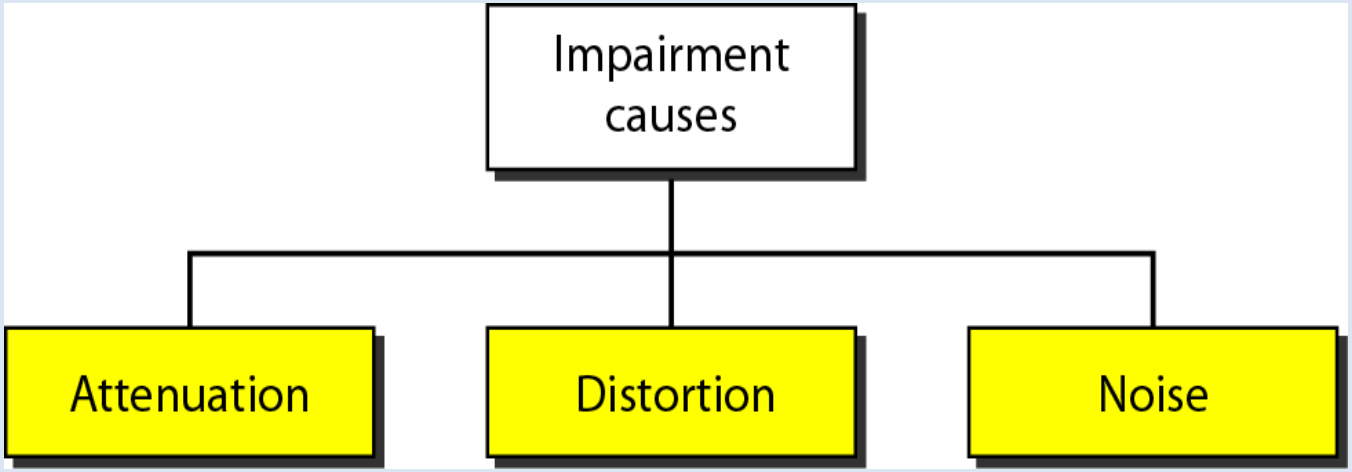
Data Communication
Chapter 6
Impairments, Error Handling and Compression Techniques

Himal Acharya

Course Instructor

Transmission Impairments

- Signal travel through transmission media, which are not perfect.
- The imperfection causes signal impairment.
- This means that the signal at the beginning of the medium is not the same as the signal at the end of the medium. What is sent not what is received
- Three causes of impairment are
  a. Attenuation
  b. Distortion
  c. Noise



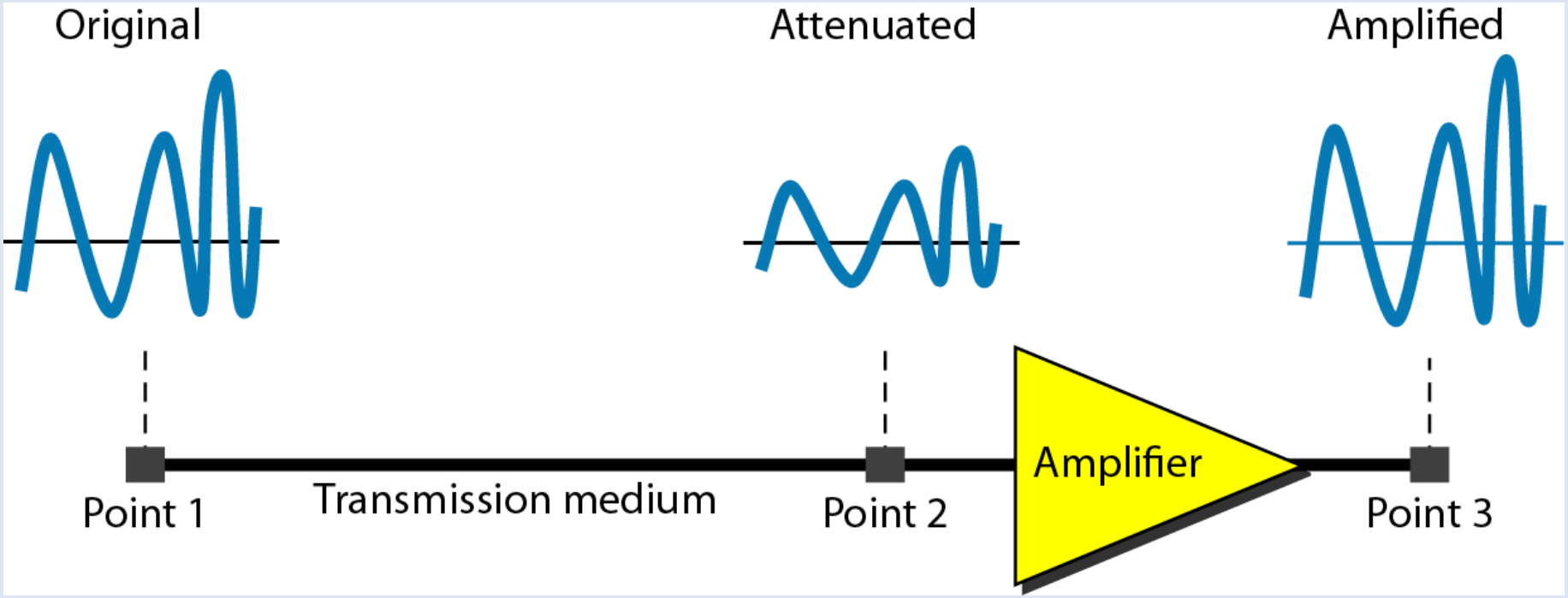
Attenuation

- Attenuation means a loss of energy.
- When a signal, simple or composite, travels through a medium, it loses some of its energy in overcoming the resistance of the medium. That is why a wire carrying electric signals get warm, if not hot, after a while. Some of the electrical energy is converted to heat.
- Amplifiers are used to compensate for this loss of energy by amplifying the signal.
- Attenuation in dB : To show the loss or gain of energy the unit “decibel” is used.

$$dB = 10\ log_{10}\ P_2/P_1$$

P_1 - input signal , P_2- output signal

Attenuation



- Suppose a signal travels through a transmission medium and its power is reduced to one-half. This means that P_2 is $(1/2)P_1$. In this case, the attenuation (loss of power) can be calculated as

$$10 \log_{10} \frac{P_2}{P_1} = 10 \log_{10} \frac{0.5 P_1}{P_1} = 10 \log_{10} 0.5 = 10(-0.3) = -3 \text{ dB}$$

A loss of 3 dB (–3 dB) is equivalent to losing one-half the power.

- A signal travels through an amplifier, and its power is increased 10 times. This means that $P_2 = 10P_1$. In this case, the amplification (gain of power) can be calculated as

$$10 \log_{10} \frac{P_2}{P_1} = 10 \log_{10} \frac{10 P_1}{P_1}$$

$$= 10 \log_{10} 10 = 10(1) = 10 \text{ dB}$$

The loss in a cable is usually defined in decibels per kilometer (dB/km). If the signal at the beginning of a cable with –0.3 dB/km has a power of 2 mW, what is the power of the signal at 5 km?

Solution

The loss in the cable in decibels is $5 \times (-0.3) = -1.5$ dB. We can calculate the power as

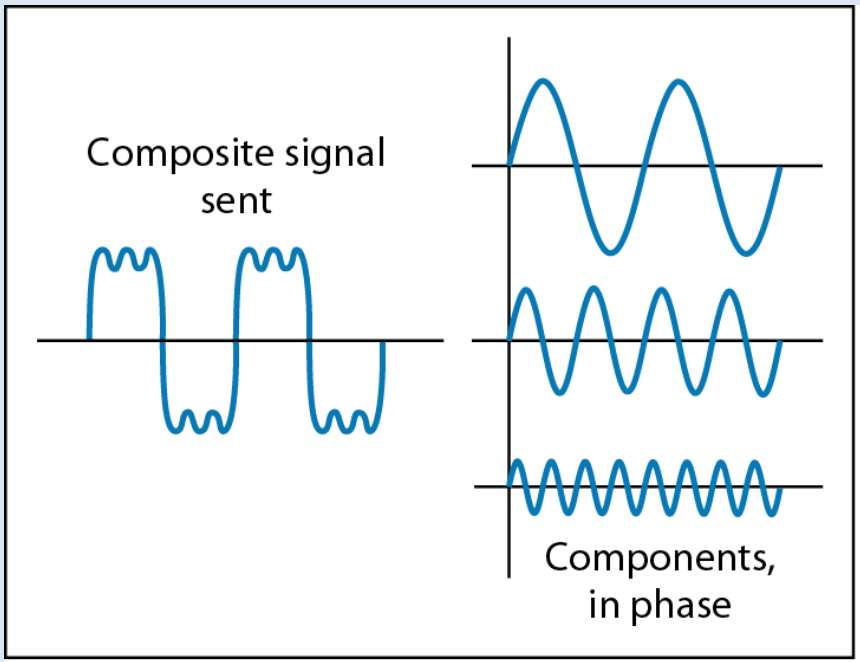
$$\text{dB} = 10 \log_{10} (P_2 / P_1) = -1.5 \quad \longrightarrow \quad (P_2 / P_1) = 10^{-0.15} = 0.71$$

$$P_2 = 0.71 P_1 = 0.7 \times 2 \text{ mW} = 1.4 \text{ mW}$$

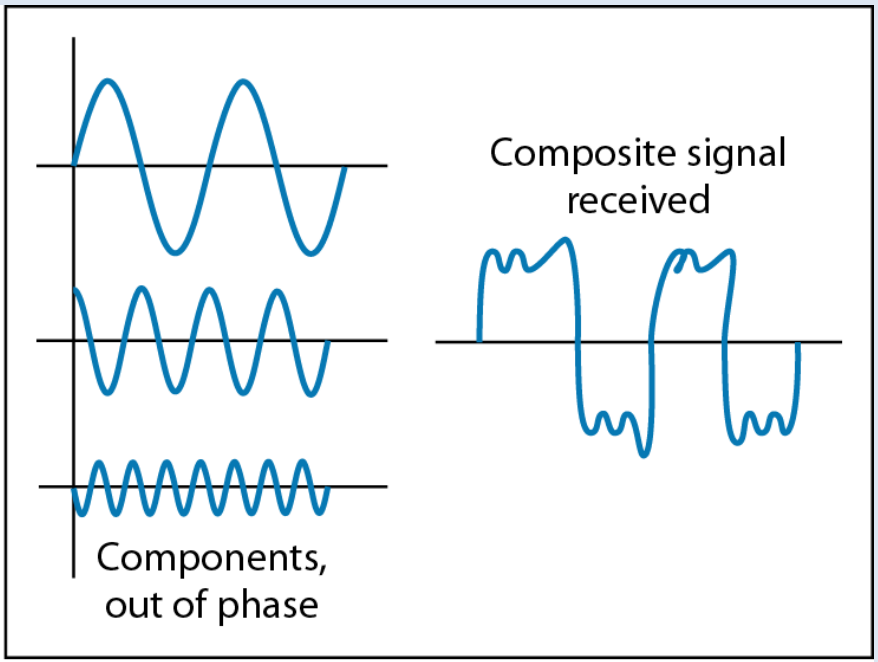
Distortion

- Distortion means that the signal changes its form or shape.
- Distortion can occur in a composite signal made of different frequencies.
- Each signal component has its own propagation speed through a medium and, therefore, its own delay in arriving at the final destination.
- Differences in delay may create a difference in phase if the delay is not exactly the same as the period duration.
- In other words, signal components at the receiver have phases different from what they had at the sender. The shape of the composite signal is therefore not the same.
- Equalization techniques can be implemented to minimize delay distortion.

Distortion



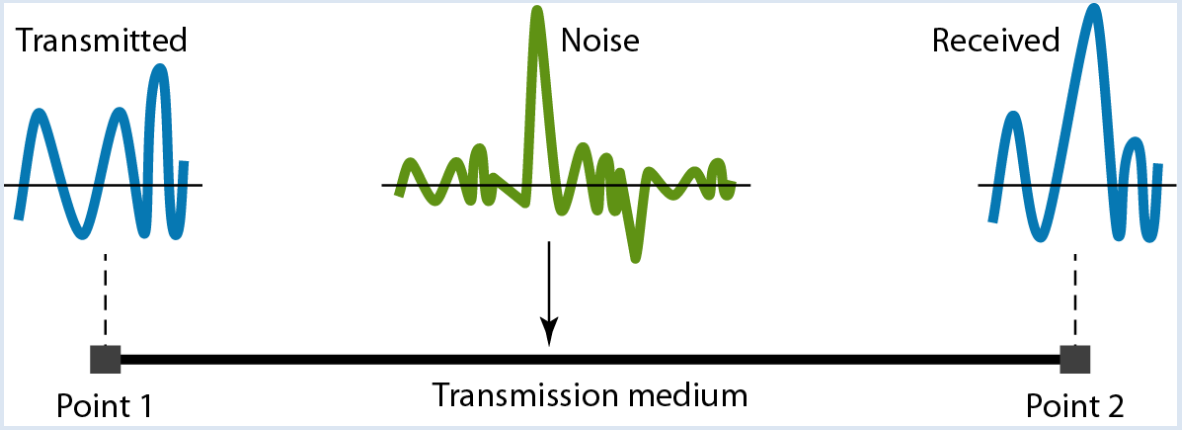
At the sender



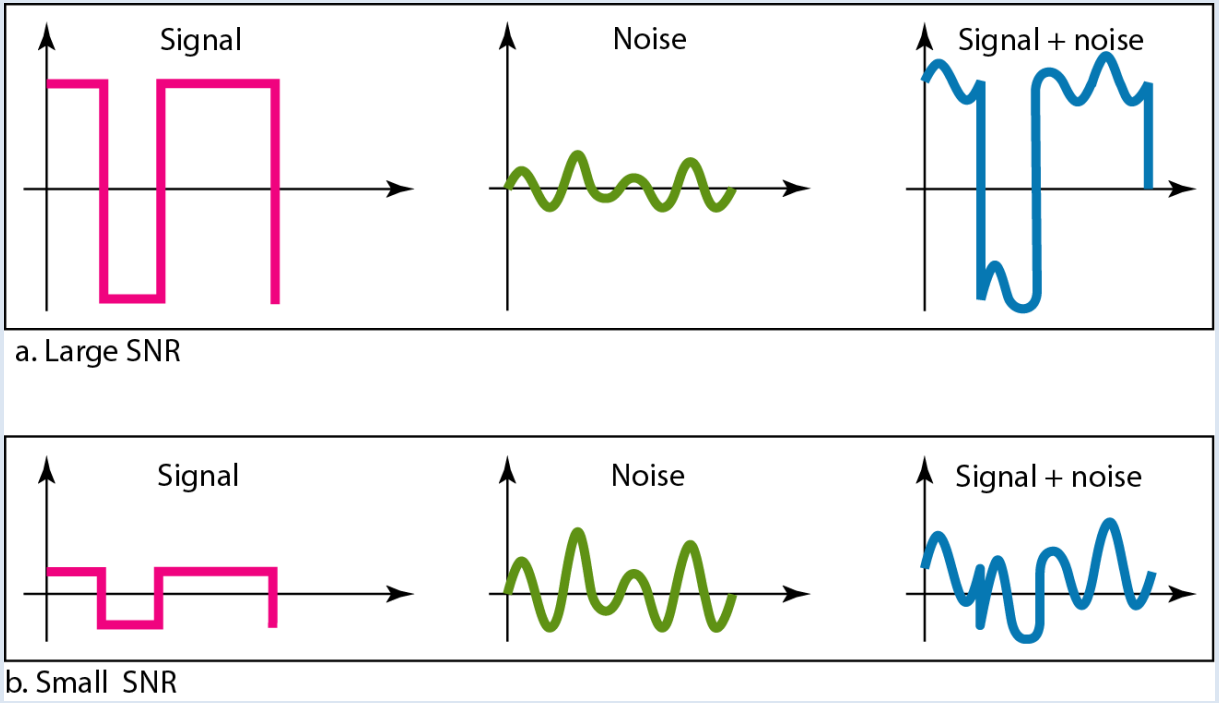
At the receiver

Noise

- Noise may be defined as an unwanted form of energy which tend to interfere with the proper reception and reproduction of transmitted signals.
- Noise is the major limiting factor in communication system performance
- Noise may be divided into four categories
  - Thermal Noise
  - Intermodulation Noise
  - Crosstalk
  - Impulse Noise



SNR – Two cases : A high SNR and a low SNR



A high SNR means the signal is less corrupted by noise; a low SNR means the signal is more corrupted by noise

Thermal Noise

- Thermal noise is the random motion of electrons in a wire, which creates an extra signal not originally sent by the transmitter.
- Also called white noise or Johnson noise
- When the temperature increases, the movement of free electrons will increase and the current flows through the conductor.
- Current flows due to the free electrons will create noise voltage, n(t).
- Noise voltage, n(t) is influenced by the temperature and therefore it is called thermal noise.
- Thermal noise cannot be eliminated and places an upper bound on communications system performance
- Thermal noise power is given by

$$P_n = kTB$$

k- Boltzman's constant = 1.38 x 10^{-23} Joule/ K

T- absolute temperature, B- bandwidth

- Thermal noise can be reduced by reducing temperature or bandwidth.

Intermodulation noise

- Undesired signals at the frequency that is multiples of sum or difference of the two original input frequencies.
  - For example, the mixing of signals at frequencies f_1 and f_2 are traveling through the same medium simultaneously. It might produce energy at the frequency components f_1+f_2 and $f_1 - f_2$
- Caused by the non-linearity of the system (transmitter, receiver and transmission medium)
- Intermodulation is rarely desirable in radio or audio processing as it creates unwanted spurious emissions, often in the form of sidebands which increases the occupied bandwidth leading to adjacent channel interference and reduce audio clarity.

Crosstalk

- Crosstalk is the effect of one wire on the other. One wire acts as a sending antenna and the other as the receiving antenna.
- Crosstalk has been experienced by anyone who, while using the telephone, has been able to hear another conversation; it is unwanted coupling between signal paths
- E.g. electrical coupling between near by twisted pair wires
- Coaxial cables are more immune to crosstalk compared to twisted pairs.

Impulse Noise

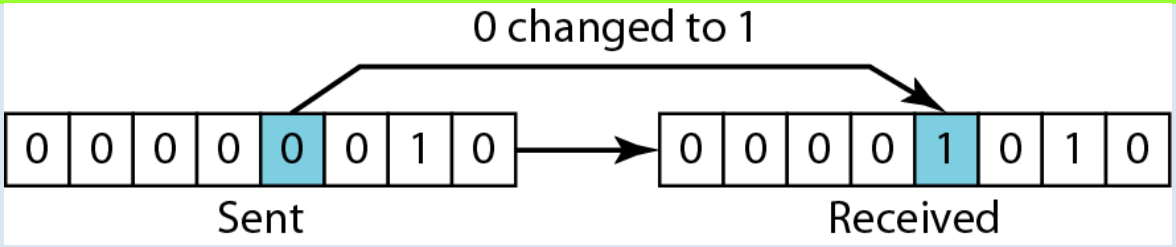
- Impulse noise is a spike (a signal with high energy in a very short time) that comes from power lines, lightning, and so on.
- Unlike previous types of noise, this one is noncontinuous – irregular pulses or spikes for short duration and high amplitude
- Minor annoyance for analog data but primary source of error in digital data communication.

Error

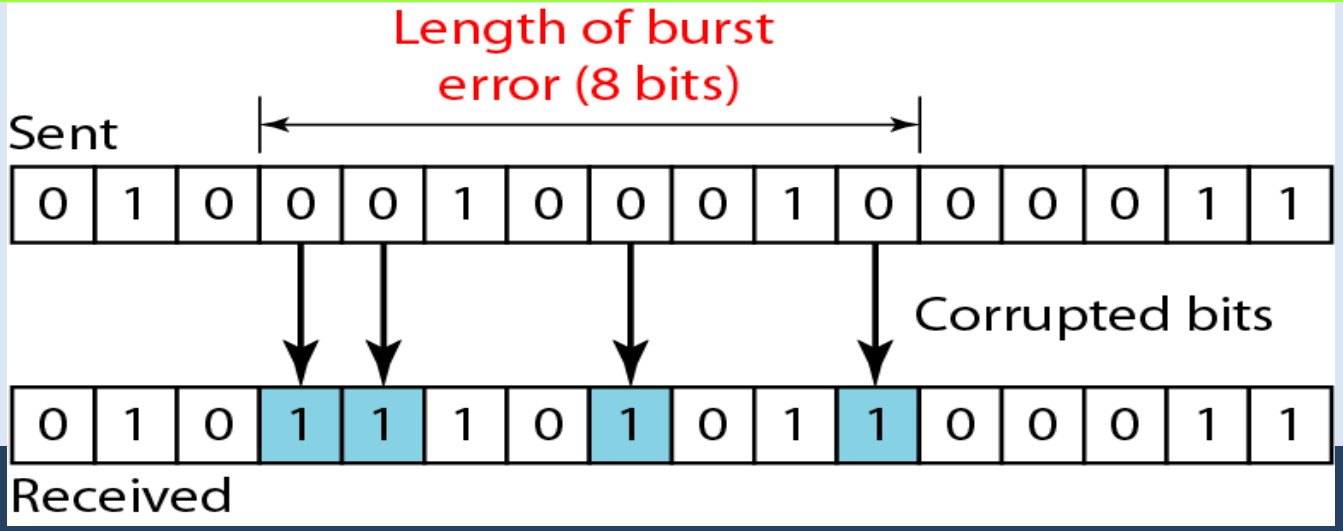
Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal.

Types of Error

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



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



Redundancy

- Adding some extra (redundant) bits with data – Added by sender and removed by the receiver
- Added redundancy in the original data stream enables error detection and correction
  - Error-detecting codes
  - Error- correcting codes

1. With error correction the error is detected and corrected at the receiver. A communication system which implements this form of channel coding is known as a *Forward-Error-Correction* (*FEC*) error-control system.

2. With error detection the error is detected and the transmitter is required retransmit the message. This requires a reliable return feedback channel from receiver to transmitter. The simplest implementation is a stop-and-wait strategy whereby the receiver acknowledges (ACK) each correct reception and negatively acknowledges (NACK) each incorrect reception. The transmitter then retransmits any message that was incorrectly received A communication system which implements this form of channel coding is known as a *Automatic-Repeat-Request* (*ARQ*) error-control system.

Coding

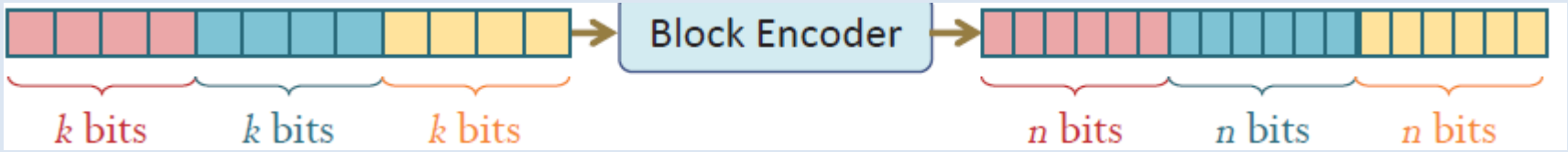
- Redundancy is achieved through various coding schemes.
- The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits.
- The receiver checks the relationships between the two sets of bits to detect errors.
- The ratio of redundant bits to data bits and the robustness of the process are important in any coding scheme.
- Two coding schemes : Block Coding and Convolution coding

Block Channel Encoding

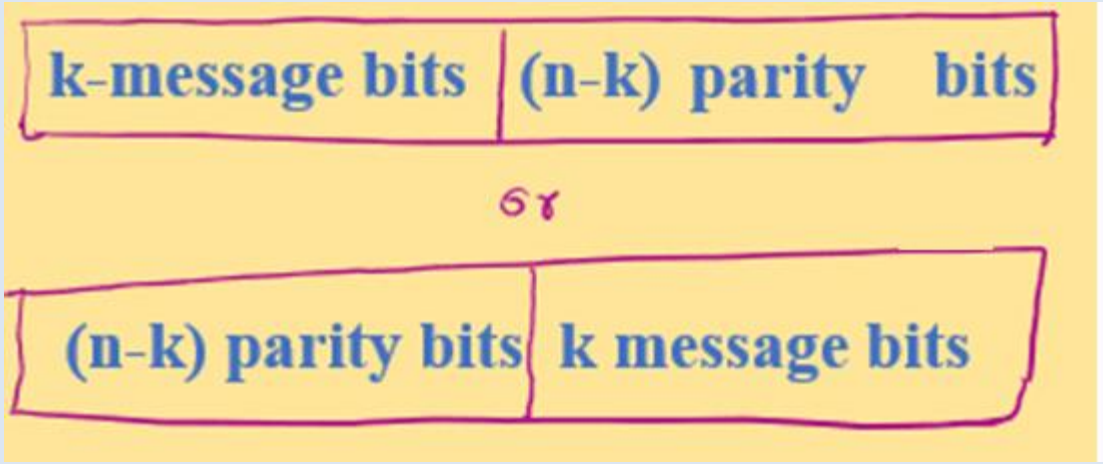
Change in notation

n = N = length of block code (entering the channel)
k = L = number of information / message bits in each block (entering channel encoder)

Take *k* (information) bits at a time and map each each *k*-bit sequence into a (unique) *n*-bit sequence, called a **code-word**.



- This code is called *(n,k)* code.
- Working with *k*-info-bit blocks means there are potentially $M=2^k$ different information blocks.


k-message bits
(n-k) parity bits
or
(n-k) parity bits
k message bits

Error Detection Technique

- Parity Check (Vertical Redundancy Check VRC)
- Longitudinal Redundancy Check LRC
- Checksum
- Cyclic Redundancy check CRC

Parity Check: Vertical Redundancy Check VRC

- Parity bit checking is used occasionally for transmitting ASCII characters, which have 7 bits, leaving the 8th bit as a parity bit.

- Two options:
  - Even Parity: Added bit ensures an even number of 1s in each codeword.

    A: 10000010
  - Odd Parity: Added bit ensures an odd number of 1s in each codeword.

    A: 10000011
- Even parity and odd parity are properties of a codeword, not a bit

Error Control using Parity Bit

- If an odd number of bits (including the parity bit) are transmitted incorrectly, the parity bit will be incorrect, thus indicating that a parity error occurred in the transmission.

- Example
  - Suppose we use even parity
  - Consider the codeword **x** = 1000001**0**

One bit error

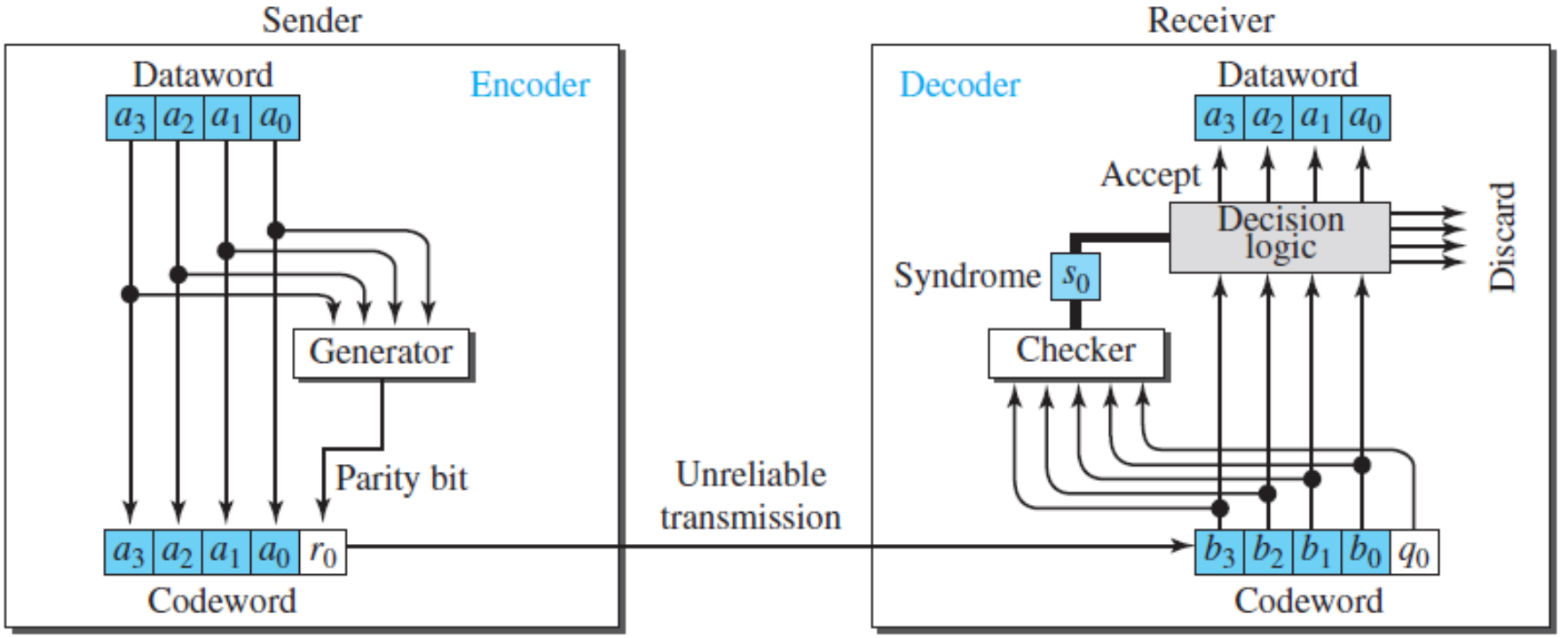
y = 10000110 -> odd parity => something wrong => error detected

Two bit error

y = 10000100 ->Even parity => Error is undetected

- Suitable for *detecting* errors; *cannot correct* any errors
- Even parity is used for synchronous transmission and odd parity for asynchronous transmission

Encoder and decoder for Simple parity-check



Longitudinal redundancy check (LRC)

- An error-detection method for a parallel group of bit streams (row of the data block)
- Each LRC bit makes the parity of each column (count 1's in each column)
- At the receiver, if the parity checker is not all 0's, then the error is detected

Original data	11001110	10111010	01110010	01010010

d_4 0 1 0 1 0 0 1 0
d_3 0 1 1 1 0 0 1 0
d_2 1 0 1 1 1 0 1 0
d_1 1 1 0 0 1 1 1 0
P_g 0 1 0 1 0 1 0 0

counts no. of 1's of each column:
using even parity (P_g)

Transmitted data	01010100	11001110	10111010	01110010	01010010

At receiver data

d_4 0 1 0 1 0 0 1 0
d_3 0 1 1 1 0 0 1 0
d_2 1 0 1 1 1 0 1 0
d_1 1 1 0 0 1 1 1 0
P_g 0 1 0 1 0 1 0 0
P_c 0 0 0 0 0 0 0 0

Error free at received data

Received errored data

d_4 0 0 0 1 0 1 1 0
d_3 0 1 1 0 0 0 1 0
d_2 1 0 1 1 1 0 1 0
d_1 1 1 0 0 1 1 1 0
P_g 0 1 0 1 0 1 0 0
P_c 0 1 0 1 0 1 0 0

Error at received data: discard

Checksum

- Checksum is an error-detecting technique that can be applied to a message of any length.
- In the Internet , the checksum technique is mostly used at the network and transport layer rather than the data-link layer.

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

For Internet checksum 16-bit words else n bits and the unit is divided into k sections

Example 1

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 arithmetic

	10101001
	00111001

Sum	11100010
Checksum	00011101

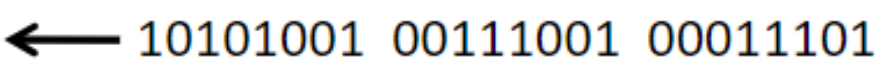
The pattern sent is :

10101001 00111001 00011101

Checksum

Example 2

Now suppose the receiver receives the pattern sent in Example 1 and there is no error.



When the receiver adds the three sections together, 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 3

Now suppose there is a burst error of length five that affects four bits.

10101*111* *11*111001 00011101

When the receiver adds the three sections together, it gets

```
                    10101111
                    11111001
                    00011101
                ------------------------
Result        1     11000101
Carry                     1
                ------------------------
Sum                 11000110
Complement          00111001  means that the pattern is corrupted.
```

Cyclic Codes

- Problem with checksum codes- Simply summing bits to form a checksum does not produce a sufficiently complex result.
- By complex we mean that the input bits should maximally affect the checksum, so that a wide range of bit errors or group of bit errors will affect the final checksum and be detected.
- A more complex “checksumming” operation can be formed by performing division rather than summation.
- 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.
- The properties of cyclic codes also allow greater scope for designing codes for specialized uses, especially in handling burst errors.
- Not surprisingly all modern channel coding techniques use cyclic codes, especially the important class of Cyclic Redundancy Check (CRC) codes for error detection.

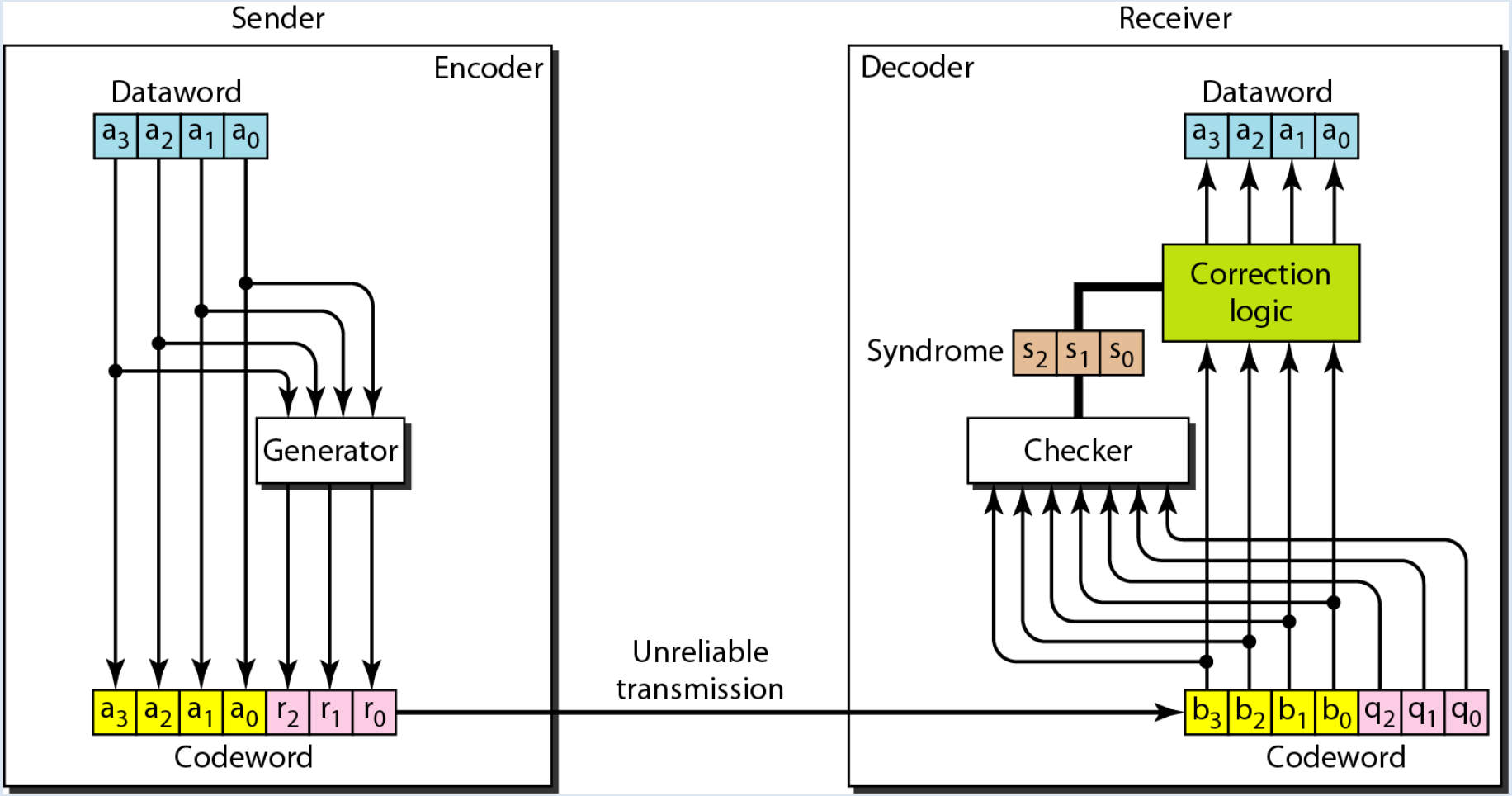
Cyclic Redundancy Check (CRC)

- A subset of cyclic codes called the cyclic redundancy check (CRC), which is used in networks such as LANs and WANs.
- The CRC codes are codes specifically designed for error detection. There are several reasons why CRC codes are the best codes for any form of practical error detection.
  i. The encoder and decoder implementation is fast and practical, especially for communication data which is processed in serial fashion.
  ii. The CRC codes handle many combinations of errors, especially combinations of burst errors, as well as random errors.

CRC code with C(7,4)

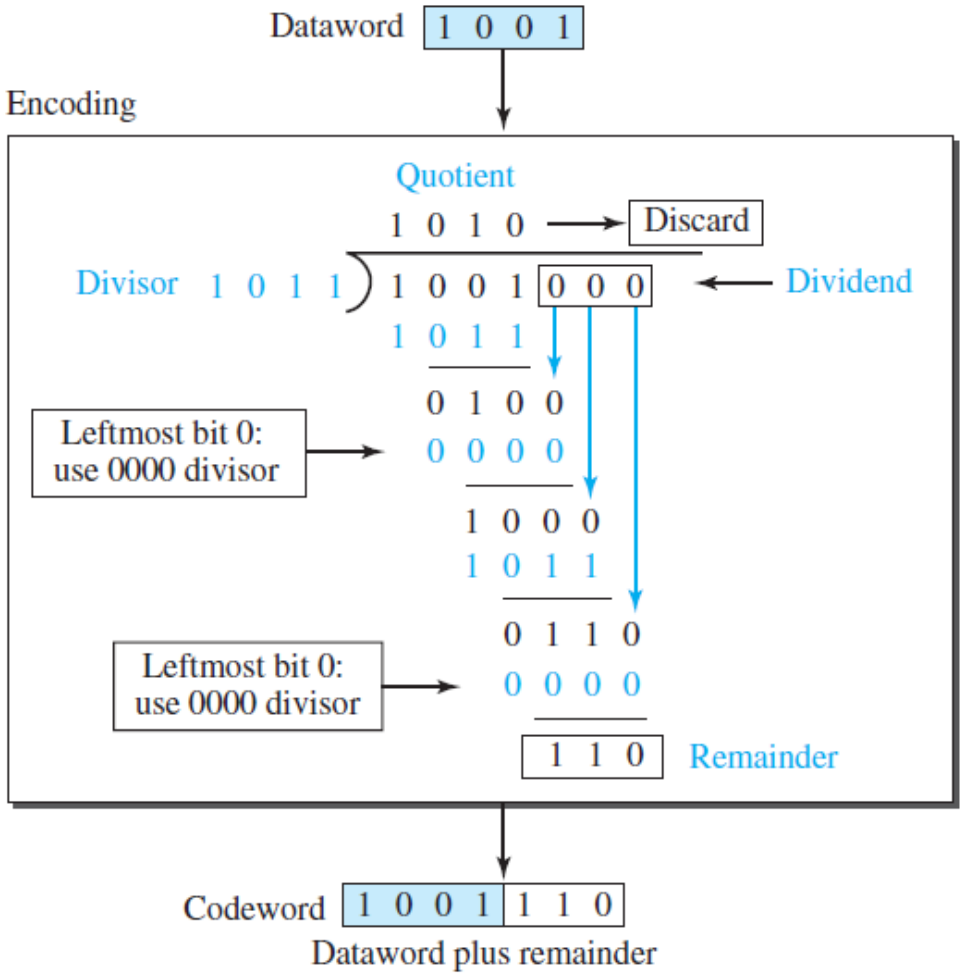
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

CRC encoder and decoder

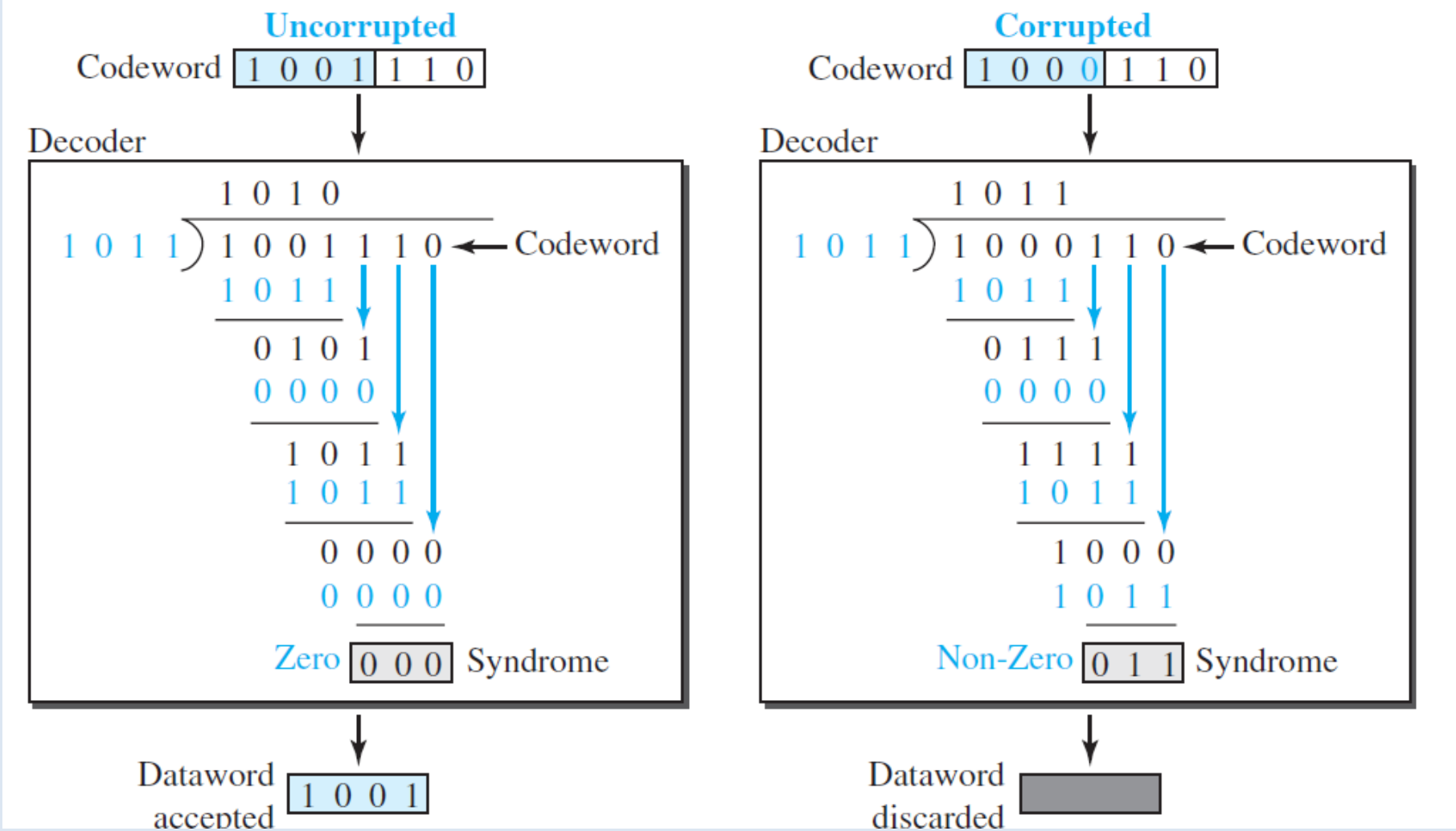


- In the encoder, the dataword has k bits (4 bits); the codeword has n bits (7 here). The size of the dataword is augmented by adding n-k (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size n – k + 1 (4 here).The generator divides the augmented dataword by divisor (modulo-2 division). The quotient of the division is discarded; the remainder (r_2 r_1 r_0) is appended to the dataword to create the codeword.
- The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to checker, which is a replica of the generator. The remainder produced by the checker is a syndrome of n – k (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all 0s, the 4 left-most bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded(error).

Division in CRC encoder

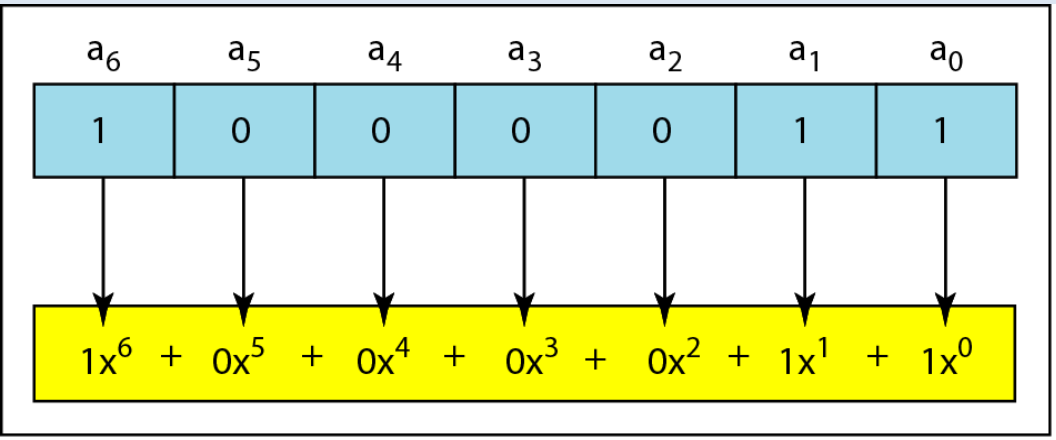


Division in the CRC decoder for two cases

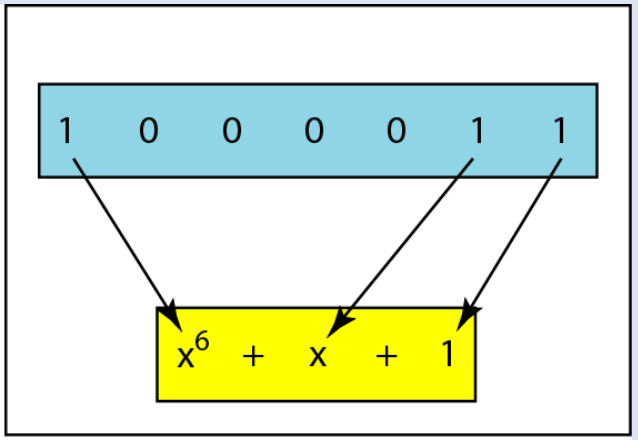


Polynomial

- The divisor in a cyclic code is normally called the generator polynomial or simply the generator
- A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1.



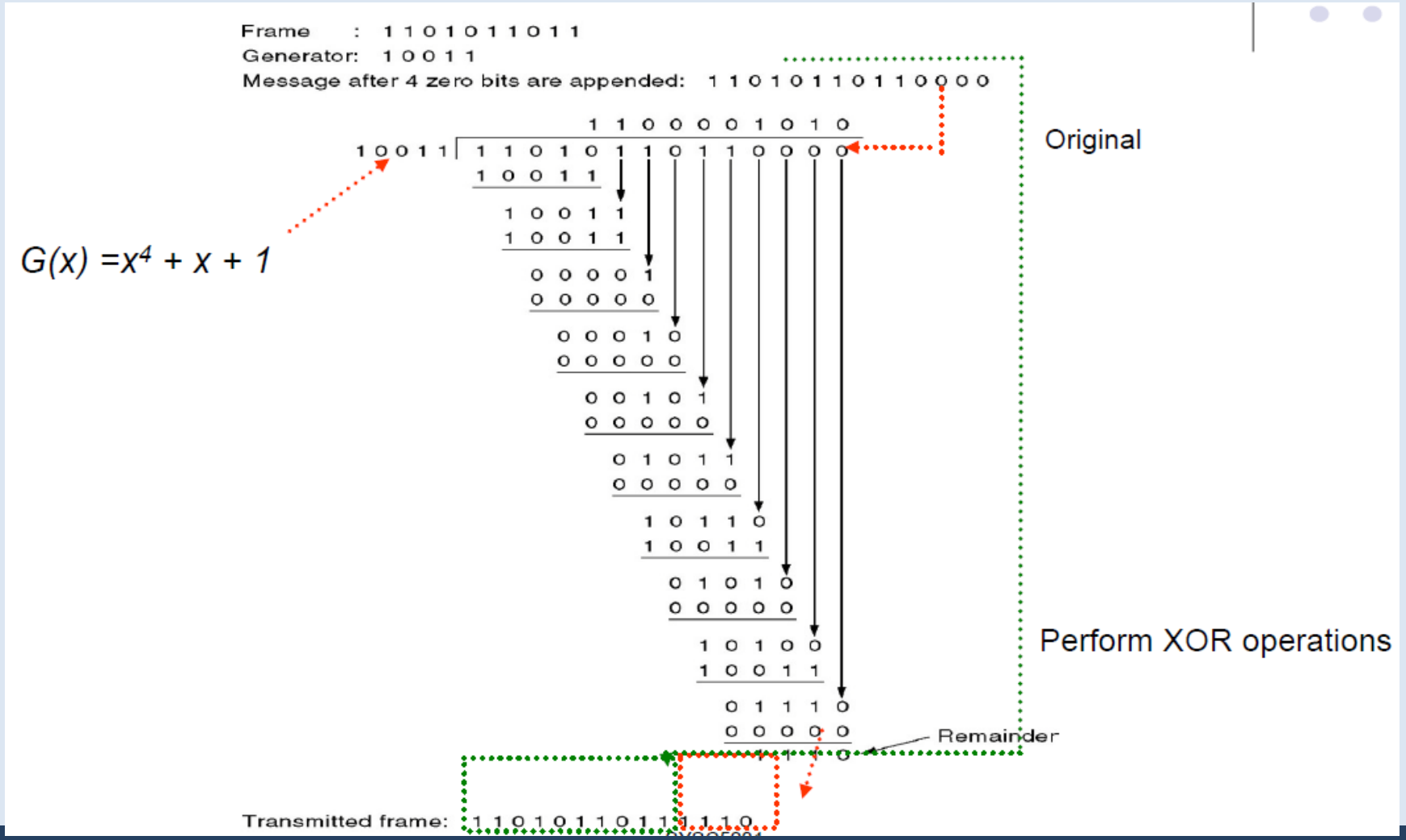
a. Binary pattern and polynomial



b. Short form

- Degree of the polynomial x^6+x+1 is 6.
- The degree of a polynomial is 1 less than the number of bits. Bit pattern in this case : 7 bits
- Format: k-bit message + (n-k) bit CRC = n bits in total

Size of generator n-k + 1 = 5 So add FCS of size n-k = 4



Numerical

A bit stream 10011101 is transmitted using the CRC method. The generator polynomial is x^3+1. Determine the transmitted codeword. Suppose the third bit from the left is inverted during the transmission. Show that this error is detected at the receiver end.

Soln Message: 10011101, size of message k = 8

Generator polynomial: x^3+1 = 1001: Generator size: n-k + 1 = 4

So add FCS of size n-k =3

Homework

Transmitted codeword: 10011101 100

Received Codeword: 10111101100 Syndrome 100 For error free, syndrome should be zero So discard it

Standard Polynomials

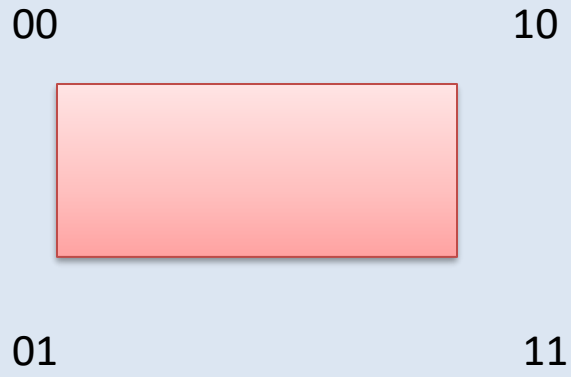
Name	*Polynomial*	*Application*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Error Correction: Hamming Codes

- It is nice to be able to detect that a transmission error occurred. It would be nicer to be able to find and correct the error. There are several schemes to do this. A standard one is calling Hamming code.
- A Hamming code is a combination of 0s and 1s, but not all combinations of 0s and 1s are valid codes. The Hamming distance between two binary numbers of the same length is the number of positions in the numbers that have different values. For example, the Hamming distance between 1101 and 1001 is 1, since they differ in only one position. The Hamming distance between 1101 and 1011 is 2, since they differ in two positions.
- Consider a Hamming code of length 1. That is, there are only two possible codes, 0 and 1. If both represent valid codes, there is no way to detect a transmission error. If an error occurs, then a 0 become a 1, or a 1 becomes a 0, and both are valid codes. All possible received words are valid, and no detection or correction is possible.

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

- Now suppose we use a two-bit code. This provides four possible values: 00, 01, 10, and 11. Suppose we specify that only 00 and 11 are valid codes. If 01 or 10 are received, we know an error has occurred. That is, we have error detection capability



- Notice that we arranged the possible codes so that the Hamming distance between any two adjacent vertices is 1, and the Hamming distance between any two non-adjacent vertices is 2.
- This provides error detection, but problems remain.

A. We can't detect which bit is wrong if we detect an error. For example, suppose we receive 01. It is not a legal code, so an error has occurred. But we don't know whether it came from 00 (with the second bit changed to 1) or from 11 (with the first bit changed to 0).

B. A double error would not be detected at all. If we receive 11, it could have originally been 00 or 11.

Suppose three bit values. There are eight possible values represented as vertices. Choose 000 and 111 are the valid codes.

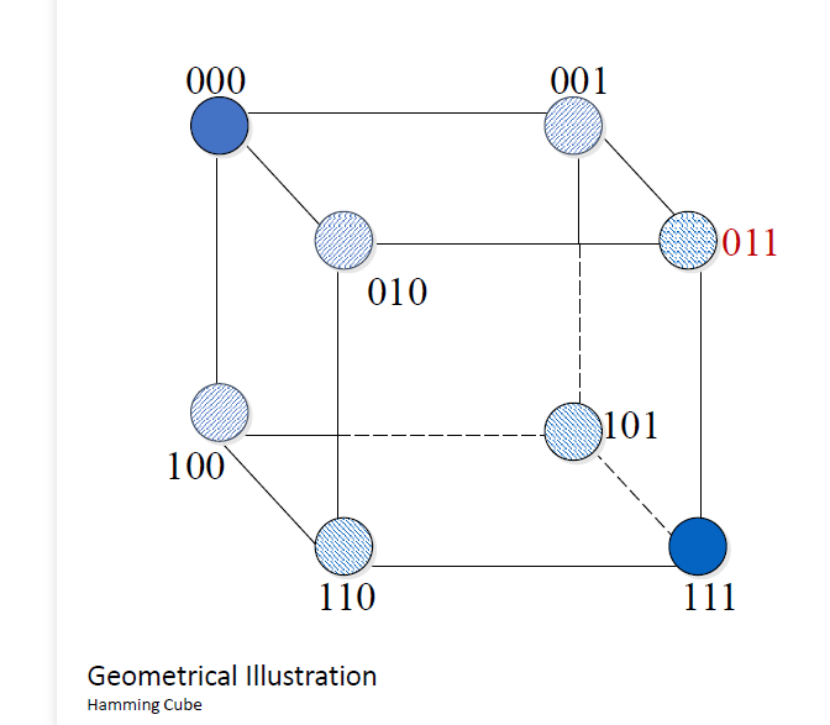
A single error in transmission produces a word that is a distance of 1 from the correct value and distance of 2 from the other value. For example, *011* is received.

d(a,b) = W(a ⊕ b)

d(011,000) = 2

d(011,111)= 1

Therefore 011 is closer to 111.



Geometrical Illustration
Hamming Cube

Indeed, if an invalid code is received, and we assume only one transmission error, we can correct the error by substituting the closest legal code word from the diagram. The invalid codes 001, 010 and 100 correct to 000; while 011, 101, and 110 correct to 111.

This error correcting code works well for single transmission errors, but there are still some problems.

- If a double error occurs, it will be corrected erroneously! For example, suppose 011 is received. It will be corrected to 111. But if the original was 000, and the second and third bits got switched in transmission, the correction would be wrong.
- A triple error would not be detected at all

Here are some general results.

- If the minimum distance between any two valid codes is 1, no detection or correction is possible.
- If the minimum distance between any two valid codes is 2, detection of single errors is possible, but not correction. A double error will not be detected.
- If the minimum distance between any two valid codes is 3, detection and correction of single errors is possible. A double error will be corrected erroneously, and a triple error will not be detected.

Another Example

Message (L=2)	Codeword (N=3)
00	000
01	001
10	011
11	111

$b=b_1b_2b_3$	Closest codeword	Action
010	000 (b_2 in error), 011 (b_3 in error)	error detected
100	000 (b_1 in error)	single-bit error corrected to 000
101	001 (b_1 in error), 111 (b_2 in error)	error detected
110	111 (b_3 in error)	single-bit error corrected to 111

Error detection

A block code is said to detect all combinations of *t* errors provided that for each codeword *a* and each received word *b* obtained by corrupting *t* bits in *a*, *b* is not a code word (and hence can be detected). This property is important in situations where the communication system uses ARQ error-control coding.

A code detects all *t* errors if and only if its minimum distance is greater than *t*.
Hamming distance > *t*

Error correction

A block code is said to correct all *t* errors provided that for each codeword *a* and each received word *b* obtained by corrupting *t* bits in *a*, the decoding (maximum likelihood) leads uniquely to *a*. This property is important in situations where the communication system uses FEC error-control coding.

A code corrects all *t* errors if and only if its minimum distance is greater than 2*t*.
Hamming distance > 2*t*

General method to obtain a Hamming code

First decode on a length for the codes. For *r* parity bits, a maximum code length of 2^r-1 bits.

- All positions that are powers of 2 are parity bits (positions 1, 2, 4, 8, …)
- The remaining positions are data bits (positions 3, 5, 6, 7, 9, …)
- Each parity bit checks the parity of some of the data bits. The position of the parity bit determines the bits that it checks. Alternately check and skip bits as follows.
    - o Bit 1: Check and skip bits in groups of 1 (bits 1, 3, 5, 7, 9, …);
    - o Bit 2: Check and skip bits in groups of 2 (bits 2, 3, 6, 7, 10, 11, …);
    - o Bit 4: Check and skip bits in groups of 4 (bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, …), and so on

Set each parity bit to 0 or 1 to produce the proper parity for the bits it checks.

Hamming Code with 3 parity bits

- Number of Parity Bits *r = 3*
- Number of data bits (Maximum) k = 2^r-1-r = 8-1-3 =4

Maximum data bits we can take for r = 3 is 4.

- Position of Parity bits (Power of 2)---1,2,4

Assuming a 7-bit code then with bits 3,5,6 and 7 holding the data bits.

Bit Position	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Parity Bit				P4		P2	P1
Data bit	D4	D3	D2		D1		
Hamming code	D4	D3	D2	P4	D1	P2	P1

Consider transmitting data: 1011 where even parity hamming code is used.

Bit Position	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Parity Bit				P4		P2	P1
Data bit	D4	D3	D2		D1		
Hamming code	1	0	1	0	1	0	1

- For P1 : Check the value on position 3, 5, 7, i.e, 111, which is odd.

For even parity, the bit value for P1 is 1.

- For P2 : Check the value on position 3, 6, 7, i.e, 101, which is even.

For even parity, the bit value for P2 is 0.

- For P4 : Check the value on position 5, 6, 7, i.e., 101, which is even.

For even parity, the bit value for P4 is 0.

The resulting Hamming code is 1010101

The transmitted Hamming code is 1010101 is transmitted and the received code is 1000101. Hamming code detects and corrects the error.

Bit Position	7	6	5	4	3	2	1
Parity Bit				P4		P2	P1
Data bit	D4	D3	D2		D1		
Received Hamming code	1	0	0	0	1	0	1

- Bits 3,5 and 7 are 101, so P1 should be 0 for even parity.
- Bits 3,6 and 7 are 101, so P2 is correct.
- Bits 5,6 and 7 are 001, so P4 should be 1 for even parity

(We can use another method to find the error parity bit. Solved in lecture class)

The transmitted Hamming code is 1010101 is transmitted and the received code is 1000101. Hamming code detects and corrects the error.

Bit Position	7	6	5	4	3	2	1
Parity Bit				P4		P2	P1
Data bit	D4	D3	D2		D1		
Received Hamming code	1	0	0	0	1	0	1

- Add the numbers of the parity bit positions that are incorrect. 1+4=5, so data bit 5 is incorrect. The correct code is 1010101, corresponding to data 1011. In general, find the bad parity bits and add their locations to find the location of the bad data bit.

Question: Find a six-bit, single bit error correcting and detecting Hamming code for the data values 000, 001, 010, 011, 100, 101, 111.

(SOLUTION IN LECTURE CLASS)

Convolution Codes

- 1955: Peter Elias introduces the concept of convolution codes in which the stream of info bits is encoded by a finite-state machine.
- Peter Elias received
  - ❑ Claude E. Shannon Award in 1977
  - ❑ IEEE Richard W. Hamming Medal in 2002
    - ➢ For "fundamental and pioneering contributions to information theory and its applications
- The encoder has memory.
  - ❑ In other word, the encoder is a sequential circuit or finite-state machine.
    - ❖ Easily implemented by shift register(s).
    - ❖ The state of the encoder is defined as the contents of its memory.

Peter Elias (1923-2001)

Binary Convolution Codes

- The encoding is done on a continuous running basis rather than by blocks of *k* data digits.
- So, we use the terms bit streams or sequences for the input and output of the encoder.
- Convolutional codes have memory that uses previous bits to encode or decode following bits

Binary Convolution Codes

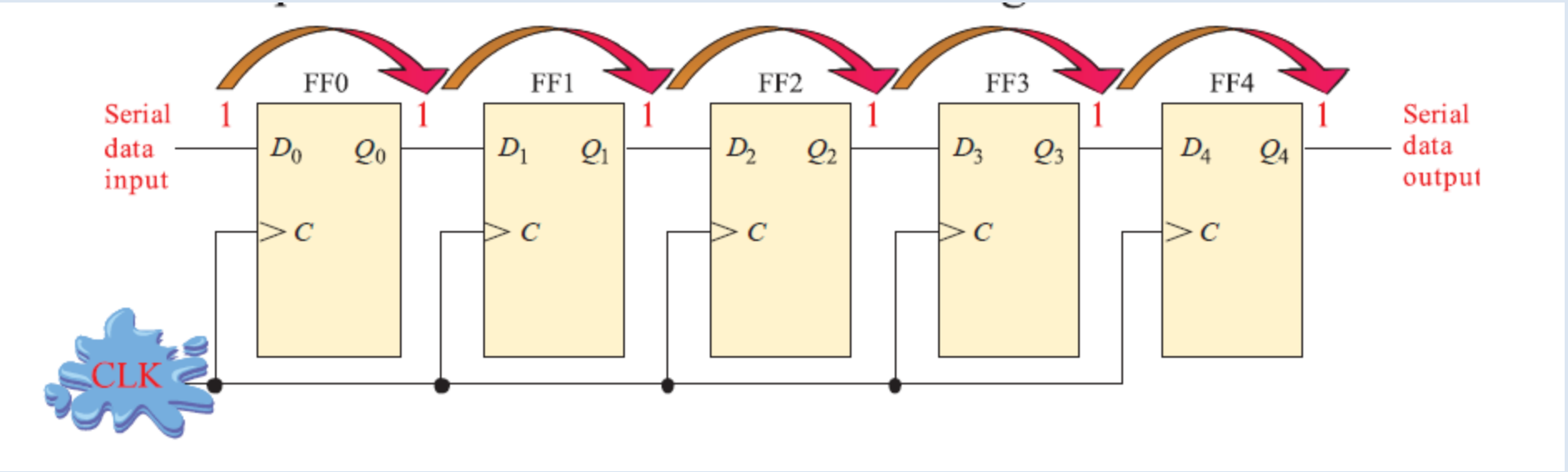
- (n,k,L) ….L- Code memory depth ,
- k inputs,
- n output coded bits

k and n are usually small.

- In general, a code rate $\frac{k}{n}$ convolutional encoder has
- For simplicity and for practical purposes, only rate $\frac{1}{n}$ binary convolutional codes are considered here.
  - k=1
  - These are the mostly used binary codes.

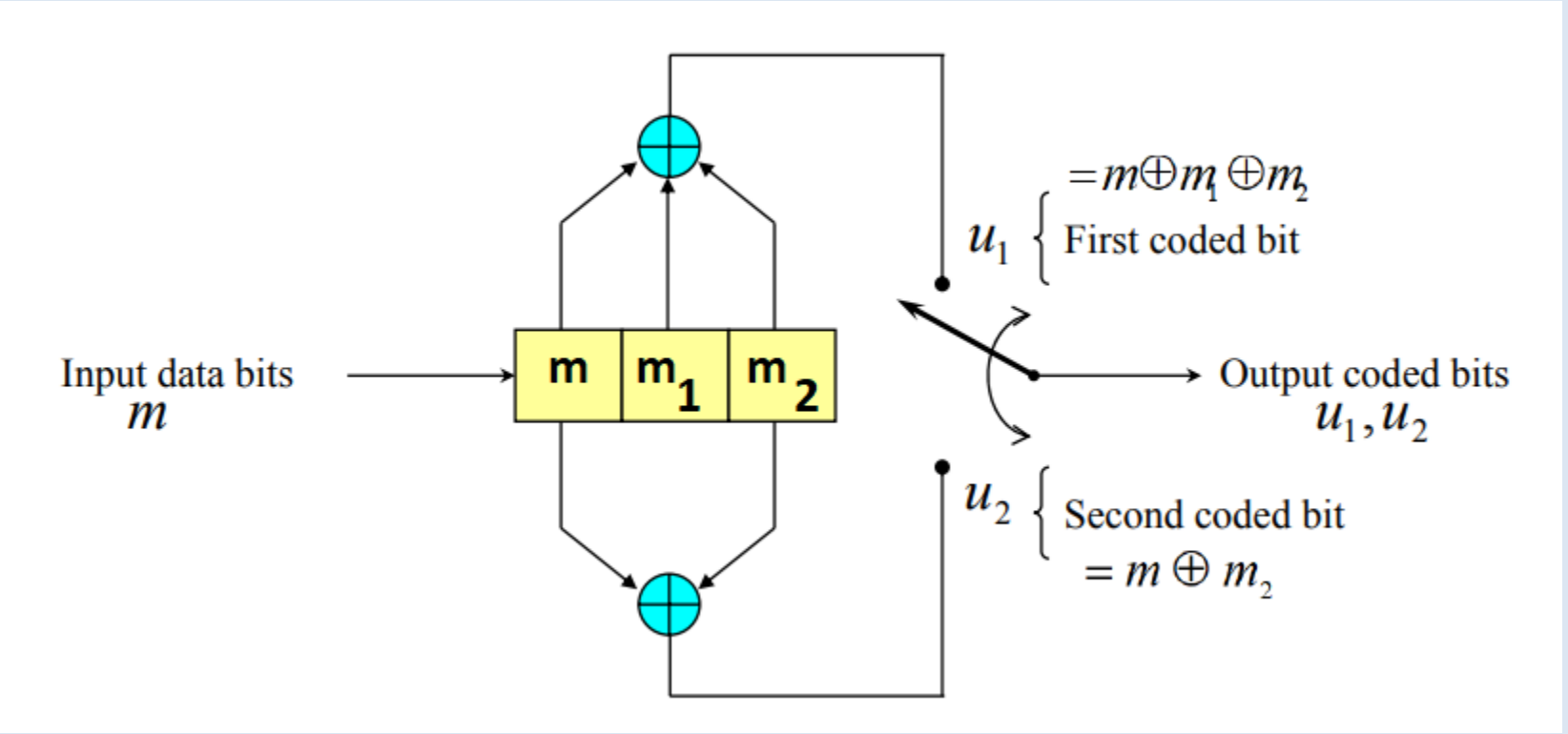
(Serial-in/Serial-out) Shift register

- Accept data serially: one bit at a time on a single line.
- Each clock pulse will move an input bit to the next FF. For example, a 1 is shown as it moves across.
- Example: five-bit serial-in serial-out register.

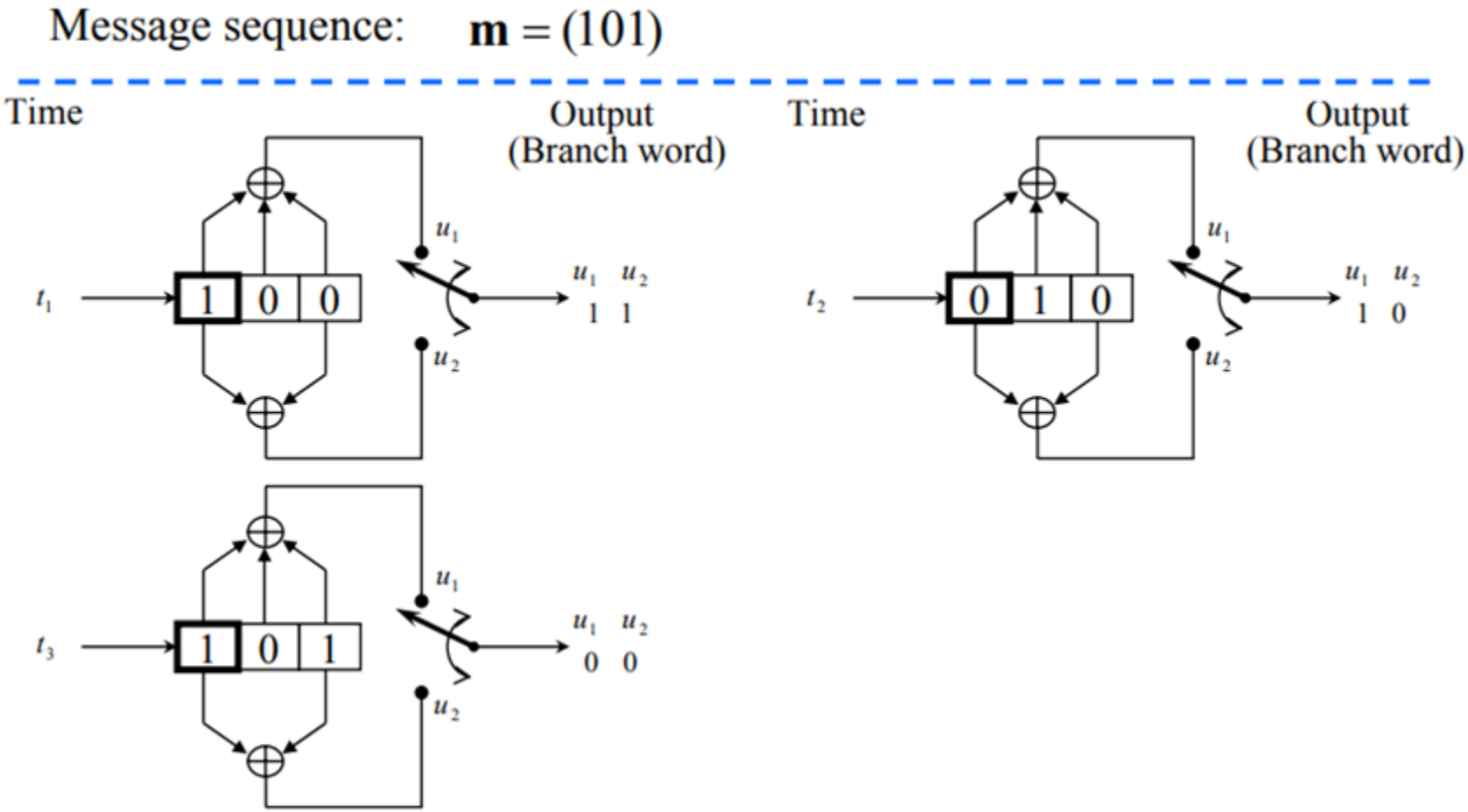


Convolutional Codes (rate ½, K=3)

– 3 shift-registers, where the first one takes the incoming data bit and the rest form the memory of the encoder.



Example of Convolution Code



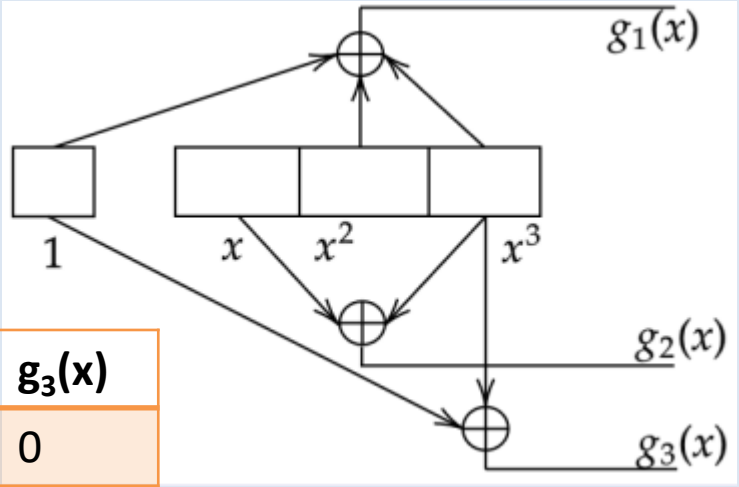
This bit is passed at first.

$\mathbf{m} = (101)$ → Encoder → $\mathbf{U} = (11\ \ 10\ \ 00)$

Convolution Code: Example

The generator polynomials of a 1/3 binary convolution encoder is given by $g_1(x) = x^3+x^2+1$ $g_2(x) = x^3+x$ $g_3(x) = x^3+1$. Calculate the encoded bit sequence for input 011.

Soln: Code rate R = k/n = 1 / 3 , k = 1 and n =3

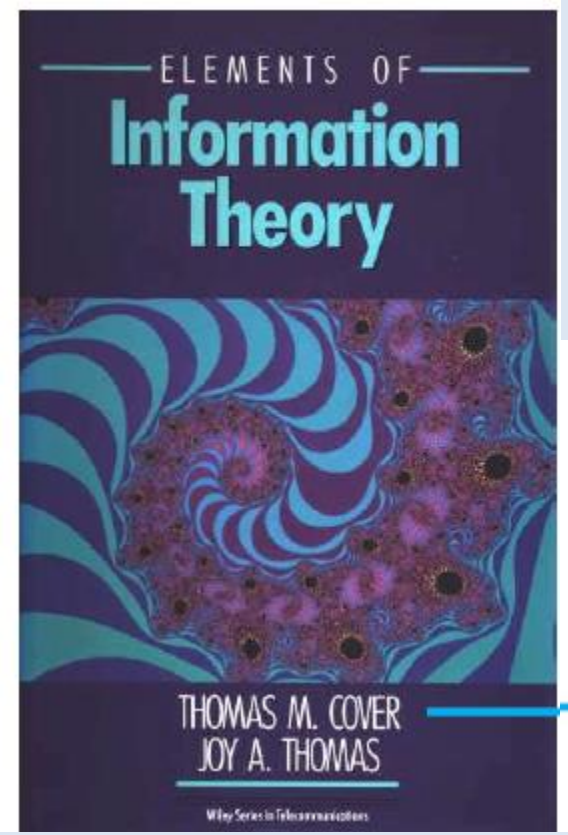


t	Input	x	x^2	x^3	$g_1(x)$	$g_2(x)$	$g_3(x)$
1	0	0	0	0	0	0	0
2	1	0	0	0	1	0	1
3	1	1	0	0	1	1	1

The encoded bit sequence for input sequence 011 is 000101111

Main Reference

- Elements of Information Theory



''the jewel in Stanford's crown"

One of the greatest information theorists since Claude Shannon (and the one most like Shannon in approach, clarity, and taste).

Terminology

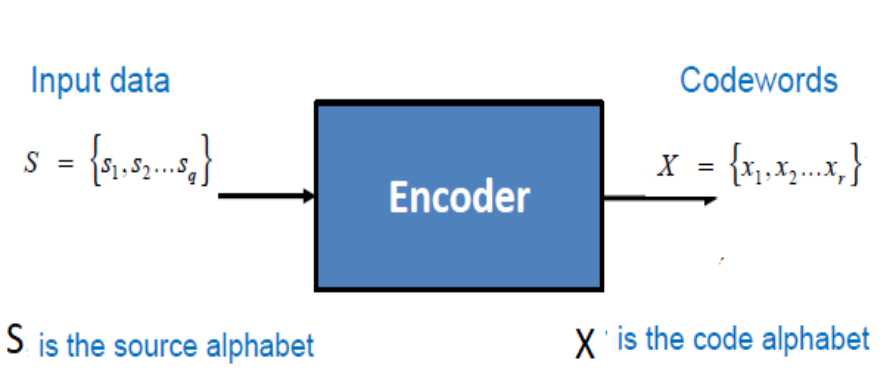
- *Letter, symbol or character:* Any individual member of alphabet set
- Message word: A finite sequence of letters of the alphabet
- Length of a word: The number of letters in a word

❑ Maximum entropy occurs when the symbols are equally probable and source symbols can be represented efficiently and close to H(S) by Source Encoder

❑ Instead of sending $\log_2|S|$ bits per symbol, we can transmit H(S) bits per symbol on average

❑ By assigning short codewords for frequent symbols and long codewords for rare symbols.

Variable – length code



- When we map each symbol s_i from S to a code word from X of length, say n:

  Source encoding

- The reverse process of mapping the word X_i to the source symbol s_i is source decoding

Coding System

Why do we use source coding?

- Necessary to convert from the source representation to the channel/storage representation. (*Example:* English language text converted to binary using ASCII encoding)
- The encoding process can exploit source redundancies (low entropy) such as to make the code representation more efficient (higher entropy) (*Example:* lossless compression of data)

NOTE

1. We will be making extensive use of binary codes, $X = \{0,1\}$, as this is the storage and transmission "alphabet" of digital communication and computer systems.
2. We will also make use of ternary codes, $X = \{0,1,2\}$, as these form of the basis of various digital signal data encoding strategies.
3. The main task of source coding is to find strategies for making the code representation, using uniquely decodable codes, efficient for both transmission and storage.

Definitions

- Nonsingular code: All the code words are distinct

Unique Decodability

- A code is said to be uniquely decodable if there are no instances of non-unique (ambiguous) source decoding for any and all possible sequences of codewords. **(operational definition)**

All practical codes must be uniquely decodable
A block code of length n which is nonsingular is also uniquely decodable

Consider the source alphabet, S = {s_1,s_2,s_3,s_4} and binary code alphabet, X = {0,1}. Consider the following three possible source encodings:

Source Symbols	Code A	Code B	Code C
s_1	0	0	00
s_2	11	11	01
s_3	00	00	10
s_4	11	010	11

- Code A is not non-singular since $X_2=X_4$ =11
- Code B is non-singular (all codewords are distinct)but it is NOT uniquely decodable since the code sequence 00 can be decoded either as s_3 or s_1s_1 (non-unique decoding)
- Code C is non-singular block code of length 2 and is thus uniquely decodable

Source coding

Basic Idea: Most sources of information contain redundant information that does not need to be actually transmitted(or stored).

Ex: Video Signal, 25 pictures/sec, 256 X 256 pixels/picture, 8 bits/pixel

→ Data Rate ≈ 13 Mbits/sec. It's too much!

Ex: Movie, 3 hours

→≈ 140 Gbits to download and store. Yet, we have not included the sound and other features…

=→ We need to compress!

Compression: Use lesser bits to represent a given amount of info.

Two techniques:

- Lossless compression(no degradation of info, low compression ratio);
- Lossy compression(degradation of info, high compression ratio).

Q. Which one do we use?

A. It depends on the application

Average Information

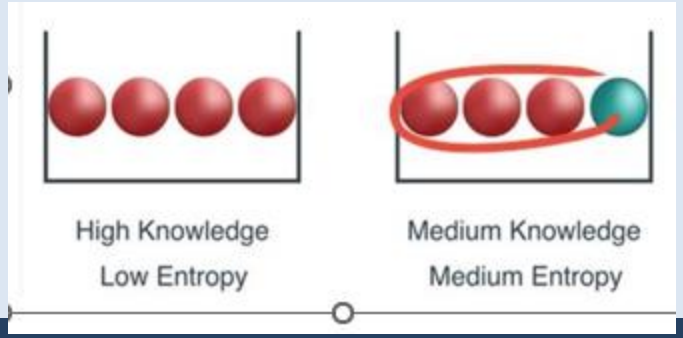
- Self information can be extended to quantifying the average information of source itself.
- Entropy defines the most important quantity of information theory.

➢ The average information or entropy H(s) of the source is

$$H(S) = \sum_S P(s_i) I(s_i) = \sum_S P(s_i) \log \frac{1}{P(s_i)} \text{ bits}$$

$S = \{s_1, s_2, s_3, \ldots\ldots, s_q\}$

- $0<P<1$ means $\log (1/P) > 0$ Then $H(S) > 0$



- Average Length of a codeword

$$L = \sum_{i=1}^{q} P_i l_i$$

Where q- symbols, P_i $-$ Probability of instantaneous code, l_i- length of instantaneous code

We can define the efficiency of our encoding technique:

$$\eta = \frac{H(s)}{L}$$

Average length of a codeword, in bits

Example

Q. Consider the source S ={s1,s2,s3} with P(s1) = 0.5, P(s2)= P(s3) = 0.25. Then

$H(S) = 0.5 \log(2) + 0.25 \log(4) + 0.25 \log(4) = 1.5$ bits

Thus a typical message from the source will require on average 1.5 bits of information per source.

Lossless compression techniques

- Compression algorithms allowing the exact original data to be reconstructed from the compressed data.
- Used in software compression tools such as the popular ZIP file format.
- Used when the original and the decompressed data must be identical → A must for executable programs and source code.

Types:

a) Run Length Coding
b) Huffman Coding
c) Dictionary Coding

Lossless compression techniques – Run length

- Basic Data compression method that replaces consecutively repeated symbols in a source with a code pair which consists of either the repeating symbol and the number of its occurrences, or non-repeating symbols.

Ex 1: Run length encoding

FAX: 1728 pixels per scan line, 3.85 scan lines/mm, and page length of 30 cm -> ≈ 2 million pixels/page!

Without source coding, transmission time for a single page using a 56-Kbit/s modem ≈ 36 seconds.

This can be reduced by taking advantage of the fact that pixels tend to be black or white in runs -> Run length encoding.

Example: String ABBBBBBBCC can be represented by A1B7C2, (1,A) (7,B), (2,C)

- Every single bit of data originally transmitted remains after decompression. After decompression, all the information is completely restored.
- One can use lossless compression whenever space is a concern, but the information must be the same. In other words, when a file is compressed, it takes up less space, but when it is decompressed, it still has the same information.
- The idea is to get rid of redundancy in the information.

Lossless Data Compression – Huffman Code

We first consider the design of compact binary codes (S = {0,1}) using the binary Huffman coding technique. This is one of the earliest techniques proposed by Huffman in 1952 which is still widely used due to simplicity of implementation and optimal characteristics.

Reduced Source

Consider the source S with q symbols: $s_1,s_2,\ldots,s_q$ and symbol probabilities: $P_1 \geq P_2 \geq \ldots\ldots \geq P_q$. By combining the last two symbols of S into one symbol (by adding their probabilities) we obtain a new source, termed a **reduced source** of S, containing only q-1 symbols. Call this reduced source S_1. Successive reduced sources $S_2S_3\ldots$ can be formed until we are left with a source with only two symbols.

Result

Assume that we have a compact code for the reduced source S_j. A compact code for the preceding source S_{j-1} (which can be the original source) is formed as follows:

Designate the two symbols from S_{j-1}, $s_{\sigma 0}$ and $s_{\sigma 1}$, as the symbols added together to form the new symbol, s_σ of S_j. We assign to each symbol of S_{j-1}, except $s_{\sigma 0}$ and $s_{\sigma 1}$, the code word used by the corresponding symbol of S_j. The code words used by $s_{\sigma 0}$ and $s_{\sigma 1}$ are formed by appending a 0 and 1, respectively, to the code word used for s_σ.

Binary Huffman Code Algorithm

1. Reduce the source S to S_1 and continue on to S_2, S_3 until we reach a source with two symbols (or one for which it is easy to design a compact code).

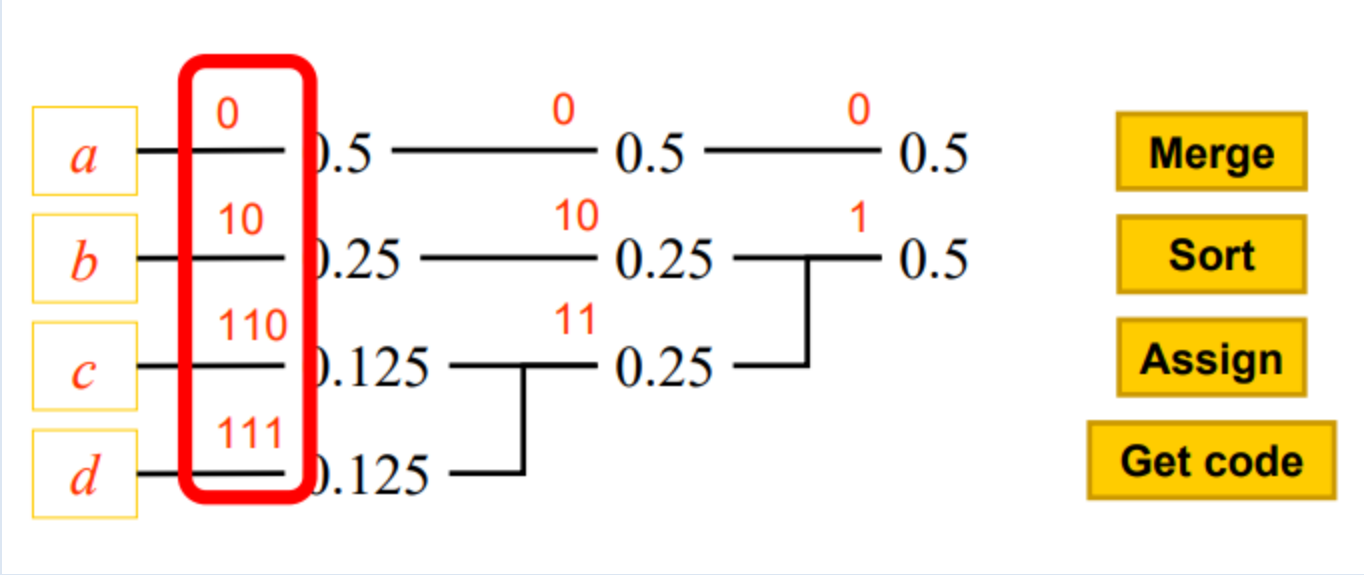
2. Assign a compact code for the final reduced source. For a two symbol source the trivial code is a 0 and 1.

3.Backtrack to the original source assigning a compact code for the j^{th} source by the method described in last slide. The compact code we assign to the original source is the binary Huffman Code.

Huffman Codes – Two step Algorithm

Two- step algorithm

1. Iterate:
   - Merge the least probable symbols
   - Sort
2. Assign Bits



Consider a 6 symbol source (q=6) with probability assignments as follows:

s_1=0.3 s_2=0.4 s_3=0.06

s_4=0.1 s_5=0.1 s_6=0.04

We re-order the symbols in decreasing order of probability and reduce the source until we are left with a 2- symbol source. At each stage of the reduction we need to keep track of s_σ which we represent as the underlined symbol probability corresponding to the combination of the $s_{\sigma 0}$ and $s_{\sigma 1}$ probabilities represented in **bold** from the previous reduced source:

S		S_1	S_2	S_3	S_4
s_2	0.4	0.4	0.4	0.4	0.6
s_1	0.3	0.3	0.3	**0.3**	0.4
s_5	0.1	0.1	**0.2**	**0.3**	
s_4	0.1	**0.1**	**0.1**		
s_3	**0.06**	**0.1**			
s_6	**0.04**				

We then back-track and form the compact code for each successive reduced source until we reach the compact code for the original source:

s_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
s_1	0.3	00	0.3	00	0.3	00	**0.3**	00	0.4	1
s_5	0.1	011	0.1	011	**0.2**	010	**0.3**	01		
s_4	0.1	0100	**0.1**	0100	**0.1**	011				
s_3	**0.06**	01010	**0.1**	0101						
s_6	**0.04**	01011								

Hence:

s_1	00
s_2	1
s_3	01010
s_4	0100
s_5	011
s_6	01011

We can define the efficiency of our encoding technique:

$$\eta = \frac{H(s)}{L}$$

Average length of a codeword, in bits

$$Average\ length\ L = \sum_{i=1} P_i l_i$$

Average Codeword Length L = 2*0.3+1*0.4+5*0.06+4*0.1+3*0.1+5*0.04 = 2.2 bits/ symbol

Entropy H(S) = 2.1435 bits/ symbol

Efficiency = 97%

Redundancy = 1 –Efficiency= 1 – 0.97 = 0.03

NOTE

1. There can be different compact code simply by interchanging how we choose when to use a 0 or a 1. The relabelling doesn't change the code word lengths.
2. When we encounter symbols of equal probability in our reduced source the added flexibility can lead to codes with different individual code word lengths. But the average length is still the same.

Lossless Data Compression – Dictionary Coding

- Given an input source, we want to
  1. Identify frequent symbol patterns
  2. Encode those more efficiently
  3. Use a default (less efficient) encoding for the rest
  4. Hopefully, the average bits per symbol gets smaller

- In general, dictionary-based techniques works well for highly correlated data (e.g. text), but less efficient for data with low correlation (e.g. independent and identically distributed i.i.d. sources)

Dictionary Coding - Example

- The source alphabet A = { a, b, c, d, r }

Based on knowledge about the source, we build the dictionary shown in Table

- Dictionary:

TABLE **A sample dictionary.**

Code	Entry	Code	Entry
000	*a*	100	*r*
001	*b*	101	*ab*
010	*c*	110	*ac*
011	*d*	111	*ad*

TABLE **A sample dictionary.**

Code	Entry	Code	Entry
000	*a*	100	*r*
001	*b*	101	*ab*
010	*c*	110	*ac*
011	*d*	111	*ad*

Suppose we wish to encode the sequence

abracadabra

The encoder reads the first two characters *ab* and checks to see if this pair of letters exists in the dictionary. It does and is encoded using the codeword 101. The encoder then reads the next two characters *ra* and checks to see if this pair occurs in the dictionary. It does not, so the encoder sends out the code for *r*, which is 100, then reads in one more character, *c*, to make the two-character pattern *ac*. This does exist in the dictionary and is encoded as 110. Continuing in this fashion, the remainder of the sequence is coded. The output string for the given input sequence is 101100110111101100000. ♦

Static Vs Adaptive Dictionary

- The dictionary holds a list of strings of symbols and it may be static or dynamic (adaptive)
- Static dictionary – permanent, sometimes allowing the addition of strings but no deletions
- Dynamic dictionary – holding strings previously found in the input stream, allowing for additions and deletions of strings as new input symbols are being read

Adaptive Compression – LZ Codes

Lempel-Ziv Algorithm

- Original ideas published by Jacob Ziv and Abraham Lempel in 1977 (LZ77/LZ1) and 1978 (LZ78/LZ2)
- The most well-known dictionary-based technique, LZW, is a modification to LZ algorithms published by Terry Welch in 1984

Advantages of LZ coding to Huffman Coding

- No need for any source statistics.
- Fast and efficient encoding and decoding implementation.

Lempel Ziv encoding

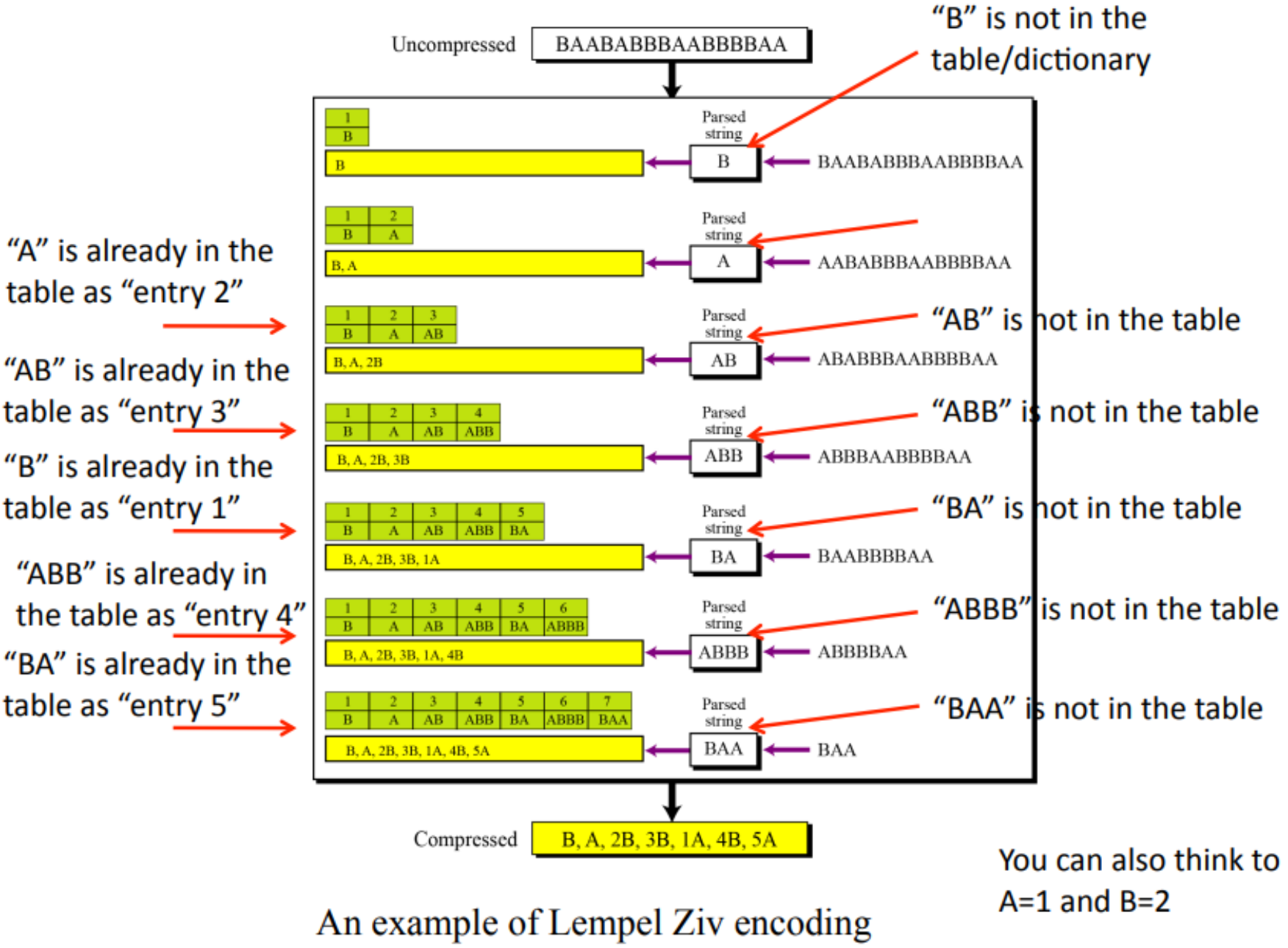
Basic idea:

In this phase there are two concurrent events:

(1) building an indexed dictionary,

(2) and compressing a string of symbols.

-The algorithm extracts the smallest substring that cannot be found in the dictionary from the remaining uncompressed string. It then stores a copy of this substring in the dictionary as a new entry and assigns it an index value.



An example of Lempel Ziv encoding

Encoder

BAABABBBAABBBBAA

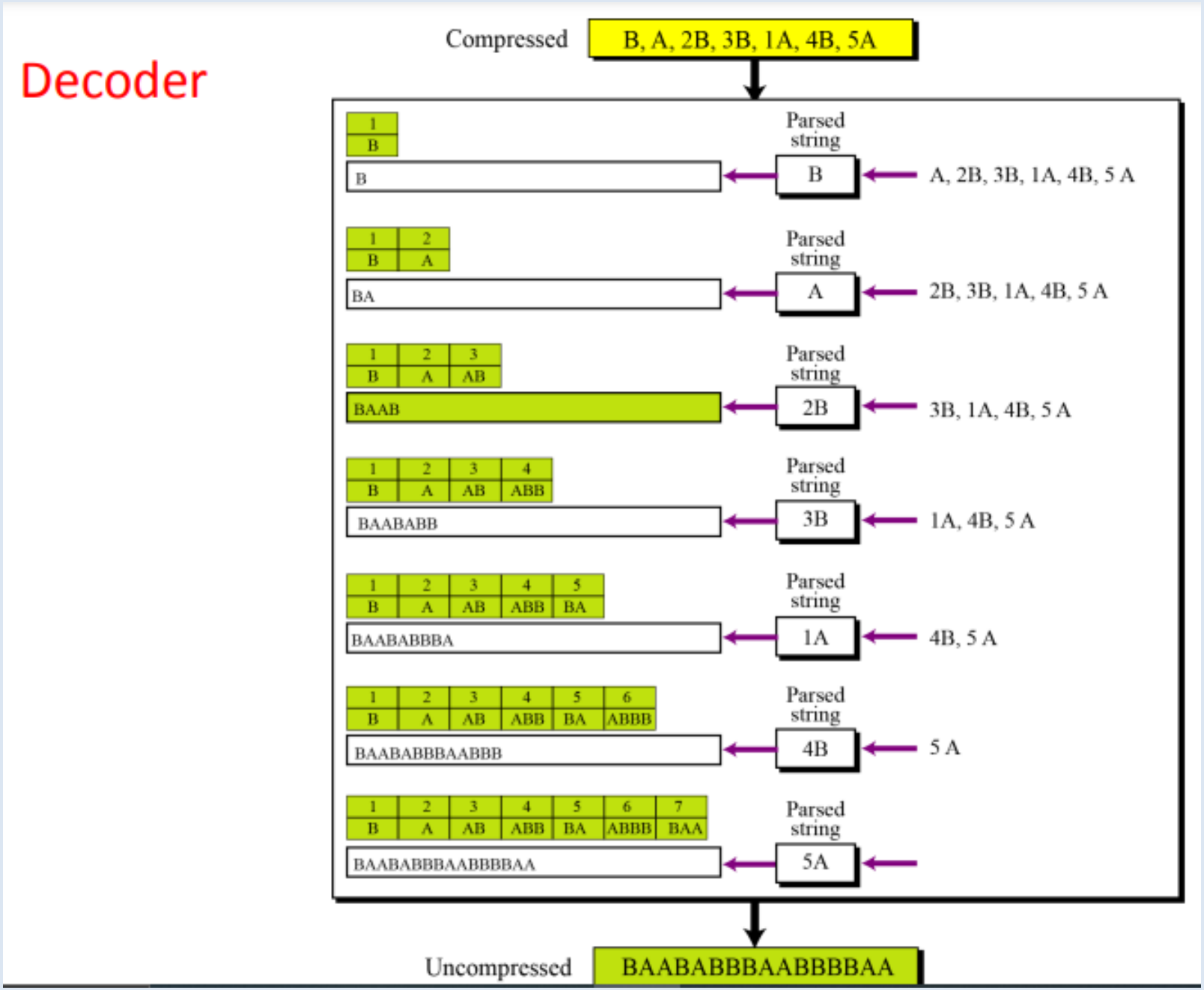
Binary symbols 0 and 1 are already stored.

B	0
A	1

Position	1	2	3	4	5	6	7
Subsequence	B	A	AB	ABB	BA	ABBB	BAA
Numerical Representation	B	A	2B	3B	1A	4B	5A
Binary Code	0	1	0100	0110	0011	1000	1011

(You can assume A: 0, B:1)

Decoder

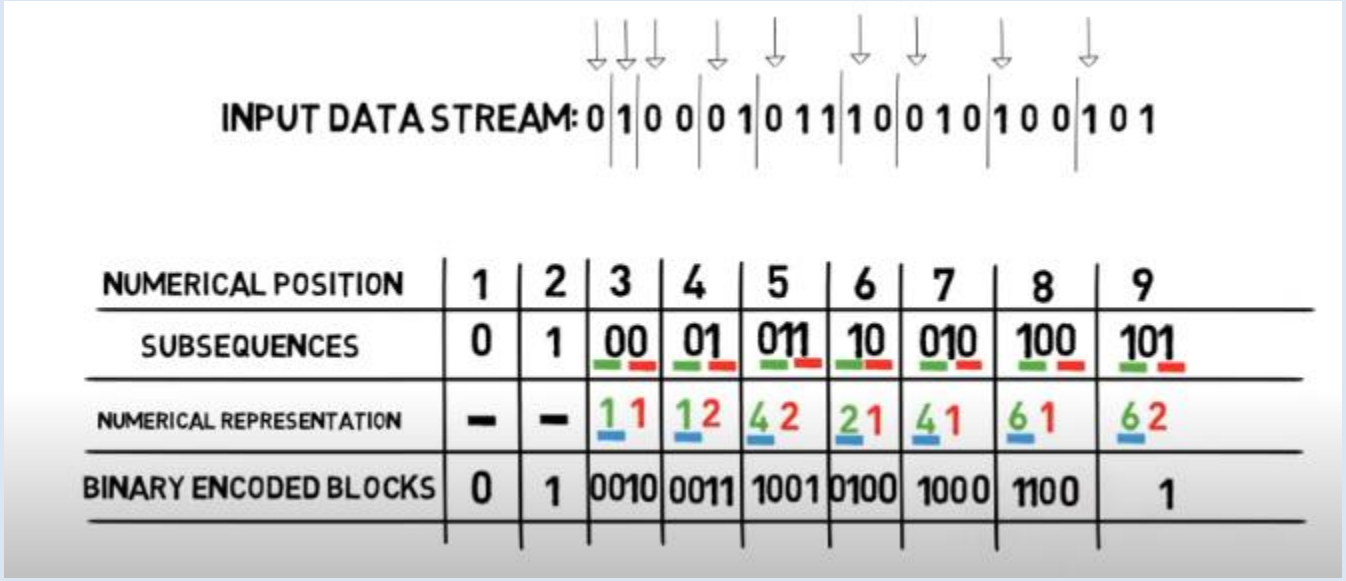

Compressed
B, A, 2B, 3B, 1A, 4B, 5A
1
B
B
Parsed string
B
A, 2B, 3B, 1A, 4B, 5 A
1 2
B A
BA
Parsed string
A
2B, 3B, 1A, 4B, 5 A
1 2 3
B A AB
BAAB
Parsed string
2B
3B, 1A, 4B, 5 A
1 2 3 4
B A AB ABB
BAABABB
Parsed string
3B
1A, 4B, 5 A
1 2 3 4 5
B A AB ABB BA
BAABABBBA
Parsed string
1A
4B, 5 A
1 2 3 4 5 6
B A AB ABB BA ABBB
BAABABBBAABBB
Parsed string
4B
5 A
1 2 3 4 5 6 7
B A AB ABB BA ABBB BAA
BAABABBBAABBBBAA
Parsed string
5A
Uncompressed
BAABABBBAABBBBAA

Homework

Q. Compress AABABBBABAABABBBABBABB by using LZ algorithm.

Ans: 0 0011 0101 1 0100 1011 1001 0110 0111 (Assuming A:0,B:1)

Q. Compress 010001011100101001 01 using LZ algorithm.



NUMERICAL POSITION	1	2	3	4	5	6	7	8	9
SUBSEQUENCES	0	1	00	01	011	10	010	100	101
NUMERICAL REPRESENTATION	–	–	1 1	1 2	4 2	2 1	4 1	6 1	6 2
BINARY ENCODED BLOCKS	0	1	0010	0011	1001	0100	1000	1100	1

Lossy Compression

A compression method is lossy compression only if it is not possible to reconstruct the original exactly from the compressed version. There are some insignificant details that may get lost during the process of compression.

Approximate reconstruction may be good in terms of the compression-ratio but usually it often requires a trade-off between the visual quality and the computation complexity (i.e. speed).

Data such as multimedia images, video and audio are more easily compressed by lossy compression techniques.

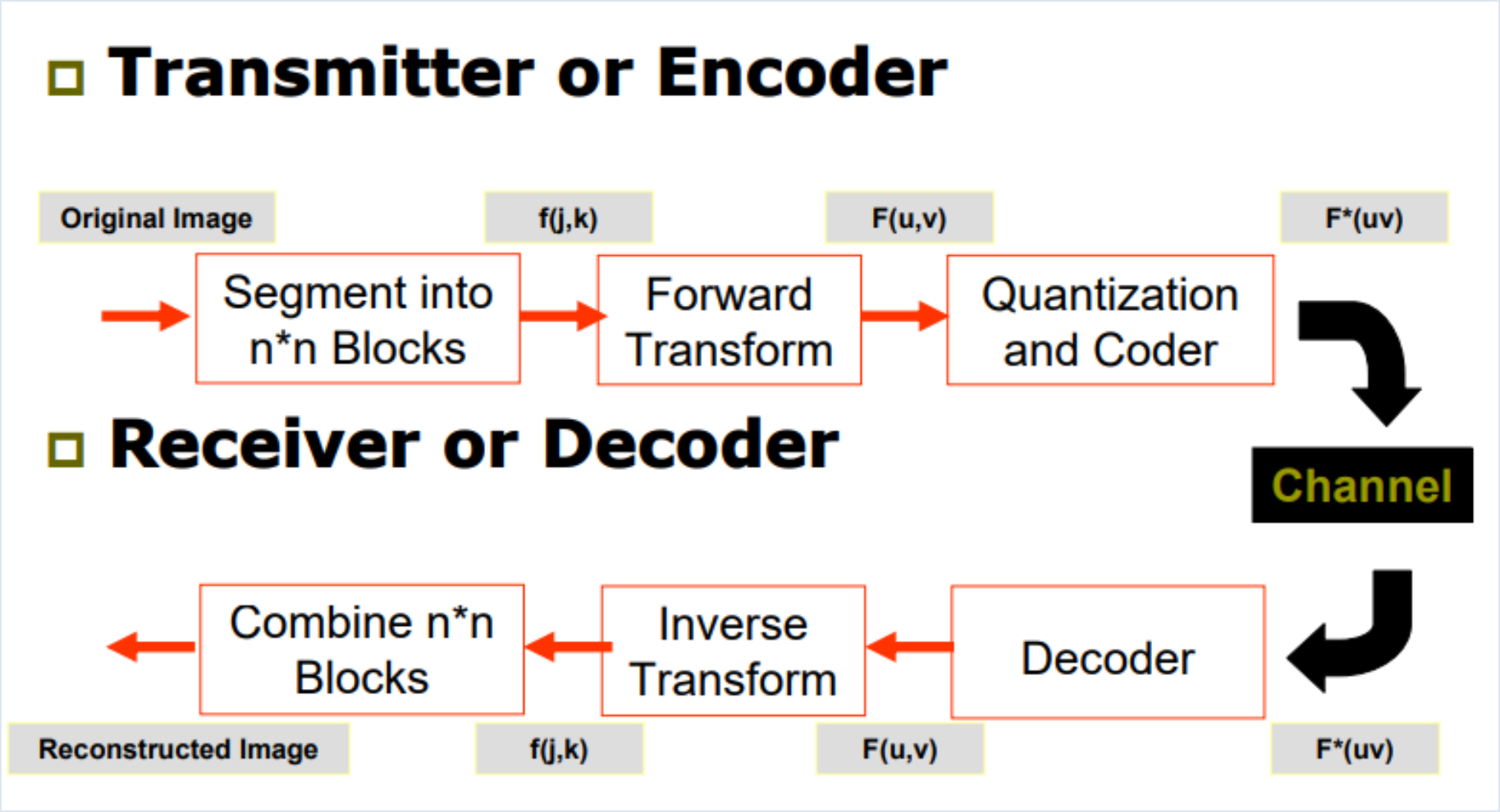
Types:

a) Transform coding
b) Predictive coding

Transform Coding

- Why transform Coding?
  - ❑ Purpose of transformation is to convert the data into a form where compression is easier.
- Transformation yields energy compaction (skewed probability distribution)
  - Facilitates reduction of irrelevant information.
- The transform coefficients can now be quantized according to their statistical properties.

Transform Coding Block Diagram



Lossy transform coding

Transform coding attempts to transform the input samples of the image from one domain to another.

Many grey scale patterns in a 2D-image transform into a much smaller number of output samples which have significant amplitude. The output samples with insignificant magnitude can then be discarded.

Spatial signal (image) <-> spatial frequency spectrum.

Low spatial frequency components imply slowly changing features in the image while sharp edges or boundaries in this image are represented by high spatial frequency components.

The 2-D discrete cosine transform (DCT) is used.

Lossy transform coding – Still images

The next stage in a transform-based compression algorithm is to quantise the output frequency samples.

The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation.

=> We can greatly reduce the amount of information in the high frequency components.

JPEG standard

JPEG (Joint Photographic Expert Group) is an international standard for the compression of single-frame monochrome and colour images.

JPEG is a transform-based standard basically relying on a 2-D DCT applied to blocks of 8*8 = 64 pixels yielding 64 output coefficients.

JPEG standard

The reduction of the amount of information at high spatial frequencies is done by dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer.

This is the main lossy operation in the whole process.

Typically, many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers.

"Lena" image

Original picture size = 12,249 bytes.

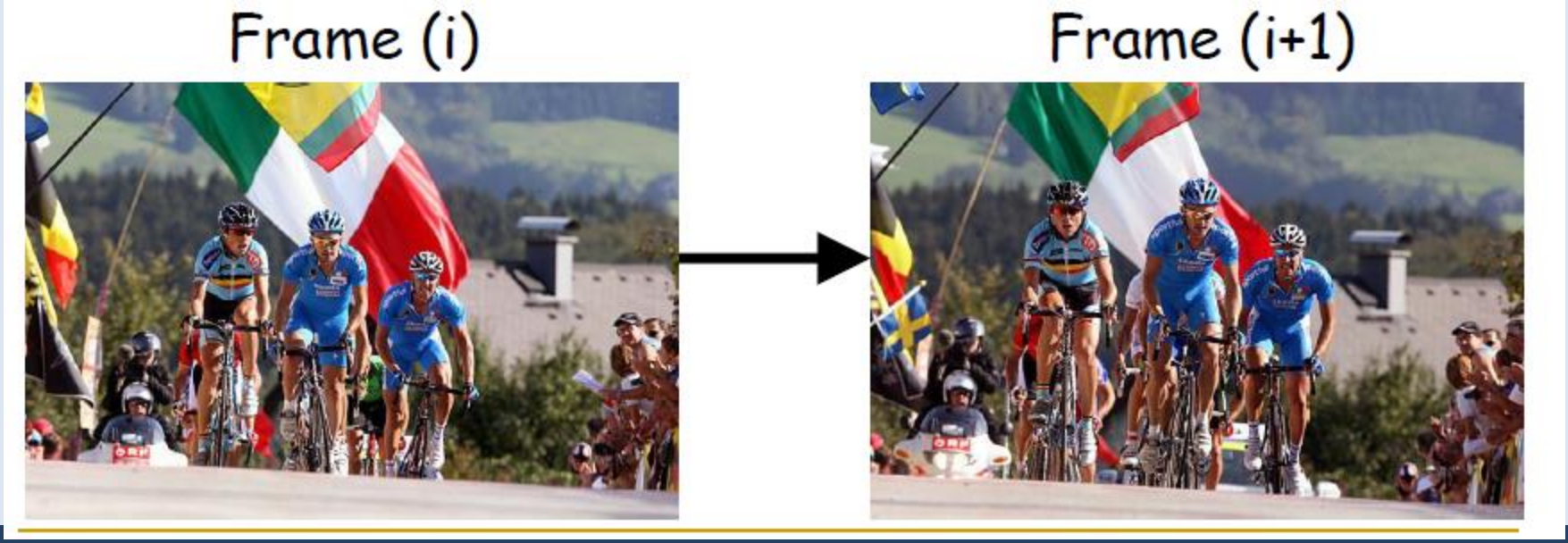
After JPEG compression from 12,249 to 1,869 bytes (≈ 6.5:1).

After JPEG compression from 12,249 to 559 bytes (≈ 22:1). The compression flaws are much more noticeable.

Lossy Predictive coding

- The next symbol can be statistically predicted from the past samples and quantize and code the error only.
- If the prediction error is typically small, then it can be represented with a lower average bit rate.

Consider the case of video data compression for which we require compression ratios ranging from 20:1 to 200:1



- Most changes between two successive frames are small -> Frame (i) can be used to predict frame (i+1).

- After transmitting/storing frame (i), we can just transmit/store the "difference" between frame (i) and frame (i+1).

- This is known as inter-frame compression. It can be combined with intra-frame compression (e.g., trans-form coding) to further increase compression ratios