Intelligent Signal Processing

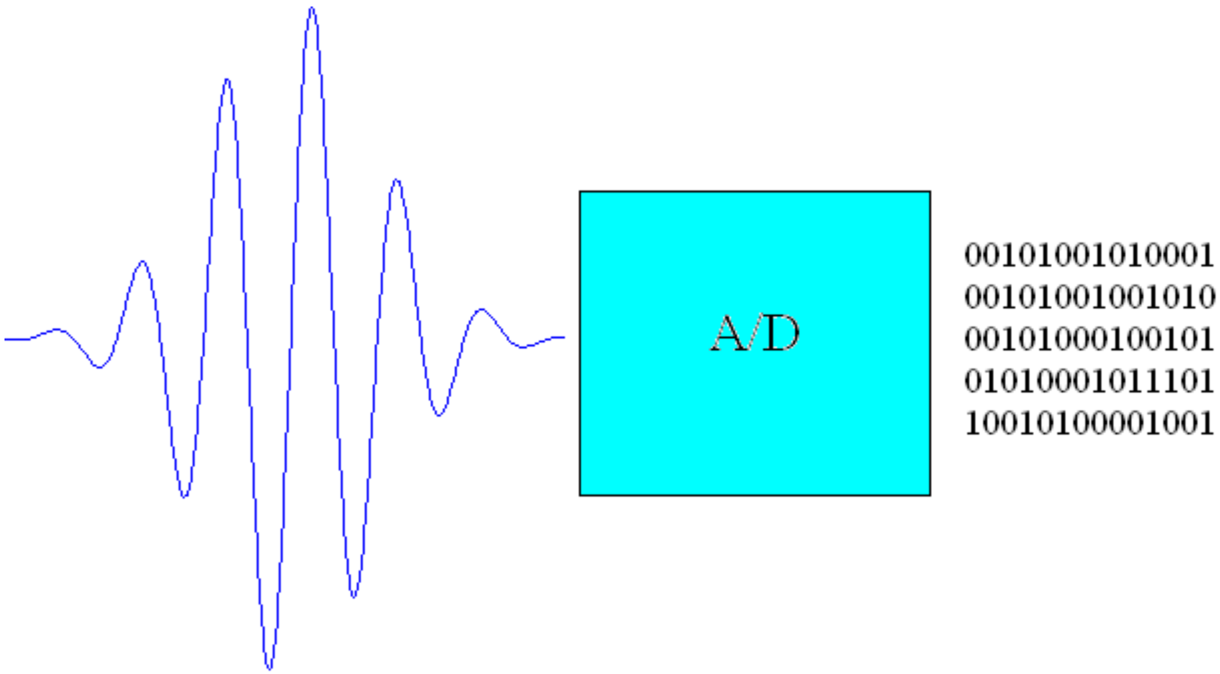
Audio Digitization

Angelo Ciaramella

Introduction

- The signals of everyday reality are continuous in time and in amplitude
  - voices
  - urban noise
  - musical listening
  - …
- In modern acquiring and transmission systems, information is binary (0/1)

Sound digitization



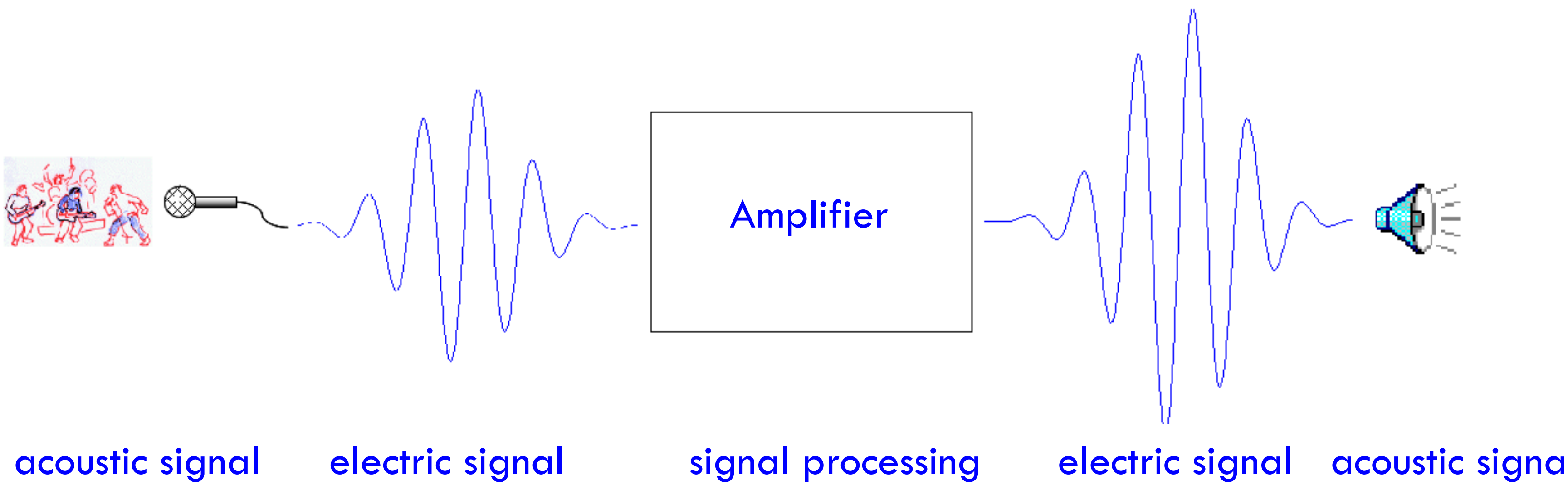
continuous signal

digital signal

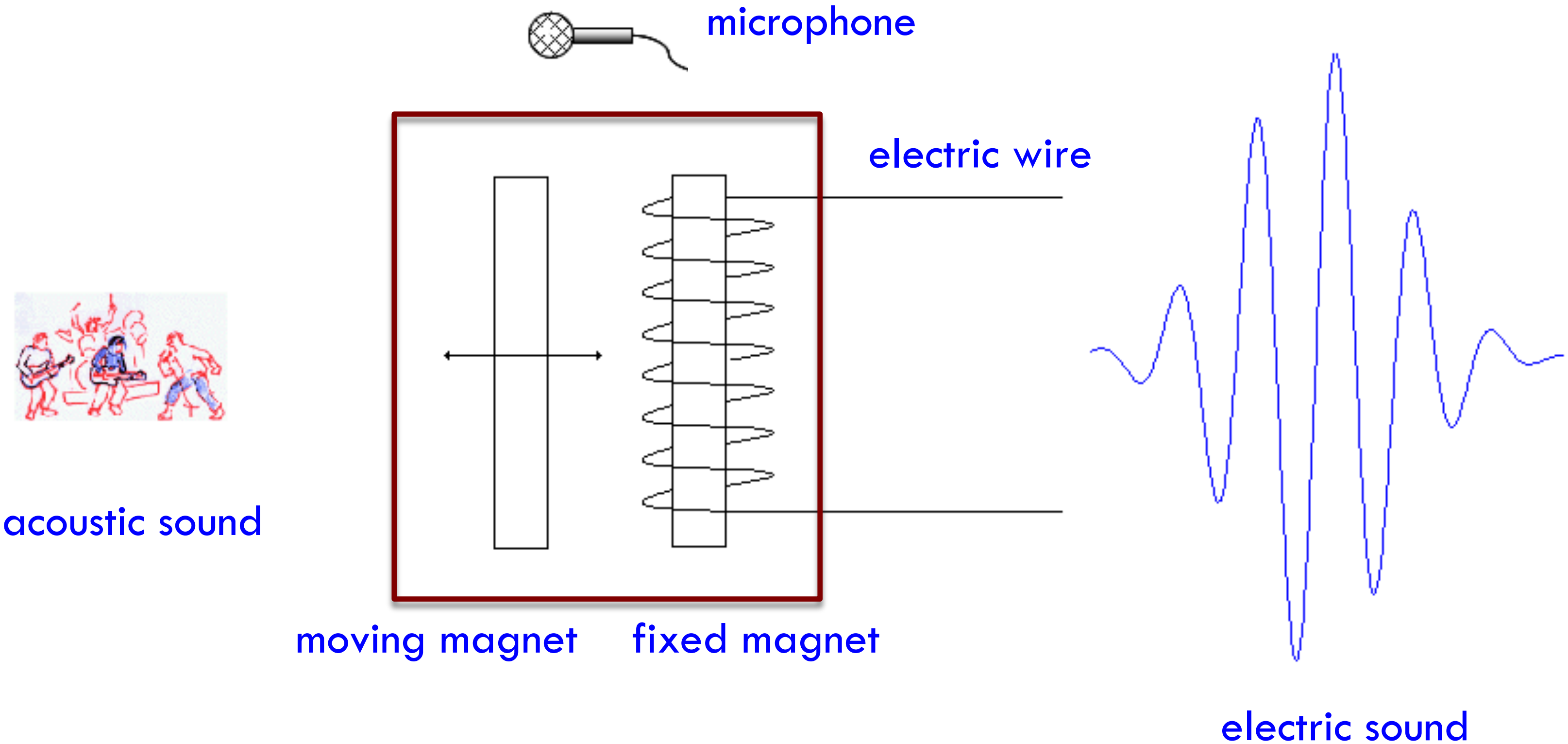
Transduction

- Transduction is the mechanism for transforming a physical signal into an electric and viceversa
  - Sensors
    - transduction from acoustic to electric signals
    - e.g., microphone
  - Actuators
    - transduction from electric to acoustic signals
    - e.g., speakers

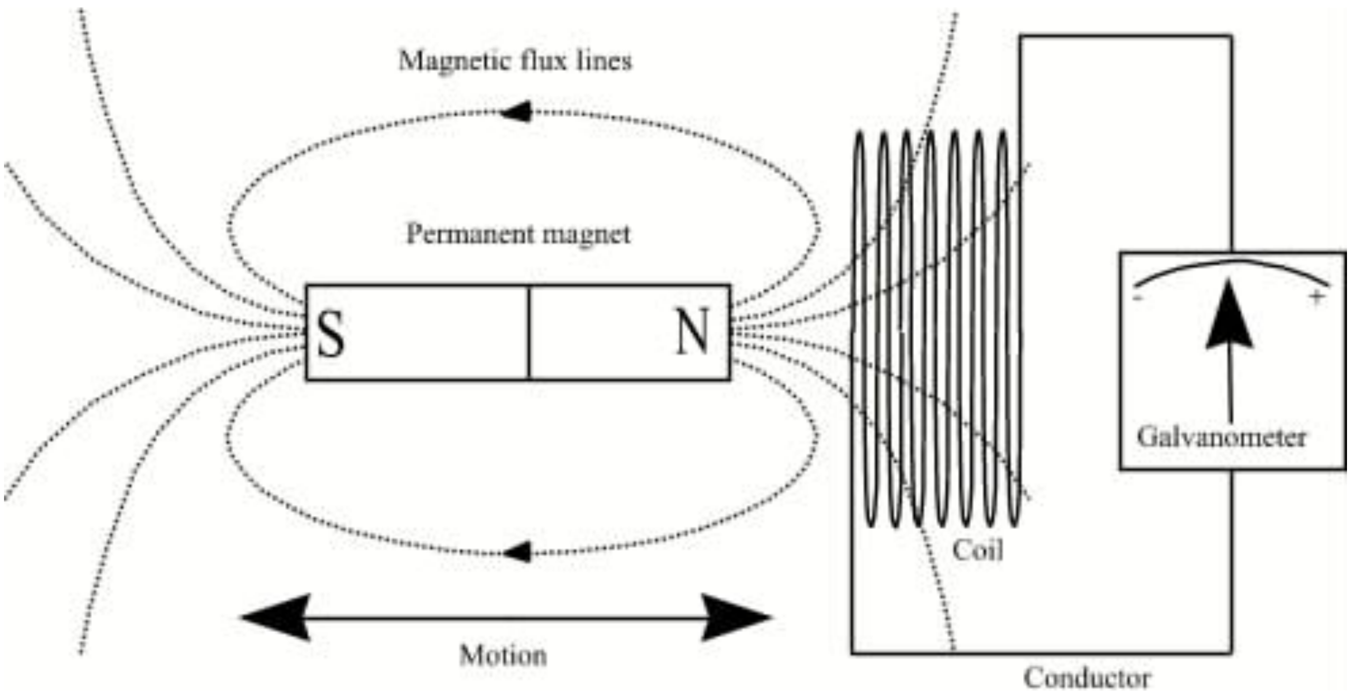
Transduction scheme



Principle of electromagnetic induction

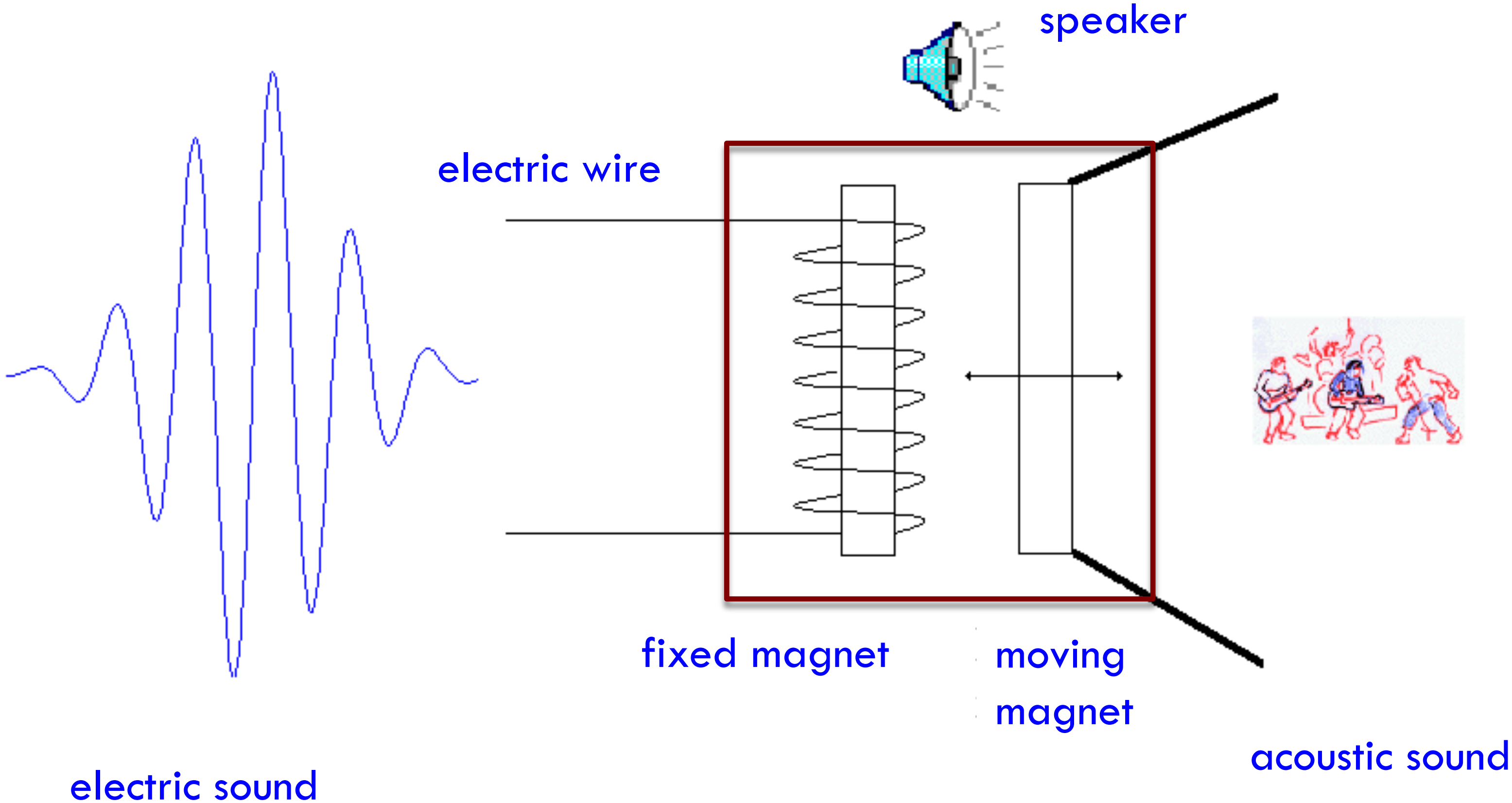


Faradays Law

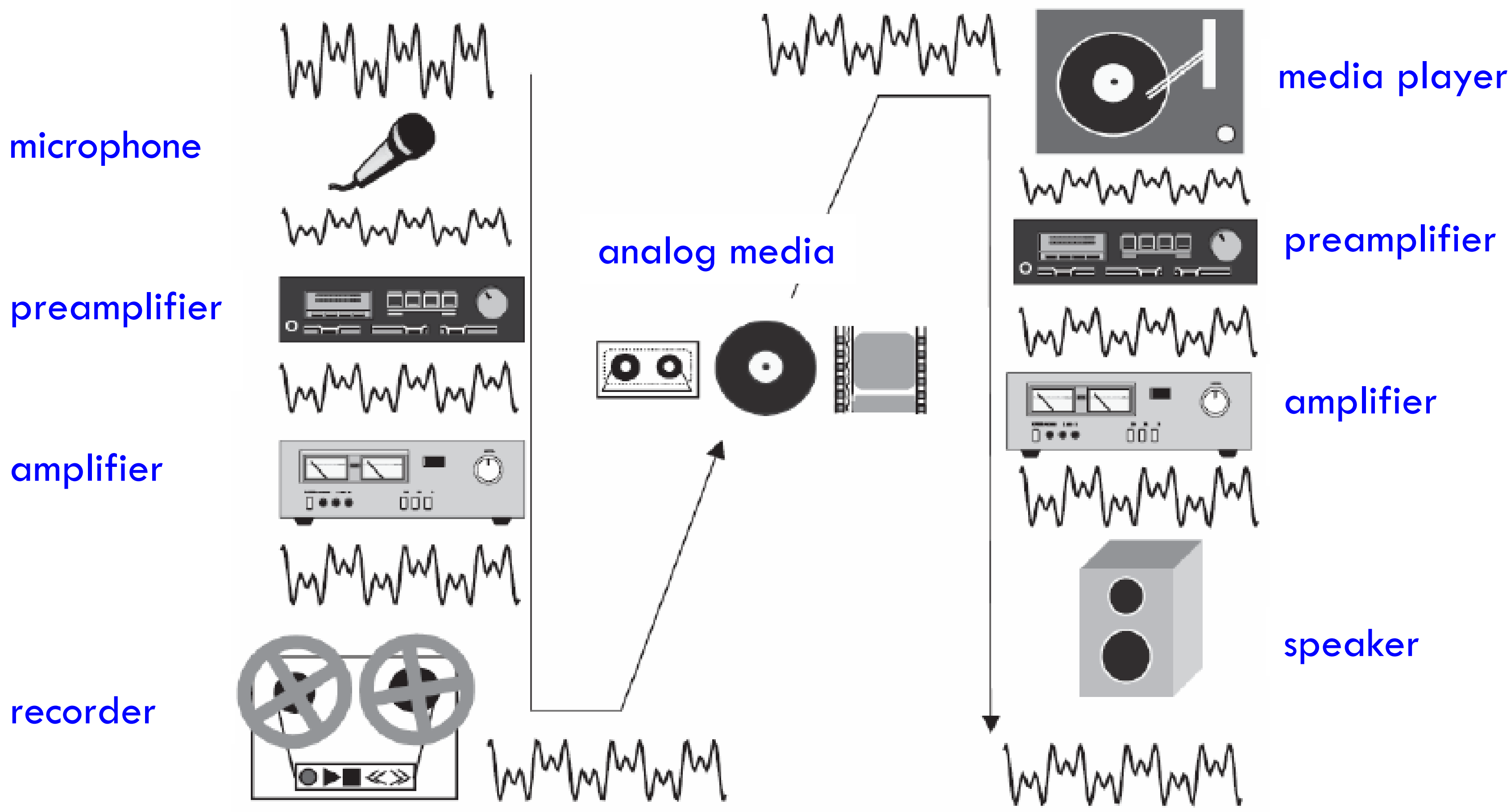


fixed magnet

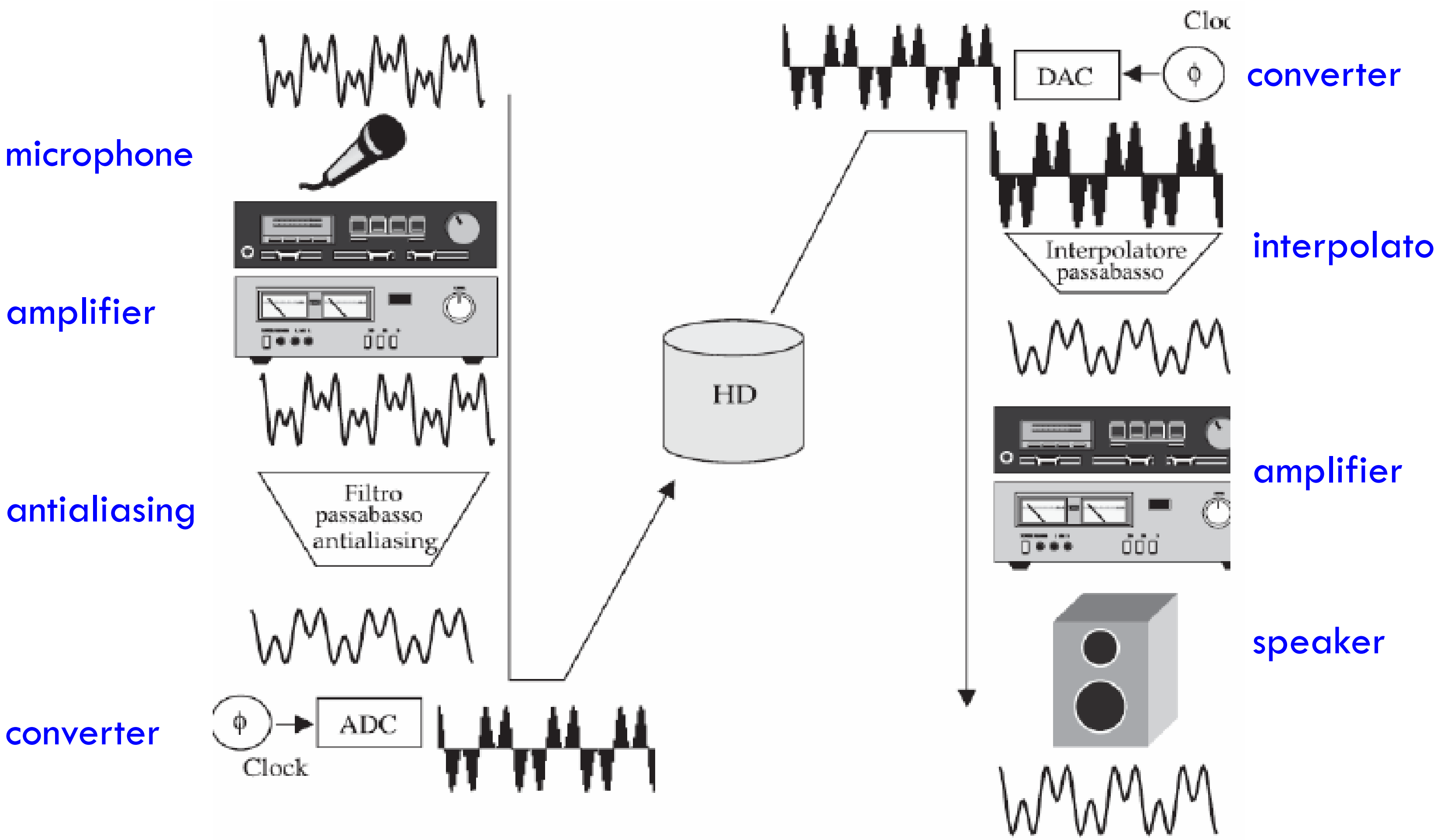
Electromagnetic induction



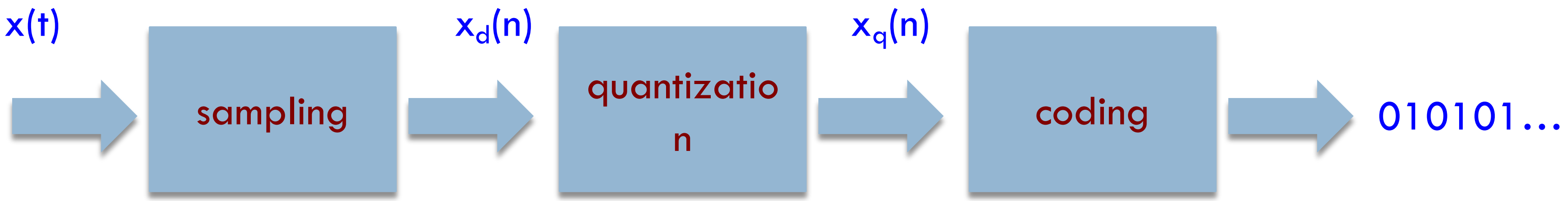
Analog components



Digital components

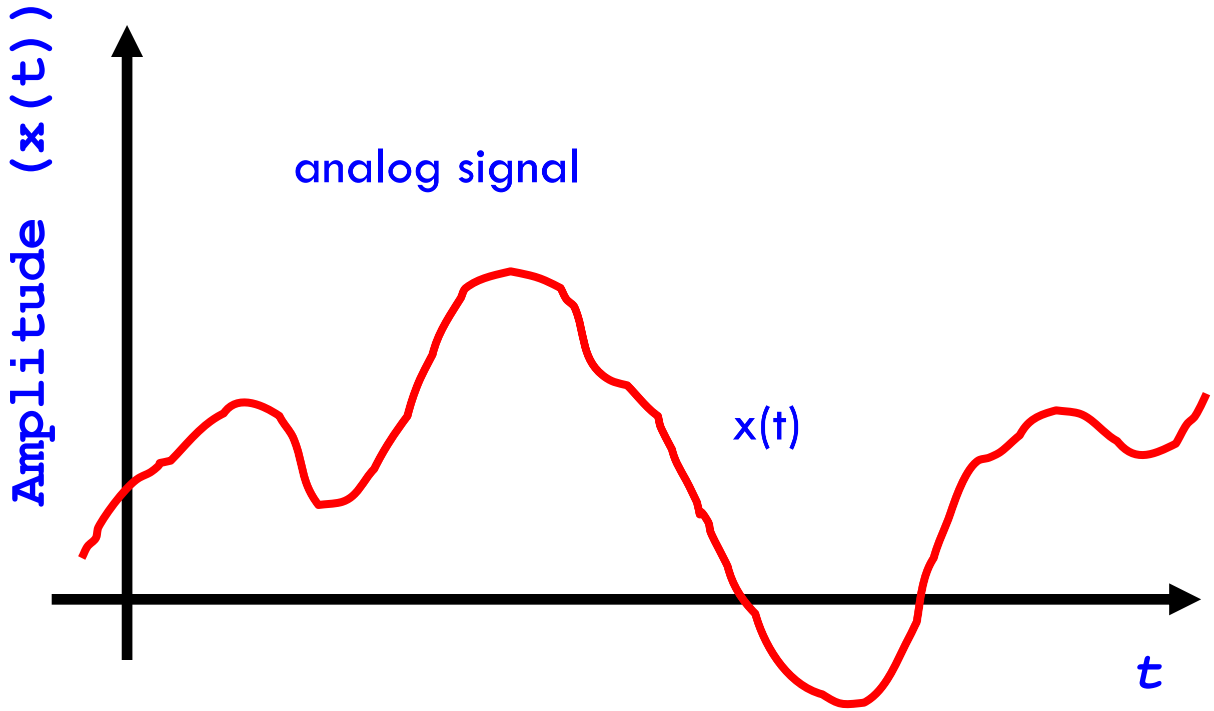


Analog to Digital Converter



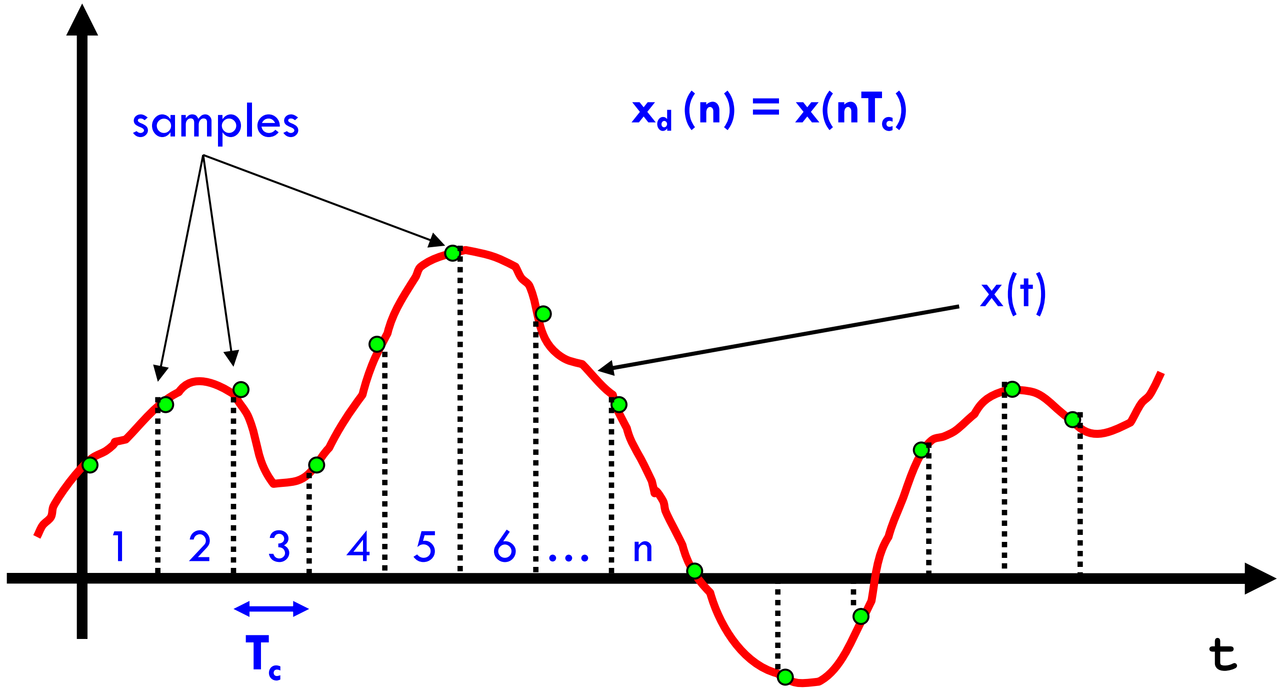
Phases of the ADC process

Analog signal



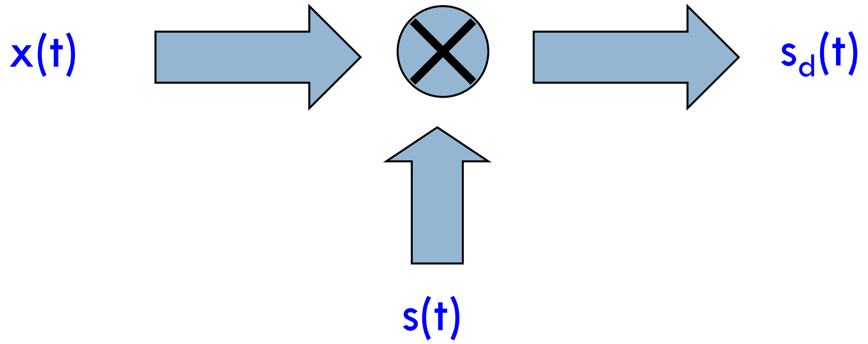
Analog signal temporal representation

Analog signal sampling



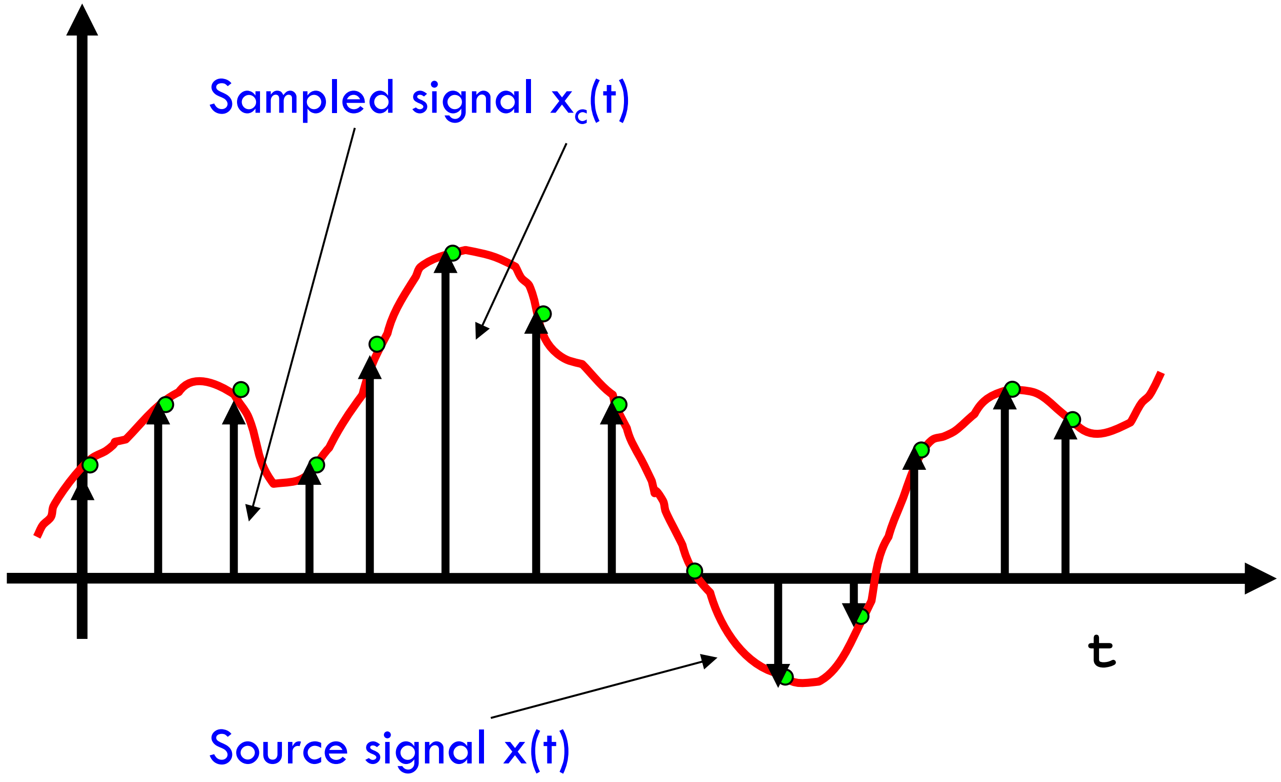
Analog signal temporal representation

Pulse Code Modulation (PCM)

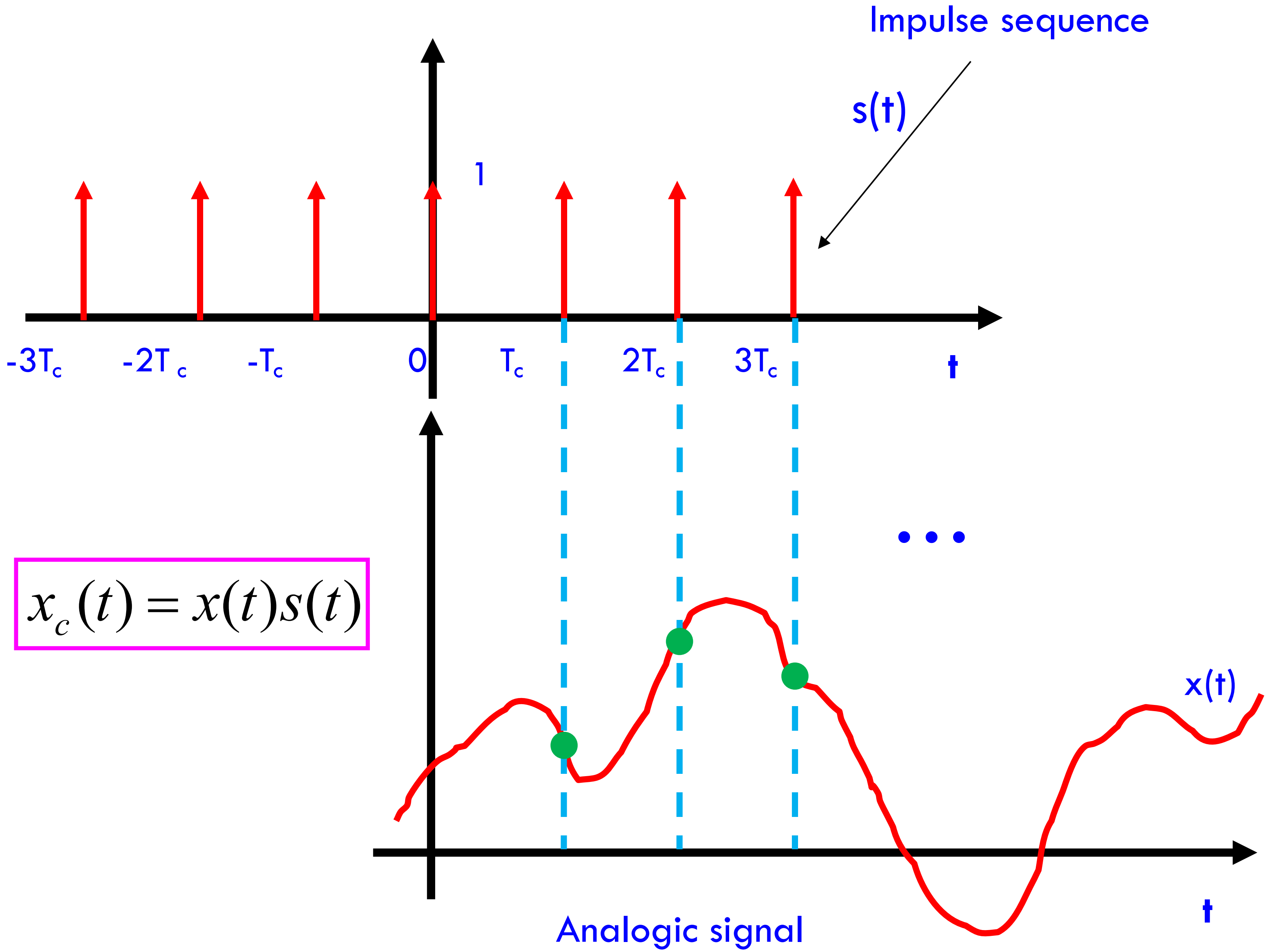


s(t) is a periodic impulse sequence (δ(t), delta di Dirac)

Pulse Code Modulation (PCM)



Pulse Code Modulation (PCM)



$$x_c(t) = x(t)s(t)$$

Sampling theorem

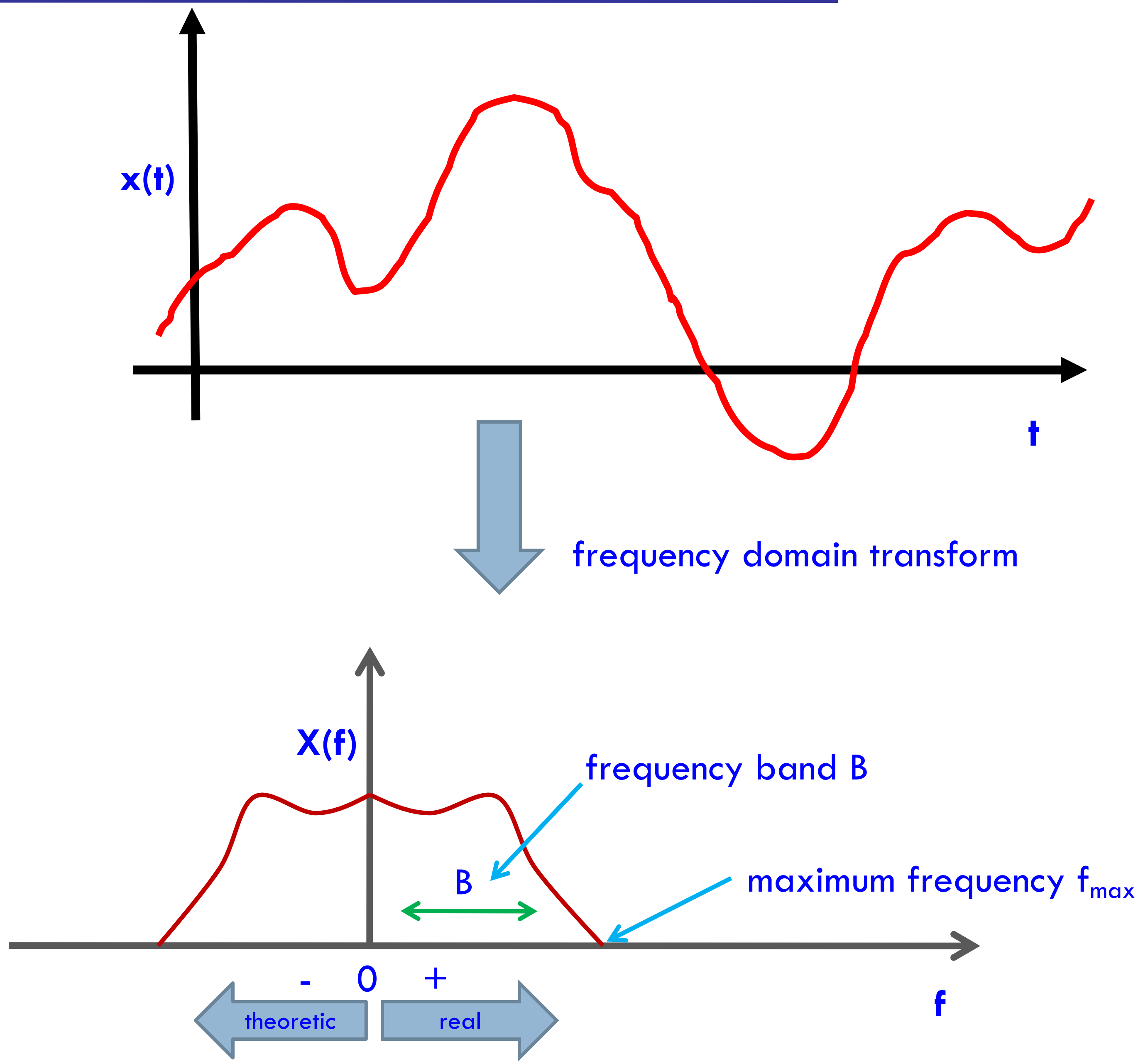
- The minimum sampling frequency which is necessary to avoid distortions in the signal reconstruction
- Introduced by Harold Nyquist, and appeared in 1949 in an article authored by E. C. Shannon
- Given a signal with a limited and known bandwidth, the minimum sampling frequency of this signal must be at least twice its highest frequency

Sampling theorem

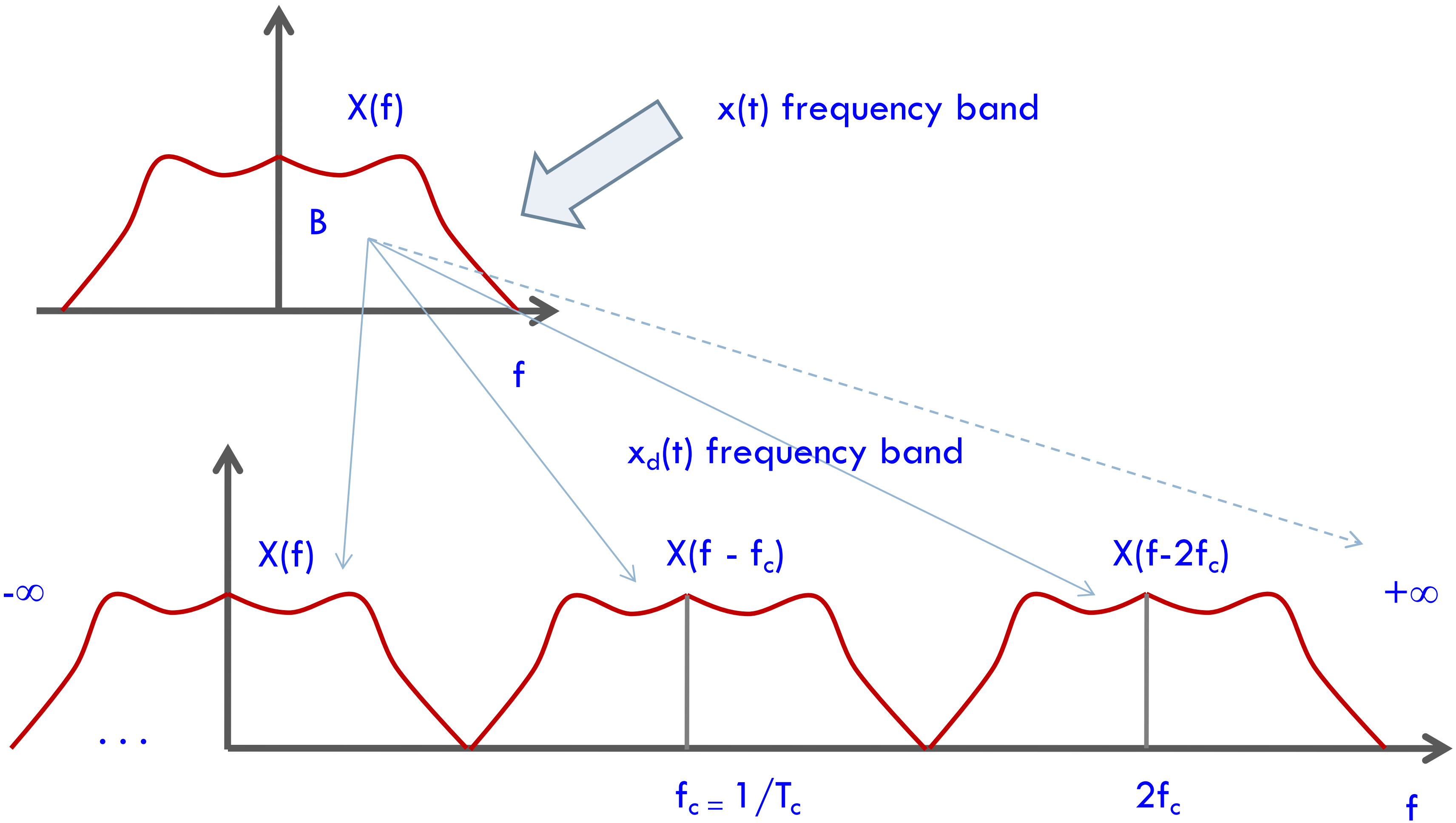
- A continuous-time signal $x(t)$ with a spectral band B strictly limited ($X(f) = 0$ for $|f| > \pm B$) can be uniquely reconstructed from its sampled version $x(n)$ ($n = 0, \pm 1, \pm 2, \pm 3, \ldots$) if the sampling frequency $f_c = 1/T_c$ satisfies the following relation

$$f_c = \frac{1}{T_c} \geq 2\mathbf{B}$$

Sampling theorem

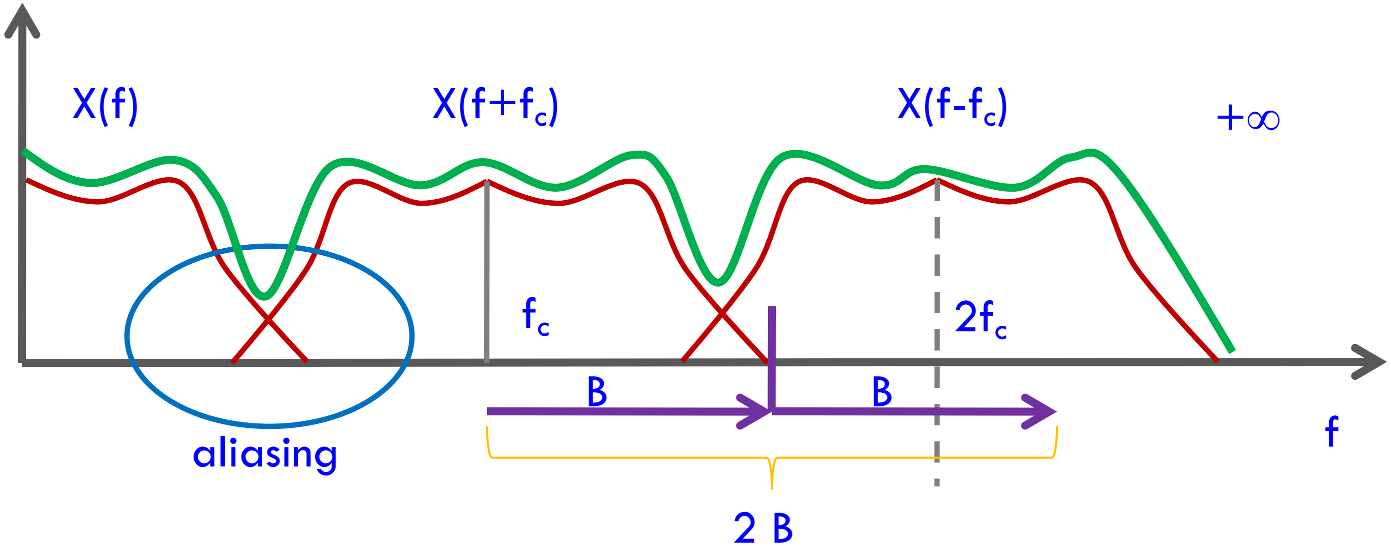


Sampling theorem



Sampling theorem

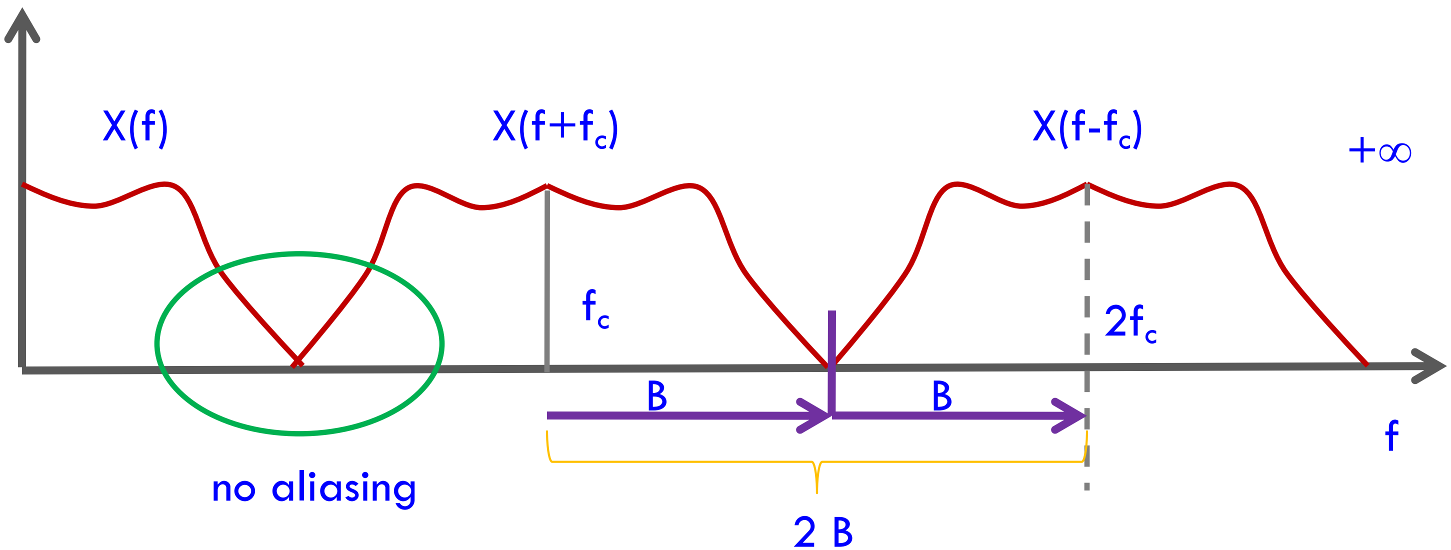
$$f_c < 2B$$



Subsampling of the signal

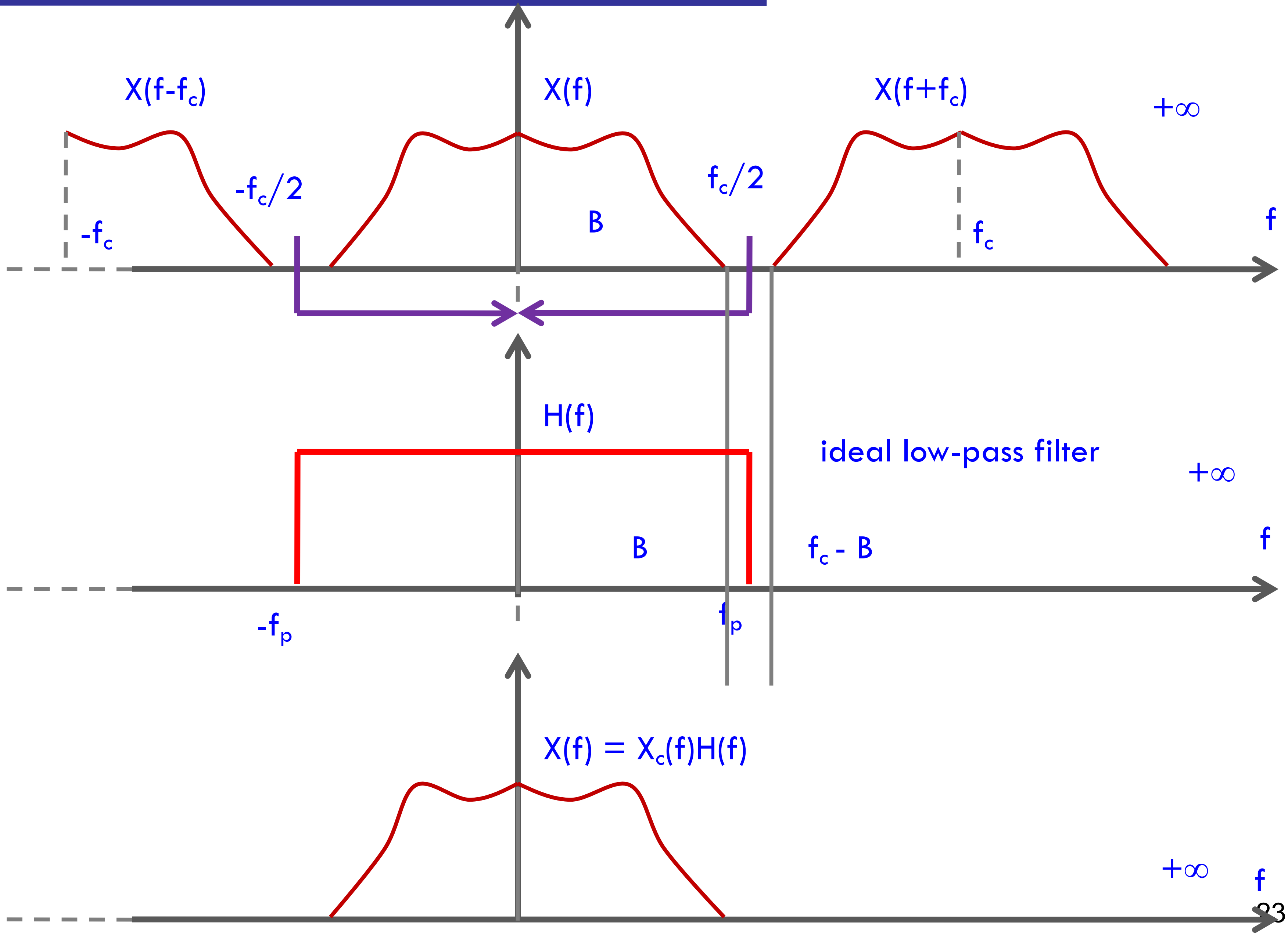
Sampling theorem

$$f_c \geq 2B$$

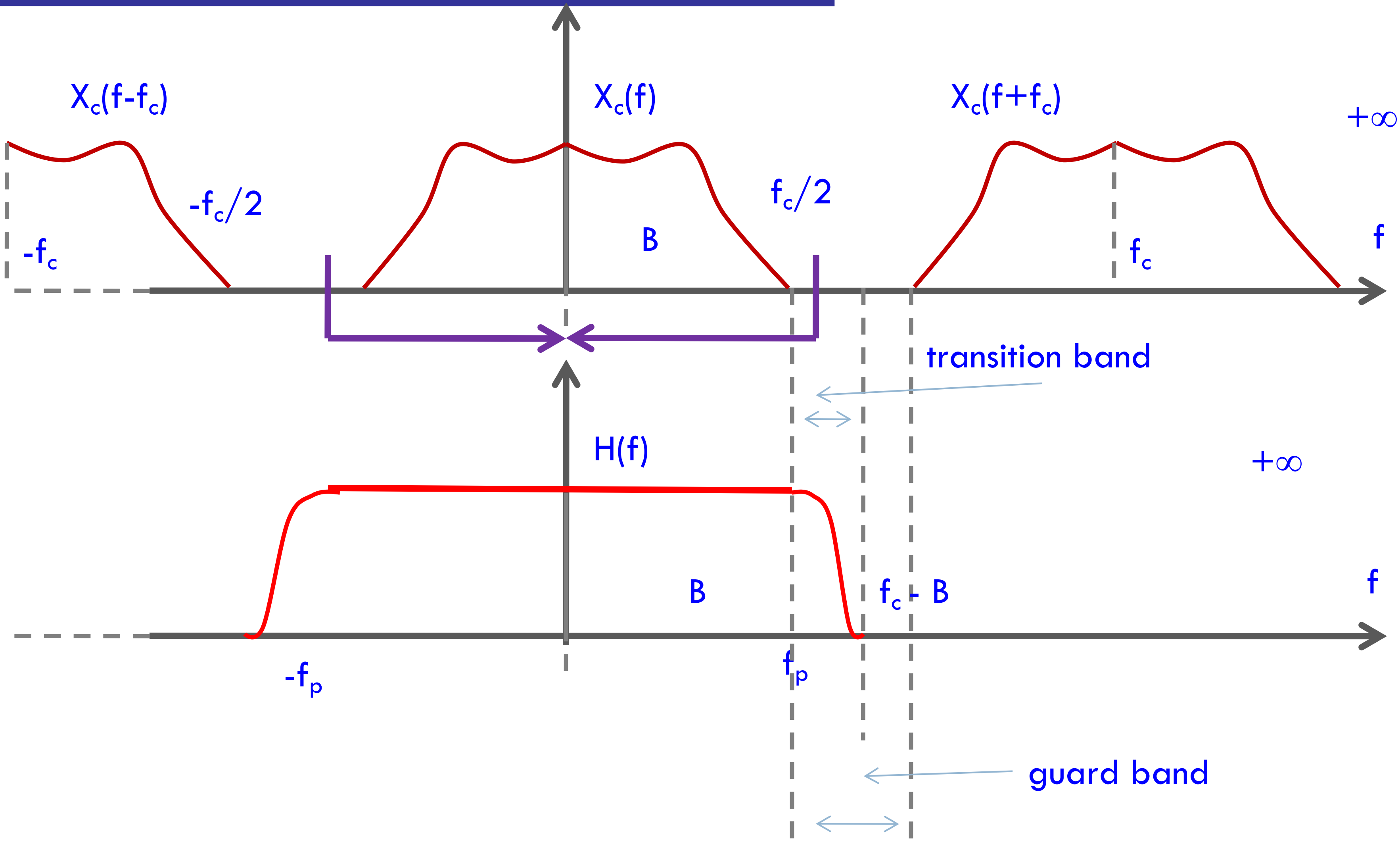


Oversampling of the signal

Signal reconstruction



Real filters



ISP – Audio Digitization

Example

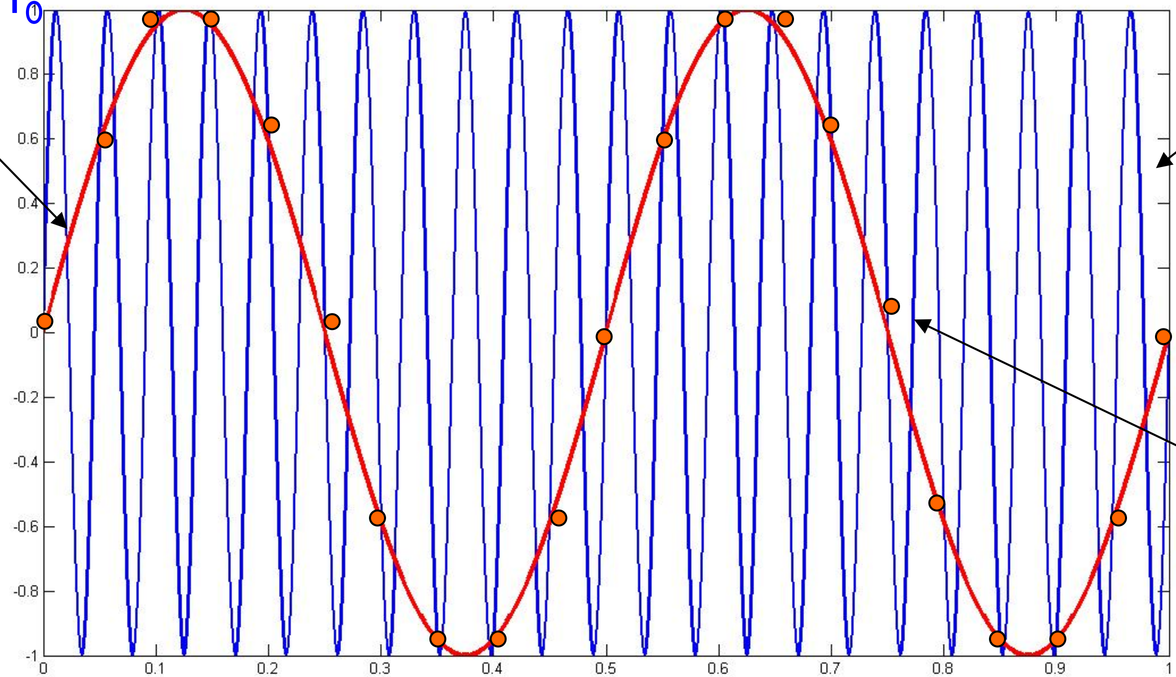
Reconstruct signal

f_{alias} =

f_c - $f_0 < f_0$

Source signal with

frequency f_0



Samples with

$f_c < 2 f_0$

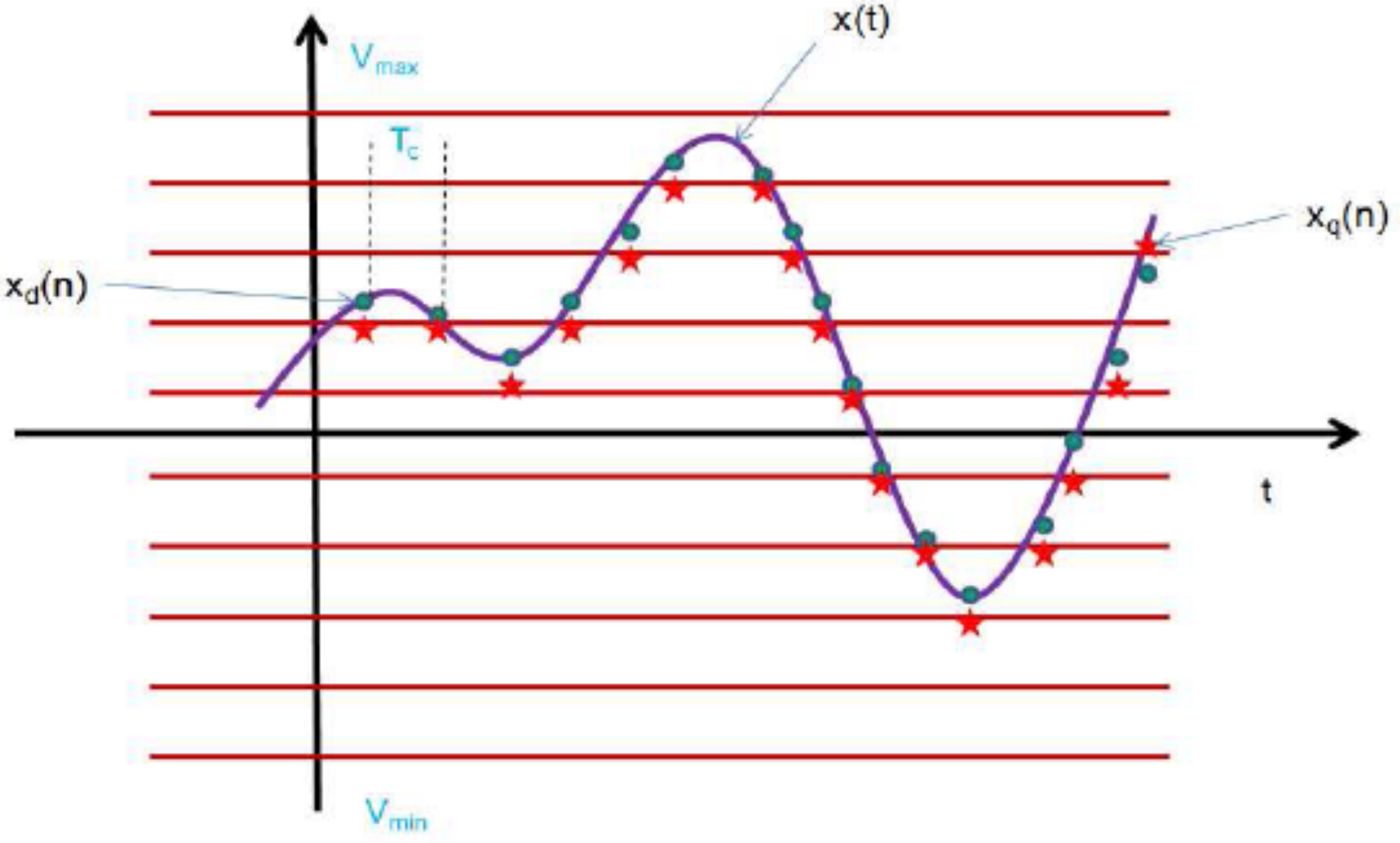
$$f_{alias} = f_c - f_{reale} < f_{reale}$$

Quantization

- Quantization is the procedure of constraining something from a continuous set of values (such as the real numbers) to a relatively small discrete set (such as the integers)

- Quantization replaces each real number with an approximation from a finite set of discrete values (levels)
  - values are represented as fixed-point words or floating-point words
  - common word-lengths are 8-bit (256 levels), 16-bit (65,536 levels), 32-bit (4.3 billion levels

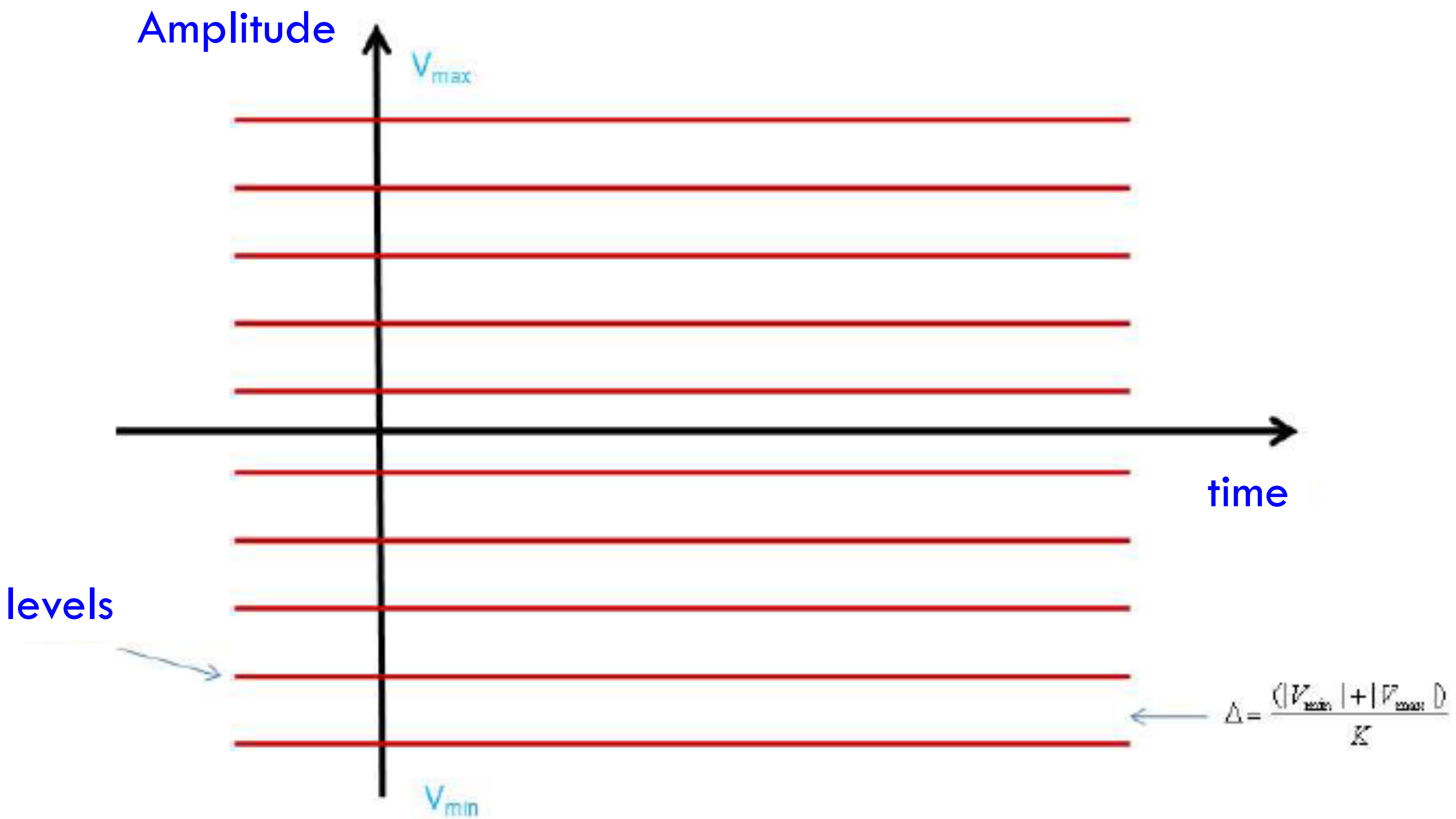
Example of quantization



$$e(n) = x_q(n) - x_d(n)$$

quantization error

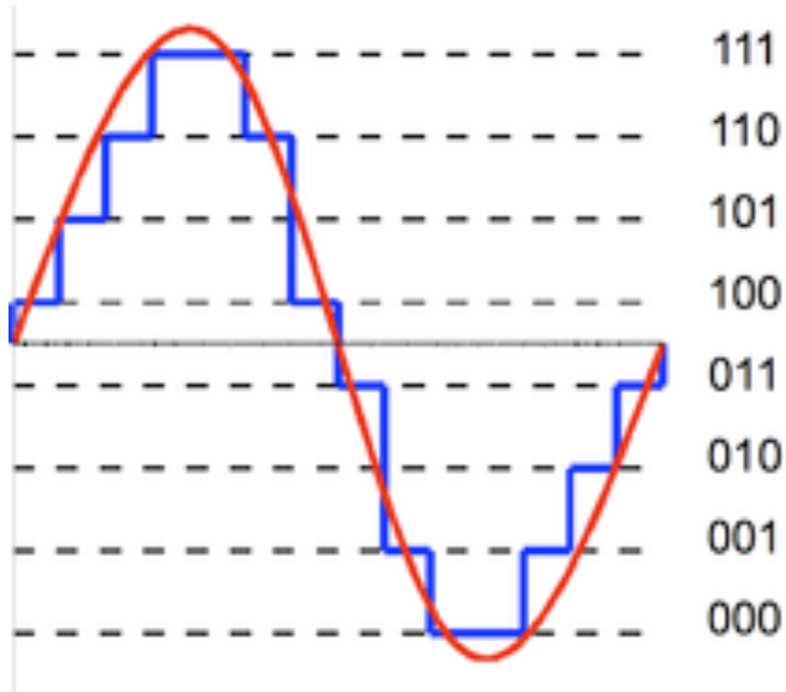
Coding



Uniform quantization

Coding

- With N bits $K = 2^N$ quantization levels are obtained
  - at each level a code of N bits can be associated



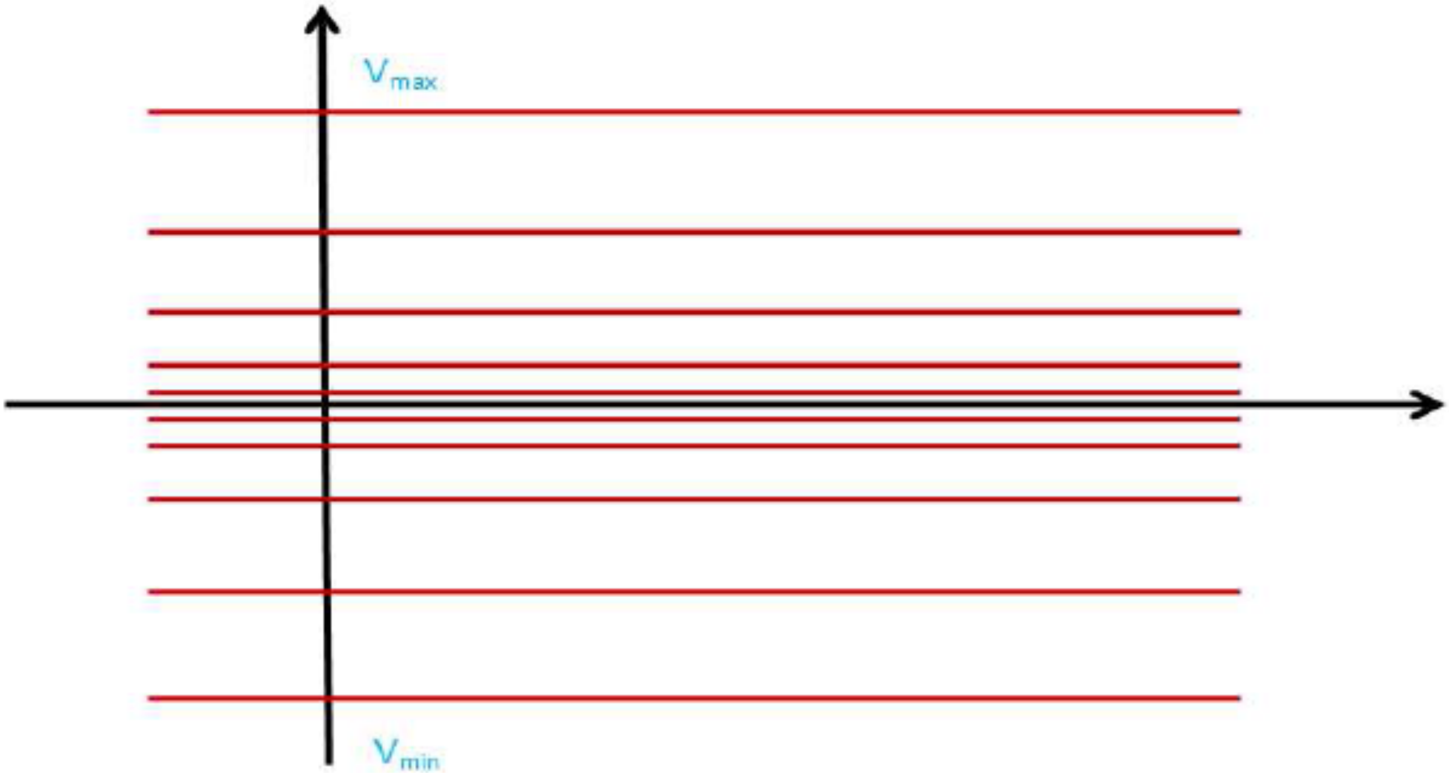
3-bit resolution with eight levels

Bit rate

- Bit rate
  - number of bits per second
  - product between the sampling frequency and the number of quantization bits

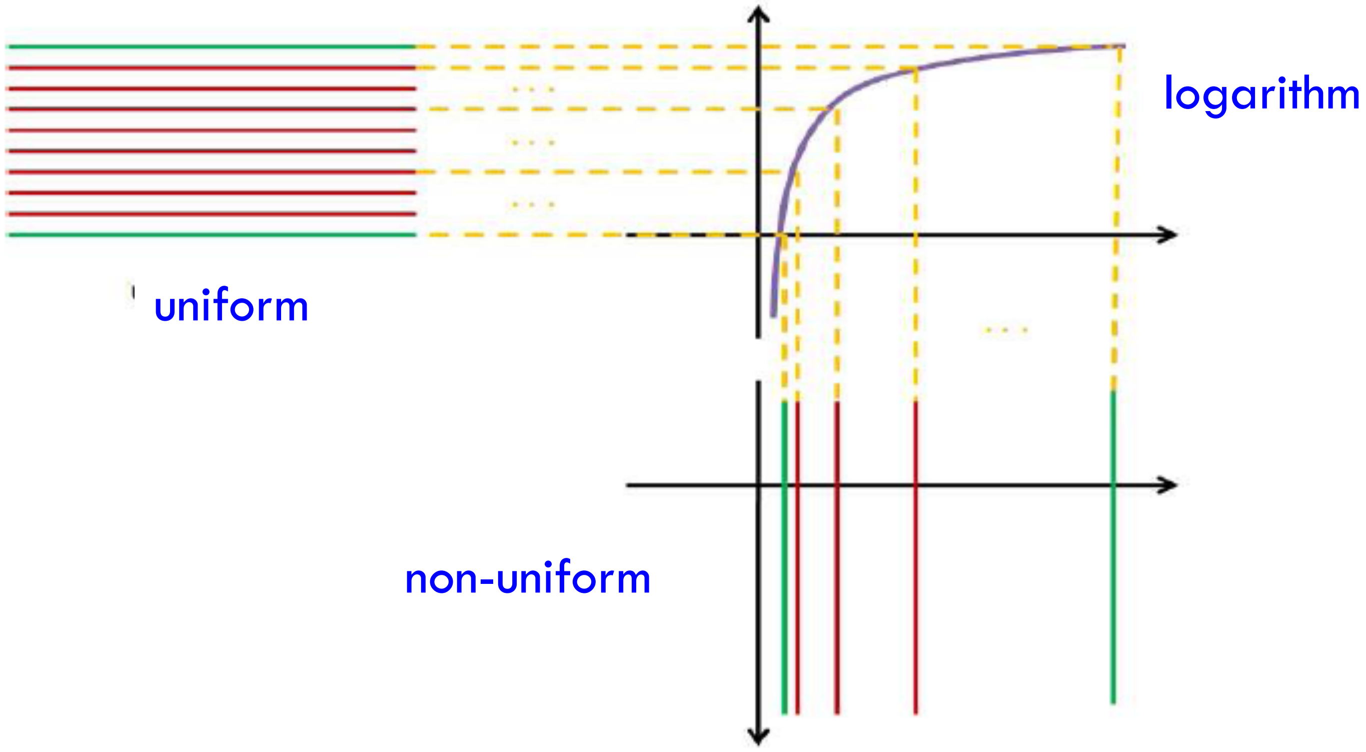
$$\text{bit rate} = f_c \cdot N$$

Non-linear quantization

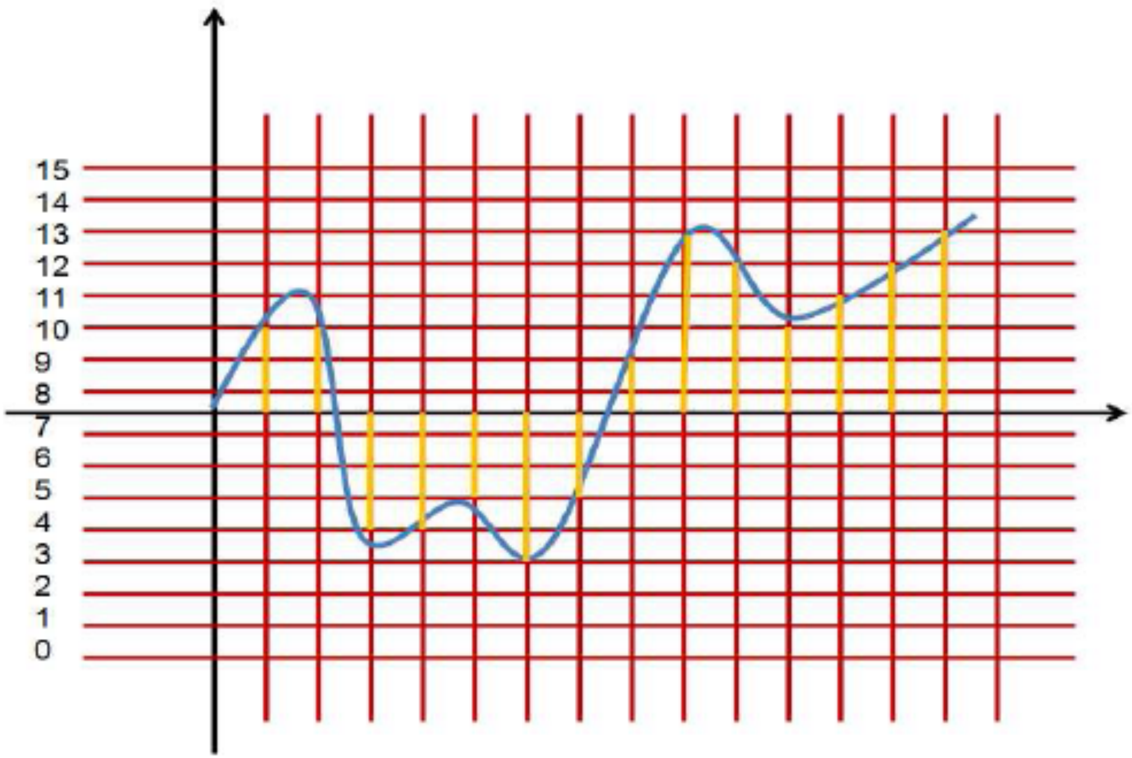


non-uniform quantization

Logarithmic quantization

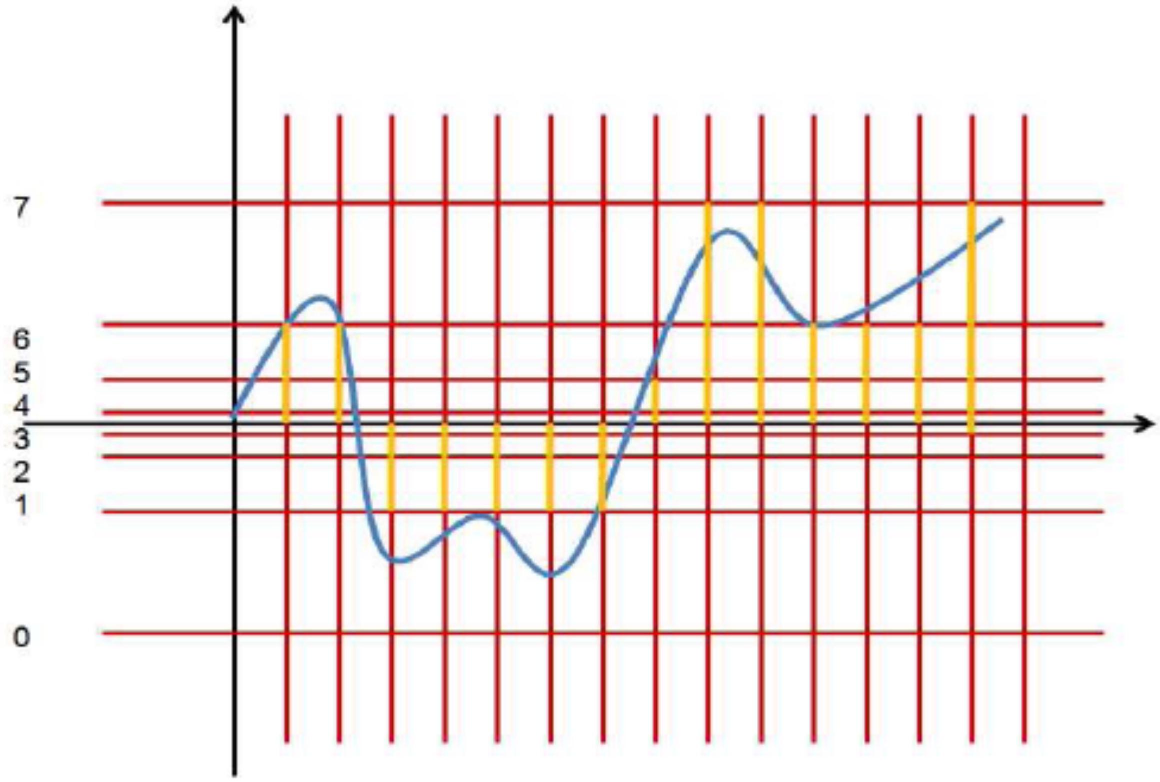


Example



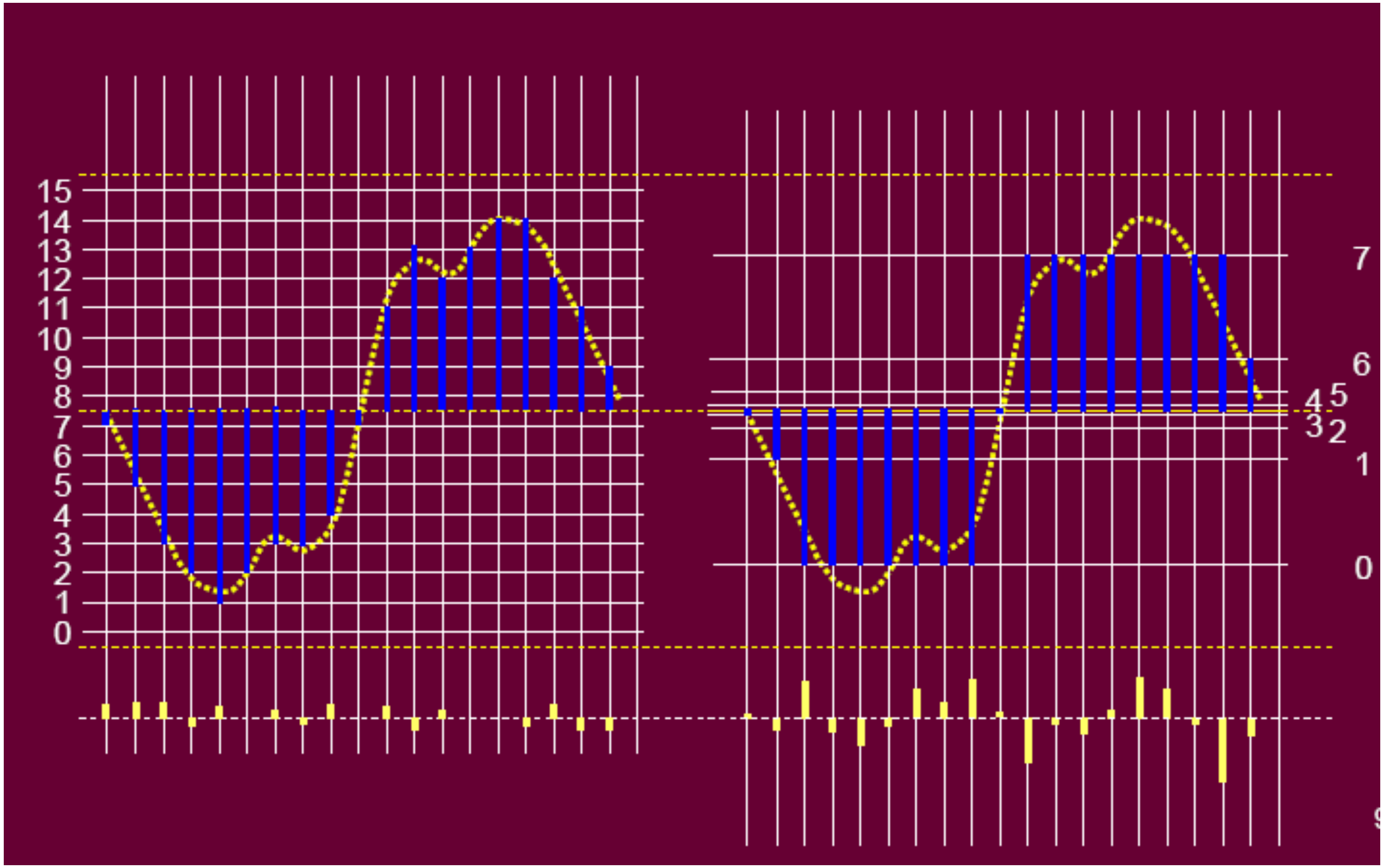
Uniform quantization

Example



non-uniform quantization

Comparison



4 bit linear

3 bit logarithmic

Comparison

- Quality
  - the dynamic range of the 8-bit logarithmic quantization corresponds to 13-14 bits linear quantizer
- Signal to Noise Rate (SNR)
  - a 8-bit logarithmic converter is better than a 8-bit linear at low amplitudes, but worse at high amplitudes

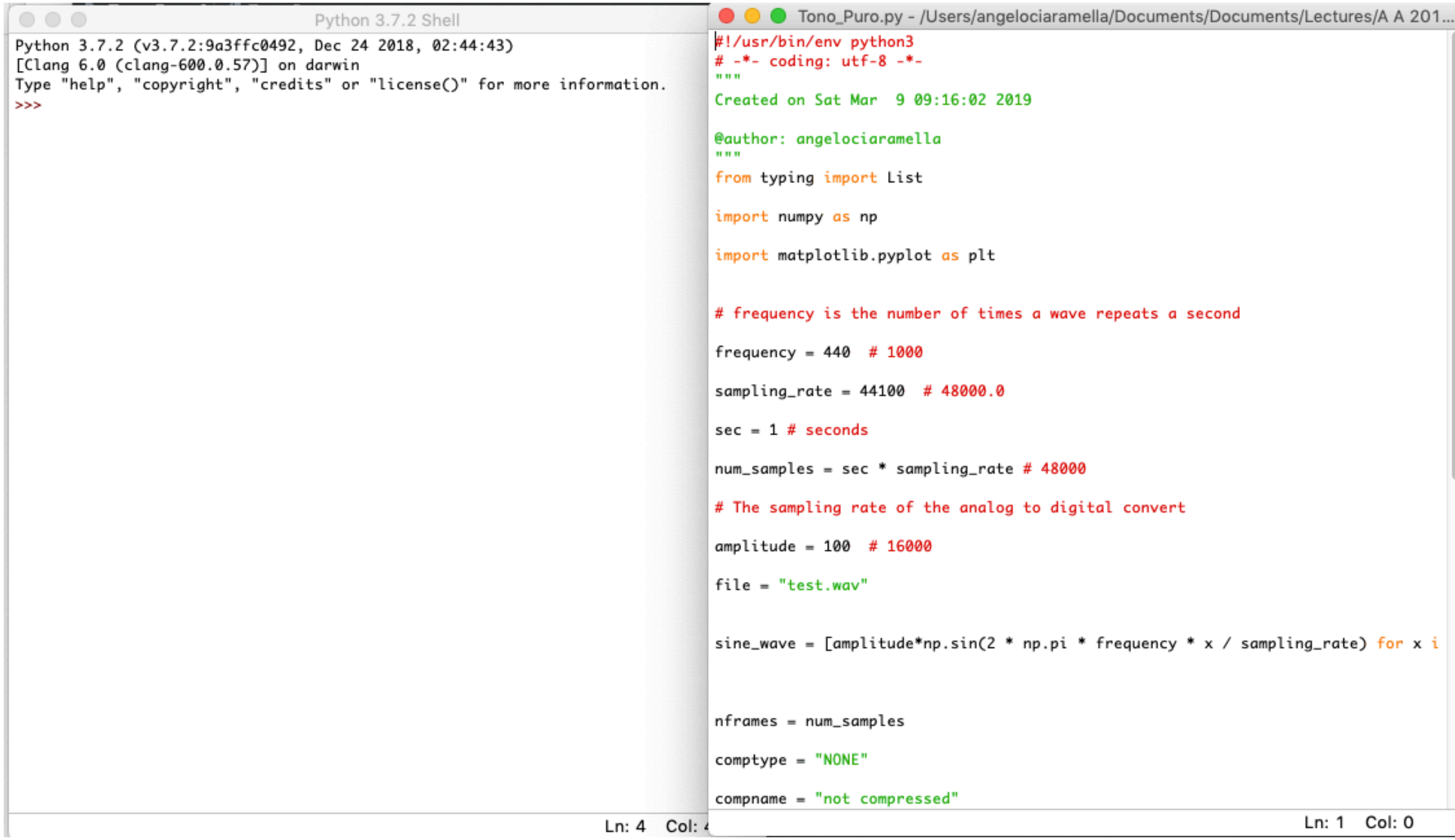
Python code (Tono_Puro.py)

```
import numpy as np

import sounddevice as sd

import matplotlib.pyplot as plt

# Signal frequency
frequency = 440

# Amplitude
amplitude = 100

# Sampling frequency
sampling_rate = 44100

# Seconds of the sequence
sec = 1

# Number of samples
num_samples = sec * sampling_rate

# time
t = np.arange(0,num_samples-1)

# sinusoidal wave
sine_wave = amplitude*np.sin(2 * np.pi * frequency * t / sampling_rate)
```

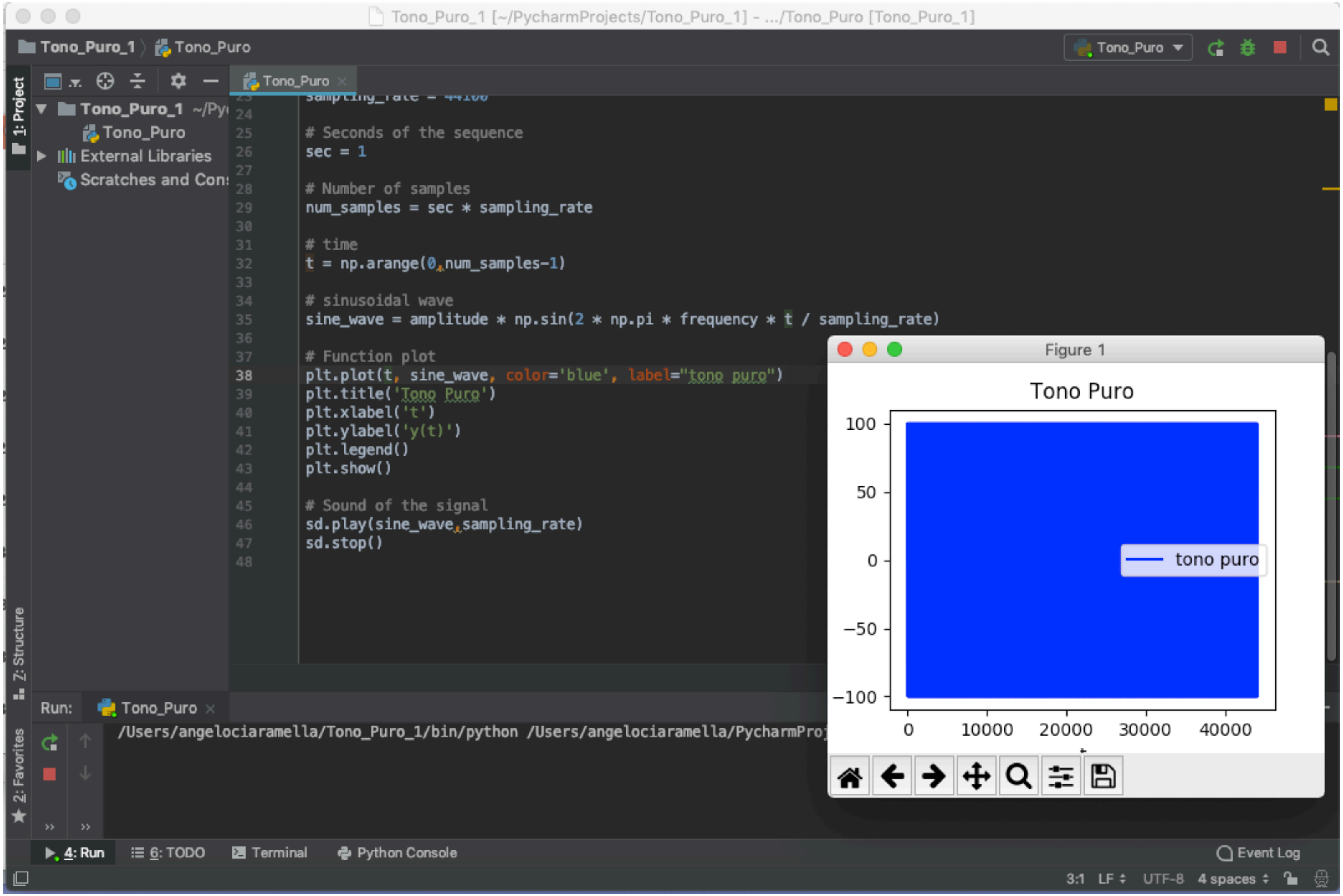
Python code

```
# Function plot
plt.plot(t/sampling_rate, sine_wave, color='blue', label="tono puro")
plt.title('Tono Puro')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.legend()
plt.show()


# Sound of the signal
sd.play(sine_wave,sampling_rate)
sd.stop()
```

Python IDE: IDLE

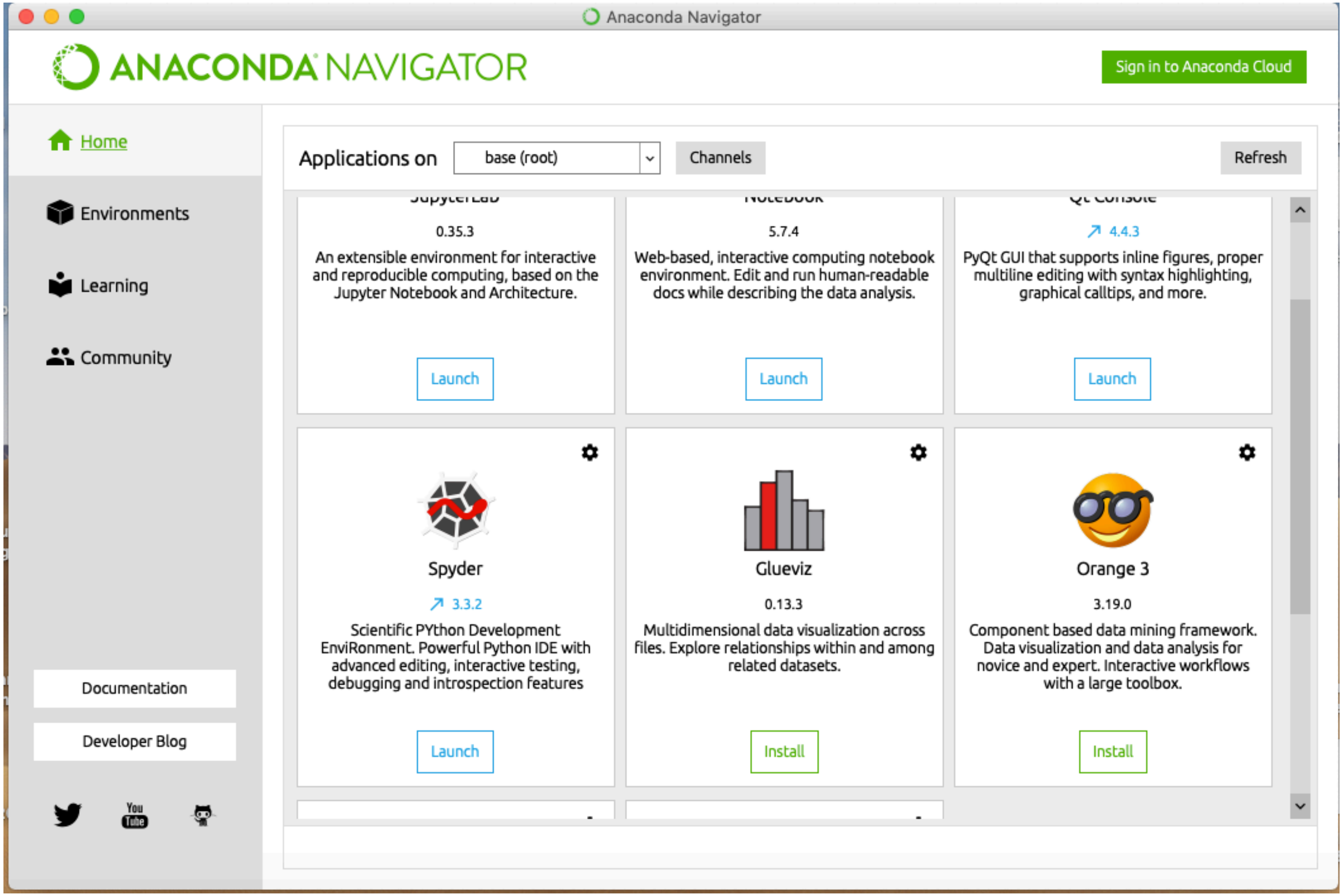


Python IDE: PyCharm

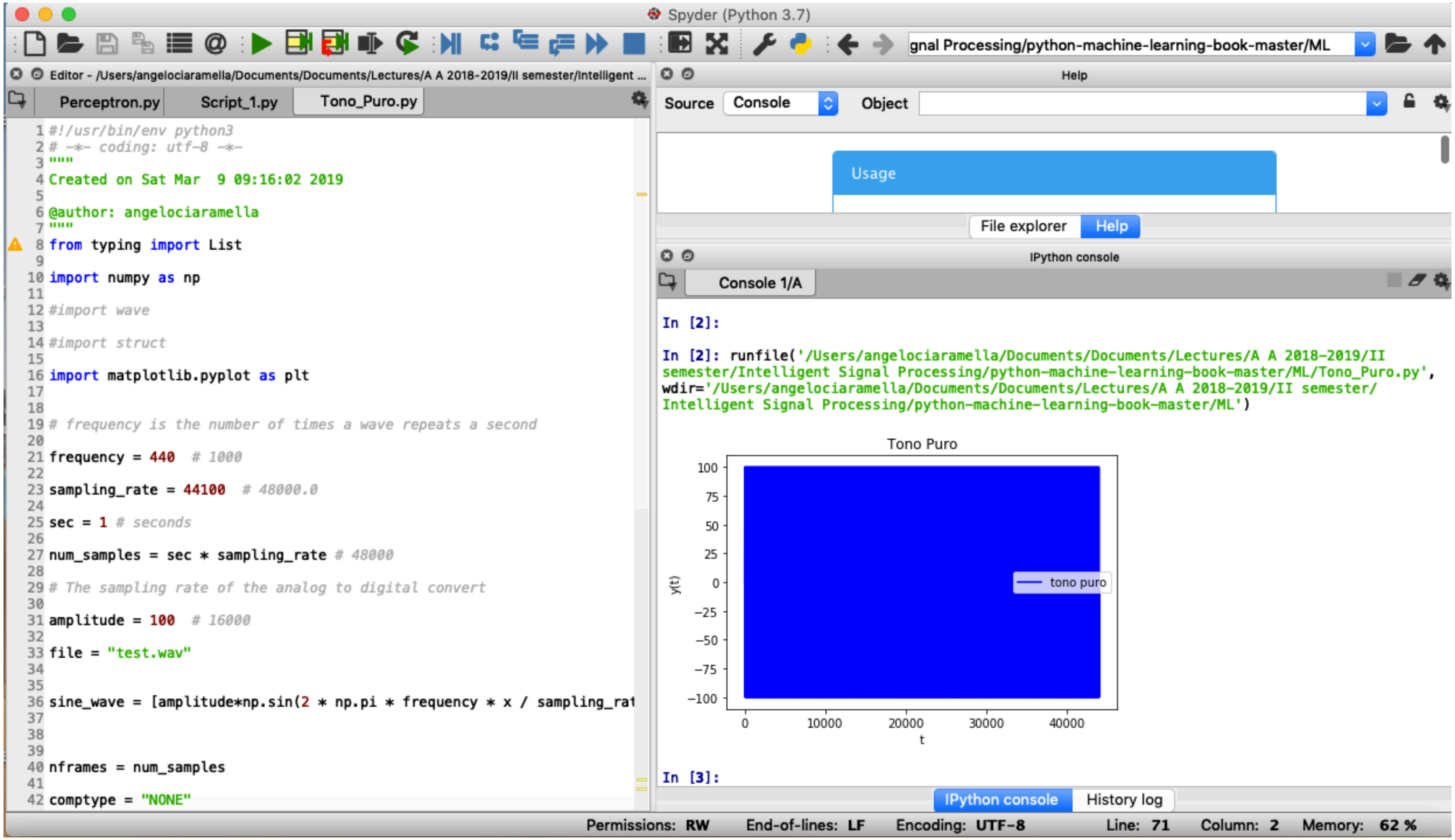


ISP – Audio Digitization

Python IDE: Anaconda



Python IDE: Spider



ISP – Audio Digitization

Homework

- Python code
  - Implementing a class by code on the Tono_Puro.py