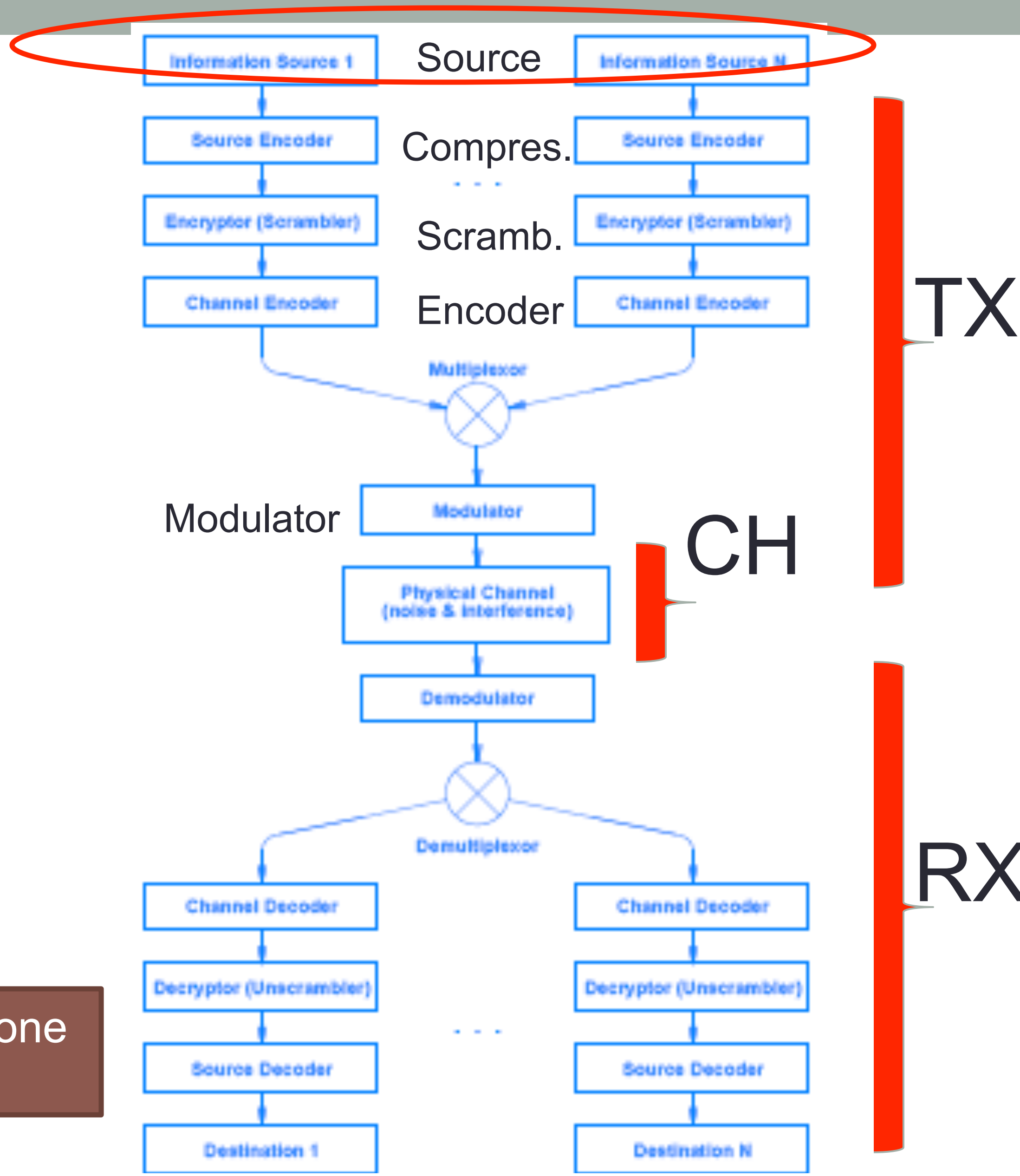
LINE CODING

11/20/11

Big Idea in Data Communications:

A conceptual framework for a data communications system. Multiple sources send to multiple destinations through an underlying physical channel

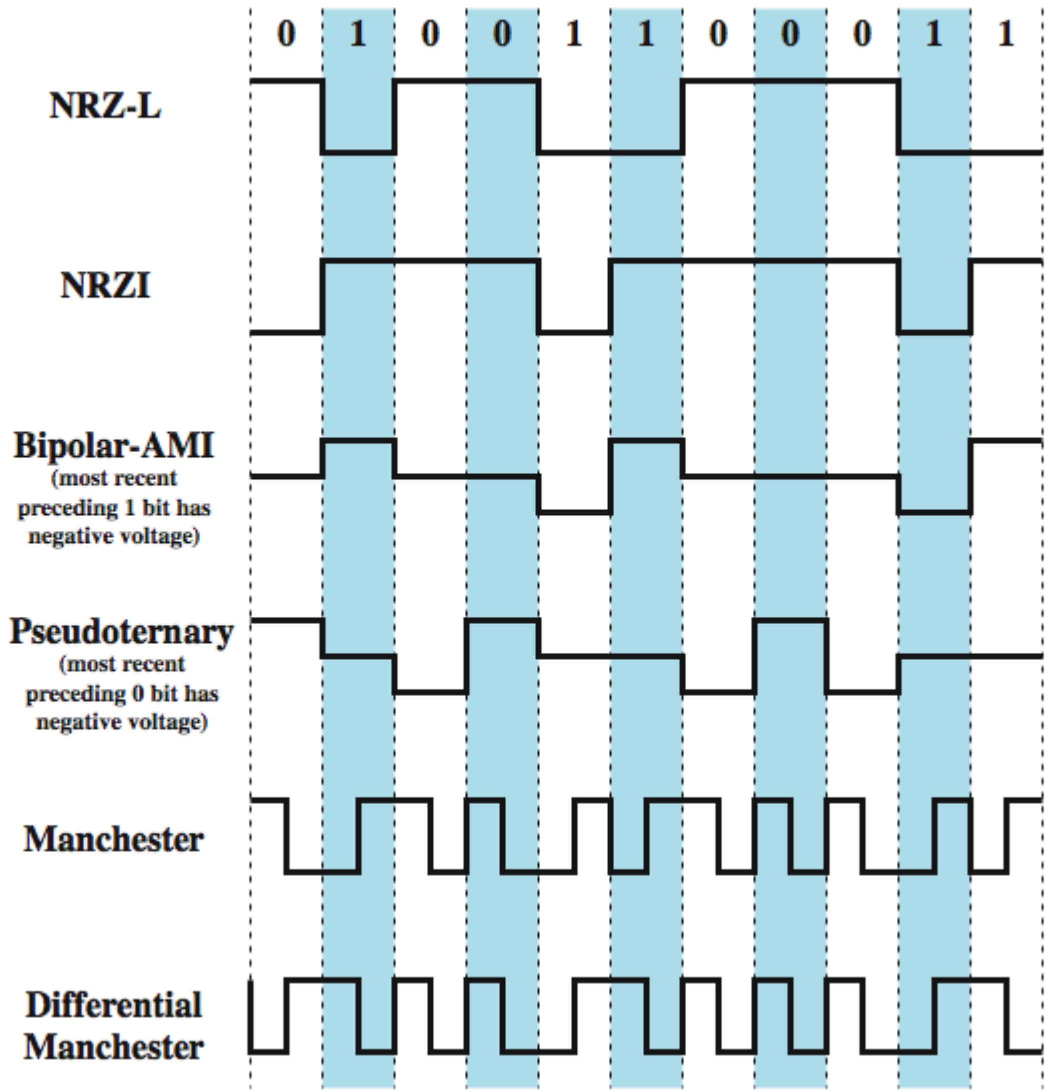


Each of the boxes corresponds to one subtopic of data communications:

Signal Encoding Design Goals

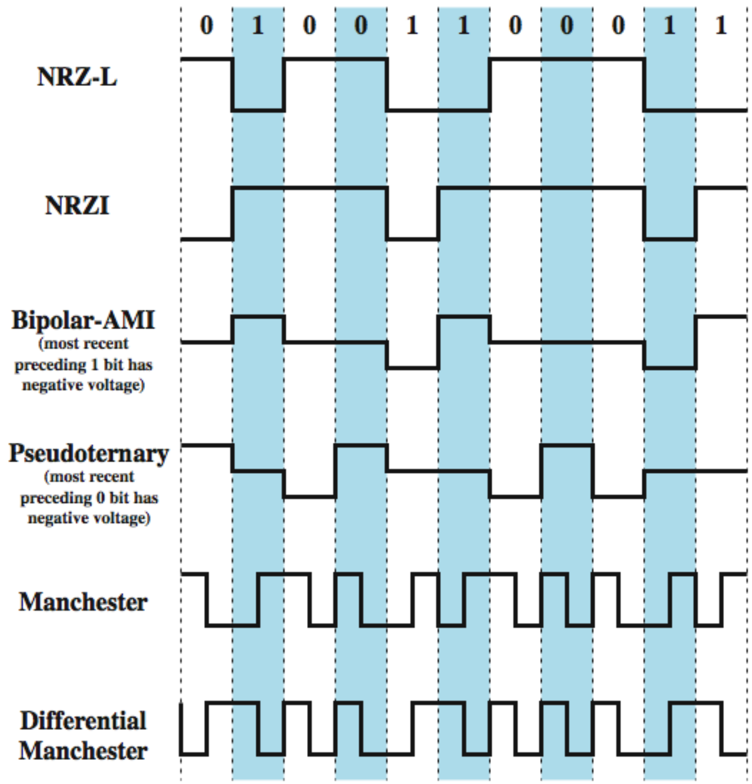
- No DC components
- No long sequence of zero-level line signals
- No reduction in data rate
- Error detection ability
- Low cost

Encoding Schemes (Line Coding Mechanisms)

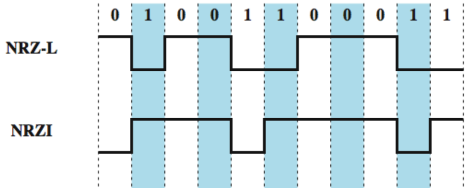


Nonreturn to Zero-Level (NRZ-L)

- two different voltages for 0 and 1 bits
  - 0= high level / 1 = low level



NRZI (Nonreturn to Zero – Invert on ones)

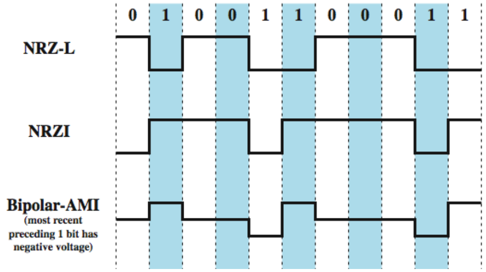


- Non-return to zero, inverted on ones
- constant voltage pulse for duration of bit
- data encoded as presence or absence of signal transition at the beginning of bit time
  - Data is based on transitions (low to high or high to low) – level change
  - Where there is a ONE → Transition occurs
  - Where there is a ZERO→ No transition occurs
- Advantages
  - data represented by changes rather than levels
  - more reliable detection of transition rather than level – when noise exists!

NRZI
Transition when we have a ONE
Otherwise→ no transition

NRZI is Differential encoding: information is transmitted based on changes between successive signal elements

Multilevel Binary Bipolar-AMI

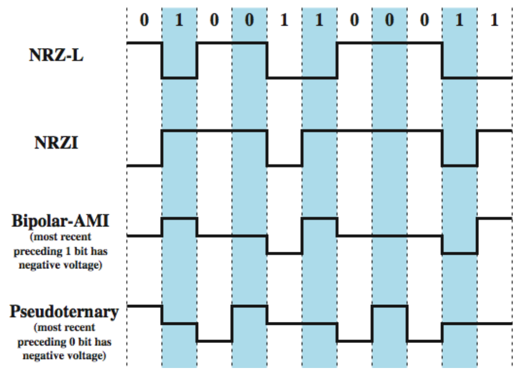


- AMI stands for alternate mark inversion
- Use more than two levels
- Bipolar-AMI
  - zero represented by no line signal
  - one represented by positive or negative pulse
  - One's pulses alternate in polarity
  - no loss of sync if a long string of ones
    - long runs of zeros still a problem
  - no net dc component
  - lower bandwidth
  - easy error detection

Bipolar - AMI
0 → 0
1,1→ +,-

Multilevel Binary Pseudoternary

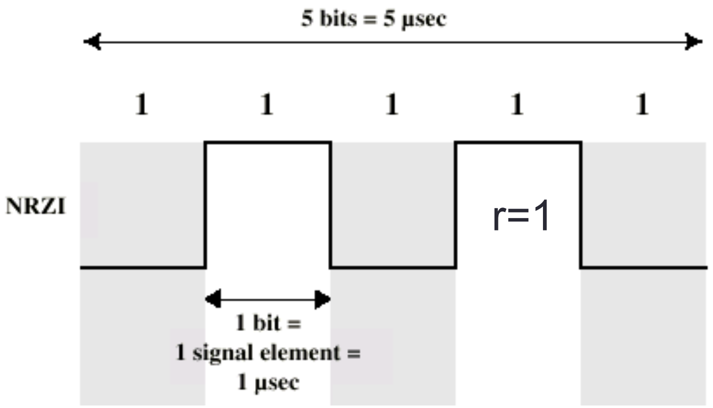
- one represented by absence of line signal
- zero represented by alternating positive and negative
- no advantage or disadvantage over bipolar-AMI
- each used in some applications



1 → 0
0,0→ +,-

Example

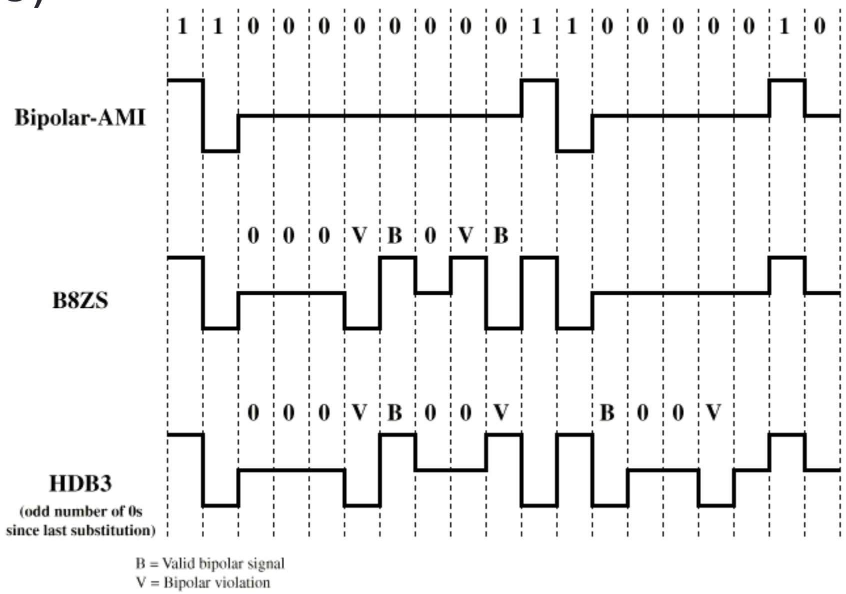
- Using NRZI, how do you represent 1 1 1 1 1?
- Assuming it takes 5usec to send 5 bits what is the duration of each bit?
- Assuming it takes 5usec to send 5 bits what is the duration of each signal element?
  - The signal will be 0 1 0 1 0 (toggling – starting with Zero as the initial state)
  - Each bit = 1 usec
  - Each signal element = 1 usec



Scrambling

- The objective is to avoid long sequences of zero level line signals and providing some type of error detection capability
- We compare two techniques:
  - B3ZS (bipolar 8-zero substitution)
  - HDB3 (High-density Bipolar-3 zeros)

B8ZS:
One octet of zero is replaced by:
000VB0VB
V = 1 code violation

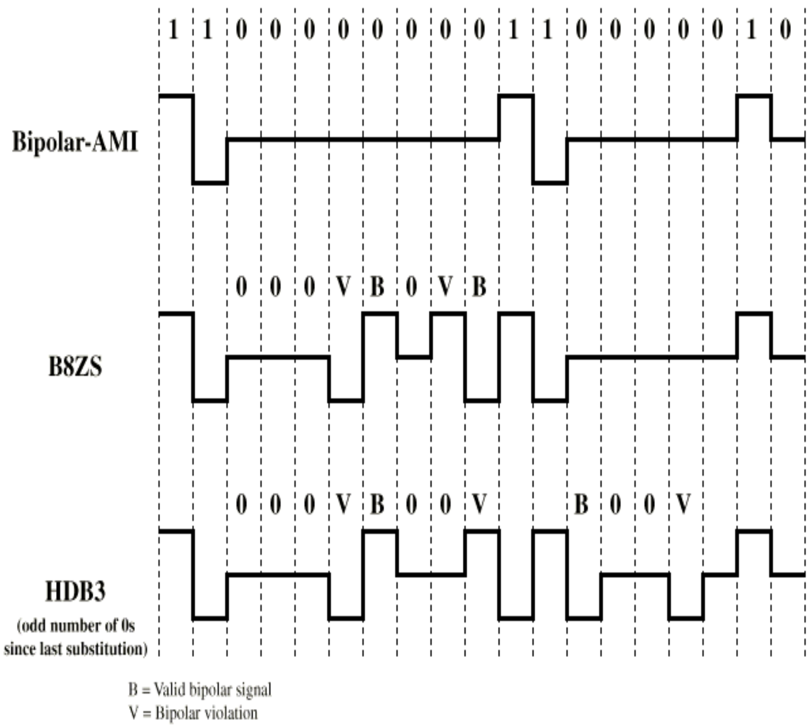


Scrambling

- The objective is to avoid long sequences of zero level line signals and providing some type of error detection capability
- We compare two techniques:
  - B3ZS (bipolar 8-zero substitution)
  - HDB3 (High-density Bipolar-3 zeros)

HDB3:
4 zeros are replaced by:
- 000V if the number of pulses (ones) since last substitution was ODD
- B00V if the number of pulses (ones) since last substitution was EVEN

V = 1code violation



Channel Coding

Error Correction in SONET

- BIT Interleaved Parity (BIP)
  - Uses Parity Bit

Two Strategies for Handling Channel Errors

- A variety of mathematical techniques have been developed that overcome errors during transmission and increase reliability
  - Known collectively as channel coding
- The techniques can be divided into two broad categories:
  - Forward Error Correction (FEC) mechanisms
  - Automatic Repeat reQuest (ARQ) mechanism
- In either case we are adding overhead
  - There is always a tradeoff - adding redundancy vs. error detection
- What is the impact of channel error?

Error Correction Motivation

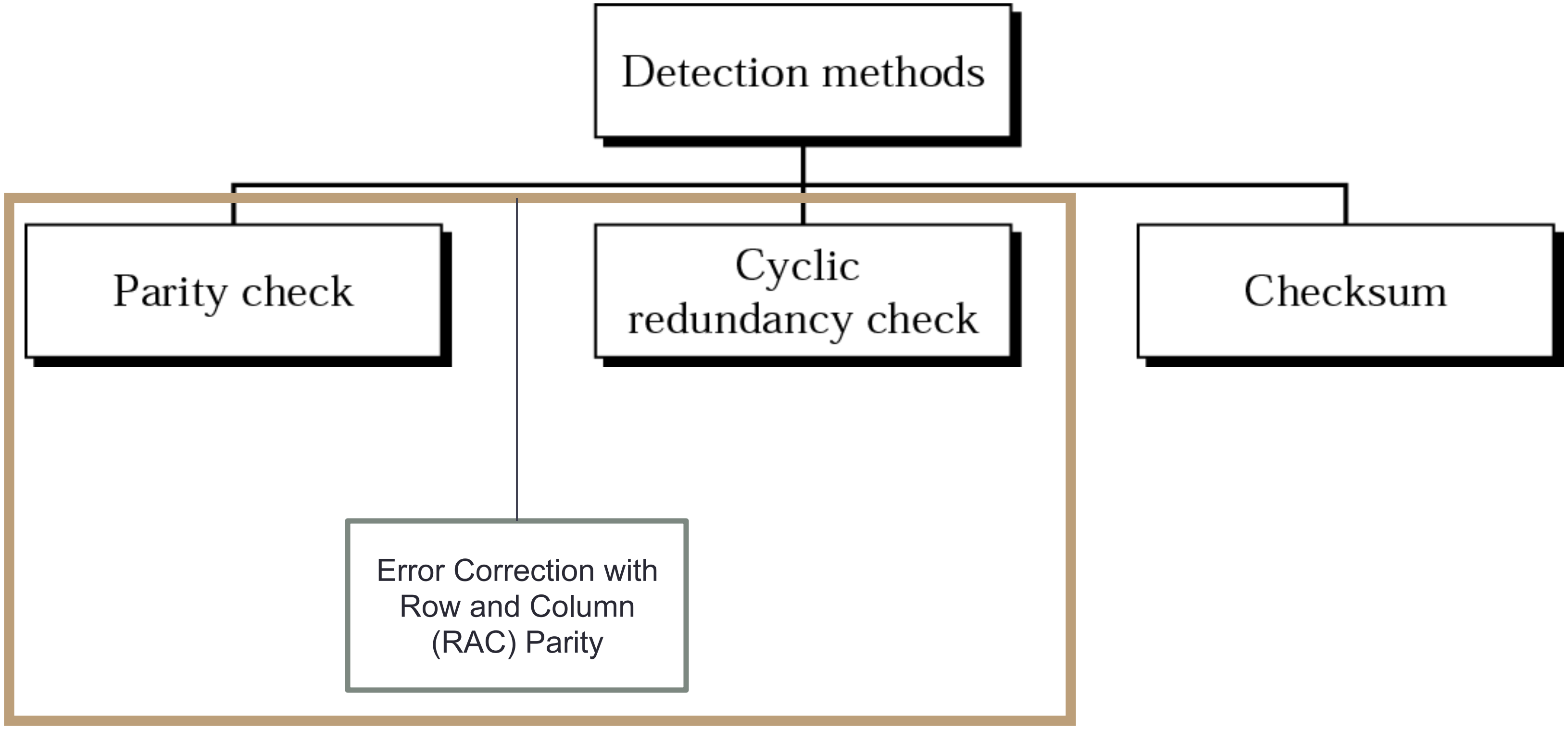
- Errors can be detected and corrected
  - Error correction is more complex
- Correction of detected errors usually requires data block to be retransmitted
- Instead need to correct errors on basis of bits received

Error Correction Basic Idea

- Adds redundancy to transmitted message
- Can deduce original despite some errors
  - Errors are detected using error-detecting code
  - Error-detecting code added by transmitter
  - Error-detecting code are recalculated and checked by receiver

  - map *k* bit input onto an *n* bit codeword
  - each distinctly different
  - When error occurs the receiver tries to guess which codeword sent was (e.g., teh → the)

Error Detection



Redundancy Check

1- Vertical Redundancy Check (VRC)
   - Parity Check
2- Longitudinal Redundancy Check (LRC)
3- Cyclic Redundancy Check

Error Detection – Parity Check

- Basic idea
  - Errors are detected using error-detecting code
  - Error-detecting code added by transmitter
  - error-detecting code are recalculated and checked by receiver
- Parity bit
  - Odd (odd parity)
    - If it had an even number of ones, the parity bit is set to a one, otherwise it is set to a zero
    - (P=0 if odd ones)→ always odd number of ones in the frame
    - Asynchronous applications and Standard in PC memory
  - Even (even parity)
    - Synchronous applications

F(1110001)→
odd parity 1 111 000 1
Parity Bit + Data Block

Error Detection – Parity Check

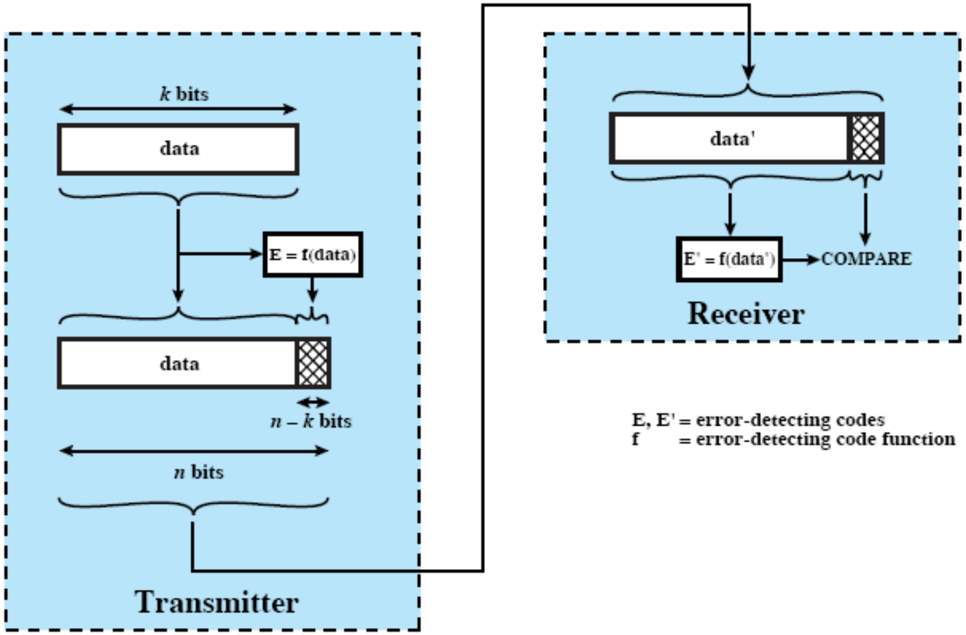
An Example Block Error Code:
Single Parity Checking

Original Data	Even Parity	Odd Parity
0 0 0 0 0 0 0 0	0	1
0 1 0 1 1 0 1 1	1	0
0 1 0 1 0 1 0 1	0	1
1 1 1 1 1 1 1 1	0	1
1 0 0 0 0 0 0 0	1	0
0 1 0 0 1 0 0 1	1	0

If even number of 1s → Even parity =0

Error Detection Basic Mechanism

- for block of *k* bits transmitter
- Represented by (n,k) encoding scl
  - k dataword length
  - n codeword
  - r added bits

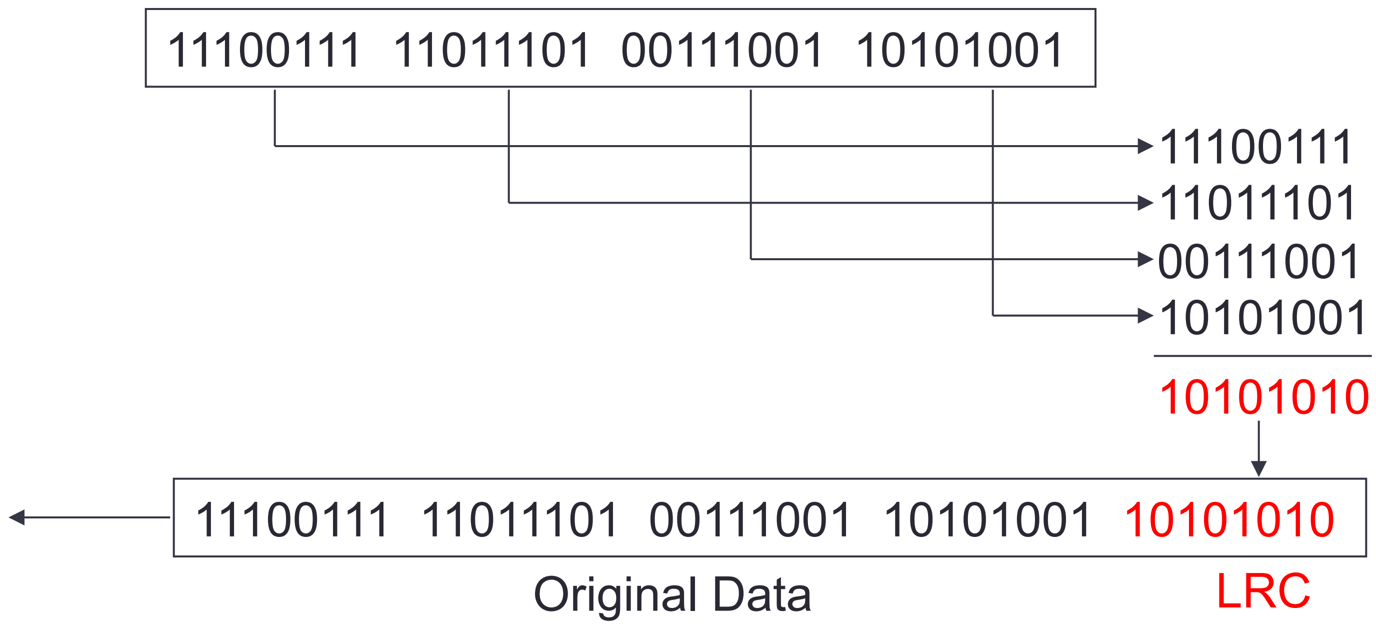


What is the minimum number of bits we should add?

Example: 8-bit data + single parity bit→ 2^9 (512) possibilities / only 2^8 (255=256-1) valid code words (excluding all-zero)

Redundancy Check

- **Longitudinal Redundancy Check (LRC)**
  - Organize data into a table and create a parity for each column



Hamming Distance: A Measure of a Code's Strength

- No channel coding scheme is ideal!
  - changing enough bits will always transform to a *valid* codeword
- What is the minimum number of bits of a valid codeword that must be changed to produce another valid codeword?
  - To answer the question, engineers use a measure known as the Hamming distance
  - Given two strings of n bits each, the Hamming distance is defined as the number of differences

d(000,001) = 1	d(000,101) = 2
d(101,100) = 1	d(001,010) = 2
d(110,001) = 3	d(111,000) = 3

The Tradeoff Between Error Detection and Overhead

- A large value of d_{min} is desirable
  - because the code is immune to more bit errors, if fewer than d_{min} bits are changed, the code can detect that error(s) occurred
- The maximum number of bit errors that can be detected:

$$e = d_{\min} - 1$$

- A code with a higher value of d_{min} sends more redundant information than an error code with a lower value of d_{min}
- Code rate that gives the ratio of a dataword size to the codeword size

$$R = \frac{k}{n}$$

Error Detection and Correction

- Relation between Hamming Distance and Error
  - When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error

---

  - Ex : if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is d (00000, 01101) = 3

- To guarantee the detection of up to e errors in all cases, the minimum Hamming distance in a block code must be

$$d_{min} = e + 1 \rightarrow e = d_{min} - 1$$

- To guarantee the maximum t correctable errors in all cases

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

Cyclic Redundancy Codes (CRC)

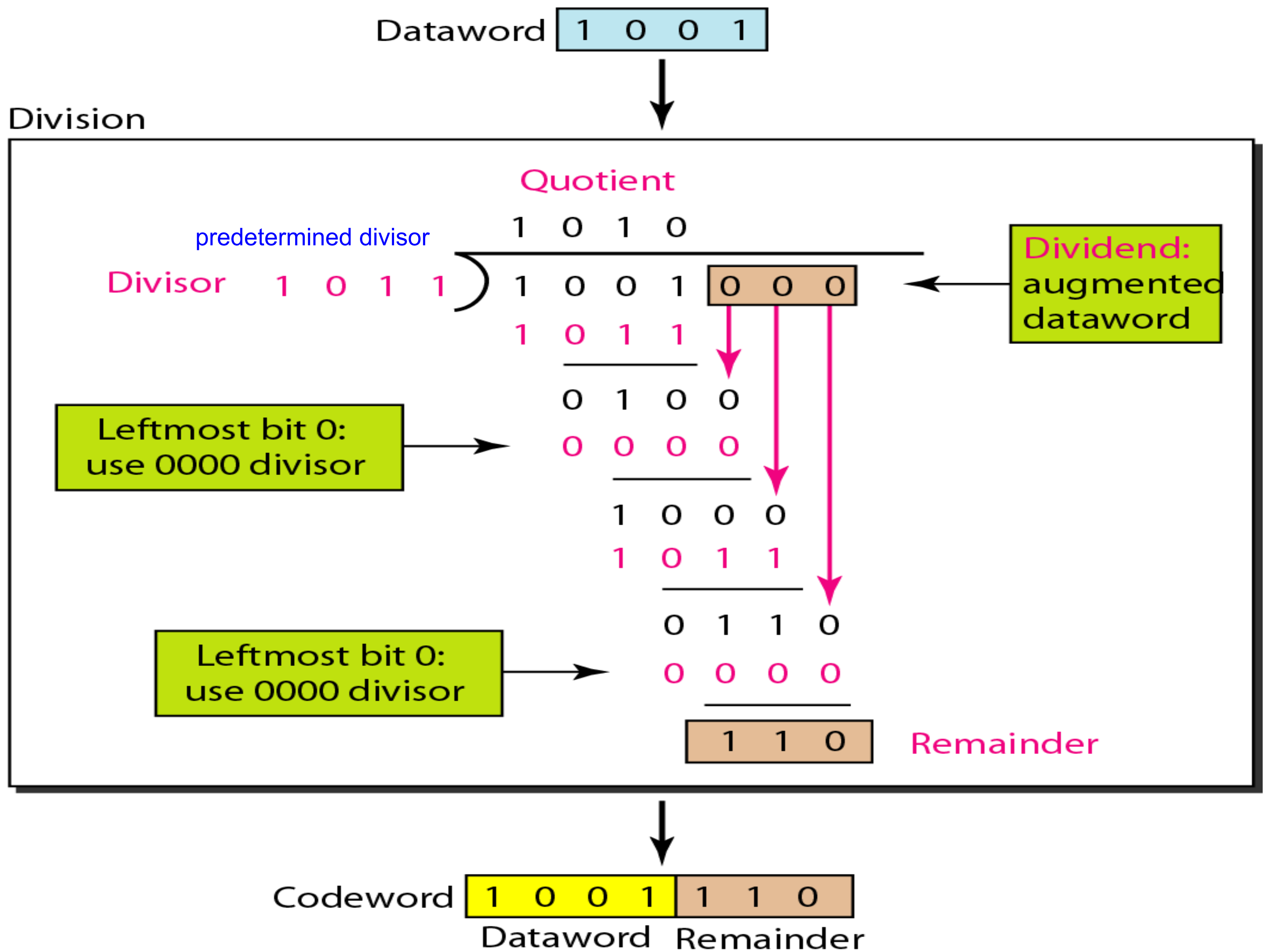
- Term cyclic is derived from a property of the codewords:
  - A circular shift of the bits of any codeword produces another one
- A (n=7, k=4) CRC by Hamming

Dataword	Codeword
0000	0000 000
0001	0001 011
0010	0010 110
0011	0011 101
0100	0100 111
0101	0101 100
0110	0110 001
0111	0111 010

Dataword	Codeword
1000	1000 101
1001	1001 110
1010	1010 011
1011	1011 000
1100	1100 010
1101	1101 001
1110	1110 100
1111	1111 111

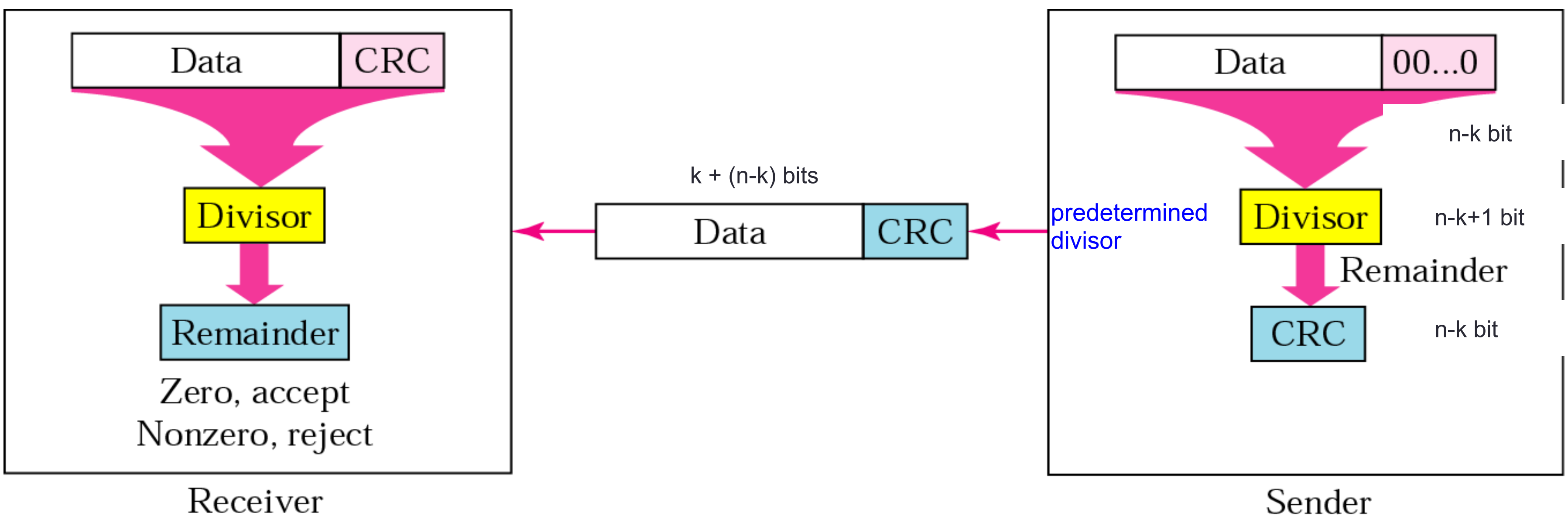
CRC generator and checker

- Example : Division in CRC Encoder



CRC generator and checker

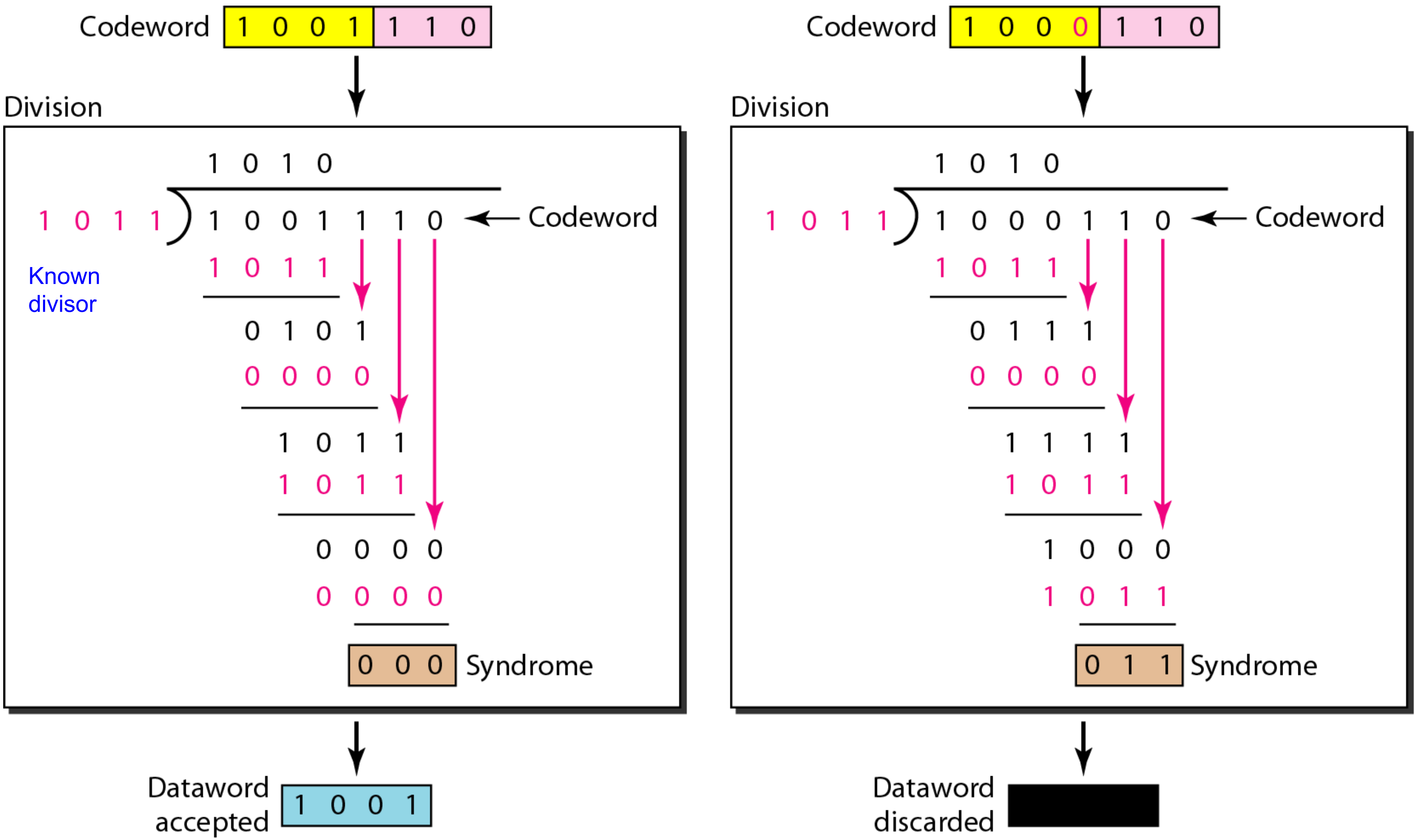
transmits *n* bits which is exactly divisible by some number (predetermined divisor)
receiver divides frame by that number



Refer to your notes for examples!

CRC generator and checker

- At the Receiver:
  - Example : Division in CRC Decoder



Cyclic Redundancy Codes (CRC)

Mathematical Representation

- Let M(x) be the **message polynomial**
- Let P(x) be the **generator polynomial (divisor)**
  - P(x) is fixed for a given CRC scheme
  - P(x) is known both by sender and receiver
- Create a block polynomial F(x) based on M(x) and P(x) such that F(x) is divisible by P(x)

$$\frac{F(x)}{P(x)} = Q(x) + \frac{0}{P(x)}$$

Example of CRC

- Send
  - M(x) = 110011 → x^5+x^4+x+1 (6 bits)
  - P(x) = 11001 → x^4+x^3+1 (5 bits, n = 4)
    → 4 bits of redundancy
  - Form $x^nM(x)$ → 110011 <u>0000</u>
    → $x^9+x^8+x^5+x^4$
  - Divide $x^nM(x)$ by P(x) to find C(x)

$$
\begin{array}{r}
100001 \\
11001\overline{)1100110000} \\
\underline{11001} \\
10000 \\
\underline{11001} \\
1001
\end{array}
$$

→ C(x)

Send the block 110011 <u>1001</u>

- Receive

$$
\begin{array}{r}
11001\overline{)1100111001} \\
\underline{11001} \\
11001 \\
\underline{11001} \\
00000
\end{array}
$$

↓

No remainder
→ Accept

$$
\frac{F(x)}{P(x)} = Q(x) + \frac{0}{P(x)}
$$

Forward Error Correction

- Used in OTN (10Gbps)
- RS codes:
  - n (symbols) = k (symbols) + r (symbols) → 125 usec
  - 1 symbol has m bits
  - 2^m-1 = n symbols
- Example:
  - N =255 ; r=16;→ k=239
  - Each symbol is 8 bytes
- Uses Reed-Solomon codes
  - (255,239), r=16; 7 (16/239) percent redundancy, Corrected errors: r/2=8
  - (255,223), r=16; 15 (32/223) percent redundancy, Corrected errors: r/2=16

System Performance

- Assume n=4, k=2 → Code rate ½
- Given BER, coding can improve Eb/No
  - Lower Eb/No is required
  - Code gain is the reduction in dB in Eb/No for a given BER
    - E.g., for BER=10^-6 → code gain is 2.77 dB
- Energy per coded bit (Eb) = ½ data bit (Eb)
  - Hence, BER will be 3dB less
  - This is because Ebit=2xEdata
- For very high BER, adding coding requires higher Eb
  - Not due to overhead