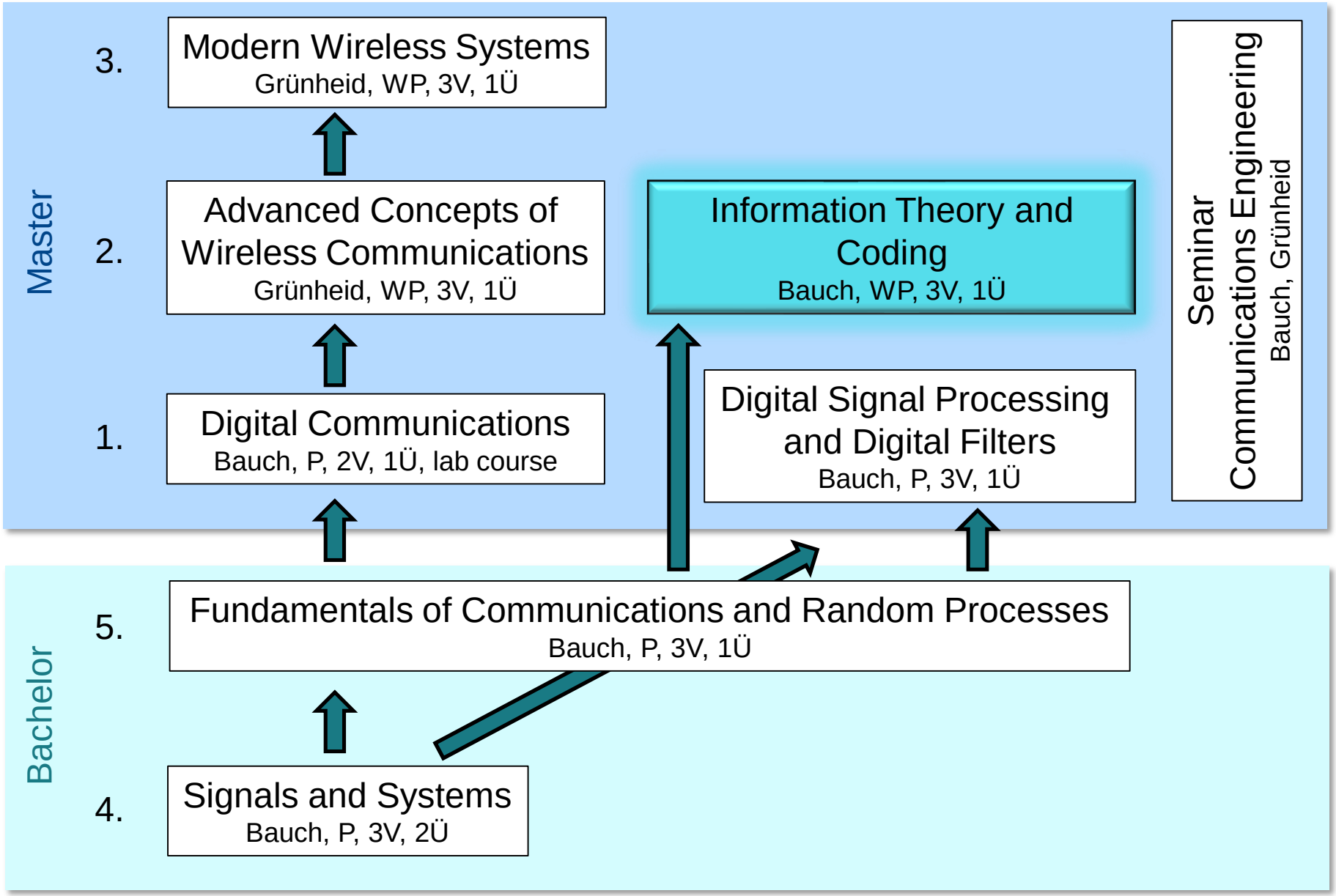
Information Theory und Coding

Prof. Dr. Gerhard Bauch

Institut für Nachrichtentechnik

Institute of Communications - Courses

Semester



Literature (1)

Error Control Coding

(1) Bossert, M.: *Channel Coding for Telecommunications.* Wiley.
(2) Friedrichs, B.: *Kanalcodierung.* Springer.

(3) Clark, G.C., Cain, J.B.: *Error-Correction Coding for Digital Communications*. Plenum Press.
(4) Huffman, W.C., Pless, V.: *Fundamentals of Error-Correcting Codes.*
(5) Lin, S., Costello, D.: Error Control Coding. Prentice Hall.
(6) Michelson, A.M., Levesque, A.H.: *Error-Control Techniques for Digital Communication.* March 1985, Wiley.
(7) Moon, T.K.: *Error Correction Coding: Mathematical Methods and Algorithms.* July 2005, Wiley.
(8) Moreira, J.C., Farrell, P.G.: *Essentials of Error-Control Coding.* July 2006, Wiley.
(9) Morelos-Zaragoza, R.: *The art of error control coding.* 2nd edition. Wiley-IEEE Press.
(10) Neubauer, A., Freudenberger, J., Kühn, V.: *Coding Theory: Algorithms, Architectures and Applications.* October 2007, Whiley.
(11) Öberg, T.: *Modulation, Detection and Coding.* October 2001, Wiley.
(12) Roth, R.: *Introduction to Coding Theory.*
(13) Ryan, W., Lin, S.: Channel Codes Classical and Modern
(14) Shokranian, S., Shokrollahi, M.A.: *Coding Theory and Bilinear Complexity*.

Literature (2)

Turbo Codes and Iterative Decoding

(1) Abbasfar, A.: *Turbo-like Codes. Design for High Speed Decoding.*

(2) Glavieux, A. (Editor): *Channel Coding in Communication Networks: From Theory to Turbocodes*. January 2007, Wiley-ISTE.

(3) Hanzo, L., Liew, T.H., Yeap, B.L.: *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission Over Fading Channels.* Wiley & Sons.

(4) Johnson, S.: *Iterative Error Correction.* Cambridge.

(5) Richardson, T., Urbanke, R.: *Modern Coding Theory.* Cambridge University Press.

(6) Schlegel, C., Perez, L.: *Trellis and Turbo Coding.* IEEE Press Series on Digital and Mobile Communications. Wiley & Sons.

(7) Sripimanwat, K. (editor): *Turbo Code Applications - a Journey from a Paper to realization.*

Literature (3)

Special Codes

(1) Johannesson, R., Zigangirov, K.S.: *Fundamentals of Convolutional Coding.* March 1999, Wiley-IEEE Press.

(2) Wicker, B., Bhargava, V.K. (editors): *Reed-Solomon Codes and Their Applications.* September 1999, Wiley-IEEE Press.

(3) Anderson, J. B., Svensson, A.:*Coded Modulation Systems*. Series: Information Technology: Transmission, Processing and Storage, Springer.

(4) Carrasco, R.A., Johnston, M.: *Non-Binary Error Control Coding for Wireless Communication and Data Storage.* December 2008, Wiley.

(5) Blahut, R.: *Algebraic Codes for Data Transmission.*

(6) Blahut, R.: *Algebraic Codes on Lines, Planes, and Curves. An Engineering Approach.*

Information Theory

(1) Gallager, R. G.: *Information theory and reliable communication*. Whiley-VCH

(2) Cover, T., Thomas, J.: *Elements of information theory.* Whiley.

(3) Shannon, C.E.: *The Collected Works*, IEEE

Probabilities, Stochastic Processes

(1) Hänsler, E.: *Statistische Signale*. Springer.

(2) Papoulis, A., Pillai, S.U.: *Probability, random variables and stochastic processes*, McGraw Hill.

Literature (4)

Communications Engineering

(1) Proakis, J.G., Salehi, M.: *Digital Communications*, McGraw-Hill Int. Book Company.

(2) Haykin, S.: *Communication Systems*, Wiley.

(3) Benedetto, S., Biglieri, E., Castellani, V.: *Digital Transmission Theory*, Prentice-Hall.

(4) Lee, E.A., Messerschmitt, D.G.: *Digital Communication*, Kluwer.

Goals and Types of Coding in Communications

Goal:

Represent and transmit data with best possible quality.

Types of Coding:

- Source Coding

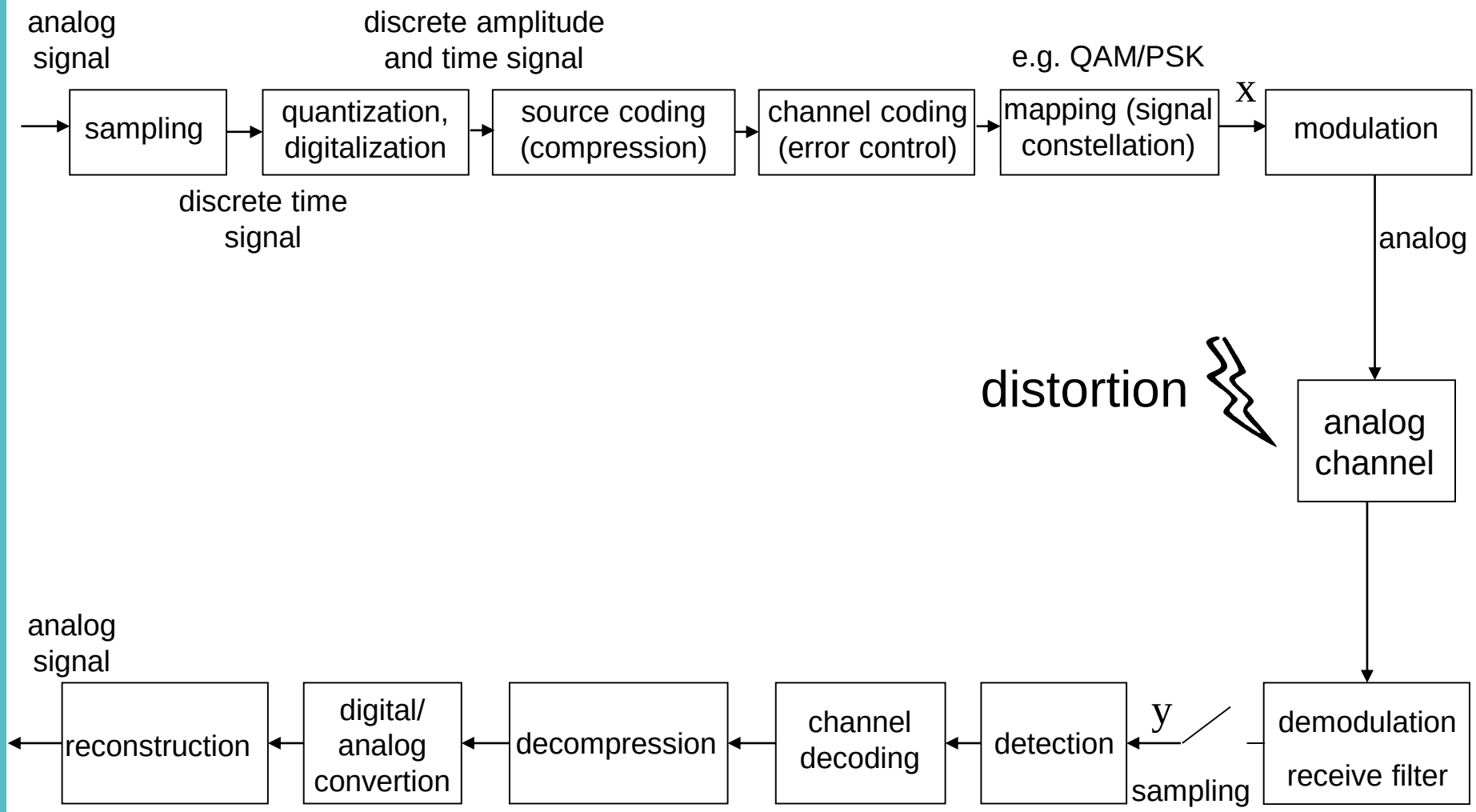
  Represent information in efficient format (compression).

- Channel Coding
  - Enable reliable (error-free) transmission.
  - Enable detection and/or correction of transmission errors.
  - Enable compensation of distortions which cannot be accounted for by other means.

- Cryptography

  Protect information (confidentiality, integrity, authenticity)

Digital Communications System



Factors in Digital Communications Revolution

Most important:

- Microelectronics
- Optical Fiber Technology

Essential:

- Channel coding
- Data compression
- Signal processing algorithms

A Simple Example: Barcodes

Information: 081547112369

European Article Number (EAN 13)

- 12 numerical characters + 1 check digit (error detection).
- Identical to Japanese Article Number (JAN) and International Article Number (IAN) (and to UPC 13 in the USA).
- Examples: ISBN number, ISSN number, supermarket products etc.

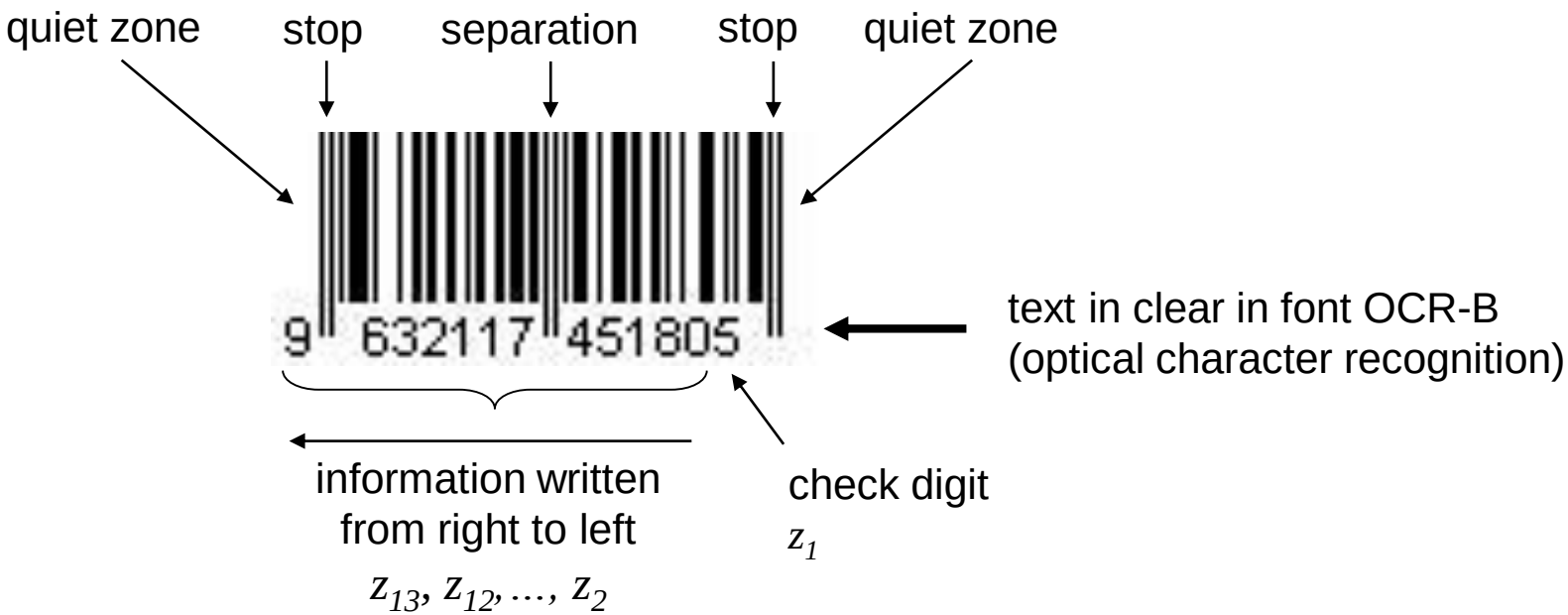
Universal Product Code (UPC-A)

- 11 numerical characters + 1 check digit (error detection).
- Used in USA (now changing to UPC 13 = EAN 13).

Aztec Code

- 8 bit values (ANSI X3.4 = ASCII , ISO 8859-1 = Latin alphabet No. 1).
- 12 – 3832 numeric or 12 - 3067 alphabetic characters.
- Reed-Solomon error correcting code.
- Examples: Train tickets (e.g. Deutsche Bahn)
- Well suited for displays of cell phones.

European Article Number (EAN 13)



Calculation of check digit:

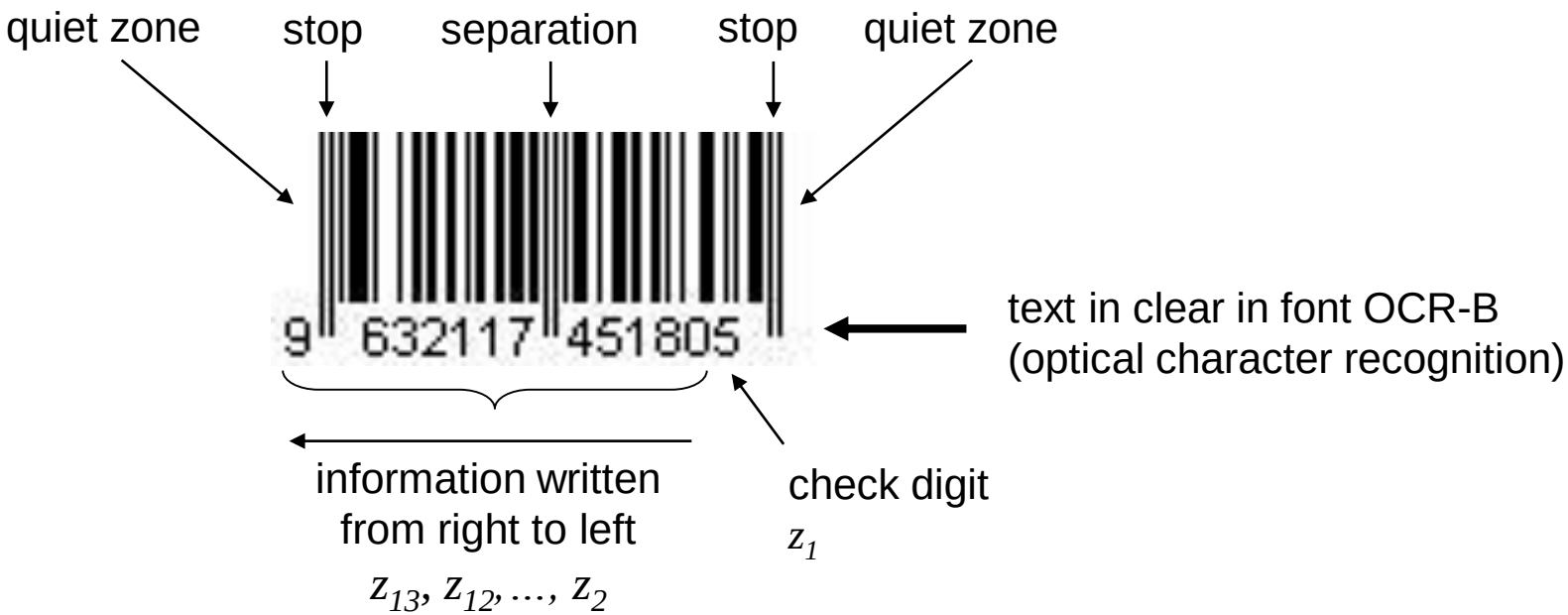
$$z_1 = \left\{ 10 - \left[\left(\sum_{i=1}^{6} \left(3z_{2i} + z_{2i+1} \right) \right) \bmod 10 \right] \right\} \bmod 10$$

Example:

$$z_1 = \left\{ 10 - \left[\underbrace{\left(3 \cdot 0 + 8 + 3 \cdot 1 + 5 + 3 \cdot 4 + 7 + 3 \cdot 1 + 1 + 3 \cdot 2 + 3 + 3 \cdot 6 + 9 \right)}_{75} \bmod 10 \right] \right\} \bmod 10$$

$$= \left\{ 10 - 75 \bmod 10 \right\} \bmod 10 = 5$$

EAN 13 – Error Detection



Calculation of check digit:

$$z_1 = \left\{ 10 - \left[\left(\sum_{i=1}^{6} \left(3z_{2i} + z_{2i+1} \right) \right) \bmod 10 \right] \right\} \bmod 10$$

Example: Error in z_{10} 9 6 3 ~~2~~ 1 1 7 4 5 1 8 0 5 → 9 6 3 4 1 1 7 4 5 1 8 0 5

$$z_1 = \left\{ 10 - \left[\underbrace{\left(3\cdot 0 + 8 + 3\cdot 1 + 5 + 3\cdot 4 + 7 + 3\cdot 1 + 1 + 3\cdot 4 + 3 + 3\cdot 6 + 9 \right)}_{81} \bmod 10 \right] \right\} \bmod 10$$

$$= \left\{ 10 - 81 \bmod 10 \right\} \bmod 10 = 9 \neq 5$$ → error detected

European Article Number (EAN 13)

text in clear in font OCR-B
(optical character recognition)

information written
from right to left

check digit

Code C is used for right hand side (z_1-z_6)

reflect

invert

symbol	code C	code A	code B
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

coding of 13th symbol z_{13}

13	12	11	10	9	8	7
0	A	A	A	A	A	A
1	A	A	B	A	B	B
2	A	A	B	B	A	B
3	A	A	B	B	B	A
4	A	B	A	A	B	B
5	A	B	B	A	A	B
6	A	B	B	B	A	A
7	A	B	A	B	A	B
8	A	B	A	B	B	A
9	A	B	B	A	B	A

Types of Coding in EAN 13 Code

source coding
(compression)

barcode represents only
12 out of 13 symbols

information

coding:
map info to number

081547112369

text in clear in font OCR-B
(optical character recognition)

information written
from right to left

check digit

channel coding for error detection

Aztec Code

- 2-dimensional barcode
- ISO/IEC 24778
- 8 bit values (ANSI X3.4 = ASCII , ISO 8859-1 = Latin alphabet No. 1).
- 12 – 3832 numeric or 12 - 3067 alphabetic characters.
- Reed-Solomon error correcting code.
  - 32 error correction levels (5%-95%, recommended: 23%)
  - Error correction is possible for 25-40% destroyed code
- Well suited for displays of cell phones (e.g. e-ticket).

Applications:

12 numerical characters:
081547112369

Deutsche Bahn

Online Ticket

Handy Ticket

Compact Disc (CD) Digital Audio System

- Channel coding for protection against scratches and dirt on the CD surface (burst errors)

- Error correcting code:

  Cross-interleaved Reed-Solomon code with rate $R=3/4$ (25% redundancy).

Error correction capability:

- Error bursts of length 4000 bits can be corrected (2.5mm long scratch).
- Distributed errors with a bit error rate of up to 10^{-3} (on average 1 in 1000 bits is corrupted) can be corrected.

Error detection capability:

- Error bursts of length 12000 bits can be detected (7.5mm long scratch)

  The lost data is reconstructed using interpolation.

Wireless Communications

Transmission errors due to fading and low received power.

Bandwidth- and power-limited channel.

Channel coding:

- Global System for Mobile Communication (GSM):
  Convolutional code of memory 4 and rate $R=1/2$.

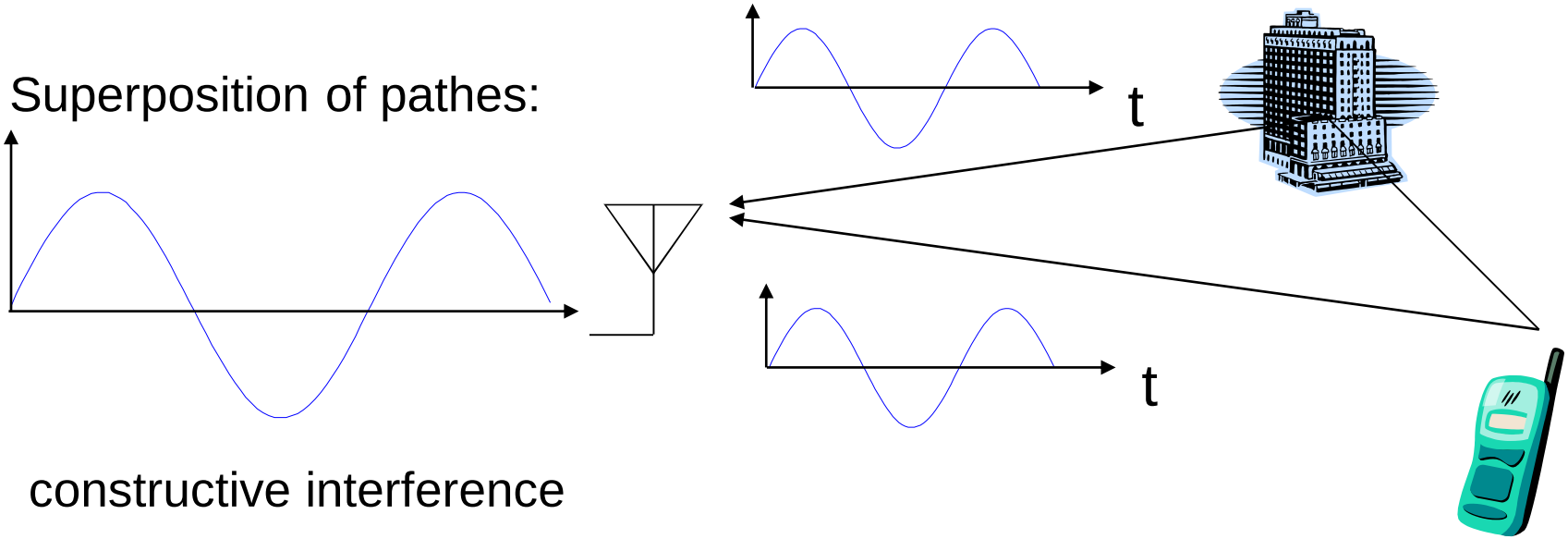
- Universal Mobile Telecommunications System (UMTS)
  - Convolutional code of memory 6 and rate $R=1/2$.
  - Turbo code with adaptive code rates $R=1/3 – 8/9$.

- 3GPP Long-Term Evolution (LTE)
  - Turbo code with adaptive code rates $R=1/6 – 12/13$.

Transmission Channels: Wireless Channel

multipath interference

Superposition of pathes:



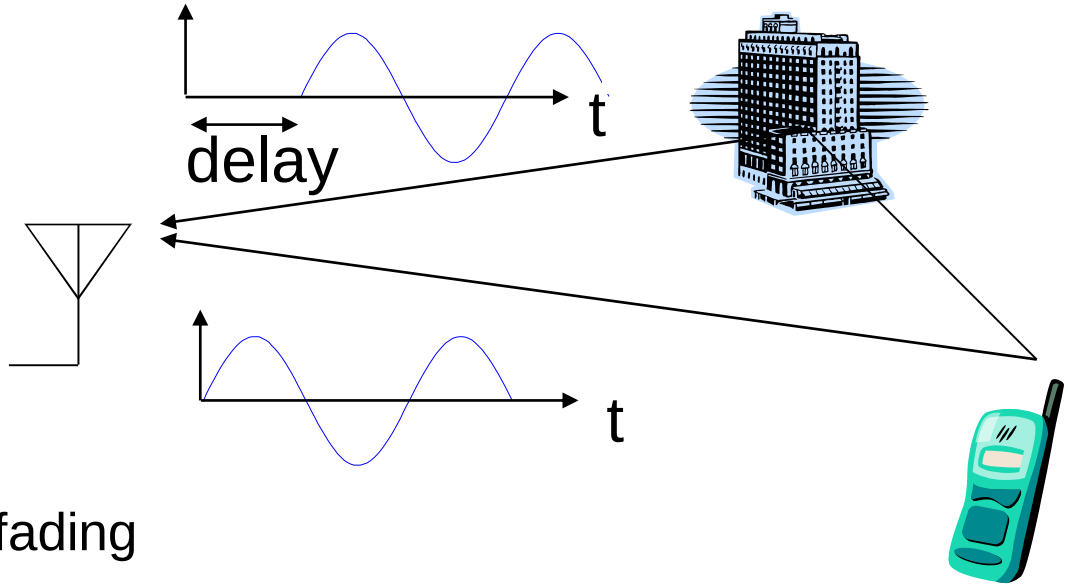
constructive interference

Transmission Channels: Wireless Channel

multipath interference

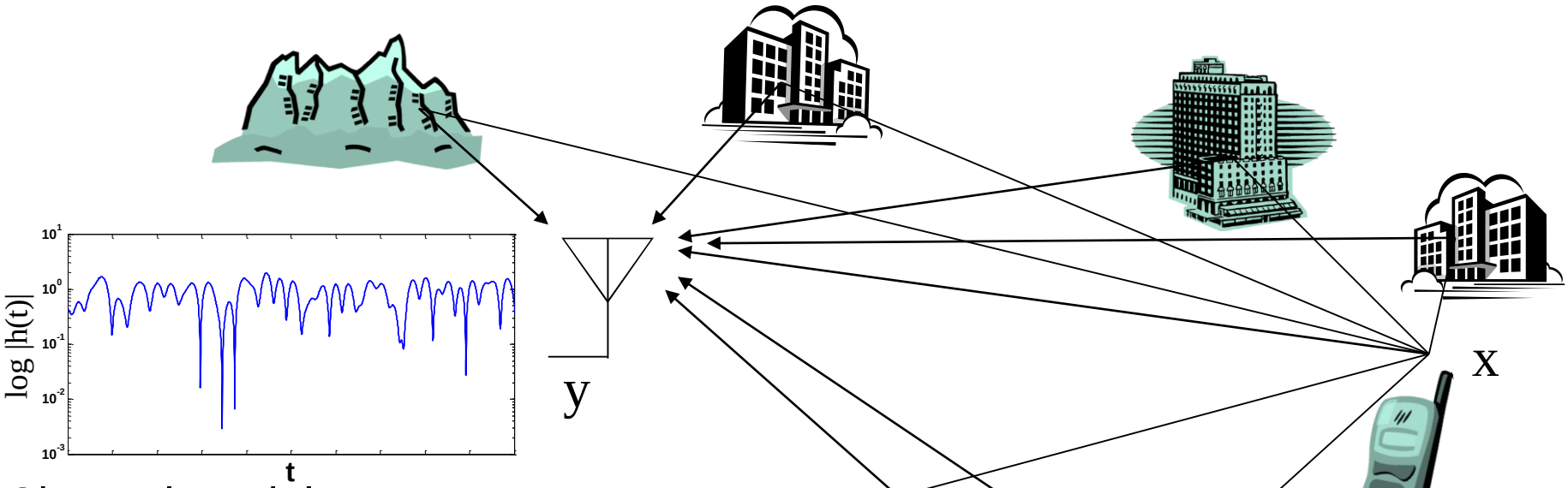
Superposition of pathes:

Fading !

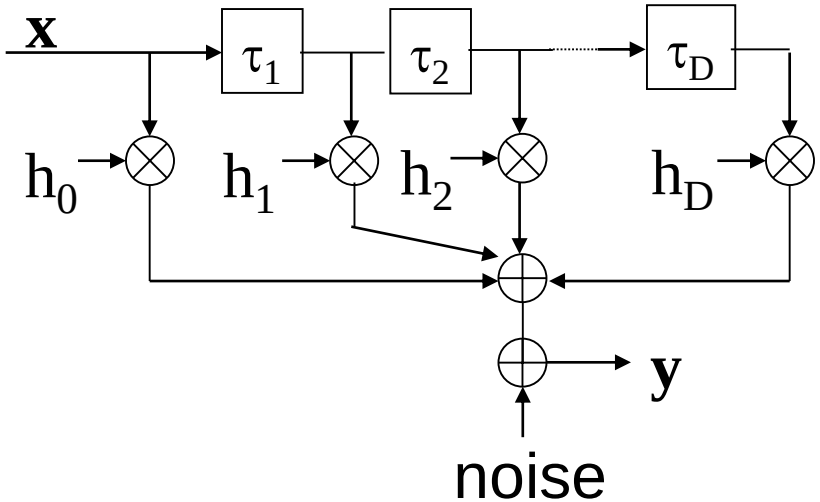


destructive interference $\Rightarrow$ fading

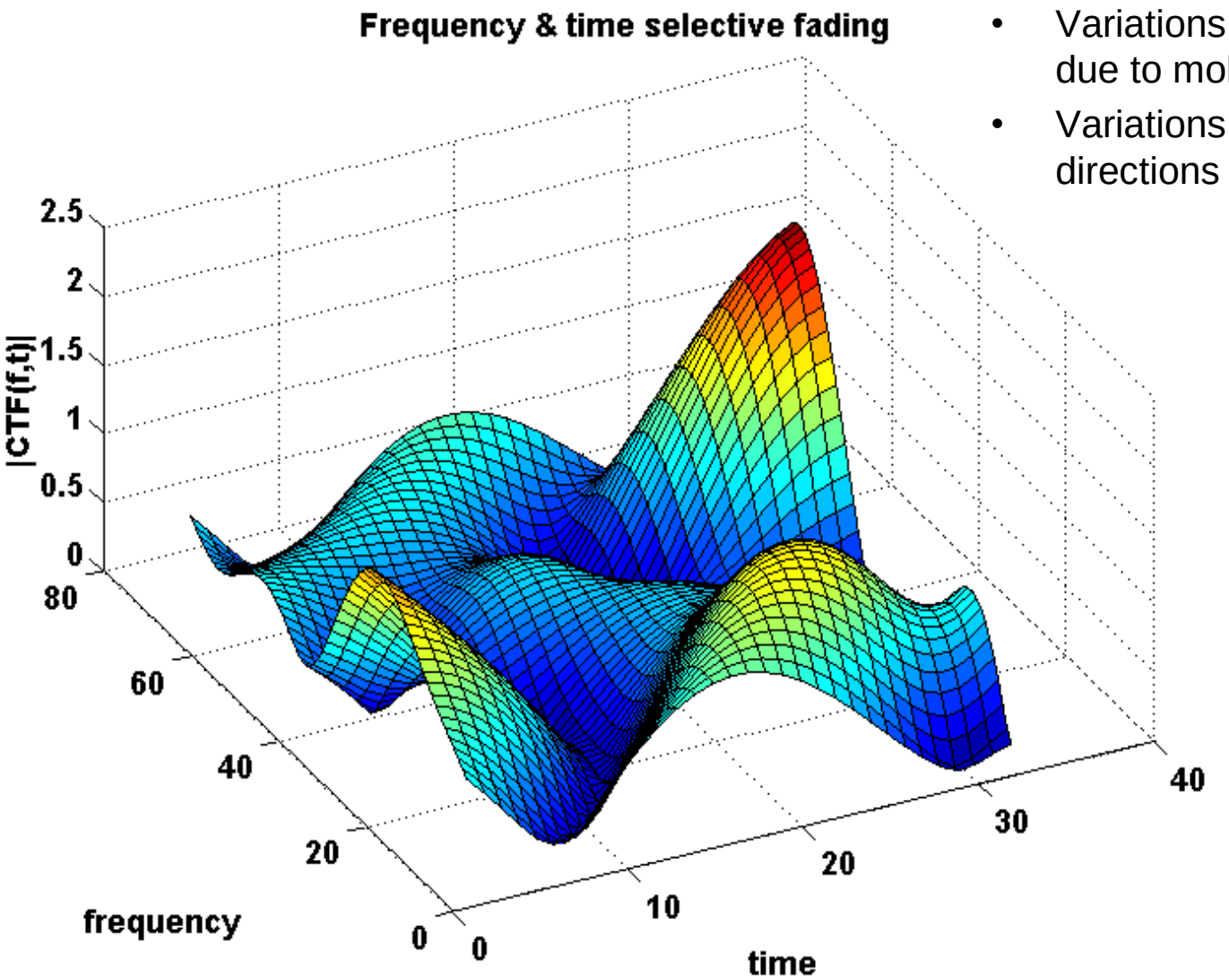
Transmission Channels: Wireless Channel



Channel model:



Wireless Channel Transfer Function



- Variations in time direction due to mobility
- Variations in frequency directions due to multi-path

Received SNR in Mobile Communications

UMTS: $B=5\ MHz,\ f_0=2\ GHz$

Noise power spectral density:

$$N_0 = FkT = 10^{9/10} \cdot 1.38 \cdot 10^{-23} \frac{Ws}{K} \cdot 294K = 3.22 \cdot 10^{-20} \frac{W}{Hz}$$

Noise figure: $F=9dB$ at terminal

Boltzmann constant: $k=1.38 \cdot 10^{-23}\ Ws/K$

Temperature: $T=294K$

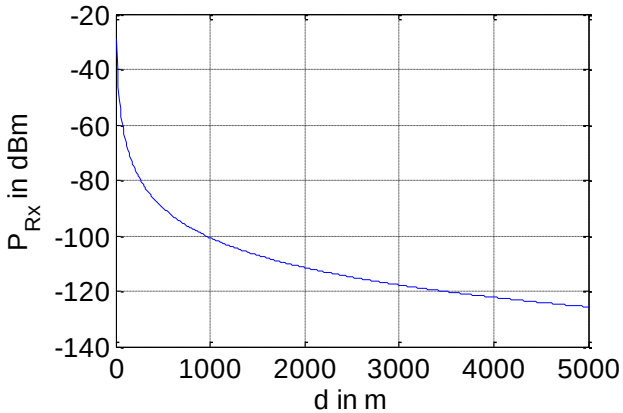
Noise power: $P_n = N_0 B = 3.22 \cdot 10^{-20} \frac{W}{Hz} \cdot 5 \cdot 10^6\ Hz = 1.6 \cdot 10^{-13} W \quad \Rightarrow 10\log_{10} \frac{P_n}{1mW} dBm = -97.96 dBm$

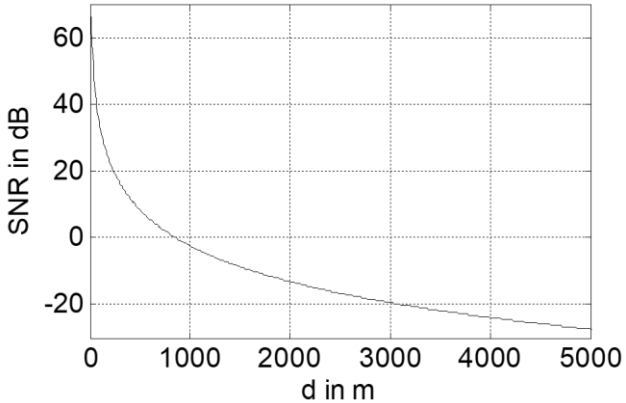
Transmit power: $P_T = 10W \quad \Rightarrow 10\log_{10} \frac{P_T}{1mW} dBm = 40 dBm$

Path-loss model from EU project WINNER:

Urban macro cell, non-line-of-sight (NLOS): $PL[dB] = 35.7\log_{10}(d[m]) + 42.6 + 23\log_{10} \frac{f_c[GHz]}{5}$

SNR at receiver: $SNR = 10\log_{10} \frac{P_T}{1mW} dBm - PL - 10\log_{10} \frac{P_n}{1mW} dBm$



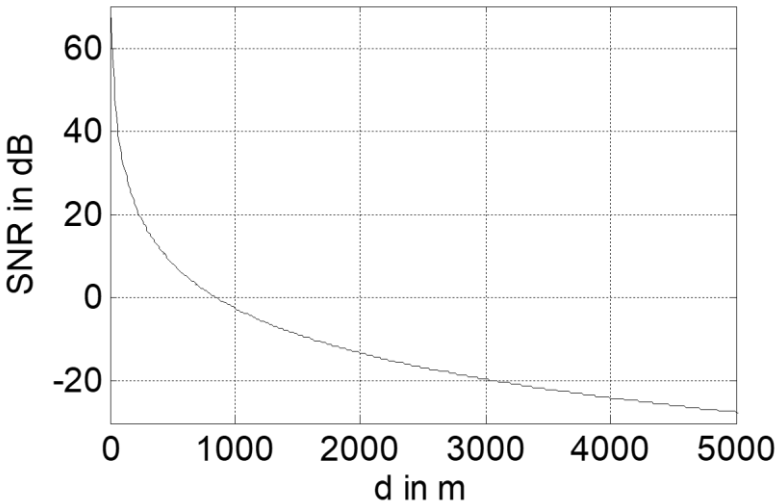


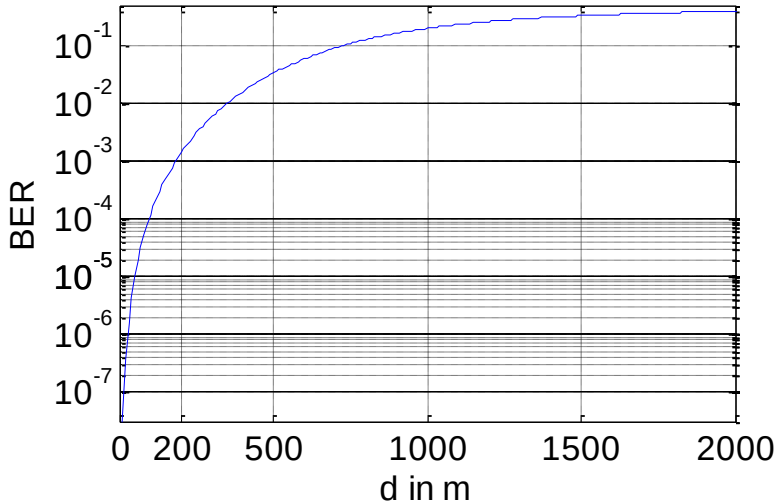
Bit Error Rate (BER) in Mobile Communications

UMTS: $B=5\ MHz,\ f_0=2\ GHz$

uncoded bit error rate (BER) for BPSK in Rayleigh fading channel

$$P_b = \frac{1}{2}\left(1 - \sqrt{\frac{\frac{P_{Rx}}{P_n}}{1+\frac{P_{Rx}}{P_n}}}\right)$$





Signal to Noise Power Ratio (SNR) in Decibel (dB)

Signal to noise power ratio:

$$SNR = \frac{\text{signal power}}{\text{noise power}} = \frac{P_x}{P_n}$$

The signal to noise power ratio is often given in the pseudo unit dB:

$$SNR \text{ in } dB = 10\log_{10}\frac{\text{signal power}}{\text{noise power}}\,dB = 10\log_{10}\frac{P_x}{P_n}\,dB$$

Examples:

SNR	*SNR* in *dB*
1	*0*
2	*3.01 ≈ 3*
4	*6.02 ≈ 6*
10	*10*
100	*20*

Space Communications

- E.g. Voyager space vehicle travelling to Neptun and Uranus
  Channel coding: Concatenation of Reed-Solomon code and convolutional code.

- Proposal to communicate from Mars to Earth with 5 W laser beam.
  → extremely reliable channel coding scheme will be required.

- Deep space channel is a power-limited AWGN channel with virtually unlimited bandwidth
  → Binary codes with binary modulation can be applied.

- Decoder complexity is not a major issue.

Information Theory

The term *information* is not well defined in habitual language use. In order to design and evaluate communications systems, we need a quantitative measure of information. This allows e.g. to analyze, how much information is generated by a source or how much information can be transmitted through a certain channel. *Information theory* provides such a mathematical quantification of information based on probability theory and statistics. Unlike most other scientific theories, the beginning of information theory can be attributed to a single paper entitled “A Mathematical Theory of Communication” which was published by Claude Elwood Shannon in 1948 in the Bell System Technical Journal. Shannon introduced communication as a statistical process. He followed up on earlier attempts to quantify information by Ralph Hartley and others. Most of this preliminary work looked at events of equal probability. Shannon defined *information* in a probabilistic sense as the *reduction of uncertainty about an event*. The *self-information* of an event x which occurs with probability $P(x)$ is defined as

$$I(x) = \log_2 \frac{1}{P(x)} = -\log_2 P(x).$$

Here, information is measured as the logarithm of the inverse of the probability that an event will happen. The unit of information is *bit (**bi**nary digi**t**)* if the logarithm to base 2 is used, *nat* if the natural logarithm is used instead and *hartley* if the common logarithm (base 10) is used.

From looking at special cases, it can be understood that this is a meaningful definition: The information conveyed in the observation of a likely event is small whereas the information conveyed in the observation of an event which is unlikely to happen is large. In a sense, self-information measures, how surprising the observation of an event is.

Information Theory

Shannon extended this definition of information by looking not only at single events, but to an information source which emits different symbols x_i from a certain alphabet which occur with different probabilities $P(x_i)$. The alphabet can be binary, i.e. $x_i \in \{0, 1\}$ or $x_i \in \pm 1$, or we can have an M-ary alphabet with M distinct symbols, e.g. an alphabet with $M=32$ characters $x_i \in \{a, b, c, d, ...\}$.

Shannon characterizes such a source by its mean self information, which he calls the *entropy* $H(X)$ of the source:

$$H(X) = E\{I(X)\} = \sum_{i=1}^{M} P(x) \log_2 \frac{1}{P(x)} = -\sum_{i=1}^{M} P(x) \log_2 P(x).$$

Entropy can be viewed as a quantification of the uncertainty in predicting the outcome of a random experiment. Entropy is the average number of bits which is at least needed in order to charaterize each possible outcome of a random experiment. For example, the random experiment *tossing a coin* has the two possible outcomes head or tail. One bit is required in order to characterize both possible outcomes. E.g. bit value 0 characterizes head and bit value 1 characterizes tail.

The random experiment *rolling a die* has six possible outcomes, i.e. six sides of the die. Hence, the uncertainty about the outcome of the experiment is higher than in case of tossing a coin. Consequently, more bits are required in order to represent each possible outcome of the random experiment.

For a fair die, each side occurs with probability $P(x_i) = 1/6$. Therefore, the entropy of the source *die* is given by

$$H(X) = -\sum_{i=1}^{M} P(x) \log_2 P(x) = -\sum_{i=1}^{6} \frac{1}{6} \log_2 \frac{1}{6} = 2.58 \text{ bit}.$$

Information Theory

We now try to represent each outcome of the random experiment by a sequence of bits called *codeword*, such that the average number of bits per side of the die is minimized. We start by allocating the same number of bits to each side. Since we cannot allocate a fraction of a bit for representation of a side, we have to use 3 bit per outcome as shown in the second column of the table below. This is obviously more than suggested by the entropy $H(X) = 2.58$ bit of the die.

side	codeword	codeword
1	*001*	*101*
2	*010*	*100*
3	*011*	*111*
4	*100*	*110*
5	*101*	*00*
6	*110*	*01*

An alternative representation is given in column 3 of the table above. Here, two codewords have only a length of two bits. Such a code which comprises codewords of different length is called a *variable length code*. The average codeword length is now given by 2.66 bit, which is closer but still slightly above the entropy of the source. Note, that the variable length code in column 3 is a so-called *prefix-free code*. I.e., no codeword appears as the first bits of another codeword. This allows to uniquely identify, where a codeword ends in a bit stream which represents the outcomes of repeated experiments..

Information Theory

The average codeword length remains longer than the entropy of the source, since we have a limited number of possible outcomes of the random experiment. Since each outcome has to be represented by uniquely decodable codeword, we have limited degrees of freedom to achieve an average codeword length which matches the entropy.

Therefore, we perform the random experiment multiple times and code each sequence of outcomes in a single codeword. I.e. the sequence of sides “1”, “6” is coded by a single codeword. By combining the outcome of two experiments to a hypersymbol (e.g. “1”, “6” $\rightarrow$ 16), we obtain 36 different hypersymbols. This allows to design a code with larger variation of the codeword length compared to the codes shown in the table above. Consequently, we can achieve an average codeword length, which is closer to the entropy of the source. The more successive outcomes of the experiment are combined to a hypersymbol, the closer the average codeword length approaches the entropy.

This is the principle of *data compression*, also called *source coding*: We combine a large number N of source symbols to a hypersymbol and code the hypersymbols by a variable length code. As the number N of source symbols which are coded together approaches infinity, the average codeword length will approach the entropy of the source. Note, that the entropy of the source has to be determined by averaging the self-information of the hypersymbols.

Since the compression factor depends on the number N of jointly coded source symbols, compression algorithms e.g. for text compression work the better, the longer the text is.

Information Theory

An important property of entropy is that it is maximized if all symbols are equally likely. This indicates, that the next symbol is most unpredictable for equally likely symbols. Since the entropy of a source becomes smaller, if the source symbols have unequal probability, the compression factor is the higher, the more the probabilities of the source symbols differ. The principle of source coding is to allocate short codewords to likely hypersymbols and long codewords to unlikely hypersymbols. This strategy will minimize the average codeword length.

E.g. in all languages, certain characters occur more frequently than others. E.g., the character e is more likely than z. Also, certain sequences of characters occur more often than other sequences: The sequence “the” is more likely then the sequence “eht.”

Based on this principle, a number of source coding algorithms has been developed. Among the most famous source coding algorithms are the Huffman algorithm and the Lempel-Ziv algorithm.

The Huffman algorithm achieves an average codeword length $E\{L\}$ which is at most one bit higher than the entropy of the source, i.e.

$$H(X) \leq E\{L\} \leq H(X)+1.$$

However, the Huffman algorithm requires accurate knowledge of the probabilities of the source symbols. In practice, this is sometimes difficult to obtain. Many applications such as video coding provide tables with probabilities which are obtained from exhaustive experiments. However, those tables may not accurately describe the data at hand and, therefore, achieve suboptimum compression.

Information Theory

The Lempel-Ziv algorithm does not require knowledge of the source symbol probabilities. Such compression algorithms which do not require knowledge of the statistics of the source symbols are called *universal source coding algorithms.*

Note, that in information theory, *information* has nothing to do with the meaning of a message but is a purely statistical definition. A warning of a life-threatening danger such as “shark close to beach” has no higher weight than a platitude such as “nice to see you.”

Source Coding Theorem

The source coding theorem is one of the major contributions in Shannon's 1948 paper "A mathematical theory of communication."

A source is considered to be any process that generates successive messages (or symbols).

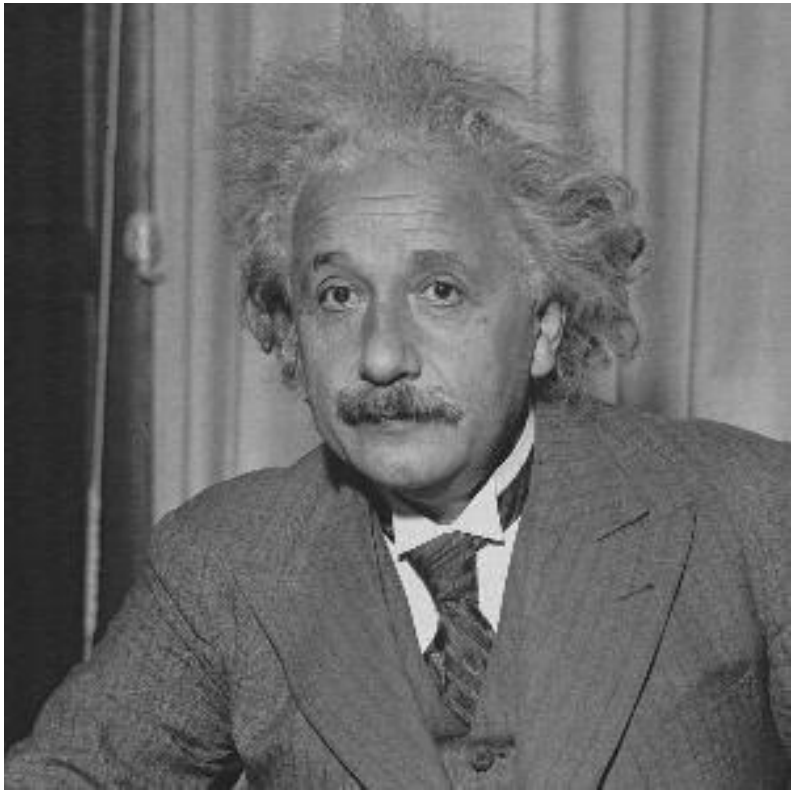
Shannon's source coding theorem states, that on average, the number of bits needed to represent a message is given by its entropy, given that the messages are statistically independent.

A particular mapping of bit sequences (codewords) to a certain message is called a source code. Unfortunately, Shannon's source coding theorem is just an existence proof. It states that it is possible to find a source code, the average codeword length of which approaches the entropy of the source. He does not provide an entropy achieving source coding algorithm. However, Shannon indicates that in order to design optimum source codes, we need to combine a large number of symbols and code them jointly as one message. Moreover, entropy is only achieved, if long message sequences are source coded.

Hence, the entropy is the asymptotic limit for lossless source coding (compression). In practice, we always have to deal with limited message size and limited sequences of messages. Therefore, apart from some special cases, there will remain a gap between the compression achieved in practice and the entropy. Examples for common source coding algorithms in practice comprise the *Huffman code* and the *Lempel-Ziv algorithm*.

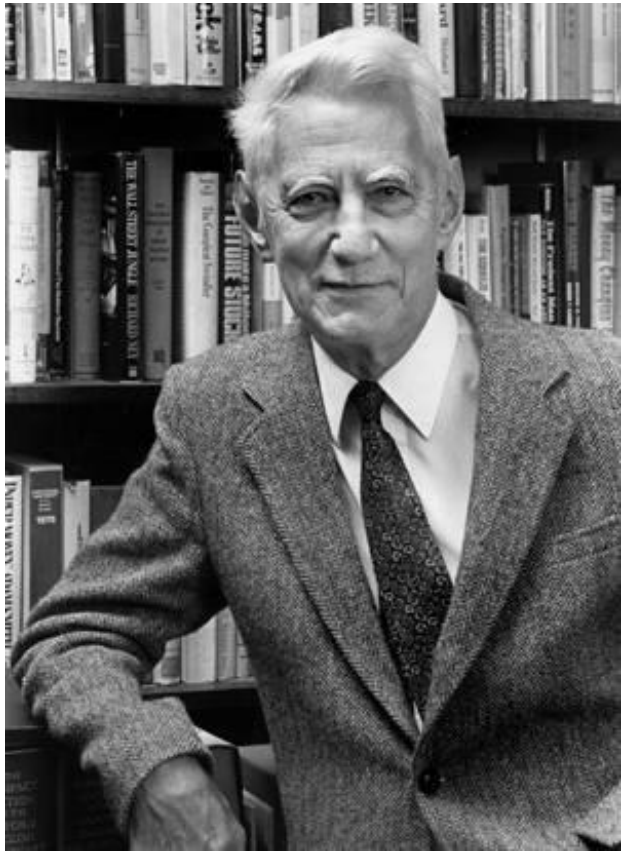
Nevertheless, knowledge of the theoretical limit is extremely useful in source coding design. It allows to evaluate how well a developed algorithm works and how much potential remains. This becomes increasingly important as source coding algorithms become more and more sophisticated and operate closer to the theoretical limit.

Albert Einstein
(14.3.1879-18.4.1955)



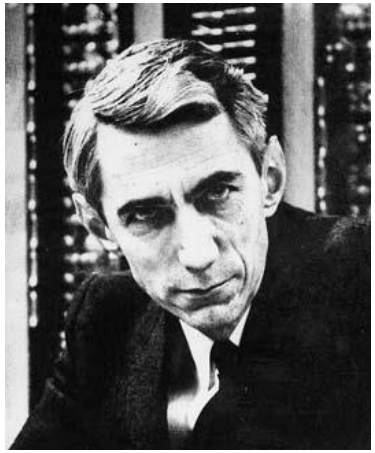
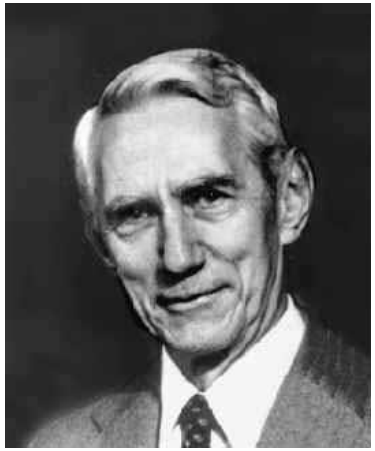
$$E = mc^2$$

Claude Elwood Shannon
(30.4.1916-26.2.2001)



$$C = \frac{1}{2}\log_2\left(1+\frac{2E_s}{N_0}\right)$$

Biography of Claude Elwood Shannon

- Born April 30, 1916
- Father businessman, mother German descendant
- 1932 BSc in El. Eng. and in Mathematics from University of Michigan
- 1936 Msc thesis MIT: *Boolean Algebra is Useful in Circuit Design*

  ”...one of the most important master‘s theses ever written... a landmark in that it helped to change digital circuit design from an art to a science.“

- 1940 PhD thesis: *An Algebra for Theoretical Genetics*
- 1940 almost failed in PhD exam (failed twice in German exam)

Biography of Claude Elwood Shannon

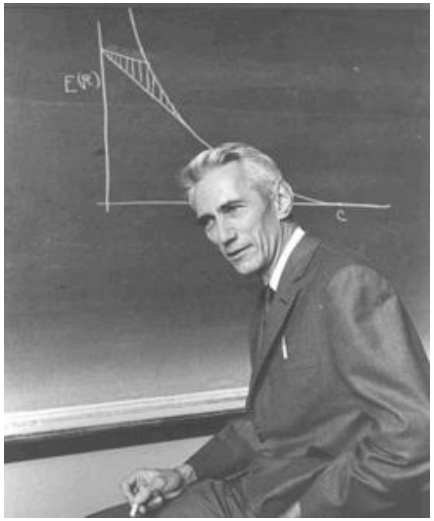
1948 paper:

A mathematical theory of communications.

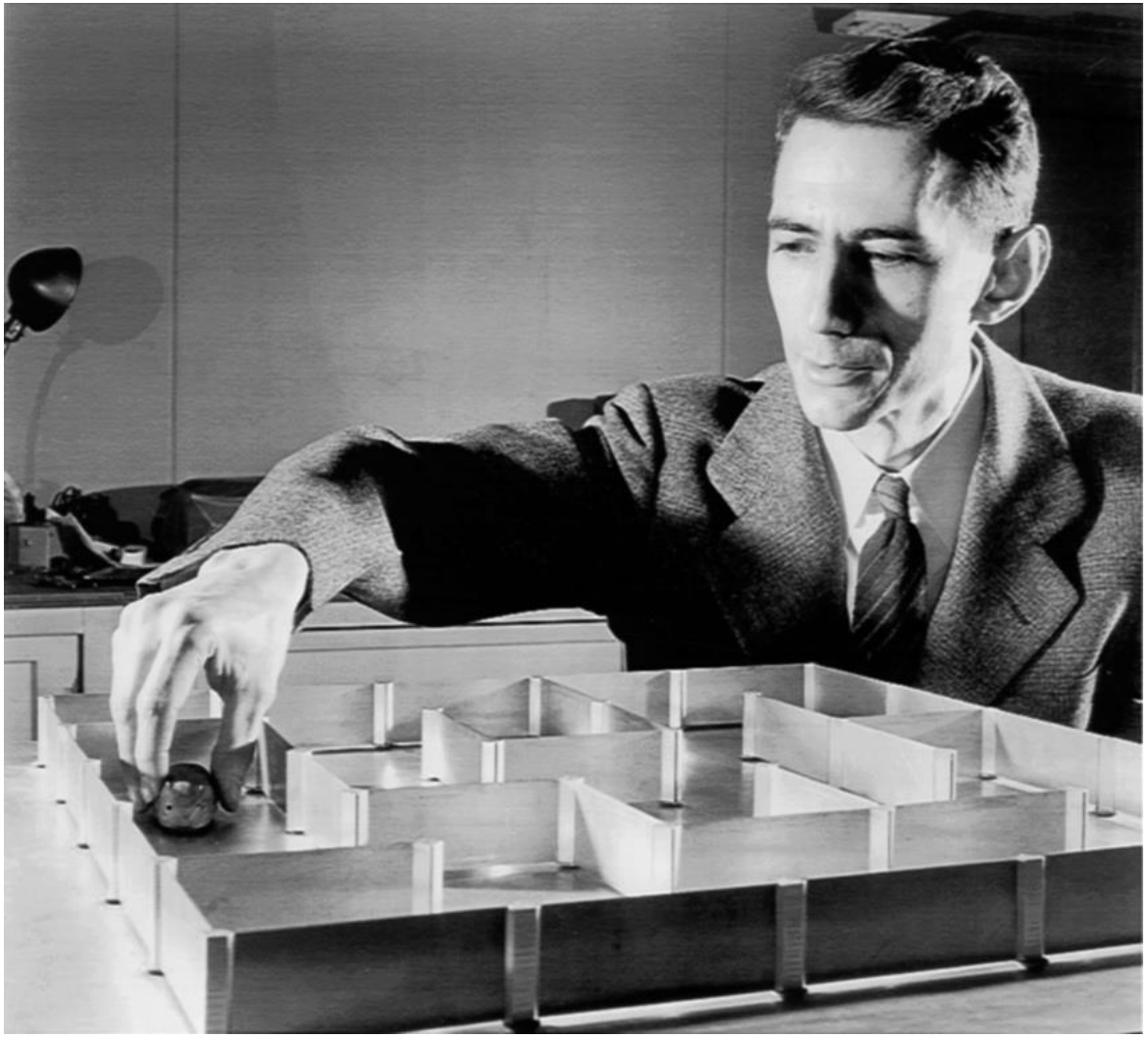
Beginning of information theory.

Biography of Claude Elwood Shannon

- 1941-1956 Bell Labs
- 1956-1966 Professor at MIT
- Some further inventions and activities by Shannon:
  - chess playing computer
  - three ball juggling machine
  - maze-solving mouse „Theseus“ with 100 relays
  - mind reading machine
  - riding unicycle
- lots of awards including German Eduard Rhein Prize
- died 26.2.2001

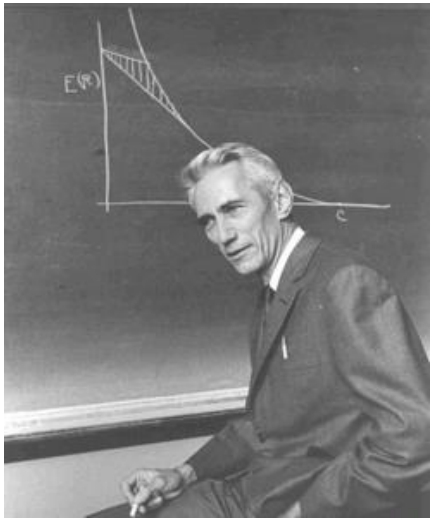


Shannon with maze-solving mouse „Theseus“



Biography of Claude Elwood Shannon

- 1941-1956 Bell Labs
- 1956-1966 Professor at MIT
- Some further inventions and activities by Shannon:
  - chess playing computer
  - three ball juggling machine
  - maze-solving mouse „Theseus“ with 100 relays
  - mind reading machine
  - riding unicycle
- lots of awards including German Eduard Rhein Prize
- died 26.2.2001

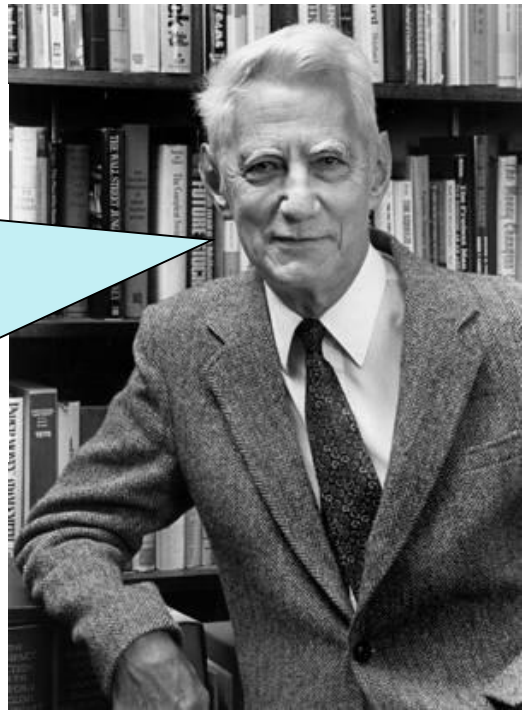


Claude Elwood Shannon (30.4.1916-26.2.2001)
Father of Information Theory

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

″Frequently the messages have meaning″

″... [which is] irrelevant to the engineering problem.″



Shannon‘s Contributions

- Theory and limits of source coding.
- Theory of channel coding and limits of information transmission.
- Separation theorem: source coding and channel coding can be treated separately.
- Definition and limits of cryptography

Definition of Information

X: Random variable with discrete realizations x_i, $i=1,\ldots,I$ and probability $P(x_i)$

Self information of event $X=x_i$:

$$I(x_i) = \log_2 \frac{1}{P(x_i)} \quad \text{in bit}$$

$P(x_i) \to 0$: $I(x_i) \to \infty$

$P(x_i) = 1$: $I(x_i) = 0$

$P(x_i) = ½$: $I(x_i) = 1$ bit

Entropy: Expected self information:

$$H(X) = E\{I(x_i)\} = \sum_{i=1}^{I} P(x_i) \log_2 \frac{1}{P(x_i)}$$

Self Information and Entropy
Example: Tossing a Coin

Self information of an event x_i with probability $P(x_i)$:

$$I(x_i) = \log_2 \frac{1}{P(x_i)} \quad \text{in bit}$$

2 possible events:

x_1 = „Tail“ $\quad P(x_1)=1/2 \Rightarrow I(x_1) = \log_2(2) = 1$ bit

x_2 = „Head“ $\quad P(x_2)=1/2 \Rightarrow I(x_2) = \log_2(2) = 1$ bit

Expected self information („entropy“):

$$H(X) = E\{I(x_i)\} = \sum_{i=1}^{2} P(x_i) I(x_i) = P(x_1) I(x_1) + P(x_2) I(x_2)$$

$$= \frac{1}{2} \cdot 1\text{bit} + \frac{1}{2} \cdot 1\text{bit} = 1\text{bit}$$

Self Information - Example: Oracle

Delphic Sibyl by Michelangelo in the Sisitine Chapel

„Will I become older than 100 years ?“

2 possible events (answers of oracle):

- „yes“

  $P(\text{yes}) = 0.01$

  $\Rightarrow I(\text{yes}) = 6.644 \text{ bit}$

  Unexpected reply → high self information (high information gain from reply of oracle)

- „no“

  $P(\text{no}) = 0.99$

  $\Rightarrow I(\text{no}) = 0.0145 \text{ bit}$

  Expected reply → low self information

Entropy of source „oracle“:

$$H(X) = E\{I(x_i)\} = \sum_{i=1}^{2} P(x_i) I(x_i) = P(x_1) I(x_1) + P(x_2) I(x_2)$$

$$= -0.01 \cdot \log_2(0.01)\,\text{bit} - 0.99 \cdot \log_2(0.99)\,\text{bit} = 0.08\,\text{bit}$$

Self Information and Entropy
Example: Die

- Fair die:

$$P(x_i) = \frac{1}{6} \quad \Rightarrow \quad I(x_i) = \log_2(6); \quad i = 1,\ldots,6$$

$$H_{\text{fair}}(X) = 6\frac{1}{6}\log_2(6) = 2.58 \text{ bit}$$

- Loaded die:

$$P(x_i) = \frac{1}{12} \quad \Rightarrow \quad I(x_i) = \log_2(12); \quad i = 1,\ldots,5$$

$$P(x_6) = \frac{7}{12} \quad \Rightarrow \quad I(x_6) = \log_2\left(\frac{12}{7}\right)$$

$$H_{\text{loaded}}(X) = 5\frac{1}{12}\log_2(12) + \frac{7}{12}\log_2\left(\frac{12}{7}\right) = 1.95 \text{ bit}$$

Information advantage of swindler (his reduction of uncertainty):

$$H_{\text{fair}}(X) - H_{\text{loaded}}(X) = 2.58 \text{ bit} - 1.95 \text{ bit} = 0.63 \text{ bit (redundancy)}$$

Entropy of a Source

An information source emits symbols from an I–ary alphabet (I distinct symbols) x_i, $i=1,\ldots,I$, with probabilities $P(x_i)$:

$$\boxed{\text{source}} \longrightarrow x_i\,, \quad i=1,\ldots,I$$

Self information of the event x_i with probability $P(x_i)$:

$$I(x_i) = \log_2 \frac{1}{P(x_i)} \quad \text{in bit}$$

Entropy of the source: Average self information of a symbol,
expected self information

$$H(x) = E\{I(x_i)\} = \sum_{i=1}^{I} P(x_i) \log_2 \frac{1}{P(x_i)}$$

Binary Entropy Function

A binary source emits the symbol „0“ with probability P_0 and the symbol „1“ with probability $1\text{-}P_0$.

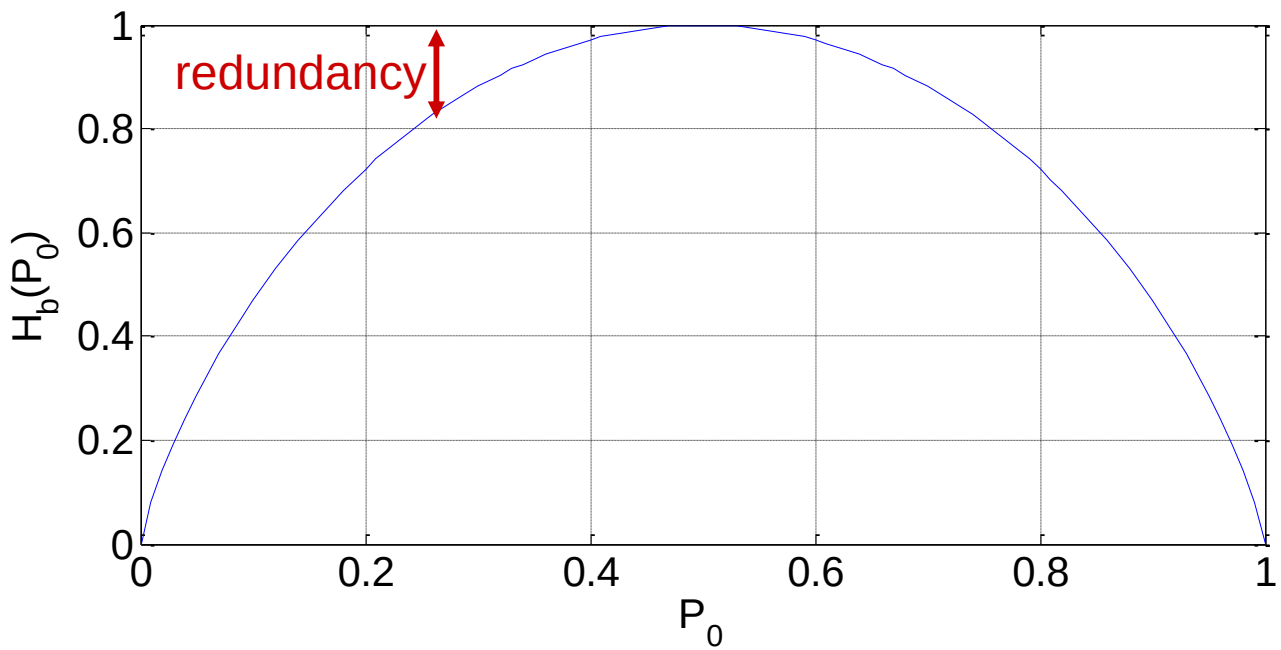
source → $0, 0, 1, 0, 1, 1, 1...$

$x_1=0\ ,\ \ P(x_1)=P_0$

$x_2=1\ ,\ \ P(x_2)=1\text{-}P_0$

Entropy of the source:

$$H_b(P_0) = -\,P_0 \log_2 P_0 - (1\text{-}P_0) \log_2(1\text{-}P_0) \quad \text{(binary entropy function)}$$



Shannon‘s Source Coding Theorem

The average number of bits which is required in order to represent a discrete symbol emitted by an information source is given by the entropy $H(X)$ of the source, given that the source symbols X are statistically independent.

The entropy determines the number of bits, which is on average required for coding of the source symbols. I.e., the entropy is the theoretical limit of data compression, which can be achieved by a lossless compression algorithm.

Examples for source coding (compression algorithms):

- Huffman code
- Lempel-Ziv algorithm

Entropy of Continuous Random Variables: Differential Entropy

The differential entropy of a continuous random variable X with probability density function $f_X(x)$ and support set S is defined as

$$h(X) = \int_S f_X(x) \log_2 \frac{1}{f_X(x)} dx$$

The support set S of a random variable X is the set, where $f_X(x) > 0$.

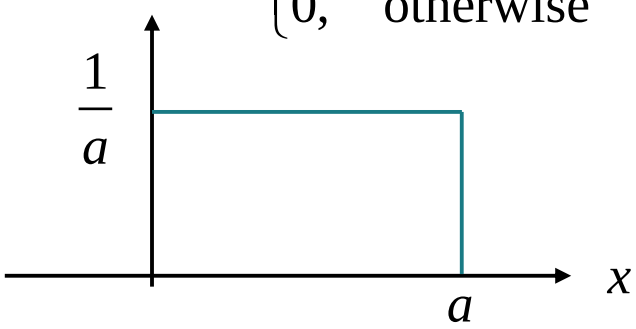
Entropy of Continuous Random Variables: Differential Entropy

Example: Uniform Distribution

The differential entropy of a continuous random variable X with probability density function $f_X(x)$ and support set S is defined as

$$h(X) = \int_S f_X(x) \log_2 \frac{1}{f_X(x)} dx$$

$$f_X(x) = \begin{cases} \frac{1}{a}, & \text{for } 0 \le x \le a \\ 0, & \text{otherwise} \end{cases}$$



$$h(X) = \int_0^a \frac{1}{a} \log_2 a \, dx = \frac{1}{a} \log_2(a) \left[x \right]_0^a$$

$$= \log_2 a$$

→ Unlike the entropy $H(X)$ for discrete random variables, the differential entropy can become negative (for $a<1$).

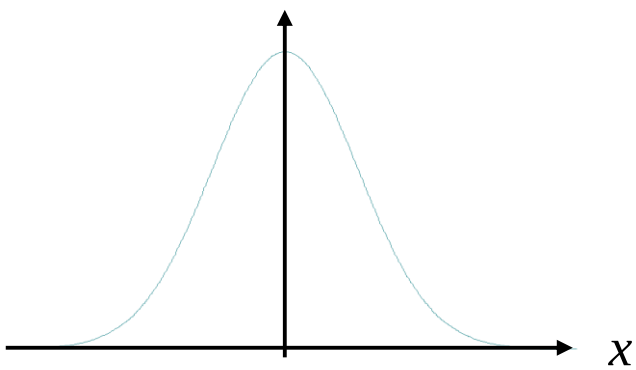
Entropy of Continuous Random Variables: Differential Entropy

Example: Normal Distribution $N(\mu, \sigma^2)$

The differential entropy of a continuous random variable X with probability density function $f_X(x)$ and support set S is defined as

$$h(X) = \int_S f_X(x) \log_2 \frac{1}{f_X(x)} dx$$

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



$$h(X) = \frac{1}{2} \log_2 \left(2\pi e \sigma^2\right)$$

Among all possible probability distributions with a given variance σ^2, the normal distribution $N(\mu, \sigma^2)$ maximizes the differential entropy.

Derivation: Differential Entropy of Normal Distribution

Normal Distribution $N(\mu, \sigma^2)$: $f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$$h(X) = \int_{-\infty}^{\infty} f_X(x) \log_2 \frac{1}{f_X(x)} dx = -\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \log_2 \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) dx$$

$$= -\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left[-\frac{1}{2} \log_2 \left(2\pi\sigma^2\right) - \log_2(e) \frac{(x-\mu)^2}{2\sigma^2} \right] dx$$

$$= \frac{1}{2} \log_2 \left(2\pi\sigma^2\right) \underbrace{\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{1} + \frac{\log_2(e)}{2\sigma^2} \underbrace{\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} (x-\mu)^2 dx}_{\sigma^2}$$

$$= \frac{1}{2} \log_2 \left(2\pi\sigma^2\right) + \frac{1}{2} \log_2(e) = \frac{1}{2} \log_2 \left(2\pi e \sigma^2\right)$$

Proof: Normal Distribution Maximizes the Differential Entropy (1)

Normal Distribution $N(\mu, \sigma^2)$:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Any probability density function with same variance σ^2:

$g_X(x)$

Relative entropy:

$$0 \le D(g \| f) = \int_{-\infty}^{\infty} g_X(x) \log_2 \frac{g_X(x)}{f_X(x)} dx$$

$$= \underbrace{\int_{-\infty}^{\infty} g_X(x) \log_2 g_X(x) dx}_{-h_g(X)} + \underbrace{\int_{-\infty}^{\infty} g_X(x) \log_2 \frac{1}{f_X(x)} dx}_{\int_S f_X(x) \log_2 \frac{1}{f_X(x)} dx \; = h_f(X)} = -h_g(X) + h_f(X)$$

(see next slides)

Proof: Normal Distribution Maximizes the Differential Entropy (2)

$$\underbrace{D\left(g \| f\right)}_{\geq 0} + h_g\left(X\right) = h_f\left(X\right) = \frac{1}{2}\log_2\left(2\pi e\sigma^2\right)$$

since $f_X(x)$ is a normal distribution

$$\Downarrow$$

$$h_g\left(X\right) \leq h_f\left(X\right) = \frac{1}{2}\log_2\left(2\pi e\sigma^2\right)$$

Proof: Normal Distribution Maximizes the Differential Entropy (3)

Proof for $\int_{-\infty}^{\infty} g_X(x)\log_2 \frac{1}{f_X(x)} dx = \int_{-\infty}^{\infty} f_X(x)\log_2 \frac{1}{f_X(x)} dx = \frac{1}{2}\log_2\left(2\pi e\sigma^2\right)$

The differential entropy $h(X) = \int_{-\infty}^{\infty} g_X(x)\log_2 \frac{1}{g_X(x)} dx$ is independent of the event space of the random variable X but depends only on the probability distribution $g_X(x)$.

→ Two random variables, X and Y, the probability distributions of which differ only in the expectation value, have the same differential entropy.

Example:

$Y = X + c$, where c is a constant $\Rightarrow E\{Y\} = \mu_Y = E\{X\} + c = \mu_X + c$

But: $h(X) = h(Y)$

→ In the proof, we can consider, that $g_X(x)$ results in the same expectation value as $f_X(x)$, i.e. $E_g\{X\} = E_f\{X\} = \mu$.

Proof: Normal Distribution Maximizes the Differential Entropy (4)

$$\int_{-\infty}^{\infty} g_X(x)\log_2\frac{1}{f_X(x)}dx \underset{\uparrow}{=} -\int_{-\infty}^{\infty} g_X(x)\log_2\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right)dx$$

$f_X(x)$ is normal distribution

$$= -\int_{-\infty}^{\infty} g_X(x)\left[-\frac{1}{2}\log_2\left(2\pi\sigma^2\right)-\log_2(e)\frac{(x-\mu)^2}{2\sigma^2}\right]dx$$

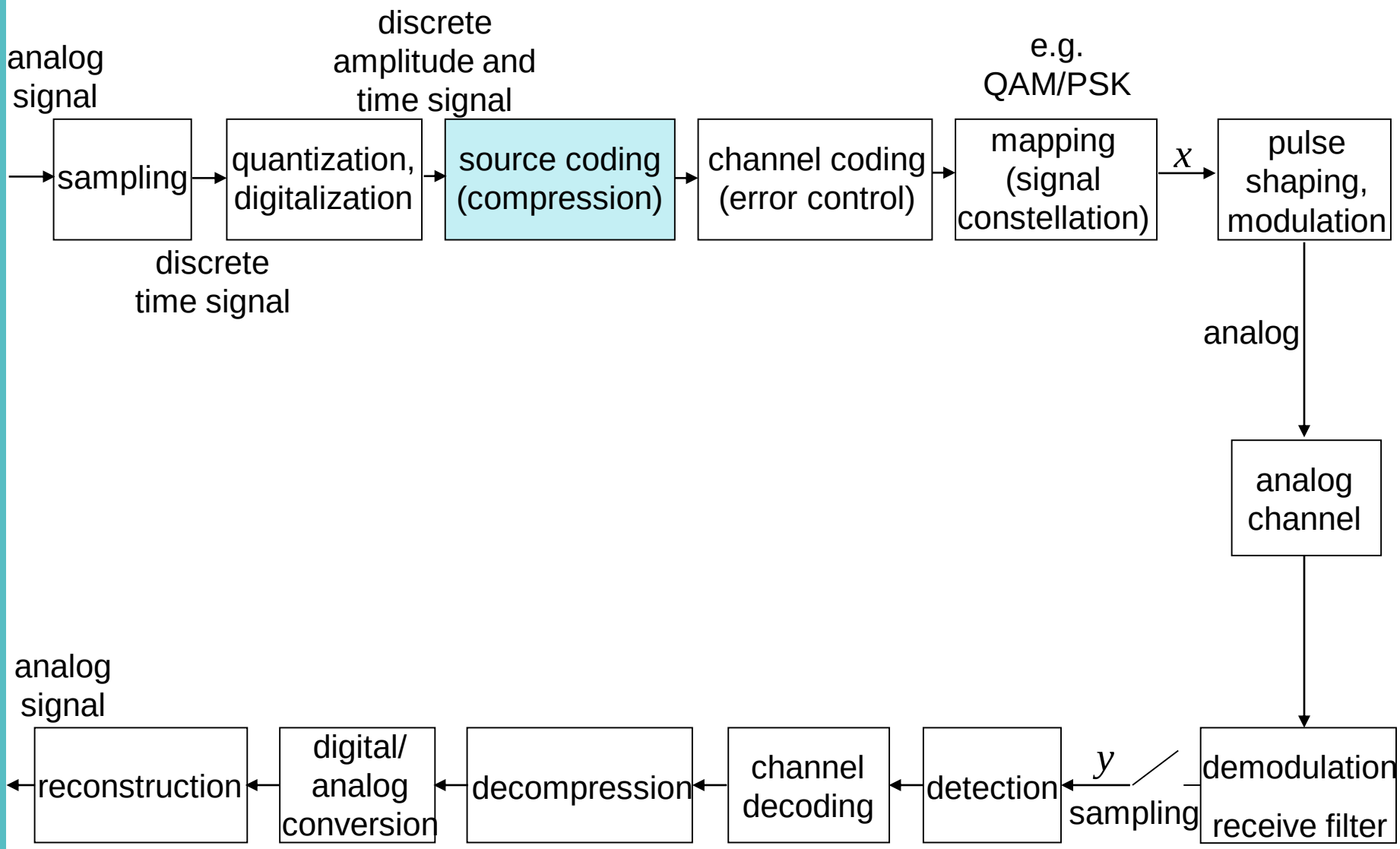
$$= \frac{1}{2}\log_2\left(2\pi\sigma^2\right)\underbrace{\int_{-\infty}^{\infty} g_X(x)dx}_{1}+\frac{\log_2(e)}{2\sigma^2}\underbrace{\int_{-\infty}^{\infty} g_X(x)(x-\mu)^2 dx}_{\sigma^2}$$

σ^2 (since $g_X(x)$ and $f_X(x)$ result in the same expectation value μ

$$= \frac{1}{2}\log_2\left(2\pi\sigma^2\right)+\frac{1}{2}\log_2(e) = \frac{1}{2}\log_2\left(2\pi e\sigma^2\right) \underset{\uparrow}{=} h_f(X)$$

entropy of Gaussian random variable

Digital Communications System



Source Coding (1)

(strings of) source symbols $\mathbf{x}$ — D-ary alphabet

$\xrightarrow{\text{mapping}}$

codewords $\mathbf{c}(\mathbf{x})$ — C-ary alphabet

source symbol x	$P(x)$
a	$1/2$
b	$1/4$
c	$1/8$
d	$1/8$

4-ary alphabet

codeword $\mathbf{c}$	codeword length $L(\mathbf{c})$	$P(\mathbf{c})$
0	1 bit	$1/2$
10	2 bit	$1/4$
110	3 bit	$1/8$
111	3 bit	$1/8$

binary alphabet

Entropy:

$$H(X) = 1.75 \text{ bit/symbol}$$

$=$

Average codeword length:

$$E\{L\} = \sum_{\mathbf{c}} P(\mathbf{c}) L(\mathbf{c}) = 1.75 \text{bit/symbol}$$

Source Coding (2)

(strings of) source symbols $\mathbf{x}$	mapping →	codewords $\mathbf{c}(\mathbf{x})$
D-ary alphabet		C-ary alphabet

source symbol x	$P(x)$
a	$1/2$
b	$1/4$
c	$1/8$
d	$1/8$

4-ary alphabet

codeword $\mathbf{c}$	codeword length $L(\mathbf{c})$	$P(\mathbf{c})$
00	2 bit	$1/2$
01	2 bit	$1/4$
10	2 bit	$1/8$
11	2 bit	$1/8$

binary alphabet

Entropy:

$H(X) = 1.75$ bit/symbol

$<$

Average codeword length:

$$E\{L\} = \sum_{\mathbf{c}} P(\mathbf{c}) L(\mathbf{c}) = 2\text{bit/symbol}$$

Optimal Source Codes

A source code is optimal, if it achieves the minimum possible average codeword length

$$E\{L\} = \min_{L(\mathbf{c}(x))} \sum_{x} P_X\left(\mathbf{c}(x)\right) L\left(\mathbf{c}(x)\right)$$

for a given probability distribution $P_X(x)$.

There exist many optimal codes. The Huffman algorithm is one example for construction of optimal codes.

Prefix Codes, Prefix-Free Codes, Instantaneous Codes

A code is called a *prefix code* or a *prefix-free code* or an *instantaneous code*, if no codeword is the prefix of another codeword.

A prefix code is uniquely decodable without reference to future codewords.

Prefix Codes, Prefix-Free Codes, Instantaneous Codes: Example

A code is called a *prefix code* or a *prefix-free code* or an *instantaneous code*, if no codeword is the prefix of another codeword.

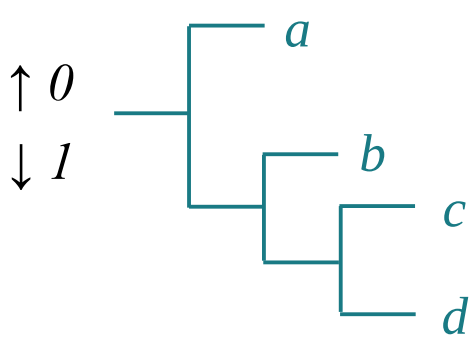
x	$\mathbf{c}(x)$
a	0
b	10
c	110
d	111

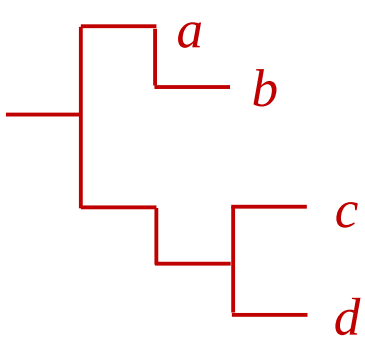
→ prefix-free

x	$\mathbf{c}(x)$
a	0
b	01
c	110
d	111

$\mathbf{c}(a)$ is prefix of $\mathbf{c}(b)$

→ not prefix-free





Decoding of Prefix Codes, Prefix-Free Codes, Instantaneous Codes

A code is called a *prefix code* or a *prefix-free code* or an *instantaneous code*, if no codeword is the prefix of another codeword.

x	$\mathbf{c}(x)$
a	0
b	10
c	110
d	111

→ prefix-free

x	$\mathbf{c}(x)$
a	0
b	01
c	110
d	111

→ not prefix-free

Source symbol sequence: $a \quad b \quad c$

Coded sequence: $0 \,|\, 10 \,|\, 110$

$0 \,|\, 0 \,|\, 1 \,|\, 11 \,|\, 0$

Decoding

$a \quad b \quad c$

$a \quad b \quad c$ or $a \quad a \quad d \quad a$

Source Coding: International Morse Code

(strings of) source symbols $\mathbf{x}$ — D-ary alphabet

mapping →

codewords $\mathbf{c}(\mathbf{x})$ — C-ary alphabet

source symbol x	$P(x)$
␣	0.1918
E	0.1041
T	0.0729
...	

codeword $\mathbf{c}$	codeword length $L(\mathbf{c})$
· · or 1100	4 bit
— · · or 011100	6 bit
· — · · or 1011100	7 bit
...	...

27-ary alphabet
(26 characters and space)

5-ary alphabet
{dot, dash, symbol- ,letter- and word gap}

Entropy:

$$H(X) = 4.07\,\text{bit/symbol}$$

<

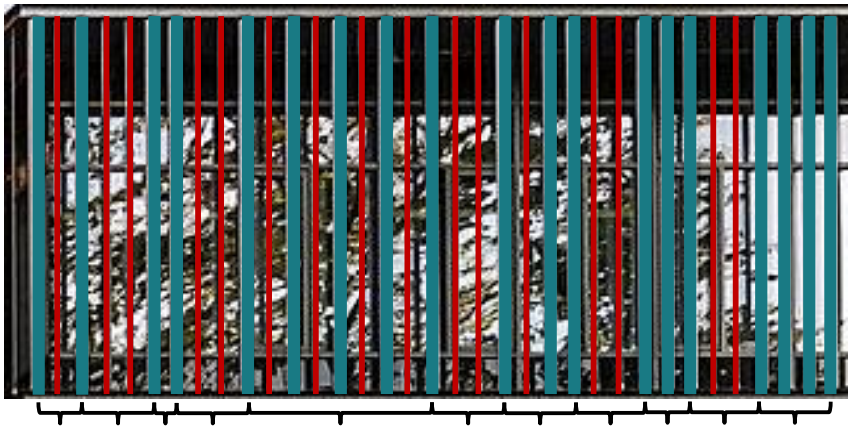
Average codeword length:

$$E\{L\} = \sum_{\mathbf{c}} P(\mathbf{c}) L(\mathbf{c}) = 4.84\,\text{bit/symbol}$$

TUHH Main Building

Morse Code[1]

T	–
E	.
CH	– – – –
N	– .
I	. .
S	. . .



T ␣ E ␣ CH ␣ N ␣ I ␣ SCHE UNIVERSITÄT HAMBURG-HARBURG

Morse Code translated to distance between bars:

. = single bar-distance

– = double bar-distance

Pause = triple bar-distance

[1]Continental version by Friedrich Gerke, 1848, Hamburg

Text Compression: Examples

character	Probability %	Bacon 1623 Baudot 1874	Morse 1844	Huffman 1952
␣	19.18	00100	00	000
E	10.41	10000	100	001
N	7.29	00110	01100	010
S	6.52	10100	11100	0110
I	5.96	01100	1100	0111
R	5.65	01010	101100	1000
⋮				
M	2.17	00111	010100	111010
⋮				
X	0.08	10111	01110100	1111111110
E{L}		**5**	**4.84**	**4.13**

Source Coding: Text Compression

26 characters + special characters = 32 symbols

- Equally likely symbols: $P(x_i)=1/32$
  $H(x) = 32 \cdot (1/32) \log_2 32 = 5$ bit/symbol

- English language: 1 bit/symbol
  (Shannon)

Claude Elwood Shannon (1916-2001)
American mathematician and electrical engineer. Founder of information theory.

- German language: $H(x) = 1.3$ bit/symbol
  (Küpfmüller)

Karl Küpfmüller (1897-1977)
German electrical engineer. Contributed to communications engineering, measurement engineering, control engineering, acoustics, information theory, electromagnetic theory.

- With state-of-the-art compression algorithms, texts can be compressed by a factor 2-3.

Huffman Code

The Huffman algorithm provides a simple method for construction of a prefix-free source code. The Huffman code is optimum in the sense that no other code can have a shorter average codeword length given a certain source symbol probability distribution.

Huffman Code

symbol x	probability $P(x)$	codeword
A	0.1	010
B	0.2	10
C	0.2	11
D	0.1	011
E	0.4	00

Huffman algorithm:

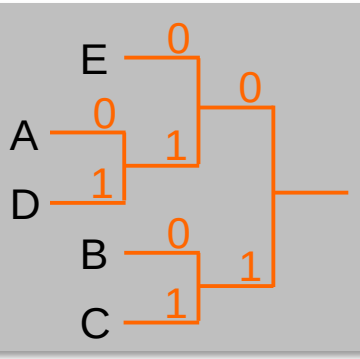
1. In each step, sort the symbols according to their probabilities in descending order.
2. Combine the two symbols with lowest probabilities to a new hypersymbol.
3. Code the decision between the two most unlikely symbols with one bit.

symbol	$P(x)$										
E	0.4		E	0.4		B,C	0.4		A,D,E	0.6	0
B	0.2		A,D	0.2		E	0.4	0	B,C	0.4	1
C	0.2		B	0.2	0	A,D	0.2	1			
A	0.1	0	C	0.2	1						
D	0.1	1									

Alternative construction:

- Allocation of bits „0“ und „1“ is arbitrary.
- Order of sorting for equally likely symbols is arbitrary.

Huffman tree:



Huffman Code

symbol x	probability $P(x)$	codeword
A	0.1	101
B	0.2	01
C	0.2	00
D	0.1	100
E	0.4	11

Huffman algorithm:

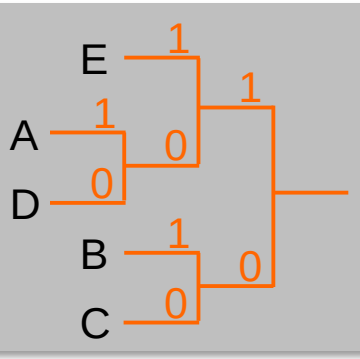
1. In each step, sort the symbols according to their probabilities in descending order.
2. Combine the two symbols with lowest probabilities to a new hypersymbol.
3. Code the decision between the two most unlikely symbols with one bit.

symbol	$P(x)$										
E	0.4		E	0.4		B,C	0.4		A,D,E	0.6	1
B	0.2		A,D	0.2		E	0.4	1	B,C	0.4	0
C	0.2		B	0.2	1	A,D	0.2	0			
A	0.1	1	C	0.2	0						
D	0.1	0									

Alternative construction:

- Allocation of bits „0“ und „1“ is arbitrary.
- Order of sorting for equally likely symbols is arbitrary.

Huffman tree:



Huffman Code

symbol x	probability $P(x)$	codeword
A	0.1	0010
B	0.2	01
C	0.2	000
D	0.1	0011
E	0.4	1

Huffman algorithm:

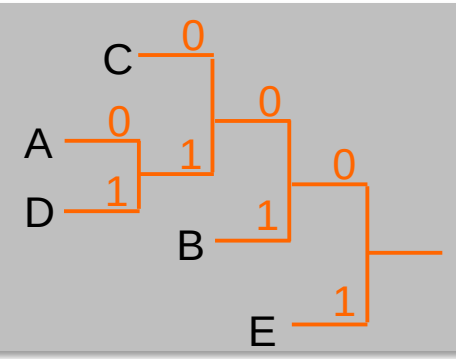
1. In each step, sort the symbols according to their probabilities in descending order.
2. Combine the two symbols with lowest probabilities to a new hypersymbol.
3. Code the decision between the two most unlikely symbols with one bit.

symbol	$P(x)$										
E	0.4		E	0.4		E	0.4		C,A,D,B	0.6	0
B	0.2		B	0.2		C,A,D	0.4	0	E	0.4	1
C	0.2		C	0.2	0	B	0.2	1			
A	0.1	0	A,D	0.2	1						
D	0.1	1									

Alternative construction:

- Allocation of bits „0“ und „1“ is arbitrary.
- Order of sorting for equally likely symbols is arbitrary.

Huffman tree:



Huffman Code

symbol x	probability $P(x)$	codeword	
		code 1	code 2
A	0.1	010	0010
B	0.2	10	01
C	0.2	11	000
D	0.1	011	0011
E	0.4	00	1
average codeword length μ_L in bit/symbol		2.2	2.2
variance of codeword length σ_L^2 in (bit/symbol)^2		0.16	1.36
maximum codeword length		3	4
minimum codeword length		2	1

The average codeword length does not depend on the choice of the Huffman code.

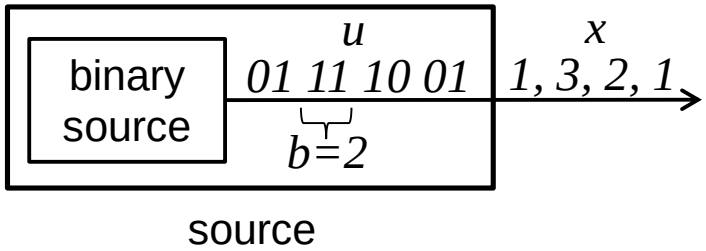
Entropy of source:

$$H(X) = \sum_x P(x) \log_2 \frac{1}{P(x)} = 2 \cdot 0.1 \cdot \log_2 \frac{1}{0.1} + 2 \cdot 0.2 \cdot \log_2 \frac{1}{0.2} + 0.4 \cdot \log_2 \frac{1}{0.4} = 2.12 \text{ bit}$$

The average codeword length $E\{L\}=\mu_L$ of a Huffman code satisfies:

$$H(X) \leq E\{L\} < H(X) + 1$$

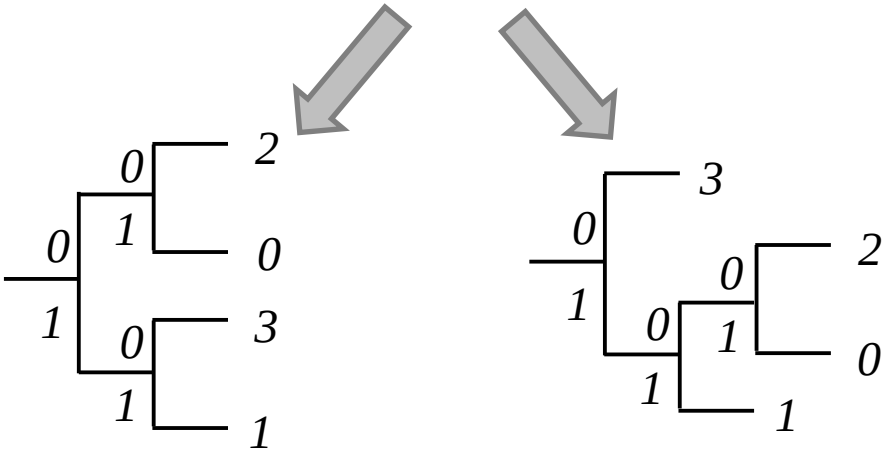
Huffman Code



$P(x=0) = p_0,$

$P(x=1) = p_1 = 1 - p_0$

→ 2 possible structures of the Huffman code:

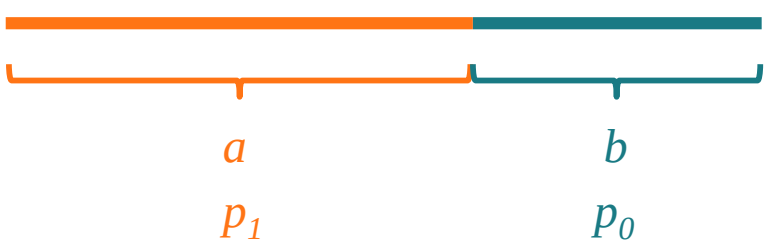


The algorithm flips between the two structures, if the ratio

$$\frac{p_1}{p_0} = \frac{p_0 + p_1}{p_1} = \frac{1}{p_1} = \frac{1+\sqrt{5}}{2}$$

corresponds to the „Golden Cut“

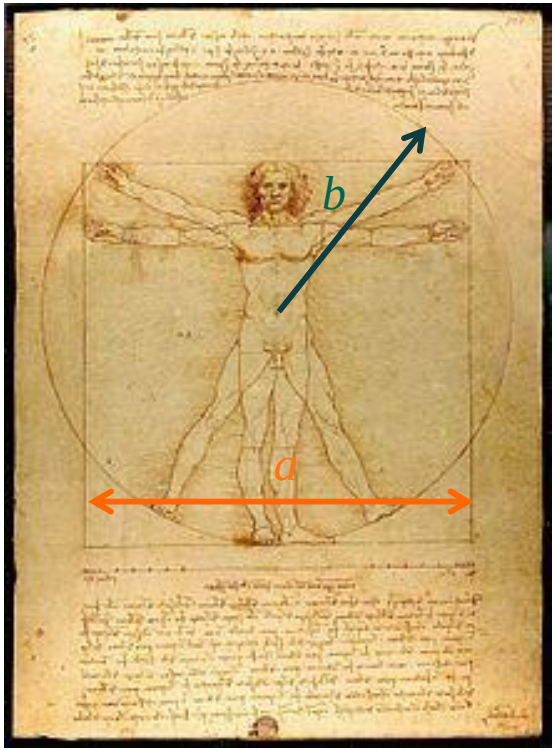
Golden Cut:

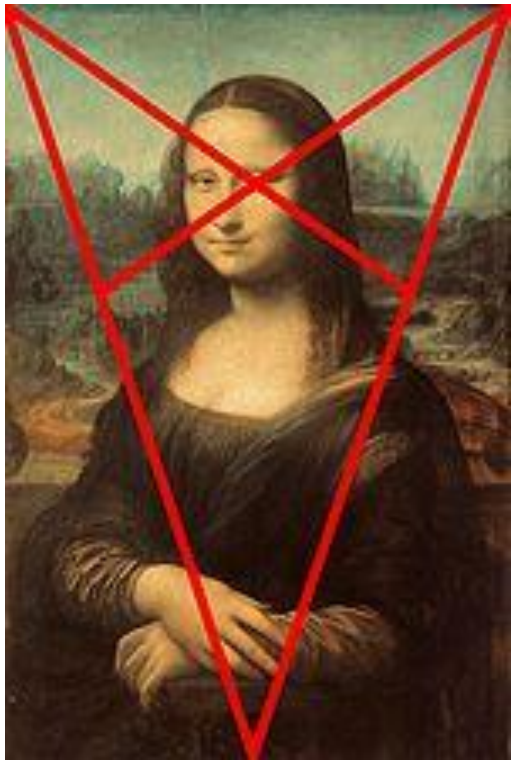


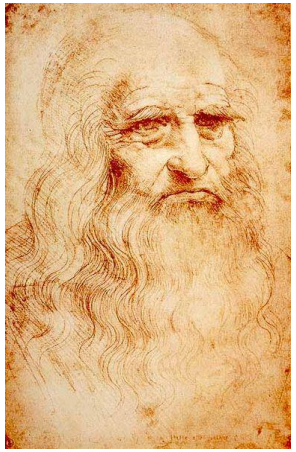
$$\frac{a}{b} = \frac{a+b}{a} = \frac{1+\sqrt{5}}{2}$$

Golden Cut in Arts

$$\frac{a}{b} = \frac{a+b}{a} = \frac{1+\sqrt{5}}{2}$$







Leonardo da Vinci
(1452-1516)

Shannon Code

Each source symbol x is mapped to a codeword $\mathbf{c}$ with codeword length according to its self-information:

A source symbol x with probability $p_X(x)$ is encoded with codeword length

$$L\{\mathbf{c}(x)\} = \left\lceil \log_2 \frac{1}{p_X(x)} \right\rceil$$

$$H(X) \leq L\{\mathbf{c}(x)\} < H(X) + 1$$

In general, the Shannon code is not an optimal source code. It achieves an average codeword length which is equal to or greater than the average codeword length of the Huffman code.

Bounds on the Codeword Length (1)

The average codeword length $E\{L\}$ of an optimum source code for a source symbol probability distribution $P_X(x)$ is bounded by

$$H(X) \leq E\{L\} < H(X) + 1,$$

where $H(X)$ is the entropy of the source.

Proof for lower bound:

→ Source coding theorem.

Bounds on the Codeword Length (2)

The average codeword length $E\{L\}$ of an optimum source code for a source symbol probability distribution $P_X(x)$ is bounded by

$$H(X) \leq E\{L\} < H(X) + 1,$$

where $H(X)$ is the entropy of the source.

Proof for upper bound:

Optimum coding: Codeword length $L(\mathbf{c}_i)$ of a codeword $\mathbf{c}_i$ is equal to the self information of the corresponding source symbol x_i:

$$L(\mathbf{c}_i) = \log_2 \frac{1}{P_X(x_i)}$$

Only integer codeword lengths can be implemented.
→ Choose codeword length according to:

$$L(\mathbf{c}_i) = \left\lceil \log_2 \frac{1}{P_X(x_i)} \right\rceil$$

$$E\{L\} = \sum_i P_X(x_i) \left\lceil \log_2 \frac{1}{P_X(x_i)} \right\rceil$$

$$< \sum_i P_X(x_i) \left(\log_2 \frac{1}{P_X(x_i)} + 1 \right)$$

$$= \underbrace{\sum_i P_X(x_i) \log_2 \frac{1}{P_X(x_i)}}_{H(X)} + \underbrace{\sum_i P_X(x_i)}_{1}$$

$$E\{L\} < H(X) + 1$$

Bounds on the Codeword Length (3)

For independent, identical distributed (i.i.d.) source symbols X_i, the average codeword length per source symbol, $E\{L\}/N$, of an optimum source code will achieve the entropy $H(X)$ of the source, if large blocks of N source symbols are coded together:

$$H(X) \le \frac{E\{L\}}{N} < H(X) + \frac{1}{N} \quad \Rightarrow \quad \frac{E\{L\}}{N} \to H(X) \text{ for } N \to \infty$$

Proof:

Combine N source symbols to an information word: $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$

Source coding: Map each information word to a codeword with average length $E\{L\}$: $\mathbf{c}(\mathbf{x}) = \mathbf{c}(x_1, x_2, \ldots, x_N)$

$$H(X_1,\ldots,X_N) \le E\{L\} < H(X_1,\ldots,X_N) + 1 \quad \Rightarrow \quad H(X) \le \frac{E\{L\}}{N} < H(X) + \frac{1}{N}$$

$$H(X_1,\ldots,X_N) = \sum_{i=1}^{N} H(X_i) = N \cdot H(X)$$

X_i statistically independent

X_i identical distributed

Relative Entropy, Kullback-Leibler Distance, Kullback-Leibler Divergence

Measure for the distance of two probability distributions $p_X(x)$ and $q_X(x)$:

$$D(p\|q) = E_p\left\{\log_2 \frac{p_X(x)}{q_X(x)}\right\} = \sum_x p_X(x)\log_2 \frac{p_X(x)}{q_X(x)}$$

$$D(p\|q) \geq 0$$

$$D(p\|q) = 0 \Leftrightarrow p_X(x) = q_X(x)$$

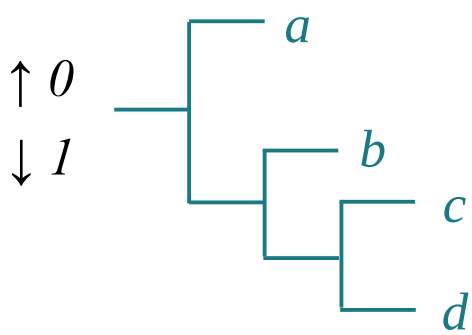
$$D(p\|q) \neq D(q\|p)$$ $\rightarrow$ not a true “distance“

The relative entropy measures the inefficiency of a source code, if the erroneous probability distribution $q_X(x)$ is used for code construction.

Relative Entropy - Example: Huffman Code

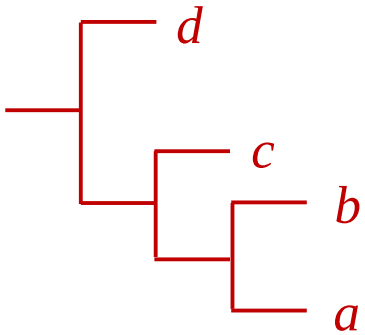
True probability distribution:

x	$p_X(x)$
a	$1/2$
b	$1/4$
c	$1/8$
d	$1/8$



Estimated (erroneous) probability distribution:

x	$q_X(x)$
a	$1/8$
b	$1/8$
c	$1/4$
d	$1/2$



Average codeword length:

$E\{L\} = H(X) = 1.75$ bit/symbol

$E\{L\} = H(X) + D(p||q) = 2.625$ bit/symbol

Cross Entropy

The *cross entropy* $H(p,q)$ between two probability distributions $p_X(x)$ and $q_X(x)$ measures the average number of bits, which is needed to identify a source symbol x, if the erroneous probability distribution $q_X(x)$ is used for code construction rather than the true probability distribution $p_X(x)$.

$$H(p,q) = E_p\left\{\log_2 \frac{1}{q_X(x)}\right\} = \sum_x p_X(x)\log_2 \frac{1}{q_X(x)} = H(X) + D(p\|q)$$

Relation to the relative entropy:

$$D(p\|q) = \sum_x p_X(x)\log_2 \frac{p_X(x)}{q_X(x)} = \underbrace{\sum_x p_X(x)\log_2 \frac{1}{q_X(x)}}_{H(p,q)} - \underbrace{\sum_x p_X(x)\log_2 \frac{1}{p_X(x)}}_{H(X)}$$

Relative Entropy, Kullback-Leibler Distance, Kullback-Leibler Divergence, Cross Entropy (1)

Relative entropy, which is also called *Kullback-Leibler distance* or *Kullback-Leibler divergence*, is a measure for the distance of two probability distributions $p_X(x)$ and $q_X(x)$. It is defined as the expectation of the logarithmic ratio of the two distributions, where the expectation is taken with respect to the propability distribution $p_X(x)$:

$$D\left(p\|q\right) = E_p\left\{\log_2 \frac{p_X(x)}{q_X(x)}\right\} = \sum_x p_X(x)\log_2 \frac{p_X(x)}{q_X(x)}$$

The relative entropy shows some properties of a distance: It is always non-zero, i.e. $D(p||q) \geq 0$ and it is zero only if $p_X(x) = q_X(x)$. However, it is not a true distance in the sense that it is not symmetric, i.e. $D(p||q) \neq D(q||p)$.
The relative entropy is related to the mutual information $I(X,Y)$ of two random variables X and Y such that the mutual information $I(X,Y)$ is the relative entropy between the joint probability distribution $p(x,y)$ and the product of the probability distributions $p(x)$ and $p(y)$:

$$I(X,Y) = \sum_x \sum_y p(x,y)\log_2 \frac{p(x,y)}{p(x)\,p(y)} = D\left(p(x,y)\|p(x)\,p(y)\right)$$

Relative Entropy, Kullback-Leibler Distance, Kullback-Leibler Divergence, Cross Entropy (2)

The relative entropy is relevant e.g. in source coding, when the true probability distribution of the source symbols X is unknown. If the true probability distribution $p_X(x)$ is known to the encoder, Shannon's source coding theorem says, that the data can be encoded by a source code with average codeword length $E\{L\}$ equal to the entropy $H(X)$, i.e. $E\{L\} = H(X)$.

The Huffman code is an example for a source code which requires knowledge of the source statistics.

In practice, the true probability distribution $p_X(x)$ of the source symbols is often unknown but has to be estimated. Therefore, the source code can only be constructed based on the estimated probability distribution $q_X(x) \neq p_X(x)$.

The relative entropy $D(p||q)$ measures the loss in terms of average codeword length, if the estimated probability distribution $q_X(x)$ is used for source code construction rather than the true probability distribution $p_X(x)$. The average codeword length of a source code which is constructed based on the estimated probability distribution $q_X(x)$ is given by

$$E\{L\} = H(X) + D(p||q).$$

I.e. the relative entropy $D(p||q)$ measures the inefficiency of a source code which is constructed based on an erroneous probability distribution $q_X(x)$.

Relative Entropy, Kullback-Leibler Distance, Kullback-Leibler Divergence, Cross Entropy (3)

The expression for the achievable average codeword length of a source code which is constructed based on the estimated probability distribution $q_X(x)$ is also called the *cross entropy*

$$H(p,q) = E_p\left\{\log_2 \frac{1}{q_X(x)}\right\} = \sum_x p_X(x)\log_2 \frac{1}{q_X(x)} = H(X) + D(p\|q)$$

between the probability distributions $p_X(x)$ and $q_X(x)$.

Lempel-Ziv Algorithm (1)

Huffman codes require a fairly accurate knowledge of the source statistics. In many applications, it is possible to estimate the symbol probabilities. However, in other compression problems such as data file compression, estimation of the source statistics is not possible and we need a source coding method which does not need a priori knowledge of the source statistics. Source coding algorithms, which minimize the codeword length without a priori knowledge of the source statistics are called *universal*.

A solution to the problem is achieved by dictionary codes, which construct their codebooks on the fly rather than using a predetermined codebook. Dictionary codes usually achieve compression only for long information sequences. In case of short information sequences, the dictionary code may deliver a coded sequence, which is even longer than the original sequence.

A famous example for dictionary codes is the class of Lempel-Ziv codes. For long information sequences, Lempel-Ziv codes are optimal source codes, i.e. the average codeword length approaches the entropy $H(X)$ of the source. Moreover, both encoder and decoder do not need a priori knowledge of the source statistics but construct the codebook on the fly.

A Lempel-Ziv algorithm codes strings of the original sequence, where the strings have variable length and the codewords have fixed length. This is kind of invers to a Huffman code, where information words of constant length are encoded by codewords of variable length.

The idea of the Lempel-Ziv algorithm is to parse the original sequence into strings which have not appeared so far. E.g. the original data sequence

0 0 1 0 1 1 1 1 0 1 1

is parsed into the $S=6$ strings

0, 01, 011, 1, 10, 11.

Lempel-Ziv Algorithm (2)

Then, each of the resulting strings consists of a prefix, which contains all symbols but the last symbol of the string, and the last symbol, which is called the *innovative symbol*. E.g. in the string 011, the first two symbols 01 are the prefix, whereas the last symbol 1 is the innovative symbol. The strings are then successively encoded, while the codebook is simultaneously constructed *on the fly*. The prefix had appeared already and is part of the codebook at a certain address. Therefore, the string can be encoded by a pointer to the codebook address, where the prefix is stored, plus the innovative symbol.

In order determine the number of bits, which are required for coding the codebook address of the prefix, we need to know the number S of strings which results from parsing the original data. For S strings, $\log_2 S$ bits are needed for coding the codebook address of the prefix. The codeword length is then given by $L = \log_2 S + 1$ bit, since we need the pointer to the prefix and the innovative symbol.

In practice, it is not desirable to first parse a long data sequence before compression can start. Therefore, the number of bits for coding the prefix address is usually fixed to a predetermined number of bits, e.g. 12 bit. In adaptive Lempel-Ziv compression, the codeword length is made variable in the sence that it starts with a smaller length, e.g. 9 bit, and can grow to a larger length, e.g. 12 bit.

A variety of modifications of the Lempel-Ziv algorithm exists and is widely used in practice. E.g. the Lempel-Ziv-Welch (LZW) algorithm is used as compression algorithm by most personal computers and also in a number of image file formats such as the Graphics Interchange Format (GIF) bitmap image format, the Tagged Image File Format (TIFF) and the Portable Document Format (PDF). One major difference of the Lempel-Ziv-Welch algorithm to the Lempel-Ziv algorithm is that a codeword consists only of a pointer but does not include an innovative symbol.

Lempel-Ziv Algorithm (1)

Contrary to the Huffman Code, knowledge of the source statistics is not required.
→ „universal source coding algorithm“

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

no prefix stored
initialization:
index 000

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1
index binary	000	001
subsequence		0
codeword		000 0

Lempel-Ziv Algorithm (2)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2
index binary	000	001	010
subsequence		0	00
codeword		000 0	001 0

Lempel-Ziv Algorithm (3)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

no prefix

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3
index binary	000	001	010	011
subsequence		0	00	1
codeword		000 0	001 0	000 1

Lempel-Ziv Algorithm (4)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4
index binary	000	001	010	011	100
subsequence		0	00	1	01
codeword		000 0	001 0	000 1	001 1

Lempel-Ziv Algorithm (5)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4	5
index binary	000	001	010	011	100	101
subsequence		0	00	1	01	11
codeword		000 0	001 0	000 1	001 1	011 1

Lempel-Ziv Algorithm (6)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4	5	6
index binary	000	001	010	011	100	101	110
subsequence		0	00	1	01	11	001
codeword		000 0	001 0	000 1	001 1	011 1	010 1

Lempel-Ziv Algorithm (7)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4	5	6	7
index binary	000	001	010	011	100	101	110	111
subsequence		0	00	1	01	11	001	010
codeword		000 0	001 0	000 1	001 1	011 1	010 1	100 0

Lempel-Ziv Algorithm (8)

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4	5	6	7	8
index binary	000	001	010	011	100	101	110	111	
subsequence		0	00	1	01	11	001	010	0101
codeword		000 0	001 0	000 1	001 1	011 1	010 1	100 0	111 1

Lempel-Ziv Algorithm (9)

Contrary to the Huffman Code, knowledge of the source statistics is not required.
→ „universal source coding algorithm“

Principle of code construction *on the fly*:

Store each subsequence of symbols, which did not occur before, by

- pointer to the already known sequence (prefix of the subsequence)
- new symbol (innovative symbol)

Example:

innovative symbol

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

shortest subsequence, which did not occur before

index	0	1	2	3	4	5	6	7	8	9
index binary	000	001	010	011	100	101	110	111		
subsequence		0	00	1	01	11	001	010	0101	10
codeword		000 0	001 0	000 1	001 1	011 1	010 1	100 0	111 1	011 0

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (1)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1

initialization

pointer

symbol

p, u	code word
0, 0	
1	

pointer codes previous symbol !

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (2)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0

$p=0$ encodes beginning of a new string

p, u	code word
0, 0	
1, 0	1
1	

pointer codes previous symbol !

codeword codes current string (up to previous symbol)

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (3)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (4)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2,	

codeword codes current string (up to previous symbol)

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (5)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1,	

codeword codes current string (up to previous symbol)

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (6)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (7)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (8)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (9)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (10)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm (11)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3, 1	
4,	

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (12)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, 0

p, u	code word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3, 1	
4, 0	4
1, -	1

1. Select next symbol u.
2. If (p,u) is already in codebook

   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

End of sequence: Store current p.

Lempel-Ziv-Welch (LZW) Algorithm: Encoding (13)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, 0

p, u	code word	Binary coded sequence (with codeword length 4 bit):
0, 0		
1, 0	1	0001
1, 0		
3, 1	3	0011
2, 0	2	0010
1, 1	1	0001
2, 1	2	0010
2, 1		
7, 0	7	0111
1, 0		
3, 1		
4, 0	4	0100
1, -	1	0001

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (1)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1,	1
	3
	2
	1
	2
	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, ?

initialization

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (2)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3,	3
	2
	1
	2
	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, ?

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (3)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2,	2
	1
	2
	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, ?

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (4)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1,	1
	2
	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, ?

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (5)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2,	2
	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, ?

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (6)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7,	7
	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, ?

encoded by next codeword

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (7)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3, 1	
4,	4
	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, ?

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (8)

Original sequence:

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

Received codeword implies:

- End of a string.
- A codeword entry is created.
- The root of the current string encodes the bit u of the previous codebook entry.
- The current pointer encodes the previous symbol.

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3, 1	
4, 0	4
1,	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, 0
10	1, ?

Lempel-Ziv-Welch (LZW) Algorithm: Decoding (9)

Original sequence:

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

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, 0

p, u	received word
0, 0	
1, 0	1
1, 0	
3, 1	3
2, 0	2
1, 1	1
2, 1	2
2, 1	
7, 0	7
1, 0	
3, 1	
4, 0	4
1,	1

Decoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
3	1, 0
4	3, 1
5	2, 0
6	1, 1
7	2, 1
8	7, 0
9	4, 0
10	1, ?

Example: Image Compression using the Lempel-Ziv-Welch (LZW) Algorithm (1)

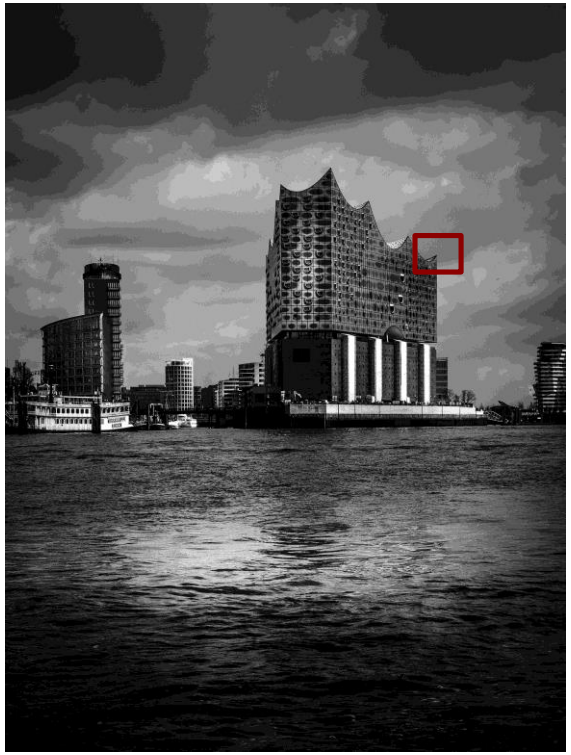
Original sequence:

57, 26, 26, 121, 0, 0, 41, 121, 2, 28, 90, 131, 11, 105, 80, 108

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
…	…
256	0, 255

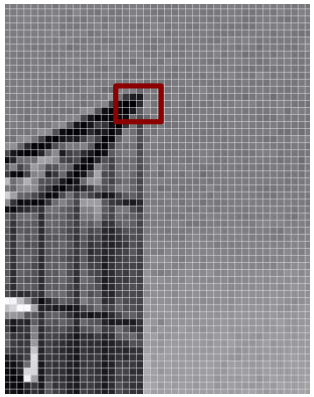
initialization



8-bit grayscale image
→ 256 intensity levels

$$\begin{bmatrix} 57 & 26 & 26 & 121 \\ 0 & 0 & 41 & 121 \\ 2 & 28 & 90 & 131 \\ 11 & 105 & 80 & 108 \end{bmatrix}$$

Intensity levels for
4 x 4 pixels



Example: Image Compression using the Lempel-Ziv-Welch (LZW) Algorithm (2)

Original sequence:

57, 26, 26, 121, 0, 0, 41, 121, 2, 28, 90, 131, 11, 105, 80, 108

Encoder codebook

address	p, u
0	0, null
1	0, 0
2	0, 1
…	…
256	0, 255
257	58, 26
257	27, 26
…	…

p, u	received word
0, 57	
58, 26	58
27, 26	27
…	…

1. Select next symbol u.
2. If (p,u) is already in codebook

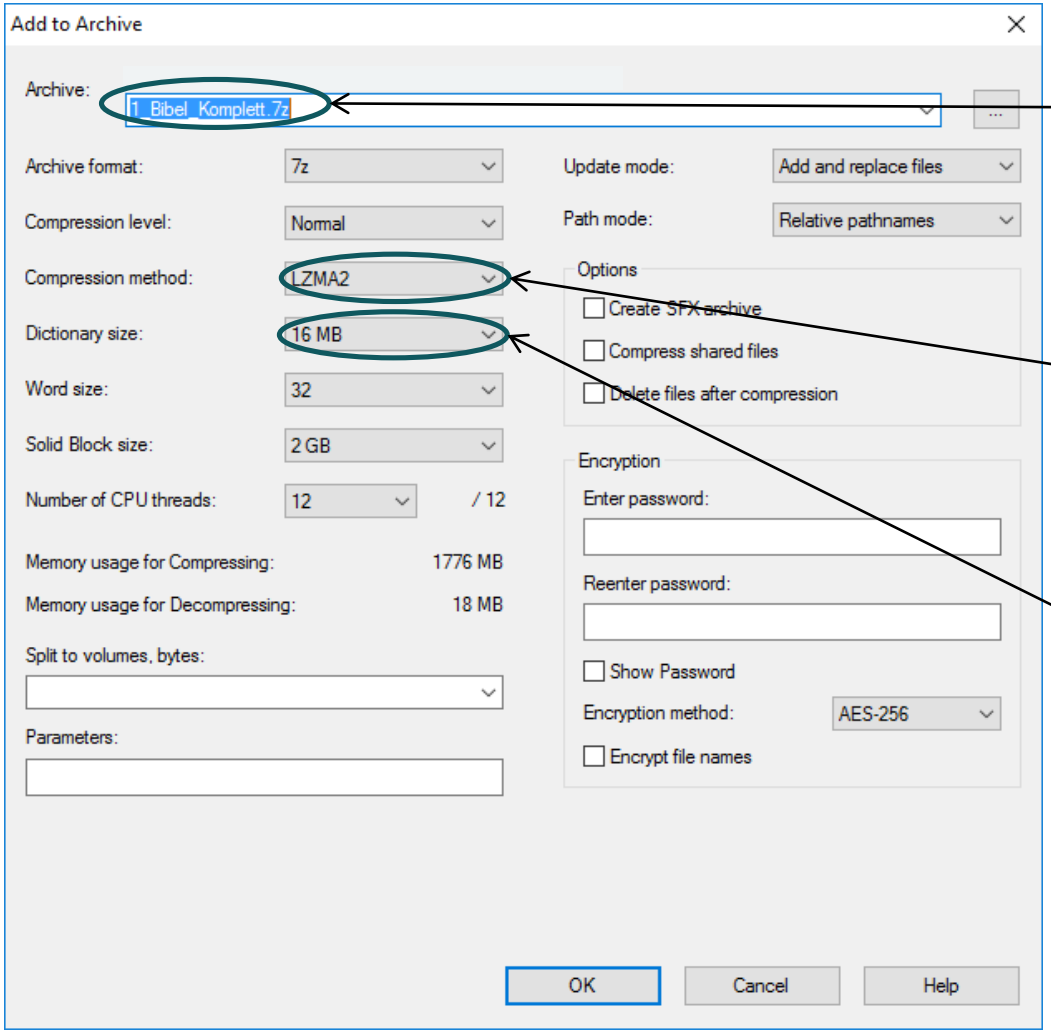
   p = address of (p,u)

   else

   - store p
   - new codebook entry (p,u)
   - p = address of $(0,u)$
3. Go to step 1.

Text Compression on a PC using 7z (1)

7z is a compressed archive file format that supports several different data compression algorithms including variants of the Lempel-Ziv algorithm.
Implementations of the compression algorithm are available for standard operating systems. E.g. 7-Zip is a Windows implementation.



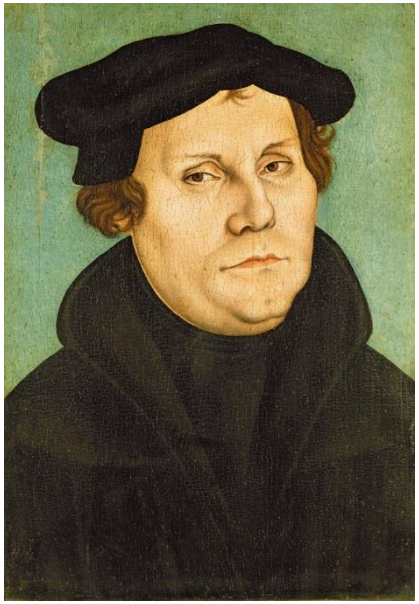
Output File

Compression algorithm

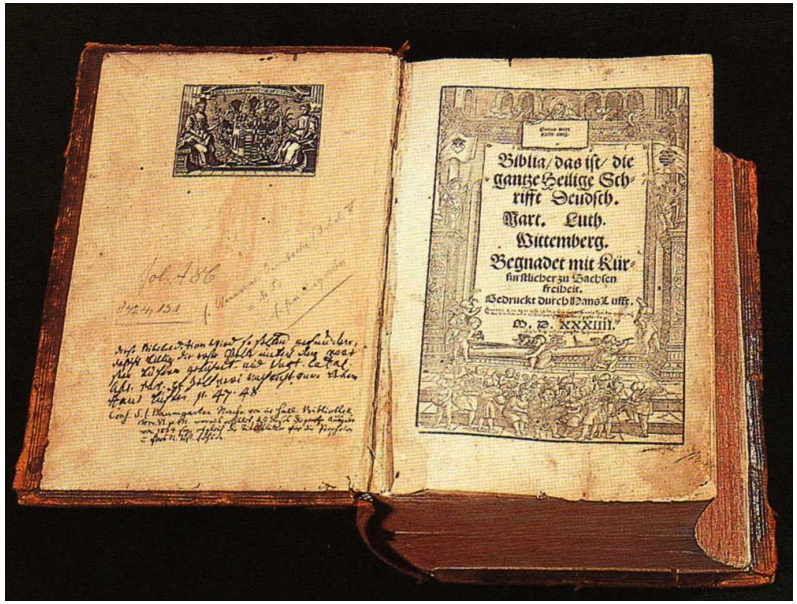
Codebook size

Text Compression on a PC using 7z (2)

Example: Text compression of the bible

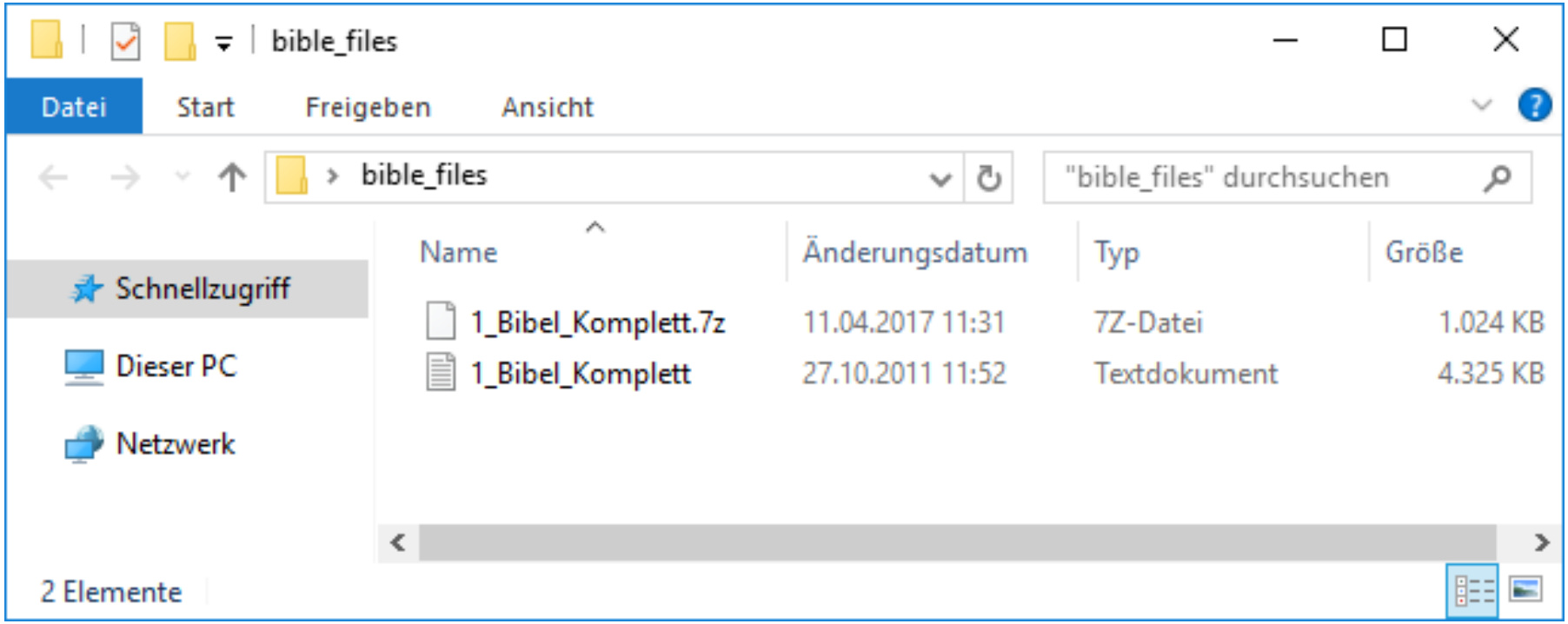


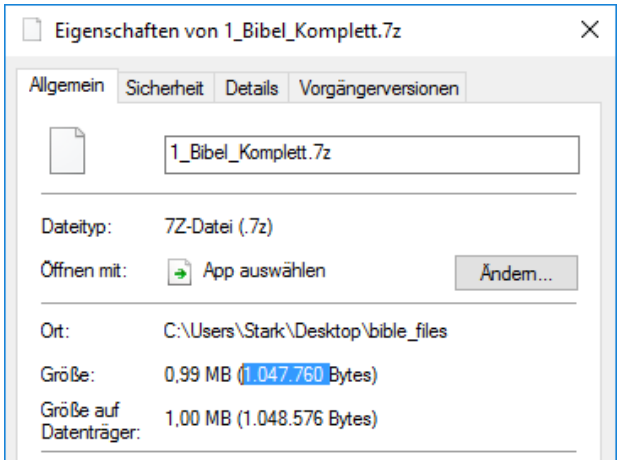
Martin Luther (1483-1546)
by Lucas Cranach the Elder
(1472-1553)
Source: Wikipedia



First complete translation of the bible
by Martin Luther (1534)
Photo: Torsten Schleese

Text Compression on a PC using 7z (3)





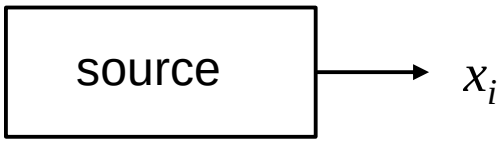
Text Compression on a PC using 7z (4)

Example: Text compression of the bible

File	Number of pages (12pt, Arial)	Original size in bytes	Compressed size in bytes	Compression ratio
Bible (German):				
complete	1.431	4.428.445	1.047.760	0.237
9 books	450	1.189.975	282.865	0.238
2 chapters	4	10.877	4.302	0.396
3 verses	<1	231	338	1.463
Bible (English)				
complete	1.435	4.467.663	9.42.068	0.211
Repetition of „a“	11	22.152	233	0.011

- 7 books of Mose, 2 books of Samuel (from the Old Testament)
- Gospel of Matthew, chapters 27, 28 (from the New Testament)
- 1st book of Mose, chapter 1, verses 1-3

Proof of Shannon's Source Coding Theorem (1)



Statistically independent random variable X with realizations x_i:

- M-ary alphabet $x_i \in \{a_1,\ldots,a_M\}$
- probabilities $P_X(X=a_m) = p_m$, $m=1,\ldots,M$

Entropy of the source:

$$H(X) = \sum_{m=1}^{M} p_m \log_2 \frac{1}{p_m}$$

Source coding theorem:

The average number of bits which, is required in order to represent a discrete symbol x_i emitted by an information source, is given by the entropy $H(X)$ of the source, given that the source symbols X are statistically independent.

$\rightarrow$ $N \cdot H(X)$ bits are on average needed for encoding of a sequence of N source symbols.

Proof of Shannon's Source Coding Theorem (2): Typical Sequences

Statistically independent random variable X with realizations x_i:

- M-ary alphabet $x_i \in \{a_1,\ldots,a_M\}$
- probabilities $P_X(X=a_m) = p_m$, $m=1,\ldots,M$

A ***typical sequence*** is a very long sequence $\mathbf{X}=[x_1, x_2, \ldots, x_N]$ of source symbols x_i, in which the relative frequency of each symbol a_m from the M-ary alphabet approaches its probability p_m.

Idea for proof:

For infinite sequence length $N \to \infty$, the probability of a non-typical sequence approaches zero.

$\to$ Non-typical sequences need no codewords.

$\to$ Find the number T of different typical sequences.

$\to$ $\log_2 T$ bits are needed in order to code each typical sequence with a unique codeword.

Proof of Shannon's Source Coding Theorem (3): Typical Sequences

Statistically independent random variable X with realizations x_i:

- M-ary alphabet $x_i \in \{a_1,\ldots,a_M\}$
- probabilities $P_X(X=a_m) = p_m$, $m=1,\ldots,M$

A ***typical sequence*** is a very long sequence $\mathbf{X}=[x_1, x_2, \ldots, x_N]$ of source symbols x_i, in which the relative frequency of each symbol a_m from the M-ary alphabet approaches its probability p_m.

Probability of a particular typical sequence $\mathbf{X}$ of length N, which contains N_m times the symbol a_m ($m=1,\ldots M$):

$$P(\mathbf{X}) = \prod_{i=1}^{N} P_X(x_i) = \prod_{m=1}^{M} p_m^{N_m}$$

Proof of Shannon's Source Coding Theorem (4): ε-Typical Sequences

Statistically independent random variable X with realizations x_i:

- M-ary alphabet $x_i \in \{a_1,\ldots,a_M\}$
- probabilities $P_X(X=a_m) = p_m$, $m=1,\ldots,M$

In an ε- ***typical sequence*** of length N, the relative frequency N_m/N of each symbol a_m differs from its probability p_m by less than εp_m, where $\varepsilon > 0$.

$$p_m - \varepsilon \underset{\substack{\uparrow \\ p_m \le 1}}{\le} p_m - \varepsilon p_m \le \frac{N_m}{N} \le p_m + \varepsilon p_m \underset{\substack{\uparrow \\ p_m \le 1}}{\le} p_m + \varepsilon$$

$\varepsilon \to 0$ for $N \to \infty$

(weak law of large numbers)

Proof of Shannon's Source Coding Theorem (5)

$$p_m - \varepsilon \le \underbrace{p_m - \varepsilon p_m \le \frac{N_m}{N}}_{} \le \underbrace{p_m + \varepsilon p_m}_{} \le p_m + \varepsilon$$

$$N_m \ge N(1-\varepsilon)\, p_m \qquad N_m \le N(1+\varepsilon)\, p_m$$

$$P(\mathbf{X}) = \prod_{m=1}^{M} p_m^{N_m} \le \prod_{m=1}^{M} p_m^{N(1-\varepsilon)p_m}$$

$$= \prod_{m=1}^{M} 2^{N(1-\varepsilon)p_m \log_2 p_m}$$

$$= 2^{N(1-\varepsilon)\sum_{m=1}^{M} p_m \log_2 p_m}$$

$$P(\mathbf{X}) = \prod_{m=1}^{M} p_m^{N_m} \ge \prod_{m=1}^{M} p_m^{N(1+\varepsilon)p_m}$$

$$= \prod_{m=1}^{M} 2^{N(1+\varepsilon)p_m \log_2 p_m}$$

$$p_m = 2^{\log_2 p_m} \quad , \quad \log p_m^k = k \log p_m$$

$$= 2^{N(1+\varepsilon)\sum_{m=1}^{M} p_m \log_2 p_m}$$

$$2^{-N(1-\varepsilon)H(X)} \ge P(\mathbf{X}) \ge 2^{-N(1+\varepsilon)H(X)}$$

Proof of Shannon's Source Coding Theorem (6)

$$2^{-N(1-\varepsilon)H(X)} \geq P(\mathbf{X}) \geq 2^{-N(1+\varepsilon)H(X)} \quad \Rightarrow \boxed{P(\mathbf{X}) \to 2^{-N \cdot H(X)}}$$

$\varepsilon \longrightarrow 0$ for $N \longrightarrow \infty$
(weak law of large numbers)

Find the number T of different typical sequences:

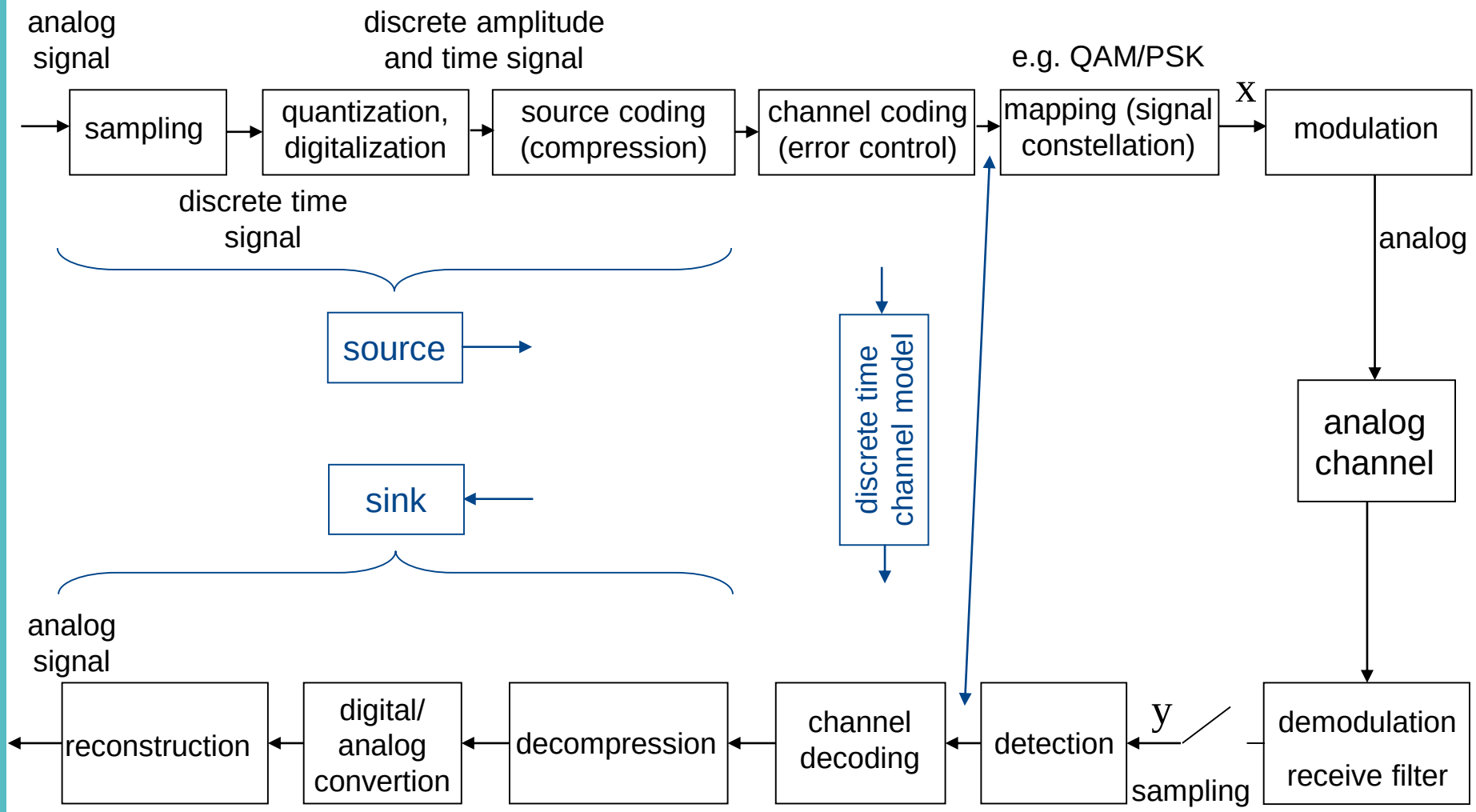
$$\sum_{t=1}^{T} P(\mathbf{X}_t) = 1 = \sum_{t=1}^{T} 2^{-N \cdot H(X)} = T \cdot 2^{-N \cdot H(X)} = 1 \quad \Rightarrow \quad \boxed{T = 2^{N \cdot H(X)}}$$

→ $N \cdot H(X)$ bits are **necessary and sufficient** in order to uniquely code each typical sequence of length N.

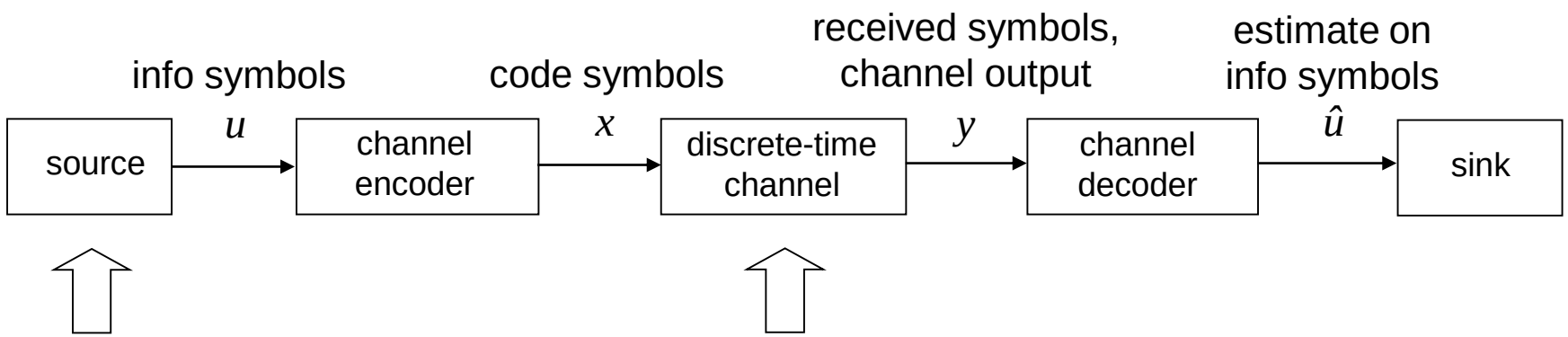
→ $H(X)$ bits on average are **sufficient** in order to uniquely code each source symbol x_i (**Source Coding Theorem**).

→ $H(X)$ bits on average are **necessary** in order to uniquely code each source symbol x_i (**Converse of the Source Coding Theorem**).

Equivalent Discrete-Time Transmission System



Equivalent Discrete-Time Transmission System



E.g. binary source:

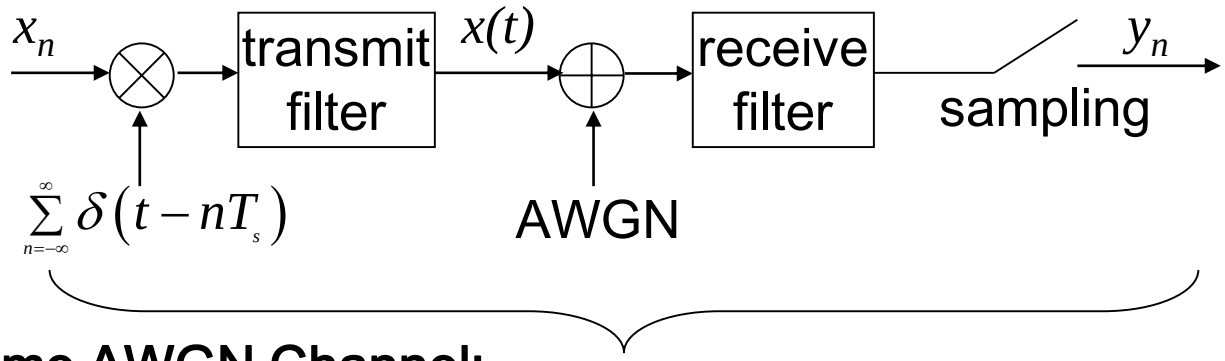
- symbols $u \in \{0,1\}$
- probabilities
  $P(u=0)=P(u=1)=0.5$

Channel model is specified by:

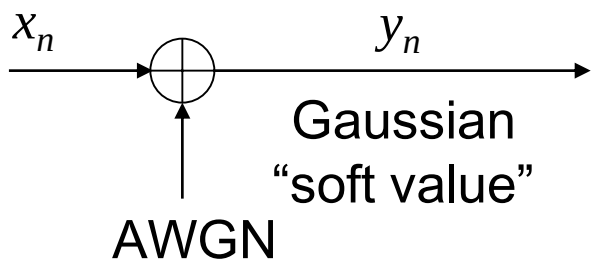
- Input alphabet $\mathcal{A}_{\text{in}}$
- Output alphabet $\mathcal{A}_{\text{out}}$
- Transition probabilities
  $P(y|x) \forall\ x \in \mathcal{A}_{\text{in}},\ y \in \mathcal{A}_{\text{out}}$

Mathematical models of the physical channel allow to treat the channel characteristics mathematically.

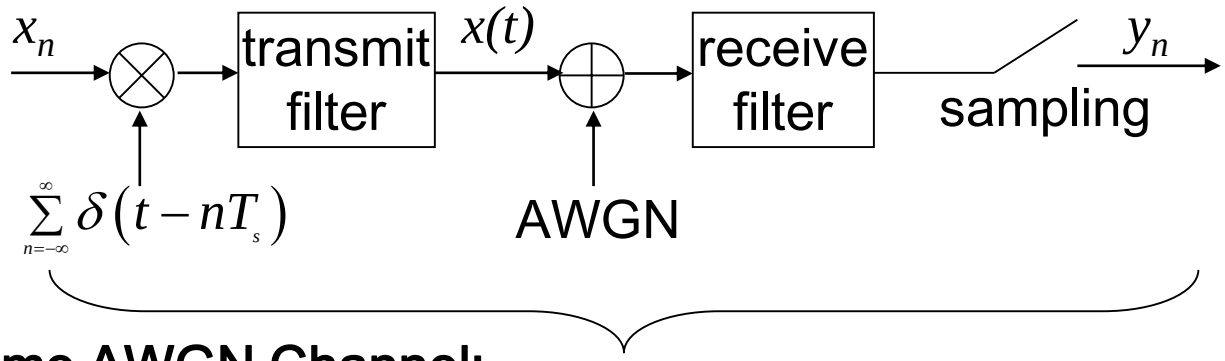
Channel Models



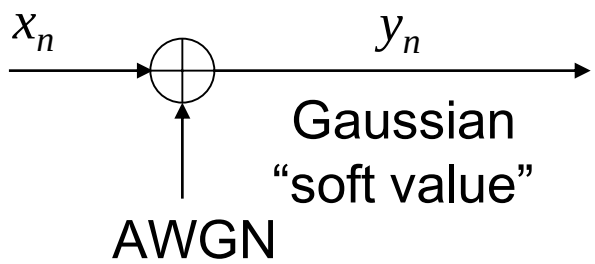
Discrete-time AWGN Channel:



Additive White Gaussian Noise (AWGN) Channel



Discrete-time AWGN Channel:



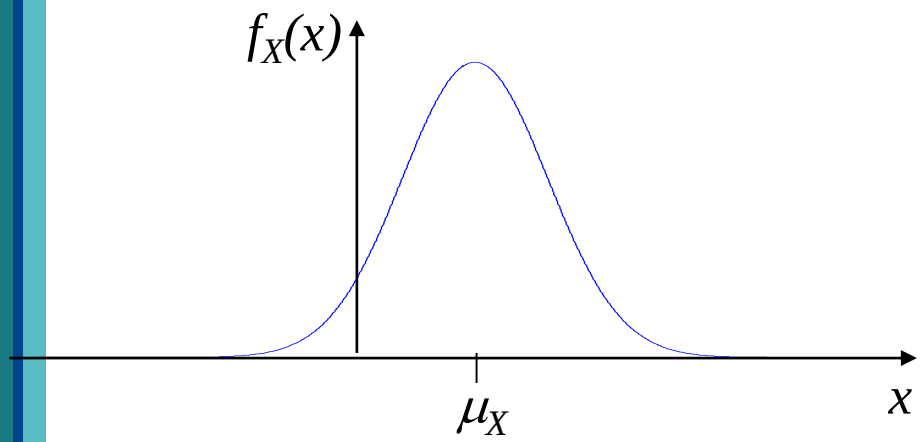
- Input alphabet $\mathcal{A}_{in} = \mathbb{R}$
- Output alphabet $\mathcal{A}_{out} = \mathbb{R}$
- Transition probability density function: $f_{Y|X}(y \mid x) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(y-x)^2}{2\sigma_N^2}}$

Additive White Gaussian Noise (AWGN) is caused by
- electronic devices (thermal noise)
- interference (e.g. in a wireless system)

Gaussian Process

Gaussian distribution
normal distribution $\mathcal{N}(\mu_X, \sigma_X^2)$

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma_X} e^{-\frac{(x-\mu_X)^2}{2\sigma_X^2}}$$



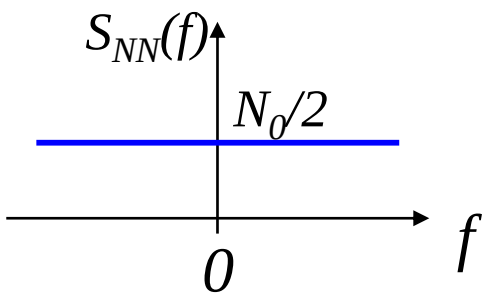
Significance of the Gaussian Process:

- The Gaussian process is appropriate to model many random processes produced by physical phenomena, e.g. receiver noise.
- The ease with which the Gaussian process can be handled mathematically makes analytic results possible.

Central Limit Theorem:
The probability distribution of a random variable, which is the sum of *N* statistically independent random variables, approaches a Gaussian distribution as the number *N* of random variables approaches infinity.

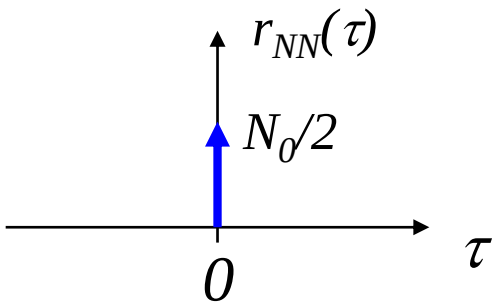
White Noise

Power spectral density:



$\bullet\!\!-\!\!\circ$

Autocorrelation function:



N_0: noise power density

$$P_N = kT\,\Delta f \text{ in } W$$

$$P_N = \int_{-\Delta f}^{\Delta f} S_{NN}(f)\,df = \int_{-\Delta f}^{\Delta f} \frac{N_0}{2}\,df = N_0 \Delta f$$

$$N_0 = kT_e \text{ in } W/Hz$$

k: Boltzmann's constant

$k = 1.38 \cdot 10^{-23}\ J/K$

T: absolute temperature in K

Δf: bandwidth

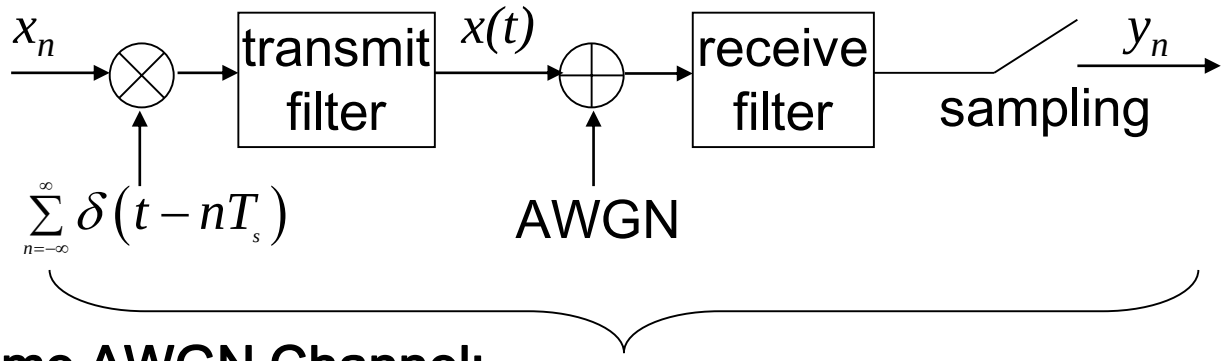
Noise samples are uncorrelated. For white Gaussian noise, they are also statistically independent.

Many sources of noise such as receiver noise are commonly modelled as additive white Gaussian noise (AWGN).

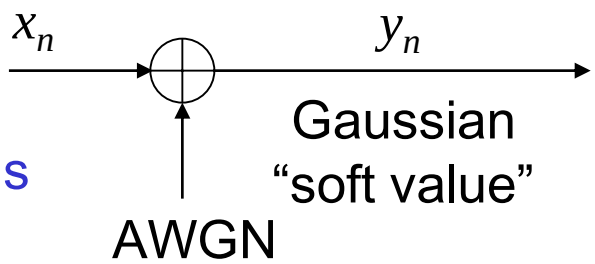
10 Deutsche Mark (DM)

Carl Friedrich Gauß (1777 - 1855)
German mathematicion, astronomer, geodesist and physicist.
Background: Historical buildings of Göttingen.

Binary Input AWGN Channel



Discrete-time AWGN Channel:



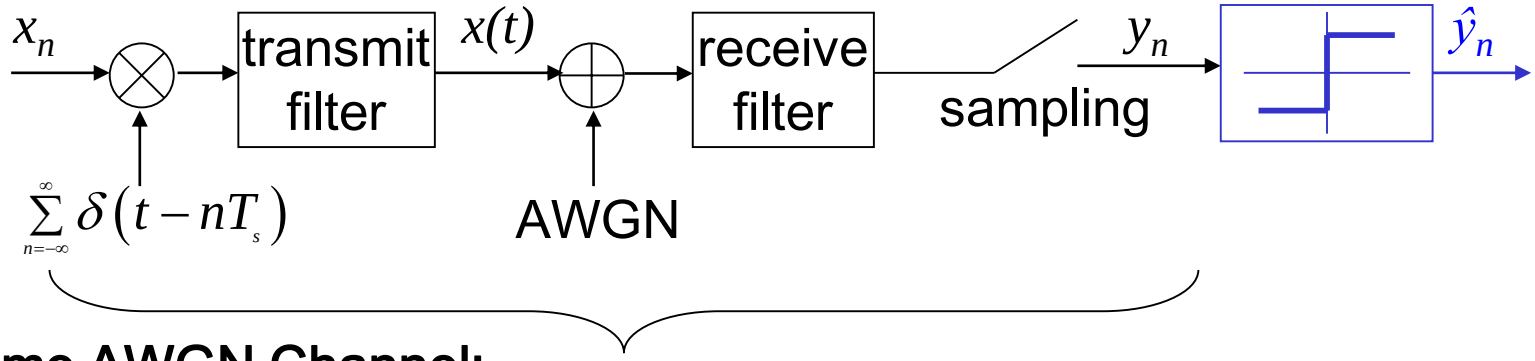
1.restriction:

binary transmit symbols

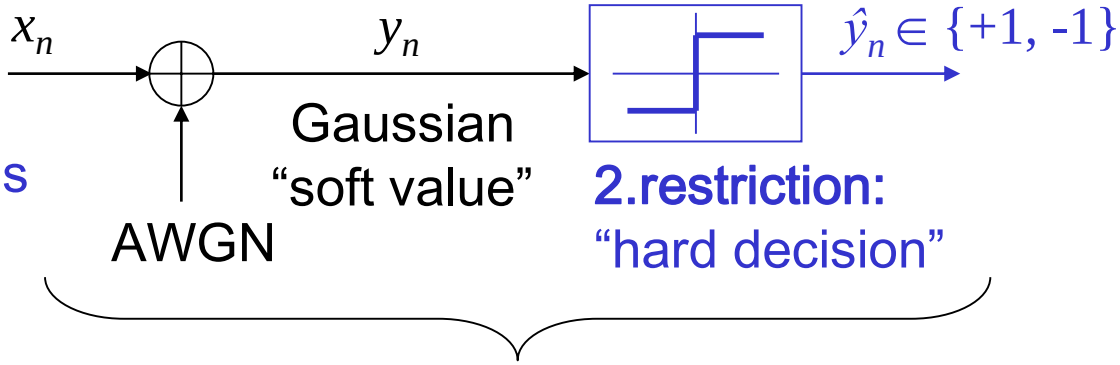
$x_n \in \{+1, -1\}$

- Input alphabet $\mathcal{A}_{in} = \{+1,-1\}$
- Output alphabet $\mathcal{A}_{out} = \mathbb{R}$
- Transition probability density function: $f_{Y|X}(y \mid x = \pm 1) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(y \pm 1)^2}{2\sigma_N^2}}$

Binary Symmetric Channel (BSC)



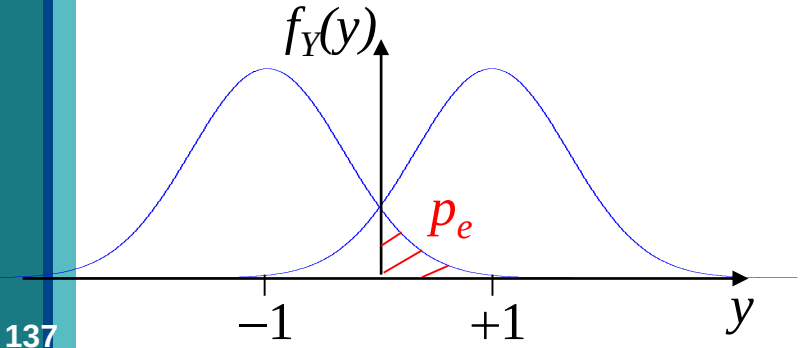
Discrete-time AWGN Channel:

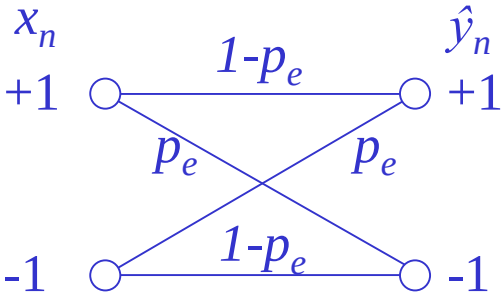


1.restriction:
binary transmit symbols
$x_n \in \{+1, -1\}$

2.restriction:
"hard decision"

Binary Symmetric Channel (BSC)

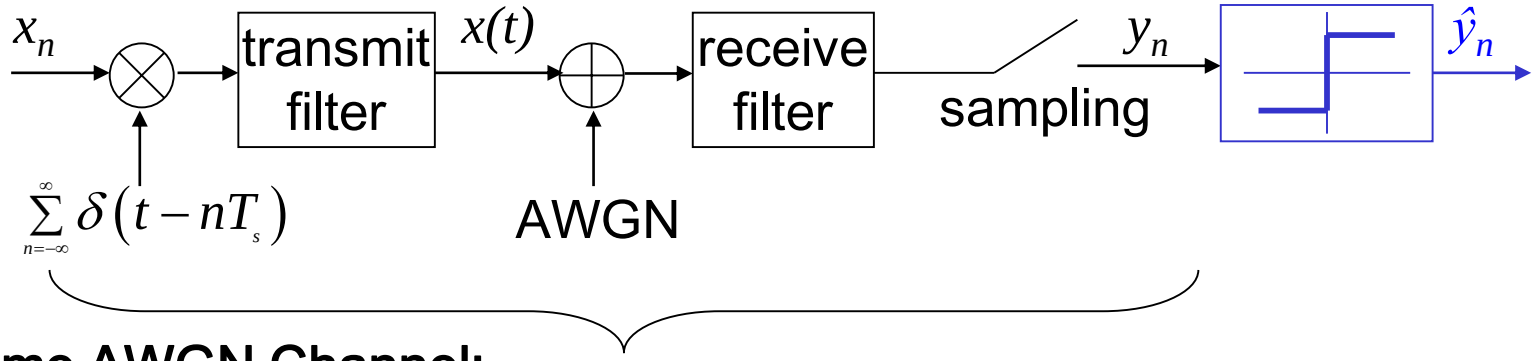




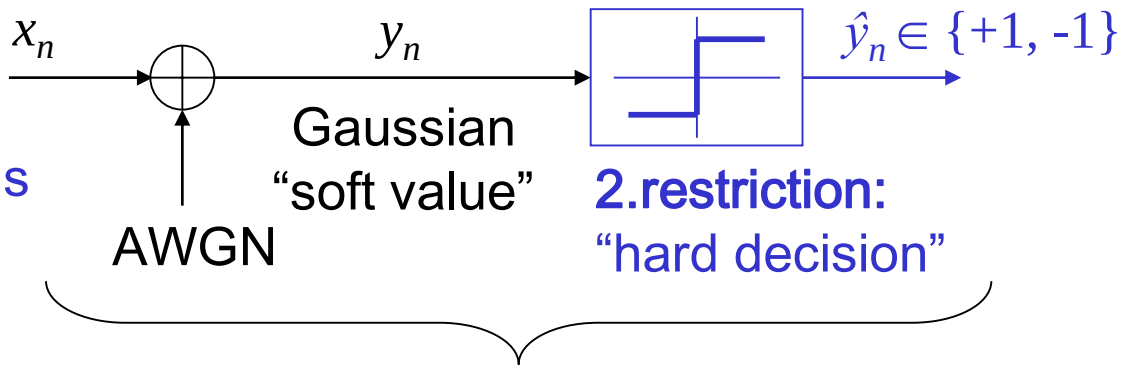
$$p_e = \int_0^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+1)^2}{2\sigma^2}} dy$$

$$\sigma^2 = \frac{N_0}{2E_s}$$

Binary Symmetric Channel (BSC)



Discrete-time AWGN Channel:



1.restriction:
binary transmit symbols
$x_n \in \{+1, -1\}$

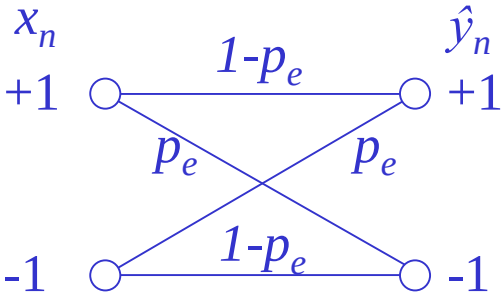
2.restriction:
"hard decision"

Binary Symmetric Channel (BSC)

- Input alphabet $\mathcal{A}_{in} = \{+1,-1\}$
- Output alphabet $\mathcal{A}_{out} = \{+1,-1\}$
- Transition probabilities:

$P(y=+1|x=+1) = P(y=-1|x=-1) = 1-p_e$

$P(y=+1|x=-1) = P(y=-1|x=+1) = p_e$



$$p_e = \int_0^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+1)^2}{2\sigma^2}} dy$$

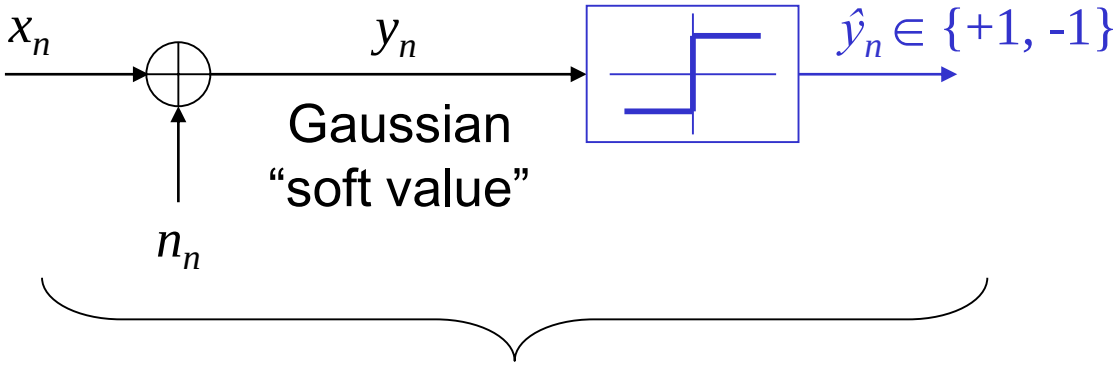
$$\sigma^2 = \frac{N_0}{2E_s}$$

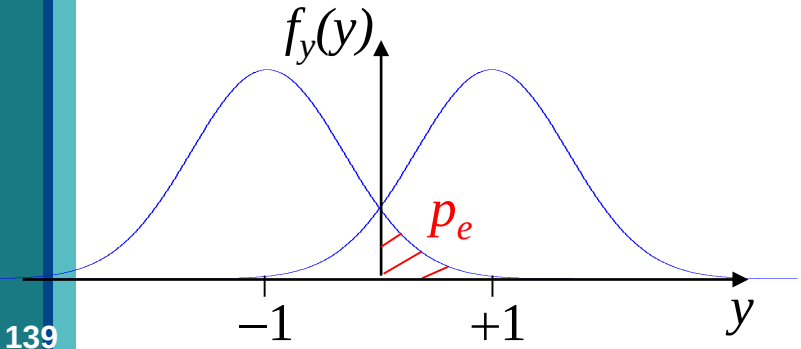
Binary Symmetric Channel (BSC)

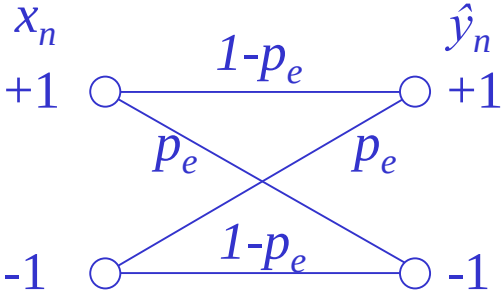
$$p_e = \int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+1)^2}{2\sigma^2}} dy = \int_1^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{n^2}{2\sigma^2}} dn = \int_{\frac{1}{\sqrt{2\sigma^2}}}^\infty \frac{\sqrt{2\sigma^2}}{\sqrt{2\pi\sigma^2}} e^{-t^2} dt = \int_{\frac{1}{\sqrt{2\sigma^2}}}^\infty \frac{1}{\sqrt{\pi}} e^{-t^2} dt$$

$$= \frac{1}{2}\,\mathrm{erfc}\left(\frac{1}{\sqrt{2\sigma^2}}\right)$$

$$t = \frac{n}{\sqrt{2\sigma^2}} \Leftrightarrow \frac{dt}{dn} = \frac{1}{\sqrt{2\sigma^2}}$$



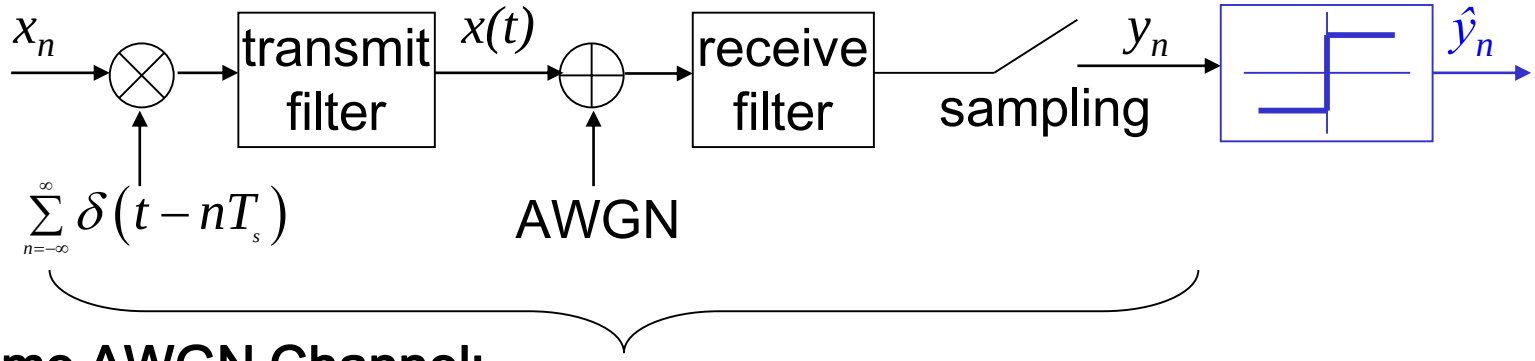




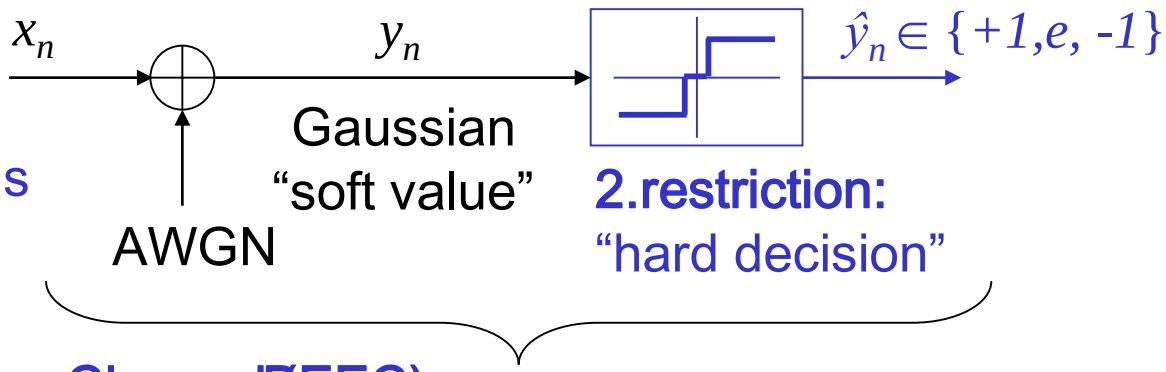
$$p_e = \frac{1}{2}\,\mathrm{erfc}\left(\frac{1}{\sqrt{2\sigma^2}}\right)$$

$$\sigma^2 = \frac{N_0}{2E_s}$$

Binary Error and Erasure Channel (BEEC)



Discrete-time AWGN Channel:



1.restriction:
binary transmit symbols
$x_n \in \{+1, -1\}$

2.restriction:
"hard decision"

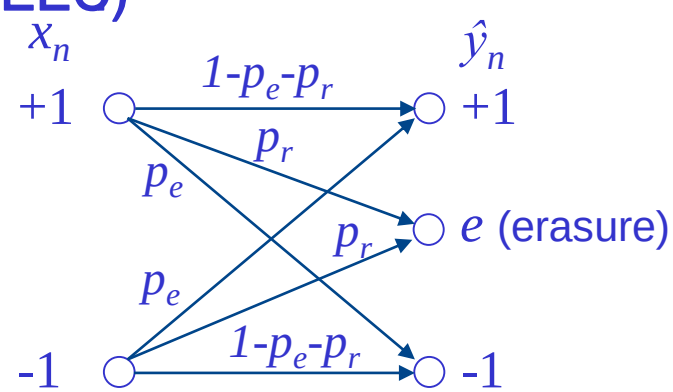
Binary Error and Erasure Channel (BEEC)

- Input alphabet $\mathcal{A}_{in} = \{+1,-1\}$
- Output alphabet $\mathcal{A}_{out} = \{+1,e,-1\}$
- Transition probabilities:

$P(y=+1|x=+1) = P(y=-1|x=-1) = 1-p_e-p_r$

$P(y=+1|x=-1) = P(y=-1|x=+1) = p_e$

$P(y=e|x=+1) = P(y=e|x=-1) = p_r$



Binary Erasure Channel (BEC)

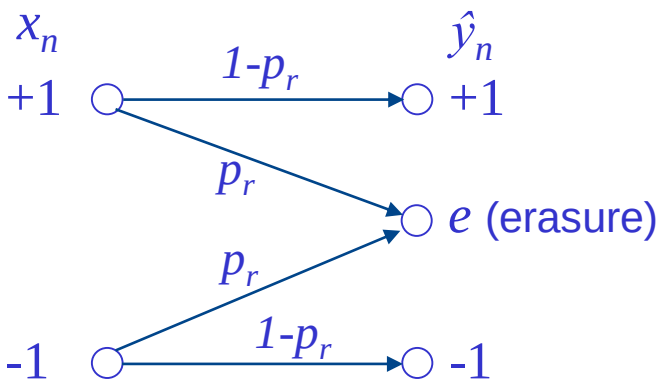
Binary Erasure Channel (BEC)

- Input alphabet $\mathcal{A}_{\text{in}} = \{+1,-1\}$
- Output alphabet $\mathcal{A}_{\text{out}} = \{+1,e,-1\}$
- Transition probabilities:

$P(y=+1|x=+1) = P(y=-1|x=-1) = 1-p_r$

$P(y=+1|x=-1) = P(y=-1|x=+1) = 0$

$P(y=e|x=+1) = P(y=e|x=-1) = p_r$



Discrete Memoryless Channels (DMC)

Memoryless means, that the transition probability density functions of symbol sequences are given by the product of the transition probability density functions of the individual symbols:

$$f_{\mathbf{Y}|\mathbf{X}}\left(y_0,\ldots,y_{N-1}\,|\,x_0,\ldots,x_{N-1}\right)=\prod_{k=0}^{N-1} f_{Y|X}\left(y_k\,|\,x_k\right)$$

Examples for DMC:

- Binary symmetric channel (BSC)
- Binary symmetric error and erasure channel (BEEC)
- Binary symmetric erasure channel (BEC)
- Frequency-flat additive white Gaussian Noise AWGN channel

Mutual Information

Mutual information between two random variables X and Y:

$$I(X;Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} = \sum_x \sum_y P(x,y) \log_2 \frac{P(x|y)}{P(x)}$$

- Statistically independent random variables X and Y:

$$P(x,y) = P(x)P(y) \qquad \Rightarrow I(X;Y) = \sum_x \sum_y P(x)\,P(y) \underbrace{\log_2 \frac{P(x)P(y)}{P(x)P(y)}}_{0} = 0$$

- Fully correlated random variables X and Y:

$$P(x|y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise} \end{cases} \qquad \Rightarrow I(X;Y) = \sum_x \underbrace{P(x,x)}_{P(x)} \log_2 \frac{1}{P(x)} = H(X)$$

entropy

Channel Capacity

Mutual information between two random variables X and Y:

$$I(X;Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} = \sum_x \sum_y P(x,y) \log_2 \frac{P(x|y)}{P(x)}$$

Channel capacity:

$$C = \max_{P(x)} I(X;Y) \quad \text{in bit/channel use}$$

Mutual Information, Entropy, Conditional Entropy

Mutual information of two real valued random variables X and Y with realizations x and y, respectively:

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

„How much information provides knowing y about x and vice versa."

„Amount of uncertainty about x that is removed by knowing y and vice versa."

Discrete random variables:

Continuous random variables:

Entropy of a random variable X with realization x:

$$H(X) = \sum_x P(x) \log_2 \frac{1}{P(x)}$$

$$h(X) = \int f_X(x) \log_2 \frac{1}{f_X(x)} dx$$

„Amount of uncertainty about x."

Conditional entropy of a random variable X with realization x given the realization y of a random variable Y:

$$H(X|Y) = \sum_x \sum_y P(x,y) \log_2 \frac{1}{P(x|y)}$$

$$h(X|Y) = \int\int f_{XY}(x,y) \log_2 \frac{1}{f_{X|Y}(x|y)} dx\, dy$$

„Amount of uncertainty about x that remains after y is known."

Mutual Information

Mutual information of two real valued random variables X and Y with realizations x and y, respectively:

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

„How much information provides knowing y about x and vice versa."

„Amount of uncertainty about x that is removed by knowing y and vice versa."

Discrete random variables:

$$I(X;Y) = \sum_x \sum_y P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x) P_Y(y)}$$

Continuous random variables:

$$I(X;Y) = \iint f_{XY}(x,y) \log_2 \frac{f_{XY}(x,y)}{f_X(x) f_Y(y)} dx\, dy$$

Discrete random variable X and continuous random variable Y:

$$I(X;Y) = \sum_i \int f_{Y|X}(y|x_i) P(x_i) \log_2 \frac{f_{Y|X}(y|x_i) P(x_i)}{P(x_i) \sum_j f_{Y|X}(y|x_j) P(x_j)} dy$$

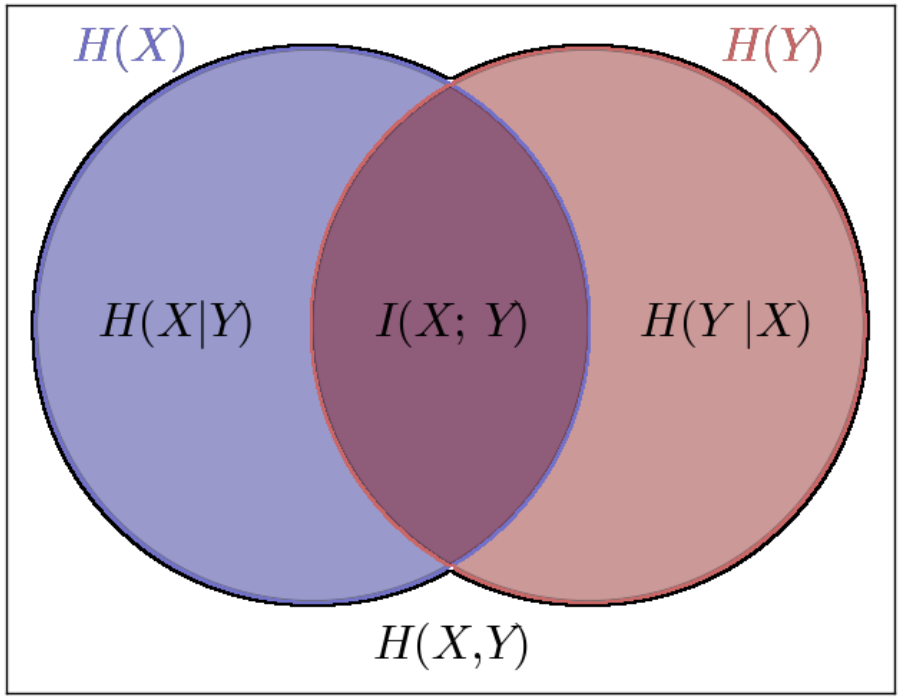
Mutual Information, Entropy, Conditional Entropy

Mutual information of two real valued random variables X and Y with realizations x and y, respectively:

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

„How much information provides knowing y about x and vice versa."

„Amount of uncertainty about x that is removed by knowing y and vice versa."



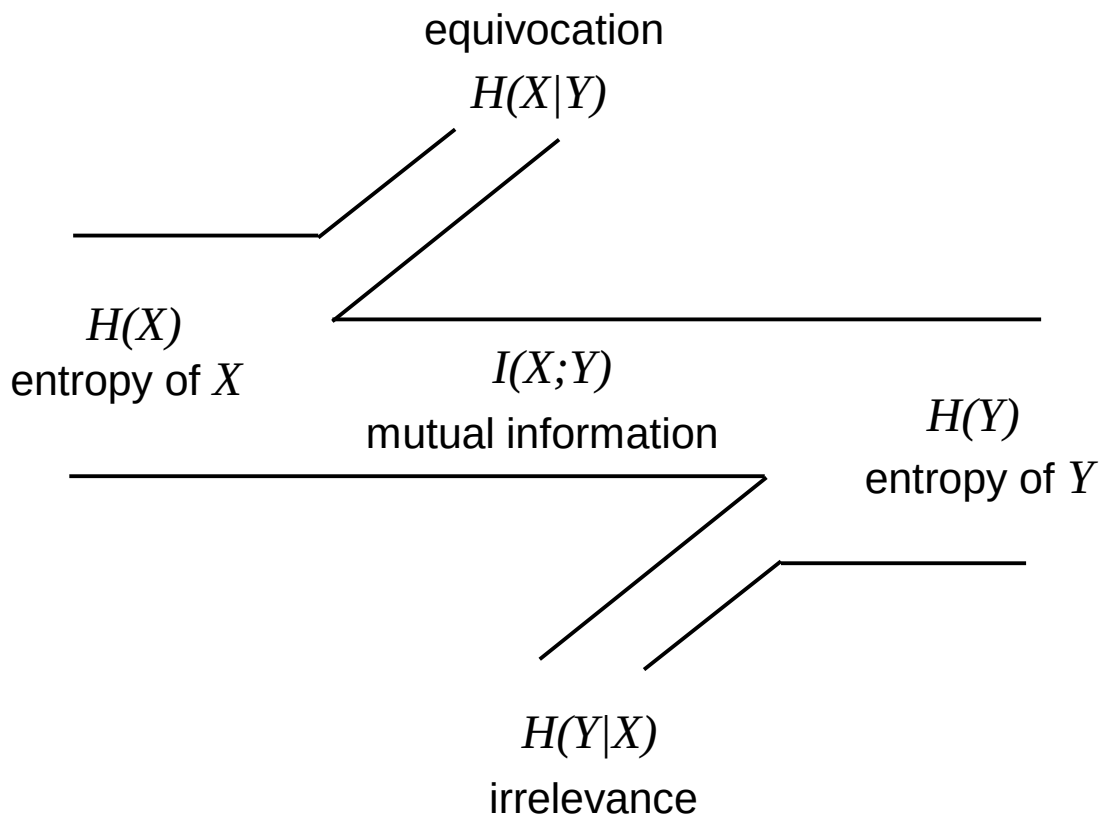
Mutual Information, Entropy, Conditional Entropy

Mutual information of two real valued random variables X and Y with realizations x and y, respectively:

$$I(X,Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

„How much information provides knowing y about x and vice versa."

„Amount of uncertainty about x that is removed by knowing y and vice versa."



Conditioning Reduces Entropy

$$H(X|Y) \leq H(X)$$

$H(X|Y) = H(X)$, if X and Y are statistically independent.

Proof:

$$0 \leq I(X;\ Y) = H(X) - H(X|Y)$$

$$H(X) \geq 0$$

$$H(X|Y) \geq 0$$

$$\Downarrow$$

$$H(X|Y) \leq H(X)$$

$$H(X|Y) = \sum_x \sum_y p(x,y) \log_2 \frac{1}{p(x|y)}$$

$$= \sum_x \sum_y p(x)\, p(y) \log_2 \frac{1}{p(x)}$$

X, Y statistically independent

$$= \sum_x p(x) \log_2 \frac{1}{p(x)} \underbrace{\sum_y p(y)}_{1}$$

$$= H(X)$$

Mutual Information (1)

The entropy $H(X)$ quantifies the information proivided by a source and defines the minimum average number of bits per symbol which is required for lossless source coding. The source coded data is then transmitted through a noisy channel. The output of the channel is characterized by another random variable Y. We are now interested in the question, how much information about the transmitted symbols x is contained in the received channel outputs y. For this purpose, a new measure of information, the mutual information $I(X;Y)$ of two random variables X and Y, is defined. Simply put, the mutual information quantifies, how much information we obtain about the channel input X by observing the received channel output Y.

Shannon defines the mutual information as

$$I(X;Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x \mid y)}{P(x)}.$$

The definition above holds for discrete random variables X and Y. In case of continuous random variables, probabilities $P(.)$ are replaced by probability density functions $f(.)$ and the sums are replaced by integrals and the mutual information is defined by

$$I(X;Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{XY}(x,y) \frac{f_{X|Y}(x \mid y)}{f_X(x)} dx\, dy.$$

Similar as for the self information $I(X)$, we can understand the meaning of this definition by looking at special cases:

Mutual Information (2)

First, consider statistically independent random variables X and Y. Using the definition of statistical independence, i.e.

$$P(x,y) = P(x)\,P(y)$$

it is easy to show that

$$P(x|y) = P(x),$$

i.e., knowledge of the realization y of the random variable Y does not reduce the uncertainty about the realization x of the random variable X. Plugging this result in the definition of the mutual information yields

$$I(X;Y) = \sum_x \sum_y P(x)P(y)\log_2 \frac{P(x)}{P(x)} = 0.$$

This is a reasonable result, since statistically independent random variables provide no information about each other.
In our context of transmission through a noisy channel, channel input and channel output could be (approximately) statistically independent, if the power of the additive channel noise is so much higher than the received information signal power that it totally dominates the received signal and, hence, virtually only noise is observed.

Mutual Information (3)

As a second special case, we consider totally correlated random variables X and Y. In this case, we have $P(x|y) = 1$ for one particular pair $[x, y]$ and $P(x|y) = 0$ otherwise. The mutual information then becomes

$$I(X;Y) = \sum_x P(x,x)\log_2 \frac{1}{P(x)} \sum_x P(x)\log_2 \frac{1}{P(x)} = H(X)$$

equal to the entropy of X. This is also a reasonable result, since in this case, observing Y yields the same information as directly observing X.

An interesting property of the mutual information is its symmetry: Observing Y yields the same information about X as observing X yields about Y. This is a consequence of Bayes theorem:

$$I(X;Y) = \sum_x \sum_y P(x,y)\log_2 \frac{P(x|y)}{P(x)} = \sum_x \sum_y P(x,y)\log_2 \frac{P(y|x)\frac{P(x)}{P(y)}}{P(x)}$$

$$= \sum_x \sum_y P(x,y)\log_2 \frac{P(y|x)}{P(y)} = I(Y,X)$$

Mutual Information (4)

The mutual information can be expressed by

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

This expression on the right hand side of the equation above can be derived as follows:

$$\begin{aligned}
I(X;Y) &= \sum_x \sum_y P(x,y) \log_2 \frac{P(x|y)}{P(x)} = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \\
&= -\sum_y \underbrace{\sum_x P(x,y)}_{P(y)} \log_2 P(y) + \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)} \\
&= H(Y) + \sum_x \sum_y P(x,y) \log_2 \frac{P(y|x)P(x)}{P(x)} \\
&= H(Y) - H(Y|X),
\end{aligned}$$

where $H(Y|X=x) = -\sum_x \sum_y P(x,y) \log_2 P(y|x)$ is called the conditional entropy.

Mutual Information (5)

In the same way, we can proof that

$$I(X;Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x|y)}{P(x)} = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

$$= -\underbrace{\sum_x \sum_y P(x,y)}_{P(x)} \log_2 P(x) + \sum_x \sum_y P(x,y) \log_2 \frac{P(x|y)P(y)}{P(y)}$$

$$= H(X) - H(X|Y)$$

with the conditional entropy $H(X|Y=y) = -\sum_x \sum_y P(x,y) \log_2 P(x|y)$.

From

$$I(X;Y) = H(X) - H(X|Y),$$

we can interprete the mutual information as the information which passes through the channel from the transmitter to the receiver. Compared to the entropy $H(X)$ which quantifies the information at the channel input, the mutual information is reduced by the so-called equivocation $H(X|Y)$, which in a sense quantifies the information which is lost during transmission.

Mutual Information (6)

From

$$I(X;Y) = H(Y) - H(Y|X) \Rightarrow H(Y) = I(X,Y) + H(Y|X),$$

we can conclude that the entropy $H(Y)$ at the channel output consists of the mutual information and the so-called irrelevance $H(Y|X)$, which may be seen as the contribution of the channel noise.

Chain Rules for Entropy and Mutual Information

Conditional entropy:

$$H(Y|X) = \sum_x \sum_y P(x,y) \log_2 \frac{1}{P(y|x)}$$

Chain rule for entropy:

$$H(X_1, X_2, \ldots, X_N) = \sum_{i=1}^{N} H(X_i | X_1, X_2, \ldots, X_{i-1})$$

Conditional mutual information:

$$I(X;Y|Z) = \sum_x \sum_y \sum_z P(x,y,z) \log_2 \frac{P(x,y|z)}{P(x|z)P(y|z)}$$

Chain rule for mutual information:

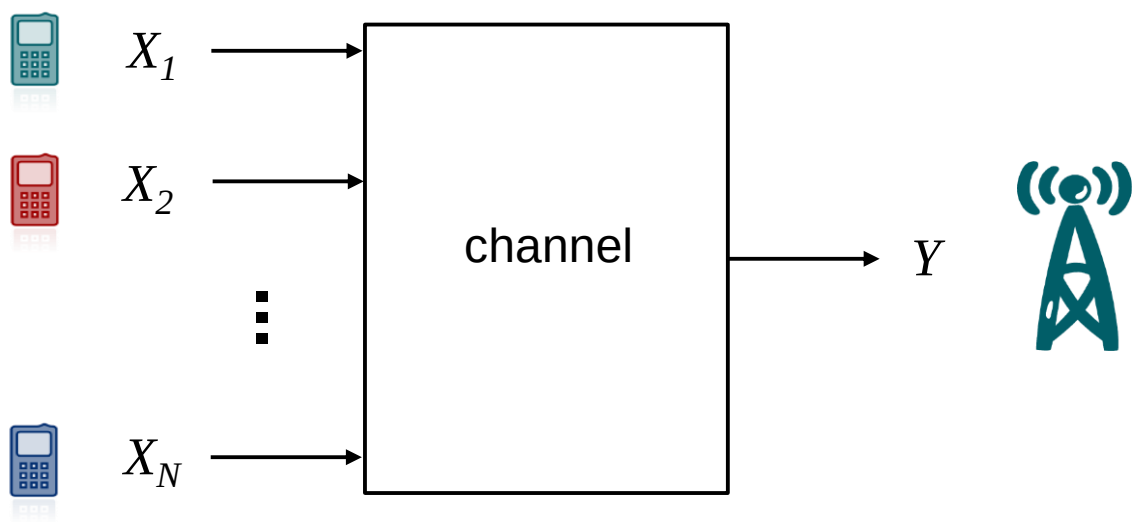
$$I(X_1, X_2, \ldots, X_N; Y) = \sum_{i=1}^{N} I(X_i; Y | X_1, X_2, \ldots, X_{i-1})$$

Mutual Information Between Multiple Random Variables

$$I(X_1, X_2, \ldots, X_N; Y) =$$
$$= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_N} \sum_{y} P(x_1, x_2, \ldots, x_N, y) \log_2 \frac{P(x_1, x_2, \ldots, x_N, y)}{P(x_1, x_2, \ldots, x_N) P(y)}$$

How much information about $X_1, X_2, \ldots, X_N$ is contained in Y ?

Example: Multiple access channel (MAC)



Channel Capacity

The mutual information quantifies the information which we obtain about a random variable X if we observe a random variable Y. In communications, we observe received values y at the receiver. The problem is to estimate the transmitted symbols x based on the observation of y. The transmit signal should be designed such that the mutual information is maximized. The maximum mutual information which can be obtained depends on the channel model and the transmit symbol alphabet. The channel cannot be influenced by the system designer. However, the symbol alphabet can be optimized for a particular channel model.

Shannon defines the maximum mutual information for a particular channel model as channel capacity C. I.e. the channel capacity is the theoretically maximum data rate in bit/channel use which can be transmitted error-free over the considered channel. Error-free transmission can be achieved if the transmit data is protected by a forward error control (FEC) code, i.e. by channel coding. For a channel with additive white Gaussian noise, the optimum transmit symbols can be shown to be Gaussian distributed.

The channel coding theorem states that there exist codes with rate $R<C$ which enables transmission with arbitrarily small error probability (error-free). No such codes exist for $R>C$. The important message is, that the channel characteristics do not limit the transmission quality (in the sense of transmission errors) but only the throughput. However, in practice codes are often used for improvement of bit error rate performance, since realization of optimum codes is a problem.

Channel Coding (1)

Shannon only defines the limit which can be achieved if channel coding is applied. However, he does not give design rules how such a code should be constructed. The proof is based on randomly chosen codes and optimal (maximum likelihood) decoding. Furthermore, he states that capacity achieving codes require infinite block length and high complexity. Since the publication of Shannon‘s channel coding theorem in 1948, finding codes which can approach the Shannon limit has been a major research topic. A breakthrough was the invention of „Turbo Codes“ in 1993. The idea of Turbo Codes is to build a strong code by a parallel concatenation of relatively simple constituent codes. The essential idea is an iterative („Turbo“) decoding algorithm. The constituent codes can be decoded in terms where each constituent decoder uses the decoding result of the other constituent decoder as additional input information on top of the received symbols. Turbo codes and coding schemes derived from the same idea allow to virtually achieve the capacity limit using codes of manageable complexity. Turbo codes have already become part of modern communications systems standards such as UMTS and LTE.

Channel Coding (2)

The principle of channel coding is to add redundancy to the data such that error correction becomes possible. Block codes operate on blocks of information bits. A block of K information bits, a so-called information word, is mapped to a sequence of $N > K$ bits, called a codeword. Hence, N-K bits redundancy are added. The ratio of information bits and codebits is called the code rate $R = K/N$, for which typically $R<1$ holds.

The simplest block code is a repetition code: A block consists of $K=1$ information bit. The codeword is just N times the repetition of the information bit. E.g. with a rate $R=1/3$ repetition code, the information word 0 is mapped to the codeword 000, whereas the information word 1 is mapped to the codeword 111.

All channel codes are based on the principle that only a subset of all possible combinations of N bits are codewods. With $N=3$, there are $2^N=8$ possible bit patterns, e.g. 000, 001, 010, 011,... . However, only two of them, the sequences 000 and 111 are codewords. Consequently, if due to a transmission error, a sequence is received, which is not a codeword, the receiver detects that a transmission error has occured. The decoder needs to find out, which codeword has most likely been transmitted. It can e.g. decide for the codeword, which differs in the smallest number of bits from the received word.

Assume e.g. that we transmit the information word 0 using the repetition code of rate $R=1/3$. The transmitted codeword is 000. Due to a transmission error, we receive the word 010. Since this sequence is not a codeword, the decoder detects a transmission error. The closest codeword to the received word is 000, which corresponds to the information word 0. Hence, the decoder will decode the information word 0.

Assume now that two transmission errors occur and the word 110 is received. In this case, the decoder will assume that the codeword 111 has been transmitted. Consequently, it will erroneously decode the information word 1.

Channel Coding (3)

The number of errors which can be corrected by a channel code depends on the number of bit positions in which the codewords differ. The number of bits in which two codewords differ is called the Hamming distance d_H. The Hamming distance of two codewords 000 and 111 of the rate $R=1/3$ repetition code is given by $d_H=3$, since the two words differ in three positions.
For more complex codes with more than two distinct codewords, the minimum Hamming distance $d_{\min}$ among all possible pairs of codewords is an important parameter, since it determines, how many errors can be corrected for sure by the code.
In general, a channel code can detect $d_{\min}-1$ errors, but it can correct only $\lfloor (dmin-1)/2 \rfloor$ errors.
Channel codes with minimum Hamming distance $d_{\min}<3$ are called error detecting codes. Channel codes with larger minimum Hamming distance are called error correcting codes or forward error correcting (FEC) codes.
The problem of code design is to find codes for a target rate R, which have as high minimum Hamming distance as possible. Since a channel code introduces redundancy, more bits have to be transmitted than in case of uncoded transmission. Consequently, more energy is required in order to transmit a message. The error correction capability of a channel code needs to overcompensate this energy and rate penalty in order to be worth being implemented.
We have used the repetition code as introductory example due to its simplicity. However, the repetition code is a highly inefficient code. More efficient codes operate on larger blocks of information bits and, consequently, larger codewords. This is in line with Shannon‘s channel coding theorem which suggests that a capacity achieving code should use infinitely large blocks, i.e. $N \rightarrow \infty$.

Channel Coding (3)

In practice, not only the minimum Hamming distance is important but also the implementation complexity. Most of the complexity is typically at the decoder side, since the decoder needs to find the most likely transmitted codeword out of 2^K candidates.

Convolutional codes are a class of channel codes for which a computationally efficient decoding algorithm, the Viterbi algorithm, exists. Therefore, convolutional codes are widely applied in practice, e.g. in the GSM and UMTS cellular communications system.

Turbo codes are a sophisticated coding scheme, which consists of the parallel concatenation of two convolutional codes. The decoder uses two Viterbi-type decoders which exchange information in an iterative decoding process. Turbo codes are very powerful and can achieve performance close to Shannon‘s capacity limit while having manageable decoding complexity. Therefore, they are applied in modern systems such as HSPA in UMTS and LTE cellular systems.

Low density parity check (LDPC) codes originate from the spirit of block codes. However, they can be efficiently decoded by a similar iterative algorithm as turbo codes and achieve similar performance. Therefore, they are also used in modern systems such as WLAN or WiMax.

Shannon‘s Separation Theorem

It may seem to be counterintuitive to first remove all redundancy from the source signal by source coding, which results in highly sensitive data and needs to be strongly protected against transmission errors by channel coding, i.e. by adding redundancy. However, Shannon‘s separation theorem states that it is optimum to treat the problems of source coding and channel coding separately. I.e., it is optimum to first remove all redundancy by source coding and then perform channel coding.

An intuitive explanation for the optimality of separate source and channel coding may be that unlike the redundancy which is contained in a source signal, the redundancy added by channel coding is particularly optimized for error correction.

However, it should be noted that the separation theorem is a theoretical result which assumes optimum source coding, optimum channel coding and infinitely long message blocks. In practice, those optimum codes are not available. Therefore, it is sometimes useful to consider source coding and channel coding jointly. In the scientific community, this is known as *joint source and channel coding*.

The redundancy is not completely removed from the source data in order to make the compressed data less sensitive to transmission errors. The channel code can take the properties of the compressed data into account. E.g. it can protect important bits better than less important bits. E.g. correct receiption of the most significant bit in a quantized speech signal may be more important for understanding the message than the least significant bit.

Channel Coding Theorem

There exists a code with rate

$$R < C$$

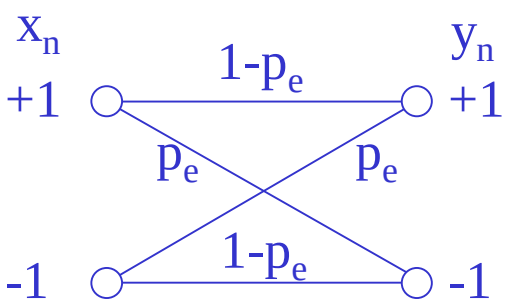
which allows transmission of blocks with infinite length with arbitrarily small error probability (error free).

Converse to the channel coding theorem:

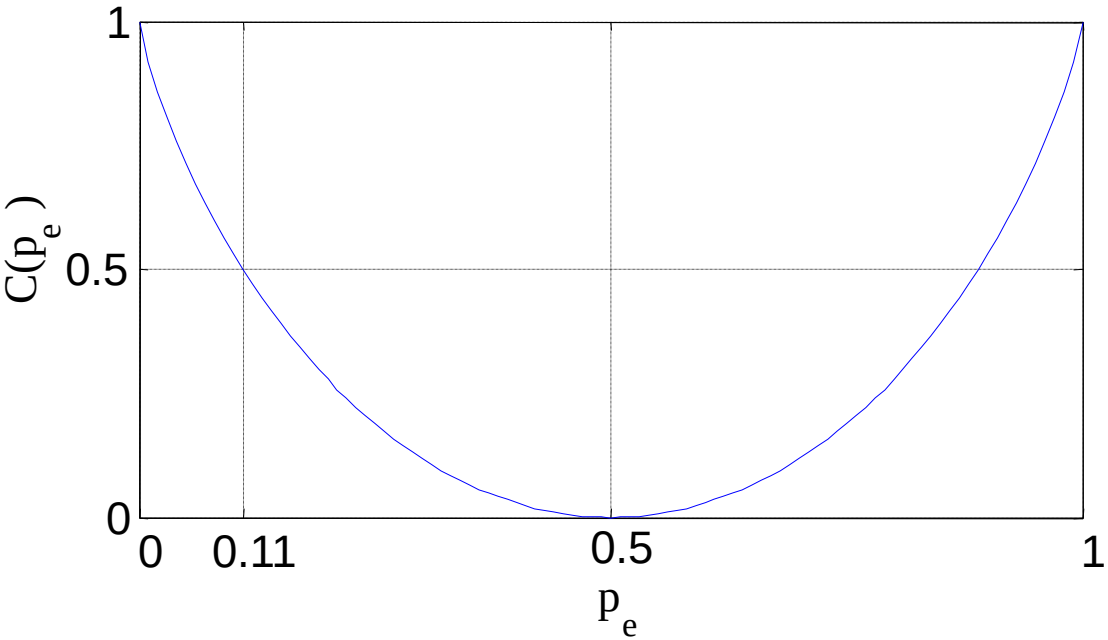
The rate R of any code with arbitrarily small error probability must satisfy

$$R \leq C.$$

Channel Capacity of the Binary Symmetric Channel (BSC)



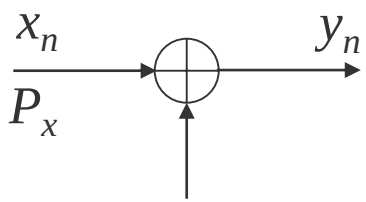
$$C(p_e) = 1 - H_b(p_e)$$
$$= 1 - p_e \log \frac{1}{p_e} - (1-p_e) \log \frac{1}{1-p_e}$$

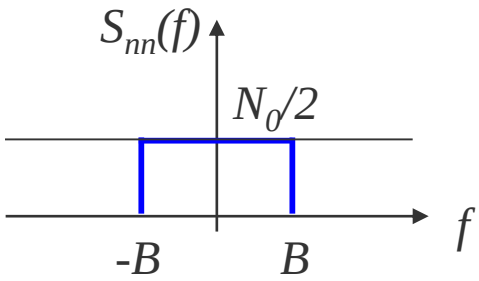


Error-free transmission is possible for a code with rate $R<C$.

E.g. for $p_e=0.11$, we have $C=1/2$, i.e. 100% redundancy is required for error-free transmission.

Channel Capacity of the AWGN Channel





AWGN $\sigma^2 = P_n = \frac{N_0}{2} 2B = N_0 B$

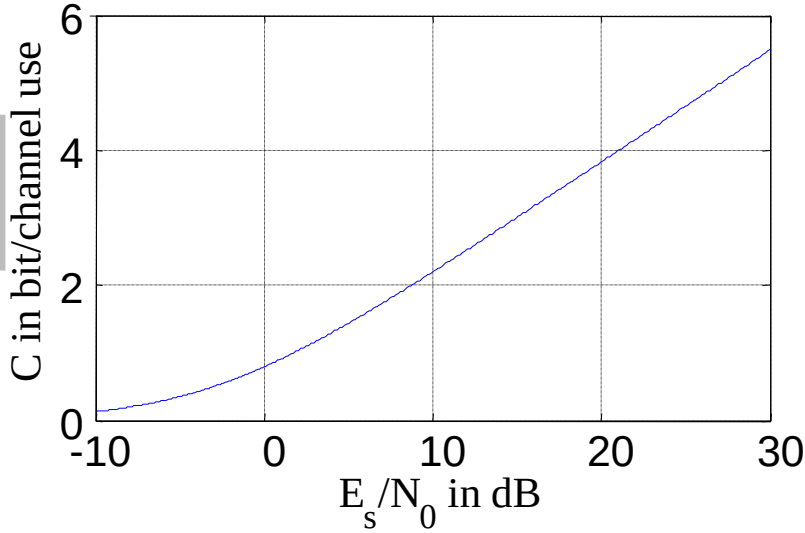
$$C^* = B \log_2\left(1 + \frac{P_x}{P_n}\right) \text{ in } bit/s$$

The optimum transmit symbols x_n are Gaussian distributed with zero mean and $\sigma^2 = P_x$.

Elimination of time:

$$C = C^* T_s = \frac{1}{2} \log_2\left(1 + \frac{2E_s}{N_0}\right) \text{ in bit/channel use}$$

$E_s = P_x T_s$: energy per transmit symbol
$1/T_s = 2B$: Nyquist rate (1 channel use per T_s)



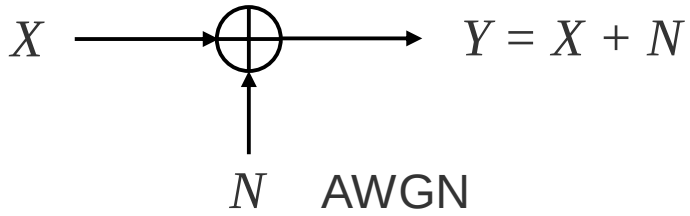
For complex channel:

$$C = \log_2\left(1 + \frac{E_s}{N_0}\right) \text{ in bit/channel use}$$

Derivation of AWGN Channel Capacity

Power constraint on transmit symbols:

$$E\left\{X^2\right\} \le P_X$$



noise power $\sigma_N^2 = P_N$

$$\sigma_Y^2 = E\left\{Y^2\right\} \le P_X + P_N$$

since X and N are uncorrelated and $E\{N\}=0$.

$$I(X;Y) = h(Y) - h(Y|X) = h(Y) - h(X+N|X) = h(Y) - h(N|X) = h(Y) - h(N)$$

X and N are statistically independent

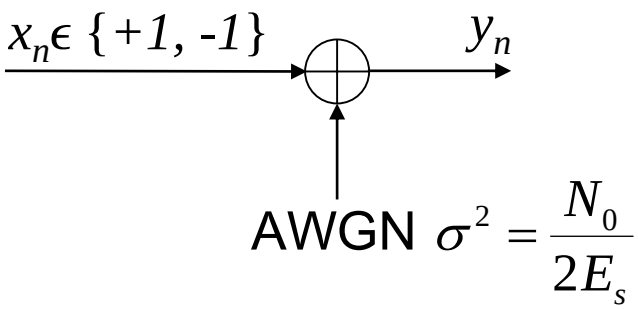
$$= h(Y) - \frac{1}{2}\log_2(2\pi e P_N) \le \frac{1}{2}\log_2\left[2\pi e(P_X + P_N)\right] - \frac{1}{2}\log_2(2\pi e P_N)$$

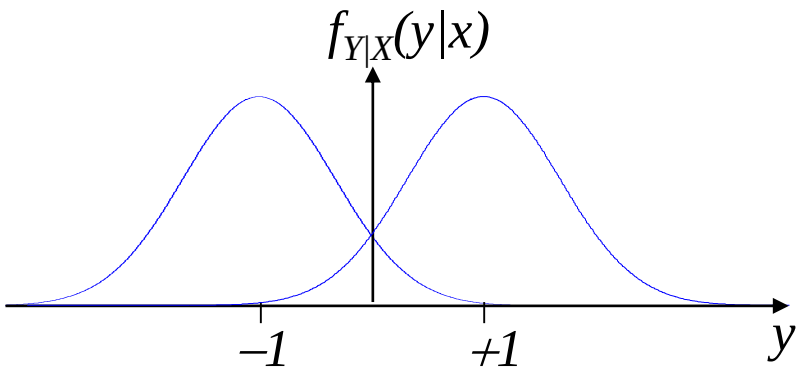
N is normal distributed with variance $\sigma_N^2 = P_N$

A normal distribution of Y maximizes entropy $h(Y)$. Y is normal distributed with variance $\sigma_Y^2 = P_X + P_N$, if **X is normal distributed ($N(0,P_X)$)**.

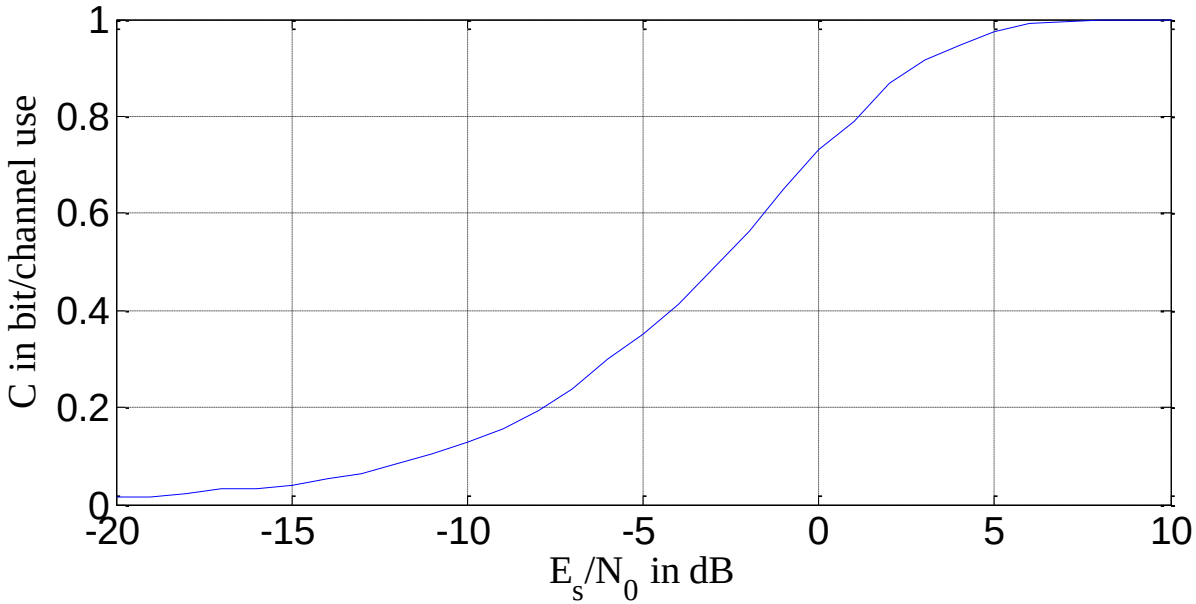
$$C = \max_{f_X(x)} I(X;Y) = \frac{1}{2}\log_2 \frac{2\pi e(P_X + P_N)}{2\pi e P_N} = \frac{1}{2}\log_2\left(1 + \frac{P_X}{P_N}\right)$$

Channel Capacity of the AWGN Channel with Binary Transmit Symbols

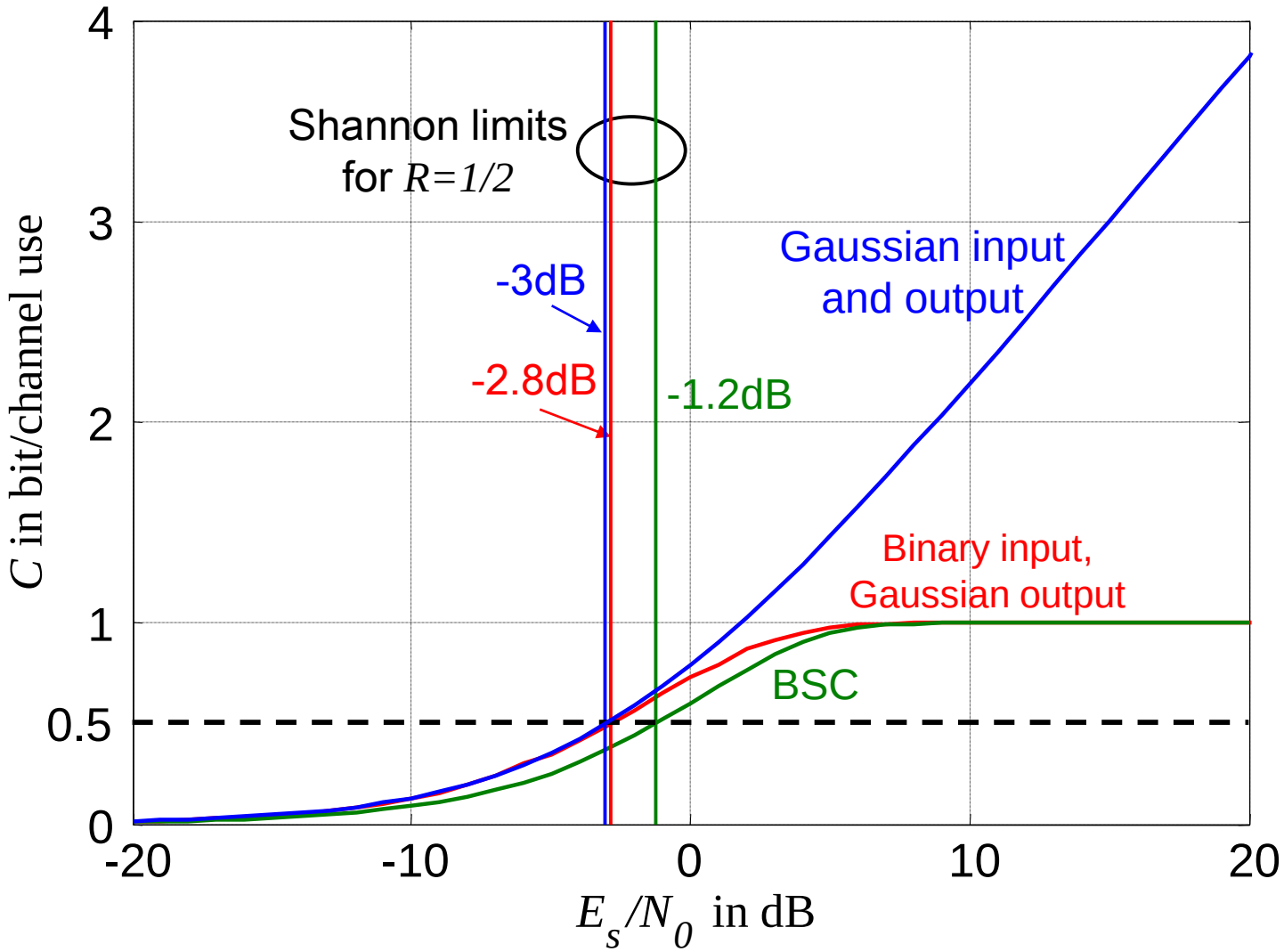




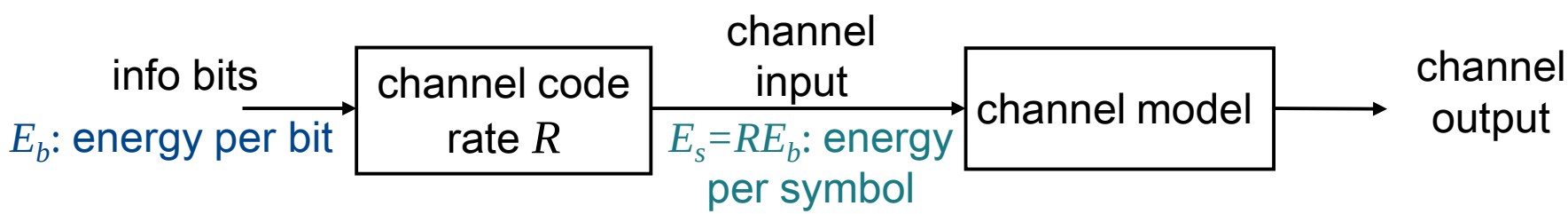
$$C = \frac{1}{2}\int_{-\infty}^{\infty}\left(f_{y|x}(y\,|\,x=+1)\log_2 \frac{f_{y|x}(y\,|\,x=+1)}{f_y(y)} + f_{y|x}(y\,|\,x=-1)\log_2 \frac{f_{y|x}(y\,|\,x=-1)}{f_y(y)} \right) dy \text{ in bit/channel use}$$



Channel Capacity of the AWGN Channel



Absolute Shannon Limit for the AWGN Channel



$$R < C = \frac{1}{2}\log_2\left(1+\frac{2E_s}{N_0}\right) = \frac{1}{2}\log_2\left(1+\frac{2RE_b}{N_0}\right) \Rightarrow 2^{2R} < 1 + 2R\frac{E_b}{N_0}$$

$$\Rightarrow \frac{E_b}{N_0} > \frac{1}{2R}\left(2^{2R}-1\right) = \frac{1}{2R}\left(e^{\log 2^{2R}}-1\right) = \frac{1}{2R}\left(e^{2R\log 2}-1\right)$$

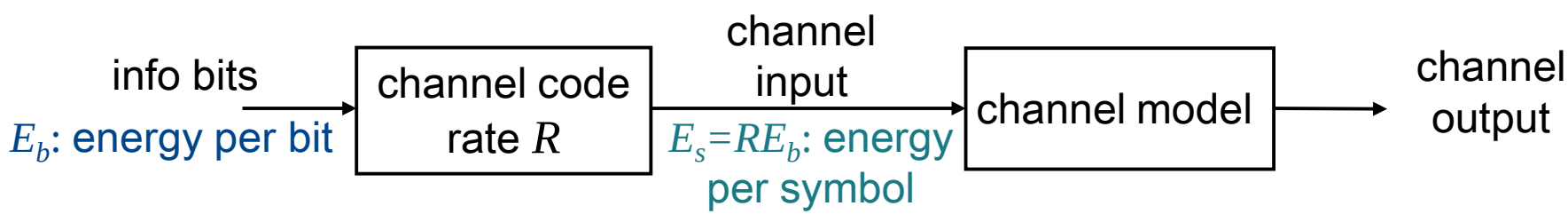
for error-free transmission with rate R

Absolute Shannon limit (lowest possible rate): $R \to 0$

$$\frac{E_b}{N_0} > \frac{1+2R\log 2+\frac{1}{2!}(2R\log 2)^2+\frac{1}{3!}(2R\log 2)^3+\ldots-1}{2R}$$

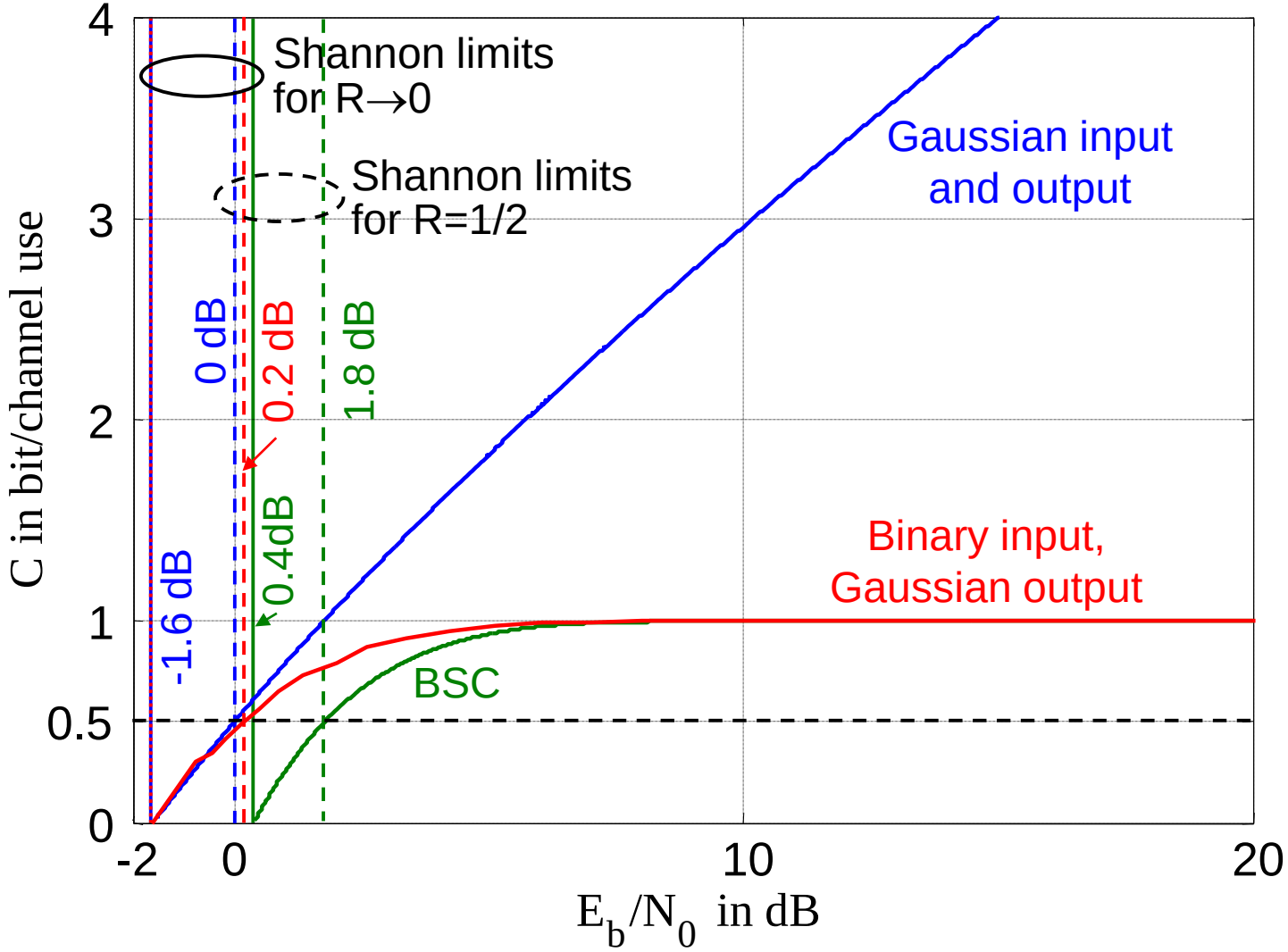
$$= \underbrace{\frac{2R\log 2}{2R}}_{\to \log 2} + \underbrace{\frac{\frac{1}{2!}(2R)^2(\log 2)^2}{2R}}_{\to 0} + \ldots \quad \to \log 2 \qquad \Rightarrow 10\log_{10}\frac{E_b}{N_0}\,dB > -1.6\,dB$$

Shannon Limits for the AWGN Channel

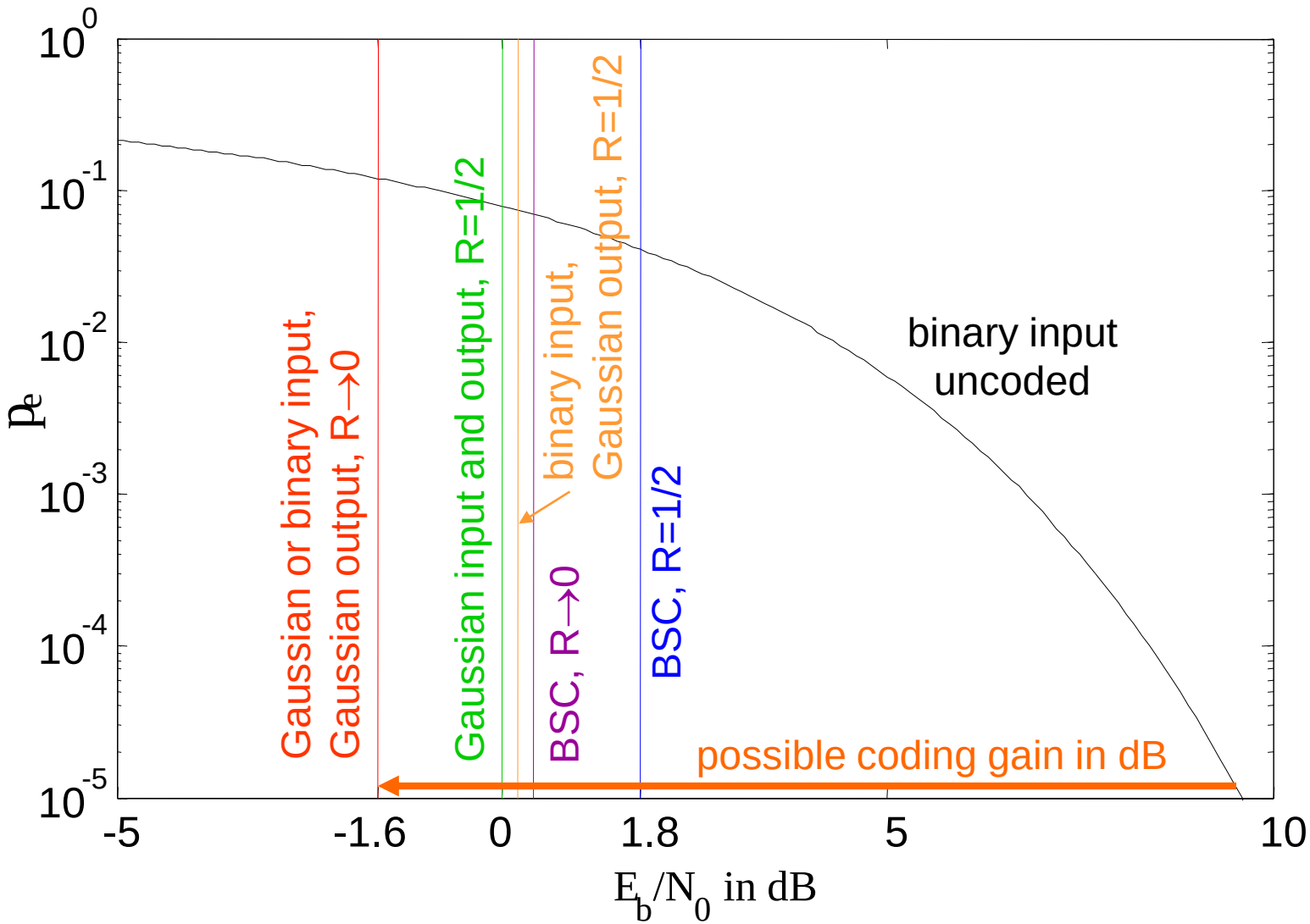


channel	input	output	code rate R	E_b/N_0
AWGN	Gaussian	Gaussian	$\to 0$	-1.6 dB
AWGN	binary	Gaussian	$\to 0$	-1.6 dB
BSC	binary	binary	$\to 0$	0.4 dB
AWGN	Gaussian	Gaussian	1/2	0 dB
AWGN	binary	Gaussian	1/2	0.2 dB
BSC	binary	binary	1/2	1.8 dB

Channel Capacity of the AWGN Channel



Shannon Limits for the AWGN Channel



The binary symmetric channel (BSC) implies a hard decision before decoding. The received values at the output of an AWGN channel contain soft information, i.e. reliability information: Low magnitudes of the received samples are close to the decision threshold and, hence, are less reliable than samples with large magnitudes.

The signal at the decoder input should contain as much information about the info symbols as possible. Hence, a hard decision before decoding means a reduction of information content in the decoder input signal. This explains the smaller capacity of the BSC compared to the binary input AWGN channel.

If soft decision is available at the decoder input, we have a soft-input decoder or soft-decision decoding.

In short, we can say: Soft is better than hard.

Power-Limited and Bandwidth-Limited Transmission (1)

From the channel capacity

$$C^* = B\log_2\left(1+\frac{P_X}{P_N}\right)$$

of an AWGN channel with bandwidth B, noise power P_N and received signal power P_X, we can derive the minimum SNR per information bit, E_b/N_0, which is required in order to achieve a certain bandwidth-efficiency

$$\eta = \frac{R_b}{B} = \frac{R}{T_s B}. \qquad (1)$$

Note, that $R_b = R/T_s$ denotes the data rate in bit/s, whereas R is the code rate in bit per channel use. A channel can be used once per symbol duration T_s. The bandwidth-efficiency η is measured in bit/s/Hz.

We express the received signal power P_X by

$$P_X = \frac{E_s}{T_s},$$

where E_s is the received energy per transmit symbol, and the noise power by

$$P_N = \frac{N_0}{2}\cdot 2B,$$

where $N_0/2$ is the noise power spectral density.

Power-Limited and Bandwidth-Limited Transmission (2)

Then, the channel capacity can be expressed by

$$C^* = B\log_2\left(1+\frac{P_X}{P_N}\right) = B\log_2\left(1+\frac{\frac{E_s}{T_s}}{\frac{N_0}{2}2B}\right) = B\log_2\left(1+\frac{E_s}{T_s B N_0}\right).$$

Using $E_b = R\,E_s$ and the bandwidth-efficiency η from (1), we obtain

$$C^* = B\log_2\left(1+\frac{RE_b}{T_s B N_0}\right) = B\log_2\left(1+\eta\frac{E_b}{N_0}\right).$$

According to Shannon‘s channel coding theorem, the channel capacity C^* is the theoretical limit for the data rate R_b, i.e.

$$R_b = \frac{R}{T_s} < C^* = B\log_2\left(1+\eta\frac{E_b}{N_0}\right).$$

From the equation above, we further obtain

$$\frac{R_b}{B} = \frac{R}{T_s B} = \eta < \log_2\left(1+\eta\frac{E_b}{N_0}\right) \quad \text{or} \quad 2^\eta < 1+\eta\frac{E_b}{N_0}.$$

Power-Limited and Bandwidth-Limited Transmission (3)

Therefore, the minimum required SNR per information bit, E_b/N_0, is given by

$$\frac{E_b}{N_0} > \frac{2^\eta - 1}{\eta}.$$

If we plot the minimum required E_b/N_0 in dB versus the bandwidth-efficiency η, we observe that the required received E_b/N_0 hardly increases with increasing η, as long as the bandwidth-efficiency is smaller than $\eta \approx 1$.
This means that increasing the data rate $R_b = R/T_s$ in this region $\eta <\approx 1$ without changing the bandwidth B requires a proportionally increasing received signal power P_X. This follows from

$$\frac{E_b}{N_0} = \frac{E_s}{RN_0} = \frac{P_X T_s}{RN_0} = \frac{P_X}{R_b N_0} \approx \text{const.}$$

Therefore, the region $\eta <\approx 1$ is called *power-limited region*: Higher data rates require proportionally more received signal power.
In contrast, the required E_b/N_0 increases dramatically with increasing η, if the bandwidth-efficiency is larger than $\eta \approx 1$. The region $\eta >\approx 1$ is called *bandwidth-limited region*, because increasing the data rate R_b without a proportional increase in bandwidth B requires a disproportionally higher received signal power. A proportional increase in bandwidth keeps the bandwidth-efficiency $\eta = R_b/B$ and, hence, the required received E_b/N_0 constant.

Power-Limited and Bandwidth-Limited Transmission (4)

From the discussion above, we may conclude that a bandwidth-efficiency of more than $\eta \approx 1$ is inefficient from an energy per bit point of view. As a conclusion, higher order modulation schemes such as QPSK, 16-QAM etc. seem to be inefficient.
However, in many applications, bandwidth is a much more scarce resource than received energy per bit. This is e.g. the case in cellular communications systems. Therefore, the application of bandwidth-efficient coding and modulation schemes with $\eta > 1$ often makes sense in practice.
One way to achieve the required high per information bit SNR in modern cellular systems is the trend towards small cells with low distance between base station and mobile station. Moreover, sophisticated interference management schemes are developed in order to reduce or avoid interference as interference is usually viewed as part of the noise at a receiver.

Power-Limited and Bandwidth-Limited Transmission (1)

$$C^* = B\log_2\left(1+\frac{P_X}{P_N}\right) = B\log_2\left(1+\frac{\frac{E_s}{T_s}}{\frac{N_0}{2}2B}\right) = B\log_2\left(1+\frac{E_s}{T_s B N_0}\right) = B\log_2\left(1+\frac{R E_b}{T_s B N_0}\right)$$

$$= B\log_2\left(1+\eta\frac{E_b}{N_0}\right)$$

Bandwidth efficiency: $\eta = \frac{R}{T_s B} = \frac{R_b}{B}$ in $\frac{bit}{s\cdot Hz}$

Channel coding theorem:

$$R_b = \frac{R}{T_s} < C^* = B\log_2\left(1+\eta\frac{E_b}{N_0}\right)$$

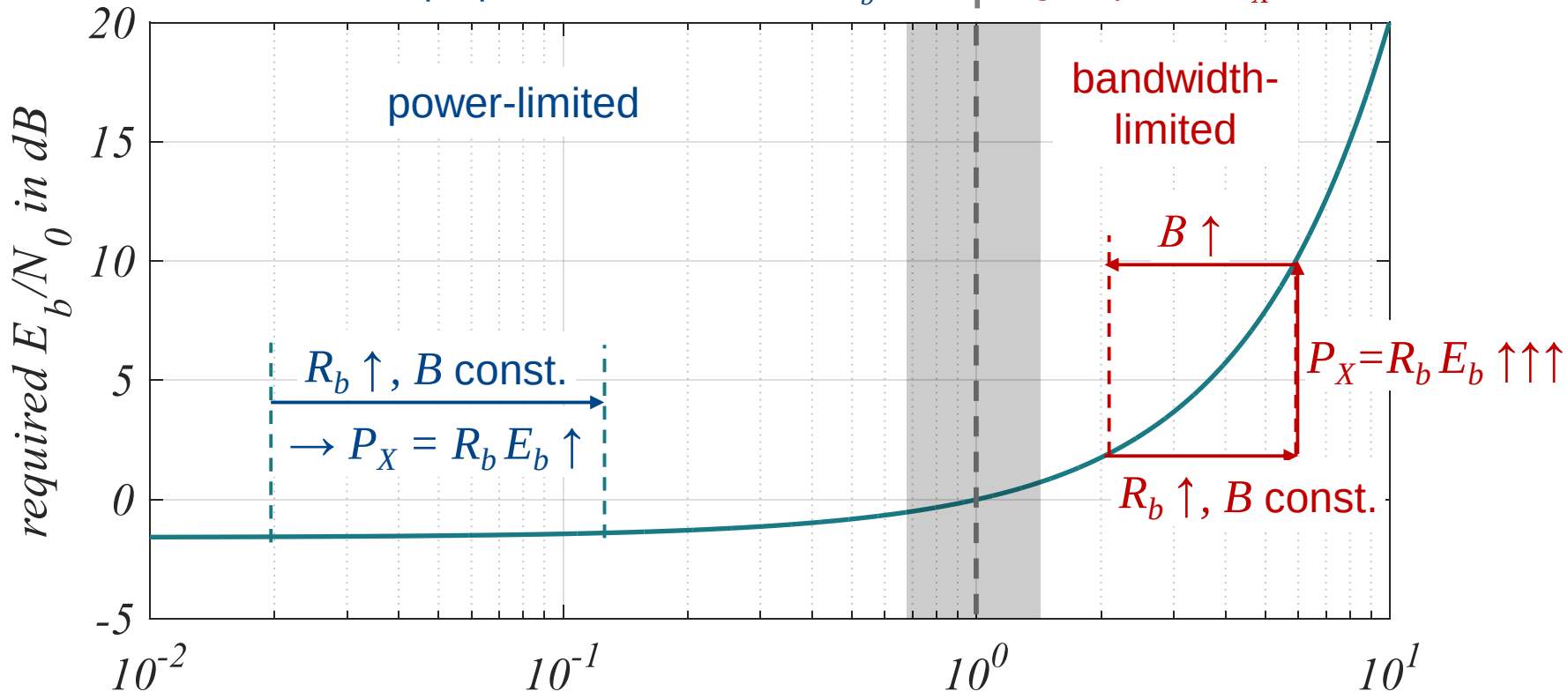
$$\frac{R_b}{B} = \frac{R}{T_s B} = \eta < \log_2\left(1+\eta\frac{E_b}{N_0}\right) \qquad \Rightarrow 2^\eta < 1+\eta\frac{E_b}{N_0}$$

$$\frac{E_b}{N_0} > \frac{2^\eta - 1}{\eta}$$

Power-Limited and Bandwidth-Limited Transmission (2)

Bandwidth needs to be increased proportionally to data rate R_b in order to avoid a disproportionally high increase of required received signal power P_X.

Required received signal power P_X increases proportional to data rate R_b.



bandwidth efficiency $\eta = \frac{R_b}{B} = \frac{R}{T_s B}$ in $\frac{bit}{s \cdot Hz}$

Capacity of Parallel AWGN Channels (1)

In many applications, data is transmitted via parralel channels. The simplest example may be two parallel wires. The parallel channels may also result from frequency division multiplex (FDMA), in which case each of the parallel channels represents a different frequency band. Another example is the transmission scheme Orthogonal Frequency Division Multiplex (OFDM), which is widely applied e.g. for Digital Subscriber Line (DSL), Wireless Local Area Networks (WLAN) and LTE cellular systems.

In OFDM, the data is precoded at the transmitter using an Inverse Discrete Fourier Transform (IDFT) and post-processed at the receiver using a Discrete Fourier Transform (DFT). By doing so, a frequency-selective channel can be elegantly decoupled into a set of parallel channels.

A similar idea can be applied in multiple input multiple output (MIMO) systems. MIMO systems are applied in modern wireless systems such as WLAN and LTE. MIMO uses multiple antennas at both transmitter and receiver. It can be shown that the capacity of a wireless system grows about linearly with the number of transmit antennas, as long as at least the same number of antennas is used at the receiver. The capacity increase can be achieved, if multiple spatial data streams are simultaneously transmitted from the multiple antennas (spatial multiplexing). A MIMO channel can be described by a channel matrix $\mathbf{H}$, which contains the channel coefficients $h^{(nm)}$ from each transmit antenna n to each receive antenna m. If the channel matrix is known to both transmitter and receiver, a singular value decomposition of the channel matrix $\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^H$ can be performed. $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, i.e. $\mathbf{U}\mathbf{U}^H=\mathbf{I}$ and $\mathbf{V}\mathbf{V}^H=\mathbf{I}$, which contain the left and right singular vectors of $\mathbf{H}$, resectively, as columns. The diagonal matrix $\mathbf{\Lambda}$ contains the singular values λ_i of $\mathbf{H}$ on the main diagonal. Similar as in OFDM, the data is precoded using the matrix $\mathbf{V}$ at the transmitter and post-processed using the matrix $\mathbf{U}^H$ at the receiver. This eliminates the interference between the simultaneously transmitted spatial data streams and results in a set of decoupled parallel channels.

Capacity of Parallel AWGN Channels (2)

In both OFDM and MIMO, the parallel channels in general have different noise power P_{Ni}. However, the noise in the parallel channels can be modelled as AWGN with power P_{Ni}, i.e. the noise is independent from channel to channel. The i-th channel can be described by

$$Y_i = X_i + N_i,$$

where the random variables Y_i, X_i and N_i describe the received signal, the transmitted signal and the additive white Gaussian noise, respectively.
In order to determine the capacity of the set of parallel Gaussian channels, we need to find, how the total transmit power P_X should be distributed among the parallel channels and which probability distribution of the transmit symbol vector $\mathbf{X}=[X_1,...,X_M]^T$ maximizes the mutual information $I(X_1, ..., X_M; Y_1, ..., Y_M)$.
The optimization problem can be formulated as

$$C = \max_{f_{X_1,\ldots,X_M}(x_1,\ldots,x_M)} I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right)$$

subject to the transmit power constraint

$$\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M} E\left\{X_i^2\right\} \leq P_X .$$

Capacity of Parallel AWGN Channels (3)

The mutual information can be shown to be upper bounded by

$$I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right) \leq \sum_{i=1}^{M} \frac{1}{2} \log_2 \left(1 + \frac{P_{X_i}}{P_{N_i}}\right),$$

where equality is achieved, if the transmit symbols X_i are statistically independent zero-mean Gaussian random variables, i.e. the transmit vector is distributed according to

$$\left[X_1, \ldots, X_M\right]^T \sim N\left(0, \begin{bmatrix} P_{X_1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & P_{X_M} \end{bmatrix}\right).$$

Hence, the capacity achieving transmit symbols are independent and Gaussian distributed. We still have to find the optimum distribution of the total transmit power P_X among the parallel channels, i.e. the powers P_{Xi} in the expression above given the power constraint

$$\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M} E\left\{X_i^2\right\} \leq P_X.$$

Capacity of Parallel AWGN Channels (4)

This is a standard optimization problem, which can be solved using Lagrange multipliers. The functional can be written as

$$J\left(P_{X_1},\ldots,P_{X_M}\right)=\sum_{i=1}^{M}\frac{1}{2}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right)+\lambda\sum_{i=1}^{M}P_{X_i}.$$

Setting the derivative with respect to P_{Xi} to zero, i.e.

$$\frac{dJ\left(P_{X_1},\ldots,P_{X_M}\right)}{dP_{X_i}}=\frac{1}{2}\frac{1}{1+\frac{P_{X_i}}{P_{N_i}}}\frac{1}{P_{N_i}}+\lambda=\frac{1}{2}\frac{1}{P_{N_i}+P_{X_i}}+\lambda=0,$$

we obtain with the substitution $\Theta = -1/(2\lambda)$ the optimum power in channel i

$$P_{X_i}=\Theta-P_{N_i}.$$

As P_{Xi} is a power, it cannot be negative. Therefore, the expression needs to be modified according to

$$P_{X_i}=\left(\Theta-P_{N_i}\right)^+ \text{, where } \left(x\right)^+=\begin{cases}x, & \text{if } x\geq 0\\ 0, & \text{otherwise}\end{cases},$$

which can be verified using the Kuhn-Tucker conditions.

Capacity of Parallel AWGN Channels (5)

The parameter Θ is chosen such that the transmit power constraint is satisfied, i.e.

$$\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M} \left(\Theta - P_{N_i} \right)^+ = P_X .$$

This solution is known as *waterfilling*, since it has a graphical interpretation as pouring water into a vessel:
Assume, the channels form columns in a vessel and the floor hight of each column is given by the noise power P_{Ni} in the respective channel. The total transmit power corresponds to the water in a can. The water is poured into the vessel and will first fill up the columns with low noise. Only if enough water is available, columns with higher noise power will also be allocated some water. The final water level Θ results, when all water has been poured into the vessel. The hight of the water in each column corresponds to the transmit power $P_{Xi} = (\Theta - P_{Ni})^+$ in the respective channel.
It is an interesting result, that more transmit power should be allocated to good channels, but also channels with higher noise power should be used, if enough total transmit power is available.
The impact of optimum power allocation according to a waterfilling solution on the channel capacity is the higher, the lower the SNR is. At high SNR, the noise power differences in the parallel channels will be small compared to the transmit power P_{Xi} in each channel. Therefore, a uniform power allocation, i.e. $P_{Xi} = P_X/M$ for all $i=1,...,M$, will yield only a negligible degradation in terms of achievable rate.

Capacity of Parallel AWGN Channels (6)

In summary, the capacity of a set of M parallel channels with independent additive white Gaussian noise with power P_{Ni} in channel i is given by

$$C = \frac{1}{2}\sum_{i=1}^{M}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right),$$

where the transmit symbols X_i are independent Gaussian distributed random variables and the transmit power shall be allocated to the channels according to a waterfilling solution

$$P_{X_i} = \left(\Theta - P_{N_i}\right)^+,$$

which satisfies the transmit power constraint

$$\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M}\left(\Theta - P_{N_i}\right)^+ = P_X .$$

Waterfilling and Bit-Loading

The waterfilling solution for optimum power allocation in parrallel Gaussian channels implies Gaussian tranmit symbols. A Gaussian transmit symbol alphabet is a theoretical assumption which can only approximately be realized in practice by using high order modulation schemes in combination with so-called signal shaping methods.

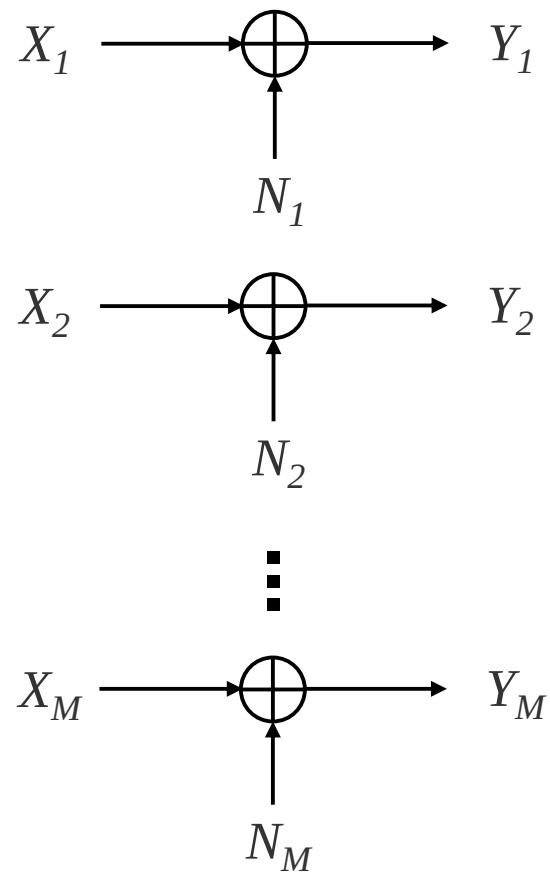
Contemporary communications systems, however, use modulation schemes with a limited number of possible transmit symbols such as PSK or QAM. As a consequence of the limited transmit symbol alphabet, the number of bits per channel use is also limited. Therefore, the achievable rate for a certain transmit symbol alphabet will saturate for high SNR as it is impossible to receive more information bits per transmit symbol than have been transmitted. E.g. with QPSK transmit symbols, the achievable rate will saturate at 2 bit/channel use. When the SNR is high enough for an achievable rate of (almost) 2 bit/channel use, putting more transmit power in the respective channel does not make sense but is a waste of resources. This is unlike Gaussian transmit symbols, for which the capacity is not bounded for SNR $\rightarrow\infty$. Therefore, the waterfilling solution is not an appropriate approach in case of limited transmit symbol alphabets. Instead or as a complement of waterfilling, so-called bit-loading should be applied . The idea is to transmit more bits per transmit symbol on good channels and less bits per transmit symbol on poor channels. The adaptation is not done via power allocation but can be done using different modulation schemes on channels of different quality, i.e. BPSK on poor channels, QPSK on fair channels and 16 QAM on good channels.

A number of bit-loading algorithms exist for the allocation of bits to parallel channels.

In summary, bit-loading can be considered a counterpart of waterfilling in case of a limited transmit symbol alphabet.

Capacity of Parallel AWGN Channels

$$C = \frac{1}{2}\sum_{i=1}^{M}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right)$$



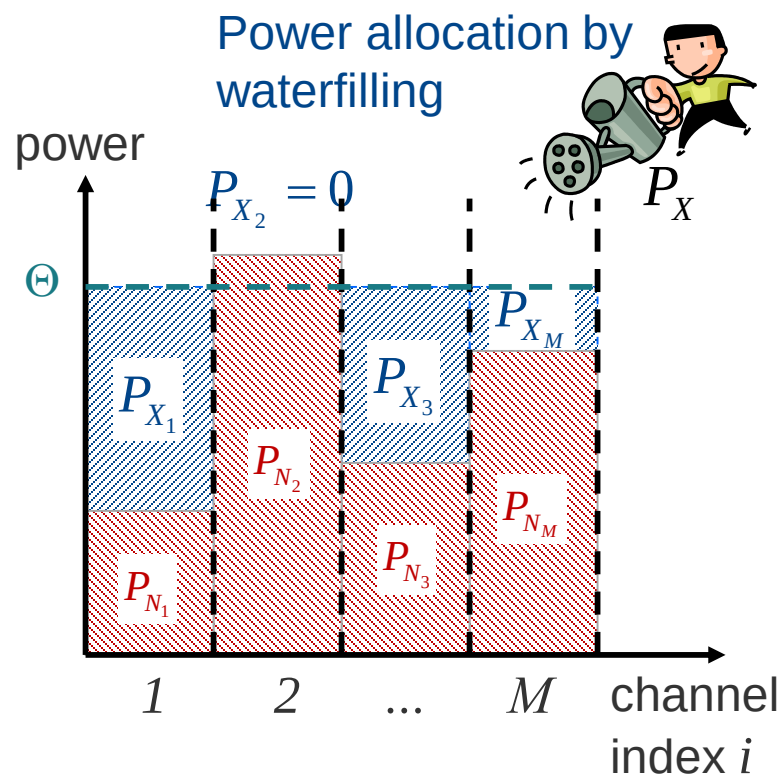
- Noise power in channel i: P_{N_i}
- Total transmit power: $P_X = \sum_{i=1}^{M} P_{X_i}$
- Transmit power which is allocated to channel i:

$$P_{X_i} = \left(\Theta - P_{N_i}\right)^+ = \begin{cases} \Theta - P_{N_i}, & \text{for } \Theta - P_{N_i} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Θ is the solution of the *waterfilling problem* $\sum_{i=1}^{M}\left(\Theta - P_{N_i}\right)^+ = P_X$

Capacity of Parallel AWGN Channels: Waterfilling

$$C = \frac{1}{2}\sum_{i=1}^{M}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right)$$



- Noise power in channel i: P_{N_i}
- Total transmit power: $P_X = \sum_{i=1}^{M} P_{X_i}$
- Transmit power which is allocated to channel i:

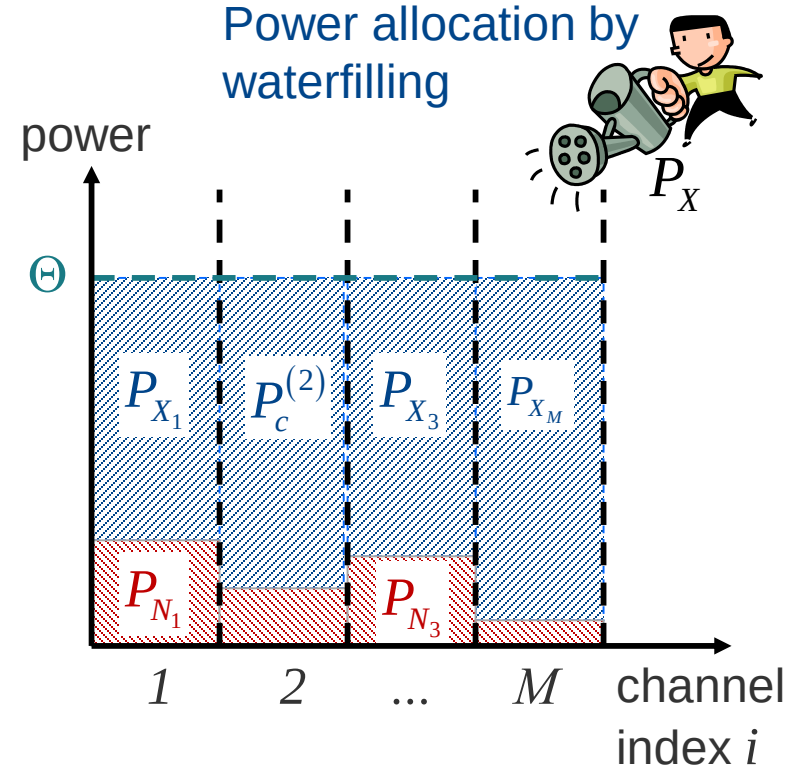
$$P_{X_i} = \left(\Theta - P_{N_i}\right)^+ = \begin{cases} \Theta - P_{N_i}, & \text{for } \Theta - P_{N_i} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Θ is the solution of the *waterfilling problem* $\sum_{i=1}^{M}\left(\Theta - P_{N_i}\right)^+ = P_X$

Capacity of Parallel AWGN Channels: Waterfilling – High SNR

$$C = \frac{1}{2}\sum_{i=1}^{M}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right)$$

Power allocation by waterfilling

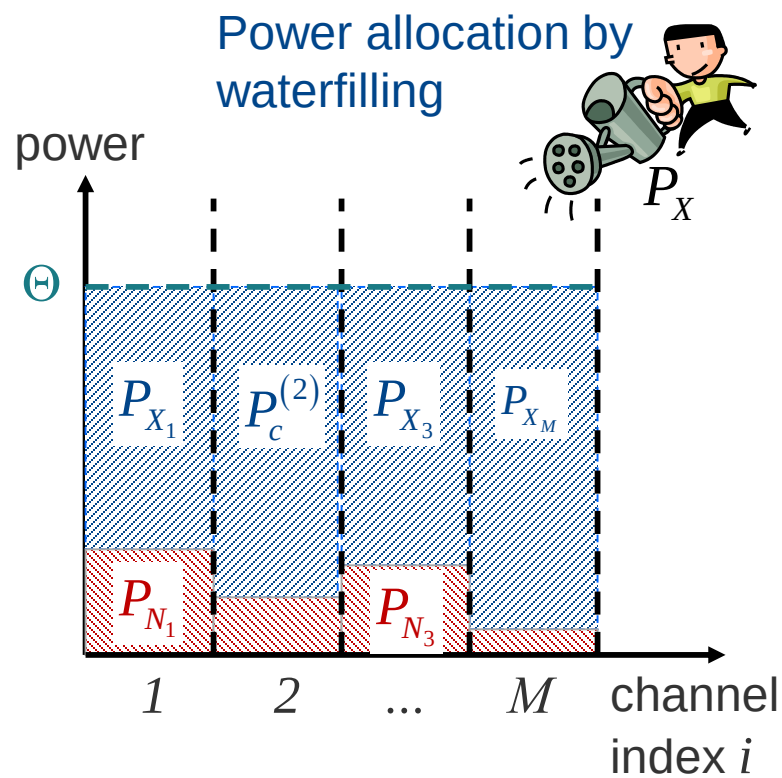


- Noise power in channel i: P_{N_i}
- Total transmit power: $P_X = \sum_{i=1}^{M} P_{X_i}$
- Transmit power which is allocated to channel i:

$$P_{X_i} = \left(\Theta - P_{N_i}\right)^+ = \begin{cases} \Theta - P_{N_i}, & \text{for } \Theta - P_{N_i} > 0 \\ 0, & \text{otherwise} \end{cases}$$

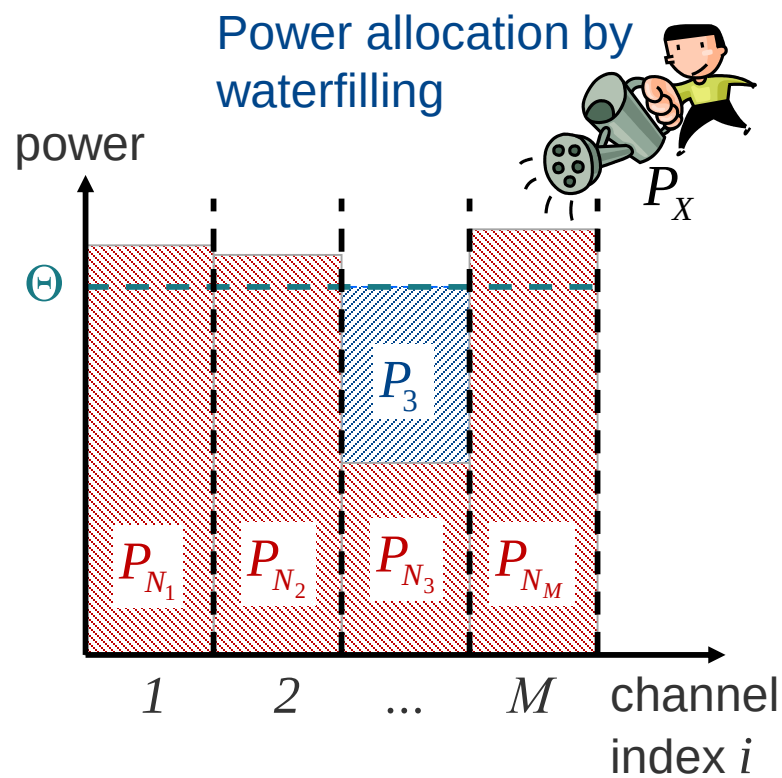
- Θ is the solution of the *waterfilling problem* $\sum_{i=1}^{M}\left(\Theta - P_{N_i}\right)^+ = P_X$

Capacity of Parallel AWGN Channels: Waterfilling – High SNR



At high SNR, the gain of waterfilling over uniform transmit power distribution is decreased.

Capacity of Parallel AWGN Channels: Waterfilling – Low SNR

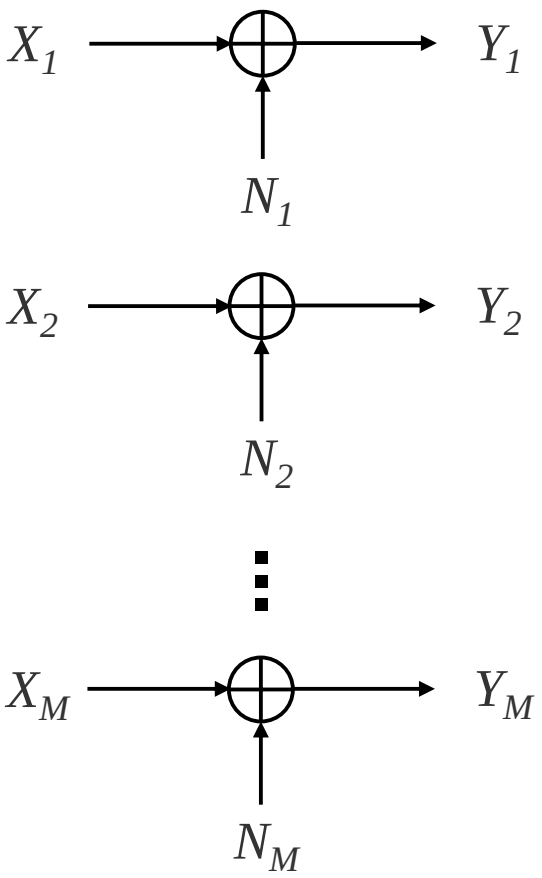


At low SNR, waterfilling allocates the total transmit power to the strongest of the parallel channels.

Derivation of Capacity of Parallel AWGN Channels (1)

$$C = \max_{f_{X_1,\ldots,X_M}(x_1,\ldots,x_M)} I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right)$$

Power constraint: $\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M} E\left\{X_i^2\right\} \leq P_X$



Derivation of Capacity of Parallel AWGN Channels (2)

$$I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right) =$$

$$= h\left(Y_1, Y_2, \ldots, Y_M\right) - h\left(Y_1, Y_2, \ldots, Y_M \middle| X_1, X_2, \ldots, X_M\right)$$

$$= h\left(Y_1, Y_2, \ldots, Y_M\right) - h\left(N_1, N_2, \ldots, N_M \middle| X_1, X_2, \ldots, X_M\right)$$

$Y_i = X_i + N_i$

$$= h\left(Y_1, Y_2, \ldots, Y_M\right) - h\left(N_1, N_2, \ldots, N_M\right)$$

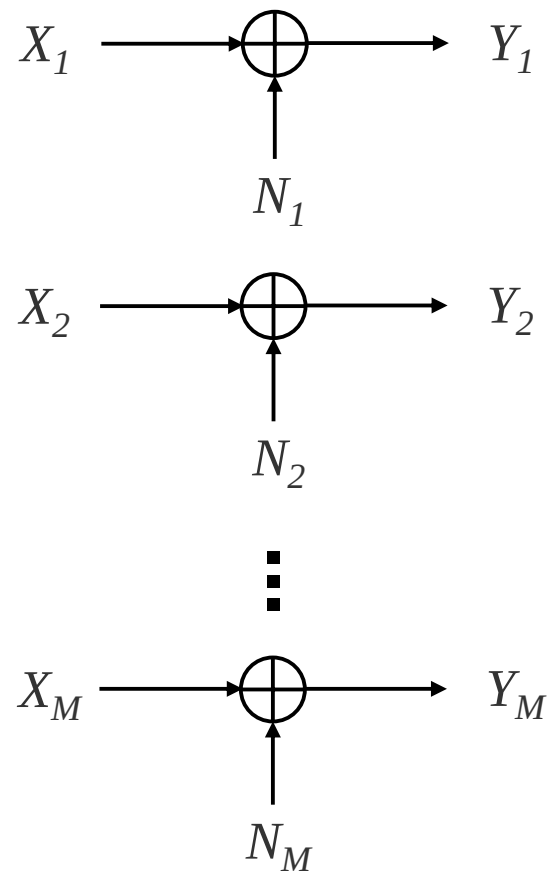
X_i and N_i are statistically independent

$$= h\left(Y_1, Y_2, \ldots, Y_M\right) - \sum_{i=1}^{M} h\left(N_i\right)$$

N_i and N_j are statistically independent for $i \neq j$ (AWGN)

$$= \sum_{i=1}^{M} h\left(Y_i \middle| Y_1, Y_2, \ldots, Y_{i-1}\right) - \sum_{i=1}^{M} h\left(N_i\right)$$

chain rule of entropy



Derivation of Capacity of Parallel AWGN Channels (3)

$$I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right) =$$

$$= \sum_{i=1}^{M} h\left(Y_i \middle| Y_1, Y_2, \ldots, Y_{i-1}\right) - \sum_{i=1}^{M} h\left(N_i\right)$$

$$\leq \sum_{i=1}^{M} h\left(Y_i\right) - \sum_{i=1}^{M} h\left(N_i\right)$$

Conditioning reduces entropy.

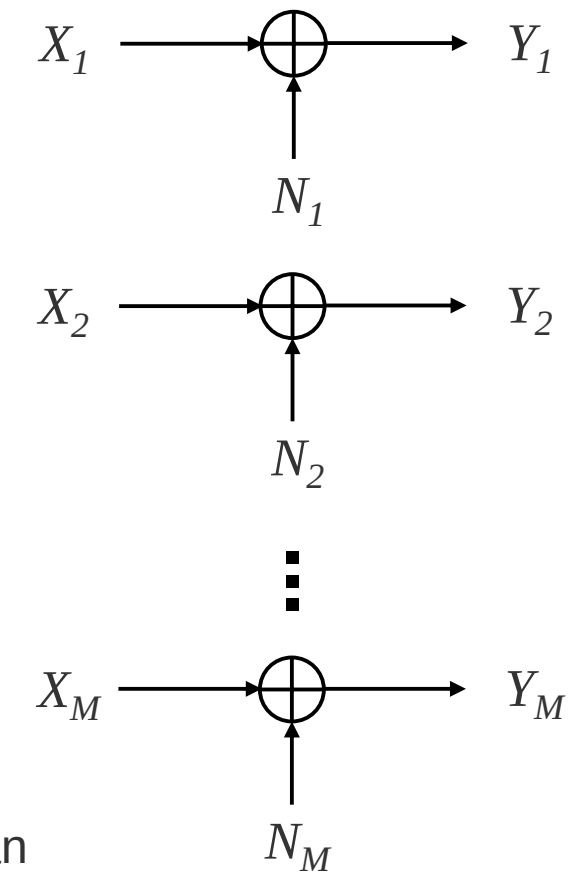
Equality holds, if Y_i and Y_j are statistically independent for $i \neq j$, i.e. if X_i and X_j are statistically independent.

$$\leq \sum_{i=1}^{M} \frac{1}{2} \log_2 \left(1 + \frac{P_{X_i}}{P_{N_i}}\right)$$

Gaussian distribution maximizes entropy.

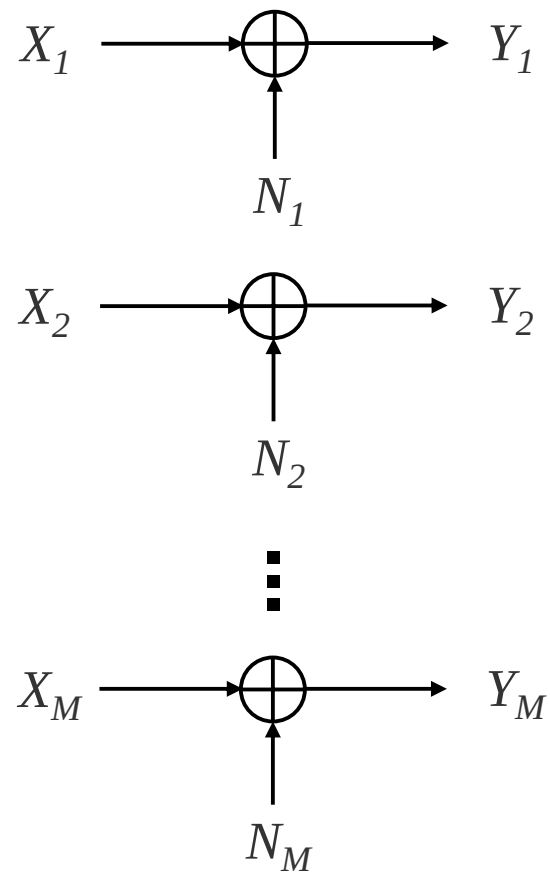
Equality holds, if X_i are statistically independent Gaussian distributed, i.e. if

$$\left[X_1, \ldots, X_M\right]^T \sim N\left(0, \begin{bmatrix} P_{X_1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & P_{X_M} \end{bmatrix}\right).$$



Derivation of Capacity of Parallel AWGN Channels (4)

$$I\left(X_1, X_2, \ldots, X_M; Y_1, Y_2, \ldots, Y_M\right) \le \sum_{i=1}^{M} \frac{1}{2} \log_2 \left(1 + \frac{P_{X_i}}{P_{N_i}}\right)$$



Find optimum transmit power allocation, which maximizes $I(X_1,\ldots, X_M,; Y_1,\ldots, Y_M)$ subject to

$$\sum_{i=1}^{M} P_{X_i} = \sum_{i=1}^{M} E\left\{X_i^2\right\} \le P_X$$

→ Lagrange multipliers:

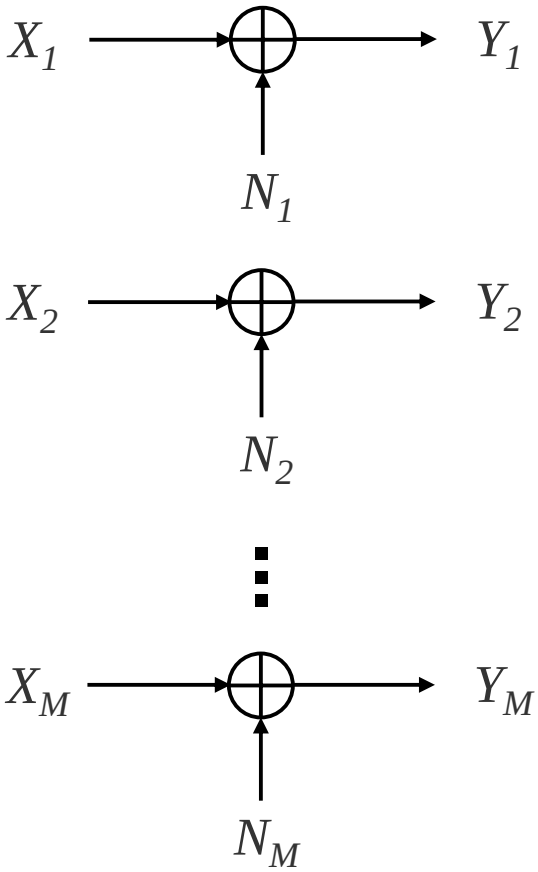
$$J\left(P_{X_1}, \ldots, P_{X_M}\right) = \sum_{i=1}^{M} \frac{1}{2} \log_2 \left(1 + \frac{P_{X_i}}{P_{N_i}}\right) + \lambda \sum_{i=1}^{M} P_{X_i}$$

$$\frac{dJ\left(P_{X_1}, \ldots, P_{X_M}\right)}{dP_{X_i}} = \frac{1}{2} \frac{1}{1 + \frac{P_{X_i}}{P_{N_i}}} \frac{1}{P_{N_i}} + \lambda = \frac{1}{2} \frac{1}{P_{N_i} + P_{X_i}} + \lambda = 0$$

$$\Rightarrow P_{N_i} + P_{X_i} = \underbrace{-\frac{1}{2\lambda}}_{\Theta} \quad \Rightarrow P_{X_i} = \Theta - P_{N_i}$$

Derivation of Capacity of Parallel AWGN Channels (5)

$$C = \frac{1}{2}\sum_{i=1}^{M}\log_2\left(1+\frac{P_{X_i}}{P_{N_i}}\right)$$



Optimum transmit power allocation:

$$P_{X_i} = \Theta - P_{N_i}$$

But: Transmit power cannot be negative !

$$P_{X_i} = \left(\Theta - P_{N_i}\right)^+ = \begin{cases} \left(\Theta - P_{N_i}\right)^+, & \text{if } \Theta - P_{N_i} \\ 0, & \text{otherwise} \end{cases}$$

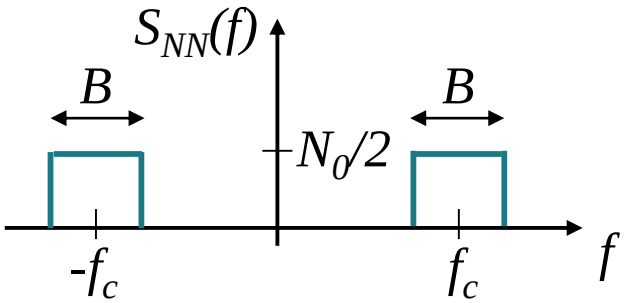
(*waterfilling solution*)

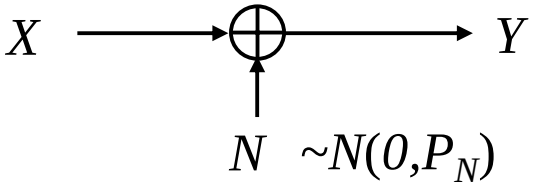
The *water level* Θ is chosen such that

$$\sum_{i=1}^{M} P_{X_i} = \left(\Theta - P_{N_i}\right)^+ = P_X$$

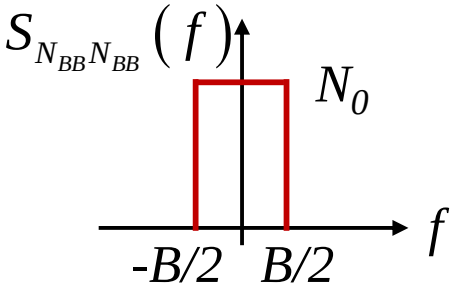
Capacity of Parallel AWGN Channels: Complex Equivalent Baseband AWGN Channel

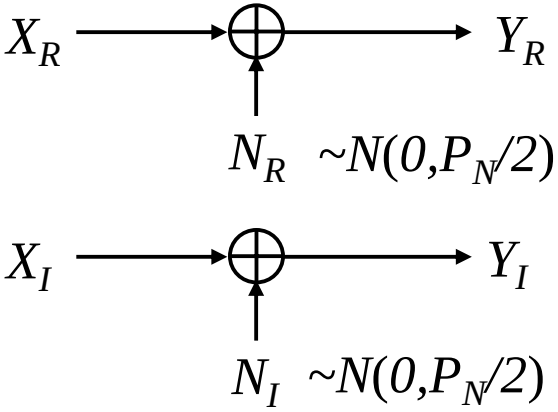
Real bandpass AWGN channel model:





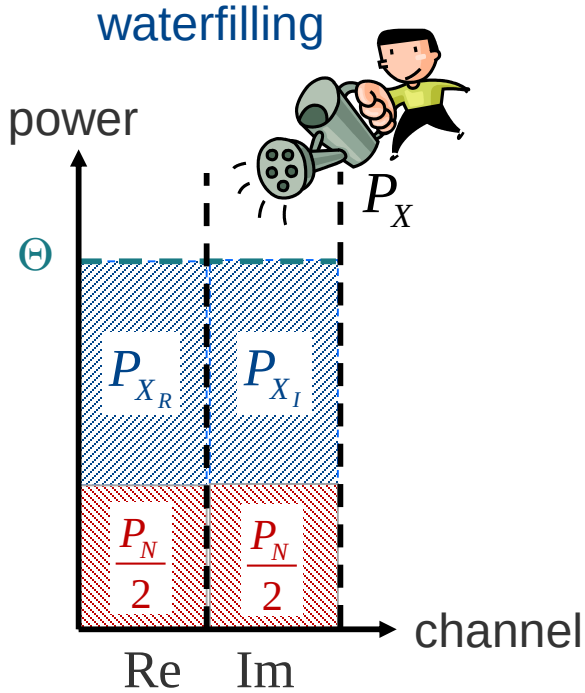
Complex equivalent baseband AWGN channel model:





Capacity of Parallel AWGN Channels: Complex Equivalent Baseband AWGN Channel

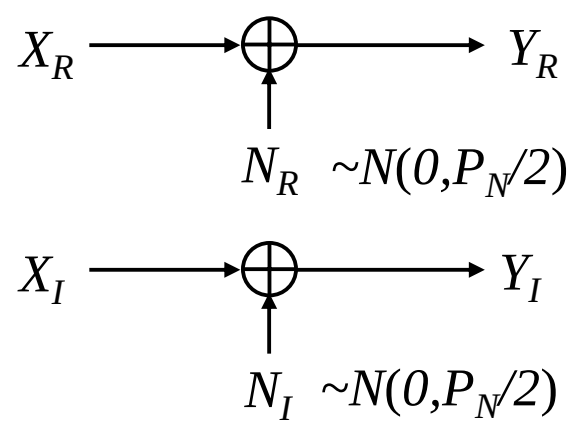
Power allocation by waterfilling



$$C = \frac{1}{2}\log_2\left(1+\frac{P_{X_R}}{P_{N_R}}\right)+\frac{1}{2}\log_2\left(1+\frac{P_{X_I}}{P_{N_I}}\right)$$

$$= 2\cdot\frac{1}{2}\log_2\left(1+\frac{P_X/2}{P_N/2}\right)$$

$$\boxed{C = \log_2\left(1+\frac{P_X}{P_N}\right)}$$



The optimum transmit power allocation is a uniform distribution, i.e.

$$P_{X_R} = P_{X_I} = \frac{P_X}{2}$$

Multiple Input Multiple Output (MIMO) Systems

Antennas Inside iPhone 6

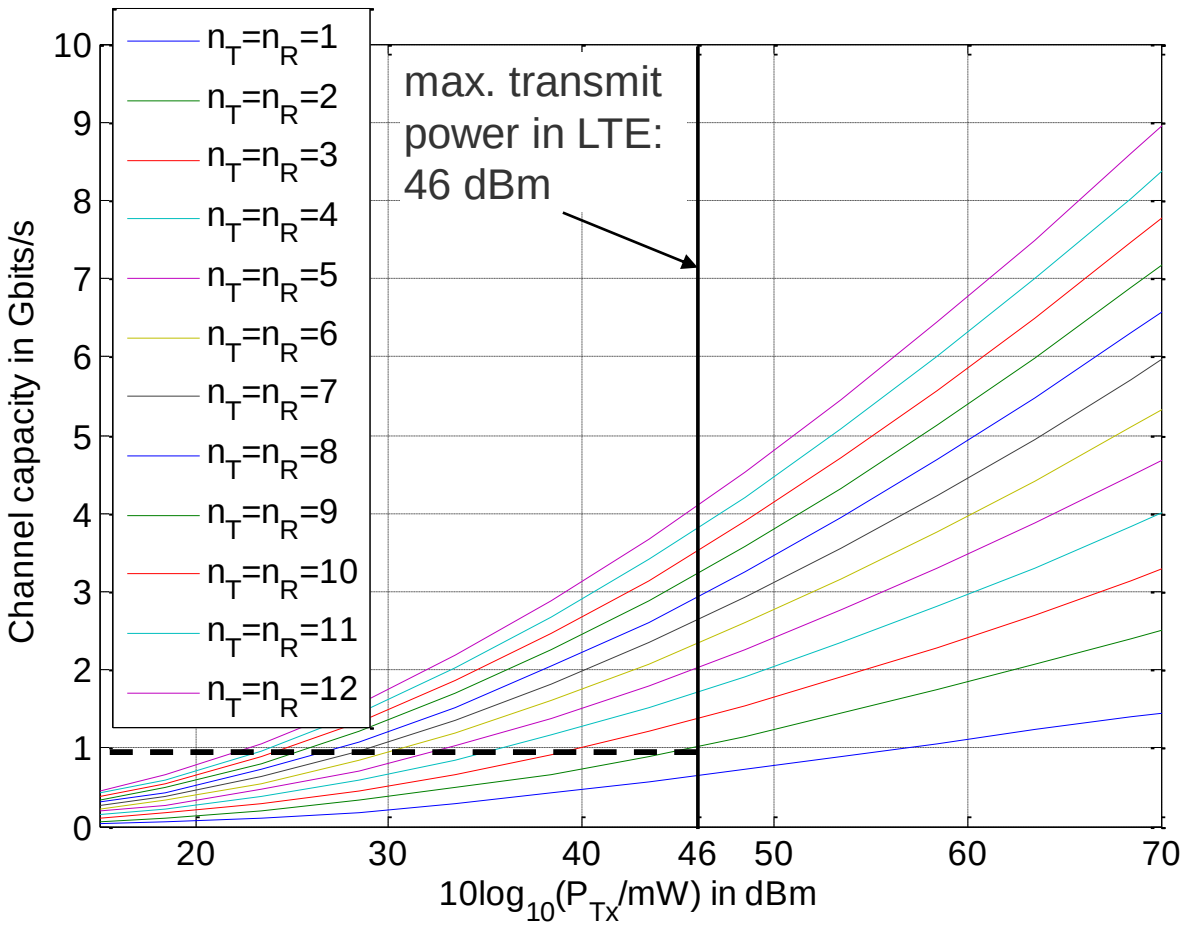
Two antennas for
cellular communications

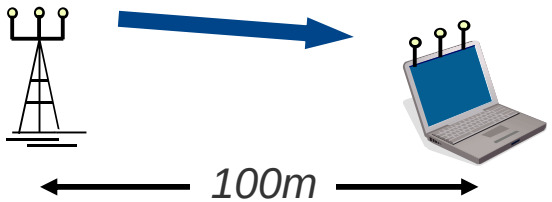
Capacity of Single-User (SU) MIMO

Carrier frequency: f_c=3.8 GHz
Bandwidth: B = 100 MHz
Transmit power: P_{Tx}
n_T transmit antennas
n_R receive antennas

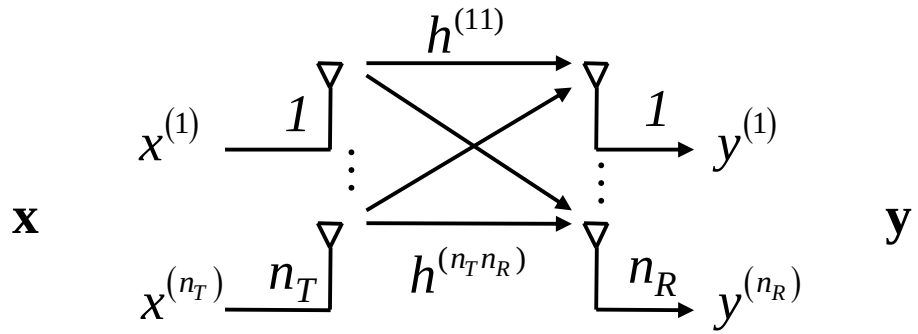
WINNER channel model:
Urban macro cell, non-line-of-sight (NLOS)

Use multiple antennas at transmitter (and receiver)
⇒ capacity increases significantly in fading environment





Capacity of Parallel AWGN Channels: MIMO Channel with Channel Knowledge at the Transmitter (CSIT) (1)



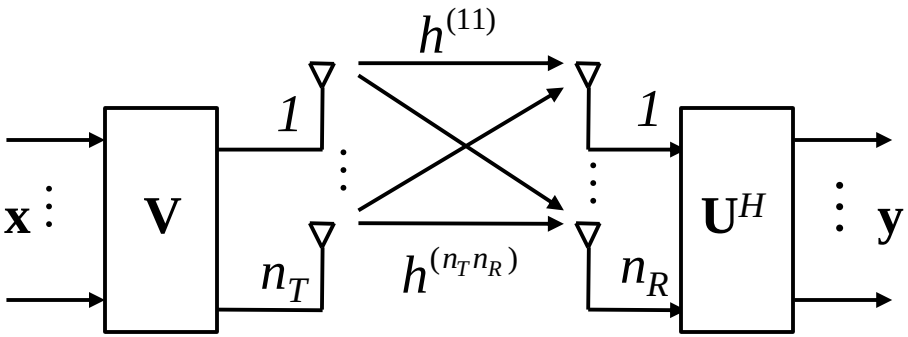
MIMO:
Multiple input multiple output

$$\mathbf{y} = \mathbf{H}\,\mathbf{x} + \mathbf{n} = \underbrace{\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^H}_{\mathbf{H}}\,\mathbf{x} + \mathbf{n}$$

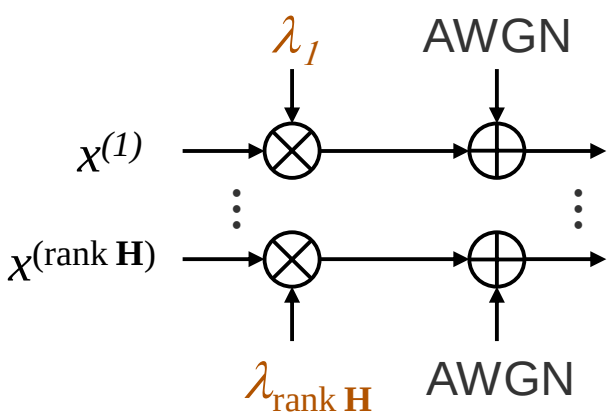
$$\underbrace{\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n_R)} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} h^{(11)} & \cdots & h^{(n_T 1)} \\ \vdots & \ddots & \vdots \\ h^{(1 n_R)} & \cdots & h^{(n_T n_R)} \end{bmatrix}}_{\mathbf{H} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^H} \underbrace{\begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(n_T)} \end{bmatrix}}_{\mathbf{y}} + \underbrace{\begin{bmatrix} n^{(1)} \\ \vdots \\ n^{(n_R)} \end{bmatrix}}_{\mathbf{n}}$$

singular value decomposition (SVD)

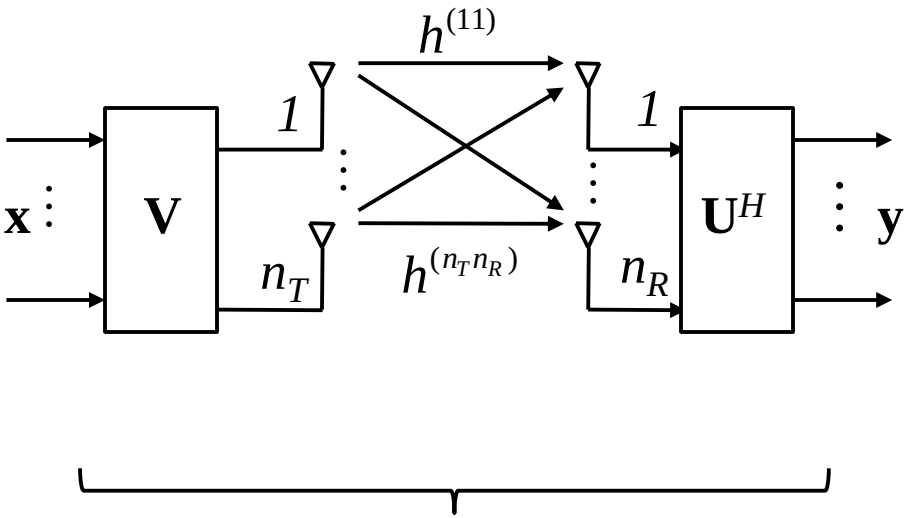
Capacity of Parallel AWGN Channels: MIMO Channel with Channel Knowledge at the Transmitter (CSIT) (2)

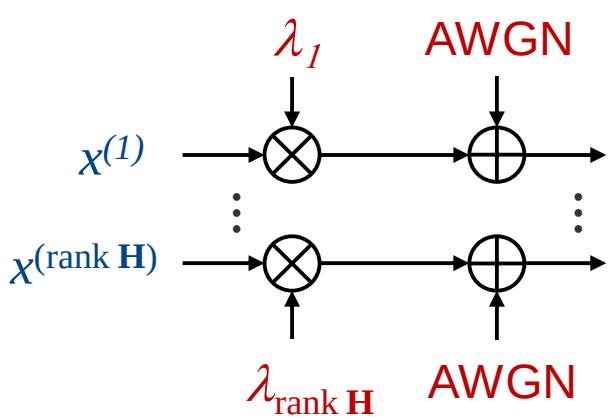


$$\mathbf{y} = \mathbf{U}^H \mathbf{H} \mathbf{V} \mathbf{x} + \mathbf{U}^H \mathbf{n} = \underbrace{\mathbf{U}^H \mathbf{U}}_{\mathbf{I}_{n_R}} \boldsymbol{\Lambda} \underbrace{\mathbf{V}^H \mathbf{V}}_{\mathbf{I}_{n_T}} \mathbf{x} + \underbrace{\mathbf{U}^H \mathbf{n}}_{\text{AWGN}} = \begin{bmatrix} \lambda_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \lambda_{\min\{n_T, n_R\}} \end{bmatrix} \mathbf{x} + \mathbf{U}^H \mathbf{n}$$



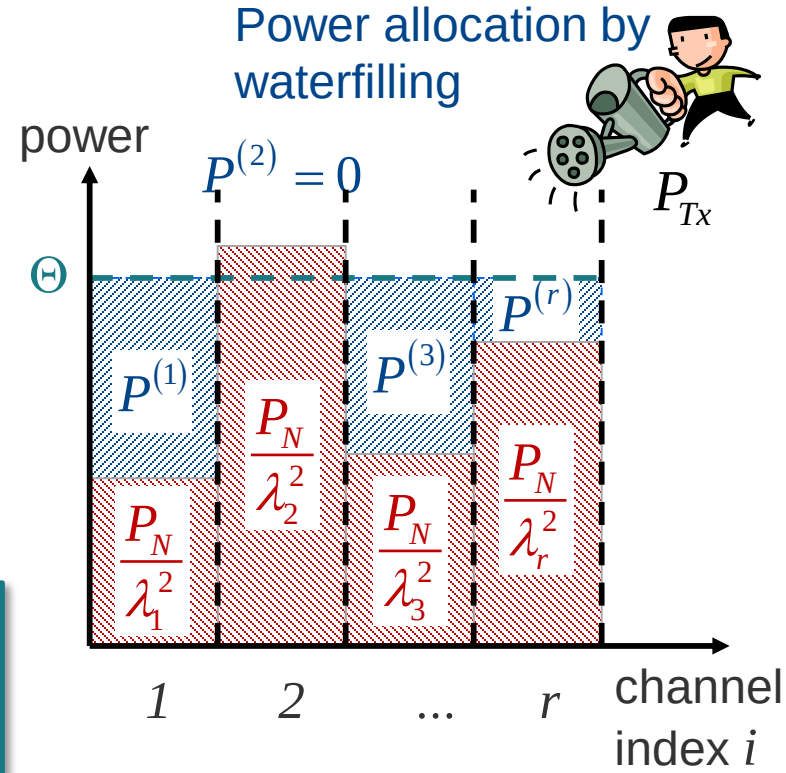
Capacity of Parallel AWGN Channels: MIMO Channel with Channel Knowledge at the Transmitter (CSIT) (3)





$$C = \sum_{i=1}^{\text{rank}\mathbf{H}} \log_2\left(1 + \frac{P^{(i)}\lambda_i^2}{P_N}\right) = \sum_{i=1}^{\text{rank}\mathbf{H}} \left[\log_2\left(\frac{\Theta\lambda_i^2}{P_N}\right)\right]^+$$

Power allocation by waterfilling



Singular Value Decomposition (SVD) of a Matrix

$$\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$$

$\mathbf{A}$: $M \times N$ matrix
$\mathbf{v}_i$: $M \times 1$ right singular vector of $\mathbf{A}$, $i=1,\ldots,M$
$\mathbf{u}_j$: $N \times 1$ left singular vector of $\mathbf{A}$, $j=1,\ldots,N$
σ_l: singular value, $l=1,\ldots,\min\{M,N\}$

Singular value decomposition of $\mathbf{A}$: $\boxed{\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H}$

where

$$\Sigma = \begin{bmatrix} \sigma_1 & & \mathbf{0} & \mathbf{0} \\ & \ddots & & \vdots \\ \mathbf{0} & & \sigma_{\min\{M,N\}} & \mathbf{0} \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_M \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_N \end{bmatrix}$$

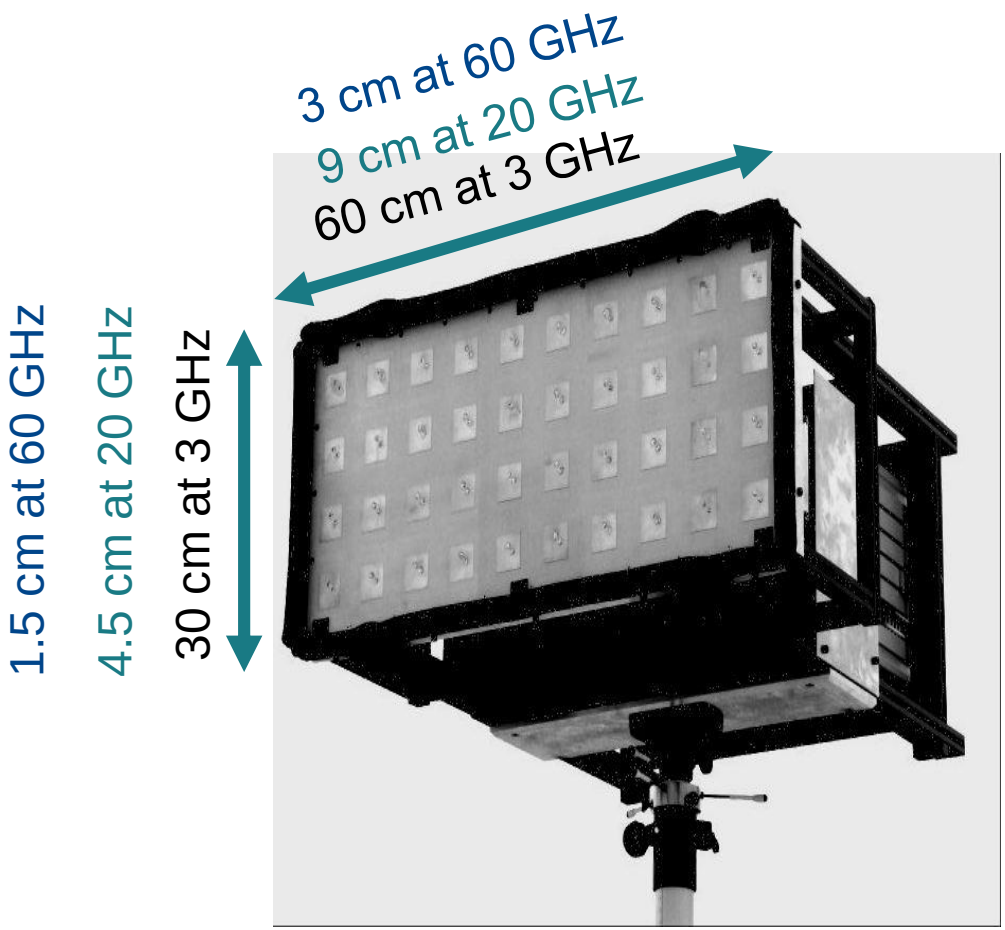
$$\mathbf{U}\mathbf{U}^H = \mathbf{I}_M \qquad \mathbf{V}\mathbf{V}^H = \mathbf{I}_N$$

$M \times N$ diagonal matrix with nonnegative real singular values σ_i of $\mathbf{A}$ on the diagonal.

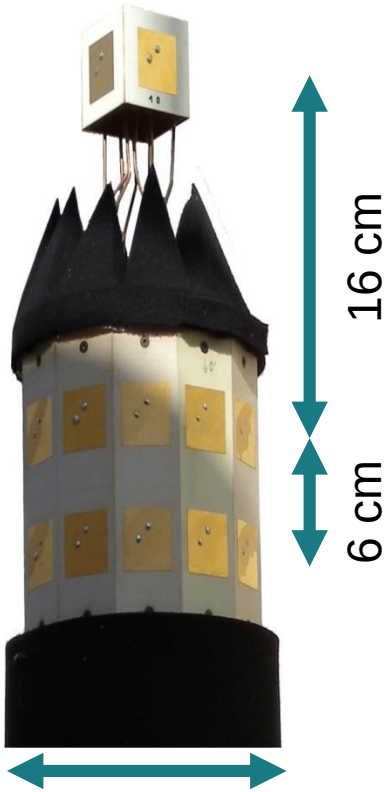
Unitary $M \times M$ matrix with right singular vectors $\mathbf{u}_i$ of $\mathbf{A}$ as columns.

Unitary $N \times N$ matrix with left singular vectors $\mathbf{v}_i$ of $\mathbf{A}$ as columns.

Massive MIMO:
Up to hundreds of antennas at base stations



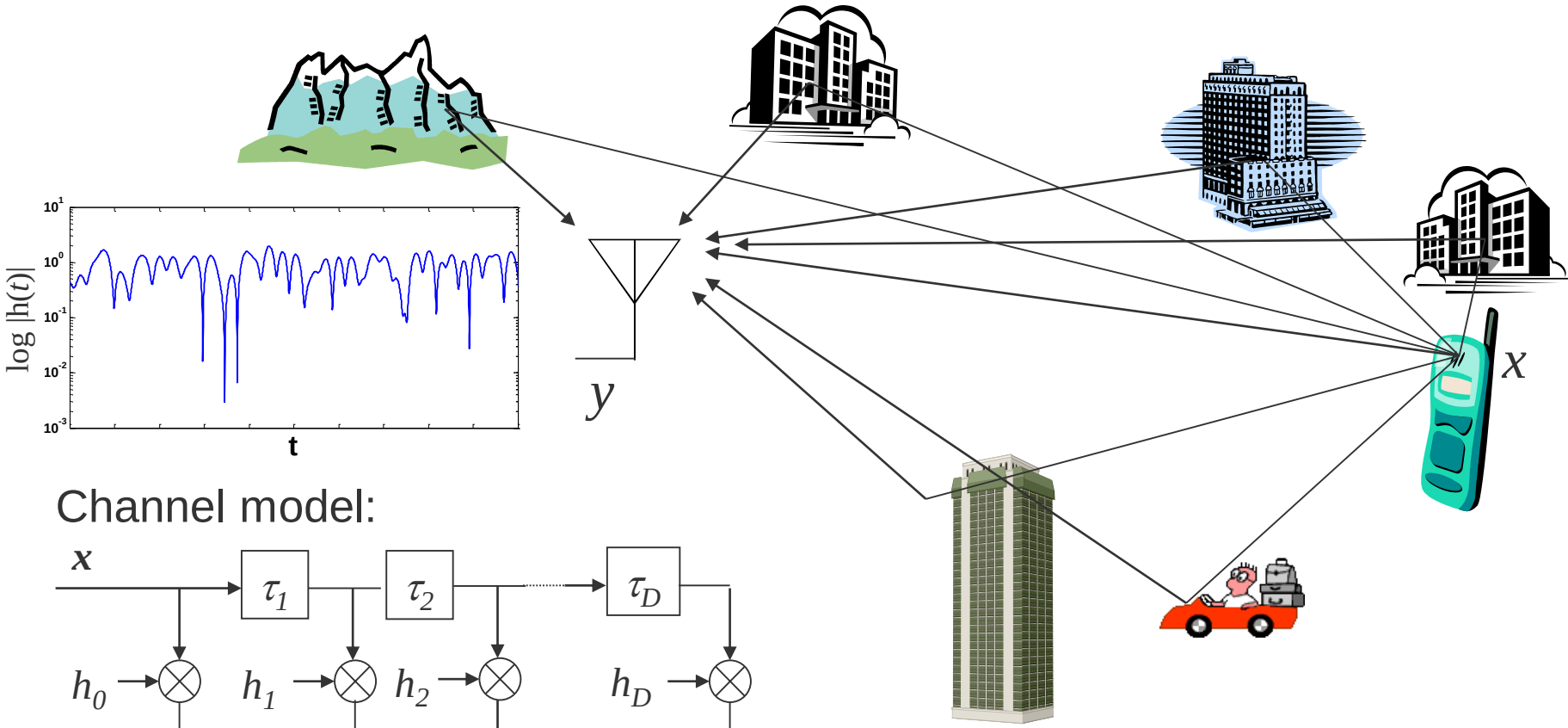
(4x10) x 2 = 80 elements



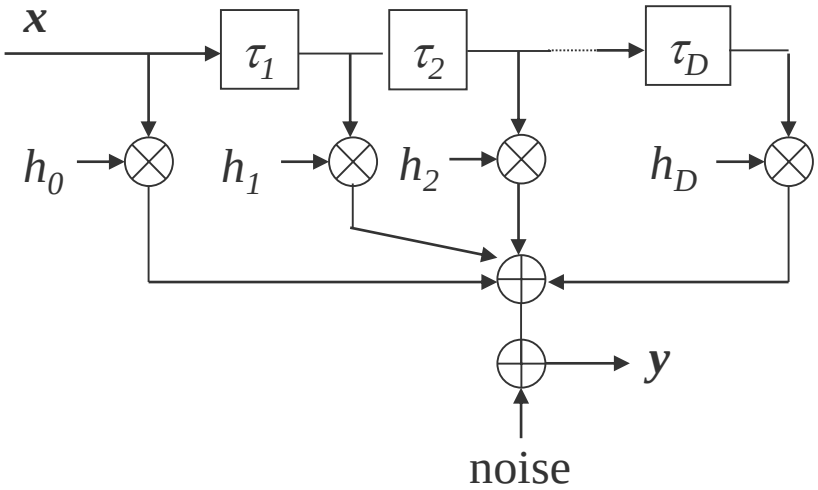
24 x 2 = 48 elements
(+10 at cube)

Fraunhofer
Heinrich Hertz Institute

Wireless Channel

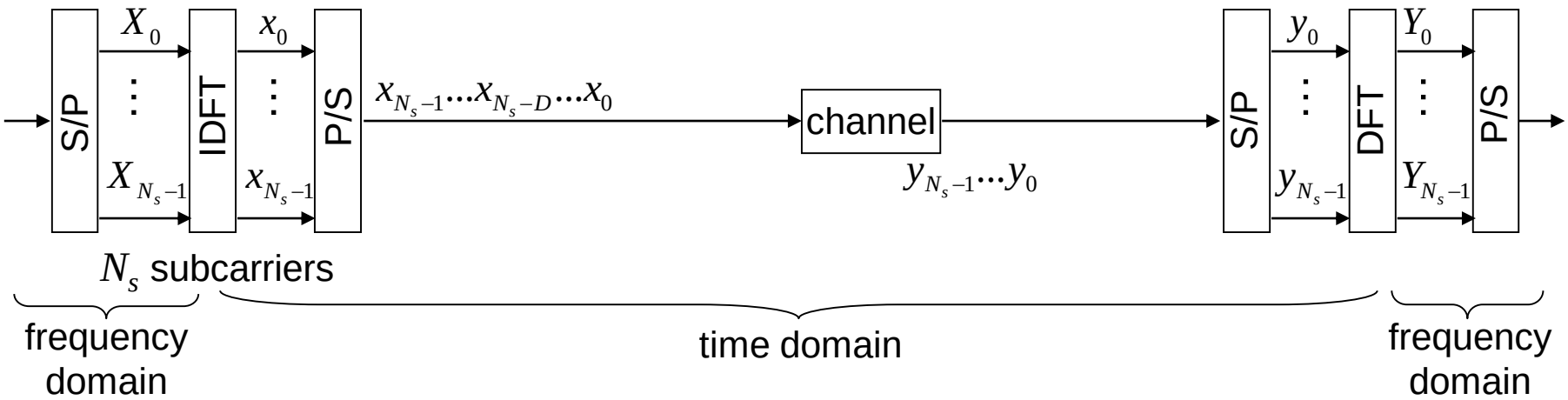


Channel model:

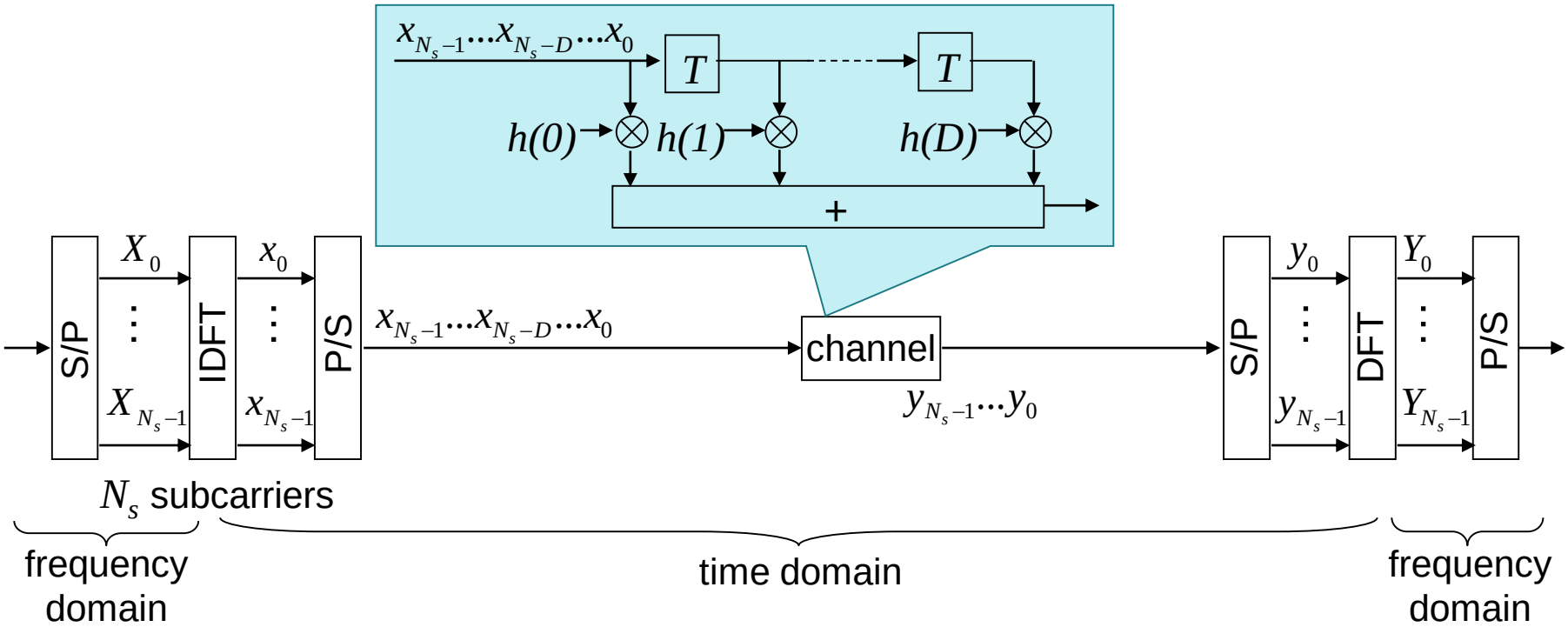


Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (1)

Idea:
- Overcome problem of frequency-selectivity by simultaneous transmission via orthogonal frequency-flat channels.
- Efficient implementation using Discrete Fourier Transform (DFT)

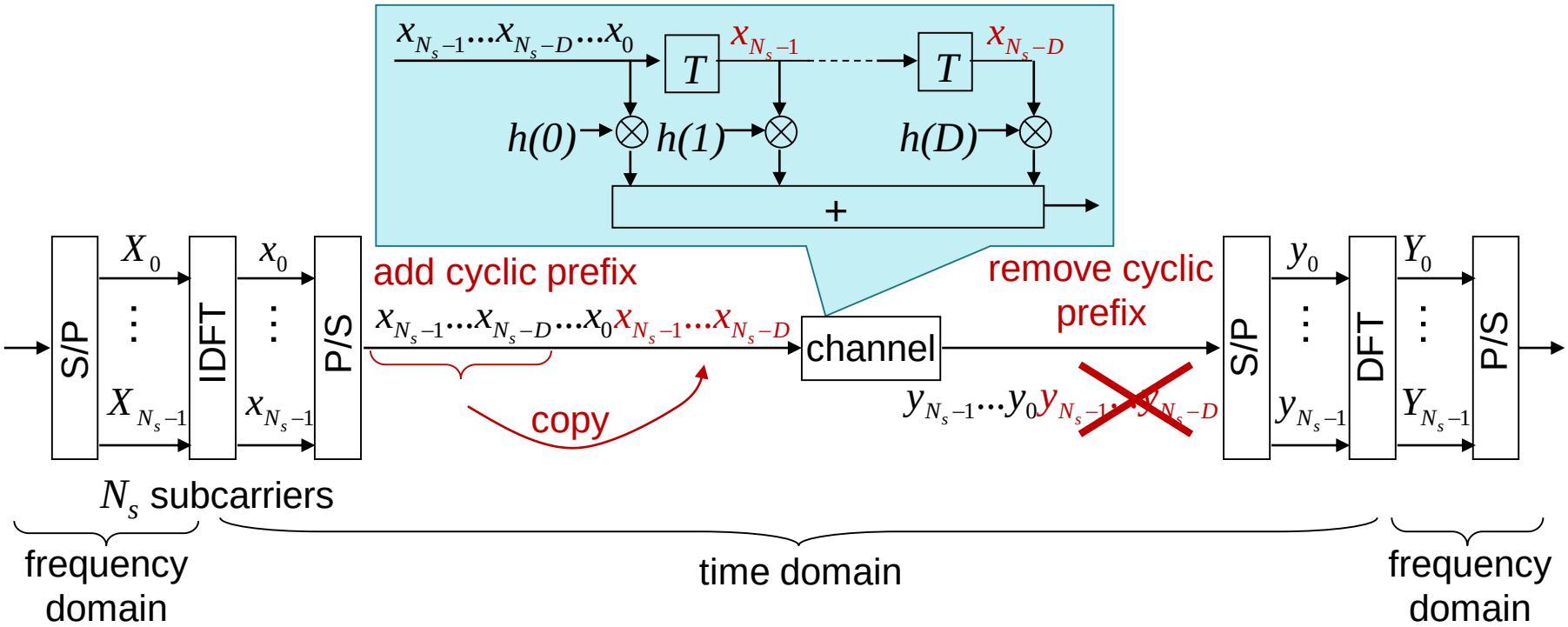


Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (2)



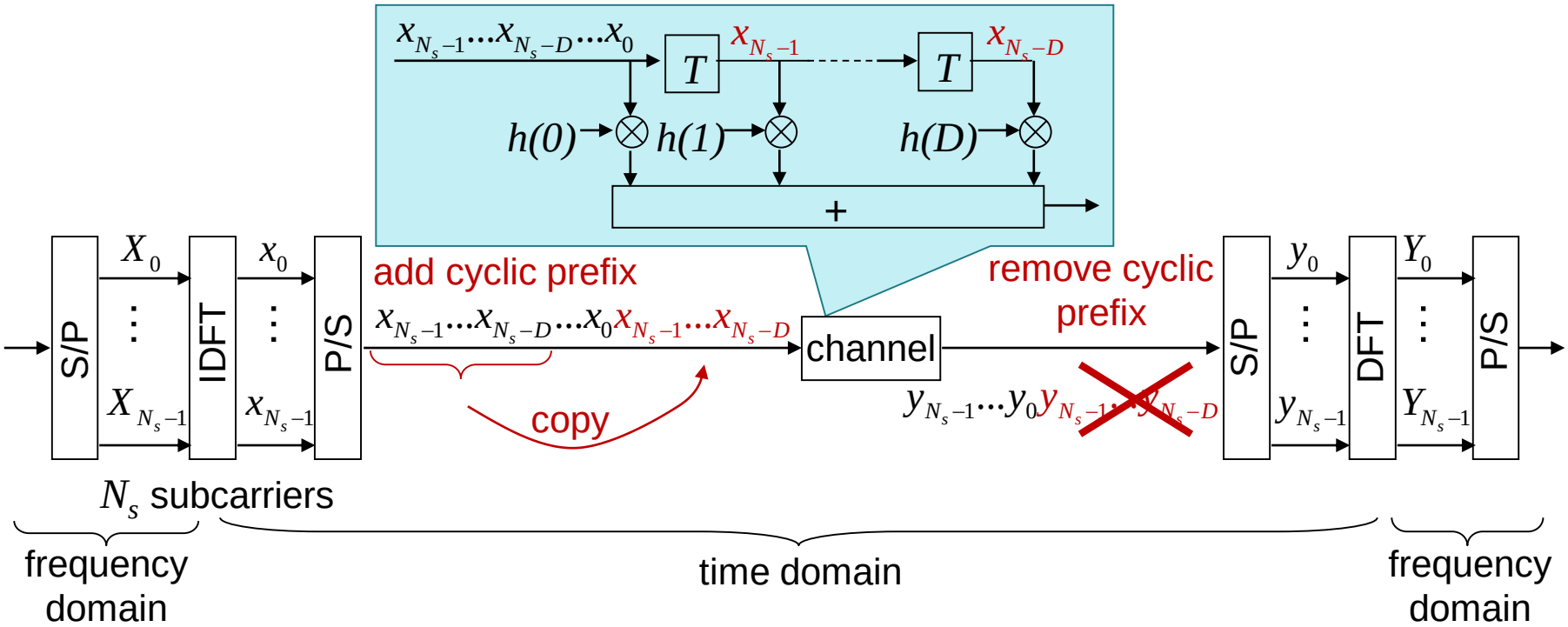
$$\begin{bmatrix} y_0 \\ \vdots \\ y_{N_s-1} \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & & & 0 \\ h(1) & h(0) & 0 & \cdots & & 0 \\ & \ddots & \ddots & & & \\ 0 & \cdots & 0 & h(D) & \cdots & h(0) \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{N_s-1} \end{bmatrix}$$

Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (3)



$$\begin{bmatrix} y_0 \\ \vdots \\ y_{N_s-1} \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & & 0 \\ h(1) & h(0) & 0 & \cdots & 0 \\ & \ddots & \ddots & & \\ 0 & \cdots & 0 & h(D) \cdots & h(0) \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{N_s-1} \end{bmatrix}$$

Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (4)

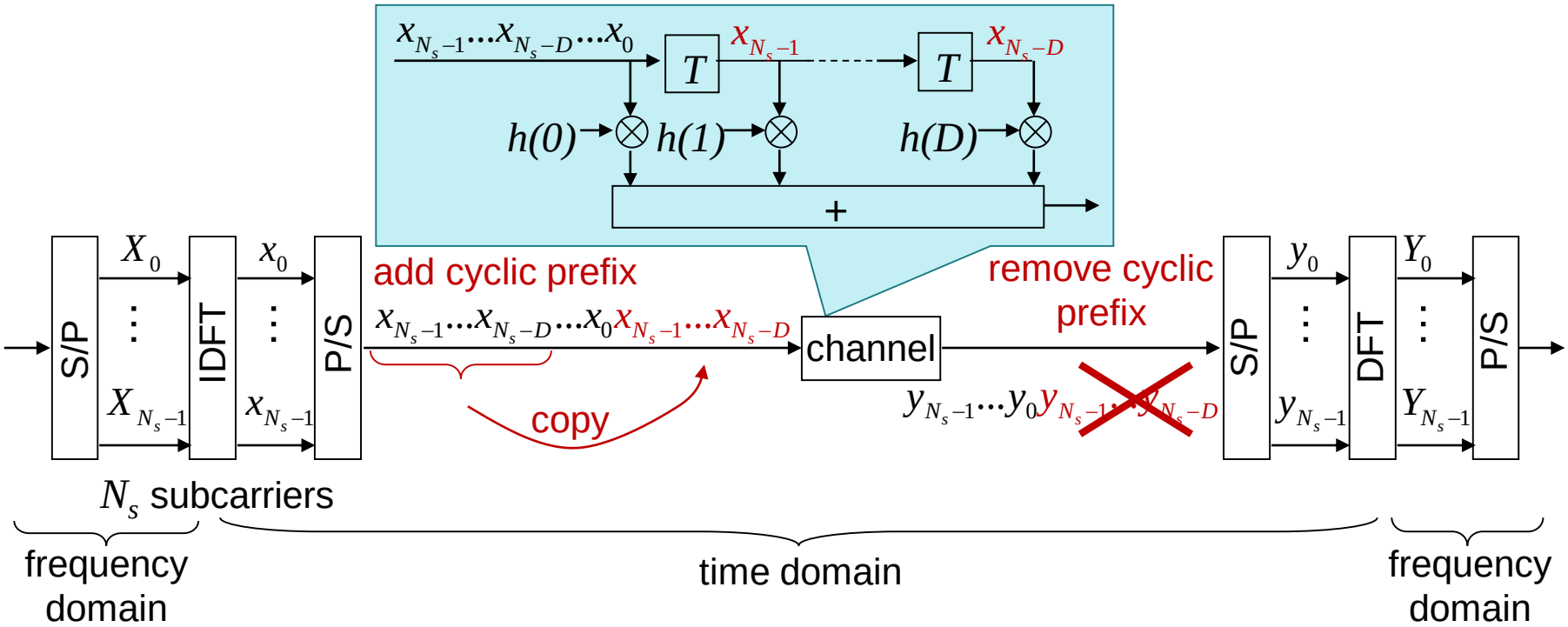


$$\underbrace{\begin{bmatrix} y_0 \\ \vdots \\ y_{N_s-1} \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & 0 & h(D) & \cdots & h(2) & h(1) \\ h(1) & h(0) & 0 & \cdots & 0 & h(D) & \cdots & h(2) \\ & \ddots & \ddots & & & & \ddots & \\ 0 & \cdots & & 0 & h(D) & & \cdots & h(0) \end{bmatrix}}_{\text{circular matrix}} \begin{bmatrix} x_0 \\ \vdots \\ x_{N_s-1} \end{bmatrix}$$

$\mathbf{H}^c \Rightarrow \text{EVD: } \mathbf{H}^c = \mathbf{F}^H \mathbf{\Lambda} \mathbf{F}$

$\mathbf{F}$: DFT matrix

Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (5)

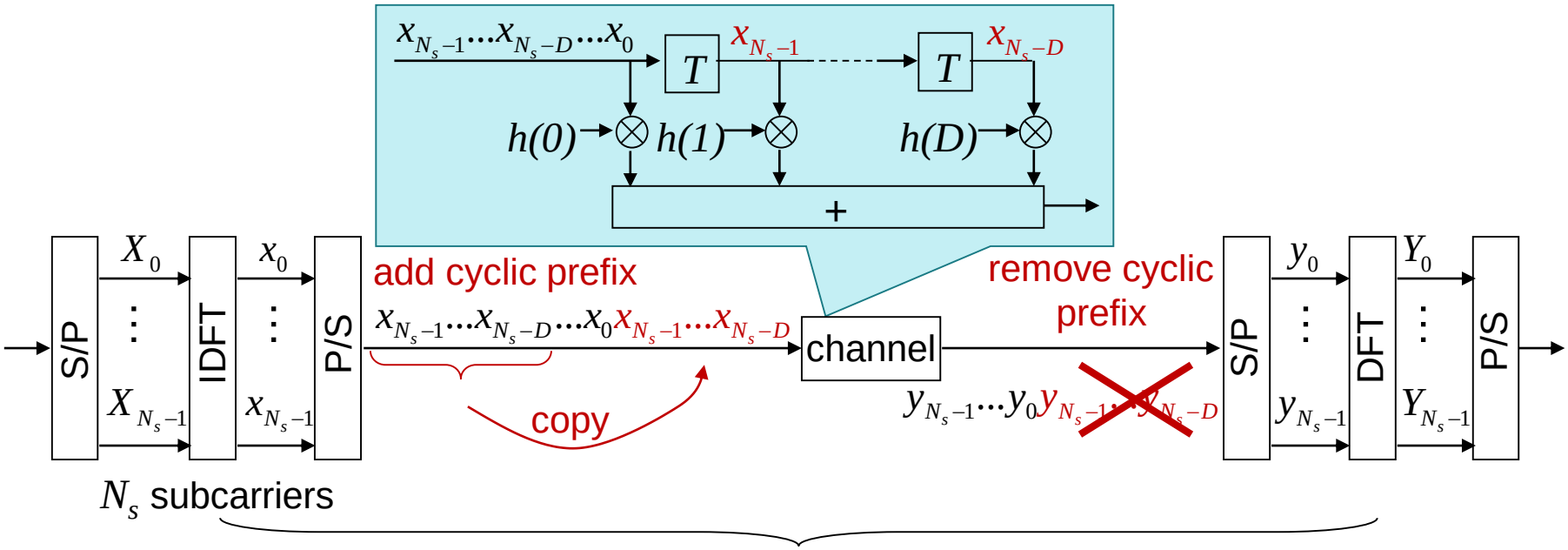


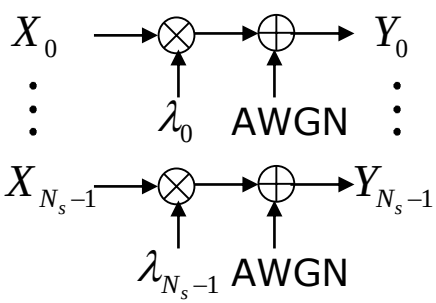
Time domain: $\mathbf{y} = \mathbf{H}^c \mathbf{x} = \mathbf{F}^H \mathbf{\Lambda} \mathbf{F} \mathbf{x}$

Frequency domain (OFDM): $\mathbf{Y} = \mathbf{F}\mathbf{H}^c\mathbf{F}^H\mathbf{X} = \underbrace{\mathbf{F}\mathbf{F}^H}_{\mathbf{I}} \mathbf{\Lambda} \underbrace{\mathbf{F}\mathbf{F}^H}_{\mathbf{I}} \mathbf{X} = \mathbf{\Lambda}\mathbf{X}$

$$\mathbf{F} = \frac{1}{\sqrt{N_s}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j\frac{2\pi}{N_s}\cdot 1\cdot 1} & e^{-j\frac{2\pi}{N_s}\cdot 1\cdot 2} & \cdots & e^{-j\frac{2\pi}{N_s}\cdot 1\cdot (N_s-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{-j\frac{2\pi}{N_s}\cdot (N_s-1)\cdot 1} & e^{-j\frac{2\pi}{N_s}\cdot (N_s-1)\cdot 2} & \cdots & e^{-j\frac{2\pi}{N_s}\cdot (N_s-1)(N_s-1)} \end{bmatrix}, \quad \mathbf{\Lambda} = \begin{bmatrix} \lambda_0 & & \mathbf{0} \\ & \lambda_1 & \\ & & \ddots \\ \mathbf{0} & & \lambda_{N_s-1} \end{bmatrix}, \quad \mathbf{I} = \begin{bmatrix} 1 & & \mathbf{0} \\ & 1 & \\ & & \ddots \\ \mathbf{0} & & 1 \end{bmatrix}$$

Capacity of Parallel AWGN Channels: Orthogonal Frequency Division Multiplex (OFDM) (6)

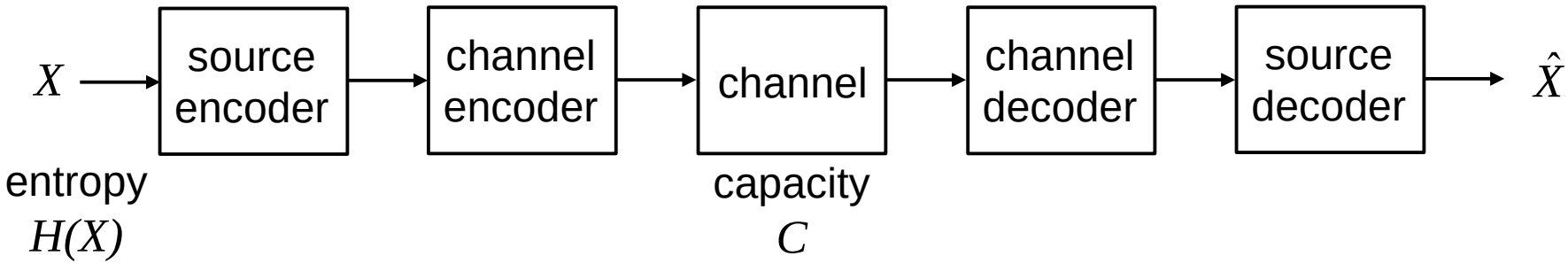




Decoupled flat channels in the frequency domain

→ Allocate transmit power to subchannels according to a waterfilling solution.

Source-Channel Coding Theorem



There exists a source-channel code with arbitrarily small error probability, if

$$H(X) < C.$$

Converse: There is no transmission scheme with arbitrarily small error probability, if $H(X) > C$.

Separate design of source coding and channel coding can achieve the optimum rate.

Multi-User Information Theory

Multiuser Information Theory –Multiple Access Channel (MAC) and Broadcast Channel (1)

So far, we have discussed the capacity of channels which are used by only a single user. However, in most communications systems multiple users wish to communicate. Therefore, the available resources, e.g. the available bandwidth, have to be shared. Multi-user information theory treats fundamental limits for achievable rates of multiple competing users as well as transmission strategies which are suitable for achieving those rates.

Two basic models of multi-user information theory are the multiple access channel (MAC) and the broadcast channel.

The multiple access channel (MAC) characterizes a many–to-one communication scenario, where multiple users want to transmit data to a single receiver. An example for a MAC is the uplink transmission in a cellular system, where the cellphones inside the coverage area of a cell transmit data to a base station.

The broadcast channel is the counterpart to the MAC and characterizes a one-to-many communication scenario, where a single transmitter transmits data towards several users. An example for the broadcast channel would be the downlink of a cellular system, where a base station transmits data towards several cellphones inside its coverage area.

Note that in information theory, the term *broadcast* is used in a broader sense than in general use: Broadcast is widely referred to as a transmission scenario, where several receivers detect the same information broadcasted by a single transmitter, e.g. a TV broadcast station. In contrast, the broadcast channel in information theory refers to a scenario, where a single transmitter sends independent data towards different users, where the independent information intended for different users is contained in a common transmit signal.

Multiuser Information Theory –Multiple Access Channel (MAC) and Broadcast Channel (2)

MAC and broadcast channel have some fundamental differences which are important for determination of the achievable rates:

- In the MAC, the signals transmitted from different users face different channels and are superimposed at the receiver. In the broadcast channel, the total transmit signal is received through the same channel at a particular receiver. This means that from the viewpoint of a particular user, the desired part of the transmit signal undergoes the same channel distortions as the interfering parts of the transmit signal which are intended for other users.
- In the MAC, there is only one additive noise source which is the additive noise at the single receiver. In contrast, in the broadcast channel there are independent additive noise sources at all receivers.
- The most important difference between MAC and broadcast channel is in the power constraint: In the MAC, each user has typically an individual power constraint, i.e. the transmit power of user k cannot exceed its maximum transmit power P_k. Putting more users in the MAC increases the maximum total received power, as more users transmit data towards the receiver. In the broadcast channel, there is a total transmit power constraint, i.e. the transmit power of the single transmitter cannot exceed a maximum transmit power P_{Tx}. I.e., if the number of users increases, the average power of the signal part per user will decrease.

Multiple Access

A communications system needs to serve multiple users within the available bandwidth. Different possibilities exist how to allocate resources efficiently to user signals. Typically, the available resources are organized in a time-frequency grid. I.e., the bandwidth is subdivided into subbands and the service time is subdivided into time slots. The resulting time-frequency blocks can be allocated to the users which compete for resources. Additional resources maybe available e.g. in space. This is e.g. the case if parallel cables are available or if multiple antennas are applied at the transmitters and/or receivers in a wireless system (MIMO).

A conventional approach for multiple access is frequency division multiple access (FDMA: A fraction of the available bandwith is exclusively allocated to a particular user during the whole service time. Different users transmit on different frequency subbands, i.e. the users are separated in frequency.

Another approach is time division multiple access (TDMA). Here, each user transmits only during a fraction of the service time (in a time slot) using the complete system bandwidth. Hence, users are separated in time.

However, it is also possible to allow multiple users to transmit simultaneously using the complete system bandwidth, i.e. multiple user signals are superimposed. Combined with coding, this approach is called superposition coding. A special case of superimposed user signals is code division multiple access (CDMA), where the user signals are multiplied (spread) by a user-specific signature code.

Often, several of these basic multiple access schemes are combined. E.g. in a TDMA/FDMA system, a user is allocated certain time slots during which he can use certain subbands. The basic multiple access schemes can be applied both in multiple access channels and in broadcast channels.

Achievable Rate Regions of Multi-User Channels

The fundamental limit for the achievable rate of a single user channel is given by a single number, the channel capacity. In contrast, the achievable transmission rates in a multiuser channel cannot be adequately characterized by a single rate per user: Consider a multiple access channel where multiple users are competing for the resources. If the users face different channel SNRs, all available resources could be allocated to the user with the better SNR as this user can benefit most from the resources. This allocation would maximize the so-called sum capacity which is the sum of the capacities of the individual users for a given resource allocation. However, obviously this allocation would be highly unfair: Only a single user is served while the other users in the network cannot transmit any data.

In order to provide some fairness, the resource allocation could be done such that each user gets a share of the available resources. This may not maximize the sum capacity but include a certain degree of fairness.

We denote the union of all user rate combinations, for which error-free transmission is theoretically possible, as the *achievable rate region*. The rate combinations of particular interest are the pareto-optimum points on the boundary of the rate region. A rate combination is called *pareto-optimum*, if the achievable rate of any user cannot be increased without decreasing the achievable rate of at least one other user.

The achievable rates depend on the type of the multiple access scheme and on the power constraints. The ultimate rate region which is achievable for a given power constraint without prior restriction of the multiple access scheme is called the *capacity region*.

Achievable Rate Region of Multiple Access using TDMA (1)

The achievable rate region of multiple access using TDMA depends on the power constraint. The most relevant power constraint in practice is the per user power constraint, i.e. the power of user k is limited by $P_{Tx} \leq P_k$.

An alternative – in practice less relevant – constraint assumes $E\{P_{Tx}(t)\} \leq P_k$, which is essentially an energy constraint. The idea is that a user terminal is battery limited. Hence, if a user k transmits only during a fraction τ_k, $0 \leq \tau_k \leq 1$, of the total time, it may use $1/\tau_k$ times the instantaneous power compared to a continuous transmission.

We will refer to the first power constraint, i.e. $P_{Tx} \leq P_k$, as *power constraint* and to the second, i.e. $E\{P_{Tx}(t)\} \leq P_k$, as *energy constraint*.

The corner points of the achievable rate region are obtained, when only a single user k is served during the total time. In this case, the maximum achievable rate is given by the single user capacity C_k of user k, i.e. by

$$R_k \leq C_k = B \log_2 \left(1 + \frac{|h_k|^2 P_k}{P_N} \right),$$

where h_k is the channel coefficient for user k and P_N is the noise power at the receiver. Obviously, if the rate C_k is achievable for user k, any rate $R_k \leq C_k$ is also achievable.

If user k transmits only during a fraction τ_k, $0 \leq \tau_k \leq 1$, of the total time in TDMA with power constraint $P_{Tx} \leq P_k$, the achievable rate for user k is upper bounded by

$$R_k \leq \tau_k C_k = \tau_k B \log_2 \left(1 + \frac{|h_k|^2 P_k}{P_N} \right). \tag{1}$$

Achievable Rate Region of Multiple Access using TDMA (2)

The achievable rate region for K users is then given by the union of all rate combinations $\{R_1,...,R_K\}$ according to (1) which can be obtained with different time sharing satisfying

$$\sum_{k=1}^{K} \tau_k = 1.$$

In the special case of $K=2$ users, the boundary of the achievable rate region can be illustrated in a two-dimensional plot as the connecting line between the single user capacities C_2 and C_1:

All rate combinations $\{R_1,R_2\}$ on the connecting line between the single user capacities C_2 and C_1 are achievable via time sharing.

In case of TDMA with energy constraint, (1) has to be replaced by

$$R_k \leq \tau_k C_k = \tau_k B \log_2 \left(1 + \frac{|h_k|^2 \frac{P_k}{\tau_k}}{P_N} \right). \qquad (2)$$

As $\tau_k \leq 1$ and, hence, the per user power is increased in (2) compared to (1), the energy constraint will yield a larger achievable rate region compared to the power constraint. The boundary of the achievable rate region is not anymore determined by straight lines.

Achievable Rate Region of Multiple Access using FDMA (1)

A fundamental difference between TDMA and FDMA is that the noise power faced by the k-th user's signal depends on the bandwidth which is allocated to user k. If user k is allocated a fraction β_k, $0 \leq \beta_k \leq 1$, of the total bandwidth B, the noise power faced by the signal of user k is given by

$$P_{N,k} = \frac{N_0}{2} 2\beta_k B = N_0 \beta_k B.$$

Note, that the frequency band allocation satisfies

$$\sum_{k=1}^{K} \beta_k = 1.$$

Similarly as for the TDMA case, we can distinguish between two types of power constraints:

If the user is able to transmit at a given maximum power P_k independent of the allocated bandwidth $\beta_k B$, the achievable rate for user k is given by

$$R_k \leq C_k = \beta B \log_2 \left(1 + \frac{|h_k|^2 P_k}{N_0 \beta B} \right),$$

which is basically the same expression as in case of TDMA with energy constraint.

Achievable Rate Region of Multiple Access using FDMA (2)

However, in practice often the power spectral density $S_{X_k X_k}(f)$ of a transmit signal needs to satisfy a spectrum mask. In such a case, there is a constraint on the power spectral density rather than on the power P_k of a user's signal. Assume that the power spectral density of user k must not exceed the constant value $P_k/(2B)$, i.e.

$$S_{X_k X_k}(f) \leq \frac{P_k}{2B}.$$

Then, if user k is allocated only a bandwidth $\beta_k B$, its transmit power is constrained by

$$P_{Tx,k} \leq \int_{-\infty}^{\infty} S_{X_k X_k}(f)\,df = \frac{P_k}{2B} \cdot 2\beta_k B = \beta_k P_k.$$

The achievable rate for user k is then given by

$$R_k \leq C_k = \beta_k B \log_2\left(1 + \frac{|h_k|^2 \beta_k P_k}{N_0 \beta_k B}\right) = \beta_k B \log_2\left(1 + \frac{|h_k|^2 P_k}{N_0 B}\right),$$

which is essentially the same expression as for TDMA with power constraint.

Achievable Rate Region of the Multiple Access Channel (MAC) (1)

In order to determine the ultimate achievable rate region of the multiple access channel (MAC), also called the *capacity region*, we have to give up restrictions on the multiple access scheme. I.e., we do not assume a certain multiple access scheme such as TDMA or FDMA anymore but now wish to determine the achievable rate region with the best possible multiple access scheme. In particular, unlike TDMA and FDMA, we do not require the multiple access scheme to be orthogonal anymore.

It is easy to understand that the MAC achievable rate region is outer bounded by two kinds of bounds:

- A particular user k cannot achieve a rate higher than its single user capacity C_k, i.e.

$$R_k \leq C_k = B \log_2 \left(1 + \frac{|h_k|^2 P_k}{P_N} \right). \tag{1}$$

- A subset $k \subset \{1,...,K\}$ of users cannot achieve a sum rate higher than the rate of a virtual super user with received power equal to the total received power from the users in the subset. The idea behind this bound is that a virtual super user could apply an optimized joint codebook for all transmitters in the subset. In contrast, the individual users in the subset can only access their own transmitter and, consequently, apply only a local codebook for their own transmit signal. Coding over all transmitters cannot be worse than local coding at the individual transmitters as local coding can be viewed as a special case of joint coding.

Achievable Rate Region of the Multiple Access Channel (MAC) (2)

The respective bound is given by

$$\sum_{k\subset\{1,\ldots,K\}} R_k \leq B\log_2\left(1+\frac{\sum_{k\subset\{1,\ldots,K\}} |h_k|^2 P_k}{P_N}\right). \qquad (2)$$

The MAC capacity region is bounded by the expressions (1) and (2), where (1) has to be evaluated for all users $k\in\{1,\ldots,K\}$ and (2) has to be evaluated for all possible subsets $k\subset\{1,\ldots,K\}$ of users.

It is easy to understand that the MAC capacity region cannot be larger than the region bounded as explained above. It is more difficult to proof that this region is also achievable. However, it has indeed been shown that the region bounded by expressions (1) and (2) as explained above is achievable and, hence, is the MAC capacity region.

Achievable Rate Region of the Multiple Access Channel (MAC) (3)

It is instructive to investigate the MAC capacity region for the special case of $K=2$ users, as then the capacity region can be illustrated in a two-dimensional plot with abscissa R_1 and ordinate R_2, where R_k denotes the achievable rate of user k.

The bounds (1) are represented as a vertical and a horizontal line at the single user capacities C_1 and C_2, respectively. The bound (2) is a diagonal line with the numerically same crossing point at abscissa and ordinate. The capacity region is the region below all those lines.

It is interesting to look at the corner points of the capacity region: A user k can transmit at its single user capacity C_k while the other user l can simultaneously transmit at a relatively low rate R_l^*. The interference from user l does not cause a capacity loss for user k compared to its single user capacity C_k. This corner point in the capacity region is achievable by a so-called *successive interference cancellation* detection strategy: It appears that the achievable rate of user l at the corner point is given by

$$R_l^* = B\log_2\left(1+\frac{|h_l|^2 P_l}{|h_k|^2 P_k + P_N}\right).$$

This implies that user l detects its signal while considering the signal of user k as additional additive noise.

Achievable Rate Region of the Multiple Access Channel (MAC) (4)

Recall that in an AWGN channel, the capacity achieving transmit symbol distribution is a normal distribution with zero mean and variance equal to the transmit power. Hence, when detecting the signal of user l, the interfering signal of user k can be regarded as additive white Gaussian noise with variance equal to the received power $|h_k|^2 P_k$. As the transmit signal of user k and the additive Gaussian channel noise N are statistically independent, the effective aditive noise power faced while detecting the signal of user l is given by $|h_k|^2 P_k + P_N$. The achievable rate R_1^* of user l is then the single user capacity of an AWGN channel with effective noise power $|h_k|^2 P_k + P_N$, i.e.

$$R_l^* = B \log_2 \left(1 + \frac{|h_l|^2 P_l}{|h_k|^2 P_k + P_N} \right).$$

Once the signal of user l has been detected, its contribution can be subtracted from the total received signal. This process is called *interference cancellation*. After cancellation of user l‘s signal, the receiver sees an effective interference-free AWGN channel for user k. Hence, user k can transmit at its single user capacity C_k.

The other corner point can be achieved in a similar way with exchanged roles of users k and l.

All points on the diagonal connecting line between the corner points of the MAC capacity region can be achieved by time sharing between the successive interference cancellation strategies at the corner points:

Achievable Rate Region of the Multiple Access Channel (MAC) (5)

This means that for a fraction τ, $0 \leq \tau \leq 1$, of the transmission time, user k transmits at its single user capacity C_k while user l transmits at the low rate R_l^*. For the remaining time, user l transmits at its single user capacity C_l while user k transmits at the lower rate R_k^*. By varying the fraction τ, any point on the connecting line between the corner points can be achieved.

We have determined upper bounds on the achievable user rates and have then identified transmission and detection strategies which can achieve the points on the bounds. If an upper bound is achievable, it must be not only a bound but the outer limit of the capacity region. Hence, we have shown that indeed the bounds are the limits of the capacity region. Any rate combination below the bounds can of course be also achieved, since it is always possible to let a user transmit at a lower rate than its achievable rate.

The pareto optimum points are located on the connecting line between the corner points of the capacity region: For all points on this line, it is not possible to increase the achievable rate of user k without reducing the achievable rate of user l.

Moreover, all points on this connecting line have the same sum capacity as the diagonal has slope -1. This is even true in the case that the users face different channel SNRs. Consequently, the maximum sum capacity of the MAC can be achieved with different rate allocations to the users. This is important when the individual user rates have to be adapted to the requirements of a service or to achieve fairness among users in the rate allocation.

Achievable Rate Region of the Multiple Access Channel (MAC) (6)

In contrast, TDMA with energy constraint and FDMA with power constraint achieve the maximum sum capacity in only a single point on the boundary of the achievable rate region. Therefore, these schemes do not allow for flexible rate allocation if the system shall be operated at the maximum sum capacity.

In case of TDMA with power constraint and FDMA with power spectral density constraint, the maximum sum capacity is achieved if only the user with the best channel SNR is served at all, i.e. if the whole bandwidth is allocated for the whole time to a single user. This is obviously a highly unfair solution. Flexible resource allocation at the maximum sum rate can only be achieved, if all users face the same channel coefficient magnitude.

The discussion on the MAC channel above can be generalized to the K user case. In general, the corner points appear in a multi-dimensional space and can be achieved via successive interference cancellation. The pareto optimum points on the connecting hyperplanes can be achieved via time sharing.

The MAC capacity region is in general larger than the achievable rate regions of orthogonal multiple access schemes such as TDMA or FDMA. The boundary of the MAC capacity region can be achieved using so-called superposition coding.

Achievable Rate Region of the Broadcast Channel (1)

The capacity region of the broadcast channel, where a single transmitter transmits towards multiple users can be found in different ways.

An elegant method has been indentified by Jindal, Vishwanath and Goldsmith in 2004: They showed, that the capacity region of a broadcast channel can be determined via the capacity region of its dual MAC.

The dual MAC to a broadcast channel is a MAC channel with the following properties:

- The channel coefficients h_k are the same as in the broadcast channel.
- The noise statistics at the single receiver of the dual MAC are the same as the noise statistics at the multiple receivers in the broadcast channel.
- The dual MAC has a sum power constraint

$$\sum_{k=1}^{K} P_k \leq P_{BC}, \tag{1}$$

  i.e. the maximum sum power of the active transmitters in the dual MAC is equal to the maximum transmit power P_{BC} in the broadcast channel.

The sum power constraint is the essential difference to the determination of the MAC capacity region. It takes care of the fact that in the broadcast channel, the limited transmit power has to be shared between multiple users.

Achievable Rate Region of the Broadcast Channel (2)

In order to determine the capacity region of the broadcast channel, we have to obtain the capacity regions of the dual MACs for all possible transmit power distributions P_k which satisfy the sum power constraint (1). The capacity region of the braodcast channel is then given by the union of those dual MAC capacity regions, i.e. by

$$C_{BC} = \bigcup_{\left\{P_1,\ldots P_K \,\middle|\, \sum_{k=1}^{K} P_k = P_{BC}\right\}} C_{MAC}.$$

The capacity region of the broadcast channel looks quite different from the capacity region of the MAC channel. In case of $K=2$ users, unlike the boundary of the MAC capacity region, the capacity region of the broadvcast channel is in general not bounded by straight lines but by a round shape. The capacity region of the broadcast channel is in general larger than the capacity region of orthogonal multiple access schemes such as TDMA or FDMA. This is a motivation for the consideration of non-orthogonal multiple access schemes for future communications systems, particularly for wireless systems.

However, in the special case that all users face channel coefficients with the same magnitude, i.e. $|h_k| = h \;\forall\; k=1,\ldots,K$, it appears that the capacity region of the broadcast channel does not exceed the capacity region of TDMA but is identical to it.

Achievable Rate Region of the Broadcast Channel (3)

This can be easily shown by considering the constraint (2) from the MAC capacity region, i.e.

$$\sum_{k\subset\{1,\ldots,K\}} R_k \leq B\log_2\left(1+\frac{\sum_{k\subset\{1,\ldots,K\}} |h_k|^2 P_k}{P_N}\right).$$

In case of equal channel coefficient magnitudes $|h_k| = h$, the constraint can be written as

$$\sum_{k\subset\{1,\ldots,K\}} R_k \leq B\log_2\left(1+\frac{|h|^2 \sum_{k\subset\{1,\ldots,K\}} P_k}{P_N}\right).$$

In the region, where this sum rate constraint limits the MAC achievable rate region, this is equal to

$$\sum_{k\subset\{1,\ldots,K\}} R_k \leq B\log_2\left(1+\frac{|h|^2 \sum_{k\subset\{1,\ldots,K\}} P_k}{P_N}\right) = B\log_2\left(1+\frac{|h|^2 P_{BC}}{P_N}\right),$$

which is the same for all power distributions which satisfy the sum power constraint (1).

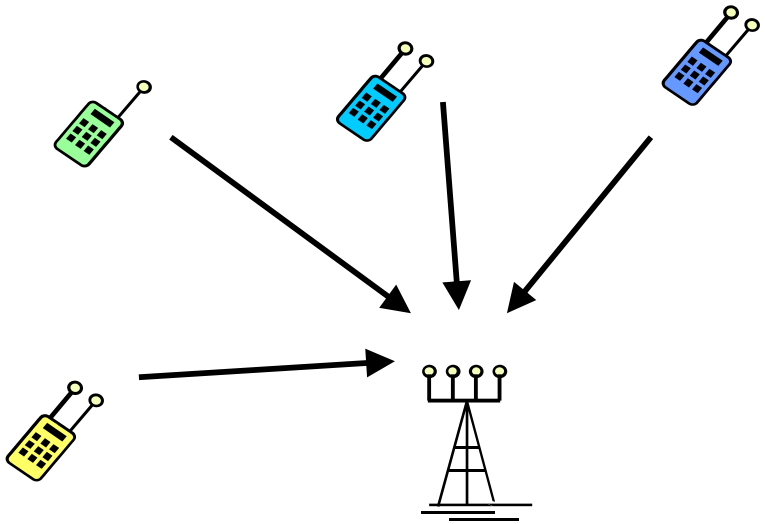
Achievable Rate Region of the Broadcast Channel (4)

As a result, the capacity region of the broadcast channel is limited by the connecting lines or hyperplanes, respectively, of the single user capacities with power constraints $P_k = P_{BC}$ (see Figures on slide).

As a result, in case of equal channel coefficient magnitudes of the multiple users, non-orthogonal multiple access schemes provide no benefit in terms of achievable rate region compared to e.g. TDMA.

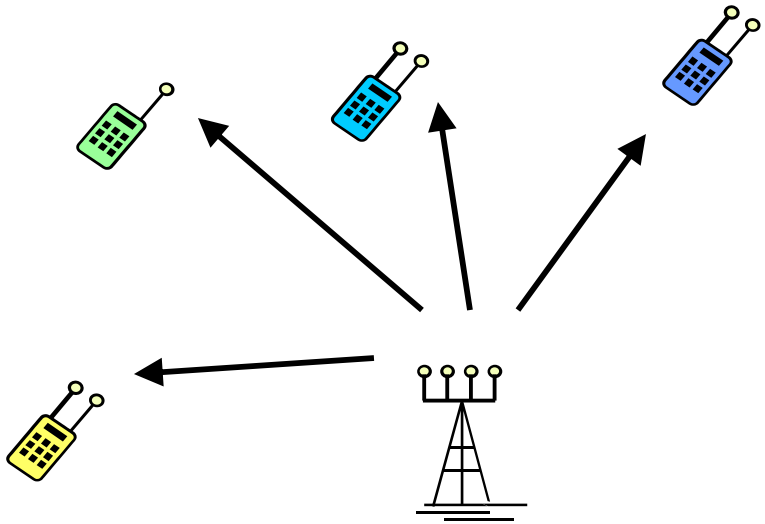
Multiple Access Channel (MAC) and Broadcast Channel

Multiple Access Channel



- Independent information is transmitted from multiple users to a common receiver.
- Different users apply independent transmit signal alphabets.
- Transmit power constraint per user.
- Each user faces a different channel.
- One additive noise term at the (single) receiver

Broadcast Channel



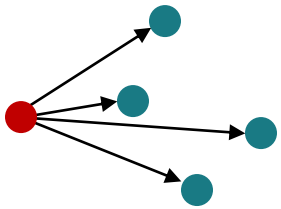
- Independent information is transmitted from a transmitter to multiple users.
- The information for all users is contained in a common signal.
- Total transmit power constraint.
- Signal and interference are distorted by the same channel.
- Additive noise term for each receiver.

Broadcast

In computer networking:

Transmitting packets that will be received by every device on the network. The packets are not requested by the devices but are part of a continuous data stream which is sent through the network.

→ Common information is sent to multiple devices.

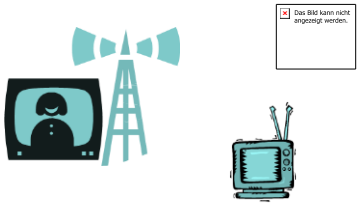


In common knowledge:

Distribute information e.g. via television or radio.

(Oxford Advanced Learner's Dictionary: *Send out in all directions especially by radio or TV*)

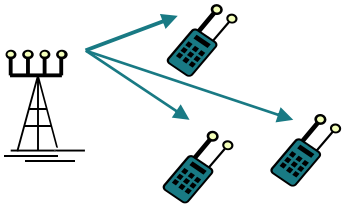
→ Common information is sent to multiple devices.



As service in wireless communications:

Cell Broadcast in GSM, UMTS and LTE is a one-to-many service for simultaneous delivery of messages to multiple users in a specified area.

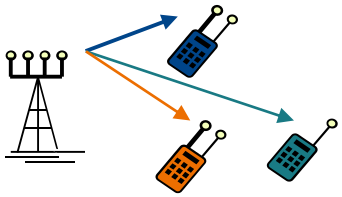
→ Common information is sent to multiple devices.



In information theory:

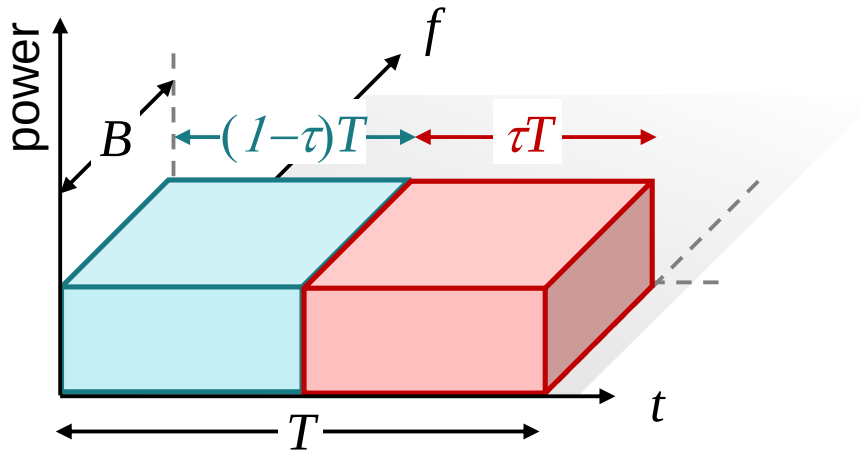
A broadcast channel is defined by a one-to-many scenario, where independent information is transmitted to multiple users via a common signal.

→ Different information is sent to multiple devices.
Transmission of common information is a special case of a broadcast channel.

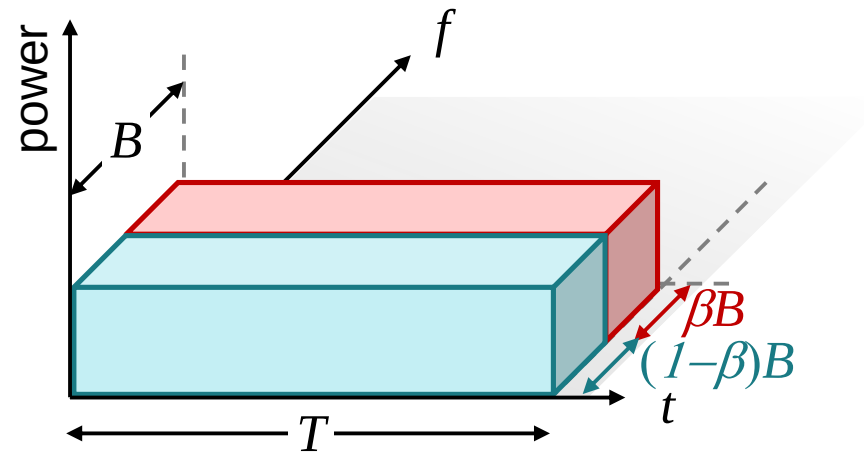


Multiple Access

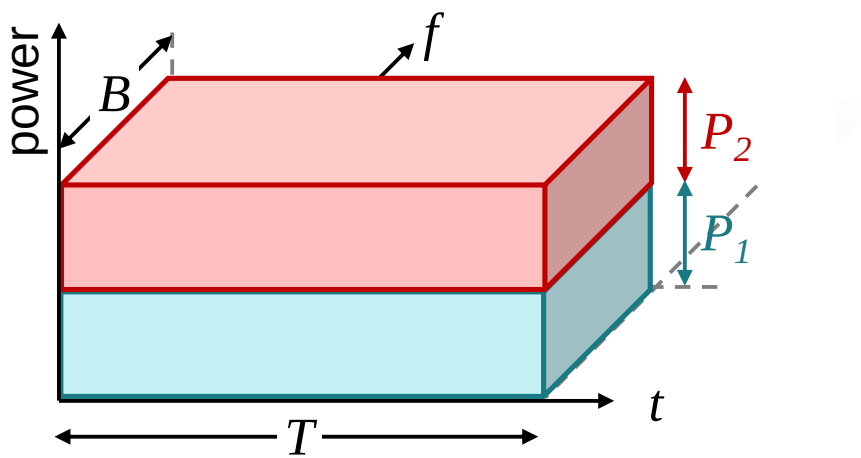
Time Division Multiple Access (TDMA)



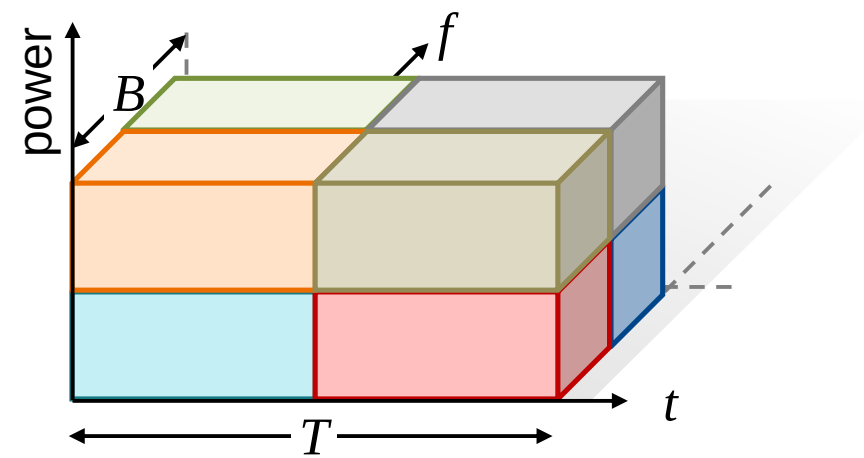
Frequency Division Multiple Access (FDMA)



Non-Orthogonal Multiple Access

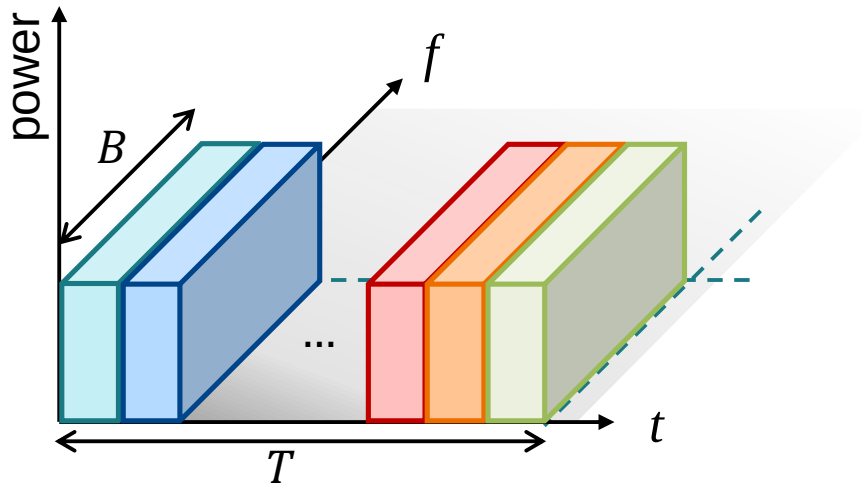


Hybrid Multiple Access

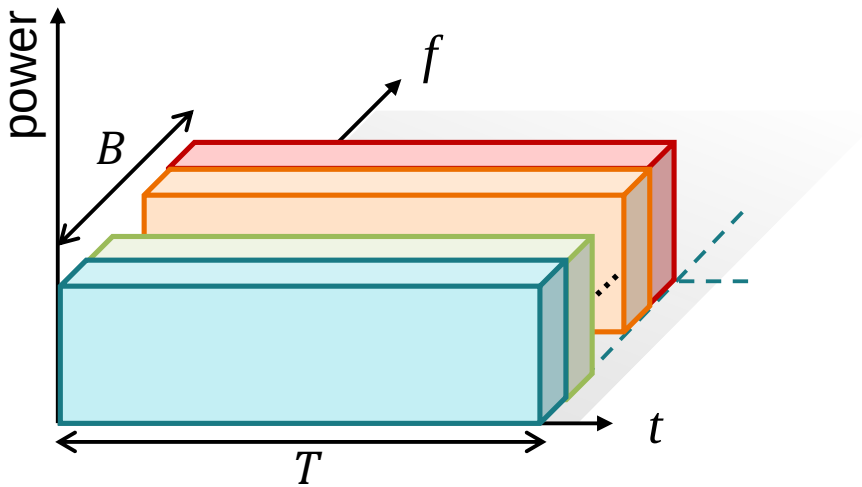


Multiple Access

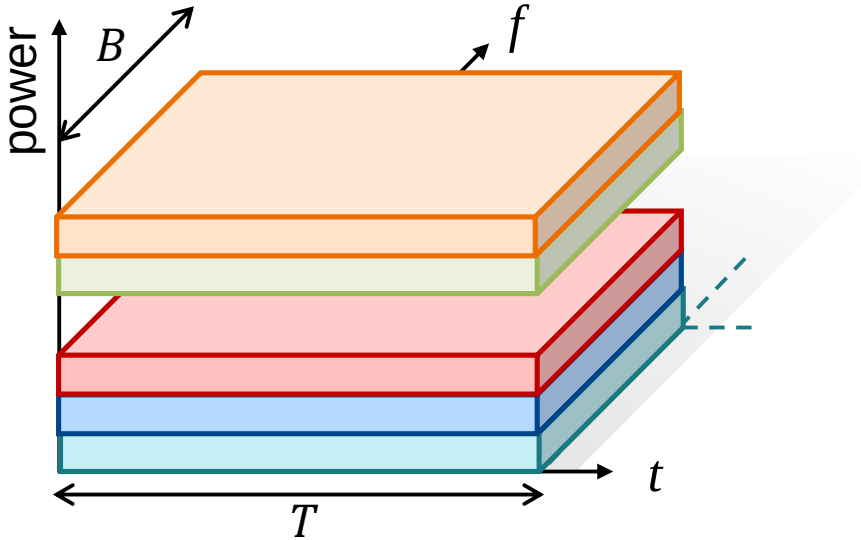
Time Division Multiple Access (TDMA)



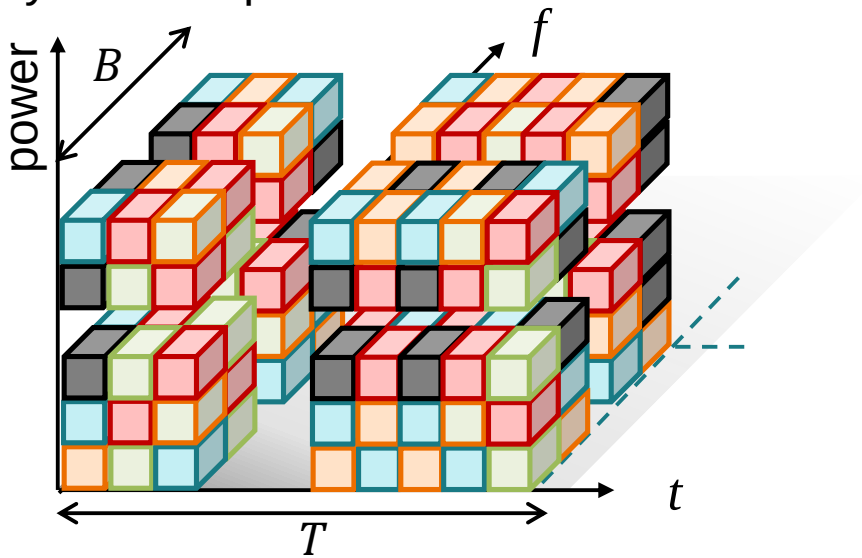
Frequency Division Multiple Access (FDMA)



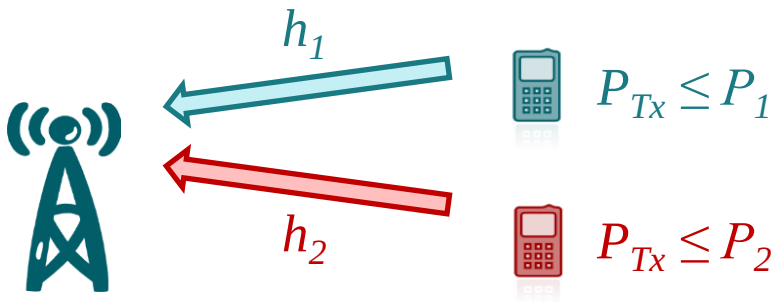
Non-Orthogonal Multiple Access

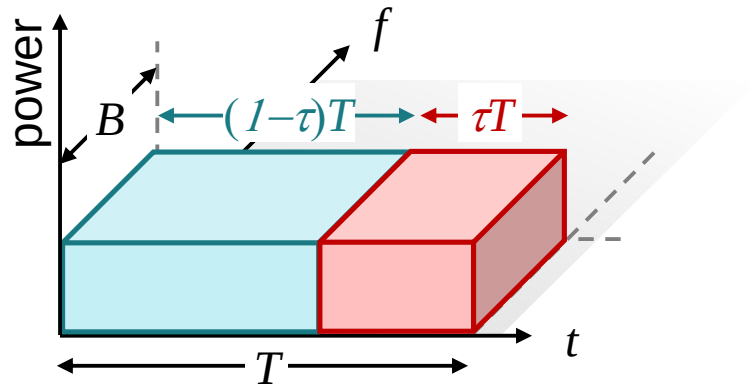


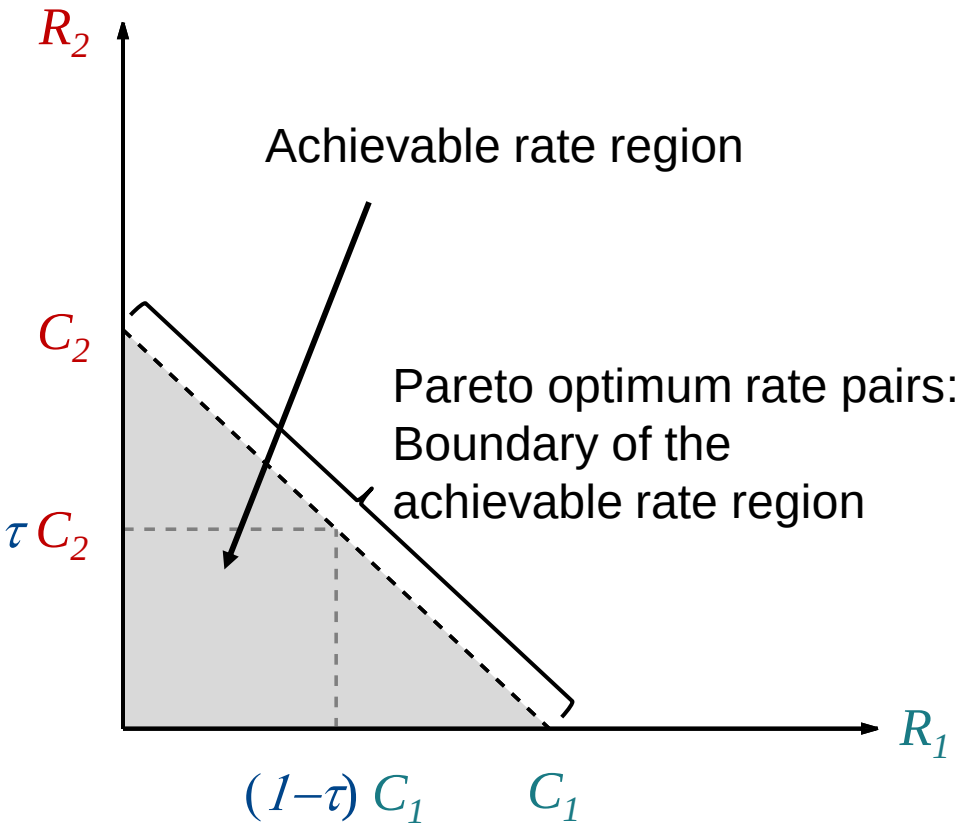
Hybrid Multiple Access



Capacity Region of Time Division Multiple Access (TDMA) with Power Constraint





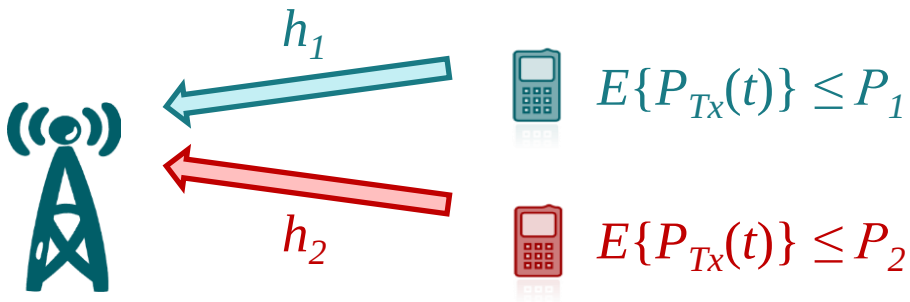


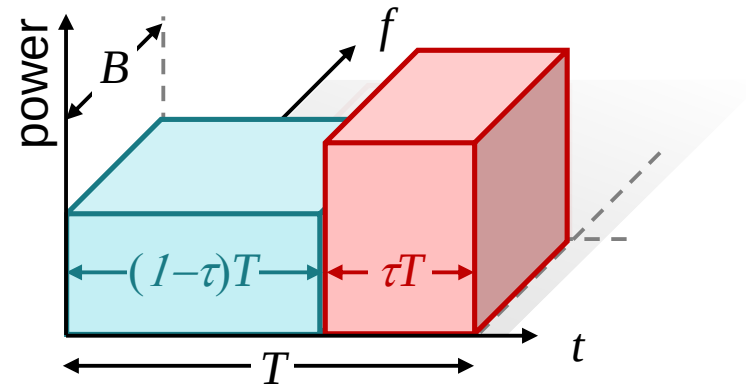
$$R_1 \le C_1 = B \log_2\left(1 + \frac{|h_1|^2 P_1}{P_N}\right)$$

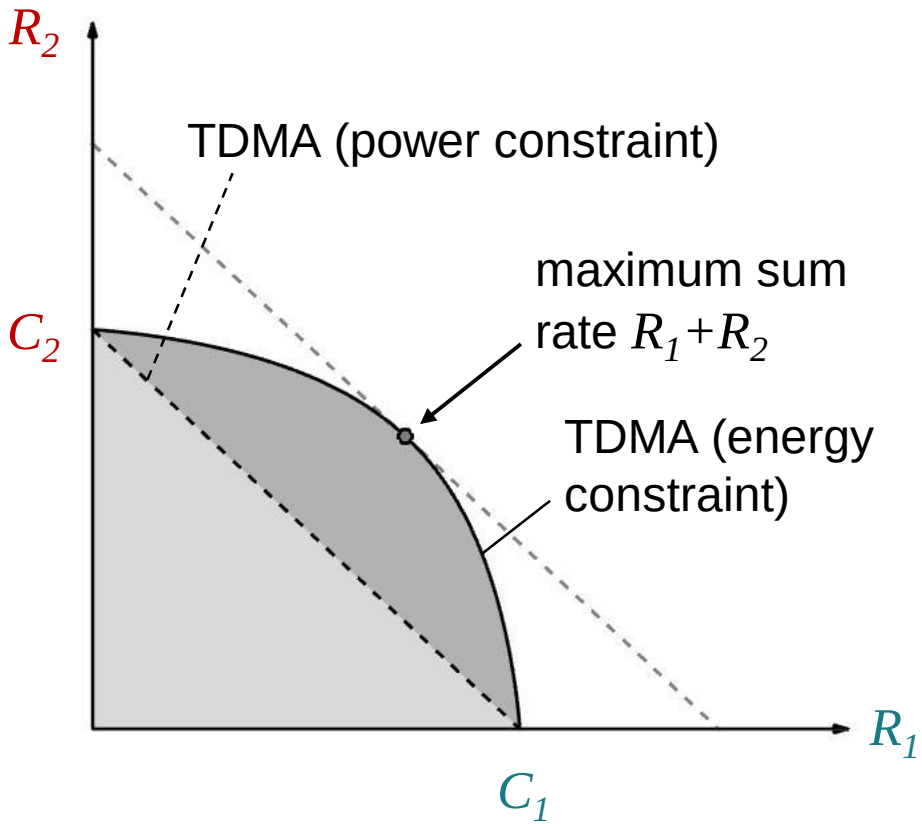
$$R_2 \le C_2 = B \log_2\left(1 + \frac{|h_2|^2 P_2}{P_N}\right)$$

$$R_1 \le (1-\tau) C_1, \quad R_2 \le \tau C_2$$

Capacity Region of Time Division Multiple Access (TDMA) with Energy Constraint







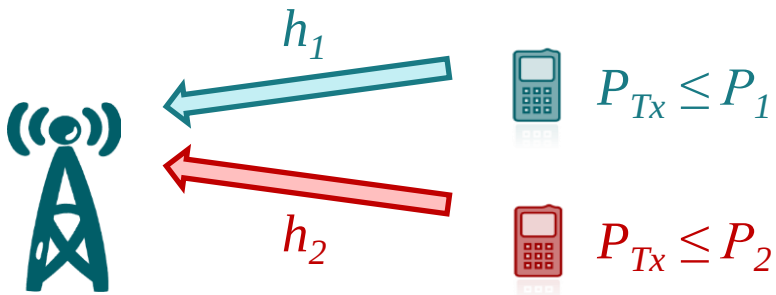
$$C_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{P_N}\right)$$

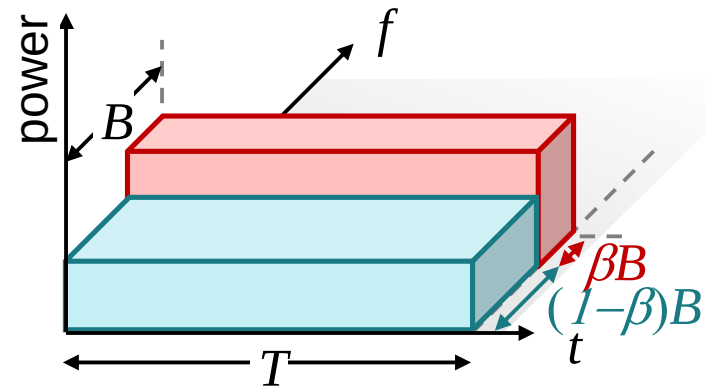
$$C_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{P_N}\right)$$

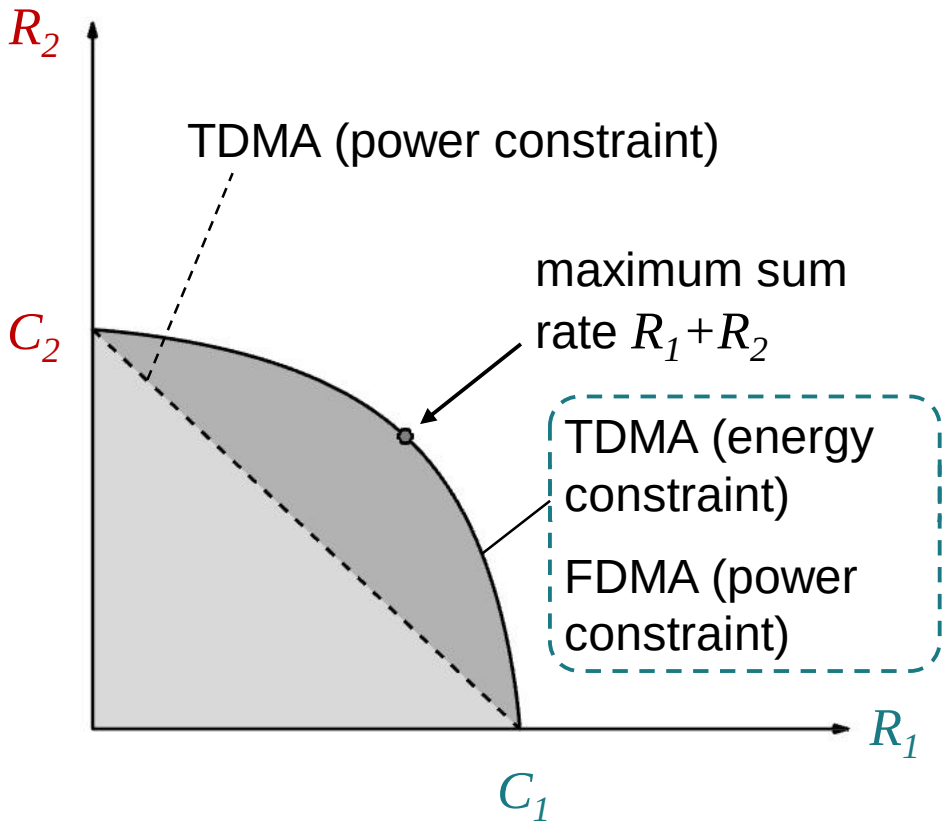
$$R_1 \le (1-\tau)B\log_2\left(1+\frac{|h_1|^2 P_1}{P_N(1-\tau)}\right)$$

$$R_2 \le \tau B\log_2\left(1+\frac{|h_2|^2 P_2}{P_N\tau}\right)$$

Capacity Region of Frequency Division Multiple Access (FDMA) with Power Constraint







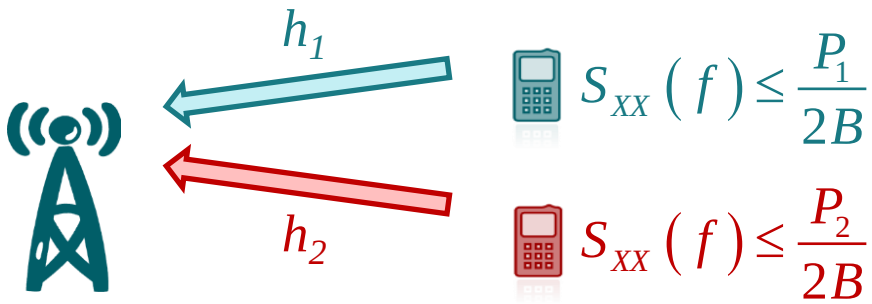
$$C_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{N_0 B}\right)$$

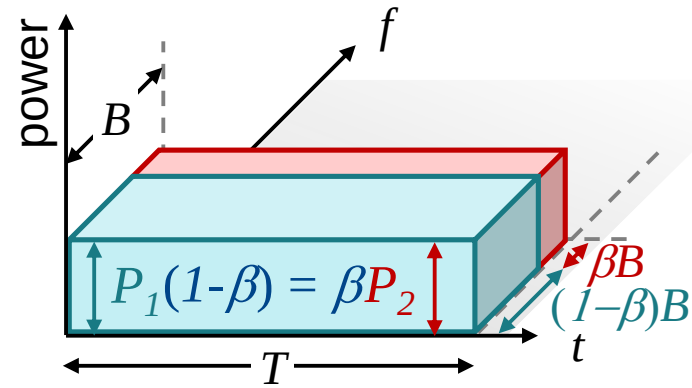
$$C_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{N_0 B}\right)$$

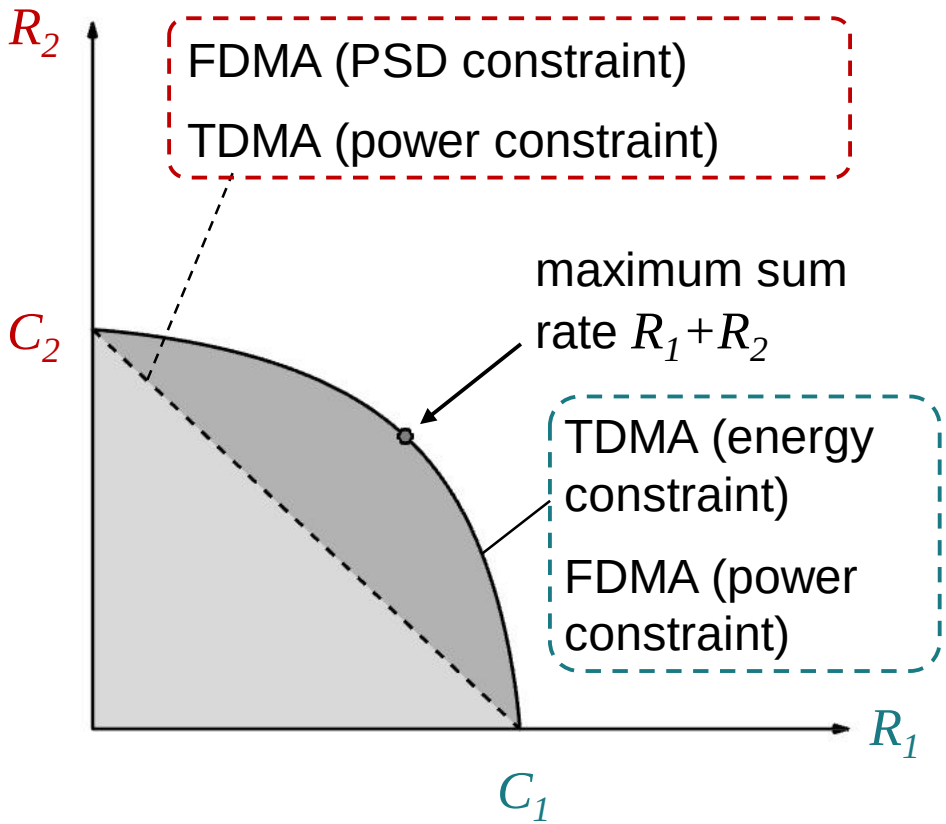
$$R_1 \le (1-\beta)B\log_2\left(1+\frac{|h_1|^2 P_1}{N_0(1-\beta)B}\right)$$

$$R_2 \le \beta B\log_2\left(1+\frac{|h_2|^2 P_2}{N_0 \beta B}\right)$$

Capacity Region of Frequency Division Multiple Access (FDMA) with Power Spectral Density Constraint







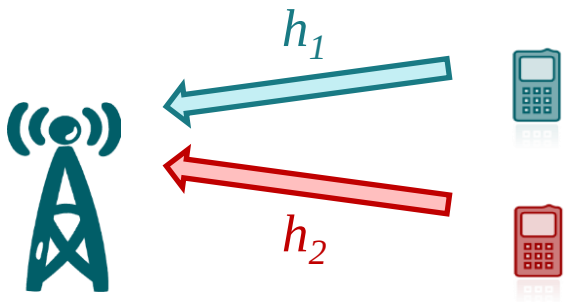
$$C_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{N_0 B}\right)$$

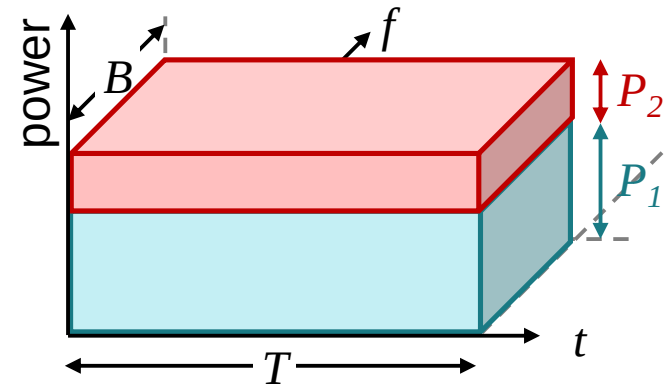
$$C_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{N_0 B}\right)$$

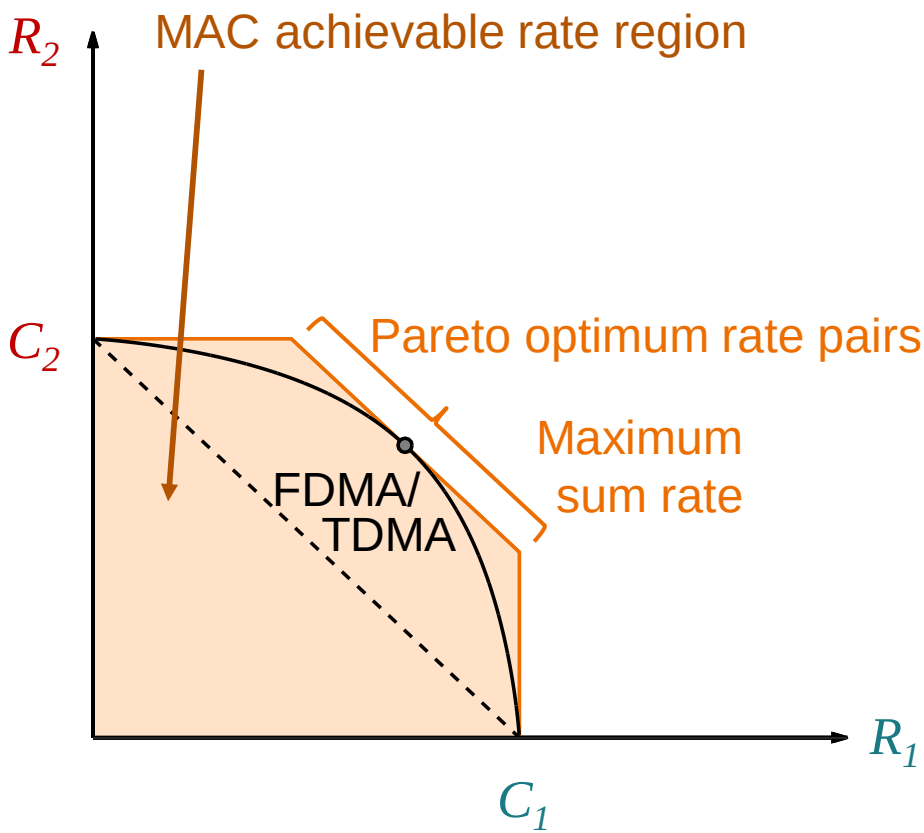
$$R_1 \le (1-\beta)B\log_2\left(1+\frac{|h_1|^2 P_1}{N_0 B}\right)$$

$$R_2 \le \beta B\log_2\left(1+\frac{|h_2|^2 P_2}{N_0 B}\right)$$

Capacity Region of the Two User Multiple Access Channel (MAC) (1)





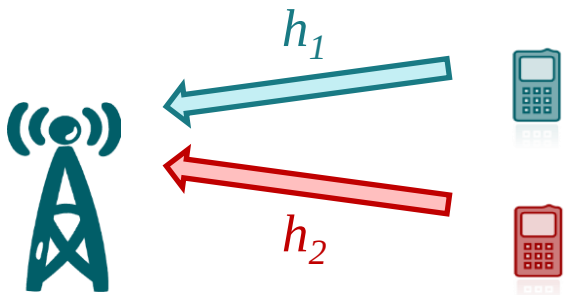


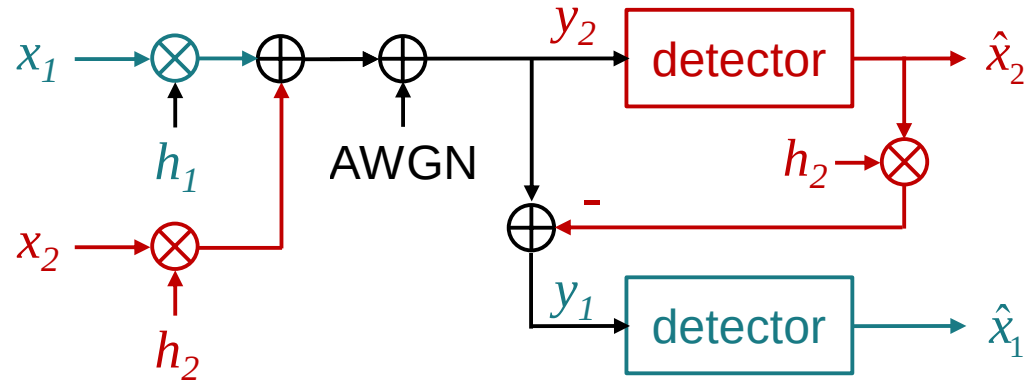
$$R_1 \le C_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{P_N}\right)$$

$$R_2 \le C_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{P_N}\right)$$

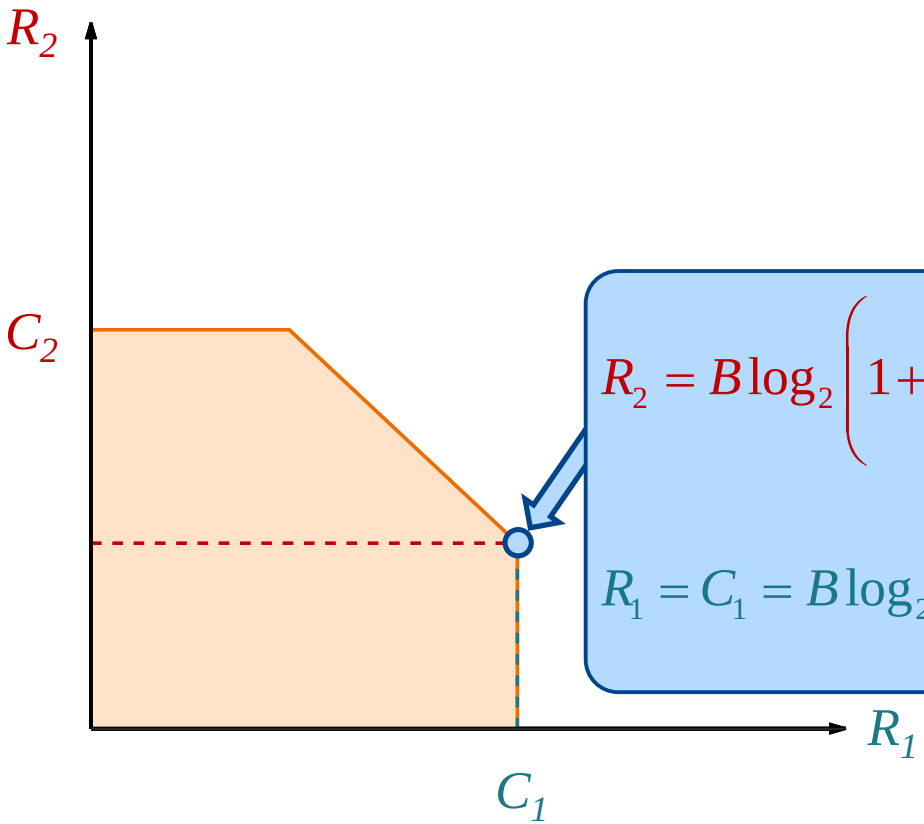
$$R_1 + R_2 \le B\log_2\left(1+\frac{|h_1|^2 P_1 + |h_2|^2 P_2}{P_N}\right)$$

Capacity Region of the Two User Multiple Access Channel (MAC) (2)





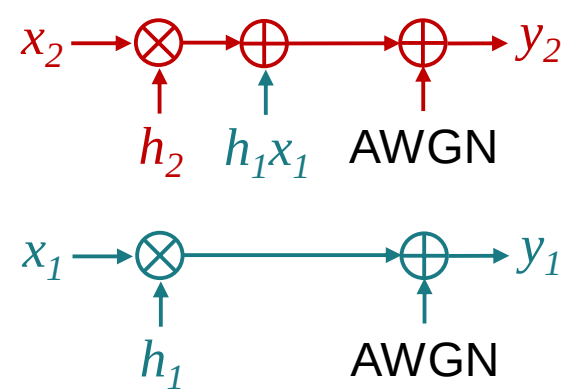
Successive interference cancellation



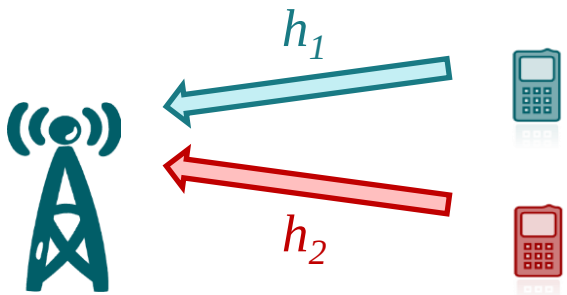
$$R_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{|h_1|^2 P_1 + P_N}\right)$$

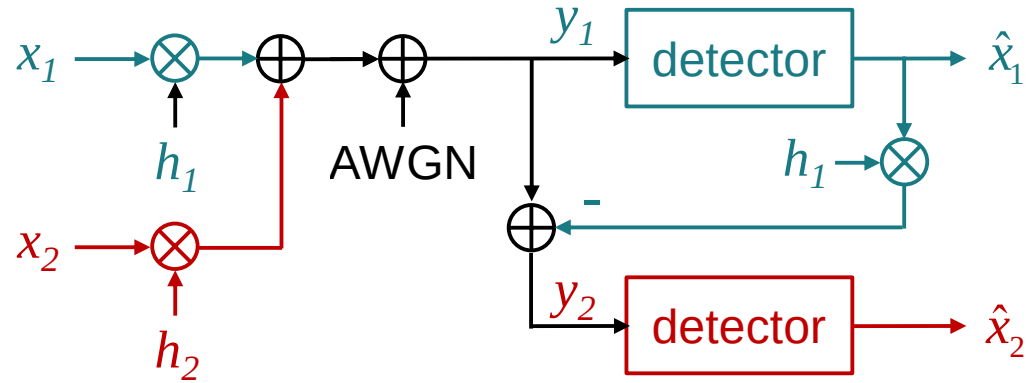
$$R_1 = C_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{P_N}\right)$$

Equivalent channels:

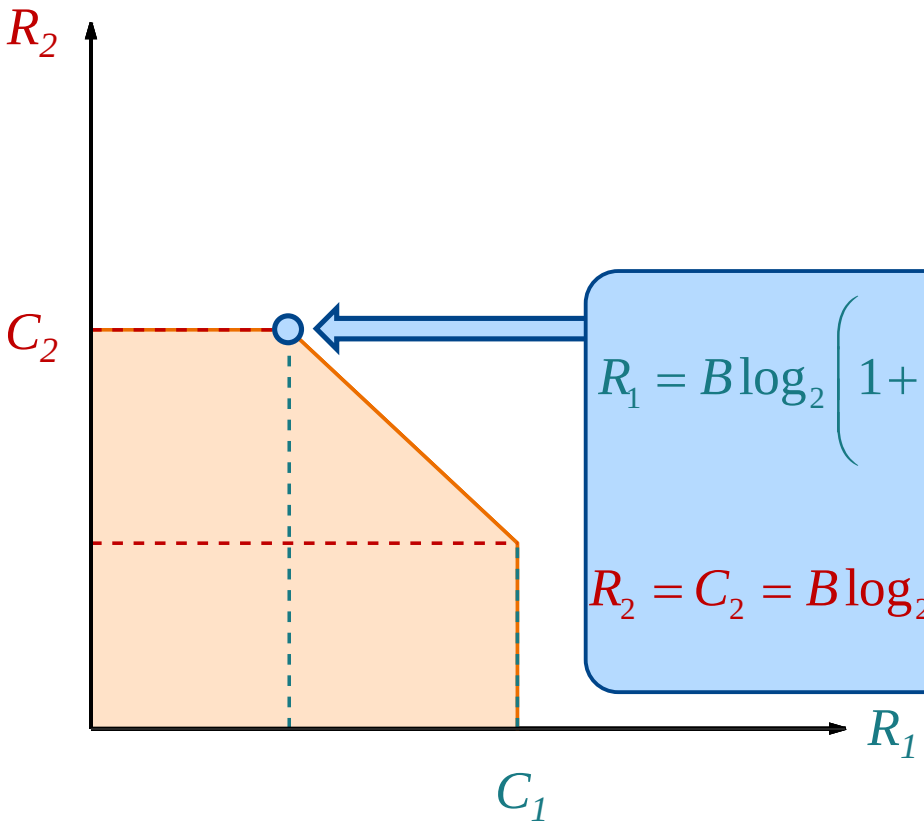


Capacity Region of the Two User Multiple Access Channel (MAC) (3)





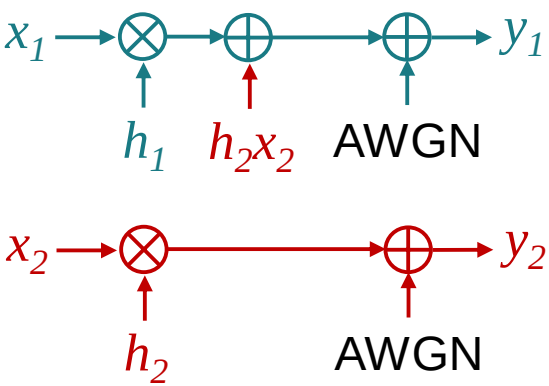
Successive interference cancellation



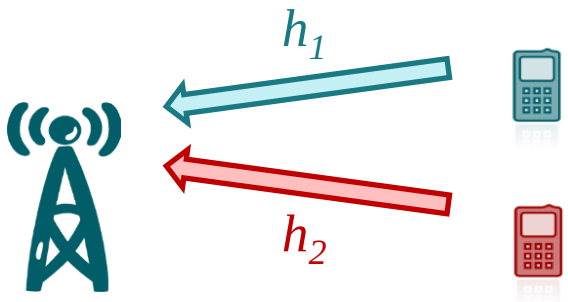
$$R_1 = B\log_2\left(1+\frac{|h_1|^2 P_1}{|h_2|^2 P_2 + P_N}\right)$$

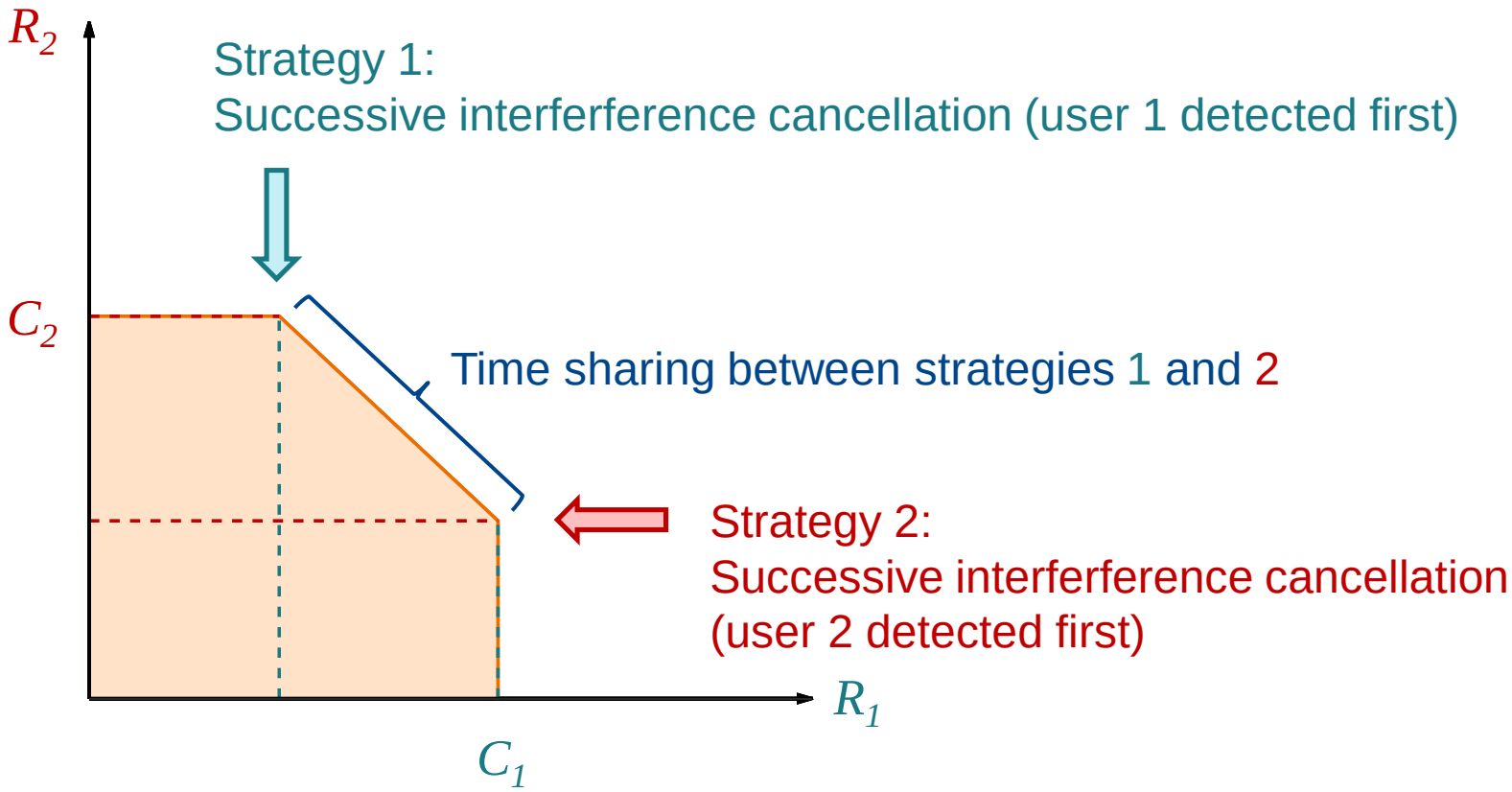
$$R_2 = C_2 = B\log_2\left(1+\frac{|h_2|^2 P_2}{P_N}\right)$$

Equivalent channels:

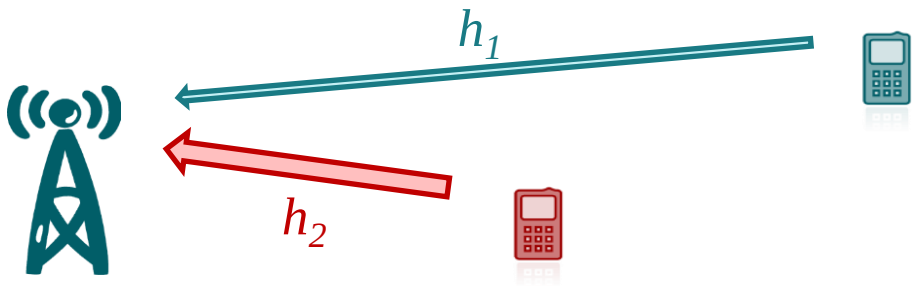


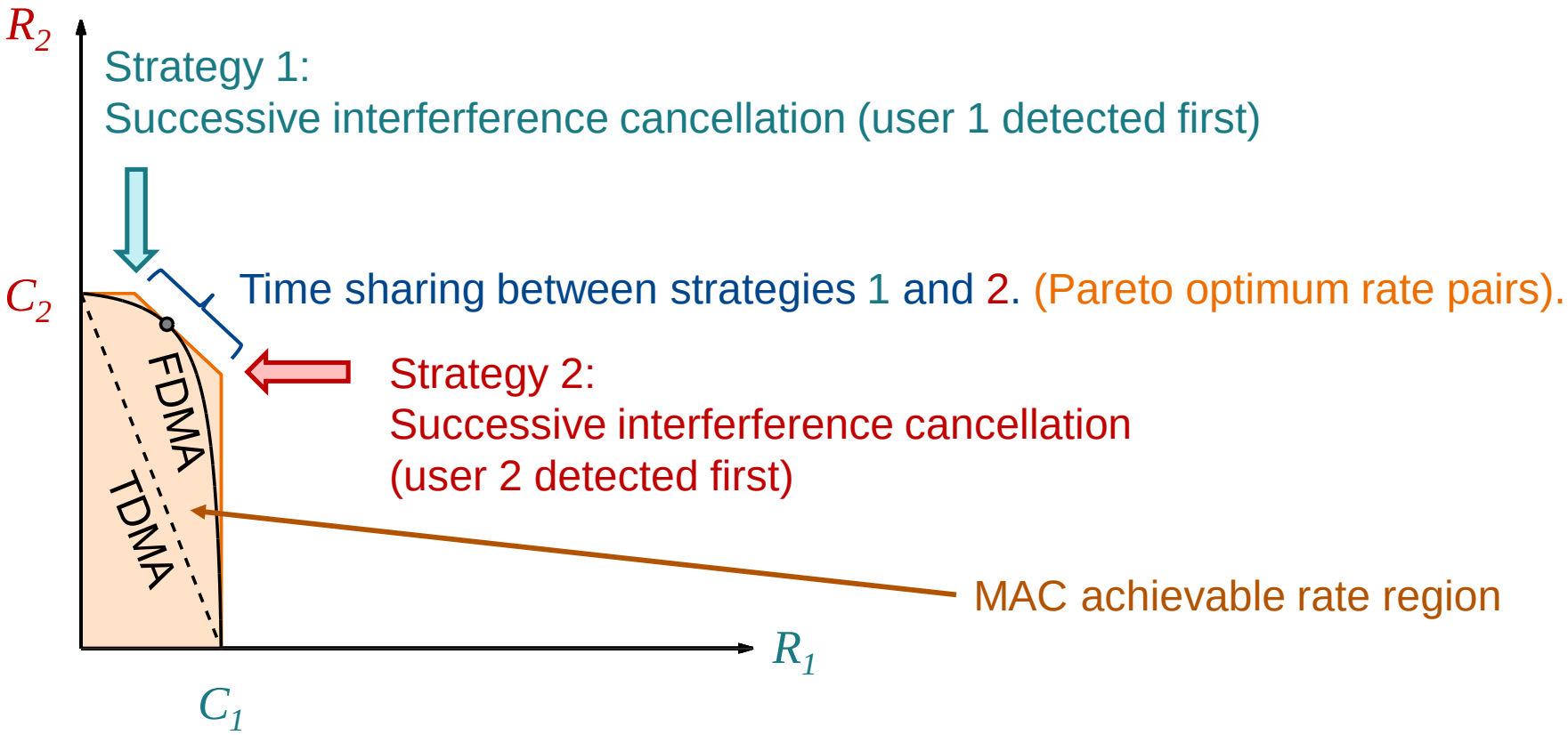
Capacity Region of the Two User Multiple Access Channel (MAC) (4)



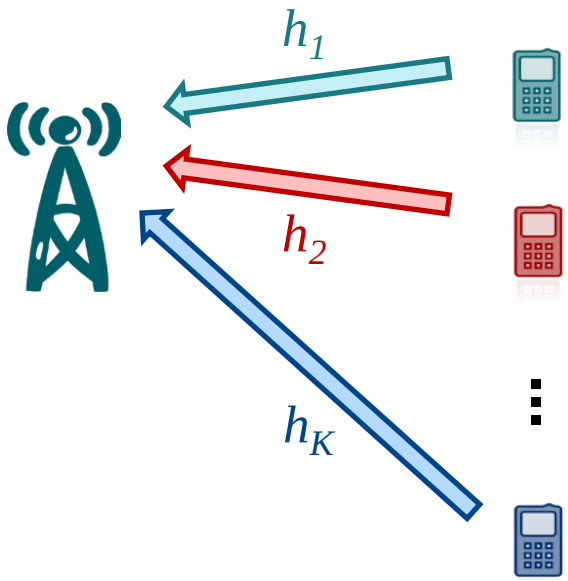


Capacity Region of the Two User Multiple Access Channel (MAC) (5)





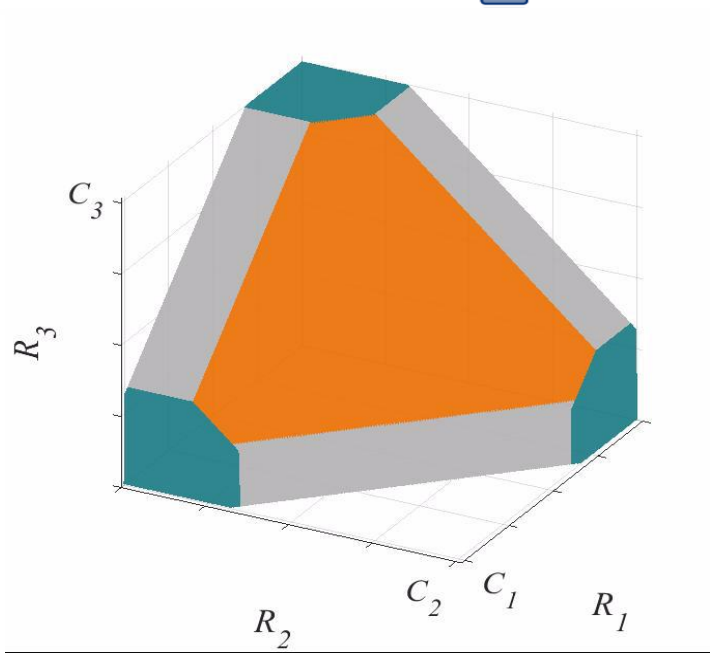
Capacity Region of the K User Multiple Access Channel (MAC)



$$R_k \le C_k = B\log_2\left(1+\frac{|h_k|^2 P_k}{P_N}\right)$$

$$k = 1,\ldots,K$$

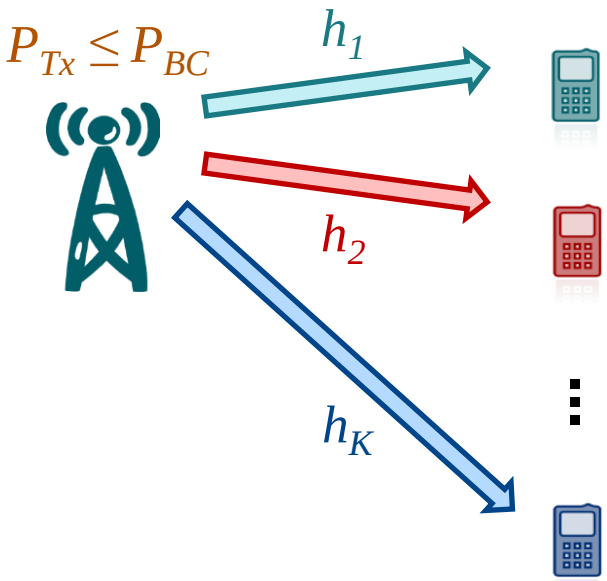
The rate of a user k cannot exceed the capacity of the individual channel of user k.



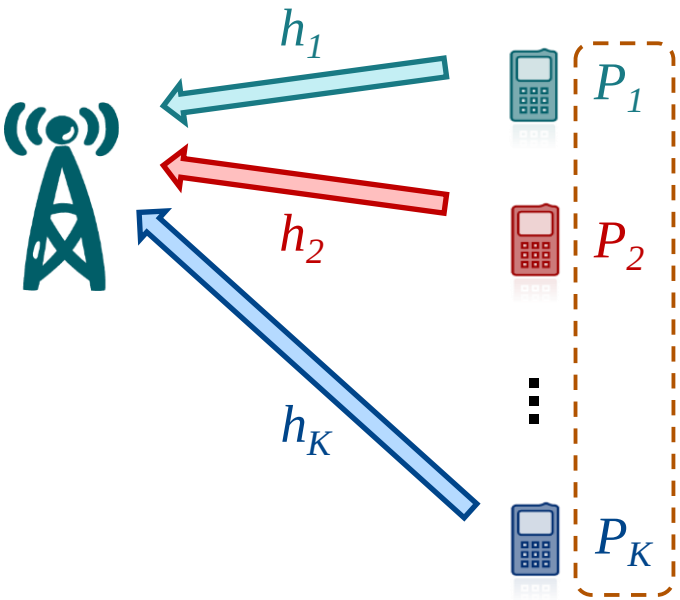
$$\sum_{k\subset\{1,\ldots,K\}} R_k \le B\log_2\left(1+\frac{\sum_{k\subset\{1,\ldots,K\}}|h_k|^2 P_k}{P_N}\right)$$

The sum rate of any subset $k \subset \{1,2,\ldots,K\}$ of users cannot exceed the capacity of a virtual superiuser with received power equal to the sum of the received powers from the users in the subset.

Capacity Region of the K User Broadcast Channel



Dual MAC



$$\sum_{k=1}^{K} P_k \leq P_{BC}$$

The capacity region of the broadcast channel can be obtained from the capacity region of the dual multiple access channel (MAC).

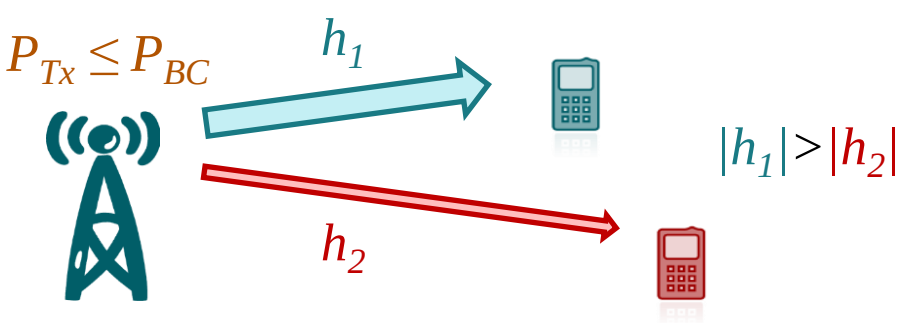
Dual MAC:

- ❑ Same channel coefficients h_k
- ❑ Same noise statistics at all receivers
- ❑ Sum power constraint

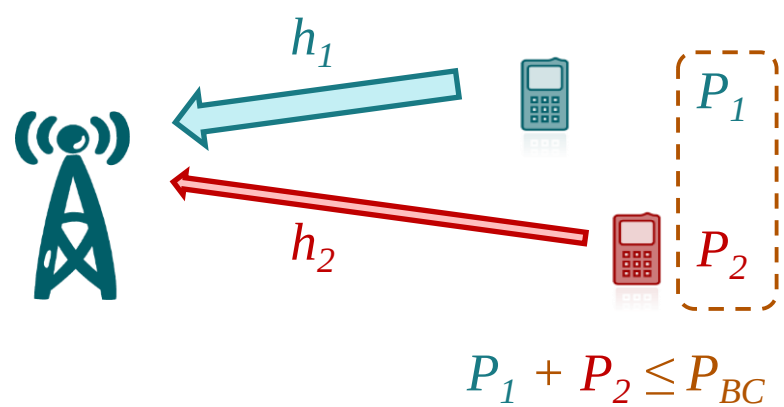
$$C_{BC} = \bigcup_{\left\{P_1, P_2, \ldots, P_K \,\middle|\, \sum_{k=1}^{K} P_k = P_{BC}\right\}} C_{MAC}$$

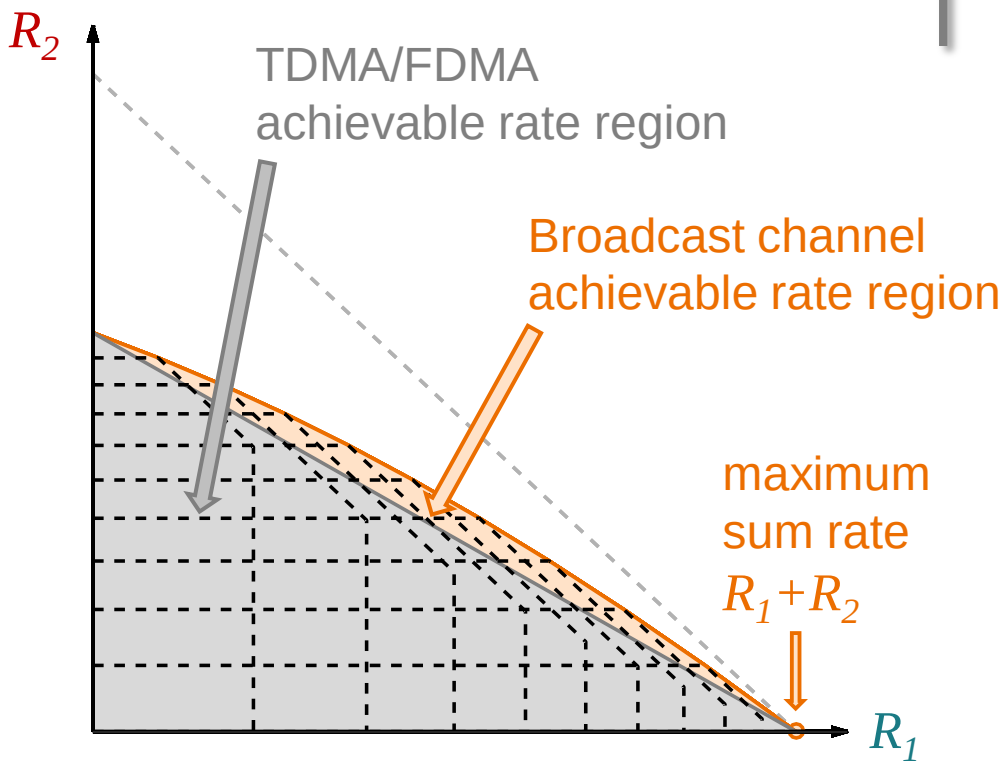
(Union over all possible power allocations which satisfy the sum power constraint).

Capacity Region of the 2 User Broadcast Channel (1)



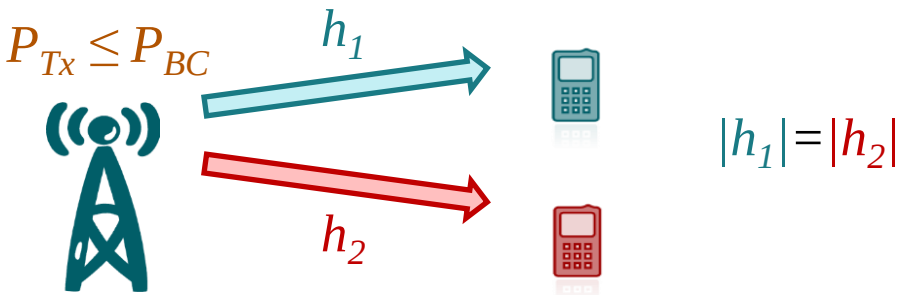
Dual MAC





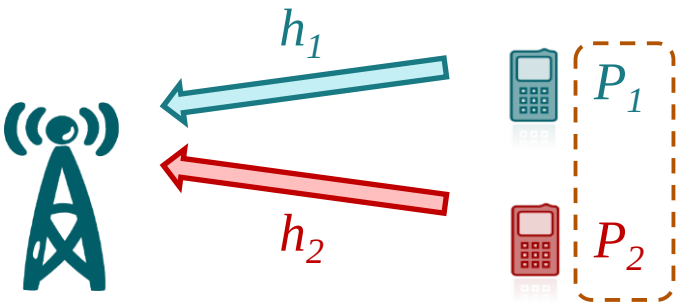
$$C_{BC} = \bigcup_{\{P_1, P_2 | P_1 + P_2 = P_{BC}\}} C_{MAC}$$

Capacity Region of the 2 User Broadcast Channel (2)

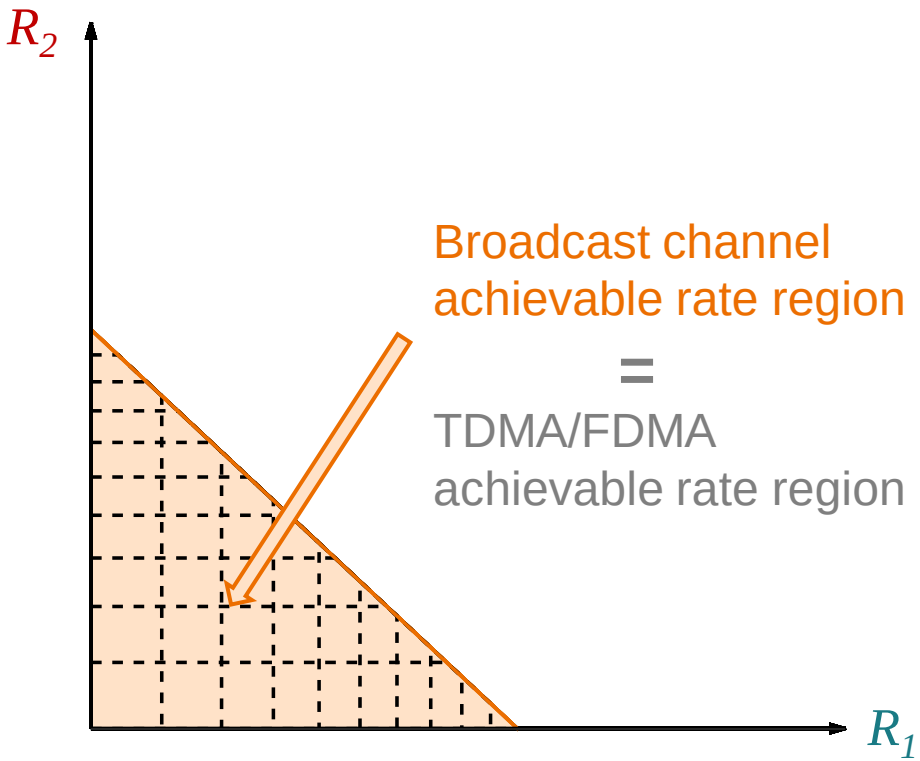


$|h_1| = |h_2|$

Dual MAC

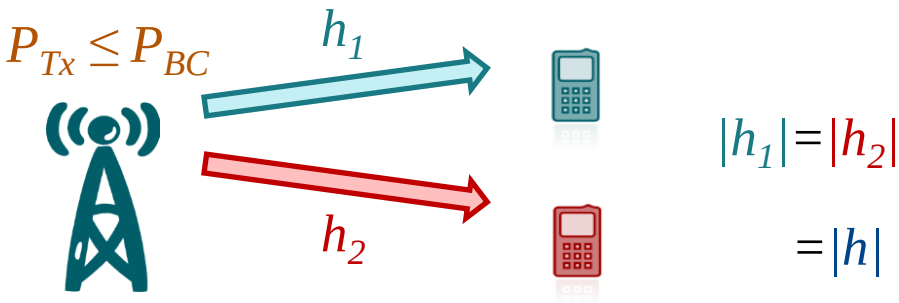


$P_1 + P_2 \leq P_{BC}$



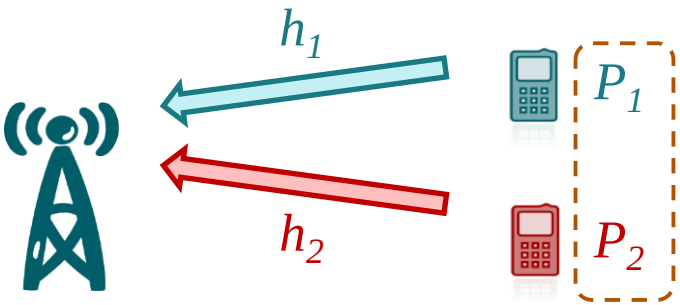
$$C_{BC} = \bigcup_{\{P_1, P_2 | P_1 + P_2 = P_{BC}\}} C_{MAC}$$

Capacity Region of the 2 User Broadcast Channel (3)

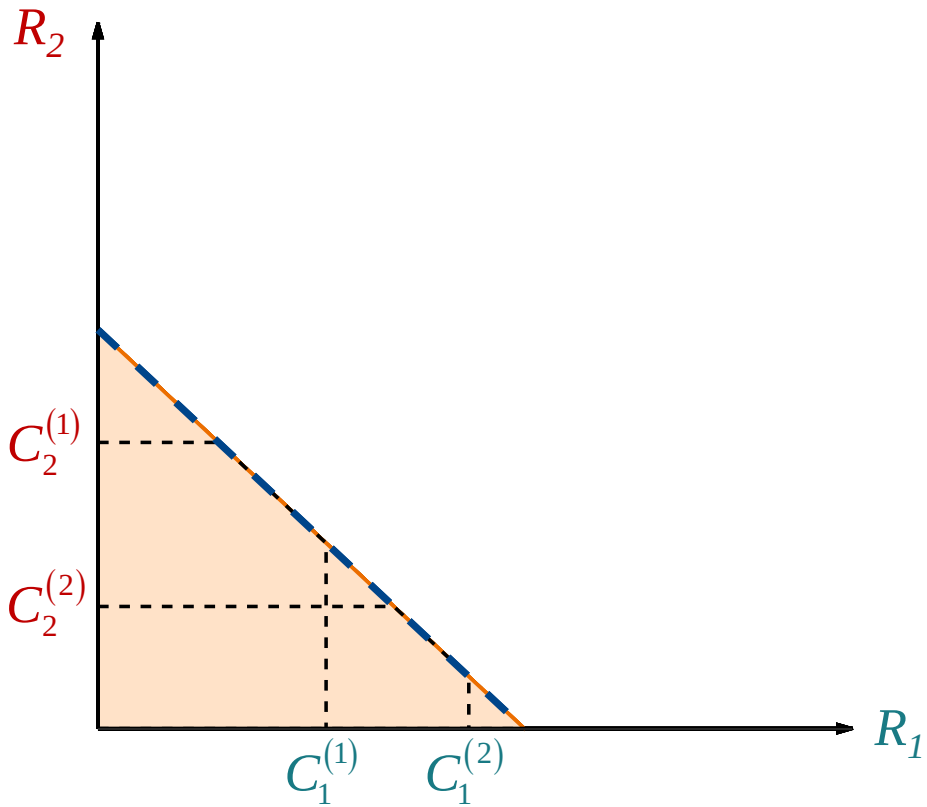


$$|h_1| = |h_2| = |h|$$

Dual MAC



$$P_1 + P_2 \le P_{BC}$$

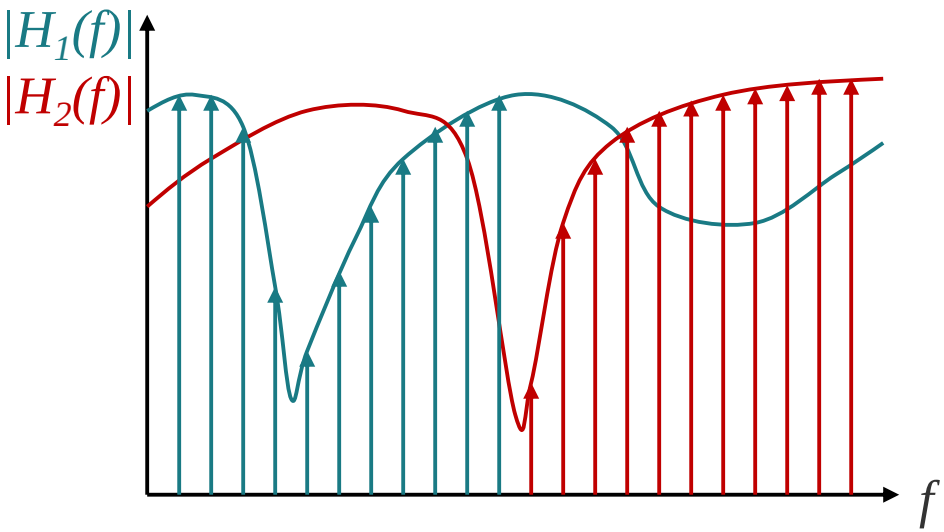


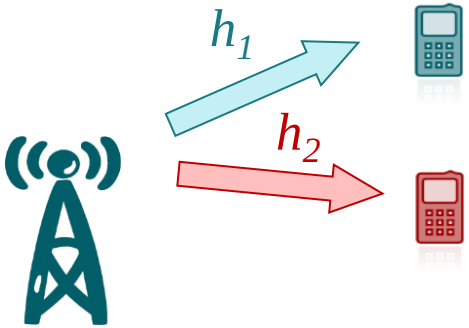
$$R_k \le C_k = B\log_2\left(1 + \frac{|h|^2 P_k}{P_N}\right), \quad k = 1,2$$

$$R_1 + R_2 \le B\log_2\left(1 + \frac{|h_1|^2 P_1 + |h_2|^2 P_2}{P_N}\right)$$

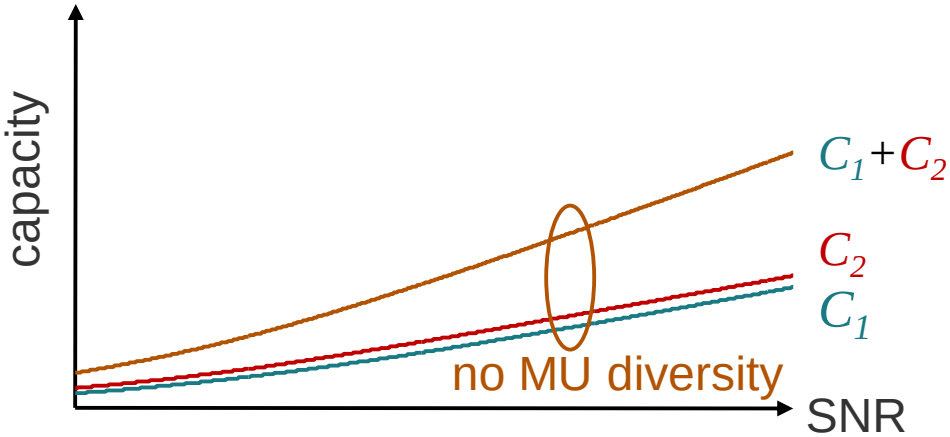
$$= B\log_2\left(1 + \frac{|h|^2 \overbrace{(P_1 + P_2)}^{P_{BC}=\text{const}}}{P_N}\right)$$

Multiuser (MU) Diversity (1)





Static FDMA: A fixed fraction of the subcarriers is allocated to each user.

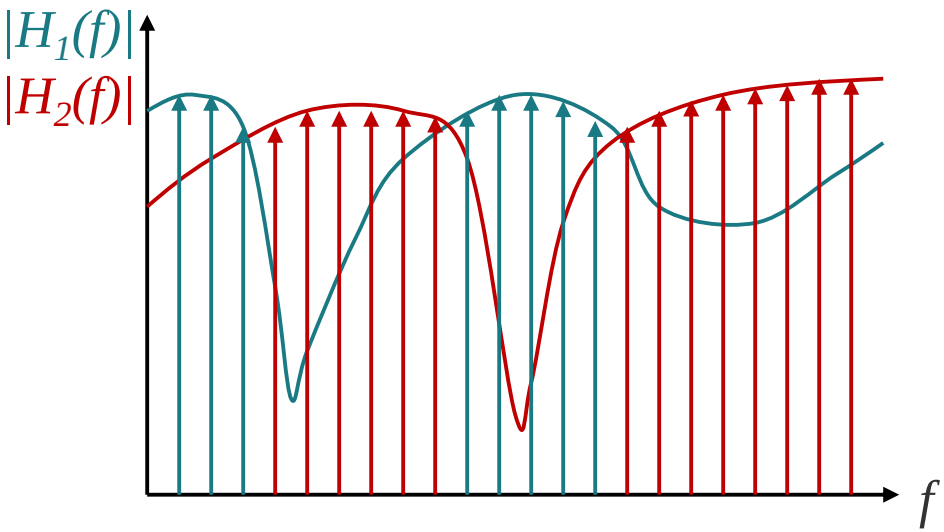


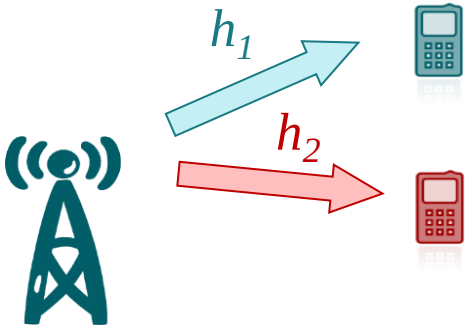
Sum capacity:

$$C = \sum_{k=1}^{K} C_k$$

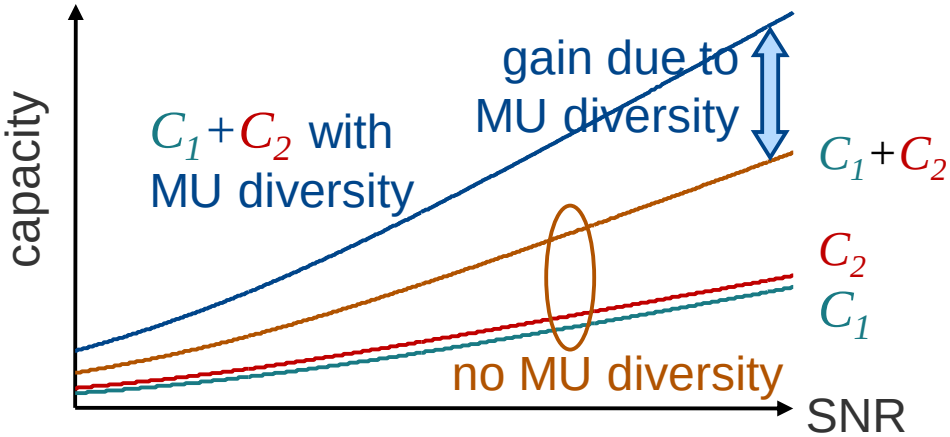
C_k: capacity of user k

Multiuser (MU) Diversity (2)





Dynamic FDMA: A subcarrier is allocated to the user with the highest capacity on that subcarrier (exploitation of MU diversity).



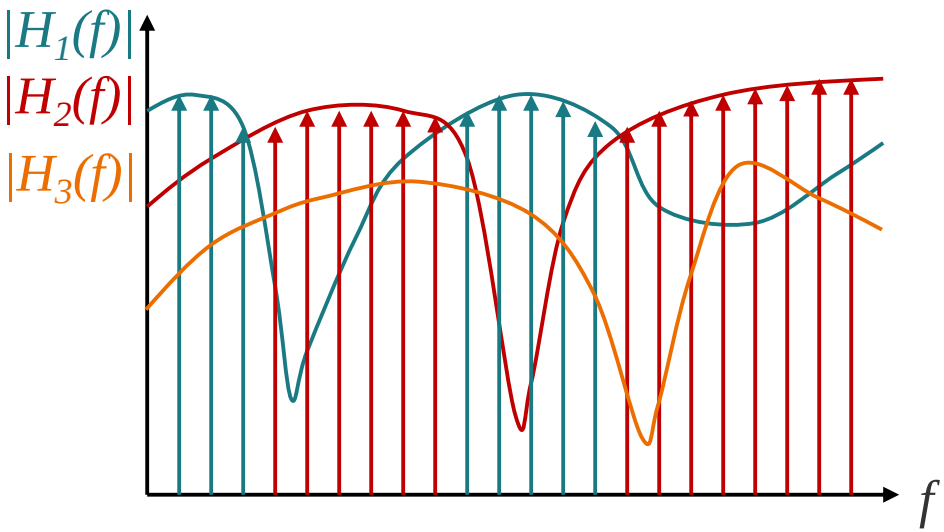
Sum capacity:

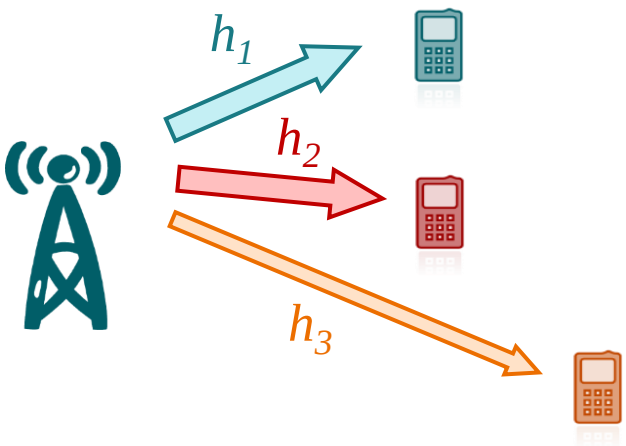
$$C = \sum_{k=1}^{K} C_k$$

C_k: capacity of user k

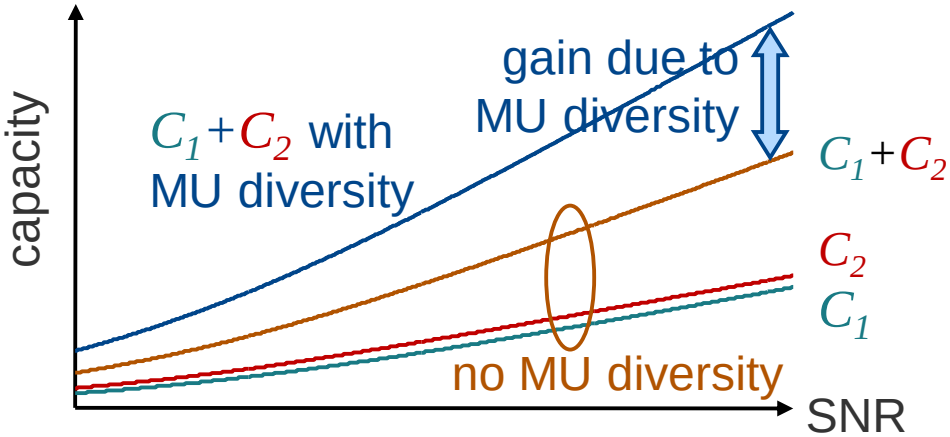
Multiuser (MU) Diversity (3)

INSTITUT FÜR NACHRICHTENTECHNIK





Dynamic FDMA: A subcarrier is allocated to the user with the highest capacity on that subcarrier (exploitation of MU diversity).

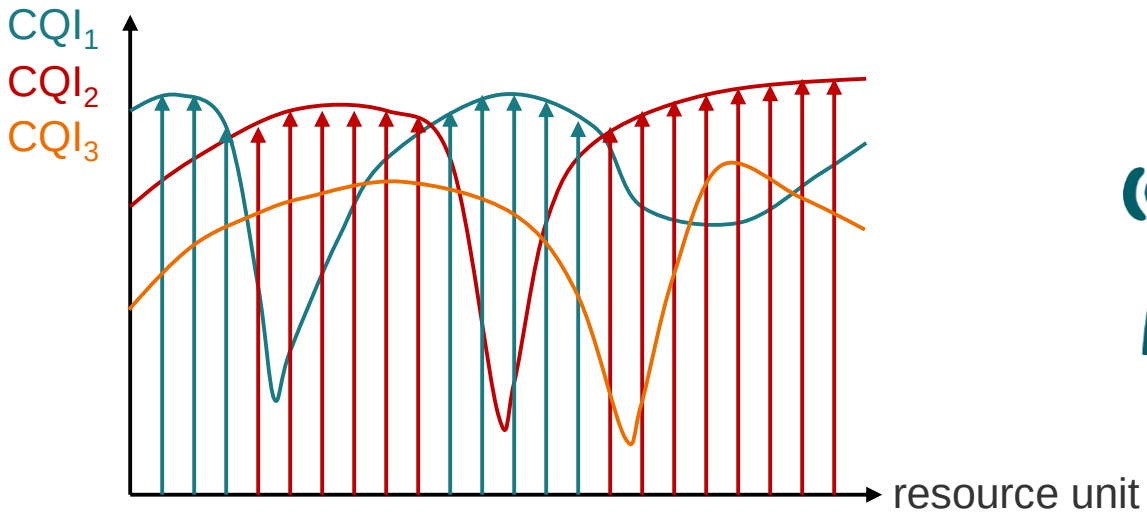


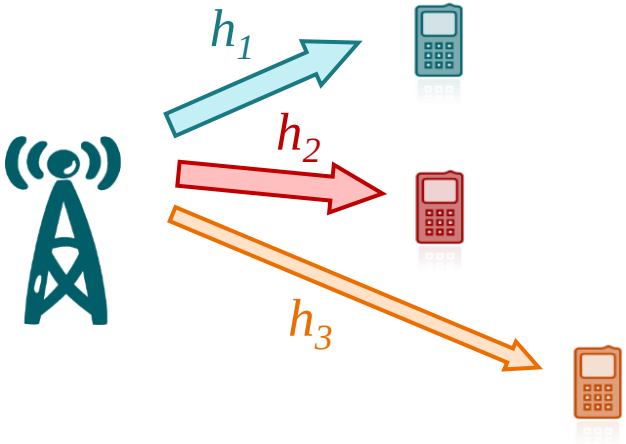
Advantage of MU diversity:
Sum capacity is increased.

Problem:
A user in a bad channel state may not be served for a long time (fairness problem, Quality of Service (QoS) is not guaranteed).

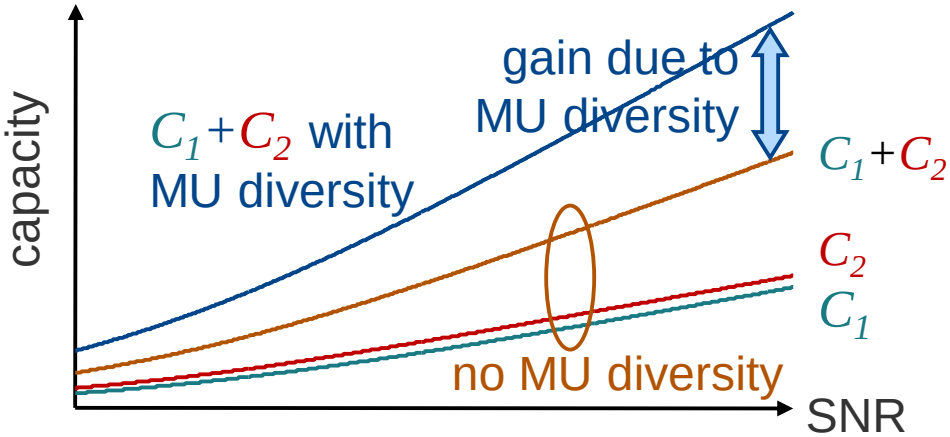
→ Exploitation of multiuser diversity is difficult for delay sensitive applications.

Multiuser (MU) Diversity (4)





A resource unit in time, frequency and space is allocated to the user with the highest capacity on that resource unit.



Advantage of MU diversity:
Sum capacity is increased.

Problem:
A user in a bad channel state may not be served for a long time (fairness problem, Quality of Service (QoS) is not guaranteed).

→ Exploitation of multiuser diversity is difficult for delay sensitive applications.

Sailing Towards Shannon's Capacity Limit

INSTITUT FÜR NACHRICHTENTECHNIK

speed of sound v_0

speed of light c_0

simple block codes since 1948

CCS telemetry channel coding
RS + convolutional code
1987

Turbo codes 1993 — 0.5 dB

2.2 dB

serially concatenated codes — 0.2 dB

irregular LDPC codes 2001 — 0.0045 dB

concatenated codes
Forney 1966

cut-off rate R_0

previously considered as practical limit for data rate

channel capacity C

maximum data rate for error free transmission

Shannon 1948

„A mathematical theory of communication"
Bell Syst. Technical Journal, 1948

data rate

Error detecting codes are often used in combination with automatic repeat request (ARQ) protocols. Errors are not corrected but a retransmission or transmission of additional redundancy is requested. This causes delay and requires a feedback channel.

In the context of transmission of speech or music, error detection can be used in combination with error concealment methods which aim at a reduced effect on subjective error perception.

Forward error correcting (FEC) codes correct errors at the decoder.

Often, FEC and ARQ are combined in order to request retransmissions if the decoding result still contains errors.

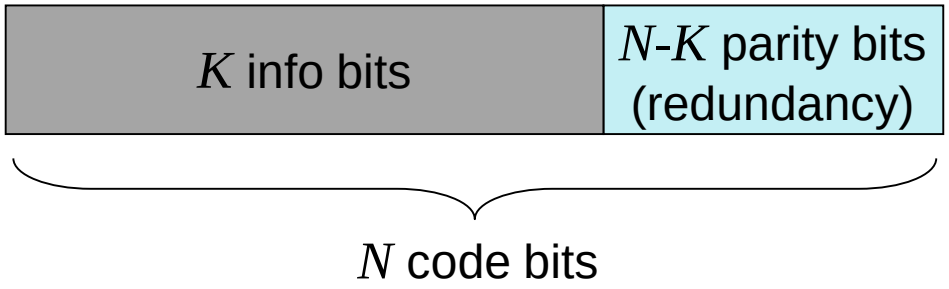
Channel Coding

INSTITUT FÜR NACHRICHTENTECHNIK

Redundancy is added to the information sequence which enables error detection and/or error correction at the receiver.

(*N*, *K*) code:

Code word:



Code rate: $R = \frac{K}{N}$

Examples:

- Block Codes
  - Repetition Code
  - Parity Check Code
  - Cyclic Redundancy Check (CRC) Codes
  - Hamming Code
  - Reed-Solomon Codes (e.g. Compact Disc)
  - BCH Codes
  - Low Density Parity Check Codes (LDPC Codes)
- Convolutional Codes (e.g. mobile communications)

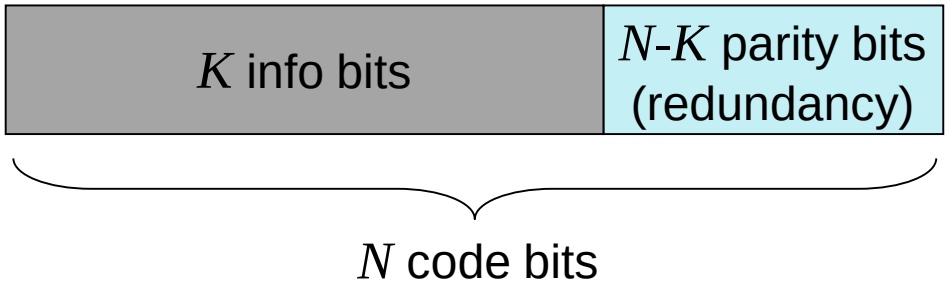
Block Codes and Convolutional Codes

INSTITUT FÜR NACHRICHTENTECHNIK

Block Codes:

A block of K information symbols is encoded into a block of N code symbols.

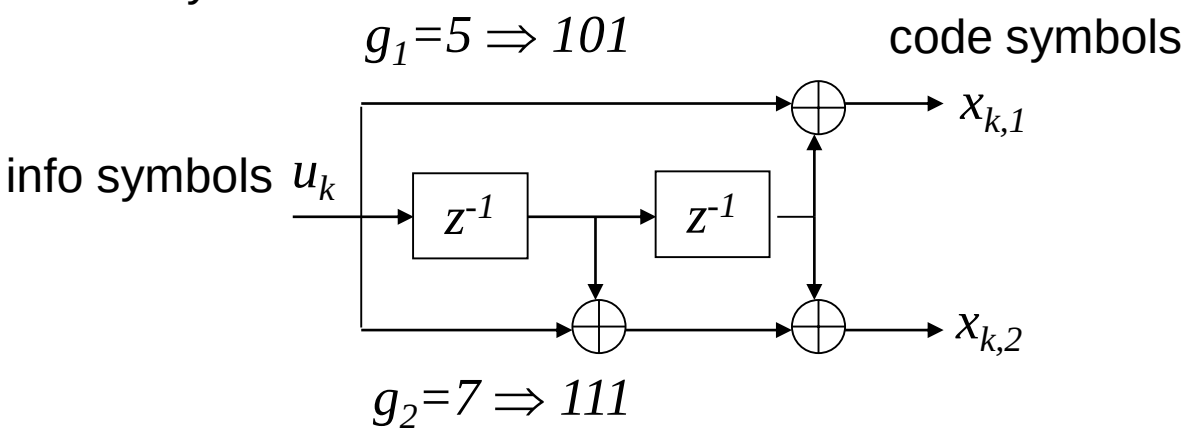
Code word:



Code rate: $R = \frac{K}{N}$

Convolutional Codes:

A continuous stream of information symbols is encoded into a stream of code symbols using a shift register. The encoder outputs N code symbols per K information symbols.



Code rate: $R = \frac{K}{N}$

Repetition Code

info bit	$\Rightarrow$	code word
0	$\Rightarrow$	000
$K=1$		$N=3$

$$R = \frac{K}{N} = \frac{1}{3}$$

Majority decision at the receiver $\Rightarrow$ N should be odd.

Two code words differ in N positions.
$\Rightarrow$ Code has minimum Hamming distance $d_{min}=N$.
$\Rightarrow$ (N, K, d_{min}) code = (3,1,3) code.

$d_{min}-1$ errors can be detected.
$(d_{min}-1)/2$ errors can be corrected.

Single Parity Check Code

info bits u_1u_2	parity bit p
00	0
01	1
10	1
11	0

$p = u_1 \oplus u_2$ (mod 2 addition)

$d_{\min} = 2$

$$R = \frac{K}{N} = \frac{2}{3}$$

$d_{min}-1 = 1$ error can be detected.

No errors can be corrected.

$\Rightarrow$ Error detecting code.

(7,4,3) Hamming Code

Code word: $\mathbf{x} = [u_1, u_2, u_3, u_4, p_1, p_2, p_3]$

Encoding:

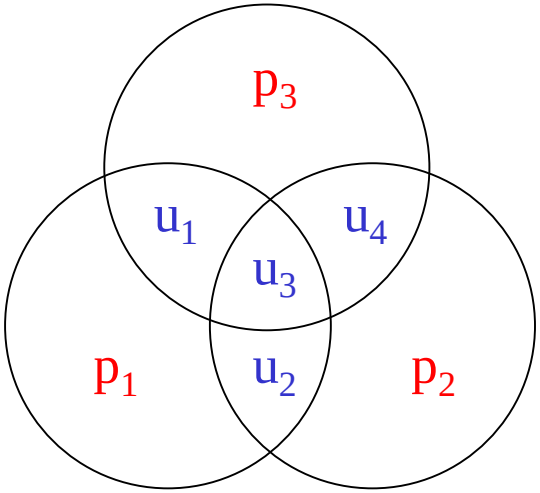
$p_1 = u_1 \oplus u_2 \oplus u_3$

$p_2 = u_2 \oplus u_3 \oplus u_4$

$p_3 = u_1 \oplus u_3 \oplus u_4$

$$R = \frac{K}{N} = \frac{4}{7}, \quad d_{min}=3$$

Van diagram:



Decoding: Compute „syndrom“:

$s_1 = u_1 \oplus u_2 \oplus u_3 \oplus p_1$

$s_2 = u_2 \oplus u_3 \oplus u_4 \oplus p_2$

$s_3 = u_1 \oplus u_3 \oplus u_4 \oplus p_3$

$d_{min}-1/2 = 1$ error can be corrected.

$\Rightarrow$ 1 error correcting code.

syndrom s_3,s_2,s_1	error in
000	
001	p_1
010	p_2
100	p_3

syndrom s_3,s_2,s_1	error in
101	u_1
011	u_2
111	u_3
110	u_4

Hamming Codes

$(N, K, d_{min})_2 = (2^r - 1, 2^r - r - 1, 3)_2$

Hamming codes are single error correcting codes.

Example:

$(7, 4, 3)_2$ Hamming code:

$r=3$: $(2^r - 1, 2^r - r - 1, 3)_2 = (2^3 - 1, 2^3 - 3 - 1, 3)_2 = (7, 4, 3)_2$

Representation of Binary Data

$\oplus$ Operator

$$u_1 \oplus u_2$$

$u_i \in \{0,1\}$

$\oplus$		u_1	
		0	1
u_2	0	0	1
	1	1	0

$0 \rightarrow +1$
$1 \rightarrow -1$

$u_i \in \{+1,-1\}$

$\oplus$		u_1	
		+1	-1
u_2	+1	+1	-1
	-1	-1	+1

$0 \rightarrow$ blue
$1 \rightarrow$ red

alternative operator:
$u_1 \oplus u_2 \Rightarrow u_1 \cdot u_2$

$u_i \in \{$blue, red$\}$

$\oplus$		u_1	
u_2			

$\cdot$		u_1	
		+1	-1
u_2	+1	+1	-1
	-1	-1	+1

Repetition Code with q-ary Symbol Alphabet

Example: $q=4$, $\mathcal{A}=\{0, 1, 2, 3\}$

info symbol	$\Rightarrow$	code word
0	$\Rightarrow$	000
1	$\Rightarrow$	111
2	$\Rightarrow$	222
3	$\Rightarrow$	333
$K=1$		$N=3$

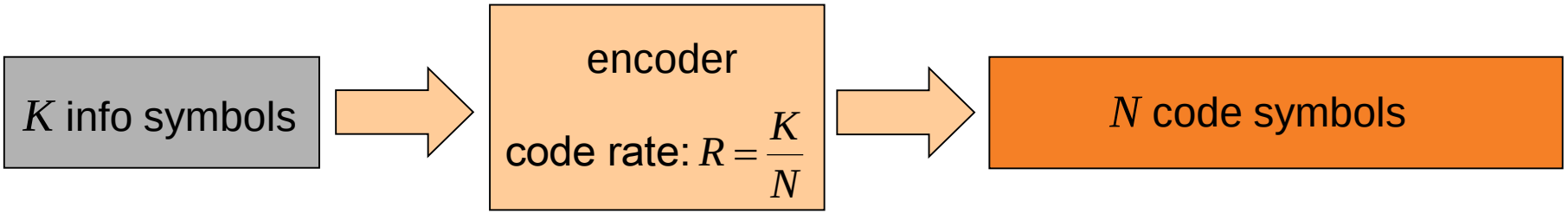
$$R = \frac{K}{N} = \frac{1}{3}$$

The two code words differ in N positions.

$\Rightarrow$ Code has minimum Hamming distance $d_{min}=N$.

$\Rightarrow$ $(N, K, d_{min})_q$ code = $(3,1,3)_4$ code.

$(N, K, d_{min})_q$ -Code



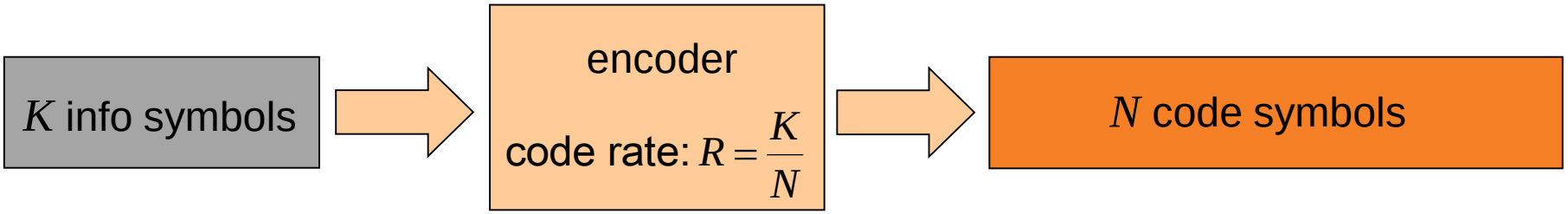
- Usually: $N>K$
  (redundancy is added)
- Any two codewords differ in at least d_{min} symbols
- q-ary alphabet $\mathcal{A}=\{x_1, \ldots, x_q\}$
  e.g. binary code: $q=2$
  $\mathcal{A}=\{0, 1\}$ or $\mathcal{A}=\{+1, -1\}$

$(N, K, d_{min})_q$ -Code

Number of info words: q^K

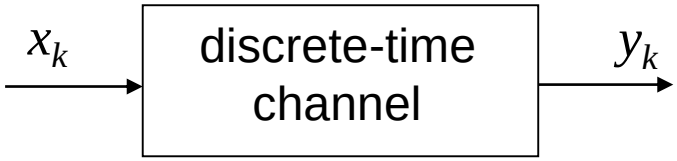
Number of codewords: $q^K = q^{RN}$

Number of possible uncoded transmit words: $q^N > q^K$



The code is a subset of the possible uncoded transmit words.
The fact, that not all possible transmit sequences are codewords allows for error detection and error correction, respectively.

Code Rate and Data Rate



One code symbol x_k can be transmitted per channel use k (1 channel use: 1 symbol is transmitted through the discrete-time channel)

Code symbol duration: T_s

1 channel use per T_s (T_s depends on channel bandwidth)

q-ary info symbols: $\log_2(q)$ info bit/info symbol

Code rate: $$R = \frac{K}{N}\frac{\text{info symbols}}{\text{code symbol}} = \frac{K}{N}\frac{\text{info symbols}}{\text{channel use}} = \frac{K\log_2 q}{N}\frac{\text{info bit}}{\text{channel use}}$$

Data rate: $$R_b = \frac{K\log_2 q}{N}\frac{\text{info bit}}{\text{channel use}}\frac{1\,\text{channel use}}{T_s} = \frac{K\log_2 q}{NT_s}\left[\frac{\text{info bit}}{s}\right]$$

Particularly for binary codes, the cardinality q of the symbol alphabet is often dropped, e.g. we write (N, K, d_{min})-code.

If only two parameters are given, the minimum Hamming distance is dropped, i.e. (N, K)-code.

Systematic and Non-Systematic Encoders

An encoder is called a *systematic encoder*, if the information sequence is an explicit part of the codeword. Otherwise, the encoder is called *non-systematic*.

K info symbols → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

$u_1, u_2, \ldots, u_K$

$0, \ 1, \ 1, \ 0$

systematic encoder: $\underbrace{u_1, u_2, \ldots, u_K}_{\text{info symbols}} \underbrace{p_1, p_2, \ldots, p_{N-K}}_{\text{parity symbols}}$

$\underbrace{0, \ 1, \ 1, \ 0,}_{\text{info symbols}} \ \underbrace{0, \ 0, \ 1}_{\text{parity symbols}}$

non-systematic encoder: $x_1, x_2, \quad \ldots \quad , x_N$

$1, \ 1, \ 1, \ 1, \ 1, \ 1, \ 1$

Systematic and Non-Systematic Encoders

An encoder is called a *systematic encoder*, if the information sequence is an explicit part of the codeword. Otherwise, the encoder is called *non-systematic*.

K info symbols → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

Example: Two representations of the systematic (3,2,2) single parity check code

info $u_1 u_2$	code 1 $u_1 u_2 p$	code2 $p\ u_1 u_2$
00	000	000
01	011	101
10	101	110
11	110	011

Code and Encoder

info	systematic encoder	non-systematic encoder
0000	0000000	0001111
0001	0001111	0010110
0010	0010110	0011001
0011	0011001	0100101
0100	0100101	0101010
0101	0101010	0110011
0110	0110011	0111100
0111	0111100	1000011
1000	1000011	1001100
1001	1001100	1010101
1010	1010101	1011010
1011	1011010	1100110
1100	1100110	1101001
1101	1101001	1110000
1110	1110000	1111111
1111	1111111	0000000

A *block code* is defined by the set of codewords.
The code may be obtained by different *encoders*, i.e. different mappings of infowords to codewords.

The encoder mapping rule is unique and time-invariant.

Two block codes are called identical, if their sets of codewords are identical.

Identical codes have the same error detection and error correction properties.

The characteristics of a code such as error correction and detection capabilities depend on the Hamming distances of the codewords. Hence, error correction and detection capabilities are determined by the set of codewords, not by the particular mapping of information words to codewords.

We distinguish between *code properties* such as Hamming distances and *encoder properties* such as the mapping of information words to codewords.

Hamming Distance, Minimum Hamming Distance

The Hamming distance $d_H(\mathbf{x}_k,\mathbf{x}_l)$ of two codewords $\mathbf{x}_k$ and $\mathbf{x}_l$ is the number of symbols in which the two codewords differ.

The minimum Hamming distance d_{min} of a code $\mathcal{C}$ is the minimum number of symbols in which two codewords $\mathbf{x}_k$ and $\mathbf{x}_l$ of the code differ among any pairs of codewords of the code:

$$d_{min} = \min \{d_H(\mathbf{x}_k,\mathbf{x}_l) \mid \mathbf{x}_k,\mathbf{x}_l \in \mathcal{C},\, k \neq l\}$$

Example: (3,2) Single parity check code

info bits u_1u_2	codeword $\mathbf{x}_k$
00	*000*
01	*011*
10	*101*
11	*110*

$d_{min} = 2$

Hamming Weight

The Hamming weight $w_H(\mathbf{x})$ of a codeword $\mathbf{x}$ is the number of symbols in the codeword which are unequal zero:

$$w_H(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0})$$

The minimum Hamming weight w_{min} of a code is the minimum number of symbols in a codeword which are unequal zero among all codewords of the code except the all zero code word.

Properties of Hamming Distance and Hamming Weight

$$d_H(\mathbf{x}_1,\mathbf{x}_2) = d_H(\mathbf{x}_2,\mathbf{x}_1)$$

$$0 \leq d_H(\mathbf{x}_1,\mathbf{x}_2) \leq N$$

$$d_H(\mathbf{x}_1,\mathbf{x}_2) = 0 \Leftrightarrow \mathbf{x}_1 = \mathbf{x}_2$$

$$d_H(\mathbf{x}_1,\mathbf{x}_3) \leq d_H(\mathbf{x}_1,\mathbf{x}_2) + d_H(\mathbf{x}_2,\mathbf{x}_3) \quad \text{(Triangle inequality)}$$

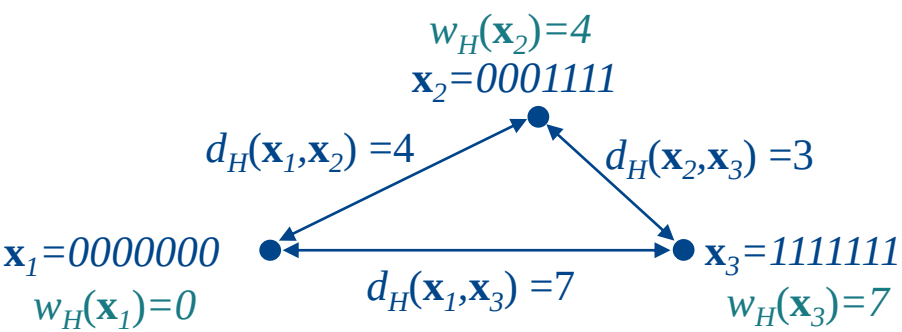
$$0 \leq w_H(\mathbf{x}) \leq N$$

$$w_H(\mathbf{x}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$$

$$w_H(\mathbf{x}_1+\mathbf{x}_2) \leq w_H(\mathbf{x}_1) + w_H(\mathbf{x}_2)$$

$$d_H(\mathbf{x}_1,\mathbf{x}_2) = w_H(\mathbf{x}_1-\mathbf{x}_2)$$

Example:



$$d_H(\mathbf{x}_1,\mathbf{x}_2) + d_H(\mathbf{x}_2,\mathbf{x}_3) = 4+3 = 7 = d_H(\mathbf{x}_1,\mathbf{x}_3)$$

$$d_H(\mathbf{x}_1,\mathbf{x}_3) + d_H(\mathbf{x}_3,\mathbf{x}_2) = 7+3 = 10 > d_H(\mathbf{x}_1,\mathbf{x}_2) = 4$$

$$w_H(\mathbf{x}_1) + w_H(\mathbf{x}_2) = 0 + 4 = 4 = w_H(\mathbf{x}_1+\mathbf{x}_2)$$

$$w_H(\mathbf{x}_2) + w_H(\mathbf{x}_3) = 4 + 7 = 11 > w_H(\mathbf{x}_2+\mathbf{x}_3) = 3$$

Decoding Spheres

Example: Rate $R=1/3$ repetition code

u	$x_1x_2x_3$	Hamming weight w_H
0	000	0
1	111	3

Minimum Hamming distance $d_{min}=3$

Decoding spheres: Spheres with radius $r=\lfloor (d_{\min}-1)/2 \rfloor$ centered around codewords.

Decoding:
All received words within a decoding sphere are mapped to the codeword at the center of the sphere.

Definition:

The sphere $K_r(\mathbf{x})$ with radius r around word $\mathbf{x}$ is defined as the union of all words with Hamming distance $d_H \leq r$ to $\mathbf{x}$:

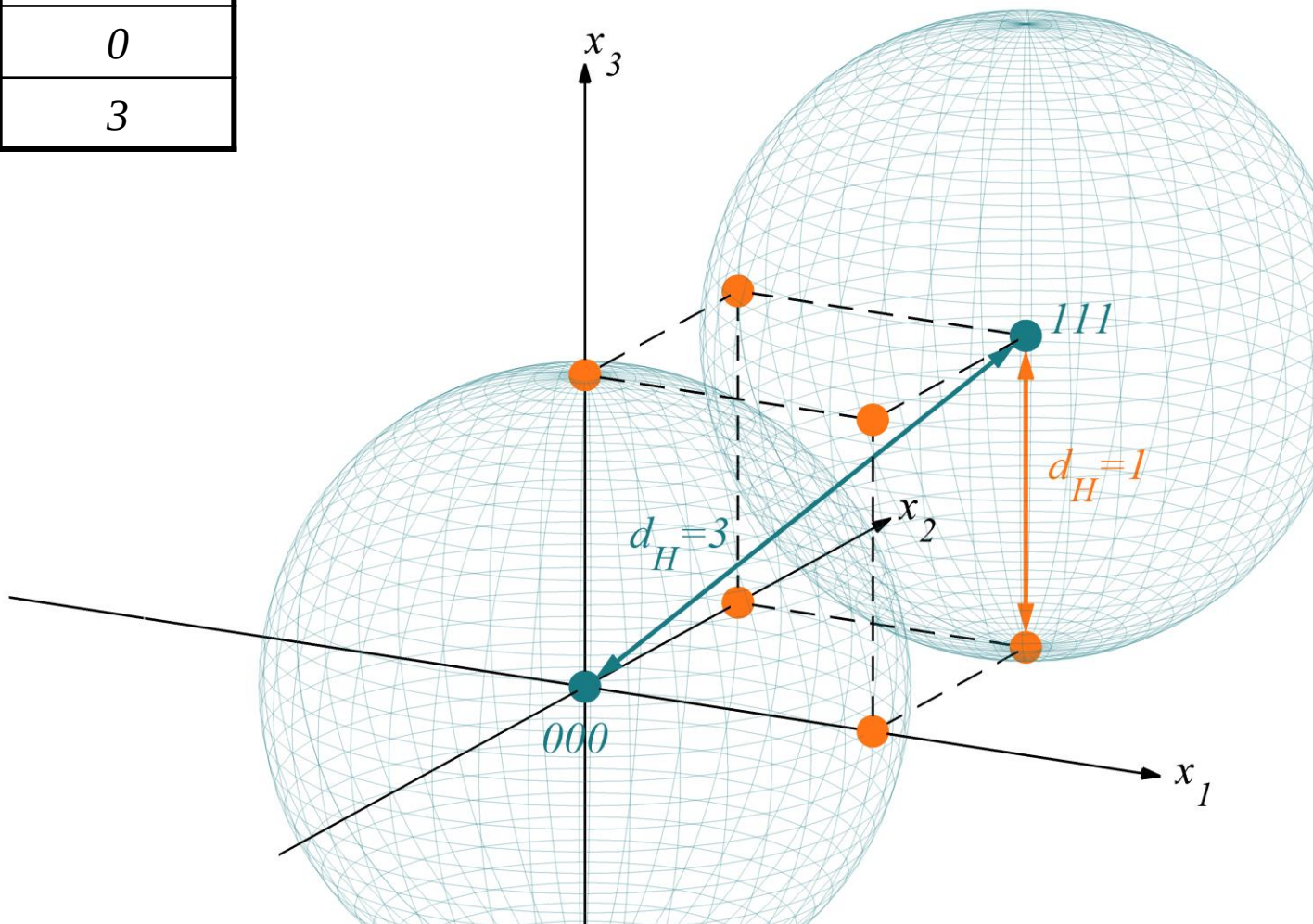
$$K_r(\mathbf{x}) = \{\mathbf{y} \mid d_H(\mathbf{x},\mathbf{y}) \leq r\}$$

Decoding Spheres

Example: Rate $R=1/3$ repetition code

u	$x_1x_2x_3$	Hamming weight w_H
0	000	0
1	111	3

Minimum Hamming distance $d_{min}=3$



Decoding Spheres

Example: Rate $R=1/2$ repetition code

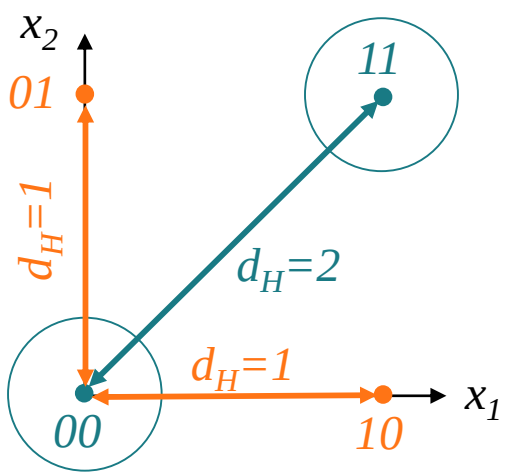
u	x_1x_2	Hamming weight w_H
0	00	0
1	11	2

Minimum Hamming distance $d_{min}=2$

Decoding spheres: Spheres with radius

$$r=\lfloor (d_{\min}-1)/2 \rfloor = \lfloor (2-1)/2 \rfloor = 0$$

centered around codewords.



Definition:

The sphere $K_r(\mathbf{x})$ with radius r around word $\mathbf{x}$ is defined as the union of all words $\mathbf{y}$ with Hamming distance $d_H \le r$ to $\mathbf{x}$:

$$K_r(\mathbf{x}) = \{\mathbf{y} \mid d_H(\mathbf{x},\mathbf{y}) \le r\}$$

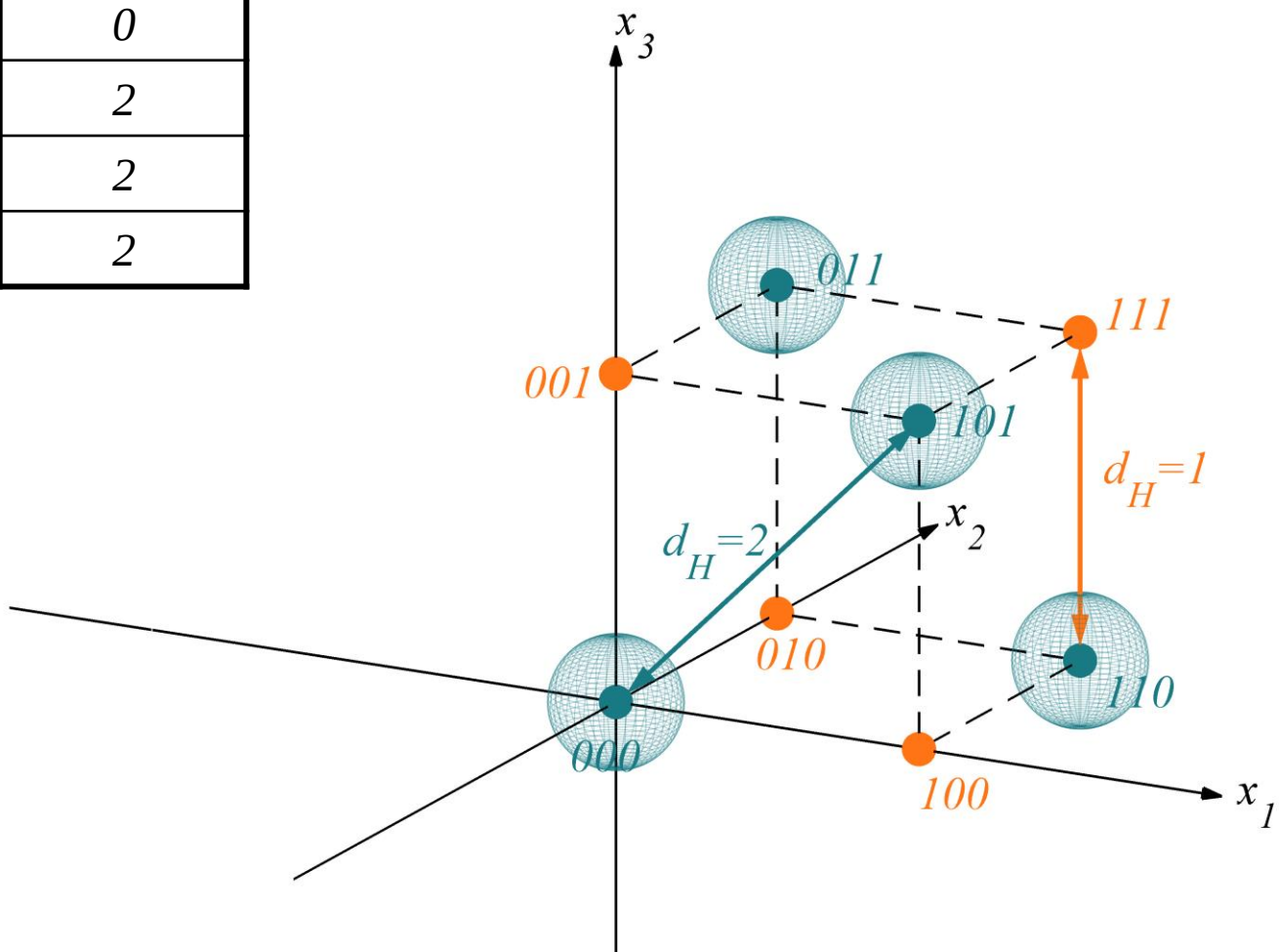
No perfect code.

Decoding Spheres

Example: Rate $R=2/3$ single parity check code

u	$x_1x_2x_3$	Hamming weight w_H
00	000	0
01	011	2
10	101	2
11	110	2

Minimum Hamming distance $d_{min}=2$



Perfect Codes

Perfect code: All possible received words (hard decision) are inside a decoding sphere.

Linear Codes

Binary codes:

A (N,K)-code $\mathcal{C}$ is *linear*, if the sum of two codewords is also a codeword:

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C} \;\Rightarrow\; \mathbf{x}_1 \oplus \mathbf{x}_2 \in \mathcal{C}$$

Non-binary (q-ary) codes:

A $(N,K)_q$-code $\mathcal{C}$ is *linear*, if the sum of two codewords over Galois Field $\mathbb{F}_q$ is also a codeword:

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C} \;\Rightarrow\; \mathbf{x}_1 + \mathbf{x}_2 \bmod q \in \mathcal{C} \quad \text{and}$$

$$\mathbf{x} \in \mathcal{C}, \alpha \in \mathbb{F}_q \;\Rightarrow\; \alpha\mathbf{x}_1 \in \mathcal{C}$$

Linear Codes

A (N,K)-code $\mathcal{C}$ is *linear*, if the sum of two codewords is also a codeword:

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C} \;\Rightarrow\; \mathbf{x}_1 \oplus \mathbf{x}_2 \in \mathcal{C}$$

Examples:

	code $\mathcal{C}_1$	code $\mathcal{C}_2$	code $\mathcal{C}_3$
$\mathbf{x}_1$	*000*	*000*	*000*
$\mathbf{x}_2$	*100*	*110*	*110*
$\mathbf{x}_3$	*010*	*011*	*101*
$\mathbf{x}_4$	*001*	*111*	*011*
	non-linear	non-linear	linear

Code $\mathcal{C}_1$:

$\mathbf{x}_2 \oplus \mathbf{x}_3 = 110 \notin \mathcal{C}_1 \Rightarrow$ non-linear code

Code $\mathcal{C}_2$:

$\mathbf{x}_2 \oplus \mathbf{x}_3 = 101 \notin \mathcal{C}_2 \Rightarrow$ non-linear code

Code $\mathcal{C}_3$:

$\mathbf{x}_1 \oplus \mathbf{x}_1 = \mathbf{x}_1$ $\quad$ $\mathbf{x}_2 \oplus \mathbf{x}_2 = \mathbf{x}_1$ $\quad$ $\mathbf{x}_3 \oplus \mathbf{x}_3 = \mathbf{x}_1$

$\mathbf{x}_1 \oplus \mathbf{x}_2 = \mathbf{x}_2$ $\quad$ $\mathbf{x}_2 \oplus \mathbf{x}_3 = \mathbf{x}_4$ $\quad$ $\mathbf{x}_3 \oplus \mathbf{x}_4 = \mathbf{x}_2$

$\mathbf{x}_1 \oplus \mathbf{x}_3 = \mathbf{x}_3$ $\quad$ $\mathbf{x}_2 \oplus \mathbf{x}_4 = \mathbf{x}_3$ $\quad$ $\mathbf{x}_4 \oplus \mathbf{x}_4 = \mathbf{x}_1$

$\mathbf{x}_1 \oplus \mathbf{x}_4 = \mathbf{x}_4$ $\quad$ $\Rightarrow$ linear code

Minimum Hamming Distance of Linear Codes

The minimum Hamming distance d_{min} of a linear code $\mathcal{C}$ is given by the minimum Hamming weight w_{min} of the code:

$$d_{min} = \min \{d_H(\mathbf{x}_1,\mathbf{x}_2) \mid \mathbf{x}_1,\mathbf{x}_2 \in \mathcal{C}, \mathbf{x}_1 \neq \mathbf{x}_2\}$$

$$= \min \{w_H(\mathbf{x}) \mid \mathbf{x} \in \mathcal{C}, \mathbf{x} \neq \mathbf{0}\}$$

Proof:

$d_H(\mathbf{x}_1,\mathbf{x}_2) = w_H(\mathbf{x}_1\text{-}\mathbf{x}_2)$

Linear code: $\mathbf{x}_1\text{-}\mathbf{x}_2 \in \mathcal{C}$

$\Rightarrow$ Minimum Hamming distance is equal to minimum Hamming weight

Minimum Hamming Distance of Linear Codes – Example: (7,4,3) Hamming Code

$\mathbf{x}_k$	w_H
$\mathbf{x}_1=0000000$	0
$\mathbf{x}_2=0001111$	4
$\mathbf{x}_3=0010110$	3
$\mathbf{x}_4=0011001$	3
$\mathbf{x}_5=0100101$	3
$\mathbf{x}_6=0101010$	3
$\mathbf{x}_7=0110011$	4
$\mathbf{x}_8=0111100$	4
$\mathbf{x}_9=1000011$	3
$\mathbf{x}_{10}=1001100$	3
$\mathbf{x}_{11}=1010101$	4
$\mathbf{x}_{12}=1011010$	4
$\mathbf{x}_{13}=1100110$	4
$\mathbf{x}_{14}=1101001$	4
$\mathbf{x}_{15}=1110000$	3
$\mathbf{x}_{16}=1111111$	7

Example:

$$d_H(\mathbf{x}_2,\mathbf{x}_3) = w_H(\mathbf{x}_2-\mathbf{x}_3) = w_H(\mathbf{x}_2 \oplus \mathbf{x}_3)$$

$\mathbf{x}_2 \oplus \mathbf{x}_3$:

$$\begin{array}{c} 0001111 \\ \oplus \\ 0010110 \\ = \\ 0011001 \end{array}$$

$$w_H(\mathbf{x}_2 \oplus \mathbf{x}_3) = d_H(\mathbf{x}_2,\mathbf{x}_3) = 3$$

Minimum Hamming Distance of Linear Codes – Example: (7,4,3) Hamming Code

$\mathbf{x}_k$	w_H
$\mathbf{x}_1=0000000$	0
$\mathbf{x}_2=0001111$	4
$\mathbf{x}_3=0010110$	3
$\mathbf{x}_4=0011001$	3
$\mathbf{x}_5=0100101$	3
$\mathbf{x}_6=0101010$	3
$\mathbf{x}_7=0110011$	4
$\mathbf{x}_8=0111100$	4
$\mathbf{x}_9=1000011$	3
$\mathbf{x}_{10}=1001100$	3
$\mathbf{x}_{11}=1010101$	4
$\mathbf{x}_{12}=1011010$	4
$\mathbf{x}_{13}=1100110$	4
$\mathbf{x}_{14}=1101001$	4
$\mathbf{x}_{15}=1110000$	3
$\mathbf{x}_{16}=1111111$	7

$d_H(\mathbf{x}_1,\mathbf{x}_2) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_3) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_4) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_5) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_6) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_7) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_8) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_9) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_{10}) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_{11}) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_{12}) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_{13}) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_{14}) = 4$
$d_H(\mathbf{x}_1,\mathbf{x}_{15}) = 3$
$d_H(\mathbf{x}_1,\mathbf{x}_{16}) = 7$

$d_H(\mathbf{x}_2,\mathbf{x}_3) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_4) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_5) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_6) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_7) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_8) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_9) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_{10}) = 3$
$d_H(\mathbf{x}_2,\mathbf{x}_{11}) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_{12}) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_{13}) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_{14}) = 4$
$d_H(\mathbf{x}_2,\mathbf{x}_{15}) = 7$
$d_H(\mathbf{x}_2,\mathbf{x}_{16}) = 3$

$d_H(\mathbf{x}_3,\mathbf{x}_4) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_5) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_6) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_7) = 3$
$d_H(\mathbf{x}_3,\mathbf{x}_8) = 3$
$d_H(\mathbf{x}_3,\mathbf{x}_9) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_{10}) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_{11}) = 3$
$d_H(\mathbf{x}_3,\mathbf{x}_{12}) = 3$
$d_H(\mathbf{x}_3,\mathbf{x}_{13}) = 3$
$d_H(\mathbf{x}_3,\mathbf{x}_{14}) = 7$
$d_H(\mathbf{x}_3,\mathbf{x}_{15}) = 4$
$d_H(\mathbf{x}_3,\mathbf{x}_{16}) = 4$

$d_H(\mathbf{x}_4,\mathbf{x}_5) = 4$
$d_H(\mathbf{x}_4,\mathbf{x}_6) = 4$
$d_H(\mathbf{x}_4,\mathbf{x}_7) = 3$
$d_H(\mathbf{x}_4,\mathbf{x}_8) = 3$
$d_H(\mathbf{x}_4,\mathbf{x}_9) = 4$
$d_H(\mathbf{x}_4,\mathbf{x}_{10}) = 4$
$d_H(\mathbf{x}_4,\mathbf{x}_{11}) = 3$
$d_H(\mathbf{x}_4,\mathbf{x}_{12}) = 3$
$d_H(\mathbf{x}_4,\mathbf{x}_{13}) = 7$
$d_H(\mathbf{x}_4,\mathbf{x}_{14}) = 3$
$d_H(\mathbf{x}_4,\mathbf{x}_{15}) = 4$
$d_H(\mathbf{x}_4,\mathbf{x}_{16}) = 4$

...

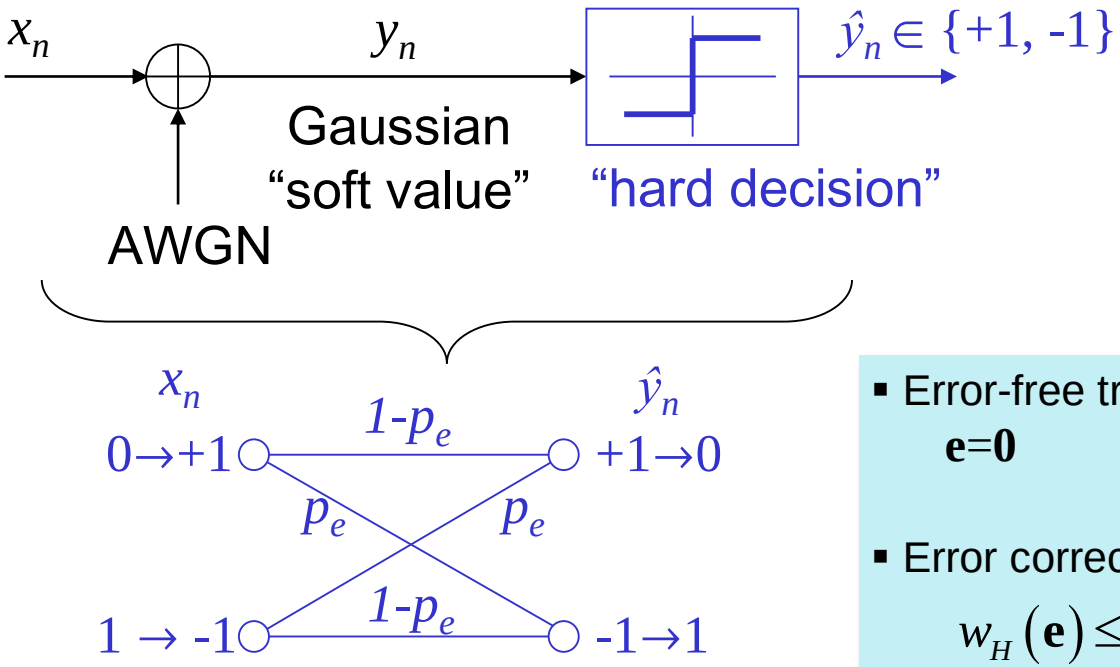
Find d_{min}: $q^K(q^K-1)/2$ comparisons (all pairs of codewords)

Find w_{min}: q^K-1 comparisons (all codewords except $\mathbf{x}_1=\mathbf{0}$)

$$d_{min}=w_{min}=3$$

In order to determine the minimum Hamming distance of a linear code, it is not necessary to compare the pairwise Hamming distances of all $q^K(q^K-1)$ pairs of codewords. It is sufficient to compare the Hamming weights of all codewords which requires only q^K-1 comparisons (The code has q^K codewords, but the all zero codeword **x**=**0** has not to be taken into account).

Error Detection and Error Correction



$\hat{\mathbf{y}} = \mathbf{x} + \mathbf{e}$ $\mathbf{e}$: error word

- Error-free transmission:
  $\mathbf{e}=\mathbf{0}$
- Error correction:
  $$w_H(\mathbf{e}) \le \lfloor (d_{\min} - 1)/2 \rfloor$$
- Error detection:
  - $\mathbf{e} \notin \mathcal{C}$ (received word $\hat{\mathbf{y}}$ is not a codeword)
  - Guaranteed error detection:
    $$w_H(\mathbf{e}) \le d_{\min} - 1$$
- Undetected errors:
  $\mathbf{e} \in \mathcal{C} \setminus \{\mathbf{0}\}$
  (received word $\hat{\mathbf{y}}$ is a wrong codeword)

Error Detection and Error Correction

An $(N, K, d_{min})_q$-code corrects $\lfloor (d_{\min} - 1)/2 \rfloor$ errors.

An $(N, K, d_{min})_q$-code detects d_{min} -1 errors.

d_{min}: Minimum Hamming distance of the code.

MAP and ML Decoding

INSTITUT FÜR NACHRICHTENTECHNIK

$\mathbf{u}=[u_1, \ldots, u_K]^T$ → encoder → $\mathbf{x}=[x_1, \ldots, x_N]^T$ → channel → $\mathbf{y}=[y_1, \ldots, y_N]^T$ → decoder → $\hat{\mathbf{u}}=[\hat{u}_1, \ldots, \hat{u}_K]^T$

Target: Minimization of word error rate:

Maximum A-Posteriori Probability (MAP) Decoding:

$$\hat{\mathbf{u}} = \arg\max P\left(\mathbf{u} = \hat{\mathbf{u}} \mid \mathbf{y}\right)$$

Bayes rule: $P\left(\mathbf{u} = \hat{\mathbf{u}} \mid \mathbf{y}\right) P\left(\mathbf{y}\right) = P\left(\mathbf{y} \mid \mathbf{u} = \hat{\mathbf{u}}\right) P\left(\mathbf{u} = \hat{\mathbf{u}}\right) \Leftrightarrow P\left(\mathbf{u} = \hat{\mathbf{u}} \mid \mathbf{y}\right) = \dfrac{P\left(\mathbf{y} \mid \mathbf{u} = \hat{\mathbf{u}}\right) P\left(\mathbf{u} = \hat{\mathbf{u}}\right)}{P\left(\mathbf{y}\right)}$

$P(\mathbf{y})$ is the same for all candidates $\hat{\mathbf{u}}$

For equally likely info words $\mathbf{u}$: $P\left(\mathbf{u} = \hat{\mathbf{u}}\right) = \dfrac{1}{q^K} = \text{const}$

Maximum Likelihood (ML) Decoding:

$$\hat{\mathbf{u}} = \arg\max P\left(\mathbf{y} \mid \mathbf{u} = \hat{\mathbf{u}}\right)$$

For equally likely info words, MAP and ML decoding are equivalent.

MAP and ML Decoding

$\mathbf{u}=[u_1, \ldots, u_K]^T \longrightarrow$ encoder $\xrightarrow{\mathbf{x}=[x_1, \ldots, x_N]^T}$ channel $\xrightarrow{\mathbf{y}=[y_1, \ldots, y_N]^T}$ decoder $\longrightarrow \hat{\mathbf{u}}=[\hat{u}_1, \ldots, \hat{u}_K]^T$

Transition probability: $P(\mathbf{y} \mid \mathbf{x})$

Memoryless channels: $P(\mathbf{y} \mid \mathbf{x}) = \prod_{n=1}^{N} P(y_n \mid x_n)$

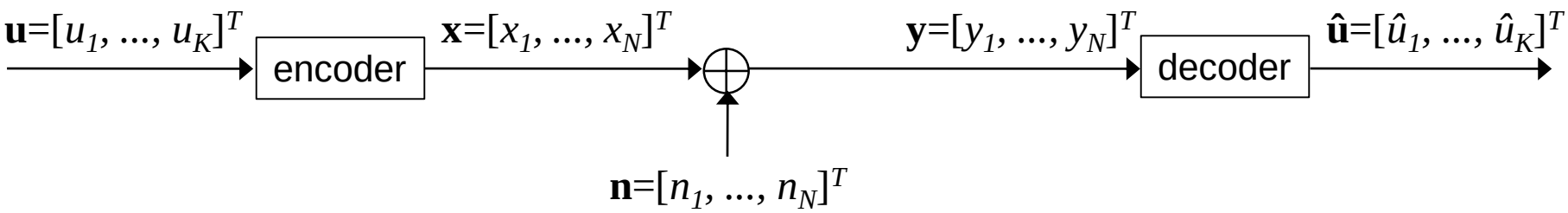
Maximum A-Posteriori Probability Decoding:

$$\begin{aligned}
\hat{\mathbf{u}} &= \arg\max P(\mathbf{u} = \hat{\mathbf{u}} \mid \mathbf{y}) \\
&= \arg\max_{\mathbf{u}\to\mathbf{x}} P(\mathbf{x} \mid \mathbf{y}) \\
&= \arg\max_{\mathbf{u}\to\mathbf{x}} P(\mathbf{y} \mid \mathbf{x}) P(\mathbf{u} = \hat{\mathbf{u}}) \\
&= \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} P(y_i \mid x_i) \prod_{k=1}^{K} P(u_k) \\
&= \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} P(y_i \mid x_i)
\end{aligned}$$

$\longleftarrow$ for memoryless channels and statistically independent info symbols

$\longleftarrow$ for equally likely info symbols (equivalent to ML decoding)

MAP and ML Decoding in AWGN Channel



AWGN $f_N(n) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{n^2}{2\sigma_N^2}}$

MAP Decoding:

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} P(y_i \mid x_i) \prod_{k=1}^{K} P(u_k)$$

$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(y_i - x_i)^2}{2\sigma_N^2}} \prod_{k=1}^{K} P(u_k)$$

$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ \underbrace{-N \log\left(\sqrt{2\pi}\sigma_N\right)}_{\text{const}} - \sum_{i=1}^{N} \frac{(y_i - x_i)^2}{2\sigma_N^2} + \sum_{k=1}^{K} \log P(u_k) \right\}$$

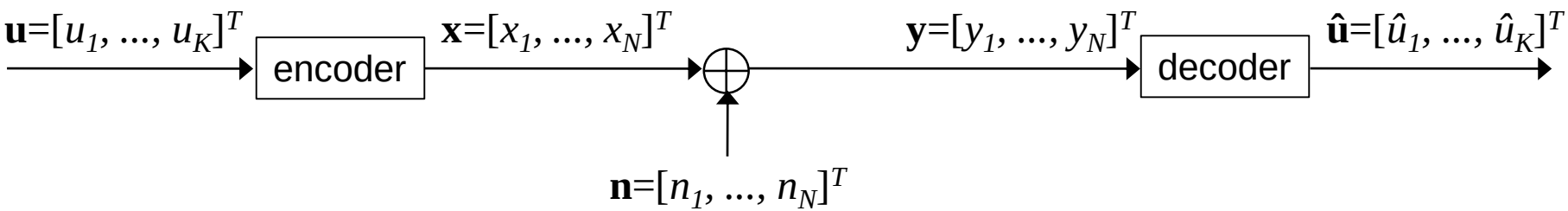
$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ -\frac{1}{2\sigma_N^2} \sum_{i=1}^{N} (y_i - x_i)^2 + \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$= \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \sum_{i=1}^{N} (y_i - x_i)^2 \right\} \longleftarrow$$ for equally likely info symbols (equivalent to ML decoding)

σ_N^2 is required for MAP decoding but not for ML decoding.

ML decoding rule: Choose codeword $\mathbf{x}$ with minimum squared Euclidean distance to received word $\mathbf{y}$.

MAP and ML Decoding of Binary Codes in AWGN Channel



AWGN $f_N(n) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{n^2}{2\sigma_N^2}}$

MAP Decoding:

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ -\frac{1}{2\sigma_N^2} \sum_{i=1}^{N} (y_i - x_i)^2 + \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ -\frac{1}{2\sigma_N^2} \sum_{i=1}^{N} \left(\overset{\text{const}}{\cancel{y_i^2}} - 2x_i y_i + \overset{\text{const}}{\cancel{x_i^2}} \right) + \sum_{k=1}^{K} \log P(u_k) \right\} \leftarrow$$

$x_i^2=1$ for binary code symbols $x_i=\pm 1$

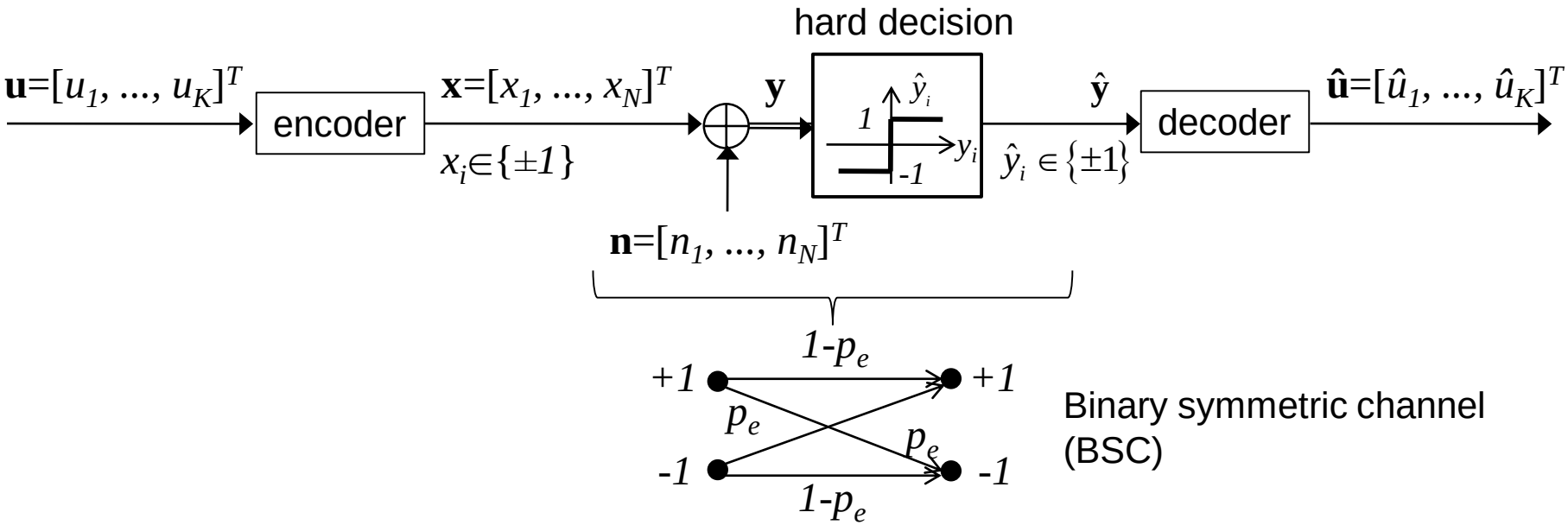
$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{\sigma_N^2} \sum_{i=1}^{N} x_i y_i + \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$= \arg\max_{\mathbf{u}\to\mathbf{x}} \left\{ \sum_{i=1}^{N} x_i y_i \right\} \leftarrow$$

for equally likely info symbols (equivalent to ML decoding)

ML decoding rule:
Choose codeword $\mathbf{x}$ with maximum correlation with the received word $\mathbf{y}$.

MAP and ML Decoding in BSC Channel



Binary symmetric channel (BSC)

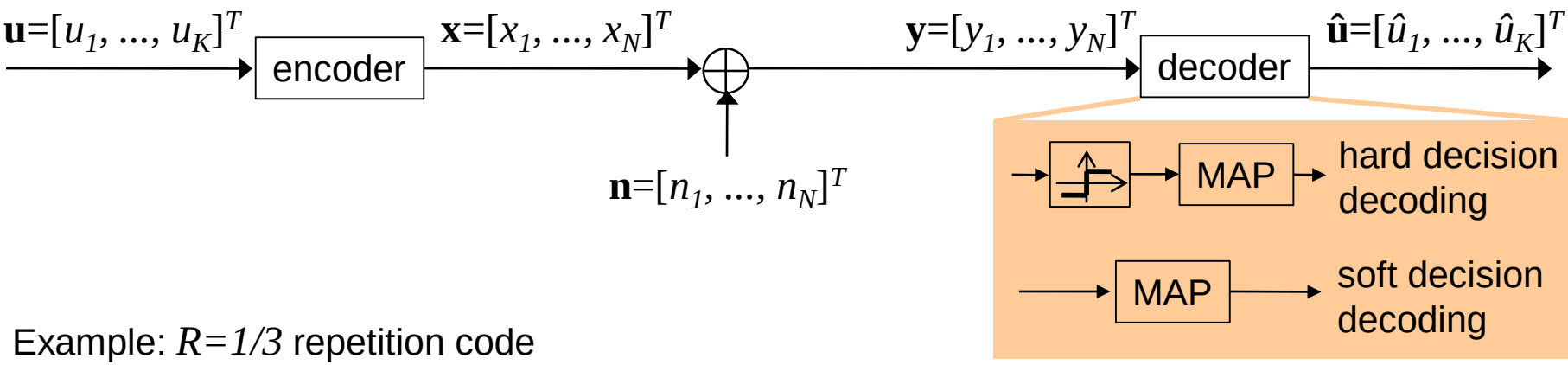
MAP Decoding:

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} P\left(\hat{y}_i \mid x_i\right) \prod_{k=1}^{K} P\left(u_k\right)$$

$$P\left(\hat{y}_i = +1 \mid x_i = +1\right) = P\left(\hat{y}_i = -1 \mid x_i = -1\right) = 1 - p_e$$

$$P\left(\hat{y}_i = -1 \mid x_i = +1\right) = P\left(\hat{y}_i = +1 \mid x_i = -1\right) = p_e$$

Hard Decision vs. Soft Decision Decoding



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$ (soft decoder input)	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]
$\hat{\mathbf{y}}$ (hard decoder input)	$[1,-1,1]$	$[1,1,-1]$	$[1,1,-1]$

A-priori info (or assumption): $P_a(u_i=+1) = P_a(u_i=-1) = 1/2$ (MAP = ML decoding):

$\hat{\mathbf{u}}$ hard decision decoding	$+1$	$+1$ ↯	$+1$ ↯
$\hat{\mathbf{u}}$ soft decision decoding	$+1$	-1	$+1$ ↯

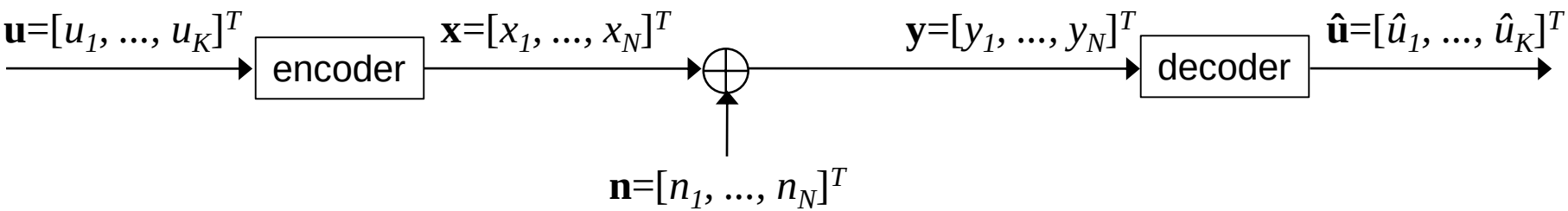
soft is better than hard

A-priori info: $P_a(u_i=+1) = 1/4$, $P_a(u_i=-1) = 3/4$, SNR<5dB (MAP decoding):

$\hat{\mathbf{u}}$ soft decision decoding	$+1$	-1	-1

MAP is better than ML

Hard Decision vs. Soft Decision Decoding
Example: Hard Decision MAP Decoding



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]
$\hat{\mathbf{y}}$	$[1,-1,1]$	$[1,1,-1]$	$[1,1,-1]$

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

(hard decision MAP decoding)

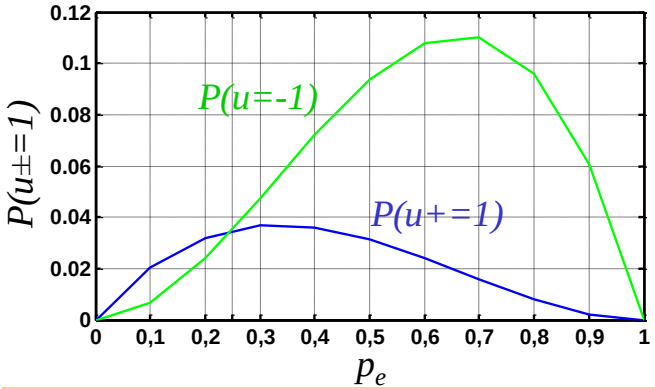
$$\hat{u}=\arg\max P(\hat{\mathbf{y}}\,|\,u=\pm1)\,P_a(u=\pm1)=\arg\max\prod_{i=1}^{N}P(\hat{y}_i\,|\,x_i=\pm1)P_a(u=\pm1)$$

$$P(\hat{y}_i=+1\,|\,x_i=+1)=P(\hat{y}_i=-1\,|\,x_i=-1)=1-p_e$$
$$P(\hat{y}_i=-1\,|\,x_i=+1)=P(\hat{y}_i=+1\,|\,x_i=-1)=p_e$$

For u_1, u_2, u_3:

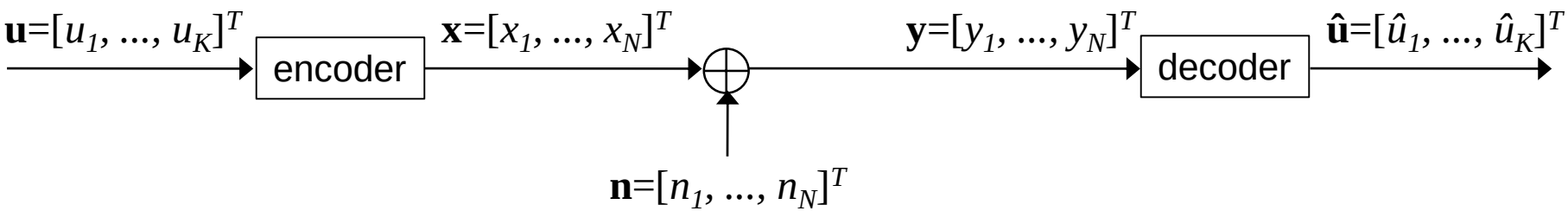
$$P(u=+1)=(1-p_e)\,p_e\,(1-p_e)\frac{1}{4}=\frac{1}{4}\left(p_e-2p_e^2+p_e^3\right)$$

$$P(u=-1)=p_e\,(1-p_e)\,p_e\,\frac{3}{4}=\frac{3}{4}\left(p_e^2-p_e^3\right)$$



MAP decision depends on p_e:
$\hat{u}=+1$ for $p_e<0.25$; $\hat{u}=-1$ for $p_e>0.25$

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision ML Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

Assumption:

$P_a(u_i=+1)=P_a(u_i=-1)=1/2$

(ML decoding)

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \sum_{i=1}^{N} (y_i - x_i)^2 \right\}$$

For $\hat{u}_1=+1$: $\sum_{i=1}^{N}(y_i-1)^2 = (0.3-1)^2 + (-0.1-1)^2 + (2-1)^2$

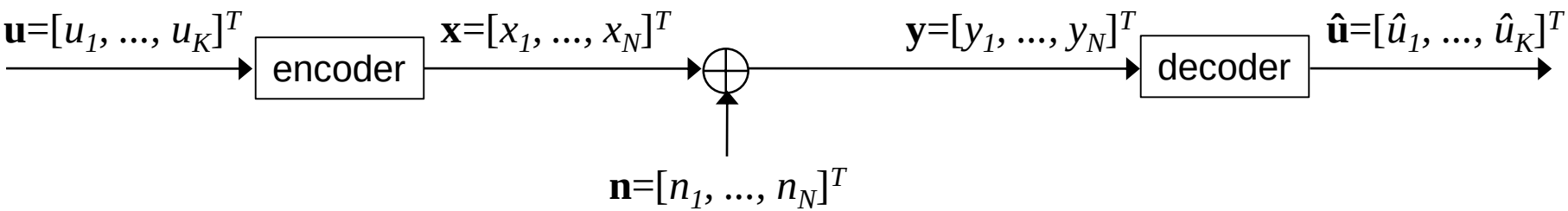
$$= (-0.7)^2 + (-1.1)^2 + (1)^2 = 2.7$$

For $\hat{u}_1=-1$: $\sum_{i=1}^{N}(y_i+1)^2 = (0.3+1)^2 + (-0.1+1)^2 + (2+1)^2$

$$= (1.3)^2 + (0.9)^2 + (3)^2 = 11.5$$

$\Rightarrow \hat{u}_1=+1$ no error

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision ML Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

Assumption:

$P_a(u_i=+1) = P_a(u_i=-1) = 1/2$

(ML decoding)

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \sum_{i=1}^{N} (y_i - x_i)^2 \right\}$$

For $\hat{u}_2=+1$: $\sum_{i=1}^{N} (y_i - 1)^2 = (0.1-1)^2 + (0.2-1)^2 + (-2-1)^2$

$$= (-0.9)^2 + (-0.8)^2 + (-3)^2 = 10.45$$

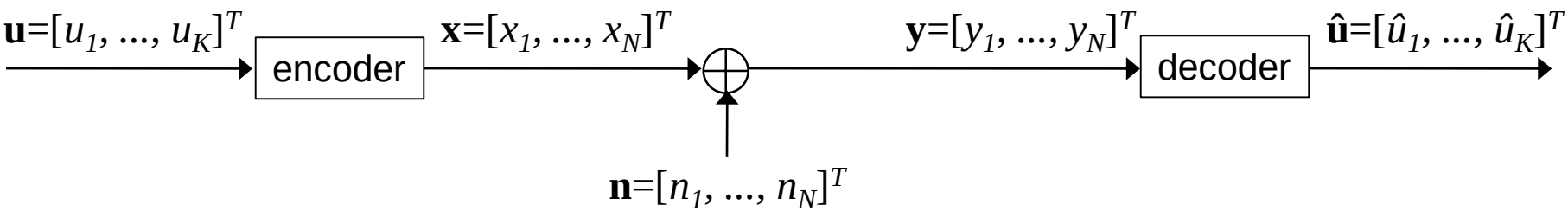
For $\hat{u}_2=-1$: $\sum_{i=1}^{N} (y_i + 1)^2 = (0.1+1)^2 + (0.2+1)^2 + (-2+1)^2$

$$= (1.1)^2 + (1.2)^2 + (-1)^2 = 3.65$$

$\Rightarrow \hat{u}_2=-1$ no error

Two (hard decision) errors can be corrected using soft decision decoding !

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision ML Decoding



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

Assumption:

$P_a(u_i=+1)=P_a(u_i=-1)=1/2$

(ML decoding)

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \sum_{i=1}^{N} (y_i - x_i)^2 \right\}$$

For $\hat{u}_3=+1$: $\sum_{i=1}^{N}(y_i-1)^2 = (0.1-1)^2 + (0.2-1)^2 + (-0.2-1)^2$

$$= (-0.9)^2 + (-0.8)^2 + (-1.2)^2 = 2.89$$

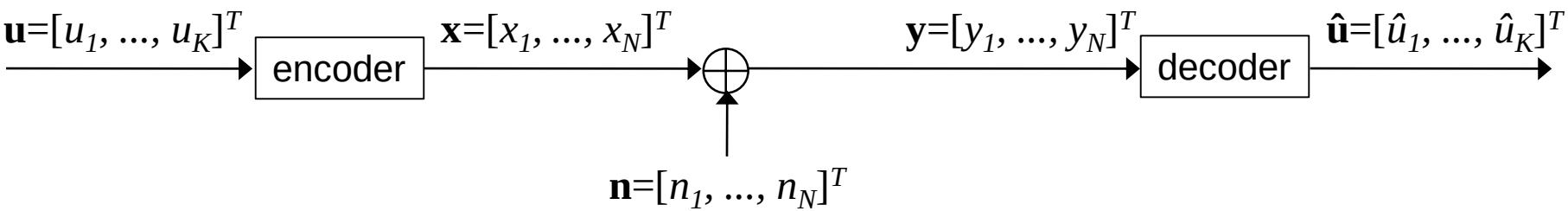
For $\hat{u}_3=-1$: $\sum_{i=1}^{N}(y_i+1)^2 = (0.1+1)^2 + (0.2+1)^2 + (-0.2+1)^2$

$$= (1.1)^2 + (1.2)^2 + (0.8)^2 = 3.29$$

$\Rightarrow \hat{u}_3=+1$ ↯ decoding error

Error correction not possible due to low reliability of received samples.

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

A-priori info:

$P_a(u_i=+1) = 1/4$, $P_a(u_i=-1) = 3/4$

SNR=100 dB

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

For $\hat{u}_3=+1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-10}}\left((0.1-1)^2+(0.2-1)^2+(-0.2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-10}}\cdot 2.89 + 1.3863 = 1.445\cdot 10^{10} + 1.3863$$

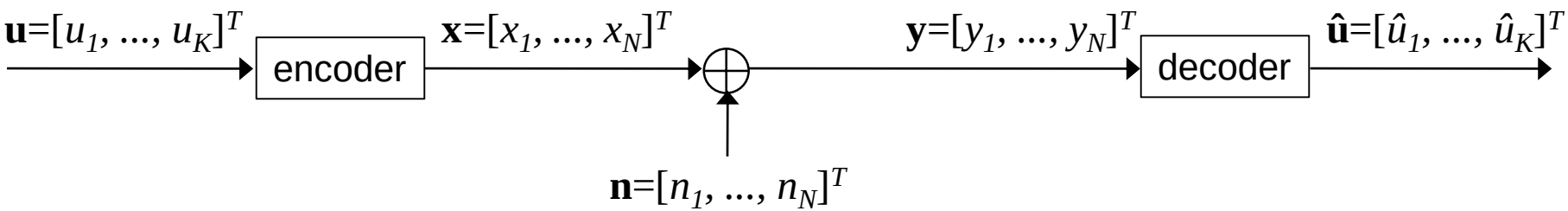
For $\hat{u}_3=-1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-10}}\left((0.1+1)^2+(0.2+1)^2+(-0.2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-10}}\cdot 3.29 + 0.2877 = 1.645\cdot 10^{10} + 0.2877$$

$\Rightarrow \hat{u}_3=+1$ ↯ decoding error

Decoder relies almost only on received samples due to high SNR (almost ML decoding).

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=0 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_3=+1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2}\left((0.1-1)^2+(0.2-1)^2+(-0.2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2}\cdot 2.89 + 1.3863 = 1.445 + 1.3863 = 2.8313$$

For $\hat{u}_3=-1$:

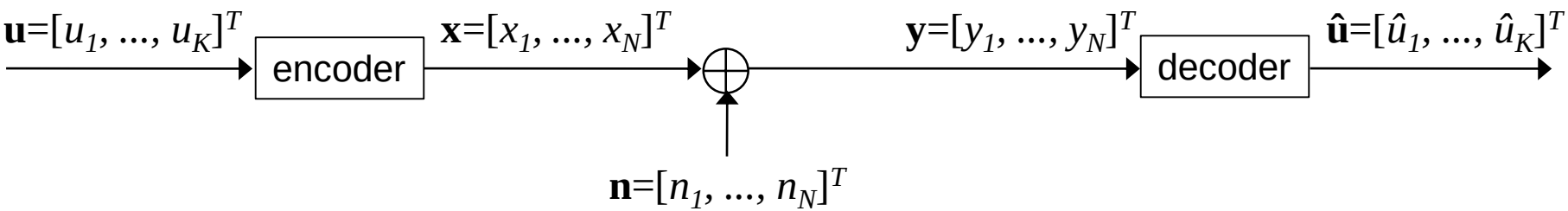
$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2}\left((0.1+1)^2+(0.2+1)^2+(-0.2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2}\cdot 3.29 + 0.2877 = 1.645 + 0.2877 = 1.9327$$

$\Rightarrow \hat{u}_3=-1$ no error

MAP decoder corrects error due to high impact of a-priori info (low SNR).

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=0 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_1=+1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2}\left((0.3-1)^2+(-0.1-1)^2+(2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2}\cdot 2.7 + 1.3863 = 2.7363$$

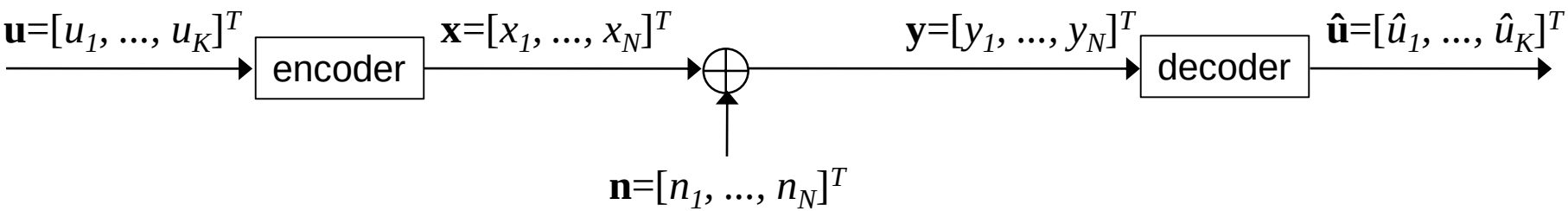
For $\hat{u}_1=-1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2}\left((0.3+1)^2+(-0.1+1)^2+(2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2}\cdot 11.5 + 0.2877 = 6.0377$$

$\Rightarrow \hat{u}_1=+1$ no error

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=0 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_2=+1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2}\left((0.1-1)^2+(0.2-1)^2+(-2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2}\cdot 10.45 + 1.3863 = 6.6113$$

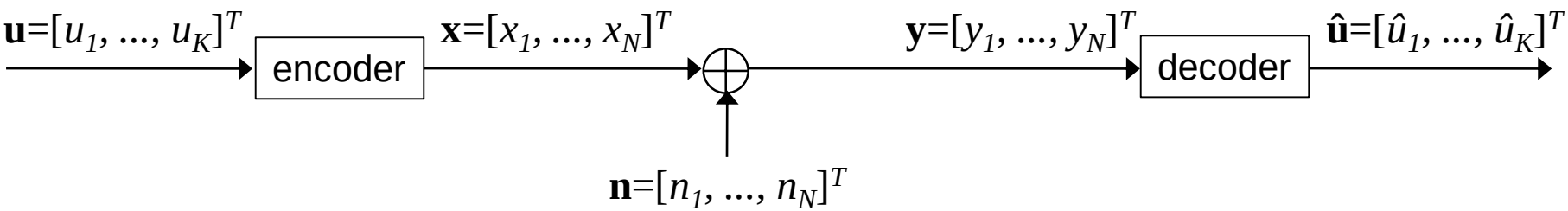
For $\hat{u}_2=-1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2}\left((0.1+1)^2+(0.2+1)^2+(-2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2}\cdot 3.65 + 0.2877 = 2.1127$$

$\Rightarrow \hat{u}_2=-1$ error corrected

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=10 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_1=+1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1}}\left((0.3-1)^2+(-0.1-1)^2+(2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1}}\cdot 2.7 + 1.3863 = 14.8863$$

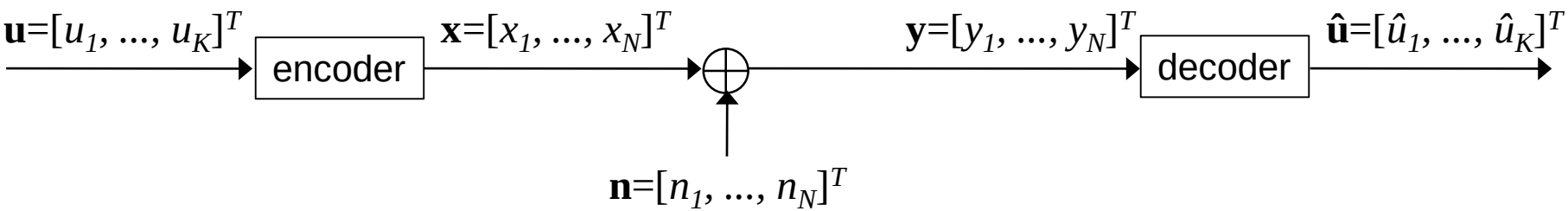
For $\hat{u}_1=-1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1}}\left((0.3+1)^2+(-0.1+1)^2+(2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1}}\cdot 11.5 + 0.2877 = 77.7877$$

$\Rightarrow \hat{u}_1=+1$ no error

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=10 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_2=+1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1}}\left((0.1-1)^2+(0.2-1)^2+(-2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1}}\cdot 10.45 + 1.3863 = 53.6363$$

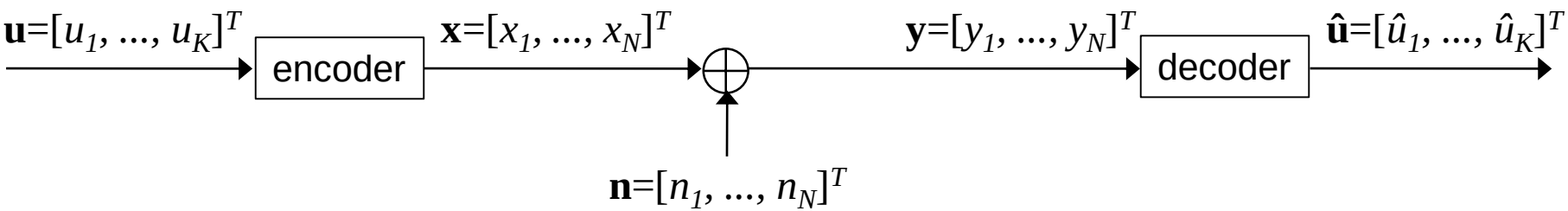
For $\hat{u}_2=-1$:

$$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1}}\left((0.1+1)^2+(0.2+1)^2+(-2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1}}\cdot 3.65 + 0.2877 = 18.5377$$

$\Rightarrow \hat{u}_2=-1$ error corrected

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=10 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_3=+1$: $\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1}}\left((0.1-1)^2+(0.2-1)^2+(-0.2-1)^2\right) - \log\left(\frac{1}{4}\right)$

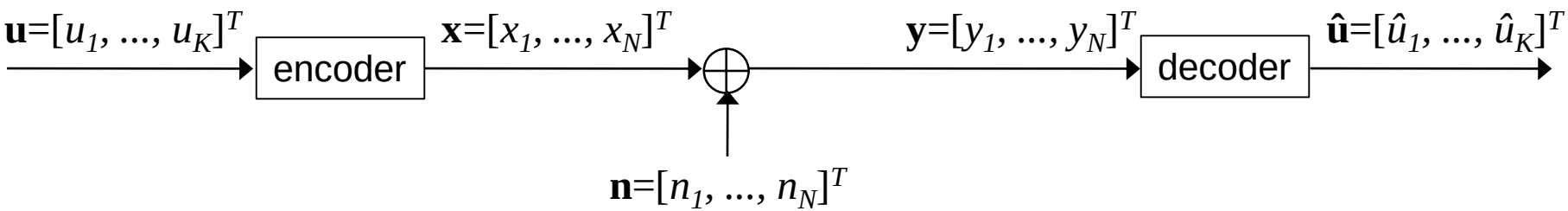
$$= \frac{1}{2\cdot 10^{-1}}\cdot 2.89 + 1.3863 = 15.8363$$

For $\hat{u}_3=-1$: $\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1}}\left((0.1+1)^2+(0.2+1)^2+(-0.2+1)^2\right) - \log\left(\frac{3}{4}\right)$

$$= \frac{1}{2\cdot 10^{-1}}\cdot 3.29 + 0.2877 = 16.7377$$

$\Rightarrow \hat{u}_3=+1$ ↯ error

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=5 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_1=+1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.3-1)^2+(-0.1-1)^2+(2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

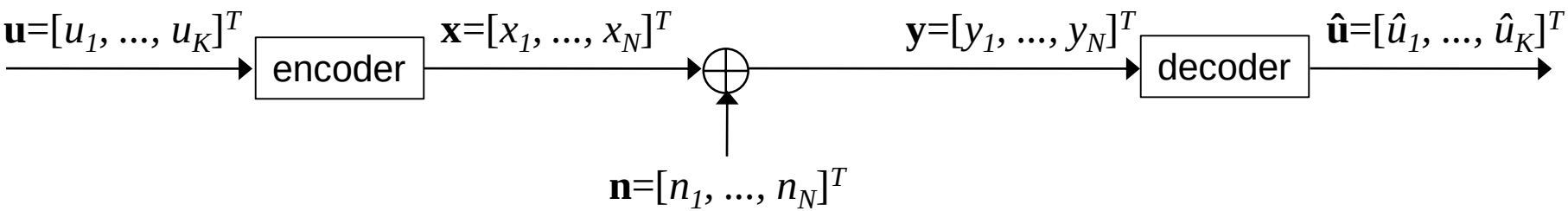
$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 2.7 + 1.3863 = 5.6554$$

For $\hat{u}_1=-1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.3+1)^2+(-0.1+1)^2+(2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 11.5 + 0.2877 = 18.4708$$

$\Rightarrow \hat{u}_1=+1$ no error

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

u	$+1$	-1	-1
x	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
y	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=5 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_2=+1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.1-1)^2+(0.2-1)^2+(-2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

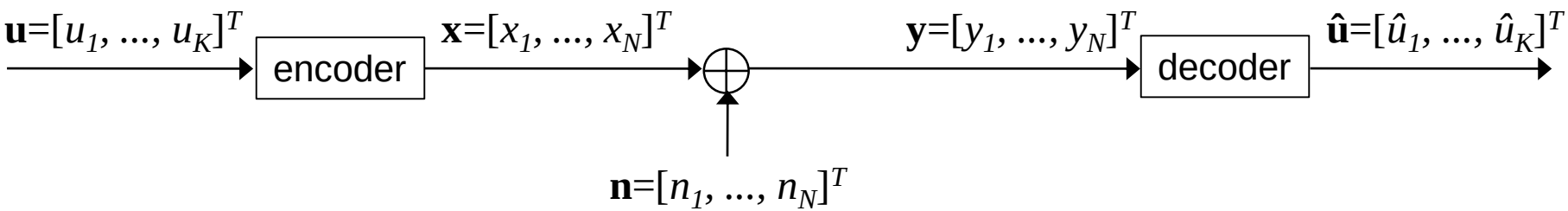
$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 10.45 + 1.3863 = 17.9092$$

For $\hat{u}_2=-1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.1+1)^2+(0.2+1)^2+(-2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 3.65 + 0.2877 = 6.0588$$

$\Rightarrow \hat{u}_2=-1$ error corrected

Hard Decision vs. Soft Decision Decoding
Example: Soft Decision MAP Decoding



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]

A-priori info:

$P_a(u_i=+1)=1/4$, $P_a(u_i=-1)=3/4$

SNR=5 dB

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

$$\hat{u} = \arg\min_{\mathbf{u}\to\mathbf{x}} \left\{ \frac{1}{2\sigma_n^2} \sum_{i=1}^{N} (y_i - x_i)^2 - \sum_{k=1}^{K} \log P(u_k) \right\}$$

$$SNR = 10\log_{10}\frac{P_x}{P_n}\,dB = 10\log_{10}\frac{1}{\sigma_n^2}\,dB$$

$$\sigma_n^2 = 10^{-SNR/10}$$

For $\hat{u}_3=+1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i-1)^2 - \log P(u_k=+1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.1-1)^2+(0.2-1)^2+(-0.2-1)^2\right) - \log\left(\frac{1}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 2.89 + 1.3863 = 5.9558$$

For $\hat{u}_3=-1$: $$\frac{1}{2\sigma_n^2}\sum_{i=1}^{N}(y_i+1)^2 - \log P(u_k=-1) = \frac{1}{2\cdot 10^{-1/2}}\left((0.1+1)^2+(0.2+1)^2+(-0.2+1)^2\right) - \log\left(\frac{3}{4}\right)$$

$$= \frac{1}{2\cdot 10^{-1/2}}\cdot 3.29 + 0.2877 = 5.4896$$

$\Rightarrow \hat{u}_3=-1$ error corrected

MAP and ML Decoding

The decoder estimates the transmitted codeword or info word, respectively. Maximum a-posteriori probability (MAP) decoding is the optimum decoding rule for minimization of the word error probability (word error rate). Maximum Likelihood (ML) decoding is in general suboptimum. However, for equally likely info words, MAP and ML decoding are equivalent.

In case of unequal info word propabilities, MAP decoding requires knowledge about the statistics of the source, i.e. about the info word probabilities $P(\mathbf{u})$. This knowledge is called *a-priori information*. We indicate a-priori information using an index a as $P_a(\mathbf{u})$. If this knowledge is not available at the decoder, the decoder simply assumes that all info words are equally likely and, hence, performs ML decoding. However, this is suboptimum in case of unequal symbol probabilities.

Another advantage of ML decoding over MAP decoding is, that ML decoding does not require knowledge of the noise variance σ_n^2 (or channel error probability, respectively, in BSC channels). In the MAP decoding rule, the noise variance appears as a weighting factor between a-priori information and received information. For channels with high noise power (high error probability), the MAP decoder relies more on the a-priori information, whereas the decision in case of low noise variance (high signal to noise power ratio (SNR)) relies more on the received samples.

In MAP decoding, the noise variance has to be estimated at the receiver. This is usually done using a known training sequence, i.e. a periodically transmitted sequence of symbols which is known to the receiver. Estimation of the noise variance causes undesired additional decoder complexity and is often not very accurate (e.g. due to the limited length of the training sequence).

In AWGN channels, the ML decoding rule states, that the codeword with minimum Euclidean distance to the received word should be chosen by the decoder.

Hard vs. Soft Decision Decoding

In soft decision decoding, the receive filter outputs (matched filter outputs) are directly fed into the decoder. In an AWGN channel, those matched filter outputs are Gaussian distributed. The magnitude of the samples provides a reliability information. Samples with low magnitude (close to the decision threshold) are more likely to be erroneous than samples with large magnitude. This reliability information can be taken into account by the decoder. Samples with reliability information are called „soft values."

In hard decision decoding, a hard decision on the received samples (i.e. $y_i = \pm 1$) is done before feeding them to the decoder. Due to the hard decision, the reliability information is lost and cannot be taken into account by the decoder. This loss of information causes a performance degradation of hard decision decoding compared to soft decision decoding („soft is better than hard").

The hard decision basically transforms an AWGN channel into a binary symmetric channel (BSC). The loss of reliabilty information reflects the loss in terms of channel capacity of the BSC channel compared to the AWGN channel. Since in case of hard decision decoding, the decoder sees a BSC rather than an AWGN channel, the signal at the decoder input contains less information than in soft decision decoding.

On the other hand, hard decision decoding is less complex than soft decision decoding since the decoder needs to deal with binary values only (1 bit per binary symbol). In soft decision decoding, the decoder needs to process real valued symbols. In practical implementations, the soft values are quantized with a few bits (e.g. 3 bits per soft sample).

The reliability information in soft decision decoding sometimes allows to correct errors even for codes with a minimum Hamming distance $d_{\min} < 3$.

Log-Likelihood Ratios

The representation of soft information in the form of log-likelihood ratios is equivalent to the representation using probabilities. Log-likelihood ratios are often convenient to use. E.g. decoding of a repetition code can be done by adding the log-likelihood ratios of the channel information.

Log-Likelihood Ratios (L-Values)

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

$\mathrm{sign}\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information

$$P(u_k = +1) > P(u_k = -1) \Rightarrow L(u_k) > 0 \Rightarrow \mathrm{sign}\{L(u_k)\} = +1$$

$$P(u_k = +1) < P(u_k = -1) \Rightarrow L(u_k) < 0 \Rightarrow \mathrm{sign}\{L(u_k)\} = -1$$

Log-Likelihood Ratios (L-Values)

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

$\mathrm{sign}\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information

$$L(y_i \mid x_i) = \log \frac{P(y_i \mid x_i = +1)}{P(y_i \mid x_i = -1)}$$

Soft decoder output: $$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})}$$

Log-Likelihood Ratios (L-Values)

INSTITUT FÜR NACHRICHTENTECHNIK

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

$\mathrm{sign}\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)} = \log \frac{P(u_k = +1)}{1 - P(u_k = +1)}$$

$$\left(1 - P(u_k = +1)\right) e^{L(u_k)} = P(u_k = +1)$$

$$P(u_k = +1) = \frac{e^{L(u_k)}}{1 + e^{L(u_k)}}$$

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)} = \log \frac{1 - P(u_k = -1)}{P(u_k = -1)}$$

$$P(u_k = -1) e^{L(u_k)} = 1 - P(u_k = -1)$$

$$P(u_k = -1) = \frac{1}{1 + e^{L(u_k)}}$$

$$E\{u_k\} = (+1) P(u_k = +1) + (-1) P(u_k = -1) = \frac{e^{L(u_k)}}{1 + e^{L(u_k)}} - \frac{1}{1 + e^{L(u_k)}} = \frac{e^{L(u_k)} - 1}{1 + e^{L(u_k)}} \frac{e^{-L(u_k)/2}}{e^{-L(u_k)/2}}$$

$$= \frac{e^{L(u_k)/2} - e^{-L(u_k)/2}}{e^{L(u_k)/2} + e^{-L(u_k)/2}} = \tanh\left(\frac{L(u_k)}{2}\right)$$

„soft bit“

Log-Likelihood Ratios (L-Values)

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

sign$\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information

$P(u_k=+1)$	$P(u_k=-1) = 1-P(u_k=+1)$	$L(u_k)$
1	0	∞
0.5	0.5	0
0	1	$-\infty$
0.7311	0.2689	1
0.2689	0.7311	-1
0.9	0.1	2.1972
0.99	0.01	4.5951
0.9999	0.00001	11,5128

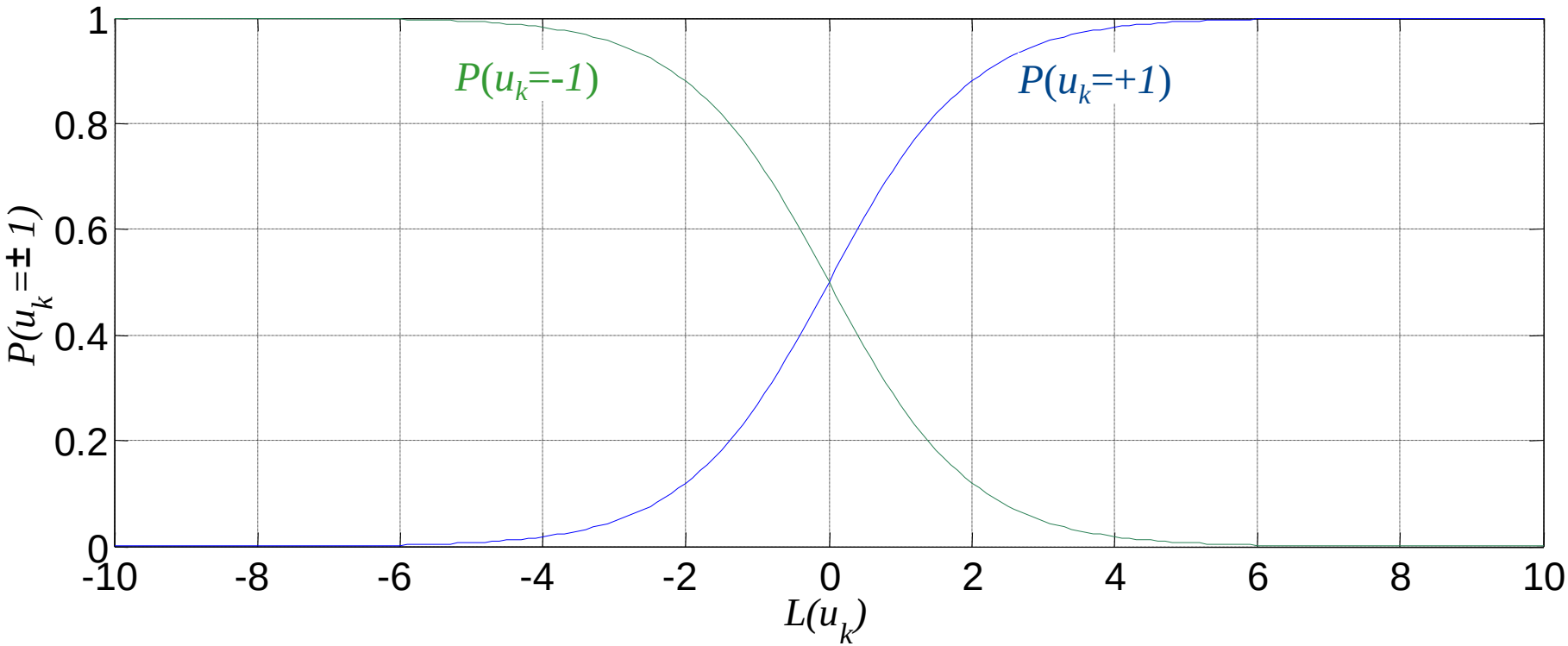
Log-Likelihood Ratios (L-Values)

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

$\mathrm{sign}\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information



Log-Likelihood Ratios (L-Values)

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

soft information:

$\mathrm{sign}\{L(u_k)\}$: hard decision

$|L(u_k)|$: reliability information

$$P(u_k = +1) = \frac{e^{L(u_k)}}{1 + e^{L(u_k)}} \qquad P(u_k = -1) = \frac{1}{1 + e^{L(u_k)}}$$

soft bit:

$$E\{u_k\} = \tanh\left(\frac{L(u_k)}{2}\right)$$

Log-Likelihood Ratios (L-Values): Boxplus Operation

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}$$

$$P(u_k = +1) = \frac{e^{L(u_k)}}{1 + e^{L(u_k)}}$$

$$P(u_k = -1) = \frac{1}{1 + e^{L(u_k)}}$$

$$\begin{aligned}
L(u_1 \oplus u_2) &= \log \frac{P(u_1 \oplus u_2 = +1)}{P(u_1 \oplus u_2 = -1)} \\
&= \log \frac{P(u_1 = +1)P(u_2 = +1) + P(u_1 = -1)P(u_2 = -1)}{P(u_1 = +1)P(u_2 = -1) + P(u_1 = -1)P(u_2 = +1)} \\
&= \log \frac{\frac{e^{L(u_1)}}{1+e^{L(u_1)}} \frac{e^{L(u_2)}}{1+e^{L(u_2)}} + \frac{1}{1+e^{L(u_1)}} \frac{1}{1+e^{L(u_2)}}}{\frac{e^{L(u_1)}}{1+e^{L(u_1)}} \frac{1}{1+e^{L(u_2)}} + \frac{1}{1+e^{L(u_1)}} \frac{e^{L(u_2)}}{1+e^{L(u_2)}}} \\
&= \log \frac{e^{L(u_1)} e^{L(u_2)} + 1}{e^{L(u_1)} + e^{L(u_2)}} \\
&= L(u_1) \boxplus L(u_2) \\
&\approx \operatorname{sign}\{L(u_1)\} \cdot \operatorname{sign}\{L(u_2)\} \cdot \min\{|L(u_1)|, |L(u_2)|\}
\end{aligned}$$

Log-Likelihood Ratios (L-Values): Boxplus Operation - Approximation

$$L(u_1) \boxplus L(u_2) = L(u_1 \oplus u_2) = \log \frac{e^{L(u_1)} e^{L(u_2)} + 1}{e^{L(u_1)} + e^{L(u_2)}}$$

$$\approx \operatorname{sign}\{L(u_1)\} \cdot \operatorname{sign}\{L(u_2)\} \cdot \min\{|L(u_1)|, |L(u_2)|\}$$

Example 1: $L(u_1) = L(u_2) = 1$

exact: $L(u_1) \boxplus L(u_2) = \log \frac{e^1 e^1 + 1}{e^1 + e^1} = \log \frac{e^2 + 1}{2e} = 0.4338$

approximation: $L(u_1) \boxplus L(u_2) \approx 1 \cdot 1 \cdot \min\{1,1\} = 1$

Example 2: $L(u_1) = 1,\ L(u_2) = 10$

exact: $L(u_1) \boxplus L(u_2) = \log \frac{e^1 e^{10} + 1}{e^1 + e^{10}} = 0.9999$

approximation: $L(u_1) \boxplus L(u_2) \approx 1 \cdot 1 \cdot \min\{1,10\} = 1$

The approximation holds the better, the more the magnitudes of $L(u_1)$ and $L(u_2)$ differ.

Log-Likelihood Ratios (L-Values): Boxplus Operation - Approximation

$$L(u_1) \boxplus L(u_2) = L(u_1 \oplus u_2) = \log \frac{e^{L(u_1)} e^{L(u_2)} + 1}{e^{L(u_1)} + e^{L(u_2)}}$$

$$\approx \operatorname{sign}\{L(u_1)\} \cdot \operatorname{sign}\{L(u_2)\} \cdot \min\{|L(u_1)|, |L(u_2)|\}$$

Example 3: $L(u_1) = L(u_2) = 0$

exact: $L(u_1) \boxplus L(u_2) = \log \frac{e^0 e^0 + 1}{e^0 + e^0} = \log \frac{1+1}{1+1} = 0$

approximation: $L(u_1) \boxplus L(u_2) \approx 0 \cdot 0 \cdot \min\{0, 0\} = 0$

Example 4: $L(u_1) = -0.1, L(u_2) = 1$

exact: $L(u_1) \boxplus L(u_2) = \log \frac{e^{-0.1} e^1 + 1}{e^{-0.1} + e^1} = -0.0462$

approximation: $L(u_1) \boxplus L(u_2) \approx -1 \cdot 1 \cdot \min\{0.1, 1\} = -0.1$

Log-Likelihood Ratios (L-Values): Boxplus Operation - Approximation

$$L(u_1) \boxplus L(u_2) = L(u_1 \oplus u_2) = \log \frac{e^{L(u_1)} e^{L(u_2)} + 1}{e^{L(u_1)} + e^{L(u_2)}}$$

$$\approx \operatorname{sign}\{L(u_1)\} \cdot \operatorname{sign}\{L(u_2)\} \cdot \min\{|L(u_1)|, |L(u_2)|\}$$

Example 5: $L(u_1) = 10, \quad L(u_2) = -20$

exact: $L(u_1) \boxplus L(u_2) = \log \frac{e^{10} e^{-20} + 1}{e^{10} + e^{-20}} = \log \frac{e^{-10} + 1}{e^{10} + e^{-20}} = -10$

approximation: $L(u_1) \boxplus L(u_2) \approx 1 \cdot (-1) \cdot \min\{10, 20\} = -10$

MAP Decoding with Log-Likelihood Ratios (LLR)

INSTITUT FÜR NACHRICHTENTECHNIK

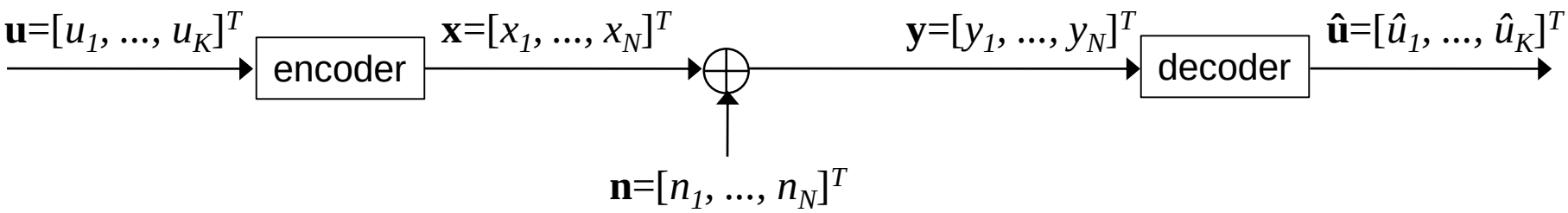
$$\mathbf{u}=[u_1, \ldots, u_K]^T \longrightarrow \boxed{\text{encoder}} \xrightarrow{\mathbf{x}=[x_1, \ldots, x_N]^T} \boxed{\text{channel}} \xrightarrow{\mathbf{y}=[y_1, \ldots, y_N]^T} \boxed{\text{decoder}} \longrightarrow \hat{\mathbf{u}}=[\hat{u}_1, \ldots, \hat{u}_K]^T$$

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \prod_{i=1}^{N} P(y_i \mid x_i) \prod_{k=1}^{K} P(u_k) = \arg\max_{\mathbf{u}\to\mathbf{x}} \sum_{i=1}^{N} \log P(y_i \mid x_i) + \sum_{k=1}^{K} \log P(u_k)$$

$$\left.\begin{aligned} P(u_k=+1) &= \frac{e^{L(u_k)}}{1+e^{L(u_k)}} = \frac{e^{L(u_k)/2} e^{L(u_k)/2}}{1+e^{L(u_k)}} = \frac{e^{L(u_k)/2}}{1+e^{L(u_k)}} e^{u_k L(u_k)/2} \\ P(u_k=-1) &= \frac{1}{1+e^{L(u_k)}} = \frac{e^{L(u_k)/2} e^{-L(u_k)/2}}{1+e^{L(u_k)}} = \frac{e^{L(u_k)/2}}{1+e^{L(u_k)}} e^{u_k L(u_k)/2} \end{aligned}\right\} P(u_k=\pm 1) = \frac{e^{L(u_k)/2}}{\underbrace{1+e^{L(u_k)}}_{\text{independent of } u_k}} e^{u_k L(u_k)/2}$$

$$\begin{aligned} \hat{\mathbf{u}} &= \arg\max_{\mathbf{u}\to\mathbf{x}} \sum_{i=1}^{N} \left(\cancel{\text{const}} + \frac{1}{2} x_i L(y_i \mid x_i) \right) + \sum_{k=1}^{K} \left(\cancel{\text{const}} + \frac{1}{2} u_k L_a(u_k) \right) \\ &= \arg\max_{\mathbf{u}\to\mathbf{x}} \frac{1}{2} \sum_{i=1}^{N} x_i L(y_i \mid x_i) + \frac{1}{2} \sum_{k=1}^{K} u_k L_a(u_k) \end{aligned}$$

MAP Decoding with Log-Likelihood Ratios (LLR) in AWGN Channel



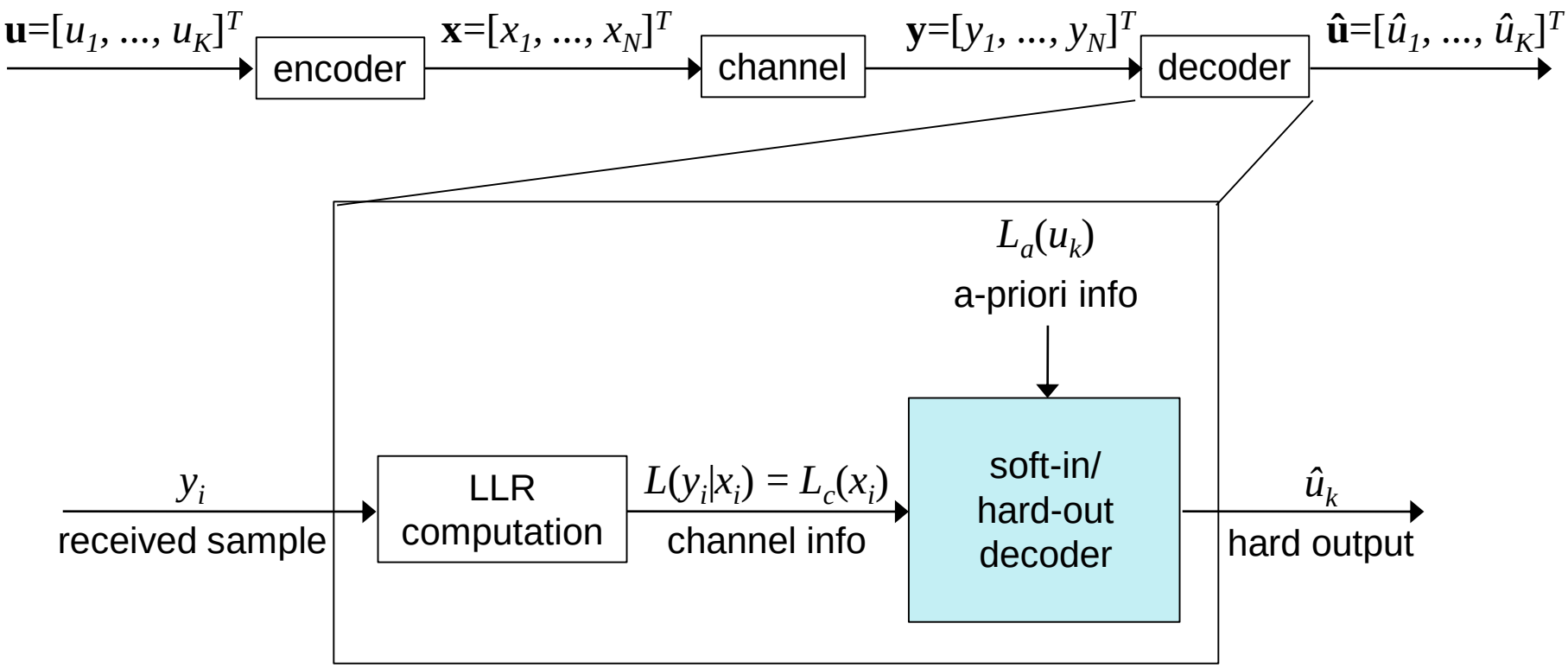
AWGN $f_N(n) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{n^2}{2\sigma_N^2}}$

$$L(y_i \mid x_i) = \log \frac{P(y_i \mid x_i = +1)}{P(y_i \mid x_i = -1)} = \log \frac{\frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(y_i-1)^2}{2\sigma_N^2}}}{\frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(y_i+1)^2}{2\sigma_N^2}}} = \frac{-(y_i-1)^2 + (y_i+1)^2}{2\sigma_N^2} = \frac{-(y_i^2 - 2y_i + 1) + (y_i^2 + 2y_i + 1)}{2\sigma_N^2}$$

$$= \frac{4y_i}{2\sigma_N^2} = \frac{2}{\sigma_N^2} y_i$$

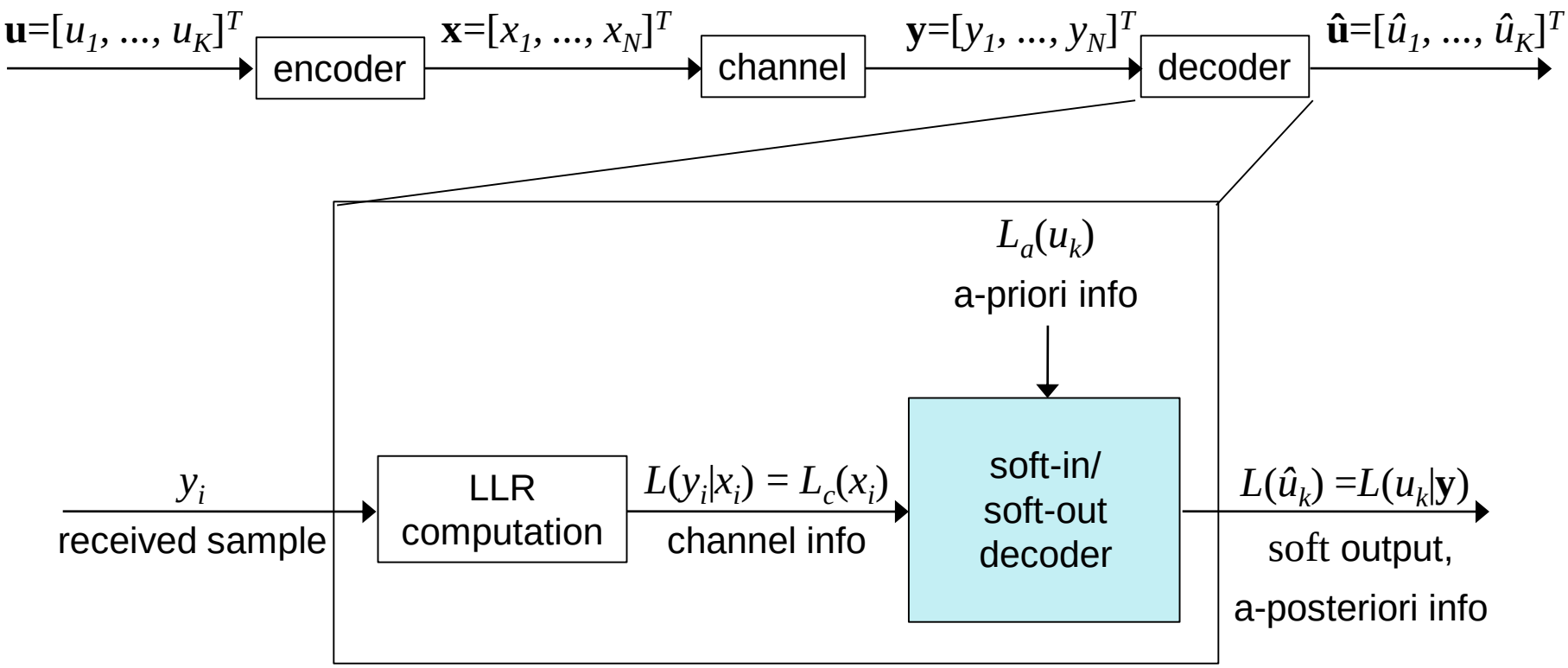
$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i) + \frac{1}{2}\sum_{k=1}^{K} u_k L_a(u_k) = \arg\max_{\mathbf{u}\to\mathbf{x}} \frac{1}{\sigma_N^2}\sum_{i=1}^{N} x_i y_i + \frac{1}{2}\sum_{k=1}^{K} u_k L_a(u_k)$$

MAP Decoding with Log-Likelihood Ratios (LLR)



$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}\to\mathbf{x}} \sum_{i=1}^{N} \frac{1}{2} x_i L(y_i \mid x_i) + \sum_{k=1}^{K} \frac{1}{2} u_k L_a(u_k)$$

Soft-In Soft-Out Decoder

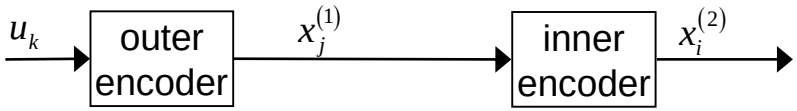


$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})}$$

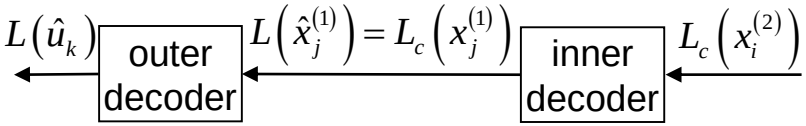
Reliability information in soft output can be exploited by outer decoder or subsequent detection stage, e.g. source decoder.

Application of Soft-Out Decoding: Concatenated Codes

Encoder

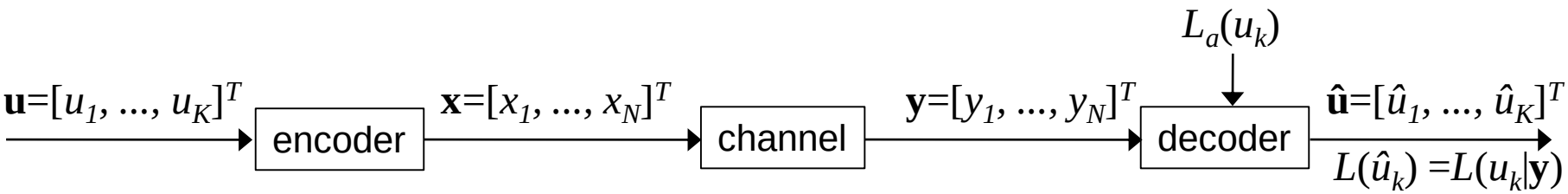


Decoder

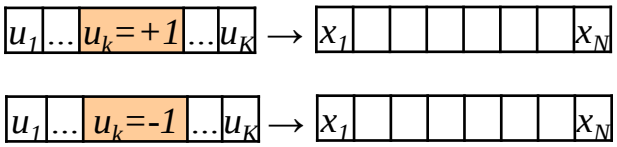


Soft-In Soft-Out Decoder

INSTITUT FÜR NACHRICHTENTECHNIK



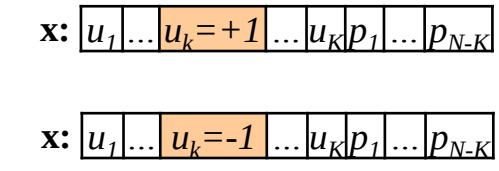
$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{x} \mid \mathbf{y})}{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{x} \mid \mathbf{y})}$$



$$= \log \frac{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{y} \mid \mathbf{x}) P(\mathbf{x}) \frac{1}{P(\mathbf{y})}}{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{y} \mid \mathbf{x}) P(\mathbf{x}) \frac{1}{P(\mathbf{y})}} = \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{y} \mid \mathbf{x}) P(u_k = +1) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{y} \mid \mathbf{x}) P(u_k = -1) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)} = \log \frac{P(u_k = +1)}{P(u_k = -1)} + \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{y} \mid \mathbf{x}) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{y} \mid \mathbf{x}) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)}$$

for systematic encoders:

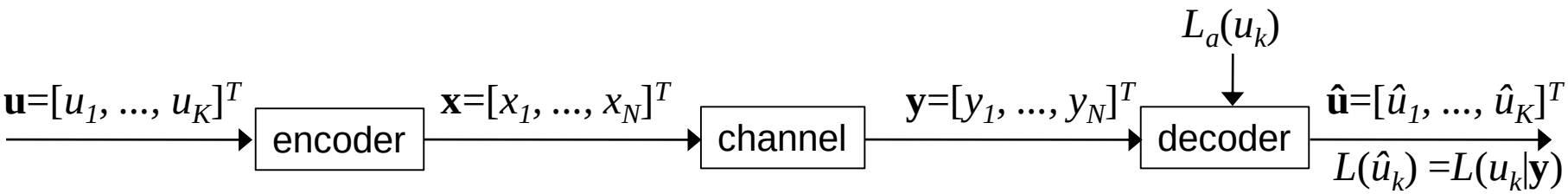
$$= \log \frac{P(u_k = +1)}{P(u_k = -1)} + \log \frac{P(y_k \mid x_k = +1) \sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \prod_{\substack{i=1 \\ i \neq k}}^{N} P(y_i \mid x_i) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)}{P(y_k \mid x_k = -1) \sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \prod_{\substack{i=1 \\ i \neq k}}^{N} P(y_i \mid x_i) \prod_{\substack{j=1 \\ j \neq k}}^{K} P(u_j)}$$



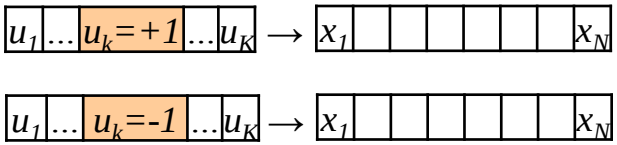
$$= \underbrace{\log \frac{P(u_k = +1)}{P(u_k = -1)}}_{\substack{L_a(u_k) \\ \text{a-priori info}}} + \underbrace{\log \frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)}}_{\substack{L_c(u_k) = L_c(x_k) \\ \text{channel info}}} + \underbrace{\log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j)\right)}}_{L_e(u_k) \text{ extrinsic info}}$$

Soft-In Soft-Out Decoder

INSTITUT FÜR NACHRICHTENTECHNIK



$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{x} \mid \mathbf{y})}{\sum_{\mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{x} \mid \mathbf{y})}$$



$$= \underbrace{\log \frac{P(u_k = +1)}{P(u_k = -1)}}_{L_a(u_k)} + \underbrace{\log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} P(\mathbf{y} \mid \mathbf{x}) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L(u_j)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} P(\mathbf{y} \mid \mathbf{x}) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L(u_j)\right)}}_{L_c(u_k) + L_e(u_k)}$$

a-priori info

channel plus extrinsic info

The **extrinsic information** is the information which is gained through the decoding process.
It is the information on the desired bit u_k which is obtained from all the other bits in the codeword via the code constraints.

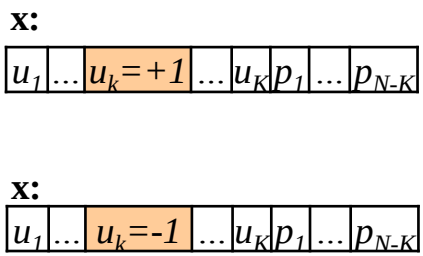
for systematic encoders:

$$L(\hat{u}_k) = \underbrace{\log \frac{P(u_k = +1)}{P(u_k = -1)}}_{L_a(u_k)} + \underbrace{\log \frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)}}_{L_c(u_k) = L_c(x_k)} + \underbrace{\log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_k L_a(u_k)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_k L_a(u_k)\right)}}_{L_e(u_k)}$$

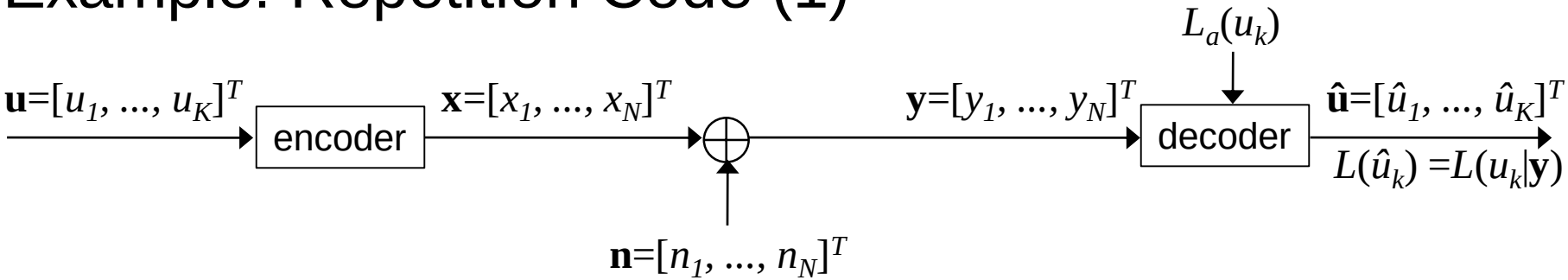
a-priori info

channel info

$L_e(u_k)$ extrinsic info



Soft-In Soft-Out Decoding
Example: Repetition Code (1)



Example: $R=1/N$ repetition code

$\mathbf{x}=[x_1, \ldots, x_N]^T = [u, \ldots, u]^T$

$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_k}^{+}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_k}^{-}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}$$

+1	...	+1
-1	...	-1

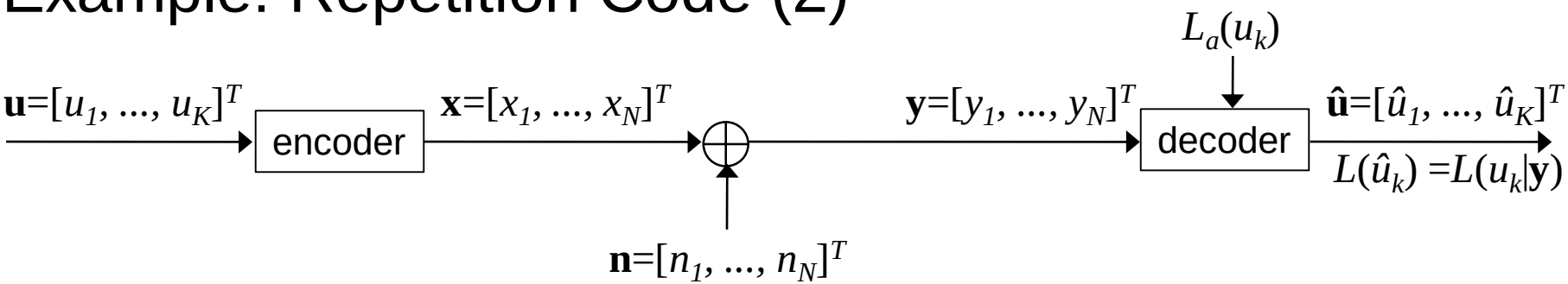
$$= \log \frac{\exp\left(\frac{1}{2}\sum_{i=1}^{N}(+1) L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}(+1) L_a(u_k)\right)}{\exp\left(\frac{1}{2}\sum_{i=1}^{N}(-1) L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}(-1) L_a(u_k)\right)}$$

$$= \sum_{i=1}^{N} L(y_i \mid x_i) + L_a(u_k)$$

$$= \frac{2}{\sigma_n^2}\sum_{i=1}^{N} y_i + L_a(u_k)$$

in AWGN channel

Soft-In Soft-Out Decoding
Example: Repetition Code (2)



Example: $R=1/3$ repetition code

$\mathbf{u}$	$+1$	-1	-1
$\mathbf{x}$	$[1,1,1]$	$[-1,-1,-1]$	$[-1,-1,-1]$
$\mathbf{y}$	[0.3, -0.1, 2]	[0.1, 0.2, -2]	[0.1, 0.2, -0.2]
$L(\hat{u})$	$4.4/\sigma_n^2$	$-3.4/\sigma_n^2$	$0.2/\sigma_n^2$
$\hat{u}$	$+1$	-1	$+1$

A-priori info:

$P_a(u_i=+1) = P_a(u_i=-1) = 1/2$

$\Rightarrow L_a(u_i)=0$

SNR in dB	σ_n^2
0	1
5	$10^{-1/2}$
10	0.1
100	10^{-10}

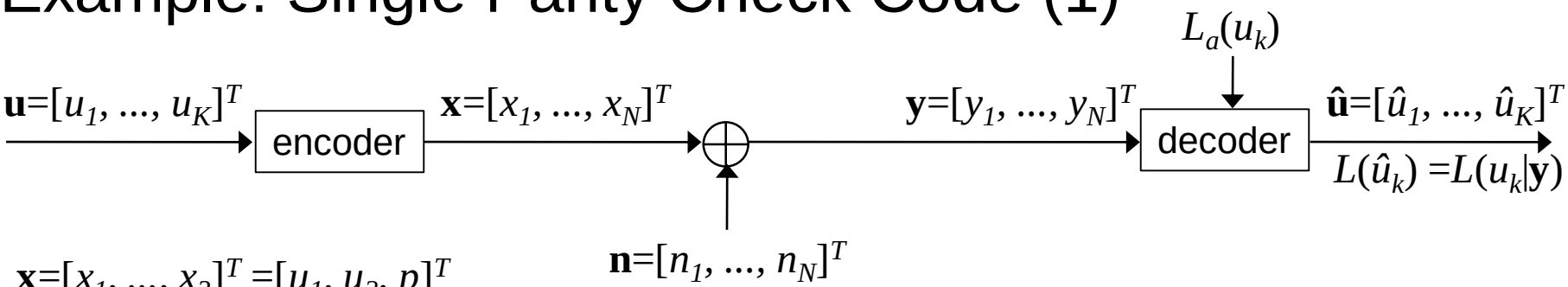
$$L(\hat{u}) = \frac{2}{\sigma_n^2}\sum_{i=1}^{N} y_i + L_a(u)$$

Soft-output LLR contains estimate on bit error probability p_e (bit error rate (BER)):

$$|L(\hat{u})| = \log\frac{1-p_e}{p_e}$$

$$p_e = \frac{1}{1+e^{|L(\hat{u})|}}$$

Soft-In Soft-Out Decoding
Example: Single Parity Check Code (1)



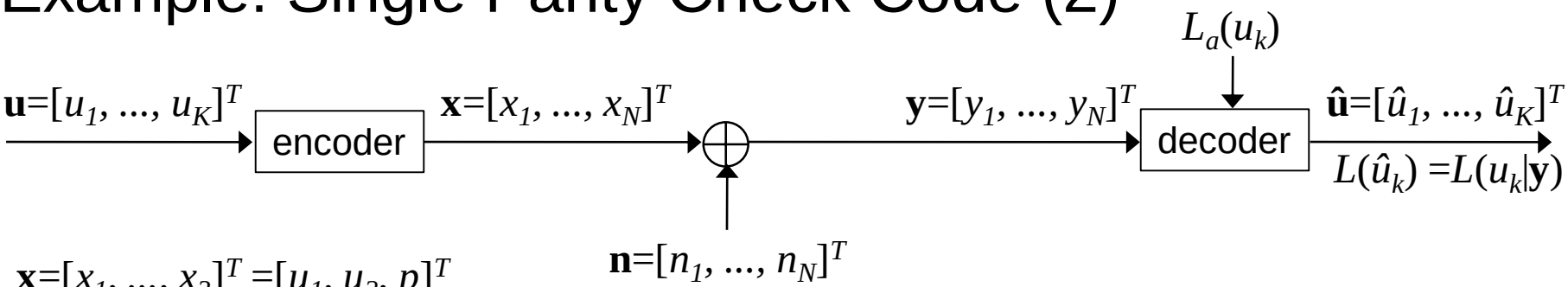
$\mathbf{x}=[x_1, \ldots, x_3]^T = [u_1, u_2, p]^T$

info bits $u_1 u_2$	parity bit $p = u_1 \oplus u_2$	
$+1\ +1$	$+1$	$\mathbf{x}_1 \in \mathcal{C}_{u_1}^{+}, \mathcal{C}_{u_2}^{+}$
$+1\ -1$	-1	$\mathbf{x}_2 \in \mathcal{C}_{u_1}^{+}, \mathcal{C}_{u_2}^{-}$
$-1\ +1$	-1	$\mathbf{x}_3 \in \mathcal{C}_{u_1}^{-}, \mathcal{C}_{u_2}^{+}$
$-1\ -1$	$+1$	$\mathbf{x}_4 \in \mathcal{C}_{u_1}^{-}, \mathcal{C}_{u_2}^{-}$

$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}$$

$$L(\hat{u}_1) = \log \frac{e^{\frac{1}{2}[(+1)L(y_1|x_1)+(+1)L(y_2|x_2)+(+1)L(y_3|x_3)]}\, e^{\frac{1}{2}[(+1)L_a(u_1)+(+1)L_a(u_2)]} + e^{\frac{1}{2}[(+1)L(y_1|x_1)+(-1)L(y_2|x_2)+(-1)L(y_3|x_3)]}\, e^{\frac{1}{2}[(+1)L_a(u_1)+(-1)L_a(u_2)]}}{e^{\frac{1}{2}[(-1)L(y_1|x_1)+(+1)L(y_2|x_2)+(-1)L(y_3|x_3)]}\, e^{\frac{1}{2}[(-1)L_a(u_1)+(+1)L_a(u_2)]} + e^{\frac{1}{2}[(-1)L(y_1|x_1)+(-1)L(y_2|x_2)+(+1)L(y_3|x_3)]}\, e^{\frac{1}{2}[(-1)L_a(u_1)+(-1)L_a(u_2)]}}$$

Soft-In Soft-Out Decoding
Example: Single Parity Check Code (2)



$\mathbf{x}=[x_1, ..., x_3]^T = [u_1, u_2, p]^T$

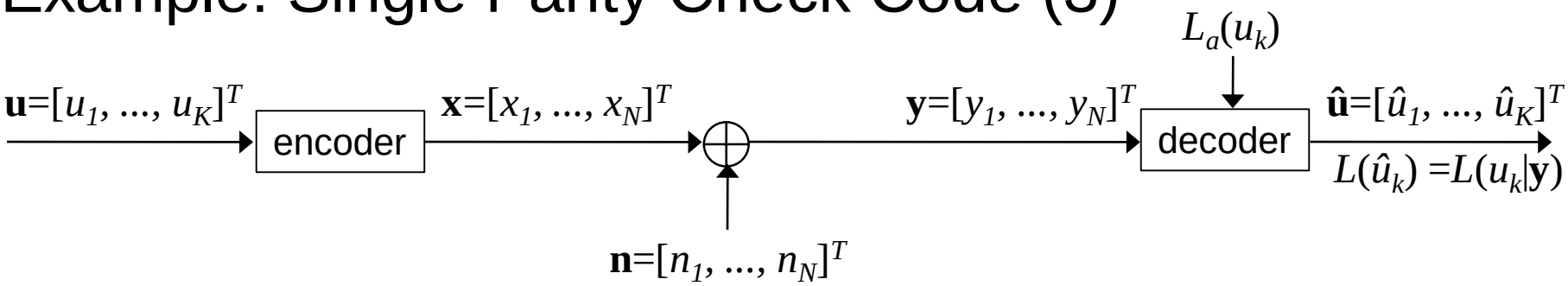
info bits $u_1 u_2$	parity bit $p = u_1 \oplus u_2$
+1+1	+1
+1-1	-1
-1+1	-1
-1-1	+1

$$u_i = u_j \oplus p$$
$$\Rightarrow L_e(u_i) = \left[L_c(u_j) + L_a(u_j)\right] \boxplus L_c(p)$$

$$L(\hat{u}_1) = \log \frac{P(u_1 = +1 \mid \mathbf{y})}{P(u_1 = -1 \mid \mathbf{y})} = \underbrace{L(y_1 \mid x_1)}_{\text{channel info}} + \underbrace{L_a(u_1)}_{\text{a-priori info}} + \underbrace{\left(L(y_2 \mid x_2) + L_a(u_2)\right) \boxplus L(y_3 \mid x_3)}_{\text{extrinsic info } L_e(u_1)}$$

$$L(\hat{u}_2) = \log \frac{P(u_2 = +1 \mid \mathbf{y})}{P(u_2 = -1 \mid \mathbf{y})} = \underbrace{L(y_2 \mid x_2)}_{\text{channel info}} + \underbrace{L_a(u_2)}_{\text{a-priori info}} + \underbrace{\left(L(y_1 \mid x_1) + L_a(u_1)\right) \boxplus L(y_3 \mid x_3)}_{\text{extrinsic info } L_e(u_2)}$$

Soft-In Soft-Out Decoding
Example: Single Parity Check Code (3)



$$L(\hat{u}_1) = L(u_1 \mid \mathbf{y}) = \log \frac{P(u_1 = +1 \mid \mathbf{y})}{P(u_1 = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{+}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{-}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}$$

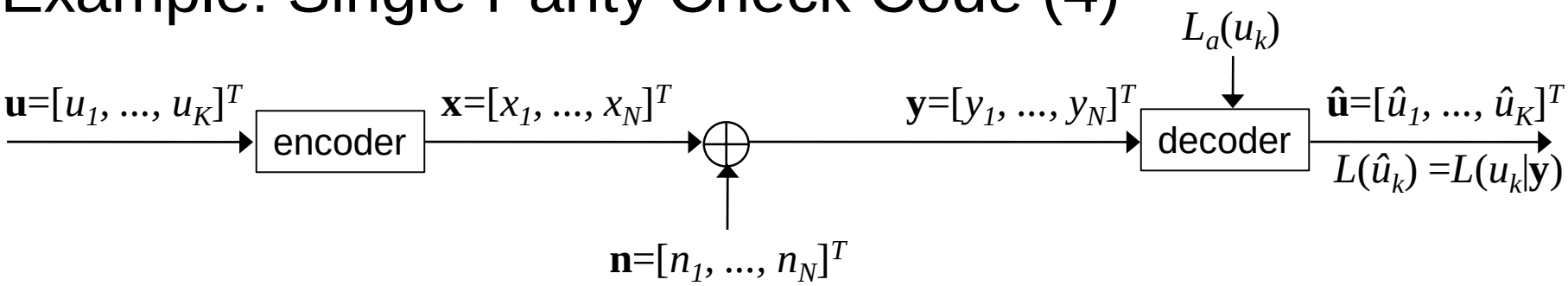
$$= \log \frac{e^{\frac{1}{2}\left[(+1)L(y_1|x_1)+(+1)L(y_2|x_2)+(+1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(+1)L_a(u_1)+(+1)L_a(u_2)\right]} + e^{\frac{1}{2}\left[(+1)L(y_1|x_1)+(-1)L(y_2|x_2)+(-1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(+1)L_a(u_1)+(-1)L_a(u_2)\right]}}{e^{\frac{1}{2}\left[(-1)L(y_1|x_1)+(+1)L(y_2|x_2)+(-1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(-1)L_a(u_1)+(+1)L_a(u_2)\right]} + e^{\frac{1}{2}\left[(-1)L(y_1|x_1)+(-1)L(y_2|x_2)+(+1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(-1)L_a(u_1)+(-1)L_a(u_2)\right]}}$$

$$= \log \frac{e^{\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L_a(u_1)} [e^{\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_2)} + e^{-\frac{1}{2}L(y_2|x_2)} e^{-\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_2)}]}{e^{-\frac{1}{2}L(y_1|x_1)} e^{-\frac{1}{2}L_a(u_1)} [e^{\frac{1}{2}L(y_2|x_2)} e^{-\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_2)} + e^{-\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_2)}]}$$

$$= L(y_1 \mid x_1) + L_a(u_1) + \log \frac{e^{\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L_a(u_2)} [e^{\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_2)} + e^{-\frac{1}{2}L(y_2|x_2)} e^{-\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_2)}]}{e^{\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L_a(u_2)} [e^{\frac{1}{2}L(y_2|x_2)} e^{-\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_2)} + e^{-\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_2)}]}$$

$$= L(y_1 \mid x_1) + L_a(u_1) + \underbrace{\log \frac{e^{\frac{1}{2}L(y_2|x_2)} e^{L(y_3|x_3)} e^{L_a(u_2)} + 1}{e^{L(y_2|x_2)} e^{L_a(u_2)} + e^{L(y_3|x_3)}}}_{(L(y_2 \mid x_2) + L_a(u_2)) \boxplus L(y_3 \mid x_3)} = L(y_1 \mid x_1) + L_a(u_1) + (L(y_2 \mid x_2) + L_a(u_2)) \boxplus L(y_3 \mid x_3)$$

Soft-In Soft-Out Decoding
Example: Single Parity Check Code (4)



$$L(\hat{u}_2) = L(u_2 \mid \mathbf{y}) = \log \frac{P(u_2 = +1 \mid \mathbf{y})}{P(u_2 = -1 \mid \mathbf{y})} = \log \frac{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_2}^{+}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_2}^{-}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right)\exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}$$

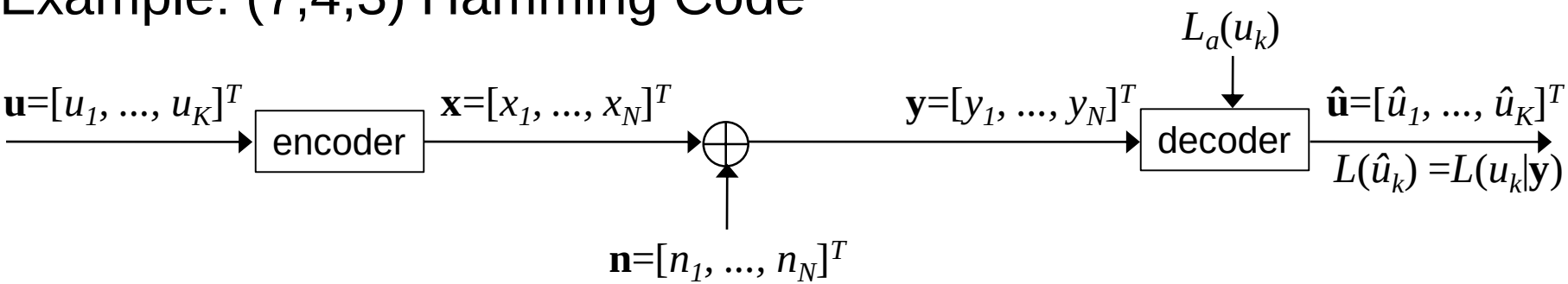
$$= \log \frac{e^{\frac{1}{2}\left[(+1)L(y_1|x_1)+(+1)L(y_2|x_2)+(+1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(+1)L_a(u_1)+(+1)L_a(u_2)\right]} + e^{\frac{1}{2}\left[(-1)L(y_1|x_1)+(+1)L(y_2|x_2)+(-1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(-1)L_a(u_1)+(+1)L_a(u_2)\right]}}{e^{\frac{1}{2}\left[(+1)L(y_1|x_1)+(-1)L(y_2|x_2)+(-1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(+1)L_a(u_1)+(-1)L_a(u_2)\right]} + e^{\frac{1}{2}\left[(-1)L(y_1|x_1)+(-1)L(y_2|x_2)+(+1)L(y_3|x_3)\right]} e^{\frac{1}{2}\left[(-1)L_a(u_1)+(-1)L_a(u_2)\right]}}$$

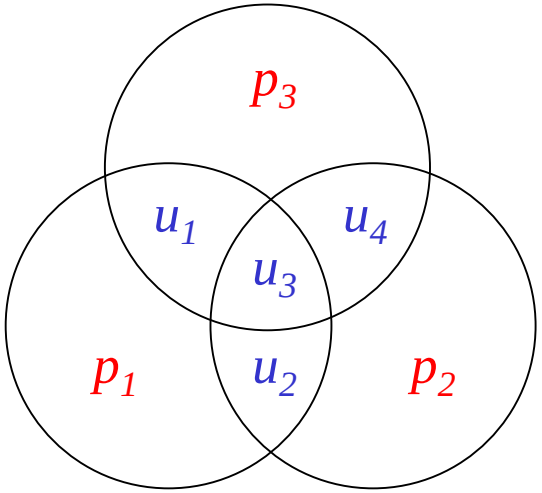
$$= \log \frac{e^{\frac{1}{2}L(y_2|x_2)} e^{\frac{1}{2}L_a(u_2)} [e^{\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_1)} + e^{-\frac{1}{2}L(y_1|x_1)} e^{-\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_1)}]}{e^{-\frac{1}{2}L(y_2|x_2)} e^{-\frac{1}{2}L_a(u_2)} [e^{\frac{1}{2}L(y_1|x_1)} e^{-\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_1)} + e^{-\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_1)}]}$$

$$= L(y_2 \mid x_2) + L_a(u_2) + \log \frac{e^{\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L_a(u_1)} [e^{\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_1)} + e^{-\frac{1}{2}L(y_1|x_1)} e^{-\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_1)}]}{e^{\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L_a(u_1)} [e^{\frac{1}{2}L(y_1|x_1)} e^{-\frac{1}{2}L(y_3|x_3)} e^{\frac{1}{2}L_a(u_1)} + e^{-\frac{1}{2}L(y_1|x_1)} e^{\frac{1}{2}L(y_3|x_3)} e^{-\frac{1}{2}L_a(u_1)}]}$$

$$= L(y_2 \mid x_2) + L_a(u_2) + \underbrace{\log \frac{e^{\frac{1}{2}L(y_1|x_1)} e^{L(y_3|x_3)} e^{L_a(u_1)} + 1}{e^{L(y_1|x_1)} e^{L_a(u_1)} + e^{L(y_3|x_3)}}}_{(L(y_1|x_1)+L_a(u_1)) \boxplus L(y_3|x_3)} = L(y_2 \mid x_2) + L_a(u_2) + (L(y_1 \mid x_1) + L_a(u_1)) \boxplus L(y_3 \mid x_3)$$

Soft-In Soft-Out Decoding
Example: (7,4,3) Hamming Code





$$\mathbf{x} = [u_1, u_2, u_3, u_4, p_1, p_2, p_3]$$

$$p_1 = u_1 \oplus u_2 \oplus u_3$$
$$p_2 = u_2 \oplus u_3 \oplus u_4$$
$$p_3 = u_1 \oplus u_3 \oplus u_4$$

$$L(\hat{u}_k) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} = \underbrace{L(y_k \mid u_k)}_{\text{channel info}} + \underbrace{L_a(u_k)}_{\text{a-priori info}} + \underbrace{\log \frac{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_k}^{+}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i\neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j\neq k}}^{K} u_j L_a(u_j) \right) \right]}{\sum_{\mathbf{u}\to\mathbf{x}\in\mathcal{C}_{u_k}^{-}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i\neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j\neq k}}^{K} u_j L_a(u_j) \right) \right]}_{\text{extrinsic info } L_e(u_k)}$$

Soft-In Soft-Out Decoding
Example: (7,4,3) Hamming Code

$\mathbf{x}_k$	
$\mathbf{x}_1 = +1+1+1+1+1+1+1$	$\mathbf{x}_1 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_2 = +1+1+1-1-1-1-1$	$\mathbf{x}_2 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_3 = +1+1-1+1-1-1+1$	$\mathbf{x}_3 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_4 = +1+1-1-1+1+1-1$	$\mathbf{x}_4 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_5 = +1-1+1+1-1+1-1$	$\mathbf{x}_5 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_6 = +1-1+1-1+1-1+1$	$\mathbf{x}_6 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_7 = +1-1-1+1+1-1-1$	$\mathbf{x}_7 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_8 = +1-1-1-1-1+1+1$	$\mathbf{x}_8 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_9 = -1+1+1+1+1-1-1$	$\mathbf{x}_9 \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{10} = -1+1+1-1-1+1+1$	$\mathbf{x}_{10} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{11} = -1+1-1+1-1+1-1$	$\mathbf{x}_{11} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{12} = -1+1-1-1+1-1+1$	$\mathbf{x}_{12} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{13} = -1-1+1+1-1-1+1$	$\mathbf{x}_{13} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{14} = -1-1+1-1+1+1-1$	$\mathbf{x}_{14} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{15} = -1-1-1+1+1+1+1$	$\mathbf{x}_{15} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{16} = -1-1-1-1-1-1-1$	$\mathbf{x}_{16} \in \mathcal{C}_{u_1}^{-}$

$$L(\hat{u}_k) = L(u_k \mid \mathbf{y}) = \log \frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})}$$

$$= \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \left[\exp\left(\frac{1}{2} \sum_{i=1}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{j=1}^{K} u_j L_a(u_j) \right) \right]}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \left[\exp\left(\frac{1}{2} \sum_{i=1}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{j=1}^{K} u_j L_a(u_j) \right) \right]}$$

$$= \log \frac{\exp\left(\frac{1}{2}(+1) L(y_k \mid x_k) \right) \exp\left((+1) L_a(u_k) \right) \sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j) \right) \right]}{\exp\left(\frac{1}{2}(-1) L(y_k \mid x_k) \right) \exp\left((-1) L_a(u_k) \right) \sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j) \right) \right]}$$

$$= L(y_k \mid x_k) + L_a(u_k) + \underbrace{\log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{+}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j) \right) \right]}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_k}^{-}} \left[\exp\left(\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{N} x_i L(y_i \mid x_i) \right) \exp\left(\frac{1}{2} \sum_{\substack{j=1 \\ j \neq k}}^{K} u_j L_a(u_j) \right) \right]}}_{L_e(u_k)}$$

Soft-In Soft-Out Decoding
Example: (7,4,3) Hamming Code

$\mathbf{x}_k$	
$\mathbf{x}_1 = +1+1+1+1+1+1+1$	$\mathbf{x}_1 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_2 = +1+1+1-1-1-1-1$	$\mathbf{x}_2 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_3 = +1+1-1+1-1-1+1$	$\mathbf{x}_3 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_4 = +1+1-1-1+1+1-1$	$\mathbf{x}_4 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_5 = +1-1+1+1-1+1-1$	$\mathbf{x}_5 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_6 = +1-1+1-1+1-1+1$	$\mathbf{x}_6 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_7 = +1-1-1+1+1-1-1$	$\mathbf{x}_7 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_8 = +1-1-1-1-1+1+1$	$\mathbf{x}_8 \in \mathcal{C}_{u_1}^{+}$
$\mathbf{x}_9 = -1+1+1+1+1-1-1$	$\mathbf{x}_9 \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{10} = -1+1+1-1-1+1+1$	$\mathbf{x}_{10} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{11} = -1+1-1+1-1+1-1$	$\mathbf{x}_{11} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{12} = -1+1-1-1+1-1+1$	$\mathbf{x}_{12} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{13} = -1-1+1+1-1-1+1$	$\mathbf{x}_{13} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{14} = -1-1+1-1+1+1-1$	$\mathbf{x}_{14} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{15} = -1-1-1+1+1+1+1$	$\mathbf{x}_{15} \in \mathcal{C}_{u_1}^{-}$
$\mathbf{x}_{16} = -1-1-1-1-1-1-1$	$\mathbf{x}_{16} \in \mathcal{C}_{u_1}^{-}$

$$L(\hat{u}_1) = L(u_1 \mid \mathbf{y}) = \log \frac{P(u_1 = +1 \mid \mathbf{y})}{P(u_1 = -1 \mid \mathbf{y})}$$

$$= \log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{+}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{-}} \exp\left(\frac{1}{2}\sum_{i=1}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{j=1}^{K} u_j L_a(u_j)\right)}$$

$$= L(y_1 \mid x_1) + L_a(u_1) + \underbrace{\log \frac{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{+}} \left[\exp\left(\frac{1}{2}\sum_{\substack{i=1 \\ i \neq 1}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{\substack{j=1 \\ j \neq 1}}^{K} u_j L_a(u_j)\right)\right]}{\sum_{\mathbf{u} \to \mathbf{x} \in \mathcal{C}_{u_1}^{-}} \left[\exp\left(\frac{1}{2}\sum_{\substack{i=1 \\ i \neq 1}}^{N} x_i L(y_i \mid x_i)\right) \exp\left(\frac{1}{2}\sum_{\substack{j=1 \\ j \neq 1}}^{K} u_j L_a(u_j)\right)\right]}}_{L_e(u_1)}$$

Soft-In Soft-Out Decoder

It is often desired to obtain a reliability information not only at the decoder input but also at the decoder output. This soft a-posteriori information on the decoder decisions can be represented in the form of log-likelihood ratios for each information bit. The reliability information on the decoder decisions can be exploited in subsequent detection stages for improved detection results. E.g. a source decoder can benefit from getting soft input values rather than hard decided channel decoder outputs.

Another application is in decoding of concatenated codes. The idea of concatenated codes is, to build a strong code by concatenation of weaker codes with manageable decoding complexity. An outer encoder encodes the information bits. The code bits of the outer encoder are then passed to an inner encoder which encodes the code bits of the outer code. The corresponding operations have to be performed at the decoder: The inner decoder decodes the received samples and produces estimates on the code bits of the outer code. If the innerd ecoder produces soft outputs, the outer decoder can make use of this reliability information and perform soft decision (soft-in) decoding.

The a-posteriori log-likelihood ratios $L(\hat{u}_k)$ (soft outputs) contain three parts of information: The a-priori information $L_a(u_k)$ which is obtained from knowledge of the source statistics. The channel information $L_c(u_k)$ which is the information which would be obtained in uncoded transmission from the received samples. The third part, the so-called extrinsic information, is the information which is gained through the decoding process. It is the information on the desired bit u_k which is obtained due to the code constraints from all information (source statistics, channel information) which is available on the other bits in the codeword. The extrinsic information is the new information which was not available before decoding.

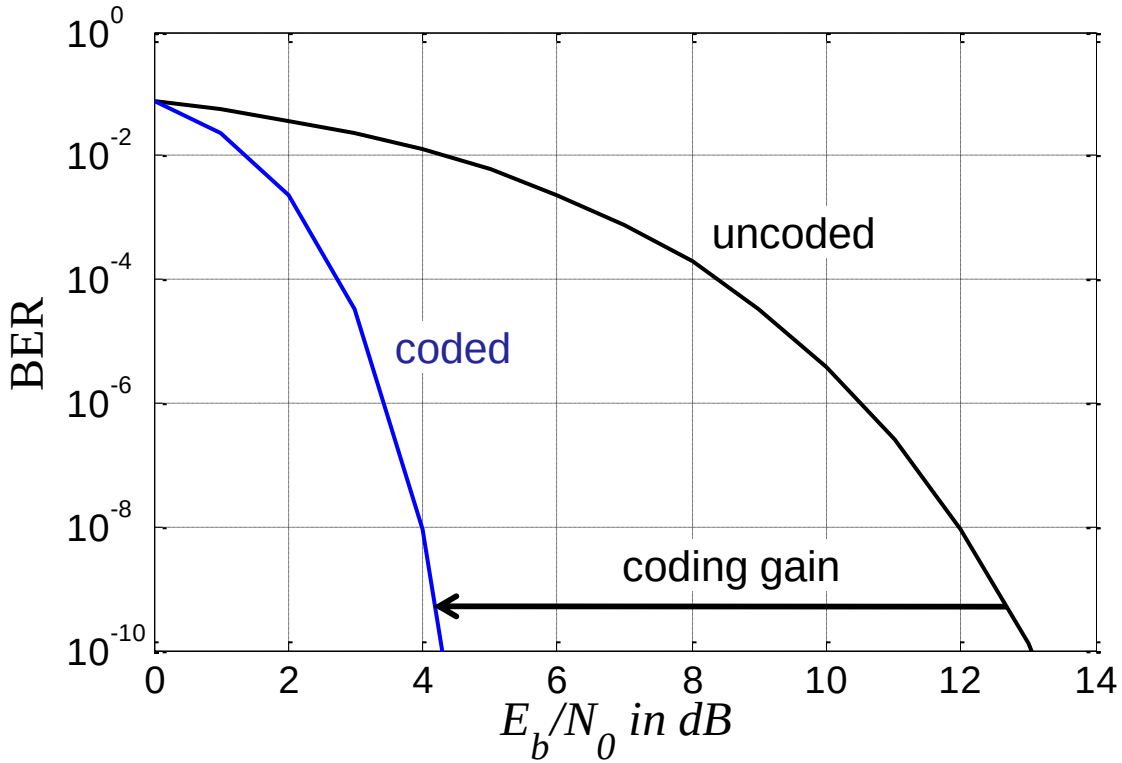
The soft-output log-likelihood ratio also provides an estimate on the error probability of the respective bit decision as reliability information.

Fair Comparison of Codes: E_b and E_s

K info bits → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

E_b: energy per info bit

$E_s = RE_b$: energy per code symbol



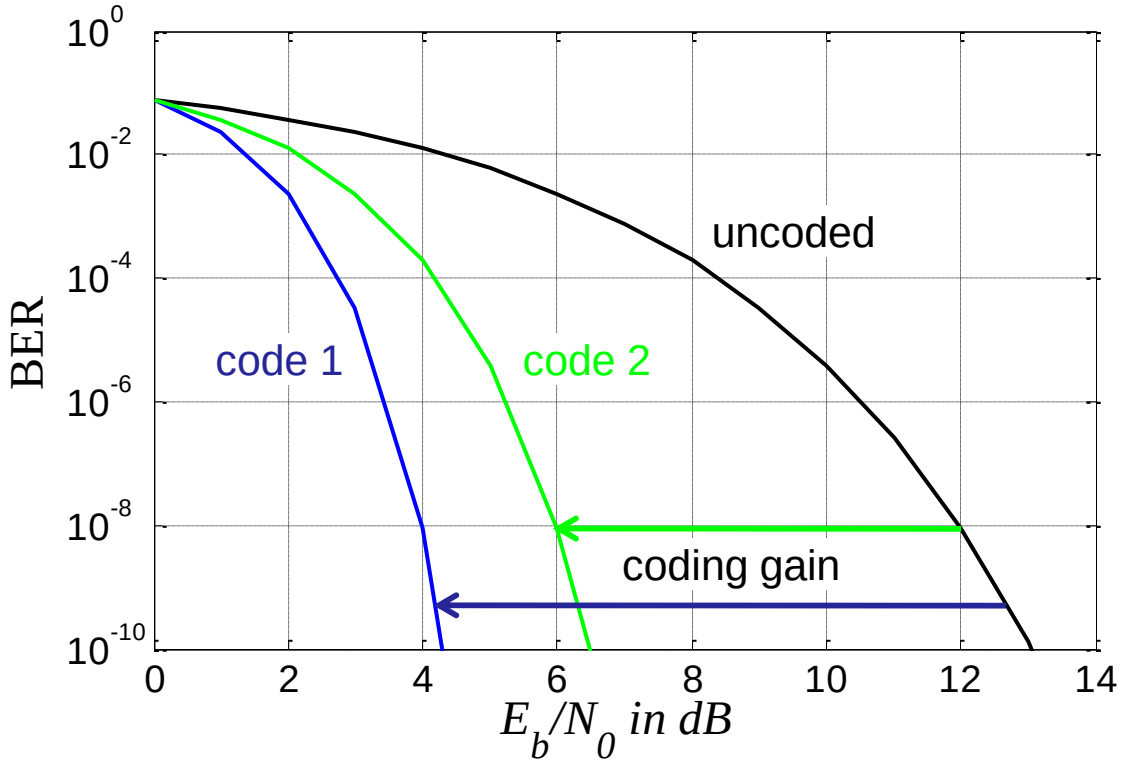
A fair comparison of codes has to be done in terms of E_b/N_0.

Fair Comparison of Codes: E_b and E_s

K info bits → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

E_b: energy per info bit

$E_s = RE_b$: energy per code symbol



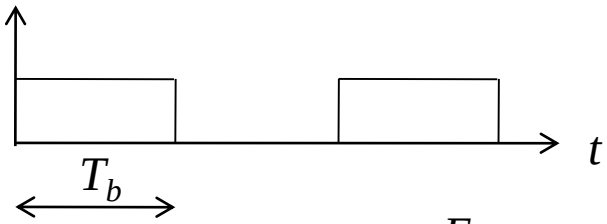
A fair comparison of codes has to be done in terms of E_b/N_0.

Fair Comparison of Codes: E_b and E_s

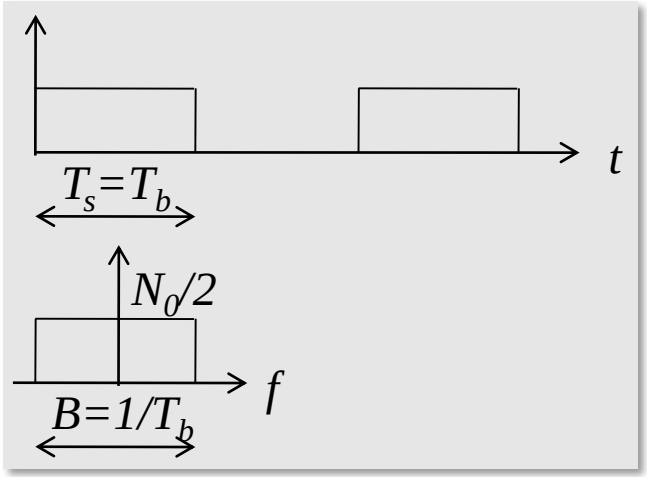
K info bits → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

E_b: energy per info bit

$E_s = RE_b$: energy per code symbol



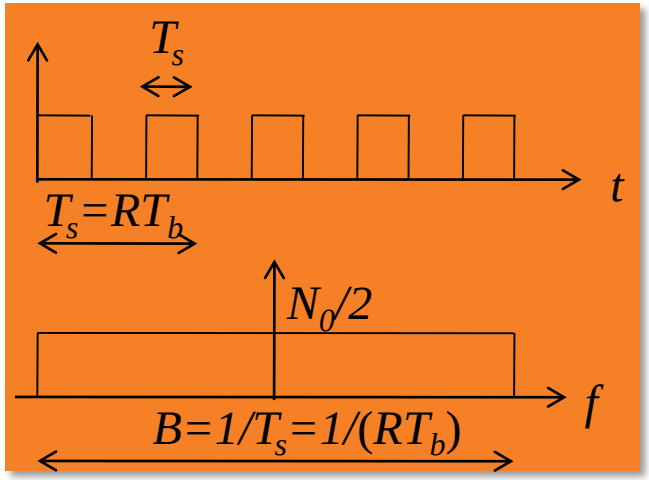
uncoded, $R=1$



coded, $R=K/N$

$$SNR_{\text{uncoded}} = \left.\frac{P_x}{P_n}\right|_{\text{uncoded}} = \frac{\frac{E_b}{T_b}}{\frac{N_0}{2T_b}} = \frac{2E_b}{N_0}$$

$$SNR_{\text{coded}} = \left.\frac{P_x}{P_n}\right|_{\text{coded}} = \frac{\frac{E_s}{T_s}}{\frac{N_0}{2T_s}} = \frac{\frac{E_b}{T_b}}{\frac{N_0}{2RT_b}} = \frac{2RE_b}{N_0} = R \cdot SNR\big|_{\text{uncoded}}$$



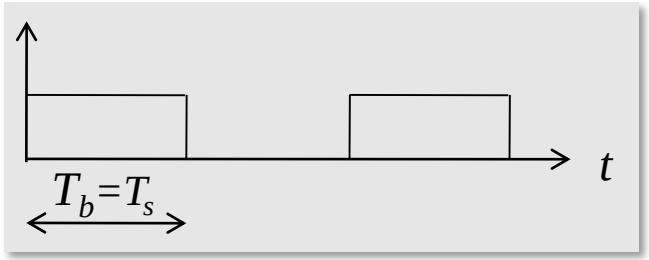
- Coding gain must overcompensate rate penalty factor R.
- Coded transmission with unchanged data rate causes bandwidth expansion.
- Fair comparison of codes always in terms of E_b/N_0.

Fair Comparison of Codes: E_b and E_s

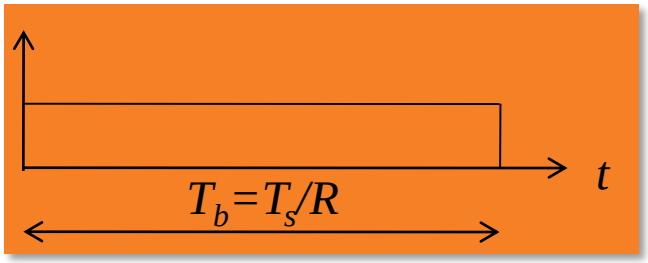
K info bits → encoder, code rate: $R = \frac{K}{N}$ → N code symbols

E_b: energy per info bit

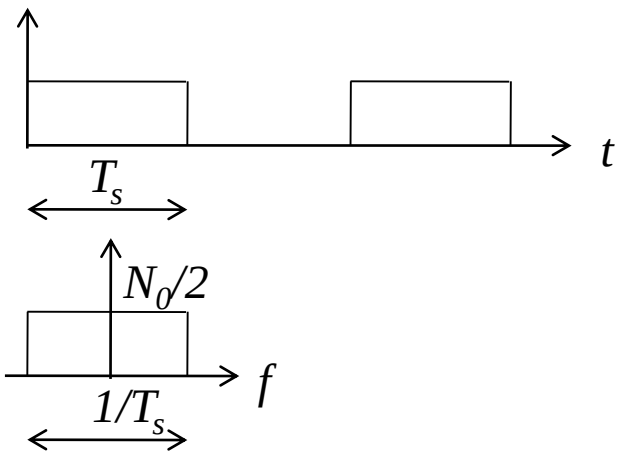
$E_s = RE_b$: energy per code symbol



uncoded, $R=1$



coded, $R=K/N$



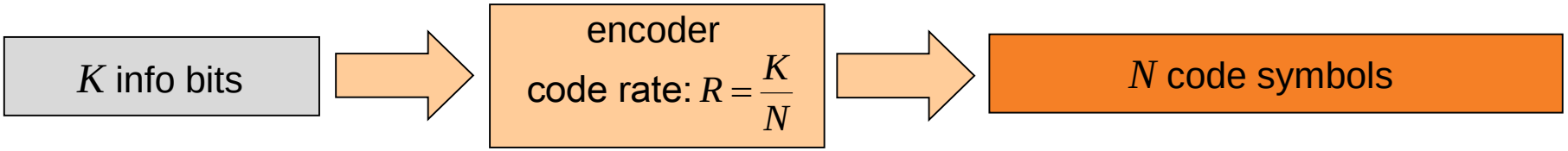
- Coding gain must overcompensate rate penalty factor R.
- Coded transmission with unchanged bandwidth causes lower information bit rate.
- Fair comparison of codes always in terms of E_b/N_0.

$$SNR_{\text{uncoded}} = \left.\frac{P_x}{P_n}\right|_{\text{uncoded}} = \frac{E_b / T_b}{N_0 / (2T_b)} = \frac{2E_b}{N_0}$$

$$SNR_{\text{coded}} = \left.\frac{P_x}{P_n}\right|_{\text{coded}} = \frac{E_s / T_s}{N_0 / (2T_s)} = \frac{E_b / T_b}{N_0 / (2T_s)}$$

$$= \frac{RE_b / T_s}{N_0 / (2T_s)} = \frac{2RE_b}{N_0} = R \cdot SNR\big|_{\text{uncoded}}$$

Fair Comparison of Codes: E_b and E_s



E_b: energy per info bit

T_b: info bit duration

$1/T_b$: info bit rate

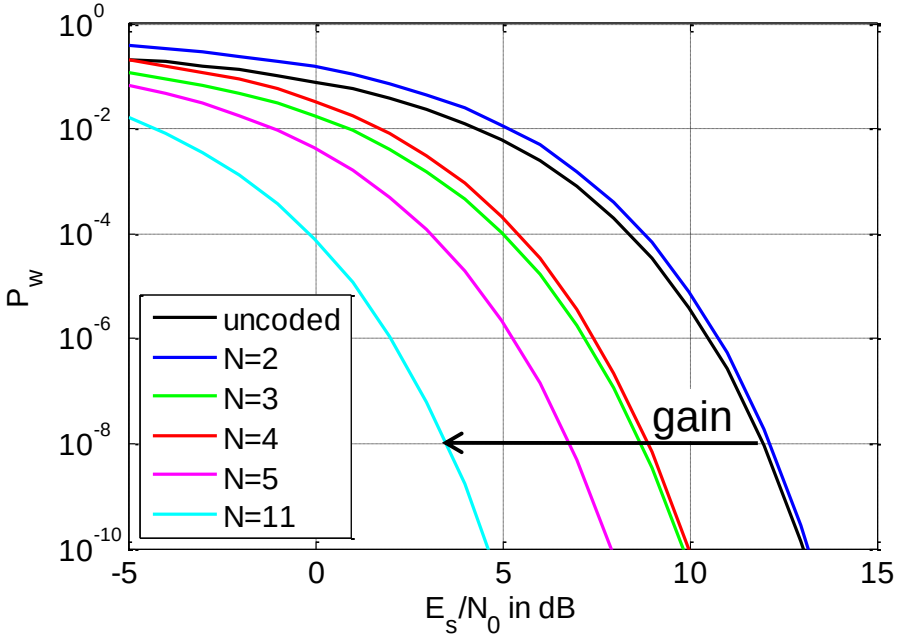
$E_s = RE_b$: energy per code symbol

$T_s = RT_b$: code symbol duration

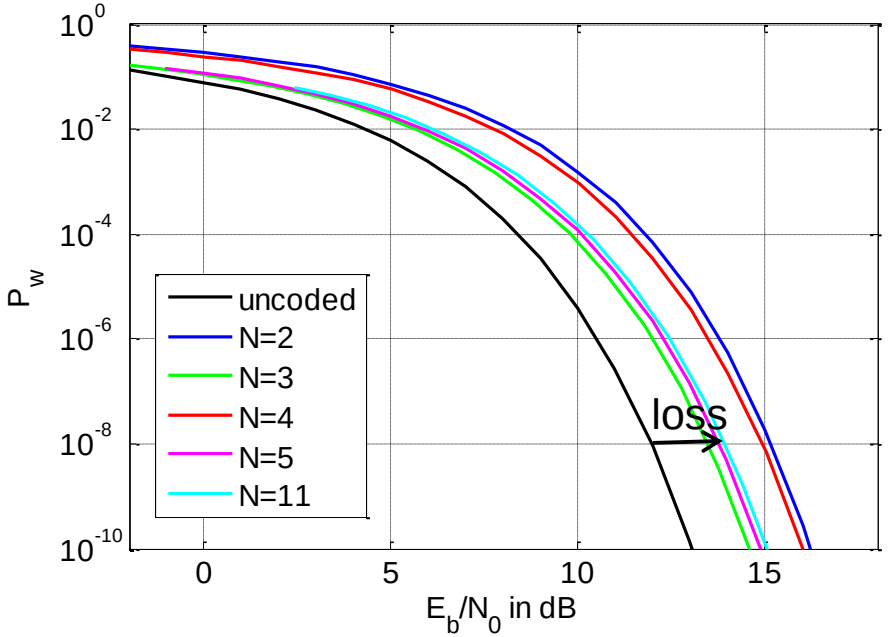
$1/T_s = 1/(RT_b)$: code symbol rate,
symbol rate on channel

Performance of Repetition Codes

- Rate $1/N$ repetition codes
- AWGN channel
- hard decision decoding



Repetition codes can improve performance in terms of E_s/N_0.

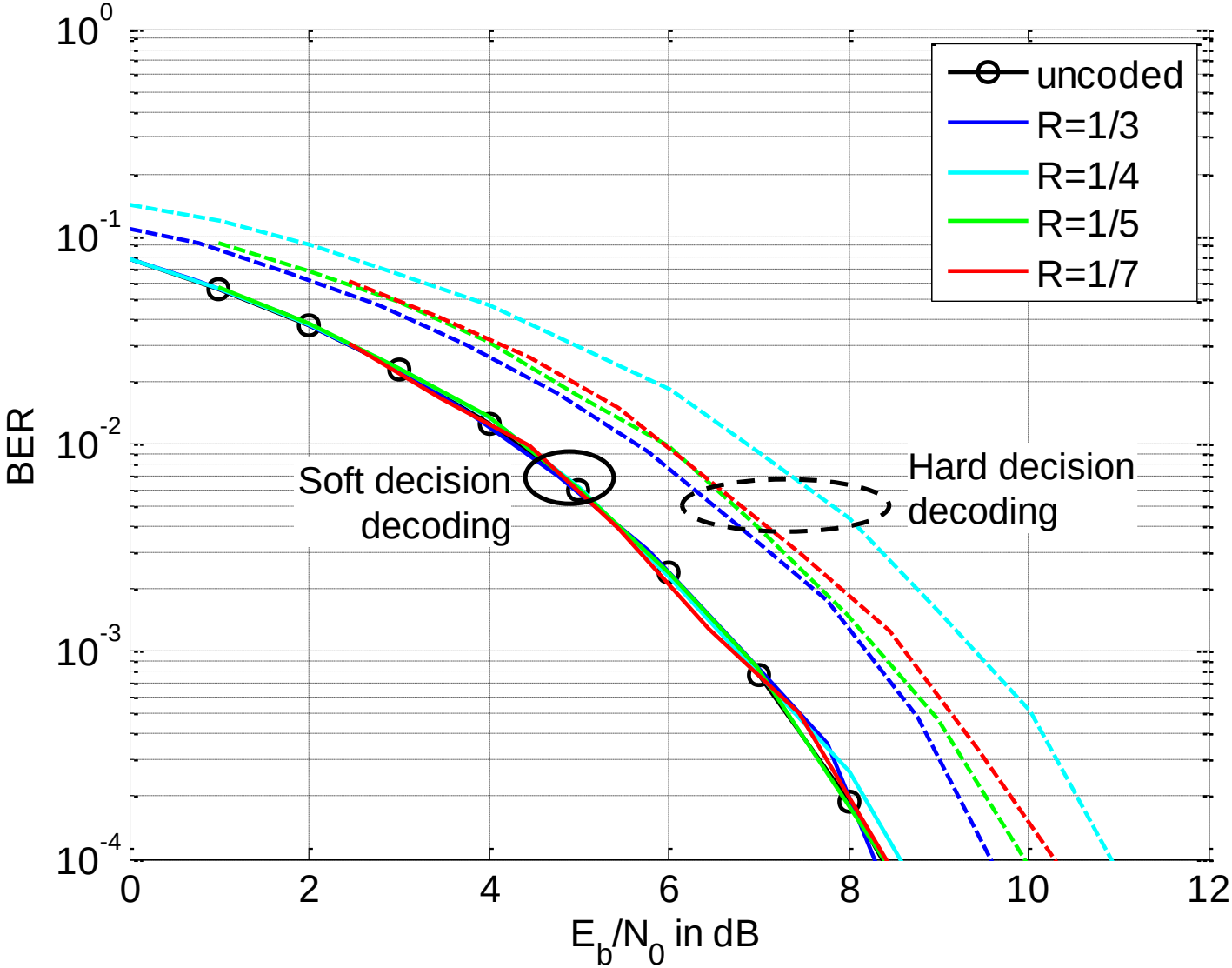


Repetition codes perform worse than uncoded transmission.

We need better codes than repetition codes !

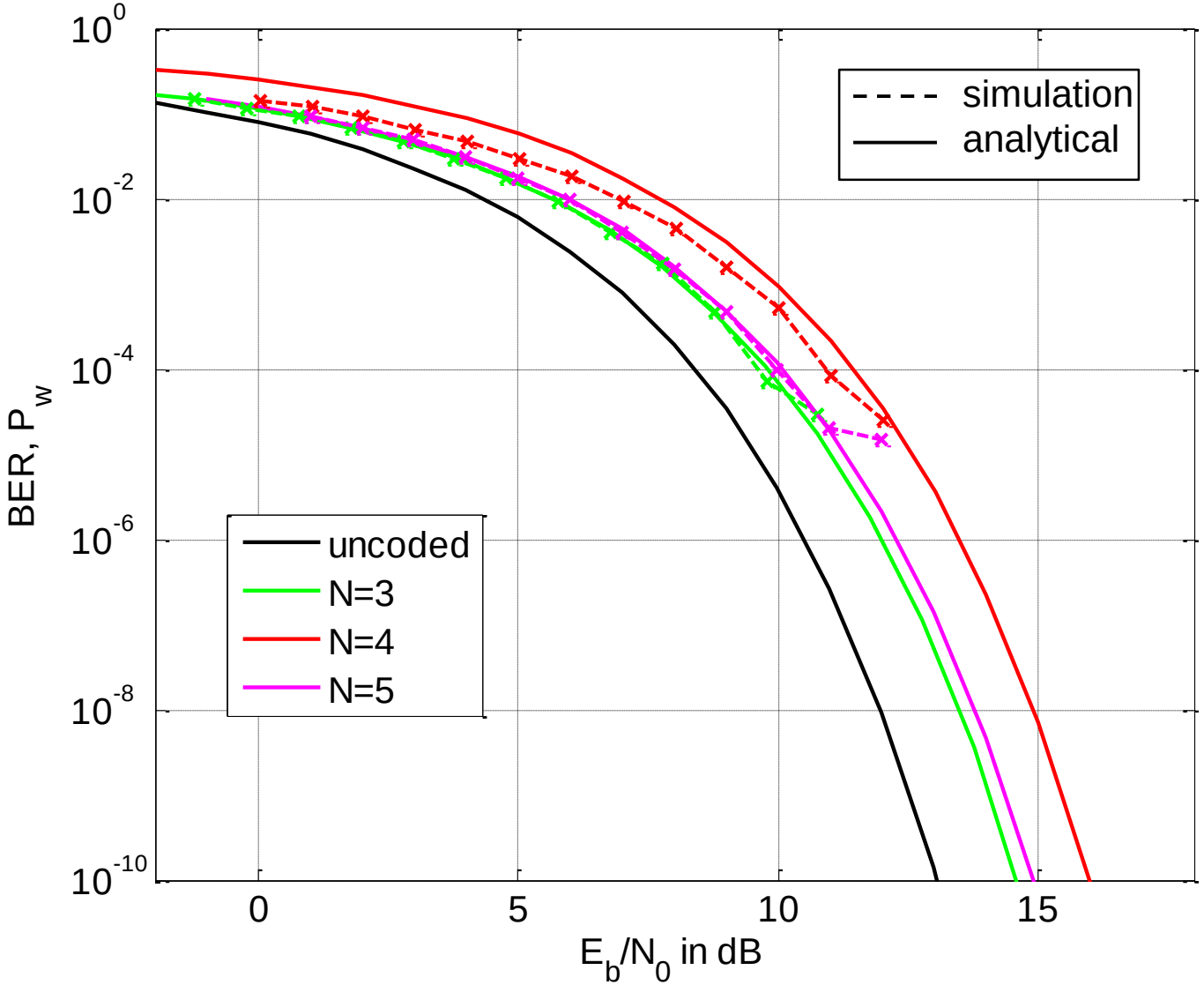
BER Performance of Repetition Codes-Simulation Results

Rate R binary repetition codes, AWGN channel



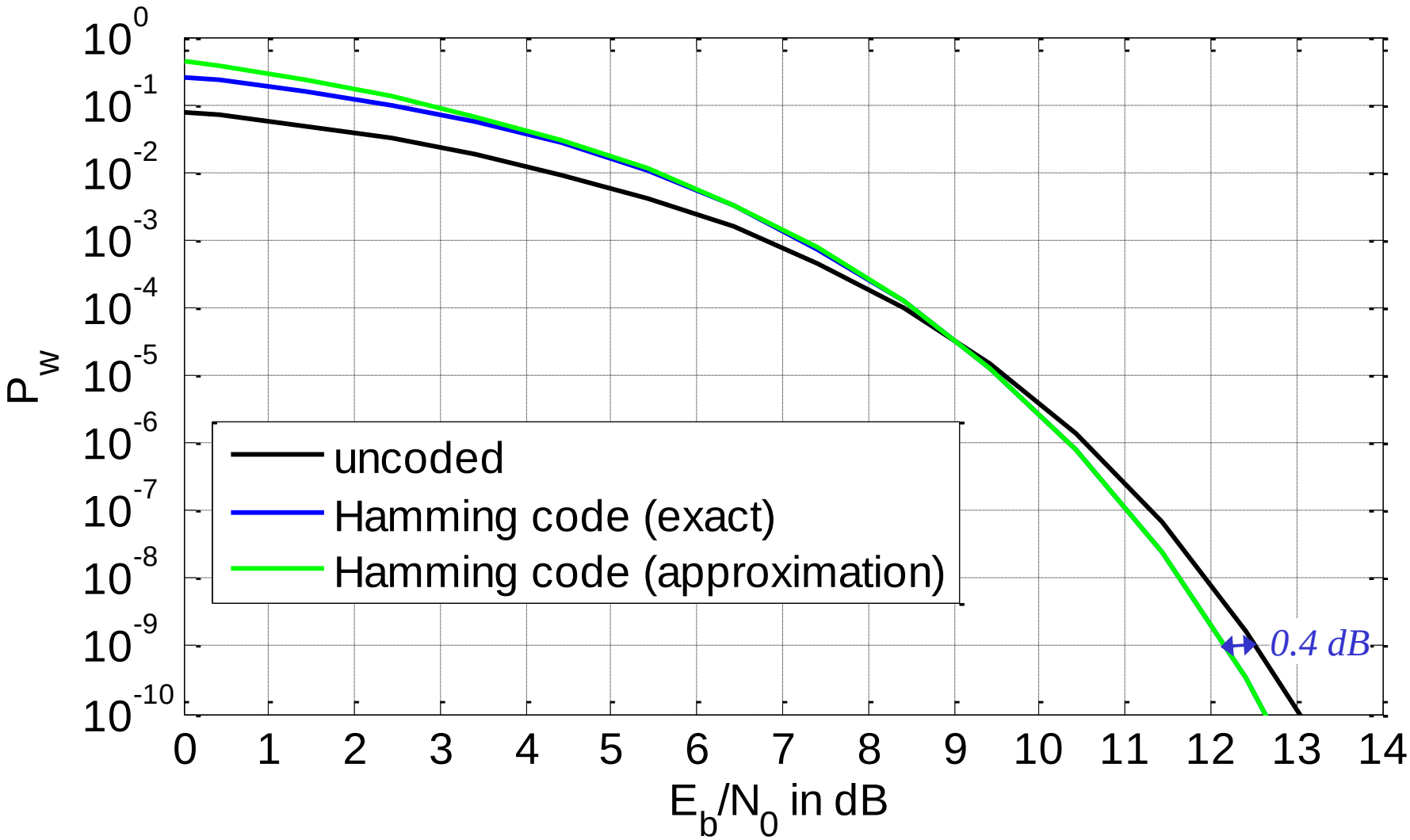
BER Performance of Repetition Codes- Analytical vs. Simulation Results

Rate R binary repetition codes, AWGN channel, hard decision decoding



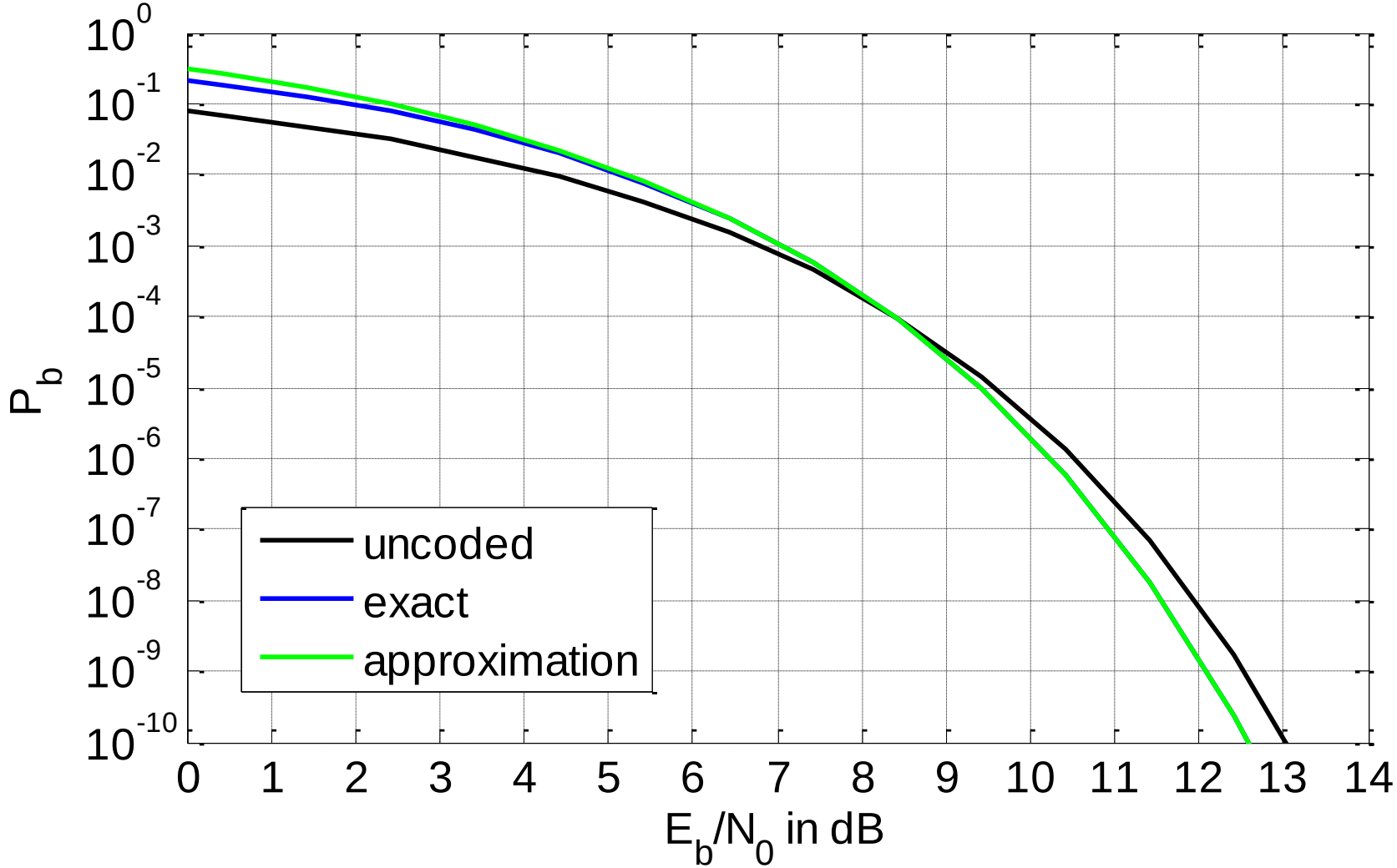
Performance of (7,4,3)-Hamming Code with Hard Decision Decoding in AWGN Channel

Word error probability P_w



Performance of (7,4,3)-Hamming Code with Hard Decision Decoding in AWGN Channel

Bit error probability P_b



Error Rate Analysis: Hard Decision Decoding
Word Error Rate and Bit Error Rate

The error probability depends on both the code properties and the channel properties. The word error probability of a perfect linear $(N, K, d_{\min})$ block code with hard decision decoding (BSC channel) can be computed exactly. The computation of the bit error probability (bit error rate, BER) is more difficult and is usually approximated.

A typical behaviour of the error rate performance of a code is that the performance is worse than uncoded transmission at very low SNR, where the channel error rate is very high and the code constraints cause additional errors. For medium and high SNR, coded transmission outperforms uncoded transmission.

Matrix Description of Linear Block Codes: Generator Matrix

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{11} & \cdots & g_{1K} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NK} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Generator matrix $\mathbf{G}$:

- columns should be linear independent
  → columns of $\mathbf{G}$ are a basis of the code $\mathcal{C}$ with dimension $\dim(\mathcal{C}) = K$.
  → all codewords are obtained as linear combinations of the columns of $\mathbf{G}$.

Matrix Description of Linear Block Codes: Generator Matrix

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{11} & \cdots & g_{1K} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NK} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

The same code $\mathcal{C}$ is generated if

- columns are exchanged
- a column is multiplied by a scalar unequal zero.
- a scaled column is added to another column.

Matrix Description of Systematic Linear Block Codes: Generator Matrix

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{11} & \cdots & g_{1K} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NK} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Systematic encoder:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 1 \\ p_{11} & \cdots & \cdots & p_{1K} \\ \vdots & \ddots & \ddots & \vdots \\ p_{N-K,1} & \cdots & \cdots & p_{N-K,K} \end{bmatrix}$$

Matrix Description of Linear Block Codes: Example: Rate *1/N* Repetition Code

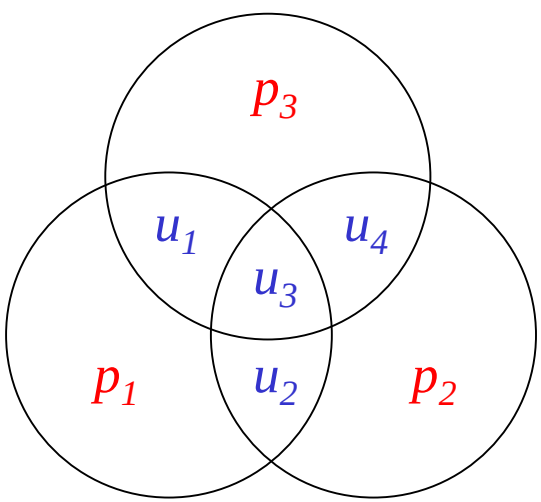
K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = [u_1] \qquad \mathbf{G} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_1 \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Matrix Description of Linear Block Codes: Example: (7,4,3) Hamming Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \mathbf{Gu}$$



Matrix Description of Linear Block Codes: Example: (7,4,3) Hamming Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{x} = \mathbf{G}\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \Rightarrow \mathbf{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} u_3 \\ u_2 \\ u_1 \\ u_4 \\ p_1' \\ p_2' \\ p_3' \end{bmatrix}$$

- identical code $\mathcal{C}$ (same set of codewords).
- different encoder (other mapping of info words to codewords), roles of u_1 and u_3 are exchanged.

Matrix Description of Linear Block Codes: Example: (7,4,3) Hamming Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{x} = \mathbf{G}\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \Rightarrow \mathbf{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_3 \\ u_2 \\ u_1 \\ u_4 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

- identical code $\mathcal{C}$ (same set of codewords).
- different encoder (other mapping of info words to codewords), roles of u_1 and u_3 are exchanged.

Matrix Description of Linear Block Codes: Parity Check Matrix

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix}$$

Generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} g_{11} & \cdots & g_{1K} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NK} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

Syndrome $\mathbf{s}$:

$\mathbf{x} \in \mathcal{C}$:

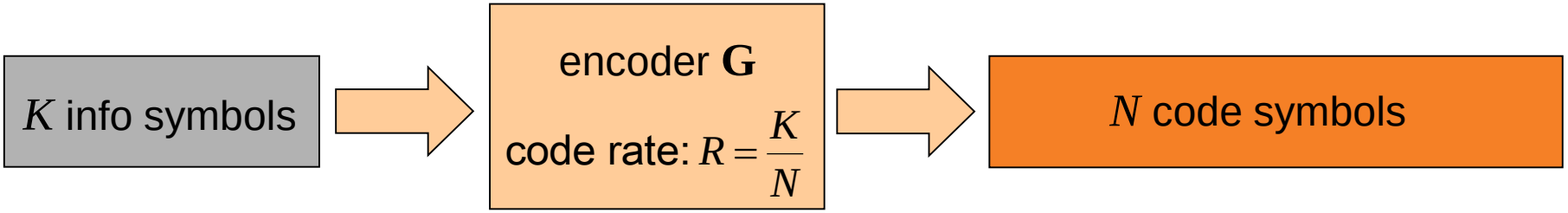
$$\mathbf{s} = \mathbf{H}^T\mathbf{x} = \mathbf{0}, \qquad \mathbf{H}^T \neq \mathbf{0}$$

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

$\mathbf{e} \notin \mathcal{C}$:

$$\mathbf{s} = \mathbf{H}^T\mathbf{e} \neq \mathbf{0}$$

Matrix Description of Systematic Linear Block Codes: Parity Check Matrix



Systematic encoder

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix}$$

Generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P}_{N-K\times K} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H}^T = \left[-\mathbf{P}_{N-K\times K}, \mathbf{I}_{N-K}\right]$$

$$\mathbf{H}^T\mathbf{G} = \left[-\mathbf{P}_{N-K\times K}, \mathbf{I}_{N-K}\right]\begin{bmatrix} \mathbf{I}_K \\ \mathbf{P}_{N-K\times K} \end{bmatrix} = \left[-\mathbf{P}_{N-K\times K}\mathbf{I}_K + \mathbf{I}_{N-K}\mathbf{P}_{N-K\times K}\right]$$

$$= \left[-\mathbf{P}_{N-K\times K} + \mathbf{P}_{N-K\times K}\right] = \mathbf{0}$$

Matrix Description of Linear Block Codes: Properties of Parity Check Matrix

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix},$$

$$\mathbf{x} \in \mathcal{C} \Rightarrow \mathbf{H}^T\mathbf{x} = \mathbf{0}$$

$$\mathbf{e} \notin \mathcal{C} \Rightarrow \mathbf{H}^T\mathbf{e} \neq \mathbf{0}$$

$$\operatorname{rank} \mathbf{H} = N - K$$

Example: Rate 1/3 repetition code

$$\mathbf{G} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

$$\mathcal{C} = \{\mathbf{x}_1, \mathbf{x}_2\} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

$$\operatorname{rank} \mathbf{H} = N - K$$

$$\mathbf{H}^T\mathbf{x}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{H}^T\mathbf{x}_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{H}^T\mathbf{e} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\operatorname{rank} \tilde{\mathbf{H}} < N - K$$

$$\tilde{\mathbf{H}}^T\mathbf{x}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\tilde{\mathbf{H}}^T\mathbf{x}_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\tilde{\mathbf{H}}^T\mathbf{e} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$\Rightarrow \tilde{\mathbf{H}}$ is not a parity check matrix

Matrix Description of Linear Block Codes: Properties of Parity Check Matrix

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

$$\mathbf{x} \in \mathcal{C} \Rightarrow \mathbf{H}^T \mathbf{x} = \mathbf{0}$$

$$\mathbf{e} \notin \mathcal{C} \Rightarrow \mathbf{H}^T \mathbf{e} \neq \mathbf{0}$$

What is the relation of the minimum Hamming distance $d_{\min}$ and $\mathbf{H}$?

$$\operatorname{rank} \mathbf{H} = N - K \geq d_{\min} - 1$$

Singleton bound: $d_{\min} - 1 \leq N - K$

Elementary permutations are allowed for columns of $\mathbf{H}$ (rows of $\mathbf{H}^T$)

but not for rows of $\mathbf{H}$ (columns of $\mathbf{H}^T$)

- exchange of columns.
- multiplication of a column with a scalar unequal zero.
- addition of a scaled column to another column.

Matrix Description of Linear Block Codes - Examples: Repetition Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

Generator matrix $\mathbf{G}$:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,K} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,K} \end{bmatrix}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

$$\mathbf{x} = \mathbf{G}\mathbf{u}$$

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

(*N,1*) Repetition code:

$$\mathbf{G} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\mathbf{H}^T = \left[\mathbf{1}_{N-1\times 1} \quad \mathbf{I}_{N-1}\right] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: $N=3$:

$$\mathbf{H}^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad N-K=2, \qquad d_{\min} = 3$$

$$\text{rank}\,\mathbf{H} = N-K = d_{\min} - 1 = 2$$

exchange rows

Permutations of columns of $\mathbf{H}$ (rows of $\mathbf{H}^T$):

row 1+row2

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{H}^T = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{H}^T = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

row 1+row2

Matrix Description of Linear Block Codes - Examples: Repetition Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

Generator matrix $\mathbf{G}$:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,K} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,K} \end{bmatrix}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

$$\mathbf{x} = \mathbf{G}\mathbf{u}$$

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

(*N*,*1*) Repetition code:

$$\mathbf{G} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\mathbf{H}^T = \left[\mathbf{1}_{N-1\times 1} \quad \mathbf{I}_{N-1}\right] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: $N=3$:

Permutations of columns of $\mathbf{H}^T$ are not allowed !

column 2+3

$$\mathbf{H}^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{H}^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{H}^T\mathbf{x}_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \neq \mathbf{0}$$

Matrix Description of Linear Block Codes - Examples: Repetition Code and Single Parity Check Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

Generator matrix $\mathbf{G}$:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,K} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,K} \end{bmatrix}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

$$\mathbf{x} = \mathbf{G}\mathbf{u}$$

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

(*N*,*1*) Repetition code:

$$\mathbf{G} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{1}_{N-1\times 1} & \mathbf{I}_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(*N*, *N-1*) Single parity check code:

$$\mathbf{G} = \begin{bmatrix} \mathbf{1}_{1\times N-1} \\ \mathbf{I}_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}^T = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}$$

Matrix Description of Linear Block Codes - Examples: Single Parity Check Code

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

Generator matrix $\mathbf{G}$:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,K} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,K} \end{bmatrix}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,N-K} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,N-K} \end{bmatrix}$$

$$\mathbf{x} = \mathbf{G}\mathbf{u}$$

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

Example: $N=3$:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow \mathbf{x} = \mathbf{G}\mathbf{u} = \begin{bmatrix} p \\ u_1 \\ u_2 \end{bmatrix}$$

column 1+2

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow \mathbf{x} = \mathbf{G}\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ p \end{bmatrix}$$

(*N*, *N-1*) Single parity check code:

$$\mathbf{G} = \begin{bmatrix} \mathbf{1}_{1\times N-1} \\ \mathbf{I}_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}^T = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}$$

Matrix Description of Linear Block Codes: Generator Matrix and Parity Check Matrix

Simple codes can be described by simply listing their codewords and the mapping rule of information words to codewords. However, more sophisticated codes of larger block size need a more efficient mathematical description. Moreover, an algebraic structure is desired for computationally efficient implementation of encoder and decoder.

A suitable description of linear block codes is obtained using matrices. A code can be defined by its generator matrix $\mathbf{G}$ or by its parity check matrix $\mathbf{H}$. $\mathbf{G}$ and $\mathbf{H}$ can be derived from each other.

The columns of $\mathbf{G}$ form a basis for the code $\mathcal{C}$ of dimension dim($\mathcal{C}$)=K. The code $\mathcal{C}$ is the null space $\mathbf{H}$.

Elementary permutations of the columns in $\mathbf{G}$ (exchanging columns, multiplication of columns with a scalar unequal zero, addition of scaled columns) generate the same code $\mathcal{C}$ (i.e. the same set of codewords), however, with a different encoder, i.e. a different mapping of info words to code words. By applying those elementary permutations, it is always possible to obtain a systematic encoder, i.e. a generator matrix $\mathbf{G}$ which contains the identity matrix $\mathbf{I}_K$ as submatrix.

Similar as for the generator matrix $\mathbf{G}$, the parity check matrix $\mathbf{H}$ of a code is not uniquely defined. The elementary permutations are allowed for the columns of $\mathbf{H}$ (but not of $\mathbf{H}^T$) such that different parity check matrices $\mathbf{H}$ can be applied to a particular code $\mathcal{C}$. .

The parity check matrix $\mathbf{H}$ has full rank N-K.

Low Density Parity Check (LDPC) Codes

Low density parity check (LDPC) codes are block codes with sparse parity check matrices $\mathbf{H}$, i.e. the parity check matrix contains only a small number of non-zero entries. The sparseness of the parity check matrix is desired because it means that a low number of parity checks have to be performed in the decoder. Hence, the decoding complexity remains manageable. In particular, the sparseness of the parity check matrix can be exploited in an iterative decoding algorithm called message passing. This decoding algorithm is based on a so-called factor graph representation of the code. For iterative message passing, the sparseness of the parity check matrix is not only desired for complexity reasons but also, the iterative message passing algorithm will not work well for non sparse parity check matrices due to correlations of the exchanged probabilities.

The decoding complexity of message passing increases only linearly with the code word length. This allows to construct block codes with long code word length which is necessary in order to achieve performance close to the Shannon capacity limit. In contrast, classical block codes are typically short, because the usually applied maximum likelihood decoding becomes prohibitively complex for long code word length.

A parity check matrix is called *regular* if each row contains the same number d_r of non-zero entries and each column contains the same number d_c of non-zero entries. Otherwise, the parity check matrix is called *irregular*. For a regular code, each code bit is involved in d_r parity checks and each parity check equation contains d_c code bits.

Tanner Graphs (1)

A parity check matrix of an LDPC code is often represented by a Tanner graph. A Tanner graph consists of two types of nodes: bit nodes (or variable nodes) and check nodes.

Each bit node represents a codebit. Hence, there are N bit nodes in the Tanner graph. Each check node represents a parity check equation. The number of check nodes is equal to the number of rows in $\mathbf{H}^T$. Typically, this will be equal to the rank of the parity check matrix $\mathbf{H}$ and in most cases rank $\mathbf{H} = N\text{-}K$, which is the number of linear independent parity check equations. It is possible to have more than rank $\mathbf{H}$ parity check equations. However, at most $N\text{-}K$ parity check equations can be linear independent.

A bit node is connected to a check node by an edge, if the code bit represented by that bit node is involved in the respective parity check equation. Hence, the edges of the Tanner graph indicate, which code bits are involved in which parity check equations.

The degree (or weight) of a node is given by the number of edges which are connected to the node. We distinguish between bit node degree and check node degree.

The degree of a bit node is given by the number of non-zero entries in the corresponding column of $\mathbf{H}^T$. The degree of a check node is given by the number of non-zero entries in the corresponding row of $\mathbf{H}^T$.

For an irregular parity check matrix, we denote the fraction of bit nodes with degree i by $d_{b,i}$ and the fraction of check nodes with degree j by $d_{c,j}$. The degree distribution of bit nodes and check nodes, respectively, is then defined by the vectors $d_b = [d_{b,1}, d_{b,2}, d_{b,3}, d_{b,4}, \ldots]$ and $d_c = [d_{c,1}, d_{c,2}, d_{c,3}, d_{c,4}, \ldots]$.

Tanner Graphs (2)

In case of a full rank parity check matrix of dimension $N \times N\text{-}K$, the code rate is determined by the degree distribution:

$$R = \frac{K}{N} = \frac{N-(N-K)}{N} = 1 - \frac{\sum_i d_{b,i} i}{\sum_j d_{c,j} j}.$$

For regular codes, this simplifies to

$$R = 1 - \frac{d_b}{d_c}.$$

In most cases, the parity check matrix is full rank or close to it. For randomly designed large parity check matrices, the rank is sometimes not explicitly measured. In this case, R according to the equation above is called the *design code rate*.

For iterative decoding, the length of cycles in the Tanner graph is decisive. A cycle is defined by a sequence of connected nodes which starts and ends in the same node and touches other nodes no more than once. The length of the shortest cycle in the graph measured in the number of edges it contains, is called the *girth* of the graph. For iterative decoding, it is desirable to have a large girth.

In practice, it is most important to construct LDPC codes such that cycles of length four are avoided in their Tanner graph. Some additional performance gain can be achieved, if also cycles of length six are avoided. However, avoiding also longer cycles hardly produces additional benefit.

Low Density Parity Check (LDPC) Codes

K info symbols → encoder $\mathbf{G}$, code rate: $R = \frac{K}{N}$ → N code symbols

A low density parity check (LDPC) code is a block code with sparse parity check matrix $\mathbf{H}$, i.e. $\mathbf{H}$ contains a small fraction of non-zero entries.

$$\mathbf{H}^T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & 1 \\ 0 & 1 & 1 & 0 & \cdots & 1 & 0 & 0 \\ 1 & 0 & & 1 & \cdots & 0 & 0 & 0 \end{bmatrix}$$

Advantage:

- Manageable decoding complexity even for long codeword length N.

  → Iterative message passing decoding based on graph representation of the code.

  → Decoding complexity increases only linearly with N.

Problem:

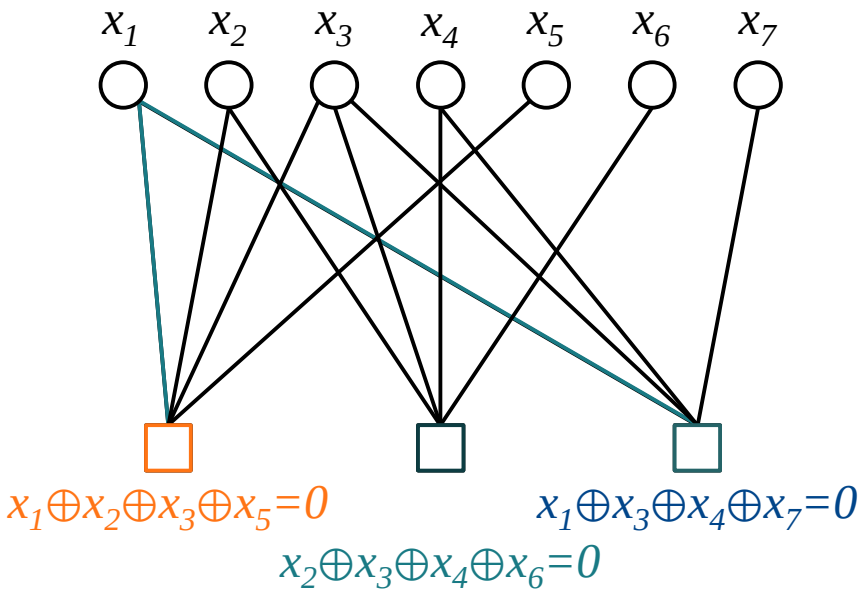
- Generator matrix $\mathbf{G}$ is in general not sparse and unstructured.

  → High encoding complexity in order of N^2.

Tanner Graph

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

(7,4,3) Hamming code



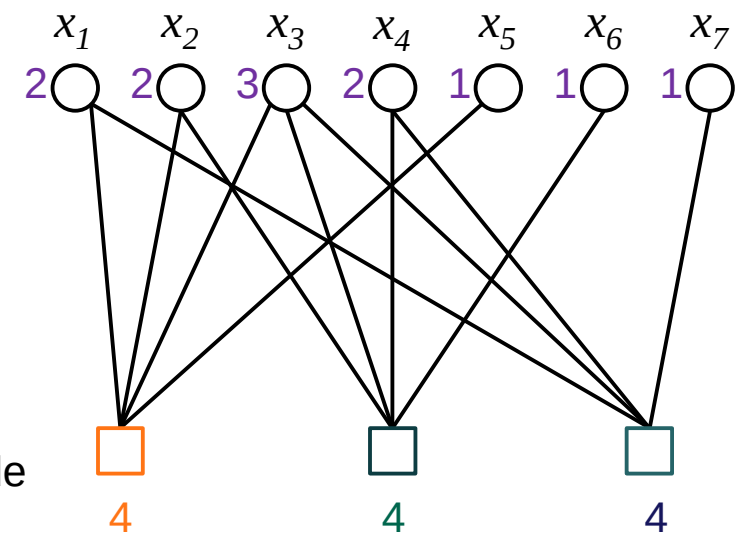
- Bit nodes
  Each bit node represents a code bit.
- Edges indicate which code bits are involved in the parity check equations.
- Check nodes
  Each check node represents a parity check equation.

Tanner Graph: Degree Distribution

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

(7,4,3) Hamming code

Bit node degree:



Check node degree:

Degree distribution:

3 bit nodes with degree $1 \rightarrow d_{b,1}=3/7$

3 bit nodes with degree $2 \rightarrow d_{b,2}=3/7$

1 bit node with degree $3 \rightarrow d_{b,3}=1/7$

$\mathbf{d}_b=[d_{b,1}, d_{b,2}, d_{b,3}, d_{b,4}...] =[3/7, 3/7, 1/7]$

$\rightarrow$ irregular code

3 check nodes with degree $4 \rightarrow d_{c,4}=1$

$\mathbf{d}_c=[d_{c,1}, d_{c,2}, d_{c,3}, d_{c,4}...]= [0,0,0,1]$

$d_c=4$

Degree of a node: number of edges which are connected to the node.

Regular code:
- All bit nodes have the same degree d_b.
- All check nodes have the same degree d_c.

Irregular code:
Bit nodes and/or check nodes have different degree.

Tanner Graph: Degree Distribution and Code Rate

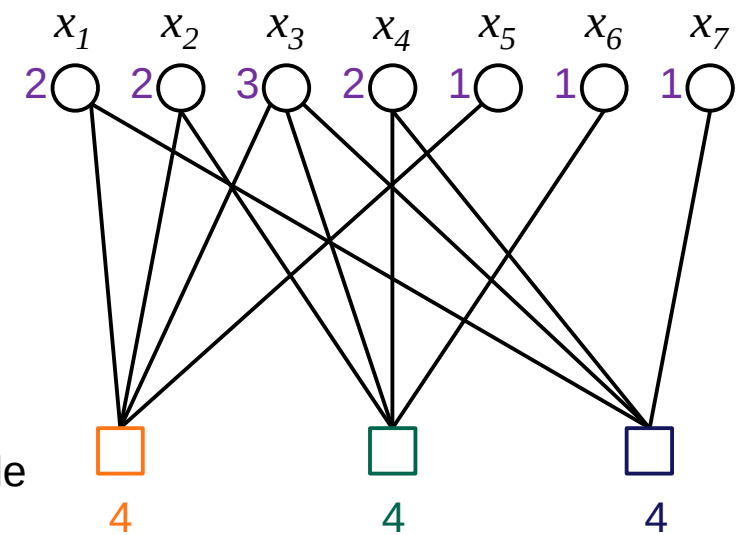
$$\mathbf{H}^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

N

(7,4,3) Hamming code

$P = N\text{-}K$ for full rank $\mathbf{H}$

Bit node degree:



Check node degree:

$$(N-K)\sum_j d_{c,j}\, j = N \sum_i d_{b,i}\, i$$

Degree distribution:

3 bit nodes with degree $1 \rightarrow d_{b,1}=3/7$

3 bit nodes with degree $2 \rightarrow d_{b,2}=3/7$

1 bit node with degree $3 \rightarrow d_{b,3}=1/7$

$\mathbf{d}_b=[d_{b,1}, d_{b,2}, d_{b,3}, d_{b,4}...] =[3/7, 3/7, 1/7]$

$\rightarrow$ irregular code

3 check nodes with degree $4 \rightarrow d_{c,4}=1$

$\mathbf{d}_c=[d_{c,1}, d_{c,2}, d_{c,3}, d_{c,4}...]= [0,0,0,1]$

$d_c=4$

Code rate for full rank parity check matrix $\mathbf{H}$:

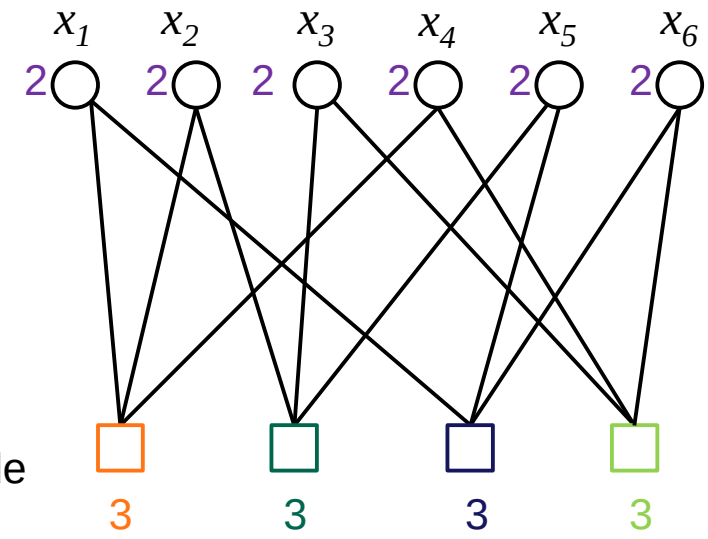
$$R = \frac{K}{N} = \frac{N-(N-K)}{N} \underset{\text{for full rank } \mathbf{H}}{=} 1 - \frac{\sum_i d_{b,i}\, i}{\sum_j d_{c,j}\, j} \underset{\text{for regular codes}}{=} 1 - \frac{d_b}{d_c}$$

Tanner Graph: Regular Code

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

Bit node degree:

$d_b = 2$



Check node degree:

$d_c = 3$

Degree of a node: number of edges which are connected to the node.

Regular code:

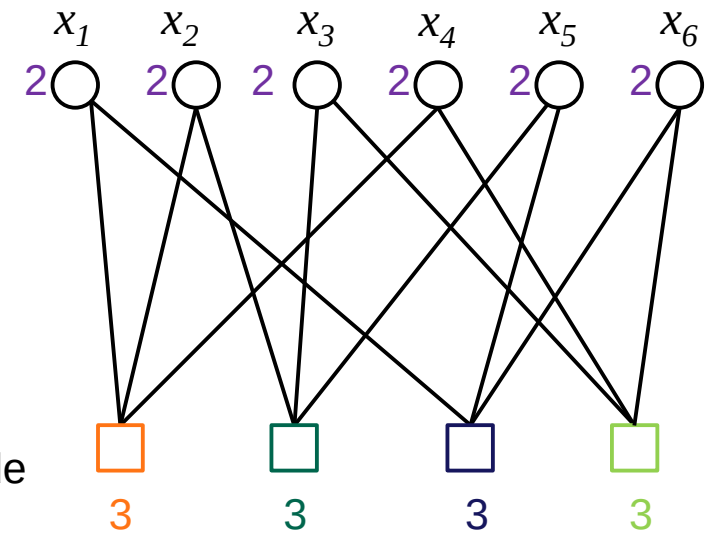
- All bit nodes have the same degree d_b.
- All check nodes have the same degree d_c.

Tanner Graph: Code Rate for Regular Code

$$\mathbf{H}^T = \overset{\longleftarrow\ N\ \longrightarrow}{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}} \quad \updownarrow\ P = N\text{-}K \text{ for full rank } \mathbf{H}$$

Bit node degree:

$d_b = 2$



Check node degree:

$d_c = 3$

Degree of a node: number of edges which are connected to the node.

$$(N\text{-}K)\, d_c = N\, d_b \quad \Rightarrow$$

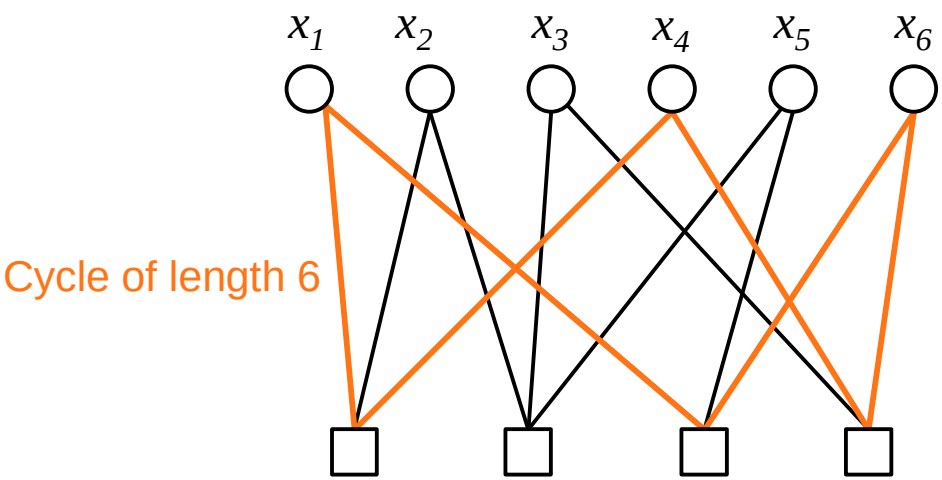
Code rate for full rank parity check matrix $\mathbf{H}$:

$$R = \frac{K}{N} = \frac{N-(N-K)}{N} = 1 - \frac{d_b}{d_c}$$

Tanner Graph: Cycles and Girth

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

A cycle of length 6 can be identified from the structure of the parity check matrix $\mathbf{H}$.



Cycle: Sequence of connected nodes which starts and ends in the same node and touches other nodes no more than once.

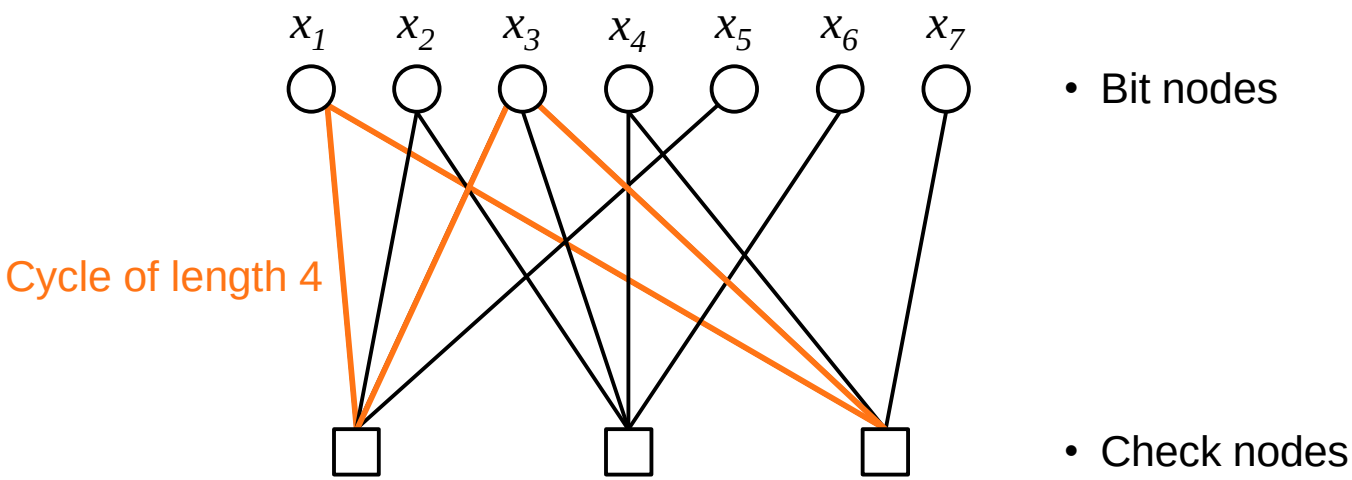
Girth: Length of the shortest cycle in the graph

Tanner Graph: Cycles and Girth

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

(7,4,3) Hamming code

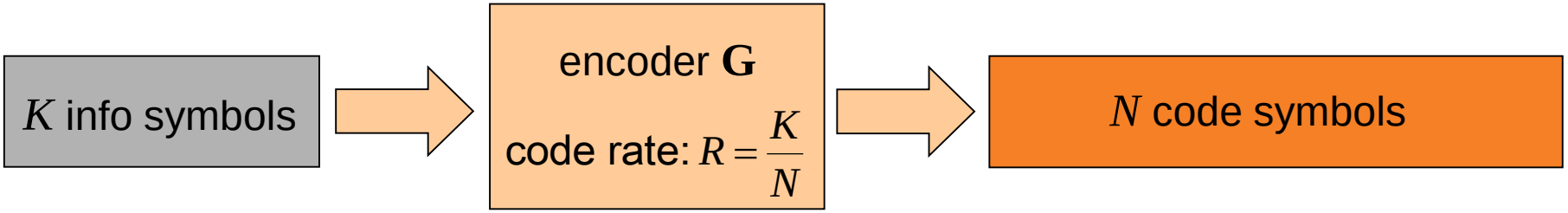
A cycle of length 4 can be identified from the structure of the parity check matrix $\mathbf{H}$.



Cycle: Sequence of connected nodes which starts and ends in the same node and touches other nodes no more than once.

Girth: Length of the shortest cycle in the graph

Encoding of LDPC Codes using Generator Matrix



$$\mathbf{x} = \mathbf{G}\mathbf{u}$$

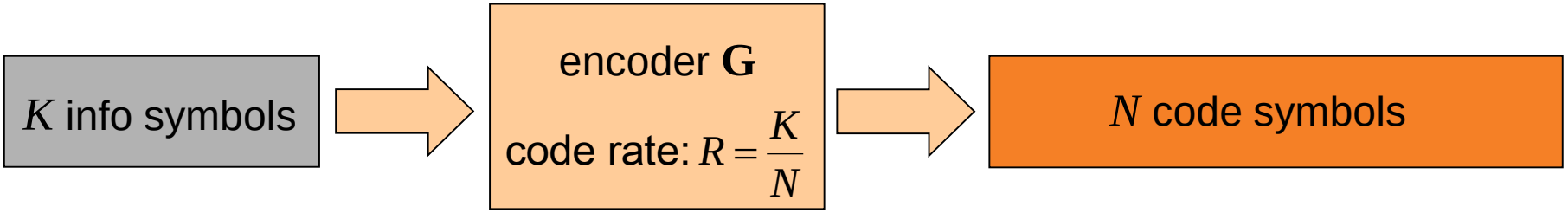
How to obtain a generator matrix $\mathbf{G}$ for a given parity check matrix $\mathbf{H}$?

Bring $\mathbf{H}^T$ into form $\mathbf{H}^T = [\mathbf{P}_{N\text{-}K\times K}, \mathbf{I}_{N\text{-}K}]$ using elementary row permutations.

$$\Rightarrow \quad \mathbf{G} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P}_{N-K\times K} \end{bmatrix}$$

Generator matrix $\mathbf{G}$ will most likely not be sparse.
→ Encoding by matrix multiplication $\mathbf{x} = \mathbf{G}\mathbf{u}$ is computational complex (order of N^2 operations).

Matrix Description of Systematic Linear Block Codes: Parity Check Matrix



Systematic encoder

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_K \end{bmatrix}$$

Generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P}_{N-K\times K} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \mathbf{G}\mathbf{u}$$

Parity check matrix $\mathbf{H}$:

$$\mathbf{H}^T = \left[-\mathbf{P}_{N-K\times K}, \mathbf{I}_{N-K}\right]$$

$$\mathbf{H}^T\mathbf{G} = \left[-\mathbf{P}_{N-K\times K}, \mathbf{I}_{N-K}\right]\begin{bmatrix} \mathbf{I}_K \\ \mathbf{P}_{N-K\times K} \end{bmatrix} = \left[-\mathbf{P}_{N-K\times K}\mathbf{I}_K + \mathbf{I}_{N-K}\mathbf{P}_{N-K\times K}\right]$$

$$= \left[-\mathbf{P}_{N-K\times K} + \mathbf{P}_{N-K\times K}\right] = \mathbf{0}$$

Message Passing Decoding

LDPC codes are usually decoded by a message passing algorithm. The name indicates that in the decoding process, messages are passed along the edges of a Tanner graph. Since the messages are several times passed forth and back between the bit nodes and the check nodes, message passing is a kind of iterative decoding.

There exist different flavours of message passing which are distinguished by the operations performed at the nodes and also by the type of messages which are passed along the edges. In paricular, the passed messages can contain hard (only one bit per message) or soft information (probabilities or log-likelihood ratios, multiple bits per message).

In most cases, message passing decoding is suboptimum, i.e. it is not equivalent to maximum likelihood decoding. However, the manageable complexity makes it an attractive solution. This is particularly important for large codeword length, where maximum likelihood decoding becomes prohibitively complex, whereas the computational complexity of message passing increases only linearly with the codeword length N.

Another important aspect from the implementation point of view is that message passing allows for a a high degree of parallel processing: All operations in the bit nodes can be performed in parallel and all operations in the check nodes can be performed in parallel.

Moreover, performance close to optimum can be achieved using the iterative message passing algorithm.

In graphs without cycles, soft message passing is equivalent to maximum likelihood decoding. However, it turns out that LDPC codes without cycles in their Tanner graph have poor properties, i.e. low minimum Hamming distance. Hence, even though they can be maximum likelihood decoded by message passing, the performance will be poor compared to codes with cyles in the Tanner graph and suboptimum message passing decoding. E.g. a linear code of rate $R \geq ½$ has a minimum Hamming distance of $d_{\min} \leq 2$.

Hard Decision Message Passing Decoding on the Binary Erasure Channel (1)

The principle of message passing is best illustrated for decoding of a received word which was transmitted through a binary erasure channel, i.e. the received bits are either correct or erased. The messages exchanged via the edges of the Tanner graph are either 0, 1 or e (for erasure). In the beginning, the bit nodes are initialized with the values received via the channel. I.e., the bit node i is initialized with the value $y_i \in \{0,1,e\}$ which is received for the i-th code bit x_i. Each bit node passes the received bit value to all check nodes to which it is connected.
Each check node tests whether its parity check equation is satisfied and tries to correct erasures, if possible. The value of an erased bit can be determined at the check node, as long as only one bit involved in the respective parity check equation is erased.
Each check node passes back a different message to each bit node it is connected to. The messages to be passed back from the check nodes to the bit nodes are determined as follows:

- Case 1: No erasures received at the check node: The same message as was received by the check node from bit node i is passed back to bit node i (all bits are correct since the binary erasure channel produces no bit-flipping).
- Case 2: 1 erasure is received by a check node: The check node determines the correct bit for the erased bit from all other bits (which have been received error-free) using the parity check equation. It passes back this corrected value to the bit node which was in erasure. The check node passes back the message „erasure“ to all other bit nodes since it cannot produce an updated information for those bits due to the erasure in the parity check equation.
- Case 3: More than 1 erasure is received by the check node: The check node cannot correct more than 1 erasure and, hence, passes back the message „erasure“ to all bit nodes.

Hard Decision Message Passing Decoding on the Binary Erasure Channel (2)

The bit node i collects the extrinsic information which it receives from all check nodes to which it is connected. If the majority of the messages received from the check nodes indicate a different bit value than its current value, the bit node flips the current value. The updated values of all bit nodes are then again passed to the check nodes. This iterative process is continued until a predefined maximum number of iterations has been performed. Alternatively, a stopping criterion can terminate the iterative process. A possible stopping criterion is that the parity check equations at all check nodes are satisfied. In this case, a valid codeword has been found and further iterations will not change the decoding result anymore. Note, that a situation can occur, where the message passing algorithm does not converge to a valid codeword.

If a bit node which was in erasure receives a value 0 or 1 from a check node, it changes its current value to the value indicated by the message from the check node. Since no bit errors but only erasures can occur, whenever a check node sends back a message other than „erasure," this message is the correct value of the respective bit.
The updated values of all bit nodes are then again passed to the check nodes. This iterative process is continued until a predefined maximum number of iterations has been performed. Alternatively, a stopping criterion can terminate the iterative process. A possible stopping criterion is that all passed messages are 0 or 1 and, consequently, all erasures have been corrected. In this case, a valid codeword has been found and further iterations will not change the decoding result anymore.

Hard Decision Message Passing Decoding on the Binary Erasure Channel (3)

Message passing based on erasures can also be used for systematic encoding: The bit nodes which correspond to the information bits (systematic bits) are initialized with the information bits. The bit nodes corresponding to the parity bits are initialized with „erasure.“ The message passing algorithm described above will determine the parity bits. This encoding method does not require a matrix multiplication and can be used for encoding with a computational complexity which increases linearly with the codeoword length N rather than with N^2 as a multiplication with a generator matrix in general does.

Message Passing Decoding: Binary Erasure Channel (BEC)

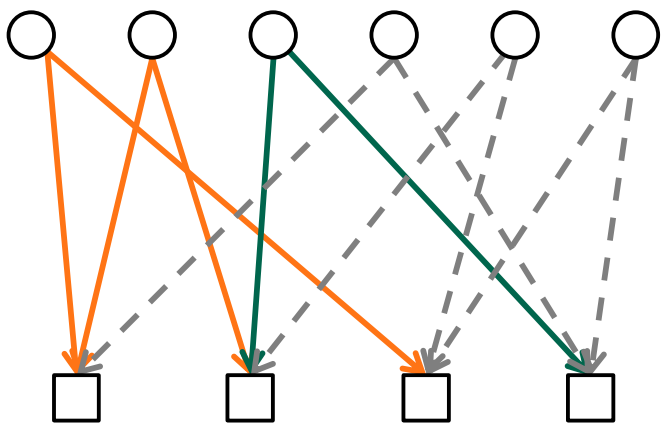
Transmitted codeword $\mathbf{x}$:

$$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$$

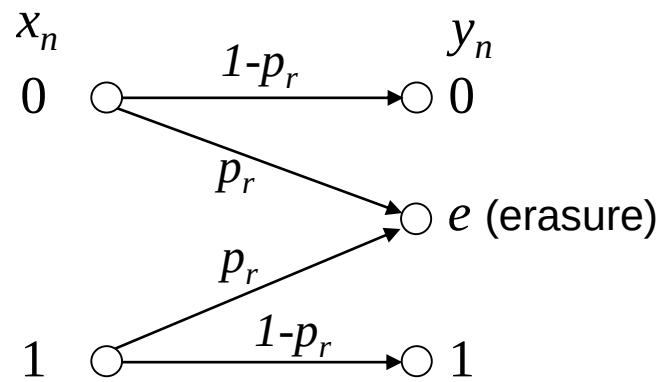
$\Downarrow$ BEC

Received word $\mathbf{y}$:

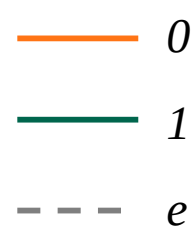
$$0 \quad 0 \quad 1 \quad e \quad e \quad e$$



$y_1 \oplus y_2 \oplus y_4 = 0$



Passed messages:



Determine messages to be passed back:

$y_1 = y_2 \oplus y_4$

$0 \quad e \quad \rightarrow \quad y_1 = e$ (message update rule)

$y_2 = y_1 \oplus y_4$

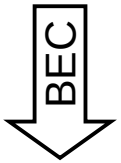
$0 \quad e \quad \rightarrow \quad y_2 = e$

$y_4 = y_1 \oplus y_2$

$0 \quad 0 \quad \rightarrow \quad y_4 = 0$

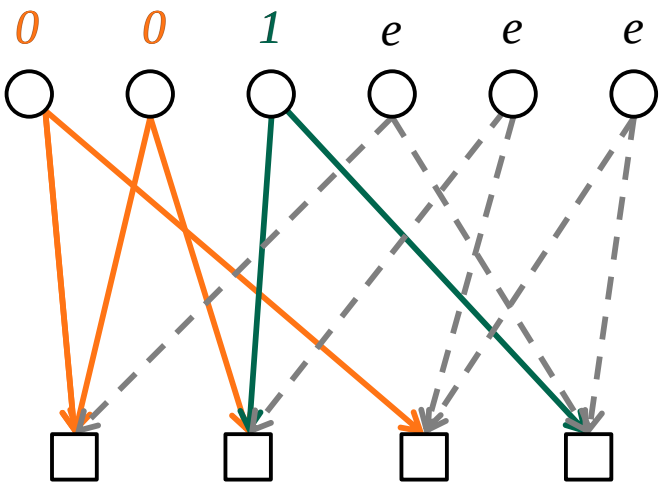
Message Passing Decoding: Binary Erasure Channel (BEC)

Transmitted codeword $\mathbf{x}$:

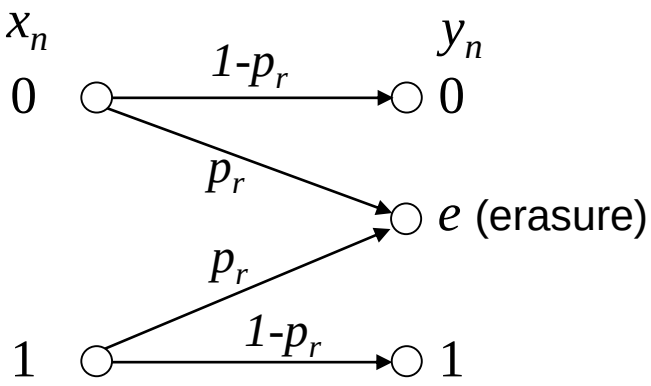


$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

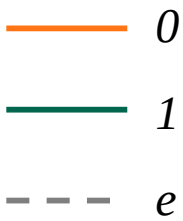
Received word $\mathbf{y}$:



$y_2 \oplus y_3 \oplus y_5 = 0$



Passed messages:



Determine messages to be passed back:

$y_2 = y_3 \oplus y_5$

$1 \quad e \quad \rightarrow \quad y_2 = e$

$y_3 = y_2 \oplus y_5$

$1 \quad e \quad \rightarrow \quad y_3 = e$

$y_5 = y_2 \oplus y_3$

$0 \quad 1 \quad \rightarrow \quad y_5 = 1$

Message Passing Decoding: Binary Erasure Channel (BEC)

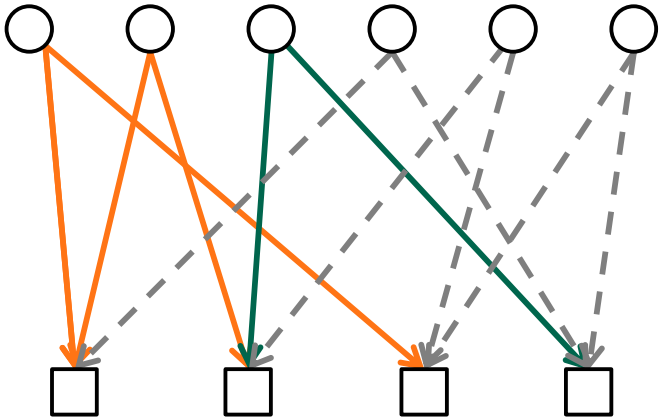
Transmitted codeword $\mathbf{x}$:

BEC

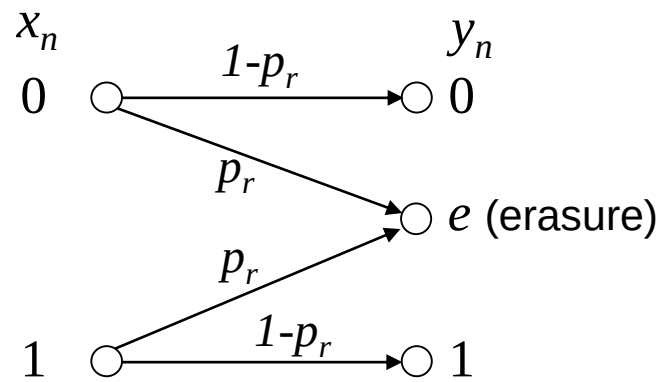
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

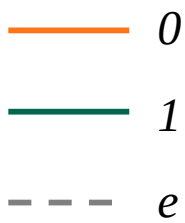
$0 \quad 0 \quad 1 \quad e \quad e \quad e$



$y_1 \oplus y_5 \oplus y_6 = 0$



Passed messages:



Determine messages to be passed back:

$y_1 = y_5 \oplus y_6$

$e \quad e \quad \rightarrow \quad y_1 = e$

$y_5 = y_1 \oplus y_6$

$0 \quad e \quad \rightarrow \quad y_5 = e$

$y_6 = y_1 \oplus y_5$

$0 \quad e \quad \rightarrow \quad y_6 = e$

Message Passing Decoding: Binary Erasure Channel (BEC)

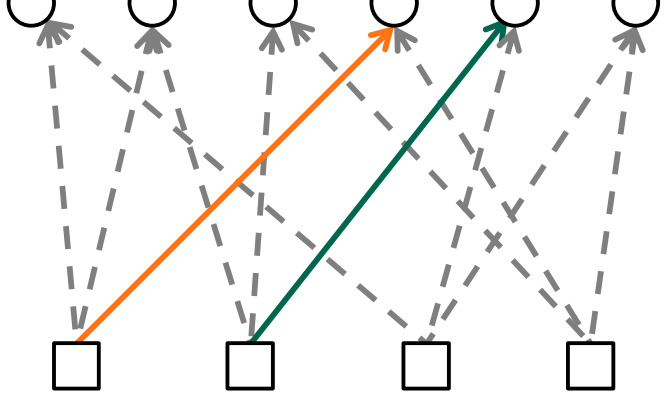
Transmitted codeword $\mathbf{x}$:

$$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$$

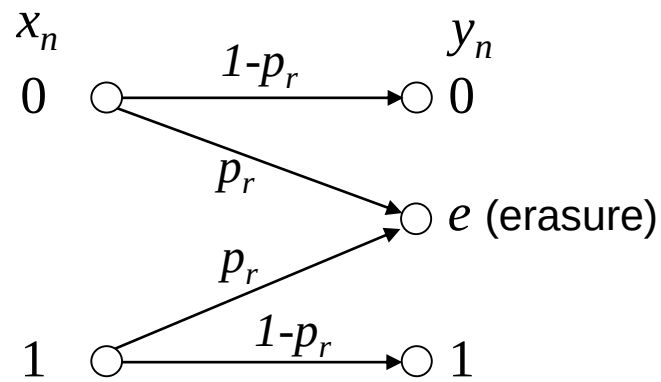
BEC

Received word $\mathbf{y}$:

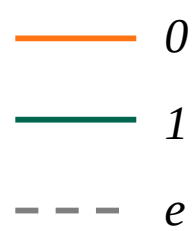
$$0 \quad 0 \quad 1 \quad e \quad e \quad e$$



$y_1 \oplus y_2 \oplus y_4 = 0$



Passed messages:



Determine messages to be passed back:

$y_1 = y_2 \oplus y_4$

$0 \quad e \quad \rightarrow \quad y_1 = e$ (message update rule)

$y_2 = y_1 \oplus y_4$

$0 \quad e \quad \rightarrow \quad y_2 = e$

$y_4 = y_1 \oplus y_2$

$0 \quad 0 \quad \rightarrow \quad y_4 = 0$

Message Passing Decoding: Binary Erasure Channel (BEC)

Transmitted codeword $\mathbf{x}$:

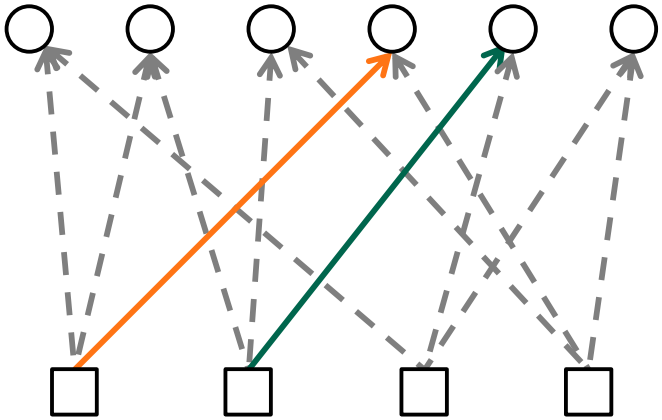
$$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$$

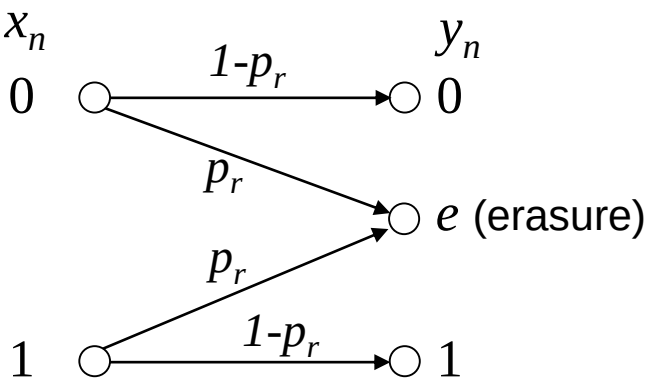
BEC

Received word $\mathbf{y}$:

Update messages if possible.

$$0 \quad 0 \quad 1 \quad e \quad e \quad e$$



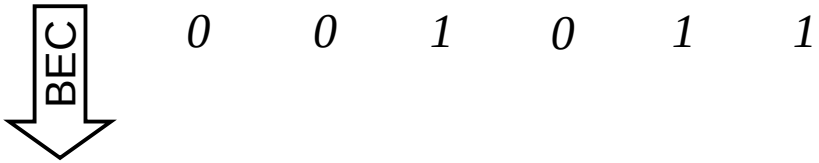


Passed messages:

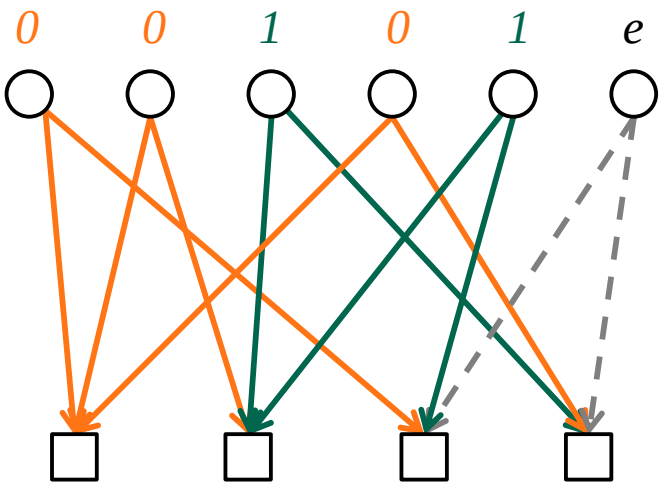
- 0
- 1
- e

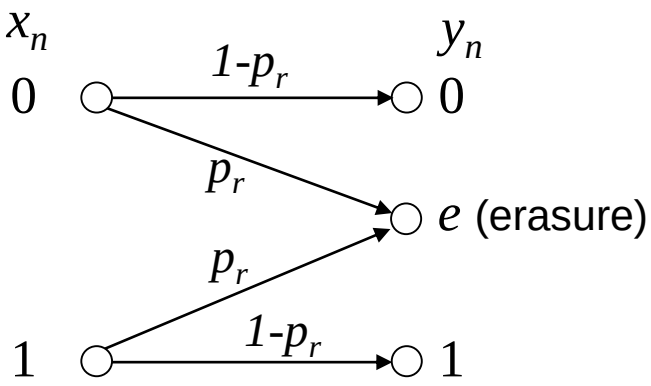
Message Passing Decoding: Binary Erasure Channel (BEC)

Transmitted codeword $\mathbf{x}$:

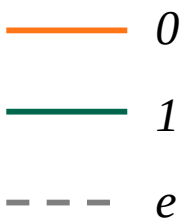


Received word $\mathbf{y}$:





Passed messages:



Message Passing Decoding: Binary Erasure Channel (BEC)

Transmitted codeword $\mathbf{x}$:

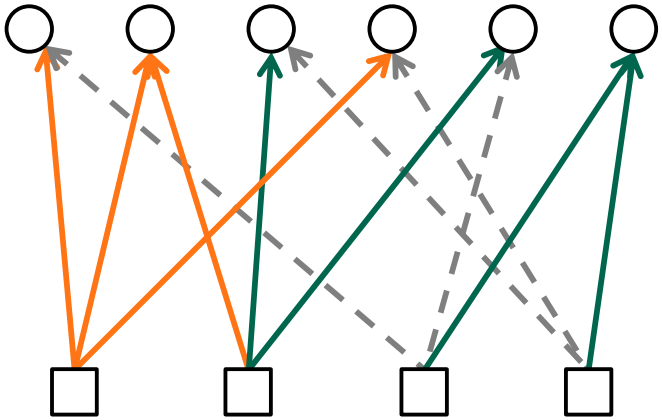
$$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$$

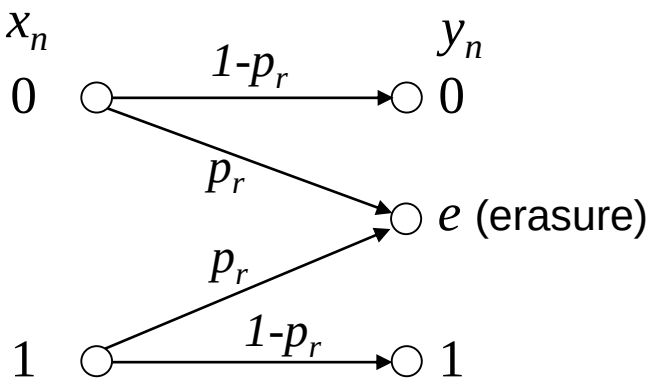
BEC

Received word $\mathbf{y}$:

Update messages if possible.

$$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad e$$





Passed messages:

- 0
- 1
- e

All erasures have been corrected after 2 iterations.

One iteration consists of one complete cycle, i.e. sending messages once from bit nodes to check nodes and back from check nodes to bit nodes.

Message Passing Decoding: Binary Erasure Channel (BEC)

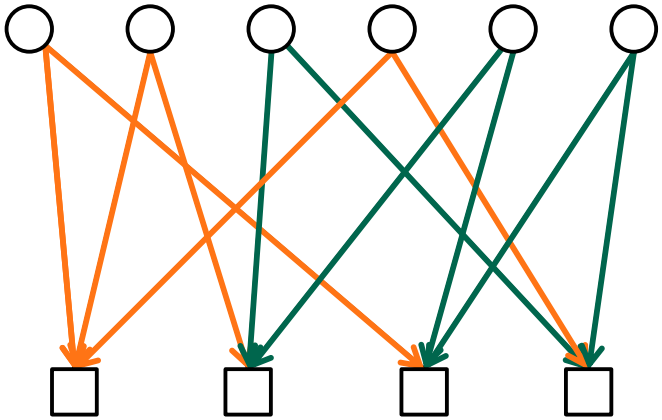
Transmitted codeword $\mathbf{x}$:

BEC

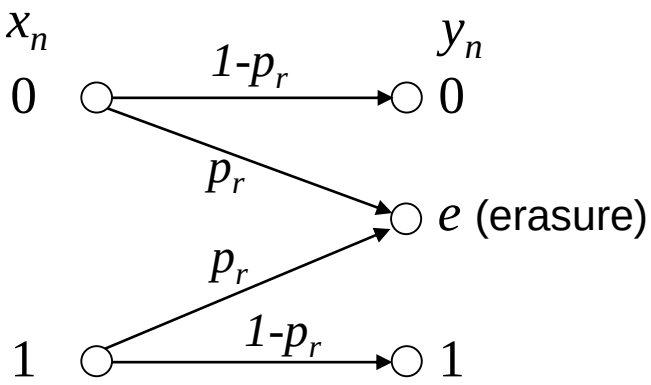
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

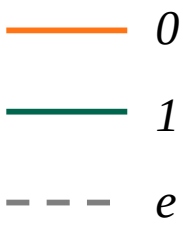
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$



All parity checks are satisfied.
→ a valid codeword is found.



Passed messages:



Message Passing Decoding: Binary Erasure Channel (BEC)

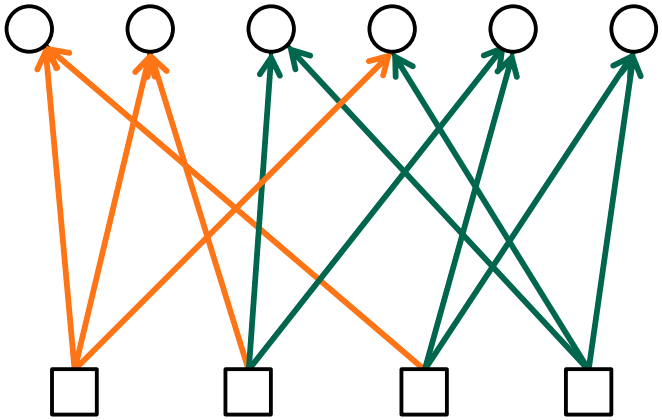
Transmitted codeword $\mathbf{x}$:

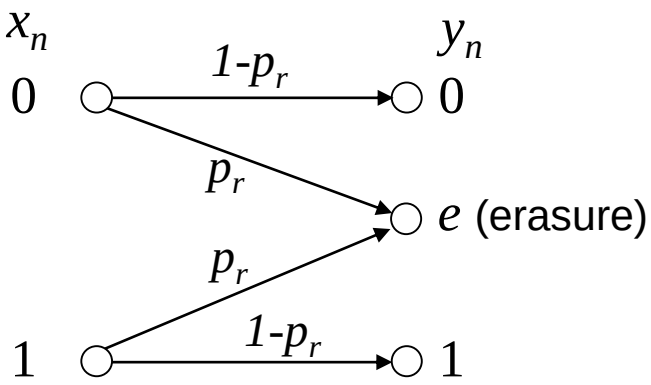
BEC

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

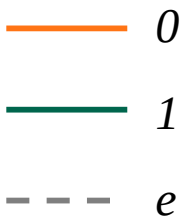
Received word $\mathbf{y}$:

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$





Passed messages:



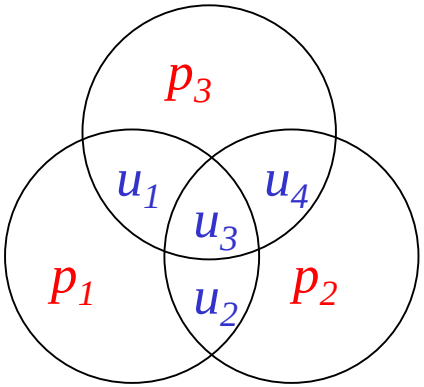
When a codeword is found, furher iterations do not change the decoding result anymore.

The iterative process is stopped when
- a maximum number of iterations has been performed or
- all parity check equations are satisfied.

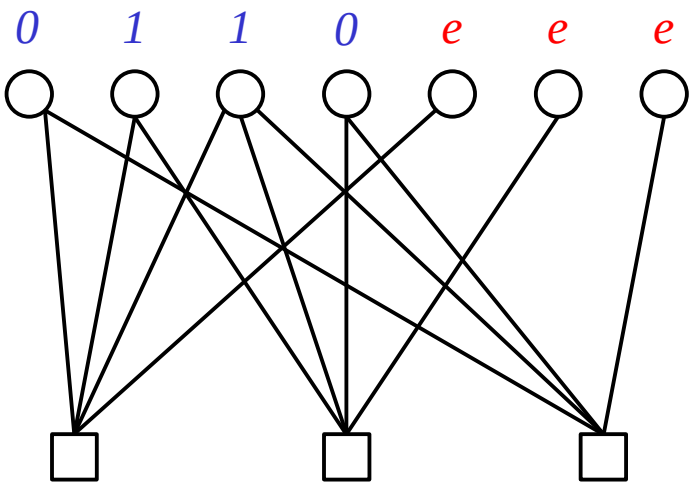
Systematic Encoding Using Erasure Message Passing Decoding

Example: (7,4,3) Hamming code

x:	u_1	u_2	u_3	u_4	p_1	p_2	p_3



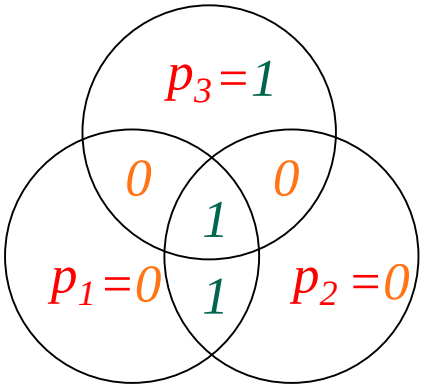
Initialize bit nodes corresponding to systematic bits with info bits and bit nodes corresponding to parity bits with erasures.



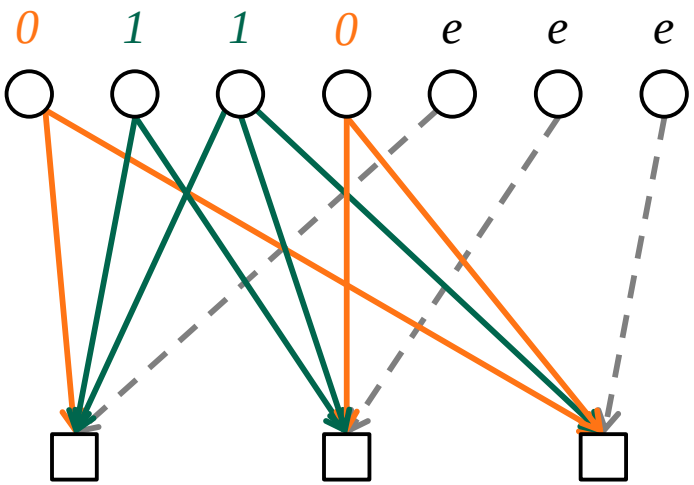
Systematic Encoding Using Erasure Message Passing Decoding

Example: (7,4,3) Hamming code

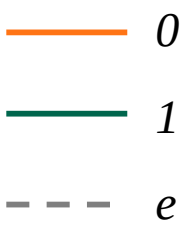
x:	u_1	u_2	u_3	u_4	p_1	p_2	p_3



Initialize bit nodes corresponding to systematic bits with info bits and bit nodes corresponding to parity bits with erasures.



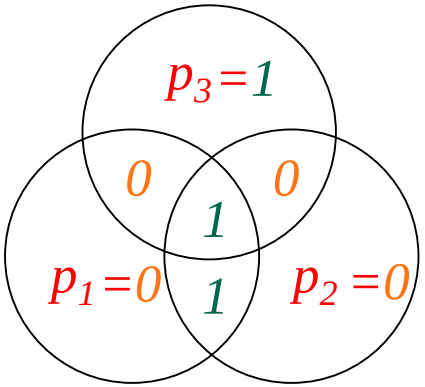
Passed messages:



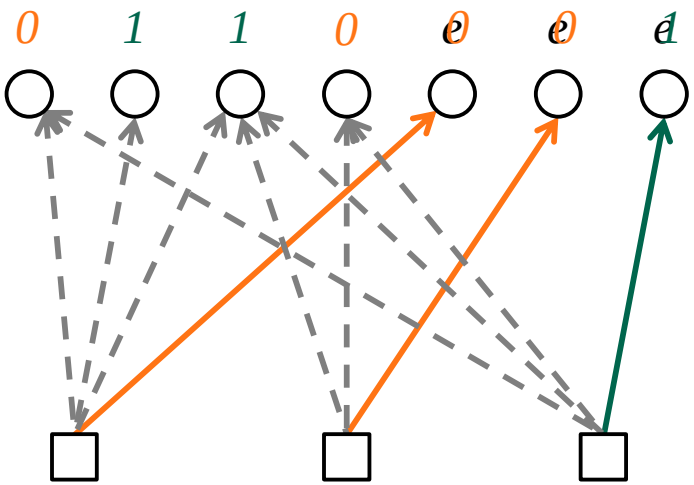
Systematic Encoding Using Erasure Message Passing Decoding

Example: (7,4,3) Hamming code

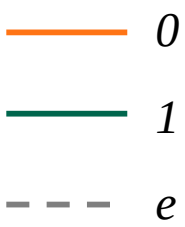
x:	u_1	u_2	u_3	u_4	p_1	p_2	p_3



Initialize bit nodes corresponding to systematic bits with info bits and bit nodes corresponding to parity bits with erasures.



Passed messages:



Hard Decision Message Passing Decoding (Bit-Flipping Decoding) (1)

Bit-flipping decoding refers to hard decision message passing decoding, where the messages exchanged via the edges of the Tanner graph are binary.

In the beginning, the bit nodes are initialized with the values received via the channel. I.e., the bit node i is initialized with the value $y_i \in \{0,1\}$ which is received for the i-th code bit x_i. Each bit node passes the received bit value to all check nodes to which it is connected.

Various possibilities exist regarding the operation in the check nodes in order to determine the binary message which is to be passed back from a check node to the i-th bit node. We first describe a method, where extrinsic information is passed back: We assume that all other bits except for the i-th bit involved in the parity check equation at a check node are correct. The i-th bit is then chosen such that the parity check equation is satisfied. This means that the message on bit i is determined by all other bits which are connected to the same check node but it is independent of the message received by the check node from bit node i. Hence, we pass back *extrinsic information* on bit i. The values provided to the check node and, hence, the extrinsic information calculated by the check node is not necessarily correct.

The bit node i collects the extrinsic information which it receives from all check nodes to which it is connected. If the majority of the messages received from the check nodes indicate a different bit value than its current value, the bit node flips the current value. The updated values of all bit nodes are then again passed to the check nodes. This iterative process is continued until a predefined maximum number of iterations has been performed. Alternatively, a stopping criterion can terminate the iterative process. A possible stopping criterion is that the parity check equations at all check nodes are satisfied. In this case, a valid codeword has been found and further iterations will not change the decoding result anymore. Note, that a situation can occur, where the message passing algorithm does not converge to a valid codeword.

Hard Decision Message Passing Decoding (Bit-Flipping Decoding) (2)

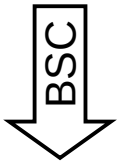
Message passing decoding relies on the assumption that an erroneous received bit will be involved in a number of parity check equations with no other incorrect bits. Since extrinsic information is determined at the check nodes, those correct other bits will help to correct the erroneous bit.

The message passing algorithm only works well for sparse parity check matrices, since sparseness of the parity check matrix reduces the probability that the same code bits are involved in the same parity check equations.

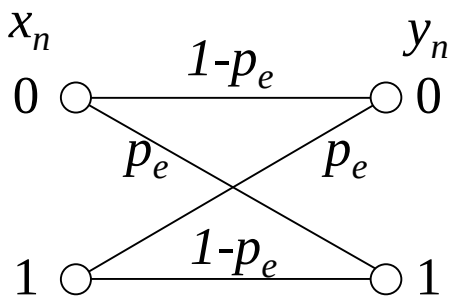
An alternative bit-flipping decoding algorithm could work in the following way: Each check node passes back the same information on all edges to which it is connected. E.g. it sends back 0, if the parity check equation is satisfied and sends back 1 otherwise. This is much simpler than computing extrinsic information since extrinsic information requires a different calculation for each connected edge. The bit node will then flip its decision if a predefined number of connected parity check equations indicate an error.

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

Transmitted codeword $\mathbf{x}$:



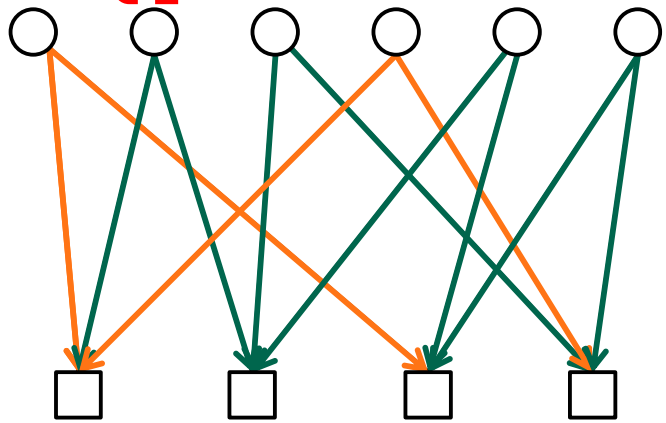
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$



Binary symmetric channel (BSC)

Received word $\mathbf{y}$: error

$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$



$y_1 \oplus y_2 \oplus y_4 = 0$

Passed messages:

Determine messages to be passed back:

$$y_1 = y_2 \oplus y_4 = 1 \oplus 0 = 1$$

$$y_2 = y_1 \oplus y_4 = 0 \oplus 0 = 0$$

$$y_4 = y_1 \oplus y_2 = 0 \oplus 1 = 1$$

Extrinsic information:
Information about bit y_i which is obtained from all other bits involved in the parity check equation.

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

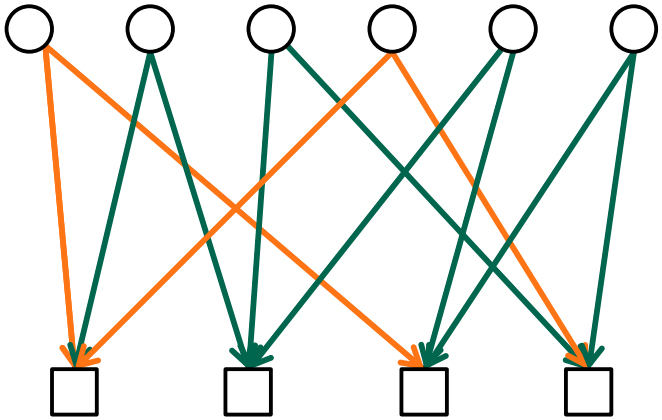
Transmitted codeword $\mathbf{x}$:

BSC

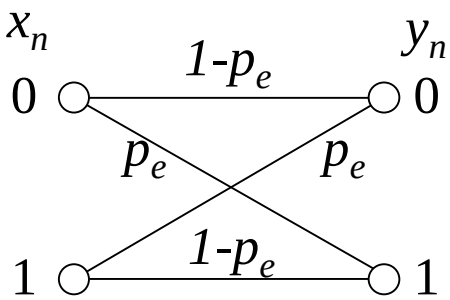
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$



$y_2 \oplus y_3 \oplus y_5 = 0$



Binary symmetric channel (BSC)

Passed messages:

0

1

Determine messages to be passed back:

$$y_2 = y_3 \oplus y_5 = 1 \oplus 1 = 0$$

$$y_3 = y_2 \oplus y_5 = 1 \oplus 1 = 0$$

$$y_5 = y_2 \oplus y_3 = 1 \oplus 1 = 0$$

Extrinsic information

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

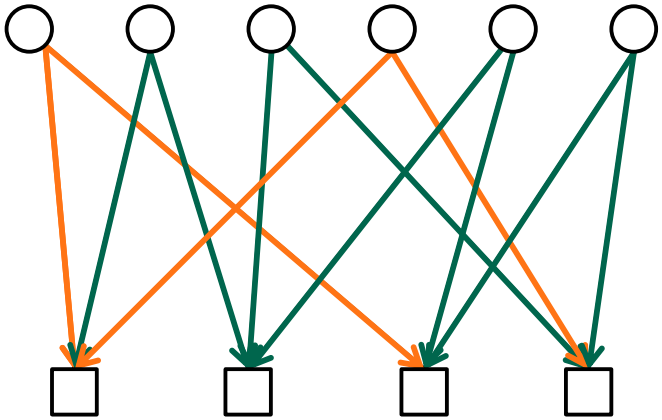
Transmitted codeword $\mathbf{x}$:

BSC

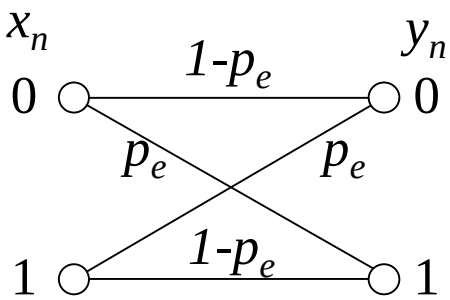
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$



$y_1 \oplus y_5 \oplus y_6 = 0$



Binary symmetric channel (BSC)

Passed messages:

0

1

Determine messages to be passed back:

$$y_1 = y_5 \oplus y_6 = 1 \oplus 1 = 0$$

$$y_5 = y_1 \oplus y_6 = 0 \oplus 1 = 1$$

$$y_6 = y_1 \oplus y_5 = 0 \oplus 1 = 1$$

Extrinsic information

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

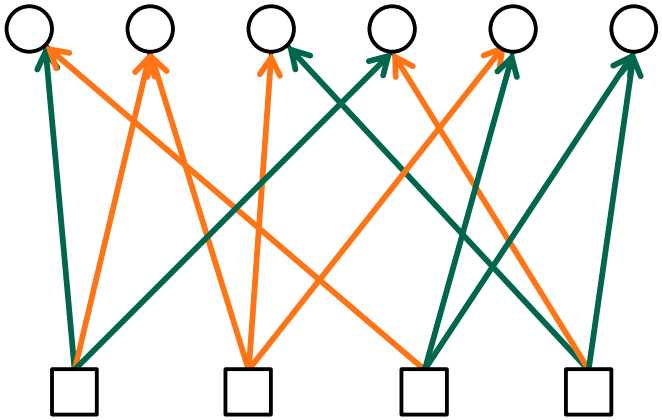
Transmitted codeword $\mathbf{x}$:

BSC

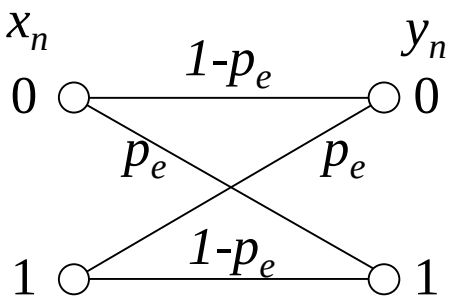
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1$



$y_1 \oplus y_5 \oplus y_6 = 0$



Binary symmetric channel (BSC)

Passed messages:

0

1

Determine messages to be passed back:

$$y_1 = y_5 \oplus y_6 = 1 \oplus 1 = 0$$

$$y_5 = y_1 \oplus y_6 = 0 \oplus 1 = 1$$

$$y_6 = y_1 \oplus y_5 = 0 \oplus 1 = 1$$

Extrinsic information

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

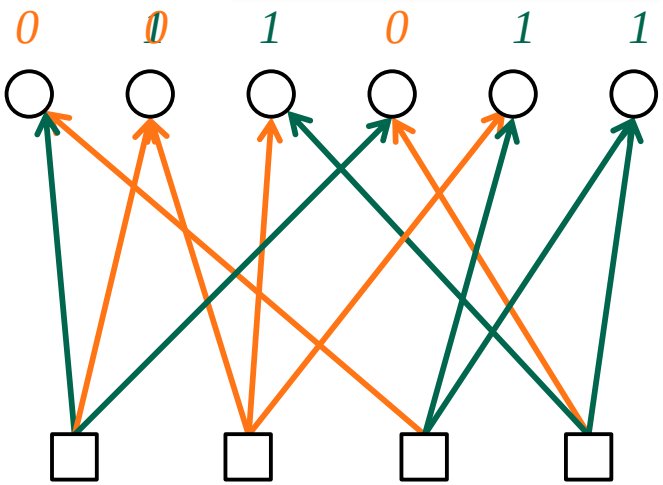
Transmitted codeword $\mathbf{x}$:

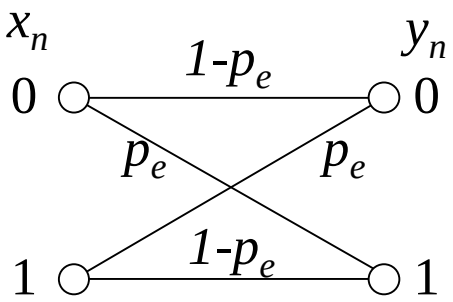
BSC

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

Update messages: Majority decision





Binary symmetric channel (BSC)

Passed messages:

0

1

Current value at bit node is flipped, if the majority of received messages indicates a different decision than the current value of the bit node.

All errors have been corrected after 1 iteration.

One iteration consists of one complete cycle, i.e. sending messages once from bit nodes to check nodes and back from check nodes to bit nodes.

Hard Decision Message Passing Decoding (Bit-Flipping Decoding)

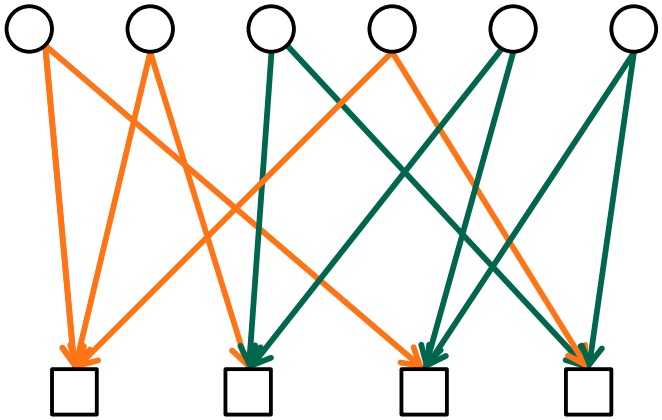
Transmitted codeword $\mathbf{x}$:

BSC

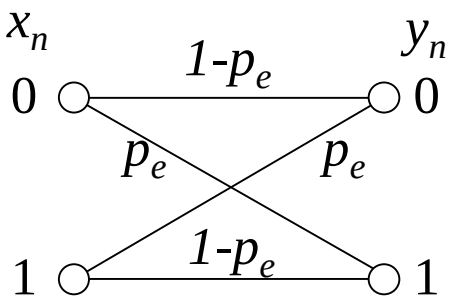
$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$



All parity checks are satisfied.
→ a valid codeword is found.



Binary symmetric channel (BSC)

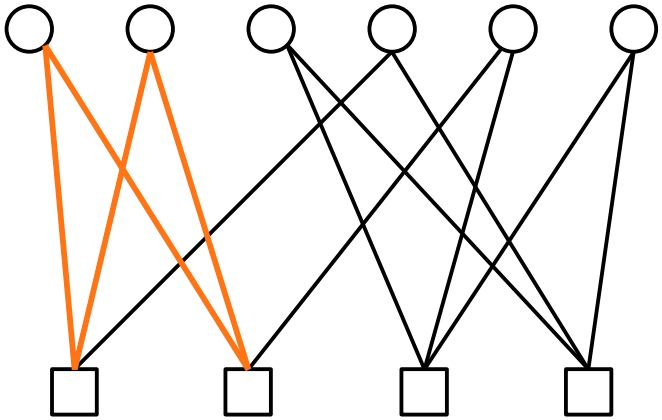
Passed messages:

When a codeword is found, furher iterations do not change the decoding result anymore.

The iterative process is stopped when
- a maximum number of iterations has been performed or
- all parity check equations are satisfied.

Hard Decision Message Passing Decoding (Bit-Flipping Decoding): Effect of Short Cycles

LDPC code with short cycle of length 4 in Tanner graph.



Hard Decision Message Passing Decoding (Bit-Flipping Decoding): Effect of Short Cycles

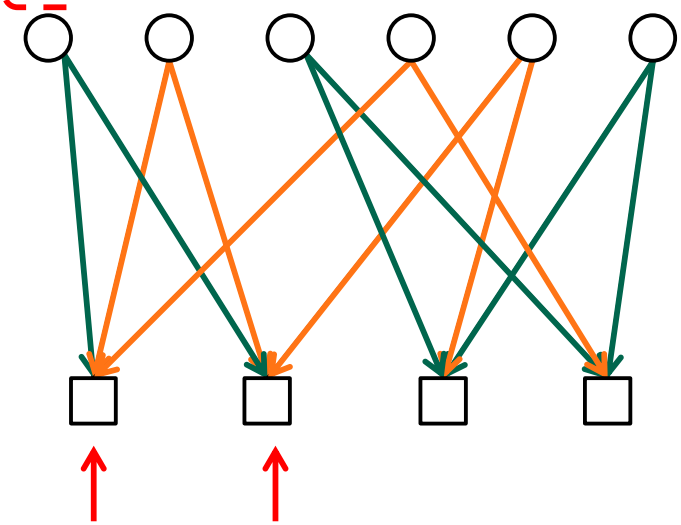
Transmitted codeword $\mathbf{x}$:

$0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$

BSC

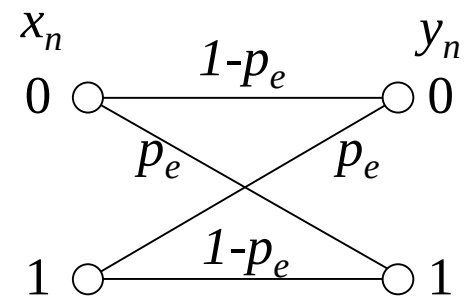
Received word $\mathbf{y}$:

error $1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$



check nodes will

- correct error in bit y_1
- introduce new error in bit y_2



Binary symmetric channel (BSC)

Passed messages:

Hard Decision Message Passing Decoding (Bit-Flipping Decoding): Effect of Short Cycles

Transmitted codeword $\mathbf{x}$:

$0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$

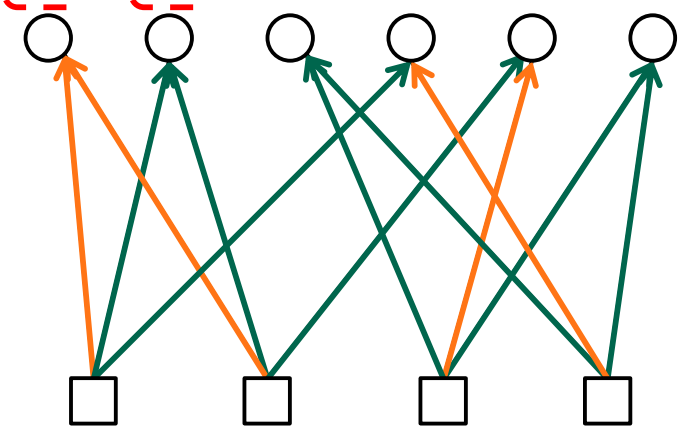
BSC

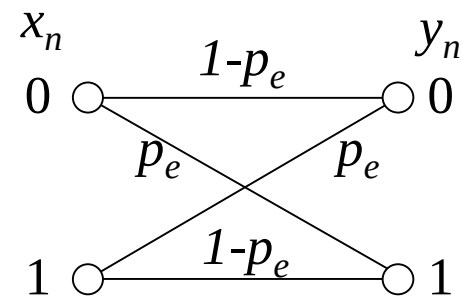
Received word $\mathbf{y}$:

error error

$1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1$

Update messages:
Majority decision





Binary symmetric channel (BSC)

Passed messages:

0

1

Hard Decision Message Passing Decoding (Bit-Flipping Decoding): Effect of Short Cycles

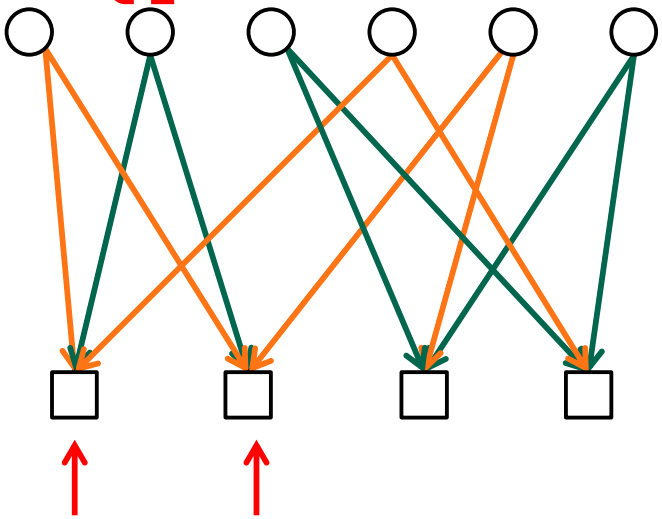
Transmitted codeword $\mathbf{x}$:

BSC

$0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$

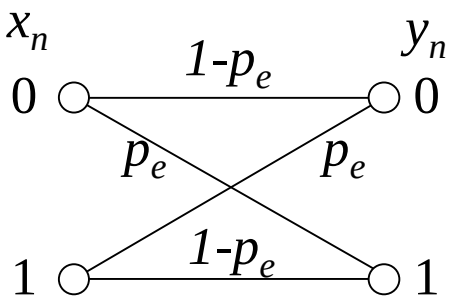
Received word $\mathbf{y}$: error

$0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1$



check nodes will

- correct error in bit y_2
- introduce new error in bit y_1



Binary symmetric channel (BSC)

Passed messages:

Hard Decision Message Passing Decoding (Bit-Flipping Decoding): Effect of Short Cycles

Transmitted codeword $\mathbf{x}$:

$0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$

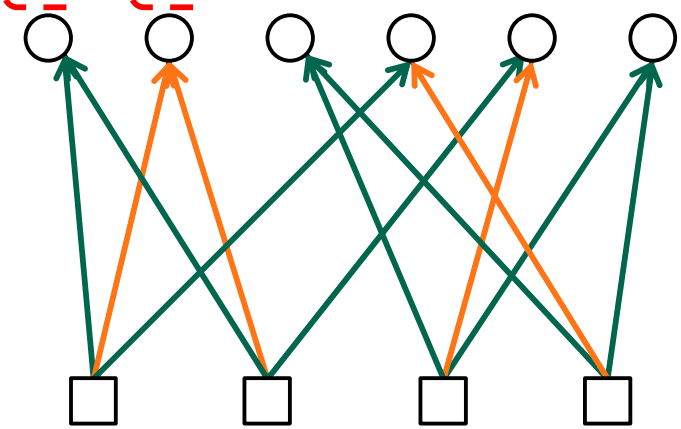
BSC

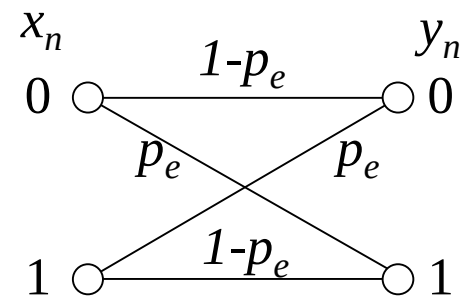
Received word $\mathbf{y}$:

error error

$1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1$

Update messages: Majority decision





Binary symmetric channel (BSC)

Passed messages:

0

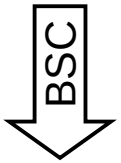
1

In further iterations, the error will continue to flip between y_1 and y_2.

y_1 and y_2 are both involved in the first two parity check equations only due to cycle of length 4.
→ both parity check equations are not satisfied
→ error can not be located

Hard Decision Message Passing Decoding (Alternative Bit-Flipping Decoding)

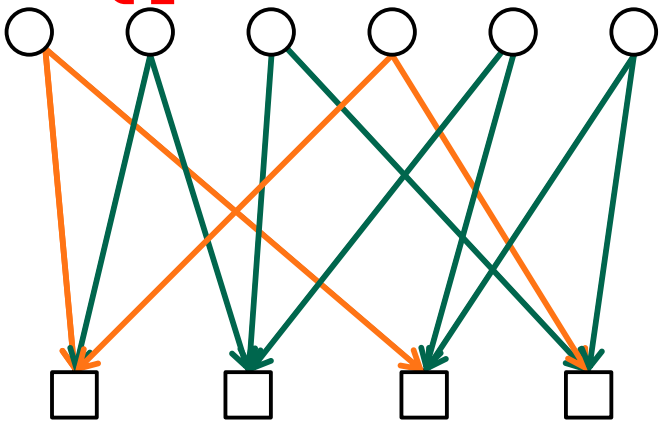
Transmitted codeword $\mathbf{x}$:



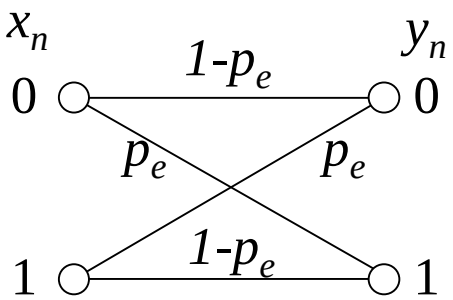
0 0 1 0 1 1

Received word $\mathbf{y}$: error

0 1 1 0 1 1



1 1 0 0



Binary symmetric channel (BSC)

Passed messages:

0

1

Same information is passed pack to bit nodes:

0 if parity check equation is satisfied

1 otherwise

→ lower complexity in check nodes

Hard Decision Message Passing Decoding (Alternative Bit-Flipping Decoding)

Transmitted codeword $\mathbf{x}$:

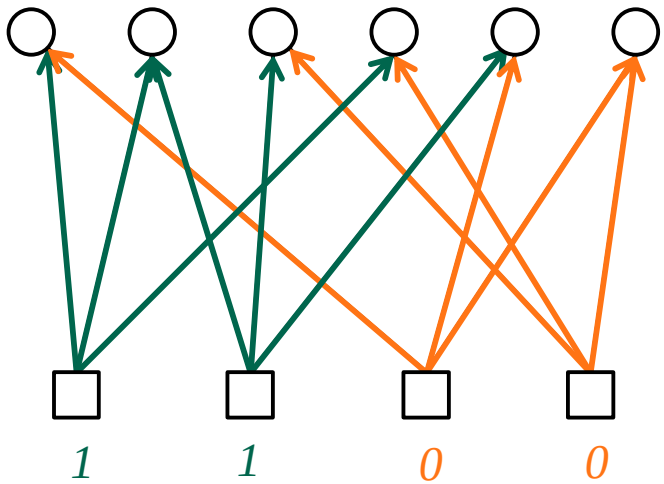
BSC

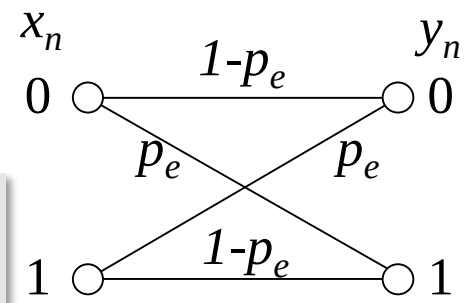
0 0 1 0 1 1

Received word $\mathbf{y}$:

0 0 1 0 1 1

Update messages:
Flip bit, if 2 connected parity checks are unsatisfied.





Binary symmetric channel (BSC)

Passed messages:

0

1

Same information is passed pack to bit nodes:

0 if parity check equation is satisfied

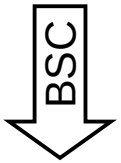
1 otherwise

→ lower complexity in check nodes

Error has been corrected after 1 iteration.

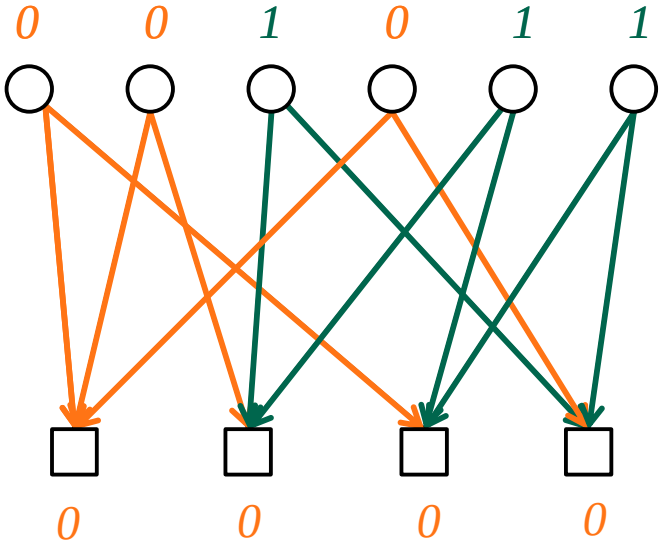
Hard Decision Message Passing Decoding (Alternative Bit-Flipping Decoding)

Transmitted codeword $\mathbf{x}$:

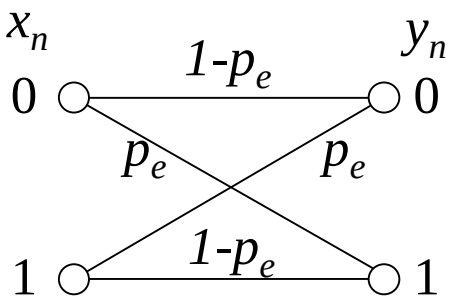


0 0 1 0 1 1

Received word $\mathbf{y}$:



All parity checks are satisfied.
→ a valid codeword is found.



Binary symmetric channel (BSC)

Passed messages:

Same information is passed pack to bit nodes:

0 if parity check equation is satisfied

1 otherwise

→ lower complexity in check nodes

Soft Decision Message Passing Decoding (Sum-Product Decoding, Min-Sum Decoding) (1)

The target of soft decision message passing is to provide a soft-in soft-out decoder which accepts soft information (e.g. in the form of log-likelihood ratios or probabilities) as input and delivers soft information at its output.

In soft decision message passing, soft extrinsic information is exchanged among bit nodes and check nodes. The algorithm works pretty much the same as the extrinsic information based bit-flipping algorithm described before. However, instead of initializing the bit nodes with the hard decided values of the received code bits, we calculate a log-likelihood ratio $L_c(\hat{x}_i)$ which represents the channel information.

This channel information is passed to the check nodes, where we denote the message passed from bit node i to the connected check nodes simply by $L(x_i)$.

At the check node j, extrinsic information about code bit x_i can be computed from the soft information which is available for all other bits $x_{i'}$, $i' \neq i$, which are involved in the parity check equation at check node j. The operation for log-likelihood ratios, which corresponds to the XOR sum of two bits is given by the boxplus operation: The boxplus sum is the log-likelihood ratio of the mod 2 or XOR sum of two bits x_1 and x_2:

$$L(x_1 \oplus x_2) = L(x_1) \boxplus L(x_2) \approx \operatorname{sgn}\{ L(x_1) \} \cdot \operatorname{sgn}\{ L(x_2) \} \cdot \min\{ |L(x_1)|, | L(x_2)| \}$$

The boxplus sum can either be evaluated exactly or the approximation on the right hand side of the equation above can be applied. In the latter case, the message passing algorithm is called *min-sum algorithm*, since the product of the signs of the log-likelihood ratios can be evaluated as mod 2 sum over the hard bit decisions and, consequently, only a sum and the computation of a minimum is required in order to compute the boxplus sum.

Soft Decision Message Passing Decoding (Sum-Product Decoding, Min-Sum Decoding) (2)

The extrinsic information which is passed back from check node j to bit node i is the boxplus sum over all log-likelihood ratios involved in the parity check equation at check node j except the one received from bit node i:

$$L_{e,j}\left(x_i\right) = \sum_{i' \neq i}^{\boxplus} L\left(x_{i'}\right)$$

The bit node i now receives extrinsic information about the code bit x_i from all check nodes to which it is connected. It assumes that the extrinsic information received from different check nodes is independent. Hence, the respective log-likelihood ratios can be simply added in order to get a more reliable estimate. Note that the assumption of independent extrinsic information is only valid as long as the Tanner graph is free of cycles. Otherwise, the same information will be used multiple times.

If the iterative message passing decoding was terminated now, the soft output for bit x_i would be the sum of the received channel information and the total extrinsic information which was gained through the decoding process:

$$L\left(\hat{x}_i\right) = L_c\left(x_i\right) + \sum_j L_{e,j}\left(x_i\right)$$

Soft Decision Message Passing Decoding (Sum-Product Decoding, Min-Sum Decoding) (3)

In case the iterative process is continued, we update the messages which are passed back from the bit nodes to the check nodes. In order to keep the exchanged information independent from previous iterations, we only pass that part of the information available at bit node i to check node j, which was not received from check node j:

$$L\left(x_i\right) = L_c\left(x_i\right) + \sum_{j' \neq j} L_{e,j'}\left(x_i\right)$$

This iterative process is continued until a predefined maximum number of iterations has been performed. Alternatively, a stopping criterion can terminate the iterative process. A possible stopping criterion is that the parity check equations at all check nodes are satisfied based on the hard decisions of the current decoder output $L\left(\hat{x}_i\right)$. In this case, a valid codeword has been found and further iterations will not change the hard decisions anymore but may only increase the reliability, i.e. the magnitudes $\left|L\left(\hat{x}_i\right)\right|$ of the output log-likelihood ratios.
Ideally, the output log-likelihood ratios would be the exact maximum-likelihood estimate $L(\mathbf{y}|x_k)$ of each code bit. This is in fact achieved if the Tanner graph is free of cycles and if all the check nodes compute exact boxplus operations rather than approximations. In case of cycles in the graph, the same information is used multiple times without taking this correlation into account. This will cause suboptimum soft output values rather than true maximum likelihood estimates. However, for properly designed LDPC codes (with large girth), message passing can fairly well approximate maximum likelihood decoding. Maximum likelihood decoding can always serve as a lower bound to the performance of message passing decoding. However, in particular for large codeword length, maximum likelihood decoding will have prohibitive computational complexity. In contrast, the complexity of message passing grows only linearly with the codeword length N.

Soft Decision Message Passing Decoding (Sum-Product Decoding)

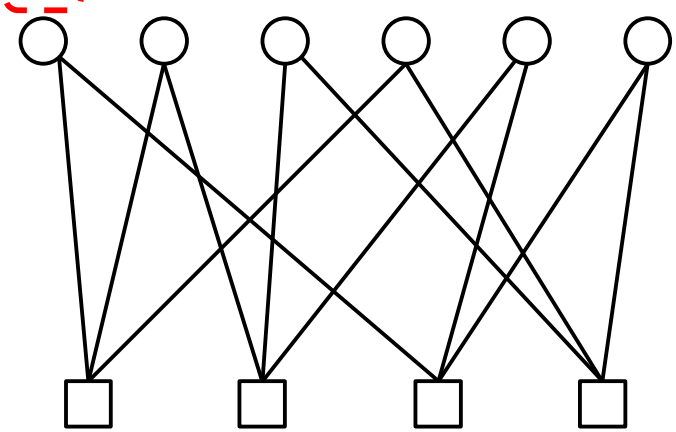
Transmitted codeword $\mathbf{x}$:

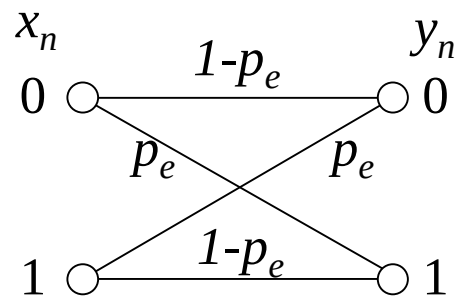
BSC

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Received word $\mathbf{y}$:

error $\quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$





$p_e = 0.2$:

$L_c(x_n) = 1.4$

$L_c(x_n) = -1.4$

Soft channel info:

$$L_c(x_n) = \log \frac{p(y_n \mid x_n = 0)}{p(y_n \mid x_n = 1)}$$

$p_e = 0.2$:

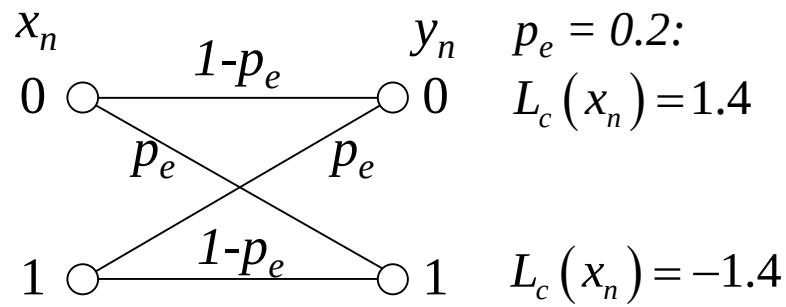
$$L_c(x_n) = \log \frac{p(y_n = 0 \mid x_n = 0)}{p(y_n = 0 \mid x_n = 1)}$$

$$= \log \frac{1 - p_e}{p_e} = 1.4$$

$$L_c(x_n) = \log \frac{p(y_n = 1 \mid x_n = 0)}{p(y_n = 1 \mid x_n = 1)}$$

$$= \log \frac{p_e}{1 - p_e} = -1.4$$

Soft Decision Message Passing Decoding (Sum-Product Decoding)



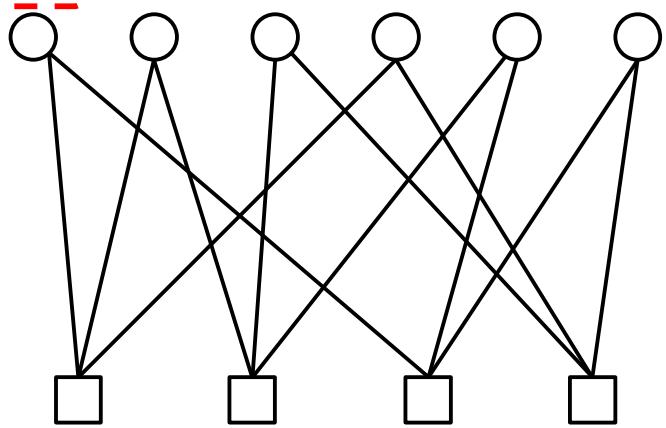
Transmitted codeword $\mathbf{x}$:

BSC

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

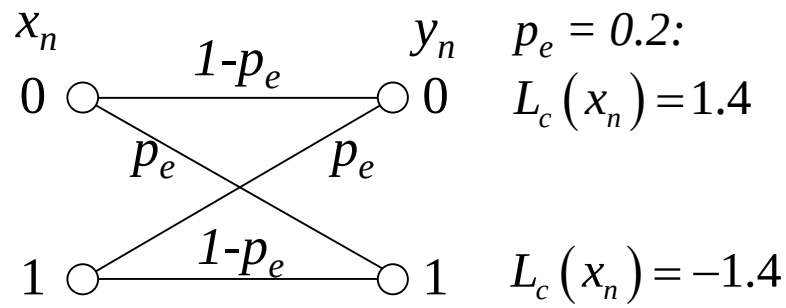
Channel info $L_c(x_n)$:

error $-1.4 \quad 1.4 \quad -1.4 \quad 1.4 \quad -1.4 \quad -1.4$



Log-likelihood ratios (soft info) are passed along the edges.
→ higher complexity (soft values need to be represented by more than 1 bit).

Soft Decision Message Passing Decoding (Sum-Product Decoding)



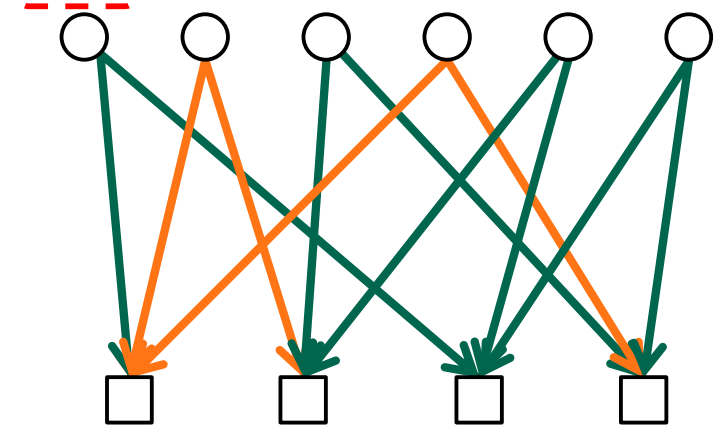
Transmitted codeword $\mathbf{x}$:

BSC

$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

Channel info $L_c(x_n)$:

error $-1.4 \quad 1.4 \quad -1.4 \quad 1.4 \quad -1.4 \quad -1.4$



$x_1 \oplus x_2 \oplus x_4 = 0$

Passed messages:

$\text{sgn}\{L_c(x_n)\} = +1$

$\text{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \triangleq |L_c(x_n)|$

Determine messages to be passed back:

$$x_1 = x_2 \oplus x_4 \;\Rightarrow L_{e,1}(x_1) = L(x_2) \boxplus L(x_4) = 0.8$$

$$x_2 = x_1 \oplus x_4 \;\Rightarrow L_{e,1}(x_2) = L(x_1) \boxplus L(x_4) = -0.8$$

$$x_4 = x_1 \oplus x_2 \;\Rightarrow L_{e,1}(x_4) = L(x_1) \boxplus L(x_2) = -0.8$$

Extrinsic information:
Information about bit x_i which is obtained from soft info about all other bits involved in the parity check equation.

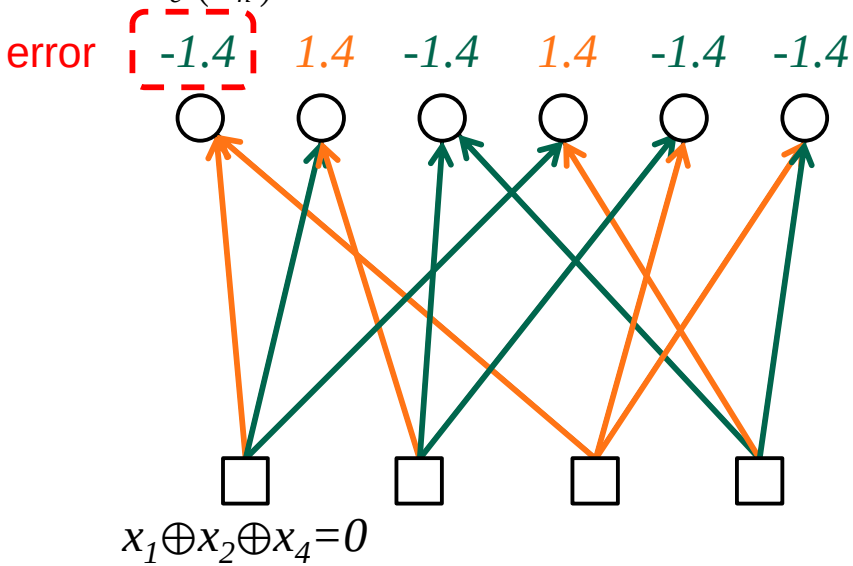
Soft Decision Message Passing Decoding (Sum-Product Decoding)

Soft decoder output: $L(\hat{x}_1) = L_c(x_1) + L_{e,1}(x_1) + L_{e,3}(x_1) = -1.4 + 0.8 + 0.8 = +0.2$

info from channel, check node 1, check node 3

independent info
→ sum up L-values

Channel info $L_c(x_n)$:



Passed messages:

$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \hat{=} |L_c(x_n)|$

Determine messages to be passed back:

$$x_1 = x_2 \oplus x_4 \;\Rightarrow L_{e,1}(x_1) = L(x_2) \boxplus L(x_4) = 0.8$$

$$x_2 = x_1 \oplus x_4 \;\Rightarrow L_{e,1}(x_2) = L(x_1) \boxplus L(x_4) = -0.8$$

$$x_4 = x_1 \oplus x_2 \;\Rightarrow L_{e,1}(x_4) = L(x_1) \boxplus L(x_2) = -0.8$$

Extrinsic information:
Information about bit x_i which is obtained from soft info about all other bits involved in the parity check equation.

Soft Decision Message Passing Decoding (Sum-Product Decoding)

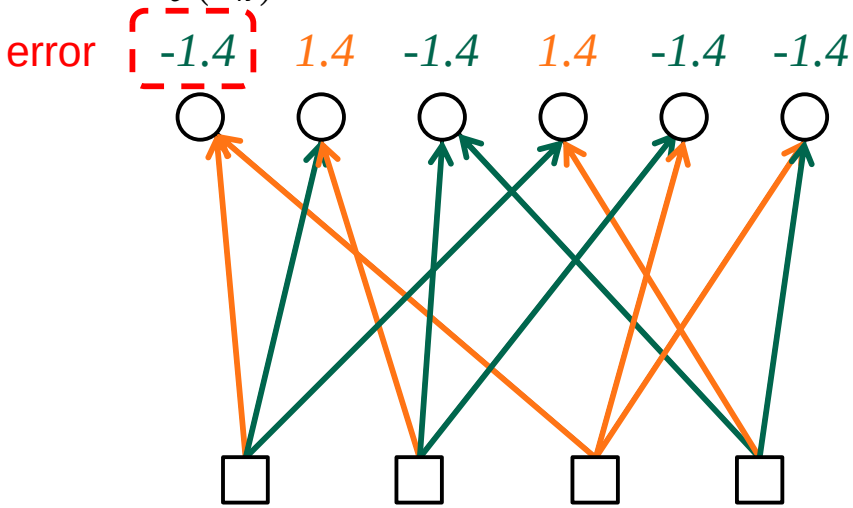
Soft decoder output: $L(\hat{x}_1) = L_c(x_1) + \cancel{L_{e,1}(x_1)} + L_{e,3}(x_1) = -1.4 + \cancel{0.8} + 0.8 = +0.2$

info from channel ($L_c(x_1)$), ~~check node 1~~ ($L_{e,1}(x_1)$), check node 3 ($L_{e,3}(x_1)$)

Pass only info to check node 1, which was not received from it:

$$L(x_1) = L_c(x_1) + L_{e,3}(x_1)$$
$$= -1.4 + 0.8 = -0.6$$

Channel info $L_c(x_n)$:



Passed messages:

$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \triangleq |L_c(x_n)|$

Soft Decision Message Passing Decoding (Sum-Product Decoding)

Soft decoder output: $L(\hat{x}_4) = L_c(x_4) + \cancel{L_{e,1}(x_4)} + L_{e,4}(x_4) = 1.4 - \cancel{0.8} + 0.8 = +1.4$

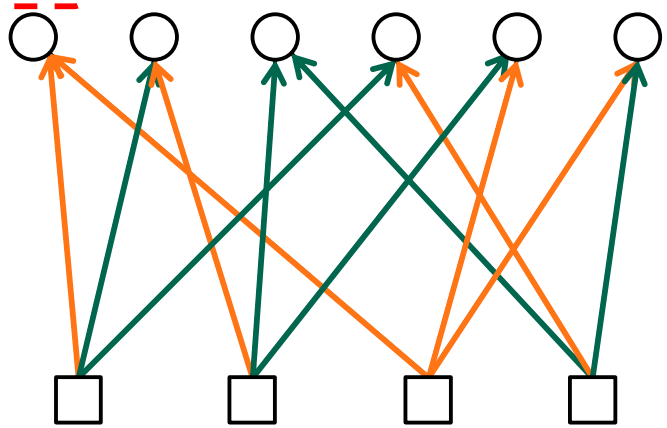
info from channel | ~~check node 1~~ | check node 4

Pass only info to check node 1, which was not received from it:

$$L(x_4) = L_c(x_4) + L_{e,4}(x_4)$$
$$= 1.4 + 0.8 = 2.2$$

Channel info $L_c(x_n)$:

error -1.4 1.4 -1.4 1.4 -1.4 -1.4



Passed messages:

$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \triangleq |L_c(x_n)|$

Soft Decision Message Passing Decoding (Sum-Product Decoding)

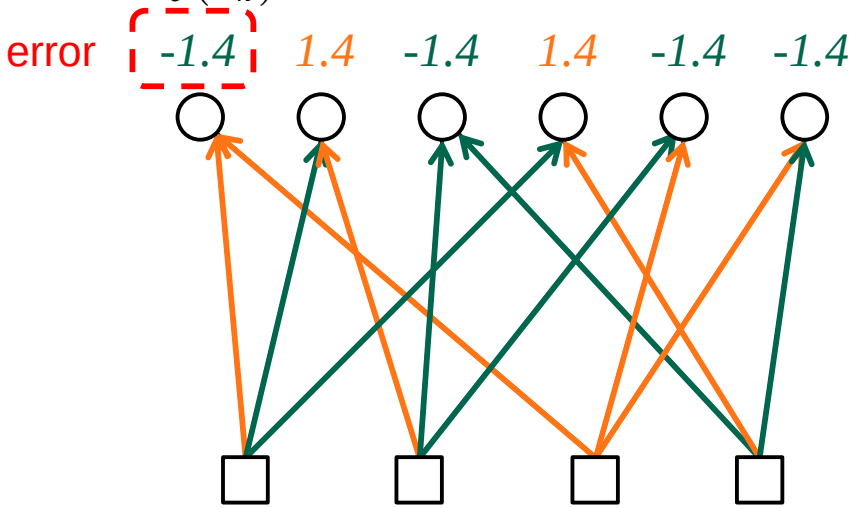
Soft decoder output: $L(\hat{x}_4) = L_c(x_4) + L_{e,1}(x_4) + \cancel{L_{e,4}(x_4)} = 1.4 - 0.8 + \cancel{0.8} = +1.4$

info from channel check node 1 check node 4

Pass only info to check node 4, which was not received from it:

$$L(x_4) = L_c(x_4) + L_{e,1}(x_4)$$
$$= 1.4 - 0.8 = 0.6$$

Channel info $L_c(x_n)$:



Passed messages:

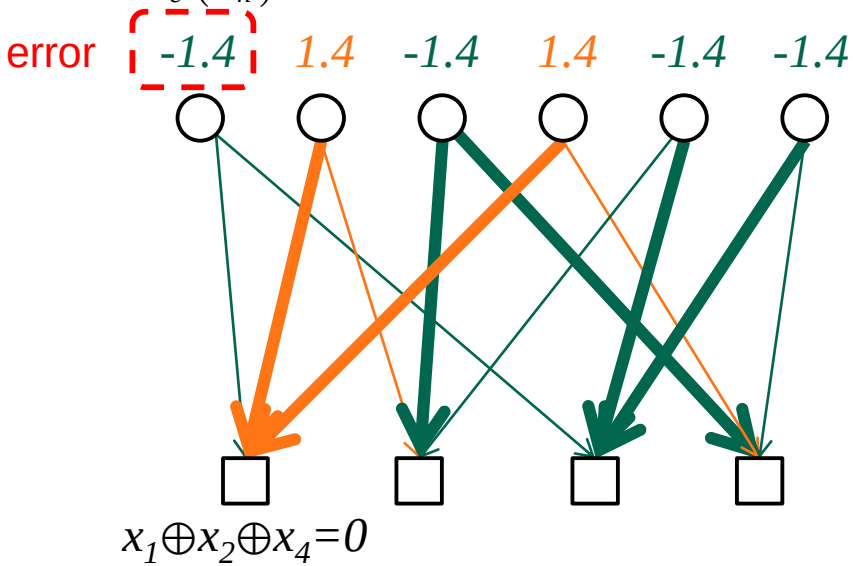
$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \hat{=} |L_c(x_n)|$

Soft Decision Message Passing Decoding (Sum-Product Decoding)

Channel info $L_c(x_n)$:



Passed messages:

$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \triangleq |L_c(x_n)|$

Determine messages to be passed back:

$$x_1 = x_2 \oplus x_4 \;\Rightarrow L_{e,1}(x_1) = L(x_2) \boxplus L(x_4) = 2.2 \boxplus 2.2 = 1.5$$

$$x_2 = x_1 \oplus x_4 \;\Rightarrow L_{e,1}(x_2) = L(x_1) \boxplus L(x_4) = -0.6 \boxplus 2.2 = -0.5$$

$$x_4 = x_1 \oplus x_2 \;\Rightarrow L_{e,1}(x_4) = L(x_1) \boxplus L(x_2) = -0.6 \boxplus 2.2 = -0.5$$

Soft Decision Message Passing Decoding (Sum-Product Decoding)

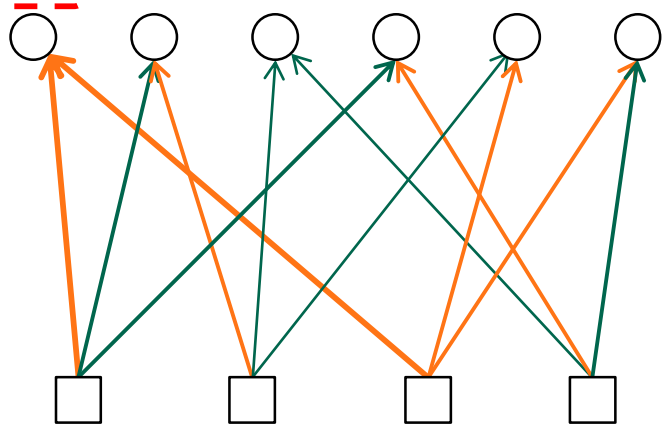
Soft decoder output $L(\hat{x}_n)$:

2.9 1.4 -1.8 1.4 -1.4 -1.4

Once a valid codeword is found, further iterations only increase the reliability of the soft decoder output.

Channel info $L_c(x_n)$:

error -1.4 1.4 -1.4 1.4 -1.4 -1.4



$x_1 \oplus x_2 \oplus x_4 = 0$

Passed messages:

$\mathrm{sgn}\{L_c(x_n)\} = +1$

$\mathrm{sgn}\{L_c(x_n)\} = -1$

$\updownarrow \triangleq |L_c(x_n)|$

Determine messages to be passed back:

$$x_1 = x_2 \oplus x_4 \;\Rightarrow L_{e,1}(x_1) = L(x_2) \boxplus L(x_4) = 2.2 \boxplus 2.2 = 1.5$$

$$x_2 = x_1 \oplus x_4 \;\Rightarrow L_{e,1}(x_2) = L(x_1) \boxplus L(x_4) = -0.6 \boxplus 2.2 = -0.5$$

$$x_4 = x_1 \oplus x_2 \;\Rightarrow L_{e,1}(x_4) = L(x_1) \boxplus L(x_2) = -0.6 \boxplus 2.2 = -0.5$$

Soft Decision Message Passing Decoding (Sum-Product Decoding)

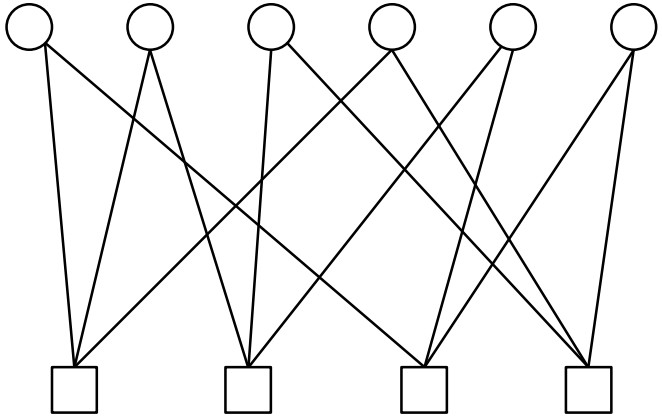
Soft decoder input:

Channel info $L_c(x_i)$

Soft decoder output:

$$L(\hat{x}_i) = L_c(x_i) + \sum_j L_{e,j}(x_i)$$

Soft-in soft-out decoder



Message from bit node i to check node j:

$$L(x_i) = L_c(x_i) + \sum_{j' \neq j} L_{e,j'}(x_i)$$

Iterative exchange of extrinsic information

Message from check node j to bit node i:

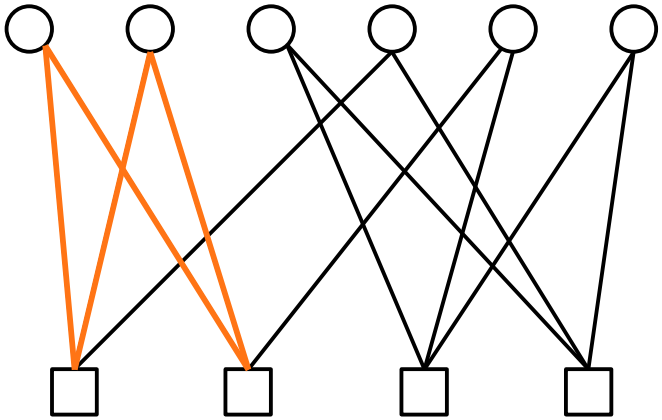
$$L_{e,j}(x_i) = \sum_{i' \neq i}^{\boxplus} L(x_{i'})$$

$$\sum_k^{\boxplus} L(x_k) = \log \frac{1 + \prod_k \tanh\left(L(x_k)/2\right)}{1 - \prod_k \tanh\left(L(x_k)/2\right)} \approx \boxed{\left[\prod_k \operatorname{sgn}\left\{L(x_k)\right\}\right] \cdot \min\left\{\left|L(x_k)\right|\right\}}$$

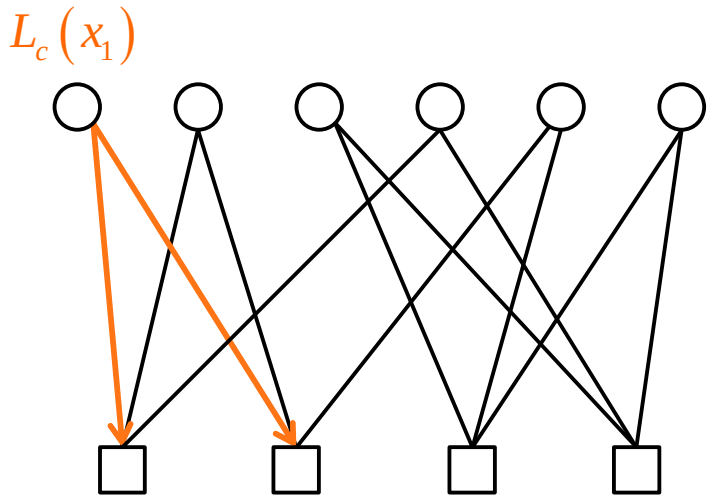
→ min-sum algorithm

Soft Decision Message Passing Decoding
Effect of Short Cycles

LDPC code with short cycle of length 4 in Tanner graph.



Soft Decision Message Passing Decoding
Effect of Short Cycles



$$L(x_i) = L_c(x_i) + \sum_{j' \neq j} L_{e,j'}(x_i)$$

$$L_{e,j}(x_i) = \sum_{i' \neq i}^{\boxplus} L(x_{i'})$$

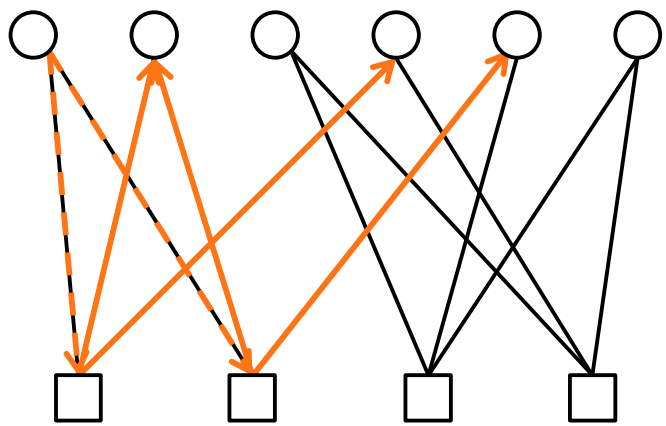
How does $L_c(x_1)$ propagate through the Tanner graph ?

Soft Decision Message Passing Decoding
Effect of Short Cycles

Current soft decoder output:

$$L(\hat{x}_1) = L_c(x_1) + \underbrace{L_{e,1}(x_1) + L_{e,2}(x_1)}_{\text{independent of } L_c(x_1)}$$

$L_c(x_1)$

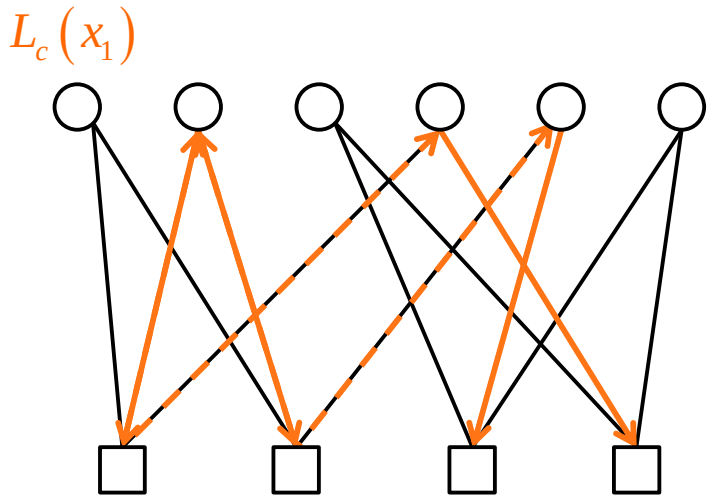


$$L(x_i) = L_c(x_i) + \sum_{j' \neq j} L_{e,j'}(x_i)$$

$$L_{e,j}(x_i) = \sum_{i' \neq i}^{\boxplus} L(x_{i'})$$

How does $L_c(x_1)$ propagate through the Tanner graph ?

Soft Decision Message Passing Decoding
Effect of Short Cycles



$$L(x_i) = L_c(x_i) + \sum_{j' \neq j} L_{e,j'}(x_i)$$

$$L_{e,j}(x_i) = \sum_{i' \neq i}^{\boxplus} L(x_{i'})$$

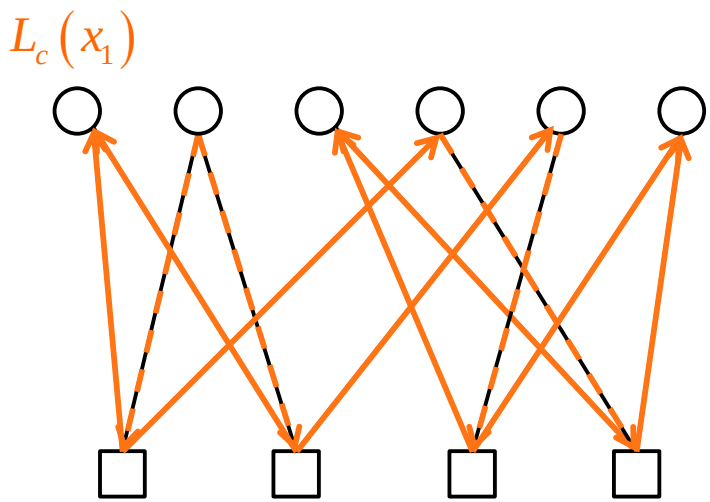
How does $L_c(x_1)$ propagate through the Tanner graph ?

Soft Decision Message Passing Decoding
Effect of Short Cycles

Current soft decoder output:

$$L(\hat{x}_1) = L_c(x_1) + \underbrace{L_{e,1}(x_1) + L_{e,2}(x_1)}_{\text{contains } L_c(x_1)}$$

Channel info $L_c(x_1)$ and extrinsic info $L_{e,j}(x_1)$ from check nodes are not anymore independent.
→ summation of L-values does not yield correct soft output $L(\hat{x}_1)$.

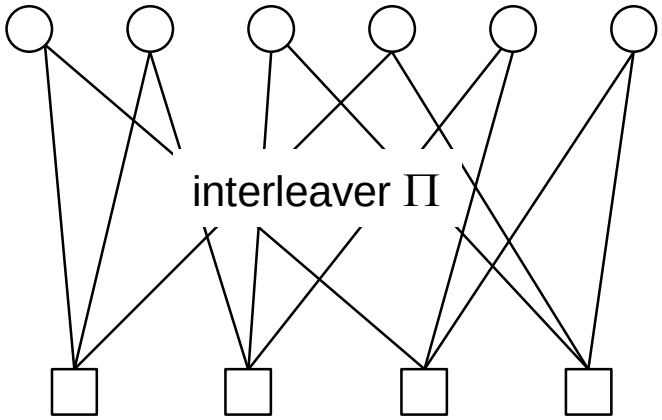


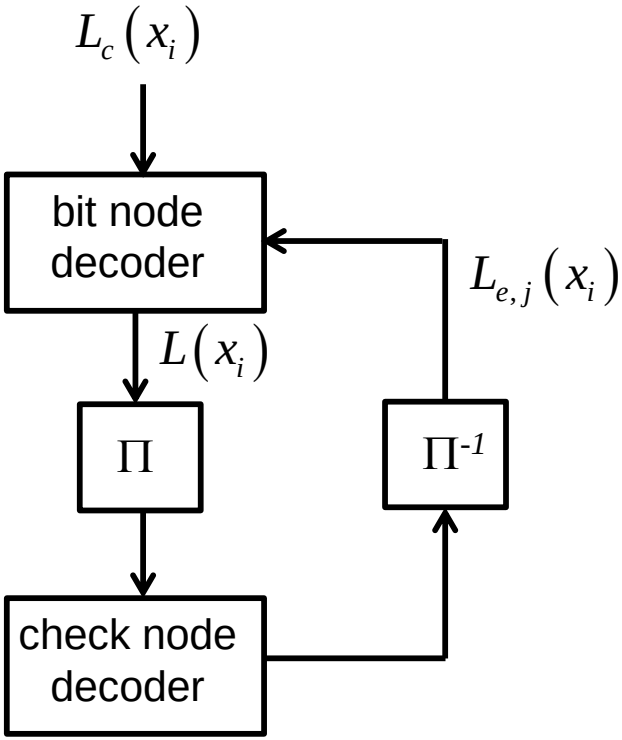
$$L(x_i) = L_c(x_i) + \sum_{j' \neq j} L_{e,j'}(x_i)$$

$$L_{e,j}(x_i) = \sum_{i' \neq i}^{\boxplus} L(x_{i'})$$

Soft message passing decoding is equivalent to maximum likelihood decoding, if the Tanner graph is free of cycles.
Otherwise, message passing decoding is suboptimum.

Soft Decision Message Passing Decoding (Sum-Product Decoding)





Iterative decoding with two constituent decoders

Message Passing Decoding as Turbo Decoder

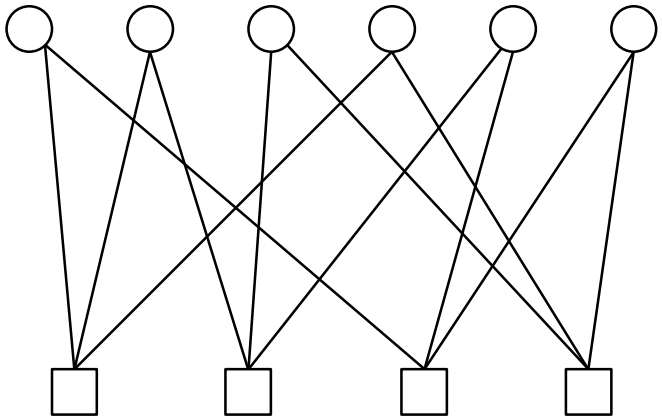
A message passing decoder can also be viewed in a structure which reminds of a turbo decoder with two constituent decoders:
We can view the operations in the bit nodes as a „bit node decoder“ and the operations in the check nodes as a „check node decoder.“
The edges of the Tanner graph in a sense indicate, how the order of code bits is changed between bit node decoder and check node decoder. In this sense, the edges play the role of an interleaver Π which is a decisive component of any turbo decoding scheme.
An interleaver simply changes the order of the bit sequence at its input. In the context of iterative decoding, we wish to make soft information which is exchanged among the constituent decoders as independent as possible. Therefore, the permutation rule of the interleaver should have as few structure as possible, i.e. it should be random.

Advantages and Problems of Low Density Parity Check (LDPC) Codes

Advantages:

- Excellent performance
- High degree of parallelization possible in decoder implementation.

→ enables high speed decoding.



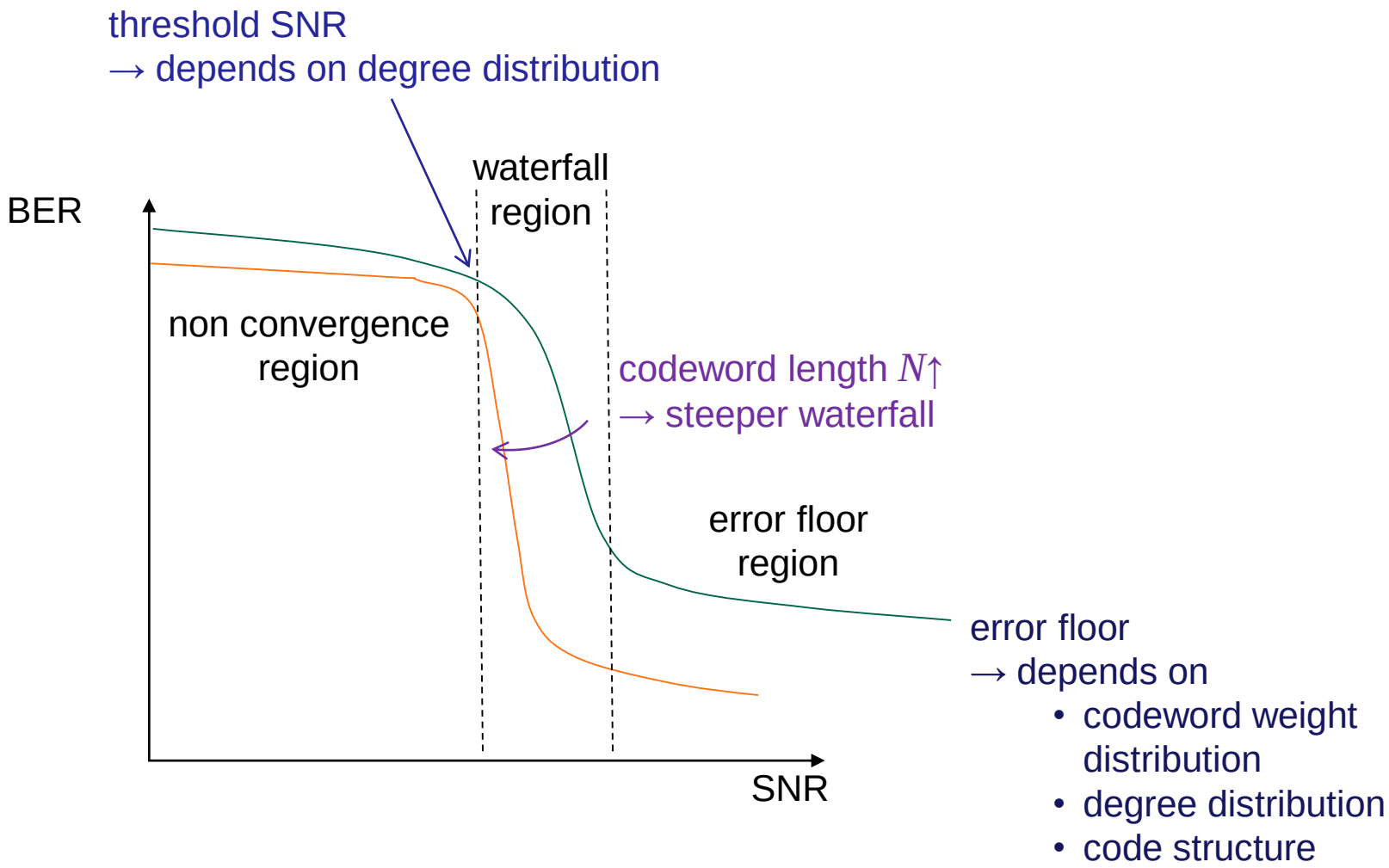
Operations at all bit nodes can be done in parallel.

Operations at all check nodes can be done in parallel.

Problems:

- Typically high number of iterations required.
- Encoding may be computationally complex unless the generator matrix has some structure which can be exploited for low complexity encoding.

Performance of Low Density Parity Check (LDPC) Codes



Performance of Low Density Parity Check Codes with Message Passing Decoding (1)

The bit error rate (BER) vs. SNR performance of iterative decoding schemes such as message passing is typically characterized by three regions:

At very low SNR, the message passing decoder will not converge to a valid codeword. This region is characterized by a BER of about 0.5 and is called the *non convergence region*.

At a certain threshold SNR, the BER curve decreases with very steep slope. This is called the *waterfall region*. The threshold SNR where the iterative („turbo“) algorithm starts to converge is sometimes called the *turbo cliff*. At high SNR, the BER curve flatens out. This region is called the *error floor region*.

The particular shape of the BER curve and the location of the three regions can be influenced by the code design.

Certainly, the performance can be improved by increasing the number of iterations. However, there is a limit to the number of iterations which bring additional benefit. Typically, the incremental BER reduction by further iterations decreases with increasing number of iterations until no additional gains are achieved by performing furher iterations.

Typically, message passing works the better the larger the codeword length is since correlations in the exchanged extrinsic messages are less likely to occur or in other words a larger girth can be achieved (girth: length of shortest cycle in the Tanner graph).

A larger codeword length will increase the steepness in the waterfall region and achieve a lower error floor.

Performance of Low Density Parity Check Codes with Message Passing Decoding (2)

The threshold SNR depends on the degree distribution of the code. It can to some extend be optimized for early convergence (turbo cliff at low SNR). A powerful tool for this optimization is the extrinsic information transfer (EXIT) chart. In particular, using irregular codes (different degrees at different bit and check nodes) allows to flexibly change properties of the code. Typically, there is a trade-off between early convergence and low error floor.
The error floor is determined by the Hamming weight distribution of the code, which depends on both the degree distribution of the nodes and the structure of the parity check matrix. Regarding the structure of the parity check matrix, a desirable property is a large girth. Therefore, short cycles in the Tanner graph should be avoided. It is most important to avoid cycles of length 4. Avoiding additionally cycles of length 6 will bring further performance improvement. However, avoiding even longer cycles typically does not result in significant additional performance gains. Hence, it is sufficient to avoid cycles of length 4 and 6. However, for short codeword length or high code rate, it is difficult to construct codes without short cycles. Sometimes, the restrictions put by the constraint to avoid short cycles are so tough that better codes could be obtained by allowing some short cycles (at least of length 6).

Maximum Likelihood (ML) Decoding versus Message Passing Decoding of LDPC Codes

It should be emphasized that maximum likelihood (ML) decoding would certainly be the optimum decoding method with respect to minimizing the bit error rate (BER) also for LDPC codes. However, in practice, maximum likelihood decoding has prohibitive computational complexity. Message passing decoding is a suboptimum decoding approach. However, it can be implemented with manageable complexity. Moreover, iterative message passing decoding can closely approach the performance of maximum likelihood decoding for properly designed codes and large codeword length.

It should also be noted that the performance of maximum likelihood decoding only depends on the Hamming weight distribution of the code but is independent of the particular parity check matrix which is chosen in order to represent the code. Recall that the parity check matrix of a block code is not unique but elementary permutations of the columns of the parity check matrix $\mathbf{H}$ are allowed.

On the other hand, message passing decoding operates on a Tanner graph which depends on the particular parity check matrix which is used to represent the code. Hence, a different choice of the parity check matrix applied to represent the same code may result in different performance under message passing decoding.

Repeat Accumulate Codes (1)

LDPC codes can be efficiently decoded using message passing decoding. Since the parity check matrix $\mathbf{H}$ is sparse, the number of edges in the Tanner graph is small. Consequently, the number of messages which are passed along the edges of the Tanner graph for decoding and their processing in the bit nodes and check nodes is of manageable complexity. However, encoding is in general a quite complex operation for LDPC codes. Encoding using the multiplication $\mathbf{x}=\mathbf{G}\mathbf{u}$ of the $K\times 1$ information word $\mathbf{u}$ and a $N\times K$ generator matrix $\mathbf{G}$ requires in general N^2 operations. The encoding complexity can be reduced if the generator matrix has some structure which can be exploited for low complexity encoding. However, since the performance of message passing relies on randomness, i.e. pseudo random connections in the Tanner graph, the generator matrix in general lacks structure. Moreover, even though the parity check matrix $\mathbf{H}$ is sparse for low complexity decoding, the generator matrix $\mathbf{G}$ will not be sparse in general.

A number of approaches exist which aim at introducing some structure to the generator matrix without sacrificing performance of message passing decoding. An example is the design of quasi-cyclic LDPC codes. A code is called *quasi-cyclic*, if a cyclic shift of the bits in any codeword by c positions results in a codeword. For $c=1$, the code is called a *cyclic code*. Quasi-cyclic codes can be efficiently encoded using shift registers.

Repeat Accumulate Codes (2)

Most of the LDPC codes in practice belong to the class of *repeat accumulate codes*. The encoder of a repeat accumulate code is a serial concatenation of a rate $1/R_R$ repetition encoder and an accumulator. An accumulator is a recursive precoder (or convolutional code) of rate $R_A=1$. In order to allow for iterative message passing decoding, some randomness has to be introduced using an interleaver Π between the repetition encoder and the accumulator. The interleaver changes the order of the code bits of the repetition code in a pseudo random manner, where a large number of code words of the repetition code have to be jointly interleaved. I.e. a large number of successive code words is passed into the interleaver and permuted according to the interleaving rule.

The encoding process of a repeat accumulate code is obviously of low complexity. However, the structure described above allows only for low rate codes, since the overall code rate is determined by the rate of the repetition code. In order to produce also higher code rates, often a combiner is applied after the interleaver before the accumulator. The combiner operates on successive non-overlapping blocks of K_C interleaved code bits of the repetition code and combines them to a single bit. It is basically a single parity check code of rate $K_C/(K_C+1)$. However, only the parity bit is forwarded to the accumulator. Hence, the combiner has rate $R_C= K_C/1=K_C$. When a combiner is used, it is necessary to also transmit the information bits u_k in order to enable decoding. Hence, the repeat accumulate code then is a systematic code.

Repeat Accumulate Codes (3)

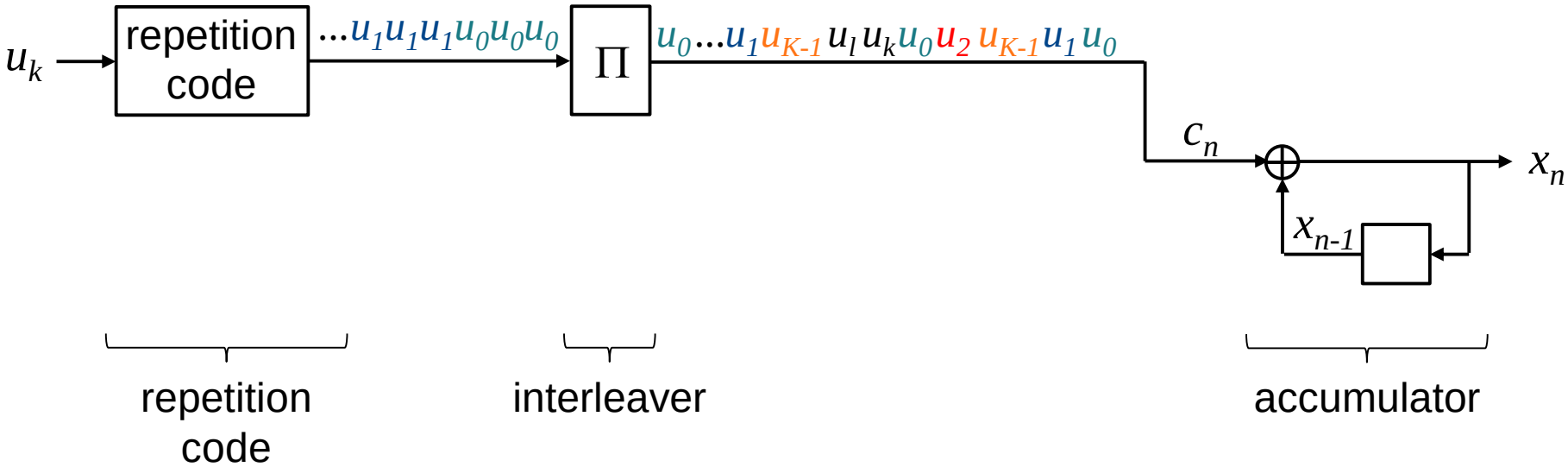
Repeat accumulate codes can be decoded in different ways:

Viewing the repeat accumulate code as a serially concatenated code, it is straightforward to apply the turbo principle for iterative decoding of the constituent codes. Alternatively, the repeat accumulate code can be represented by a Tanner graph which allows for message passing decoding as it is typically done for LDPC codes.

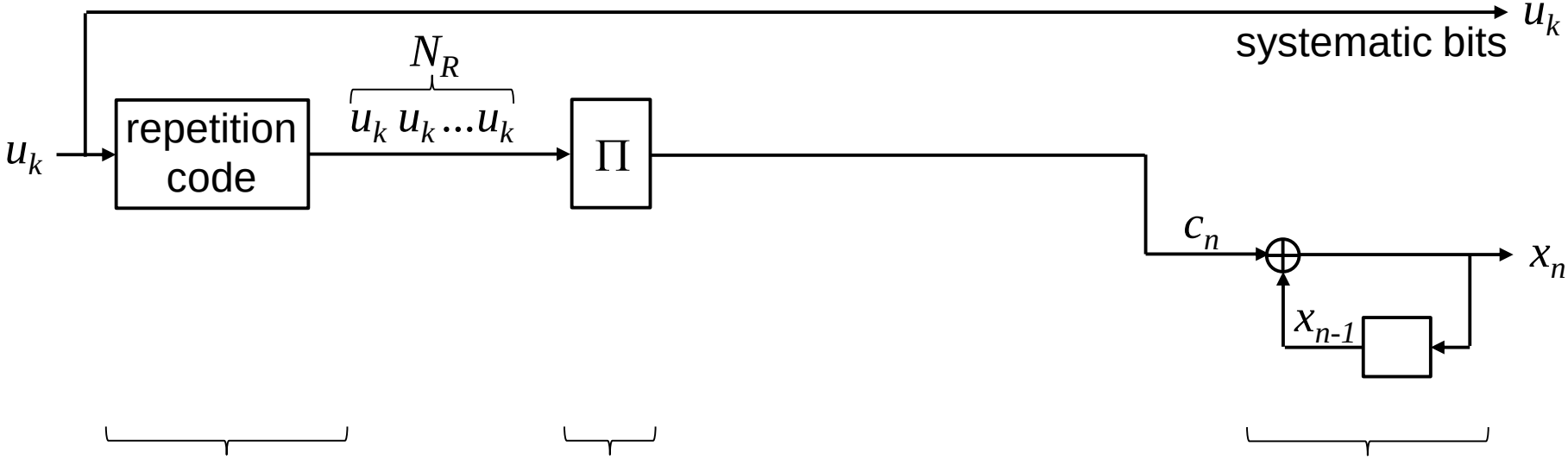
Turbo decoding comprises a more complex constituent decoder for the accumulator compared to the decoding operations in the bit nodes and check nodes during message passing decoding. However, turbo decoding typically requires significantly less iterations than message passing decoding. Hence, the computational complexity of one iteration is higher in turbo decoding but this is compensated for by the lower number of iterations.

Both decoding alternatives can achieve comparable performance in terms of bit error rate (BER). Therefore, the decision for either turbo decoding or message passing decoding is essentially an implementation issue. The decision has to be taken by the manufacturer based on considerations regarding e.g. technology and possibilities for parallelized decoder implementation for lower decoding delay etc..

Repeat Accumulate Codes (1)



Systematic Repeat Accumulate Codes (1)



repetition code

interleaver

accumulator

$$R_R = \frac{1}{N_R}$$

$$R_A = 1$$

Overall code rate:

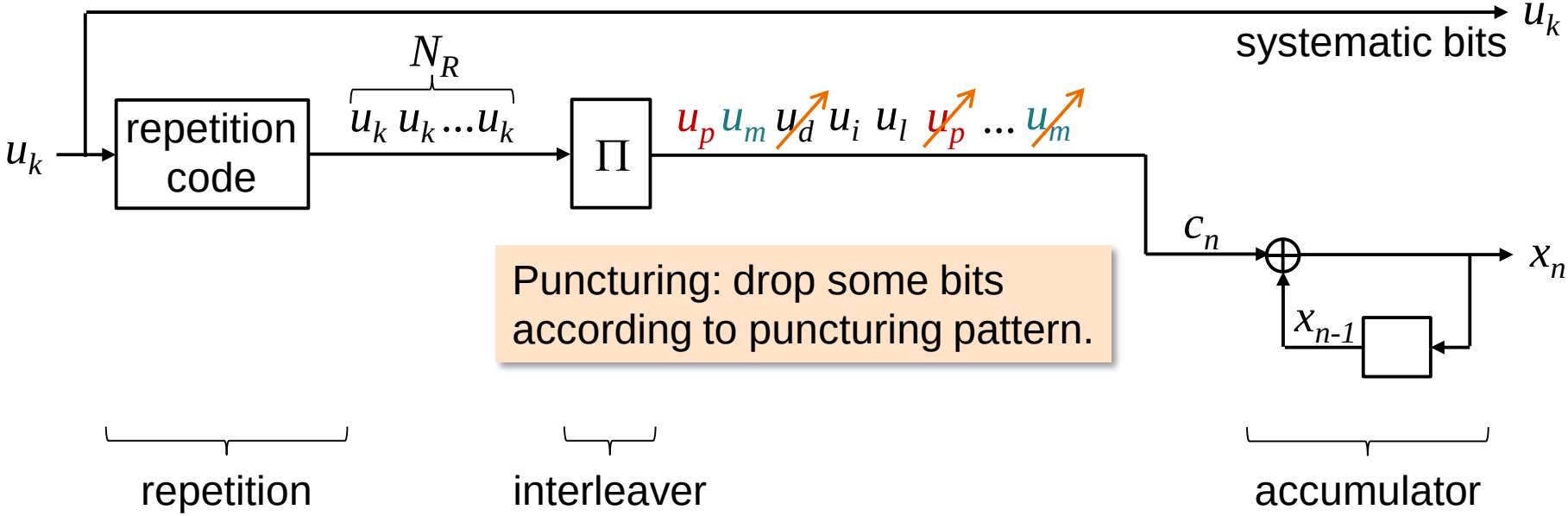
$$R = \frac{K}{K\left(1 + N_R\right)}$$

Problem:
Only codes with low code rate can be obtained.

Possible solutions:

- Puncturing before accumulator
- Combiner before accumulator

Systematic Repeat Accumulate Codes (2)



Puncturing: drop some bits according to puncturing pattern.

repetition code

interleaver

accumulator

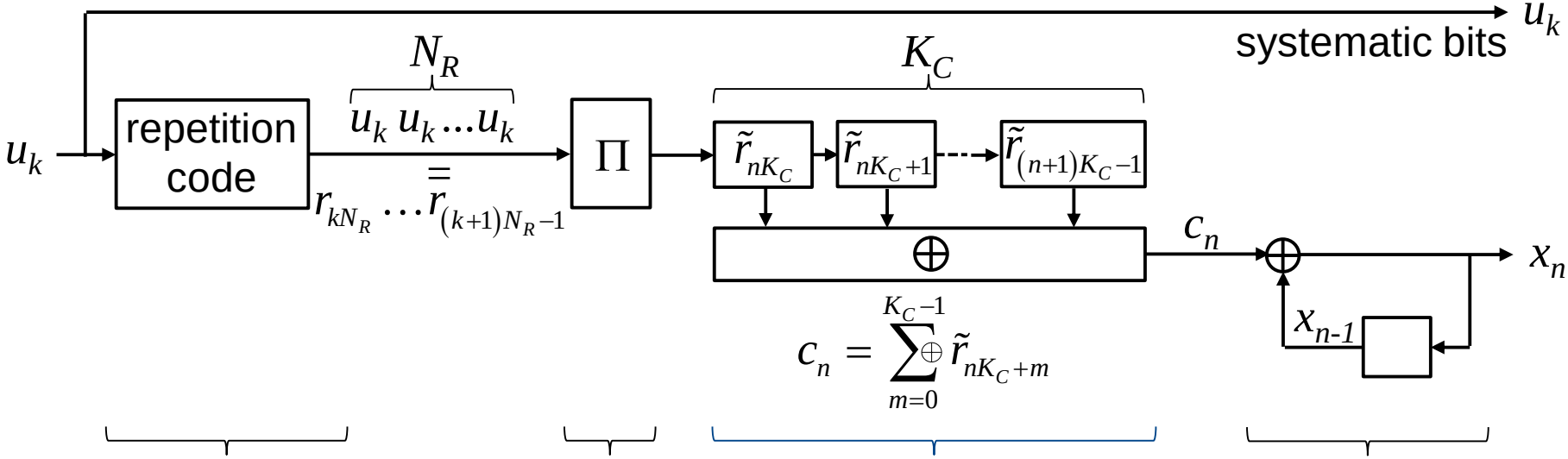
$$R_R = \frac{1}{N_R}$$

$$R_A = 1$$

Overall code rate:

$$R = \frac{K}{K\left(1 + \frac{N_R}{R_P}\right)}$$

Systematic Repeat Accumulate Codes (3)



$$R_R = \frac{1}{N_R} \qquad R_C = \frac{K_C}{1} \qquad R_A = 1$$

Overall code rate:

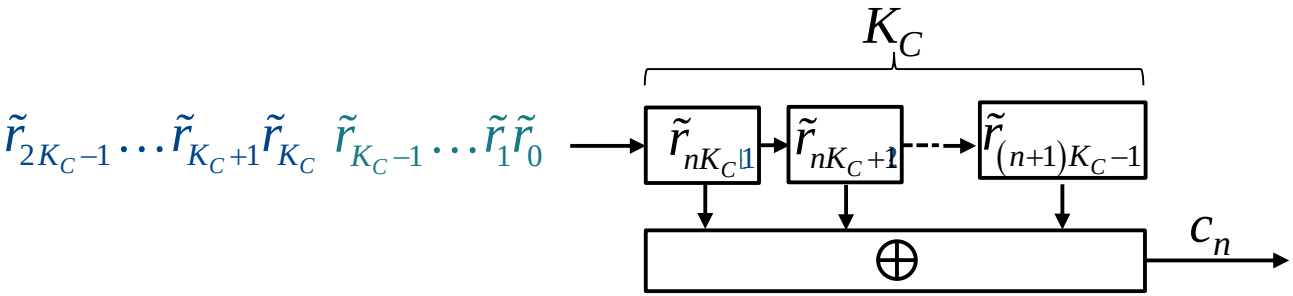
$$R = \frac{K}{K\left(1 + \frac{N_R}{R_C}\right)}$$

Problem:
Only codes with low code rate can be obtained.

Possible solutions:

- Puncturing before accumulator
- **Combiner before accumulator**

Combiner



$$c_n = \sum_{m=0}^{K_C-1} \tilde{r}_{nK_C+m}$$

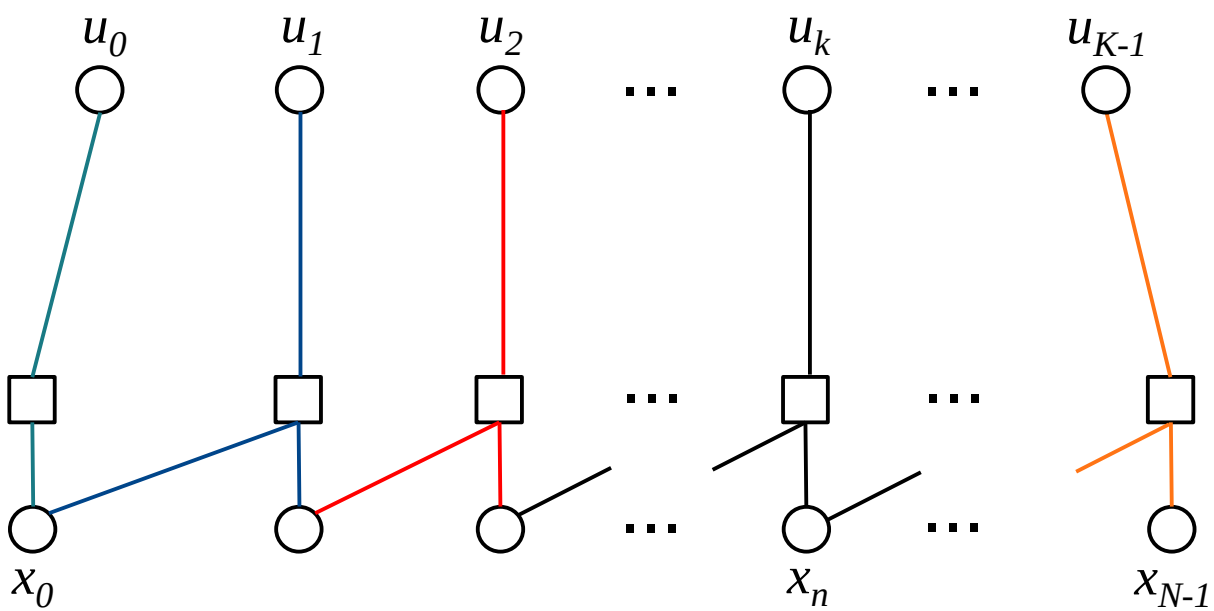
$$c_0 = \tilde{r}_0 \oplus \tilde{r}_1 \oplus \ldots \oplus \tilde{r}_{K_C-1}$$

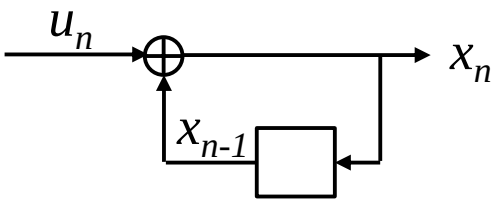
$$c_1 = \tilde{r}_{K_C} \oplus \tilde{r}_{K_C+1} \oplus \ldots \oplus \tilde{r}_{2K_C-1}$$

Accumulator

INSTITUT FÜR NACHRICHTENTECHNIK

$$\mathbf{H}^T = \begin{array}{c} \begin{matrix} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \ddots & 1 & 1 \end{bmatrix} \end{array} \begin{array}{l} u_0 \oplus x_0 = 0 \\ u_1 \oplus x_1 \oplus x_0 \\ u_2 \oplus x_2 \oplus x_1 \\ \\ u_{K-1} \oplus x_{K-1} \oplus x_{K-2} \end{array}$$

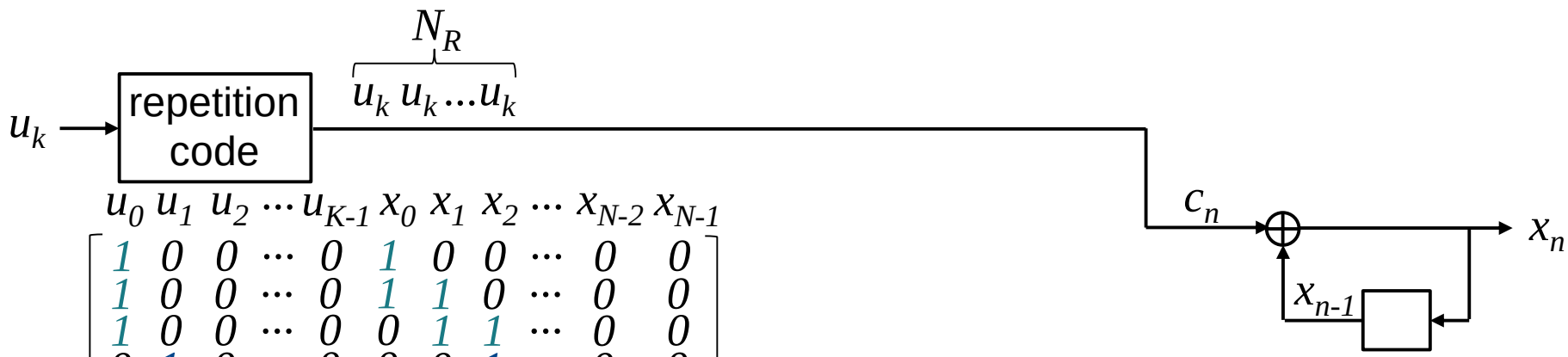




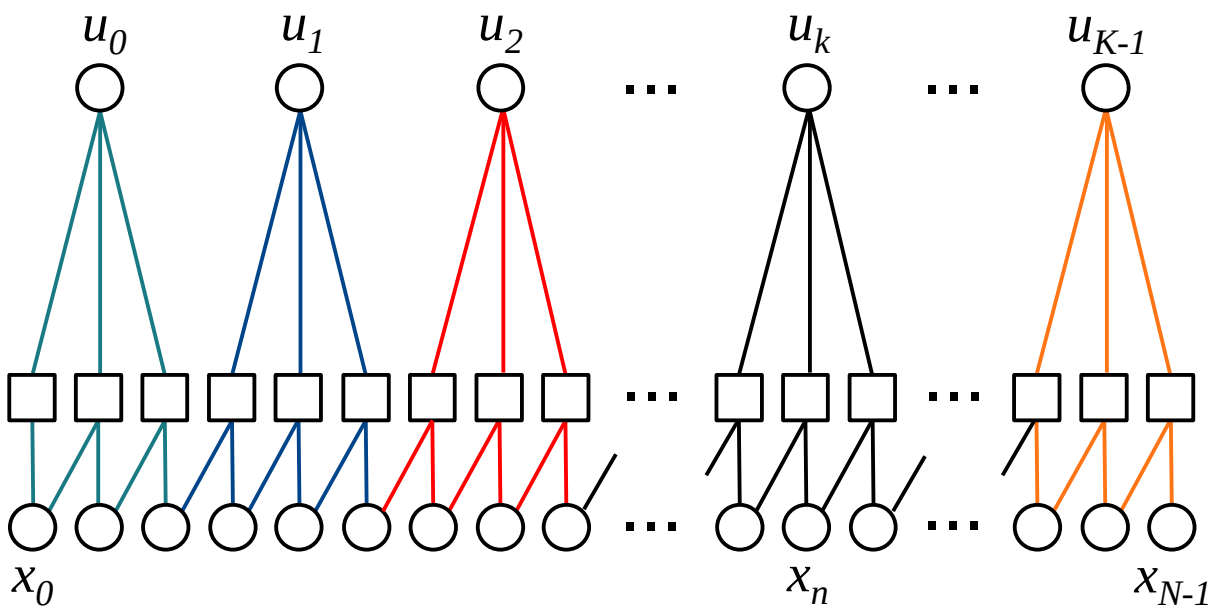
$$\begin{aligned} x_n &= u_n \oplus x_{n-1} \\ &= u_n \oplus u_{n-1} \oplus x_{n-2} \\ &= u_n \oplus u_{n-1} \oplus \ldots \oplus u_0 \\ &= \sum_{m=0}^{n} \oplus u_m \end{aligned}$$

$\rightarrow$ accumulator

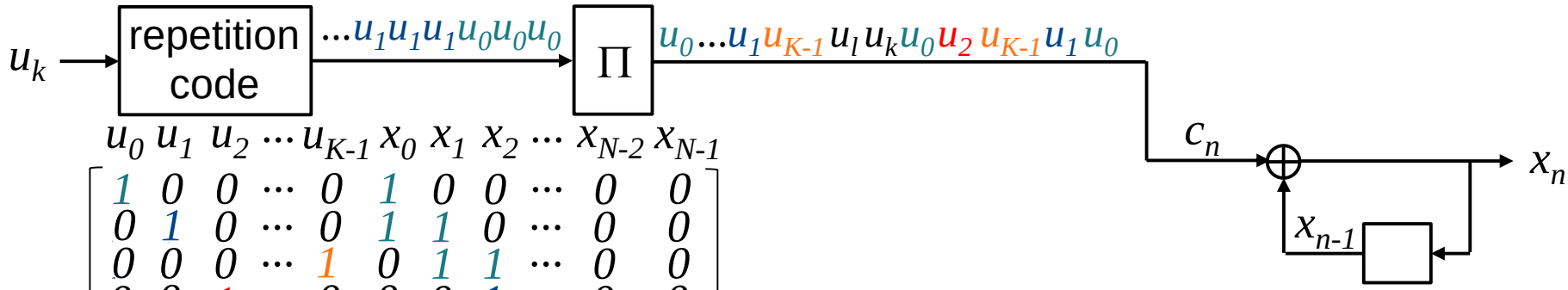
Repeat Accumulate Code



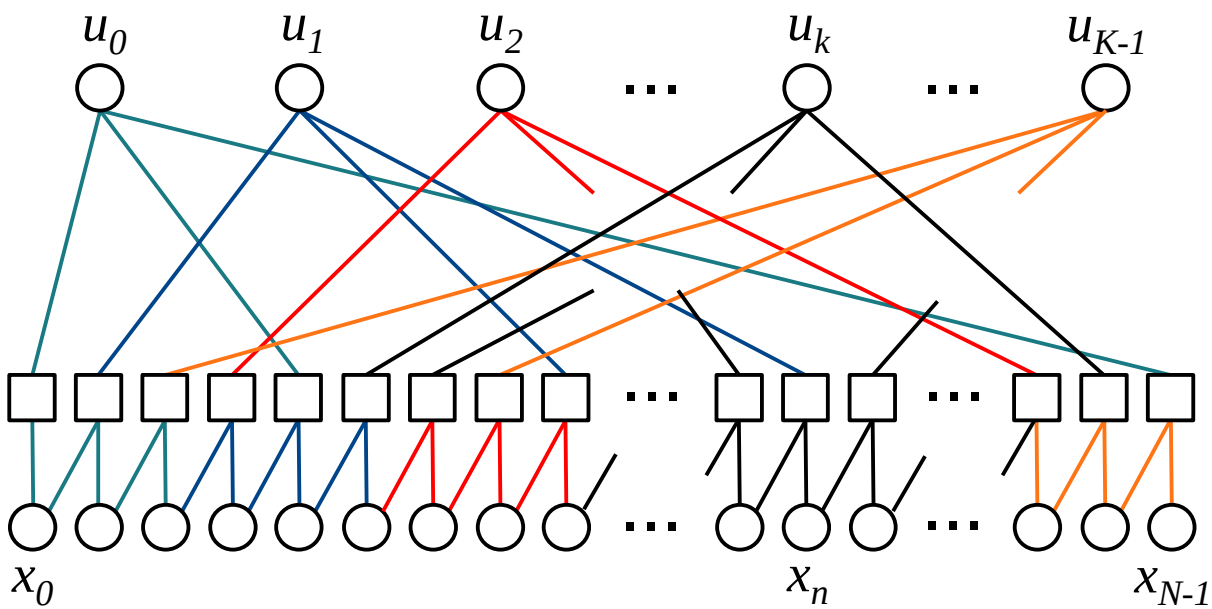
$$
\begin{array}{c}
\begin{array}{cccccccccc} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{array} \\
\mathbf{H}^T = \begin{bmatrix}
1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\
1 & 0 & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\
1 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\
0 & 1 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\
0 & 1 & 0 & & 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\
\vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \ddots & 1 & 1
\end{bmatrix}
\end{array}
$$



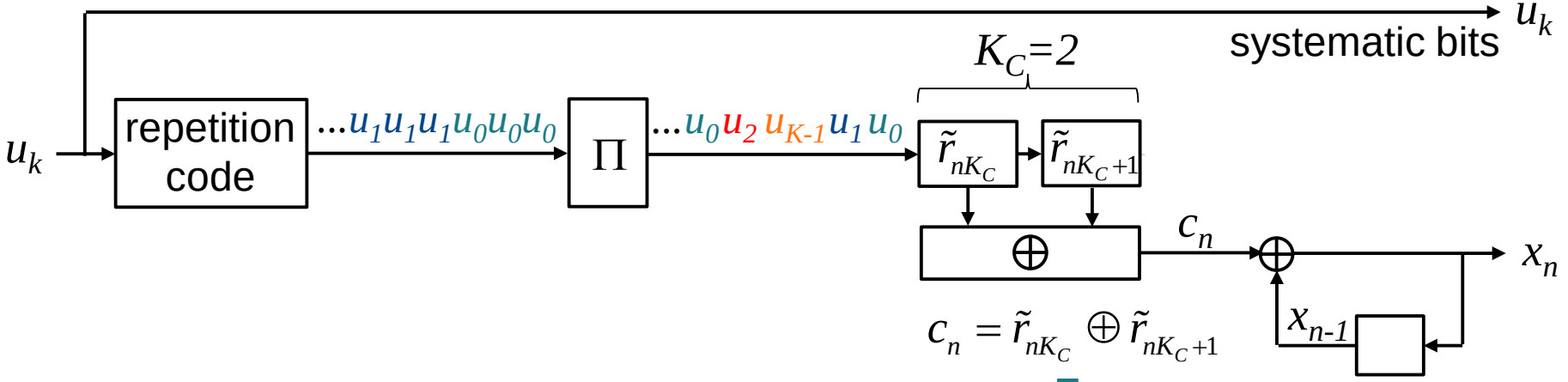
Repeat Accumulate Code



$$\mathbf{H}^T = \begin{array}{c} \begin{matrix} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & 0 & & 0 & 0 & 0 & 0 & & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \ddots & 1 & 1 \end{bmatrix} \end{array}$$



Repeat Accumulate Code

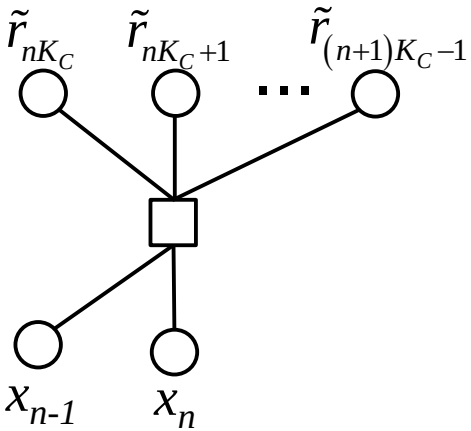


$$c_n = \tilde{r}_{nK_C} \oplus \tilde{r}_{nK_C+1}$$

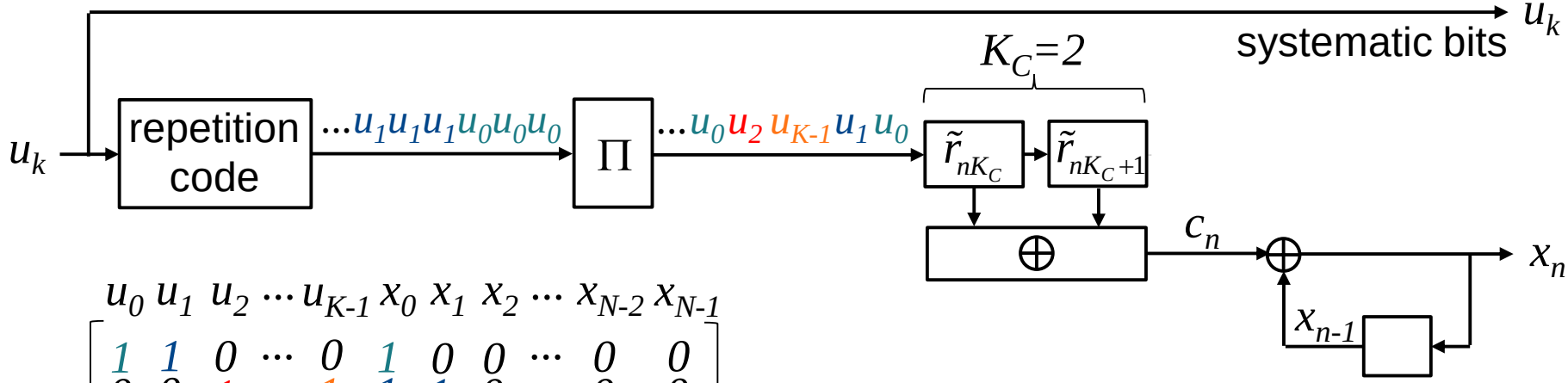
$$x_n = c_n \oplus x_{n-1}$$

$$\Rightarrow c_n = x_n \oplus x_{n-1}$$

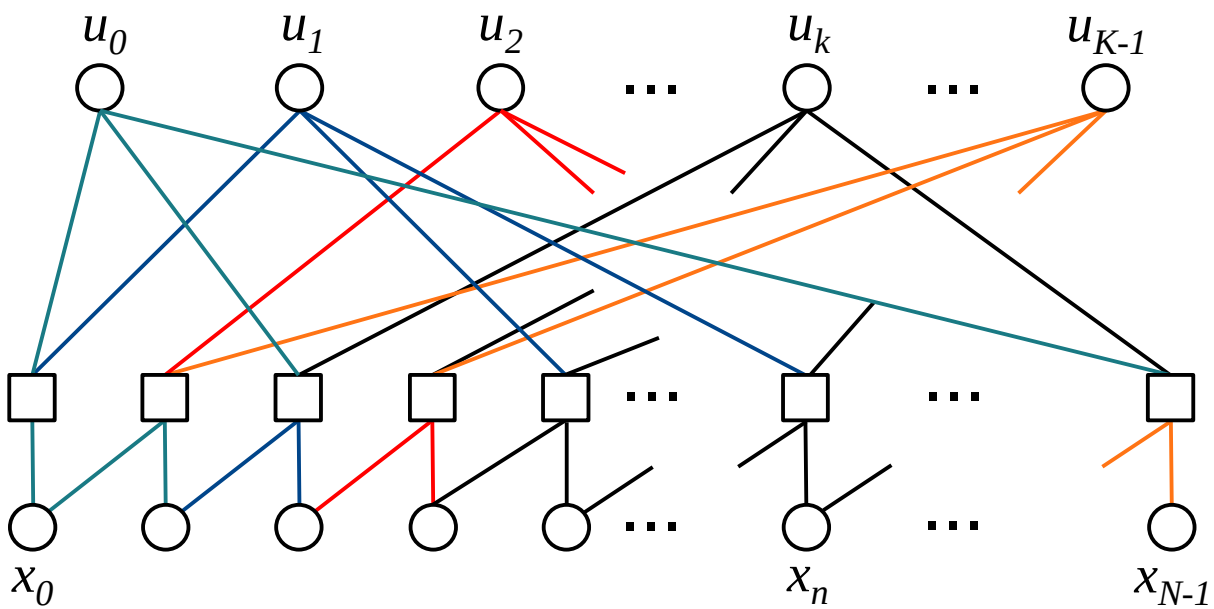
$$\tilde{r}_{nK_C} \oplus \tilde{r}_{nK_C+1} \oplus x_n \oplus x_{n-1} = 0$$



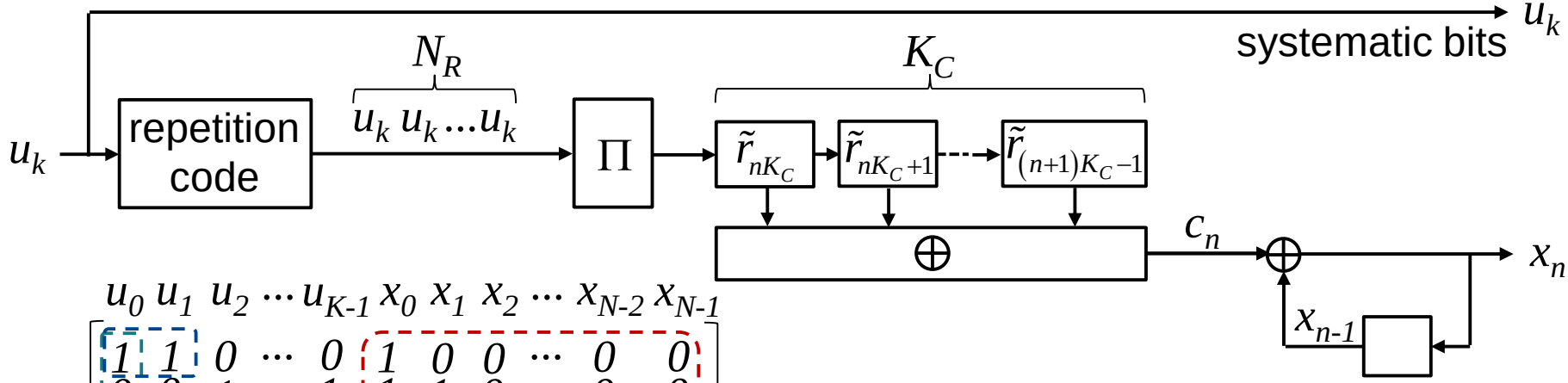
Repeat Accumulate Code



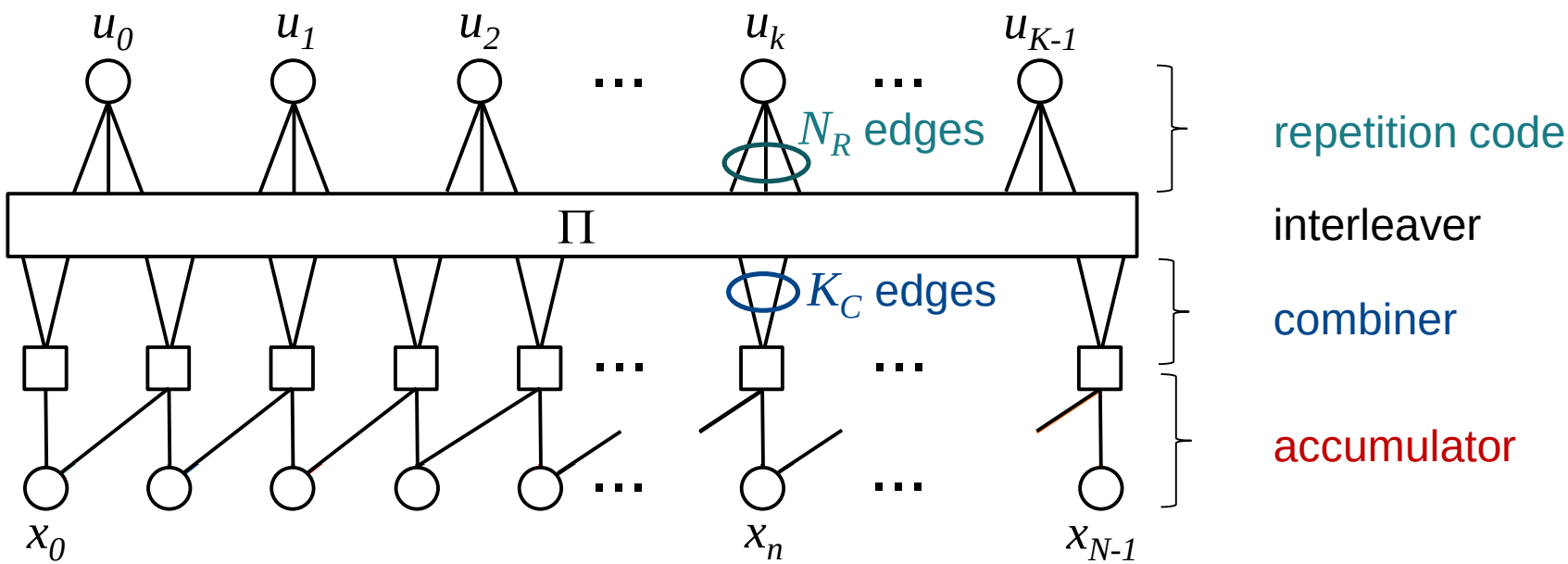
$$\mathbf{H}^T = \begin{array}{c} \begin{matrix} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & & 0 & 0 & 1 & 1 & & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \ddots & 1 & 1 \end{bmatrix} \end{array}$$



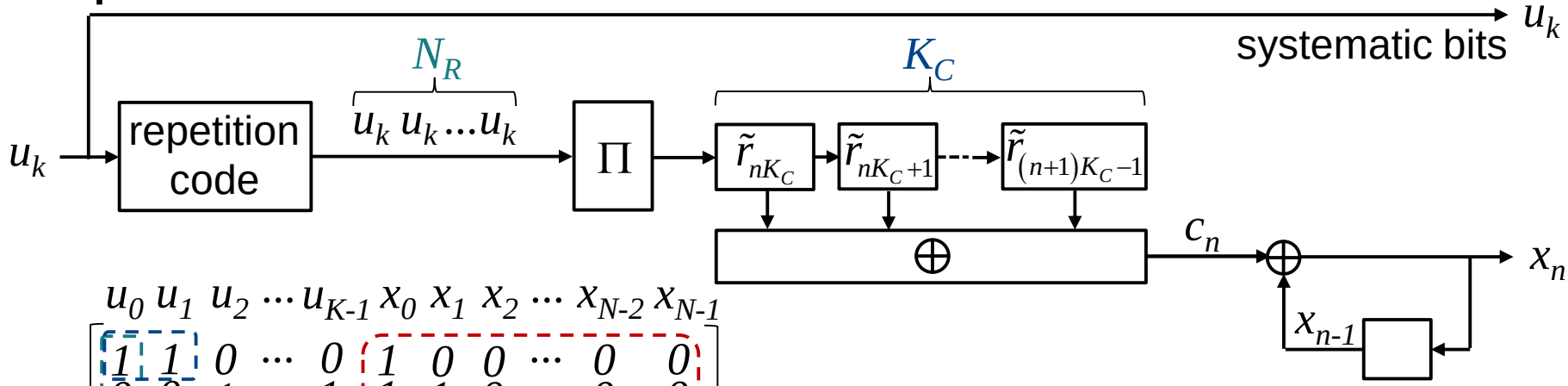
Repeat Accumulate Code



$$\mathbf{H}^T = \begin{array}{c} \begin{matrix} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & & & & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \ddots & 1 & 1 \end{bmatrix} \end{array}$$



Regular and Irregular Repeat Accumulate Codes



$$\mathbf{H}^T = \begin{array}{c} \begin{matrix} u_0 & u_1 & u_2 & \cdots & u_{K-1} & x_0 & x_1 & x_2 & \cdots & x_{N-2} & x_{N-1} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & & & & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \ddots & 1 & 1 \end{bmatrix} \end{array}$$

column weight:

$$\frac{1}{R_R} = N_R$$

row weight:

$$R_C = K_C$$

column weight:

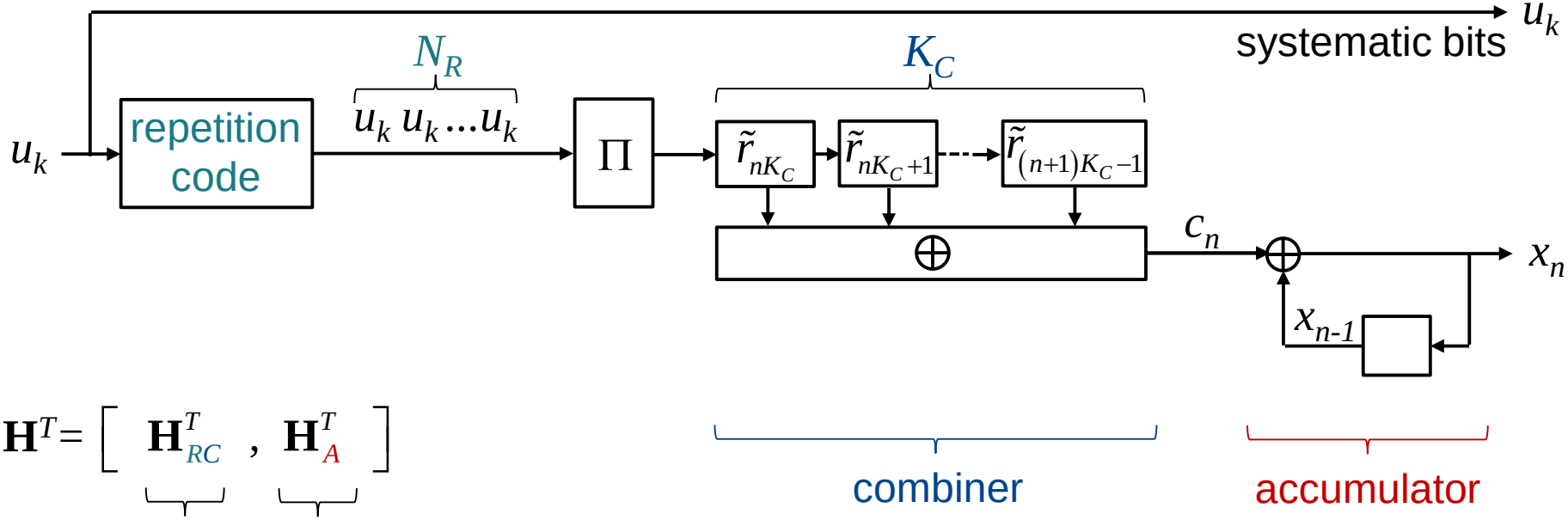
$2 \quad \ldots \quad 1$ → irregular, but:

A repeat accumulate code is called a *regular code*, if both

- the rate $R_R = 1/N_R$ of the repetition code and
- the rate $R_C = K_C$ of the combiner

are constant over the whole codeword. Otherwise, the repeat accumulate code is called an *irregular code*.

Repeat Accumulate Code



$$\mathbf{H}^T = \left[\ \underbrace{\mathbf{H}_{RC}^T}_{m \times K}\ ,\ \underbrace{\mathbf{H}_A^T}_{m \times m}\ \right]$$

combiner

accumulator

Message Passing Decoding of Repeat Accumulate Codes (1)

In order to perform message passing decoding, the repeat accumulate code needs to be represented by a Tanner graph. In order to obtain the Tanner graph, we first determine a parity check matrix $\mathbf{H}$ for the repeat accumulate code.

In general, the parity check matrix is of the following form:

$$\mathbf{H}^T = \left[\mathbf{H}_{RC}^T, \mathbf{H}_A^T\right] = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\underbrace{\qquad\qquad}_{\mathbf{H}_{RC}^T} \quad \underbrace{\qquad\qquad\qquad\qquad}_{\mathbf{H}_A^T}$$

The $m \times K$ submatrix $\mathbf{H}_{RC}{}^T$ corresponds to the repetition code and to the combiner, whereas the $m \times m$ matrix $\mathbf{H}_A{}^T$ corresponds to the accumulator.

Message Passing Decoding of Repeat Accumulate Codes (2)

Each column of the $m \times K$ submatrix $\mathbf{H}_{RC}{}^T$ represents an information bit u_k. The one entries in the k-th column of $\mathbf{H}_{RC}{}^T$ indicate in which parity checks the bit u_k is involved. The number of ones in the k-th column of $\mathbf{H}_{RC}{}^T$ is equal to $N_R=1/R_R$, where R_R is the rate of the repetition code part of the repeat accumulate code. Hence, the column weight of $\mathbf{H}_{RC}{}^T$ (row weight of $\mathbf{H}_{RC}$) is determined by the rate of the repetition code.

The number of ones in each row of $\mathbf{H}_{RC}{}^T$ is given by the number K_C of bits, which are combined by the combiner. Hence, the row weight of $\mathbf{H}_{RC}{}^T$ (column weight of $\mathbf{H}_{RC}$) is determined by the rate $R_C=K_C$ of the combiner.

The positions of the ones in $\mathbf{H}_{RC}{}^T$ are determined by the interleaver Π.

The accumulator submatrix $\mathbf{H}_A{}^T$ is characterized by a band structure. Each column of $\mathbf{H}_A{}^T$ represents a non-systematic code bit x_n. The accumulator matrix $\mathbf{H}_A{}^T$ has two one entries in each column except for the last column which has only a single one entry. Each row in $\mathbf{H}_A{}^T$ has two one entries except for the first row, which has only a single one entry. Consequently, the accumulator always causes a repeat accumulate code to be an irregular LDPC code in the strict sense. Remember that a regular code is defined by a Tanner graph, where all bit nodes have the same degree d_b and all check nodes have the same degree d_c. This corresponds to a parity check matrix $\mathbf{H}^T$ with constant column weight d_b and constant row weight d_c. If row weight and column weight of $\mathbf{H}^T$ are not constant, the LDPC code is called an irregular code.

In the special case of repeat accumulate codes, this strict definition is a bit relaxed: Even though the accumulator submatrix $\mathbf{H}_A$ is always irregular, we call a repeat accumulate code a regular code, if the submatrix $\mathbf{H}_{RC}$ describes a regular code, i.e. if $\mathbf{H}_{RC}$ has constant row weight $N_R=1/R_R$ and constant column weight $R_C=K_C$.

Message Passing Decoding of Repeat Accumulate Codes (3)

Similar as for general LDPC codes, an irregular repeat accumulate code can achieve a lower decoding threshold, i.e. the waterfall region of the BER vs. SNR curve will occur at a lower SNR. However, the lower decoding threshold typically comes on the expense of a higher error floor.

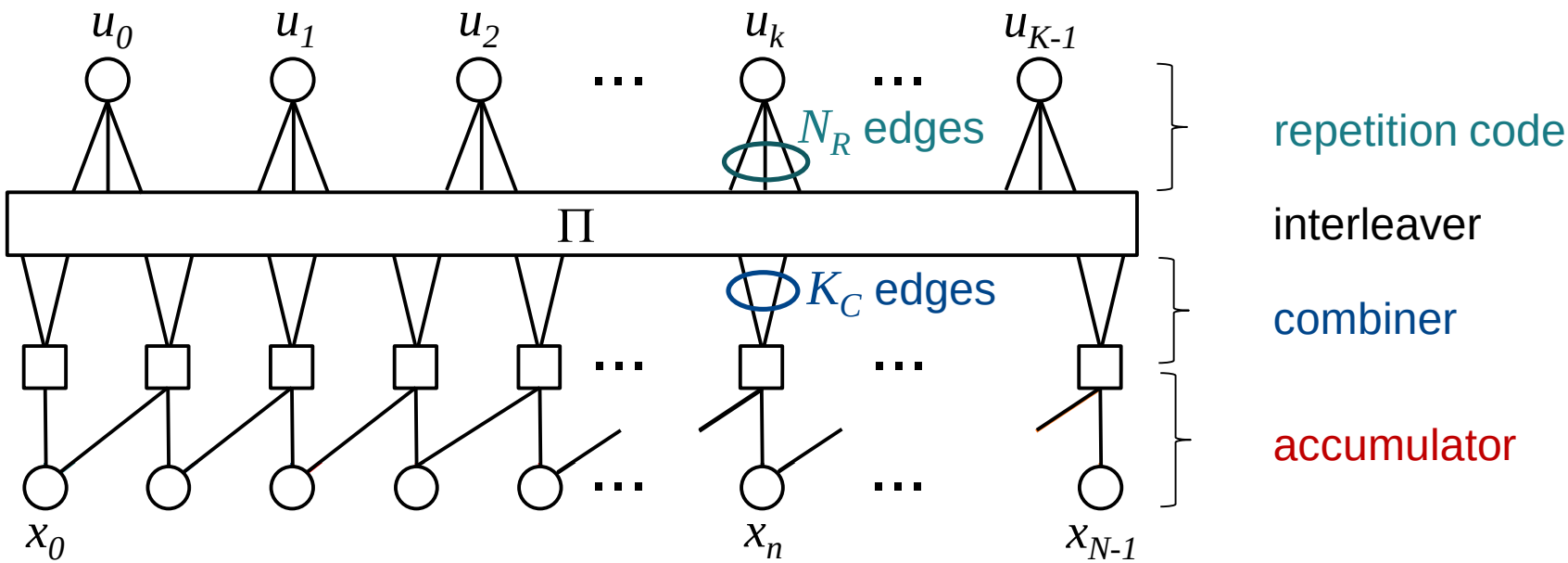
For good performance of a message passing decoder, the Tanner graph should avoid short cycles. This can be achieved by proper design of the interleaver Π.

Also, construction methods known from LDPC codes can be applied to repeat accumulate codes. In this case, the accumulator part in the Tanner graph is kept untouched in order to preserve the accumulator structure. However, we can apply the progressive edge growth algorithm (PEG) or the concept of protographs to the upper part of the Tanner graph, i.e. the edges between systematic bit nodes and check nodes.

Message Passing Decoding of Repeat Accumulate Codes (3)

An irregular repeat accumulate code varies the rate R_R of the repetition code and/or the rate R_C of the combiner over a codeword of the overall code.

The parity check matrix $\mathbf{H}$ of a repeat accumulate code can be represented by the following Tanner graph:



In the Tanner graph above, we distinguish between two types of bit nodes: The bit nodes at the top of the Tanner graph represent the information bits, i.e. the systematic code bits. The bit nodes at the bottom of the Tanner graph represent the non-systematic code bits (parity bits), i.e. the output bits of the accumulator.

Message Passing Decoding of Repeat Accumulate Codes (4)

Message passing decoding can be performed as explained earlier in the course: The bit nodes are initialized by the channel information $L_c(u_k)$ and $L_c(x_n)$, respectively. Then, messages are exchanged along the edges of the Tanner graph. In case of non-systematic repeat accumulate codes, the bit nodes representing the systematic bits u_k on top of the Tanner graph are initialized by $L_c(u_k)=0$.

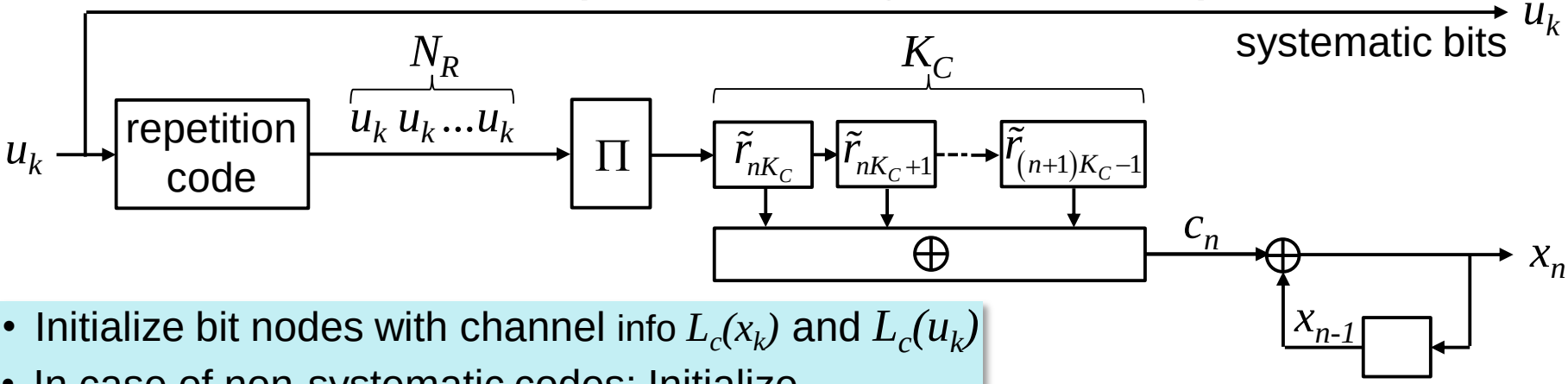
The representation of the Tanner graph above allows for different scheduling strategies for message passing:

One option is to pass messages from all bit nodes to the check nodes first. After updating in the check nodes, messages are passed back to the bit nodes. These two steps constitute one iteration of the decoding process.

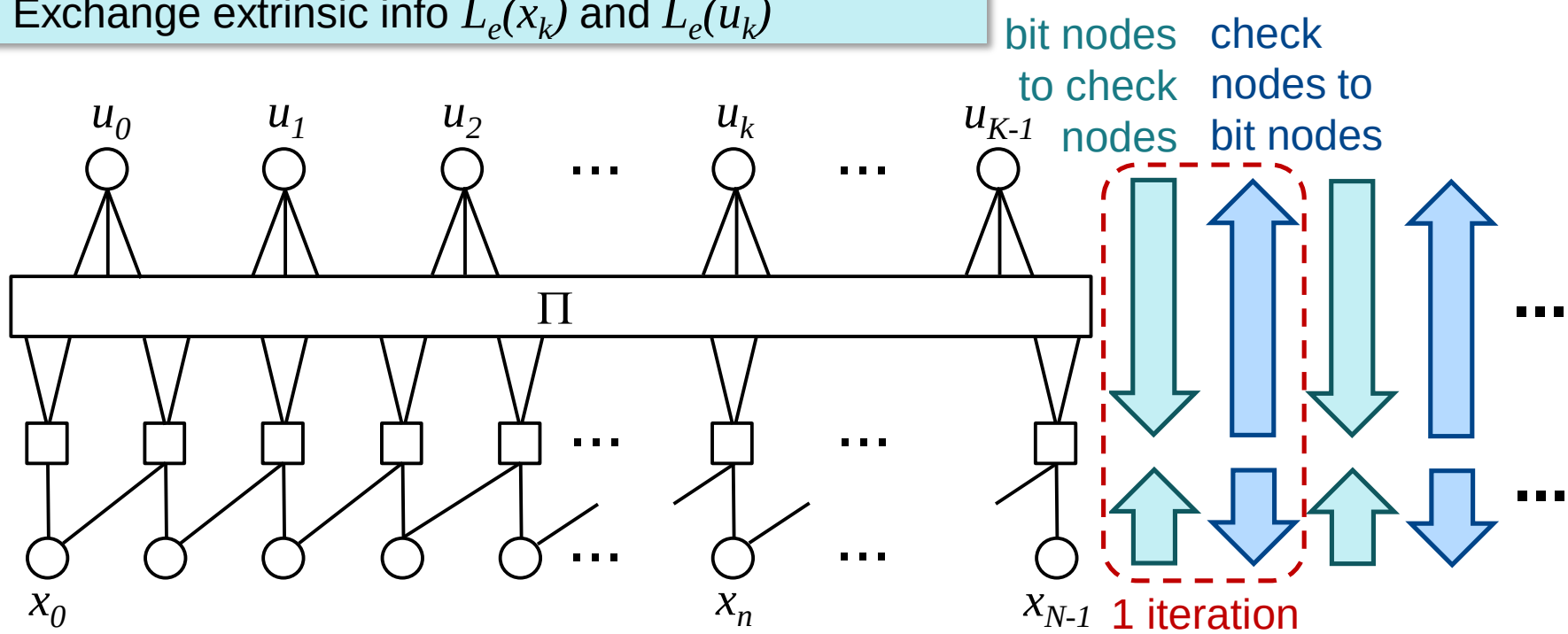
Another option would be to pass first messages from the systematic bit nodes at the top of the Tanner graph to the check nodes. Then, the messages are updated in the check nodes and passed to the parity bit nodes at the bottom of the Tanner graph. In the next step, updated messages are passed from the parity bit nodes to the check nodes. After updating, messages are passed from the check nodes to the systematic bit nodes. In this scheduling strategy, four steps constitute one complete iteration of the decoding process.

Typically, both scheduling strategies will yield comparable BER performance. However, they may differ in the number of required iterations and, hence, in decoding complexity. The choice which strategy is preferable has to be taken case by case depending on the particular code properties and channel characteristics.

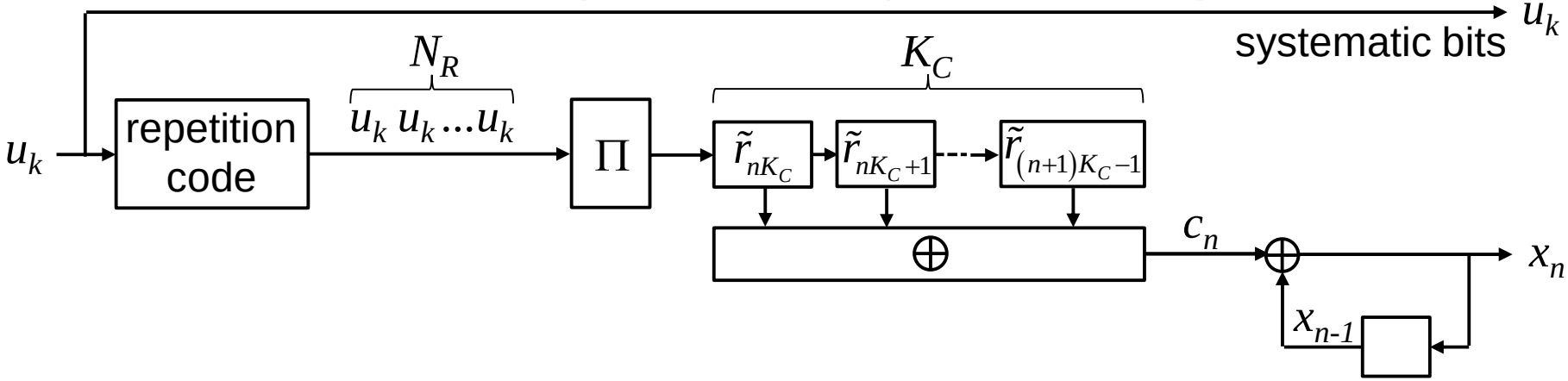
Message Passing Decoding of Repeat Accumulate Codes (Scheduling Version 1)

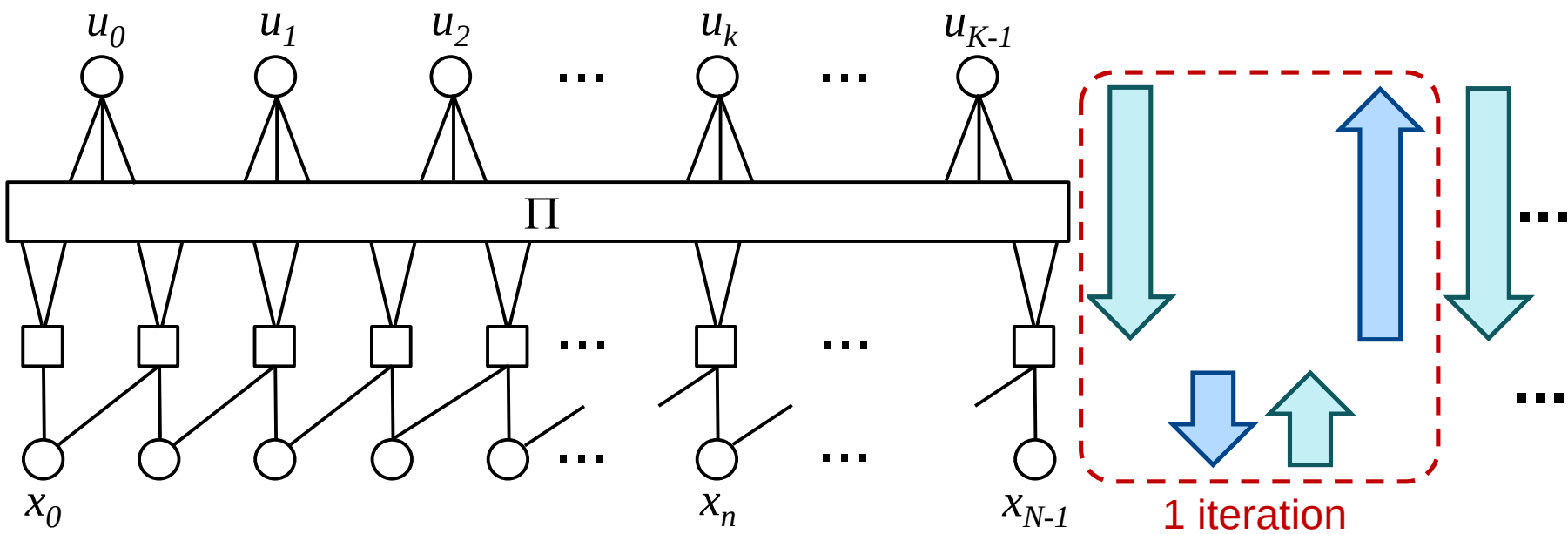


- Initialize bit nodes with channel info $L_c(x_k)$ and $L_c(u_k)$
- In case of non-systematic codes: Initialize systematic bit nodes with $L_c(u_k)=0$
- Exchange extrinsic info $L_e(x_k)$ and $L_e(u_k)$



Message Passing Decoding of Repeat Accumulate Codes (Scheduling Version 2)





Higher Weight Repeat Accumulate Codes

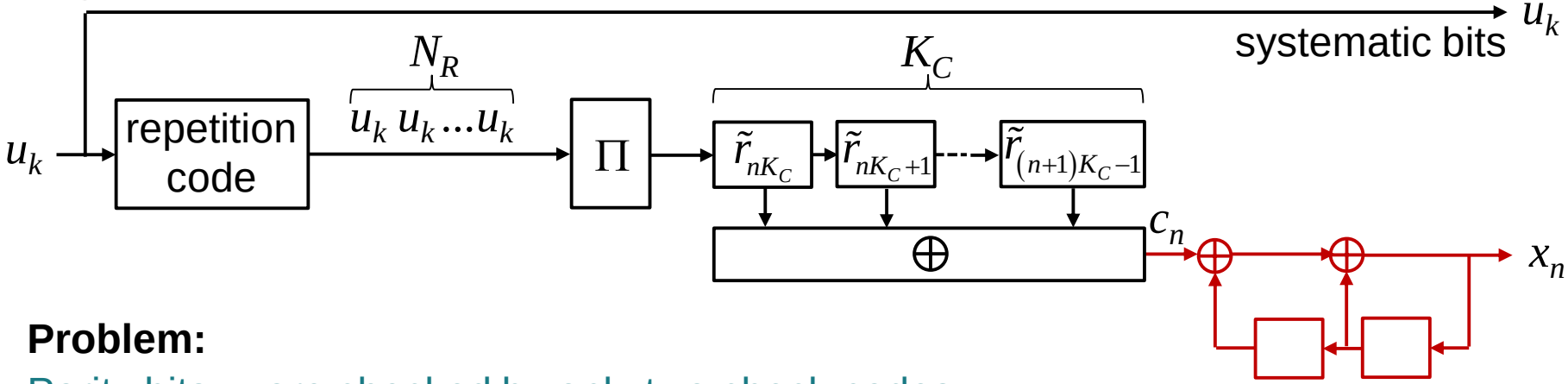
As for all iterative decoding schemes, the BER curve of a repeat accumulate code shows an error floor. I.e. after a steep drop of the BER vs. SNR curve in the waterfall region, the BER decreases only slowly when the SNR is further increased. The error floor is a problem particularly in applications which require a very low guaranteed BER. This is e.g. the case in fiber optical communications, where often a BER of about 10^{-15} or lower is required.

The error floor of a repeat accumulate code can be improved, if the accumulator is replaced by a recursive filter of higher order.

This is due to the fact that bit nodes of degree two contribute significantly to a higher error floor since the respective bit is checked by only two check nodes. As can be seen from the column weight of the accumulator part $\mathbf{H}_A{}^T$ in the parity check matrix $\mathbf{H}^T$, or from the Tanner graph above, the bit nodes at the bottom of the Tanner graph representing the output bits of the accumulator have degree two (except for the last bit node which has degree one). The impact of those degree two bit nodes is the stronger the higher the code rate. This is due to the fact that for high rate codes, the accumulator part takes a larger portion of the overall Tanner graph or parity check matrix $\mathbf{H}$, respectively.

A recursive filter of higher order adds edges to the parity bit nodes resulting in a reduced number of low degree bit nodes and, consequently, lower error floor. The additional complexity of message passing decoding due to the larger number of edges in the Tanner graph is small.

Repeat Accumulate Codes with Higher Order Precoder



Problem:

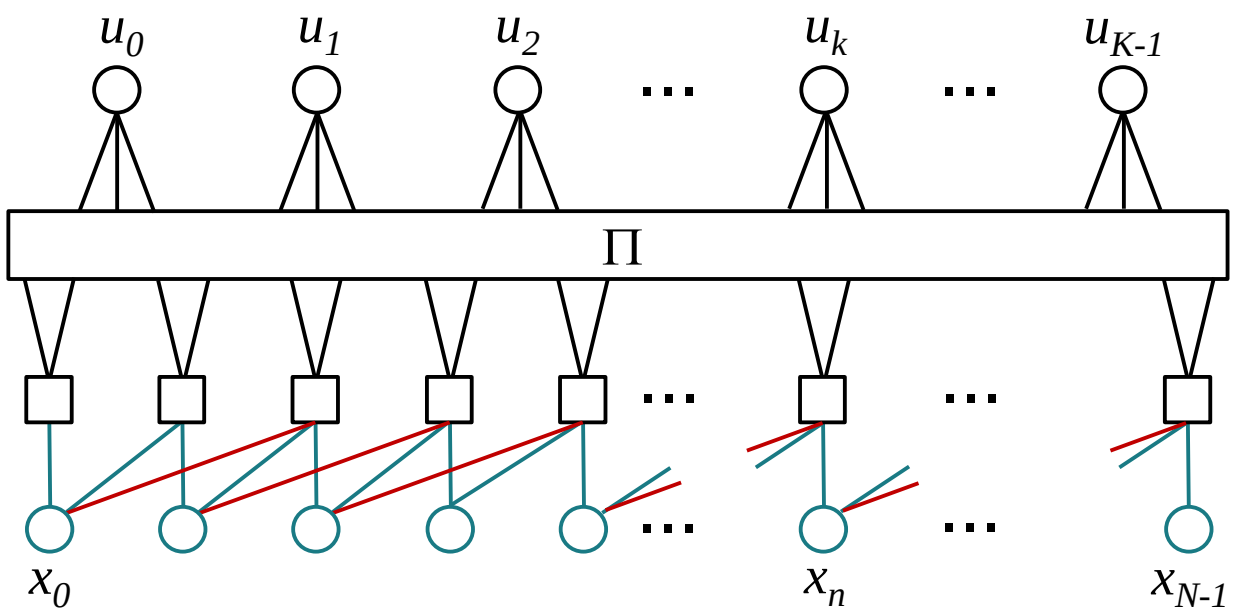
Parity bits x_k are checked by only two check nodes

→ high error floor

→ use higher order precoder as inner code

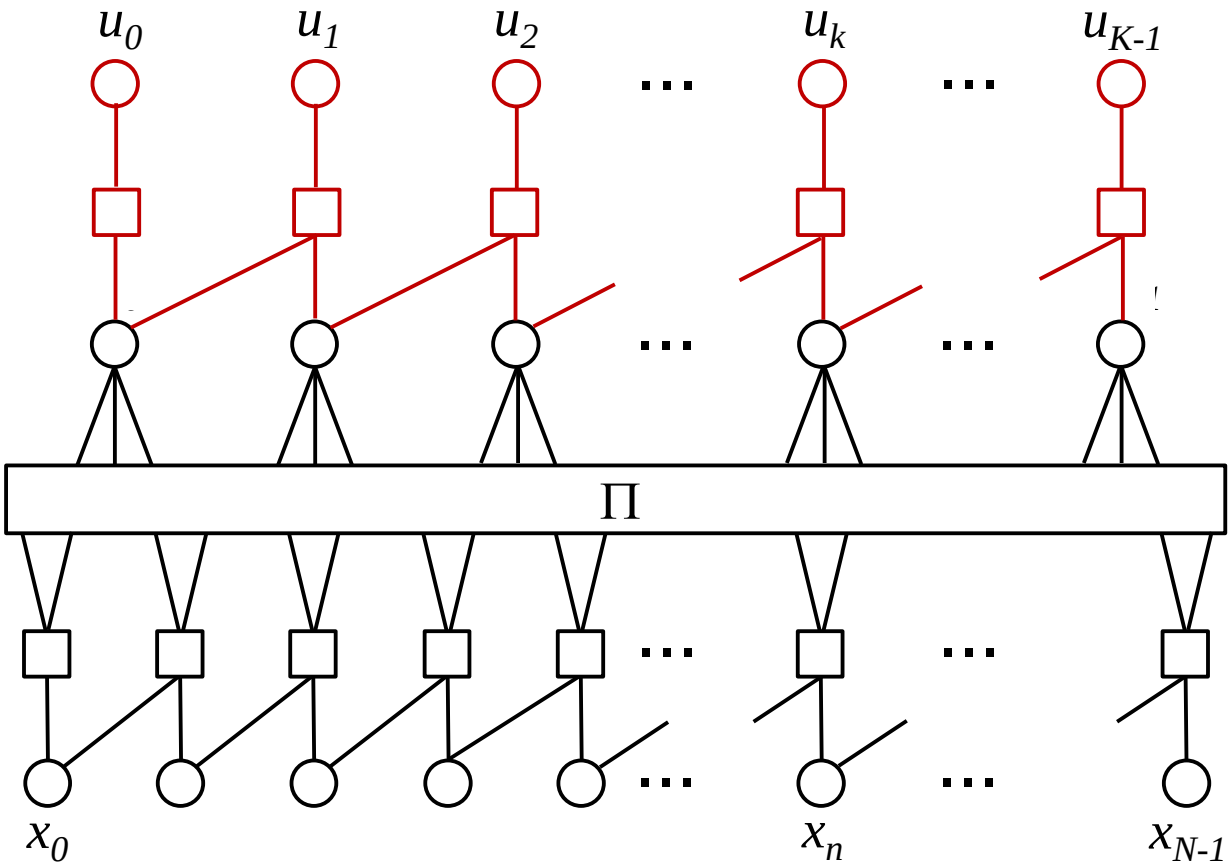
$$x_n = c_n \oplus x_{n-1} \oplus x_{n-2}$$

$$c_n \oplus x_n \oplus x_{n-1} \oplus x_{n-2} = 0$$



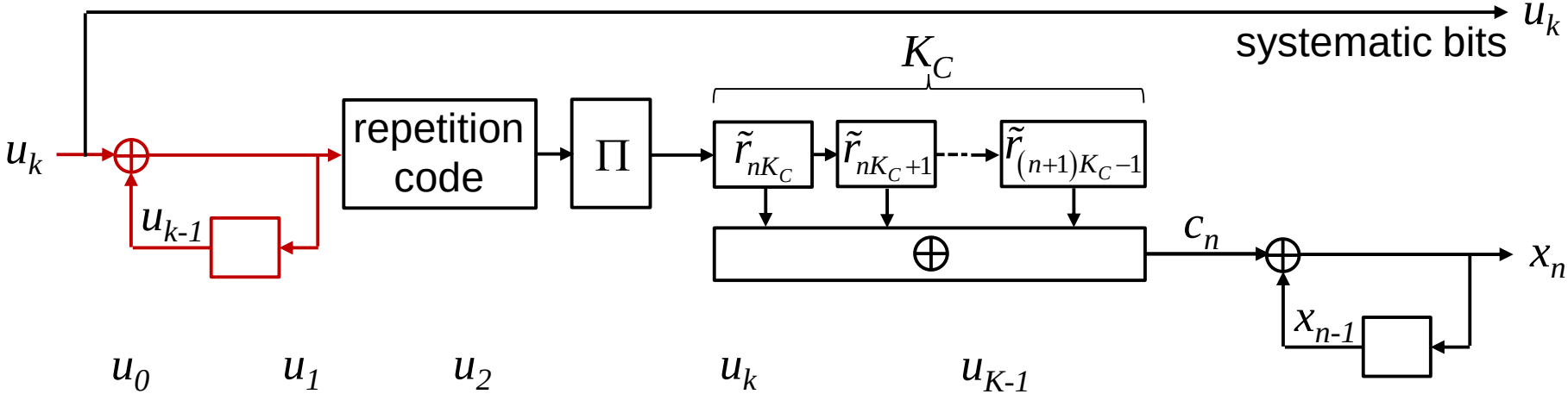
Accumulate Repeat Accumulate Codes

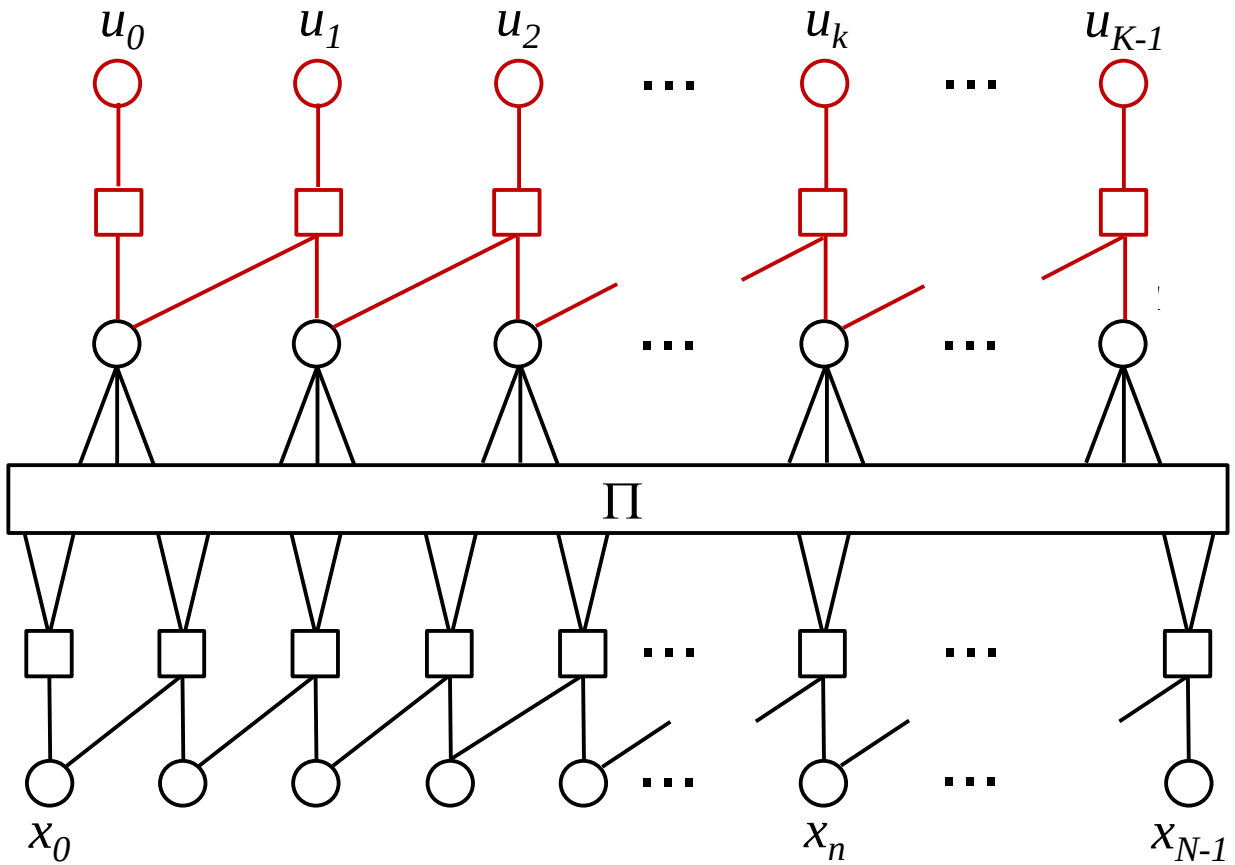
The convergence threshold of the iterative decoder, i.e. the SNR at which the waterfall region in the BER vs. SNR curve occurs, can be shifted towards lower SNR by adding another accumulator before the repetition encoder. In the Tanner graph, a regular structure representing the accumulator occurs also at the top of the Tanner graph:



In literature, further extensions can be found, where even more accumulators are used in the encoder. This results e.g. in *accumulate repeat accumulate accumulate codes*. In some proposals, only a fraction of the information bits are input to the first accumulator while the remaining information bits are fed directly into the repetition encoder.

Accumulate Repeat Accumulate Codes





Advantage:
Message passing decoding converges at lower SNR threshold

Disadvantage:
Slightly increased decoding complexity due to extra edges in Tanner graph

Turbo Decoding of Repeat Accumulate Codes (1)

Repeat accumulate codes can also be decoded using the *turbo principle*. The turbo principle can be applied whenever we have a concatenation of decoders or other detection stages in a receiver.

Let us first consider a repeat accumulate code without a combiner. In this case, the encoder is the serial concatenation of a repetition code and an accumulator separated by an interleaver. The respective decoder consists of the respective decoders for the two constituent codes.

The turbo decoder first decodes the accumulator. An optimum soft output a posteriori probability (APP) decoder for the accumulator is the BCJR algorithm[1]. The BCJR decoder uses channel information $L_c(c_n)$ about the bits c_n, which have been input to the accumulator as its input. Based on this channel information, it delivers a posteriori log-likelihood ratios $L(\hat{c}_n)$ about the bits c_n. The L-values $L(\hat{c}_n)$ contain channel information and the extrinsic information obtained by the accumulator decoder. The a posteriori log-likelihood ratios $L(\hat{c}_n)$ are deinterleaved and passed to the repetition decoder. The repetition decoder delivers not only preliminary estimates $L(\hat{u}_k)$ on the information bits u_k but also a posteriori log likelihood ratios $L(\hat{r}_l)$ on the code bits r_l of the repetition code. The a posteriori L-values $L(\hat{r}_l)= L_c(r_l)+ L_e(r_l)$ consist of the channel information $L_c(r_l)= L(\hat{c}_n)$ received from the accumulator decoder via the deinterleaver and the extrinsic information $L_e(r_l)$ obtained by the repetition decoder.

[1]BCJR algorithm: Optimum APP algorithm named after the inventors **B**ahl, **C**ocke, **J**elinek, **R**aviv. The BCJR algorithm can eg. be applied to soft-output decoding of convolutional codes or to soft-output equalization of frequency-selective channels.

Turbo Decoding of Repeat Accumulate Codes (2)

The idea of turbo decoding is to feed back information from the outer decoder, i.e. the repetition decoder, to the inner decoder, i.e. the accumulator decoder. The accumulator decoder will then make a second decoding attempt. However, unlike in the first decoding step, not only the stored channel information $L_c(c_n)$ is available at the accumulator decoder input, but the accumulator decoder can take the fed back information into account as an additional a priori information $L_a(c_n)$. This will help the accumulator decoder to provide a posteriori log-likelihood ratios $L(\hat{c}_n)$ which are improved compared to the decisions in the previous iteration. However, the essential trick in turbo decoding is, that only a part of the a posteriori information $L(\hat{r}_l)$ of the repetition decoder is fed back to the accumulator decoder: The a priori information $L_a(c_n)$ used by the accumulator decoder should be independent of previous decisions of the accumulator decoder. Therefore, only the new information obtained by the repetition decoder, i.e. the extrinsic information $L_e(r_l)$, is fed back to the accumulator decoder. The extrinsic information $L_e(r_l)$ is obtained by subtracting the input information $L_c(r_l) = L(\hat{c}_n)$, which was received from the accumulator decoder, from the a posteriori information $L(\hat{r}_l) = L_c(r_l) + L_e(r_l)$ of the repetition decoder:

$$L_e(r_l) = L_a(c_n) = L(\hat{r}_l) - L_c(r_l).$$

For the same reasons, the a priori information $L_a(c_n)$ is subtracted from the updated accumulator a posteriori information $L(\hat{c}_n)$ before passing it to the repetition decoder. The a posteriori log-likelihood ratios $L(\hat{c}_n) = L_a(c_n) + L_{c,e}(c_n)$ of the accumulator decoder now consists of channel information, extrinsic information and a priori information, where the a priori information $L_a(c_n) = L_e(r_l)$ was received from the repetition decoder via the turbo feedback.

Turbo Decoding of Repeat Accumulate Codes (3)

However, the updated log-likelihood ratios $L_c(r_l)$ which are passed to the repetition decoder should be independent of previous decisions of the repetition decoder. Therefore, the a priori information needs to be subtracted from the a posteriori log-likelihood ratios:

$$L_c(r_l) = L_{c,e}(c_n) = L(\hat{c}_n) - L_a(c_n).$$

Some iterations can be performed in that way until the log-likelihood ratios $L(\hat{u}_k)$ at the output of the repetition decoder are accepted as final decisions. We call one complete decoding of both consituent codes an iteration. After some iterations, the information which is exchanged among the constituent codes will not be independent anymore and further iterations will not improve the decoding result. The largest gain in terms of bit error rate is obtained in the first iteration, while the incremental gain is reduced for further iterations.

The principle of exchanging extrinsic information among constituent decoders is similar in spirit as message passing decoding. However, in particular the BCJR algorithm is significantly more complex than the operations in the bit nodes and check nodes in message passing decoding. On the other hand, turbo decoding typically requires significantly less iterations than message passing decoding. This compensates for the higher complexity in each iteration.

Often, five to eight iterations are sufficient for turbo decoding while 20 or 50 iterations have to be performed with message passing decoding.

The bit error rate (BER) performance of turbo decoding and message passing decoding is usually comparable.

Turbo Decoding of Repeat Accumulate Codes (4)

The name *turbo decoding* can be interpreted in analogy to a turbo engine: The not yet completely exploited information is fed back to the inner decoder in the turbo loop.

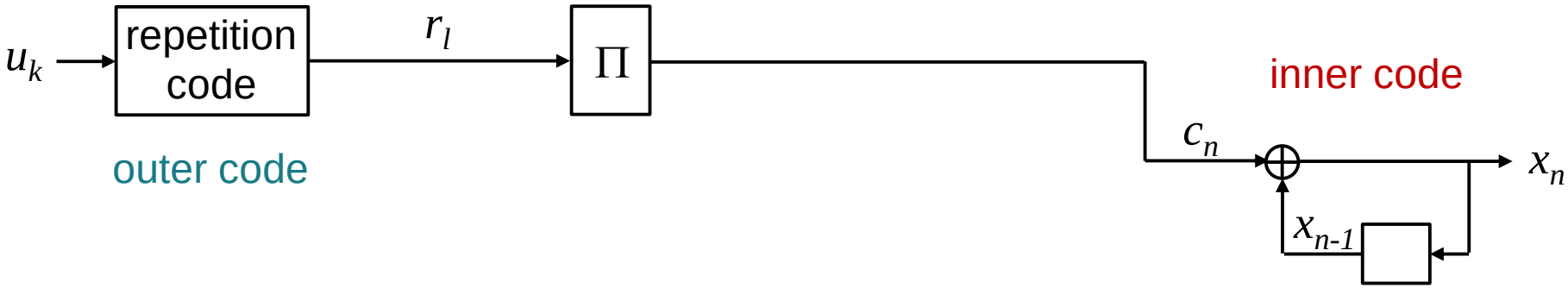
If the repeat accumulate code comprises also a combiner, turbo decoding becomes a bit more sophisticated. If a combiner is involved, the repeat accumulate code needs to be systematic. Otherwise, decoding of the combined bits is not possible.

Turbo decoding can now start by decoding the systematic bits based on the received channel information $L_c(u_k)$. In further iterations, it will additionally take the information received from the combiner decoder into account. The extrinsic and systematic channel information of the repetition decoder is encoded by a soft combiner encoder which produces the a priori information of the accumulator decoder. Since the soft combiner encoder operates on L-values, the XOR sums in the original encoder have to be replaced by boxplus operations.

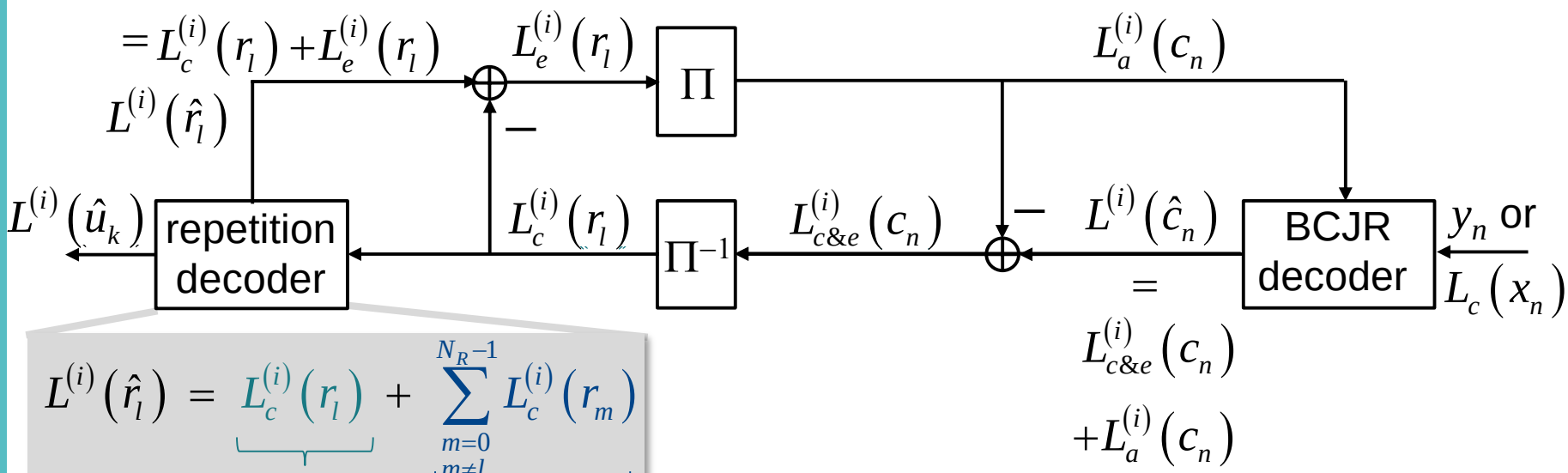
The extrinsic and channel information of the accumulator decoder is then passed to the combiner decoder.The combiner decoder uses the fed back extrinsic information from the repetition decoder as additional a priori information. Finally, the extrinsic and channel information of the combiner decoder is passed to the repetition decoder. Again some iterations can be performed until the log-likelihood ratios $L(\hat{u}_k)$ at the output of the repetition decoder are accepted as final decisions.

Turbo Decoding of Repeat Accumulate Codes

Encoder: Serial concatenation of repetition code and accumulator

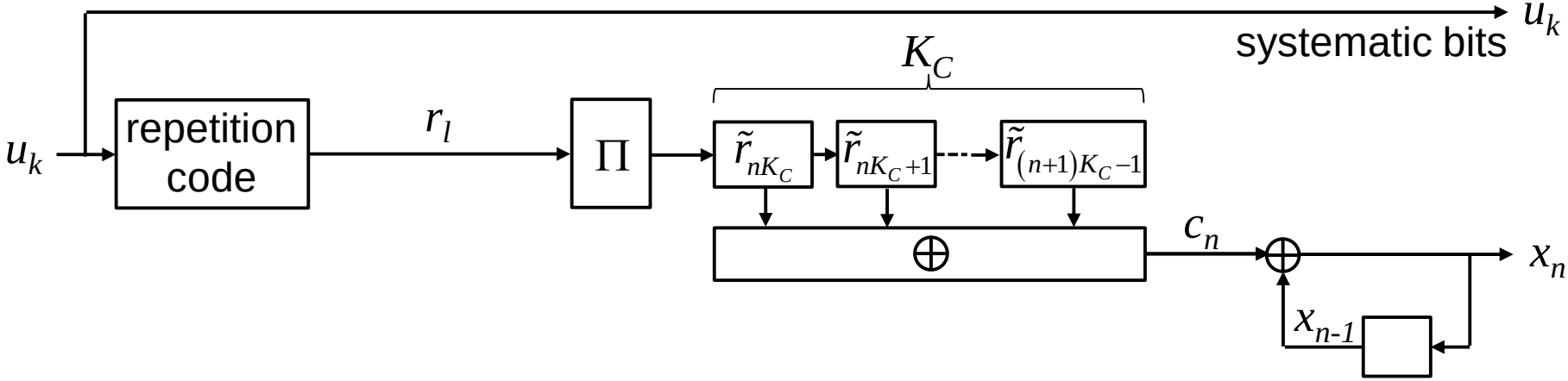


Turbo decoder:

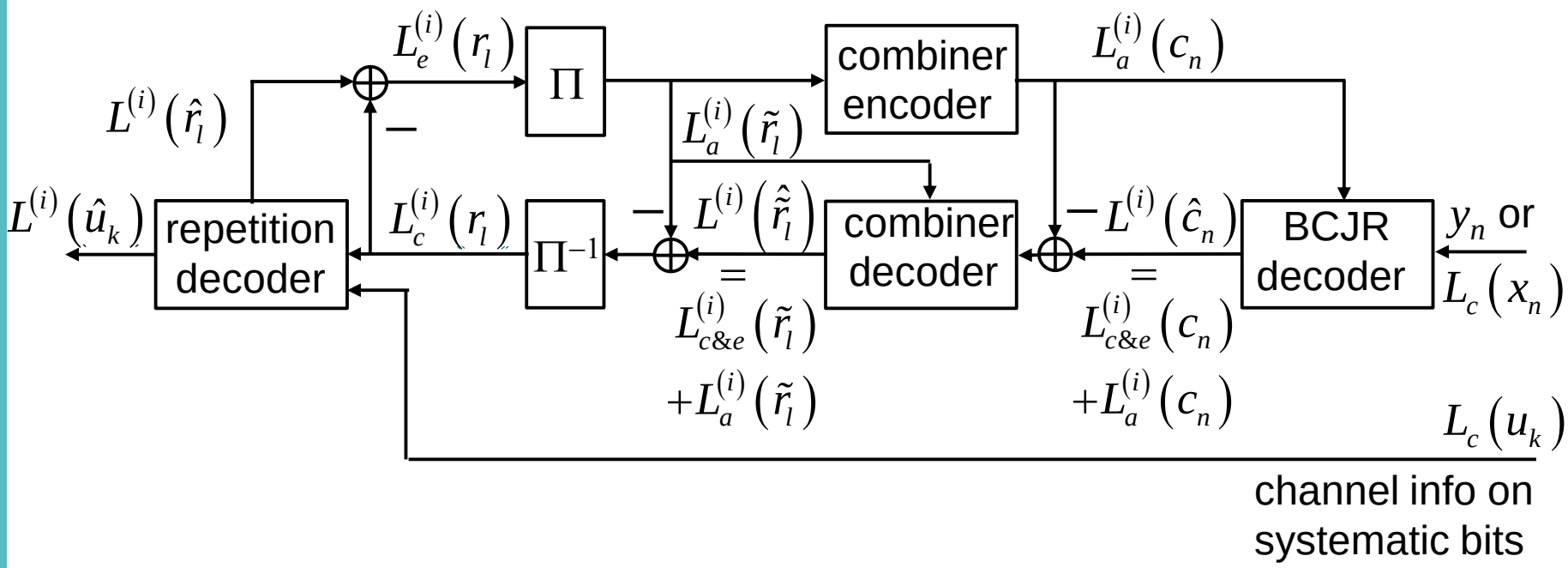


$$L^{(i)}(\hat{r}_l) = \underbrace{L_c^{(i)}(r_l)}_{\text{channel info}} + \underbrace{\sum_{\substack{m=0 \\ m \neq l}}^{N_R-1} L_c^{(i)}(r_m)}_{\text{extrinsic info}}$$

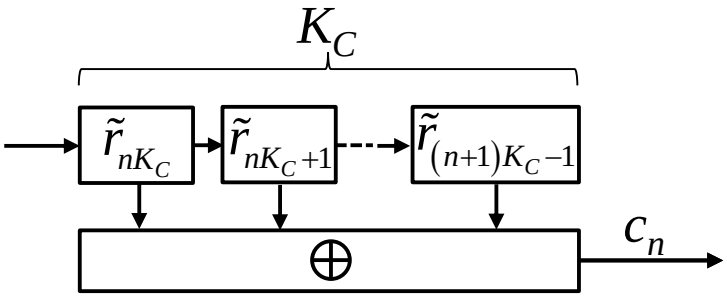
Turbo Decoding of Repeat Accumulate Codes



Turbo decoder:



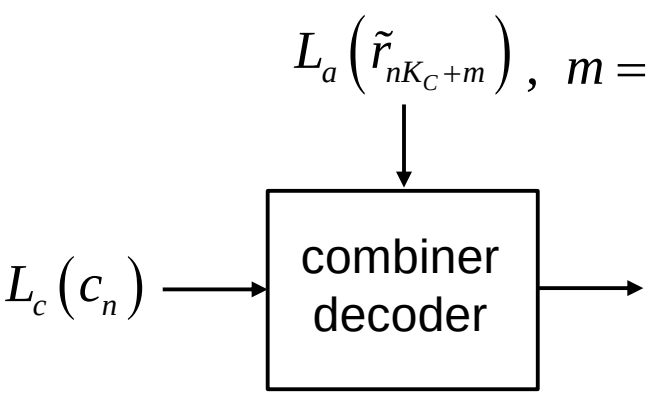
Soft-In Soft-Out Combiner Encoder and Decoder



$$c_n = \sum_{m=0}^{K_C-1}{}^{\oplus} \tilde{r}_{nK_C+m}$$

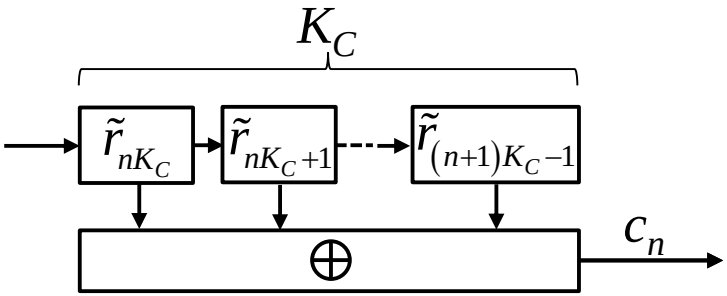
Soft-in soft-out combiner encoder: $L(\hat{c}_n) = \sum_{m=0}^{K_C-1}{}^{\boxplus} L(\tilde{r}_{nK_C+m})$

Soft-in soft-out combiner decoder:

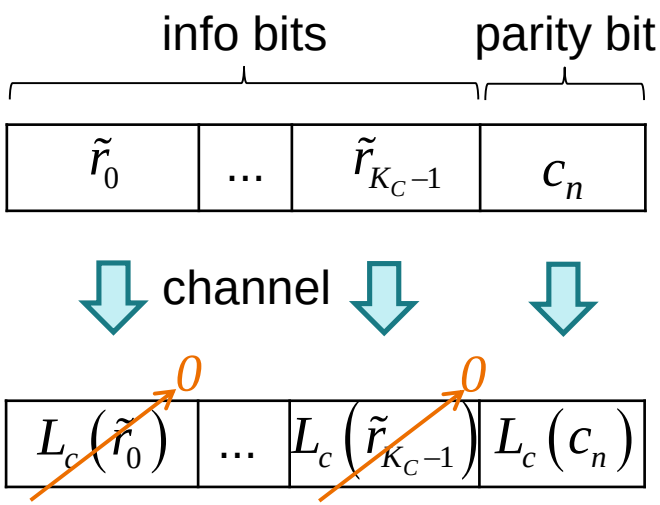


$$L(\hat{\tilde{r}}_{nK_C+l}) = \underbrace{L_c(c_n) \boxplus \sum_{\substack{m=0 \\ m\neq l}}^{K_C-1}{}^{\boxplus} L_a(\tilde{r}_{nK_C+m})}_{\text{extrinsic info}} + \underbrace{L_a(\tilde{r}_{nK_C+l})}_{\text{a priori info}}$$

Soft-In Soft-Out Single Parity Check Decoder



$$c_n = \sum_{m=0}^{K_C-1} \oplus \tilde{r}_{nK_C+m}$$



Combiner:
Channel info is only available for parity bit c_n

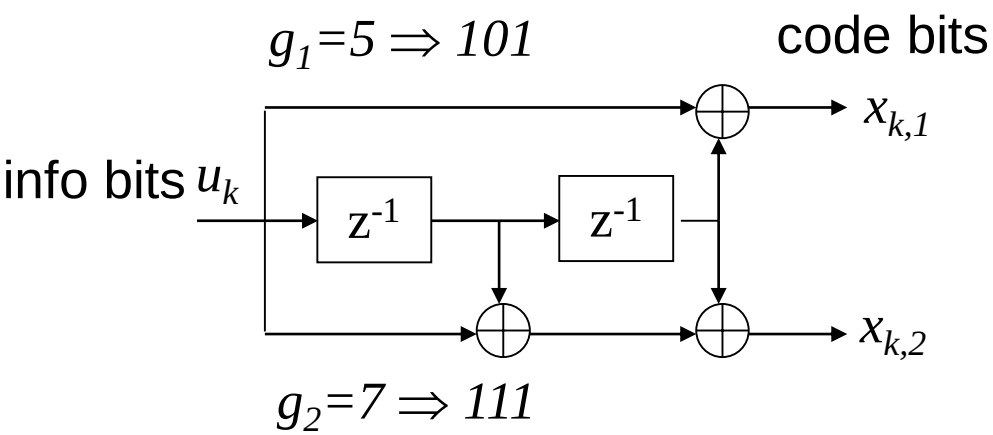
$$L(\hat{\tilde{r}}_l) = \underbrace{L_c(\tilde{r}_l)}_{\text{channel info}} + \underbrace{L_a(\tilde{r}_l)}_{\text{a priori info}} + \underbrace{\sum_{\substack{m=0 \\ m\neq l}}^{K_C-1} \boxplus \left[L_c(\tilde{r}_m) + L_a(\tilde{r}_m) \right]}_{\text{extrinsic info}} \boxplus L_c(c_n)$$

$L_c(\tilde{r}_l) \to 0$, $L_c(\tilde{r}_m) \to 0$

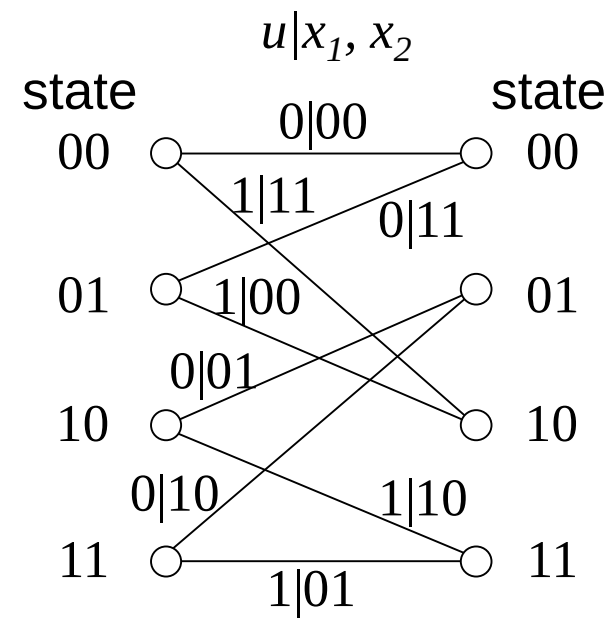
Convolutional Codes

A continuous info bit stream is encoded rather than a block of information bits. The encoder is a shift register the taps of which are determined by generators $\mathbf{g}$. The codewords differ at least in d_{free} positions, where d_{free} is the free distance of the code.

Example: $R=1/2$,
constraint length $C=3$ (memory $M=C-1=2$)
generators $\mathbf{g}=[5,7]$



Trellis diagram:



The decoder finds the most likely path through the trellis using e.g. the Viterbi algorithm or the BCJR algorithm.

Convolutional Codes

Convolutional codes are a class of codes with high practical relevance. Convolutional codes encode a sequence of information symbols into a sequence of code symbols. In principle, the sequences can be of infinite length. However, in practical systems, convolutional codes operate on finite blocks of information symbols. Hence, convolutional codes may also be regarded as a kind of block codes.

The encoder of a convolutional code can be realized by a shift register. This shift register performs the convolution of the info sequence with a set of generator coefficients.

In contrast to block codes, the generator coefficients are usually obtained by computer search rather than by analytical methods.

The number of memory elements of the shift register is called the *memory* M of the convolutional code. Since each code symbol depends on the $M+1$ previous info symbols, the number $M+1$ is called the *constraint length* of the code.

K info symbols are input to the shift register at a time and N code symbols are output of the shift register at a time step. Since N code symbols are generated per K info symbols, the coder rate is defined by $R=K/N$.

The symbols which are content of the memory elements of the shift register determine the state of the encoder. For an q-ary info symbol alphabeth, there are q^M different states.

For encoding of the first info symbols, the encoder must start from a predetermined state, i.e. the memory elements of the shift register must contain known symbols before the first info symbol is shifted in. The choice of the start state is arbitrary. Usually, we initialize the shift register with zeros, i.e. the encoder starts from the all-zero state.

Convolutional Codes

The last M info symbols of a block have less protection since they have impact on a lower number of codebits. E.g. in a rate $1/N$ convolutional code, the last info symbol has impact only on the last N code symbols. In contrast, an info symbol in the middle of the block has impact on $N(M+1)$ code symbols. This clearly enables better error correction.

In order to provide the same error protection to the last symbols of an info block, the code is usually terminated, i.e. M known symbols are appended to the info block such that the encoder shift register is forced to a predetermined state. Usually, this is the all-zero state. For non-recursive convolutional codes, termination in the zero state is achieved by appending M zero symbols to the info block. The M termination symbols contain no information. However, encoding of M additional symbols results in MN additional code bits, which have to be transmitted. Consequently, termination of convolutional codes improves performance on the expense of reduced rate. However, for block sizes $L \gg M$, the rate loss due to termination is negligible.

The encoding process can be illustrated in a trellis diagram. Therefore, convolutional codes are also called *trellis codes*. The trellis is determined by the q^M possible states and the possible state transitions. Each state transition is labelled with the info symbols which cause this state transition and with the code symbols which are generated by the encoder if the respective info symbols are shifted into the encoder shift register. Each shift operation corresponds to a trellis segment. The encoding process is viewed by a sequence of trellis segments.

Each info sequence defines a particular path through the trellis. The corresponding code sequence can be read from the respective transition labels of that path.

Convolutional Codes

Since the encoder starts from the zero state, the trellis does not contain all possible state transitions in the first M segments. Also, if the code is terminated, the last M trellis segments do only contain those transitions, which lead to the zero state within the last M segments.
In order to characterize the convolutional code, we can plot all possible state transitions into one trellis segment.
The trellis is conveniently used for decoding. For that purpose, we illustrate not only the (to the decoder unknown) path which corresponds to the current info sequence, but we show all possible paths through the trellis. It is the task of the decoder to find the most likely of those paths. The maximum likelihood path can be found by the Viterbi algorithm. Searching for the maximum likelihood path through the trellis (maximum likelihood sequence estimation). minimizes the code sequence error probability. Another decoding principle is symbol-by-symbol decoding which searchs for the most likely info symbols in each trellis segment rather than for the complete maximum likelihood sequence. This decoding principle minimizes the symbol error probability. The resulting decoded info sequence is not necessarily equal to the maximum likelihood sequence.
Since the decoding complexity depends on the number of trellis states which grows exponentially with growing code memory, the Viterbi algorithm is not feasible for decoding of convolutional codes with very high memory. Convolutional codes with very high memory such as M=100 can be decoded by sequential algorithms. Examples for sequential decoding algorithms are the Fano algorithm or the stack algorithm. The problem of sequential algorithms is that the decoding delay depends on the error pattern. Therefore, they may not finish decoding within a given time. If they are able to decode, they deliver the ML sequence. A possible implementation of sequential decoding is to fix the decoding delay (the decoding complexity) and to accept outage if the decoder is unable to decode within the predetermined decoding delay.

Viterbi Algorithm

The Viterbi algorithm performs maximum likelihood decoding based on the trellis representation of a convolutional code. It works through the trellis segment by segment starting at the beginning of the trellis. For all states, it computes a metric for each path which passes through a certain state at the end of the current segment („time“). An important finding is, that a path which has a lower metric value (lower likelihood) than another path which passes through the same state, can never become more likely than that other path in later trellis segments. Consequently, only the path with the highest metric in a state needs to be further considered (survivor path). All other paths which pass through the state can be dropped. This decoding principle enables efficient decoding: In general, with growing length of the code sequence, the number of possible paths through the trellis grows exponentially. However, it is a major advantage of convolutional codes with Viterbi decoding, that the decoding complexity grows only linearly with the code sequence length. This is a result of the fact that only one survivor path is further considered per state and trellis segment (time).

The Viterbi decoding computes the metric up to the end of the block (trellis). The path with the highest metric is the maximum likelihood path. It is traced back and the decoded info sequence is obtained from the labels of the transitions along the ML path.

A disadvantage of the Viterbi algorithm is the decoding delay: We have to wait until all code symbols have been received and the path metrics up to the end of the trellis can be computed. A solution to this problem is, to make decisions already after a predetermined decoding delay. I.e., we can decide on the info symbols in trellis segment k, when we have computed the metric increments up to segment $k+D$. If the decoding delay D is sufficiently long, all survivor paths have merged with the ML path at segment k with high probability. A practical value for the decoding delay is about five times the code memory, i.e. $D \approx 5M$. A fixed decoding delay is also desirable from an implementation point of view, since the memory for storing the survivor paths can be limited to the decoding delay.

Viterbi Algorithm

A further simplification can be achieved, if the trace back does not start at the state with highest metric in segment $k+D$, but always in a predetermined state, i.e. in the all-zero state. In this case, we do not need to determine the maximum among the survivor path metrics at segment $k+D$. This simplification causes virtually no performance degradation, if the decoding delay is chosen sufficiently long, since all survivor paths have merged with the ML path at segment k with high probability.

The decoding complexity of the Viterbi algorithm strongly depends on the memory M of the code, since the number q^M of trellis states grows exponentially with the code memory. Hence, a stronger code causes higher decoding complexity.

The number of metric increments, which have to be computed per trellis segment, depends not only on the number of states but also on the number of trellis transitions (branches) per trellis segment.

A higher code memory causes more states but not necessarily more trellis transitions per state. The number q^K of trellis transitions per state is determined by the number K of info symbols per segment and by the cardinality q of the info symbol alphabeth. However, the number q^N of different metric increments which have to be computed depends on the number N of code symbols per trellis transitions, i.e. on the code rate $R=K/N$. Since usually $N<M+K$, the number of different metric increments is smaller than the number of transitions per trellis segment. For further complexity reduction, symmetries can be exploited. In this context, it is advantageous to use the symbol alphabet $x_{k,n}, y_{k,n} \in \{\pm 1\}$ rather than $x_{k,n}, y_{k,n} \in \{0,1\}$, since in this case we can exploit the metric increment symmetry $y_{k,n}\, x_{k,n}, = -[\, y_{k,n}\, (-x_{k,n})]$

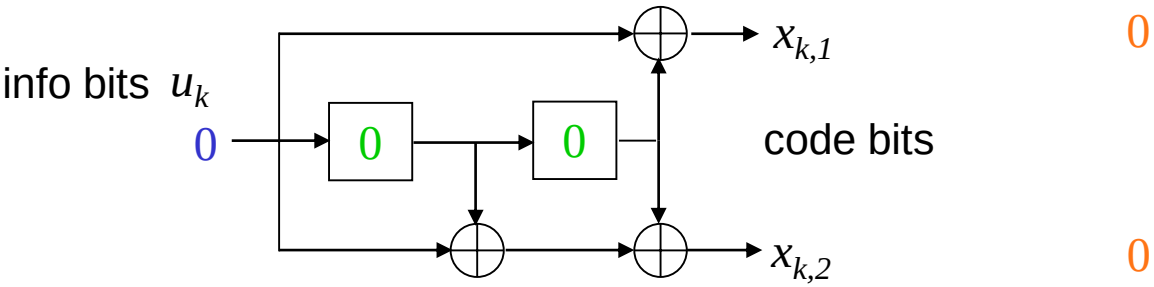
Convolutional Codes

Major advantages of convolutional codes over block codes are implementation aspects. Encoding and decoding can be realized in an efficient way: Encoding is done using a shift register. Maximum-likelihood decoding can be efficiently implemented using the Viterbi algorithm. Moreover, soft information can be processed in a much easier way by decoders of convolutional codes than with block codes. Also, soft-output information can be produced in a relatively simple way, which in general is not easy with block codes. This makes convolutional codes attractive components in concatenated coding or detection schemes. A powerful concatenated code is obtained by the concatenation of relatively simple constituent codes. The decoder of the second constituent codes uses the output of the first constituent code as its input data. Hence, the first decoder should provide soft output information in order to allow the second decoder to benefit from soft decision decoding.

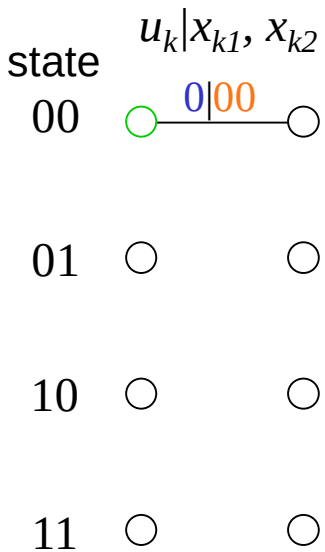
Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



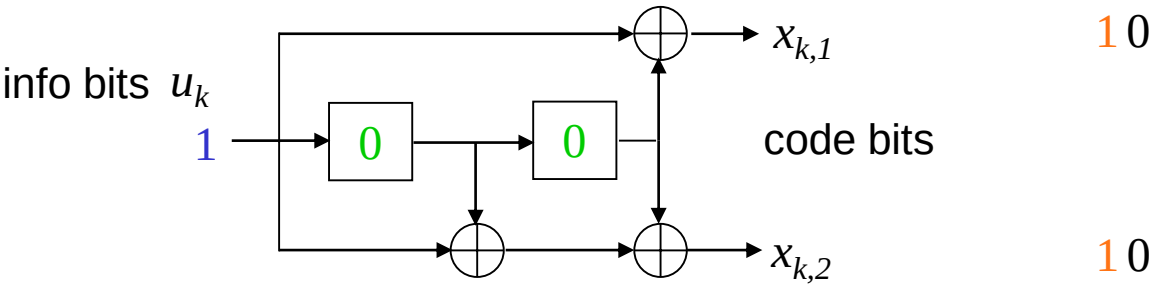
Trellis:



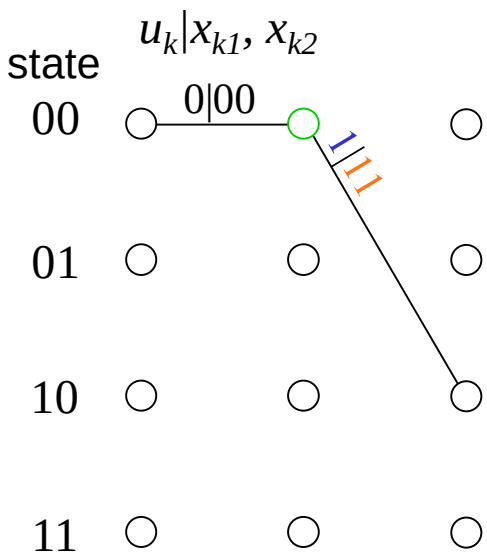
Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



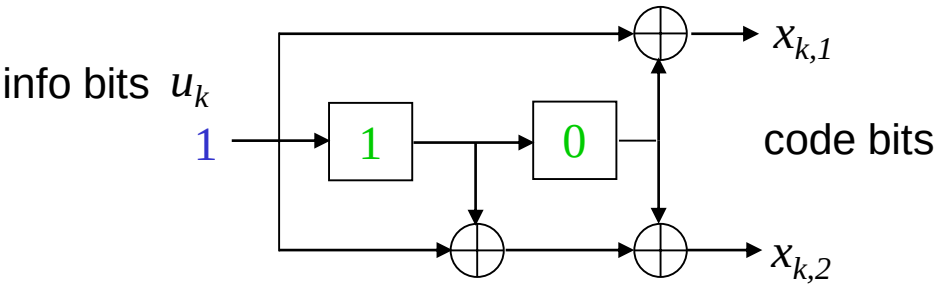
Trellis:



Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

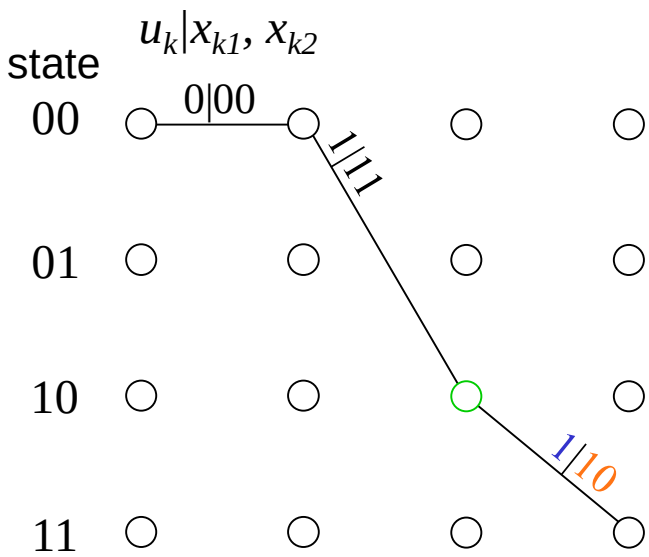
Code rate: $R = \frac{K}{N} = \frac{1}{2}$



1 1 0

0 1 0

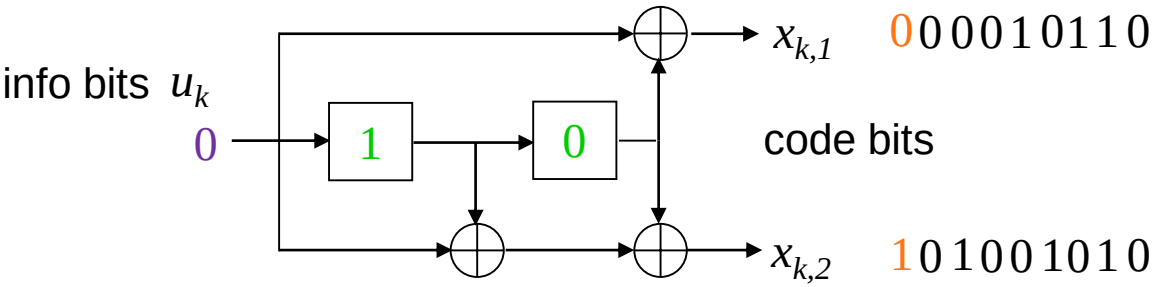
Trellis:



Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

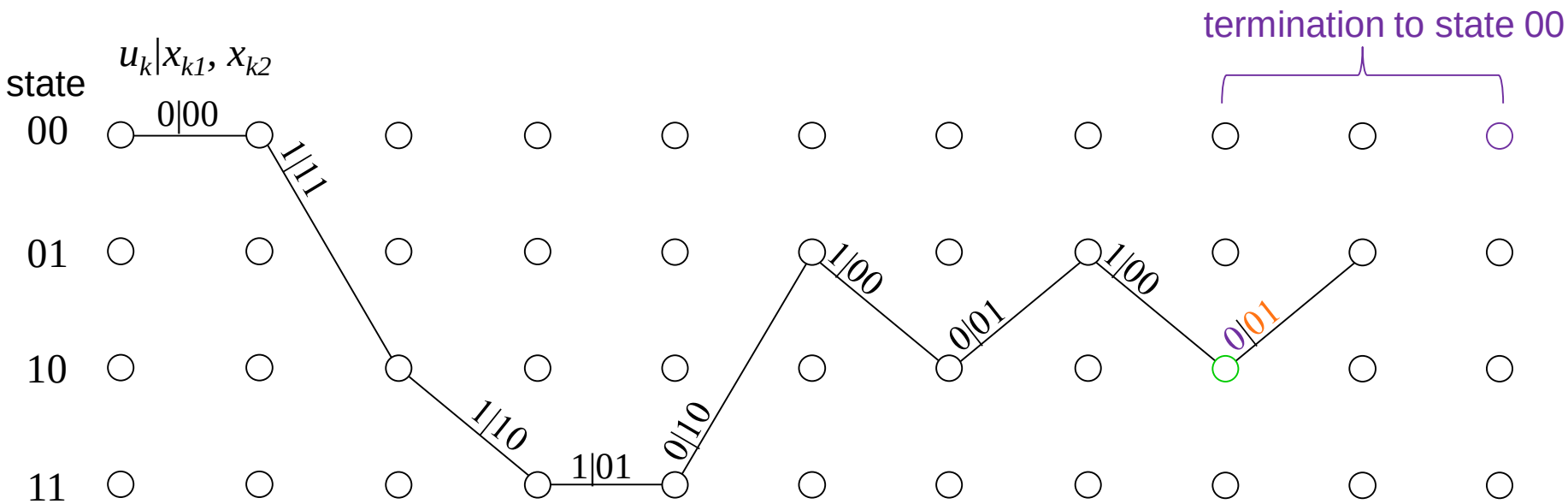
Code rate: $R = \frac{K}{N} = \frac{1}{2}$



0 0 0 0 1 0 1 1 0

1 0 1 0 0 1 0 1 0

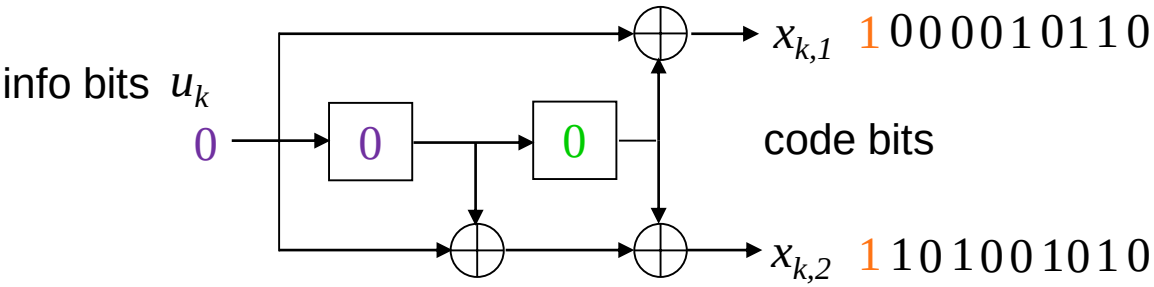
Trellis:



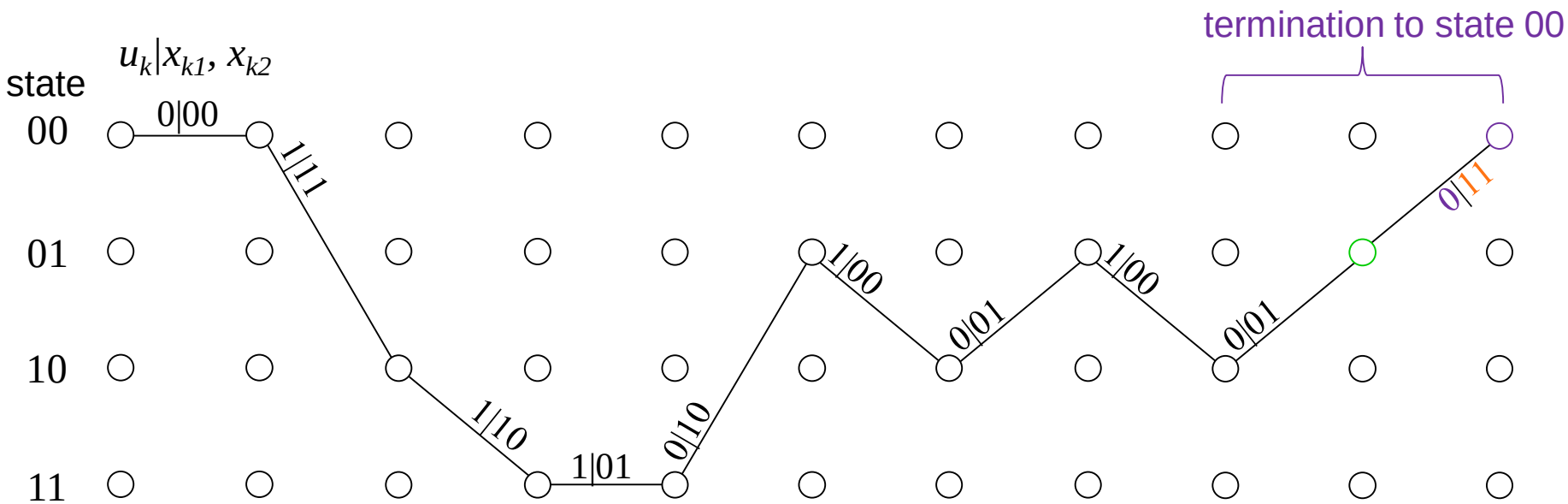
Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



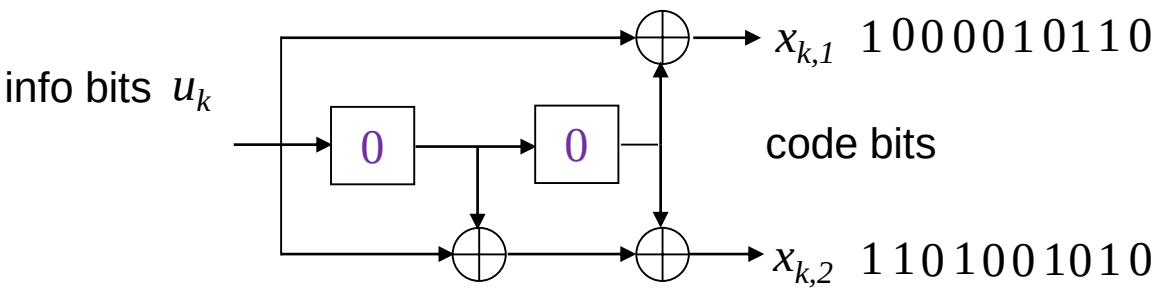
Trellis:



Convolutional Codes

Example:
Memory: $M=2$
Constraint length: $M+1=3$

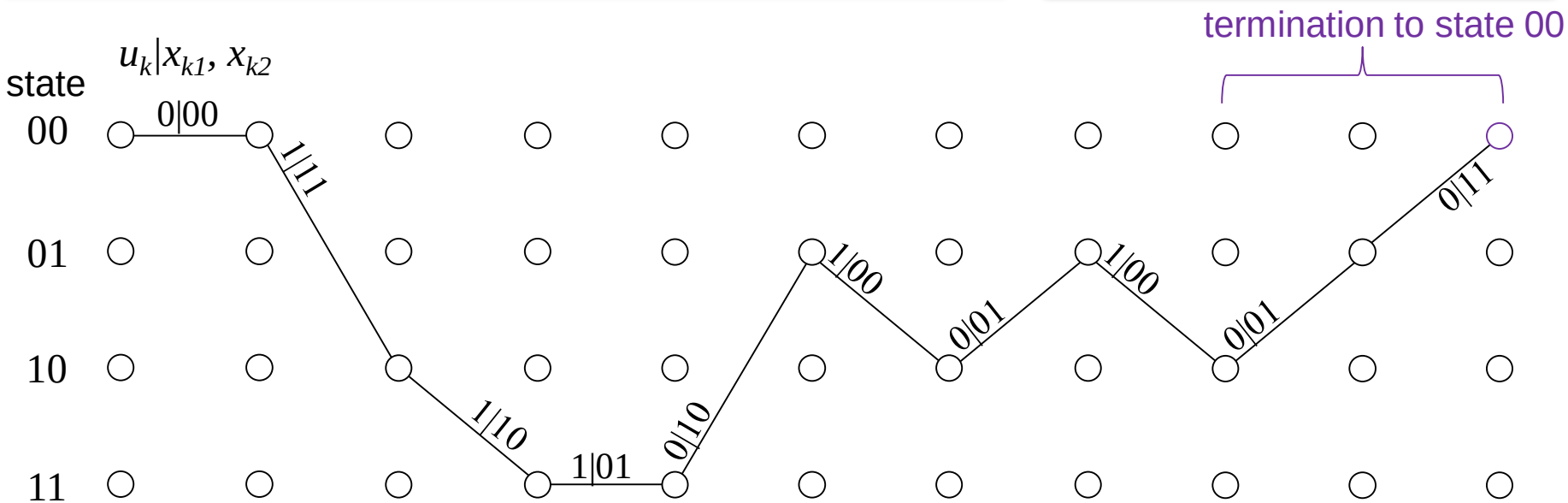
Code rate: $R = \frac{K}{N} = \frac{1}{2}$



info sequence:	0	1	1	1	0	1	0	1	0	0
transmitted code sequence:	00	11	10	01	10	00	01	00	01	11

$$R = \frac{8}{2 \cdot 8 + 2M} = \frac{8}{20} = \frac{2}{5} < \frac{1}{2}$$

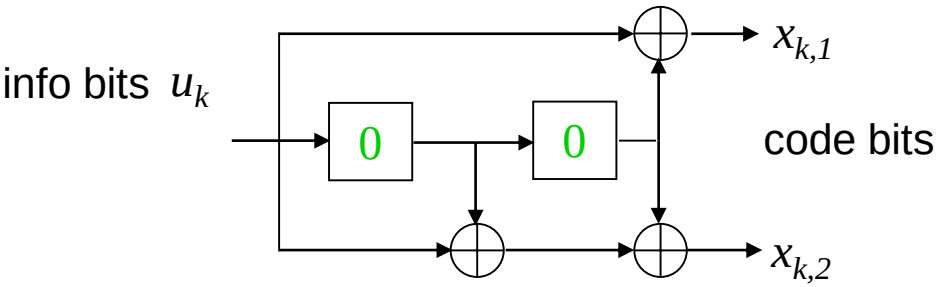
→ Trellis termination causes a rate loss.

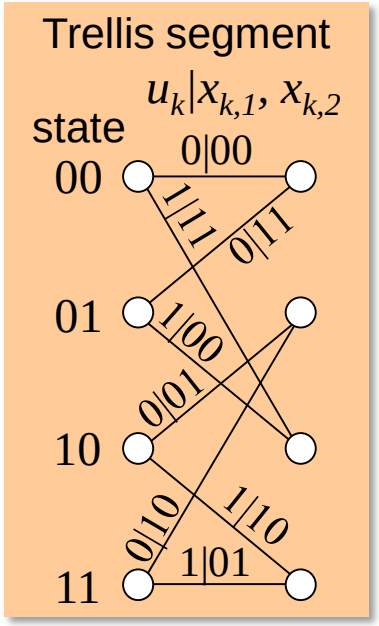


Convolutional Codes

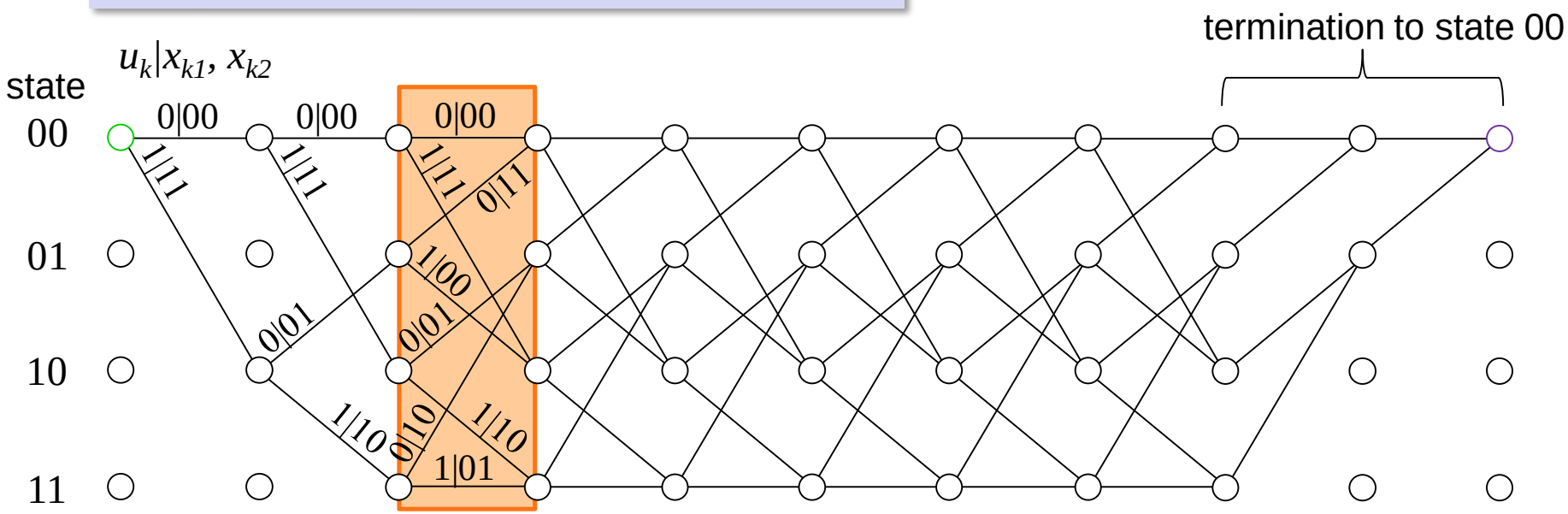
Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$

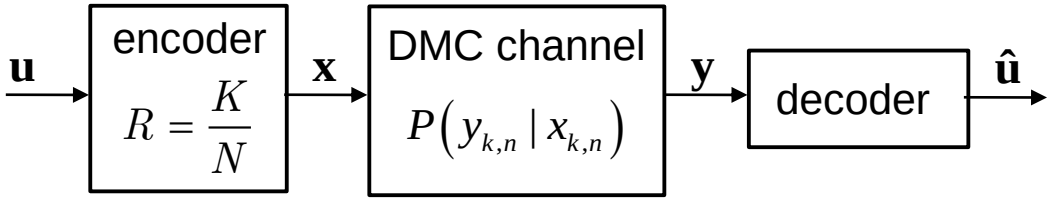




The decoder needs to find the most likely path through the trellis given the received sequence.



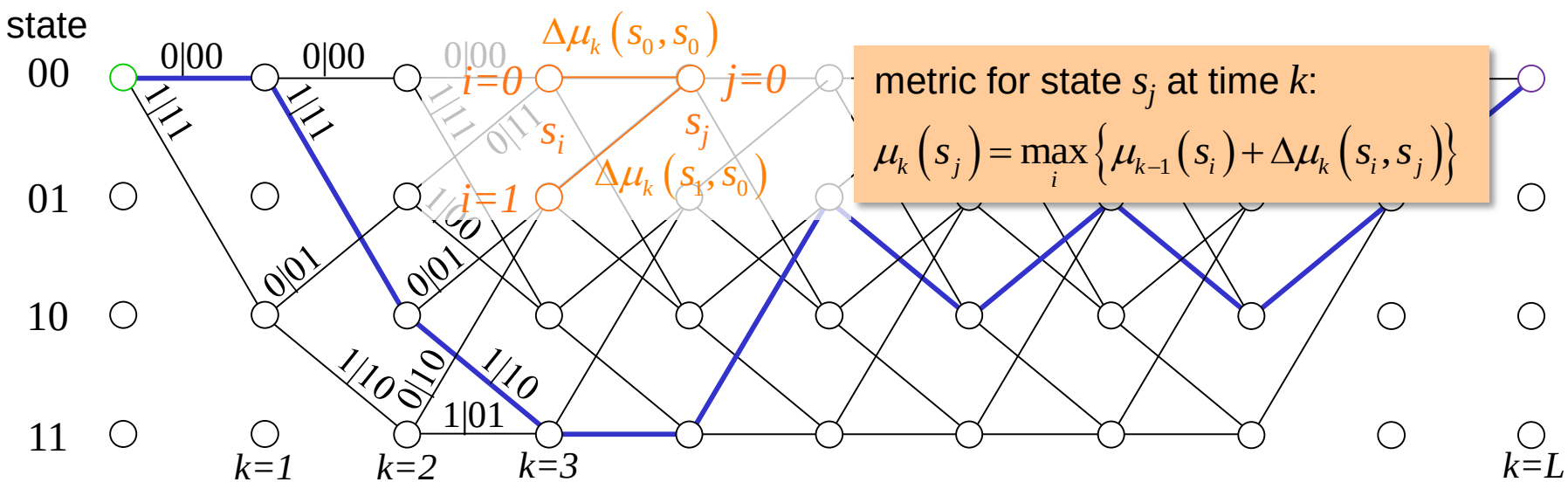
Viterbi Decoding



Maximum likelihood sequence estimation:

$$\max_{\mathbf{x}} P(\mathbf{y} \mid \mathbf{x}) = \max_{\mathbf{x}} \prod_{k=1}^{L} \prod_{n=1}^{N} P(y_{k,n} \mid x_{k,n}) = \max_{\mathbf{x}} \sum_{k=1}^{L} \sum_{n=1}^{N} \log P(y_{k,n} \mid x_{k,n})$$

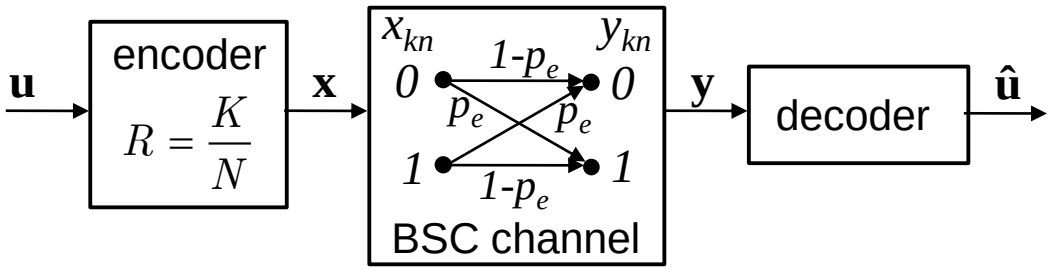
$$= \max_{\mathbf{x}} \left\{ \sum_{l=1}^{k-1} \sum_{n=1}^{N} \log P(y_{l,n} \mid x_{l,n}) + \underbrace{\sum_{n=1}^{N} \log P(y_{k,n} \mid x_{k,n})}_{\text{metric increment } \Delta\mu_k(s_i, s_j)} + \sum_{t=k+1}^{L} \sum_{n=1}^{N} \log P(y_{t,n} \mid x_{t,n}) \right\}$$



metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\}$$

Viterbi Decoding: Hard Decision Decoding



$x_{k,n}, y_{k,n} \in \{0,1\}$

$$P\left(y_{k,n} \mid x_{k,n}\right) = \begin{cases} (1-p_e) & \text{for } y_{k,n} = x_{k,n} \\ p_e & \text{for } y_{k,n} \neq x_{k,n} \end{cases}$$

Maximum likelihood sequence estimation:

$$\max_{\mathbf{x}} P(\mathbf{y} \mid \mathbf{x}) = \max_{\mathbf{x}} \prod_{k=1}^{L} \prod_{n=1}^{N} P\left(y_{k,n} \mid x_{k,n}\right) = \max_{\mathbf{x}} \sum_{k=1}^{L} \sum_{n=1}^{N} \log P\left(y_{k,n} \mid x_{k,n}\right)$$

$$= \max_{\mathbf{x}} \left\{ \left[\sum_{l=1}^{k-1} \sum_{n=1}^{N} \alpha + \beta \log P\left(y_{l,n} \mid x_{l,n}\right) \right] + \underbrace{\left[\sum_{n=1}^{N} \alpha + \beta \log P\left(y_{k,n} \mid x_{k,n}\right) \right]} + \left[\sum_{t=k+1}^{L} \sum_{n=1}^{N} \alpha + \beta \log P\left(y_{t,n} \mid x_{t,n}\right) \right] \right\}$$

metric increment $\Delta\mu_{\mathrm{k}}\left(s_i, s_j\right) = \alpha + \beta \log P\left(y_{k,n} \mid x_{k,n}\right), \qquad \beta > 0$

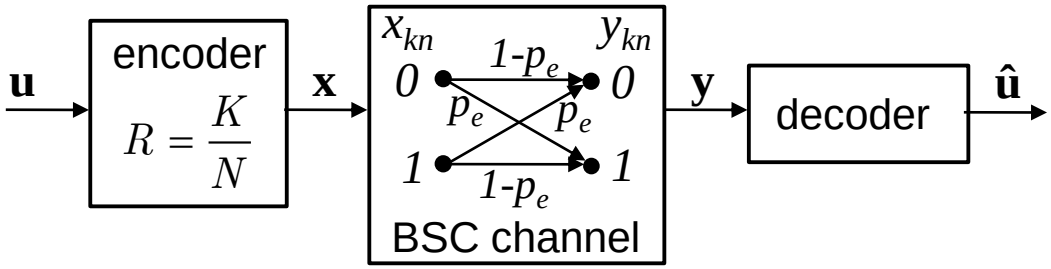
$$\Rightarrow \begin{cases} \alpha + \beta \log(1-p_e) = -\beta \log(1-p_e) + \beta \log(1-p_e) \\ \alpha + \beta \log(p_e) = -\beta \log(1-p_e) + \beta \log(p_e) \end{cases}$$

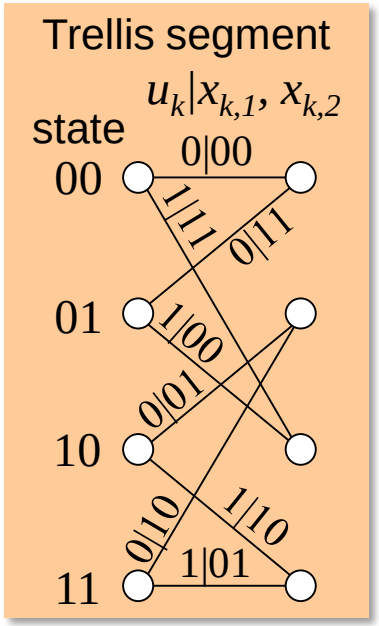
$$= \begin{cases} 0 & = -\left(y_{k,n} \oplus x_{k,n}\right) \text{ for } y_{k,n} = x_{k,n} \\ \beta \log \dfrac{p_e}{1-p_e} = -1 & = -\left(y_{k,n} \oplus x_{k,n}\right) \text{ for } y_{k,n} \neq x_{k,n} \end{cases}$$

$$\boxed{\Delta\mu_{\mathrm{k}}\left(s_i, s_j\right) = -\sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n}\right)}$$

$$\beta = -\frac{1}{\log \dfrac{p_e}{1-p_e}} > 0 \text{ for } p_e < \frac{1}{2}$$

Viterbi Decoding: Hard Decision Decoding



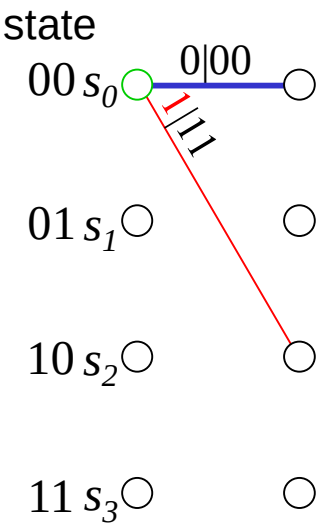


Maximum likelihood sequence estimation:

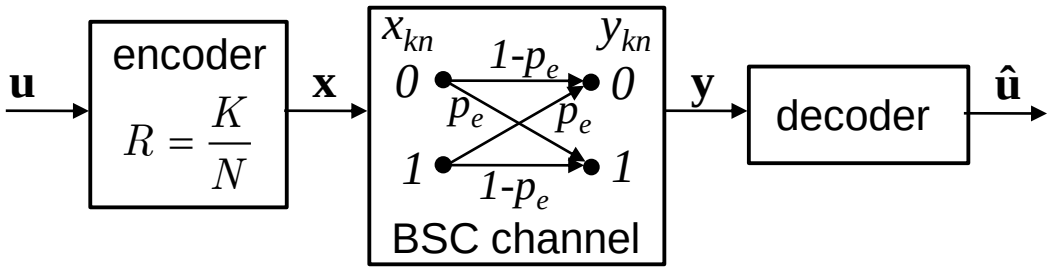
Metric for state s_j at time k:

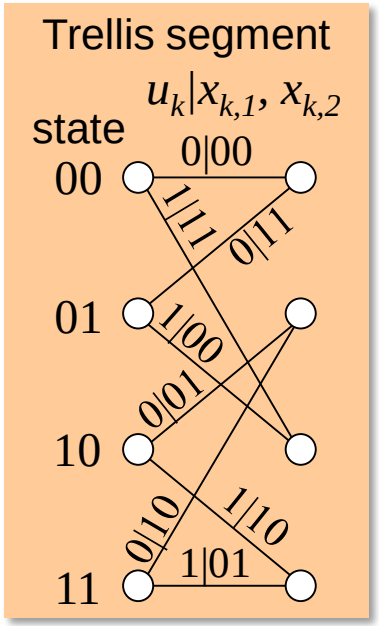
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



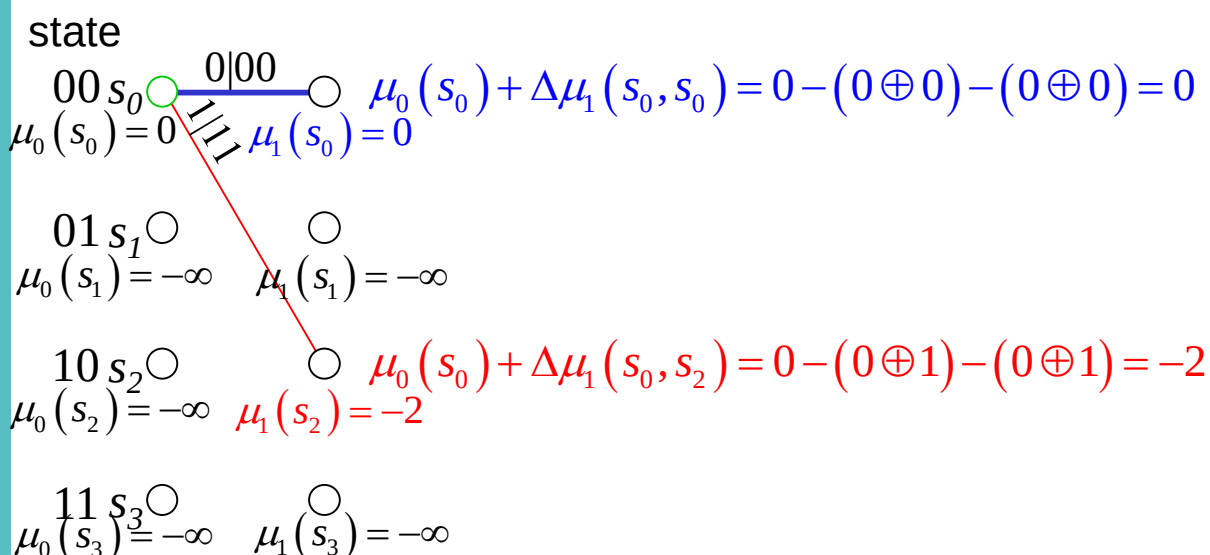


Maximum likelihood sequence estimation:

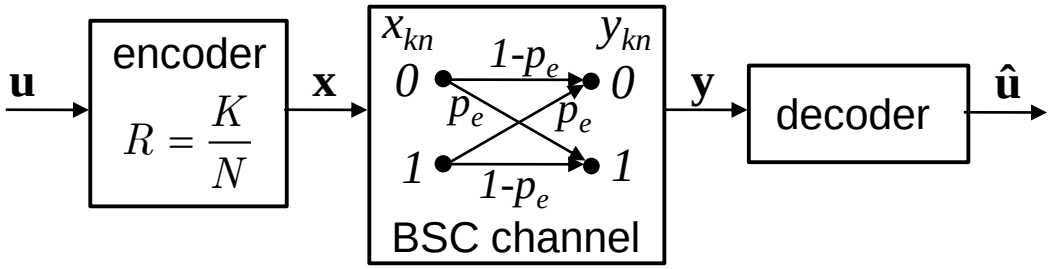
Metric for state s_j at time k:

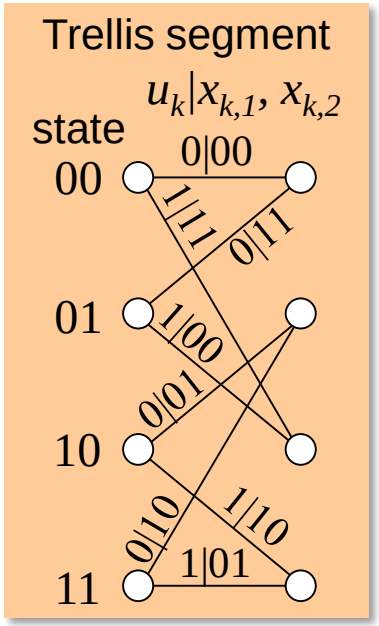
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



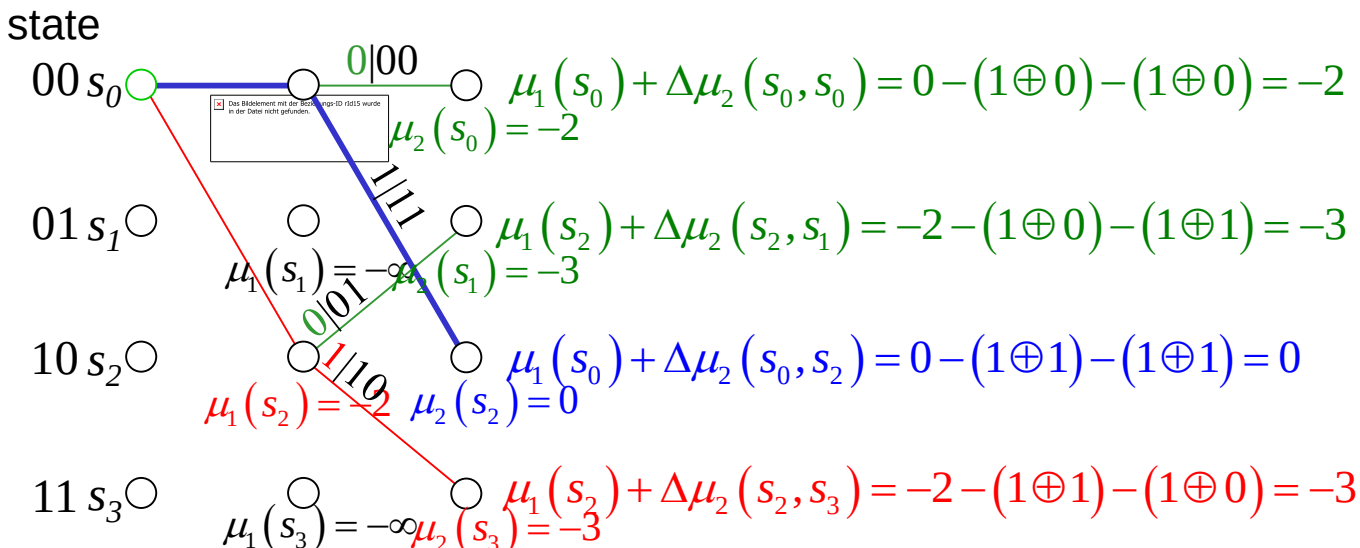


Maximum likelihood sequence estimation:

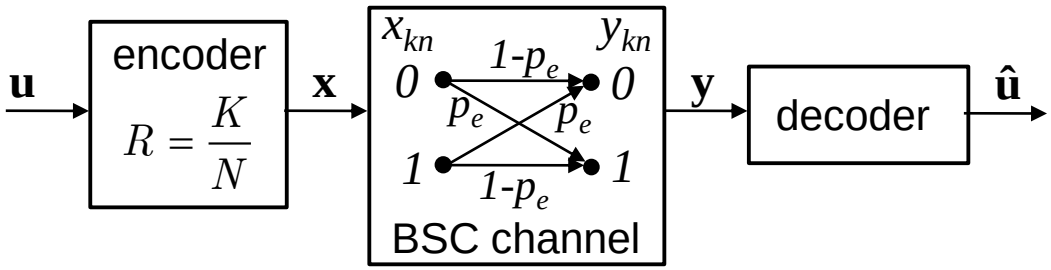
Metric for state s_j at time k:

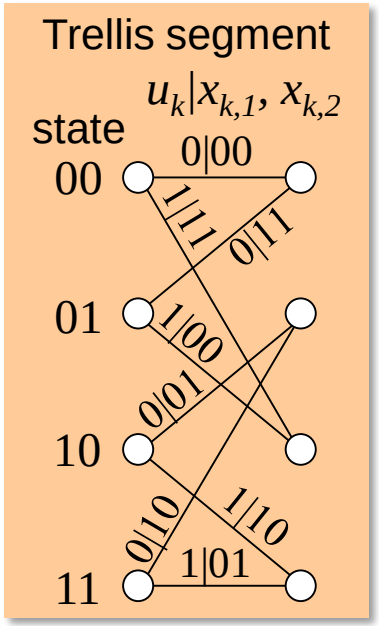
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



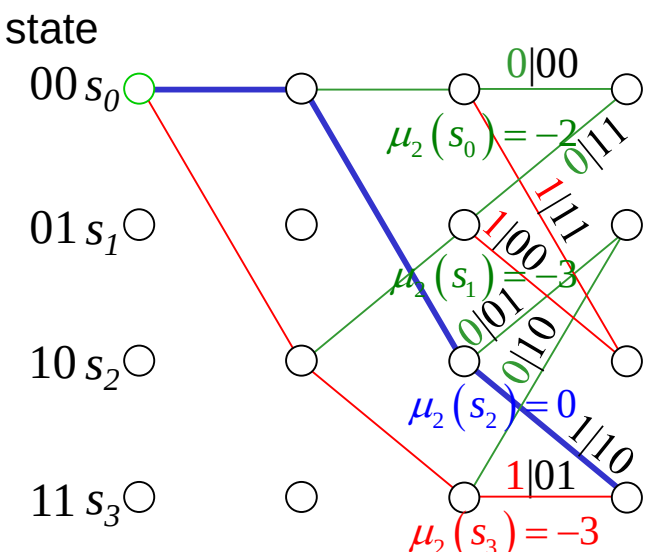


Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



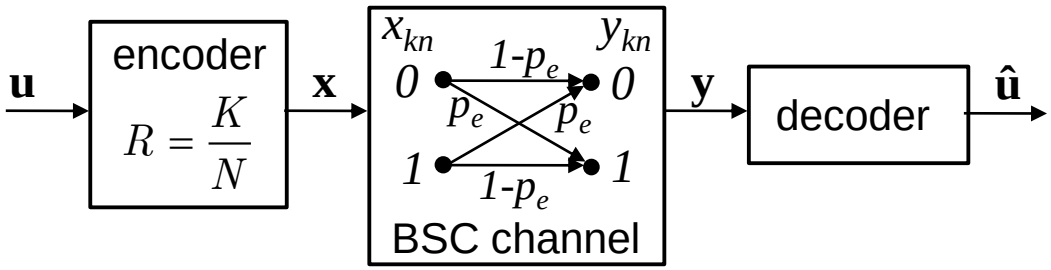
$$\mu_2(s_0) + \Delta\mu_3(s_0, s_0) = -2 - (1 \oplus 0) - (1 \oplus 0) = -4$$
$$\mu_2(s_1) + \Delta\mu_3(s_1, s_0) = -3 - (1 \oplus 1) - (1 \oplus 1) = -3$$

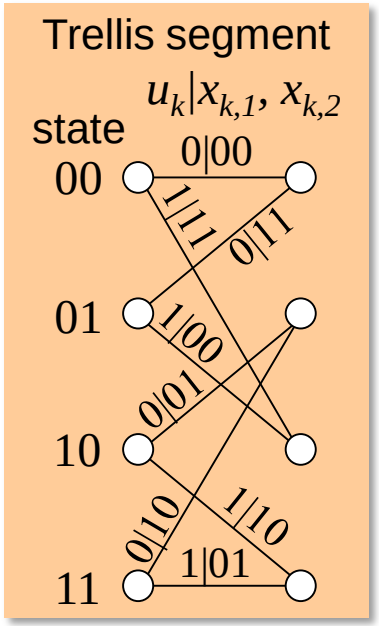
$$\mu_2(s_2) + \Delta\mu_3(s_2, s_1) = 0 - (1 \oplus 0) - (1 \oplus 1) = -1$$
$$\mu_2(s_3) + \Delta\mu_3(s_3, s_1) = -3 - (1 \oplus 1) - (1 \oplus 0) = -4$$

$$\mu_2(s_0) + \Delta\mu_3(s_0, s_2) = -2 - (1 \oplus 1) - (1 \oplus 1) = -2$$
$$\mu_2(s_1) + \Delta\mu_3(s_1, s_2) = -3 - (1 \oplus 0) - (1 \oplus 0) = -5$$

$$\mu_2(s_2) + \Delta\mu_3(s_2, s_3) = 0 - (1 \oplus 1) - (1 \oplus 0) = -1$$
$$\mu_2(s_3) + \Delta\mu_3(s_3, s_3) = -3 - (1 \oplus 0) - (1 \oplus 1) = -4$$

Viterbi Decoding: Hard Decision Decoding



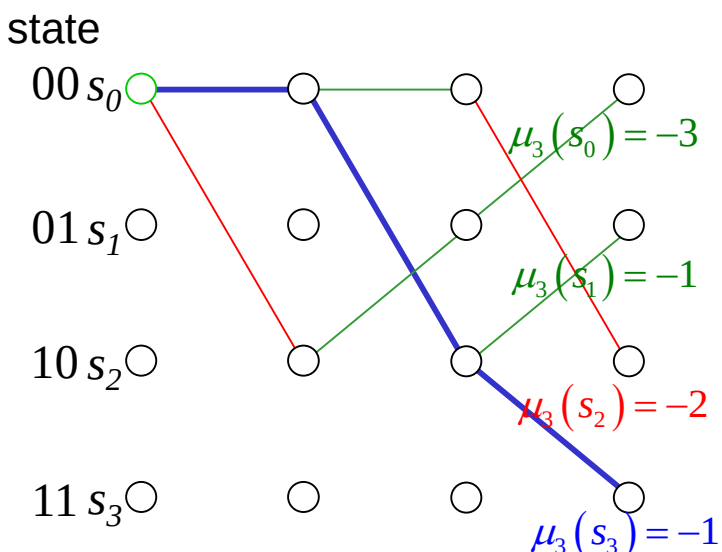


Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



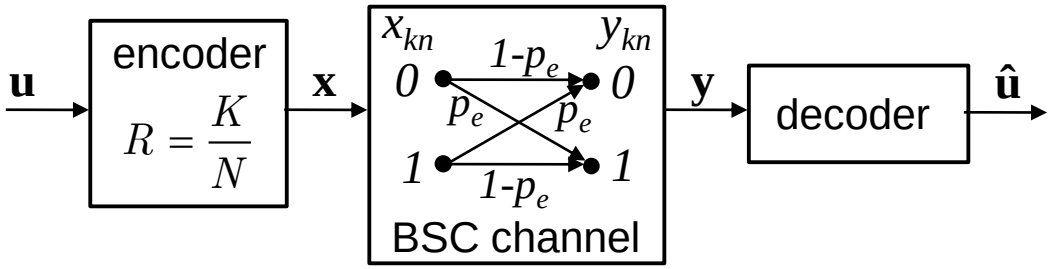
$$\mu_2(s_0) + \Delta\mu_3(s_0, s_0) = -2 - (1 \oplus 0) - (1 \oplus 0) = -4$$
$$\mu_2(s_1) + \Delta\mu_3(s_1, s_0) = -3 - (1 \oplus 1) - (1 \oplus 1) = -3$$

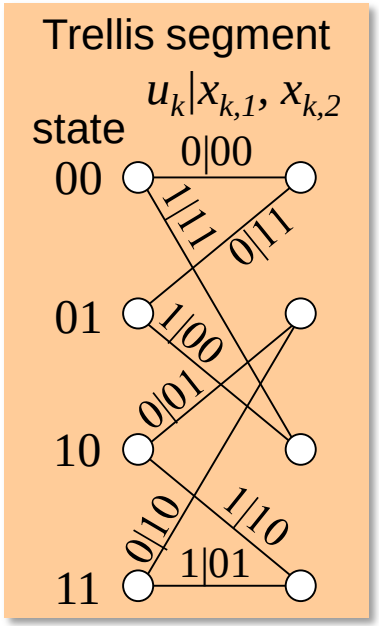
$$\mu_2(s_2) + \Delta\mu_3(s_2, s_1) = 0 - (1 \oplus 0) - (1 \oplus 1) = -1$$
$$\mu_2(s_3) + \Delta\mu_3(s_3, s_1) = -3 - (1 \oplus 1) - (1 \oplus 0) = -4$$

$$\mu_2(s_0) + \Delta\mu_3(s_0, s_2) = -2 - (1 \oplus 1) - (1 \oplus 1) = -2$$
$$\mu_2(s_1) + \Delta\mu_3(s_1, s_2) = -3 - (1 \oplus 0) - (1 \oplus 0) = -5$$

$$\mu_2(s_2) + \Delta\mu_3(s_2, s_3) = 0 - (1 \oplus 1) - (1 \oplus 0) = -1$$
$$\mu_2(s_3) + \Delta\mu_3(s_3, s_3) = -3 - (1 \oplus 0) - (1 \oplus 1) = -4$$

Viterbi Decoding: Hard Decision Decoding



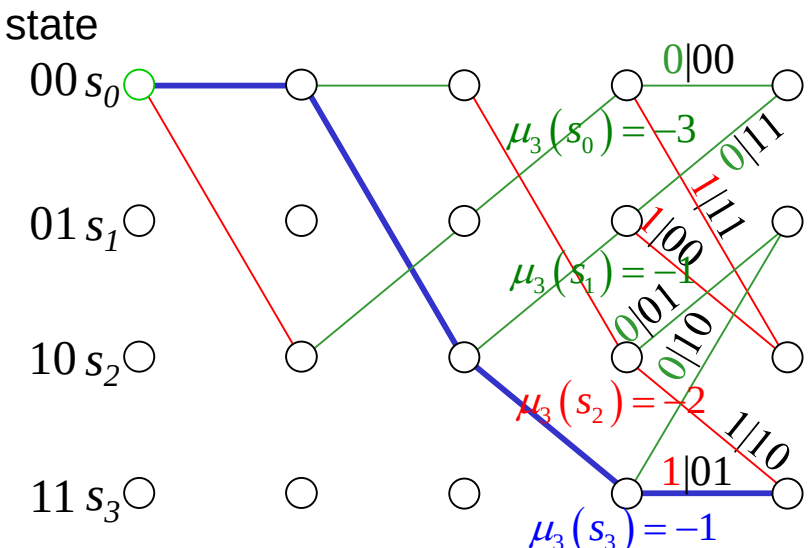


Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



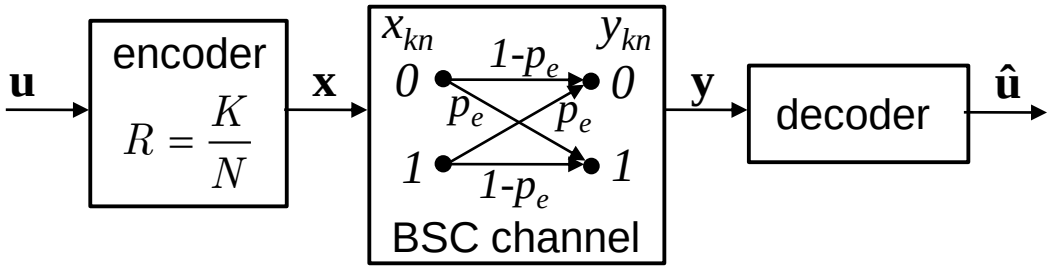
$$\mu_3(s_0) + \Delta\mu_4(s_0, s_0) = -3 - (0 \oplus 0) - (0 \oplus 0) = -3$$
$$\mu_3(s_1) + \Delta\mu_4(s_1, s_0) = -1 - (0 \oplus 1) - (0 \oplus 1) = -3$$

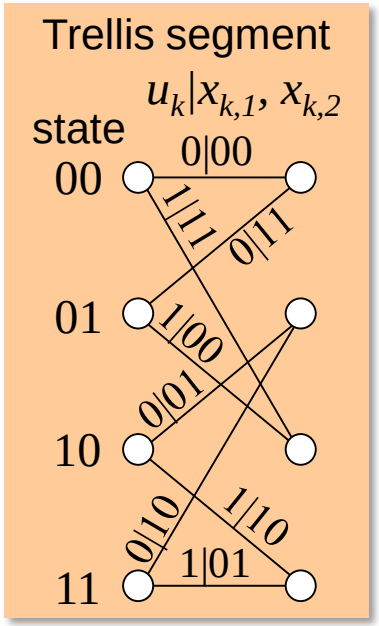
$$\mu_3(s_2) + \Delta\mu_4(s_2, s_1) = -2 - (0 \oplus 0) - (0 \oplus 1) = -3$$
$$\mu_3(s_3) + \Delta\mu_4(s_3, s_1) = -1 - (0 \oplus 1) - (0 \oplus 0) = -2$$

$$\mu_3(s_0) + \Delta\mu_4(s_0, s_2) = -3 - (0 \oplus 1) - (0 \oplus 1) = -5$$
$$\mu_3(s_1) + \Delta\mu_4(s_1, s_2) = -1 - (0 \oplus 0) - (0 \oplus 0) = -1$$

$$\mu_3(s_2) + \Delta\mu_4(s_2, s_3) = 2 - (0 \oplus 1) - (0 \oplus 0) = -3$$
$$\mu_3(s_3) + \Delta\mu_4(s_3, s_3) = -1 - (0 \oplus 0) - (0 \oplus 1) = -2$$

Viterbi Decoding: Hard Decision Decoding



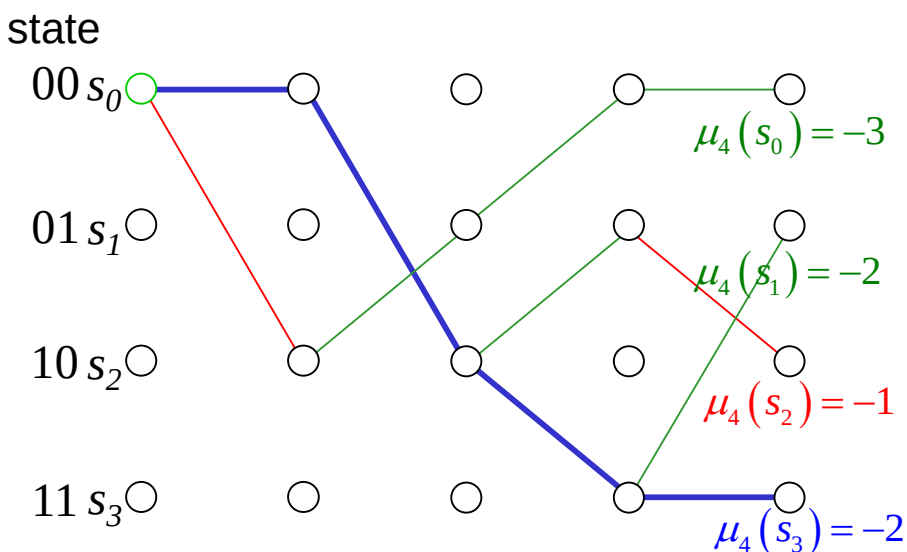


Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



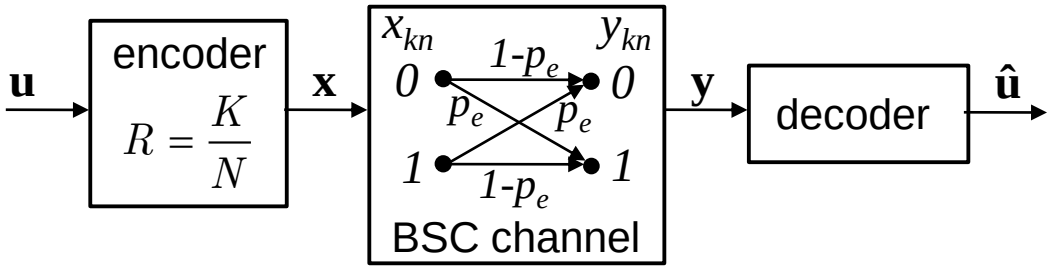
$$\mu_3(s_0) + \Delta\mu_4(s_0, s_0) = -3 - (0 \oplus 0) - (0 \oplus 0) = -3$$
$$\mu_3(s_1) + \Delta\mu_4(s_1, s_0) = -1 - (0 \oplus 1) - (0 \oplus 1) = -3$$

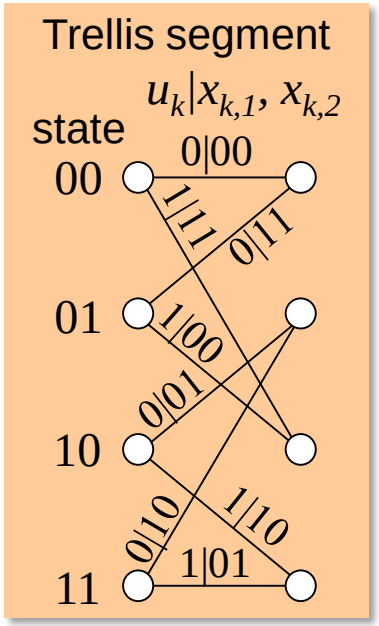
$$\mu_3(s_2) + \Delta\mu_4(s_2, s_1) = -2 - (0 \oplus 0) - (0 \oplus 1) = -3$$
$$\mu_3(s_3) + \Delta\mu_4(s_3, s_1) = -1 - (0 \oplus 1) - (0 \oplus 0) = -2$$

$$\mu_3(s_0) + \Delta\mu_4(s_0, s_2) = -3 - (0 \oplus 1) - (0 \oplus 1) = -5$$
$$\mu_3(s_1) + \Delta\mu_4(s_1, s_2) = -1 - (0 \oplus 0) - (0 \oplus 0) = -1$$

$$\mu_3(s_2) + \Delta\mu_4(s_2, s_3) = 2 - (0 \oplus 1) - (0 \oplus 0) = -3$$
$$\mu_3(s_3) + \Delta\mu_4(s_3, s_3) = -1 - (0 \oplus 0) - (0 \oplus 1) = -2$$

Viterbi Decoding: Hard Decision Decoding



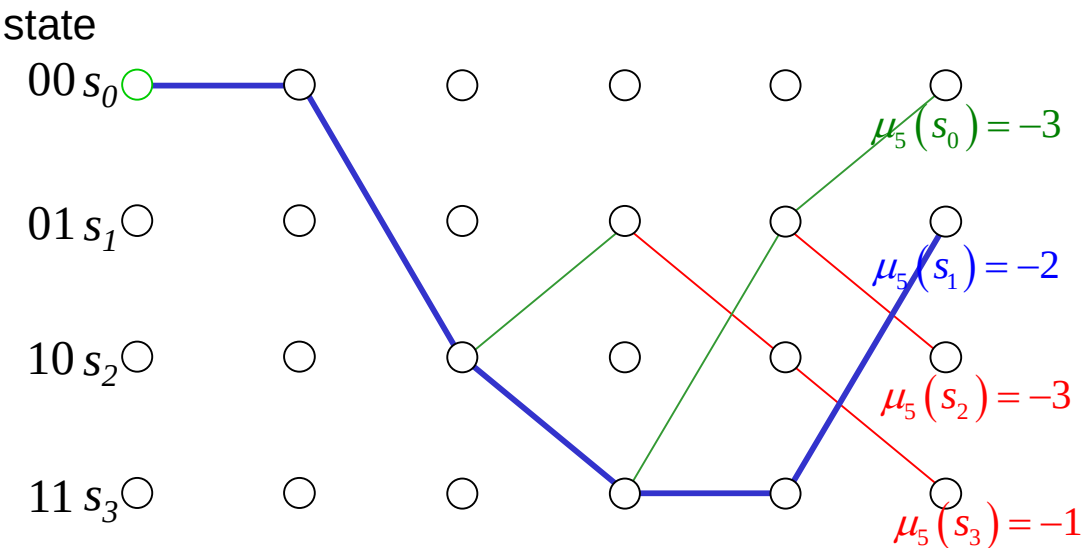


Maximum likelihood sequence estimation:

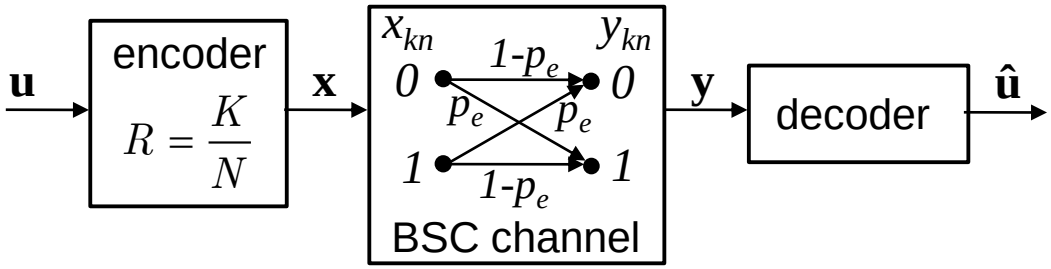
Metric for state s_j at time k:

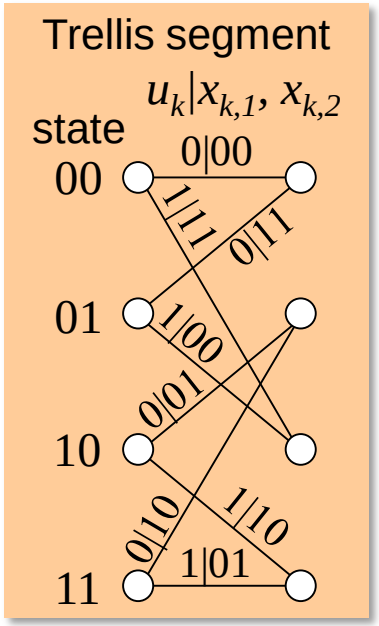
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



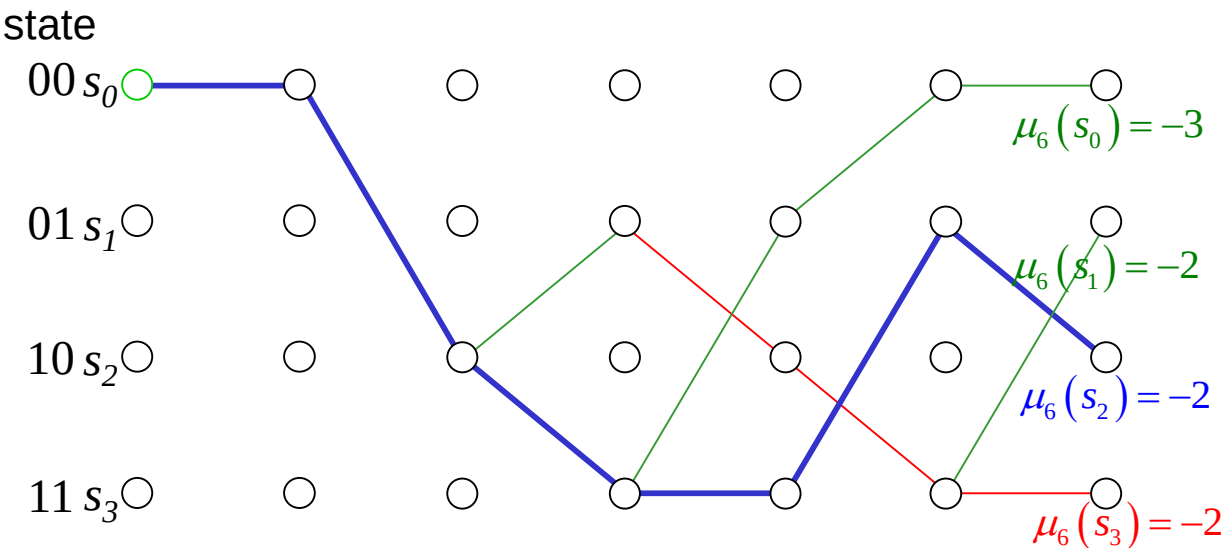


Maximum likelihood sequence estimation:

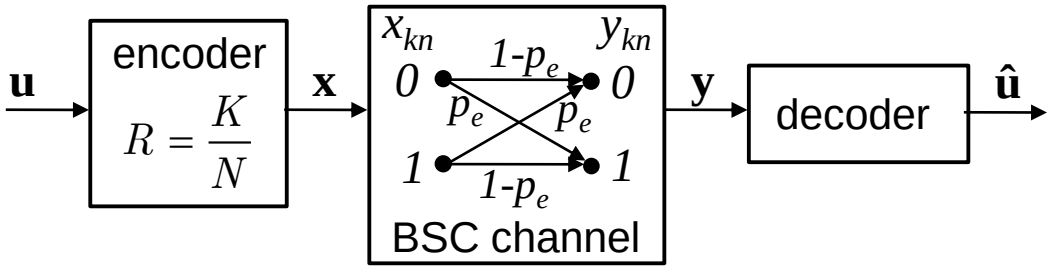
Metric for state s_j at time k:

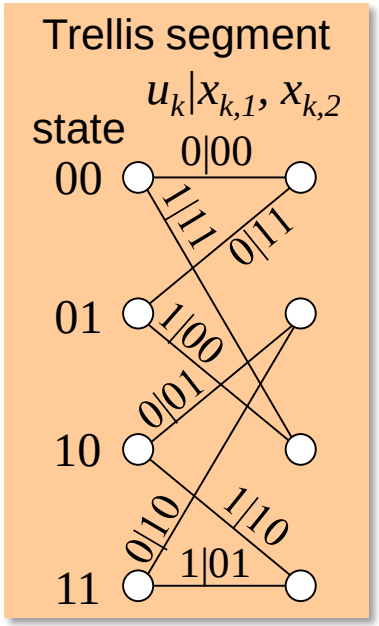
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



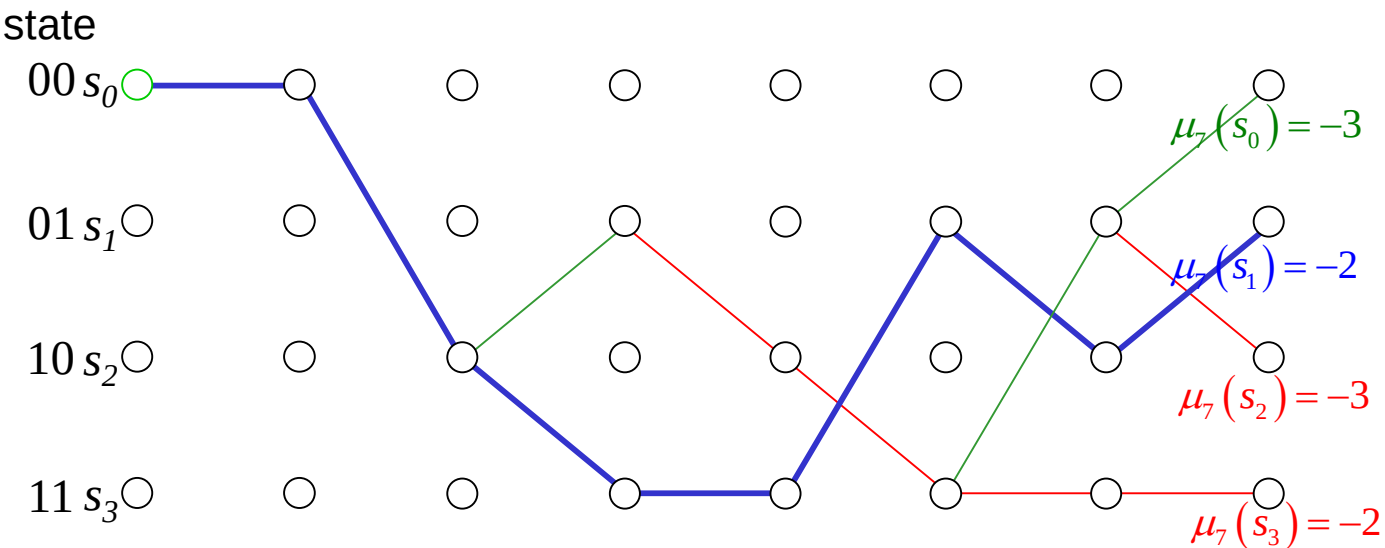


Maximum likelihood sequence estimation:

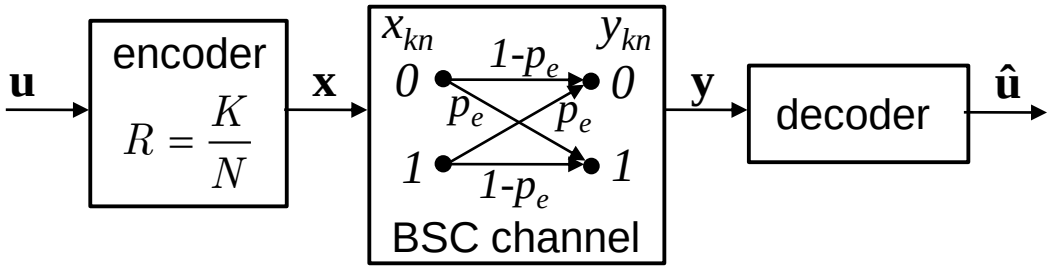
Metric for state s_j at time k:

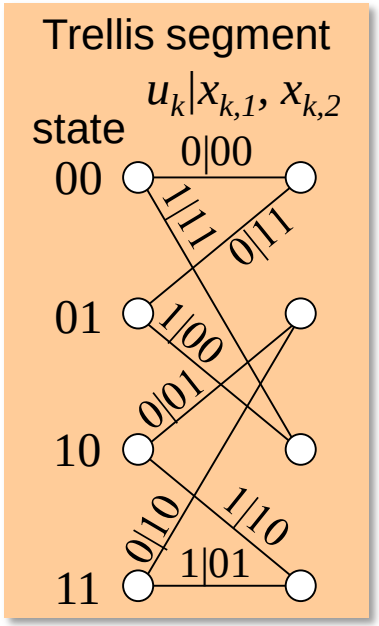
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



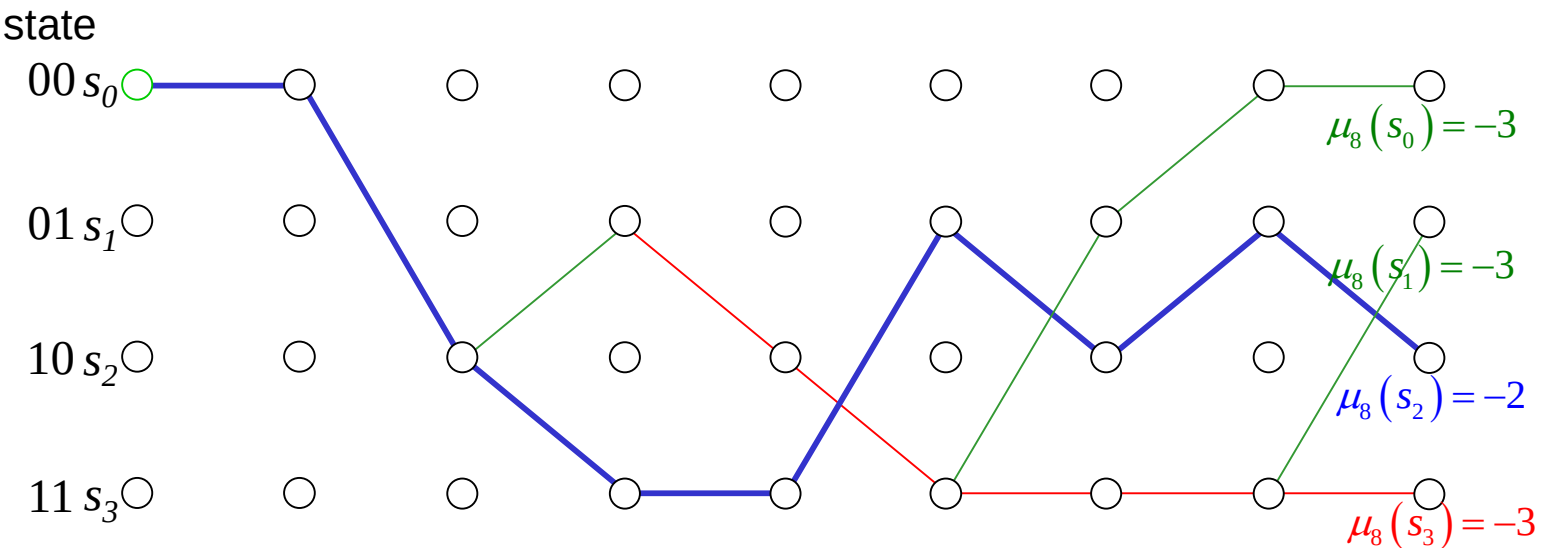


Maximum likelihood sequence estimation:

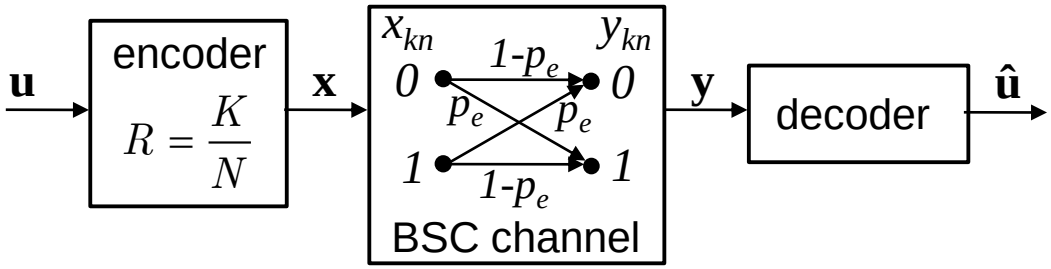
Metric for state s_j at time k:

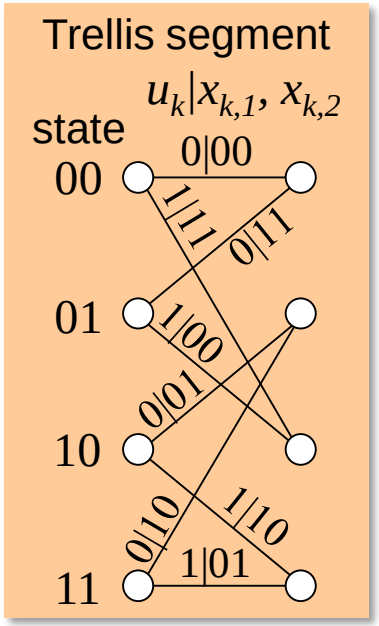
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



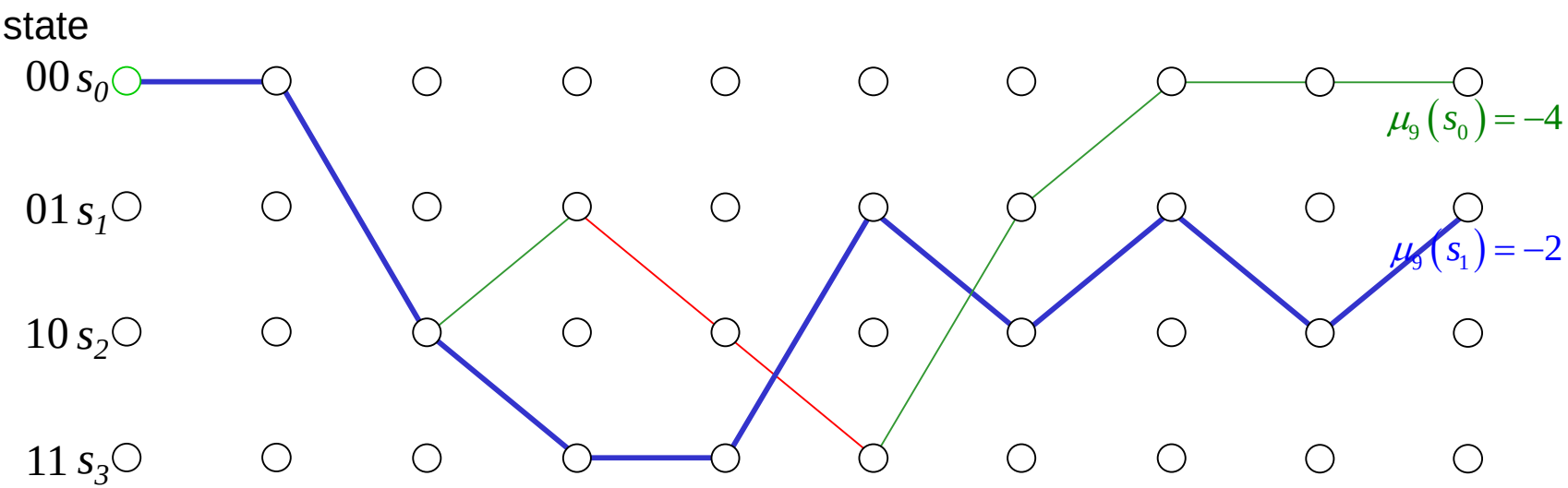


Maximum likelihood sequence estimation:

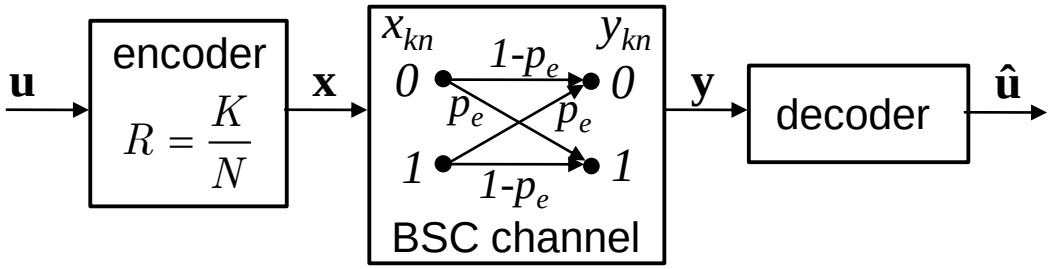
Metric for state s_j at time k:

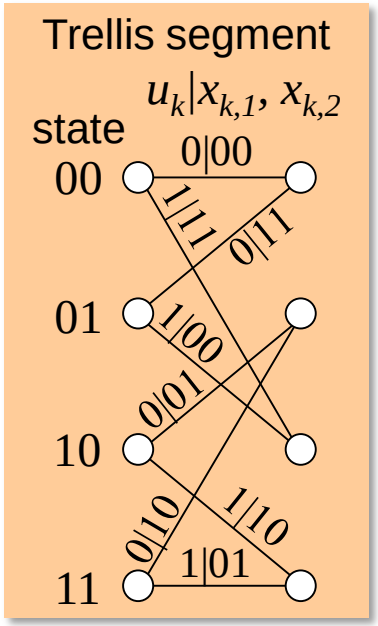
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



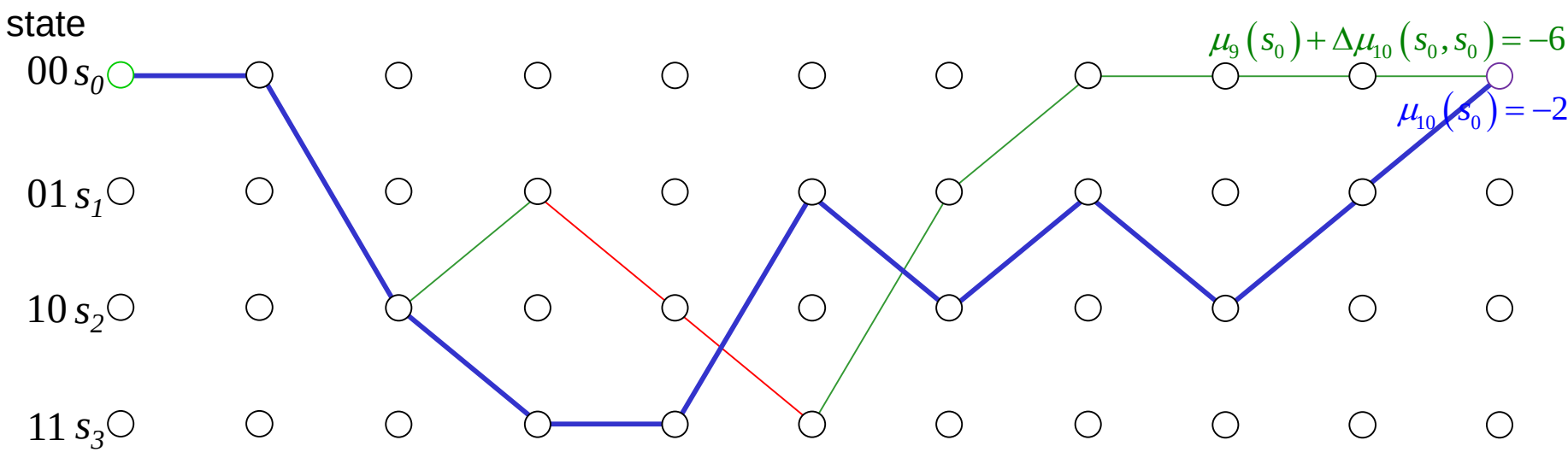


Maximum likelihood sequence estimation:

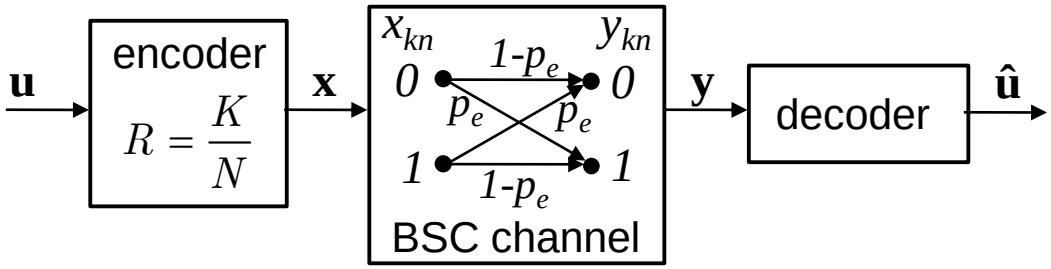
Metric for state s_j at time k:

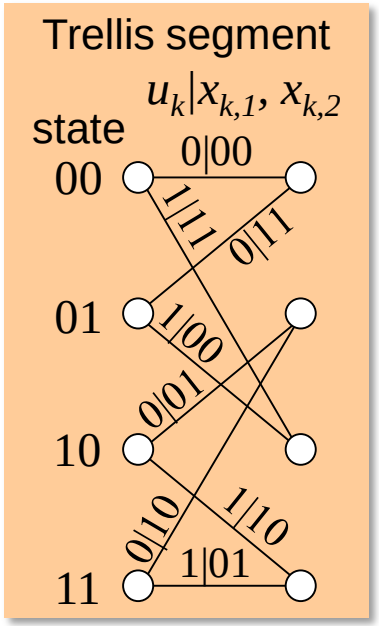
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11



Viterbi Decoding: Hard Decision Decoding



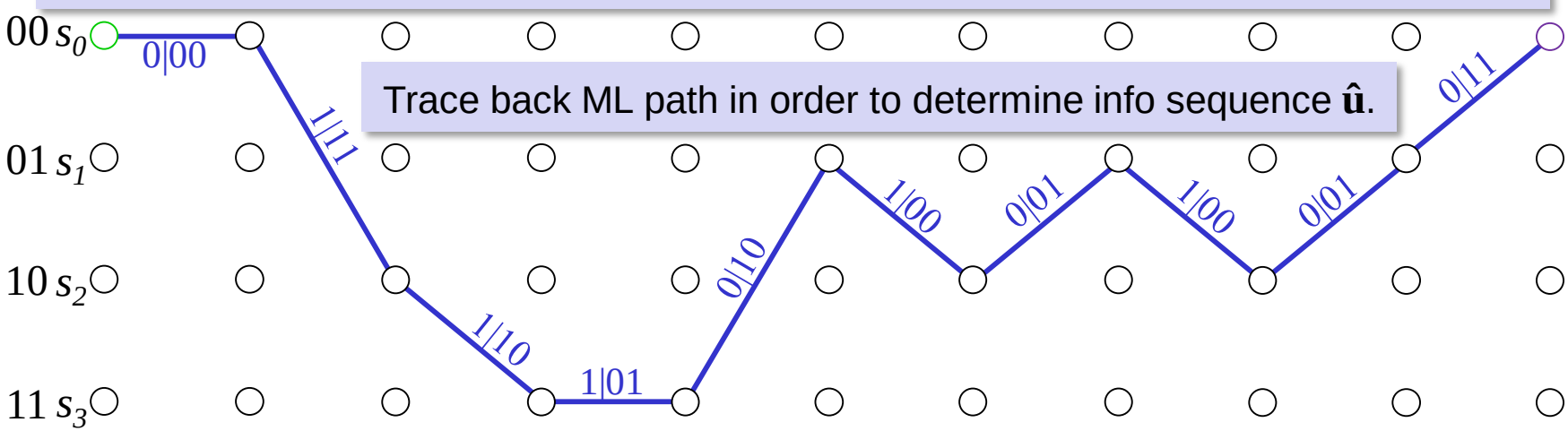


Maximum likelihood sequence estimation:

Metric for state s_j at time k:

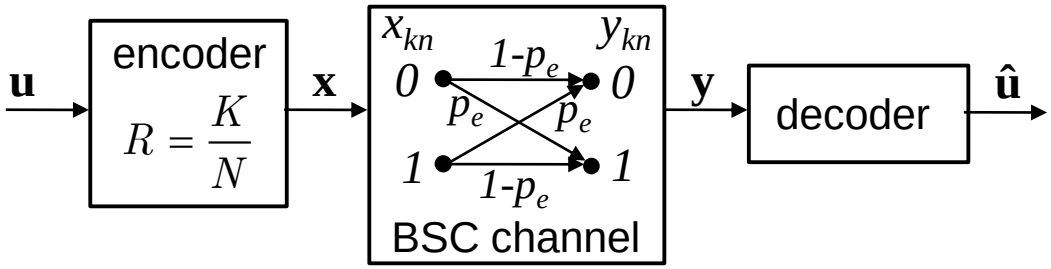
$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\} = \max_i \left\{ \mu_{k-1}(s_i) - \sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n} \right) \right\}$$

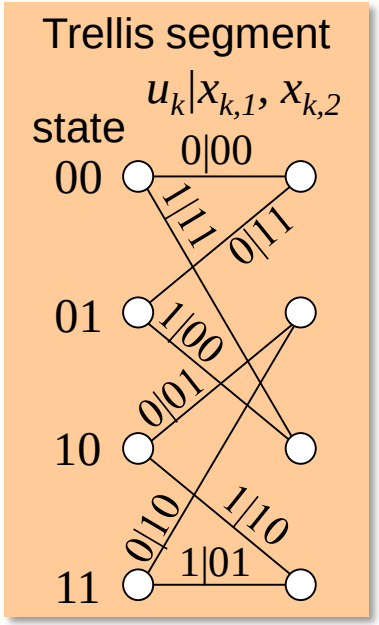
x:	00	11	10	01	10	00	01	00	01	11
y:	00	11	11	00	10	00	01	00	01	11
û:	0	1	1	1	0	1	0	1	0	0



Trace back ML path in order to determine info sequence **û**.

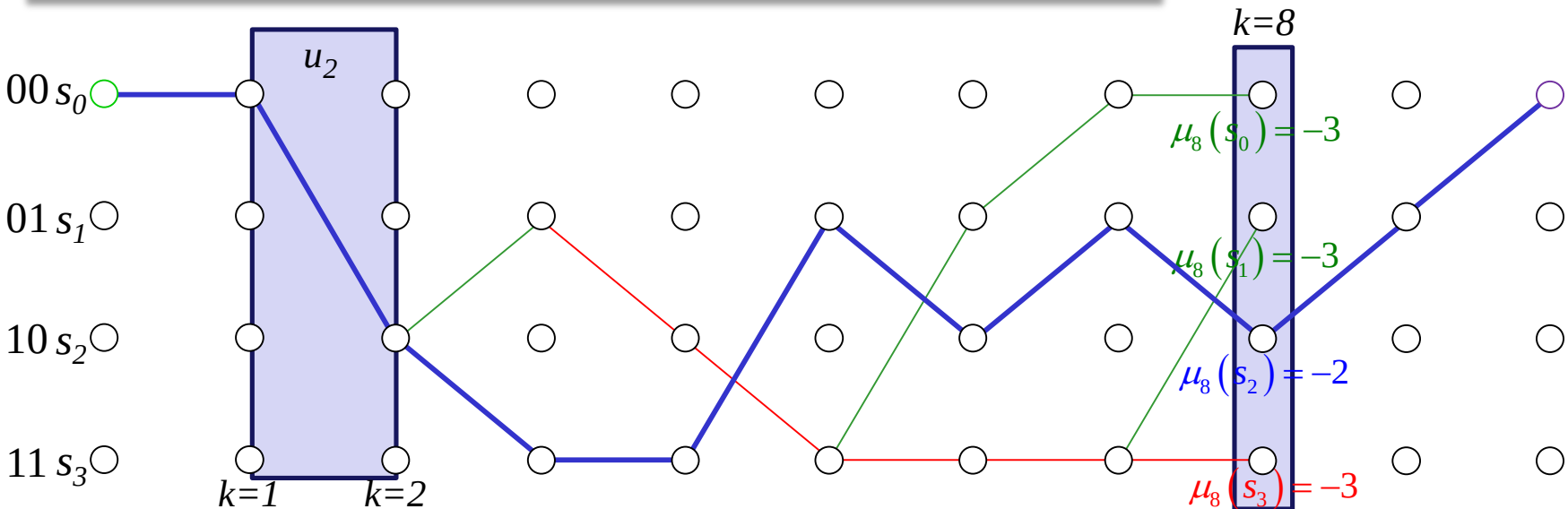
Viterbi Decoding: Hard Decision Decoding



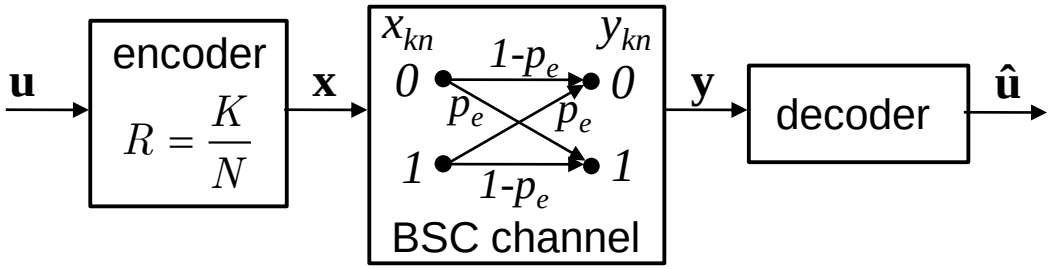


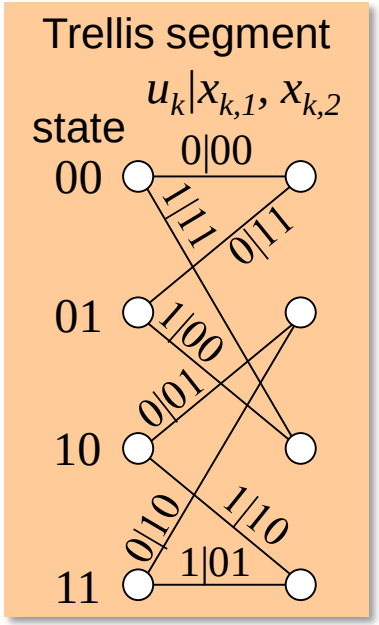
Assume: Decision on info bit u_2 would be taken at $k=8$
→ correct decision

For convolutional codes with memory M, there is virtually no performance degradation, if decisions are taken by trace-back after a fixed decoding delay of $\approx 5M$.
A further simplification is achieved, if tracing back always starts from the zero state rather than from the state with highest metric.

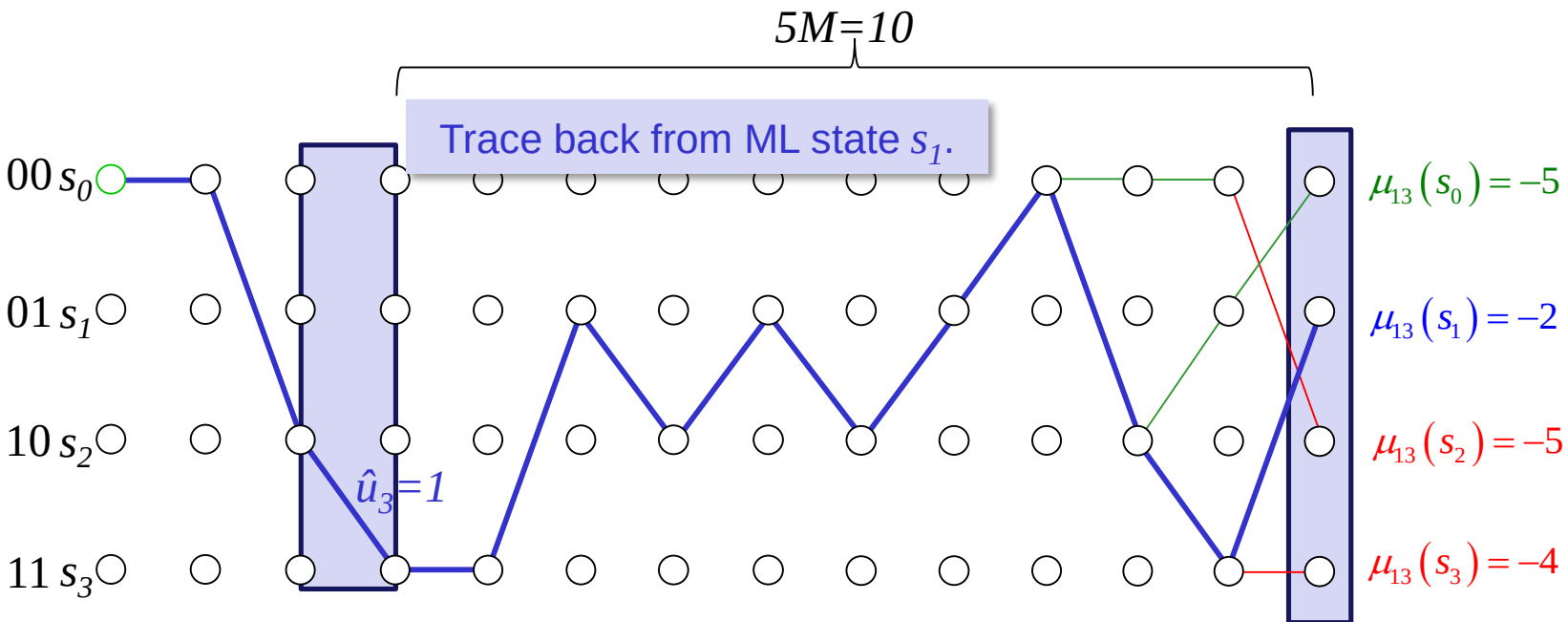


Viterbi Decoding: Hard Decision Decoding

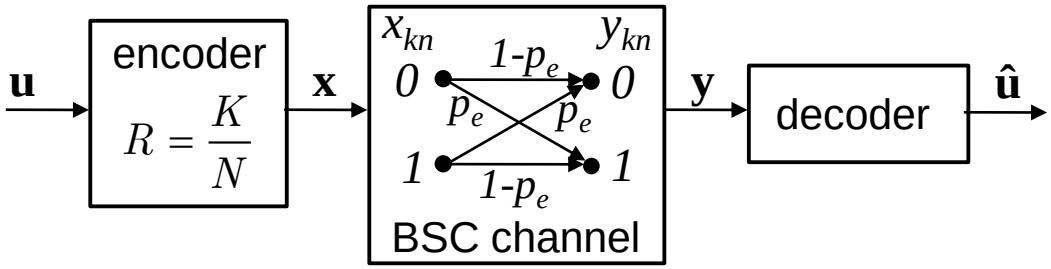


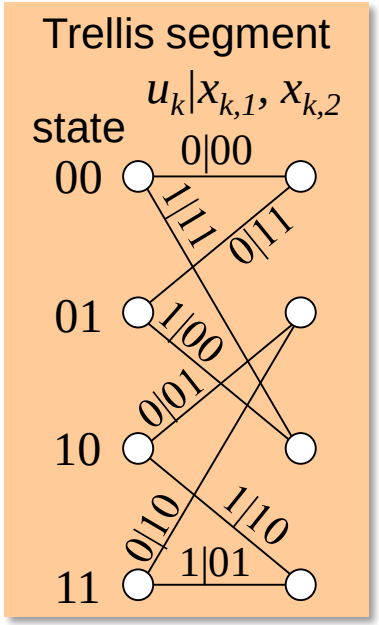


For convolutional codes with memory M, there is virtually no performance degradation, if decisions are taken by trace-back after a fixed decoding delay of $\approx 5M$.
A further simplification is achieved, if tracing back always starts from the zero state rather than from the state with highest metric.

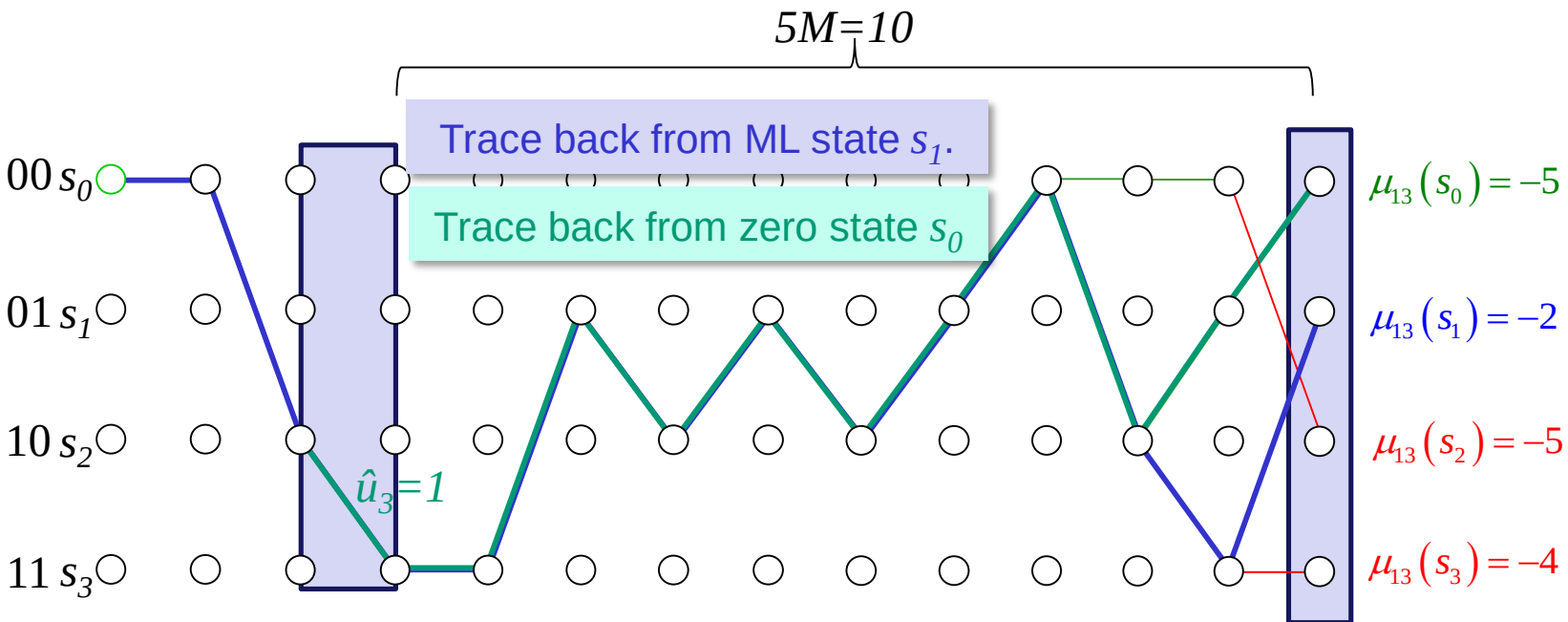


Viterbi Decoding: Hard Decision Decoding

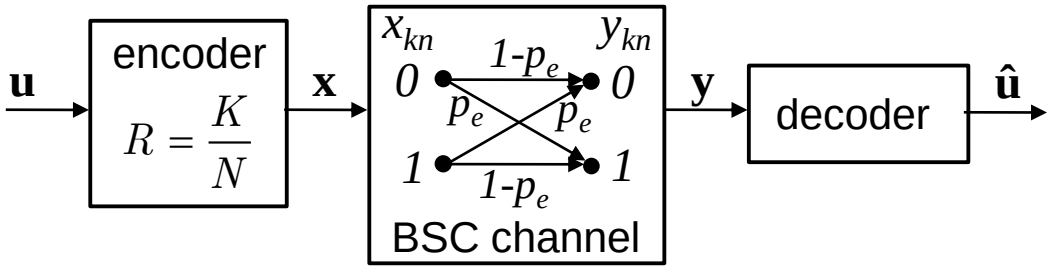


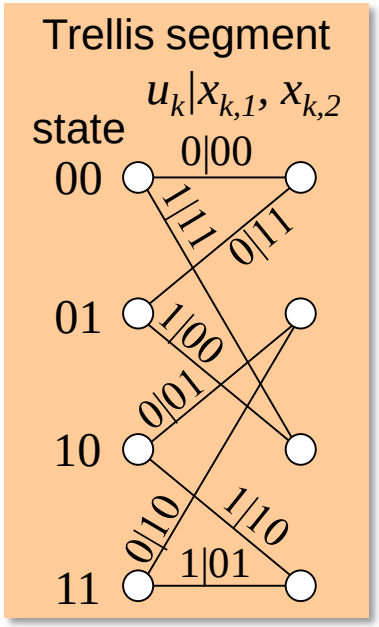


For convolutional codes with memory M, there is virtually no performance degradation, if decisions are taken by trace-back after a fixed decoding delay of $\approx 5M$.
A further simplification is achieved, if tracing back always starts from the zero state rather than from the state with highest metric.



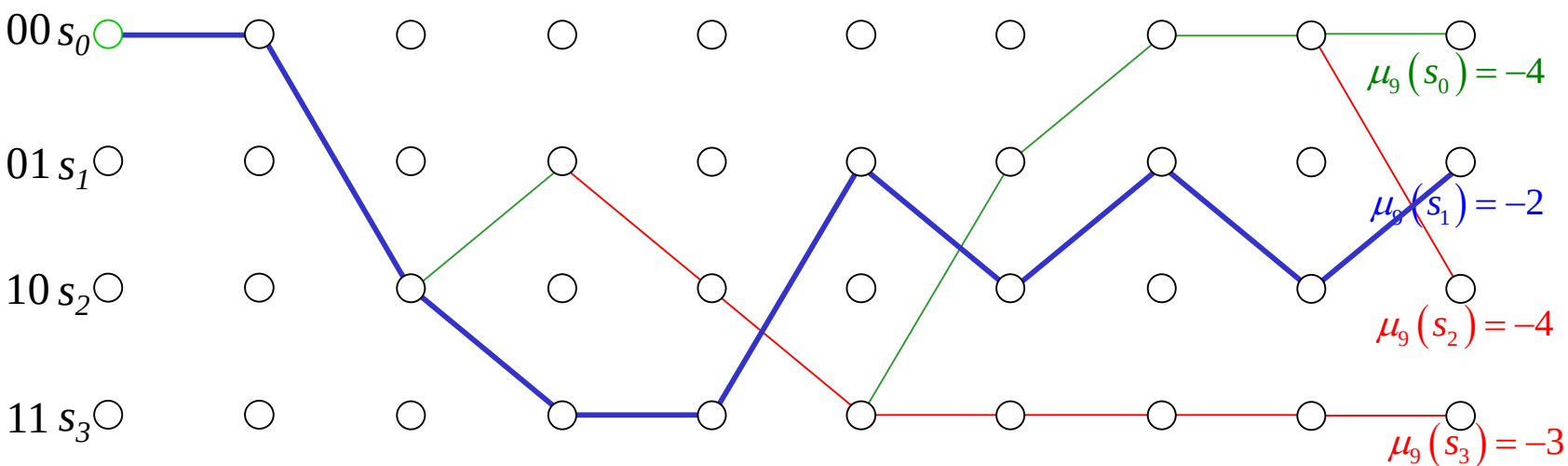
Viterbi Decoding: Hard Decision Decoding



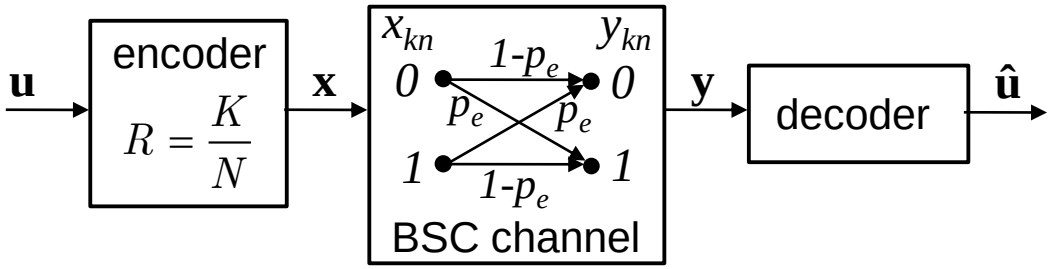


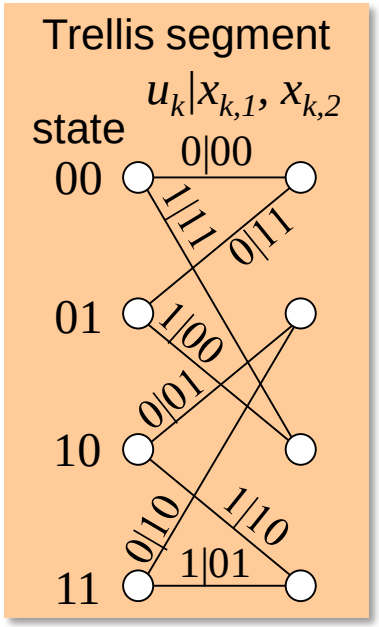
For convolutional codes with memory M, there is virtually no performance degradation, if decisions are taken by trace-back after a fixed decoding delay of $\approx 5M$.
A further simplification is achieved, if tracing back always starts from the zero state rather than from the state with highest metric.

no termination



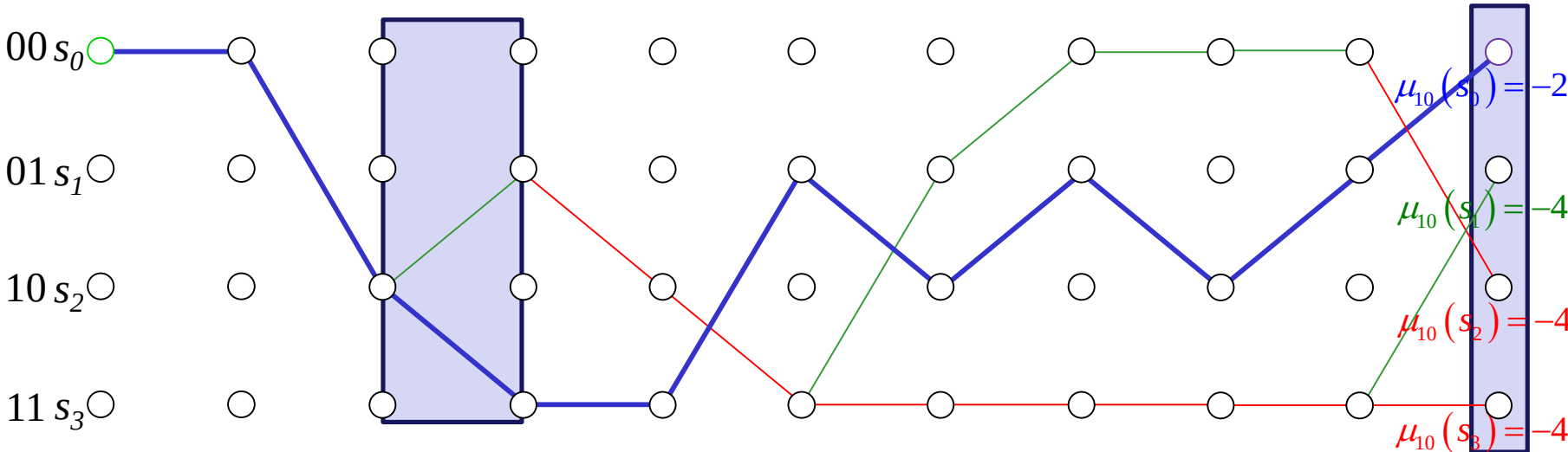
Viterbi Decoding: Hard Decision Decoding



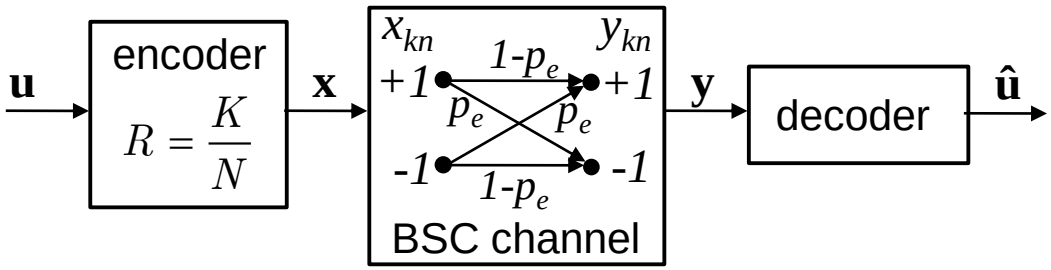


For convolutional codes with memory M, there is virtually no performance degradation, if decisions are taken by trace-back after a fixed decoding delay of $\approx 5M$.
A further simplification is achieved, if tracing back always starts from the zero state rather than from the state with highest metric.

no termination



Viterbi Decoding: Hard Decision Decoding



$x_{k,n}, y_{k,n} \in \{+1, -1\}$

$$P\left(y_{k,n} \mid x_{k,n}\right) = \begin{cases} (1-p_e) & \text{for } y_{k,n} = x_{k,n} \\ p_e & \text{for } y_{k,n} \neq x_{k,n} \end{cases}$$

Maximum likelihood sequence estimation:

$$\max_{\mathbf{x}} P(\mathbf{y} \mid \mathbf{x}) = \max_{\mathbf{x}} \prod_{k=1}^{L} \prod_{n=1}^{N} P\left(y_{k,n} \mid x_{k,n}\right) = \max_{\mathbf{x}} \sum_{k=1}^{L} \sum_{n=1}^{N} \log P\left(y_{k,n} \mid x_{k,n}\right)$$

$$= \max_{\mathbf{x}} \left\{ \left[\sum_{l=1}^{k-1} \sum_{n=1}^{N} \alpha + \beta \log P\left(y_{l,n} \mid x_{l,n}\right) \right] + \underbrace{\left[\sum_{n=1}^{N} \alpha + \beta \log P\left(y_{k,n} \mid x_{k,n}\right) \right]} + \left[\sum_{t=k+1}^{L} \sum_{n=1}^{N} \alpha + \beta \log P\left(y_{t,n} \mid x_{t,n}\right) \right] \right\}$$

metric increment $\Delta\mu_{\mathrm{k}}\left(s_i, s_j\right) = \alpha + \beta \log P\left(y_{k,n} \mid x_{k,n}\right), \qquad \beta > 0$

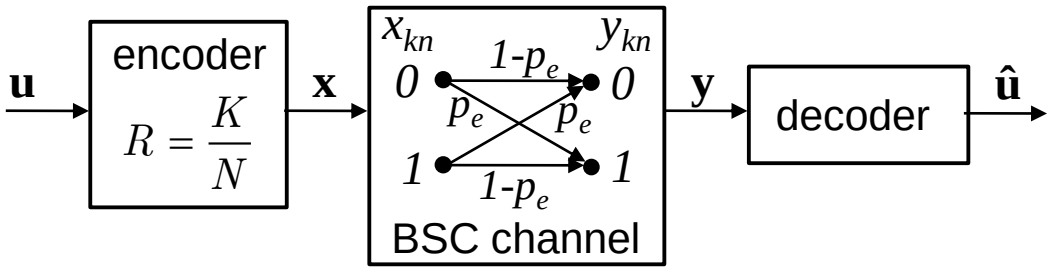
$$\Rightarrow \begin{cases} \alpha + \beta \log(1-p_e) = -1 - \beta \log(p_e) + \beta \log(1-p_e) \\ \alpha + \beta \log(p_e) = -1 - \beta \log(p_e) + \beta \log(p_e) \end{cases}$$

$$= \begin{cases} -1 + \beta \log \dfrac{1-p_e}{p_e} = +1 = y_{k,n} x_{k,n} & \text{for } y_{k,n} = x_{k,n} \\ -1 \qquad\qquad\qquad = y_{k,n} x_{k,n} & \text{for } y_{k,n} \neq x_{k,n} \end{cases}$$

$$\boxed{\Delta\mu_{\mathrm{k}}\left(s_i, s_j\right) = \sum_{n=1}^{N} \left(y_{k,n} x_{k,n}\right)}$$

$$\beta = \frac{2}{\log \dfrac{1-p_e}{p_e}} > 0 \text{ for } p_e < \frac{1}{2}$$

Viterbi Decoding: Hard Decision Decoding



Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\}$$

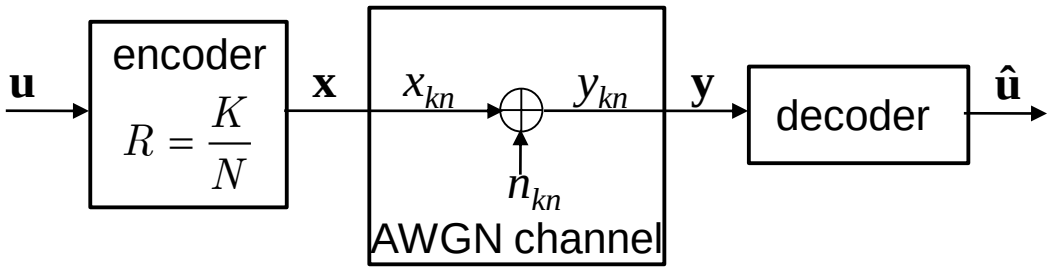
Metric increment for transition from state s_i at time $k\text{-}1$ to state s_j at time k :

$$\Delta\mu_k(s_i, s_j) = \sum_{n=1}^{N} \left[\alpha + \beta \log P\left(y_{k,n} \mid x_{k,n}\right) \right]$$

$$\Delta\mu_k(s_i, s_j) = -\sum_{n=1}^{N} \left(y_{k,n} \oplus x_{k,n}\right) \qquad \text{for } x_{k,n}, y_{k,n} \in \{0,1\}$$

$$\Delta\mu_k(s_i, s_j) = \sum_{n=1}^{N} y_{k,n} x_{k,n} \qquad \text{for } x_{k,n}, y_{k,n} \in \{+1,-1\}$$

Viterbi Decoding: Soft Decision Decoding



$$p\left(y_{k,n} \mid x_{k,n}\right) = \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{\left(y_{k,n}-x_{k,n}\right)^2}{2\sigma_n^2}}$$

Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k\left(s_j\right) = \max_i \left\{\mu_{k-1}\left(s_i\right) + \Delta\mu_k\left(s_i, s_j\right)\right\}$$

Metric increment for transition from state s_i at time k-1 to state s_j at time k :

$$\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N}\left[\alpha + \beta \log p\left(y_{k,n} \mid x_{k,n}\right)\right]$$

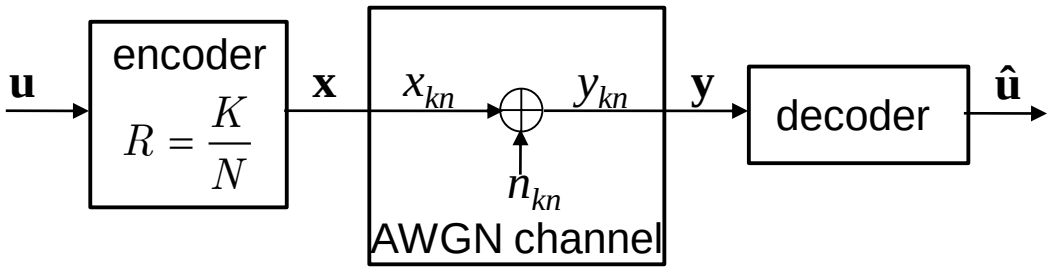
const for $x_{k,n}^2 = \text{const}$

$$\Rightarrow \log p\left(y_{k,n} \mid x_{k,n}\right) = \log \frac{1}{\sqrt{2\pi\sigma_n^2}} - \frac{\left(y_{k,n} - x_{k,n}\right)^2}{2\sigma_n^2} = \underbrace{\log \frac{1}{\sqrt{2\pi\sigma_n^2}}}_{\text{const}} - \underbrace{\frac{1}{2\sigma_n^2}}_{\text{const}}\left(\cancel{y_{k,n}^2} - 2y_{k,n}x_{k,n} + \cancel{x_{k,n}^2}\right)$$

$$\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N}\left(y_{k,n}x_{k,n} - \frac{1}{2}x_{k,n}^2\right) \qquad \text{for } x_{k,n} \in \{0,1\}$$

$$\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N} y_{k,n}x_{k,n} \qquad \text{for } x_{k,n} \in \{+1,-1\}$$

Viterbi Decoding: Soft Decision Decoding



$$p\left(y_{k,n} \mid x_{k,n}\right) = \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{\left(y_{k,n}-x_{k,n}\right)^2}{2\sigma_n^2}}$$

Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k\left(s_j\right) = \max_i \left\{\mu_{k-1}\left(s_i\right) + \Delta\mu_k\left(s_i, s_j\right)\right\}$$

Metric increment for transition from state s_i at time k-1 to state s_j at time k :

$$\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N}\left[\alpha + \beta \log p\left(y_{k,n} \mid x_{k,n}\right)\right]$$

const

const

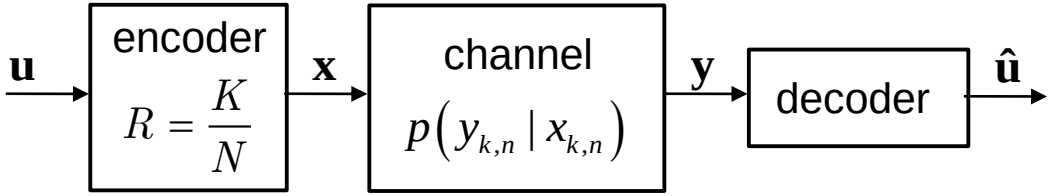
const for $x_{k,n}^2 = \text{const}$

$$\Rightarrow \log p\left(y_{k,n} \mid x_{k,n}\right) = \log \frac{1}{\sqrt{2\pi\sigma_n^2}} - \frac{\left(y_{k,n} - x_{k,n}\right)^2}{2\sigma_n^2} = \log \frac{1}{\sqrt{2\pi\sigma_n^2}} - \frac{1}{2\sigma_n^2}\left(y_{k,n}^2 - 2y_{k,n}x_{k,n} + x_{k,n}^2\right)$$

$$L\left(y_{k,n} \mid x_{k,n}\right) = L_c\left(x_{k,n}\right) = \frac{2}{\sigma_n^2} y_{k,n}$$

$$\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N} y_{k,n}x_{k,n} = \frac{1}{2}\sum_{n=1}^{N} L_c\left(x_{k,n}\right)x_{k,n} \qquad \text{for } x_{k,n} \in \{+1, -1\}$$

Viterbi Decoding: Soft Decision Decoding



Maximum likelihood sequence estimation:

Metric for state s_j at time k:

$$\mu_k(s_j) = \max_i \left\{ \mu_{k-1}(s_i) + \Delta\mu_k(s_i, s_j) \right\}$$

Metric increment for transition from state s_i at time k-1 to state s_j at time k :

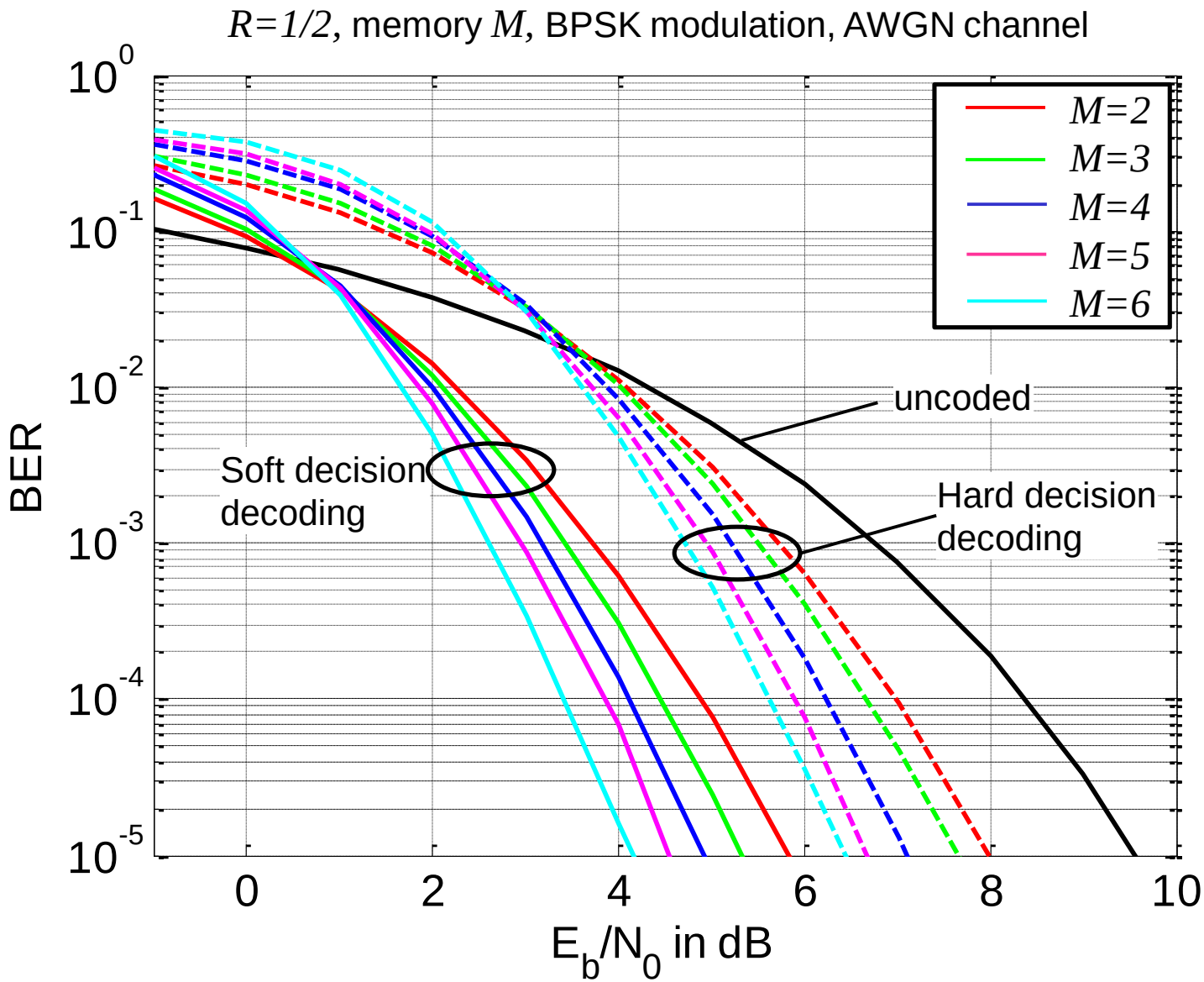
$$\Delta\mu_k(s_i, s_j) = \sum_{n=1}^{N} \left[\alpha + \beta \log p(y_{k,n} \mid x_{k,n}) \right]$$

$$L(y_{k,n} \mid x_{k,n}) = L_c(x_{k,n}) = \log \frac{p(y_{k,n} \mid x_{k,n} = +1)}{p(y_{k,n} \mid x_{k,n} = -1)}$$

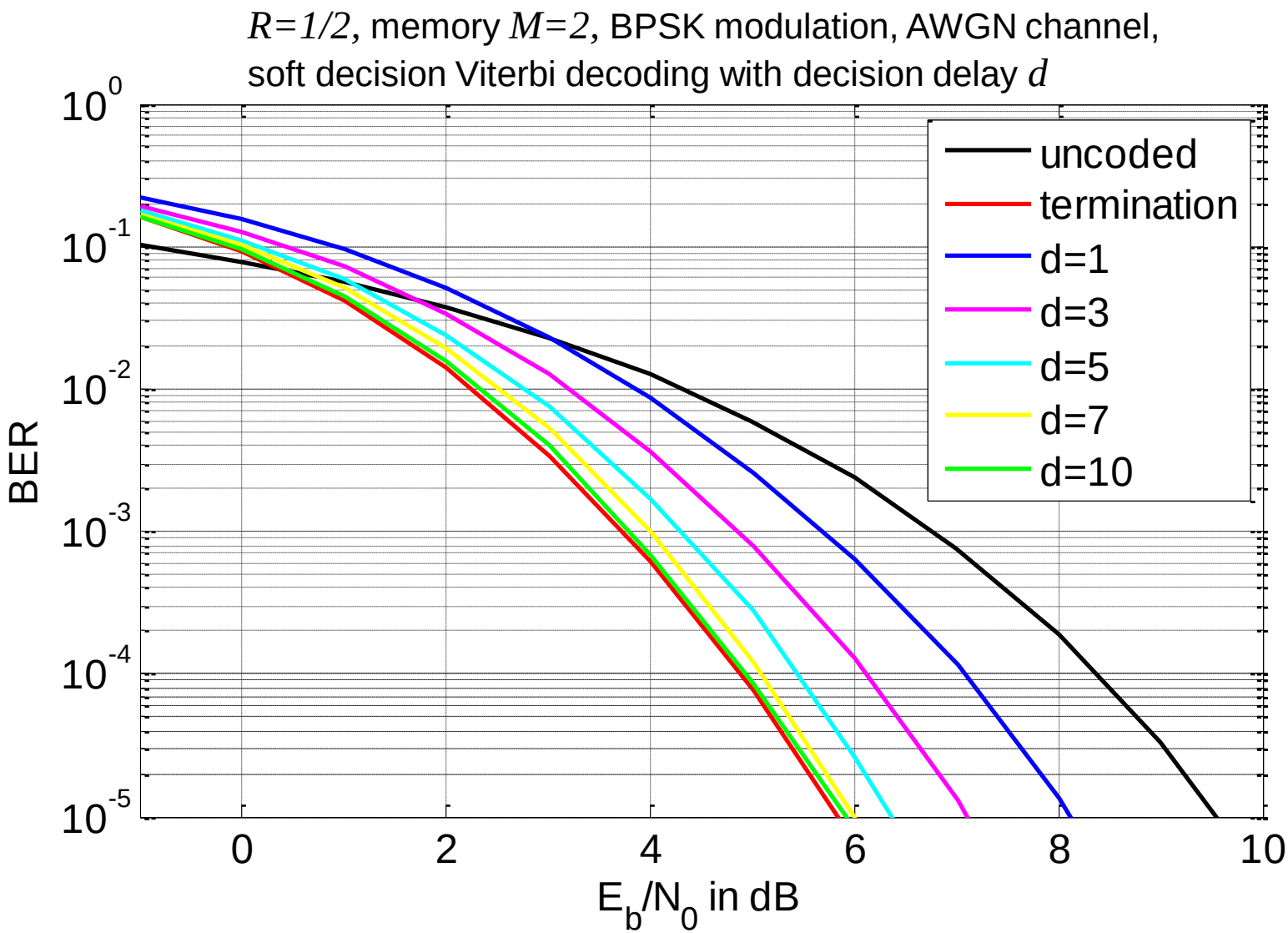
$$\Rightarrow p(y_{k,n} \mid x_{k,n} = \pm 1) = \underbrace{\frac{e^{L_c(x_{k,n})/2}}{1 + e^{L_c(x_{k,n})}}}_{\text{const}} e^{x_{k,n} \cdot L_c(x_{k,n})/2} \Rightarrow \log p(y_{k,n} \mid x_{k,n} = \pm 1) = \text{const} + \frac{1}{2} x_{k,n} L_c(x_{k,n})$$

$$\Delta\mu_k(s_i, s_j) = \sum_{n=1}^{N} L_c(x_{k,n}) x_{k,n} \qquad \text{for } x_{k,n} \in \{+1, -1\}$$

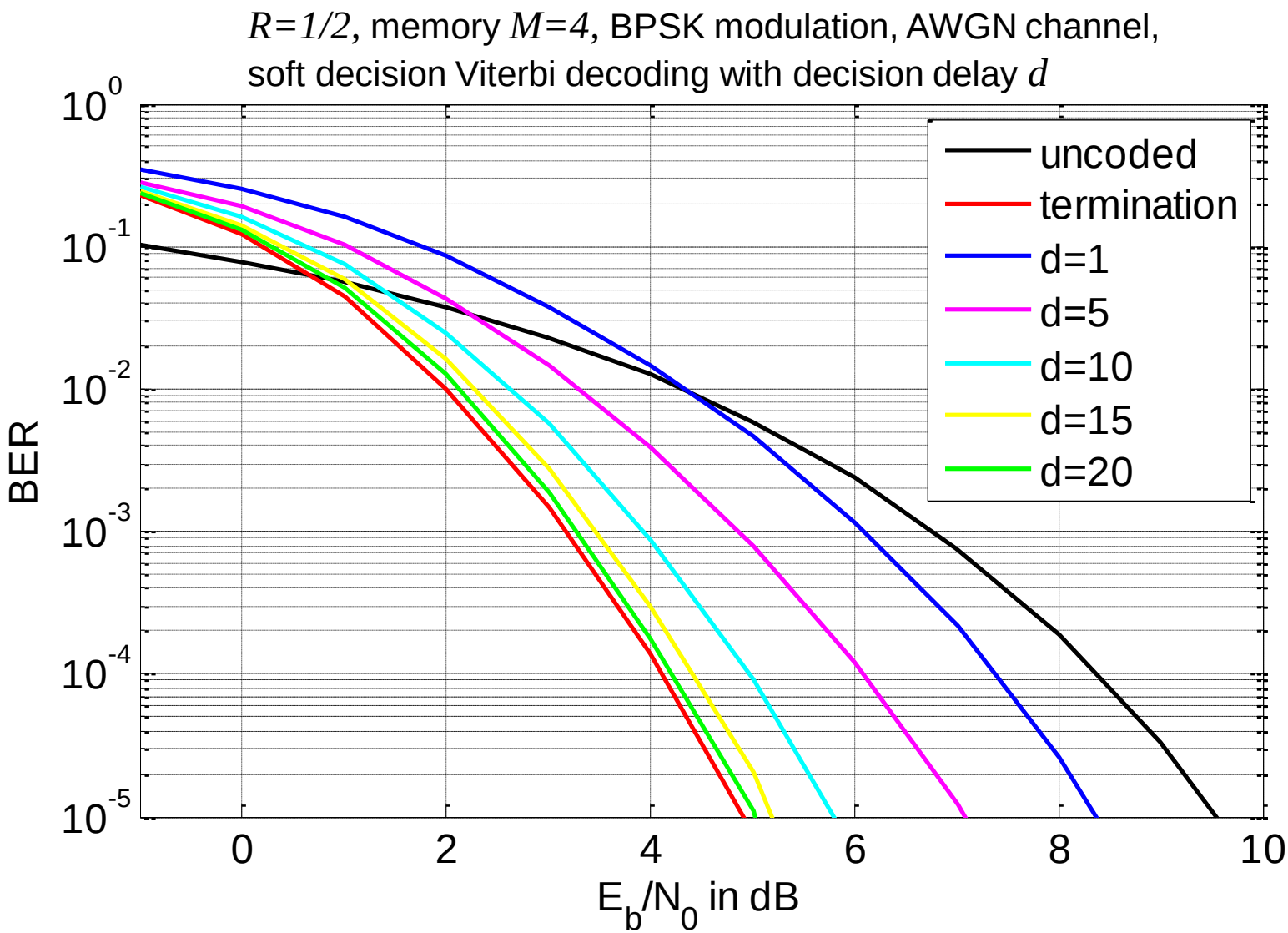
BER Performance of Convolutional Codes: Hard vs. Soft Decision Decoding



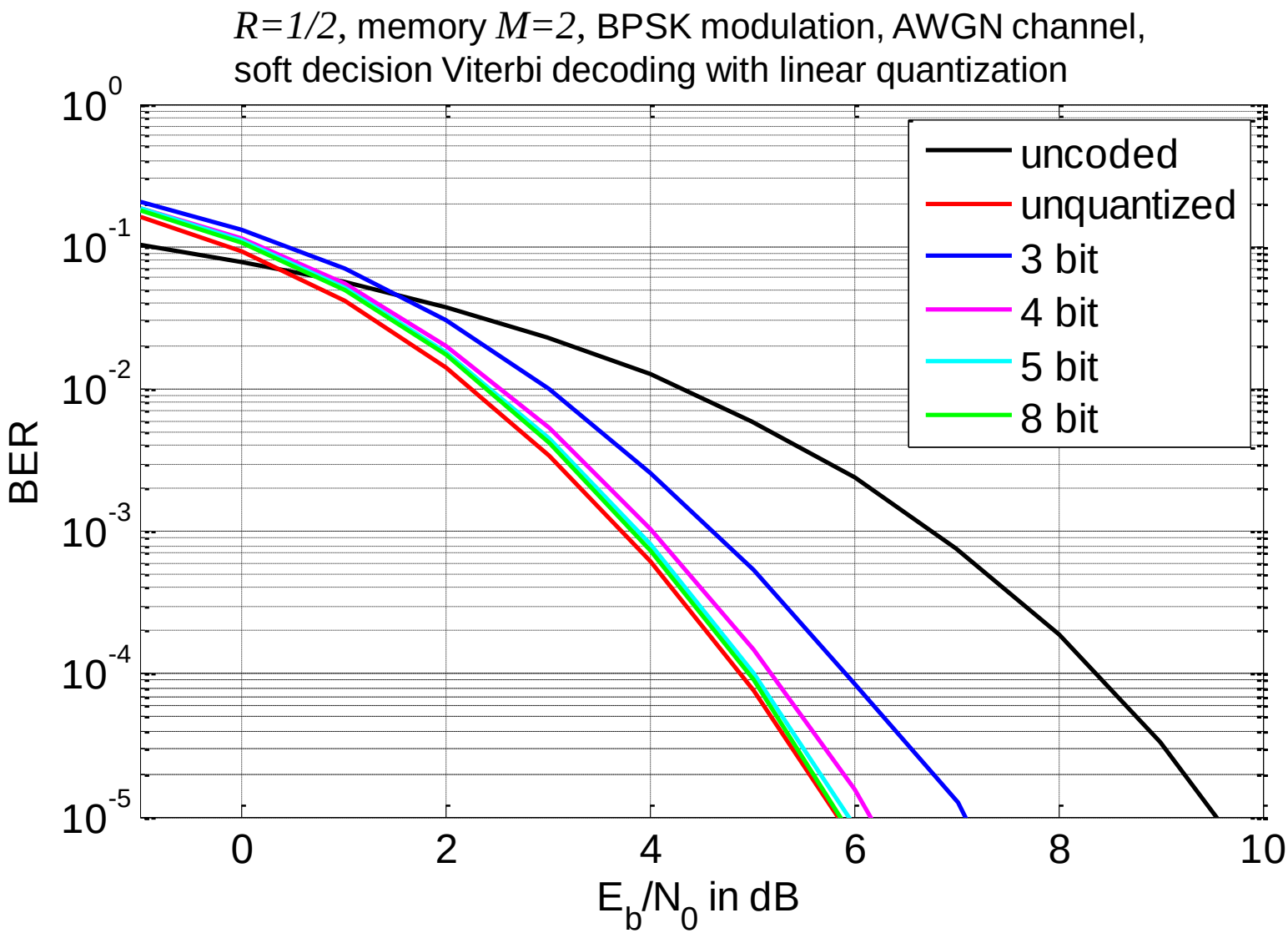
BER Performance of Convolutional Codes: Viterbi Decoding with Decision Delay d



BER Performance of Convolutional Codes: Viterbi Decoding with Decision Delay d



BER Performance of Convolutional Codes: Viterbi Decoding with Quantized Soft Input



Asymptotic Coding Gain of Optimum Convolutional Codes

$$R = \frac{1}{2}$$

Memory M	d_{free}	G_{hard} in dB	G_{soft} in dB
2	5	0.97	3.98
3	6	1.76	4.77
4	7	2.43	5.44
5	8	3.01	6.02
6	10	3.98	6.99
7	10	3.98	6.99
8	12	4.77	7.78

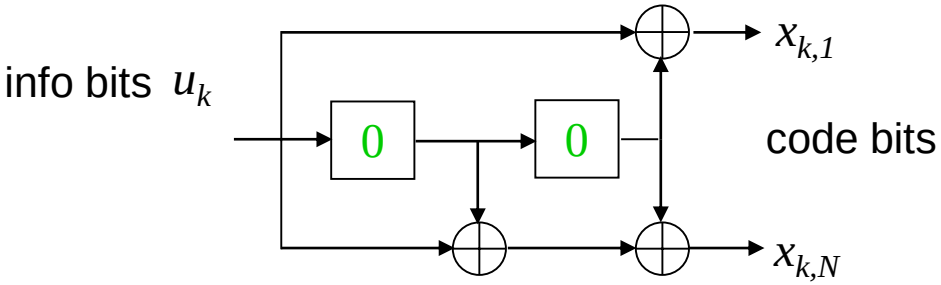
$$R = \frac{1}{3}$$

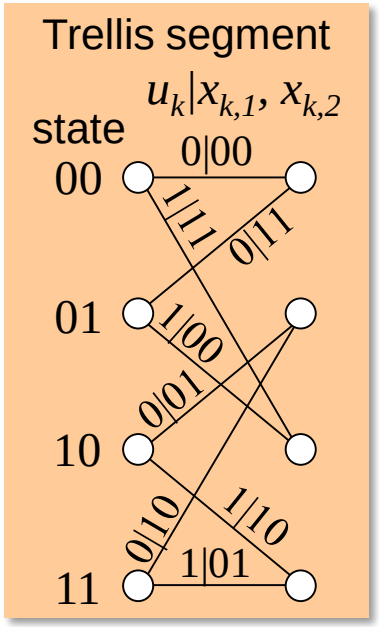
Memory M	d_{free}	G_{hard} in dB	G_{soft} in dB
2	8	1.25	4.26
3	10	2.22	5.23
4	12	3.01	6.02
5	13	3.36	6.37
6	15	3.98	6.99
7	16	4.26	7.27
8	18	4.77	7.78

Viterbi Decoding: Complexity

Memory: M
q-ary symbol alphabeth

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



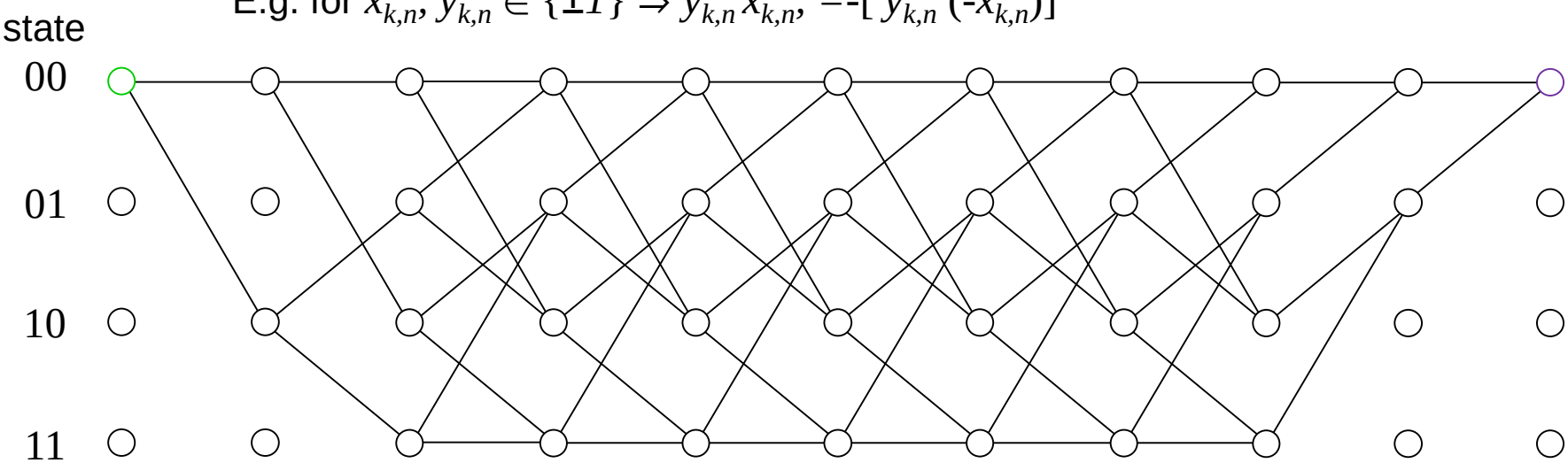


- q^M states $\Rightarrow$ complexity grows exponentially with memory M.
- q^K transitions per state $\Rightarrow$ $q^K q^M = q^{M+K}$ transitions per trellis segment.
- N code symbols per transition segment

  $\Rightarrow$ $q^N < q^{M+K}$ different metric increments per trellis segment.

  Symmetries can be exploited in metric calculation:

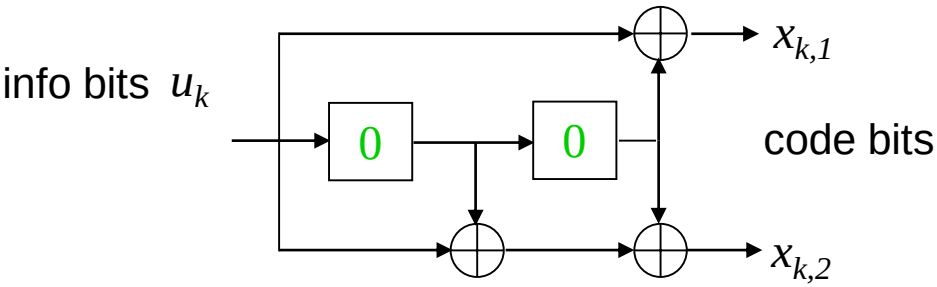
  E.g. for $x_{k,n}, y_{k,n} \in \{\pm 1\} \Rightarrow y_{k,n} x_{k,n}, = -[\, y_{k,n} (-x_{k,n})]$



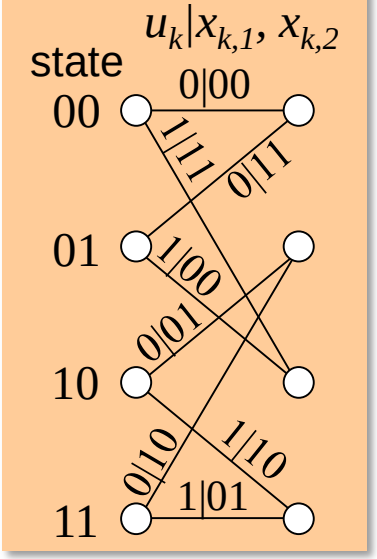
Convolutional Codes: Free Distance

Example:
Memory: $M=2$
Constraint length: $M+1=3$

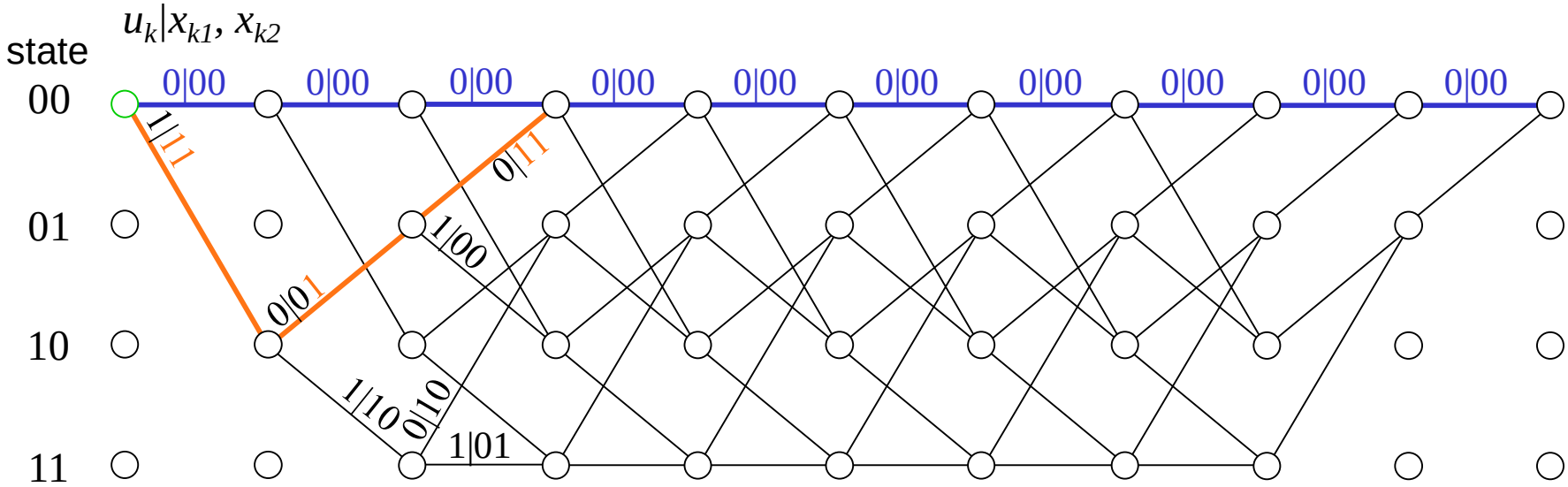
Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Trellis segment



Two code words differ in at least $d_{\text{free}}=5$ positions, where d_{free} is the free distance of the code.

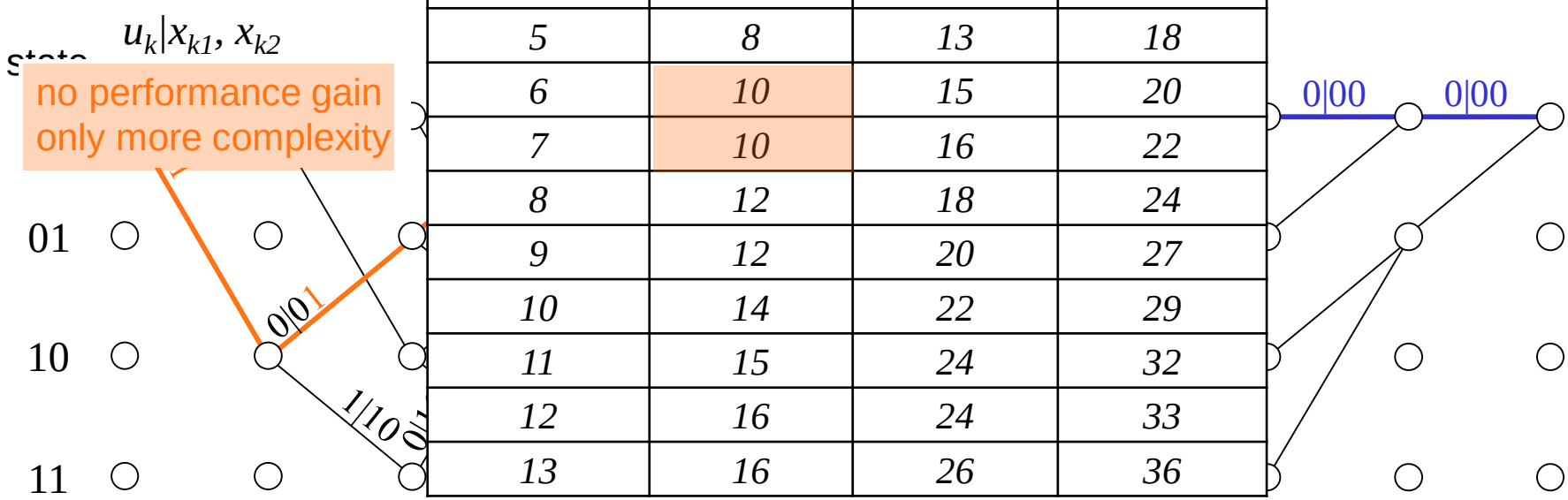


Convolutional Codes: Free Distance and Optimum Codes

The free distance d_{free} of a convolutional code is defined as the minimum Hamming weight among all code sequences.

A convolutional code is optimum, if the free distance d_{free} is maximum among all convolutional codes with same rate R and same memory M.

Memory M	d_{free}		
	$R=1/2$	$R=1/3$	$R=1/4$
2	5	8	10
3	6	10	13
4	7	12	16
5	8	13	18
6	10	15	20
7	10	16	22
8	12	18	24
9	12	20	27
10	14	22	29
11	15	24	32
12	16	24	33
13	16	26	36

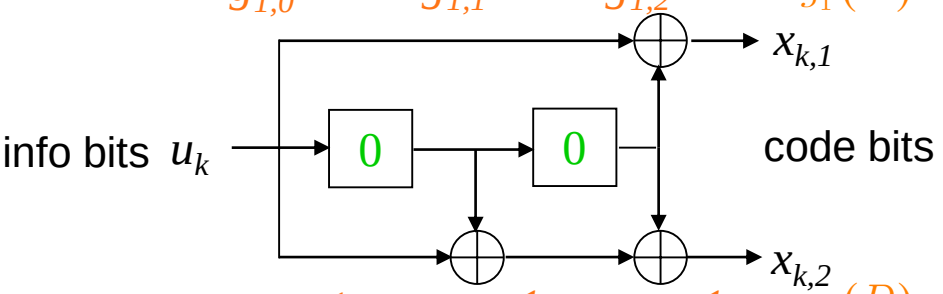


Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$

$g_{1,0}=1$ $g_{1,1}=0$ $g_{1,2}=1$ $g_1(D) = g_{1,0} + g_{1,2}D^2$



$g_{2,0}=1$ $g_{2,1}=1$ $g_{2,2}=1$ $g_2(D) = g_{2,0} + g_{2,1}D + g_{2,2}D^2$

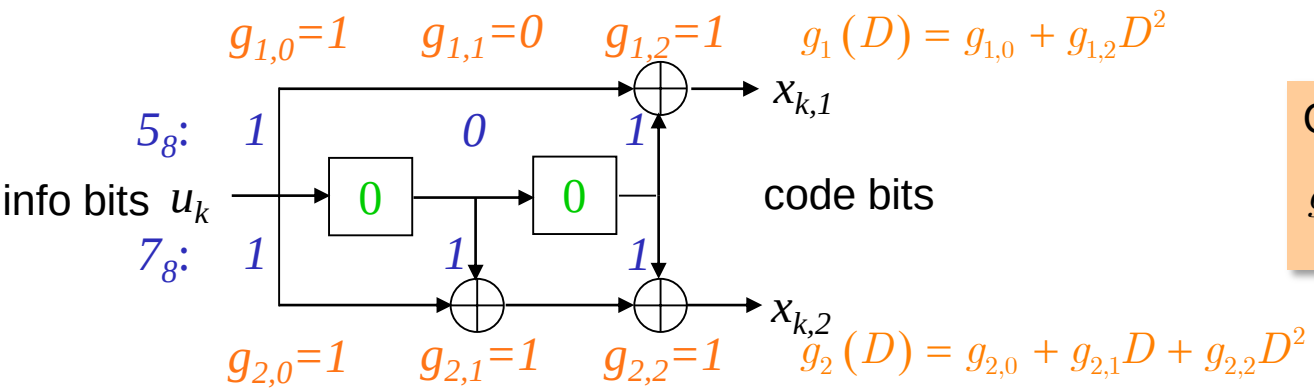
Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Octal representation: $(5,7)_8$ convolutional code

$$5_8 \Rightarrow 101 \hat{=} g_1(D) = 1 + 0 \cdot D + 1 \cdot D^2$$

$$7_8 \Rightarrow 111 \hat{=} g_2(D) = 1 + 1 \cdot D + 1 \cdot D^2$$

General octal representation of convolutional codes:

$$(g_n)_8 \Rightarrow \underset{\text{MSB}}{g_0}, g_1, \ldots, \underset{\text{LSB}}{g_M} \hat{=} g_n(D) = g_0 + g_1 D + \ldots + g_M D^M$$

Example: Memory $M=4$ $(23,35)_8$ convolutional code:

$$23_8 \Rightarrow 010\ 011 \hat{=} g_1(D) = 1 + 0 \cdot D + 0 \cdot D^2 + 1 \cdot D^3 + 1 \cdot D^4$$

$$35_8 \Rightarrow 011\ 101 \hat{=} g_2(D) = 1 + 1 \cdot D + 1 \cdot D^2 + 0 \cdot D^3 + 1 \cdot D^4$$

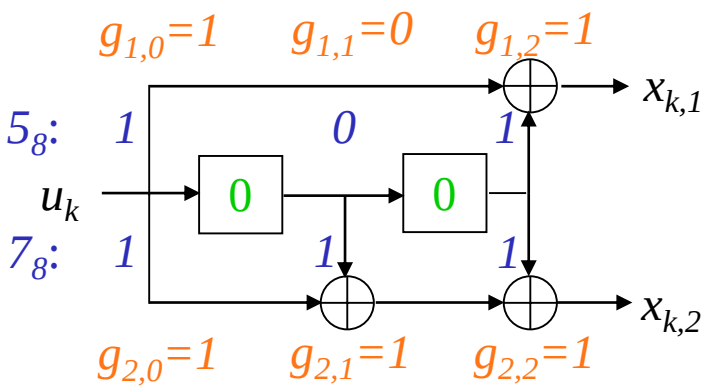
Convolutional Codes: Octal Description

Octal representation of convolutional codes:

$$(g_n)_8 \Rightarrow \underset{\substack{\uparrow \\ \text{MSB}}}{g_0}, g_1, \ldots, \underset{\substack{\uparrow \\ \text{LSB}}}{g_M} \triangleq g_n(D) = g_0 + g_1 D + \ldots + g_M D^M$$

Examples: Code rate: $R = \frac{K}{N} = \frac{1}{2}$

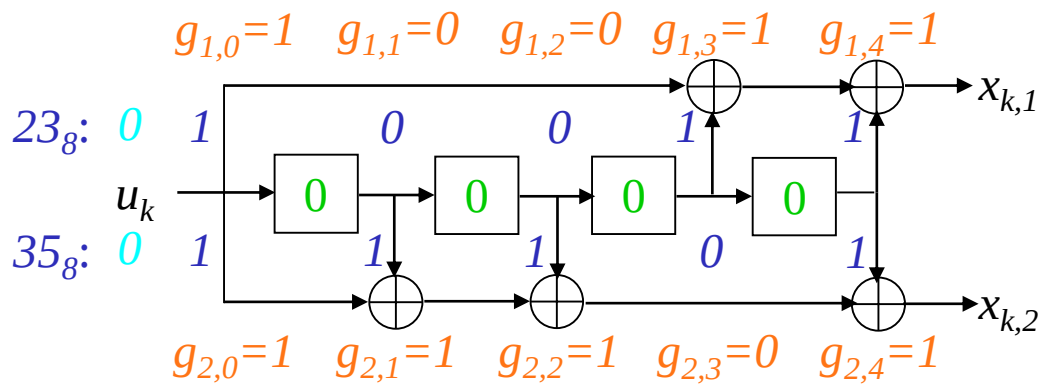
$(5,7)_8$ code (memory $M=2$)



$$5_8 \Rightarrow 101 \triangleq g_1(D) = 1 + 0 \cdot D + 1 \cdot D^2$$

$$7_8 \Rightarrow 111 \triangleq g_2(D) = 1 + 1 \cdot D + 1 \cdot D^2$$

$(23,35)_8$ code (memory $M=4$)



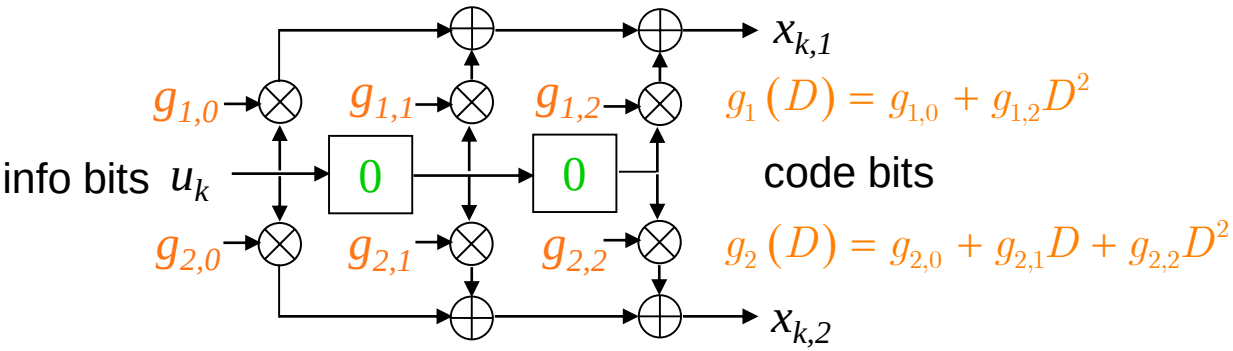
$$23_8 \Rightarrow 010\ 011 \triangleq g_1(D) = 1 + 0 \cdot D + 0 \cdot D^2 + 1 \cdot D^3 + 1 \cdot D^4$$

$$35_8 \Rightarrow 011\ 101 \triangleq g_2(D) = 1 + 1 \cdot D + 1 \cdot D^2 + 0 \cdot D^3 + 1 \cdot D^4$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

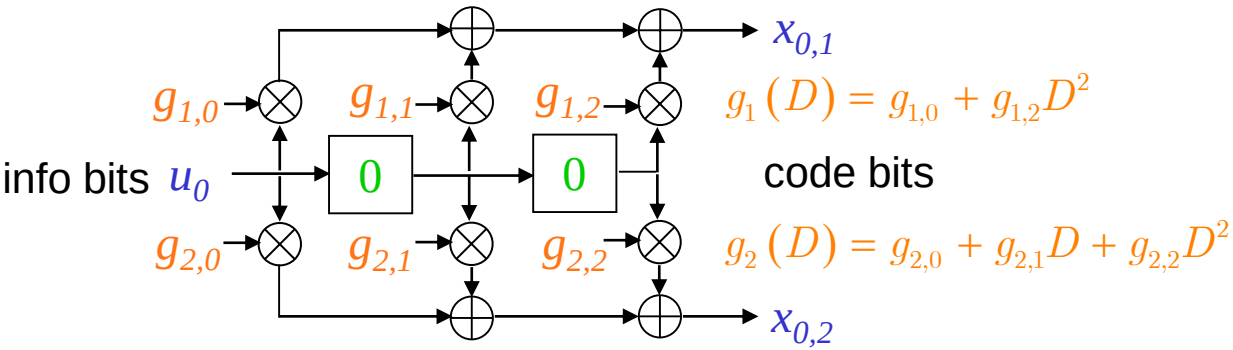
$$x_n(D) = g_n(D)u(D) = \left(g_{n,0} + g_{n,1}D + g_{n,2}D^2 + \ldots + g_{n,M}D^M\right)\left(u_0 + u_1 D + u_2 D^2 + \ldots + u_{L-1}D^{L-1}\right)$$

$$\begin{aligned}
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

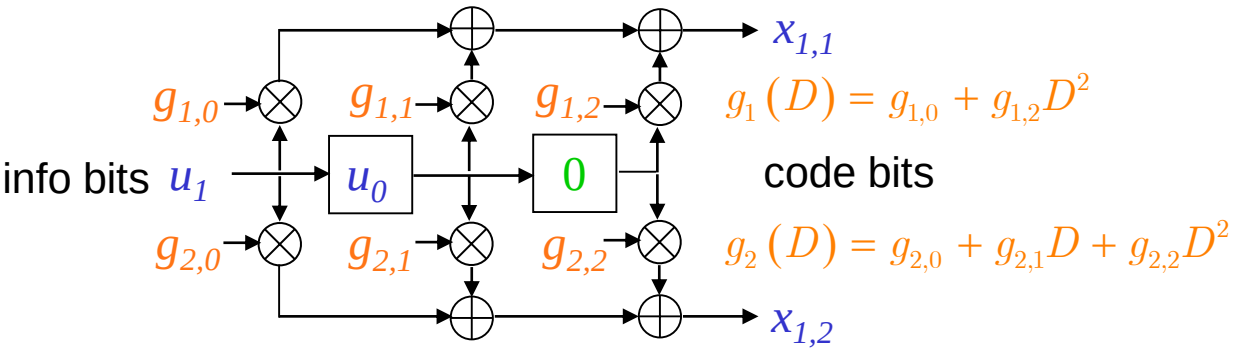
$$x_n(D) = g_n(D)u(D) = \left(g_{n,0} + g_{n,1}D + g_{n,2}D^2 + \ldots + g_{n,M}D^M\right)\left(u_0 + u_1 D + u_2 D^2 + \ldots + u_{L-1}D^{L-1}\right)$$

$$\begin{aligned}
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

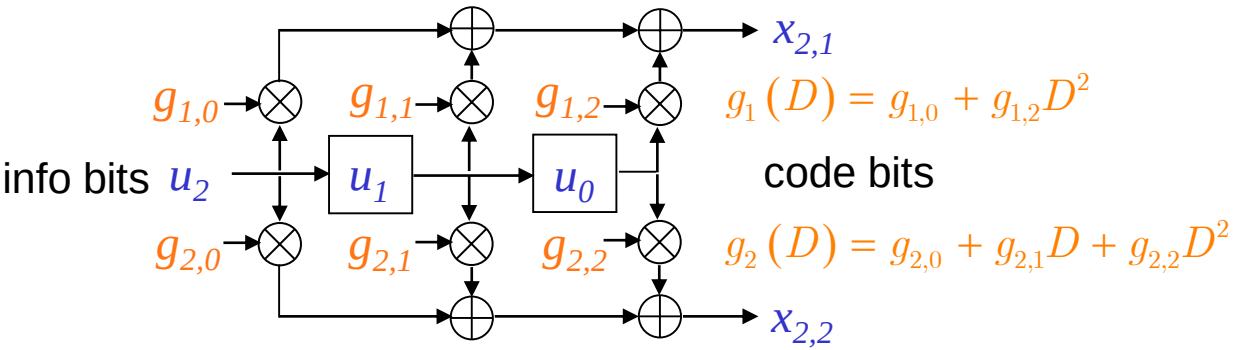
$$x_n(D) = g_n(D)u(D) = \left(g_{n,0} + g_{n,1}D + g_{n,2}D^2 + \ldots + g_{n,M}D^M\right)\left(u_0 + u_1 D + u_2 D^2 + \ldots + u_{L-1}D^{L-1}\right)$$

$$\begin{aligned}
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

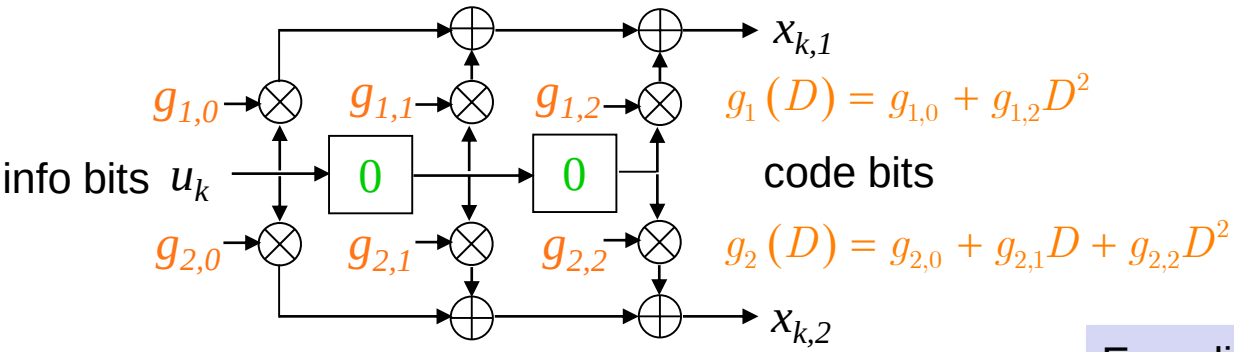
$$x_n(D) = g_n(D)u(D) = \left(g_{n,0} + g_{n,1}D + g_{n,2}D^2 + \ldots + g_{n,M}D^M\right)\left(u_0 + u_1 D + u_2 D^2 + \ldots + u_{L-1}D^{L-1}\right)$$

$$\begin{aligned}
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

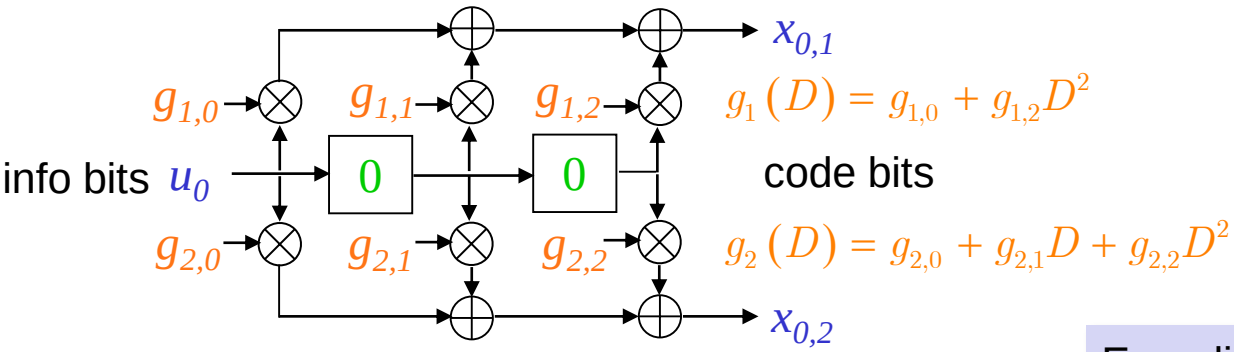
Encoding: $\mathbf{x} = \mathbf{G}\mathbf{u}$

$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{M,1} \\ x_{M,2} \\ \vdots \\ x_{L-1,1} \\ x_{L-1,2} \end{bmatrix} = \begin{bmatrix} g_{1,0} & 0 & 0 & 0 \\ g_{2,0} & 0 & \ddots & \vdots \\ g_{1,1} & g_{1,0} & \ddots & 0 \\ g_{2,1} & g_{2,0} & 0 & \ddots \\ \vdots & g_{1,1} & g_{1,0} & \\ g_{1,M} & \vdots & g_{2,0} & \\ g_{2,M} & \vdots & \vdots & \ddots \\ 0 & g_{1,M} & \vdots & \ddots \\ 0 & g_{2,M} & \vdots & \\ \vdots & \ddots & g_{2,M} & \ddots \\ 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

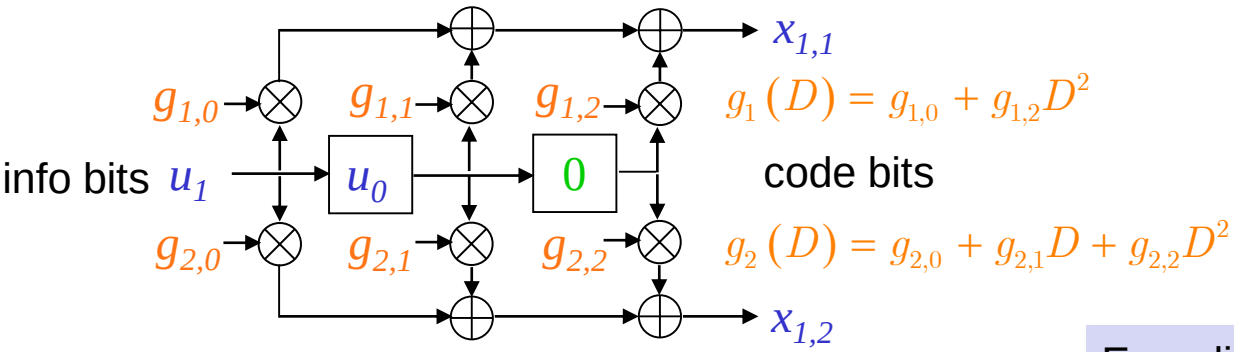
Encoding: $\mathbf{x} = \mathbf{G}\mathbf{u}$

$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{M,1} \\ x_{M,2} \\ \vdots \\ x_{L-1,1} \\ x_{L-1,2} \end{bmatrix} = \begin{bmatrix} g_{1,0} & 0 & 0 & 0 \\ g_{2,0} & 0 & \ddots & \vdots \\ g_{1,1} & g_{1,0} & \ddots & 0 \\ g_{2,1} & g_{2,0} & 0 & \ddots \\ \vdots & g_{1,1} & g_{1,0} & \\ g_{1,M} & \vdots & g_{2,0} & \\ g_{2,M} & \vdots & \vdots & \ddots \\ 0 & g_{1,M} & \vdots & \ddots \\ 0 & g_{2,M} & \vdots & \\ \vdots & \ddots & g_{2,M} & \ddots \\ 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

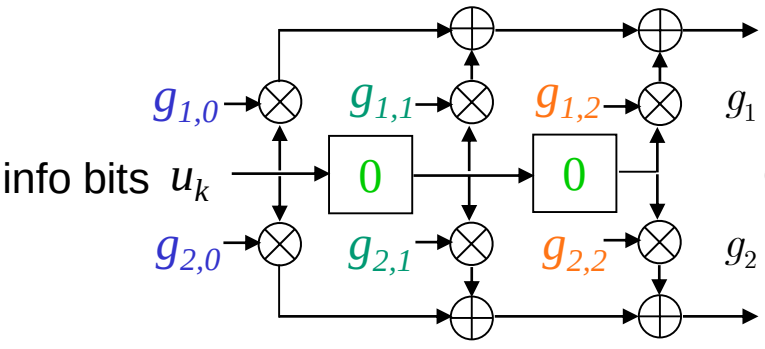
Encoding: $\mathbf{x} = \mathbf{G}\mathbf{u}$

$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{M,1} \\ x_{M,2} \\ \vdots \\ x_{L-1,1} \\ x_{L-1,2} \end{bmatrix} = \begin{bmatrix} g_{1,0} & 0 & 0 & 0 \\ g_{2,0} & 0 & \ddots & \vdots \\ g_{1,1} & g_{1,0} & \ddots & 0 \\ g_{2,1} & g_{2,0} & 0 & \ddots \\ \vdots & g_{1,1} & g_{1,0} & \\ g_{1,M} & \vdots & g_{2,0} & \\ g_{2,M} & \vdots & \vdots & \ddots \\ 0 & g_{1,M} & \vdots & \ddots \\ 0 & g_{2,M} & \vdots & \\ \vdots & \ddots & g_{2,M} & \ddots \\ 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



$g_1(D) = g_{1,0} + g_{1,2}D^2$

code bits

$g_2(D) = g_{2,0} + g_{2,1}D + g_{2,2}D^2$

Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Encoding: $\mathbf{x} = \mathbf{G}\mathbf{u}$

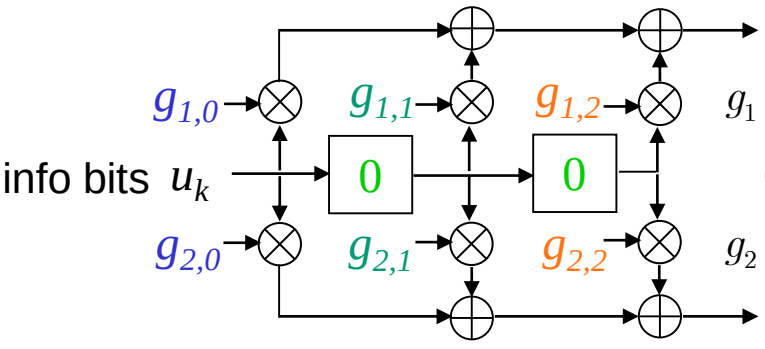
$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{M,1} \\ x_{M,2} \\ \vdots \\ x_{L-1,1} \\ x_{L-1,2} \end{bmatrix} = \begin{bmatrix} g_{1,0} & 0 & 0 & 0 \\ g_{2,0} & 0 & \ddots & \vdots \\ g_{1,1} & g_{1,0} & \ddots & 0 \\ g_{2,1} & g_{2,0} & 0 & \ddots \\ \vdots & g_{1,1} & g_{1,0} & \\ g_{1,M} & \vdots & g_{2,0} & \\ g_{2,M} & \vdots & \vdots & \ddots \\ 0 & g_{1,M} & \vdots & \ddots \\ 0 & g_{2,M} & \vdots & \ddots \\ \vdots & \ddots & g_{2,M} & \ddots \\ 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

$\mathbf{G}_0$ $\mathbf{G}_1$ $\mathbf{G}_M$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



$g_1(D) = g_{1,0} + g_{1,2}D^2$

code bits

$g_2(D) = g_{2,0} + g_{2,1}D + g_{2,2}D^2$

Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Encoding: $x_n(D) = g_n(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

Encoding: $\mathbf{x} = \mathbf{G}\mathbf{u}$

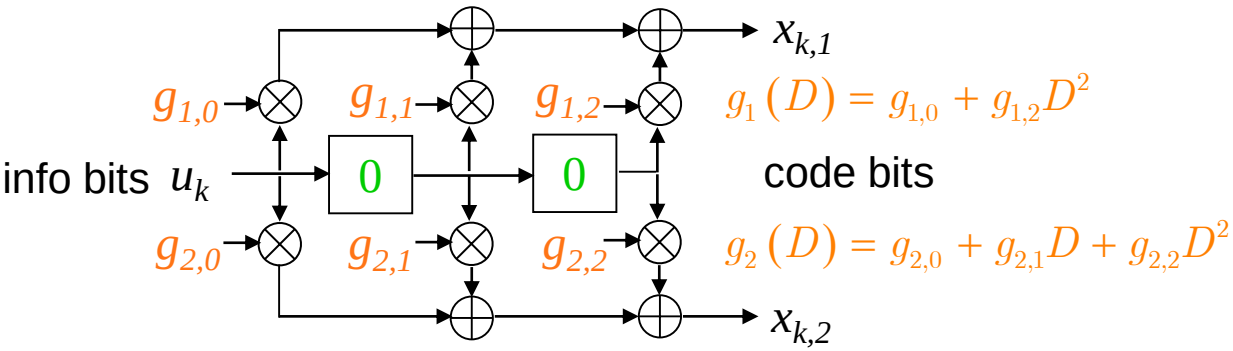
$$\begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{M,1} \\ x_{M,2} \\ \vdots \\ x_{L-1,1} \\ x_{L-1,2} \end{bmatrix} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{G}_1 & \mathbf{G}_0 & \ddots & \ddots \\ \vdots & \mathbf{G}_1 & \mathbf{G}_0 & \ddots \\ \mathbf{G}_M & \vdots & \mathbf{G}_1 & \ddots \\ \mathbf{0} & \mathbf{G}_M & \vdots & \ddots \\ \vdots & \mathbf{0} & \mathbf{G}_M & \ddots \\ \mathbf{0} & & \mathbf{0} & \ddots \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

Generator matrix $\mathbf{G}$

Convolutional Codes: Description with Polynomials and Matrices

Example:
Memory: $M=2$
Constraint length: $M+1=3$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

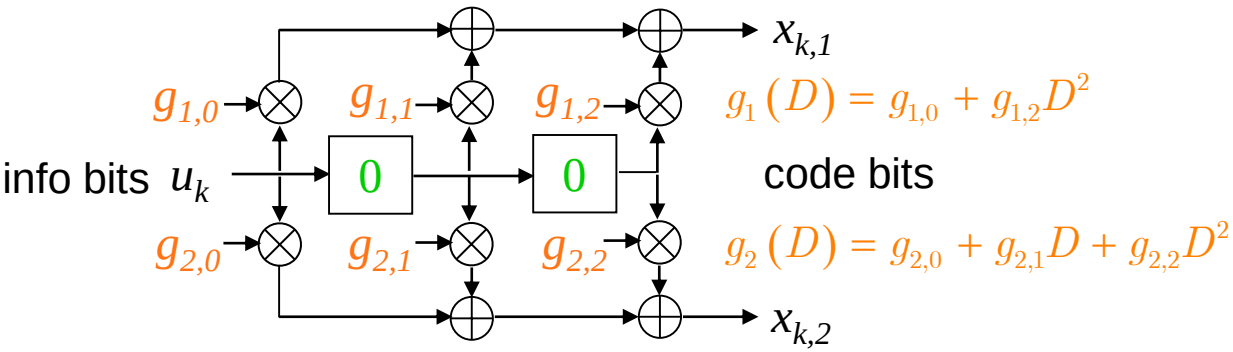
Encoding: $x_n(D) = g_n(D)u(D)$

Encoding: $\mathbf{x}(D) = \mathbf{G}(D)u(D)$

$$\begin{aligned}
x_n(D) &= g_n(D)u(D) \\
&= g_{n,0}u_0 \\
&+ \left(g_{n,1}u_0 + g_{n,0}u_1\right)D \\
&+ \left(g_{n,2}u_0 + g_{n,1}u_1 + g_{n,0}u_2\right)D^2 \\
&\vdots \\
&+ \left(g_{n,M}u_0 + g_{n,M-1}u_1 + \ldots + g_{n,0}u_M\right)D^M \\
&\vdots \\
&+ \left(g_{n,M}u_{L-1} + g_{n,L-2}u_1 + \ldots + g_{n,0}u_{M+L-1}\right)D^{M+L-1}
\end{aligned}$$

$$\begin{bmatrix} x_1(D) \\ \vdots \\ x_N(D) \end{bmatrix} = \underbrace{\begin{bmatrix} g_1(D) \\ \vdots \\ g_N(D) \end{bmatrix}}_{\text{Polynomial generator matrix } \mathbf{G}(D)} u(D)$$

Generator Polynomials for Good Convolutional Codes



Generator polynomials:

$$g_n(D) = \sum_{m=0}^{M} g_{n,m} D^m$$

Code rate: $R = \frac{K}{N} = \frac{1}{2}$:

Memory M	$g_1(D)$	$g_2(D)$	d_{free}
2	$1+D+D^2$	$1+D^2$	5
3	$1+D+D^3$	$1+D+D^2+D^3$	6
4	$1+D^3+D^4$	$1+D+D^2+D^4$	7
5	$1+D^2+D^4+D^5$	$1+D+D^2+D^3+D^5$	8
6	$1+D^2+D^3+D^5+D^6$	$1+D+D^2+D^3+D^6$	10

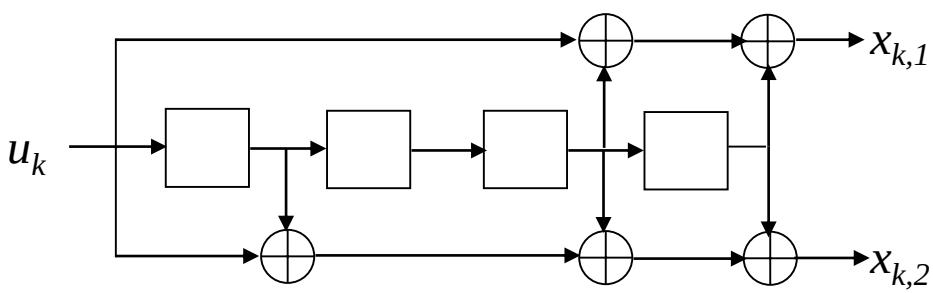
$R = \frac{1}{3}$:

Memory M	$g_1(D)$	$g_2(D)$	$g_3(D)$	d_{free}
2	$1+D+D^2$	$1+D^2$	$1+D+D^2$	8
3	$1+D+D^3$	$1+D+D^2+D^3$	$1+D^2+D^3$	10
4	$1+D^2+D^4$	$1+D+D^3+D^4$	$1+D+D^2+D^3+D^4$	12
5	$1+D^2+D^4+D^5$	$1+D+D^2+D^3+D^5$	$1+D^3+D^4+D^5$	13
6	$1+D^2+D^3+D^5+D^6$	$1+D+D^4+D^6$	$1+D+D^2+D^3+D^4+D^6$	15

Convolutional Code in GSM

GSM: Global System for Mobile Communications

$(23,33)_8$ code, code rate $R=1/2$, memory $M=4$



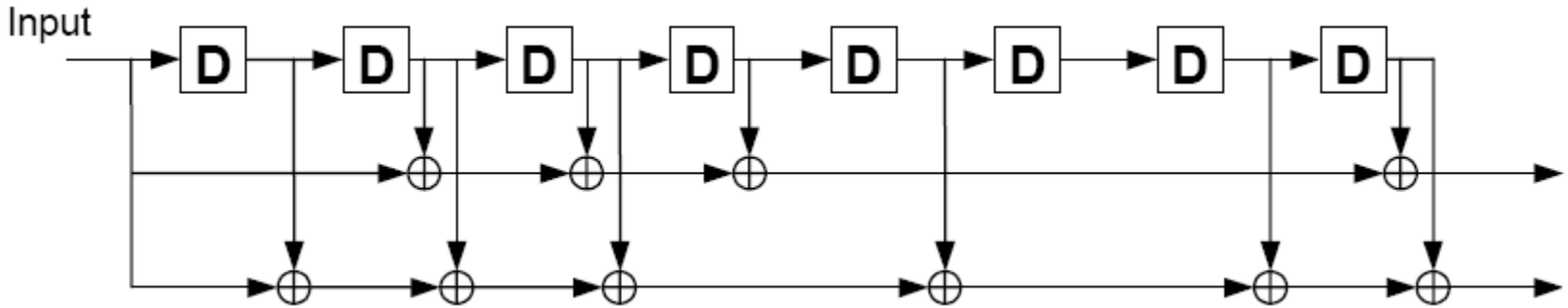
$$g_1(D) = 1 + D^3 + D^4$$

$$g_2(D) = 1 + D + D^3 + D^4$$

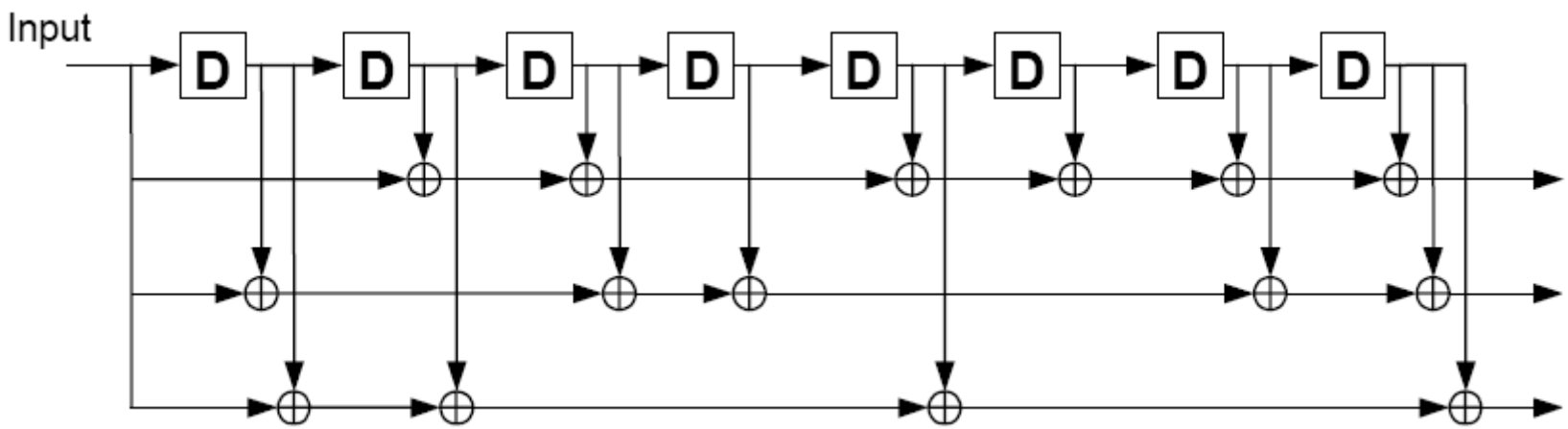
Convolutional Codes UMTS

UMTS: Universal Mobile Telecommunications System

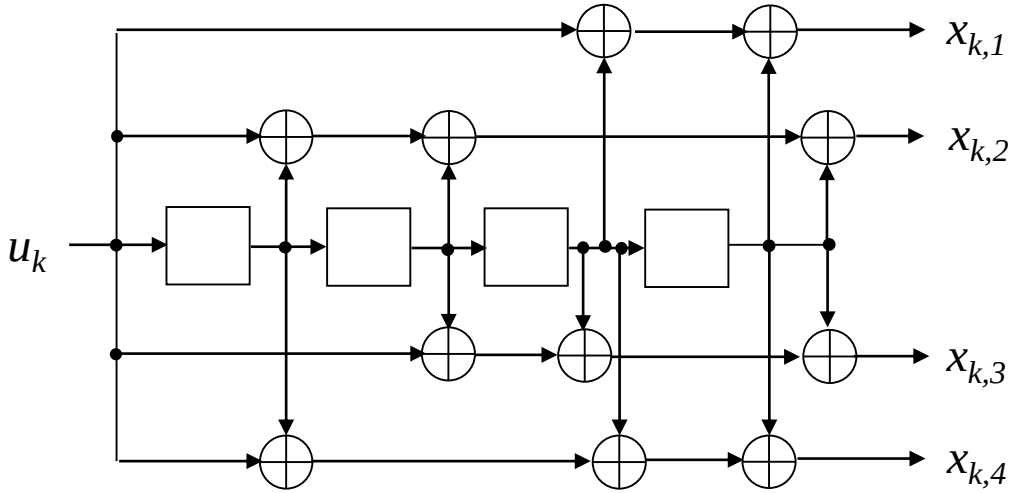
$(561,753)_8$ code, code rate $R=1/2$, memory $M=8$



$(557,663, 711)_8$ code, code rate $R=1/3$, memory $M=8$



Rate $R=1/4$ Convolutional Code

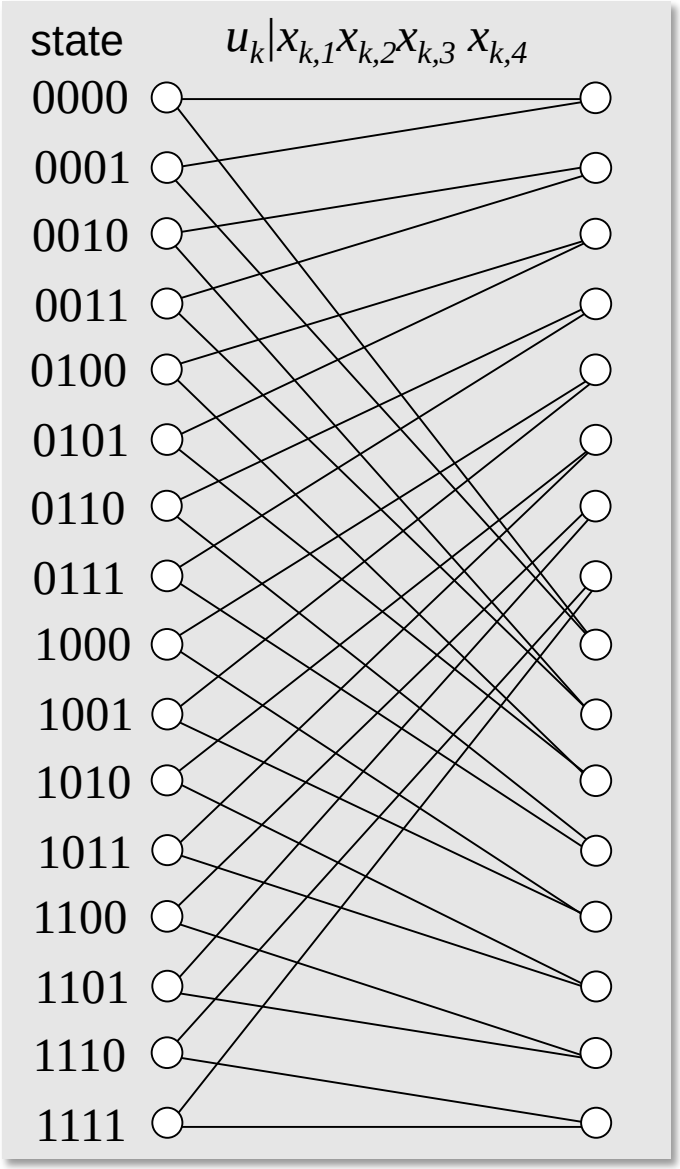


$$g_1(D) = 1 + D^3 + D^4$$

$$g_2(D) = 1 + D + D^2 + D^4$$

$$g_3(D) = 1 + D^2 + D^3 + D^4$$

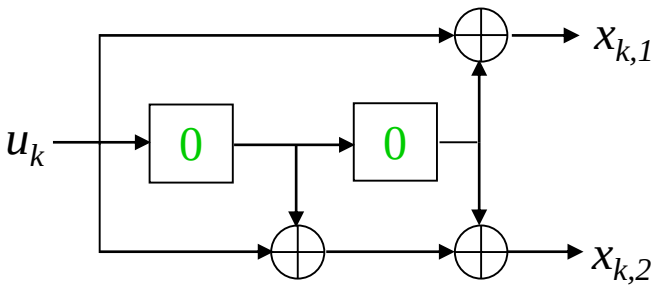
$$g_4(D) = 1 + D + D^3 + D^4$$



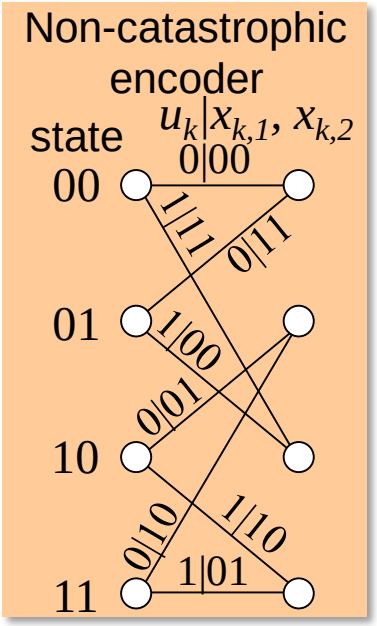
Catastrophic Convolutional Codes

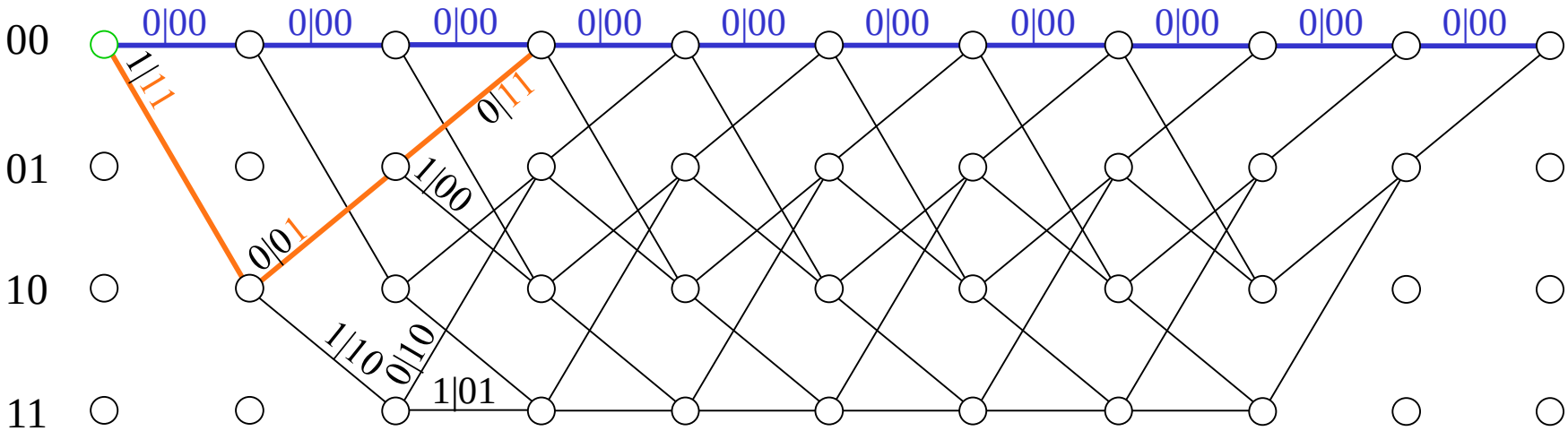
Non-catastrophic encoder

$$g_1(D) = 1 + D^2$$



$$g_2(D) = 1 + D + D^2$$

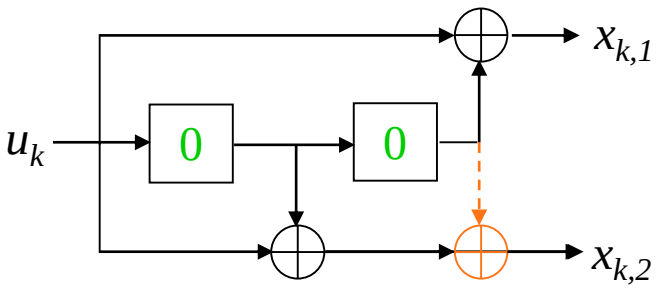




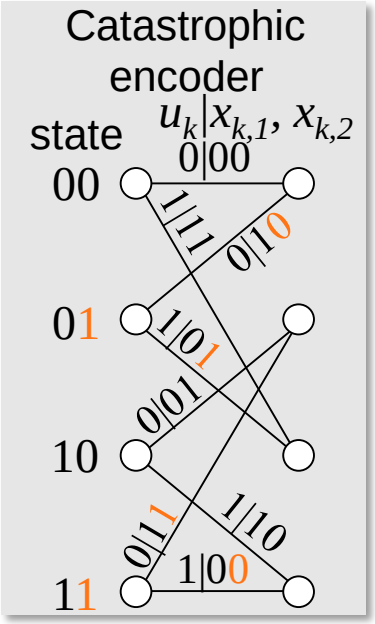
Catastrophic Convolutional Codes

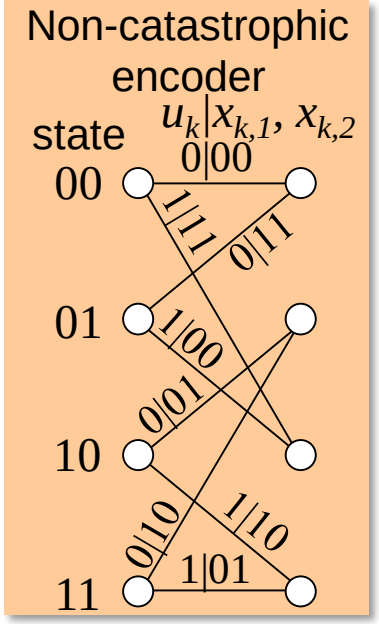
Catastrophic encoder

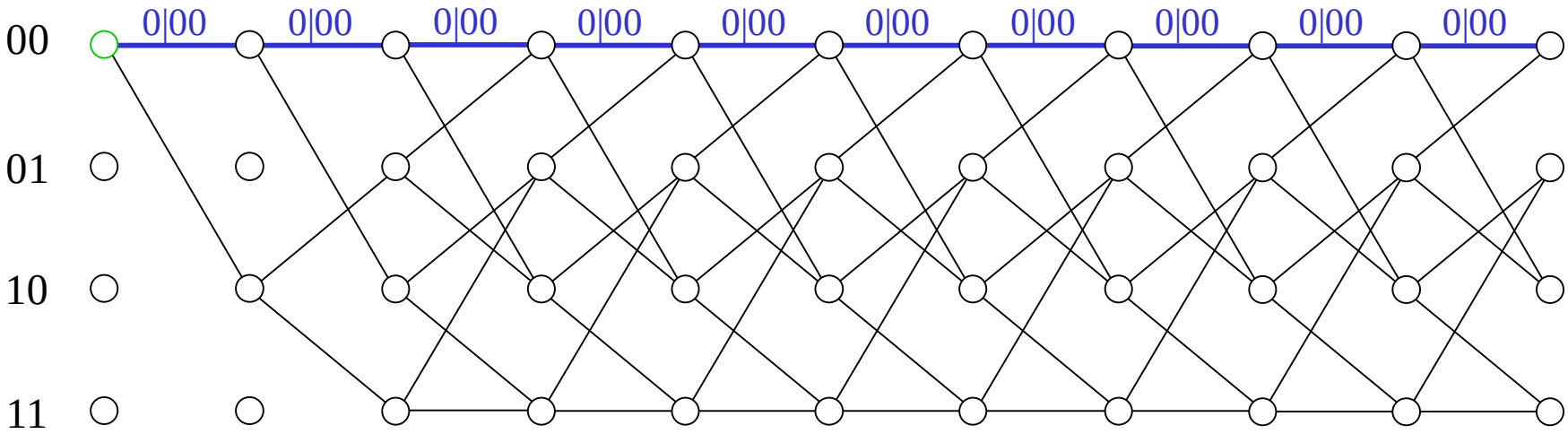
$g_1(D) = 1 + D^2$



$g_2(D) = 1 + D$



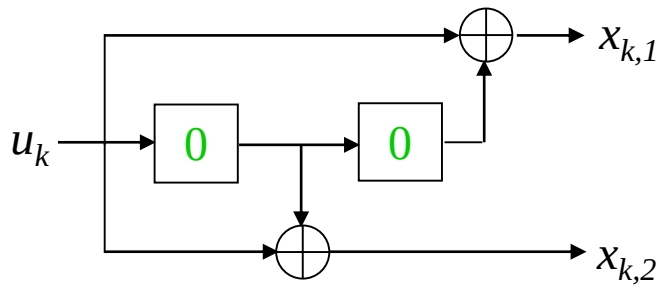




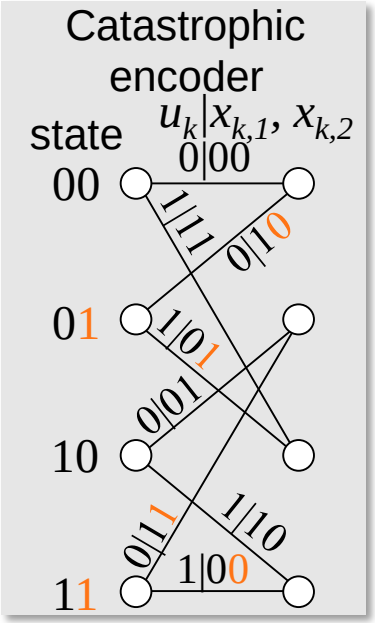
Catastrophic Convolutional Codes

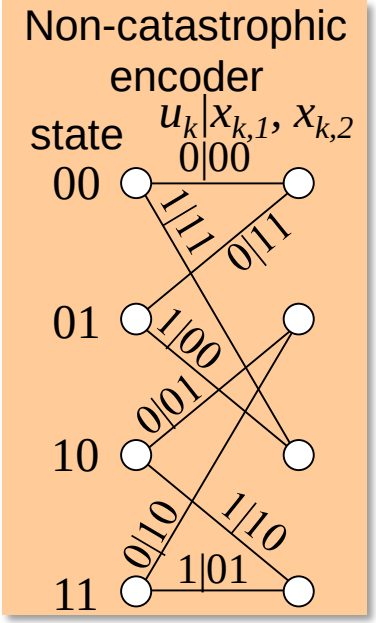
Catastrophic encoder

$g_1(D) = 1 + D^2$



$g_2(D) = 1 + D$

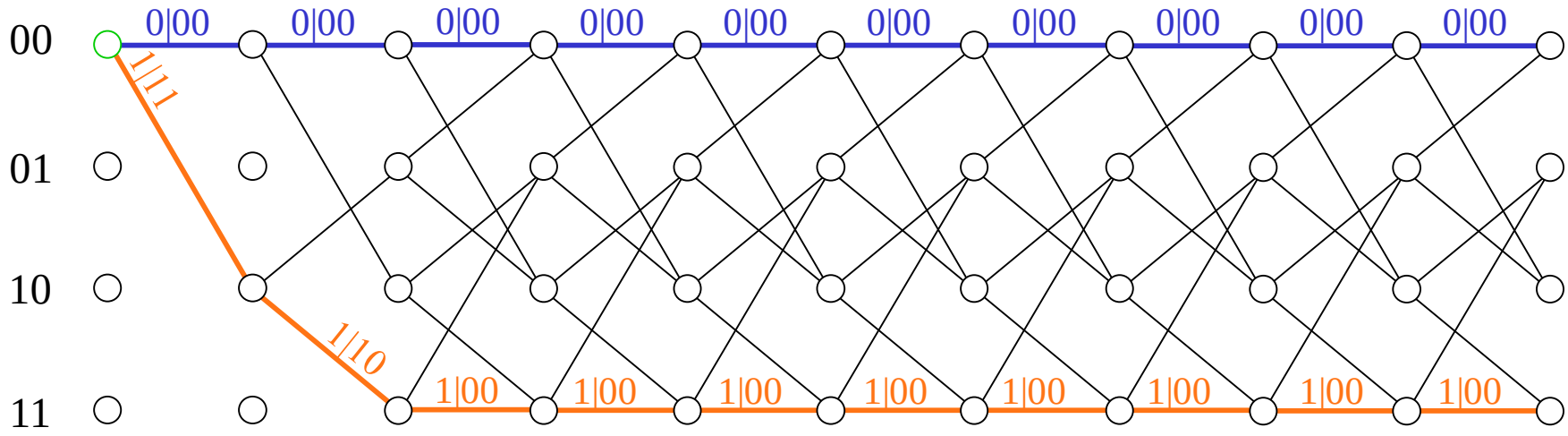




$\mathbf{u}_1$=[0,0,...,0] → $\mathbf{x}_1$=[0,0,0,0,0...,0]
$\mathbf{u}_2$=[1,1,...,1] → $\mathbf{x}_2$=[1,1,1,0,0,...,0]

d_H=($\mathbf{x}_1$, $\mathbf{x}_2$)=3

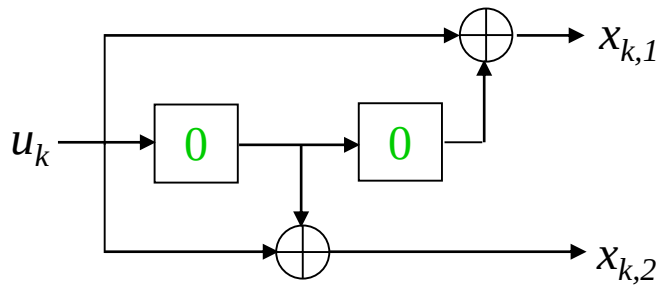
3 transmission errors cause infinite error sequence in decoded data.



Catastrophic Convolutional Codes

Catastrophic encoder

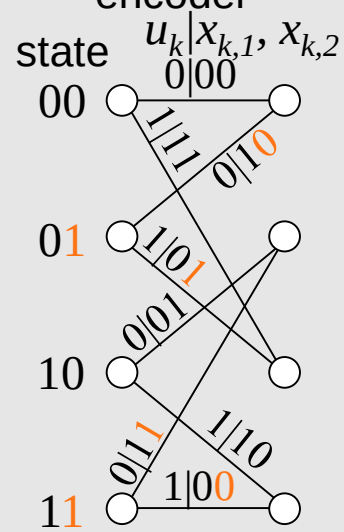
$g_1(D) = 1 + D^2$



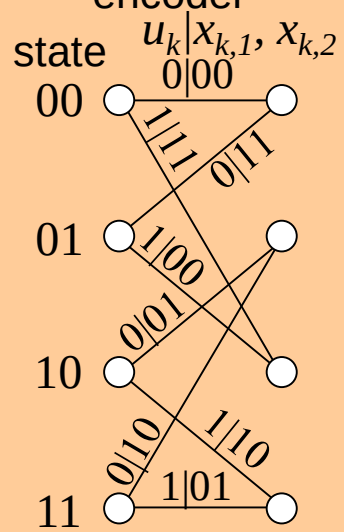
$g_2(D) = 1 + D$

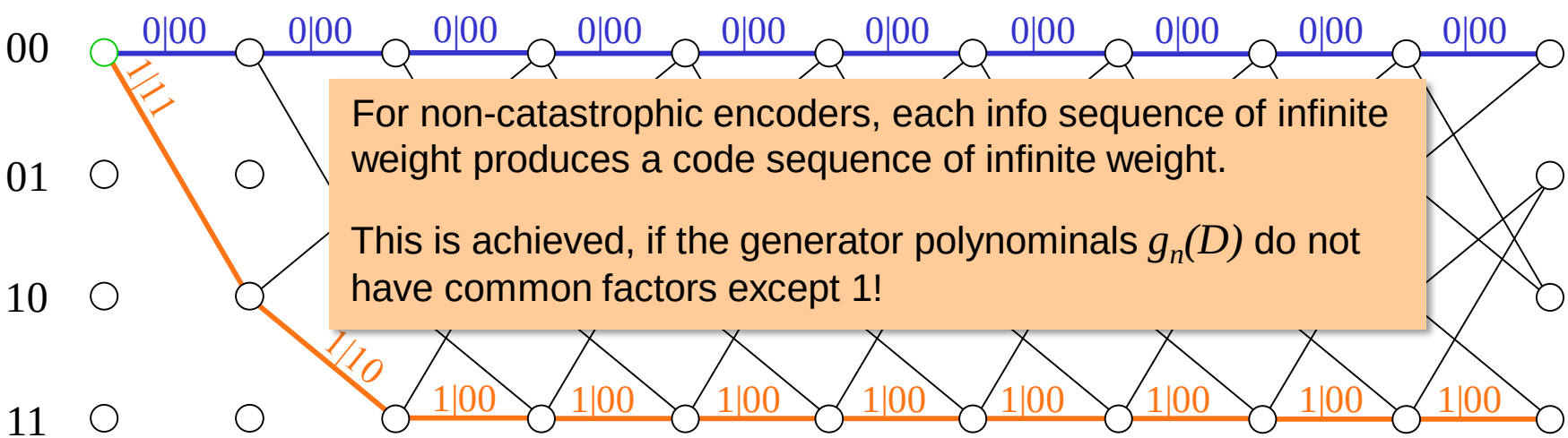
$$g_1(D) : g_2(D) = \left(D^2 + 1\right) : (D + 1) = D + 1$$

Catastrophic encoder



Non-catastrophic encoder





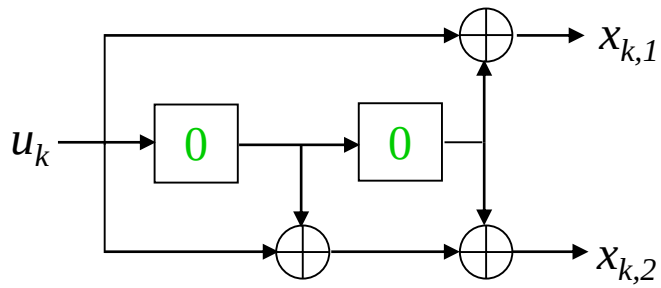
For non-catastrophic encoders, each info sequence of infinite weight produces a code sequence of infinite weight.

This is achieved, if the generator polynominals $g_n(D)$ do not have common factors except 1!

Catastrophic Convolutional Codes

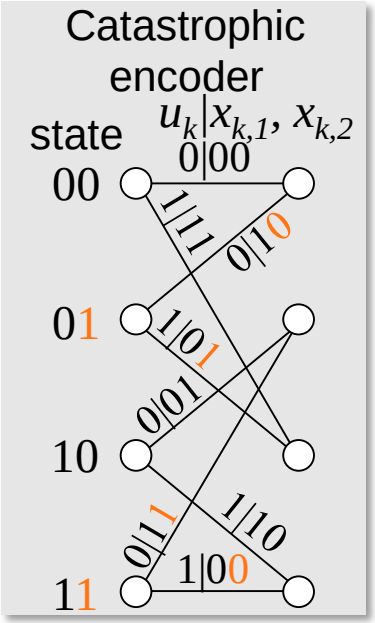
Non-catastrophic encoder

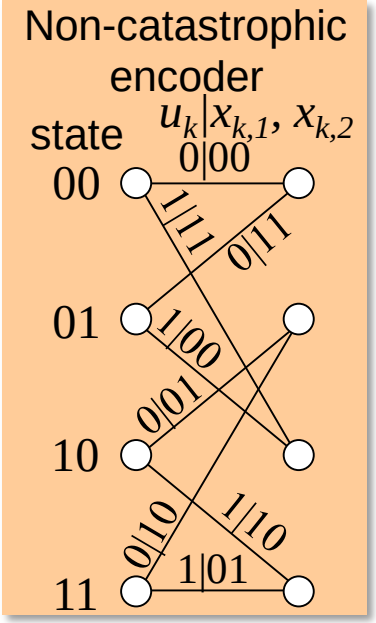
$$g_1(D) = 1 + D^2$$

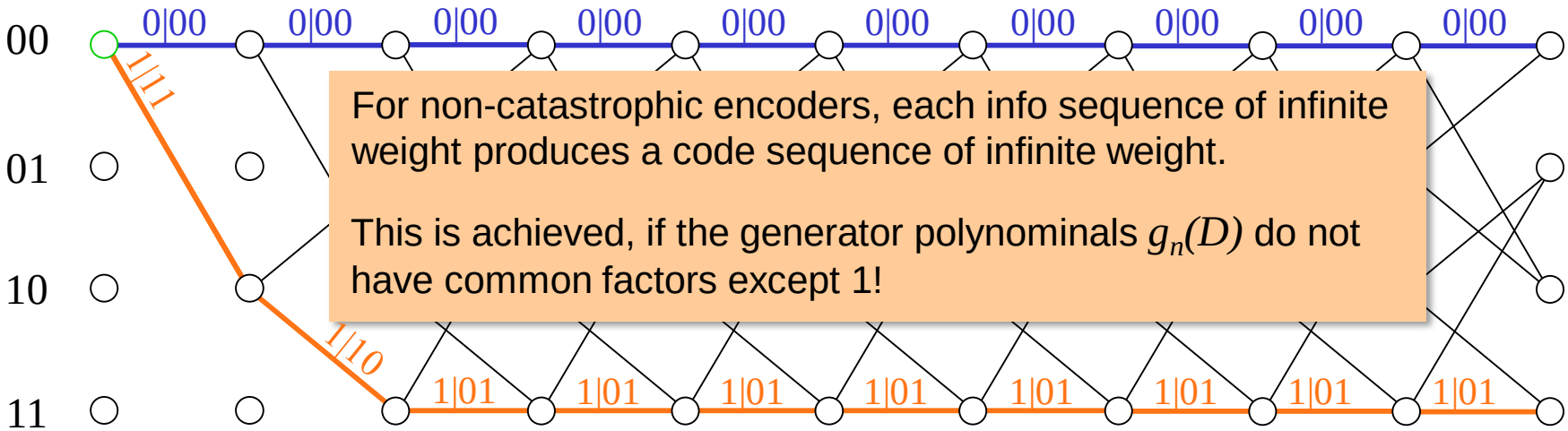


$$g_2(D) = 1 + D + D^2$$

$$g_2(D) : g_1(D) = \left(D^2 + D + 1\right) : \left(D^2 + 1\right) = 1$$
$$\underline{-\left(D^2 + 1\right)}$$
$$D = r(D)$$



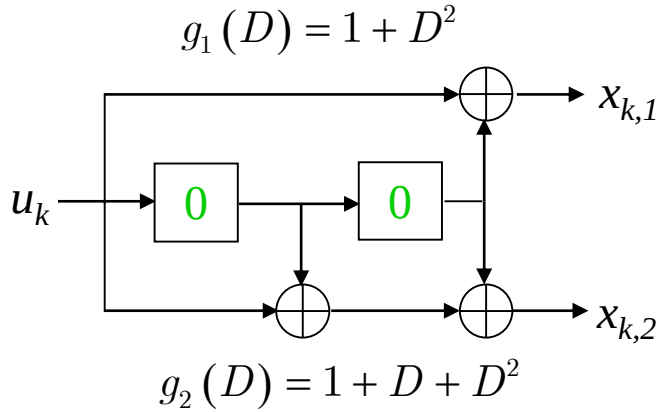




For non-catastrophic encoders, each info sequence of infinite weight produces a code sequence of infinite weight.

This is achieved, if the generator polynominals $g_n(D)$ do not have common factors except 1!

Convolutional Codes: Free Distance

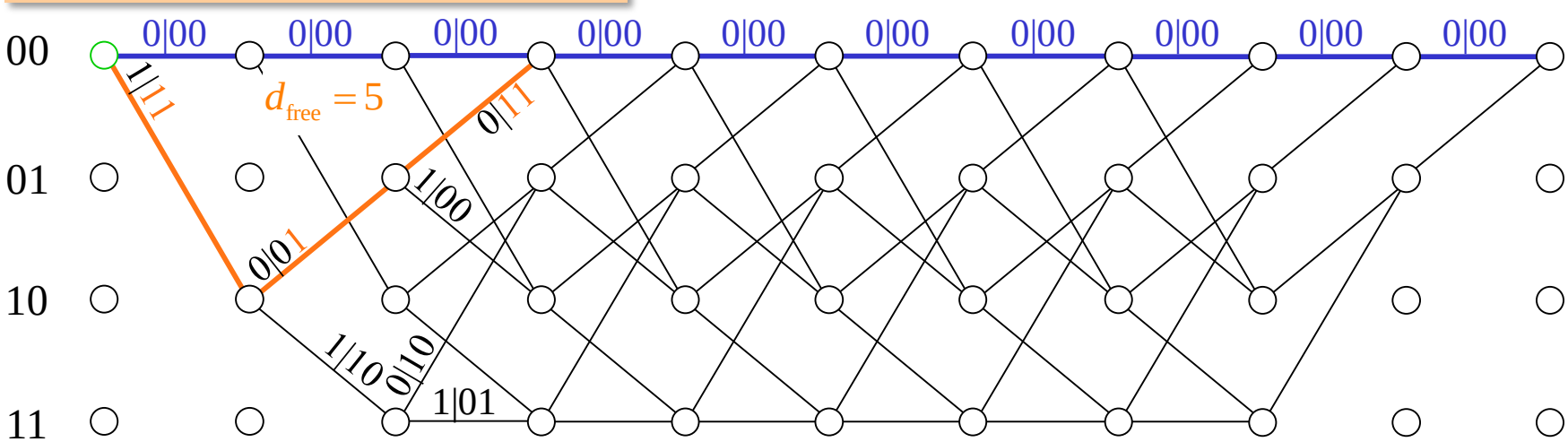


$$d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} = 2+3=5$$

$$\le N(M+1) = 2(2+1) = 6$$

$$u(D)=1 \Rightarrow x_n(D) = u(D)g_n(D) = g_n(D)$$

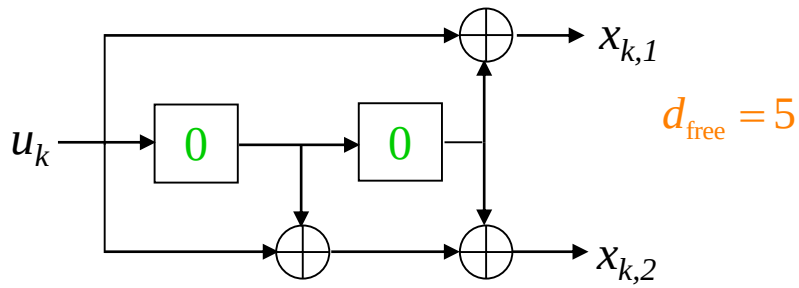
$$\Rightarrow d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} \le N(M+1)$$



Convolutional Codes: Free Distance

Non-systematic encoder

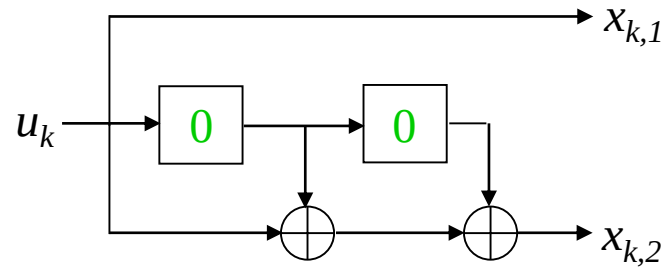
$g_1(D) = 1 + D^2$



$g_2(D) = 1 + D + D^2$

Systematic encoder

$g_1(D) = 1$



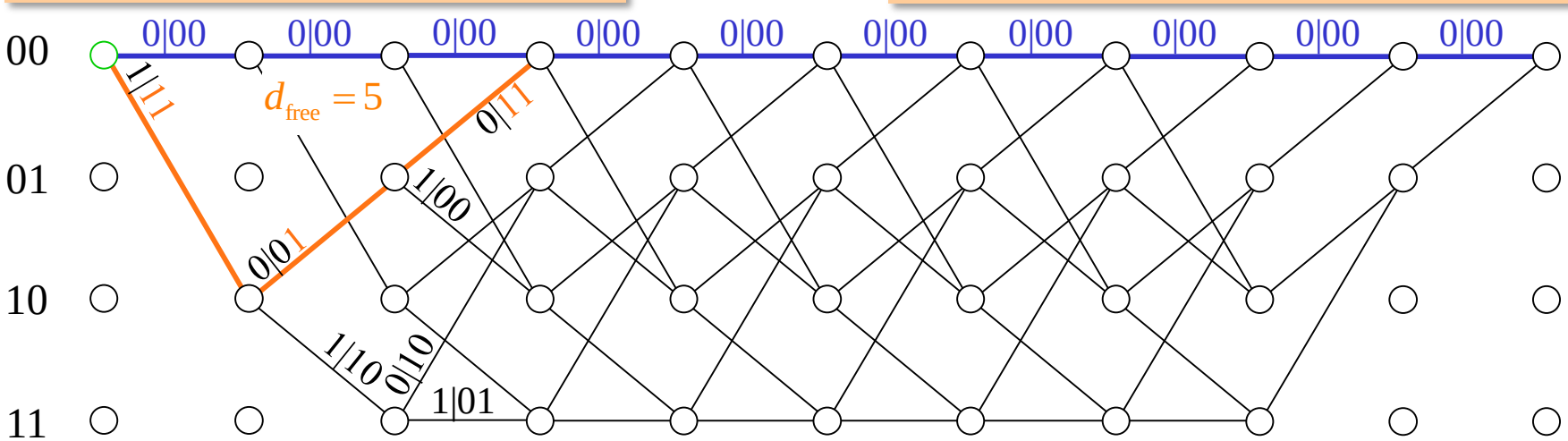
$g_2(D) = 1 + D + D^2$

$$u(D) = 1 \Rightarrow x_n(D) = u(D)\, g_n(D) = g_n(D)$$

$$\Rightarrow d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} \le N(M+1)$$

$$d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} = 1 + 3 = 4 \le 1 + (M+1) = 4$$

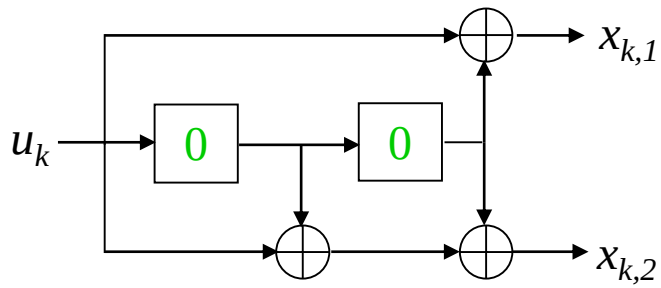
Systematic FIR encoders do not generate optimum convolutional codes.



Convolutional Codes: Free Distance

Non-systematic encoder

$g_1(D) = 1 + D^2$

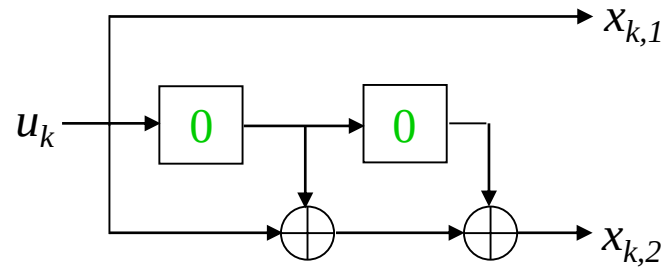


$d_{\text{free}} = 5$

$g_2(D) = 1 + D + D^2$

Systematic encoder

$g_1(D) = 1$

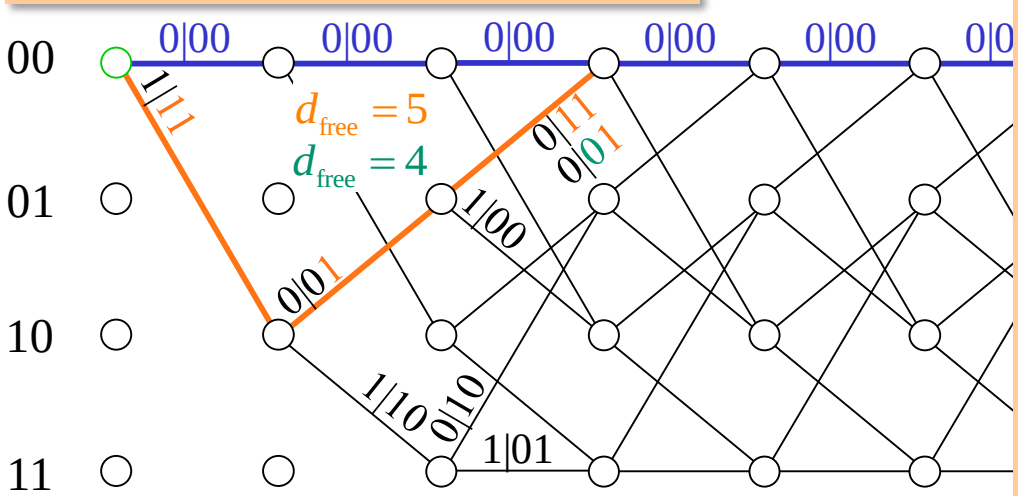


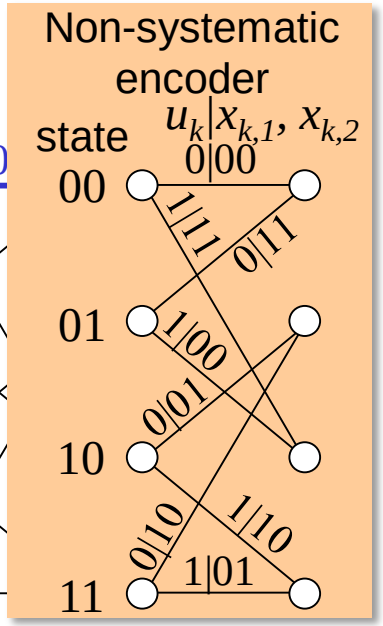
$g_2(D) = 1 + D + D^2$

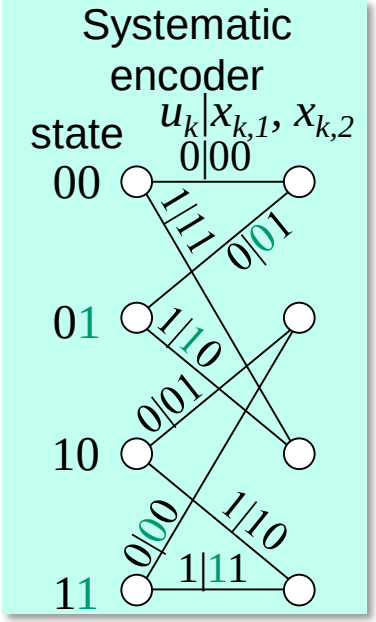
$$u(D) = 1 \Rightarrow x_n(D) = u(D)\,g_n(D) = g_n(D)$$

$$d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} = 1 + 3 = 4 \le 1 + (M+1) = 4$$

$$\Rightarrow d_{\text{free}} \le \sum_{n=1}^{N} w_H\{g_n(D)\} \le N(M+1)$$



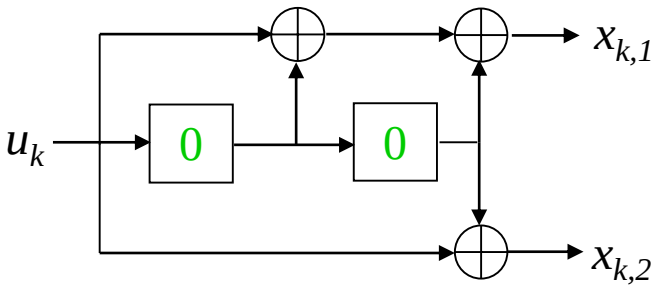




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder $(7,5)_8$

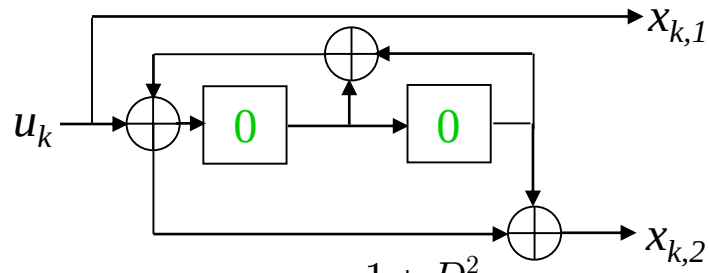
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

Systematic encoder $(1,5/7)_8$

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

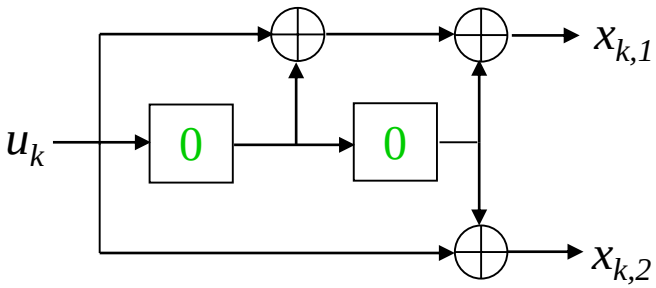
$$\begin{bmatrix} x_1(D) \\ \vdots \\ x_N(D) \end{bmatrix} = \underbrace{\begin{bmatrix} g_1(D) \\ \vdots \\ g_N(D) \end{bmatrix}}_{\mathbf{G}(D)} u(D) \quad \Rightarrow \quad \begin{bmatrix} x_1(D) \\ \vdots \\ x_N(D) \end{bmatrix} = \frac{1}{g_1(D)} G(D) = \underbrace{\begin{bmatrix} 1 \\ \frac{g_2(D)}{g_1(D)} \\ \vdots \\ \frac{g_N(D)}{g_1(D)} \end{bmatrix}}_{\mathbf{G}_{sys}(D)} u(D)$$

Any convolutional code specified by the polynomial generator matrix $\mathbf{G}(D)$ has an equivalent systematic encoder specified by the polynomial generator matrix $\mathbf{G}_{\text{sys}}(D)$.

Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

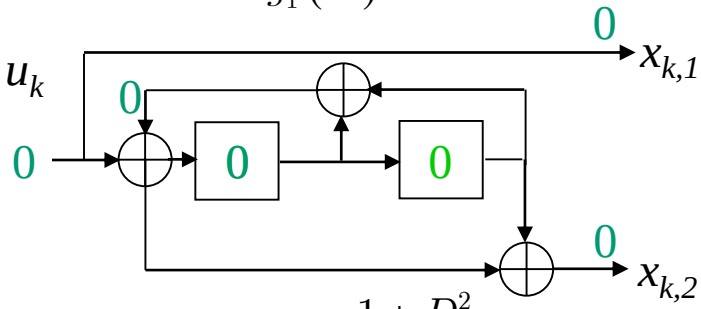
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

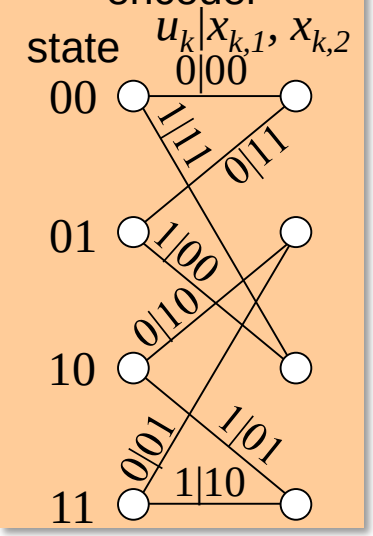
Systematic encoder

$$g_1(D) = 1$$

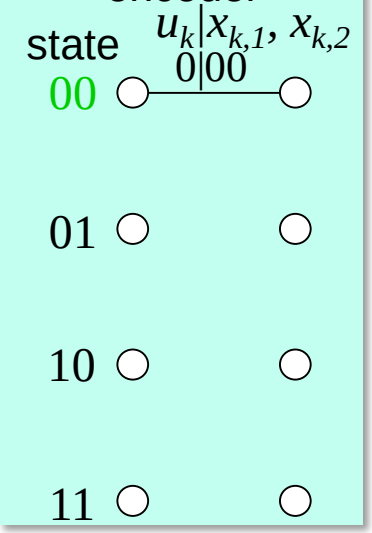


$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Non-systematic encoder



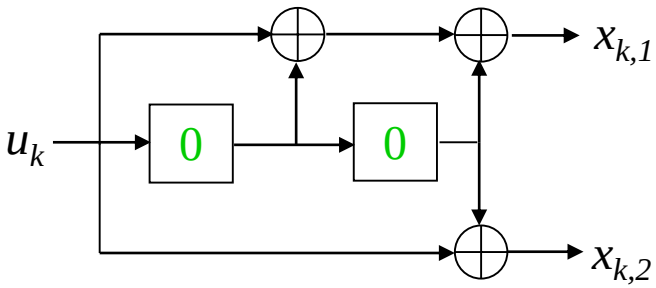
Systematic encoder



Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

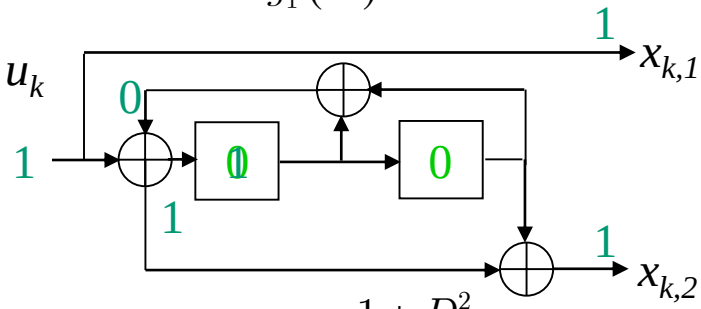
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

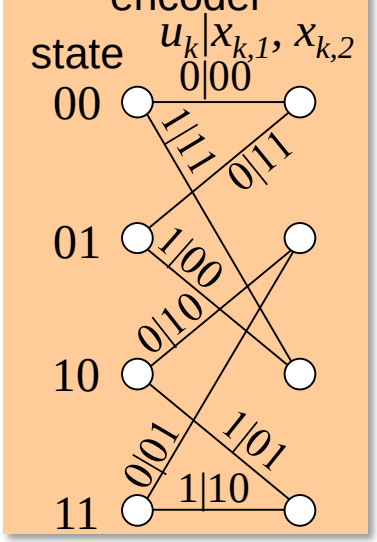
Systematic encoder

$$g_1(D) = 1$$

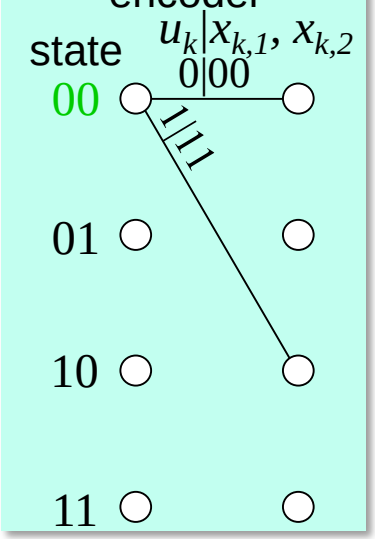


$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Non-systematic encoder



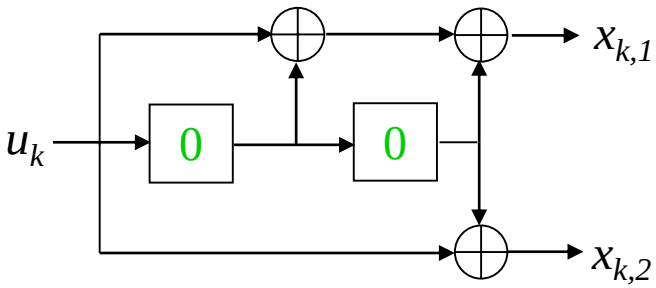
Systematic encoder



Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

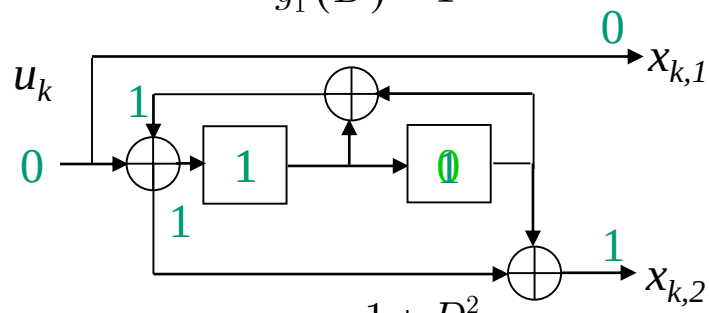
$$g_1(D) = 1 + D + D^2$$



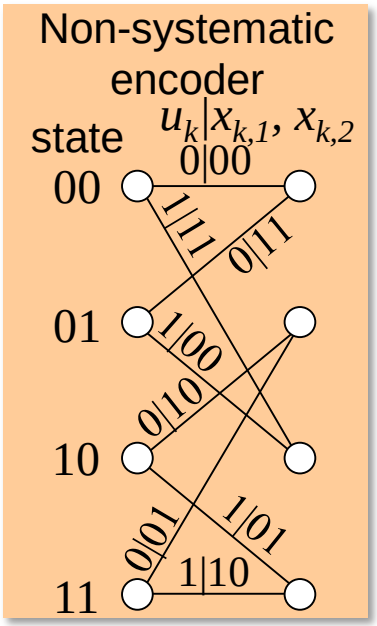
$$g_2(D) = 1 + D^2$$

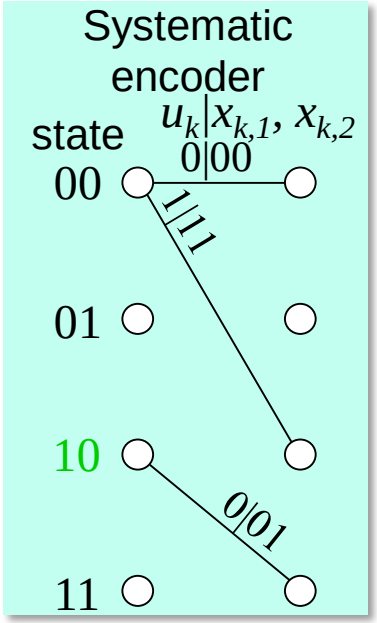
Systematic encoder

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

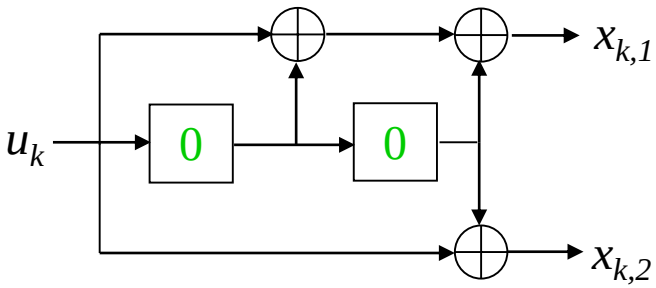




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

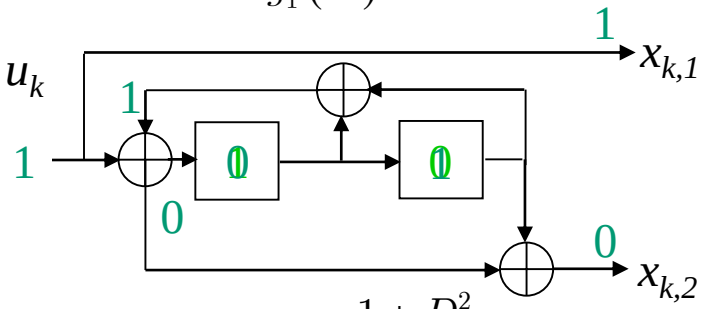
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

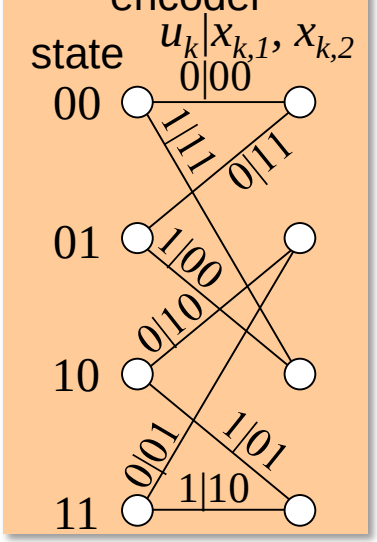
Systematic encoder

$$g_1(D) = 1$$

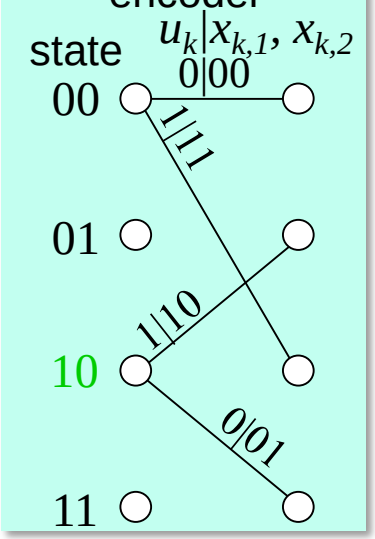


$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Non-systematic encoder



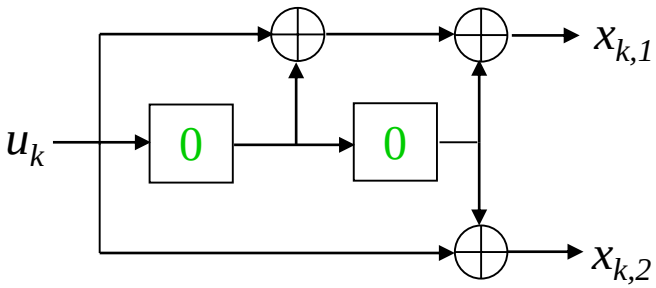
Systematic encoder



Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

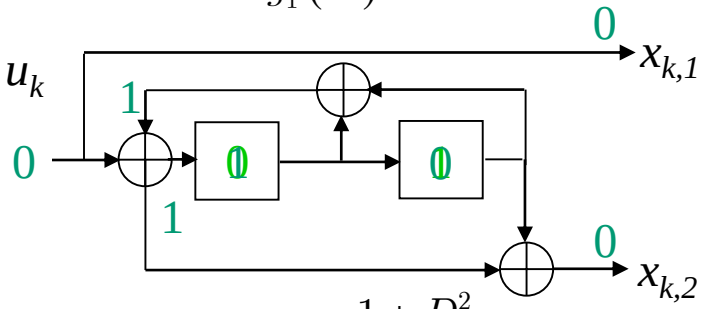
$$g_1(D) = 1 + D + D^2$$



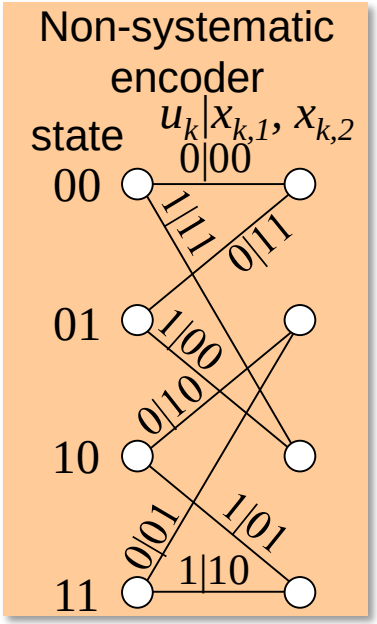
$$g_2(D) = 1 + D^2$$

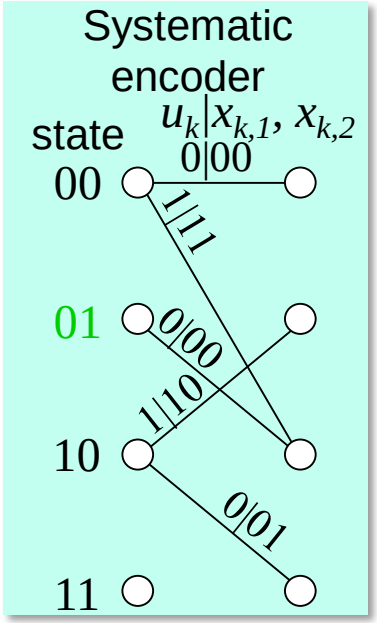
Systematic encoder

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

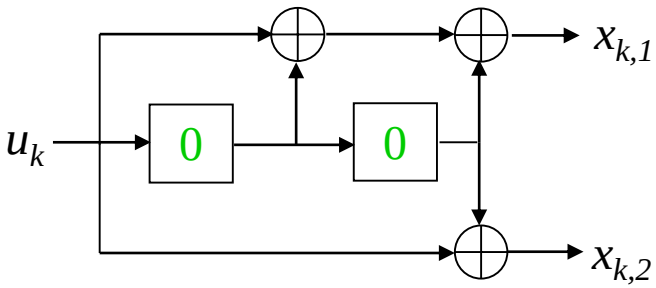




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

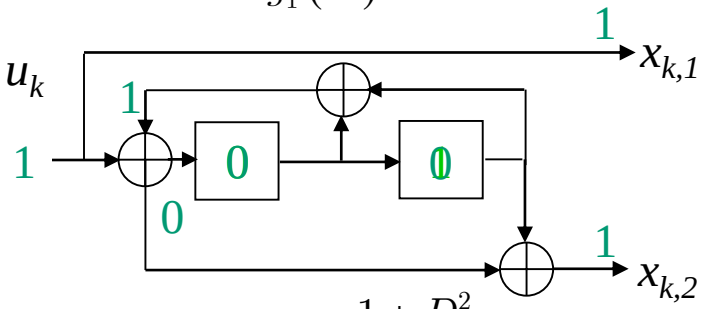
$$g_1(D) = 1 + D + D^2$$



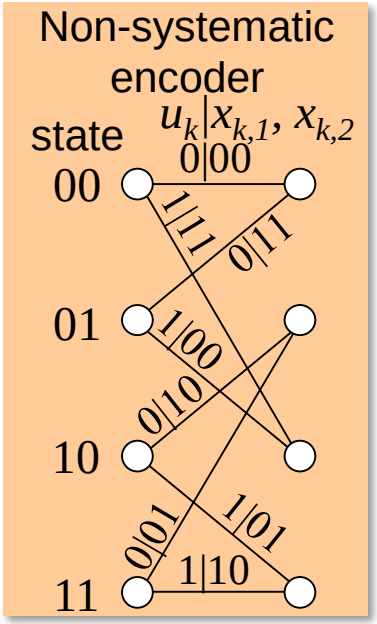
$$g_2(D) = 1 + D^2$$

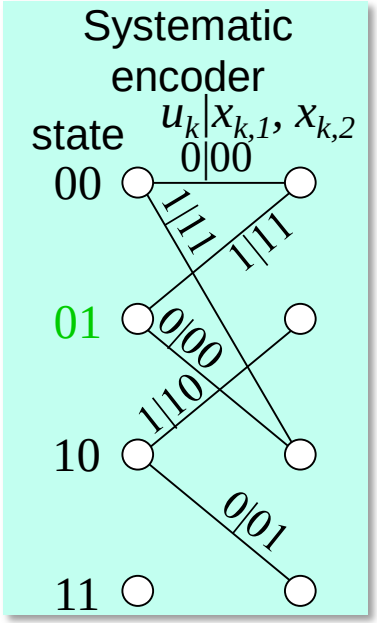
Systematic encoder

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

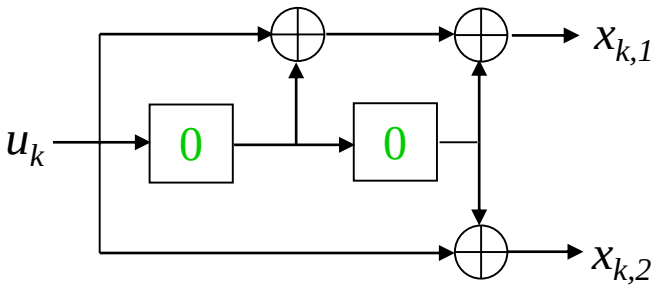




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

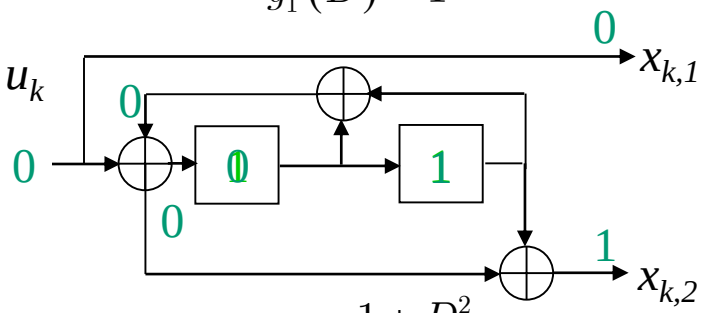
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

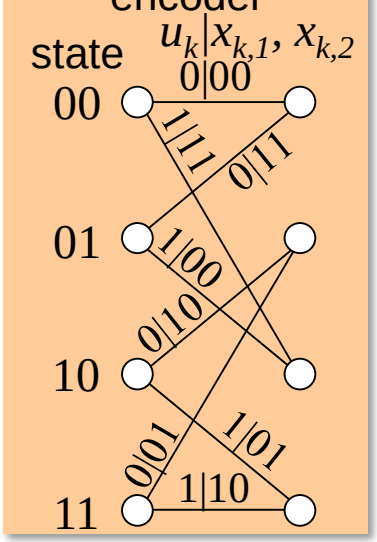
Systematic encoder

$$g_1(D) = 1$$

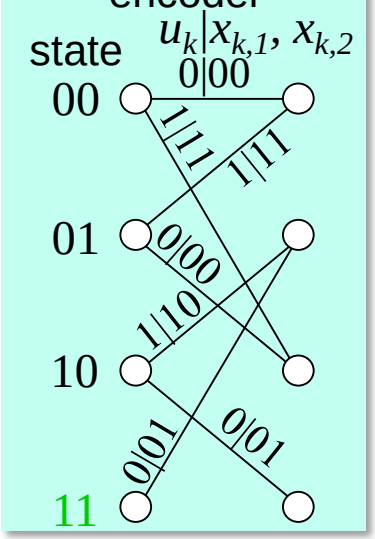


$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Non-systematic encoder



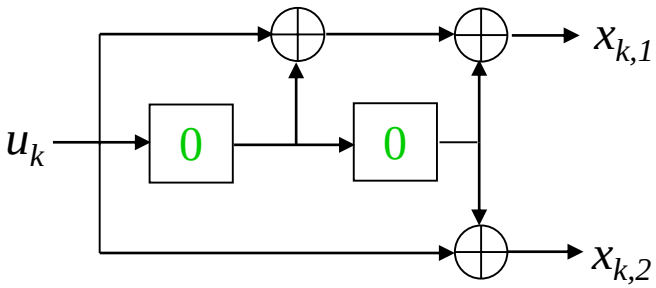
Systematic encoder



Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

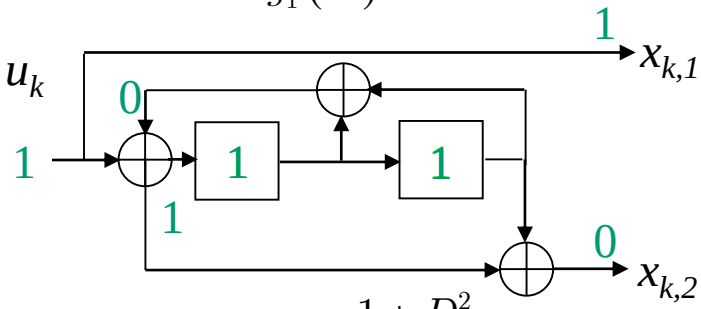
$$g_1(D) = 1 + D + D^2$$



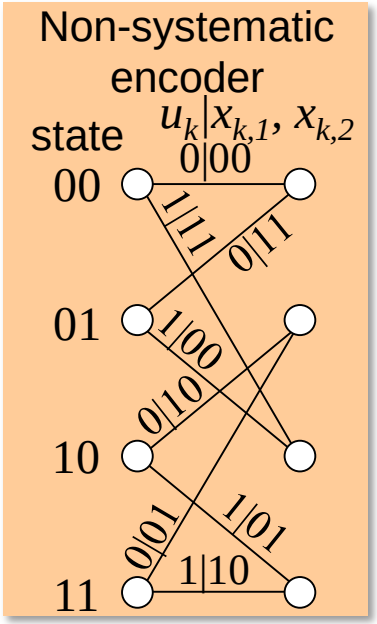
$$g_2(D) = 1 + D^2$$

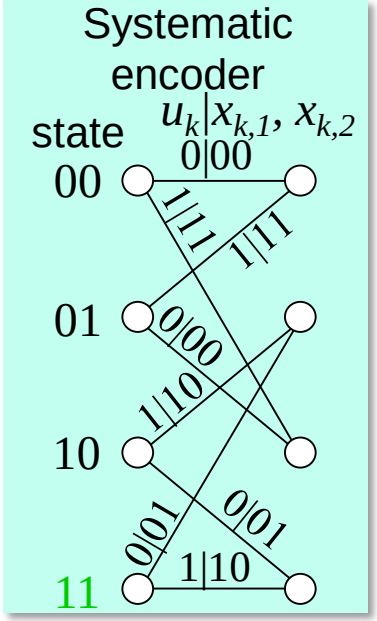
Systematic encoder

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

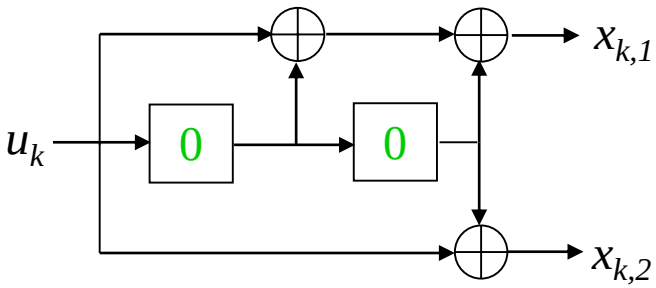




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

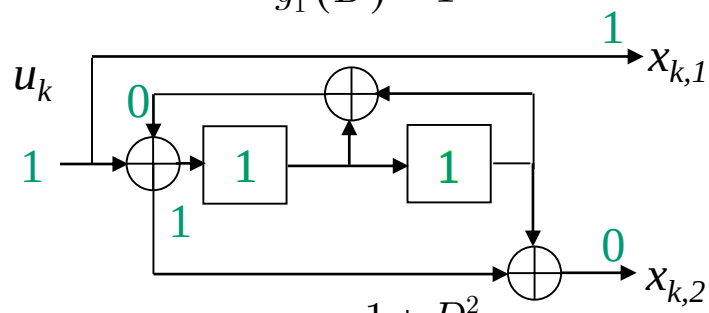
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

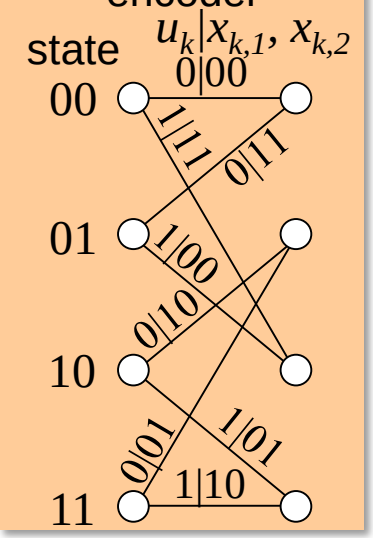
Systematic encoder

$$g_1(D) = 1$$



$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Non-systematic encoder

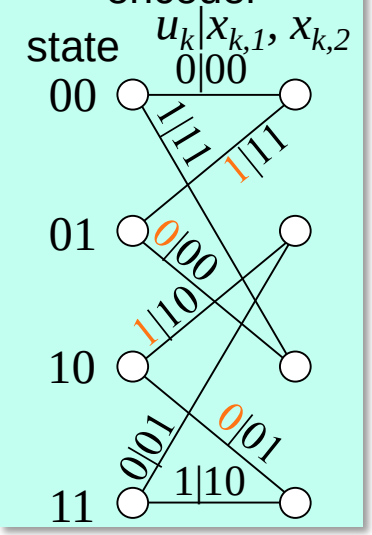


Same code,
but different encoder.

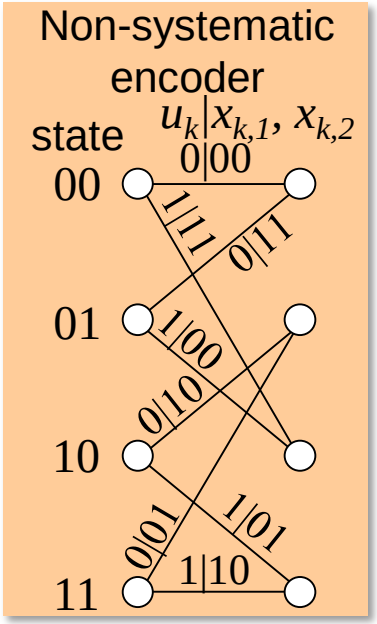
Transitions which merge into the same state are labelled with the **same** info bit.

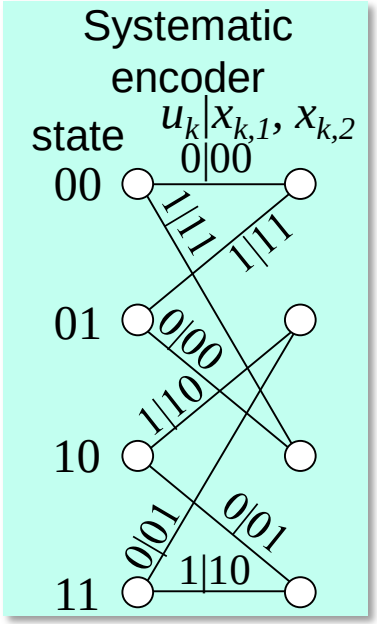
Transitions which merge into the same state are labelled with **different** info bits.

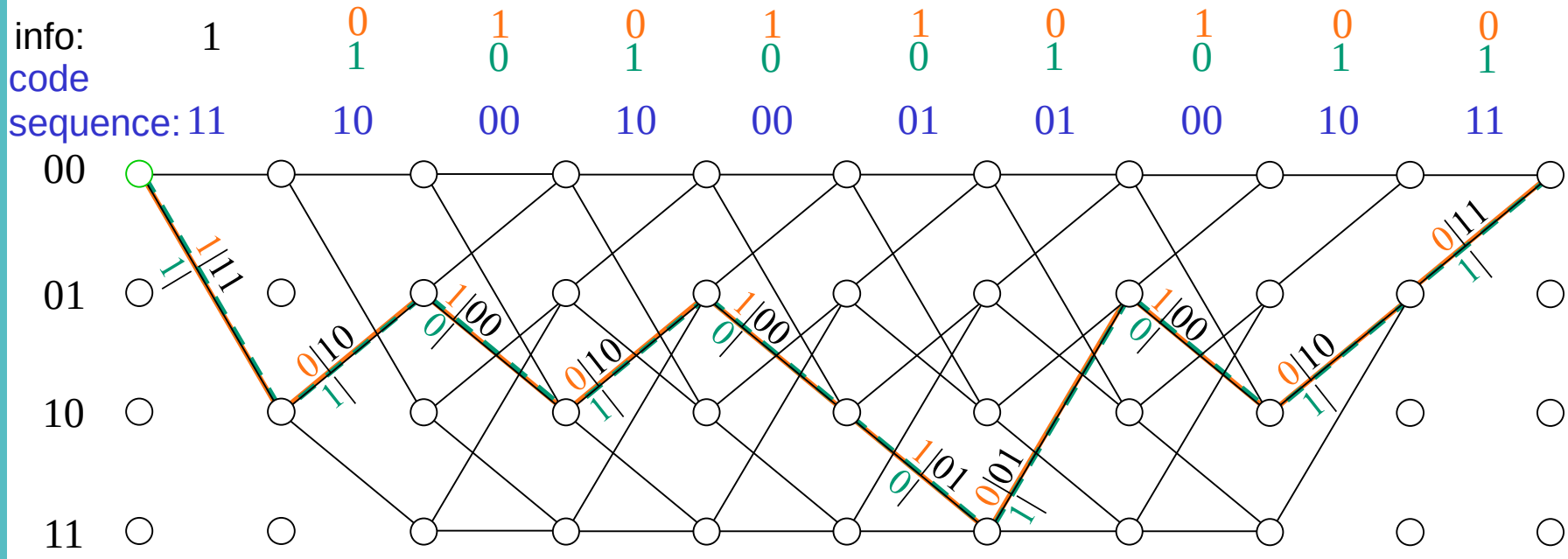
Systematic encoder



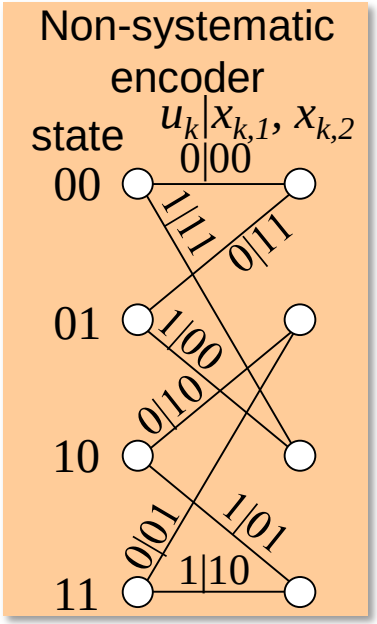
Recursive Systematic Convolutional (RSC) Codes

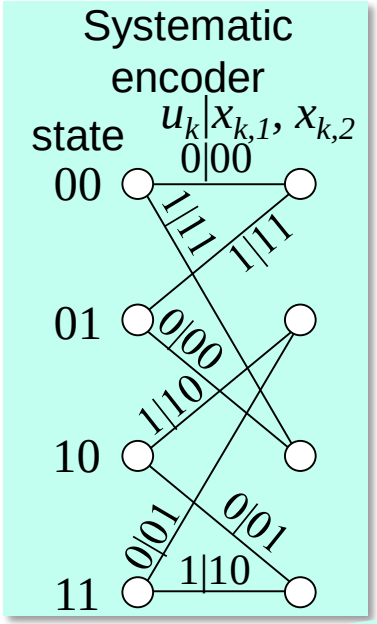


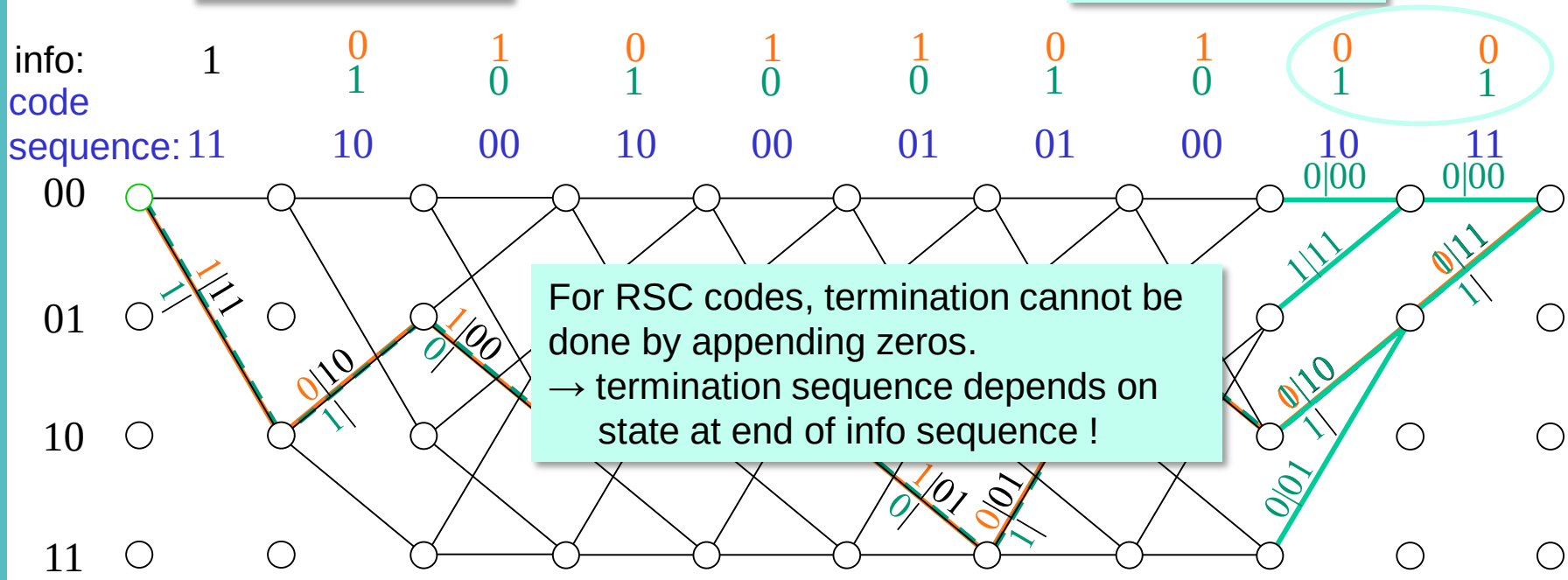




Recursive Systematic Convolutional (RSC) Codes

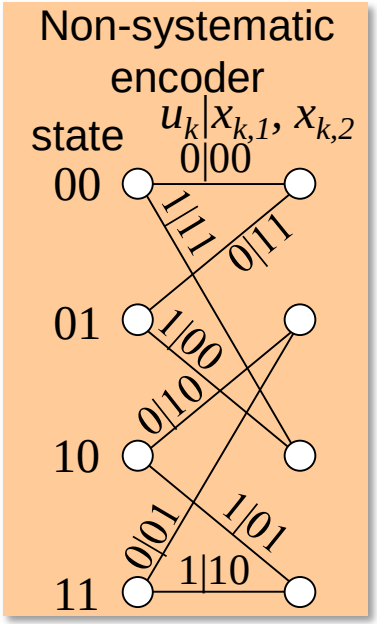


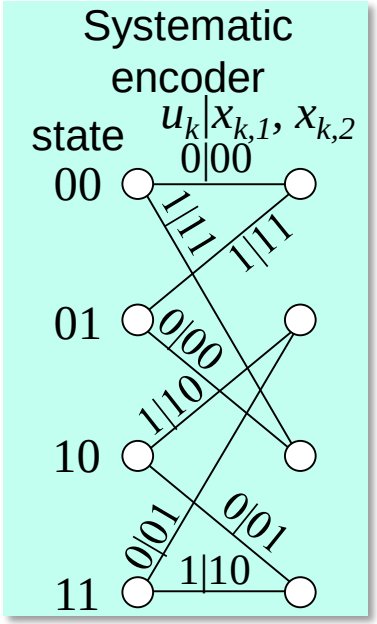


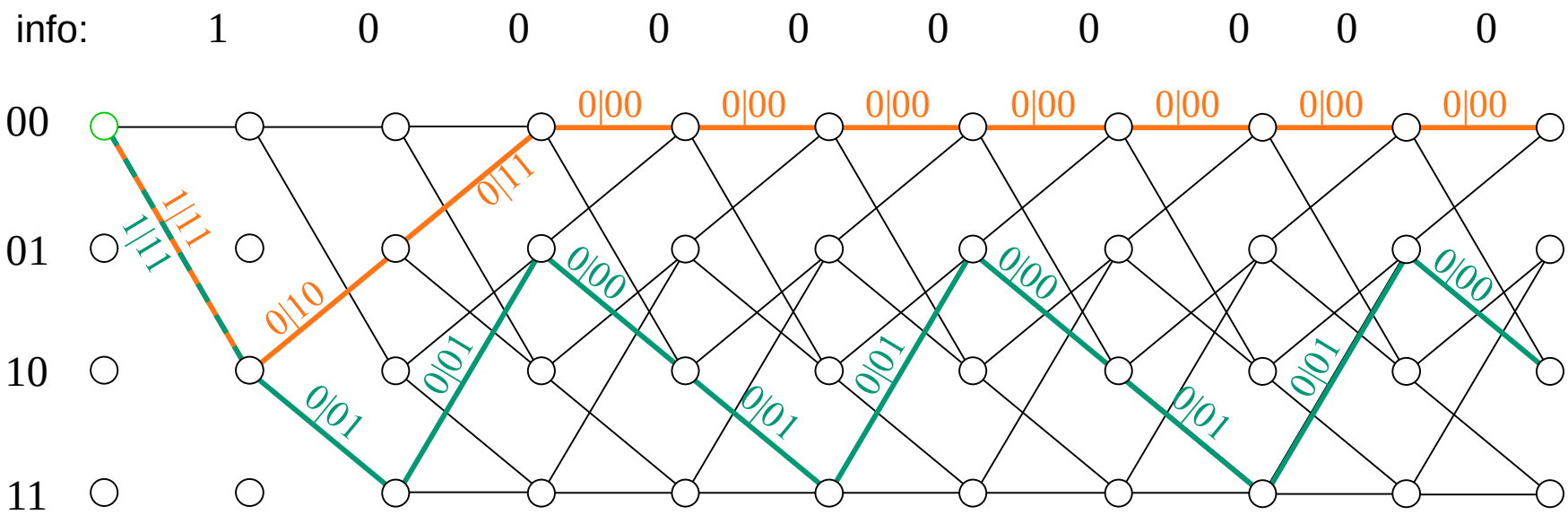


For RSC codes, termination cannot be done by appending zeros.
→ termination sequence depends on state at end of info sequence !

Recursive Systematic Convolutional (RSC) Codes



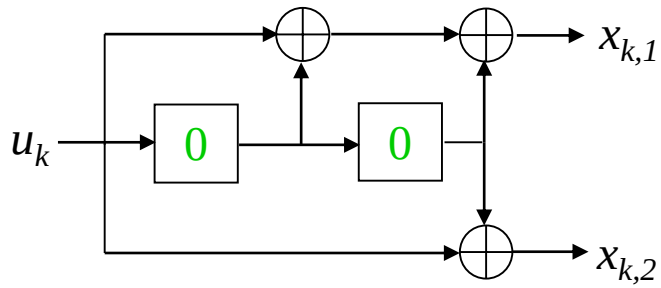




Recursive Systematic Convolutional (RSC) Codes

Non-systematic encoder

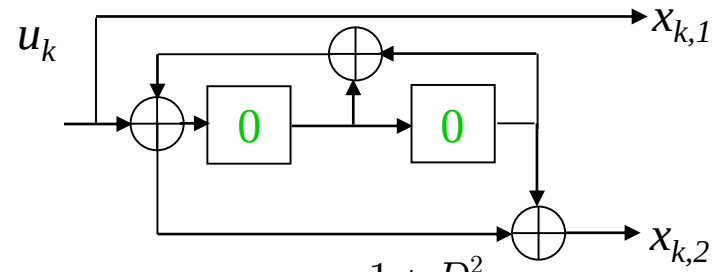
$$g_1(D) = 1 + D + D^2$$



$$g_2(D) = 1 + D^2$$

Systematic encoder

$$g_1(D) = 1$$



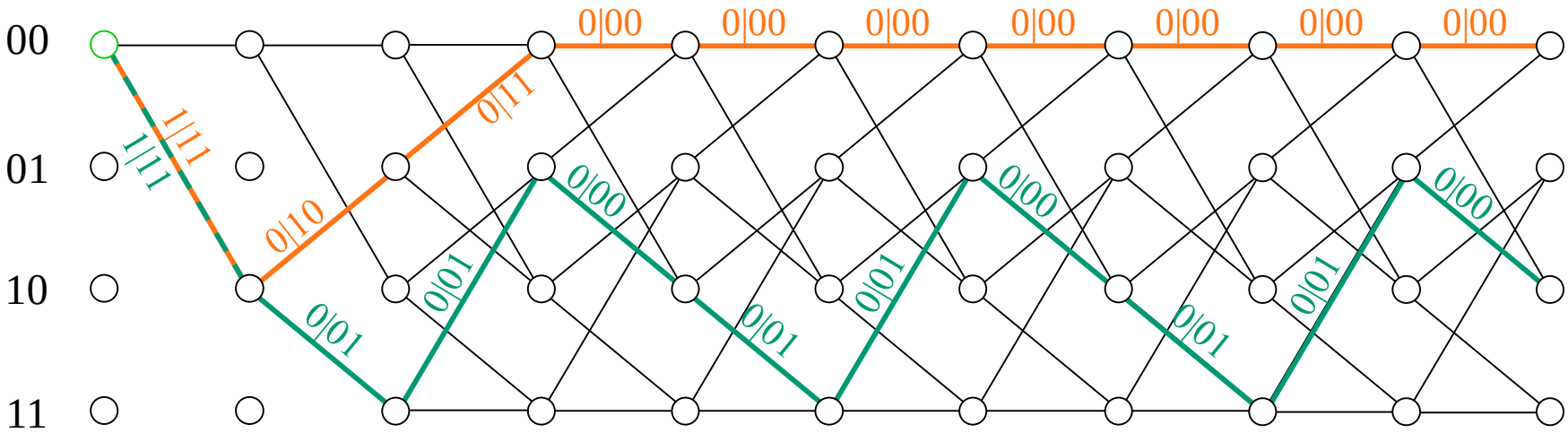
$$g_2(D) = \frac{1 + D^2}{1 + D + D^2}$$

Finite impulse response (FIR) encoder

An info sequence with low Hamming weight results in a code sequence with low Hamming weight.

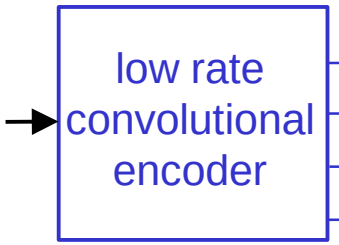
Infinite impulse response (IIR) encoder

An info sequence with low Hamming weight results in a code sequence with high Hamming weight.



Punctured Convolutional Codes

mother code



puncturing

$x_{k,1}$	+1	-1	+1	-1	+1	+1	+1	-1
$x_{k,2}$	+1	-1	+1	-1	+1	-1	+1	-1
$x_{k,3}$	+1	-1	-1	+1	+1	-1	+1	-1
$x_{k,4}$	-1	+1	+1	+1	-1	+1	-1	+1

$$R = \frac{K}{N} = \frac{1}{4}$$

Punctured Convolutional Codes

$R=8/30=4/15$

mother code | puncturing

low rate convolutional encoder

$x_{k,1}$	+1 -1 +1 -1 +1 +1 +1 -1
$x_{k,2}$	+1 -1 +1 -1 +1 -1 +1 -1
$x_{k,3}$	+1 -1 -1 +1 +1 -1 +1 -1
$x_{k,4}$	-1 +1 +1 +1 -1 +1 -1 +1

$$R=\frac{K}{N}=\frac{1}{4}$$

channel

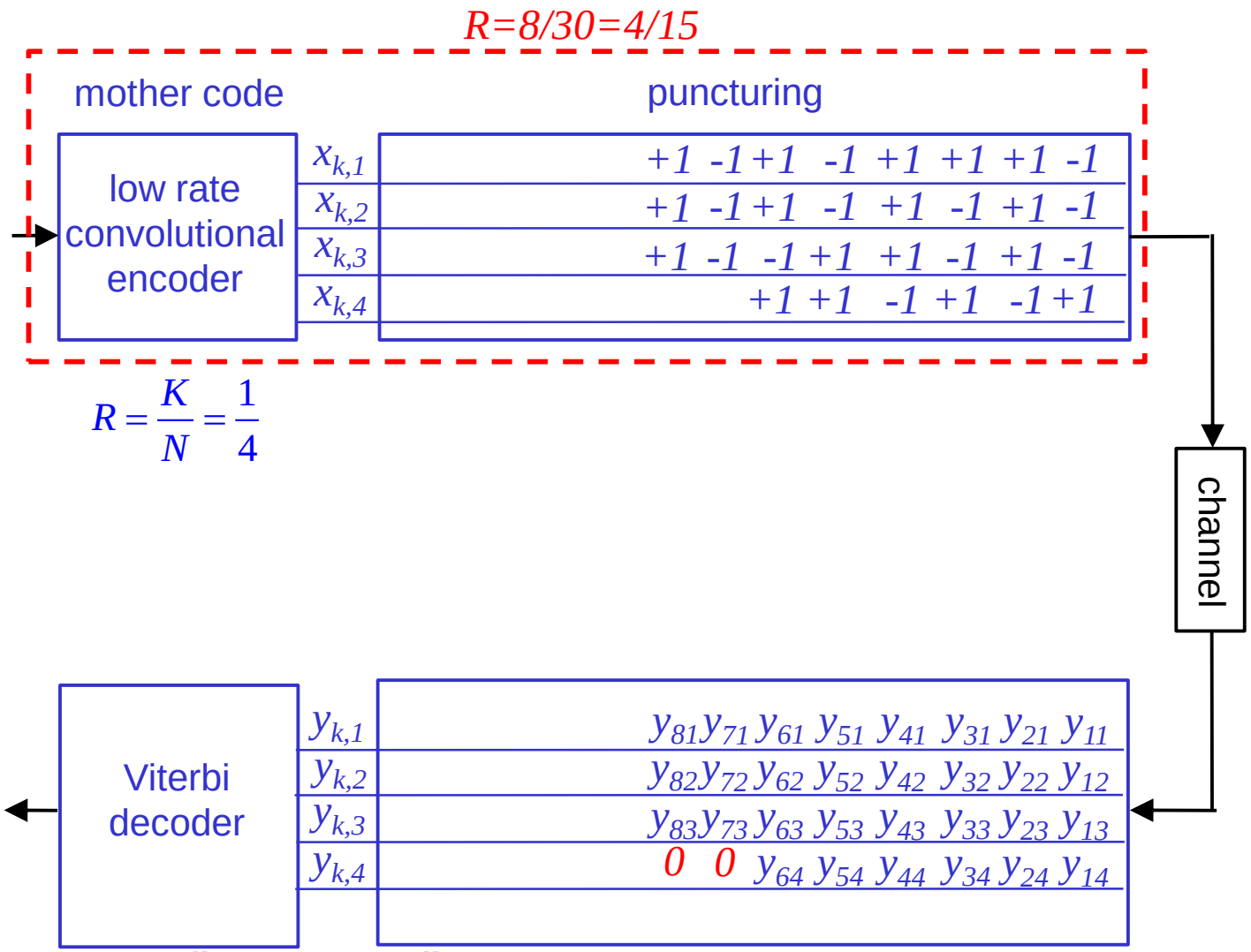
Viterbi decoder

$y_{k,1}$	$y_{81}\ y_{71}\ y_{61}\ y_{51}\ y_{41}\ y_{31}\ y_{21}\ y_{11}$
$y_{k,2}$	$y_{82}\ y_{72}\ y_{62}\ y_{52}\ y_{42}\ y_{32}\ y_{22}\ y_{12}$
$y_{k,3}$	$y_{83}\ y_{73}\ y_{63}\ y_{53}\ y_{43}\ y_{33}\ y_{23}\ y_{13}$
$y_{k,4}$	$0\ \ 0\ \ y_{64}\ y_{54}\ y_{44}\ y_{34}\ y_{24}\ y_{14}$

Metric increment: $\Delta\mu_k\left(s_i,s_j\right)=\sum_{n=1}^{N} y_{k,n}x_{k,n}=\frac{1}{2}\sum_{n=1}^{N} L_c\left(x_{k,n}\right)x_{k,n}$ for $x_{k,n},y_{k,n}\in\{+1,-1\}$

$\Rightarrow y_{k,n}x_{k,n}=0$ for $y_{k,n}=0$

Punctured Convolutional Codes

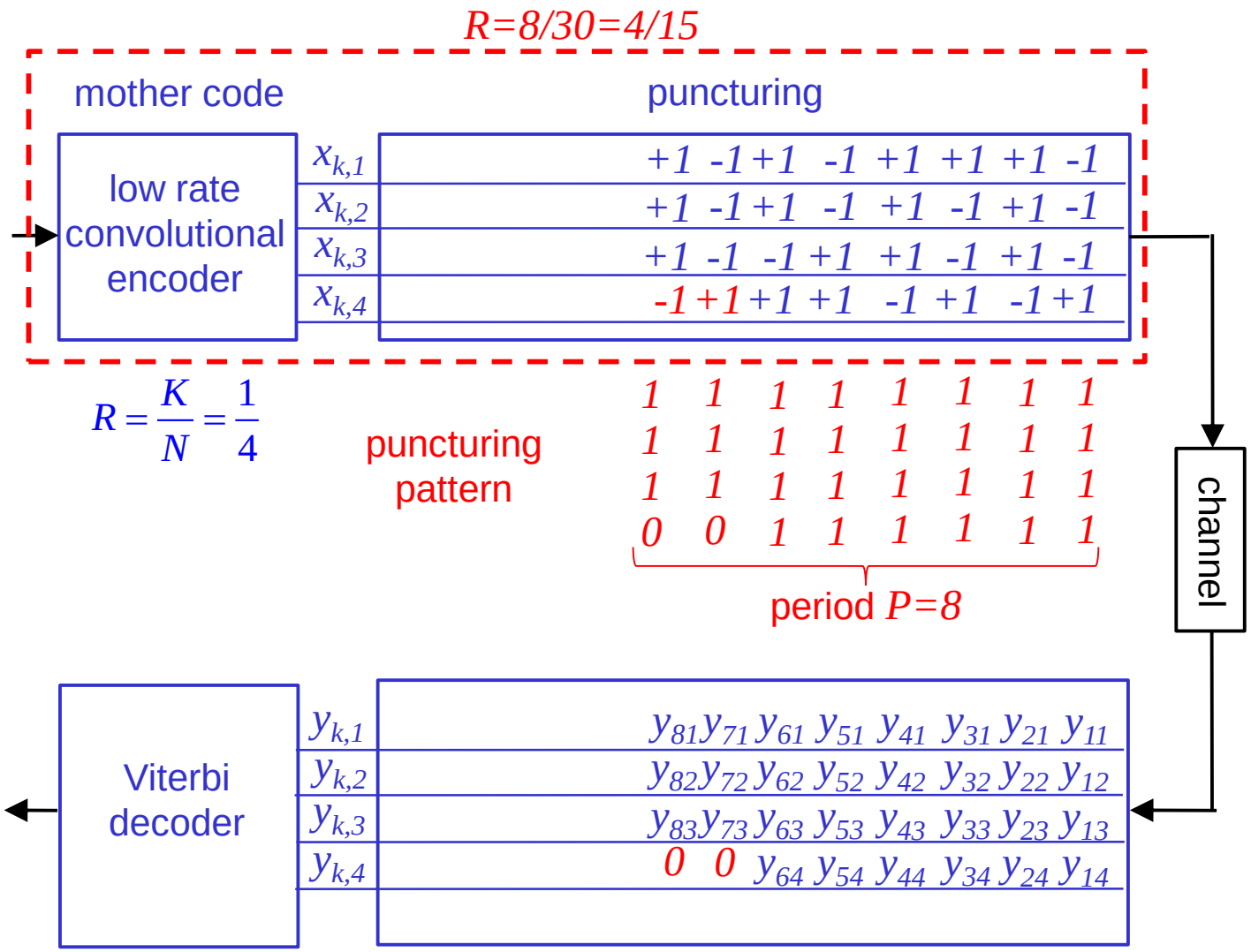


$$R = \frac{K}{N} = \frac{1}{4}$$

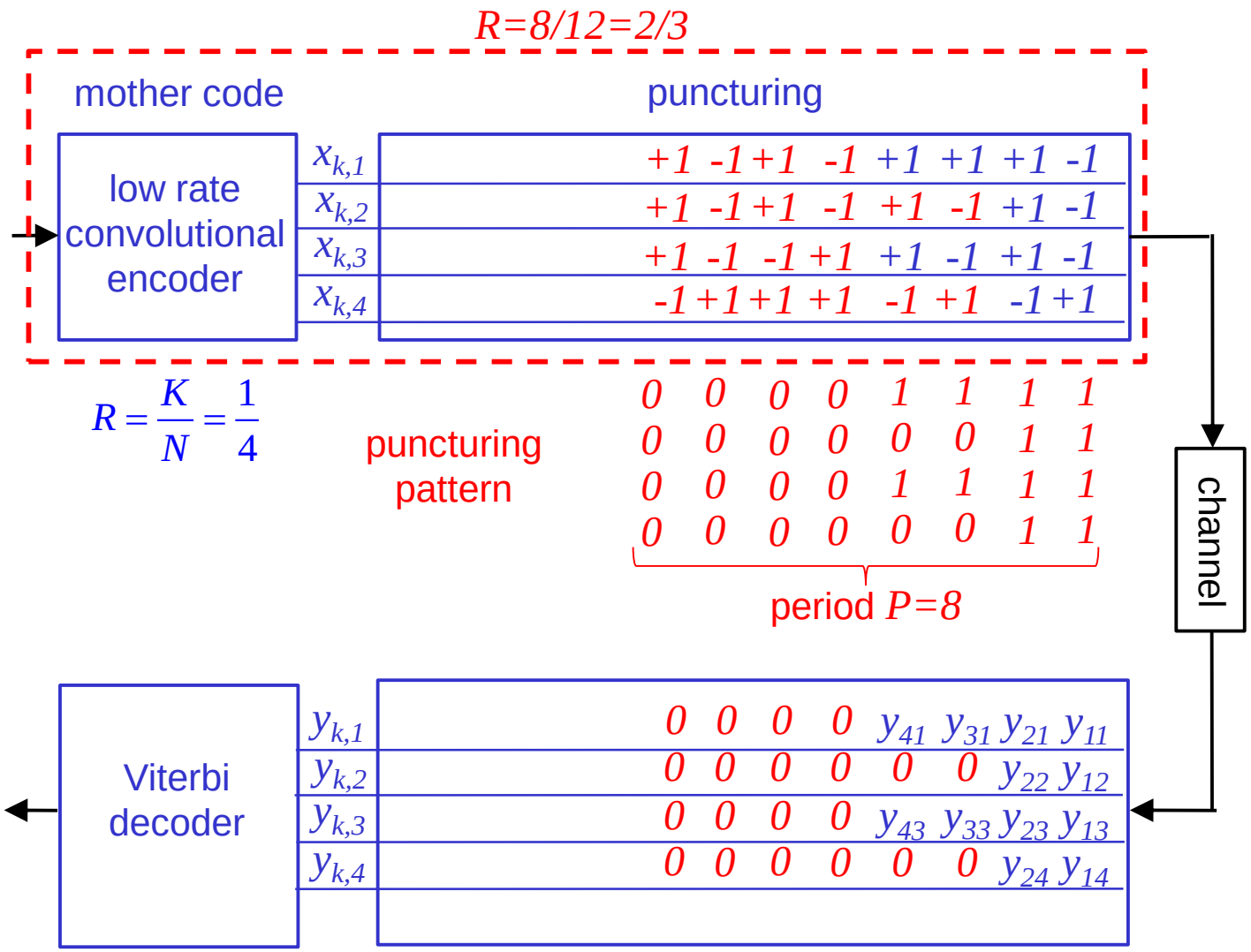
Metric increment: $\Delta\mu_k\left(s_i, s_j\right) = \sum_{n=1}^{N} y_{k,n} x_{k,n} = \frac{1}{2}\sum_{n=1}^{N} L_c\left(x_{k,n}\right) x_{k,n}$ for $x_{k,n}, y_{k,n} \in \{+1, -1\}$

$$L_c\left(x_{k,n}\right) = \log \frac{p\left(y_{k,n} | x_{k,n} = +1\right)}{p\left(y_{k,n} | x_{k,n} = -1\right)} \Rightarrow L_c\left(x_{k,n}\right) = 0 \text{ for } p\left(y_{k,n} \mid x_{k,n} = +1\right) = p\left(y_{k,n} \mid x_{k,n} = -1\right)$$

Punctured Convolutional Codes



Punctured Convolutional Codes



Punctured Convolutional Codes

$R=8/12=2/3$

mother code — puncturing

low rate convolutional encoder

$x_{k,1}$	+1	-1	+1	+1	+1	-1	+1	-1	+1	+1	+1	-1
$x_{k,2}$	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
$x_{k,3}$	-1	+1	+1	-1	+1	-1	-1	+1	+1	-1	+1	-1
$x_{k,4}$	+1	+1	-1	+1	-1	+1	+1	+1	-1	+1	-1	+1

puncturing pattern

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

period $P=8$ — period $P=8$

channel

Viterbi decoder

$y_{k,1}$	$y_{12,1}$	$y_{11,1}$	$y_{10,1}$	y_{91}	0	0	0	0	y_{41}	y_{31}	y_{21}	y_{11}
$y_{k,2}$	0	0	$y_{10,2}$	y_{92}	0	0	0	0	0	0	y_{22}	y_{12}
$y_{k,3}$	$y_{12,3}$	$y_{11,3}$	$y_{10,3}$	y_{93}	0	0	0	0	y_{43}	y_{33}	y_{23}	y_{13}
$y_{k,4}$	0	0	$y_{10,4}$	y_{94}	0	0	0	0	0	0	y_{24}	y_{14}

Rate Compatible Punctured Convolutional Codes (RCPC Codes)

$R=8/12=2/3$

mother code — puncturing

low rate convolutional encoder

$x_{k,1}$	+1	-1	+1	+1	+1	-1	+1	-1	+1	+1	+1	-1
$x_{k,2}$	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
$x_{k,3}$	-1	+1	+1	-1	+1	-1	-1	+1	+1	-1	+1	-1
$x_{k,4}$	+1	+1	-1	+1	-1	+1	+1	+1	-1	+1	-1	+1

$$R=\frac{K}{N}=\frac{1}{4}$$

puncturing pattern for $R=2/3$

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

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

not rate compatible

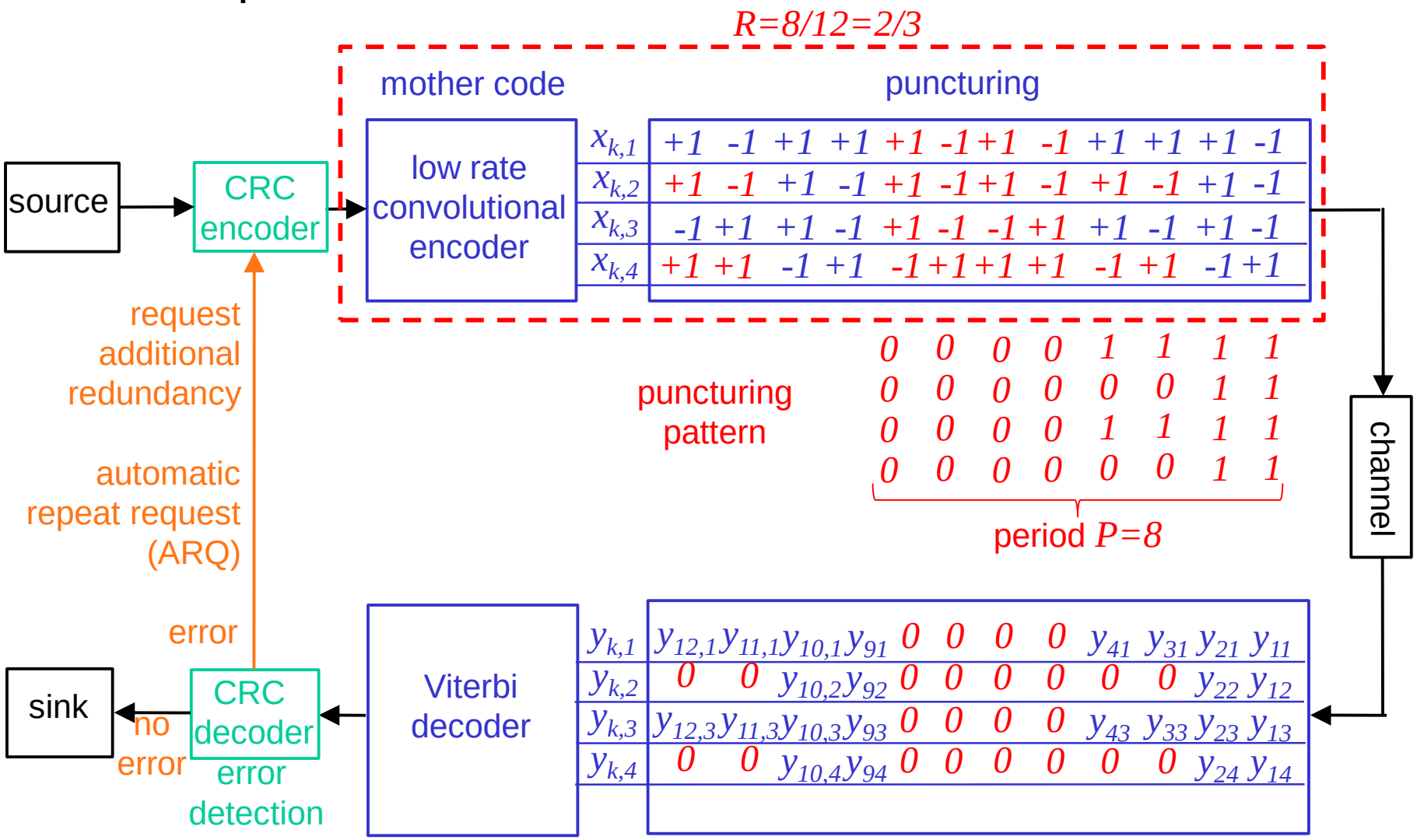
puncturing pattern for $R=1/2$

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

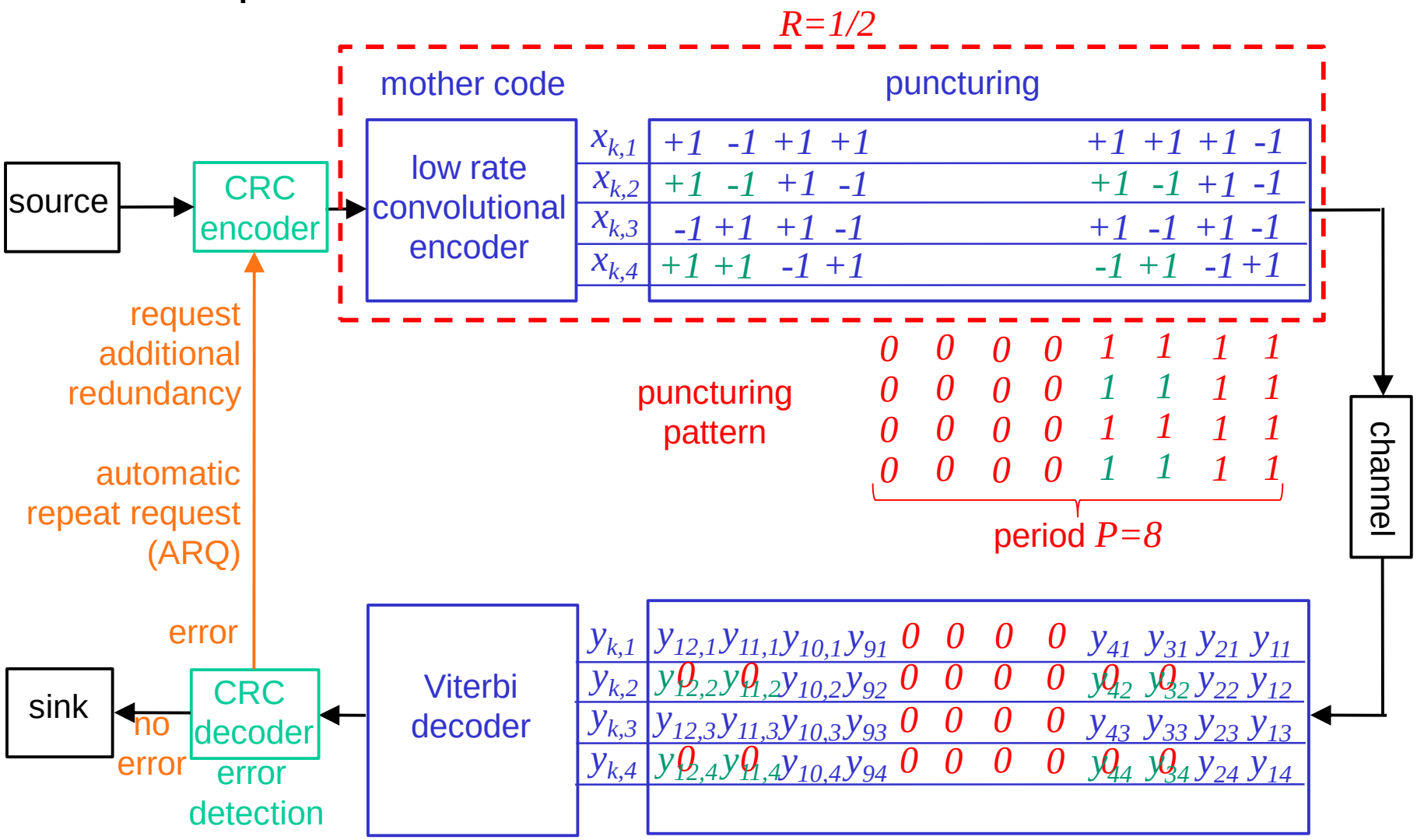
rate compatible

Rate compatible punctured convolutional (RCPC) codes:
The puncturing patterns are defined such that all code bits, which are transmitted at a code rate R, are also transmitted for all lower code rates $R'<R$.

Automatic Repeat Request (ARQ) with Rate Compatible Punctured Convolutional Codes



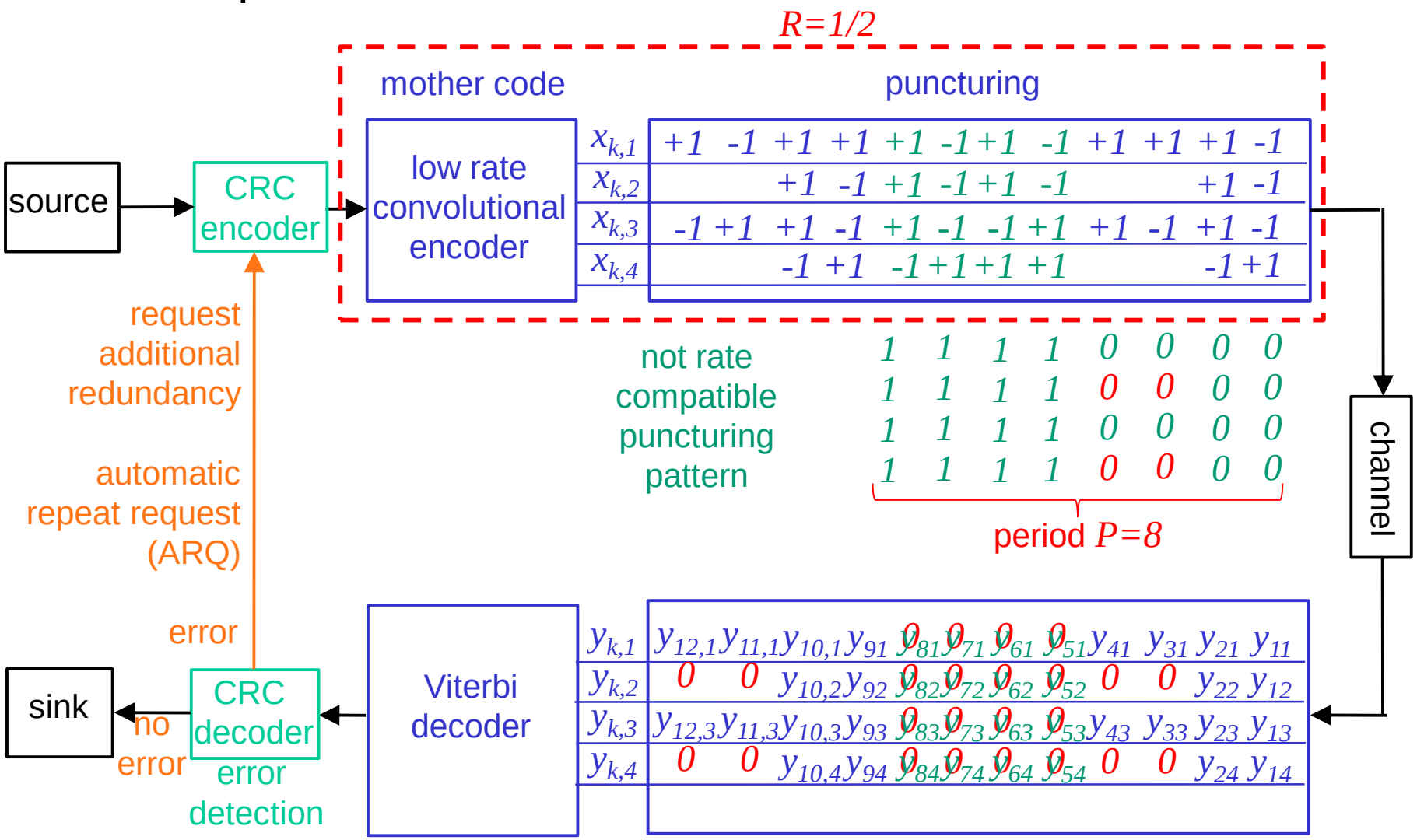
Automatic Repeat Request (ARQ) with Rate Compatible Punctured Convolutional Codes



Transmission of only 4 additional bits per puncturing period $P=8$ provides decoder with code rate $R=1/2$.

Retransmission of complete codeword with rate $R=1/2$ would require transmission of 16 bits per period $P=8$.

Automatic Repeat Request (ARQ) with Rate Compatible Punctured Convolutional Codes



Transmission of 16 additional bits per puncturing period $P=8$ required in order to provide decoder with code rate $R \leq 1/2$.

no advantage

Retransmission of complete codeword with rate $R=1/2$ would require transmission of 16 bits per period $P=8$.

Unequal Error Protection (UEP) with Rate Compatible Punctured Convolutional Codes

Data frame:

important bits	less important bits	least important bits

Puncturing pattern:	$P=P_1$	$P=P_2$	$P=P_3$
Code rate:	$R=R_1$	$R=R_2>R_1$	$R=R_3>R_2$

- Encode complete frame with mother code.
- Use different punturing patterns in different sections

Application: Speech coding

Rate Compatible Punctured Convolutional Codes: Puncturing Patterns

Example:

Mother code $R=1/4$
$M=4$

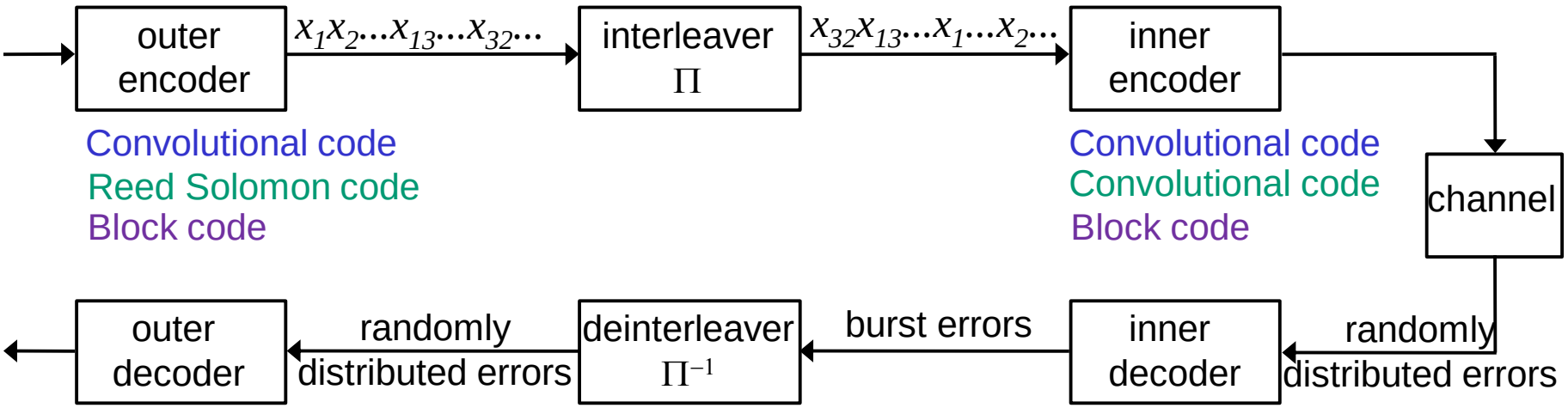
$$\mathbf{G}(D)=\begin{bmatrix} 1+D^3+D^4 \\ 1+D+D^2+D^4 \\ 1+D^2+D^3+D^4 \\ 1+D+D^3+D^4 \end{bmatrix}$$

R	puncturing pattern
8/9	10110000 11000000 11000000 11000000
4/5	11110000 11000000 11000000 11000000
2/3	11110000 11000000 11110000 11000000
4/7	11110000 11110000 11110000 11000000
1/2	11110000 11110000 11110000 11110000
4/9	11111100 11110000 11110000 11110000

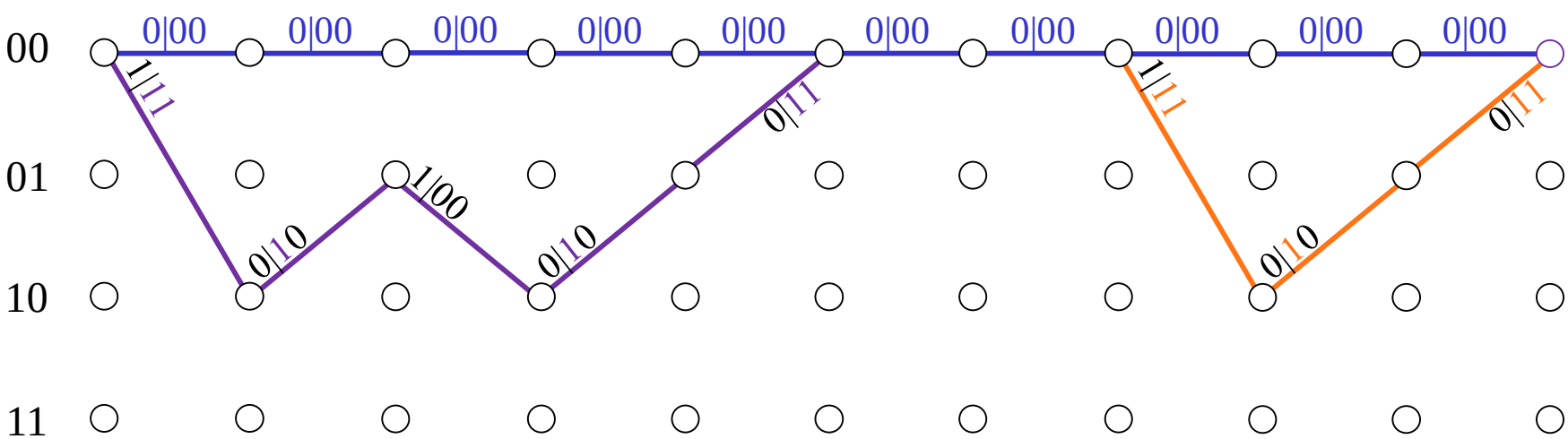
R	puncturing pattern
2/5	11111100 11111100 11110000 11110000
4/11	11111100 11111100 11111100 11110000
1/3	11111100 11111100 11111100 11111100
4/13	11111111 11111100 11111100 11111100
2/7	11111111 11111100 11111111 11111100
4/15	11111111 11111111 11111111 11111100
1/4	11111111 11111111 11111111 11111111

Convolutional Codes: Error Patterns

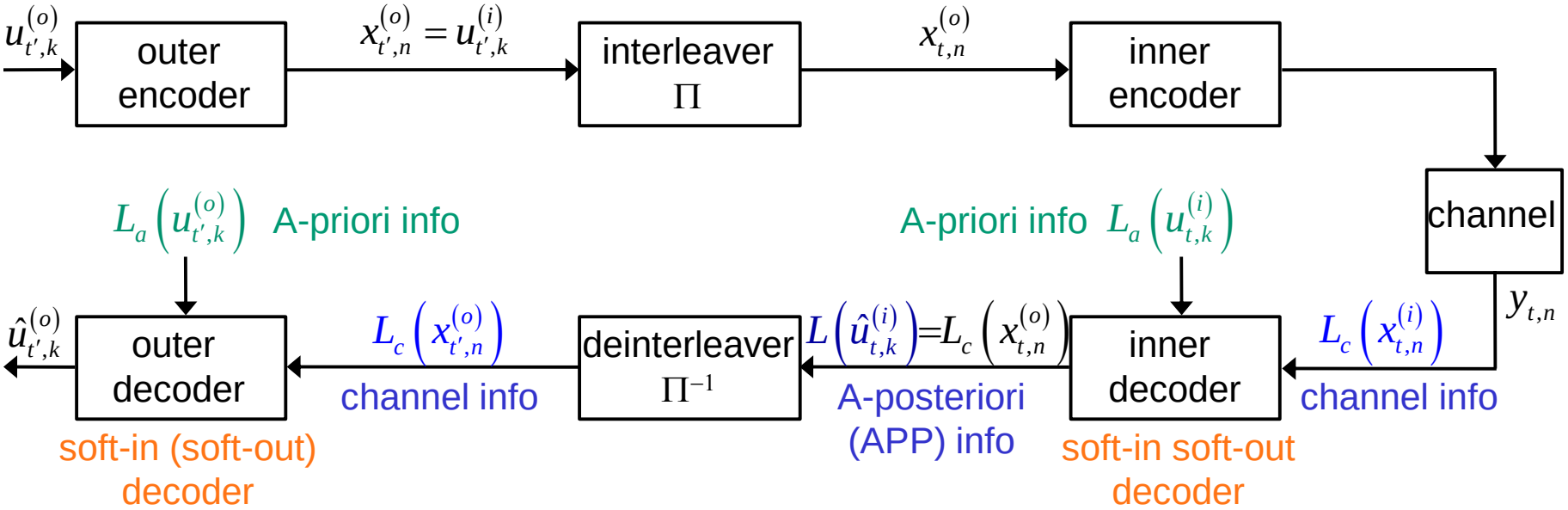
Concatenated codes:



Decoding errors occure typically as burst errors.



Convolutional Codes: Concatenated Codes



$$L\left(\hat{u}_{t,k}^{(i)}\right) = \log \frac{P\left(u_{t,k}^{(i)} = +1 \mid \mathbf{y}\right)}{P\left(u_{t,k}^{(i)} = +1 \mid \mathbf{y}\right)} \qquad L_c\left(x_{t,n}^{(i)}\right) = \log \frac{P\left(y_{t,n} \mid x_{t,n}^{(i)} = +1\right)}{P\left(y_{t,n} \mid x_{t,n}^{(i)} = -1\right)}$$

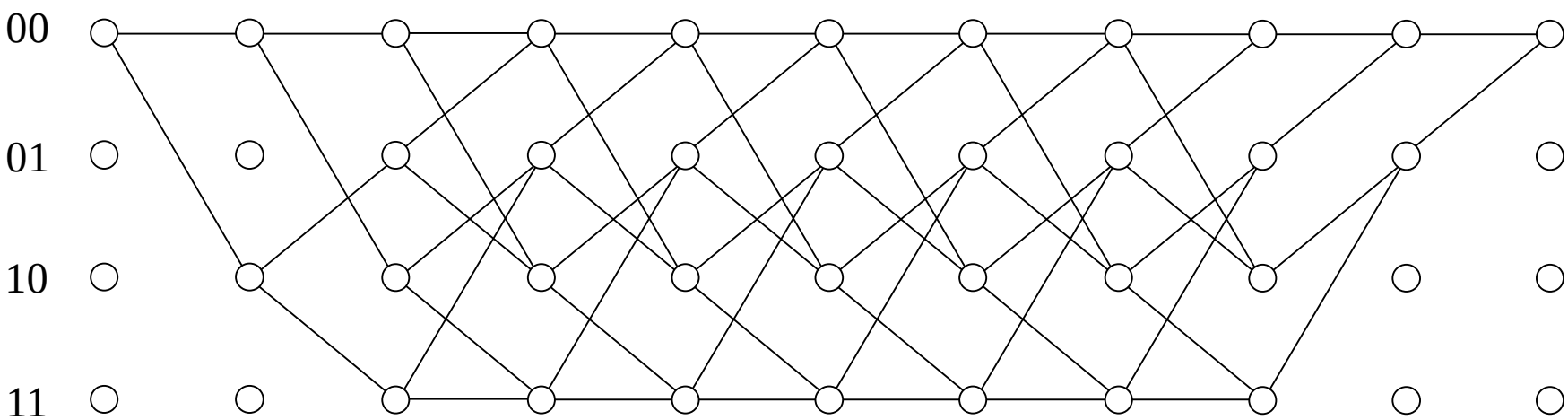
Log-likelihood ratios: $L(x) = \log \frac{P(x=+1)}{P(x=-1)}$

Soft-In Soft-Out Decoder

A-priori info $L_a\left(u_{t,k}\right) = \log \frac{P_a\left(u_{t,k}=+1\right)}{P_a\left(u_{t,k}=+1\right)}$ e.g. from
- source statistics
- other decoding stage

channel info $L_c\left(x_{t,n}\right)$ → soft-in soft-out decoder → $L\left(\hat{u}_{t,k}\right) = \log \frac{P\left(u_{t,k}=+1 \mid \mathbf{y}\right)}{P\left(u_{t,k}=-1 \mid \mathbf{y}\right)}$ A-posteriori (APP) info

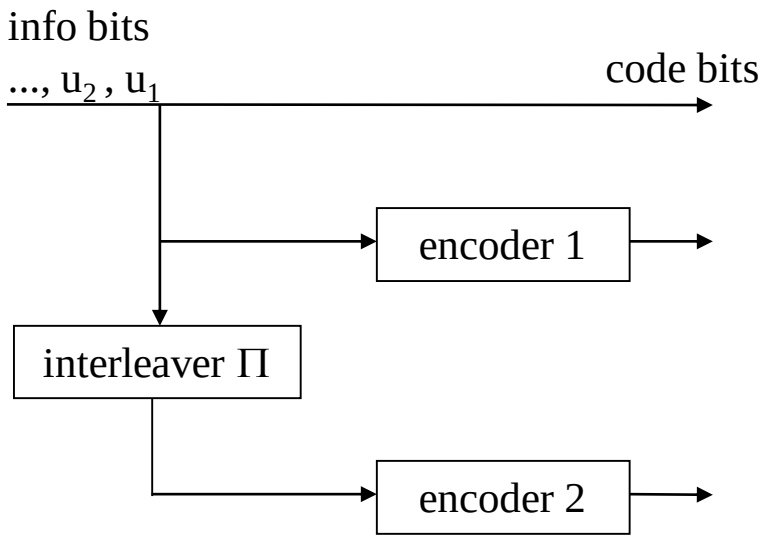
$$= \log \frac{P\left(y_{t,n} \mid x_{t,n}=+1\right)}{P\left(y_{t,n} \mid x_{t,n}=-1\right)}$$



Turbo Codes

Idea:

Build strong code by concatenation of relatively simple constituent codes.



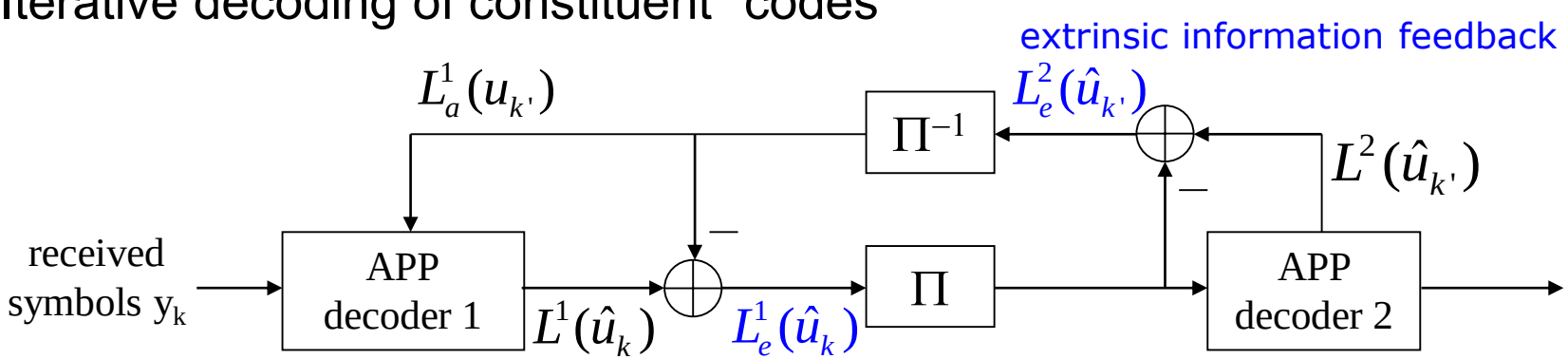
Soft information:

$$L(u_k) = \log \frac{p(u_k = +1)}{p(u_k = -1)}$$

hard decision: sign $L(u_k)$

reliability: $|L(u_k)|$

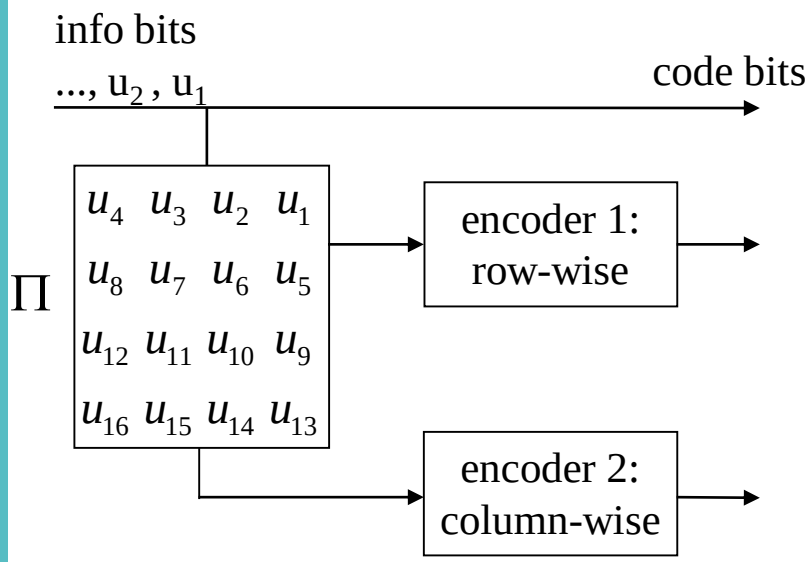
Iterative decoding of constituent "codes"



Turbo Codes

Idea:

Build strong code by concatenation of relatively simple constituent codes.



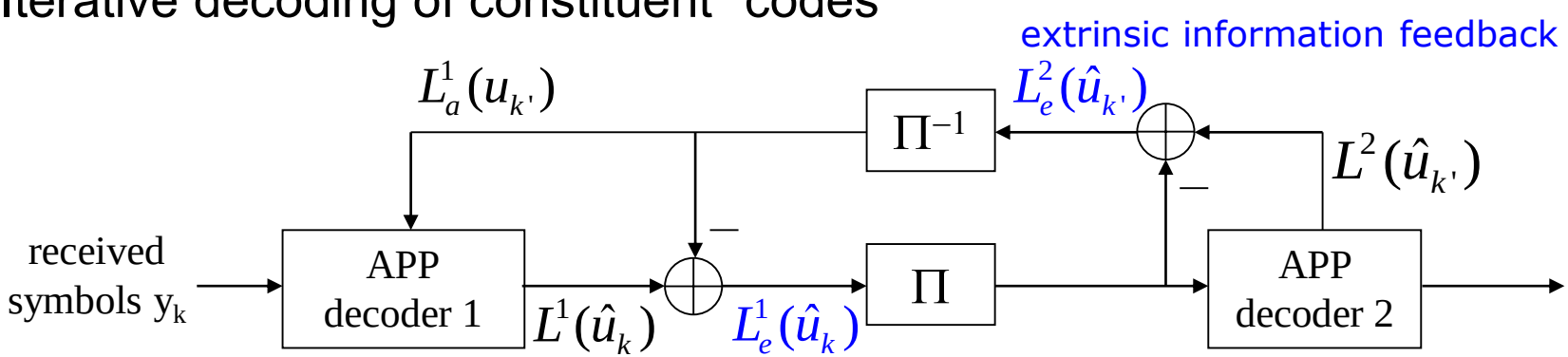
Soft information:

$$L(u_k) = \log \frac{p(u_k = +1)}{p(u_k = -1)}$$

hard decision: sign $L(u_k)$

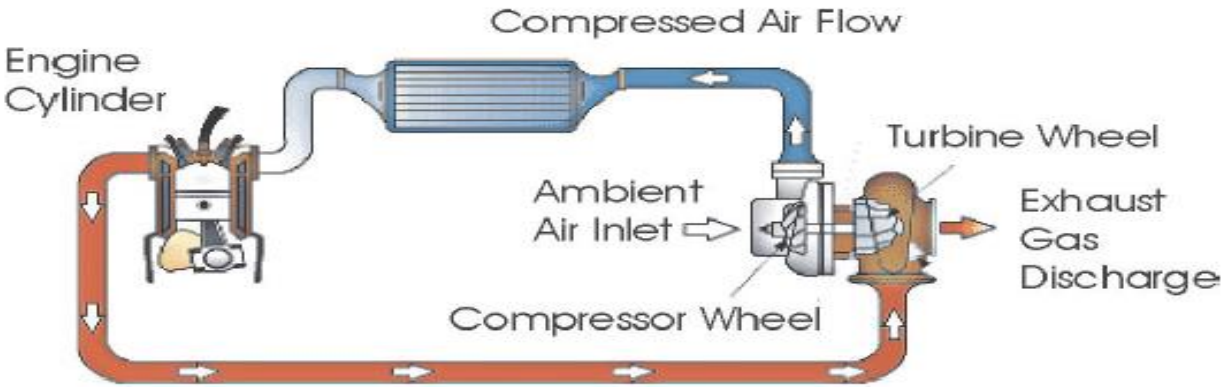
reliability: $|L(u_k)|$

Iterative decoding of constituent "codes"

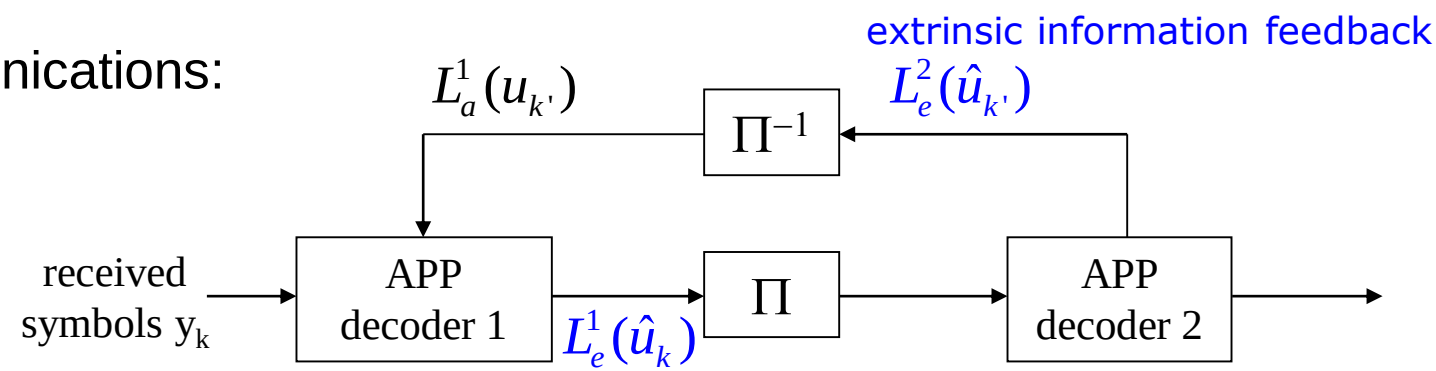


The Turbo Principle in Communications– Analogy to Turbo Engine

mechanics:

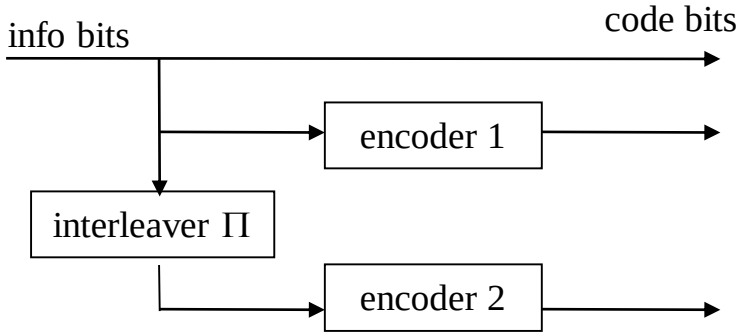


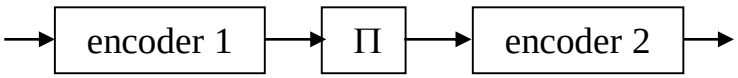
communications:



The Turbo Principle

Parallel or serial concatenation of constituent “codes”

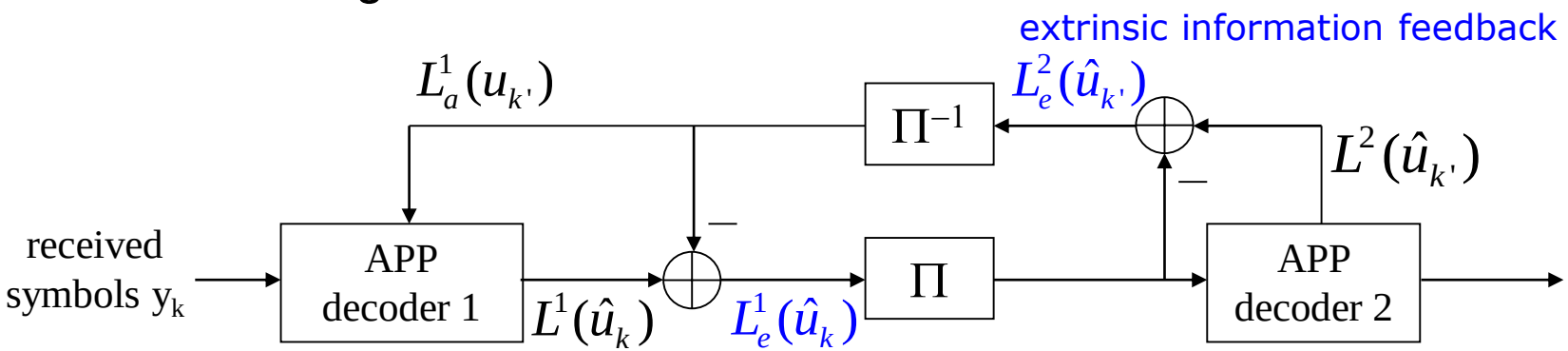




constituent “codes” can be

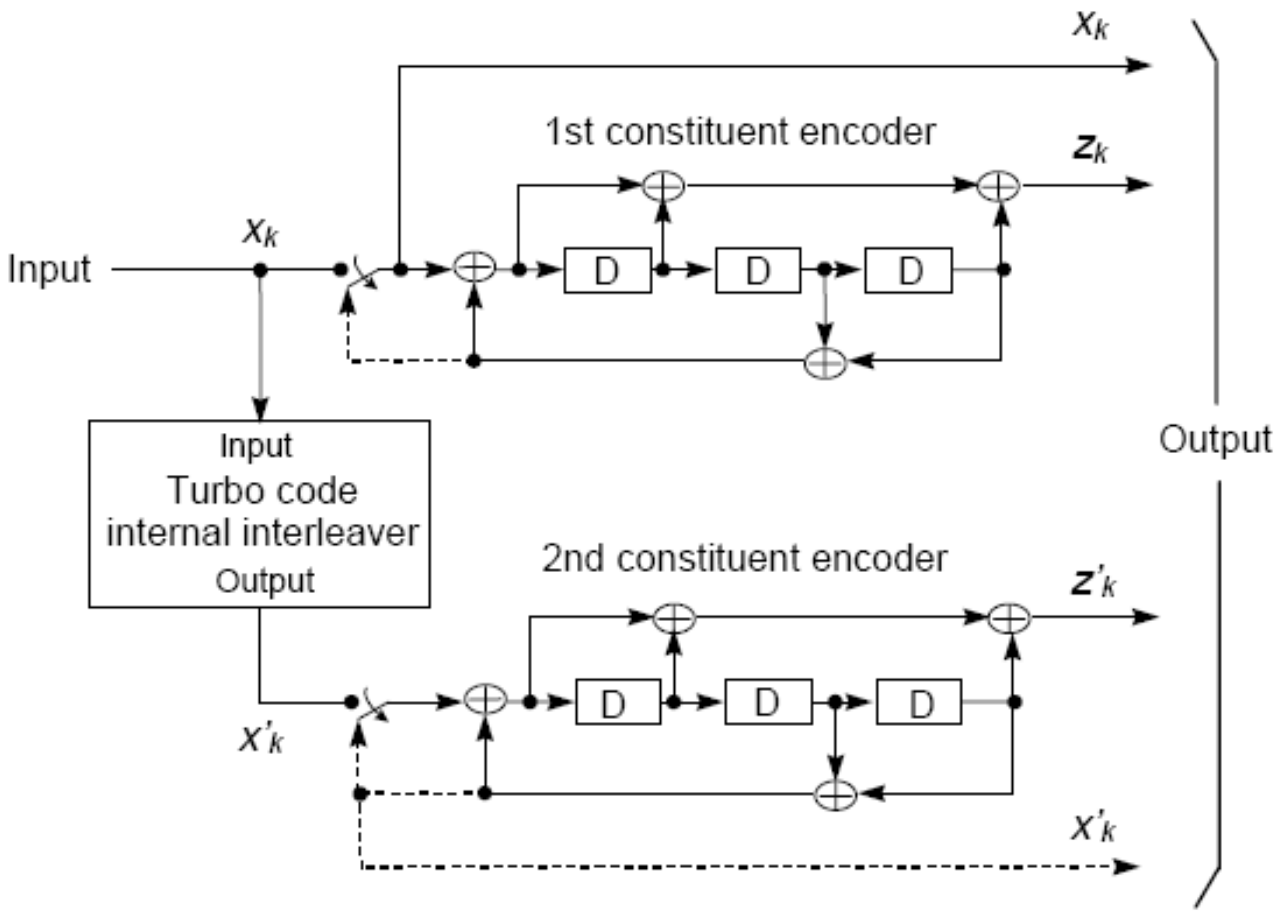
- FEC codes
- ISI channel
- MIMO channel
- multiuser channel
- QAM/PSK mapper
- diversity combiner
- ...

Iterative decoding of constituent “codes”



Turbo Code in UMTS and LTE

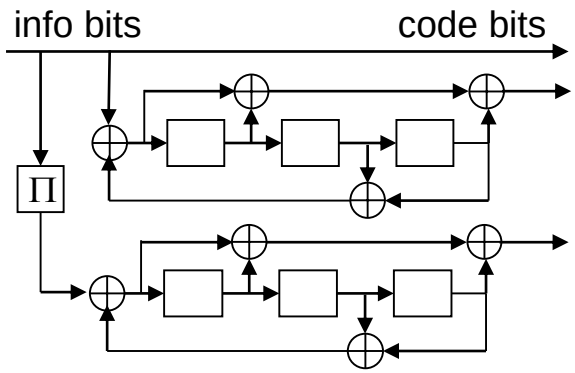
UMTS: Universal Mobile Telecommunications System
LTE: Long Term Evolution



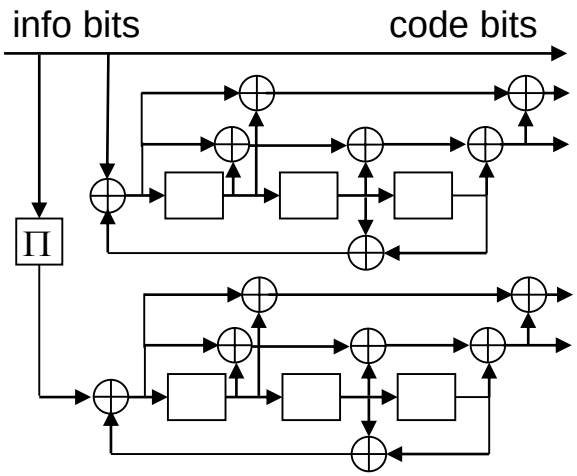
Applications of Turbo Codes

- CCSDS[1] Telemetry Channel Coding Standard 2001
  - Constituent codes: *R=1/4*, 16-state convolutional codes with selectable rates and block lengths.
  - Code rates of turbo code: *R=1/2, 1/3, ¼, 1/6*
  - Max. coding gain of 2.5 dB at BER 10^{-5} with *R=1/6* over traditional coding scheme (Reed Solomon Code + 64 state convolutional code).
- 3G Wireless Standard (IMT[2] 2000) and LTE (Long Term Evolution)
  - 256-state convolutional codes and turbo codes

WCDMA (UMTS), LTE



CDMA 2000

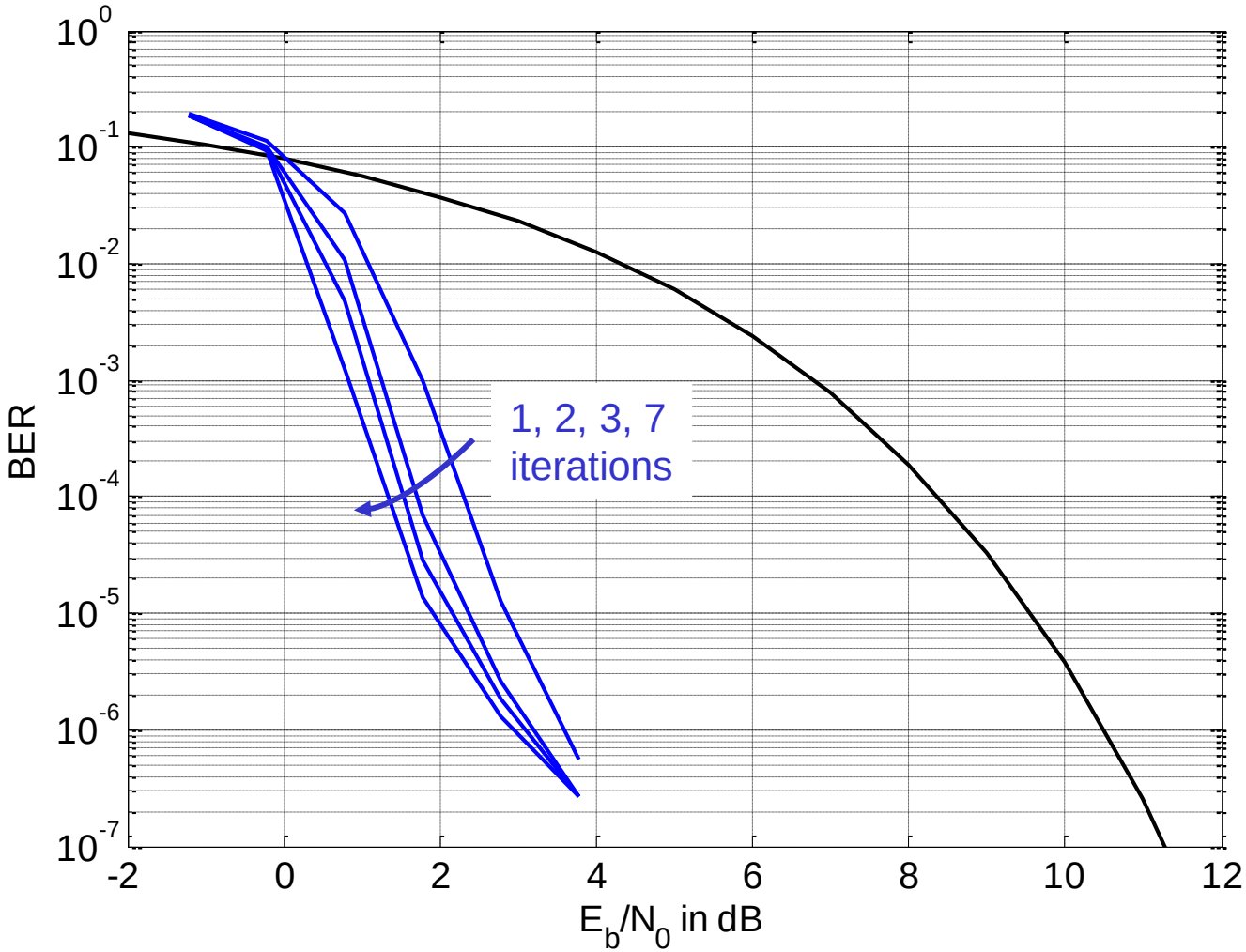


- Digital Video Broadcasting (DVB)

[1]CCSDS: Consultative Committee on Space Data Systems
[2]IMT: International Mobile Telecommunications

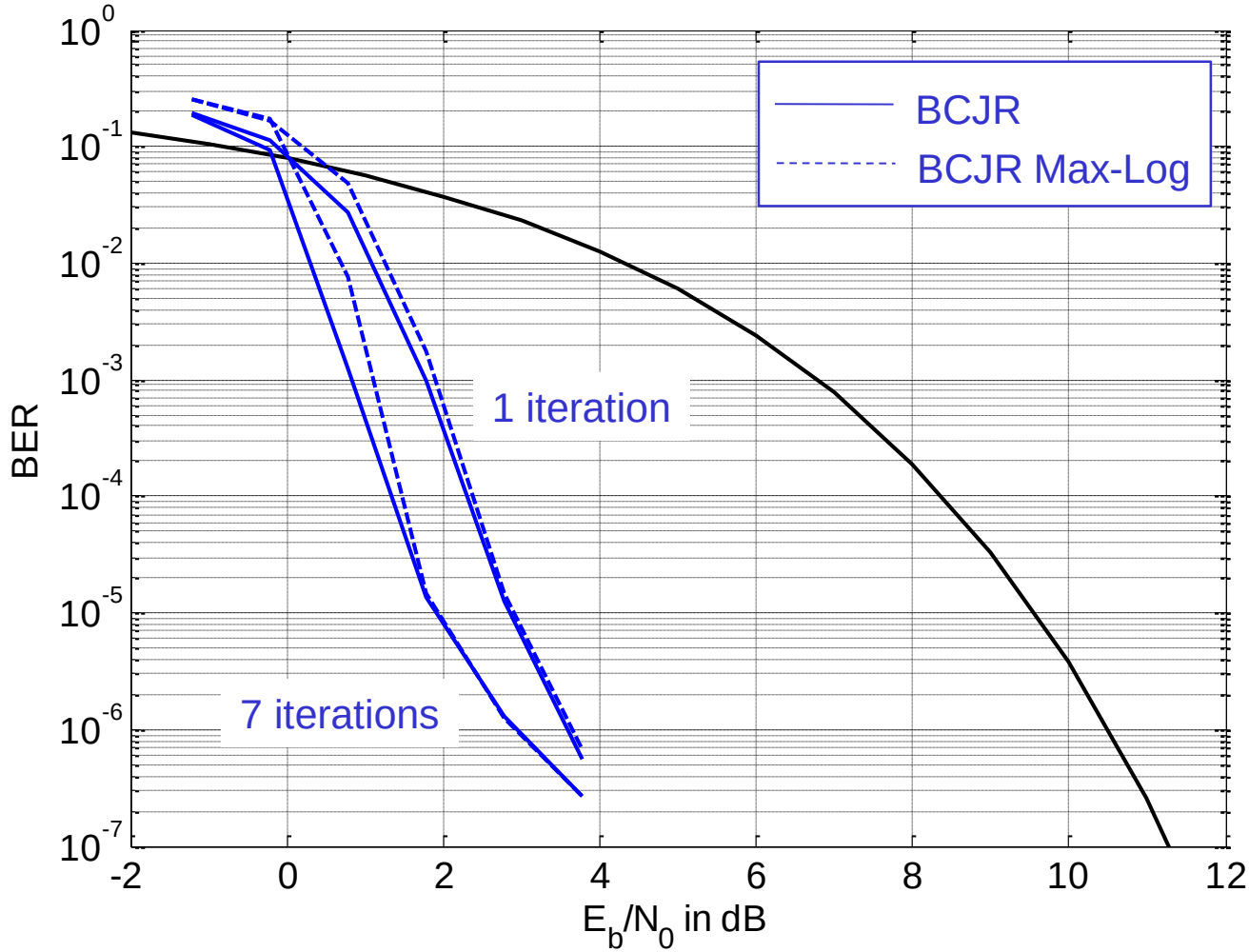
BER Performance of Turbo Codes

R=1/3 turbo code
Constituent codes: RSC codes with memory M=2 and generators g=[1, 5/7]
BCJR decoder, random interleaver
BPSK modulation, 100 info bits/codeword
AWGN channel



BER Performance of Turbo Codes

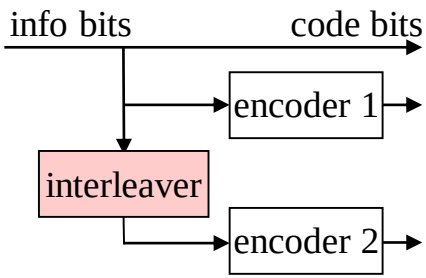
R=1/3 turbo code
Constituent codes: RSC codes with memory M=2 and generators g=[1, 5/7]
BCJR or BCJR Max.Log decoder, random interleaver
BPSK modulation, 100 info bits/codeword
AWGN channel



Interleaver Design

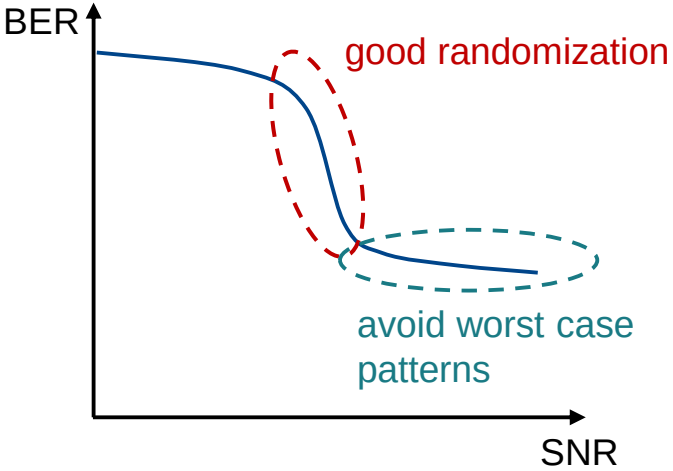
Typical interleaver design criteria for iterative decoding:

- Good performance with iterative decoding:
  - Good randomization/spread.
  - Avoid worst case patterns.



- Flexibility, memory requirements:
  Communications systems use variable block size.
  Storing of interleaver patterns for all supported block sizes needs huge memory.
  $\Rightarrow$ Interleaving rule should be easy to generate e.g. based on a simple equation or simple operations on a common mother interleaver.

- Allow parallel implementation of decoder (memory collision problem).



Coded Modulation

It is the idea of coded modulation to combine the design of error control coding and digital modulation, i.e. the mapping of information or code symbols to symbols of a QAM constellation. Info bits are the input to the encoder of a coded modulation scheme, the QAM transmit symbols are the output. The joint design of error control coding and modulation allows substantial gains over separate treatment of coding and modulation.

Error control coding adds redundancy (parity symbols) to the signal and, hence, requires more bandwidth then uncoded transmission. The fundamental idea is, to use a higher order modulation scheme in order to transmit the redundancy without bandwidth expansion. In uncoded transmission, higher order modulation schemes are more sensitive to channel distortions such as noise, since the minimum Euclidean distance of constellation points is decreased. I.e., the error rate is increased at a given SNR. However, this degradation is overcompensated by the coding gains.

Information theory suggests that it is sufficient to transmit one additional code bit per transmit (QAM) symbol, i.e. to double cardinality of the symbol alphabeth: The channel capacity in an AWGN channel is obtained with Gaussian distributed transmit symbols. A degradation results from using real world QAM constellations instead of Gaussian transmit symbols. In particular, the maximum achievable rate is limited by the number of bits per transmit symbol E.g. with QPSK modulation, we can transmit at most 2 bits/symbol, i.e. 2 bit/channel use. At a bandwidth efficiency of 2 bit/channel use, the capacity curve for 8-PSK is almost identical with the optimum capacity curve for Gaussian transmit symbols. This means, that theoretically, we can obtain a bandwidth efficiency of 2 bit/channel use by expanding the constellation of the modulation scheme from QPSK to 8-PSK and using channel coding with a code rate of R=2/3.

Coded Modulation

The approach of coded modulation is, to increase the Euclidean distance of allowed transmit sequences. In uncoded transmission, or if error control coding and modulation are not designed jointly, transmit sequences with low Euclidean distance can occur. This results in a high error probability of the uncoded sequence or at the input of the decoder, respectively. Coded modulation makes sure that transmit sequences with low Euclidean distance donot occur.

Coded modulation can be based on block codes or convolutional codes. Coded modulation which is based on convolutional codes is called *Trellis Coded Modulation (TCM).* TCM codes can be represented by a trellis segment or using the generator polynomial description of convolutional codes.

Efficient ML decoding can be done using trellis decoders such as the BCJR or Viterbi algorithm. In particular, the trellis representation allows efficient soft decision decoding.

The mapping of code bits to transmit symbols is a non-linear operation. Therefore, it is not sufficient anymore to consider only the Euclidean distance to the all zero sequence for performance assessment. This makes analytical treatment of coded modulation more difficult than for linear codes.

Coded Modulation in Rayleigh Fading Channels

In Rayleigh fading channels, the minimum Hamming distance among any two allowed transmit sequences turns out to be a more important design criterion than the Euclidean distance. In Rayleigh fading channels, the uncoded error rate decreases only linearly with increasing SNR, whereas it decreases exponentially in AWGN channels. Therefore, the main target of channel coding is to increase the slope of the error rate vs. SNR curve. The minimum Hamming distance of the code determines the diversity level of the coding scheme. Diversity refers to the slope of the bit error rate curve: A diversity level of $d=1$ means that in a double logarithmic schale, the error rate vs. SNR curve decreases linearly by d decades per 10 dB increasing SNR. Maximization of the Euclidean distance yields an additional coding gain, i.e. a parallel shift of the BER curve towards lower SNR, but no steeper slope. This coding determined is given by the so-called squared product distance.

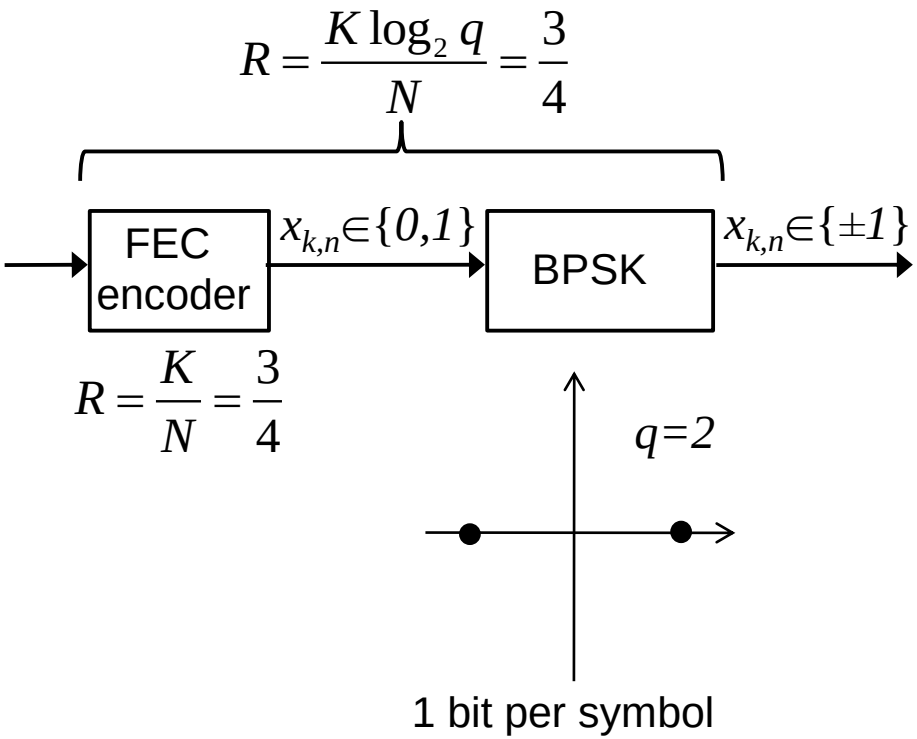
Trellis Coded Modulation (TCM)

A powerful class of TCM codes are Ungerböck codes. Ungerböck codes use a q-ary modulation format in order to transmit q-1 info bits per transmit symbol.
They are based on the idea of set partitioning: Set partitioning refers to a mapping of code bits to QAM symbols such that the first bit chooses one of two subsets of constellation points. The second bit chooses a subset within the subset which was defined by the first bit etc.
The subsets are chosen such that the Euclidean distance within a subset increases from step to step. Consequently, the error probability within a subset decreases from step to step. This can be exploited by a matched error control coding scheme: The first bit, which differentiates between two subsets at minimum Euclidean distance needs stronger protection than bits which differentiate between later subsets at higher Euclidean distance. For the subsets at sufficiently large Euclidean distance, the repective bits may even be left uncoded.
For Ungerböck codes, a subset of info bits is encoded using a convolutional encoder. The code bits of this convolutional code choose a subset of the QAM constellation. The remaining info bits choose a constellation point within this subset. Since the Euclidean distance within the subset is larger than the minimum Euclidean distance of the QAM constellation, the uncoded bits are expected to be detected with sufficiently low error probability.
A TCM scheme which contains uncoded bits results in a trellis with parallel transitions since the uncoded bits have no impact on the convolutional encoder part. However, trellis decoders such as the BCJR or Viterbi algorithm can still be used in a straightforward manner: The sums in the BCJR have to be taken over all transitions (including the parallel transitions). The Viterbi algorithm computes metric increments for all transitions, no matter wether they originate from the same or from different states. Only one incoming branch survives per state. The trellis decoders automatically exploit the increased Euclidean distances which are obtained for the uncoded schemes by set partitioning.

Trellis Coded Modulation (TCM)

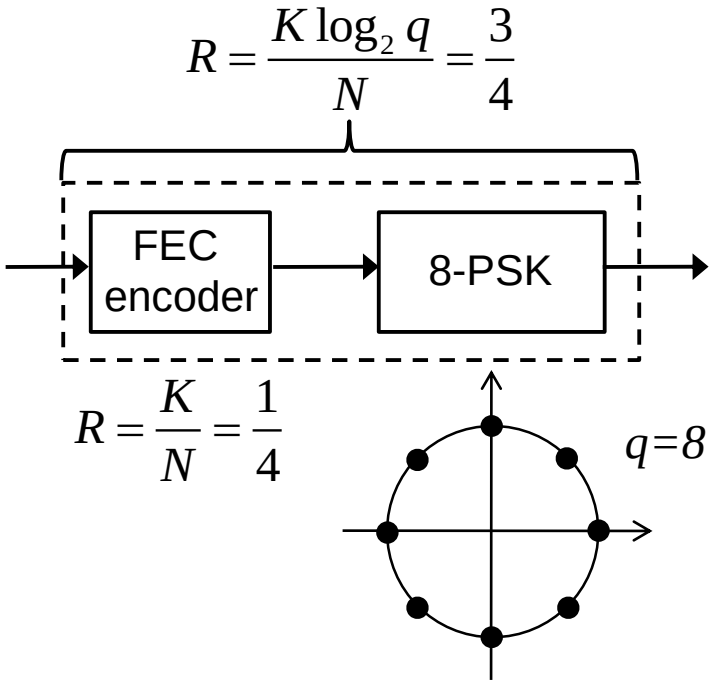
In the same way as for original convolutional codes, the convolutional encoder in a Ungerböck encoder can be realized as non-systematic FIR encoder or as an equivalent recursive systematic encoder.
The convolutional codes to be used in TCM have to be chosen carefully: Optimum binary convolutional codes are not necessarily optimum in TCM. TCM requires a search for new codes. The design criterium is to maximize the free Euclidean distance which is defined as the minimum Euclidean distance among all pairs of possible transmit sequences. The free Euclidean distance plays a similar role for TCM as the free Hamming distance for classical binary convolutional codes. Since the mapping of code bits to transmit sequences is a non-linear operation, it is **not** sufficient to maximize only the Euclidean distance between the all zero sequence and all other possible transmit sequences. However, there exist simplification methods for computationally efficient search for codes with maximum free Euclidean distance.

Coded Modulation



1 bit per symbol

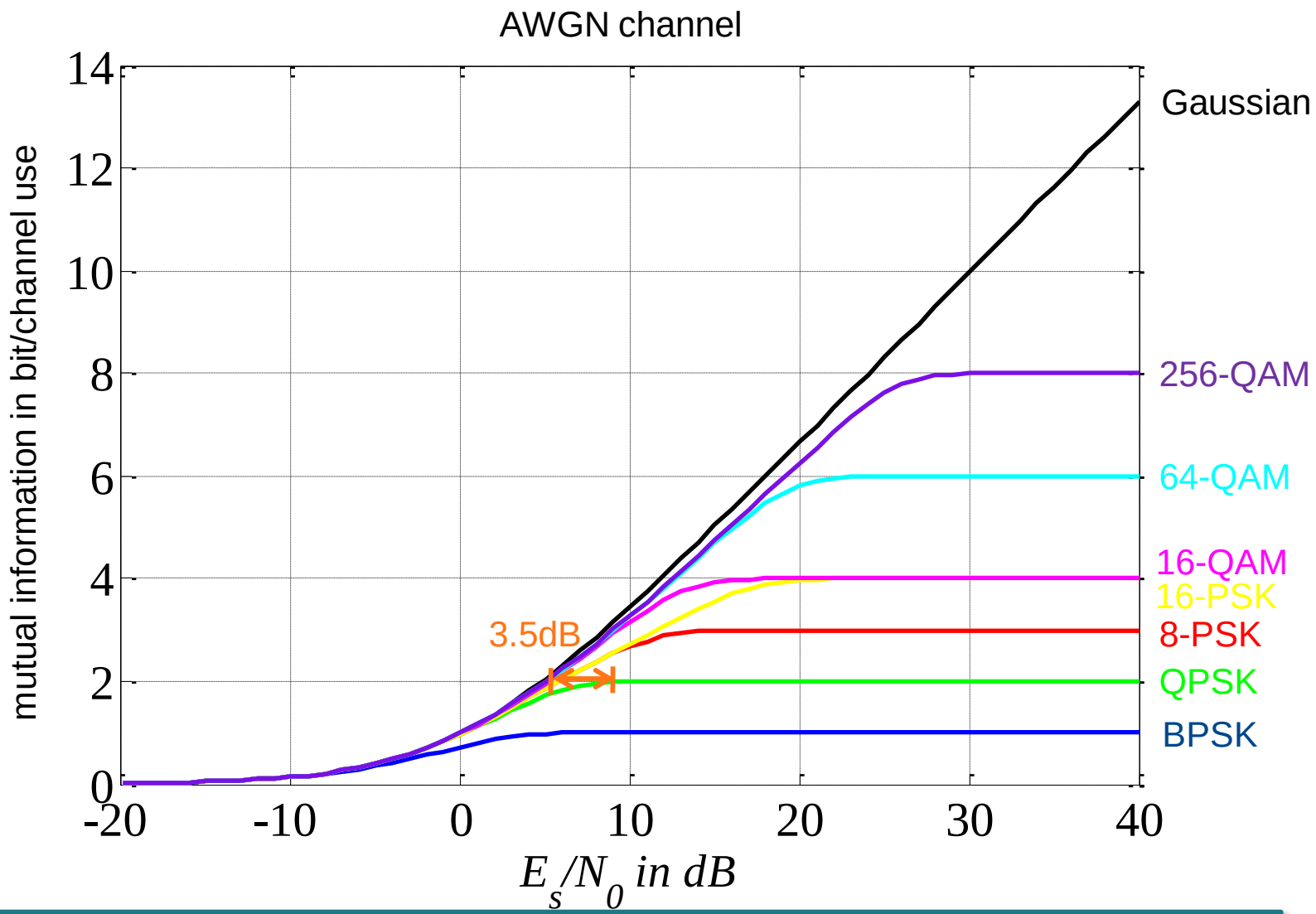
joint design of FEC code and q-ary modulation



3 bits per symbol

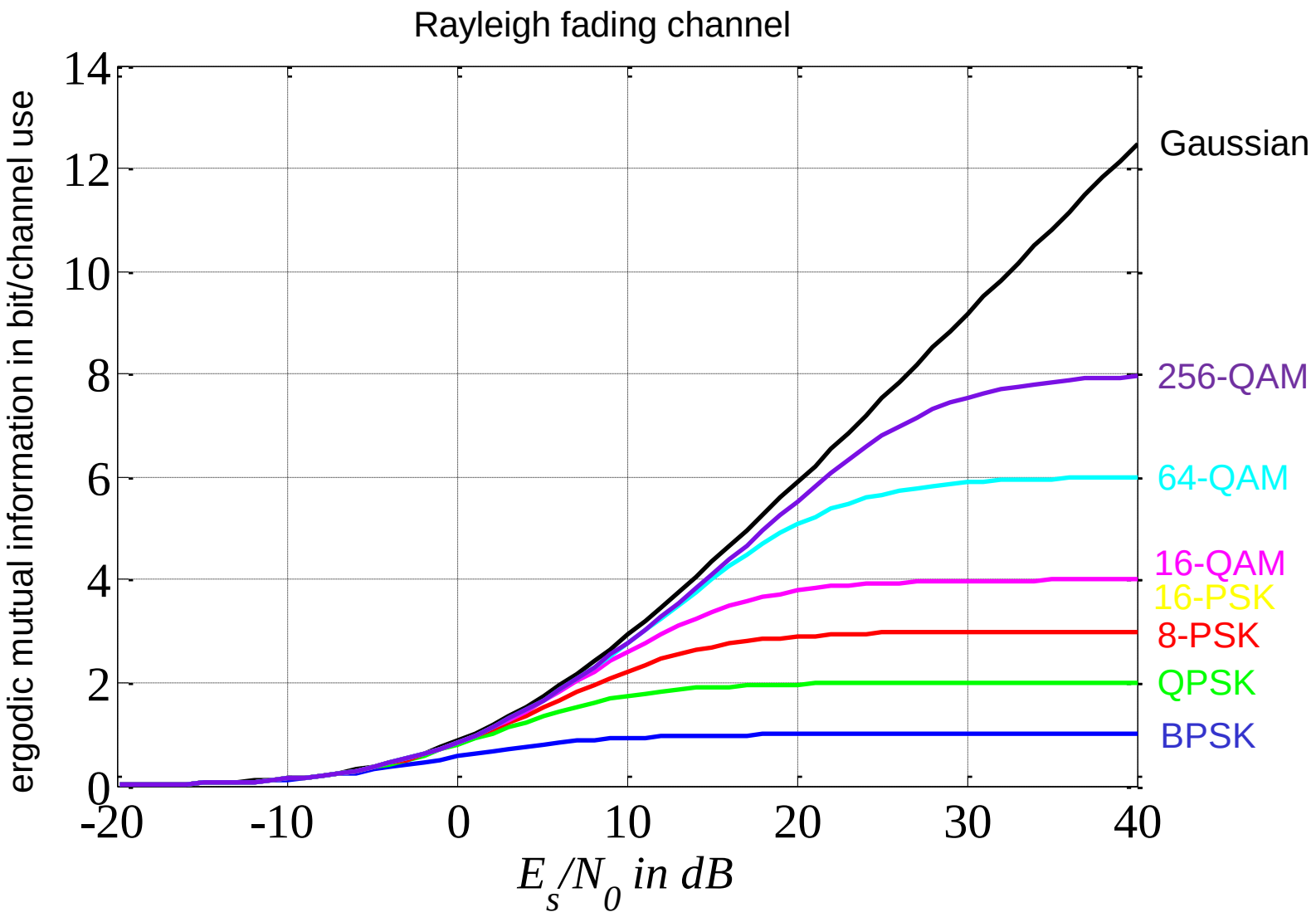
Idea:
- Use higher order modulation in order to allow for lower code rate without bandwidth expansion.
- Design FEC code and QAM constellation jointly.

Achievable Rates for PSK/QAM Constellations



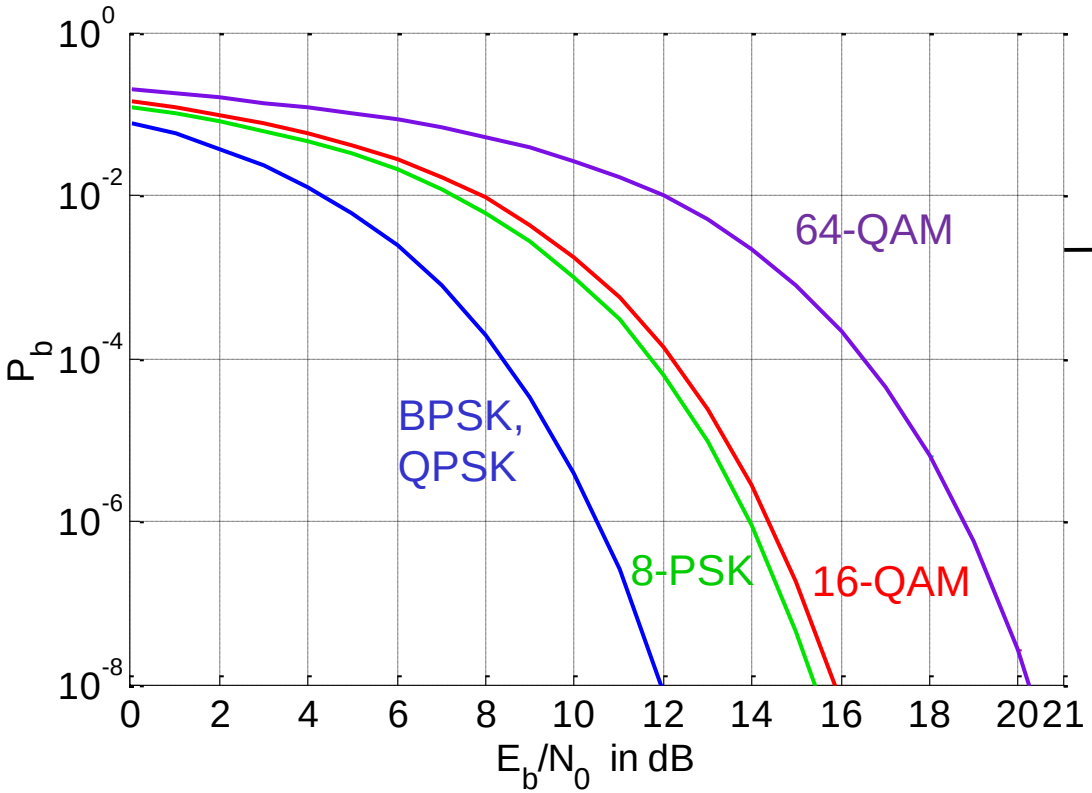
→ Use higher order modulation with lower code rate
→ 1 additional bit per symbol is sufficient (double cardinality of modulation scheme)

Achievable Rates for PSK/QAM Constellations



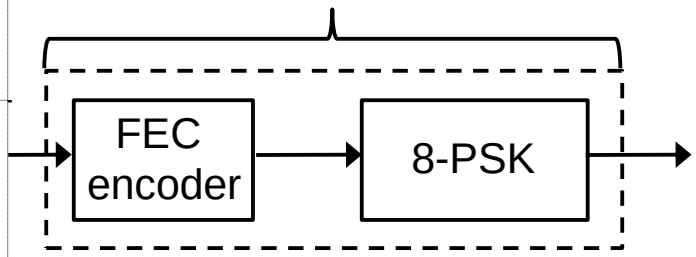
Coded Modulation

Uncoded bit error probability of higher order modulation

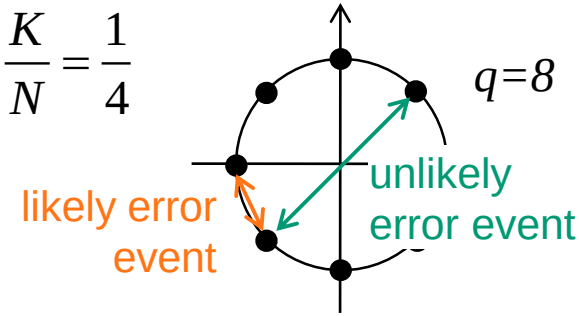


joint design of FEC code and q-ary modulation

$$R = \frac{K \log_2 q}{N} = \frac{3}{4}$$

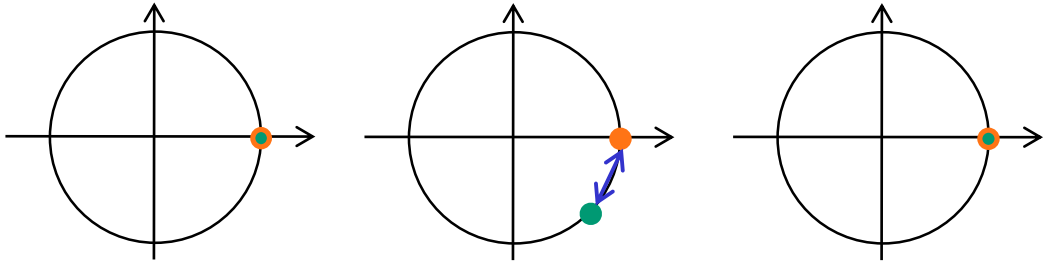


$$R = \frac{K}{N} = \frac{1}{4}$$



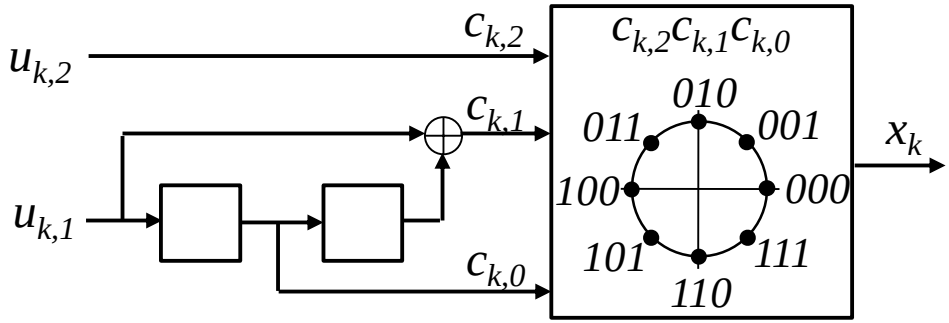
3 bits per symbol

Transmit sequences with low Euclidean distance (independent coding and modulation):



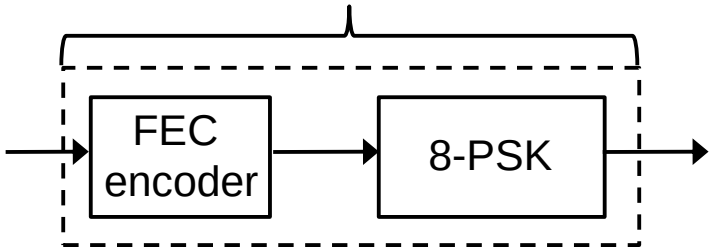
Joint design of FEC coding and modulation should increase the Euclidean distance of valid sequences.

Trellis Coded Modulation (TCM)

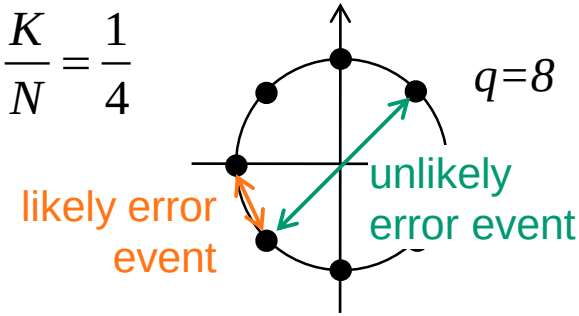


joint design of FEC code and q-ary modulation

$$R = \frac{K \log_2 q}{N} = \frac{3}{4}$$

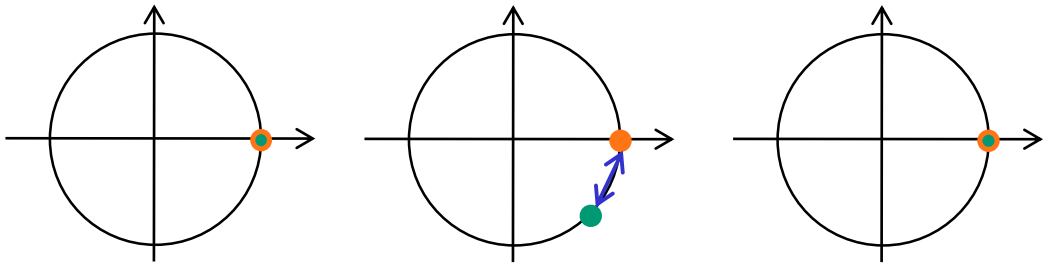


$$R = \frac{K}{N} = \frac{1}{4}$$



3 bits per symbol

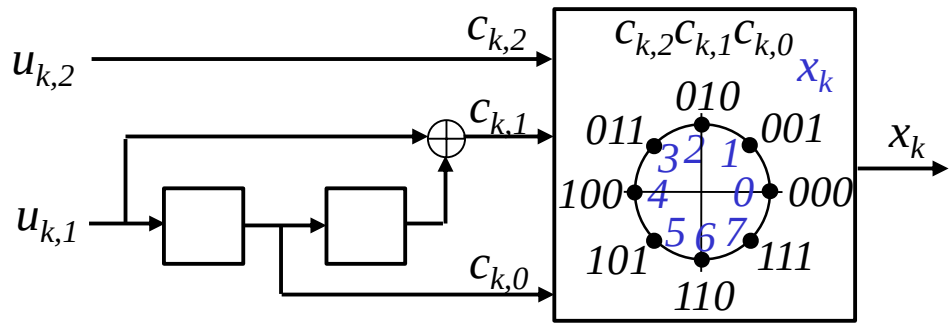
Transmit sequences with low Euclidean distance (independent coding and modulation):

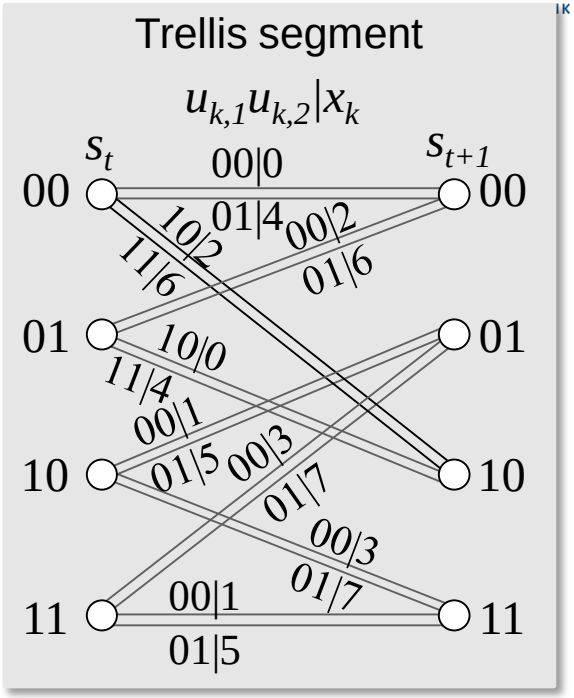


Euclidean distance:

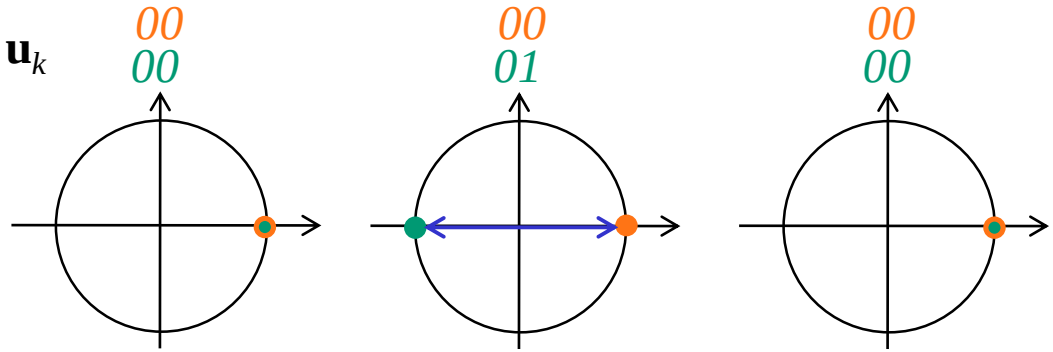
$$\Delta = \sqrt{2 - \sqrt{2}}$$

Trellis Coded Modulation (TCM)





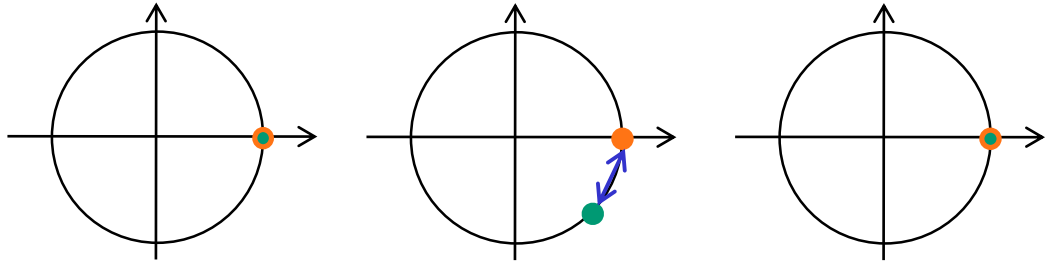
Transmit sequences:



Euclidean distance:

$\Delta = 2$

Transmit sequences with low Euclidean distance (independent coding and modulation):

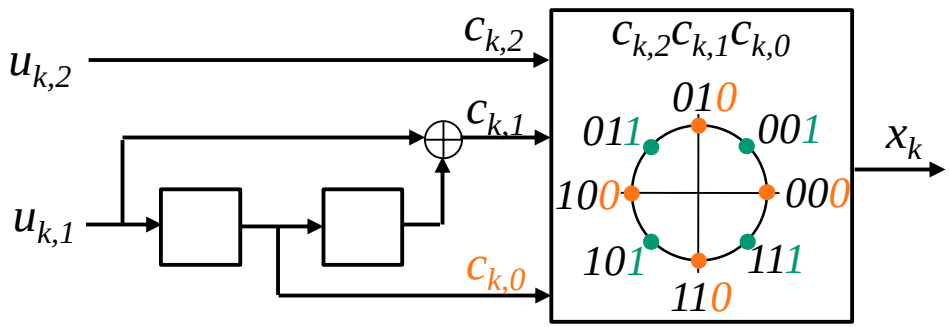


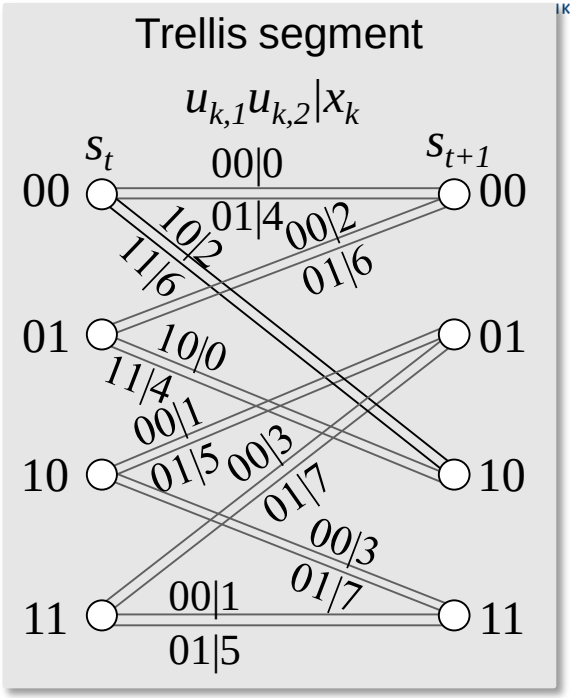
Euclidean distance:

$\Delta = \sqrt{2-\sqrt{2}}$

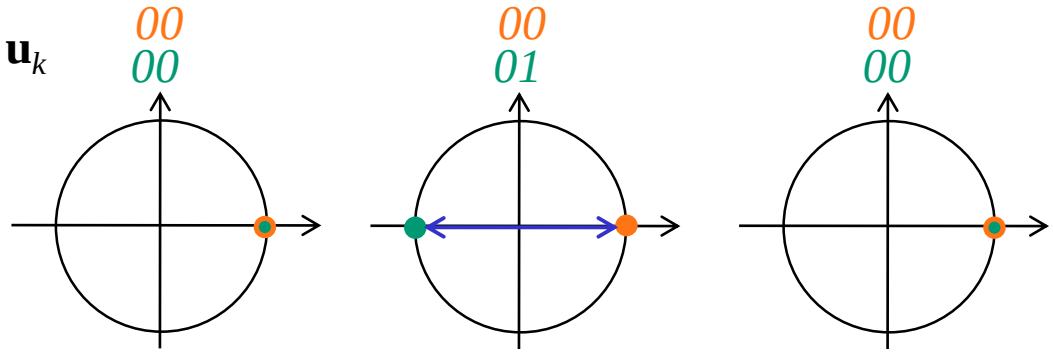
Trellis Coded Modulation (TCM)

Set partitioning





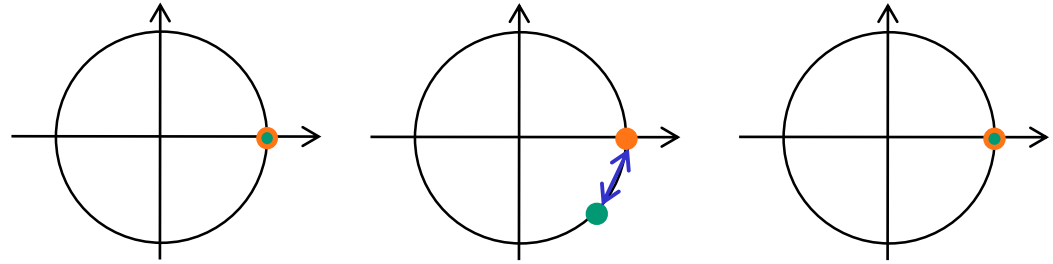
Transmit sequences:



Euclidean distance:

$\Delta = 2$

Transmit sequences with low Euclidean distance (independent coding and modulation):

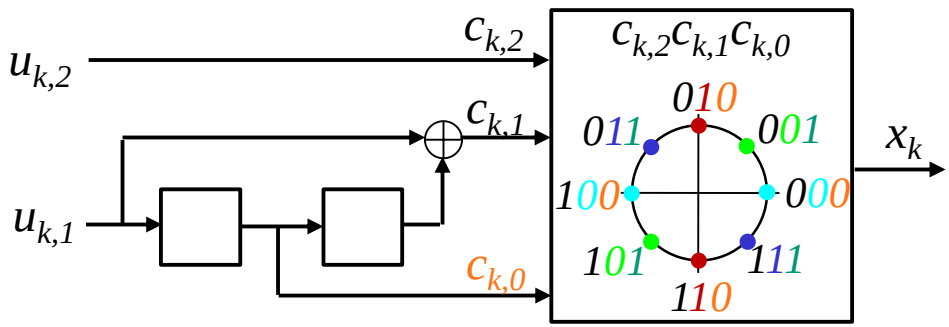


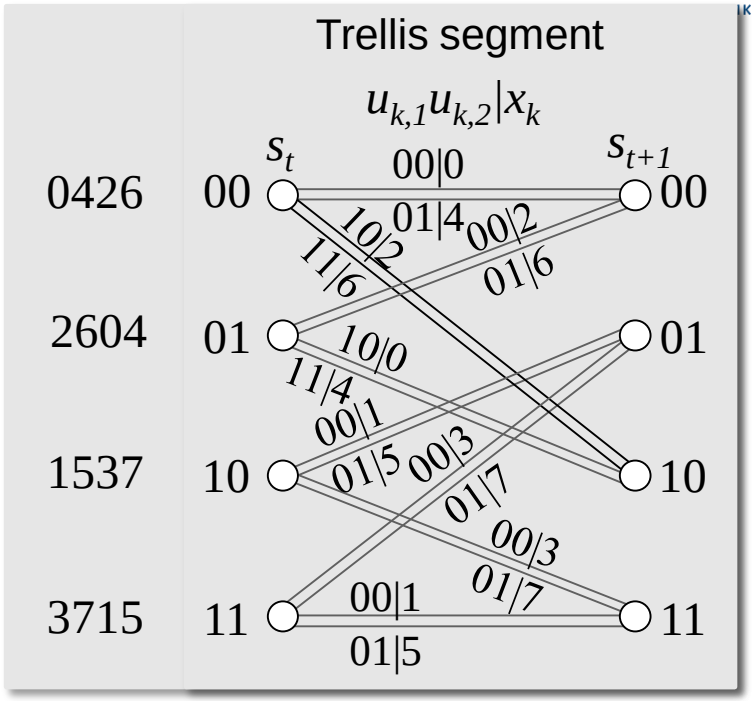
Euclidean distance:

$\Delta = \sqrt{2-\sqrt{2}}$

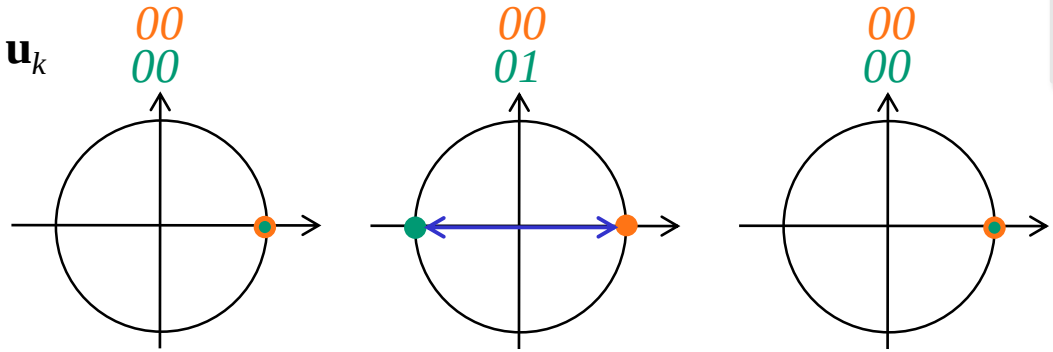
Trellis Coded Modulation (TCM)

Set partitioning





Transmit sequences:



Euclidean distance:

$\Delta = 2$

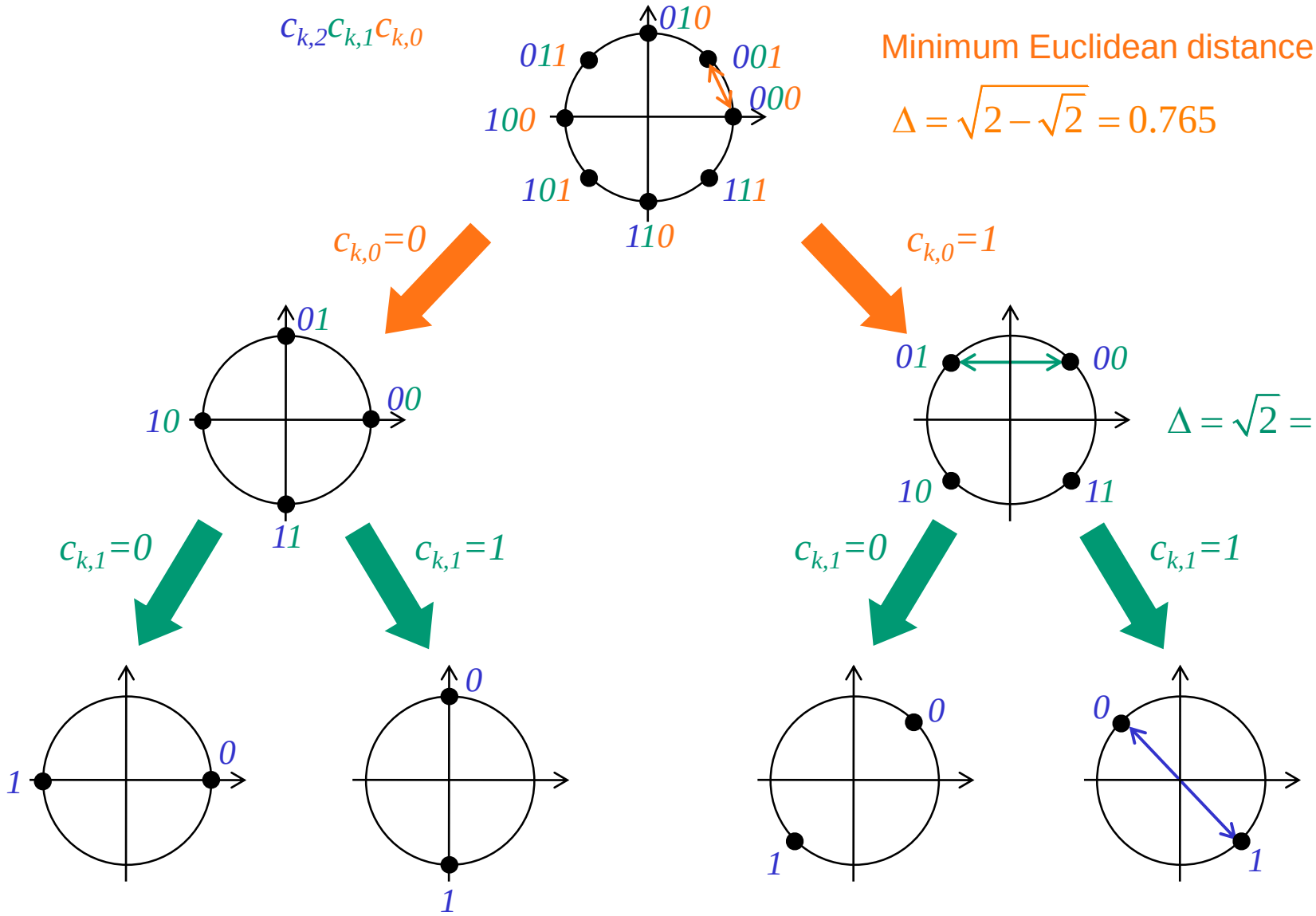
Set partitioning:

Protected (coded) bits $c_{k,0}$ and $c_{k,1}$ determine sets.

Euclidean distance between different sets is small, but Euclidean distance within a set is increased by set partitioning.

Uncoded bits $c_{k,2}$ see BPSK constellation (maximum Euclidean distance).

Set Partitioning for 8-PSK



Minimum Euclidean distance:

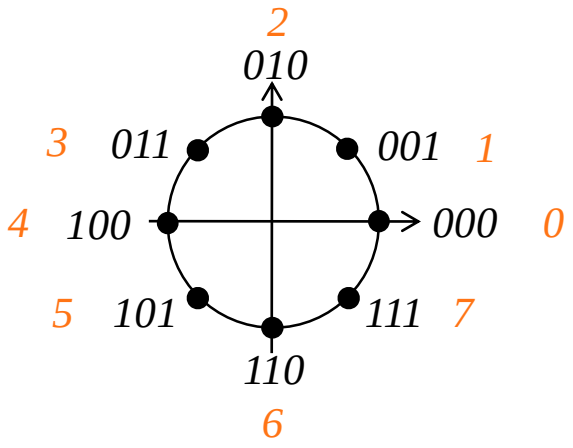
$$\Delta = \sqrt{2-\sqrt{2}} = 0.765$$

$$\Delta = \sqrt{2} = 1.414$$

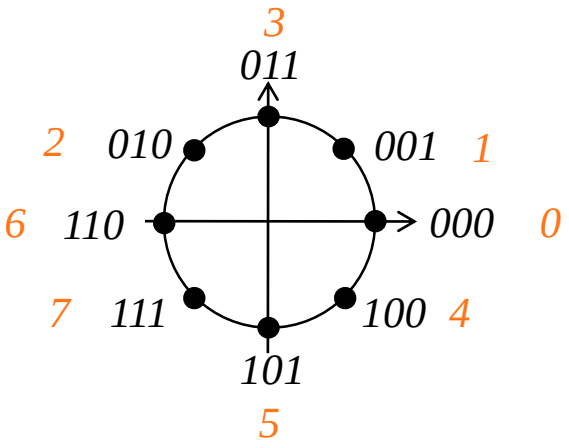
$$\Delta = 2$$

8-PSK Mappings

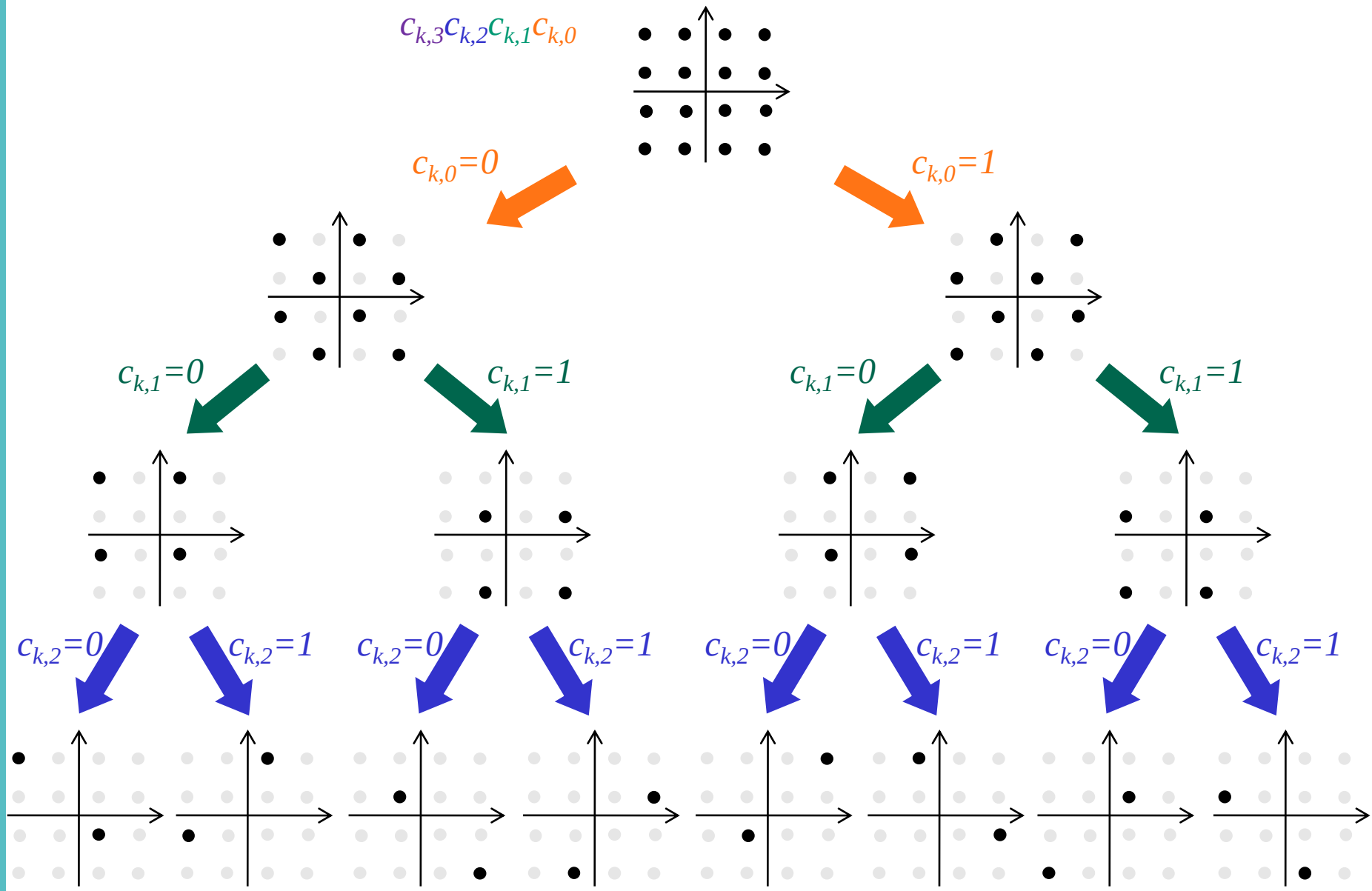
Natural mapping



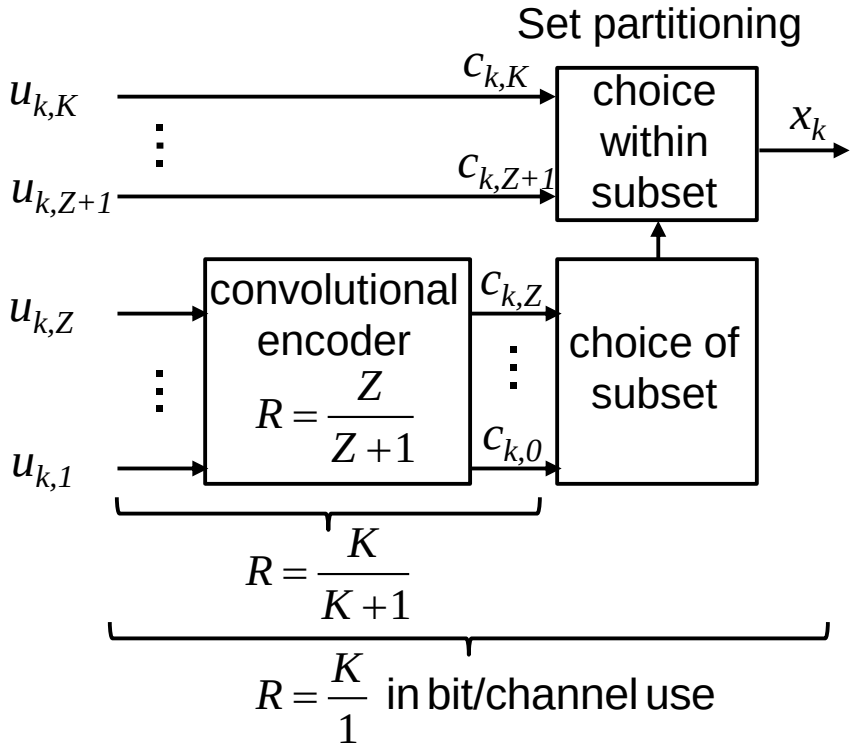
Gray mapping



Set Partitioning for 16-QAM



Trellis Coded Modulation: Ungerböck Codes



$Z=1$: Only 1 info bit is encoded.

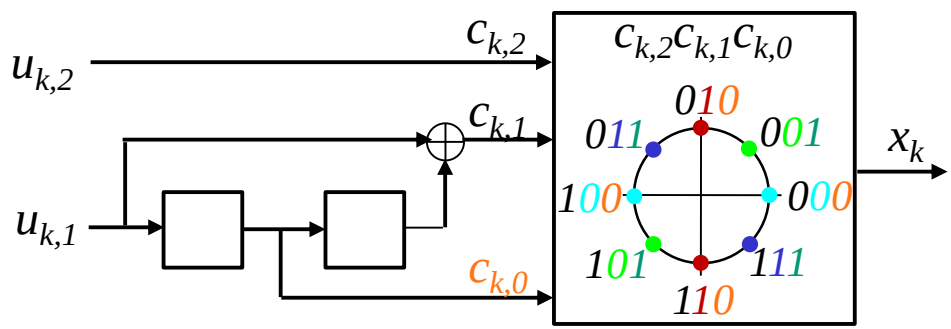
$Z=K$: All info bits are encoded.

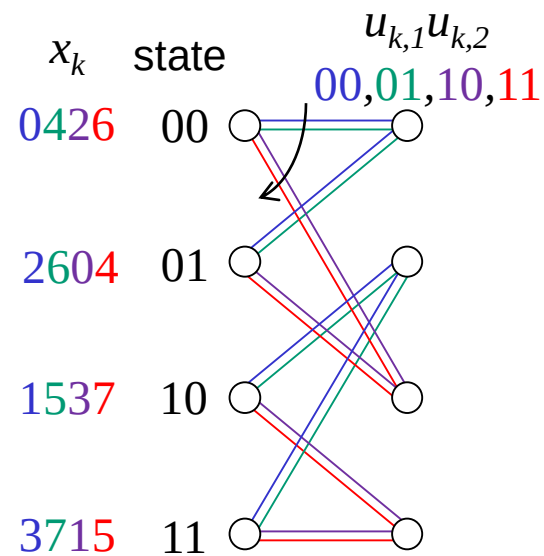
→ chosen subset consists of a single constellation point.

Large Z:

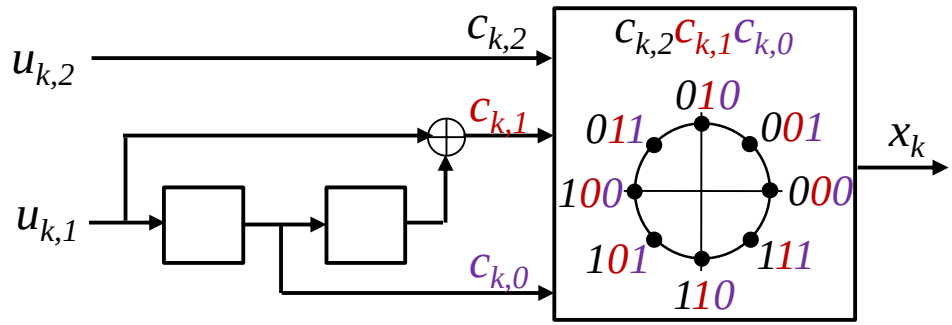
- Chosen subset is small with large Euclidean distance +
- Large number of subsets -
- Weaker convolutional code ($R\uparrow$) for protection of subset choice. -

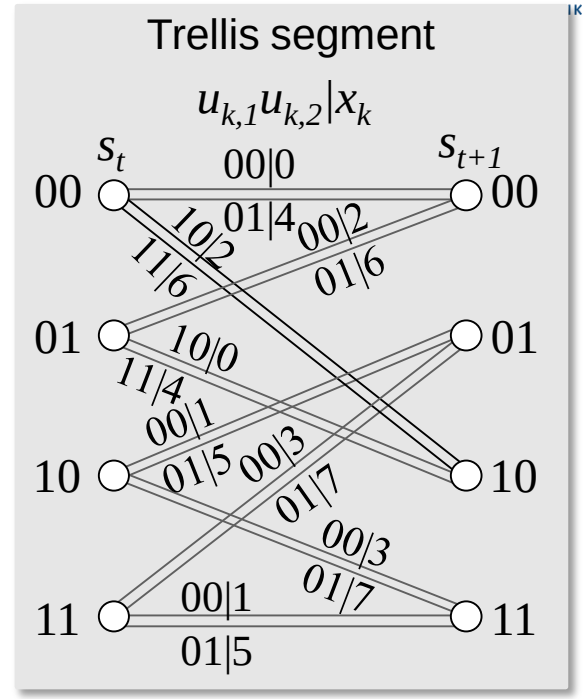
Example: $R=2$, $M=2$, 8-PSK, $K=2$, $Z=1$





Trellis Coded Modulation (TCM): Example



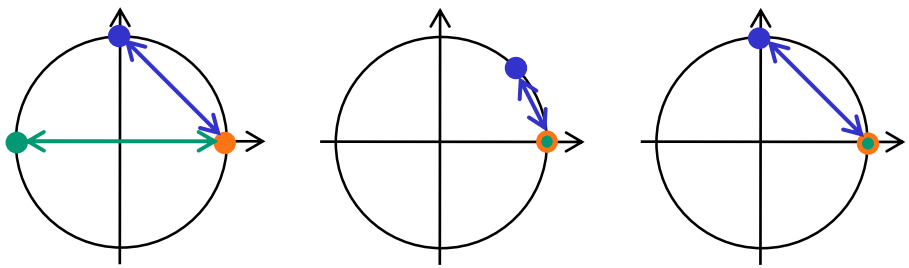


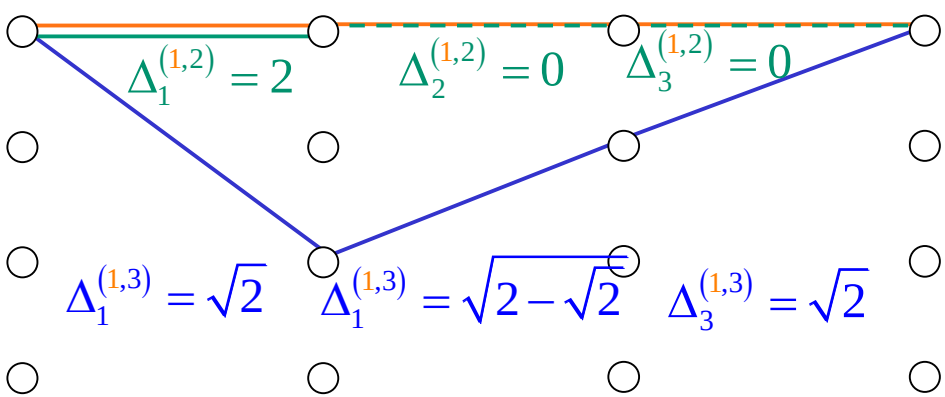
Transmit sequences:

$\mathbf{u}_k$:

00	00	00
01	00	00
10	00	00

x_k:



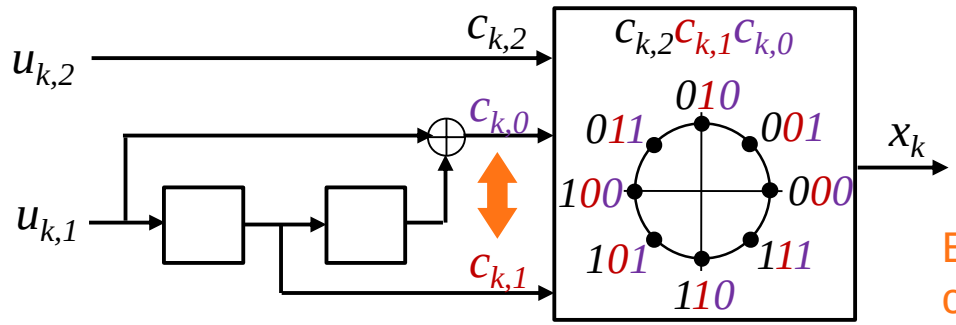


Free Euclidean distance: $\Delta_f = 2$

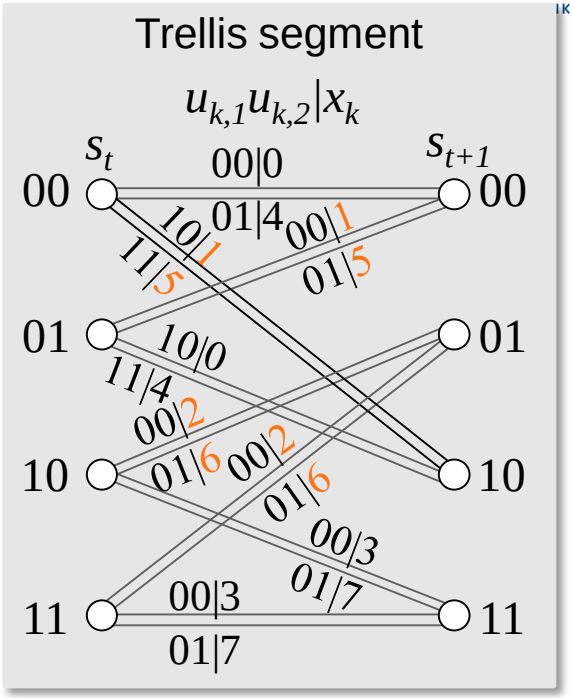
$$\Delta^{(1,2)} = \sqrt{\sum_k \left(\Delta_k^{(1,2)}\right)^2} = 2$$

$$\Delta^{(1,3)} = \sqrt{\sum_k \left(\Delta_k^{(1,3)}\right)^2} = 2.14$$

Trellis Coded Modulation (TCM): Example



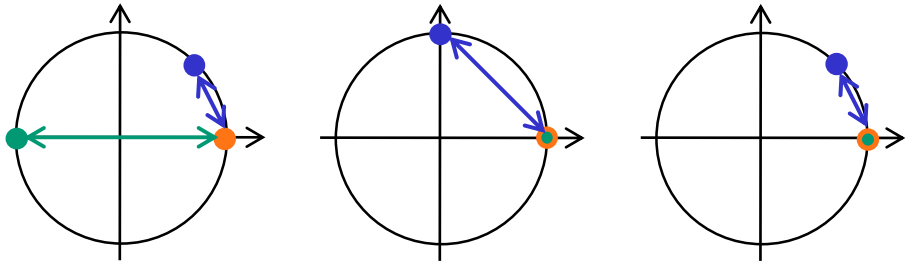
Exchange code bits

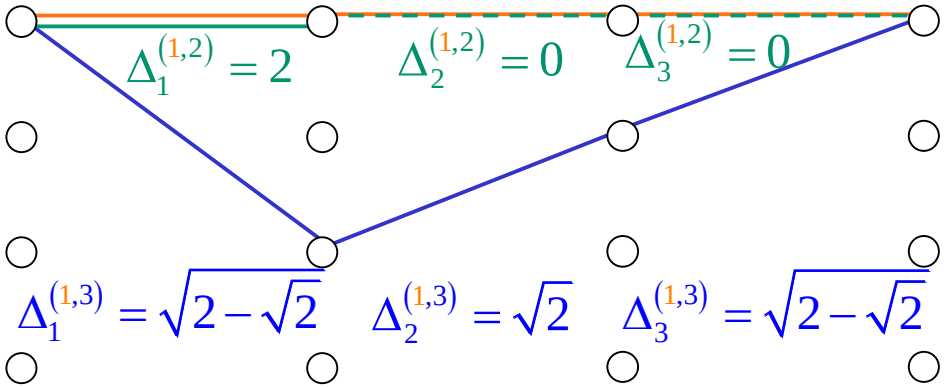


Transmit sequences:

$\mathbf{u}_k$:	00	00	00
	01	00	00
	10	00	00

x_k:

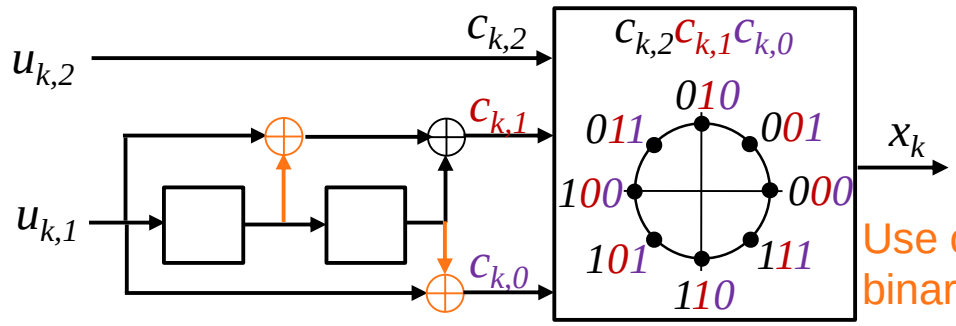




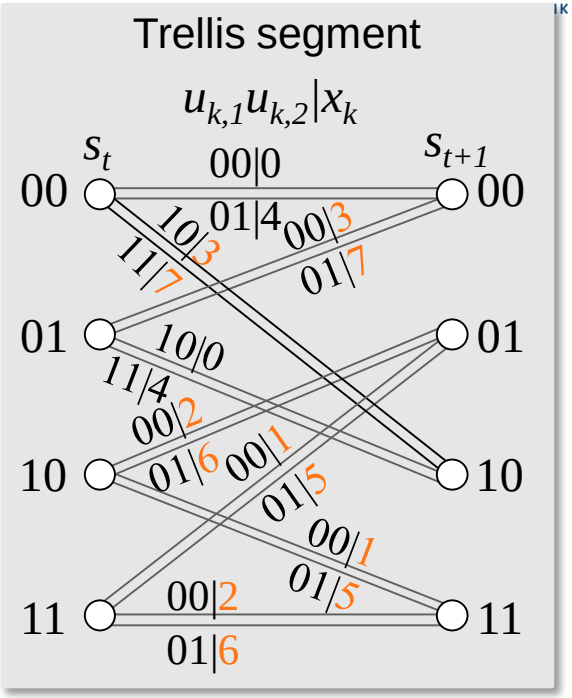
$$\Delta^{(1,2)} = \sqrt{\sum_k \left(\Delta_k^{(1,2)}\right)^2} = 2$$

$$\Delta^{(1,3)} = \sqrt{\sum_k \left(\Delta_k^{(1,3)}\right)^2} = 1.78 < 2$$

Trellis Coded Modulation (TCM): Example



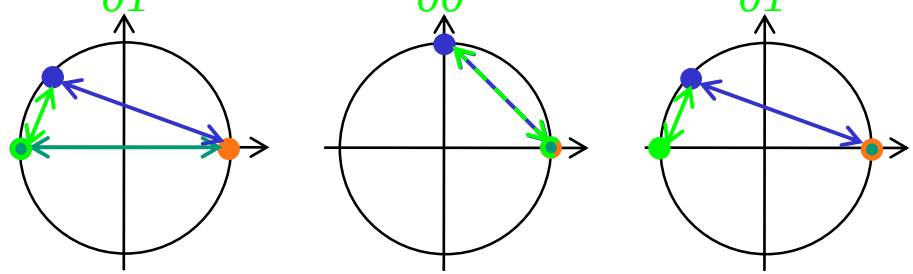
Use optimum binary convolutional code.

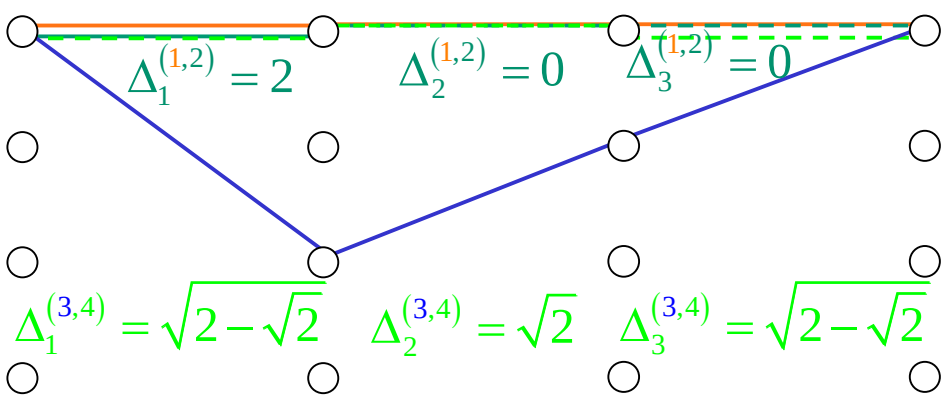


Transmit sequences:

$\mathbf{u}_k$: 00 01 10 01 | 00 00 00 00 | 00 00 00 01

x_k:



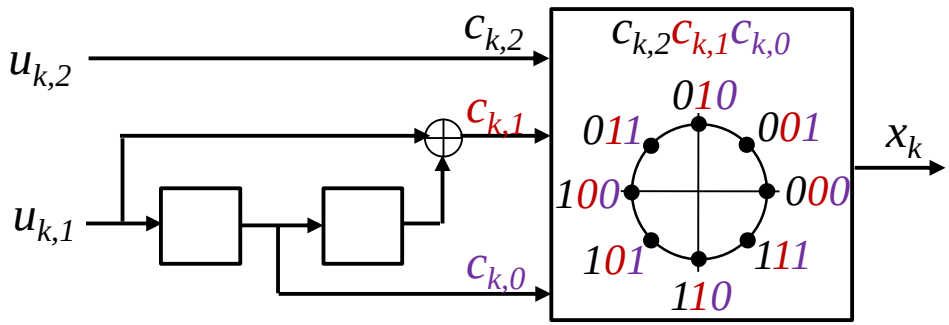


Optimum binary convolutional codes are not necessarily optimum for TCM.

$$\Delta^{(3,4)} = \sqrt{\sum_k \left(\Delta_k^{(3,4)}\right)^2} = 1.78 < 2 \quad ↯$$

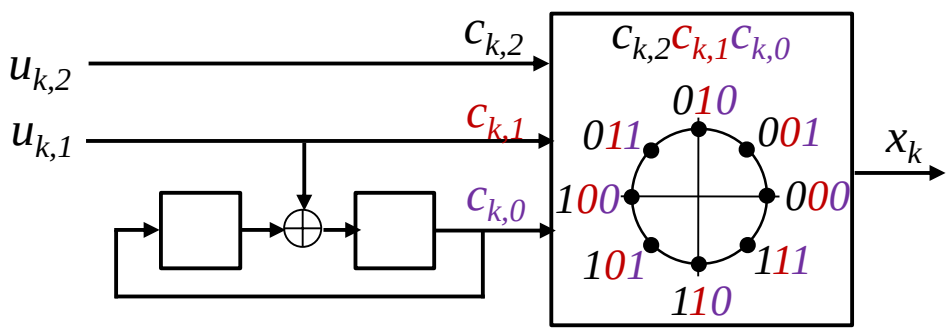
Trellis Coded Modulation (TCM): Non-Systematic and Recursive Systematic Encoder

Non-systematic non-recursive encoder:



$$\underbrace{\begin{bmatrix} c_0(D) \\ c_1(D) \\ c_2(D) \end{bmatrix}}_{\mathbf{c}(D)} = \underbrace{\begin{bmatrix} D & 0 \\ 1+D^2 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{G}(D)} \underbrace{\begin{bmatrix} u_1(D) \\ u_2(D) \end{bmatrix}}_{\mathbf{u}(D)}$$

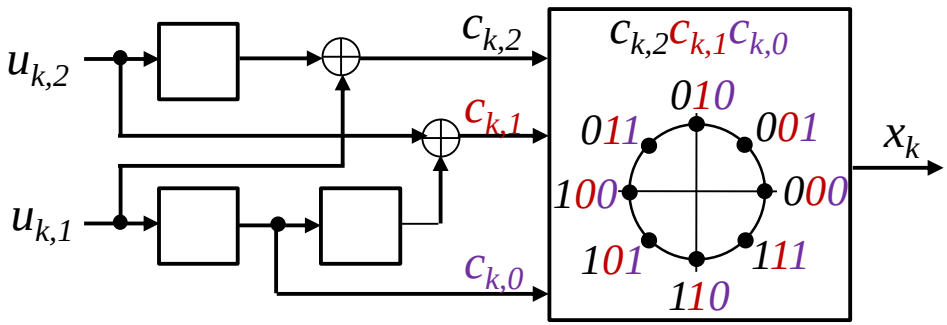
Recursive systematic encoder:



$$\underbrace{\begin{bmatrix} c_0(D) \\ c_1(D) \\ c_2(D) \end{bmatrix}}_{\mathbf{c}(D)} = \underbrace{\begin{bmatrix} \frac{D}{1+D^2} & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{G}(D)} \underbrace{\begin{bmatrix} u_1(D) \\ u_2(D) \end{bmatrix}}_{\mathbf{u}(D)}$$

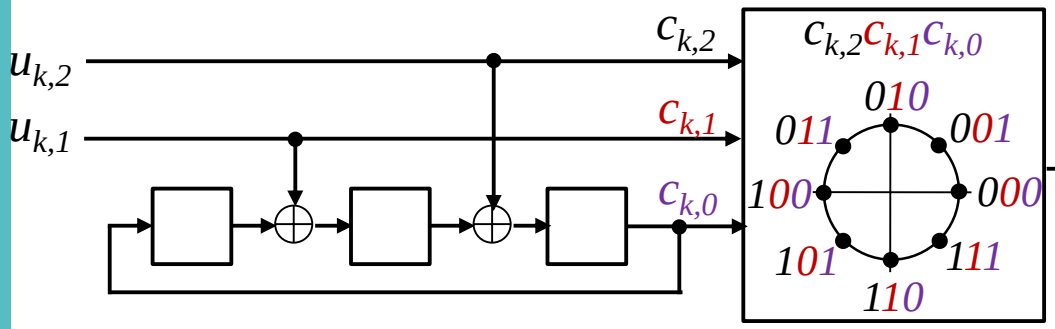
Trellis Coded Modulation (TCM): Non-Systematic and Recursive Systematic Encoder

Non-systematic non-recursive encoder:



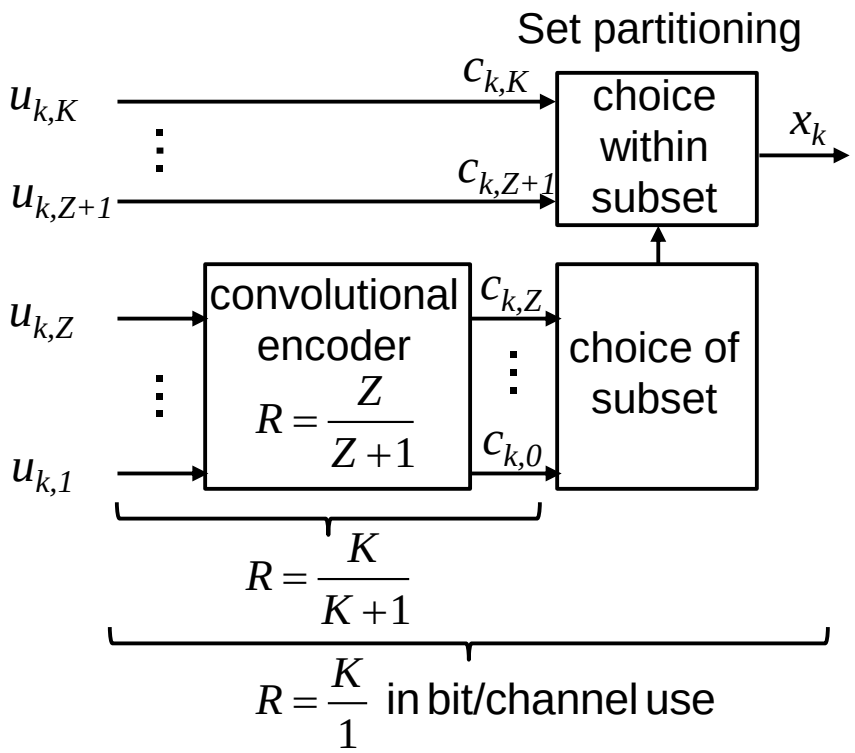
$$\underbrace{\begin{bmatrix} c_0(D) \\ c_1(D) \\ c_2(D) \end{bmatrix}}_{\mathbf{c}(D)} = \underbrace{\begin{bmatrix} D & 0 \\ D^2 & 1 \\ 1 & D \end{bmatrix}}_{\mathbf{G}(D)} \underbrace{\begin{bmatrix} u_1(D) \\ u_2(D) \end{bmatrix}}_{\mathbf{u}(D)}$$

Recursive systematic encoder:

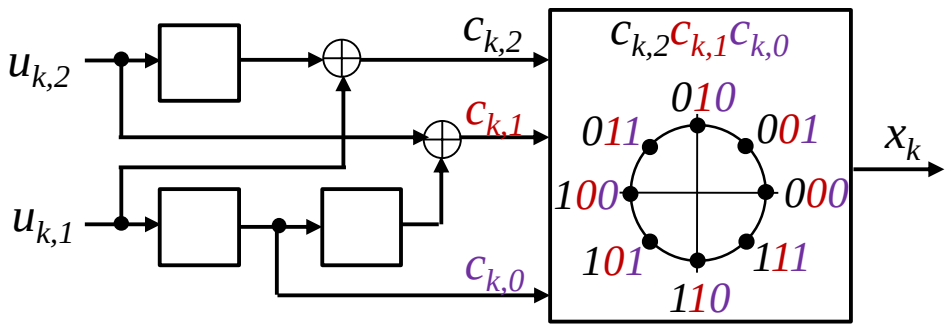


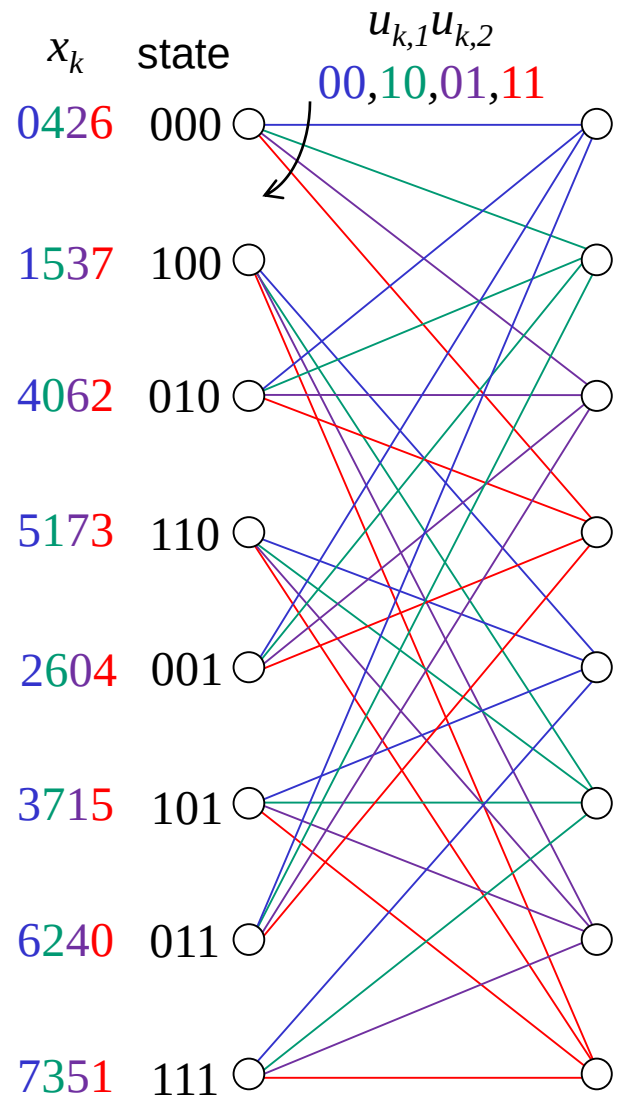
$$\underbrace{\begin{bmatrix} c_0(D) \\ c_1(D) \\ c_2(D) \end{bmatrix}}_{\mathbf{c}(D)} = \underbrace{\begin{bmatrix} \frac{D^2}{1+D^3} & \frac{D}{1+D^3} \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{G}(D)} \underbrace{\begin{bmatrix} u_1(D) \\ u_2(D) \end{bmatrix}}_{\mathbf{u}(D)}$$

Trellis Coded Modulation: Ungerböck Codes

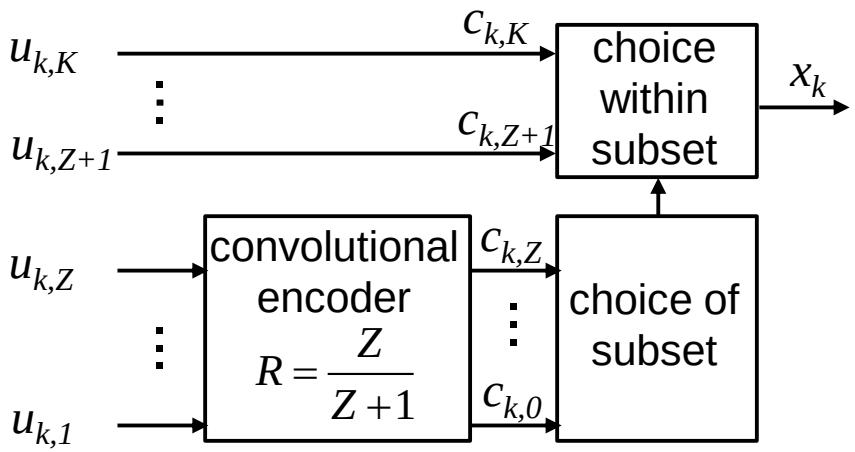


Example: $R=2$, $M=3$, 8-PSK,
$K=2$, $Z=2$





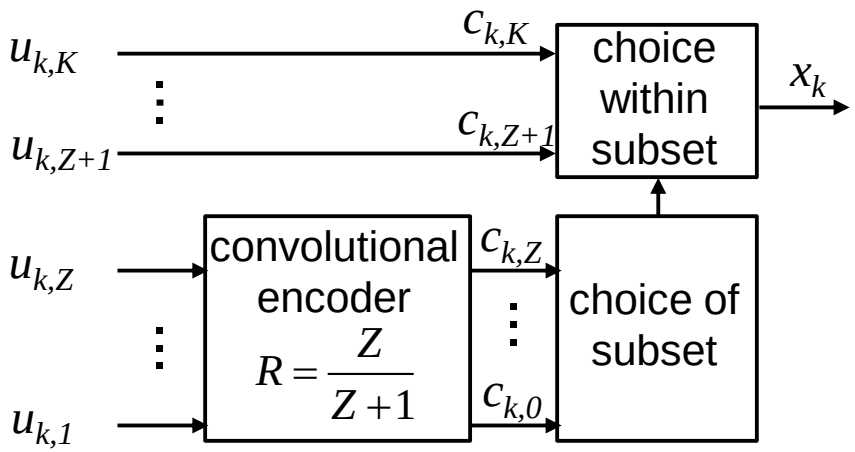
Asymptotic Coding Gain of Optimum 8-PSK Ungerböck Codes in AWGN Channel



Coding gain of 8-PSK TCM over uncoded QPSK

Memory M	Z	asymptotic coding gain in dB	coding gain in dB at $P_b=10^{-5}$	Δ_f
2	1	3.01	2.4	2
3	2	3.60	2.8	2.1415
4	2	4.13	3.0	2.2742
5	2	4.59	3.3	2.3996
6	2	5.01	3.6	2.5185
7	2	5.17		2.5663
8	2	5.75		2.7414

Asymptotic Coding Gain of Optimum 16-QAM Ungerböck Codes in AWGN Channel

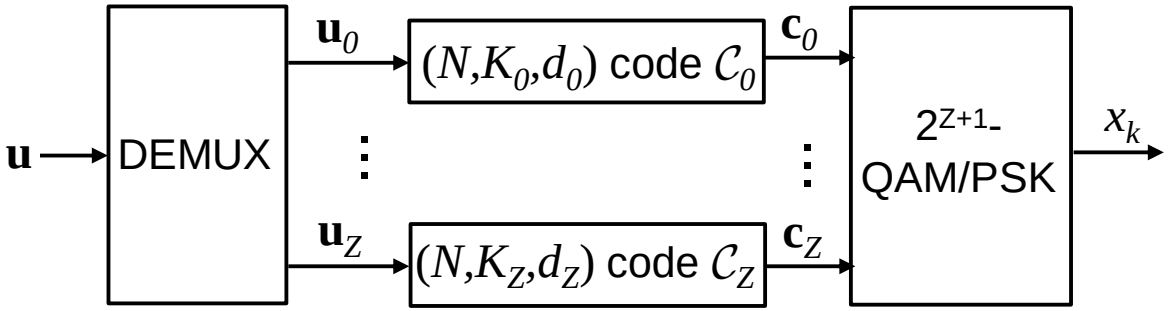


Coding gain of 16-QAM TCM over uncoded 8-PSK

Memory M	Z	asymptotic coding gain in dB
2	1	4.36
3	2	5.33
4	2	6.12
5	2	6.12
6	2	6.79
7	2	7.37
8	2	7.37

Multi-Level Coding

Imai, Hirakawa 1977



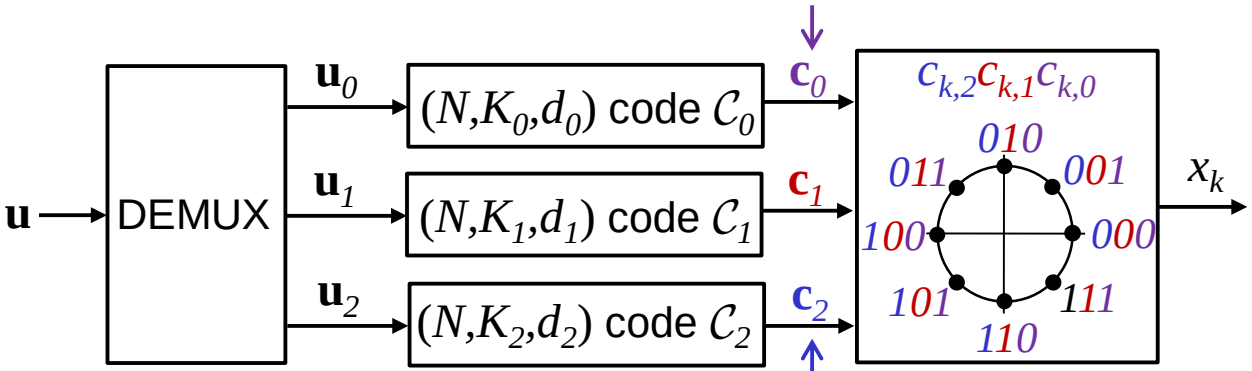
increasing code rate

$$R_z = \frac{K_z}{N}$$

separate constituent code for
each bit of a constellation point

Example: 8-PSK

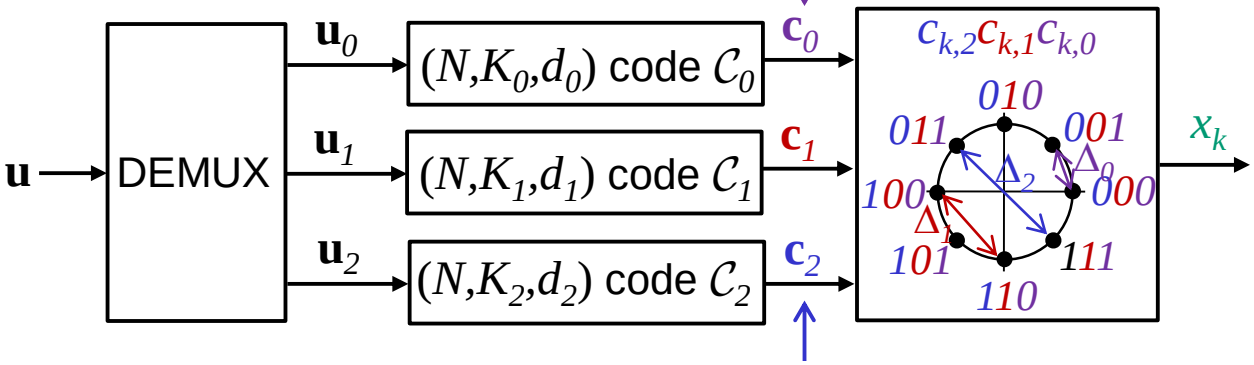
strongest code
→ choice between constellation points at minimum Euclidean distance



weakest code
→ choice between constellation points at maximum Euclidean distance

Multi-Level Coding

strongest code
→ choice between constellation point at minimum Euclidean distance



increasing code rate

$$R_z = \frac{K_z}{N}$$

weakest code
→ choice between constellation point at maximum Euclidean distance

code bit matrix:

$$\begin{bmatrix} c_{0,0} & \cdots & c_{N-1,0} \\ c_{0,1} & \cdots & c_{N-1,1} \\ c_{0,2} & \cdots & c_{N-1,2} \end{bmatrix}$$

x_0 $\quad$ x_{N-1}

Two codewords of code $\mathcal{C}_z$ differ in at least d_z positions.

→ transmit sequences have Euclidean distance

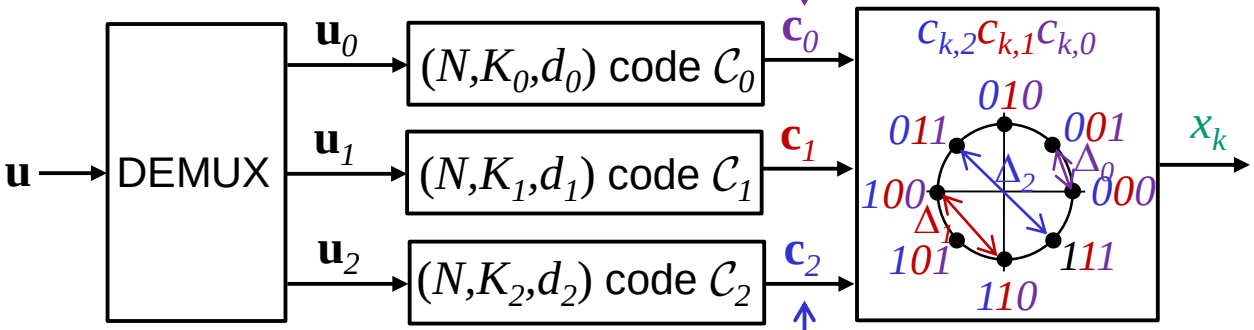
$$\Delta_z \sqrt{d_z}$$

→ Minimum Euclidean distance of multi-level code:

$$\Delta_{\min} = \min_z \left\{ \Delta_z \sqrt{d_z} \right\}$$

Multi-Level Coding

strongest code
→ choice between constellation point at minimum Euclidean distance



increasing code rate

$$R_z = \frac{K_z}{N}$$

weakest code
→ choice between constellation point at maximum Euclidean distance

code bit matrix:

$$\begin{bmatrix} c_{0,0} & \cdots & c_{N-1,0} \\ c_{0,1} & \cdots & c_{N-1,1} \\ c_{0,2} & \cdots & c_{N-1,2} \end{bmatrix}$$

x_0 $\quad$ x_{N-1}

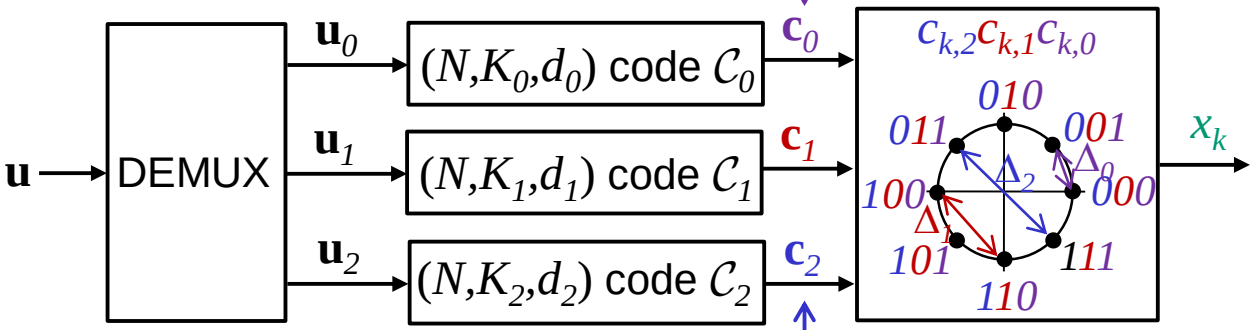
Soft decision maximum likelihood (ML) decoding:

Find codeword matrix with minimum Euclidean distance to received matrix.

→ High decoding complexity for large code matrices.

Multi-Level Coding and Multi-Level Decoding

strongest code
→ choice between constellation point at minimum Euclidean distance

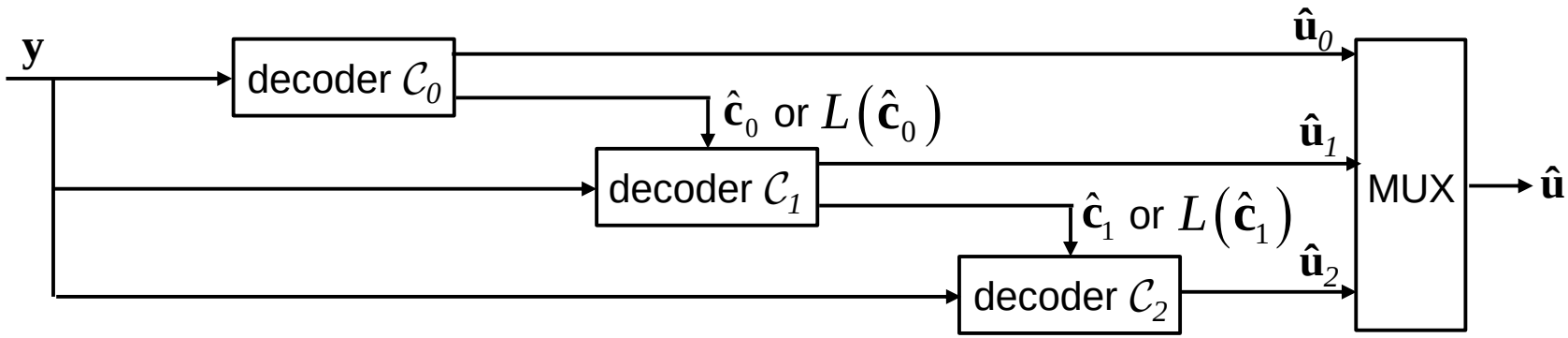


weakest code
→ choice between constellation point at maximum Euclidean distance

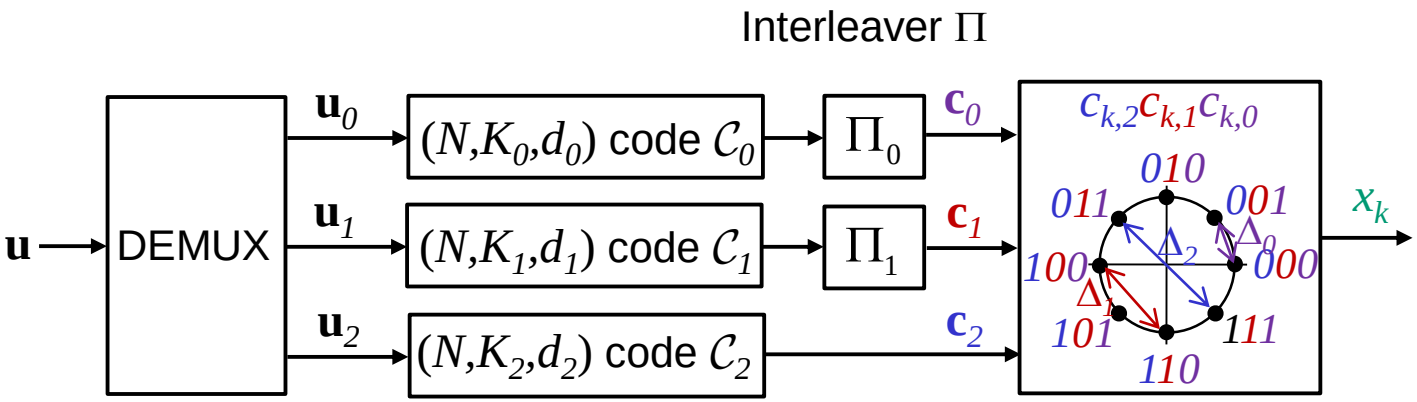
increasing code rate

$$R_z = \frac{K_z}{N}$$

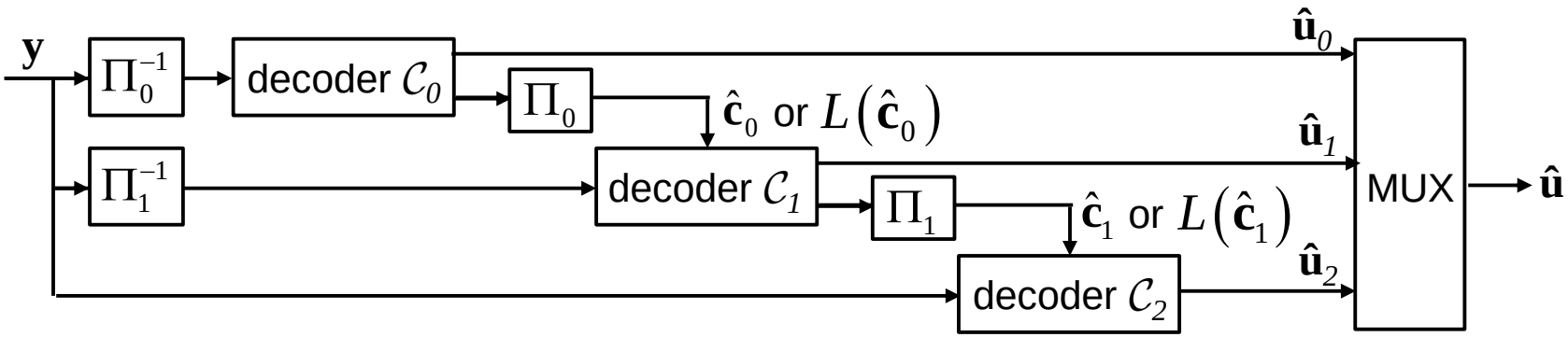
Suboptimum multi-level decoding:



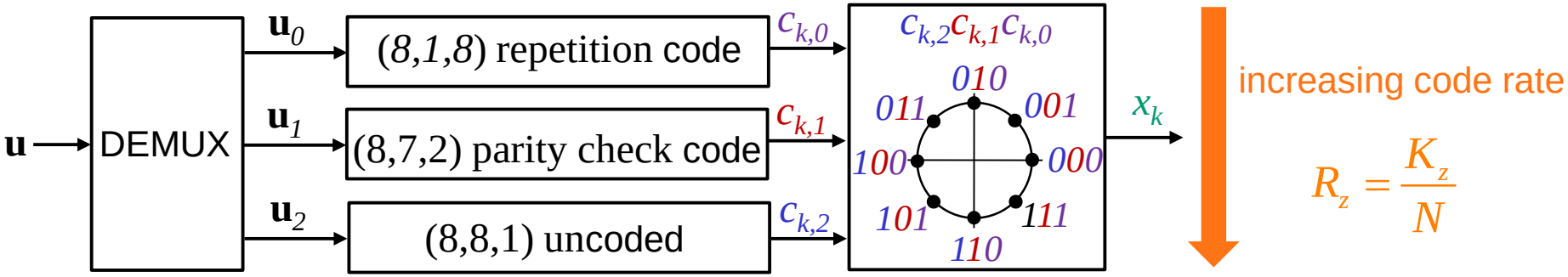
Multi-Level Coding and Multi-Level Decoding



Suboptimum multi-level decoding:



Multi-Level Coding: Example



$$R_z = \frac{K_z}{N}$$

Achieves asymptotic coding gain of 3.01 dB with very simple constituent codes.

Multi-Level Coding

In mult-level coding with a q-ary modulation scheme, the information word of length K bits is demultiplexed into $\log_2 q$ streams. Each of those streams is encoded by a separate encoder, where the codeword length N of all $\log_2 q$ constituent codes is equal. However, the infoword lengths K_i may be different, i.e. the constituent codes have different code rate.
The codebit of stream (level) i at time k determines the i-th bit of the QAM transmit symbol x_k. The code rate increases with increasing stream index i. The last stream may even be left uncoded. The mapping of bits to QAM symbols is defined such that the codebits of the strongest code (with lowest code rate) differentiate between constellation points at small Euclidean distance, whereas the codebits of the weakest code differentiate between constellation points at large Euclidean distance.
The optimum soft decision decoding rule would be a maximum likelihood decoder which chooses the overall codeword with minimum Euclidean distance to the received word. For already moderate infoword length, this becomes prohibitively complex since the number of candidate codewords increases exponentially with the info word length.
It is a major advantage of multi-level coding, that a suboptimum multi-level decoder with reduced complexity can be applied: The levels are decoded successively starting from the strongest code which differentiates subsets with minimum Euclidean distance in the QAM constellation. Under the assumption of error-free decoding, the next level can now be decoded, where only constellation points within the subset defined by the previous level have to be taken into consideration. Since the Euclidean distance within the subset is increased, a higher coding rate (weaker code) can be used. The decoder complexity of the multi-level decoder is determined by the sum of the constituent decoder complexities which is significantly less than the complexity of the overall ML decoder.

Multi-Level Coding

A problem of multi-level decoding is error propagation: A decoding error effects the decoding results in successive levels. Clearly, the multi-level decoder performance can be improved if soft outputs (in the form of log-likelihood ratios) rather than hard decisions are computed by earlier decoded levels and forwarded to the next decoder.

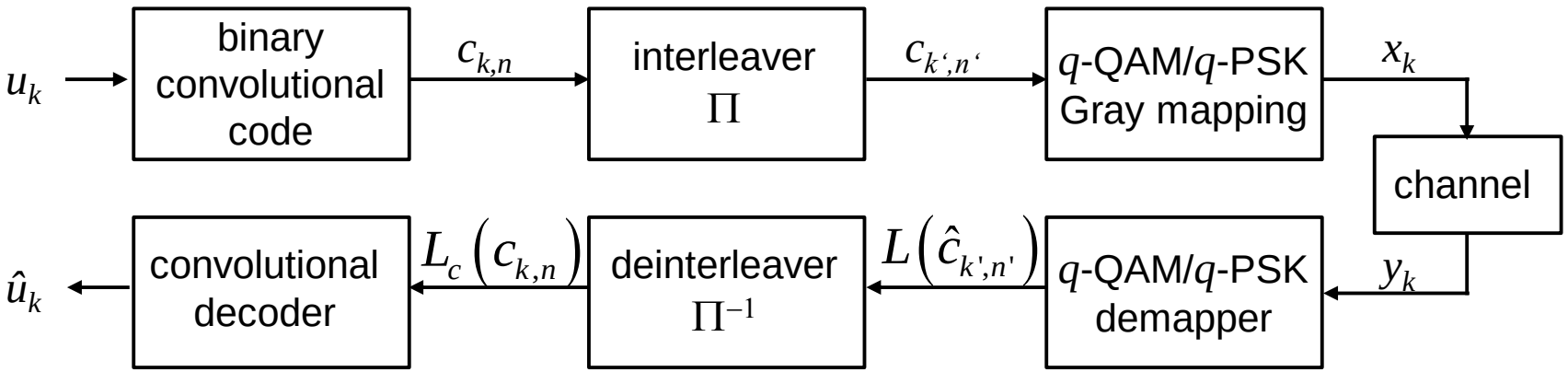
Another disadvantage is the decoding delay due to successive decoding of the constituent codes.

On the other hand, multi-level coding allows substantial coding gains with relatively simple constituent codes. In order to benefit from soft decision decoding in all decoding stages, codes which allow efficient soft decision decoding should be used. Convolutional codes can be efficiently soft decoded using the Viterbi algorithm. For many block codes, soft decision decoding is a challenge. However, there exist classes of block codes which allow efficient soft decision decoding, e.g. Reed-Muller codes.

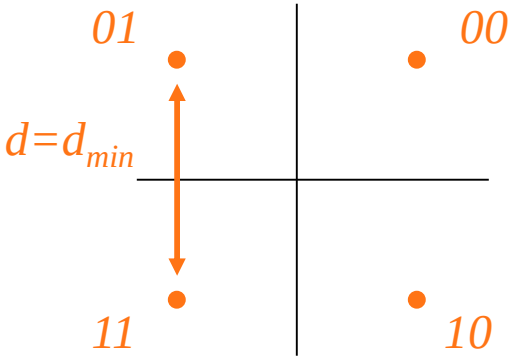
If convolutional codes are used as constituent codes, the Viterbi decoder produces burst errors in case of erroneous decoding. This may cause a too large number of adjacent errors which cannot be corrected by the successive decoding stage. A solution to this problem is to use an interleaver Π before mapping the codebits to QAM symbols. An interleaver simply changes the order of bits. Interleaving spreads the burst error over a wider range such that the successive Viterbi decoder sees an error pattern which he may be able to decode.

Multi-level coding can be considered as a kind of code concatenation. The principle of building a strong code out of several relatively simple constituent codes is a powerful concept which appears in various flavours in error control coding. The major advantage is that efficient decoding with manageable decoding complexity is enabled. A single powerful code has usually a prohibitively high decoding complexity. A concatenated coding scheme can be decoded with manageable complexity by decoding the constituent codes successively and taking the information of other constituent decoders into account in order to improve the decoding result.

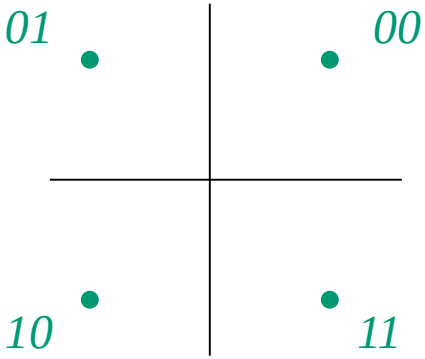
Bit-Interleaved Coded Modulation (BICM)



Gray Mapping

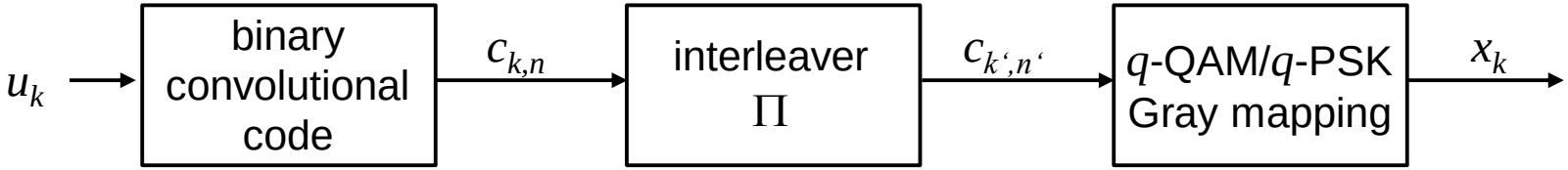


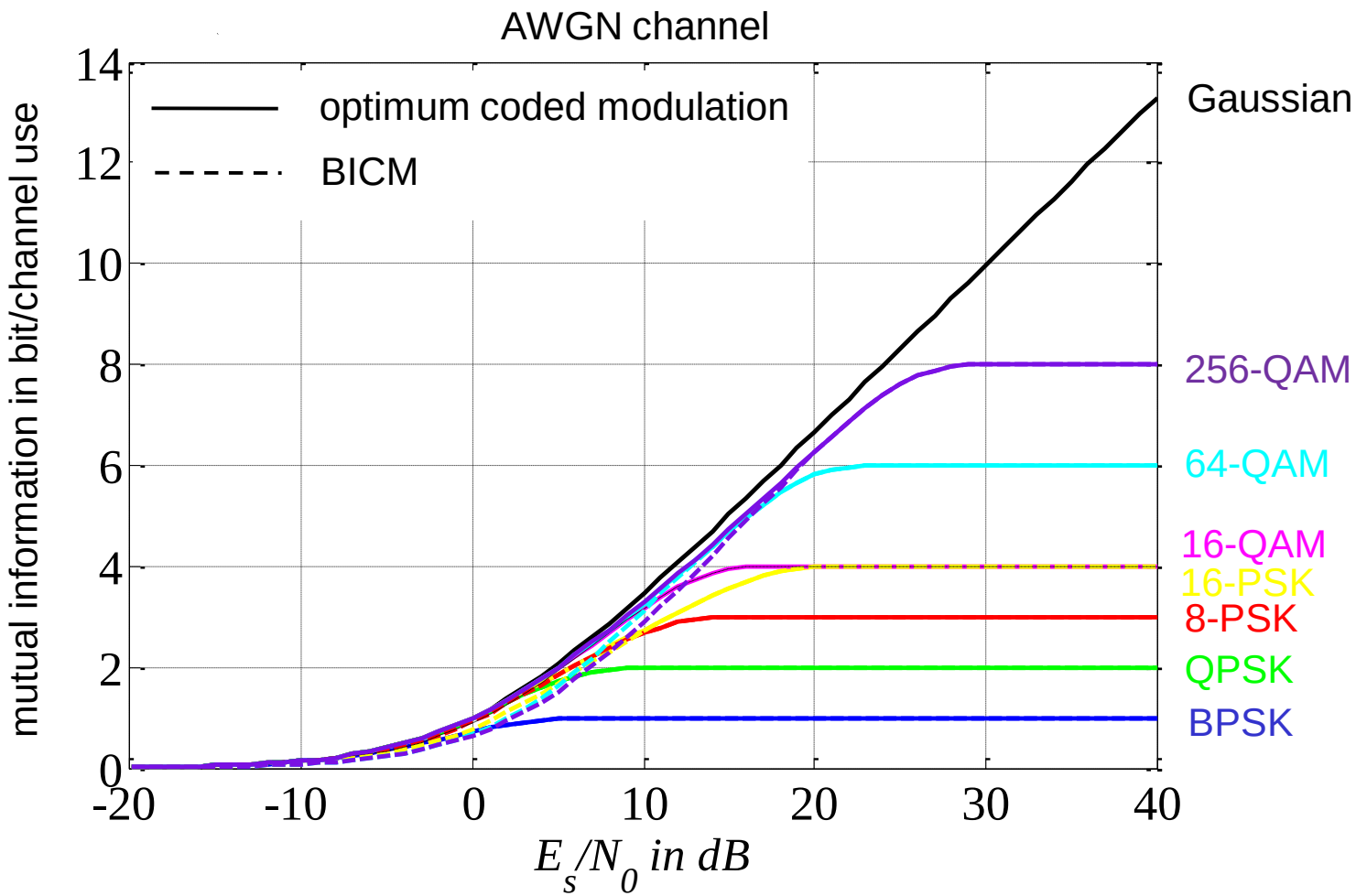
Natural Mapping



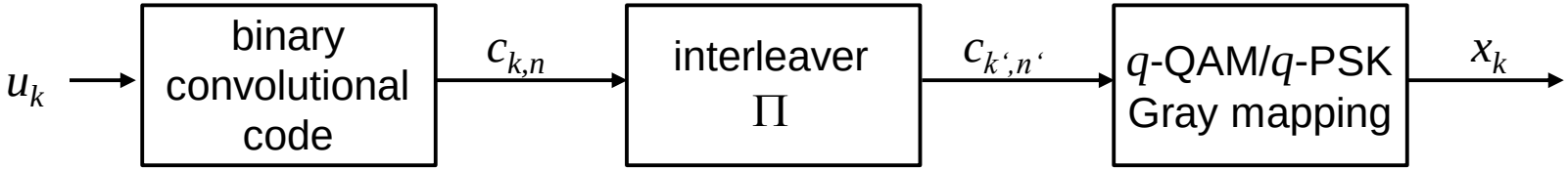
Constellation points with minimum Euclidean distance d_{min} differ in only 1 bit.

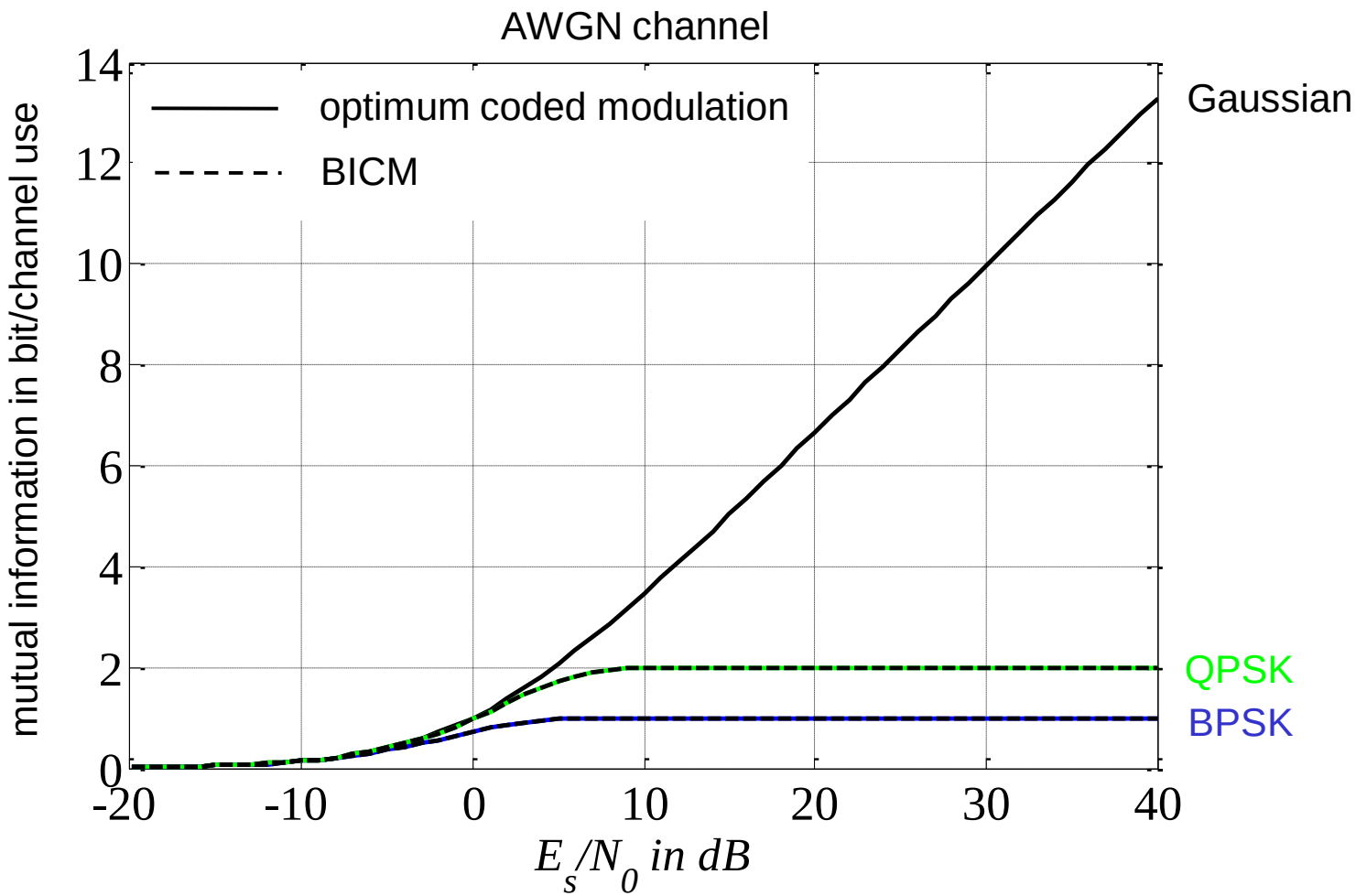
Bit-Interleaved Coded Modulation (BICM)



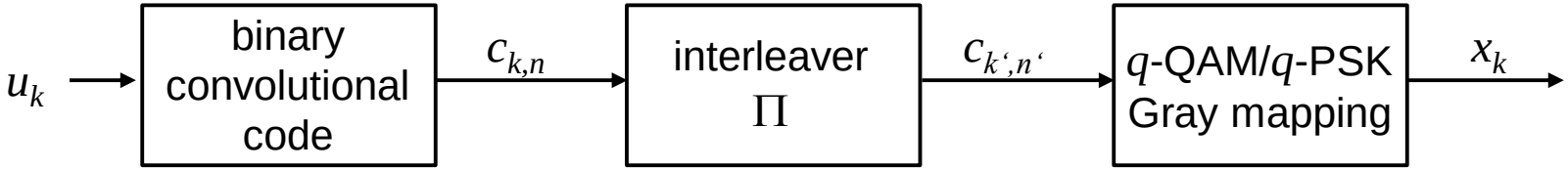


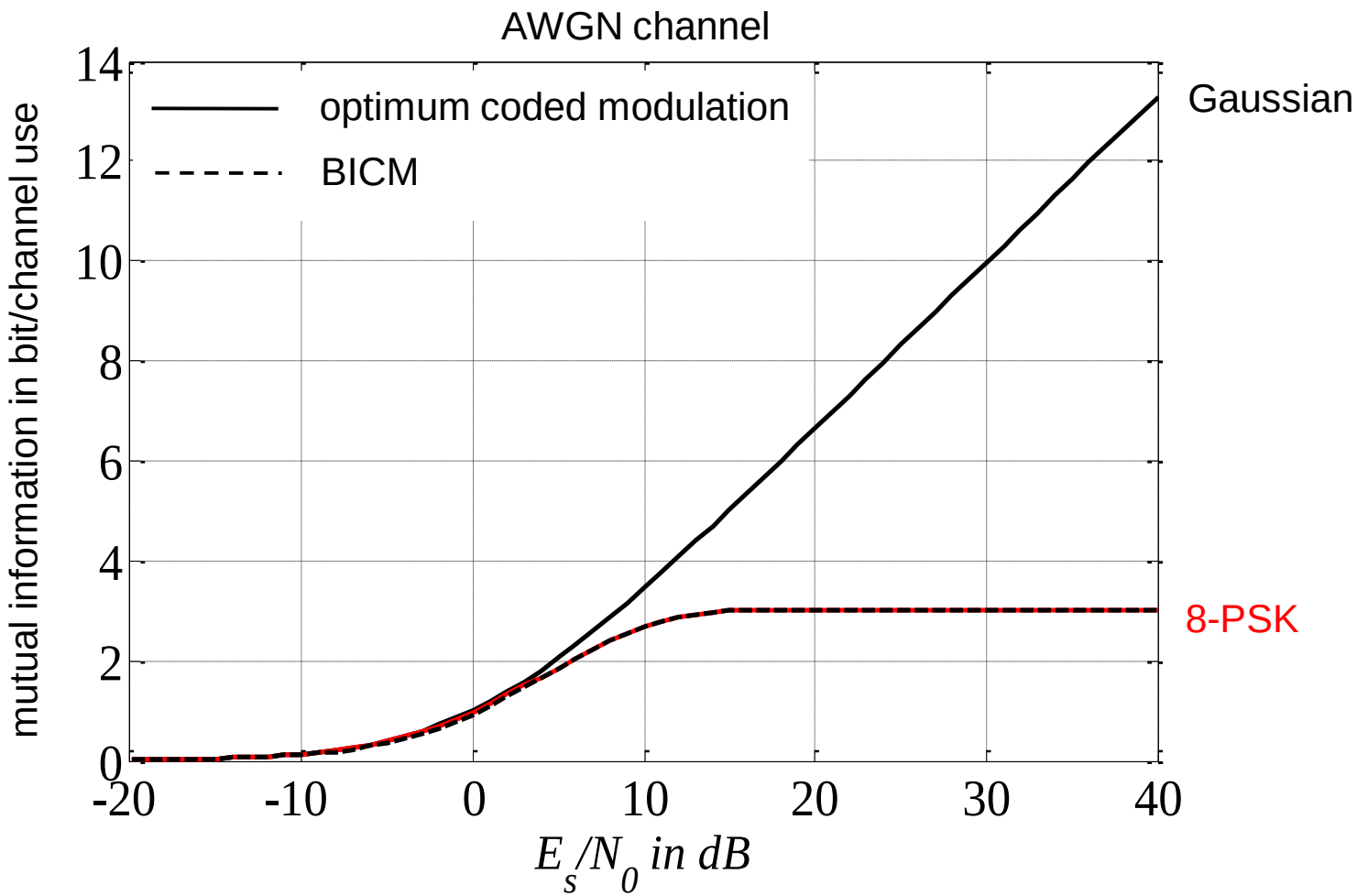
Bit-Interleaved Coded Modulation (BICM)



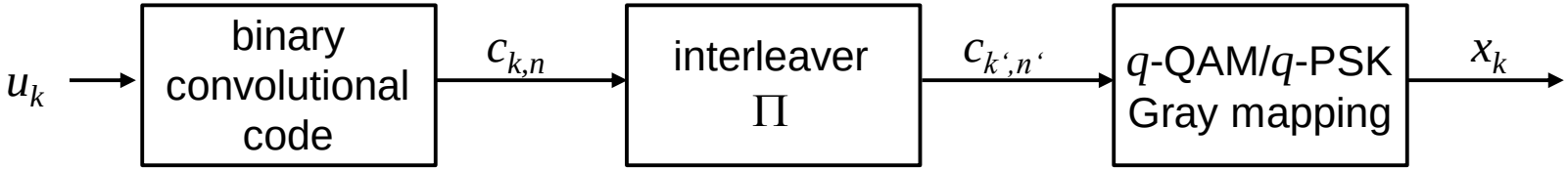


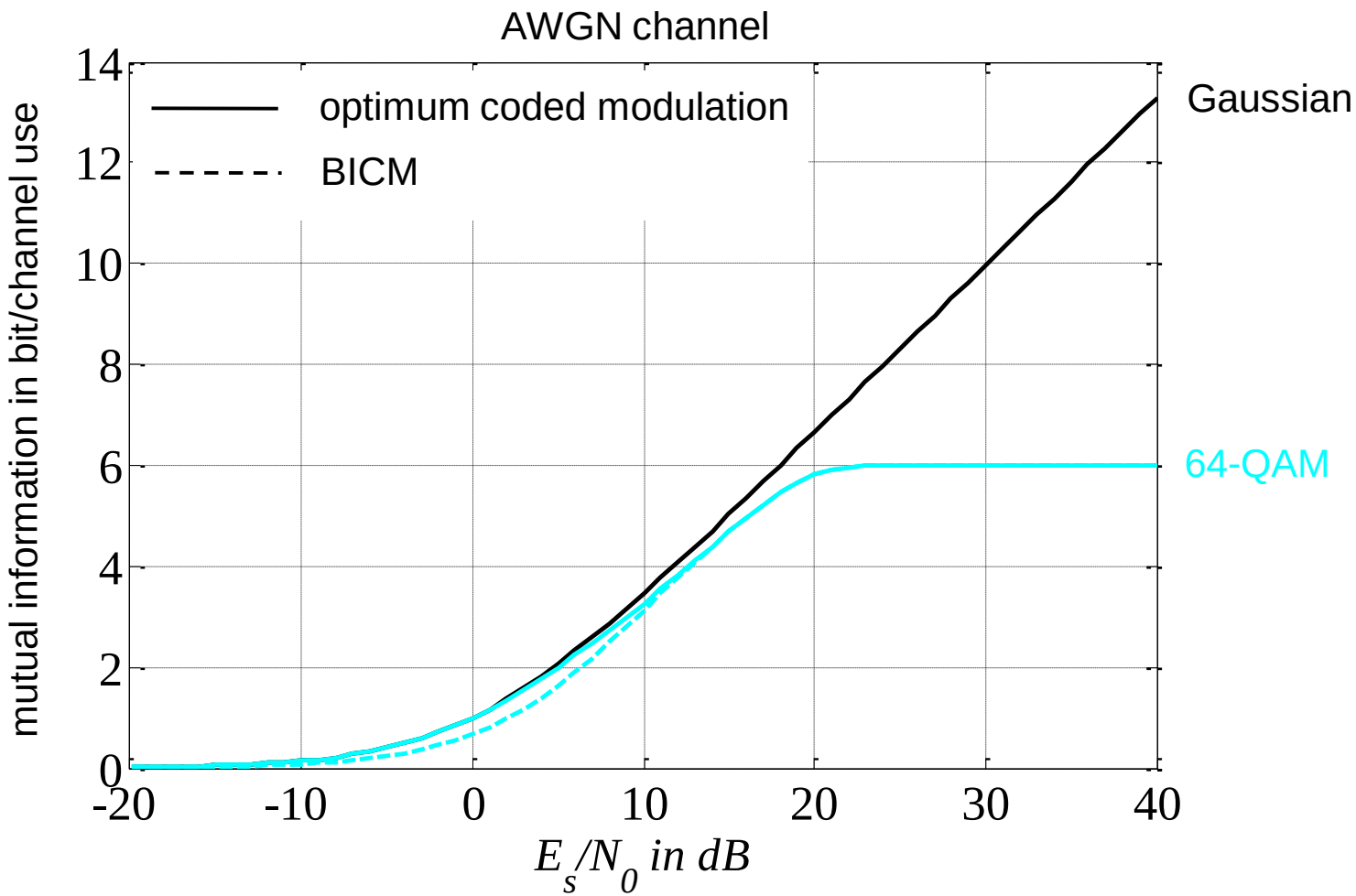
Bit-Interleaved Coded Modulation (BICM)



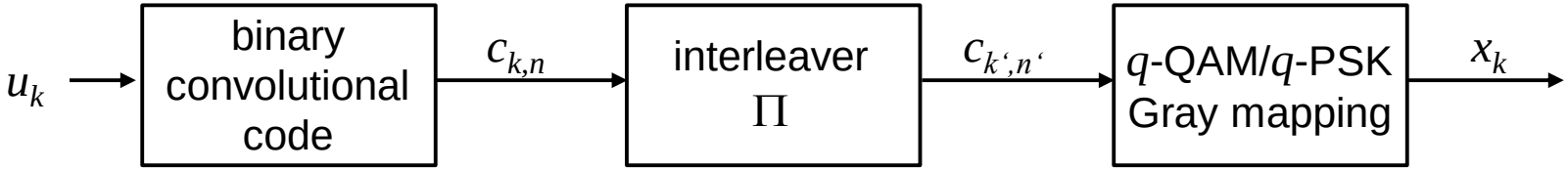


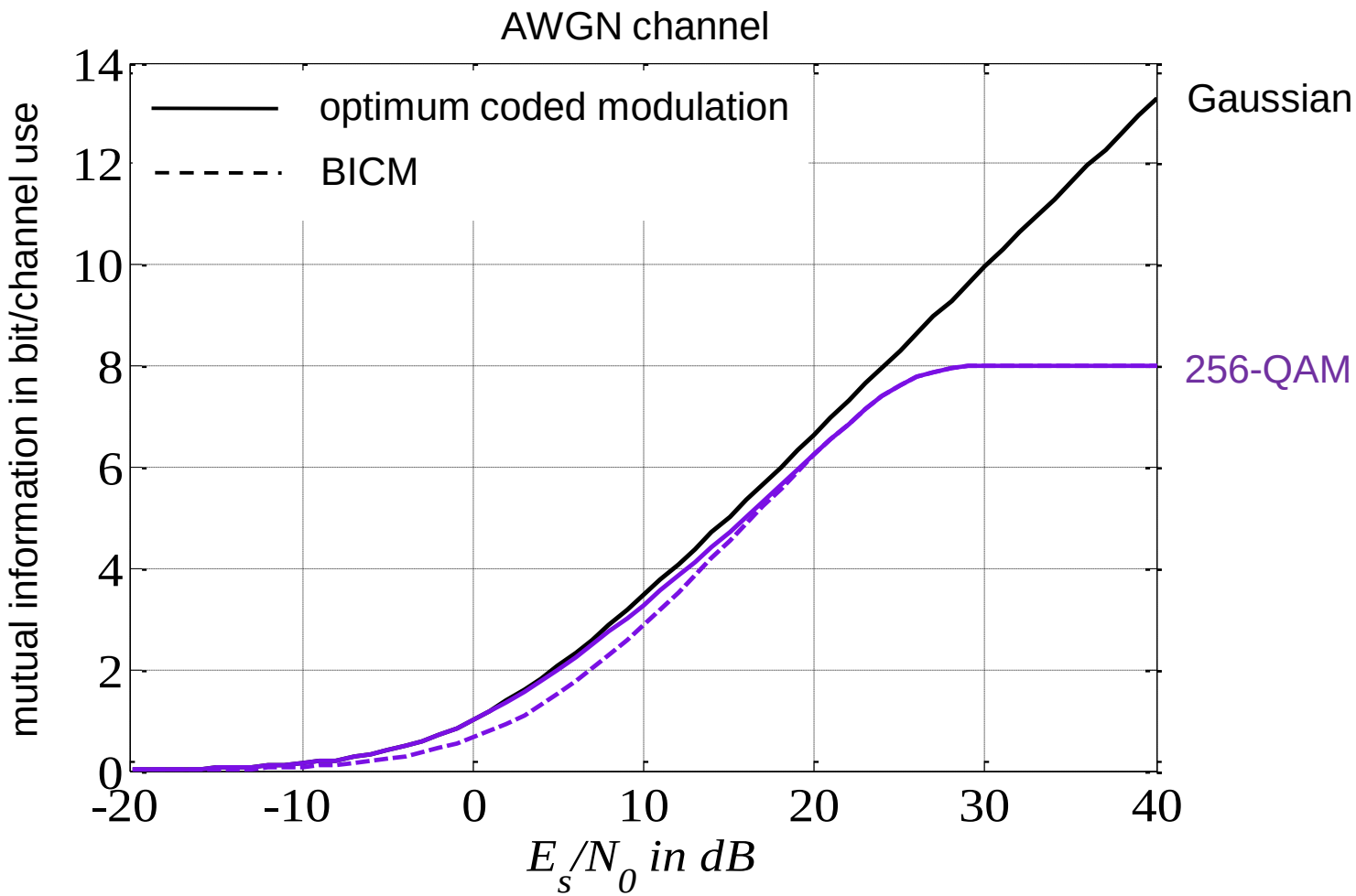
Bit-Interleaved Coded Modulation (BICM)



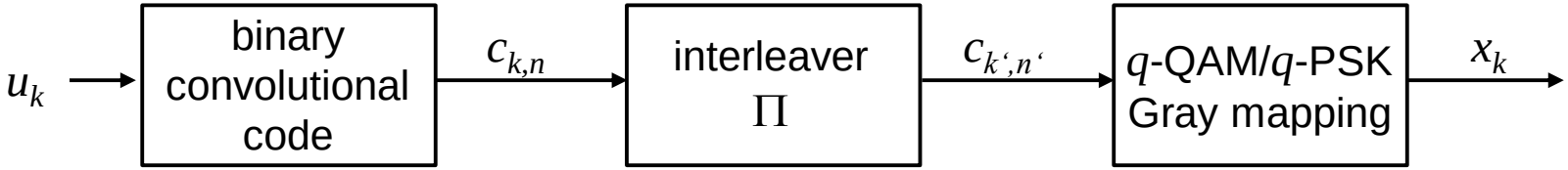


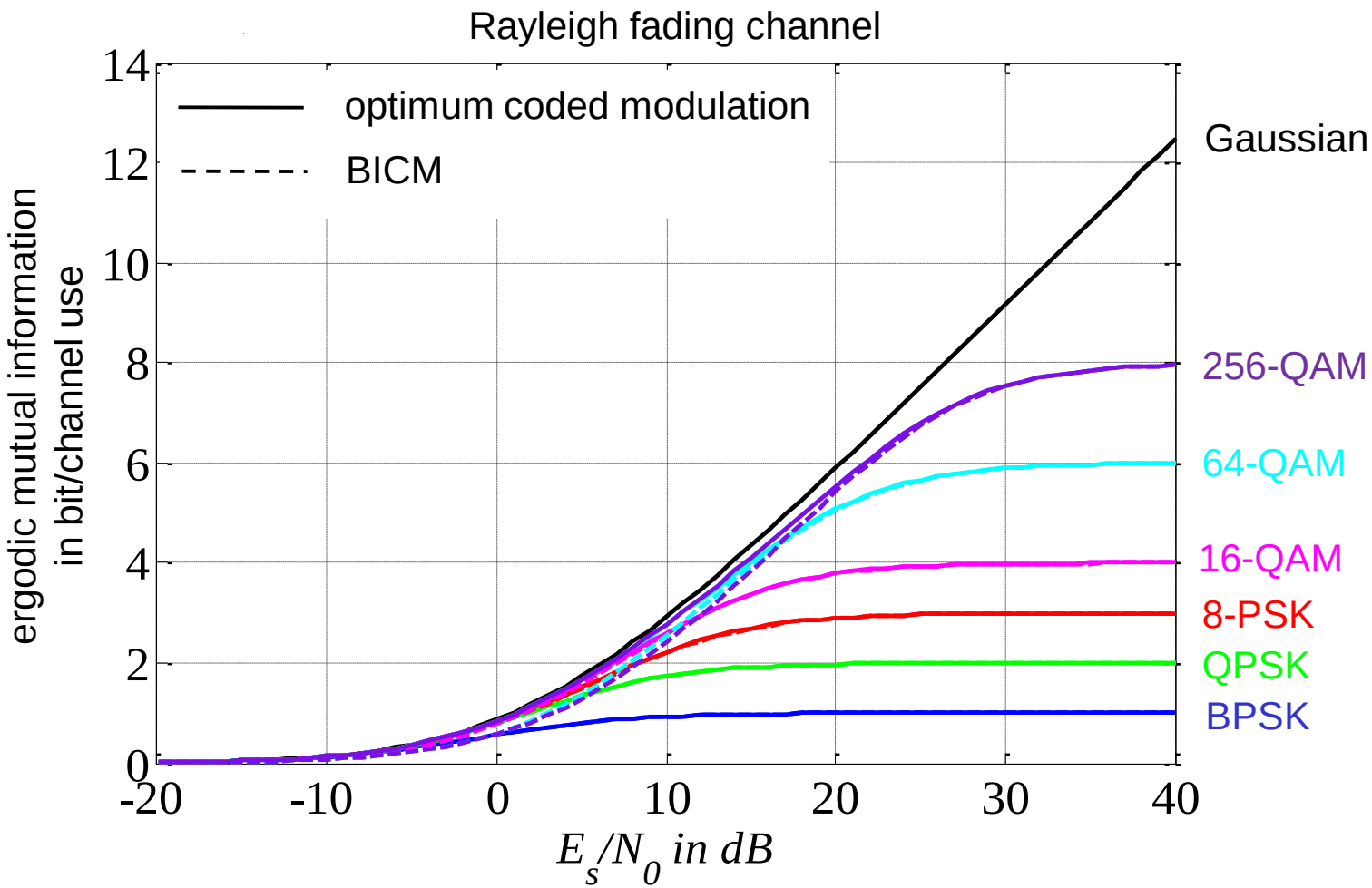
Bit-Interleaved Coded Modulation (BICM)





Bit-Interleaved Coded Modulation (BICM)





Bit-Interleaved Coded Modulation (BICM)

Bit-interleaved coded modulation is a very flexible coded modulation concept. The info bits are encoded by a binary convolutional code. The code bits are then interleaved and mapped to QAM transmit symbols. Interleaving means simply to change the order of the code bits. At the receiver, a demapper delivers log-likelihood ratios (soft information) about the code bits from the received noisy samples. A deinterleaver brings the log-likelihood ratios in the correct order as desired for decoding. Finally, a Viterbi or BCJR decoder is used in order to deliver estimates on the info bits.

The purpose of the interleaver is mainly to decorrelate the noise: Code bits which are transmitted in the same QAM symbol are affected by the same noise sample. This correlation would have to be taken into consideration in the decoder metric. By spreading the code bits in a pseudo random manner using an interleaver, the noise can be assumed to be uncorrelated in the decoder metric. Another purpose of the interleaver is to break up burst errors. Particularly in a fading channel, a burst of transmit symbols may be received errouneously due to a deep fade. The interleaver spreads the error burst such that the decoder may be able to correct the errors.

Bit-interleaved coded modulation is motivated by the observation that the information theoretic limit (achievable rate, mutual information, capacity) of bit interleaved coded modulation is almost the same as for optimum coded modulation, where coding and modulation is designed jointly. For QPSK modulation, there is virtually no difference between the achievable rate of bit-interleaved coded modulation and optimum coded modulation. However, for higher order modulation schemes, there is a degradation of bit interleaved coded modulation which increases with increasing cardinality of the modulation scheme.

Bit-Interleaved Coded Modulation (BICM)

Nevertheless, bit-interleaved coded modulation is widely used. The major advantage is its flexibility. We can easily switch between different code rates and modulation schemes (adaptive coding and modulation scheme). The code rate can be adapted by puncturing. The mapping of interleaved code bits to QAM symbols is usually done by Gray mapping in order to minimize the bit error probability in the received and demapped signal. With Gray-Mapping, constellation points with minimum Euclidean distance d_{min} differ in only 1 bit. The most likely symbol error at reasonably high SNR is to confuse constellation points at minimum Euclidean distance. For this most likely error event, we have only 1 bit error per symbol error. This minimizes the average BER. In contrast, for natural mapping error events at minimum Euclidean distance with 2 bit errors per symbol error can occur.

Adaptive coding and modulation is particularly important in wireless communications, where the transmission rate needs to be adapted to the current quality of the time-varying channel. Modern wireless communications systems use a table of different combinations of code rates and modulation schemes. The receiver estimates the channel based on a transmitted training sequence of known symbols (pilot symbols, reference symbols) and feeds back a channel quality indicator (CQI) (or channel state information (CSI)). Based on this feedback, the transmitter can choose the most appropriate combination of code rate and modulation scheme for the current channel state from the table of available combinations for transmission of the next frame.