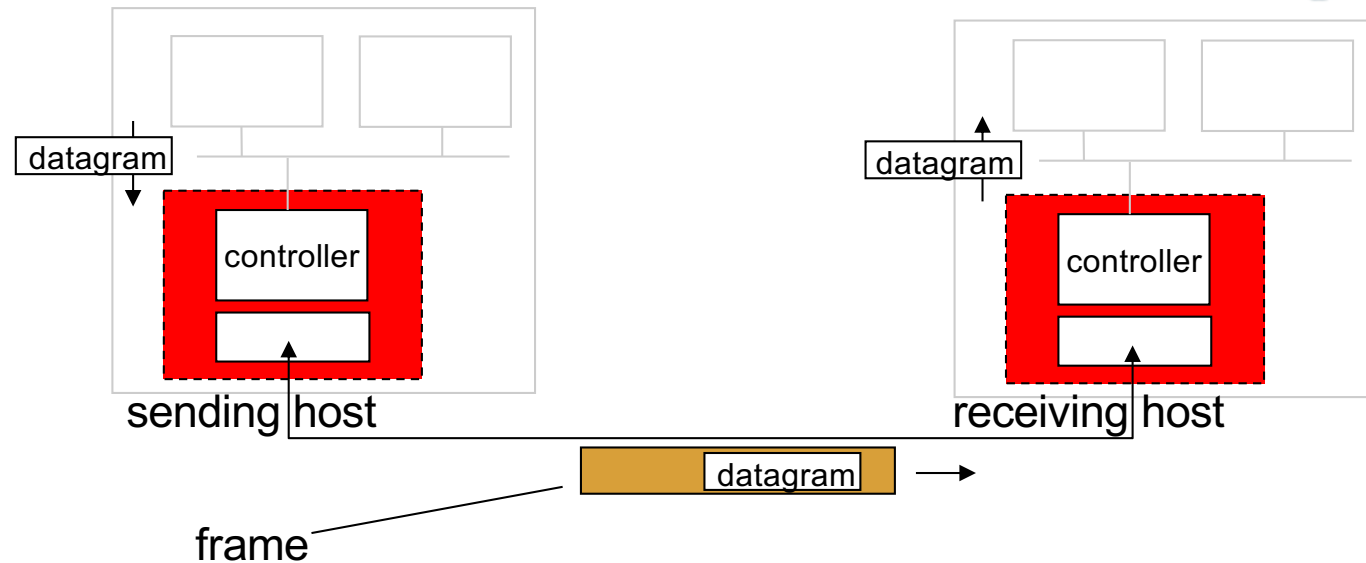
- Ethernet card;
- 802.11 card;
- Ethernet chipset;
- Virtual card.

Physical attaches into host's system buses.  
It is a combination of hardware, software, firmware.



# Introduction, Services

## Adaptors communicating



Sending side:

- encapsulates datagram in frame
- adds error checking bits, rdt, flow control, etc.

Receiving side

- looks for errors, rdt, flow control, etc
- extracts datagram, passes to upper layer at receiving side

# Error Detection and Correction

## Information Theory

### Data Rate Limits

Data rate in bits per second over a channel depends on three factors:

1. The bandwidth available
2. The level of the signals we use
3. The quality of the channel (the level of noise)

Changing one limit changes the rest.

Example. Increasing the levels of a signal increases the probability of an error occurring. Why?

- The bit rate of a system increases with an increase in the number of signal levels we use to denote a symbol. **It's good!**
- A symbol can consist of a “single” or “n” bits. The number of signal levels  $L=2^n$ .
- As the number of levels  $L$  increases, the spacing between level decreases => increasing the error probability in the presence of transmission noise. **This is bad !**



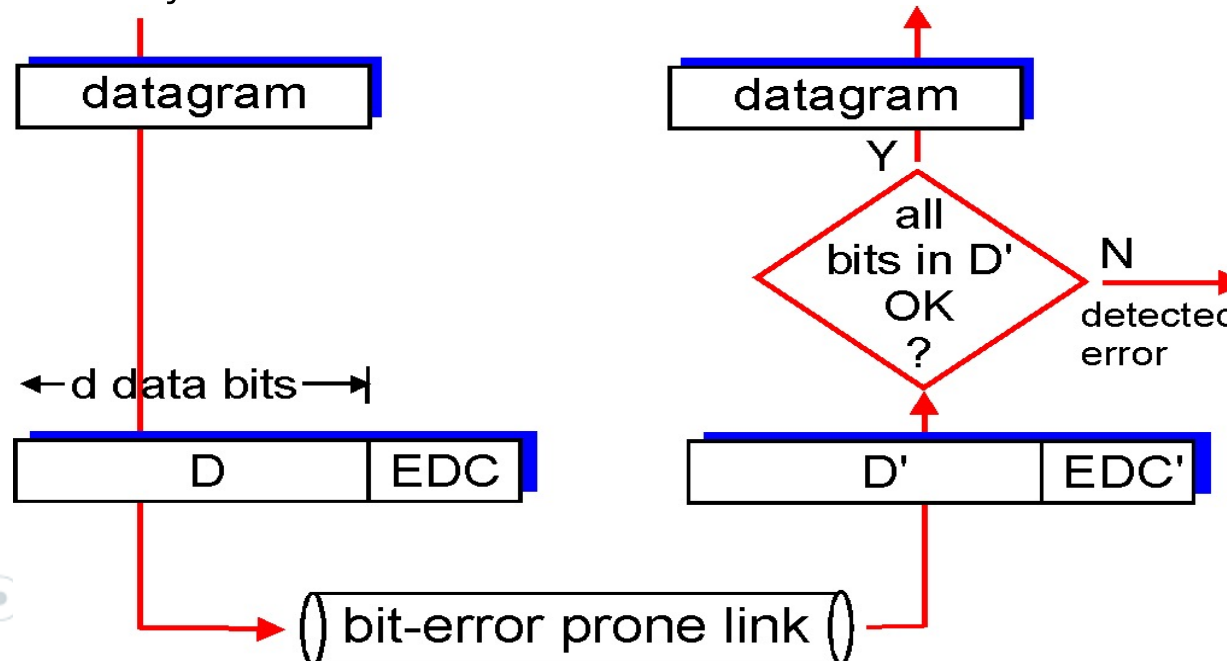
# Error Detection and Correction

## Information Theory

EDC= Error Detection and Correction bits (redundancy)

D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
- Protocol may miss some errors, but rarely
- Larger EDC field yields better detection and correction



# Error Detection and Correction

## Information Theory: Signal to Noise Ratio (SNR)

**SNR** used to measure the quality of a system.

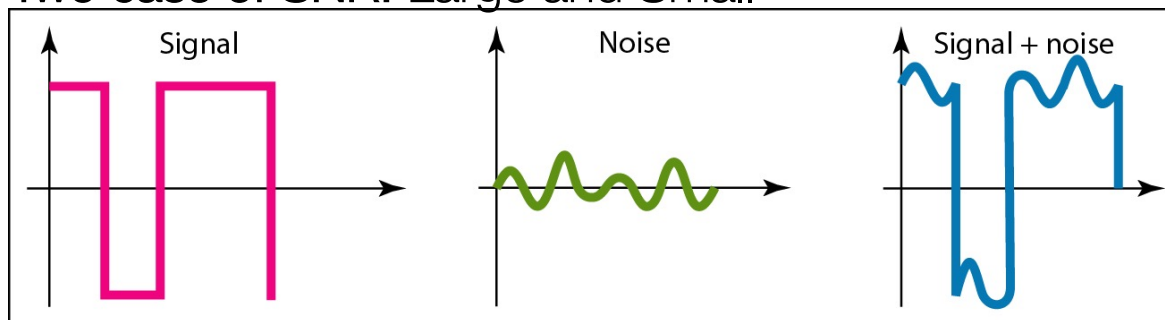
SNR indicates the power (strength) of the signal  $S$  relative to the noise power  $N$  in the system

Spower/Npower.

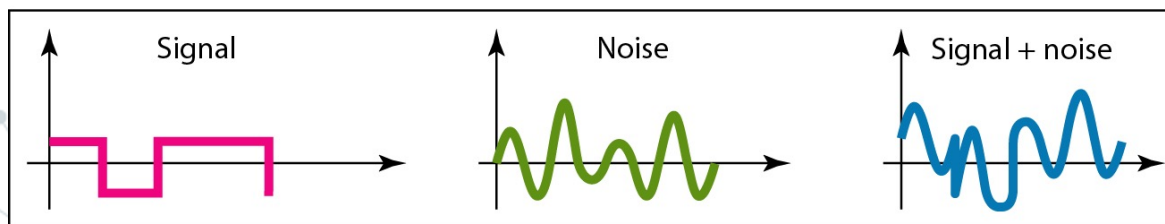
It is usually given in **decibel (dB)** and referred to as

$$\text{SNR}_{\text{dB}} = 10 \log_{10}(S/N) [\text{dB}]$$

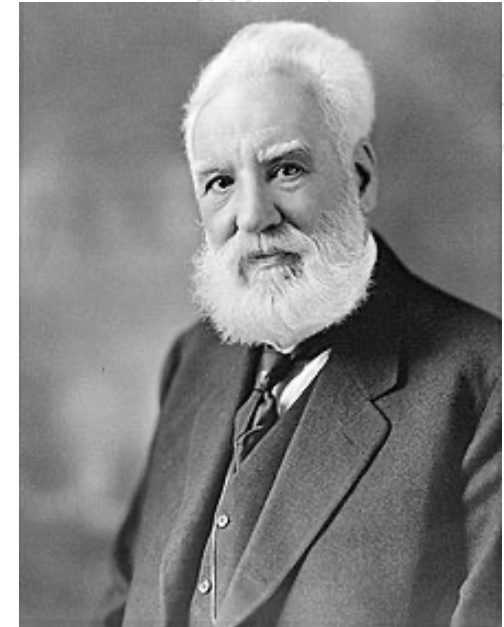
Two case of SNR: Large and Small



a. Large SNR



b. Small SNR



*Alexander Graham Bell (1847-1922)*

The **bel (B)** was originally conceived as a *power ratio* unit that could be used to quantify signal loss in telegraph and telephone circuits.

# Error Detection and Correction

## Information Theory: Nyquist theorem

**Nyquist theorem** states that for a noiseless channel:

$$C = 2 \cdot B \cdot \log_2 L = 2 \cdot B \cdot \log_2 2^n$$

Where

C = capacity in bps (or bit rate);

B = bandwidth in Hz (assuming 2 symbols/per cycle and first harmonic);

L = signal levels;

n = numbers of bit per symbols (level).

### Example 1.

We have a noiseless channel with a bandwidth of 3000 Hz transmitting a signal with 2 signal levels.

The maximum bit rate can be calculated as

$$C = 2 \cdot 3000 \cdot \log_2 2 = 6000 \text{ bps}$$

### Example 2.

We have a noiseless channel with a bandwidth of 3000 Hz transmitting a signal with 4 signal levels (for each level, we send 2 bits).

The maximum bit rate equal

$$C = 2 \cdot 3000 \cdot \log_2 4 = 12000 \text{ bps}$$

Harry Nyquist



Harry Nyquist

**Born**

February 7, 1889

Nilsby, Stora Kil, [Värmland](#),  
[Sweden](#)

**Died**

April 4, 1976 (aged 87)

[Harlingen](#), [Texas](#), U.S.

# Error Detection and Correction

## Information Theory: Nyquist theorem (Cont.1)

### Example 3.

We need to send 265 kbps over a noiseless channel with a bandwidth of 20 kHz.  
How many signal levels do we need?

We can use the Nyquist formula as shown:

$$C = 2 \cdot B \cdot \log_2 L = 2 \cdot B \cdot \log_2 2^n \rightarrow \log_2 L = C / (2 \cdot B) \rightarrow L = 2^{C / (2 \cdot B)}$$

If  $C=265\,000$ ,  $B=20\,000$  then

$$L = 2^{265000 / (2 \cdot 20000)} = 2^{265 / (2 \cdot 20)} = 2^{26.5 / 4} = 2^{6.625} = 98.7 \text{ [levels]}$$

Since this result is not a power of 2, we need to either or increase the number of levels or reduce the bit rate:

If we have 128 levels, the bit rate  $C$  is 280 kbps.

If we have 64 levels, the bit rate  $C$  is 240 kbps.

# Error Detection and Correction

## Information Theory: Shannon theorem

Shannon's theorem gives the capacity  $C$  of a system with bandwidth  $B$  in the presence of noise with SNR

$$C = B \cdot \log_2(1 + \text{SNR})$$

### Example 4.

We have an extremely noisy channel in which the value of the **signal-to-noise ratio** is almost zero ( $\text{SNR}=0$ ).

For this channel the capacity  $C$  is calculated as

$$C = B \cdot \log_2(1 + 0) = B \cdot \log_2(1) = B \cdot 0 = 0$$

Capacity of this channel is 0 regardless of the bandwidth.

We **cannot receive any data** through this channel!

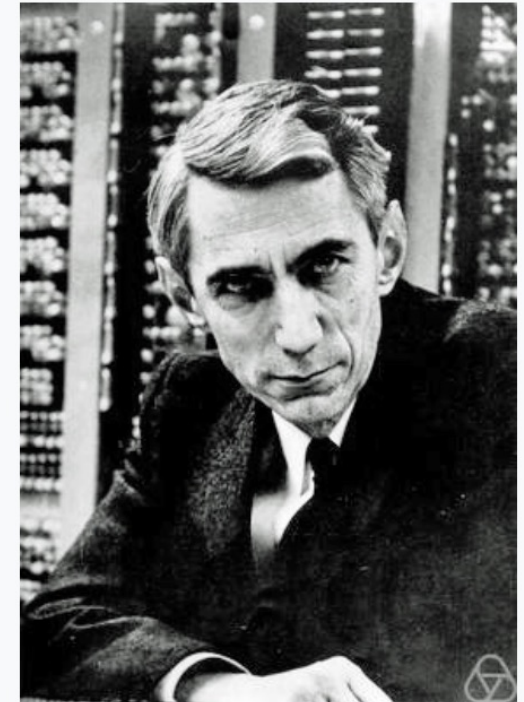
### Example 5.

If  $\text{SNR}_{\text{dB}} = 36$  dB and the channel bandwidth is 2 MHz, then **signal-to-noise ratio** SNR and theoretical channel capacity  $C$  can be calculated as:

$$\text{SNR}_{\text{dB}} = 10 \log_{10} \text{SNR} \rightarrow \text{SNR} = 10^{(\text{SNR}_{\text{dB}})/10} \rightarrow \text{SNR} = 10^{36/10} = 10^{3.6} = 3981$$

$$C = B \cdot \log_2(1 + \text{SNR}) = 2 \cdot 10^6 \cdot \log_2(1 + 3981) = 2 \cdot 10^6 \cdot \log_2 3982 = 2 \cdot 10^6 \cdot 11,9 = 24 \text{ Mbps}$$

Claude Shannon



**Born**

April 30, 1916

[Petoskey, Michigan](#), United States

**Died**

February 24, 2001 (aged 84)

[Medford, Massachusetts](#), United States

# Error Detection and Correction

## Information Theory: Nyquist & Shannon's theorems

### Example 6.

We have a channel with a 1-MHz bandwidth ( $B = 10^6$  Hz). The SNR for this channel is 63. What are the appropriate bit rate  $C$ ? and signal level  $L$ ?

### Solution

The Shannon capacity gives us the upper limit of  $C$ ; the Nyquist formula tells us how many signal levels  $L$  we need:

$$C_{\max} = B \cdot \log_2(1 + \text{SNR})$$
$$C = 2 \cdot B \cdot \log_2 L = 2 \cdot B \cdot \log_2 2^n \rightarrow L = 2^{C_{\max}/(2 \cdot B)}$$

First, we use the Shannon formula to find the upper limit:

$$C_{\max} = B \cdot \log_2(1 + \text{SNR}) = 10^6 \cdot \log_2(1 + 63) = 10^6 \cdot \log_2 64 = 10^6 \cdot 6 = 6 \text{ Mbps}$$

The Shannon formula gives us  $C_{\max} = 6$  Mbps, the upper limit.

For better performance we choose something lower,  $C = 4$  Mbps, for example.

Then we use the Nyquist formula to find the number of signal levels:

$$4 \text{ Mbps} = 2 \cdot 1 \text{ MHz} \cdot \log_2 L \rightarrow L = 2^{4\,000\,000/(2 \cdot 1\,000\,000)} = 2^{4/2} = 2^2 = 4$$



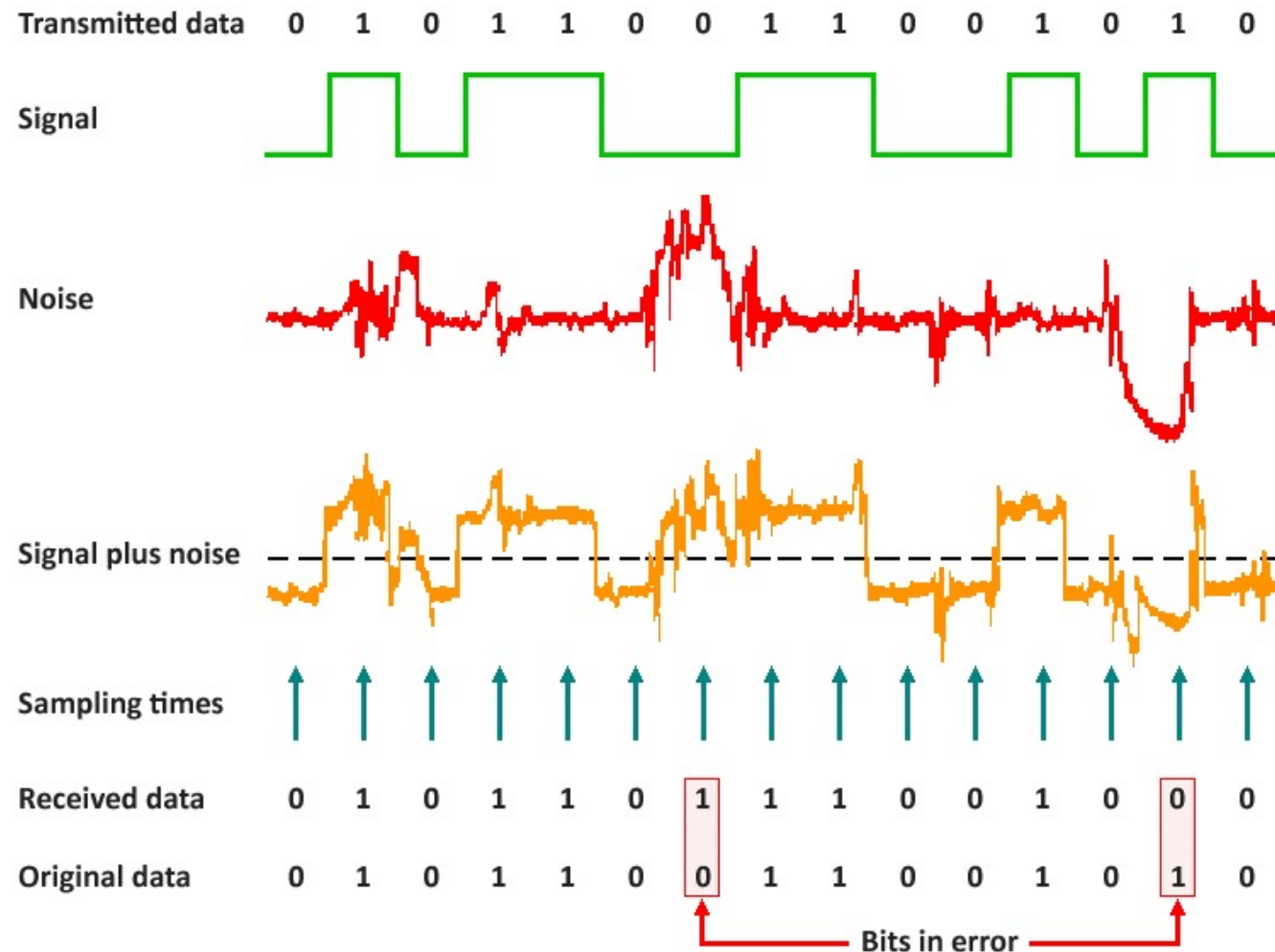
# Error Detection and Correction

## Error Detection and Correction – Error Introduction

An electromagnetic signal is subject to interference from heat, magnetism, and other forms of electricity

Error is result of  
Attenuation,  
Distortion,  
Noise sum.

Noise Problems:



# Error Detection and Correction

## Error Introduction: Attenuation

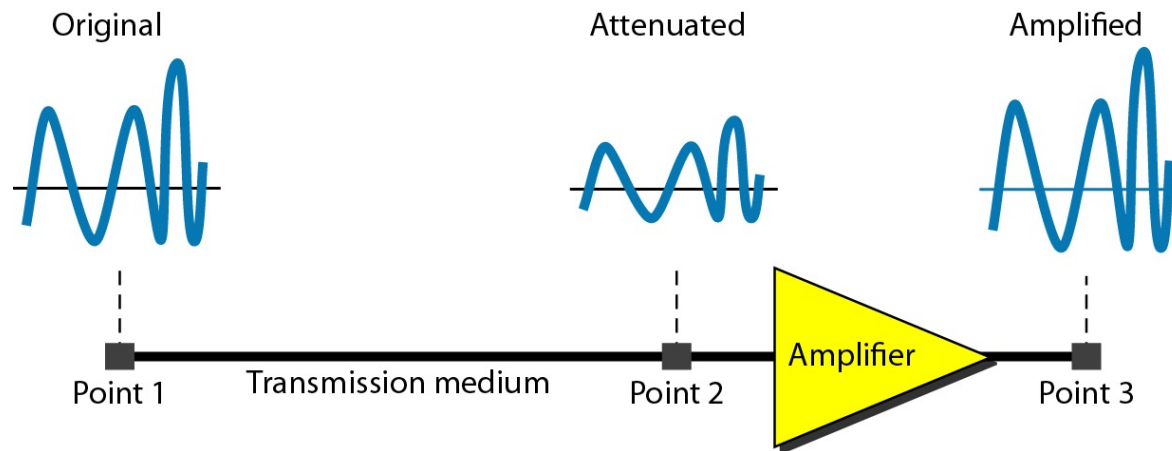
**Attenuation** means loss of energy -> weaker signal.

When a signal travels through a medium it loses energy overcoming the resistance of the medium

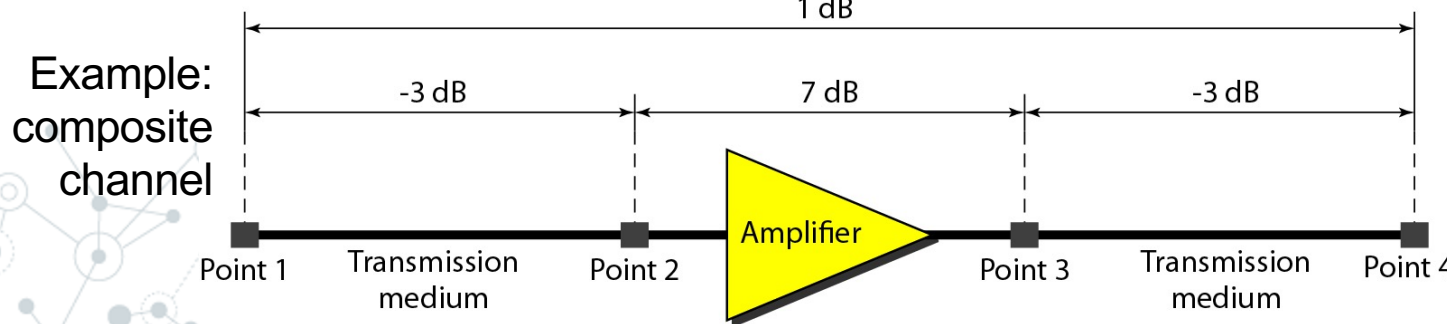
Amplifiers are used to compensate for this loss of energy by amplifying the signal

To show the loss or gain of energy the unit “decibel” is used (P1 - input signal, P2 - output signal):

$$A_{dB} = 10\log_{10}P2/P1$$



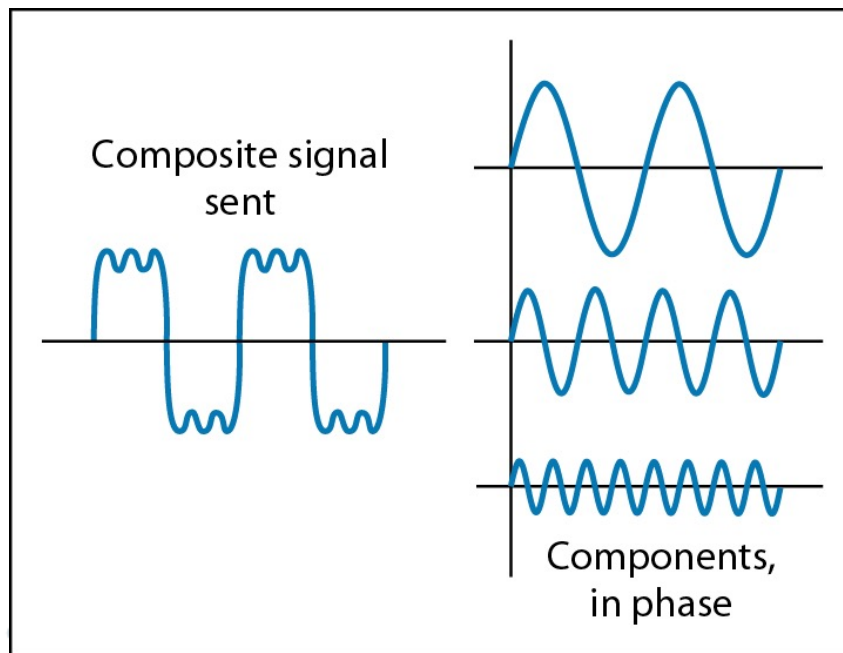
P1	P2	10LG(P2/P1)
10	0,1	-20,0
10	1	-10,0
10	0,001	-40,0
10	0,01	-30,0
10	0,1	-20,0
10	1	-10,0
10	2	-7,0
10	4	-4,0
10	5	-3,0
10	9	-0,5
10	10	0,0
10	20	3,0
10	30	4,8
10	40	6,0
10	50	7,0
10	60	7,8
10	70	8,5
10	80	9,0
10	90	9,5
10	100	10,0



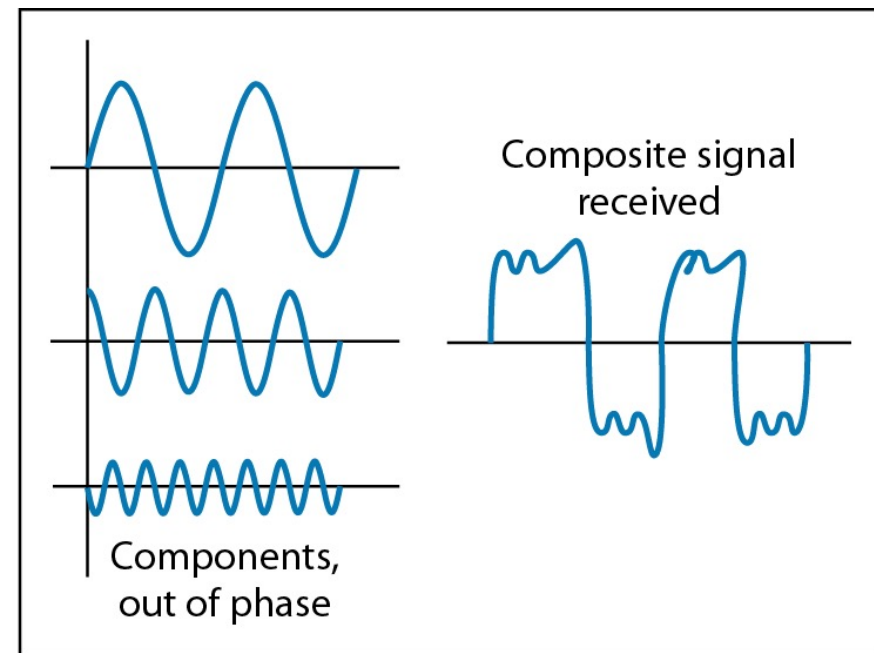
# Error Detection and Correction

## Error Introduction: Distortion

**Distortion** means that the signal changes its form or shape  
Distortion occurs in **composite** signals  
Each frequency component has its own **propagation speed** traveling through a medium.  
The different components therefore arrive with **different delays** at the receiver.  
That means that the signals have **different phases** at the receiver than they did at the source.



At the sender



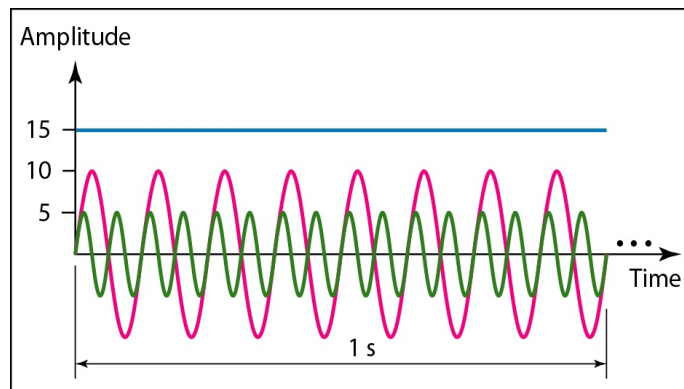
At the receiver

**Fourier analysis** is a tool which changes a time-domain signal into a frequency-domain signal and vice versa.

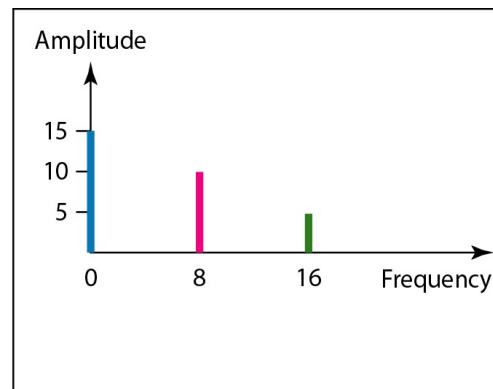
# Error Detection and Correction

## Error Introduction: Distortion – Fourier Analysis

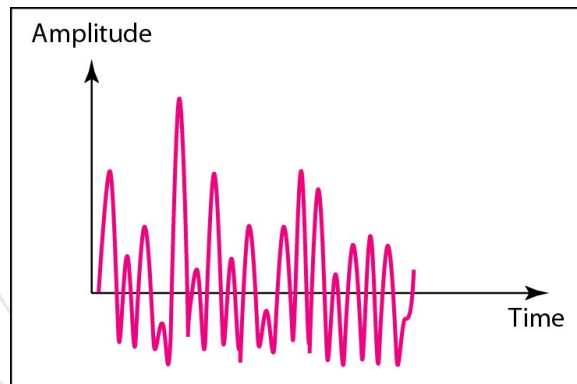
**Fourier analysis** is a tool which changes a time-domain signal into a frequency-domain and inverse.



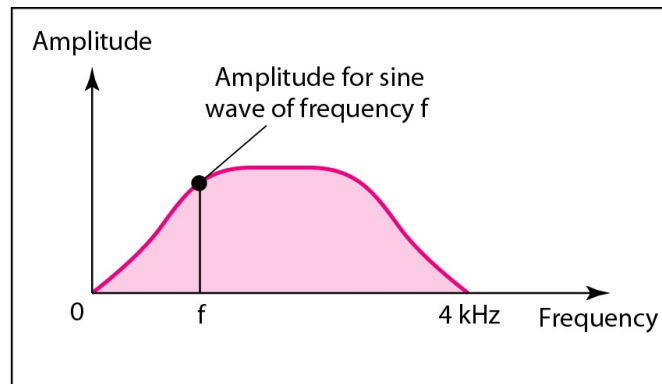
a. Time-domain representation of three sine waves with frequencies 0, 8, and 16



b. Frequency-domain representation of the same three signals



a. Time domain



b. Frequency domain

**Joseph Fourier**



Jean-Baptiste Joseph Fourier

**Born** 21 March 1768  
Auxerre, Burgundy, Kingdom of France (now in Yonne, France)  
**Died** 16 May 1830 (aged 62)  
Paris, Kingdom of France  
**Nationality** French

$$S(f) = \int_{-\infty}^{\infty} s(t) e^{-j2\pi ft} dt$$

Fourier transform

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{j2\pi ft} dt$$

Inverse Fourier transform



# Error Detection and Correction

## Error Introduction: Noise

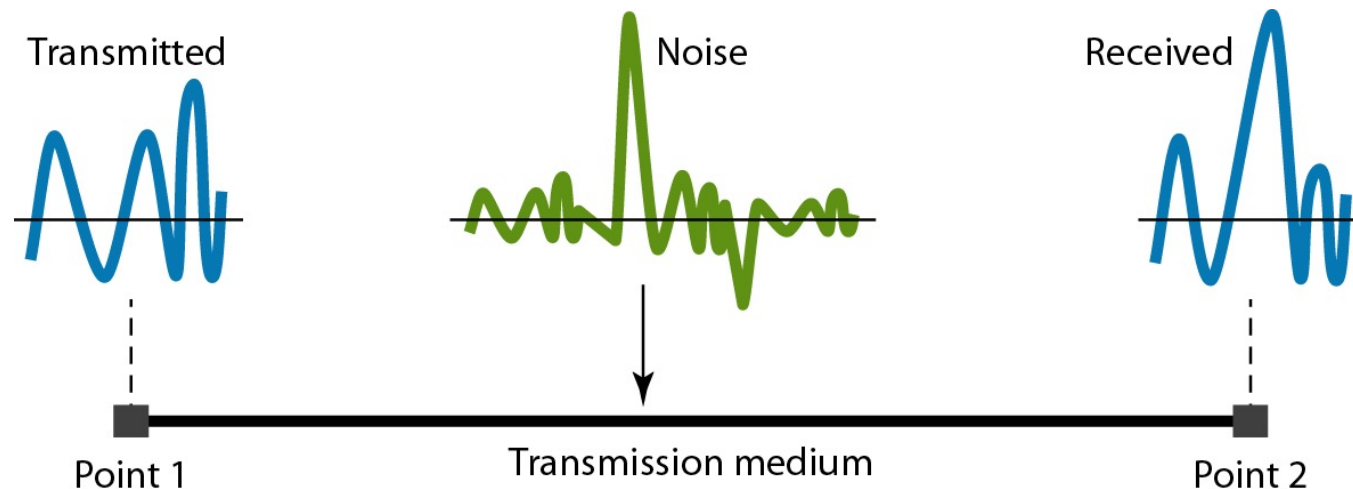
There are different types of noise:

**Thermal** - random noise of electrons in the wire creates an extra signal

**Induced** - from motors and appliances, devices act as transmitter antenna and medium as receiving antenna.

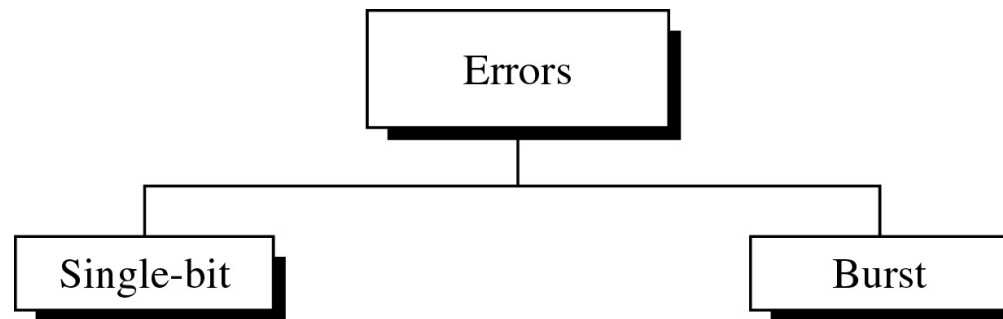
**Crosstalk** - same as above but between two wires.

**Impulse** - Spikes that result from power lines, lightning, etc.



# Error Detection and Correction

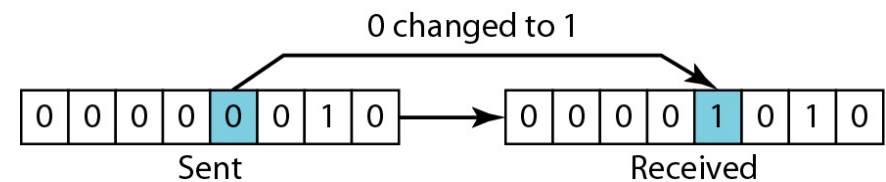
## Error Introduction: Single-Bit & Burst Error



**Single-Bit Error.** Only one bit of a given data unit is changed ( $0 \rightarrow 1$  or  $1 \rightarrow 0$ )

The least likely type of error in serial transmission

Single-bit error can happen in parallel transmission

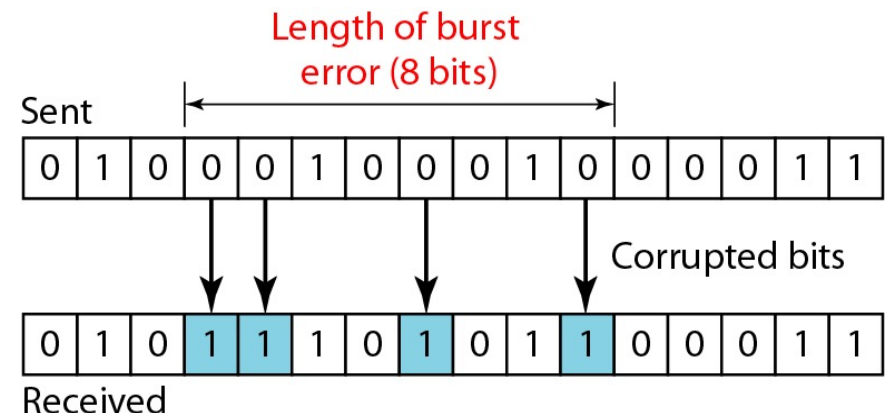


**Burst (Multiple-Bits) Error.** Two or more bits in the data unit have changed

Burst error does not necessarily mean that the errors occur in consecutive bits

Most likely to happen in a serial transmission

Number of bits affected depends on the data rate and duration of noise





# Error Detection and Correction

## Error Introduction: Redundancy & Error Control

**Error Control** uses the concept of redundancy, which means adding extra (redundant) bits for detecting errors at the destination.

### Types of Error Control - Detection and Correction:

- **Detection**: simple. Error in the message? yes or no;
- **Correction**: need to know the exact number of bits that are corrupted, and their location in the message.

### Methods of Error Correction - Forward and Backward:

- **Forward**: Coding with redundancy (Block, Convolution, Concatenated, ...) In Convolution coding, code Word depends on k-bit message block and also on 'm' previous message blocks.
- **Backward**: Retransmission (resending) if error detected.

### The four main error correction codes are

- Hamming Codes
- Binary Convolution Code
- Reed Solomon Code
- Low-Density Parity-Check Code

# Error Detection and Correction

## Error Introduction: Modular Arithmetic

In modulo-N arithmetic, we use only the integers in the range 0 to N-1, inclusive. Examples, for binary modular arithmetic:

**Adding:**

$$0 + 0 = 0 \quad 0 + 1 = 1 \quad 1 + 0 = 1 \quad 1 + 1 = 0$$

**Subtracting:**

$$0 - 0 = 0 \quad 0 - 1 = 1 \quad 1 - 0 = 1 \quad 1 - 1 = 0$$

**XORing** of two single bits or two words:

$$0 \oplus 0 = 0$$

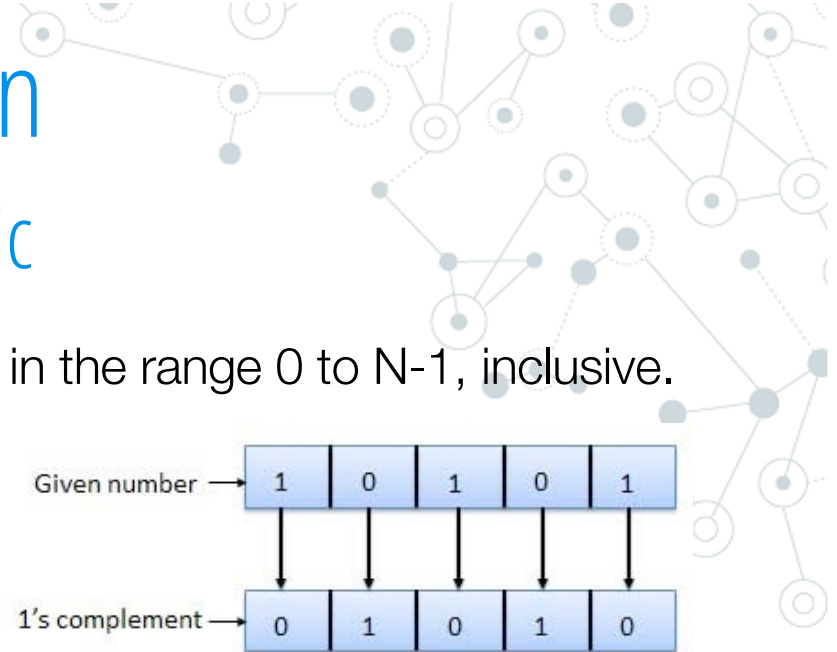
$$1 \oplus 1 = 0$$

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

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

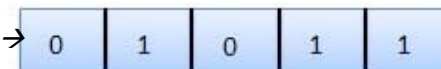
b. Two bits are different, the result is 1.



Add 1 +

1

2's complement →



	1	0	1	1	0
$\oplus$	1	1	1	0	0
<hr/>					
	0	1	0	1	0

c. Result of XORing two patterns

# Error Detection and Correction

## Error Detection and Correction - Block Coding

Divide the message into blocks, each of  $k$  bits, called **datawords**.

Add  $r$  redundant bits to each block to make the length  $n = k + r$ .

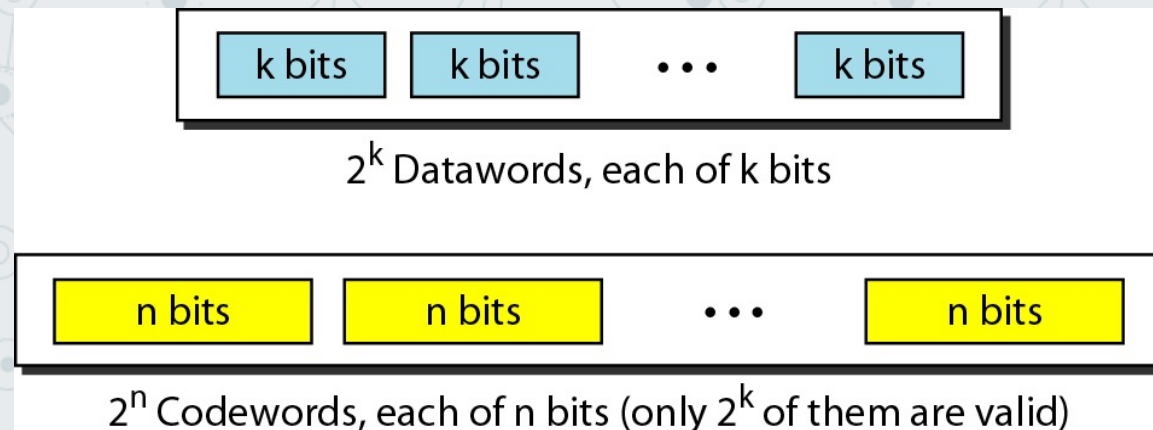
The resulting  $n$ -bit blocks are called **codewords**.

Select valid codewords.

Example: 4B/5B block coding

$k = 4$  and  $n = 5$ .

$2^k = 16$  datawords and  $2^n = 32$  codewords.



# Error Detection and Correction

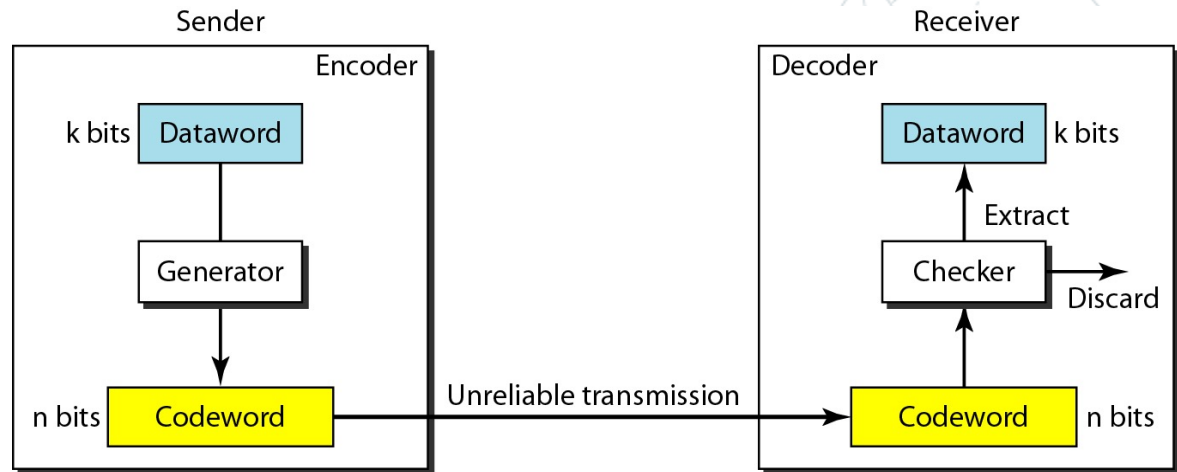
## Block Coding: Error Detection

$$n = k + r$$

**Example 1:**

Assume that  $k = 2$ ,  $r = 1$  and  $n = 3$ .

Datawords	Codewords
00	000
01	011
10	101
11	110



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

If sender encodes the dataword 01 as 011 and sends it to the receiver. Have 3 cases:

1. The receiver receives 011 which is a valid codeword. The receiver **extracts** the dataword 01 from it.
2. The codeword is corrupted during transmission, and **111** is received. This is not a valid codeword and is **discarded** (and possibly sends a request to retransmit the message).
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**.

# Error Detection and Correction

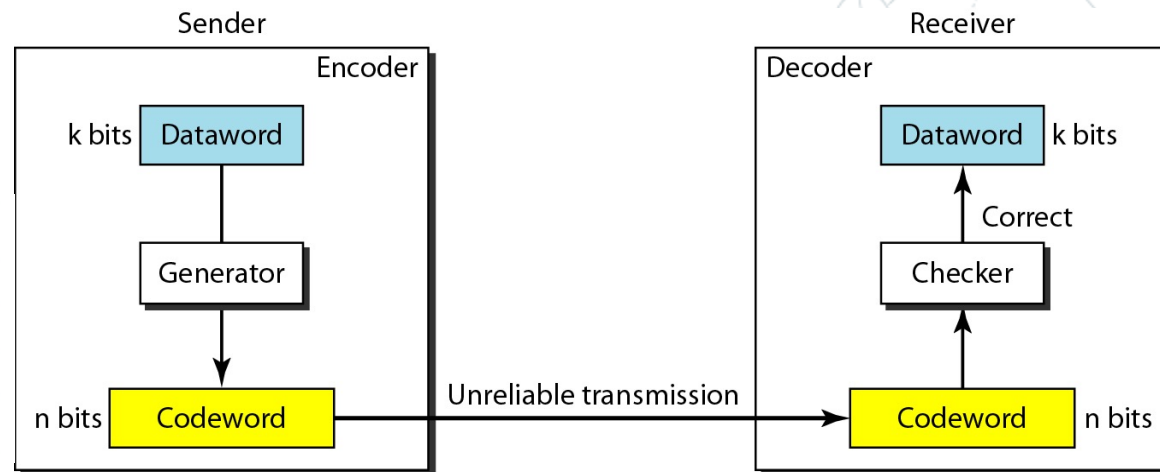
## Block Coding: Error Correction

$$n = k + r$$

### Example 2:

Assume that  $k = 2$ ,  $r = 3$  and  $n = 5$ .

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110



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

If sender encodes the dataword 01 as 01011 and sends it to the receiver. The codeword is corrupted during transmission, and 01001 is received.

Receiver detected then received codeword is not in the table → error detected. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy for correcting dataword:

1. Receiver comparing the received codeword with the 1 codeword in the table (01001 versus 00000), decides that the first codeword is not suitable, because there are two different bits (same for 3 and 4 rows in the table).

2. The original codeword must be 2 in the table because this is the only one that differs from the received codeword by 1 bit.



# Error Detection and Correction

## Block Coding: Hamming Distance

In information theory, the **Hamming distance** between two strings of equal length is the number of positions at which the corresponding symbols are different. Examples:

"Karolin-1" and "Kerstin-1" is 3;

2173896 and 2233796 is 3;

10101 and 11110 is 3;

000 and 111 is 3.

The **Hamming distance between two binary words  $n$  and  $m$**  is the number of differences between corresponding bits.

$$d(n,m) = n \text{ XOR } m$$

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

**Example** for Table 1 of  $C(2,3)$   $d_{\min} = 2$

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

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

This code guarantees detection of only a single error.

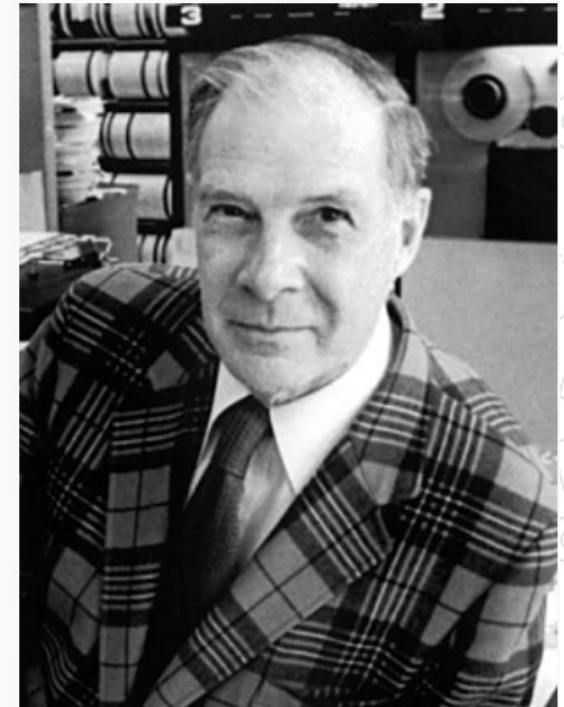
Datawords	Codewords
00	000
01	011
10	101
11	110

**Example** for Table 2  $C(2,5)$  has  $d_{\min} = 3$

This code can detect up to 2 errors or correct 1 error

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Richard Hamming



**Born** February 11, 1915  
Chicago, Illinois, U.S.

**Died** January 7, 1998 (aged 82)  
Monterey, California, U.S.



# Error Detection and Correction

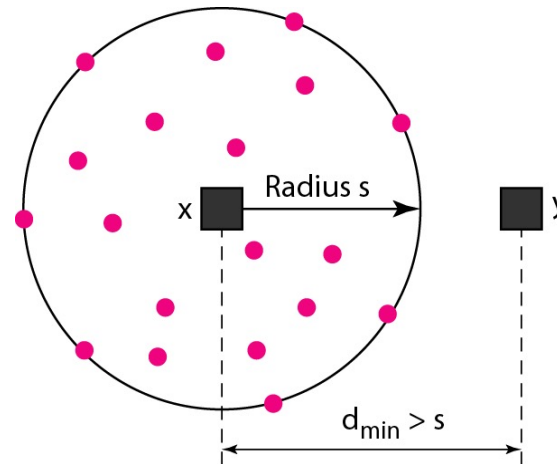
## Block Coding: Hamming Distance (Cont.1)

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

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

**Optimal** error correction codes need to have an odd minimum distance  
 $d_{\min} = (3, 5, 7, \dots)$

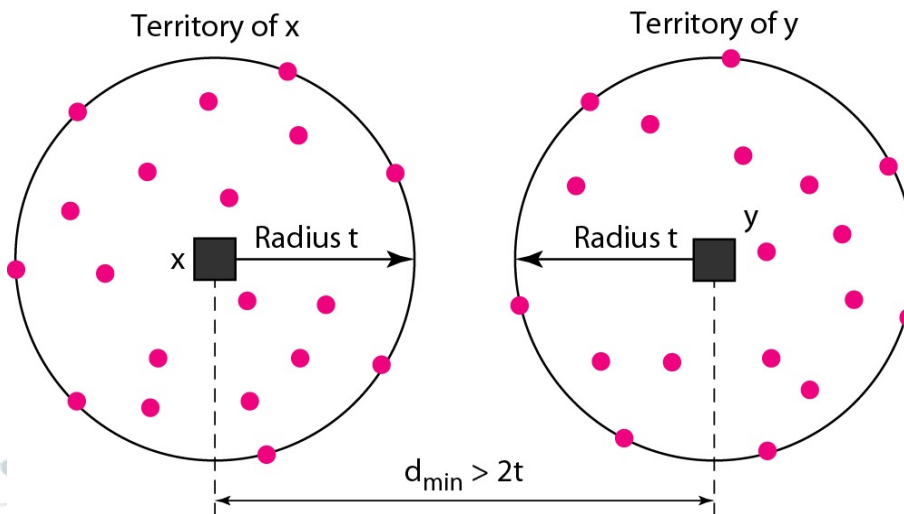
Geometric concept for finding  $d_{\min}$  in error **detection**



Legend

- Any valid codeword
- Any corrupted codeword with 0 to  $s$  errors

Geometric concept for finding  $d_{\min}$  in error **correction**



Legend

- Any valid codeword
- Any corrupted codeword with 1 to  $t$  errors

# Error Detection and Correction

## Linear Block Coding: Definition

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.

Examples for Tables C(2,3) and C(2,5) belong to the class of linear block codes:

1. The scheme C(2,3) 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 codeword:  
 $011 \text{ XOR } 101 = 110$ .

2. The scheme C(2,5) is also a linear block code. We can create all four codewords by XORing two other codewords.

<i>Datawords</i>	<i>Codewords</i>
00	000
01	011
10	101
11	110

<i>Dataword</i>	<i>Codeword</i>
00	00000
01	01011
10	10101
11	11110

# Error Detection and Correction

## Linear Block Coding 1: Simple Parity-Check Code (with Detection)

A codeword consists of  $n$  bits of which  $k$  are data bits and  $r$  are check bits  $n=k+r$ .

A **Simple Parity-check Code** is a **single-bit error-detecting code** in which  $n=k+1$  with  $d_{\min} = 2$ . The extra bit called parity bit is selected to make the number of 1s in the codeword even (чётным).

**Example** for Simple Parity Code  $k=4$ ,  $r=1$ ,  $n=5$

The addition of the 4 bits of the data word is the parity bit (modulo 2 )

If the number of 1s are even or 0, the result is 0;

if the number of 1s are odd the result is 1.

<i>Datawords</i>	<i>Codewords</i>	<i>Datawords</i>	<i>Codewords</i>
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

# Error Detection and Correction

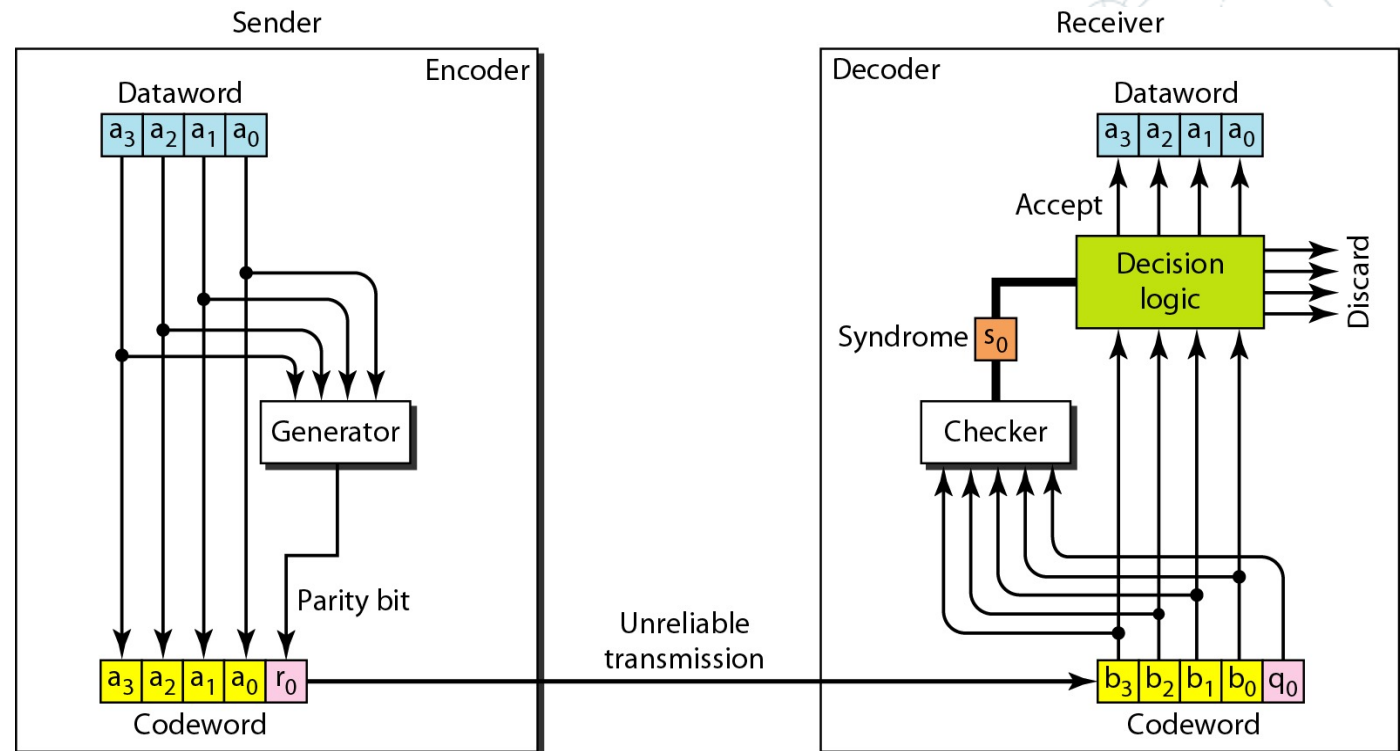
## Linear Block Coding 1: Enc./Decoder for Sim. Parity-Check Code

Checker - the result of mod2 addition all 5 bits:  
**syndrome**.

$$(1+0+1+1+1)\text{mod}2=0$$

If the syndrome is 0, there is no error in the codeword, the data portion of the codeword (dataword) is accepted.

Assume the sender sends 1011, codeword is 10111.



See next different cases:

- Syndrome 0, no error and dataword 1011 is created.
  - One bit  $a_1$  changes, Syndrome 1, No dataword is created.
  - One bit  $r_0$  changes, Syndrome 1, No dataword is created.
  - Two error with  $r_0$  and  $a_3$ . Codeword is 00110, Syndrome is 0, **wrong** dataword 0011 is accepted. So parity check detector **can not detect** even numbers of errors!
  - Three bits  $a_1, a_2$  and  $a_3$  are changed. Codeword is 01011, Syndrome is 1, No dataword is created.
- Parity check guaranteed to detect single error, can also find odd number of errors (3,5,...), no have correction.

# Error Detection and Correction

## Linear Block Coding 2: Two-dimensional Parity-check Code (with Correction)

Data blocks write as two-dimension (n-m) massive, example (7-4), and after adding row and column parities bits. Result massive size is (8-5) send to line.

Two-dimensional Parity-check guaranteed to detect one, two and three error, and can also CORRECT one single errors.

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

a. Design of row and column parities

1	1	0	0	1	1	1	1	1
1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1

b. One error affects two parities

1	1	0	0	1	1	1	1	1
1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1

d. Three errors affect four parities

1	1	0	0	1	1	1	1	1
1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1

c. Two errors affect two parities

1	1	0	0	1	1	1	1	1
1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1

e. Four errors cannot be detected

# Error Detection and Correction

## Linear Block Coding 3: Hamming Code with Correction

$$n=k+m \ (7=4+3)$$

We have Hamming code  $d_{\min}=3$  (2 bit error detection and single bit error correction)

Dataword (k bits), codeword (n bits), choose  $m \geq 3$  as the relationship between m and n in these codes is  $n = 2^m - 1$  and  $k=n-m$ .

If  $m=3$ , then  $n=7$  and  $k=4$ , thus  $C(7,4)$  with  $d_{\min} = 3$

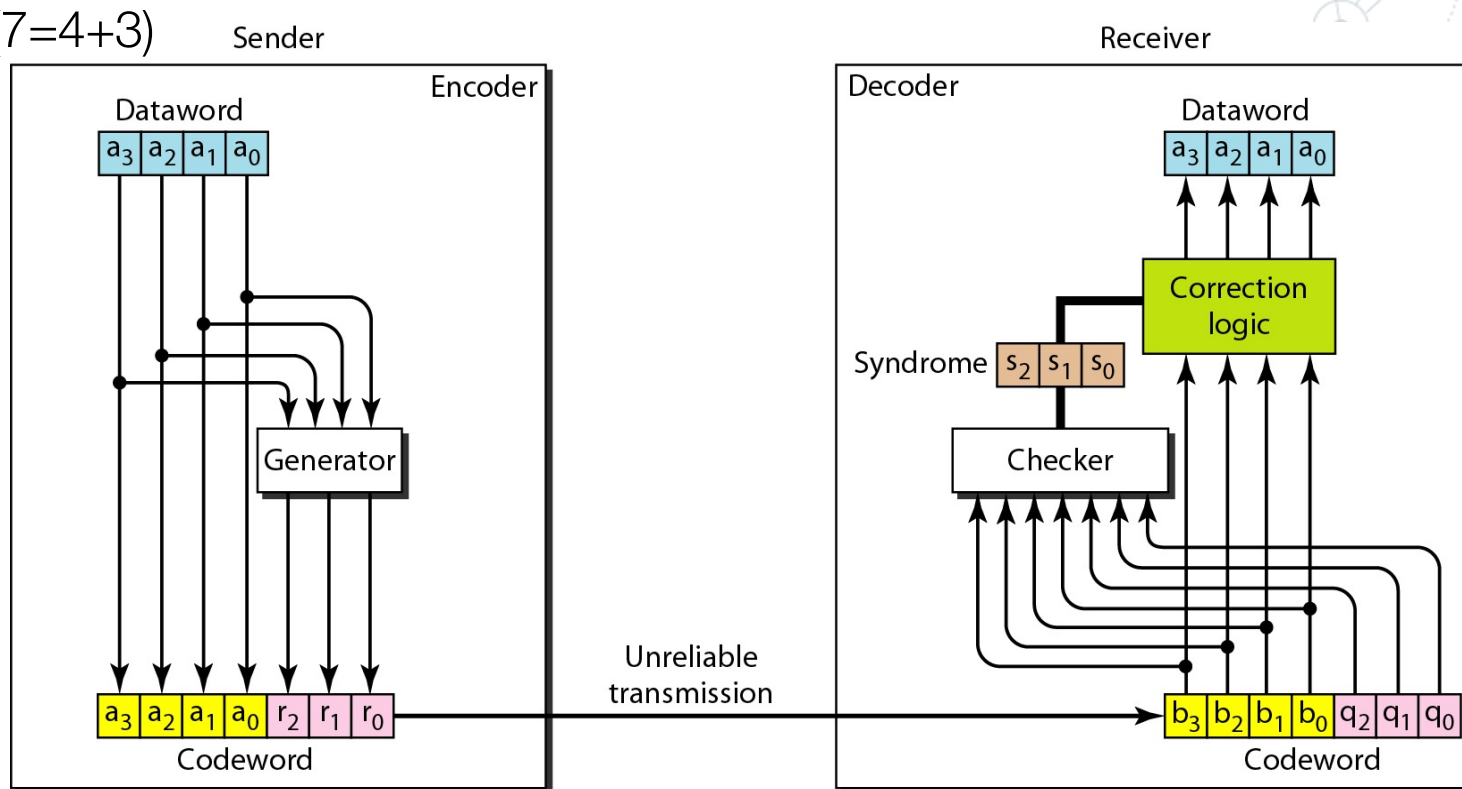
<i>Datawords</i>	<i>Codewords</i>	<i>Datawords</i>	<i>Codewords</i>
0000	0000 <b>000</b>	1000	1000 <b>110</b>
0001	0001 <b>101</b>	1001	1001 <b>011</b>
0010	0010 <b>111</b>	1010	1010 <b>001</b>
0011	0011 <b>010</b>	1011	1011 <b>100</b>
0100	0100 <b>011</b>	1100	1100 <b>101</b>
0101	0101 <b>110</b>	1101	1101 <b>000</b>
0110	0110 <b>100</b>	1110	1110 <b>010</b>
0111	0111 <b>001</b>	1111	1111 <b>111</b>



# Error Detection and Correction

## Linear Block Coding 3: Enc./Decoder Hamming Code with Correct

$$n=k+m \ (7=4+3)$$



Calculating the parity bits at the transmitter:

$$r_0 = (a_2 + a_1 + a_0) \text{ modulo-2}$$

$$r_1 = (a_3 + a_2 + a_1) \text{ modulo-2}$$

$$r_2 = (a_1 + a_0 + a_3) \text{ modulo-2}$$

Calculating the syndrome at the receiver:

$$s_0 = (b_2 + b_1 + b_0 + q_0) \text{ modulo-2}$$

$$s_1 = (b_3 + b_2 + b_1 + q_1) \text{ modulo-2}$$

$$s_2 = (b_1 + b_0 + b_3 + q_2) \text{ modulo-2}$$

Correction Logic block make Codeword correct after one error.

# Error Detection and Correction

## Linear Block Coding 4: Burst Error Correction with Mod.HamCode

$$n=k+m \ (7=4+3)$$

Burst errors are very common in wireless environments.

One way to counter burst errors, is to use modify Hamming Code with Correction:

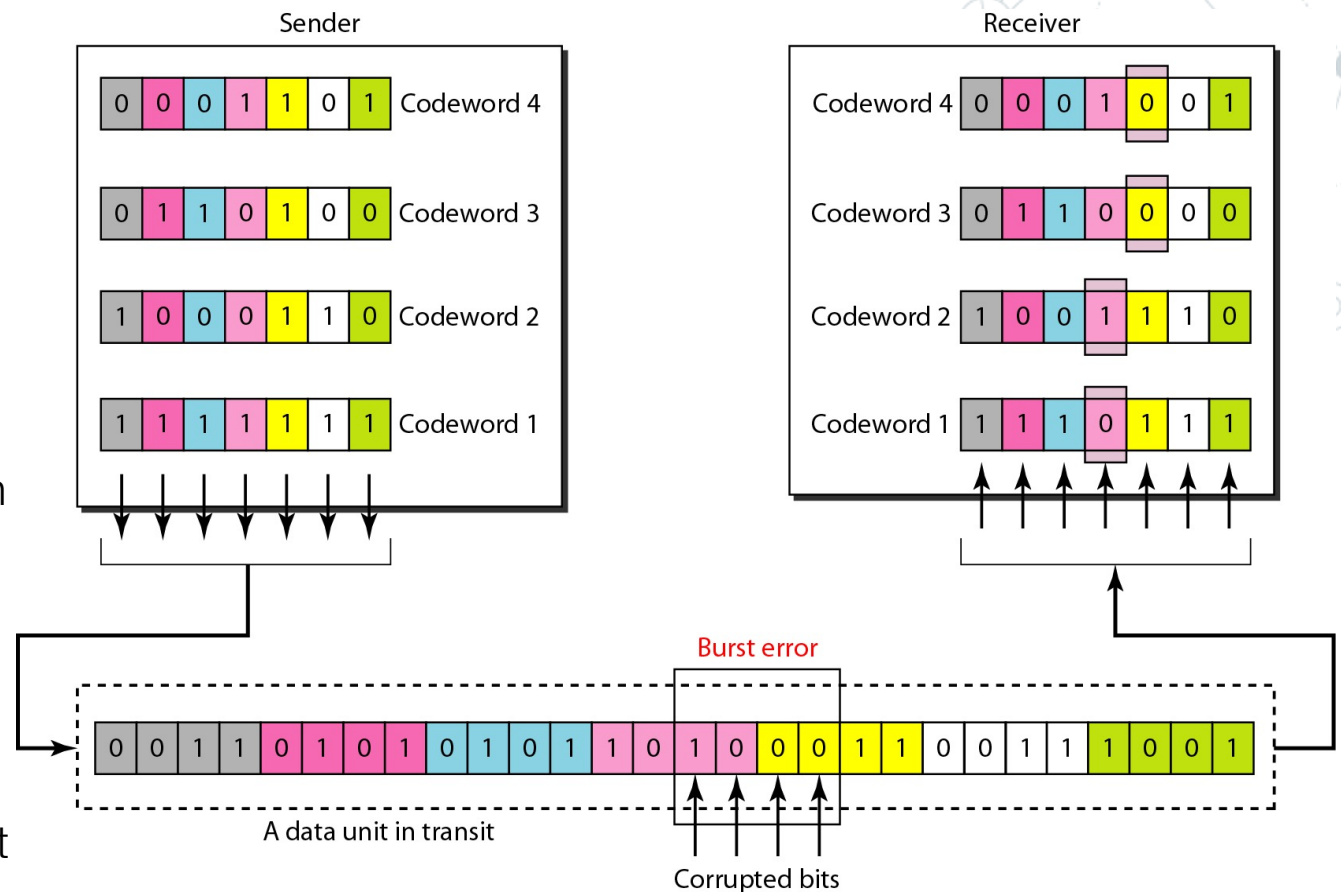
1. Break up a transmission into shorter words and create a block (one word per **row**).

2. Then make a parity check bit or bits per word.

3. The words are then sent column by **column**.

4. When a **burst error occurs**, it will **affect only 1 bit** in several words (see picture).

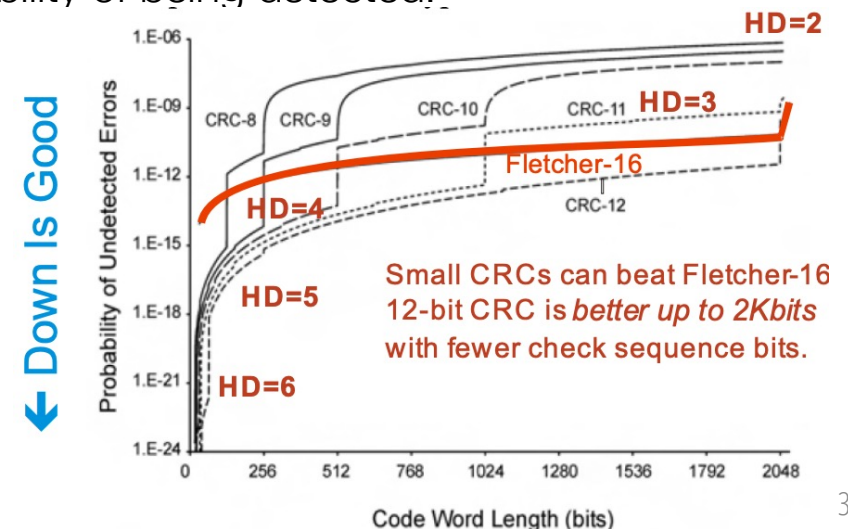
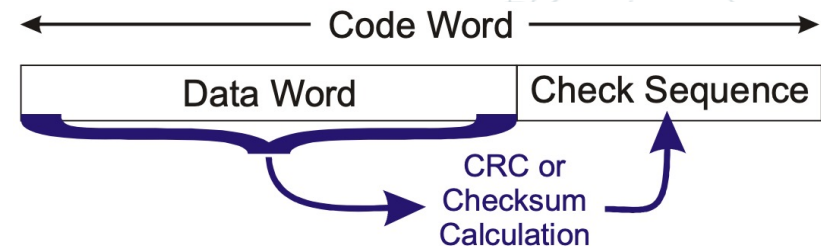
5. Only 1 bit because the transmission is read back into the block format and each word is checked individually.



# Error Detection and Correction

## Error Detection: CRC/Checksum Introduction

- CRC - Check Redundance Code
- Potential CRC/Checksum Usage Scenarios
  - Network packet integrity check
  - Image integrity check for software update
  - Boot-up integrity check of program image – ex., flash memory data integrity check
  - FPGA (Field-Programmable Gate Array –ПЛИМ/ПЛИС) configuration integrity check
  - Configuration integrity check – ex., operating parameters in EEPROM
  - RAM value integrity check
- Ideally, every bit in the data word affects many check sequence bits.
- Ideally, bit errors in the code word have high probability of being detected.
- A good CRC is almost always much better than a good Checksum at about the same cost.
- There are many checksum algorithms
  - 1's complement Checksum (HammingDist=2)
  - Adler Checksum
  - Fletcher Checksum (HD=3)
  - ATN-32 Checksum



# Error Detection and Correction

## Error Detection: Checksum Introduction

- Simple Checksum used 1's **complement arithmetic** (complement for 10101 is 01010)
  - We can represent unsigned numbers between 0 and  $2^n - 1$  using only  $n$  bits
  - If the number has more than  $n$  bits, the extra leftmost bits need to be added to the  $n$  rightmost bits (wrapping)
  - A negative number can be represented by inverting all bits. It is the same as subtracting the number from  $2^n - 1$ .
- There are many Checksum **algorithms**:
  - Simple Checksum-1. Used 1 parity bit. Disadvantage – no detect even number errors. Theoretical  $P_{\text{NoErrorDetection}} = 1/2 = 0.5$  per Byte. Used for byte transfer.
  - Simple Checksum-4. Used as **Example**.  
 $P_{\text{NoErrorDetection}} = 1/16 = 0.06$  per Byte.
  - Simple Checksum-8. **Used to microcontrollers**.  
Theoretical  $P_{\text{NoErrorDetection}} = 1/256 = 0.004$  per Byte.
  - Internet Checksum-16. **Used to IPv4, ICMP, TCP, UDP as Header Checksum**. RFC-1071 Computing the Internet Checksum. Not used to Ethernet.  
Theoretical  $P_{\text{NoErrorDetection}} = 1/2^{16} = 1/65536 = 0.00002$  per Byte.  
This is a lot for frame: 0.02 per 1.5KiB EtherMTU; 0.06 per 4KiB FDDI MTU, so on Ethernet used CRC.
- **Checksum cannot detect burst errors!** IT tendency is to **replace** the checksum with a CRC, because as stronger in error-checking capability.

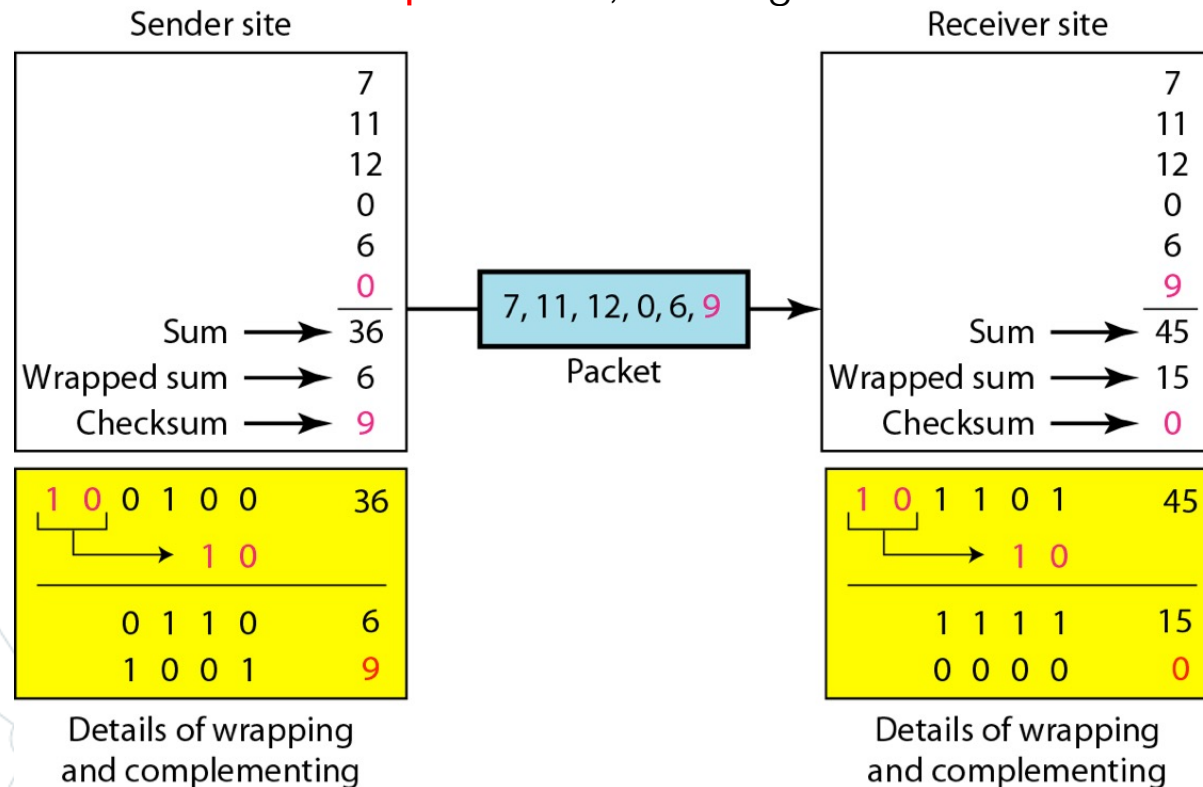
# Error Detection and Correction

## Error Detection: Example Checksum-4 (4 bit)

**Sender site.** The sender divided message into 4 bits; **initializes** the checksum to 0 and adds all data items and the checksum ( $7+11+12+0+6+0=36$ ) (**used 1's complement arithmetic**).

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.



### Receiver site.

Receiver for test data repeats all operations with data and the checksum from received packet:

Stream divided into 4 bits

Sum:  $7+11+12+0+6+9=45$

Wrapped sum = 15

Complement = 0

➔ Constated **No Error**



# Error Detection and Correction

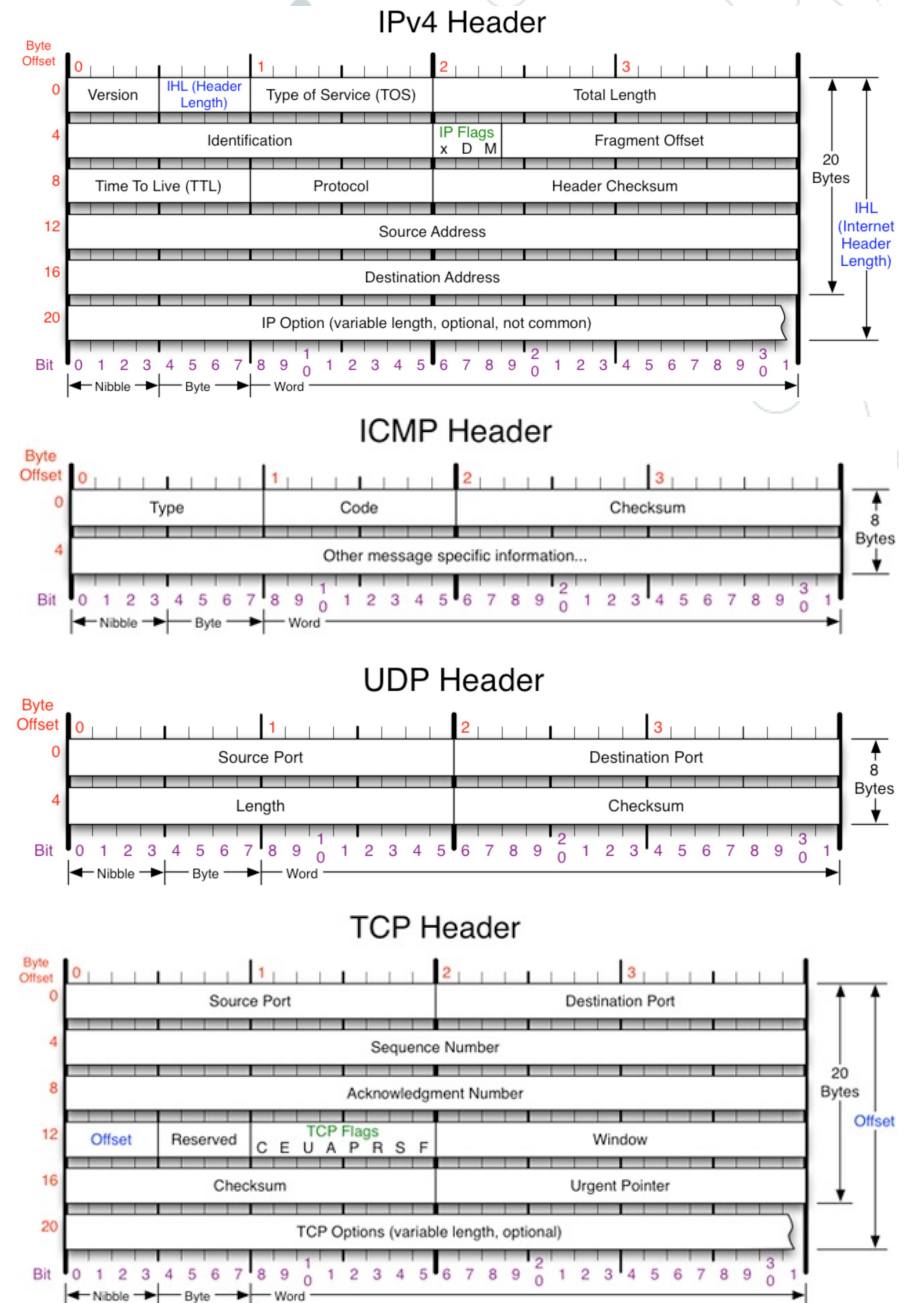
## Error Detection: Internet Checksum

### 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.

### Receiver site:

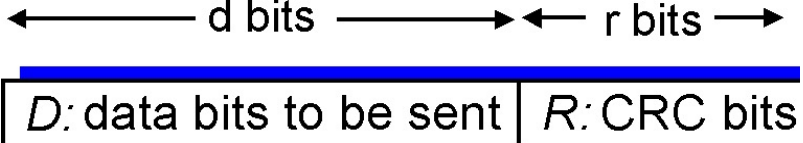
1. The message (including checksum) is divided into 16-bit words.
2. All words are added using one's complement addition.
3. The sum is complemented and becomes the new checksum.
4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.





# Error Detection and Correction

## Error Detection: CRC - Cyclic Redundancy Check (since 1957)

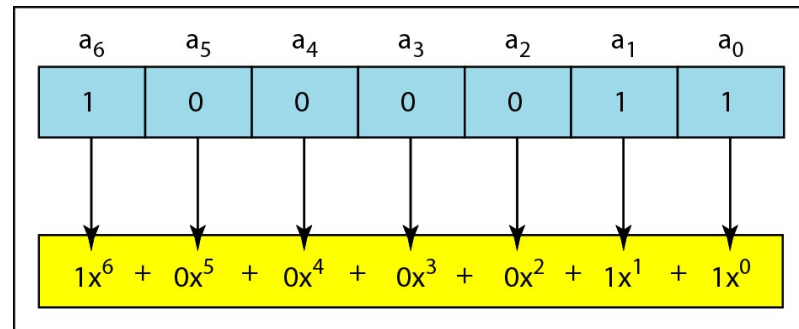
- There are many coding **algorithms**: CRC-12, CRC-CCITT, CRC-16, CRC-32, with differ. **polynomials** with specifically characteristics.
- CRC more powerful error-detection coding then Checksum.
- View data bits, **D**, as a binary number.
- Choose  $r+1$  bit pattern (polynom-generator), **G** 
- Goal: choose  $r$  CRC bits, **R**, such that
  - $\langle D, R \rangle$  exactly divisible by  $G$  (modulo 2)
  - receiver knows  $G$ , divides  $\langle D, R \rangle$  by  $G$ .  $R = \text{Reminder} \{ (D \cdot 2^r) \text{ XOR } G \}$
  - If non-zero Remainder  $R$ , then **error detected!**
- **CRC can detect all Burst errors less than  $r+1$  bits**
- CRC used in practice (**Ethernet, 802.11 WiFi, ATM**)
- In CRC error detection not 100% reliable! The probability of non-detection of errors for CRC is equal to  $(1/2)^r$ , where  $r$  is the degree of the generating polynomial. In CRC-32, for example, theoretical  
**P<sub>NonDetectionError</sub>**  $= (1/2)^r = (1/2)^{32} = 1/4,000,000,000 = 0.0000000003$  per DoubleWord -32bit

# Error Detection and Correction

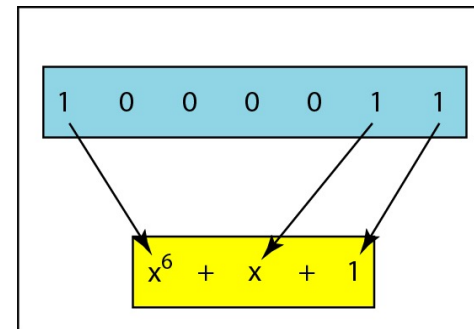
## Error Detection: CRC - Standard Polynomials

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 (multiplier)  $x + 1$ .
5. Degree of a polynomial: the highest power in the polynomial, example:  $0x43 = x^6 + x + 1 \rightarrow$  the degree '6'

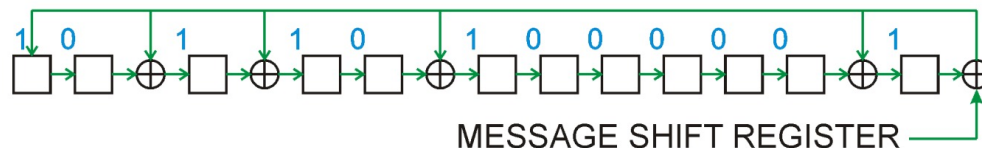


a. Binary pattern and polynomial



b. Short form

**POLYNOMIAL:** 1011 0100 0001 = 0xB41



c. Hardware View of CRC

**Task 1:**

Draw a generator for CRC8, CRC10

**Task 2:**

Read the article about  
Checksum/CRC Data Integrity

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

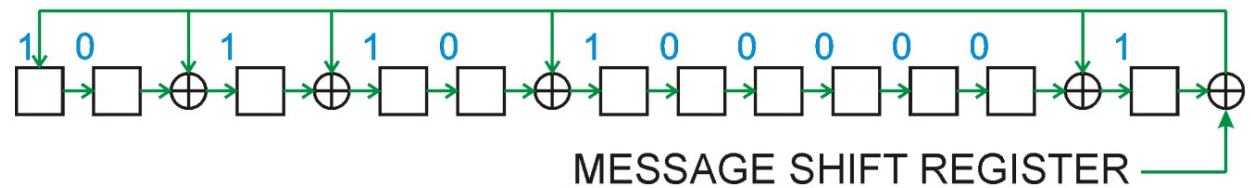
# Error Detection and Correction

## Error Detection: CRC - Standard Polynomials

Optimal Polynomials For Small CRCs, example:

1. Hamming Distance – HD=6.
2. CRC Size=12
3. Data Word=27
4. Polyn. Code=0xB41
5.  $x^{11} + x^9 + x^8 + x^6 + 1$

POLYNOMIAL: 1011 0100 0001 = 0xB41



Max length at HD Polynomial	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD=2	2048+ <u>0x5</u>	2048+ <u>0x9</u>	2048+ <u>0x12</u>	2048+ <u>0x21</u>	2048+ <u>0x48</u>	2048+ <u>0xA6</u>	2048+ <u>0x167</u>	2048+ <u>0x327</u>	2048+ <u>0x64D</u>	–	–	–	–	–
HD=3		11 <u>0x9</u>	26 <u>0x12</u>	57 <u>0x21</u>	120 <u>0x48</u>	247 <u>0xA6</u>	502 <u>0x167</u>	1013 <u>0x327</u>	2036 <u>0x64D</u>	2048 <u>0xB75</u>	–	–	–	–
HD=4			10 <u>0x15</u>	25 <u>0x2C</u>	56 <u>0x5B</u>	119 <u>0x97</u>	246 <u>0x14B</u>	501 <u>0x319</u>	1012 <u>0x583</u>	2035 <u>0xC07</u>	2048 <u>0x102A</u>	2048 <u>0x21E8</u>	2048 <u>0x4976</u>	2048 <u>0xBAAD</u>
HD=5						9 <u>0x9C</u>	13 <u>0x185</u>	21 <u>0x2B9</u>	26 <u>0x5D7</u>	53 <u>0x8F8</u>	none	113 <u>0x212D</u>	136 <u>0x6A8D</u>	241 <u>0xAC9A</u>
HD=6							8 <u>0x13C</u>	12 <u>0x28E</u>	22 <u>0x532</u>	27 <u>0xB41</u>	52 <u>0x1909</u>	57 <u>0x372B</u>	114 <u>0x573A</u>	135 <u>0xC86C</u>
HD=7									12 <u>0x571</u>	none	12 <u>0x12A5</u>	13 <u>0x28A9</u>	16 <u>0x5BD5</u>	19 <u>0x968B</u>
HD=8										11 <u>0xA4F</u>	11 <u>0x10B7</u>	11 <u>0x2371</u>	12 <u>0x630B</u>	15 <u>0x8FDB</u>