INFORMATION THEORY and CODING

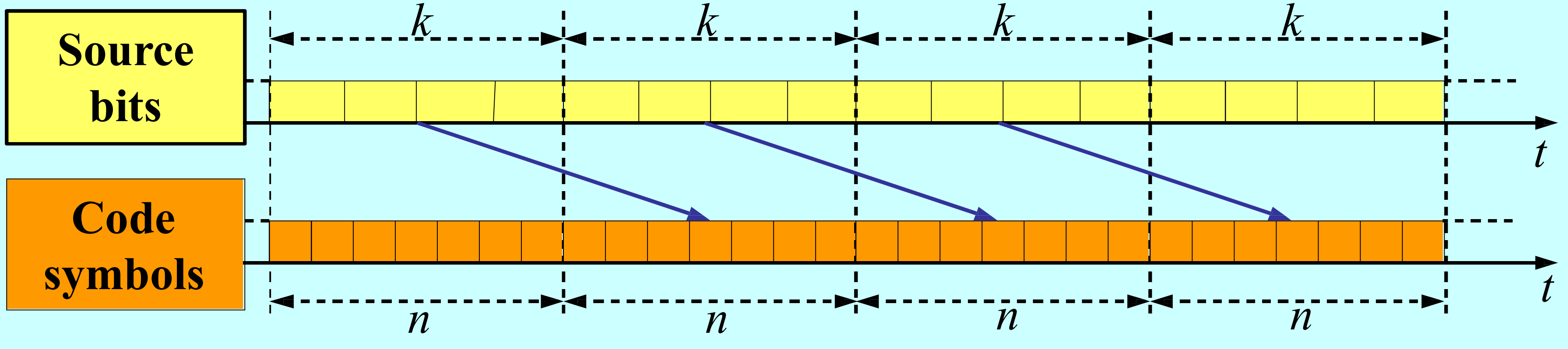
Husam Abduldaem Mohammed

LECTURE 4

Fundamentals of Convolutional Codes

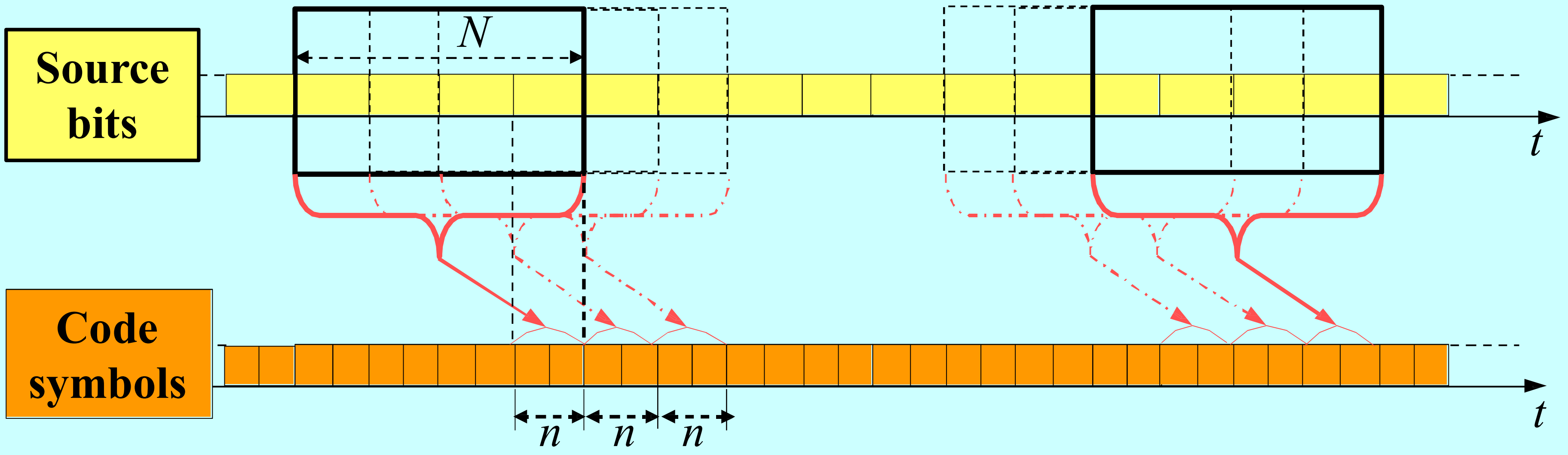
Basic Definitions

In parallel with block codes there exists a large and effective class of tree or trellis codes among which **convolutional** codes are of special interest. Their distinctive feature is somewhat different way (in comparison with block ones) of mapping source bit stream into code symbols. In block coding successive non-intersecting source *k*-bit blocks are converted into successive non-intersecting *n*-symbol codewords, so that each word protects "its individual" *k* data bits and occupies in real time space assigned to transmit these *k* bits:



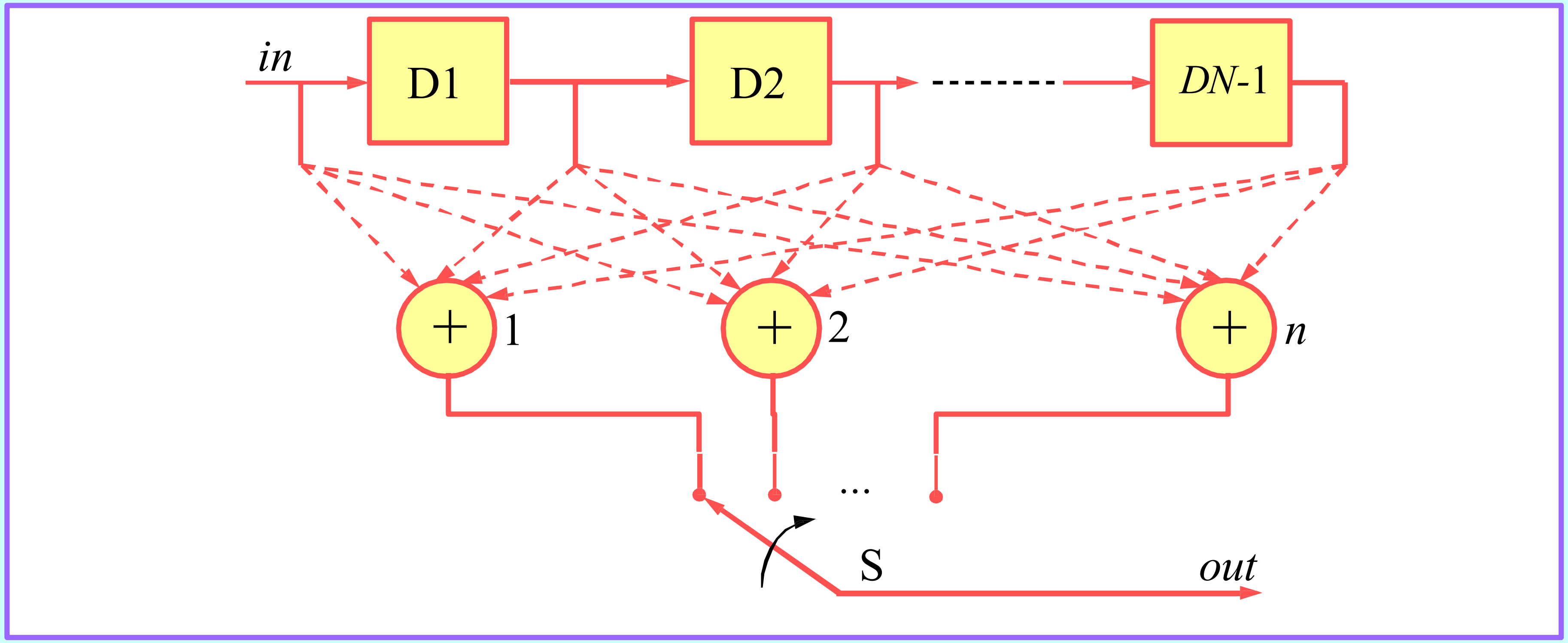
Specifically in this example k=4, n=7, R=4/7.

Under convolutional (tree, in general) coding another mode of mapping data bits into code symbols is arranged. We act as though we have a sliding window of the N-bit width picking out N-bit data block and mapping it into a group of n code symbols transmitted during only one data bit duration. After that window shifts to the right by one bit duration and renewed N-bit block is mapped into the next n-symbol group, etc. Unlike block code we have now not successive individual codewords separated in time but, instead, continuous stream of code symbols in which every n-symbol group is responsible for protection the current data bit along with the N-1 previous:



Parameter N is the number of the shift register cells. Parameter L of the convolutional code shows how many source bits affect current n-symbol group after encoding and is named **constraint length**. In our example above $N = 4$ and $n = 2$.

Structure based on shift register is a usual tool for embodying the described encoding principle. It includes *N*-1 binary cells (delay elements) the only function of which is to store *N*-1 previous source bits in order to have, along with a current bit, *m*-bit segment encoded into a current code *n*-symbol group. Encoding itself is executed by *n* modulo 2 adders. Switch *S* performs multiplexing (connecting in



turn all modulo 2 adders to the output) so that during one bit duration *n* code symbols will be generated. After present *N* bits are encoded the register is clocked and new bit appears at the input while the "oldest" is clocked out of the register and does not influence longer the encoder output.

Code rate is defined as k/n. the parameter L is called the **constraint length** of the convolutional code, which is **the number of information bits over which the convolution takes place.**

N is the number of the shift register cells.

k is the input message digits within the time unit.

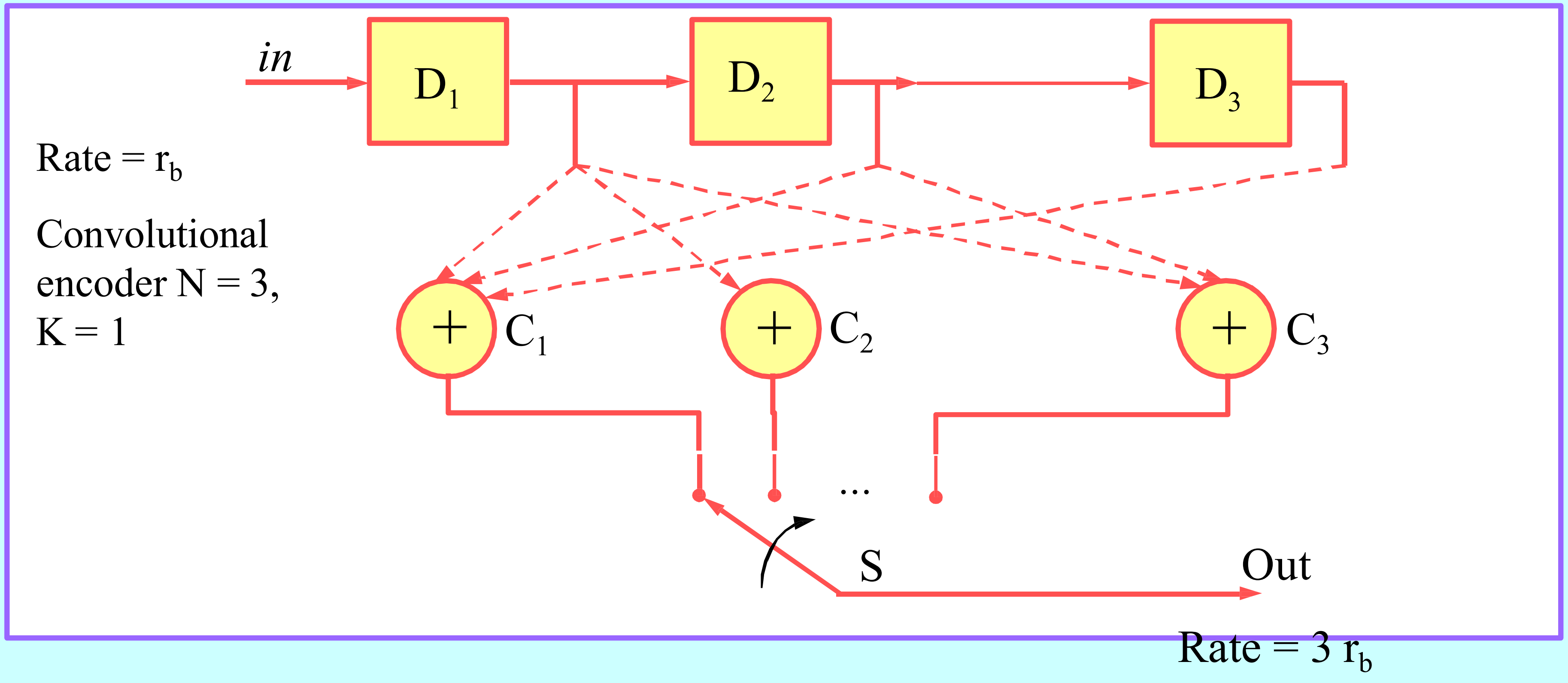
n code block digits. $k < n$.

L **constraint length** of the convolutional code.

Code rate k/n

(n,k) convolutional code.

Example 1:

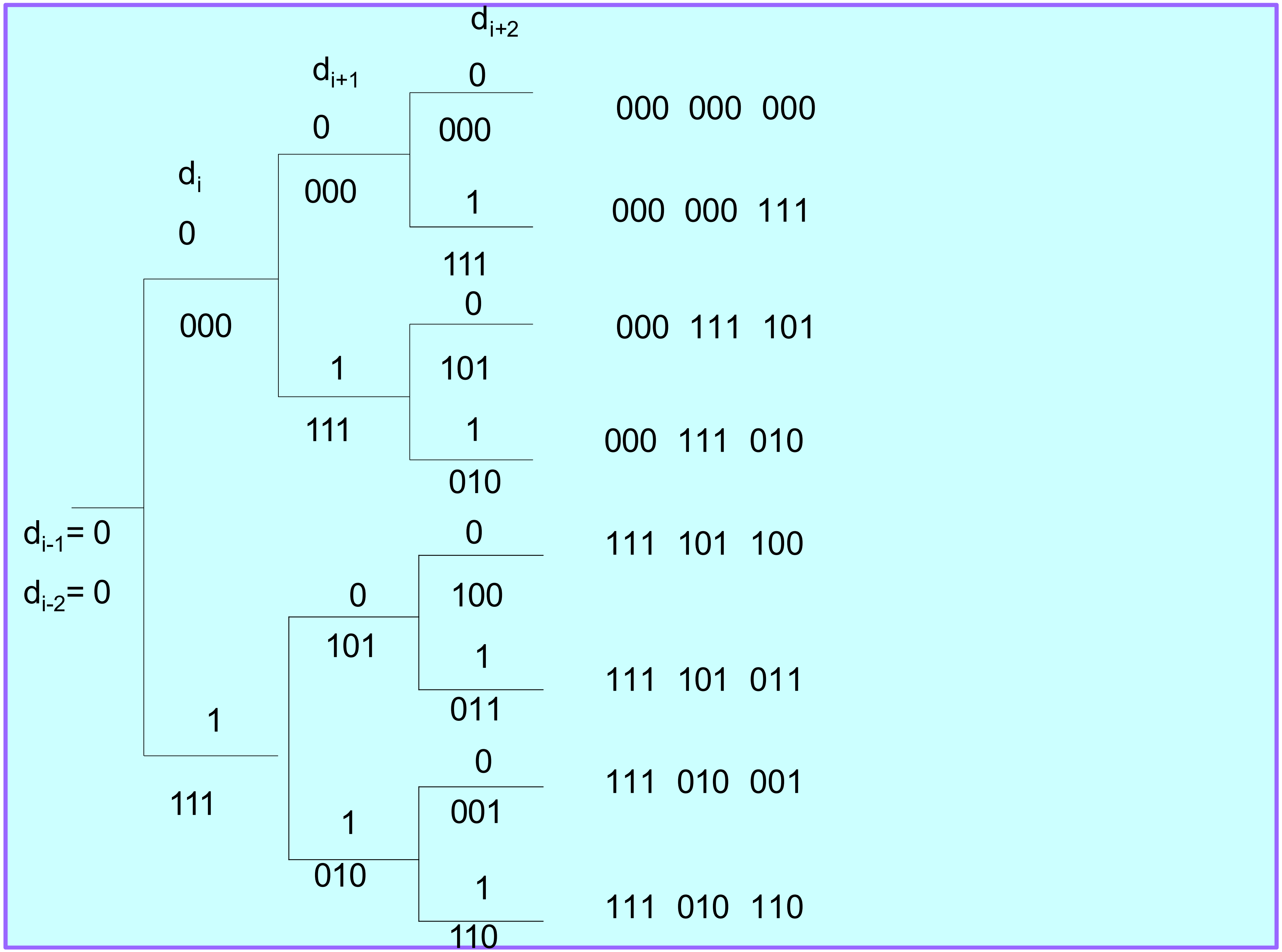


$\mathbf{C_1 = D_1 + D_2 + D_3}$

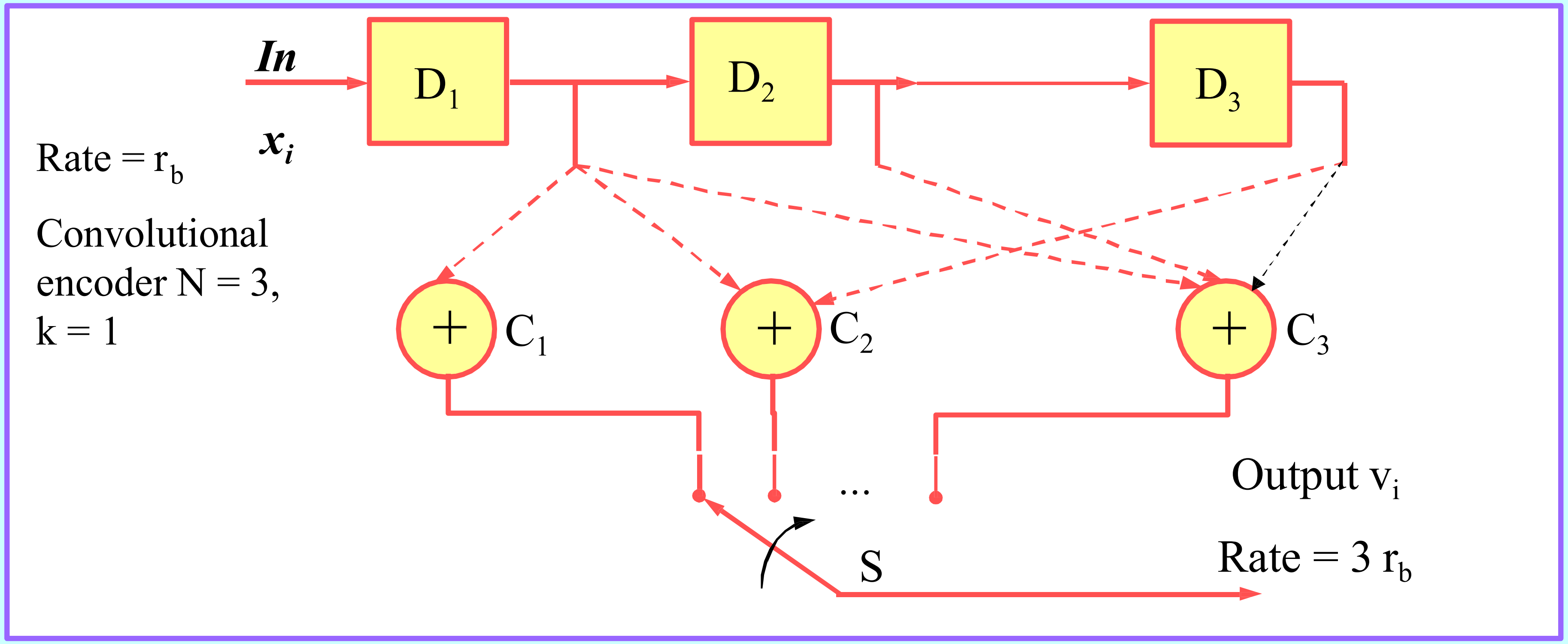
C2 = D1

C3 = D1 + D2

Code tree for the convolutional encoder is:


d_{i+2}
d_{i+1}
d_i
$d_{i-1} = 0$
$d_{i-2} = 0$
0
000
0
000
1
111
0
000
1
101
111
1
010
0
100
0
101
1
011
1
111
0
001
1
010
1
110
000 000 000
000 000 111
000 111 101
000 111 010
111 101 100
111 101 011
111 010 001
111 010 110

Example 2



The first function generator is : $G^1 = g^1_0 g^1_1 g^1_2 = [\ 1\ 0\ 0\ ]$,

the second function generator is : $G^2 = g^2_0 g^2_1 g^2_2 = [\ 1\ 0\ 1\ ]$,

and the third function generator is : $G^3 = g^3_0 g^3_1 g^3_2 = [\ 1\ 1\ 1\ ]$

In octal form (4, 5, 7).

$$v_i^1 = \sum_{n=0}^{2} x_{i-n} g_n^1$$

$$v_i^1 = x_0 g_0^1 + x_g g_2 g_1^1 + x_{1 \quad 2}$$

$$v_i^1 = x_i$$

$$v_i^2 = \sum_{n=0}^{2} x_{i-n} g_n^2$$

$$v_i^2 = x_i + x_{i-2}$$

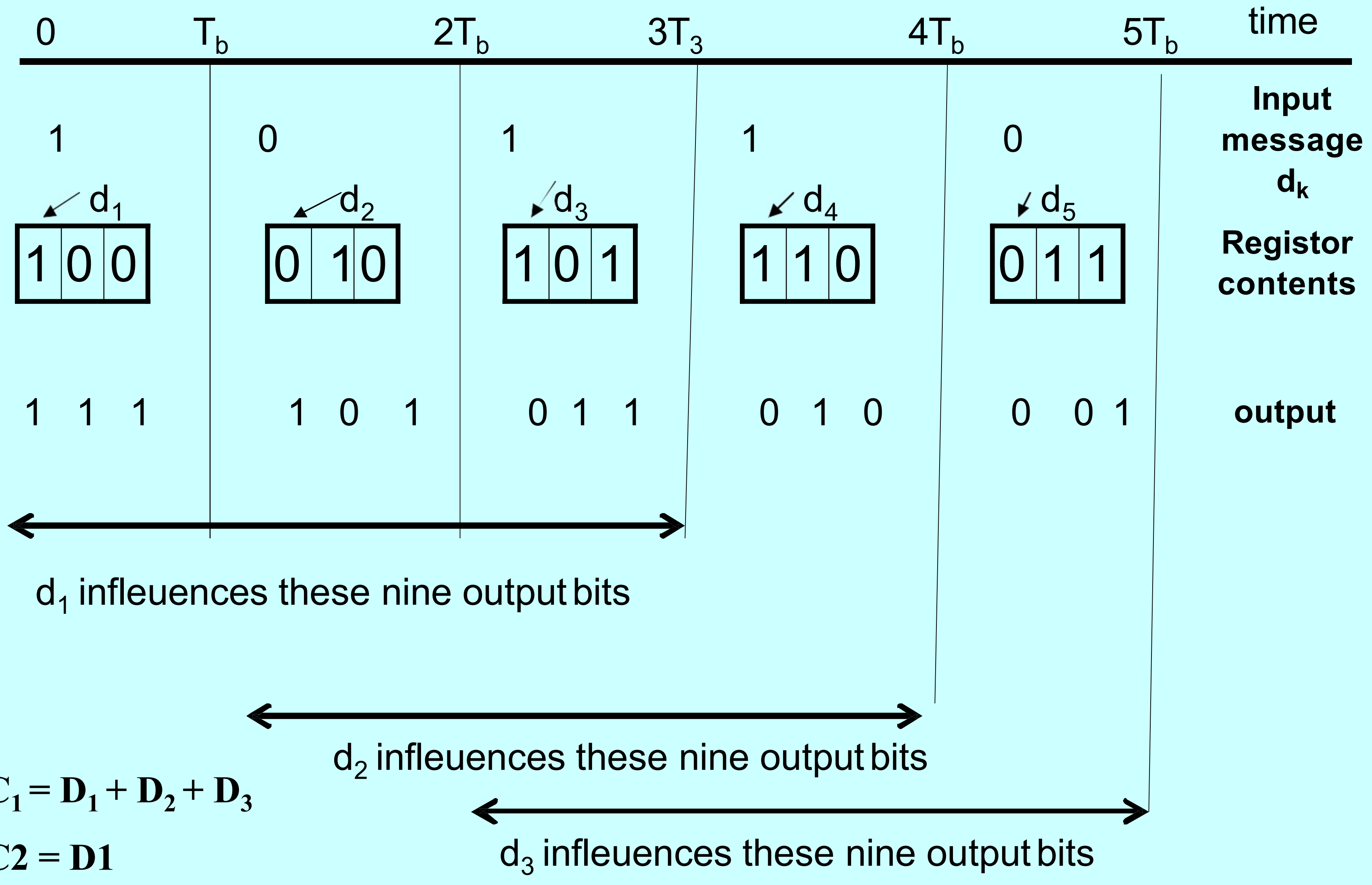
$$v_i^3 = \sum_{n=0}^{2} x_{i-n} g_n^3$$

$$v_i^3 = x_i + x_{i-1} + x_{i-2}$$

It is convenient to put generator polynomials in octal notation: all polynomial coefficients written in succession from left to right are treated as single binary number then converted into octal number system. For our example (g_1, g_2, g_3)= (7, 1, 3).

It can be easily checked that convolutional codes are **linear**. Further still their name itself is due to the fact that convolutional encoder is nothing more than binary linear finite impulse response (transversal) filter, i.e. device calculating **convolution** of the input vector with the filter coefficient vector.

One of the main reasons for a great popularity of convolutional codes in telecommunication is existence of elegant and resource-saving decoding algorithm (Viterbi algorithm) .



0
T_b
$2T_b$
$3T_3$
$4T_b$
$5T_b$
time
Input message d_k
1 0 1 1 0
d_1 d_2 d_3 d_4 d_5
1 0 0
0 1 0
1 0 1
1 1 0
0 1 1
Registor contents
1 1 1
1 0 1
0 1 1
0 1 0
0 0 1
output
d_1 infleuences these nine output bits
d_2 infleuences these nine output bits
d_3 infleuences these nine output bits

$C_1 = D_1 + D_2 + D_3$

C2 = D1

C3 = D1 + D2

Encoding operation of the convolutional encoder

State and Trellis Diagrams. Free Distance

Any convolutional encoder may be treated as an **automaton** or **finite-state machine**. Any specific state of this machine is simply a content of its memory. Depending on which of two values of the source bit is present currently at the input the encoder passes into one of two possible states in the next clock interval. In its turn, at the current clock interval the encoder can appear in some specific state only as a result of passing into it from one of two possible previous states. This binary character of transitions between states is due to the inputting to encoder only one bit at a time ($R = 1/n$). If, wishing to have rate $R = k/n$, we moved our coding window by k bits per clock interval, the number of allowed transitions from and to some fixed state would be 2^k.

This approach is embodied in the **state diagram** of the convolutional code depicting all states as nodes with arrows pointing all possible transitions from a current state to the next one. The way to make up such a diagram can be explained in terms of example.

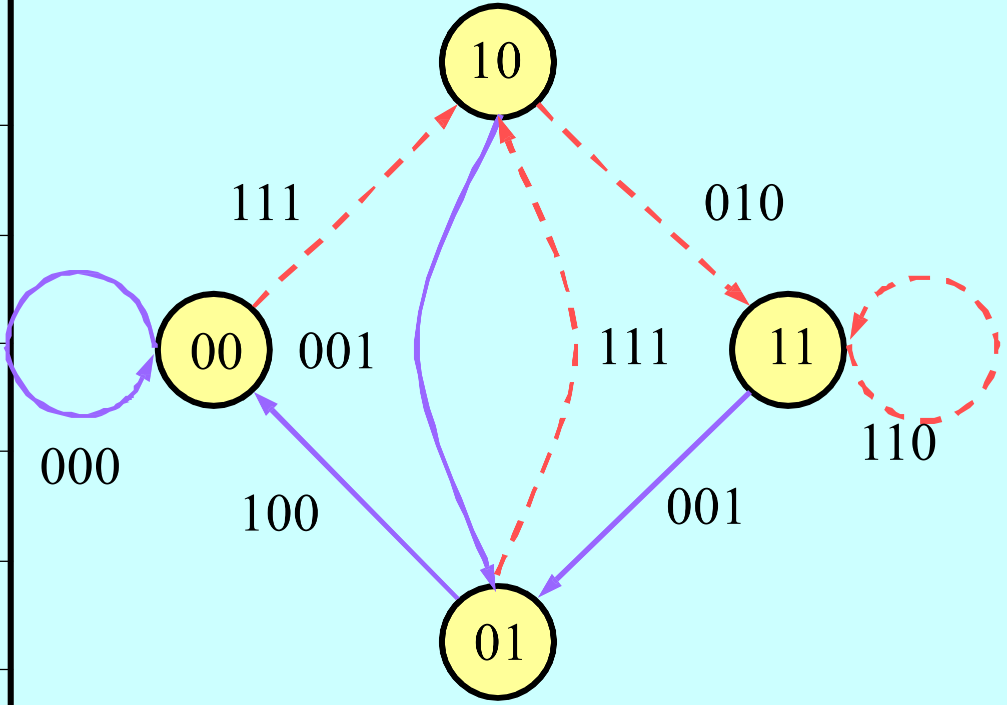
Example 2. Come back to the encoder of Example 1. States of this automaton are 00, 10, 01, 11 (first symbol is a state of the leftmost cell). It is clear that from the state 00 the encoder can only move into the next state 00 if the input bit is zero or into 10 with the current input bit 1. The same way the rest of allowed transitions are obvious which are depicted in the state diagram where nodes are shown as circles with states denoted inside. Blue solid arrows show transitions caused by input bit 0, while red dashed arrows are assigned to transitions caused by input bit 1. It is readily seen that two paths leave each state and two paths enter it. As for output code symbols, they are shown by labeling branches. Say, when the encoder is in state 00 and input bit is 1 output symbols are 11. Therefore branch corresponding to the transition 00$\rightarrow$10 is labeled as 11, etc.

Continuing with this example rechart the state diagram as shown below.
We only illustrate graphically the same idea for two successive clock intervals. Again current state is changed into the other one depending on current input bit. But this version of the state diagram gives way to the pattern demonstrating all the dynamics of the encoding process. Start with 00 encoder state. After the first clock

State table

Old state	input	Next state	output
00	0	00	000
00	1	10	111
10	0	01	011
10	1	11	010
01	0	00	100
01	1	10	111
11	0	01	011
11	1	11	110

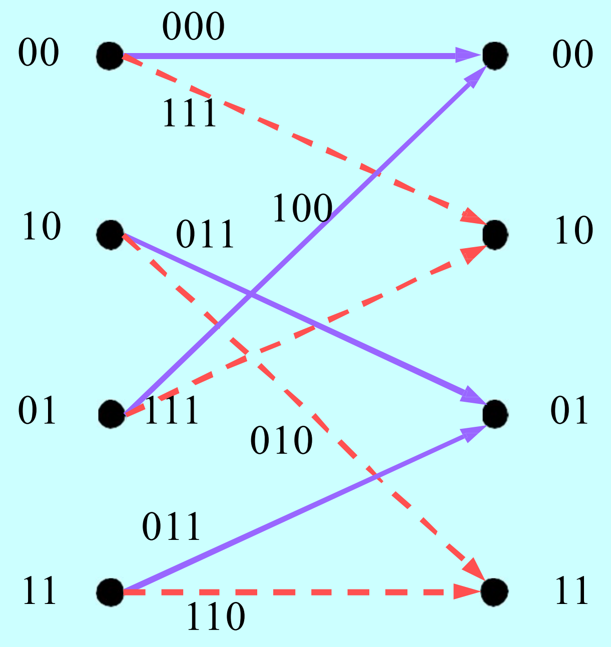
State diagram



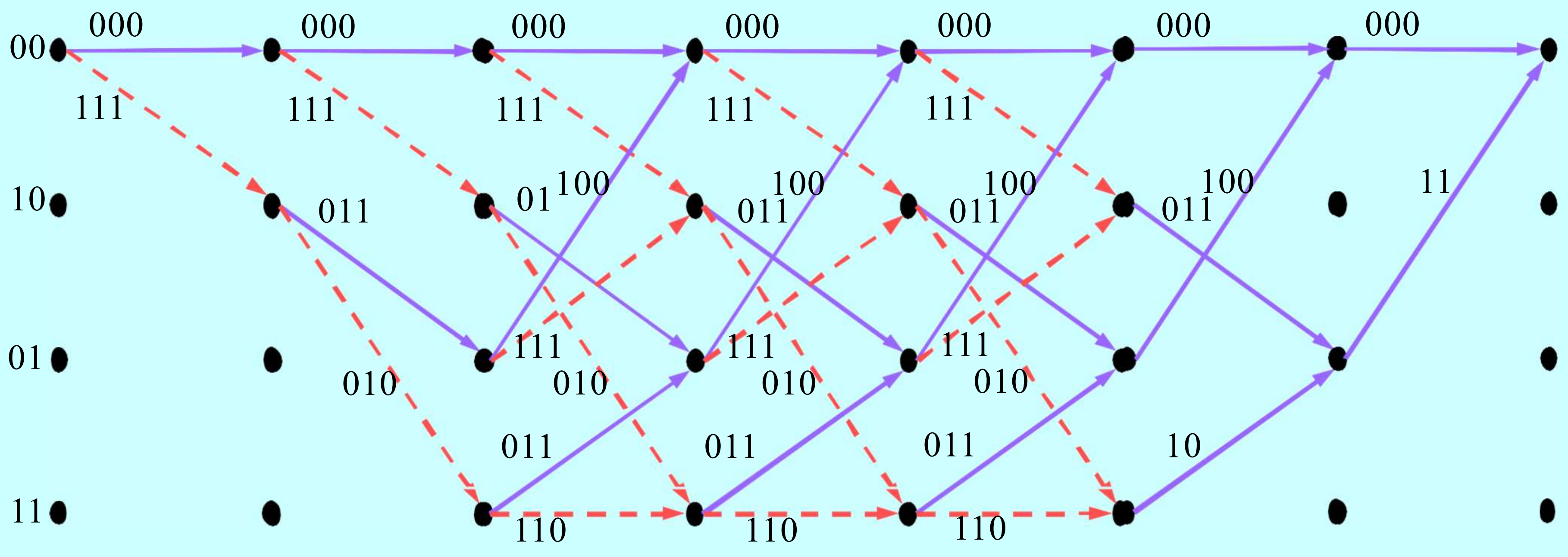
$C_1 = D_1 + D_2 + D_3$

$C2 = D1$

$C3 = D1 + D2$



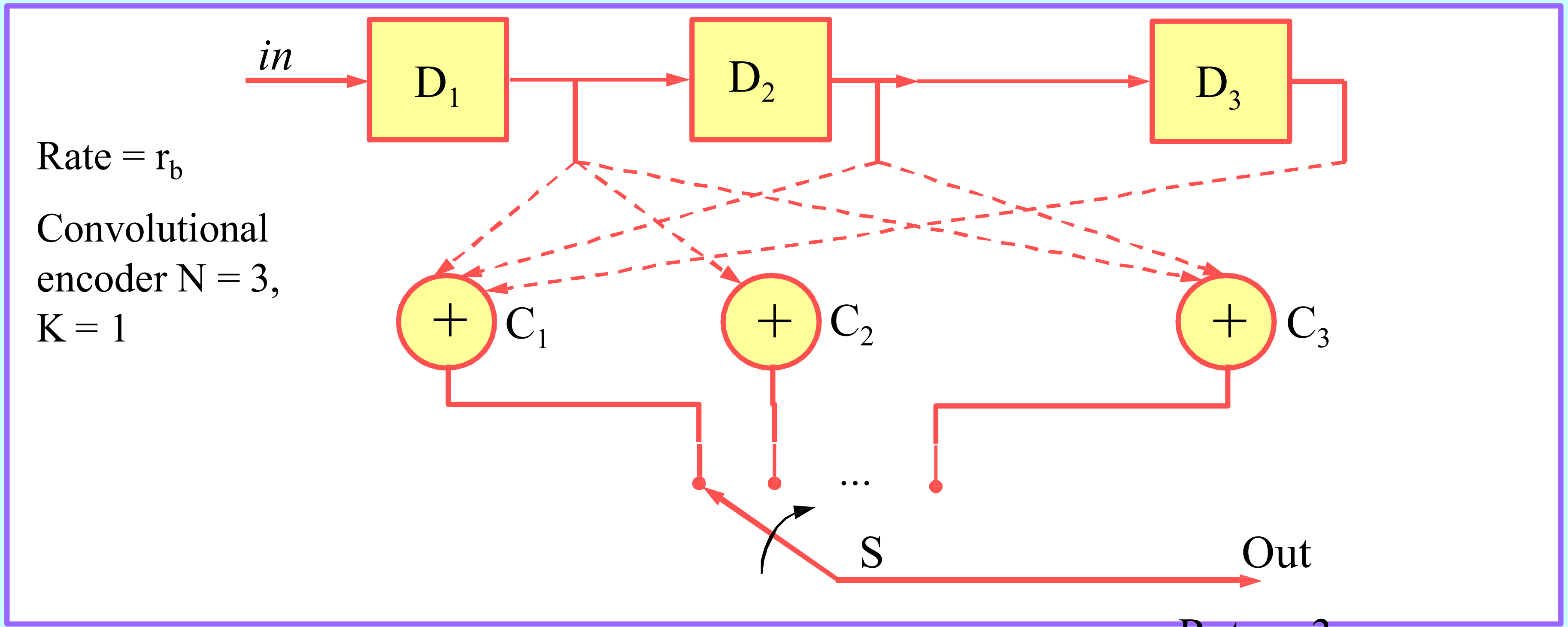
interval decoder comes to states either 00 or 10. If it came to 00 then after the second clock interval it again comes to either 00 or 10. But if its state after the first clocking is 10 it moves into 01 or 11. Starting the third clock interval in states 00 or 10 it behaves like earlier but from state 01 it comes to either 00 or 10 and from state 11 it enters states either 01 or 11. After the third interval behavior of encoder becomes steady-state until input bit stream stops. When this



happens N-1 clock intervals are spent to clear register during which time output code symbols are still generated.

The pattern thus obtained is called **trellis diagram**. All paths on it are nothing more or less than different **codewords** of the convolutional code. Say, source bits 01010 are encoded into the word 00110100011100. To figure out error correction capability minimum Hamming distance between paths should be found. For the convolutional codes is widely accepted to call minimum distance **free distance**. Because of the code linearity free distance is simply minimum weight of the “nonzero” (that is not the uppermost on the trellis diagram) path. Unlike block codes free distance for the convolutional codes can be found rather easily via state diagram which is considered in the following section.

Decoder for Convolutional Codes

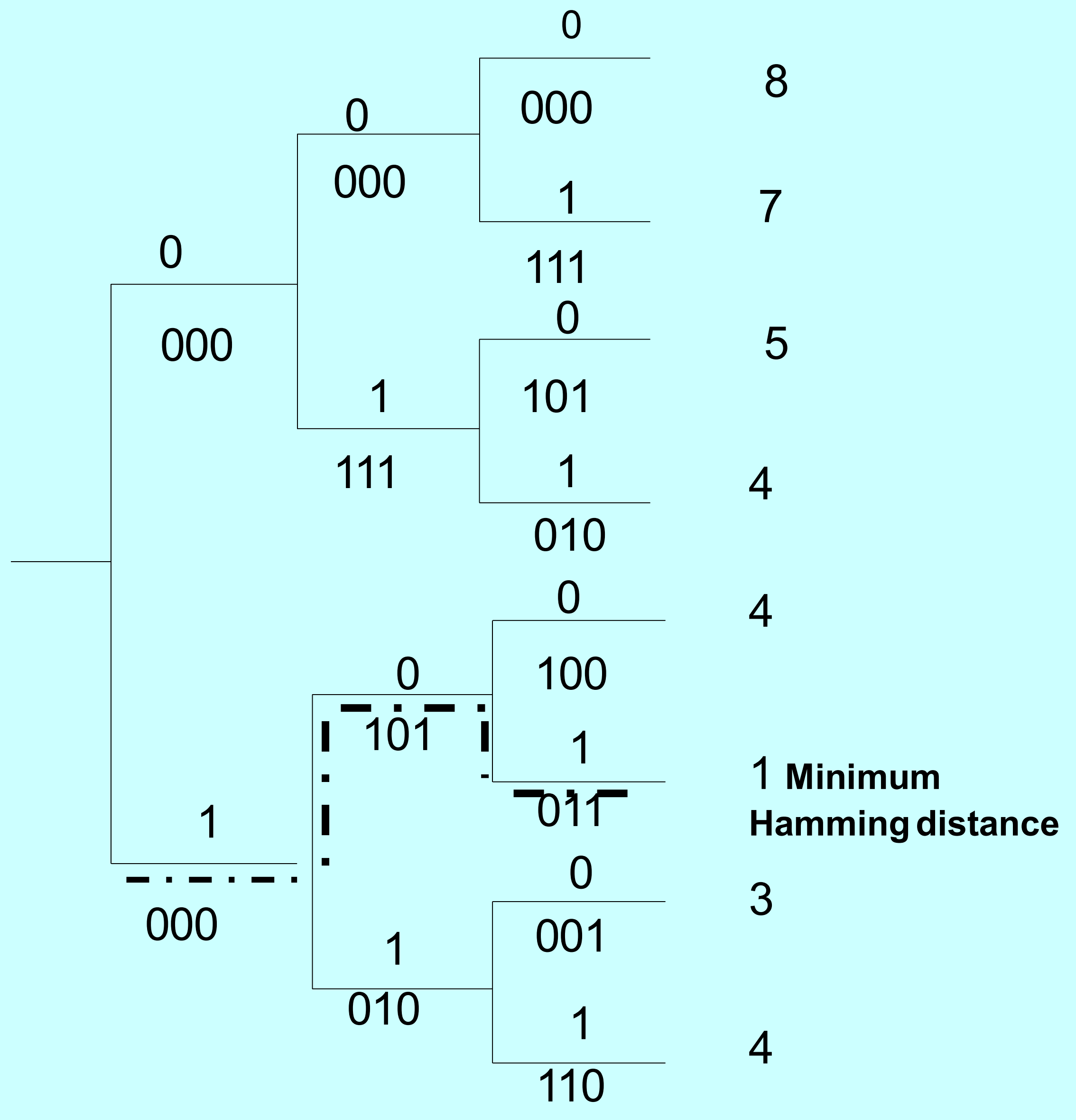


Rate = r_b

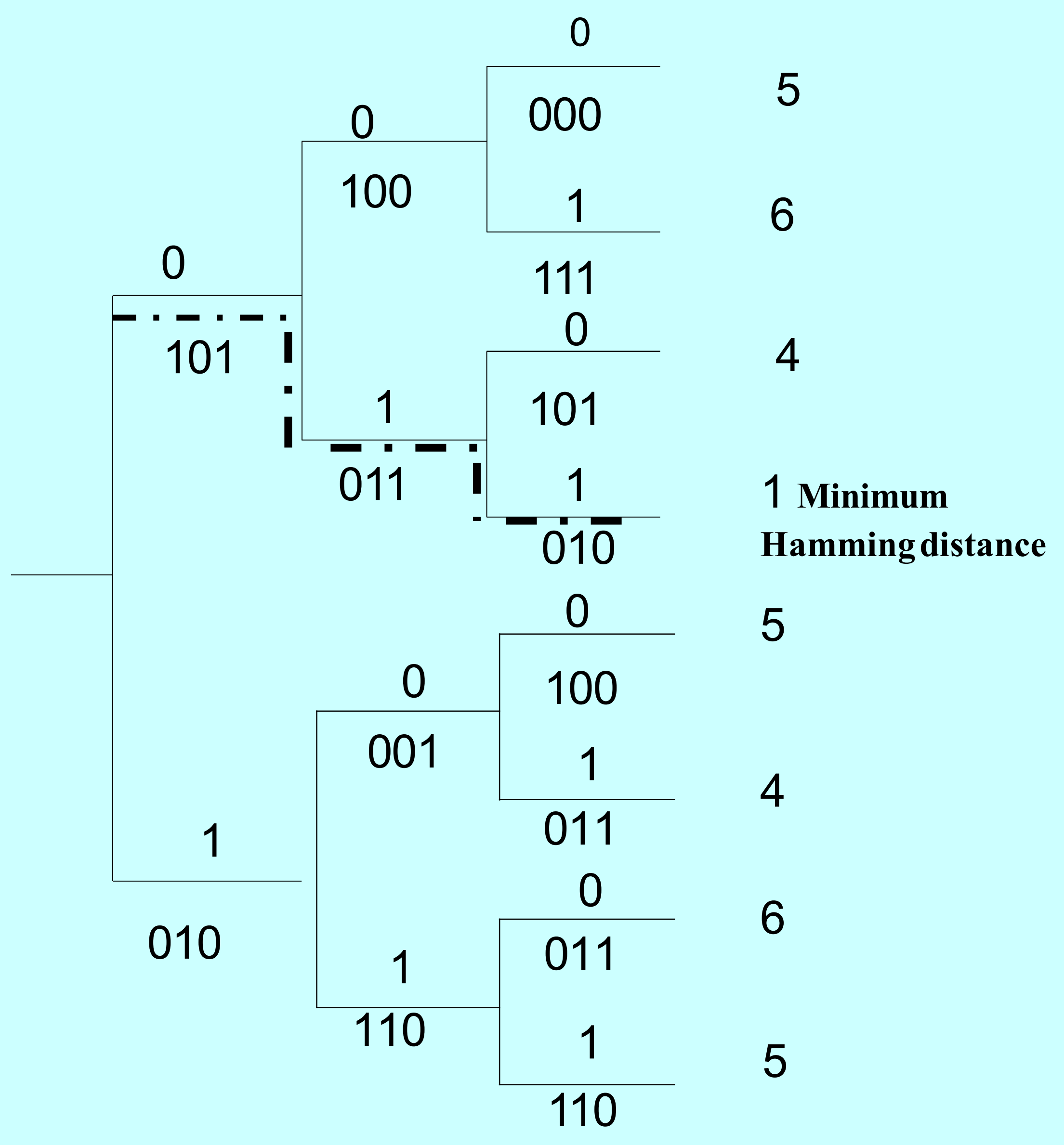
Convolutional encoder N = 3, K = 1

Rate = 3 r_b

1	0	1	1	**Input data** **d_k**
1 1 1	1 0 1	0 1 1	0 1 0	**Coded word** **C**
1 1 1	1 1 1	0 1 1	0 1 0	**Received word** **R**
	error			

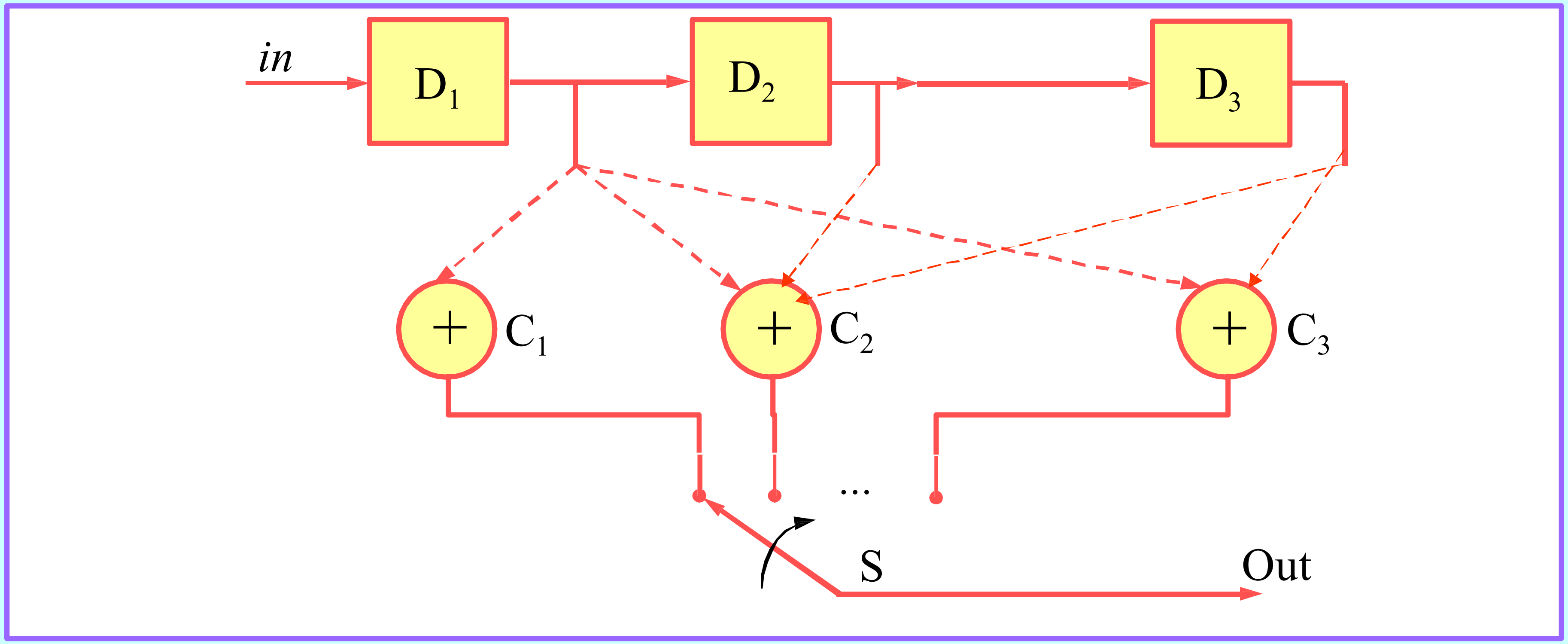


From tree $d_1 = 1$



From tree $d_2 = 0$

Homework: Find the state diagram for the encoder shown in below.


in
D_1
D_2
D_3
C_1
C_2
C_3
...
S
Out