GEL7114

Module 5

Chapter 6: Introduction to FEC

Topics covered

- Time coding and orthogonal codes (Sec. 6.1-6.1.3.1)
- Channel models, parity, code rate and redundancy (Sec. 6.3.1-6.3.3)
- Forward error correction (Sec. 6.3.4)

Coding

- To improve P_e
  - Accept expansion in bandwidth
  - Orthogonal, bi orthogonal, simplex
- To detect errors
  - Parity and retransmission
- To correct errors
  - Block codes
  - Convolutional codes
  - The best codes minimize bandwidth expansion

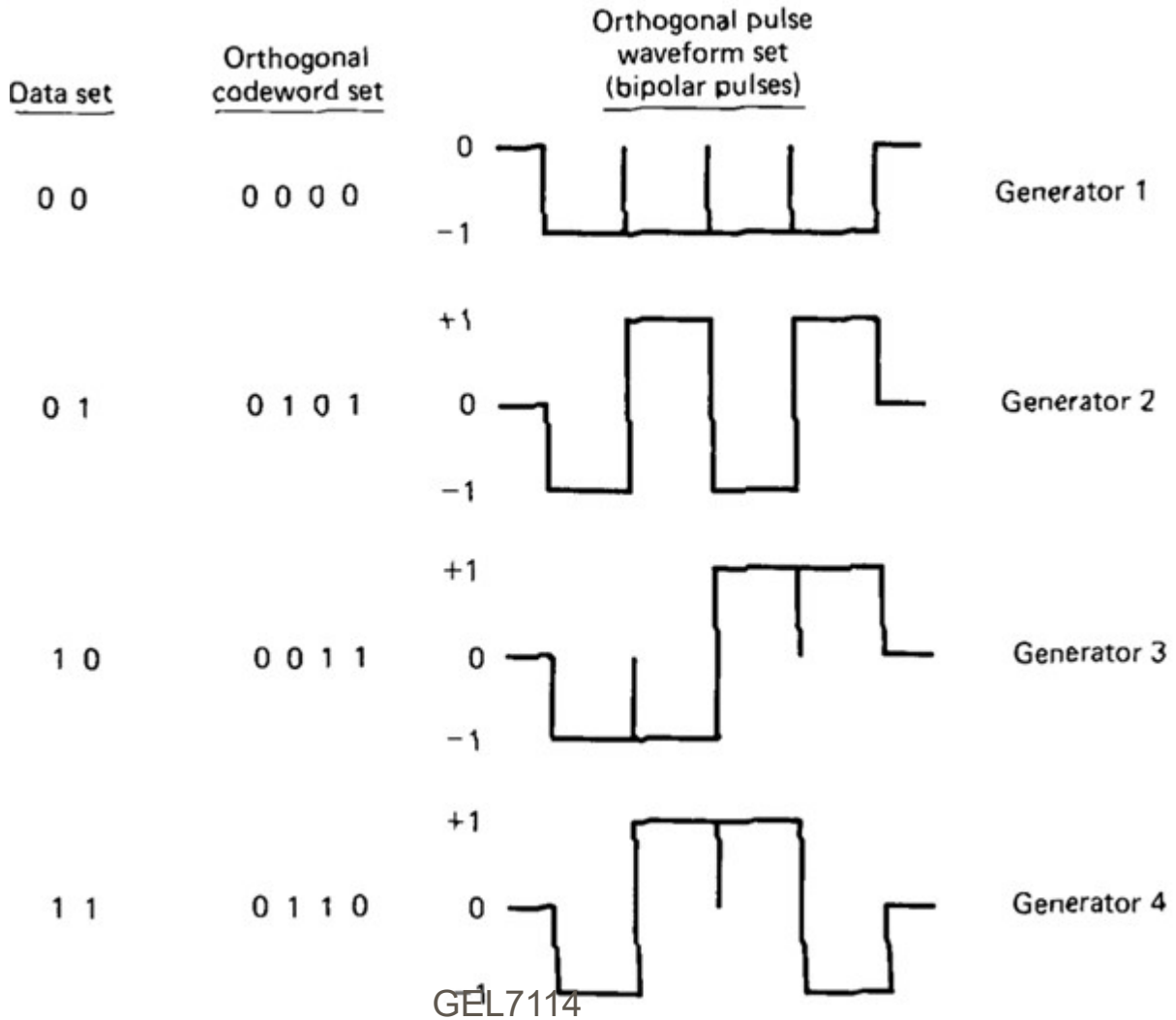
Code rate

- Block encoder
- Block of k data bits enter
  - Block of n coded bits exit
- Code redundancy $\frac{n-k}{k}$
- Code rate $\frac{k}{n}$
- Bandwidth expansion $\frac{n}{k}$

GEL7114

Hadamard Modulation

Orthogonal Hadamard Modulation



Cross correlation

- Piece-wise constant

  that is, just +1 et -1

$$z_{ij} = \frac{1}{E}\left\langle s_i, s_j \right\rangle$$

$$= \frac{\#\ \text{bits same - \# bits differents}}{\#\ \text{bits}}$$

bits same = # bits differents for Hadamard

Orthogonal Hadamard Modulation

- Improved performance

$$P_e(M) = (M-1)Q\left(\sqrt{\frac{E_s}{N_0}}\right) = (M-1)Q\left(\sqrt{\frac{E_b \log_2 M}{N_0}}\right)$$

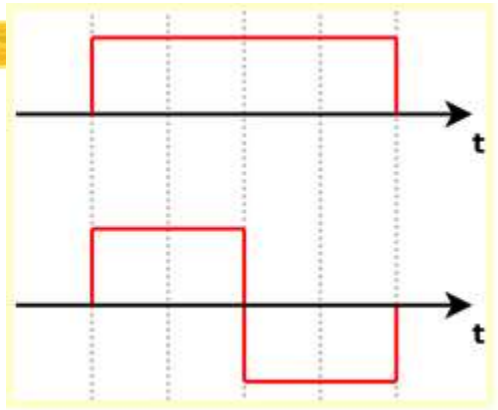
- BW inefficient

k bits sans codage

$M = 2^k$ bits avec codage

$$\eta_{\text{Hada}} = \frac{\log_2 M}{M} \text{ b/s/Hz}$$

Orthogonal codes



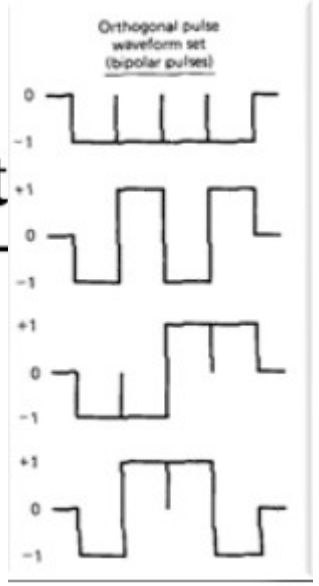
- Hadamard matrices
- Binary case $\left(k=1, M=2^1=2\right)$

$$H_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

  - First row 0 0 for logical zero
  - Second row 0 1 for logical one

Data set	Orthogonal codeword set
0 0	0 0 0 0
0 1	0 1 0 1
1 0	0 0 1 1
1 1	0 1 1 0

$$\mathbf{H}_2 = \left[\begin{array}{cc:cc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hdashline 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] = \left[\begin{array}{cc} \mathbf{H}_1 & \mathbf{H}_1 \\ \mathbf{H}_1 & \overline{\mathbf{H}_1} \end{array}\right]$$



Hadamard Matrices

$$H_{n+1} = \begin{bmatrix} H_n & H_n \\ H_n & \bar{H}_n \end{bmatrix}$$

- For any pair of rows
  - #bits different = #bits same

$$\boxed{z_{ij} = \frac{1}{E}\langle s_i, s_j \rangle = 0}$$

$$\eta_{\text{ortho}} = \frac{\log_2 M}{M} \text{ b/s/Hz}$$

ta set

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

Orthogonal codeword set

$$\mathbf{H}_3 = \left[\begin{array}{cccc:cccc} 0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&1&0&0&1&1&0 \\ \hdashline 0&0&0&0&1&1&1&1 \\ 0&1&0&1&1&0&1&0 \\ 0&0&1&1&1&1&0&0 \\ 0&1&1&0&1&0&0&1 \end{array}\right] = \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_2 \\ \mathbf{H}_2 & \overline{\mathbf{H}_2} \end{bmatrix}$$

Probability of error

- Orthogonal Codes

$$P_e = \text{symbol error probability}$$

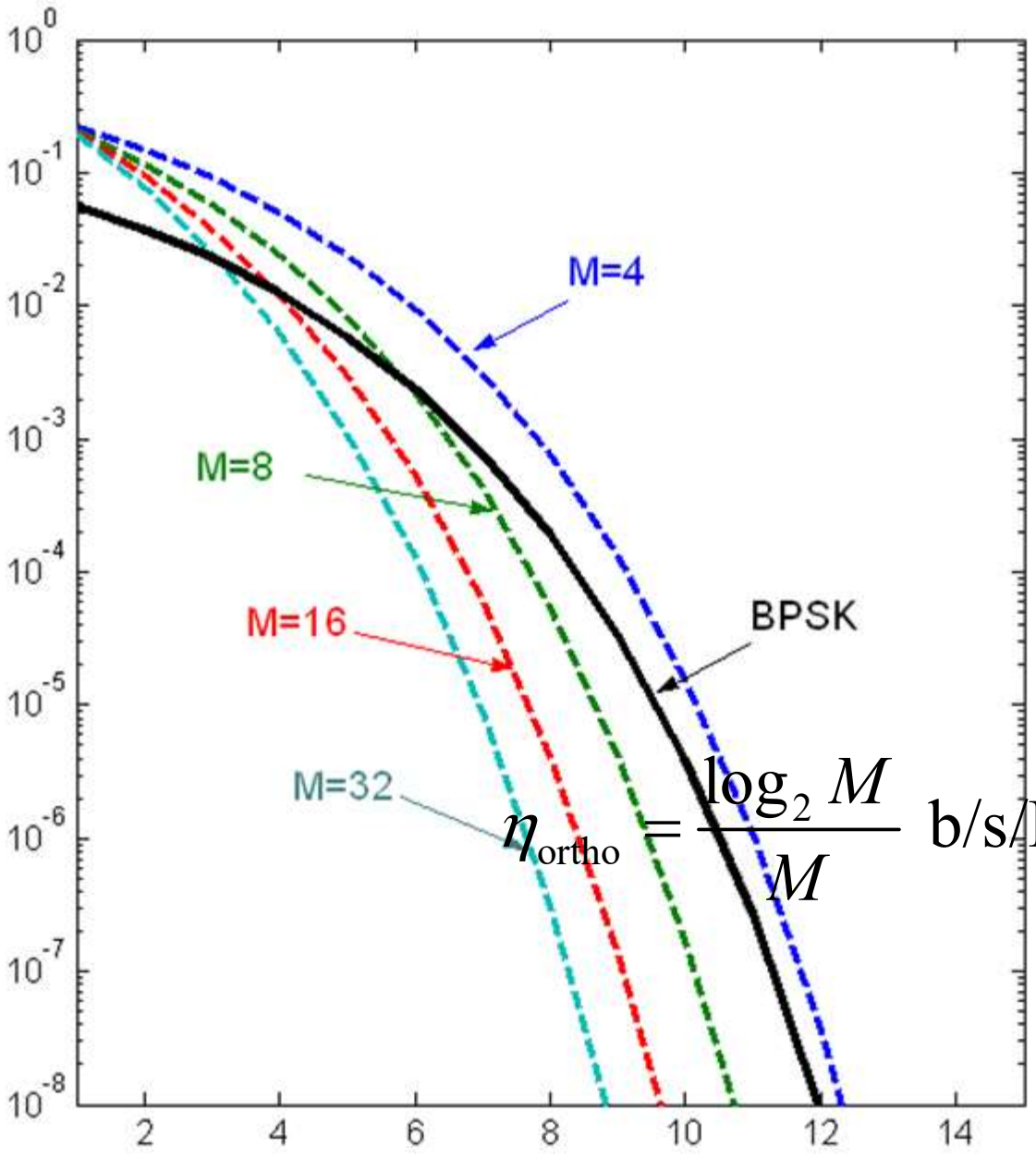
$$P_e(M) = (M-1)Q\left(\sqrt{\frac{E_s}{N_0}}\right) = (M-1)Q\left(\sqrt{\frac{E_b \log_2 M}{N_0}}\right)$$

$$\frac{P_b}{P_e} = \frac{M/2}{M-1}$$

$$P_b \approx \frac{M}{2} Q\left(\sqrt{k\frac{E_b}{N}}\right)$$

Performance

$$P_e(M) = (M-1)Q\left(\sqrt{\frac{E_b \log_2 M}{N_0}}\right)$$



$$\eta_{\text{ortho}} = \frac{\log_2 M}{M} \text{ b/s/Hz}$$

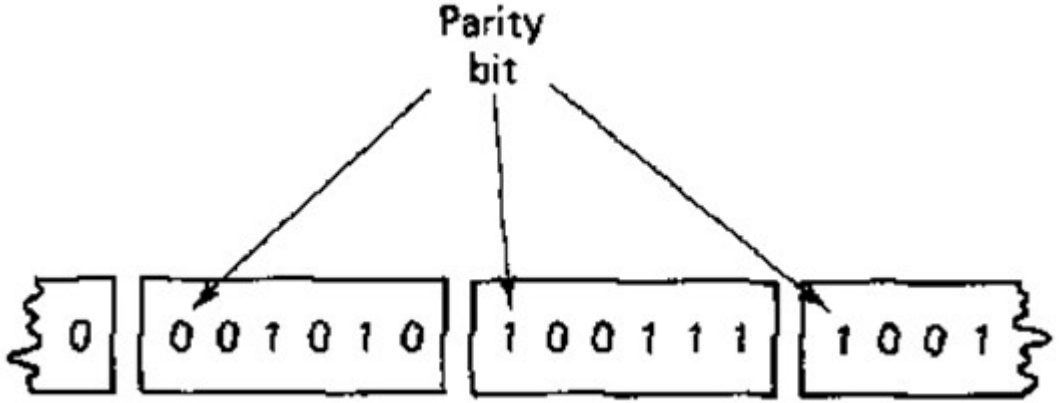
GEL7114

parity

Parity

- Identify errors, but do not correct them

  …
- Even and odd parity
- Rate $k/k+1$
  - Detects an odd number of errors



Probability

- How many positions for j errors in a block of n bits

$$\binom{n}{j} = \frac{n!}{j!(n-j)!}$$

- Probability of j errors in n bits

$$\Pr(j,n) = \binom{n}{j} p^j (1-p)^{n-j}$$

- Probability of missing an error

$$P_{nd} = \sum_{j=1}^{\lfloor n/2 \rfloor} \binom{n}{2j} p^{2j} (1-p)^{n-2j}$$

i.e., even # of errors

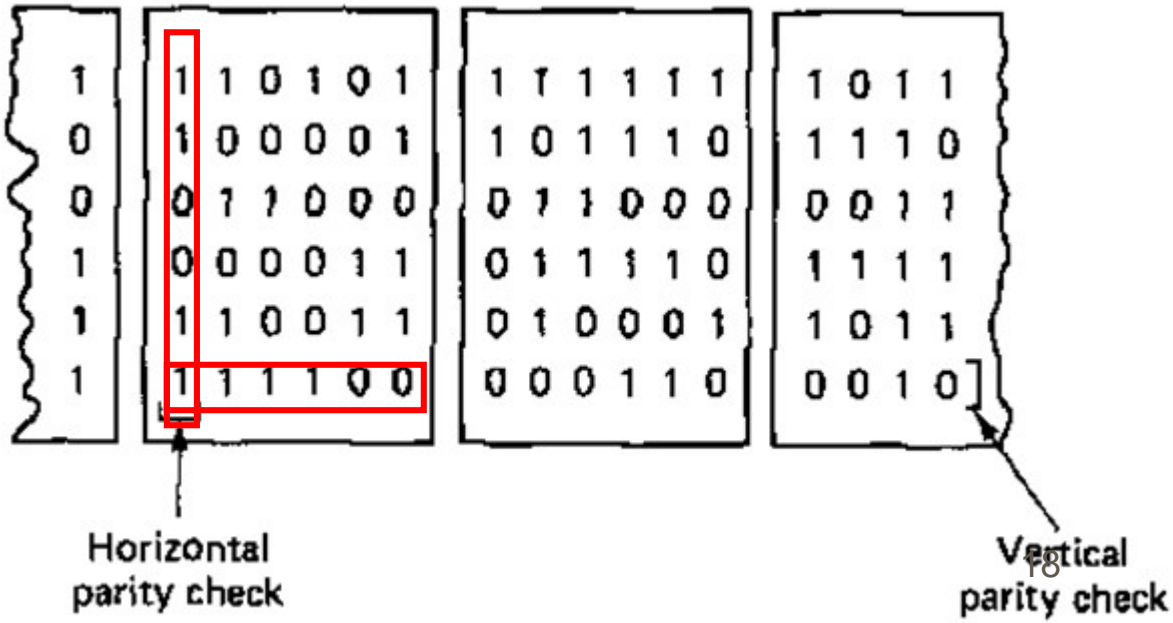
Vertical and horizontal parity

- Two-dimensional parity, block of $M \times N$ bits
- Add a bit to each row and column
  - Rate

$$\frac{MN}{(M+1)(N+1)}$$

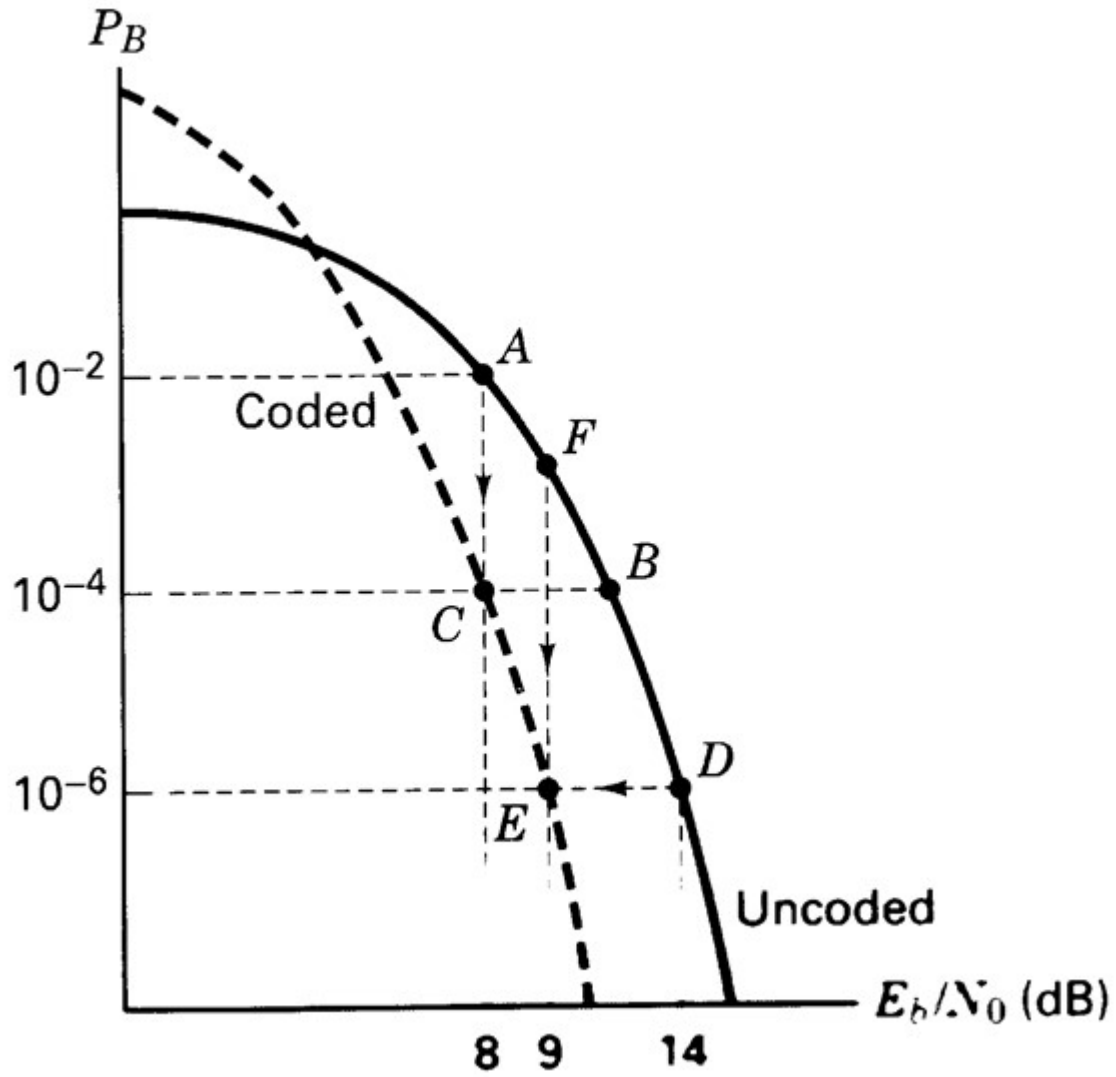
$$= \frac{MN}{MN+N+M+1}$$

  - Error correction possible



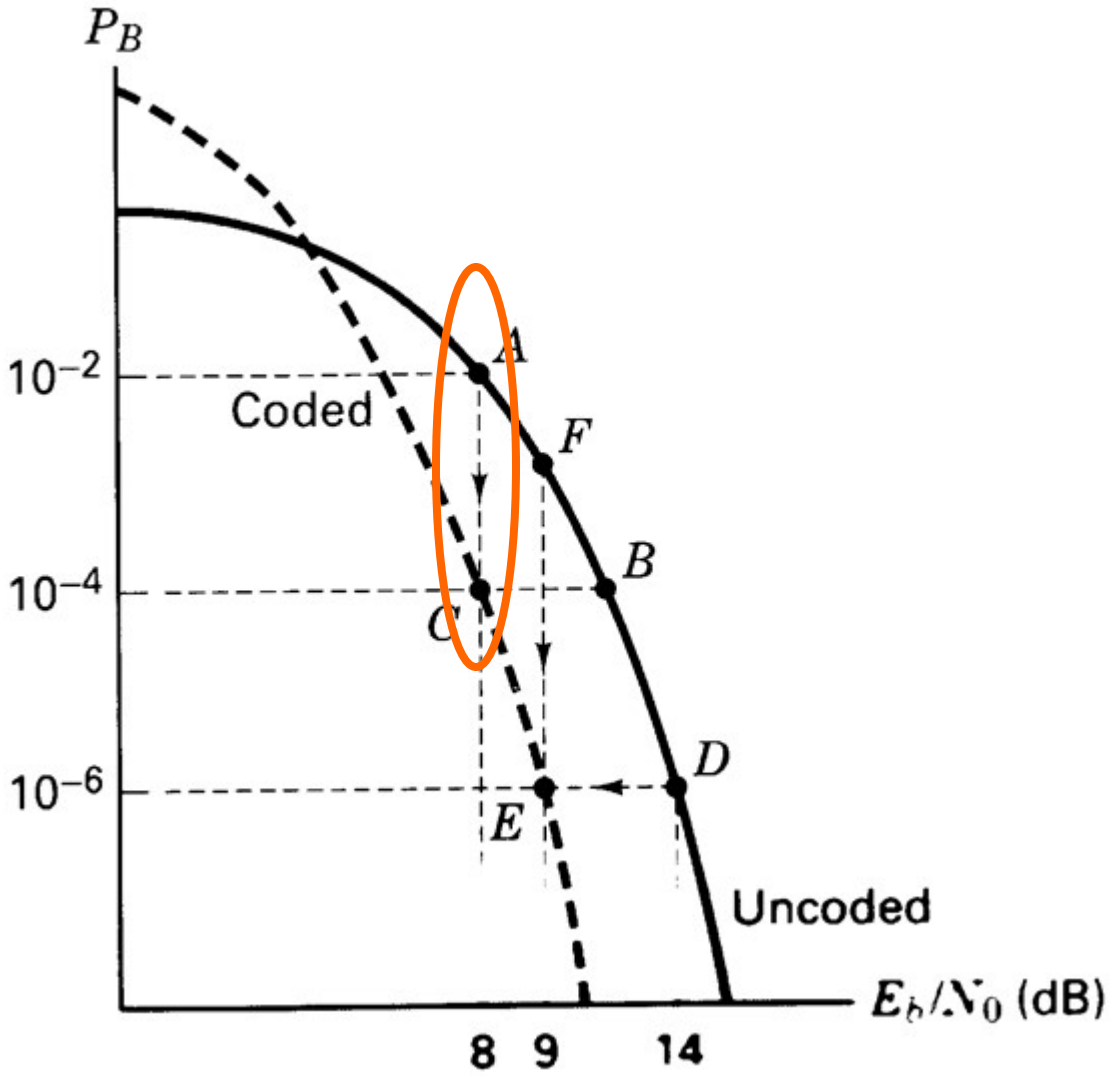
Forward error correction (FEC)

Performance



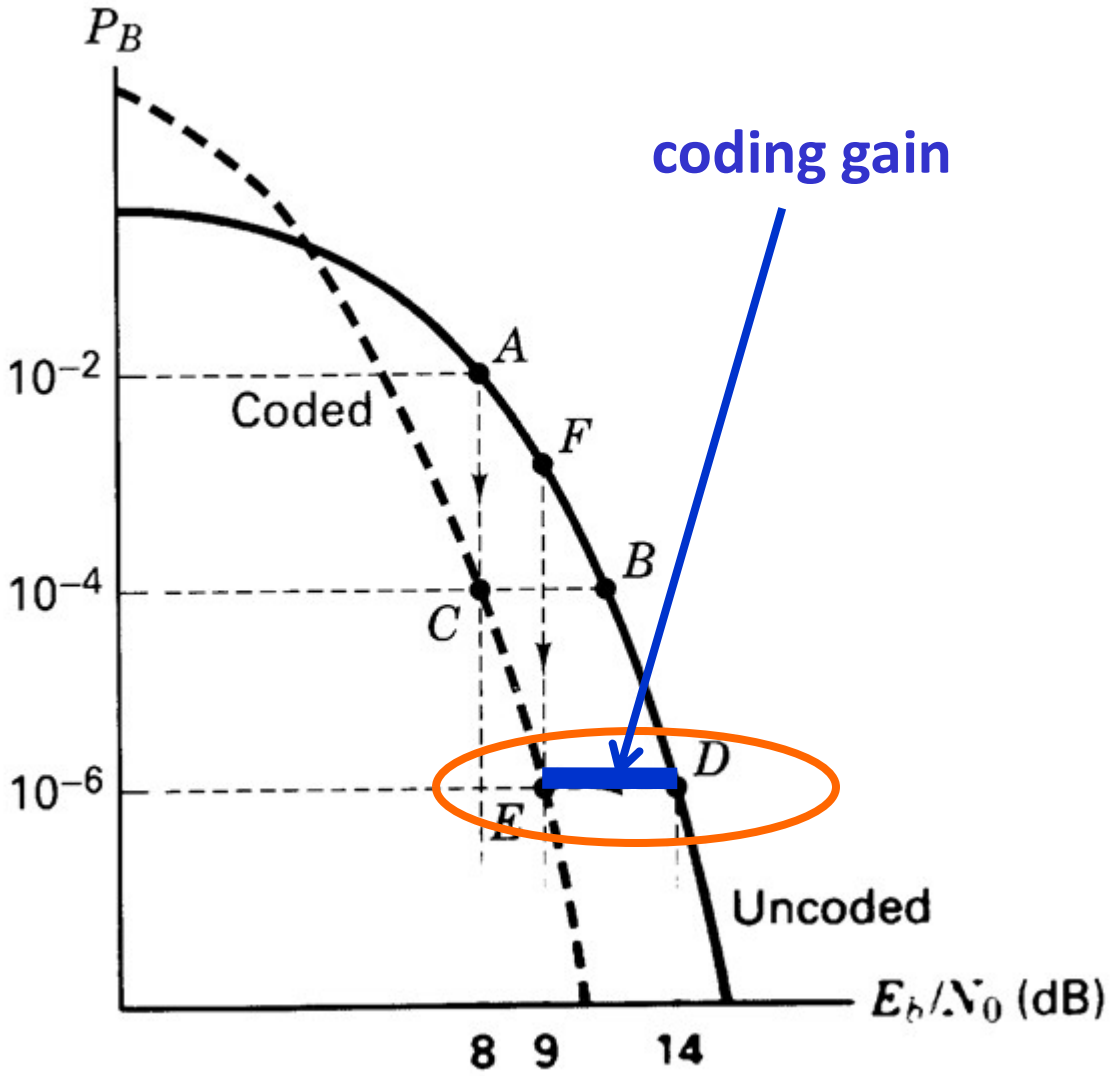
Performance

Compromise:
less Power,
but less bandwidth
efficient



Performance

Compromise:
less Power,
but less bandwidth
efficient

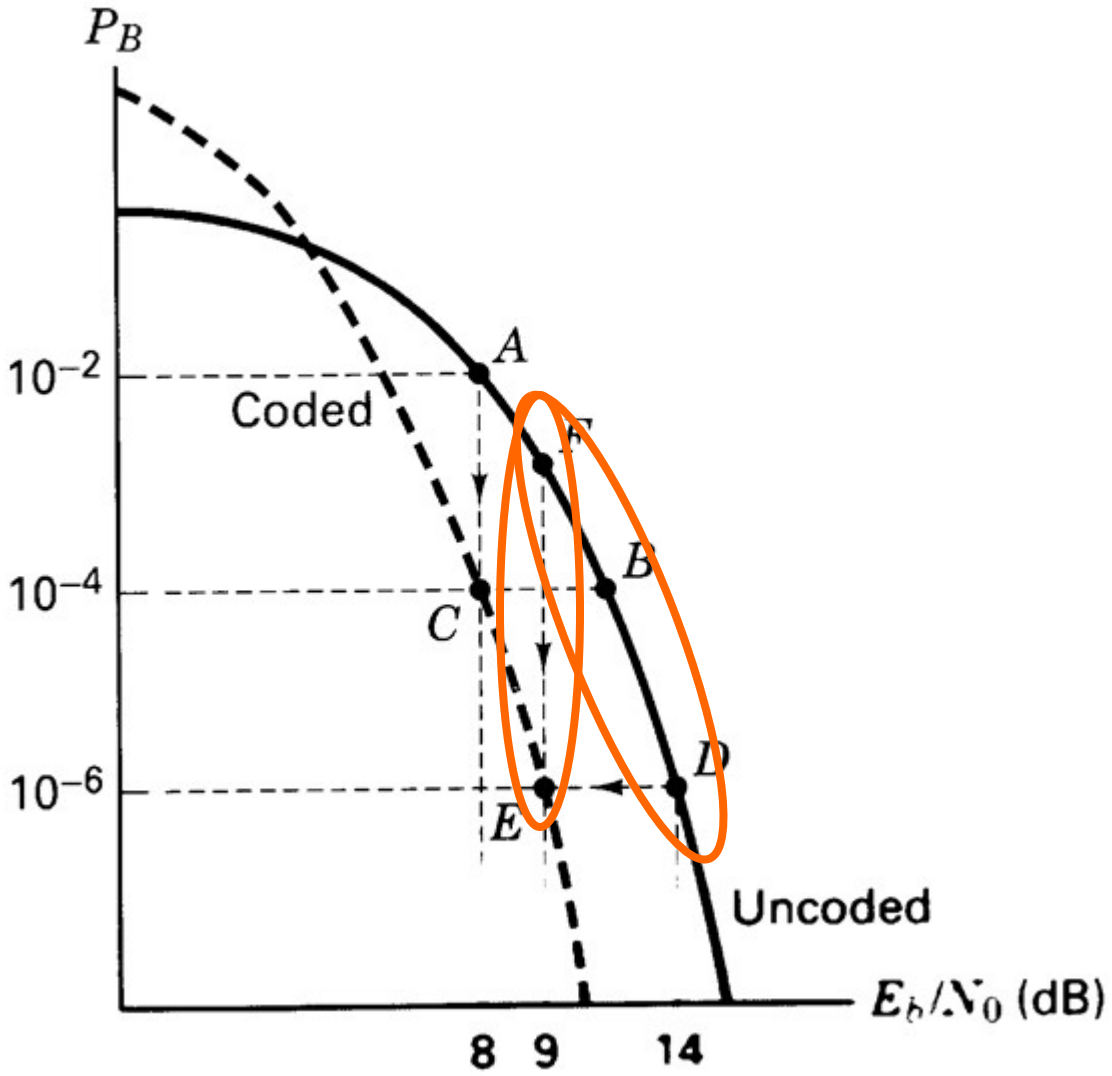


Performance

D□F with R↑

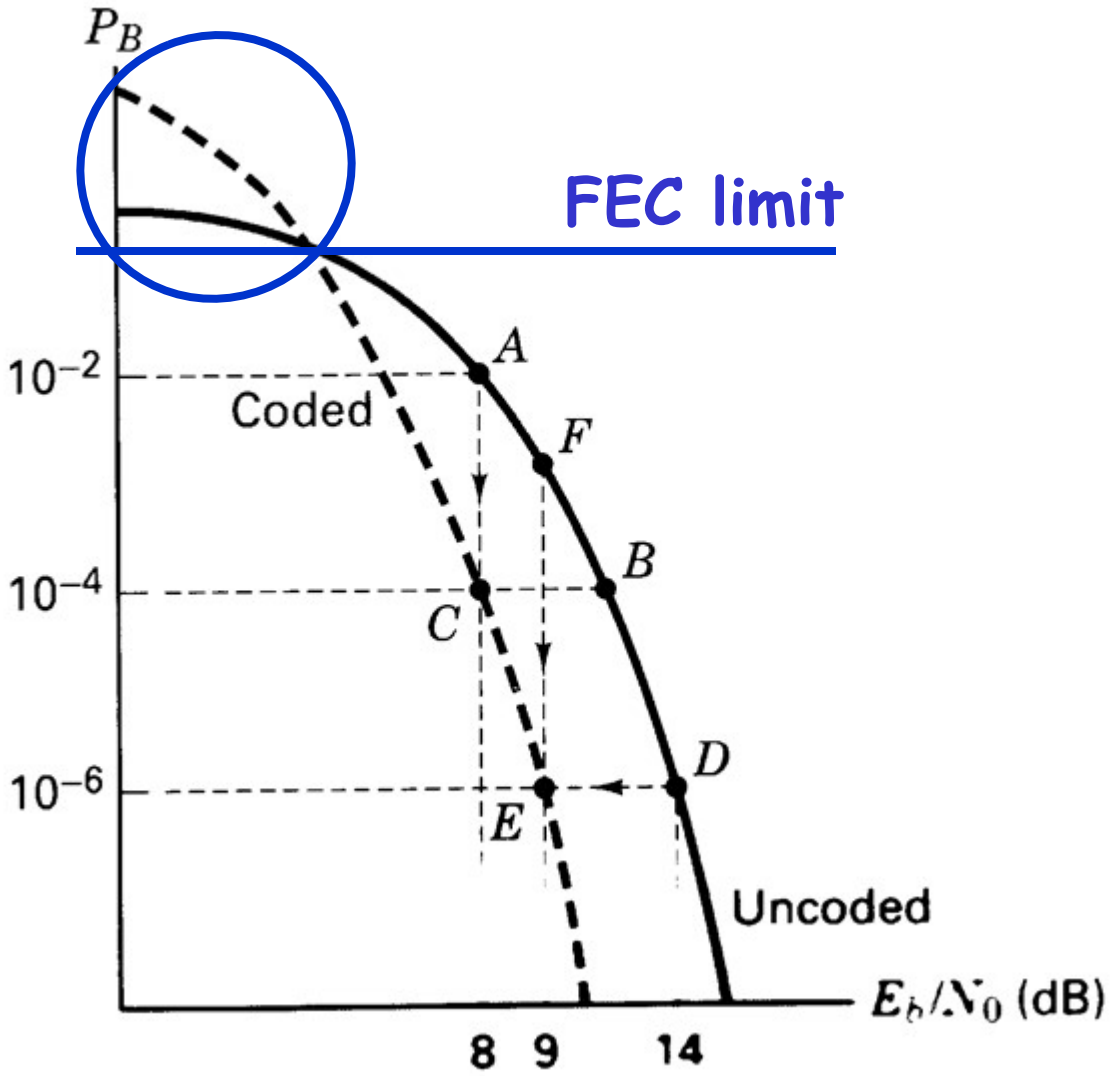
F□E with BW↑

Compromise:
less Power,
but less bandwidth
efficient



???

How can coding make performance worse???



Example

- $\frac{E_b}{N_0} = 9.6 \text{ dB} = 9.12 \quad \Rightarrow \quad p_u = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) = 1.02 \times 10^{-5}$
- Code $(15,11)$ corrects 1 error per block
- Bandwidth expansion

$$\frac{15}{11} \Rightarrow 36\%$$

- BER improvement???

Errors in a block

- Probability to have AT LEAST one error in k bits, without coding

$$P_M^u = 1-\left(1-p_u\right)^k = 1-\left(1-10^{-5}\right)^{11} = 1.12\times10^{-4}$$

BW Expansion

- Bandwidth expansion has an impact on average energy per bit
- $$\frac{E_C}{N_0} = \frac{E_B}{N_0}\frac{k}{n}$$
- Our example

$$\frac{E_C}{N_0} = 9.6 \text{ dB} + 10\log_{10}\frac{11}{15} = 8.3 \text{ dB} = 6.69$$

After error correction

- BER for coded block $p_c = Q\left(\sqrt{\frac{2E_C}{N_0}}\right) = 1.36\times10^{-4}$
- BER worsened before starting the correction process…
- A block with a single error will be corrected, but not others blocks $P_M = \sum_{j=2}^{n}\binom{n}{j}p_c^j\left(1-p_c\right)^{n-j} \approx \binom{n}{2}p_c^2\left(1-p_c\right)^{n-2}$
- Our example $P_M^c \approx \binom{15}{2}\left(1.4\times10^{-4}\right)^2\left(1-1.4\times10^{-4}\right)^{13} = 1.94\times10^{-6}$
-

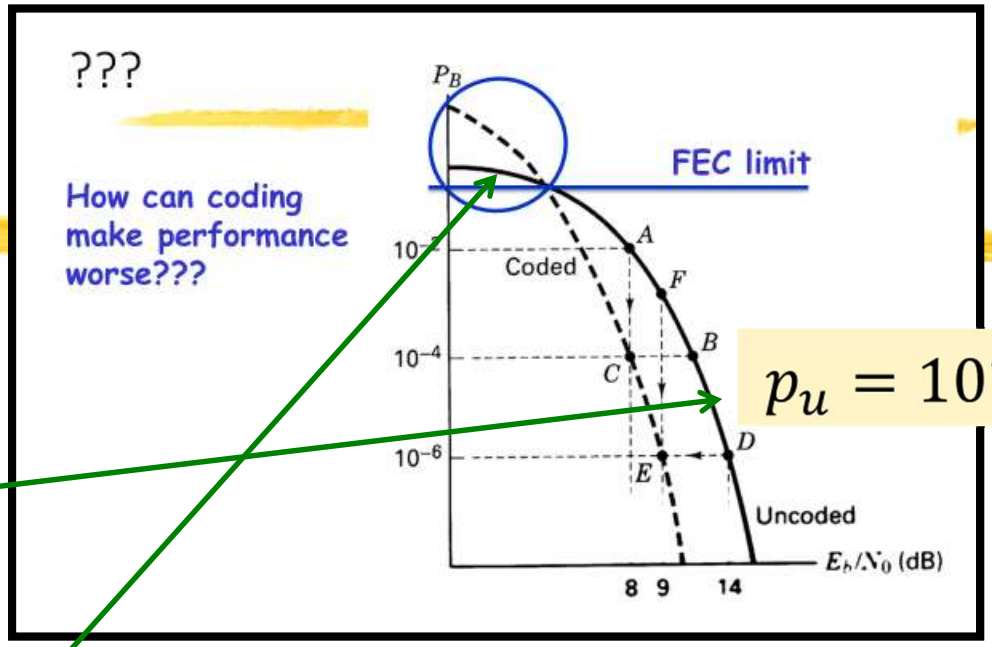
Coding gain?

- Yes!
- In a block of 11 bits …

$$P_M^u = 1-\left(1-p_u\right)^k = 1-\left(1-10^{-5}\right)^{11} = 1.12\times10^{-4}$$

$$P_M^c \approx \binom{15}{2}\left(1.4\times10^{-4}\right)^2\left(1-1.4\times10^{-4}\right)^{13} = 1.94\times10^{-6}$$

- Improvement of 60× for 36% expansion of BW



We were there

What if we were here?

After error correction

- BER for coded block $p_c = Q\left(\sqrt{\frac{2E_C}{N_0}}\right) = 1.36\times10^{-4}$
- BER worsened before starting the correction process...
- A block with a single error will be corrected, but not others blocks $P_M = \sum_{j=2}^{n}\binom{n}{j}p_c^j\left(1-p_c\right)^{n-j} \approx \binom{n}{2}p_c^2\left(1-p_c\right)^{n-2}$
- Our example $P_M^c \approx \binom{15}{2}\left(1.4\times10^{-4}\right)^2\left(1-1.4\times10^{-4}\right)^{13} = 1.94\times10^{-6}$
-

this probability would have been terrible

that would result in a deterioration, not an improvement...

GEL7114

Chapter 7

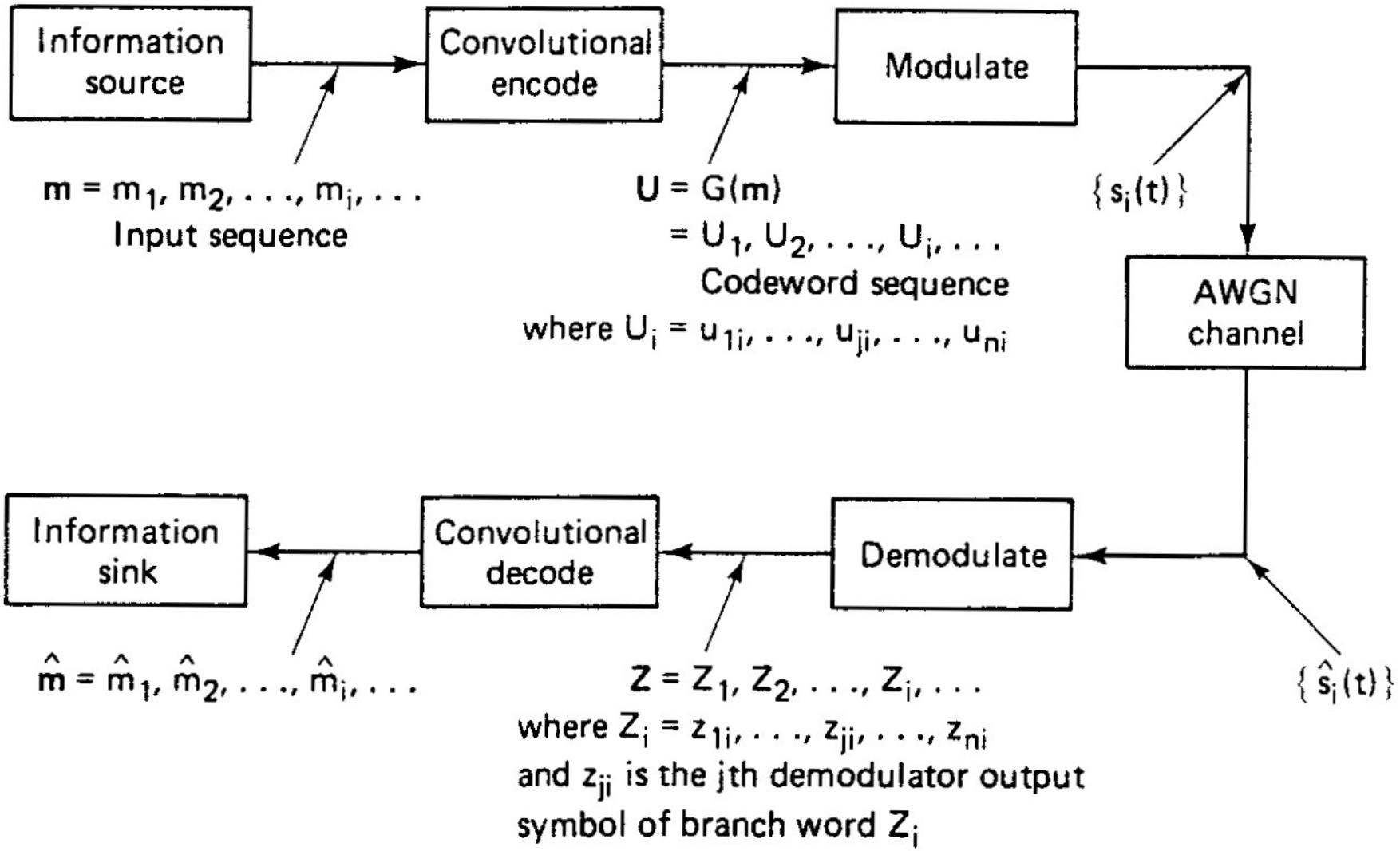
Topics covered

- Convolutional encoding (Sec. 7.1)
- Representation of encoders (Sec. 7.2)
- The decoding algorithm (Sec. 7.3)
- Properties of convolutional codes (Sec. 7.4)

Block vs. convolutional coding

- Same code rate k/n
- Block coding
  - isolated and independent block
  - no "memory" from one block to another
- Convolutional encoding
  - continuously encoded data
  - memory of K-1 blocks of k bits

Constraint length


Information source
Convolutional encode
Modulate
AWGN channel
Demodulate
Convolutional decode
Information sink
m = m1, m2, . . ., mi, . . .
Input sequence
U = G(m)
= U1, U2, . . ., Ui, . . .
Codeword sequence
where Ui = u1i, . . ., uji, . . ., uni
{si(t)}
{ŝi(t)}
Z = Z1, Z2, . . ., Zi, . . .
where Zi = z1i, . . ., zji, . . ., zni
and zji is the jth demodulator output symbol of branch word Zi
m̂ = m̂1, m̂2, . . ., m̂i, . . .

Convolutional encoders

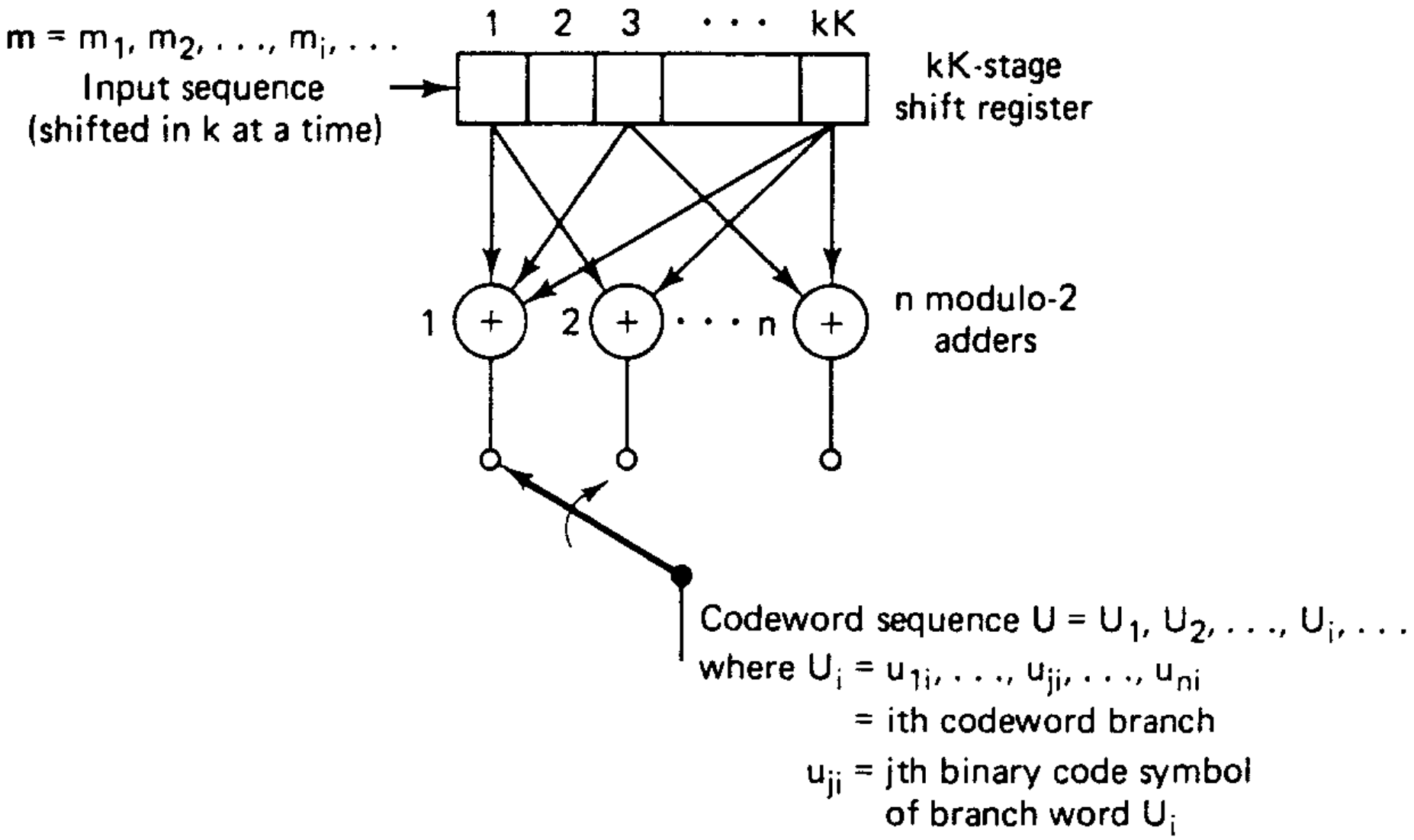
- Block diagram of shift registers
- Connection vectors and polynomials interpretation
- Impulse response
- State diagram
- Tree representation
- Trellis representation

Why …

- Physical implementations
- Mathematical manipulations
- Time behavior

Block diagram of shift registers

- $k \times K$ registers
- Blocks of *k* symbols enter and blocks of *n* symbols come out of the encoder
- Output a symbol function at the input
- Interconnections of shift registers determine the function
- We examine the case $k=1$


m = m1, m2, . . ., mi, . . .
Input sequence
(shifted in k at a time)
1 2 3 · · · kK
kK-stage
shift register
1 + 2 + · · · n +
n modulo-2
adders
Codeword sequence U = U1, U2, . . ., Ui, . . .
where Ui = u1i, . . ., uji, . . ., uni
= ith codeword branch
uji = jth binary code symbol
of branch word Ui

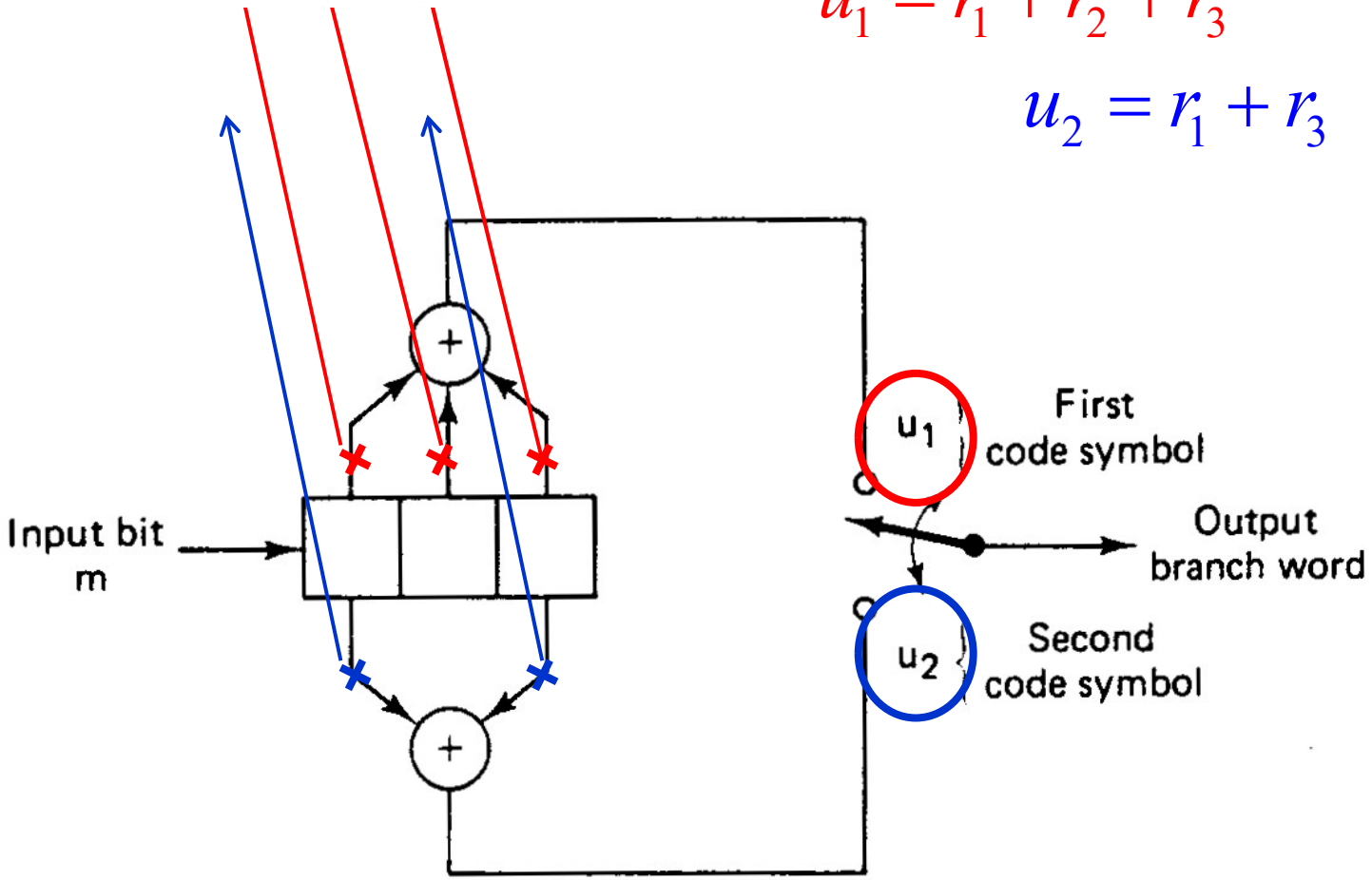
Connection vectors, polynomial interpretation

- The interconnections of the shift registers determines the code
- Let us use a vector of length $k \times K$ to specify the interconnections for each of the output symbols

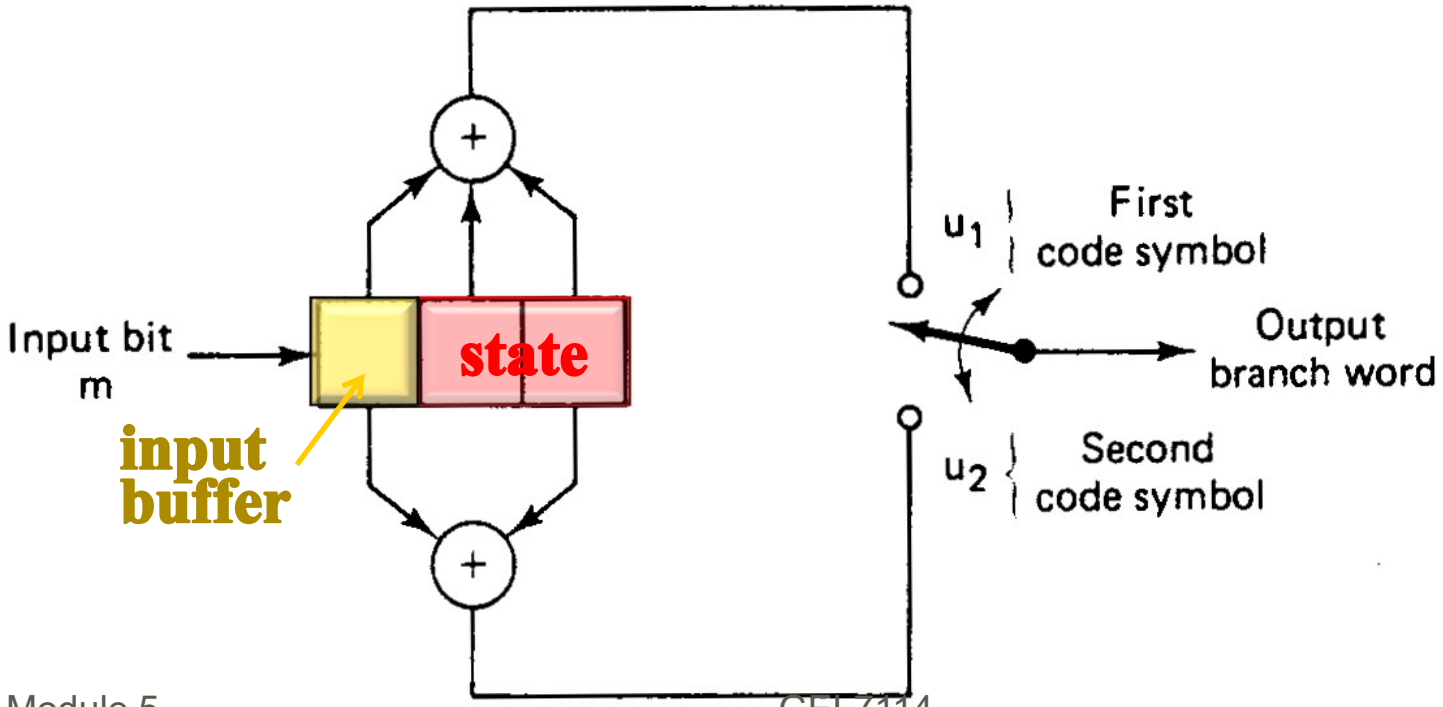
Exemple: $k=1,\ n=2,\ K=3$

$$u_1 = r_1 + r_2 + r_3$$

$$u_2 = r_1 + r_3$$



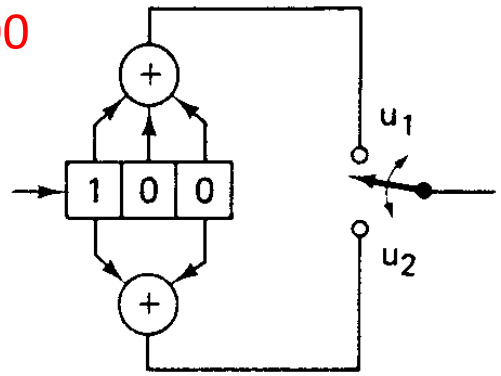
Exemple: $k=1,\ n=2,\ K=3$



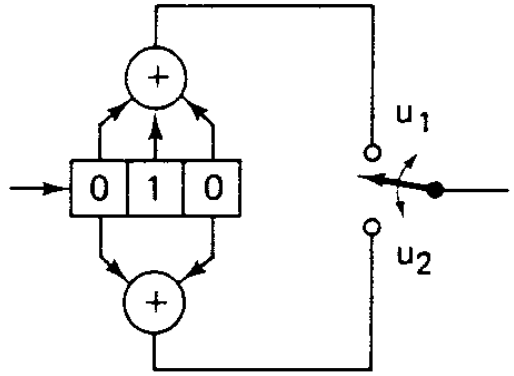
Initialization

- Output is a function of the original state
- Typically assume that the registers are empty (0) to begin with
- Example : send message $m=101$
  - preceded by zeros
  - followed by zeroes

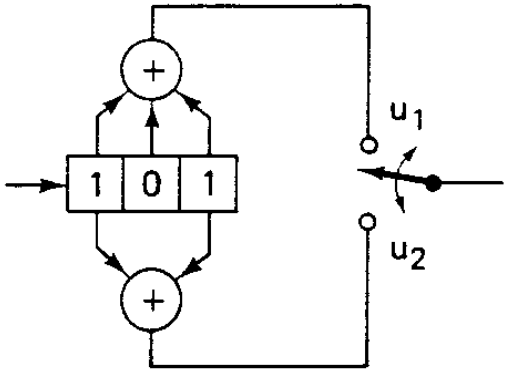
0010100



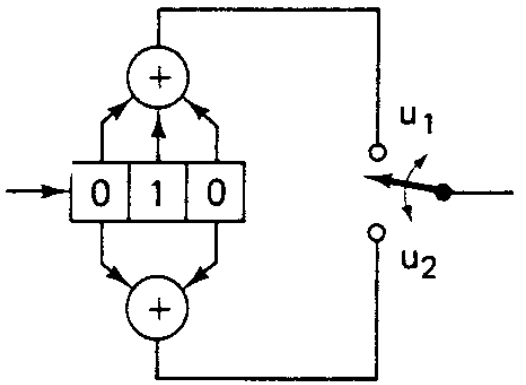
t_1 u_1 u_2 = 1 1



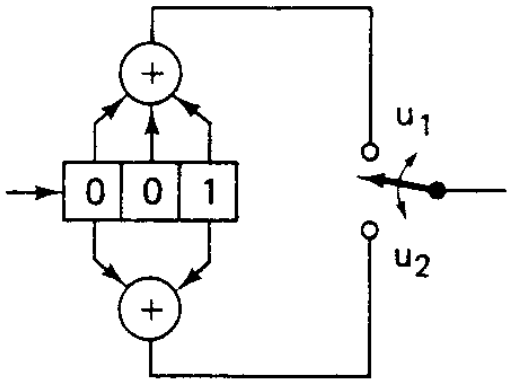
t_2 u_1 u_2 = 1 0



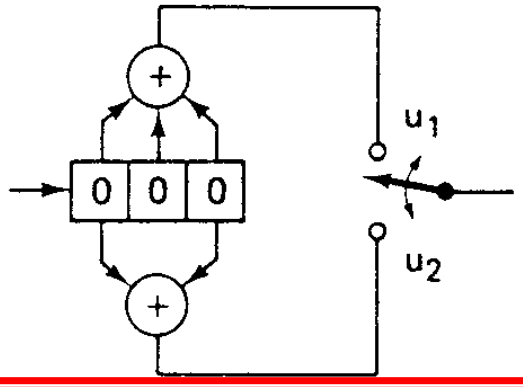
t_3 u_1 u_2 = 0 0



t_4 u_1 u_2 = 1 0



t_5 u_1 u_2 = 1 1



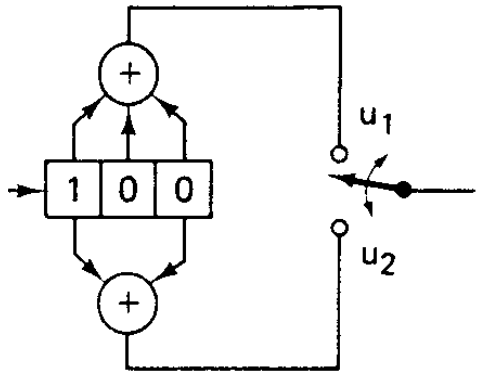
t_6 u_1 u_2 = 0 0

Output sequence: 11 10 00 10 11

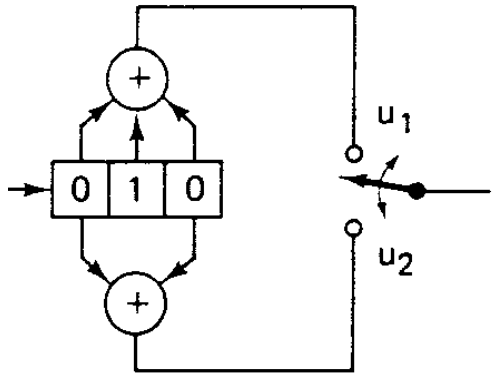
Impulse response and polynomial form

Impulse response

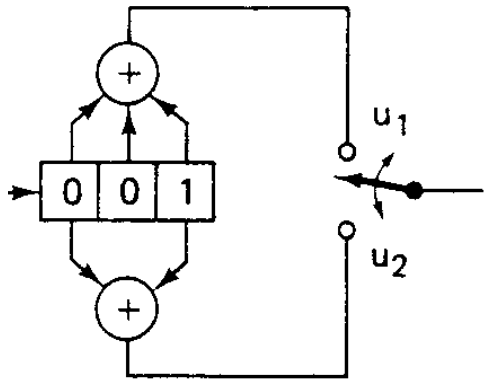
1 1 1 0 1 1



u_1	u_2
1	1



u_1	u_2
1	0



u_1	u_2
1	1

Impulse response

Register contents	Branch word	
	u_1	u_2
1 0 0	1	1
0 1 0	1	0
0 0 1	1	1

Input sequence: 1 0 0
Output sequence: 1 1 1 0 1 1

Linearity

Input **m**	Output				
1	1 1	1 0	1 1		
0		0 0	0 0	0 0	
1			1 1	1 0	1 1
Modulo-2 sum:	1 1	1 0	0 0	1 0	1 1

Output sequence: 11 10 00 10 11

Polynomial multiplication

***m*=101**

***m*(*X*)=1+*X*²**

$$\mathbf{g}_1(X) = 1 + X + X^2 \qquad g_1 = [1 \quad 1 \quad 1]$$

$$\mathbf{g}_2(X) = 1 + X^2 \qquad g_2 = [1 \quad 0 \quad 1]$$

Polynomial multiplication

$m=101$

$m(X)=1+X^2$

$$\mathbf{g}_1(X) = 1 + X + X^2 \qquad g_1 = [1 \quad 1 \quad 1]$$

$$\mathbf{g}_2(X) = 1 + X^2 \qquad g_2 = [1 \quad 0 \quad 1]$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = (1 + X^2)(1 + X + X^2) = 1 + X + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = (1 + X^2)(1 + X^2) = 1 + X^4$$

Polynomial multiplication

$m=101$

$m(X)=1+X^2$

$$\mathbf{g}_1(X) = 1 + X + X^2 \qquad g_1 = [1 \quad 1 \quad 1]$$

$$\mathbf{g}_2(X) = 1 + X^2 \qquad g_2 = [1 \quad 0 \quad 1]$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = (1 + X^2)(1 + X + X^2) = 1 + X + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = (1 + X^2)(1 + X^2) = 1 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = 1 + X + 0X^2 + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = 1 + 0X + 0X^2 + 0X^3 + X^4$$

Polynomial multiplication

$m=101$

$m(X)=1+X^2$

$$\mathbf{g}_1(X) = 1 + X + X^2 \qquad g_1 = [1 \quad 1 \quad 1]$$

$$\mathbf{g}_2(X) = 1 + X^2 \qquad g_2 = [1 \quad 0 \quad 1]$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = (1 + X^2)(1 + X + X^2) = 1 + X + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = (1 + X^2)(1 + X^2) = 1 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = 1 + X + 0X^2 + X^3 + X^4$$

m

Output sequence: 11 10 00 10 11

, $1)X^4$

$$\mathbf{U} = 1\ 1 \qquad 1\ 0 \qquad 0\ 0 \qquad 1\ 0 \qquad 1\ 1$$

Convolutional encoders

- Block diagram of shift registers
- Connection vectors and polynomials interpretation
- Impulse response
- State diagram
- Tree representation
- Trellis representation

Finite state machine

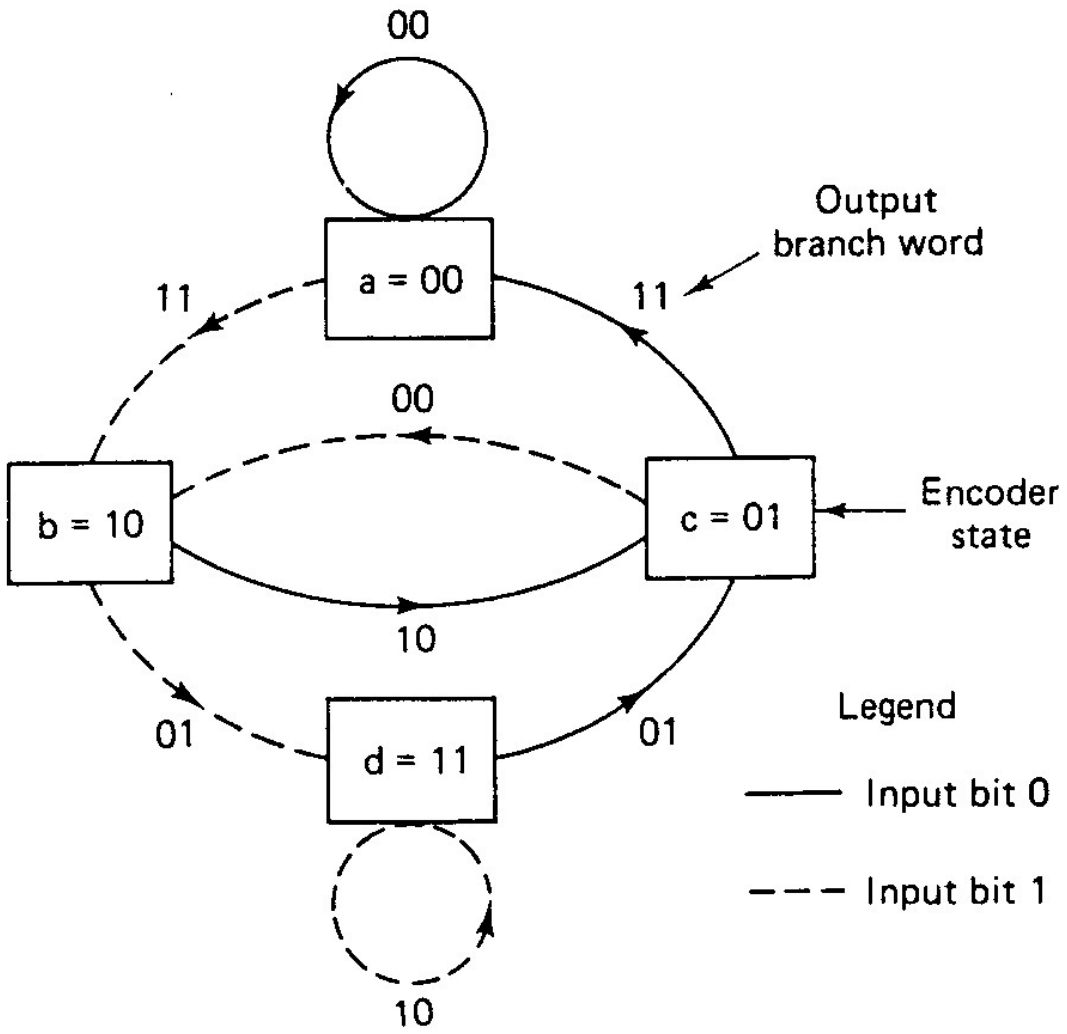
Finite state machine

- The output at time t is a function of the contents of the registers
  - "The state of the registers"
- Encoding is the transition from one state to another state
- The number of states is finite
- Finite state machine

State diagram

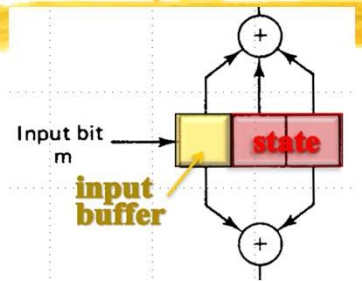
$$g_1 = [1 \quad 1 \quad 1]$$

$$g_2 = [1 \quad 0 \quad 1]$$



State diagram

- Fix the 2^{K-1} states



a = 00

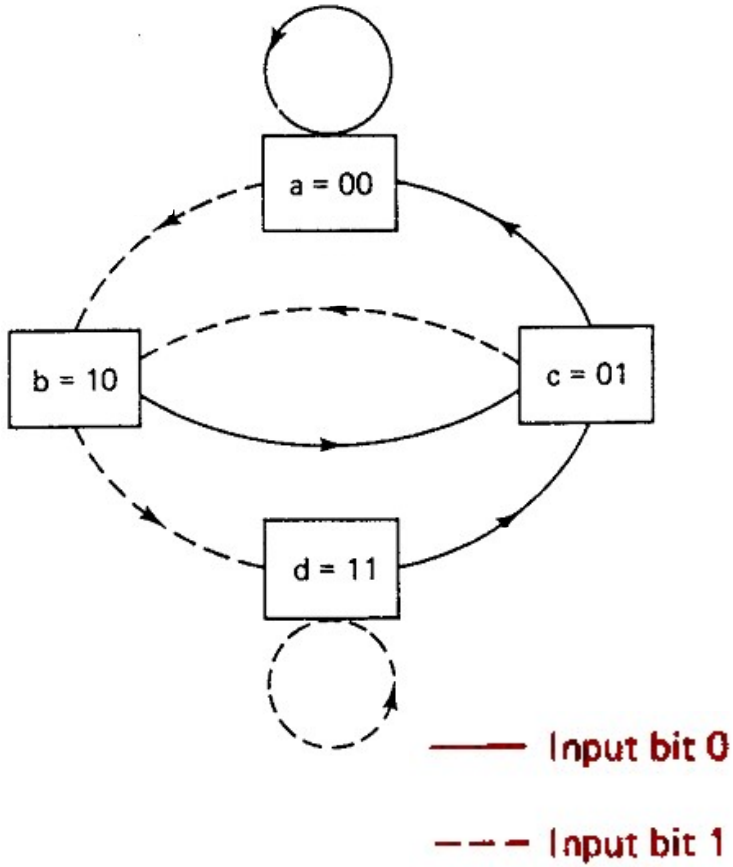
b = 10

c = 01

d = 11

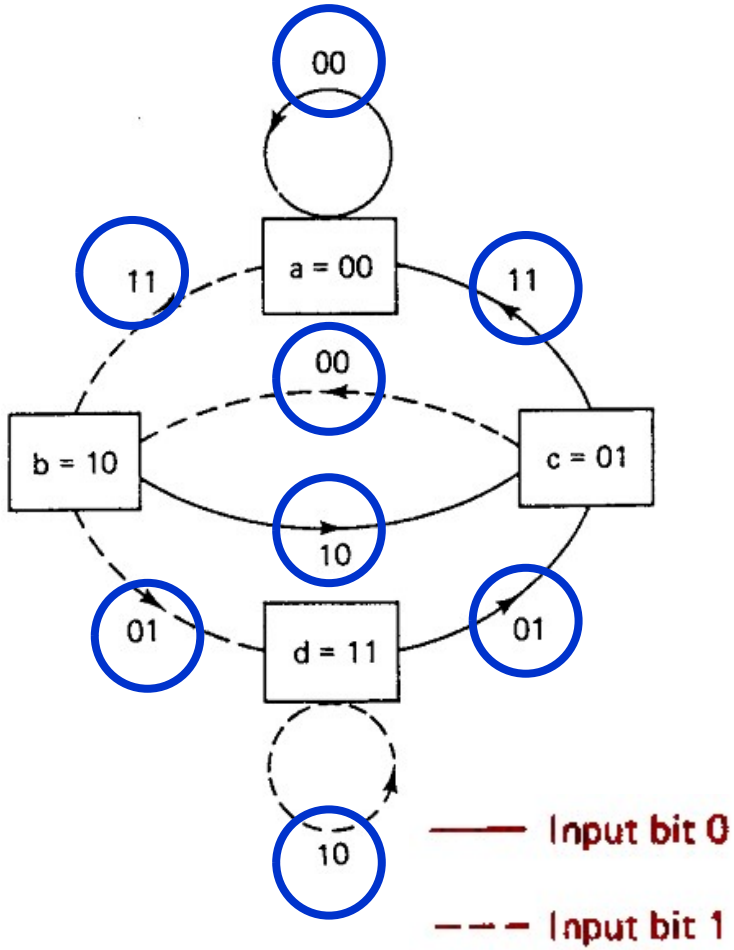
State diagram

- Fix the 2^{K-1} states
- Establish valid transitions



State diagram

- Fix the 2^{K-1} states
- Establish valid transitions
- Generate codes for each transition

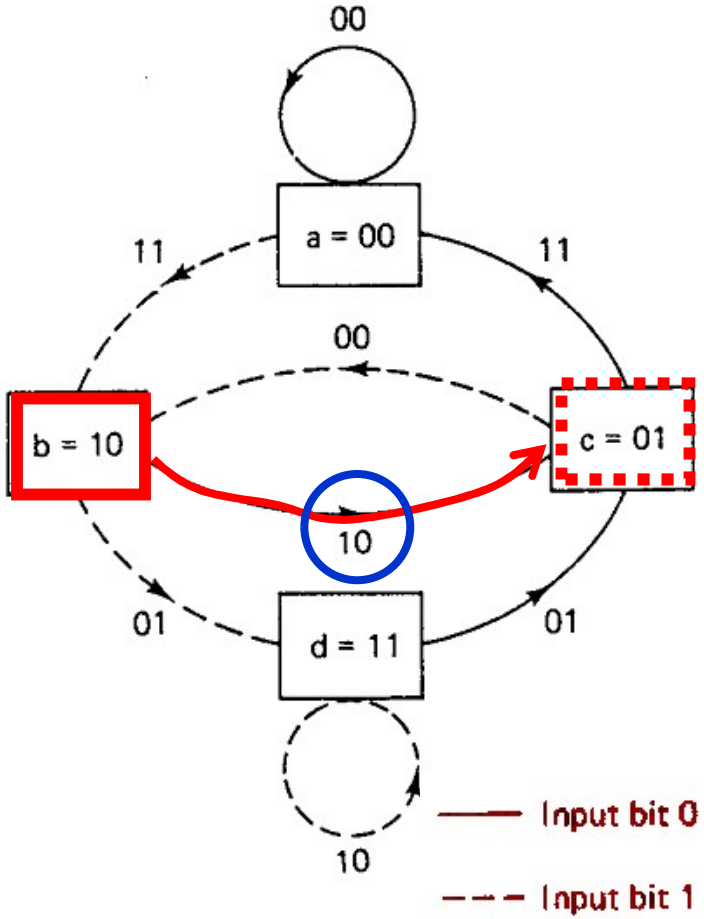


$$g_1 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$g_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

Exemple

- Let's start at the state b
  - $\underline{r} = [*\quad 1\quad 0]$
- Next bit 0
  - Next state $\underline{r} = [*\quad 0\quad 1]$
- Output
  - $\underline{r} = [0\quad 1\quad 0]$
  - $u_1 = r_1 + r_2 + r_3 = 0 + 1 + 0 = 1$
  - $u_2 = r_1 + r_3 = 0 + 0 = 0$



For the encoder shown in Figure 6.3, show the state changes and the resulting output codeword sequence **U** for the message sequence **m** = 1 1 0 1 1, followed by $K - 1 = 2$ zeros to flush the register. Assume that the initial contents of the register are all zeros.

Solution

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i	
				u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: **U** = 1 1 0 1 0 1 0 0 0 1 0 1 1 1

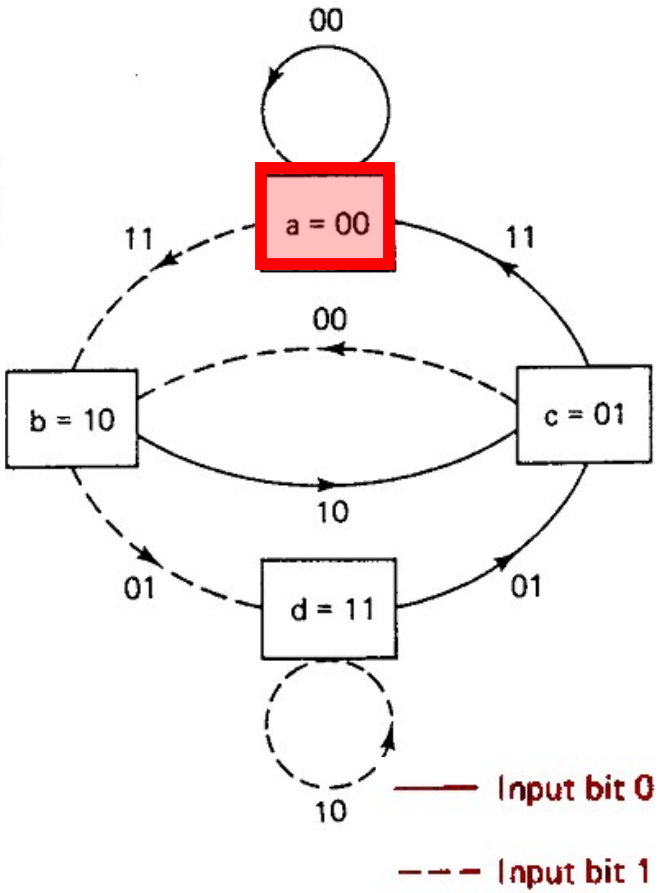
State a

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



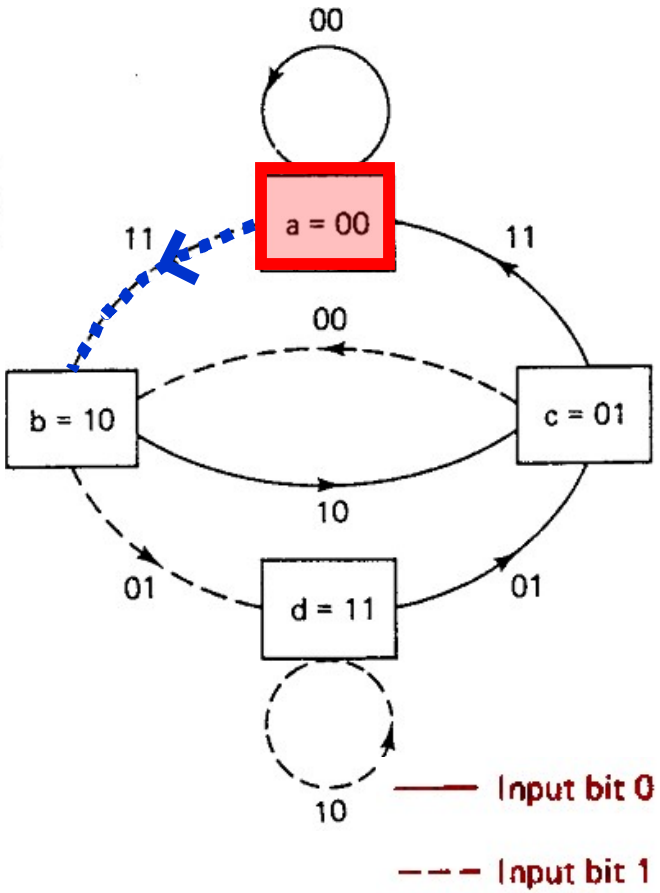
Input: 1

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



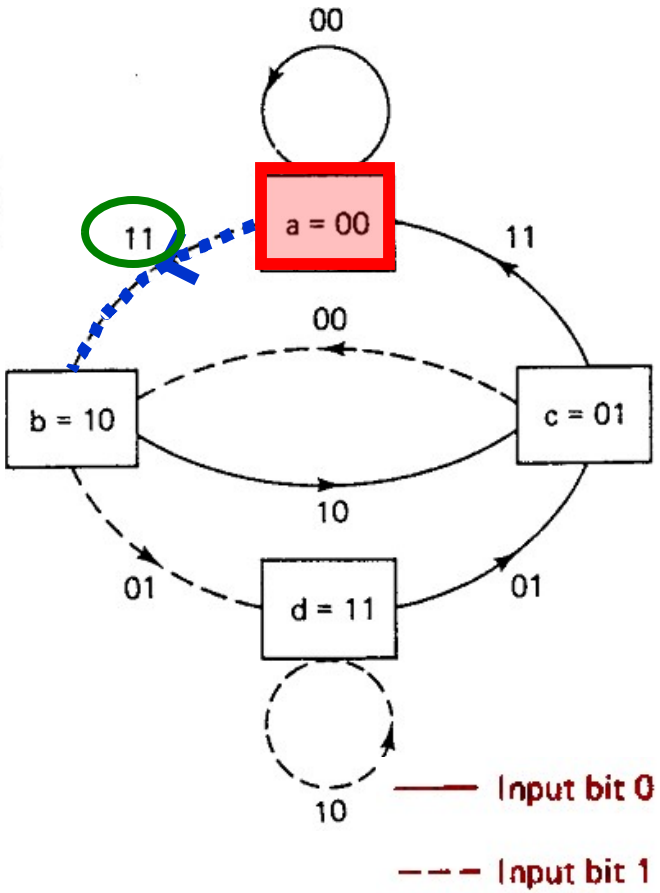
Output: 11

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



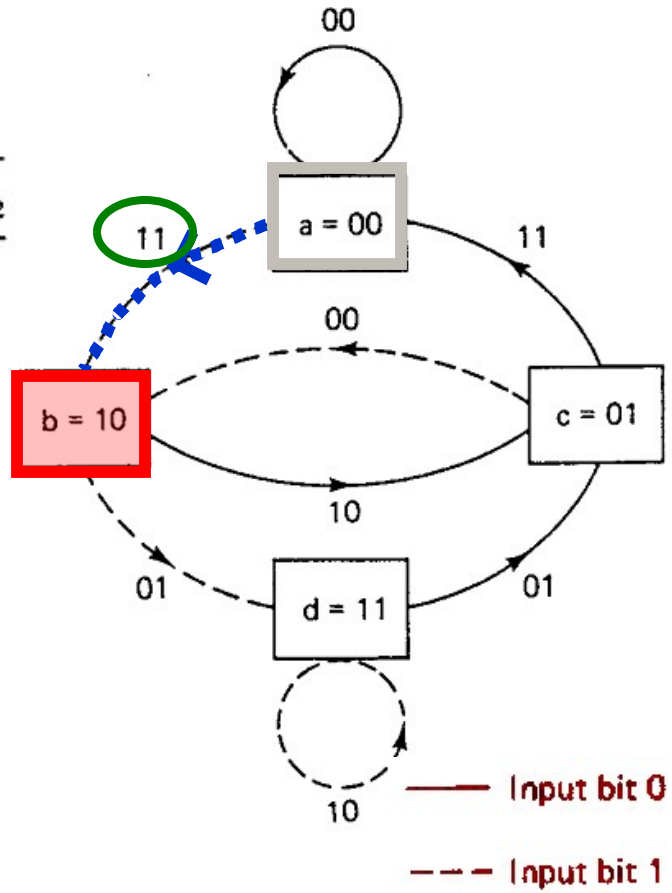
State: b = 10

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



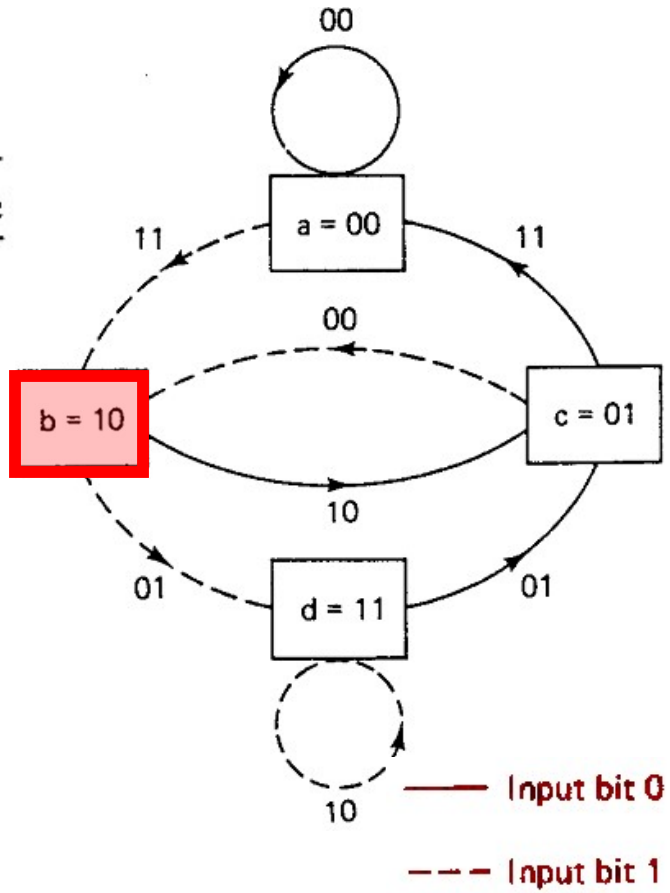
State: b = 10

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



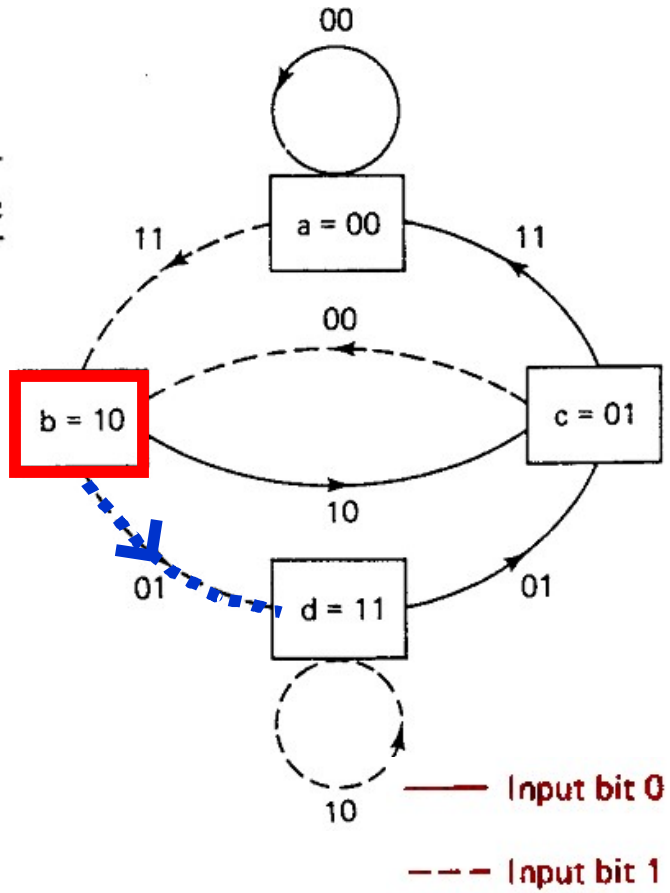
Input: 1

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



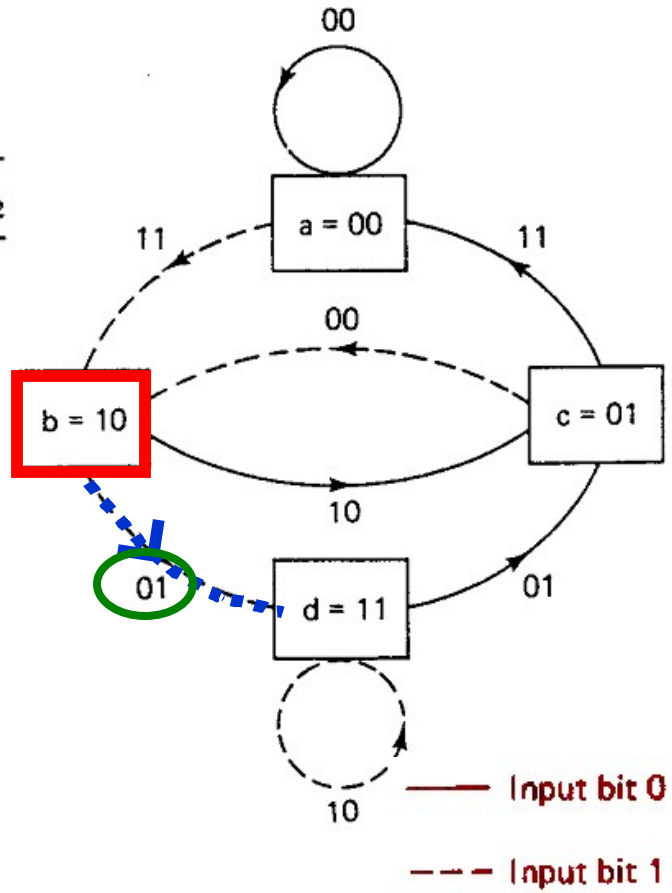
Output: 01

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



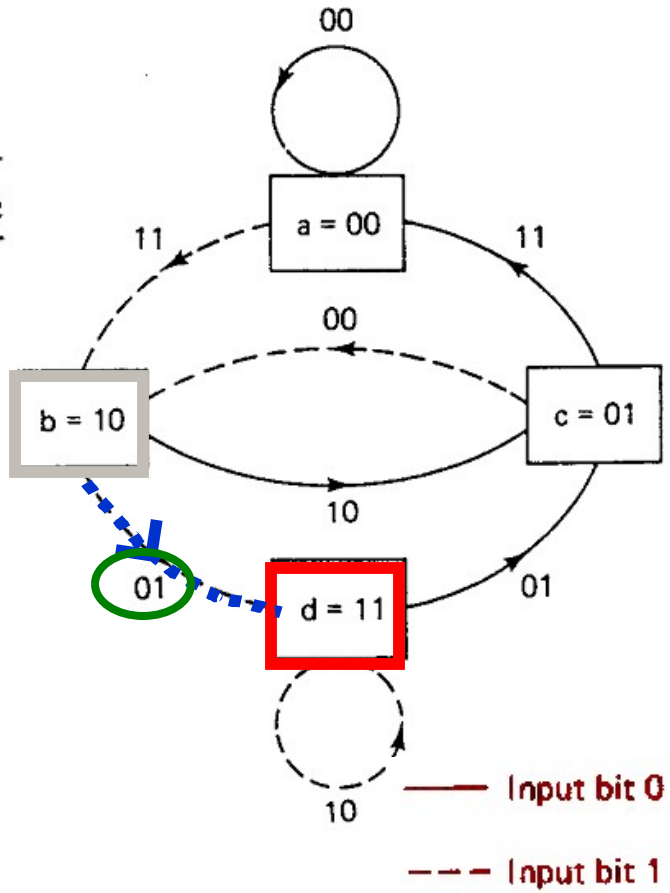
State: d = 11

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



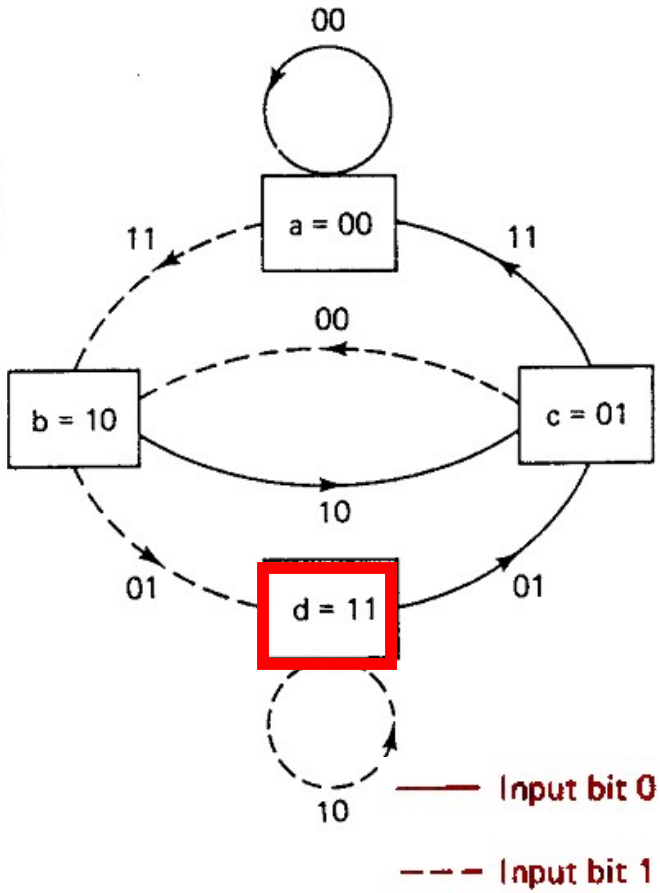
State: d = 11

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



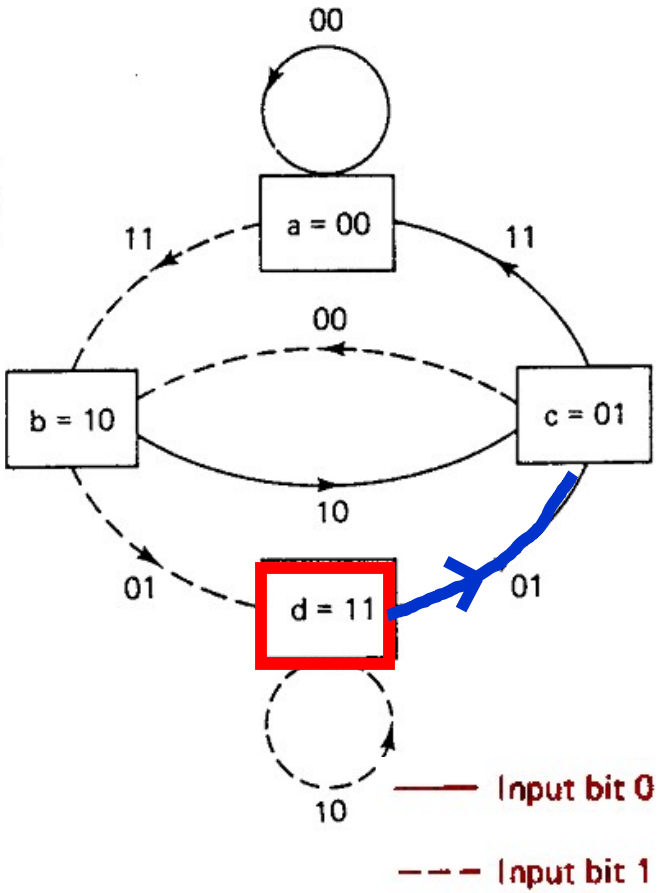
Input: 0

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



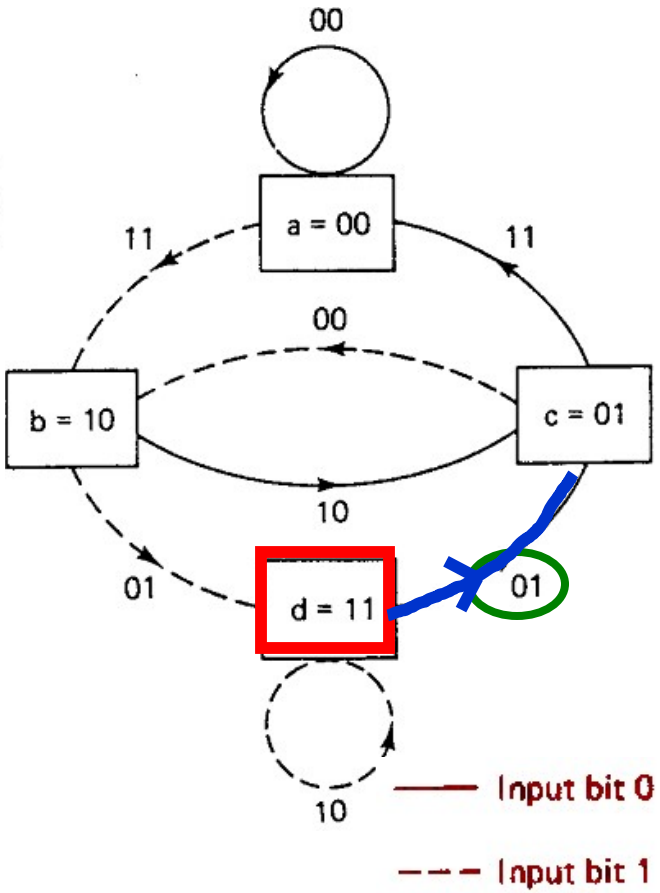
Output: 01

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



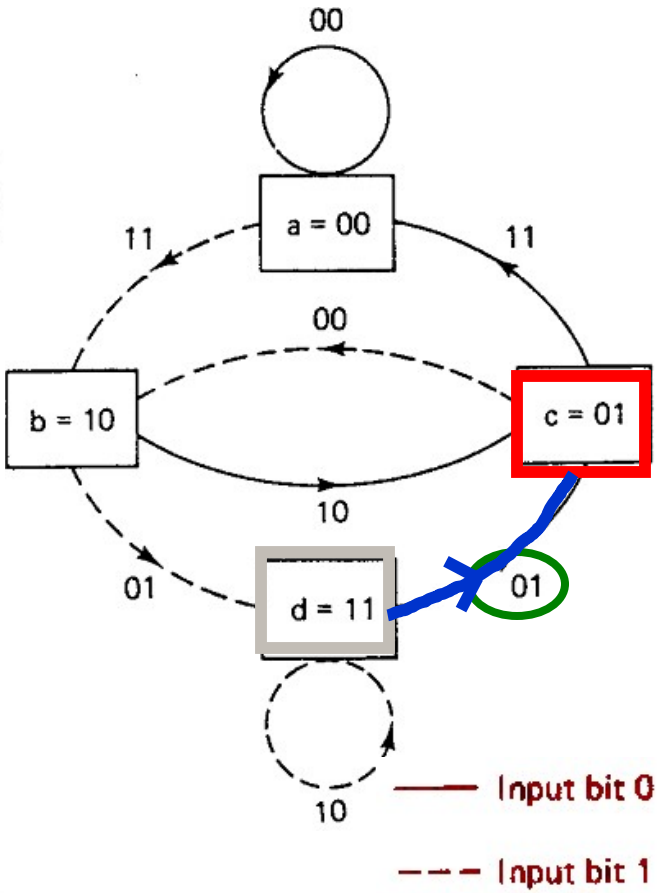
State: c = 01

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



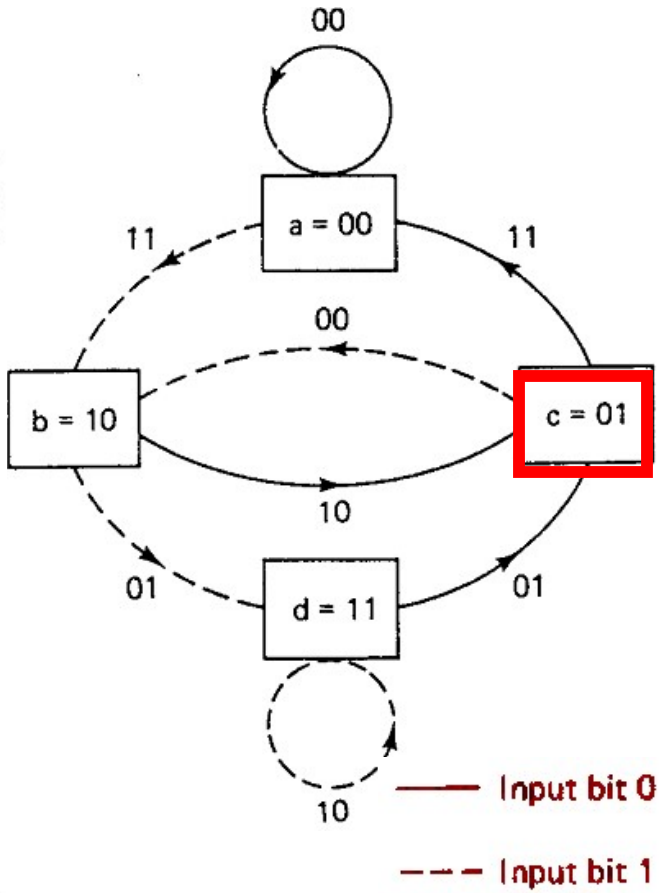
State: c = 01

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



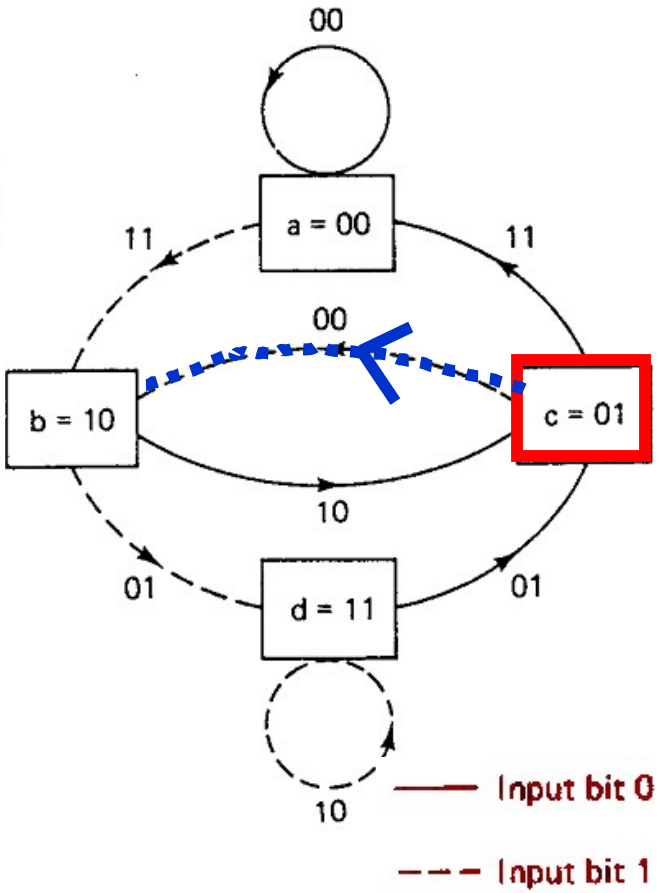
Input: 1

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



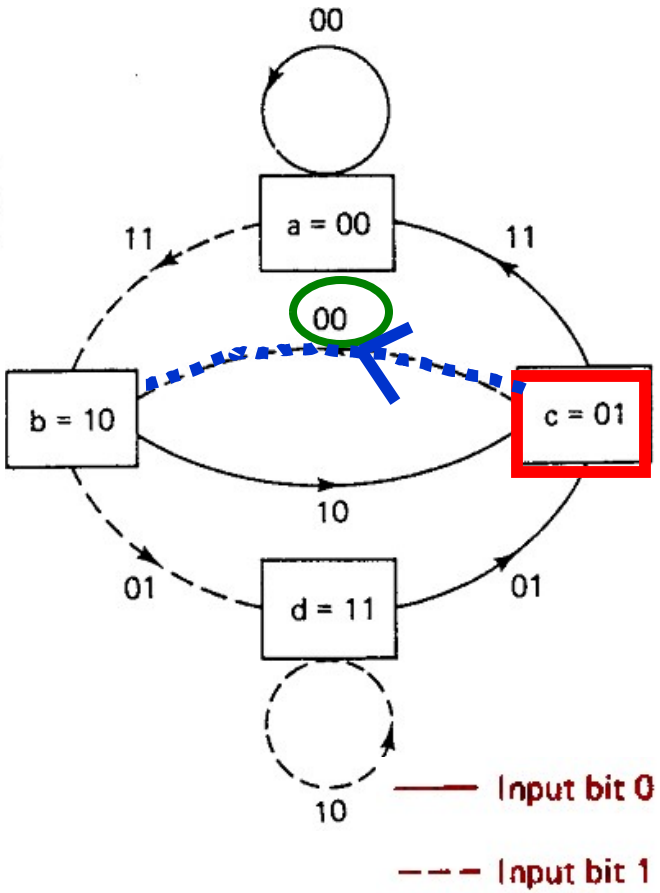
Output: 00

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



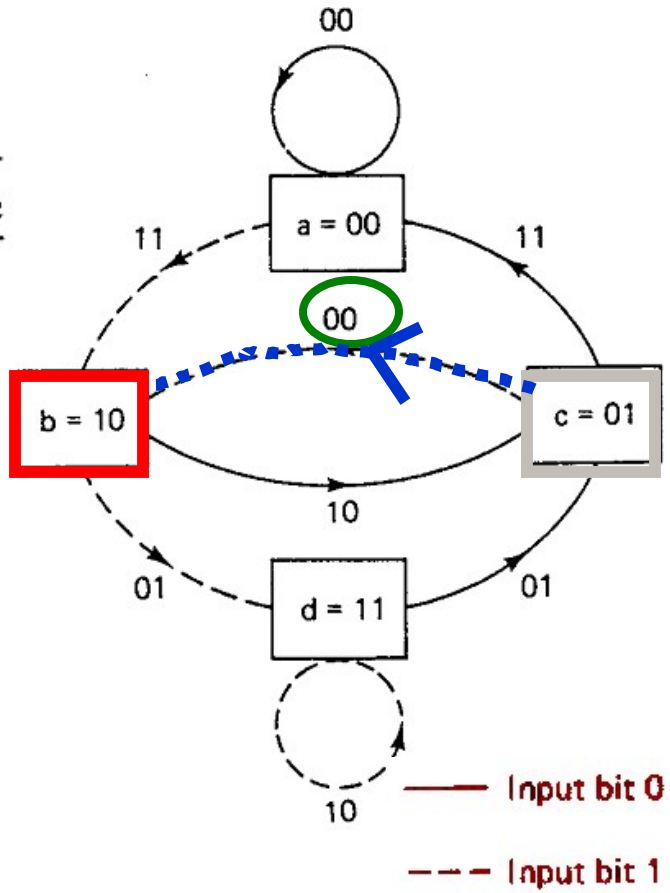
State: b = 10

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



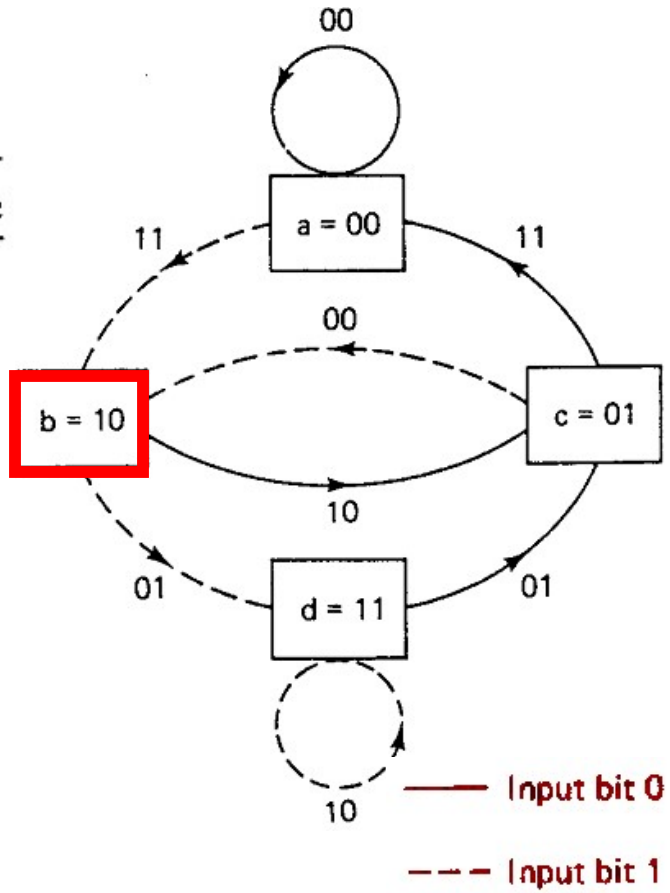
State: b = 10

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



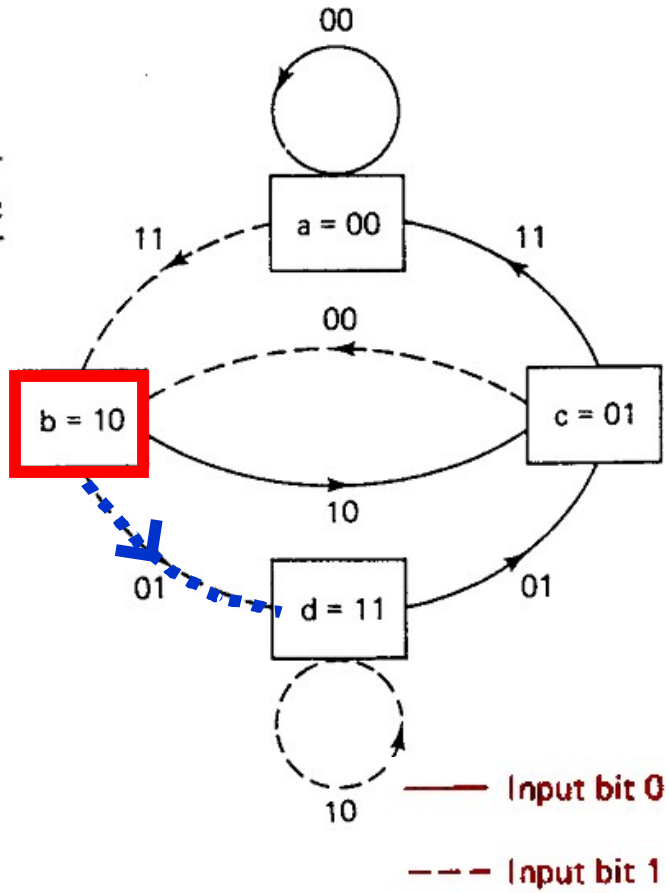
Input: 1

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1



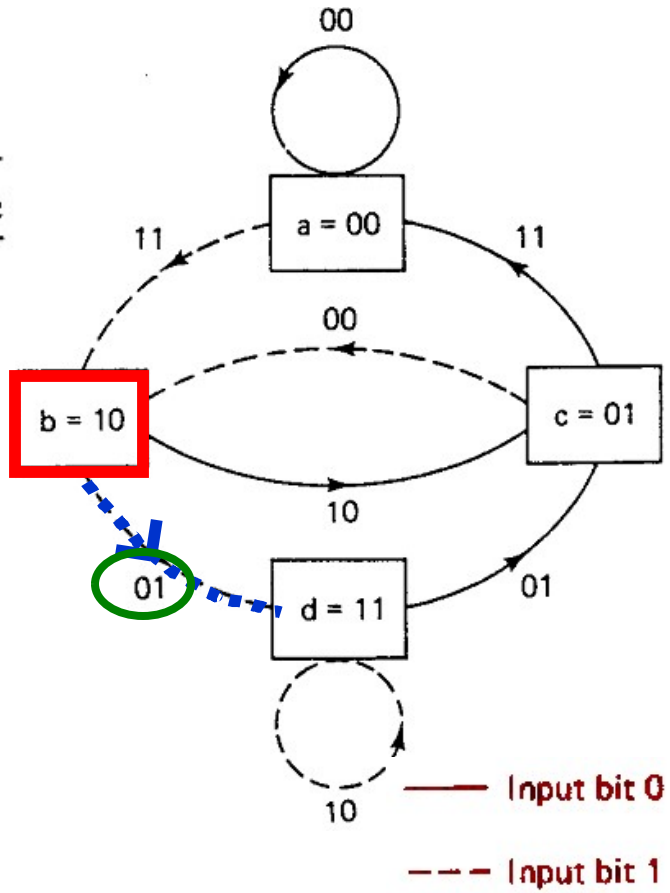
Output: 01

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1

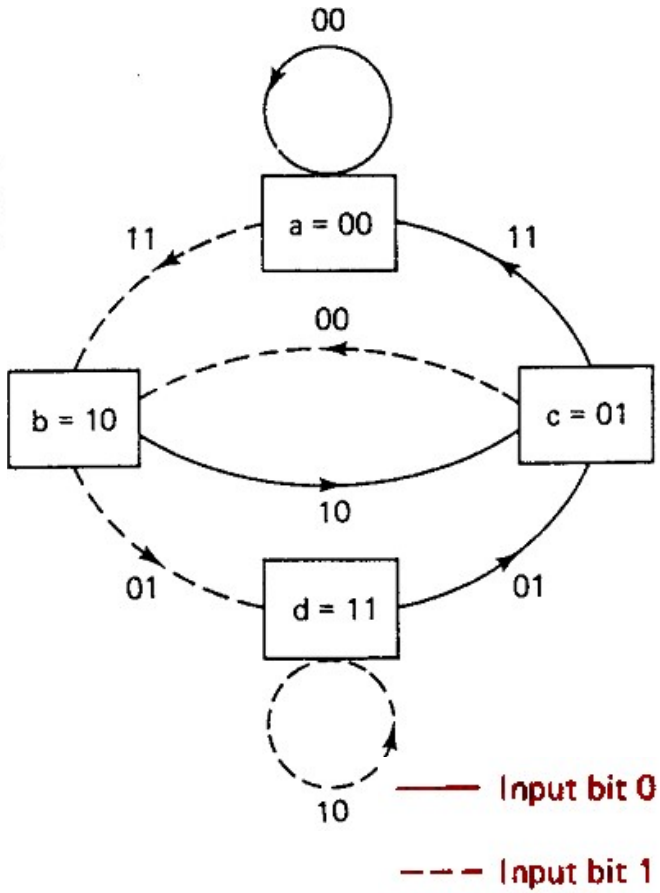


Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

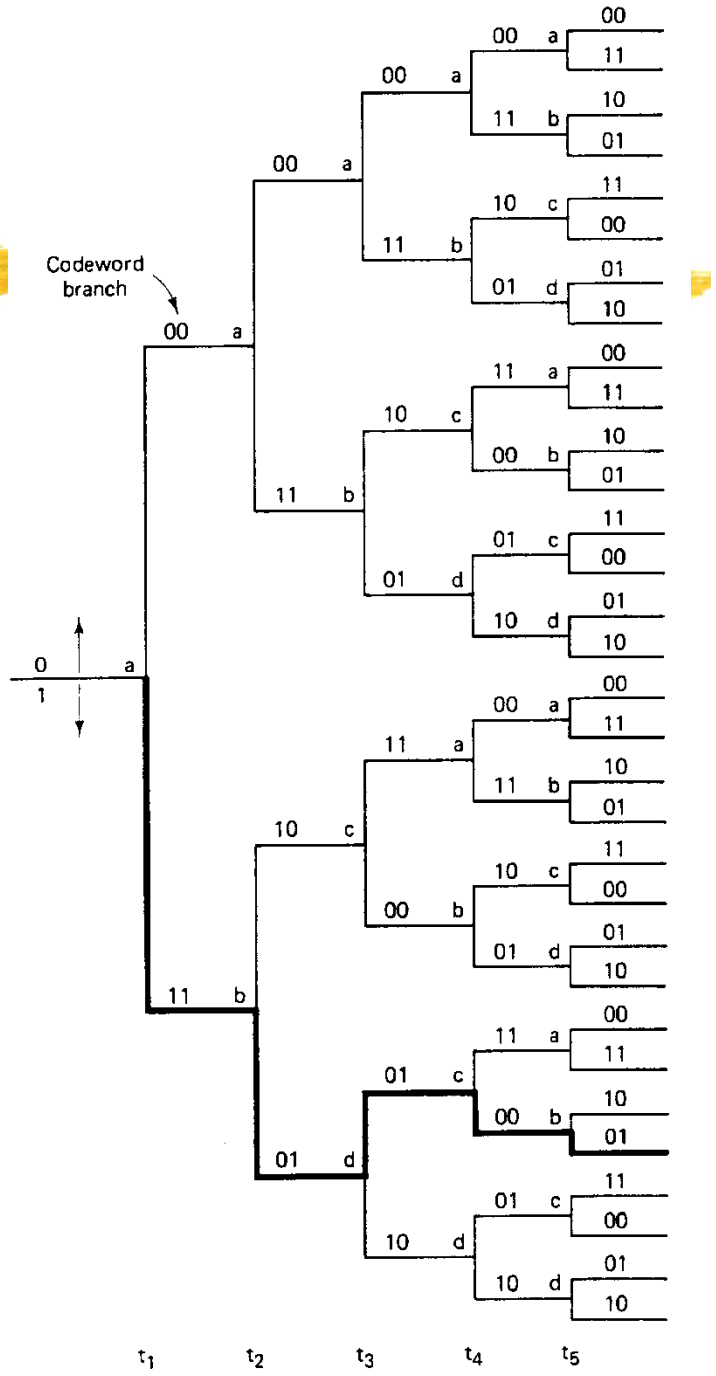
Output sequence: U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1

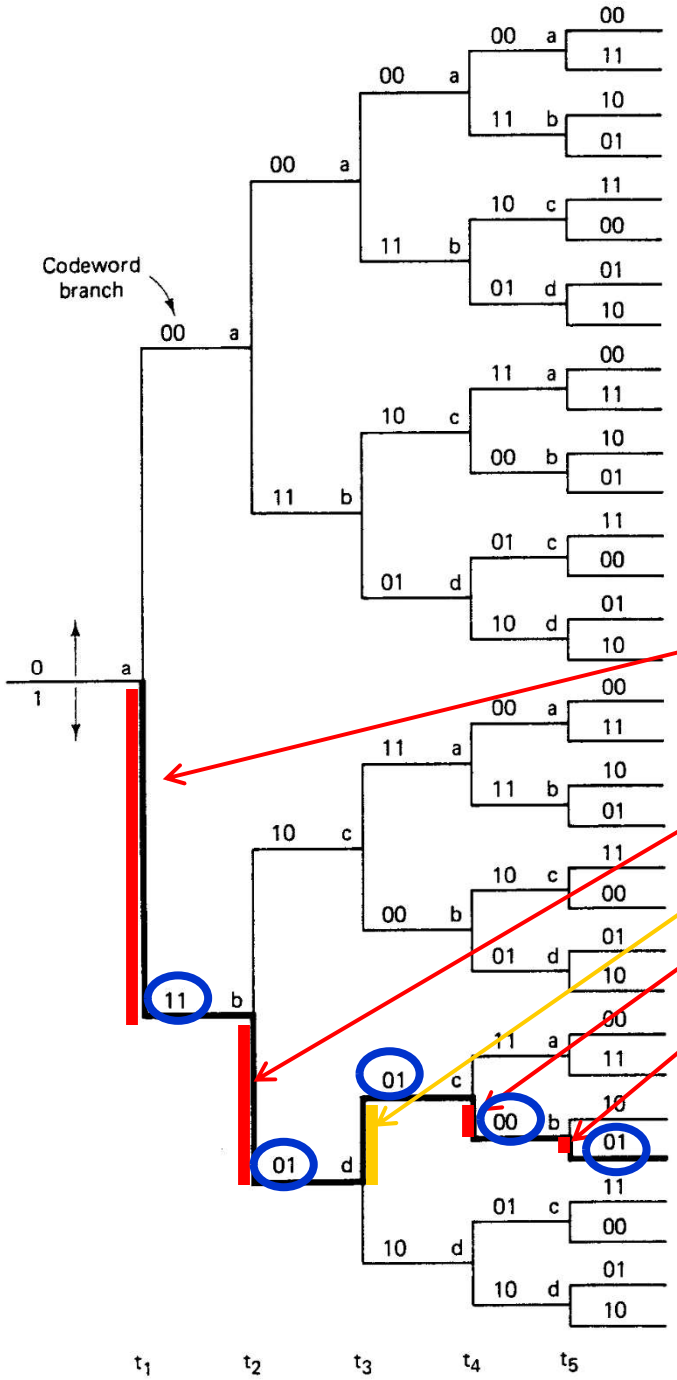


Tree and trellis

Tree diagram

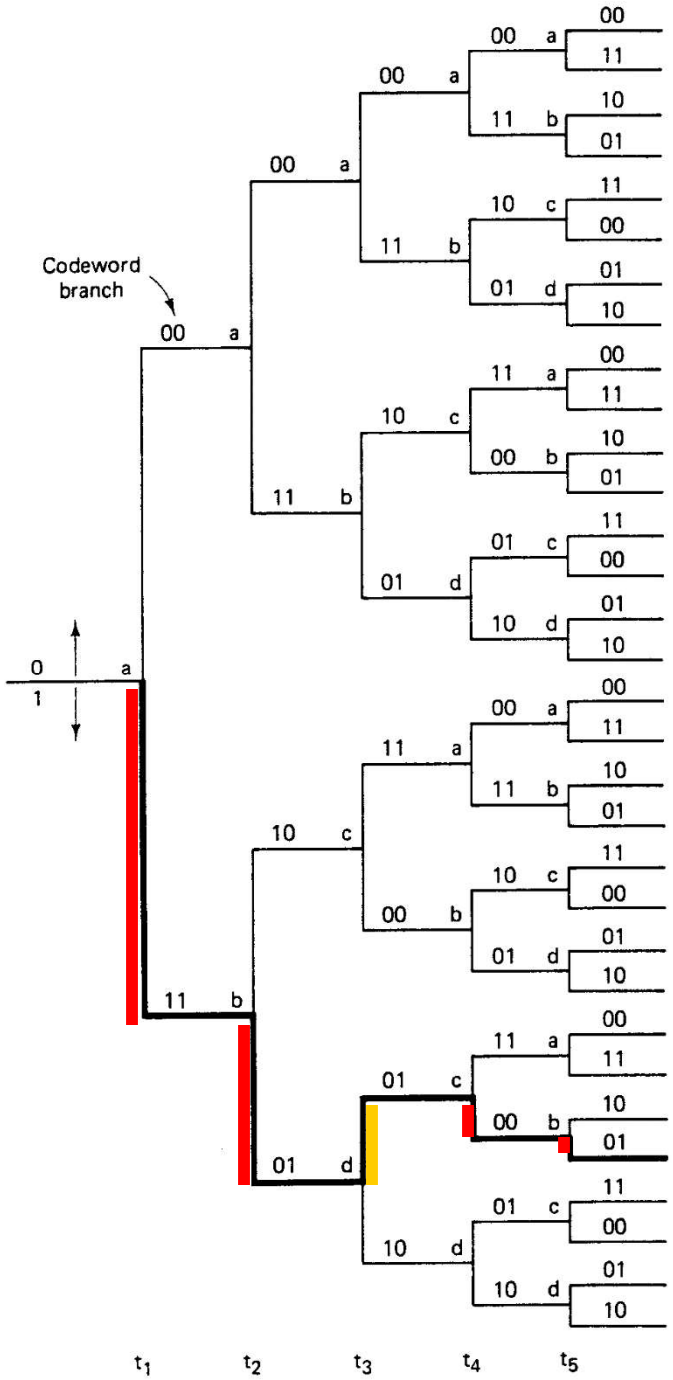
- Same principle as state diagram
- We can keep the behavior in time
- Input bit zero - go up
- Input bit one - go down

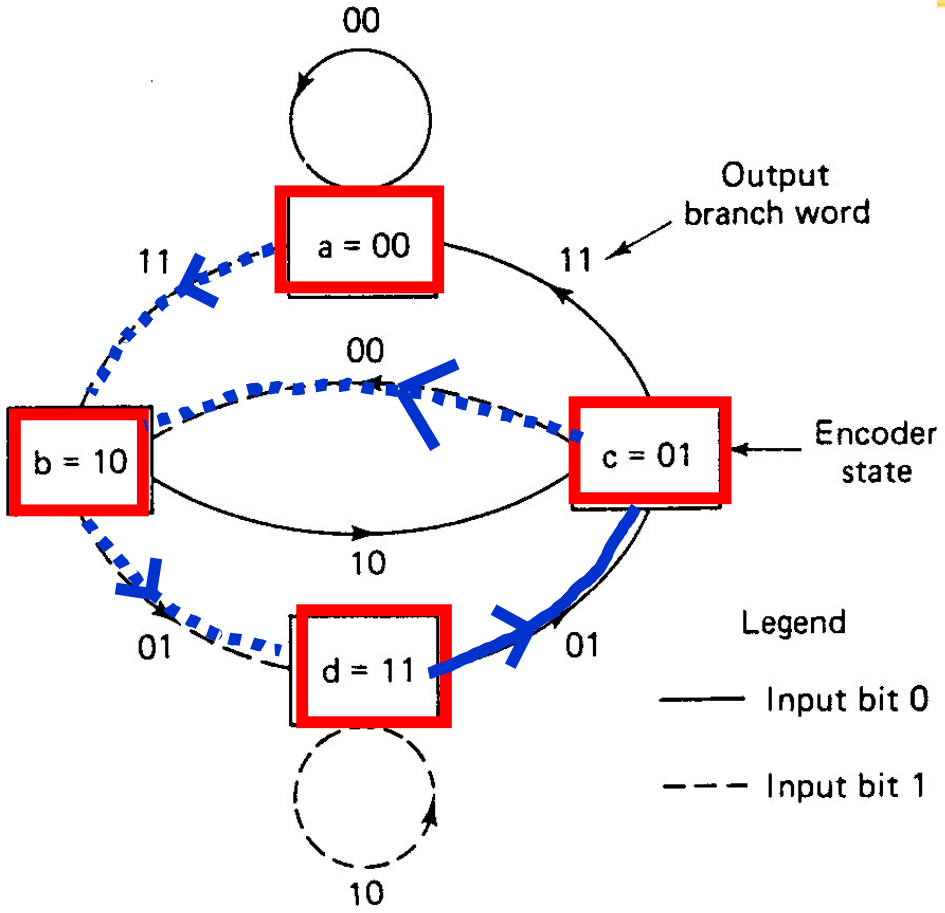




Input bit m_i
–
1
1
0
1
1
0
0

U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1

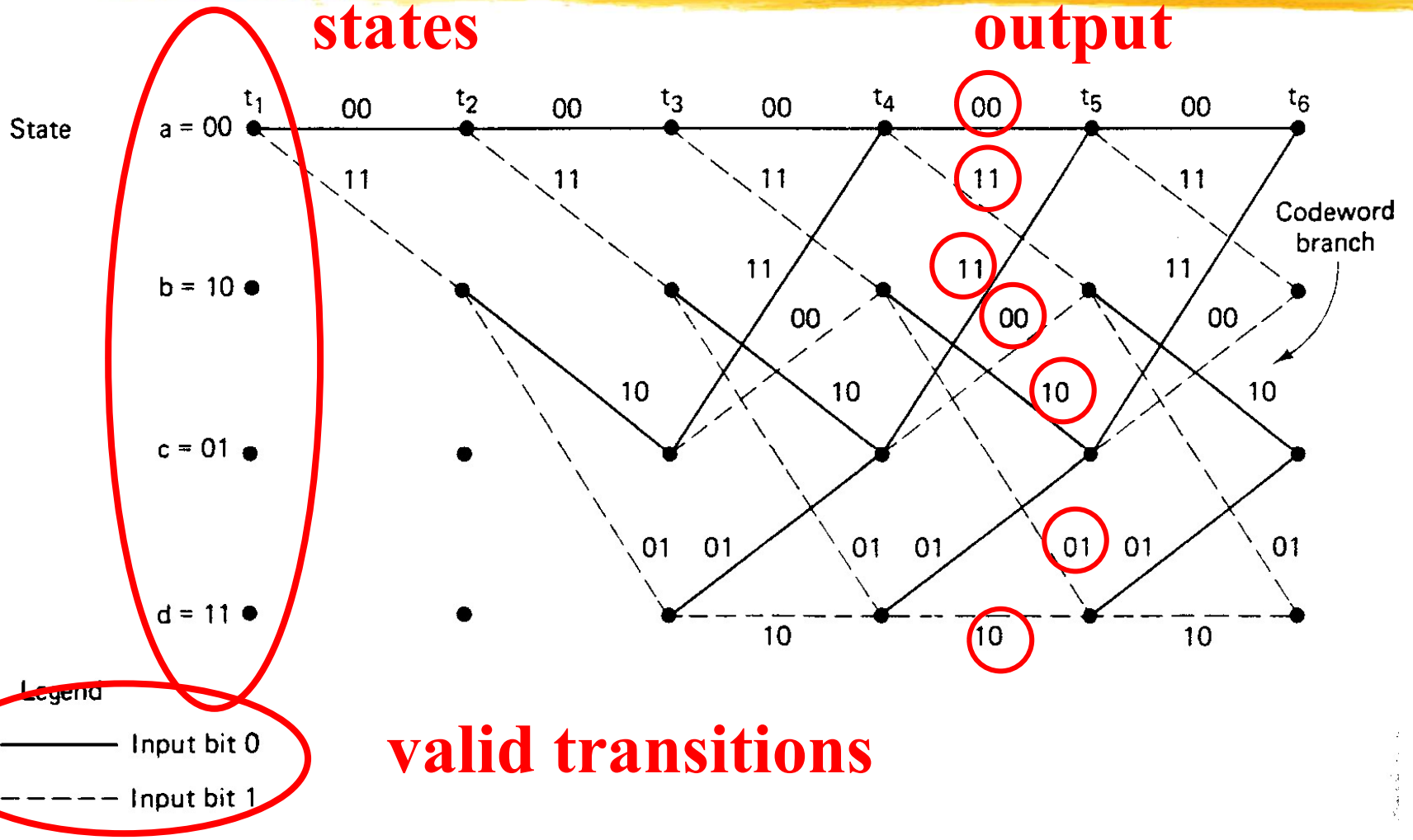

Codeword branch
0
1
a
00
11
10
01
b
c
d
t1
t2
t3
t4
t5


a = 00
b = 10
c = 01
d = 11
Output branch word
Encoder state
Legend
Input bit 0
Input bit 1
00
11
10
01

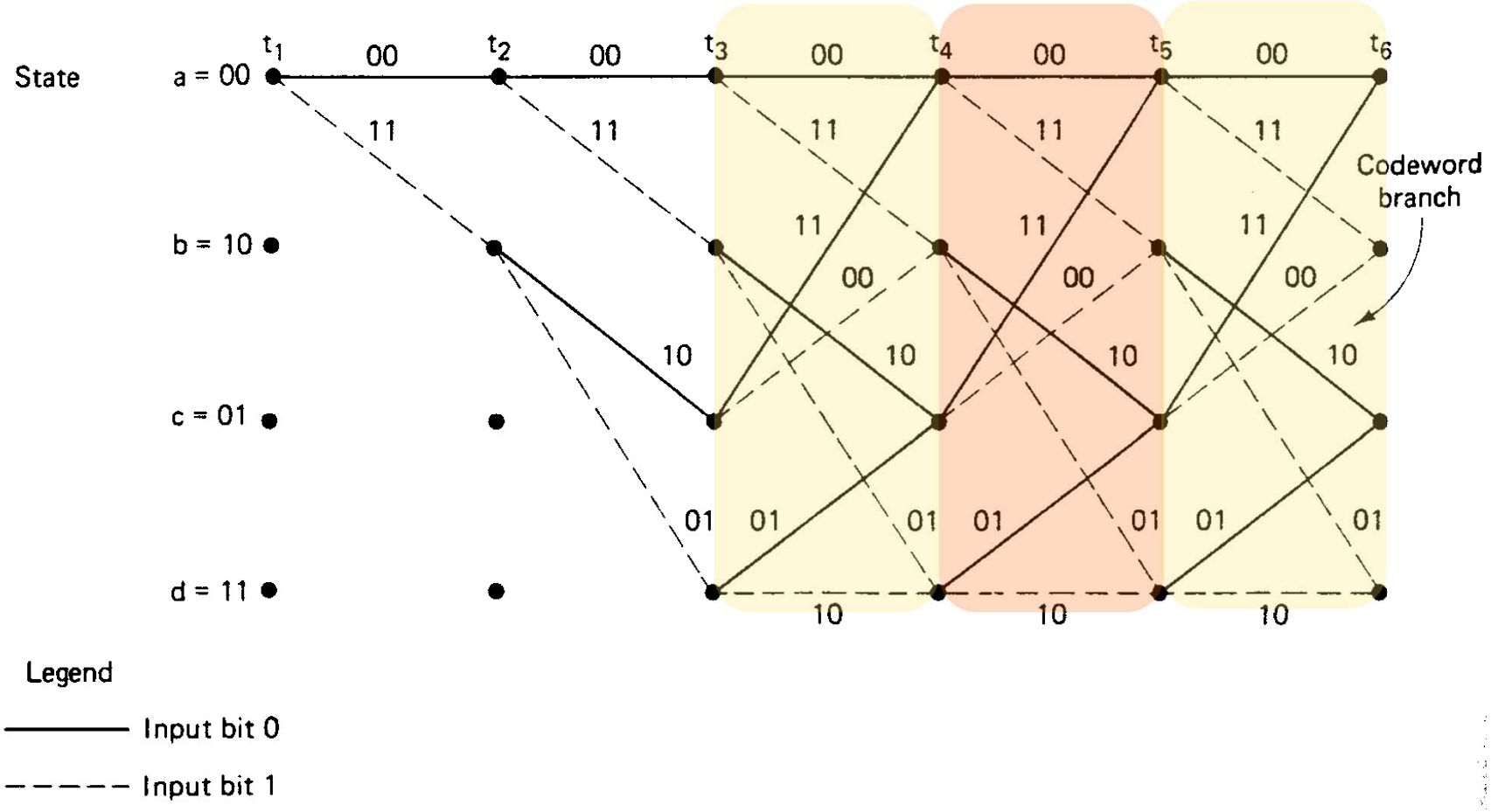
Trellis vs. tree

- Similar display that retains information about temporal behavior
- Trellis more compact than the tree
- Very important for decoding

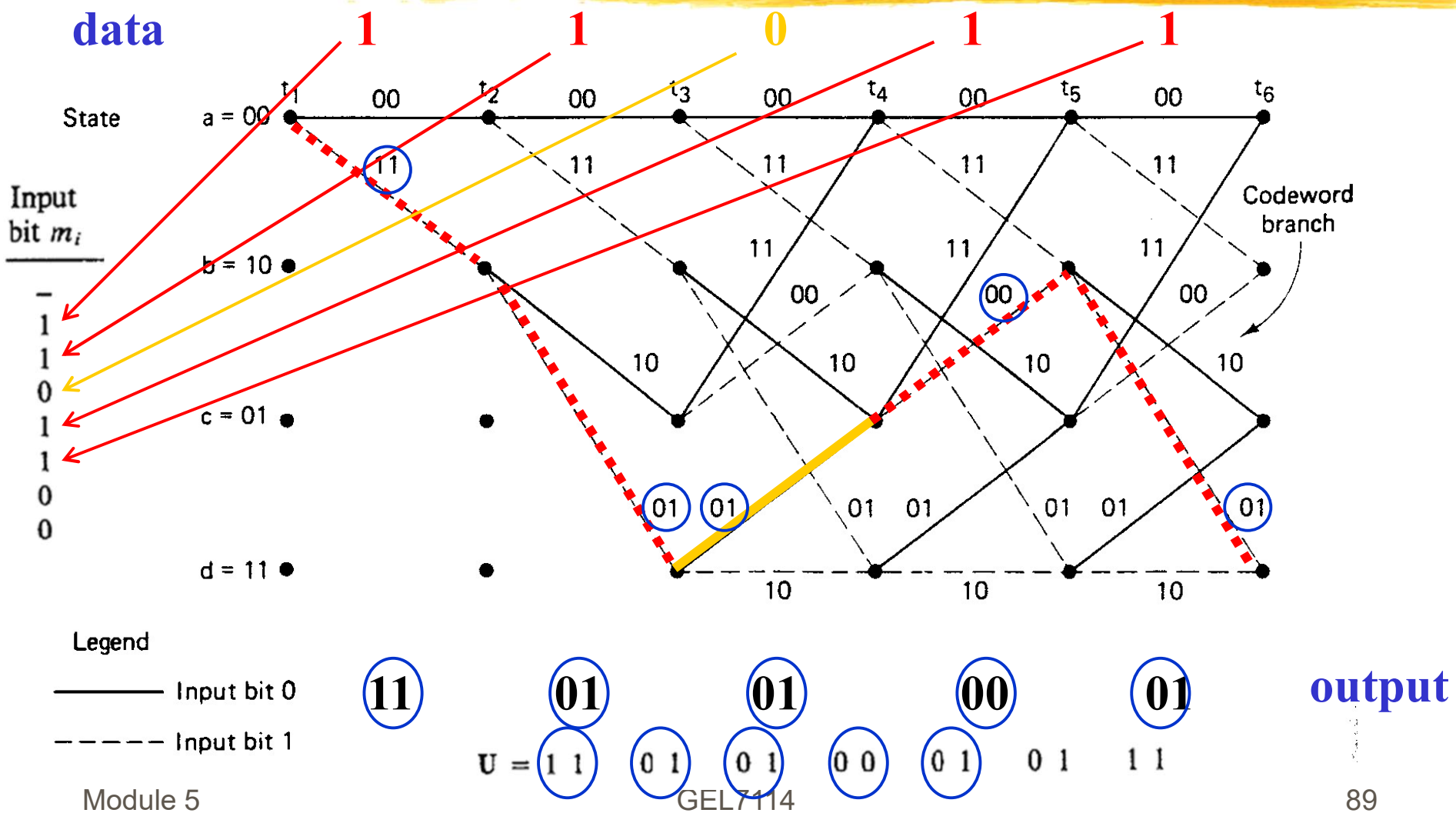
Treillis


states
output
State
a = 00
b = 10
c = 01
d = 11
t1
t2
t3
t4
t5
t6
00
11
10
01
Codeword branch
Legend
Input bit 0
Input bit 1
valid transitions

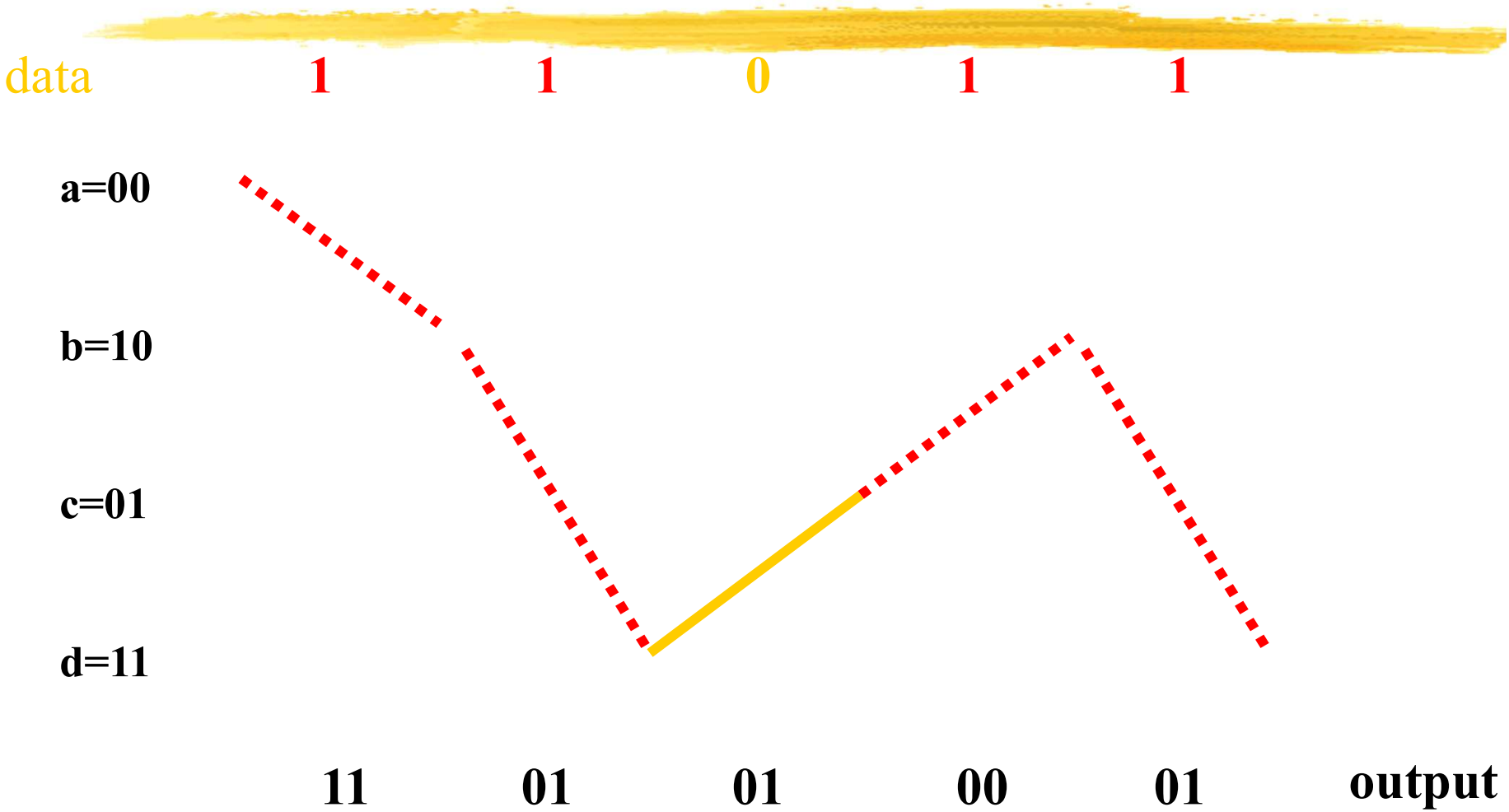
Treillis


State
a = 00
b = 10
c = 01
d = 11
t1
t2
t3
t4
t5
t6
00
11
10
01
Codeword branch
Legend
Input bit 0
Input bit 1

Treillis


data
1
1
0
1
1
State
a = 00
b = 10
c = 01
d = 11
Input bit m_i
–
1
1
0
1
1
0
0
Codeword branch
Legend
Input bit 0
Input bit 1
11
01
01
00
01
output
U = 1 1 0 1 0 1 0 0 0 1 0 1 1 1

Treillis

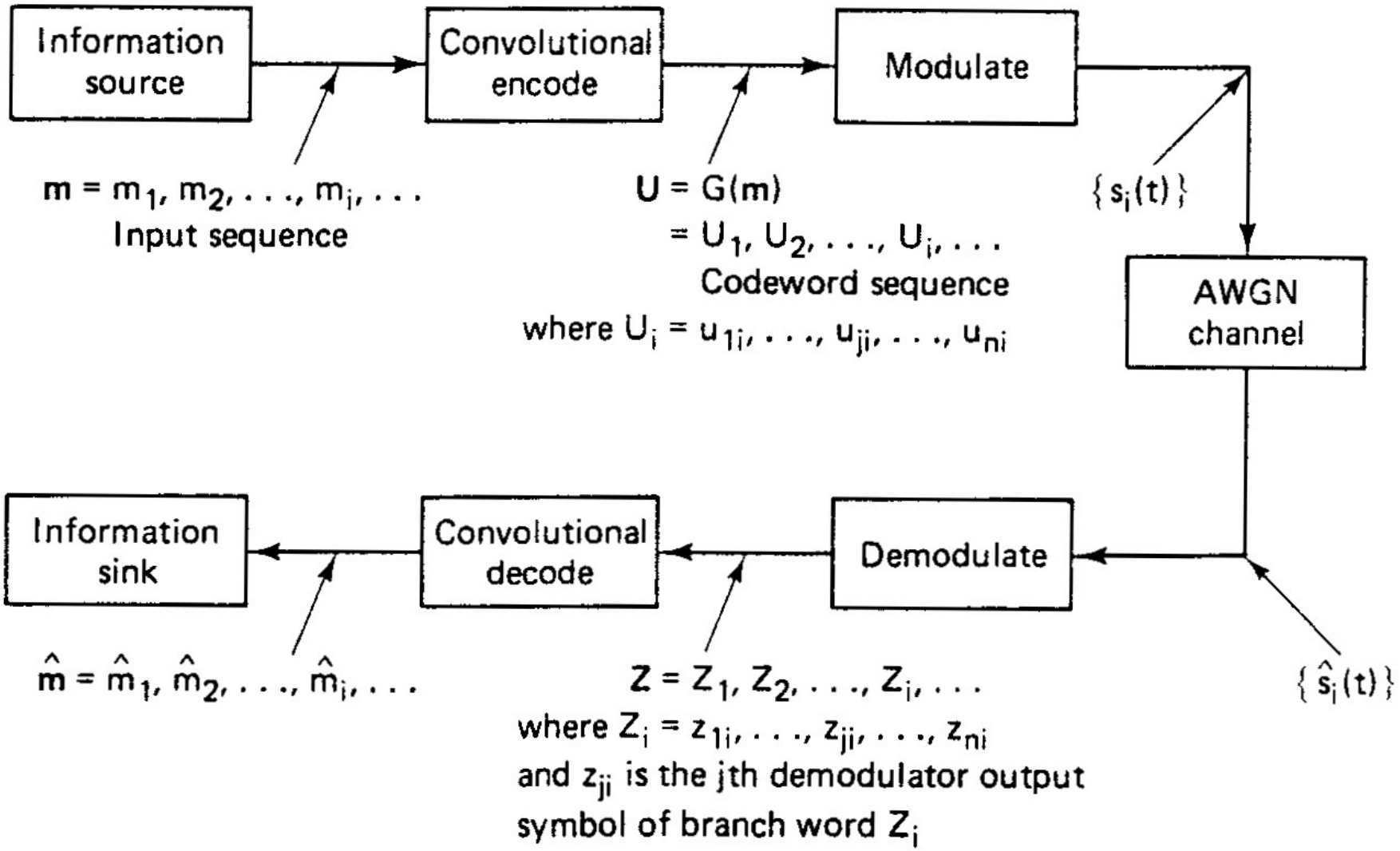


GEL7114

Decoding …

Topics covered

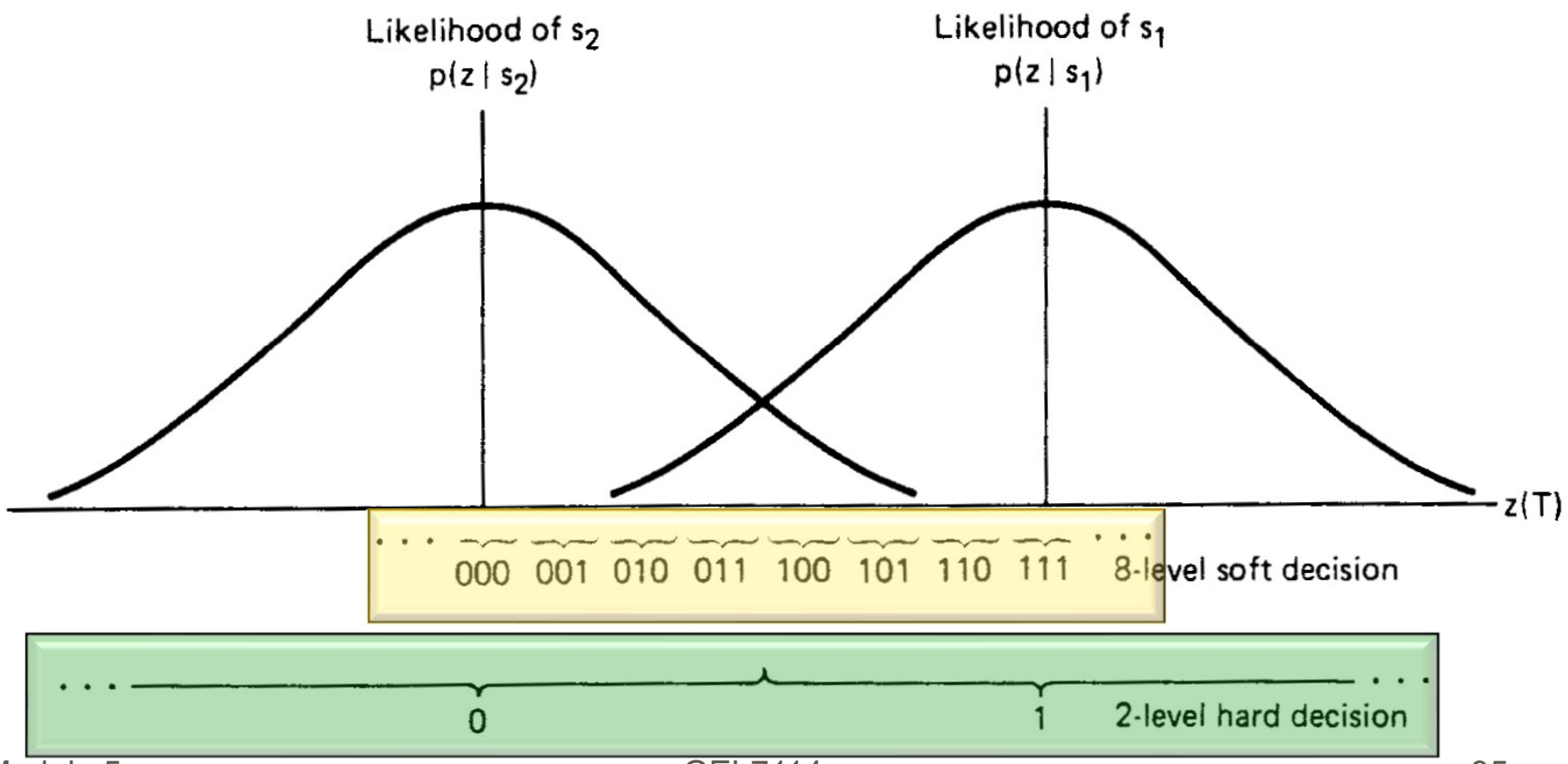
- Convolutional encoding (Sec. 7.1)
- Representation of encoders (Sec. 7.2)
- The decoding algorithm (Sec. 7.3)
- Properties of convolutional codes (Sec. 7.4)


Information source
Convolutional encode
Modulate
AWGN channel
Demodulate
Convolutional decode
Information sink
m = m1, m2, . . ., mi, . . .
Input sequence
U = G(m)
= U1, U2, . . ., Ui, . . .
Codeword sequence
where Ui = u1i, . . ., uji, . . ., uni
{si(t)}
{ŝi(t)}
Z = Z1, Z2, . . ., Zi, . . .
where Zi = z1i, . . ., zji, . . ., zni
and zji is the jth demodulator output symbol of branch word Zi
m̂ = m̂1, m̂2, . . ., m̂i, . . .

Decisions

- Hard decisions
  - decisions coming from a finite set
  - e.g. binary 1 or -1
  - no information about confidence in the decision ...
- Soft decisions
  - Continuous decisions
  - For binary modulation .3 or -.9 instead of 1, -1
  - Keep information about confidence in the decision ...

Soft v. Hard Decisions



Two metrics $dist(z_i, u_i)$

- Hamming distance
  - For hard decisions
  - $dist\left(\underline{z}_i, \underline{u}_i\right) = \#$ of differents bits
- Euclidian distance
  - For soft decisions
  - $dist\left(\underline{z}_i, \underline{u}_i\right) = \underline{z}_i^T \cdot \underline{u}_i = \sqrt{\left(z_{i,1} - u_{i,1}\right)^2 + \cdots + \left(z_{i,n} - u_{i,n}\right)^2}$

Binary Symmetric Channel

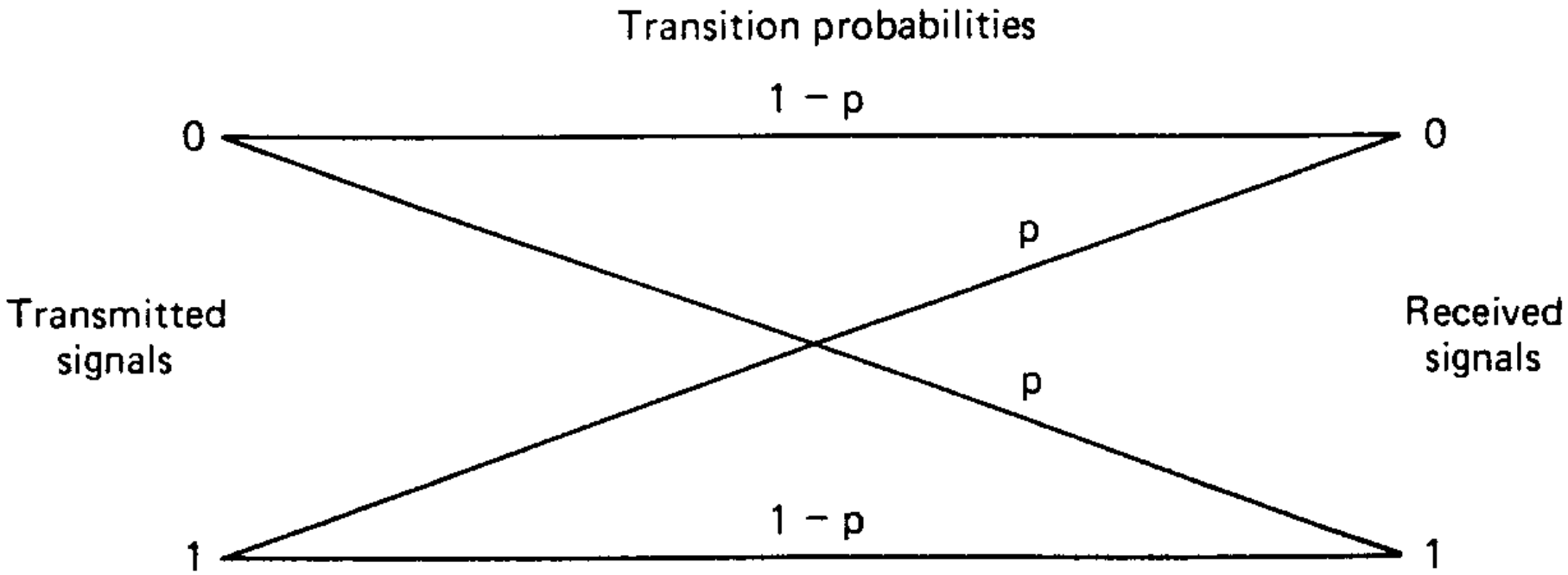


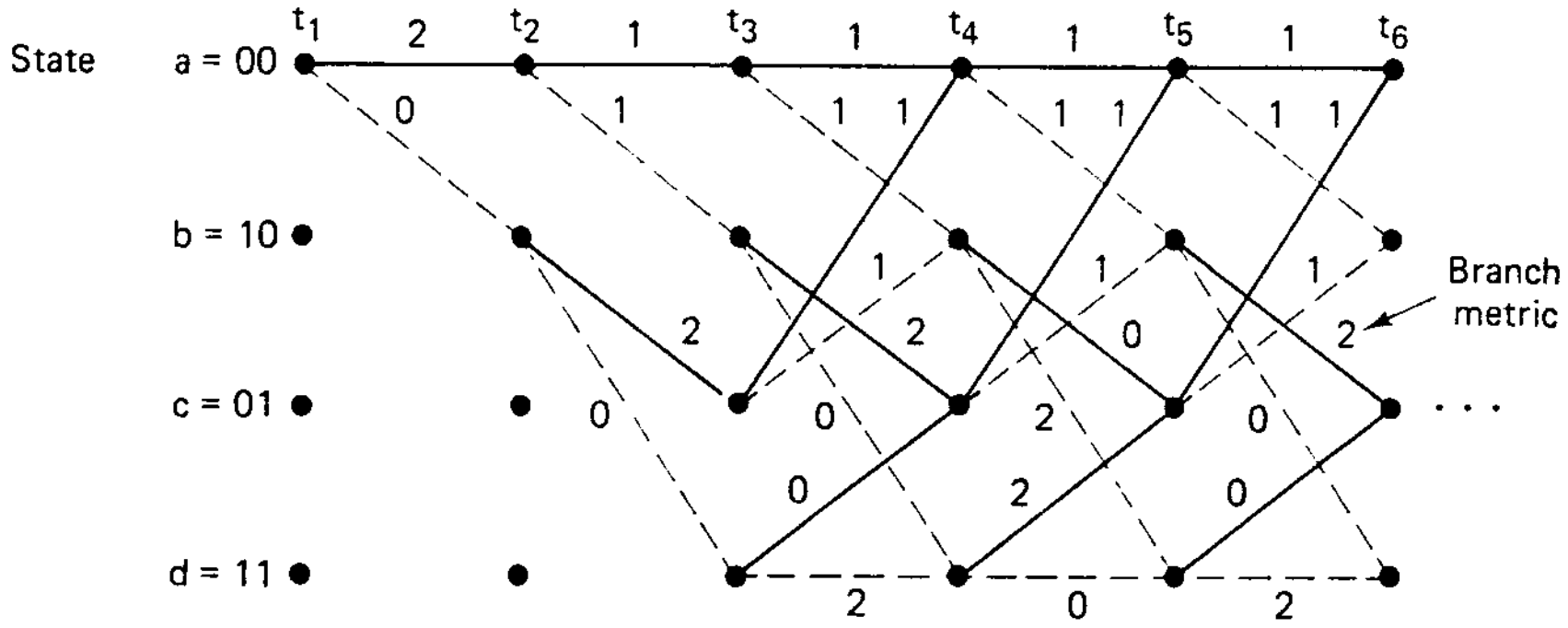
Figure 6.9 Binary symmetric channel (hard-decision channel).

Decoding

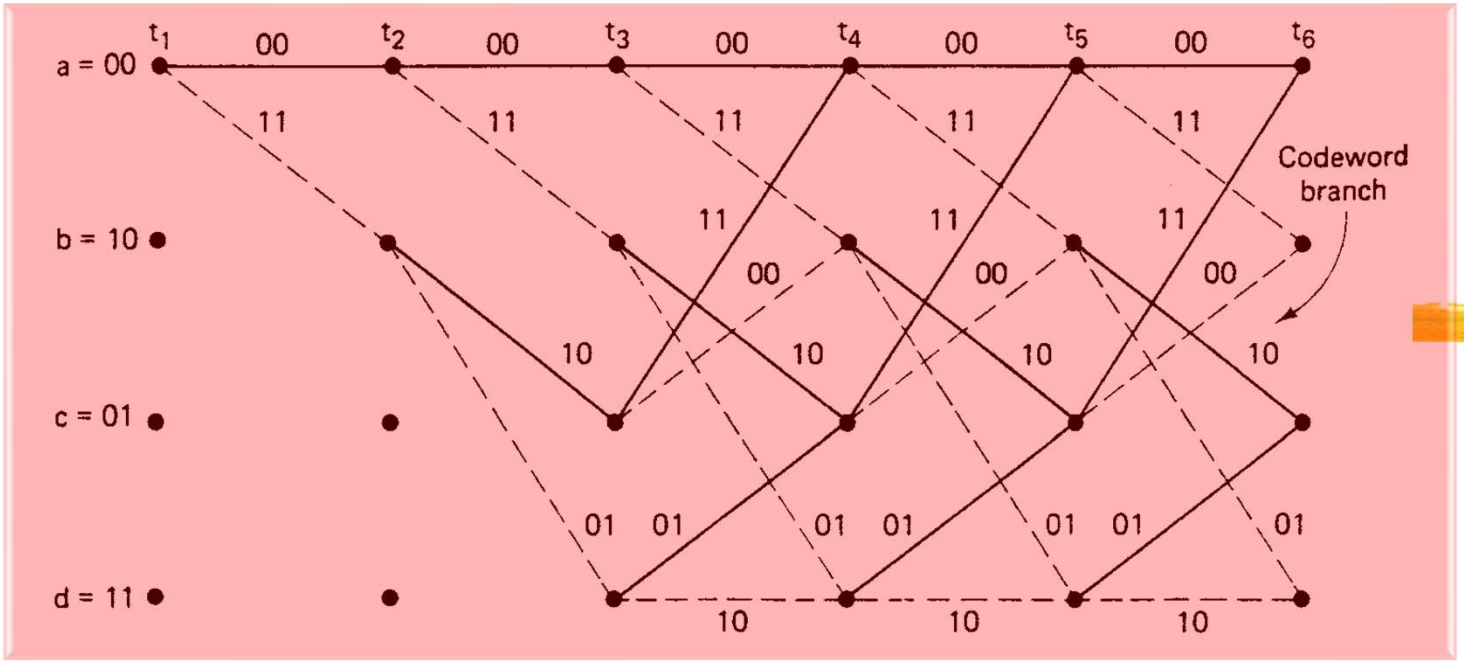
- Correlation introduced between the code words
- Decoder must exploit this correlation
- Maximum Likely Detector (ML) is a **Maximum likelihood sequence estimation (MLSE)**
- Effective algorithm for MLSE: Viterbi

Branch metrics

Input data sequence	m:	1	1	0	1	1	· · ·
Transmitted codeword	U:	11	01	01	00	01	· · ·
Received sequence	Z:	11	01	01	10	01	· · ·

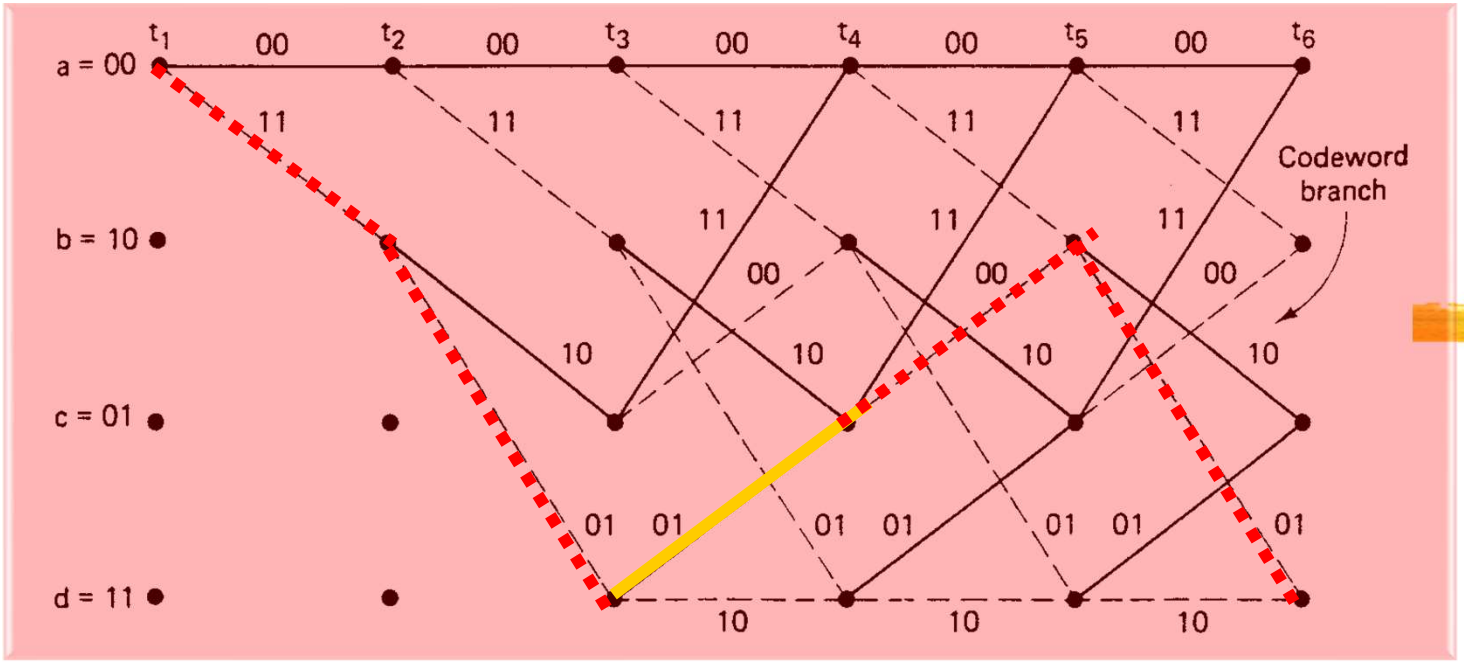


Encoder



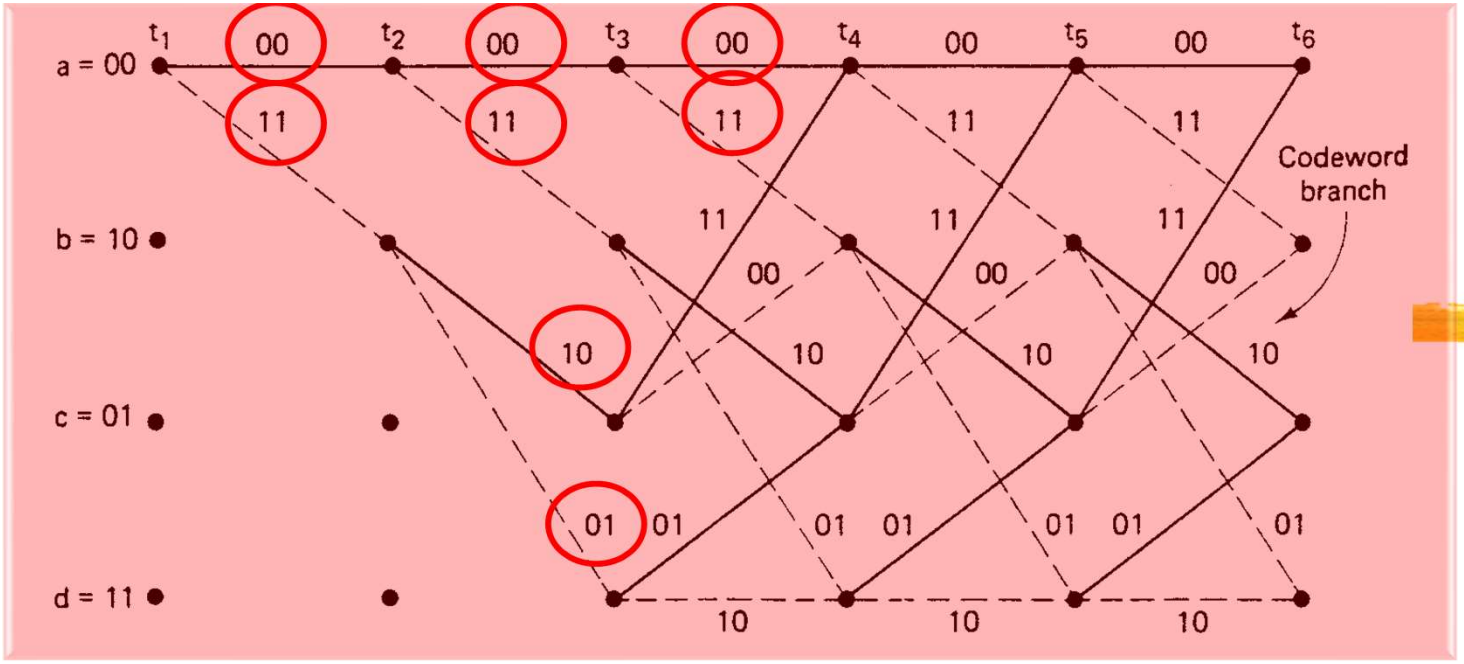
$$m = 1 \quad 1 \quad 0 \quad 1 \quad 1$$

Encoder

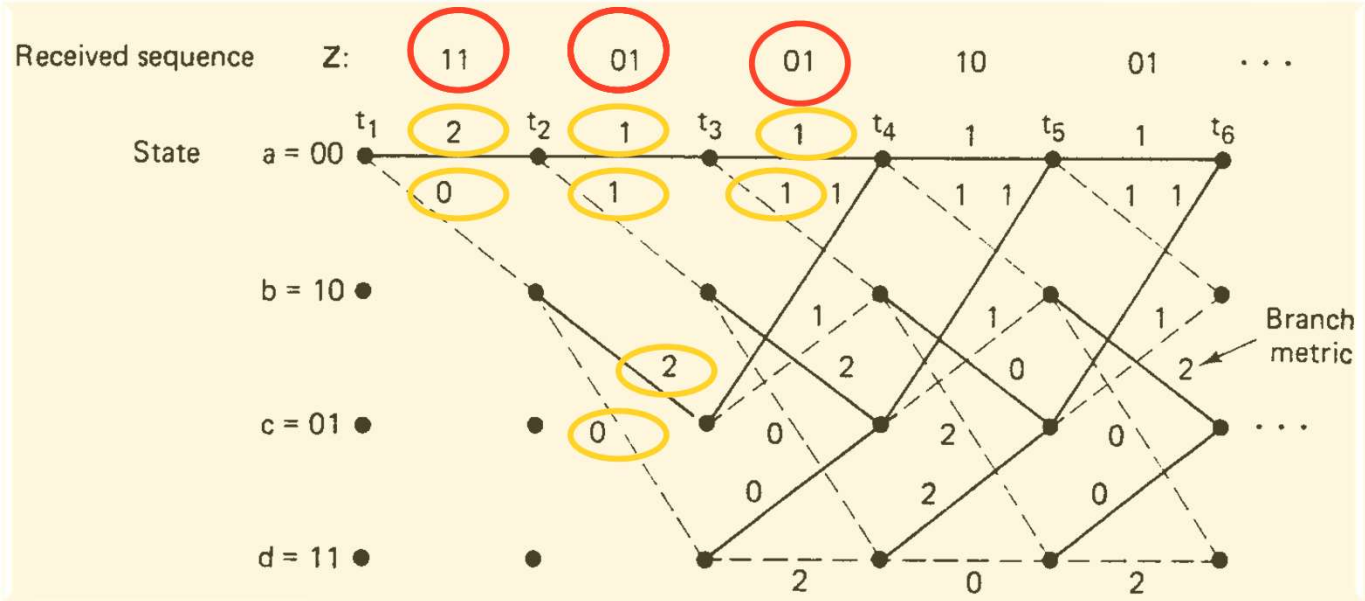


11 01 01 00 01 output

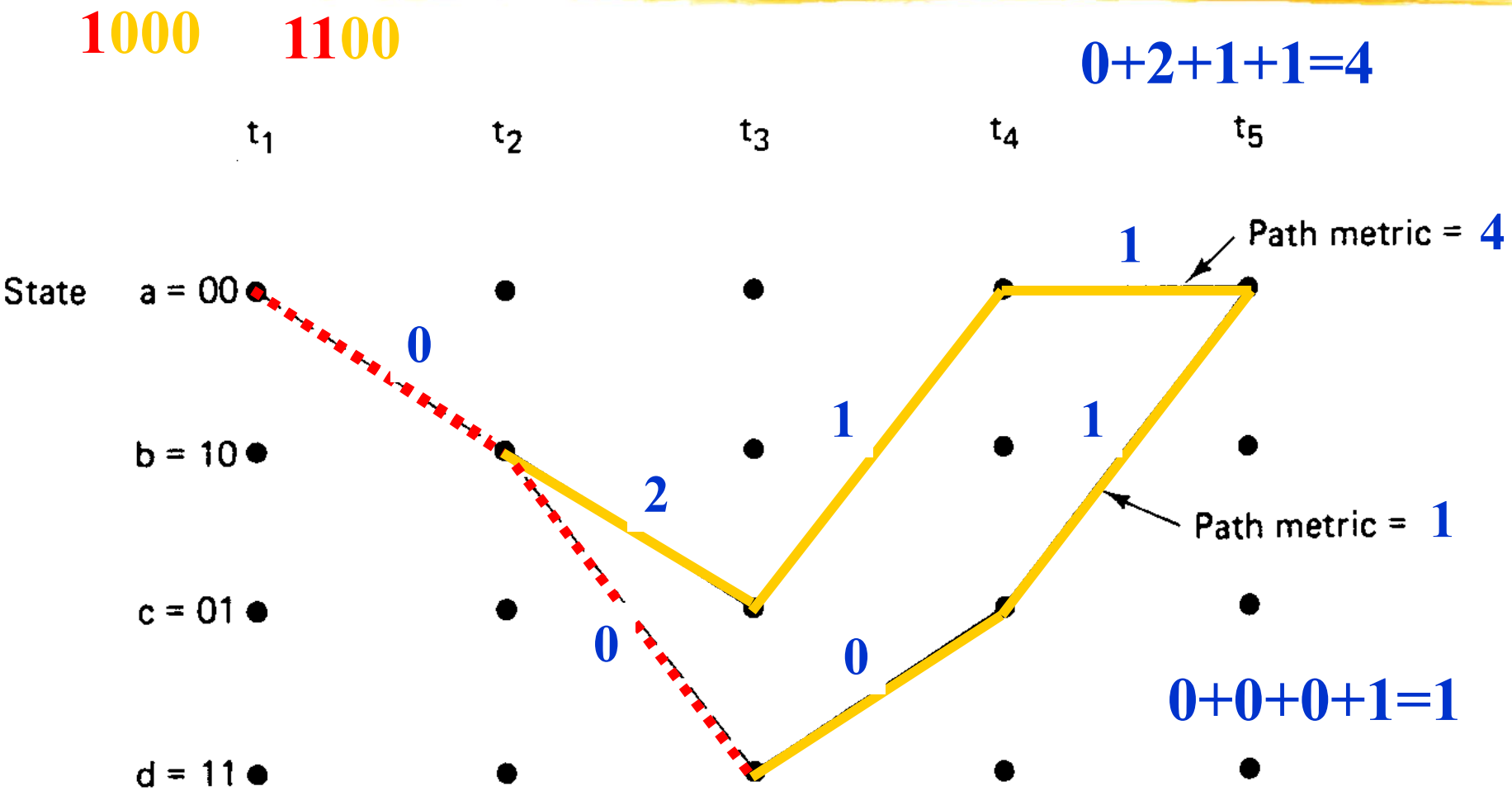
Encoder



Decoder



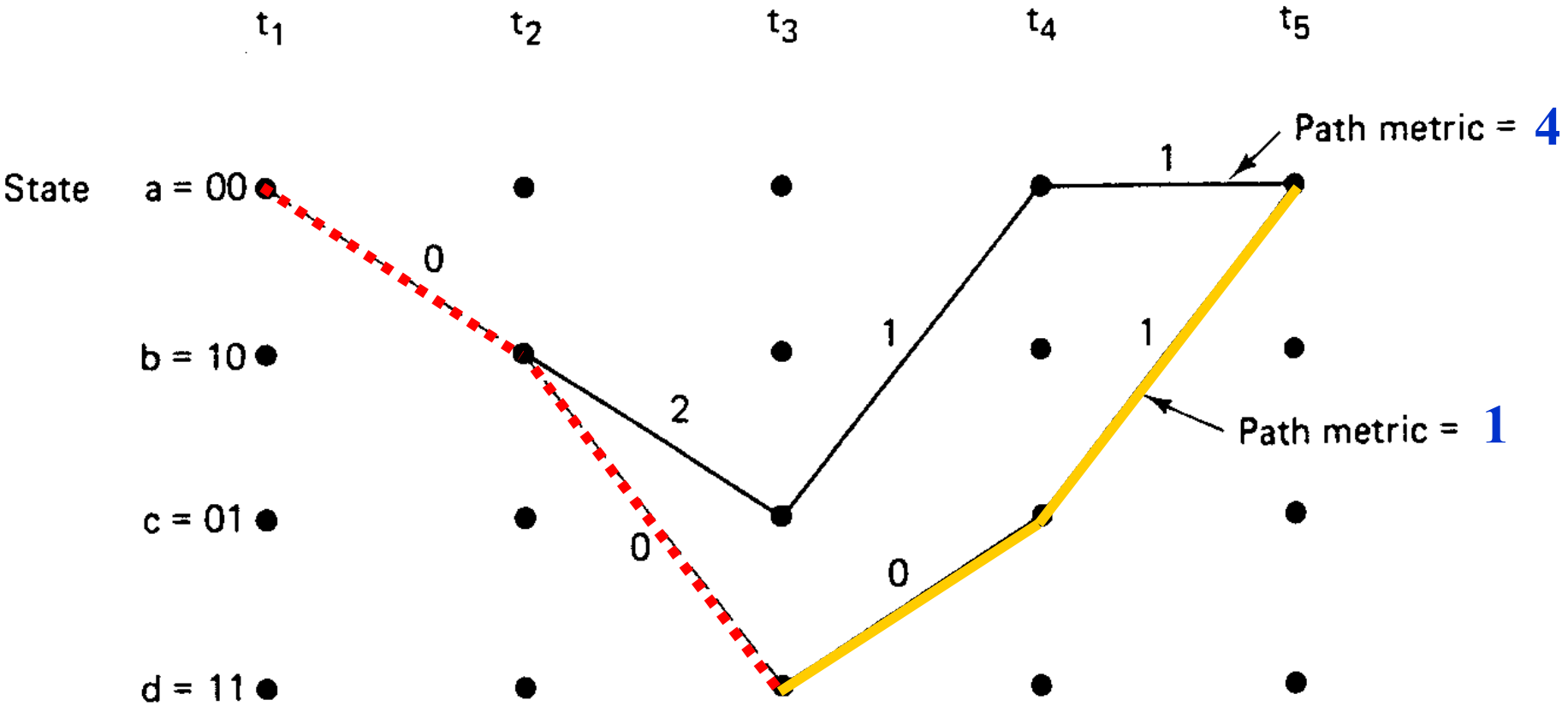
Path choice


1000
1100
0+2+1+1=4
t1
t2
t3
t4
t5
State
a = 00
b = 10
c = 01
d = 11
0
2
1
1
1
Path metric = 4
Path metric = 1
0
0
0+0+0+1=1

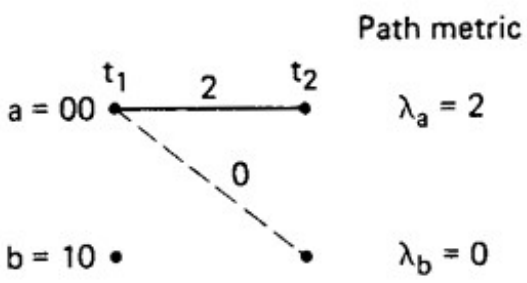
Path choice

<

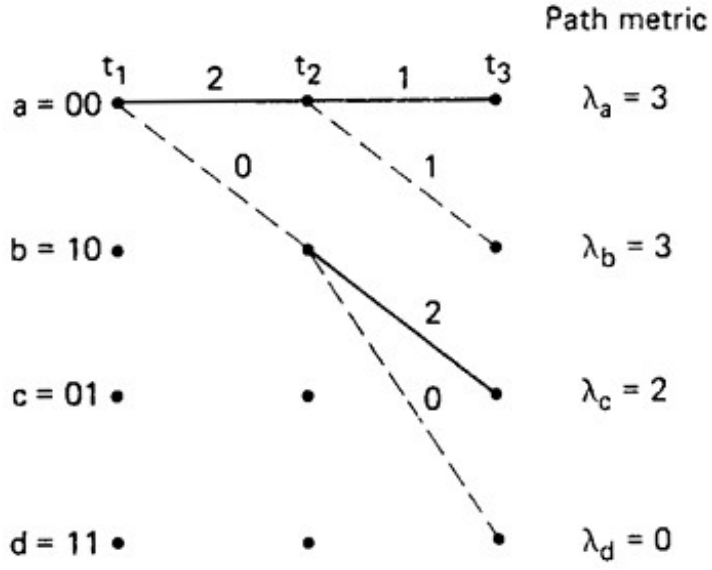
1100 **surviving path**



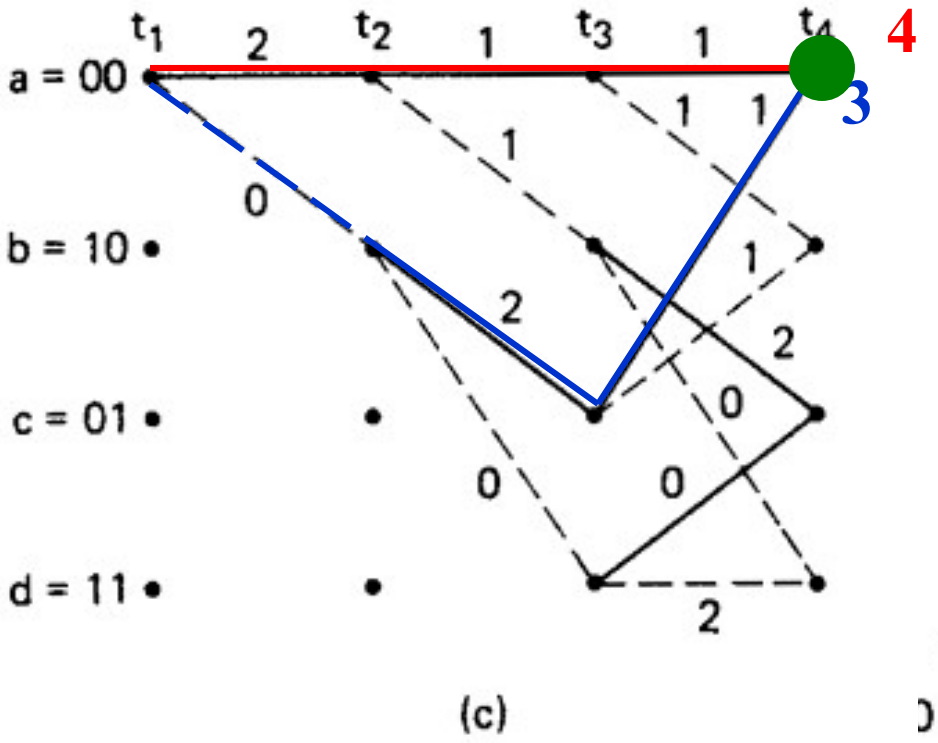
Example in Sklar


Path metric
t_1
2
t_2
a = 00
$\lambda_a = 2$
0
b = 10
$\lambda_b = 0$

(a)

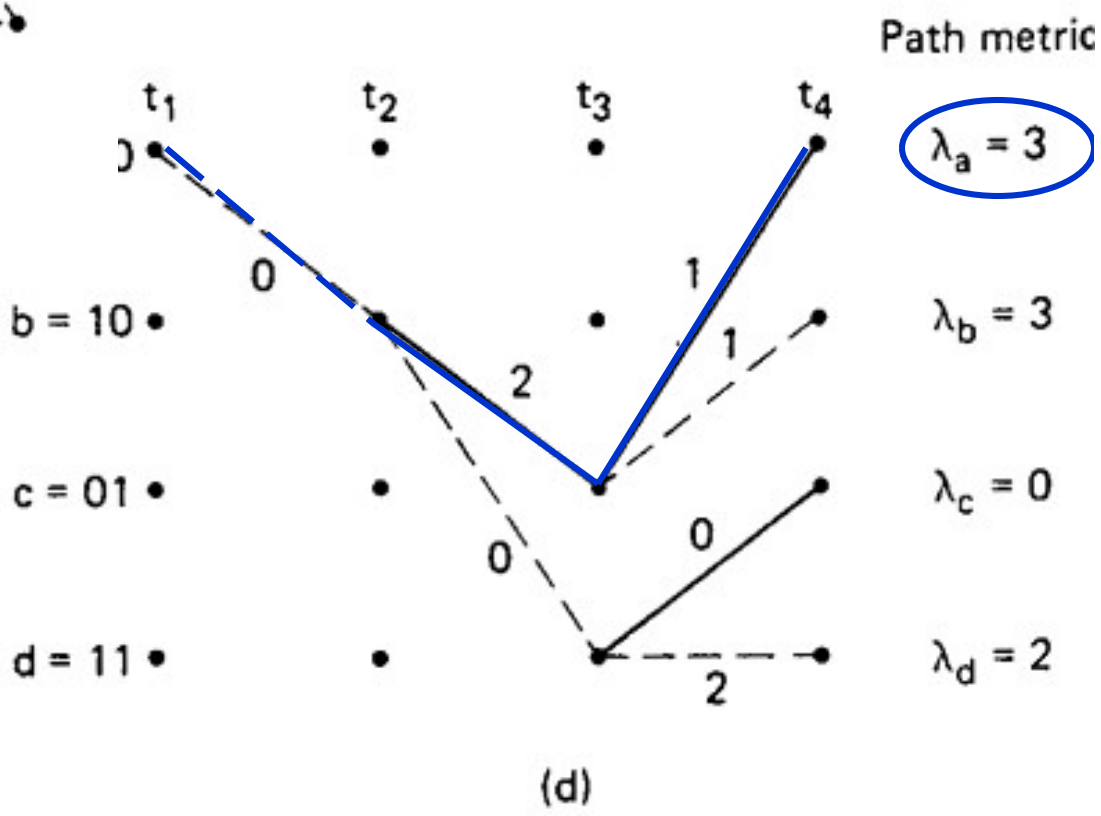

Path metric
t_1
2
t_2
1
t_3
a = 00
$\lambda_a = 3$
0
1
b = 10
$\lambda_b = 3$
2
c = 01
$\lambda_c = 2$
0
d = 11
$\lambda_d = 0$

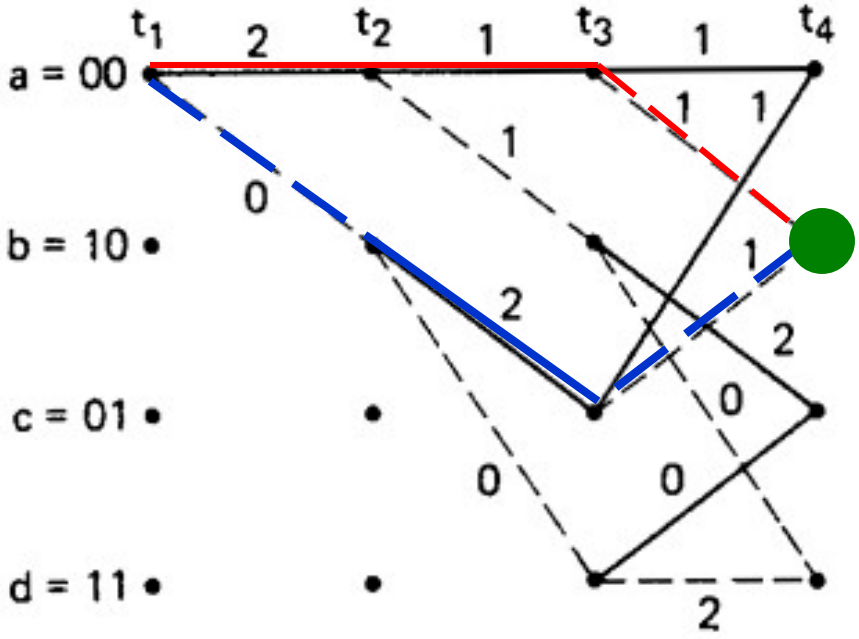
(b)



Focus: endpoint - where does the path come from?

Losing path removed



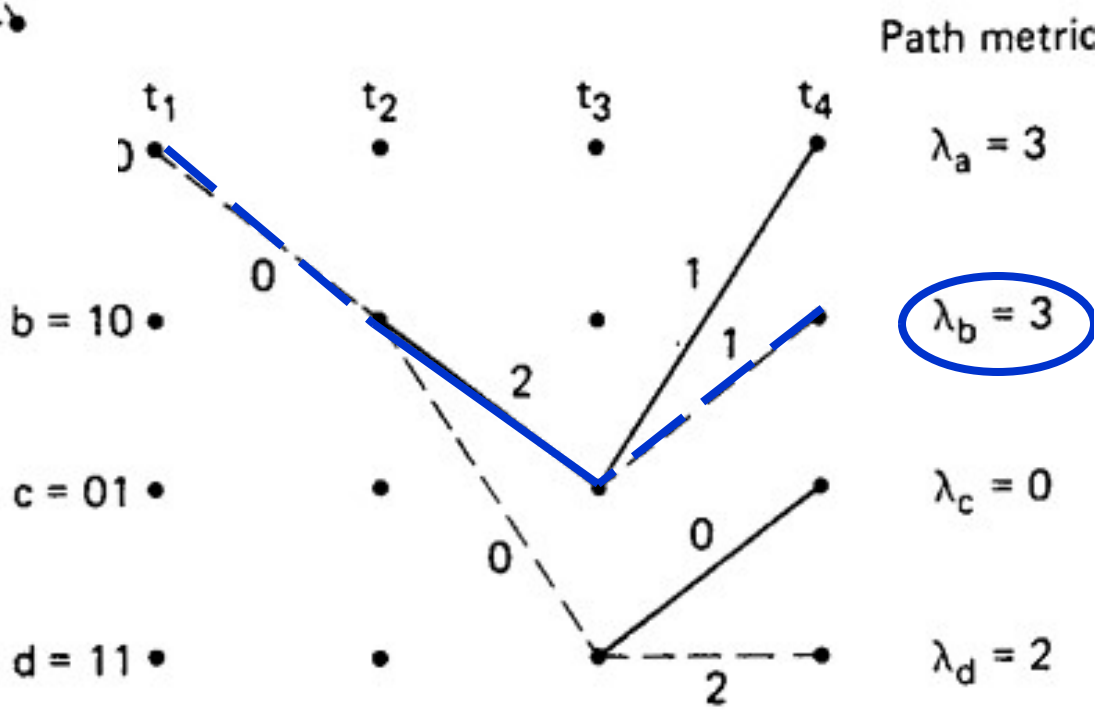


(c)

4

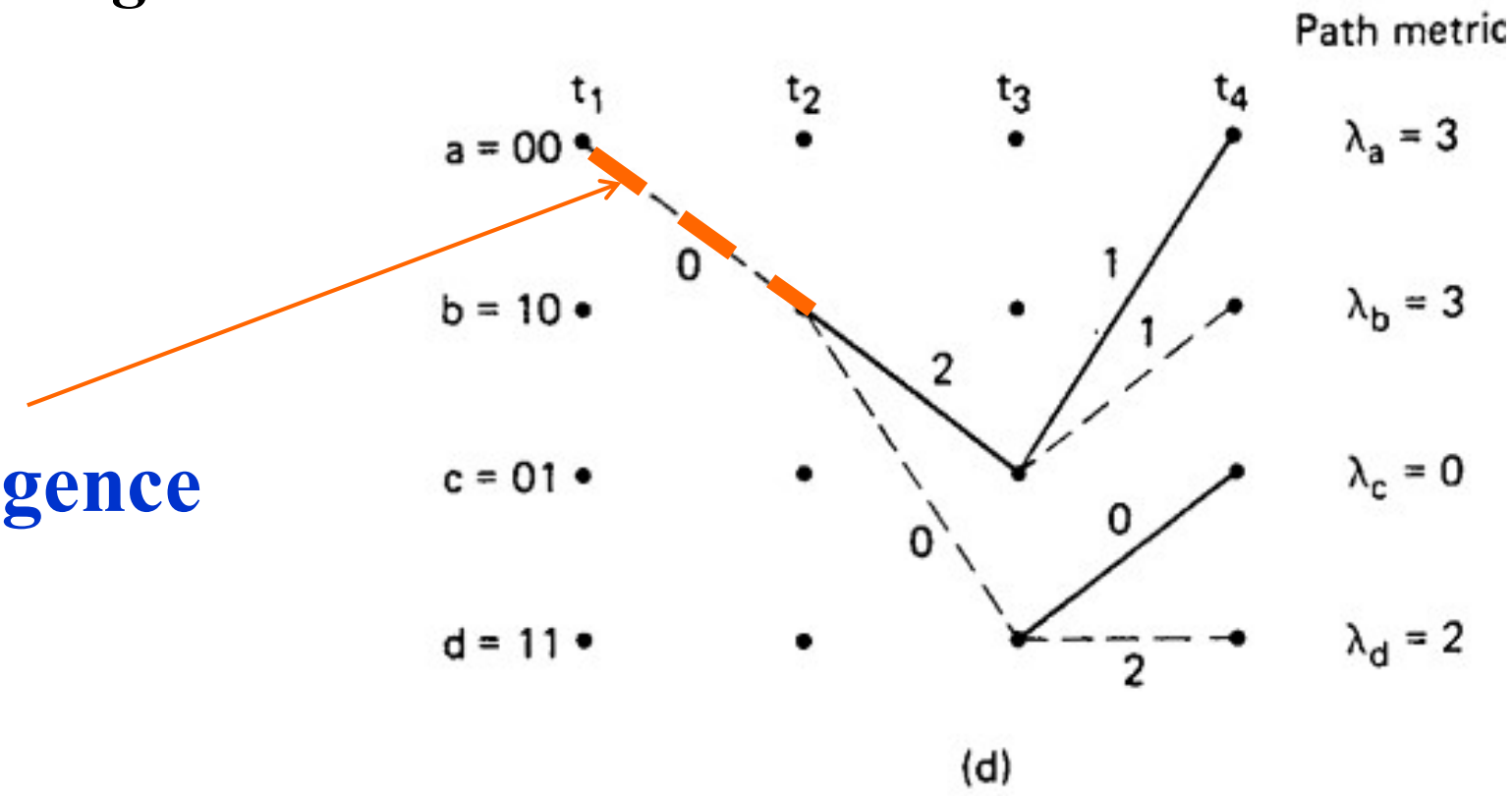
3

Focus: the endpoint - where does the path come from?

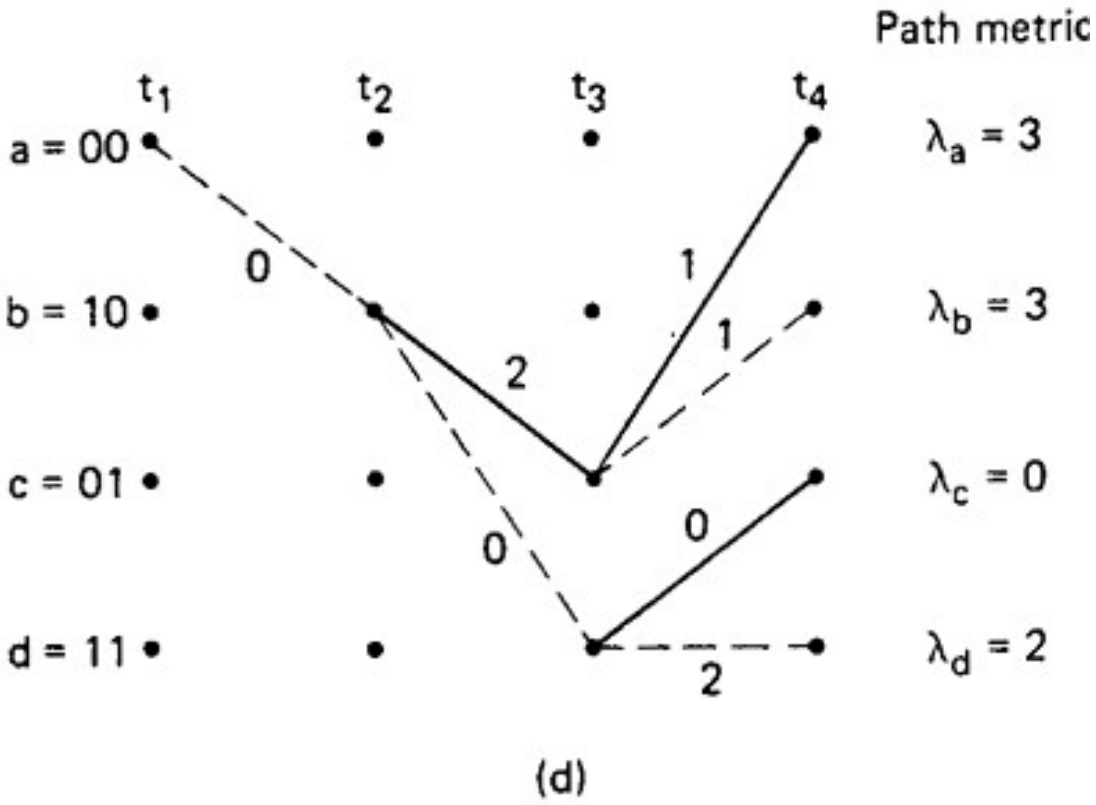


(d)

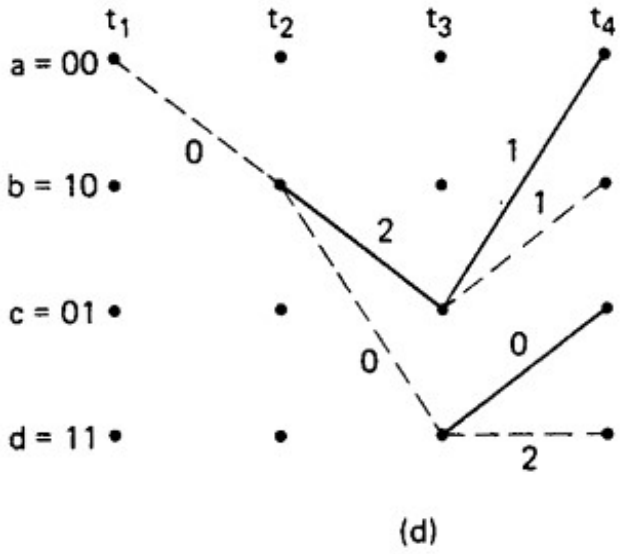
Forcing convergence after a
truncation length
trace back length



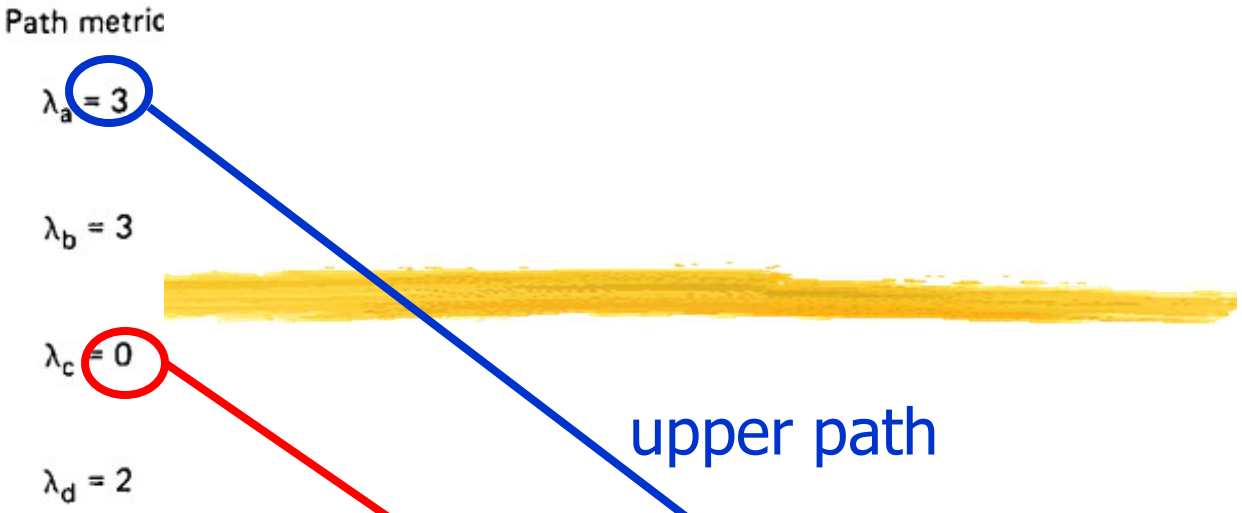
Convergence
merge



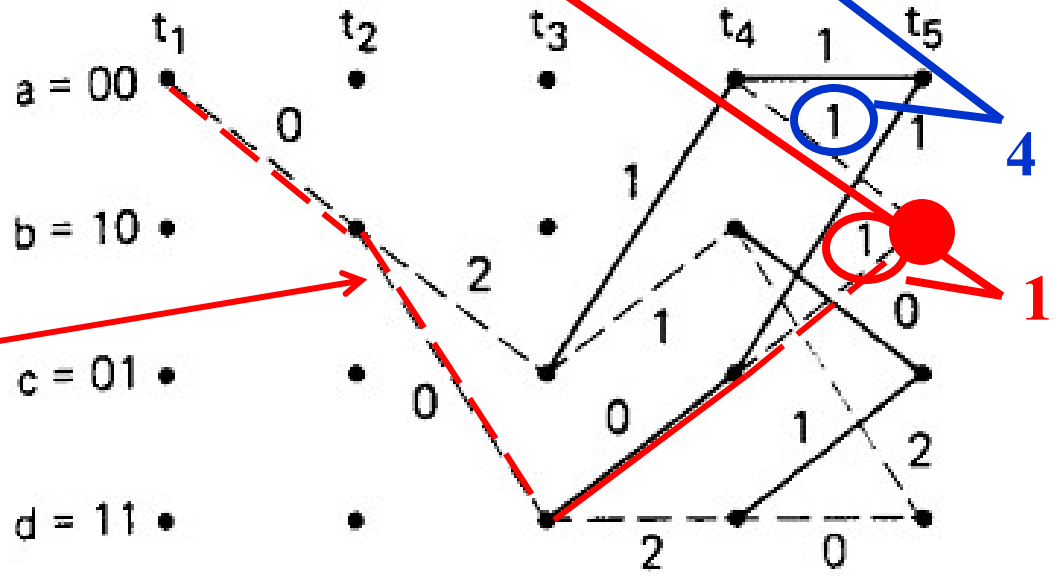
(d)



(d)

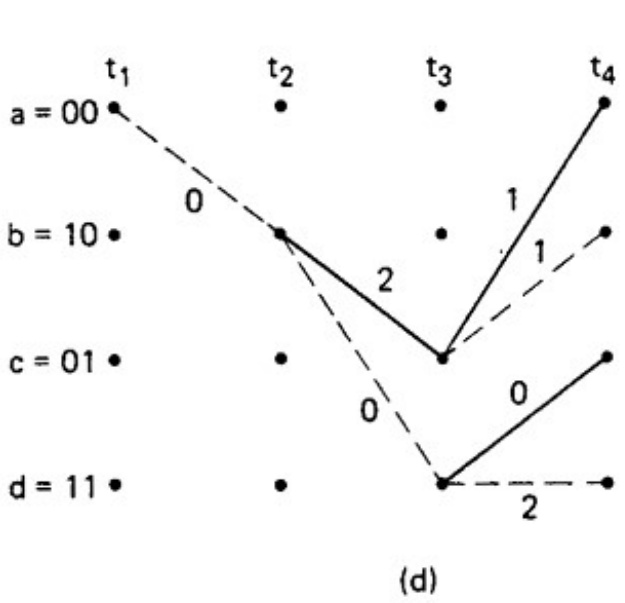


surviving path

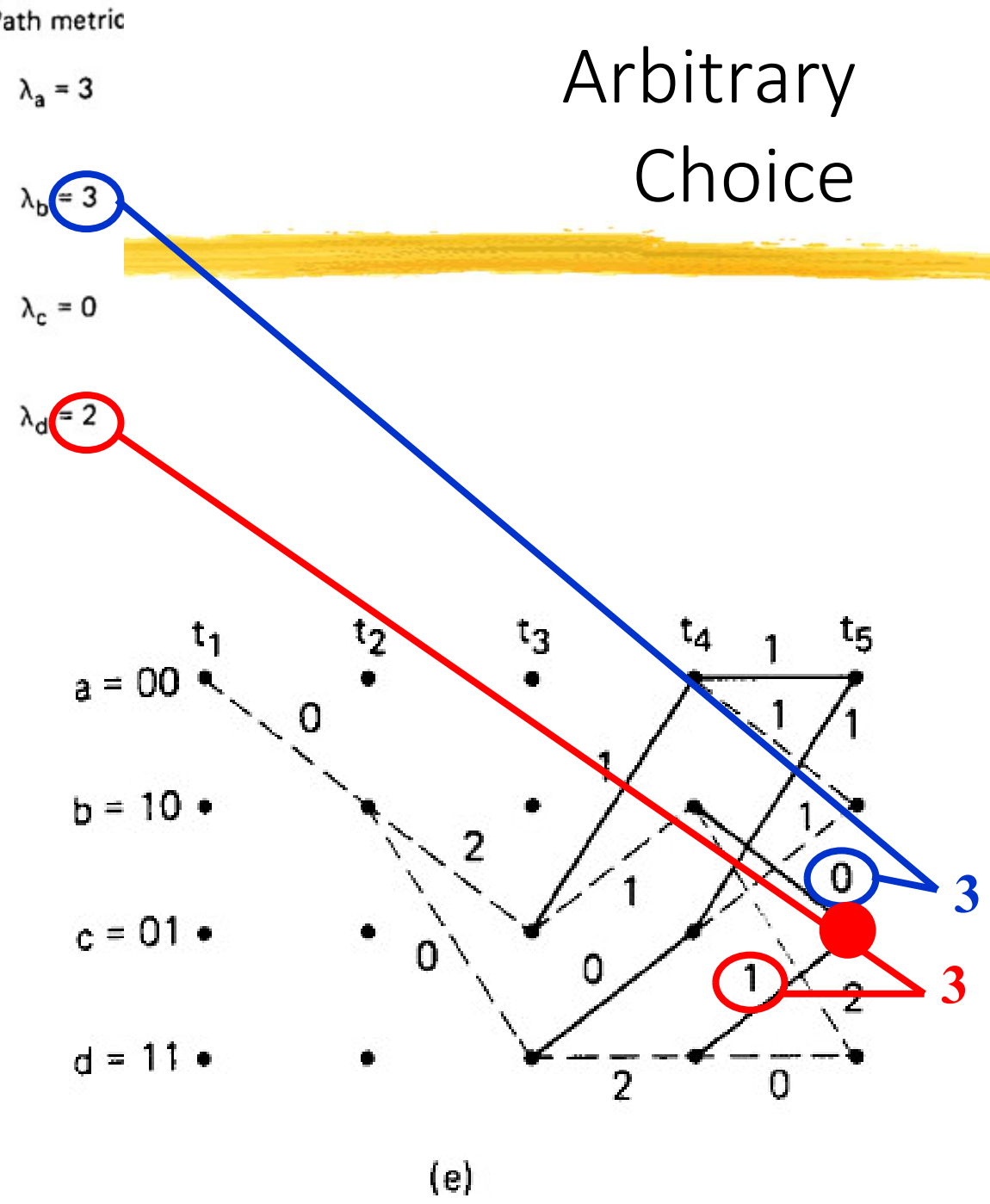


(e)

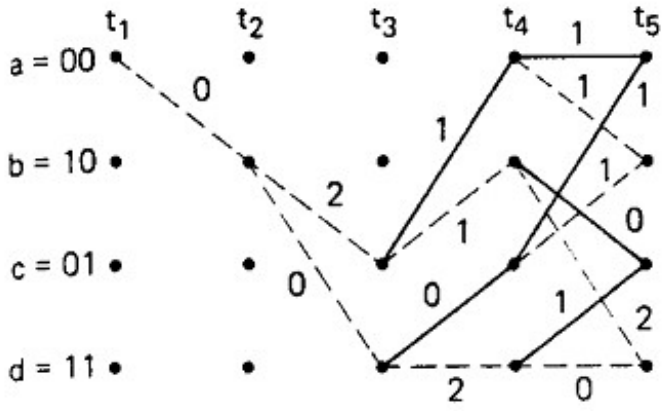
Arbitrary Choice



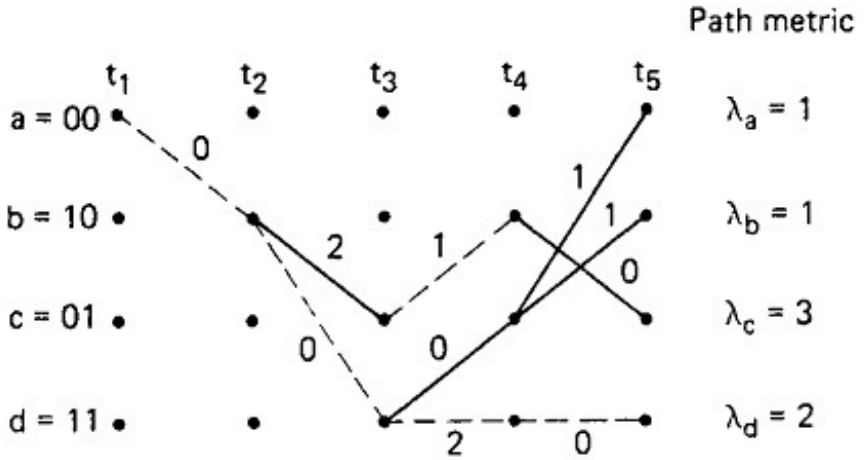
(d)



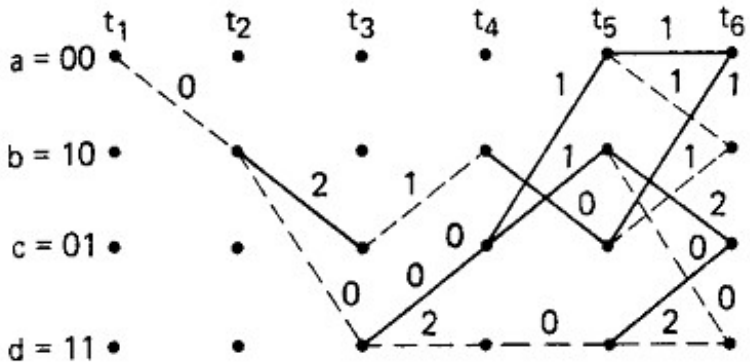
(e)


t1 t2 t3 t4 t5
a = 00
b = 10
c = 01
d = 11

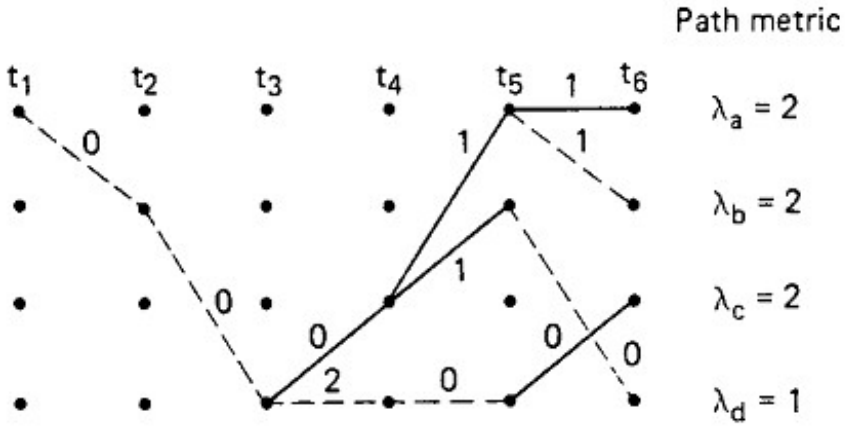
(e)


Path metric
t1 t2 t3 t4 t5
a = 00
b = 10
c = 01
d = 11
λa = 1
λb = 1
λc = 3
λd = 2

(f)


t1 t2 t3 t4 t5 t6
a = 00
b = 10
c = 01
d = 11

(g)


Path metric
t1 t2 t3 t4 t5 t6
λa = 2
λb = 2
λc = 2
λd = 1

(h)

Complexity and minimum distance

Complexity

- Operations
  - Add-compare-select (ACS)
- Memory
  - h – maximum length of a path
    - force a convergence ($merge$)
    - typically 4 or 5 × the constraint length K
  - $h2^{K-1}$
    - path length × # of states
    - bit sequence for each path
  - 2^{K-1} path metrics

Final exam 2006

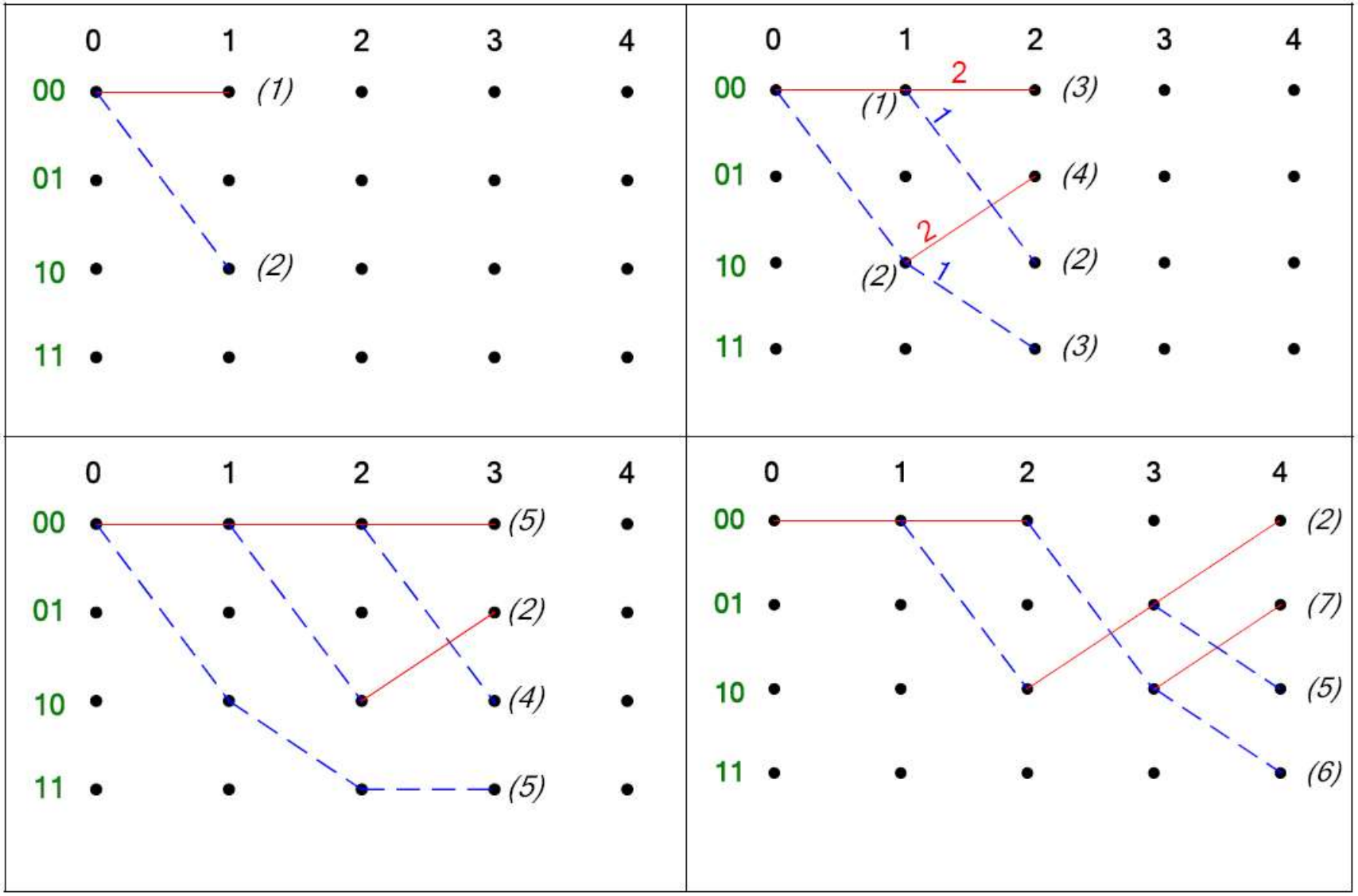
Considérons l'algorithme de Viterbi pour décoder un code convolutif. Supposons que la longueur de contrainte est K, qu'il y a un bit (k=1) qui entre dans les K registres à chaque intervalle, qu'il y a n bits de parité, que nous utilisons des décisions fermes, et que nous avons choisi de forcer une décision après h=5K bits.

A. (15 points) Quelle information doit être sauvegardée pour chaque intervalle de bit et quelle est la quantité d'information à sauvegarder?

B. (10 points) Donnez l'information sauvegardée pour les quatre intervalles de bits illustrés dans la table suivante.

Donnée 0 : ——————

Donnée 1 : — — — —



B. ~~CHEMIN~~ $K=3$ $2^{3-1}=4$ états $n = 5 \times 3 = 15$

	État	Chemin	Distance
t=1	Chemin état A	01	dist état A: 1
	B		-
	C	11	dist état C: 2
	D		-
t=2	A	00	dist A 3
	B	10	4
	C	01	2
	D	11	3
t=3	A	000	5
	B	010	2
	C	001	4
	D	111	5
t=4	A	0100	2
	B	0010	7
	C	0101	5
	D	0011	6

Distance between code words

- We are still talking about codeword sequences for convolutional codes.
- Consider all possible sequences, and the distance between each pair
- Minimum distance
  - minimum among all pairs of sequences
- Linear code
  - minimum distance between zero paths and other sequences is equivalent

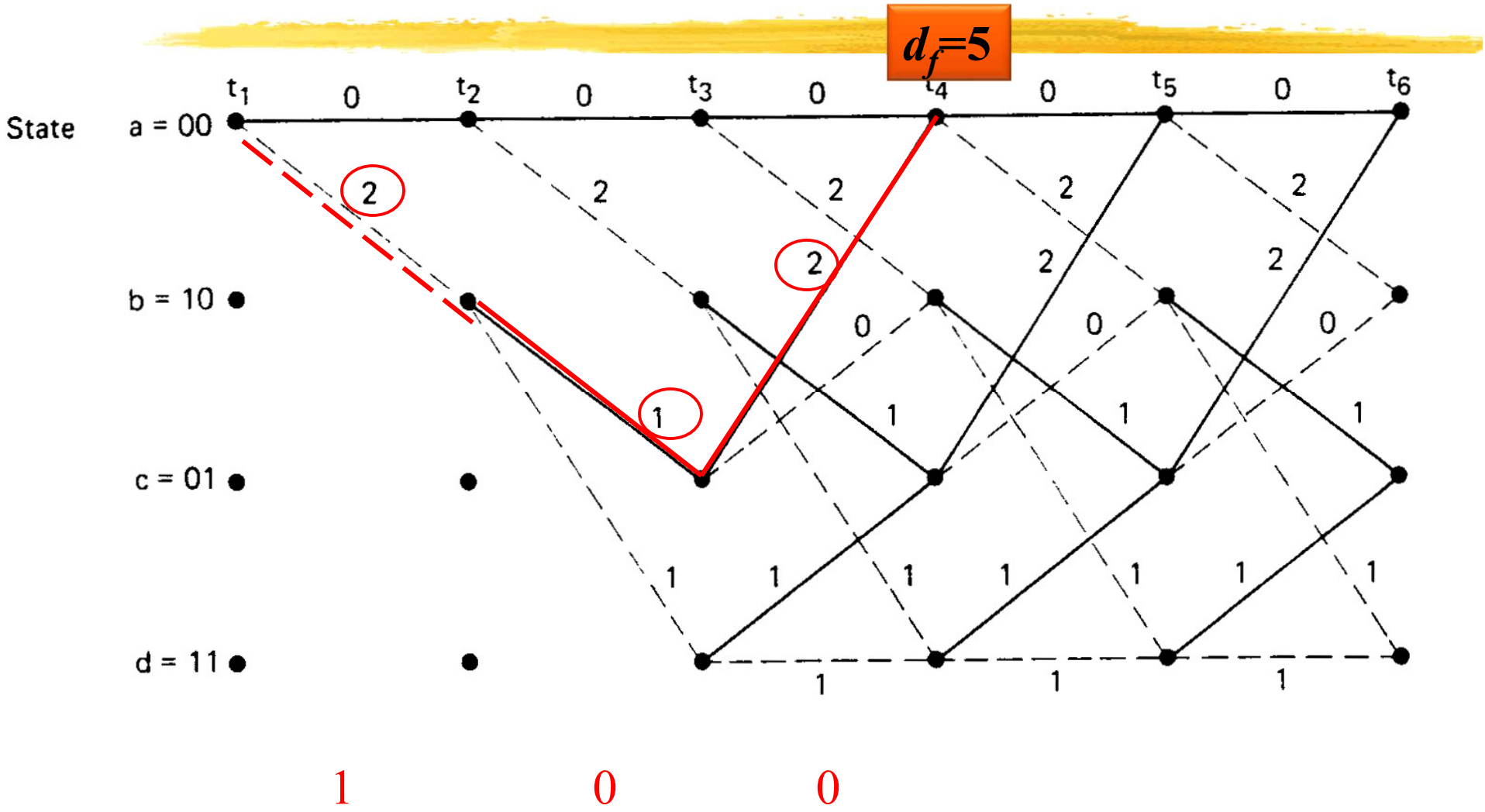
Minimum distance

- Procedure
  1. Starts in state a
  2. Finish in state a
  3. Smallest distance □
     length = free distance

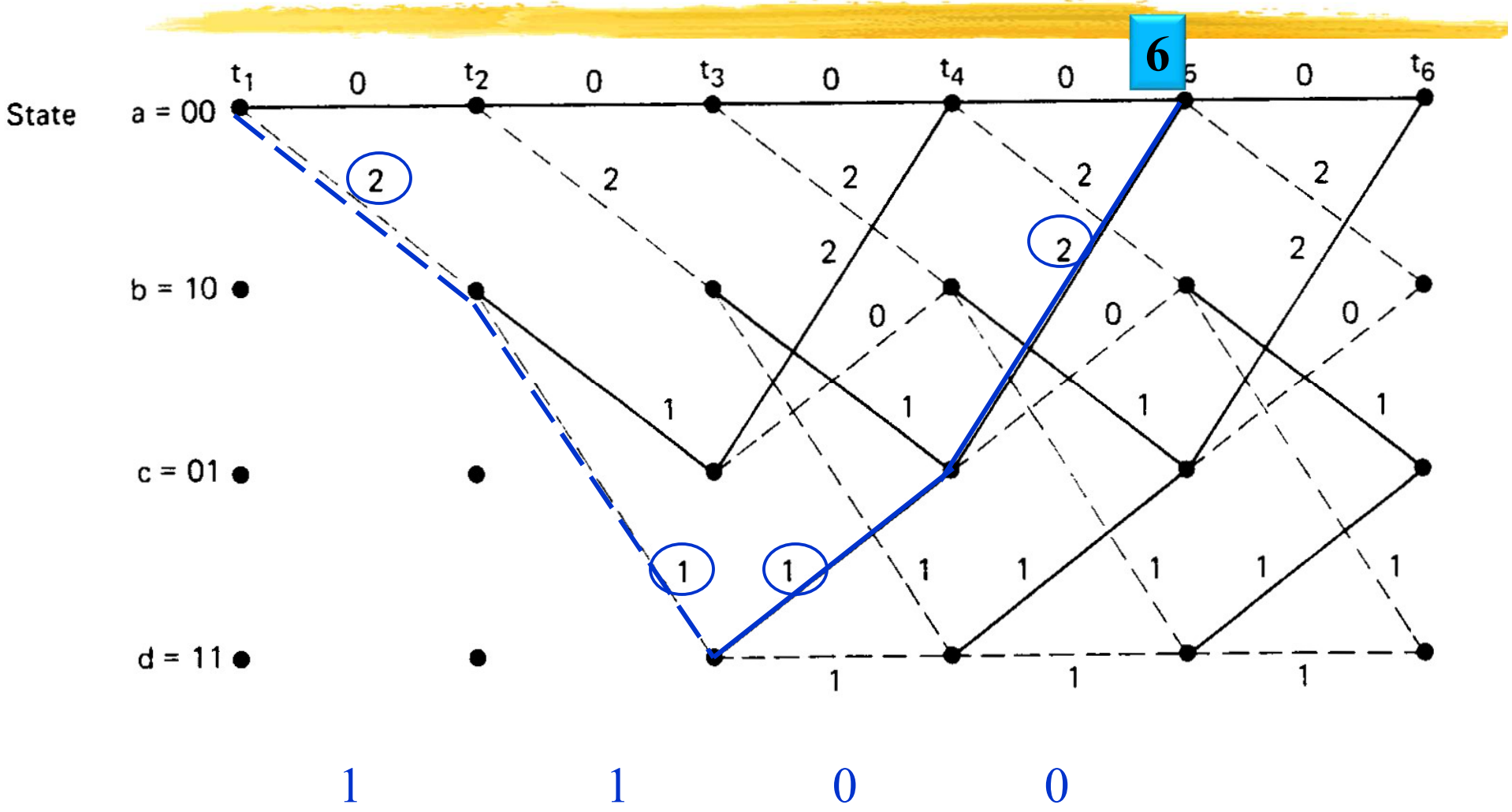
- The minimum distance is a measure of the code's ability to correct an erroneous path

$$t = \left\lfloor \frac{d_f - 1}{2} \right\rfloor$$

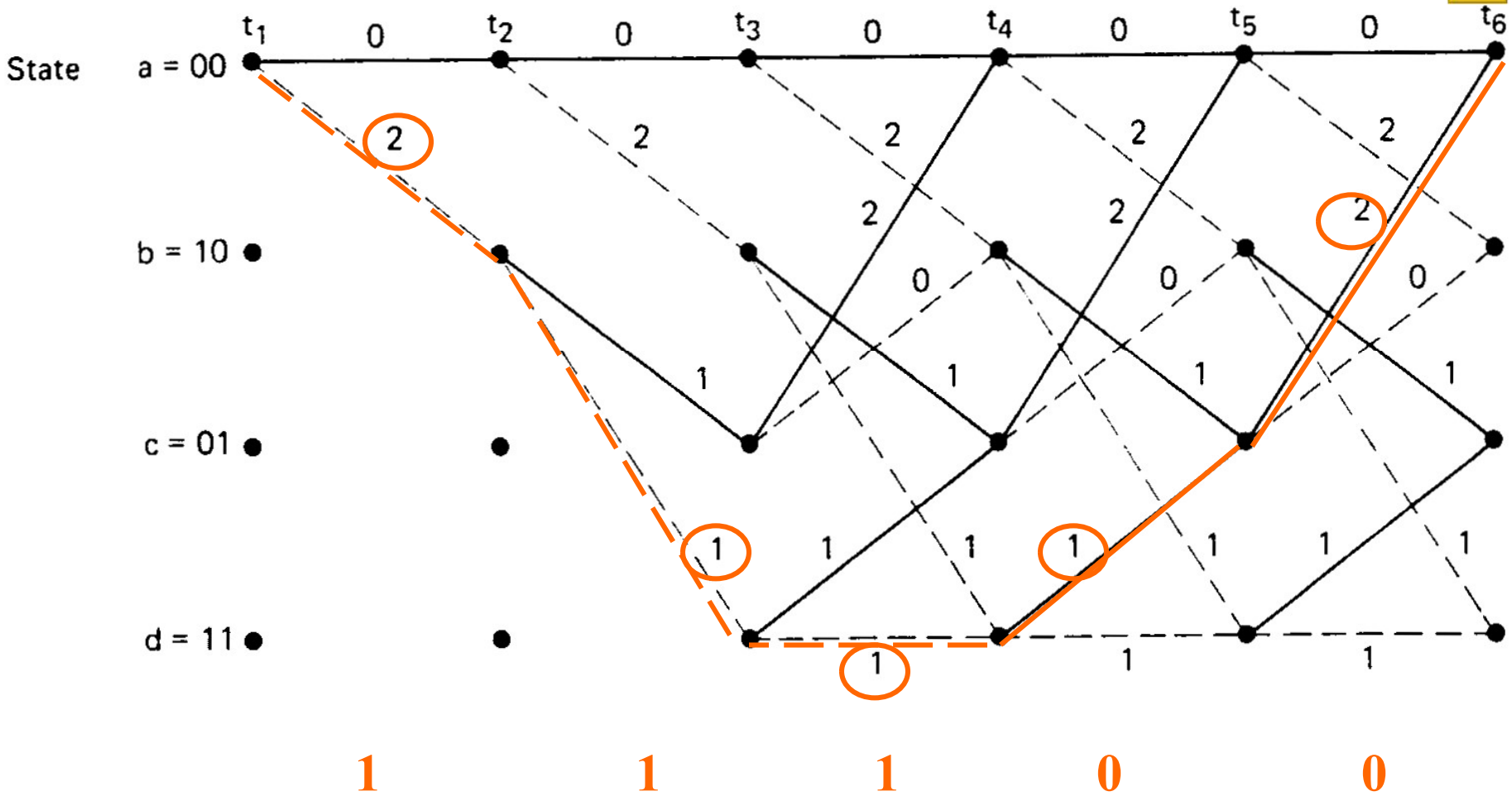
Free distance : 5



Exemple

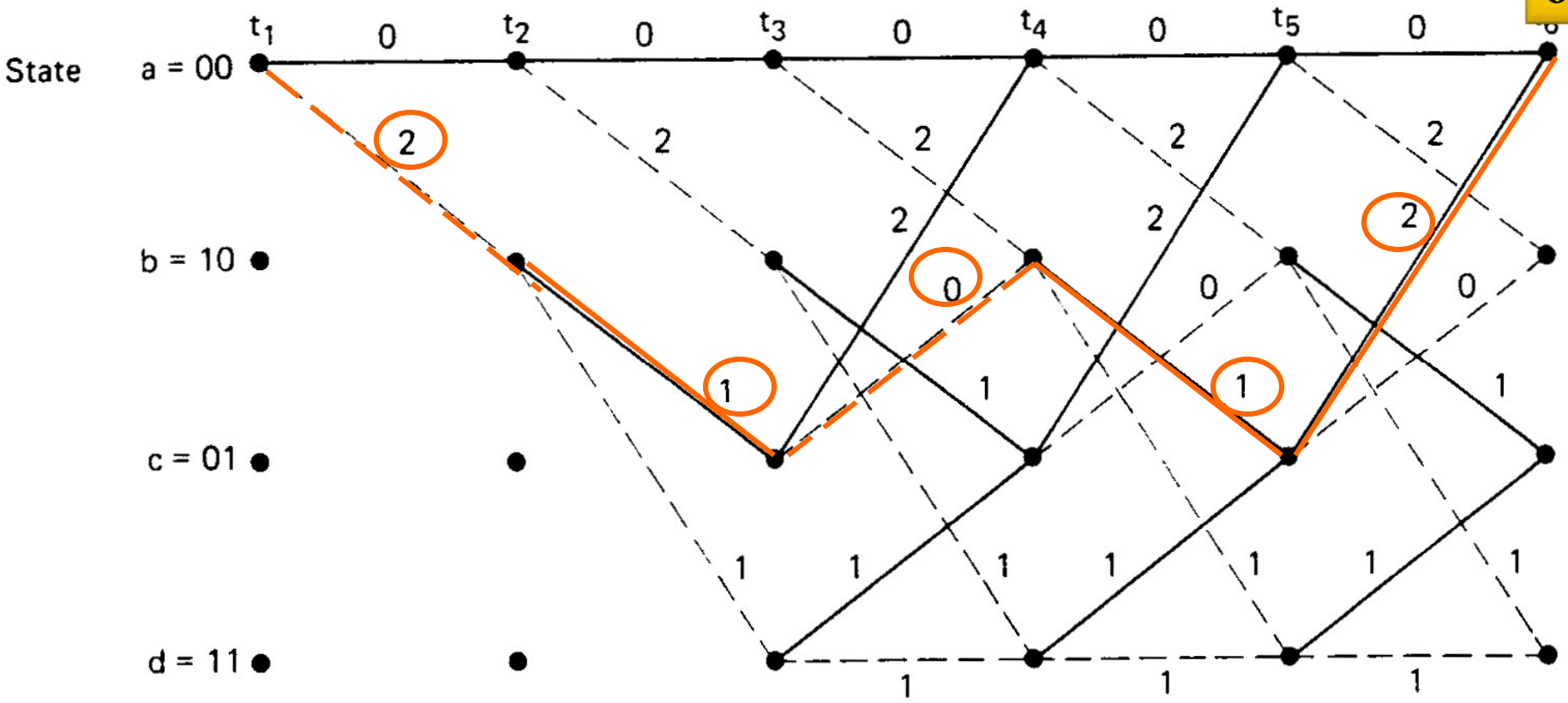


Exemple

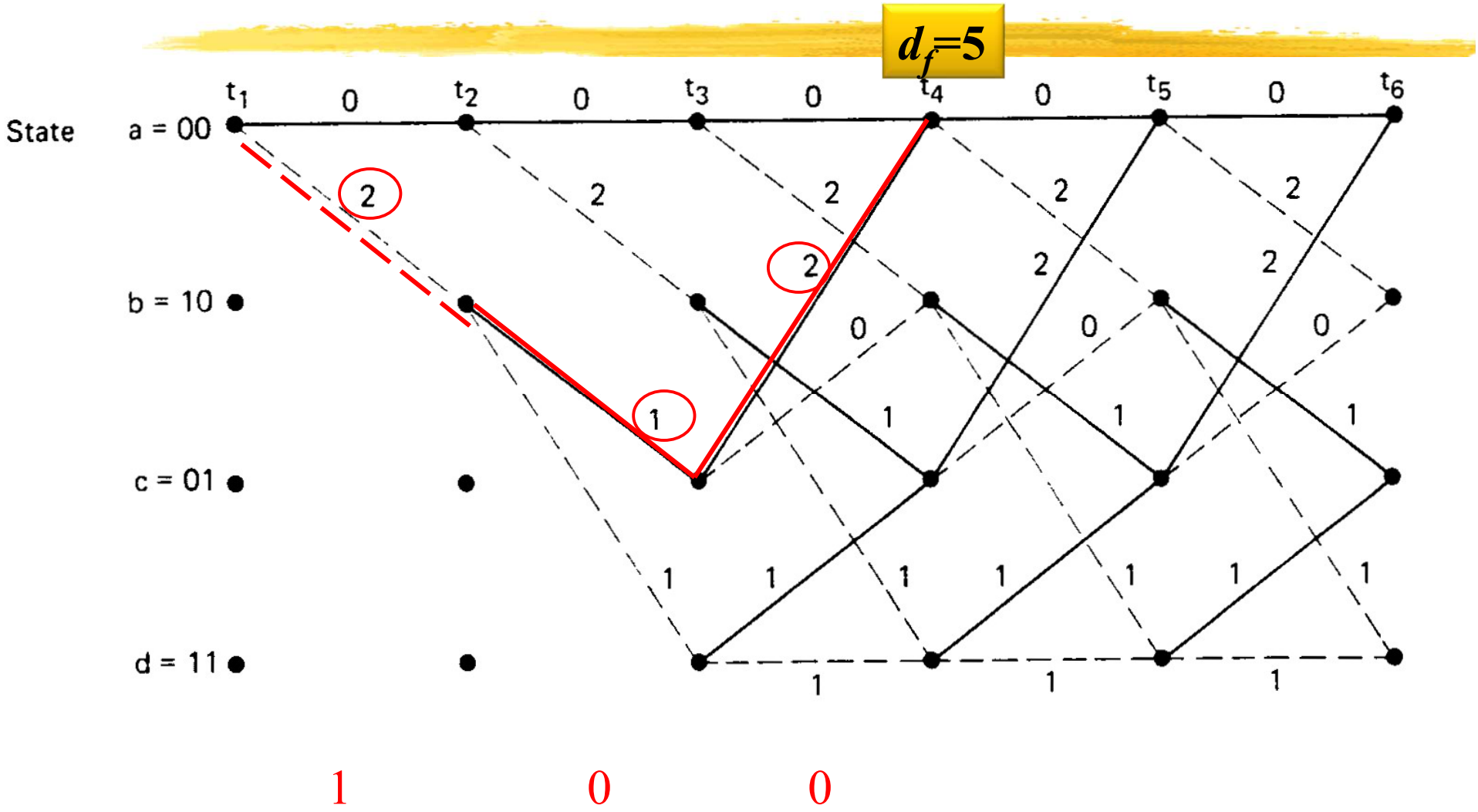


Exemple

6



Free distance : 5



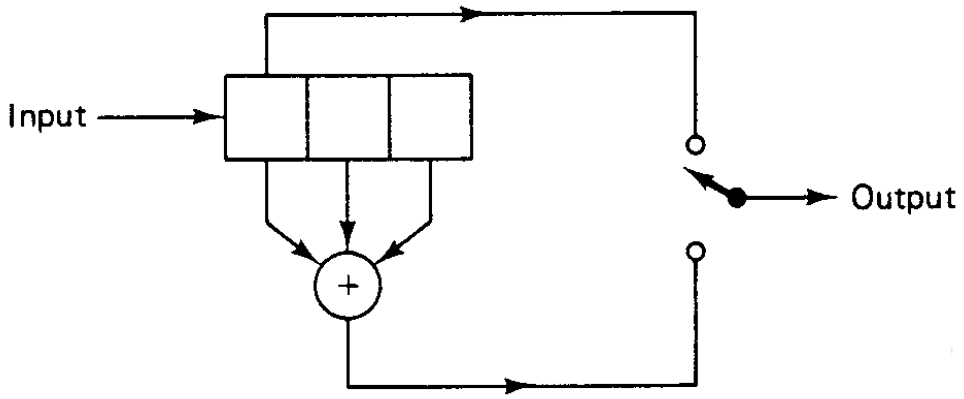
GEL4200

Definitions and coding gain

Definitions

- Systematic code
  - Data readable in code
- Cyclical code
  - Each code word is a cyclic rotation of another code word
- Catastrophic code
  - Code where finite # of errors in the code can generate an infinite # of errors in the data
- Perfect code
  - $t = \frac{d_f - 1}{2}$

Systematic codes



- Less efficient than non-systematic codes for convolutional codes

TABLE 6.1 Comparison of Systematic and Nonsystematic Free Distance, Rate ½

Constraint length	Free distance systematic	Free distance nonsystematic
2	3	3
3	4	5
4	4	6
5	5	7
6	6	8
7	6	10
8	7	10

Source: A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill Book Company, New York, 1979, p. 251.

Possible gain

$$10\log_{10} rd_f$$

TABLE 6.2 Coding Gain Upper Bounds for Some Convolutional Codes

Rate ½ codes			Rate ⅓ codes		
K	d_f	Upper bound (dB)	K	d_f	Upper bound (dB)
3	5	3.97	3	8	4.26
4	6	4.76	4	10	5.23
5	7	5.43	5	12	6.02
6	8	6.00	6	13	6.37
7	10	6.99	7	15	6.99
8	10	6.99	8	16	7.27
9	12	7.78	9	18	7.78

Source: V. K. Bhargava, D. Haccoun, R. Matyas, and P. Nuspl, *Digital Communications by Satellite*, John Wiley & Sons, Inc., New York, 1981.

Gain for given SNR

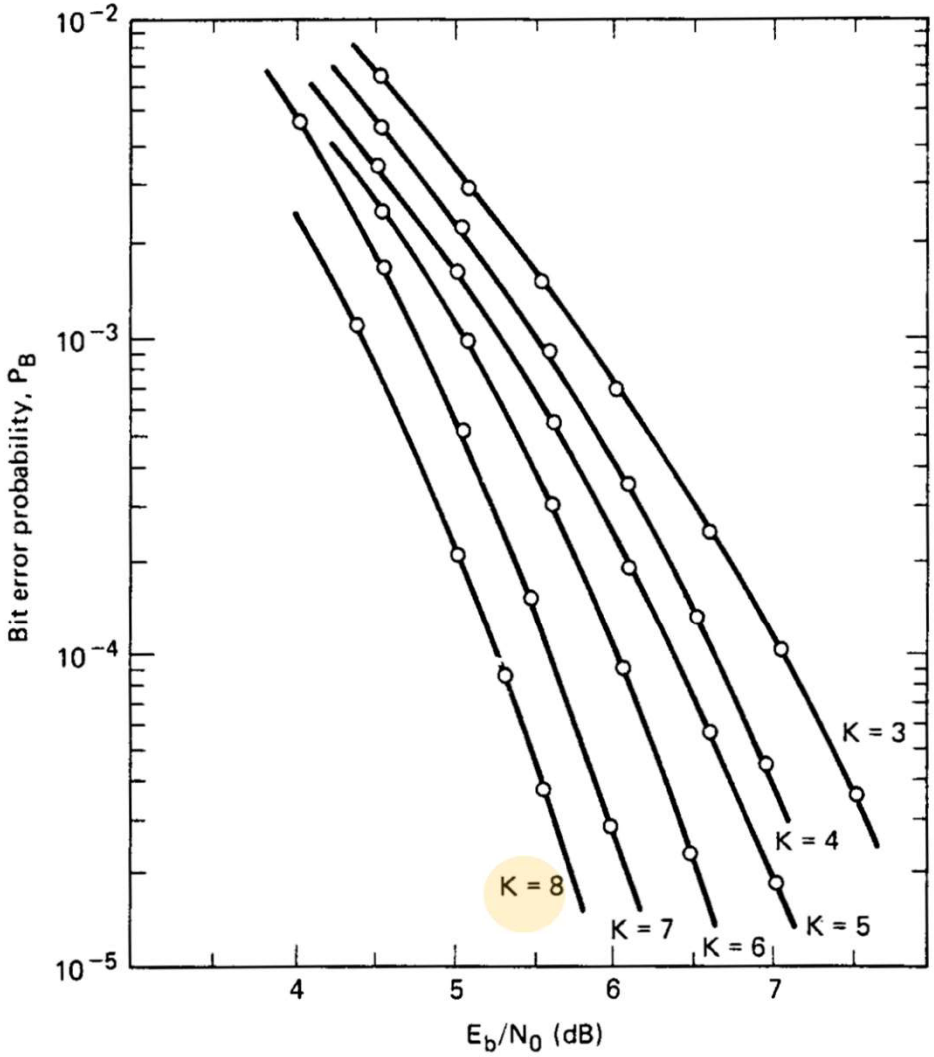
TABLE 6.3 Basic Coding Gain (dB) for Soft Decision Viterbi Decoding AWGN

Uncoded E_b/N_0 (dB)	P_B	code rate / K	1/3		1/2			2/3		3/4	
			7	8	5	6	7	6	8	6	9
6.8	10^{-3}		4.2	4.4	3.3	3.5	3.8	2.9	3.1	2.6	2.6
9.6	10^{-5}		5.7	5.9	4.3	4.6	5.1	4.2	4.6	3.6	4.2
11.3	10^{-7}		6.2	6.5	4.9	5.3	5.8	4.7	5.2	3.9	4.8
Upper bound			7.0	7.3	5.4	6.0	7.0	5.2	6.7	4.8	5.7

Source: I. M. Jacobs, "Practical Applications of Coding," *IEEE Trans. Inf. Theory*, vol. IT20, May 1974, pp. 305–310.

BER

- Fixed rate, therefore fixed bandwidth
- BER improved by adding complexity in the decoder
  - Memory
  - Number of calculations per symbol



BER vs. E_b/N_0, rate ½, coherent BPSK, BSC, Viterbi hard decision, 32-bit path memory

Optimal codes

- Exhaustive research
- Eliminate catastrophic codes
- Minimize
  - Free distance d_f
  - # Paths @ d_f
  - # Paths @ d_f+1
  - # Paths @ d_f+2, etc.

TABLE 6.4 Optimum Short Constraint Length Convolutional Codes (Rate $\frac{1}{2}$ and Rate $\frac{1}{3}$)

Rate	Constraint length	Free distance	Code vector
$\frac{1}{2}$	3	5	111 101
$\frac{1}{2}$	4	6	1111 1011
$\frac{1}{2}$	5	7	10111 11001
$\frac{1}{2}$	6	8	101111 110101
$\frac{1}{2}$	7	10	1001111 1101101
$\frac{1}{2}$	8	10	10011111 11100101
$\frac{1}{2}$	9	12	110101111 100011101
$\frac{1}{3}$	3	8	111 111 101
$\frac{1}{3}$	4	10	1111 1011 1101
$\frac{1}{3}$	5	12	11111 11011 10101
$\frac{1}{3}$	6	13	101111 110101 111001
$\frac{1}{3}$	7	15	1001111 1010111 1101101
$\frac{1}{3}$	8	16	11101111 10011011 10101001

Source: J. P. Odenwalder, *Error Control Coding Handbook*, Linkabit Corp., San Diego, Calif., July 15, 1976.

Typical values

- Decisions
  - Hard
  - Soft with 3 quantization bits
- Constraint length : $3 \leq K \leq 9$
- Code rate : $r \geq 1/3$
- Maximum path : $h \leq 5K$

Chapter 8

- Reed-Solomon code
  - Cyclic code, non-binary
  - Maximum free distance
  - Very strong against « burst » noise
- Interleaving
- Concatenated codes
  - Two levels of coding
  - Example: audio CD
- Turbo codes
  - Decoding that exploits concatenated codes

Shannon Capacity

Shannon's Law

- The capacity of a channel with signal-to-noise ratio SNR and bandwidth W

$$C = W \log_2 \left(1 + SNR\right)$$

- Ability to send C b/s with as small a P_e as you want…

Alternative form

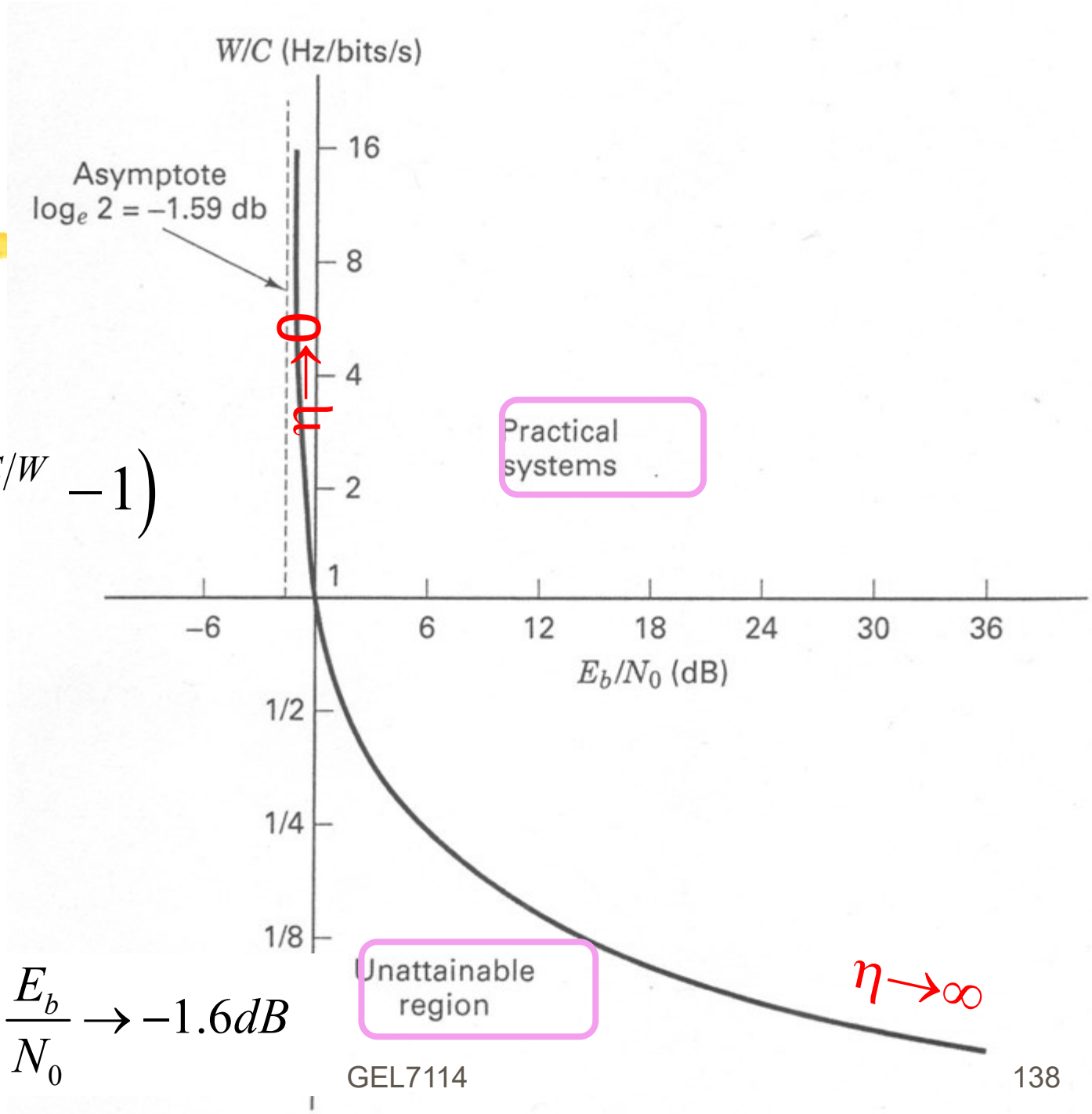
$$\left.\frac{C}{W}\right|_{C=R} = \log_2\left(1+\frac{S}{WN_0}\right)\Bigg|_{C=R} = \log_2\left(1+\frac{S}{CN_0}\frac{C}{W}\right)\Bigg|_{C=R}$$

$$= \log_2\left(1+\frac{S}{RN_0}\frac{C}{W}\right)$$

$$= \log_2\left(1+\frac{ST_b}{N_0}\frac{C}{W}\right) = \log_2\left(1+\frac{E_b}{N_0}\frac{C}{W}\right)$$

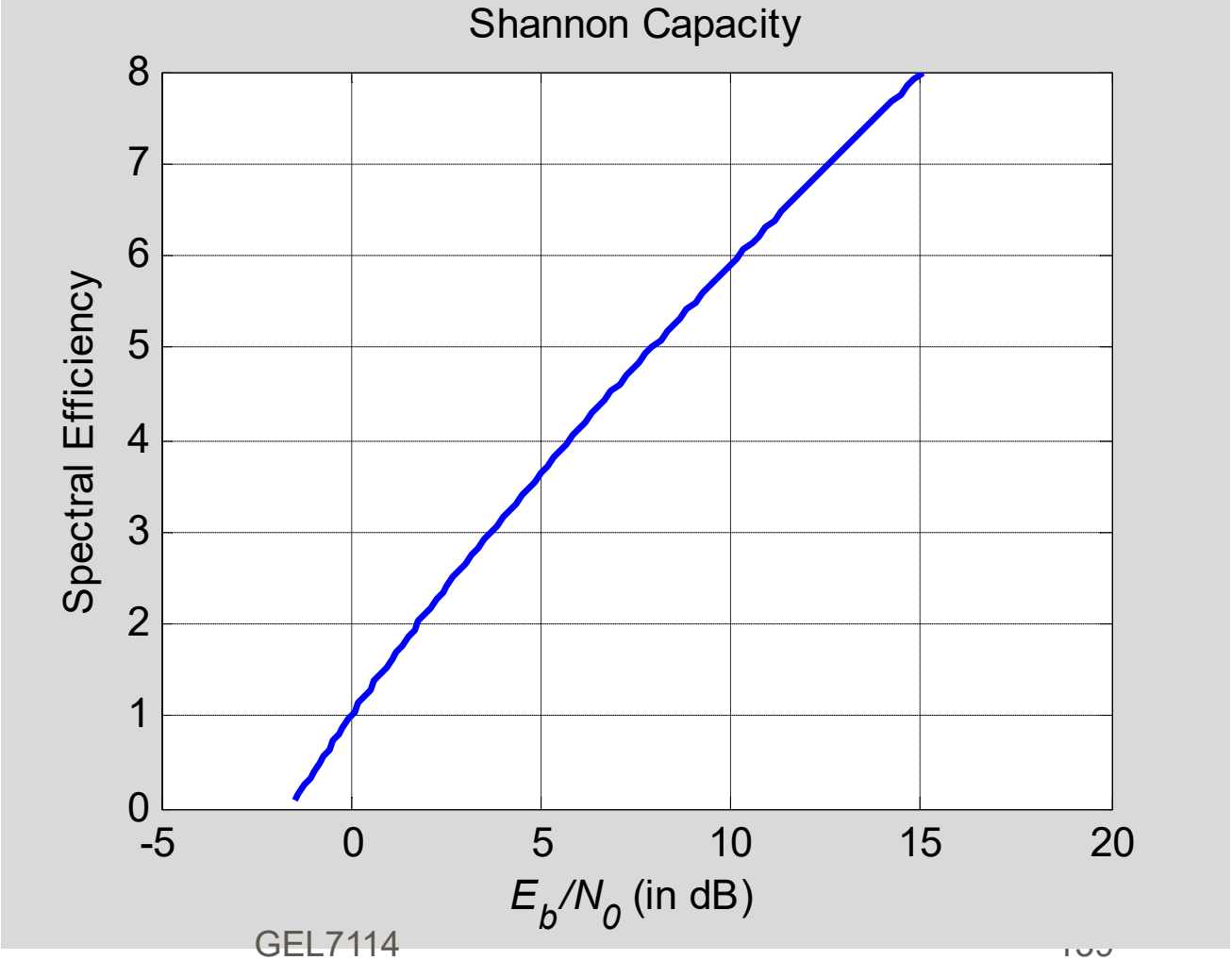
$$\frac{E_b}{N_0} = \frac{W}{C}\left(2^{C/W}-1\right) \qquad \frac{C}{W}\to 0 \quad \Rightarrow \quad \frac{E_b}{N_0}\to -1.6dB$$

$$\frac{E_b}{N_0} = \frac{W}{C}\left(2^{C/W} - 1\right)$$



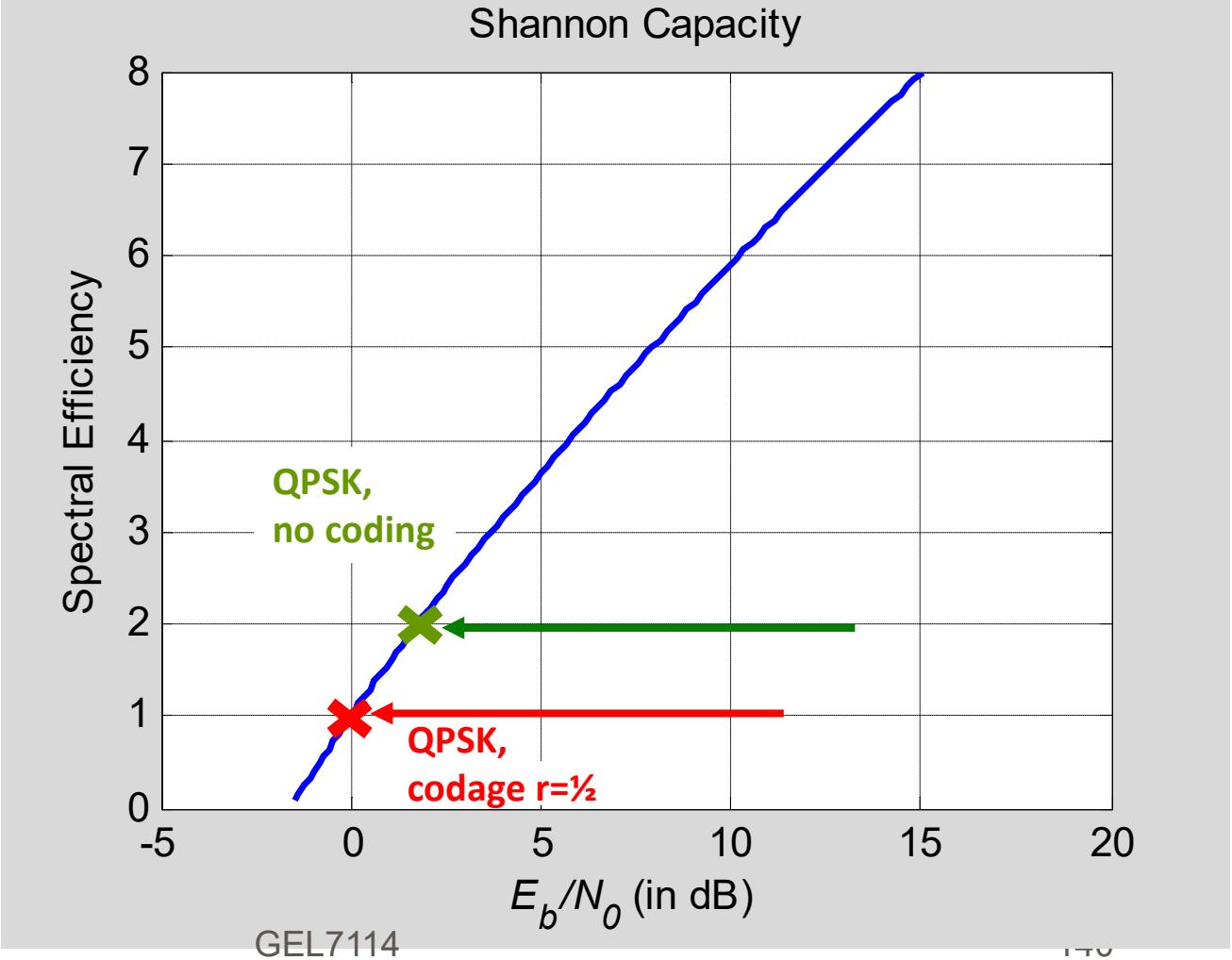
$$\frac{C}{W} \to 0 \quad \Rightarrow \quad \frac{E_b}{N_0} \to -1.6dB$$

Linear scale



$$\frac{E_b}{N_0} = \frac{W}{C}\left(2^{C/W} - 1\right)$$

Exemple QPSK



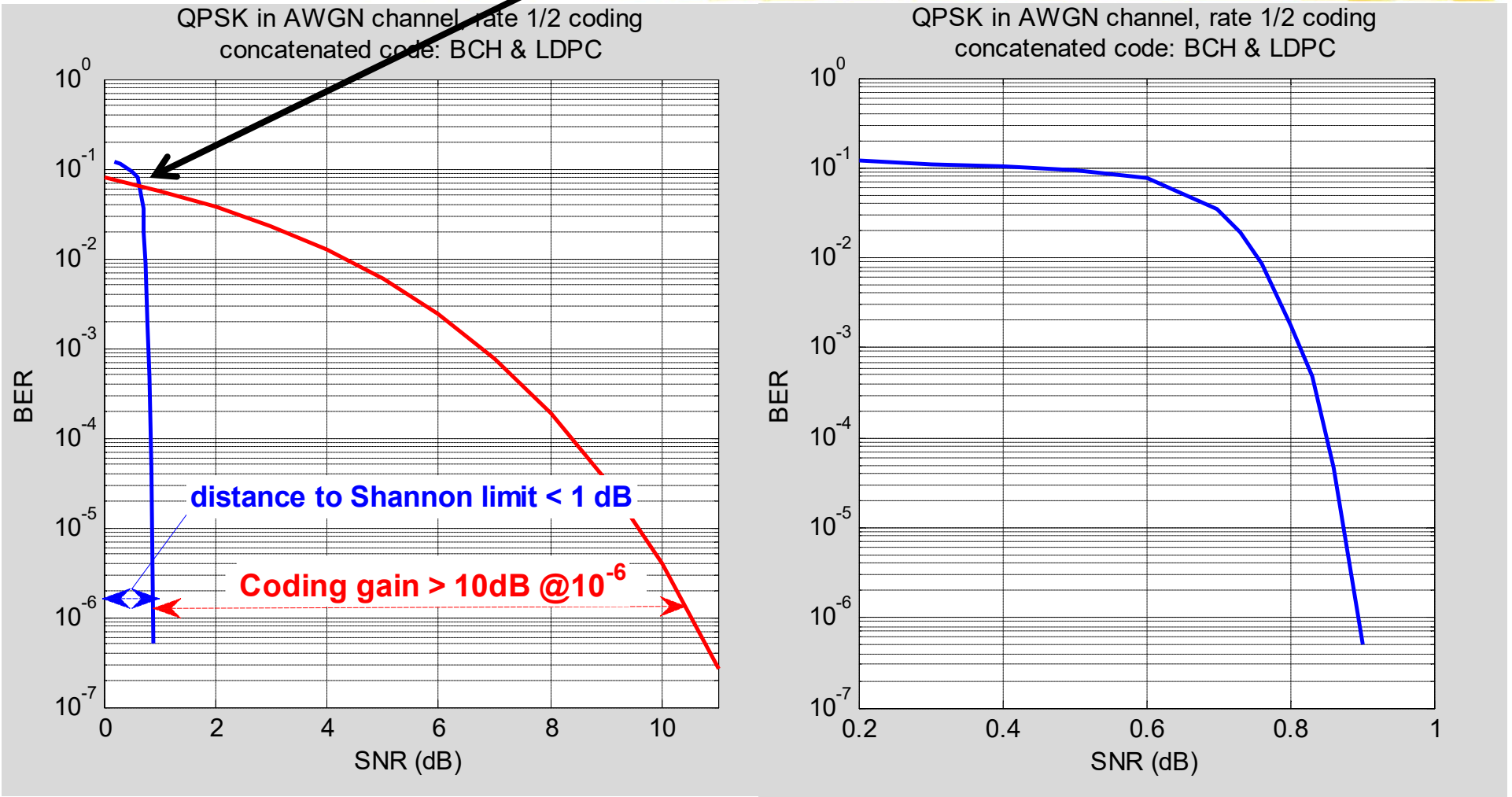
$$\frac{E_b}{N_0} = \frac{W}{C}\left(2^{C/W} - 1\right)$$

Exemple –
Digital Video Broadcast

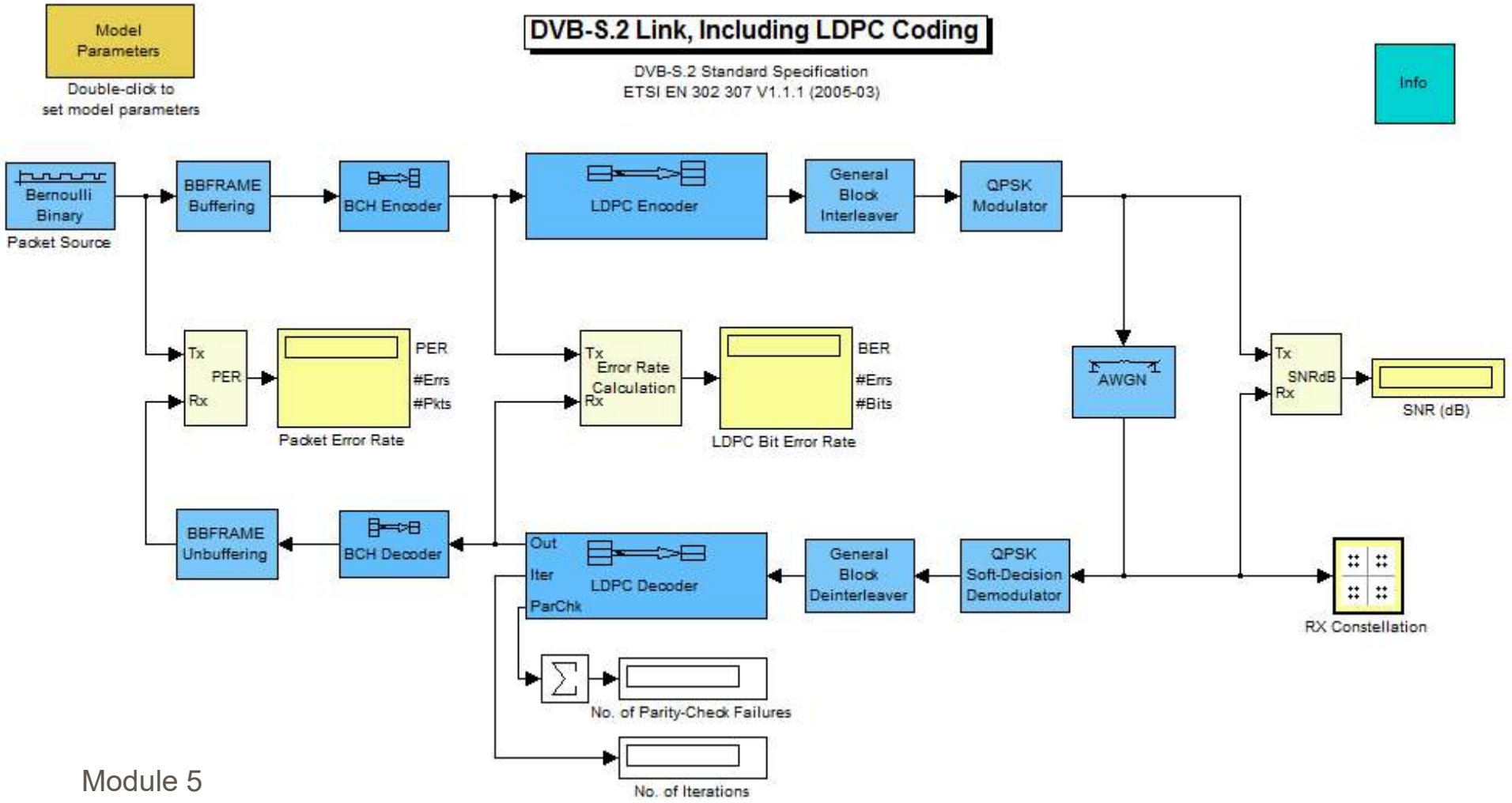
DVB-S2

- Satellite video
- Concatenated codes
  - BCH (32400,32208) & LDPC (64800,32400)
  - Interleaving
- Example Matlab demo
  - QPSK & coding rate ½
- Close to Shannon limit

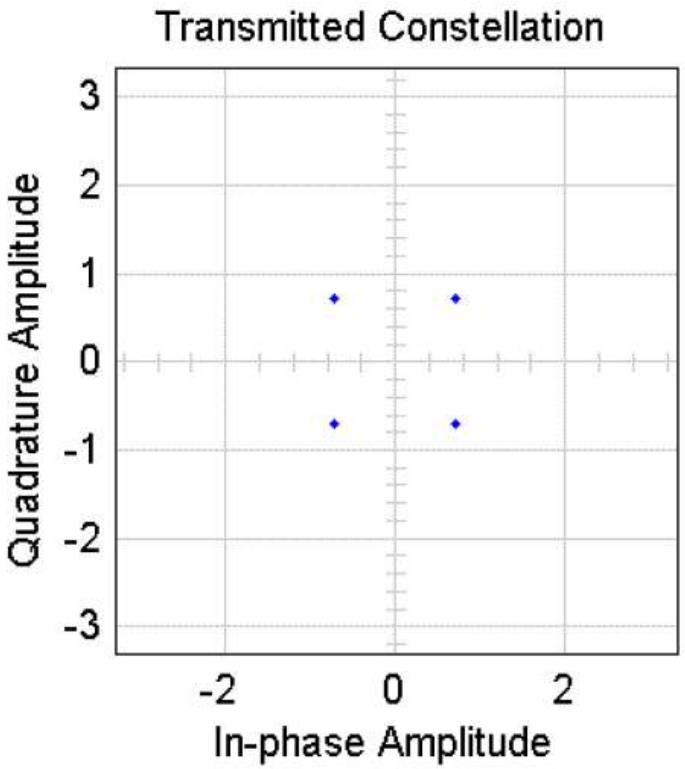
FEC limit very high: .05

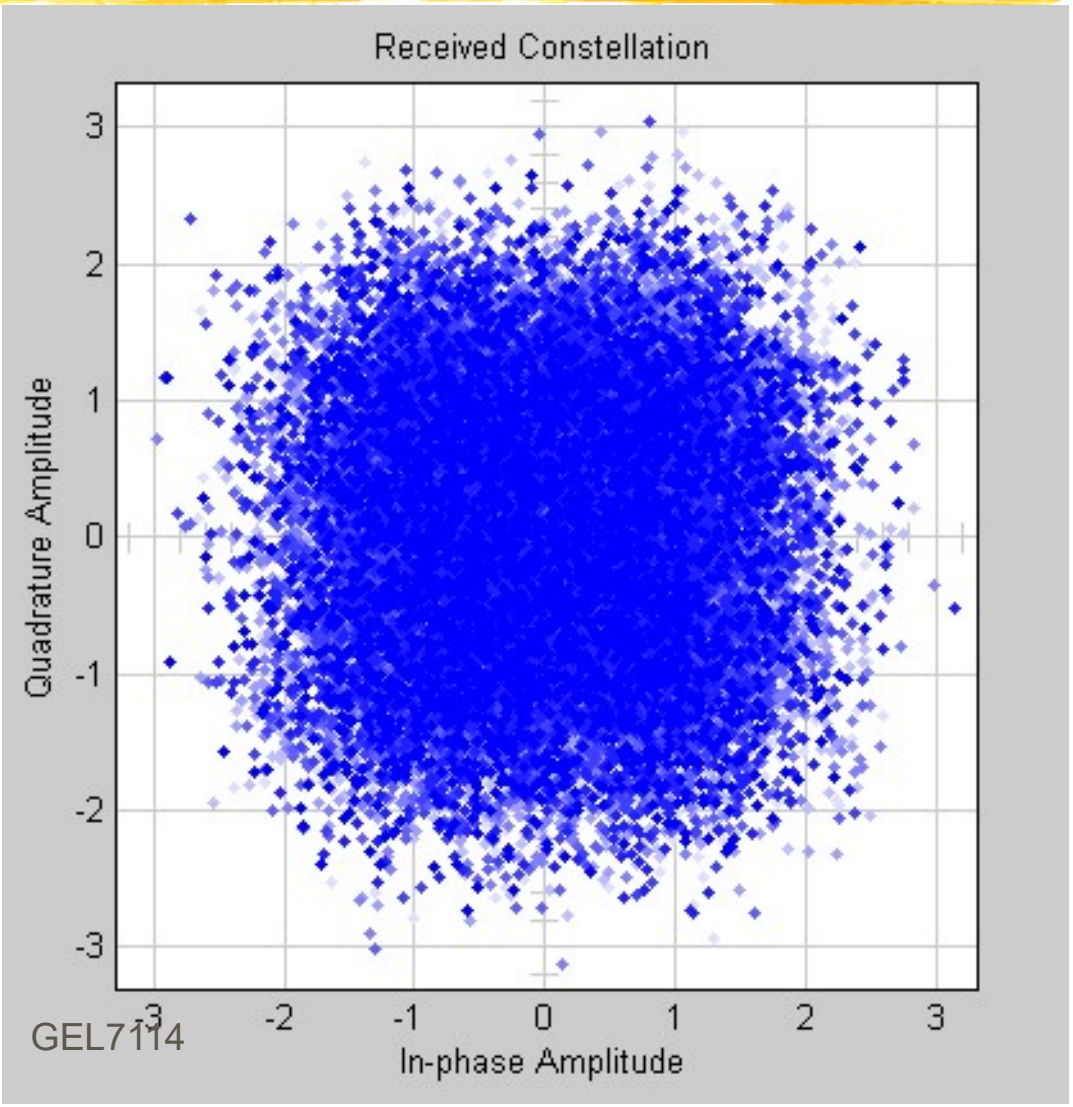


Matlab demo

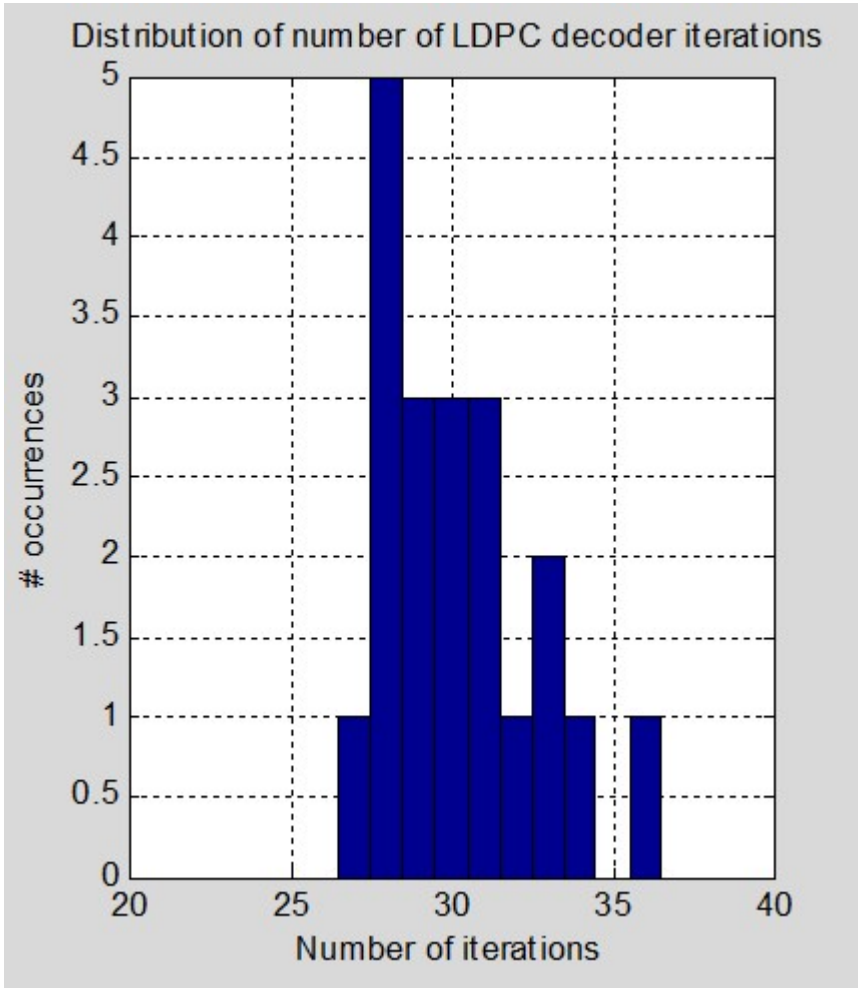

Model Parameters
Double-click to set model parameters
DVB-S.2 Link, Including LDPC Coding
DVB-S.2 Standard Specification
ETSI EN 302 307 V1.1.1 (2005-03)
Info
Bernoulli Binary
Packet Source
BBFRAME Buffering
BCH Encoder
LDPC Encoder
General Block Interleaver
QPSK Modulator
Tx
PER
Rx
PER
#Errs
#Pkts
Packet Error Rate
Tx
Error Rate Calculation
Rx
BER
#Errs
#Bits
LDPC Bit Error Rate
AWGN
Tx
SNRdB
Rx
SNR (dB)
BBFRAME Unbuffering
BCH Decoder
Out
Iter
ParChk
LDPC Decoder
General Block Deinterleaver
QPSK Soft-Decision Demodulator
RX Constellation
No. of Parity-Check Failures
No. of Iterations

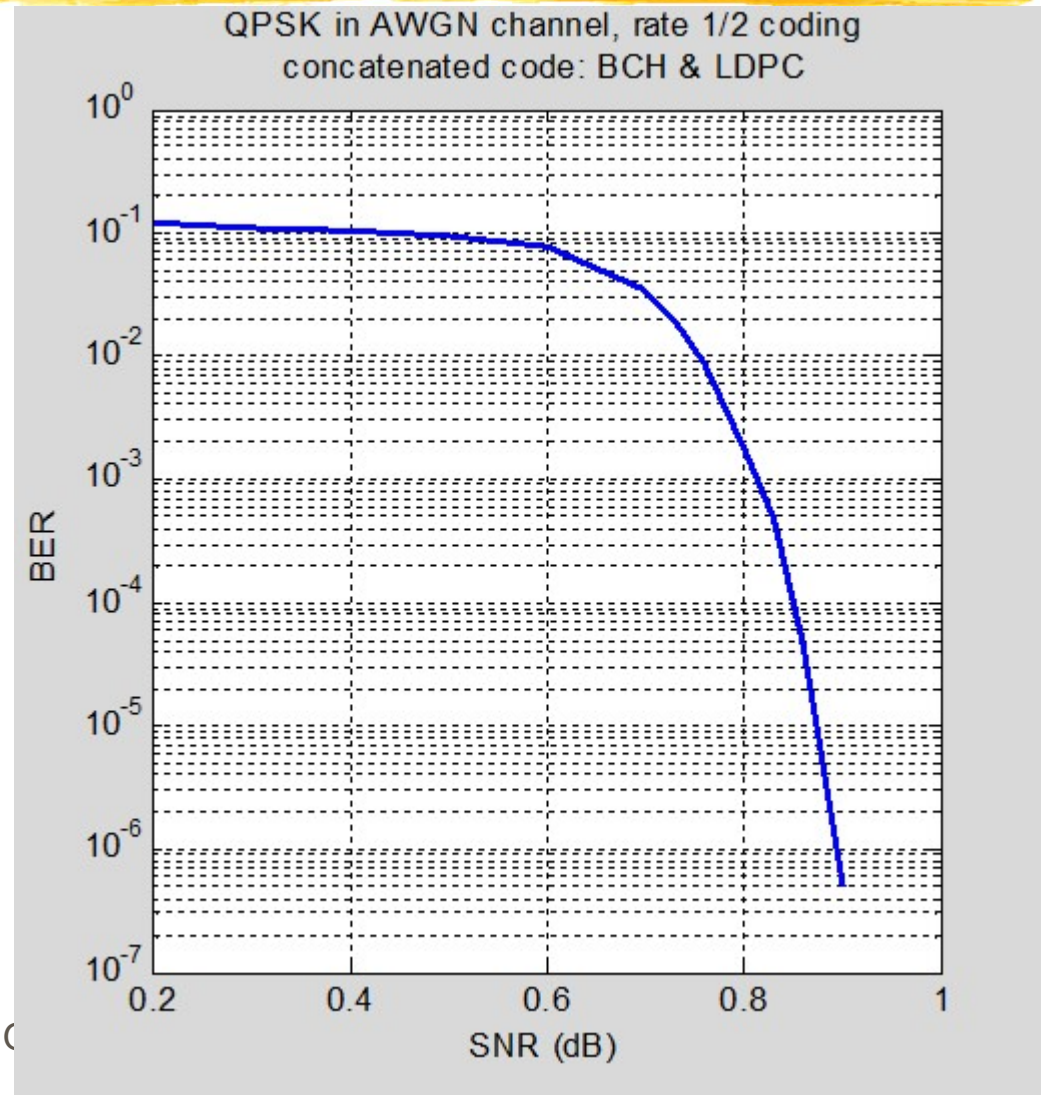
Very low SNR


Transmitted Constellation
Quadrature Amplitude
3
2
1
0
-1
-2
-3
-2
0
2
In-phase Amplitude


Received Constellation
Quadrature Amplitude
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
In-phase Amplitude

DEMO MATLAB

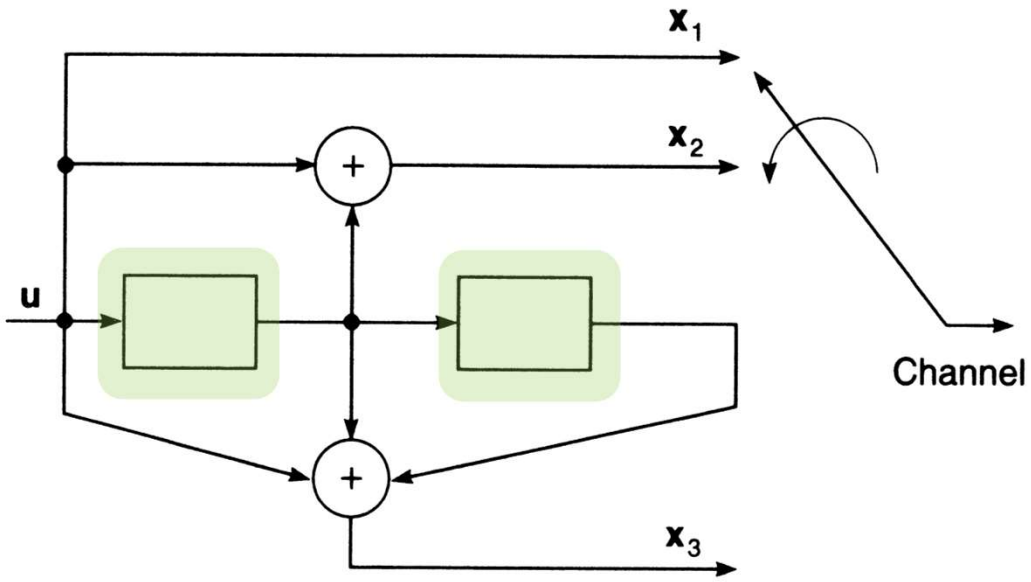

Distribution of number of LDPC decoder iterations
occurrences
Number of iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
20
25
30
35
40


QPSK in AWGN channel, rate 1/2 coding
concatenated code: BCH & LDPC
BER
SNR (dB)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0.2
0.4
0.6
0.8
1

Encoding

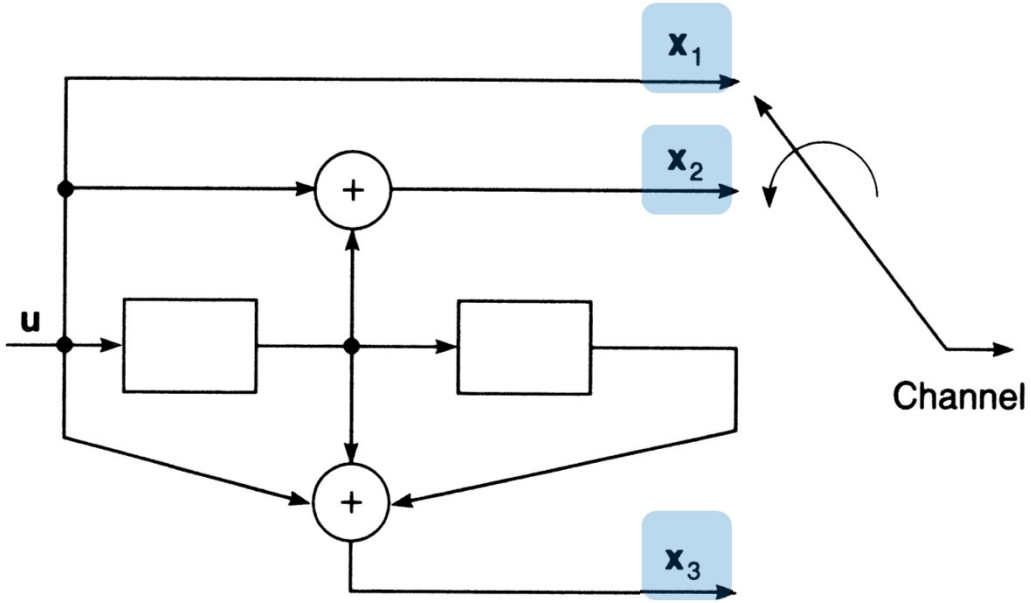
Example r=1/3, K=3

Encoder



K=3

Encoder



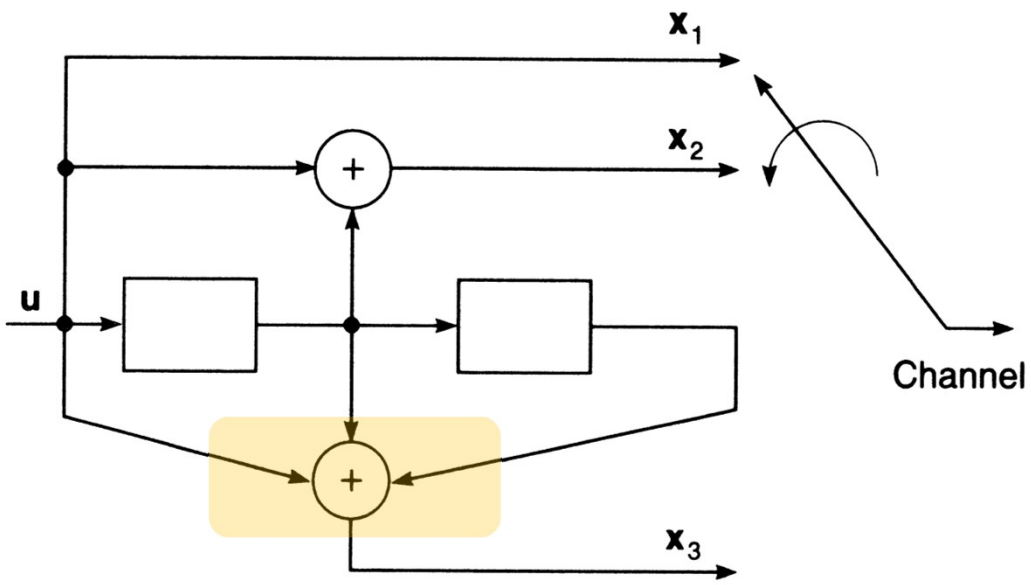
r=1/3

$g_1=[1\ 0\ 0]=4$ hex

$g_2=[1\ 1\ 0]=6$ hex

$g_3=[1\ 1\ 1]=5$ hex

Encoder

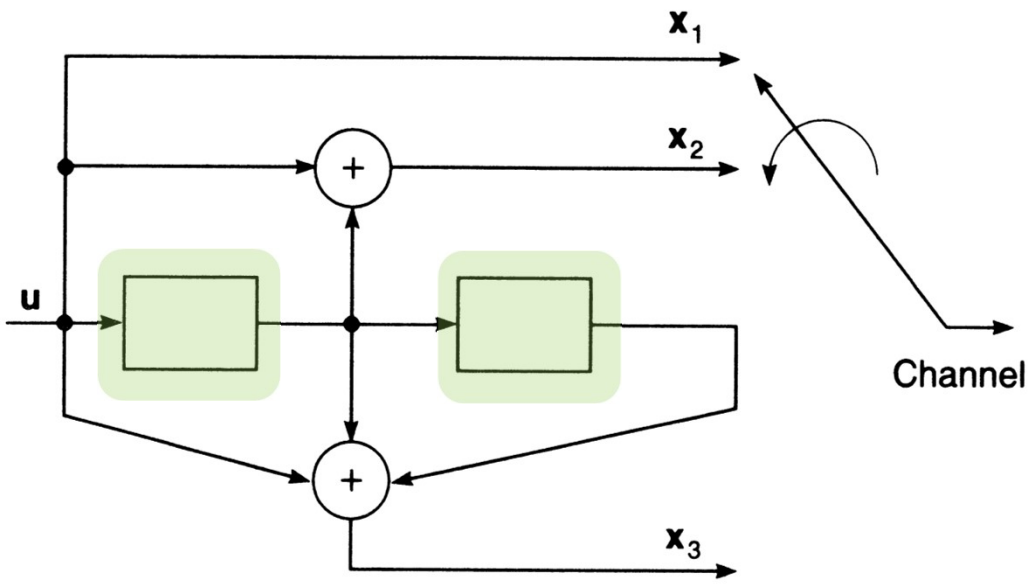


g_1=[1 0 0]=4 hex

g_2=[1 1 0]=6 hex

g_3=[1 1 1]=7 hex

Encoder



00

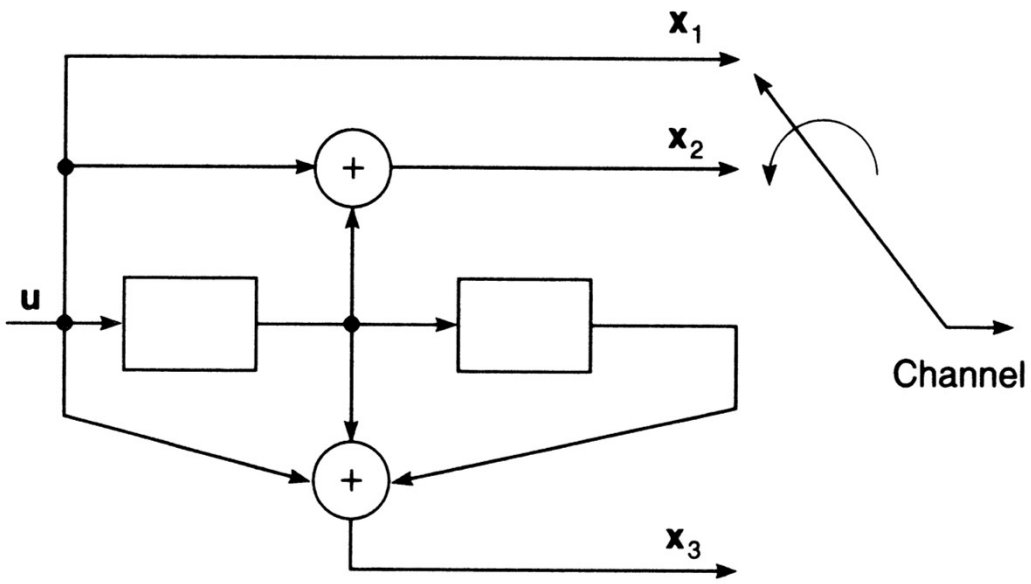
01

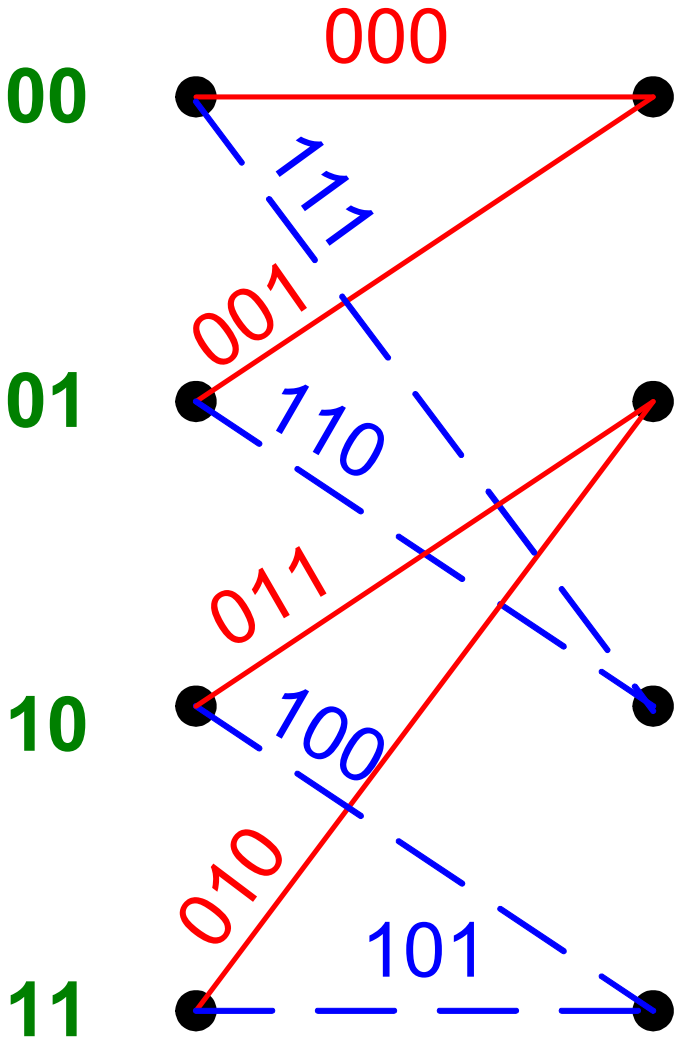
10

11

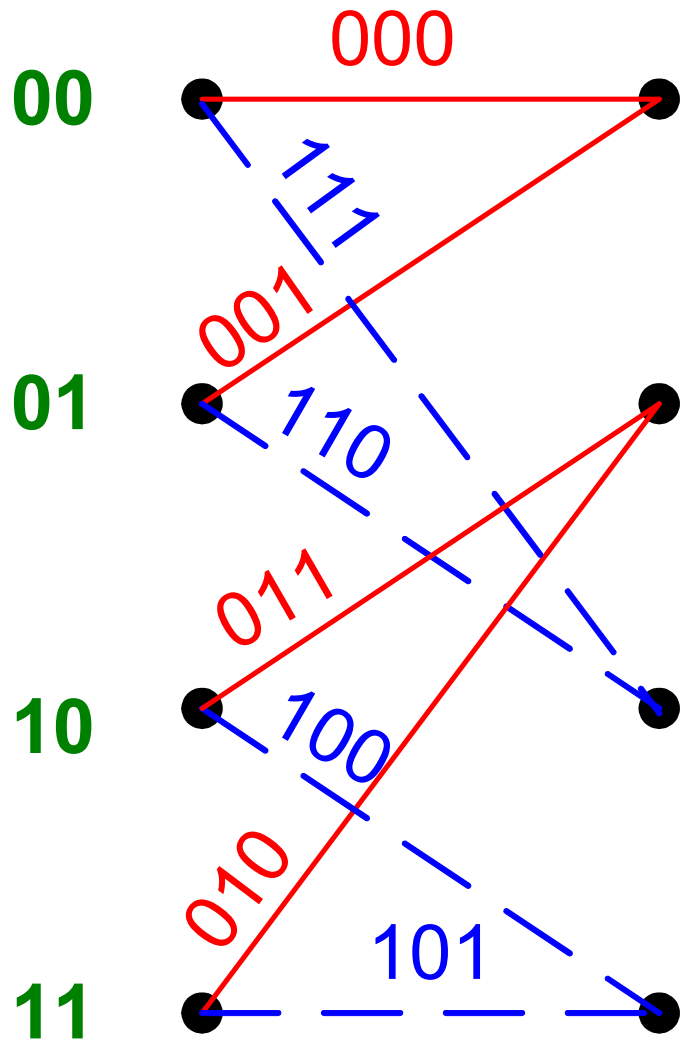
K=3

Encoder

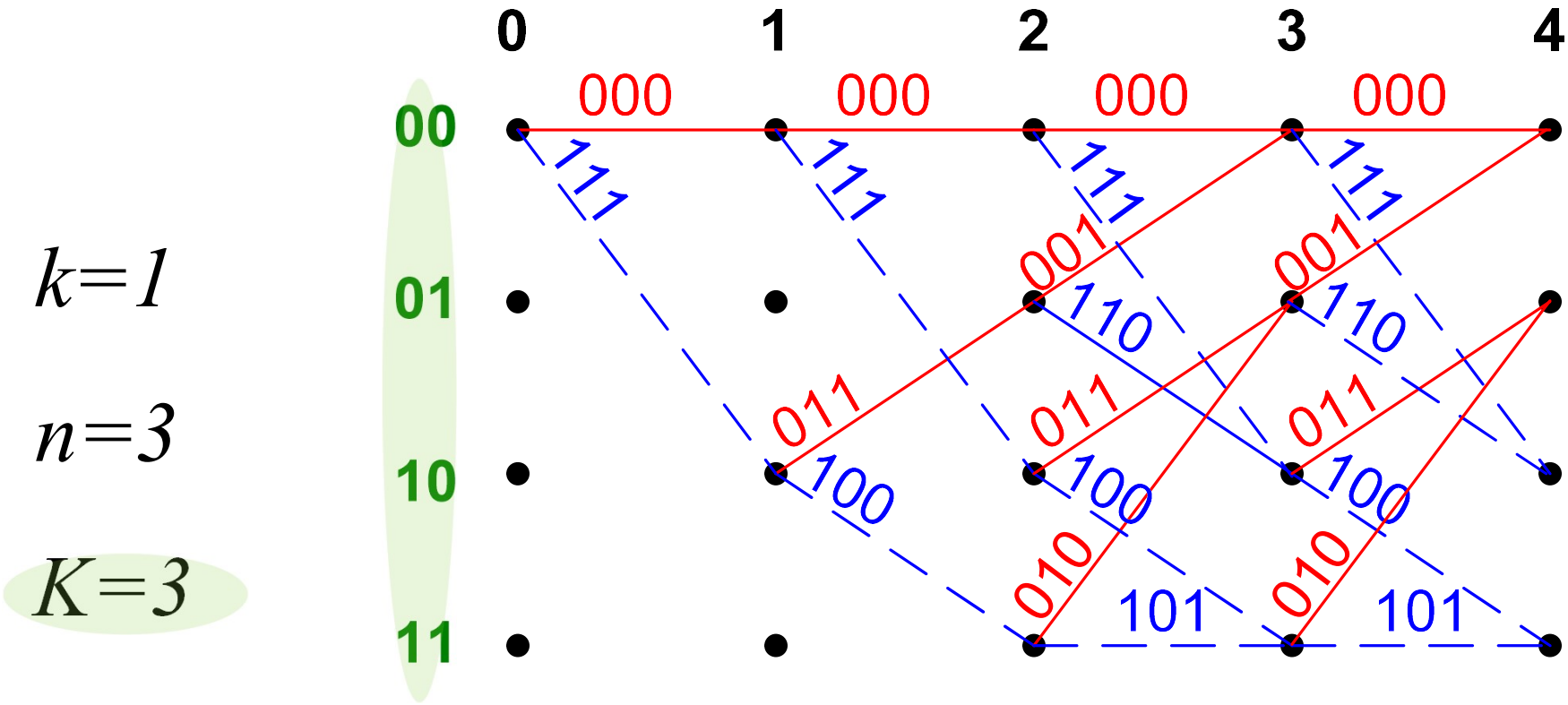




Trellis encoder



All possible transitions

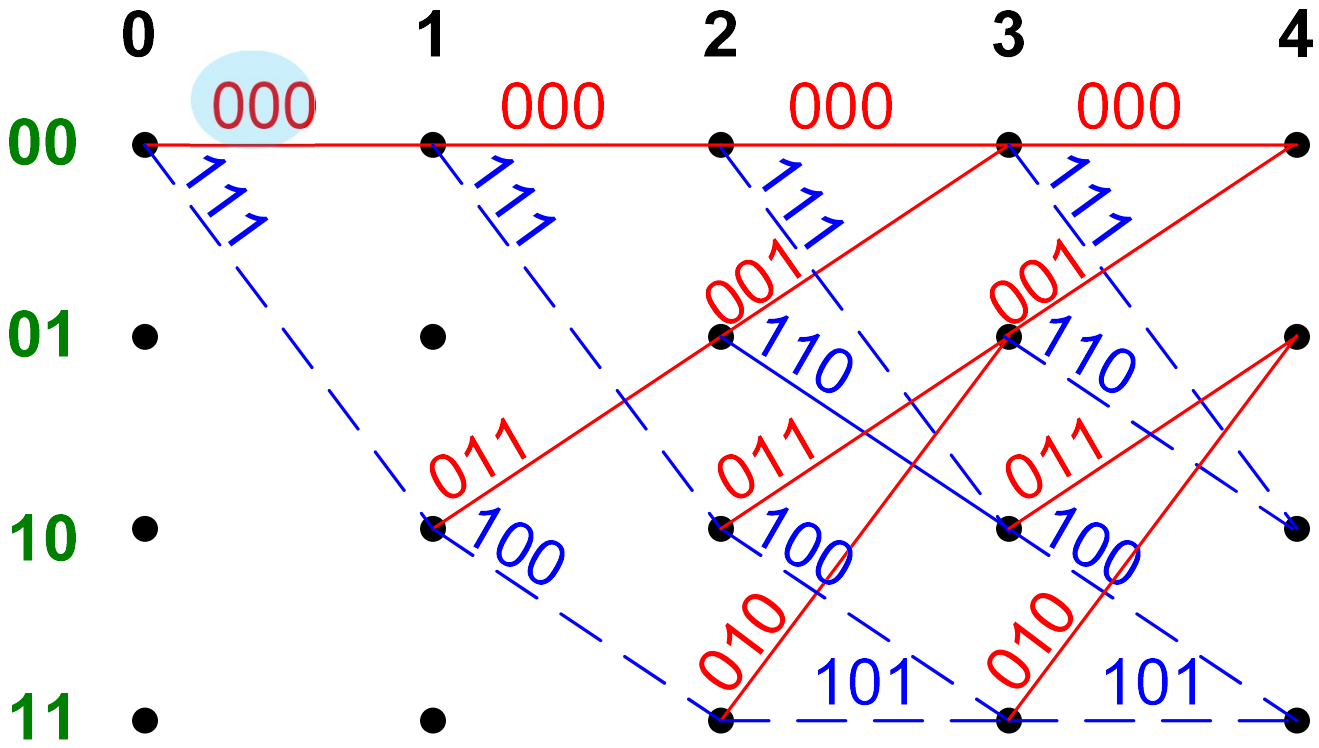


All possible transitions

$k=1$

$n=3$

$K=3$

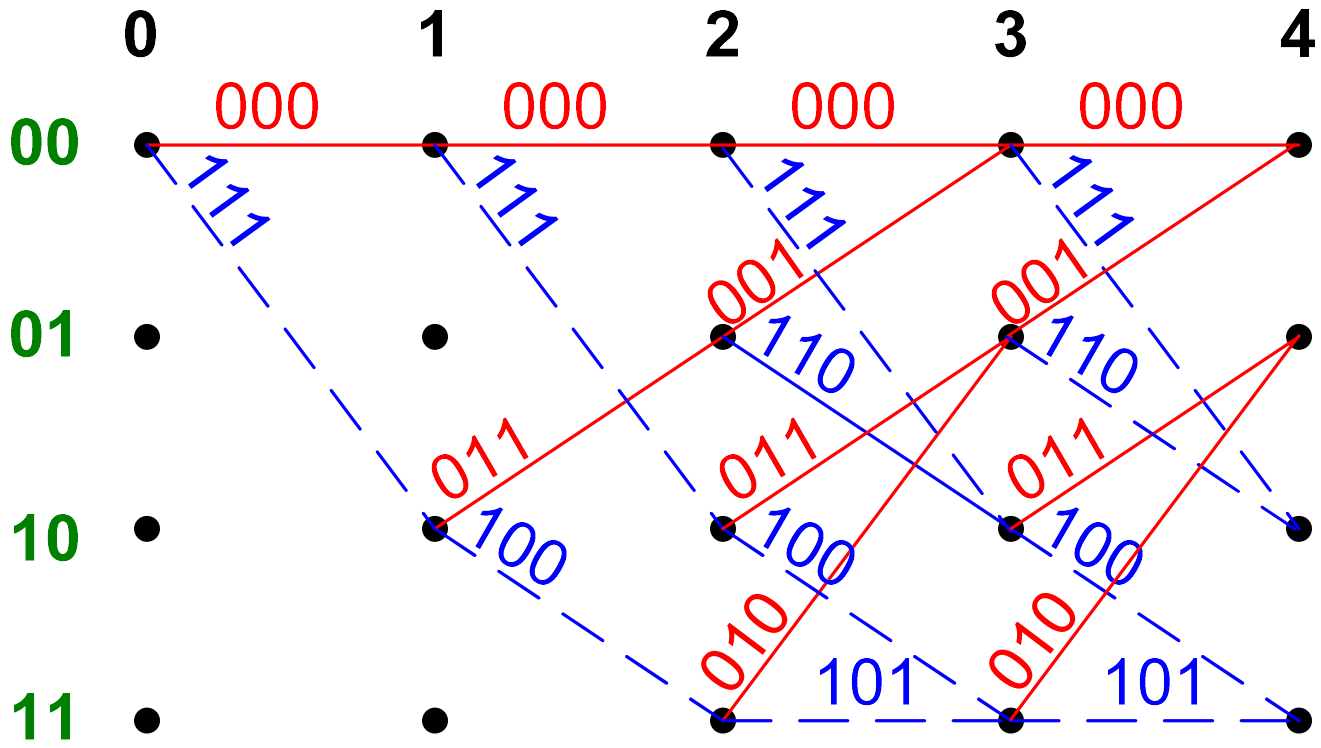


Encoder

$k=1$

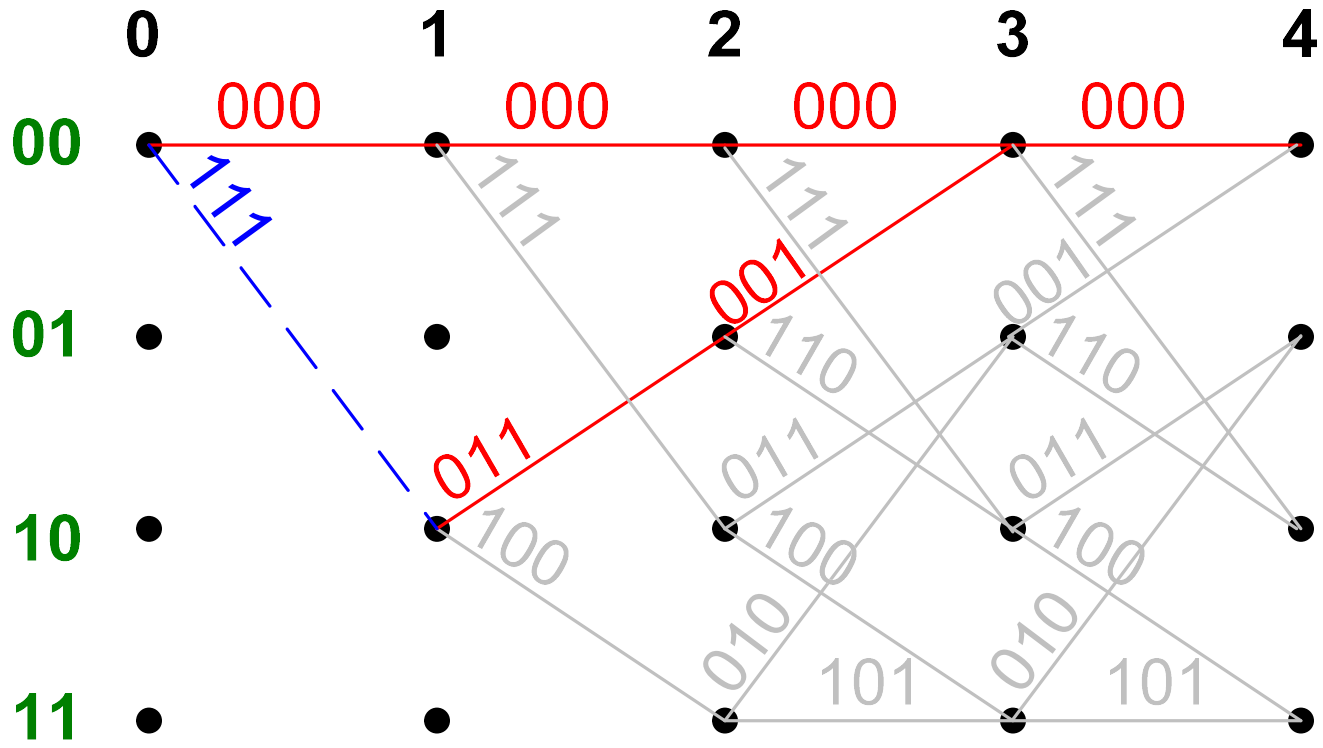
$n=3$

$K=3$

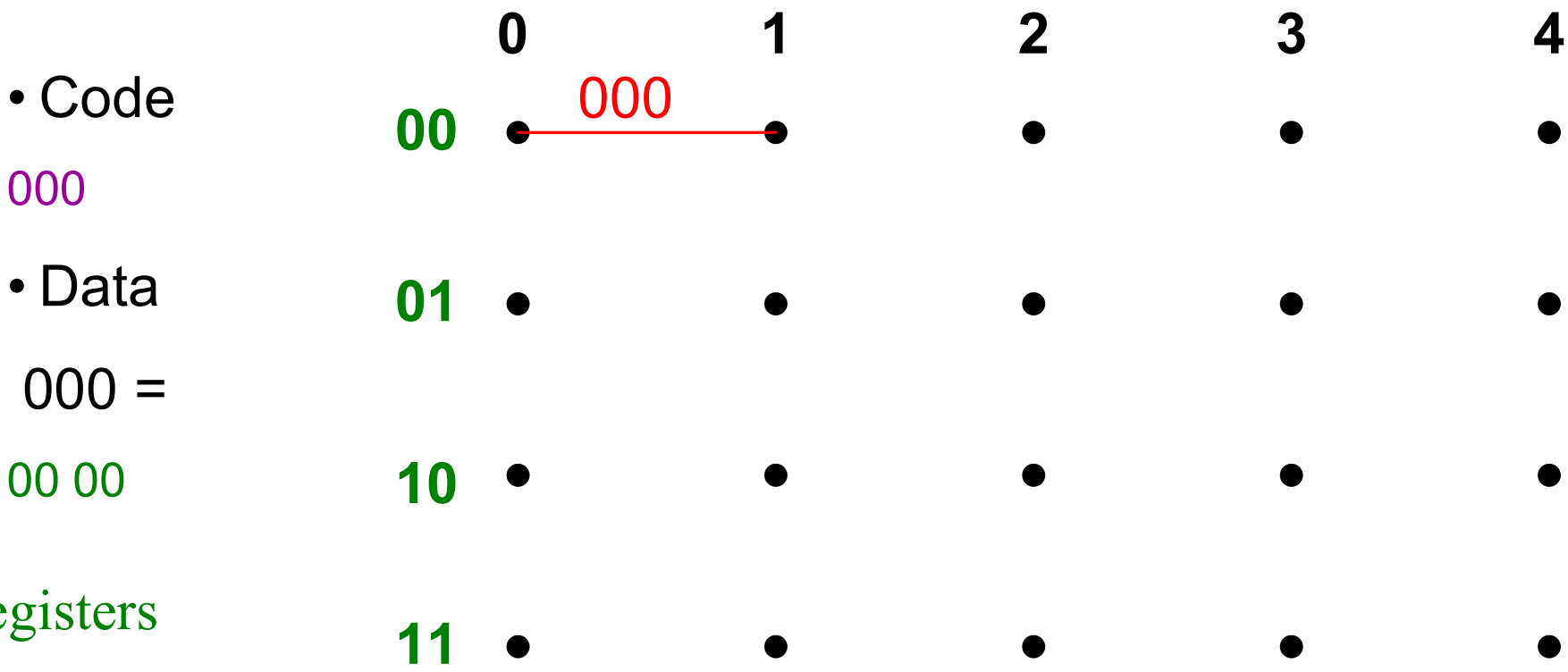


Minimum distance

- Distance to zero that is minimal
- d=6



Example - *Encoding*



Exemple - *encodage*

- Code

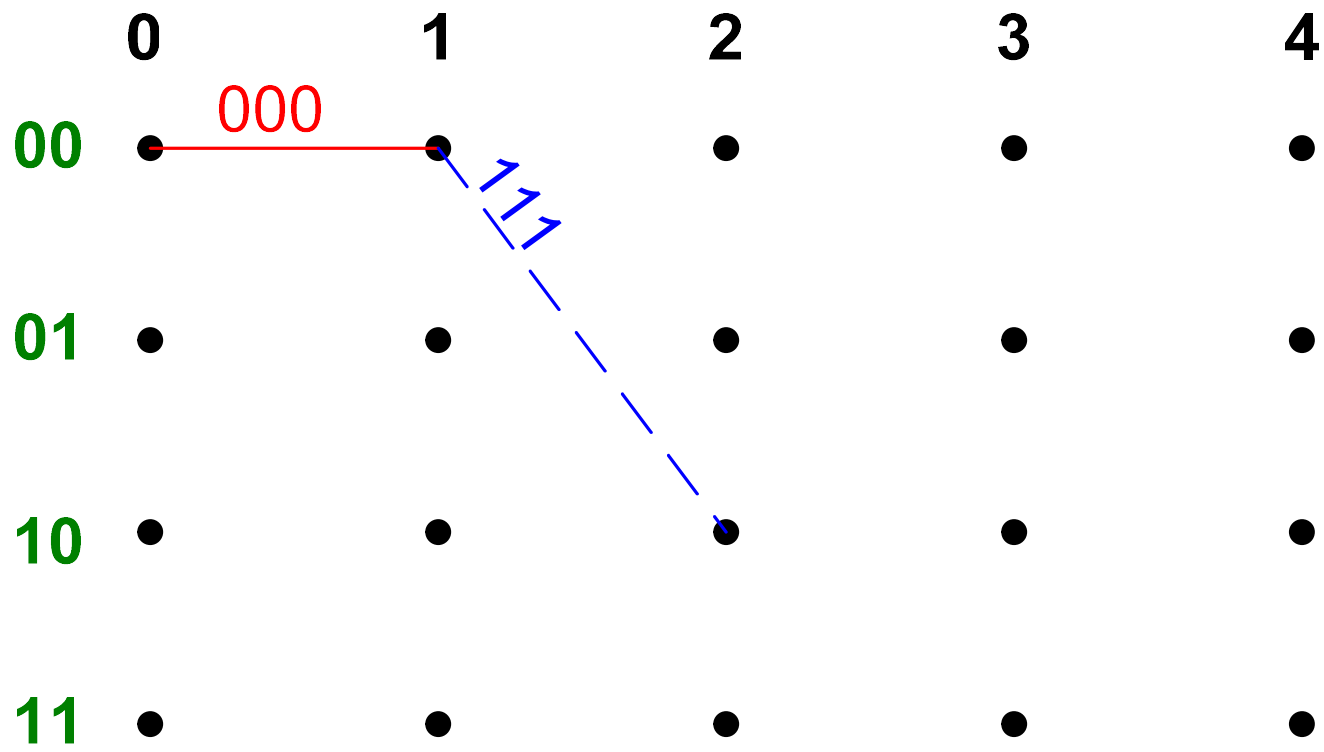
000111

- Data

(00)01 =

00 00 10

registers



Exemple - *encodage*

- Code

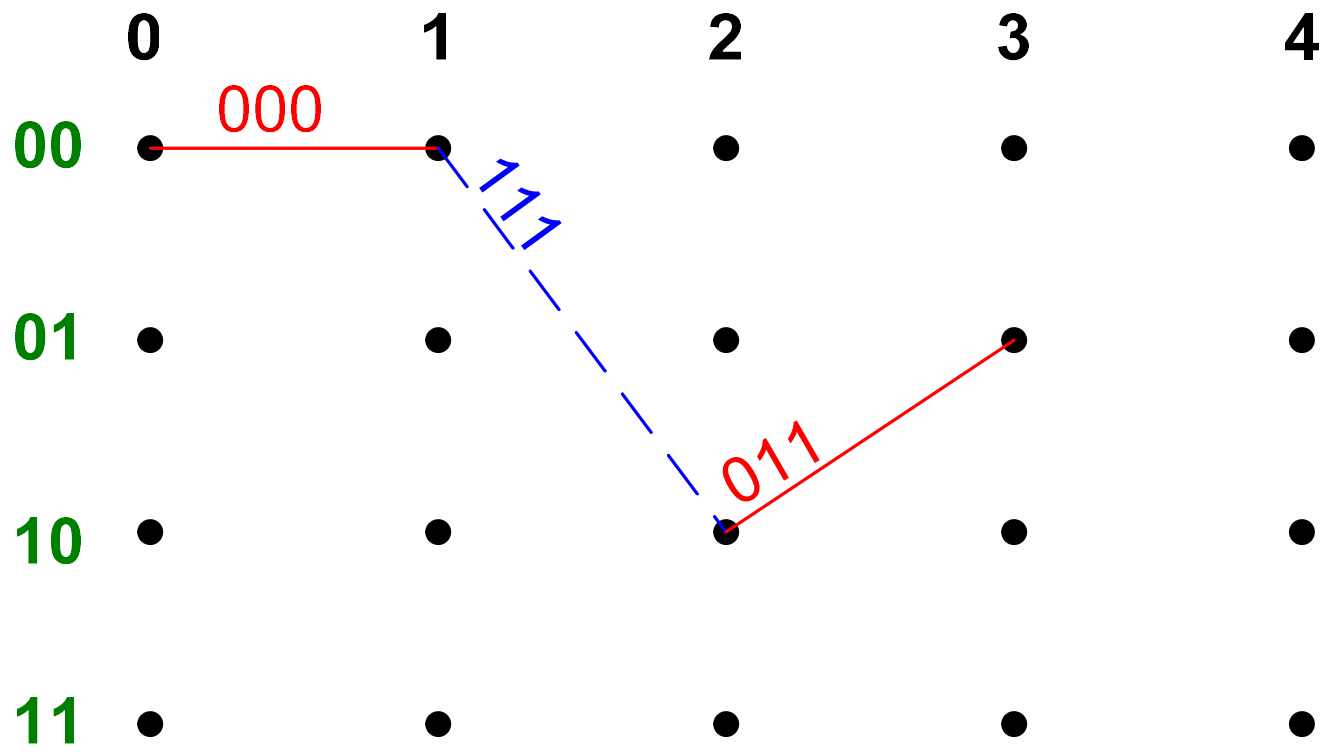
000111011

- Data

(00)010 =

00 00 10 01

registers



Exemple - *encodage*

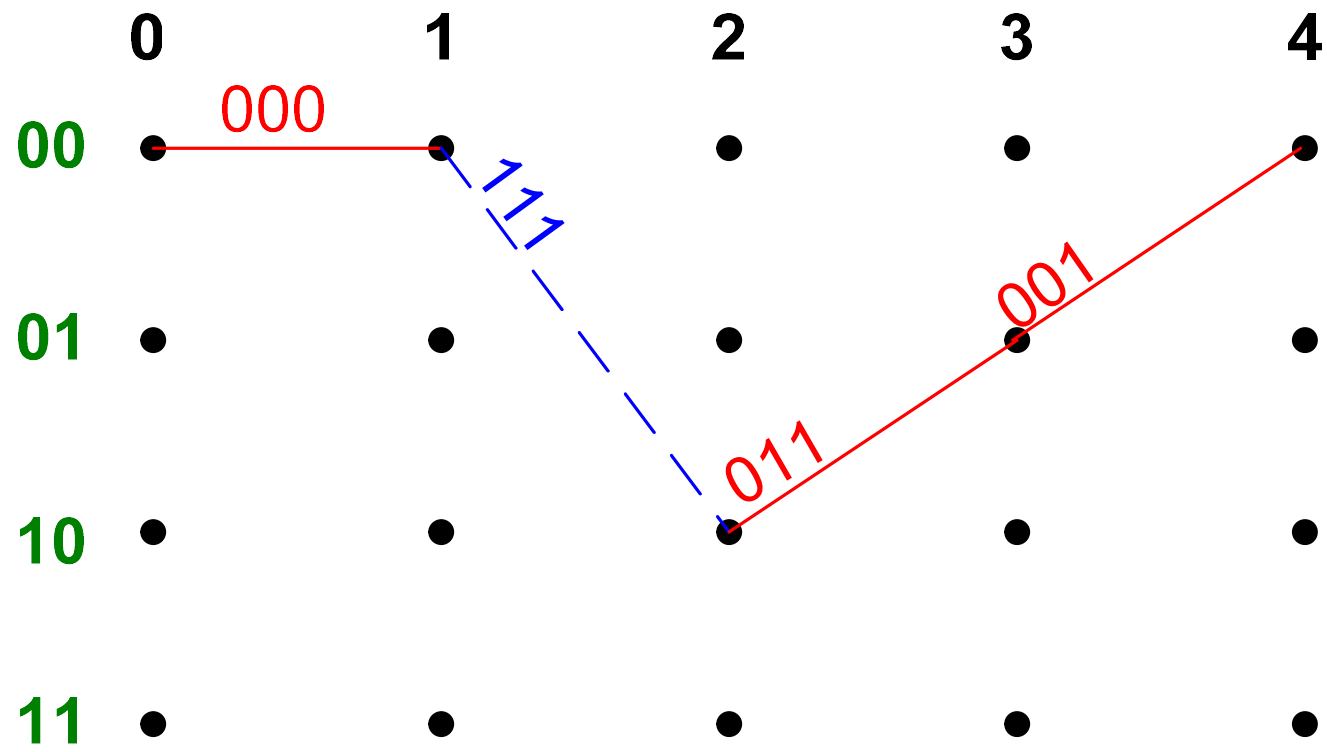
- Code

000111011001

- Data

(00)0100 =

00 00 10 01 00



Hamming Weight

- Code

111-100-010-110

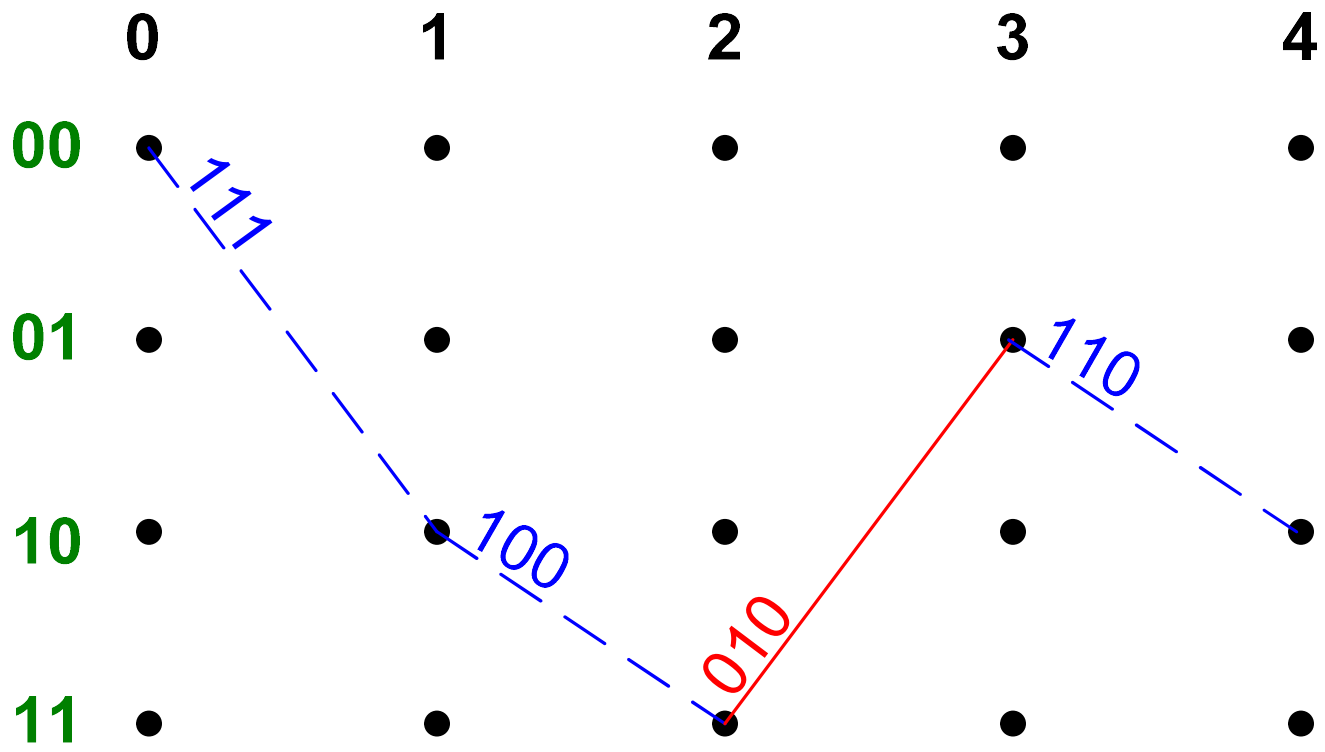
- Data

001101

- Weight

= # of ones

= 7



Hamming Weight

- Code

111100010110

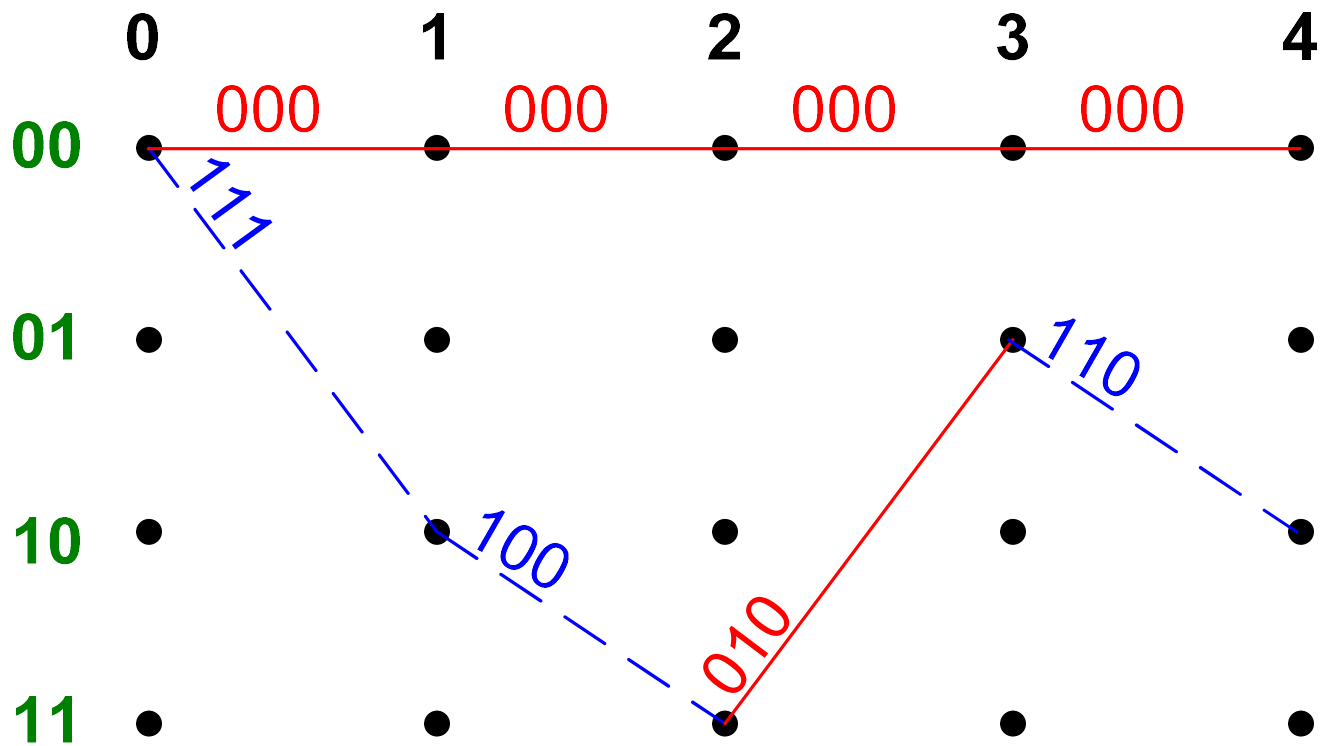
- Weight

= # of ones

= 7

- Distance to zero

-



Decoding *Viterbi algorithm*

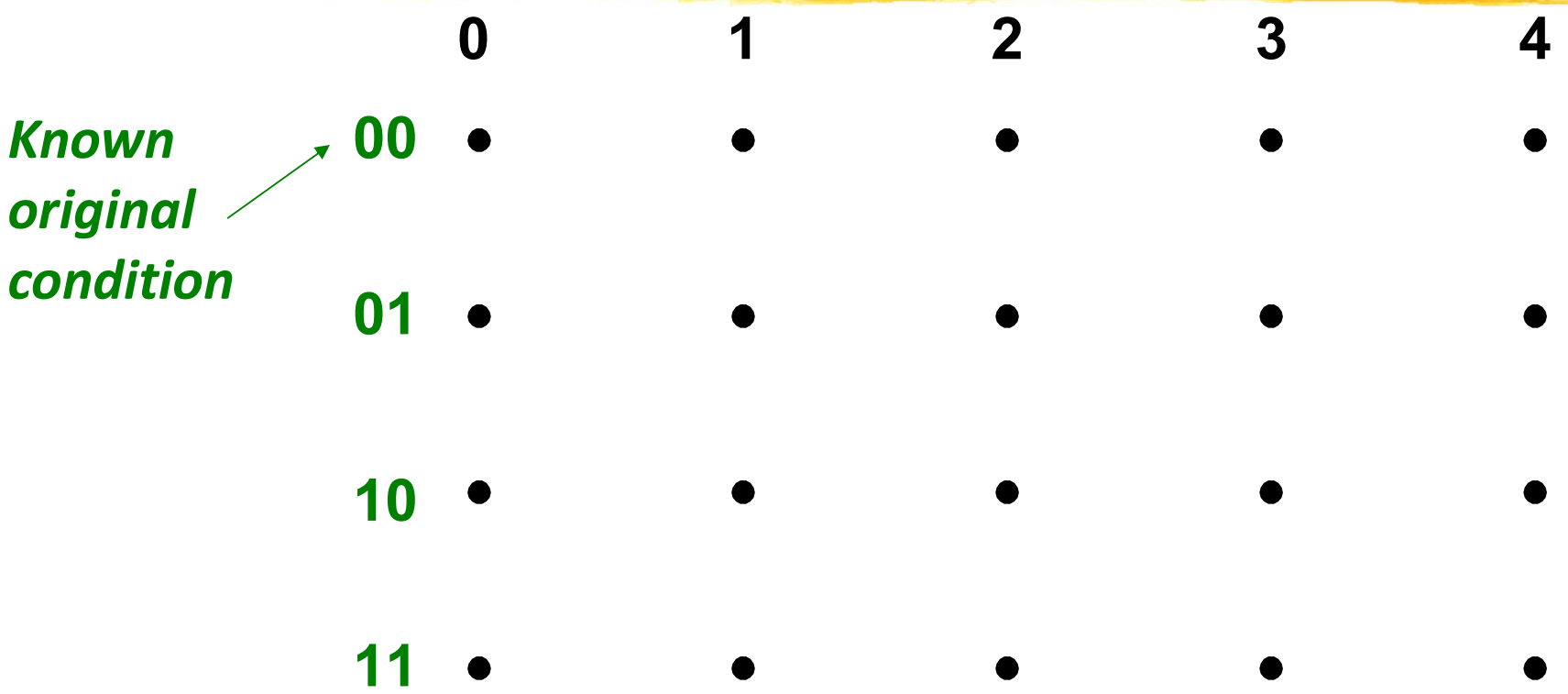
Exemple r=1/3, K=3

Two transmission errors

- Data (00)0100
- Sent codes 000-111-011-001
- Received codes 001-110-011-001
- Data (00)0100

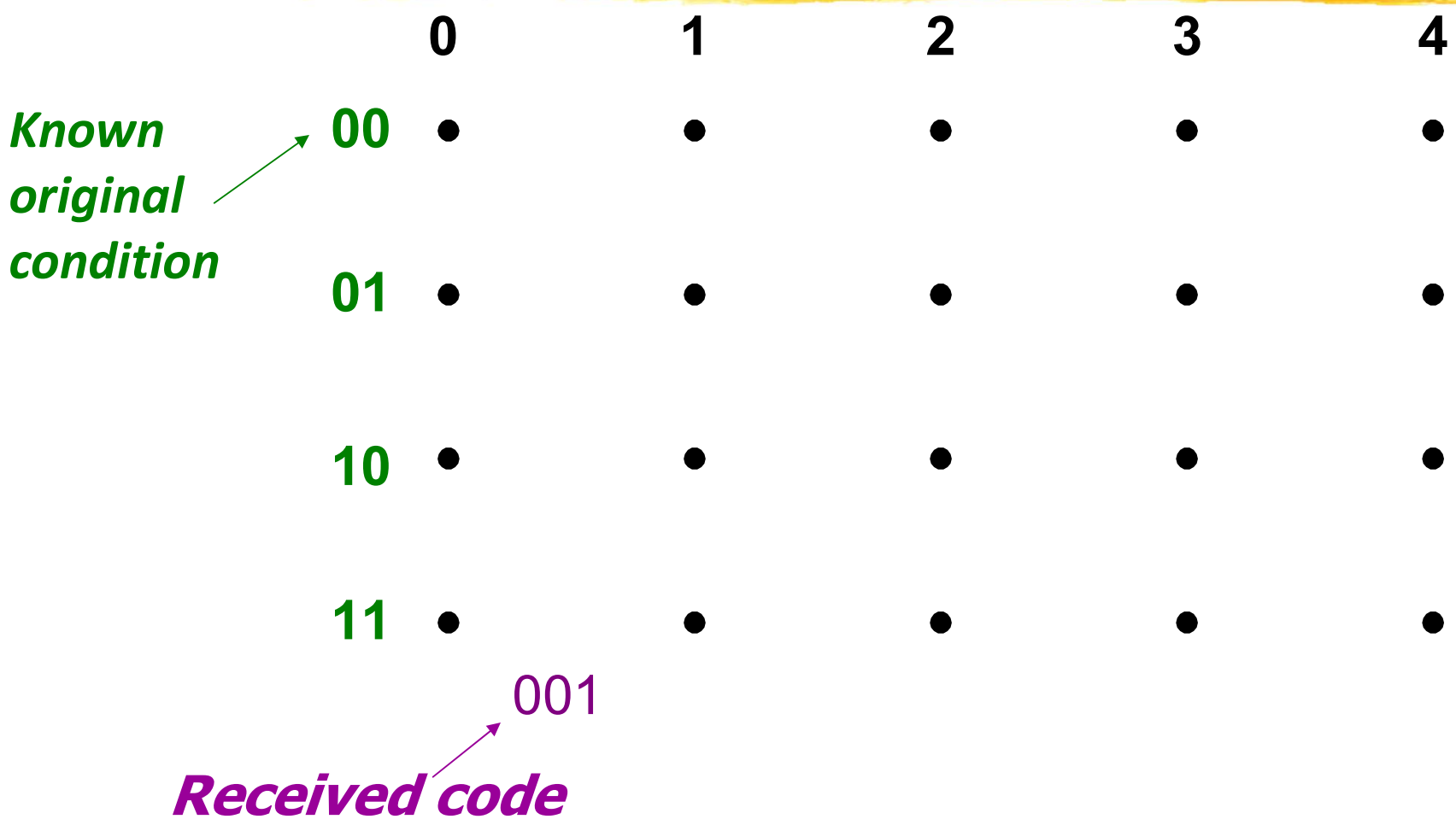
Sent codes 000-111-011-001
Received codes 001-110-011-001

Step one



Sent codes 000-111-011-001
Received codes 001-110-011-001

Step one

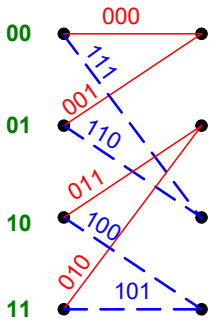


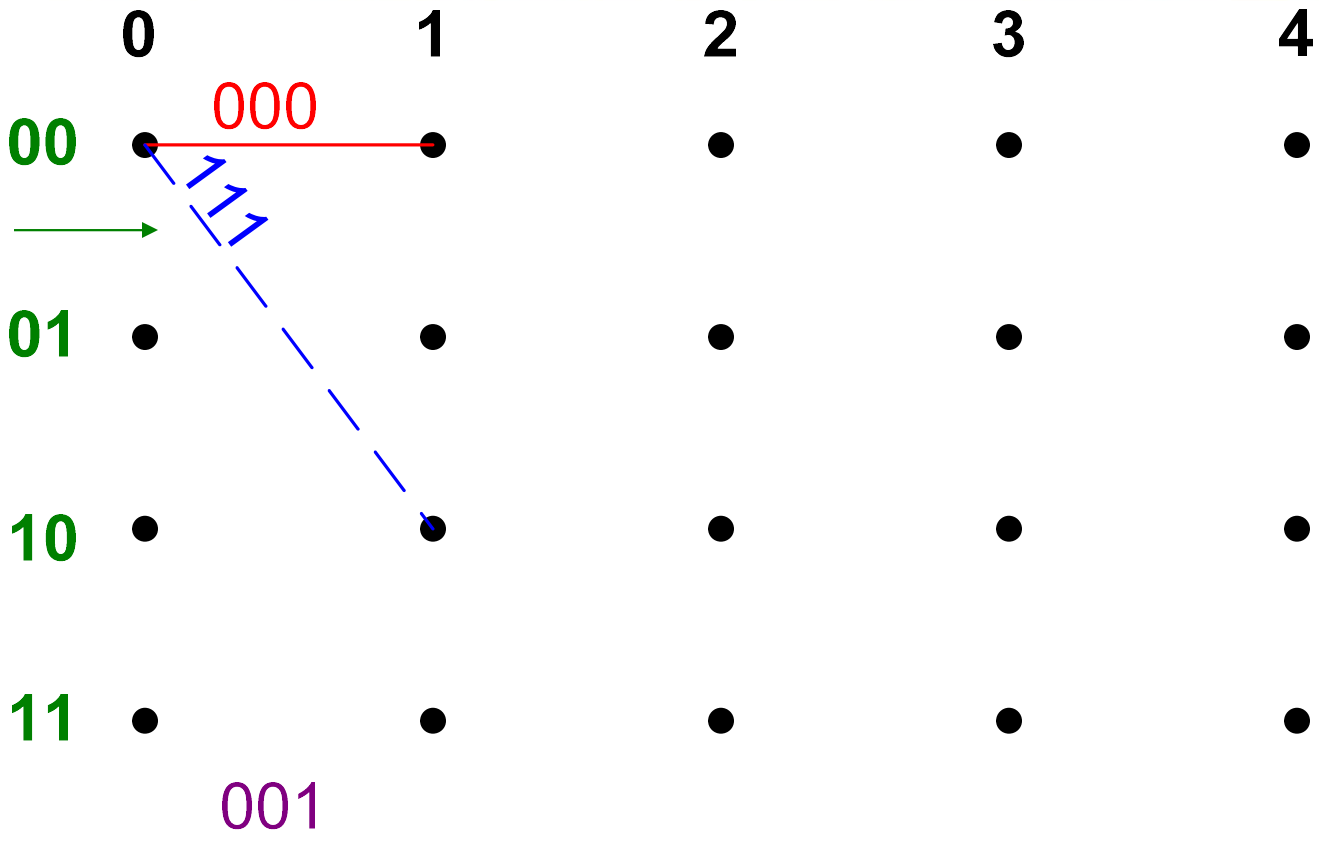
Step one

Sent codes 000-111-011-001

Received codes 001-110-011-001

Formulate possible transitions

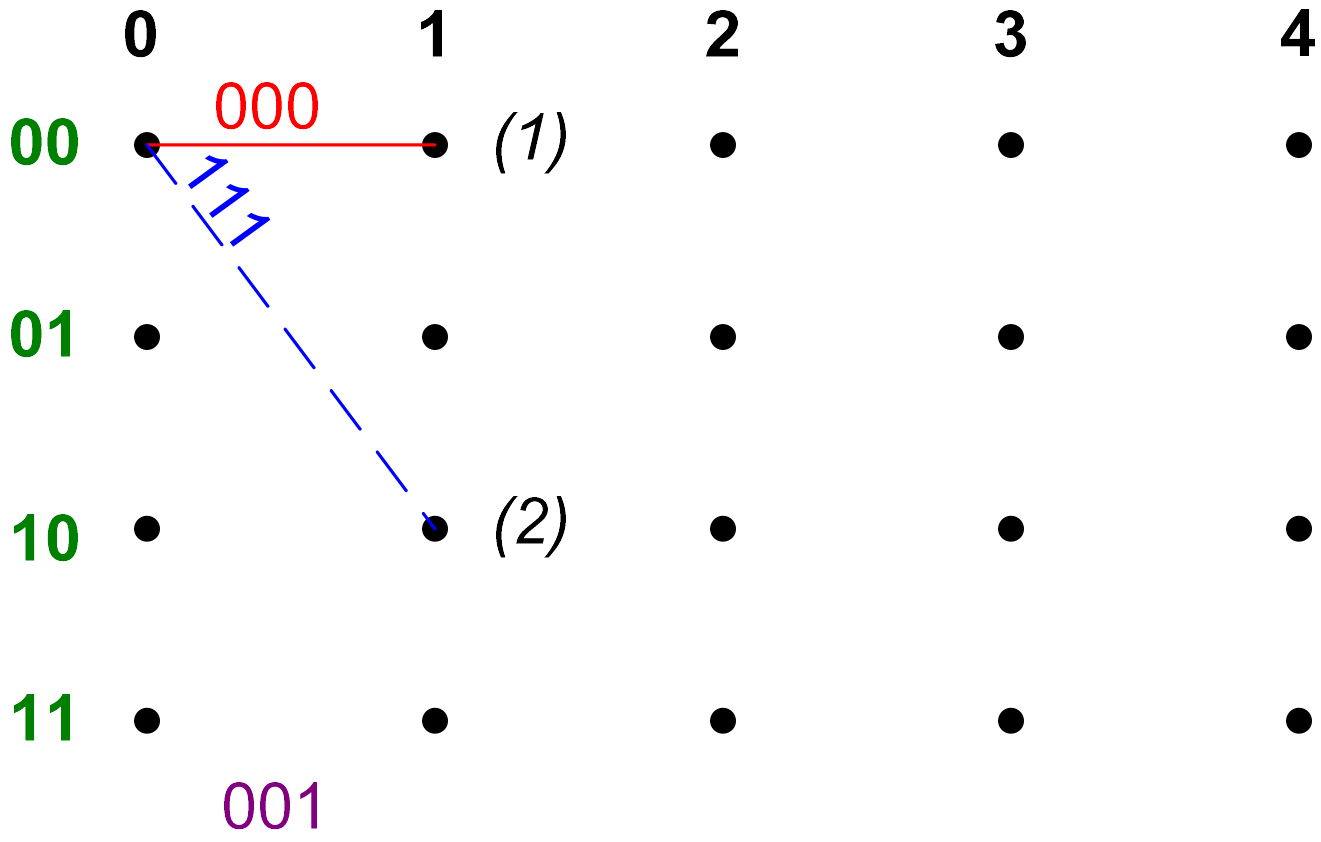




Sent codes 000-111-011-001
Received codes 001-110-011-001

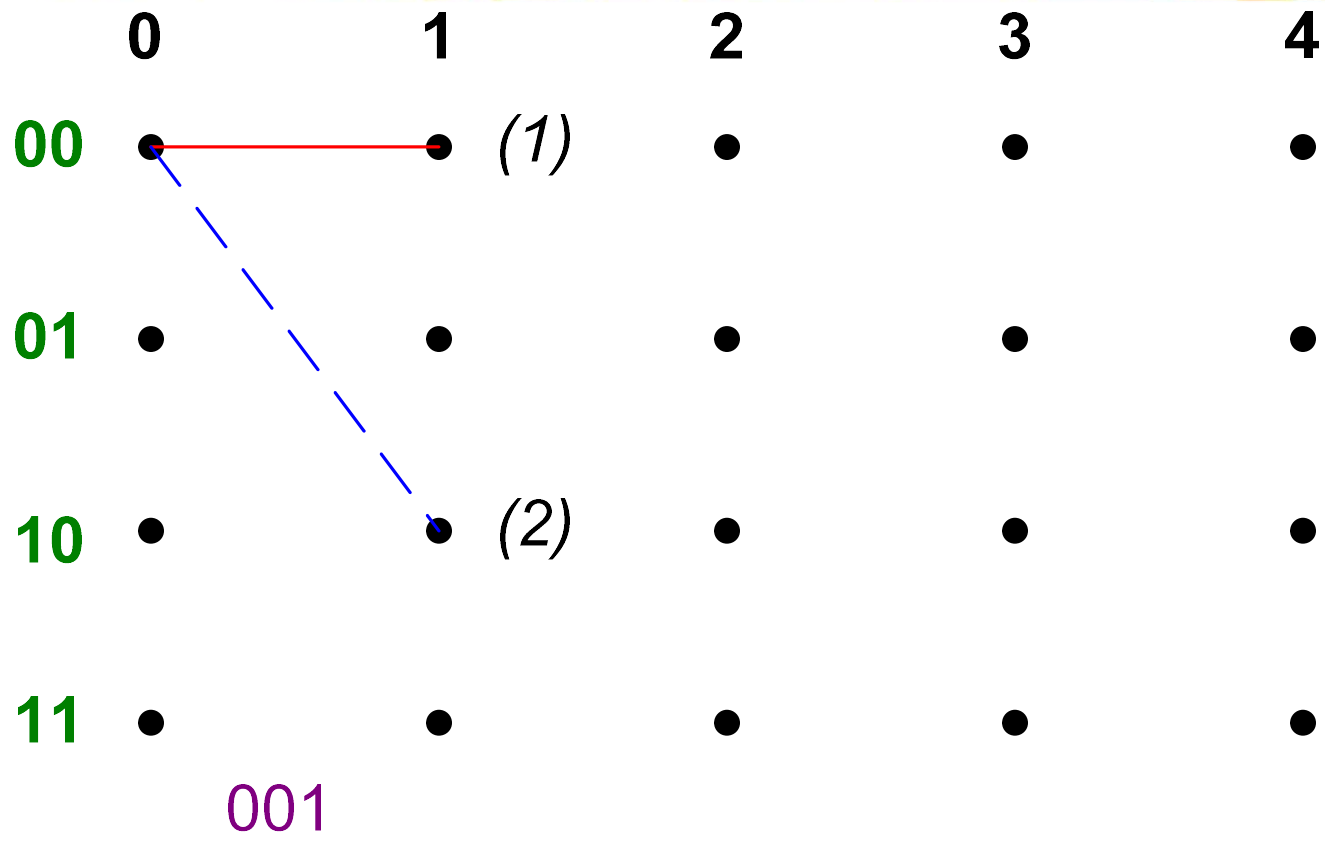
Step one

Calculate distances to the received code



Sent codes 000-111-011-001
Received codes 001-110-011-001

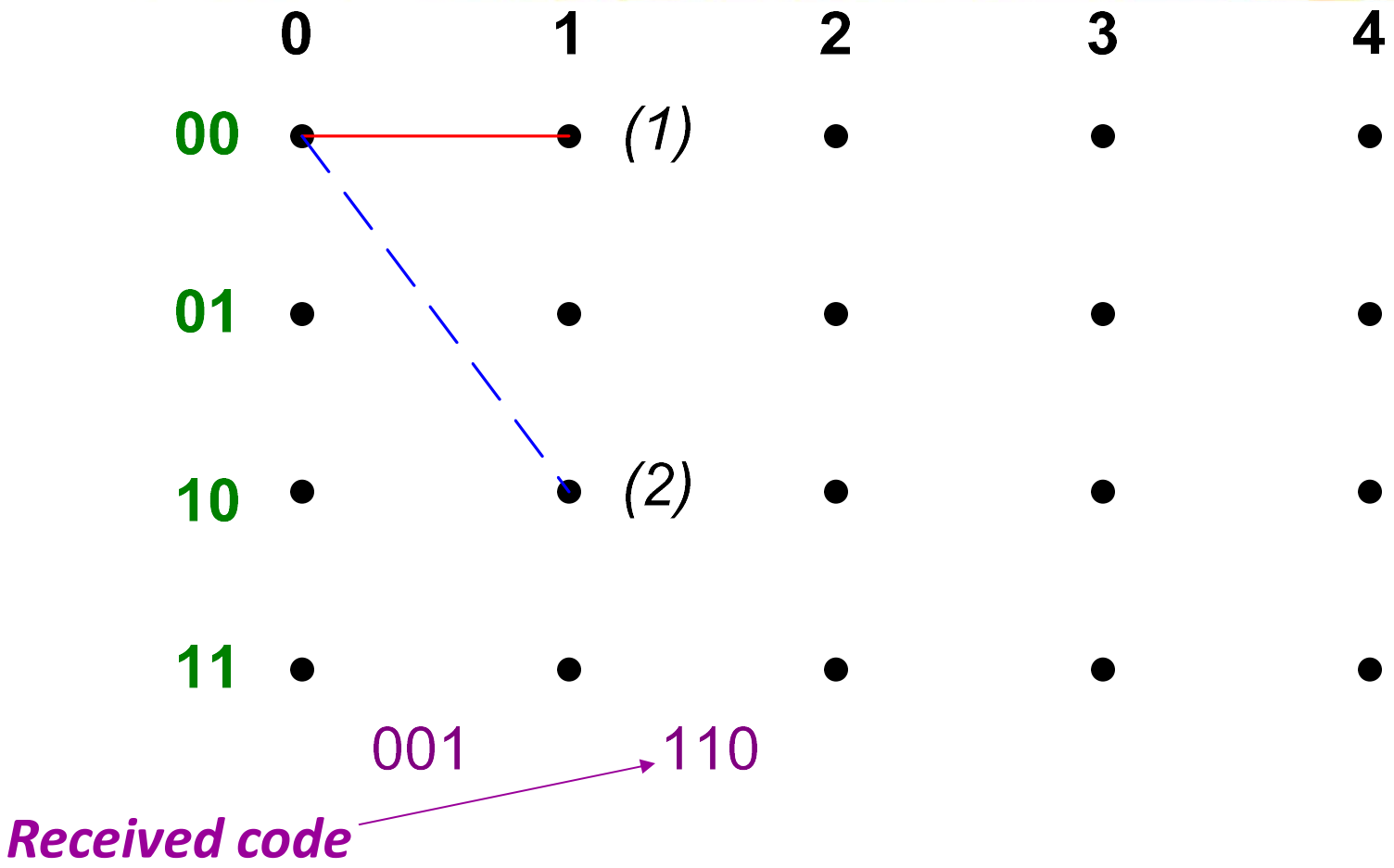
Step one



Save the paths and their distance in memory

Step two

Sent codes 000-111-011-001
Received codes 001-110-011-001

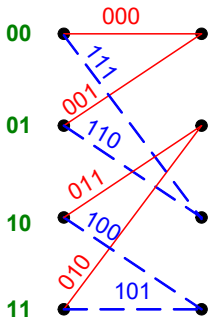


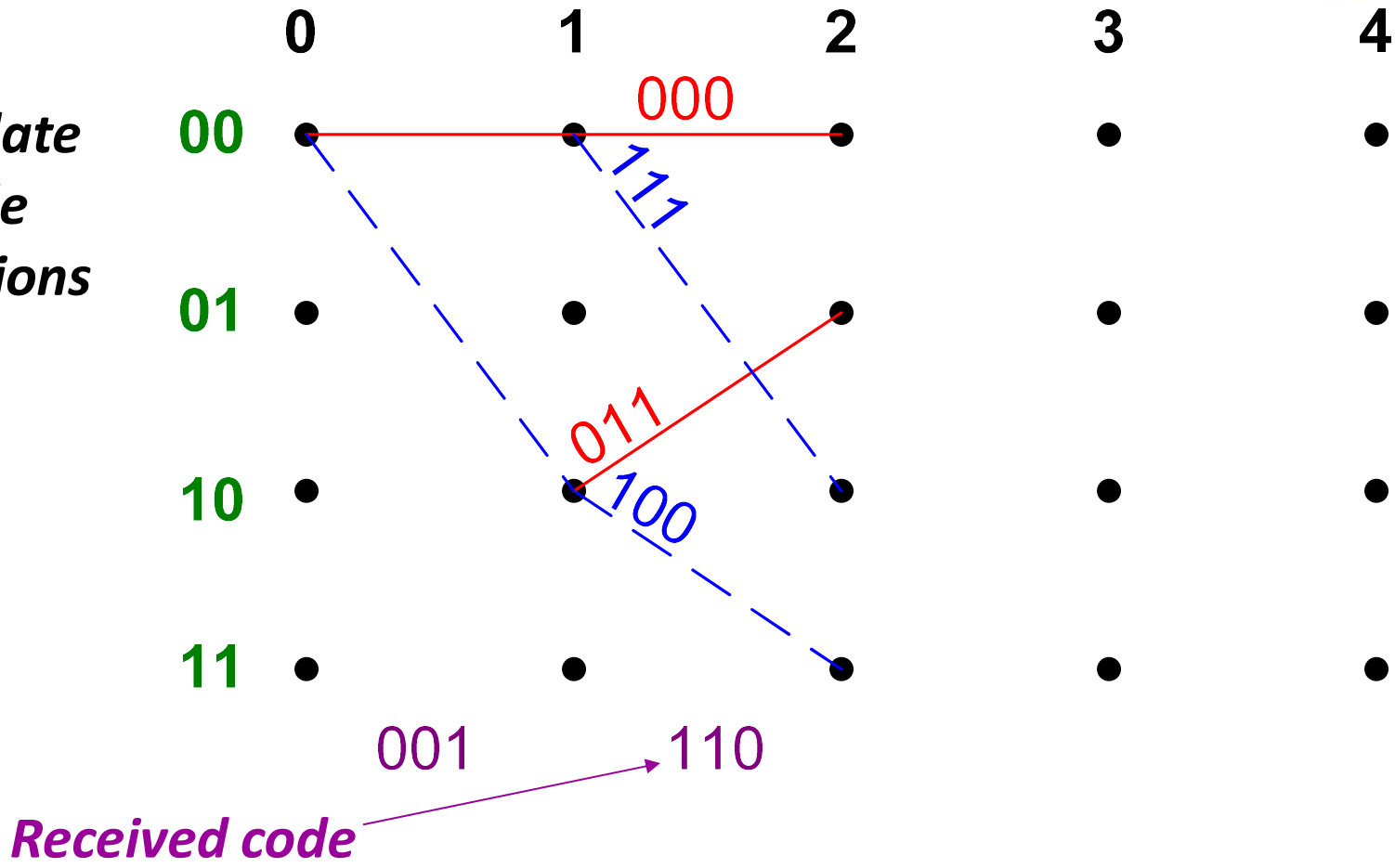
Sent codes 000-111-011-001

Received codes 001-110-011-001

Step two

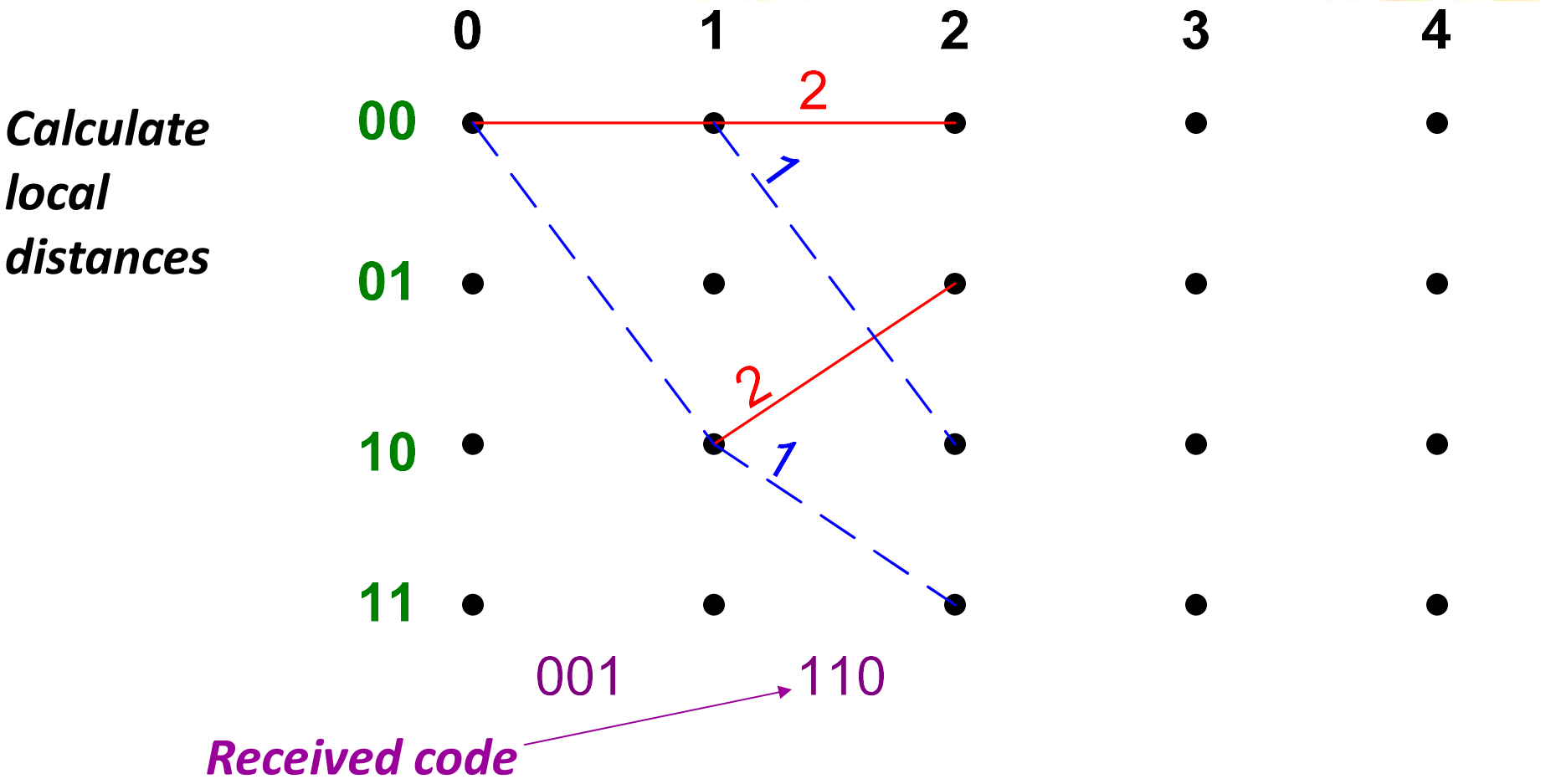
Formulate possible transitions





Sent codes 000-111-011-001
Received codes 001-110-011-001

Step two

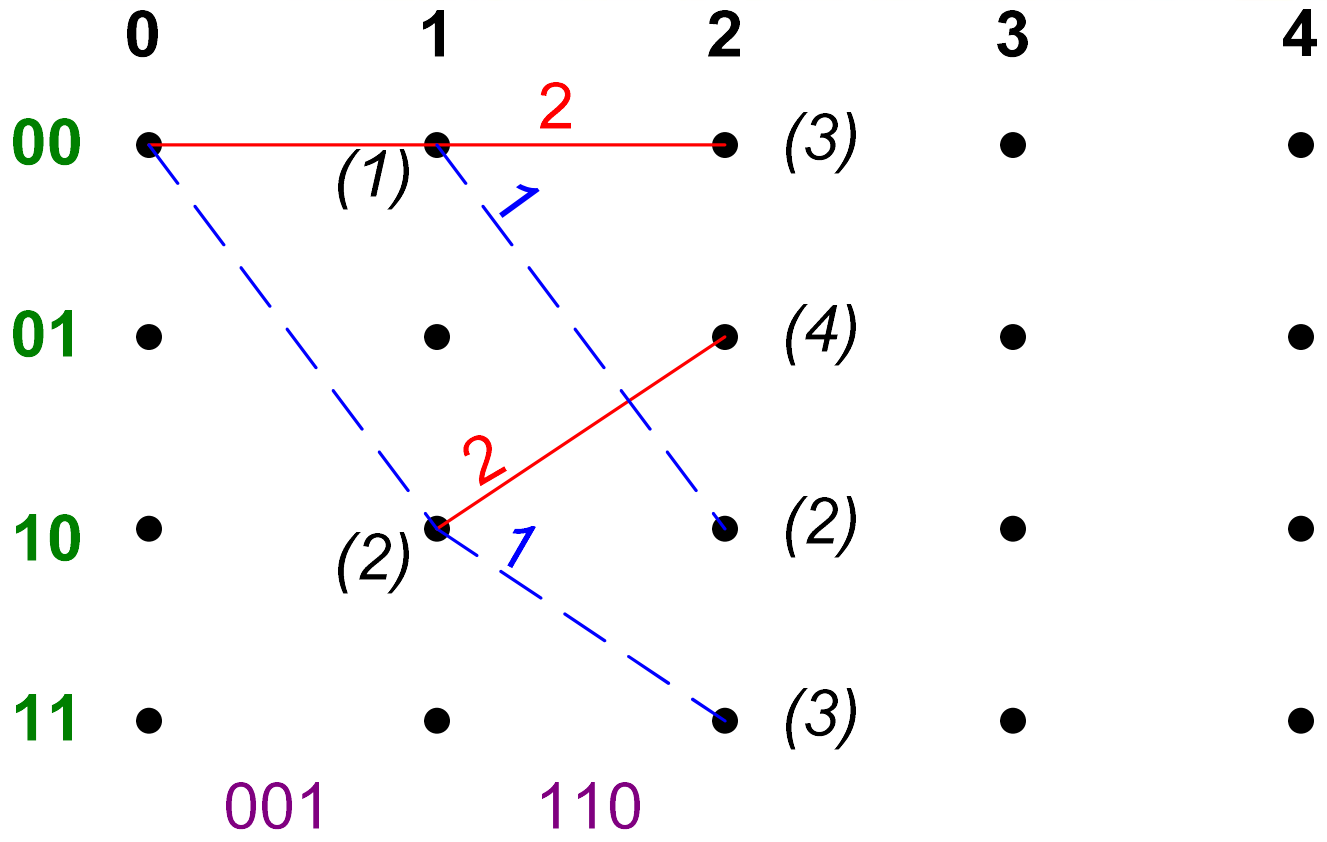


Calculate local distances

Sent codes 000-111-011-001
Received codes 001-110-011-001

Step two

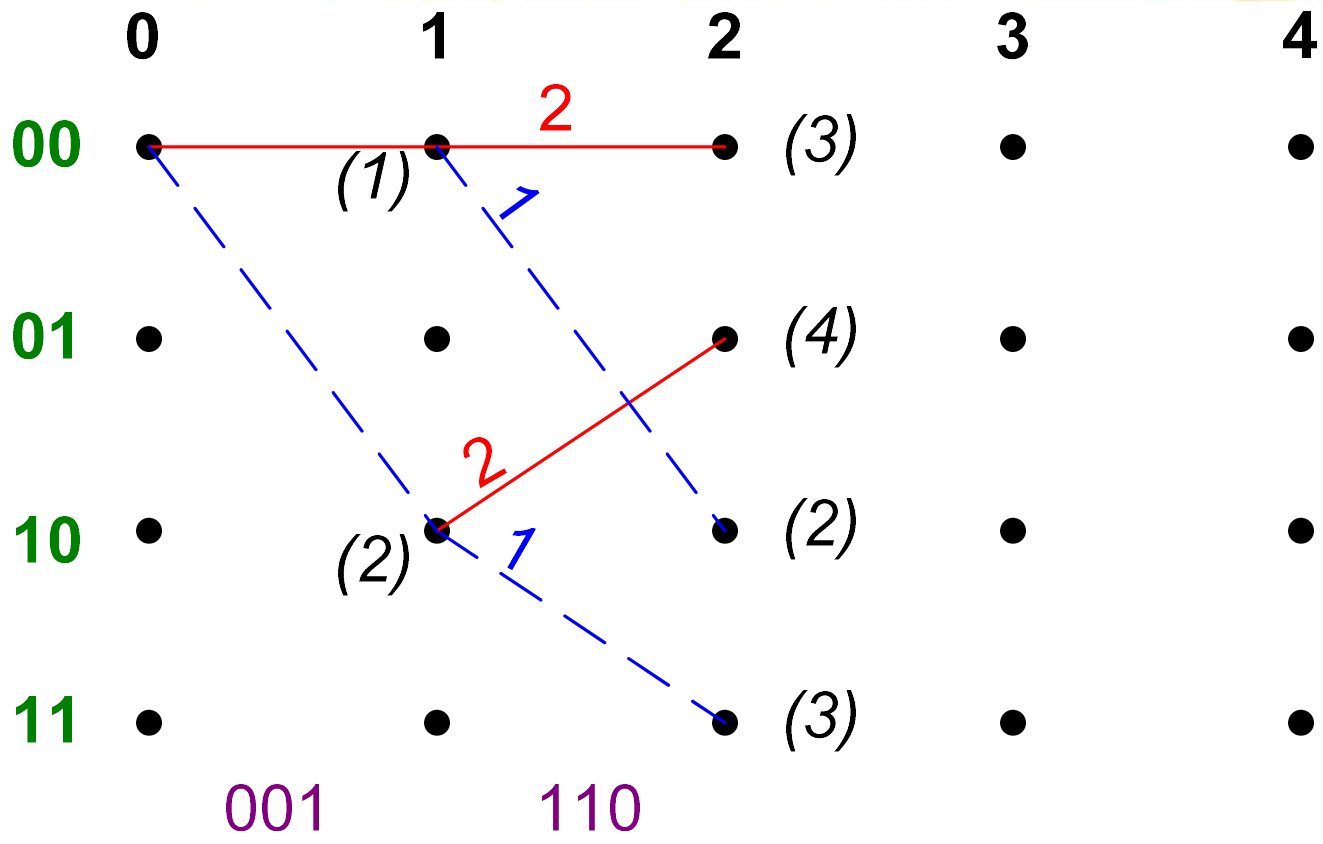
Calculate distances to the received code



Sent codes 000-111-011-001
Received codes 001-110-011-001

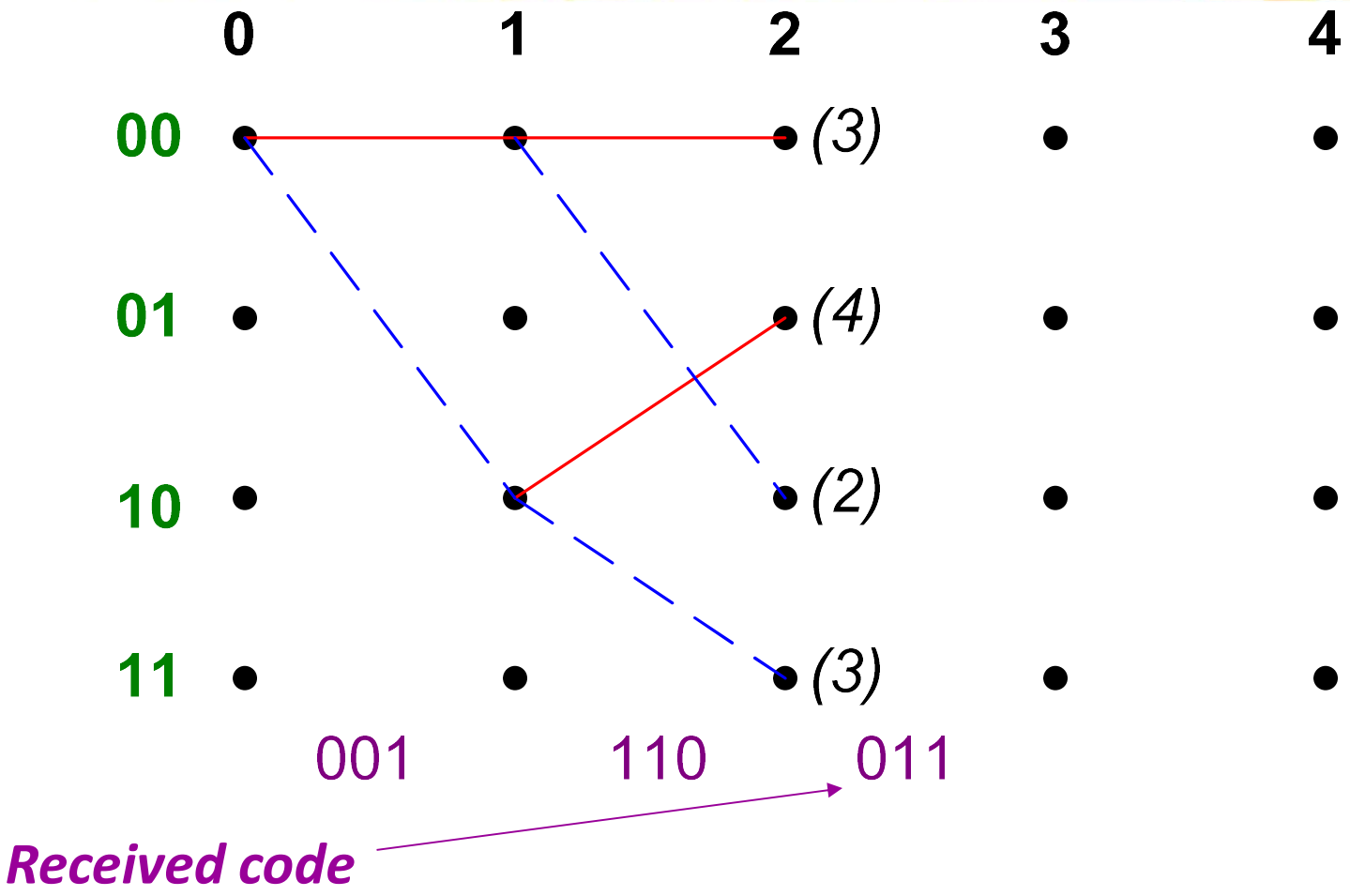
Step two

Save the paths and their distance in memory



Step three

Sent codes 000-111-011-001
Received codes 001-110-011-001

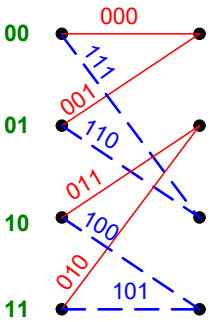


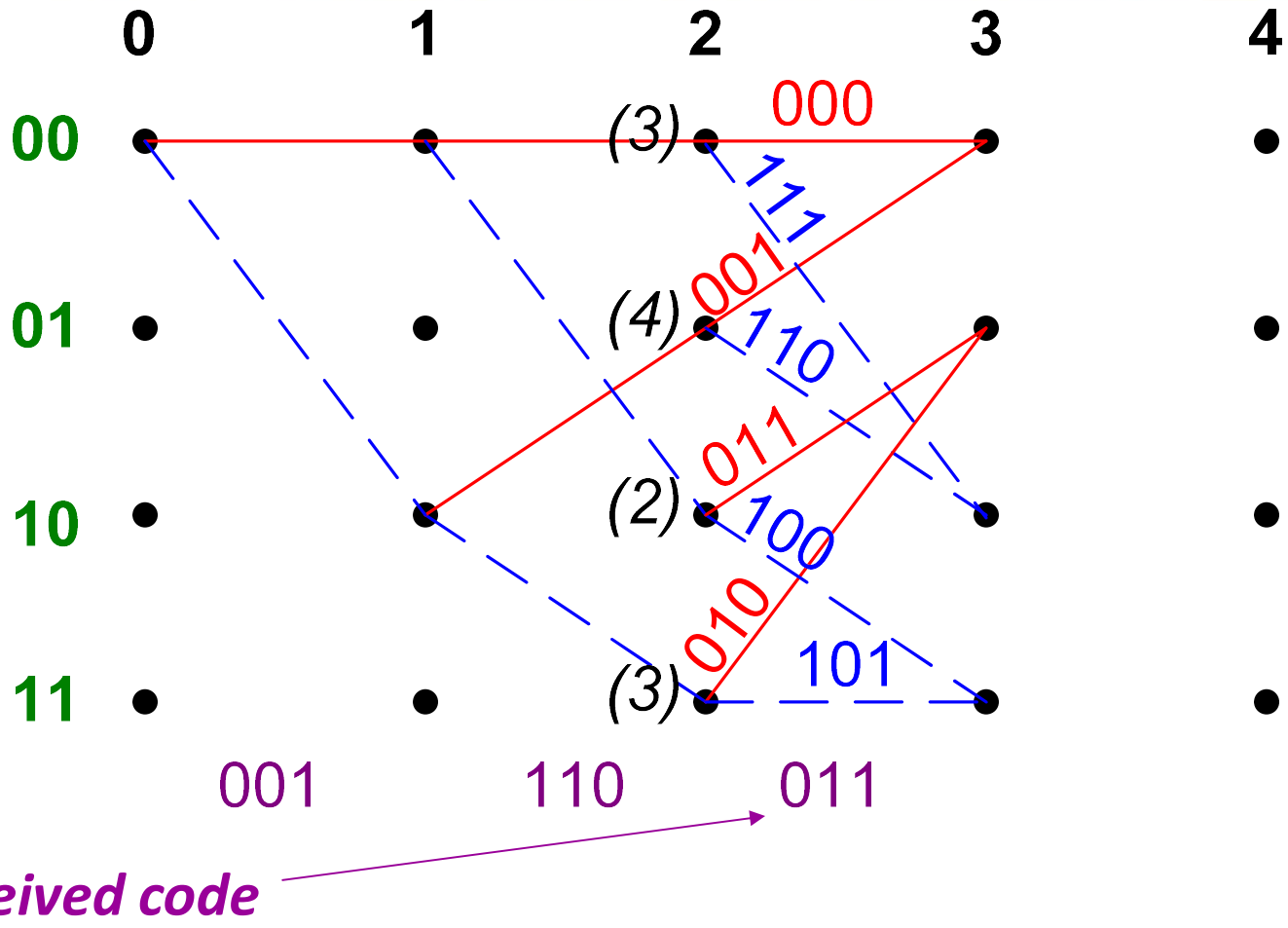
Step three

Sent codes 000-111-011-001

Received codes 001-110-011-001

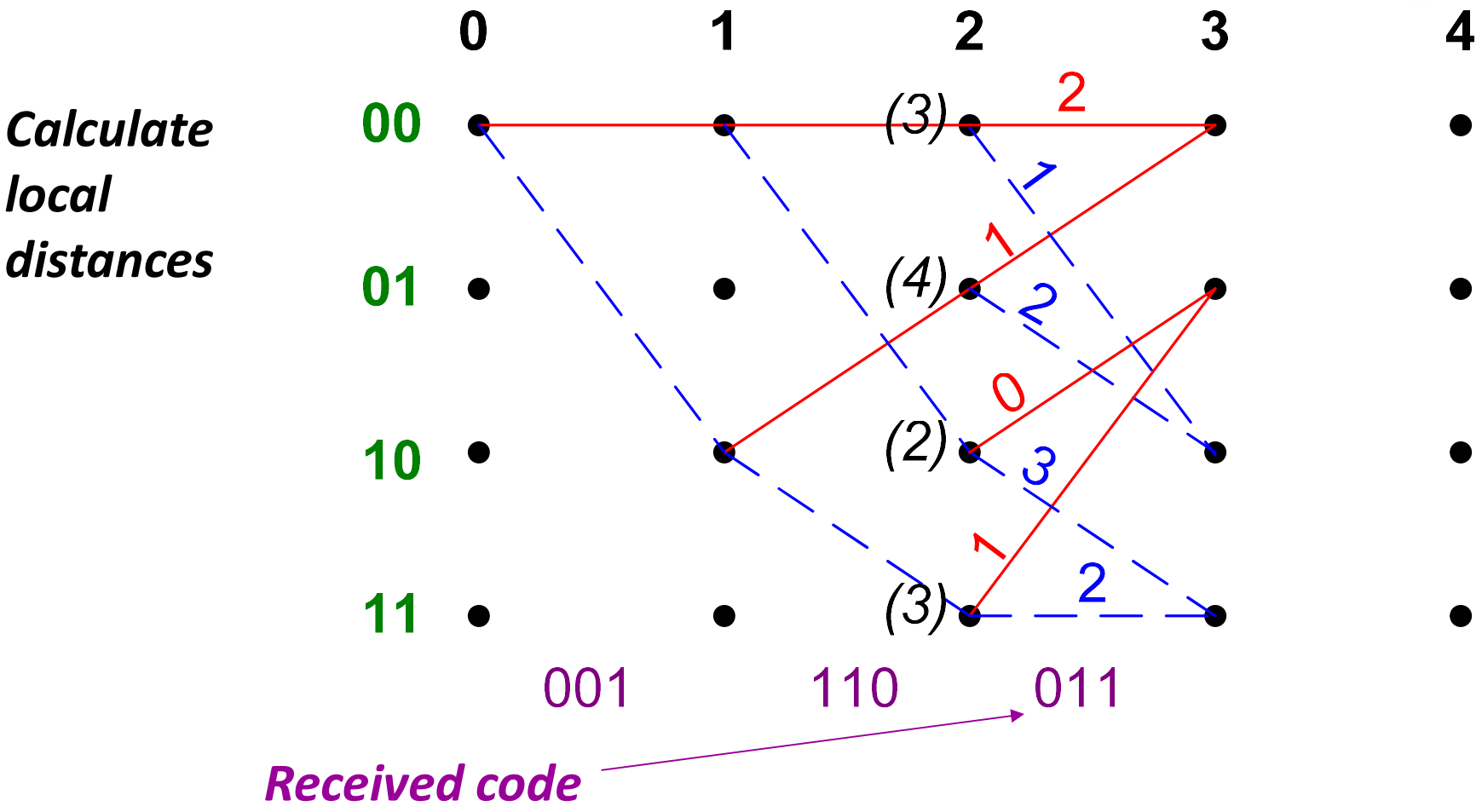
Formulate possible transitions





Sent codes 000-111-011-001
Received codes 001-110-011-001

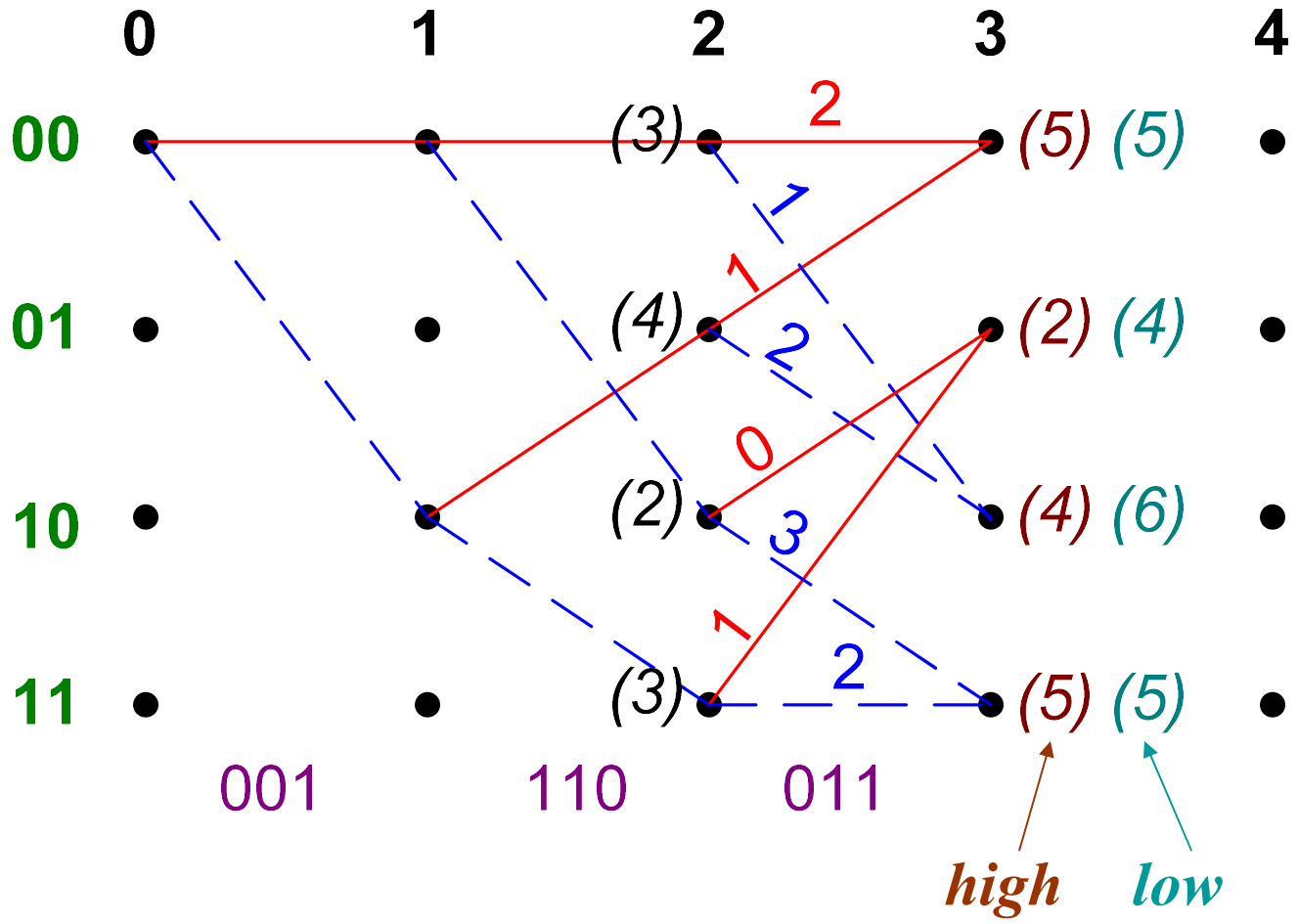
Step three



Step three

Sent codes 000-111-011-001
Received codes 001-110-011-001

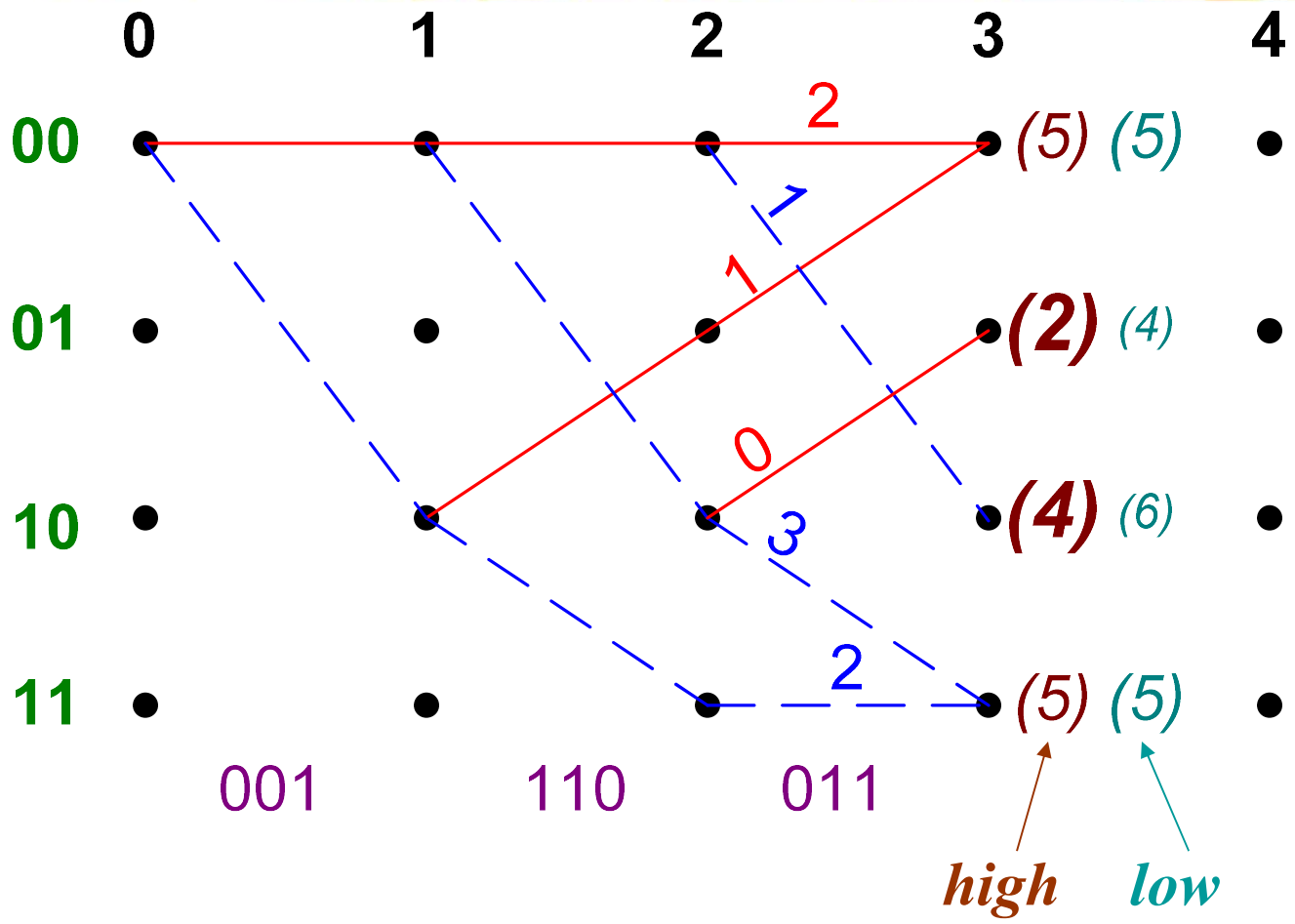
Calculate global distances



Step three

Sent codes 000-111-011-001
Received codes 001-110-011-001

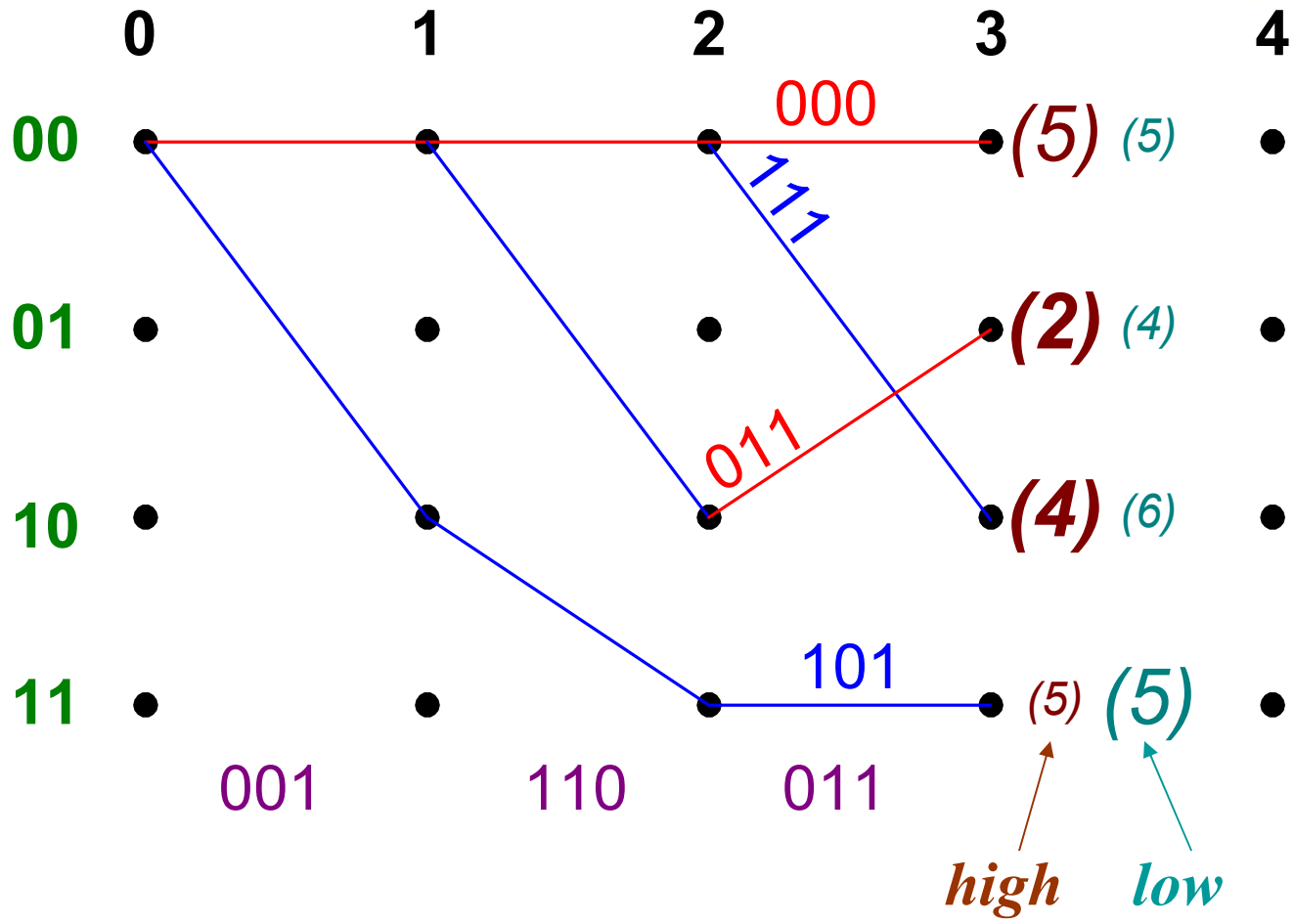
Choose the paths with the smallest distance



Step three

Sent codes 000-111-011-001
Received codes 001-110-011-001

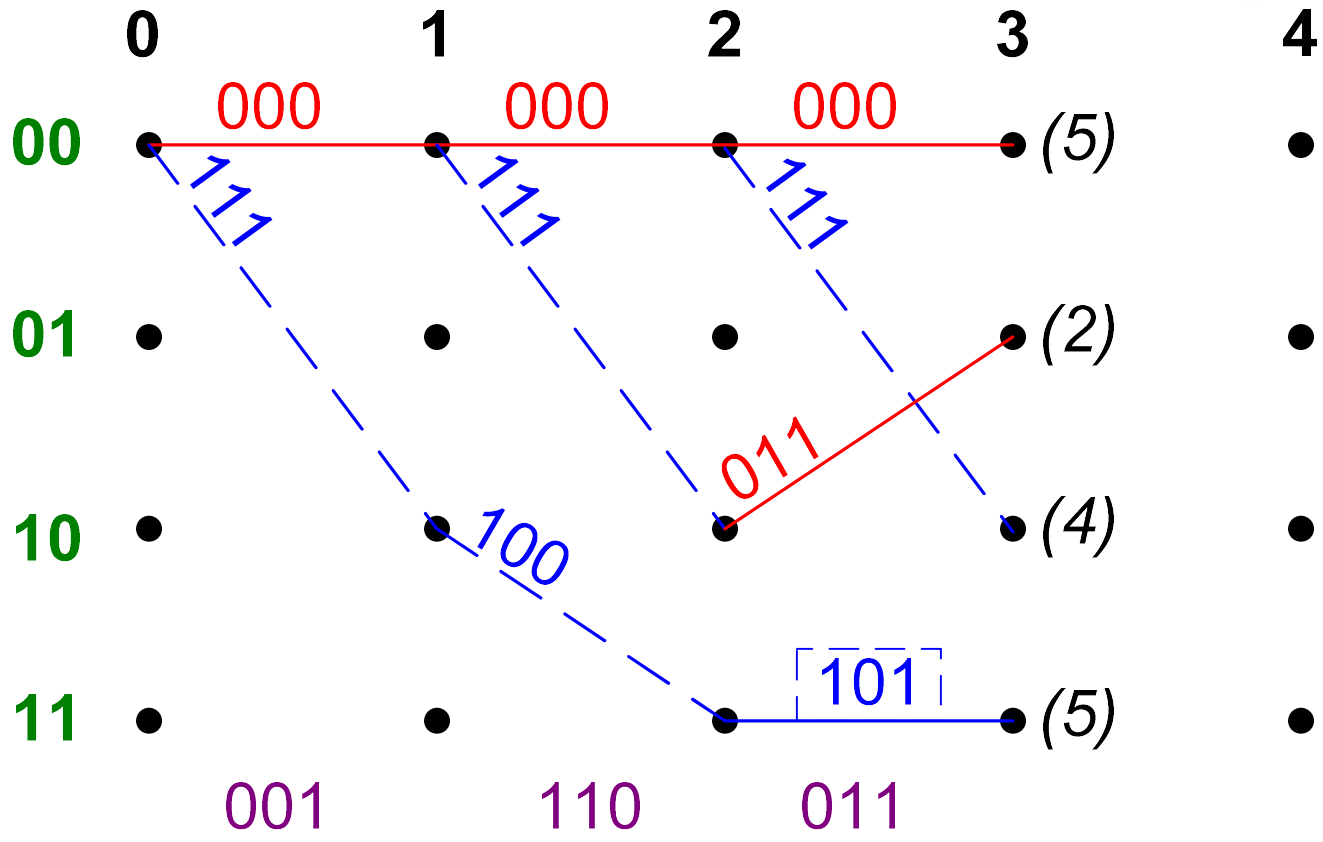
Choose by chance when the distances are the same



Step three

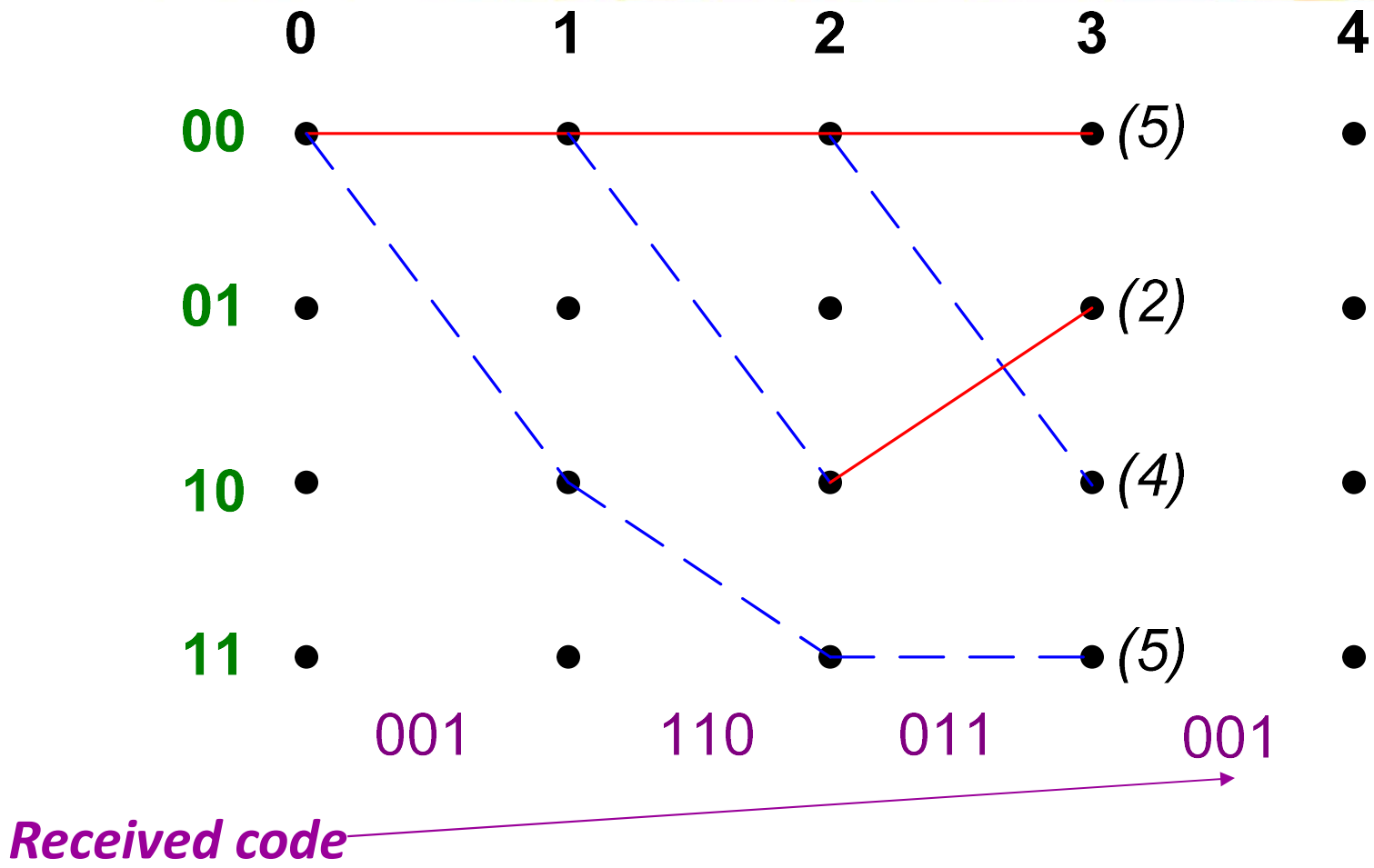
Sent codes 000-111-011-001
Received codes 001-110-011-001

Save the paths and their distance in memory



Step four

Sent codes 000-111-011-001
Received codes 001-110-011-001

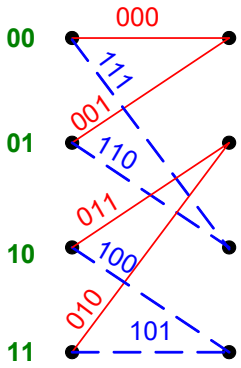


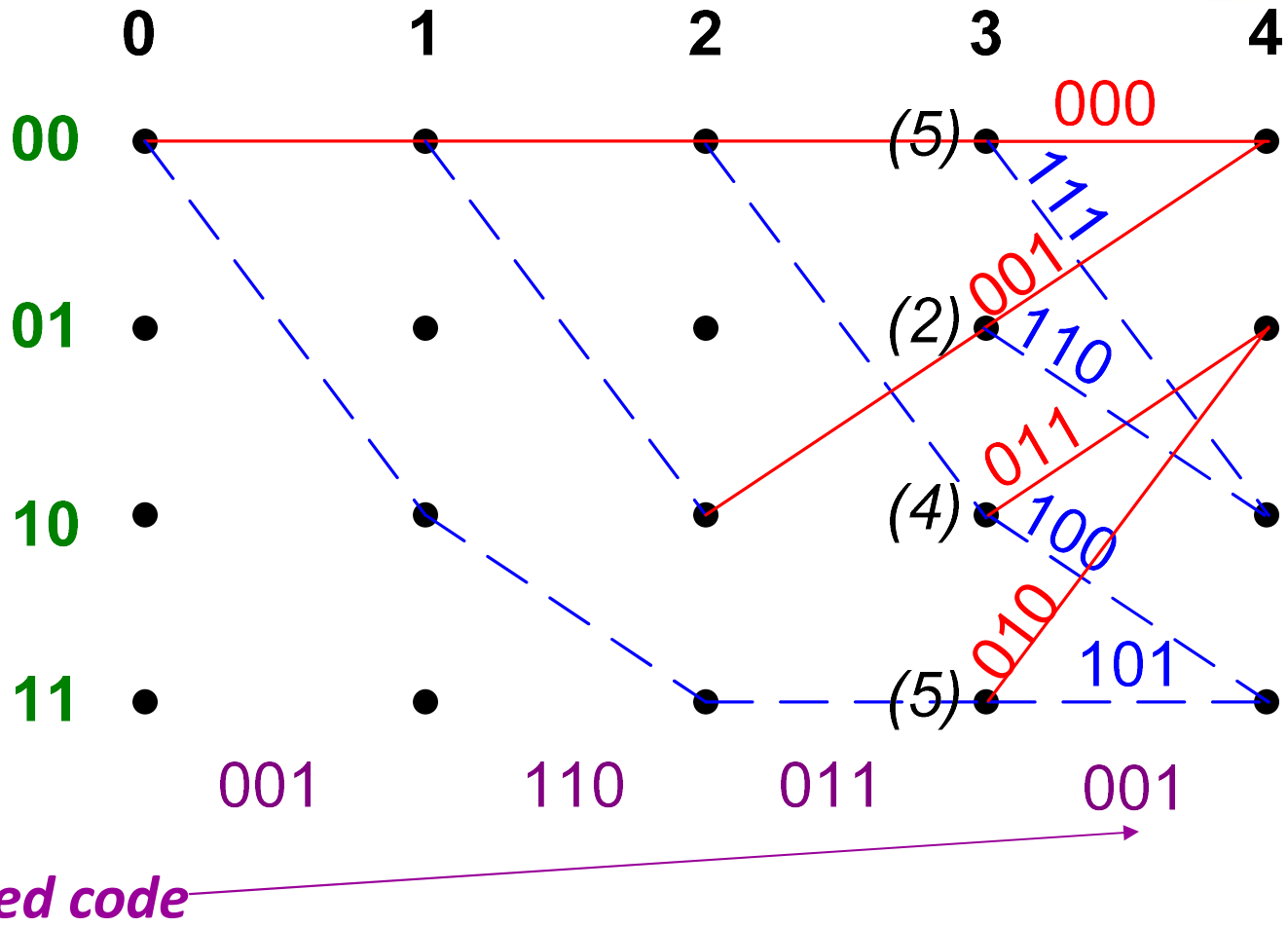
Step four

Sent codes 000-111-011-001

Received codes 001-110-011-001

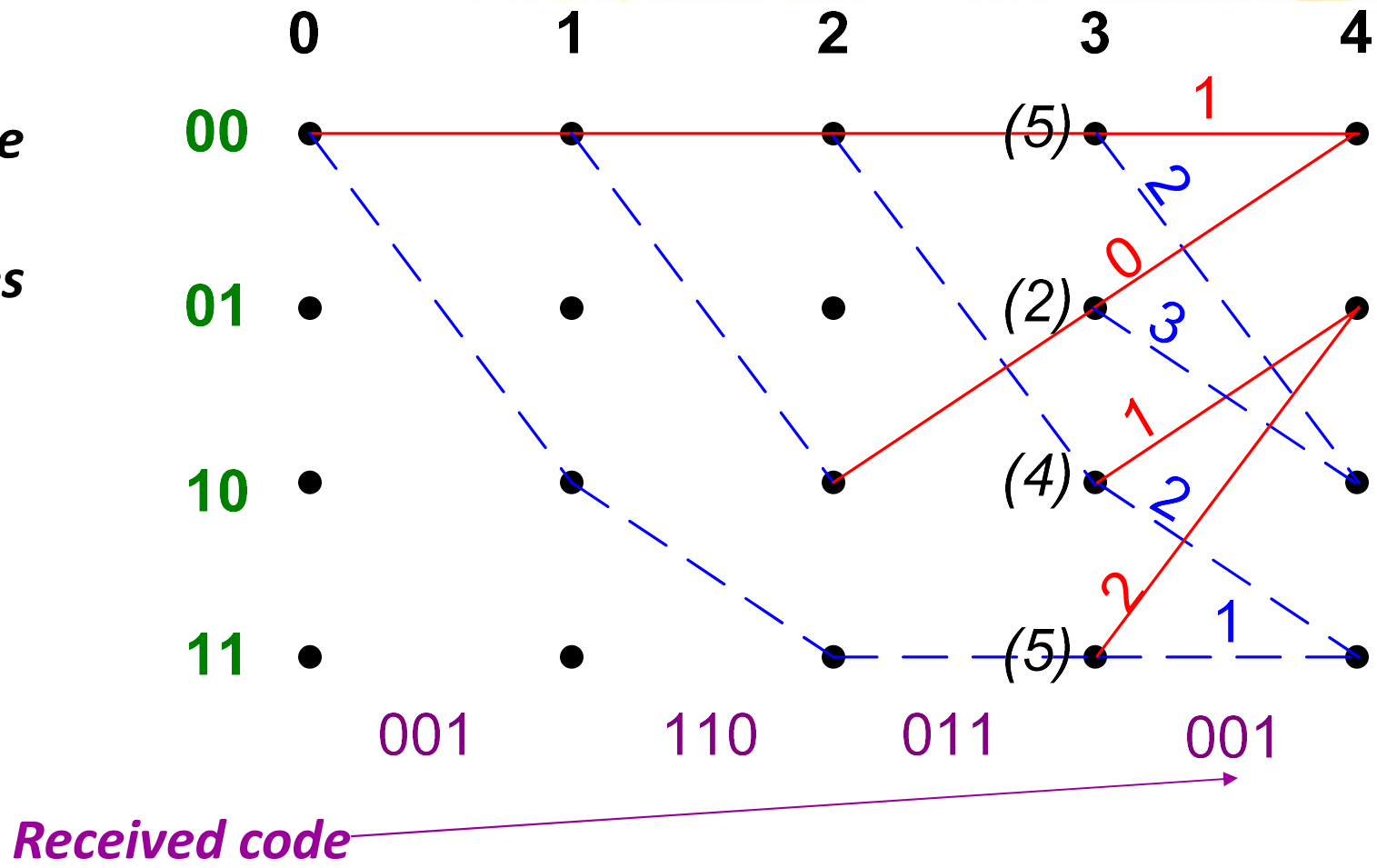
Formulate possible transitions





Sent codes 000-111-011-001
Received codes 001-110-011-001

Step four

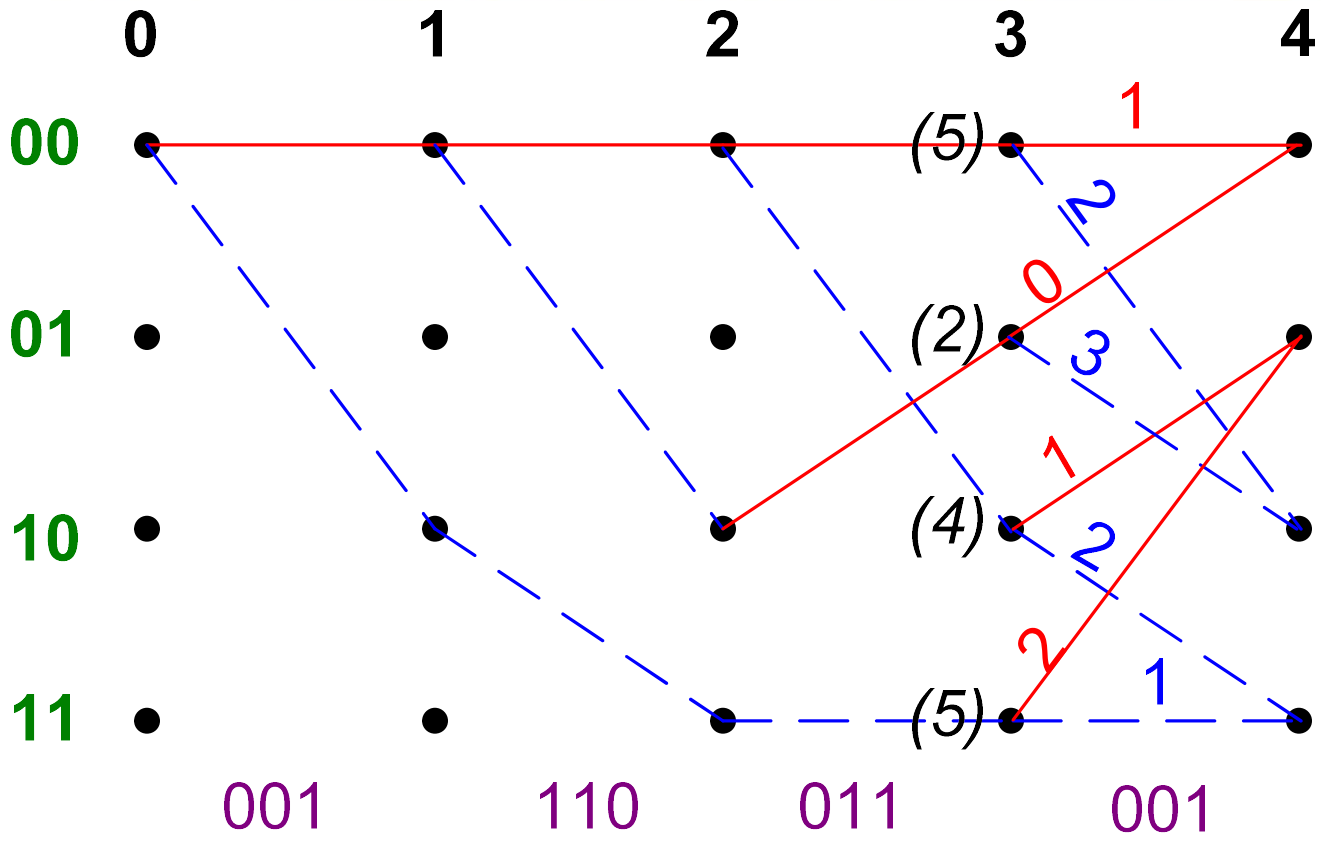


Calculate local distances

Step four

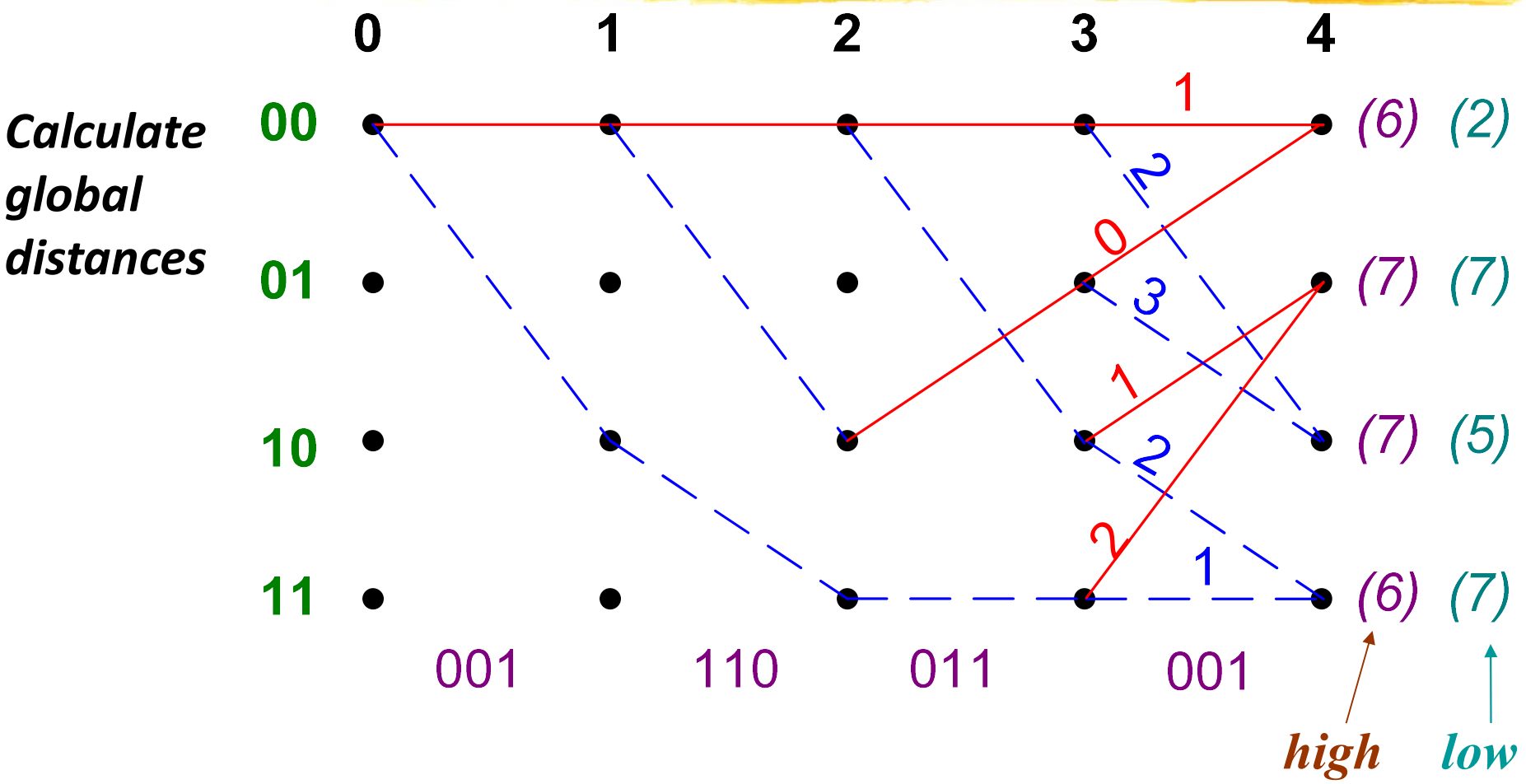
Sent codes 000-111-011-001
Received codes 001-110-011-001

Calculate global distances



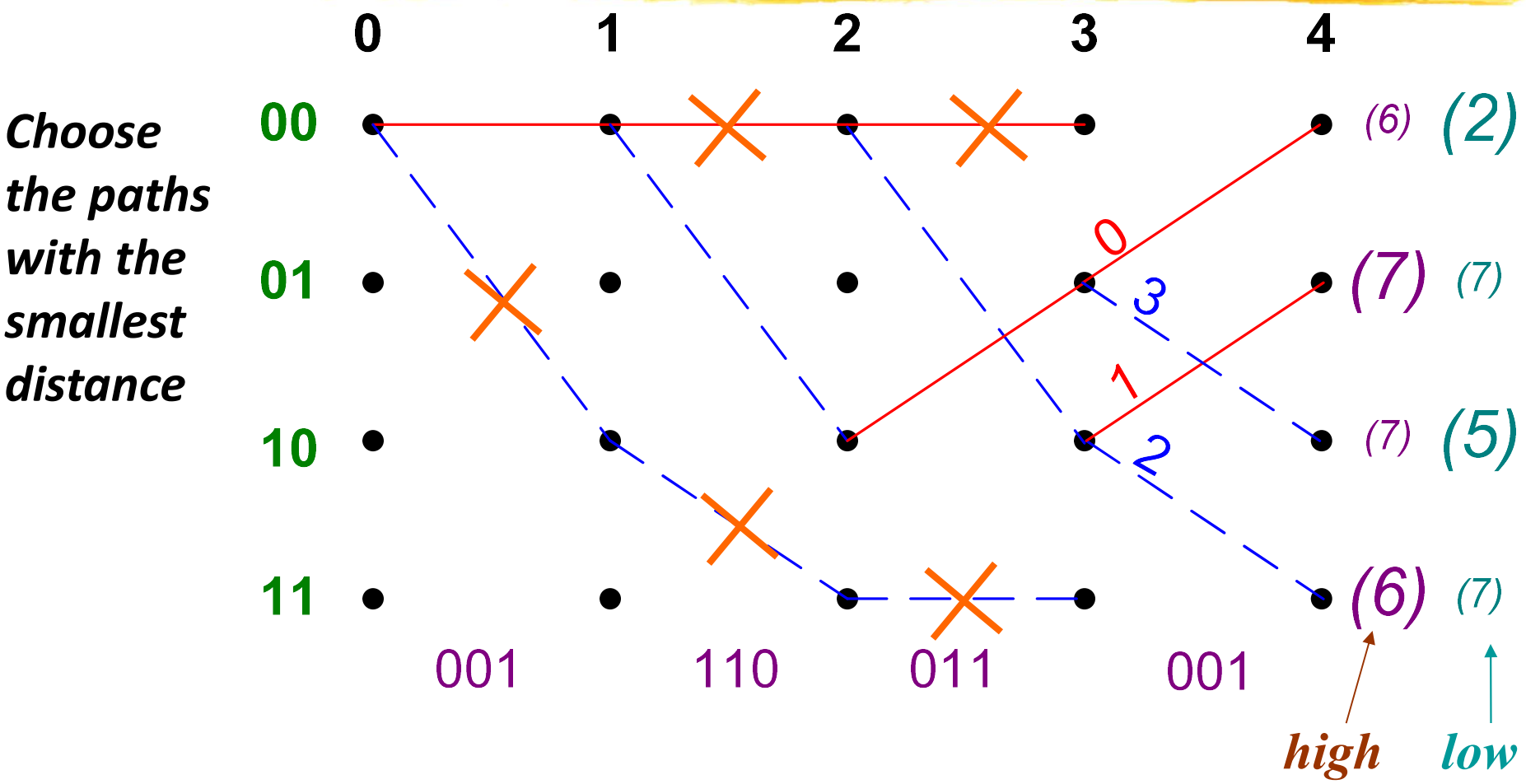
Step four

Sent codes 000-111-011-001
Received codes 001-110-011-001



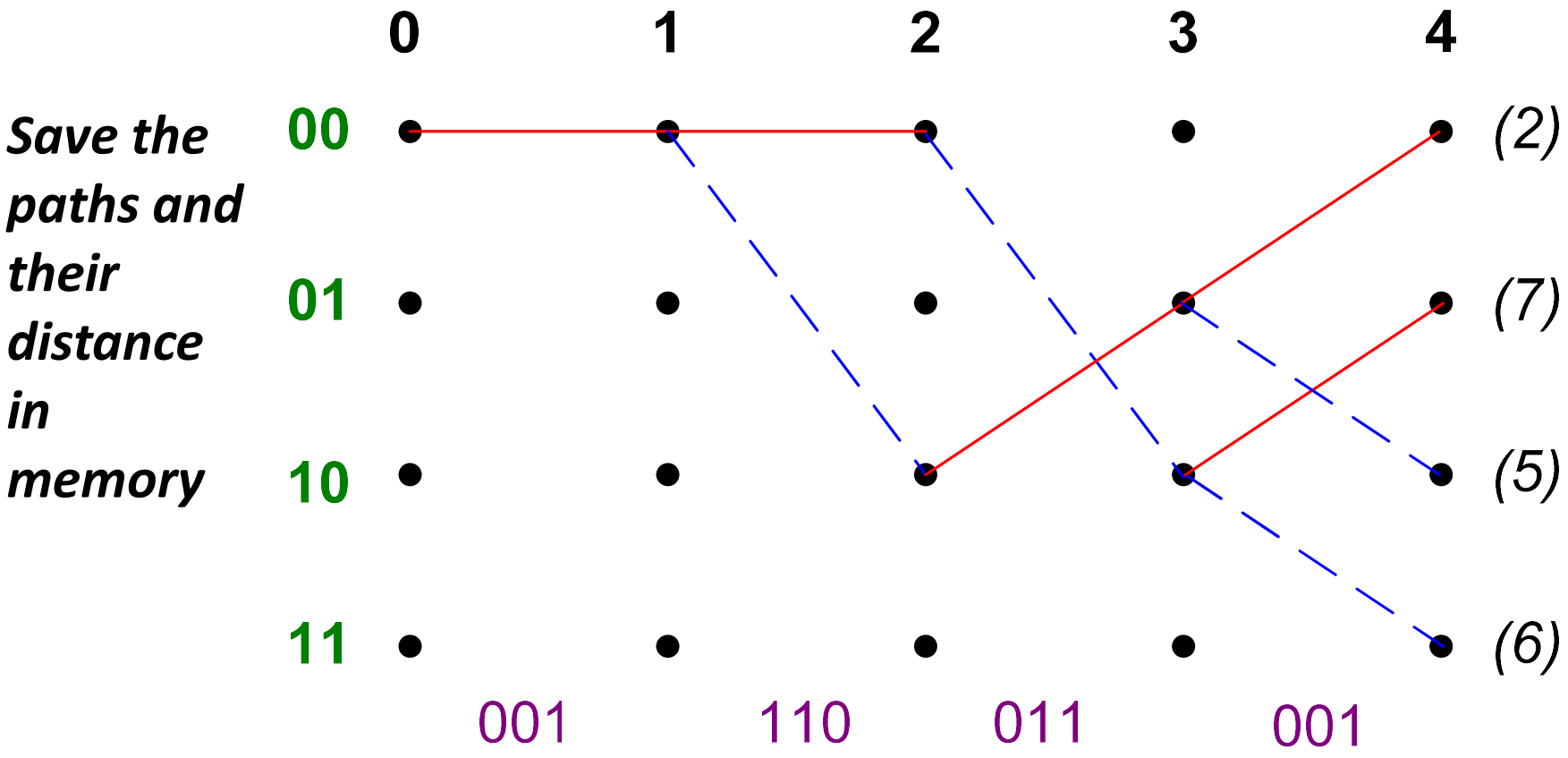
Sent codes 000-111-011-001
Received codes 001-110-011-001

Step four



Step four

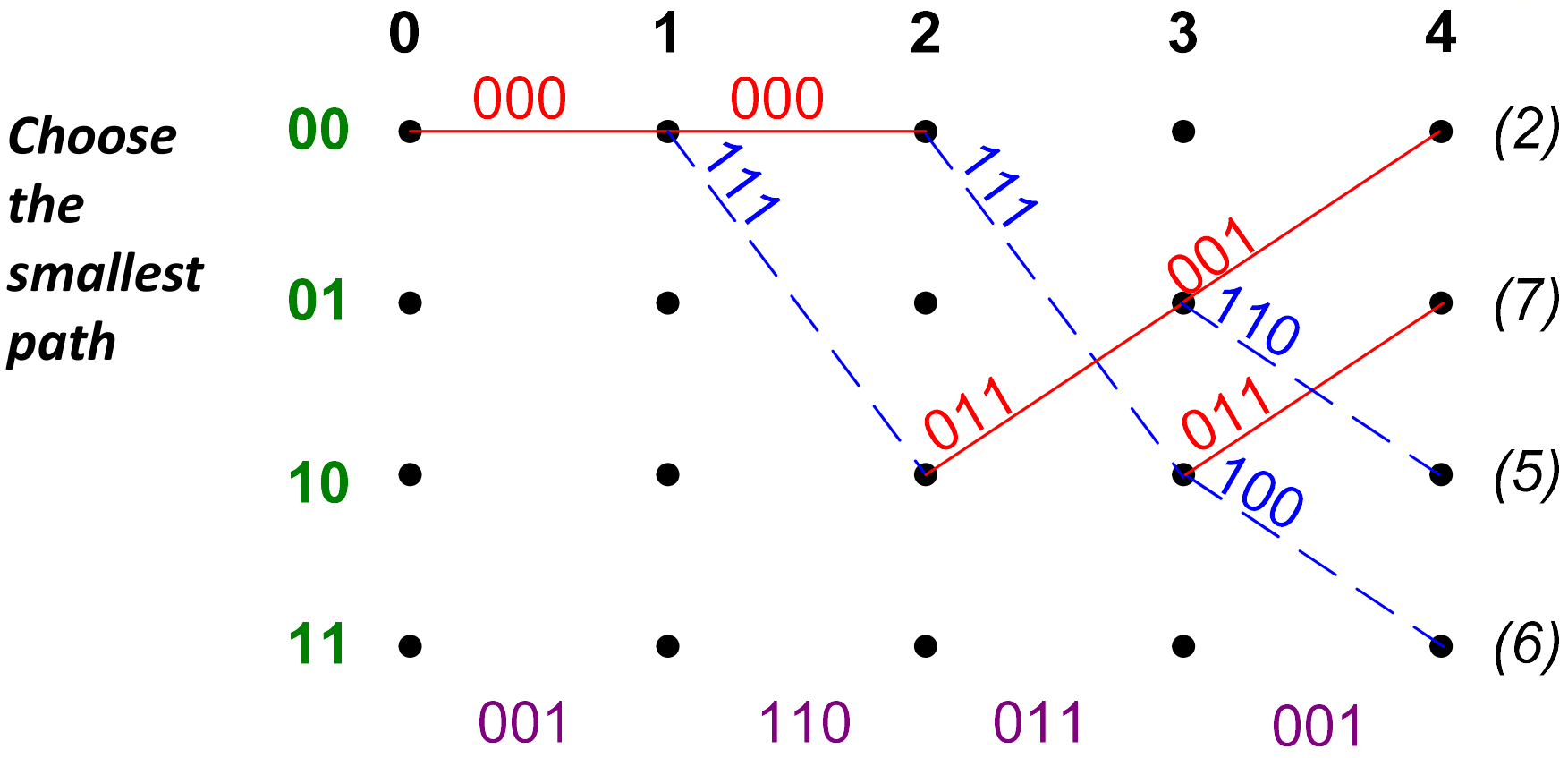
Sent codes 000-111-011-001
Received codes 001-110-011-001



Final step

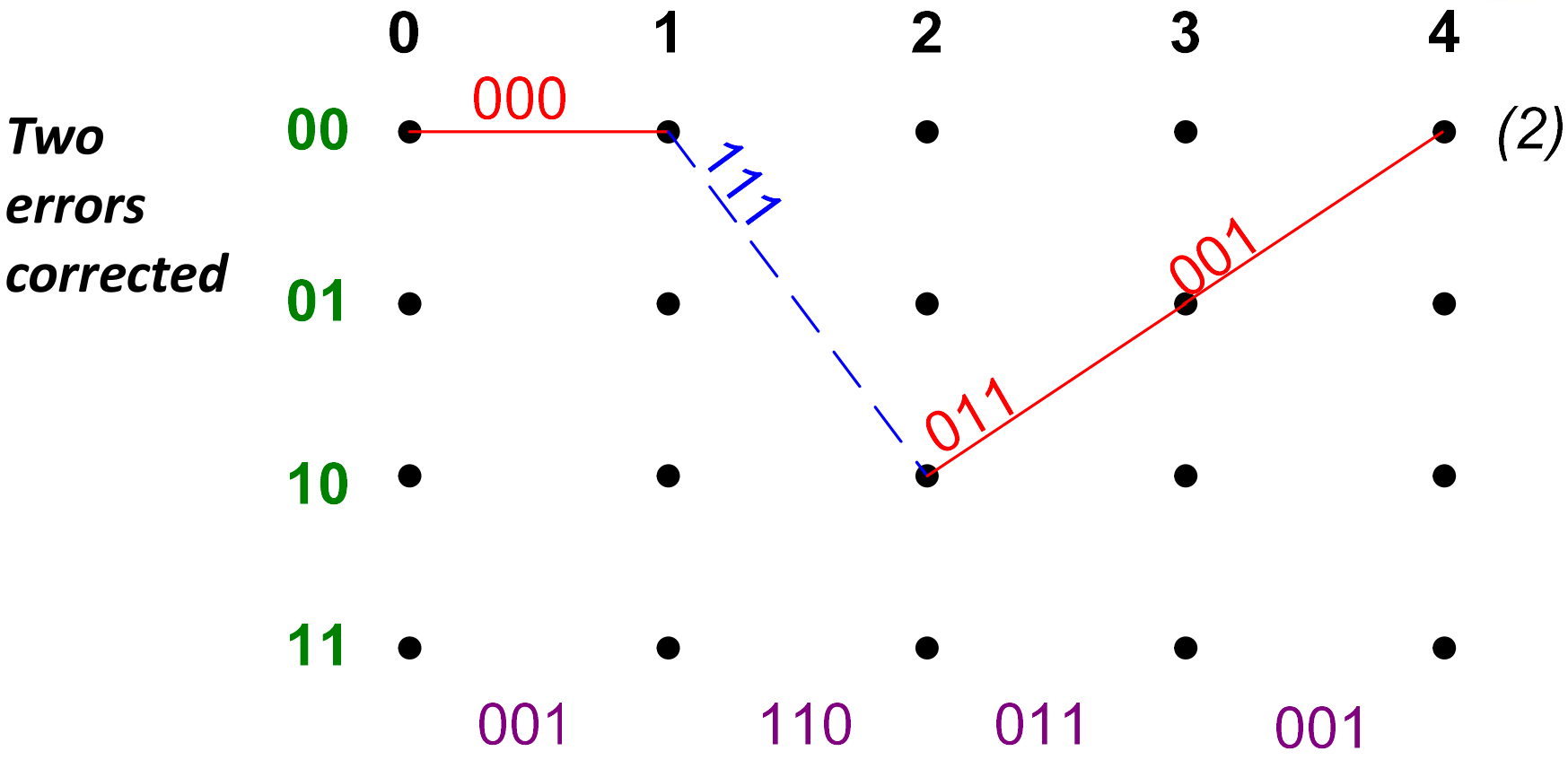
Sent codes 000-111-011-001

Received codes 001-110-011-001



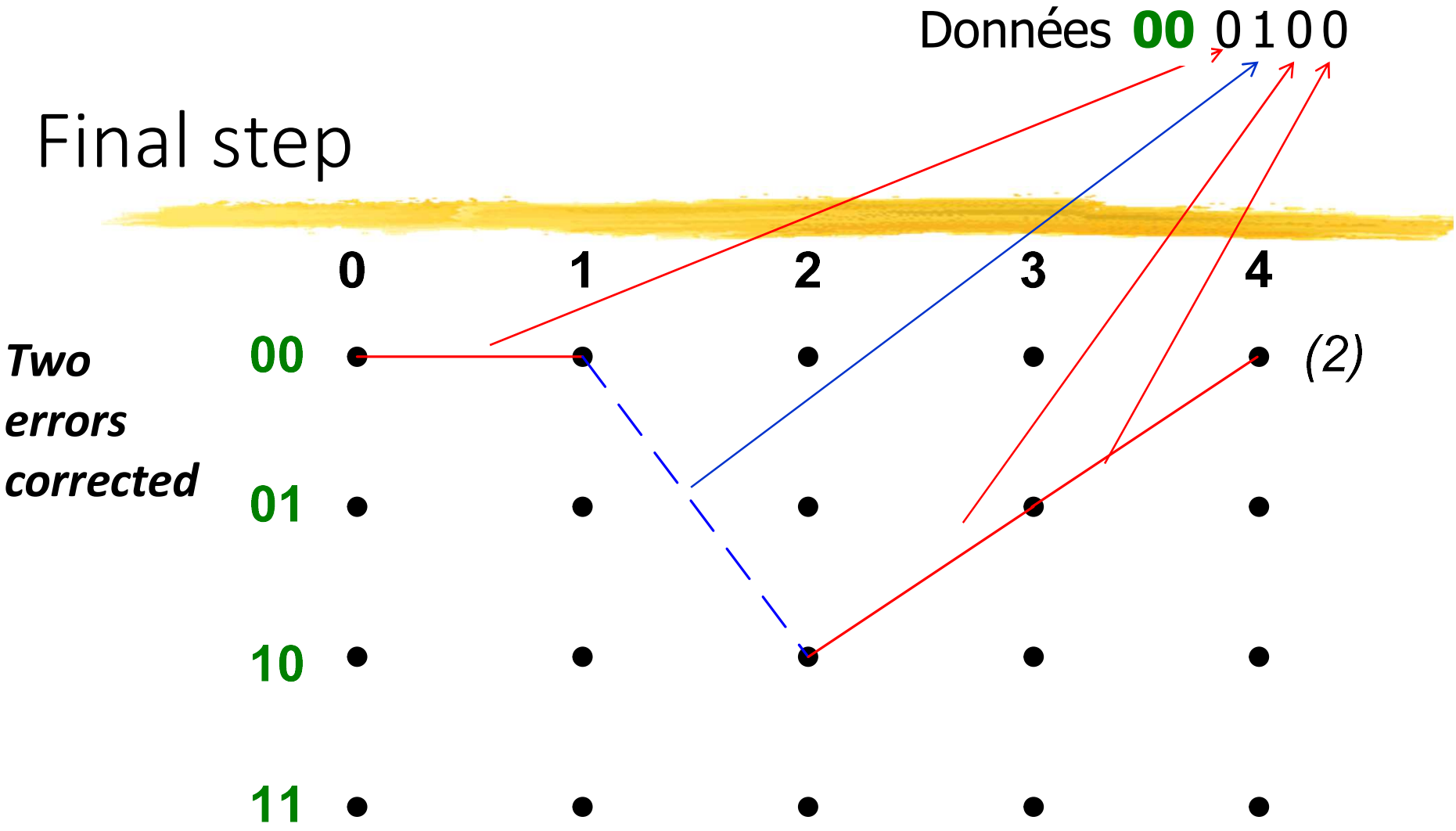
Final step

Sent codes 000 111 011 001
Received codes 001 110 011 001



Two errors corrected

Final step



Animation - Viterbi