

# Information Theory & Coding - EEE2004

---

Dr. S. Le Goff  
School of EECE @ Newcastle University  
Spring 2012

## Recommended books

- Error-correction coding - Mathematical methods and algorithms, T.K. Moon, John Wiley & Sons, 2005.
- Applied coding & information theory for engineers, R.B. Wells, Prentice-Hall, 1999.
- Digital communications, 2<sup>nd</sup> edition, I.A. Glover & P.M. Grant, Prentice-Hall, 2004.
- Error control coding, Shu Lin & D.J. Costello, Jr., Prentice-Hall, 2004.

# Lecture notes

If you want a PDF version of these notes, please send me an email or go to blackboard (tell me in case you do not find the latest set of notes there).

If you find any typo/mistake, please let me know.

I also welcome negative comments as long as they are not too destructive...

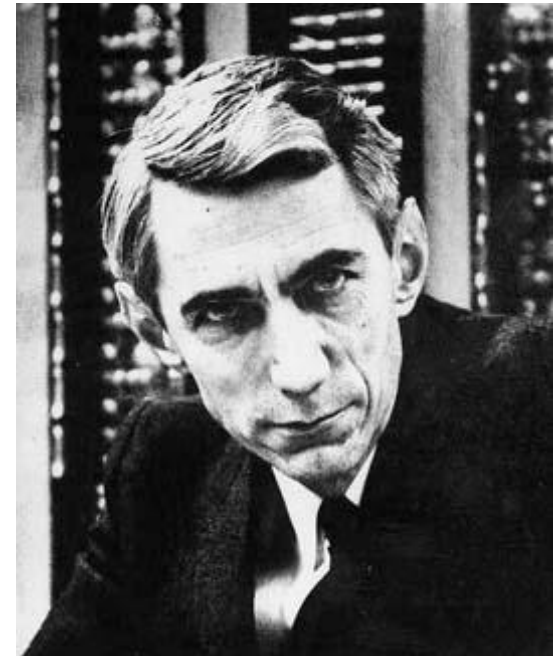
# Part 1

# Introduction

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point"

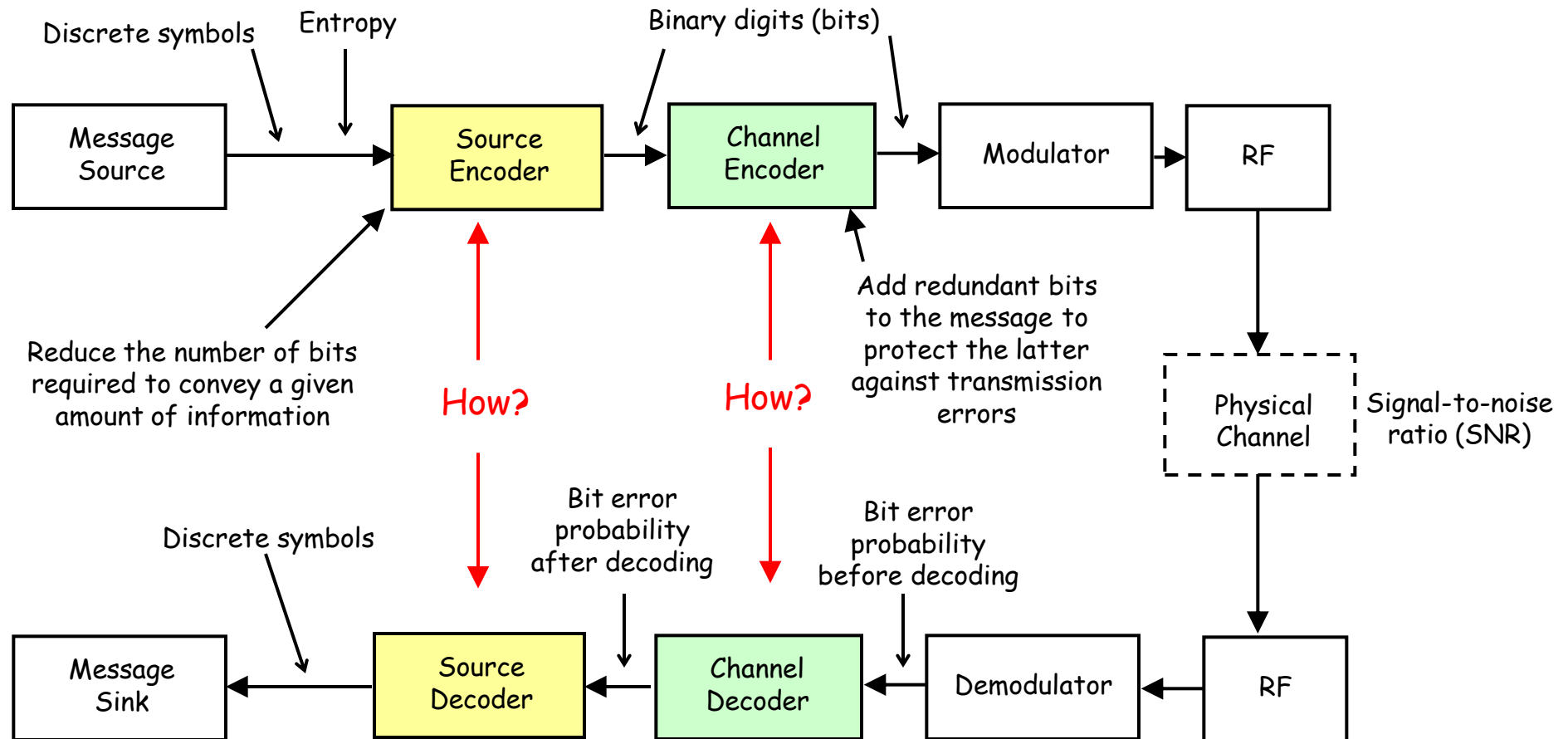
Claude E. Shannon, 1948.

To solve this problem,  
Shannon created a branch of  
applied mathematics:  
Information Theory.

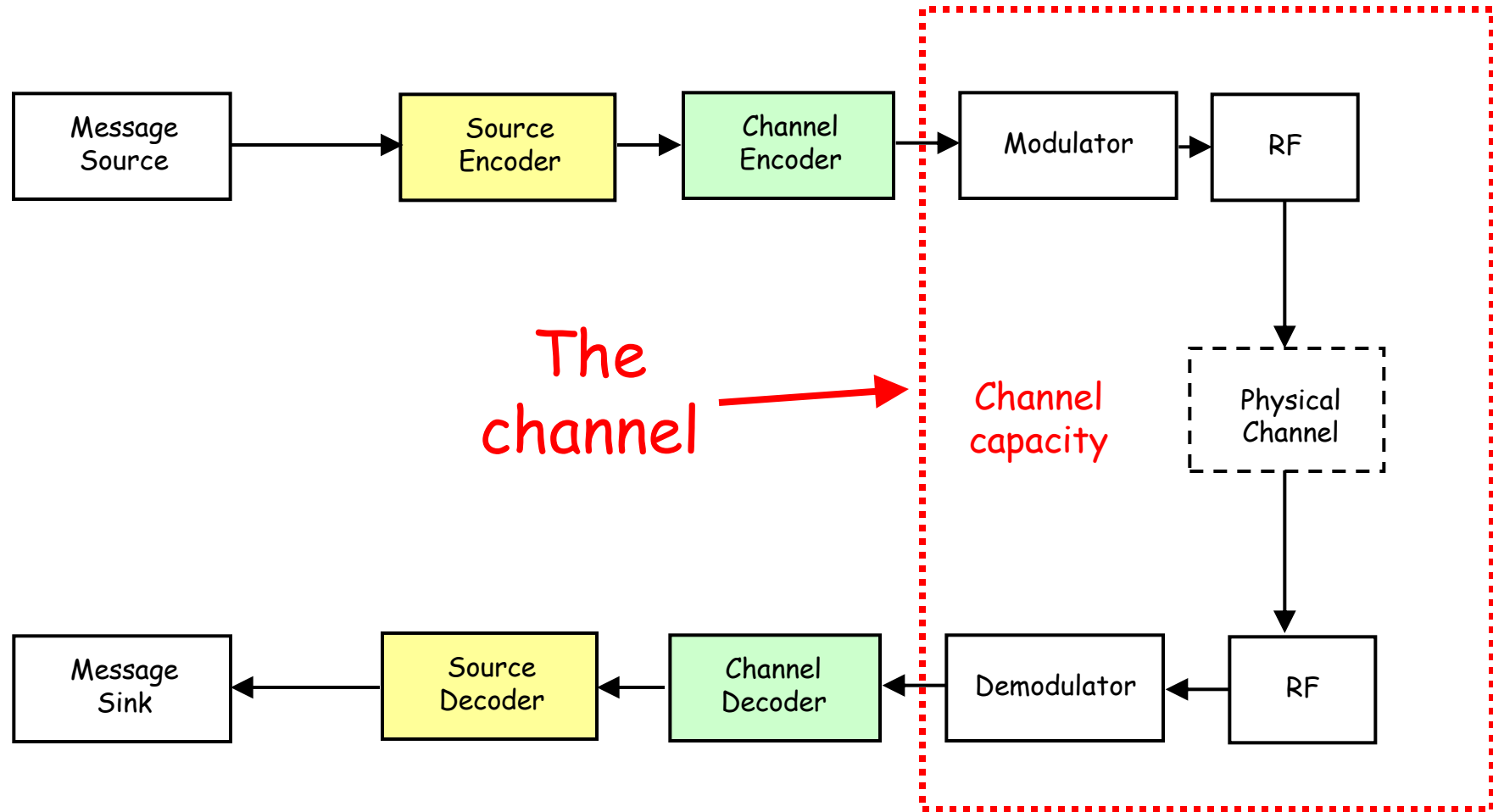


Claude E. Shannon (1916-2001)

# Digital communication systems



# Digital communication systems for information theorists



# Part 2

## Overview of Source Coding Techniques



# Source coding

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

Ex: "Let's go to the beach this aftertoon!"  
What did I say?

What about pictures and video signals?



# Source coding

Ex: Video signal, 25 pictures/sec,  $256 \times 256$  pixels/picture, 8 bits/pixel

→ Data rate  $\approx 13$  Mbits/sec. It's too much! ☹

Ex: Movie, 3 hours

→  $\approx 140$  Gbits to download and store. Yet, we have not included the sound and other features... ☹

=> We need to compress!

# Source coding

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

Two techniques:

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

Q: Which one do we use?

A: It depends on the application.

# Lossless compression techniques

Compression algorithms allowing the exact original data to be reconstructed from the compressed data.

Used in software compression tools such as the popular ZIP file format.

Used when the original and the decompressed data must be identical → A must for executable programs and source code.

# Lossless compression techniques - Run length

## Ex 1: Run length encoding

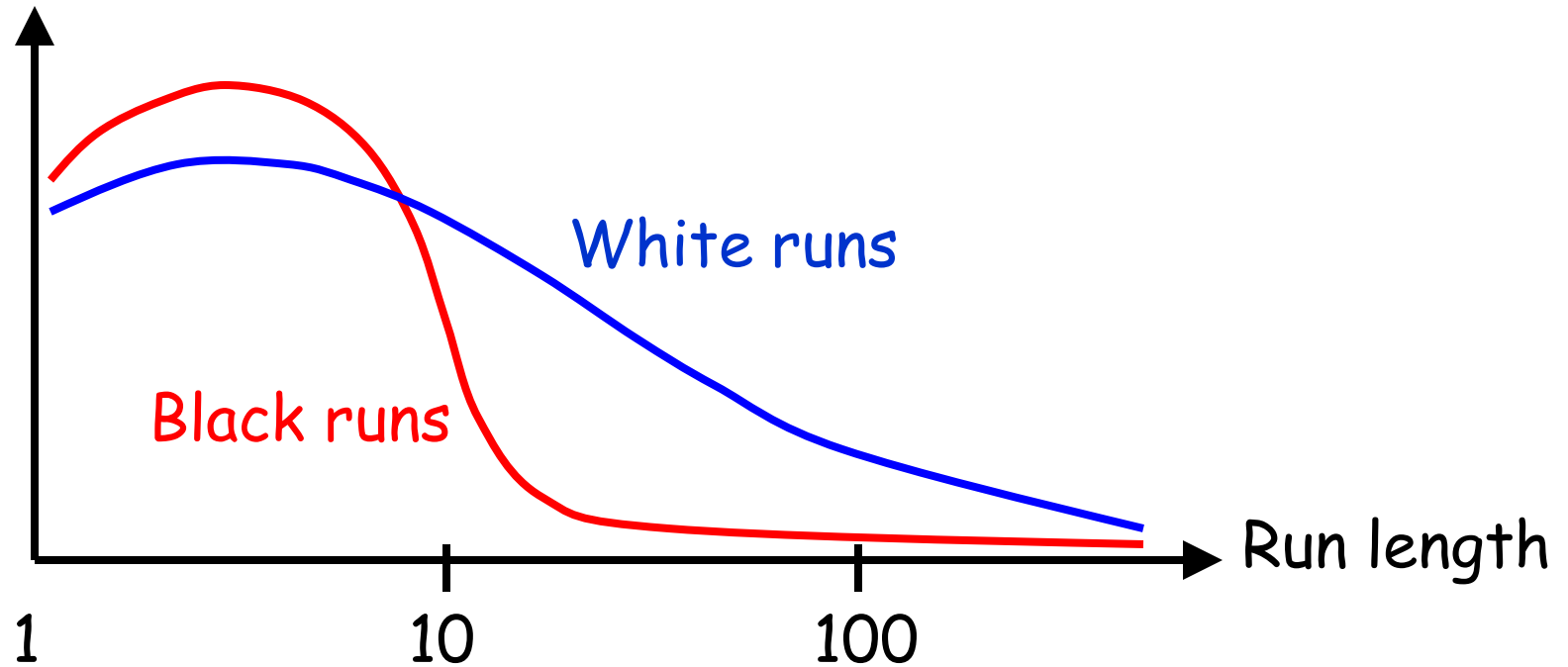
FAX: 1728 pixels per scan line, 3.85 scan lines/mm, and page length of 30 cm  $\rightarrow \approx 2$  million pixels/page!

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

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

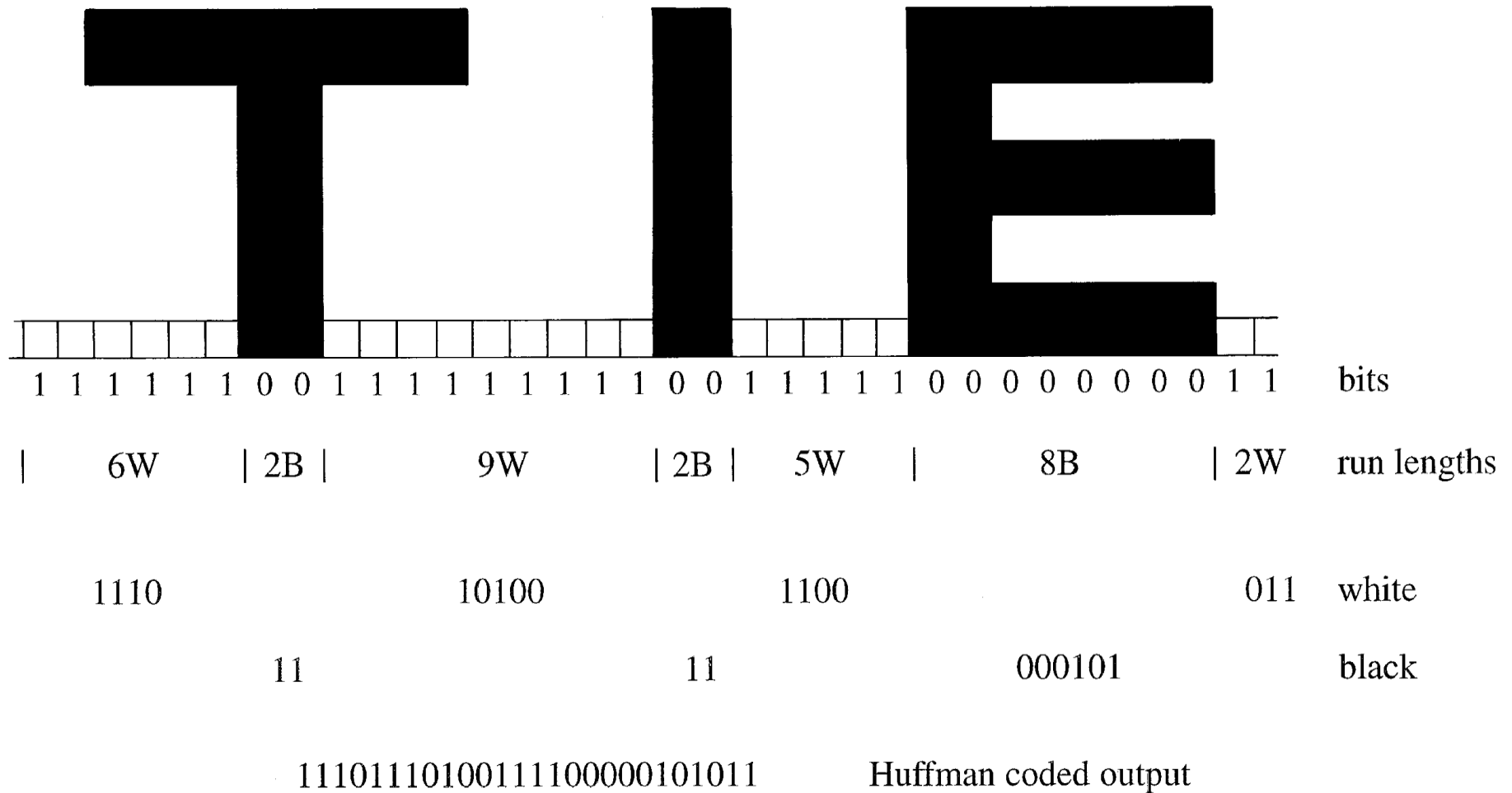
# Lossless compression techniques - Run length

Probability of occurrence



The black letters are generally not more than 10 pixels wide while large white areas are much more common.

# Lossless compression techniques - Run length



# Lossless compression techniques - Run length

We do not achieve significant compression in this case (34 bits reduced to 26), but in practice the results are more impressive.

If you insert a document in a FAX machine, it will quickly pass through the white areas of a page but slow down on dense text or images.

An average page has a compression factor of around 7 and therefore takes around 5 seconds to pass through the FAX machine.

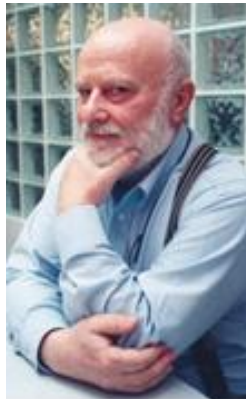


# Lossless compression techniques - Lempel-Ziv

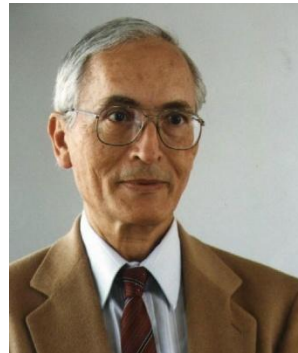
## Ex 2: Lempel-Ziv algorithm

The LZ (Abraham Lempel<sup>1</sup>, Jacob Ziv<sup>2</sup>, 1977) algorithm is a statistical modelling algorithm for text.

1



2



# Lossless compression techniques - Lempel-Ziv

LZ algorithm uses the fact that small strings within messages often repeat locally, i.e. the source has memory.

LZ method uses a history buffer and looks for matches. If a match is found, then the position and length of the match within the buffer are transmitted instead of the character string.

# Lossless compression techniques - Lempel-Ziv

Assume we want to transmit the string "the\_mat".

The history buffer of length 16 (already sent) is as follows: "\_the\_cat\_sat\_on\_".

We found a match for "the\_" in the history buffer.

So, encoding "the\_" gives: 1 to indicate that a match has been found, 0001 to indicate that the match starts at position 1, and 011 to indicate that the length of the match is 4 (max length is 8) -> 8 bits.

# Lossless compression techniques - Lempel-Ziv

Now we need to encode the string "mat".

The history buffer is now as follows:  
"\_cat\_sat\_on\_the\_".

There is no match for "m". So, encoding "m" gives: 0 to indicate that no match has been found, and the 8-bit ASCII code for "m" -> 9 bits.

Then we need to encode the string "at", using the history buffer "cat\_sat\_on\_the\_m".

# Lossless compression techniques - Lempel-Ziv

We found a match for "at" in the history buffer.

So, encoding "at" gives: 1 to indicate that a match has been found, 0001 to indicate that the match starts at position 1, and 001 to indicate that the length of the match is 2 characters → 8 bits.

Conclusion: We have used 25 bits to encode the string "the\_mat". Without encoding, we would have needed 7 times 8 bits, i.e. 56 bits → Compression ratio =  $56/25:1 \approx 2.24:1$ .

# Lossless compression techniques - Lempel-Ziv

LZ compression technique is efficient and can adapt to changes in data statistics.

However, encoding time is long as searching is needed to identify the matches and it does not work effectively for short bursts of data.

Technique used for compression of computer files for transport (ZIP), for saving space on hard disks and in some modem standards.

# Lossy compression techniques

In lossy data compression, decompression generates data which is different from the compressed one, but still "close enough" to be useful in some way.

Lossy methods are used for compressing sound, images or videos. For these applications, a lossy method can produce a much smaller compressed file than any lossless method, while still meeting the requirements of the application.

# Lossy transform coding - Still images

## Ex 1: Still image compression

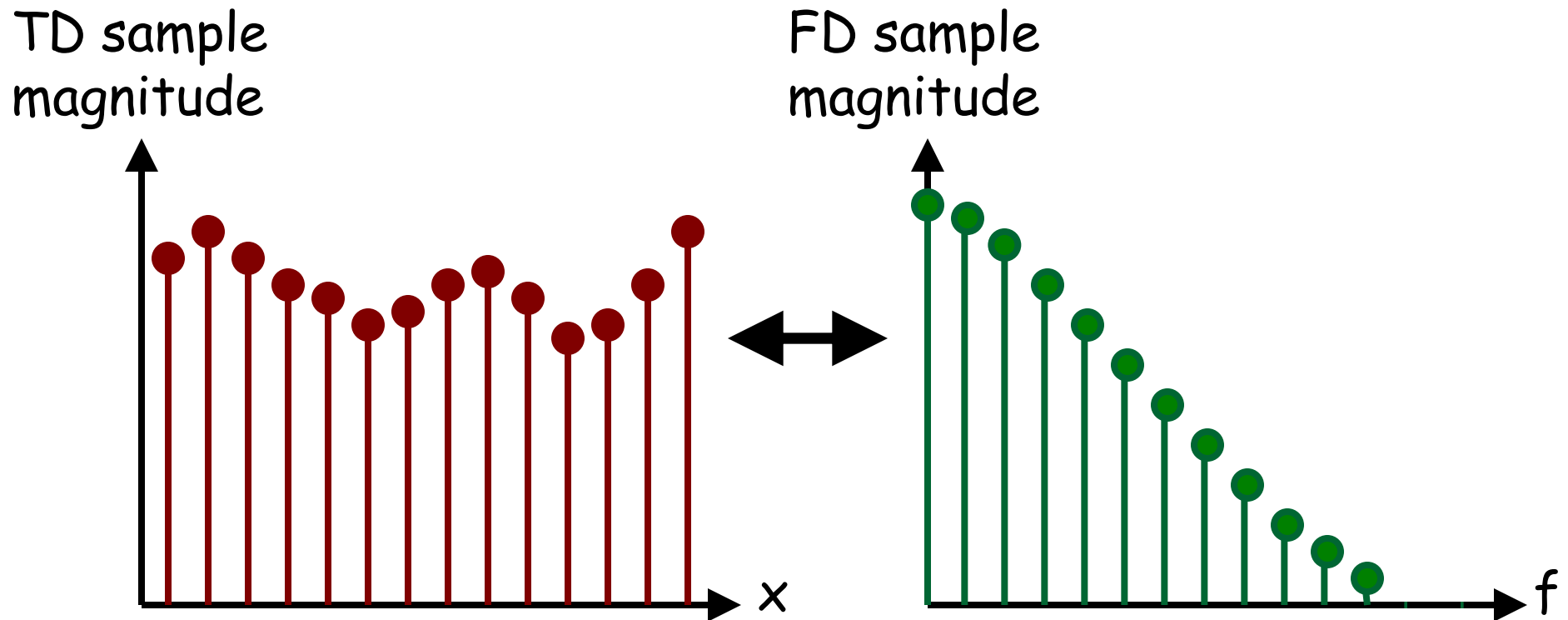
Transform coding attempts to transform the input samples of the image from one domain to another.

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



# Lossy transform coding - Still images

Equivalent to a "time domain  $\leftrightarrow$  frequency domain" operation in signal processing.



# Lossy transform coding - Still images

Spatial signal (image)  $\leftrightarrow$  spatial frequency spectrum.

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

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

# Lossy transform coding - Still images

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

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

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

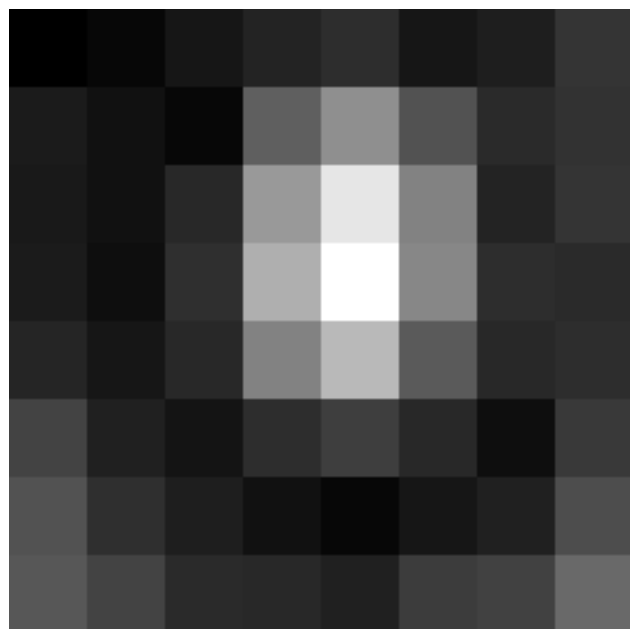
# JPEG standard

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

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

# JPEG standard

A block of 64 pixels with 256 grey scale levels per pixel.



-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-73	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

# JPEG standard

Taking the DCT of this block of pixels, we obtain:

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

The large value of the top-left corner is the DC coefficient. The remaining 63 coefficients are called the AC coefficients.

# JPEG standard

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

This is the main lossy operation in the whole process.

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

# JPEG standard

## DCT output

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

## Quantization matrix

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$



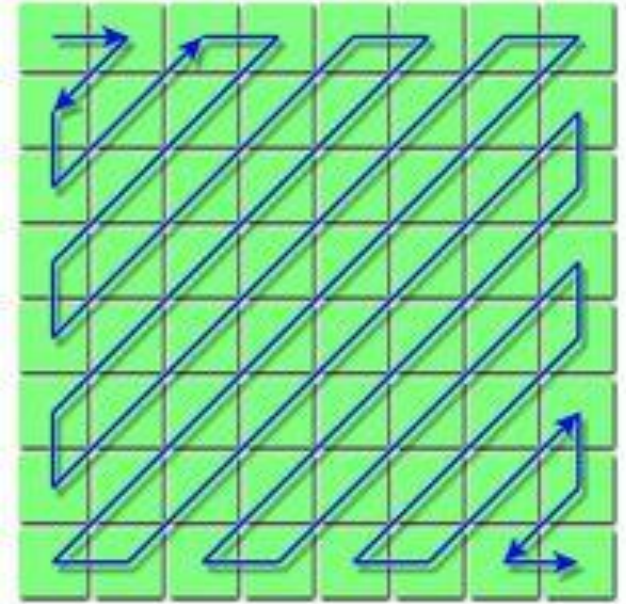
$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

DCT output after  
quantization



# JPEG standard

The quantized DCT output coefficients are arranged in a "zig-zag" order, and then encoded using run length and Huffman coding (lossless techniques).



Here, the "zig-zag" sequence is: -26, -3, 0, -3, -2,  
-6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0,  
-1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.

# JPEG standard

The compression ratio can be varied by being more or less aggressive in the divisors used in the quantization phase.

A compression ratio of 10:1 compression usually results in an image that cannot be distinguished by eye from the original.

A compression ratio of 100:1 is usually possible, but the resulting image looks clearly degraded compared to the original.

# JPEG standard

"Lena" image



Original picture size = 12,249 bytes.



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



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

# JPEG standard

In recent years, wavelet-based encoders have gained popularity over the DCT-based systems => JPEG 2000.

Comparison of  
JPEG 2000 with  
the original JPEG  
format (?)



Uncompressed  
378 KiB  
1:1

JPEG JFIF  
11.2 KiB  
1:33.65  
IJG q 30

JPEG 2000  
11.2 KiB  
1:33.65



# Lossy predictive coding

## Ex 2: Predictive coding

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

Frame (i)



Frame (i+1)



# Lossy predictive coding

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

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

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

# Part 3

## Introduction to Channel Coding

# Channel coding (error-correcting coding)

Somewhat the opposite of source coding in the sense that it adds redundancy to the message to be transmitted (or stored).

=> Results in a loss of data rate for a given bandwidth OR an increase of bandwidth for a given data rate ☹️

The “controlled” redundancy allows for detection or correction of transmission errors at the receiver side 😊

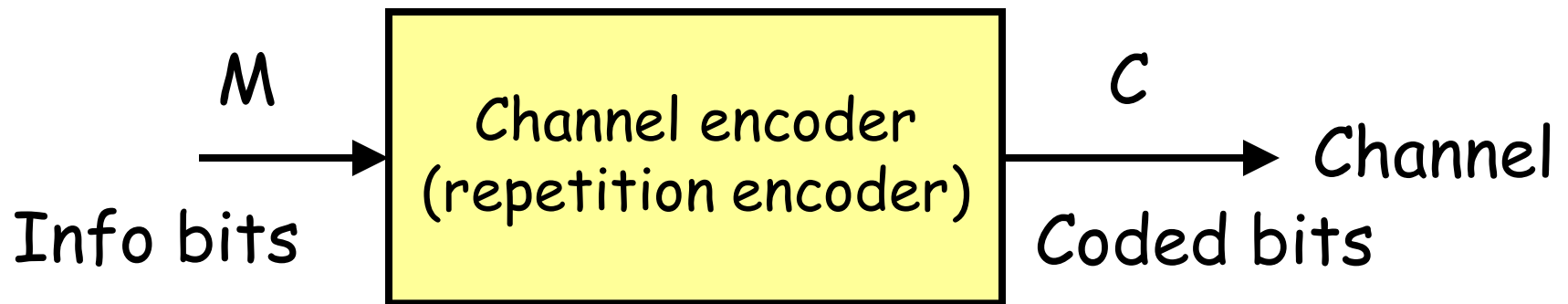


## An example: Rate-1/3 repetition code

The info bit stream to be transmitted is encoded as follows:

$$M = 0 \rightarrow C = 000 \text{ and } M = 1 \rightarrow C = 111.$$

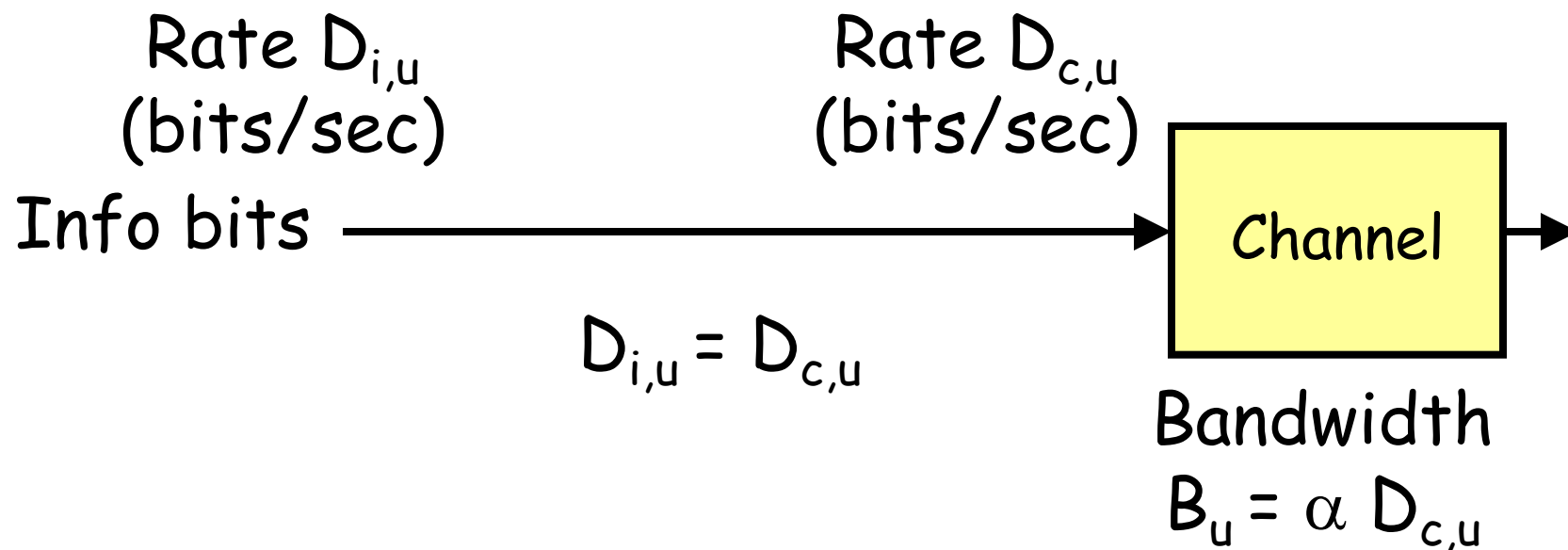
$M$  is the "message" and  $C$  is the "codeword". The rate of such code is  $R_c = 1/3$ .



# An example: Rate-1/3 repetition code

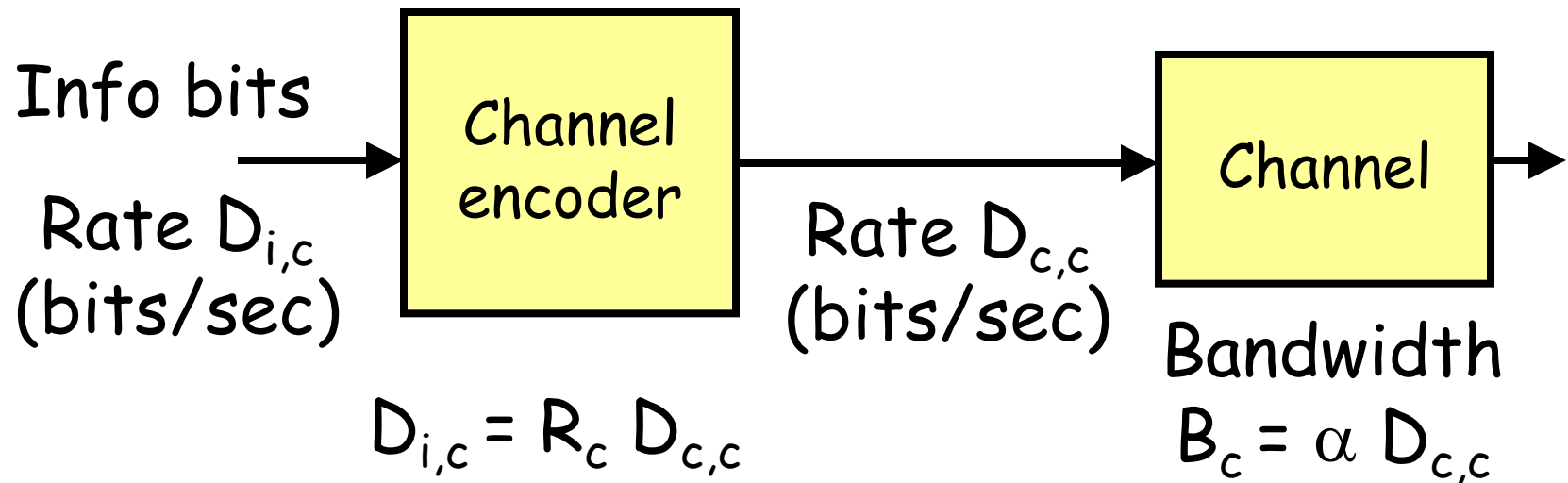
Consider first an uncoded system.

Info bits  $\longrightarrow$  Channel



## An example: Rate-1/3 repetition code

Consider now a coded system with  $R_c = 1/3$ .



## An example: Rate-1/3 repetition code

If the bandwidth is to be kept constant, i.e.  $B_c = B_u$ , then the info bit rate must be decreased three-fold.

$$\begin{array}{ccc} B_c = B_u & \longrightarrow & D_{c,c} = D_{c,u} \\ & & \downarrow \\ D_{i,c} = R_c D_{i,u} & \longleftarrow & D_{i,c}/R_c = D_{i,u} \\ & & D_{i,c} < D_{i,u} \text{ ☹️} \end{array}$$

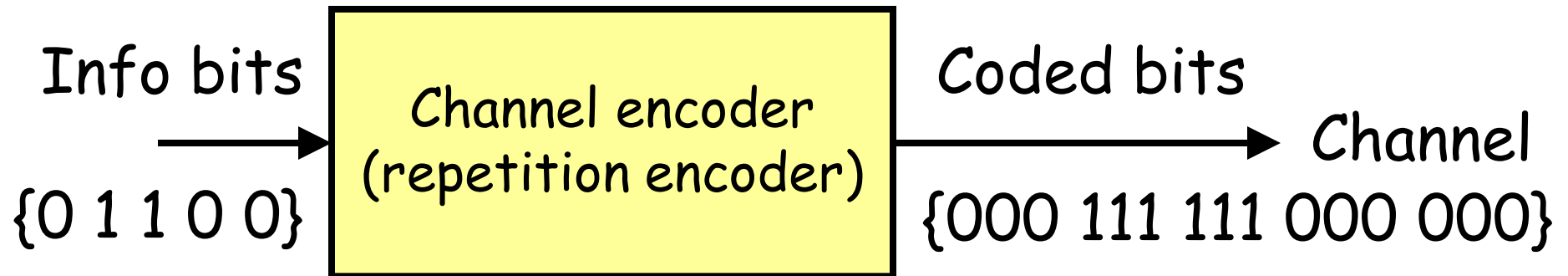
## An example: Rate-1/3 repetition code

If the info bit rate is to be kept constant, i.e.  $D_{i,c} = D_{i,u}$ , then the bandwidth must be increased three-fold.

$$\begin{array}{ccc} D_{i,c} = D_{i,u} & \longrightarrow & R_c D_{c,c} = D_{c,u} \\ & & \downarrow \\ B_c = B_u / R_c & \longleftarrow & D_{c,c} = D_{c,u} / R_c \\ & & B_c > B_u \text{ ☹️} \end{array}$$

## An example: Rate-1/3 repetition code

Ex: Sequence of info bits = {0 1 1 0 0}  
=> Coded sequence = {000 111 111 000 000}.



## An example: Rate-1/3 repetition code

Assume the corresponding received sequence is {000 101 011 010 011}. Find the decoded sequence.

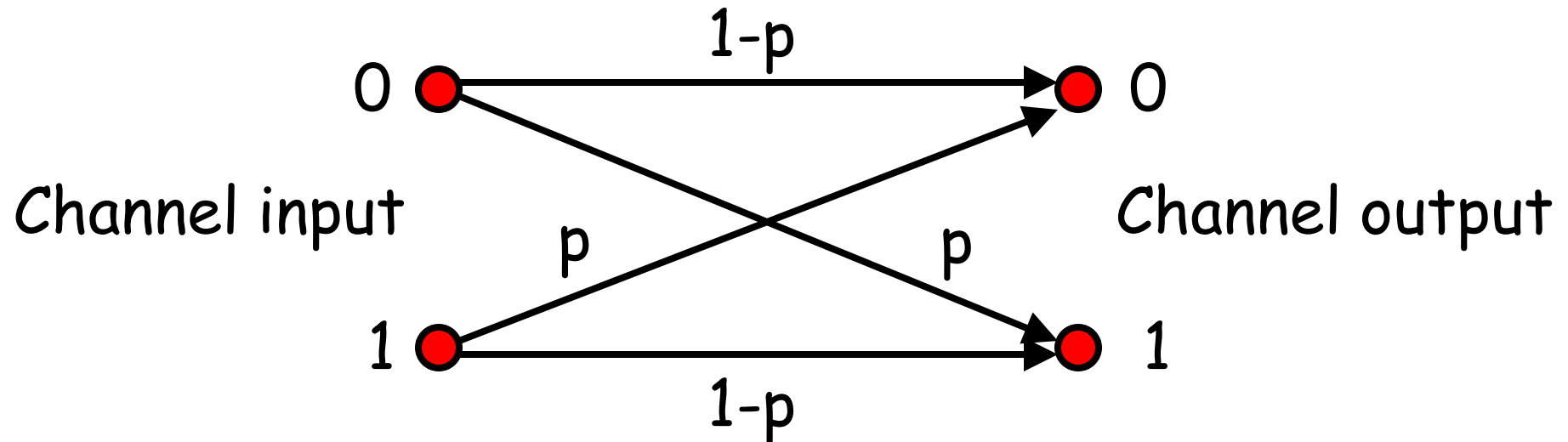


Answer:

## An example: Rate-1/3 repetition code

We want to express the bit error probability *after* decoding ( $P_{eb}$ ) as a function of the bit error probability *before* decoding ( $p$ ).

Assume a Binary Symmetric Channel (BSC):



$p$  is the error probability over the channel ( $p < 0.5$ ).



## An example: Rate-1/3 repetition code

Probability of wrong detection of the transmitted codeword:

$$P_{wd} = \Pr\{ 2 \text{ or } 3 \text{ errors in a received word of 3 bits} \}$$

$$P_{wd} = \binom{3}{2} p^2 (1-p) + \binom{3}{3} p^3 \quad P_{wd} = \frac{3!}{2!1!} p^2 (1-p) + \frac{3!}{3!0!} p^3$$

Binomial Coefficient:  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$  If  $p \ll 1$ :  $P_{wd} \approx 3p^2$

We need to compute the bit error probability *after* decoding, not  $P_{wd}$ .

## An example: Rate-1/3 repetition code

For a repetition code, we are fortunate since  $P_{eb} = P_{wd}$ , but it is not the case for other codes. In general,  $P_{eb} < P_{wd}$ .

$$\rightarrow \text{If } p \ll 1: P_{eb} \approx 3p^2$$

$$p = 10^{-2} \Rightarrow P_{eb} \approx 3 \times 10^{-4}$$

$$p = 10^{-3} \Rightarrow P_{eb} \approx 3 \times 10^{-6}$$

$$p = 10^{-4} \Rightarrow P_{eb} \approx 3 \times 10^{-8}$$

$$p = 10^{-5} \Rightarrow P_{eb} \approx 3 \times 10^{-10}$$

The use of a repetition code does seem to improve the transmission reliability, at a rather high cost.

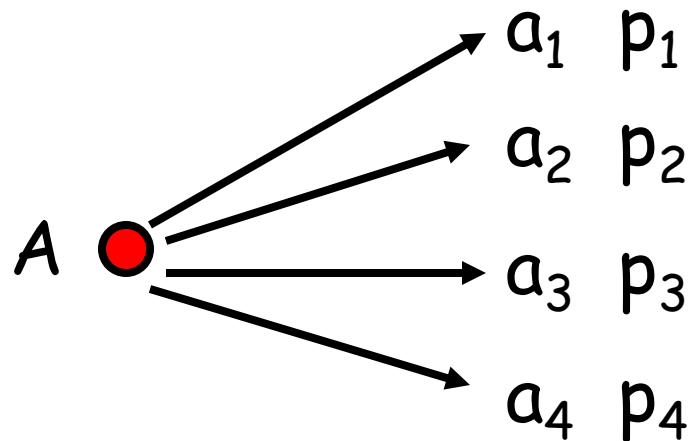
# Part 4

# Shannon Theory

# Sources of information

An information source  $A$  consists of an alphabet of  $M$  symbols  $a_i$ .

Each symbol  $a_i$  has a probability  $p_i = \Pr\{a_i\}$  to be generated by  $A$ .



$$\sum_{i=1}^M p_i = 1$$

# Information generated by a source

Shannon: Information is a measurable quantity with a precise definition.

When a symbol  $a_i$  with probability  $p_i$  is generated by  $A$ , the amount of info produced by  $A$  is

$$I_i = \log_2 \left( \frac{1}{p_i} \right) = -\log_2(p_i)$$

Unit: Shannon (or bit)

# Information generated by a source

Remember that  $\log_2(x) = \frac{\log_{10}(x)}{\log_{10}(2)}$

when using your calculators.

Generation of a highly probable symbol contains little info. For example, if  $p_i = 0.999$ , then the info conveyed is  $\approx 0.00144$  Shannon.

Generation of a highly improbable symbol provides a lot of info. For example, if  $p_i = 0.001$ , then the info conveyed is  $\approx 9.96578$  Shannon.

# Entropy of a source of information

The entropy of source  $A$  is simply the average amount of info generated by  $A$ :

$$H(A) = \sum_{i=1}^M p_i I_i = - \sum_{i=1}^M p_i \log_2(p_i)$$

The entropy of a source is a measure of its randomness.

The more random a source, the higher its entropy.

# Entropy of a source of information

First theorem of Shannon:

The entropy of  $A$  gives the minimum average number of bits required to represent a symbol of  $A$ , without loosing info (lossless source coding)

→ If you compress beyond the entropy, you destroy info.

Ex: A source can transmit one of three symbols  $a_1, a_2, a_3$  with associated probabilities  $p_1 = 0.60, p_2 = p_3 = 0.20$  → Its entropy is 1.371 Shannon.



# Entropy of a source of information

=> The best possible lossless source coding scheme would represent these symbols using an average of 1.371 bits per symbol.

It is easy to show that the entropy of  $A$  is maximal when symbols  $a_i$  have equal probabilities:

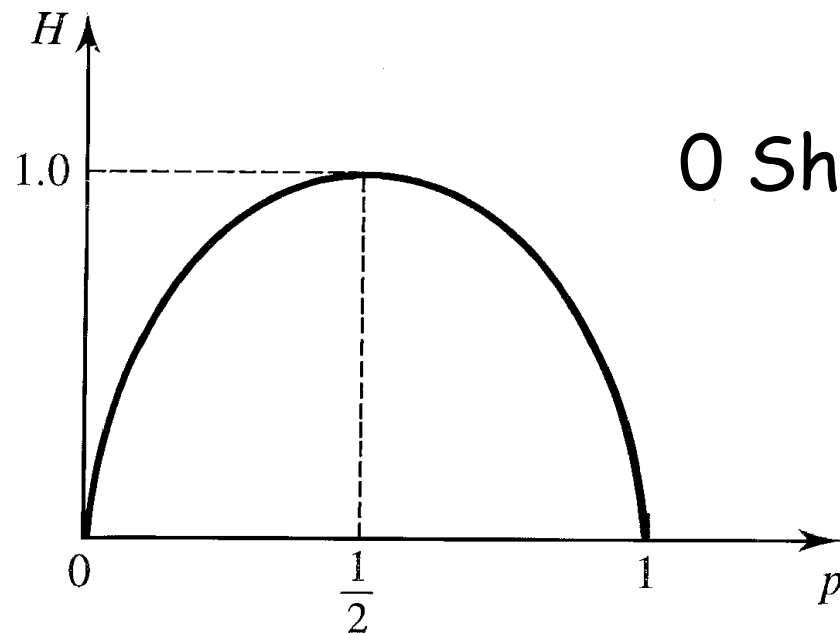
$$H(A)_{\max} = -\sum_{i=1}^M \frac{1}{M} \log_2 \left( \frac{1}{M} \right) = \log_2(M)$$

Does such result make sense to you?

# Entropy of a binary source

For a two-symbol (binary) source with probabilities  $(p, 1-p)$ , we have:

$$H = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$$



$$0 \text{ Shannon} \leq H(A) \leq 1 \text{ Shannon}$$

# Joint entropy of two sources

Consider two sources  $A (a_i, p_i, M_A)$  and  $B (b_j, p_j, M_B)$ .

The joint entropy  $H(A, B)$  is the total amount of info contained in both sources  $A$  and  $B$ , and is defined as:

$$H(A, B) = - \sum_{i=1}^{M_A} \sum_{j=1}^{M_B} p_{i,j} \log_2(p_{i,j})$$

Notations:

$$p_{i,j} = \Pr\{a_i, b_j\} \quad p_{j|i} = \Pr\{b_j|a_i\}$$

# Joint entropy of two sources

$$H(A, B) = - \sum_{i=1}^{M_A} \sum_{j=1}^{M_B} p_{j|i} p_i \left[ \log_2(p_{j|i}) + \log_2(p_i) \right]$$

$$H(A, B) = - \sum_{i=1}^{M_A} p_i \underbrace{\sum_{j=1}^{M_B} p_{j|i} \log_2(p_{j|i})}_{= H(B/a_i)} - \sum_{i=1}^{M_A} p_i \log_2(p_i) \underbrace{\sum_{j=1}^{M_B} p_{j|i}}_{= 1}$$

$$H(A, B) = \sum_{i=1}^{M_A} p_i H(B|a_i) - \sum_{i=1}^{M_A} p_i \log_2(p_i)$$

# Joint entropy of two sources

We obtain the expression of the joint entropy  $H(A,B)$ :

$$H(A,B) = H(B|A) + H(A) \quad (= H(A|B) + H(B))$$

$H(A)$ ,  $H(B)$ : Entropies of  $A$  and  $B$ ;

$H(B/A)$ ,  $H(A/B)$ : Conditional entropies of  $B$  given  $A$ , and  $A$  given  $B$ .

$H(B/A)$  measures the average amount of info provided by  $B$  when the outcome of  $A$  is known.

$H(A/B)$  measures the average amount of info provided by  $A$  when the outcome of  $B$  is known.

# Joint entropy of two sources

If  $A$  and  $B$  are independent:

$$H(A, B) = H(B) + H(A)$$

Does such result make sense to you?

If  $A$  completely defines  $B$ :

$$H(A, B) = H(A)$$

Does such result make sense to you?

# Mutual information of two sources

The reduction in uncertainty about source B due to the knowledge of the symbol generated by source A is called the mutual information  $I(B;A)$ .

It can be computed using:

$$I(B;A) = \underbrace{H(B)}_{\text{Amount of info in source B}} - \underbrace{H(B|A)}_{\text{Amount of info provided by B when the outcome of A is known}}$$

Crucial parameter to assess the quality of a transmission channel.

# Mutual information of two sources



The value of the mutual information  $I(X;Y)$  indicates how much the channel output  $Y$  can tell us about the channel input  $X \Rightarrow$  The quality of a channel is measured by  $I(X;Y)$ .

Ex:  $I(X;Y) = 0$  when  $H(X) = H(X/Y) \Rightarrow$  Useless channel.

Ex:  $I(X;Y) = H(X)$  when  $H(X/Y) = 0 \Rightarrow$  Noiseless channel.



# Mutual information of two sources

An alternative definition of  $I(B;A)$ :

$$I(B; A) = - \sum_{j=1}^{M_B} p_j \log_2(p_j) + \sum_{i=1}^{M_A} p_i \sum_{j=1}^{M_B} p_{j|i} \log_2(p_{j|i})$$

$$I(B; A) = - \sum_{j=1}^{M_B} p_j \log_2(p_j) \sum_{i=1}^{M_A} p_{i|j} + \sum_{i=1}^{M_A} p_i \sum_{j=1}^{M_B} p_{j|i} \log_2(p_{j|i})$$

$$I(B; A) = \sum_{i=1}^{M_A} \sum_{j=1}^{M_B} p_{i,j} \log_2 \left( \frac{p_{i,j}}{p_i p_j} \right) = E \left\{ \log_2 \left( \frac{p_{i,j}}{p_i p_j} \right) \right\}$$

Very useful for evaluating mutual info via simulations.

# Part 5

## Huffman Coding and Shannon's first theorem

# Source coding theorem (Shannon's first theorem)

The number of bits necessary to uniquely describe any data source can approach the corresponding entropy as closely as desired.

When the probabilities  $p_i$  of symbols  $a_i$  are known, Huffman coding can be used .

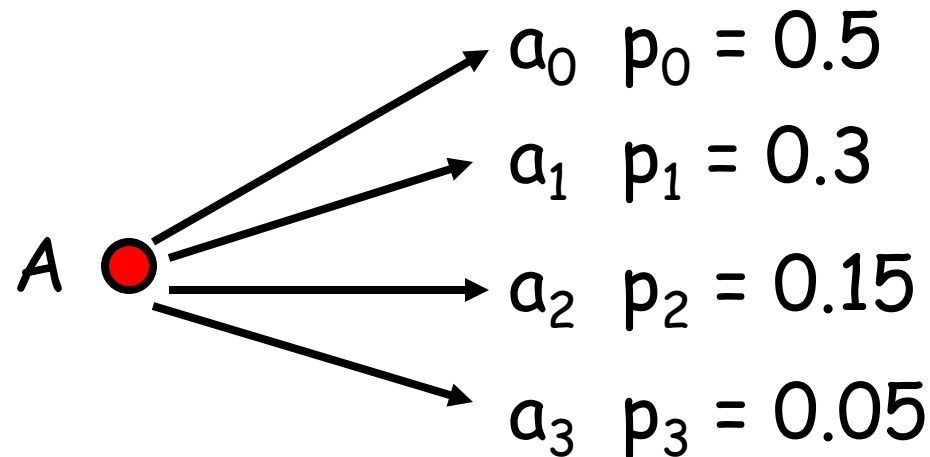


David A Huffman (1925 - 1999)

Introduced the Huffman coding algorithm when he was a graduate student at MIT in 1953.

# Huffman coding

Consider the following source of info:



Its entropy  $H(A)$  is equal to 1.6477 Shannon  $\rightarrow$  We do not need 2 bits to encode the symbols generated by  $A$ . We can do it using only 1.6477 bits (on average)

# Huffman coding

We can define the efficiency of our encoding technique:

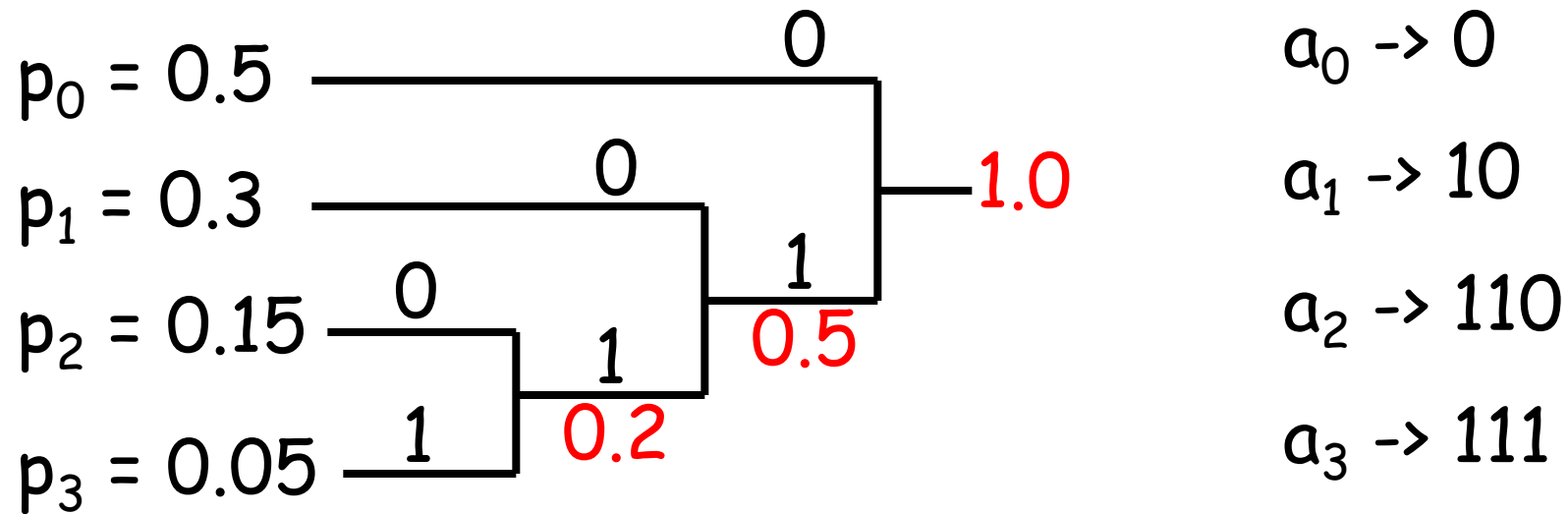
$$\eta = \frac{H(A)}{\bar{L}}$$

Average length of a codeword, in bits

If we use two bits for each symbol, we obtain an efficiency  $\eta = 1.6477/2 \approx 82.4\%$ , which is below the 100% promised by Shannon.

Huffman coding will allow us to reach 100%.

# Huffman coding



$$\bar{L} = \sum_{i=1}^3 p_i L_i = (0.5 \times 1) + (0.3 \times 2) + (0.15 \times 3) + (0.05 \times 3) = 1.7 \text{ bits}$$

The new efficiency is  $\eta' = 1.6477/1.7 \approx 96.9\%$ .

Can we increase it further with Huffman coding?

# Huffman coding

To increase the efficiency, deal with pairs of symbols  
→ New set of probabilities:

$$\Pr(a_0, a_0) = 0.25, \Pr(a_0, a_1) = 0.15, \Pr(a_0, a_2) = 0.075,$$

$$\Pr(a_0, a_3) = 0.025, \Pr(a_1, a_0) = 0.15, \Pr(a_1, a_1) = 0.09,$$

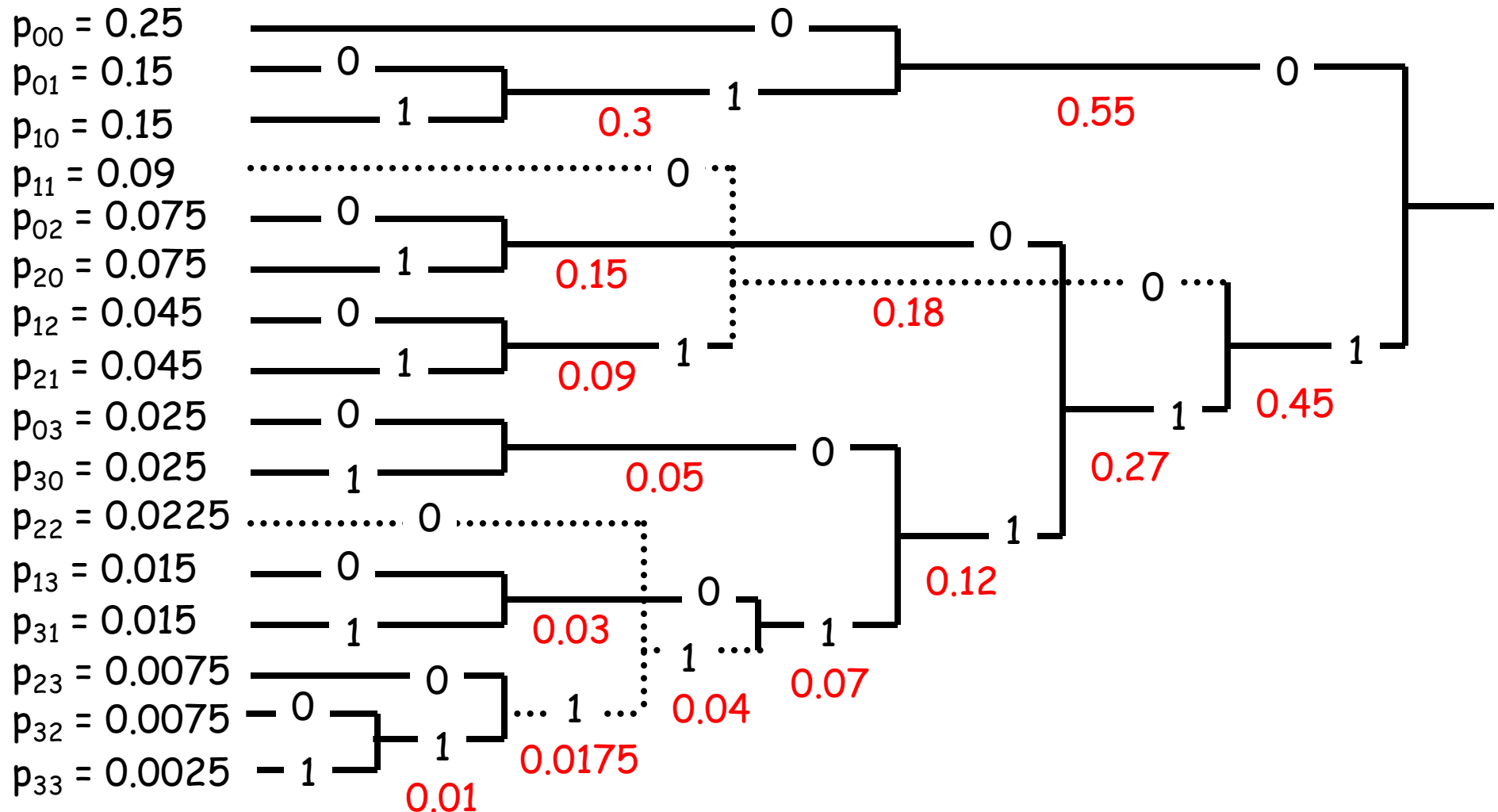
$$\Pr(a_1, a_2) = 0.045, \Pr(a_1, a_3) = 0.015, \Pr(a_2, a_0) = 0.075,$$

$$\Pr(a_2, a_1) = 0.045, \Pr(a_2, a_2) = 0.0225, \Pr(a_2, a_3) = 0.0075,$$

$$\Pr(a_3, a_0) = 0.025, \Pr(a_3, a_1) = 0.015, \Pr(a_3, a_2) = 0.0075,$$

$$\Pr(a_3, a_3) = 0.0025$$

# Huffman coding





# Huffman coding

Improved Huffman coding:

$(a_0, a_0) \rightarrow 00$ ,  $(a_0, a_1) \rightarrow 010$ ,  $(a_0, a_2) \rightarrow 1100$ ,  
 $(a_0, a_3) \rightarrow 11100$ ,  $(a_1, a_0) \rightarrow 011$ ,  $(a_1, a_1) \rightarrow 100$ ,  
 $(a_1, a_2) \rightarrow 0110$ ,  $(a_1, a_3) \rightarrow 111100$ ,  $(a_2, a_0) \rightarrow 1010$ ,  
 $(a_2, a_1) \rightarrow 1011$ ,  $(a_2, a_2) \rightarrow 111110$ ,  $(a_2, a_3) \rightarrow 1111110$ ,  
 $(a_3, a_0) \rightarrow 11101$ ,  $(a_3, a_1) \rightarrow 111101$ ,  $(a_3, a_2) \rightarrow 11111110$ ,  
 $(a_3, a_3) \rightarrow 11111111$

# Huffman coding

The efficiency becomes:

$$\eta'' = \frac{2 \times H(A)}{3.3275} \approx 99\%$$

By considering bigger and bigger groups of symbols, we, asymptotically, reach an efficiency of 100% using Huffman coding.

Note : Huffman codes are prefix codes, i.e. a code-word is never the beginning of another one. This property makes the decoding of such codes simple.

# Huffman coding

Ex: Decode the sequence {0110111100} assuming the codewords are transmitted from left to right, and the original Huffman coding.

→ The sequence must be broken as follows: {0, 110, 111, 10, 0}.  
There is no other possibility. 😊

→ { $a_0$ ,  $a_2$ ,  $a_3$ ,  $a_1$ ,  $a_0$ }

$a_0 \rightarrow 0$

$a_1 \rightarrow 10$

$a_2 \rightarrow 110$

$a_3 \rightarrow 111$

# Part 6

## Channel Capacity and Shannon's Second Theorem

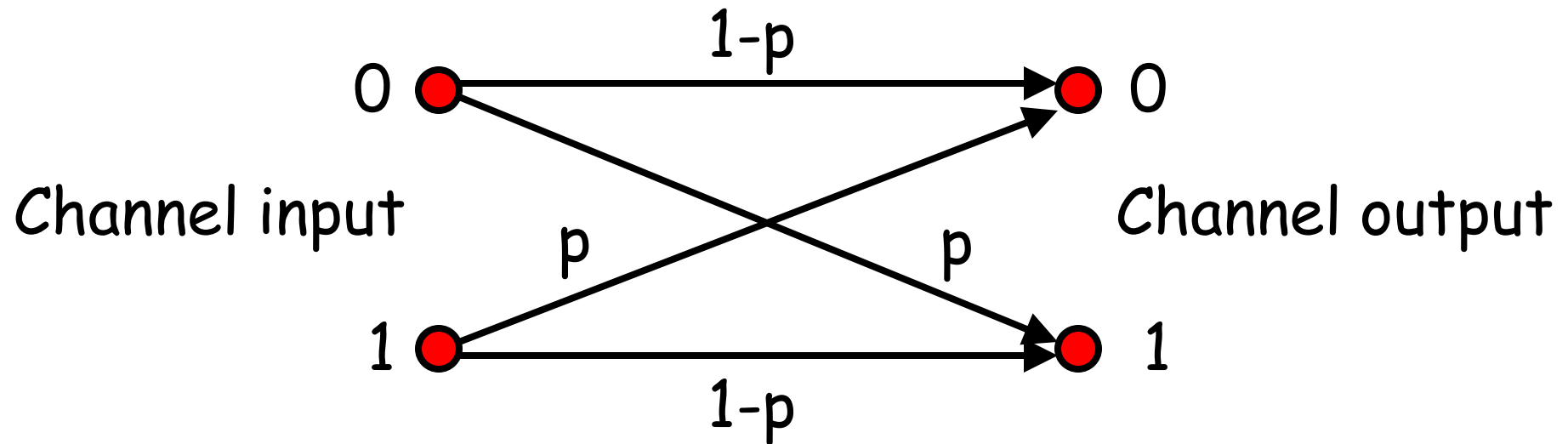
# Communication channels

We consider two practical channels:

1. The binary symmetric channel (BSC);
2. The BPSK (binary phase shift keying), AWGN (additive white Gaussian noise) channel.

# Communication channels

## BSC model



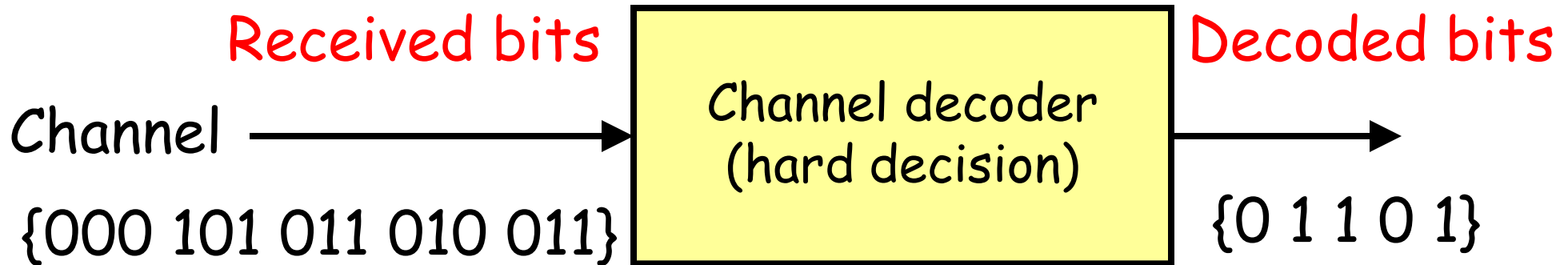
$p$  is the error probability over the channel ( $p < 0.5$ ).  
At the channel input,  $\Pr\{0\} = \Pr\{1\} = \frac{1}{2} \rightarrow 1$  Shannon of info is sent through the channel whenever a bit is transmitted.

# Communication channels

## BSC model

The sequence at the channel output, i.e. decoder input, is composed of bits.

Ex:



# Communication channels

## BPSK, AWGN channel model

Channel output:

$$r = s + n$$

$s$  is the transmitted symbol ( $s = +1$  or  $-1$  with  $\Pr\{s = +1\} = \Pr\{s = -1\} = \frac{1}{2}$ )

$n$  is a Gaussian noise sample, with zero mean and variance :

$$\sigma^2 = \frac{1}{2 \cdot \text{SNR}}$$

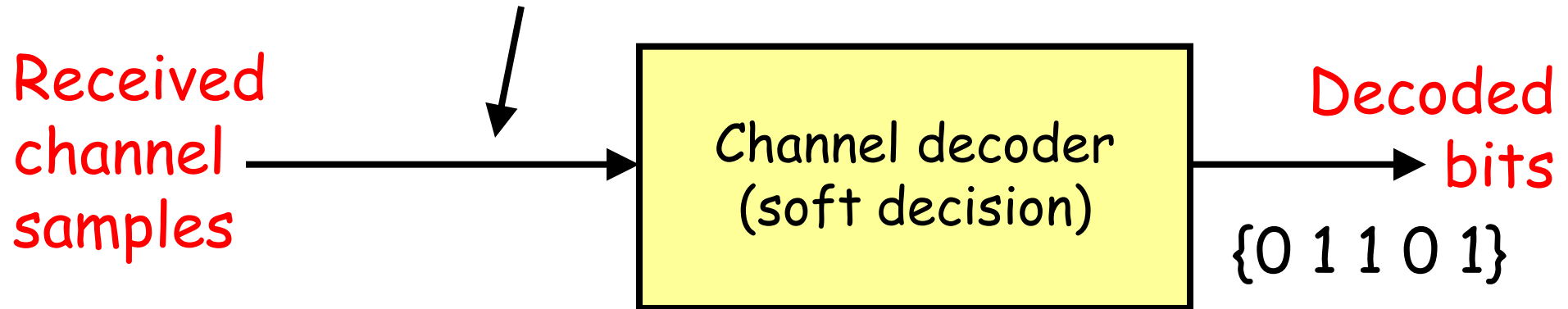


# Communication channels

## BPSK, AWGN channel model

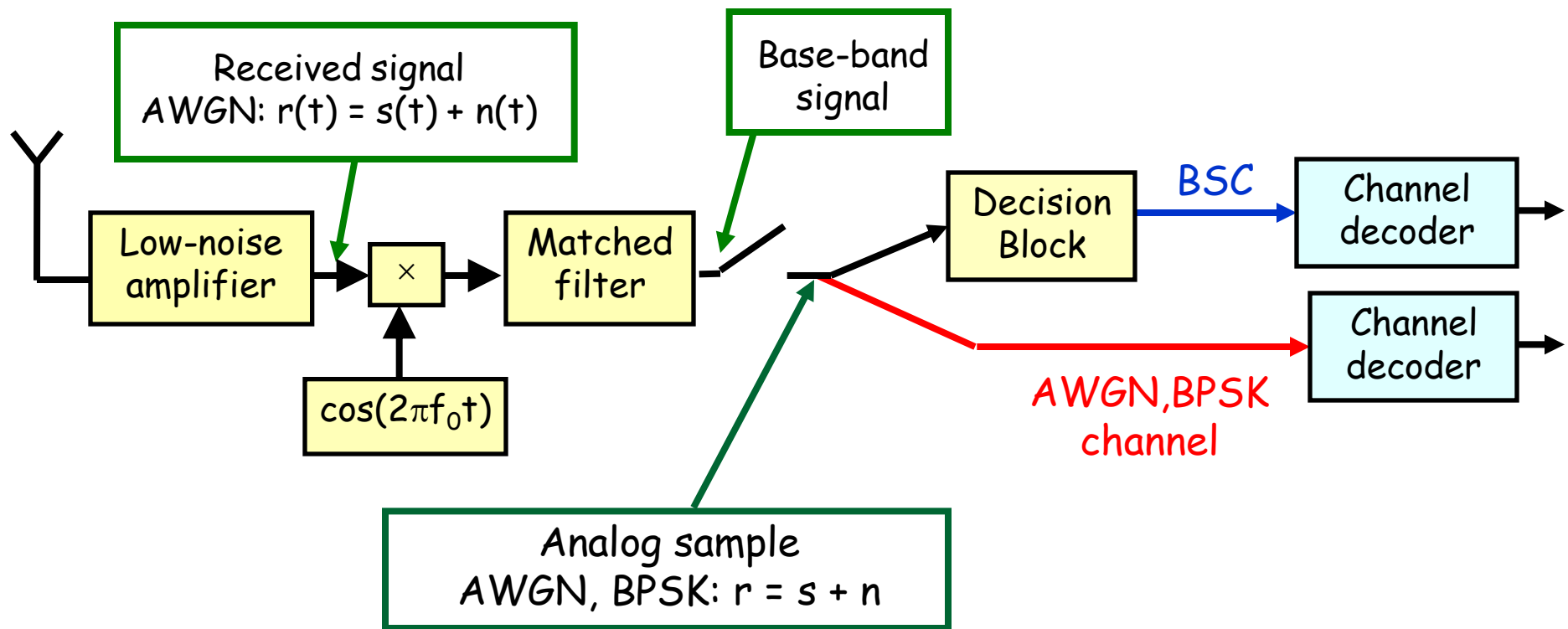
The sequence at the channel output, i.e. decoder input, is composed of analogue estimates of the transmitted symbols  $s = +1$  or  $-1$ .

Ex:  $\{-1.5, -1.1, -0.2, 0.4, -0.2, 2.7, \dots\}$



# Communication channels

The use of a decision block inside the demodulator is often the only difference between a BSC and a BPSK, AWGN channel.



# Channel capacity



The capacity  $C$  of a channel (to transmit information) is, in the context of this course, measured by the mutual information  $I(X;Y)$ .

$I(X;Y)$  indicates how much the channel output  $Y$  can tell us about the channel input  $X \rightarrow$  The quality of a channel is measured by  $I(X;Y)$ .

# BSC capacity

$$C = E \left\{ \log_2 \left( \frac{p_{i,j}}{p_i p_j} \right) \right\} = E \left\{ \log_2 \left( \frac{p_{j|i}}{p_j} \right) \right\}$$

- $p_{i,j}$ :  $\Pr\{\text{symbol } i \in \{0, 1\} \text{ at the input \& symbol } j \in \{0, 1\} \text{ at the output}\}$
- $p_i$ :  $\Pr\{\text{symbol } i \text{ at the input}\}$
- $p_j$ :  $\Pr\{\text{symbol } j \text{ at the output}\}$
- $p_{j|i}$ :  $\Pr\{\text{symbol } j \text{ at the output given symbol } i \text{ at the input}\}$

# BSC capacity

$$C = E \left\{ \log_2 \left( \frac{p_{j|i}}{\sum_i p_{j|i} p_i} \right) \right\} = E \left\{ \log_2 \left( \frac{2p_{j|i}}{p_{j|0} + p_{j|1}} \right) \right\} = 1 + E \left\{ \log_2 (p_{j|i}) \right\}$$

$$C = 1 + \sum_{i,j} p_{i,j} \log_2 (p_{j|i}) = 1 + \frac{1}{2} \sum_{i,j} p_{j|i} \log_2 (p_{j|i})$$

$$C = 1 + p \log_2(p) + (1-p) \log_2(1-p) = 1 - H(p)$$

$H(p)$  is the entropy of a binary source with probability distribution  $(p, 1-p)$ .

# BSC capacity

$C$  is traditionally expressed in Shannon/channel use.

Useless channel:  $C = 0$  Shannon/channel use, if  $p = \frac{1}{2}$ .

Perfect channel:  $C = 1$  Shannon/channel use, if  $p = 0$  or  $p = 1$ . Is it surprising?

The capacity  $C$  depends on the error probability at the channel output (expected).

For  $0 < p < 0.5$ , we have  $0 < C < 1$  Shannon/channel use.

# BPSK, AWGN channel capacity

$$C = E \left\{ \log_2 \left( \frac{p_{i,j}}{p_i p_j} \right) \right\} = E \left\{ \log_2 \left( \frac{p_{j|i}}{p_j} \right) \right\}$$

The channel output is now an analogue source of info  
→ Replace probabilities with probability density functions (PDFs).

The channel output is a Gaussian sample with mean equal to  $s = +1$  or  $-1$ , and variance

$$\sigma^2 = \frac{1}{2 \cdot \text{SNR}}$$

# BPSK, AWGN channel capacity

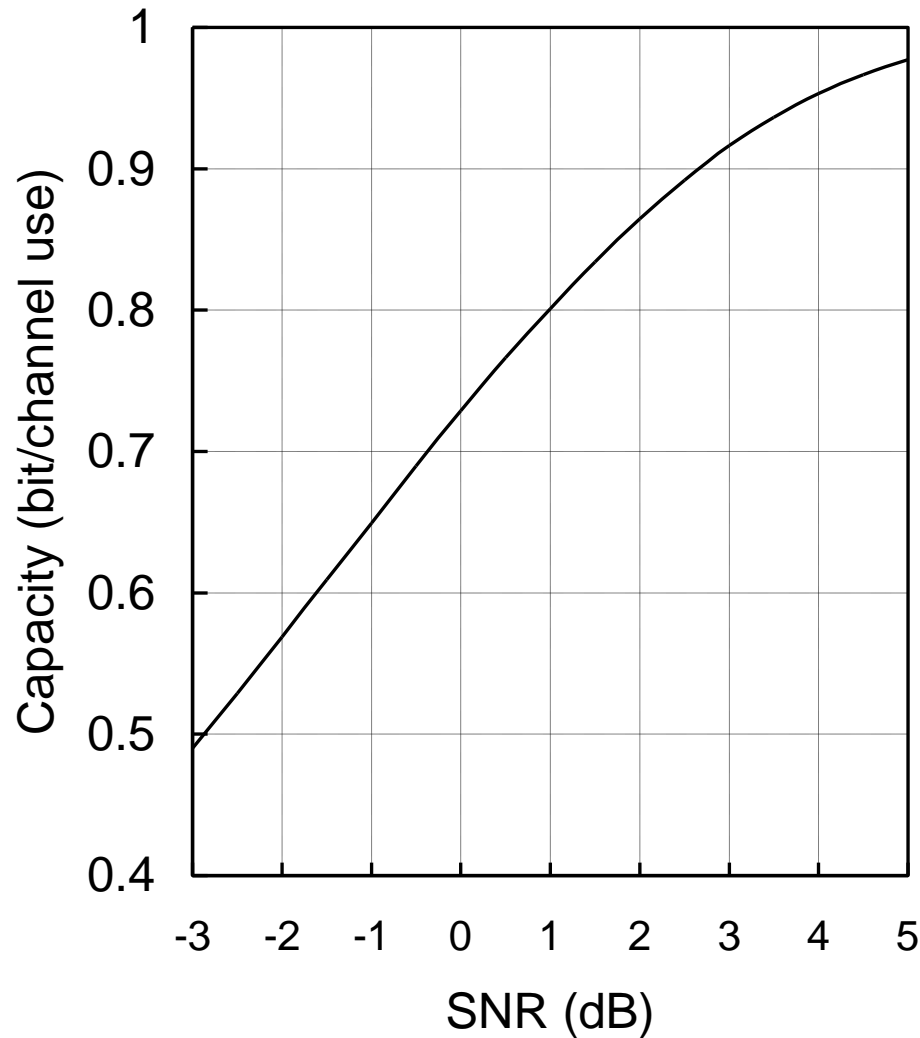
$$C = E \left\{ \log_2 \left( \frac{p_{j|i}}{p_j} \right) \right\} = E \left\{ \log_2 \left( \frac{2p_{j|i}}{\sum_i p_{j|i}} \right) \right\} = E \left\{ \log_2 \left( \frac{2 \exp \left\{ -SNR \cdot (r-s)^2 \right\}}{\sum_i \exp \left\{ -SNR \cdot (r-i)^2 \right\}} \right) \right\}$$

$$C = 1 + E \left\{ \log_2 \left( \frac{\exp \left\{ -SNR \cdot (r-s)^2 \right\}}{\exp \left\{ -SNR \cdot (r-1)^2 \right\} + \exp \left\{ -SNR \cdot (r+1)^2 \right\}} \right) \right\}$$

Evaluate the expectation by Monte-Carlo simulations  
→ Send millions of symbols  $s$  and find the average value of  $\log_2(\dots)$ .



# BPSK, AWGN channel capacity



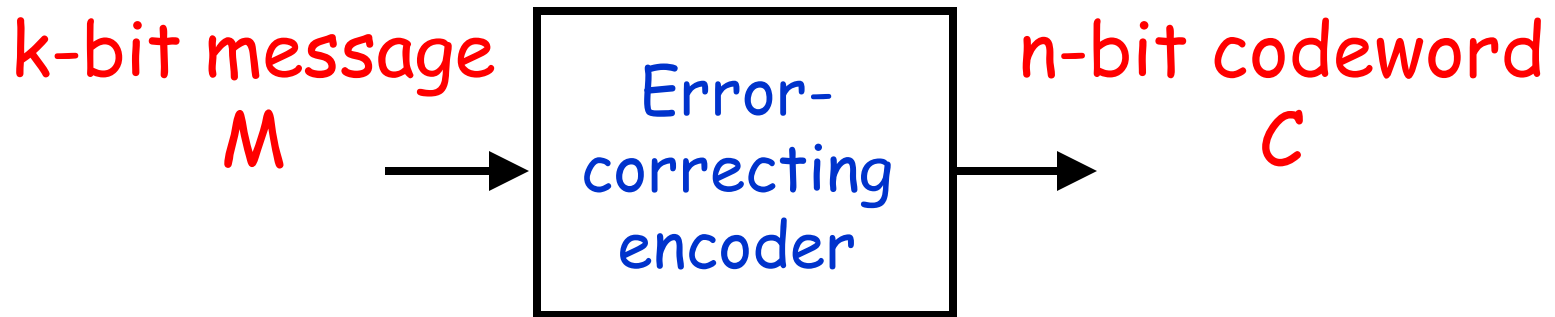
$C$  depends on the SNR over the channel.

The higher the SNR, the higher the capacity.

SNR is the signal-to-noise ratio per transmitted symbol  $s$ , also denoted as  $E_s/N_0$ .

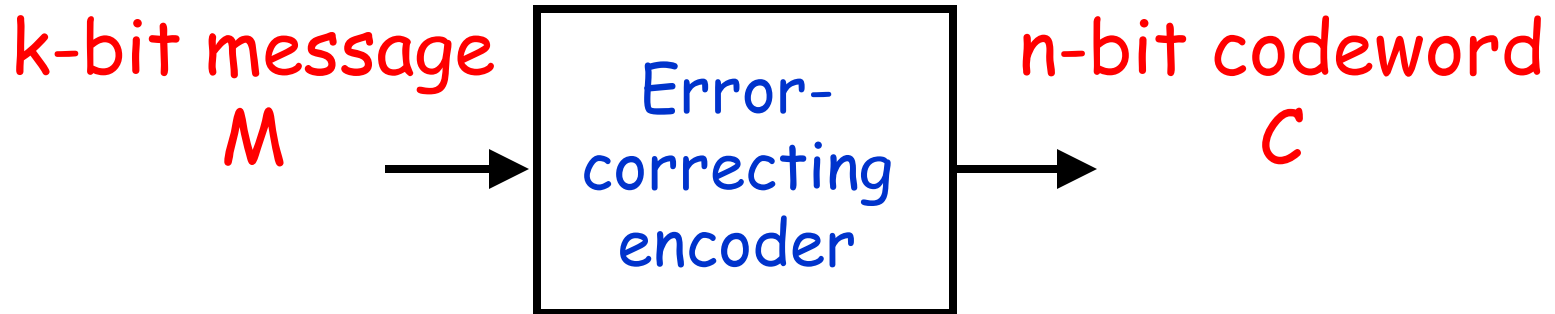
# Shannon's second theorem

Consider the use of an error-correcting code at the transmitter side:



A message of  $k$  bits contains  $k$  Shannon of info  $\rightarrow$  A codeword of  $n$  bits also contains  $k$  Shannon of info since the encoding process does not add any info.

# Shannon's second theorem



Each coded bit carries  $k/n$  Shannon of info.

The ratio  $k/n$  is also the coding rate  $R_c$  of the code, and the info rate  $R$  over the channel (each time we use the channel in order to transmit a coded bit, we actually transmit only  $k/n$  Shannon of info).

# Shannon's second theorem

Shannon: "The error probability at the receiver output can be reduced to zero (using error-correcting coding) provided that the info rate  $R$  is less than or equal to the channel capacity  $C$ ."

$$\text{Shannon limit: } R = C$$

This theorem allows us to determine the upper limit on info rate while maintaining reliable communications ( $P_{eb} = 0$ ).

It does not explicitly tell us how to reach this limit...

# Application to BSC

$$C = 1 + p \log_2(p) + (1 - p) \log_2(1 - p) = 1 - H(p)$$

As long as the coding rate  $R_c \leq C$ , there is an error-correcting code allowing for error-free communications.

Ex:

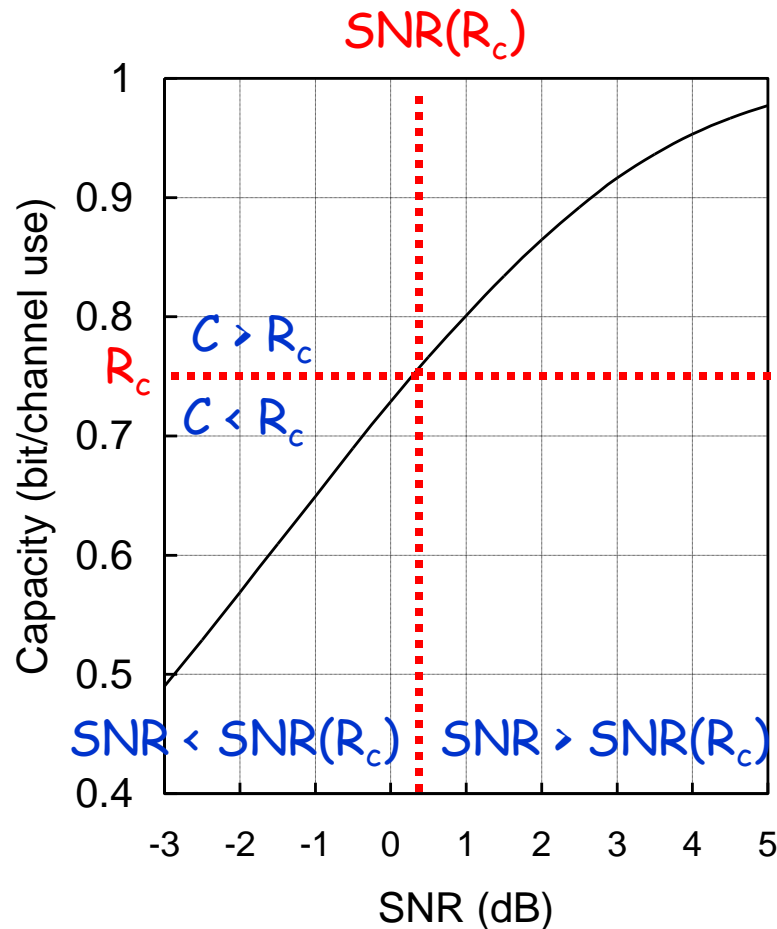
$$p = 10^{-1} \rightarrow R_c \leq 0.531;$$

$$p = 10^{-2} \rightarrow R_c \leq 0.91921;$$

$$p = 10^{-3} \rightarrow R_c \leq 0.98859;$$

$$p = 10^{-4} \rightarrow R_c \leq 0.99853.$$

# Application to BPSK, AWGN channel

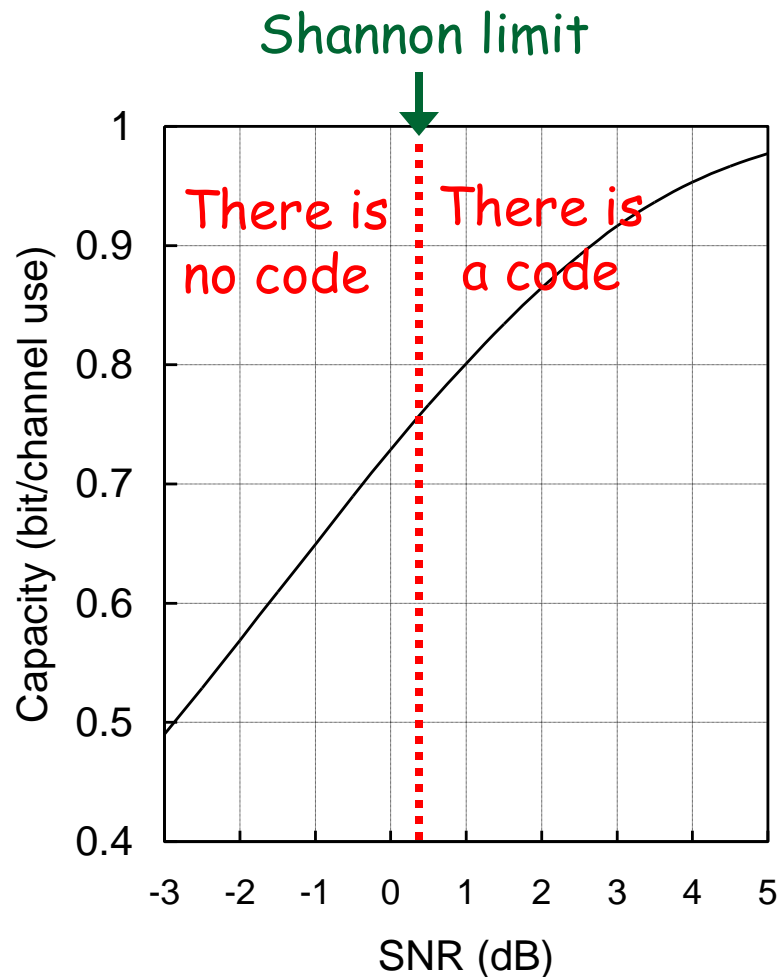


There is a rate- $R_c$  code allowing for error-free communications, provided that  $R_c \leq C$ .

Assume the rate  $R_c$  is given. Then, error-free communications is possible if  $C \geq R_c$ .

Since  $C$  depends on the channel SNR, this means that error-free communications is possible if  $\text{SNR} > \text{SNR}(C = R_c)$ .

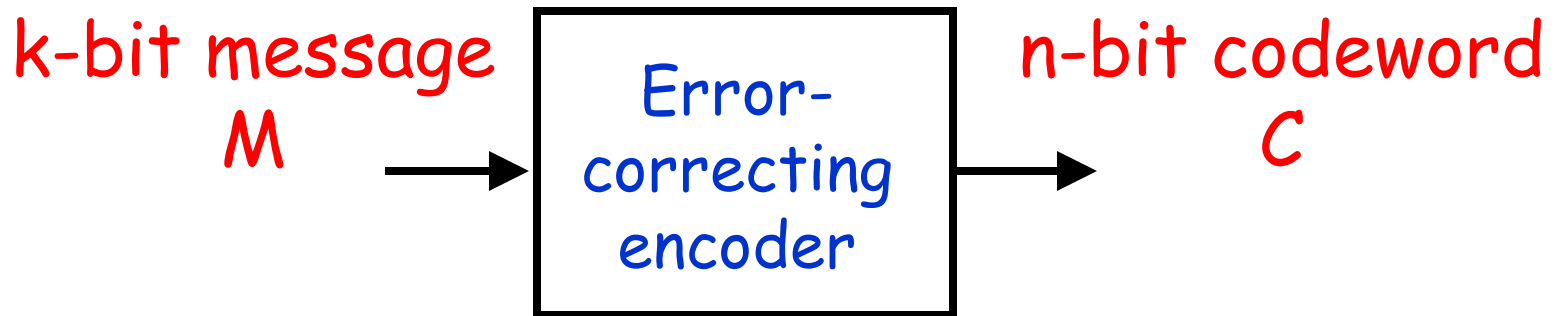
# Application to BPSK, AWGN channel



Example: If we want to use a rate-1/2 code, error-free communications is possible if  $\text{SNR} \geq -2.8 \text{ dB}$ .

We must consider the SNR per Shannon of info  $E_b/N_0$ , instead of the SNR per transmitted symbol  $E_s/N_0$ .

# Application to BPSK, AWGN channel



Total Energy in message  $M = kE_b$ .

Total Energy in codeword  $C = nE_s$ .

We must have:  $kE_b = nE_s \rightarrow E_b = E_s/R_c \rightarrow E_b/N_0 = (E_s/N_0)/R_c$ . In dB,  $E_b/N_0 = E_s/N_0 - 10\log_{10}(R_c)$ .



# Application to BPSK, AWGN channel

Ex: If  $R_c = 1/2$ , then  $E_b/N_0 \approx -2.8 \text{ dB} - 10\log_{10}(1/2) \approx 0.2 \text{ dB}$ .

→ There is a rate-1/2 code that allows for error-free communications if the SNR per Shannon is greater than or equal to  $\approx 0.2 \text{ dB}$

→ Shannon limit  $\approx 0.2 \text{ dB}$ .

What code should we use to reach the Shannon limit?

(1) Message and codeword of infinite length  
( $k$  and  $n \rightarrow +\infty$ ).

# Application to BPSK, AWGN channel

(2) For any message, the corresponding codeword is chosen randomly and independently of the other codewords --> Random coding.

These two rules were simply assumptions used in the mathematical demonstration of the second theorem of Shannon in 1948.

# The challenge after 1948

In 1948 (and for many years until the mid-90s), the clues offered by Shannon were not considered very useful since the implementation of long random codes seemed far too complex.

Think of the encoding and decoding complexity of such codes at a time where there were no integrated circuits or computers around.

At the encoder side, how to implement a random encoder? Use a look-up table with a capacity of  $n \times 2^k$  bits?

# The challenge after 1948

At the decoder side, how to search for the most likely codeword among the  $2^k$  possible codewords?

As a result, researchers focused on the search for codes that possess a lot of "structure", i.e. for which the encoding and decoding complexities are reasonable.

For a given coding rate  $R_c$  and complexity, the goal was mainly to maximize a parameter called the minimum Hamming distance between codewords.

# The challenge after 1948

The problem with such approach is that it leads to codes that do not perform close to the Shannon limit.

For 45 years, finding a “practical” code that could perform close to the Shannon limit was considered an utopia.

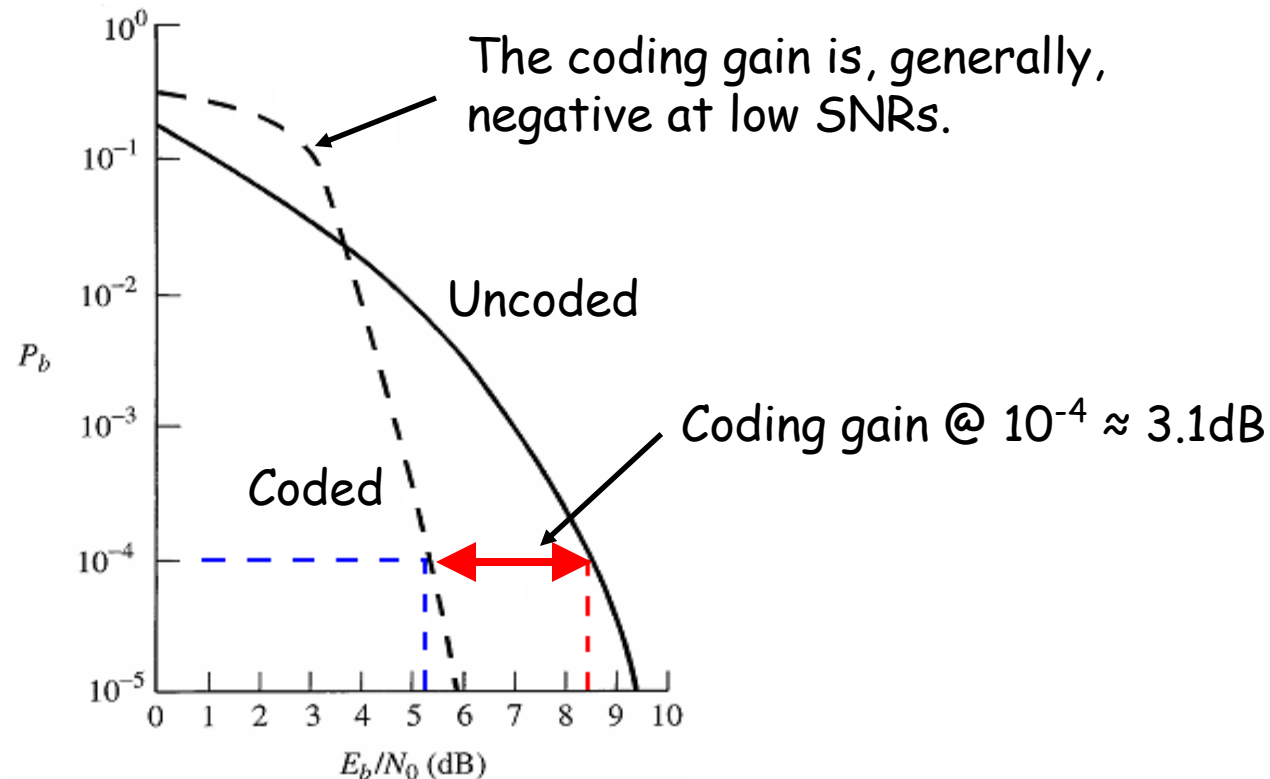
Then came the IEEE International Conference on Communications, Geneva, May 1993...

# Part 7

## Error-Correcting Codes

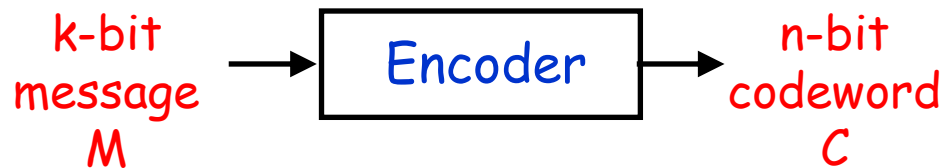
# Rationale of error-correcting coding

Goal: Achieve a target  $P_{eb}$  using a lower SNR.



# Types of codes

## Block codes



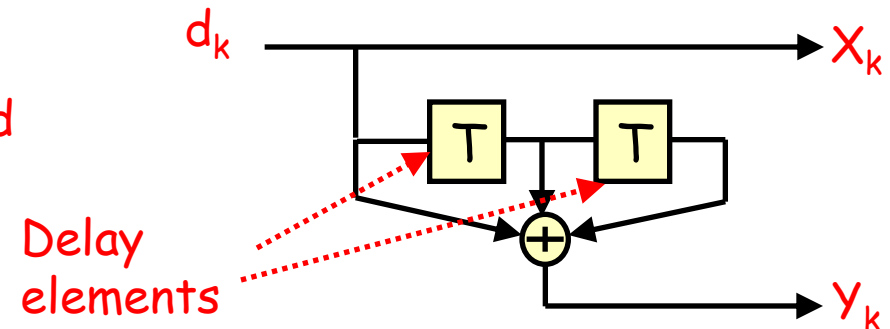
No memory from one message to the other

M: Message

C : Codeword

Coding rate  $R_c = k/n < 1$

## Convolutional codes



$d_k$ : Info bit at  $t = kT$

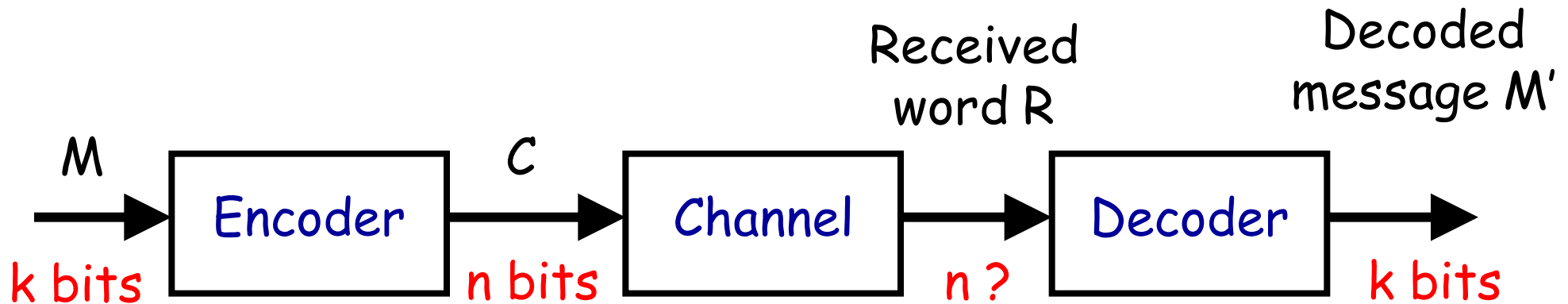
$X_k, Y_k$ : Coded bits at  $t = kT$

Coding rate  $R_c = \frac{1}{2}$ , can be increased using puncturing

Convolutional codes can be considered as block codes under certain conditions.



# The big picture (block codes)



$$M = (m_0, m_1, \dots, m_{k-1})$$

$$C = (c_0, c_1, \dots, c_{n-1})$$

$$M' = (m'_0, m'_1, \dots, m'_{k-1})$$

} Binary vectors

$R = (r_0, r_1, \dots, r_{n-1}) \longrightarrow$  Binary vector or vector of analogue samples.

# The big picture (block codes)

Coding rate:  $R_c = \frac{k}{n}$  ( $R_c < 1$  since  $k < n$ )

The use of an error-correcting code increases the bandwidth by a factor  $n/k$  for a given data rate  
OR decreases the data rate by a factor  $n/k$  for a given bandwidth. ☹️

# The big picture (block codes)

Hamming distance between two binary words  $C_1$  and  $C_2$ : Number of positions at which these two vectors are different.

Ex:  $C_1 = (0\ 1\ 1\ 0\ 1)$ ,  $C_2 = (1\ 1\ 1\ 1\ 0) \rightarrow d_H(C_1, C_2) = 3$ .

$2^k$  possible messages  $M \rightarrow$  Need for  $2^k$  codewords  $C$ .

But there are actually  $2^n (> 2^k)$  possible codewords  $C$ .

# The big picture (block codes)

Our primary goal is to choose the set of  $2^k$  codewords (among  $2^n$  available codewords) so that the Hamming distance between these codewords is maximal.

In general, block codes are linear codes.

A linear block code is completely defined using a generator matrix  $G$  so that  $C = M.G$ .

# Linear block codes

Ex 1: ( $n = 7, k = 4$ ) Hamming code

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{aligned} c_0 &= m_0 \\ c_1 &= m_1 \\ c_2 &= m_2 \\ c_3 &= m_3 \\ c_4 &= m_0 + m_2 + m_3 \\ c_5 &= m_0 + m_1 + m_2 \\ c_6 &= m_1 + m_2 + m_3 \end{aligned}$$

In other words, a linear block code is completely defined by a set of  $n$  linear equations.

# Linear block codes

Ex 2:  $(n = 5, k = 2)$  code

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{aligned} c_0 &= m_0 \\ c_1 &= m_1 \\ c_2 &= m_0 \\ c_3 &= m_1 \\ c_4 &= m_0 + m_1 \end{aligned}$$

Let us have a closer look at this simple linear code.

# The $(n = 5, k = 2)$ code

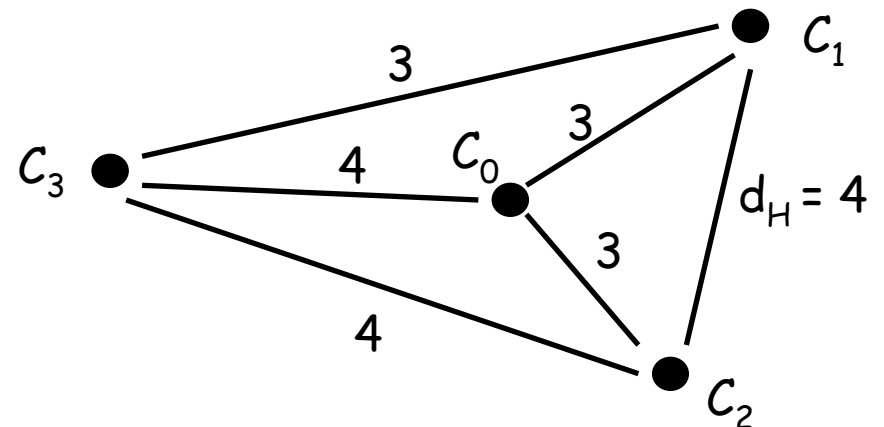
$M$		$C$				
$m_0$	$m_1$	$c_0$	$c_1$	$c_2$	$c_3$	$c_4$
0	0	0	0	0	0	0
0	1	0	1	0	1	1
1	0	1	0	1	0	1
1	1	1	1	1	1	0

$$C_0 = (00000)$$

$$C_1 = (01011)$$

$$C_2 = (10101)$$

$$C_3 = (11110)$$



## The $(n = 5, k = 2)$ code

Assume  $C_0$  is transmitted over a BSC, and there is a transmission error, i.e.  $d_H(R, C_0) = 1$ .

In such case, we have  $d_H(R, C_1) \geq 2$ ,  $d_H(R, C_2) \geq 2$ , and  $d_H(R, C_3) \geq 2$ .

The decision rule consists of choosing the codeword which is at minimum Hamming distance from  $R$ .

→ We choose  $C_0$ , and thus select the right codeword despite the transmission error.



## The $(n = 5, k = 2)$ code

Assume that there are two errors, i.e.  $d_H(R, C_0) = 2$ .

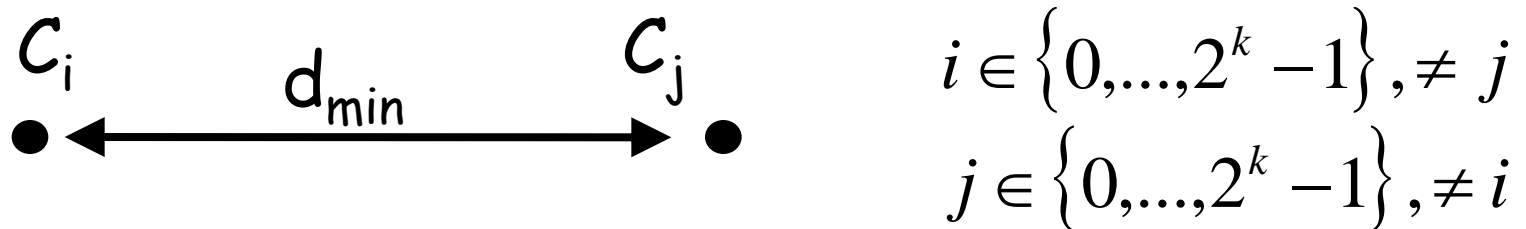
In such case, nothing guarantees us that  $d_H(R, C_1) \geq 3$ ,  $d_H(R, C_2) \geq 3$ ,  $d_H(R, C_3) \geq 3$  (it will, probably, not be the case).

→ We are probably going to choose a codeword different from  $C_0$ , and thus fail to decode properly the received word  $R$ .

We can correct up to one error in a received word of 5 bits.

# Error-correction power of a code

Assume a code for which the minimal Hamming distance between any pair of codewords is  $d_{\min}$ .



The maximum number of errors that we can correct in a received word  $R$  of  $n$  bits is the error-correction power  $t$  computed as:

$$t = \text{Int} \left[ \frac{d_{\min} - 1}{2} \right]$$

# Examples of linear block codes

Ex 1: Repetition code ( $k = 1, n$  odd)  $G = [1 \quad \dots \quad 1]$

$C = \{c_0, c_1, \dots, c_{n-1}\}$  with  $c_0 = c_1 = \dots = c_{n-1} = m_0$

$$d_{\min} = n \longrightarrow t = \frac{n-1}{2}$$

Low coding rate since  $R_c = 1/n$  ☹

# Examples of linear block codes

Ex 2: Parity-Check Code ( $n = k + 1$ ,  $R_c = k/(k+1)$ )

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

High, especially  
when  $k$  is large

$$c_0 = m_0, c_1 = m_1, \dots, c_{k-1} = m_{k-1}, c_k = m_0 + m_1 + \dots + m_{k-1}$$

$d_{\min} = 2 \rightarrow t = 0 \rightarrow$  Not able to correct errors. Just able to detect one error in a word of  $n$  bits.

# Examples of linear block codes

Ex 3: The ( $n = 7$ ,  $k = 4$ ,  $d_{\min} = 3$ ) Hamming code

M	C	M	C
0 0 0 0	0 0 0 0 0 0 0	1 0 0 0	1 0 0 0 1 1 0
0 0 0 1	0 0 0 1 1 0 1	1 0 0 1	1 0 0 1 0 1 1
0 0 1 0	0 0 1 0 1 1 1	1 0 1 0	1 0 1 0 0 0 1
0 0 1 1	0 0 1 1 0 1 0	1 0 1 1	1 0 1 1 1 0 0
0 1 0 0	0 1 0 0 0 1 1	1 1 0 0	1 1 0 0 1 0 1
0 1 0 1	0 1 0 1 1 1 0	1 1 0 1	1 1 0 1 0 0 0
0 1 1 0	0 1 1 0 1 0 0	1 1 1 0	1 1 1 0 0 1 0
0 1 1 1	0 1 1 1 0 0 1	1 1 1 1	1 1 1 1 1 1 1

$$d_{\min} = 3 \rightarrow t = 1$$

# Minimum Hamming distance of a linear code

How to easily determine the minimum Hamming distance of a linear block code?

Linear block codes have some interesting properties:

1. The all-zero binary word (" $C_0$ ") is a valid codeword
2. The sum of 2 codewords is another codeword.

Finding the  $d_{\min}$  of a linear code is made easier thanks to those properties.

# Minimum Hamming distance of a linear code

$d_{\min} = \min d_H(C_i, C_j)$ , for any  $i$  and  $j \in \{0, \dots, 2^k-1\}$ ,  $i \neq j$ .

$d_H(C_i, C_j) = w_H(C_i + C_j) = w_H(C_k)$ , where  $w_H(.)$  denotes the Hamming weight of "." and  $C_k$  is another valid codeword.

The Hamming weight of a binary vector is simply defined as the number of 1s in this vector.

# Minimum Hamming distance of a linear code

Therefore, we have  $d_{\min} = \min w_H(C_k)$ , for any  $k \in \{1, \dots, 2^k-1\}$ .

=> To find the  $d_{\min}$  of a linear block code, we just need to find the codeword having the smallest Hamming weight among all codewords (except  $C_0$ , of course).



# Systematic block codes

A block code is said to be systematic if the info bits are explicitly included in the codeword.

In other words, a  $n$ -bit codeword is composed of the  $k$  info bits to which  $(n-k)$  parity bits are appended:  $C = M + P$ .

Many practical codes are systematic.

# Decoding of error correcting codes

Optimal decoding: Maximum Likelihood Decoding

Let us start with ML decoding over BSC (AKA hard-decision decoding)

- Two codewords  $C_i$  and  $C_j$ ,  $i$  and  $j \in \{0, \dots, 2^k - 1\}$
- Received binary word  $R$

Optimal decision rule:

$$\Pr\{R, C_i\} \underset{'C_j'}{>} \Pr\{R, C_j\}$$

# ML decoding over BSC (hard-decision decoding)

Applying Bayes' rule, we find:

$$\Pr\{C_i\} \Pr\{R | C_i\} \overset{\text{'C}_i\text{'}}{>} \Pr\{C_j\} \Pr\{R | C_j\} \overset{\text{'C}_j\text{'}}{<}$$

$$\text{with } \Pr\{C_i\} = \Pr\{C_j\} = \frac{1}{2^k}$$

# ML decoding over BSC (hard-decision decoding)

$$p^{d_H(i)} (1-p)^{n-d_H(i)} \underset{\substack{> \\ <}}{\overset{C_i}{\underset{C_j}}} p^{d_H(j)} (1-p)^{n-d_H(j)}$$

- $d_H(i)$  : Hamming distance between  $R$  and  $C_i$
- $d_H(j)$  : Hamming distance between  $R$  and  $C_j$
- $p$ : Error probability over the BSC ( $p < 0.5$ )

# ML decoding over BSC (hard-decision decoding)

$$\left(\frac{p}{1-p}\right)^{d_H(i)-d_H(j)} \begin{matrix} > \\ < \end{matrix} \begin{matrix} 'C_i' \\ 'C_j' \end{matrix} \quad , \quad \text{with} \quad \frac{p}{1-p} = \alpha < 1$$

Decision ' $C_i$ ' if  $\alpha^{d_H(i)-d_H(j)} > 1 \Leftrightarrow d_H(j) > d_H(i)$

Decision ' $C_j$ ' if  $\alpha^{d_H(i)-d_H(j)} < 1 \Leftrightarrow d_H(i) > d_H(j)$

ML decoding over BSC: Choose the codeword which is at minimum Hamming distance from  $R$

# ML decoding over BPSK, AWGN channel (soft-decision decoding)

- Two codewords  $C_i = (c_{i,l})$  and  $C_j = (c_{j,l})$ ,  
 $i$  and  $j \in \{0, \dots, 2^k - 1\}$ ,  $l \in \{0, \dots, n-1\}$ ,  $c_{i,l}$  and  $c_{j,l} \in \{-1, +1\}$ .
- Received word  $R = (r_l)$ ,  $l \in \{0, \dots, n-1\}$ . The analogue samples  $r_l$  follow a Gaussian distribution.

Optimal decision rule:

$$\Pr\{R, C_i\} \begin{matrix} > \\ < \end{matrix} \Pr\{R, C_j\}$$

$\begin{matrix} 'C_i' \\ 'C_j' \end{matrix}$

# ML decoding over BPSK, AWGN channel (soft-decision decoding)

$$\rightarrow \Pr\{R | C_i\} \underset{\substack{\text{'C}_i\text{'}}{\text{'C}_j\text{'}}} > \Pr\{R | C_j\} \quad \text{since } \Pr\{C_i\} = \Pr\{C_j\}$$

$$\rightarrow \prod_{l=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left\{-\frac{(r_l - c_{i,l})^2}{2\sigma^2}\right\} \underset{\substack{\text{'C}_i\text{'}}{\text{'C}_j\text{'}}} > \prod_{l=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left\{-\frac{(r_l - c_{j,l})^2}{2\sigma^2}\right\}$$

# ML decoding over BPSK, AWGN channel (soft-decision decoding)

$$\begin{aligned} &\rightarrow \exp \left\{ - \frac{\sum_{l=0}^{n-1} (r_l - c_{i,l})^2}{2\sigma^2} \right\} \begin{matrix} 'C_i' \\ > \\ < \\ 'C_j' \end{matrix} \exp \left\{ - \frac{\sum_{l=0}^{n-1} (r_l - c_{j,l})^2}{2\sigma^2} \right\} \\ &\rightarrow \sum_{l=0}^{n-1} (r_l - c_{i,l})^2 \begin{matrix} 'C_j' \\ > \\ < \\ 'C_i' \end{matrix} \sum_{l=0}^{n-1} (r_l - c_{j,l})^2 \rightarrow d_E(i) \begin{matrix} 'C_j' \\ > \\ < \\ 'C_i' \end{matrix} d_E(j) \end{aligned}$$

- $d_E(i)$  : Euclidean distance between  $R$  and  $C_i$
- $d_E(j)$  : Euclidean distance between  $R$  and  $C_j$



# ML decoding over BPSK, AWGN channel (soft-decision decoding)

ML decoding over BPSK, AWGN channel: Choose the codeword which is at minimum Euclidean distance from  $\mathbf{R}$ .

Note that the decision rule can be simplified as:

$$\sum_{l=0}^{n-1} (r_l - c_{i,l})^2 \underset{\substack{\text{'C}_j\text{'}}}{>} \sum_{l=0}^{n-1} (r_l - c_{j,l})^2 \underset{\substack{\text{'C}_i\text{'}}}{<} \sum_{l=0}^{n-1} (r_l - c_{j,l})^2 \Leftrightarrow \sum_{l=0}^{n-1} r_l \cdot c_{i,l} \underset{\substack{\text{'C}_j\text{'}}}{<} \sum_{l=0}^{n-1} r_l \cdot c_{j,l} \underset{\substack{\text{'C}_i\text{'}}}{>} \sum_{l=0}^{n-1} r_l \cdot c_{j,l}$$

# ML decoding procedure - Summary

- a. Compute the distance (Hamming or Euclidean) between the received word and ALL codewords.
- b. Choose the codeword corresponding to the minimum distance → Can be impractical since there are  $2^k$  distances to compute.

ML decoding provides optimal decoding performance, but is inherently too complex to implement for practical codes for which  $k$  is often large (long blocks of info bits).

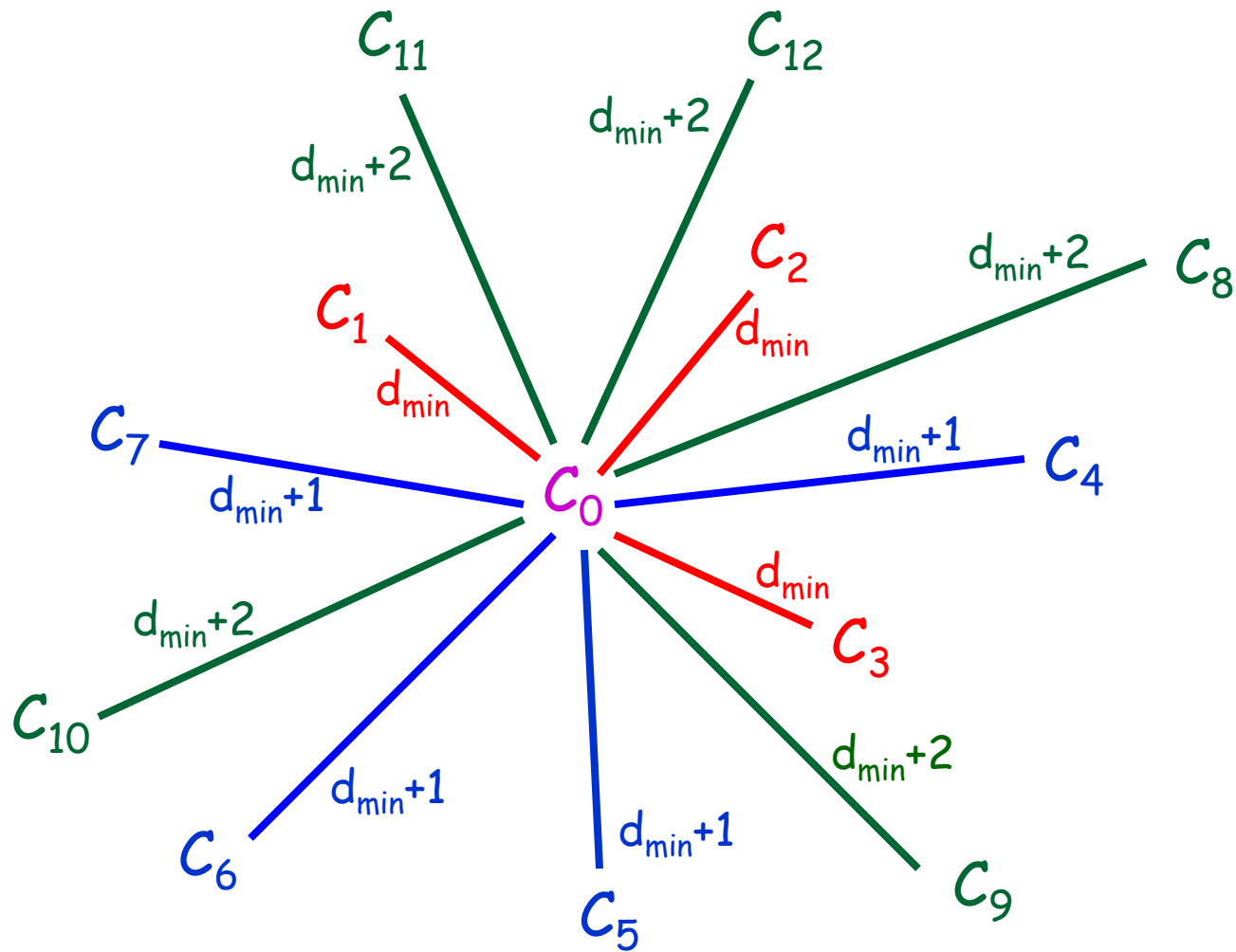
# ML decoding procedure - Summary

If ML decoding is not practical, we must use other (sub-optimal) decoding algorithms. In some cases, the performance of those algorithms is close to that of ML decoding (iterative decoding for instance).

For codes whose operation can be described using a trellis, a practical ML decoding algorithm, called Viterbi algorithm, does actually exist.

The Viterbi algorithm can operate both in hard and soft-decision modes.

# Pictorial view of a code



# Pictorial view of a code

There are:

- 3 codewords at minimal distance  $d_{\min}$  from  $C_0$ ;
- 4 codewords at  $(d_{\min}+1)$  from  $C_0$ ;
- 5 codewords at  $(d_{\min}+2)$  from  $C_0$ .

$C_1$ ,  $C_2$  and  $C_3$  are nearest neighbors of  $C_0$ .

We need to understand this pictorial view to determine the expression of the bit error probability at the decoder output, for a given block code.

# Bit error probability

Assume that the all-zero codeword  $C_0$  is transmitted.

Bit error probability after decoding:

$$P_{eb} = \Pr\{\underbrace{\text{erroneous decoded bit, wrong detection}}_{\text{edb}}\}$$

$$= \Pr\{(\text{edb}, C_0 \rightarrow C_1) \cup \dots \cup (\text{edb}, C_0 \rightarrow C_{2^k-1})\}$$

$$\leq \sum_{i=1}^{2^k-1} \Pr\{\text{edb}, C_0 \rightarrow C_i\}$$

Union Bound

# Bit error probability

$$\Pr\{\text{edb}, C_0 \rightarrow C_i\} = \Pr\{\text{edb} | C_0 \rightarrow C_i\} \Pr\{C_0 \rightarrow C_i\}$$

Only depends on the Hamming distance between  $C_0$  and  $C_i$

Grouping together terms associated with codewords being at the same distance  $d$  from  $C_0$ , we obtain:

$$\sum_{i=1}^{2^k-1} \Pr\{\text{edb} | C_0 \rightarrow C_i\} \Pr\{C_0 \rightarrow C_i\} = \sum_{d=d_{\min}}^{+\infty} \frac{w_d}{k} P_d$$

# Bit error probability

We might want to consider an example to better understand this equation.

Consider our ( $n = 5, k = 2$ ) linear block code.

$$\begin{aligned} \sum_{i=1}^{2^k-1} \Pr\{\text{edb} | C_0 \rightarrow C_i\} \Pr\{C_0 \rightarrow C_i\} = \\ \Pr\{\text{edb} | C_0 \rightarrow C_1\} \Pr\{C_0 \rightarrow C_1\} + \dots \\ \dots + \Pr\{\text{edb} | C_0 \rightarrow C_2\} \Pr\{C_0 \rightarrow C_2\} + \dots \\ \dots + \Pr\{\text{edb} | C_0 \rightarrow C_3\} \Pr\{C_0 \rightarrow C_3\} \end{aligned}$$



# Bit error probability

$$C_0 = (00000)$$

$$C_1 = (01011)$$

$$C_2 = (10101)$$

$$C_3 = (11110)$$

M		C				
$m_0$	$m_1$	$c_0$	$c_1$	$c_2$	$c_3$	$c_4$
0	0	0	0	0	0	0
0	1	0	1	0	1	1
1	0	1	0	1	0	1
1	1	1	1	1	1	0

$$\begin{aligned}
 &= [\Pr\{\text{edb}|C_0 \rightarrow C_1\} + \Pr\{\text{edb}|C_0 \rightarrow C_2\}]P_{d=3} + \dots \\
 &\quad \dots + \Pr\{\text{edb}|C_0 \rightarrow C_3\}P_{d=4} \\
 &= \left[\frac{1}{2} + \frac{1}{2}\right]P_{d=3} + \frac{2}{2}P_{d=4} = \left[\frac{1+1}{2}\right]P_{d=3} + \frac{2}{2}P_{d=4} = \sum_{d=3}^4 \frac{w_d}{k}P_d
 \end{aligned}$$

# Bit error probability

The expression of the union bound becomes:

$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} \frac{w_d}{k} P_d$$

- $P_d$ : Probability of decoding a codeword which is at distance  $d$  from  $C_0$ ;
- $w_d$ : Total Hamming weight of the messages associated with the codewords at distance  $d$  from  $C_0$ .

# Compute $P_d$ for an AWGN, BPSK channel

From now on, we are going to focus on the AWGN, BPSK channel, and forget the BSC.

Why?

Decoders perform better when they operate on an AWGN, BPSK channel, i.e. using analog samples (soft decisions), rather than over an equivalent BSC, i.e. using bits (hard decisions).

This will become very clear later.

# Compute $P_d$ for an AWGN, BPSK channel

$$P_d = \Pr\{C_0 \rightarrow C_i | d_H(C_0, C_i) = d\}$$

$$= \Pr\left\{\sum_{l=0}^{n-1} r_l \cdot c_{i,l} > \sum_{l=0}^{n-1} r_l \cdot c_{0,l} \mid d_H(C_0, C_i) = d\right\}$$

with  $\begin{cases} C_i = (c_{i,1}, c_{i,2}, \dots, c_{i,n-1}) \\ C_0 = (c_{0,1}, c_{0,2}, \dots, c_{0,n-1}) = (-1, -1, \dots, -1) \end{cases}$

$$P_d = \Pr\left\{\sum_{l=0}^{n-1} r_l \cdot (c_{i,l} - c_{0,l}) > 0 \mid d_H(C_0, C_i) = d\right\}$$

$c_{i,l} = c_{0,l}$  in  $(n-d)$  positions &  $c_{i,l} = -c_{0,l}$  in  $d$  positions

## Compute $P_d$ for an AWGN, BPSK channel

$$\begin{aligned} P_d &= \Pr\left\{\sum_{l=1}^d -r_l \cdot (2c_{0,l}) > 0\right\} = \Pr\left\{\sum_{l=1}^d r_l > 0\right\} \\ &= \Pr\left\{\sum_{l=1}^d (-1 + n_l) > 0\right\} = \Pr\left\{\sum_{l=1}^d n_l > d\right\} \end{aligned}$$

$\sum_{l=1}^d n_l$  is a Gaussian sample, with zero-mean, and

variance equal to  $\sigma^2 = d \times \frac{1}{2} \left( R_c \frac{E_b}{N_0} \right)^{-1}$

# Compute $P_d$ for an AWGN, BPSK channel

Therefore, 
$$P_d = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left\{-\frac{x^2}{2\sigma^2}\right\} dx$$

Let 
$$u = \frac{x}{\sqrt{2\sigma^2}} \quad dx = du \cdot \sqrt{2\sigma^2}$$

$$P_d = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left\{-u^2\right\} \cdot du \cdot \sqrt{2\sigma^2}$$

# Compute $P_d$ for an AWGN, BPSK channel

$$P_d = \frac{1}{\sqrt{\pi}} \int_{\frac{d}{\sqrt{2\sigma^2}}}^{+\infty} \exp\{-u^2\} \cdot du = \frac{1}{2} \operatorname{erfc}\left(\sqrt{\frac{d^2}{2\sigma^2}}\right)$$

with  $\frac{d^2}{2\sigma^2} = \frac{d^2}{d R_c^{-1}\left(\frac{E_b}{N_0}\right)^{-1}} = d R_c \left(\frac{E_b}{N_0}\right)$

$$\Rightarrow P_d = \frac{1}{2} \operatorname{erfc}\left(\sqrt{d R_c \frac{E_b}{N_0}}\right)$$

# Bit error probability over AWGN, BPSK channel

Finally, 
$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} \frac{w_d}{2k} \cdot \operatorname{erfc} \left( \sqrt{dR_c \frac{E_b}{N_0}} \right)$$

At high SNR, the dominant term in the sum is the term corresponding to the smallest value of  $d$  for the code, i.e. the minimal Hamming distance  $d_{\min}$ .



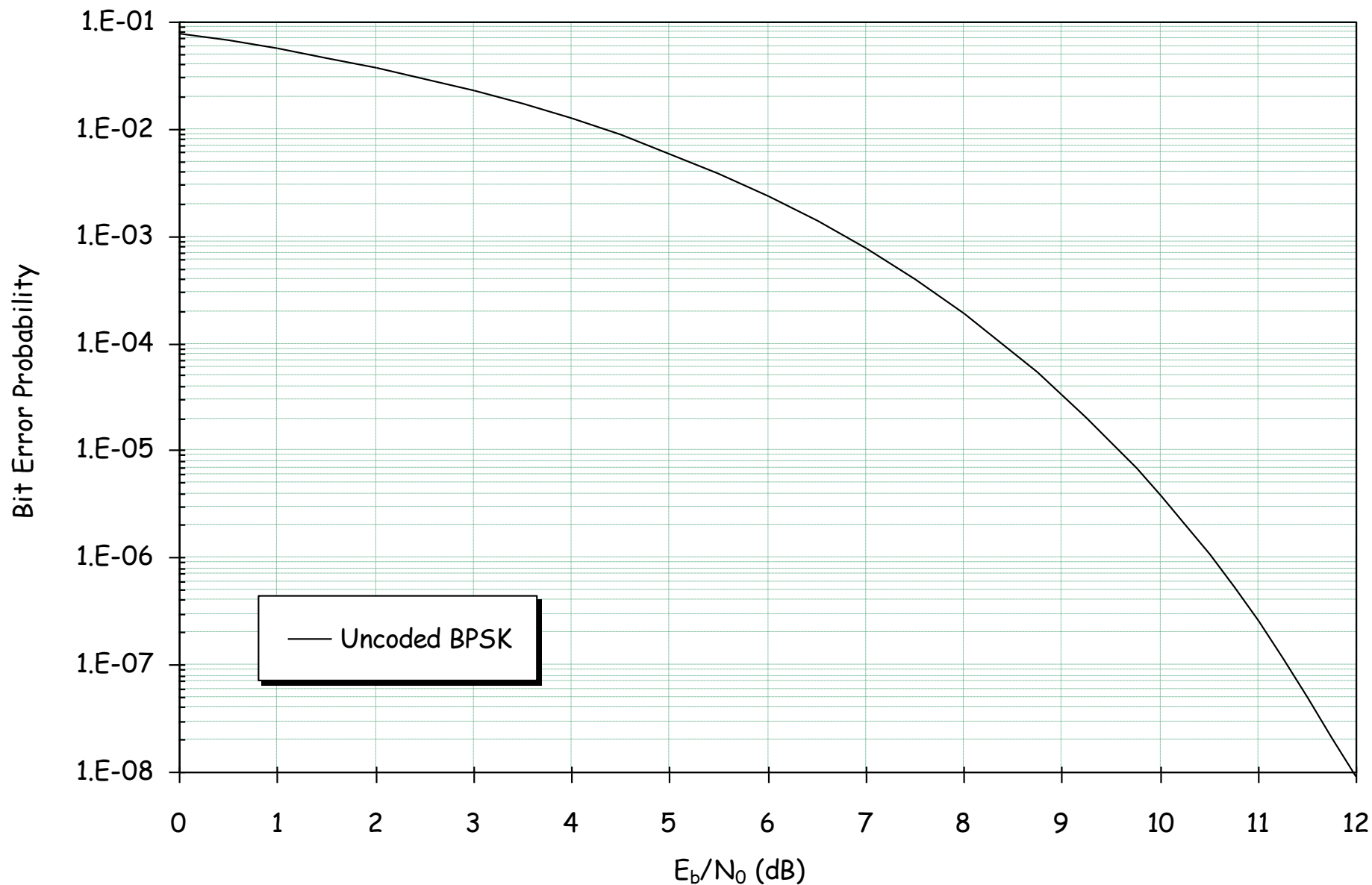
# Bit error probability over AWGN, BPSK channel

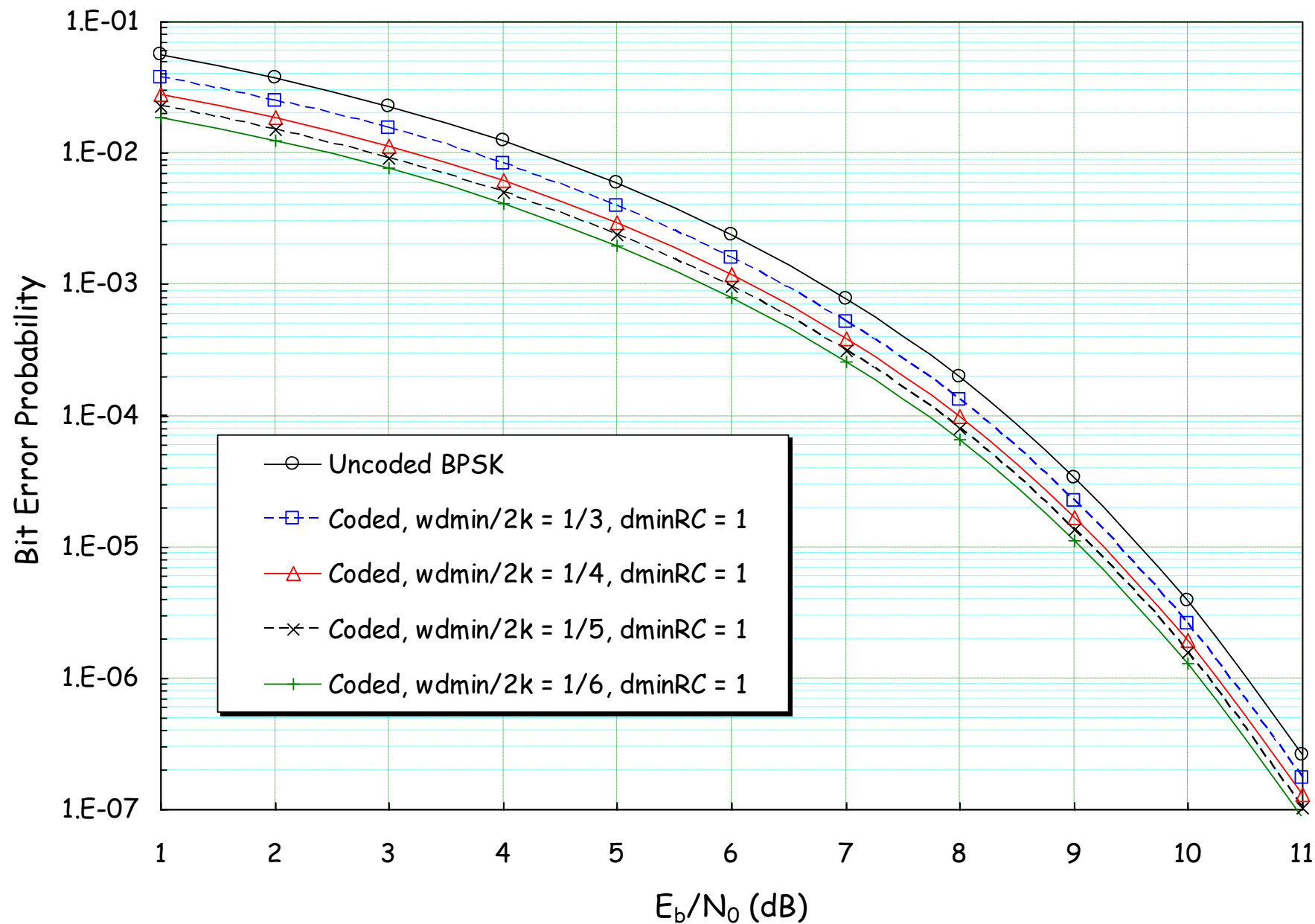
At high SNR: 
$$P_{eb} \approx \frac{w_{d_{\min}}}{2k} \cdot \operatorname{erfc} \left( \sqrt{d_{\min} R_c \frac{E_b}{N_0}} \right)$$

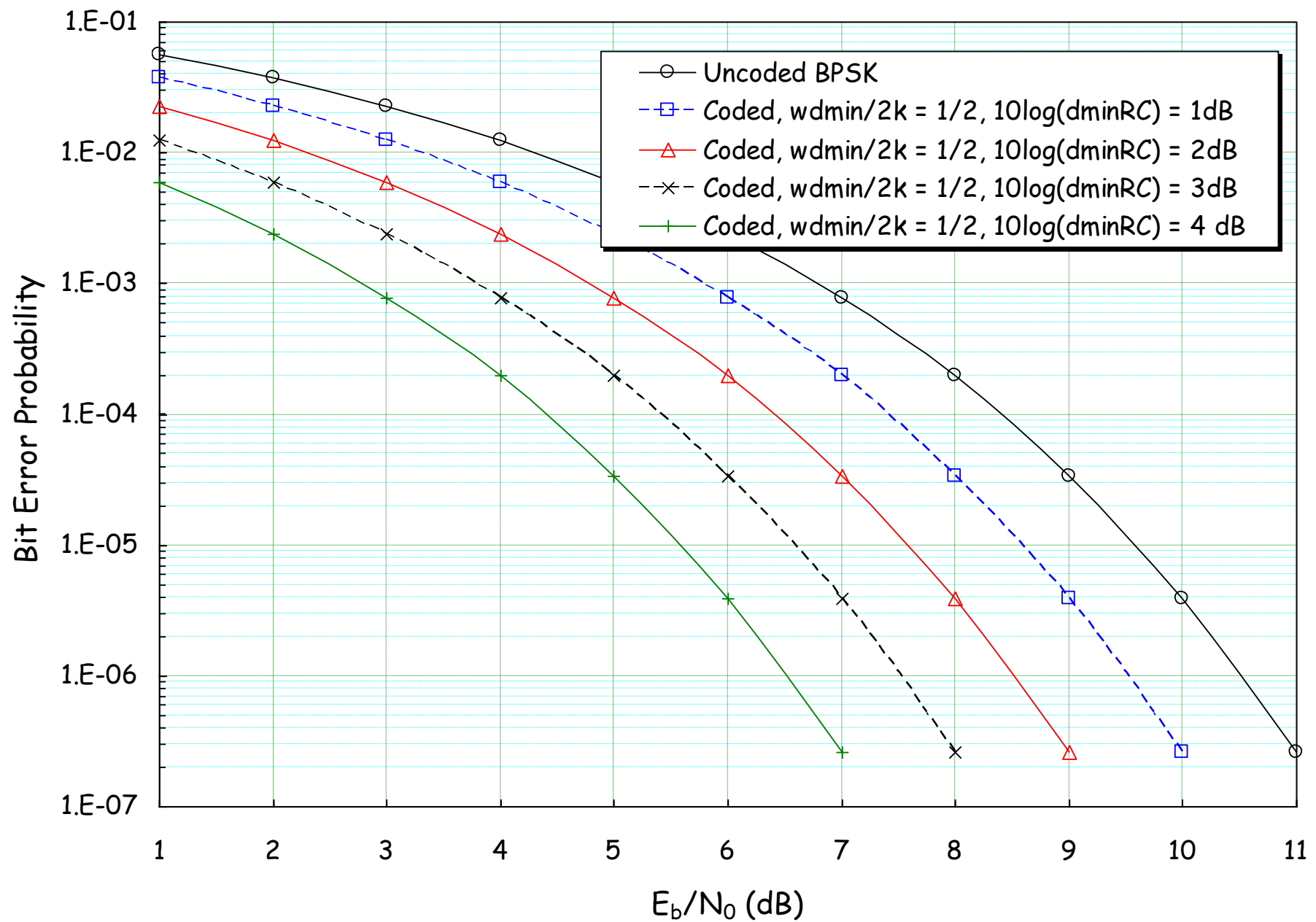
For uncoded BPSK: 
$$P_{eb} = \frac{1}{2} \operatorname{erfc} \left( \sqrt{\frac{E_b}{N_0}} \right)$$

At high SNR, the error probability  $P_{eb}$  is mainly dependent on the argument inside the  $\operatorname{erfc}(\cdot)$  function.

At high SNR, the error coefficient ( $w_{d_{\min}}/2k$  and  $\frac{1}{2}$ , above) has a negligible effect on the value of  $P_{eb}$ .







# Bit error probability over AWGN, BPSK channel

Comparison between coded and uncoded systems at high SNR:

$$P_{eb, \text{uncoded}} = P_{eb, \text{coded}}$$

$$\Rightarrow \left( \frac{E_b}{N_0} \right)_u \approx d_{\min} R_c \left( \frac{E_b}{N_0} \right)_c$$

In decibels (dB),

$$\left( \frac{E_b}{N_0} \right)_u - \left( \frac{E_b}{N_0} \right)_c \approx 10 \cdot \log_{10} (d_{\min} R_c)$$

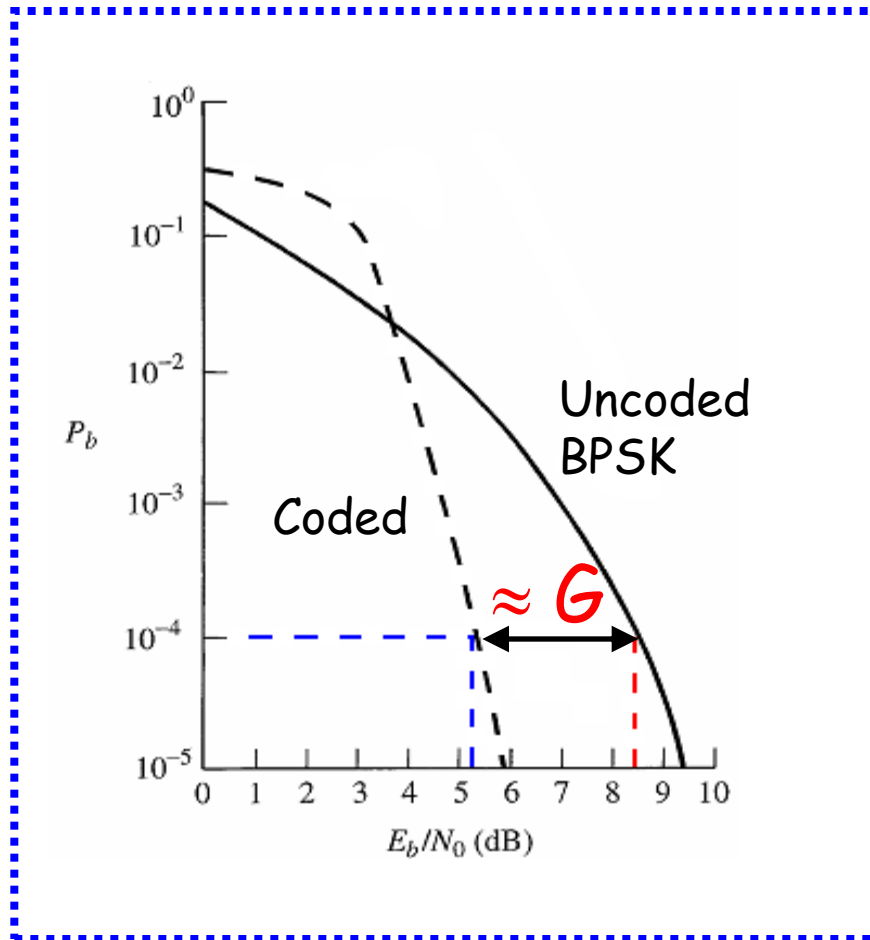
# Bit error probability over AWGN, BPSK channel

This result shows that, at high SNR, both  $P_{eb}$  curves are actually “parallel”.

The coded system can provide the same  $P_{eb}$  as uncoded BPSK with a SNR which is  $10 \cdot \log_{10}(d_{\min} R_c)$  dB smaller. The coding gain at high SNR, also called “asymptotic coding gain”, is thus given by

$$G = 10 \cdot \log_{10}(d_{\min} R_c)$$

# Bit error probability over AWGN, BPSK channel



An asymptotic coding gain value of  $10 \cdot \log_{10}(d_{\min} R_c)$  can only be achieved using ML decoding.

Without ML decoding:  
 $G < 10 \cdot \log_{10}(d_{\min} R_c)$ .

# Bit error probability over AWGN, BPSK channel

The “error coefficients”  $w_d/2k$  are to be taken into account at lower SNR. At high SNR, the critical parameter is the product  $d_{\min} \cdot R_c$ .

In order to obtain a “good” code:

1. Maximise the product  $d_{\min} \cdot R_c$  in order to maximise the asymptotic coding gain over uncoded BPSK.

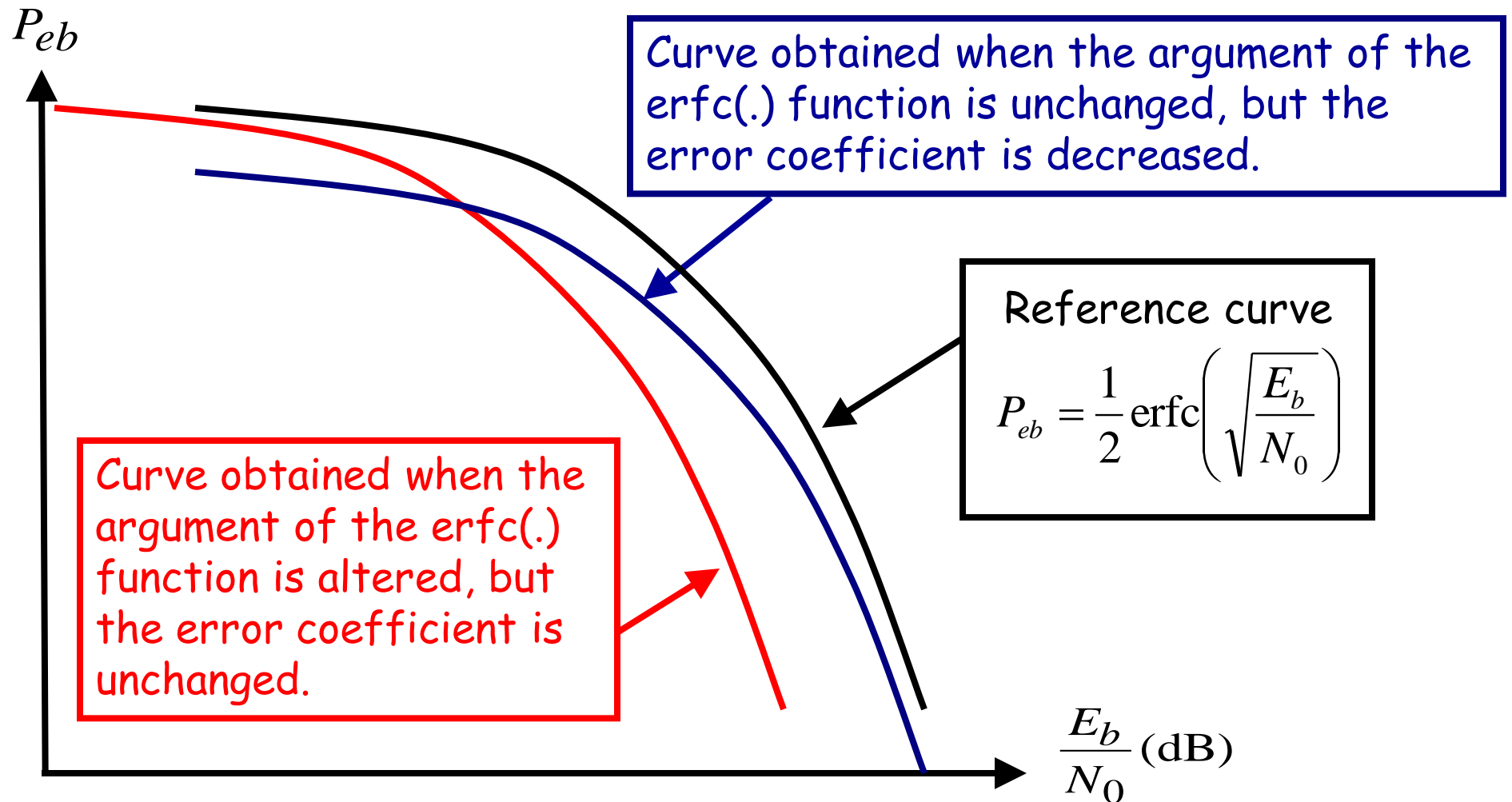


# Bit error probability over AWGN, BPSK channel

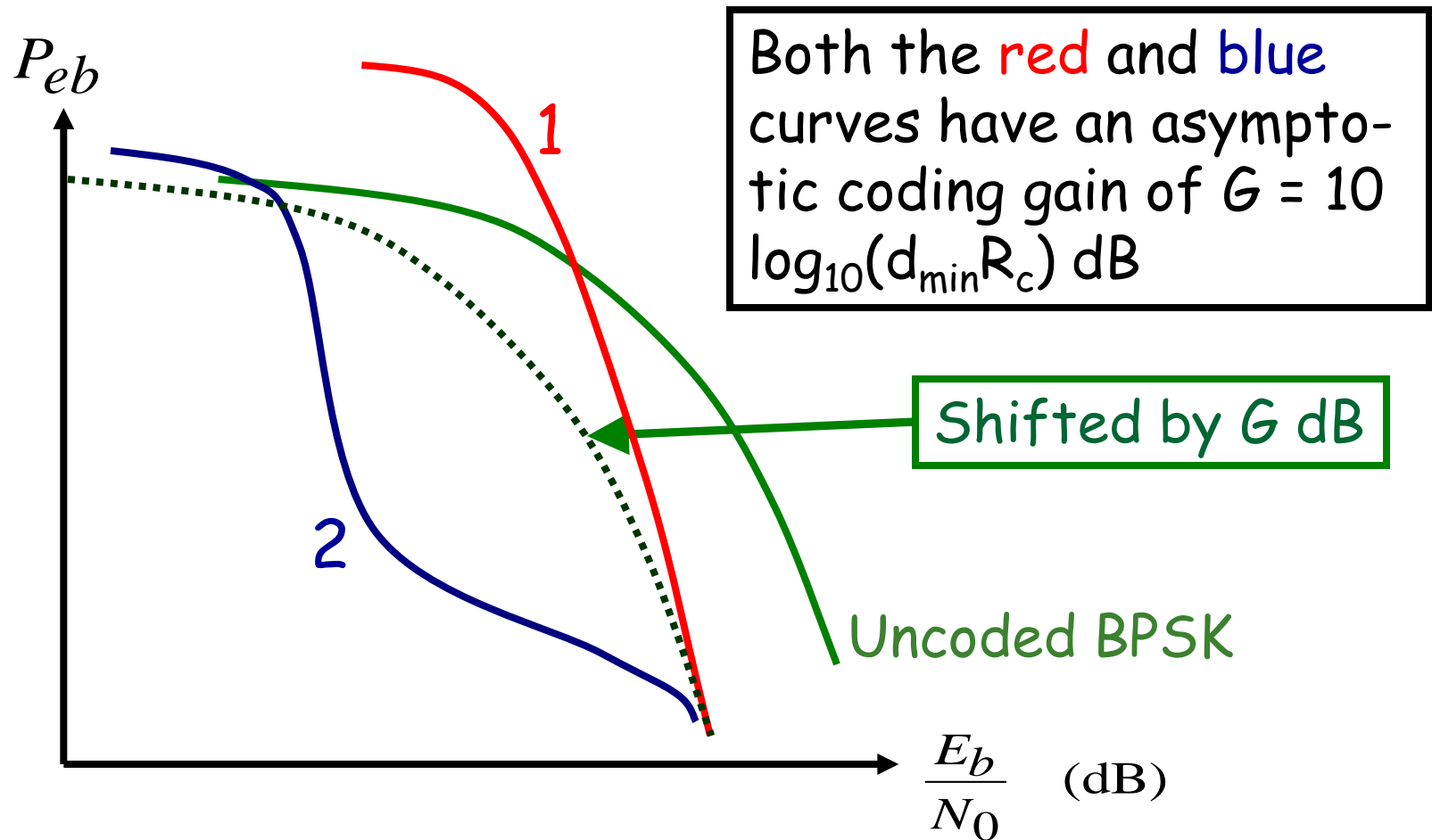
2. For a given value of  $k$ , minimise the parameters  $w_d$ , especially that corresponding to the minimal Hamming distance  $d_{\min}$ .

=> Minimise the number of nearest neighbours of a given codeword.

# Bit error probability over AWGN, BPSK channel

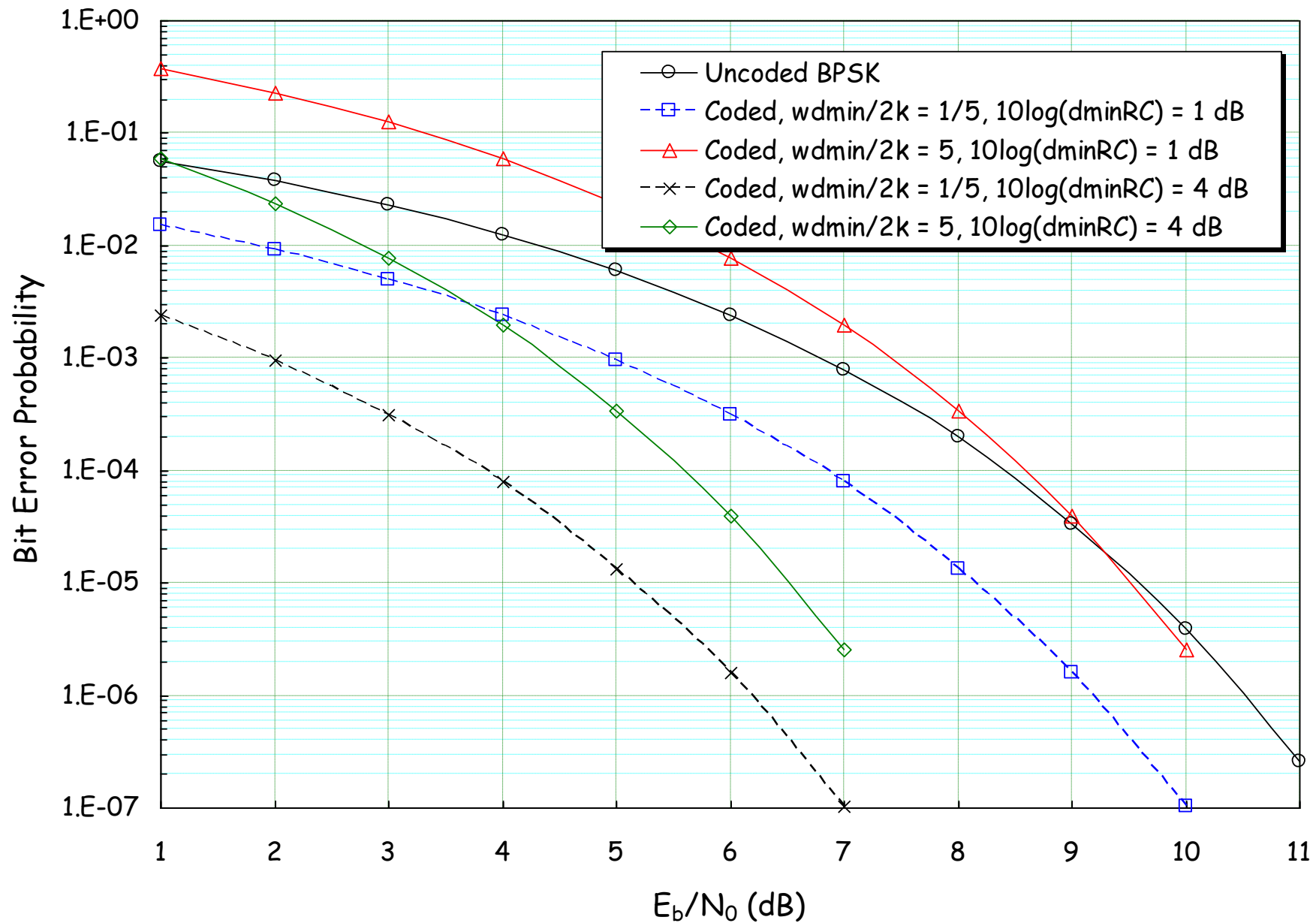


# Bit error probability over AWGN, BPSK channel



# Bit error probability over AWGN, BPSK channel

1. Obtained when the "error coefficient" values are large ( $\gg \frac{1}{2}$ )
  - The asymptotic coding gain  $G = 10\log_{10}(d_{\min}R_c)$  is not reached at error probabilities of practical interest.
2. Obtained when the "error coefficient" values are small ( $\ll \frac{1}{2}$ )
  - The coding gain achieved at low SNR is higher than the asymptotic coding gain.



# Example 1

$(n = 7, k = 4, d_{\min} = 3)$  Hamming code

M				C						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	0	1
0	0	1	0	0	0	1	0	1	1	1
0	0	1	1	0	0	1	1	0	1	0
0	1	0	0	0	1	0	0	0	1	1
0	1	0	1	0	1	0	1	1	1	0
0	1	1	0	0	1	1	0	1	0	0
0	1	1	1	0	1	1	1	0	0	1

M				C						
1	0	0	0	1	0	0	0	1	1	0
1	0	0	1	1	0	0	1	0	1	1
1	0	1	0	1	0	1	0	0	0	1
1	0	1	1	1	0	1	1	1	0	0
1	1	0	0	1	1	0	0	1	0	1
1	1	0	1	1	1	0	1	0	0	0
1	1	1	0	1	1	1	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1

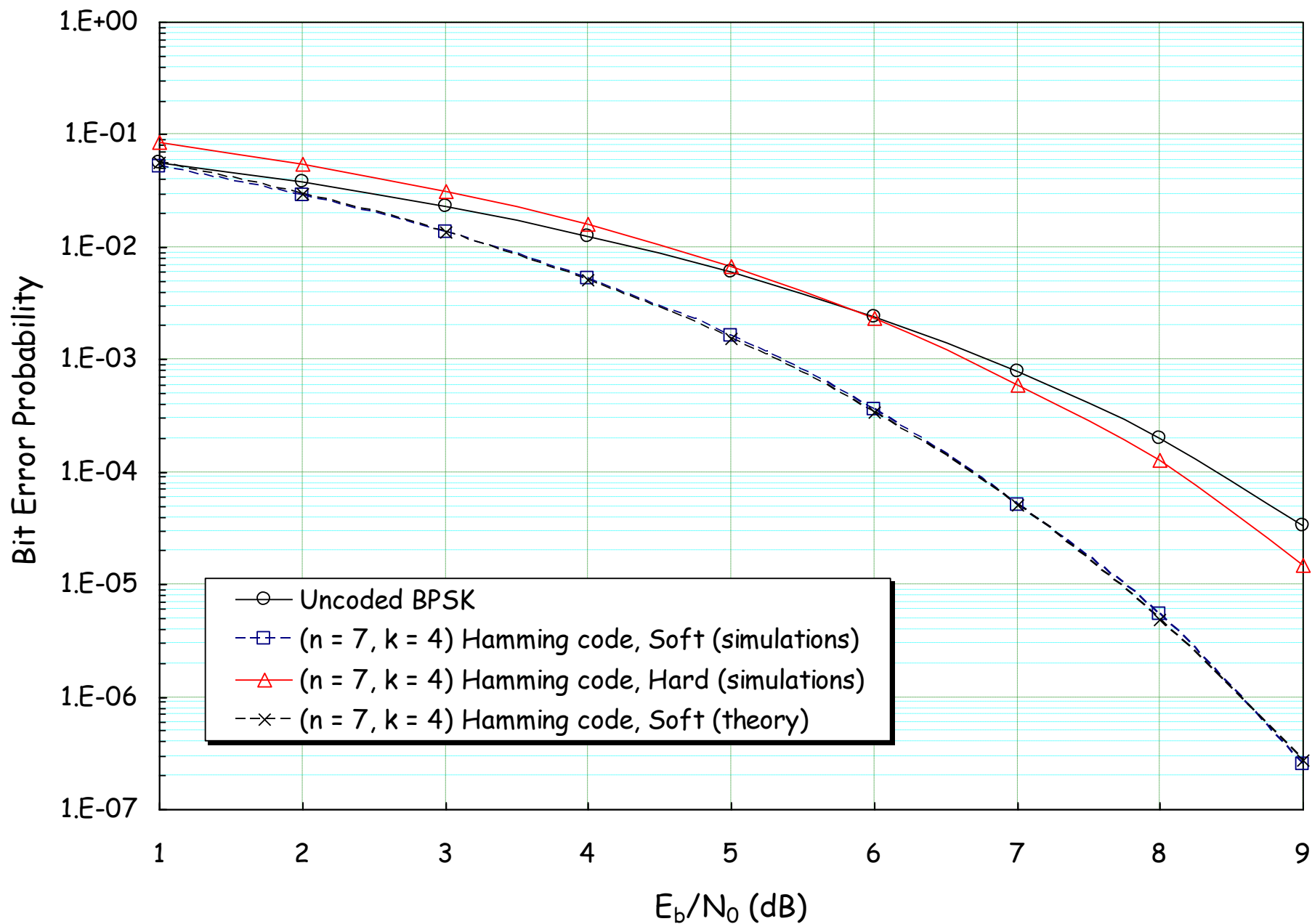
7 codewords at distance  $d_{\min} = 3$  from  $C_0 \rightarrow w_{d\min} = 12$

# Example 1

$$P_{eb} \approx \frac{12}{2 \times 4} \operatorname{erfc} \left[ \sqrt{\frac{12}{7} \frac{E_b}{N_0}} \right] = \frac{3}{2} \operatorname{erfc} \left[ \sqrt{\frac{12}{7} \frac{E_b}{N_0}} \right]$$

Asymptotic coding gain  $G = 2.34$  dB.

But, are you sure ML decoding can be implemented?





## Example 2

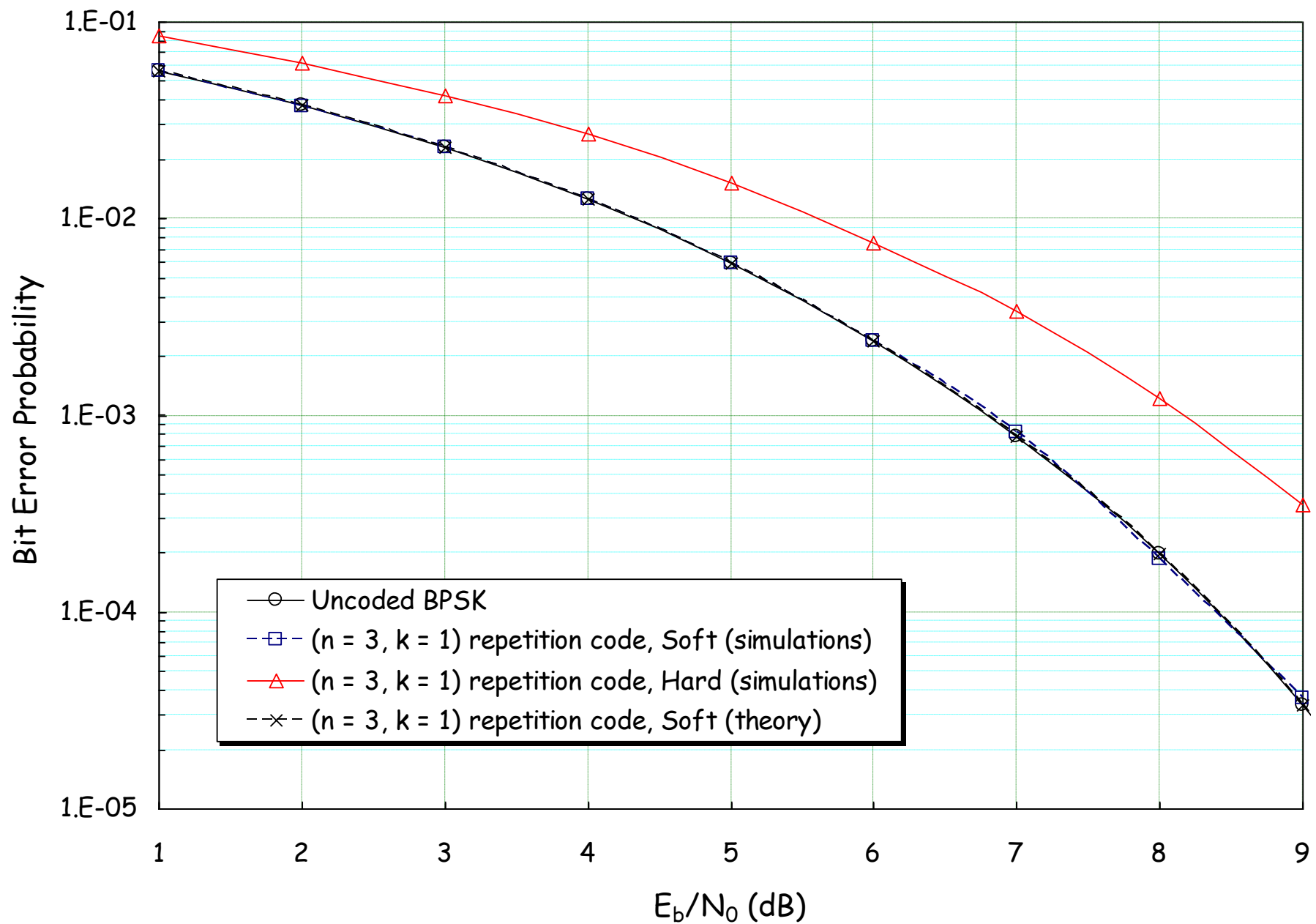
$(n = 3, k = 1, d_{\min} = 3)$  repetition code

M	C
0	000
1	111

1 codeword at distance  $d_{\min} = 3$   
from  $C_0 \rightarrow w_{d\min} = 1$

$$P_{eb} \approx \frac{1}{2} \operatorname{erfc} \left[ \sqrt{\frac{E_b}{N_0}} \right]$$

No coding gain over BPSK!



## Example 3

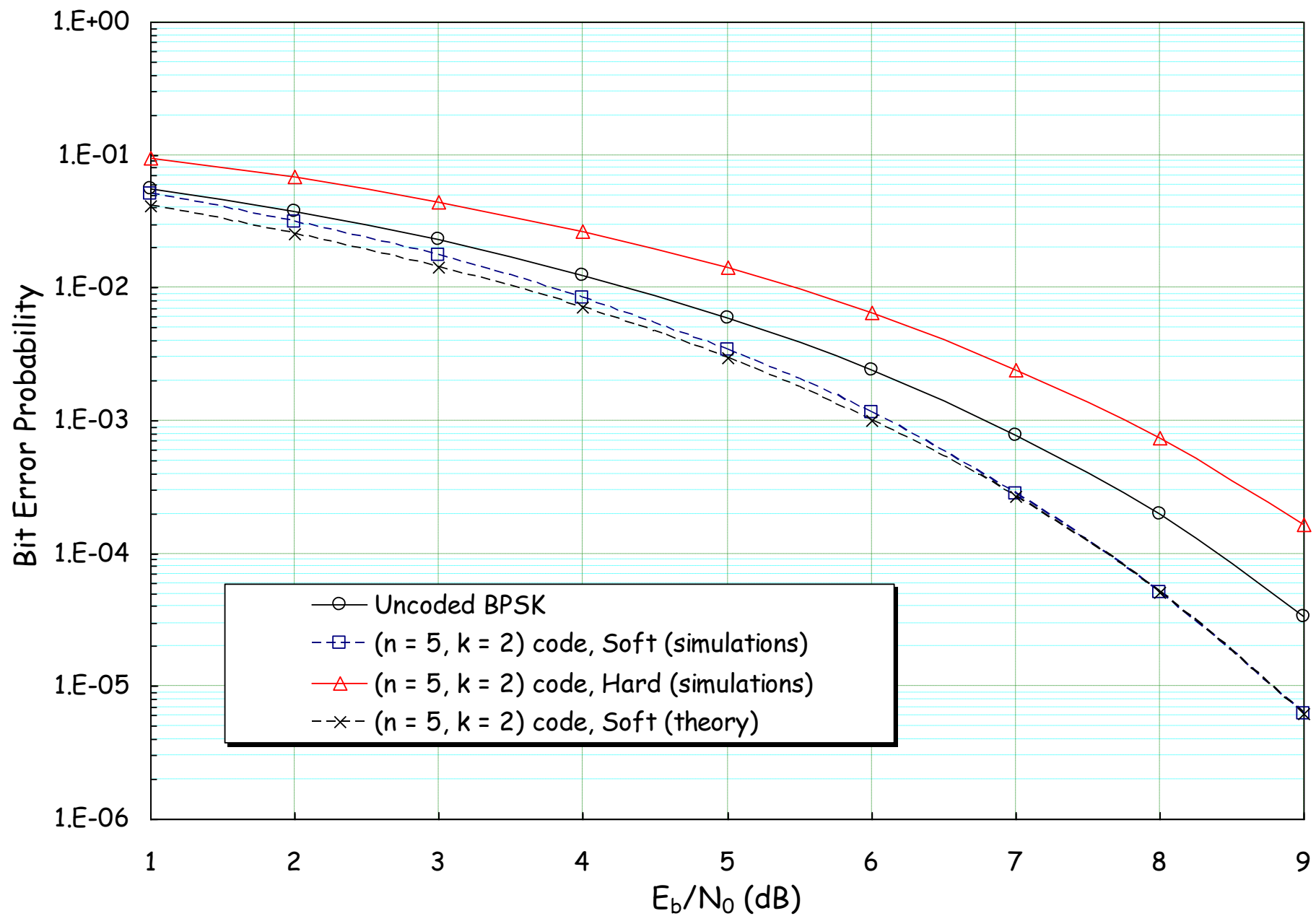
Our  $(n = 5, k = 2, d_{\min} = 3)$  code

M	C
00	00000
01	01011
10	10101
11	11110

2 codewords at distance  $d_{\min} = 3$  from  $C_0$   
 $\rightarrow w_{d\min} = 2$

$$P_{eb} \approx \frac{1}{2} \operatorname{erfc} \left[ \sqrt{\frac{6}{5} \frac{E_b}{N_0}} \right]$$

Asymptotic coding gain  $G = 0.79$  dB.



## Example 4

$(n = 3, k = 2, d_{\min} = 2)$  parity-check code

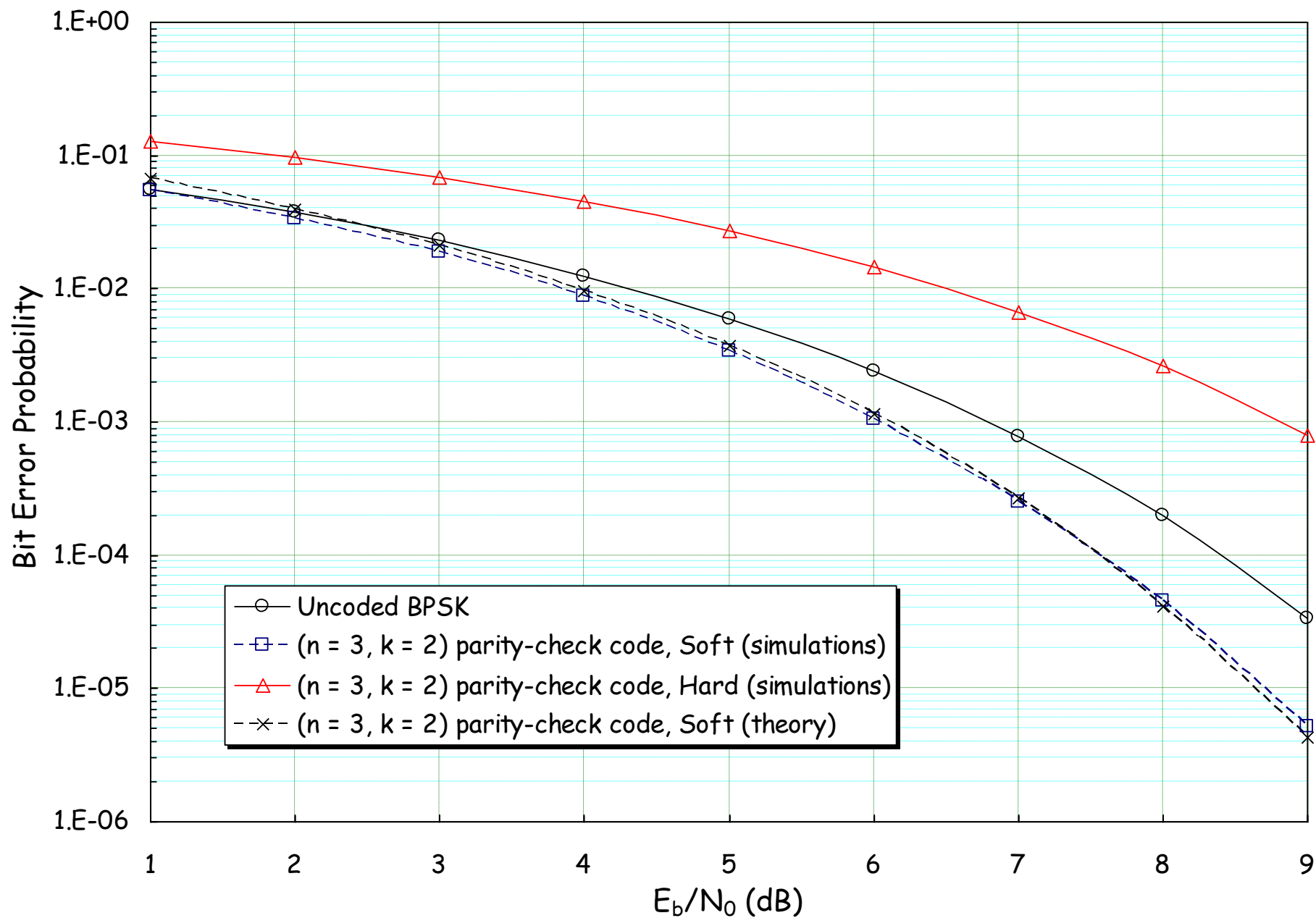
M	C
00	000
01	011
10	101
11	110

2 codewords at distance  $d_{\min} = 2$  from  $C_0$   
 $\rightarrow w_{d\min} = 4$

$$P_{eb} \approx \operatorname{erfc} \left[ \sqrt{\frac{4}{3} \frac{E_b}{N_0}} \right]$$

Asymptotic coding gain  $G = 1.25$  dB.

-> Parity-check codes can actually correct transmission errors over BPSK, AWGN channels ( $\neq$  BSC).



## Example 5

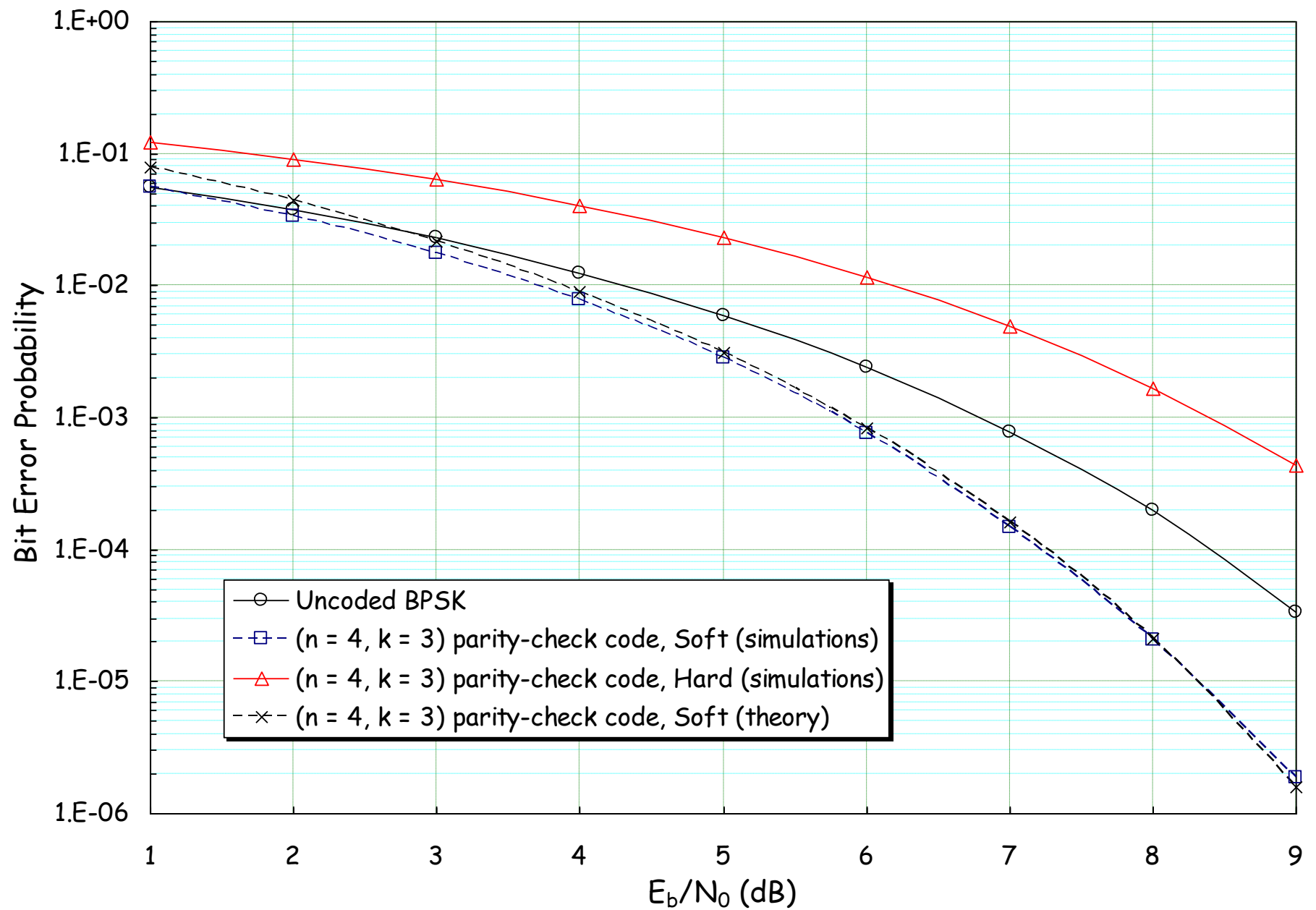
( $n = 4, k = 3, d_{\min} = 2$ ) parity-check code

M	C
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

6 codewords at distance  $d_{\min} = 2$  from  $C_0$   
 $\rightarrow w_{d\min} = 9$

$$P_{eb} \approx \frac{3}{2} \operatorname{erfc} \left[ \sqrt{\frac{3}{2} \frac{E_b}{N_0}} \right]$$

Asymptotic coding gain  $G = 1.76$  dB.





## Example 6

$(n = 5, k = 4, d_{\min} = 2)$  parity-check code

M				C				
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	0	1
0	0	1	1	0	0	1	1	0
0	1	0	0	0	1	0	0	1
0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0
0	1	1	1	0	1	1	1	1

M				C				
1	0	0	0	1	0	0	0	1
1	0	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0	0
1	0	1	1	1	0	1	1	1
1	1	0	0	1	1	0	0	0
1	1	0	1	1	1	0	1	1
1	1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1	0

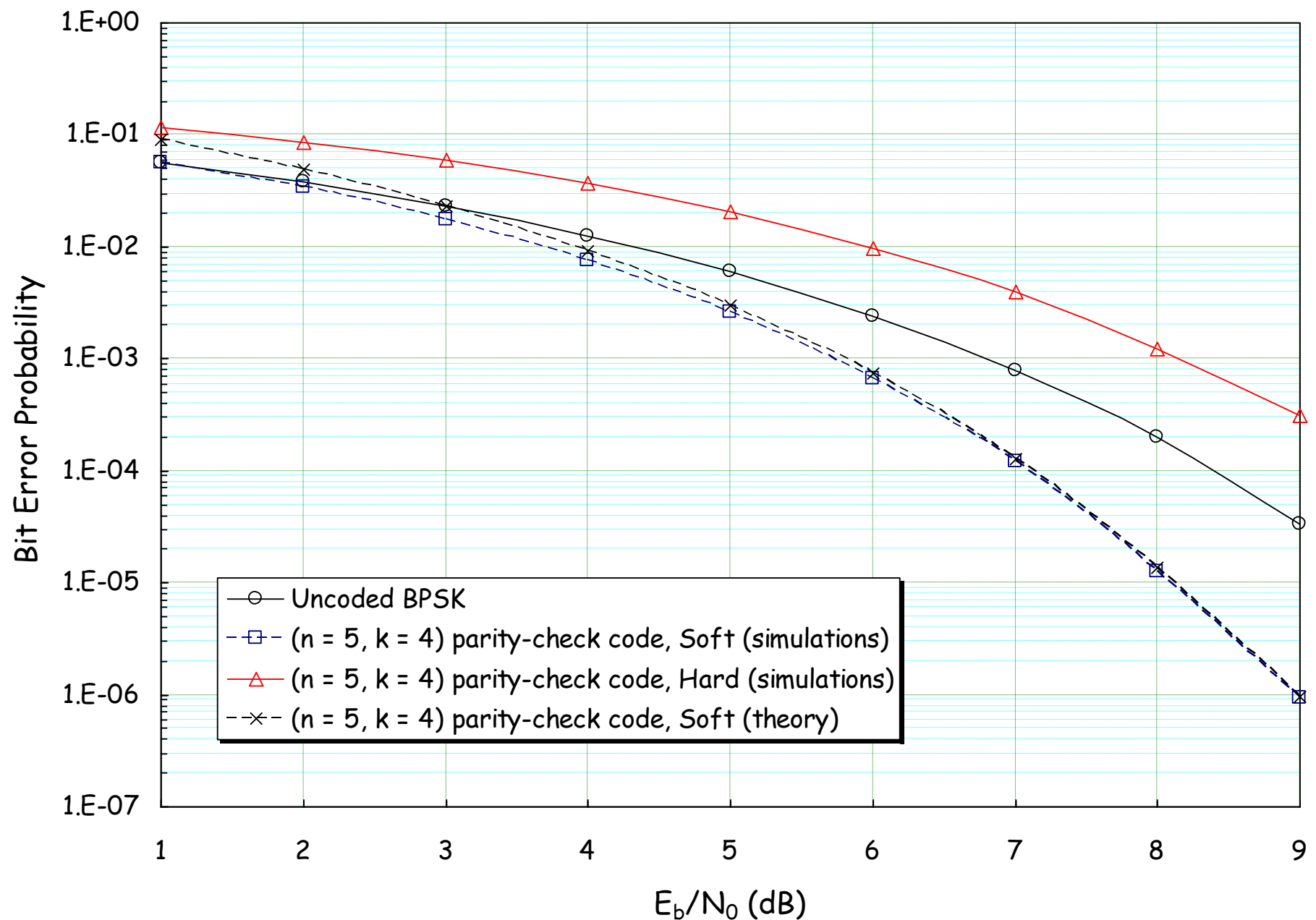
10 codewords at distance  $d_{\min} = 2$  from  $C_0 \rightarrow w_{d_{\min}} = 16$

## Example 6

$$P_{eb} \approx \frac{16}{2 \times 4} \operatorname{erfc} \left[ \sqrt{\frac{8}{5} \frac{E_b}{N_0}} \right] = 2 \operatorname{erfc} \left[ \sqrt{\frac{8}{5} \frac{E_b}{N_0}} \right]$$

Asymptotic coding gain  $G = 2.04$  dB.

As  $k \rightarrow \infty$ ,  $G \rightarrow 10 \log(2) = 3.01$  dB, but the error coefficient values are increasing fast  $\rightarrow$  This 3-dB asymptotic coding gain value is not achieved at error probabilities of practical interest.

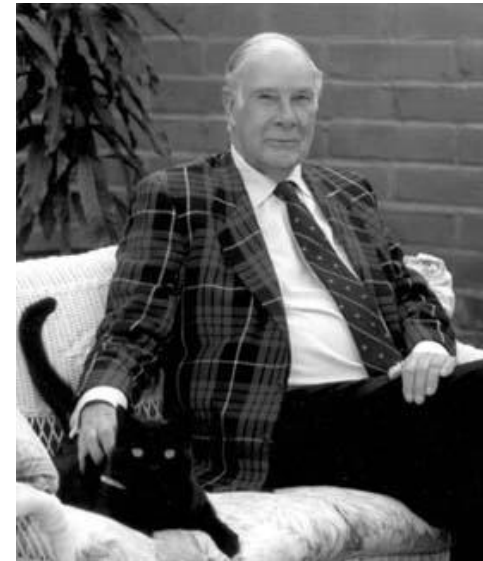
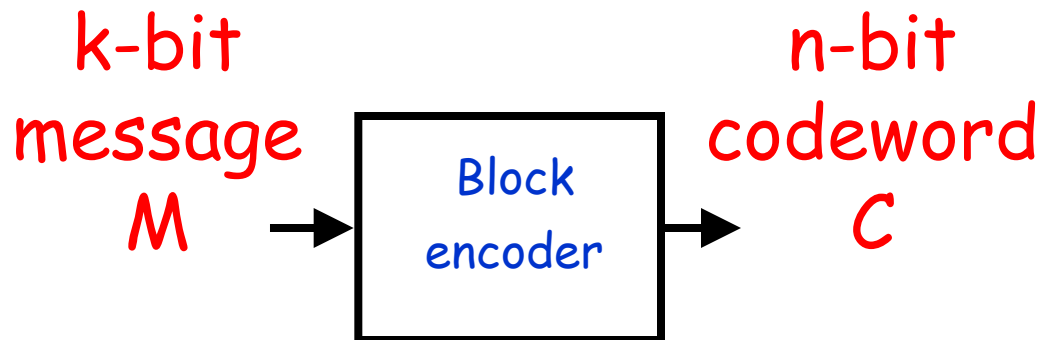


# Part 8

## Practical Error-Correcting Codes (1948-1993)

# Hamming codes (1950)

1950: Richard W Hamming introduces the first ever class of linear block codes, applied even today → Hamming codes.



Richard W Hamming (1915 - 1998)

# Hamming codes

For any integer  $m \geq 3$ , there is a Hamming code, with  $n = 2^m - 1$  and  $k = 2^m - 1 - m$ , so that one error can be corrected in a received word of  $n$  bits ( $t = 1$ ,  $d_{\min} = 3$ ).

- $n = 7$ ,  $k = 4$  ( $R_c \approx 0.57$ )
- $n = 15$ ,  $k = 11$  ( $R_c \approx 0.73$ )
- $n = 31$ ,  $k = 26$  ( $R_c \approx 0.84$ )
- $n = 63$ ,  $k = 57$  ( $R_c \approx 0.90$ )
- $n = 127$ ,  $k = 120$  ( $R_c \approx 0.94$ ), and so on.

Hamming codes are decoded using the syndrome decoding algorithm which is quite simple to implement.

# Hamming codes

## Encoding table for the (7, 4) Hamming code

M				C			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1

M				C			
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	1
1	1	0	0	1	1	0	0
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1

# Hamming codes

Hamming codes are linear codes: They are defined by a generator matrix  $G$  so that  $C = M.G$ .

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{aligned} C_0 &= m_0 \\ C_1 &= m_1 \\ C_2 &= m_2 \\ C_3 &= m_3 \\ C_4 &= m_0 + m_2 + m_3 \\ C_5 &= m_0 + m_1 + m_2 \\ C_6 &= m_1 + m_2 + m_3 \end{aligned}$$

The code is completely defined by a set of  $n$  linear equations

$C_0, C_1, C_2$ , and  $C_3$ : info bits -  $C_4, C_5, C_6$ : parity bits.



# Hamming codes

Another possible representation for a linear code:  
The parity-check matrix  $H$ .

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

A binary word  $W$  of  $n = 7$  bits is a codeword if and only if  $w \cdot H^T = 0$ .

# Hamming codes

All codewords  $C$  are such that  $C.H^T = 0$

$$(c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6) \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}^T = (000)$$

In other words, all codewords  $C$  satisfy the equations:

$$C_0 + C_2 + C_3 + C_4 = 0$$

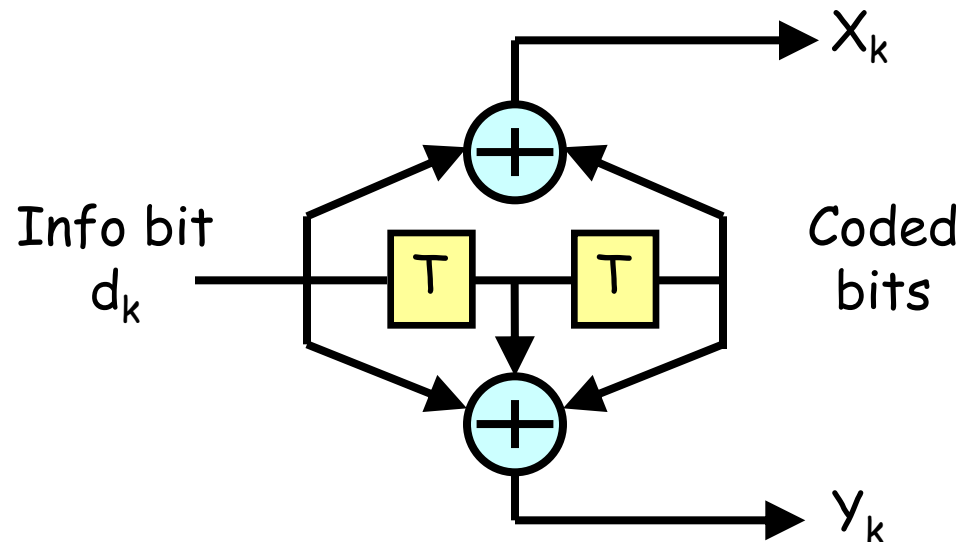
$$C_0 + C_1 + C_2 + C_5 = 0$$

$$C_1 + C_2 + C_3 + C_6 = 0$$

# Convolutional codes (1955)

1955: Peter Elias introduces the concept of convolutional codes (CCs) in which the stream of info bits is encoded by a finite-state machine.

Convolutional encoder,  $R_c = \frac{1}{2}$ , constraint length  $K = 3$ , generator polynomials 5 and 7

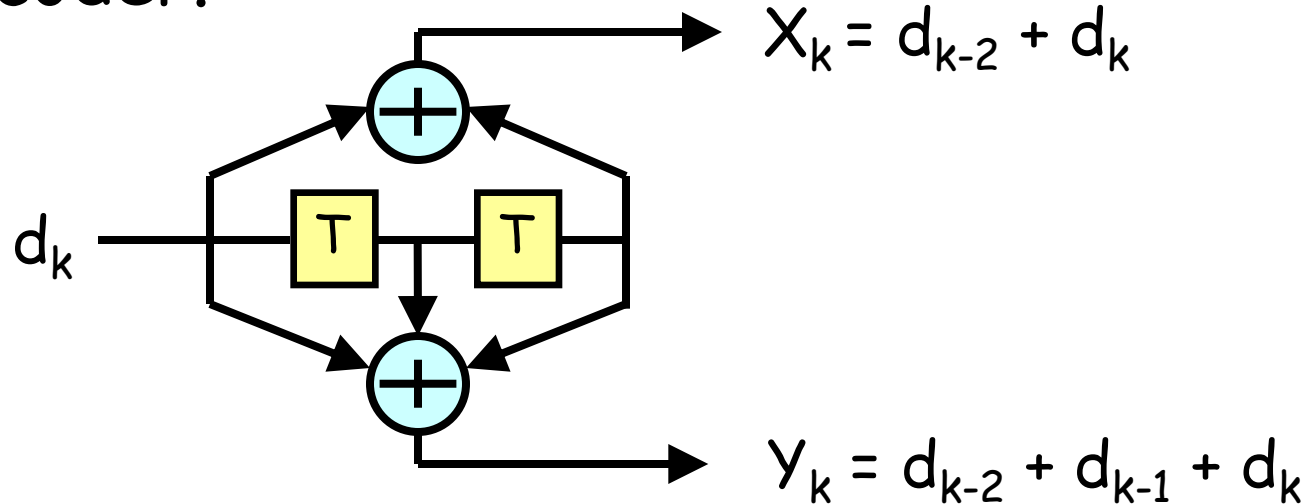


Peter Elias (1923 - 2001)

# Convolutional codes

The constraint length  $K$  is the number of info bits that are taken into account in the computation of the coded bits.

Since  $X_k = d_{k-2} + d_k$  and  $Y_k = d_{k-2} + d_{k-1} + d_k$ , we find  $K = 3$  for our encoder.



# Convolutional codes

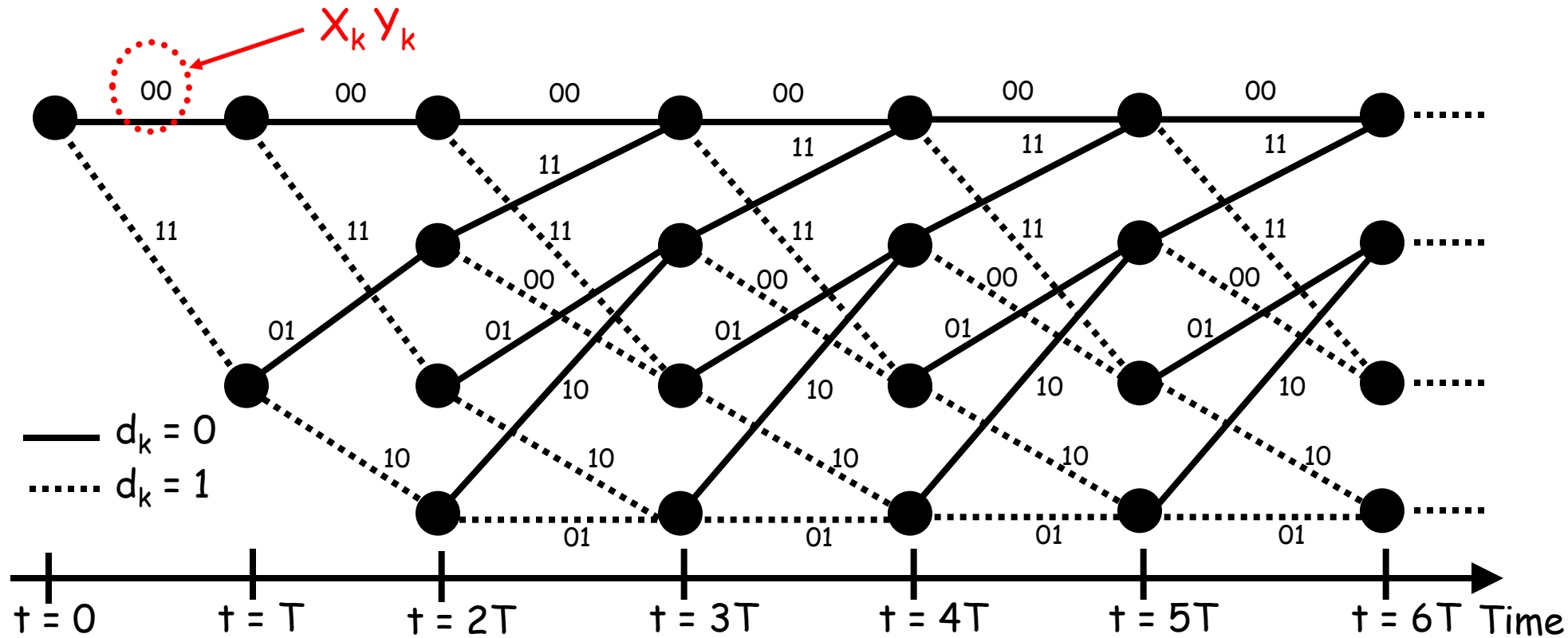
- The values of  $d_{k-2}$  and  $d_{k-1}$  define the current state of the encoder.
- The values of  $d_{k-1}$  and  $d_k$  define the next state of the encoder.

→ There are 4 states for such encoder.

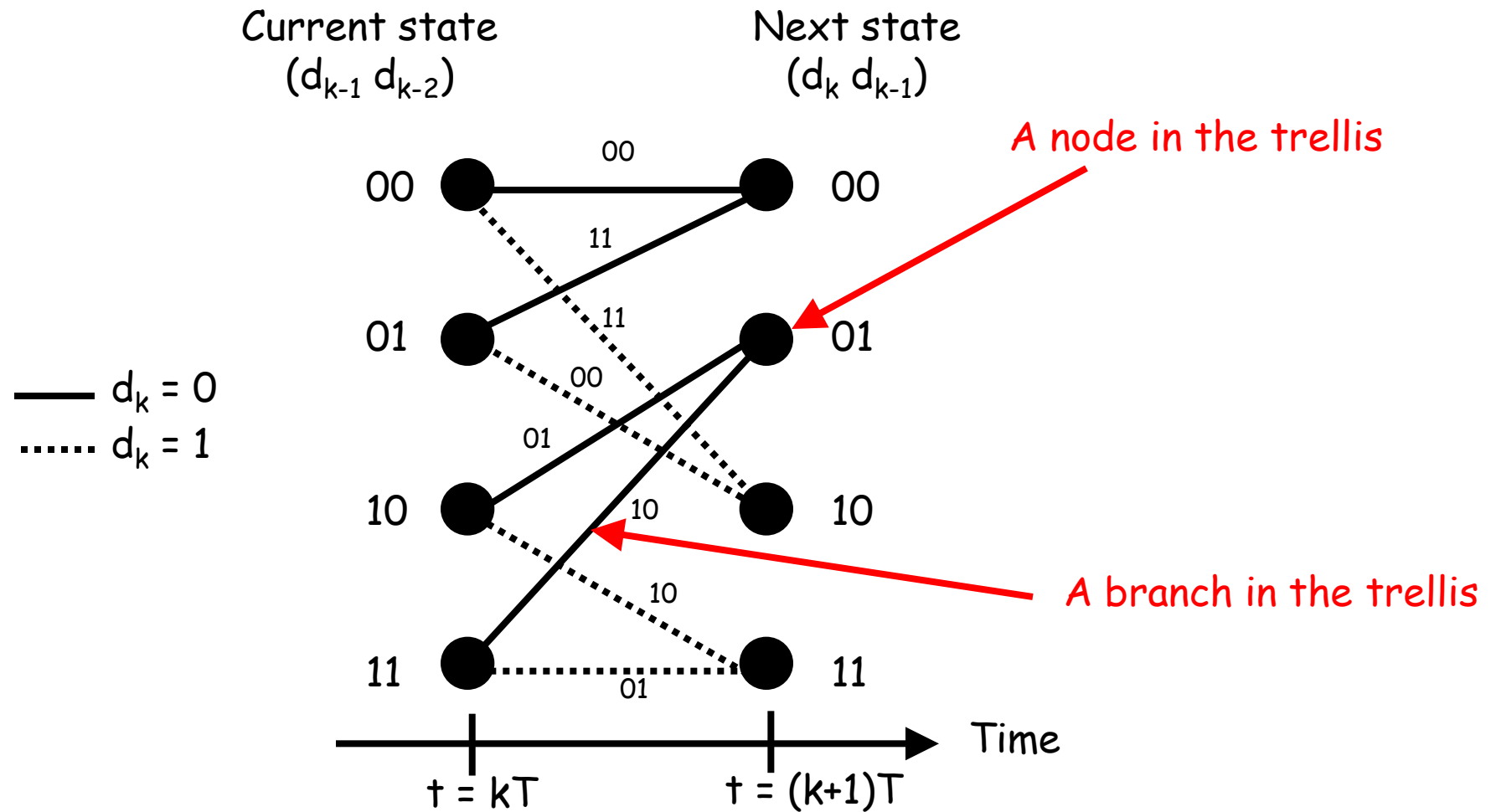
A code with a constraint length  $K$  has  $2^{K-1}$  states.

# Convolutional codes

The trellis diagram describes the operation of the encoder. For the previous encoder, the trellis is:



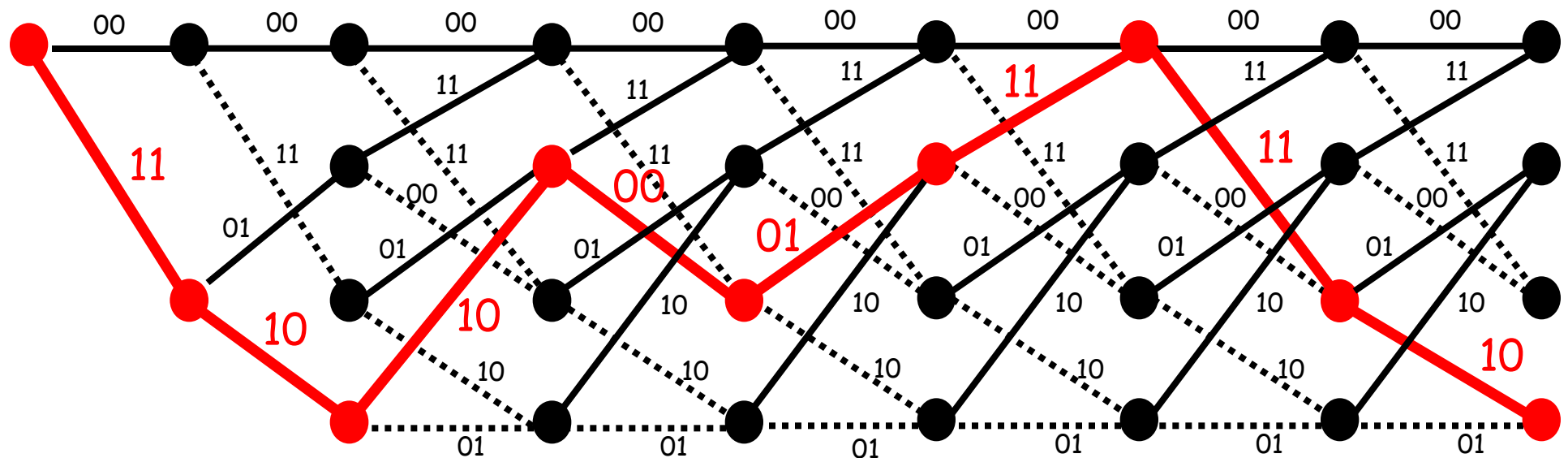
# Convolutional codes



# Convolutional codes

Consider the info sequence  $\{d_k\} = \{11010011\}$ .

The coded sequence  $\{X_k Y_k\}$  corresponds to a particular path in the trellis.



$$\rightarrow \{X_k Y_k\} = \{11 \ 10 \ 10 \ 00 \ 01 \ 11 \ 11 \ 10\}$$



# Convolutional codes

Each possible path in the trellis is a codeword.

Ex:  $M = (11010011) \Rightarrow C = (1110100001111110)$

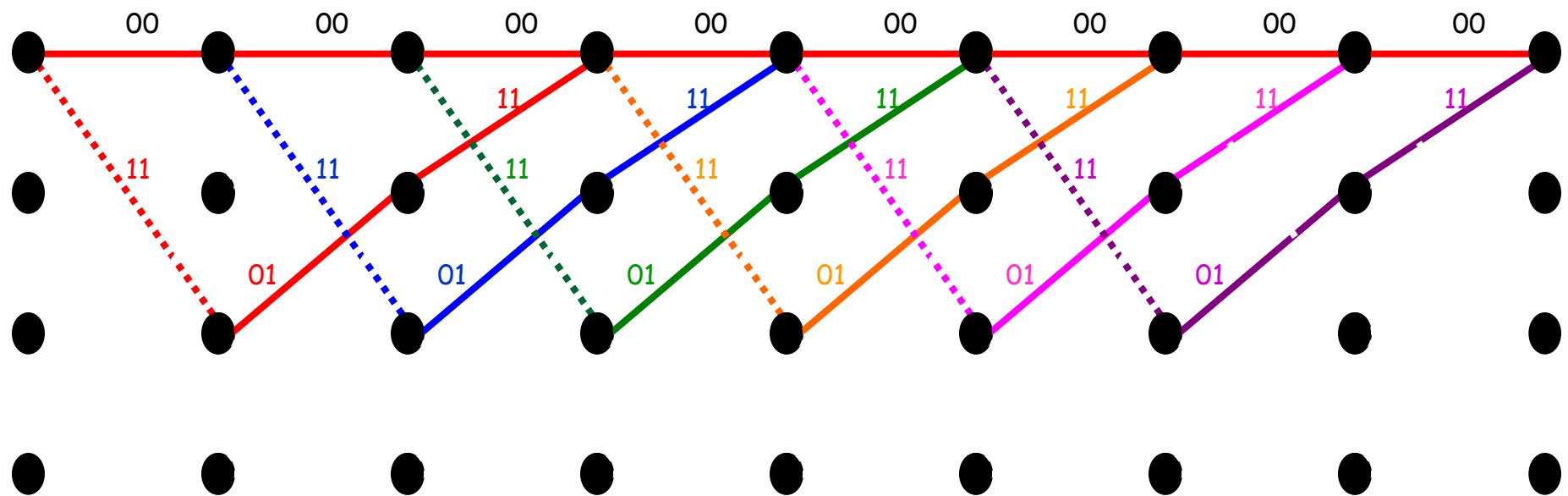
We can apply the results obtained for block codes to CCs.

What is the value of  $d_{\min}$  for the  $(5, 7)$  CC?

CCs are linear codes  $\Rightarrow$  Assume that the all-zero codeword was transmitted and search for the codeword with minimum Hamming weight.

# Convolutional codes

Here,  $k = 8$  and  $n = 16$ . We consider the weight-1 messages:

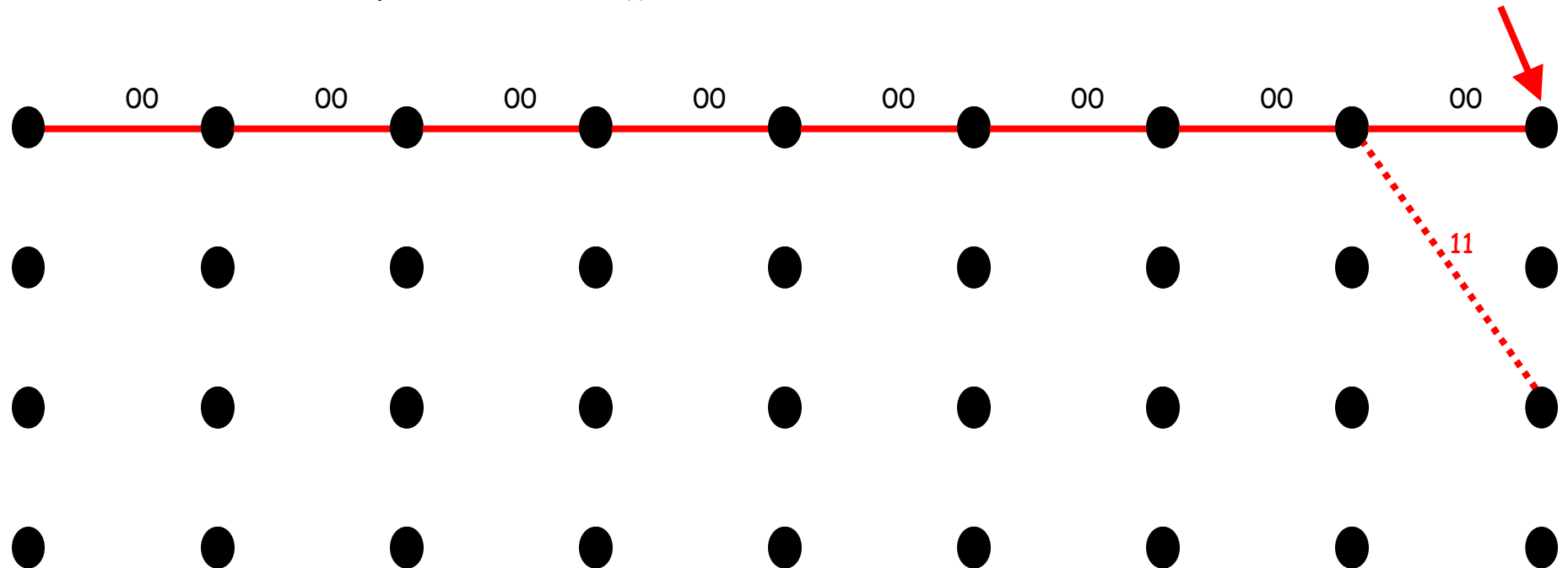


$d_{\min} = 5$  and number of codewords  $= k - 2 = 6$ . If  $k \gg 1$ , number of codewords  $\approx k$  ( $w_H = k - 2 \approx k$  also).

# Convolutional codes

Do we really have  $d_{\min} = 5$ ?

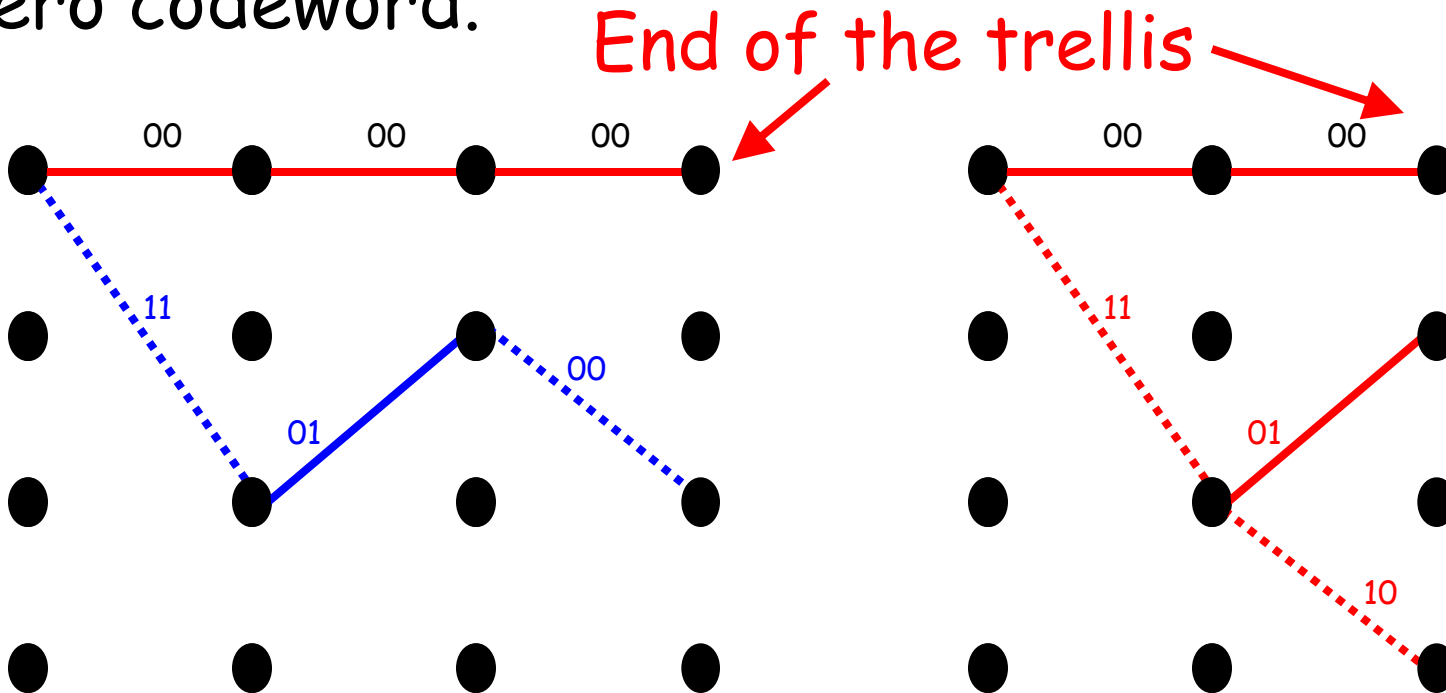
End of the trellis



$\Rightarrow d_{\min} = 2$  and number of codewords = 1 ( $w_H = 1$ ) ☹

# Convolutional codes

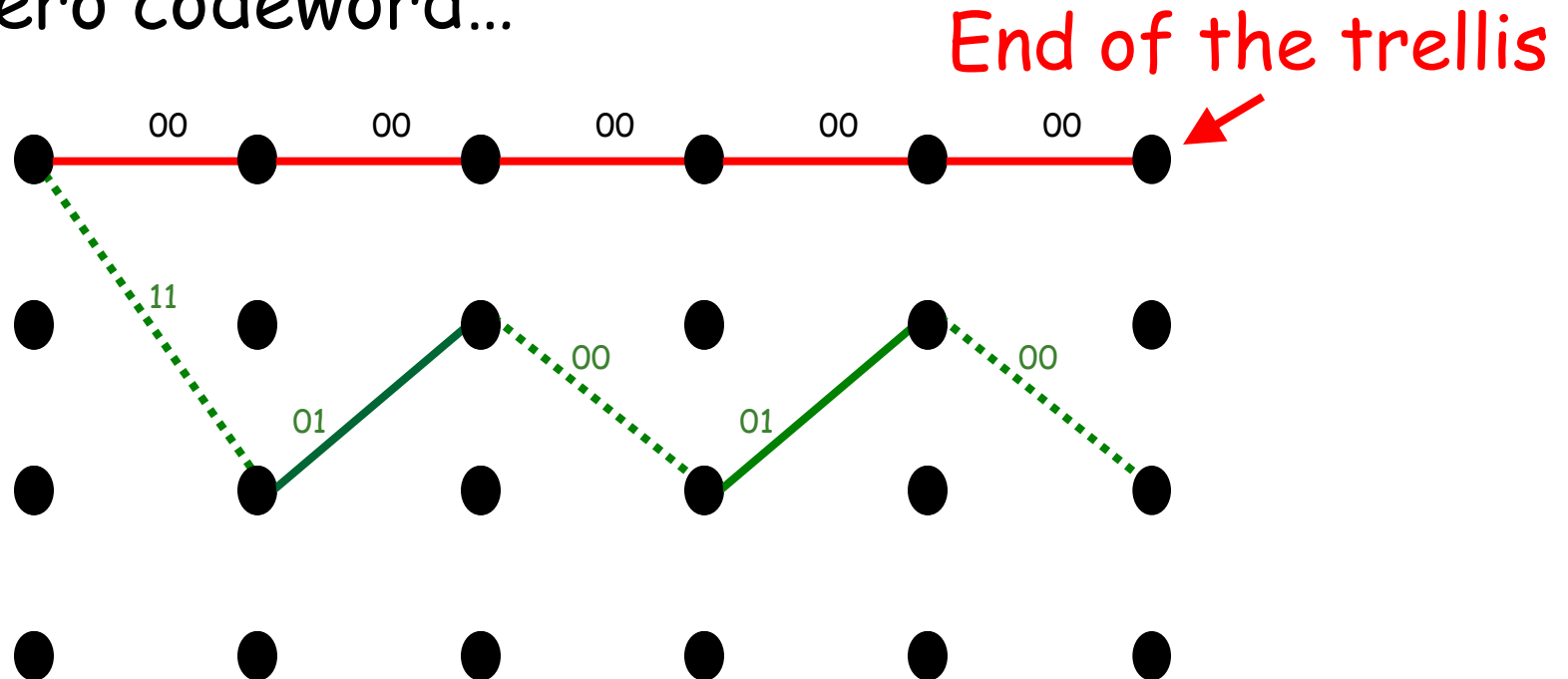
We also have codewords at distance  $d = 3$  from the all-zero codeword.



$d = 3$  and number of codewords = 3 ( $w_H = 5$ )

# Convolutional codes

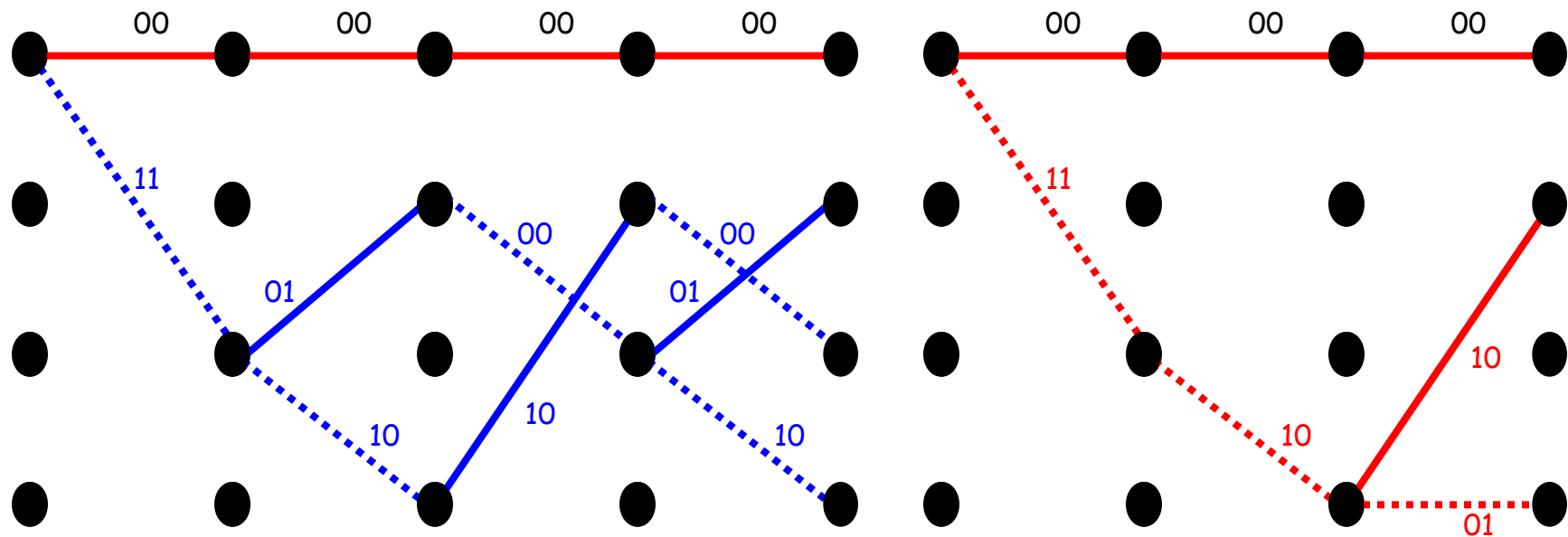
Plus some other codewords at distance  $d = 4$  from the all-zero codeword...



$d = 4$  and number of codewords = 6 ( $w_H = 16$ ).

# Convolutional codes

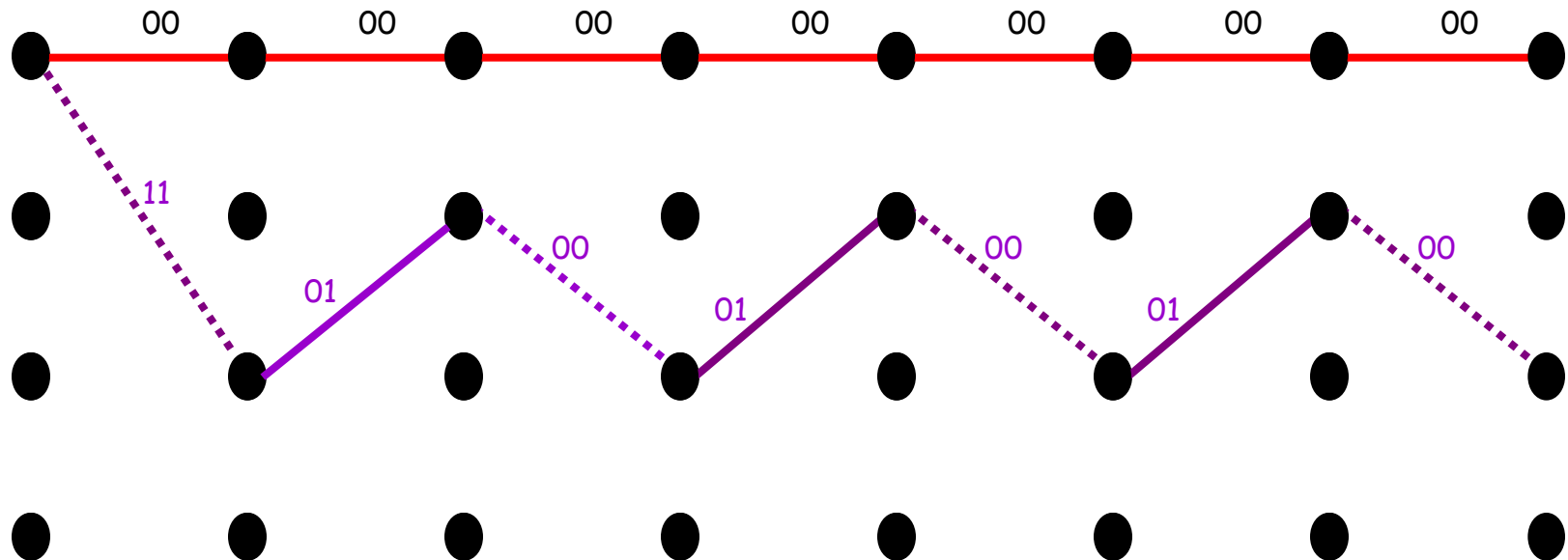
Plus some other codewords at distance  $d = 4$  from the all-zero codeword...



$d = 4$  and number of codewords = 6 ( $w_H = 16$ ).

# Convolutional codes

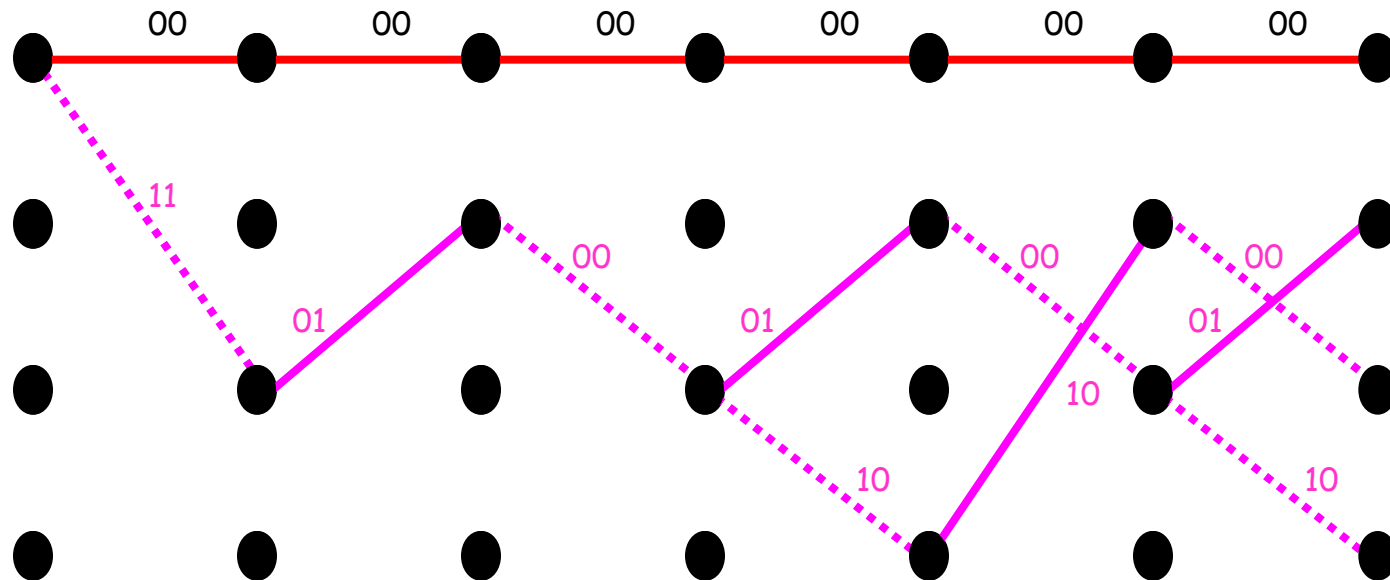
Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.



$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).

# Convolutional codes

Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.

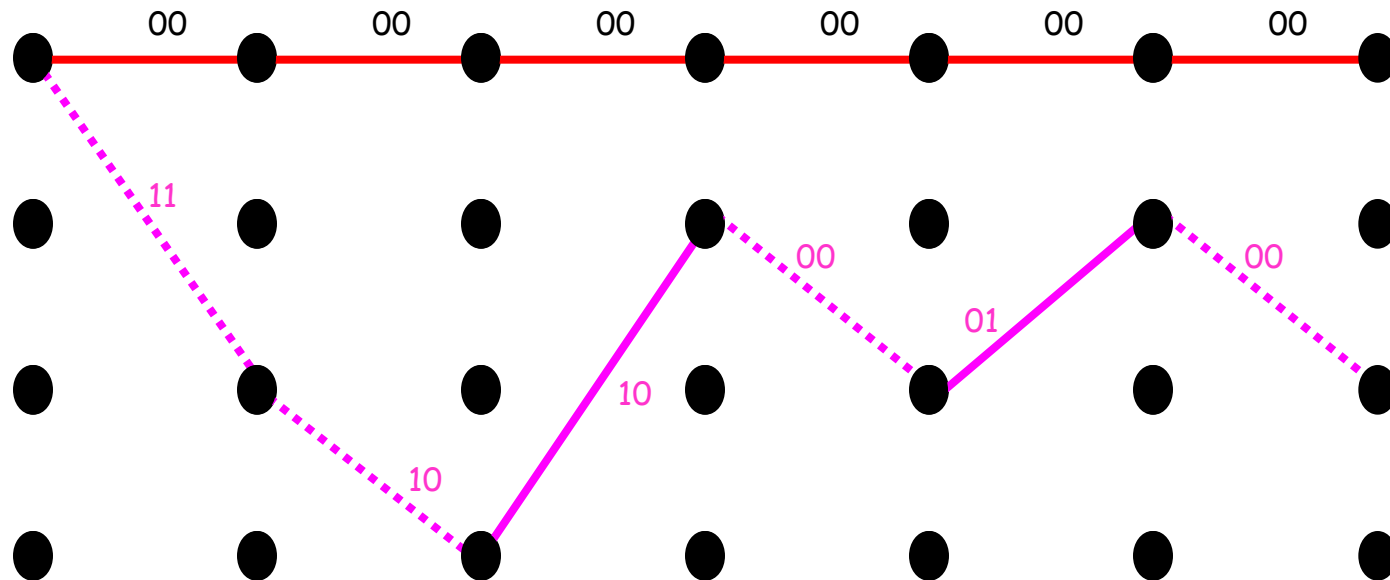


$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).



# Convolutional codes

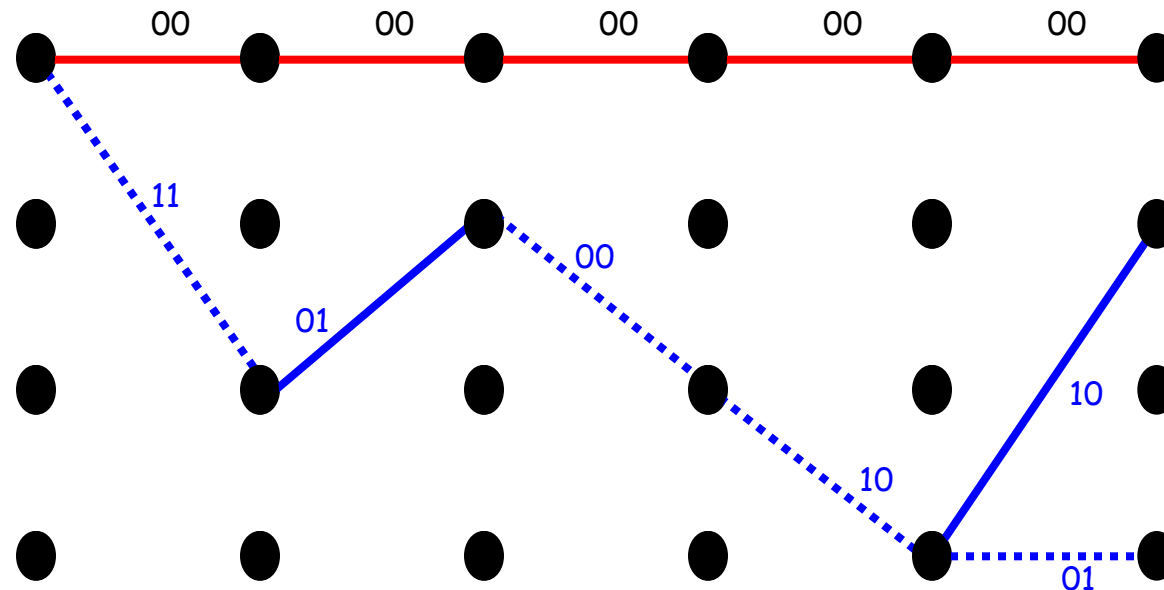
Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.



$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).

# Convolutional codes

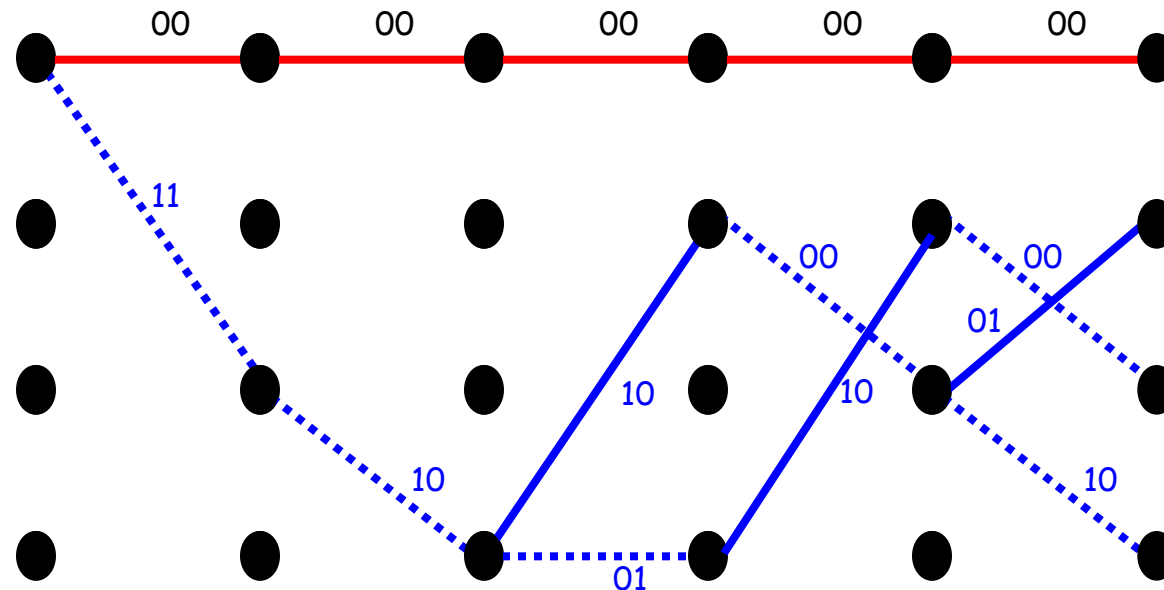
Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.



$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).

# Convolutional codes

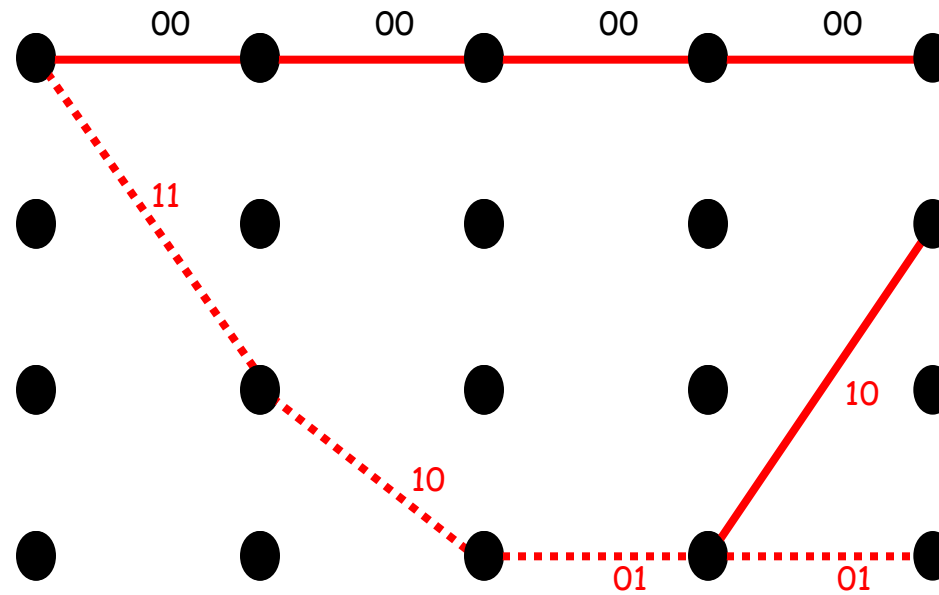
Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.



$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).

# Convolutional codes

Plus some more codewords at distance  $d = 5$  from the all-zero codeword, and so on.



$d = 5$  and number of codewords = 12 ( $w_H = 44$ ).

# Convolutional codes

Do you remember the union bound expression?

$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} \frac{w_d}{2k} \cdot \operatorname{erfc} \left( \sqrt{dR_c \frac{E_b}{N_0}} \right)$$

Error coefficient

$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} e(d) \cdot \operatorname{erfc} \left( \sqrt{dR_c \frac{E_b}{N_0}} \right) \quad \text{with } e(d) = \frac{w_d}{2k}$$

# Convolutional codes

Error coefficients for this code:  $e(d=2) = \frac{1}{2k} \ll 1$

$$e(d=3) = \frac{5}{2k} \ll 1 \quad e(d=4) = \frac{16}{2k} \ll 1 \quad e(d=5) = \frac{k-2+44}{2k} \approx \frac{1}{2}$$

In practice,  $k$  is very large ( $k \gg 1$ ).

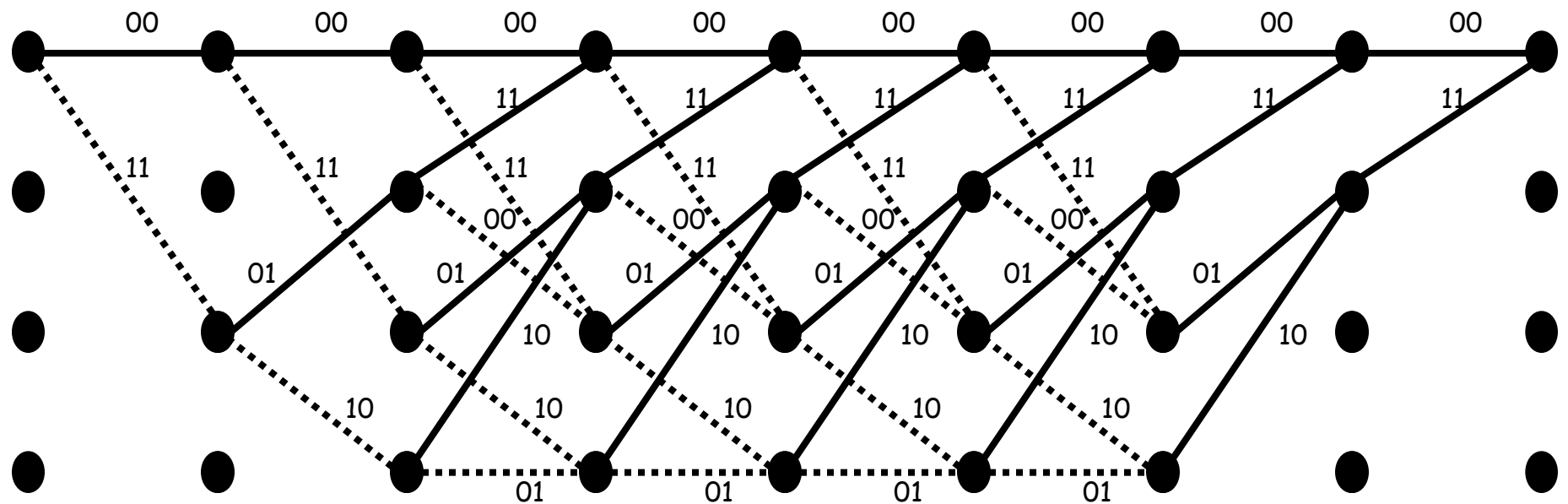
The first 3 terms in the union bound can possibly be ignored at low-to-medium SNR.

But, at high SNR, they certainly have a very detrimental effect on the error performance of the code (presence of an error floor).

# Convolutional codes

How to solve this problem? Make sure that the trellis returns to the zero state at the end.

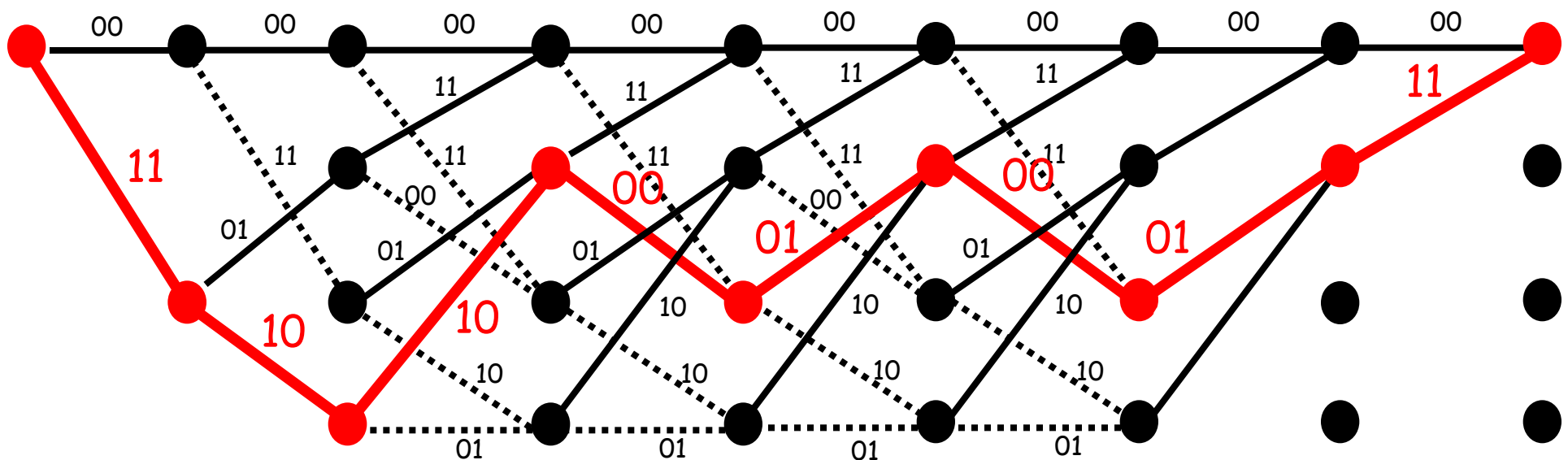
=> Append  $(K-1)$  0s ("tail bits") to the end of the message  $M$ . For  $k = 8$ , the trellis becomes:



# Convolutional codes

You could afford ignoring the end of the trellis if the message had an infinite length, but in practice messages seldom have an infinite length...

Consider the message  $M = (110101\underline{00})$ .





# Convolutional codes

CCs are block codes if the path in the trellis always starts and ends in the same state.

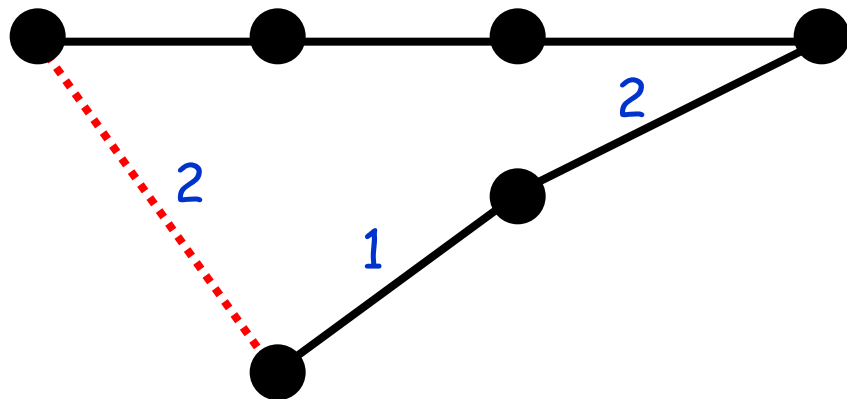
The minimum Hamming distance  $d_{\min}$  of a CC can thus be defined as the smallest Hamming distance between any pair of code sequences beginning and ending on the same state.

For a CC, the distance  $d_{\min}$  is often referred to as "free distance" ( $d_{\text{free}}$ ).

# Convolutional codes

Because CCs are linear codes, we can consider only sequences that begin and end on the zero state.

For our 4-state, rate-1/2, (5, 7) CC:



We find one path with  $d_{\text{free}} = 5$  and the Hamming weight of this path is  $N_{\text{dfree}} = 1$ .

All other paths are at least at a distance 6 from the all-zero codeword.

# Convolutional codes

If we extend this reasoning further, we can state that a CC with a terminated trellis is a block code for which:

- $d_{\min} = d_{\text{free}}$
- The parameter  $w_{d\min}$  is equal to  $\approx k$  times the total Hamming weight  $N_{d\text{free}}$  of the input sequences associated with the coded sequences that:

(1) leave the zero state at time  $t = 0$ ;

(2) are at distance  $d_{\text{free}}$  from the all-zero sequence.

# Convolutional codes

By generalizing this to any distance  $d > d_{\text{free}}$ , we can state that the parameter  $w_d$  is equal to  $\approx k$  times the total Hamming weight  $N_d$  of the input sequences associated with the coded sequences that:

- (1) leave the zero state at time  $t = 0$ ;
- (2) are at distance  $d$  from the all-zero sequence.

# Convolutional codes

The union bound expression for a CC can therefore be obtained easily:

$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} \frac{k \cdot N_d}{2k} \cdot \text{erfc} \left( \sqrt{dR_c \frac{E_b}{N_0}} \right)$$

$$P_{eb} \leq \sum_{d=d_{\text{free}}}^{+\infty} \frac{N_d}{2} \cdot \text{erfc} \left( \sqrt{dR_c \frac{E_b}{N_0}} \right)$$

# Convolutional codes

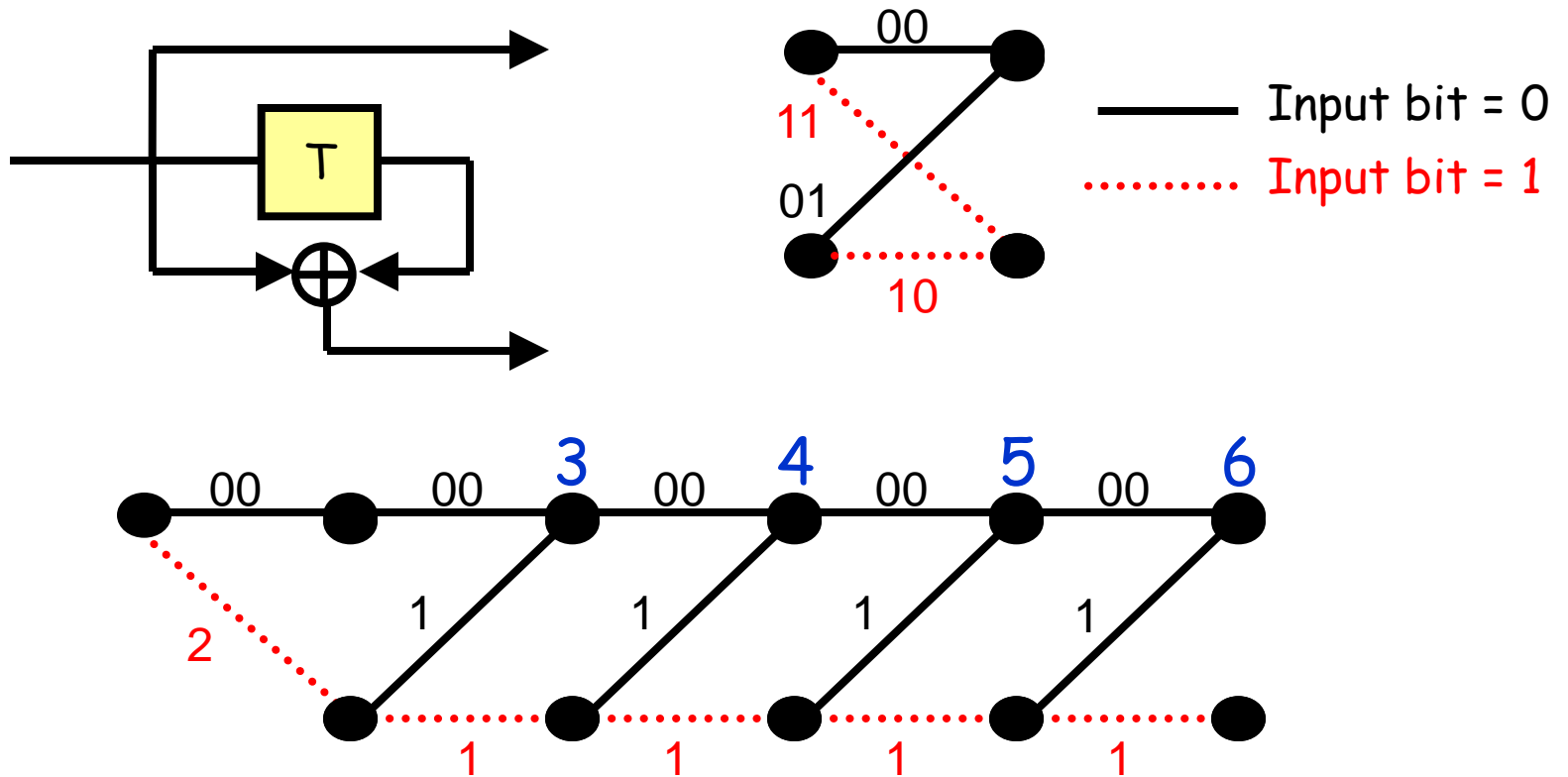
Remember that this union bound expression is only valid for ML decoding over BPSK, AWGN channel.

At high SNR, the dominant term in the sum is generally the term corresponding to the smallest value of  $d$  for the code, i.e. the free distance  $d_{\text{free}}$ :

$$P_{eb} \approx \frac{N_{d_{\text{free}}}}{2} \cdot \text{erfc} \left( \sqrt{d_{\text{free}} R_c \frac{E_b}{N_0}} \right)$$

# Convolutional codes

Example 1: 2-state, rate-1/2, (2, 3) code.



# Convolutional codes

2-state, rate-1/2, (2, 3) code:

$$P_{eb} \leq \frac{1}{2} \cdot \text{erfc}\left(\sqrt{\frac{3}{2} \frac{E_b}{N_0}}\right) + \frac{2}{2} \cdot \text{erfc}\left(\sqrt{\frac{4}{2} \frac{E_b}{N_0}}\right) + \frac{3}{2} \cdot \text{erfc}\left(\sqrt{\frac{5}{2} \frac{E_b}{N_0}}\right) + \dots$$

We find  $d_{\text{free}} = 3$  and  $N_{\text{dfree}} = 1$ .

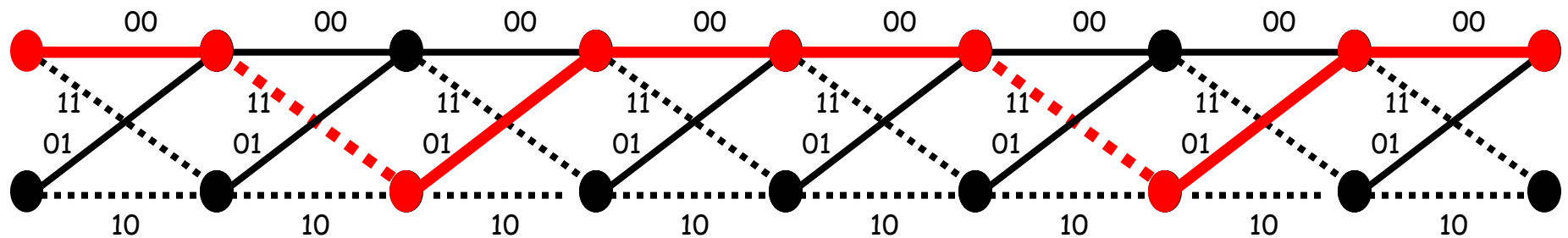
→ Asymptotic coding gain  $G \approx 1.76$  dB.



# Convolutional codes

We have to be careful not to forget the paths that leave the zero state and come back to it more than once.

Ex:  $M = (01000100) \Rightarrow d = 6$



These paths do not exist for distances  $d < 2d_{\text{free}}$ .

# Convolutional codes

An exhaustive search starting with the weight-1 messages might be the best way to evaluate the union bound expression for a CC.

To find the first few terms in the union bound, focusing on the messages with the smallest Hamming weights is generally sufficient.

As an example, let us determine the first 5 terms in the union bound expression of the  $(2, 3)$  CC.

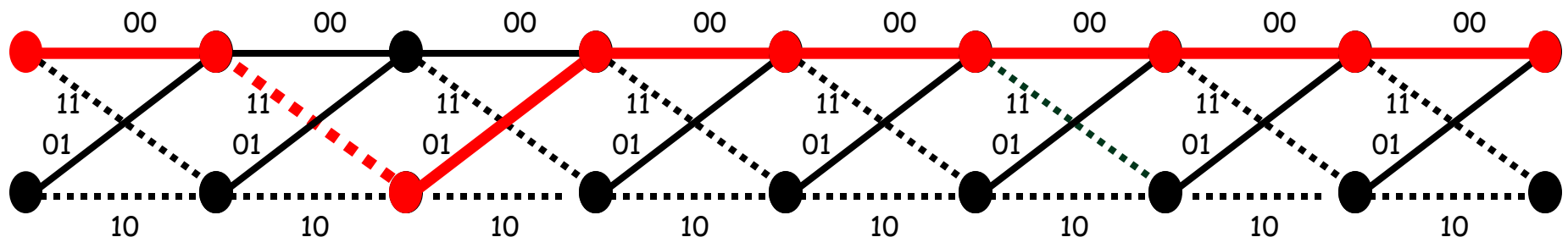
# Convolutional codes

Consider weight-1 messages first.

All weight-1 messages lead to codewords at distance  $d = 3$  from the all-zero codeword.

There are  $(k-1)$  possible weight-1 messages.

Ex:  $M = (01000000) \Rightarrow d = 3$



# Convolutional codes

The contribution of these weight-1 messages to the union bound is the following error coefficient value:

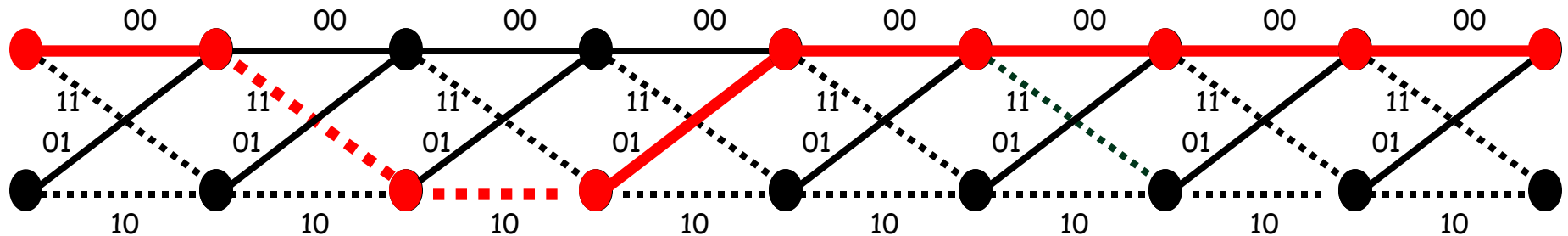
$$e(d=3) = \frac{1 \cdot (k-1)}{2k} \approx \frac{1}{2}$$

Consider now weight-2 messages. We have to consider two possible configurations.

1.  $M = (...0110...)$  in which the 1s are together. These configurations lead to codewords at  $d = 4$ .

# Convolutional codes

Ex:  $M = (01100000) \Rightarrow d = 4$



The number of these configurations is  $(k-2)$  and their contribution to the union bound is therefore the following error coefficient value:

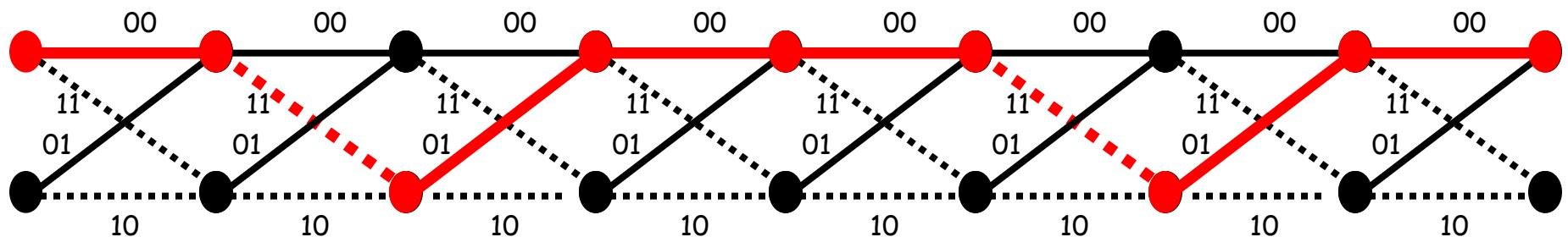
$$e(d = 4) = \frac{2 \cdot (k - 2)}{2k} \approx 1$$

# Convolutional codes

2.  $M = (...010...010...)$  in which the 1s are separated by at least one 0.

These configurations lead to codewords at  $d = 6$ .

Ex:  $M = (01000100)$  generates a codeword at  $d = 6$ .



# Convolutional codes

The number of messages in the form (...010...010...) is given by

$$\binom{k-1}{2} - (k-2) = \frac{(k-1) \cdot (k-2)}{2} - (k-2) \approx \frac{k^2}{2}$$

The contribution of these weight-2 configurations to the union bound is therefore the following error coefficient value:

$$e(d=6) \approx \frac{2k^2}{4k} = \frac{k}{2}$$

# Convolutional codes

Now, consider the weight-3 messages. We have three possible configurations:

1.  $M = (...01110...)$  in which the three 1s are together. These configurations lead to codewords at  $d = 5$ . The number of these configurations is  $(k-3)$ .

Their contribution to the union bound is the following error coefficient:

$$e(d = 5) = \frac{3 \cdot (k - 3)}{2k} \approx \frac{3}{2}$$



# Convolutional codes

2.  $M = (...0110...010...)$ , i.e. two 1s together and separated from the third 1 by at least one 0.

These configurations lead to codewords at  $d = 7$ . Their number is given by

$$2 \cdot \left( \binom{k-2}{2} - (k-3) \right) = \frac{2 \cdot (k-2) \cdot (k-3)}{2} - 2 \cdot (k-3) \approx k^2$$

and their contribution to the union bound expression is thus represented by the error coefficient:

$$e(d=7) \approx \frac{3k^2}{2k} = \frac{3k}{2}$$

# Convolutional codes

3.  $M = (...010...010...010...)$ , i.e. three 1s separated from the each others by at least one 0.

These configurations lead to codewords at  $d = 9$  and, as a result, do not have to be considered since the first 5 terms in the union bound corresponds to the distances  $d = 3, 4, 5, 6$ , and 7 only.

Now, consider the weight-4 messages. We have several possible configurations:

# Convolutional codes

1.  $M = (...011110...)$  in which the four 1s are together. These configurations lead to codewords at  $d = 6$ .

The number of these configurations is  $(k-4)$ , and their contribution to the union bound is therefore the following error coefficient value:

$$e(d = 6) = \frac{4 \cdot (k - 4)}{2k} \approx 2$$

All other weight-4 configurations do not have to be considered because they lead to codewords at a distance  $d > 7$  from the all-zero codeword.

# Convolutional codes

Finally, we also need to consider the weight-5 messages in the form  $M = (...011110...)$  as they lead to codewords at a distance  $d = 7$  from the all-zero codeword.

Their contribution to the union bound expression is given by the error coefficient:

$$e(d = 7) = \frac{5 \cdot (k - 5)}{2k} \approx \frac{5}{2}$$

# Convolutional codes

All other weight-5 configurations do not have to be considered because they lead to codewords at a distance  $d > 7$  from the all-zero codeword.

As for messages with weights  $> 5$ , they can also be ignored for the same reason.

Finally, by putting all these results together, we obtain the error coefficients for the first 5 terms in the union bound:

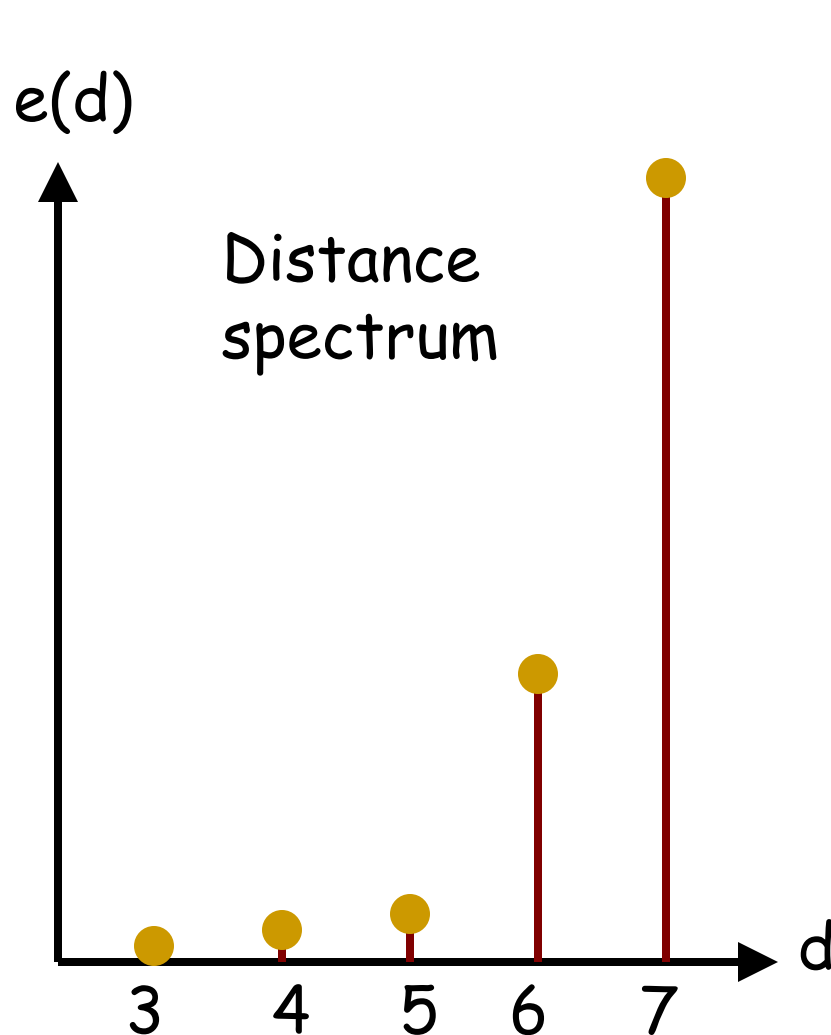
# Convolutional codes

$$P_{eb} \leq \frac{1}{2} \cdot \text{erfc}\left(\sqrt{\frac{3}{2} \frac{E_b}{N_0}}\right) + 1 \cdot \text{erfc}\left(\sqrt{\frac{4}{2} \frac{E_b}{N_0}}\right) + \frac{3}{2} \cdot \text{erfc}\left(\sqrt{\frac{5}{2} \frac{E_b}{N_0}}\right) + \dots$$
$$\dots + \left(\frac{k}{2} + 2\right) \cdot \text{erfc}\left(\sqrt{\frac{6}{2} \frac{E_b}{N_0}}\right) + \left(\frac{3k}{2} + \frac{5}{2}\right) \cdot \text{erfc}\left(\sqrt{\frac{7}{2} \frac{E_b}{N_0}}\right) + \dots$$

The first 3 terms in the union bound are identical to those previously determined using a trellis search.

The 4<sup>th</sup> and 5<sup>th</sup> terms are essentially generated by weight-2 and -3 messages, respectively.

# Convolutional codes



$$e(d=3) \approx \frac{1}{2}$$

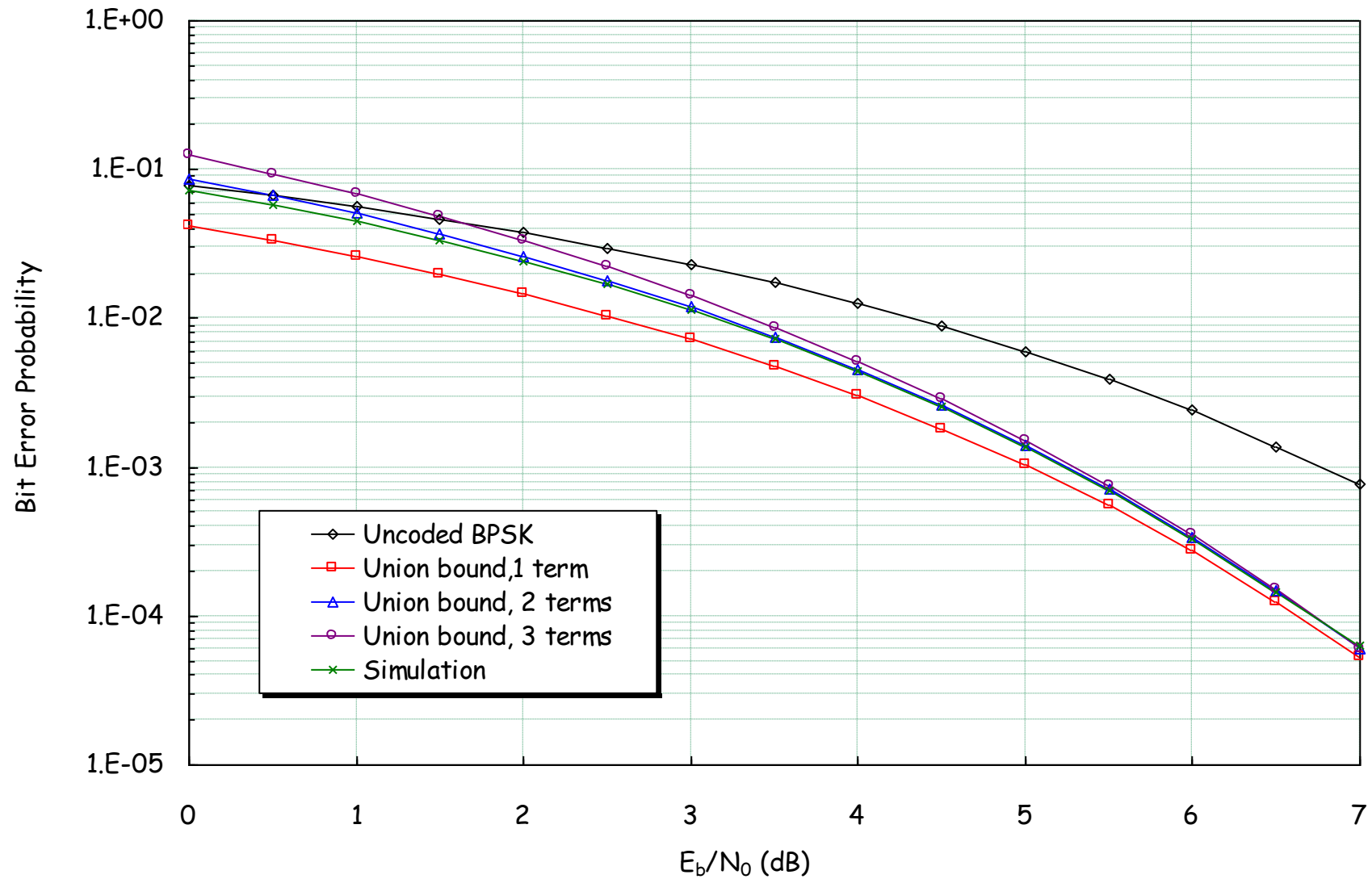
$$e(d=4) \approx 1$$

$$e(d=5) \approx \frac{3}{2}$$

$$e(d=6) \approx \frac{k}{2} + 2 \approx \frac{k}{2}$$

$$e(d=7) \approx \frac{3k}{2} + \frac{5}{2} \approx \frac{3k}{2}$$

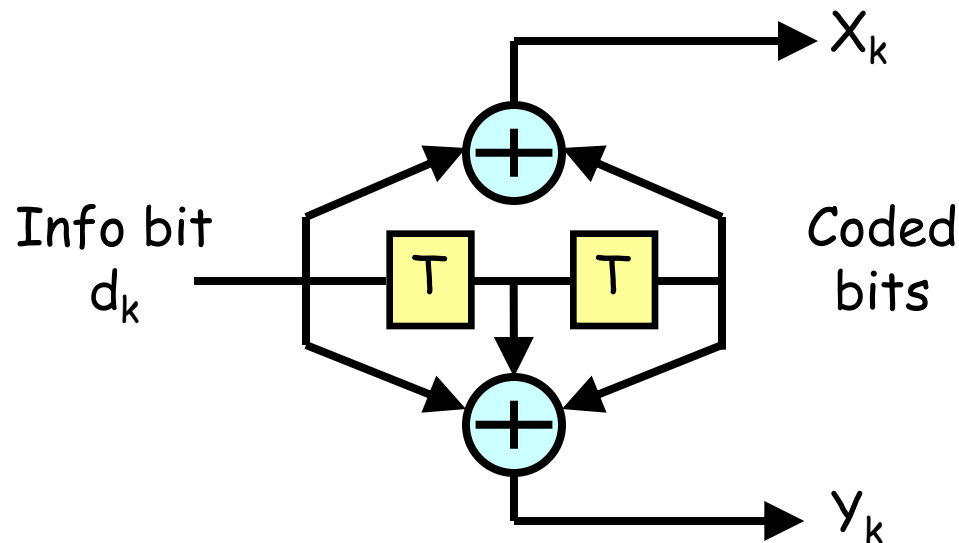
CC:  $R_c = 1/2$ ,  $K = 2, (1, 3)$ , soft-decision Viterbi decoding



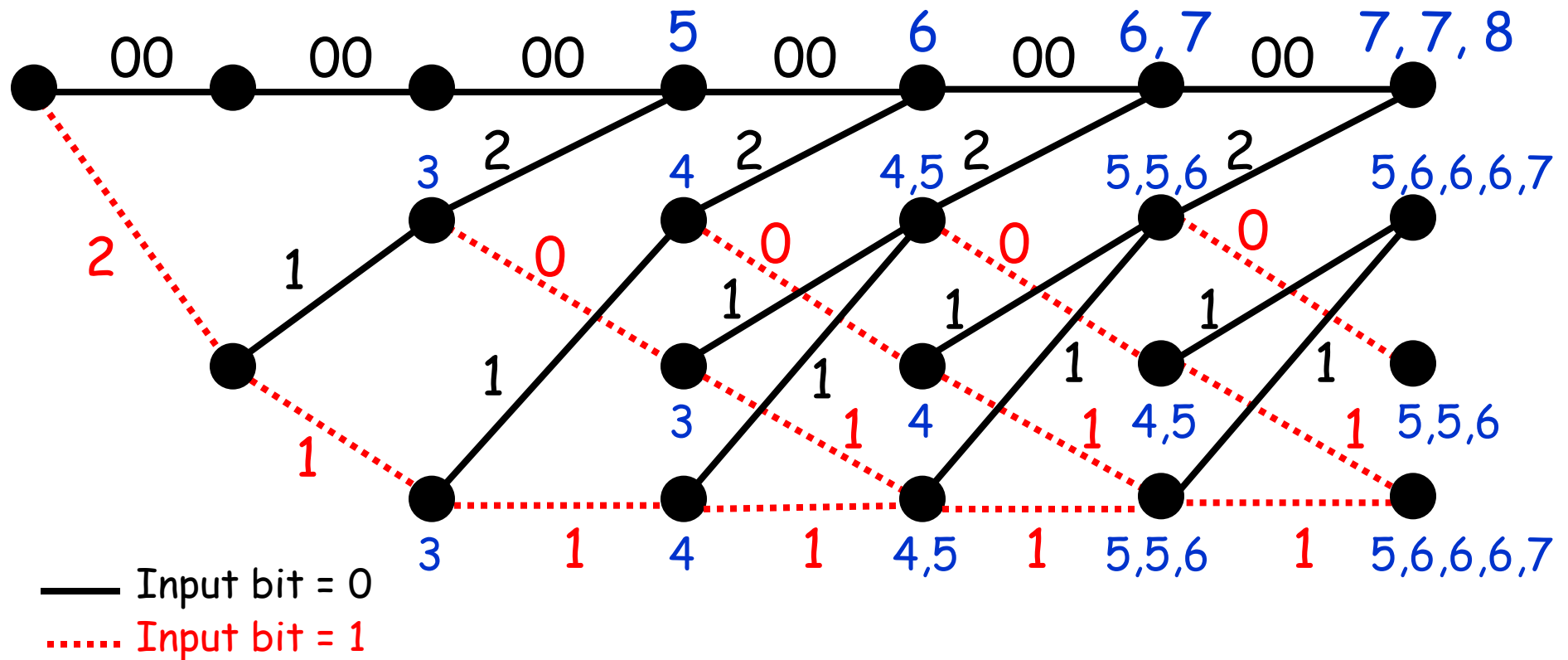


# Convolutional codes

Example 2: 4-state, rate-1/2, (5, 7) code



# Convolutional codes



# Convolutional codes

4-state, rate-1/2, (5, 7) code:

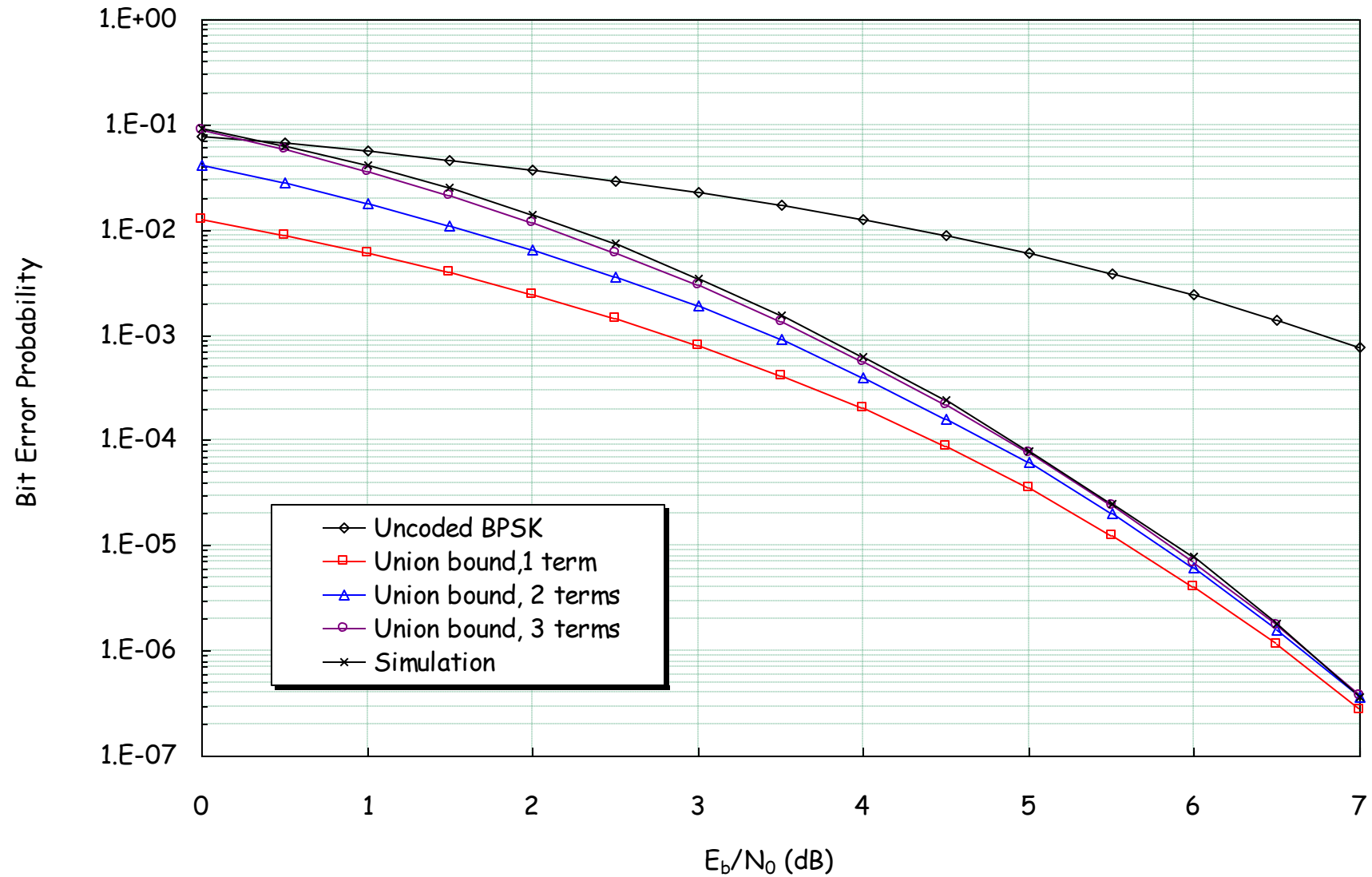
$$P_{eb} \leq \frac{1}{2} \cdot \text{erfc} \left( \sqrt{\frac{5}{2} \frac{E_b}{N_0}} \right) + \frac{4}{2} \cdot \text{erfc} \left( \sqrt{\frac{6}{2} \frac{E_b}{N_0}} \right) + \dots$$

We find  $d_{\text{free}} = 5$  and  $N_{\text{dfree}} = 1$ .

→ Asymptotic coding gain  $G \approx 3.98$  dB

Very good, but how to implement ML decoding for CCs?

CC:  $R_c = 1/2$ ,  $K = 3, (5, 7)$ , soft-decision Viterbi decoding



# Convolutional codes

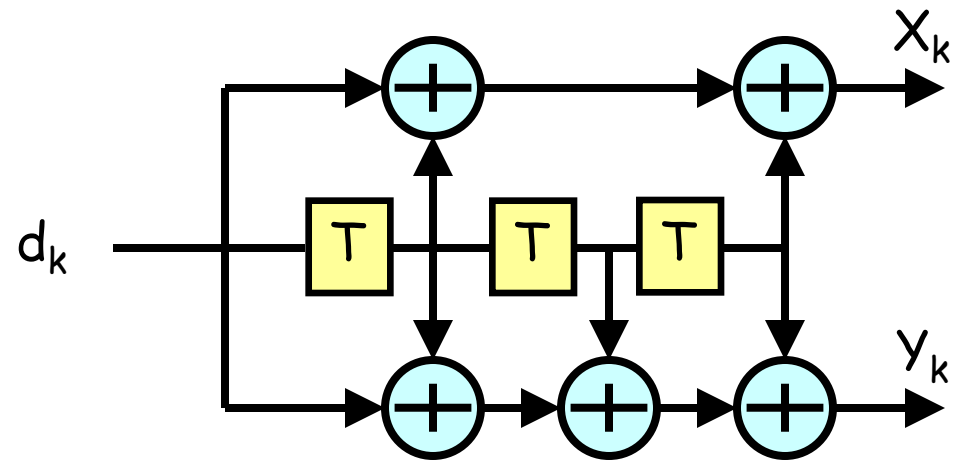
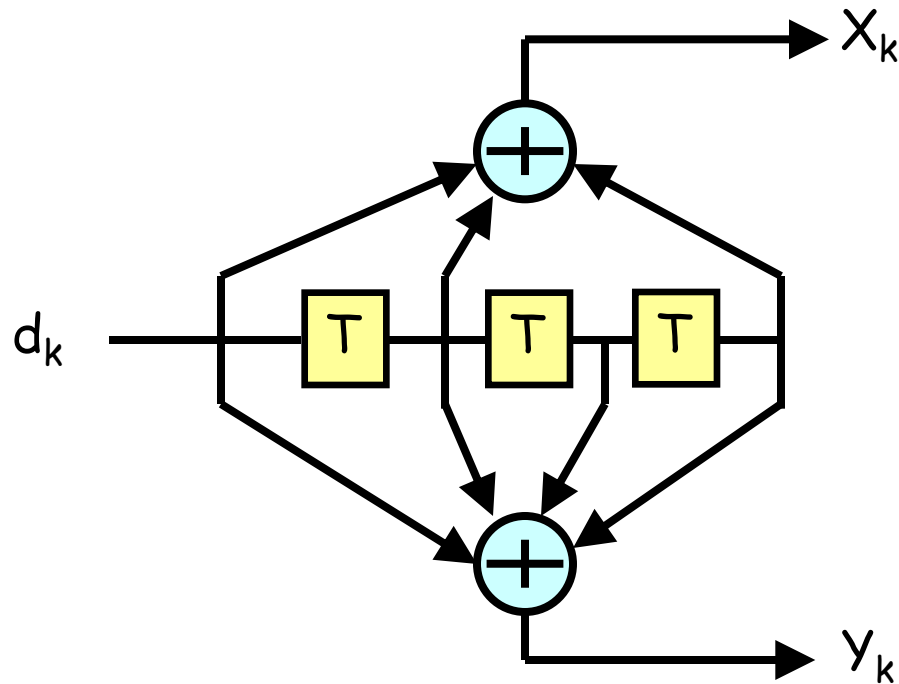
Some “popular” rate-1/2 CCs:

- $(5, 7)$ ,  $K = 3 \rightarrow d_{\text{free}} = 5$  ( $G = 3.98$  dB)  
 $N_{\text{dfree}} = 1$ ,  $N_{\text{dfree}+1} = 4$ ,  $N_{\text{dfree}+2} = 12$ .

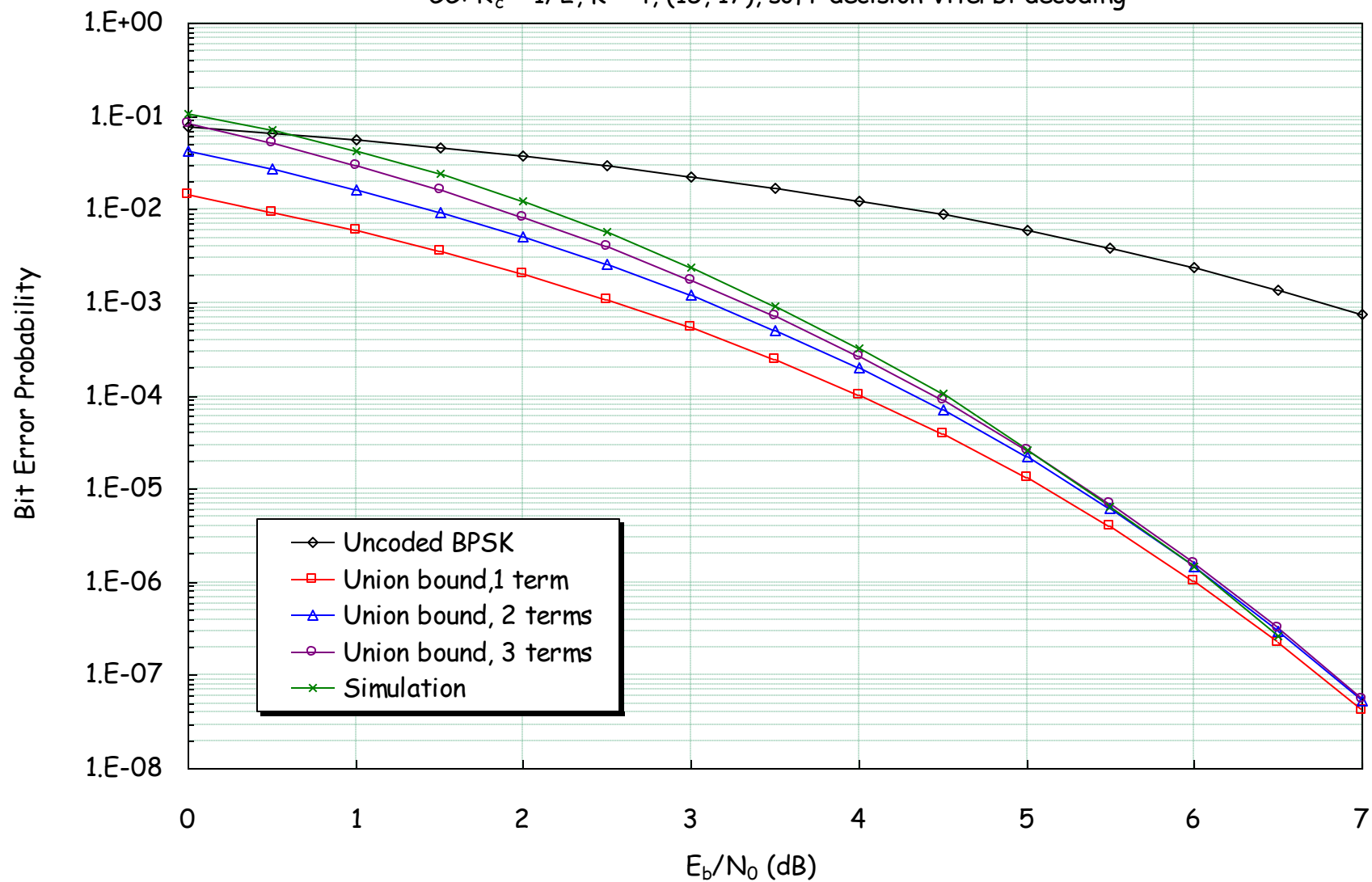
- $(15, 17)$ ,  $K = 4 \rightarrow d_{\text{free}} = 6$  ( $G = 4.77$  dB)  
 $N_{\text{dfree}} = 2$ ,  $N_{\text{dfree}+1} = 7$ ,  $N_{\text{dfree}+2} = 18$ .

# Convolutional codes

8-state, rate-1/2, (15, 17) code:



CC:  $R_c = 1/2$ ,  $K = 4$ , (15, 17), soft-decision Viterbi decoding



# Convolutional codes

Some “popular” rate-1/2 CCs:

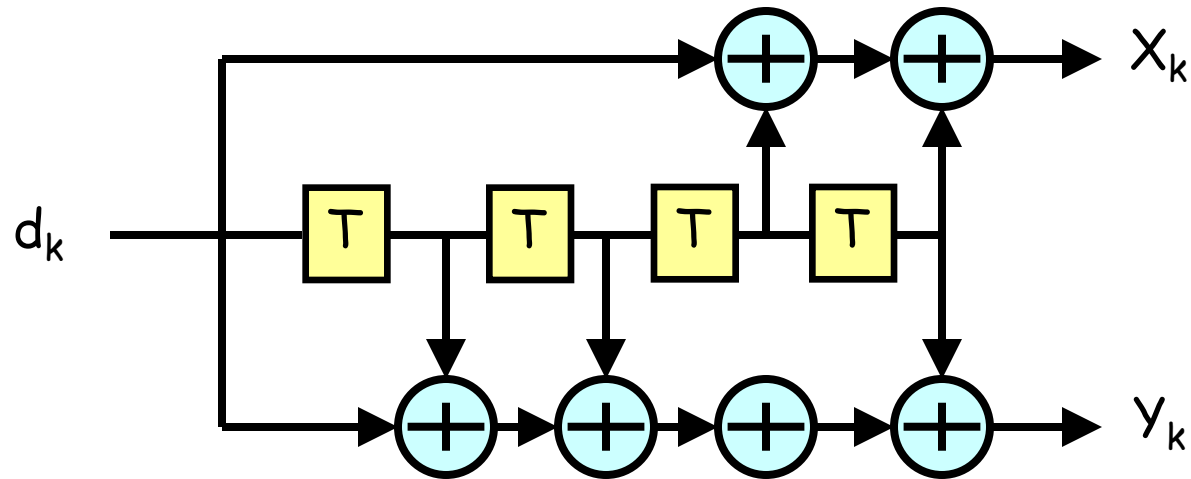
- $(23, 35)$ ,  $K = 5 \rightarrow d_{\text{free}} = 7$  ( $G = 5.44$  dB)  
 $N_{\text{dfree}} = 4$ ,  $N_{\text{dfree}+1} = 12$ ,  $N_{\text{dfree}+2} = 20$ .

- $(133, 171)$ ,  $K = 7 \rightarrow d_{\text{free}} = 10$  ( $G = 6.99$  dB)  
 $N_{\text{dfree}} = 36$ ,  $N_{\text{dfree}+1} = 0$ ,  $N_{\text{dfree}+2} = 211$ .



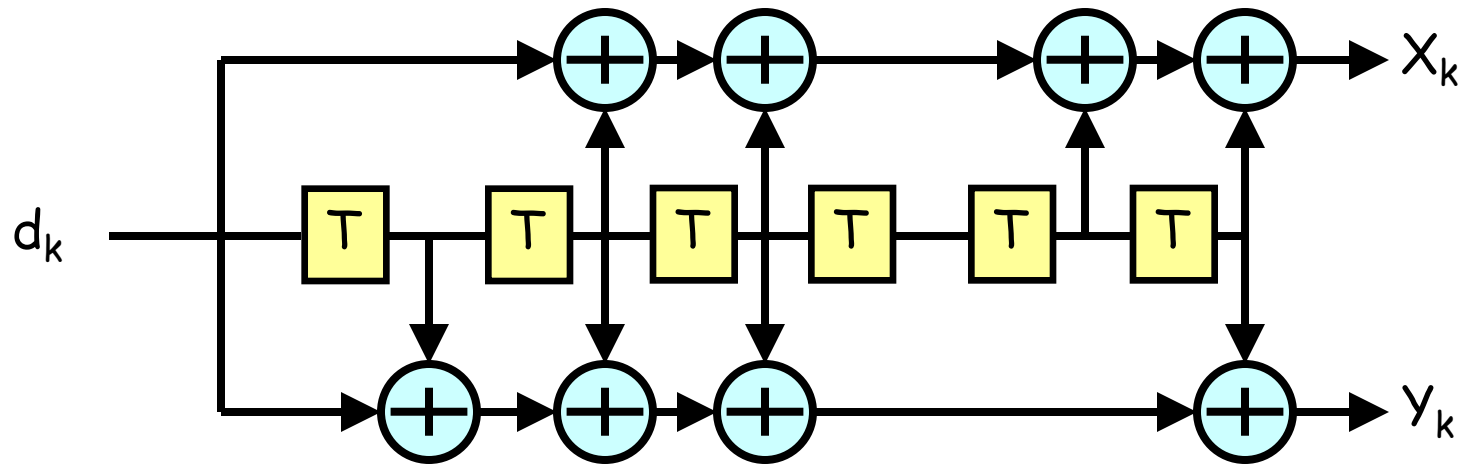
# Convolutional codes

16-state, rate-1/2, (23, 35) code:

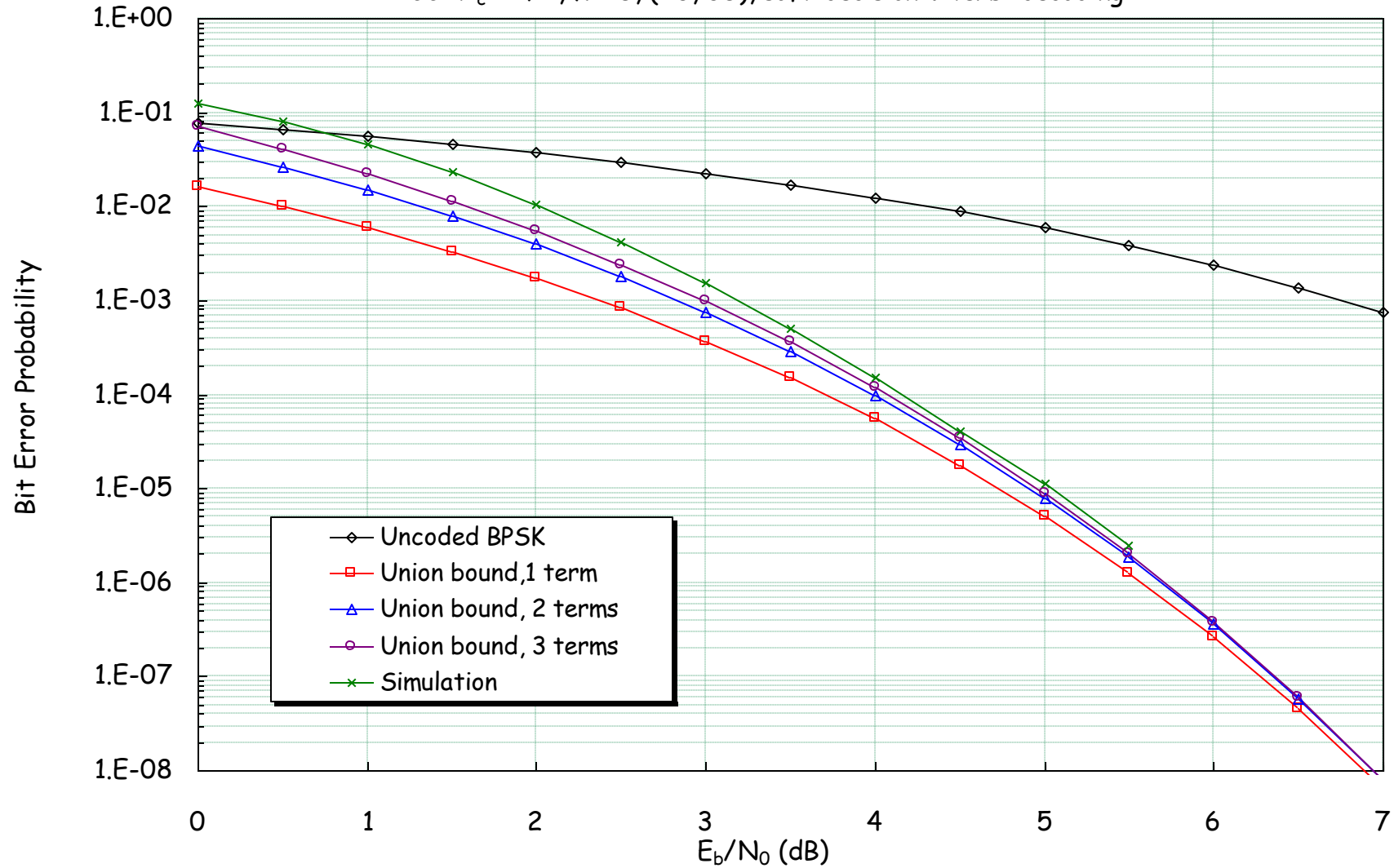


# Convolutional codes

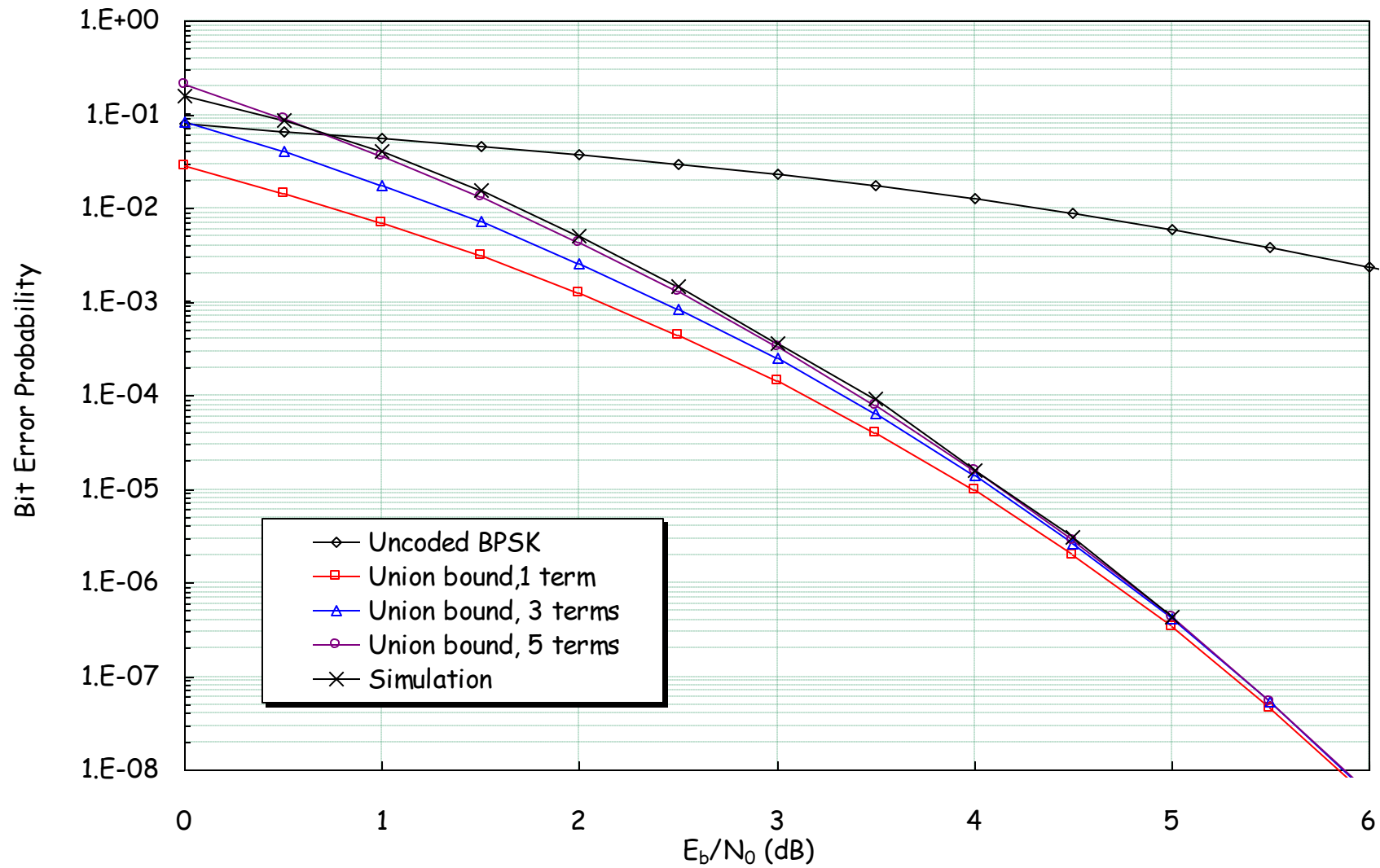
64-state, rate-1/2, (133, 171) code:



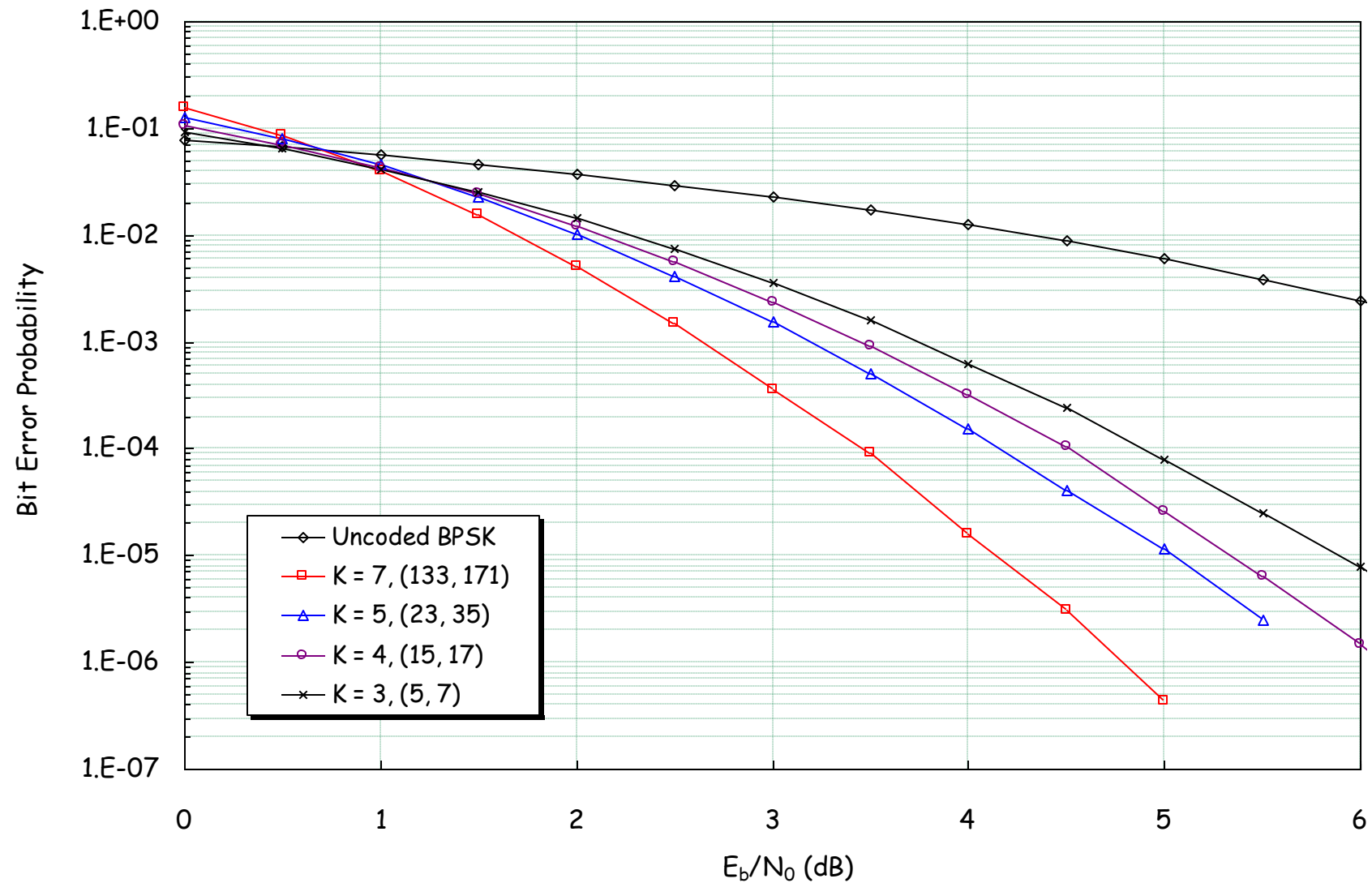
CC:  $R_c = 1/2$ ,  $K = 5$ , (23, 35), soft-decision Viterbi decoding



CC:  $R_c = 1/2$ ,  $K = 7$ , (133, 171), soft-decision Viterbi decoding



BER performance of several rate-1/2 convolutional codes (soft-decision Viterbi decoding)

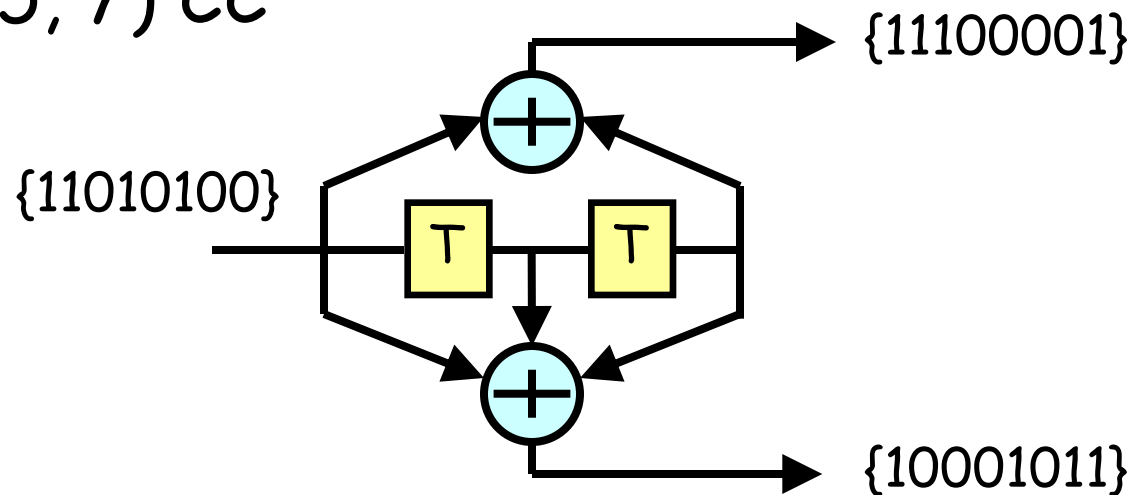


# Punctured convolutional codes

What about other coding rates?

The most popular technique to generate CCs with coding rates  $R_c > \frac{1}{2}$  consists of periodically deleting coded bits at the output of a rate-1/2 encoder.

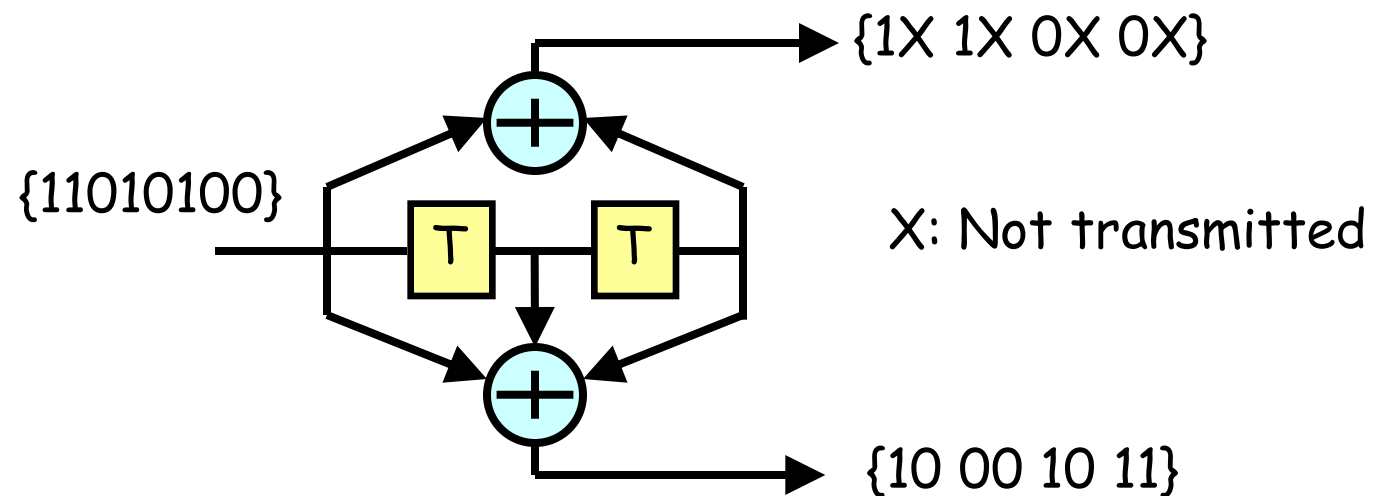
Ex: Rate-1/2, (5, 7) CC



# Punctured convolutional codes

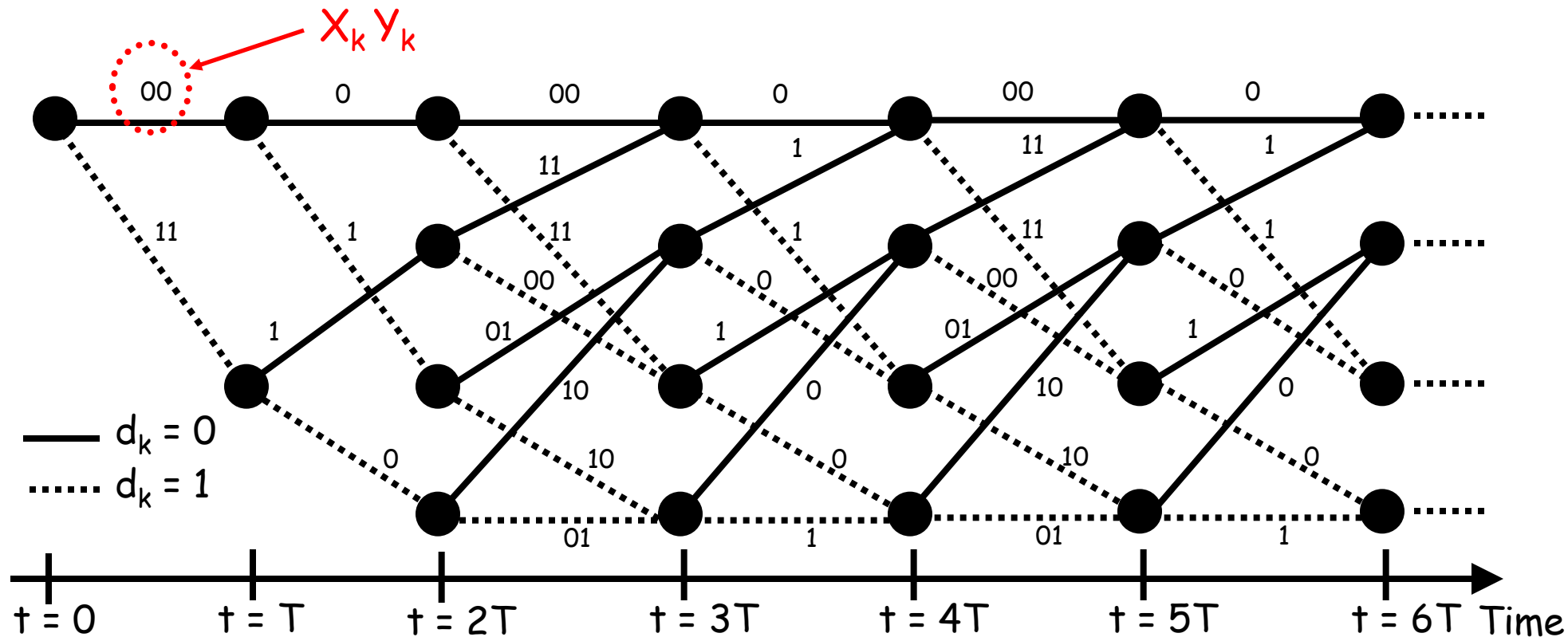
A 4-state, rate-2/3 CC can be generated from such code using, for instance, the puncturing pattern

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



# Punctured convolutional codes

Trellis of the rate-2/3 code thus obtained:

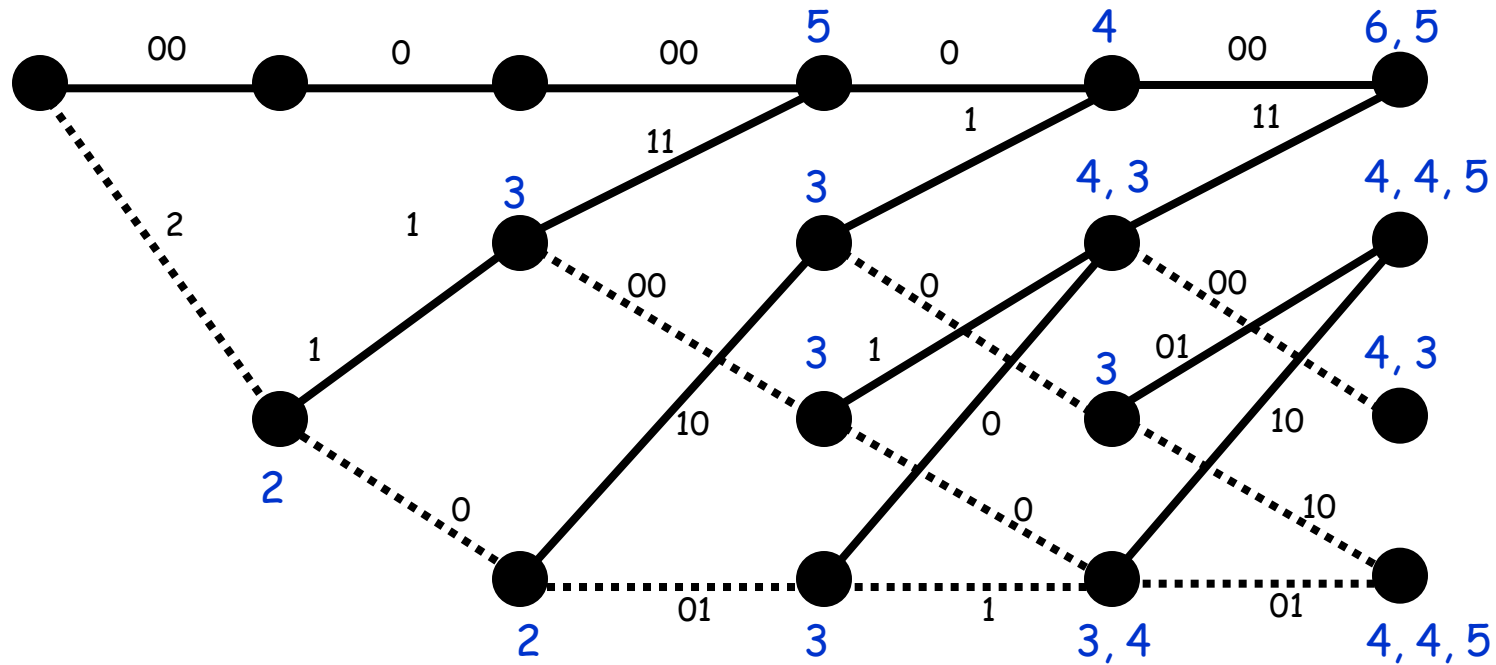




# Punctured convolutional codes

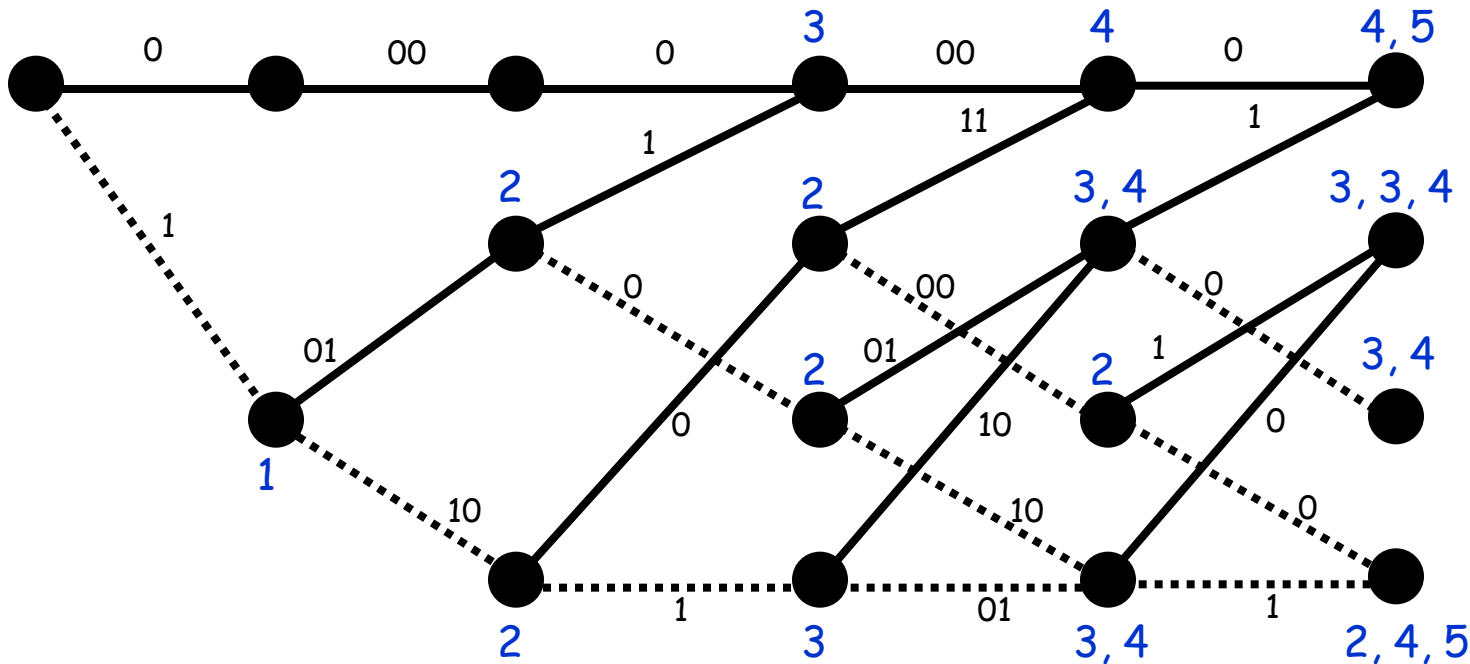
Two cases to consider in order to evaluate  $d_{\text{free}}$ :

(1) The erroneous codeword leaves the all-zero codeword during the "non-punctured" time-slot.



# Punctured convolutional codes

(2) The erroneous codeword leaves the all-zero codeword during the "punctured" time-slot.



Take the worst-case scenario:  $d_{\text{free}} = 3$ .  
Puncturing a code decreases its free distance.

# Punctured convolutional codes

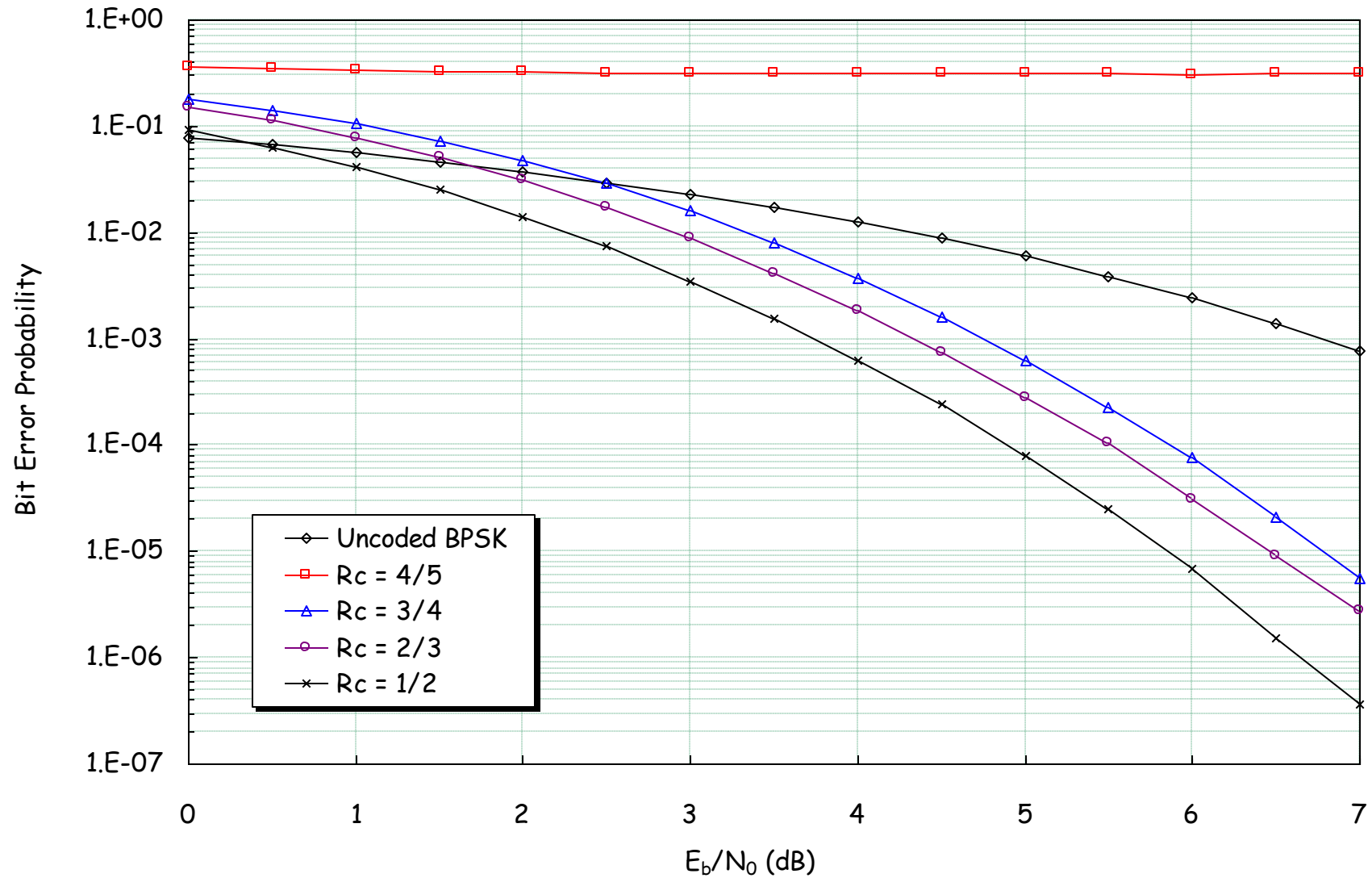
A few examples of punctured codes:

Rate-2/3 CC obtained using  $P = \begin{bmatrix} 1110 \\ 1011 \end{bmatrix}$

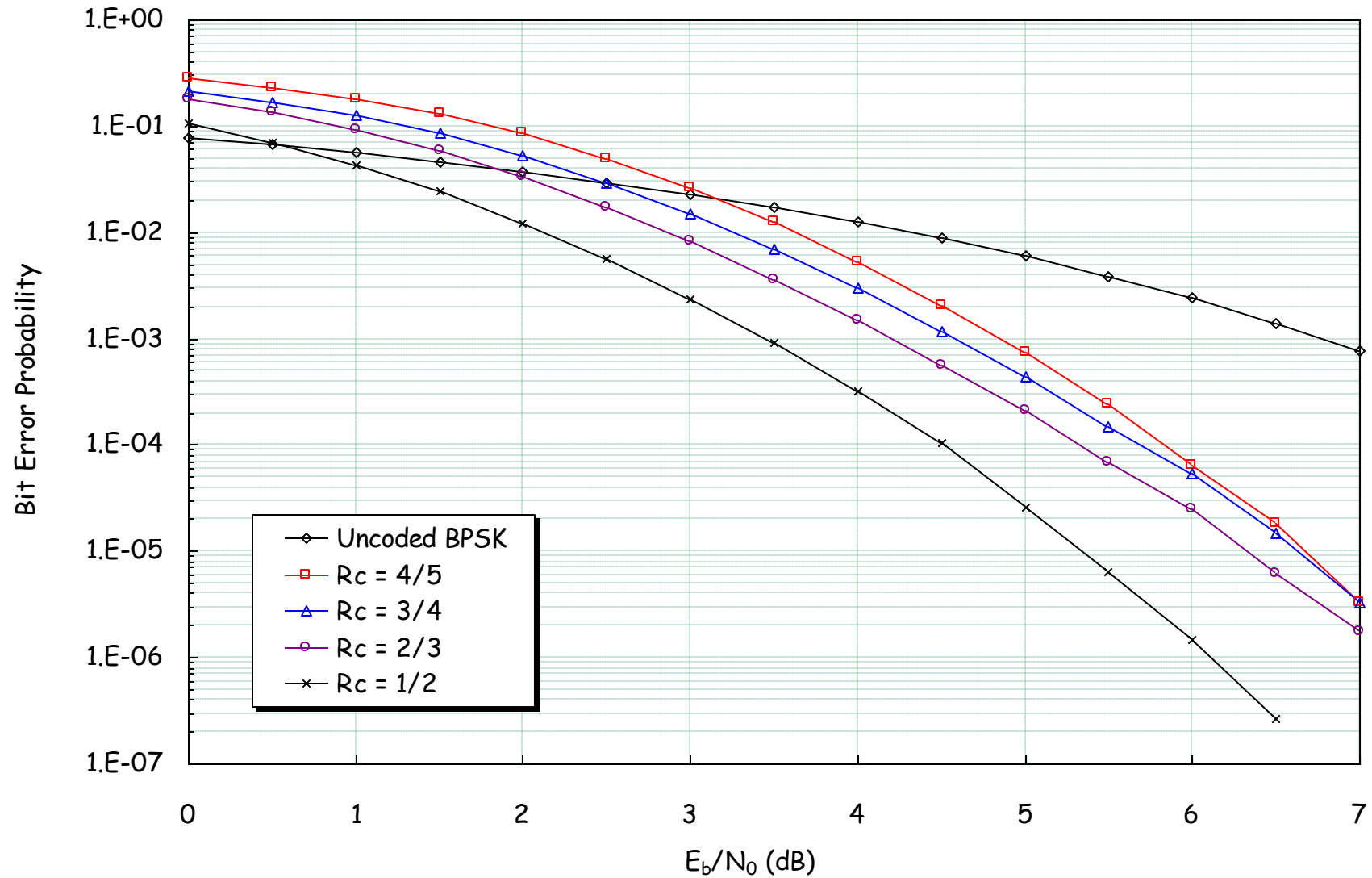
Rate-3/4 CC obtained using  $P = \begin{bmatrix} 110 \\ 101 \end{bmatrix}$

Rate-4/5 CC obtained using  $P = \begin{bmatrix} 10101101 \\ 11011010 \end{bmatrix}$

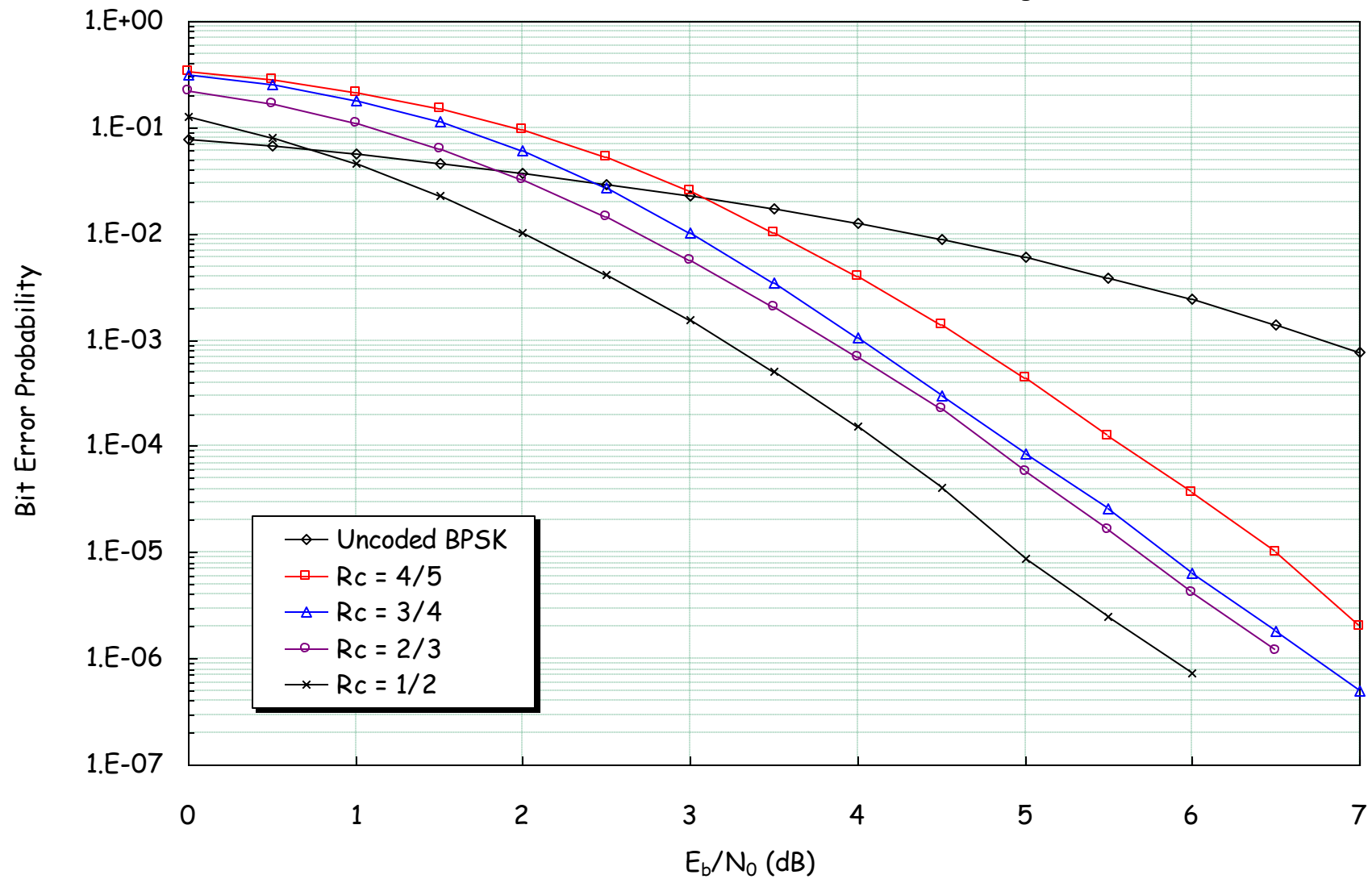
$K = 3, (5, 7)$ , soft-decision Viterbi decoding



$K = 4, (15, 17)$ , soft-decision Viterbi decoding



$K = 5, (23, 35)$ , soft-decision Viterbi decoding

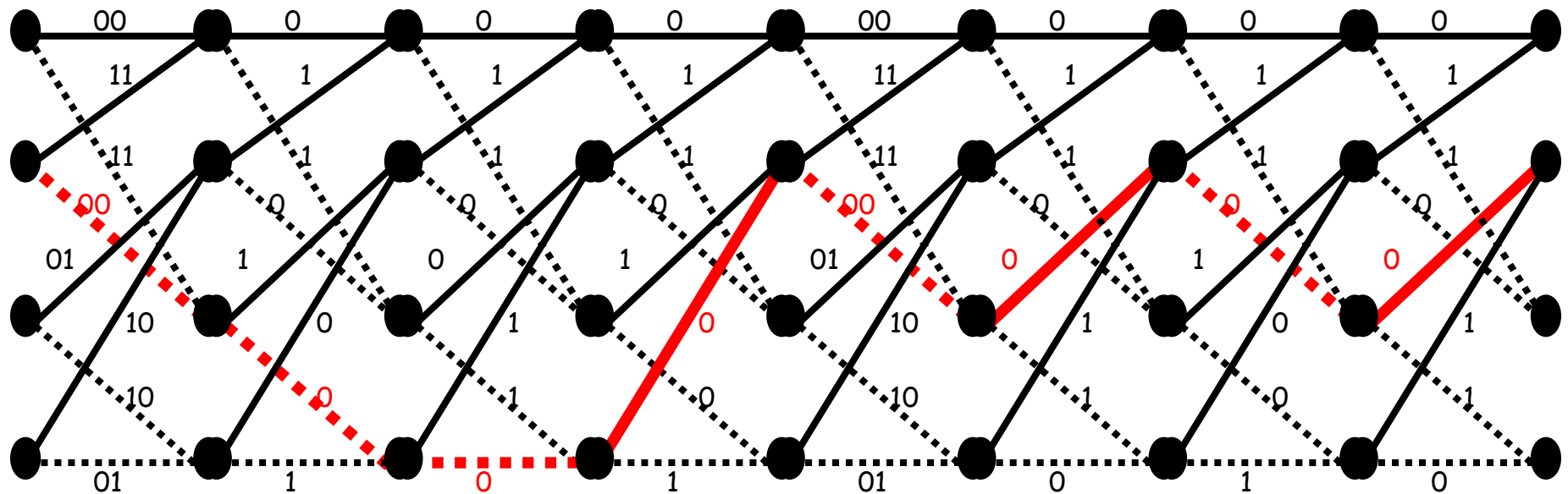


# Punctured convolutional codes

What is wrong with the (5, 7),  
rate-4/5 code ?

Consider a trellis section over the  
length of the puncturing pattern

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



# Punctured convolutional codes

Through the trellis, there is a weight-0 path!

In other words, a message with infinite weight can generate a codeword with finite weight. This should never happen with a convolutional encoder.

=> The code is said to be catastrophic.

Try the following puncturing pattern instead:

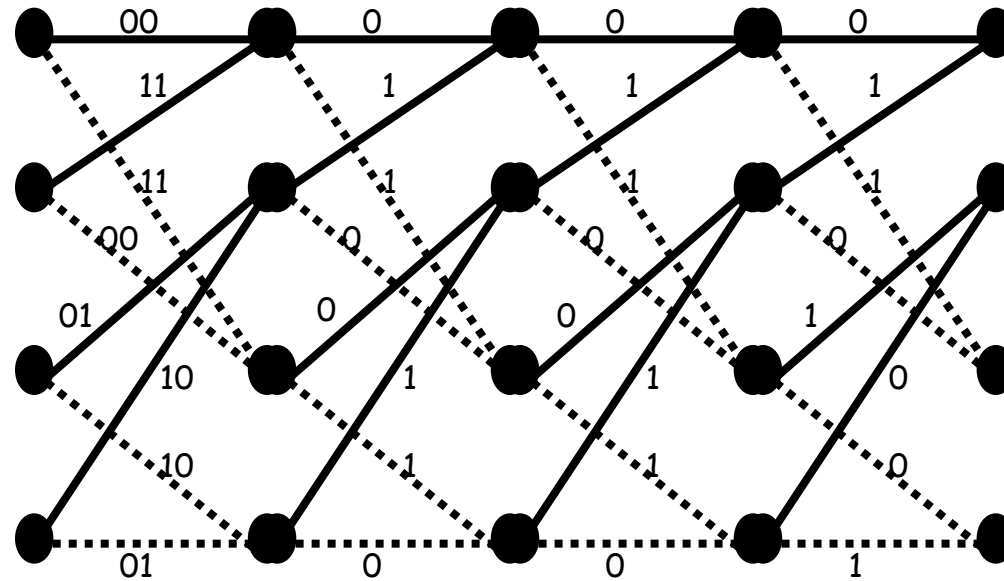
$$P = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$



# Punctured convolutional codes

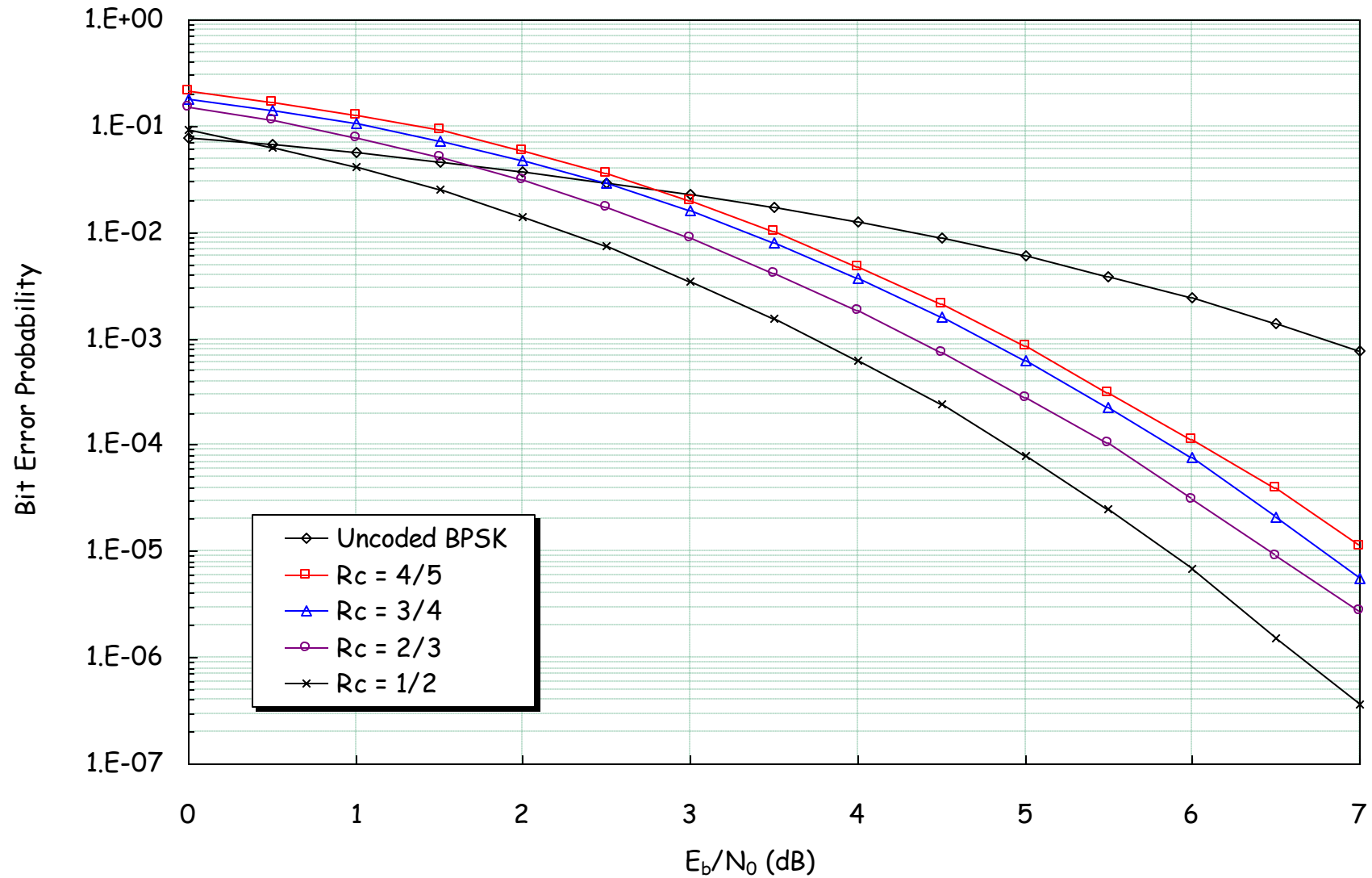
There is no longer any weight-0 path through the trellis.

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



Design the puncturing pattern carefully to guarantee good error performance.

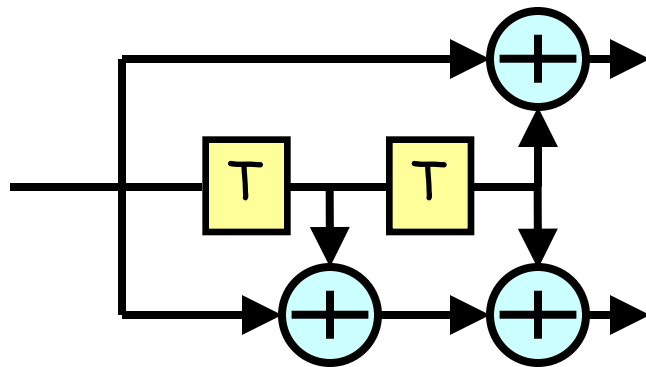
$K = 3, (5, 7)$ , soft-decision Viterbi decoding



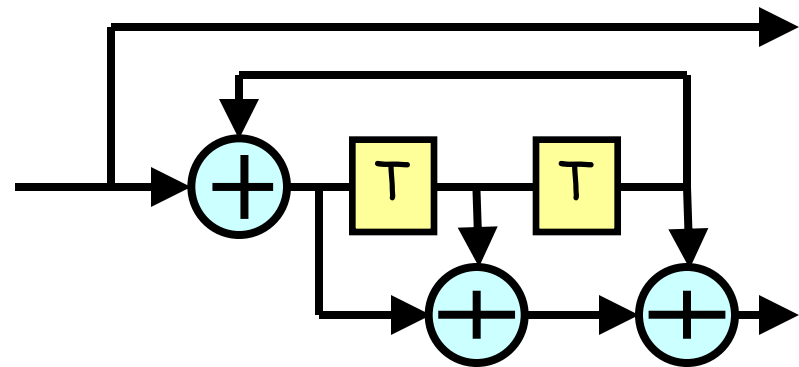
# Recursive Systematic Convolutional codes

RSC codes are convolutional codes with interesting properties.

Non-recursive convolutional (NRC) encoder



Equivalent RSC encoder



Systematic codes cannot be catastrophic. 😊

# Recursive Systematic Convolutional codes

Systematic codes, whether recursive or not, often perform better than non-systematic codes at low SNR.

This can be another interesting property, especially when trying to operate close to the capacity limit...

Non-recursive systematic codes are often outperformed at high SNR by their non-systematic equivalent codes, due to their smaller free distance (in general).

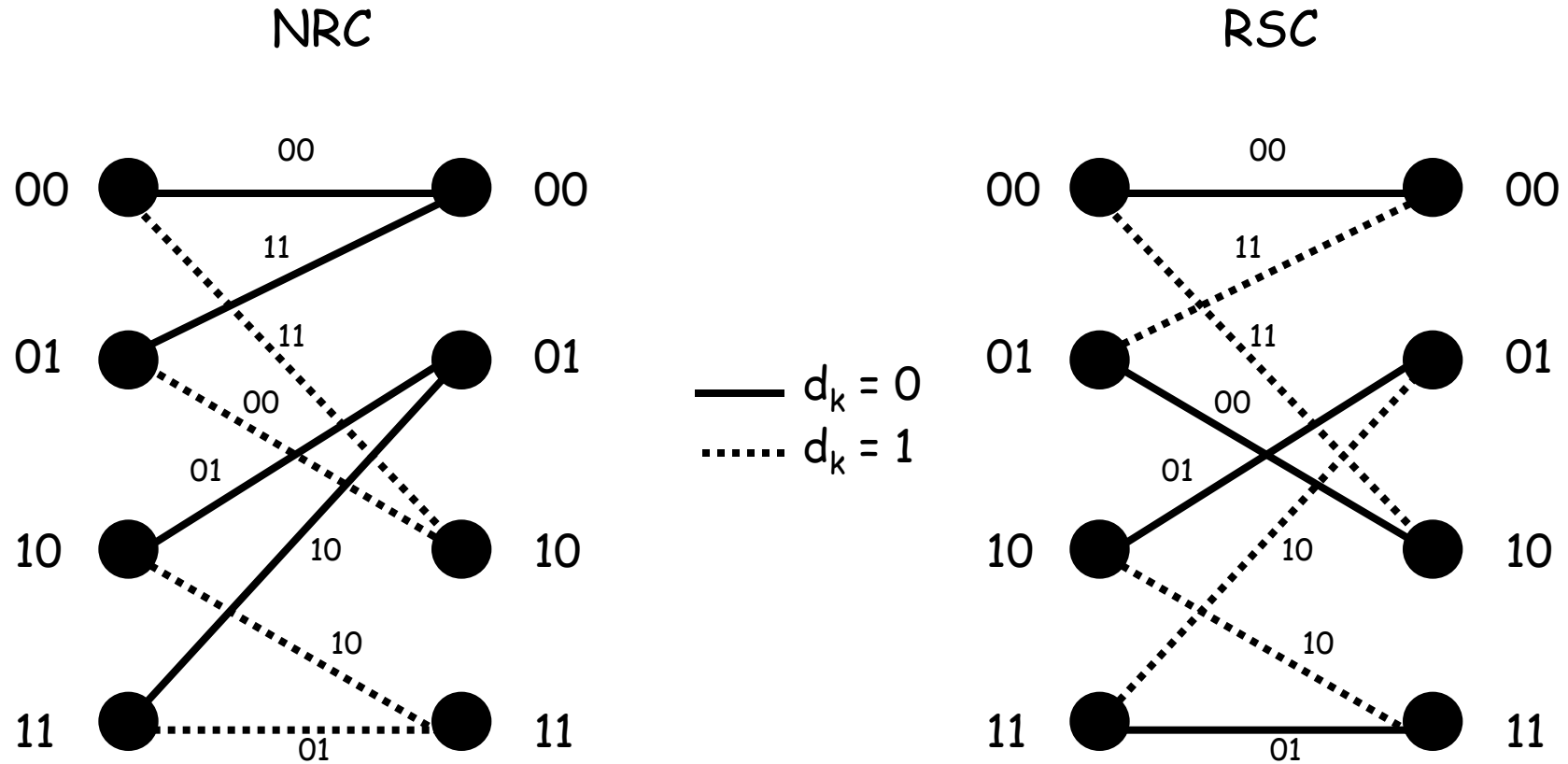
# Recursive Systematic Convolutional codes

This is however not true for RSC codes because these codes, despite being systematic, are as good as non-systematic codes at high SNR.

The reason is simply that a NRC code and its RSC equivalent have the same free distance.

In summary, RSC codes combine the advantages of both the systematic codes (which is not surprising since they are systematic) and the non-systematic ones.

# Recursive Systematic Convolutional codes



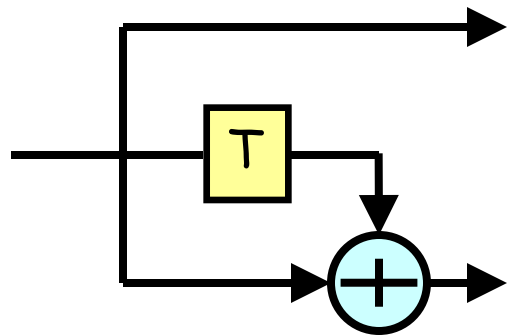
Identical free distance  $d_{\text{free}} = 5$

# Recursive Systematic Convolutional codes

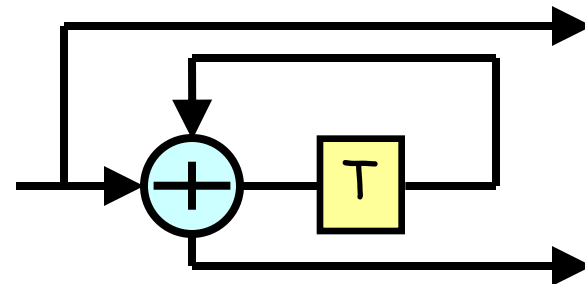
Let us compare the distance spectrum of an NRC code with that of its equivalent RSC code.

The  $(5, 7)$  code is too complicated for manual analysis. Let us instead focus on the  $(2, 3)$  code.

$(2, 3)$  NRC encoder

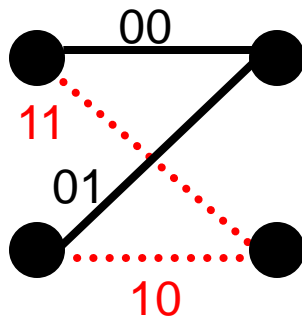
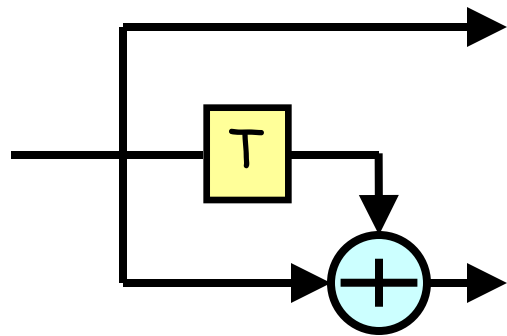


$(2, 3)$  RSC code



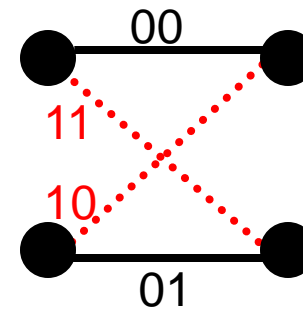
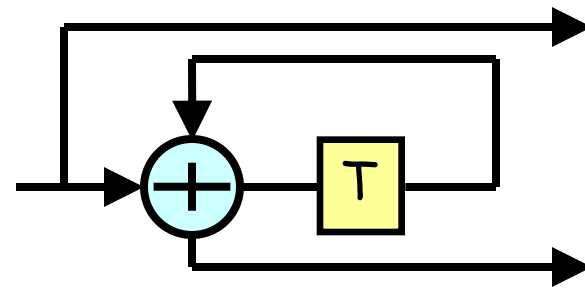
# Recursive Systematic Convolutional codes

(2, 3) NRC encoder



—— Input bit = 0  
..... Input bit = 1

(2, 3) RSC code





# Recursive Systematic Convolutional codes

We are going to compare the first 5 terms in the distance spectrum for both  $(2, 3)$  codes.

The results are already available for the NRC code.

We assume that the trellis of both codes is terminated.

Let us focus on the  $(2, 3)$  RSC code.

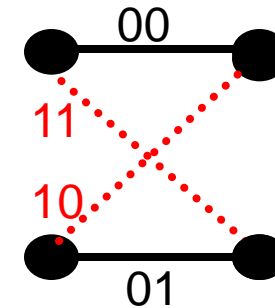
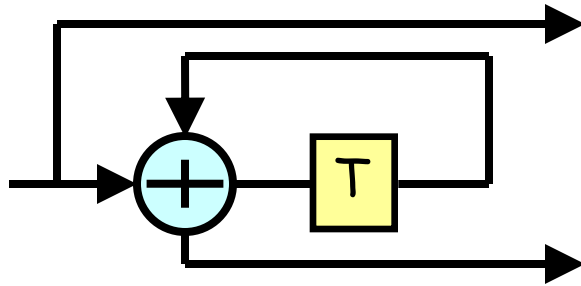
# Recursive Systematic Convolutional codes

We assume a tail bit is appended at the end of the message  $M$  in order to terminate the trellis of the RSC encoder in the zero state.

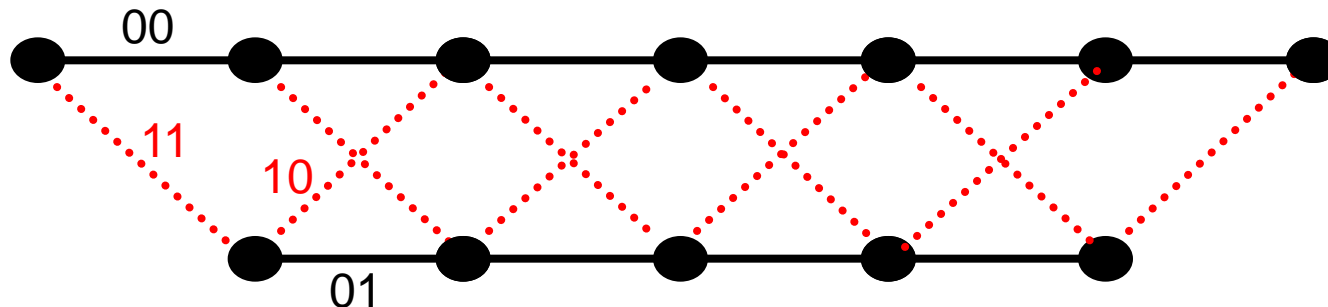
Appending a tail bit at the end of a  $k$ -bit message results in a negligible data rate loss since  $k \gg 1$  in practice.

However, unlike what happens with an NRC code, the tail bit is not necessarily equal to 0.

# Recursive Systematic Convolutional codes



—— Input bit = 0  
 ..... Input bit = 1



$M = (10000\underline{1}) \Rightarrow 1$  tail bit (= 1 in this case) is sufficient to return to the zero state.

# Recursive Systematic Convolutional codes

To return to the zero state, another 1 is needed in the message.

As long as the 1<sup>st</sup> 1 is followed by 0s, the encoder does not return to the zero state.

Messages with weight-1, 3, 5, 7, etc. cannot exist at the (2, 3) RSC encoder input.

In a terminated NRC encoder, any weight would be acceptable in the message as NRC encoders are terminated by appending a string of 0s as tail bits.

# Recursive Systematic Convolutional codes

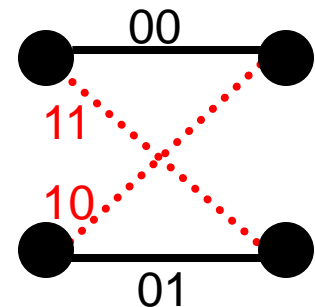
Consider weight-2 messages first.

If the two 1s are separated by  $L$  0s, the corresponding codeword is at distance  $d(L) = 3 + L$ ,  $L \in \{0, \dots, k-2\}$ , from the all-zero codeword.

There are  $(k-L-1)$  configurations in which two 1s are separated by  $L$  0s.

Ex:  $M = (\dots 0 \mathbf{1000000001} 0 \dots)$

$\Rightarrow C = (\dots 00 \mathbf{110101010101010110} 00 \dots)$



# Recursive Systematic Convolutional codes

Weight-2 messages can generate codewords at large distance from the all-zero codeword.

With an RSC code, a small-weight message often generates a large-weight codeword because, in a long small-weight message, the few 1s are more likely to be isolated rather than grouped together...

A NRC code does not exhibit such characteristic as, with such an encoder, a small-weight message always generates a small-weight codeword ☹️

# Recursive Systematic Convolutional codes

The contribution of the weight-2 messages to the union bound are the following  $(k-1)$  error coefficients, for  $L \in \{0, \dots, k-2\}$ :

$$e(d = L+3) = \frac{2 \cdot (k - L - 1)}{2k}$$

The first 5 error coefficients are thus given by

$$e(d = 3) \approx 1 \qquad e(d = 4) \approx 1 \qquad e(d = 5) \approx 1$$

$$e(d = 6) \approx 1 \qquad e(d = 7) \approx 1$$

# Recursive Systematic Convolutional codes

Consider now the weight-4 messages.

Consider messages, denoted as  $M(L, L')$ , where the first two 1s are separated by  $L$  0s and the last two 1s are separated by  $L'$  0s.

The corresponding codeword is at distance  $d(L, L') = 3 + L + 3 + L' = 6 + L + L'$ ,  $L$  and  $L' \in \{0, \dots, k-4\}$ , from the all-zero codeword.

Ex:  $M = (...0\mathbf{10001}0000\mathbf{1001}0...)$ , i.e.  $L = 3$  and  $L' = 2$

$\Rightarrow C = (...00\mathbf{1101010110}000000000\mathbf{11010110}00...)$ ,  $d = 11$



# Recursive Systematic Convolutional codes

The number of weight-4 messages  $M(L, L')$  is given by

$$\binom{k-L-L'-2}{2} = \frac{(k-L-L'-2) \cdot (k-L-L'-3)}{2}$$

If  $k \gg L + L'$ , this number is equal to  $\approx \frac{k^2}{2}$ .

The contribution of the weight-4 messages to the first three error coefficients  $e(d=3)$ ,  $e(d=4)$ , and  $e(d=5)$  is 0 since no message  $M(L, L')$  can lead to a codeword with a Hamming weight smaller than 6.

# Recursive Systematic Convolutional codes

The contribution of the weight-4 messages to the error coefficient  $e(d=6)$  is only due to the messages  $M(L, L')$  with  $L = L' = 0$ .

Therefore, this contribution is represented by the error coefficient:

$$e(d=6) \approx \frac{4k^2}{4k} = k$$

The contribution of the weight-4 messages to the error coefficient  $e(d=7)$  is due to the messages  $M(L, L')$  with either  $(L = 0, L' = 1)$  or  $(L = 1, L' = 0)$ .

# Recursive Systematic Convolutional codes

Therefore, this contribution is represented by the error coefficient:

$$e(d = 7) \approx \frac{4k^2}{4k} + \frac{4k^2}{4k} = 2k$$

Messages with weights 6, 8 and so on do not have to be considered because they lead to codewords at a distance  $d > 7$  from the all-zero codeword.

Finally, by putting all these results together, we obtain the error coefficients for the first 5 terms in the union bound:

# Recursive Systematic Convolutional codes

$$P_{eb} \leq \operatorname{erfc}\left(\sqrt{\frac{3}{2} \frac{E_b}{N_0}}\right) + \operatorname{erfc}\left(\sqrt{\frac{4}{2} \frac{E_b}{N_0}}\right) + \operatorname{erfc}\left(\sqrt{\frac{5}{2} \frac{E_b}{N_0}}\right) + \dots$$
$$\dots + (k+1) \cdot \operatorname{erfc}\left(\sqrt{\frac{6}{2} \frac{E_b}{N_0}}\right) + (2k+1) \cdot \operatorname{erfc}\left(\sqrt{\frac{7}{2} \frac{E_b}{N_0}}\right) + \dots$$

$$e(d=3) \approx e(d=4) \approx e(d=5) \approx 1 \quad e(d=6) \approx k+1 \approx k$$

$$e(d=7) \approx 2k+1 \approx 2k$$

This distance spectrum is similar to that of the (2, 3) NRC code. So, what is the big deal about RSC codes?

# Recursive Systematic Convolutional codes

(2, 3) NRC code

$$\approx \frac{1}{2}$$

$$\approx 1$$

$$\approx \frac{3}{2}$$

$$\approx \frac{k}{2}$$

$$\approx \frac{3k}{2}$$

$$e(d=3)$$

$$e(d=4)$$

$$e(d=5)$$

$$e(d=6)$$

$$e(d=7)$$

(2, 3) RSC code

$$\approx 1$$

$$\approx 1$$

$$\approx 1$$

$$\approx k$$

$$\approx 2k$$

# BCH codes (1959)

1959: Raj C Bose (1901 - 1987) and D K Chaudhuri propose a new class of multiple-error correcting linear block codes, discovered independently by A. Hocquenghem → BCH codes.

BCH codes are cyclic codes. A code is said to be cyclic if any shift of a codeword is also a codeword.

Cyclic codes have considerable algebraic structure, which simplifies the encoding and decoding procedures.

# BCH codes

For any positive integers  $m \geq 3$  and  $t < 2^{m-1}$ , there is a binary BCH code with  $n = 2^m - 1$ ,  $k = n - mt$ ,  $d_{\min} = 2t + 1$ .

Ex:  $t = 1 \rightarrow$  Hamming codes

Ex:  $t = 2$ ,  $d_{\min} = 5$

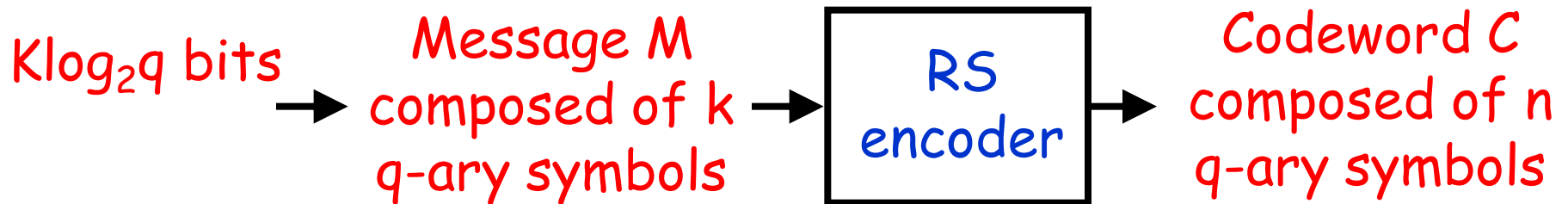
- $n = 15$ ,  $k = 7$
- $n = 31$ ,  $k = 21$
- $n = 63$ ,  $k = 51$
- $n = 127$ ,  $k = 113$
- $n = 255$ ,  $k = 239$  etc.

Decoded using either the Berlekamp algorithm or the Euclidean algorithm.

# Reed-Solomon codes (1960)

1960: Irving S Reed and Gustave Solomon (1930 - 1996) develop a block coding scheme particularly powerful for correcting bursts of errors → Reed-Solomon codes.

RS codes are non-binary cyclic codes (subset of non-binary BCH codes). Each codeword  $C$  is composed of  $n$   $q$ -ary symbols, where  $q$  is a power of 2.





# Reed-Solomon codes

A RS code is defined by the following parameters:

$n = q-1$ ,  $k = 1, 3, \dots, n-2$ , and  $d_{\min} = n - k + 1 \Rightarrow t = (n-k)/2$ .

Ex:  $q = 256$  (1 symbol  $\equiv$  8 bits)

- $n = 255, k = 223 \rightarrow d_{\min} = 33 \rightarrow t = 16$
- $n = 255, k = 239 \rightarrow d_{\min} = 17 \rightarrow t = 8$

RS codes are particularly powerful for situations where errors tend to happen in "bursts" rather than randomly (e.g., CD-Rom).

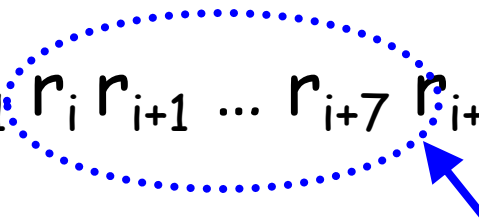
# Reed-Solomon codes

Ex:  $(n = 255, k = 239)$  RS code with  $d_{\min} = 17$  ( $t = 8$ )

Such code can correct up to 8 256-ary symbols in a received word of 255 256-ary symbols.

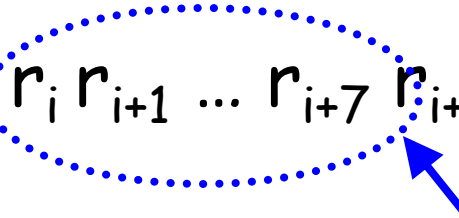
Length of a codeword  $C = (c_0 c_1 c_2 \dots c_{254}) = 2040$  bits.

Assume we receive the word

$$R = (r_0 r_1 r_2 \dots r_{i-1} r_i r_{i+1} \dots r_{i+7} r_{i+8} \dots r_{254})$$


Error burst  $\equiv$  8 consecutive symbols are received erroneously  
(1 bit out of 8 is wrong or all 8 bits are wrong)

# Reed-Solomon codes

$$R = (r_0 \ r_1 \ r_2 \ \dots \ r_{i-1} \ r_i \ r_{i+1} \ \dots \ r_{i+7} \ r_{i+8} \ \dots \ r_{254})$$


Such error burst can be corrected since  $t = 8$

If all bits in every symbol are wrong  $\rightarrow$  Up to 64 bits in a received word of 2040 bits can be corrected.

If only one bit per symbol is wrong  $\rightarrow$  Up to 8 bits in a received word of 2040 bits can be corrected

$\rightarrow$  RS codes are very attractive for correcting error bursts.

# LDPC codes (1962)

1962: Robert G Gallager introduces low-density parity-check (LDPC) codes.



LDPC codes can perform very close to the channel capacity limit.

But nobody noticed it, and LDPC codes were somewhat forgotten until 1995.

LDPC codes have been a very hot research topic over the last decade.

# LDPC codes

LDPC codes are linear block codes.

=> They can be completely defined using a parity-check matrix  $H$ .

A binary word  $C$  is a valid codeword if and only if  $C$  satisfies the equation  $C.H^T = 0$ .

The parity-check matrix of a LDPC code has a low density of 1s (sparse matrix) and the number of 1s in common between any two columns is either 0 or 1.

# LDPC codes

Ex: (3, 3)-regular LDPC code

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{aligned} c_0 + c_1 + c_3 &= 0 \\ c_1 + c_2 + c_4 &= 0 \\ c_2 + c_3 + c_5 &= 0 \\ c_3 + c_4 + c_6 &= 0 \\ c_0 + c_4 + c_5 &= 0 \\ c_1 + c_5 + c_6 &= 0 \\ c_0 + c_2 + c_6 &= 0 \end{aligned}$$

The code is completely defined by a set of parity-check equations

When  $H$  is large, A LDPC code can be made look like a random code. But, how to implement efficient decoding?

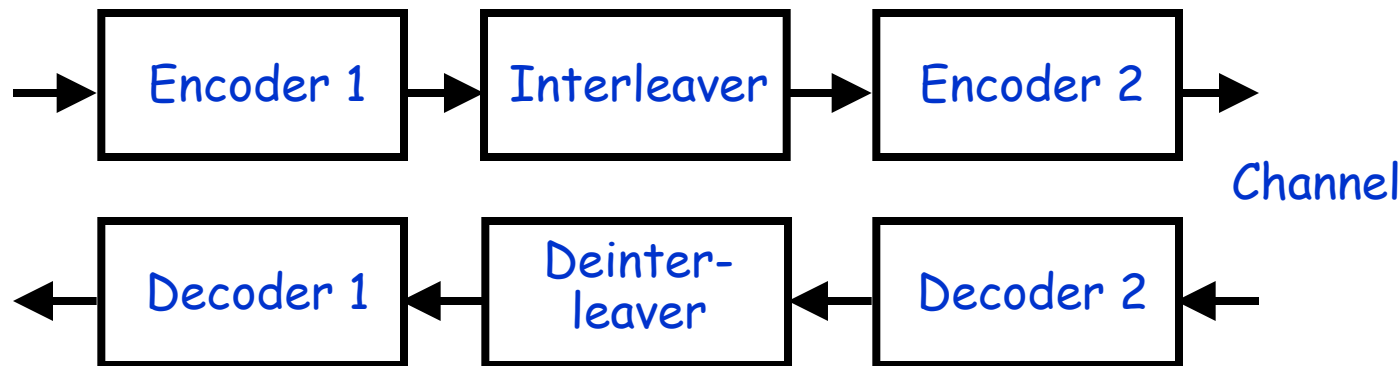
# LDPC codes

LDPC codes were quickly forgotten because their true potential could not be unveiled (no ML or near-ML decoding algorithm, no powerful computers to perform simulations during early 60s).

In addition, a new technique, called "concatenation of codes", was introduced at about the same time, and seemed more promising than LDPC codes for practical applications.

# Concatenation of codes (1966)

1966: G David Forney proposes to concatenate codes for improving error performance without having to use complex codes.



NASA standard in the mid-90s:

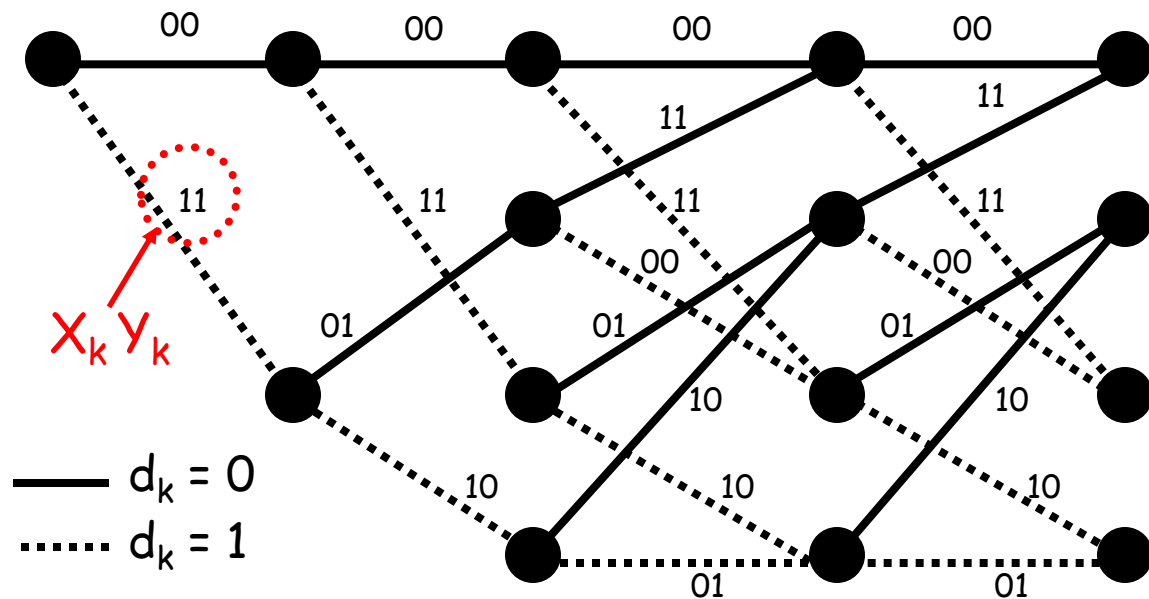
Code 1 (outer code): Reed-Solomon code;

Code 2 (inner code):  $R_c = \frac{1}{2}$ ,  $K = 7$ , (133, 171) CC  
decoded using soft-decision Viterbi decoding.



# Viterbi decoding algorithm (1967)

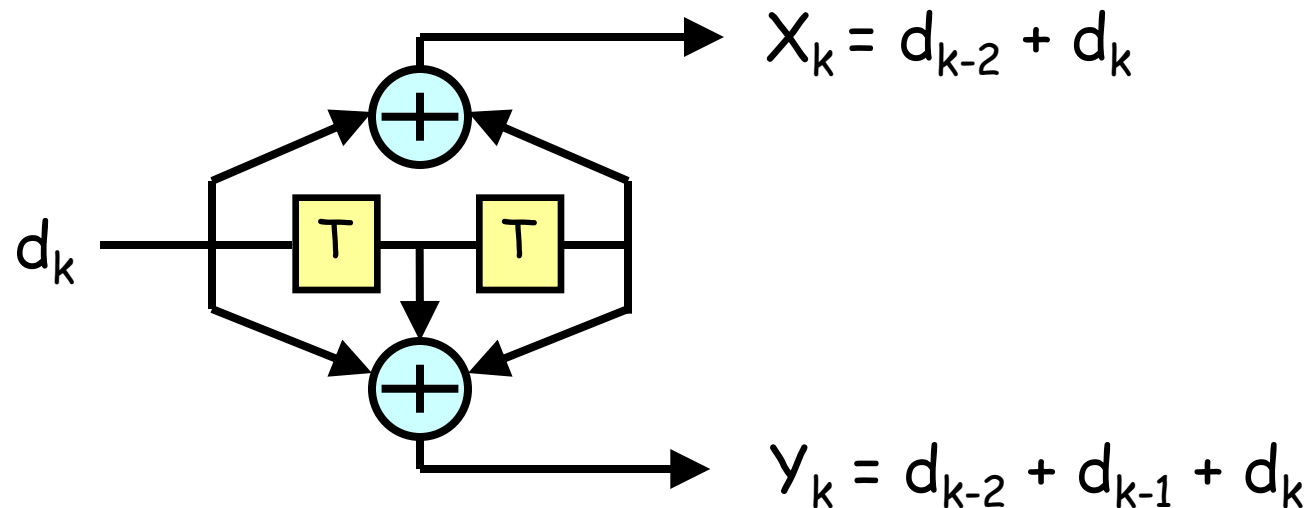
1967: Andrew J Viterbi introduces a practical ML decoding algorithm for CCs (the "Viterbi algorithm").



Trellis representation of the  $R_c = \frac{1}{2}$ ,  $K = 3$ ,  $(5, 7)$  CC, used for Viterbi decoding

# Viterbi decoding algorithm

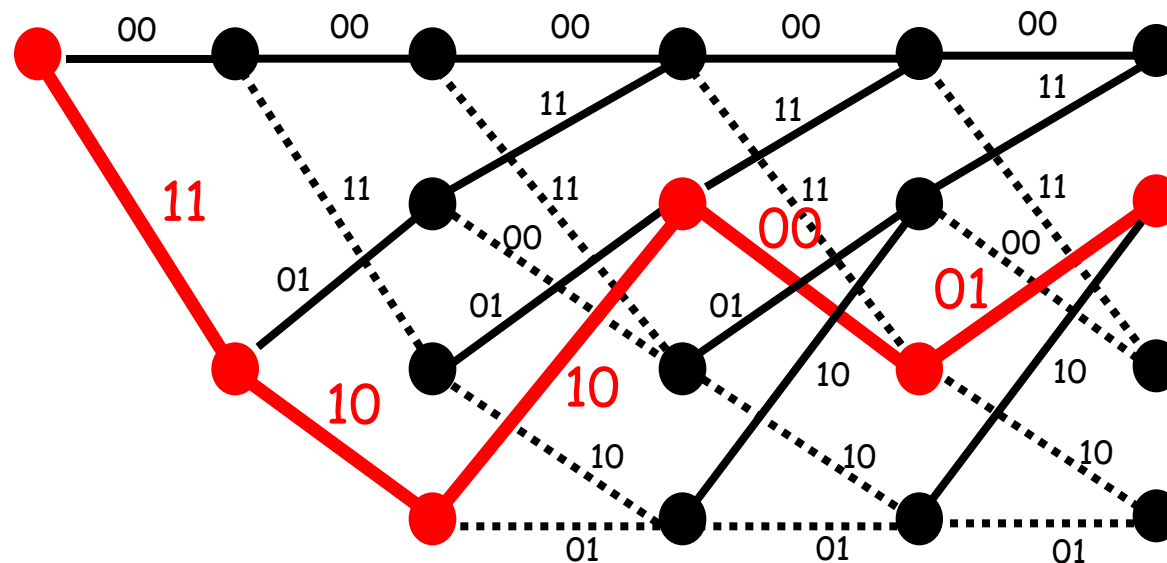
To explain the Viterbi decoding algorithm, let us consider the 4-state rate-1/2 CC previously studied.



Assume the info sequence  $\{d_k\} = \{11010\}$ .

# Viterbi decoding algorithm

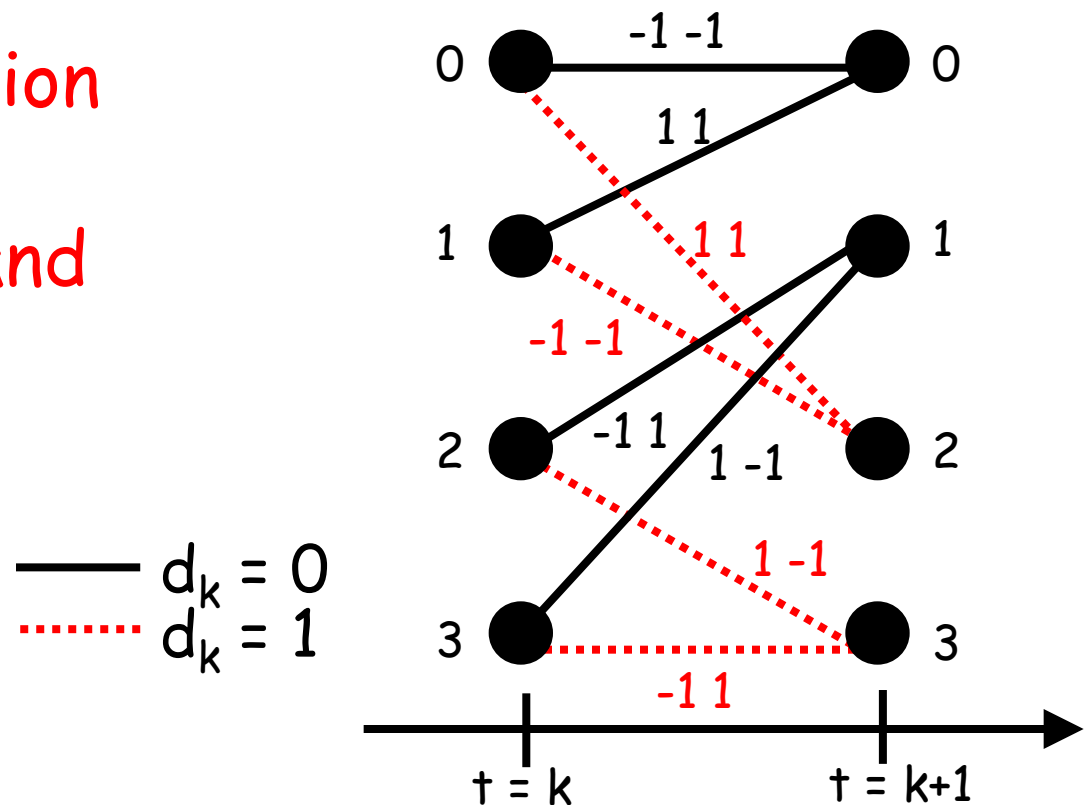
The corresponding coded sequence  $\{X_k Y_k\} = \{11\ 10\ 10\ 00\ 01\}$  is represented by the red path in the trellis of the code.



# Viterbi decoding algorithm

We use the soft-decision Viterbi decoding algorithm, i.e. assume transmission over a BPSK, AWGN channel.

The BPSK modulation scheme is defined such that  $0 \rightarrow -1$  and  $1 \rightarrow +1$ .



# Viterbi decoding algorithm

The algorithm searches for the sequence (codeword) which is at minimal Euclidean distance from the received vector, i.e. the sequence which minimizes the term

$$\sum_{l=0}^9 (r_l - c_{i,l})^2$$

where  $r_l$  is the  $(l+1)$ -th received channel sample, and  $c_{i,l}$  is the  $(l+1)$ -th symbol in a particular codeword  $C_i$ .

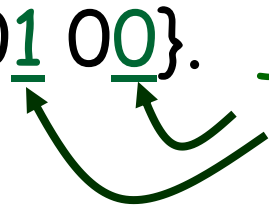
# Viterbi decoding algorithm

Equivalently, and more simply, the algorithm searches for the sequence which maximizes the term

$$\sum_{l=0}^9 r_l \cdot c_{i,l}$$

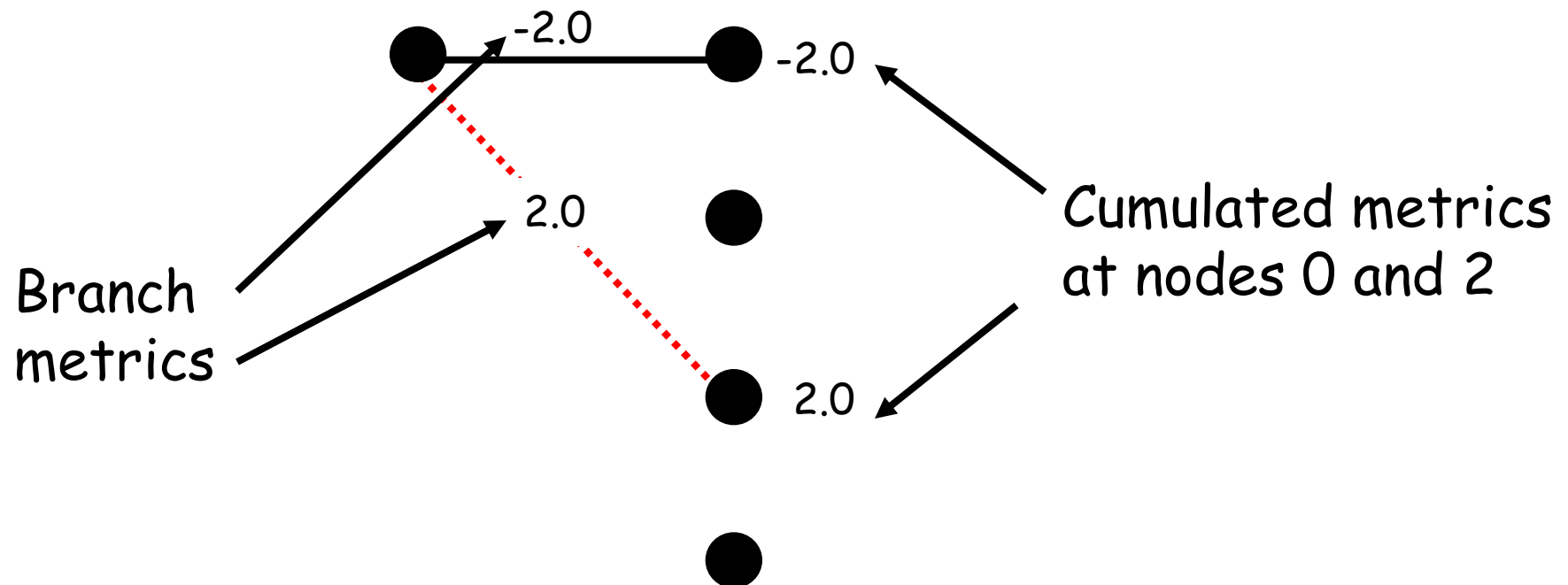
Assume that the received sequence of samples  $r_l$  is  $\{1.3 \ 0.7, 0.2 \ -0.2, 0.9 \ -0.9, -2.2 \ +0.2, -2.5 \ -0.5\}$ , which corresponds to the received binary sequence  $\{11 \ 10 \ 10 \ 01 \ 00\}$ .

Two transmission errors



# Viterbi decoding algorithm

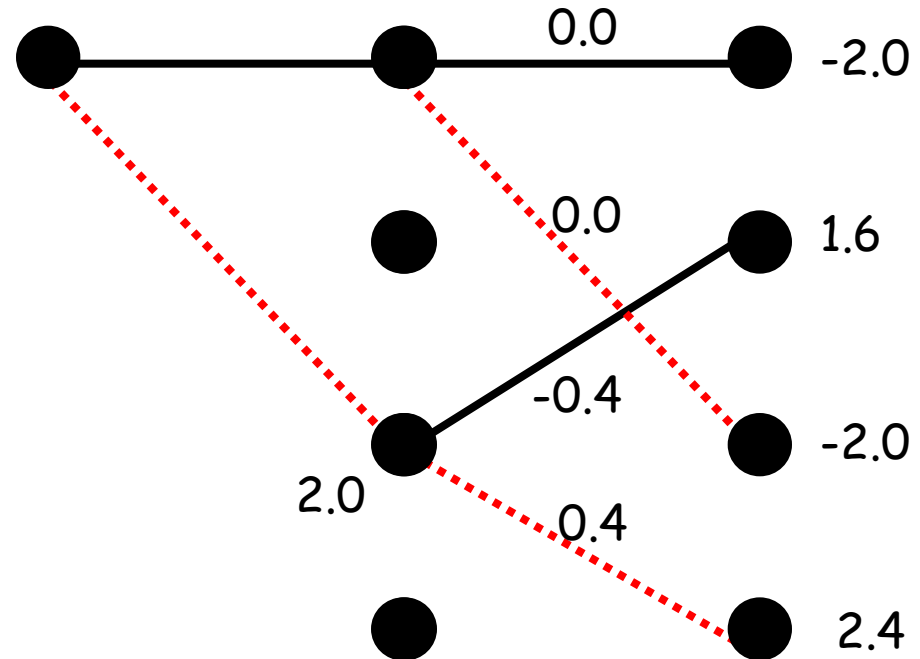
We start in the zero state with the 1<sup>st</sup> received pair  $\{r_0 = 1.3, r_1 = 0.7\}$ . We compute the term  $(r_0.c_{i,0} + r_1.c_{i,1})$  for both branches.



# Viterbi decoding algorithm

We now consider the 2nd pair  $\{r_2 = 0.2, r_3 = -0.2\}$ . We compute the term  $(r_2 \cdot c_{i,2} + r_3 \cdot c_{i,3})$  for the 4 possible branches (4 branch metrics).

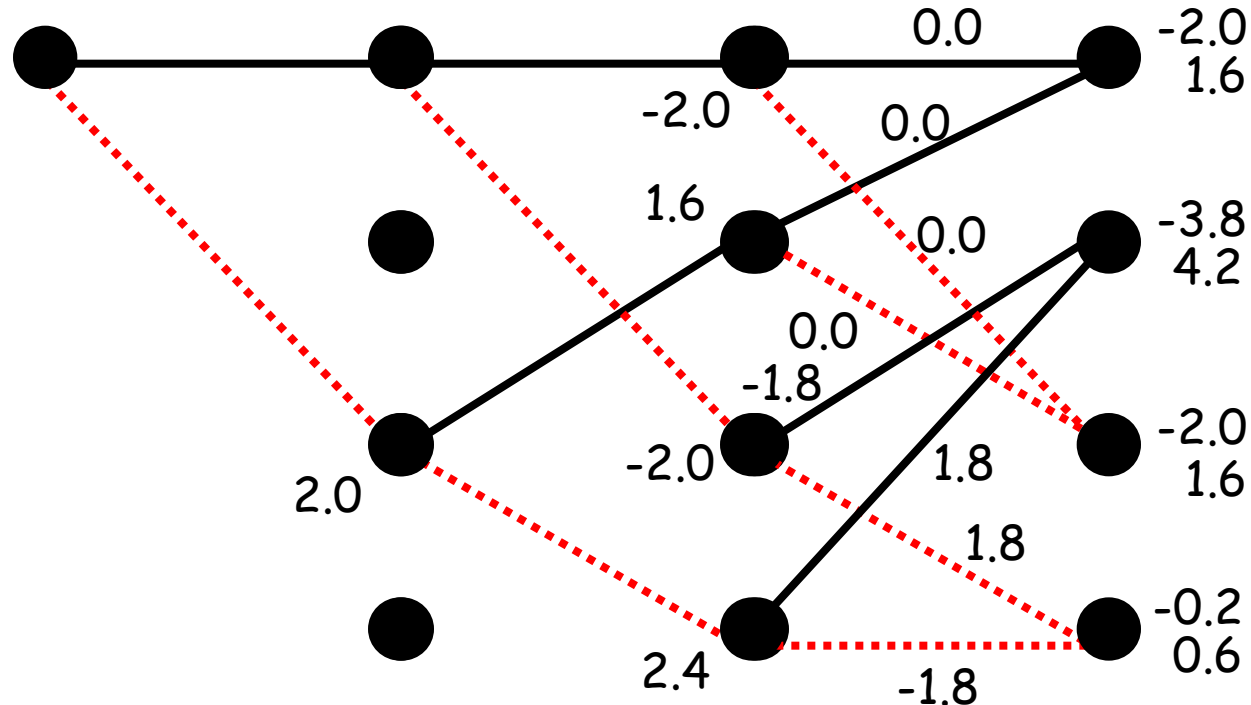
We compute the cumulated metric for each node by adding the branch metric to the cumulated metric of the previous node.





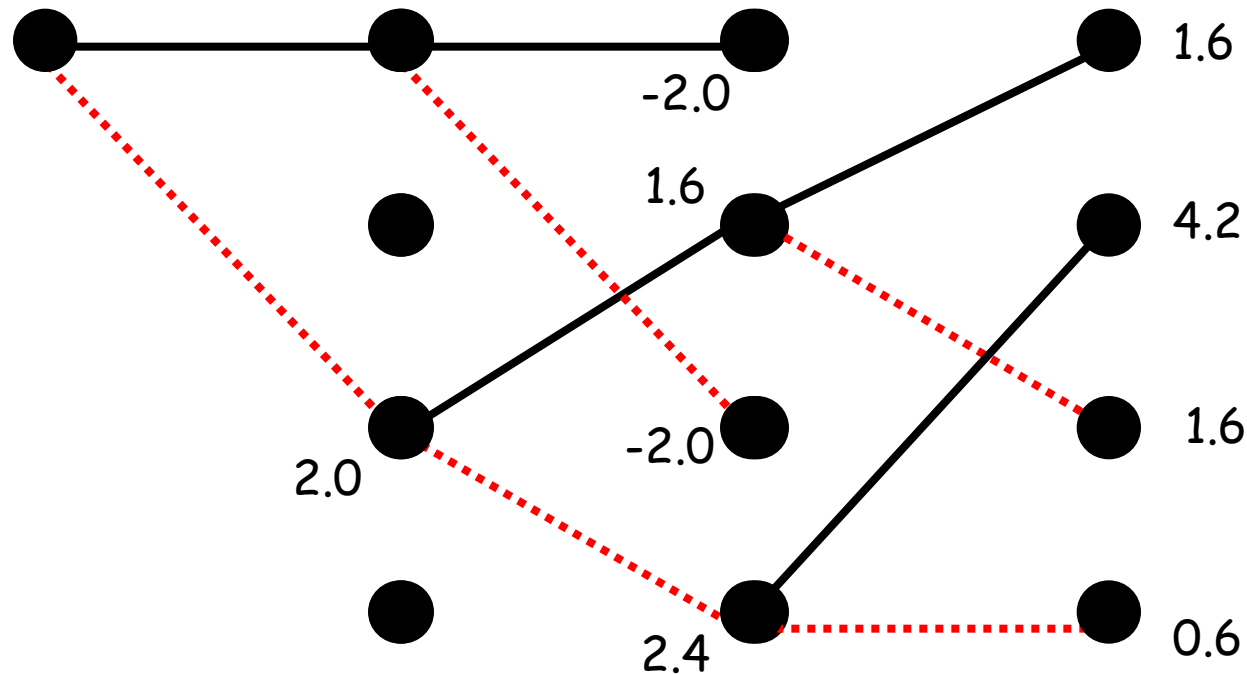
# Viterbi decoding algorithm

We now consider the 3rd pair  $\{r_4 = 0.9, r_5 = -0.9\}$ . We compute the term  $(r_4 \cdot c_{i,4} + r_5 \cdot c_{i,5})$  for the 8 possible branches (8 branch metrics).



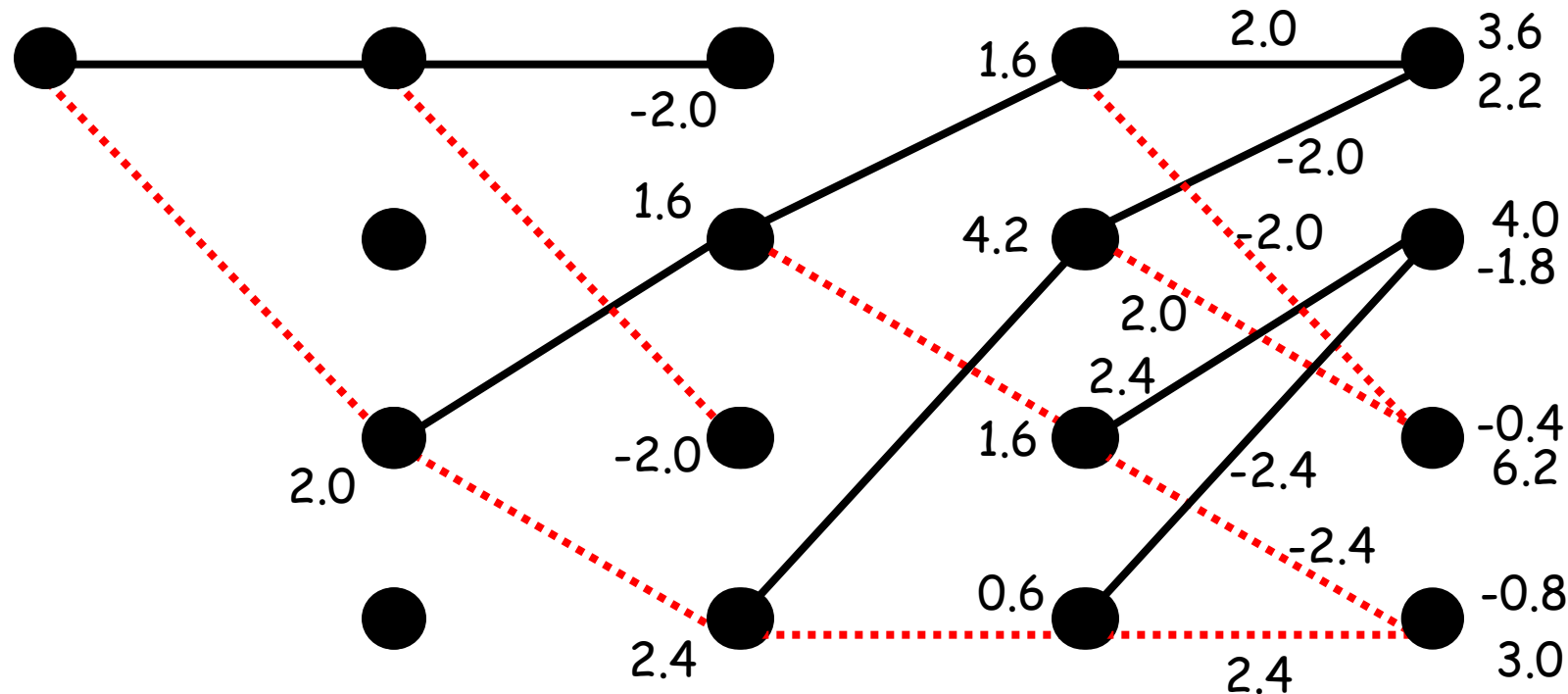
# Viterbi decoding algorithm

For each node, we only keep the branch with the highest cumulated metric, and we repeat the process until the whole received sequence is processed.



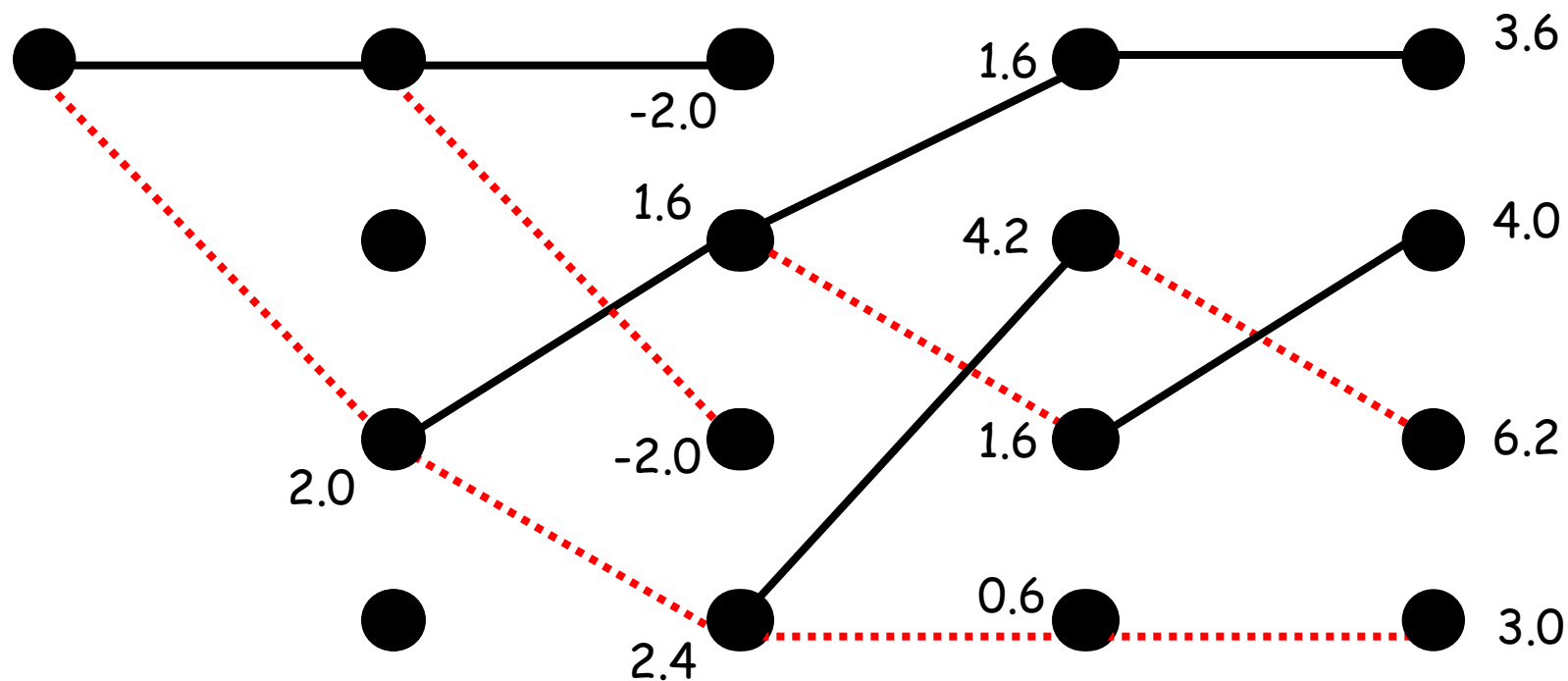
# Viterbi decoding algorithm

We now consider the 4th pair  $\{r_6 = -2.2, r_7 = 0.2\}$ .



# Viterbi decoding algorithm

At every step, half of the possible codewords are suppressed so that the number of possible codewords remains equal to 4 (thanks to the trellis structure).



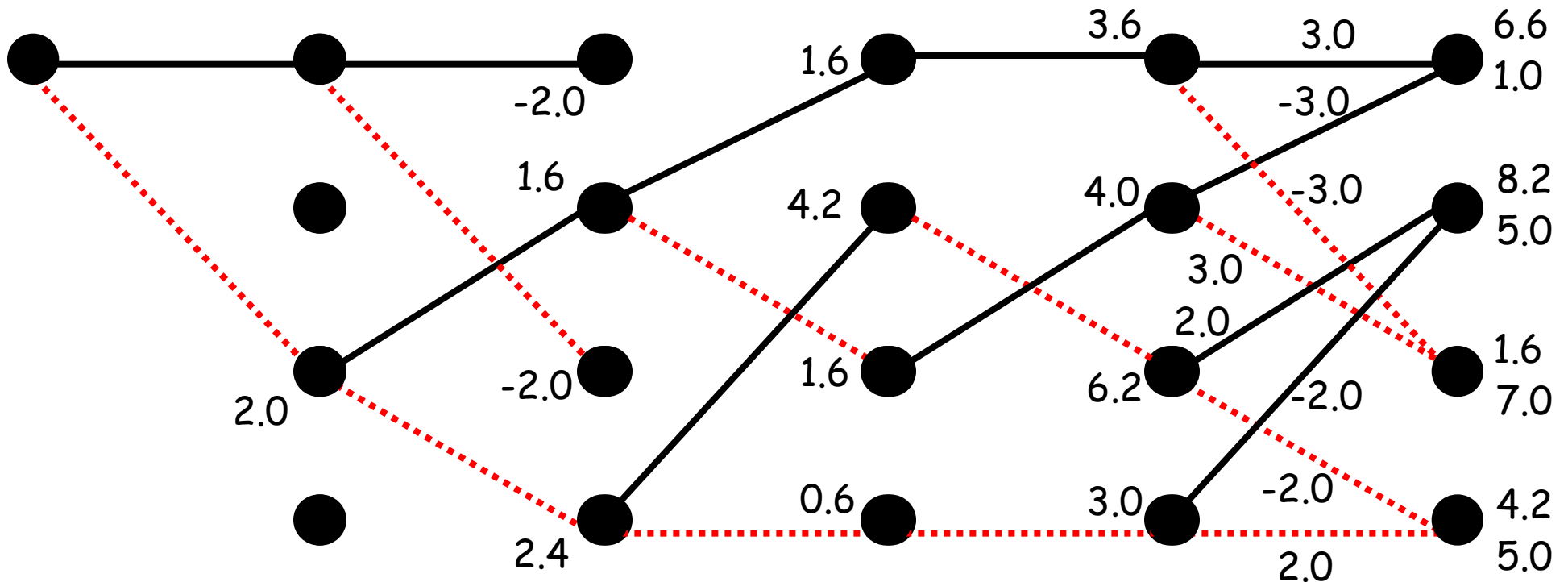
# Viterbi decoding algorithm

The Viterbi algorithm is an ML decoding algorithm and its use therefore leads to optimal error performance at the decoder output.

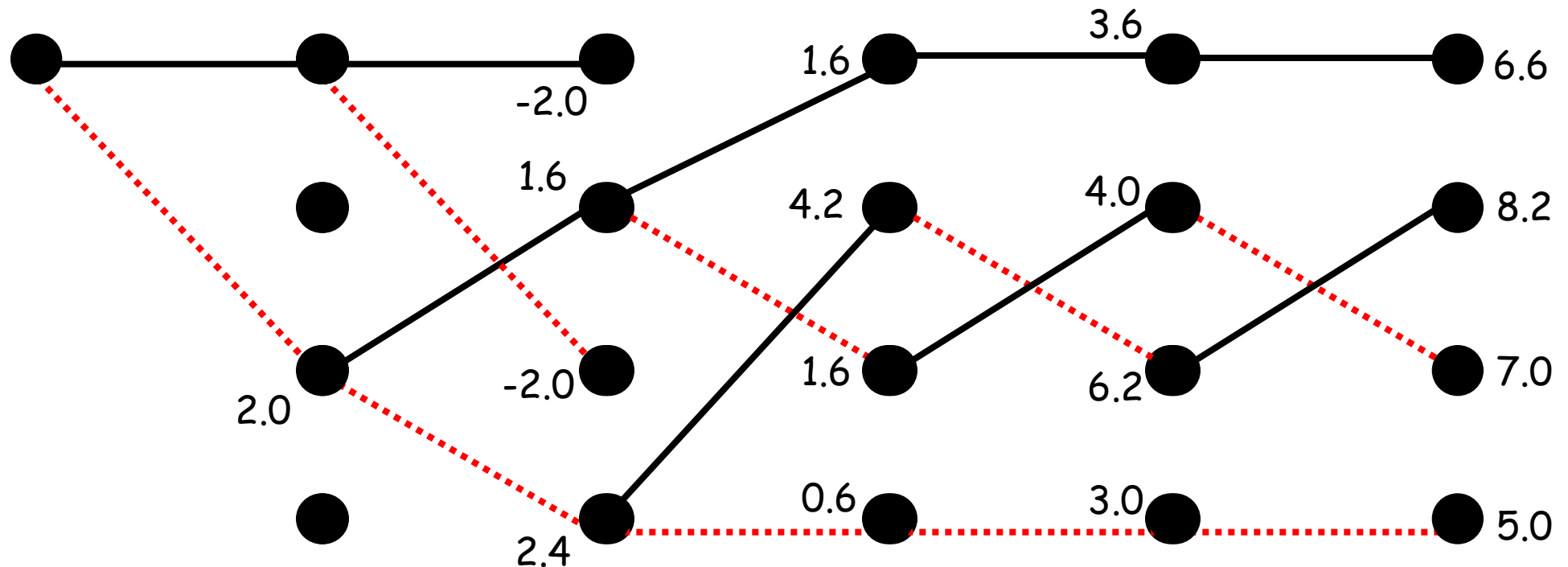
The number of codewords to consider is not prohibitive since, at every step of the decoding process, it is possible to eliminate half of the remaining codewords with the certainty that the ML codeword is being kept.

# Viterbi decoding algorithm

We finally consider the 5<sup>th</sup> pair  $\{r_8 = -2.5, r_9 = -0.5\}$ .



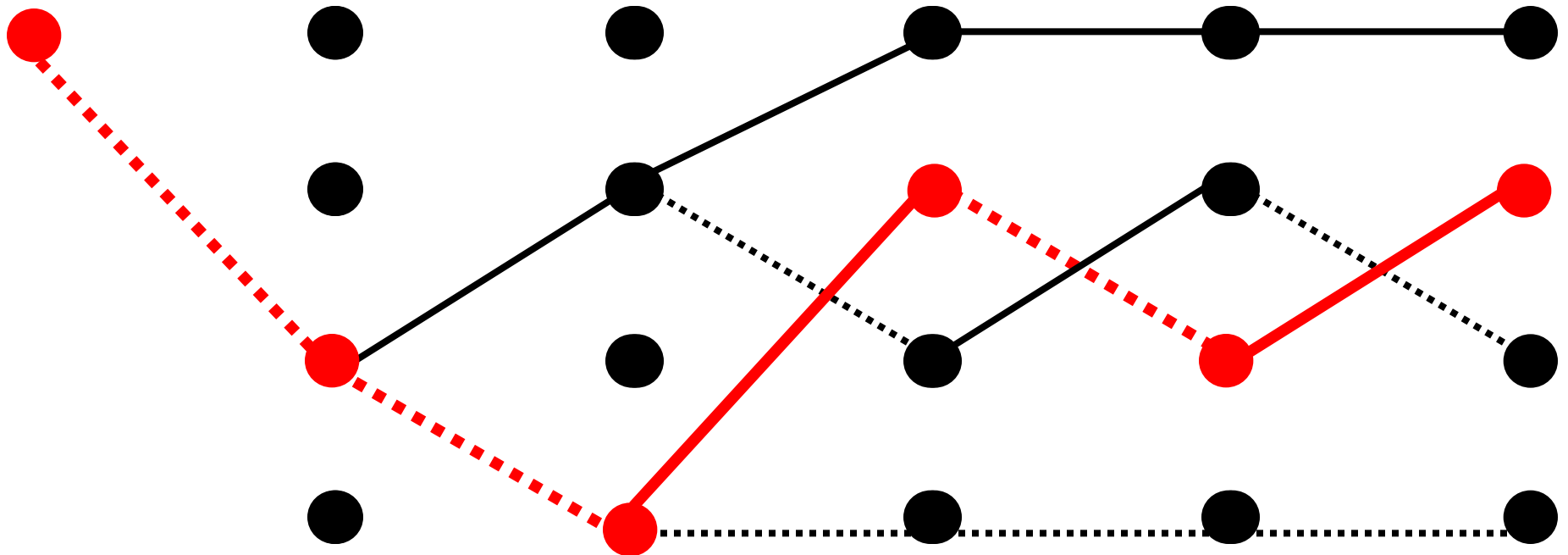
# Viterbi decoding algorithm



Once the received sequence has been processed, we are left with only 4 possible codewords, no matter how long this sequence was.

# Viterbi decoding algorithm

We finally select the path/sequence/codeword having the largest cumulated metric (shown in red below).





# Viterbi decoding algorithm

In this example, despite both transmission errors, we are able to determine the right path in the trellis, i.e. recover the transmitted coded sequence  $\{11\ 10\ 10\ 00\ 01\}$ , which corresponds to the info sequence  $\{11010\}$ .

The Viterbi algorithm can be used for decoding any code whose operation is described using a trellis.

# Trellis-coded modulation (1982)

1982: Gottfried Ungerboeck invents trellis-coded modulation, a coding approach suitable for bandwidth-limited channels → Modems of the 80s, initially.



The redundancy needed for correcting errors is achieved by employing more constellation signal points

→ Unlike traditional coding techniques, TCM coding does not result in bandwidth expansion.

# Trellis-coded modulation

Traditional channel coding reduces bandwidth, which is not acceptable in some applications.

=> We sometimes need to obtain a coding gain without sacrificing bandwidth.

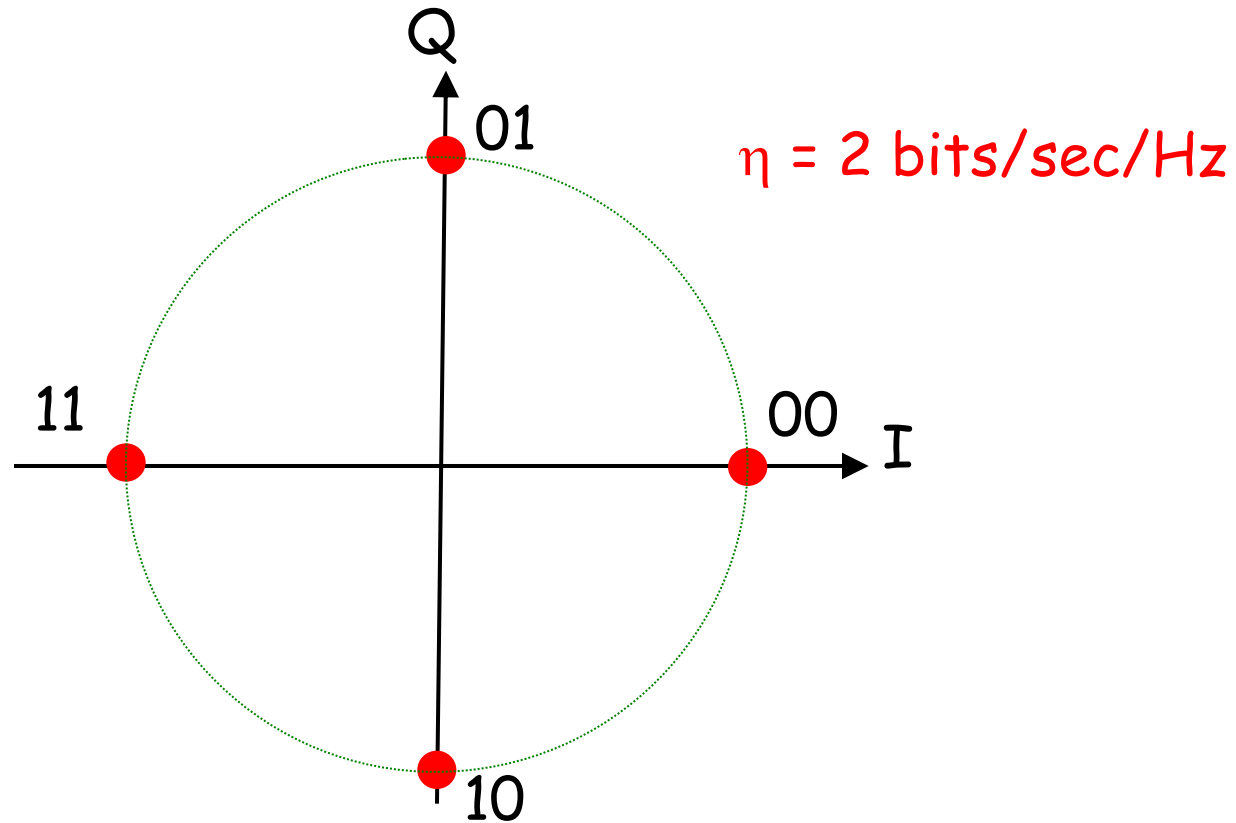
Example: If a spectral efficiency of 2 bits/s/Hz is needed, we can use for example:

(1) Uncoded QPSK or

(2) 8PSK + rate-2/3, 4-state convolutional code.

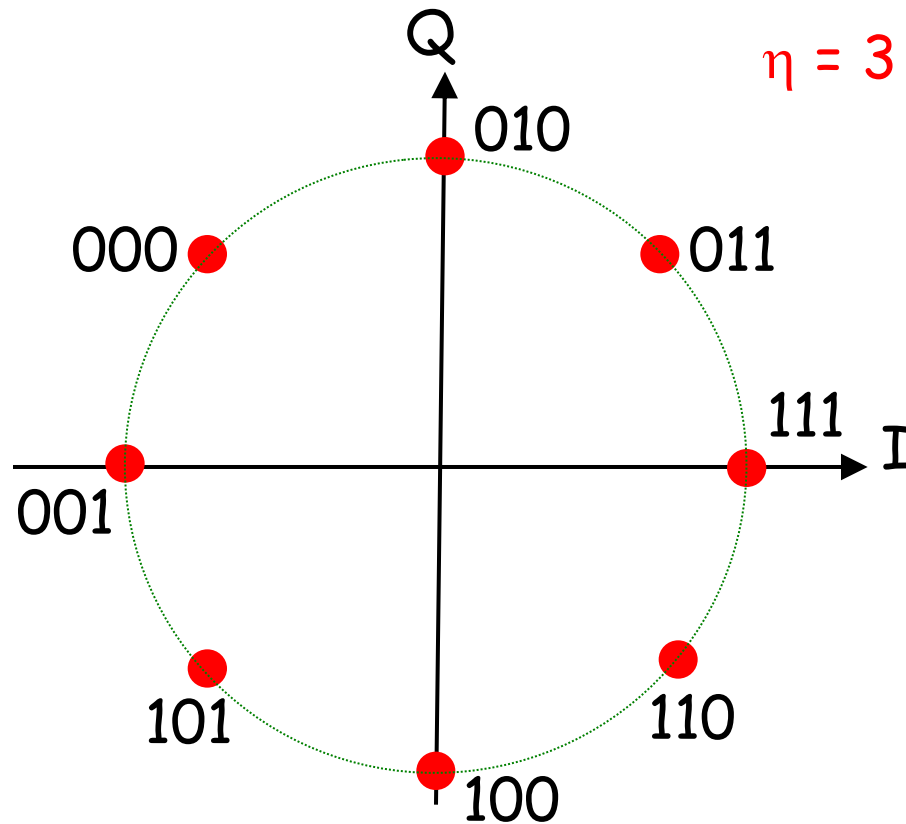
Will (2) outperform (1) in terms of  $P_{eb}$ ?

# Trellis-coded modulation - An example



QPSK (Gray mapping)

# Trellis-coded modulation - An example

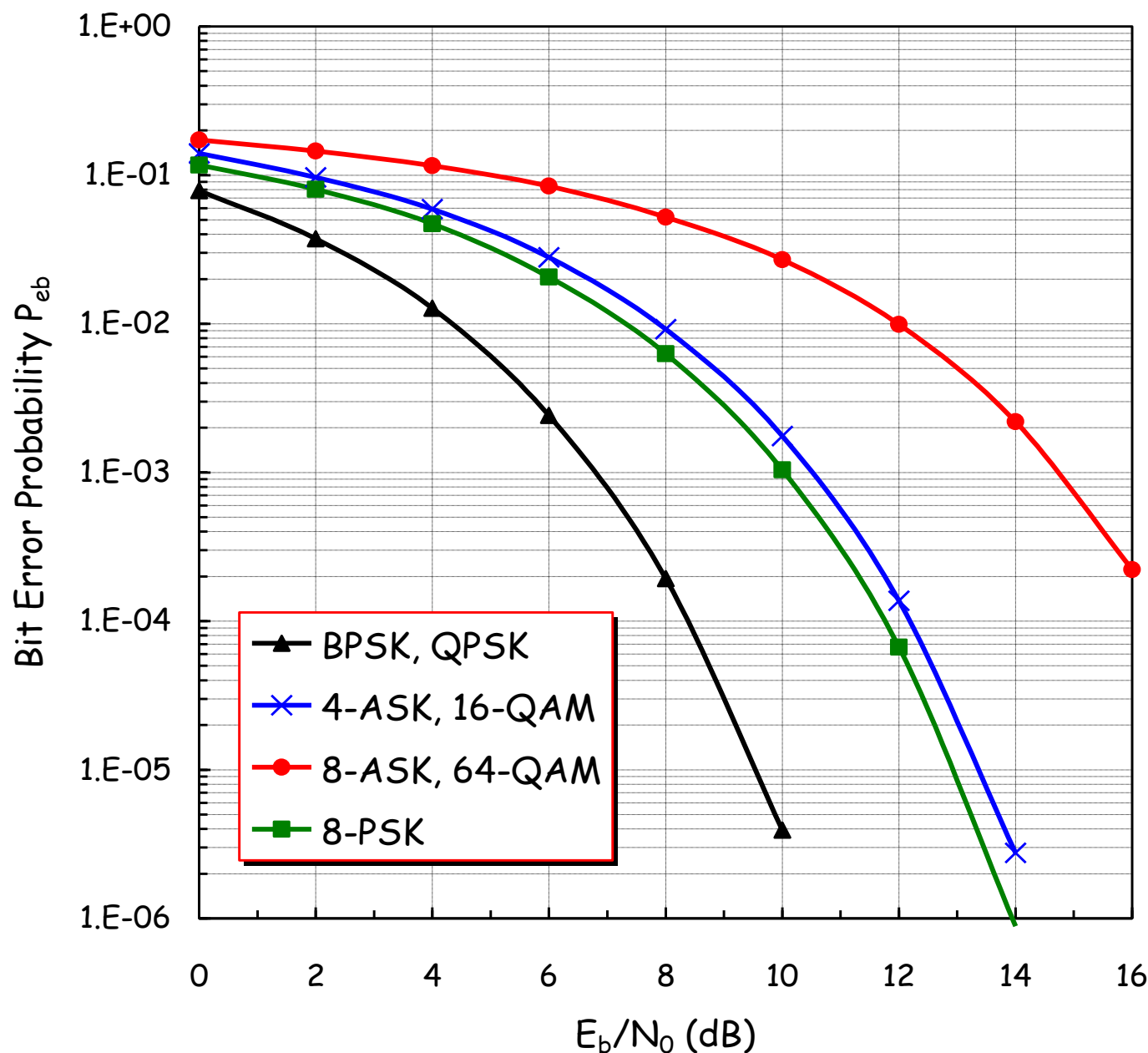


$$\eta = 3 \text{ bits/sec/Hz}$$

If the 3<sup>rd</sup> bit is used as a redundant bit, then the spectral efficiency is reduced to 2 bits/sec/Hz

8PSK (Gray mapping)

But, do not forget that QPSK is inherently more robust than 8PSK in the absence of coding...



# Trellis-coded modulation - An example

Symbol error probability of an uncoded modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N}{2} \cdot \text{erfc} \left( \sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

where:

- $N$  is the average number of nearest neighbour signal points in the constellation,
- $E_s/N_0$  is the SNR per transmitted signal,

# Trellis-coded modulation - An example

Symbol error probability of an uncoded modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N}{2} \cdot \text{erfc} \left( \sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

where:

- $d_0$  is the Euclidean distance between two nearest-neighbor signal points in the constellation, under the constraint of unit average energy for these signal points.



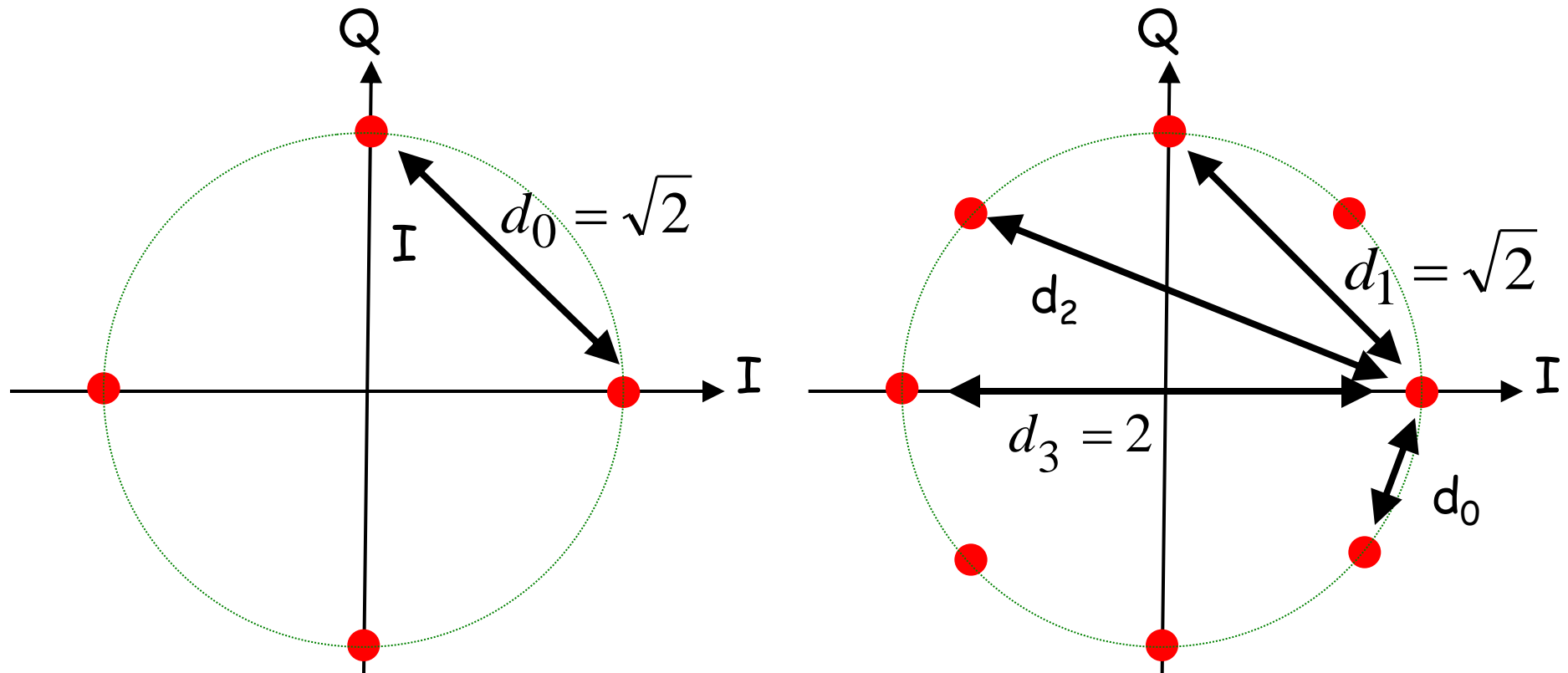
# Trellis-coded modulation - An example

Symbol error probability of an uncoded modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N}{2} \cdot \text{erfc} \left( \sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

If you want to know more about this (important) equation, just take the module entitled "Advanced Modulation and Coding Techniques" (EEE8003 for MSc students and EEE8104 for MEng students).

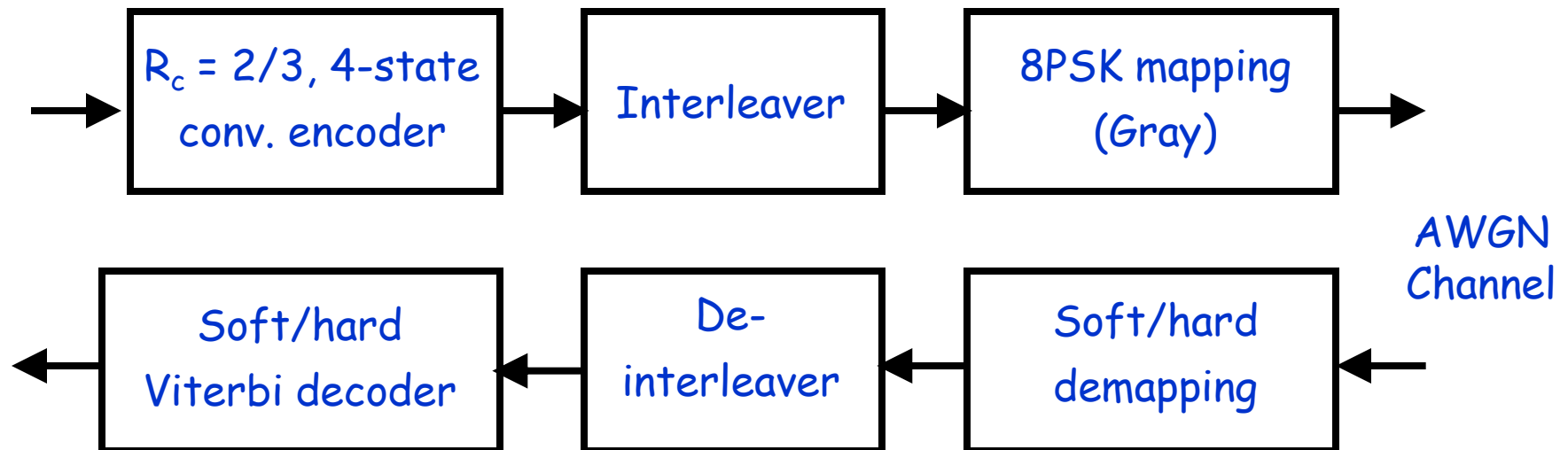
# Trellis-coded modulation - An example



$$d_0 = 2 \cdot \sin\left(\frac{\pi}{8}\right) \quad d_2 = 2 \cdot \cos\left(\frac{\pi}{8}\right)$$

# Trellis-coded modulation - An example

Before 1982: The coding and modulation were seen as two separate entities.

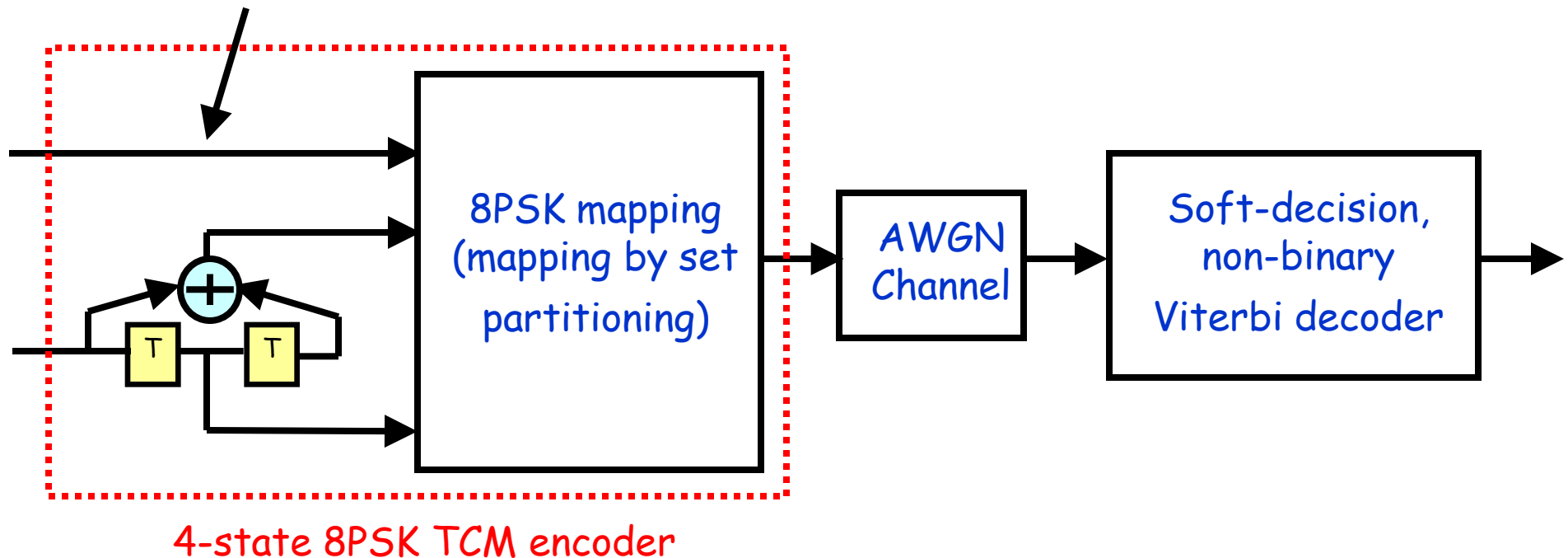


Coding and modulation are optimized separately. When soft-demapping is employed, such approach is known as "bit-interleaved coded modulation" (BICM).

# Trellis-coded modulation - An example

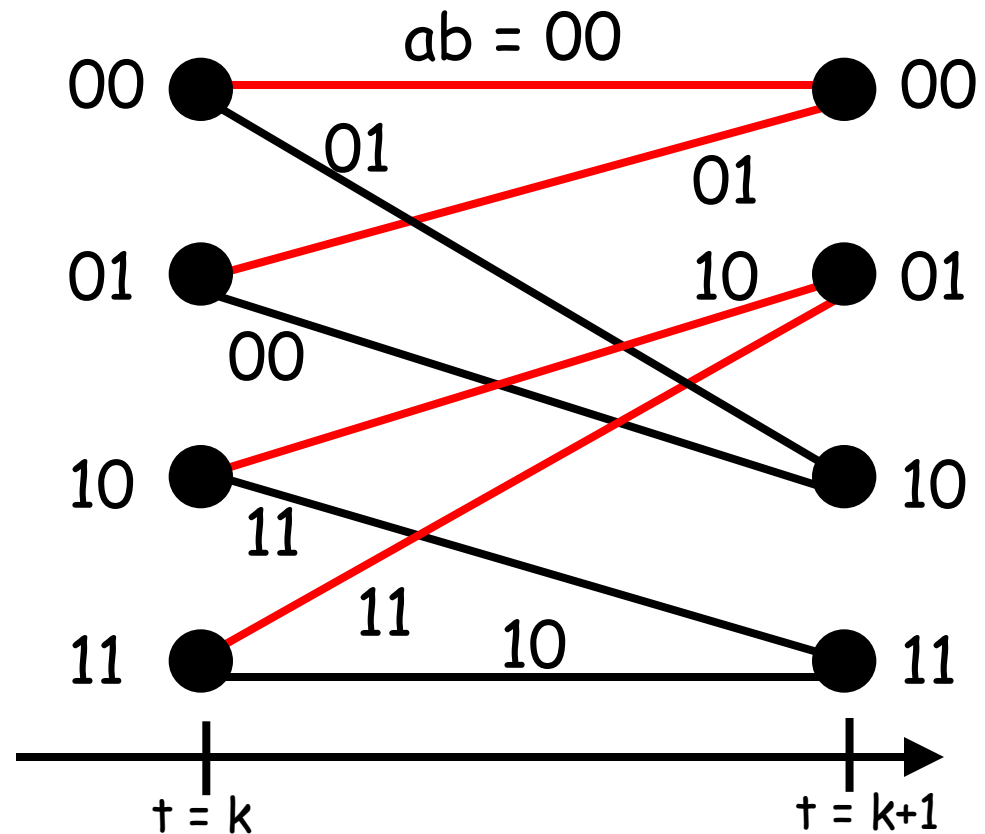
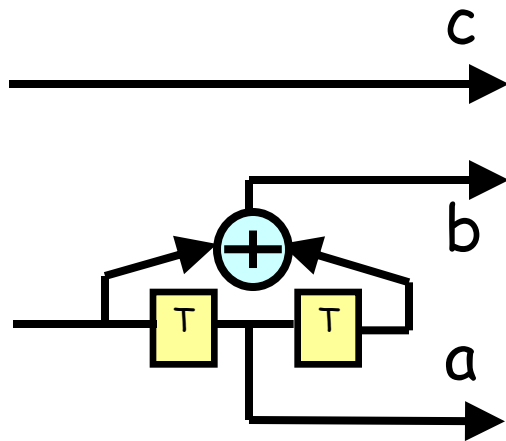
TCM: Coding and modulation are jointly optimized.

Some info bits are left uncoded



# Trellis-coded modulation - An example

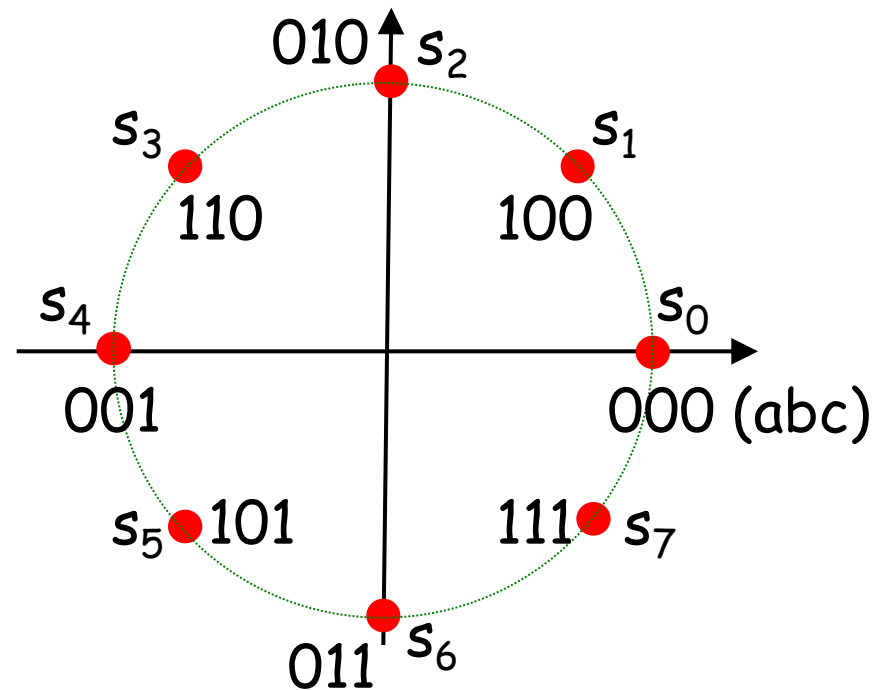
The encoder:



This binary encoder is not optimal ( $d_{\text{free}} = 3$  only).

# Trellis-coded modulation - An example

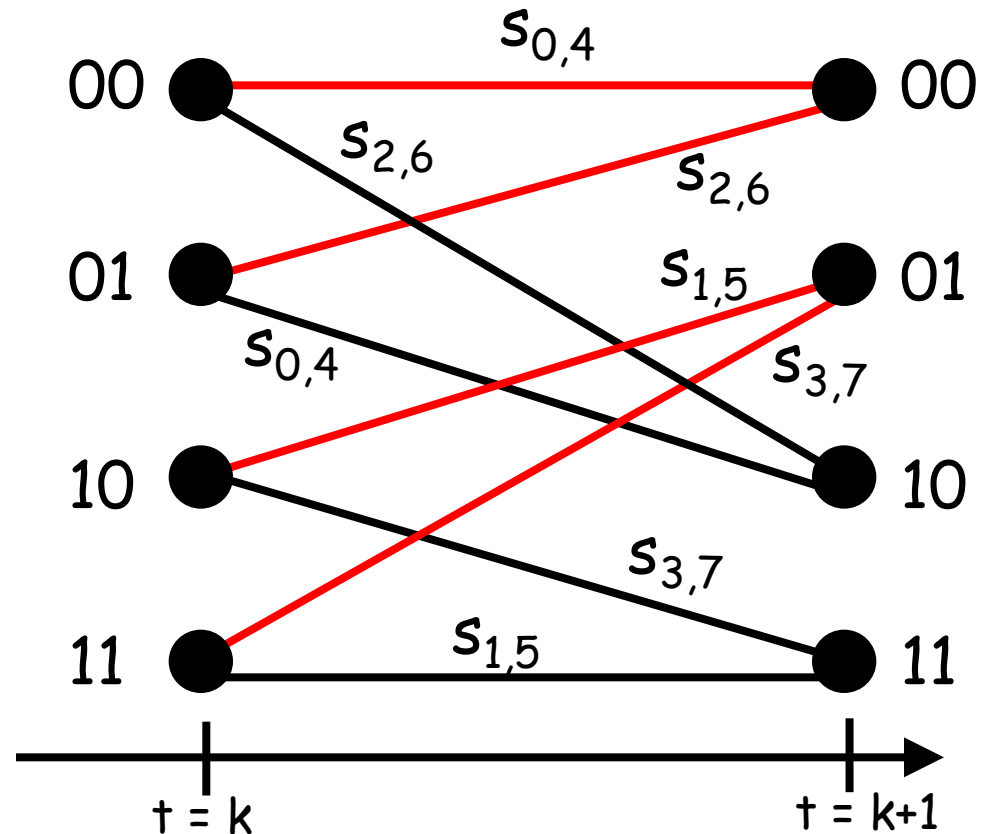
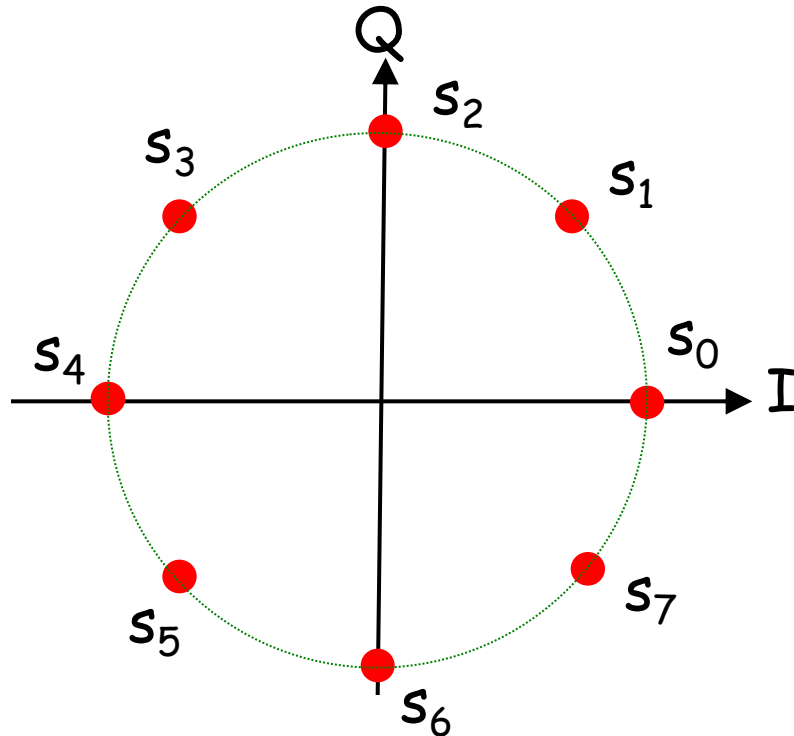
The mapping:



Not a Gray mapping !

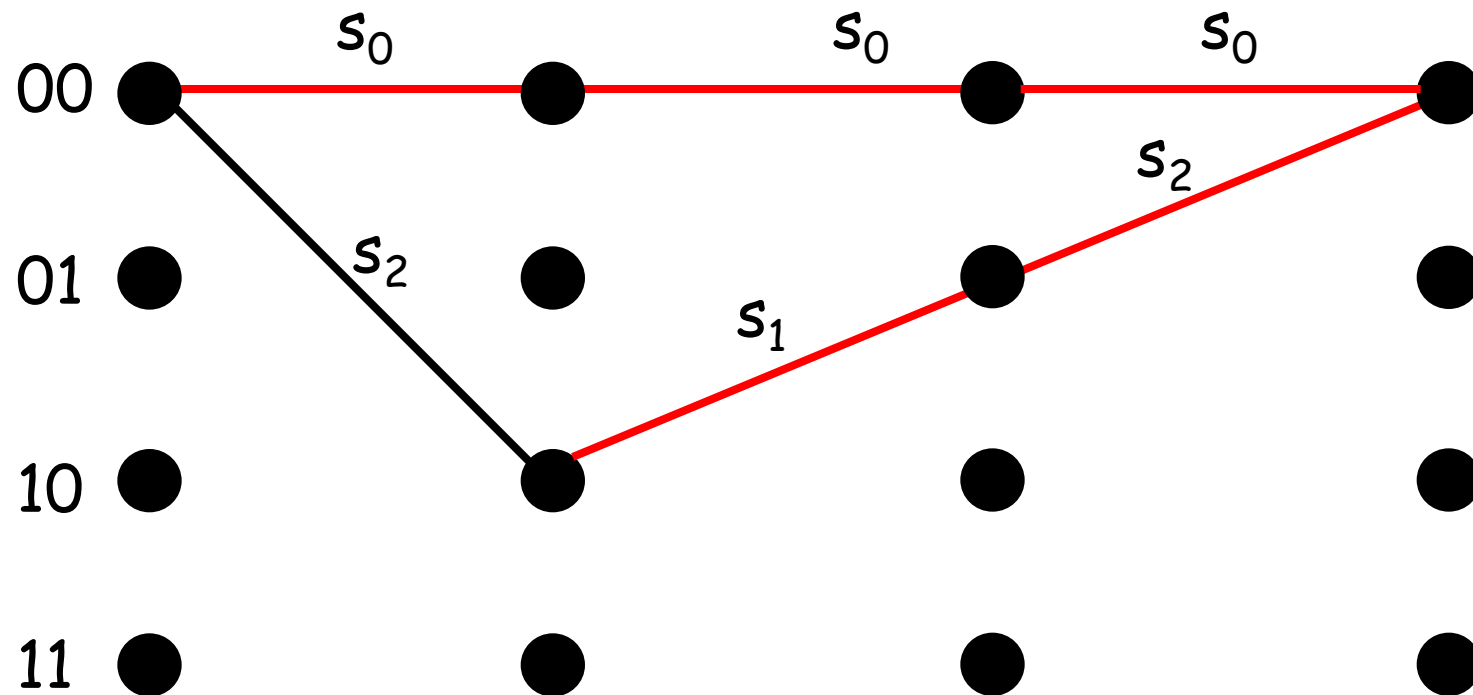
# Trellis-coded modulation - An example

Trellis of the TCM encoder:



# Trellis-coded modulation - An example

Free distance of the TCM encoder:

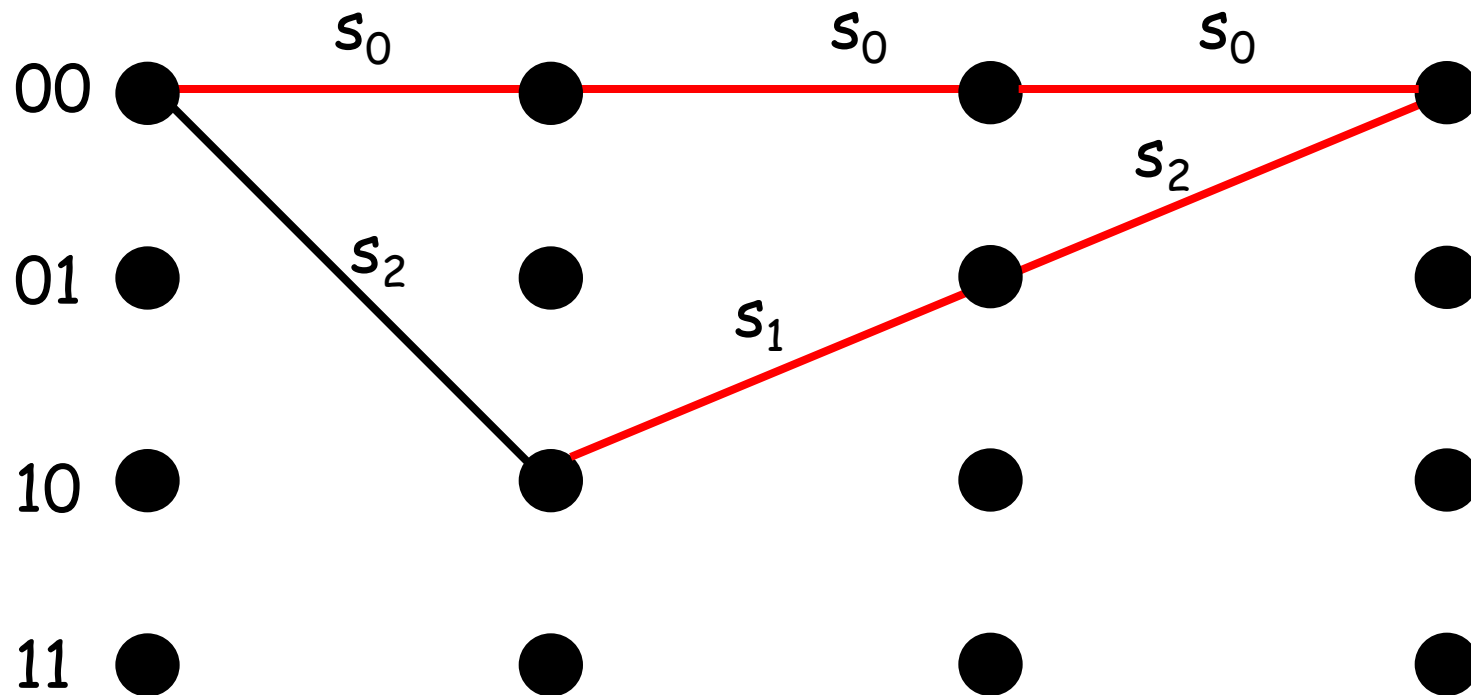


Computed in terms of squared Euclidean distance ( $\neq$  Hamming distance)



# Trellis-coded modulation - An example

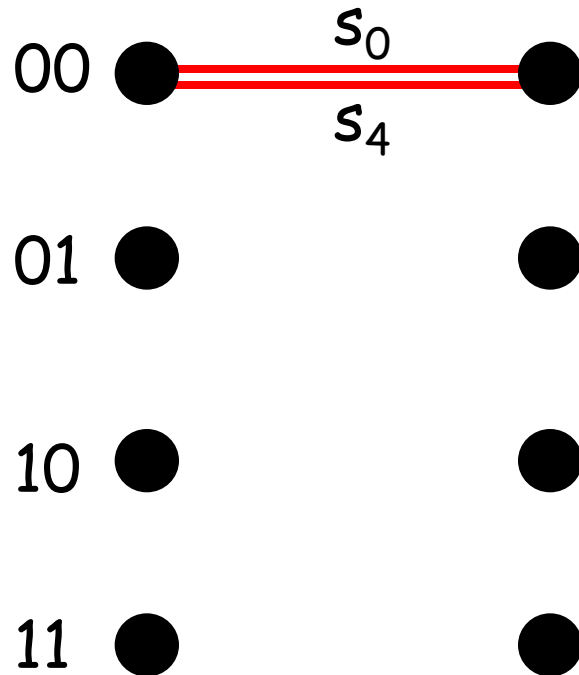
Free distance of the TCM encoder:



$$(d_{free,1})^2 = d_1^2 + d_0^2 + d_1^2 = 4 \cdot \left[ 1 + \sin^2\left(\frac{\pi}{8}\right) \right]$$

# Trellis-coded modulation - An example

Free distance of the TCM encoder:



Do not forget the parallel branches!

$$(d_{free,2})^2 = d_3^2 = 4$$

$$(d_{free})^2 = \text{Min} \{ (d_{free,1})^2, (d_{free,2})^2 \} = 4$$

# Trellis-coded modulation - An example

Symbol error probability of a TCM modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N_{free}}{2} \cdot \text{erfc} \left( \sqrt{\frac{d_{free}^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

where  $N_{free}$  is the average number of nearest neighbour signal sequences with distance  $d_{free}$  that diverge at any state from a transmitted signal sequence and remerge with it after one or more transitions ( $N_{free} = 1$  in our example).

# Trellis-coded modulation - An example

For uncoded QPSK:

$$P_{es} \approx \frac{N}{2} \cdot \text{erfc} \left( \sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

with  $N = 2$  and  $d_0 = \sqrt{2}$ .

Asymptotic coding gain of 4-state, 8PSK TCM over uncoded QPSK (in dB):

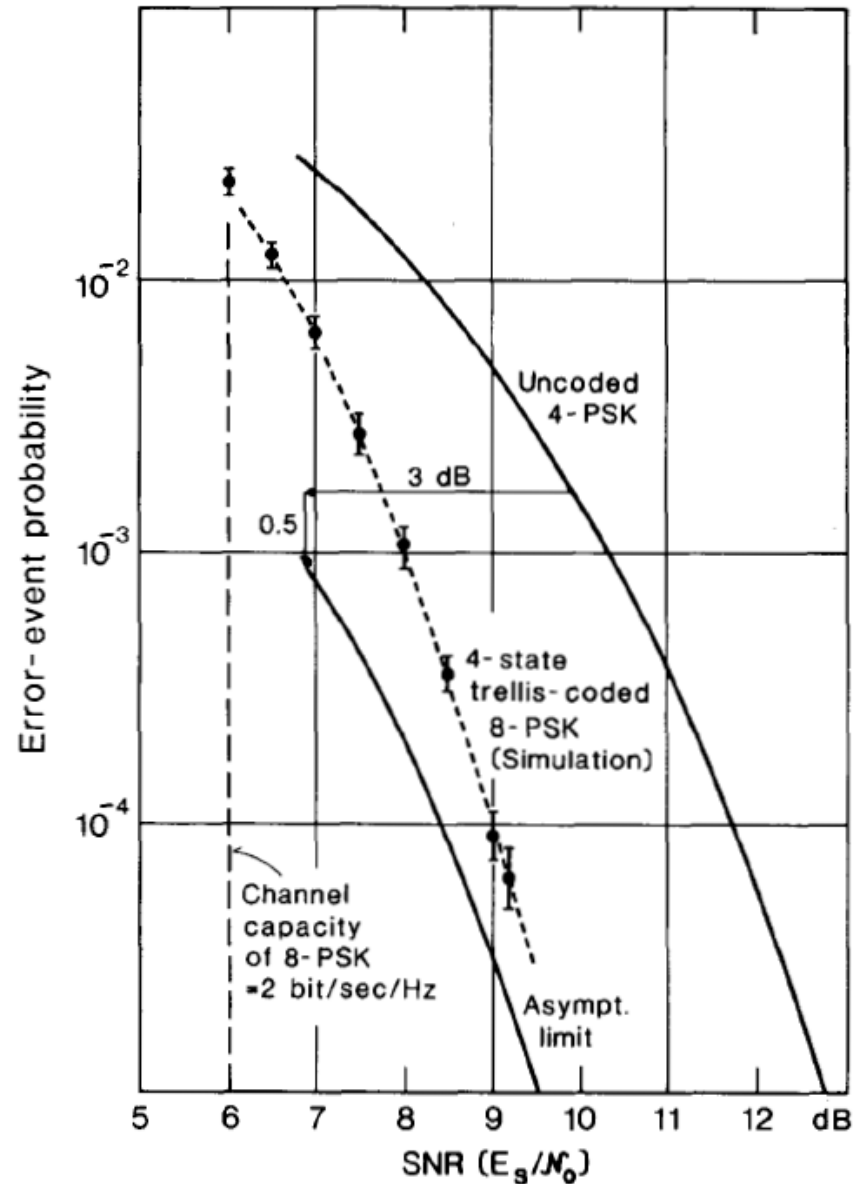
$$G \approx 10 \cdot \log_{10} \left[ \frac{d_{free}^2}{d_0^2} \right] = 10 \cdot \log_{10} \left[ \frac{4}{2} \right] = 3.01$$

# Trellis-coded modulation - An example

This 3-dB asymptotic gain is obtained at no cost in terms of spectral efficiency!

This is more than with any other coding approach (e.g., BICM).

But, TCM is only suitable for an AWGN channel... ☹



# Part 9

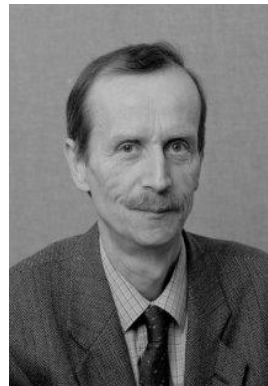
## Practical Error-Correcting Codes (1993-Today)

# The turbo revolution (1993)

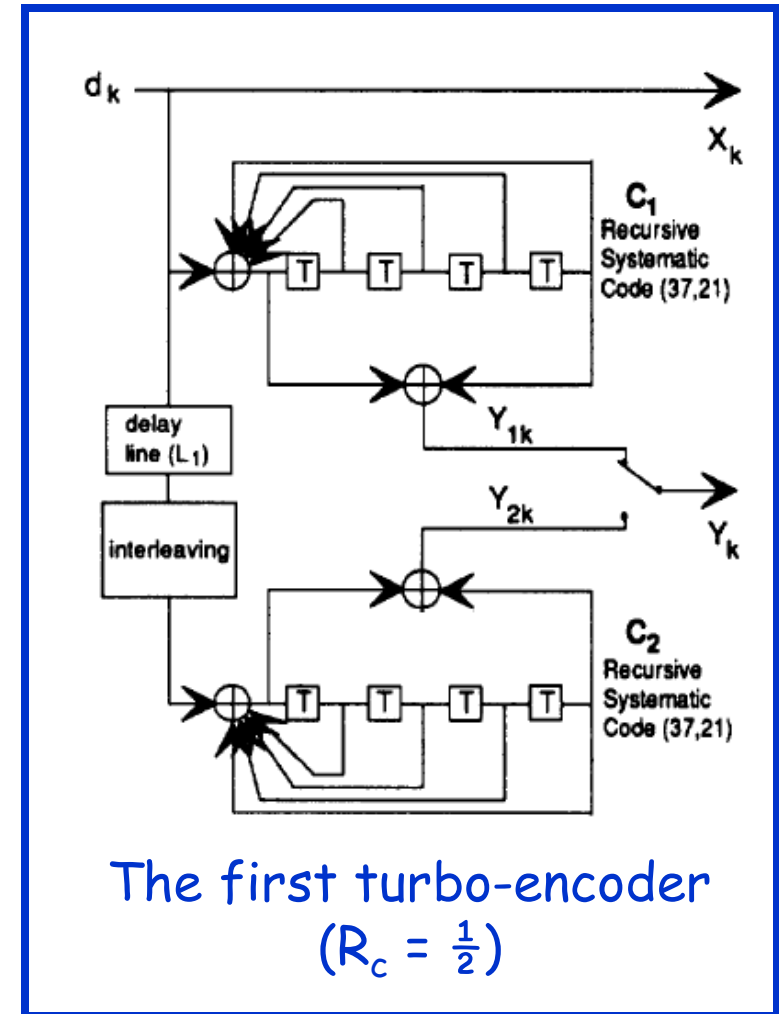
1993: Claude Berrou and Alain Glavieux introduce turbo codes\*, a class of codes that perform 0.5 dB away from the Shannon limit.



Claude Berrou



Alain Glavieux  
(1949 - 2004)



The first turbo-encoder  
( $R_c = \frac{1}{2}$ )

\* With the help of Punya Thitimajshima (1955-2006)

# Turbo codes

Prof. Claude Berrou served as an external PhD examiner at our School.

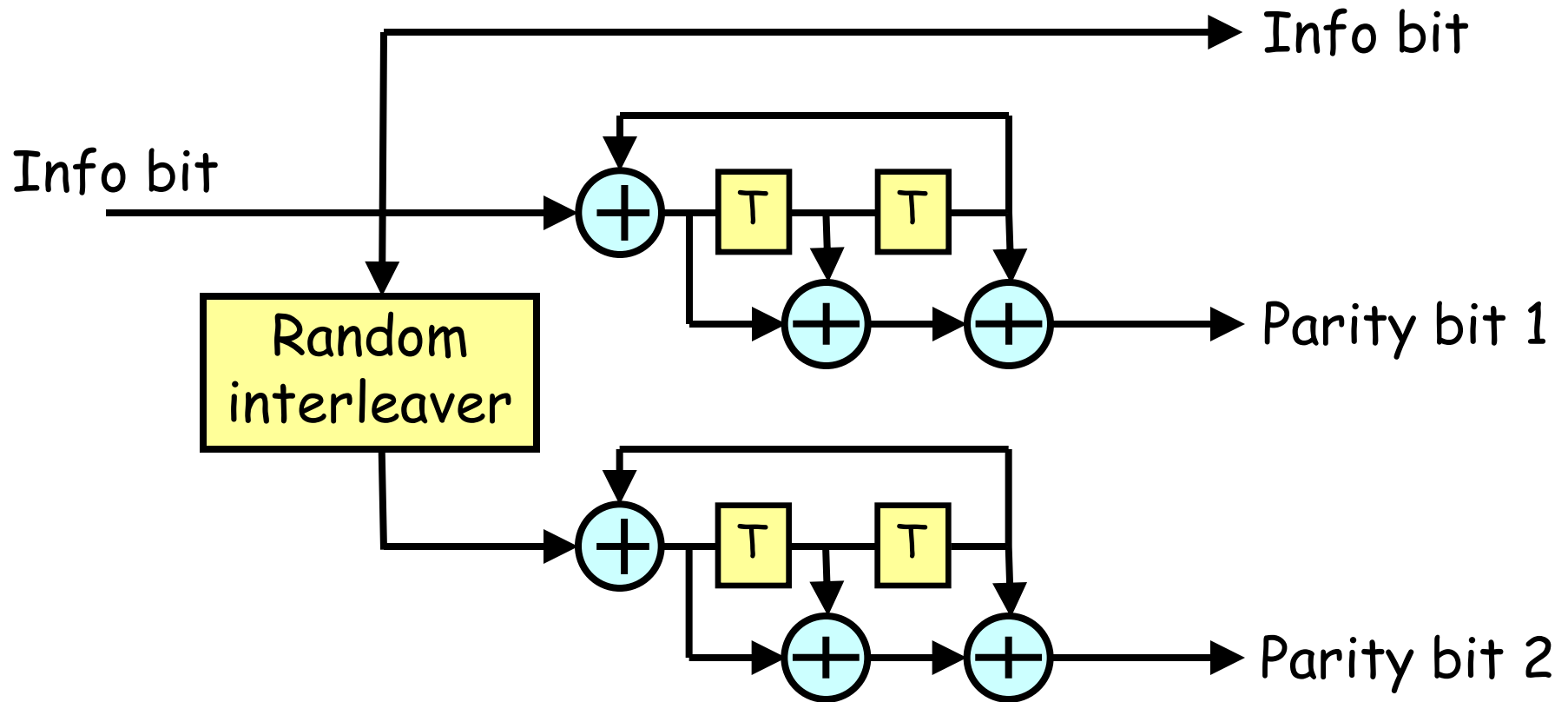


Photo taken in December 2007, a few minutes after the PhD viva of Dr. Boon Kien Khoo.



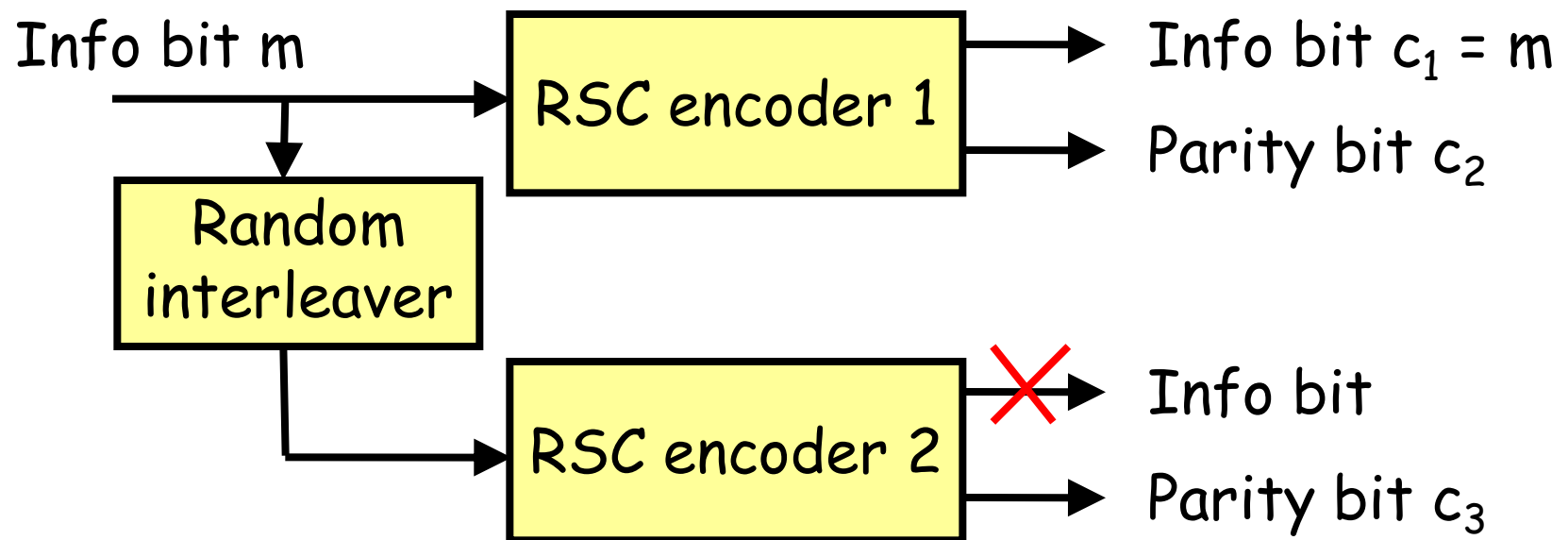
# Turbo codes

Turbo codes are parallel-concatenated recursive and systematic convolutional (RSC) codes.



# Turbo codes

The basic rate of a turbo-code is  $R_c = 1/3$ , but any rate can be obtained by puncturing parity bits.



Turbo codes are random codes thanks to the presence of the random interleaver.

# Turbo codes

Example of a 8-bit interleaver, i.e. an interleaver operating on blocks of 8 bits

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$

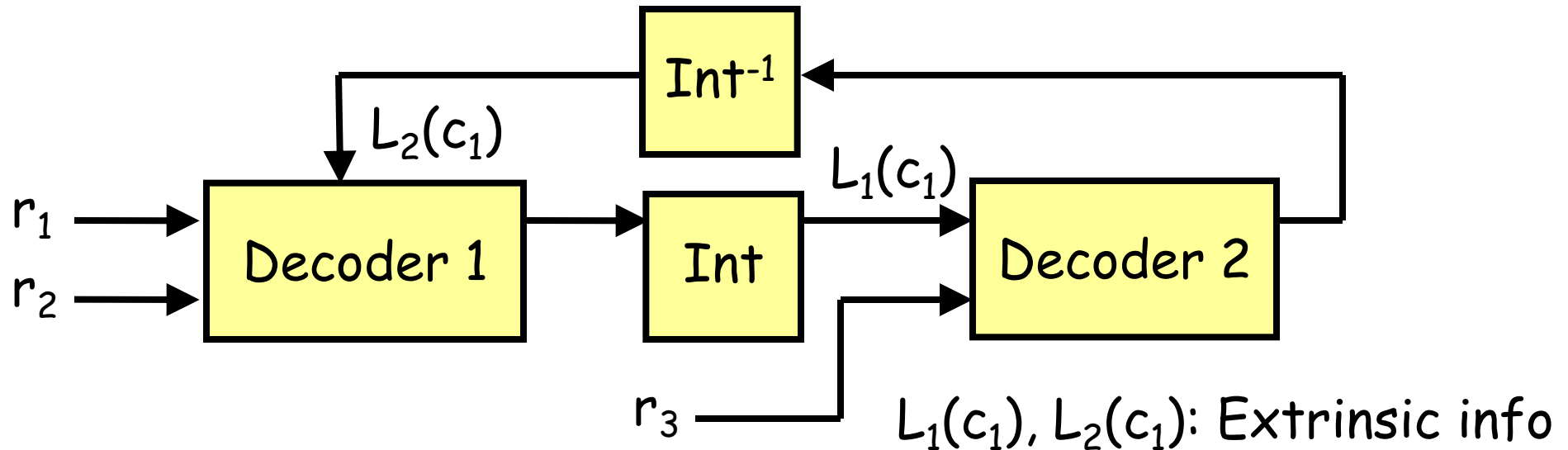


$(a_4, a_6, a_0, a_3, a_7, a_1, a_5, a_2)$

For a random interleaver, the interleaving of the input block is done randomly, or at least in a way that looks random.

# Turbo codes

ML decoding is too complex to implement →  
Iterative decoding algorithm instead.



Iterative decoding displays performance very close  
to that of ML decoding!

# Turbo codes

Both decoders must be able to produce soft decisions => Soft-input, soft-output (SISO) decoders.

The Viterbi algorithm cannot be used. Instead, we have the choice between the following algorithms:

1. Soft-output Viterbi algorithm (SOVA, 1989);
2. Maximum a posteriori probability (MAP) algorithm (1974, optimal but complex to implement);
3. Log-MAP (a simplification of MAP);
4. Max-log-MAP (a simplification of log-MAP).

# Turbo codes

The iterative decoding algorithm is almost as good as ML decoding.

Nowadays, iterative algorithms are commonly applied in any receiver design.

The “message passing”/“belief propagation” algorithm, according to Claude Berrou:

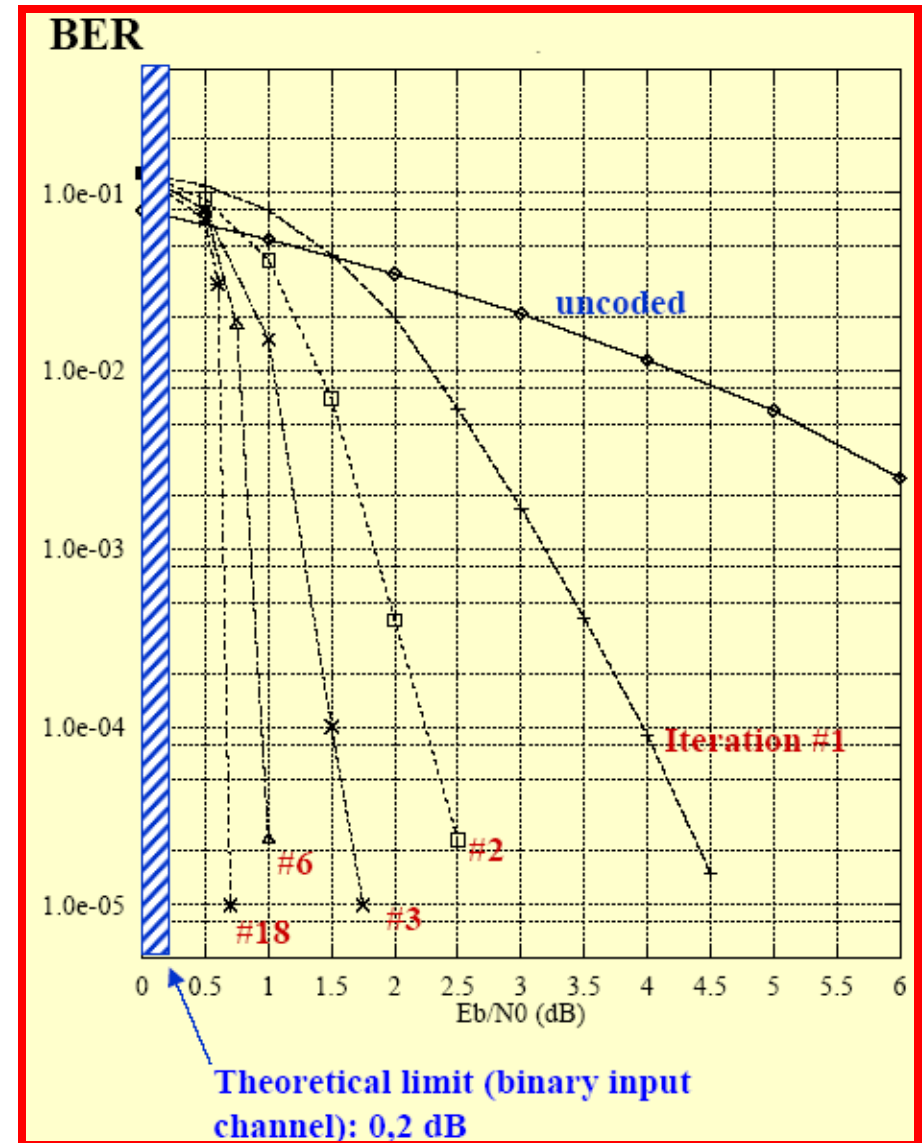
“When several probabilistic machines work together on the estimation of a common set of symbols, all the machines must ultimately give the same decision about each symbol, as a single (global) decoder would.”

# Turbo codes

Performance of a rate-1/2 turbo code over BPSK, AWGN channel.

Error-free communications is achieved for  $E_b/N_0 = 0.7$  dB.

→ Performs only 0.5 dB away from the Shannon limit! 😊

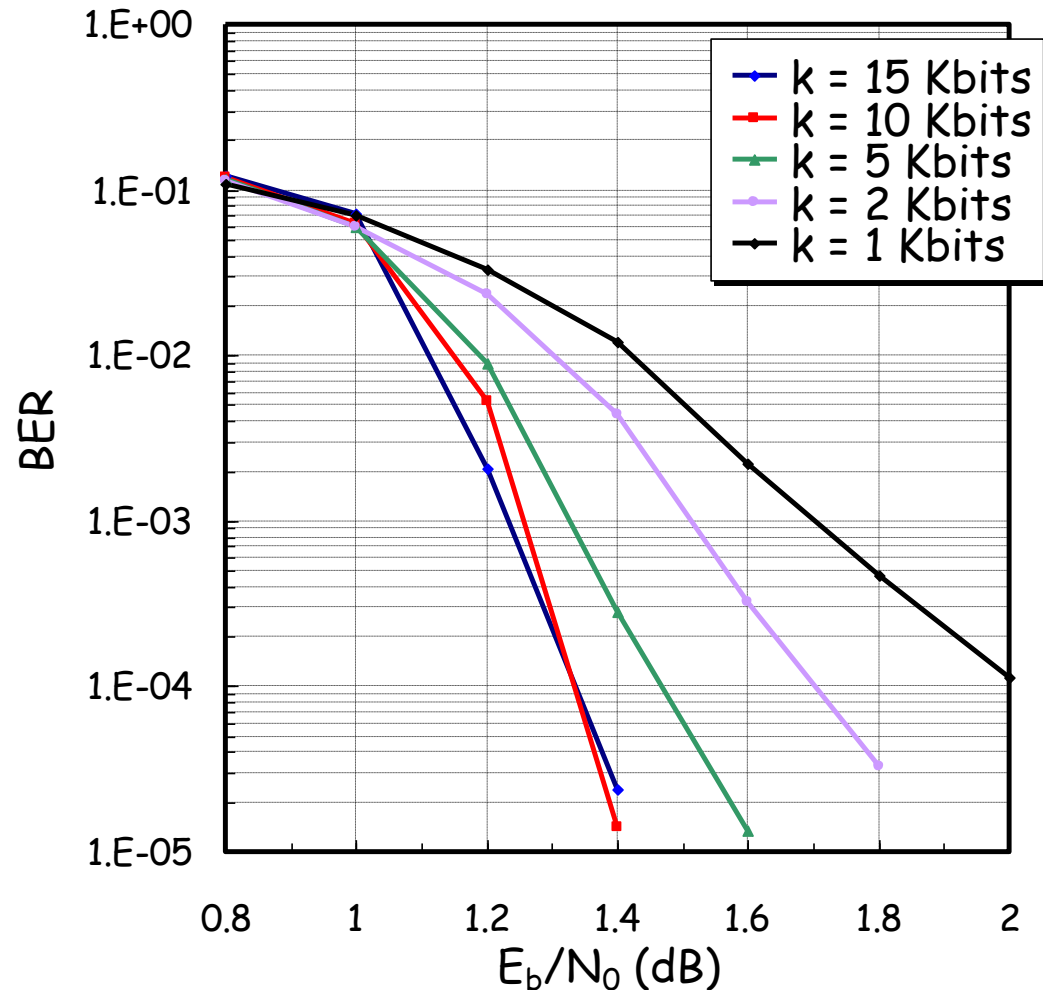


# Turbo codes

$R_c = 1/2$ , BPSK, AWGN channel, random interleaving,  
10 iterations, max-log-MAP decoding

BER vs. SNR curves  
for a turbo code  
obtained by parallel  
concatenation of  
two rate-2/3 (23,  
31) RSC codes.

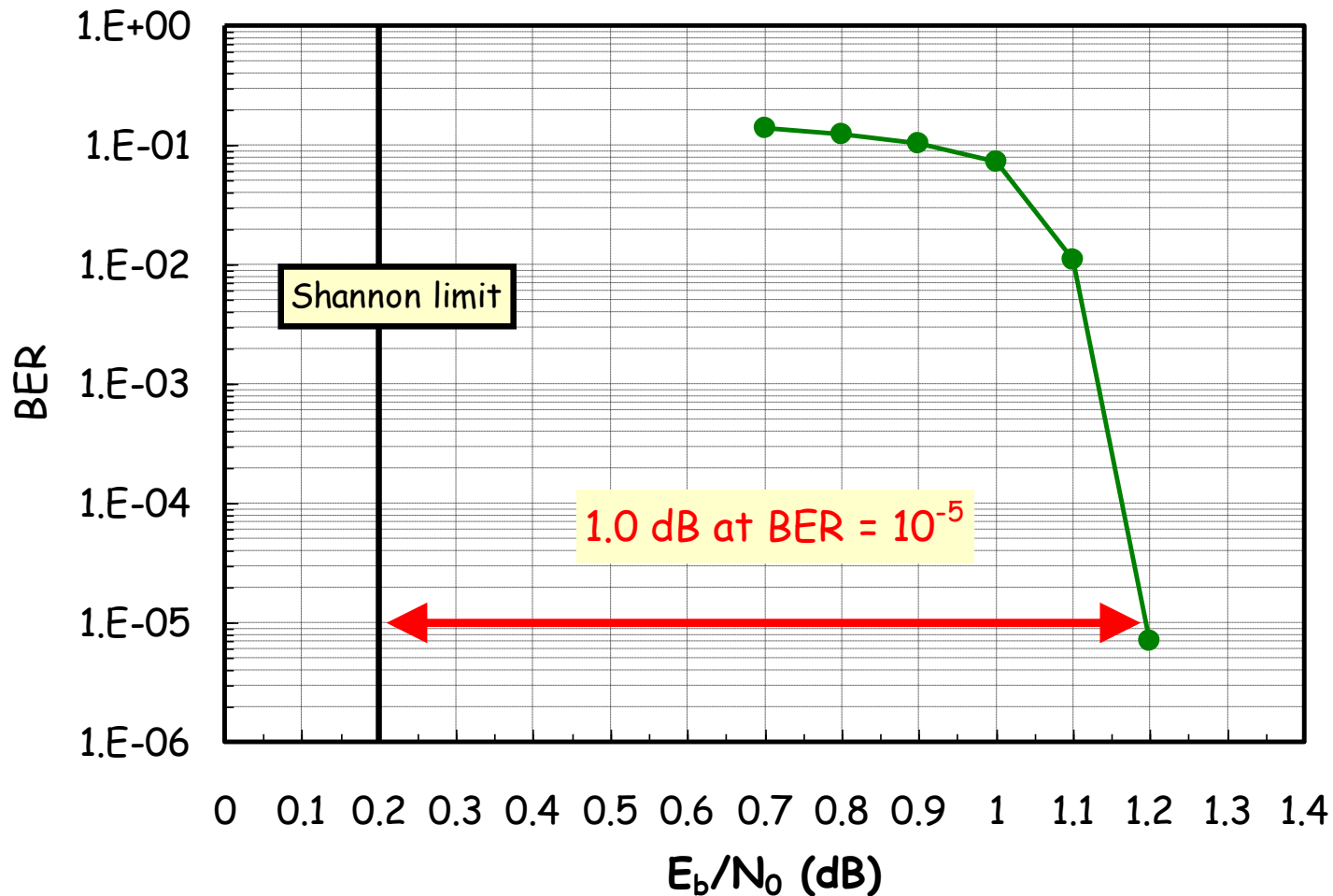
$k$  is the length of  
the message  $M$ .





# Turbo codes

Performance of a rate-1/2 turbo code (Max-log-MAP decoding, 10 iterations,  $k = 50$  Kbits, AWGN, BPSK channel)



# Turbo codes

How to explain the astonishing performance of turbo codes?

Remember the union bound equation for BPSK, AWGN channel (valid under ML decoding assumption):

$$P_{eb} \leq \sum_{d=d_{\min}}^{+\infty} e(d) \cdot \operatorname{erfc} \left( \sqrt{d R_c \frac{E_b}{N_0}} \right)$$

Turbo codes do not necessarily have a large minimum Hamming distance between codewords, but their distance spectrum ( $e(d)$  vs.  $d$ ) is very good!

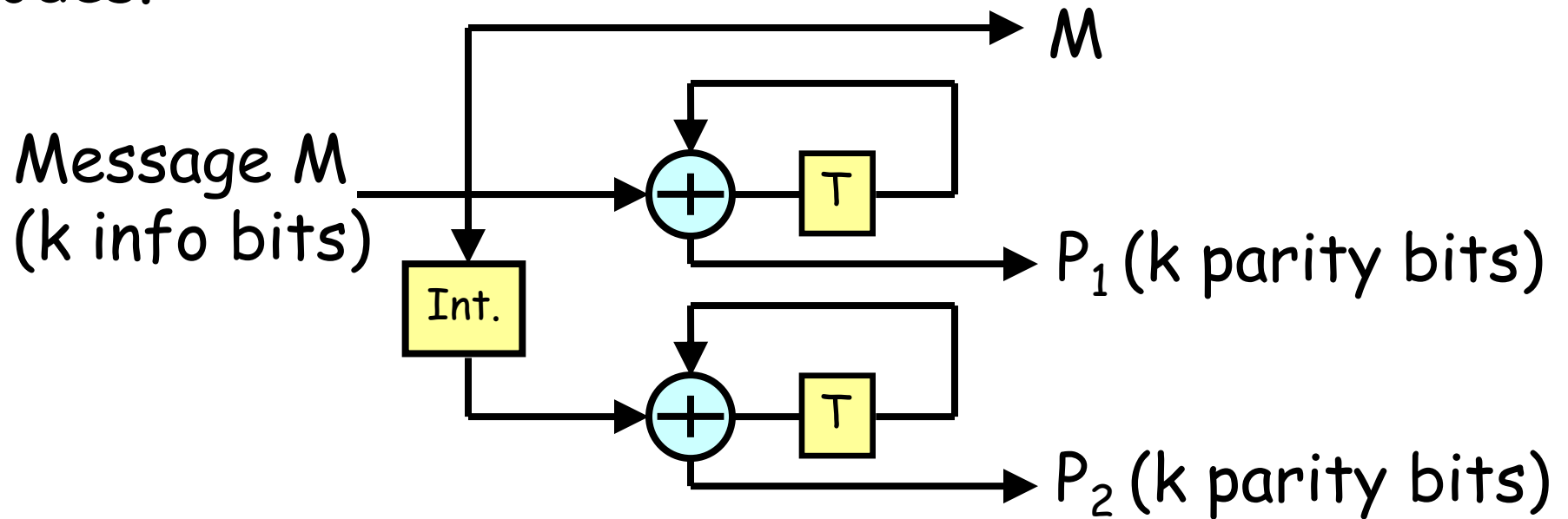
# Turbo codes

$$P_{eb} \leq e(d_{\min}) \cdot \operatorname{erfc} \left( \sqrt{d_{\min} R_c \frac{E_b}{N_0}} \right) + e(d_{\min} + 1) \cdot \operatorname{erfc} \left( \sqrt{(d_{\min} + 1) R_c \frac{E_b}{N_0}} \right) \\ + e(d_{\min} + 2) \cdot \operatorname{erfc} \left( \sqrt{(d_{\min} + 2) R_c \frac{E_b}{N_0}} \right) + \dots$$

What matters at low SNR is not necessarily the minimum Hamming distance  $d_{\min}$ , but the values of  $e(d)$  (especially those associated with the first few terms in this sum).

# Turbo codes - A simple example

Let us study a simple turbo code designed by parallel concatenation of two rate-1/2, (2, 3) RSC codes.



The codeword  $C \equiv M + P_1 + P_2$  is composed of  $k$  info bits and  $2k$  parity bits  $\Rightarrow R_c = 1/3$ .

# Turbo codes - A simple example

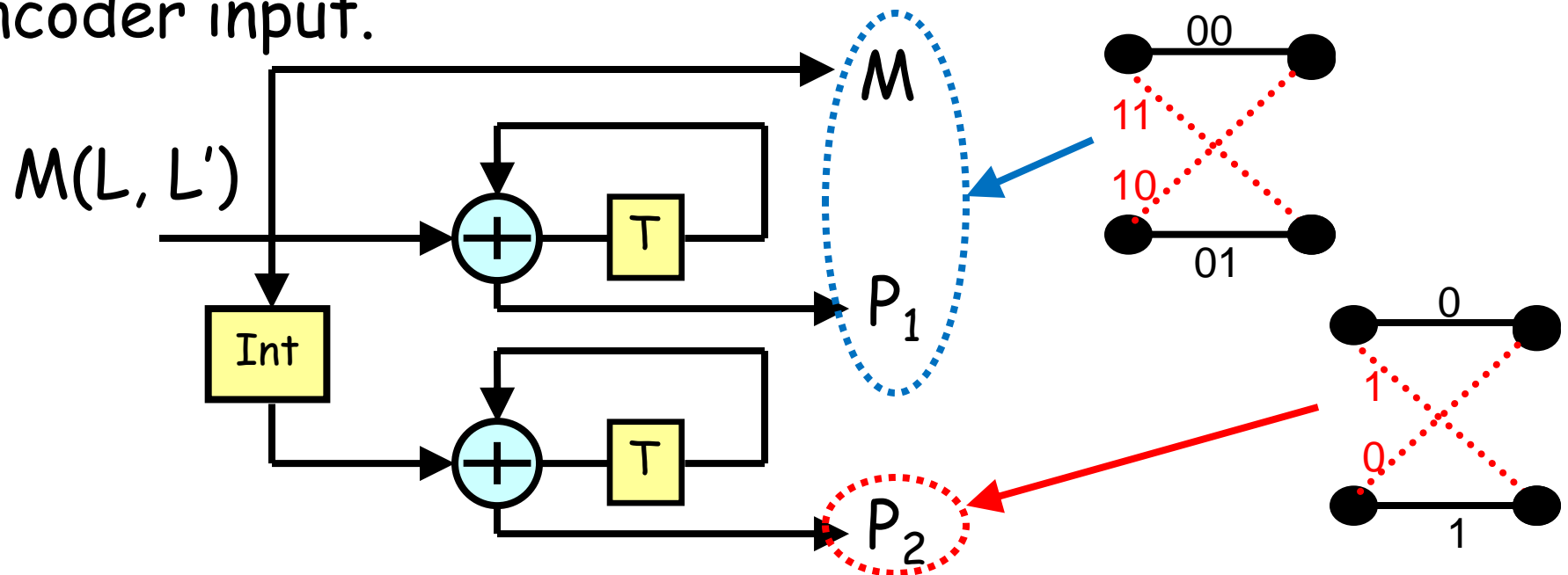
Let us determine the first five terms in the union bound for this turbo code.

Turbo codes are linear codes. Therefore, we can take the all-zero codeword as a reference and study the Hamming weights of the codewords associated with small-weight messages.

We focus on what happens when the turbo encoder input is a weight-2 or weight-4 message (we already know that messages with weights equal to 1, 3, 5 and so on cannot exist)

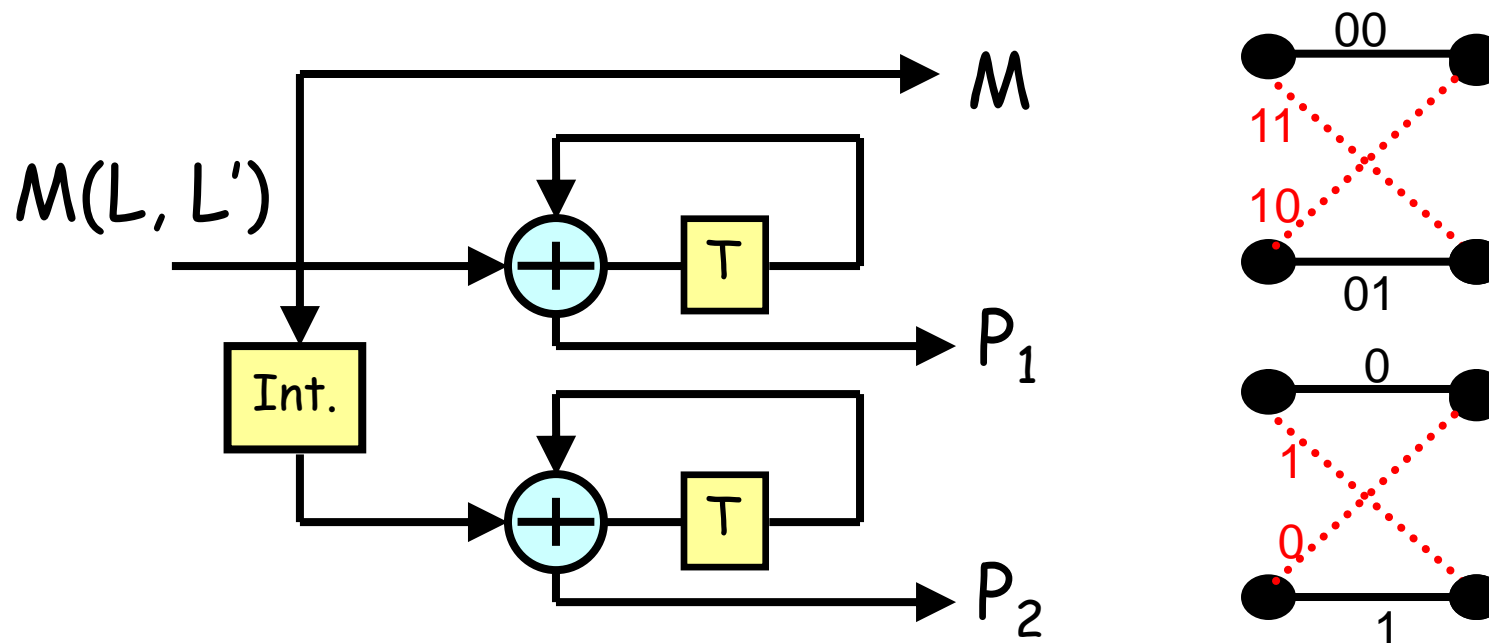
# Turbo codes - A simple example

Consider a weight-2 message  $M(L, L')$  at the turbo encoder input.



$M(L, L')$  is a message in which both 1s are separated by  $L$  0s before interleaving and separated by  $L'$  0s after interleaving.

# Turbo codes - A simple example



The weight of the codeword  $(M + P_1)$  produced by the first RSC encoder is  $w_H(L) = 3 + L$ , with  $L$  ranging from 0 to  $k-2$ .

# Turbo codes - A simple example

Ex:  $M = (...010010...)$   $\Rightarrow C = (...001101011000...)$ , i.e.  $M = (...010010...)$  and  $P_1 = (...011100...)$ .

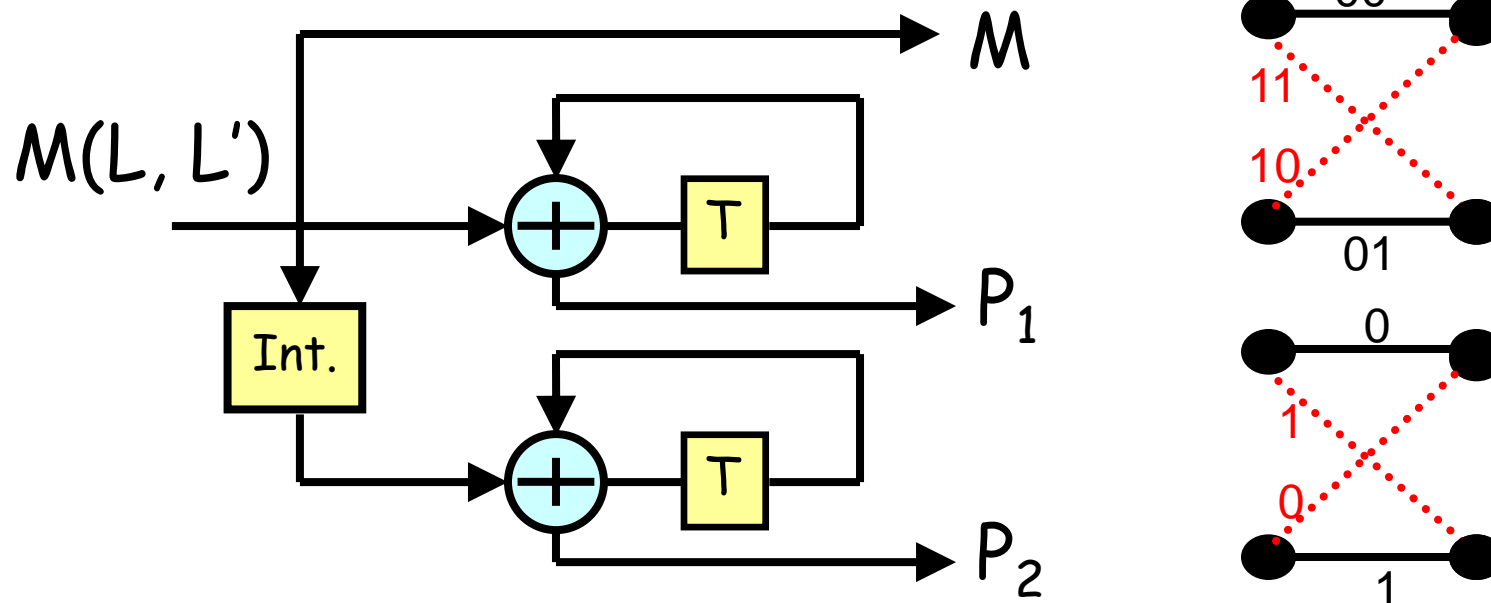
We recall that, with RSC encoders, weight-2 messages can often generate codewords at large distance from the all-zero codeword.

This is due to the "recursivity" of the encoder.

A NRC code would not exhibit such characteristic as, with such an encoder, a small-weight message always generates a small-weight codeword ☹



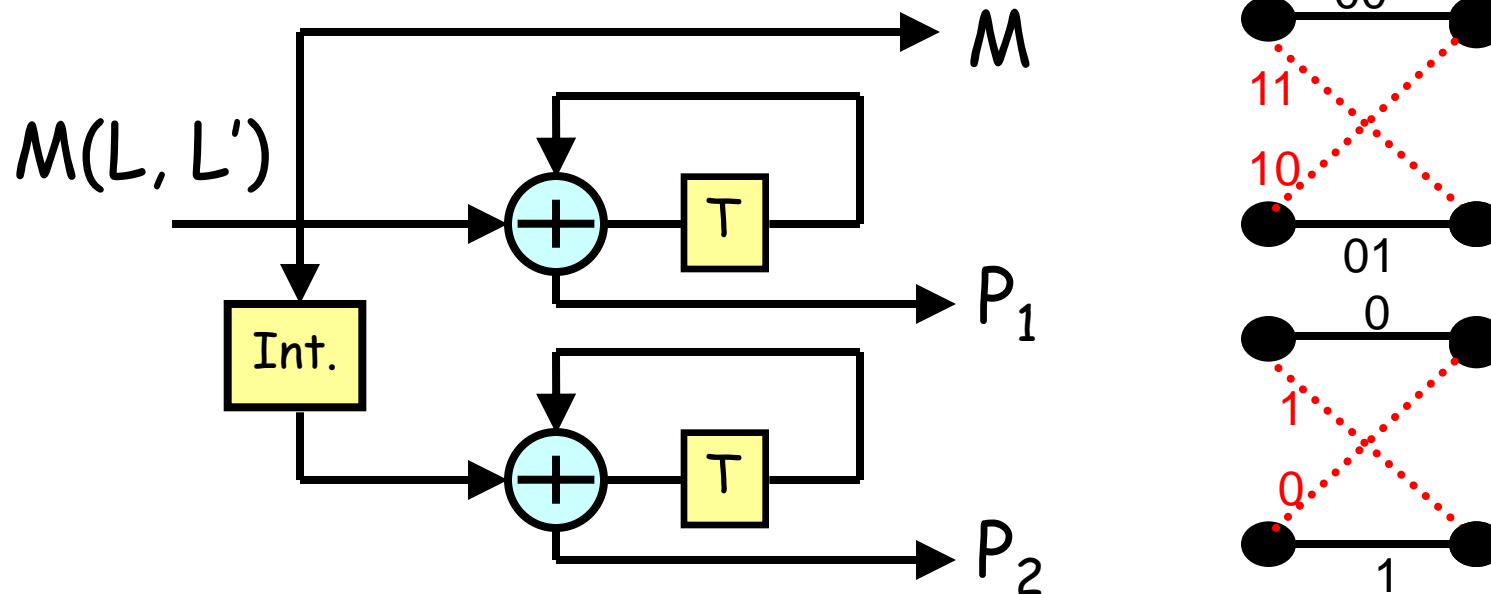
# Turbo codes - A simple example



The weight of the parity sequence  $P_2$  is  $w_H(L') = 1 + L'$ , with  $L'$  ranging from 0 to  $k-2$ .

Ex:  $M = (...01000010...) \Rightarrow P_2 = (...01111100...)$

# Turbo codes - A simple example



The weight of the codeword  $(M + P_1 + P_2)$  is finally  $w_H(L, L') = 3 + L + 1 + L' = 4 + L + L'$ , with  $L$  and  $L'$  ranging from 0 to  $k-2$ .

# Turbo codes - A simple example

But, how many messages  $M(L, L')$  are there?

When studying the RSC codes alone, we saw that there are  $(k-L-1)$  different messages for which both 1s are separated by  $L$  0s.

For each of these messages, what is the probability that after random interleaving both 1s are separated by  $L'$  0s?

The total number of possible permutations of 2 bits in a word of  $k$  bits is given by the binomial coefficient

# Turbo codes - A simple example

$$\binom{k}{2} = \frac{k \cdot (k-1)}{2} \approx \frac{k^2}{2}$$

The number of configurations for which two 1s are separated by  $L'$  0s is  $(k-L'-1)$ .

Therefore, the probability to have one of those configurations after interleaving is thus given by

$$\Pr\{L'\} = \frac{(k-L'-1)}{\binom{k}{2}} \approx \frac{2 \cdot (k-L'-1)}{k^2}$$

# Turbo codes - A simple example

The average number of messages  $M(L, L')$  is thus given by

$$\Pr\{L'\} \cdot (k - L - 1) \approx \frac{2 \cdot (k - L' - 1) \cdot (k - L - 1)}{k^2}$$

As long as  $L \ll k$  and  $L' \ll k$ , the number of messages is given by

$$\Pr\{L'\} \cdot (k - L - 1) \approx \frac{2k^2}{k^2} = 2$$

A weight-4 codeword is obtained only when the message is such that  $L = L' = 0$ .

# Turbo codes - A simple example

The contribution of this message to the union bound is thus represented by the error coefficient:

$$e(d = 4) \approx \frac{2 \times 2}{2k} = \frac{2}{k}$$

A weight-5 codeword is obtained when the message is such that  $L + L' = 1$ , i.e. ( $L = 0$  and  $L' = 1$ ) or ( $L = 1$  and  $L' = 0$ ). The contribution of these messages to the union bound is thus represented by the error coefficient:

$$e(d = 4) \approx \frac{2}{k} + \frac{2}{k} = \frac{4}{k}$$

# Turbo codes - A simple example

A weight-6 codeword is obtained when the message is such that  $L + L' = 2$ , i.e. ( $L = 0$  and  $L' = 2$ ) or ( $L = 1$  and  $L' = 1$ ) or ( $L = 2$  and  $L' = 0$ ).

The contribution of these three weight-2 messages to the union bound is thus represented by the error coefficient:

$$e(d=6) \approx \frac{2}{k} + \frac{2}{k} + \frac{2}{k} = \frac{6}{k}$$

# Turbo codes - A simple example

A weight-7 codeword is obtained when the message is such that  $L + L' = 3$ , i.e. ( $L = 0$  and  $L' = 3$ ) or ( $L = 1$  and  $L' = 2$ ) or ( $L = 2$  and  $L' = 1$ ) or ( $L = 3$  and  $L' = 0$ ).

The contribution of these four weight-2 messages to the union bound is thus represented by the error coefficient:

$$e(d=7) \approx \frac{2}{k} + \frac{2}{k} + \frac{2}{k} + \frac{2}{k} = \frac{8}{k}$$



# Turbo codes - A simple example

A weight-8 codeword is obtained when the message is such that  $L + L' = 4$ , i.e. ( $L = 0$  and  $L' = 4$ ) or ( $L = 1$  and  $L' = 3$ ) or ( $L = 2$  and  $L' = 2$ ) or ( $L = 3$  and  $L' = 1$ ) or ( $L = 4$  and  $L' = 0$ ).

The contribution of these five weight-2 messages to the union bound is thus represented by the error coefficient:

$$e(d = 7) \approx \frac{2}{k} + \frac{2}{k} + \frac{2}{k} + \frac{2}{k} + \frac{2}{k} = \frac{10}{k}$$

# Turbo codes - A simple example

These results show that:

- (1) The minimal distance of the turbo code is only equal to 4, and is thus hardly larger than that of a constituent RSC code. That is not very impressive. ☹️
- (2) The first 5 error coefficients are (so far) much smaller than those of a constituent RSC code, thanks to the use of the random interleaver. 😊

This phenomenon is called “spectral thinning”.

# Turbo codes - A simple example

(3) As  $k$  is increased, the error coefficient values become even smaller. 😊

This is often referred to as the “interleaver gain”.

We also need to have a close look at the weight-4 messages to make sure that there is no bad surprise around the corner...

# Turbo codes - A simple example

Consider a weight-4 message where the first two 1s are separated by  $L$  0s before interleaving and separated by  $L'$  0s after interleaving, whereas the last two 1s are separated by  $L''$  0s before interleaving and separated by  $L'''$  0s after interleaving.

Such message is denoted as  $M(L, L', L'', L''')$ .

The weight of the codeword  $(M + P_1)$  produced by the first RSC encoder is  $w_H(L, L'') = 6 + L + L''$ , with  $L$  and  $L''$  ranging from 0 to  $k-4$ .

# Turbo codes - A simple example

The weight of the parity sequence  $P_2$  is  $w_H(L', L''') = 2 + L' + L'''$ , with  $L'$  and  $L'''$  ranging from 0 to  $k-4$ .

The weight of the codeword  $(M + P_1 + P_2)$  produced by the turbo encoder is therefore  $w_H(L, L', L'', L''') = 8 + L + L' + L'' + L'''$ .

But, how many messages  $M(L, L', L'', L''')$  are there?

When studying the RSC codes alone, we saw that the number of different messages for which the first two 1s are separated by  $L$  0s and the last two 1s are separated by  $L''$  0s is given by

# Turbo codes - A simple example

$$\binom{k-L-L''-2}{2} = \frac{(k-L-L''-2) \cdot (k-L-L''-3)}{2} \approx \frac{k^2}{2}$$

as long as  $L + L'' \ll k$

For each of these messages, what is the probability that, after random interleaving, the first two 1s are separated by  $L'$  0s and the last two 1s are separated by  $L''$  0s?

# Turbo codes - A simple example

The total number of possible permutations of 4 bits in a word of  $k$  bits is given by the binomial coefficient

$$\binom{k}{4} = \frac{k \cdot (k-1) \cdot (k-2) \cdot (k-3)}{2 \times 3 \times 4} \approx \frac{k^4}{24}$$

The number of configurations for which the first two 1s are separated by  $L'$  0s and the last two 1s are separated by  $L''$  0s is given by

# Turbo codes - A simple example

$$\binom{k - L' - L''' - 2}{2} = \frac{(k - L' - L''' - 2) \cdot (k - L' - L''' - 3)}{2}$$
$$\approx \frac{k^2}{2} \quad \text{if } L' + L''' \ll k$$

Therefore, the probability to have one of those configurations after interleaving is

$$\Pr\{L', L'''\} \approx \frac{24 \cdot k^2}{2k^4} = \frac{12}{k^2}$$



# Turbo codes - A simple example

The average number of messages  $M(L, L', L'', L''')$  is thus given by

$$\Pr\{L', L'''\} \cdot \binom{k - L - L'' - 2}{2} \approx \frac{12k^2}{2k^2} = 6$$

The contribution of the weight-4 messages to the first four error coefficients  $e(d=4)$ ,  $e(d=5)$ ,  $e(d=6)$  and  $e(d=7)$  is 0 since no message  $M(L, L', L'', L''')$  can lead to a codeword with a Hamming weight smaller than 8.

# Turbo codes - A simple example

The contribution of the weight-4 messages to the error coefficient  $e(d=8)$  is only due to the messages  $M(L, L', L'', L''')$  with  $L = L' = L'' = L''' = 0$ .

Therefore, this contribution is represented by the error coefficient:

$$e(d=8) \approx \frac{4 \times 6}{2k} = \frac{12}{k}$$

# Turbo codes - A simple example

All other messages with weights = 6, 8 and so on do not have to be considered because they lead to codewords with Hamming weights  $> 11$ .

Finally, by putting all these results together, we obtain the error coefficients for the first 5 terms in the union bound:

$$P_{eb} \leq \frac{2}{k} \cdot \text{erfc}\left(\sqrt{\frac{4}{2} \frac{E_b}{N_0}}\right) + \frac{4}{k} \cdot \text{erfc}\left(\sqrt{\frac{5}{2} \frac{E_b}{N_0}}\right) + \frac{6}{k} \cdot \text{erfc}\left(\sqrt{\frac{6}{2} \frac{E_b}{N_0}}\right) + \dots$$
$$\dots + \frac{8}{k} \cdot \text{erfc}\left(\sqrt{\frac{7}{2} \frac{E_b}{N_0}}\right) + \frac{22}{k} \cdot \text{erfc}\left(\sqrt{\frac{8}{2} \frac{E_b}{N_0}}\right) + \dots$$

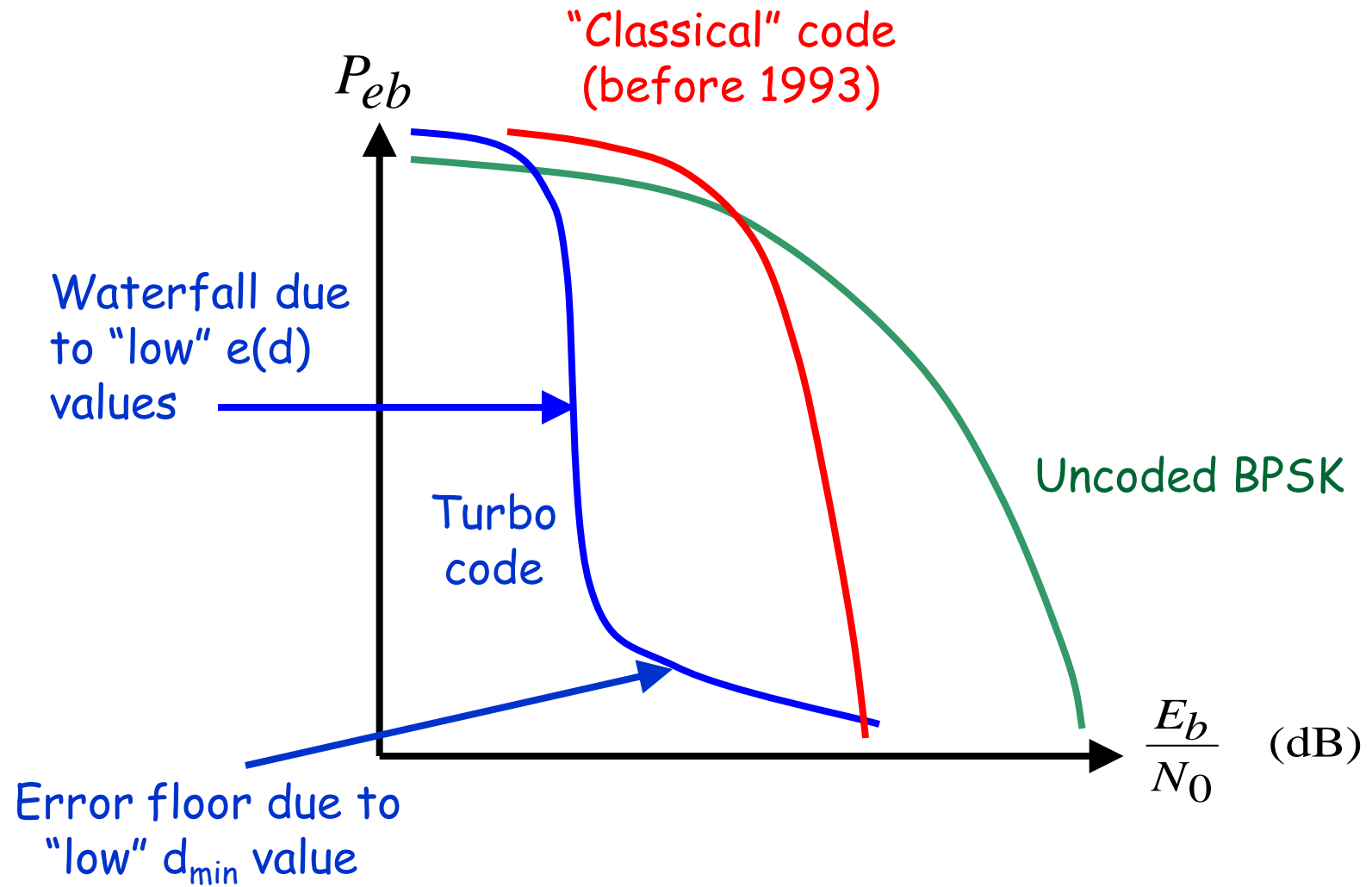
# Turbo codes - A simple example

The effect of concatenating in parallel two RSC codes instead of using only one RSC has resulted in:

- A decrease in coding rate from  $1/2$  to  $1/3$  ☹️
- A very small increase in  $d_{\min}$  from 3 to 4 ☹️
- A huge decrease in the error coefficient values 😊😊:

$$\begin{array}{ccccc} e(d=3) \approx 1 & e(d=4) \approx 1 & e(d=5) \approx 1 & e(d=6) \approx k & e(d=7) \approx 2k \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ e(d=4) \approx \frac{2}{k} & e(d=5) \approx \frac{4}{k} & e(d=6) \approx \frac{6}{k} & e(d=7) \approx \frac{8}{k} & e(d=8) \approx \frac{22}{k} \end{array}$$

# Turbo codes



# Turbo codes

In practice, it is possible to design the interleaver so as to obtain a high  $d_{\min}$ .

Make sure that  $(L + L')$  is always a high number →  
Use a "structured" interleaver instead of a random one.

Carefully designing the interleaver can lead to very high values of  $d_{\min}$ , i.e. very good error performance at high SNR.

# Turbo codes

Example 1: CCSDS (Consultative Committee for Space Data Systems) 16-state turbo code with  $k = 3568$ .

- $R_c = 1/2$ :  $d_{\min} = 20$ ,  $w_{d\min} = 1 \rightarrow G = 10.00$  dB
- $R_c = 1/3$ :  $d_{\min} = 40$ ,  $w_{d\min} = 9 \rightarrow G = 11.25$  dB
- $R_c = 1/4$ :  $d_{\min} = 56$ ,  $w_{d\min} = 6 \rightarrow G = 11.46$  dB
- $R_c = 1/6$ :  $d_{\min} = 93$ ,  $w_{d\min} = 6 \rightarrow G = 11.90$  dB

Example 2: UMTS rate-1/3 8-state turbo code with  $k = 40 \rightarrow 5114$ . For  $k = 320$ ,  $d_{\min} = 24$ ,  $w_{d\min} = 4 \rightarrow G = 9.03$  dB

# Turbo codes

Current standards using turbo codes

- CCSDS (deep-space missions)
- UMTS (3G mobile)
- DVB-RCS and DVB-RCT (Return Channel over Satellite & Return Channel over Terrestrial)
- M4 (Inmarsat)
- Skyplex (Eutelsat)
- WiMAX (IEEE 802.16)



# The rediscovery of LDPC codes

1995: David J C McKay (University of Cambridge!) rediscovers LDPC codes.

He shows that these codes, when decoded in an iterative fashion, can actually perform very close to the Shannon limit.



Today, some coding experts consider LDPC codes as even more attractive than turbo codes for future practical applications.

# The rediscovery of LDPC codes

Ex: (3, 3)-regular LDPC code

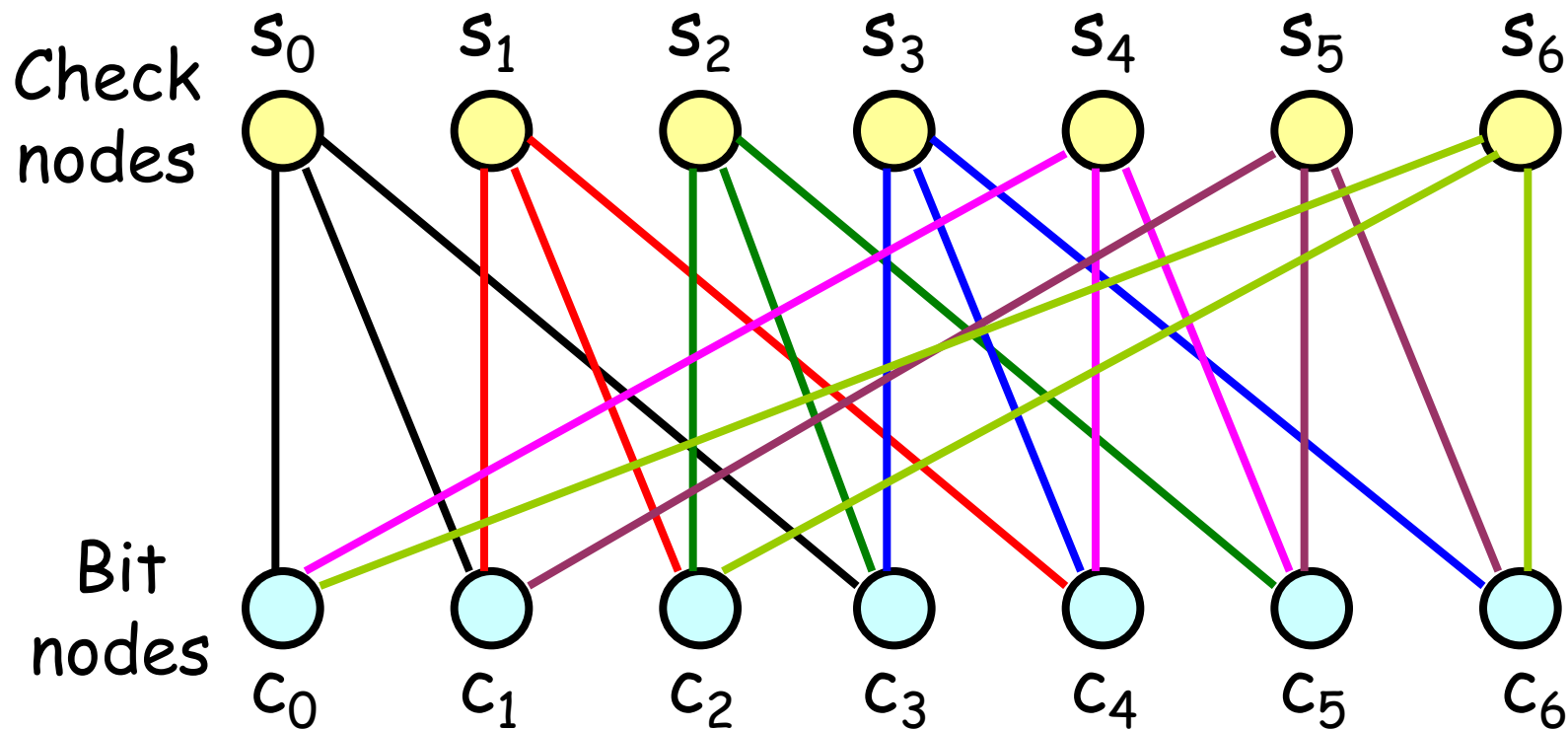
$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{aligned} c_0 + c_1 + c_3 &= 0 \\ c_1 + c_2 + c_4 &= 0 \\ c_2 + c_3 + c_5 &= 0 \\ c_3 + c_4 + c_6 &= 0 \\ c_0 + c_4 + c_5 &= 0 \\ c_1 + c_5 + c_6 &= 0 \\ c_0 + c_2 + c_6 &= 0 \end{aligned}$$

The code is completely defined by a set of parity-check equations

When  $H$  is large, A LDPC code can be made look like a random code. But, how to implement efficient decoding?

# The rediscovery of LDPC codes

Graphical representation of LDPC codes:  
Tanner graph (R. M. Tanner, 1981)



# The rediscovery of LDPC codes

How to decode LDPC codes?

Assume transmission over BPSK, AWGN channel.  
We receive a word  $R = (r_0, r_1, \dots, r_6)$  composed of 7 analogue channel samples.

This word  $R$  is an estimate of the transmitted 7-bit codeword  $C = (c_0, c_1, \dots, c_6)$ . For instance:

$$r_0 = (2c_0 - 1) + n_0$$

Transmitted bit

Gaussian noise sample

# The rediscovery of LDPC codes

First, we take a hard decision on  $R$ , thus yielding the binary word  $R'$ , and then compute the vector syndrome  $S = R'.H^T$ :

- If  $S = 0$ ,  $R'$  is a valid codeword and no further processing is required.
- If  $S \neq 0$ ,  $R'$  is not a valid codeword and we need to decode  $R = (r_0, r_1, \dots, r_6)$ .

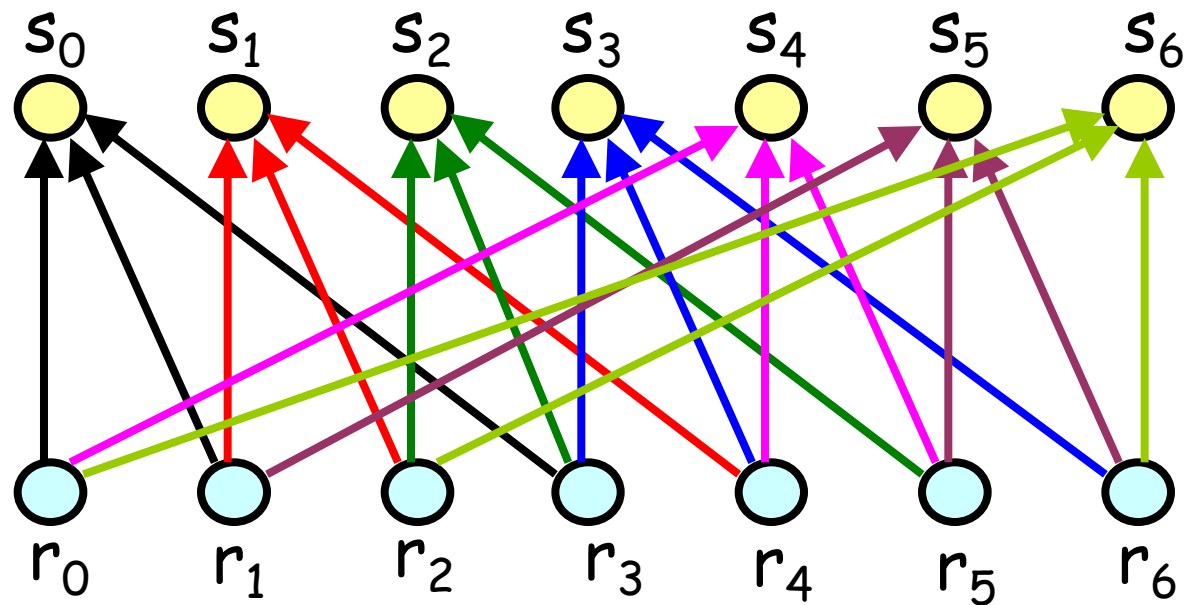
Once again, ML decoding is not an option when dealing with long blocks  $\rightarrow$  Iterative decoding algorithm (as with turbo codes).

# The rediscovery of LDPC codes

The Tanner graph can be used to understand the iterative decoding algorithm (AKA belief propagation, message passing, sum-product).

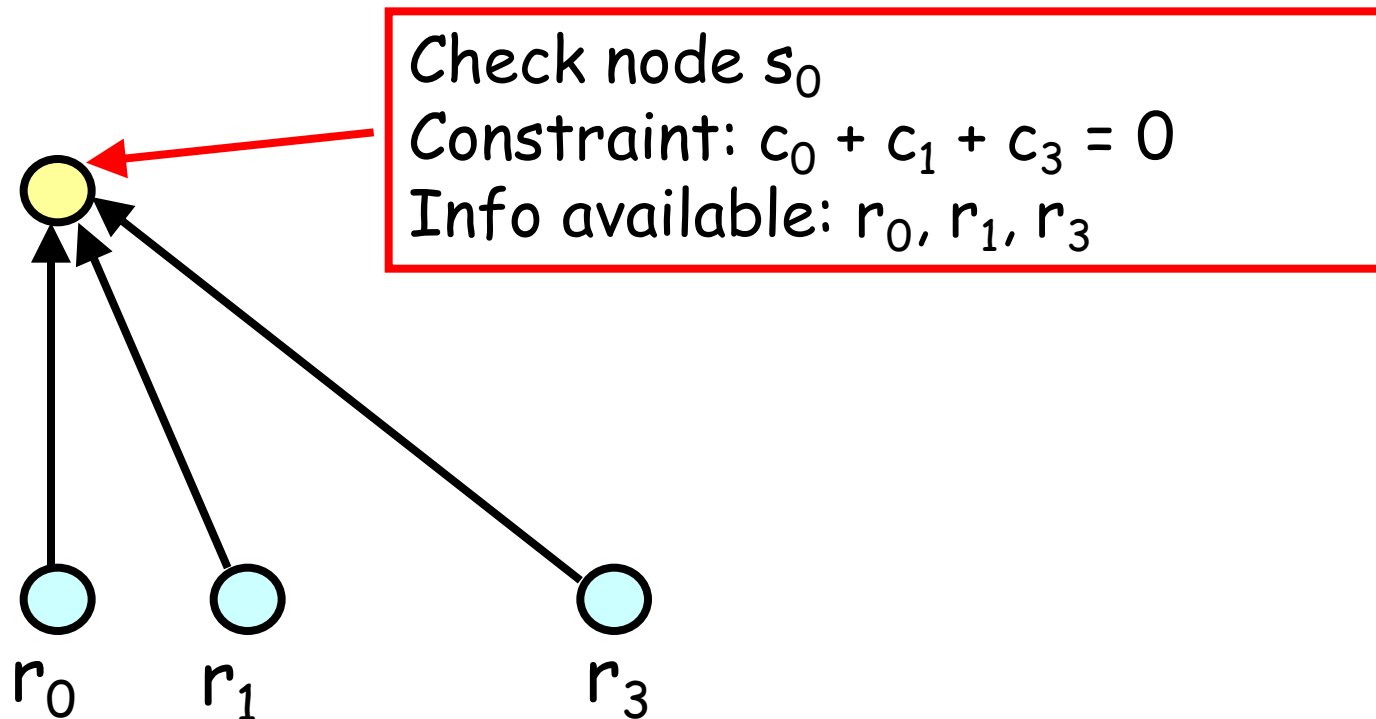
First decoding iteration:

Channel samples are sent to the check nodes



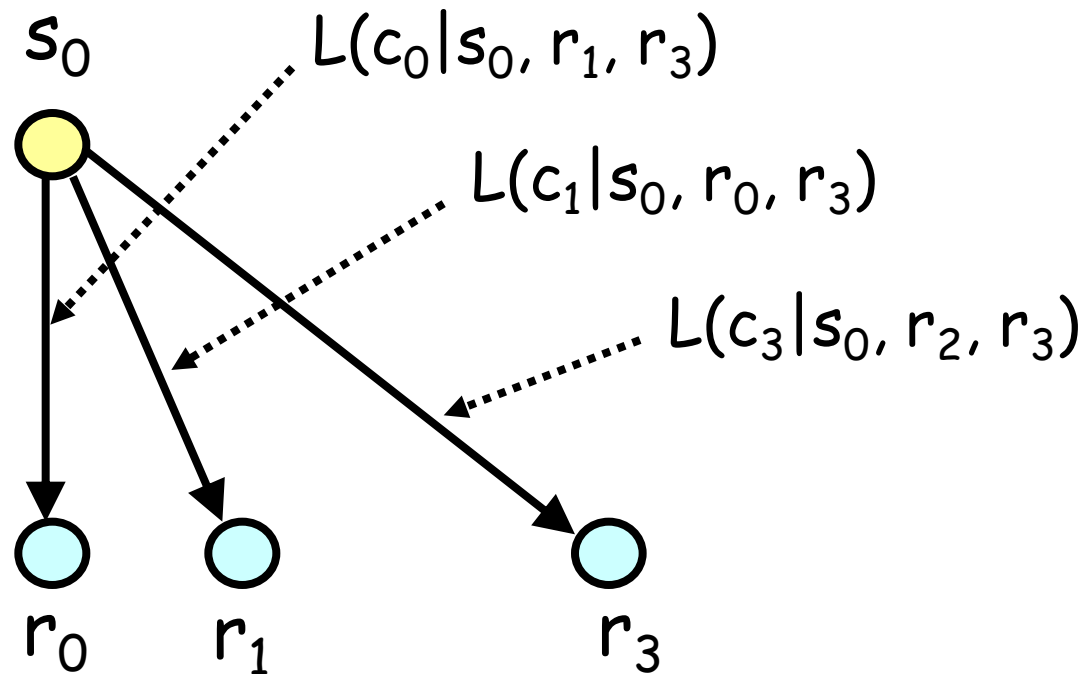
# The rediscovery of LDPC codes

Each check node computes new estimates of each input bit based on the node constraint and the estimates of the other input bits.



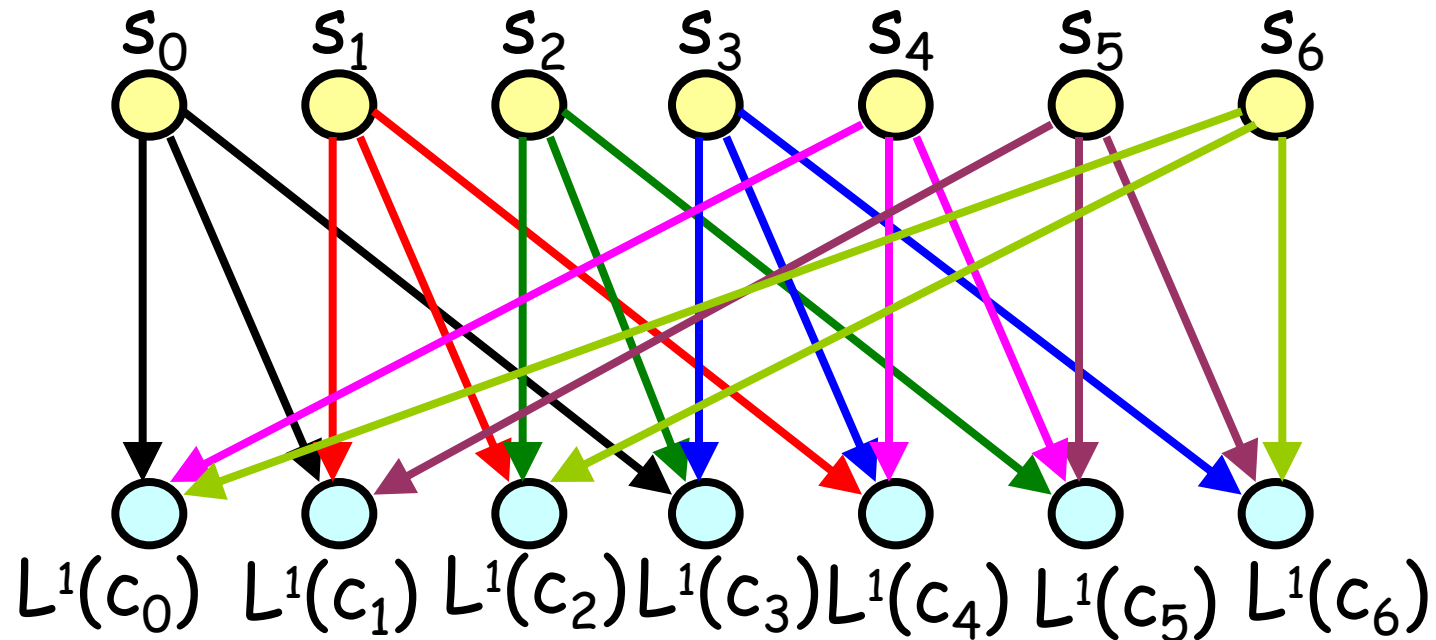
# The rediscovery of LDPC codes

The new estimates of each input bit are sent back to the relevant bit nodes.





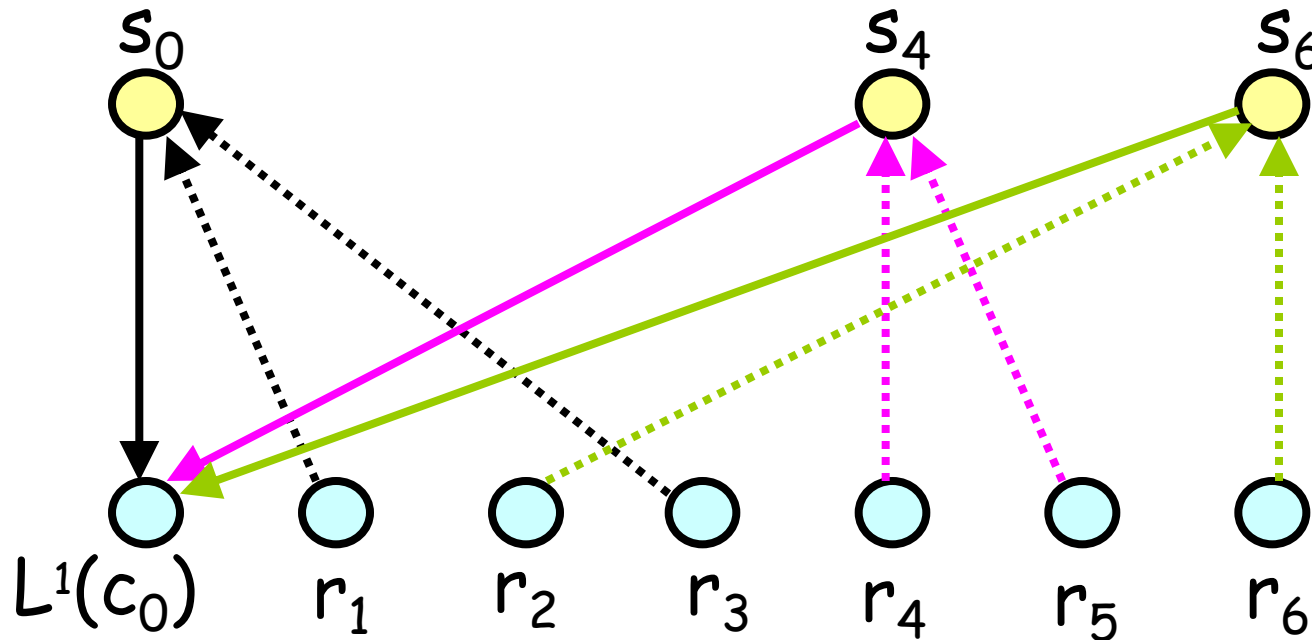
# The rediscovery of LDPC codes



The bit node values are updated using info coming back from the check nodes.

# The rediscovery of LDPC codes

Ex: The updated estimate  $L^1(c_0)$  of the coded bit  $c_0$  is given by  $L^1(c_0) = r_0 + L(c_0|s_0, r_1, r_3) + L(c_0|s_4, r_4, r_5) + L(c_0|s_6, r_2, r_6)$



# The rediscovery of LDPC codes

Since  $L^1(c_0)$  is the combination of 4 independent estimates, it is (almost certainly) more reliable than the original channel sample  $r_0$  alone.

We now have a new estimate  $R^1 = (L^1(c_0), L^1(c_1), \dots, L^1(c_6))$  of the transmitted codeword  $C = (c_0, c_1, \dots, c_6)$ .

A hard decision is taken, yielding the binary word  $R^1$ , and the syndrome vector  $S = R^1 \cdot H^T$  is computed once again.

# The rediscovery of LDPC codes

- If  $S = 0$ ,  $R^{1'}$  is a valid codeword and the decoding process can be stopped.
- If  $S \neq 0$ ,  $R^{1'}$  is still not a valid codeword and the previous decoding procedure can be repeated again.

→ We need to go for a second decoding iteration.

Ex: The estimate  $L^1(c_0)_{\text{ext}} = L^1(c_0) - L(c_0|s_0, r_1, r_3) = r_0 + L(c_0|s_4, r_4, r_5) + L(c_0|s_6, r_2, r_6)$  is sent to  $s_0$ .

# The rediscovery of LDPC codes

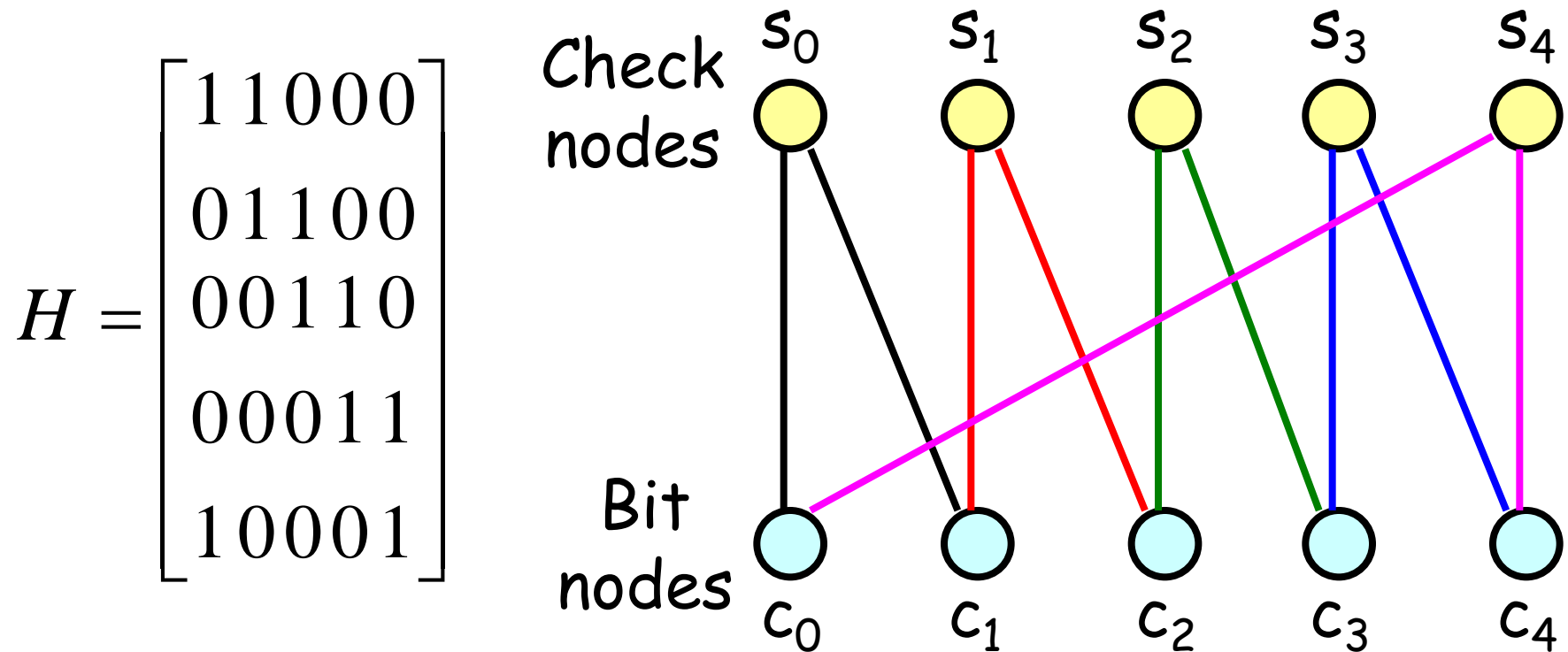
We must NOT send directly  $L^1(c_0)$  back to check node  $s_0$  because  $L^1(c_0)$  contains some info which has already been produced by  $s_0$ .

In iterative decoding, one must NOT send to a node info that was previously generated by this node.

The estimate  $L^1(c_0)_{\text{ext}}$  that is exchanged between nodes is called "extrinsic info".

# The rediscovery of LDPC codes

Example: Consider the following Tanner graph, which corresponds to a (2, 2)-regular LDPC code.



# The rediscovery of LDPC codes

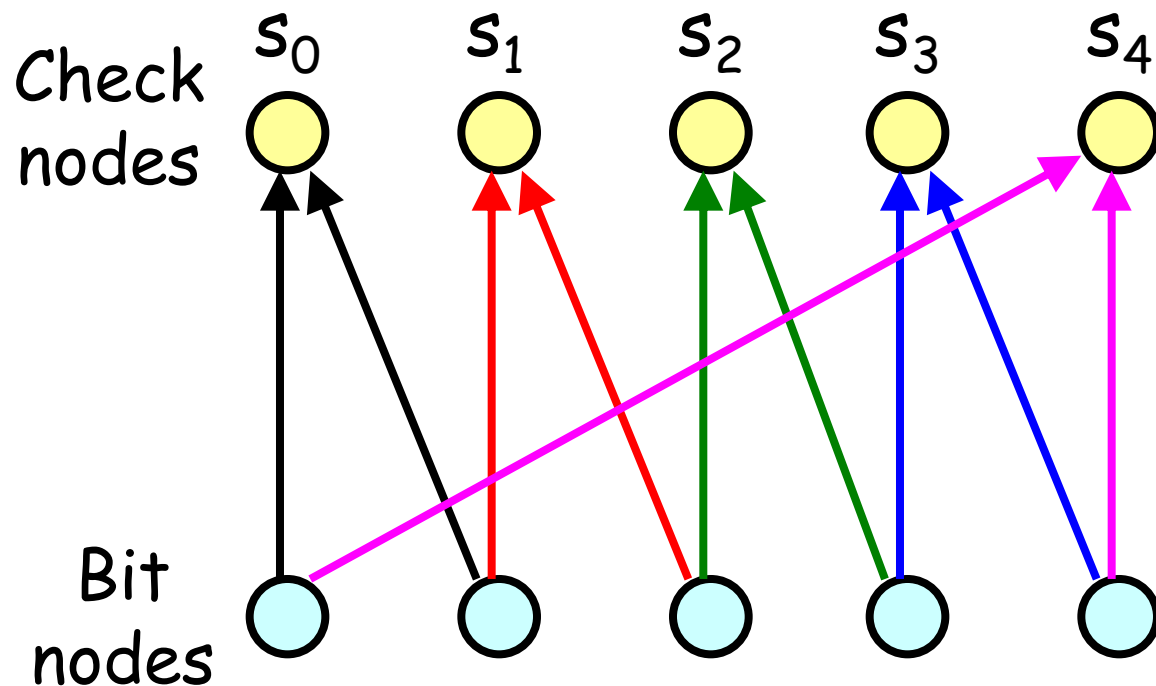
Assume we receive the word  $R = (r_0, \dots, r_4) = (+1. +1. -2. +2. -1.)$ , which after hard decision corresponds to the binary word  $R' = (11010)$ .

Unfortunately,  $R'$  is not a valid codeword because:

$$S = (11010) \cdot \begin{bmatrix} 10001 \\ 11000 \\ 01100 \\ 00110 \\ 00011 \end{bmatrix} = (01111) \neq 0$$

# The rediscovery of LDPC codes

Iteration 1:





# The rediscovery of LDPC codes

How to update the bit nodes?

If a particular check node  $s_k$  is associated with the constraint  $(c_i + c_j = 0)$ , then we can write:

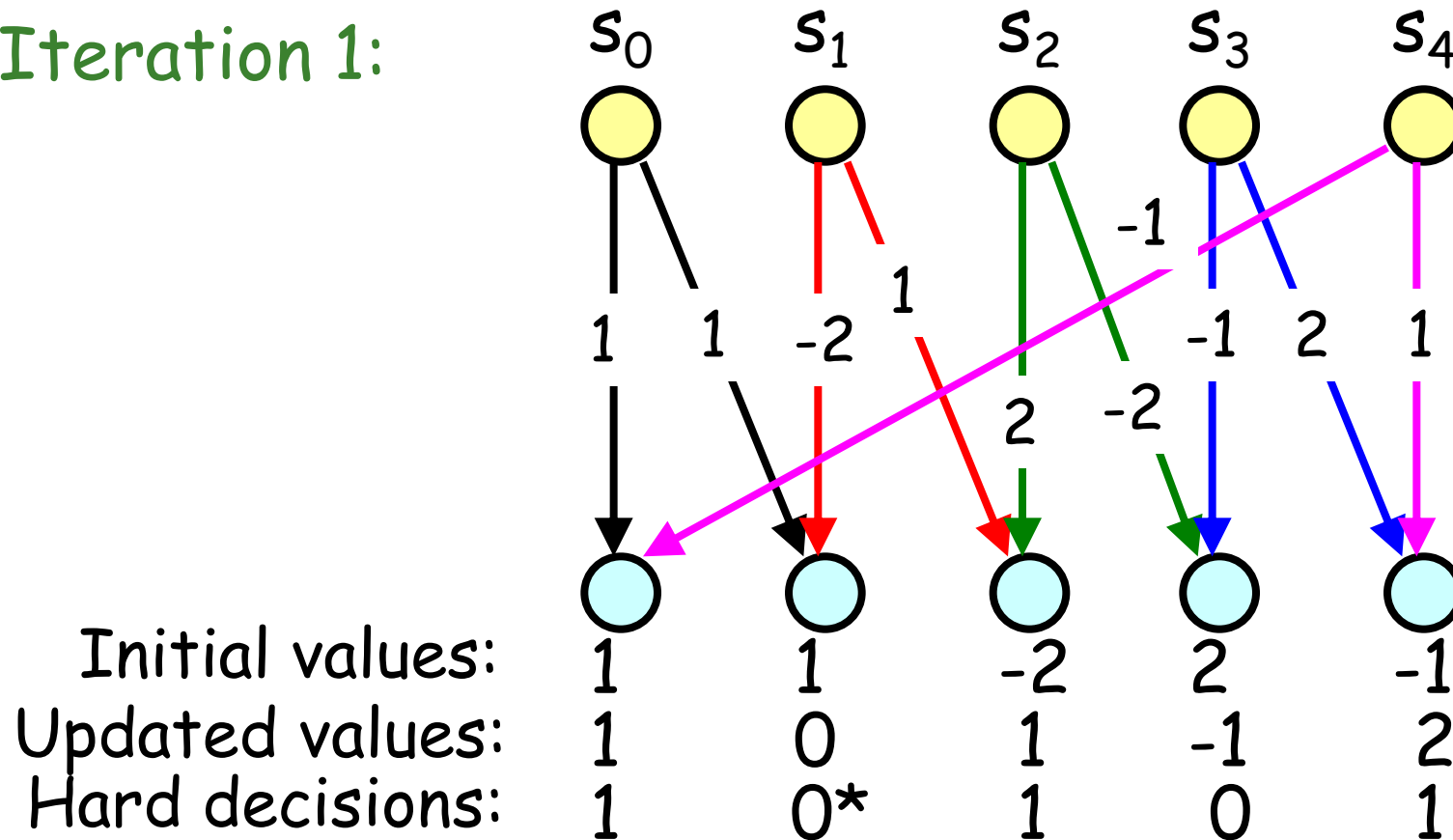
$$\Pr\{c_i = 0\} = \Pr\{c_j = 0\} \text{ and } \Pr\{c_i = 1\} = \Pr\{c_j = 1\}$$

$$\Rightarrow L(c_i | s_k, r_j) = r_j$$

With 3 or more bits involved in the check nodes, equations are much more complicated.

# The rediscovery of LDPC codes

Iteration 1:



\* for instance

# The rediscovery of LDPC codes

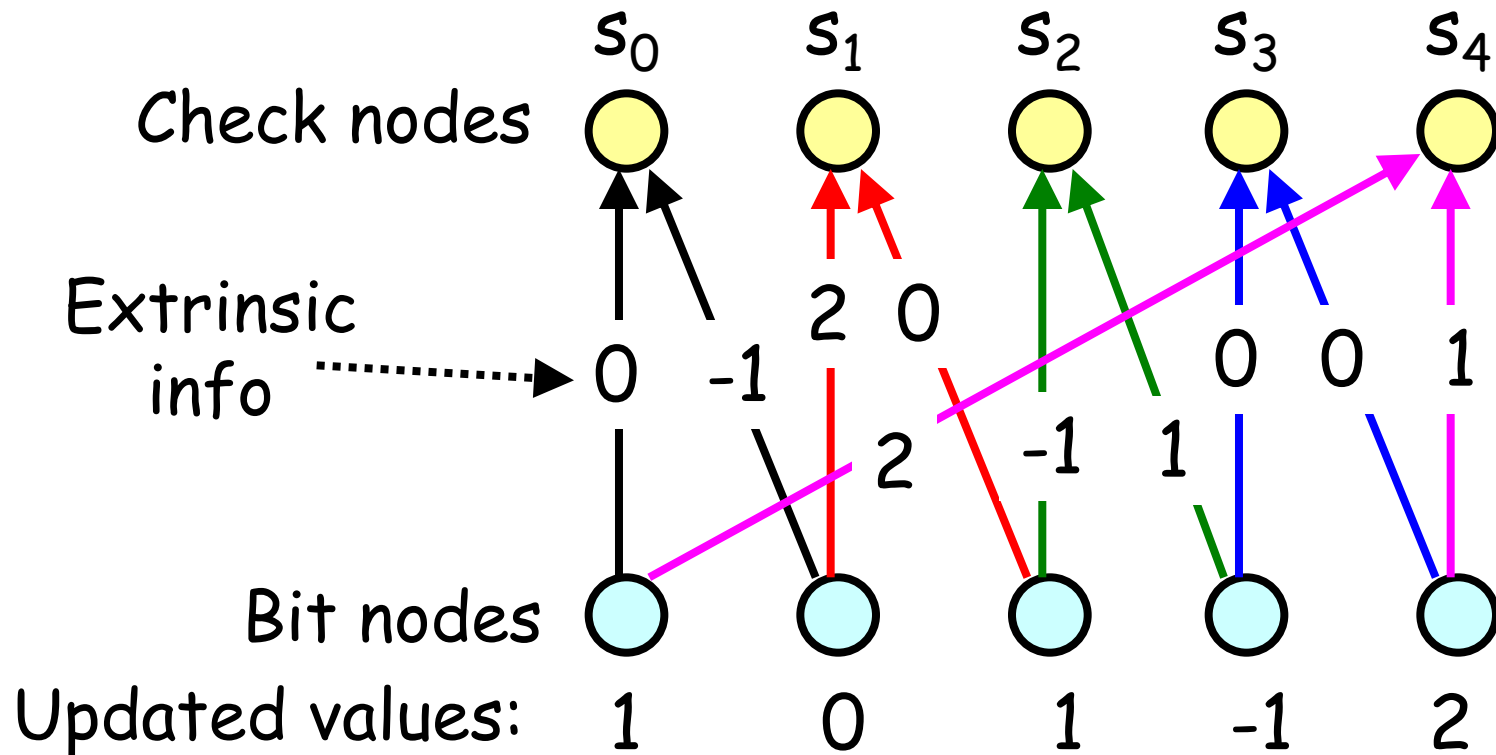
The updated hard decision is not a valid codeword since:

$$S = (10101) \cdot \begin{bmatrix} 10001 \\ 11000 \\ 01100 \\ 00110 \\ 00011 \end{bmatrix} = (11110) \neq 0$$

We need to perform another decoding iteration.

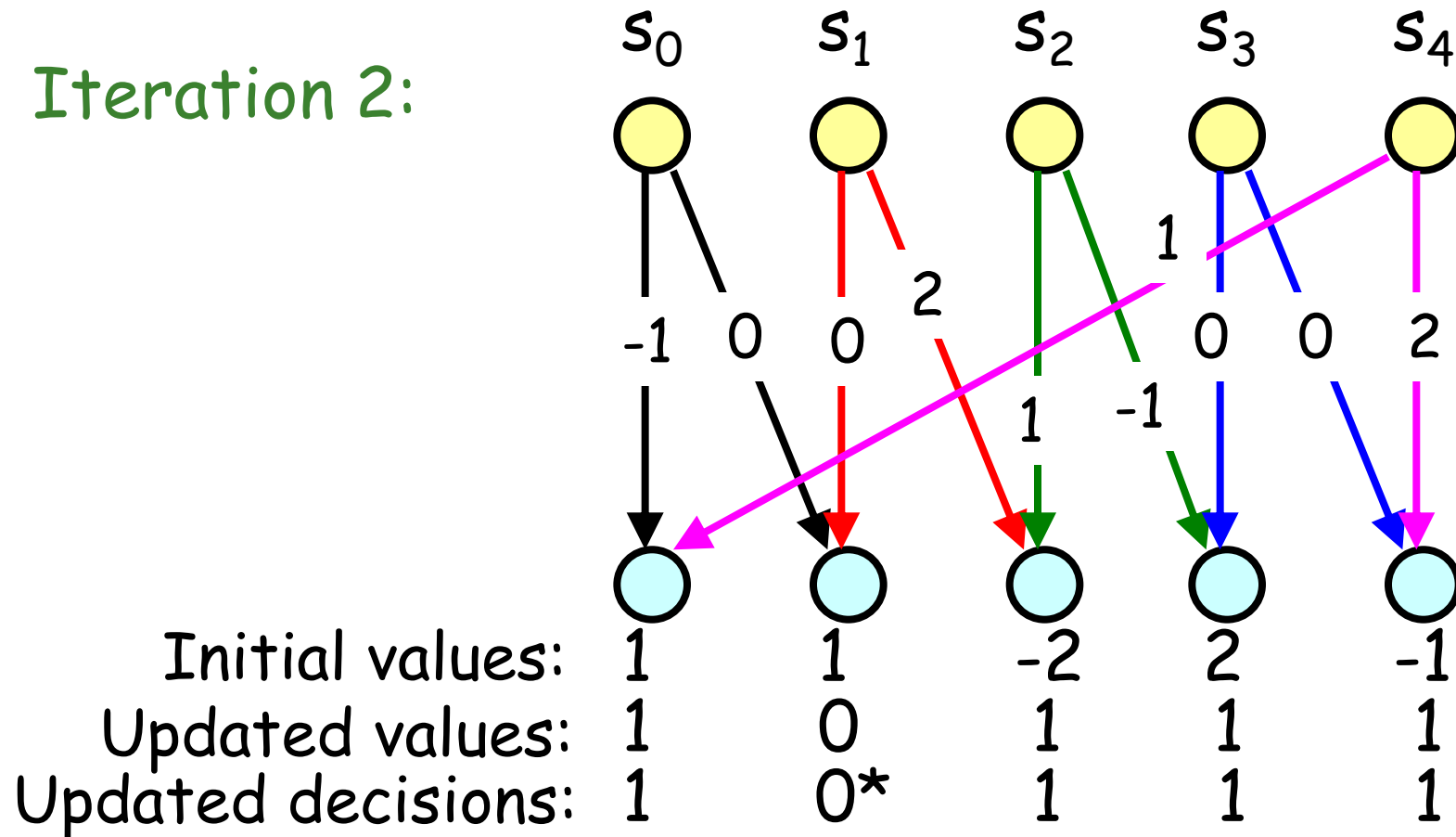
# The rediscovery of LDPC codes

## Iteration 2:



# The rediscovery of LDPC codes

Iteration 2:



# The rediscovery of LDPC codes

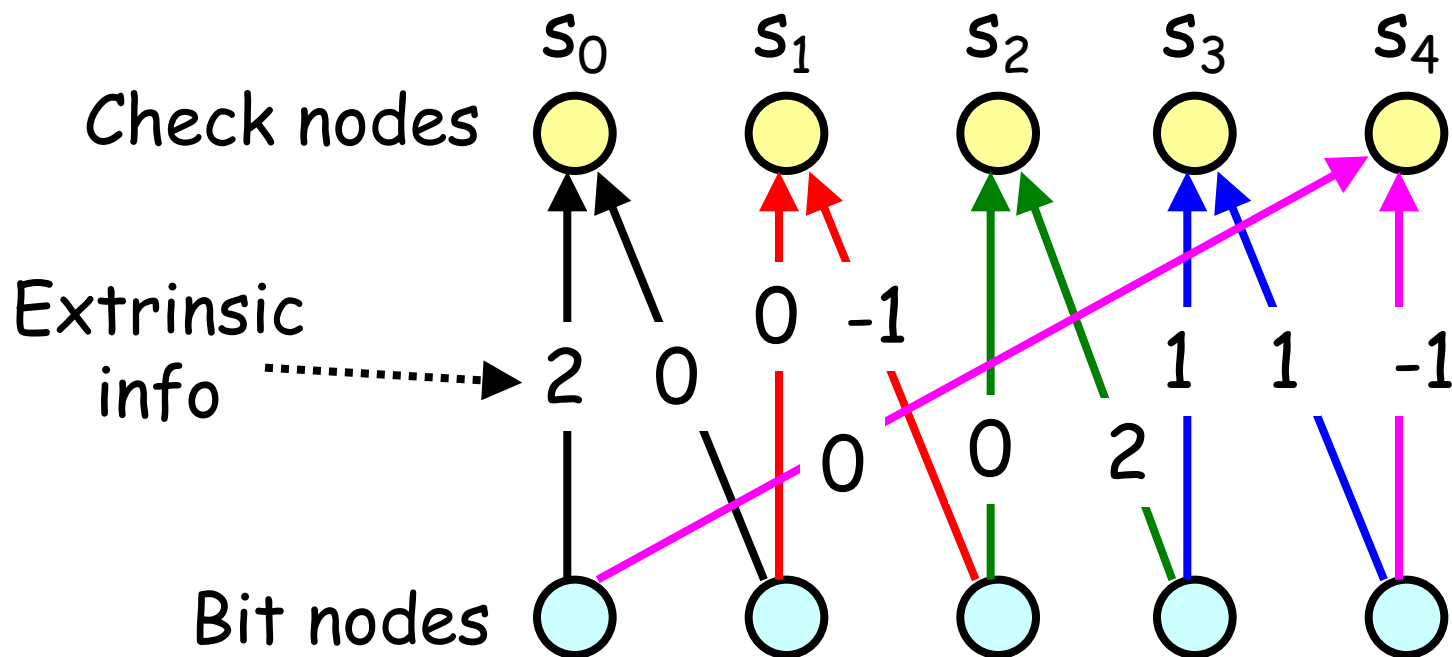
The updated hard decision is not a valid codeword since:

$$S = (10111) \cdot \begin{bmatrix} 10001 \\ 11000 \\ 01100 \\ 00110 \\ 00011 \end{bmatrix} = (11000) \neq 0$$

We need to proceed with another decoding iteration.

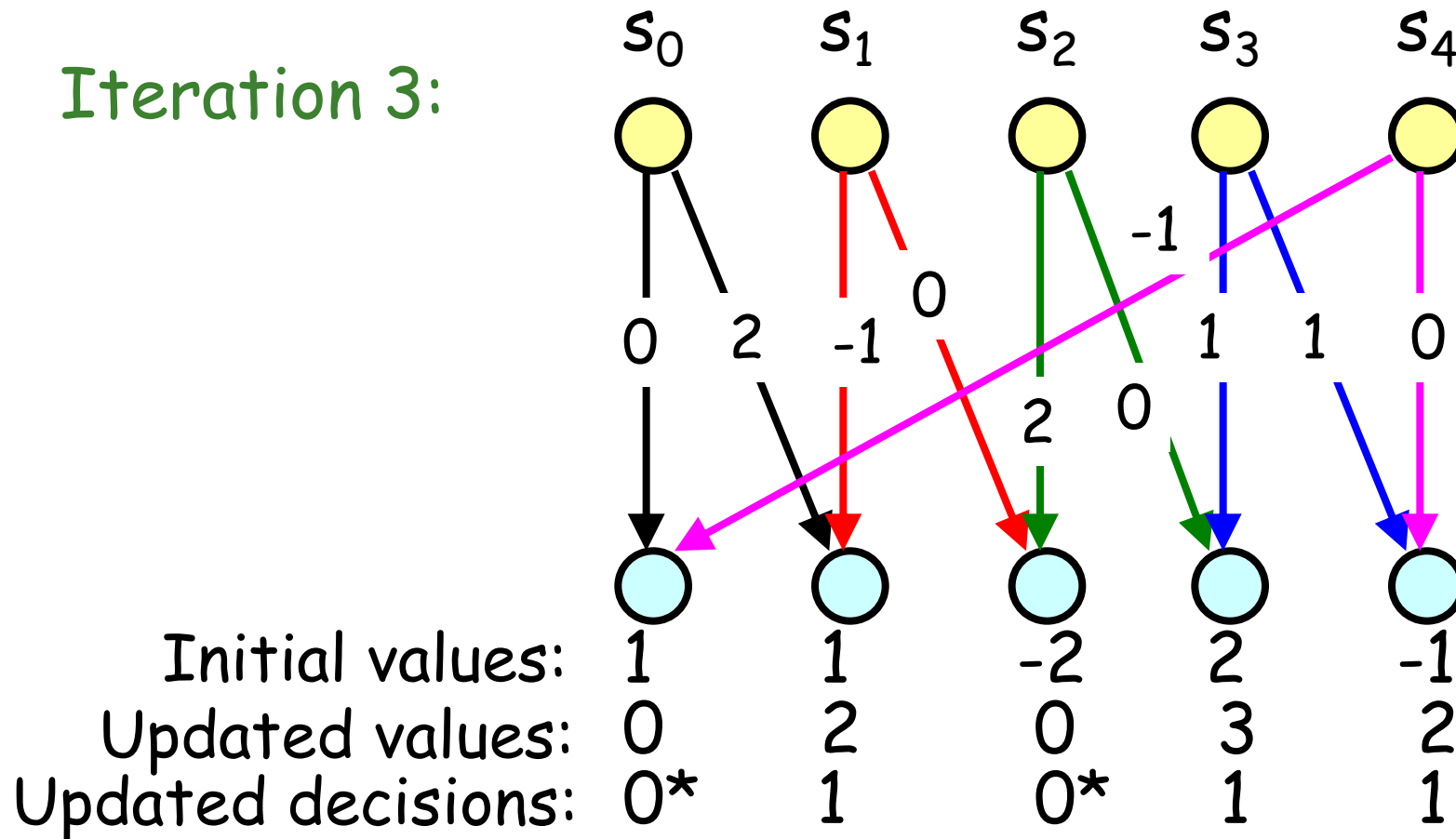
# The rediscovery of LDPC codes

## Iteration 3:



# The rediscovery of LDPC codes

Iteration 3:





# The rediscovery of LDPC codes

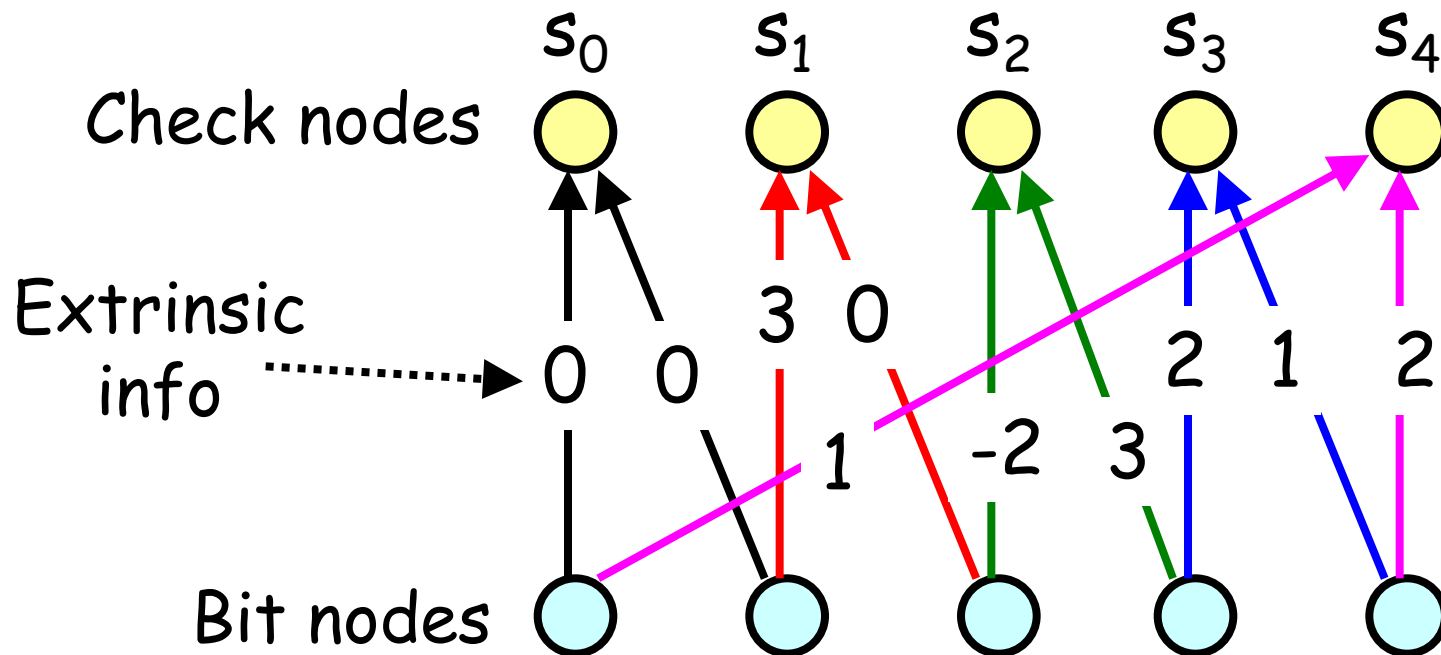
The updated hard decision is not a valid codeword since:

$$S = (01011) \cdot \begin{bmatrix} 10001 \\ 11000 \\ 01100 \\ 00110 \\ 00011 \end{bmatrix} = (11101) \neq 0$$

We need to proceed with another decoding iteration.

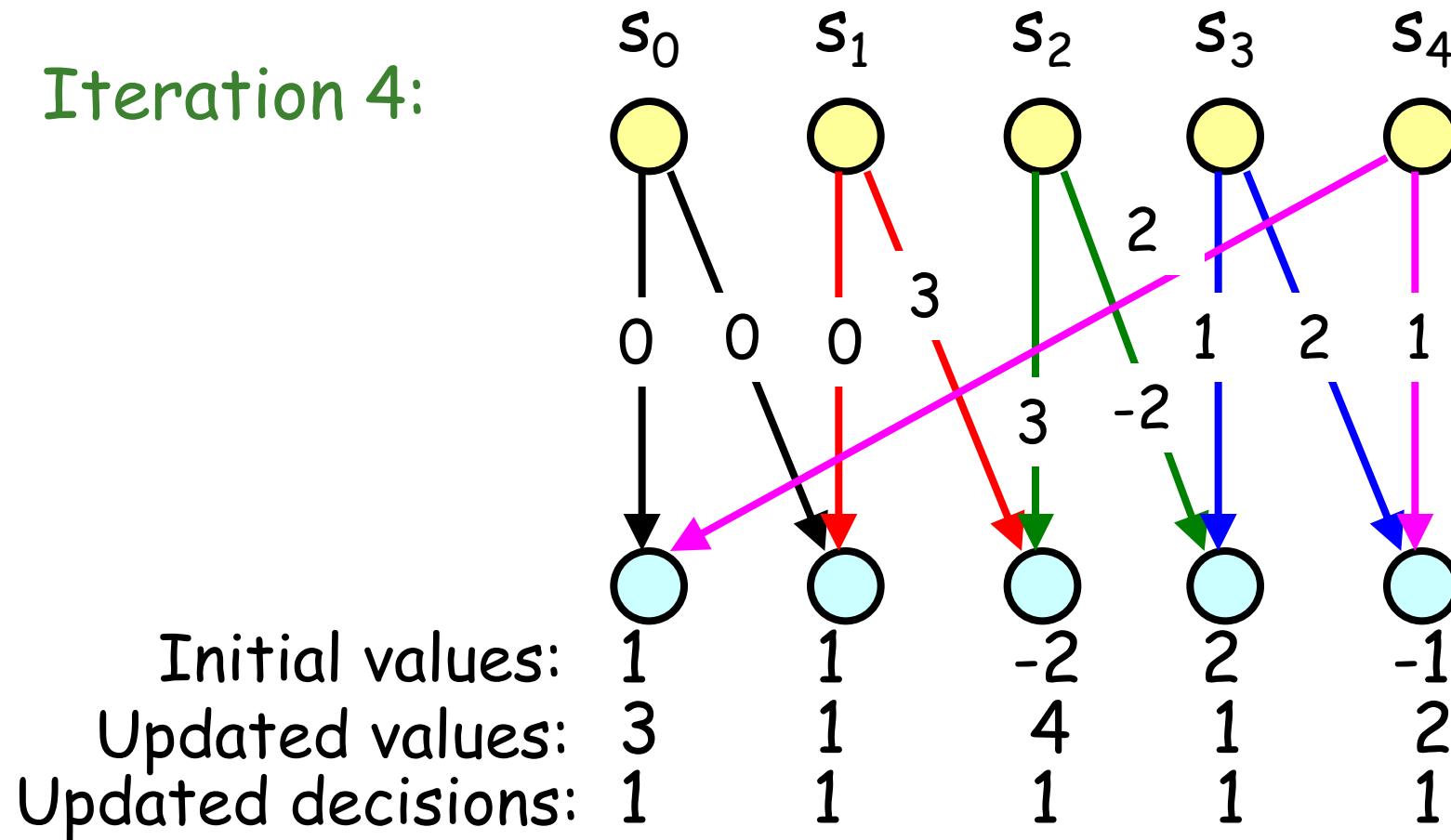
# The rediscovery of LDPC codes

## Iteration 4:



# The rediscovery of LDPC codes

Iteration 4:



# The rediscovery of LDPC codes

The updated hard decision is a valid codeword since:

$$S = (11111) \cdot \begin{bmatrix} 10001 \\ 11000 \\ 01100 \\ 00110 \\ 00011 \end{bmatrix} = (00000)$$

The iterative decoding algorithm has converged in 4 iterations. Two errors could be corrected in the received word of 5 bits.

# A big leap into the future

The last word for Robert MacEliece:

Claude Shannon - Born on the planet Earth (Sol III) in the year 1916 A.D. Generally regarded as the father of the Information Age, he formulated the notion of channel capacity in 1948 A.D. Within several decades, mathematicians and engineers had devised practical ways to communicate reliably at data rates within 1% of the Shannon limit...

Encyclopedia Galactica, 166<sup>th</sup> edition