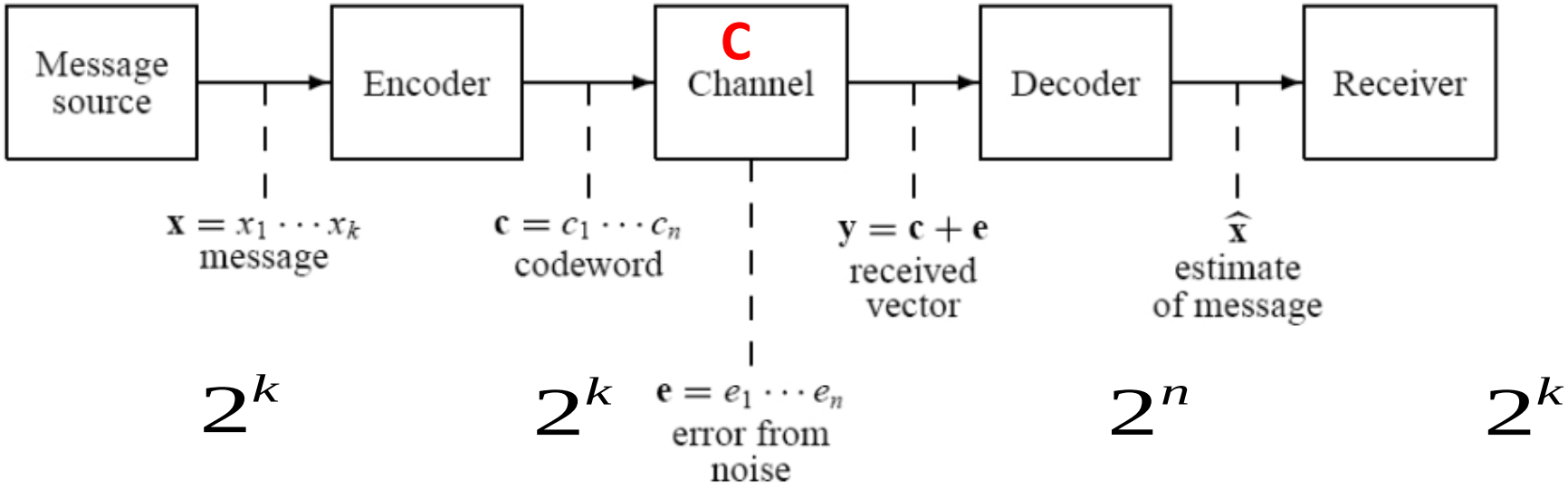
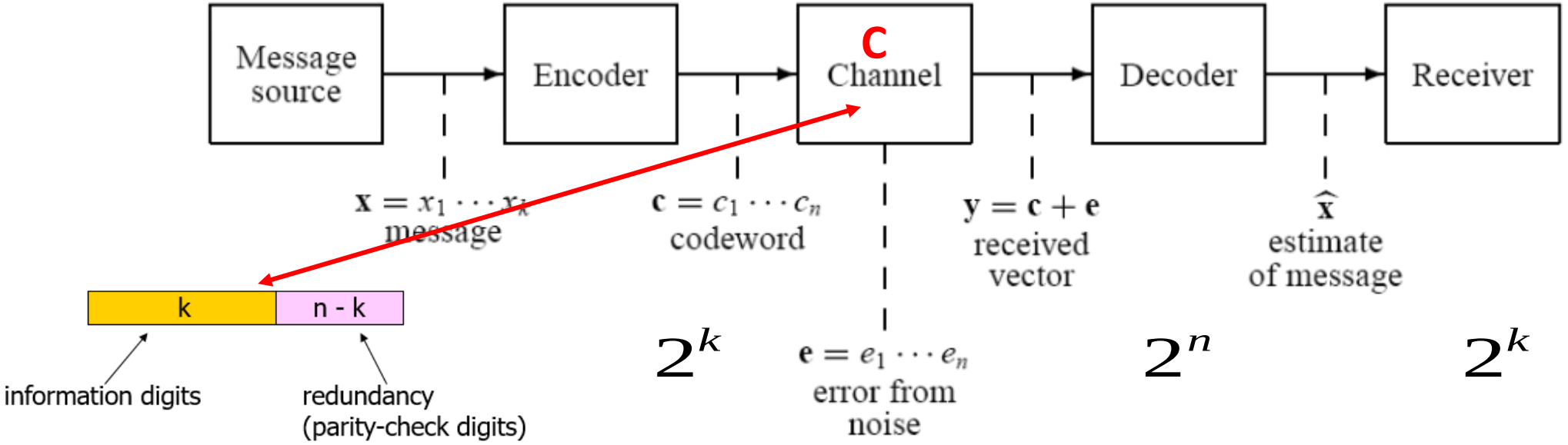
Channel Coding

Lecture 18



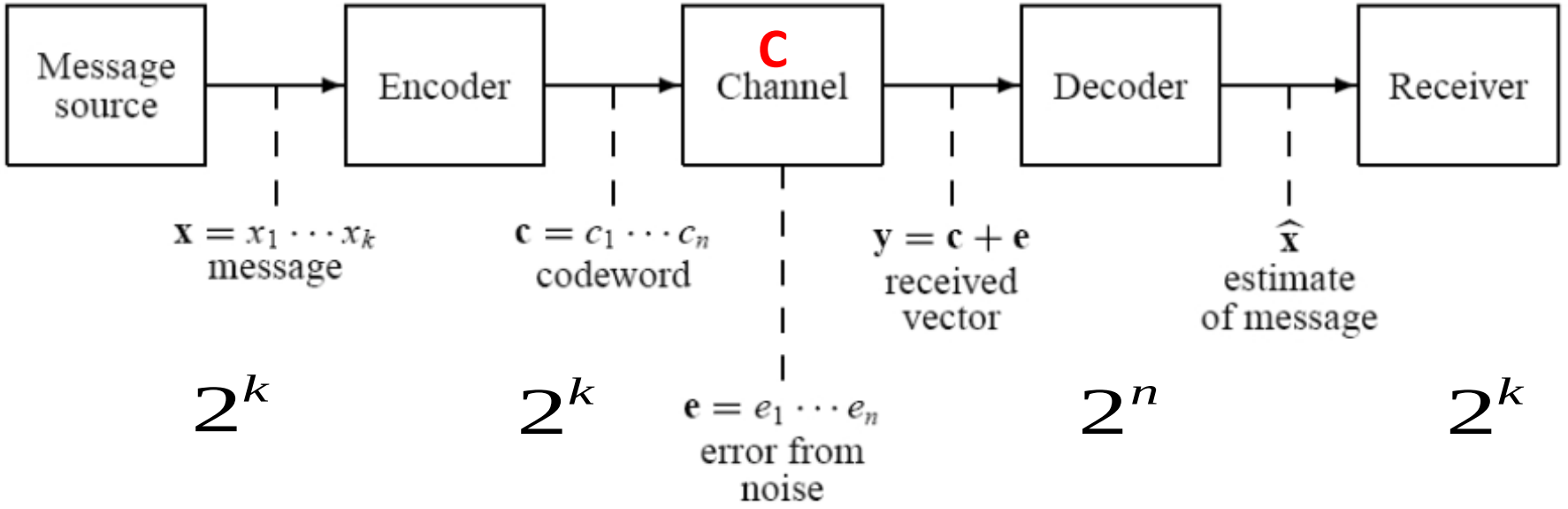
- The purpose of channel encoding is to reduce the probability of error when transmitting data over a noisy communication channel. It is a consequence of Shannon Channel Coding theorem that it it is possible to find an encoding scheme whereby an arbitrarily small error probability can be achieved as long as data is transmitted at a rate smaller than the channel capacity.
- It is performed by mapping the incoming data sequence into a channel input sequence and inverse mapping the channel output sequence into an output data sequence in such a way that the overall effect of channel noise on the system is minimized
- Error reduction is accomplished by introducing redundant bits (parity bits) in the channel encoder.
- The channel decoder uses the redundancy to decide which message bits were actually transmitted.
- We will study two types of channel codes in this series:
  - linear block codes (memoryless code)
  - convolutional codes (code with memory).

Linear Block Codes



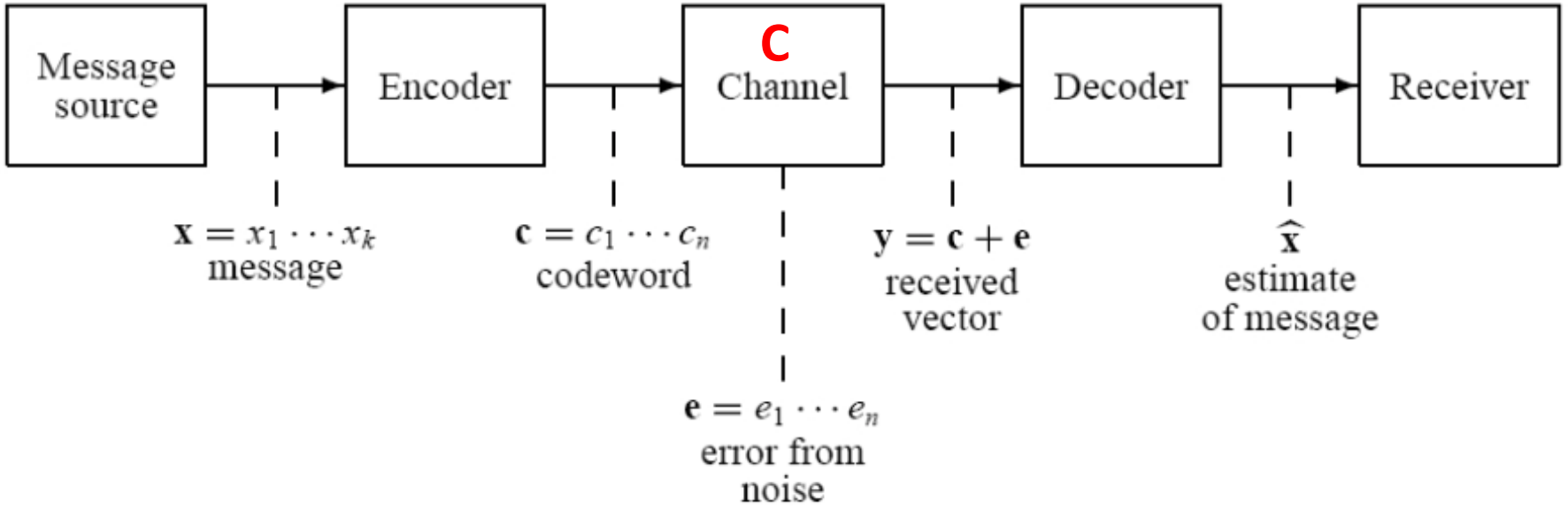
- **Notation on the (n, k) linear block code**. n is the number of bits in the codeword and k is the number of bits in the binary message
- This code can detect and correct errors.
- To generate an (n, k) block code, the channel encoder accepts information in successive k-bit blocks. Adds (n-k) redundant bits to each message block to produce an encoded block of n-bits called a code-word.
- The (n-k) redundant bits are algebraically related to the k message bits.
- 2^k **: Number of possible messages. Each message should have a channel codeword.**
- 2^k **: Number of possible codewords.**

Linear Block Codes



- Design Issue: How to select the 2^k codewords from the 2^n possible sequences?
- During transmission each bit in the codeword is subject to error. Hence there are 2^n possible received sequences.
- Decoding: Given $y \in 2^n$, decide which codeword was transmitted. Use the maximum likelihood rule (minimum distance rule)
- Design a code that can correct up to t errors. The design parameters are k, n, and t.

Linear Block Codes



- The channel encoder produces bits at a rate called the channel data rate, R_C
- Let R_s: rate in bits/sec at the channel encoder input. Then,
- $R_c = \left(\frac{n}{k}\right) R_s$ bits/sec is the bit rate at the channel encoder output.
- The ratio $\left(\frac{k}{n}\right)$ is called the Code Rate

One source bit (Ts)	One source bit (Ts)	One source bit (Ts)	One source bit (Ts)

k

One channel bit (Tc)	One channel bit (Tc)	One channel bit (Tc)	One channel bit (Tc)	One channel bit (Tc)	One channel bit (Tc)	One channel bit (Tc)

n

Binary Fields and Vectors

- Field: The binary alphabet **A={0,1} is properly referred to as a Galois field** with two elements denoted **GF(2).**
- Addition (XOR): 0 + 0 = 0, 0 + 1 = 1 + 0 = 1, 1 + 1 = 0
- Multiplication (AND): 0.1 = 0.0 = 1.0 = 0, 1.1 = 1
- This is also referred to as modulo-2 arithmetic (no carry)

- **Examples**
  - 01001 + 01110 = 00111 (bit by bit addition)
  - 10010 . 01110 = 00010 (bit by bit multiplication)
  - (1111+0011) . 0011 = 1100 . 0011 = 0000 (addition and multiplication)

Some Features of Linear Block Codes

- **Linearity**: A code is linear if **it contains the zero codeword** and **the Boolean sum of any two codeword must be another codeword**. As such, the set of codewords forms a vector space.
  - Advantage
    - simpler procedure for maximum likelihood detection
    - faster encoding and less storage space
    - error patterns are easily describable
- Maximum Likelihood Decoding reduces to Minimum Distance Decoding (if the a priori probabilities are equal P(0)=P(1)).
- Many decoding algorithms are based on minimum distance rule

Nonlinear Code: This is an example of a nonlinear code. Note that the sum of any two codewords does not yield another codeword. Also, the zero vector is not a codeword.

$$(0\ \ 0) \rightarrow (1\ \ 0\ \ 0\ \ 0)$$
$$(1\ \ 0) \rightarrow (0\ \ 1\ \ 0\ \ 0)$$
$$(0\ \ 1) \rightarrow (0\ \ 0\ \ 1\ \ 0)$$
$$(1\ \ 1) \rightarrow (0\ \ 0\ \ 0\ \ 1)$$

Hamming Weight and Hamming Distance

- **Def.: The Hamming weight, $W_H(C)$,** of codeword **C_j**, is the number of 1's in **C_j**
- **Def.: The Hamming distance** between two codewords C_1 and C_2, denoted by **$d_H[C_1,C_2]$**, is the number of components in which they differ.

$d_H[C_1,C_2]=W_H(C_1+C_2)$; modulo two addition

Def. : Minimum Distance d =min($d_H[C_i, C_j]$) , for all codewords, i # j

- $W_H(011)= 2$ $W_H(001)= 1$ $W_H(000)= 0$

  $d_H(011,000) = 2$ $d_H(011,111) = 1$ $d_H(011,101) = 2$

  Therefore 011 is *closer* to 111 in terms of the Hamming distance

- Consider the codewords(000, 101, 111, 010, 011)
  - You can easily verify that dmin=1.
  - Simple Procedure: Calculate the Hamming distance between each codeword and the zero vector. **min($d_H[C_i, C_j]$) is the smallest**

Motivation: Bit Error Probability and the Repetition Code

- Consider the binary phase shift keying scheme for transmitting bits 1 and 0:
- $s_1(t) = Acos(2\pi f_c t); s_0(t) = -Acos(2\pi f_c t);$
- For equally probable bits, the average bit probability of error is:

$$p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

$E_b = A^2 T_b / 2;$ average energy/bit

$N_0 =$ Noise power spectral density

This equation says that the bit error probability can only be reduced by increasing the symbol energy.

- **Question**: Is there another way of reducing the probability of error over the same noisy channel with a finite symbol energy?

Motivation: Repetition Code

Example: (3, 1) Binary Repetition Codes

0 (message) ==> 000 (codeword) 1 (message) ==> 111 (codeword)

The channel is independently used three times for each symbol.

Any one of the following eight sequences can be received:

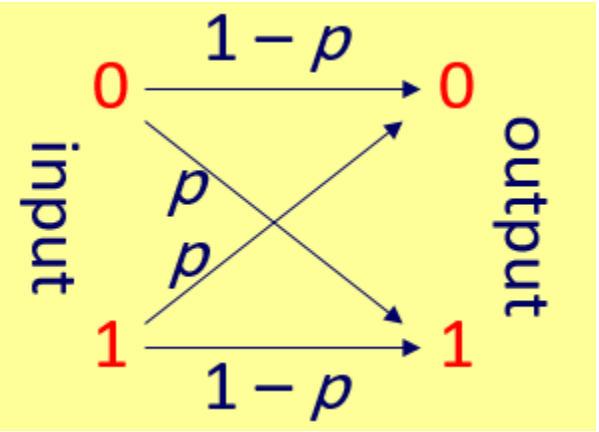
{000, 001, 010, 100, 110, 011, 101, 111}; Observation Space

To minimize the message (block) probability of error, the receiver decides 1 and 0 when any one of the following sequences are received (**MAJORITY RULE**)

S_0 : {000, 001, 010, 100 }: Decide **0**

S_1 :{ 110, 011, 101, 111}: Decide **1**

An error occurs when (**000**) is sent but any one of the sequences in S_1 **is received** and vice versa.



Repetition Code: Probability of Error After Decoding

Partitioning of the observation space; S_0 : {000, 001, 010, 100 }: Decide 0

Following the minimum distance rule; S_1: { 110, 011, 101, 111}: Decide 1

Let p be the crossover probability, which is the bit error probability when the channel is used once.

$$\text{Undetected block error probability} = P_u = \binom{3}{2}(1-p)p^2 + \binom{3}{3}p^3$$

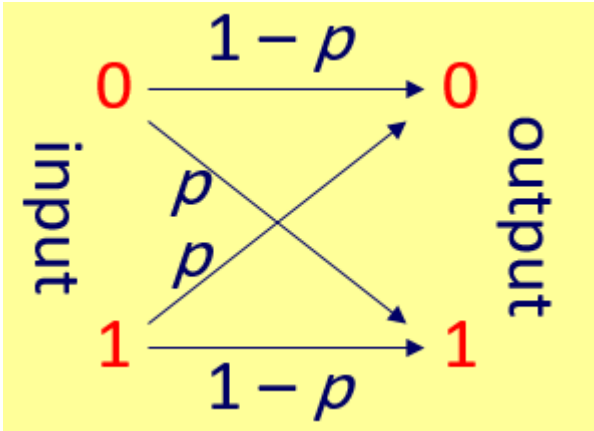
$$P_u = p^2(3-2p) << p$$

Gain: For a BSC with $p = 10^{-2}$, $P_u = 3\times10^{-4}$. (In this example, the block error = bit error)

Reason for gain: **(3,1) repetition code can correct single errors.**

Cost: Expansion in bandwidth or smaller code rate.

Code Rate: 1/3 = (k/n) = [≠ of message bits/≠ of transmitted bits]



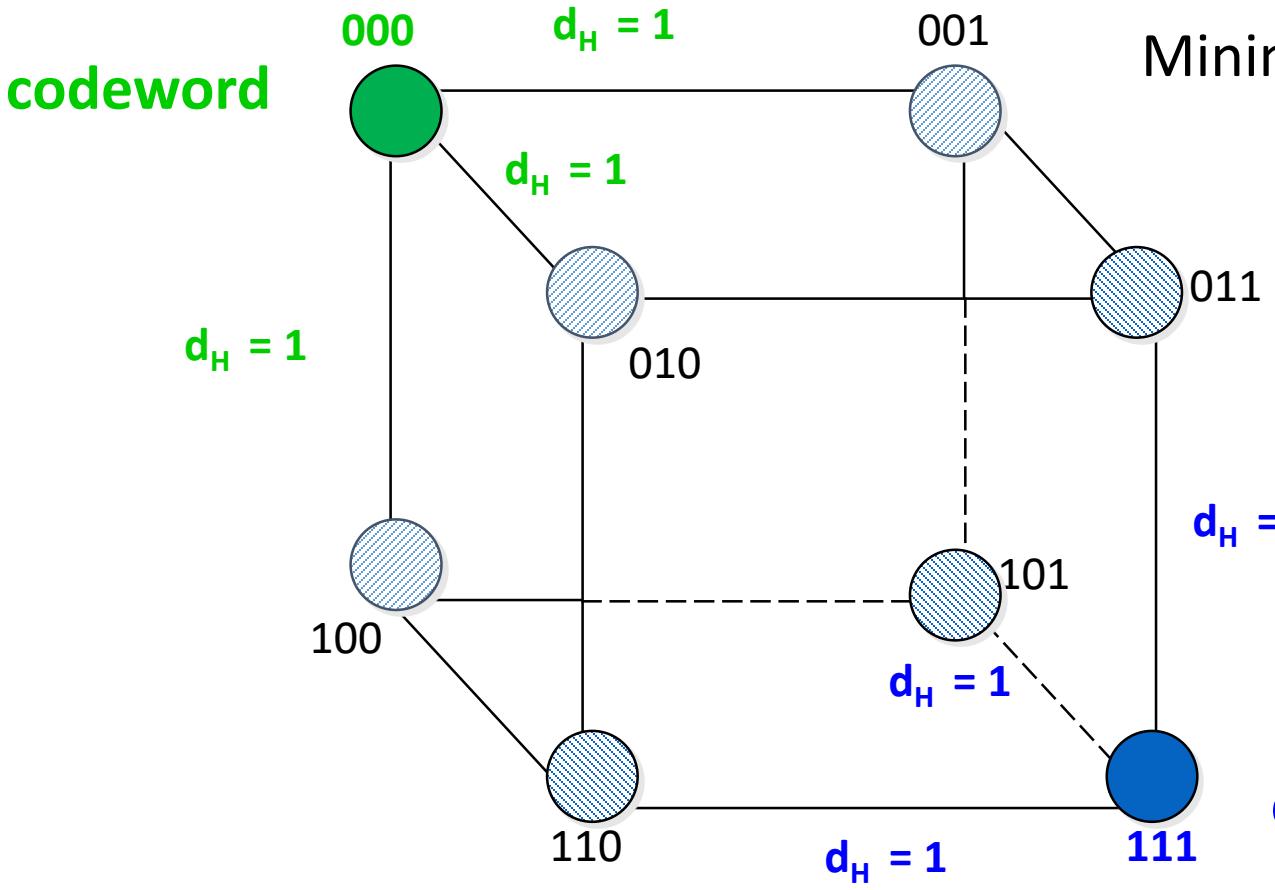
Exercise: Find the probability of error when a (5, 1) repetition code is used. How many errors can this code correct?

Exercise: Find the probability of error when a (7, 1) repetition code is used. How many errors can this code correct?

(3, 1) Repetition Code: Geometrical Illustration

Set S_0 : {000, 001, 010, 100 }:

Decoding Rule:
Minimum Hamming Distance



S_1: { 110, 011, 101, 111}:

Minimum Distance of a Code

- Consider the code C= $\{C_1\ C_2\ \ldots\ C_M\}$; $M = 2^k$
- **Def.**: The minimum distance of a code *C* is the minimum Hamming distance between any two different codewords.

$$d_{\min} = \min_{i \neq j} d(C_i, C_j) \quad \forall \ C_i \text{ and } C_j \text{ in } C$$

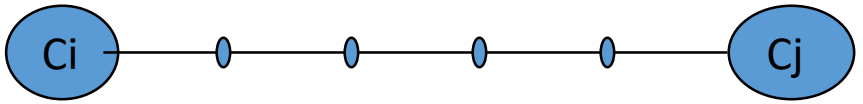
Theorem

A code with minimum distance d_{min} can detect (d_{min} -1) error bits

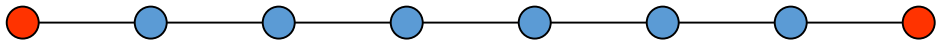
A code with minimum distance d_{min} can correct all error patterns up to and including *t*-error patterns, where

$$d_{min} = 2t + 1$$

$$t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor;$$



When, d_{min} =5, t=2.



When, d_{min} =7 , t=3.

- define $t_{max} = \lfloor \frac{d_{min}-1}{2} \rfloor$ ($\lfloor x \rfloor$is the largest integer ≤ *x*)

Example: The (6, 3) Linear Block Code

In this example, we compute the minimum distance and the error correcting capability of the (6, 3) linear code.

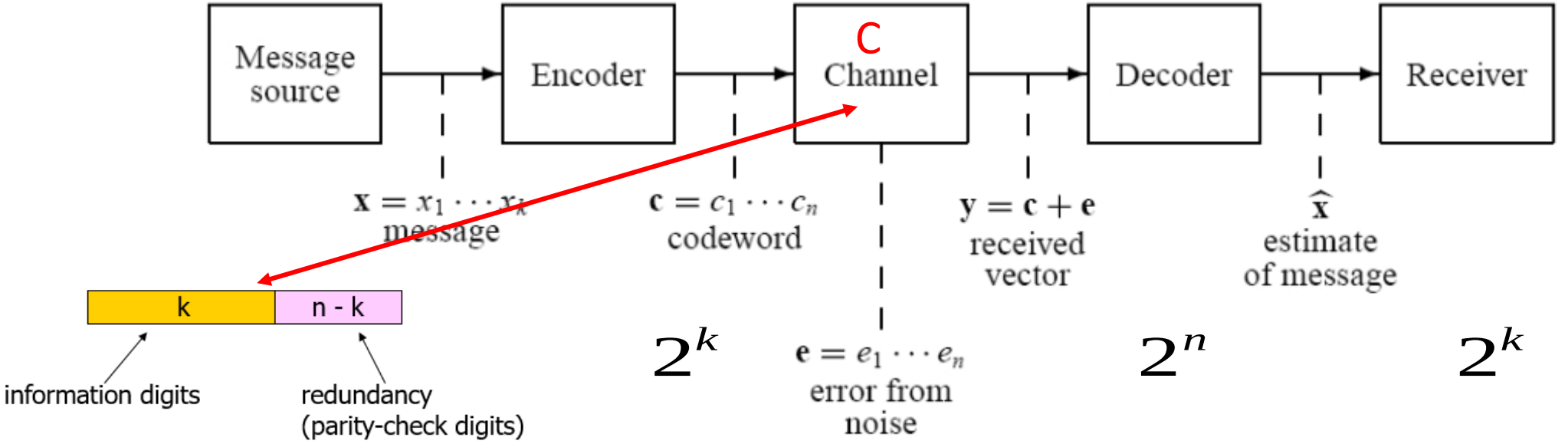
message	codeword	Hamming weight
000	000000	0
100	110100	3
010	011010	3
110	101110	4
001	101001	3
101	011101	4
011	110011	4

The minimum distance of a code is the smallest Hamming weight of the code, other than the all-zero codeword C_0 .

$d_{min} = 3$

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor = \left\lfloor \frac{3 - 1}{2} \right\rfloor = 1$$

Linear Block Codes



Lecture 19

- **Main Results from previous video: Basics of Channel Coding and Block Coding: Part a.**
- To generate an (n, k) block code, the channel encoder accepts information in successive k-bit blocks. Adds (n-k) redundant bits to produce an encoded block of n-bits called a code-word.
- 2^k **: Number of possible messages. Each message should have a channel codeword.**
- 2^k **: Number of possible codewords.**
- **Design Issue: How to select the 2^k codewords from the 2^n possible sequences?**
- **Decoding: Given $y \in 2^n$, decide which codeword was transmitted. Use the maximum likelihood rule (minimum distance rule)**
- **Design a code that can correct up to t errors. The design parameters are k, n, and t.**

Minimum Distance of a Code

- Consider the code C= $\{C_1\ C_2\ \ldots\ C_M\}$; $M = 2^k$
- **Def.**: The minimum distance of a code *C* is the minimum Hamming distance between any two different codewords.

$$d_{\min} = \min_{i \neq j} d(C_i, C_j) \quad \forall \ C_i \text{ and } C_j \text{ in } C$$

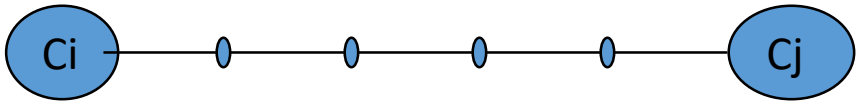
Theorem

A code with minimum distance d_{min} can detect (d_{min} -1) error bits

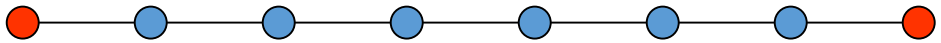
A code with minimum distance d_{min} can correct all error patterns up to and including *t*-error patterns, where

$$d_{min} = 2t + 1$$

$$t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor;$$



When, d_{min} =5, t=2.



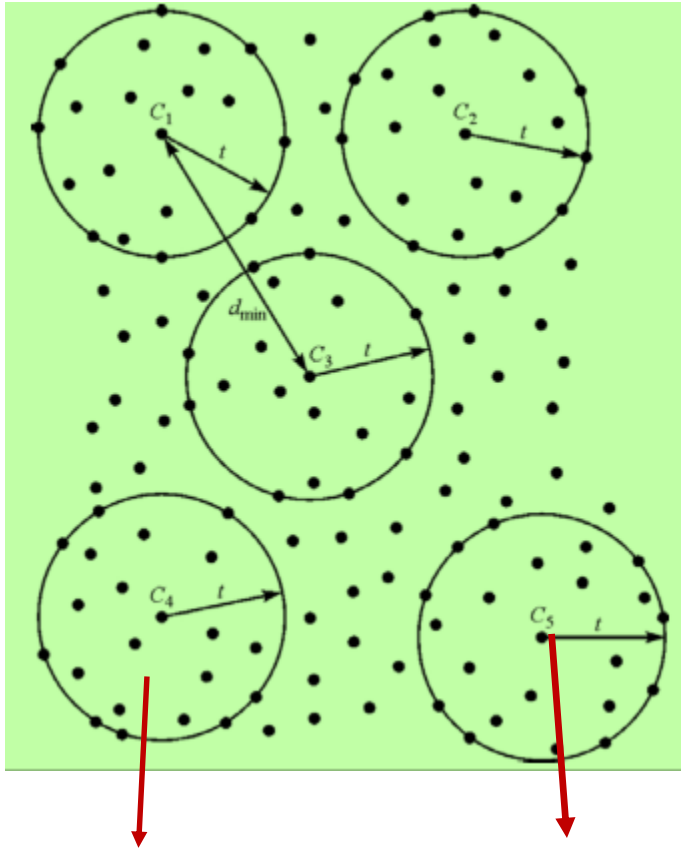
When, d_{min} =7 , t=3.

- define $t_{max} = \lfloor \frac{d_{min}-1}{2} \rfloor$ ($\lfloor x \rfloor$is the largest integer ≤ *x*)

Hamming Bound

- **The Hamming bound provides a *necessary*, but not a *sufficient*, condition for the construction of an (*n*,*k*) *t*-error correcting code..**
- **For an (*n*,*k*) code, there are 2^k codewords and 2^n possible sequences to chose from.**
- **Think of the 2^k codewords as centers of spheres in an *n*-dimensional space.**
- **All sequences that differ from codeword C_i in *t* or less positions lie within the sphere S_i of center C_i and radius *t*.**
- **For the code to be *t*-error correcting, all spheres S_i , i =1,.., 2^k , must be non-overlapping.**

In total, there are 2^n possible sequences



2^{n-k} spheres

2^k codewords

Copied from *Proakis's Digital Communications*

Hamming Bound

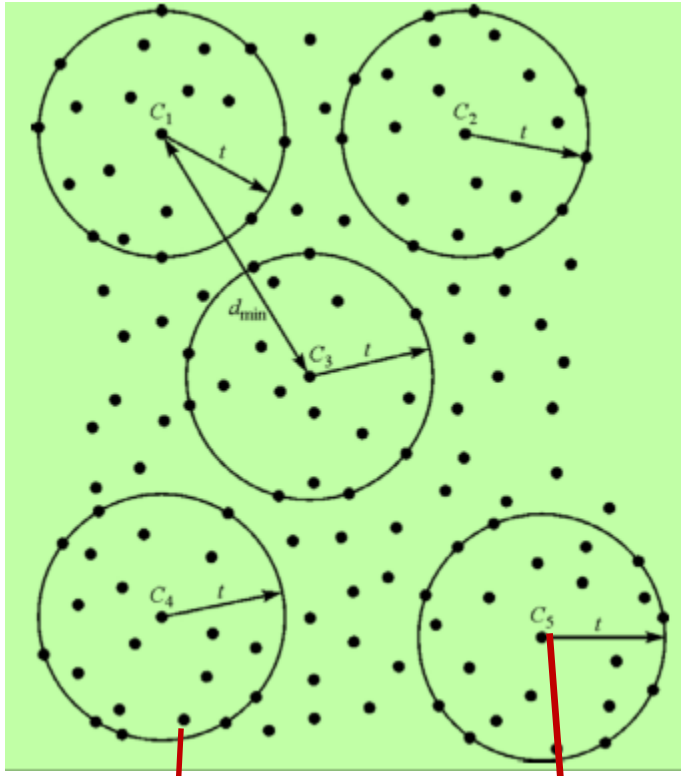
- When a codeword is selected, none of the *n*-bit sequences that differ from that codeword by *t* or less locations can be selected as a codeword.
- Consider the all-zero codeword C_0. The number of sequences that differ from this codeword by *j* locations is $\binom{n}{j}$
- The total number of sequences in any sphere (C_0 at the center plus all sequences that differ from C_0 by t or less digits) is
- $1 + \sum_{j=1}^{t} \binom{n}{j}$
- For a t bit error correcting code,
- $2^k\left(1 + \sum_{j=1}^{t} \binom{n}{j}\right) \leq 2^n$; When equality holds, code is called perfect.
- $(n-k) \geq log_2\left(1 + \sum_{j=1}^{t} \binom{n}{j}\right)$; Hamming Bound

0	0	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
.	.	.	.	.	.	.
1	0	0	0	1	0	0

$\binom{7}{2} = 21$, number of sequences that differ from 00000 by two bits

$\binom{7}{1} = 7$, number of sequences that differ from 00000 by two bits

In total, there are 2^n possible sequences



2^{n-k} spheres

2^k codewords

Copied from *Proakis's Digital Communications*

Hamming Bound: Example

- The above bound is known as the Hamming Bound. **It provides a *necessary*, but not a *sufficient*, condition for the construction of an (*n*,*k*) *t*-error correcting code.**
- Example: Is it theoretically possible to design a (10,7) single-error correcting code?
- **Condition:** $2^k\left(1+\sum_{j=1}^{t}\binom{n}{j}\right) \le 2^n$; $\left(1+\sum_{j=1}^{t}\binom{n}{j}\right) \le 2^{n-k}$ **; t = 1**

$$\binom{10}{0}+\binom{10}{1}=1+10=11>2^3, \text{ No, it is not possible}$$

$\binom{10}{0}+\binom{10}{1}$ Is it less than 2^3 = 8; Answer NO

Generation of a Linear Block Code

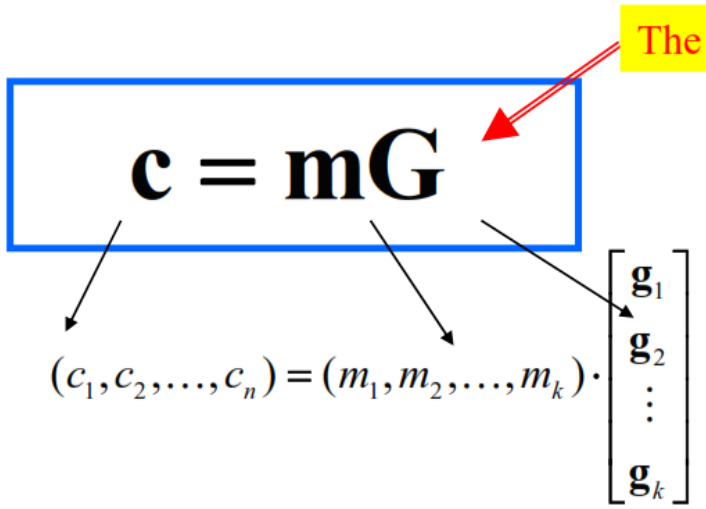
- Let **G** be the generator matrix for the linear code C. The rows of **G** are linearly independent and form the basis for the (n, k) code.
- If **m** is the k-bit message, then the codeword corresponding to **m** can be obtained as:
- $[c] = [m][G]$; **(1 x k)(k x n)**

- **Systematic (n, k) linear codes**: The first (or last) k bits are the information bits.

$$\mathbf{G} = \left[\mathbf{I}_k \vdots \mathbf{P}\right]$$

$$\mathbf{I}_k = k \times k \quad \text{identity matrix}$$

$$\mathbf{P}_k = k \times (n-k) \quad \text{matrix}$$



Generally, the rows of G, $\{\mathbf{g}_i\}$, are linearly independent. They form a basis for the (n, k) codes.

$$\mathbf{c} = (c_1, c_2, \ldots, c_n) = (\underbrace{m_1, m_2, \ldots, m_k}_{\text{message bits}}, \underbrace{p_1, p_2, \ldots, p_{n-k}}_{\text{parity check bits}})$$

Parity Check Matrix of a Linear Block Code

For any linear code we can find a matrix $\mathbf{H}_{(n-k)\times n}$, whose rows are orthogonal to rows of $\mathbf{G}$:

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}$$

$\mathbf{H}$ is called ***the parity check matrix*** and its rows are linearly independent.

The relation between any non-zero code word $\mathbf{c}$ and the parity check matrix is :

$$\mathbf{c}\mathbf{H}^T = \mathbf{m}\mathbf{G}\mathbf{H}^T = \mathbf{0}$$

Systematic linear block code:

$$\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}]; \qquad \mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_{n-k}] = [\mathbf{P}^T \mid \mathbf{I}_{n-k}]$$

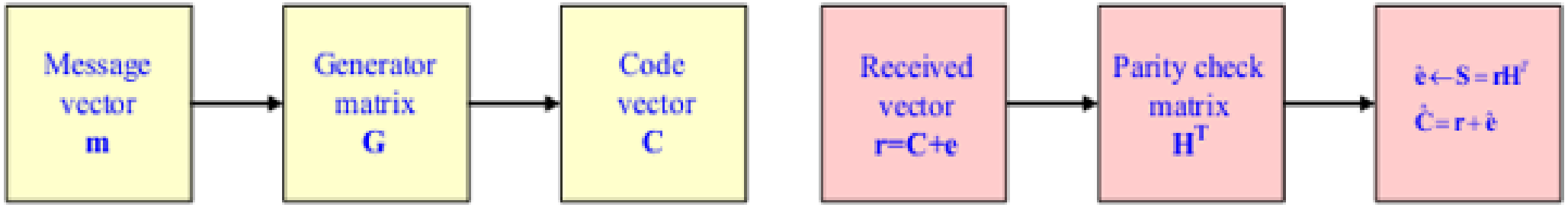
binary codes

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right] \qquad \mathbf{H} = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathbf{G}\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{0} \qquad H^T = \begin{bmatrix} P \\ I_{n-k} \end{bmatrix}$$

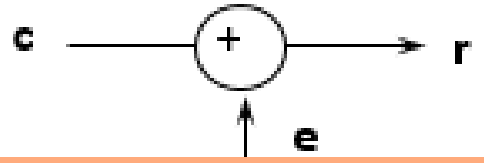
- **This equation forms the basis for demodulation.**
- **A codeword c always satisfies cH^T**
- **Hence, the receiver can use this fact for decoding and for error correction, as we shall see next.**

Syndrome Decoding



Error Pattern	Syndrome
(0000001)	$s = eH^T$
(0000010)	$s = eH^T$
(0000100)	$s = eH^T$

$$\mathbf{r} = \mathbf{C} + \mathbf{e}$$



$\mathbf{r} = (r_1, r_2, ..., r_n)$ received codeword or vector

$\mathbf{e} = (e_1, e_2, ..., e_n)$ error pattern or vector

- Syndrome testing:
  - **S** is syndrome of **r**, corresponding to an error pattern **e**.

$$\mathbf{S} = \mathbf{rH}^T = (\mathbf{C} + \mathbf{e})\mathbf{H}^T = \mathbf{eH}^T$$

How to decode?

1. Calculate $\mathbf{S} = \mathbf{rH}^T$
2. Find the error pattern , $\hat{\mathbf{e}} = \mathbf{e}_i$ corresponding to **S**, based on *syndrome table*.
3. Calculate $\hat{\mathbf{C}} = \mathbf{r} + \hat{\mathbf{e}}$ and the corresponding $\hat{\mathbf{m}}$.

– Note that $\hat{\mathbf{C}} = \mathbf{r} + \hat{\mathbf{e}} = (\mathbf{C} + \mathbf{e}) + \hat{\mathbf{e}} = \mathbf{C} + (\mathbf{e} + \hat{\mathbf{e}})$

- If $\hat{\mathbf{e}} = \mathbf{e}$, error is corrected.
- If $\hat{\mathbf{e}} \neq \mathbf{e}$, undetectable decoding error occurs.

Syndrome Decoding

- For the (7,4) Hamming code, the parity check matrix is

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

- Generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- The corresponding syndrome table is:

$$s = eH^T$$

s	e
000	0000000
001	0000001
010	0000010
100	0000100
111	0001000
110	0010000
101	0100000
011	1000000

- When a coded vector is received, the syndrome is calculated and any single error identified, and corrected by exchanging the relevant bit with the other binary value – However, problems can occur if there is more than one error.

Example: Syndrome Decoding

- Consider the (7,4) Hamming code. If the code vector 1000011 is sent while 1000001 and 1001100 are received, decode the information bits.
- Answer:
  - The first vector

    s=(1000001)H^T=(010) Calculate $S = rH^T$

    ⇒e=(0000010)

    ⇒x=(1000001)+(0000010)=(1000011)

    ⇒u=(1000) correct (as there is one error)

  - The second vector

    s=(1001100)H^T=(000) Calculate $S = rH^T$

    ⇒error-free

    ⇒x=(1001100)

    ⇒u=(1001) wrong (as there are 4 errors)

s	e
000	0000000
001	0000001
010	0000010
100	0000100
111	0001000
110	0010000
101	0100000
011	1000000

Hamming Codes

Hamming codes

- Hamming codes are a subclass of linear block codes and belong to the category of *perfect codes*.
- Hamming codes are expressed as a function of a single integer $m \geq 2$. (Number of parity bits)

For m=2, we have the (3, 1) repetition code.
For m=3, we have the (7, 4) code studies earlier).

Code length:	$n = 2^m - 1$
Number of information bits:	$k = 2^m - m - 1$
Number of parity bits:	$n - k = m$
Error correction capability:	$t = 1$
Minimum distance:	$d_{min} = 3$

(n, k) = (3, 1), (7,4), (15,11) (31,26),

m=2 Rate: k/n = 1/3

m=3 Rate: k/n = 4/7

m=4 Rate: k/n = 11/15

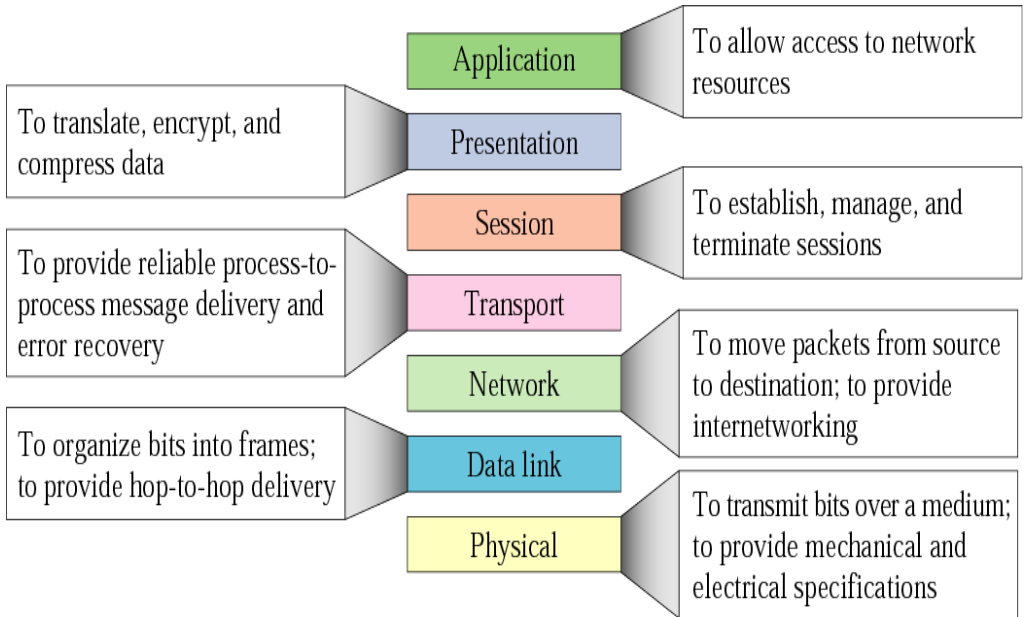
Note that while all codes correct 1 bit, the cost becomes less as m increases.

- The columns of the parity-check matrix, **H**, *consist of all non-zero binary m-tuples*.

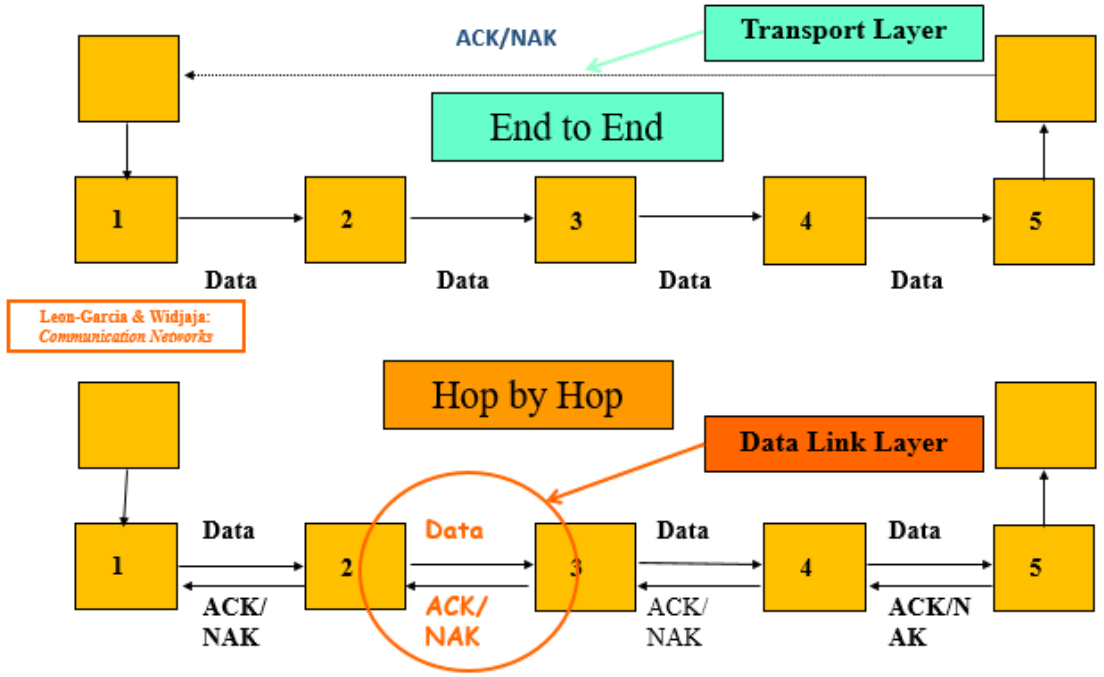
Error Control at the Data Link Layer: The Cyclic Redundancy Check Code

Lecture 20

- The data link layer is concerned with providing error free communication between adjacent nodes in a computer communication network.
- **Error detection and correction are implemented at the data link layer and the transport layer of the OSI model.**
- The main functions of the data link layer are:
  - Framing: breaking the information bit strings (Received from network layer) into smaller strings and encapsulates them into frames.
  - Error Control: making sure that the delivered frames are error free. We have either error correction or error detection or both. In this video, we will address the error detection problem.



- Framing: Data link layer breaks every packet received from the network layer into smaller units (frames). Adds a header and a trailer.
- The header contains the destination MAC address which is used to identify the intended receiver of the packet when the channel is a multiple access.
- The cyclic redundancy check bits (CRC), which are used for error detection, are provided in the trailer.



Data Link Layer: Framing

- The data link layer (DLL) on the receiving node is provided a sequence of bits from the **physical layer**
- It needs to determine where a frame begins and where it ends. It needs to identify the **header** and **trailer** before it can do subsequent processing.
- The **header** appears at the beginning of a frame. (easy to know)
- Identifying the **trailer** is more challenging because the length of the payload may be variable.

Two Possible Solutions

a. **Framing using payload length**: Include the length of the payload in the frame header

- **problem:** An error in the length field can cause some other bits to be interpreted as CRC bits affecting the current frame and subsequent ones even if they are error-free.

Header	Length	Payload	CRC

Data Link Layer: Framing

Framing using flag delimiters: Add flag bytes at the start and end of a frame, say the byte

0 1 1 1 1 1 1 0 (**0 + 6 1's in a row + 0**)

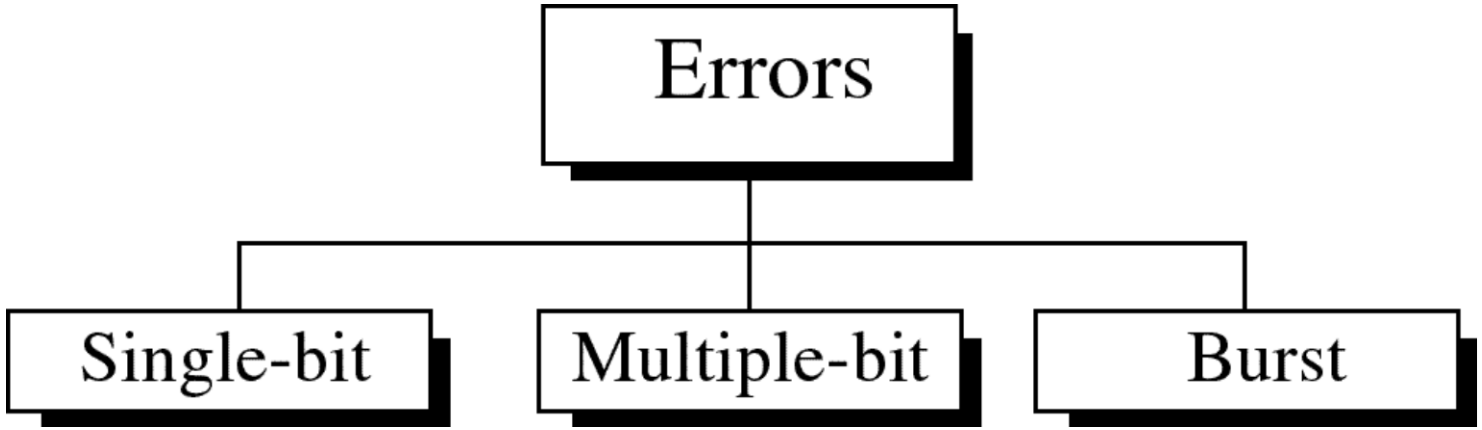
Problems:

- If the flag bytes are corrupted by channel errors, the frame cannot be correctly identified.
- The appearance of the flag byte in the payload can cause the frame boundaries to be erroneously identified
- To solve this problem, the DLL inserts a 0 bit whenever it encounters **five** consecutive 1's in the payload. This is called **bit stuffing .**

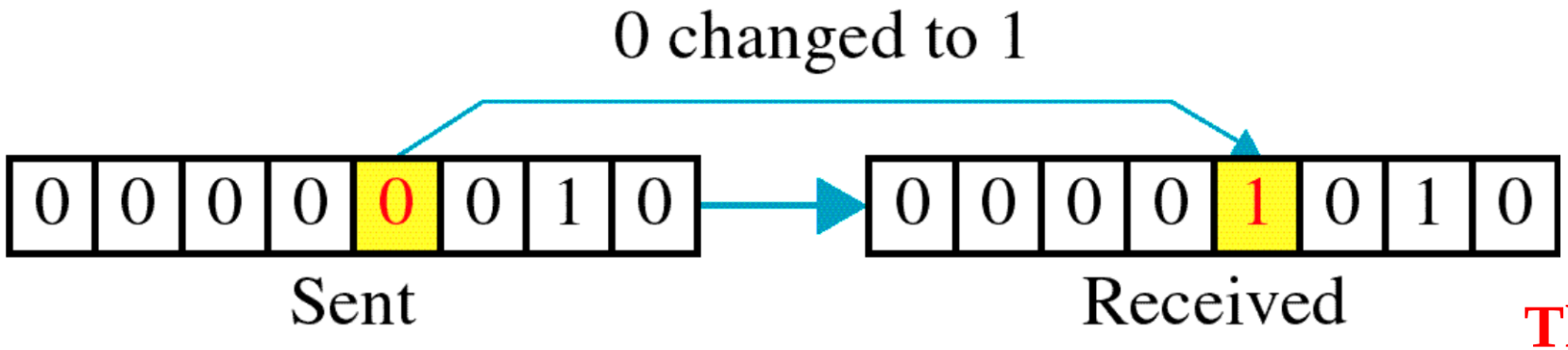
Example on Bit Stuffing

- Payload bits 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
- After bit stuffing 1 0 1 0 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 0 1 1 0
- After destuffing 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0

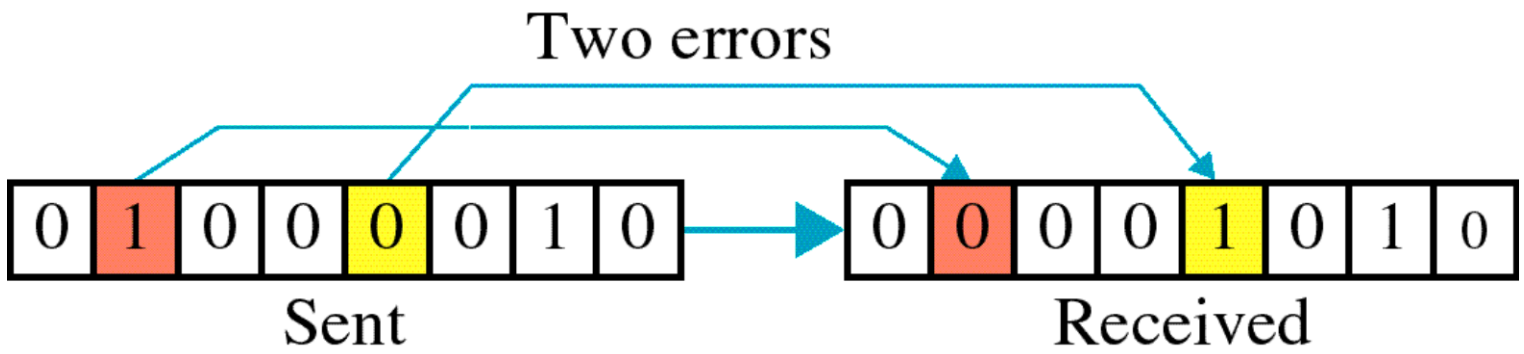
Data Link Layer: Error Detection



Single bit Error



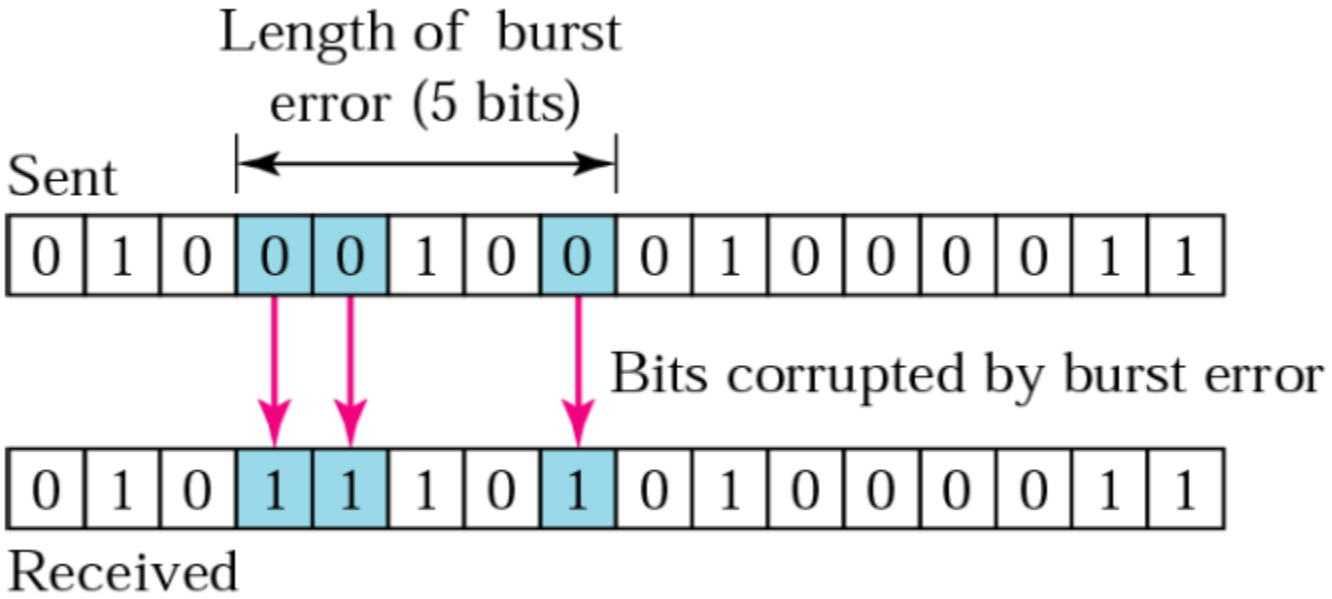
Double bit Errors



The location of errors in the frame is random

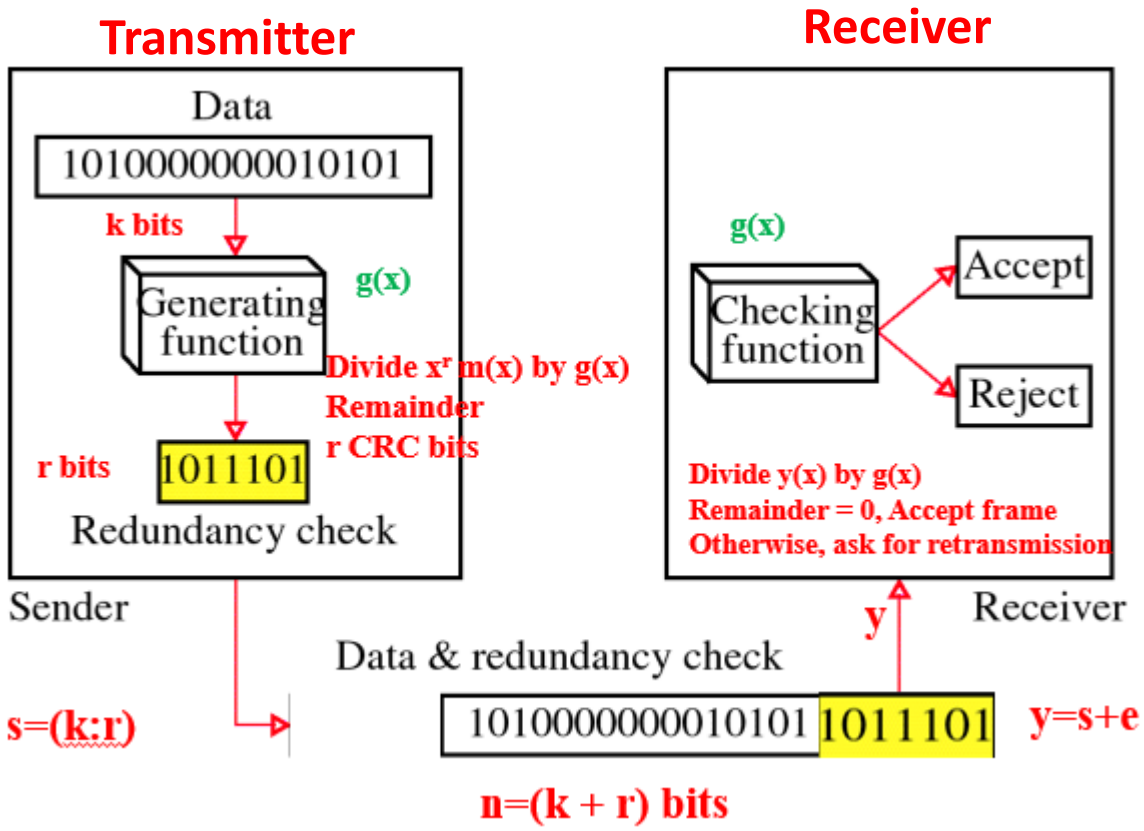
Burst Error

- An **error burst** of length r in a received frame is defined as a contiguous sequence of **r** bits in which the first and last bits or any number of intermediate bits are received in error.
- **Burst error does not necessarily mean that the errors occur in consecutive bits**, the length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.
- **Burst error is most likely to happen in serial transmission** since the duration of noise is normally longer than the duration of a bit.
- The number of bits affected depends on the data rate and duration of noise spike.



The Cyclic Redundancy Check (CRC) Code

- Error detection means to decide whether the received data is correct or not **without having a copy of the original message**.
- Error detection **uses the concept of redundancy, which means** adding extra bits for detecting errors at the destination.
- The cyclic redundancy check is a type of linear block code (based on the theory of cyclic codes) mainly used for frame error detection in the data link layer. If the frame is error-free, then it is accepted, otherwise a retransmission is requested.
- It is capable of detecting single, double, bust errors of length r, and many burst error patterns of length greater than r, where r is the order of the prime polynomial used for encoding and decoding.
- The systematic CRC: Here, the codeword consists of the message followed by the **error control bits**

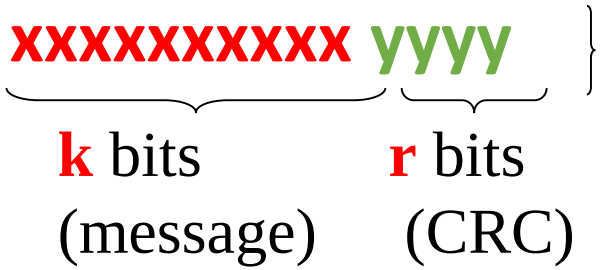


The Cyclic Redundancy Check (CRC) Code

- CRCs are popular because they are simple to implement in binary hardware easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.
- The CRC is widely used in many applications such as:
  - GSM Mobile Network (Global system for mobile communication)
  - CDMA2000 Mobile Networks (Code division multiple access)
  - Train Communication Networks
  - Bluetooth wireless connectivity network
  - WCDMA Mobile Networks (Wide band Code division multiple access)
  - Ethernet local area networks
  - Cellular area networks in vehicles

Cyclic Redundancy Check

- Given a **k**-bit frame or message, the transmitter generates an **r**-bit sequence, known as the **error control bits**, so that the resulting frame, consisting of (**k** + **r**) bits, is exactly divisible by some predetermined generator polynomial g(x) (the properties of this polynomial will be explored later)



Block of length **n = k + r**
(Transmitted Frame)

- **The receiver then divides the incoming frame by the same polynomial g(x) and, if there is no remainder, accepts the frame, otherwise requests a retransmission.**

Representing a Binary Sequence by a Polynomial

- **Binary Arithmetic**

$$0 + 0 = 0 \qquad 0 + 1 = 1$$

$$1 + 0 = 1 \qquad 1 + 1 = 0$$ **No Carry**

- **Polynomial Representation of a bit stream**

  1 1 0 1 = $1x^3 + 1x^2 + 0x + 1(x^0)$

  = x^3+x^2+1

- Addition: $(x^3 + x^2 + 1) + (x + 1) = x^3 + x^2 + x + (1 + 1)$

  $= x^3 + x^2 + x$

- Multiplication: $x (x^3 + x^2 + 1) = x^4 + x^3 + x + (0)x;$ **11010**
- Multiplication: $x^2 (x^3 + x^2 + 1) = x^5 + x^4 + x^2 + (0)x + 0(x^0)$; **110100**
- Multiplication: $x^3 (x^3 + x^2 + 1) = x^6 + x^5 + x^3 + (0)\, x^2 + (0)x + 0(x^0)$; **1101000**
- Multiplication by x implies a shift to the left by one digit
- Multiplication by x^2 implies a shift to the left by two digits.
- Multiplication by x^r implies a shift to the left by **r** digits.

Modulo Two Division

$$\frac{x^4 + x^2}{x} = x^3 + x \text{ with remainder } 0$$

$$\frac{x^4 + x^2 + 1}{x + 1} = x^3 + x^2 \text{ with remainder } 1$$

$$
\begin{array}{r|l}
 & x^3 + x^2 + 0x + 0 \\
\hline
x + 1 & x^4 + 0x^3 + x^2 + 0x + 1 \\
 & \underline{x^4 + x^3} \\
 & \quad x^3 + x^2 \\
 & \quad \underline{x^3 + x^2} \\
 & \qquad\quad 0x^2 + 0x \\
 & \qquad\quad \underline{0x + 1} \\
 & \qquad\quad\quad\ \ 1
\end{array}
$$

Remainder 1

Cyclic Redundancy Check

- Let m(x) be the **message polynomial**
- Let g(x) be the **generator polynomial of order r**
  - g(x) is fixed for a given CRC scheme
  - g(x) is known to both; the sender and receiver
  - The number of parity bits = r, the order of g(x)
- Create a block polynomial s(x) based on m(x) and g(x) such that s(x) is divisible by g(x). Here, $s(x) = x^r m(x) + c(x)$

$$\frac{s(x)}{g(x)} = q(x) + \frac{0}{g(x)};\ \text{Zero remainder}$$

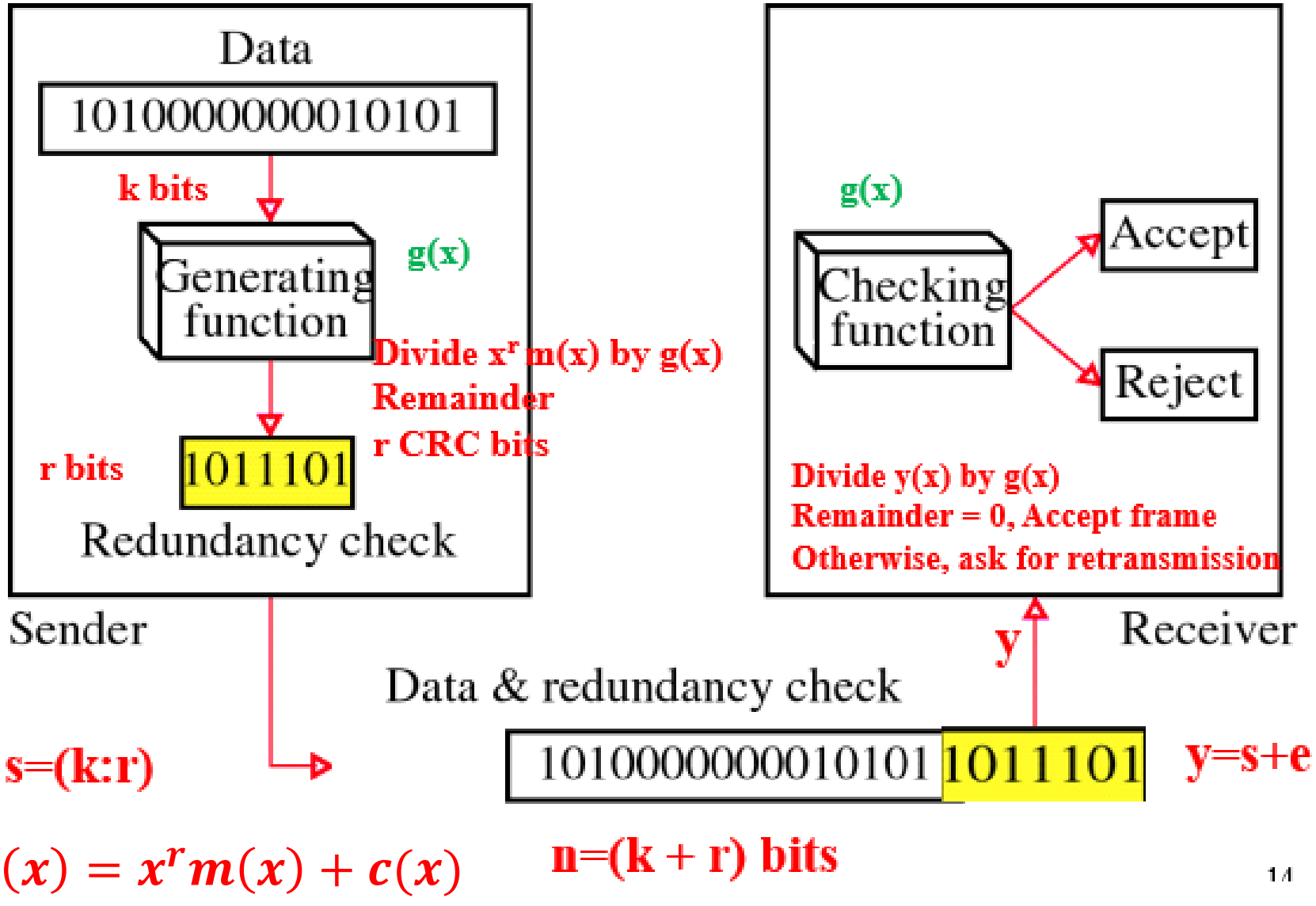
CRC Generation and Detection

CRC Sending Side

- Multiply m(x) by x^r
- Divide $x^r m(x)$ by g(x)
- Ignore the quotient and keep the remainder c(x)
- Form and send s(x) = $x^r m(x)+c(x)$

CRC Receiving Side

- Receive y(x) = s(x) + e(x); e(x) is some error pattern
- Divide y(x) by g(x)
- Accept if remainder is 0, reject otherwise



$$s(x) = x^r m(x) + c(x)$$

Example

Sender

- m(x) = 110011 → x^5+x^4+x+1 (k= 6 bits)
- g(x) = 11001 → x^4+x^3+1 (5 bits, r = 4)
  → 4 bits of redundancy
- Form $x^r m(x)$ → $x^9+x^8+x^5+x^4$
  → 1100110000
- Divide $x^r m(x)$ by g(x) to find remainder c(x)
- We can show that c(x)= x^3+1
- C =(1001); Has to be 4 bits = r
- Send the block S = 1100111001

Receiver

- Case 1: Assume no errors
- y= 1100111001
- y(x)= $x^9+x^8+x^5+x^4+x^3+1$
- Divide y(x) by g(x)
- Remainder = 0 (Verify)
- Accept Sequence, else, ask for retransmission
- Case 2: If y= 1010111001
- y(x)= $x^9+x^7+x^5+x^4+x^3+1$
- find the remainder, if any, at the receiver

CRC Generation and Detection

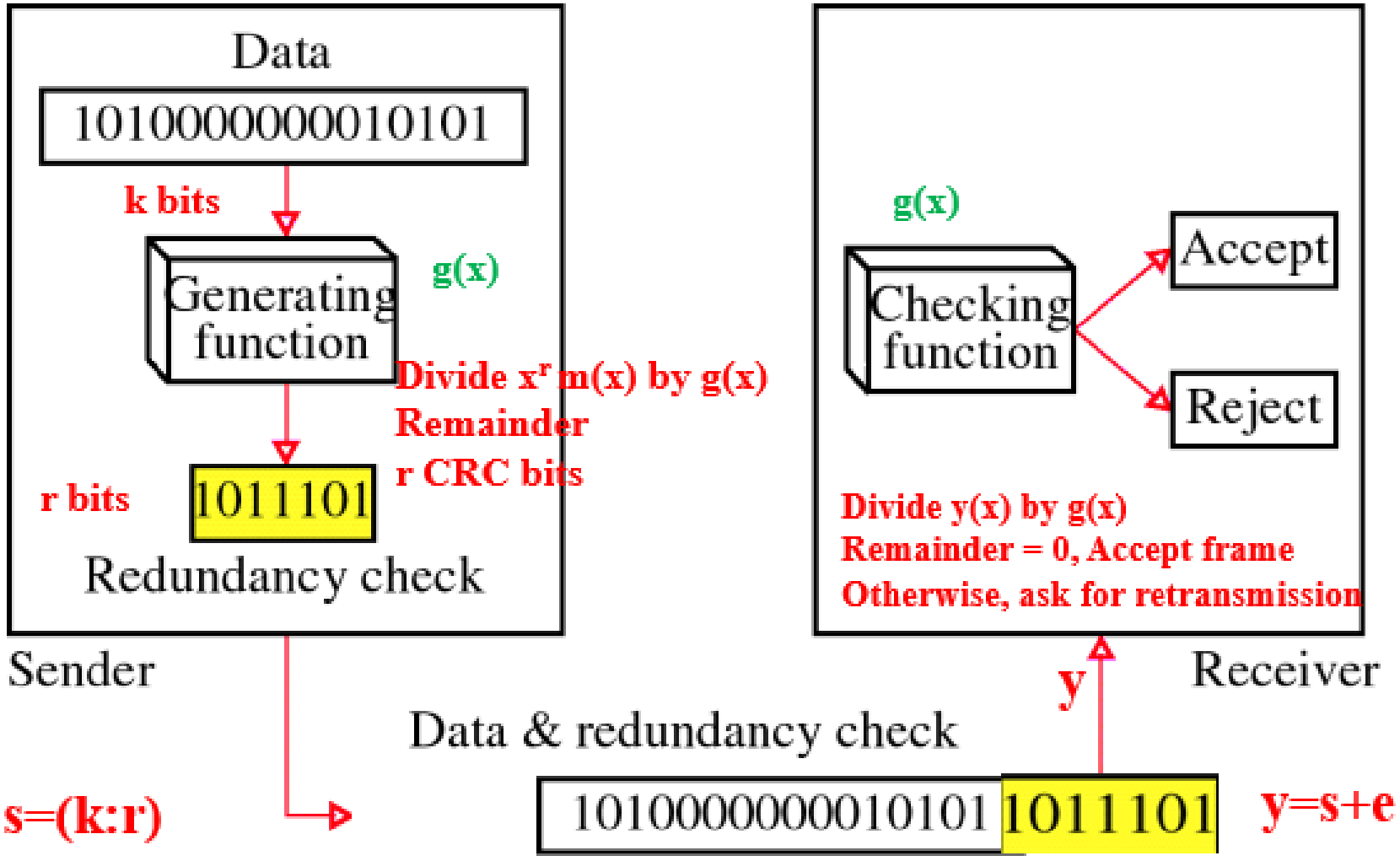
Generator polynomial g(x) is available at both sending and receiving ends.

CRC Sending Side

- Multiply m(x) by x^r
- Divide $x^r m(x)$ by g(x)
- Ignore the quotient and keep the remainder c(x)
- Form and send
- $s(x) = x^r m(x) + c(x)$

CRC Receiving Side

- Receive y(x) = s(x) + e(x); e(x) is some error pattern
- Divide y(x) by g(x)
- Accept if remainder is 0, reject otherwise



$$s(x) = x^r m(x) + c(x)$$

$$\text{Remainder } \frac{s(x)}{g(x)} = 0$$

Proof of CRC Generation

Prove that $s(x) = x^r m(x) + c(x)$ is divisible by $g(x)$, i.e.,

Prove that remainder $(s(x) / g(x)) = 0$

$$\frac{x^r m(x)}{g(x)} = q(x) + \frac{c(x)}{g(x)};$$ c(x): remainder. Transmitter side

$$x^r m(x) = g(x)q(x) + c(x)$$

Add c(x) to both sides and divide by g(x)

$$\frac{x^r m(x) + c(x)}{g(x)} = \frac{g(x)q(x)}{g(x)} + \frac{c(x) + c(x)}{g(x)} \Rightarrow remainder = 0$$

Remainder 0 Remainder 0

Note: Binary modular addition is equivalent to binary modular subtraction → c(x) + c(x) = 0

The Prime Polynomial

A polynomial g(x) of degree **r** is said to be primitive (prime) if the **smallest** value of m for which it divides (x^m+1) is $\boldsymbol{m = 2^r - 1}$

That is: $\text{remainder}\left(\frac{x^m+1}{g(x)}\right) = 0 \; for \;\; m = 2^r - 1 \; and \;\; \neq 0 \; for \;\; m < 2^r - 1$

Example: Is the polynomial $g(x) = x^2+x+1$ primitive?

Solution: Here, r =2. For g(x) to be primitive, the smallest value of m = 2^2 -1=3. Yes, since

$$\left(\frac{x^3+1}{x^2+x+1}\right) = x+1; \text{ Remainder } = 0$$

$$\text{Remainder}: \left(\frac{x^2+1}{x^2+x+1}\right) \neq 0 \text{ and } \left(\frac{x+1}{x^2+x+1}\right) \neq 0$$

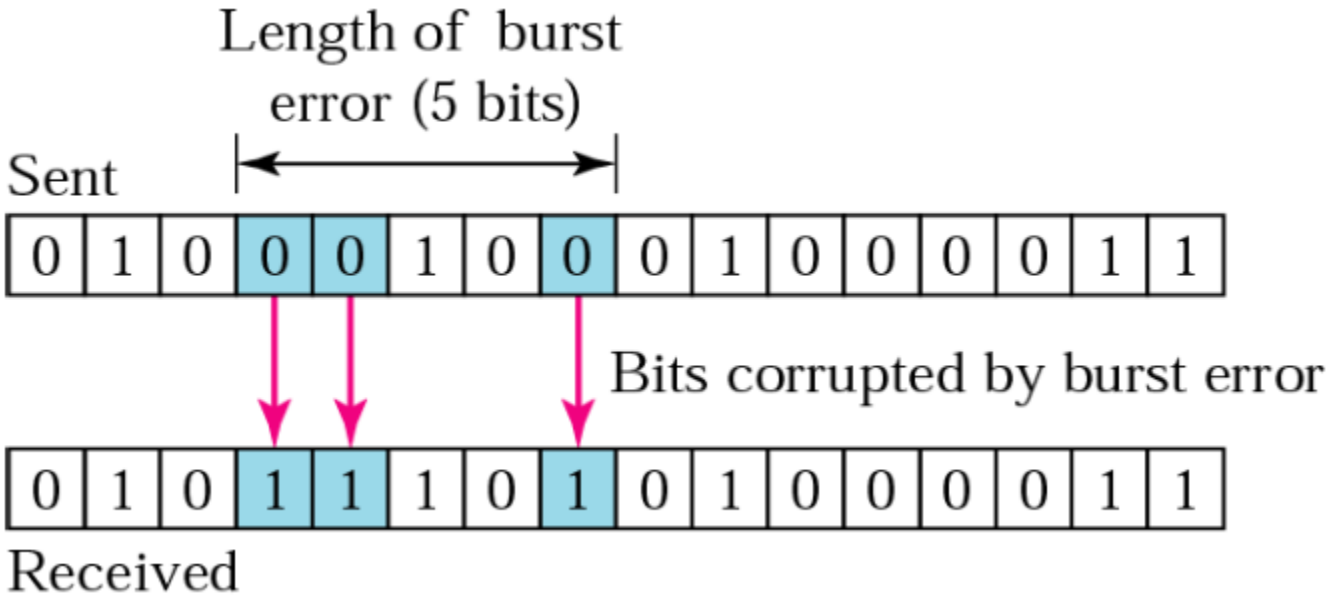
Remark:

If r =3 ($g(x)= x^3+x+1$), so that the maximum frame length n =7; r = 3, k =4.

If r =4 ($g(x)= x^4+x^3+1$), so that the maximum frame length $\boldsymbol{n = 2^4 - 1 = 15}$; r = 4, k =11

Burst Error

- An **error burst** of length r in a received frame is defined as a contiguous sequence of **r** bits in which the first and last bits or any number of intermediate bits are received in error.
- **Burst error does not necessarily mean that the errors occur in consecutive bits**, the length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.
- **Burst error is most likely to happen in serial transmission** since the duration of noise is normally longer than the duration of a bit.
- The number of bits affected depends on the data rate and duration of noise.



Desirable Properties of g(x) to Detect Errors

- Send s(x), receive, y(x) = s(x)+e(x), $\mathbf{rem}\left(\frac{s(x)+e(x)}{g(x)}\right) = \mathbf{rem}\left(\frac{e(x)}{g(x)}\right)$ $\boldsymbol{since}$ $\text{rem}\left(\frac{s(x)}{g(x)}\right) = \mathbf{0}$
- **Question:** When will CRC fail to catch an error?
  - An error will be detected if the error pattern e(X) is not divisible by g(x)
  - When e(x)/g(x) has no remainder, error will go undetected
  - When e(x)/g(x) has a remainder, error will be detected

Here are some properties that a g(x) should have in order to detect many error patterns (i.e., to ensure that there is a remainder in the division e(x)/g(x).

1. g(x) contains two or more terms. In particular, the high and low order coefficients must be 1.
2. g(x) is not divisible by x.
3. g(x) does not divide (x^m+1) for any m up to **m= 2^r -1, which is the maximum frame length**. A g(x) which satisfies this condition is a **primitive polynomial**.

Error Detection Capability of CRC

Here we briefly describe how these properties make it possible to detect single, double, and burst error patterns.

Single Bit Error → $e(x) = x^i$, i = 0, 1, …, $2^{(m-1)}$

Since g(x) has two or more terms, g(x) will not divide e(x). That is there is a remainder (remainder $x^i/g(x) \neq 0$).

Example: Let g(x)= x^3+x+1, Here r=3, maximum frame length n = $2^{(r-1)}$ =7 , so that the message length k=4. Here, find remainder $x^i/g(x)$

$$\text{Remainder}:\left(\frac{1}{x^3+x+1}\right)\neq 0 \quad \text{(i=0, error in first bit)}$$

$$\text{Remainder}:\left(\frac{x}{x^3+x+1}\right)\neq 0 \quad \text{(i=1, error in second bit)}$$

$$\text{Remainder}:\left(\frac{x^6}{x^3+x+1}\right)\neq 0 \quad \text{(i=6, error in seven'th bit)}$$

$$\#\text{ of single error patterns}=\binom{7}{1}=7$$

6	5	4	3	2	i=1	0

A typical received frame y = s + e with s single error bit

Error Detection Capability of CRC

Two Isolated Bit Errors (double errors)

$$e(x) = x^i + x^j, i > j \Longrightarrow e(x) = x^j(x^{i-j}+1)$$

Provided that g(x) is not divisible by x, a sufficient condition to detect all double errors is that g(x) does not divide (x^m+1) for any m up to $m = i - j$ (i.e., block length); This condition is met since g(x) is a primitive polynomial

$$\text{Remainder}: \left(\frac{x+1}{x^3+x+1}\right) \neq 0 \quad \text{(error in first and second bits)}$$

$$\text{Remainder}: \left(\frac{x^4+x}{x^3+x+1}\right) \neq 0 \quad \text{(error in second fifth bits)}$$

$$\text{Remainder}: \left(\frac{x^6+1}{x^3+x+1}\right) \neq 0 \quad \text{(error in first and seven'th bit).}$$

$$\text{\# of double error patterns} = \binom{7}{2} = 21$$

6	i=5	4	3	J=2	1	0

Error Detection Capability of CRC

Short Burst Errors

(Length b ≤ r, number of redundant bits); burst of length b starting at position i.

$$e(x) = x^i + x^{i+1} + \cdots + x^{i+b-1} = x^i(x^{b-1} + x + 1)$$

If g(x) has an x^0 (i.e. 1) term and b ≤ r, g(x) will not divide e(x) . The order of the factored polynomial in the numerator < order of g(x)

∴ **All burst errors up to length r are detected**

$$\text{Remainder}: \left(\frac{x^2+x+1}{x^3+x+1}\right) \neq 0$$ (error in bits 1, 2, 3), burst length=3

$$\text{Remainder}: \left(\frac{x^3+x^2+x}{x^3+x+1}\right) = \left(\frac{x(x^2+x+1)}{x^3+x+1}\right) \neq 0$$ (error in bits 2, 3, 4)

$$\text{Remainder}: \left(\frac{x^6+x^5+x^4}{x^3+x+1}\right) = \left(\frac{x^4(x^2+x+1)}{x^3+x+1}\right) =\neq 0$$ (error in bits 5, 6. 7)

Hence, all burst errors of size 3 are detected.

Note that power in numerator < power in denominator in all second terms

6	5	4	3	2	1	0

Error Detection Capability of CRC

Long Burst Errors (Length **b = r + 1**)

Only the error pattern that matches the coefficient of g(x) will be undetected. All other patterns will be detected. Undetectable only if

burst error pattern e(x) matches g(x)

$$g(x) = x^r + \cdots + x + 1;\ e(x) = x^r + \cdots + x + 1$$

Some longer error bursts can be detected

$$\text{Remainder}:\left(\frac{x^3+x+1}{x^3+x+1}\right)=0$$ (error in bits 1, 2, 4), burst length= 4 (undetected)

$$\text{Remainder}:\left(\frac{x^3+x^2+x+1}{x^3+x+1}\right)\#0$$ (error in bits 1, 2,3, 4), burst length= 4 (detected)

6	5	4	3	2	1	0

Detected burst of length 4

6	5	4	3	2	1	0

Undetected burst of length 4

Further Properties of CRC

Odd Number of Bit Errors

If x+1 is a factor of g(x), all odd number of bit errors are detected

Justification:

- In practice, the generator polynomial is chosen to be the product of a primitive polynomial of degree r − 1 and the polynomial x + 1.
- A polynomial e(x) is divisible by x + 1 if and only if it contains an even number of non-zero coefficients. This ensures that all error patterns with odd number of errors in the transmitted bit string are detected.

Example CRC-16 : Used in Ethernet (Number of check bits **r**= 16)

CRC-16: $g(x) = x^{16} + x^{15} + x^2 + 1$; generating polynomial

$$g(x) = (x+1)(x^{15} + x + 1)$$

g(x) is a product of (x+1) and a prime polynomial of order 15

This enables the receiver to catch **odd errors**, in addition to the ability of the prime polynomial to catch **single, double, and burst errors.**

Maximum frame length = $(2^{15}-1)$ = 32, 767 bits (determined by prime polynomial of order 15)

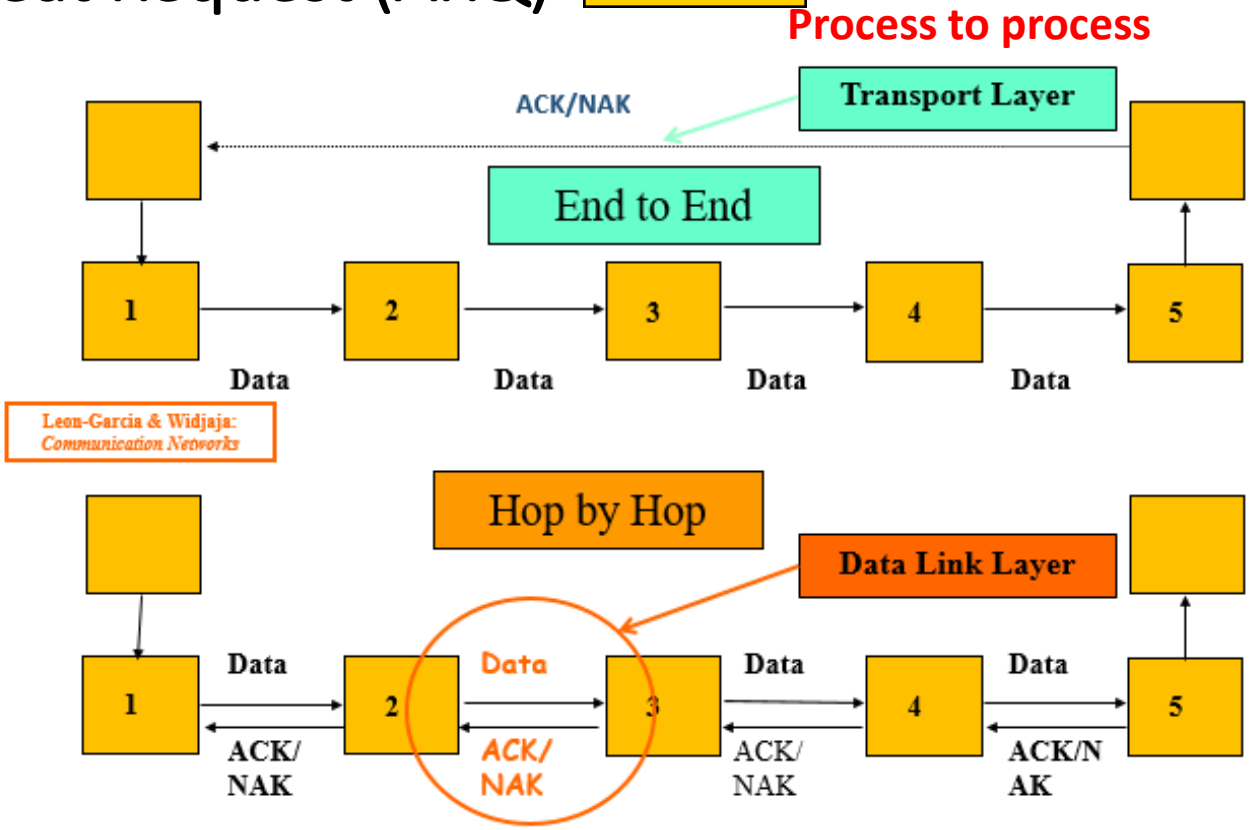
CRC-16 catches all

- Single, double, and odd number of bit errors
- Bursts of length 15 or less
- Many burst errors of length 16 bits
- Many burst errors of length 17 bits and longer.

Automatic Repeat Request (ARQ)

Lecture 22

- The idea behind ARQ is to initiate frame retransmission when errors are detected in the received frame.
- Retransmission are initiated using a combination of timeouts and acknowledgements.
- There are three types of ARQ protocols.
  a. Stop and Wait
  b. Go back N
  c. Selective repeat
- Here we will discuss in detail the first one only.



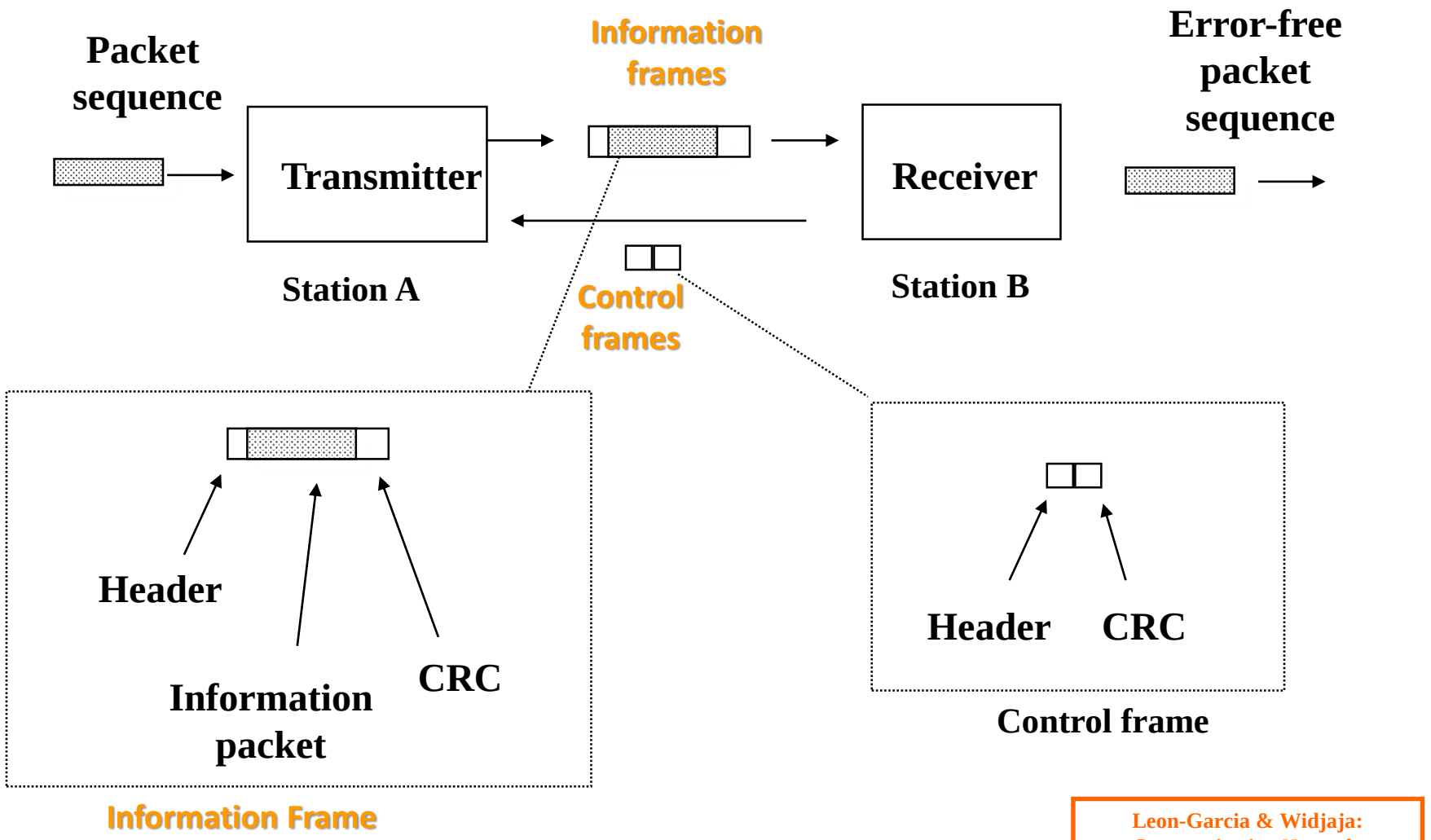
CRC Sending Side

Multiply m(x) by x^r
Divide $x^r m(x)$ by g(x)
Ignore the quotient and keep the remainder c(x)
Form and send
$s(x) = x^r m(x) + c(x)$

CRC Receiving Side

- **Receive y(x) = s(x) + e(x); e(x) is some error pattern**
- **Divide y(x) by g(x)**
- **Accept if remainder is 0, reject otherwise**

Basic Elements of ARQ



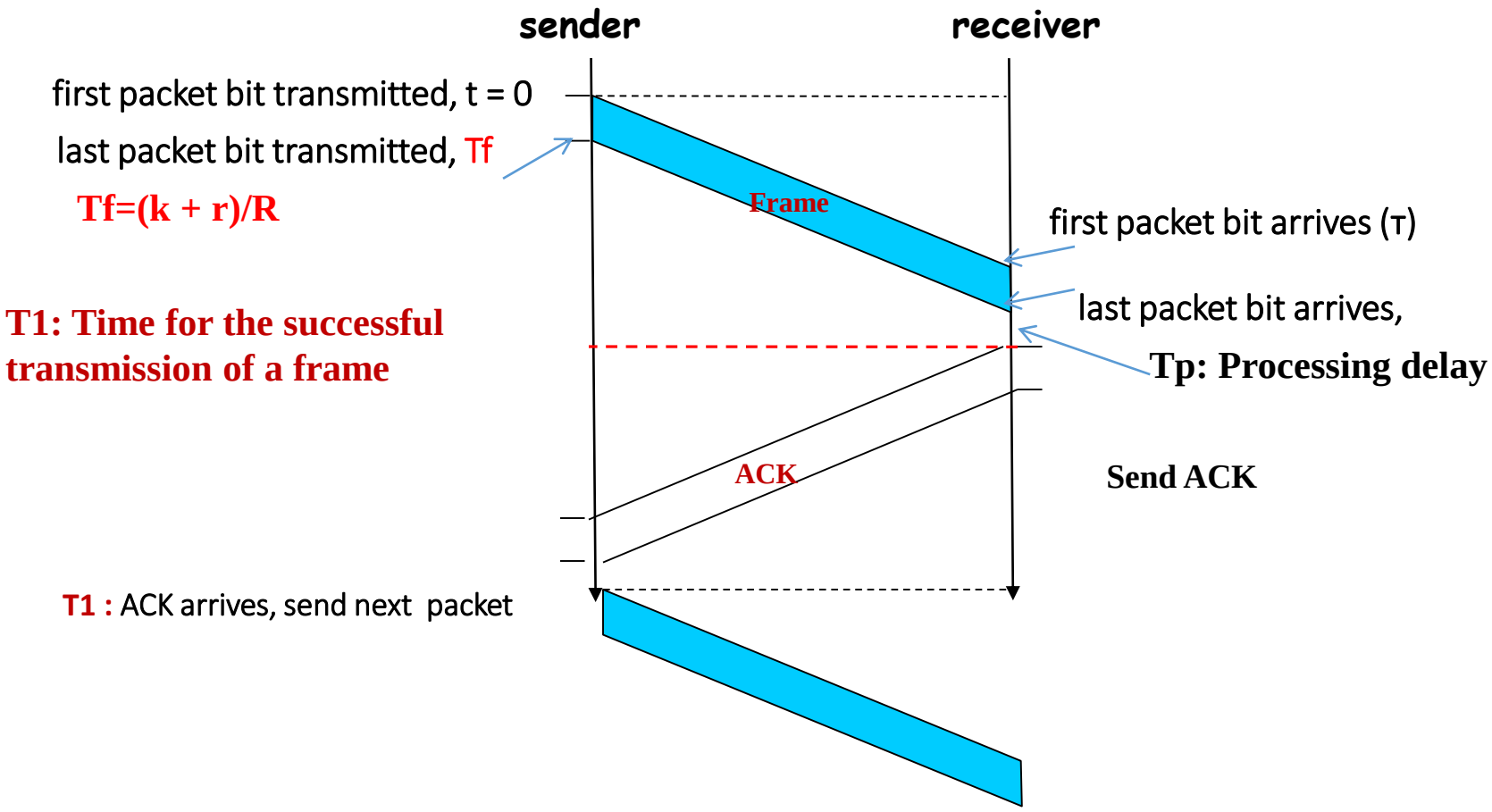
Leon-Garcia & Widjaja: *Communication Networks*

Stop and Wait ARQ Protocol

- The idea is to ensure that a frame has been received correctly before initiating the transmission of the next frame.
- A frame is retransmitted until it is received correctly.
- When a frame is sent, a timer starts.
  - If an ACK is received, a new frame is transmitted.
  - If the elapsed time exceeds a predetermined time (called the **timeout**) and ACK is not received, the frame is retransmitted.
- A frame is transmitted **once** if no errors are detected and the ACK is received by the sender.
- If the frame is in error, or the ACK is lost or was in error, then the frame is retransmitted.

Performance of Stop and Wait

Only one frame can be sent per round trip:



Stop and Wait ARQ

k: number of information bits in a frame

r: number of CRC bits in frame.

(k + r): frame size in bits

m: number of bits in ACK

r: number of CRC bits in ACK.

(m + r): size of ACK in bits.

R: channel capacity in bps.

Tf = (k + r)/R, frame transmission time.

Ta = (m + r)/R, ACK transmission time.

τ: propagation delay.

Tp: processing delay

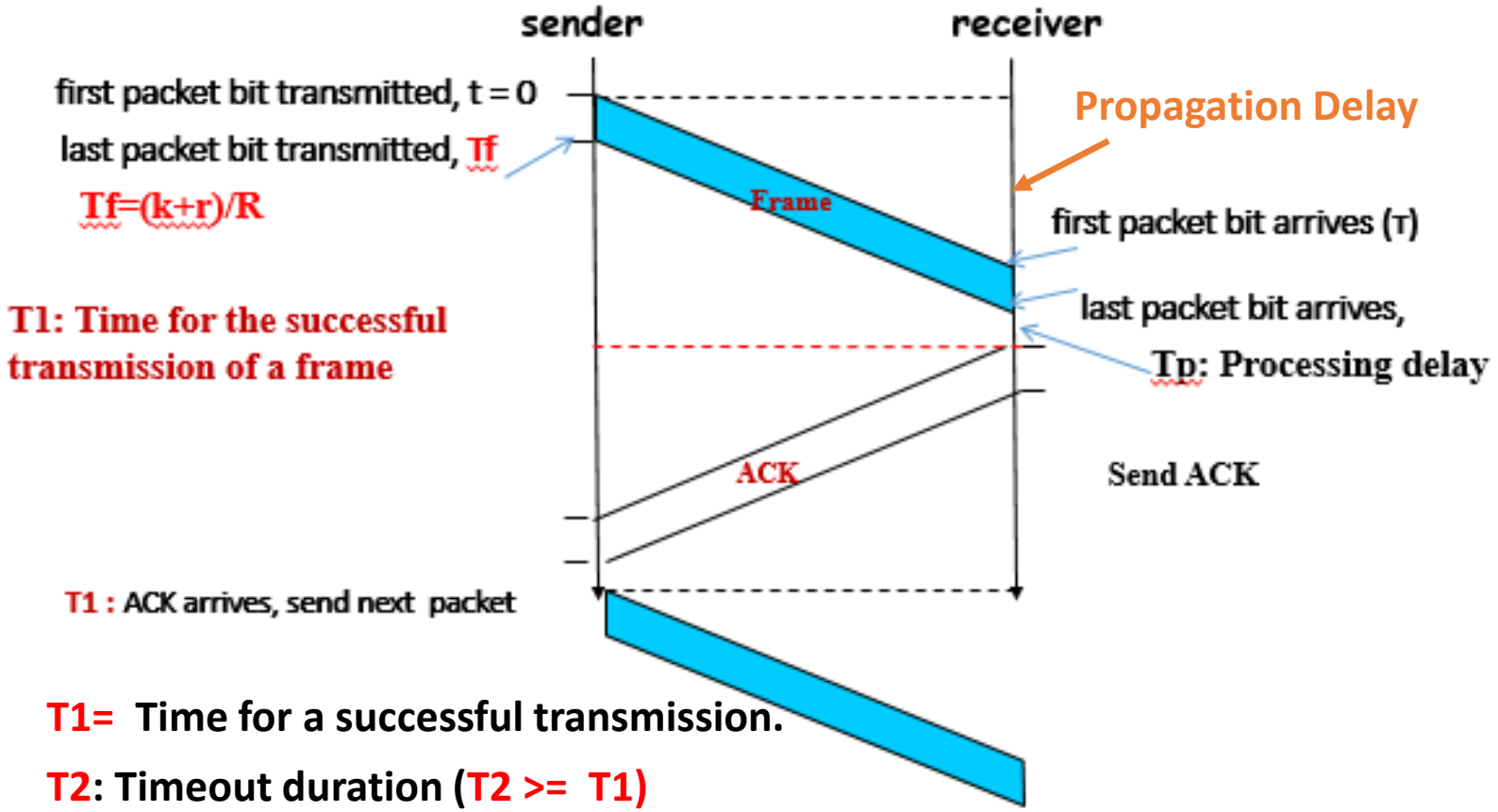
T1= 2τ + Tf + Ta + Tp, Time for a successful transmission.

T2: Timeout duration (time for a failed transmission)

T2≥ T1

P(S): Probability of a successful transmission

P(F): Probability of a failed transmission



T1= Time for a successful transmission.

T2: Timeout duration (T2 >= T1)

Frame = k	CRC = r	Tf=(k + r)/R
ACK = m	CRC = r	Ta=(m + r)/R

Stop and Wait ARQ

- Let T be the random variable representing the time needed to transmit one frame successfully.
- If the first transmission was successful, then

  T= T1; (**T1= 2τ + Tf + Ta + Tp**)
- If first transmission was a failure, but the second one was a success, then

  T= T2 + T1 (T2 is the first timeout)
- If first and second transmissions were failures, but third one was a success, then

  T= 2T2 + T1 (two timeouts + success)
- In general, if **X is the random variable representing the number of transmissions,** then

  T= (X-1)T2 + T1; xX= 1, 2, 3, …

  T is a random variable

- The mean value of T is:

  E(T) = [E(X)-1]T2 + T1
- The random variable X has geometric distribution

$$P(X = x) = P(S)[1 - P(S)]^{x-1}, \quad x = 1, 2, 3, \ldots$$

- The mean value of X is given as

$$E(X) = \frac{1}{P(S)}$$

- The expected value of T is:

$$E(T) = [\frac{1}{P(S)} - 1]T2 + T1 = \frac{P(F)}{P(S)}T2 + T1$$

Transmission Efficiency in Stop and Wait ARQ

- Note that k bits of information need an average time E(T) to be successfully transmitted. In fact, most of the time the transmitter, the receiver, and the channel are idle.
- A measure of the transmission efficiency is what is called the **throughput**, which in this case is best defined as

$$THR = \frac{\text{Time to transmit the information bits}}{\text{Actual mean time taken to deliver the bits}}$$

$$THR = \frac{k/R}{T1 + P(F)T2/P(S)}$$

- If we let T2 = T1, then by substituting the parameters back into the throughput equation, we get

$$THR = \frac{k}{(k+r)+(m+r)+(Tp+2\tau)R}P(S)$$

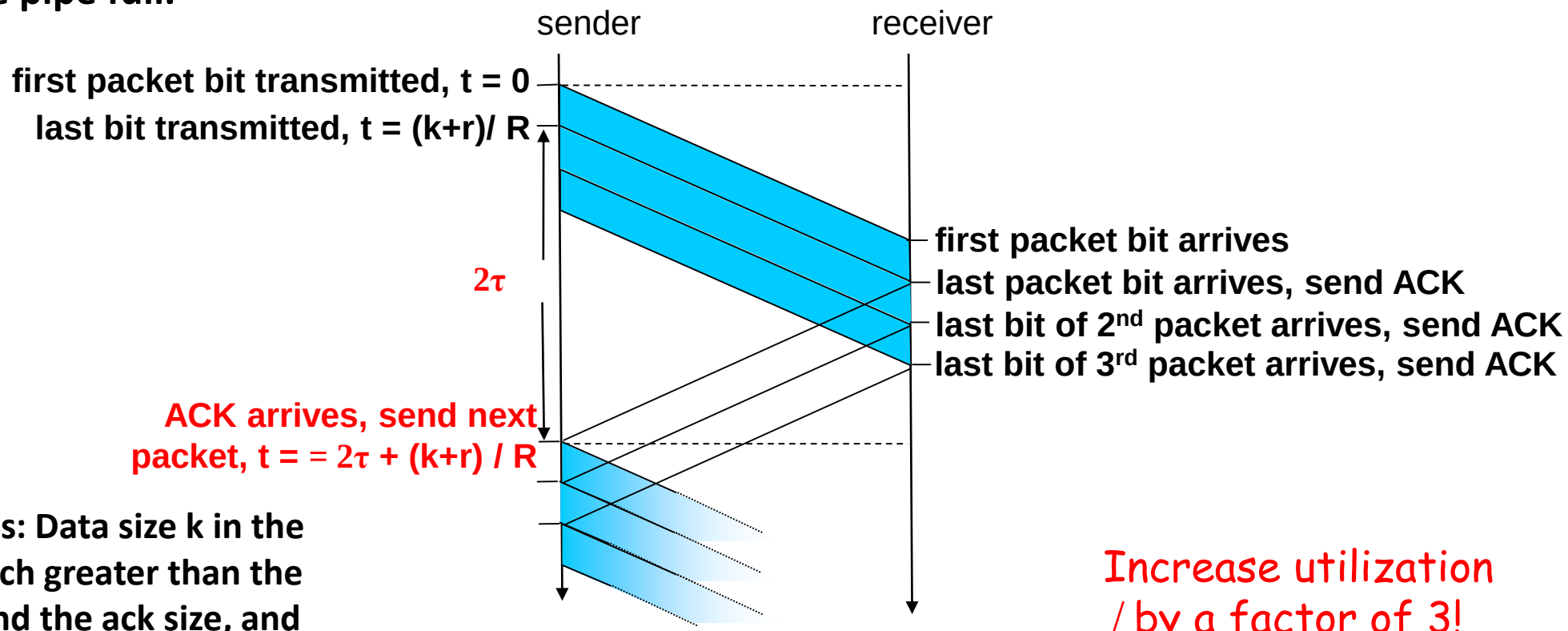
- Furthermore, if the data size k in the frame is much greater than the CRC bits r, m <<k, and the processing delay is ignored, the throughput becomes

$$THR = \frac{k}{k+2\tau R}P(S) = \frac{k/R}{k/R+2\tau}P(S)$$

As can be seen the determining parameter is the round trip delay.

Pipelining: Increasing Utilization

- In stop-and-wait, at any point in time, there is only one frame that is sent and waiting to be acknowledged.. This is not a good use of transmission medium. To improve efficiency, multiple frames should be in transition while waiting for ACK. Two protocol use the above concept; Go-Back-N ARQ and Selective Repeat ARQ
- **Pipelining:** sender allows multiple, yet-to-be-acknowledged packets without waiting for first to be ACKed to keep the pipe full.



Assumptions: Data size k in the frame is much greater than the CRC bits r and the ack size, and the processing delay is ignored.

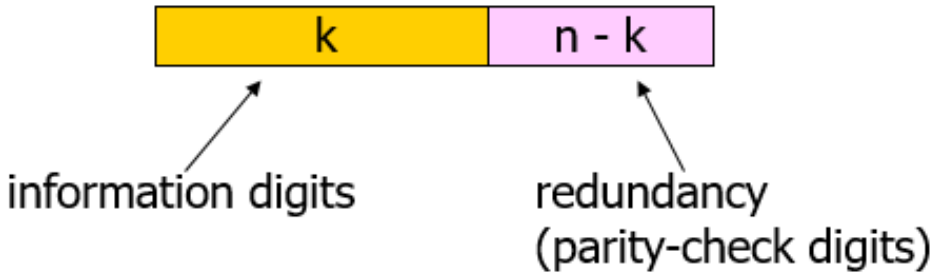
$$THR = \frac{3k}{k + 2\tau R} P(S) = \frac{3k / R}{k / R + 2\tau} P(S)$$

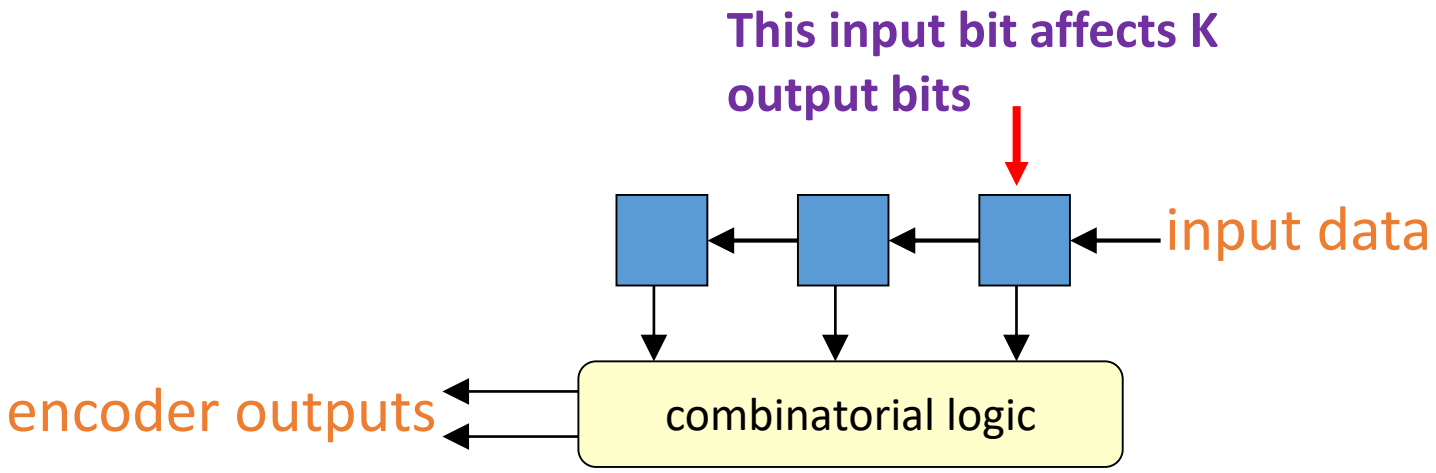
Increase utilization by a factor of 3!

Convolutional Codes

Lecture 23

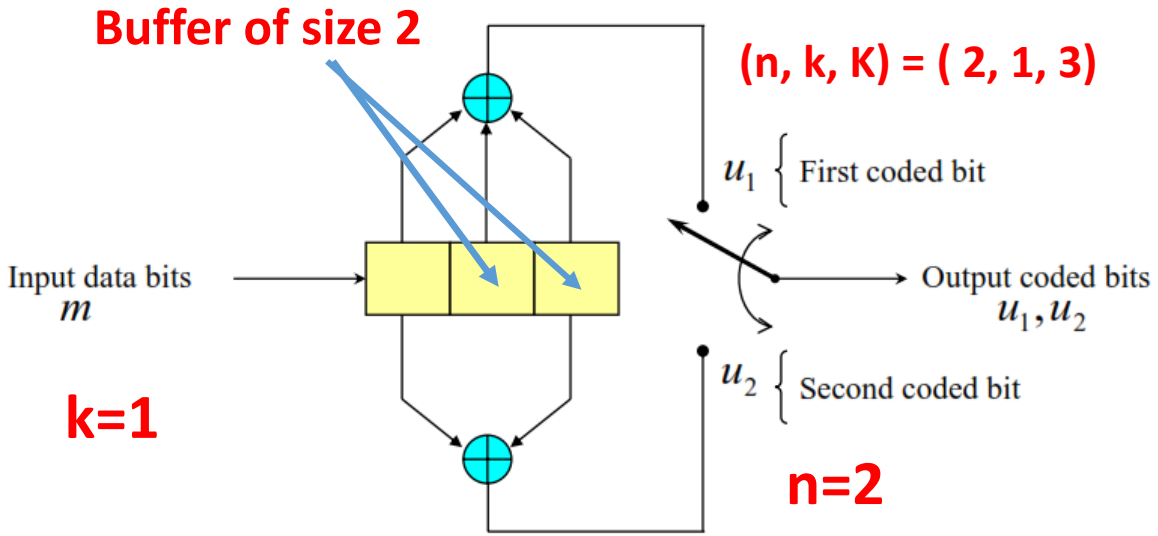
- **Linear Block Codes**
- In a linear (n, k) block code, a block of *k*-bit data is encoded into a codeword of length *n by adding (n-k) redundant bits*
- The encoding is done independently from block to block (memory-less encoding)
- **Convolutional Codes**
  - Encoding is done in a bit-by-bit manner
  - Previous inputs are stored in shift-registers in the encoder, and affect future encoding





Convolutional Codes

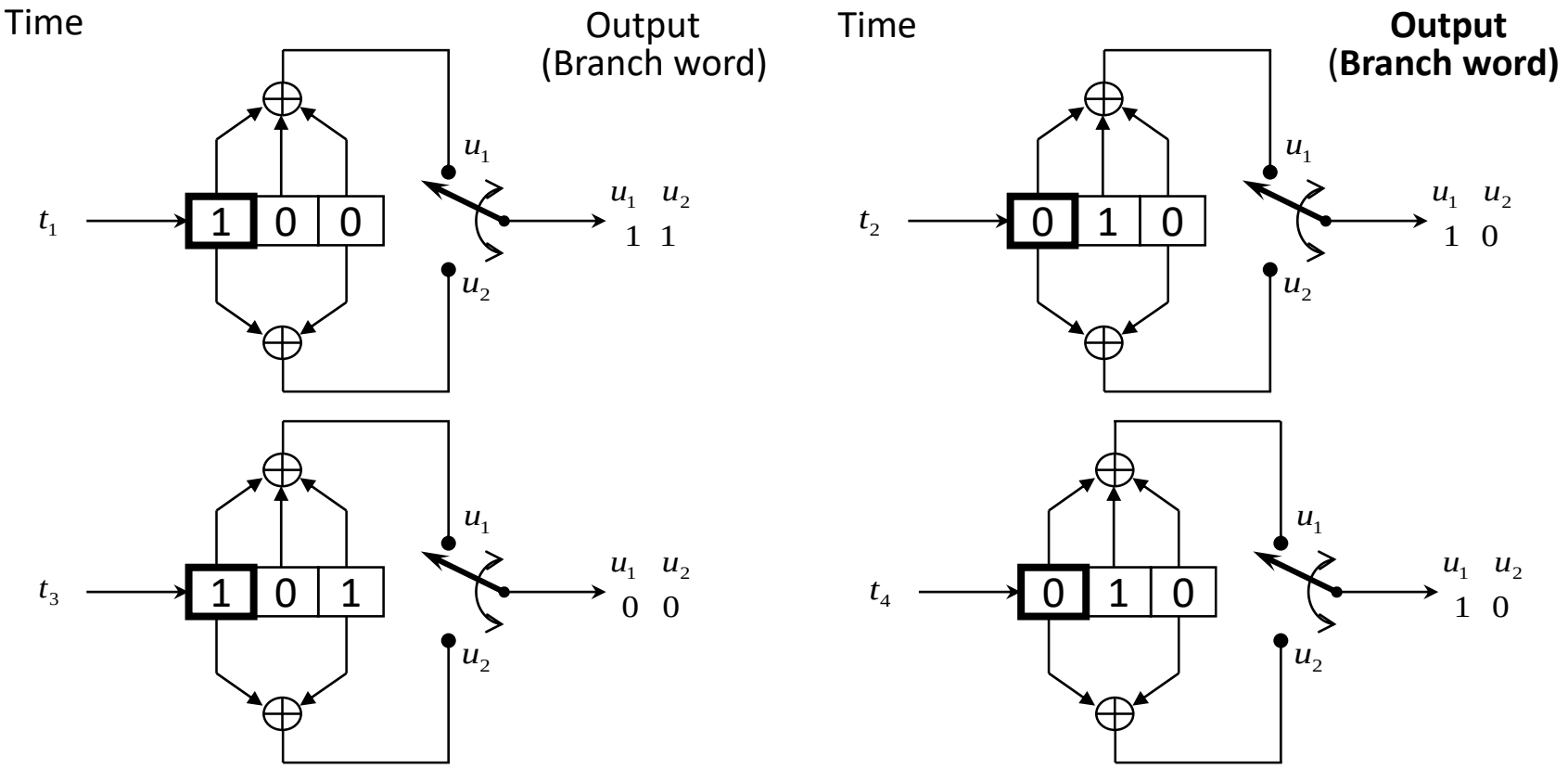
- A convolutional code is specified by three parameters (n, k, K)
- *k input bits will produce n output bits* (Here we will take k = 1)
- *n: will be the number of modulo 2 adders in the encoder.*
- The most recent *(K-1) input message bits will be* recorded (buffered).
- (K-1): number of memory elements in the encoder.
- The *n output bits will be given by a linear combination of the* buffered input bits and the current input bits.
- K: is called the Constraint length *of the convolutional code and represents* the number of output bits influenced by a single input bit.
- The encoder output depends on the current input message bit and the previous (K – 1) input message bits (stored in the buffer).



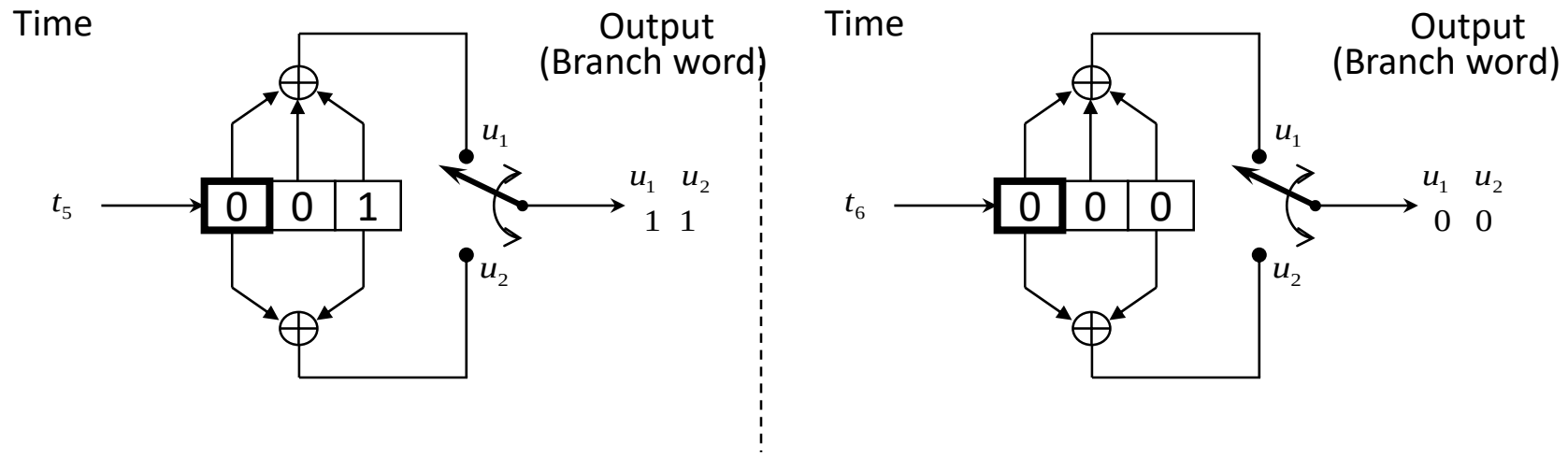
- In this convolutional encoder there are two output binary digits for each one input binary digit. This gives a rate (k/n = ½) for the code.
- Each adder adds its inputs modulo 2.
- There are 3 shift registers, where the first one takes the incoming data bit and the rest from the memory of the encoder.

Example: A Rate ½ Convolutional Encoder

Message sequence: $\mathbf{m} = (101)$



Example: A Rate ½ Convolutional Encoder



$$\mathbf{m} = (101) \longrightarrow \boxed{\text{Encoder}} \longrightarrow \mathbf{U} = (11\ \ 10\ \ 00\ \ 10\ \ 11)$$

n = 2, k = 1, K = 3,

L = 3 input bits -> 10 output bits

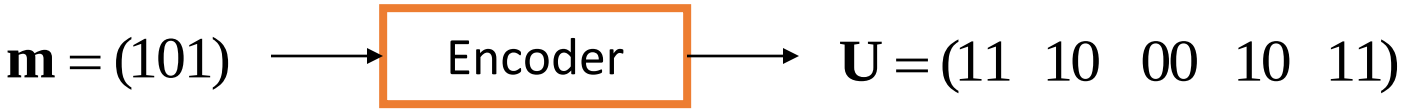
Note that two extra zeroes have been appended to the message to return the registers to the zero state

Effective code rate

- Initialize the memory before encoding the first bit (all-zero)
- Clear out the memory after encoding the last bit (all-zero)
  - Hence, a tail of zero-bits is appended to data bits.

- Effective code rate :
  - L is the number of data bits and *k=1* is assumed:

$$R_{eff} = \frac{L}{n(L+K-1)} < R_c$$



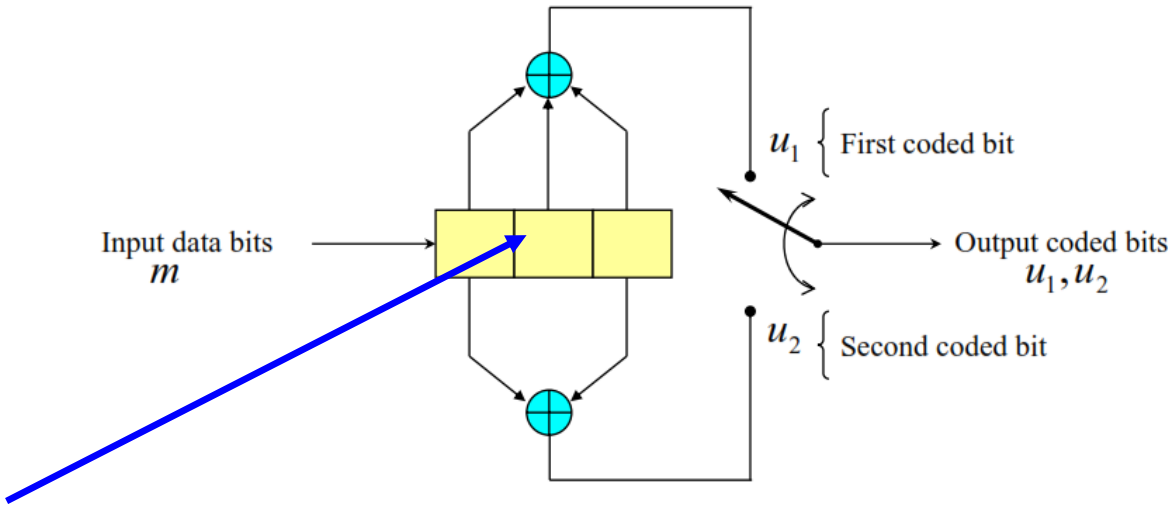
Example: n = 2, k = 1, K = 3, L = *3 input bits.*
Output = n(L + K -1) = 2*(3 + 3 – 1) = *10 output bits*

State diagram of a convolutional code

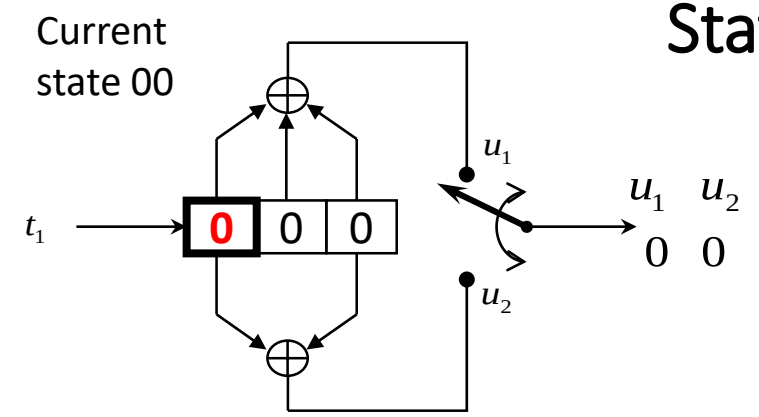
- One way or representing the convolutional code is by means of a state diagram.
- By a state we mean the contents of the first two registers. At any time, the contents of the registers can be at any of the four states (0, 0), (0,1), (1, 0), and (1, 1).
- The output sequence at each stage is determined by the input bits and the state of the encoder.
- A state diagram is simply a graph of the possible states of the encoder and the possible transitions from one state to another. It can be used to show *the relationship between the encoder state, input, and output.*

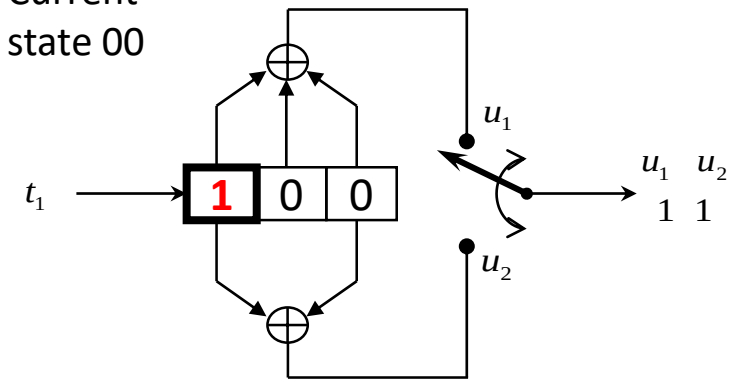
$$u_1 = m_n \oplus m_{n-1} \oplus m_{n-2}$$
$$u_2 = m_n \oplus m_{n-2}$$

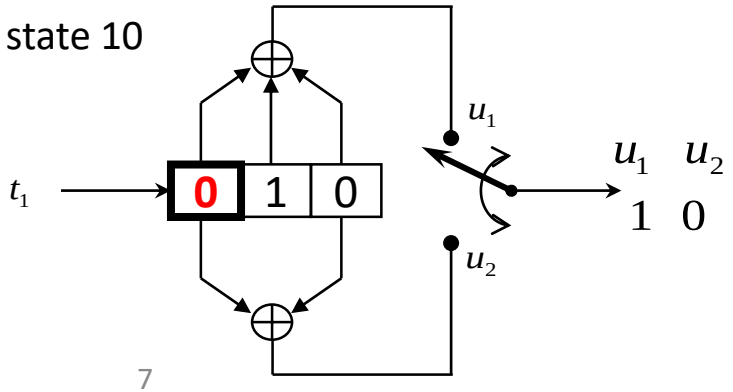
The state refers to the contents of the first two registers

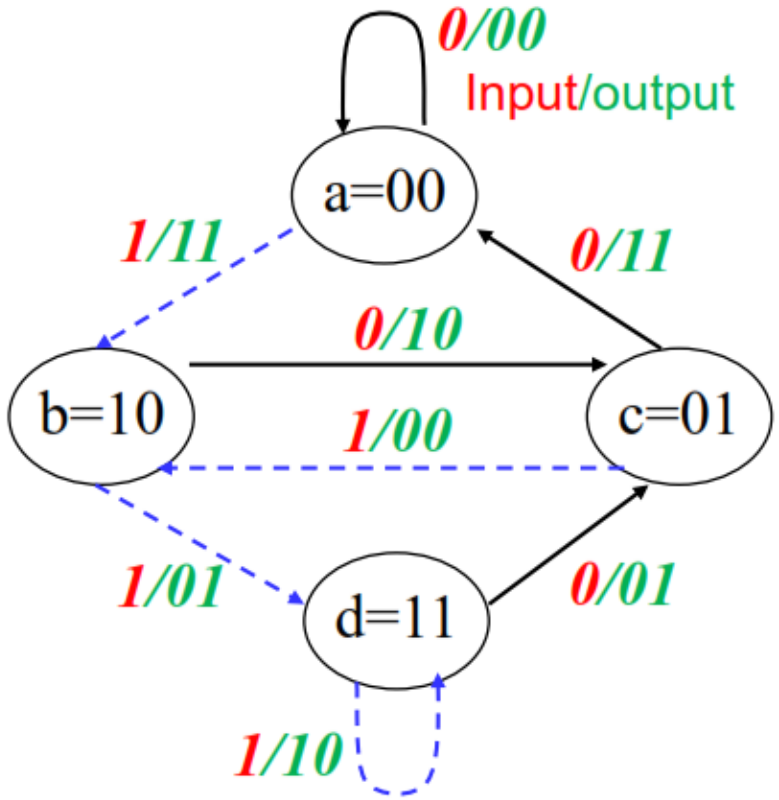


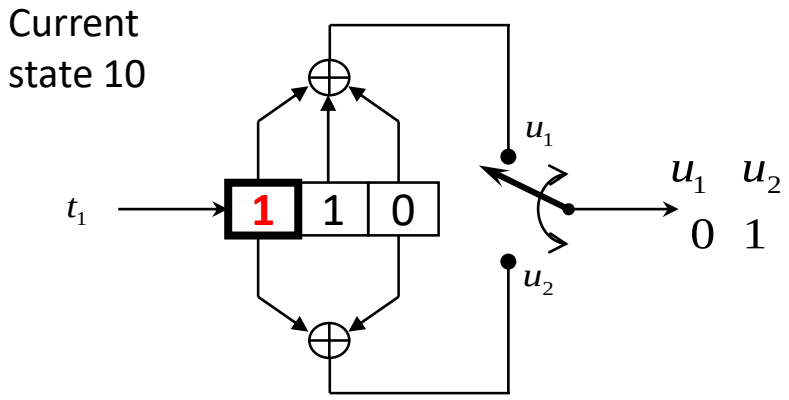
State diagram of a rate 1/2 convolutional code











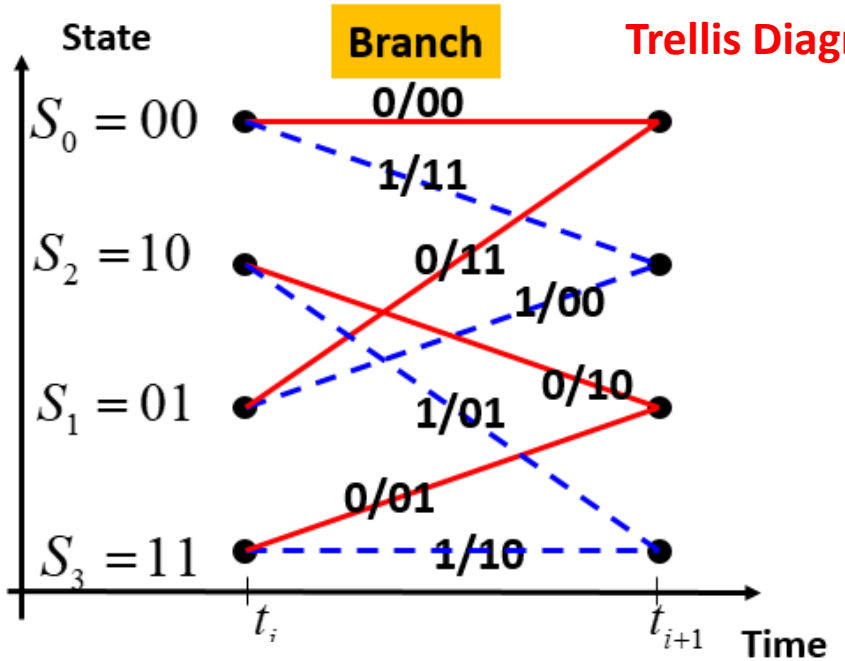
Current State	Input	Next State	Output
00	0	00	***00***
	1	10	***11***
10	0	01	***10***
	1	11	***01***
01	0	00	***11***
	1	10	***00***
11	0	01	***01***
	1	11	***10***

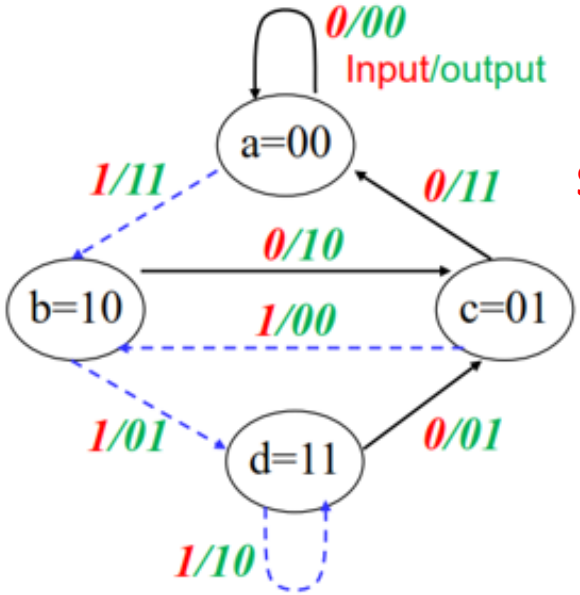
$$\mathbf{m} = (11100)$$

$$\mathbf{U} = (11\ \ 01\ \ 10\ \ 01\ \ 11)$$

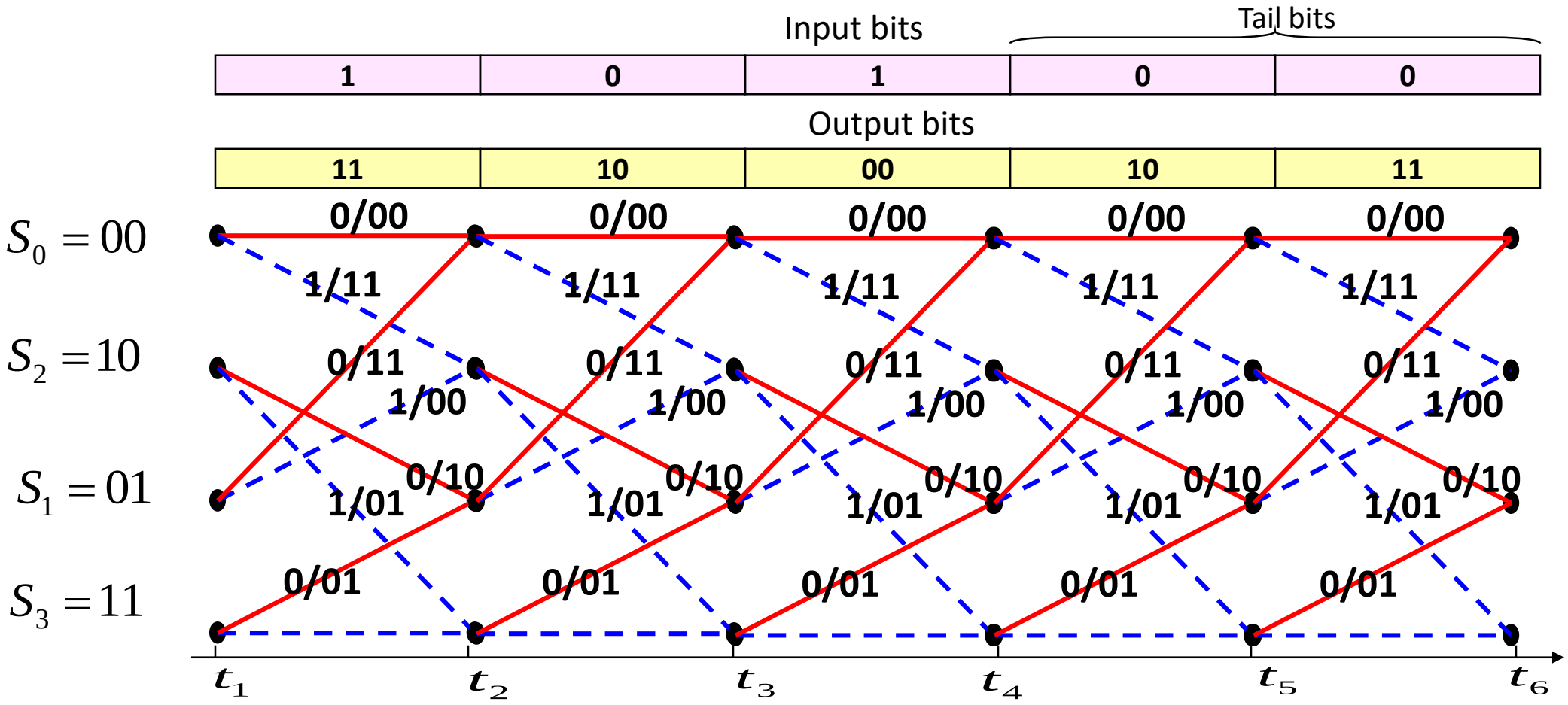
Trellis diagram of a convolutional code

- Trellis diagram is an extension of state diagram which *explicitly shows the passage of time.*
- All the possible states are shown for each instant of time.
- Time is indicated by a movement to the right.
- The input data bits and output code bits are represented by a unique path through the trellis (as we shall see on the next slide)
- Each line designated with the input digit and output digits in the form x/y.
- After the second stage, each node in the trellis has *2 incoming paths and 2* outgoing paths.
- A solid line indicates that the input bit is 0.
- A dotted line indicates that the input bit is 1.

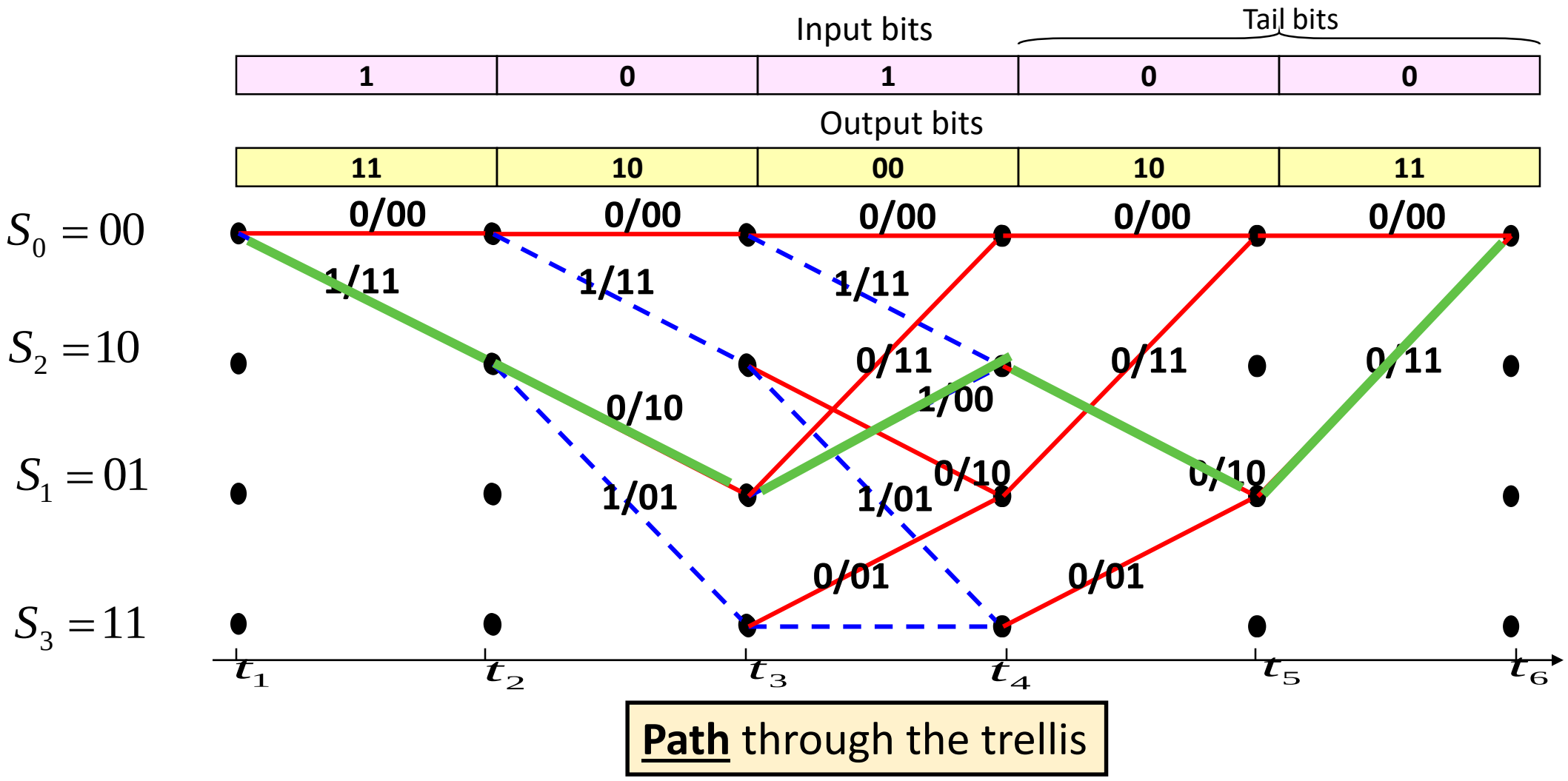




State diagram of a rate 1/2 convolutional code



State diagram of a rate 1/2 convolutional code

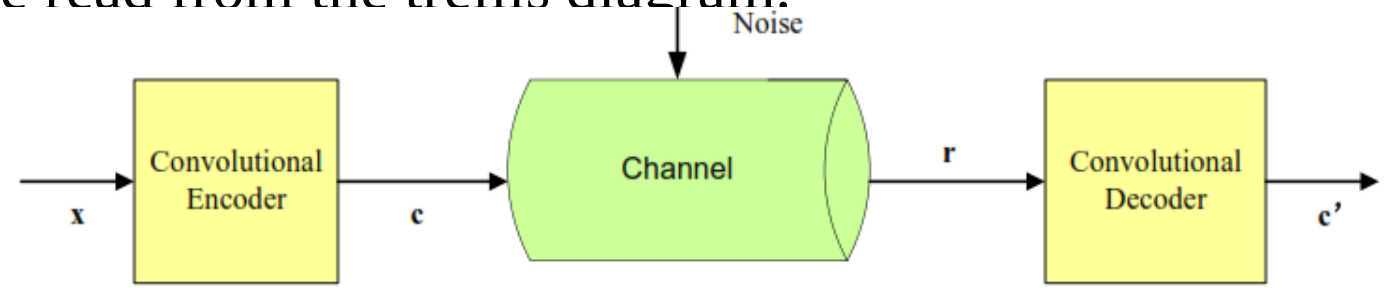


Path through the trellis

Maximum Likelihood Decoding of Convolutional Code

- As we have seen earlier, bits transmitted over the channel will be subject to errors. Here,

  Receiver sequence = Transmitted sequence + Error pattern
- Given the received code word **r, determine the most** likely path through the trellis that caused this **r.**
- Compare **r with the code bits associated with each path. (each path represents a codeword)**
- Pick the path whose code bits are "closest" to **r.**
- Measure distance using either *Hamming distance* for hard decision decoding or *Euclidean distance* for soft-decision decoding. In this lecture, only hard decision decoding will be considered.
- Once the most likely path has been selected, the estimated data bits can be read from the trellis diagram.

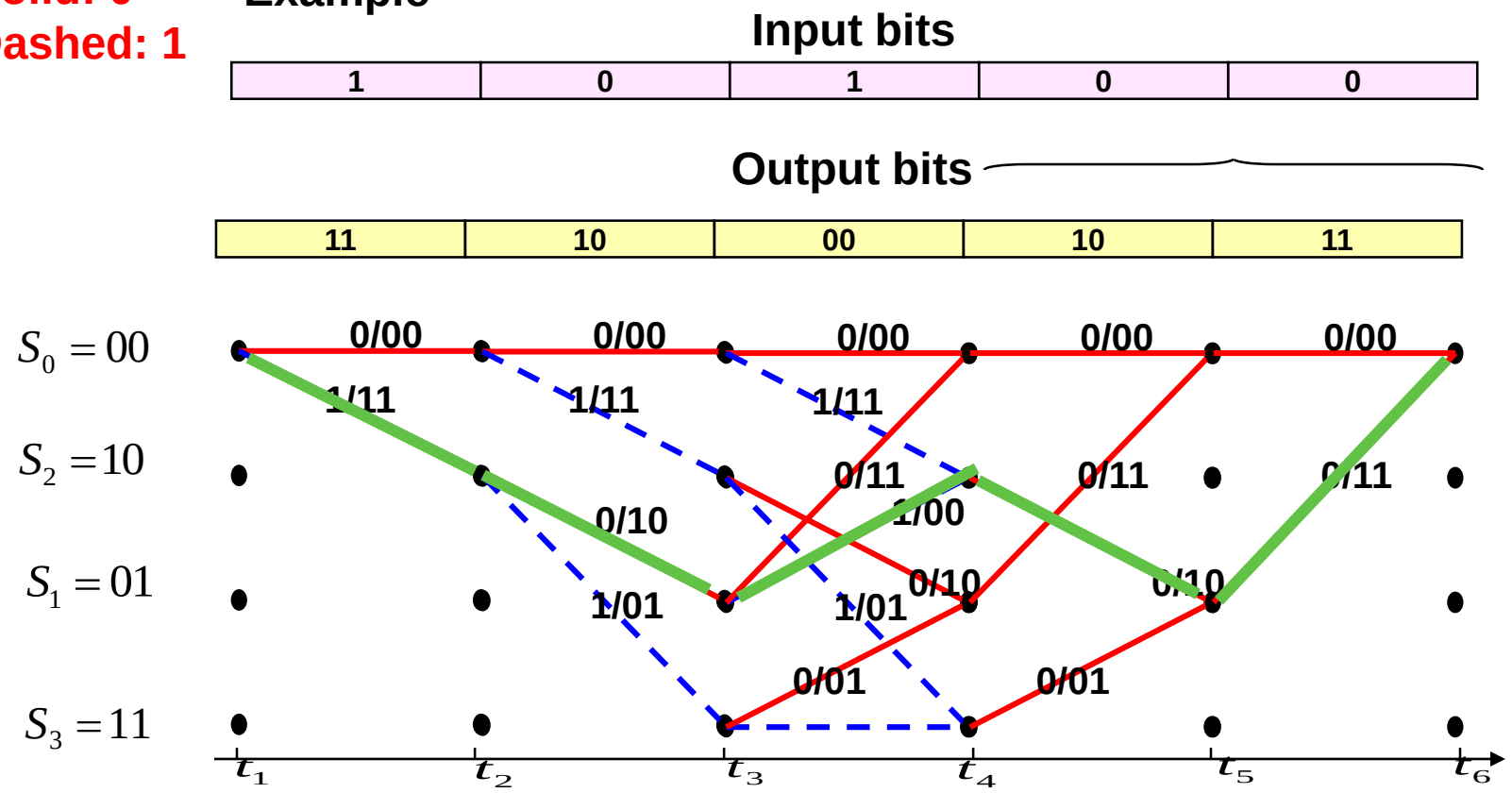


State diagram of a rate 1/2 convolutional code

Each path in the trellis is a codeword

Solid: 0
Dashed: 1

Example

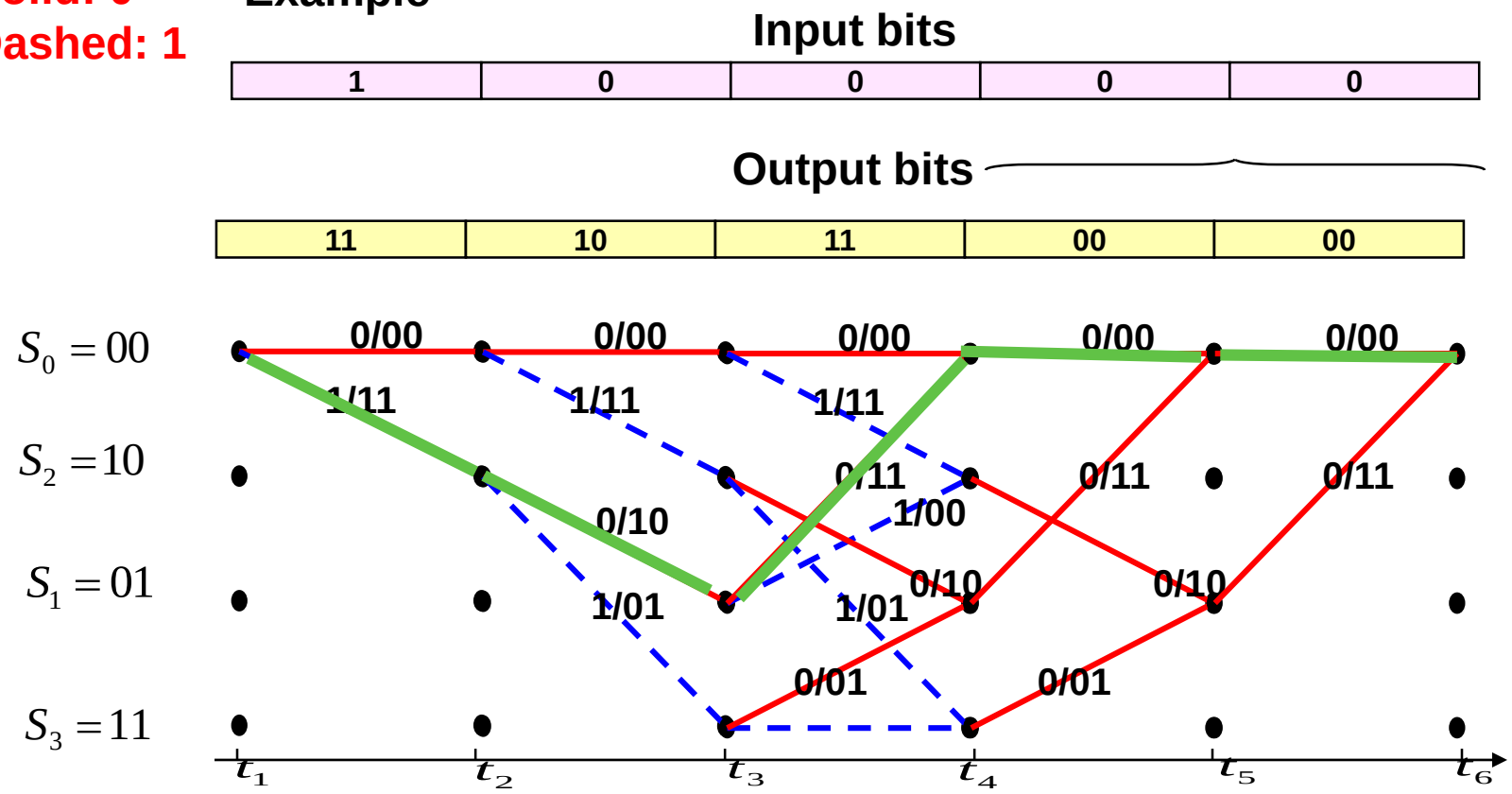


Hamming Distance from the received sequence r = 10 01 10 11 00 = 7

State diagram of a rate 1/2 convolutional code

Solid: 0
Dashed: 1

Example

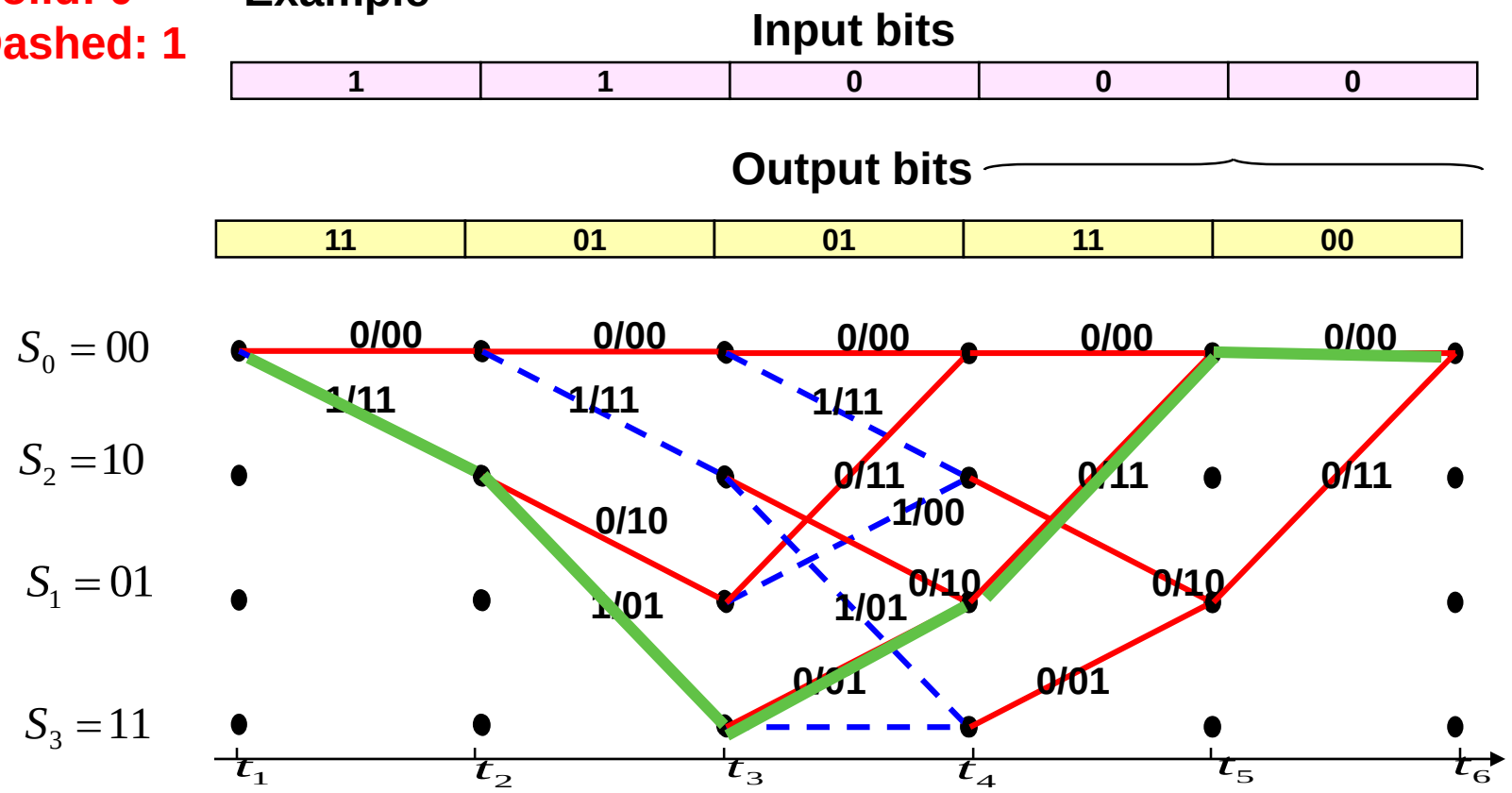


Hamming Distance from the received sequence r = 10 01 10 11 00 = 6

State diagram of a rate 1/2 convolutional code

Solid: 0
Dashed: 1

Example



Hamming Distance from the received sequence r = 10 01 10 11 00 = 3

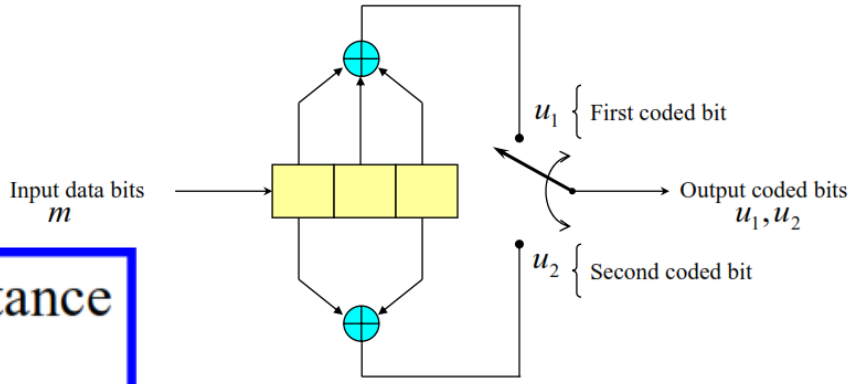
Example on Maximum Likelihood Decoding

- First, we use the maximum likelihood method to decode the message encoded by the convolutional encoder below when the received sequence of digits is:
- **r = 10 01 10 11 00.** The message size L=3 and hence the number of codewords is 8.
- The state diagram and the trellis diagrams for this encoder were derived in the previous lecture.

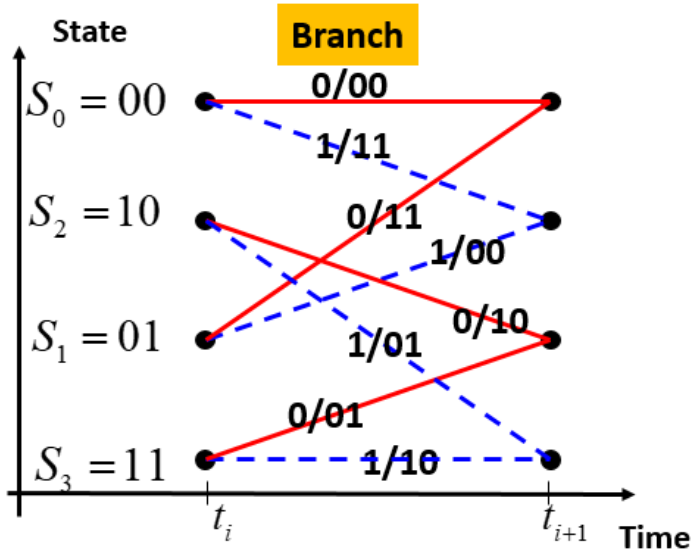
Maximum Likelihood decoding

00 11 10 11 00 --> 01000

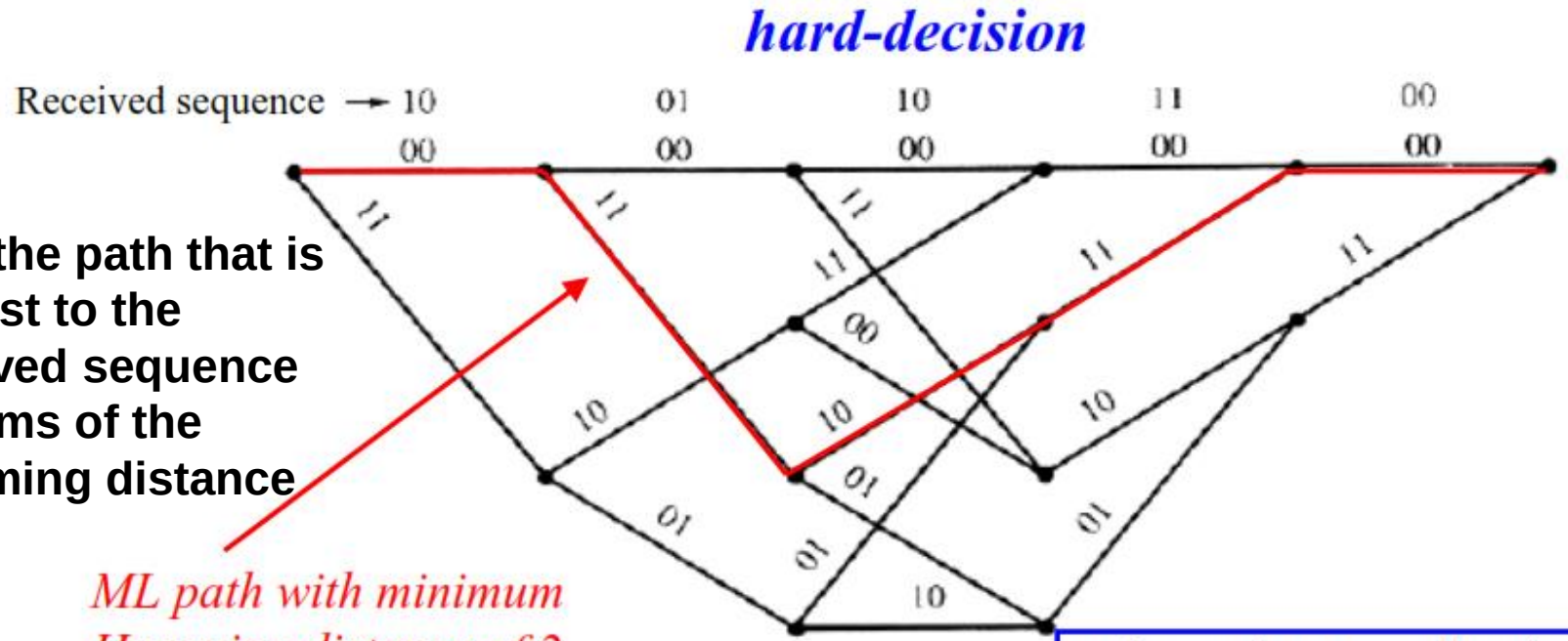
r = 10 01 10 11 00



path	code sequence	Hamming distance
0 0 0 0 0	00 00 00 00 00	5
0 0 1 0 0	00 00 11 10 11	6
0 1 0 0 0	00 11 10 11 00	2
0 1 1 0 0	00 11 01 01 11	7
1 0 0 0 0	11 10 11 00 00	6
1 0 1 0 0	11 10 00 10 11	7
1 1 0 0 0	11 01 01 11 00	3
1 1 1 0 0	11 01 10 01 11	4



Example on Maximum Likelihood Decoding



Find the path that is closest to the received sequence in terms of the Hamming distance

path	code sequence	Hamming distance
0 0 0 0 0	00 00 00 00 00	5
0 0 1 0 0	00 00 11 10 11	6
0 1 0 0 0	00 11 10 11 00	2
0 1 1 0 0	00 11 01 01 11	7
1 0 0 0 0	11 10 11 00 00	6
1 0 1 0 0	11 10 00 10 11	7
1 1 0 0 0	11 01 01 11 00	3
1 1 1 0 0	11 01 10 01 11	4

All path metrics should be computed.

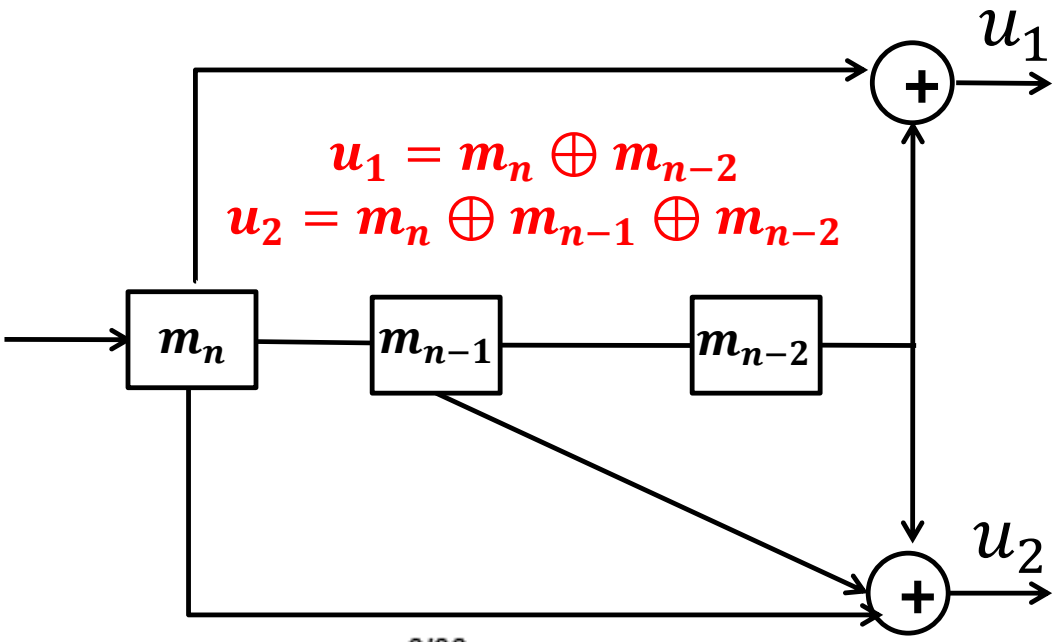
Received Sequence: 10 01 10 11 00
Maximum Likelihood decoding
00 11 10 11 00 --> 01000

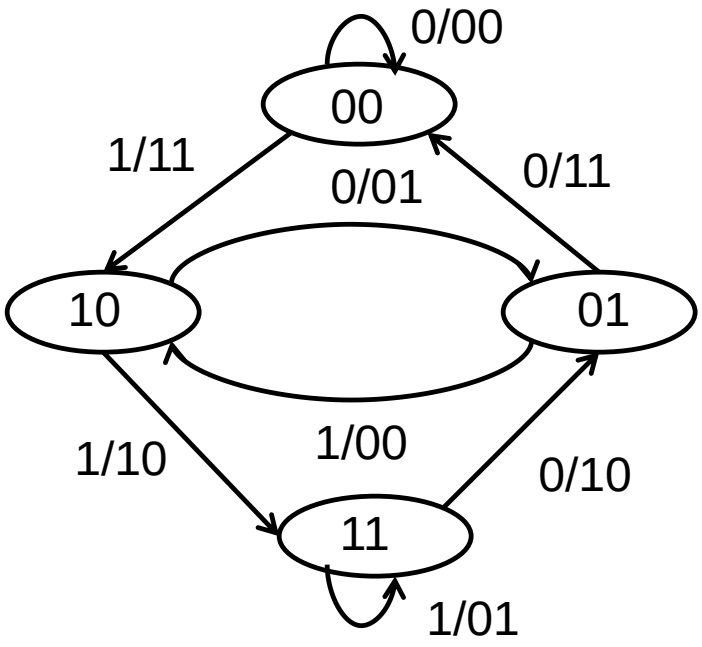
Error Correction Power
For this code, we find that minimum distance of the code dmin=5.
Since dmin = 2t + 1, hence this code can correct up to two bits.

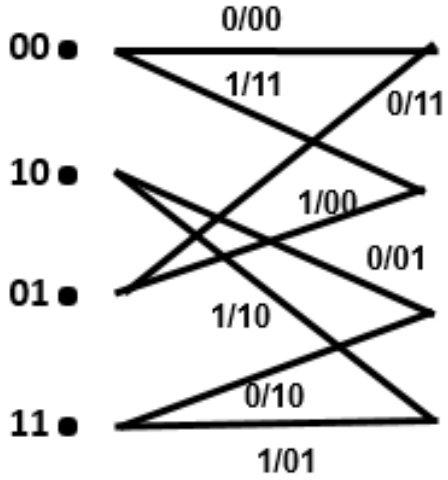
The Viterbi Algorithm

- The Viterbi algorithm is used to decode convolutional codes and any structure or system that can be described by a trellis.
- It is a maximum likelihood decoding algorithm that selects the most probable path that maximizes the likelihood function.
- The algorithm is based on
- Walk through the trellis and compute the Hamming distance between the branches in the trellis and the received sequence r
- At each level, consider the two paths entering the same node and are identical from this node onwards. From these two paths, the one that is closer to r at this stage will still be so at any time in the future. This path is retained, and the other path is discarded.
- Proceeding this way, at each stage **one path will be saved for each node**. These paths are called the **survivors**. The decoded sequence (based on Minimum Distance Decoding MDD) is guaranteed to be one of these survivors.
- Each survivor is associated with a metric of the accumulated Hamming distance (the Hamming distance up to this stage).
- Carry out this process until the received sequence is considered completely.
- Choose the survivor with the smallest metric.

Example on Convolutional Encoder







- When the data sequence **1 1 0 0 1 0 1 0** is applied to the encoder, the coded output bit sequence is

  x=[11 10 10 11 11 01 00 01].

Step 1:

Viterbi Decoding Example

- Consider an example
- When the data sequence **1 1 0 0 1 0 1 0** is applied to the encoder, the coded output bit sequence is

  x=[11 10 10 11 11 01 00 01].

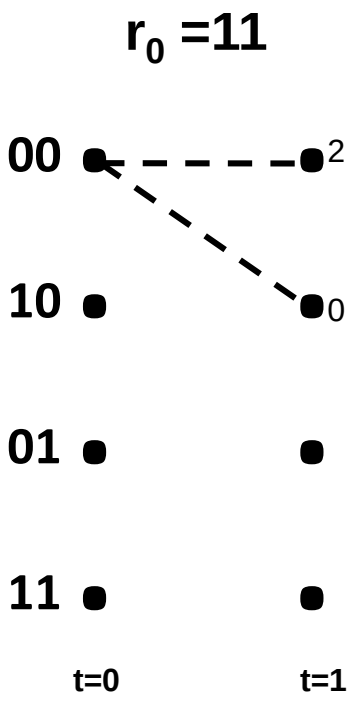
- The coded output sequence passes through a channel, producing the received sequence

  r= [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01].

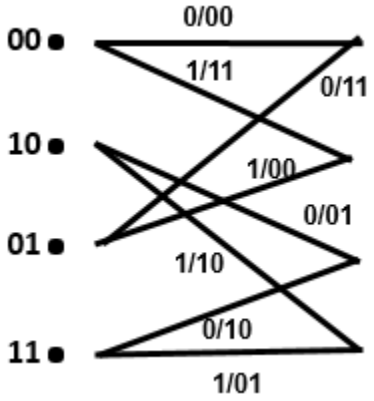
- The two underlined bits are flipped by noise in the channel.

Step 2:

Viterbi Decoding Example

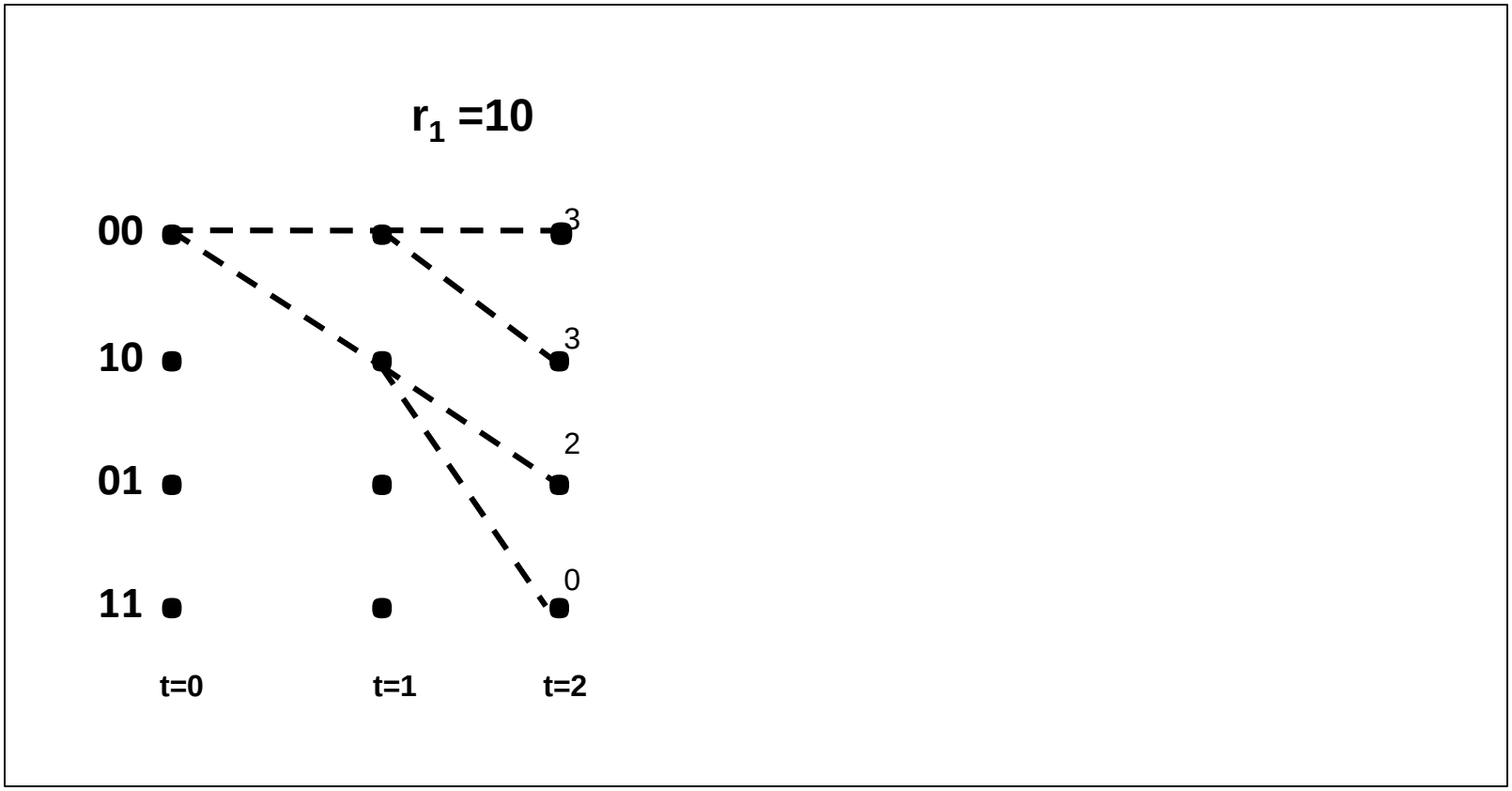


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

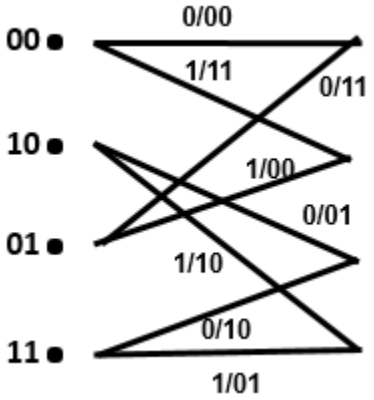


Step 3:

Viterbi Decoding Example

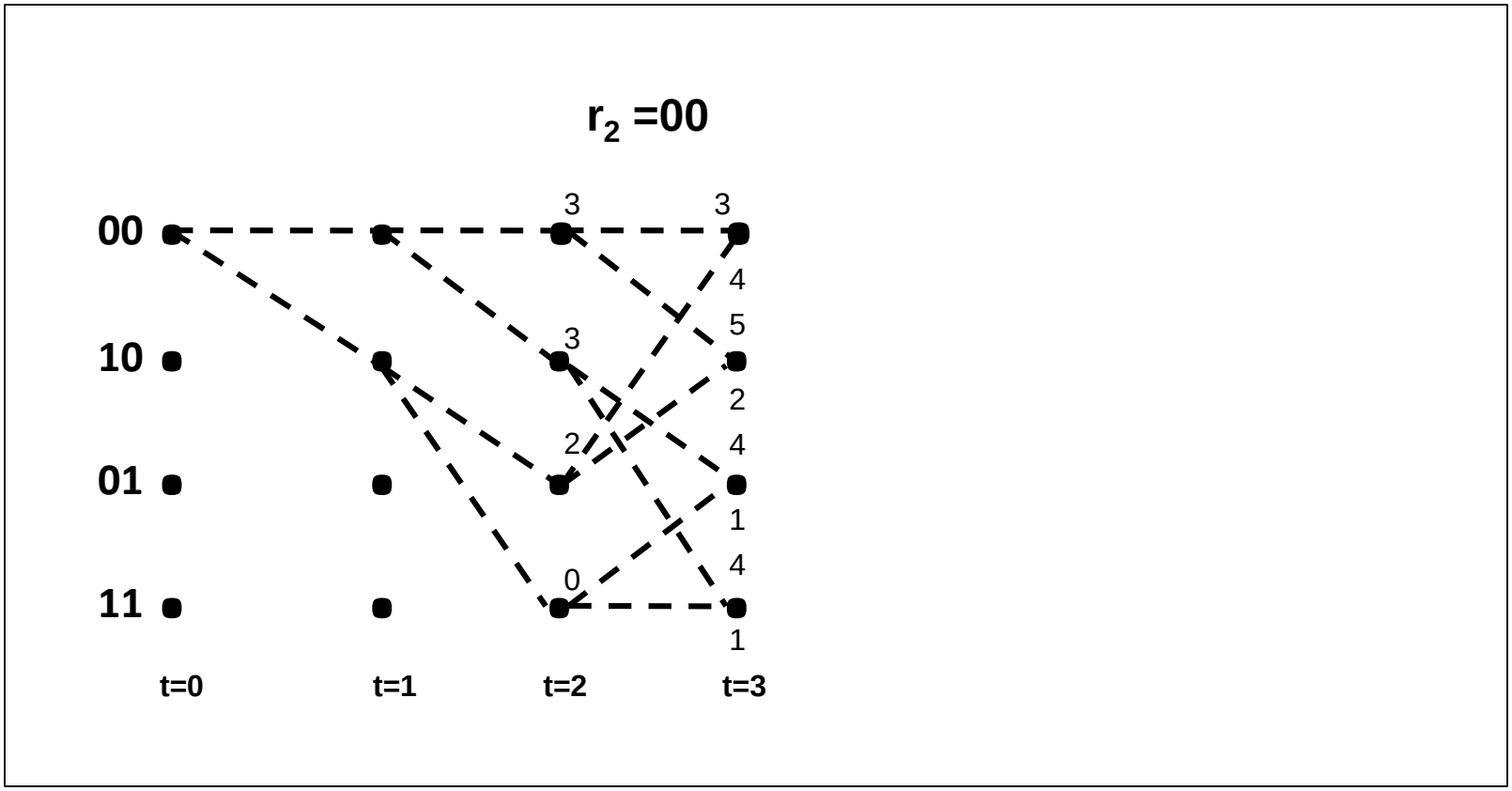


R = [11 10 <u>00</u> 1<u>0</u> 11 01 00 01]

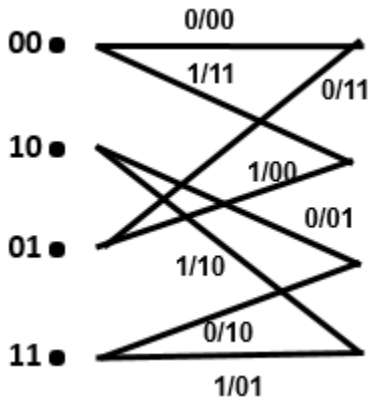


Step 4:

Viterbi Decoding Example

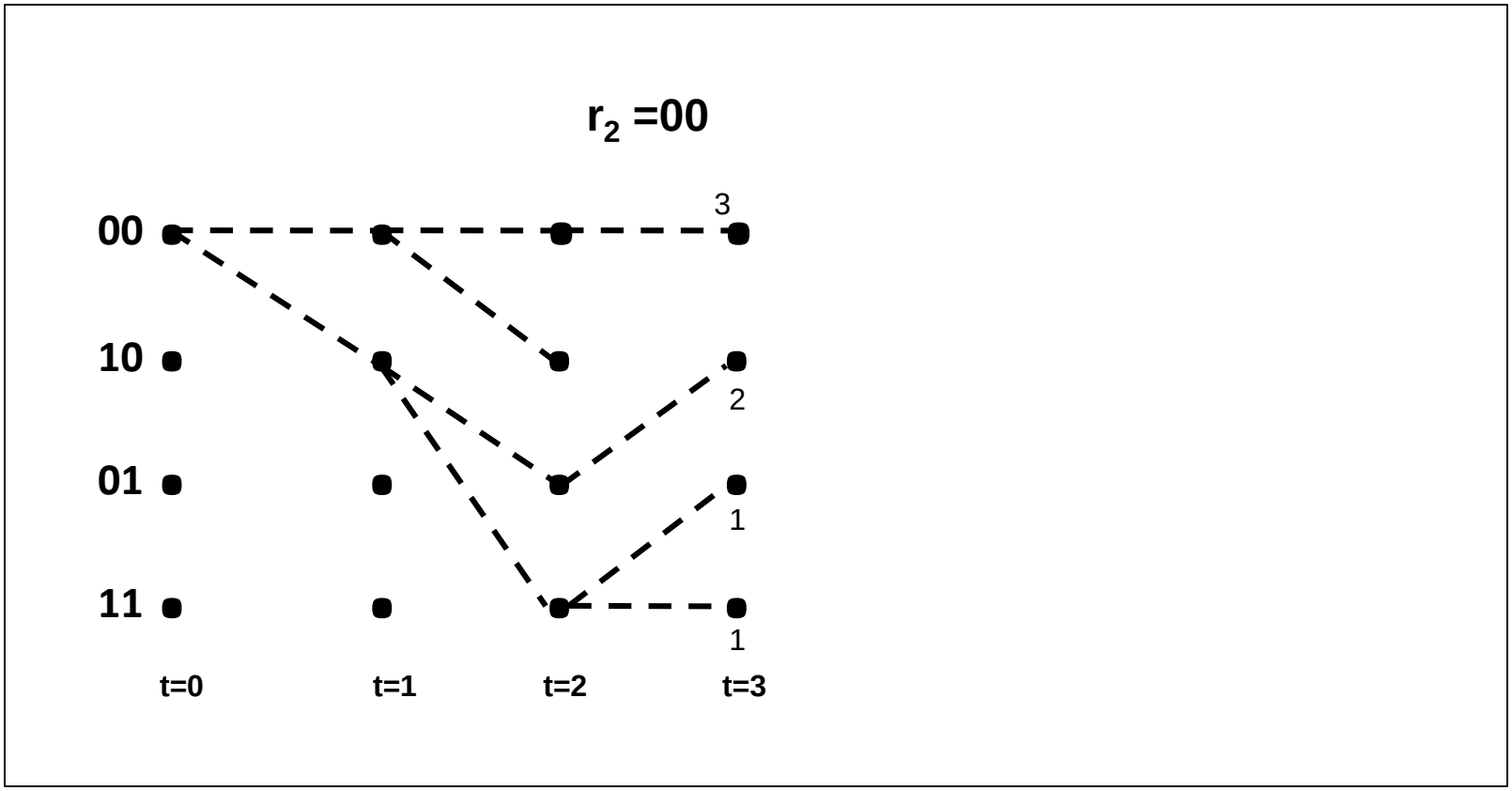


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

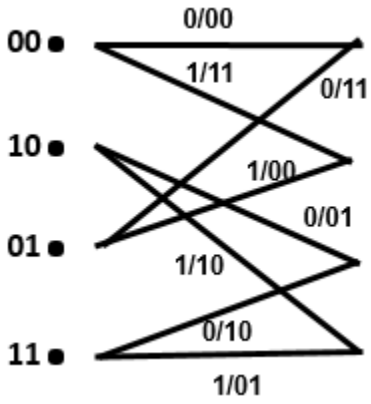


Step 5:

Viterbi Decoding Example

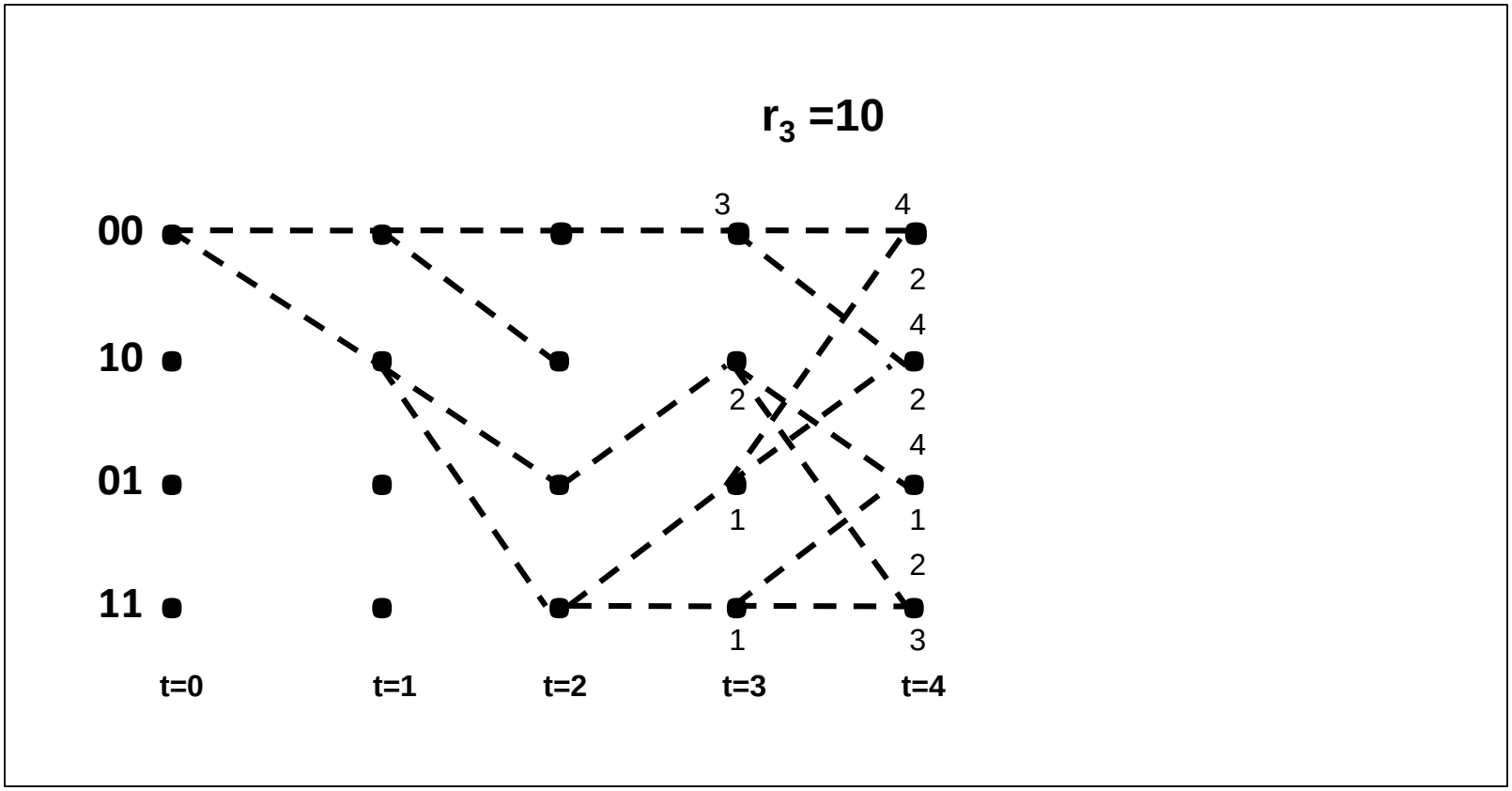


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

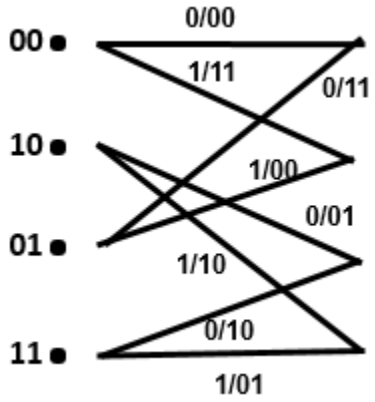


Step 6:

Viterbi Decoding Example

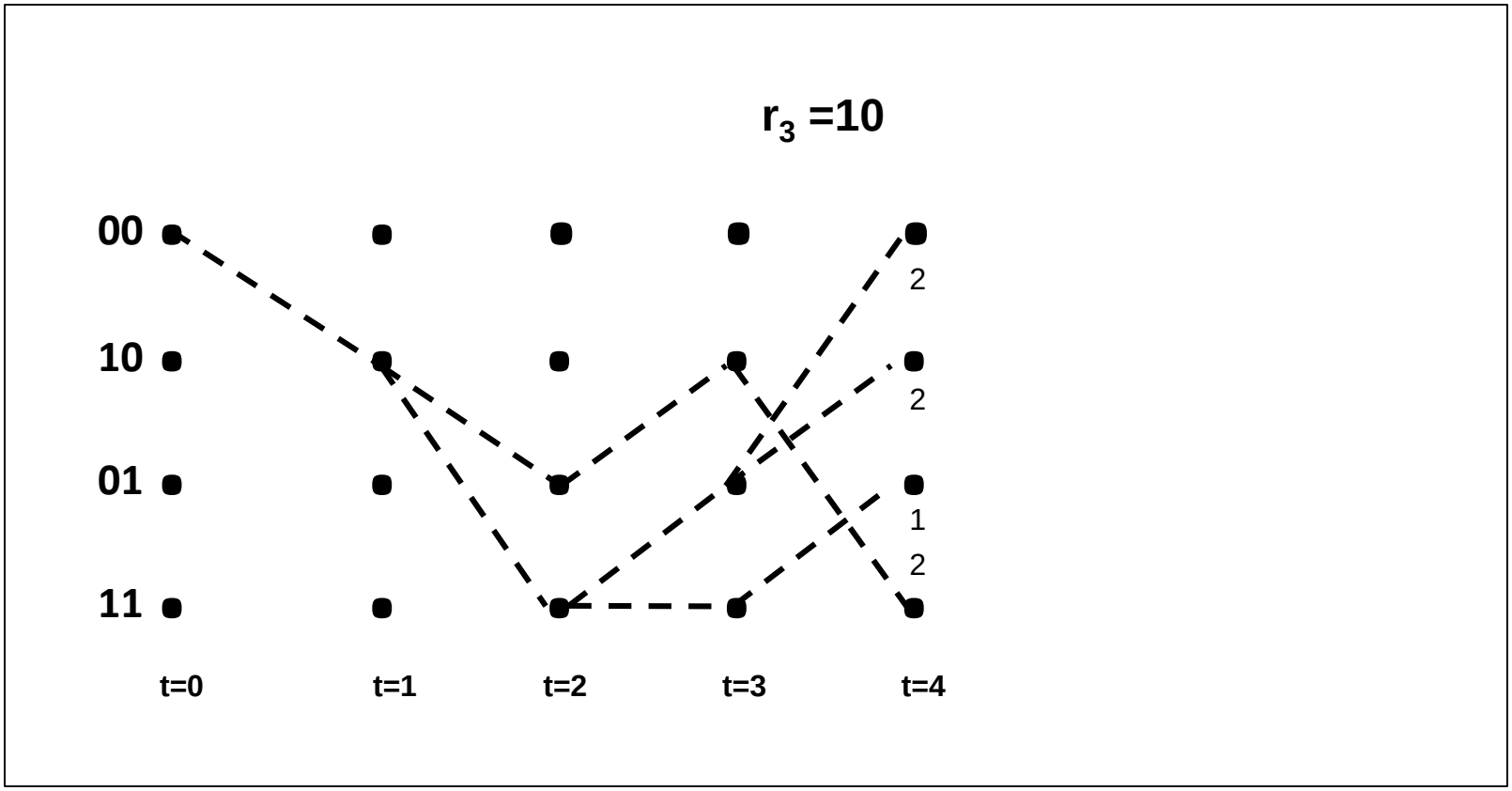


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

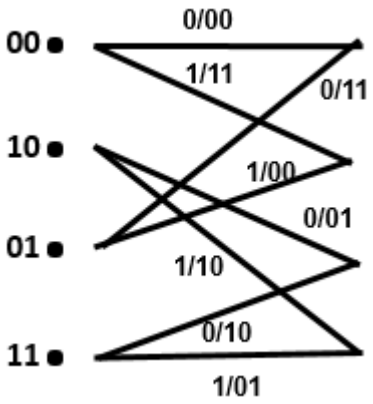


Step 7:

Viterbi Decoding Example

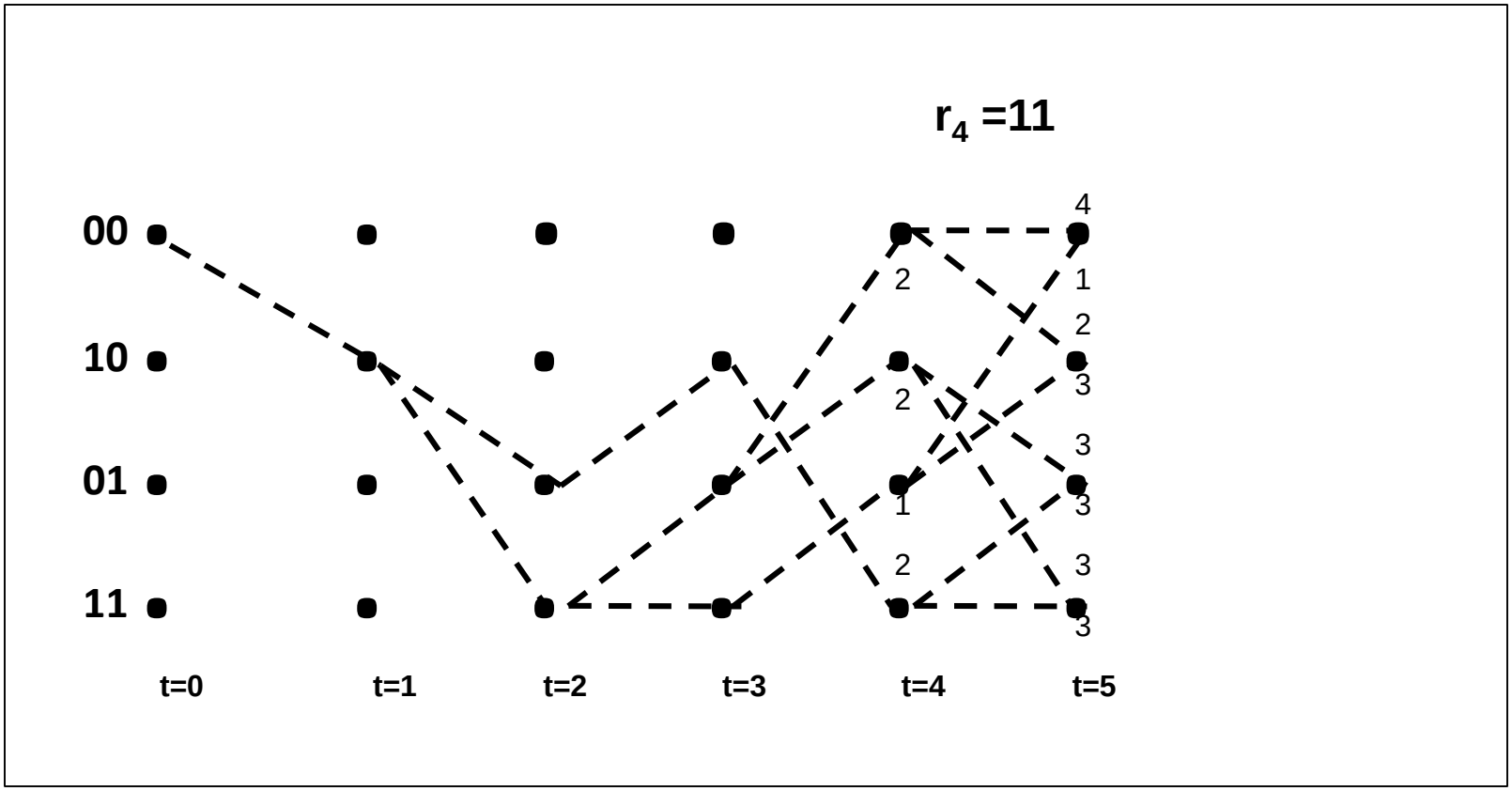


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

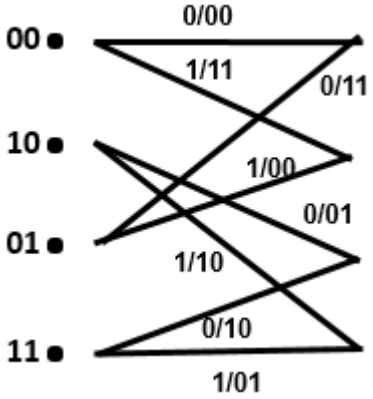


Step 8:

Viterbi Decoding Example

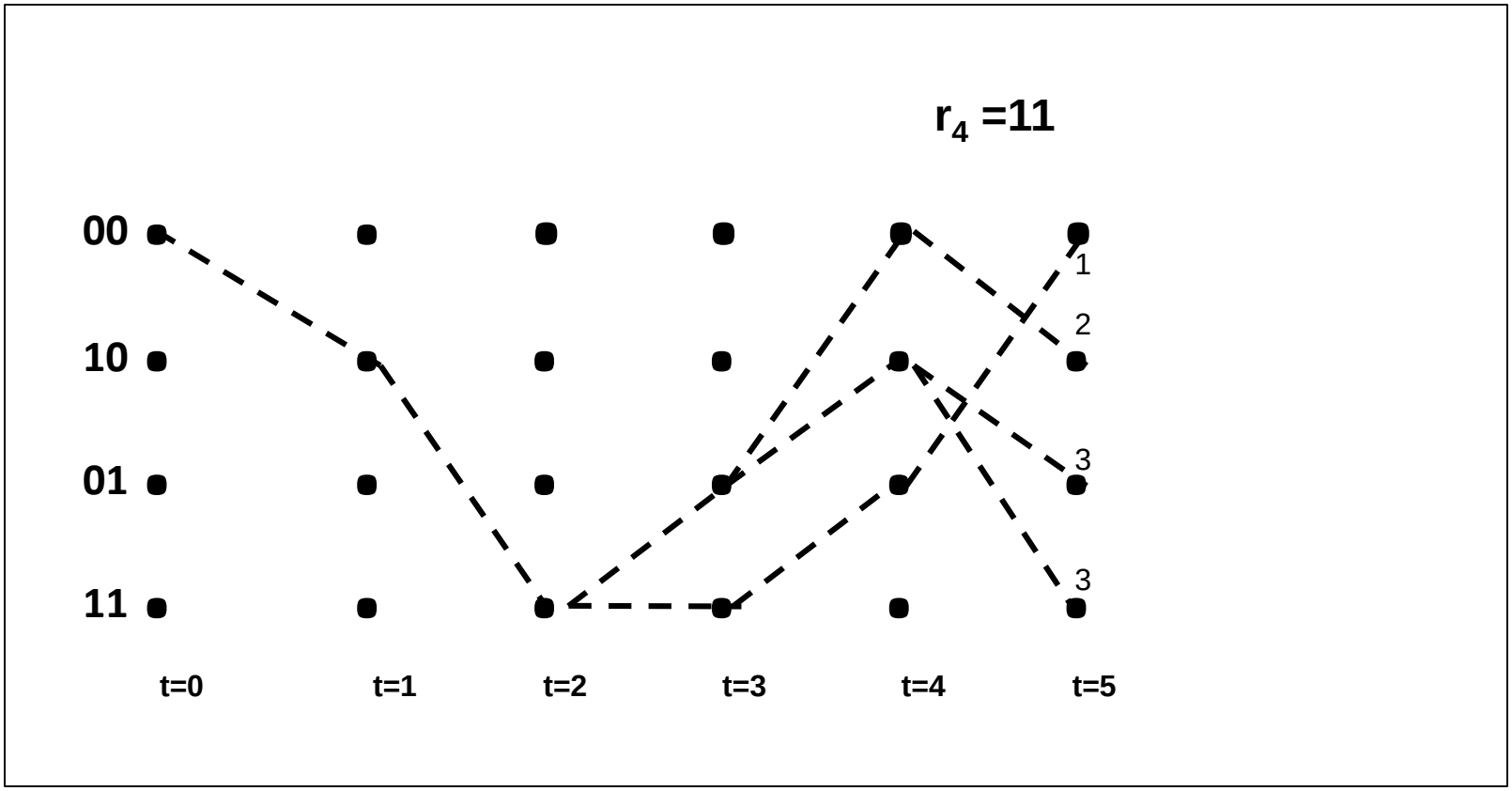


R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

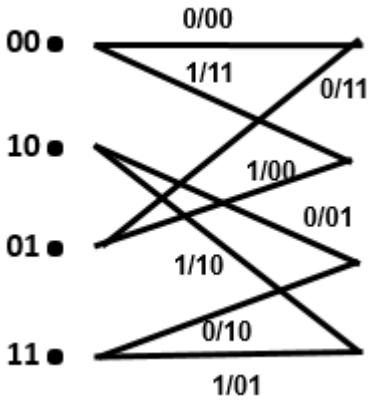


Step 9:

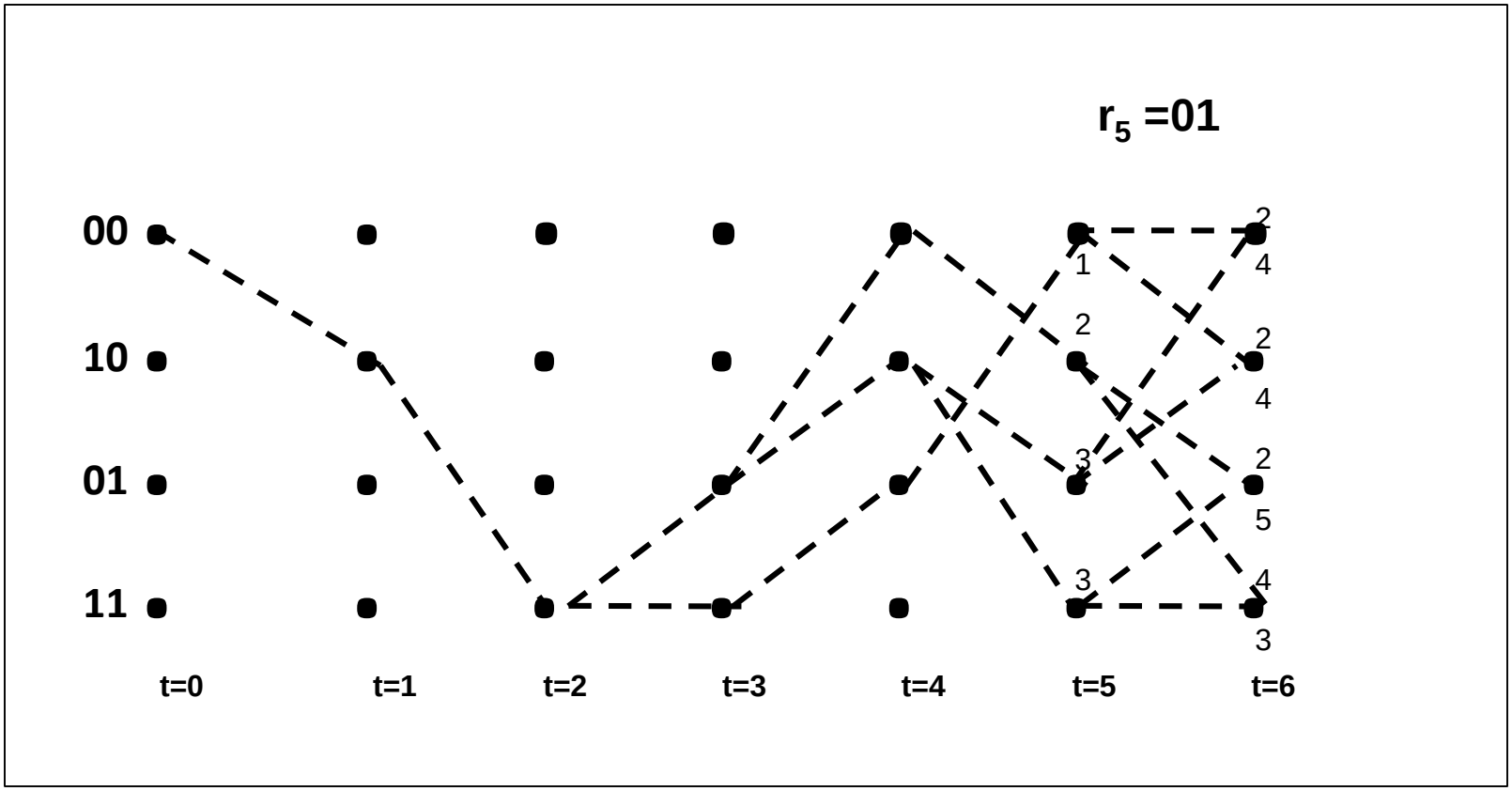
Viterbi Decoding Example



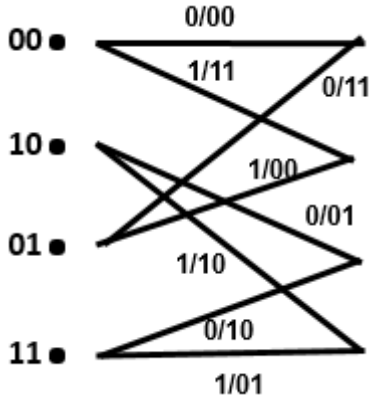
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



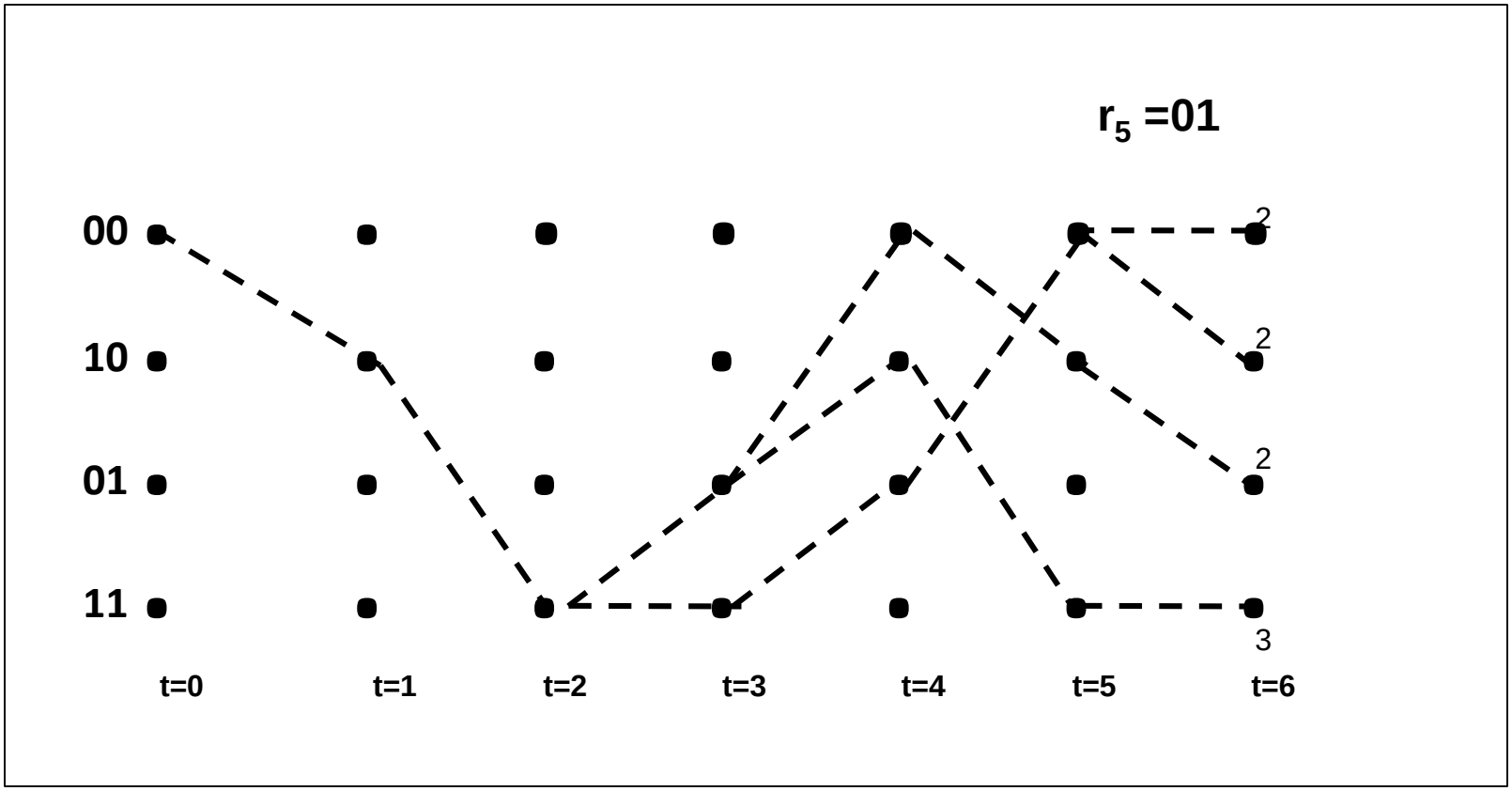
Step 10: Viterbi Decoding Example



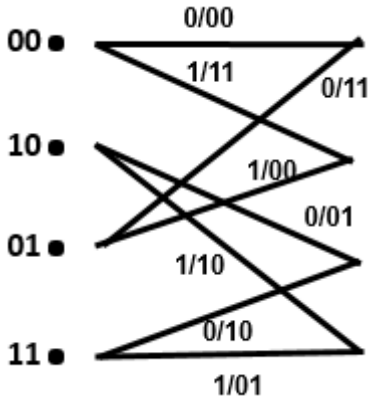
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



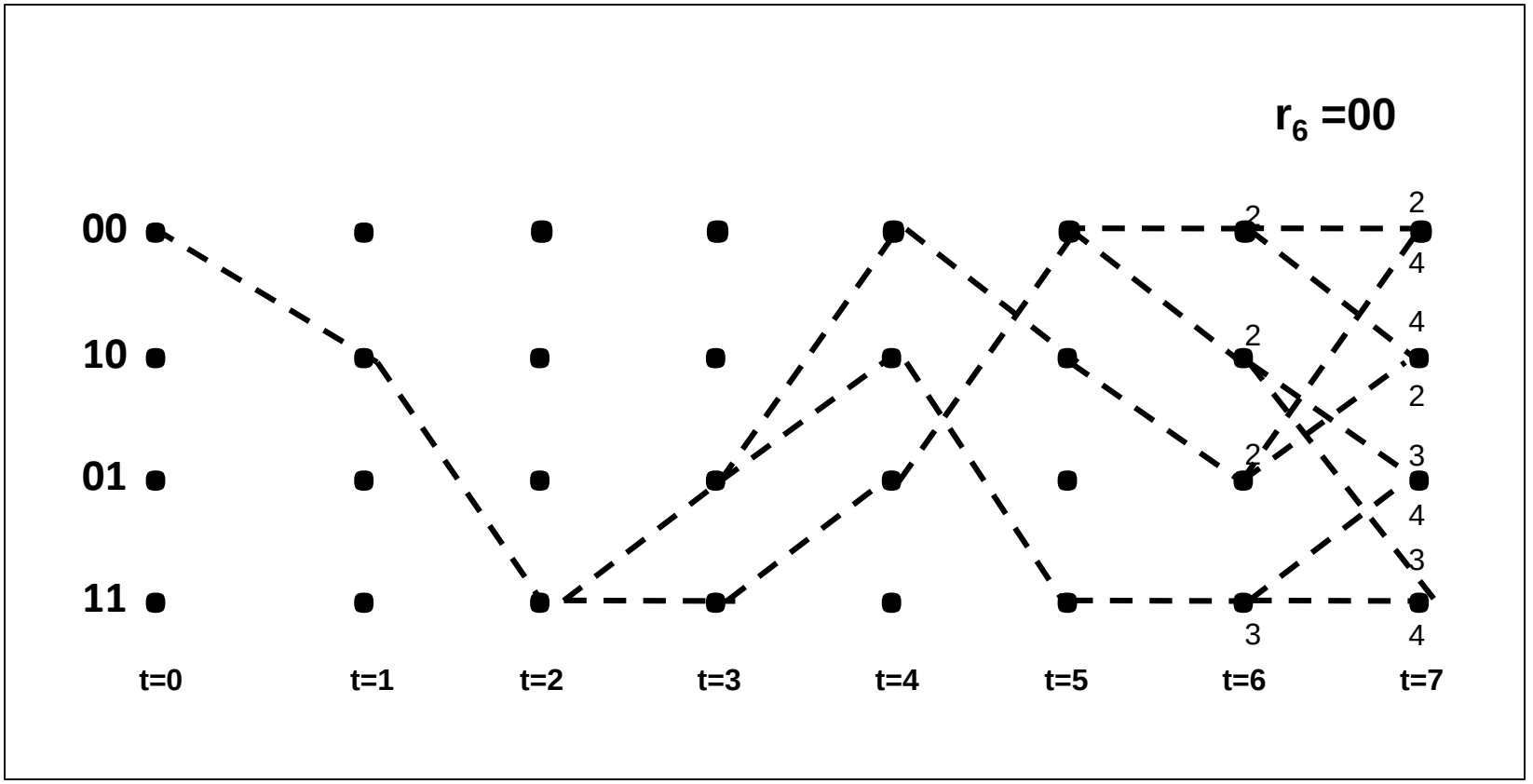
Step 11: Viterbi Decoding Example



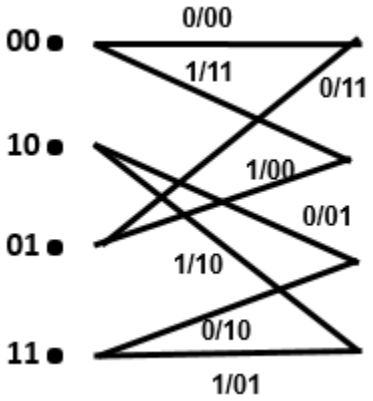
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



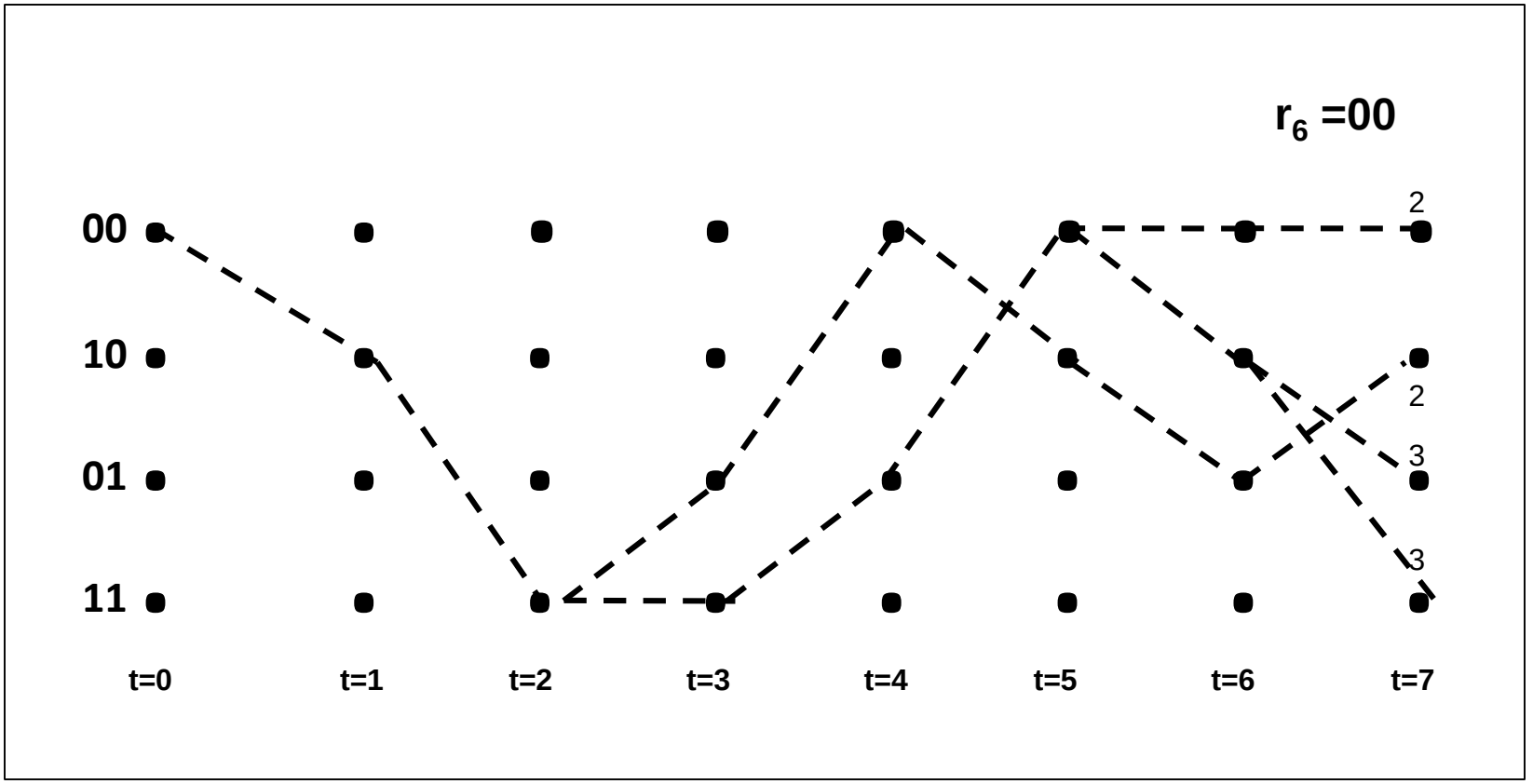
Step 12: Viterbi Decoding Example



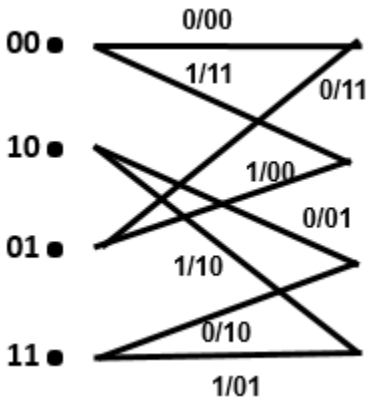
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



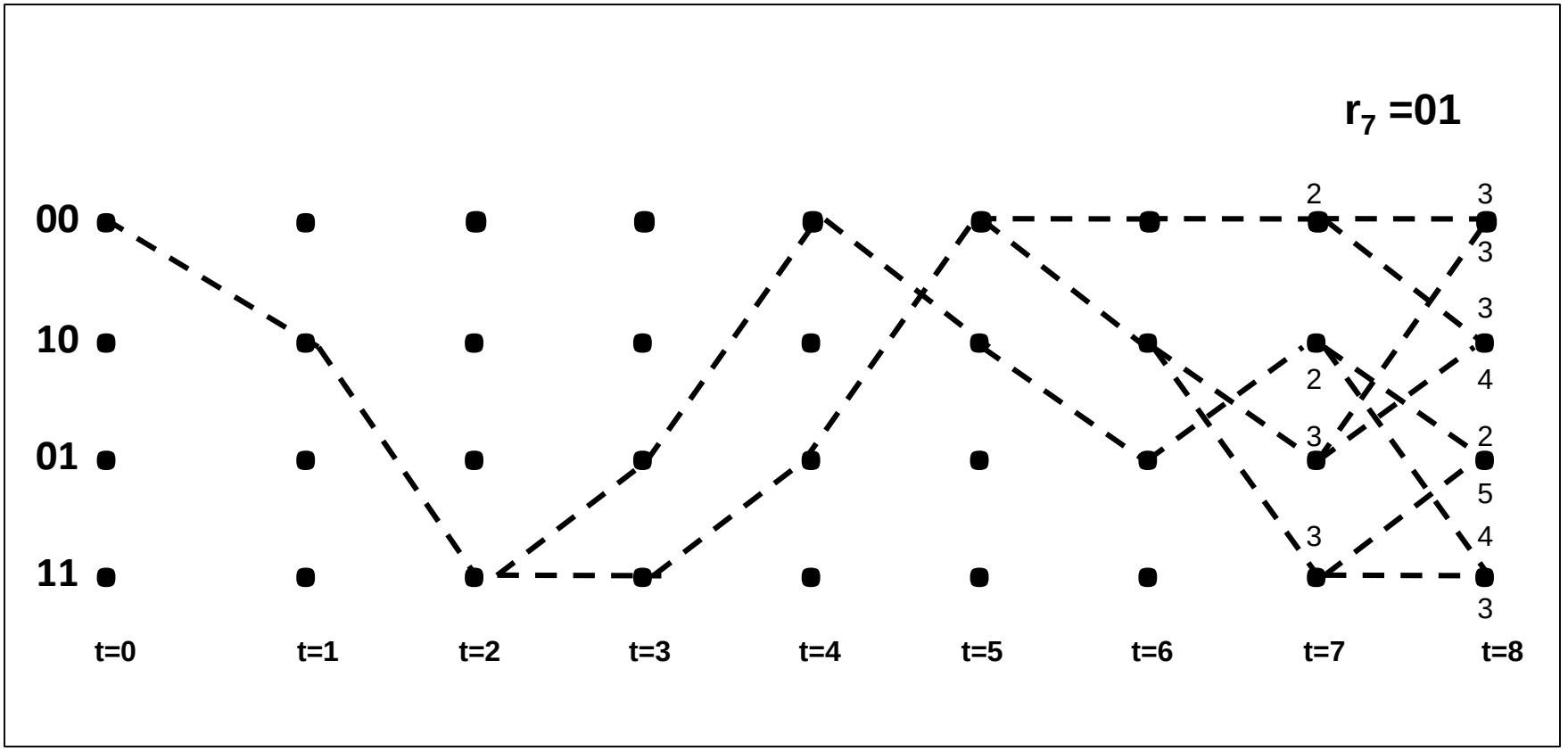
Step 13: Viterbi Decoding Example



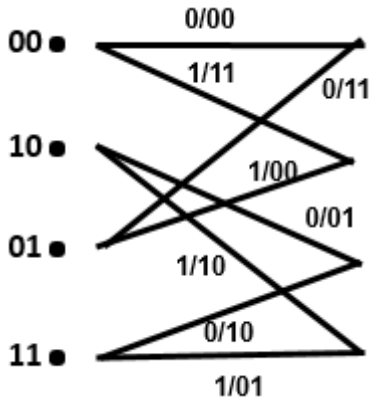
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



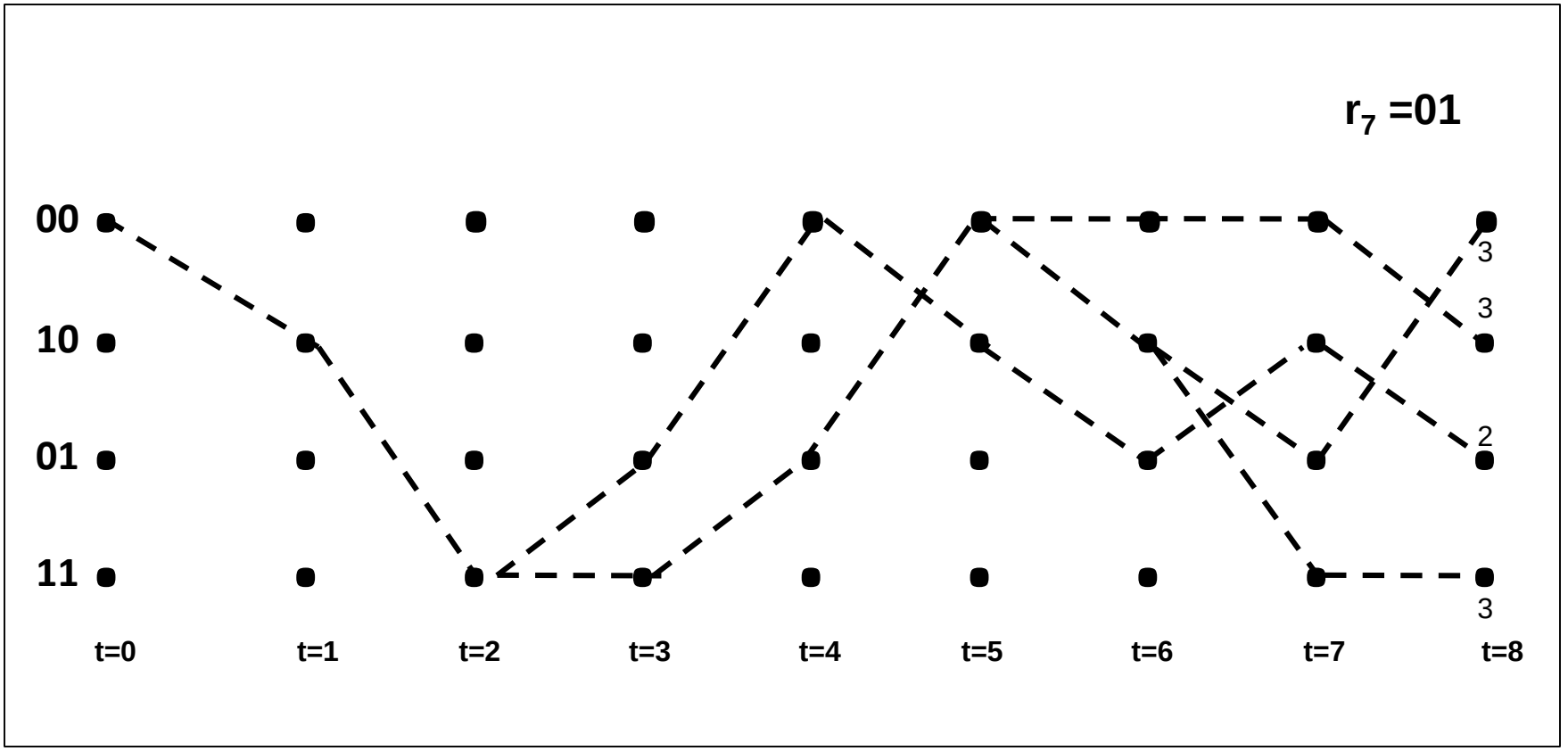
Step 14: Viterbi Decoding Example



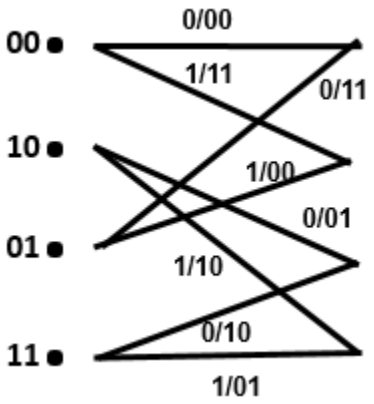
R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



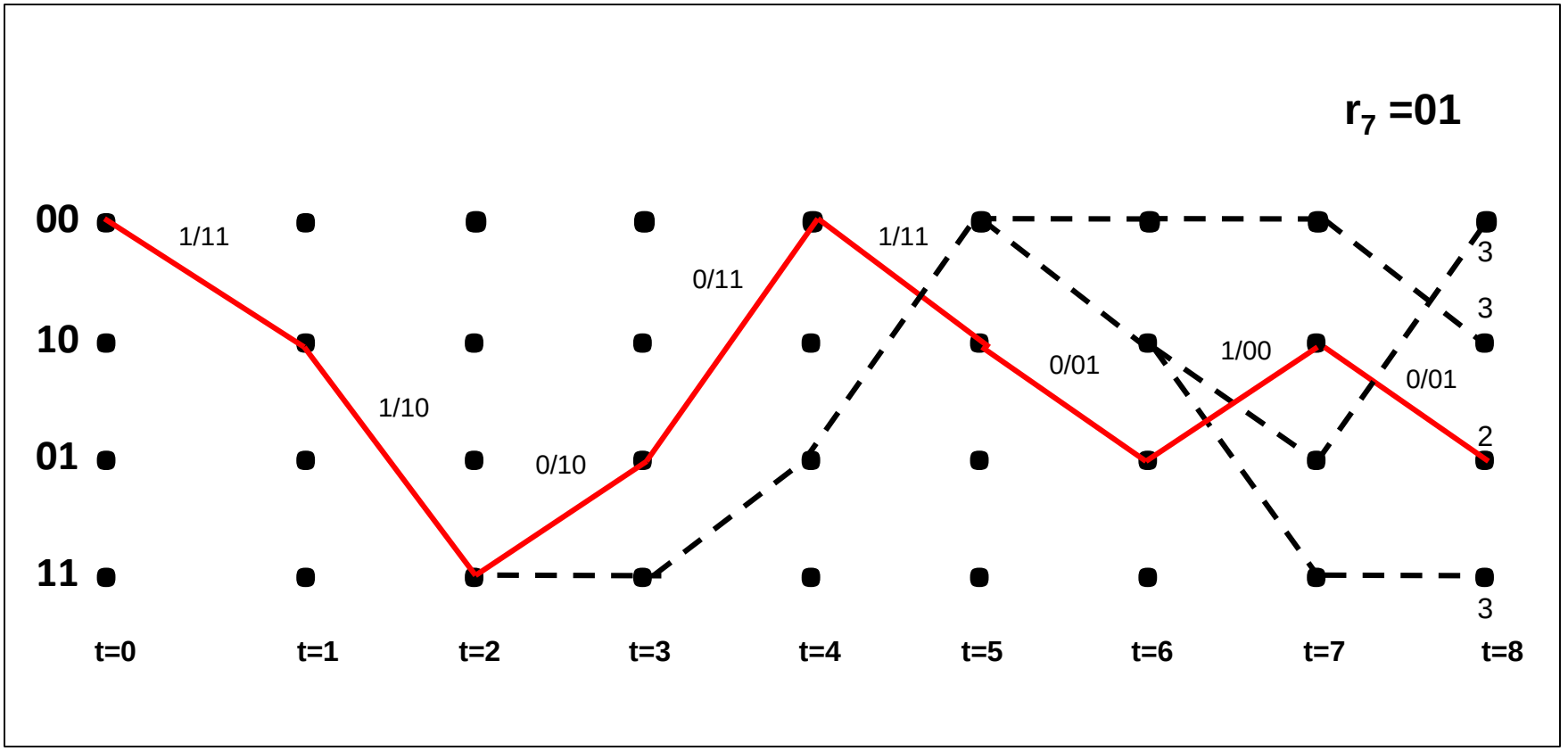
Step 15: Viterbi Decoding Example



R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]



Step 16: Viterbi Decoding Example



R = [11 10 <u>0</u>0 1<u>0</u> 11 01 00 01]

- Decoded output bit sequence is

  x=[11 10 10 11 11 01 00 01].

Decoded data sequence **1 1 0 0 1 0 1 0**