

Digital Signal Processing

Module 5: Linear Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ **Module 5.1–5.3:** LTI systems and convolution; examples; stability
- ▶ **Module 5.4:** Convolution theorem
- ▶ **Module 5.5–5.6:** Ideal filters and their approximation
- ▶ **Module 5.7–5.8:** Realizable filters; implementation
- ▶ **Module 5.9–5.10:** Filter design

Digital Signal Processing

Module 5.1: Linear Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Linearity and time invariance
- ▶ Convolution

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Linearity and time invariance
- ▶ Convolution

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

A generic signal processing device



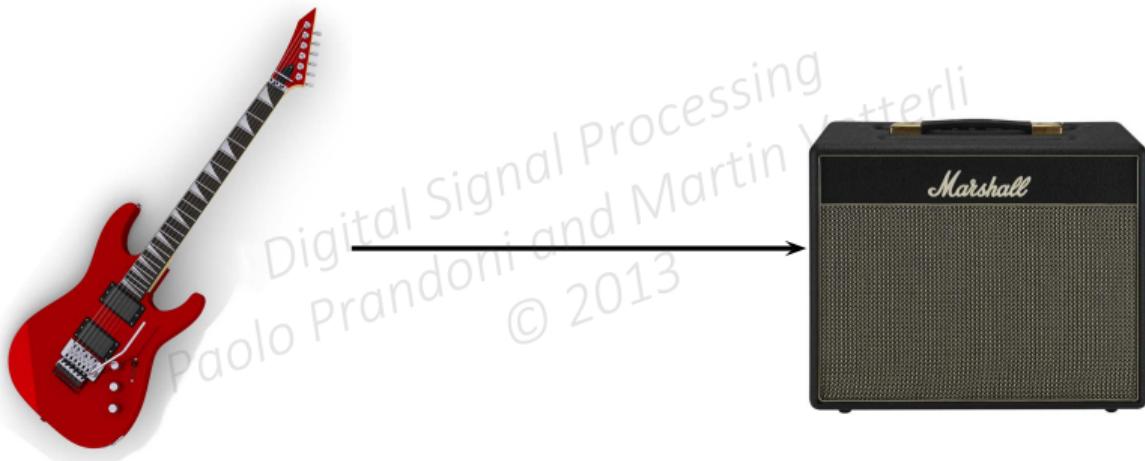
$$y[n] = \mathcal{H}\{x[n]\}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

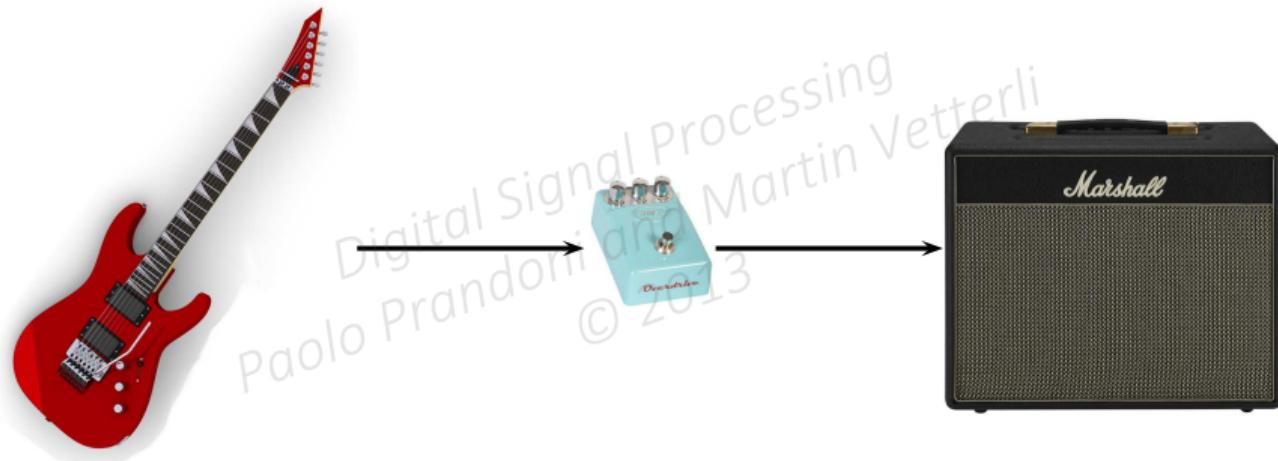
$$\mathcal{H}\{\alpha x_1[n] + \beta x_2[n]\} = \alpha \mathcal{H}\{x_1[n]\} + \beta \mathcal{H}\{x_2[n]\}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Linearity



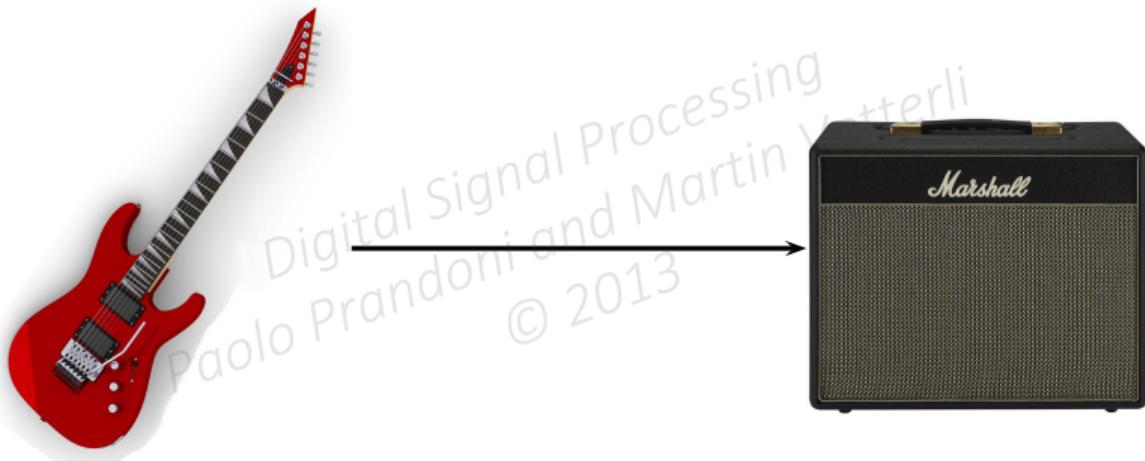
(Non) Linearity



$$y[n] = \mathcal{H}\{x[n]\} \iff \mathcal{H}\{x[n - n_0]\} = y[n - n_0]$$

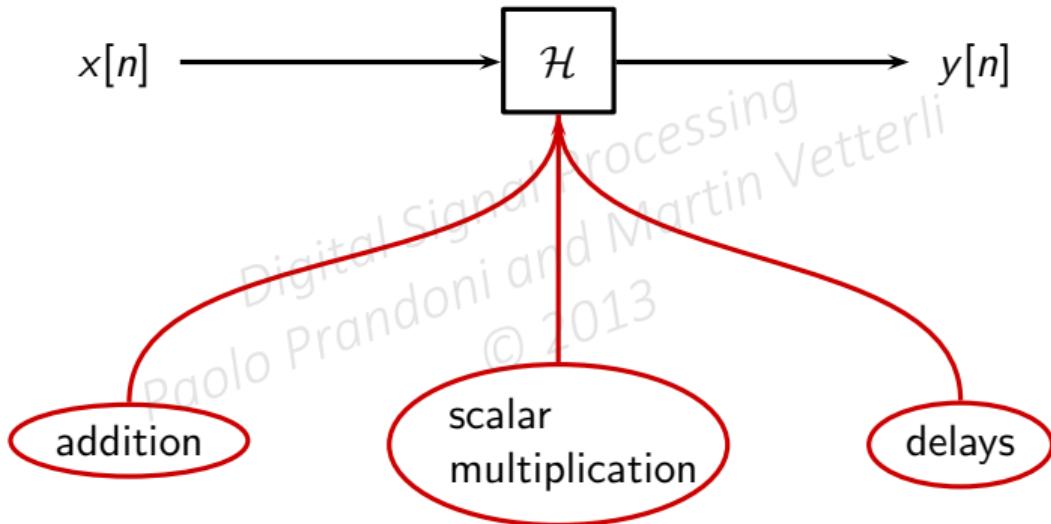
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Time invariance



Time (in)variance





$$y[n] = H(x[n], x[n-1], x[n-2], \dots, y[n-1], y[n-2], \dots)$$

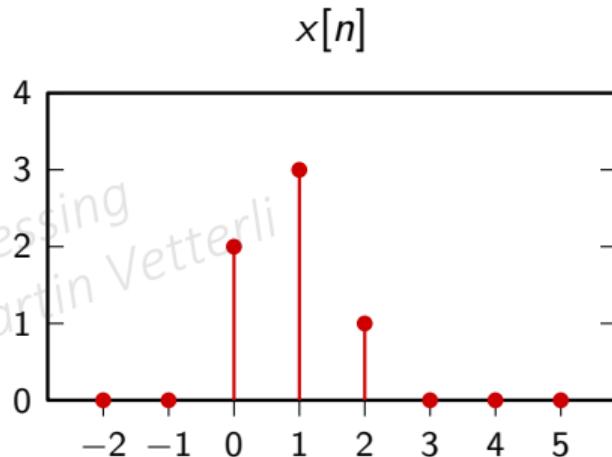
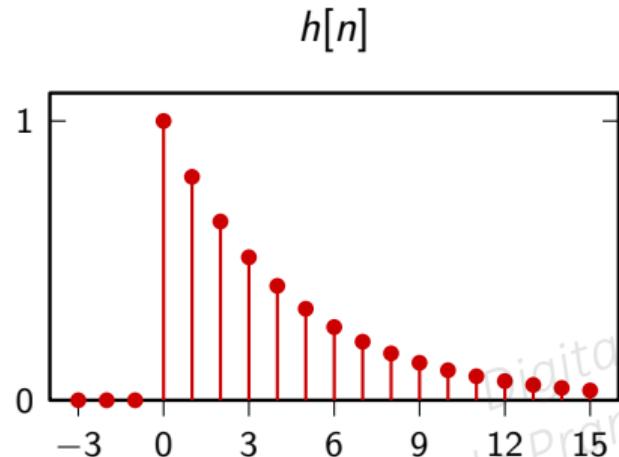
with $H(\cdot)$ a linear function of its arguments

$$h[n] = \mathcal{H}\{\delta[n]\}$$

Fundamental result: impulse response fully characterizes the LTI system!

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Example



$$h[n] = \alpha^n u[n]$$

$$x[n] = \begin{cases} 2 & n = 0 \\ 3 & n = 1 \\ 1 & n = 2 \\ 0 & \text{otherwise} \end{cases}$$

Example

- ▶ $x[n] = 2\delta[n] + 3\delta[n - 1] + \delta[n - 2]$
- ▶ we know the impulse response $h[n] = \mathcal{H}\{\delta[n]\}$
- ▶ compute $y[n] = \mathcal{H}\{x[n]\}$ exploiting linearity and time-invariance

Signal Processing
Digital Signal Processing
Paolo Pogdoni and Martin Vetterli
© 2013

Example

- ▶ $x[n] = 2\delta[n] + 3\delta[n - 1] + \delta[n - 2]$
- ▶ we know the impulse response $h[n] = \mathcal{H}\{\delta[n]\}$;
- ▶ compute $y[n] = \mathcal{H}\{x[n]\}$ exploiting linearity and time-invariance

Example

- ▶ $x[n] = 2\delta[n] + 3\delta[n - 1] + \delta[n - 2]$
- ▶ we know the impulse response $h[n] = \mathcal{H}\{\delta[n]\}$;
- ▶ compute $y[n] = \mathcal{H}\{x[n]\}$ exploiting linearity and time-invariance

Example

$$\begin{aligned}y[n] &= \mathcal{H}\{2\delta[n] + 3\delta[n - 1] + \delta[n - 2]\} \\&= 2h[n] + 3h[n - 1] + h[n - 2]\end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

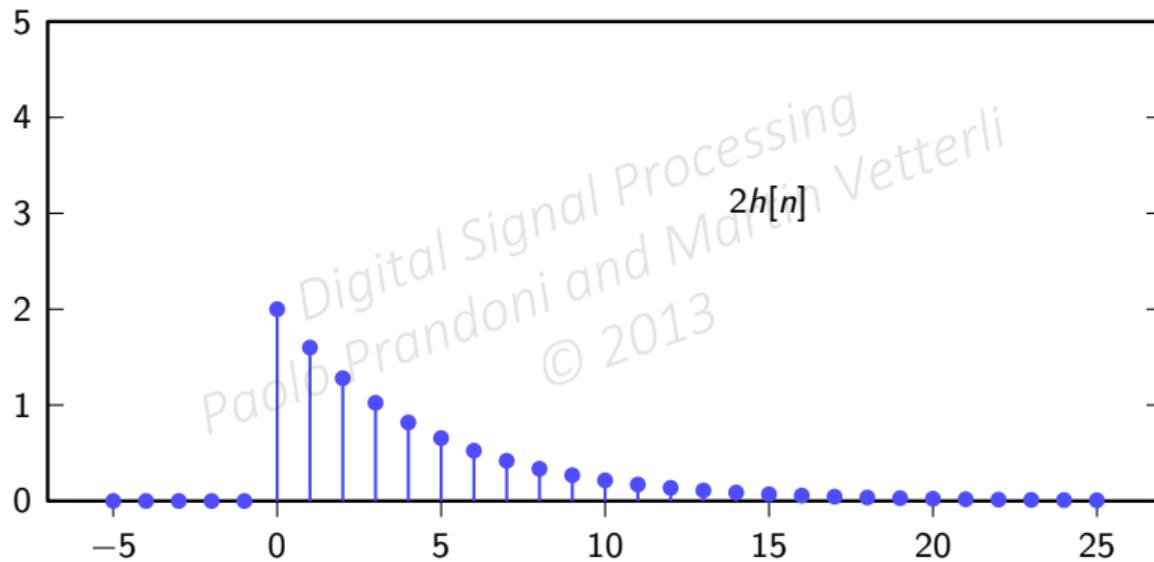
Example

$$\begin{aligned}y[n] &= \mathcal{H}\{2\delta[n] + 3\delta[n - 1] + \delta[n - 2]\} \\&= 2\mathcal{H}\{\delta[n]\} + 3\mathcal{H}\{\delta[n - 1]\} + \mathcal{H}\{\delta[n - 2]\} \\&= 2h[n] + 3h[n - 1] + h[n - 2]\end{aligned}$$

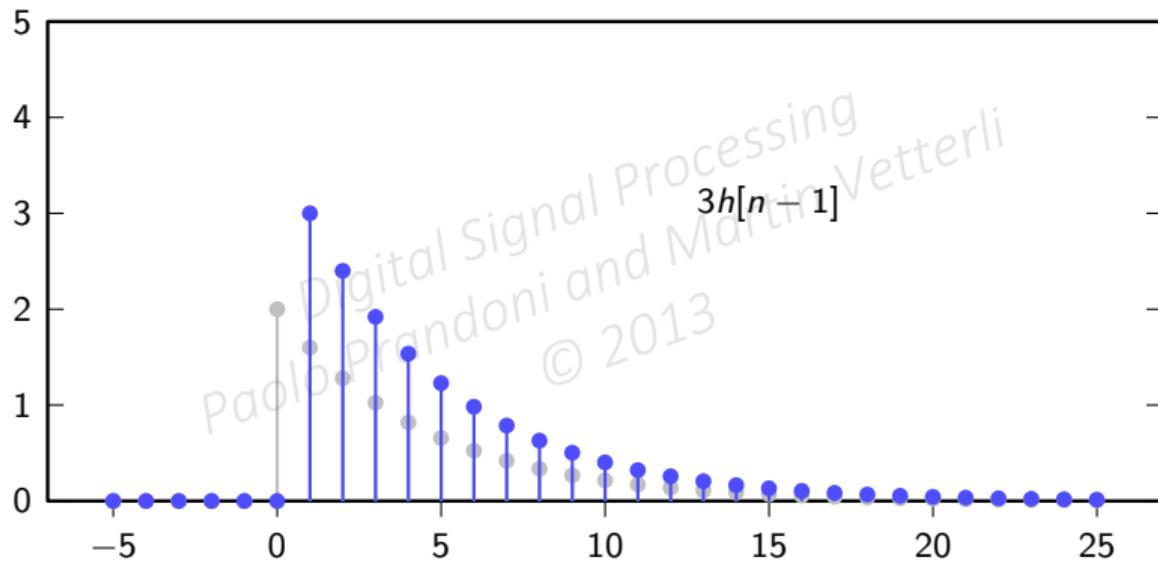
Example

$$\begin{aligned}y[n] &= \mathcal{H}\{2\delta[n] + 3\delta[n - 1] + \delta[n - 2]\} \\&= 2\mathcal{H}\{\delta[n]\} + 3\mathcal{H}\{\delta[n - 1]\} + \mathcal{H}\{\delta[n - 2]\} \\&= 2h[n] + 3h[n - 1] + h[n - 2]\end{aligned}$$

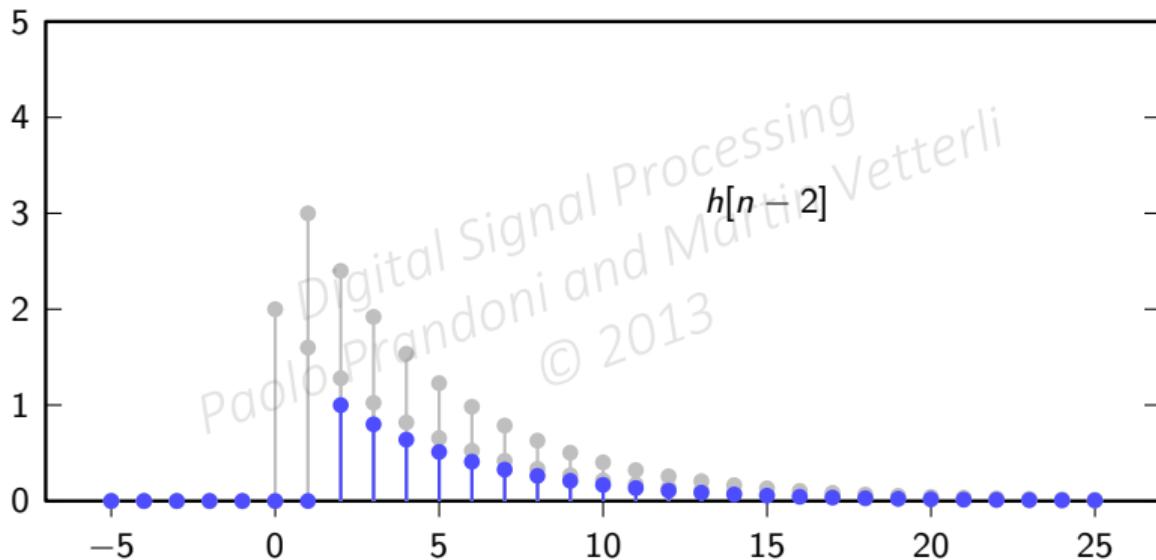
Example



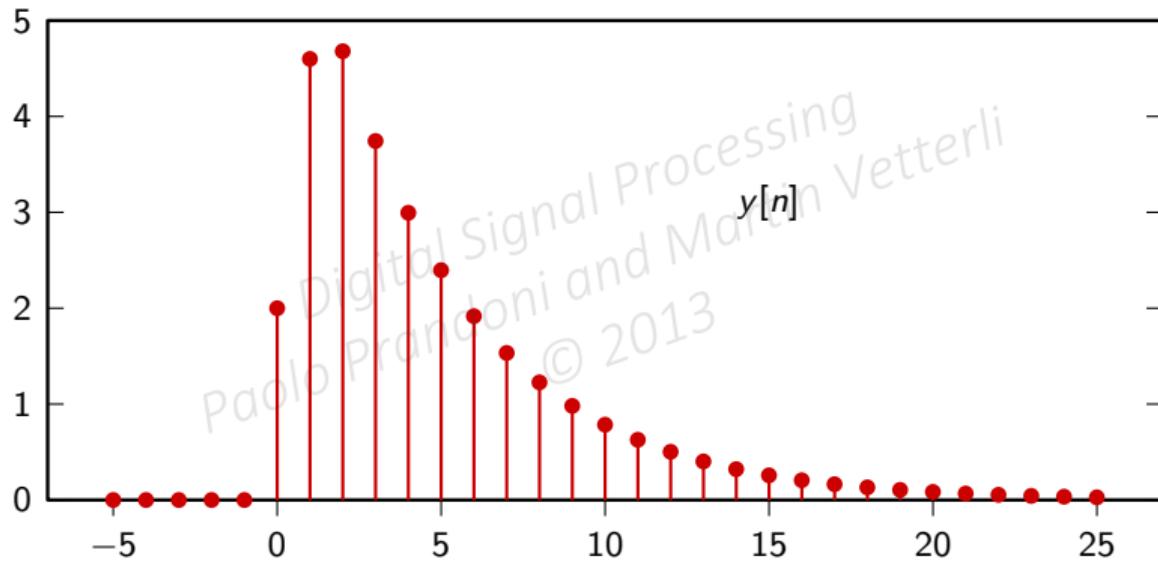
Example



Example



Example



We can always write (remember Module 3.2):

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

by linearity and time invariance:
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \\ &= x[n] * h[n] \end{aligned}$$

We can always write (remember Module 3.2):

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - k]$$

by linearity and time invariance:

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} x[k]h[n - k] \\ &= x[n] * h[n] \end{aligned}$$

Performing the convolution algorithmically

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Ingredients:

- ▶ a sequence $x[m]$
- ▶ a second sequence $h[m]$

- Digital Signal Processing
paolo Prandoni and Martin Vetterli
The recipe:
© 2013
- ▶ time-reverse $h[m]$
 - ▶ at each step n (from $-\infty$ to ∞):
 - center the time-reversed $h[m]$ in n
(i.e. shift by $-n$)
 - compute the inner product

Performing the convolution algorithmically

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Ingredients:

- ▶ a sequence $x[m]$
- ▶ a second sequence $h[m]$

- Digital Signal Processing
Paolo Prandoni and Martin Vetterli
The recipe:
© 2013
- ▶ time-reverse $h[m]$
 - ▶ at each step n (from $-\infty$ to ∞):
 - center the time-reversed $h[m]$ in n
(i.e. shift by $-n$)
 - compute the inner product

Performing the convolution algorithmically

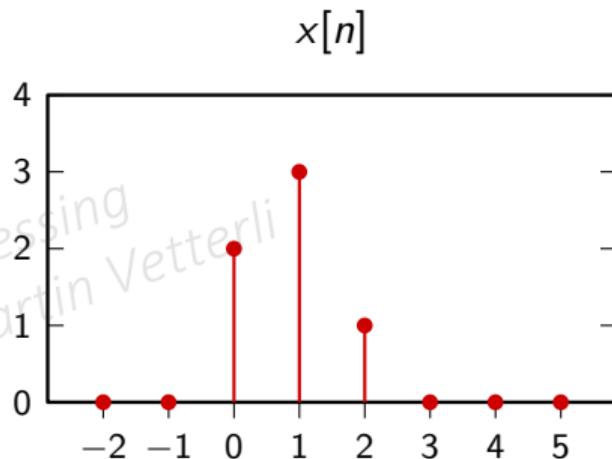
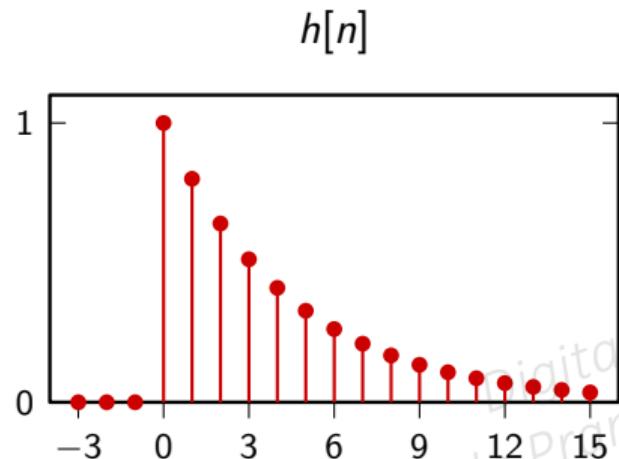
$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

The recipe:

Ingredients:

- ▶ a sequence $x[m]$
 - ▶ a second sequence $h[m]$
- ▶ time-reverse $h[m]$
 - ▶ at each step n (from $-\infty$ to ∞):
 - center the time-reversed $h[m]$ in n
(i.e. shift by $-n$)
 - compute the inner product

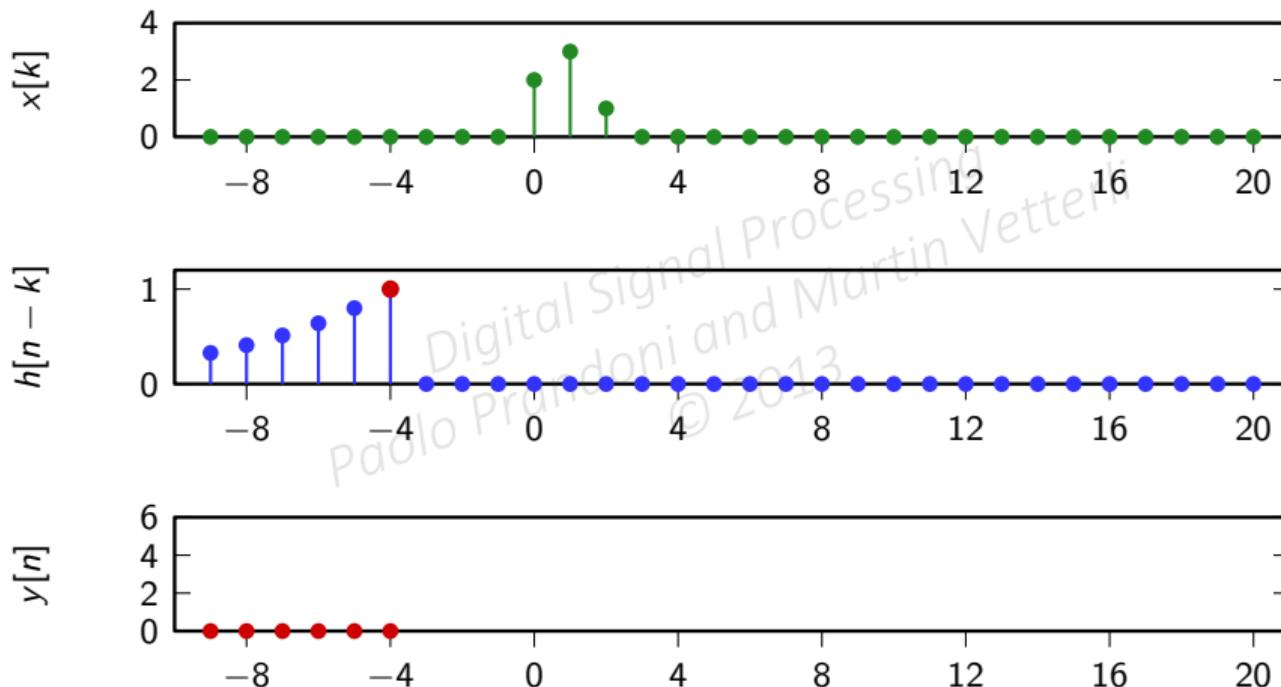
Same example, different perspective



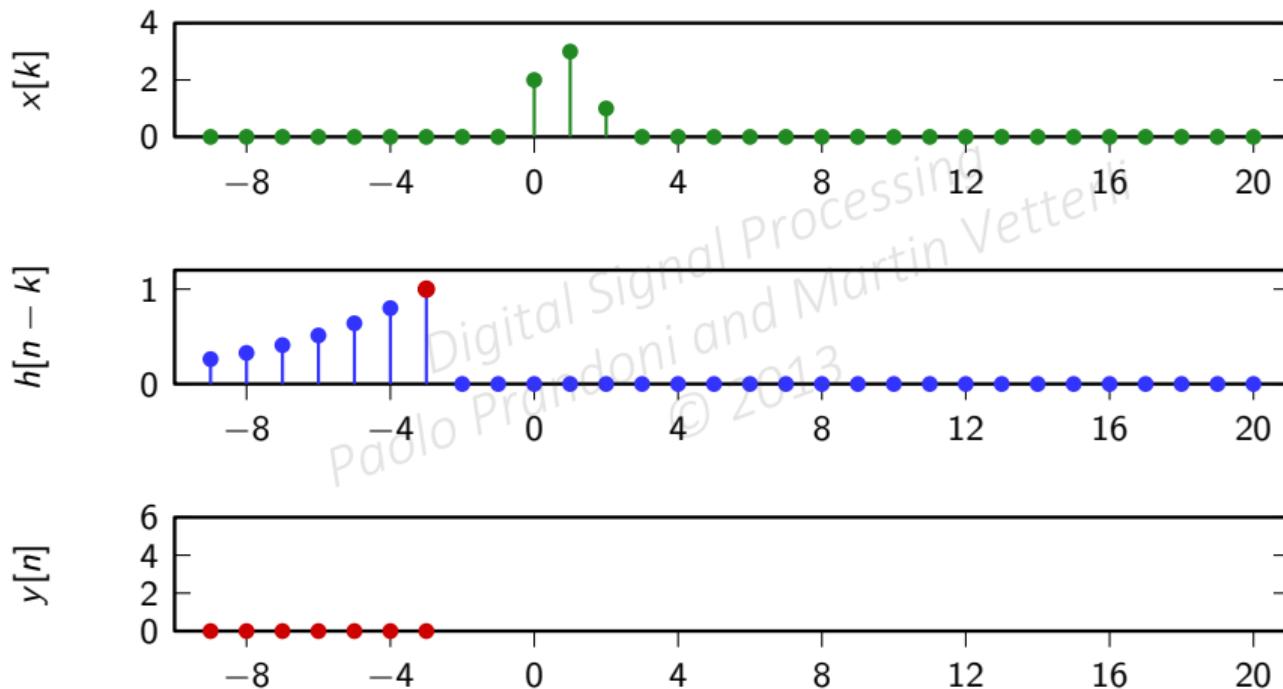
$$h[n] = \alpha^n u[n]$$

$$x[n] = \begin{cases} 2 & n = 0 \\ 3 & n = 1 \\ 1 & n = 2 \\ 0 & \text{otherwise} \end{cases}$$

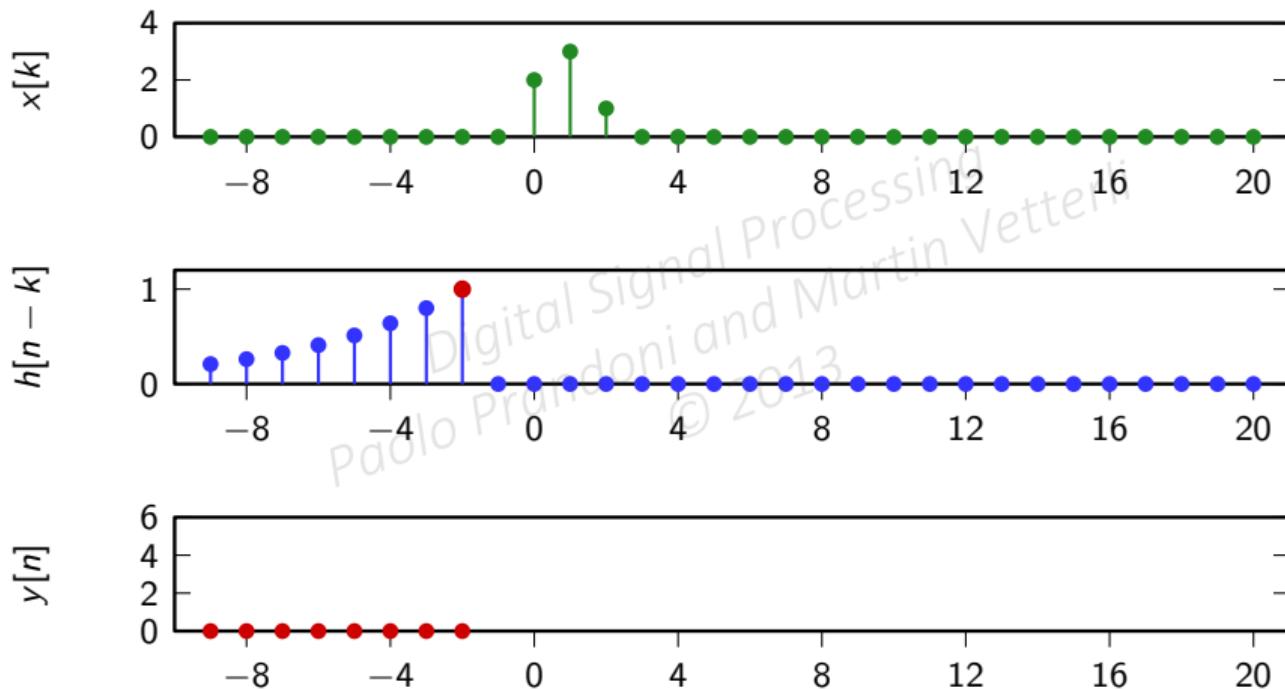
Convolution example



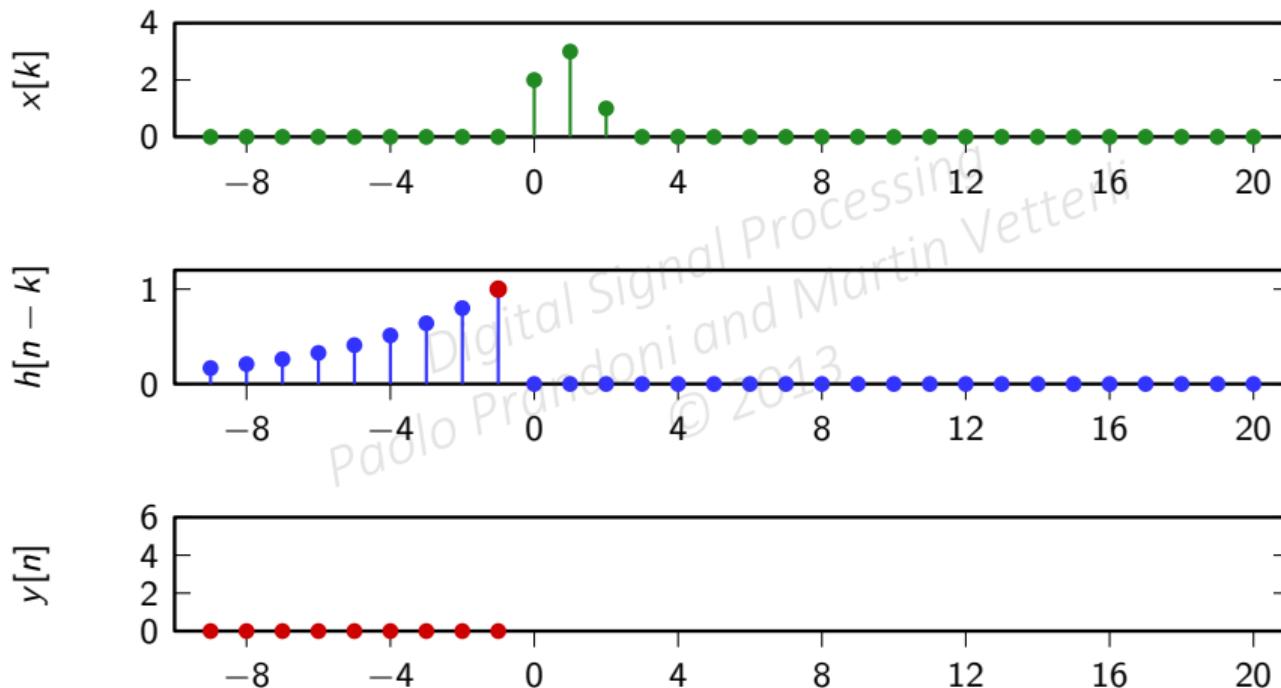
Convolution example



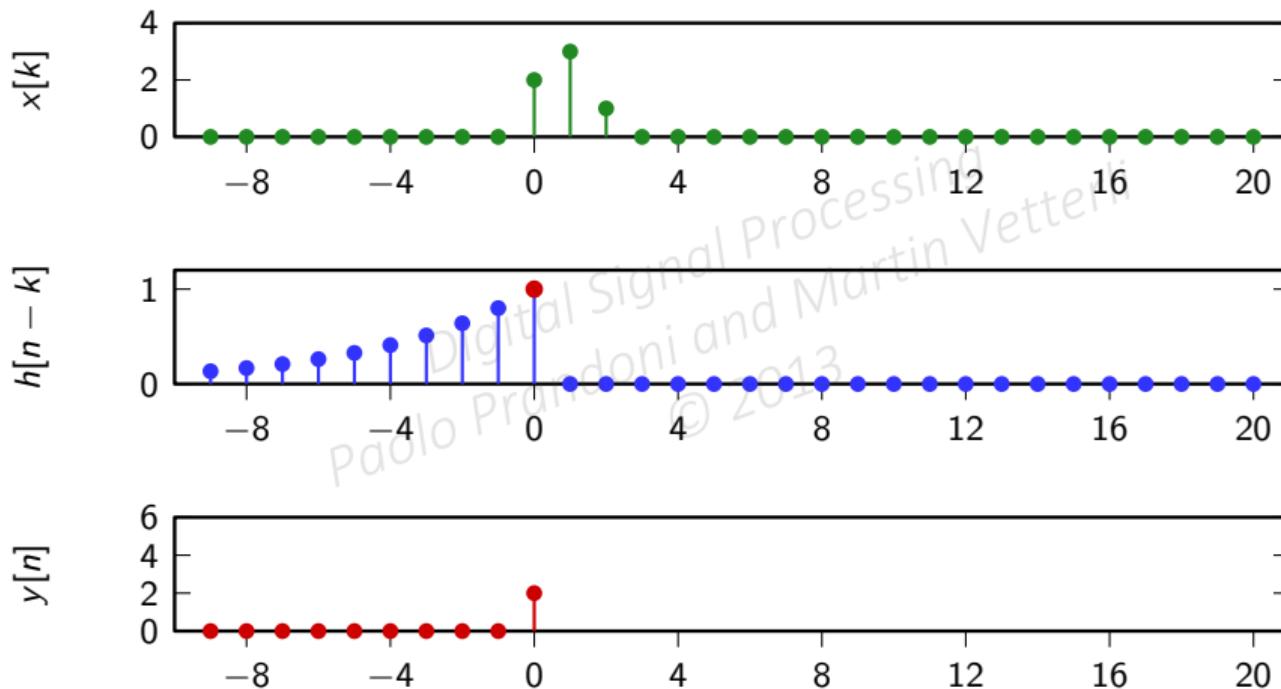
Convolution example



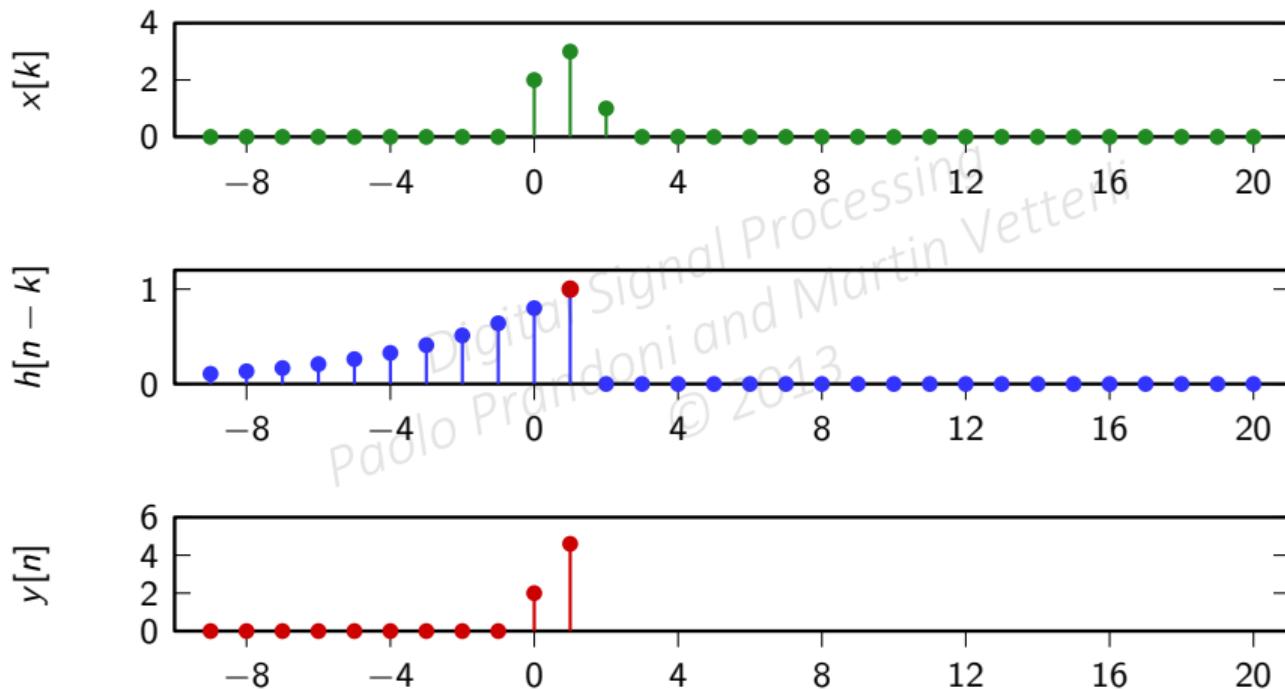
Convolution example



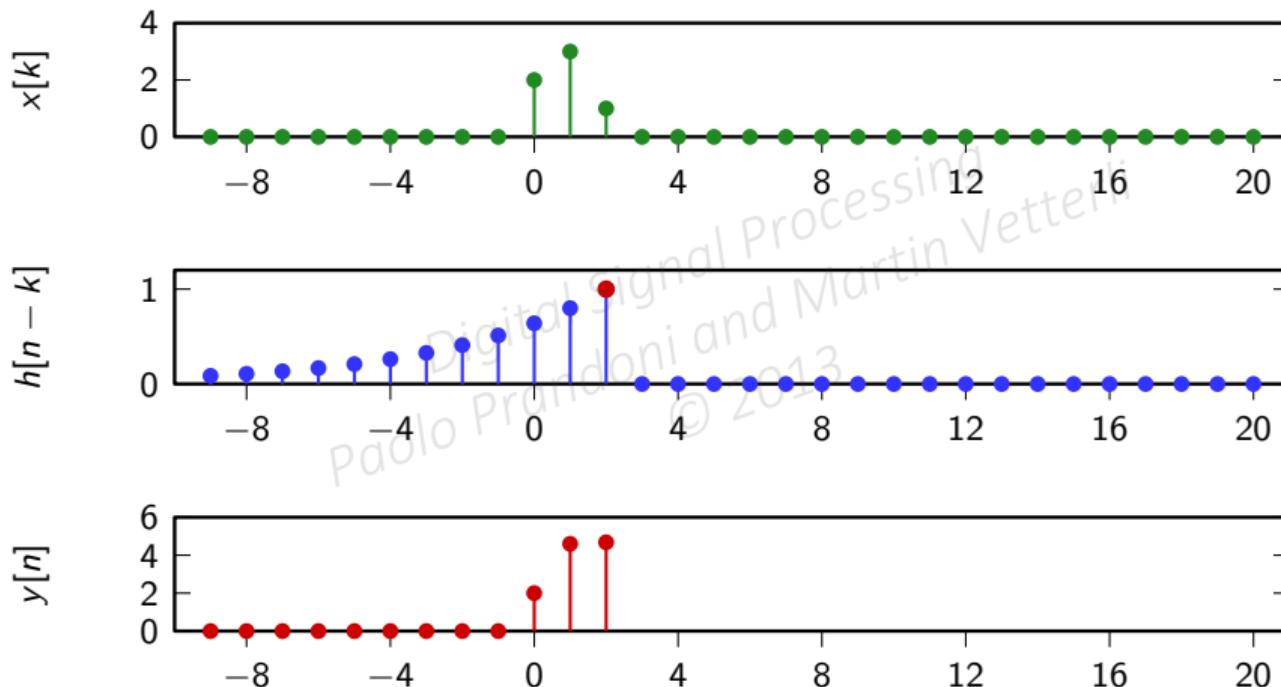
Convolution example



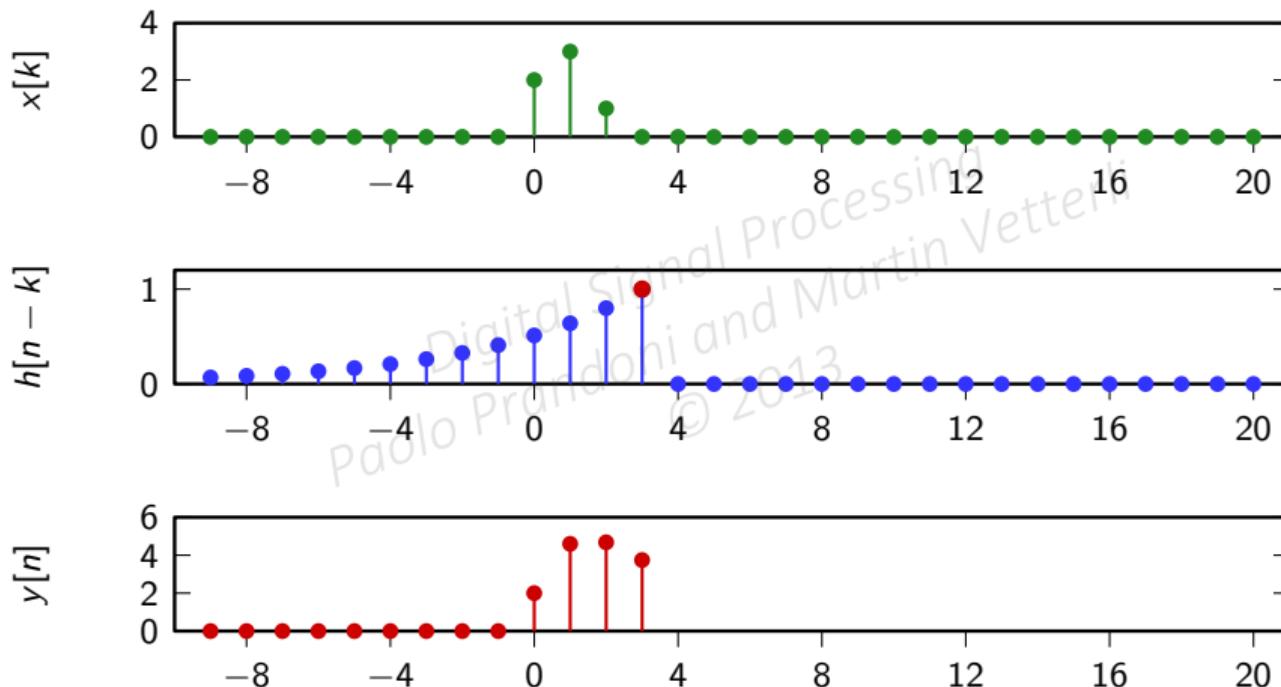
Convolution example



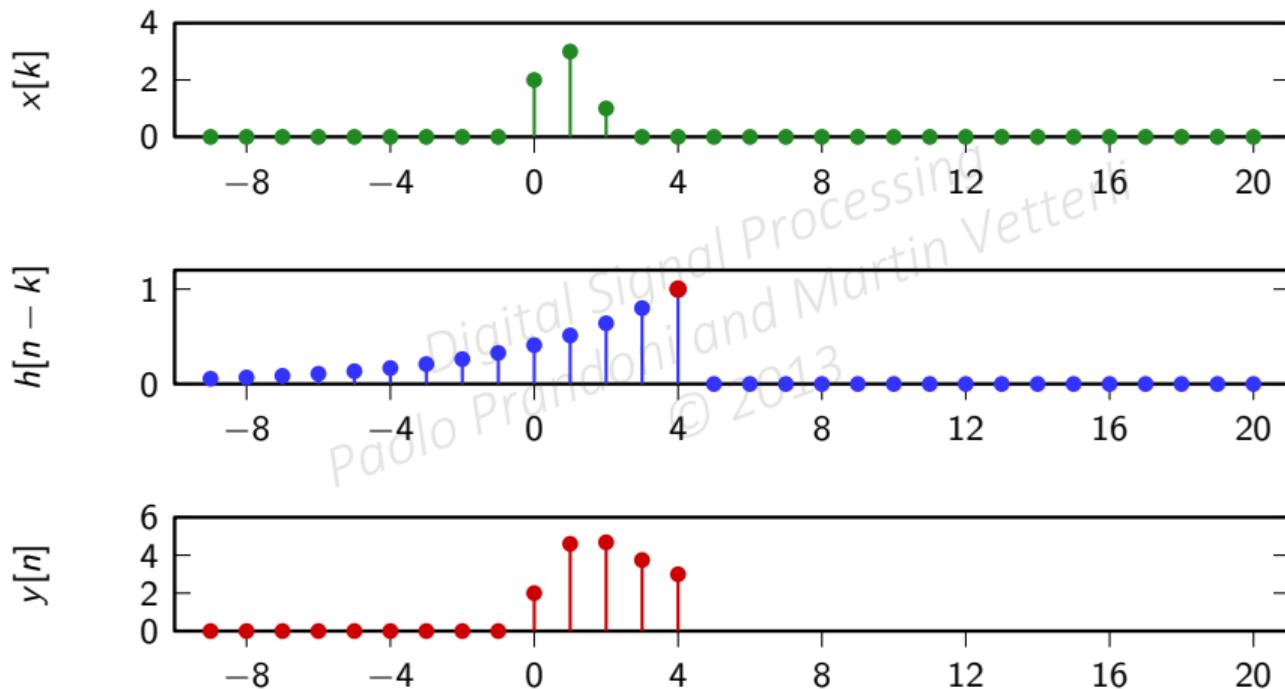
Convolution example



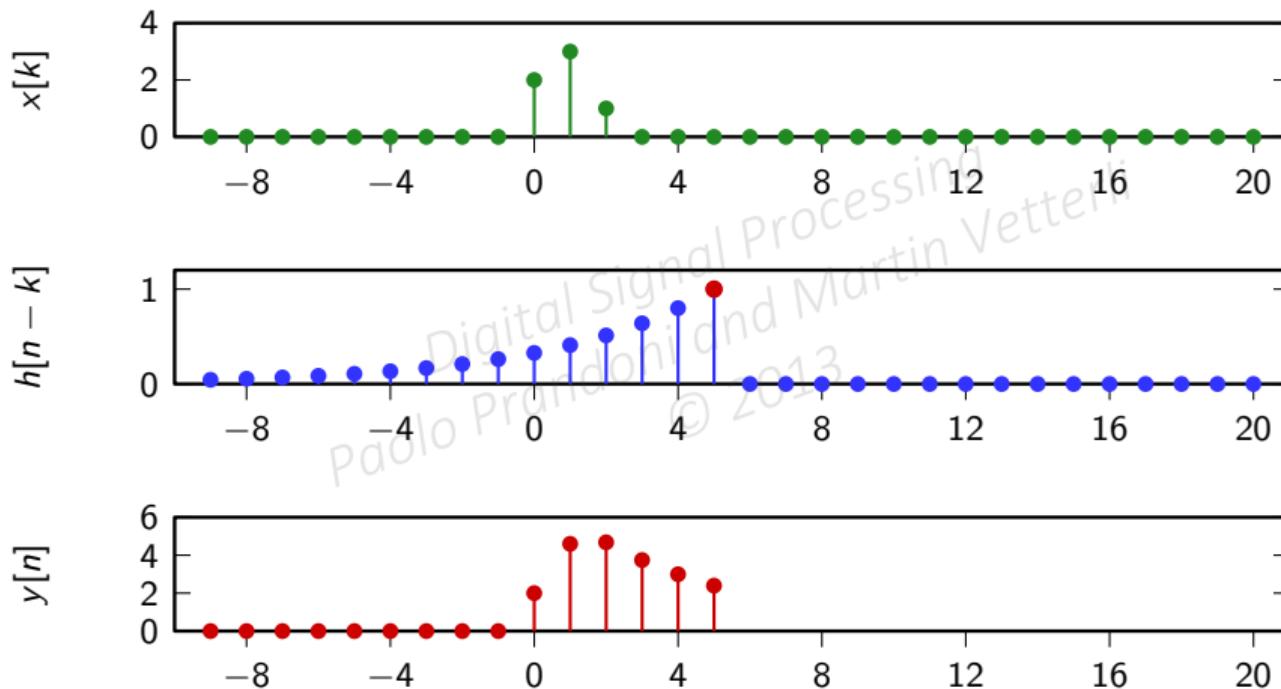
Convolution example



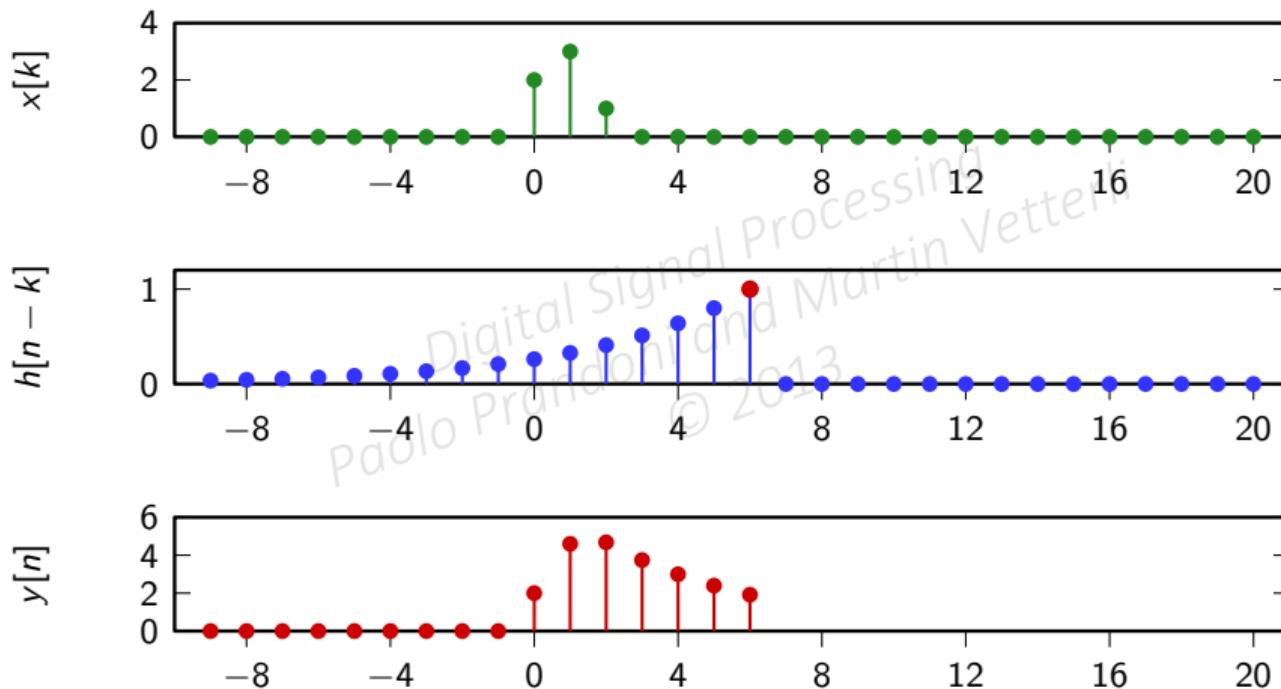
Convolution example



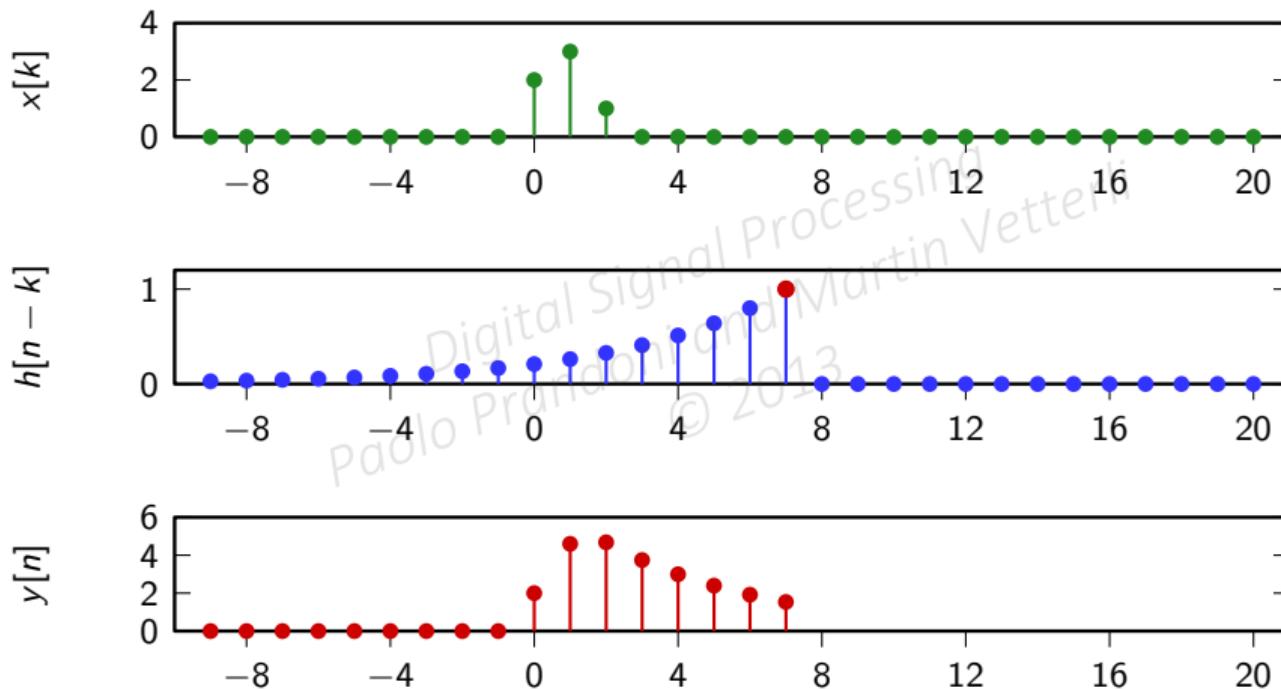
Convolution example



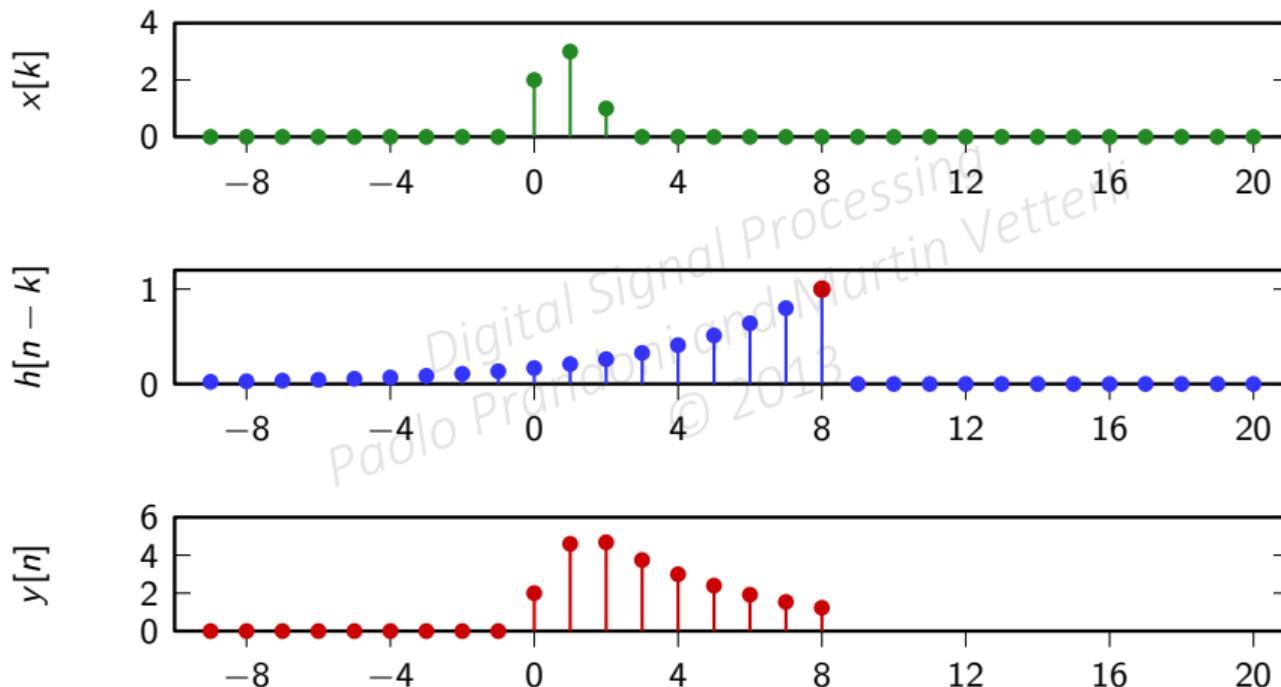
Convolution example



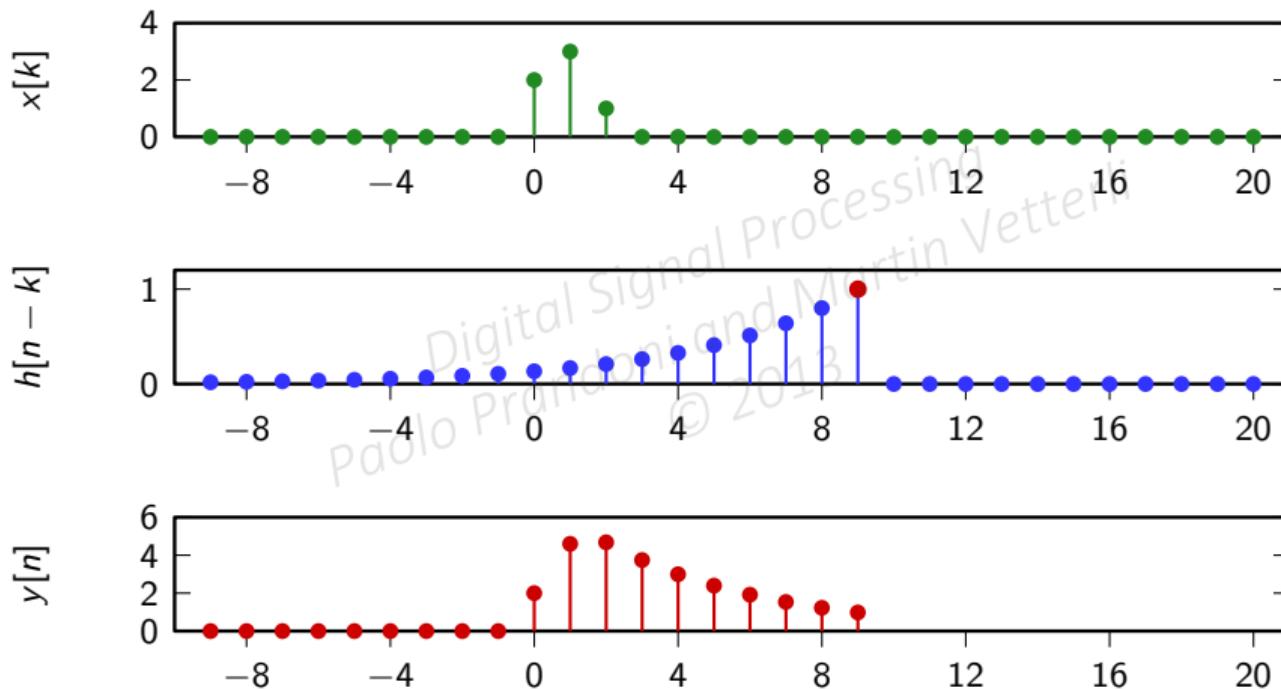
Convolution example



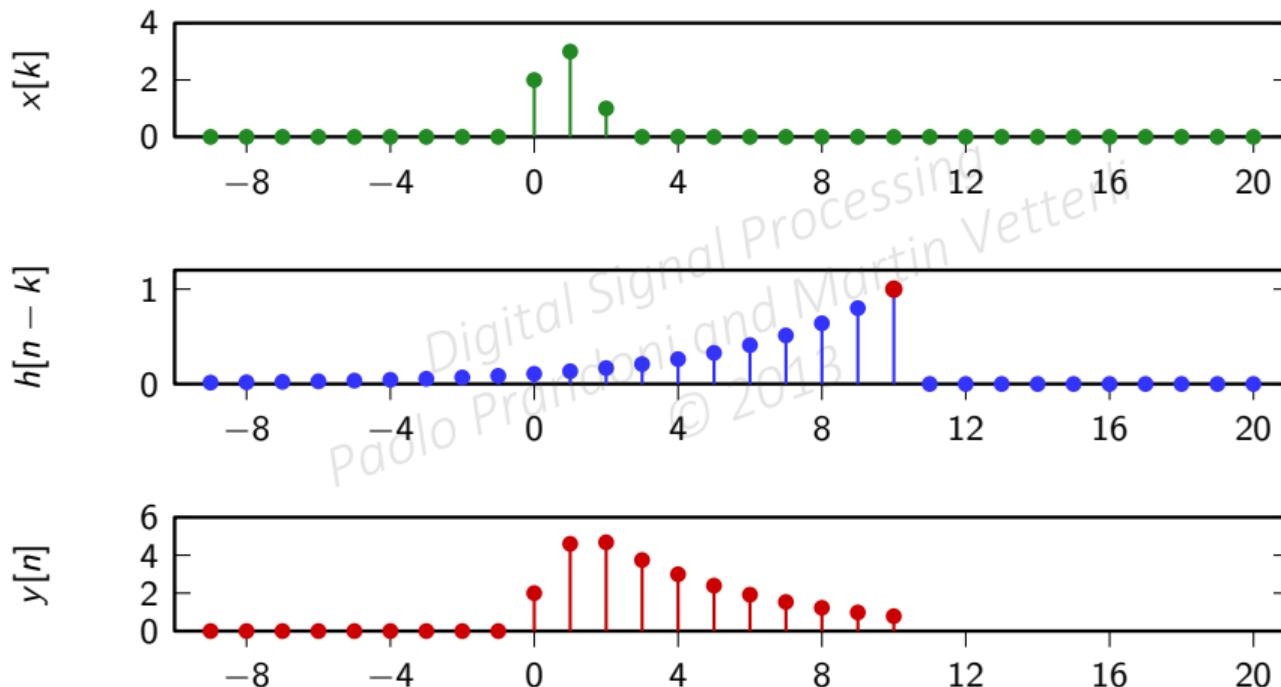
Convolution example



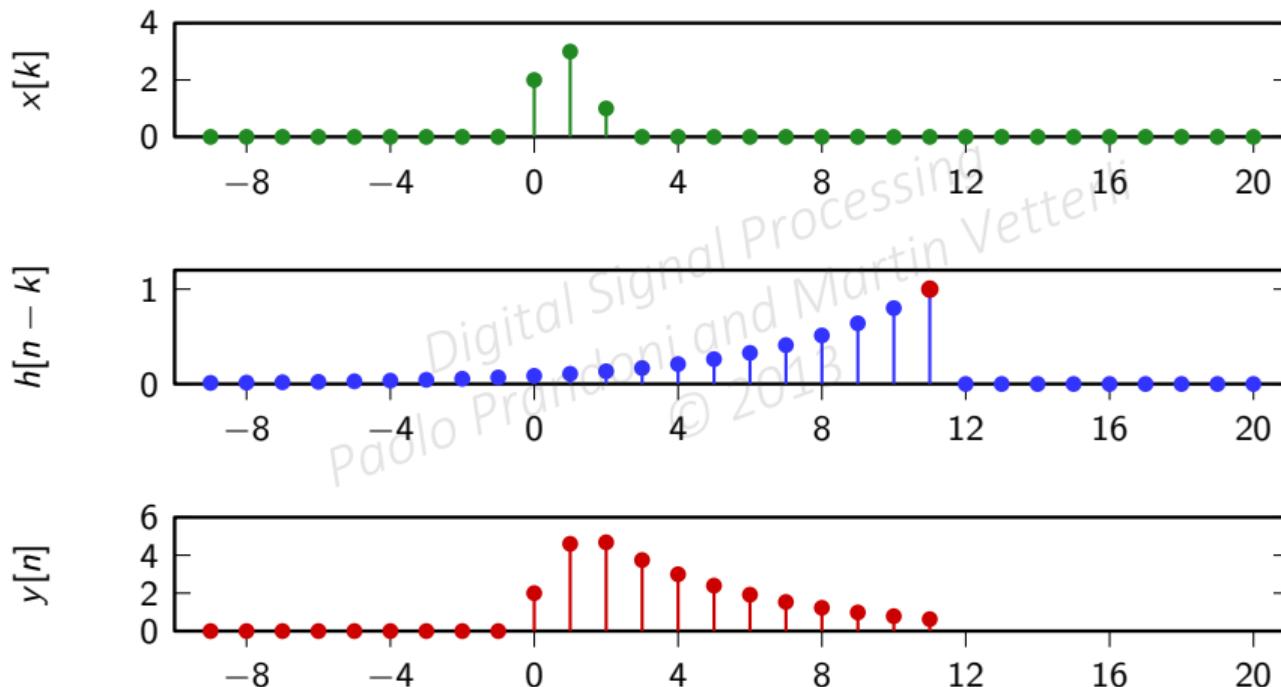
Convolution example



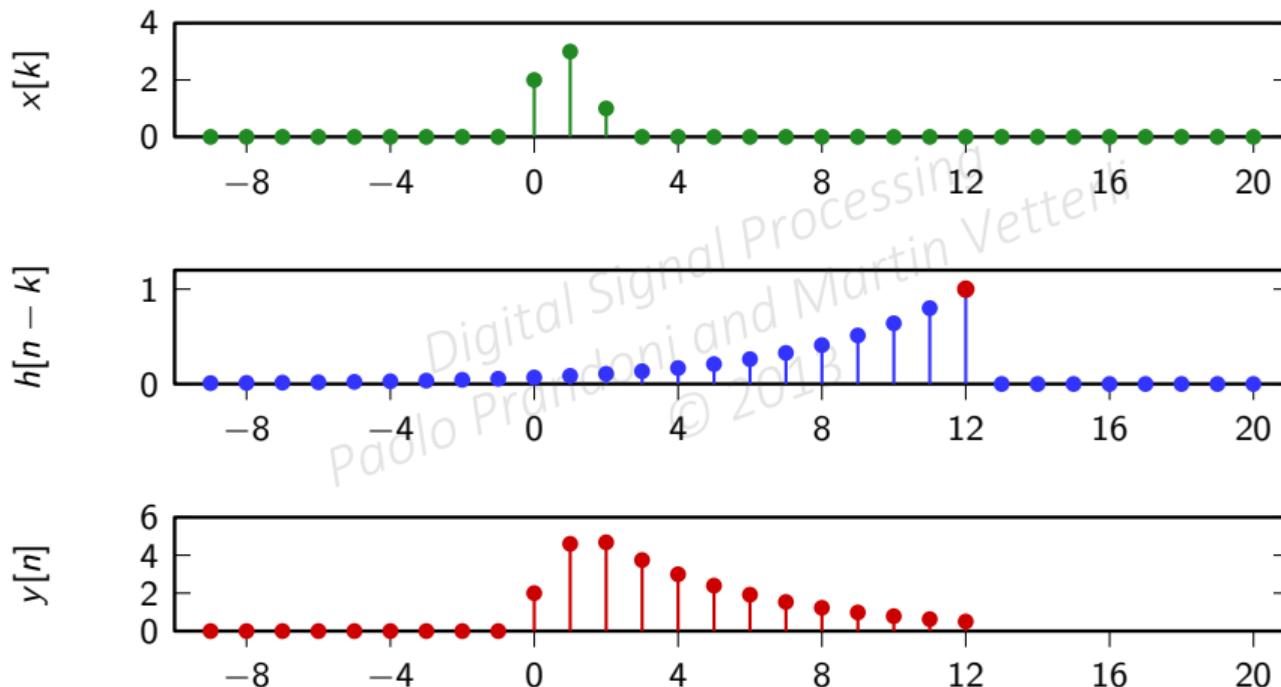
Convolution example



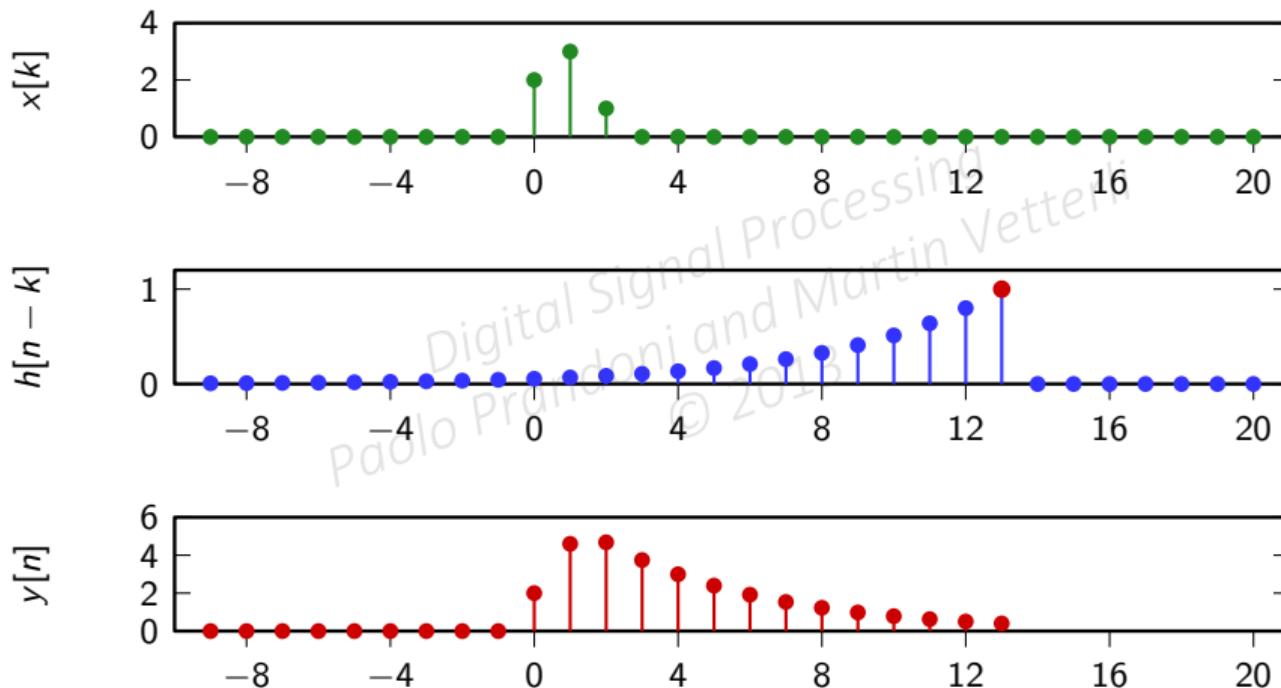
Convolution example



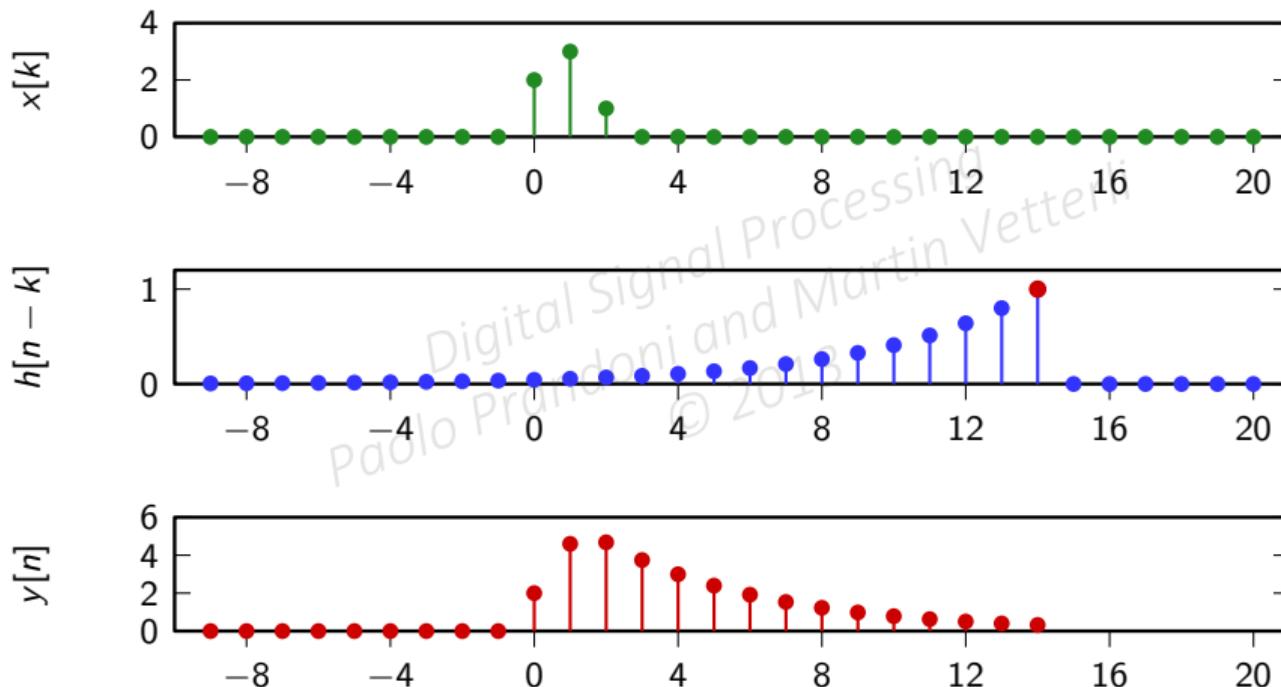
Convolution example



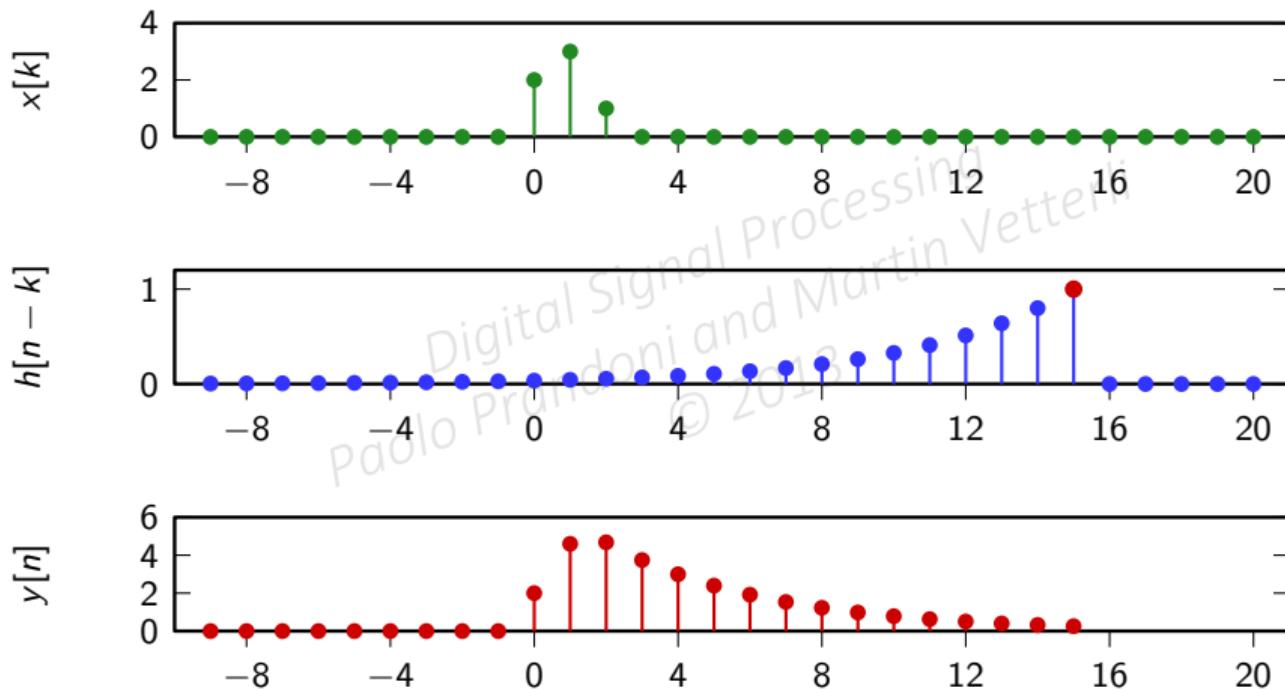
Convolution example



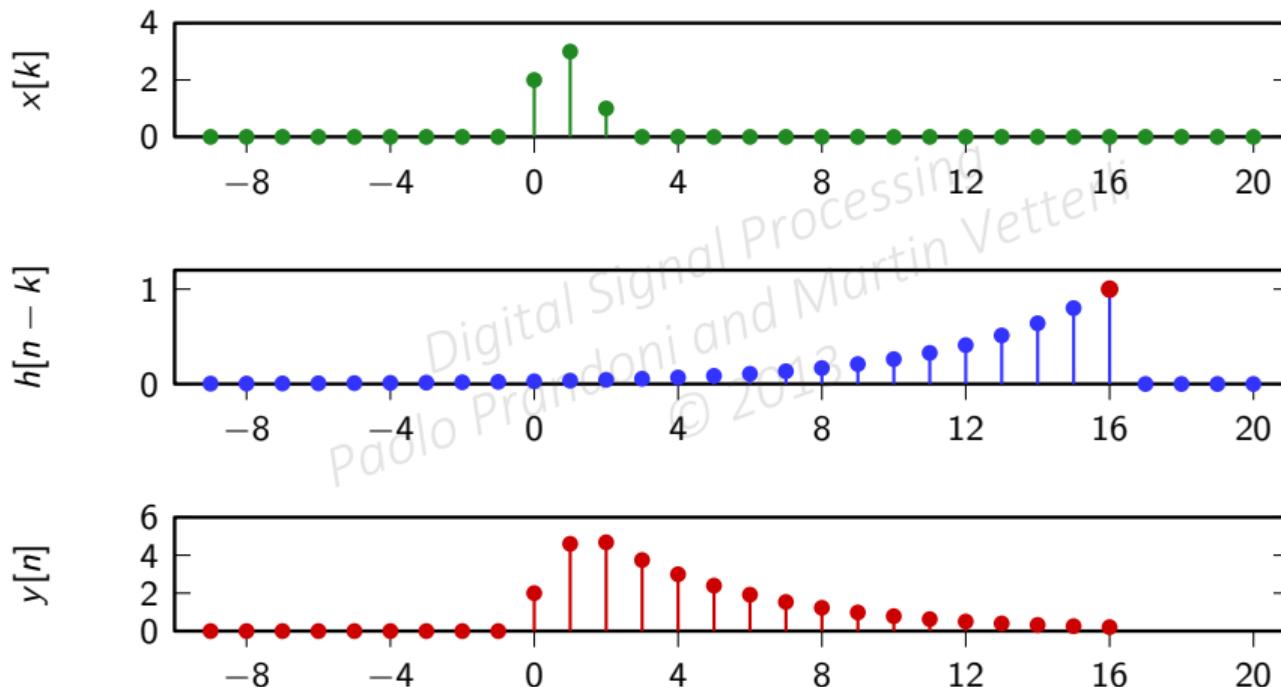
Convolution example



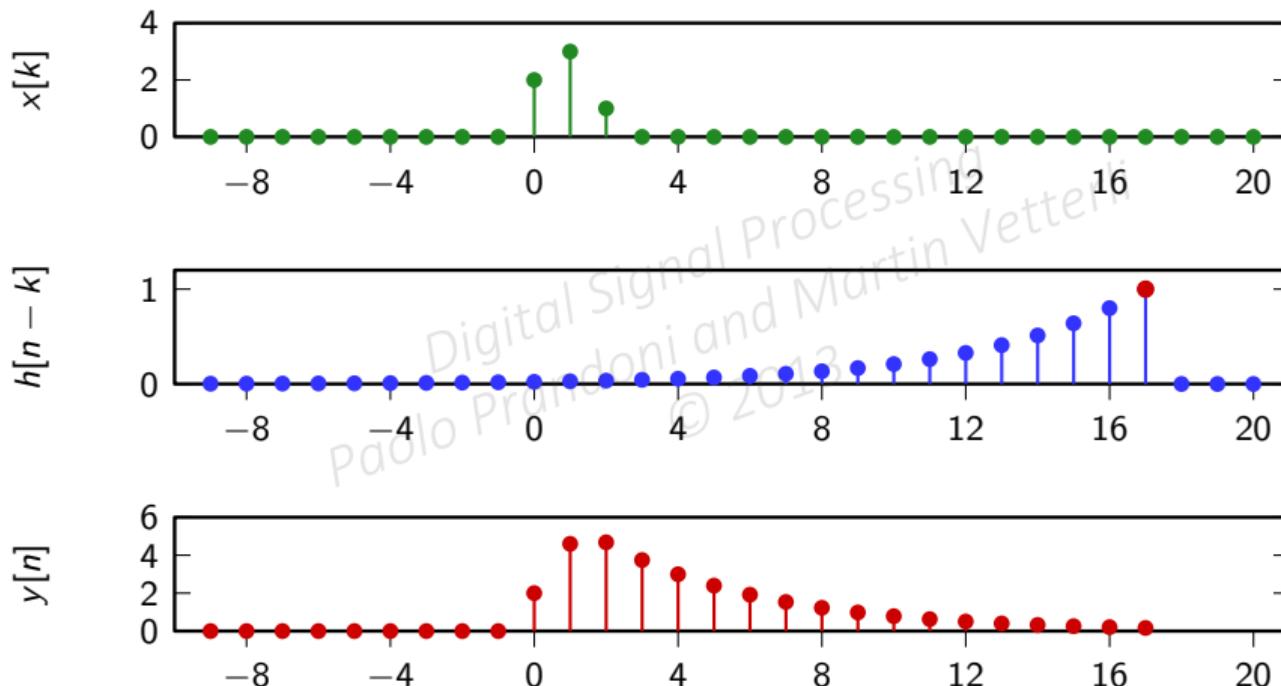
Convolution example



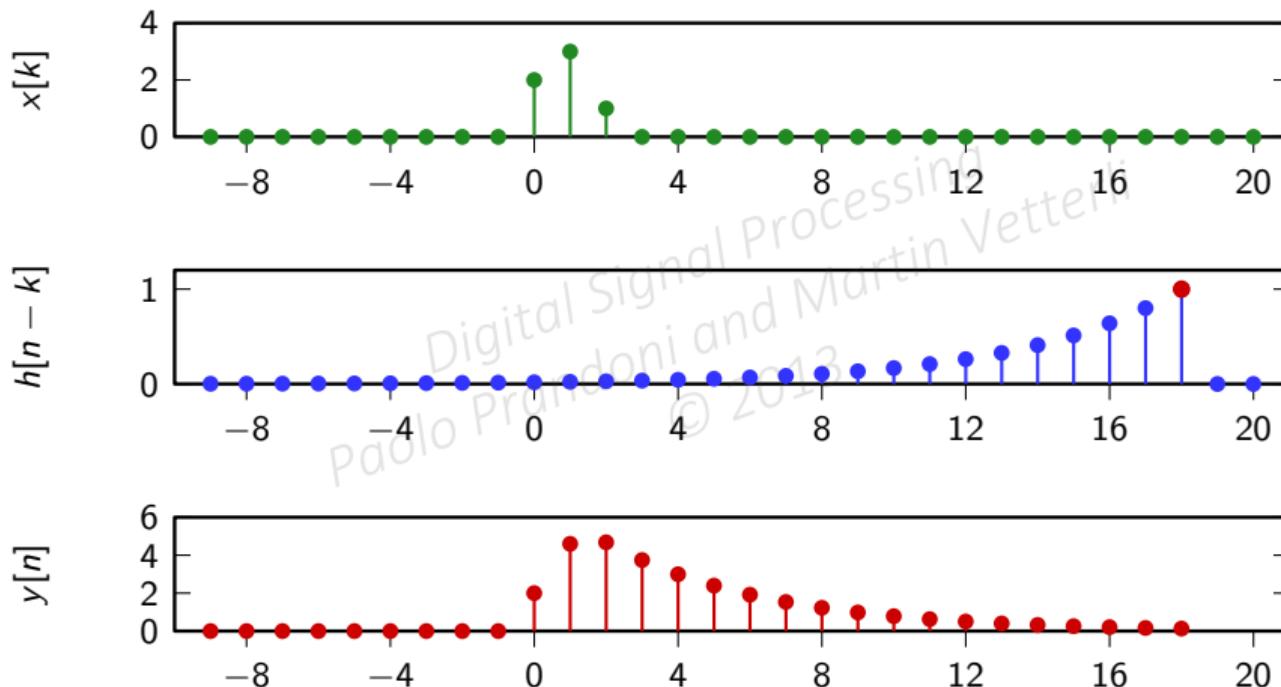
Convolution example



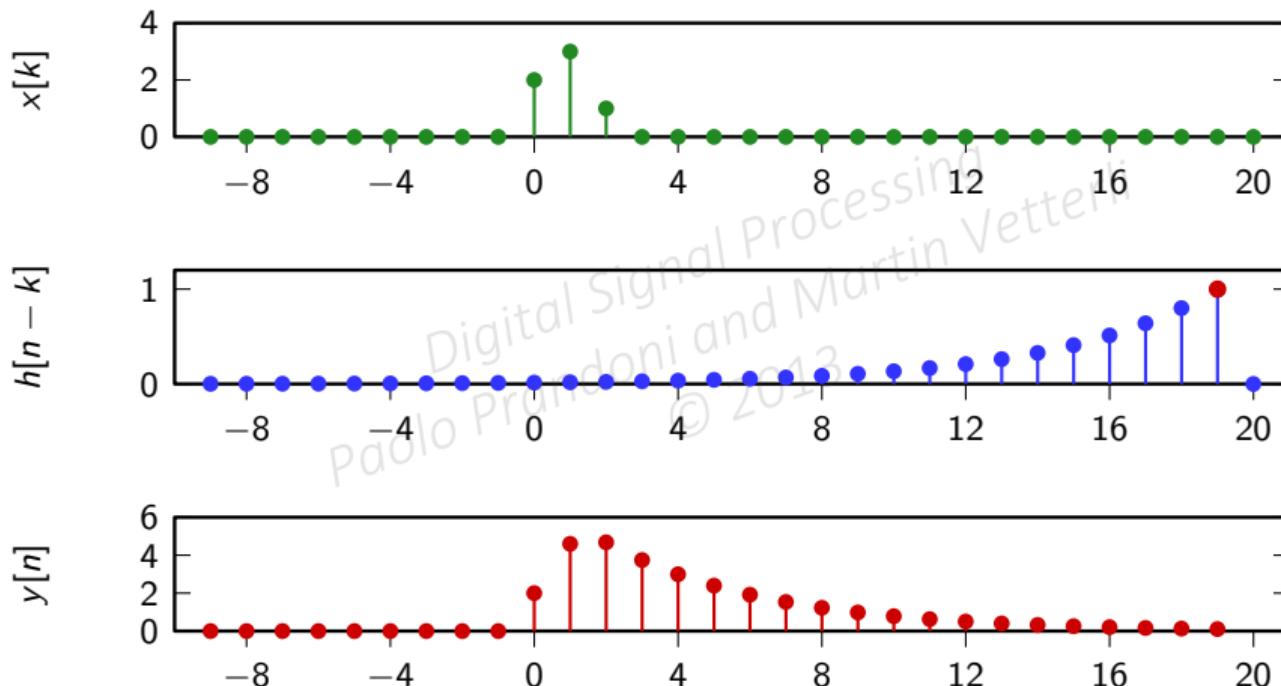
Convolution example



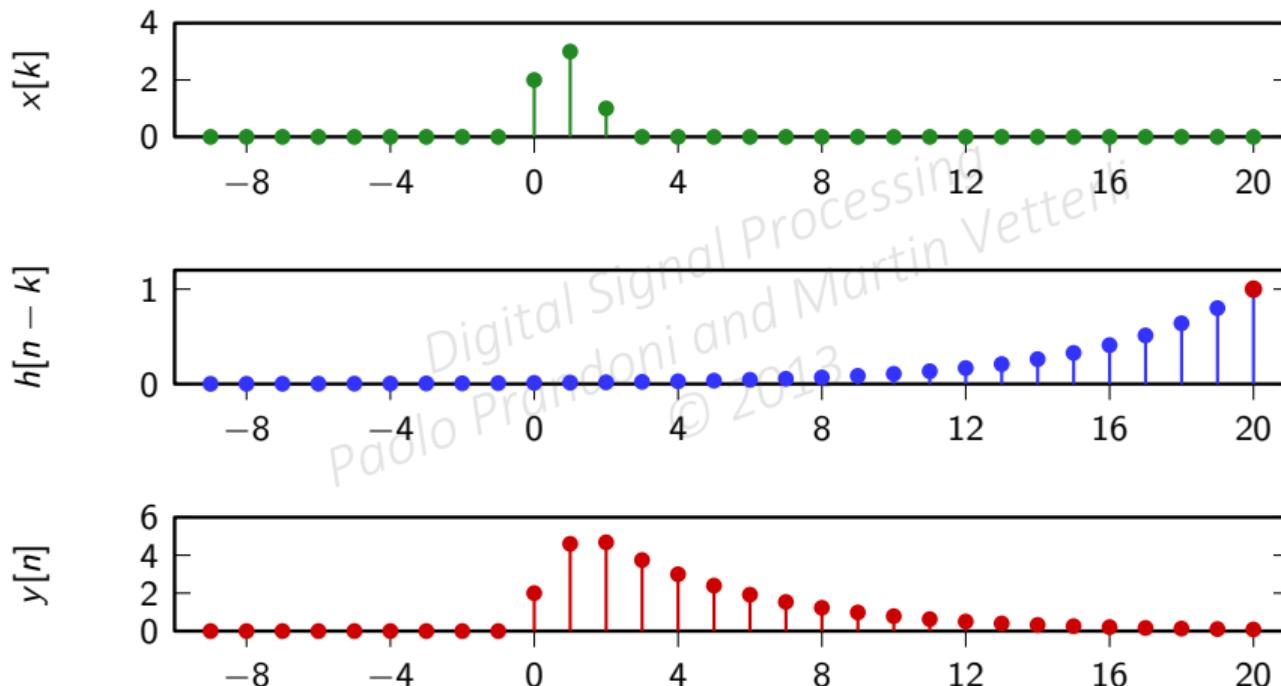
Convolution example



Convolution example



Convolution example



Convolution properties

- ▶ linearity and time invariance (by definition)
- ▶ commutativity: $(x * h)[n] = (h * x)[n]$
- ▶ associativity for absolutely- and square-summable sequences:
 $((x * h) * w)[n] = (x * (h * w))[n]$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

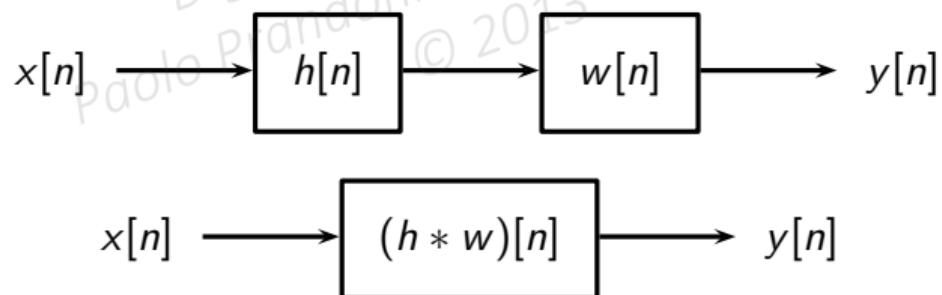
- ▶ linearity and time invariance (by definition)
- ▶ commutativity: $(x * h)[n] = (h * x)[n]$

▶ associativity for absolutely- and square-summable sequences:
 $((x * h) * w)[n] = (x * (h * w))[n]$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Convolution properties

- ▶ linearity and time invariance (by definition)
- ▶ commutativity: $(x * h)[n] = (h * x)[n]$
- ▶ associativity for absolutely- and square-summable sequences:
 $((x * h) * w)[n] = (x * (h * w))[n]$



END OF MODULE 5.1

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.2: Filtering by example

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Moving average filter
- ▶ Leaky integrator

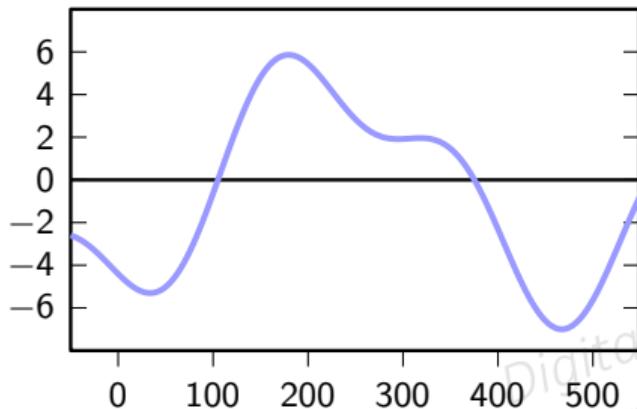
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Overview:

- ▶ Moving average filter
- ▶ Leaky integrator

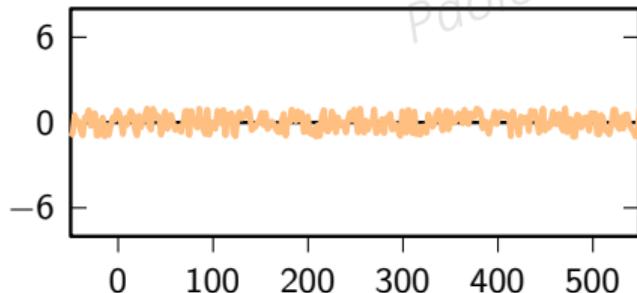
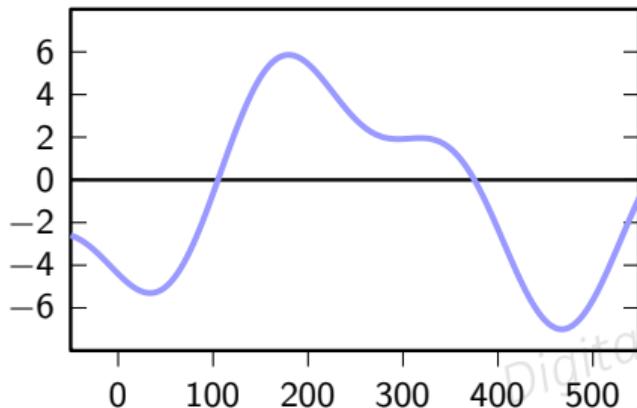
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Typical filtering scenario: denoising



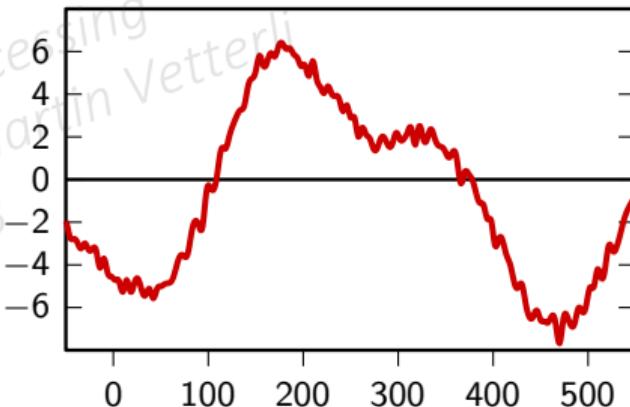
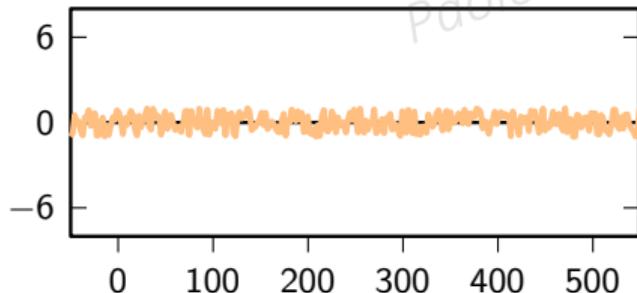
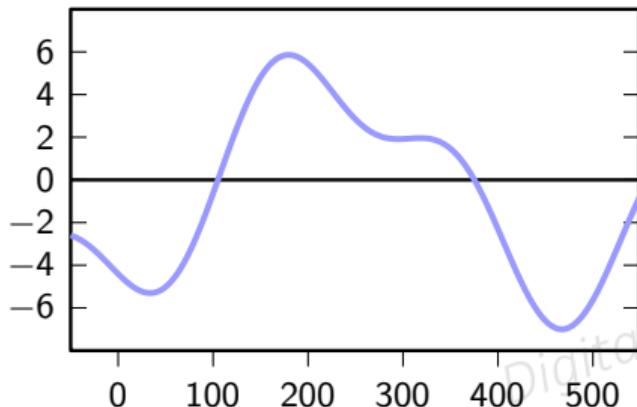
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Typical filtering scenario: denoising



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Typical filtering scenario: denoising



Digital Signal Processing
© 2013
Paolo Prandoni and Martin Vetterli

Denoising by Moving Average

- ▶ idea: replace each sample by the local average

- ▶ for instance: $y[n] = (x[n] + x[n - 1])/2$

- ▶ more generally:

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]$

Denoising by Moving Average

- ▶ idea: replace each sample by the local average
- ▶ for instance: $y[n] = (x[n] + x[n - 1])/2$
- ▶ more generally:

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]$

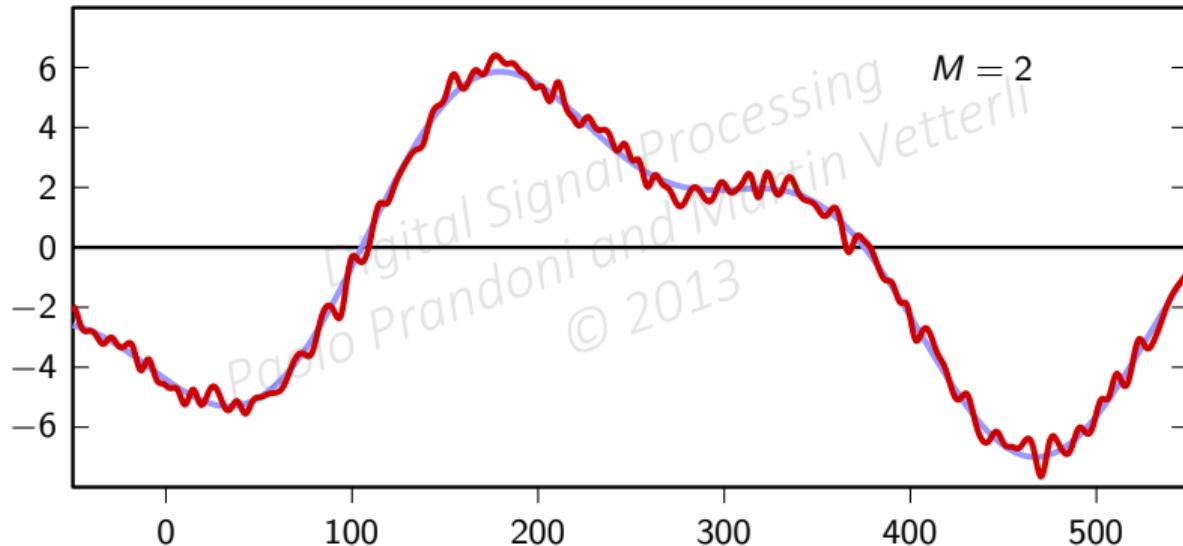
Denoising by Moving Average

- ▶ idea: replace each sample by the local average
- ▶ for instance: $y[n] = (x[n] + x[n - 1])/2$
- ▶ more generally:

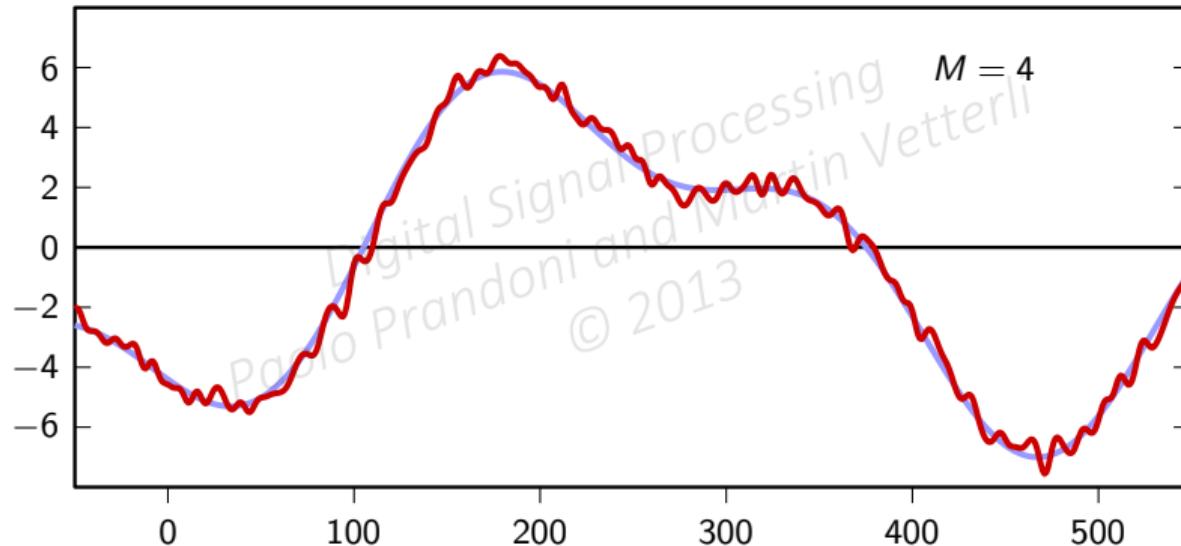
$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

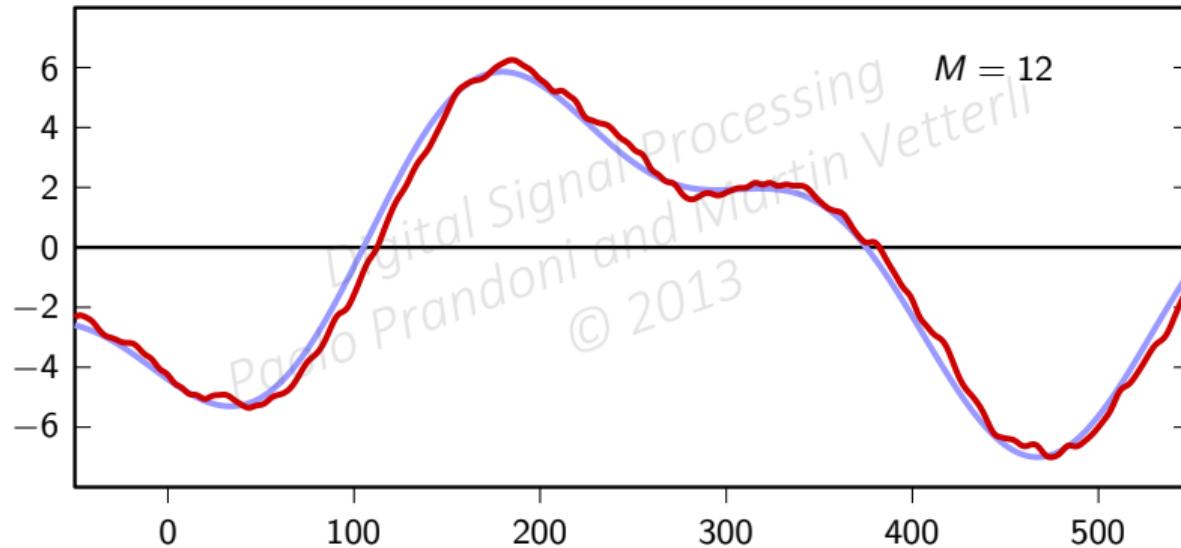
Denoising by Moving Average



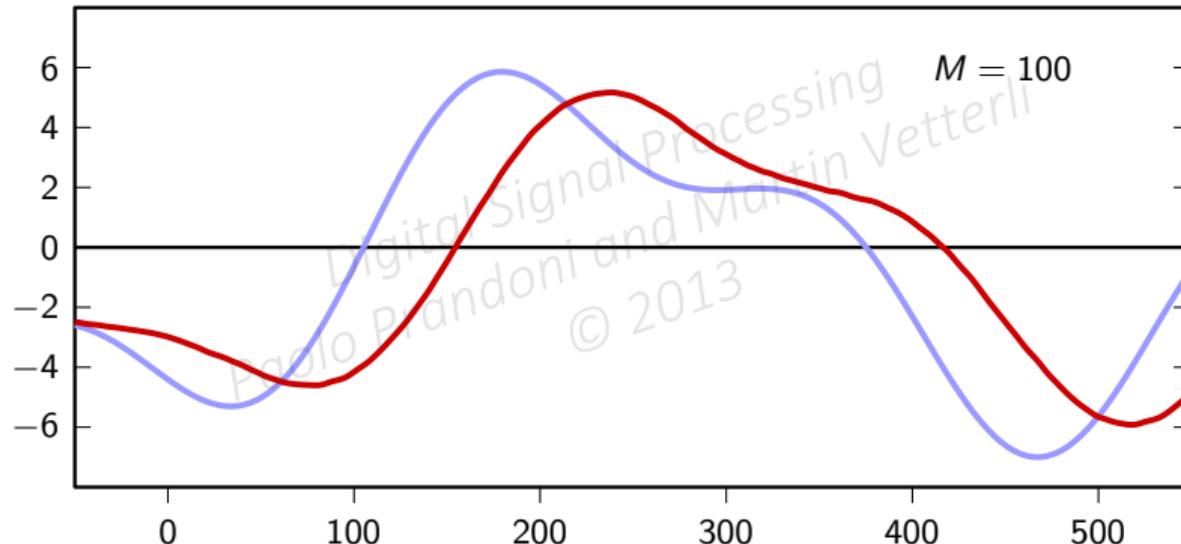
Denoising by Moving Average



Denoising by Moving Average



Denoising by Moving Average



$M = 100$

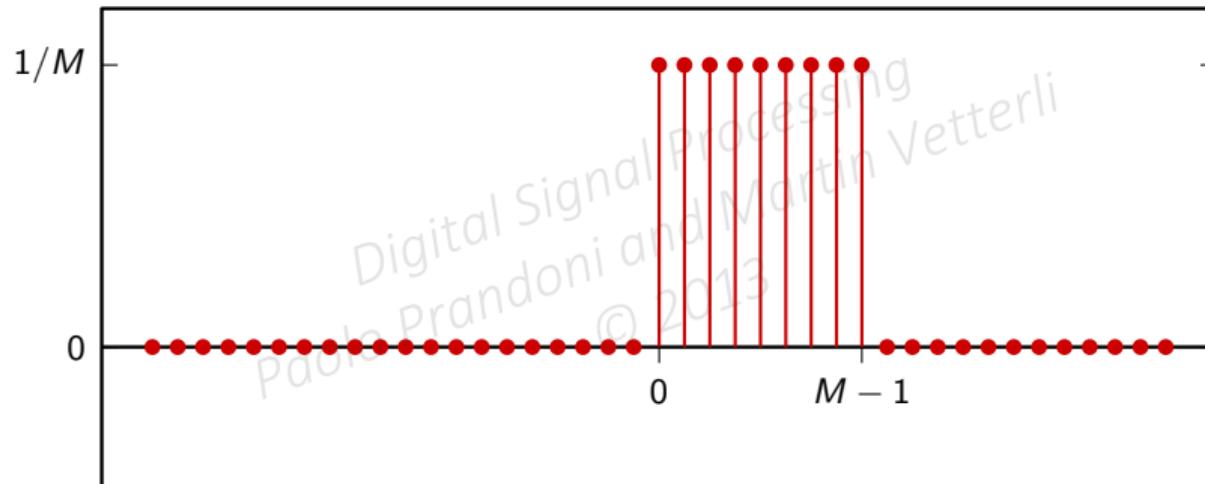
MA: impulse response

$$h[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n - k]$$
$$= \begin{cases} 1/M & \text{for } 0 \leq n < M \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} h[n] &= \frac{1}{M} \sum_{k=0}^{M-1} \delta[n - k] \\ &= \begin{cases} 1/M & \text{for } 0 \leq n < M \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

MA: impulse response



Digital Signal Processing
Paola Prandoni and Martin Vetterli
© 2013

- ▶ smoothing effect proportional to M
- ▶ number of operations and storage also proportional to M

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$y_M[n] = \frac{1}{M} (x[n] + x[n-1] + x[n-2] + \dots + x[n-M+1])$$

moving average over M points

$$y_M[n] = \frac{1}{M}x[n] + \frac{1}{M} (x[n-1] + x[n-2] + \dots + x[n-M+1])$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$y_M[n] = \frac{1}{M}x[n] + \frac{1}{M}(x[n-1] + x[n-2] + \dots + x[n-M+1])$$

“almost” $y_{M-1}[n-1]$

i.e., moving average over $M - 1$ points, delayed by one

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-1-k]$$

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=0}^{M-1} x[(n-1)-k]$$

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-(k+1)]$$

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=0+1}^{M-1+1} x[n-k]$$

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=1}^M x[n-k]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2003

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=1}^M x[n-k]$$

$$y_{M-1}[n] = \frac{1}{M-1} \sum_{k=0}^{M-2} x[n-k]$$

Formally:

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_M[n-1] = \frac{1}{M} \sum_{k=1}^M x[n-k]$$

$$y_{M-1}[n-1] = \frac{1}{M-1} \sum_{k=1}^{M-1} x[n-k]$$

From the MA to a first-order recursion

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$$y_{M-1}[n-1] = \frac{1}{M-1} \sum_{k=1}^{M-1} x[n-k]$$

Digital Signal Processing
Perlen und Martin Vetterli

$$\sum_{p=0}^{M-1} x[n-p] = x[n] + \sum_{k=1}^{M-1} x[n-k]$$

$$My_M[n] = x[n] + (M-1)y_{M-1}[n-1]$$

From the MA to a first-order recursion

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$
$$y_{M-1}[n-1] = \frac{1}{M-1} \sum_{k=1}^{M-1} x[n-k]$$

Digital Signal Processing
Riccardo Adorni and Martin Vetterli

$$\sum_{k=0}^{M-1} x[n-k] = x[n] + \sum_{k=1}^{M-1} x[n-k]$$

$$My_M[n] = x[n] + (M-1)y_{M-1}[n-1]$$

From the MA to a first-order recursion

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k] \quad y_{M-1}[n-1] = \frac{1}{M-1} \sum_{k=1}^{M-1} x[n-k]$$

$$\sum_{k=0}^{M-1} x[n-k] = x[n] + \sum_{k=1}^{M-1} x[n-k]$$

$$My_M[n] = x[n] + (M-1)y_{M-1}[n-1]$$

From the MA to a first-order recursion

$$y_M[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$
$$y_{M-1}[n-1] = \frac{1}{M-1} \sum_{k=1}^{M-1} x[n-k]$$

$$\sum_{k=0}^{M-1} x[n-k] = x[n] + \sum_{k=1}^{M-1} x[n-k]$$

$$My_M[n] = x[n] + (M-1)y_{M-1}[n-1]$$

From the MA to a first-order recursion

$$y_M[n] = \frac{M-1}{M}y_{M-1}[n-1] + \frac{1}{M}x[n]$$
$$y_M[n] = \lambda y_{M-1}[n-1] + (1-\lambda)x[n], \quad \lambda = \frac{M-1}{M}$$

$$y_M[n] = \frac{M-1}{M}y_{M-1}[n-1] + \frac{1}{M}x[n]$$
$$y_M[n] = \lambda y_{M-1}[n-1] + (1-\lambda)x[n], \quad \lambda = \frac{M-1}{M}$$

The Leaky Integrator

- ▶ when M is large, $y_{M-1}[n] \approx y_M[n]$ (and $\lambda \approx 1$)
- ▶ try the filter
 - ▶ filter is now recursive, since it uses its previous output value

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $y[n] = y[n-1] + M(1-\lambda)x[n]$
© 2013

- ▶ when M is large, $y_{M-1}[n] \approx y_M[n]$ (and $\lambda \approx 1$)
- ▶ try the filter

$$y[n] = \lambda y[n - 1] + (1 - \lambda)x[n]$$

- ▶ filter is now recursive, since it uses its previous output value

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ when M is large, $y_{M-1}[n] \approx y_M[n]$ (and $\lambda \approx 1$)
- ▶ try the filter

$$y[n] = \lambda y[n - 1] + (1 - \lambda)x[n]$$

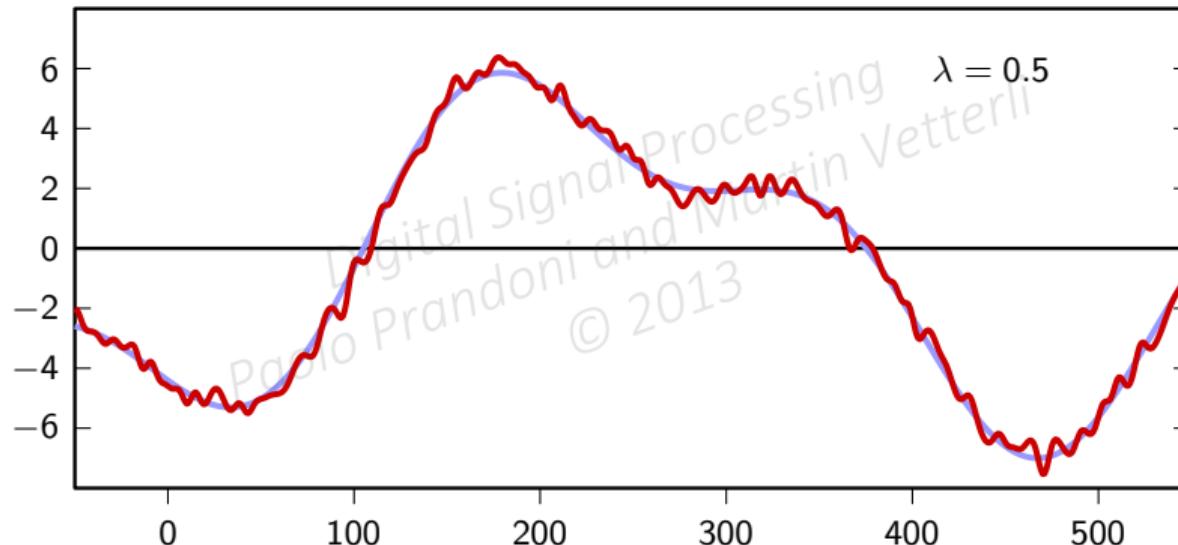
- ▶ filter is now recursive, since it uses its previous output value

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

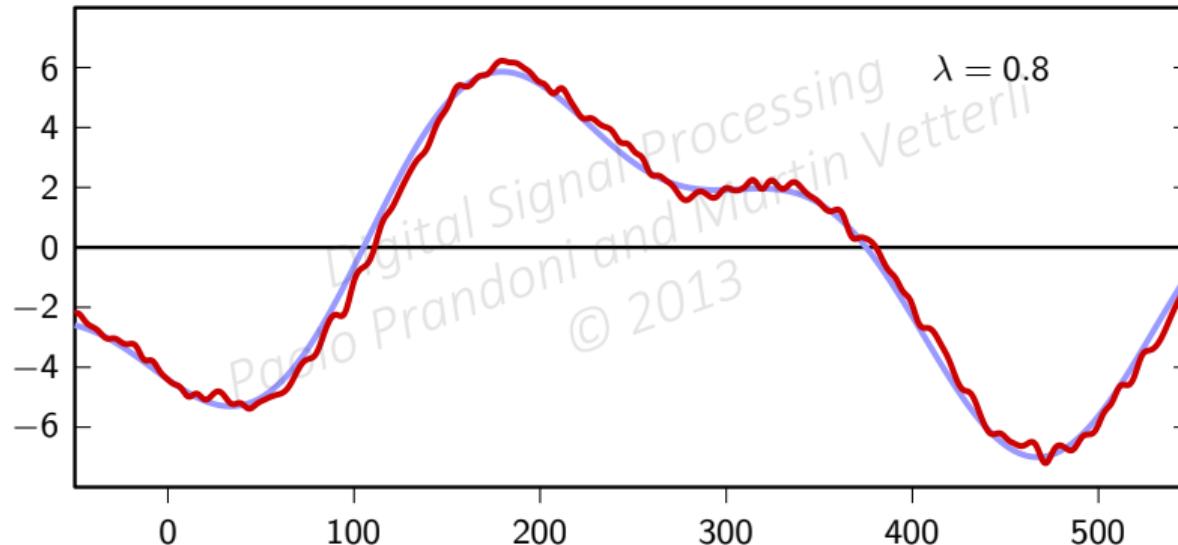
Denoising recursively with the Leaky Integrator



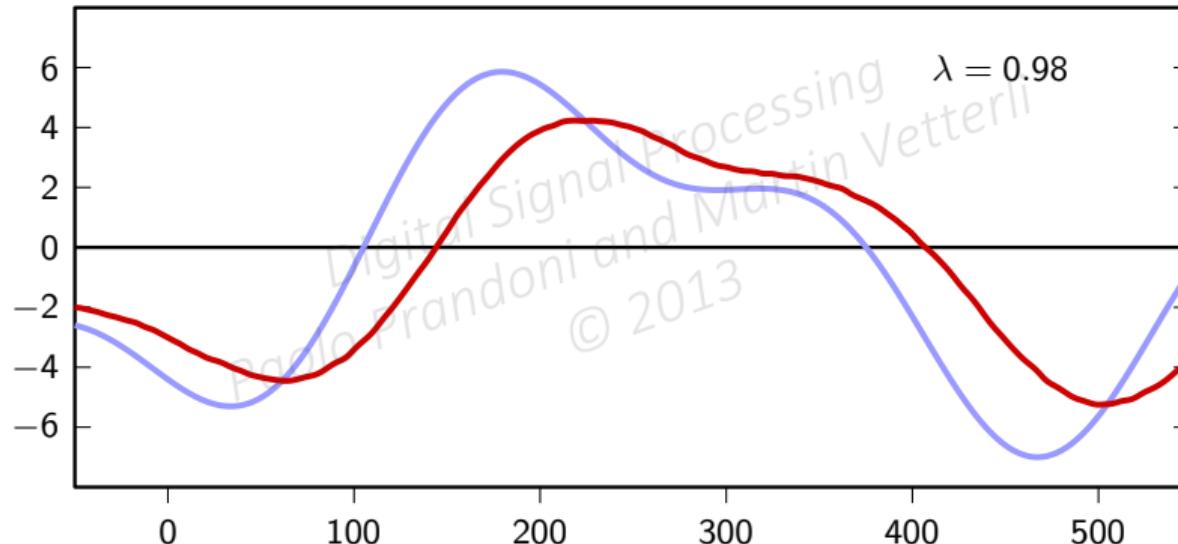
Denoising recursively with the Leaky Integrator



Denoising recursively with the Leaky Integrator



Denoising recursively with the Leaky Integrator



What about the impulse response?

$$y[n] = \lambda y[n - 1] + (1 - \lambda)\delta[n]$$

- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1 - \lambda)\delta[0] = (1 - \lambda)$
- ▶ $y[1] = \lambda y[0] + (1 - \lambda)\delta[1] = \lambda(1 - \lambda)$
- ▶ $y[2] = \lambda y[1] + (1 - \lambda)\delta[2] = \lambda^2(1 - \lambda)$
- ▶ $y[3] = \lambda y[2] + (1 - \lambda)\delta[3] = \lambda^3(1 - \lambda)$
- ▶ ...

What about the impulse response?

$$y[n] = \lambda y[n - 1] + (1 - \lambda)\delta[n]$$

- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1 - \lambda)\delta[0] = (1 - \lambda)$
- ▶ $y[1] = \lambda y[0] + (1 - \lambda)\delta[1] = \lambda(1 - \lambda)$
- ▶ $y[2] = \lambda y[1] + (1 - \lambda)\delta[2] = \lambda^2(1 - \lambda)$
- ▶ $y[3] = \lambda y[2] + (1 - \lambda)\delta[3] = \lambda^3(1 - \lambda)$
- ▶ ...

What about the impulse response?

$$y[n] = \lambda y[n-1] + (1-\lambda)\delta[n]$$

- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1-\lambda)\delta[0] = (1-\lambda)$
- ▶ $y[1] = \lambda y[0] + (1-\lambda)\delta[1] = \lambda(1-\lambda)$
- ▶ $y[2] = \lambda y[1] + (1-\lambda)\delta[2] = \lambda^2(1-\lambda)$
- ▶ $y[3] = \lambda y[2] + (1-\lambda)\delta[3] = \lambda^3(1-\lambda)$
- ▶ ...

What about the impulse response?

$$y[n] = \lambda y[n-1] + (1 - \lambda) \delta[n]$$

- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1 - \lambda) \delta[0] = (1 - \lambda)$
- ▶ $y[1] = \lambda y[0] + (1 - \lambda) \delta[1] = \lambda(1 - \lambda)$
- ▶ $y[2] = \lambda y[1] + (1 - \lambda) \delta[2] = \lambda^2(1 - \lambda)$
- ▶ $y[3] = \lambda y[2] + (1 - \lambda) \delta[3] = \lambda^3(1 - \lambda)$
- ▶ ...

What about the impulse response?

$$y[n] = \lambda y[n-1] + (1 - \lambda) \delta[n]$$

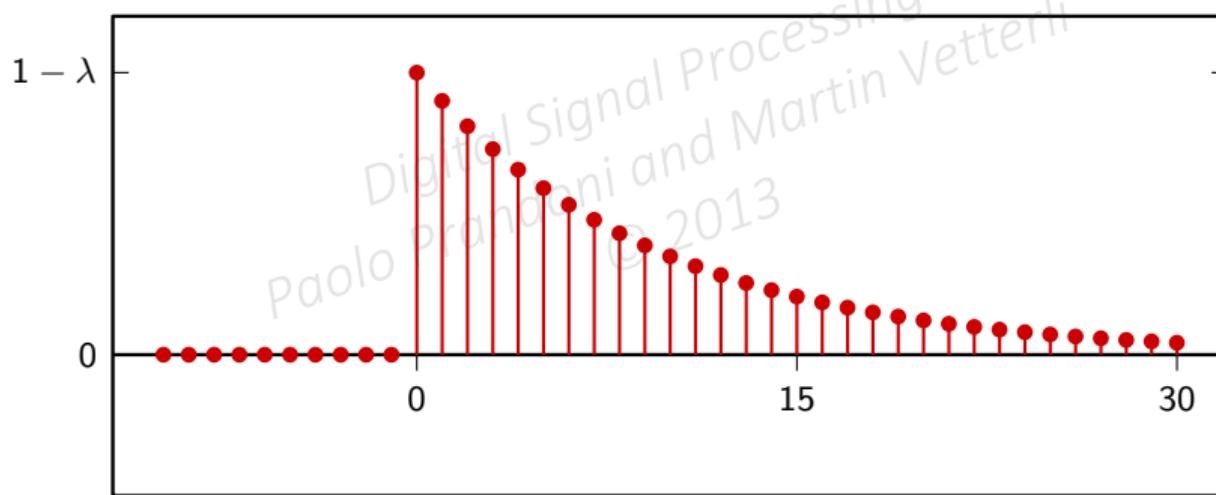
- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1 - \lambda) \delta[0] = (1 - \lambda)$
- ▶ $y[1] = \lambda y[0] + (1 - \lambda) \delta[1] = \lambda(1 - \lambda)$
- ▶ $y[2] = \lambda y[1] + (1 - \lambda) \delta[2] = \lambda^2(1 - \lambda)$
- ▶ $y[3] = \lambda y[2] + (1 - \lambda) \delta[3] = \lambda^3(1 - \lambda)$
- ▶ ...

What about the impulse response?

$$y[n] = \lambda y[n-1] + (1-\lambda)\delta[n]$$

- ▶ $y[n] = 0$ for all $n < 0$
- ▶ $y[0] = \lambda y[-1] + (1-\lambda)\delta[0] = (1-\lambda)$
- ▶ $y[1] = \lambda y[0] + (1-\lambda)\delta[1] = \lambda(1-\lambda)$
- ▶ $y[2] = \lambda y[1] + (1-\lambda)\delta[2] = \lambda^2(1-\lambda)$
- ▶ $y[3] = \lambda y[2] + (1-\lambda)\delta[3] = \lambda^3(1-\lambda)$
- ▶ ...

$$h[n] = (1 - \lambda)\lambda^n u[n]$$



Discrete-time integrator is a boundless accumulator:

$$y[n] = \sum_{k=-\infty}^n x[k]$$

We can rewrite the integrator as

$$y[n] = y[n - 1] + x[n]$$

To prevent “explosion” pick $\lambda < 1$

$$y[n] = \lambda y[n - 1] + (1 - \lambda)x[n]$$

keep only a fraction λ of
the accumulated value
so far and forget
(“leak”) a fraction $1 - \lambda$

add only a fraction $1 - \lambda$
of the current value to
the accumulator

END OF MODULE 5.2

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.3: Filter Stability

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter classification in the time domain
- ▶ Filter stability

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter classification in the time domain
- ▶ Filter stability

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to impulse response

- ▶ Finite Impulse Response (FIR)
- ▶ Infinite Impulse Response (IIR)
- ▶ causal
- ▶ noncausal

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to impulse response

- ▶ Finite Impulse Response (FIR)
- ▶ Infinite Impulse Response (IIR)
- ▶ causal
- ▶ noncausal

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to impulse response

- ▶ Finite Impulse Response (FIR)
- ▶ Infinite Impulse Response (IIR)
- ▶ causal
- ▶ noncausal

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to impulse response

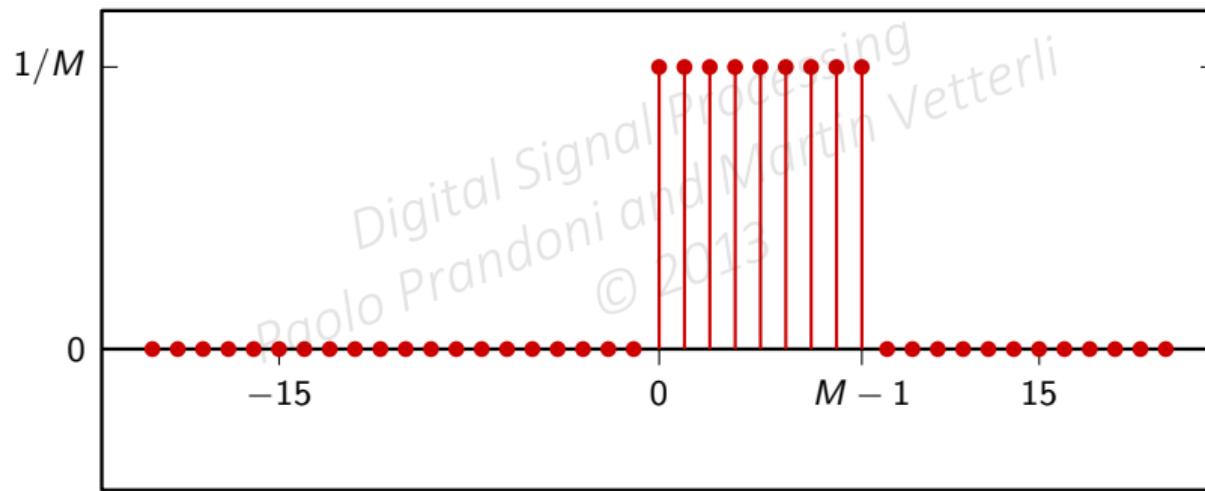
- ▶ Finite Impulse Response (FIR)
- ▶ Infinite Impulse Response (IIR)
- ▶ causal
- ▶ noncausal

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ impulse response has finite support
- ▶ only a finite number of samples are involved in the computation of each output sample

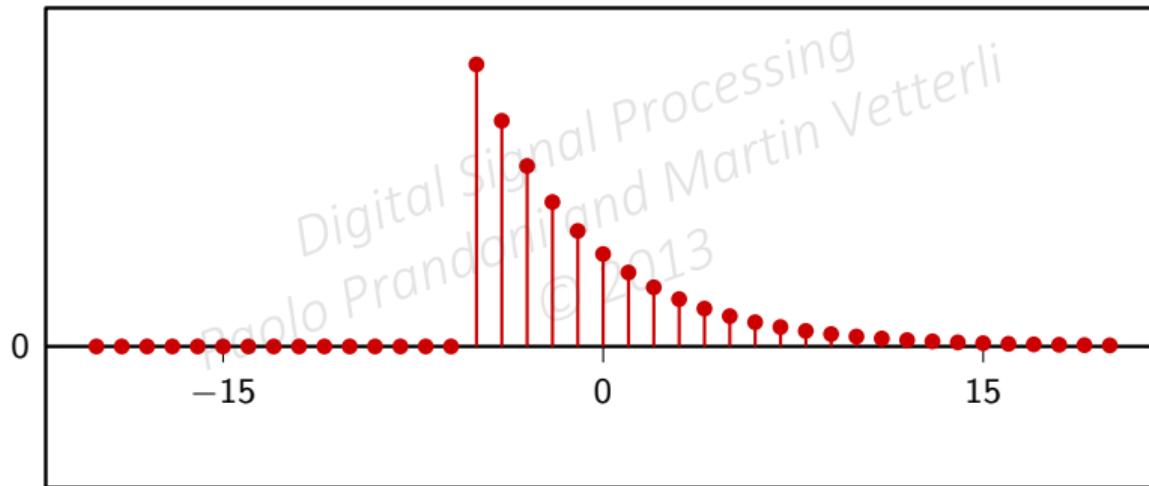
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average filter



- ▶ impulse response has infinite support
- ▶ a potentially infinite number of samples are involved in the computation of each output sample
- ▶ surprisingly, in many cases the computation can still be performed in a finite amount of steps

Leaky Integrator



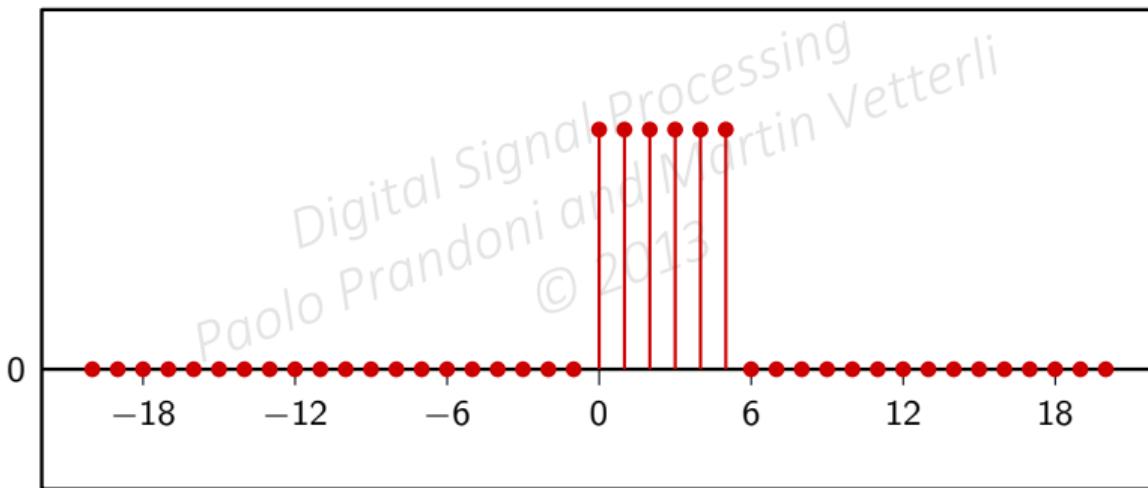
► causal:

- impulse response is zero for $n < 0$
- only past samples (with respect to the present) are involved in the computation of each output sample
- causal filters can work “on line” since they only need the past

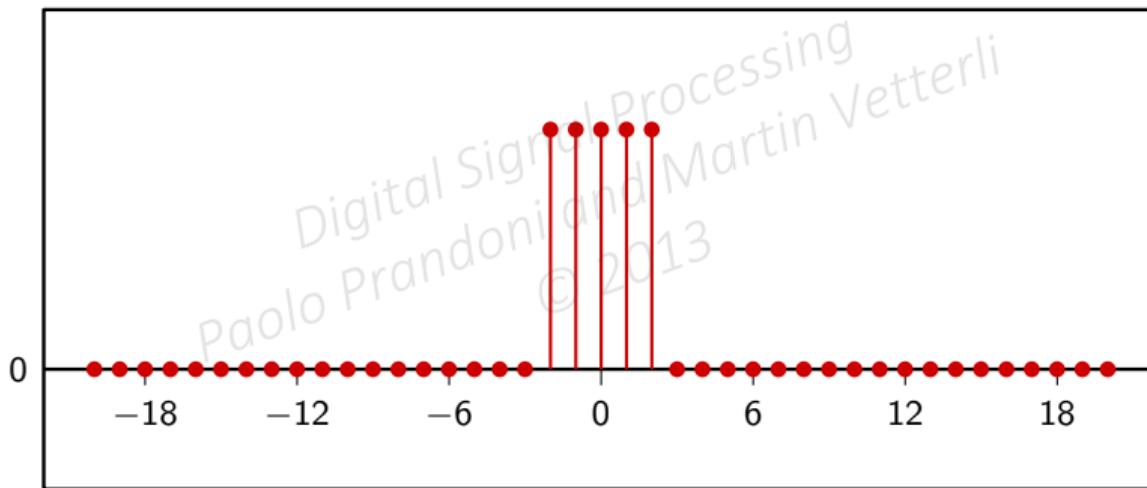
► noncausal:

- impulse response is nonzero for some (or all) $n < 0$
- can still be implemented in an offline fashion (when all input data is available on storage, e.g. in Image Processing)

Moving Average filter



Zero-centered Moving Average filter



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ key concept: avoid “explosions” if the input is nice
 - ▶ a nice signal is a bounded signal: $|x[n]| < M$ for all n
 - ▶ Bounded-Input Bounded-Output (BIBO) stability: if the input is nice the output should be nice

- ▶ key concept: avoid “explosions” if the input is nice
- ▶ a nice signal is a bounded signal: $|x[n]| < M$ for all n
- ▶ Bounded-Input Bounded-Output (BIBO) stability: if the input is nice the output should be nice

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ key concept: avoid “explosions” if the input is nice
- ▶ a nice signal is a bounded signal: $|x[n]| < M$ for all n
- ▶ Bounded-Input Bounded-Output (BIBO) stability: if the input is nice the output should be nice

A filter is BIBO stable if and only if its impulse response is absolutely summable

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Proof:

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $\sum_n |h[n]| = L < \infty$

Thesis:

- ▶ $|y[n]|$ bounded

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$\begin{aligned} |y[n]| &= \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right| \\ &\leq \sum_{k=-\infty}^{\infty} |h[k]x[n-k]| \\ &\leq M \sum_{k=-\infty}^{\infty} |h[k]| \\ &\leq ML \end{aligned}$$

Proof:

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $\sum_n |h[n]| = L < \infty$

Thesis:

- ▶ $|y[n]|$ bounded

$$|y[n]| = \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right|$$

$$\leq \sum_{k=-\infty}^{\infty} |h[k]x[n-k]|$$

$$\leq M \sum_{k=-\infty}^{\infty} |h[k]|$$

$$\leq ML$$

Proof:

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $\sum_n |h[n]| = L < \infty$

Thesis:

- ▶ $|y[n]|$ bounded

$$|y[n]| = \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right|$$

$$\leq \sum_{k=-\infty}^{\infty} |h[k]x[n-k]|$$

$$\leq M \sum_{k=-\infty}^{\infty} |h[k]|$$

$$\leq ML$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Proof:

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $\sum_n |h[n]| = L < \infty$

Thesis:

- ▶ $|y[n]|$ bounded

$$|y[n]| = \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right|$$

$$\leq \sum_{k=-\infty}^{\infty} |h[k]x[n-k]|$$

$$\leq M \sum_{k=-\infty}^{\infty} |h[k]|$$

$$\leq ML$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Proof:

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $\sum_n |h[n]| = L < \infty$

Thesis:

- ▶ $|y[n]|$ bounded

$$|y[n]| = \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right|$$

$$\leq \sum_{k=-\infty}^{\infty} |h[k]x[n-k]|$$

$$\leq M \sum_{k=-\infty}^{\infty} |h[k]|$$

$$\leq ML$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Proof (by contradiction):

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $|y[n]| < P$

Thesis:

- ▶ $h[n]$ absolutely summable

- ▶ assume $\sum_n |h[n]| = \infty$

- ▶ build $x[n] \begin{cases} \text{if } h[-n] \neq 0 \\ 0 \end{cases}$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ clearly, $x[n]$ is bounded
- ▶ however

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

Proof (by contradiction):

- ▶ assume $\sum_n |h[n]| = \infty$

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $|y[n]| < P$

Thesis:

- ▶ $h[n]$ absolutely summable

- ▶ build $x[n] \begin{cases} \text{if } h[-n] \neq 0 \\ 0 \end{cases}$
- ▶ clearly, $x[n]$ is bounded
- ▶ however

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $|y[n]| < P$

Thesis:

- ▶ $h[n]$ absolutely summable

Proof (by contradiction):

- ▶ assume $\sum_n |h[n]| = \infty$
- ▶ build $x[n] = \begin{cases} +1 & \text{if } h[-n] \geq 0 \\ -1 & \text{if } h[-n] < 0 \end{cases}$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

Proof (by contradiction):

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $|y[n]| < P$

Thesis:

- ▶ $h[n]$ absolutely summable

▶ assume $\sum_n |h[n]| = \infty$

▶ build $x[n] = \begin{cases} +1 & \text{if } h[-n] \geq 0 \\ -1 & \text{if } h[-n] < 0 \end{cases}$

▶ clearly, $x[n]$ is bounded

▶ however

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

Hypotheses:

- ▶ $|x[n]| < M$
- ▶ $|y[n]| < P$

Thesis:

- ▶ $h[n]$ absolutely summable

Proof (by contradiction):

- ▶ assume $\sum_n |h[n]| = \infty$
- ▶ build $x[n] = \begin{cases} +1 & \text{if } h[-n] \geq 0 \\ -1 & \text{if } h[-n] < 0 \end{cases}$
- ▶ clearly, $x[n]$ is bounded
- ▶ however

$$y[0] = (x * h)[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} |h[k]| = \infty$$

FIR filters are always stable

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Let's check the Leaky Integrator:

$$\sum_{n=-\infty}^{\infty} |h[n]| = |1 - \lambda| \sum_{n=0}^{\infty} |\lambda|^n$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $\lim_{n \rightarrow \infty} |1 - \lambda| \frac{1 - |\lambda|^{n+1}}{1 - |\lambda|}$
© 2013

$$< \infty \text{ for } |\lambda| < 1$$

stability is guaranteed for $|\lambda| < 1$

Let's check the Leaky Integrator:

$$\sum_{n=-\infty}^{\infty} |h[n]| = |1 - \lambda| \sum_{n=0}^{\infty} |\lambda|^n$$
$$= \lim_{n \rightarrow \infty} |1 - \lambda| \frac{1 - |\lambda|^{n+1}}{1 - |\lambda|}$$
$$< \infty \quad \text{for } |\lambda| < 1$$

stability is guaranteed for $|\lambda| < 1$

Let's check the Leaky Integrator:

$$\sum_{n=-\infty}^{\infty} |h[n]| = |1 - \lambda| \sum_{n=0}^{\infty} |\lambda|^n$$
$$= \lim_{n \rightarrow \infty} |1 - \lambda| \frac{1 - |\lambda|^{n+1}}{1 - |\lambda|}$$
$$< \infty \quad \text{for } |\lambda| < 1$$

stability is guaranteed for $|\lambda| < 1$

Let's check the Leaky Integrator:

$$\sum_{n=-\infty}^{\infty} |h[n]| = |1 - \lambda| \sum_{n=0}^{\infty} |\lambda|^n$$
$$= \lim_{n \rightarrow \infty} |1 - \lambda| \frac{1 - |\lambda|^{n+1}}{1 - |\lambda|}$$
$$< \infty \quad \text{for } |\lambda| < 1$$

stability is guaranteed for $|\lambda| < 1$

We will study indirect methods for filter stability later in this Module.

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

END OF MODULE 5.3

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.4: Frequency Response

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Overview:

- ▶ Eigensequences
- ▶ Convolution theorem
- ▶ Frequency and phase response

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Eigensequences
- ▶ Convolution theorem
- ▶ Frequency and phase response

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Eigensequences
- ▶ Convolution theorem
- ▶ Frequency and phase response

Digital Signal Processing
Paolo Frandoni and Martin Vetterli
© 2013

A remarkable result



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

A remarkable result

$$y[n] = e^{j\omega_0 n} * h[n]$$

$$= h[n] * e^{j\omega_0 n}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$= e^{j\omega_0 n} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k}$$

$$= H(e^{j\omega_0}) e^{j\omega_0 n}$$

A remarkable result

$$y[n] = e^{j\omega_0 n} * h[n]$$

$$= h[n] * e^{j\omega_0 n}$$

$$= e^{j\omega_0 n} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k}$$

$$= H(e^{j\omega_0}) e^{j\omega_0 n}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

A remarkable result

$$\begin{aligned}y[n] &= e^{j\omega_0 n} * h[n] \\&= h[n] * e^{j\omega_0 n} \\&= \sum_{k=-\infty}^{\infty} h[k] e^{j\omega_0(n-k)} \\&= e^{j\omega_0 n} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k} \\&= H(e^{j\omega_0}) e^{j\omega_0 n}\end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

A remarkable result

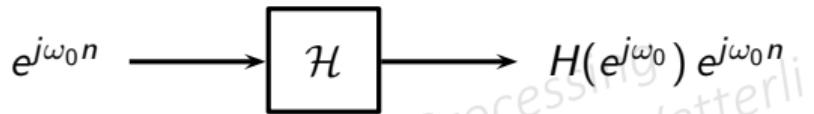
$$\begin{aligned}y[n] &= e^{j\omega_0 n} * h[n] \\&= h[n] * e^{j\omega_0 n} \\&= \sum_{k=-\infty}^{\infty} h[k] e^{j\omega_0(n-k)} \\&= e^{j\omega_0 n} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k} \\&= H(e^{j\omega_0}) e^{j\omega_0 n}\end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

A remarkable result

$$\begin{aligned}y[n] &= e^{j\omega_0 n} * h[n] \\&= h[n] * e^{j\omega_0 n} \\&= \sum_{k=-\infty}^{\infty} h[k] e^{j\omega_0(n-k)} \\&= e^{j\omega_0 n} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k} \\&= H(e^{j\omega_0}) e^{j\omega_0 n}\end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



- ▶ complex exponentials are *eigensequences* of LTI systems, i.e., linear filters cannot change the frequency of sinusoids
- ▶ DTFT of impulse response determines the frequency characteristic of a filter



- ▶ complex exponentials are *eigensequences* of LTI systems, i.e., linear filters cannot change the frequency of sinusoids
- ▶ DTFT of impulse response determines the frequency characteristic of a filter

If $H(e^{j\omega_0}) = Ae^{j\theta}$, then

$$\mathcal{H}\{e^{j\omega_0 n}\} = Ae^{j(\omega_0 n + \theta)}$$

amplitude:

amplification ($A > 1$)

or attenuation ($0 \leq A < 1$)

phase shift:

delay ($\theta < 0$)

or advancement ($\theta > 0$)



In general:

$$\text{DTFT}\{x[n] * h[n]\} = ?$$

Intuition: the DTFT reconstruction formula tells us how to build $x[n]$ from a set of complex exponential “basis” functions. By linearity...

In general:

$$\text{DTFT}\{x[n] * h[n]\} = ?$$

Intuition: the DTFT reconstruction formula tells us how to build $x[n]$ from a set of complex exponential “basis” functions. By linearity...

The convolution theorem

$$\begin{aligned} \text{DTFT } \{x[n] * h[n]\} &= \sum_{n=-\infty}^{\infty} (x * h)[n] e^{-j\omega n} \\ &= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n} \\ &\quad \text{Digital Signal Processing} \\ &\quad \text{Paolo Prandoni and Martin Vetterli} \\ &= \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)} \\ &= H(e^{j\omega}) X(e^{j\omega}) \end{aligned}$$

The convolution theorem

$$\begin{aligned}\text{DTFT } \{x[n] * h[n]\} &= \sum_{n=-\infty}^{\infty} (x * h)[n] e^{-j\omega n} \\&= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n} \\&\quad \text{Digital Signal Processing} \\&\quad \text{Paolo Prendini and Martin Vetterli} \\&\quad \text{#sum @2013} \\&= \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)} \\&= H(e^{j\omega}) X(e^{j\omega})\end{aligned}$$

The convolution theorem

$$\begin{aligned}\text{DTFT } \{x[n] * h[n]\} &= \sum_{n=-\infty}^{\infty} (x * h)[n] e^{-j\omega n} \\&= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n} \\&\stackrel{\text{Digital Signal Processing}}{=} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega(n-k)} e^{-j\omega k} \\&= \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)} \\&= H(e^{j\omega}) X(e^{j\omega})\end{aligned}$$

The convolution theorem

$$\begin{aligned}\text{DTFT } \{x[n] * h[n]\} &= \sum_{n=-\infty}^{\infty} (x * h)[n] e^{-j\omega n} \\&= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n} \\&\stackrel{\text{Digital Signal Processing}}{=} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega(n-k)} e^{-j\omega k} \\&= \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)} \\&= H(e^{j\omega}) X(e^{j\omega})\end{aligned}$$

Digital Signal Processing
Paolo Poggi and Martin Vetterli
@2013

The convolution theorem

$$\begin{aligned}\text{DTFT } \{x[n] * h[n]\} &= \sum_{n=-\infty}^{\infty} (x * h)[n] e^{-j\omega n} \\&= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n} \\&\stackrel{\text{Digital Signal Processing}}{=} \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega(n-k)} e^{-j\omega k} \\&= \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)} \\&= H(e^{j\omega}) X(e^{j\omega})\end{aligned}$$

$$H(e^{j\omega}) = \text{DTFT}\{h[n]\}$$

Two effects:

- ▶ **magnitude:** amplification ($|H(e^{j\omega})| > 1$) or attenuation ($|H(e^{j\omega})| < 1$) of input frequencies
- ▶ **phase:** overall delay and shape changes

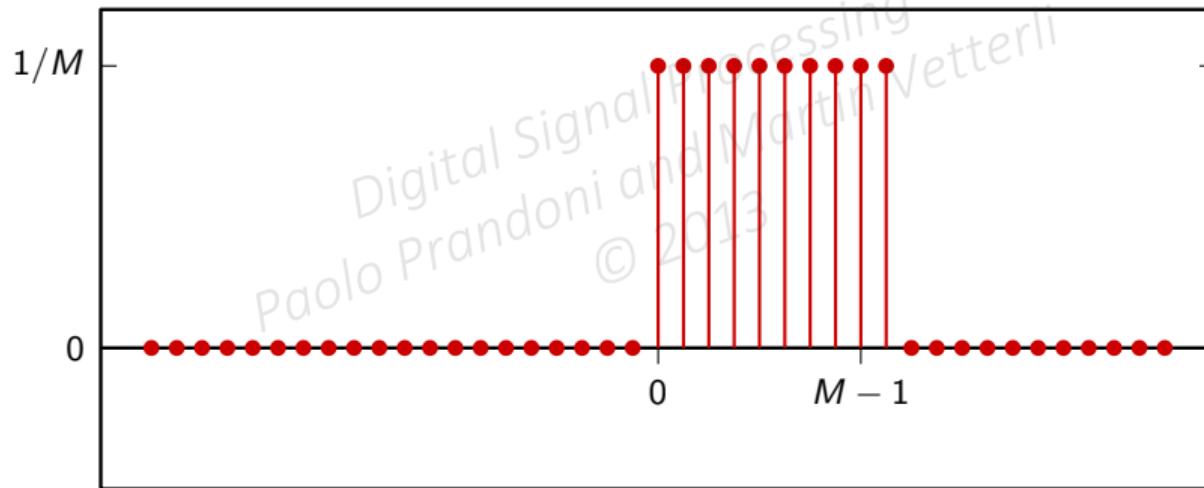
$$H(e^{j\omega}) = \text{DTFT}\{h[n]\}$$

Two effects:

- ▶ **magnitude:** amplification ($|H(e^{j\omega})| > 1$) or attenuation ($|H(e^{j\omega})| < 1$) of input frequencies
- ▶ **phase:** overall delay and shape changes

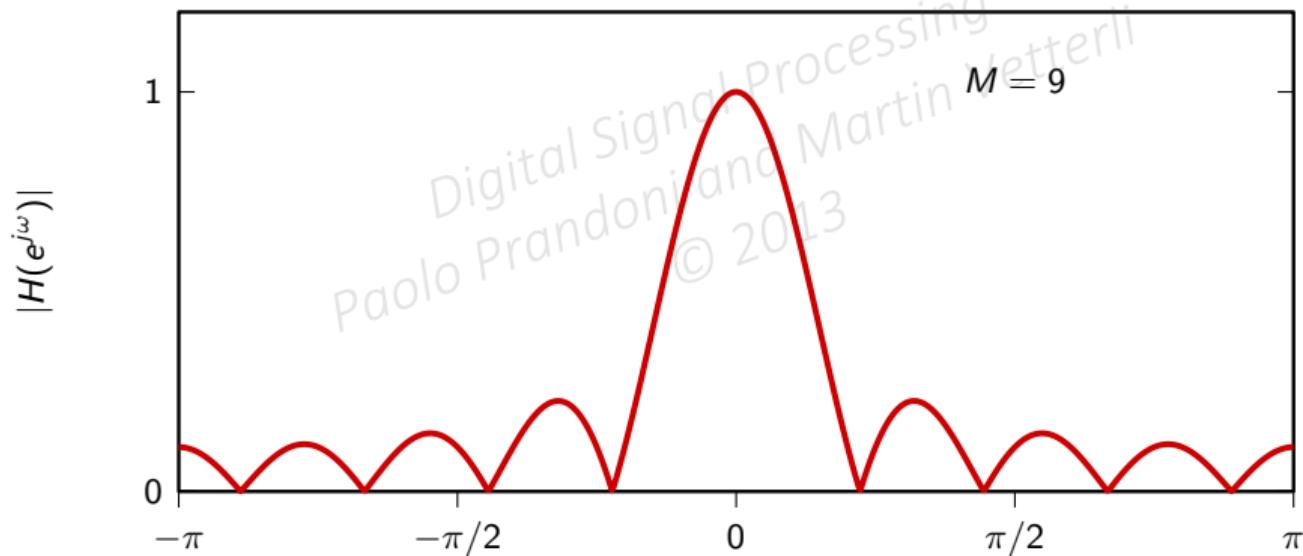
Moving Average revisited

$$h[n] = (u[n] - u[n - M])/M$$



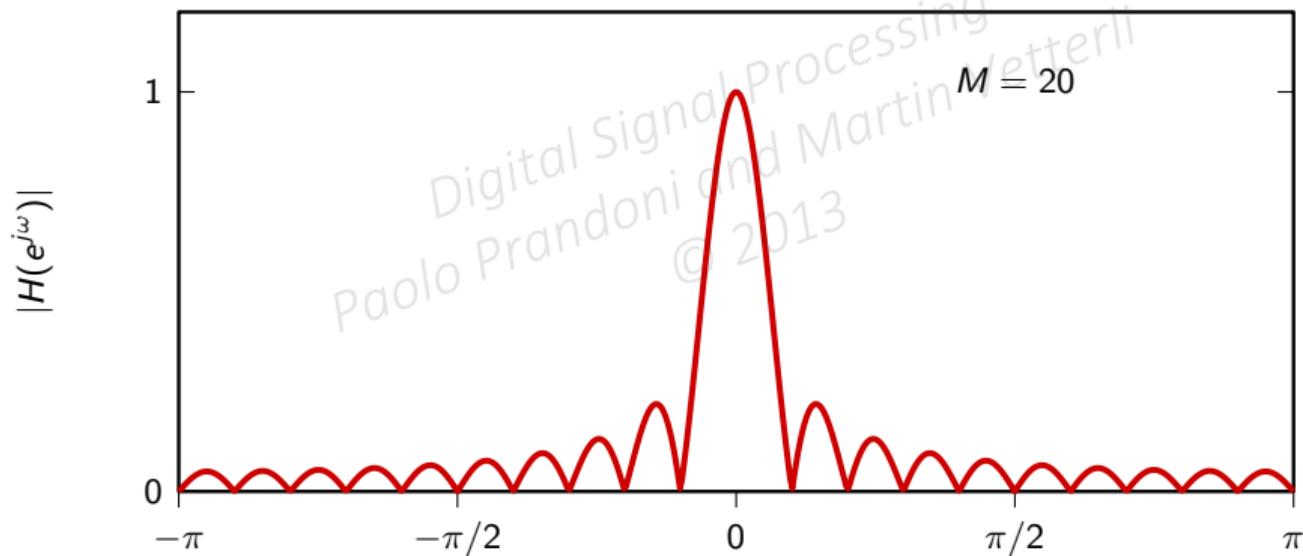
Moving Average, magnitude response

$$|H(e^{j\omega})| = \frac{1}{M} \left| \frac{\sin(\frac{\omega}{2}M)}{\sin(\frac{\omega}{2})} \right| \quad (\text{see Module 4.7})$$



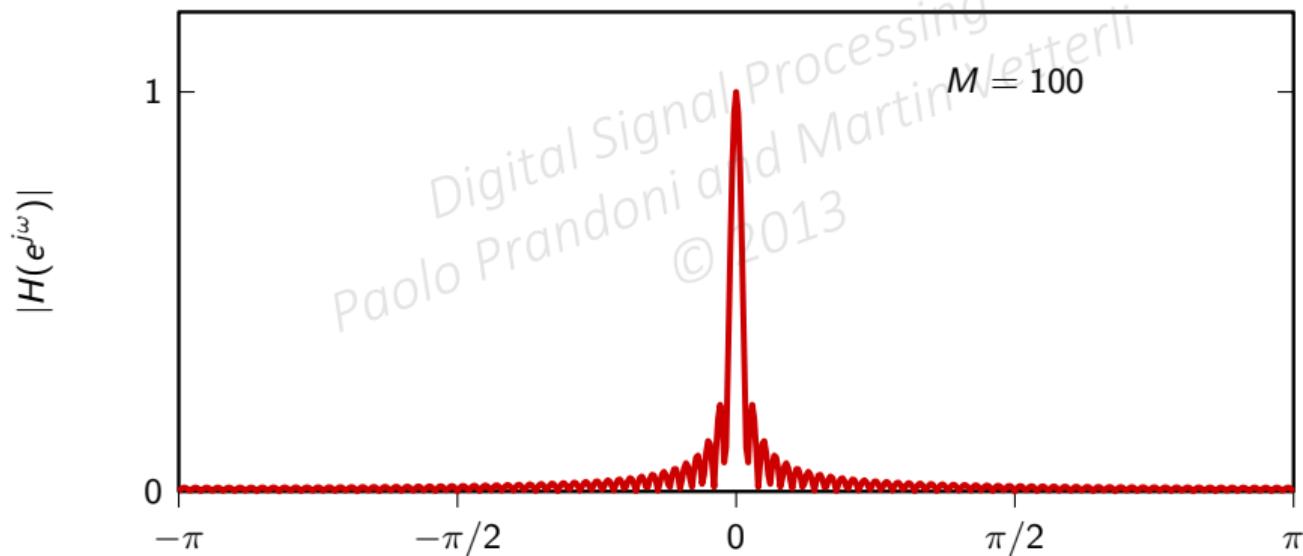
Moving Average, magnitude response

$$|H(e^{j\omega})| = \frac{1}{M} \left| \frac{\sin(\frac{\omega}{2}M)}{\sin(\frac{\omega}{2})} \right| \quad (\text{see Module 4.7})$$

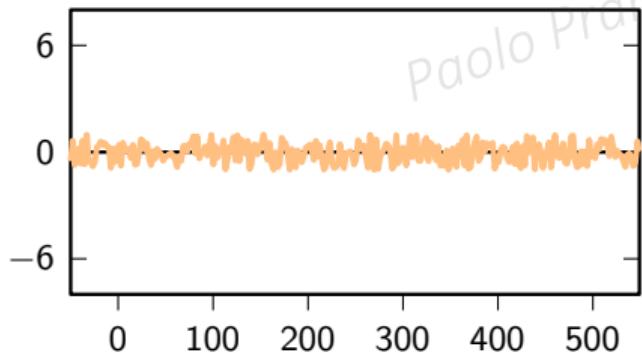
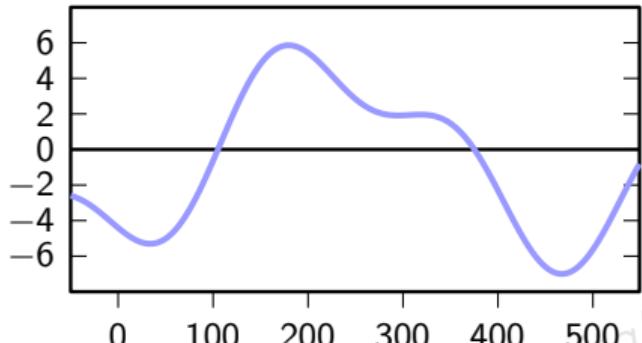


Moving Average, magnitude response

$$|H(e^{j\omega})| = \frac{1}{M} \left| \frac{\sin(\frac{\omega}{2}M)}{\sin(\frac{\omega}{2})} \right| \quad (\text{see Module 4.7})$$

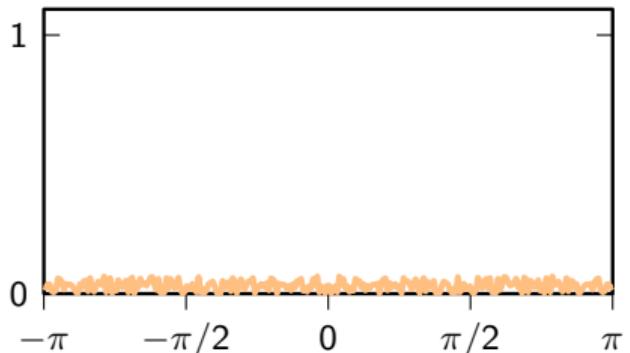
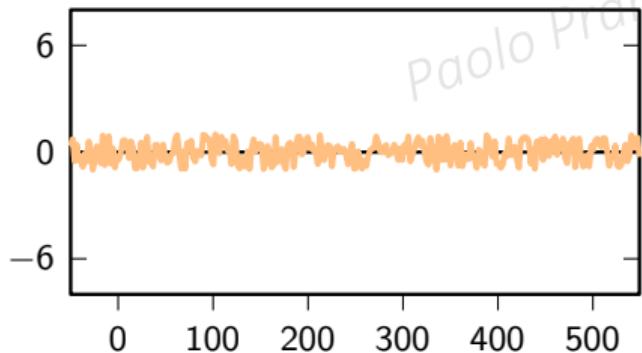
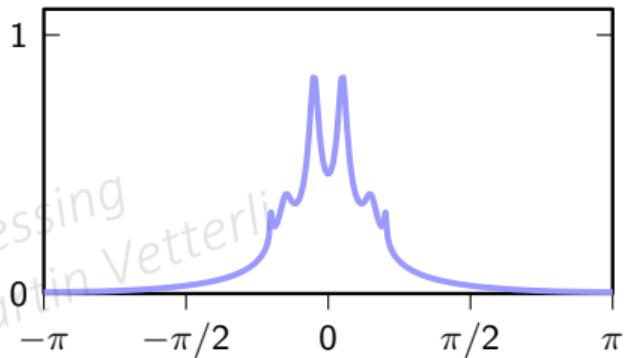
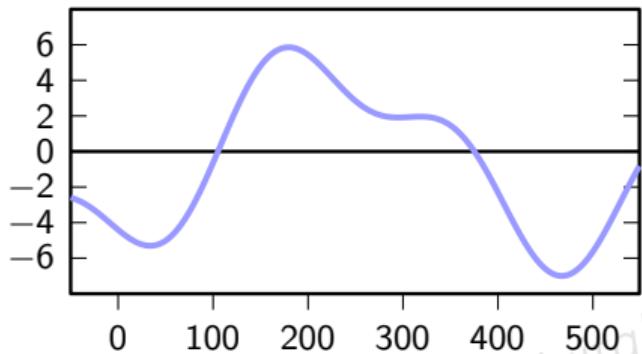


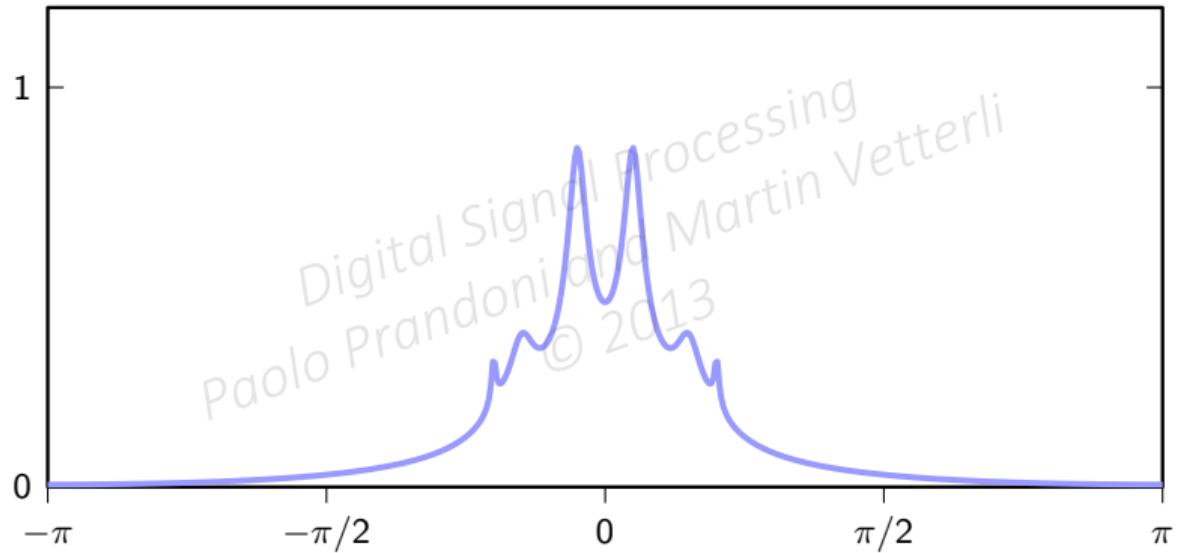
Denoising revisited

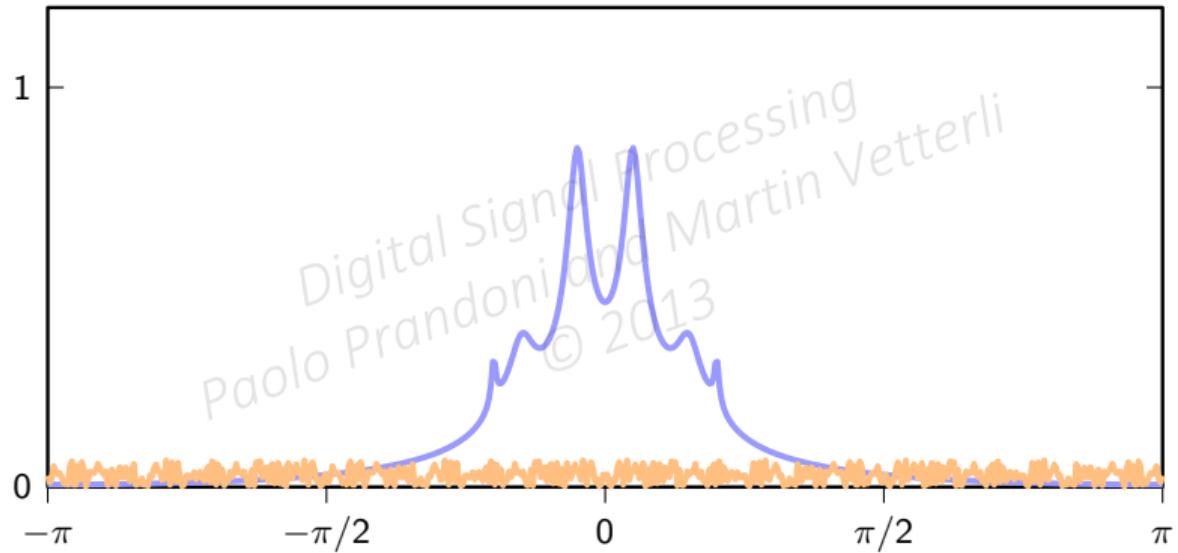


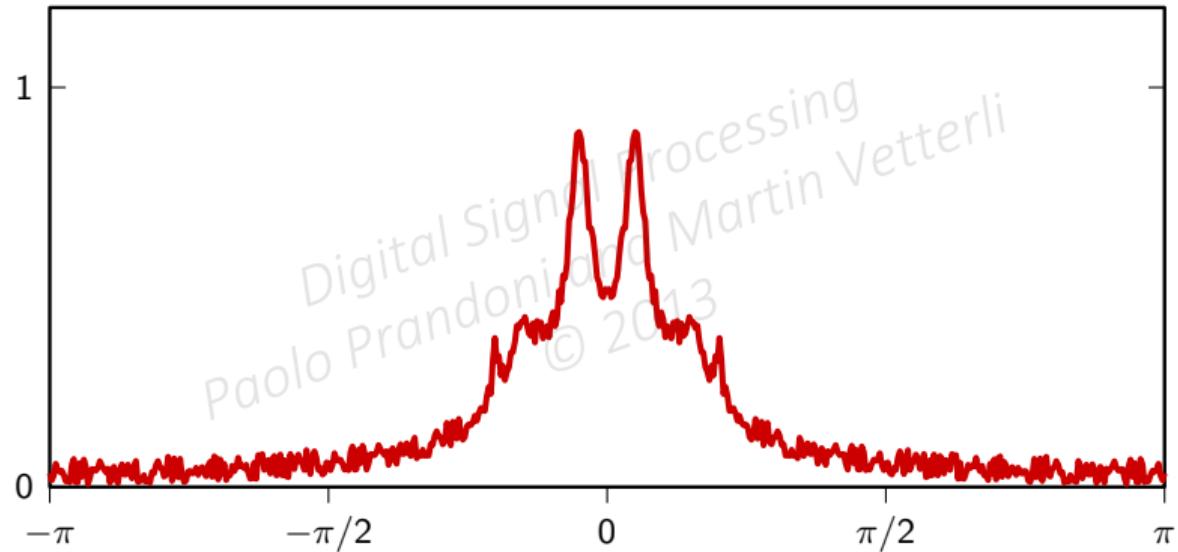
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

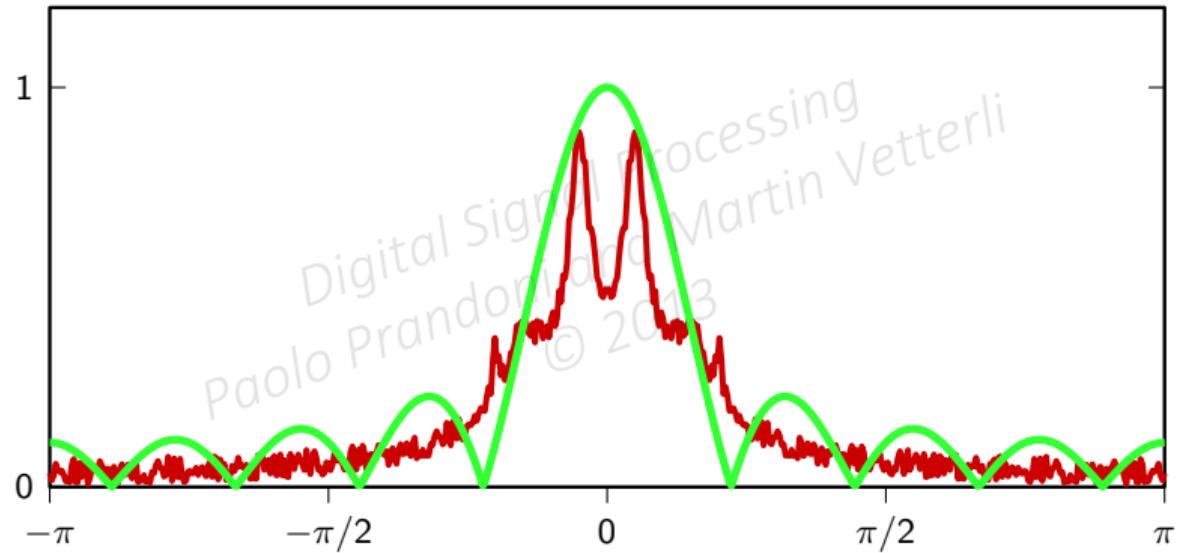
Denoising revisited

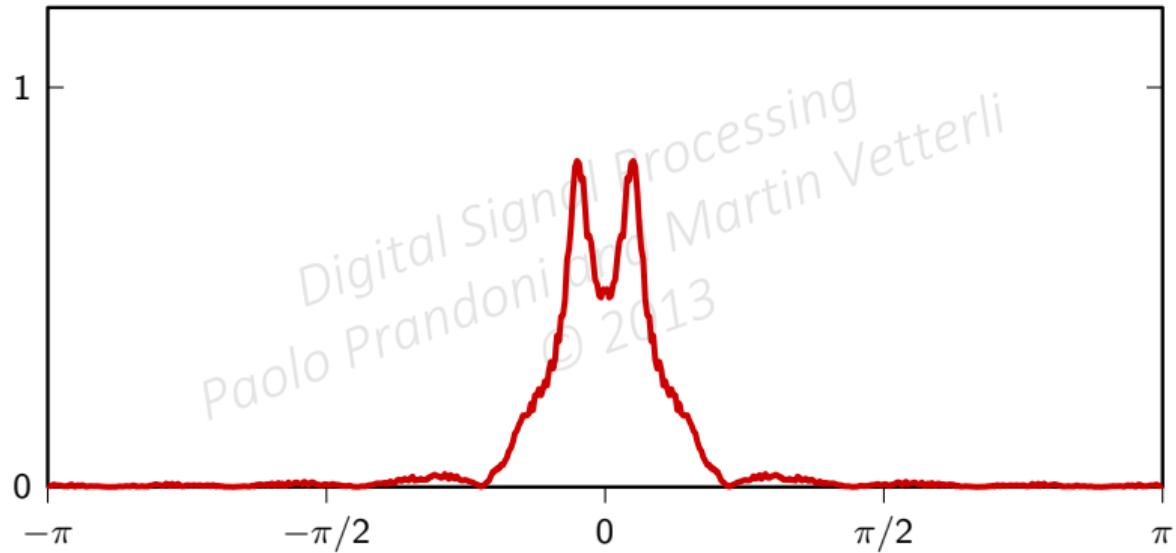


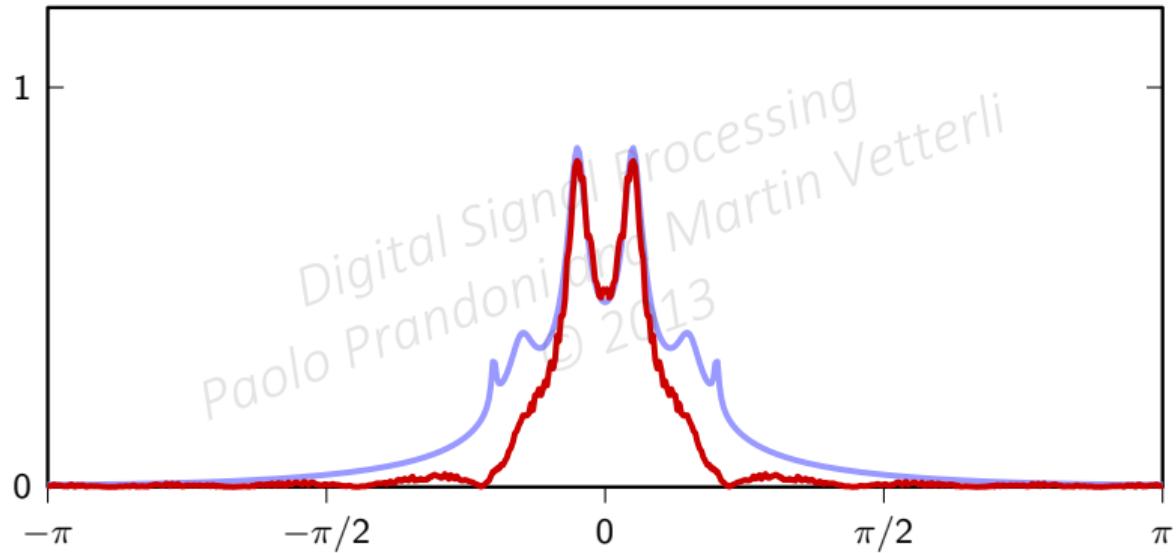




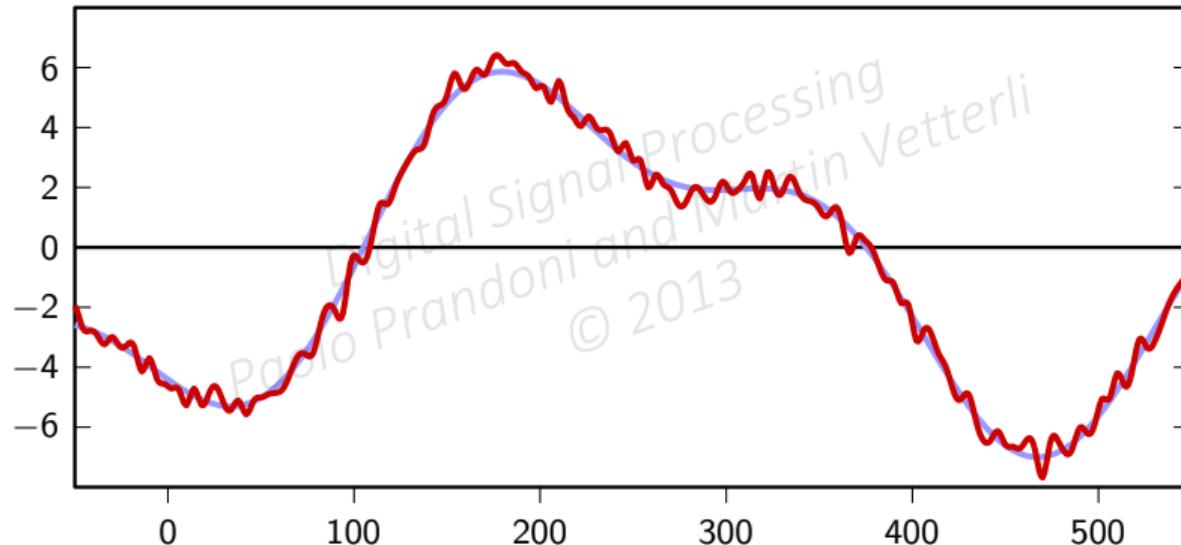




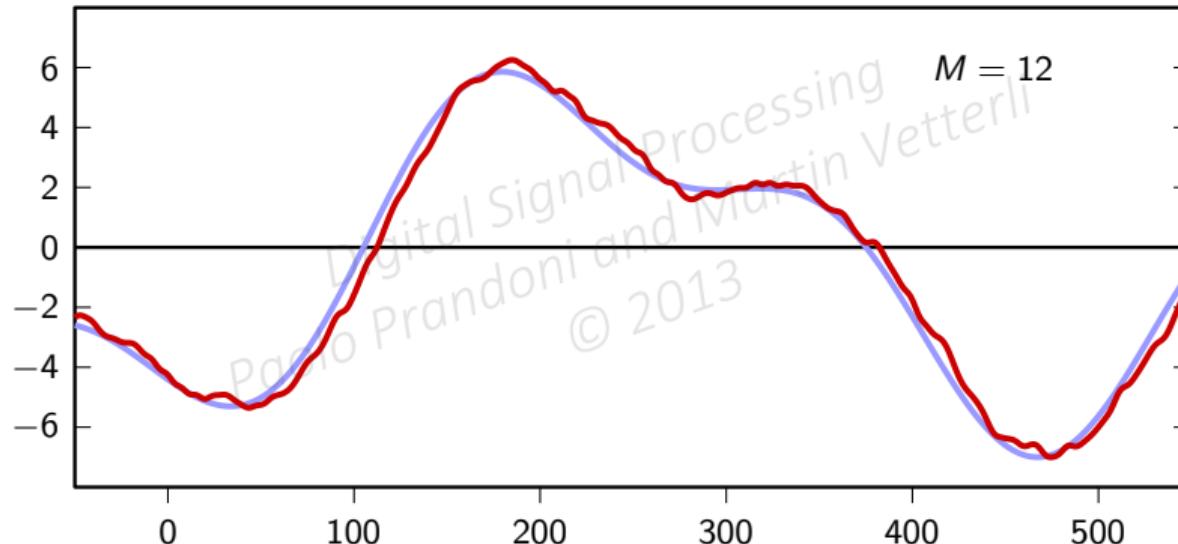




By the way, remember the time-domain analysis...



By the way, remember the time-domain analysis...



What about the phase?

Assume $|H(e^{j\omega})| = 1$

- ▶ zero phase: $\angle H(e^{j\omega}) = 0$
- ▶ linear phase: $\angle H(e^{j\omega}) = \phi$
- ▶ nonlinear phase

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

What about the phase?

Assume $|H(e^{j\omega})| = 1$

- ▶ zero phase: $\angle H(e^{j\omega}) = 0$
- ▶ linear phase: $\angle H(e^{j\omega}) = d\omega$
- ▶ nonlinear phase

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

What about the phase?

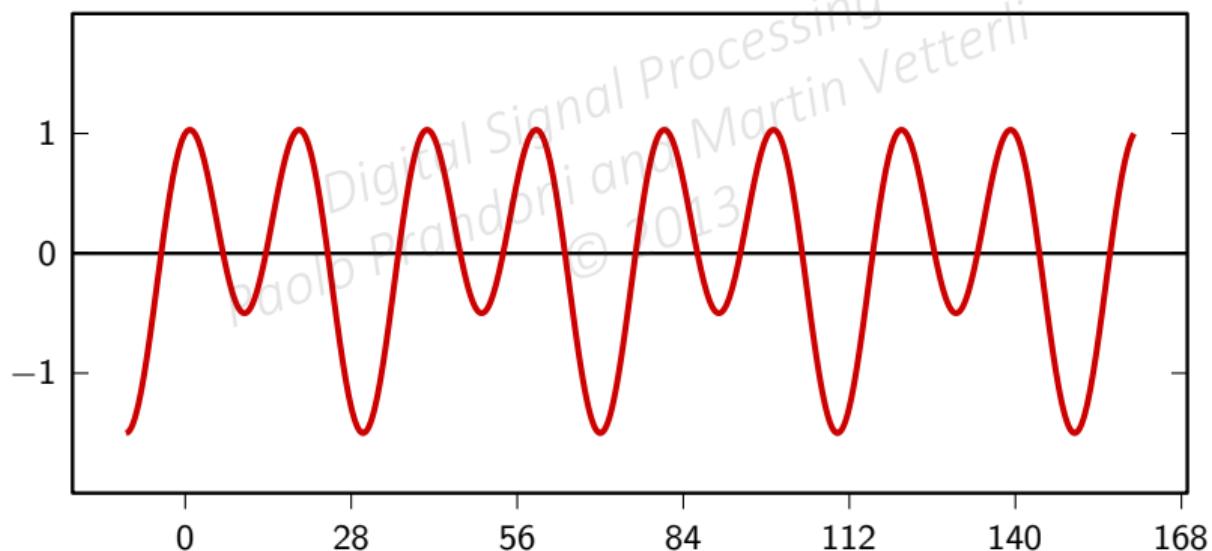
Assume $|H(e^{j\omega})| = 1$

- ▶ zero phase: $\angle H(e^{j\omega}) = 0$
- ▶ linear phase: $\angle H(e^{j\omega}) = d\omega$
- ▶ nonlinear phase

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

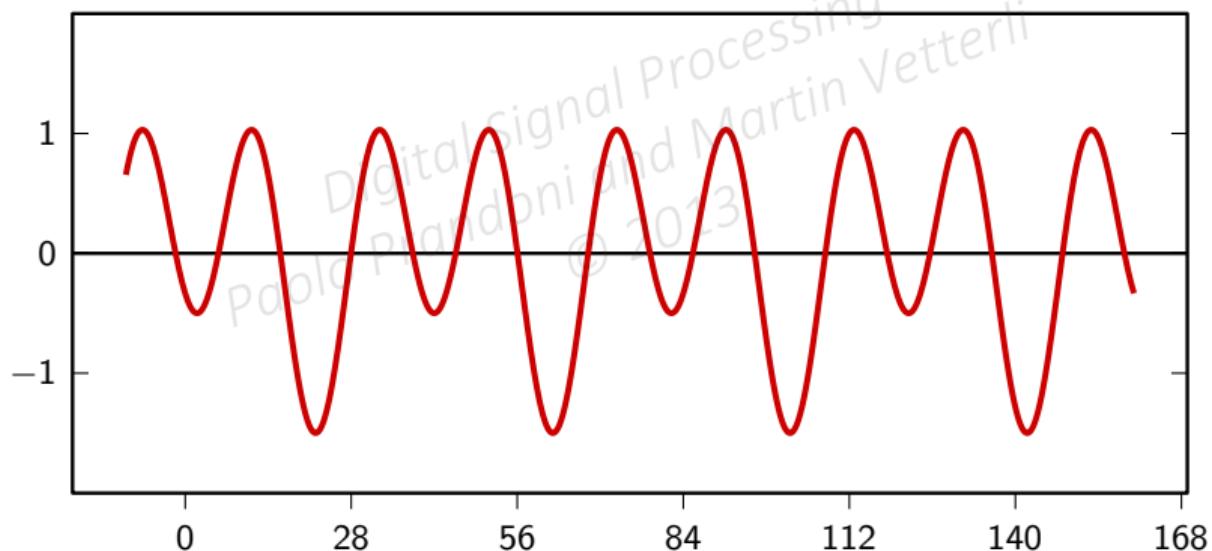
Phase and signal shape

$$x[n] = \frac{1}{2} \sin(\omega_0 n) + \cos(2\omega_0 n) \quad \omega_0 = \frac{2\pi}{40}$$



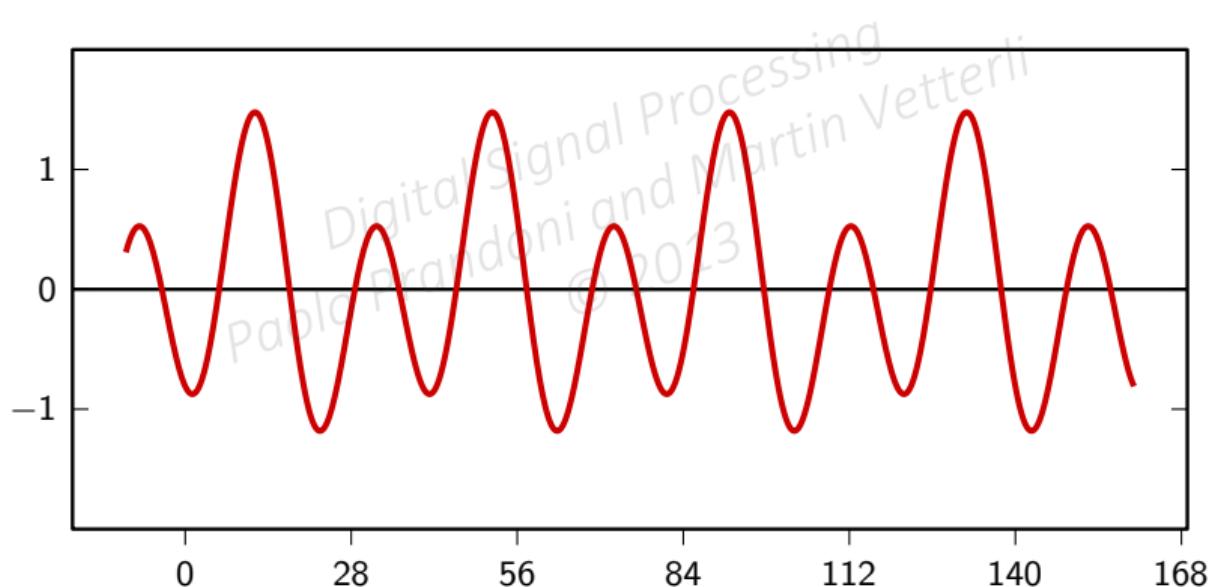
Phase and signal shape: linear phase

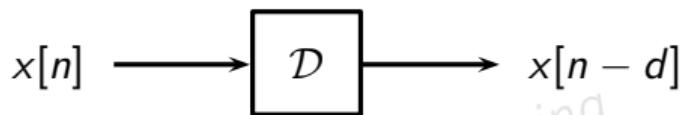
$$x[n] = \frac{1}{2} \sin(\omega_0 n + \theta_0) + \cos(2\omega_0 n + 2\theta_0) \quad \theta_0 = \frac{8\pi}{5}$$



Phase and signal shape: nonlinear phase

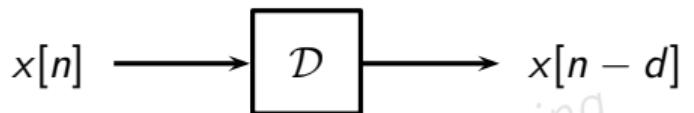
$$x[n] = \frac{1}{2} \sin(\omega_0 n) + \cos(2\omega_0 n + 2\theta_0)$$





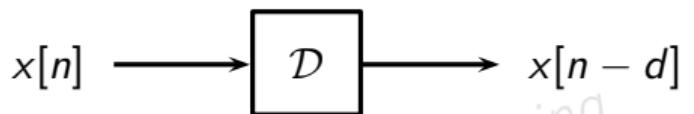
- ▶ $y[n] = x[n - d]$
- ▶ $Y(e^{j\omega}) = e^{-j\omega d} X(e^{j\omega})$
- ▶ $H(e^{j\omega}) = e^{-j\omega d}$
- ▶ linear phase term

Digital Signal Processing
Pdolo Prandoni and Martin Vetterli
© 2013



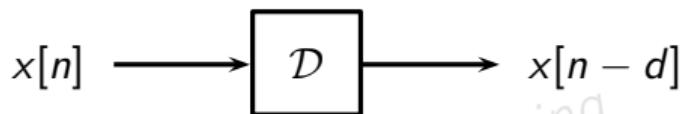
- ▶ $y[n] = x[n - d]$
- ▶ $Y(e^{j\omega}) = e^{-j\omega d} X(e^{j\omega})$
- ▶ $H(e^{j\omega}) = e^{-j\omega d}$
- ▶ linear phase term

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



- ▶ $y[n] = x[n - d]$
- ▶ $Y(e^{j\omega}) = e^{-j\omega d} X(e^{j\omega})$
- ▶ $H(e^{j\omega}) = e^{-j\omega d}$
- ▶ linear phase term

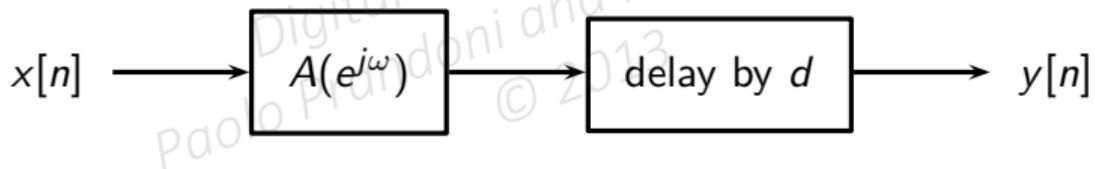
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



- ▶ $y[n] = x[n - d]$
- ▶ $Y(e^{j\omega}) = e^{-j\omega d} X(e^{j\omega})$
- ▶ $H(e^{j\omega}) = e^{-j\omega d}$
- ▶ linear phase term

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

In general, if $H(e^{j\omega}) = A(e^{j\omega})e^{-j\omega d}$, with $A(e^{j\omega}) \in \mathbb{R}$



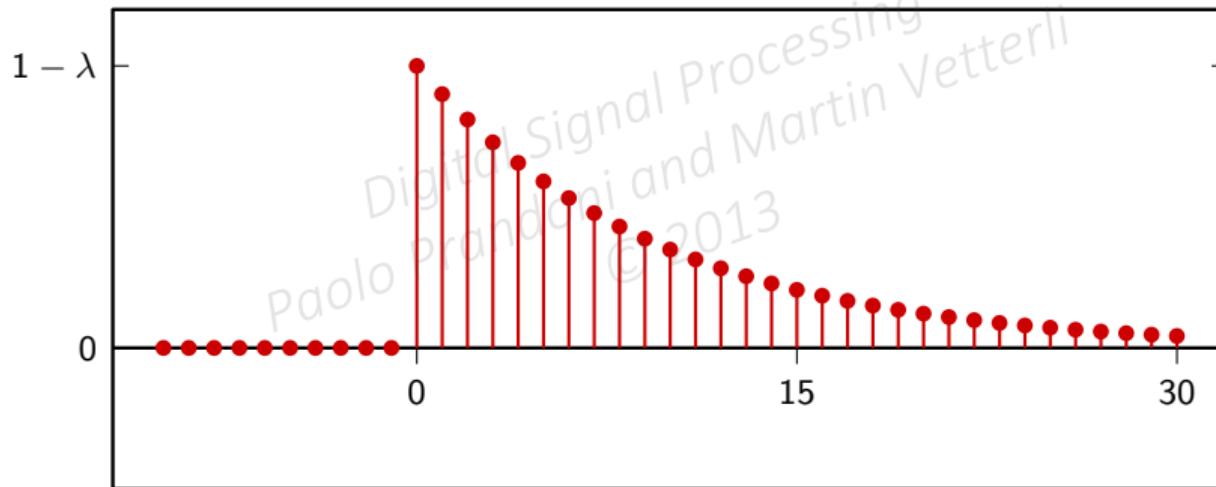
Moving Average is linear phase

$$H(e^{j\omega}) = \frac{1}{M} \frac{\sin\left(\frac{\omega}{2}M\right)}{\sin\left(\frac{\omega}{2}\right)} e^{-j\frac{M-1}{2}\omega}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Leaky integrator revisited

$$h[n] = (1 - \lambda)\lambda^n u[n]$$



$$H(e^{j\omega}) = \frac{1 - \lambda}{1 - \lambda e^{-j\omega}} \quad (\text{Module 4.4})$$

Finding magnitude and phase require a little algebra...

Leaky integrator revisited

Recall from complex algebra:

$$\frac{1}{a+jb} = \frac{a-jb}{a^2+b^2}$$

so that if $x = 1/(a+jb)$,

$$|x|^2 = \frac{1}{a^2+b^2}$$

$$\angle x = \tan^{-1} \left[-\frac{b}{a} \right]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

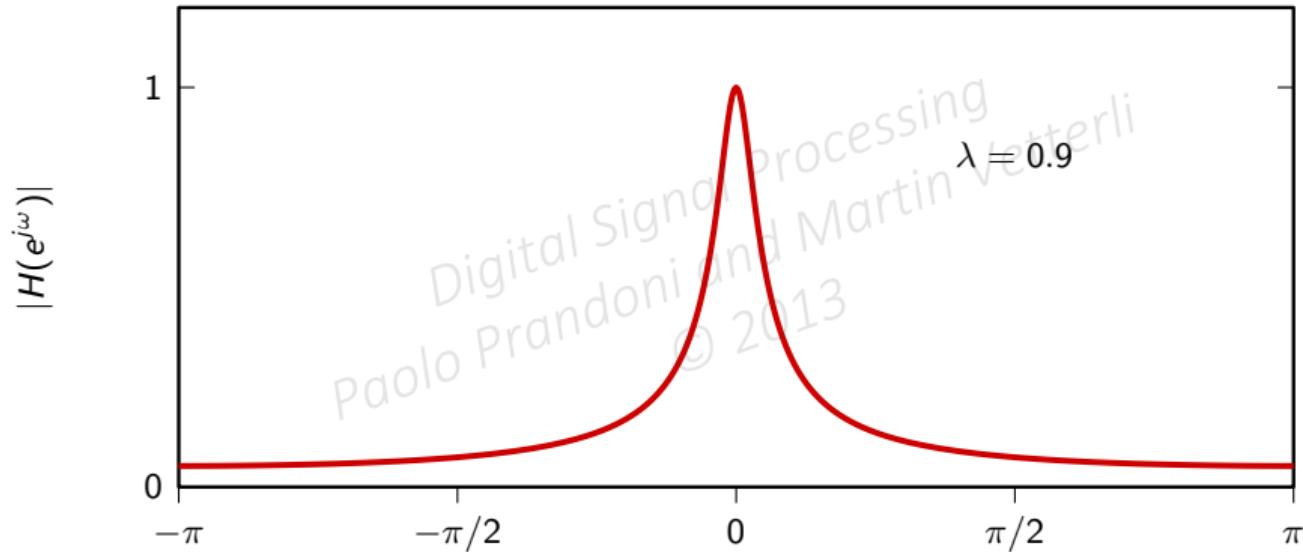
$$H(e^{j\omega}) = \frac{1 - \lambda}{(1 - \lambda \cos \omega) - j\lambda \sin \omega}$$

so that:

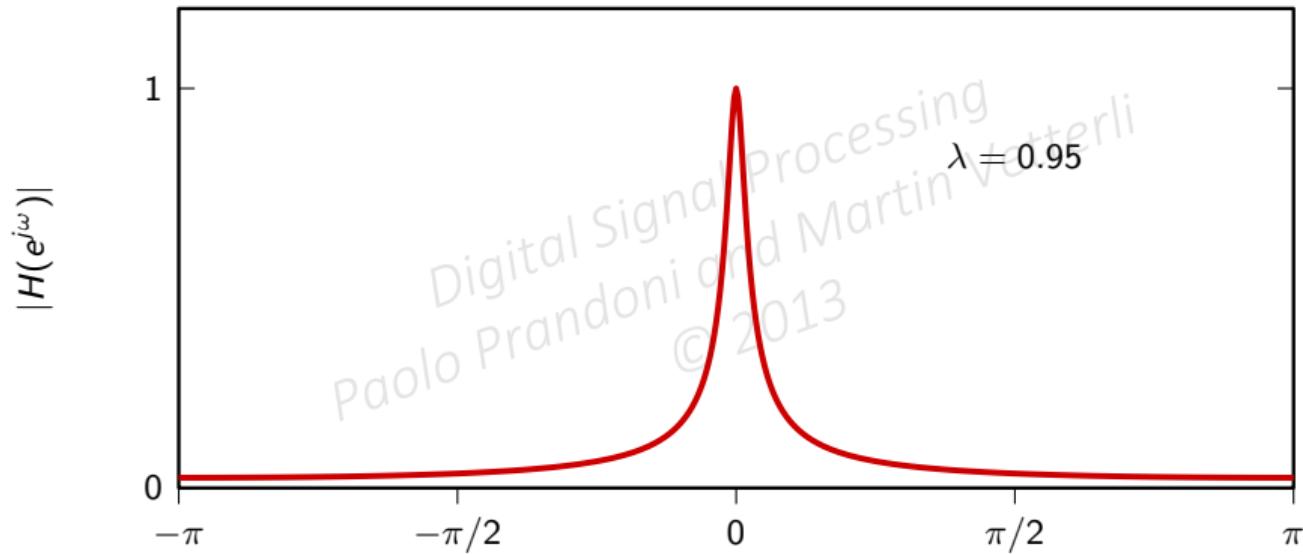
$$|H(e^{j\omega})|^2 = \frac{(1 - \lambda)^2}{1 - 2\lambda \cos \omega + \lambda^2}$$

$$\angle H(e^{j\omega}) = \tan^{-1} \left[\frac{\lambda \sin \omega}{1 - \lambda \cos \omega} \right]$$

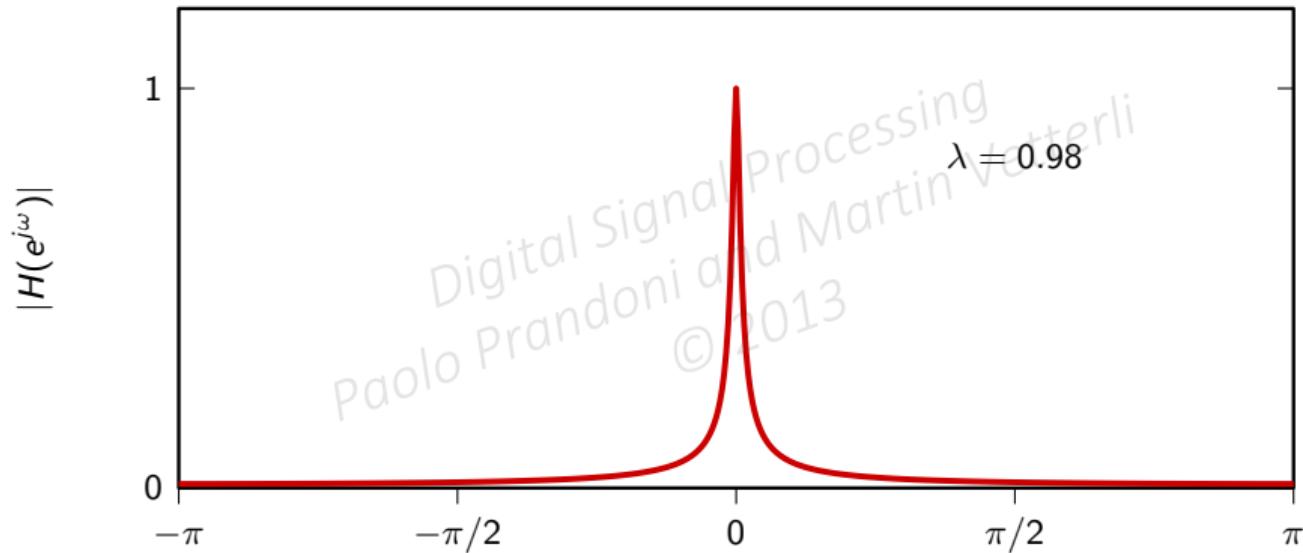
Leaky integrator, magnitude response



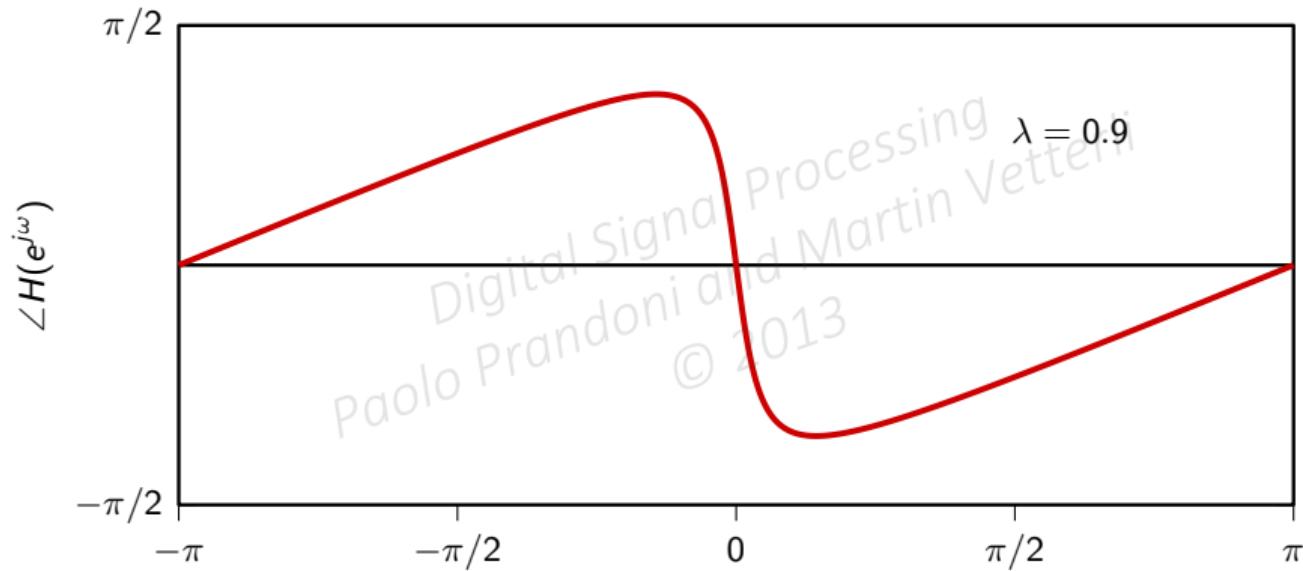
Leaky integrator, magnitude response



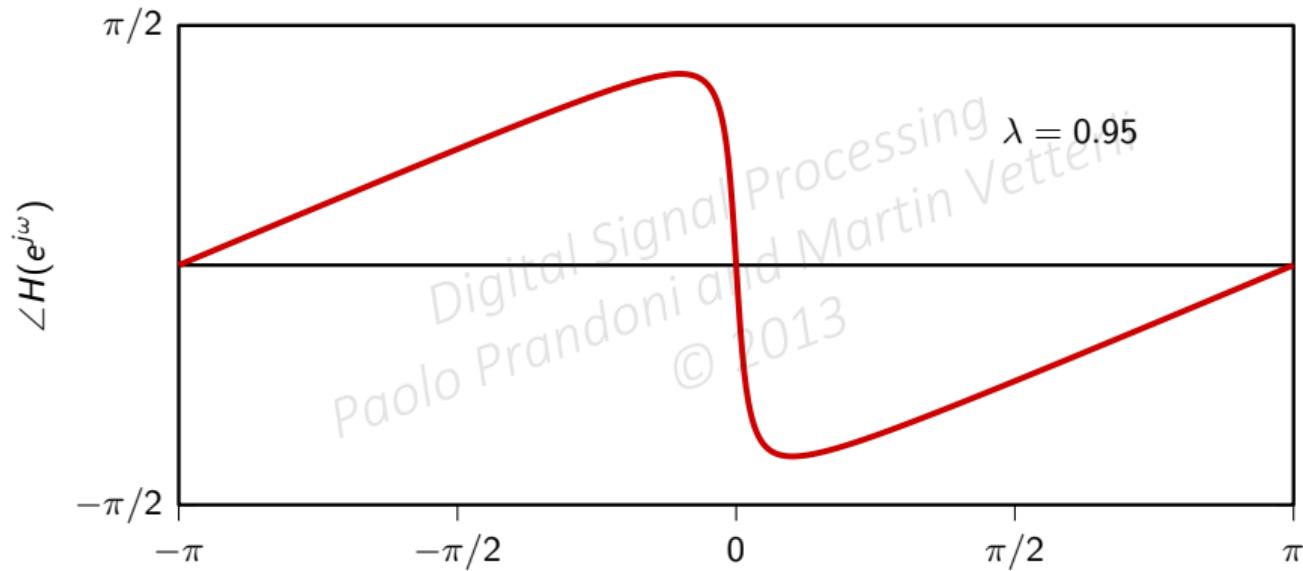
Leaky integrator, magnitude response



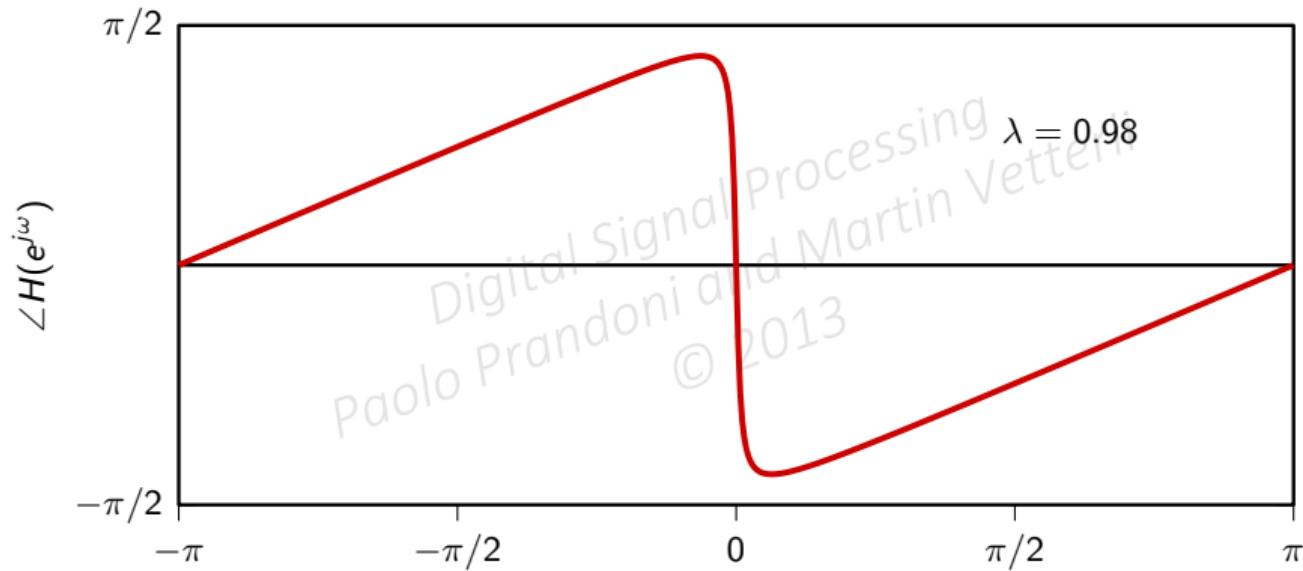
Leaky integrator, phase response



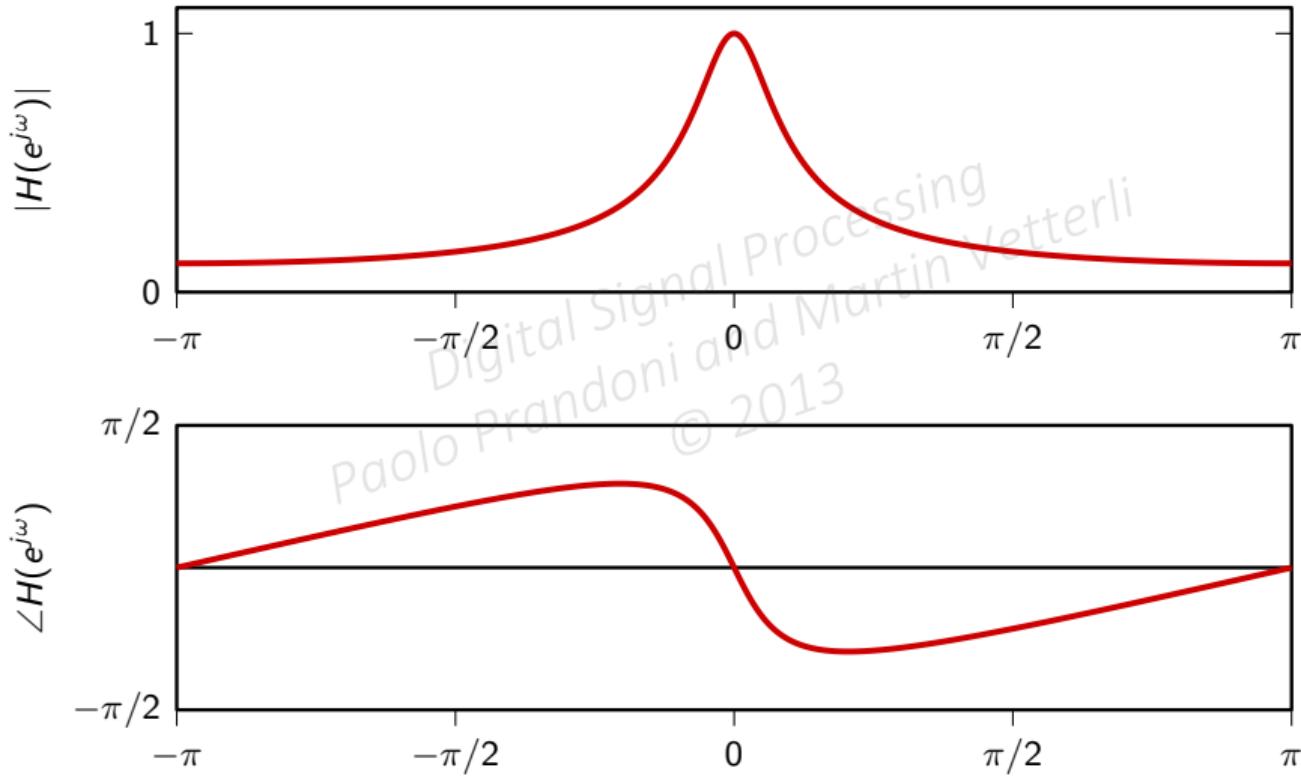
Leaky integrator, phase response



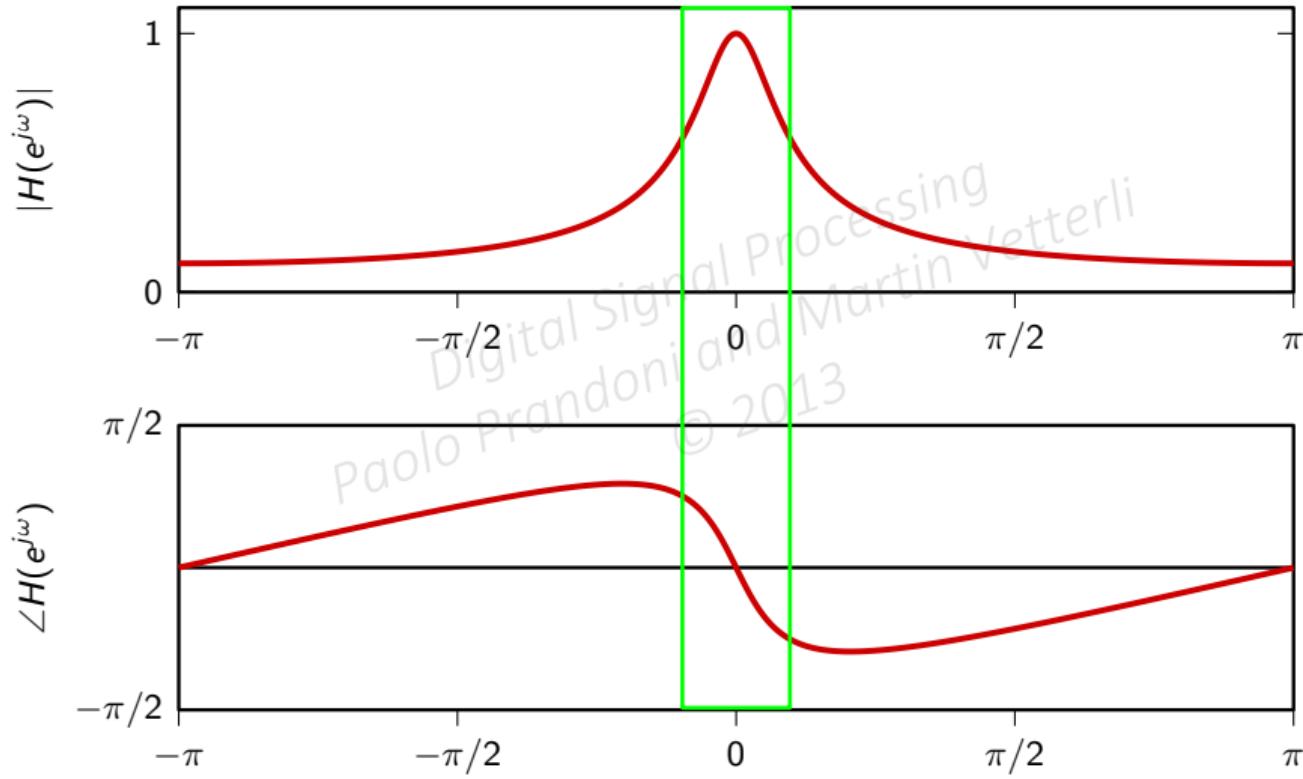
Leaky integrator, phase response



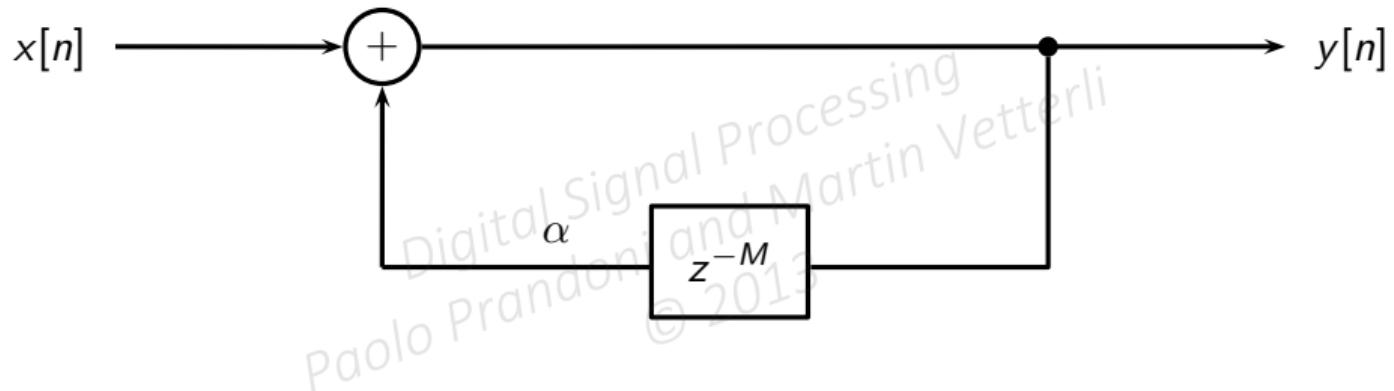
Phase is sufficiently linear where it matters



Phase is sufficiently linear where it matters

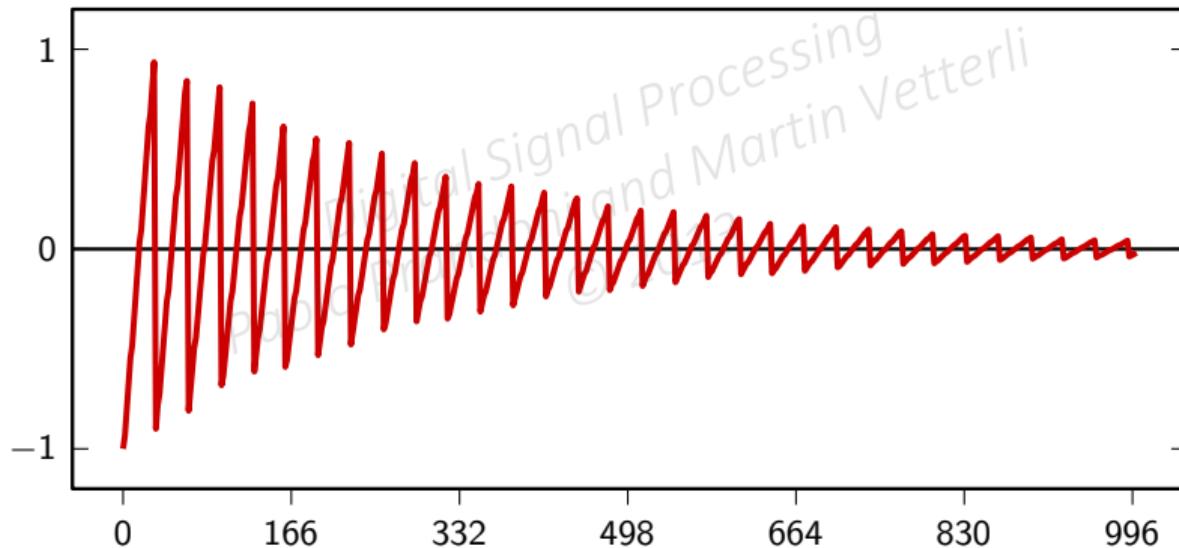


Karplus-Strong revisited, again!



$$y[n] = \alpha y[n - M] + x[n]$$

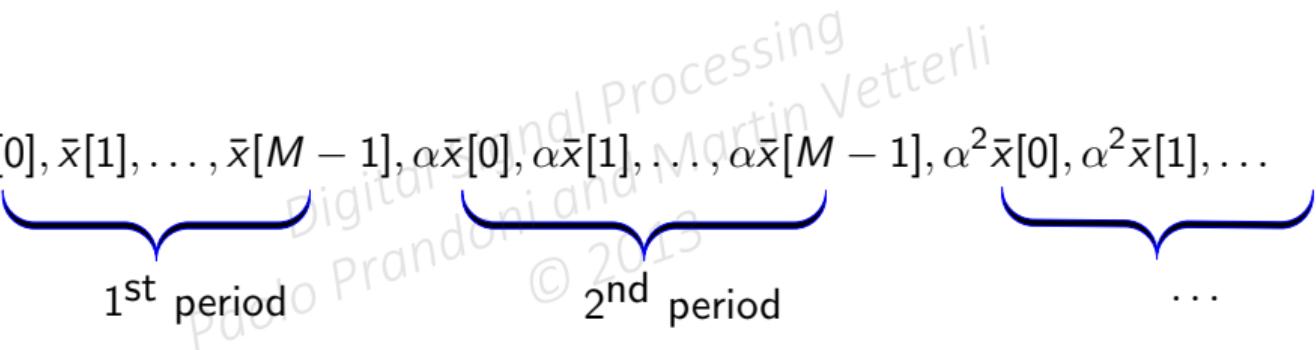
$$y[n] = \alpha^{\lfloor n/M \rfloor} \bar{x}[n \bmod M] u[n]$$



$$y[n] = \bar{x}[0], \bar{x}[1], \dots, \bar{x}[M-1], \alpha\bar{x}[0], \alpha\bar{x}[1], \dots, \alpha\bar{x}[M-1], \alpha^2\bar{x}[0], \alpha^2\bar{x}[1], \dots$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$y[n] = \bar{x}[0], \bar{x}[1], \dots, \bar{x}[M-1], \alpha\bar{x}[0], \alpha\bar{x}[1], \dots, \alpha\bar{x}[M-1], \alpha^2\bar{x}[0], \alpha^2\bar{x}[1], \dots$$


1st period 2nd period ...

key observation:

$$y[n] = \bar{x}[n] * w[n], \quad w[n] = \begin{cases} \alpha^k & \text{for } n = kM \\ 0 & \text{otherwise} \end{cases}$$

$$Y(e^{j\omega}) = \bar{X}(e^{j\omega})W(e^{j\omega})$$

key observation:

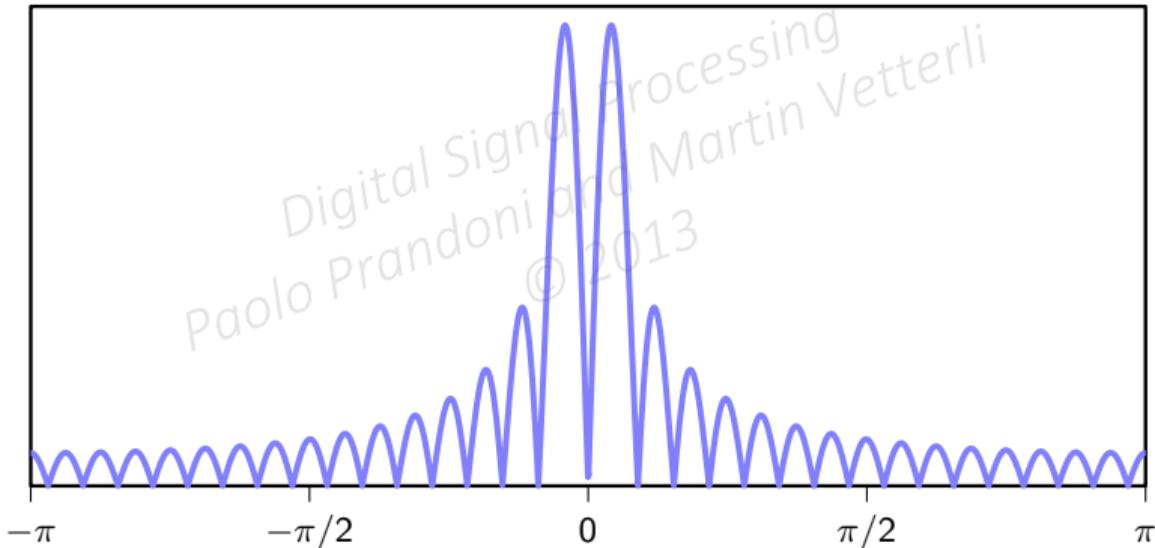
$$y[n] = \bar{x}[n] * w[n], \quad w[n] = \begin{cases} \alpha^k & \text{for } n = kM \\ 0 & \text{otherwise} \end{cases}$$

$$Y(e^{j\omega}) = \bar{X}(e^{j\omega})W(e^{j\omega})$$

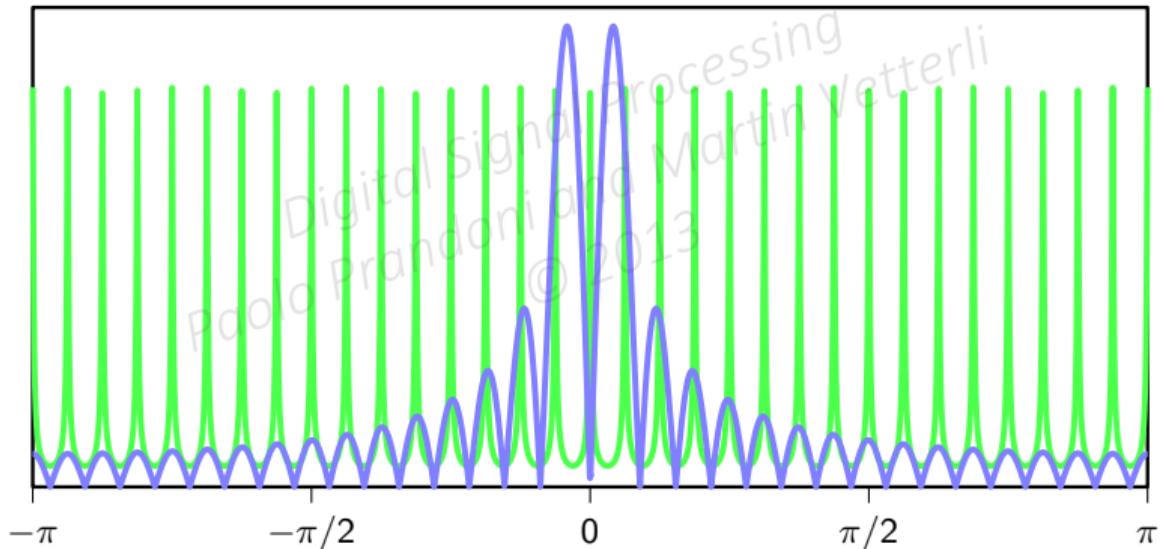
$$\bar{X}(e^{j\omega}) = e^{-j\omega} \left(\frac{M+1}{M-1} \right) \frac{1 - e^{-j(M-1)\omega}}{(1 - e^{-j\omega})^2} + \frac{1 - e^{-j(M+1)\omega}}{(1 - e^{-j\omega})^2}$$

$$W(e^{j\omega}) = \frac{1}{1 - \alpha e^{-j\omega M}}$$

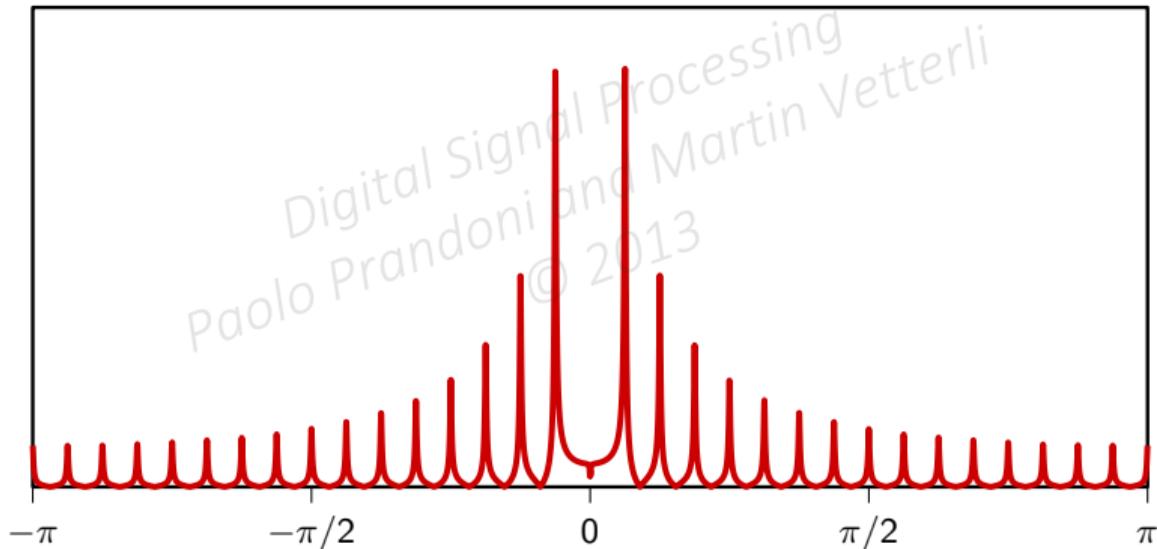
$$|\bar{X}(e^{j\omega})|$$



$$|W(e^{j\omega})|$$



$$|Y(e^{j\omega})|$$



END OF MODULE 5.4

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.5: Ideal Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter classification in the frequency domain
- ▶ Ideal filters
- ▶ Demodulation revisited

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter classification in the frequency domain
- ▶ Ideal filters
- ▶ Demodulation revisited

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter classification in the frequency domain
- ▶ Ideal filters
- ▶ Demodulation revisited

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to magnitude response

- ▶ Lowpass
- ▶ Highpass
- ▶ Bandpass
- ▶ Allpass

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average and Leaky Integrator are lowpass filters

Filter types according to magnitude response

- ▶ Lowpass
- ▶ Highpass
- ▶ Bandpass
- ▶ Allpass

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average and Leaky Integrator are lowpass filters

Filter types according to magnitude response

- ▶ Lowpass
- ▶ Highpass
- ▶ Bandpass
- ▶ Allpass

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average and Leaky Integrator are lowpass filters

Filter types according to magnitude response

- ▶ Lowpass
- ▶ Highpass
- ▶ Bandpass
- ▶ Allpass

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average and Leaky Integrator are lowpass filters

Filter types according to magnitude response

- ▶ Lowpass
- ▶ Highpass
- ▶ Bandpass
- ▶ Allpass

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Moving Average and Leaky Integrator are lowpass filters

Filter types according to phase response

- ▶ Linear phase
- ▶ Nonlinear phase

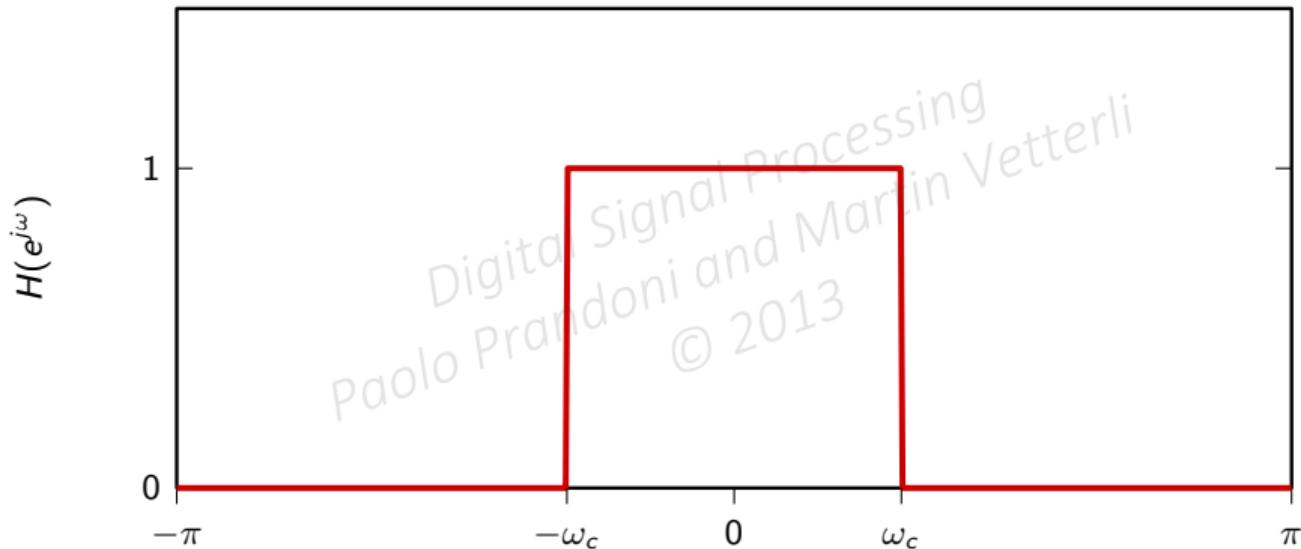
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Filter types according to phase response

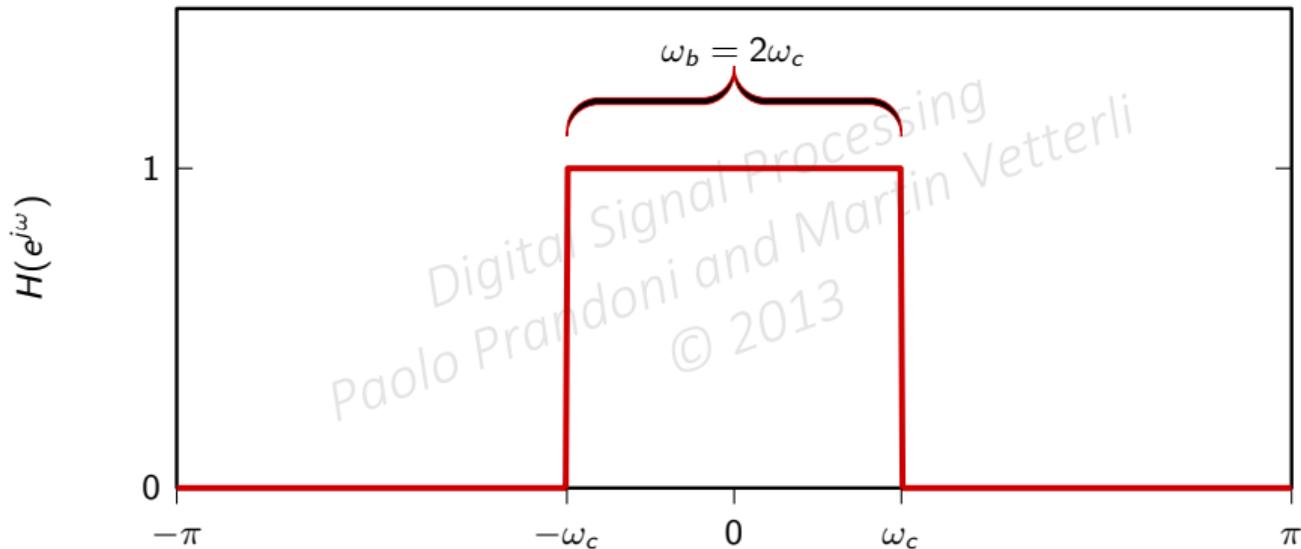
- ▶ Linear phase
- ▶ Nonlinear phase

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

What is the best lowpass we can think of?



What is the best lowpass we can think of?



$$H(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega| \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

- ▶ perfectly flat passband
- ▶ infinite attenuation in stopband
- ▶ zero-phase (no delay)

Digital Signal Processing
Prandoni and Martin Veccetti
PDPO © 2013

$$H(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega| \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

- ▶ perfectly flat passband
- ▶ infinite attenuation in stopband
- ▶ zero-phase (no delay)

$$H(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega| \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

- ▶ perfectly flat passband
- ▶ infinite attenuation in stopband
- ▶ zero-phase (no delay)

Ideal lowpass filter: impulse response

$$h[n] = \text{IDTFT} \{ H(e^{j\omega}) \}$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{jn\omega} d\omega$$

$$= \frac{1}{\pi n} \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j}$$

$$= \frac{\sin \omega_c n}{\pi n}$$

Ideal lowpass filter: impulse response

$$h[n] = \text{IDTFT} \{ H(e^{j\omega}) \}$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega$$

$$= \frac{1}{\pi n} \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j}$$

$$= \frac{\sin \omega_c n}{\pi n}$$

Ideal lowpass filter: impulse response

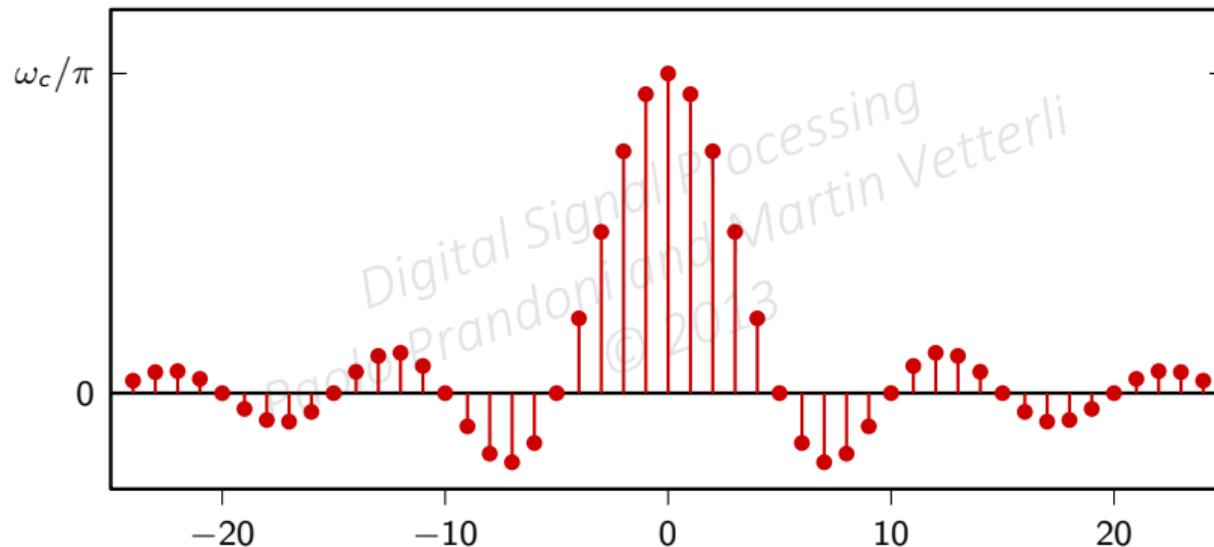
$$\begin{aligned} h[n] &= \text{IDTFT} \{ H(e^{j\omega}) \} \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega \\ &= \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} d\omega \\ &= \frac{1}{\pi n} \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j} \\ &= \frac{\sin \omega_c n}{\pi n} \end{aligned}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

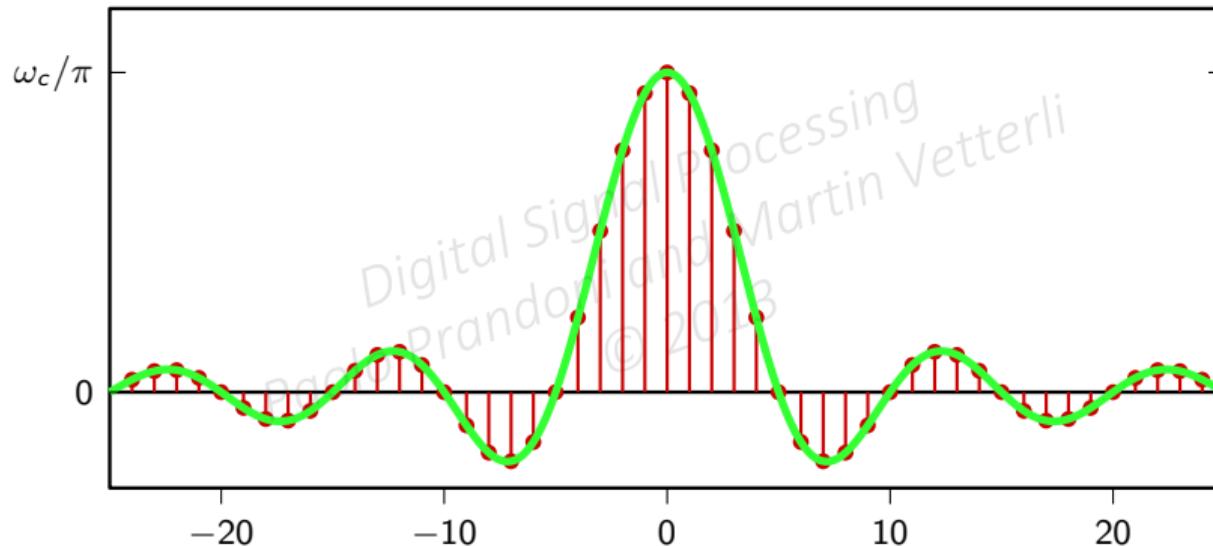
$$\begin{aligned} h[n] &= \text{IDTFT} \{ H(e^{j\omega}) \} \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega \\ &= \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} d\omega \\ &= \frac{1}{\pi n} \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j} \\ &= \frac{\sin \omega_c n}{\pi n} \end{aligned}$$

$$\begin{aligned} h[n] &= \text{IDTFT} \{ H(e^{j\omega}) \} \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega \\ &= \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} d\omega \\ &= \frac{1}{\pi n} \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j} \\ &= \frac{\sin \omega_c n}{\pi n} \end{aligned}$$

Ideal lowpass filter: impulse response



Ideal lowpass filter: impulse response



- ▶ impulse response is infinite support, two-sided
 - ⇒ cannot compute the output in a finite amount of time
 - ⇒ that's why it's called "ideal"
- ▶ impulse response decays slowly in time
 - ⇒ we need a lot of samples for a good approximation

- ▶ impulse response is infinite support, two-sided
 - ⇒ cannot compute the output in a finite amount of time
 - ⇒ that's why it's called "ideal"
- ▶ impulse response decays slowly in time
 - ⇒ we need a lot of samples for a good approximation

The sinc-rect pair:

$$\text{rect}(x) = \begin{cases} 1 & |x| \leq 1/2 \\ 0 & |x| > 1/2 \end{cases}$$

$$\text{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

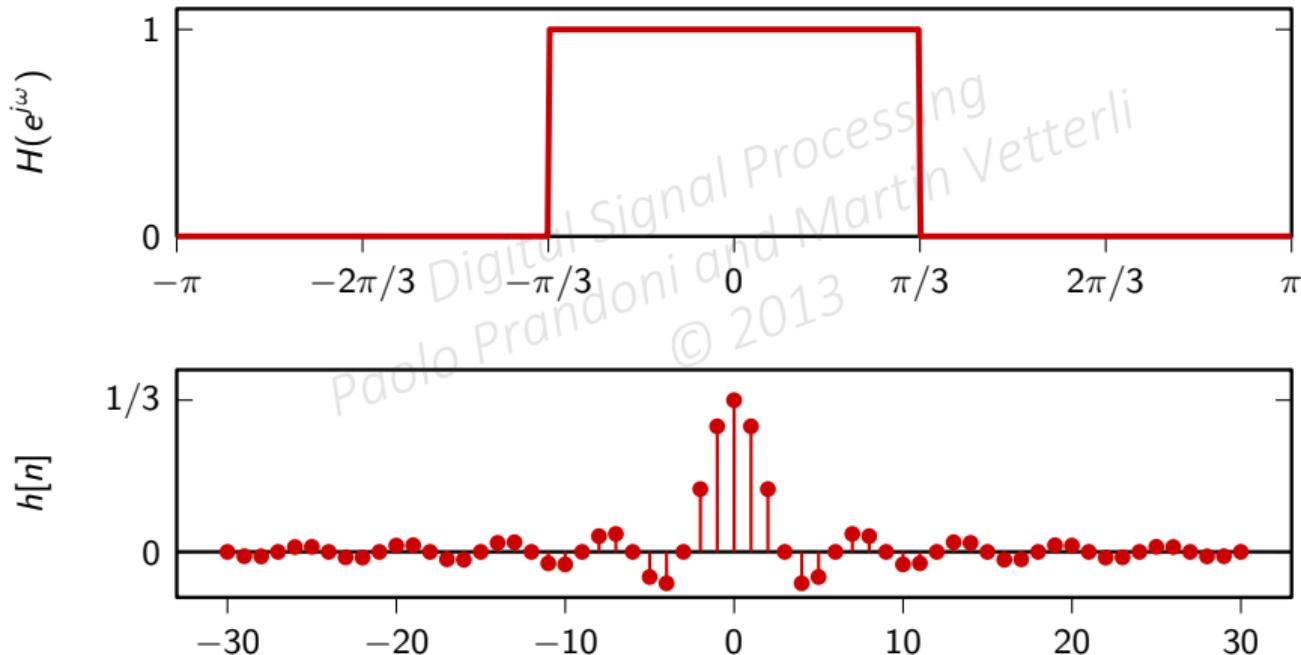
(note that $\text{sinc}(x) = 0$ when x is a nonzero integer)

The ideal lowpass in canonical form

$$\text{rect}\left(\frac{\omega}{2\omega_c}\right) \xleftrightarrow{\text{DTFT}} \frac{\omega_c}{\pi} \text{sinc}\left(\frac{\omega_c}{\pi}n\right)$$

Example

$$\omega_c = \pi/3: H(e^{j\omega}) = \text{rect}(3\omega/2\pi), h[n] = (1/3)\text{sinc}(n/3)$$



- ▶ the sinc is not absolutely summable

- ▶ the ideal lowpass is not BIBO stable!

- ▶ example for $\omega_c = \pi/3$: $h[n] = (1/3) \cdot \text{sinc}(n/3)$

- ▶ take $x[n] = \text{sign}\{\text{sinc}(-n/3)\}$ and

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$y[0] = (x * h)[0] = \frac{1}{3} \sum_{k=-\infty}^{\infty} |\text{sinc}(k/3)| = \infty$$

- ▶ the sinc is not absolutely summable
- ▶ the ideal lowpass is not BIBO stable!
- ▶ example for $\omega_c = \pi/3$: $h[n] = (1/3) \cdot \text{sinc}(n/3)$
- ▶ take $x[n] = \text{sign}\{\text{sinc}(-n/3)\}$ and

$$y[0] = (x * h)[0] = \frac{1}{3} \sum_{k=-\infty}^{\infty} |\text{sinc}(k/3)| = \infty$$

- ▶ the sinc is not absolutely summable
- ▶ the ideal lowpass is not BIBO stable!
- ▶ example for $\omega_c = \pi/3$: $h[n] = (1/3) \operatorname{sinc}(n/3)$
- ▶ take $x[n] = \operatorname{sign}\{\operatorname{sinc}(-n/3)\}$ and

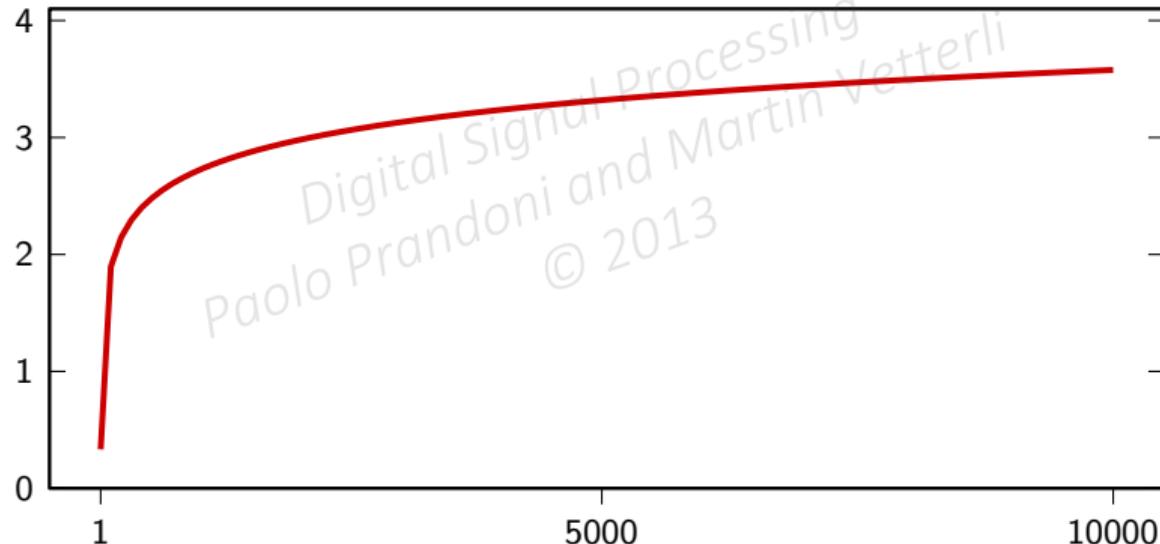
$$y[0] = (x * h)[0] = \frac{1}{3} \sum_{k=-\infty}^{\infty} |\operatorname{sinc}(k/3)| = \infty$$

- ▶ the sinc is not absolutely summable
- ▶ the ideal lowpass is not BIBO stable!
- ▶ example for $\omega_c = \pi/3$: $h[n] = (1/3) \operatorname{sinc}(n/3)$
- ▶ take $x[n] = \operatorname{sign}\{\operatorname{sinc}(-n/3)\}$ and

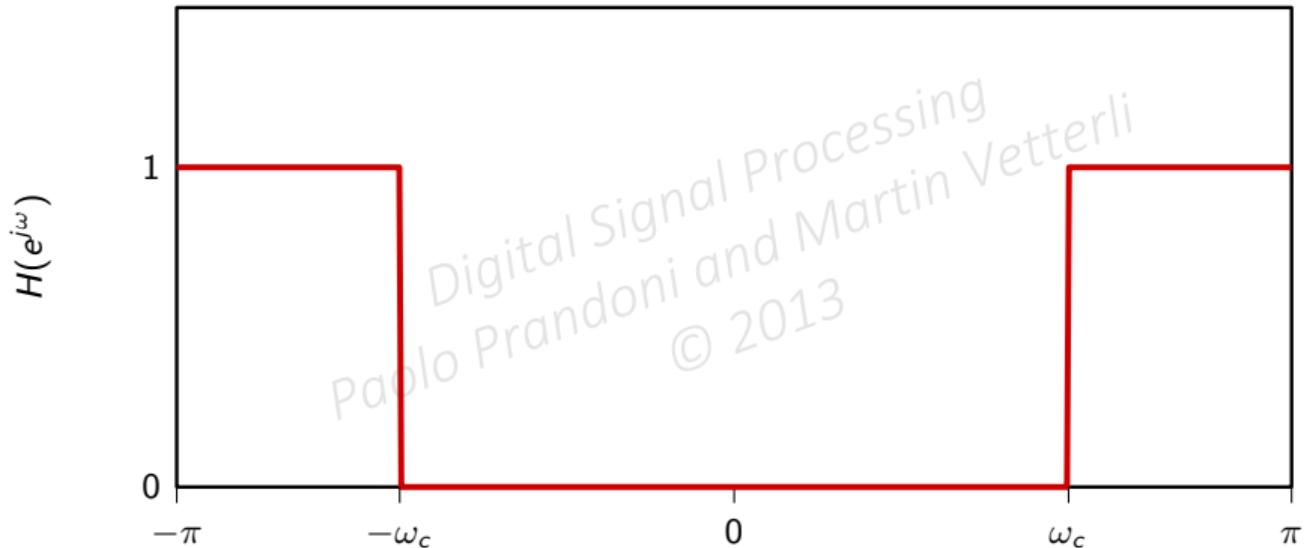
$$y[0] = (x * h)[0] = \frac{1}{3} \sum_{k=-\infty}^{\infty} |\operatorname{sinc}(k/3)| = \infty$$

Divergence is however very slow...

$$s(n) = (1/3) \sum_{k=-n}^n |\text{sinc}(k/3)|$$



Ideal highpass filter



Ideal highpass filter

$$H_{hp}(e^{j\omega}) = \begin{cases} 1 & \text{for } \pi \geq |\omega| \geq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $H_{hp}(e^{j\omega}) \stackrel{\text{Definition}}{=} 1 - H_{lp}(e^{j\omega})$

$$h_{hp}[n] = \delta[n] - \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right)$$

$$H_{hp}(e^{j\omega}) = \begin{cases} 1 & \text{for } \pi \geq |\omega| \geq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

$$H_{hp}(e^{j\omega}) = 1 - H_{lp}(e^{j\omega})$$

$$h_{hp}[n] = \delta[n] - \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right)$$

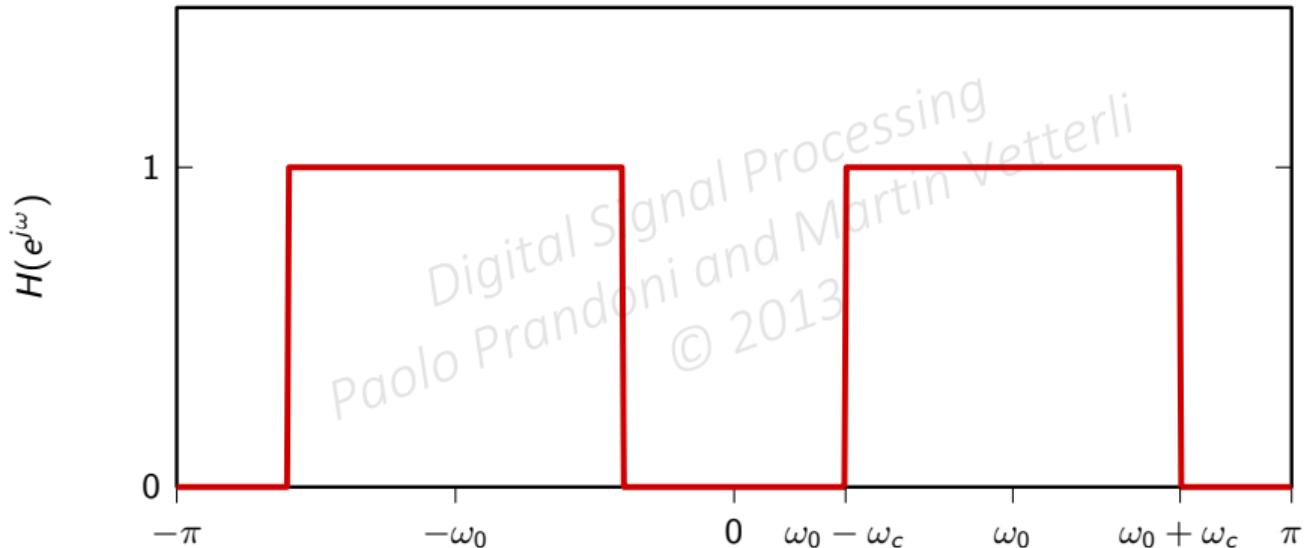
Ideal highpass filter

$$H_{hp}(e^{j\omega}) = \begin{cases} 1 & \text{for } \pi \geq |\omega| \geq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (\text{2}\pi\text{-periodicity implicit})$$

$$H_{hp}(e^{j\omega}) = 1 - H_{lp}(e^{j\omega})$$

$$h_{hp}[n] = \delta[n] - \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right)$$

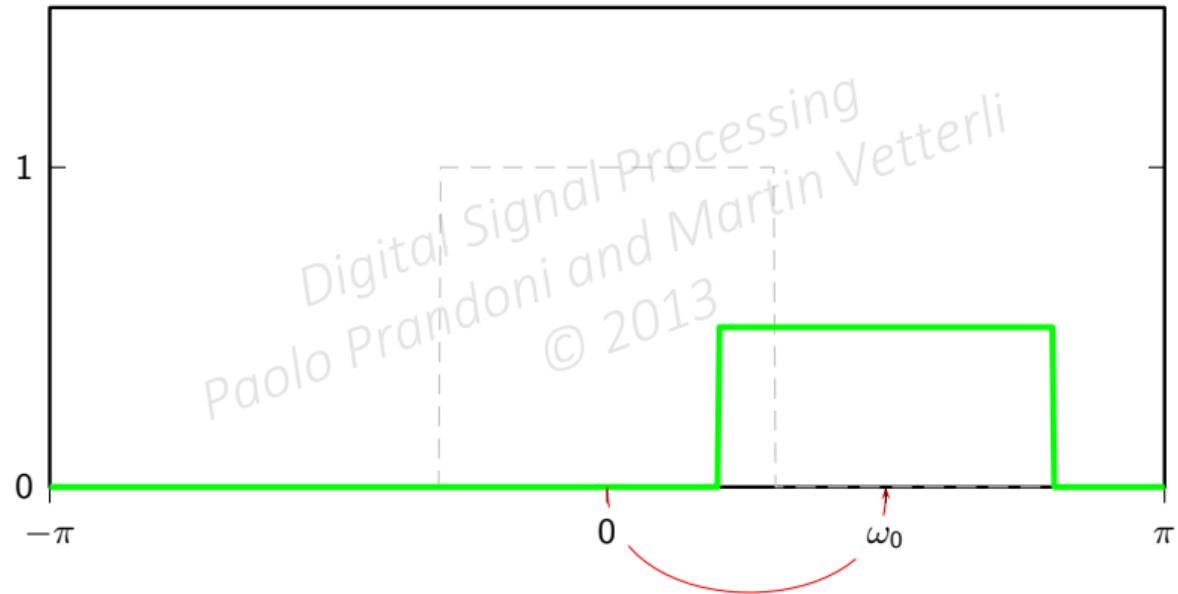
Ideal bandpass filter



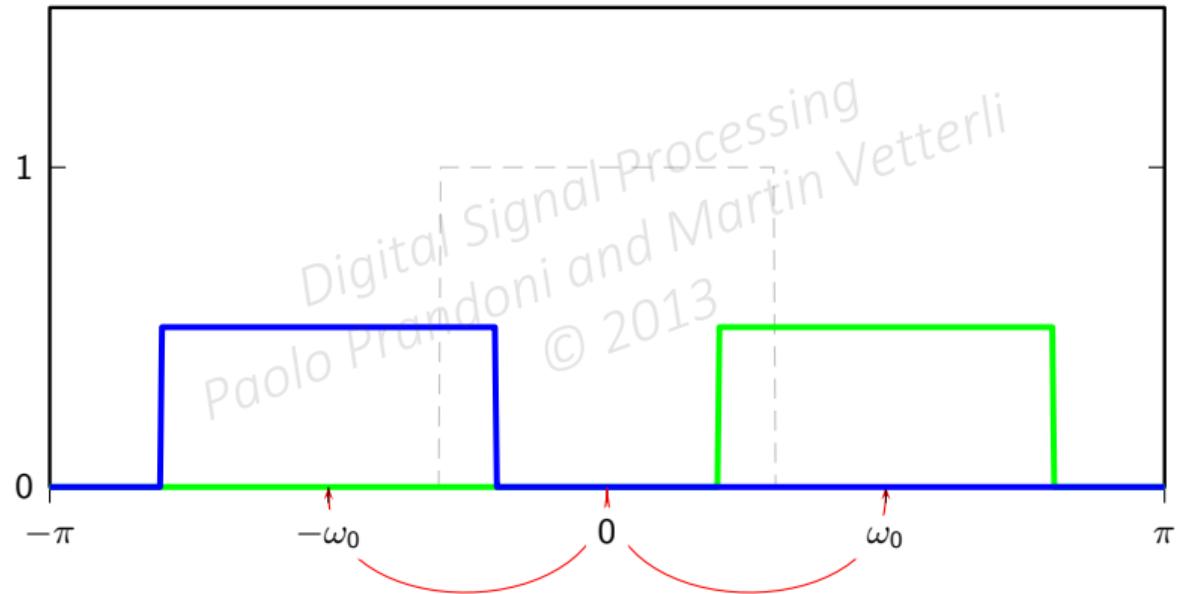
Ideal bandpass filter



Ideal bandpass filter



Ideal bandpass filter



$$H_{bp}(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega \pm \omega_0| \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (2\pi\text{-periodicity implicit})$$

Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

$$h_{bp}[n] = 2 \cos(\omega_0 n) \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right)$$

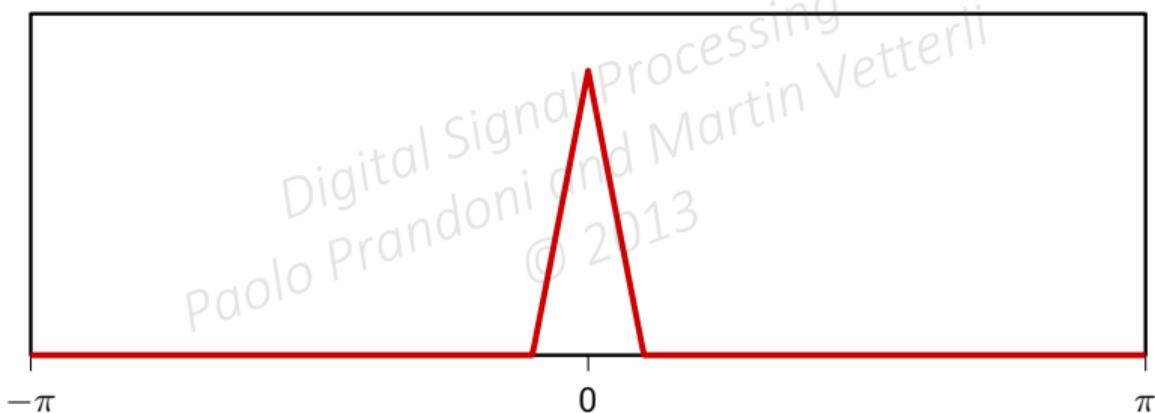
$$H_{bp}(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega \pm \omega_0| \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \quad (2\pi\text{-periodicity implicit})$$

$$h_{bp}[n] = 2 \cos(\omega_0 n) \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right)$$

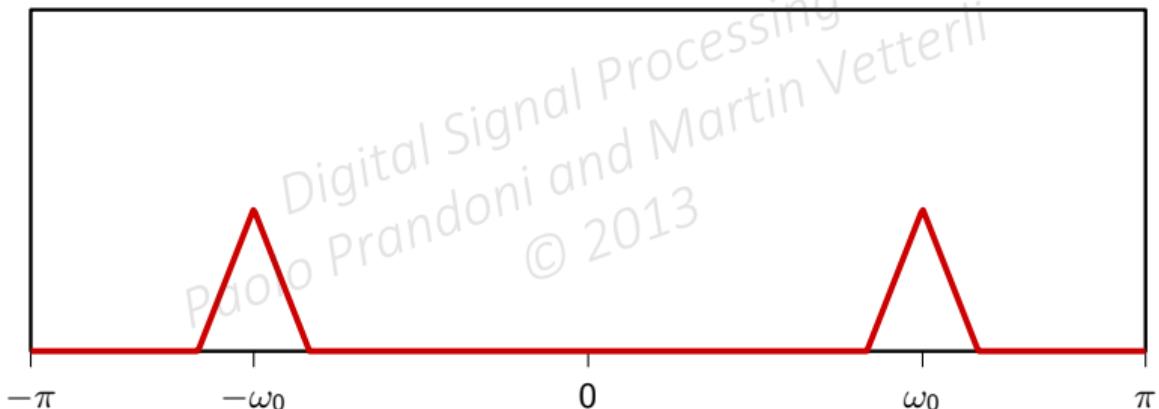
remember Module 4.8:

- ▶ apply sinusoidal modulation to $x[n]$: $y[n] = x[n] \cos \omega_0 n$
- ▶ demodulate by multiplying by the carrier $x'[n] = y[n] \cos \omega_0 n$
- ▶ demodulated signal contains unwanted high-frequency components

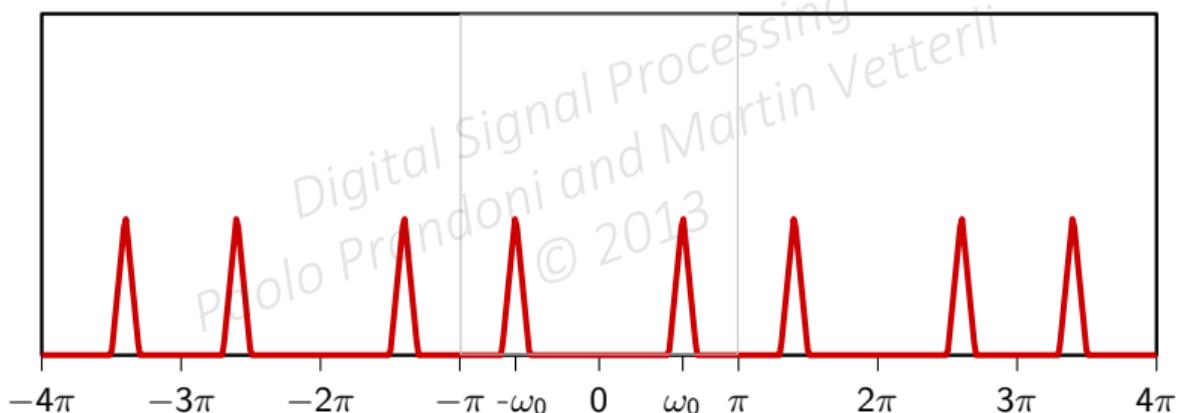
$$X(e^{j\omega})$$



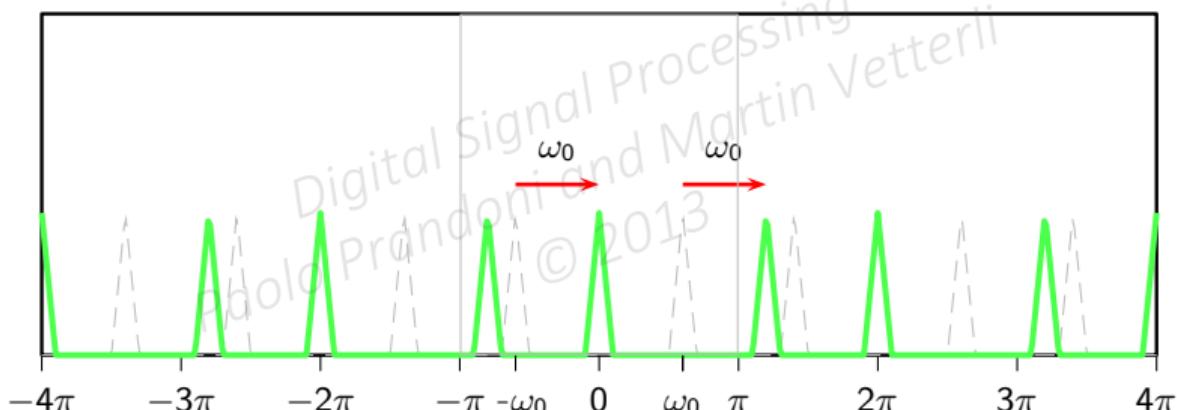
$$Y(e^{j\omega})$$



$$Y(e^{j\omega})$$

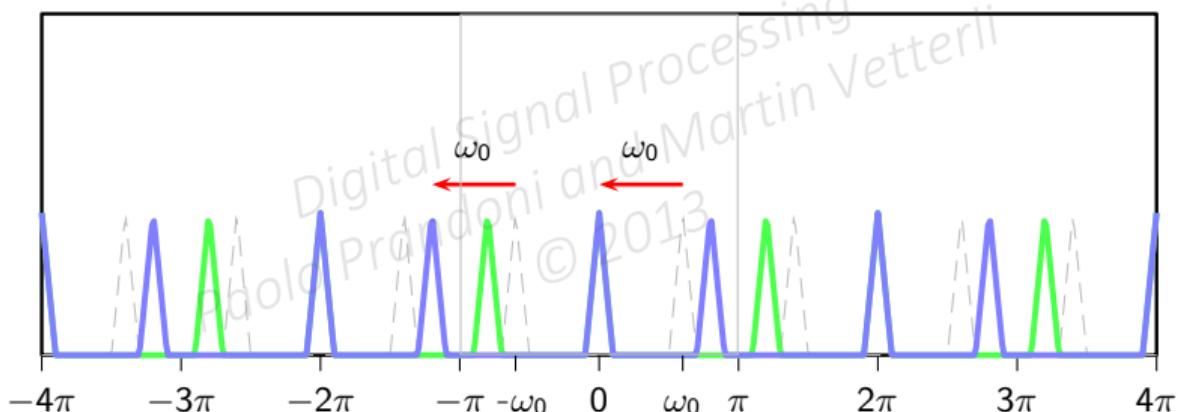


$$X'(e^{j\omega})$$

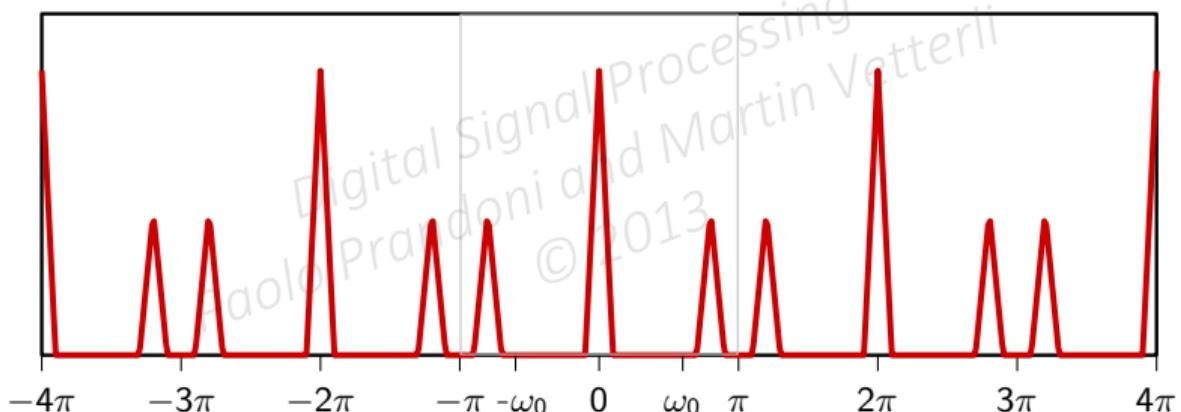


Digital Signal Processing
Abol肇 Prandoni and Martin Vetterli
© 2013

$$X'(e^{j\omega})$$



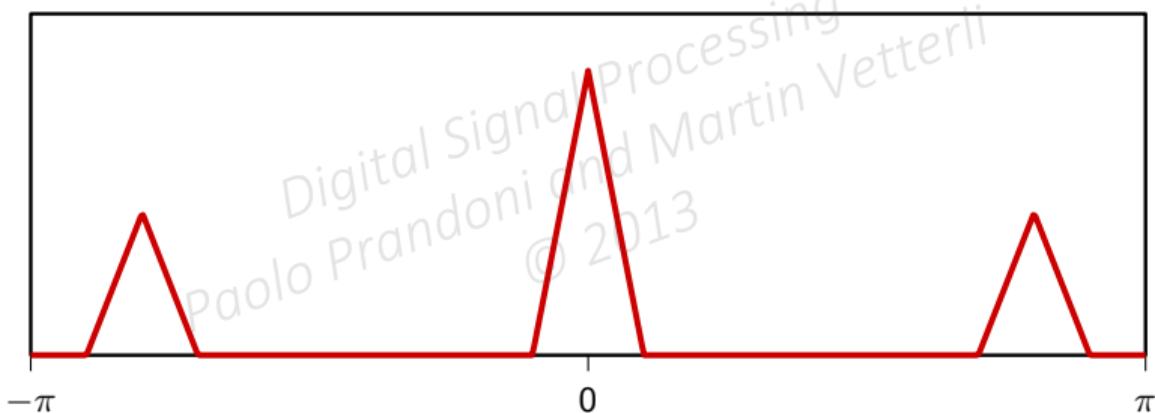
$$X'(e^{j\omega})$$



Digital Signal Processing
© 2013 Giacomo Prandoni and Martin Vetterli

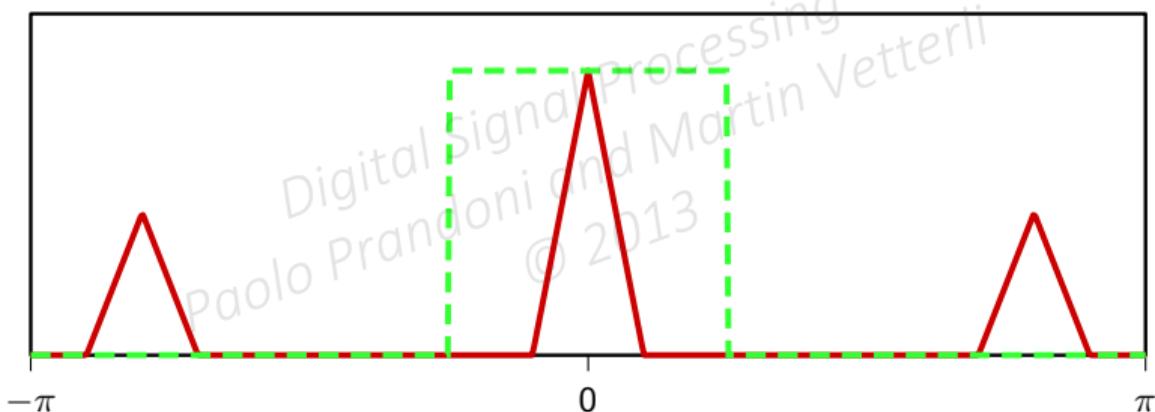
Solution: lowpass filtering

$$X'(e^{j\omega})$$



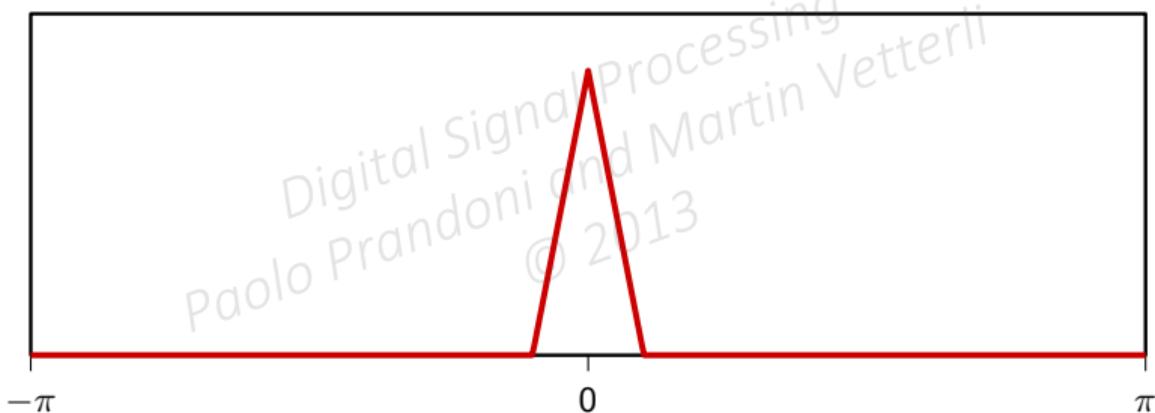
Solution: lowpass filtering

$$X'(e^{j\omega})$$



Solution: lowpass filtering

$$X(e^{j\omega})$$



END OF MODULE 5.5

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.6: Filter Design - Part I: Approximation of Ideal Filters

- ▶ Impulse truncation
- ▶ Window method
- ▶ Frequency sampling

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Impulse truncation
- ▶ Window method
- ▶ Frequency sampling

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Impulse truncation
- ▶ Window method
- ▶ Frequency sampling

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

How can we approximate an ideal lowpass?

Idea #1:

- ▶ pick ω_c
- ▶ compute ideal impulse response $h[n]$
- ▶ truncate $h[n]$ to a finite-support $\hat{h}[n]$
- ▶ $\hat{h}[n]$ defines an FIR filter

Digital Signal Processing
Palo Piantanida and Martin Vetterli
© 2013

How can we approximate an ideal lowpass?

Idea #1:

- ▶ pick ω_c
- ▶ compute ideal impulse response $h[n]$
- ▶ truncate $h[n]$ to a finite-support $\hat{h}[n]$
- ▶ $\hat{h}[n]$ defines an FIR filter

Digital Signal Processing
Palo Piantanida and Martin Vetterli
© 2013

How can we approximate an ideal lowpass?

Idea #1:

- ▶ pick ω_c
- ▶ compute ideal impulse response $h[n]$
- ▶ truncate $h[n]$ to a finite-support $\hat{h}[n]$
- ▶ $\hat{h}[n]$ defines an FIR filter

Digital Signal Processing
Paolo Pogliani and Martin Vetterli
© 2013

How can we approximate an ideal lowpass?

Idea #1:

- ▶ pick ω_c
- ▶ compute ideal impulse response $h[n]$
- ▶ truncate $h[n]$ to a finite-support $\hat{h}[n]$
- ▶ $\hat{h}[n]$ defines an FIR filter

Digital Signal Processing
Sergio Salvi and Martin Vetterli
© 2013

FIR approximation of length $M = 2N + 1$:

$$\hat{h}[n] = \begin{cases} \frac{\omega_c}{\pi} \operatorname{sinc}\left(\frac{\omega_c}{\pi} n\right) & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

Why it could be a good idea

$$\begin{aligned} \text{MSE} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= \|H(e^{j\omega}) - \hat{H}(e^{j\omega})\|_2^2 \\ &\stackrel{\text{Digital Signal Processing}}{=} \|h[n] - \hat{h}[n]\|^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2 \end{aligned}$$

MSE is minimized by symmetric impulse truncation around zero

Why it could be a good idea

$$\begin{aligned}\text{MSE} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= \|H(e^{j\omega}) - \hat{H}(e^{j\omega})\|^2 \\ &\stackrel{\text{Digital Signal Processing in Vetterli}}{=} \|h[n] - \hat{h}[n]\|^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2\end{aligned}$$

MSE is minimized by symmetric impulse truncation around zero

$$\begin{aligned}\text{MSE} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= \|H(e^{j\omega}) - \hat{H}(e^{j\omega})\|^2 \\ &= \|h[n] - \hat{h}[n]\|^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2\end{aligned}$$

MSE is minimized by symmetric impulse truncation around zero

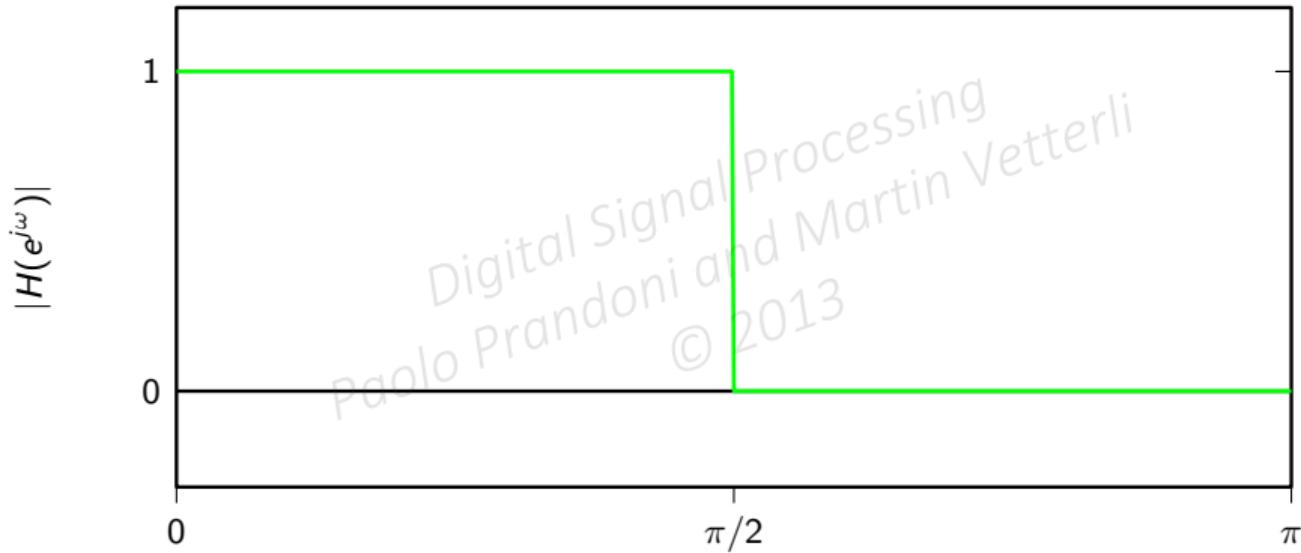
$$\begin{aligned}\text{MSE} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= \|H(e^{j\omega}) - \hat{H}(e^{j\omega})\|^2 \\ &= \|h[n] - \hat{h}[n]\|^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2\end{aligned}$$

MSE is minimized by symmetric impulse truncation around zero

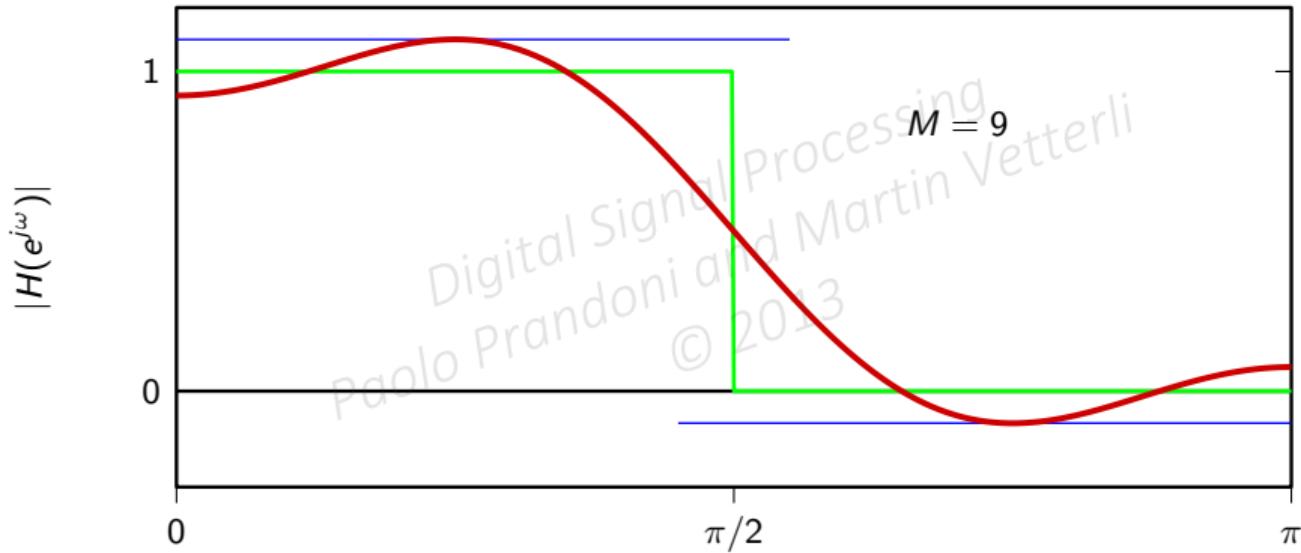
$$\begin{aligned}\text{MSE} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= \|H(e^{j\omega}) - \hat{H}(e^{j\omega})\|^2 \\ &= \|h[n] - \hat{h}[n]\|^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2\end{aligned}$$

MSE is minimized by symmetric impulse truncation around zero

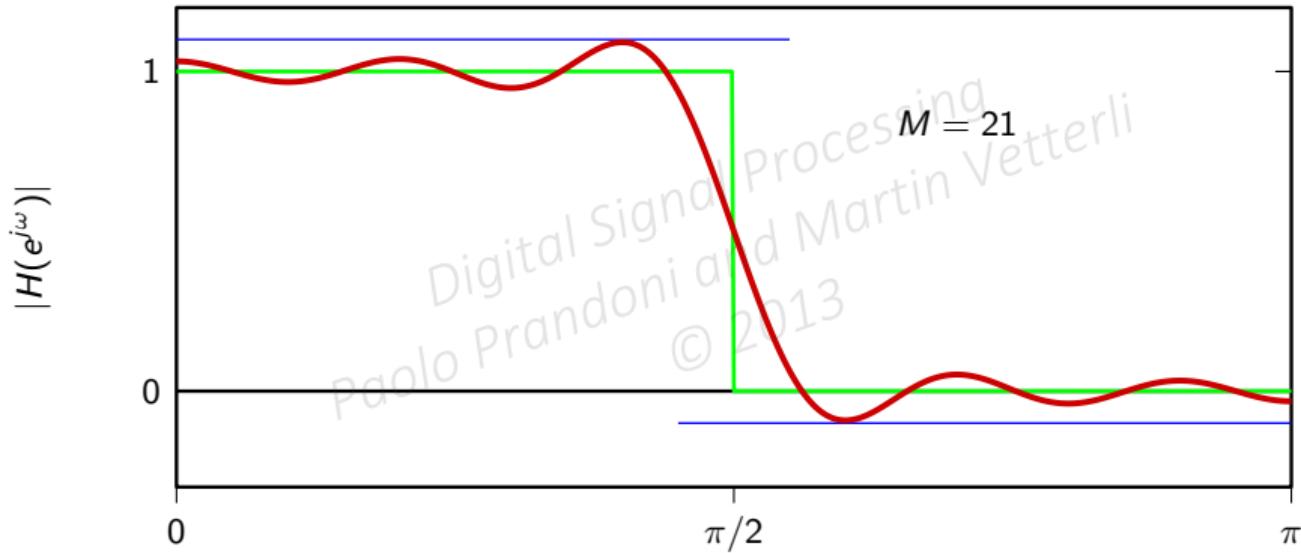
Why it's not such a good idea



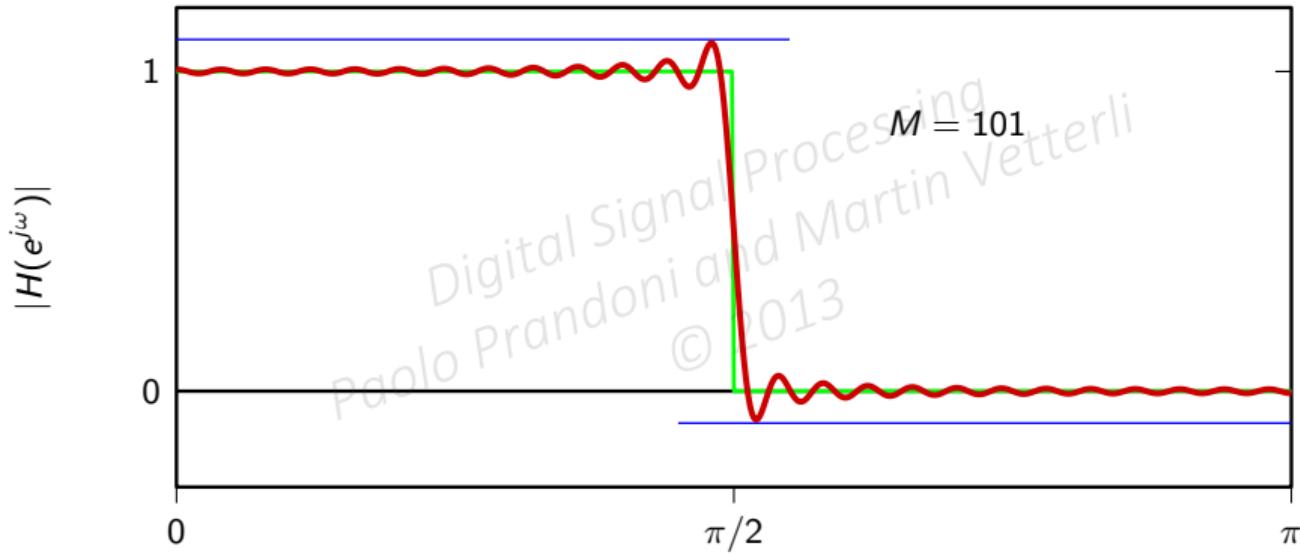
Why it's not such a good idea



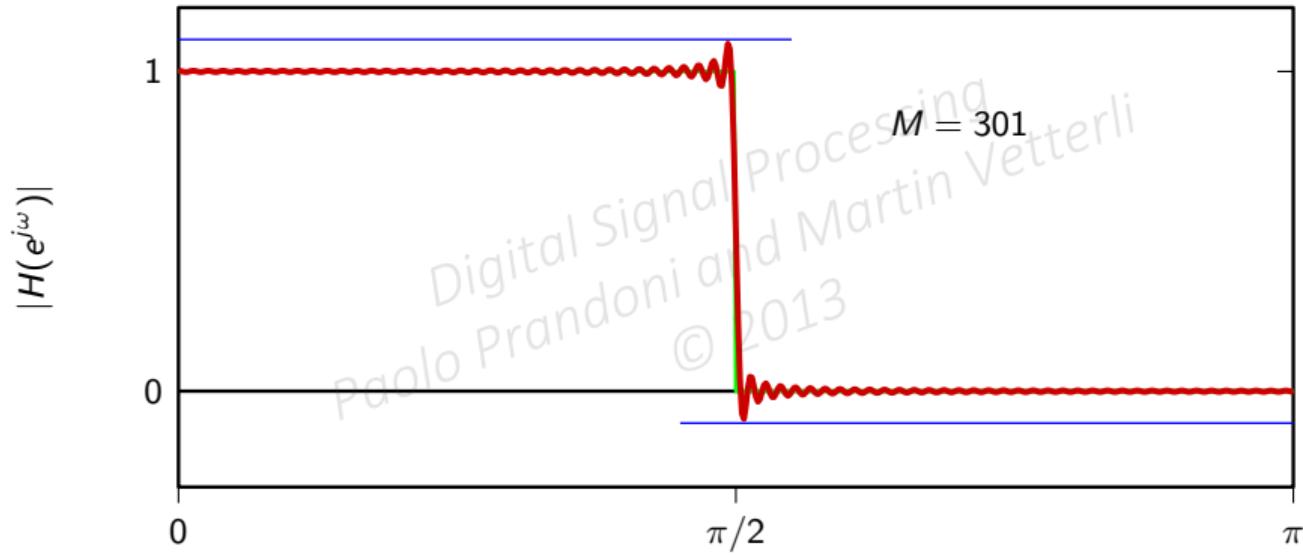
Why it's not such a good idea



Why it's not such a good idea



Why it's not such a good idea



The maximum error around the cutoff frequency
is around 9% of the height of the jump
regardless of N

$$\hat{h}[n] = h[n] w[n]$$

$$w[n] = \begin{cases} 1 & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{H}(e^{j\omega}) = ?$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$\hat{h}[n] = h[n] w[n]$$

$$w[n] = \begin{cases} 1 & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{H}(e^{j\omega}) = ?$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$\text{DTFT}\{(x * y)[n]\} = X(e^{j\omega}) Y(e^{j\omega})$$

Digital Signal Processing
DTFT $\{x[n]y[n]\} = X(e^{j\omega})Y(e^{j\omega})$

© 2018

$$\text{DTFT}\{(x * y)[n]\} = X(e^{j\omega}) Y(e^{j\omega})$$

$$\text{DTFT}\{x[n]y[n]\} = (X * Y)(e^{j\omega})$$

in \mathbb{C}^∞ :

$$(x * y)[n] = \langle x^*[k], y[n - k] \rangle$$

$$= \sum_{k=-\infty}^{\infty} x[k]y[n - k]$$

$$(X * Y)(e^{j\omega}) = \langle X^*(e^{j\sigma}), Y(e^{j(\omega-\sigma)}) \rangle$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) d\sigma$$

in \mathbb{C}^∞ :

$$(x * y)[n] = \langle x^*[k], y[n - k] \rangle$$

$$= \sum_{k=-\infty}^{\infty} x[k]y[n - k]$$

in $L_2([-\pi, \pi])$:

$$(X * Y)(e^{j\omega}) = \langle X^*(e^{j\sigma}), Y(e^{j(\omega-\sigma)}) \rangle$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) d\sigma$$

Modulation theorem: proof

$$\begin{aligned}\text{IDTFT} \{(X * Y)(e^{j\omega})\} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} (X * Y)(e^{j\omega}) e^{j\omega n} d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\omega n} d\sigma d\omega \\&\stackrel{\text{Digital Signal Processing}}{=} \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\sigma n} e^{j(\omega-\sigma)n} d\sigma d\omega \\&= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) e^{j\sigma n} d\sigma \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(e^{j(\omega-\sigma)}) e^{j(\omega-\sigma)n} d\omega \\&= x[n] y[n]\end{aligned}$$

Modulation theorem: proof

$$\begin{aligned}\text{IDTFT} \{(X * Y)(e^{j\omega})\} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} (X * Y)(e^{j\omega}) e^{j\omega n} d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\omega n} d\sigma d\omega \\&\stackrel{\text{Paolo P. (2011) Digital Signal Processing and Martin Vetterli}}{=} \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\sigma n} e^{j(\omega-\sigma)n} d\sigma d\omega \\&= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) e^{j\sigma n} d\sigma \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(e^{j(\omega-\sigma)}) e^{j(\omega-\sigma)n} d\omega \\&= x[n] y[n]\end{aligned}$$

Modulation theorem: proof

$$\begin{aligned}\text{IDTFT} \{(X * Y)(e^{j\omega})\} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} (X * Y)(e^{j\omega}) e^{j\omega n} d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\omega n} d\sigma d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\sigma n} e^{j(\omega-\sigma)n} d\sigma d\omega \\&= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) e^{j\sigma n} d\sigma \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(e^{j(\omega-\sigma)}) e^{j(\omega-\sigma)n} d\omega \\&= x[n] y[n]\end{aligned}$$

Modulation theorem: proof

$$\begin{aligned}\text{IDTFT} \{(X * Y)(e^{j\omega})\} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} (X * Y)(e^{j\omega}) e^{j\omega n} d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\omega n} d\sigma d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\sigma n} e^{j(\omega-\sigma)n} d\sigma d\omega \\&= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) e^{j\sigma n} d\sigma \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(e^{j(\omega-\sigma)}) e^{j(\omega-\sigma)n} d\omega \\&= x[n] y[n]\end{aligned}$$

$$\begin{aligned}\text{IDTFT} \{(X * Y)(e^{j\omega})\} &= \frac{1}{2\pi} \int_{-\pi}^{\pi} (X * Y)(e^{j\omega}) e^{j\omega n} d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\omega n} d\sigma d\omega \\&= \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) Y(e^{j(\omega-\sigma)}) e^{j\sigma n} e^{j(\omega-\sigma)n} d\sigma d\omega \\&= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) e^{j\sigma n} d\sigma \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(e^{j(\omega-\sigma)}) e^{j(\omega-\sigma)n} d\omega \\&= x[n] y[n]\end{aligned}$$

$$\begin{aligned}\text{DTFT } \{x[n] \cos \omega_c n\} &= X(e^{j\omega}) * \frac{1}{2} [\tilde{\delta}(\omega - \omega_c) + \tilde{\delta}(\omega + \omega_c)] \\ &= \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega + \omega_c) d\sigma + \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega - \omega_c) d\sigma \\ &= \frac{1}{2} [X(e^{j(\omega - \omega_c)}) + X(e^{j(\omega + \omega_c)})]\end{aligned}$$

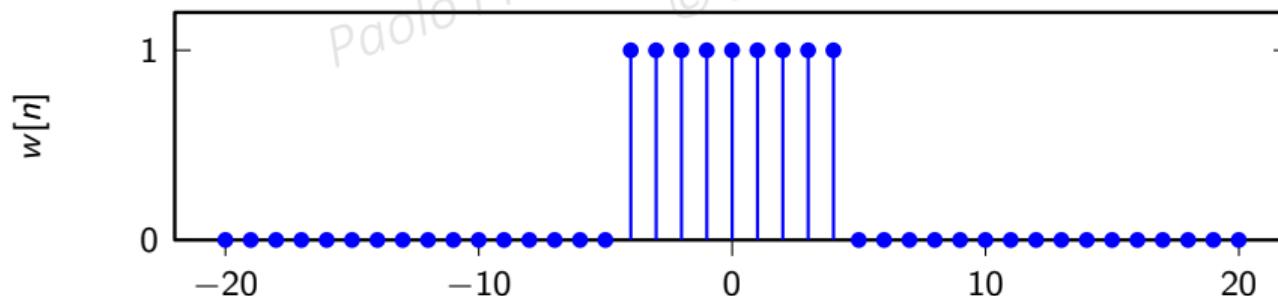
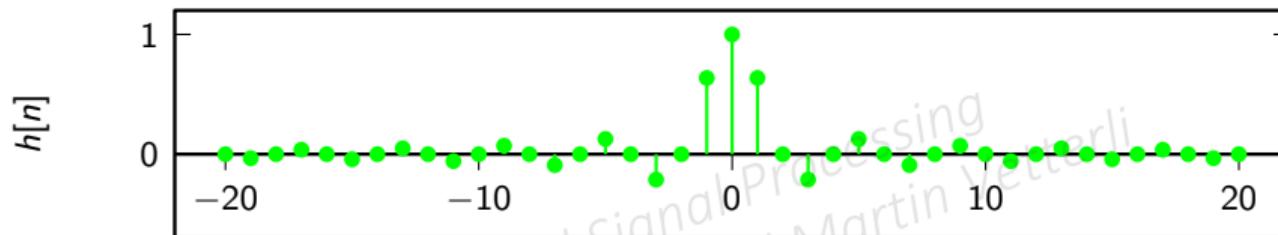
Digital Signal Processing
Digital Random and Martin Vetterli
© 2013

$$\begin{aligned}\text{DTFT } \{x[n] \cos \omega_c n\} &= X(e^{j\omega}) * \frac{1}{2} [\tilde{\delta}(\omega - \omega_c) + \tilde{\delta}(\omega + \omega_c)] \\ &= \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega + \omega_c) d\sigma + \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega - \omega_c) d\sigma \\ &= \frac{1}{2} [X(e^{j(\omega - \omega_c)}) + X(e^{j(\omega + \omega_c)})]\end{aligned}$$

$$\begin{aligned}\text{DTFT } \{x[n] \cos \omega_c n\} &= X(e^{j\omega}) * \frac{1}{2} [\tilde{\delta}(\omega - \omega_c) + \tilde{\delta}(\omega + \omega_c)] \\ &= \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega + \omega_c) d\sigma + \frac{1}{4\pi} \int_{-\pi}^{\pi} X(e^{j\sigma}) \tilde{\delta}(\sigma - \omega - \omega_c) d\sigma \\ &= \frac{1}{2} [X(e^{j(\omega - \omega_c)}) + X(e^{j(\omega + \omega_c)})]\end{aligned}$$

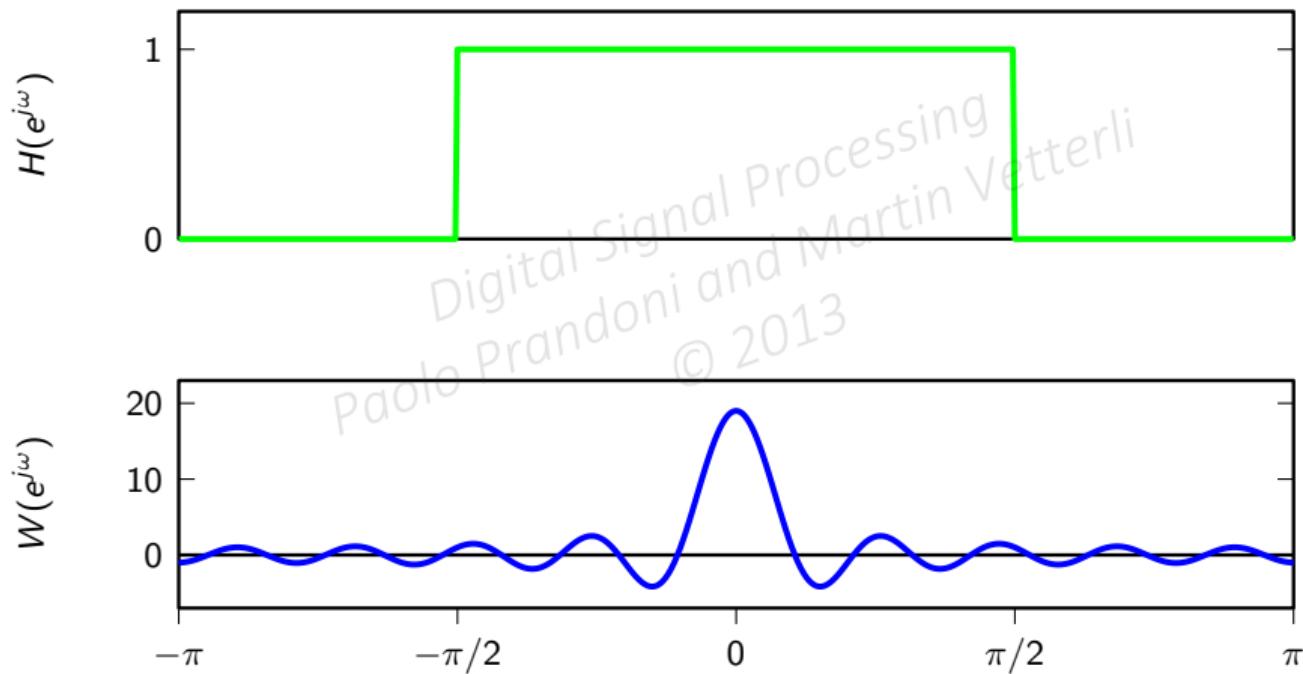
Understanding the Gibbs phenomenon

$$\hat{h}[n] = h[n] w[n]$$

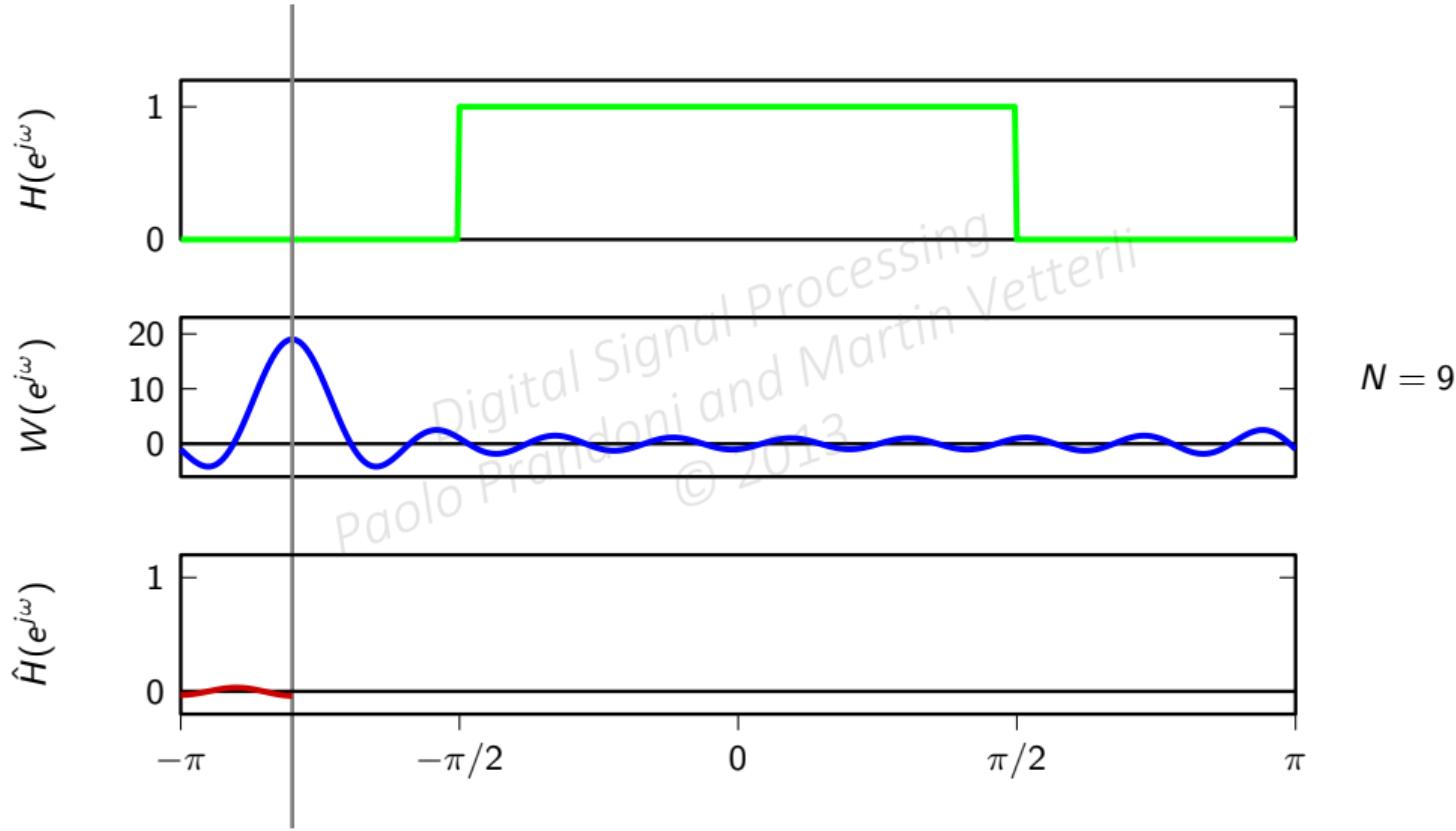


Understanding the Gibbs phenomenon

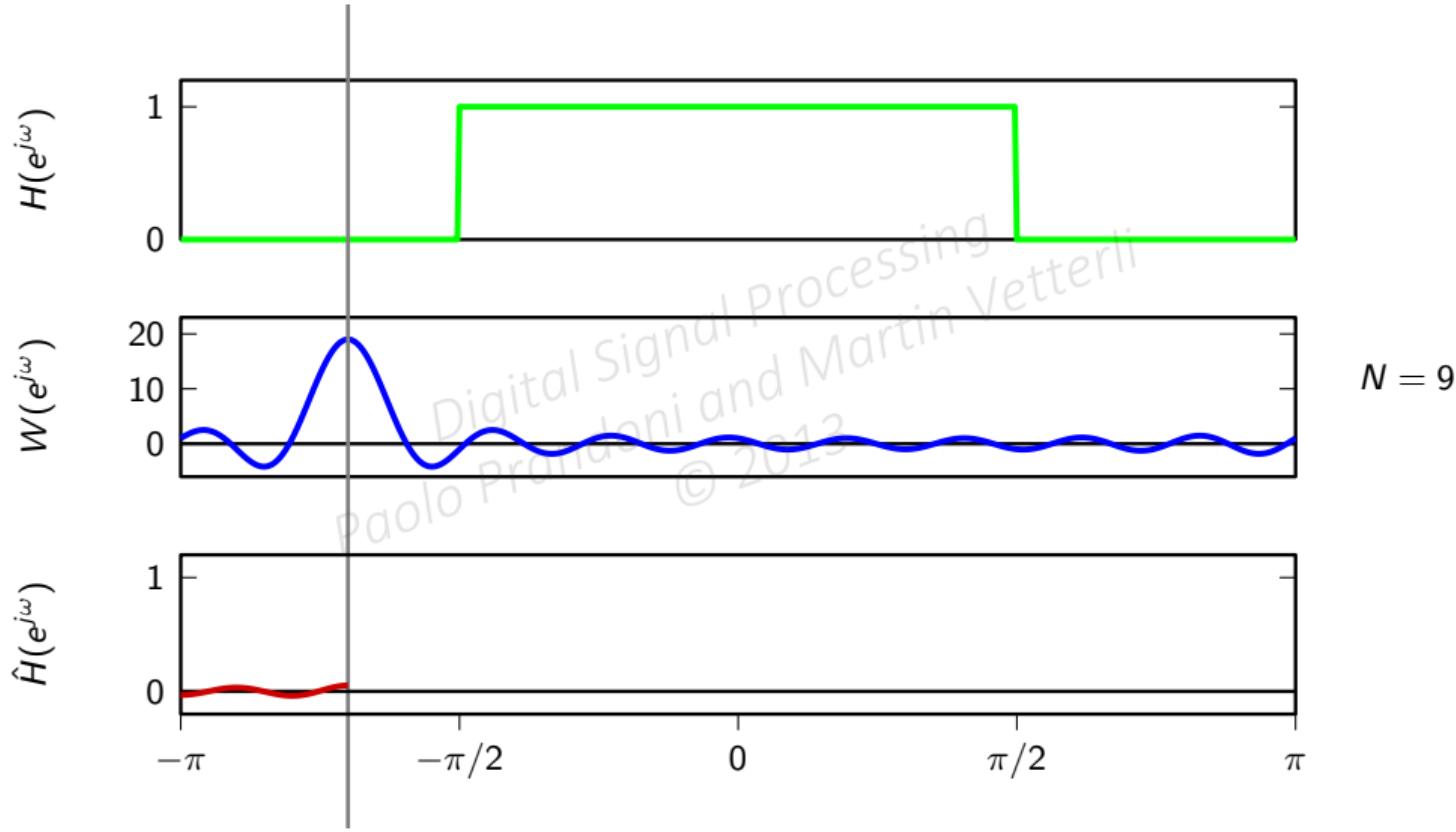
$$\hat{H}(e^{j\omega}) = (H * W)(e^{j\omega}), \quad W(e^{j\omega}) = \sin(\omega(2N+1)/2) / \sin(\omega/2) \quad (\text{Module 4.7})$$



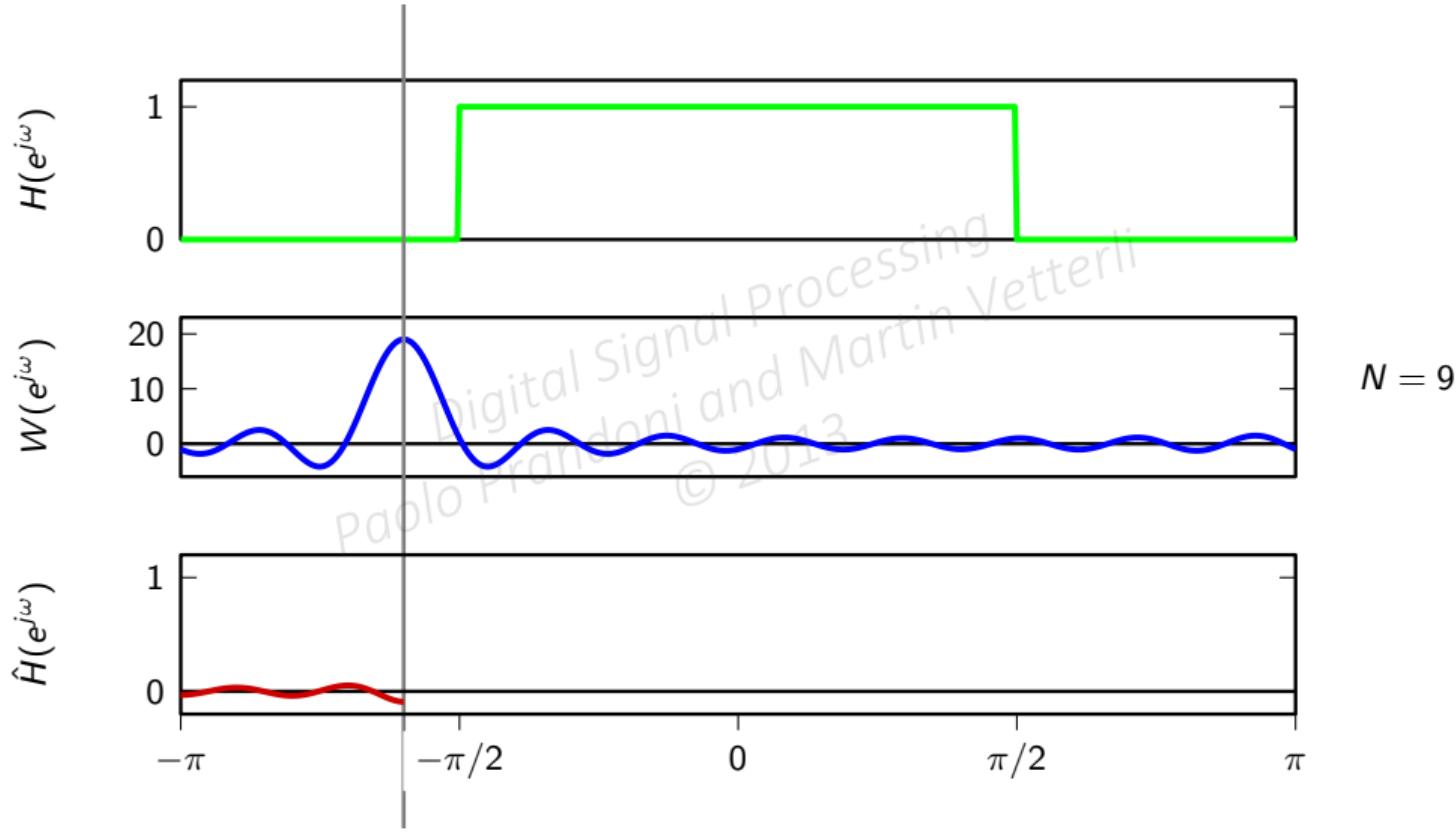
Implicit frequency-domain convolution



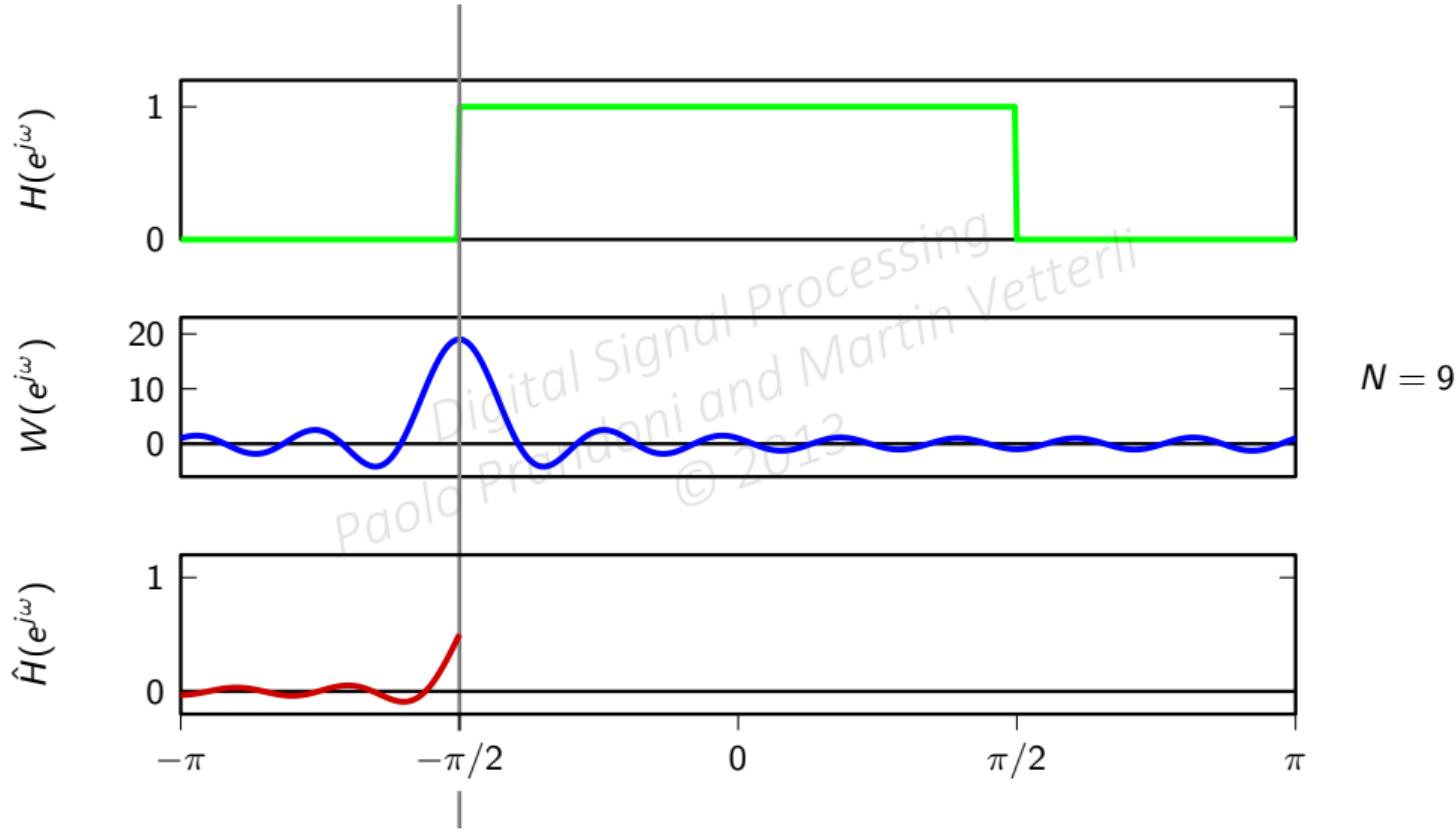
Implicit frequency-domain convolution



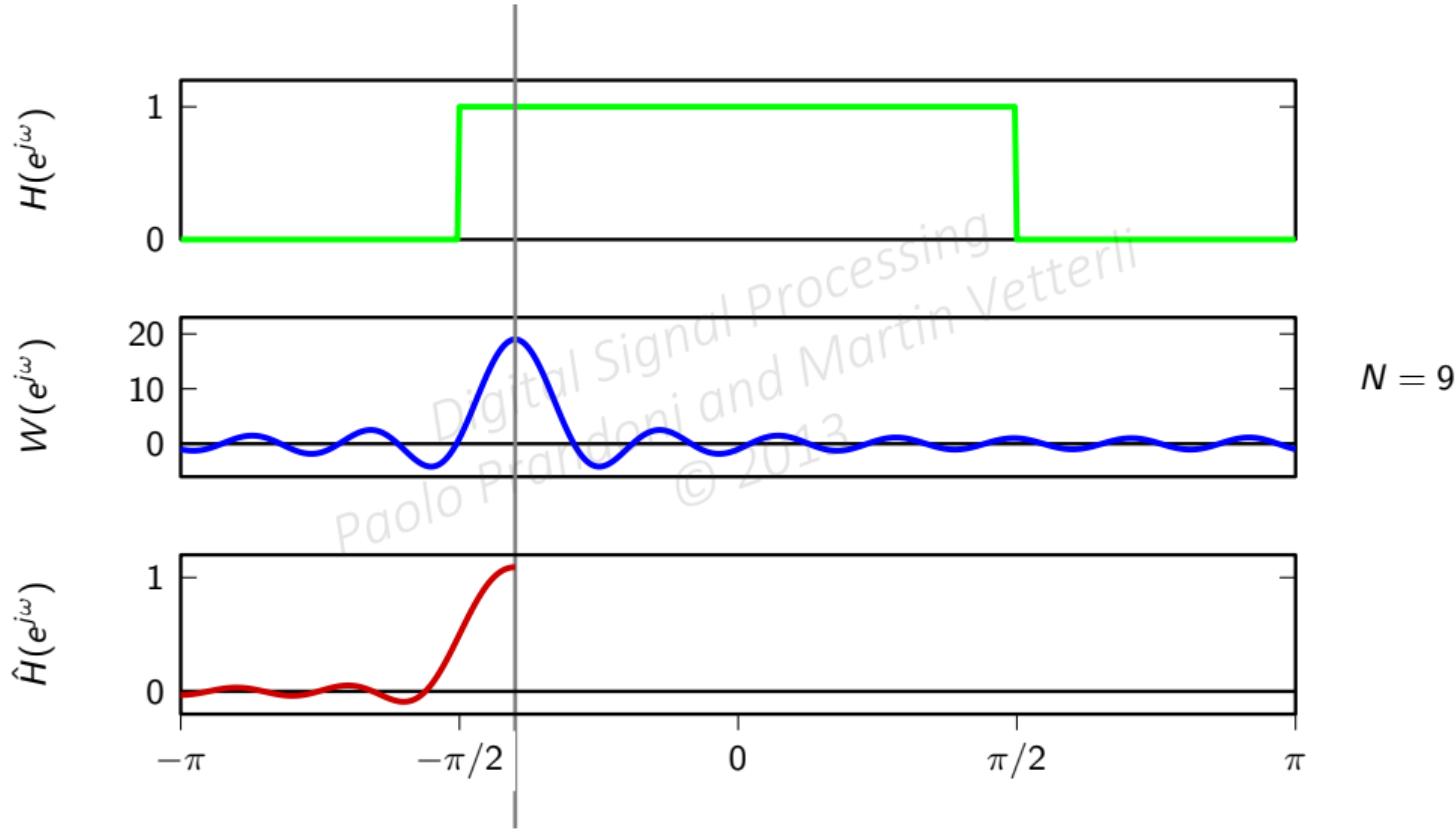
Implicit frequency-domain convolution



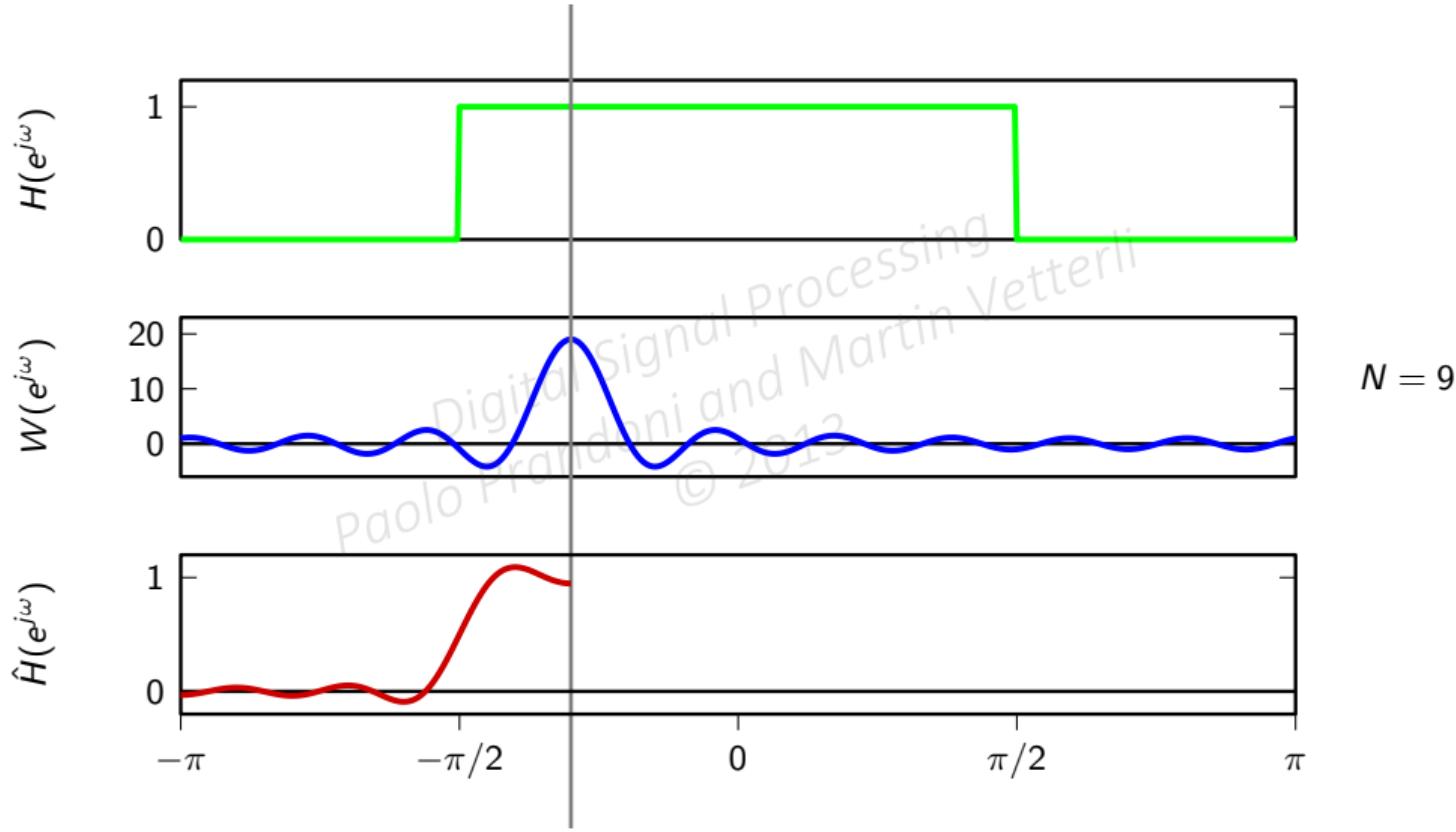
Implicit frequency-domain convolution



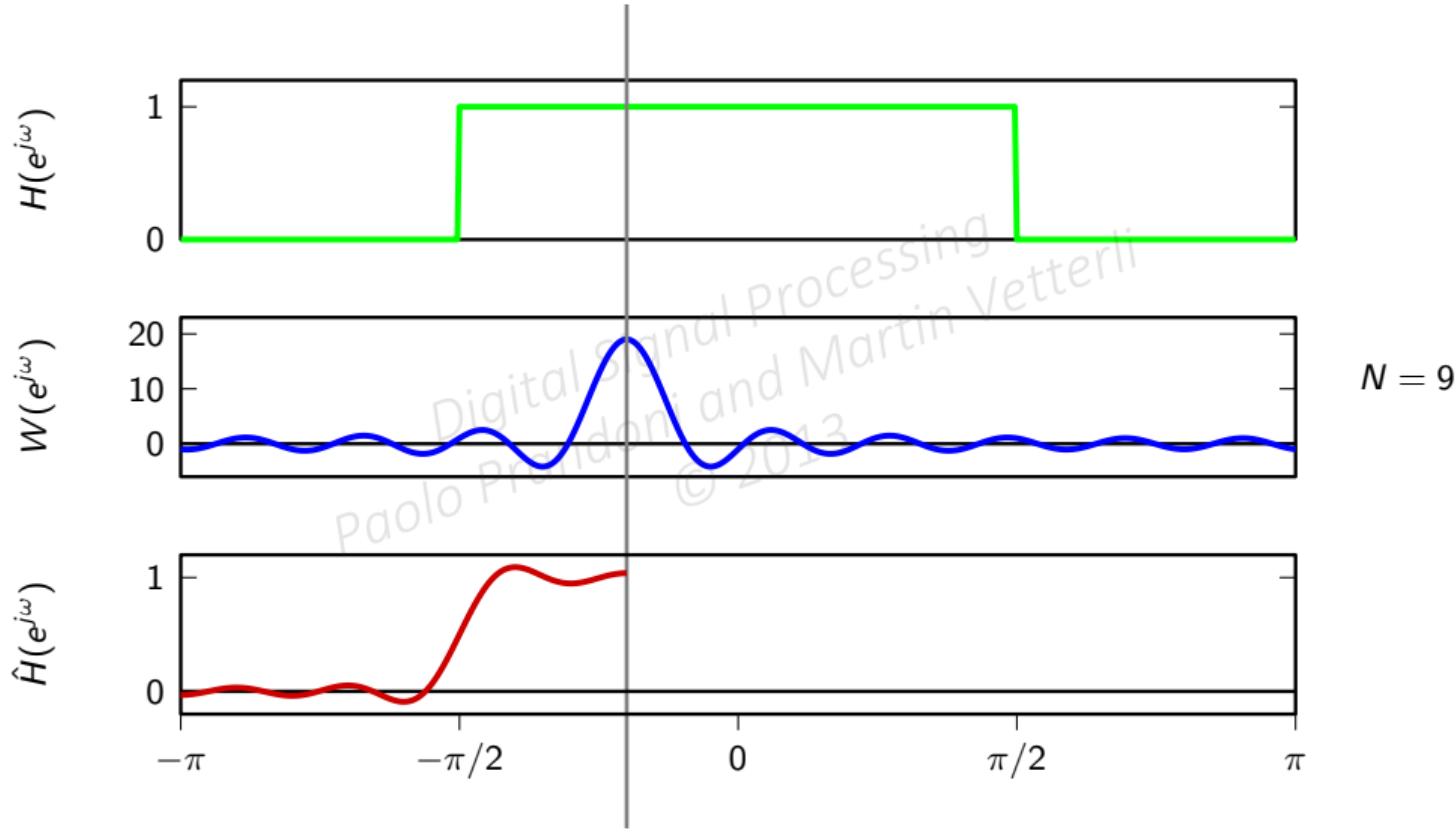
Implicit frequency-domain convolution



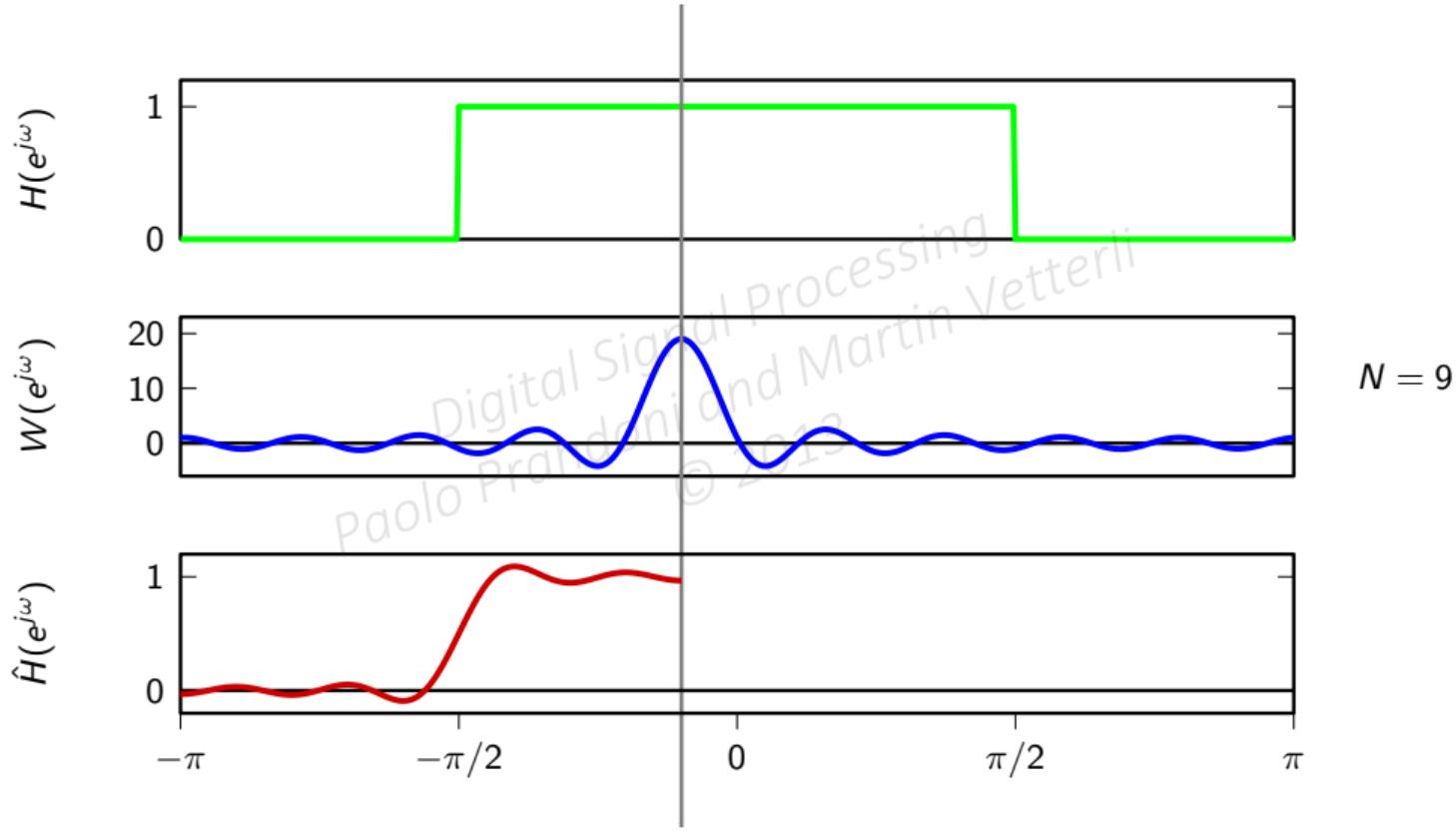
Implicit frequency-domain convolution



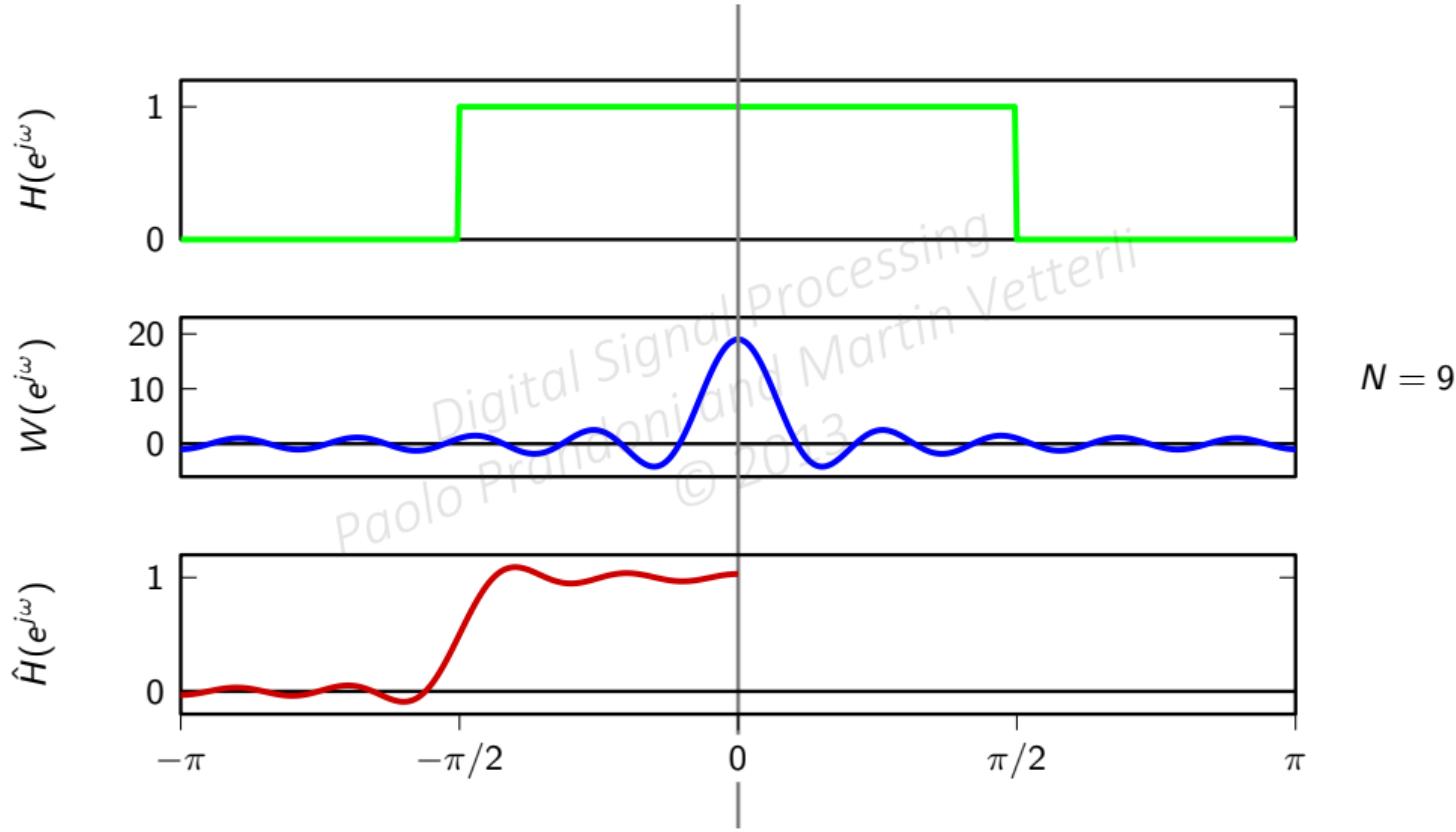
Implicit frequency-domain convolution



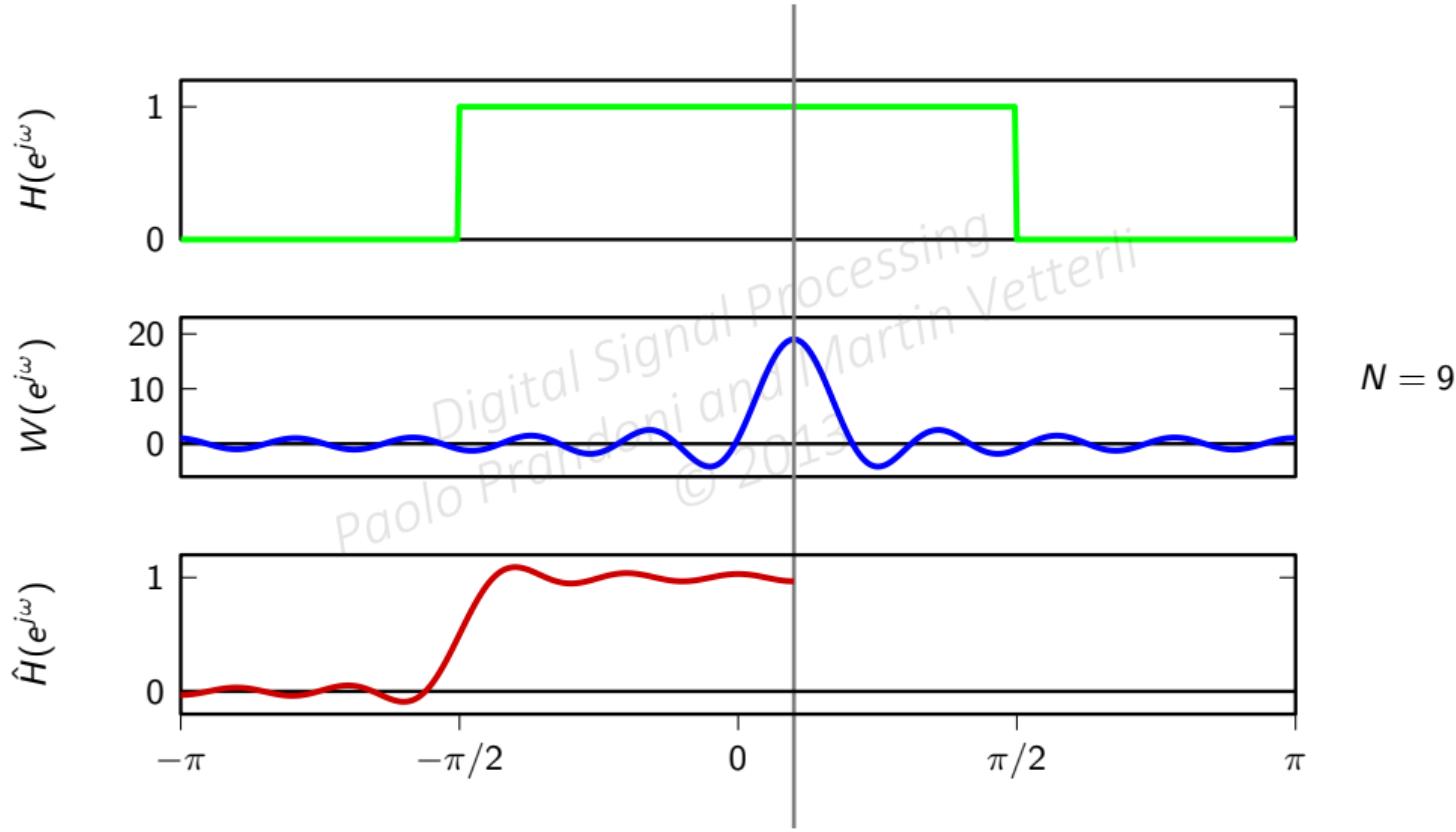
Implicit frequency-domain convolution



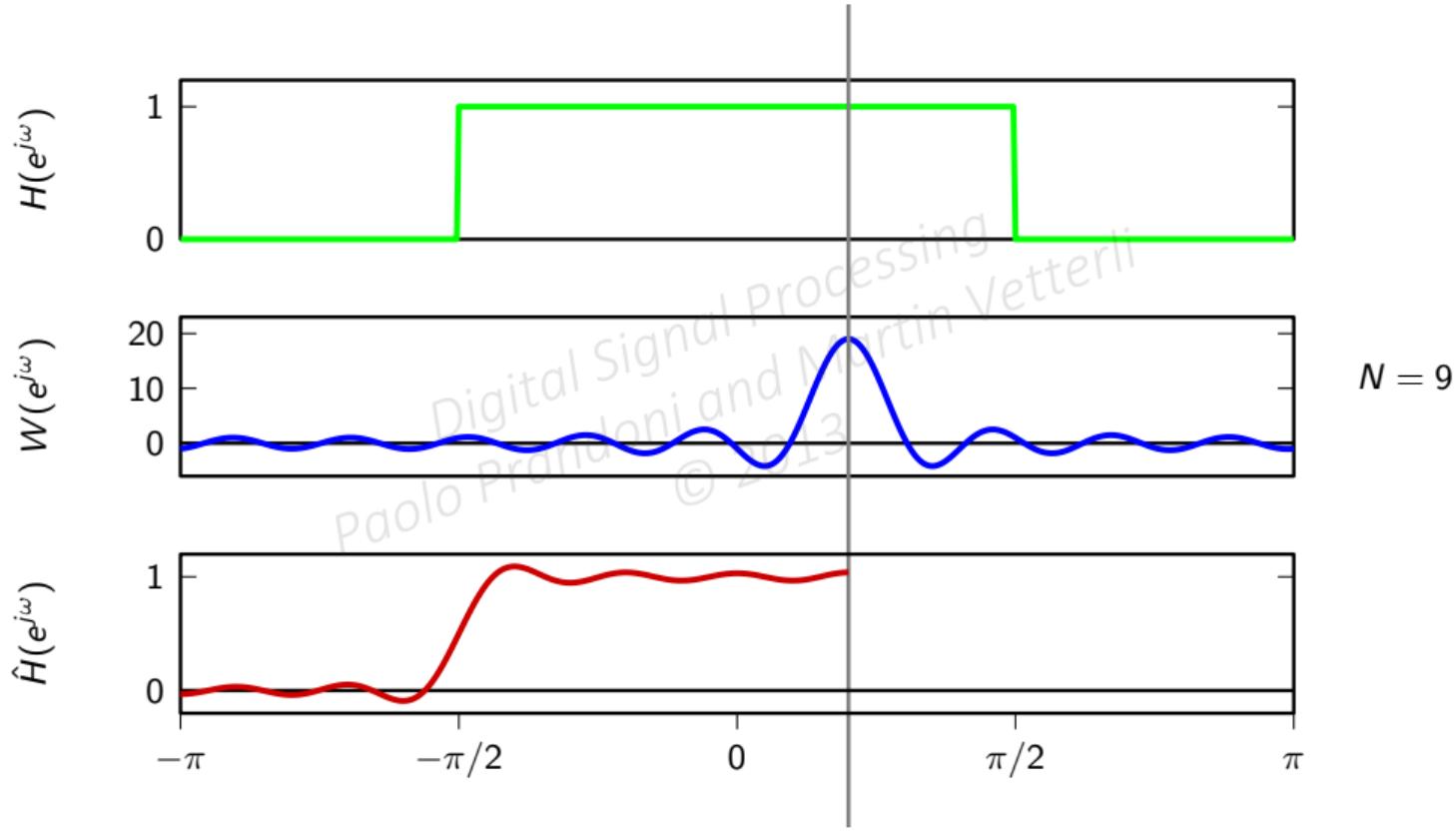
Implicit frequency-domain convolution



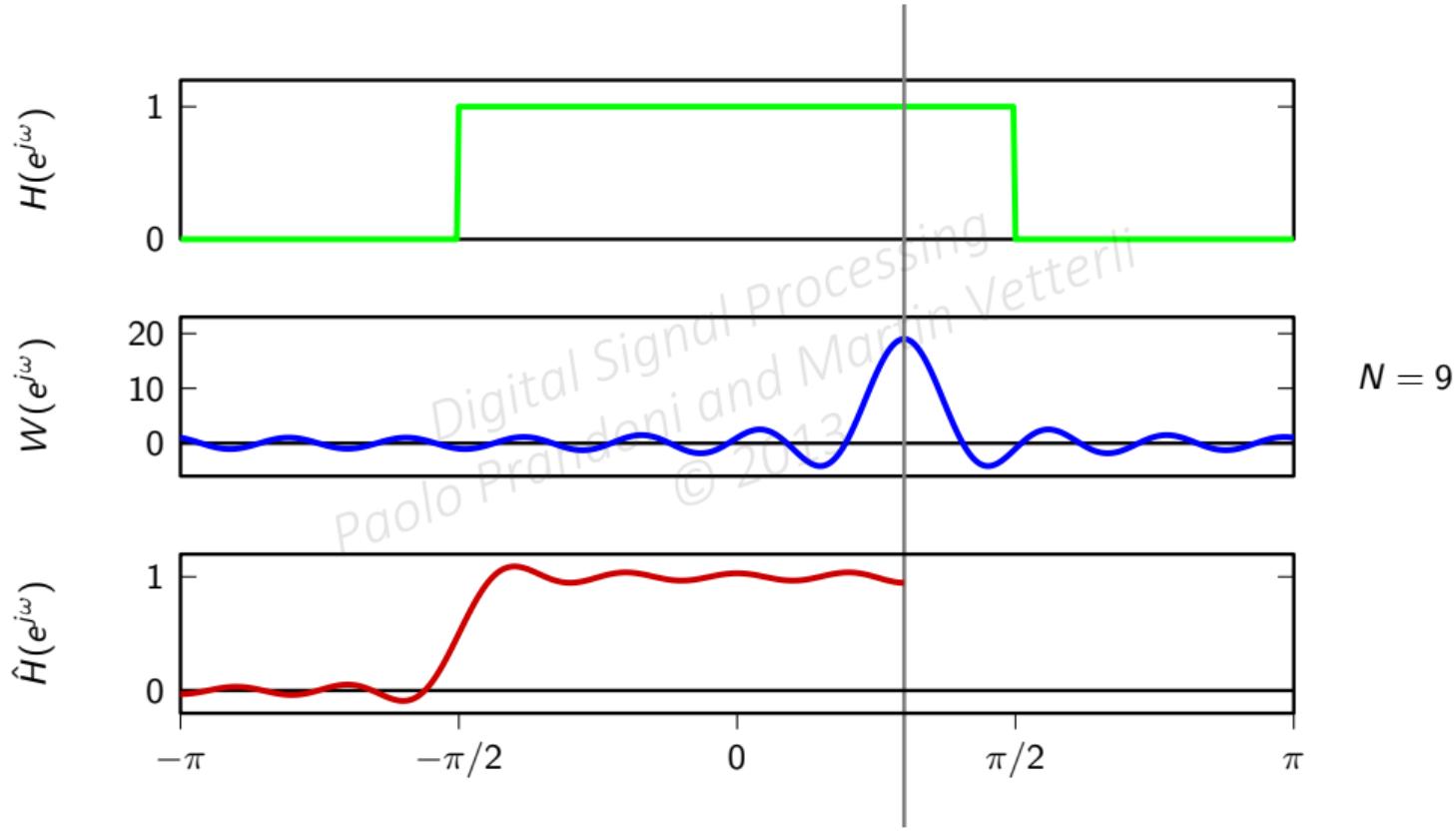
Implicit frequency-domain convolution



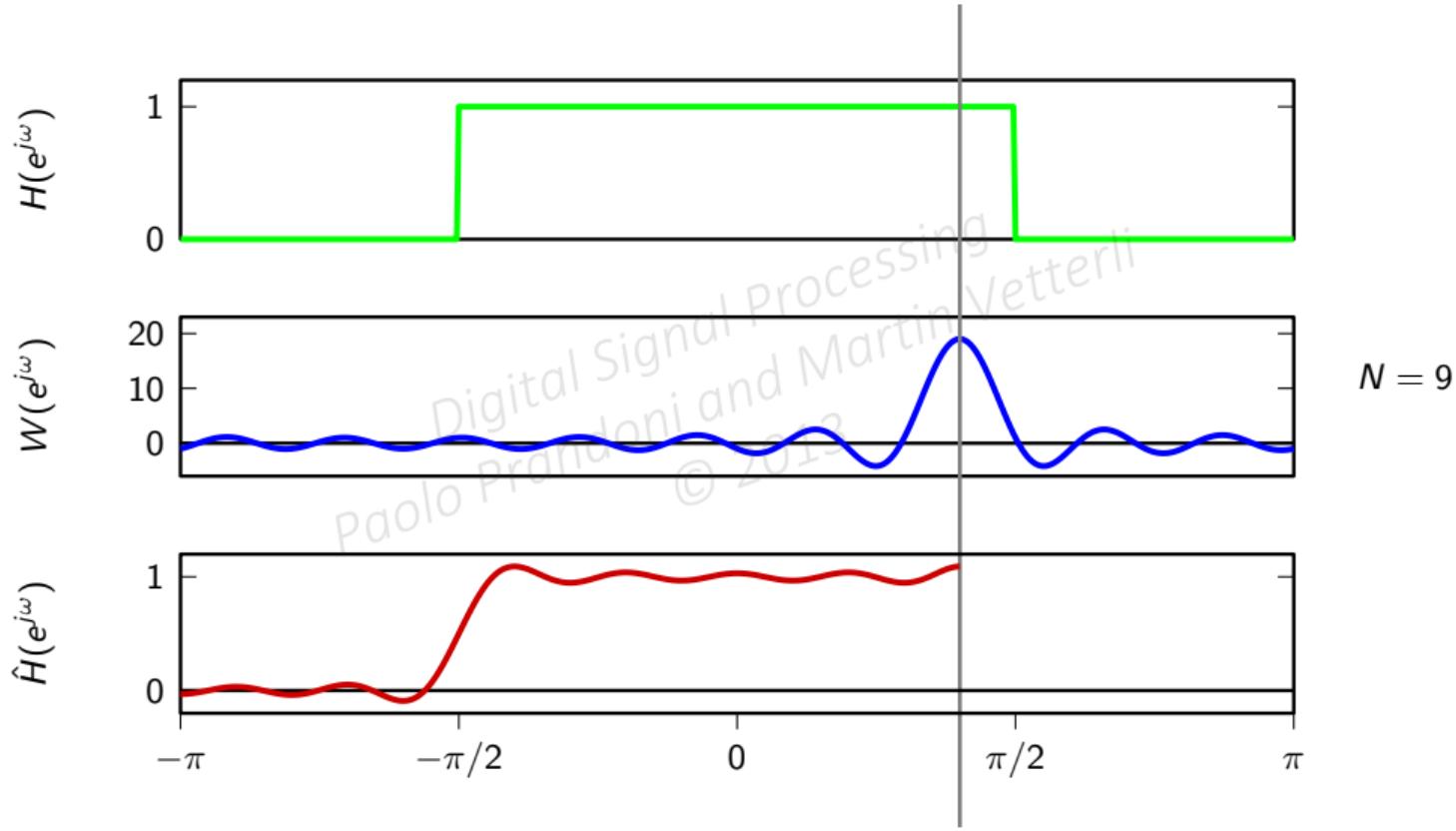
Implicit frequency-domain convolution



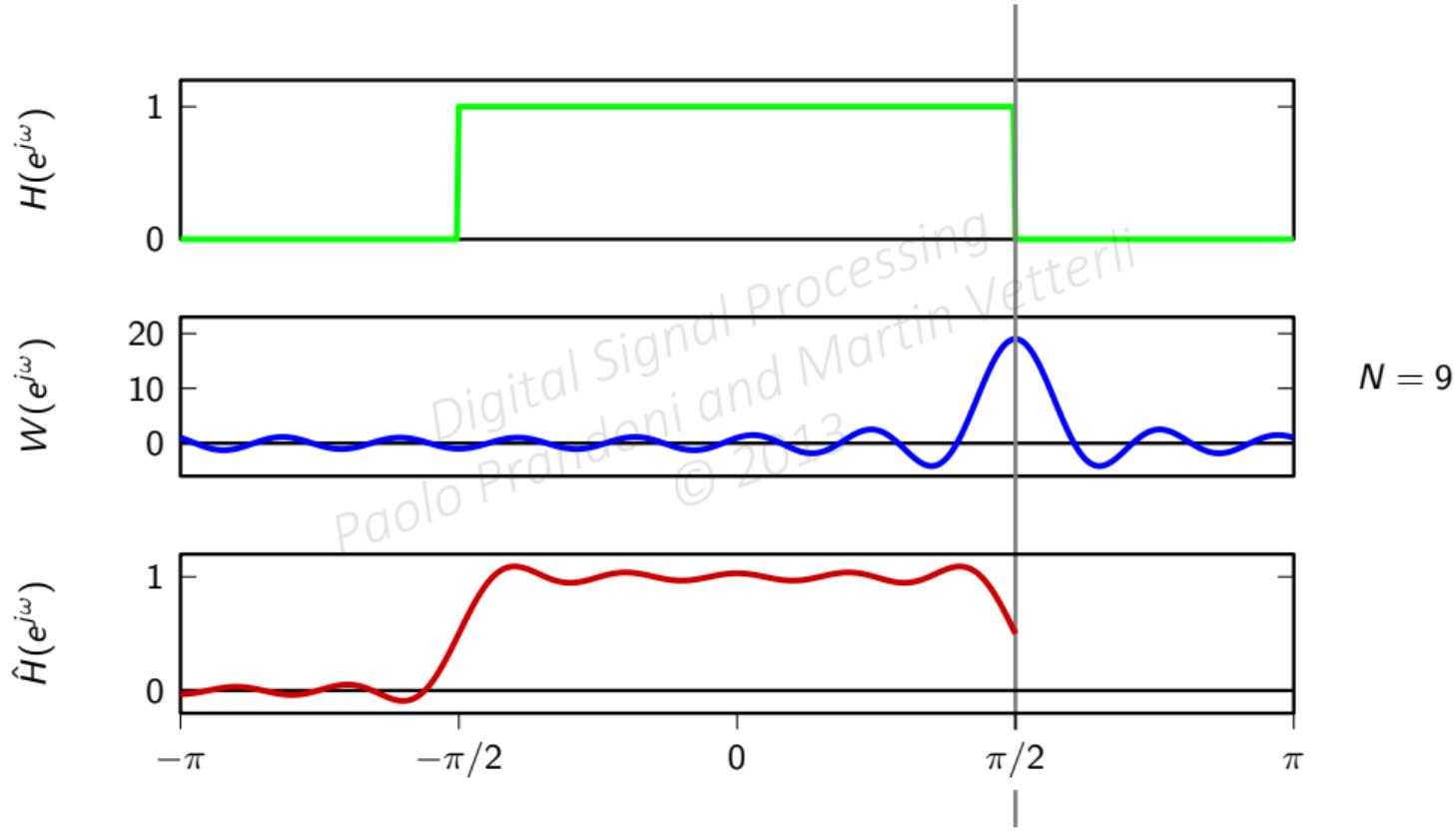
Implicit frequency-domain convolution



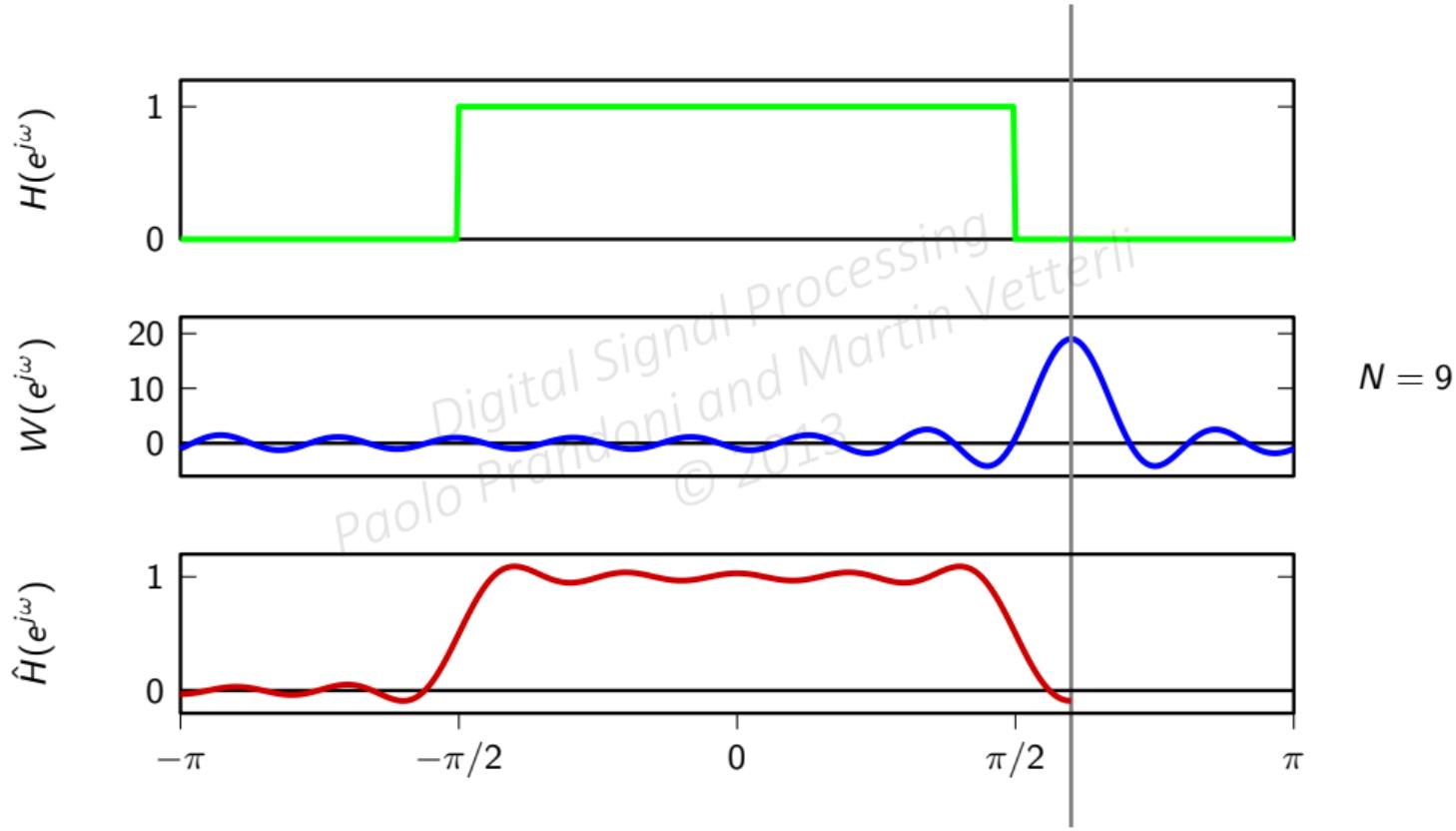
Implicit frequency-domain convolution



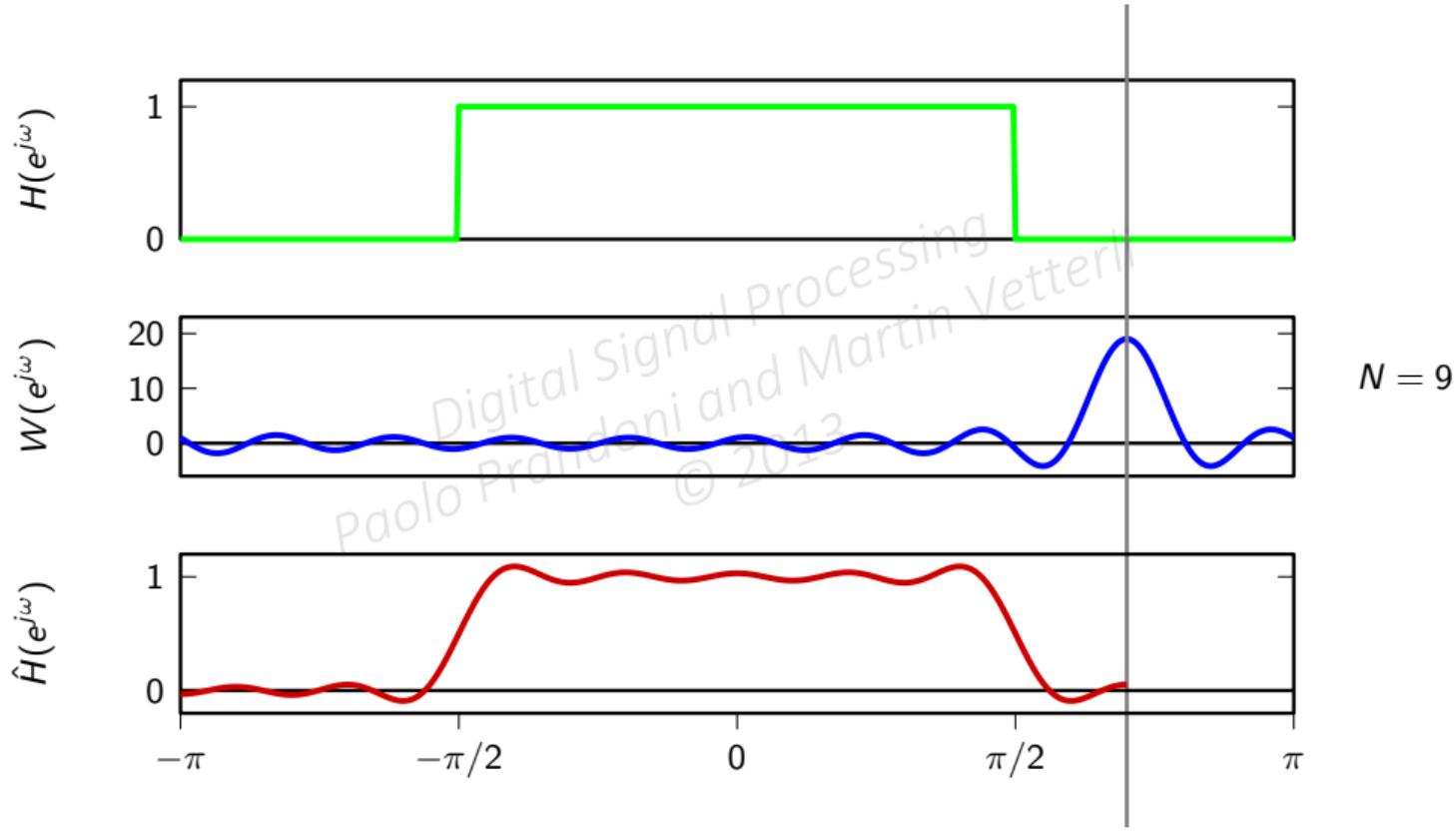
Implicit frequency-domain convolution



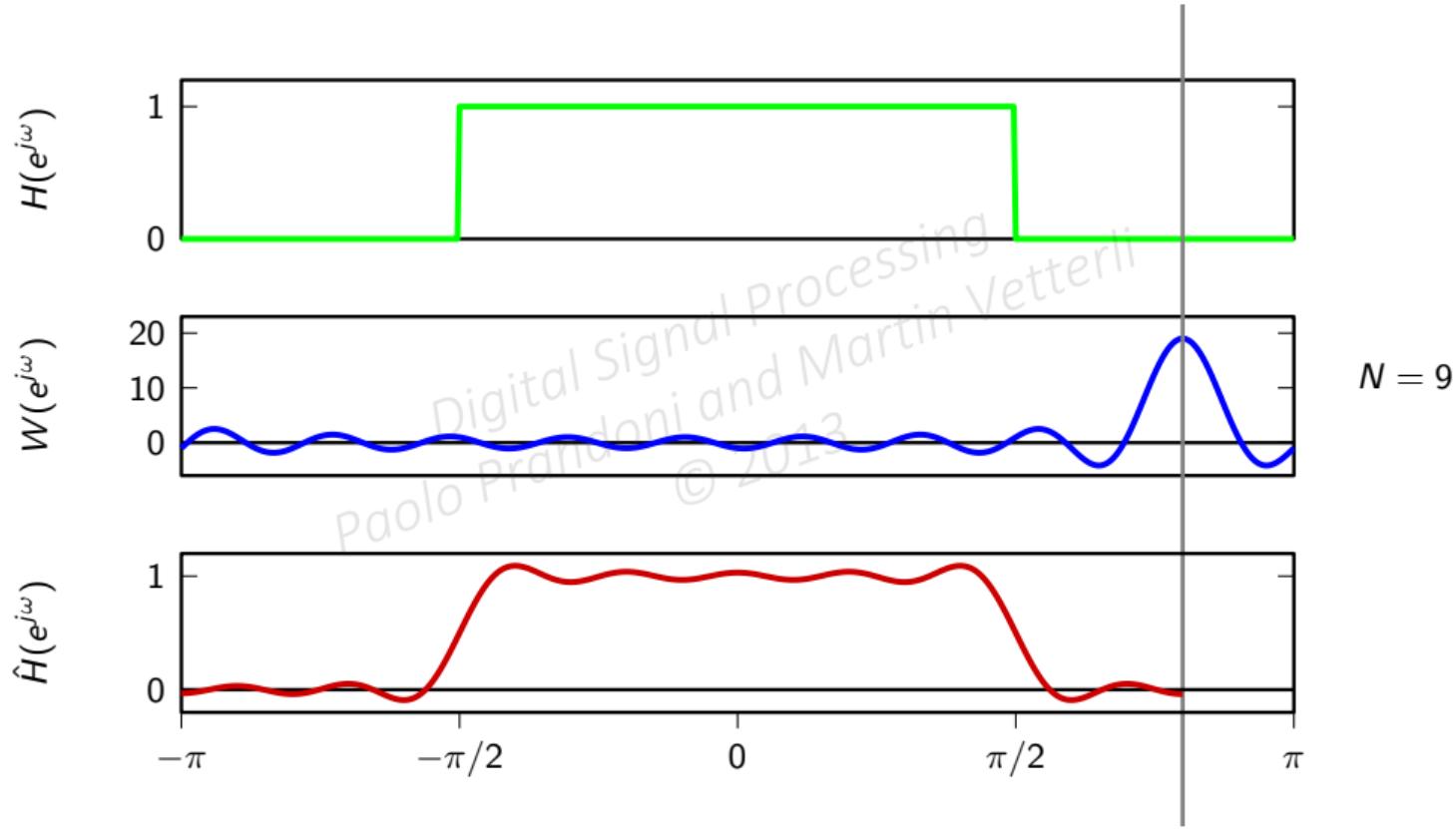
Implicit frequency-domain convolution



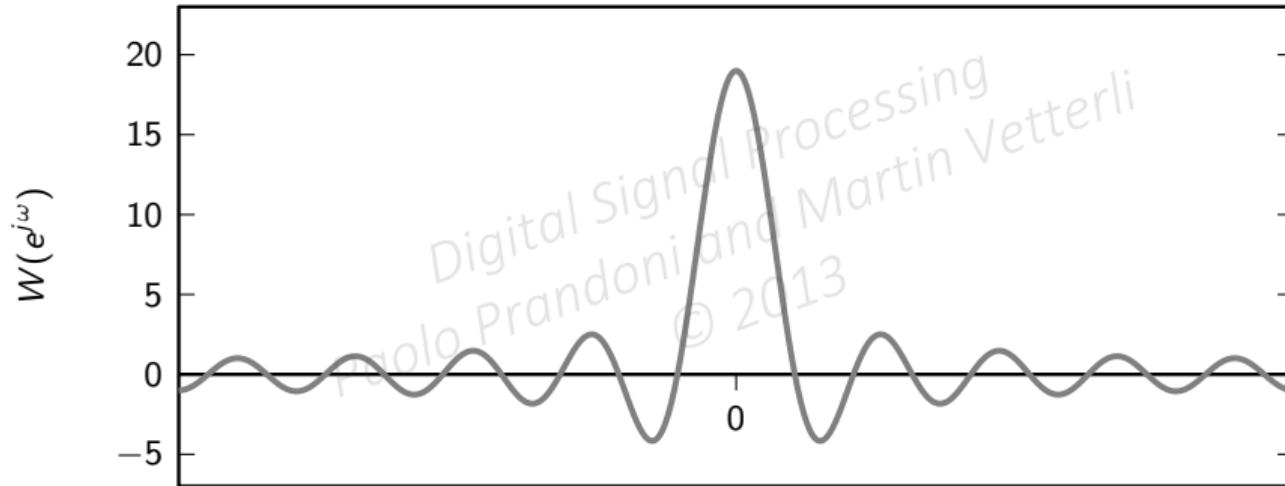
Implicit frequency-domain convolution



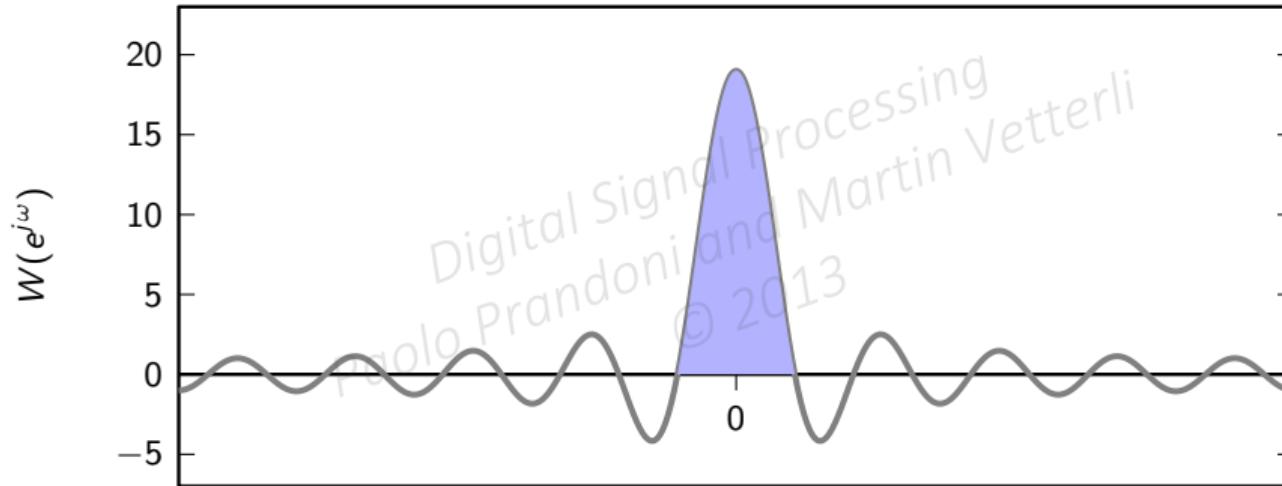
Implicit frequency-domain convolution



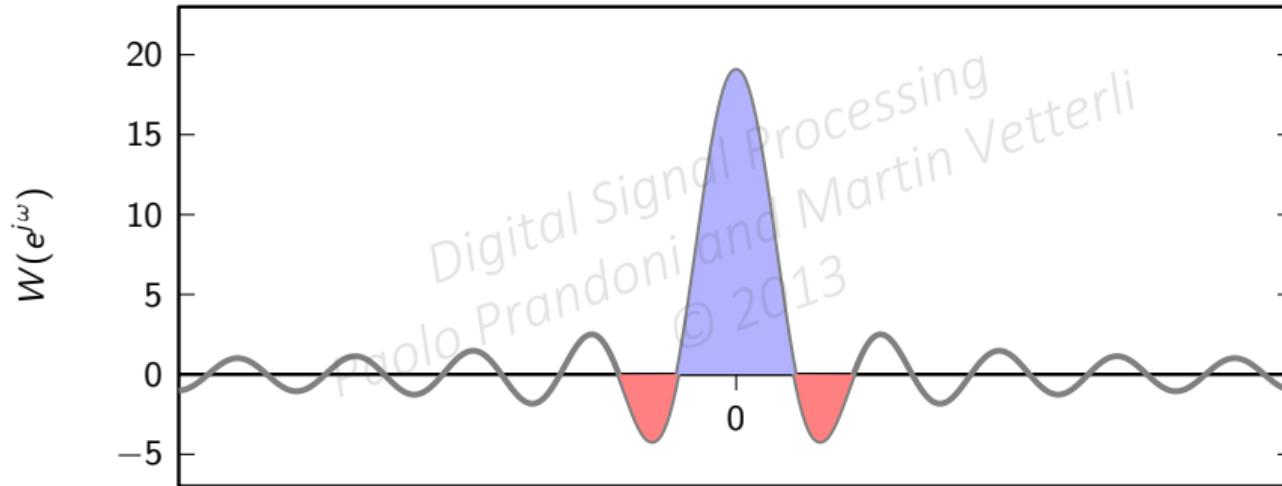
Mainlobe and sidelobes



Mainlobe and sidelobes



Mainlobe and sidelobes



What if we change the window?

We want:

- ▶ narrow mainlobe so that transition is sharp
- ▶ small sidelobe so Gibbs error is small
- ▶ short window so FIR is efficient

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

What if we change the window?

We want:

- ▶ narrow mainlobe so that transition is sharp
- ▶ small sidelobe so Gibbs error is small
- ▶ short window so FIR is efficient

What if we change the window?

We want:

- ▶ narrow mainlobe so that transition is sharp
- ▶ small sidelobe so Gibbs error is small
- ▶ short window so FIR is efficient

Digital Signal Processing
Paolo Brandolini and Martin Vetterli
© 2013

What if we change the window?

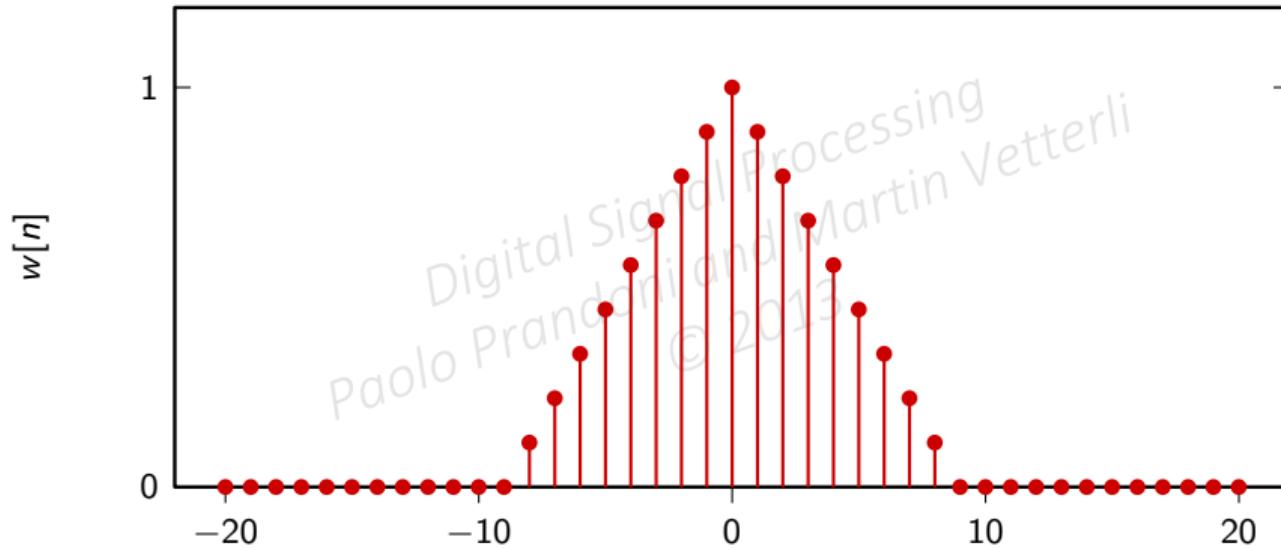
We want:

- ▶ narrow mainlobe so that transition is sharp
- ▶ small sidelobe so Gibbs error is small
- ▶ short window so FIR is efficient

very conflicting requirements!

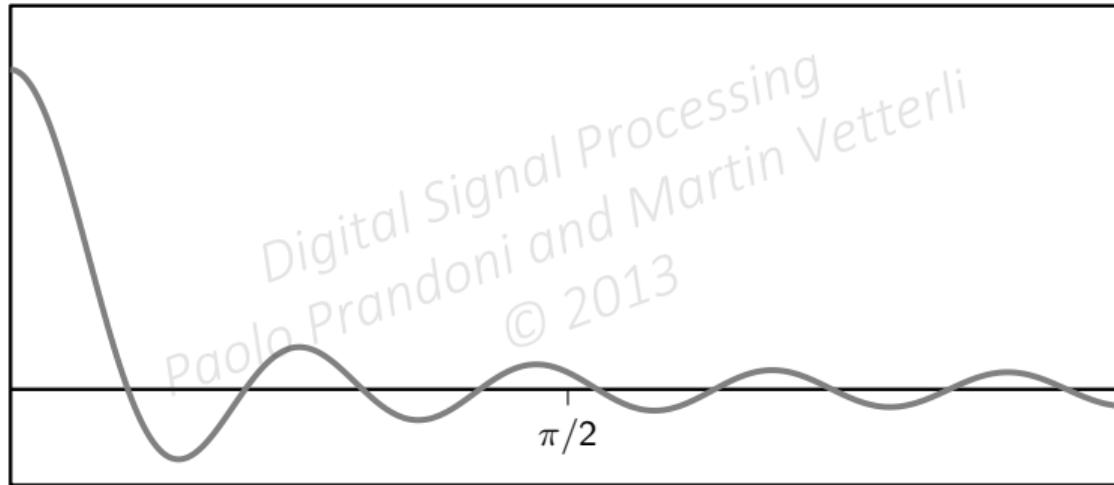
Digital Signal Processing
Paolo Plandoni and Martin Vetterli
© 2013

Triangular window

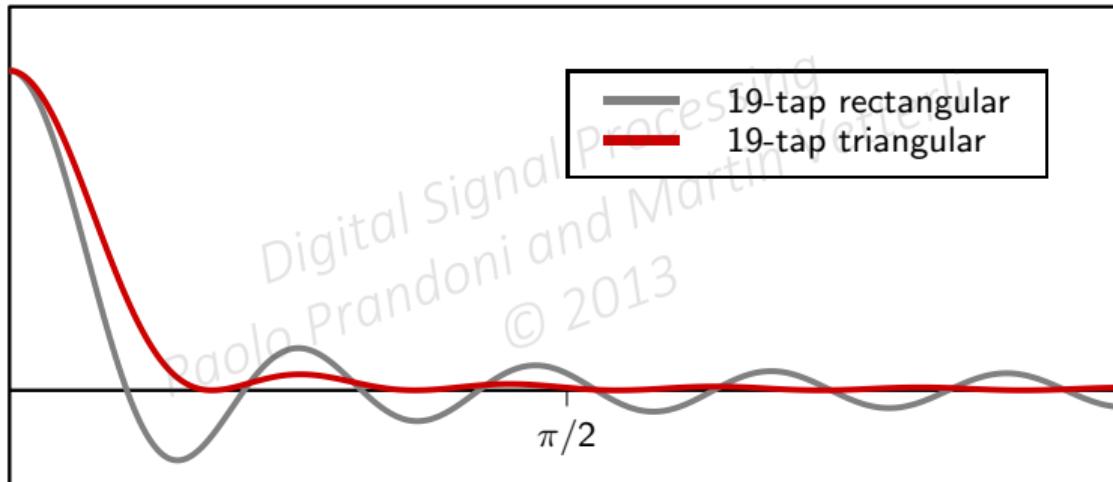


Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Rectangular vs Triangular Window



Rectangular vs Triangular Window



Idea #2:

- ▶ draw desired frequency response $H(e^{j\omega})$
- ▶ take M values at $\omega_k = (2\pi/M)k$
- ▶ compute IDFT of values
- ▶ use result as an M -tap impulse response $\hat{h}[n]$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Idea #2:

- ▶ draw desired frequency response $H(e^{j\omega})$
- ▶ take M values at $\omega_k = (2\pi/M)k$
- ▶ compute IDFT of values
- ▶ use result as an M -tap impulse response $\hat{h}[n]$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Idea #2:

- ▶ draw desired frequency response $H(e^{j\omega})$
- ▶ take M values at $\omega_k = (2\pi/M)k$
- ▶ compute IDFT of values
- ▶ use result as an M -tap impulse response $\hat{h}[n]$

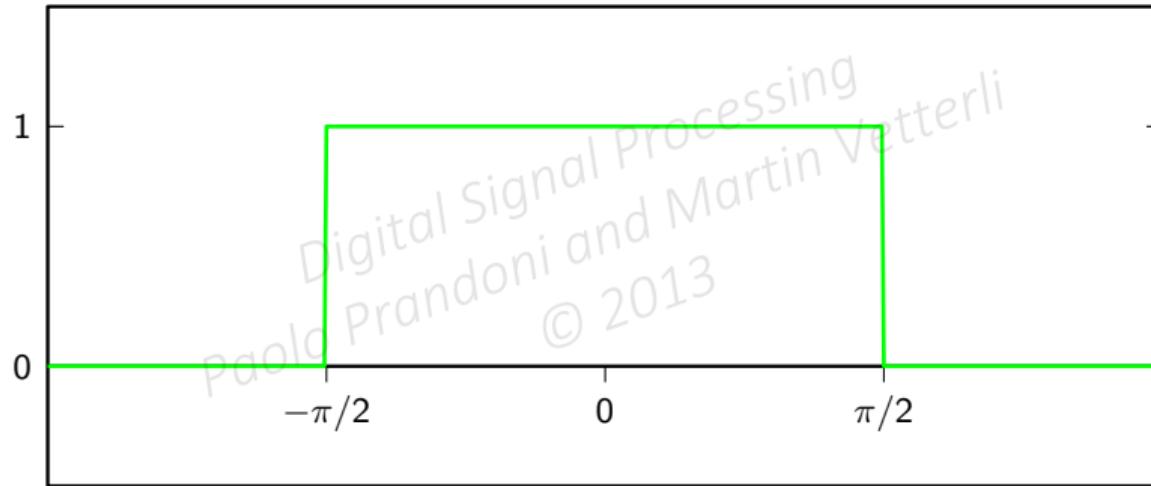
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Idea #2:

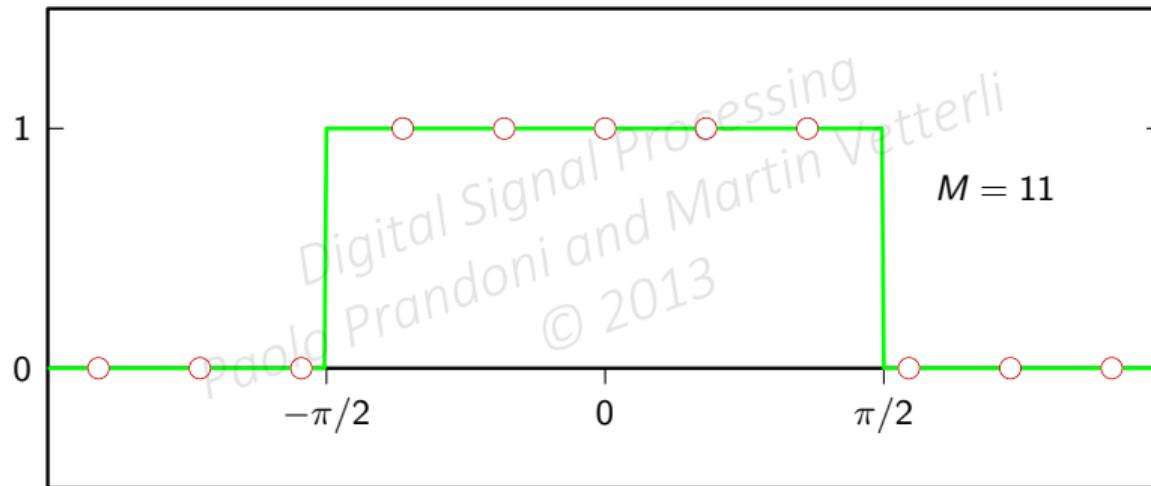
- ▶ draw desired frequency response $H(e^{j\omega})$
- ▶ take M values at $\omega_k = (2\pi/M)k$
- ▶ compute IDFT of values
- ▶ use result as an M -tap impulse response $\hat{h}[n]$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

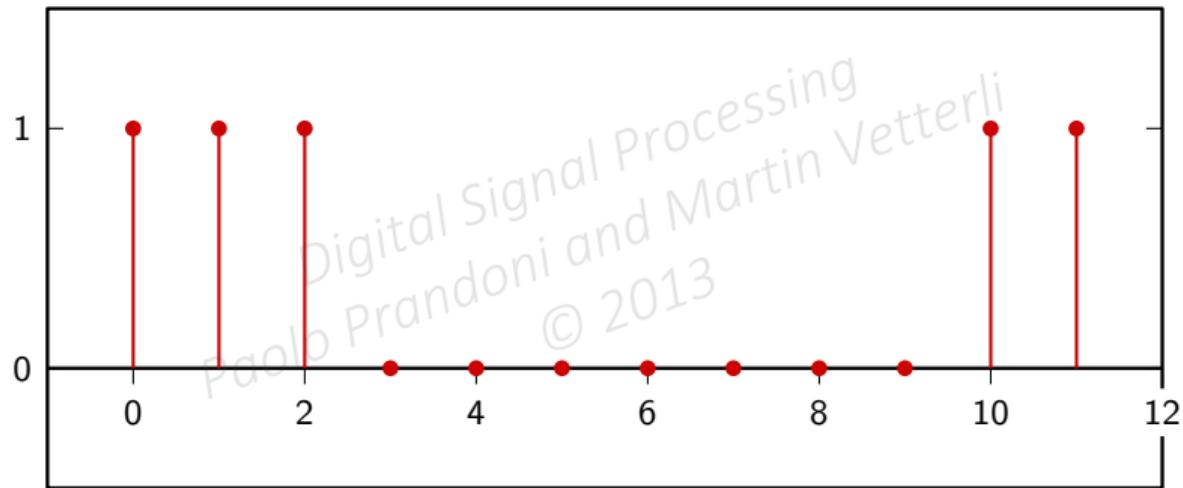
Frequency sampling: desired response



Frequency sampling: desired response

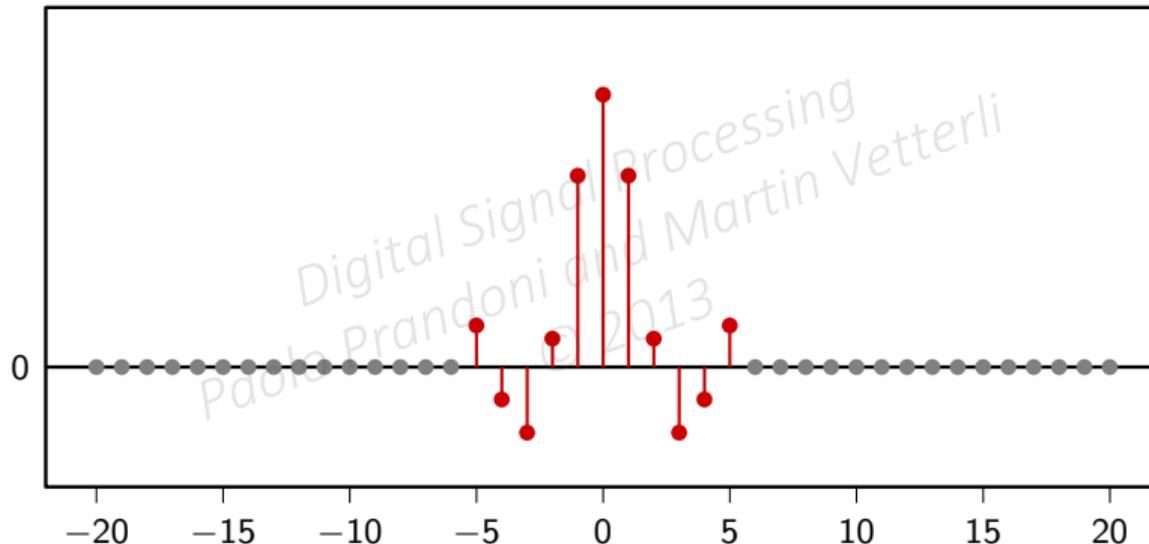


Frequency sampling: DFT samples

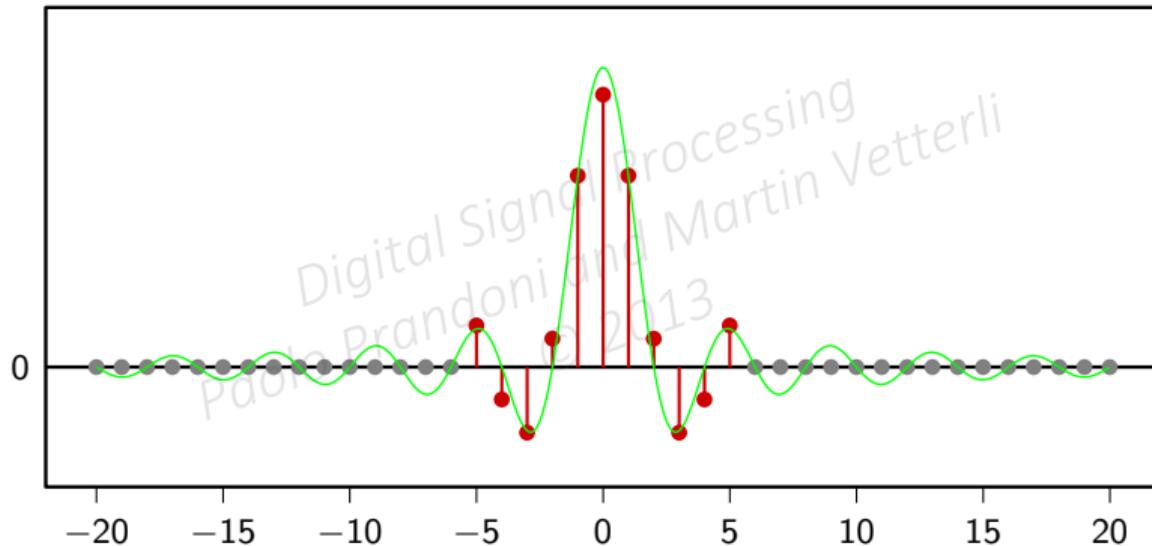


Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Frequency sampling: impulse response from IDFT



Frequency sampling: impulse response from IDFT



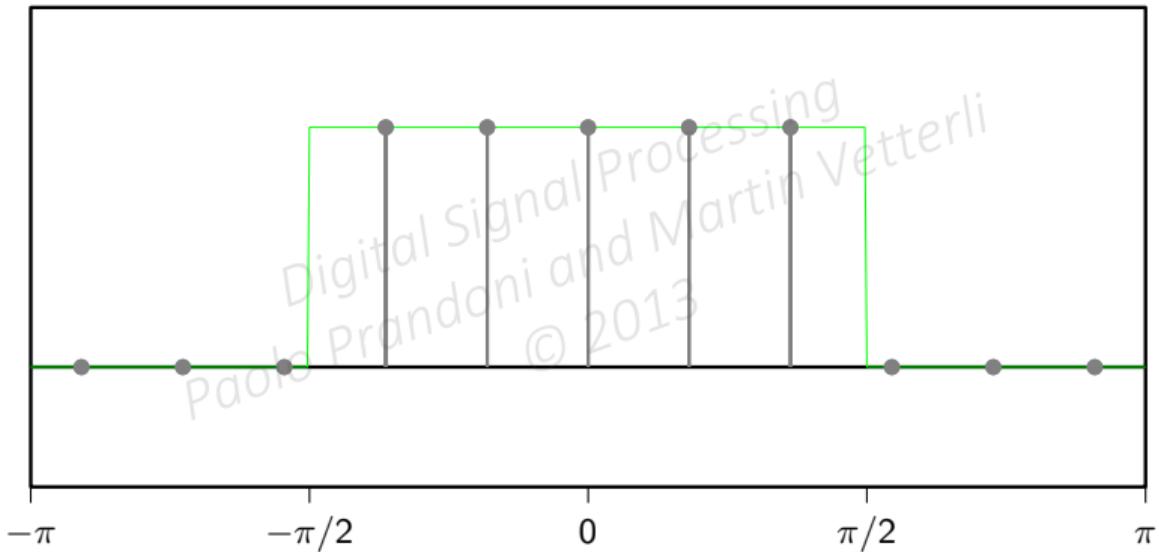
- ▶ frequency response is DTFT of finite-support, whose DFT we know (see Module 4.7)
- ▶ frequency response is interpolation of frequency samples
- ▶ interpolator is transform of N -tap rectangular window
- ▶ again, no control over mainlobe and sidelobes

- ▶ frequency response is DTFT of finite-support, whose DFT we know (see Module 4.7)
- ▶ frequency response is interpolation of frequency samples
- ▶ interpolator is transform of N -tap rectangular window
- ▶ again, no control over mainlobe and sidelobes

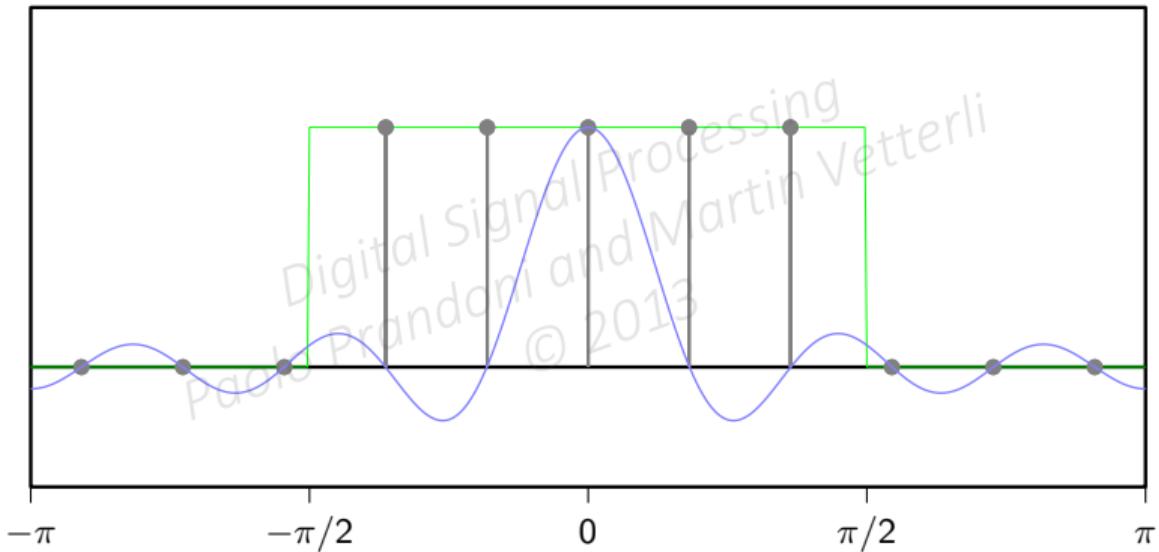
- ▶ frequency response is DTFT of finite-support, whose DFT we know (see Module 4.7)
- ▶ frequency response is interpolation of frequency samples
- ▶ interpolator is transform of N -tap rectangular window
- ▶ again, no control over mainlobe and sidelobes

- ▶ frequency response is DTFT of finite-support, whose DFT we know (see Module 4.7)
- ▶ frequency response is interpolation of frequency samples
- ▶ interpolator is transform of N -tap rectangular window
- ▶ again, no control over mainlobe and sidelobes

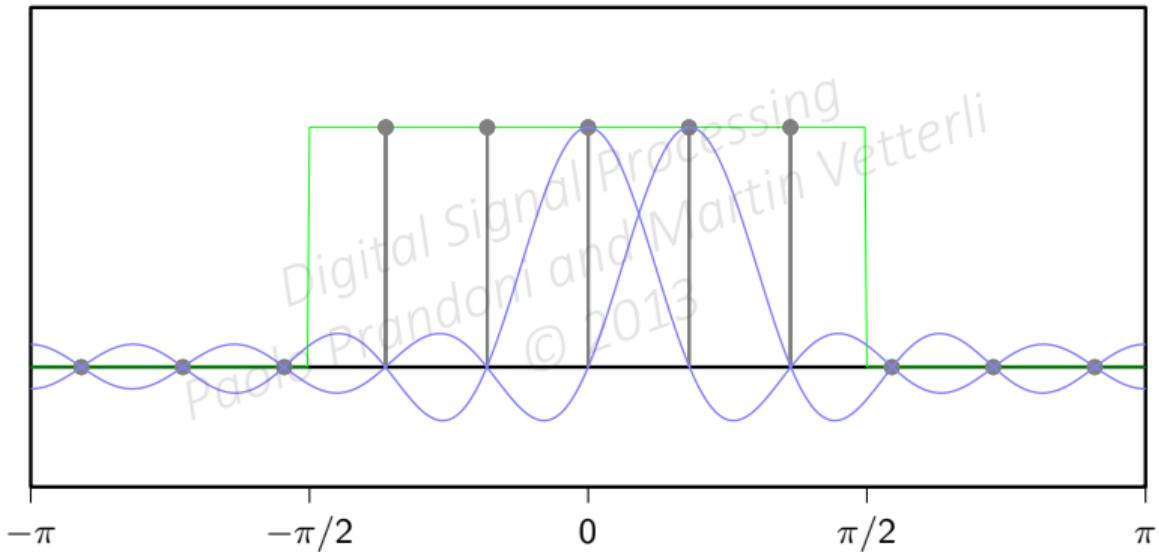
Frequency sampling: frequency response



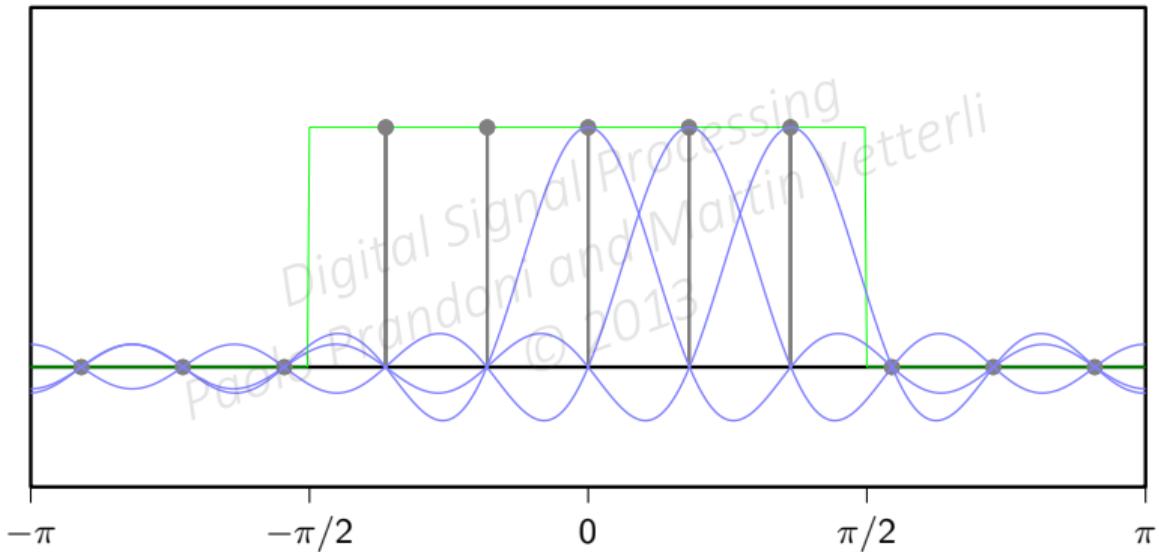
Frequency sampling: frequency response



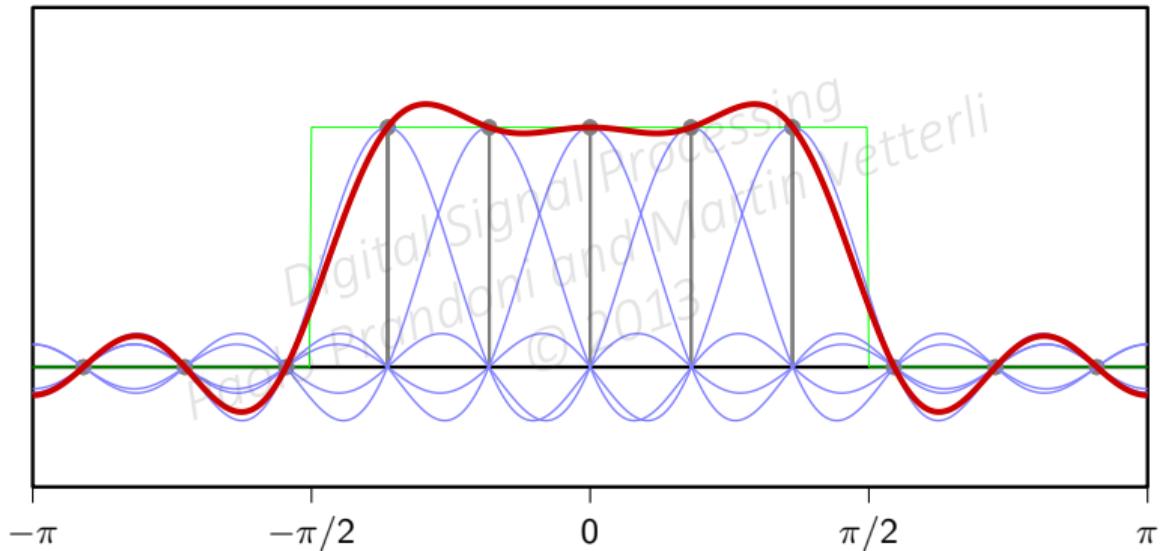
Frequency sampling: frequency response



Frequency sampling: frequency response



Frequency sampling: frequency response



END OF MODULE 5.6

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.7: Realizable Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Constant-Coefficient Difference Equations
- ▶ The z-transform
- ▶ System transfer function
- ▶ Region of convergence and system stability

Digital Signal Processing
Polo Prandoni and Martin Vetterli
© 2013

- ▶ Constant-Coefficient Difference Equations
- ▶ The z -transform
- ▶ System transfer function
- ▶ Region of convergence and system stability

Digital Signal Processing
Polo Prandoni and Martin Vetterli
© 2013

- ▶ Constant-Coefficient Difference Equations
- ▶ The z -transform
- ▶ System transfer function
- ▶ Region of convergence and system stability

Digital Signal Processing
Polo Prandoni and Martin Vetterli
© 2013

- ▶ Constant-Coefficient Difference Equations
- ▶ The z -transform
- ▶ System transfer function
- ▶ Region of convergence and system stability

Digital Signal Processing
P. Prandoni and M. Vetterli
@ 2013

- ▶ ideal filters cannot be implemented
- ▶ what is the most general, realizable LTI transformation?
 - linearity: only sums and multiplications
 - time-invariance: only multiplications by constants
 - realizability: only finite number of past and future samples

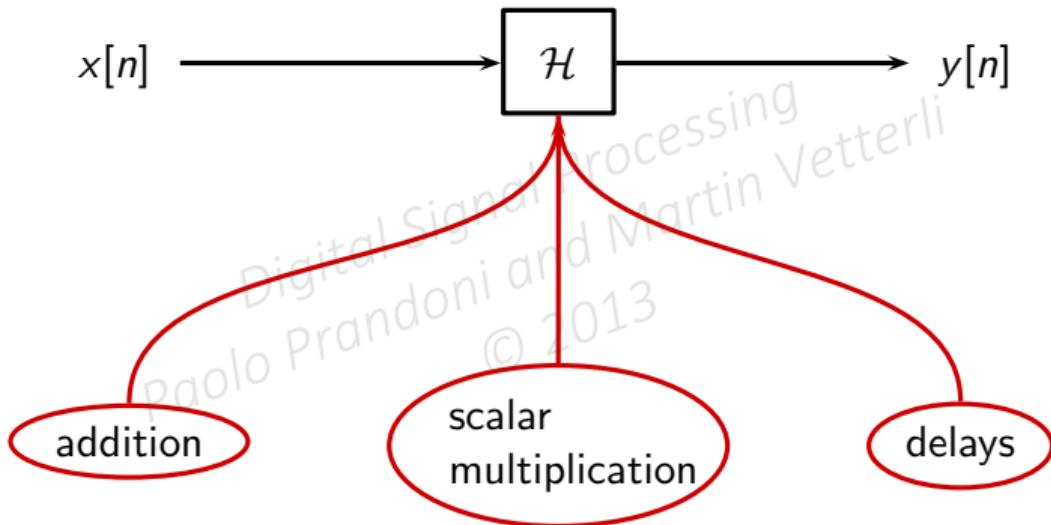
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ ideal filters cannot be implemented
- ▶ what is the most general, realizable LTI transformation?
 - linearity: only sums and multiplications
 - time-invariance: only multiplications by constants
 - realizability: only finite number of past and future samples

- ▶ ideal filters cannot be implemented
- ▶ what is the most general, realizable LTI transformation?
 - linearity: only sums and multiplications
 - time-invariance: only multiplications by constants
 - realizability: only finite number of past and future samples

- ▶ ideal filters cannot be implemented
- ▶ what is the most general, realizable LTI transformation?
 - linearity: only sums and multiplications
 - time-invariance: only multiplications by constants
 - realizability: only finite number of past and future samples

- ▶ ideal filters cannot be implemented
- ▶ what is the most general, realizable LTI transformation?
 - linearity: only sums and multiplications
 - time-invariance: only multiplications by constants
 - realizability: only finite number of past and future samples



$$\sum_{k=0}^{N-1} a_k y[n - k] = \sum_{k=0}^{M-1} b_k x[n - k]$$

- ▶ uses M input and N output values
- ▶ how do we compute the frequency response?
- ▶ we need a new tool!

$$\sum_{k=0}^{N-1} a_k y[n - k] = \sum_{k=0}^{M-1} b_k x[n - k]$$

- ▶ uses M input and N output values
- ▶ how do we compute the frequency response?
- ▶ we need a new tool!

$$\sum_{k=0}^{N-1} a_k y[n - k] = \sum_{k=0}^{M-1} b_k x[n - k]$$

- ▶ uses M input and N output values
- ▶ how do we compute the frequency response?
- ▶ we need a new tool!

$$\sum_{k=0}^{N-1} a_k y[n - k] = \sum_{k=0}^{M-1} b_k x[n - k]$$

- ▶ uses M input and N output values
- ▶ how do we compute the frequency response?
- ▶ we need a new tool!

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}, \quad z \in \mathbb{C}$$

- ▶ think of it as a formal operator
- ▶ ... or as the extension of the DTFT to the complex plane:

$$X(z)|_{z=e^{j\omega}} = \text{DTFT}\{x[n]\}$$

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}, \quad z \in \mathbb{C}$$

- ▶ think of it as a formal operator...
- ▶ ... or as the extension of the DTFT to the whole complex plane:

$$X(z)|_{z=e^{j\omega}} = \text{DTFT}\{x[n]\}$$

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}, \quad z \in \mathbb{C}$$

- ▶ think of it as a formal operator...
- ▶ ... or as the extension of the DTFT to the whole complex plane:

$$X(z)|_{z=e^{j\omega}} = \text{DTFT}\{x[n]\}$$

linearity:

$$\mathcal{Z}\{\alpha x[n] + \beta y[n]\} = \alpha X(z) + \beta Y(z)$$

time shift:

$$\mathcal{Z}\{x[n - N]\} = z^{-N}X(z)$$

linearity:

$$\mathcal{Z}\{\alpha x[n] + \beta y[n]\} = \alpha X(z) + \beta Y(z)$$

time shift:

$$\mathcal{Z}\{x[n - N]\} = z^{-N}X(z)$$

Applying the z -transform to CCDE's

$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k]$$

$$Y(z) \sum_{k=0}^{N-1} a_k z^{-k} \circledcirc H(z) \sum_{k=0}^{M-1} b_k z^{-k}$$

$$Y(z) = H(z)X(z)$$

Applying the z -transform to CCDE's

$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k]$$

$$Y(z) \sum_{k=0}^{N-1} a_k z^{-k} = X(z) \sum_{k=0}^{M-1} b_k z^{-k}$$

$$Y(z) = H(z)X(z)$$

Applying the z -transform to CCDE's

$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k]$$

$$Y(z) \sum_{k=0}^{N-1} a_k z^{-k} = X(z) \sum_{k=0}^{M-1} b_k z^{-k}$$

$$Y(z) = H(z)X(z)$$

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}}$$

by setting $z = e^{j\omega}$ we have the frequency response!

(and now the notation $X(e^{j\omega})$ should make more sense)

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}}$$

by setting $z = e^{j\omega}$ we have the frequency response!

(and now the notation $X(e^{j\omega})$ should make more sense)

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}}$$

by setting $z = e^{j\omega}$ we have the frequency response!

(and now the notation $X(e^{j\omega})$ should make more sense)

Leaky Integrator revisited

$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$

*Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $H(z) = \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$*

$$H(e^{j\omega}) = \frac{(1 - \lambda)}{1 - \lambda e^{-j\omega}}$$

Leaky Integrator revisited

$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$

$$Y(z) = (1 - \lambda)X(z) + \lambda z^{-1} Y(z)$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $H(z) \circledcirc \frac{(1-\lambda)}{1-\lambda z^{-1}}$

$$H(e^{j\omega}) = \frac{(1 - \lambda)}{1 - \lambda e^{-j\omega}}$$

Leaky Integrator revisited

$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$

$$Y(z) = (1 - \lambda)X(z) + \lambda z^{-1} Y(z)$$

$$H(z) \stackrel{\textcircled{c}}{=} \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$$

$$H(e^{j\omega}) = \frac{(1 - \lambda)}{1 - \lambda e^{-j\omega}}$$

Leaky Integrator revisited

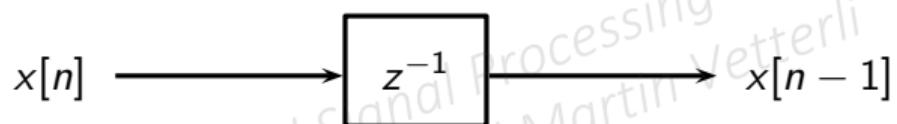
$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$

$$Y(z) = (1 - \lambda)X(z) + \lambda z^{-1} Y(z)$$

$$H(z) \stackrel{\circ}{=} \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$$

$$H(e^{j\omega}) = \frac{(1 - \lambda)}{1 - \lambda e^{-j\omega}}$$

BTW, remember the delay block?



now the notation should make more sense!

the z -transform is a power series, so convergence is always absolute

$$|X(z)| < \infty \iff \sum_{n=-\infty}^{\infty} |x[n]z^{-n}| < \infty$$

Region of convergence (ROC)

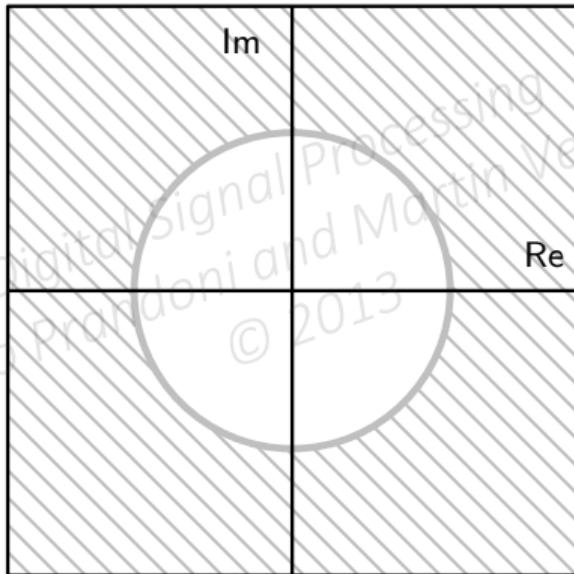
- ▶ ROC is whole complex plane for finite-support sequences
- ▶ ROC has circular symmetry (depends only on $|z|$)
- ▶ ROC of causal sequences extends from a circle to infinity

Digital Signal Processing
Paolo Pandolfi and Martin Vetterli
© 2013

- ▶ ROC is whole complex plane for finite-support sequences
- ▶ ROC has circular symmetry (depends only on $|z|$)
- ▶ ROC of causal sequences extends from a circle to infinity

- ▶ ROC is whole complex plane for finite-support sequences
- ▶ ROC has circular symmetry (depends only on $|z|$)
- ▶ ROC of causal sequences extends from a circle to infinity

ROC for causal systems



Digital Signal Processing
Paolo Brandoli and Martin Vetterli
© 2013

Consider the transfer function for an LTI system:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_{M-1} z^{-(M-1)}}{a_0 + a_1 z^{-1} + \dots + a_{N-1} z^{-(N-1)}}$$

It can always be factored as:

$$H(z) = C \frac{\prod_{n=1}^{M-1} (1 - z_n z^{-1})}{\prod_{n=1}^{N-1} (1 - p_n z^{-1})}$$

- ▶ z_n 's: zeros of the transfer function
- ▶ p_n 's: poles of the transfer function
- ▶ only trouble spots for ROC are the poles

Digital Signal Processing
Paolo Pandoni and Martin Vetterli
© 2013

- ▶ z_n 's: *zeros* of the transfer function
- ▶ p_n 's: *poles* of the transfer function
- ▶ only trouble spots for ROC are the poles

Digital Signal Processing
Paolo PANDONI and Martin Vetterli
© 2013

- ▶ z_n 's: zeros of the transfer function
- ▶ p_n 's: poles of the transfer function
- ▶ only trouble spots for ROC are the poles

Digital Signal Processing
Paolo Frandolini and Martin Vetterli
© 2013

We know:

- ▶ ROC extends outwards
- ▶ ROC cannot include poles

ROC extends outwards from a circle touching the largest-magnitude pole

Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

We know:

- ▶ ROC extends outwards
- ▶ ROC cannot include poles

ROC extends outwards from a circle touching the largest-magnitude pole

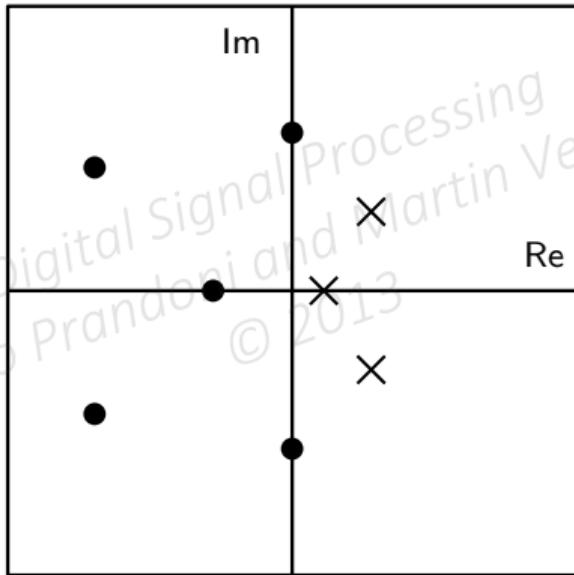
Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

We know:

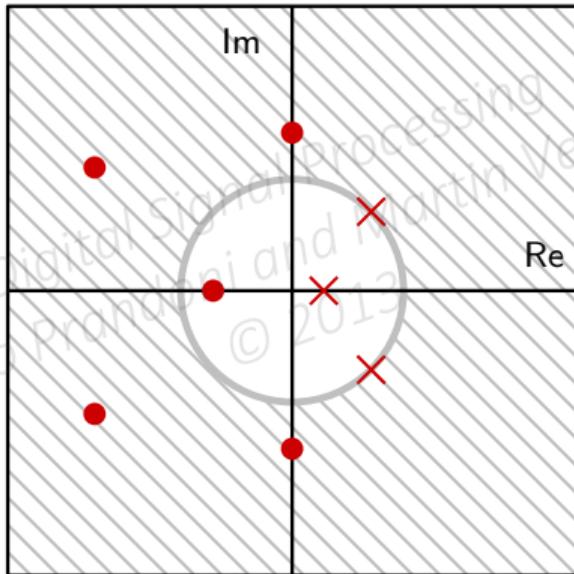
- ▶ ROC extends outwards
- ▶ ROC cannot include poles

ROC extends outwards from a circle touching the largest-magnitude pole

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



Consider a filter with impulse response $h[n]$

- ▶ BIBO stability $\iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (Module 5.3)
- ▶ $H(z)|_{|z|=1} < \infty \iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (Absolute convergence of z-transform)
Digital Signal Processing
Paolo Prahadi and Martin Vetterli
© 2013

system is stable if and only if ROC includes the unit circle!

Consider a filter with impulse response $h[n]$

- ▶ BIBO stability $\iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (Module 5.3)
- ▶ $H(z)|_{|z|=1} < \infty \iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (absolute convergence of z-transform)

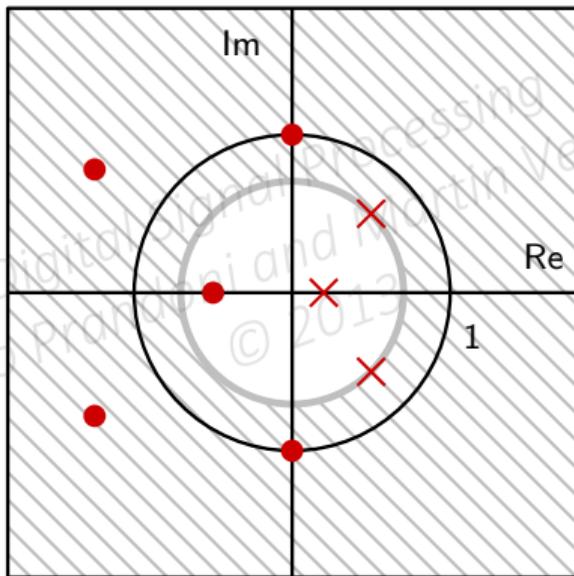
system is stable if and only if ROC includes the unit circle!

Consider a filter with impulse response $h[n]$

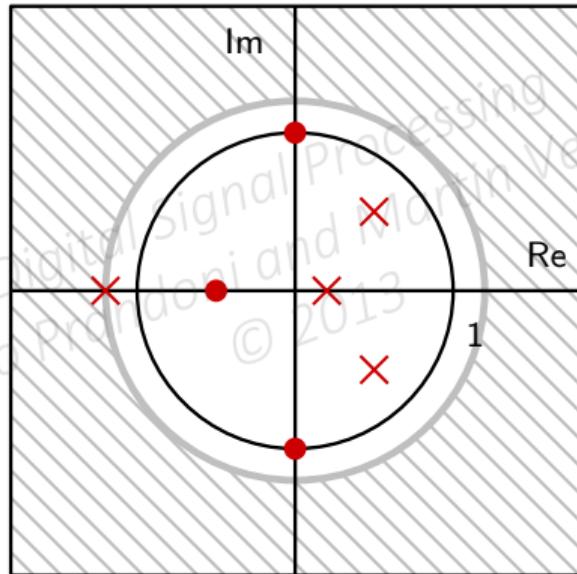
- ▶ BIBO stability $\iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (Module 5.3)
- ▶ $H(z)|_{|z|=1} < \infty \iff \sum_{n=-\infty}^{\infty} |h[n]| < \infty$ (absolute convergence of z-transform)

system is stable if and only if ROC includes the unit circle!

Stable system



Unstable system



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

The “circus tent” method:

- ▶ magnitude of z transform is like a rubber sheet over the complex plane
- ▶ zeros glue the sheet to the ground
- ▶ poles are like ... poles, pushing it up
- ▶ frequency response (in magnitude) is sheet profile around the unit circle

Digital Signal Processing
paolo pandini and Martin Vetterli
© 2013

The “circus tent” method:

- ▶ magnitude of z transform is like a rubber sheet over the complex plane
- ▶ zeros glue the sheet to the ground
- ▶ poles are like ... poles, pushing it up
- ▶ frequency response (in magnitude) is sheet profile around the unit circle

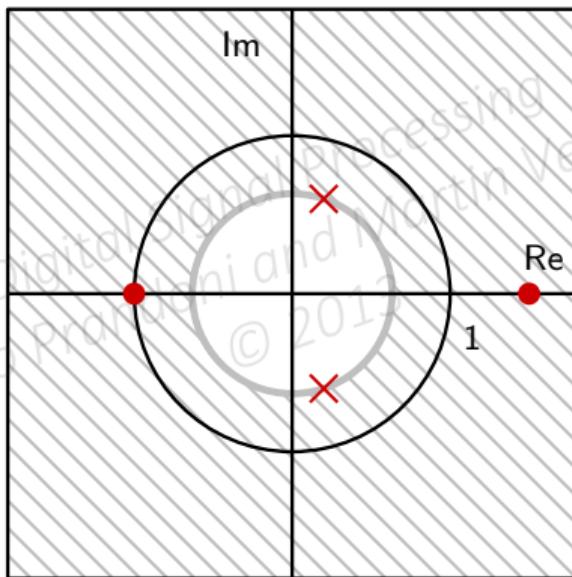
The “circus tent” method:

- ▶ magnitude of z transform is like a rubber sheet over the complex plane
- ▶ zeros glue the sheet to the ground
- ▶ poles are like ... poles, pushing it up
- ▶ frequency response (in magnitude) is sheet profile around the unit circle

The “circus tent” method:

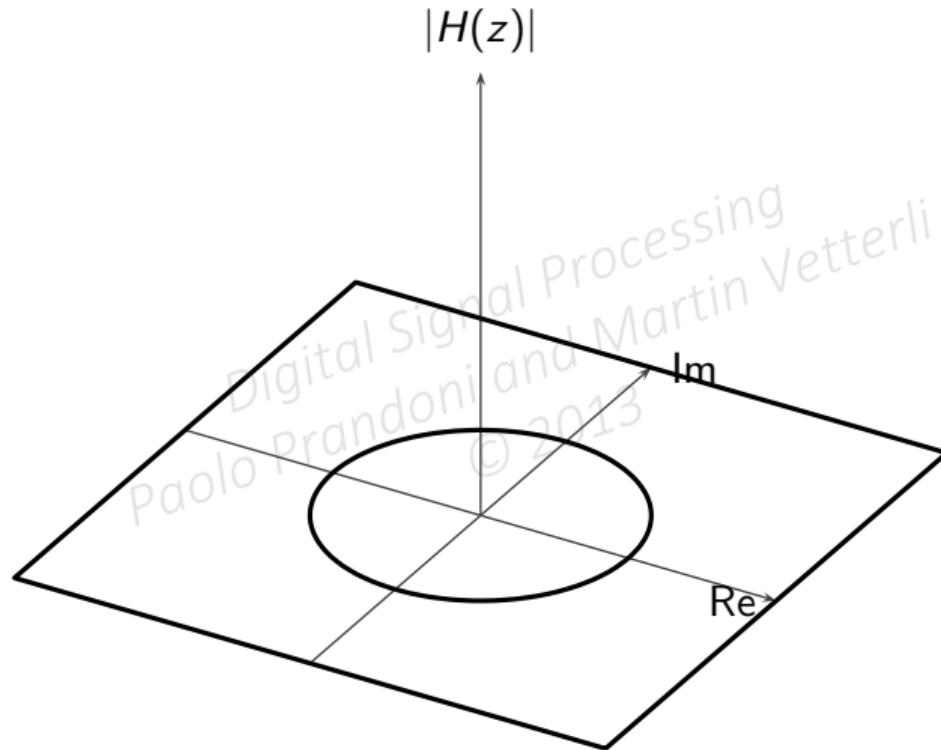
- ▶ magnitude of z transform is like a rubber sheet over the complex plane
- ▶ zeros glue the sheet to the ground
- ▶ poles are like ... poles, pushing it up
- ▶ frequency response (in magnitude) is sheet profile around the unit circle

Estimating the frequency response

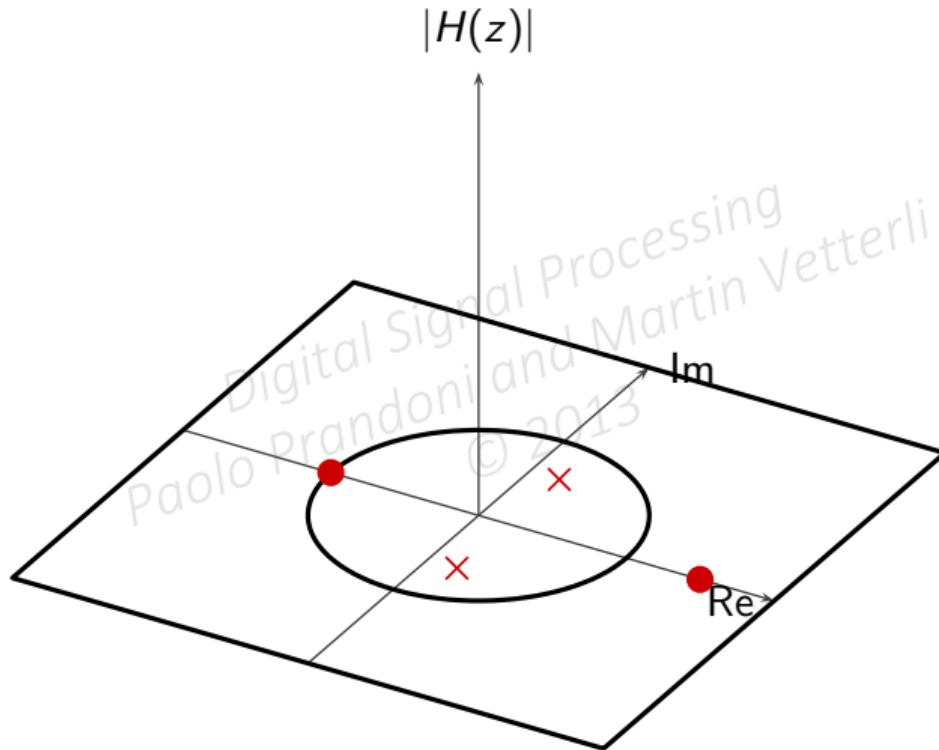


Digital Signal Processing
Paolo Frasconi and Martin Vetterli
© 2013

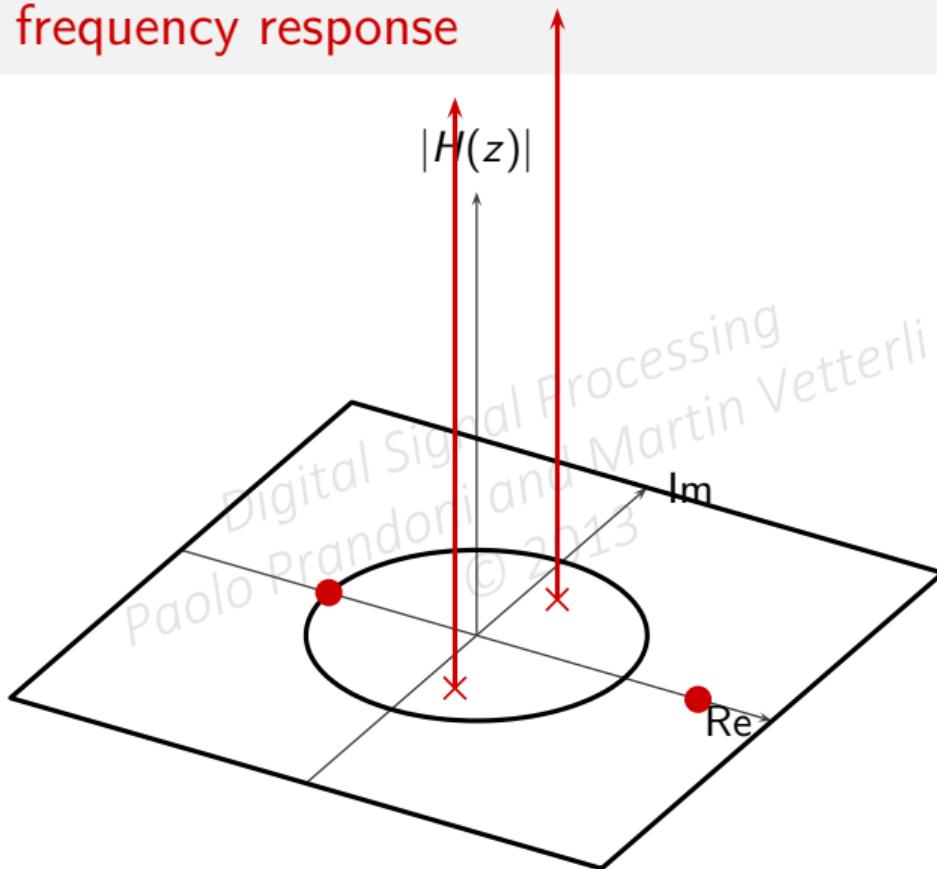
Estimating the frequency response



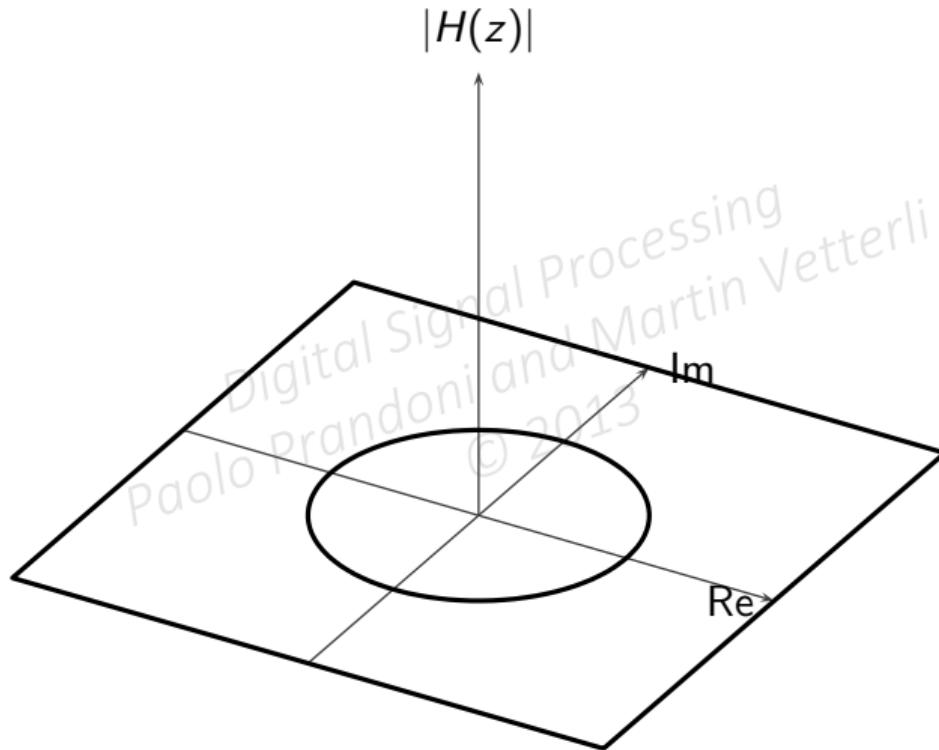
Estimating the frequency response



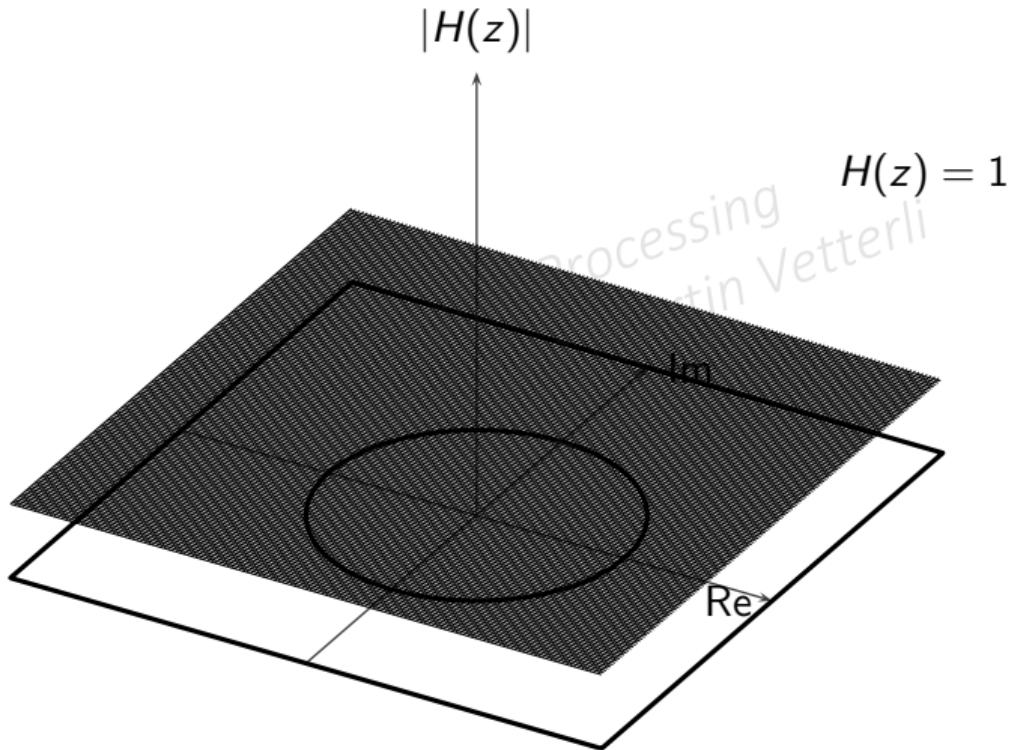
Estimating the frequency response



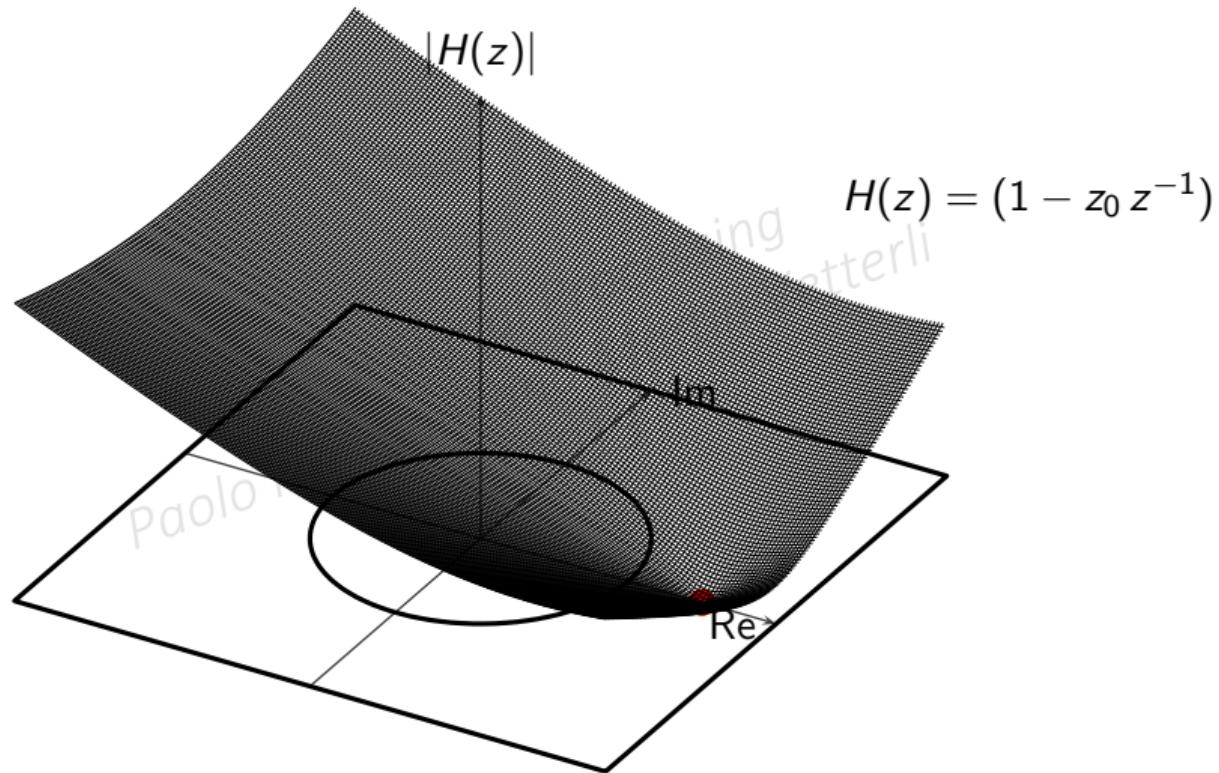
Estimating the frequency response



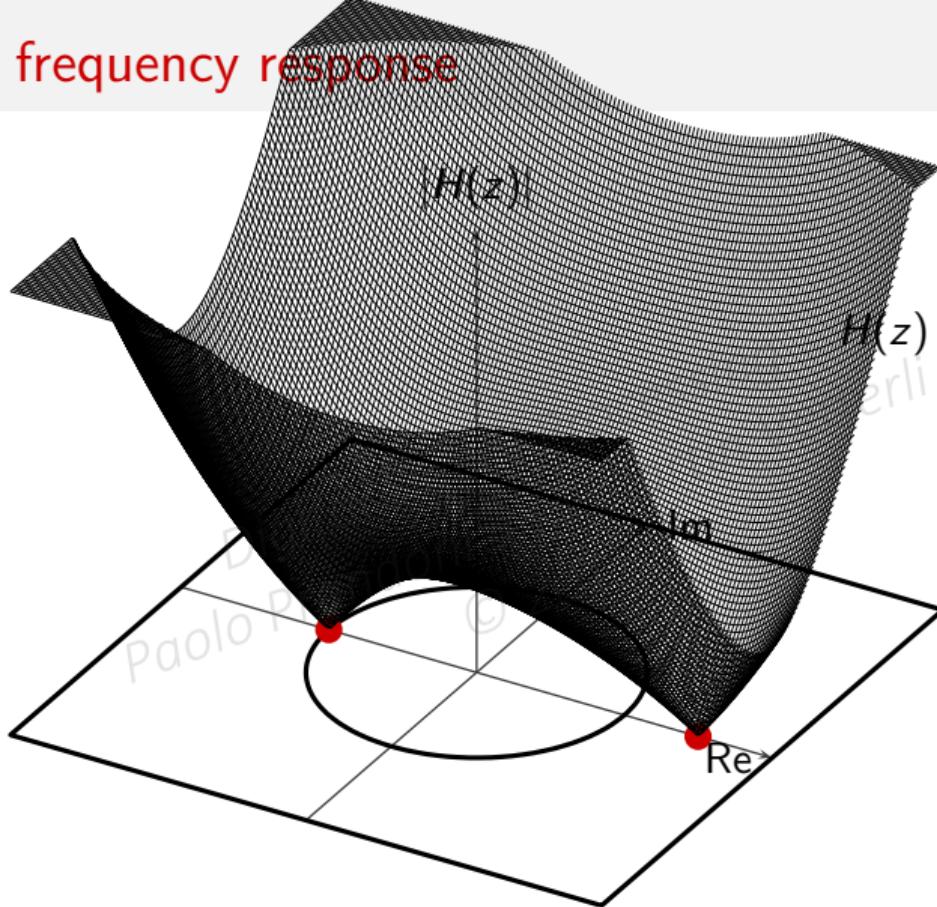
Estimating the frequency response



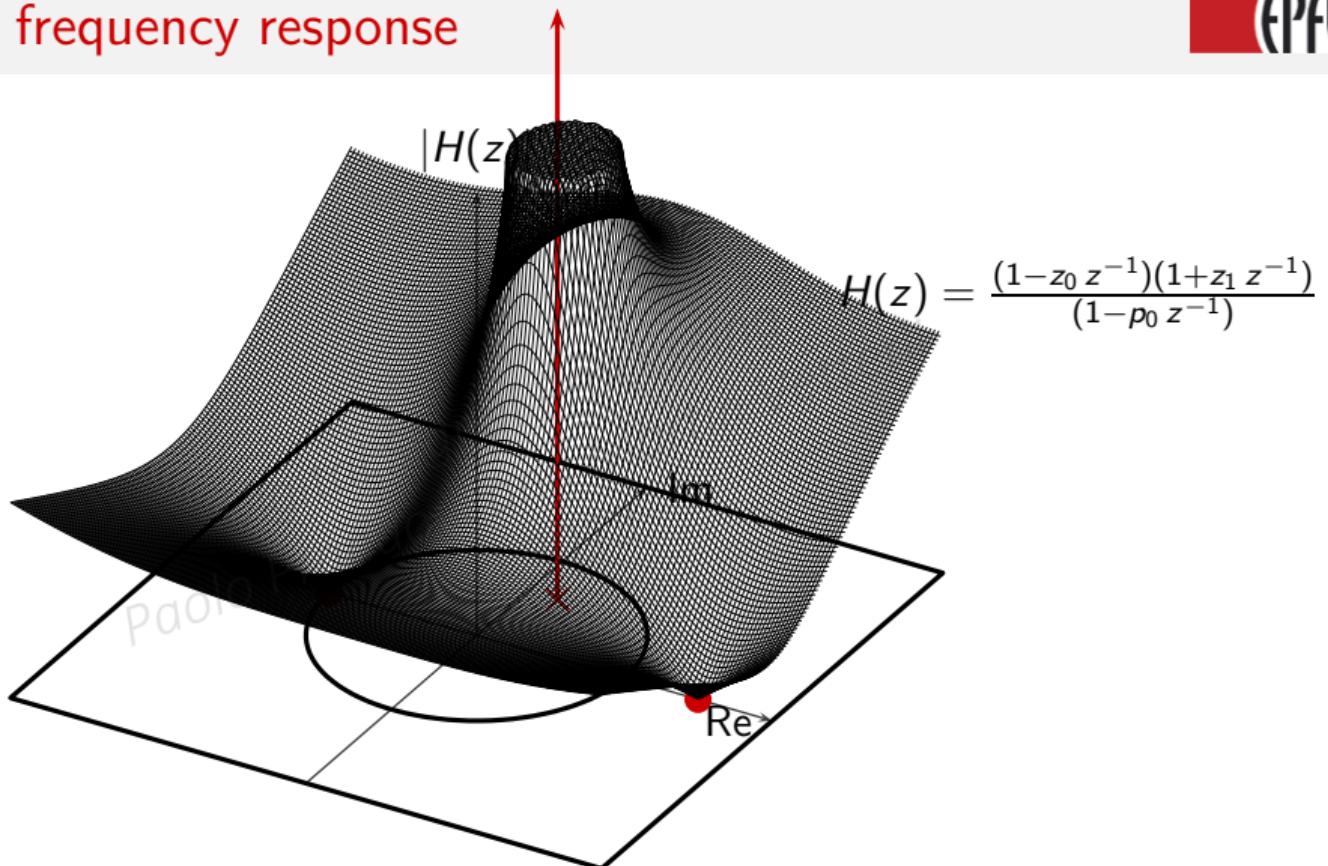
Estimating the frequency response



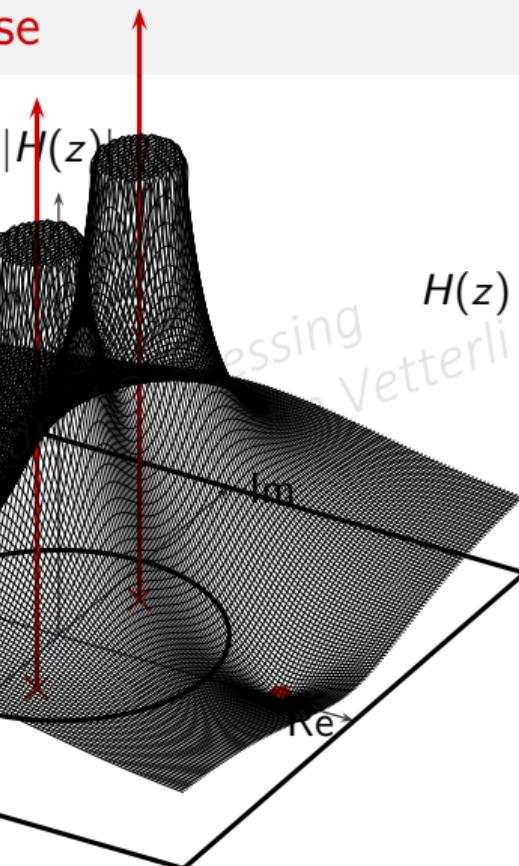
Estimating the frequency response



Estimating the frequency response



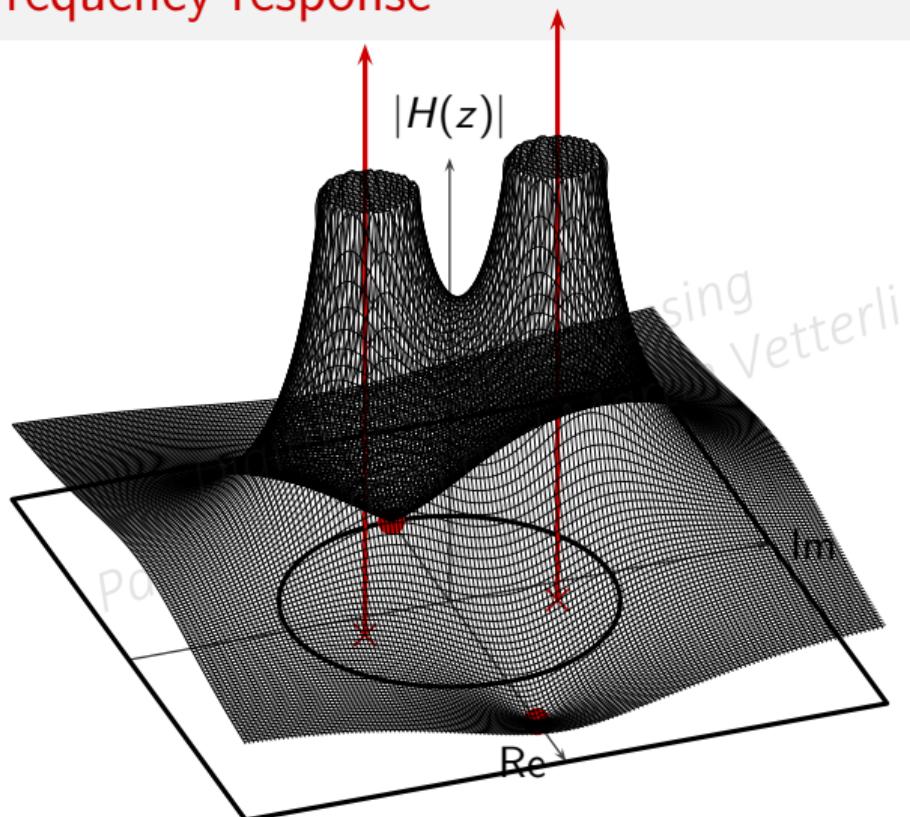
Estimating the frequency response



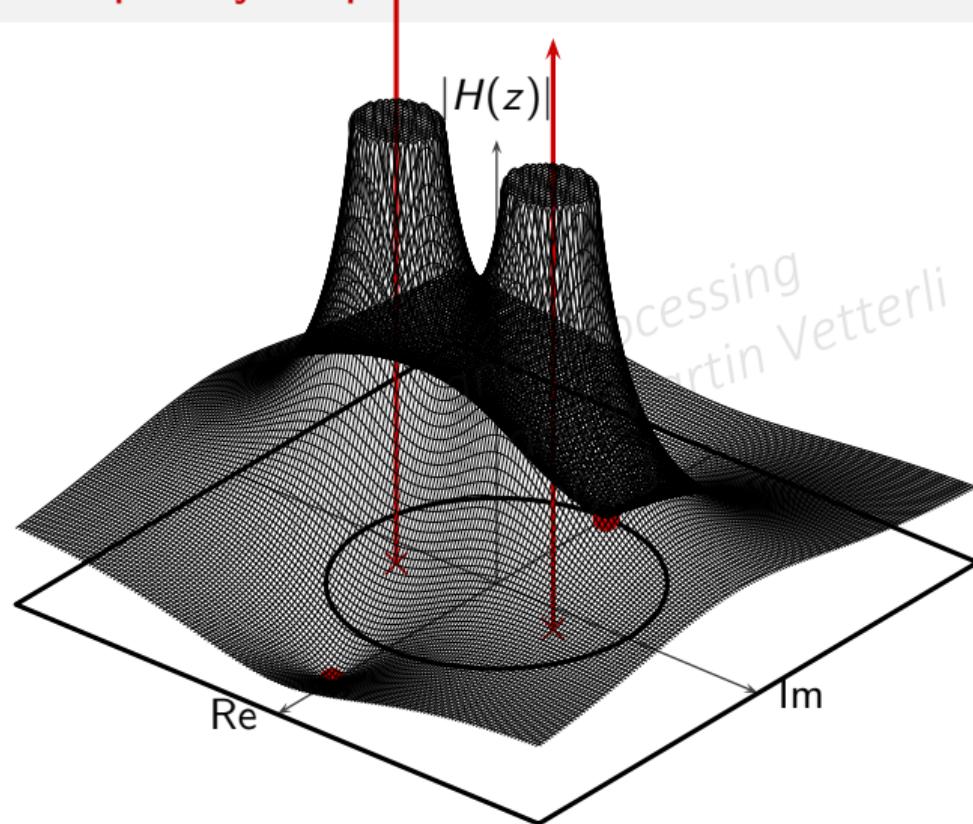
$$H(z) = \frac{(1-z_0 z^{-1})(1+z_1 z^{-1})}{(1-p_0 z^{-1})(1-p_0^* z^{-1})}$$

Processing
Vetterli

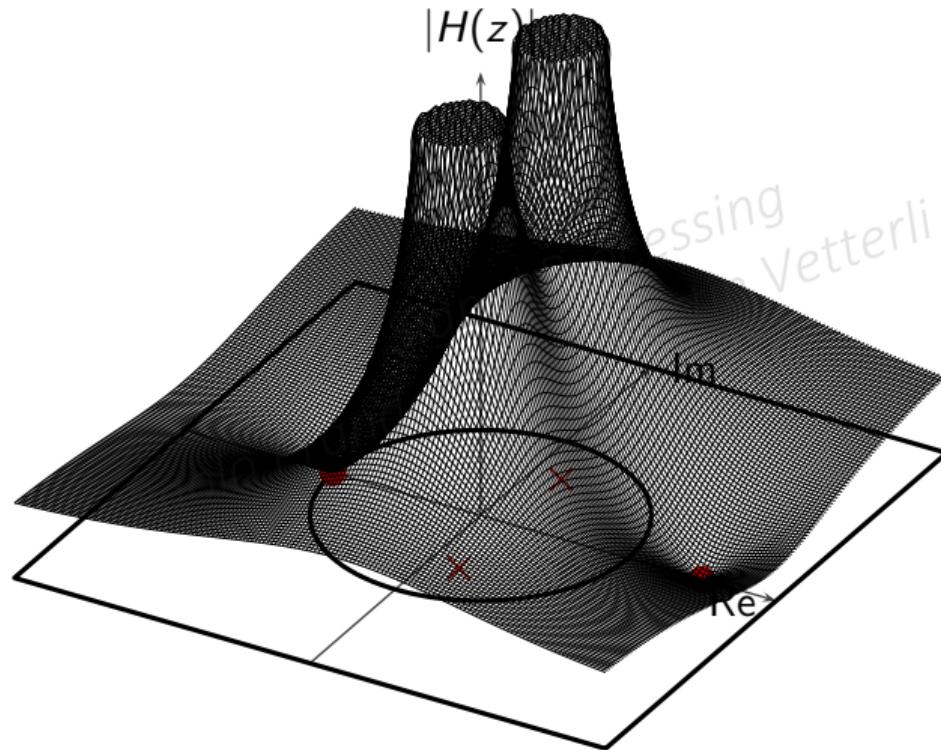
Estimating the frequency response



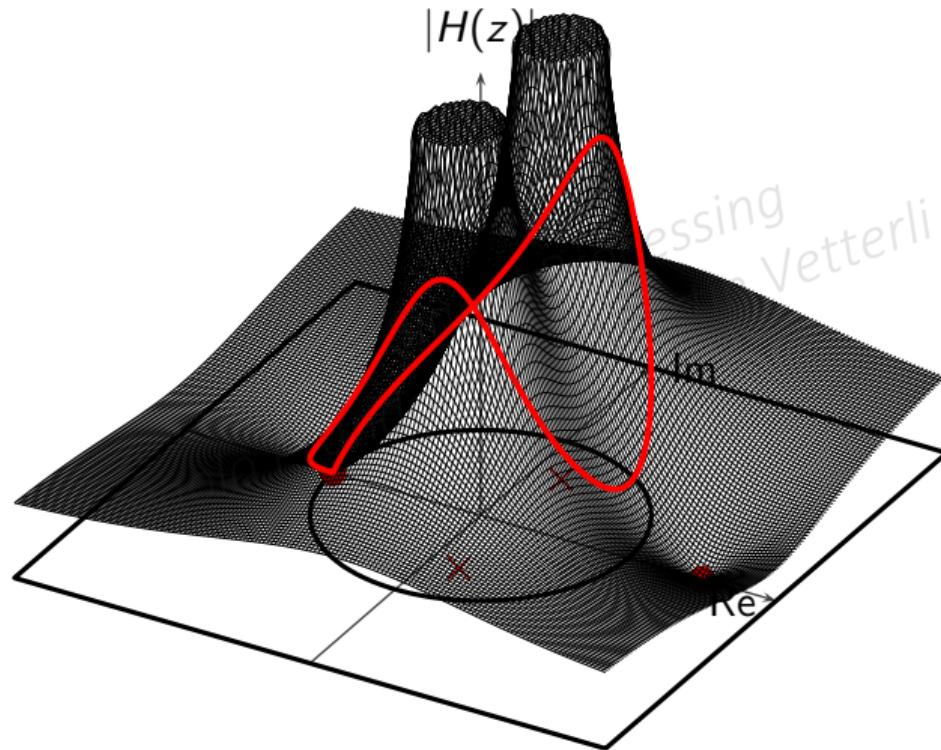
Estimating the frequency response



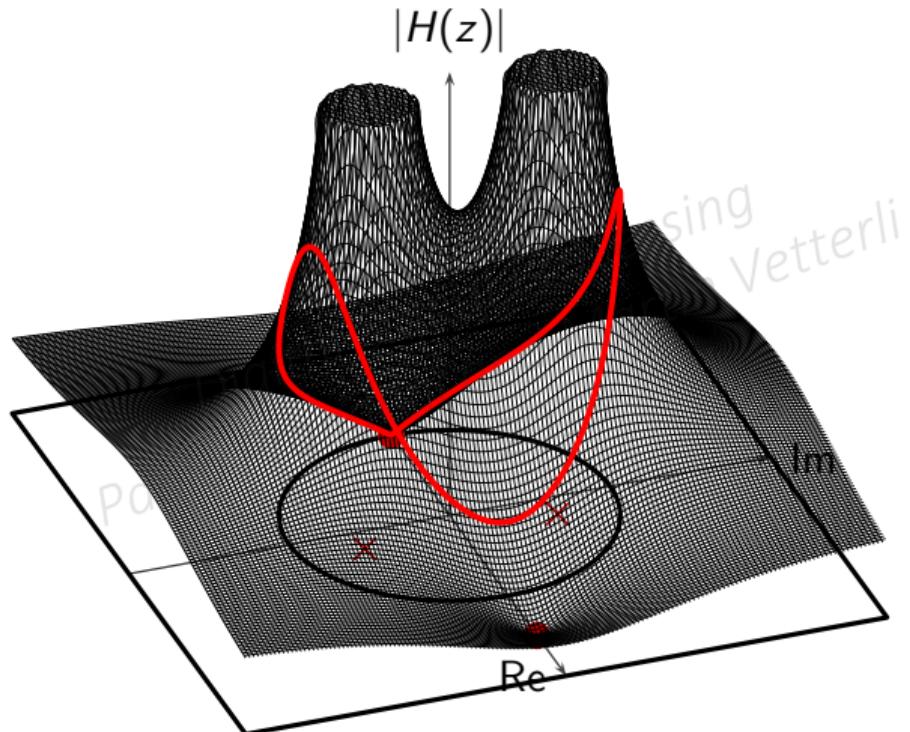
Estimating the frequency response



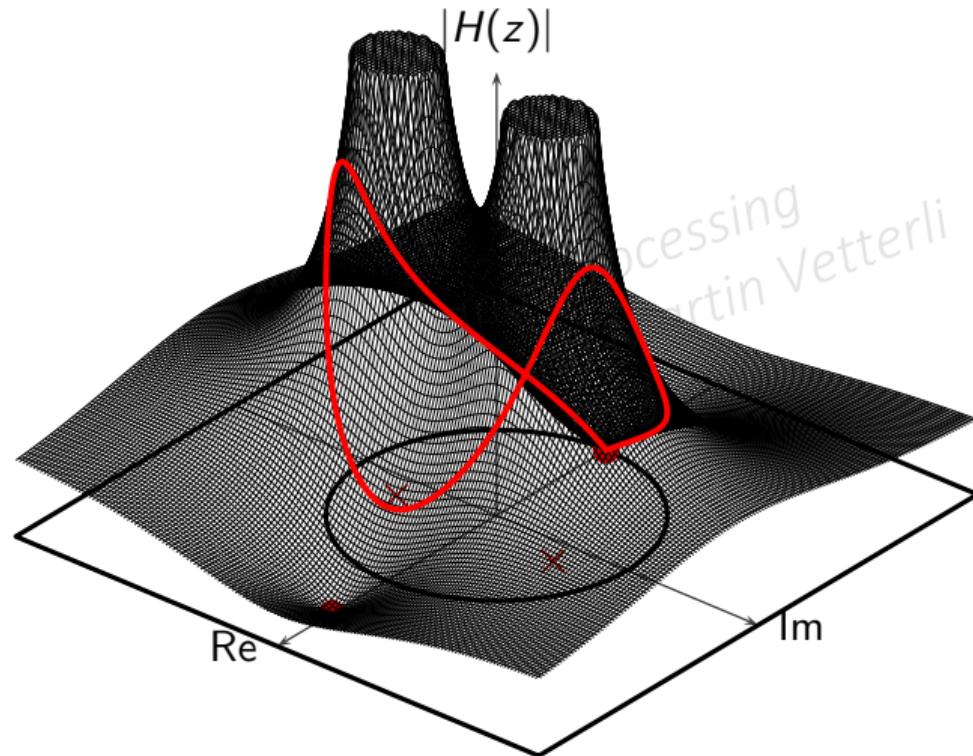
Estimating the frequency response



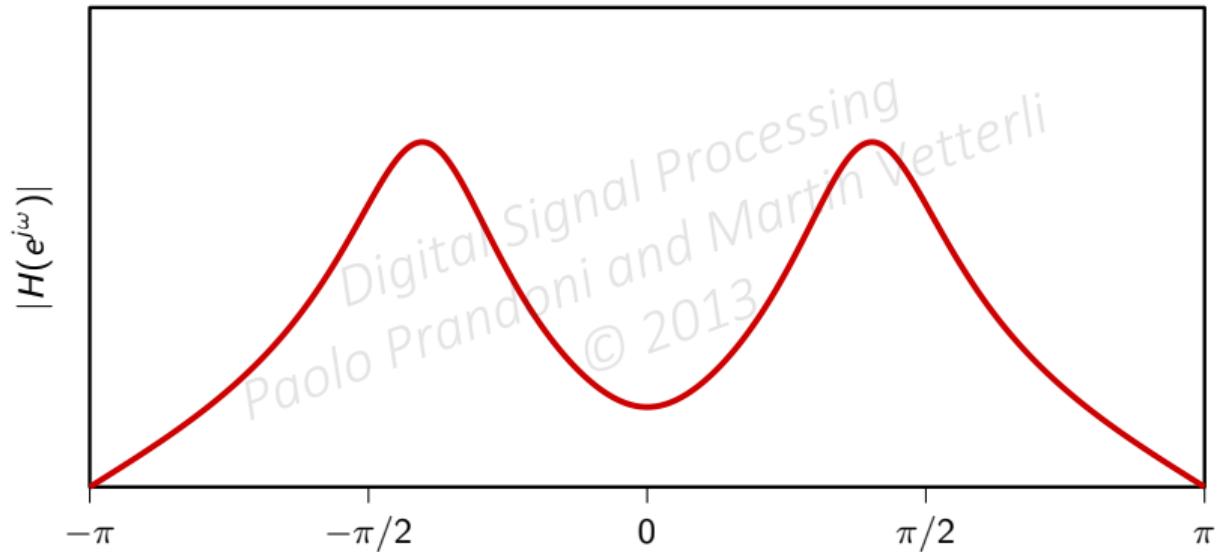
Estimating the frequency response



Estimating the frequency response



Estimating the frequency response



END OF MODULE 5.7

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.8: Implementation of Digital Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Algorithms for CCDE's
- ▶ Block diagram
- ▶ Real-time processing

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

```
double Leaky(double x) {  
    static const double lambda = 0.95;  
    static double y = 0;  
  
    y = lambda * y + (1 - lambda) * x;  
    return y;  
}
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

```
int main() {  
    int n;  
    for (n = 0; n < 20; n++)  
        printf("%.4f ", Leaky(n==0 ? 1.0 : 0.0));  
}
```

0.0500 0.0475 0.0451 0.0429 0.0407 0.0387 0.0368 0.0349 0.0332 0.0315
0.0299 0.0284 0.0270 0.0257 0.0244 0.0232 0.0220 0.0209 0.0199 0.0189

- ▶ we need a “memory cell” to store previous output
- ▶ we need to initialize the storage before first use
- ▶ we need 2 multiplications and one addition per output sample

Some like it OOP

```
class Leaky:  
    def __init__(self, lmb):  
        self.lmb=lmb  
        self.y=0  
  
    def compute(self, x):  
        res = []  
        for v in x:  
            self.y = self.lmb * self.y + (1 - self.lmb) * v  
            res.append(self.y)  
        return res
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Testing the code

```
>>> from leaky import Leaky  
>>> L=Leaky(0.95)  
>>> print L.compute([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0])  
[0.0, 0.0, 0.0, 0.0, 0.0500000000000000, 0.0475000000000000,  
 0.0451250000000000, 0.0428687500000000, 0.0407253125000000,  
 0.038689046875000, 0.0367545945312500]  
>>>
```

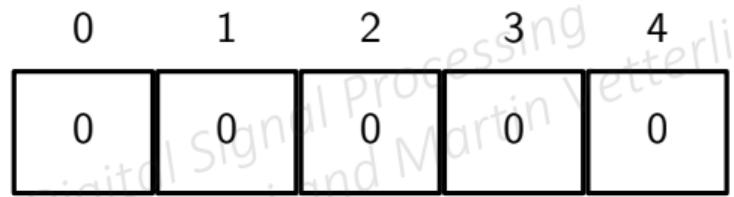
$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

```
double MA(double x) {  
    static const int M = 10;  
    static double z[M];  
    static int ix = -1;  
  
    int n;  
    double avg = 0;
```

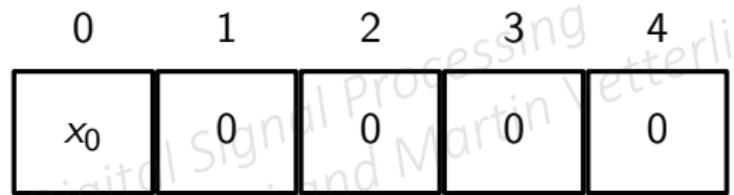
```
        if (ix == -1) {  
            for (n = 0; n < M; n++)  
                z[n] = 0;  
            ix = 0;  
        }  
  
        z[ix] = x;  
        ix = (ix + 1) \% M;  
  
        for (n = 0; n < M; n++)  
            avg += z[n];  
        return avg / M;  
    }
```

The circular buffer



Digital Signal Processing
Paolo Plandoni and Martin Vetterli
© 2013

The circular buffer



The circular buffer



Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© ix ↑ 2013

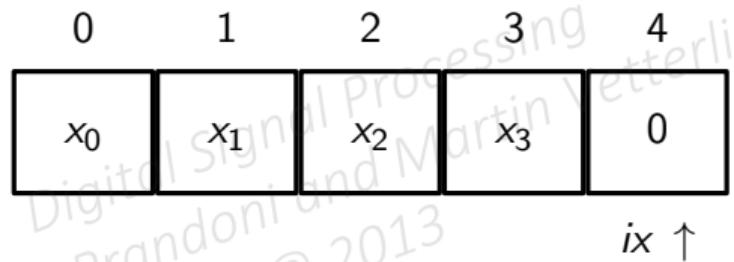
The circular buffer



$ix \uparrow$

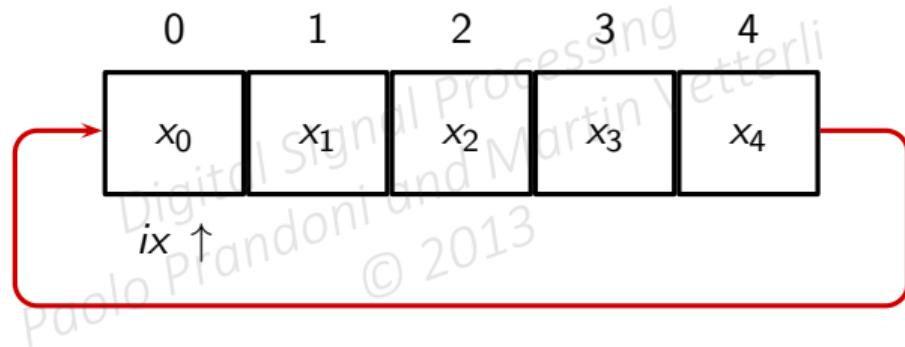
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

The circular buffer

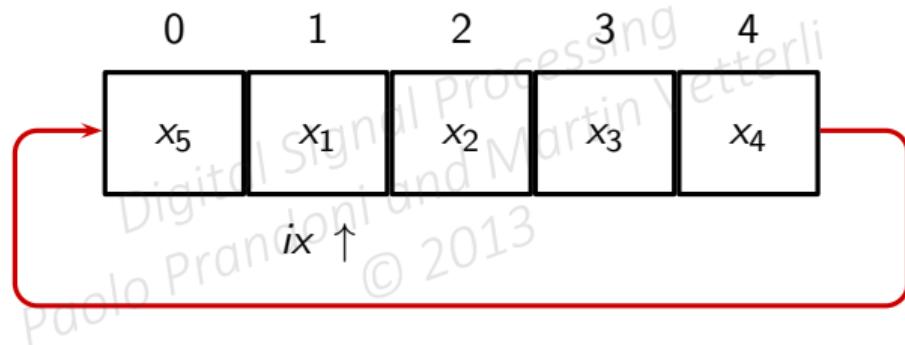


Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

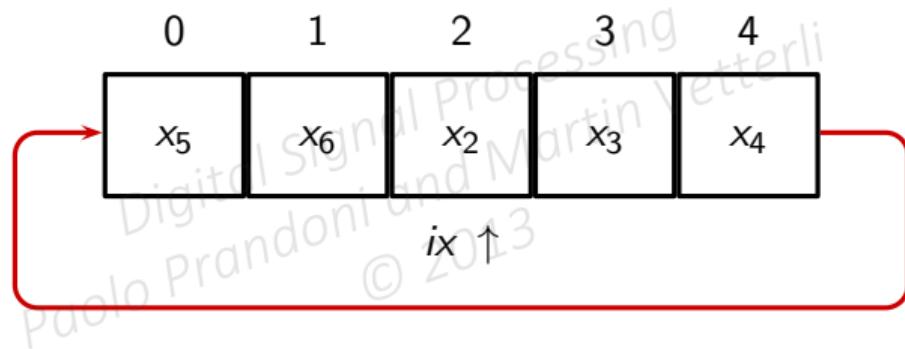
The circular buffer



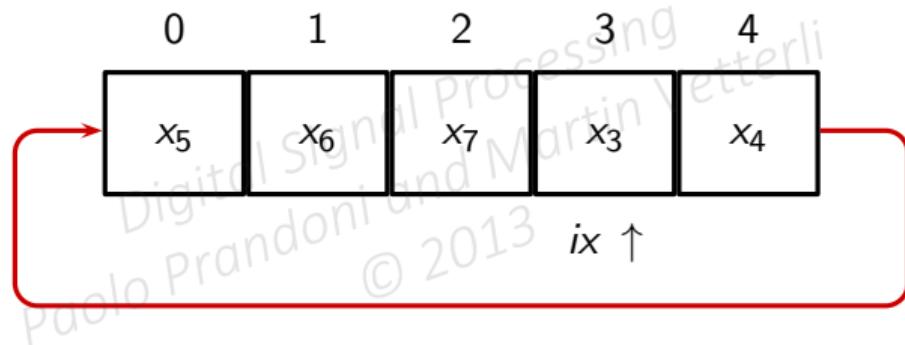
The circular buffer



The circular buffer

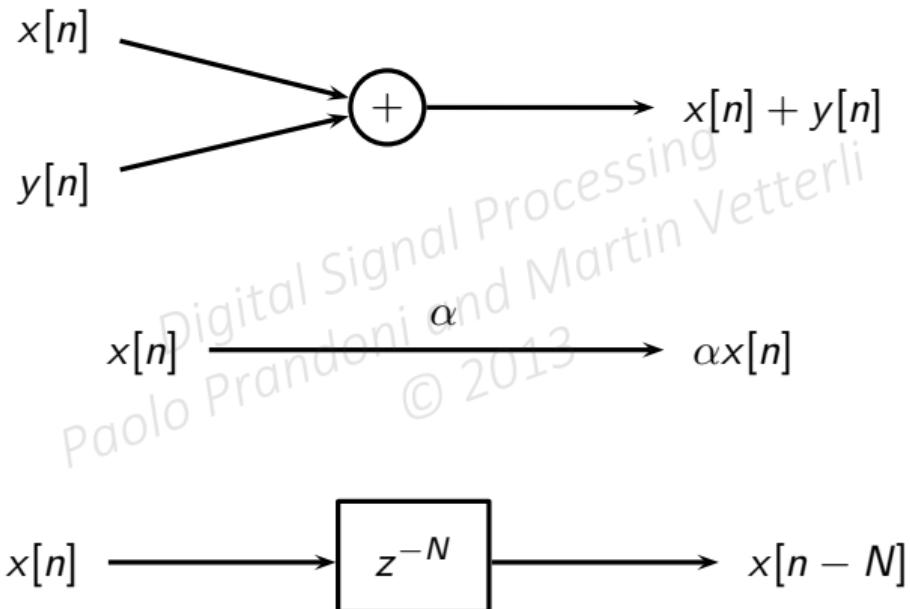


The circular buffer

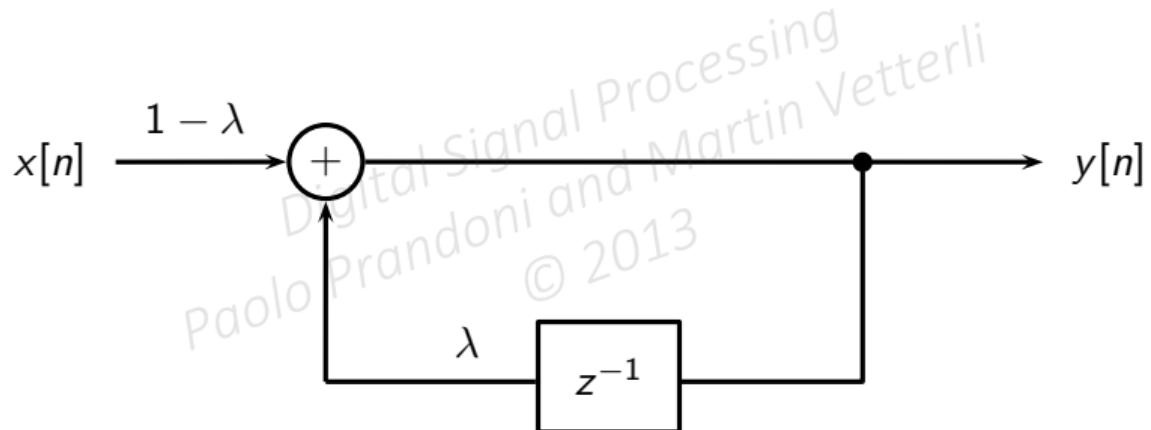


- ▶ we now need M memory cells to store previous input values
- ▶ we need to initialize the storage before first use
- ▶ we need 1 division and M additions per output sample

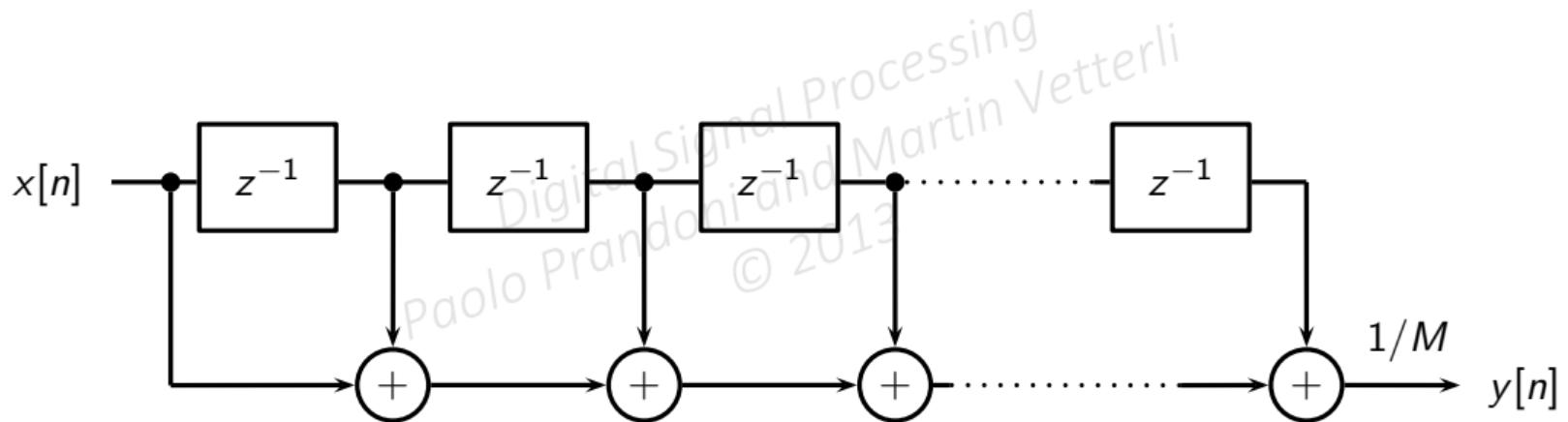
We can abstract from the implementation



$$y[n] = \lambda y[n - 1] + (1 - \lambda)x[n]$$



$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]$$



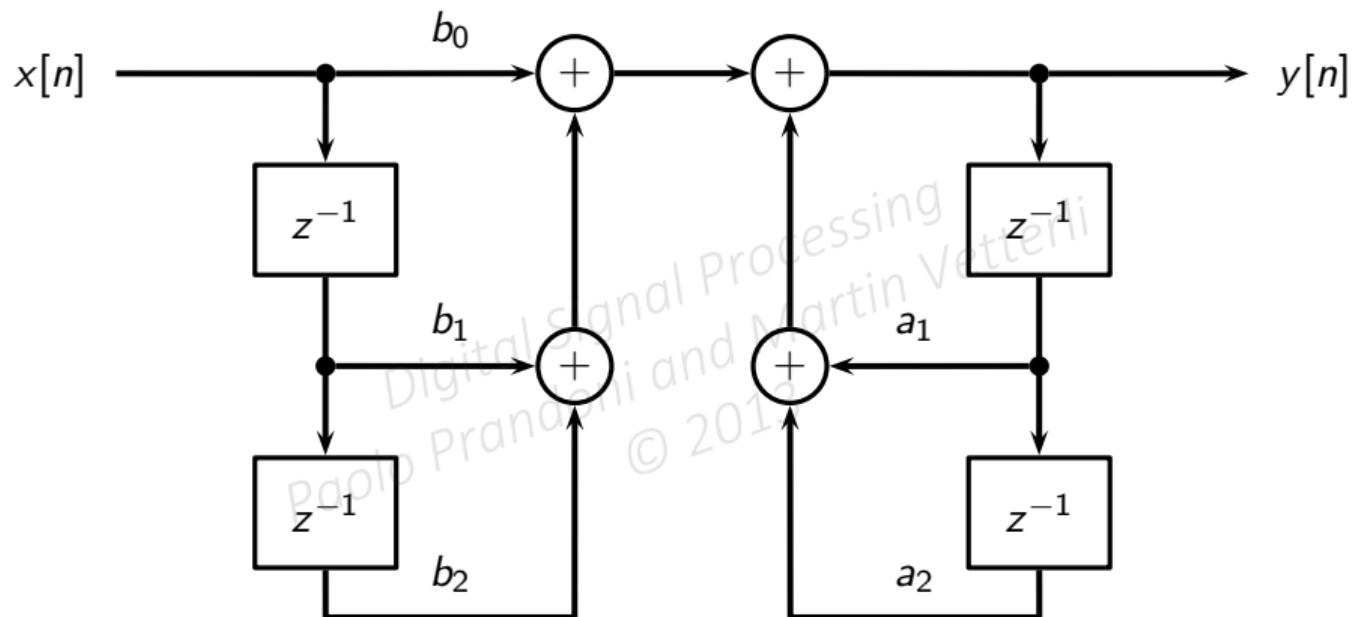
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 - a_1 z^{-1} - a_2 z^{-2}} = \frac{B(z)}{A(z)}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{B(z)}{A(z)}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

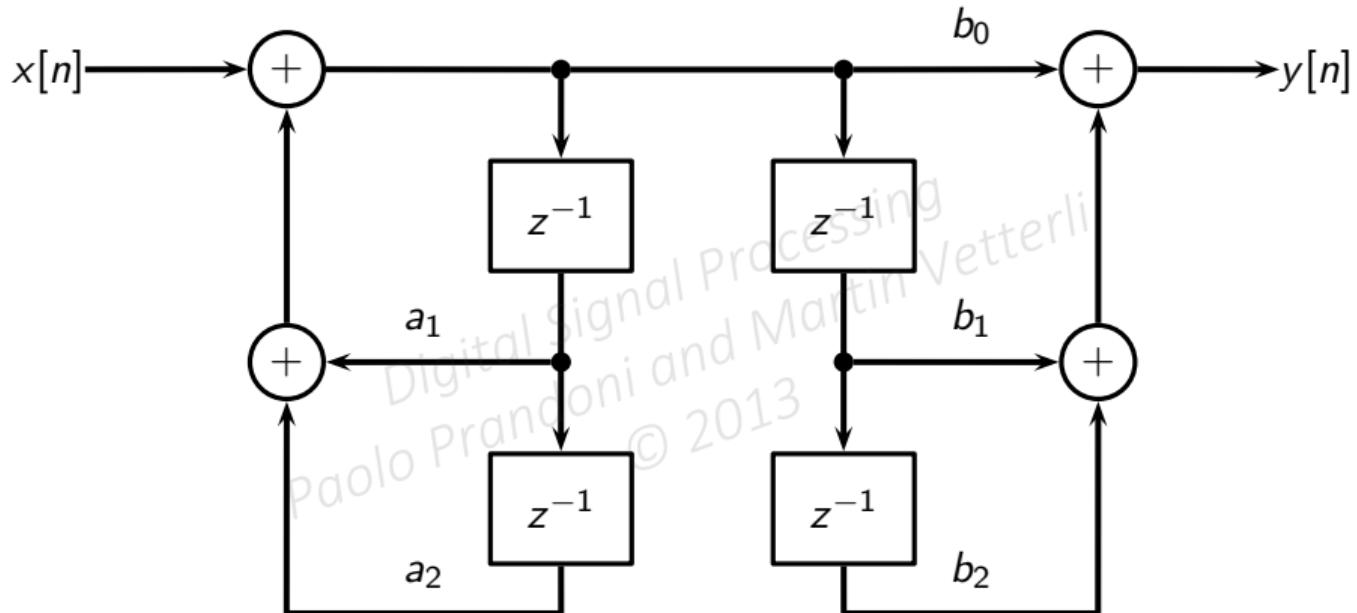
Second-order section, direct form I



$$B(z)$$

$$1/A(z)$$

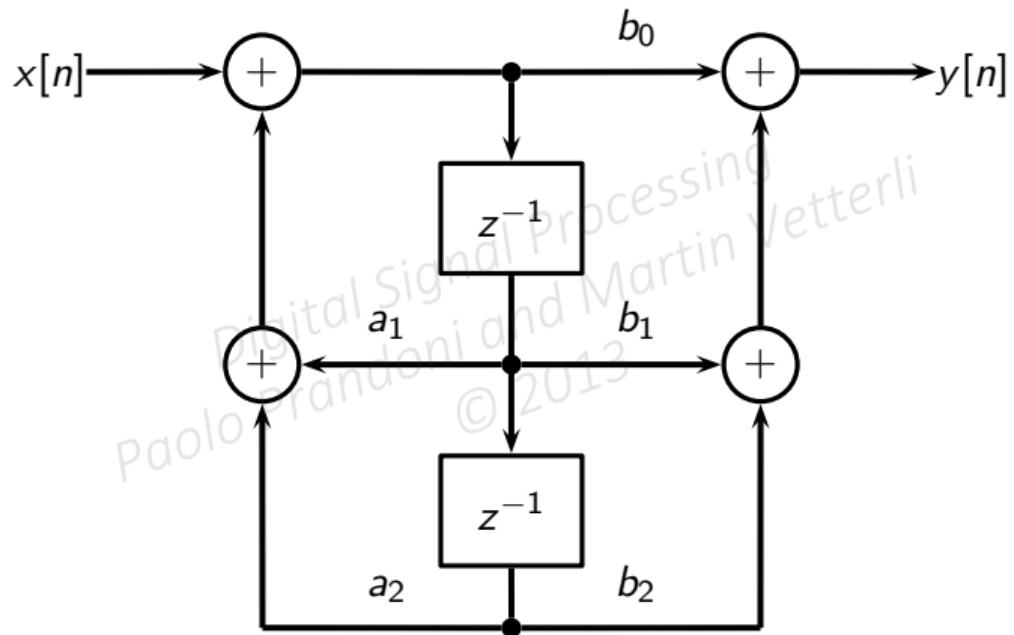
Second-order section, direct form I, inverted order



$$1/A(z)$$

$$B(z)$$

Second-order section, direct form II



If input samples arrive every T seconds,

- ▶ each output sample must be computed in at most T seconds
- ▶ number of operations becomes important (IIR vs FIR)
- ▶ some common tricks:
 - circular buffers are size 2^K (mod operation faster)
 - exploit the parallelism some processor operations (fetch and compute)

Processing unit have finite precision arithmetic:

- ▶ overflow or underflow are a real problem
- ▶ more complex structures are more resilient (but less efficient)
- ▶ some common tricks:
 - use double precision in the accumulator
 - split the filter in low-order sections
 - use floating point

END OF MODULE 5.8

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.9: Filter Design - Part II: Intuitive Filters

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ General problem
- ▶ “Intuitive” IIR design

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ many signal processing problems can be solved using simple filters
- ▶ we have seen simple lowpass filters already (Moving Average, Leaky Integrator)
- ▶ simple (low order) transfer functions allow for intuitive design and tuning

- ▶ let only low frequencies pass
- ▶ used to remove high frequency components (e.g. noise)
- ▶ useful in audio, communication, control systems
- ▶ we know a simple answer: leaky integrator

Digital Signal Processing
Prof. Joao Prandoni and Martin Vetterli
© 2013

- ▶ let only low frequencies pass
- ▶ used to remove high frequency components (e.g. noise)
- ▶ useful in audio, communication, control systems
- ▶ we know a simple answer! leaky integrator

- ▶ let only low frequencies pass
- ▶ used to remove high frequency components (e.g. noise)
- ▶ useful in audio, communication, control systems
- ▶ we know a simple answer! leaky integrator

- ▶ let only low frequencies pass
- ▶ used to remove high frequency components (e.g. noise)
- ▶ useful in audio, communication, control systems
- ▶ we know a simple answer: leaky integrator

Leaky Integrator

$$H(z) = \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

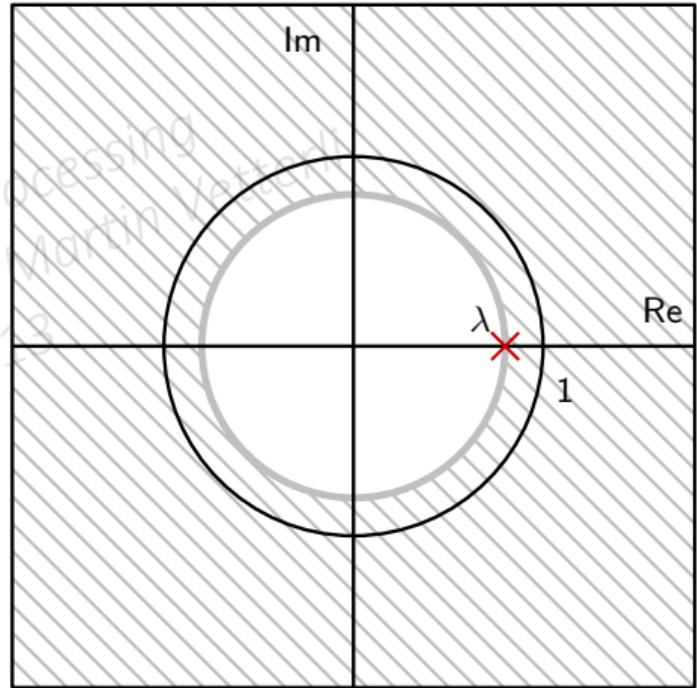
$$H(z) = \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$$

$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$

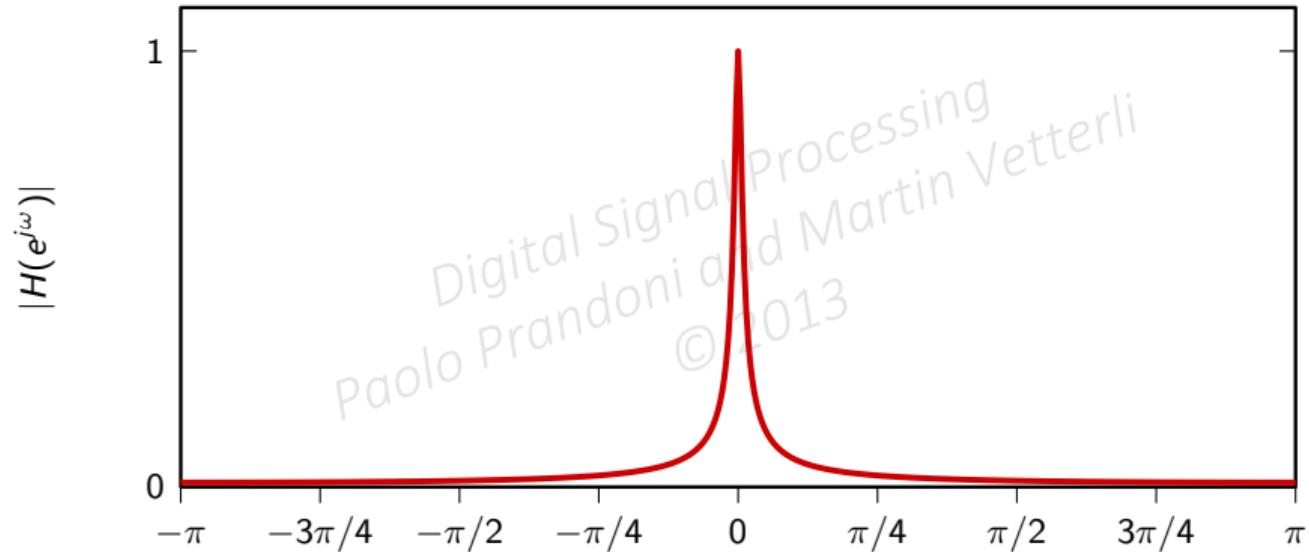
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$H(z) = \frac{(1 - \lambda)}{1 - \lambda z^{-1}}$$

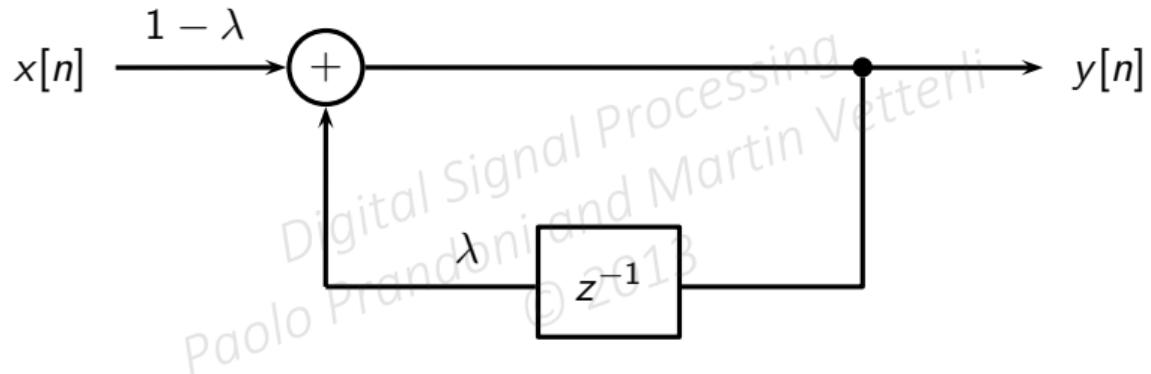
$$y[n] = (1 - \lambda)x[n] + \lambda y[n - 1]$$



Leaky Integrator, $\lambda = 0.98$



Leaky Integrator, filter structure

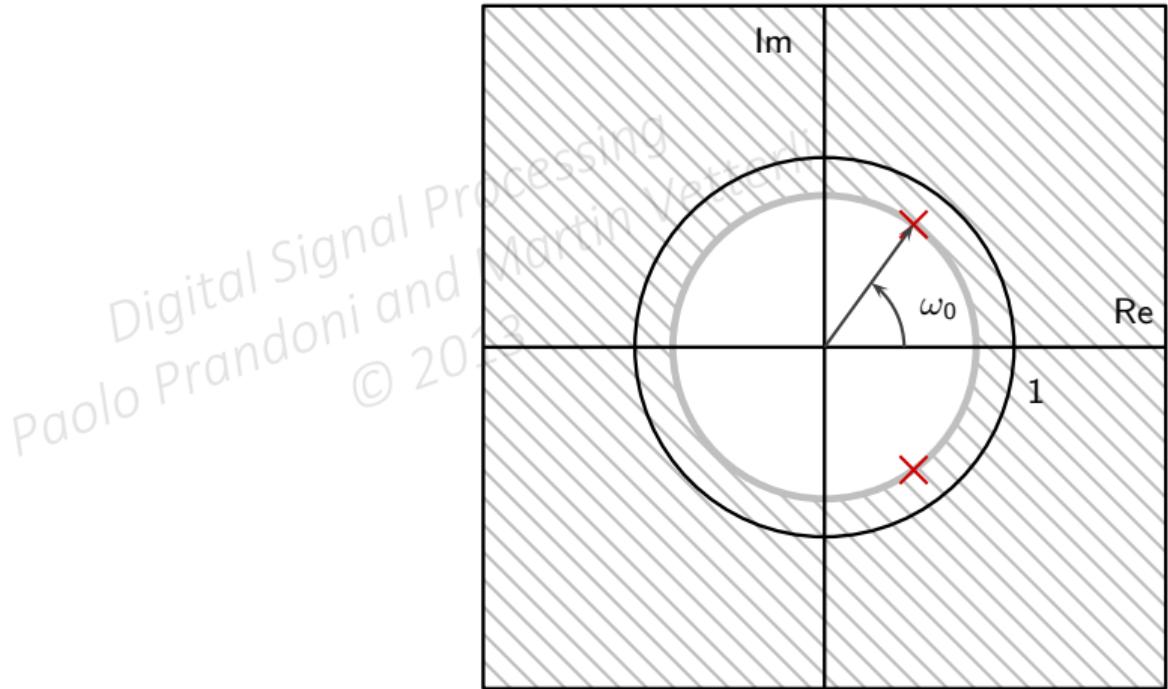


- ▶ a resonator is a narrow bandpass filter
 - ▶ used to detect the presence of a sinusoid of a given frequency
 - ▶ useful in communication systems and telephony (DTMF)
 - ▶ idea: shift the passband of the Leaky Integrator!

- ▶ a resonator is a narrow bandpass filter
- ▶ used to detect the presence of a sinusoid of a given frequency
- ▶ useful in communication systems and telephony(DTMF)
- ▶ idea: shift the passband of the Leaky Integrator!

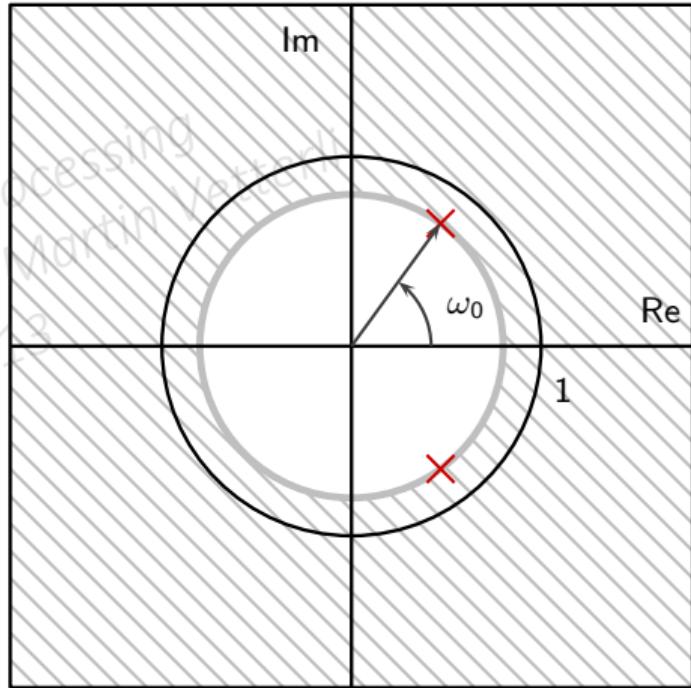
- ▶ a resonator is a narrow bandpass filter
- ▶ used to detect the presence of a sinusoid of a given frequency
- ▶ useful in communication systems and telephony (DTMF)
- ▶ idea: shift the passband of the Leaky Integrator!

- ▶ a resonator is a narrow bandpass filter
- ▶ used to detect the presence of a sinusoid of a given frequency
- ▶ useful in communication systems and telephony (DTMF)
- ▶ idea: shift the passband of the Leaky Integrator!



$$H(z) = \frac{G_0}{(1 - pz^{-1})(1 - p^*z^{-1})}$$

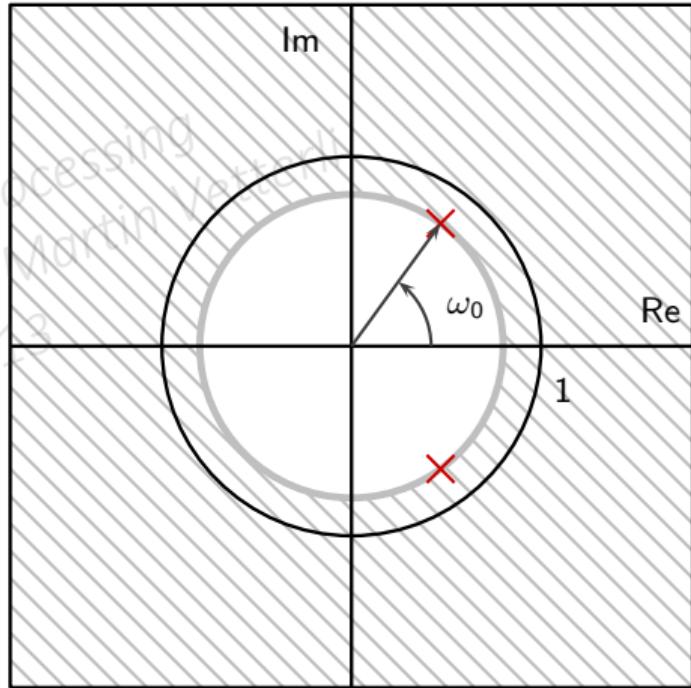
$$p = \lambda e^{j\omega_0}$$



$$H(z) = \frac{G_0}{(1 - pz^{-1})(1 - p^*z^{-1})}$$

$$p = \lambda e^{j\omega_0}$$

$$y[n] = G_0x[n] - a_1y[n-1] - a_2y[n-2]$$



$$H(z) = \frac{G_0}{(1 - pz^{-1})(1 - p^*z^{-1})}, \quad p = \lambda e^{j\omega_0}$$

$$= \frac{G_0}{1 - 2\Re\{p\}z^{-1} + |p|^2 z^{-2}}$$

Digital Signal Processing
Paolo Pandini and Martin Vetterli
© 2013

$$\frac{1}{1 - 2\cos\omega_0 z^{-1} + |\lambda|^2 z^{-2}}$$

$$a_1 = 2\lambda \cos \omega_0$$

$$a_2 = -|\lambda|^2$$

$$\begin{aligned} H(z) &= \frac{G_0}{(1 - pz^{-1})(1 - p^*z^{-1})}, \quad p = \lambda e^{j\omega_0} \\ &= \frac{G_0}{1 - 2\Re\{p\} z^{-1} + |p|^2 z^{-2}} \\ &= \frac{G_0}{1 - 2\lambda \cos \omega_0 z^{-1} + |\lambda|^2 z^{-2}} \end{aligned}$$

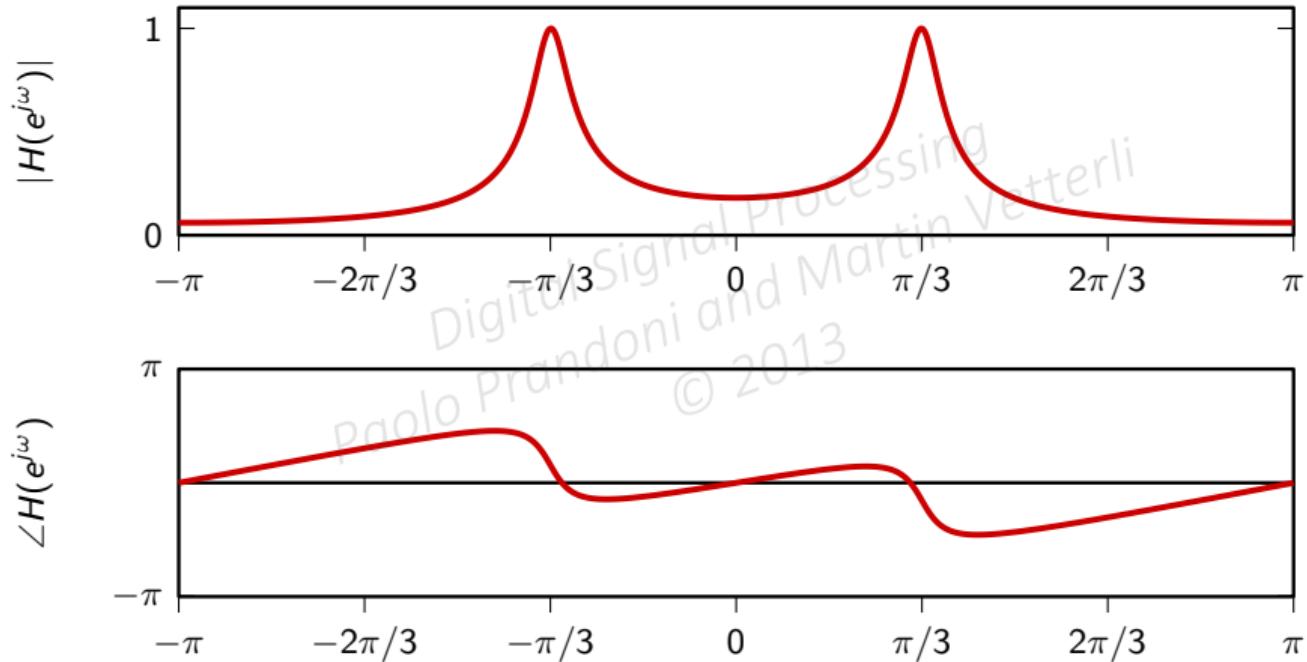
$$a_1 = 2\lambda \cos \omega_0$$

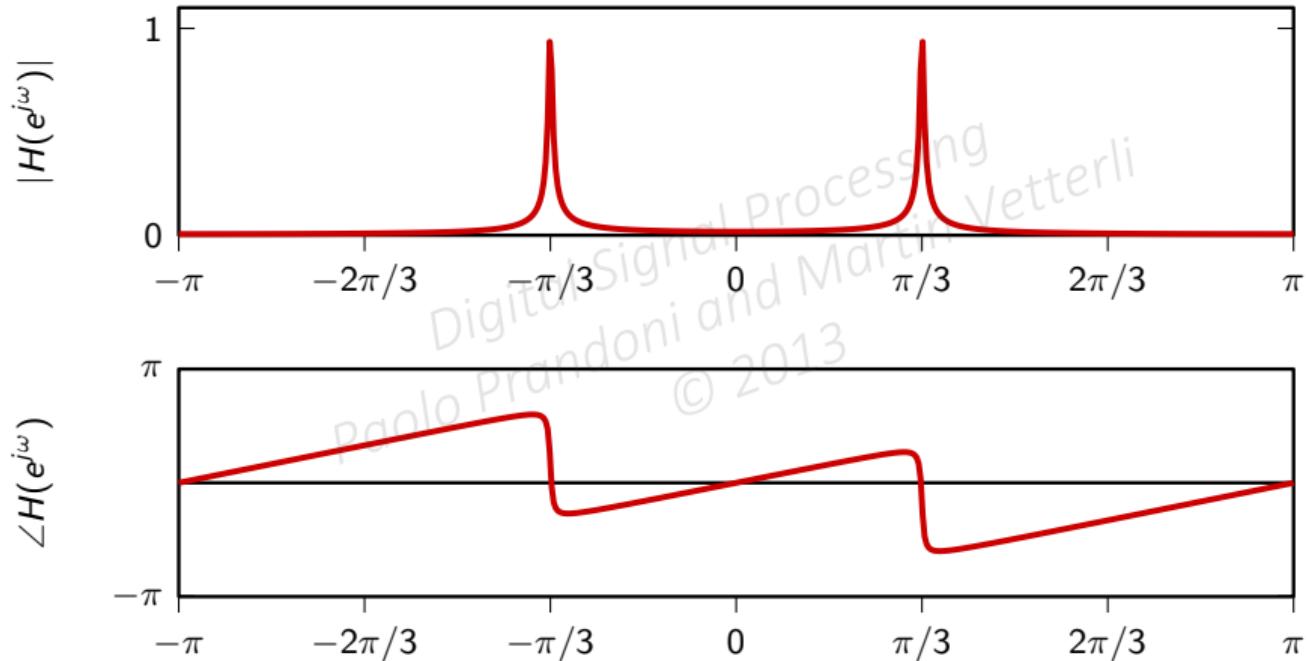
$$a_2 = -|\lambda|^2$$

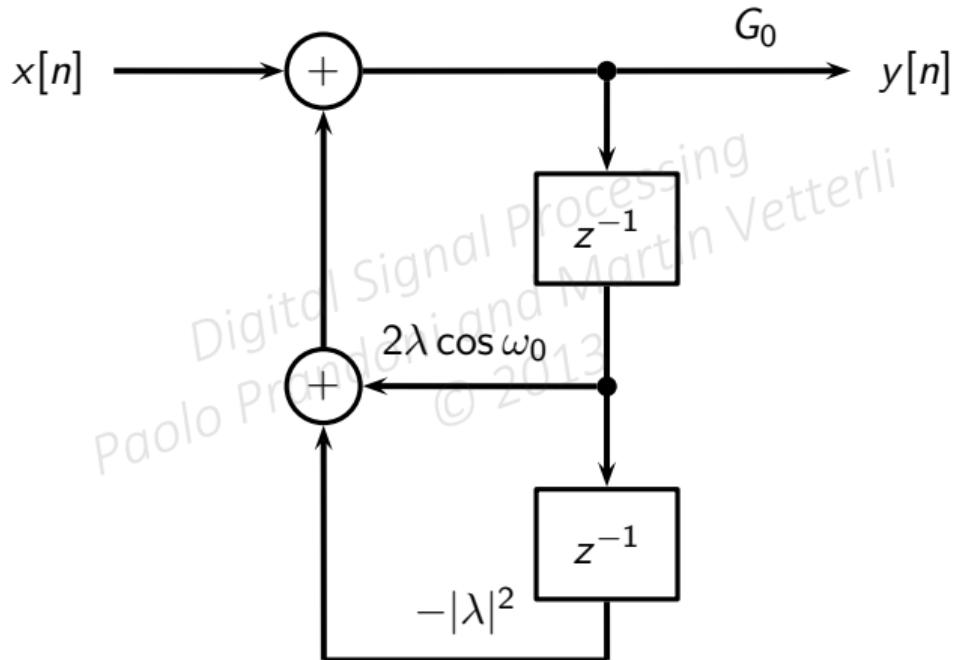
$$\begin{aligned} H(z) &= \frac{G_0}{(1 - pz^{-1})(1 - p^*z^{-1})}, \quad p = \lambda e^{j\omega_0} \\ &= \frac{G_0}{1 - 2\Re\{p\} z^{-1} + |p|^2 z^{-2}} \\ &= \frac{G_0}{1 - 2\lambda \cos \omega_0 z^{-1} + |\lambda|^2 z^{-2}} \end{aligned}$$

$$a_1 = 2\lambda \cos \omega_0$$

$$a_2 = -|\lambda|^2$$





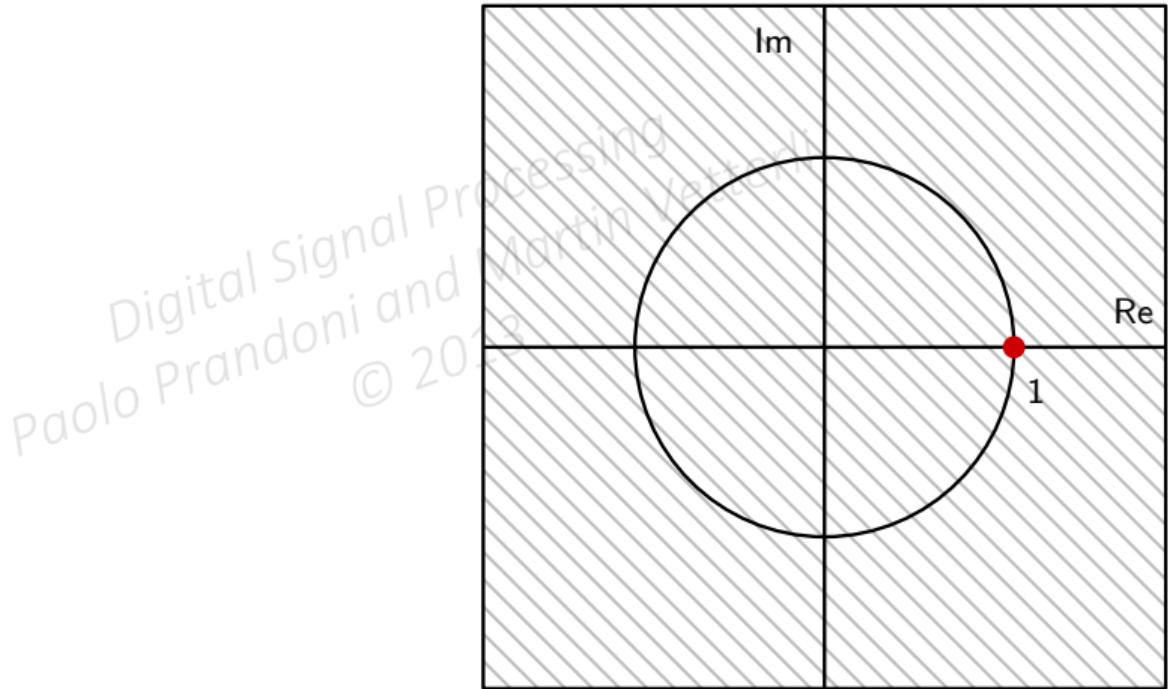


- ▶ a DC-balanced signal has zero sum: $\lim_{N \rightarrow \infty} \sum_{n=-N}^N x[n] = 0$
i.e. there is no Direct Current component
- ▶ its DTFT value at zero is zero
- ▶ we want to remove the DC bias from a non zero-centered signal
- ▶ we want to kill the frequency component at $\omega = 0$

- ▶ a DC-balanced signal has zero sum: $\lim_{N \rightarrow \infty} \sum_{n=-N}^N x[n] = 0$
i.e. there is no Direct Current component
- ▶ its DTFT value at zero is zero
- ▶ we want to remove the DC bias from a non zero-centered signal
- ▶ we want to kill the frequency component at $\omega = 0$

- ▶ a DC-balanced signal has zero sum: $\lim_{N \rightarrow \infty} \sum_{n=-N}^N x[n] = 0$
i.e. there is no Direct Current component
- ▶ its DTFT value at zero is zero
- ▶ we want to remove the DC bias from a non zero-centered signal
- ▶ we want to kill the frequency component at $\omega = 0$

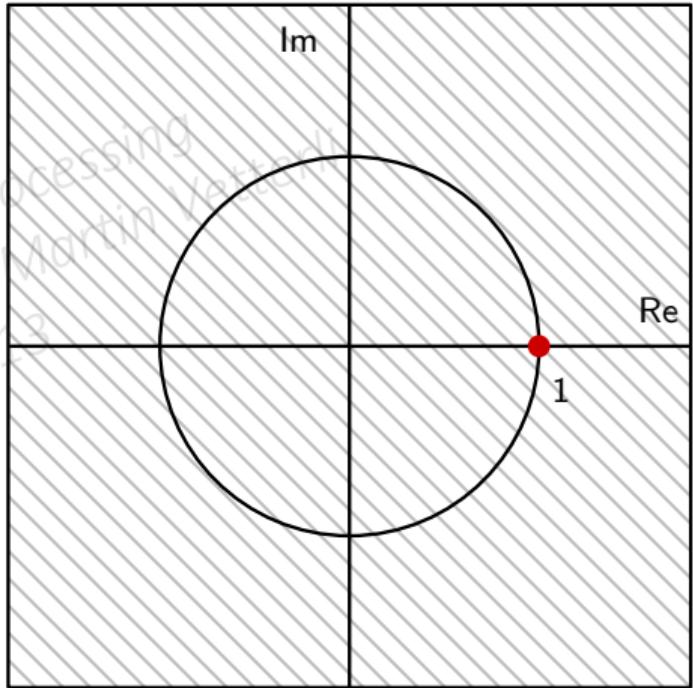
- ▶ a DC-balanced signal has zero sum: $\lim_{N \rightarrow \infty} \sum_{n=-N}^N x[n] = 0$
i.e. there is no Direct Current component
- ▶ its DTFT value at zero is zero
- ▶ we want to remove the DC bias from a non zero-centered signal
- ▶ we want to kill the frequency component at $\omega = 0$



Digital Signal Processing
Martin Vetterli
Paolo Prandoni and
© 2004

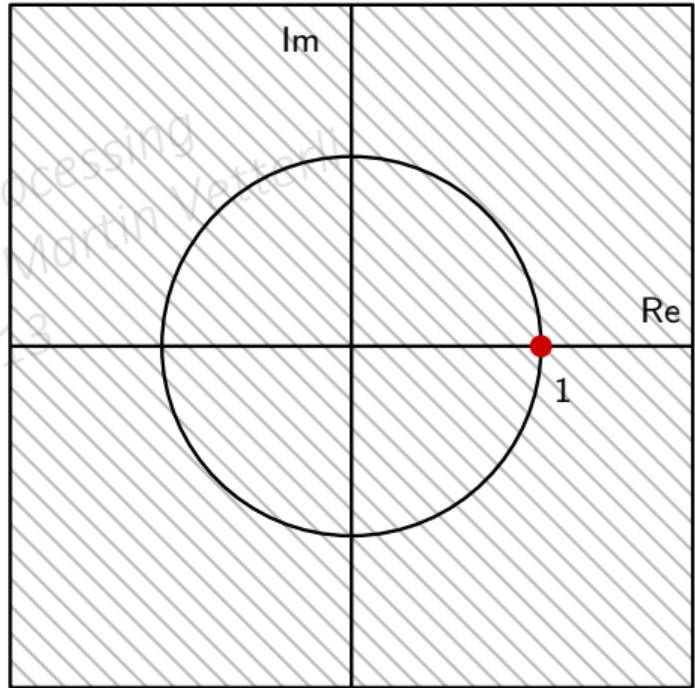
$$H(z) = 1 - z^{-1}$$

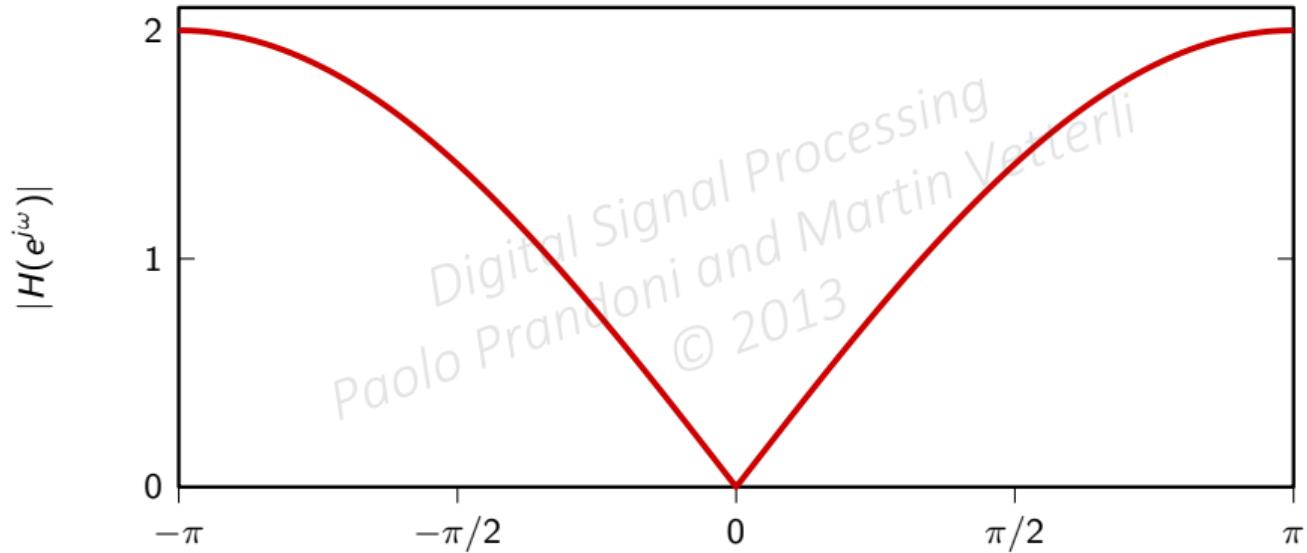
Digital Signal Processing
Martin Vetterli
Paolo Prandoni and
© 2004



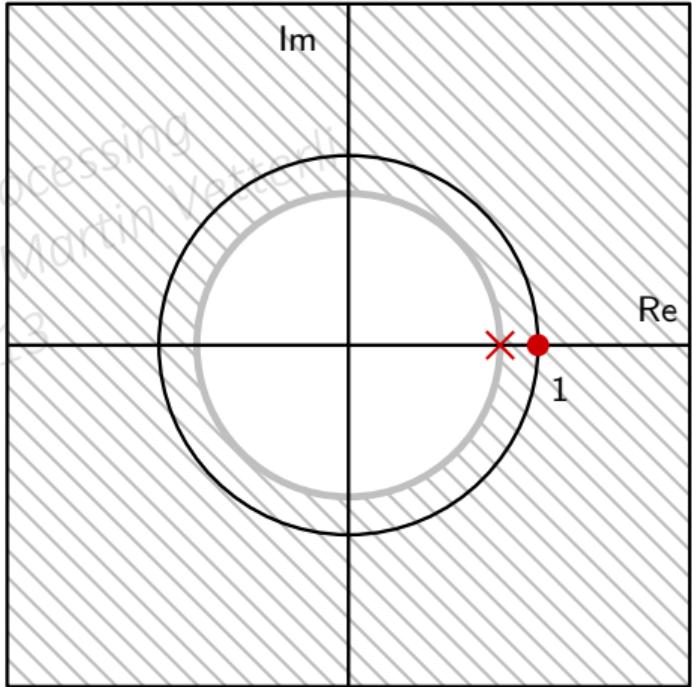
$$H(z) = 1 - z^{-1}$$

$$y[n] = x[n] - x[n - 1]$$





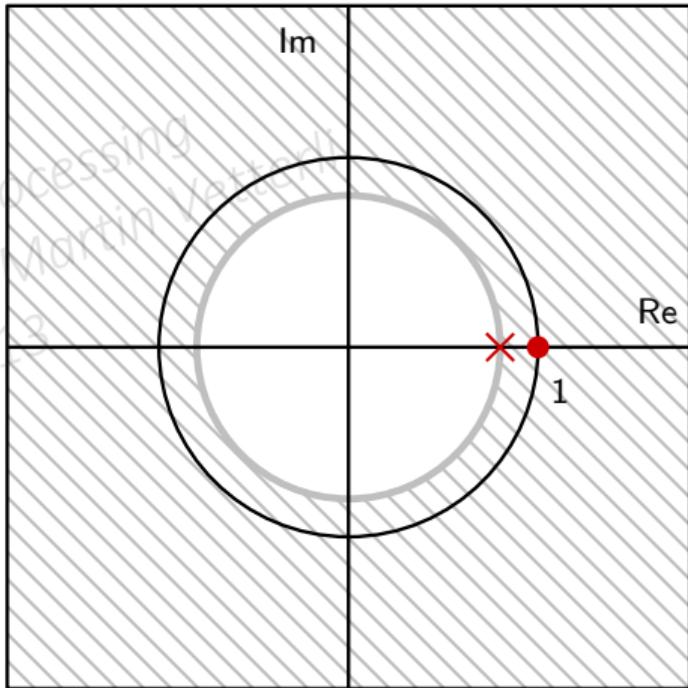
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2004



DC removal, improved

$$H(z) = \frac{1 - z^{-1}}{1 - \lambda z^{-1}}$$

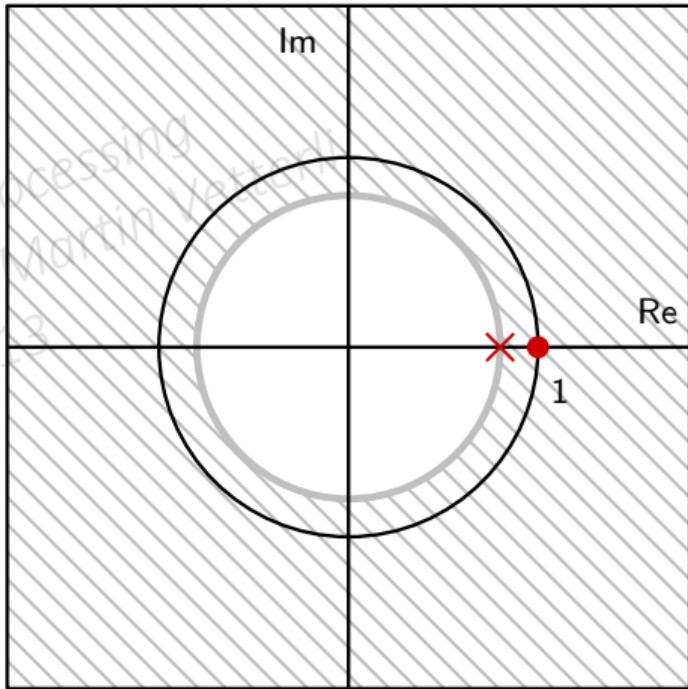
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2014

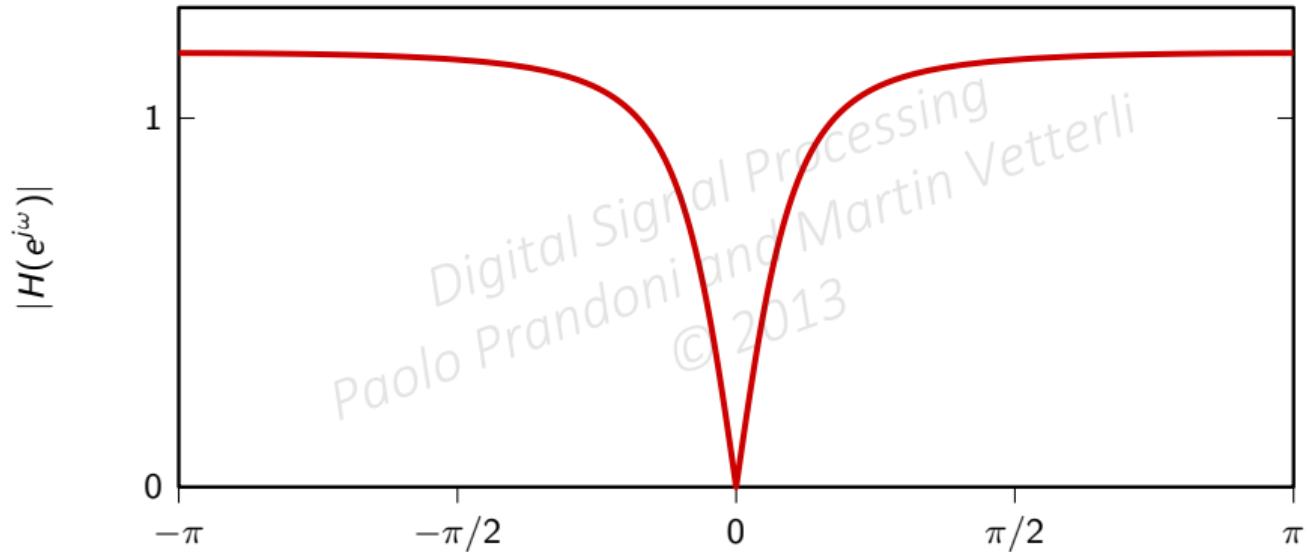


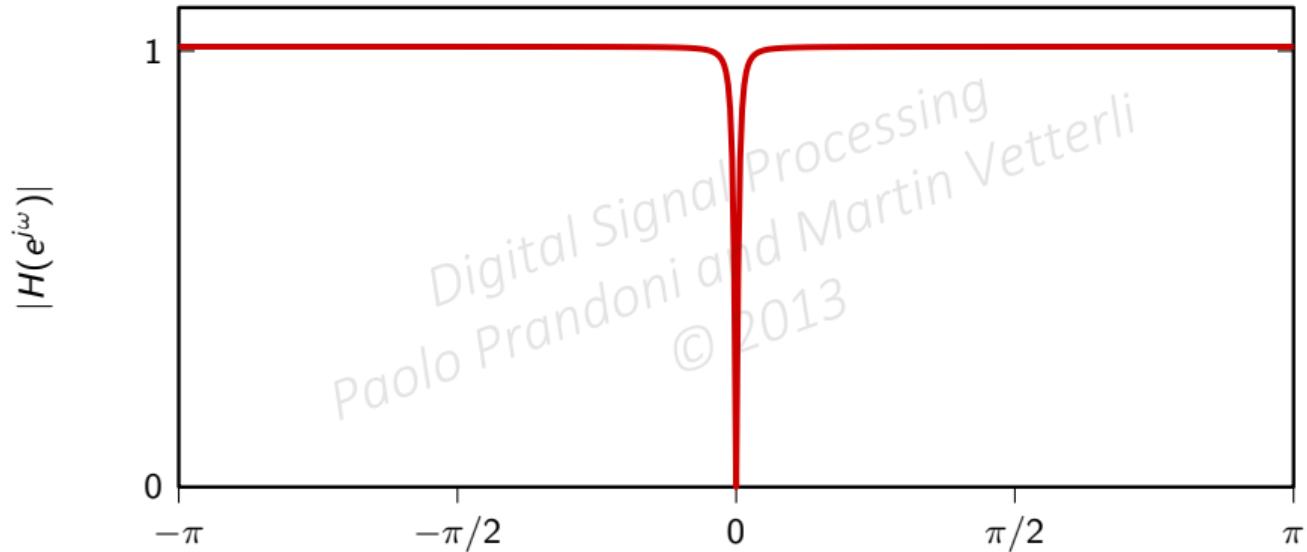
DC removal, improved

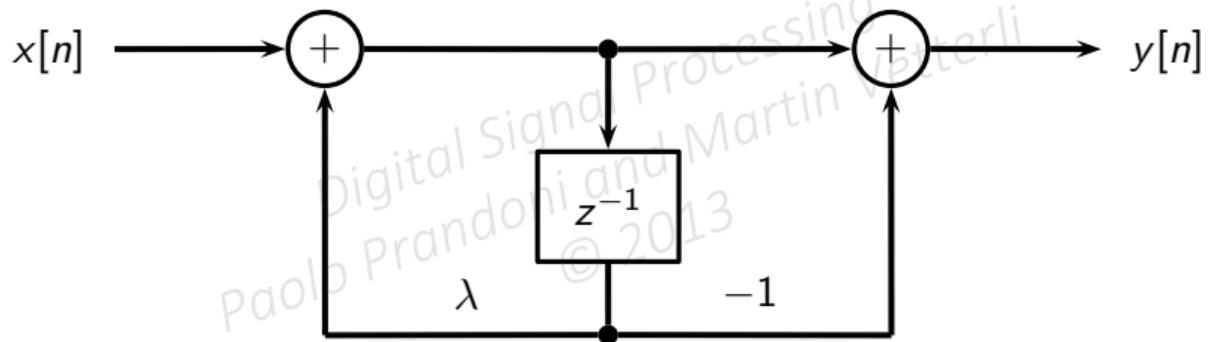
$$H(z) = \frac{1 - z^{-1}}{1 - \lambda z^{-1}}$$

$$y[n] = \lambda y[n-1] + x[n] - x[n-1]$$





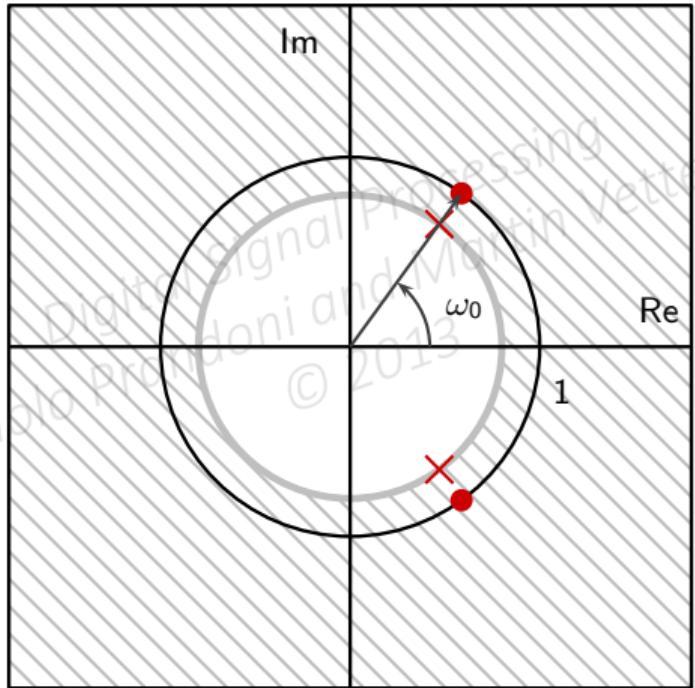




- ▶ similar to DC removal but we want to remove a specific nonzero frequency
- ▶ very useful for musicians: amplifiers for electric guitars pick up the hum from the electric mains (50Hz in Europe and 60Hz in North America)
- ▶ we need to tune the hum removal according to country

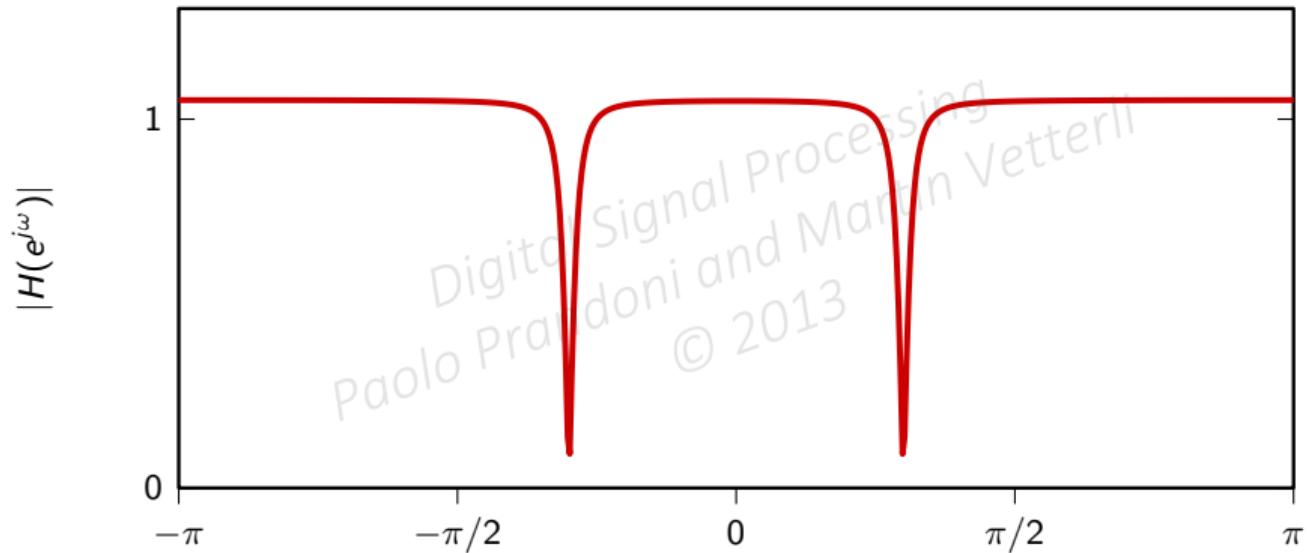
- ▶ similar to DC removal but we want to remove a specific nonzero frequency
- ▶ very useful for musicians: amplifiers for electric guitars pick up the hum from the electric mains (50Hz in Europe and 60Hz in North America)
- ▶ we need to tune the hum removal according to country

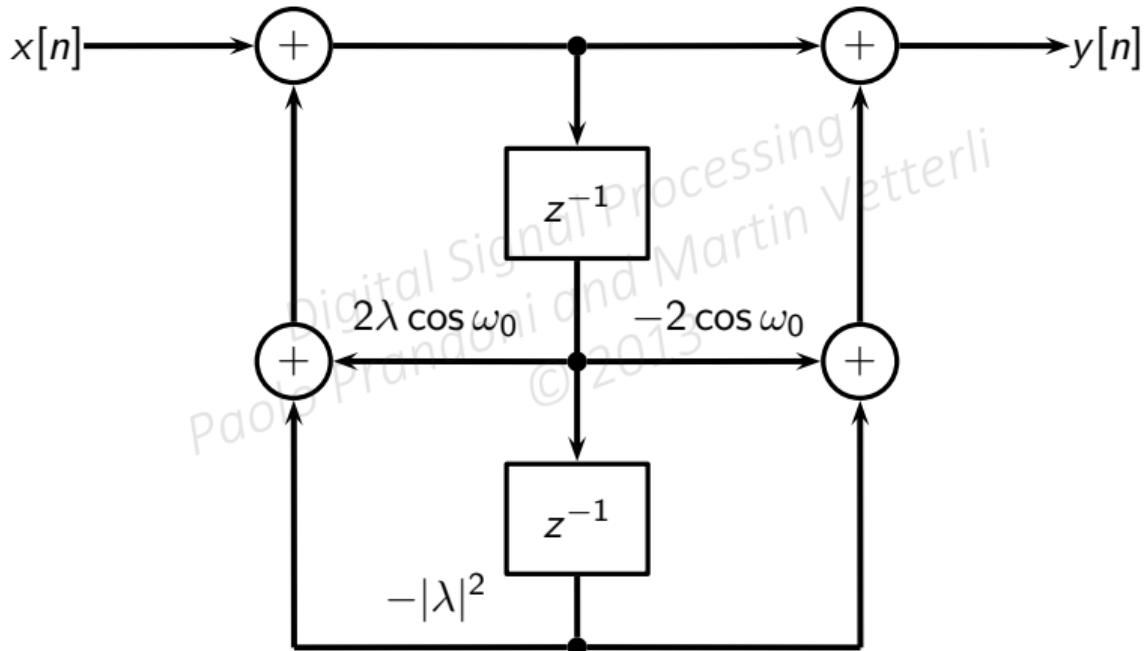
- ▶ similar to DC removal but we want to remove a specific nonzero frequency
- ▶ very useful for musicians: amplifiers for electric guitars pick up the hum from the electric mains (50Hz in Europe and 60Hz in North America)
- ▶ we need to tune the hum removal according to country



$$H(z) = \frac{(1 - e^{j\omega_0} z^{-1})(1 - e^{-j\omega_0} z^{-1})}{(1 - \lambda e^{j\omega_0} z^{-1})(1 - \lambda e^{-j\omega_0} z^{-1})}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013





END OF MODULE 5.9

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.10: Filter Design - Part III: Design from Specs

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter specifications
- ▶ IIR design
- ▶ FIR design

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter specifications
- ▶ IIR design
- ▶ FIR design

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ Filter specifications
- ▶ IIR design
- ▶ FIR design

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

The filter design problem

You are given a set of requirements:

- ▶ frequency response: passband(s) and stopband(s)
- ▶ phase: overall delay, linearity
- ▶ some limit on computational resources and/or numerical precision

You must determine N , M , a_k 's and b_k 's in

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_{M-1} z^{-(M-1)}}{a_0 + a_1 z^{-1} + \dots + a_{N-1} z^{-(N-1)}}$$

in order to best fulfill the requirements

The filter design problem

You are given a set of requirements:

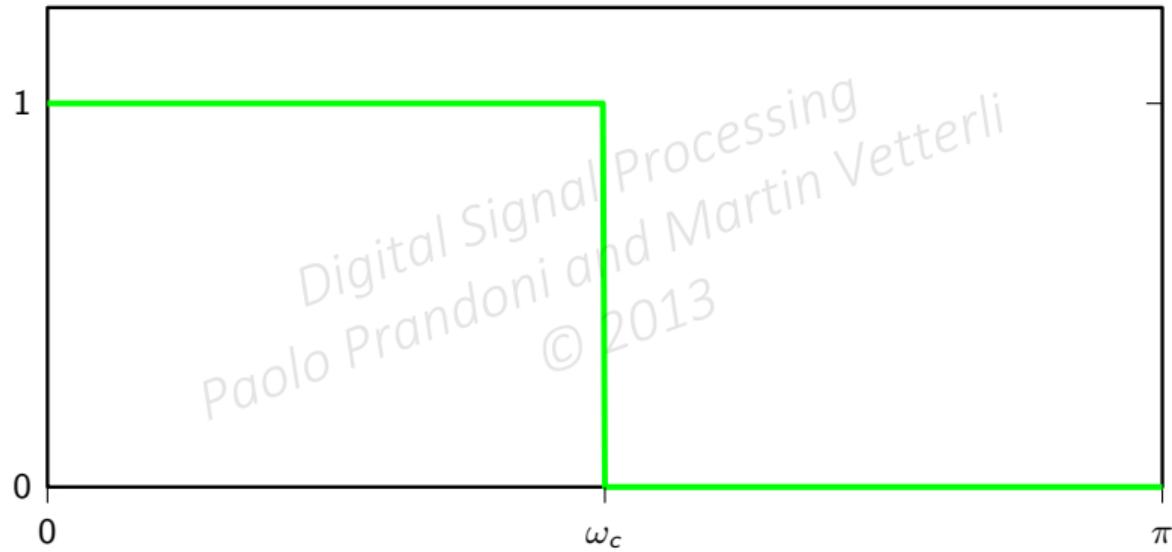
- ▶ frequency response: passband(s) and stopband(s)
- ▶ phase: overall delay, linearity
- ▶ some limit on computational resources and/or numerical precision

You must determine N , M , a_k 's and b_k 's in

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_{M-1} z^{-(M-1)}}{a_0 + a_1 z^{-1} + \dots + a_{N-1} z^{-(N-1)}}$$

in order to best fulfill the requirements

Example: lowpass specs



- ▶ passband/stopband transitions cannot be infinitely sharp
⇒ use *transition bands*
- ▶ magnitude response cannot be constant over an interval
⇒ specify magnitude *tolerances* over the interval
- ▶ in general:
 - smaller transition bands ⇒ higher filter order
 - smaller error tolerances ⇒ higher filter order
 - higher filter order ⇒ more expensive, larger delay

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ passband/stopband transitions cannot be infinitely sharp
⇒ use *transition bands*
- ▶ magnitude response cannot be constant over an interval
⇒ specify magnitude *tolerances* over the interval
- ▶ in general:
 - smaller transition bands ⇒ higher filter order
 - smaller error tolerances ⇒ higher filter order
 - higher filter order ⇒ more expensive, larger delay

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ passband/stopband transitions cannot be infinitely sharp
⇒ use *transition bands*
- ▶ magnitude response cannot be constant over an interval
⇒ specify magnitude *tolerances over bands*
- ▶ in general:
 - smaller transition bands ⇒ higher filter order
 - smaller error tolerances ⇒ higher filter order
 - higher filter order ⇒ more expensive, larger delay

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

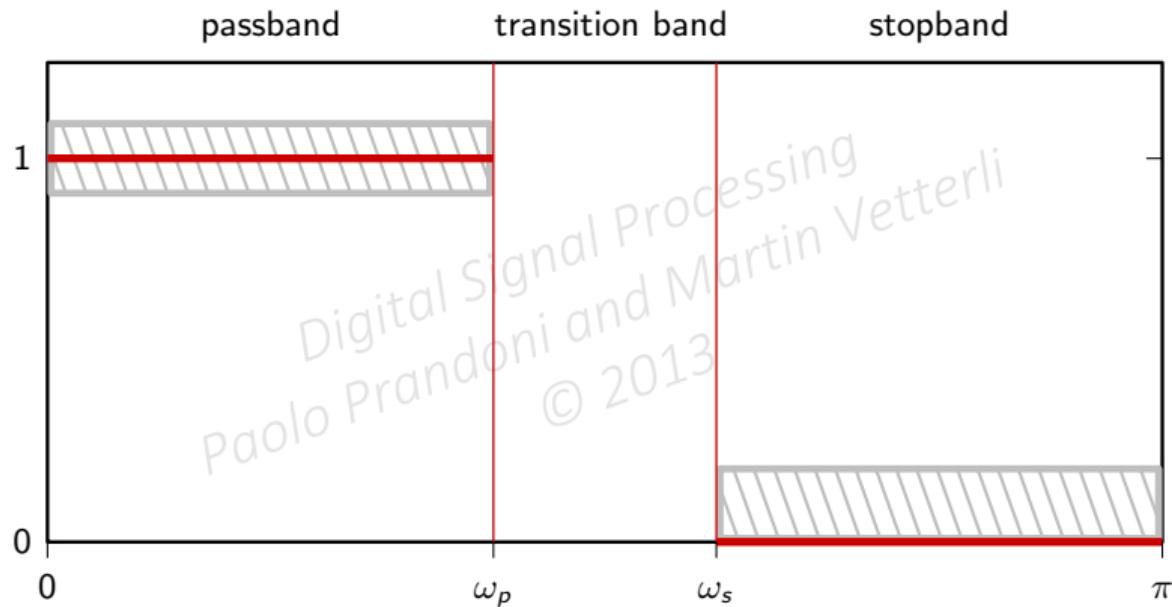
- ▶ passband/stopband transitions cannot be infinitely sharp
⇒ use *transition bands*
- ▶ magnitude response cannot be constant over an interval
⇒ specify magnitude *tolerances over bands*
- ▶ in general:
 - smaller transition bands ⇒ higher filter order
 - smaller error tolerances ⇒ higher filter order
 - higher filter order ⇒ more expensive, larger delay

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ passband/stopband transitions cannot be infinitely sharp
⇒ use *transition bands*
- ▶ magnitude response cannot be constant over an interval
⇒ specify magnitude *tolerances over bands*
- ▶ in general:
 - smaller transition bands ⇒ higher filter order
 - smaller error tolerances ⇒ higher filter order
 - higher filter order ⇒ more expensive, larger delay

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Example: realistic lowpass specs



Why we can't have a flat response

$$H(z) = B(z)/A(z), \quad \text{with } A \text{ and } B \text{ polynomials}$$

*Digital Signal Processing
Paolo Prati
© 2013*

$H(e^{j\omega}) = c$ over an interval $\Rightarrow B(z) - cA(z) = 0$ over an interval
 $\Rightarrow B(z) - cA(z)$ has an infinite number of roots
 $\Rightarrow B(z) - cA(z) = 0$ for all values of z
 $\Rightarrow H(e^{j\omega}) = c$ over the entire $[-\pi, \pi]$ interval.

Why we can't have a flat response

$$H(z) = B(z)/A(z), \quad \text{with } A \text{ and } B \text{ polynomials}$$

$$H(e^{j\omega}) = c \text{ over an interval} \Rightarrow B(z) - cA(z) = 0 \text{ over an interval}$$

$$\begin{aligned} &\Rightarrow B(z) - cA(z) \text{ has an infinite number of roots} \\ &\Rightarrow B(z) - cA(z) = 0 \text{ for all values of } z \\ &\Rightarrow H(e^{j\omega}) = c \text{ over the entire } [-\pi, \pi] \text{ interval.} \end{aligned}$$

$$H(z) = B(z)/A(z), \quad \text{with } A \text{ and } B \text{ polynomials}$$

- $H(e^{j\omega}) = c$ over an interval $\Rightarrow B(z) - cA(z) = 0$ over an interval
 $\Rightarrow B(z) - cA(z)$ has an infinite number of roots
 $\Rightarrow B(z) - cA(z) = 0$ for all values of z
 $\Rightarrow H(e^{j\omega}) = c$ over the entire $[-\pi, \pi]$ interval.

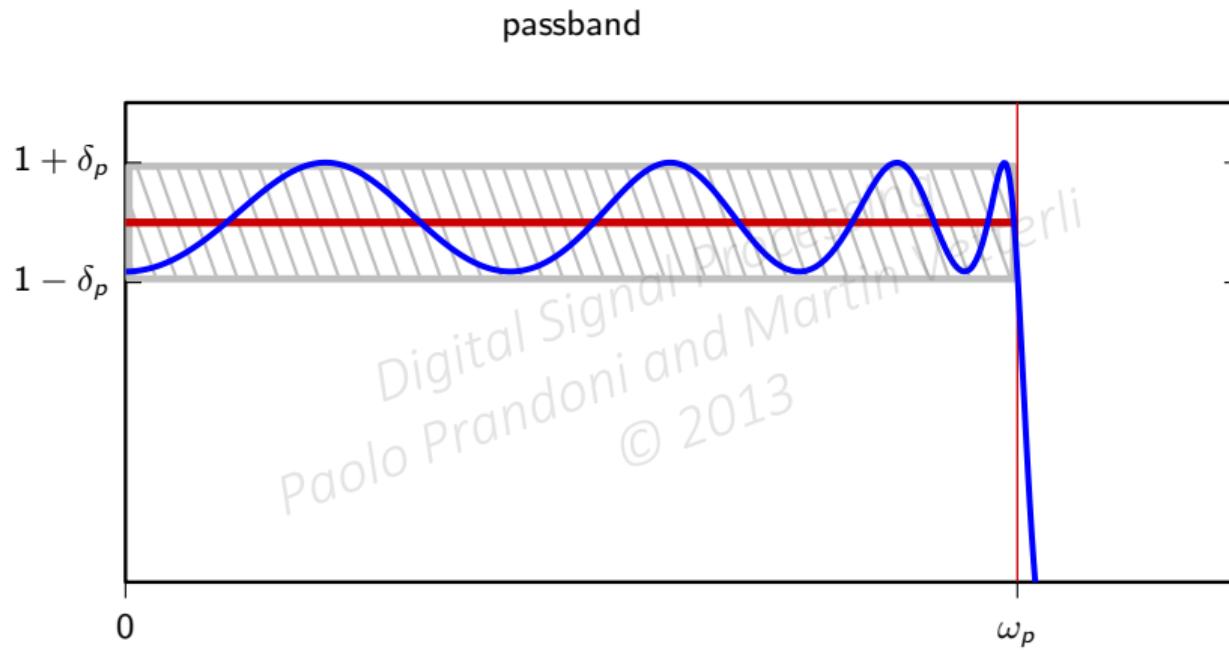
$$H(z) = B(z)/A(z), \quad \text{with } A \text{ and } B \text{ polynomials}$$

- $H(e^{j\omega}) = c$ over an interval
- $\Rightarrow B(z) - cA(z) = 0$ over an interval
 - $\Rightarrow B(z) - cA(z)$ has an infinite number of roots
 - $\Rightarrow B(z) - cA(z) = 0$ for all values of z
 - $\Rightarrow H(e^{j\omega}) = c$ over the entire $[-\pi, \pi]$ interval.

$$H(z) = B(z)/A(z), \quad \text{with } A \text{ and } B \text{ polynomials}$$

- $H(e^{j\omega}) = c$ over an interval
- $\Rightarrow B(z) - cA(z) = 0$ over an interval
 - $\Rightarrow B(z) - cA(z)$ has an infinite number of roots
 - $\Rightarrow B(z) - cA(z) = 0$ for all values of z
 - $\Rightarrow H(e^{j\omega}) = c$ over the entire $[-\pi, \pi]$ interval.

Important case: equiripple error



- ▶ IIR or FIR?
- ▶ how to determine the coefficients?
- ▶ how to evaluate the performance?

Digital Signal Processing
Paolo Frasconi and Martin Vetterli
© 2013

Pros:

- ▶ computationally efficient
- ▶ strong attenuation easy
- ▶ good for audio

Cons:

- ▶ stability issues
- ▶ difficult to design for arbitrary response
- ▶ nonlinear phase

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Pros:

- ▶ always stable
- ▶ optimal design techniques exist
- ▶ can be designed with linear phase

Cons:

- ▶ computationally much more expensive
- ▶ may “sound” harsh

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ finding N , M , a_k 's and b_k 's from specs is a hard nonlinear problem
- ▶ established methods:
 - IIR: conversion of analog design
 - FIR: optimal minimax filter design

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ finding N , M , a_k 's and b_k 's from specs is a hard nonlinear problem
- ▶ established methods:
 - IIR: conversion of analog design
 - FIR: optimal minimax filter design

- ▶ finding N , M , a_k 's and b_k 's from specs is a hard nonlinear problem
- ▶ established methods:
 - IIR: conversion of analog design
 - FIR: optimal minimax filter design

- ▶ finding N , M , a_k 's and b_k 's from specs is a hard nonlinear problem
- ▶ established methods:
 - IIR: conversion of analog design
 - FIR: optimal minimax filter design

Filter design was an established art long before digital processing appeared

- ▶ lots of nice analog filters exist
- ▶ methods exist to “translate” the analog design into a rational transfer function
- ▶ most numerical packages (Matlab, etc.) provide ready-made routines
- ▶ design involves specifying some parameters and testing that the specs are fulfilled

Digital Signal Processing
Polo P. Pandori and Martin Vetterli
© 2013

Filter design was an established art long before digital processing appeared

- ▶ lots of nice analog filters exist
- ▶ methods exist to “translate” the analog design into a rational transfer function
- ▶ most numerical packages (Matlab, etc.) provide ready-made routines
- ▶ design involves specifying some parameters and testing that the specs are fulfilled

Filter design was an established art long before digital processing appeared

- ▶ lots of nice analog filters exist
- ▶ methods exist to “translate” the analog design into a rational transfer function
- ▶ most numerical packages (Matlab, etc.) provide ready-made routines
- ▶ design involves specifying some parameters and testing that the specs are fulfilled

Filter design was an established art long before digital processing appeared

- ▶ lots of nice analog filters exist
- ▶ methods exist to “translate” the analog design into a rational transfer function
- ▶ most numerical packages (Matlab, etc.) provide ready-made routines
- ▶ design involves specifying some parameters and testing that the specs are fulfilled

Magnitude response:

- ▶ maximally flat
- ▶ monotonic over $[0, \pi]$

Design parameters:

- ▶ order N
- ▶ cutoff frequency

Digital Signal Processing
Test values.
Paolo Prandoni and Martin Vetterli
width of transition band
© 2013
passband error

Magnitude response:

- ▶ maximally flat
- ▶ monotonic over $[0, \pi]$

Design parameters:

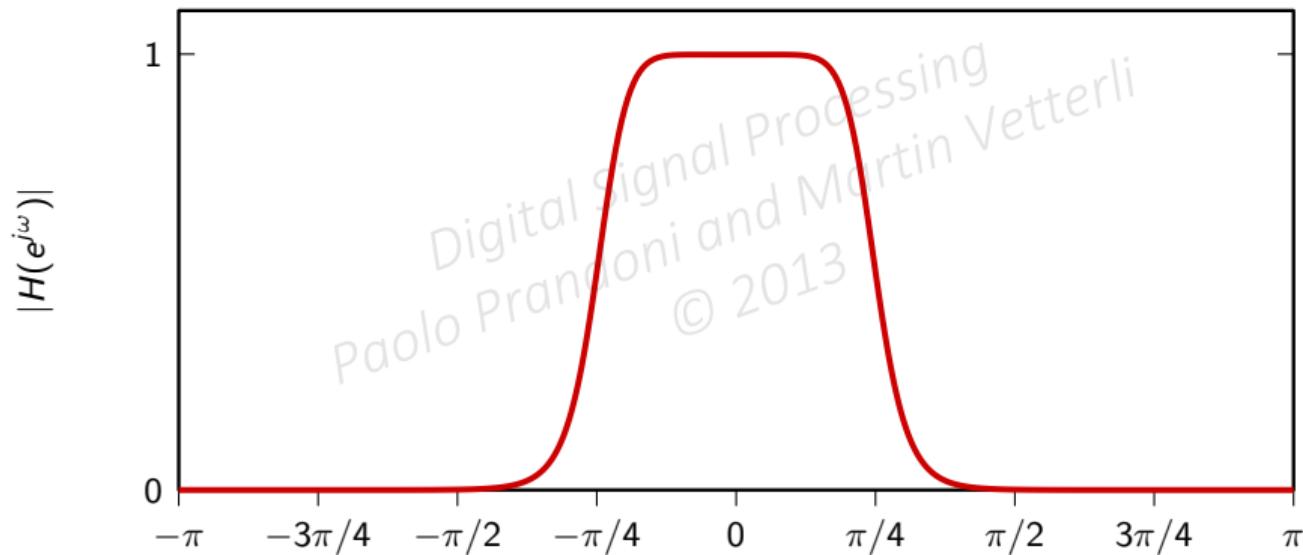
- ▶ order N
- ▶ cutoff frequency

Test values:

- ▶ width of transition band
- ▶ passband error

Butterworth lowpass example

$$N = 4, \omega_c = \pi/4$$



Magnitude response:

- ▶ equiripple in passband
- ▶ monotonic in stopband

Design parameters:

- ▶ order N
- ▶ passband max error
- ▶ cutoff frequency

▶ width of transition band

▶ stopband error

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
Test values:
© 2013

Magnitude response:

- ▶ equiripple in passband
- ▶ monotonic in stopband

Design parameters:

- ▶ order N
- ▶ passband max error
- ▶ cutoff frequency

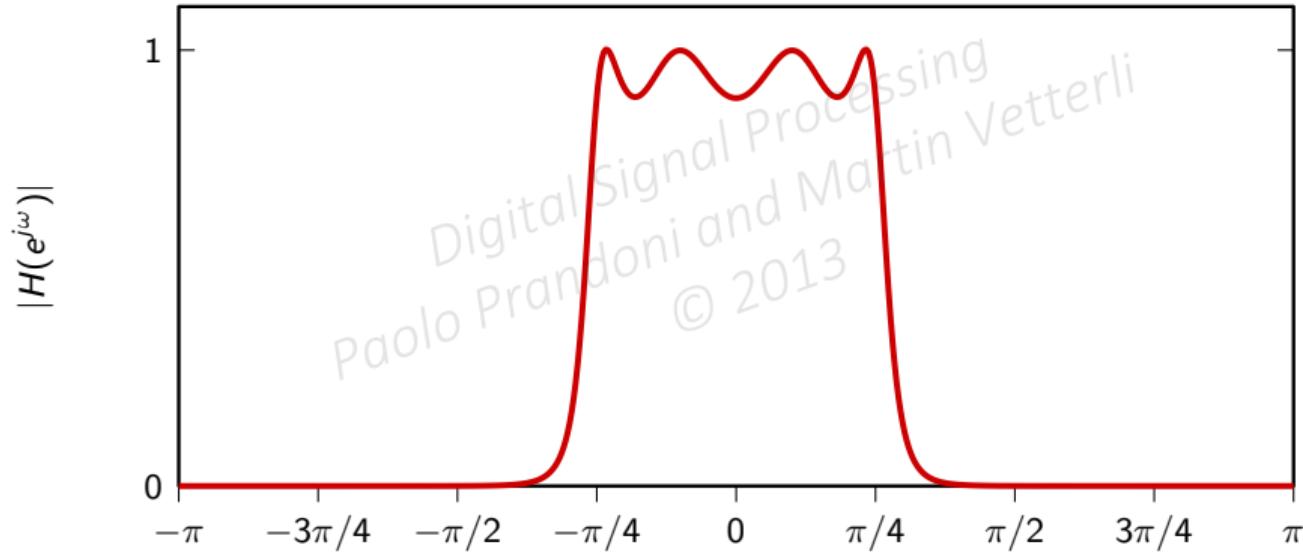
Test values:

- ▶ width of transition band
- ▶ stopband error

Digital Signal Processing
Paolo Prandoni and Marta Vetterli
© 2013

Chebyshev lowpass example

$$N = 4, \omega_c = \pi/4, e_{\max} = 12\%$$



Magnitude response:

- ▶ equiripple in passband and stopband

Design parameters:

- ▶ order N
- ▶ cutoff frequency
- ▶ passband max error
- ▶ stopband min attenuation

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013 width of transition band

Magnitude response:

- ▶ equiripple in passband and stopband

Design parameters:

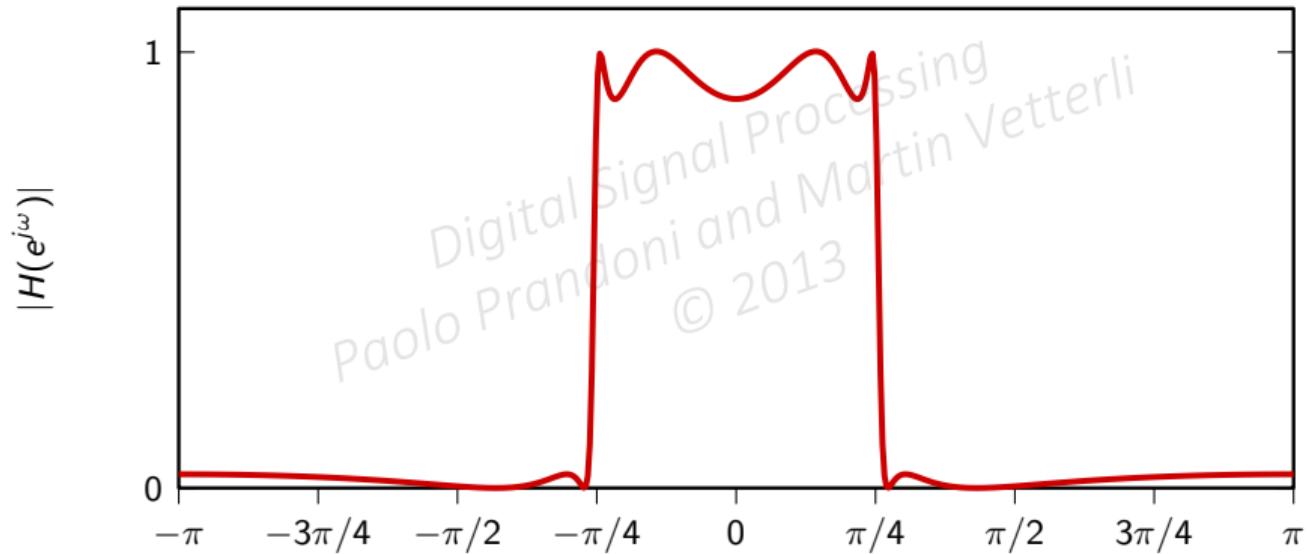
- ▶ order N
- ▶ cutoff frequency
- ▶ passband max error
- ▶ stopband min attenuation

Test value:

- ▶ width of transition band

Elliptic lowpass example

$$N = 4, \omega_c = \pi/4, e_{\max} = 12\%, \text{att}_{\min} = 0.03$$



FIR filters are a digital signal processing “exclusivity”.

In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

- ▶ linear phase
- ▶ equiripple error in passband and stopband

algorithm proceeds by **minimizing the maximum error** in passband and stopband

FIR filters are a digital signal processing “exclusivity”.

In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

- ▶ linear phase
- ▶ equiripple error in passband and stopband

algorithm proceeds by **minimizing the maximum error** in passband and stopband

FIR filters are a digital signal processing “exclusivity”.

In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

- ▶ linear phase
- ▶ equiripple error in passband and stopband

algorithm proceeds by **minimizing the maximum error** in passband and stopband

FIR filters are a digital signal processing “exclusivity”.

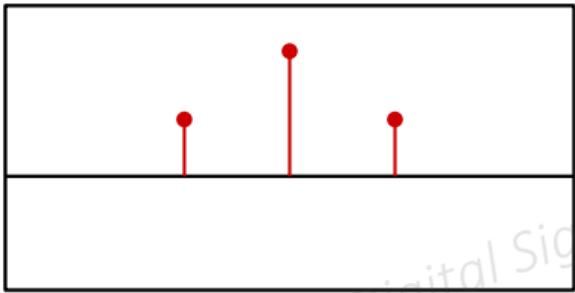
In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

- ▶ linear phase
- ▶ equiripple error in passband and stopband

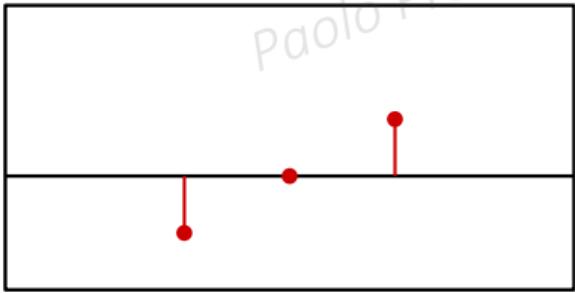
algorithm proceeds by **minimizing** the **maximum** error in passband and stopband

Linear phase derives from a symmetric or antisymmetric impulse response

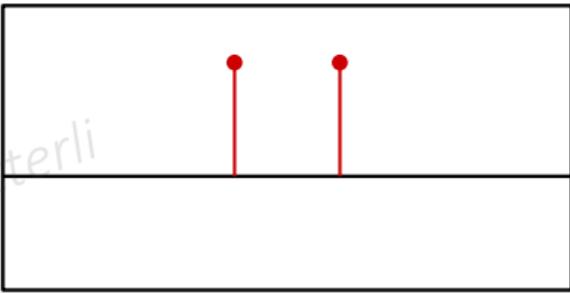
Type I



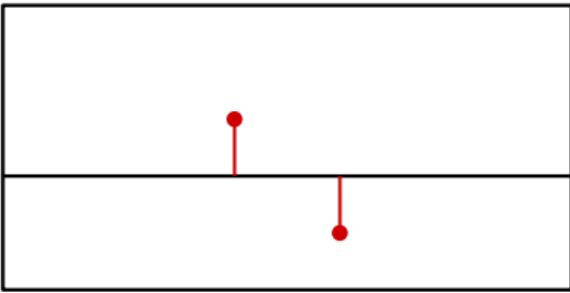
Type III



Type II



Type IV



$$h[C + n] = h[C - n]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $h[n] = h[-n]$
 $\odot 2^{13}$

$$H(z) = z^{-C} H'(z)$$

Linear phase (Type I)

$$h[C + n] = h[C - n]$$

$$h'[n] = h[n + C]$$

$$H(z) = z^{-C} H'(z)$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
 $h[n] \bar{\equiv} h'[m]$
© 2013

Linear phase (Type I)

$$h[C + n] = h[C - n]$$

$$h'[n] = h[n + C]$$

$$h'[n] = h'[-n]$$

$$H(z) = z^{-C} H'(z)$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$h[C + n] = h[C - n]$$

$$h'[n] = h[n + C]$$

$$h'[n] = h'[-n]$$

$$H(z) = z^{-C} H'(z)$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$H'(z) = \sum_{n=-M}^M h'[n]z^{-n}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

$$= h'[0] + \sum_{n=1}^M h'[n](z^{-n} + z^n)$$

$$H'(e^{j\omega}) = h'[0] + \sum_{n=1}^M h'[n](e^{j\omega n} + e^{-j\omega n})$$

$$= h'[0] + 2 \sum_{n=1}^M h'[n] \cos \omega n \quad \in \mathbb{R}$$

$$H'(z) = \sum_{n=-M}^M h'[n]z^{-n}$$

$$= h'[0] + \sum_{n=1}^M h'[n](z^n + z^{-n})$$

$$H'(e^{j\omega}) = h'[0] + \sum_{n=1}^M h'[n](e^{j\omega n} + e^{-j\omega n})$$

$$= h'[0] + 2 \sum_{n=1}^M h'[n] \cos \omega n \quad \in \mathbb{R}$$

$$H'(z) = \sum_{n=-M}^M h'[n]z^{-n}$$

$$= h'[0] + \sum_{n=1}^M h'[n](z^n + z^{-n})$$

$$H'(e^{j\omega}) = h'[0] + \sum_{n=1}^M h'[n](e^{j\omega n} + e^{-j\omega n})$$

$$= h'[0] + 2 \sum_{n=1}^M h'[n] \cos \omega n \quad \in \mathbb{R}$$

$$H'(z) = \sum_{n=-M}^M h'[n]z^{-n}$$

$$= h'[0] + \sum_{n=1}^M h'[n](z^n + z^{-n})$$

$$H'(e^{j\omega}) = h'[0] + \sum_{n=1}^M h'[n](e^{j\omega n} + e^{-j\omega n})$$

$$= h'[0] + 2 \sum_{n=1}^M h'[n] \cos \omega n \quad \in \mathbb{R}$$

$$H(e^{j\omega}) = \left[h[C] + 2 \sum_{n=1}^M h[n+C] \cos n\omega \right] e^{-j\omega C}$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Magnitude response:

- ▶ equiripple in passband and stopband

Design parameters:

- ▶ order N (number of taps)
 - ▶ passband edge ω_p
 - ▶ stopband edge ω_s
 - ▶ ratio of passband to stopband error δ_p/δ_s
- Test value:
© 2013 Paolo Prandoni and Marton Vetterli

Magnitude response:

- ▶ equiripple in passband and stopband

Design parameters:

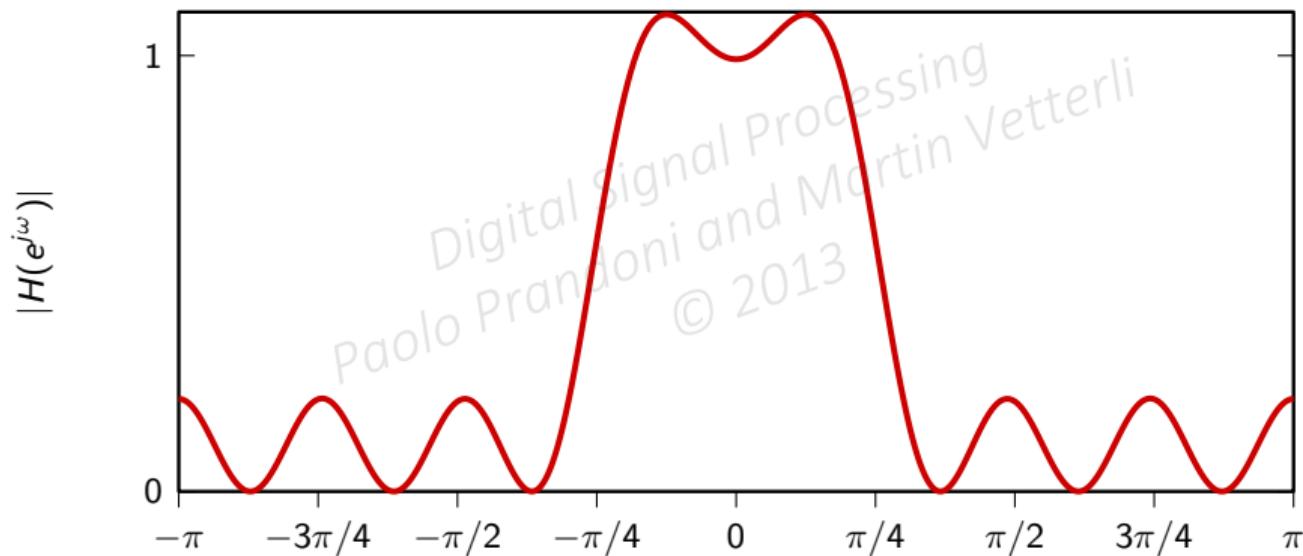
- ▶ order N (number of taps)
- ▶ passband edge ω_p
- ▶ stopband edge ω_s
- ▶ ratio of passband to stopband error δ_p/δ_s

Test value:

- ▶ passband max error
- ▶ stopband max error

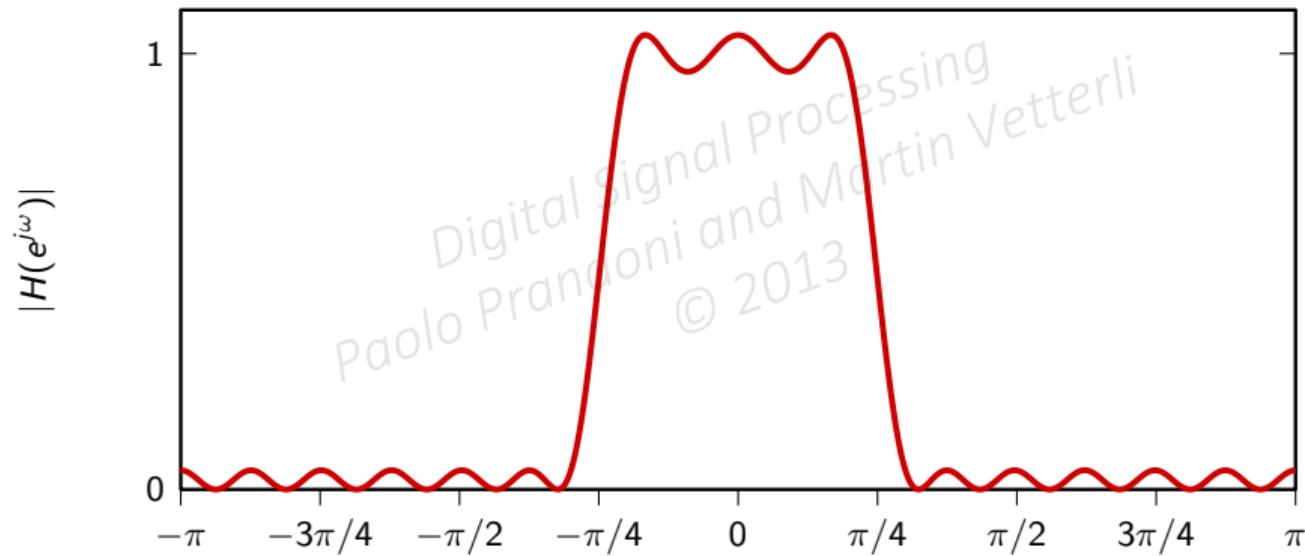
Minimax lowpass example

$$N = 9, \omega_s = 0.2\pi, \omega_p = 0.3\pi, \delta_p/\delta_s = 10$$



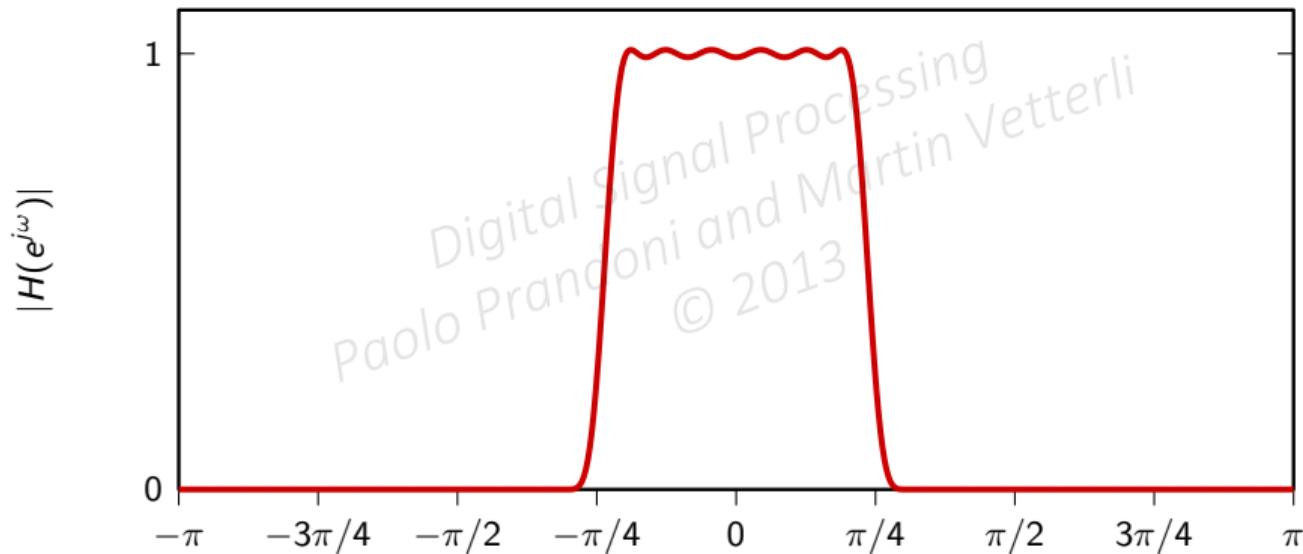
Minimax lowpass example

$$N = 19, \omega_s = 0.2\pi, \omega_p = 0.3\pi, \delta_p/\delta_s = 10$$



Minimax lowpass example

$$N = 51, \omega_s = 0.2\pi, \omega_p = 0.3\pi, \delta_p/\delta_s = 1$$

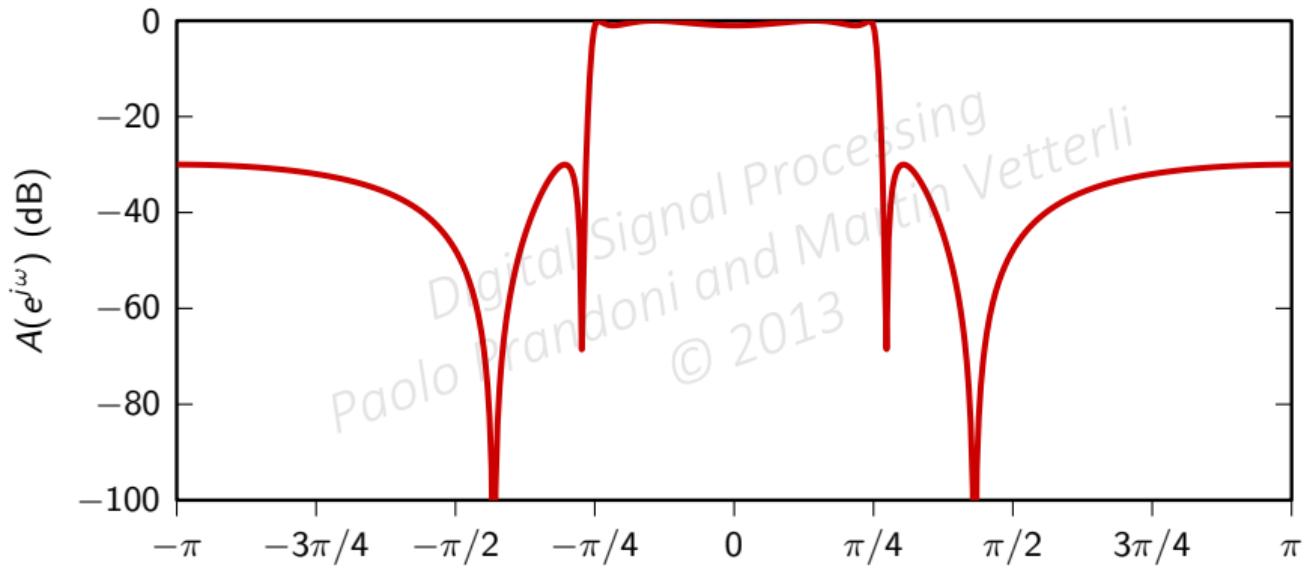


- ▶ filter max passband magnitude G
- ▶ filter attenuation expressed in decibels as:

$$A_{\text{dB}} = 20 \log_{10}(|H(e^{j\omega})|/G)$$

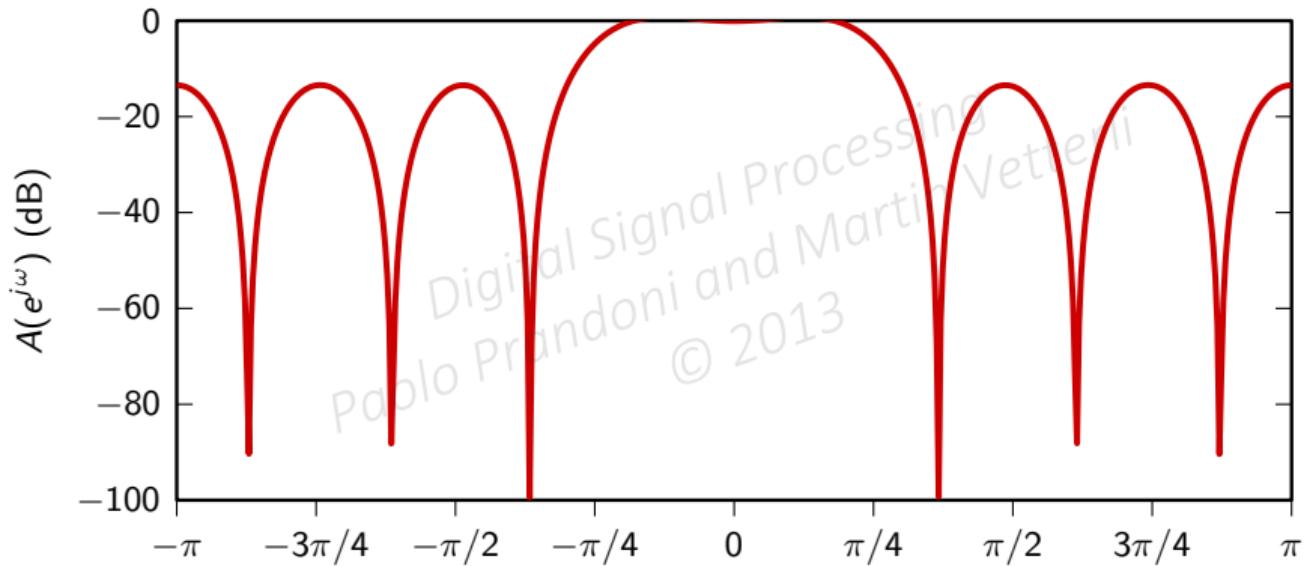
- ▶ useful to compare attenuations between filters

4th-order elliptic lowpass, $\omega_c = \pi/4$, log scale



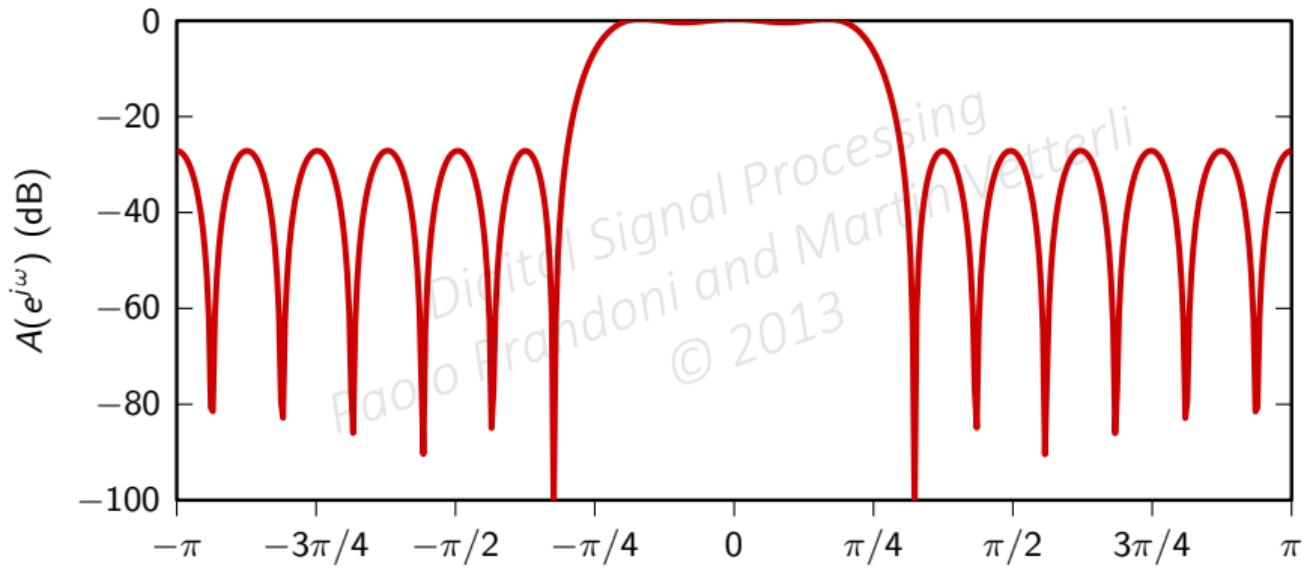
Digital Signal Processing
Paolo Brandolini and Martin Vetterli
© 2013

9-tap minimax lowpass, $\omega_c = \pi/4$, log scale

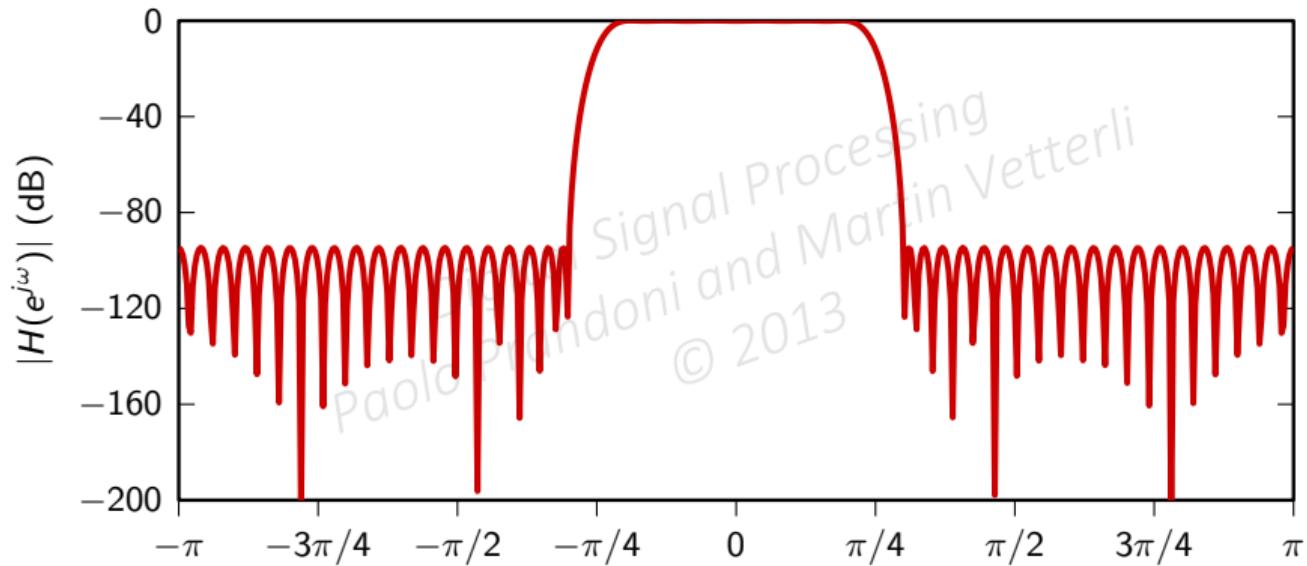


Digital Signal Processing
Pablo Prandoni and Martin Vetterli
© 2013

19-tap minimax lowpass, $\omega_c = \pi/4$, log scale



51-tap minimax lowpass, $\omega_c = \pi/4$, log scale



The IIR and FIR methods we just described can be used to design more general filter types than lowpass, with only minor modifications

- ▶ IIR bandpass and highpass can be obtained by modulating the lowpass response
- ▶ optimal FIR bandpass and highpass can be designed by the Parks-McClellan algorithm
- ▶ optimal FIR can also be designed with piecewise linear magnitude response
- ▶ the literature on filter design is vast: this is just the tip of the iceberg!

The IIR and FIR methods we just described can be used to design more general filter types than lowpass, with only minor modifications

- ▶ IIR bandpass and highpass can be obtained by modulating the lowpass response
- ▶ optimal FIR bandpass and highpass can be designed by the Parks-McClellan algorithm
- ▶ optimal FIR can also be designed with piecewise linear magnitude response
- ▶ the literature on filter design is vast: this is just the tip of the iceberg!

The IIR and FIR methods we just described can be used to design more general filter types than lowpass, with only minor modifications

- ▶ IIR bandpass and highpass can be obtained by modulating the lowpass response
- ▶ optimal FIR bandpass and highpass can be designed by the Parks-McClellan algorithm
- ▶ optimal FIR can also be designed with piecewise linear magnitude response
- ▶ the literature on filter design is vast: this is just the tip of the iceberg!

The IIR and FIR methods we just described can be used to design more general filter types than lowpass, with only minor modifications

- ▶ IIR bandpass and highpass can be obtained by modulating the lowpass response
- ▶ optimal FIR bandpass and highpass can be designed by the Parks-McClellan algorithm
- ▶ optimal FIR can also be designed with piecewise linear magnitude response
- ▶ the literature on filter design is vast: this is just the tip of the iceberg!

END OF MODULE 5.10

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Digital Signal Processing

Module 5.11: Real-Time Processing

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Summary:

- ▶ I/O and DMA

- ▶ multiple buffering
- ▶ implementation framework
- ▶ some guitar effects

Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

Summary:

- ▶ I/O and DMA
- ▶ multiple buffering
- ▶ implementation framework
- ▶ some guitar effects

Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

Summary:

- ▶ I/O and DMA
- ▶ multiple buffering
- ▶ implementation framework
- ▶ some guitar effects

Digital Signal Processing
paolo Prandoni and Martin Vetterli
© 2013

Summary:

- ▶ I/O and DMA
- ▶ multiple buffering
- ▶ implementation framework
- ▶ some guitar effects

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Everything works in sync with a *system clock* of period T_s :

- ▶ “record” a value $x_i[n]$
 - ▶ process the value in a causal filter
 - ▶ “play” the output $x_o[n]$
- everything needs to happen in at most T_s seconds!

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Everything works in sync with a *system clock* of period T_s :

- ▶ “record” a value $x_i[n]$
- ▶ process the value in a causal filter
- ▶ “play” the output $x_o[n]$

everything needs to happen in at most T_s seconds!

Everything works in sync with a *system clock* of period T_s :

- ▶ “record” a value $x_i[n]$
 - ▶ process the value in a causal filter
 - ▶ “play” the output $x_o[n]$
- everything needs to happen in at most T_s seconds!

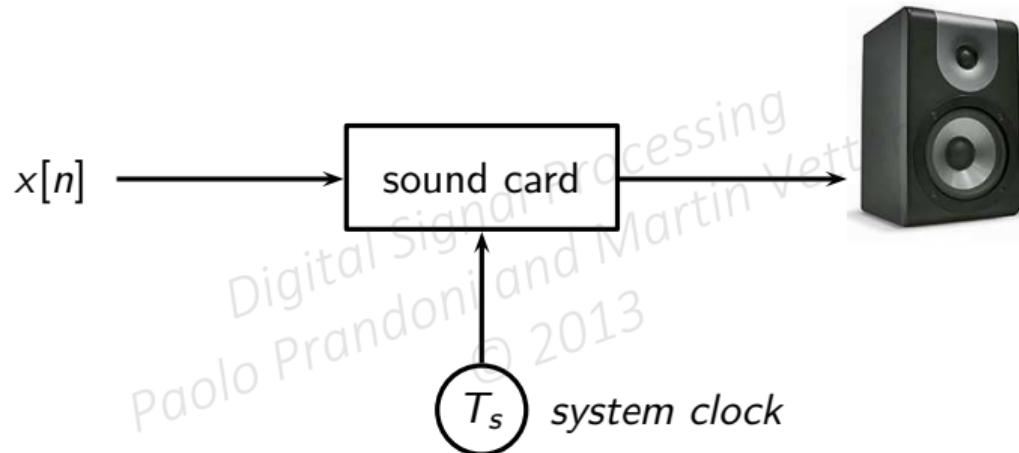
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Everything works in sync with a *system clock* of period T_s :

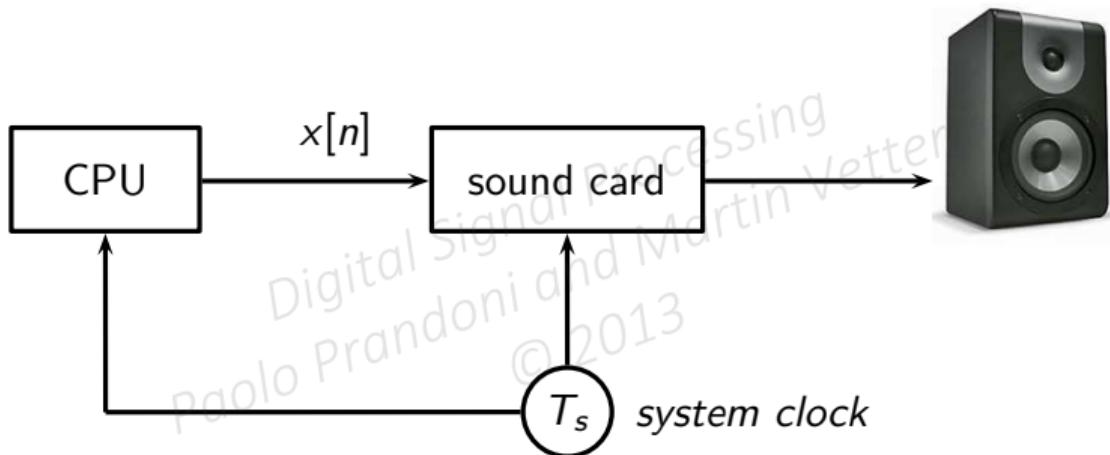
- ▶ “record” a value $x_i[n]$
- ▶ process the value in a causal filter
- ▶ “play” the output $x_o[n]$

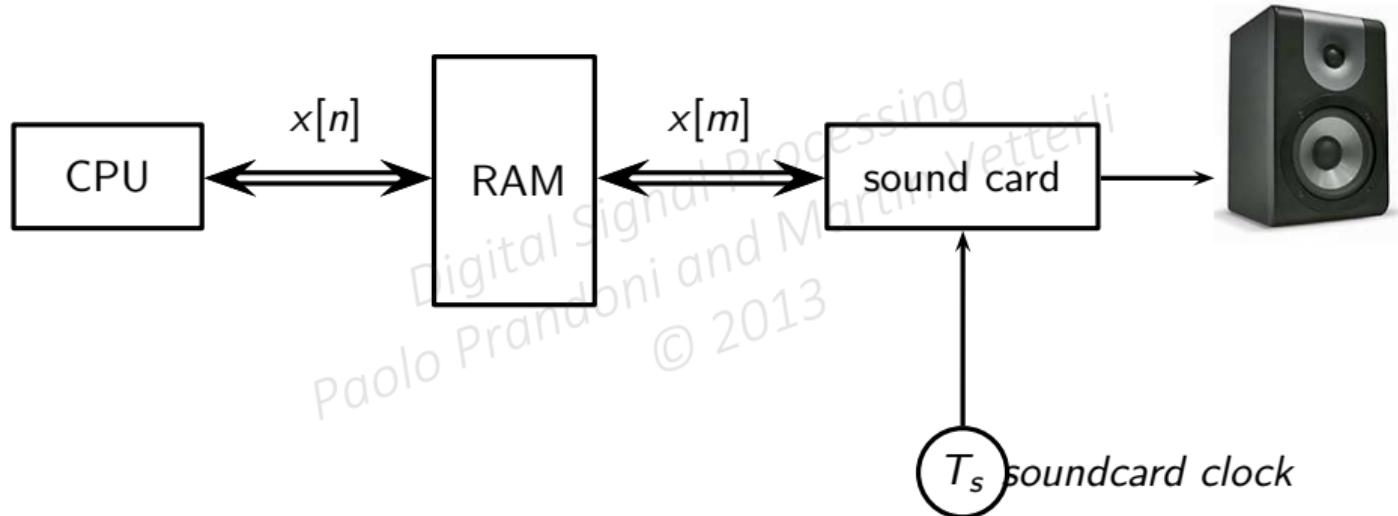
everything needs to happen in at most T_s seconds!

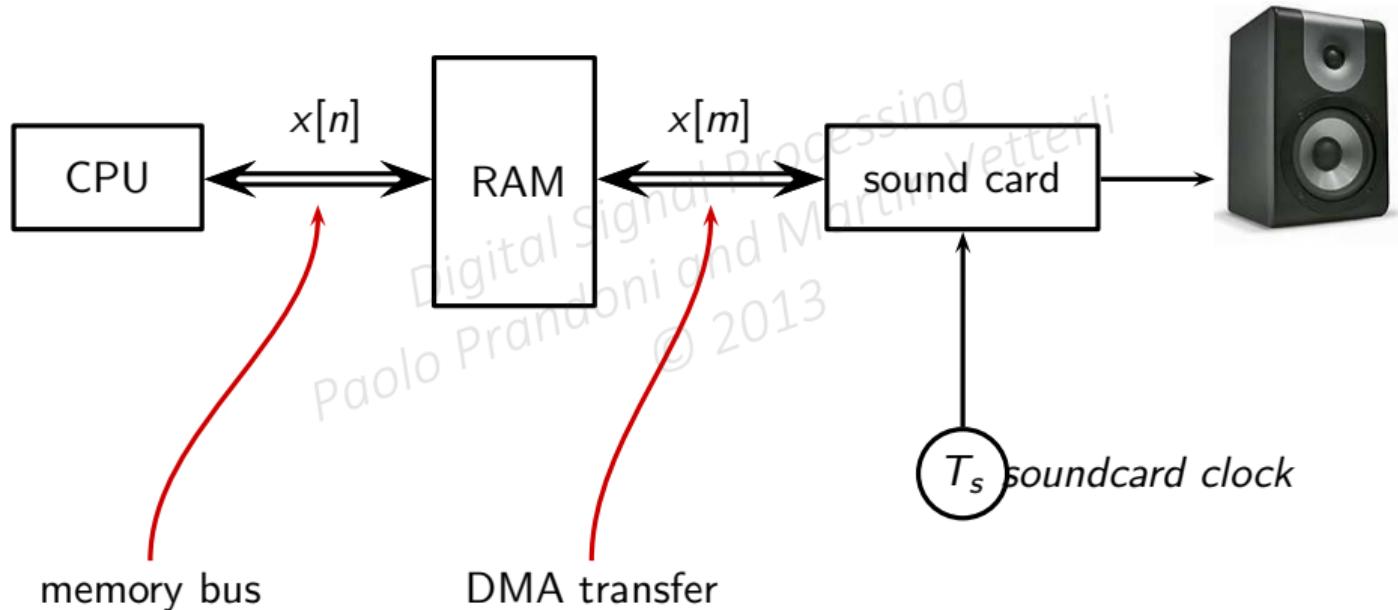
Playing a sound

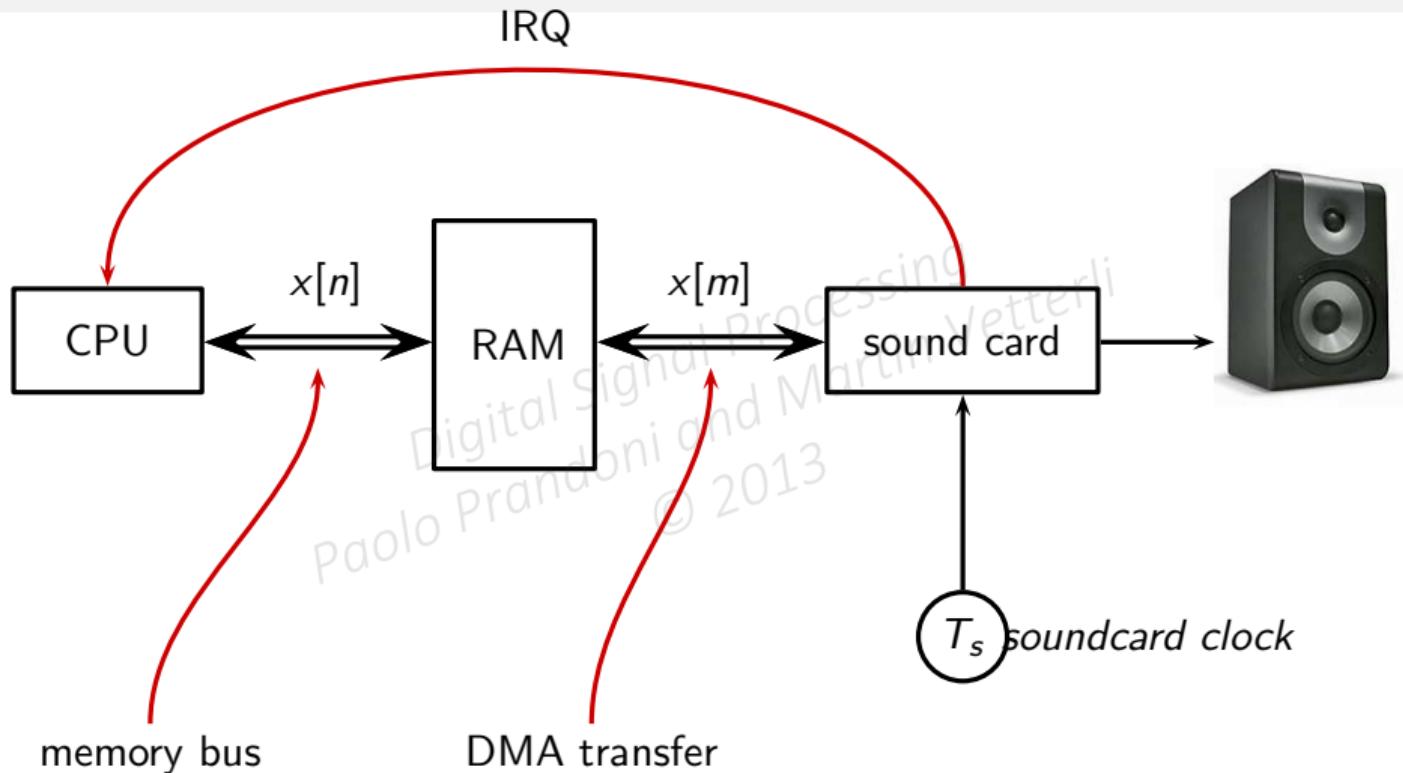


On dedicated hardware...









- ▶ interrupt for each sample would be too much overhead
- ▶ soundcard consumes sample in buffers
- ▶ soundcard notifies when buffer used up
- ▶ CPU can fill a buffer in less time than soundcard can empty it

buffering introduces delay!

Digital Signal Processing
Paolo Pironi and Martin Vetterli
© 2013

- ▶ interrupt for each sample would be too much overhead
- ▶ soundcard consumes sample in buffers
- ▶ soundcard notifies when buffer used up
- ▶ CPU can fill a buffer in less time than soundcard can empty it

buffering introduces delay!

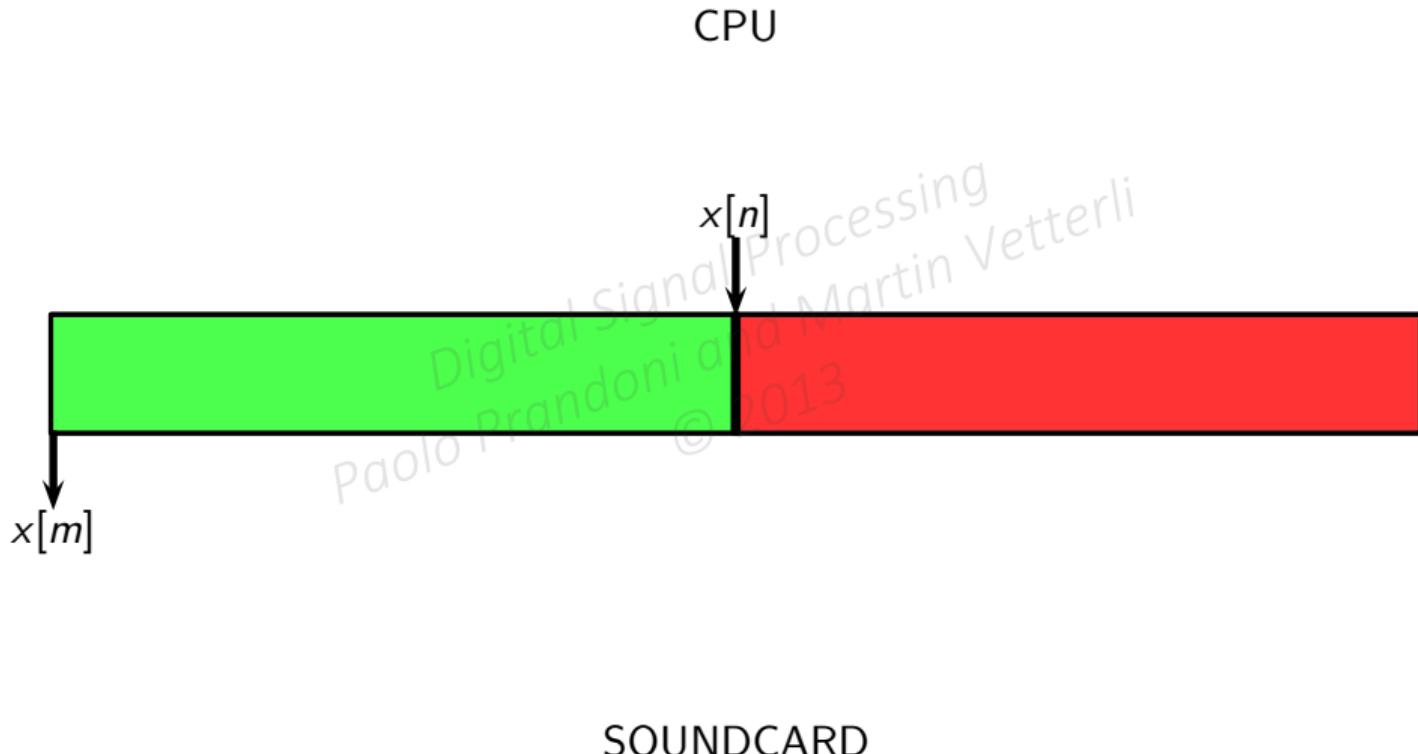
- ▶ interrupt for each sample would be too much overhead
- ▶ soundcard consumes sample in buffers
- ▶ soundcard notifies when buffer used up
- ▶ CPU can fill a buffer in less time than soundcard can empty it

buffering introduces delay!

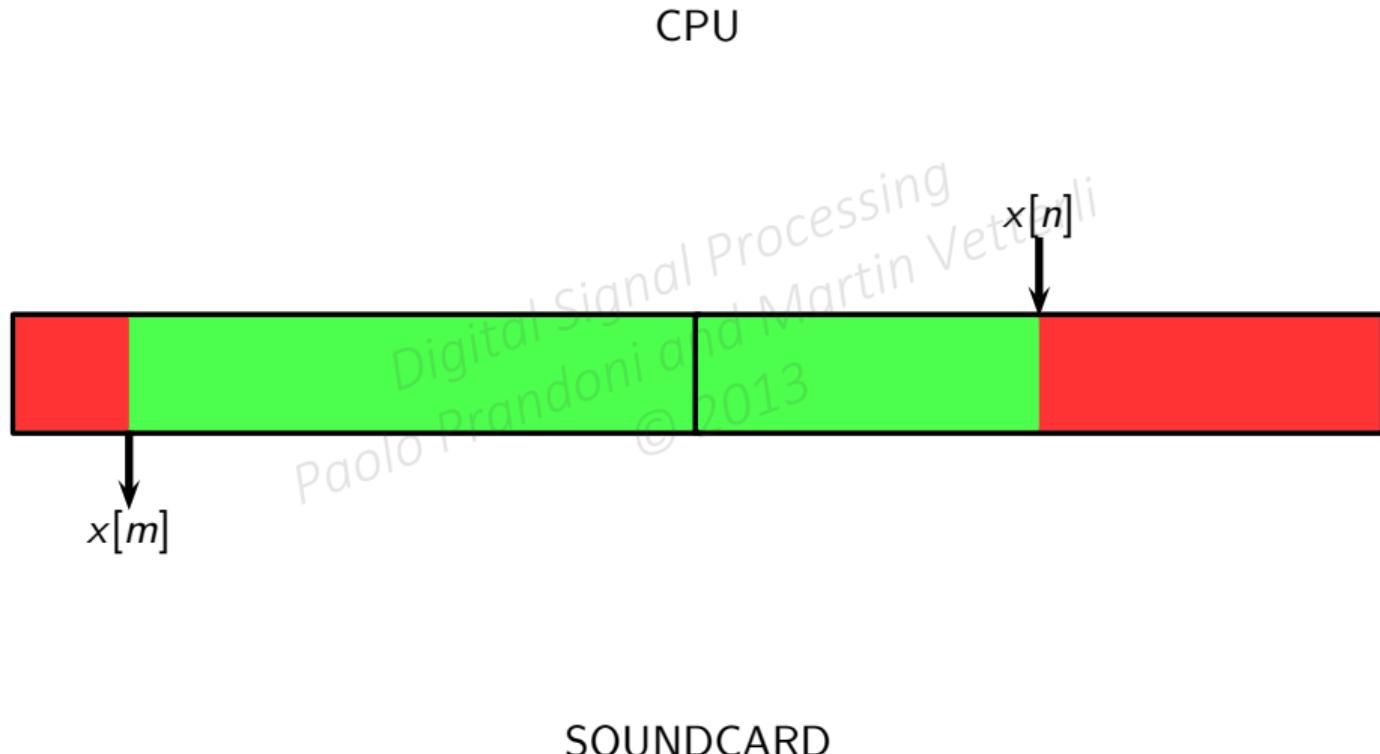
- ▶ interrupt for each sample would be too much overhead
- ▶ soundcard consumes sample in buffers
- ▶ soundcard notifies when buffer used up
- ▶ CPU can fill a buffer in less time than soundcard can empty it

buffering introduces delay!

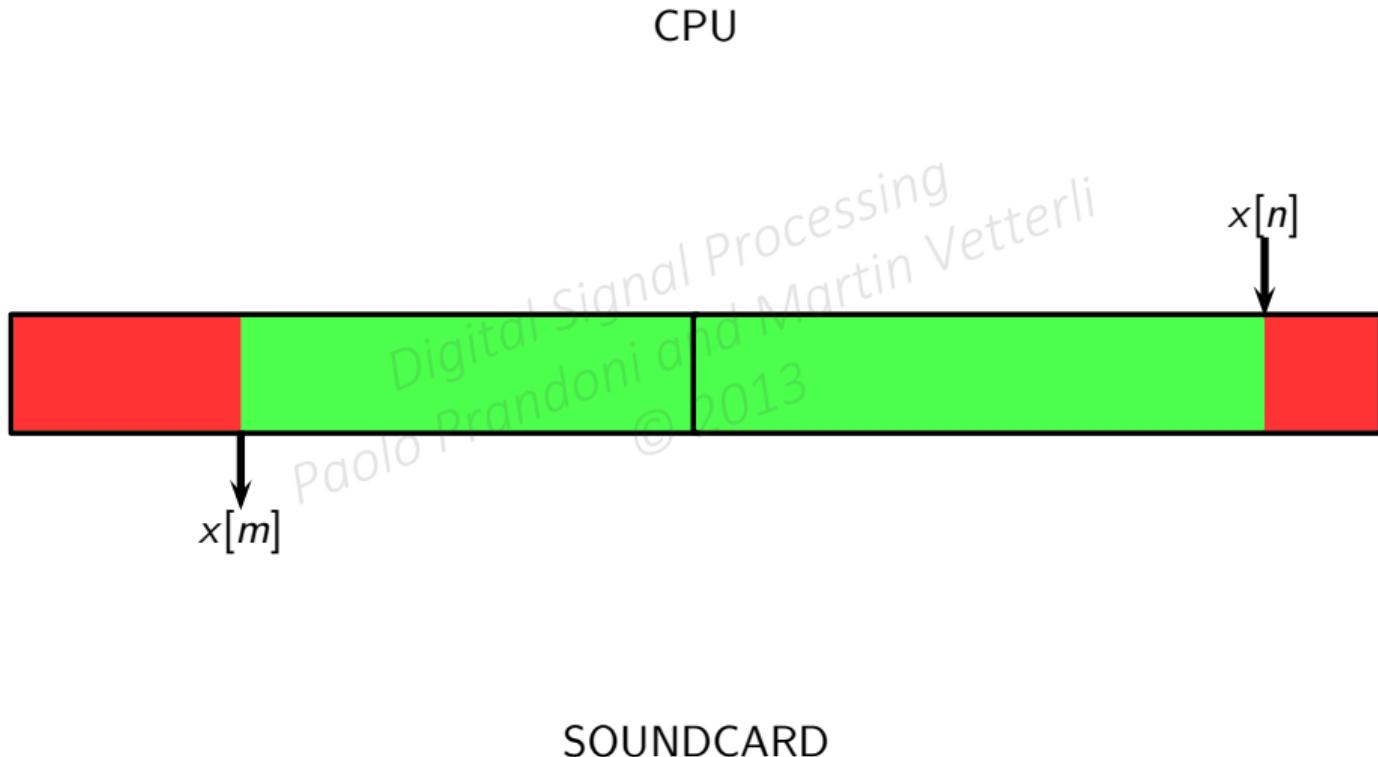
Example: double buffering (output)



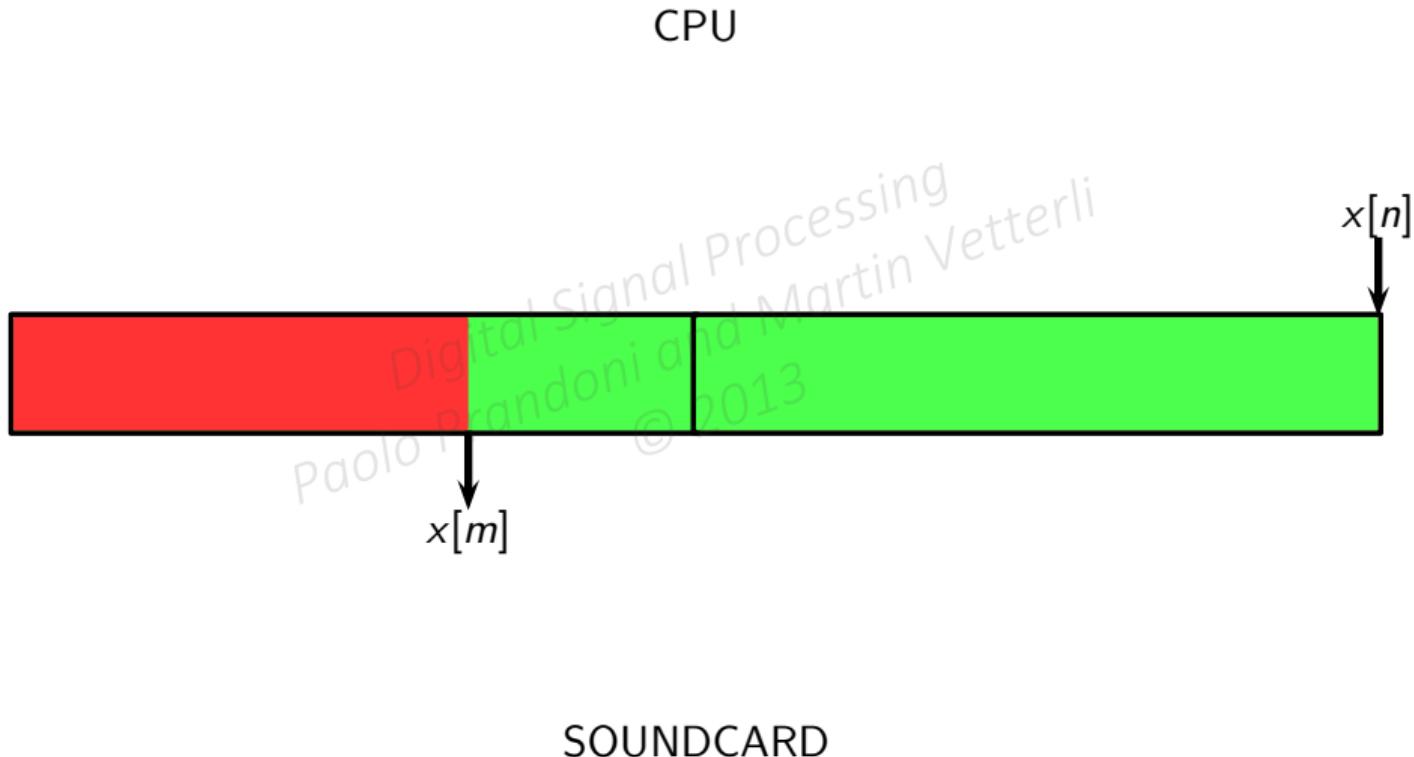
Example: double buffering (output)



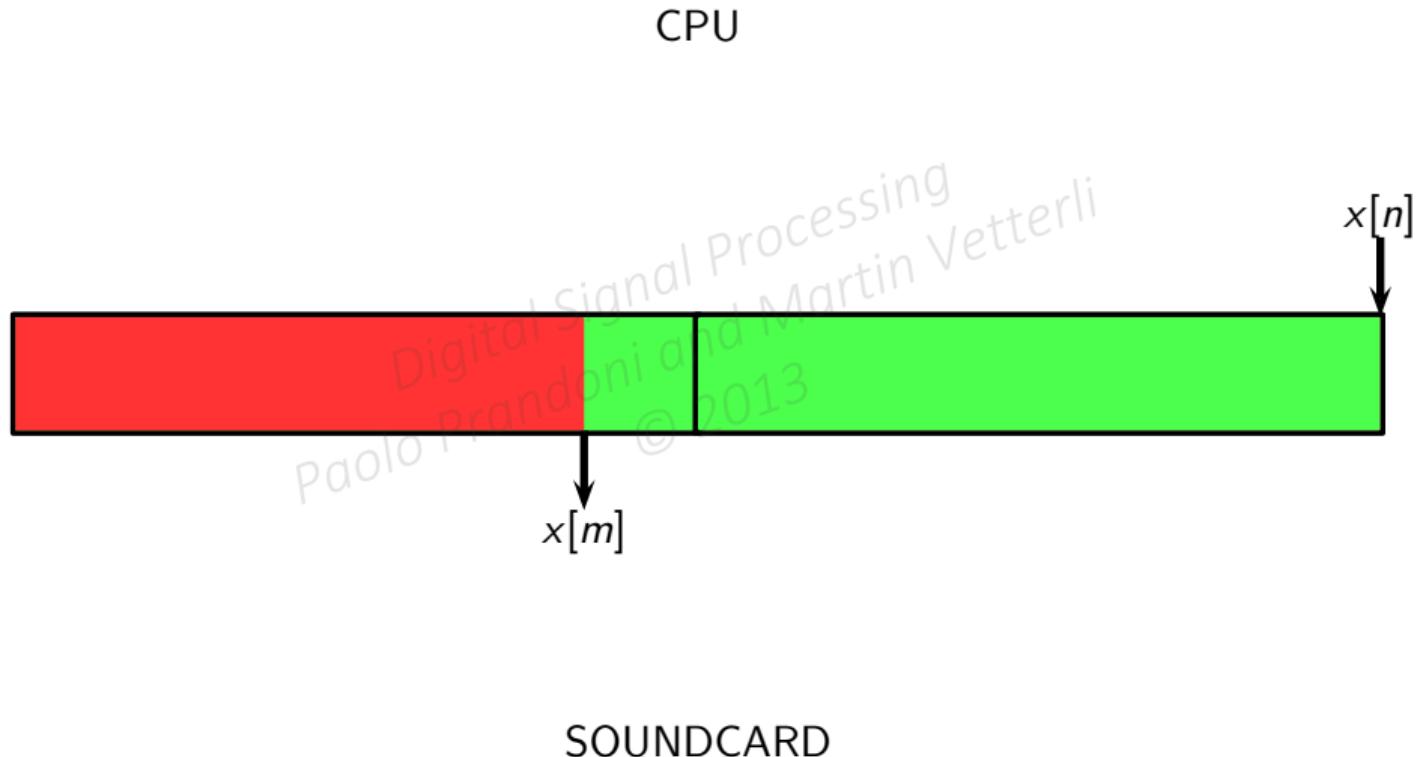
Example: double buffering (output)



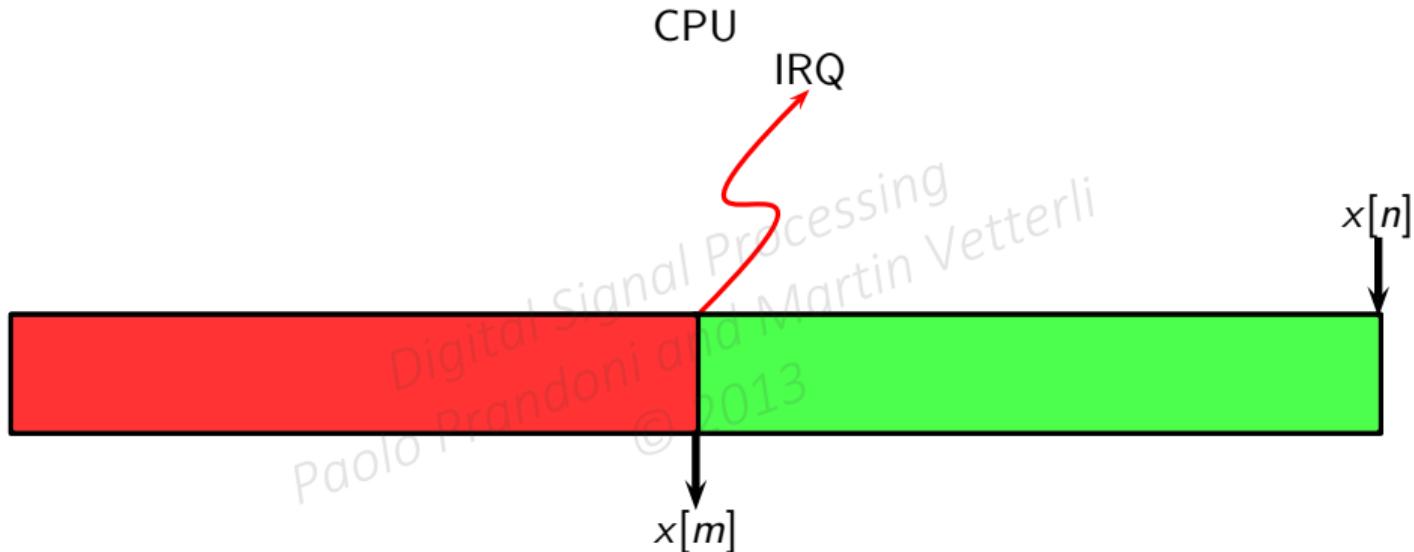
Example: double buffering (output)



Example: double buffering (output)

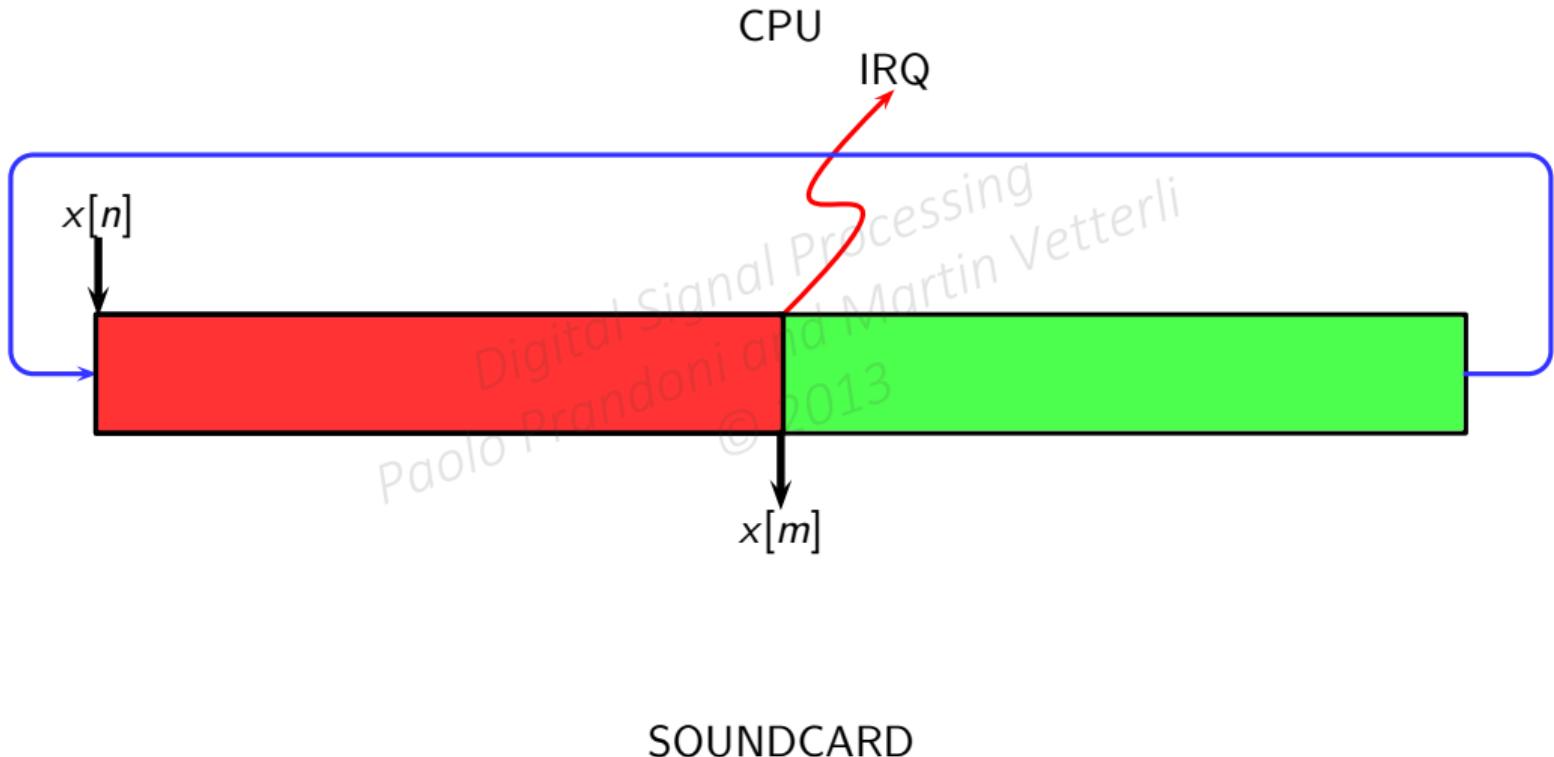


Example: double buffering (output)

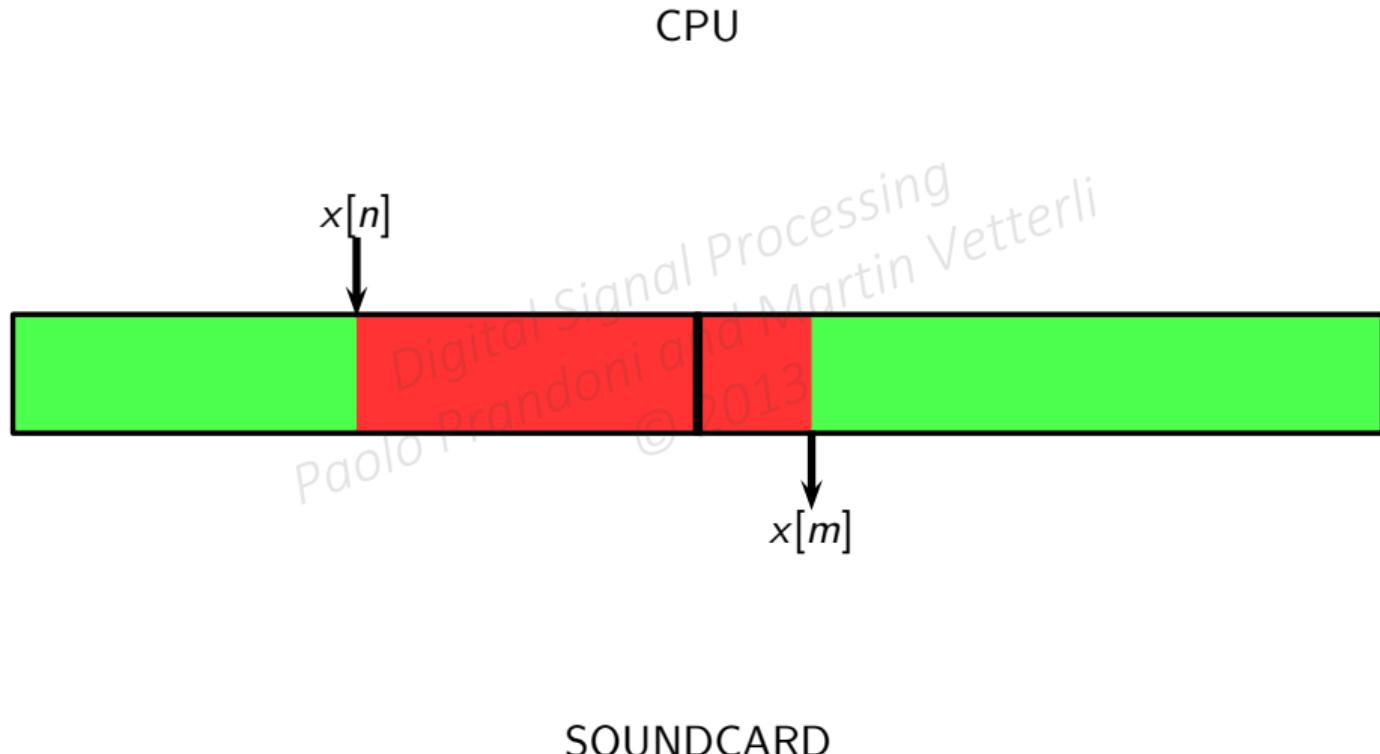


SOUNDCARD

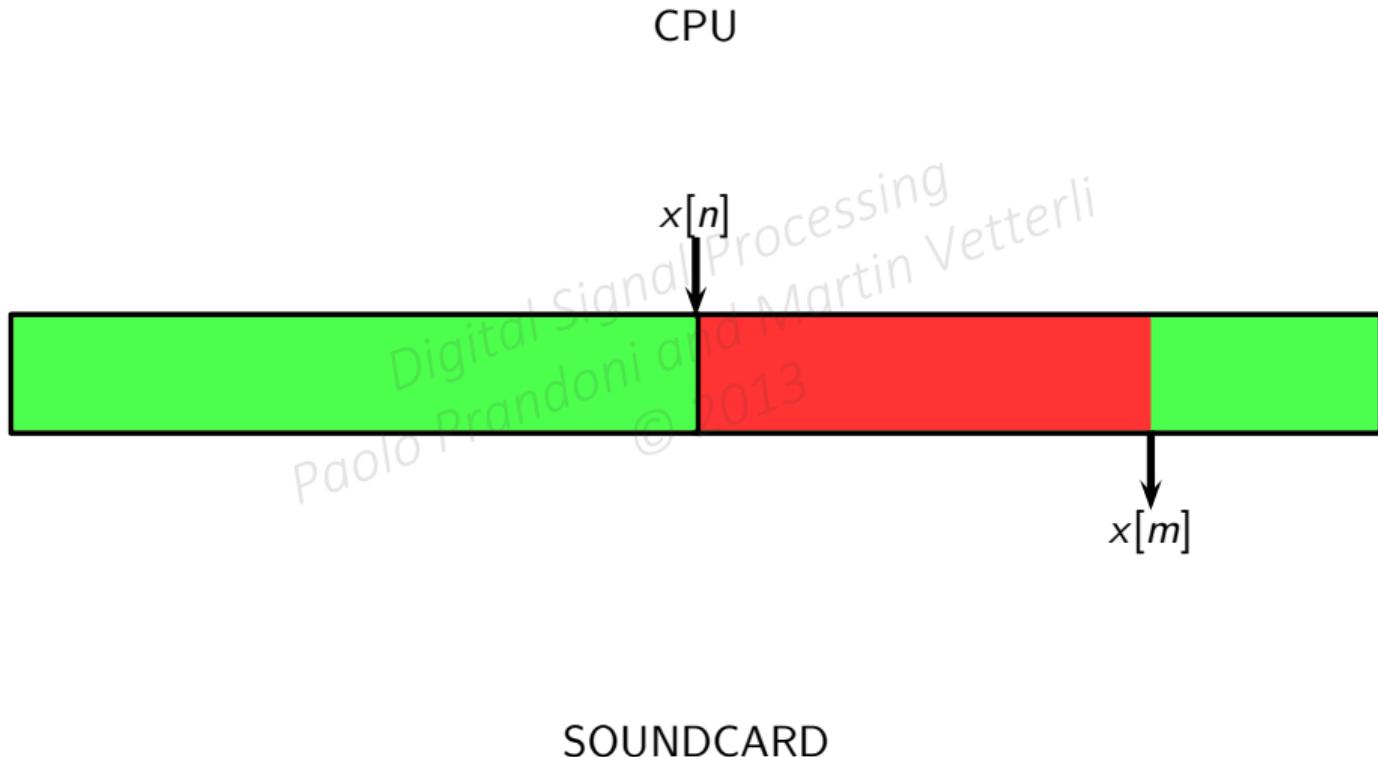
Example: double buffering (output)



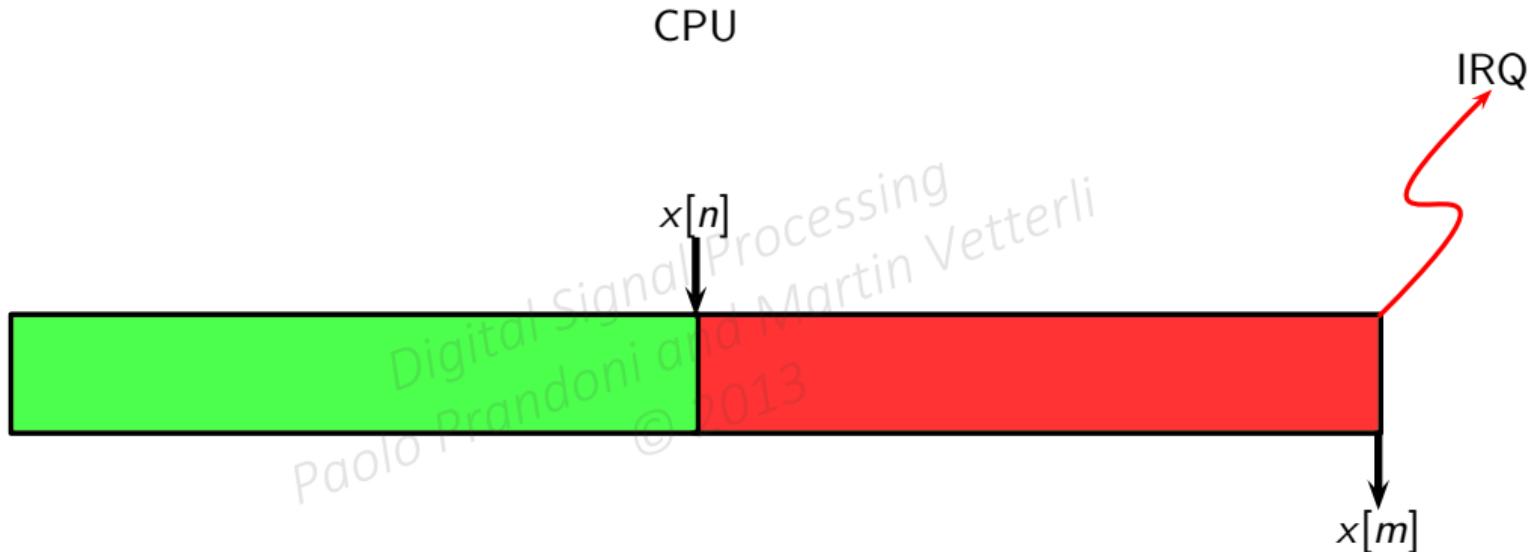
Example: double buffering (output)



Example: double buffering (output)

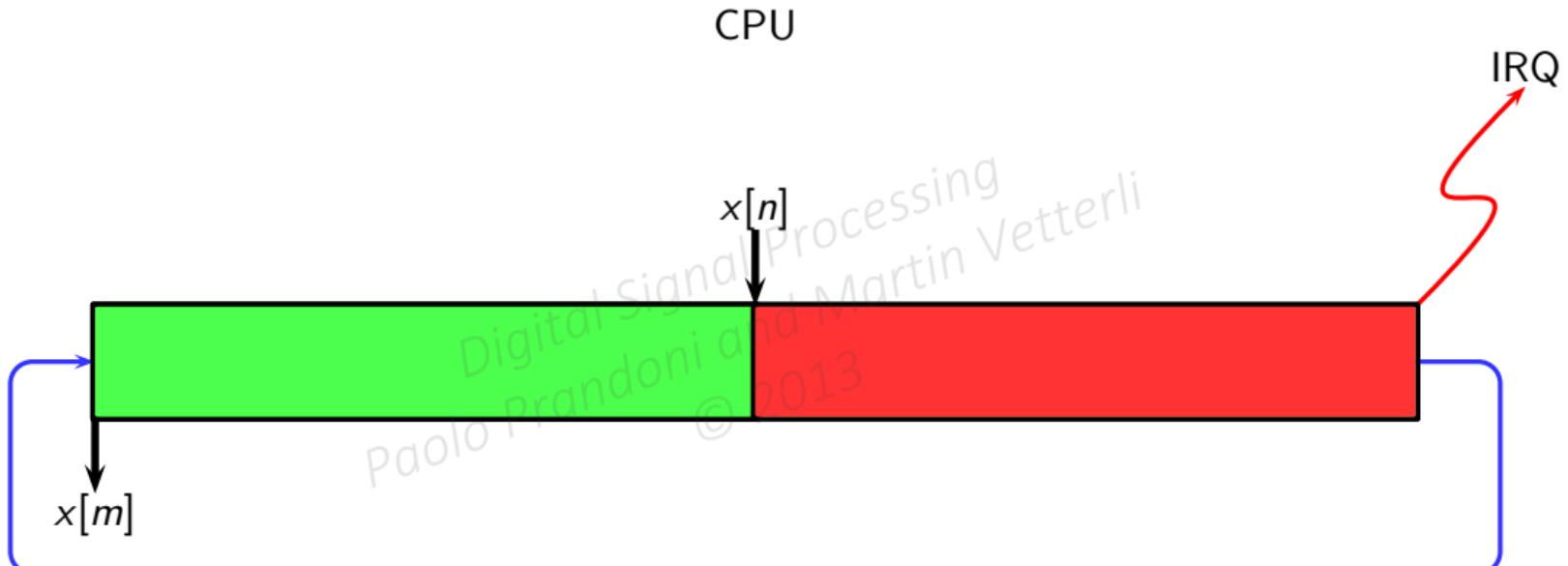


Example: double buffering (output)



SOUNDCARD

Example: double buffering (output)



SOUNDCARD

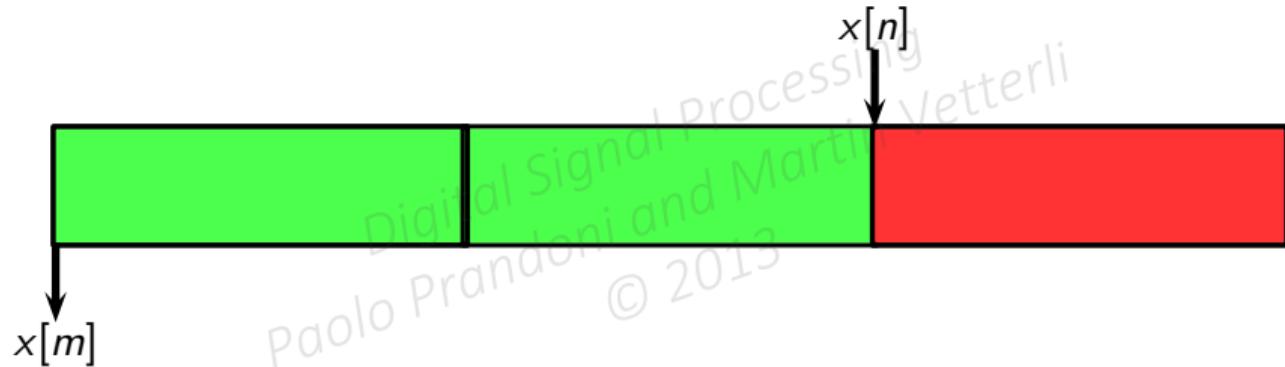
Example: double buffering

- ▶ double buffering introduces a delay $d = T_s \times \frac{L}{2}$ seconds
- ▶ if CPU doesn't fill the buffer fast enough: **underflow**

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2015

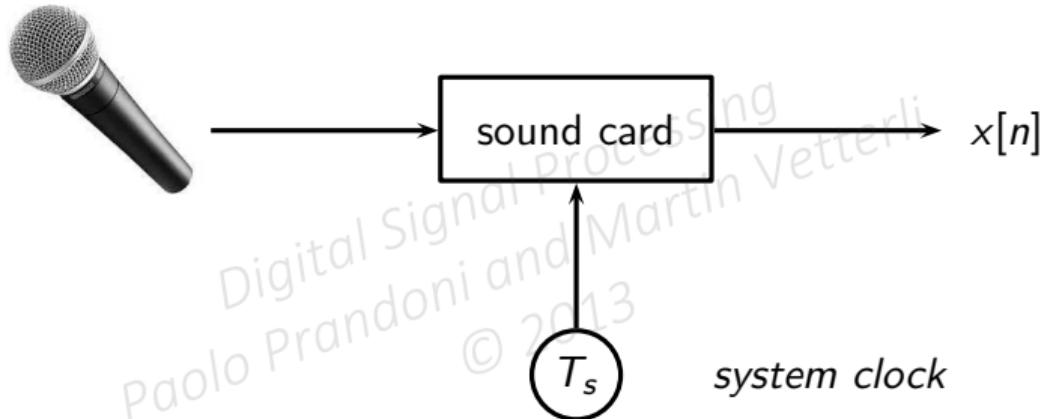
Example: double buffering

- ▶ double buffering introduces a delay $d = T_s \times \frac{L}{2}$ seconds
- ▶ if CPU doesn't fill the buffer fast enough: **underflow**

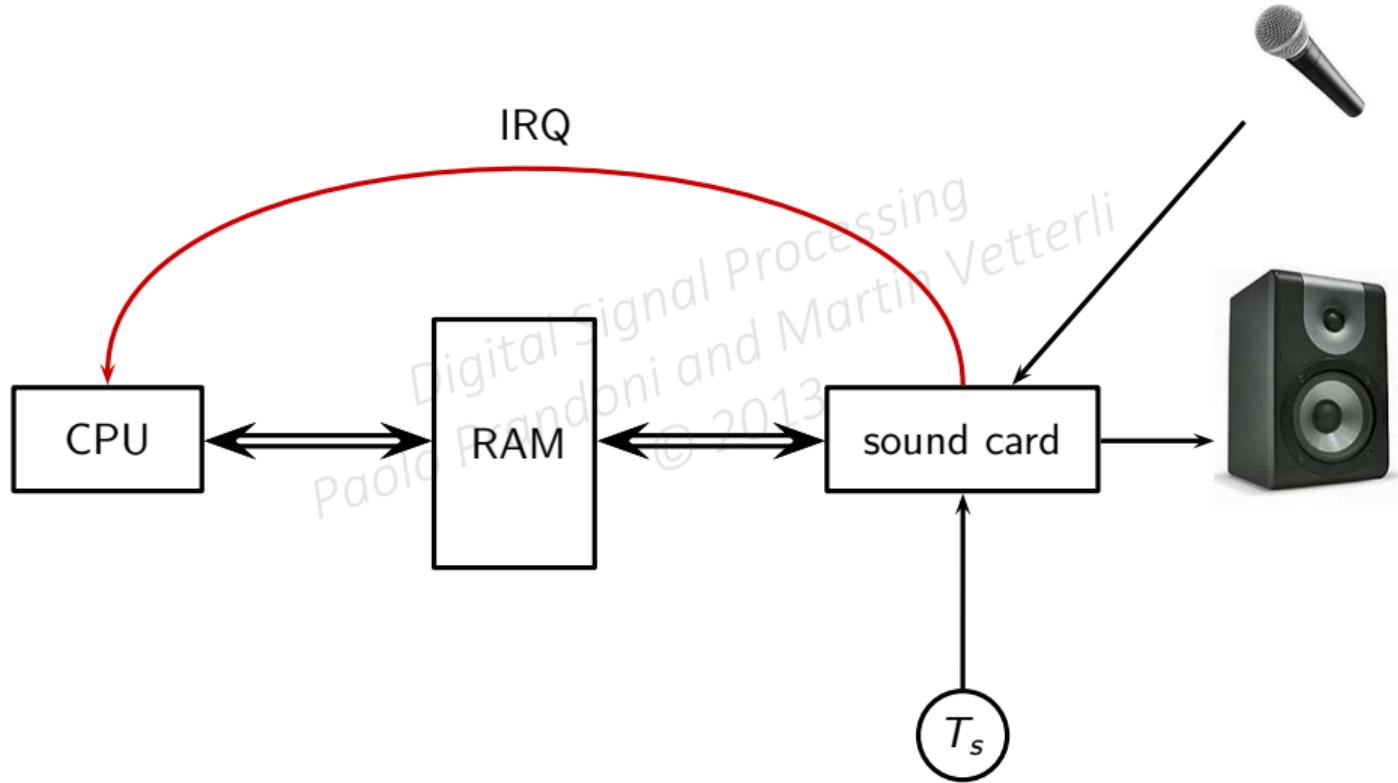


- ▶ call the CPU more often (balance load)
- ▶ keep reasonable underflow protection

What about the input?

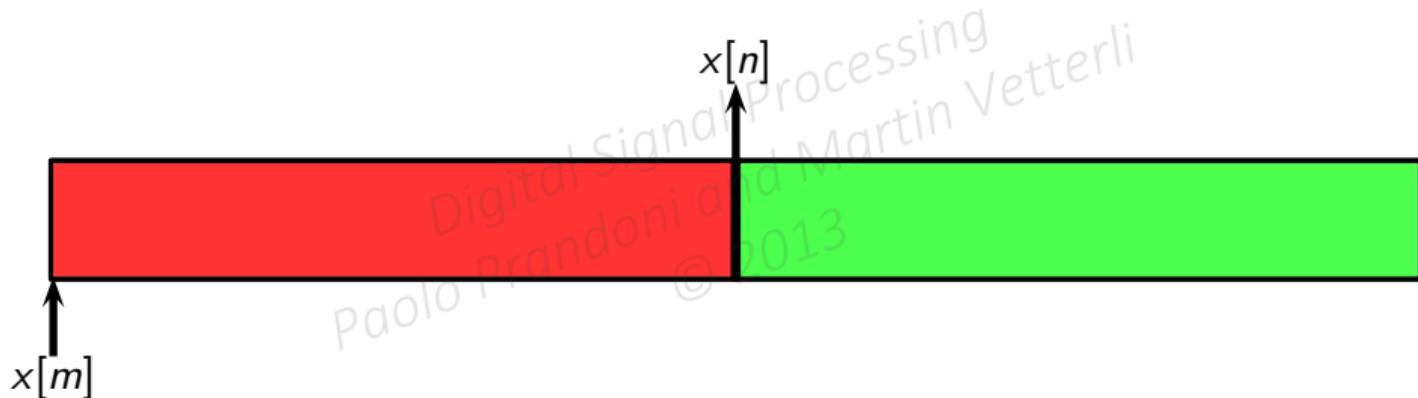


Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



Example: double buffering (input)

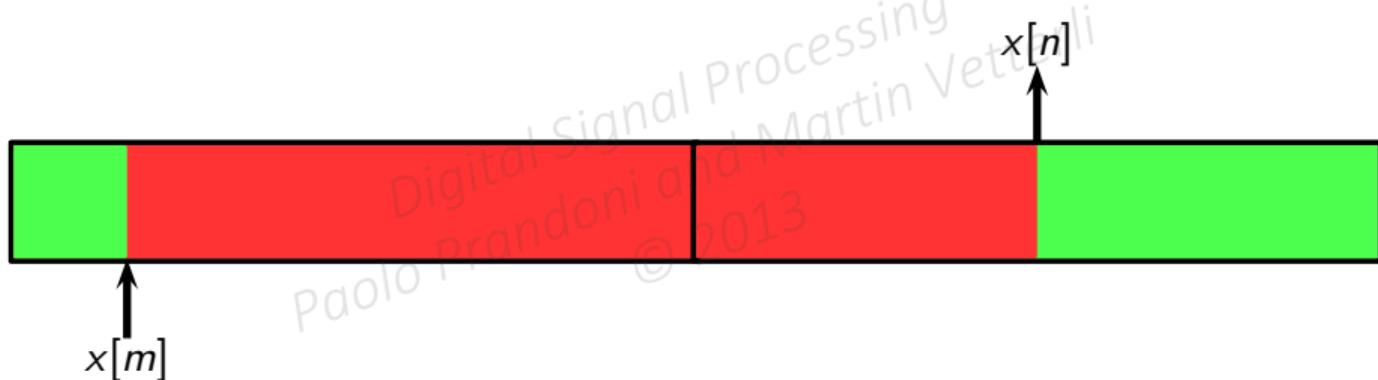
CPU



SOUNDCARD

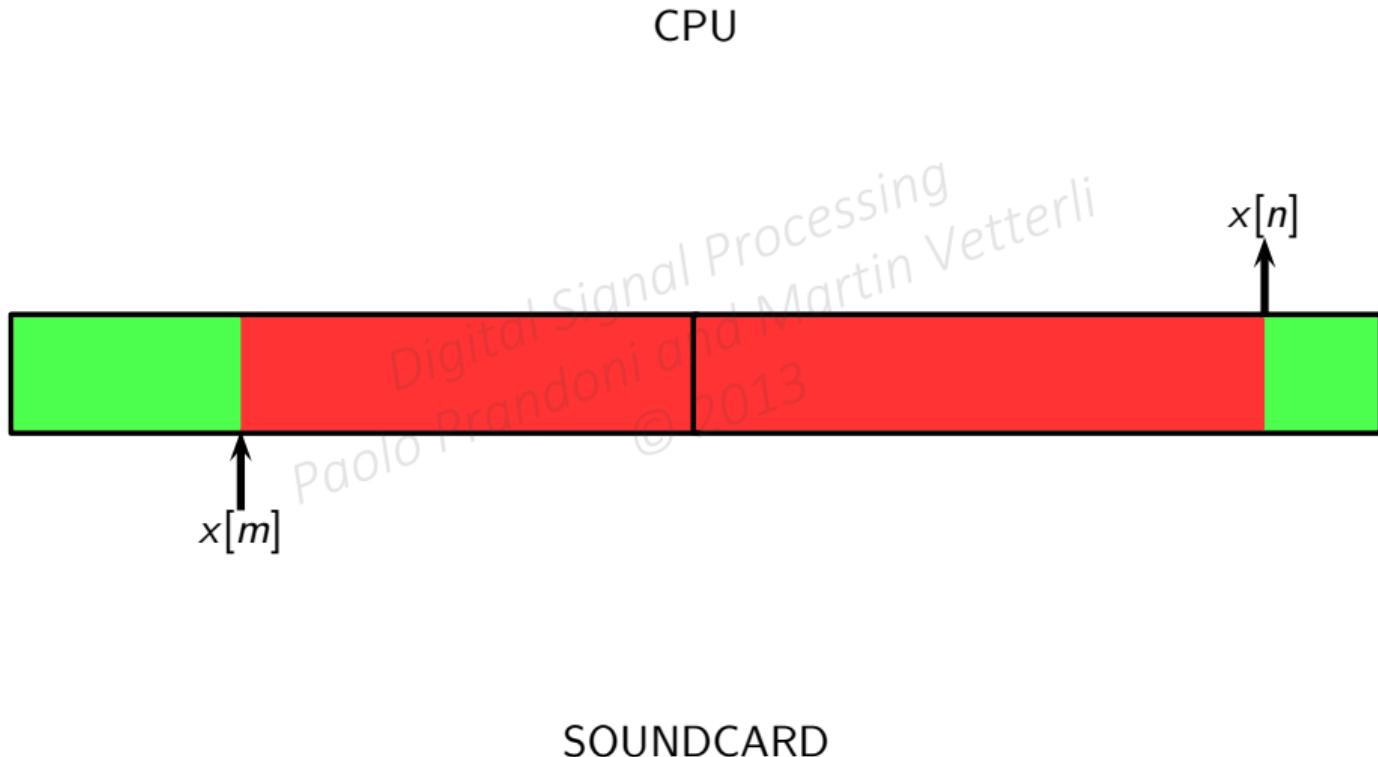
Example: double buffering (input)

CPU

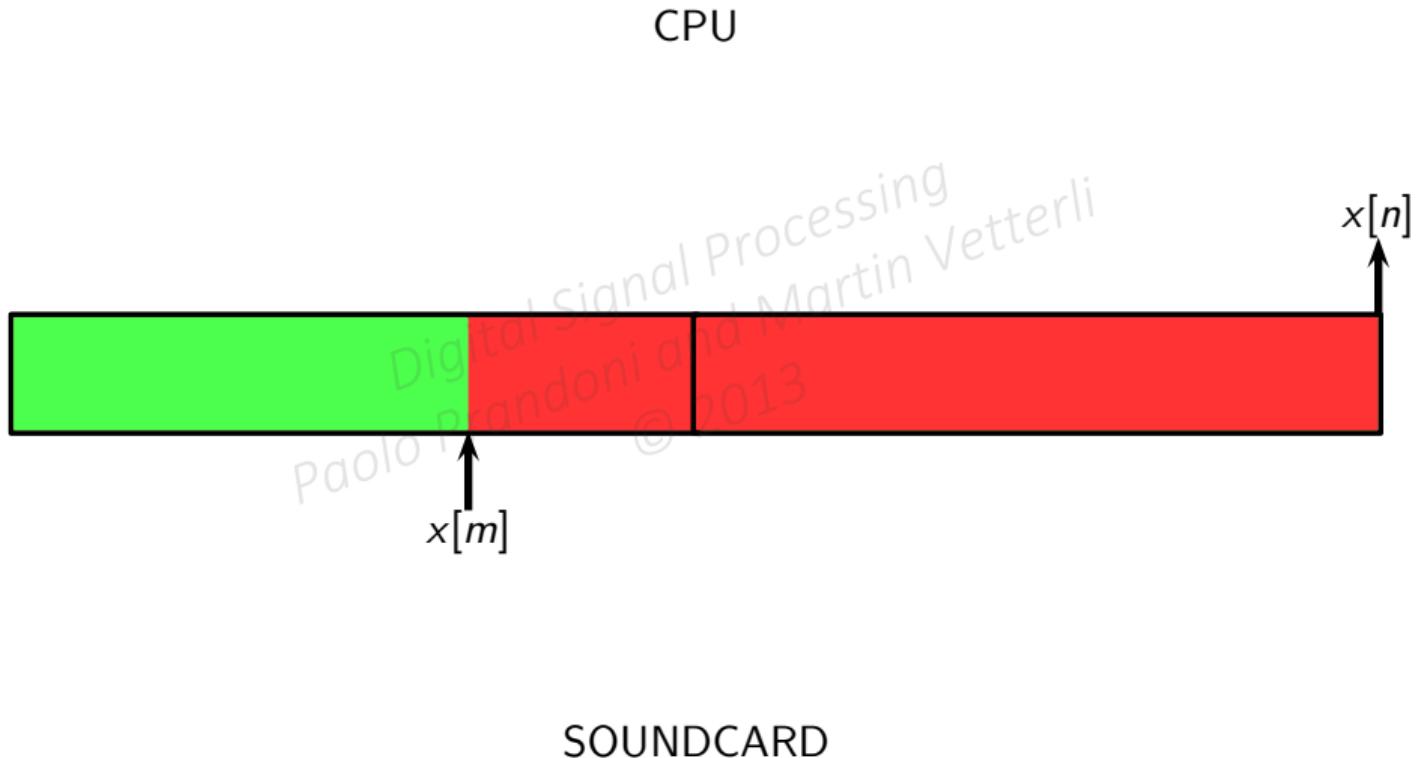


SOUNDCARD

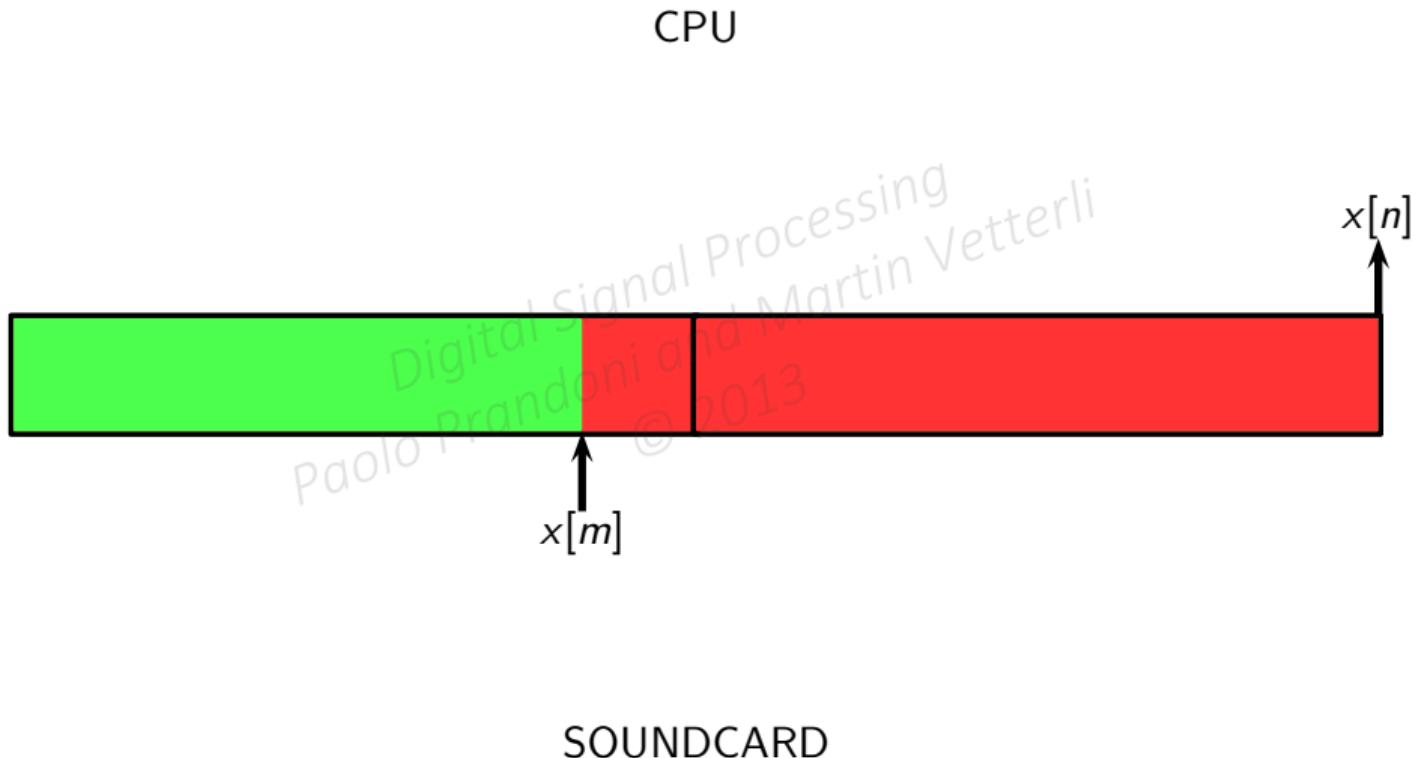
Example: double buffering (input)



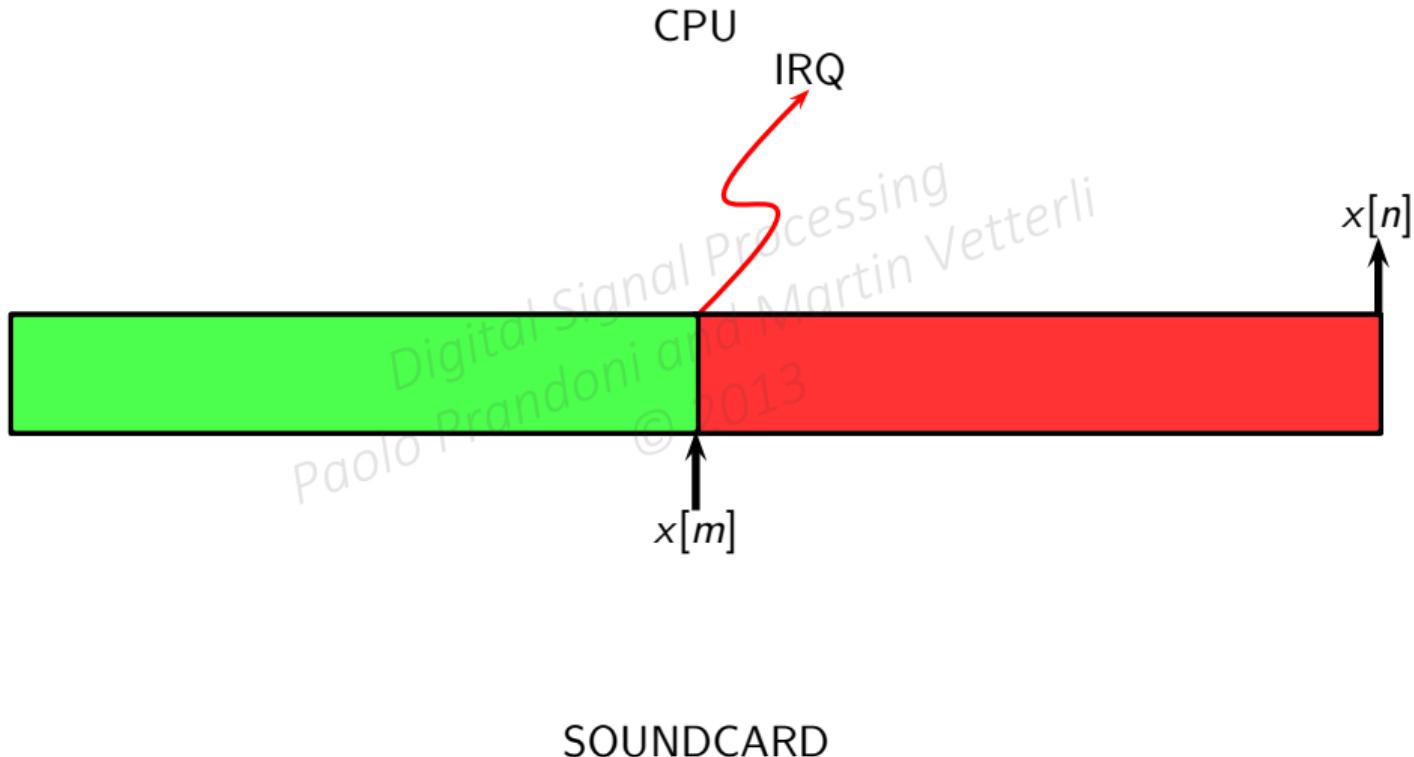
Example: double buffering (input)



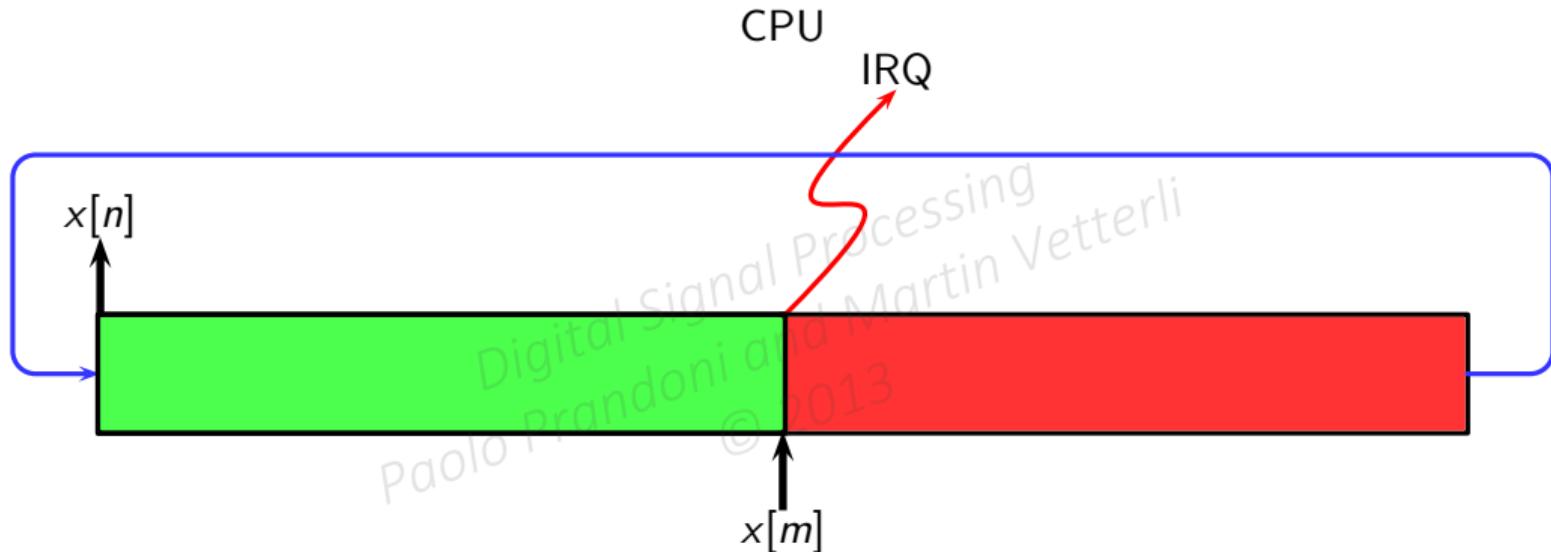
Example: double buffering (input)



Example: double buffering (input)

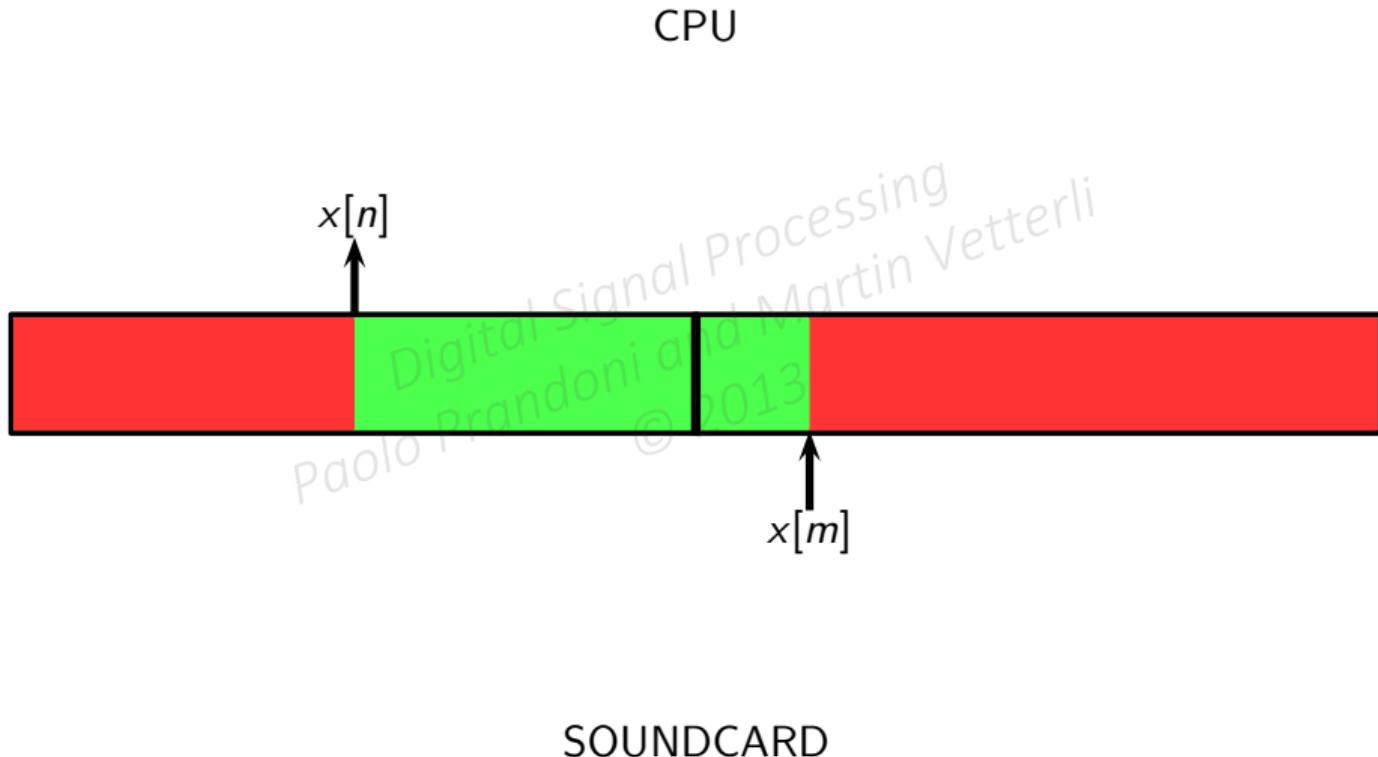


Example: double buffering (input)



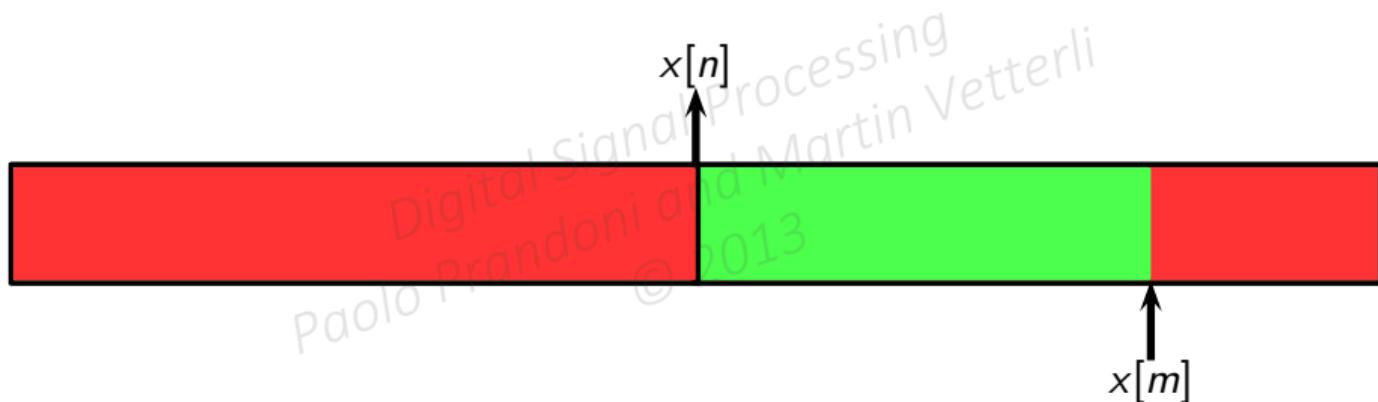
SOUNDCARD

Example: double buffering (input)



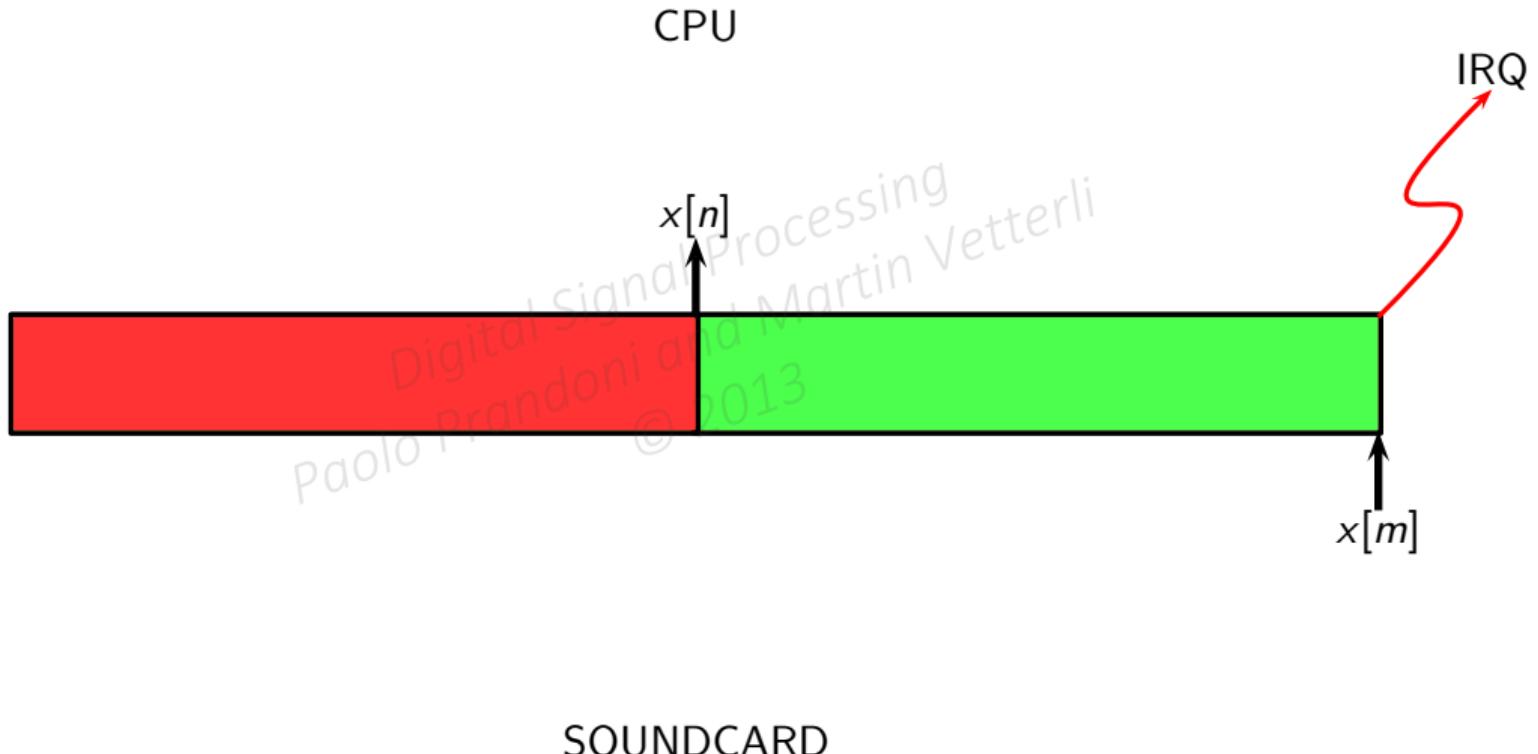
Example: double buffering (input)

CPU

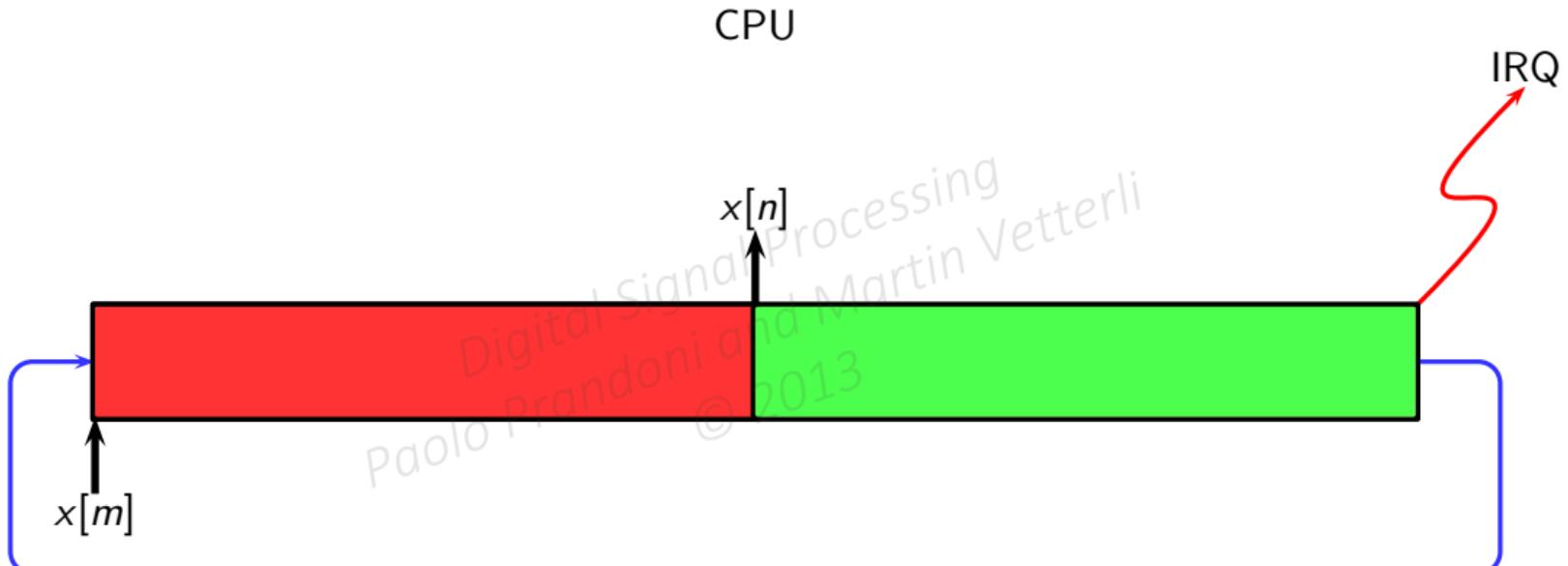


SOUNDCARD

Example: double buffering (input)



Example: double buffering (input)



SOUNDCARD

Putting it all together

- ▶ multiple input buffers and output buffers
- ▶ equal chunk sizes
- ▶ input IRQ drives processing

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Putting it all together

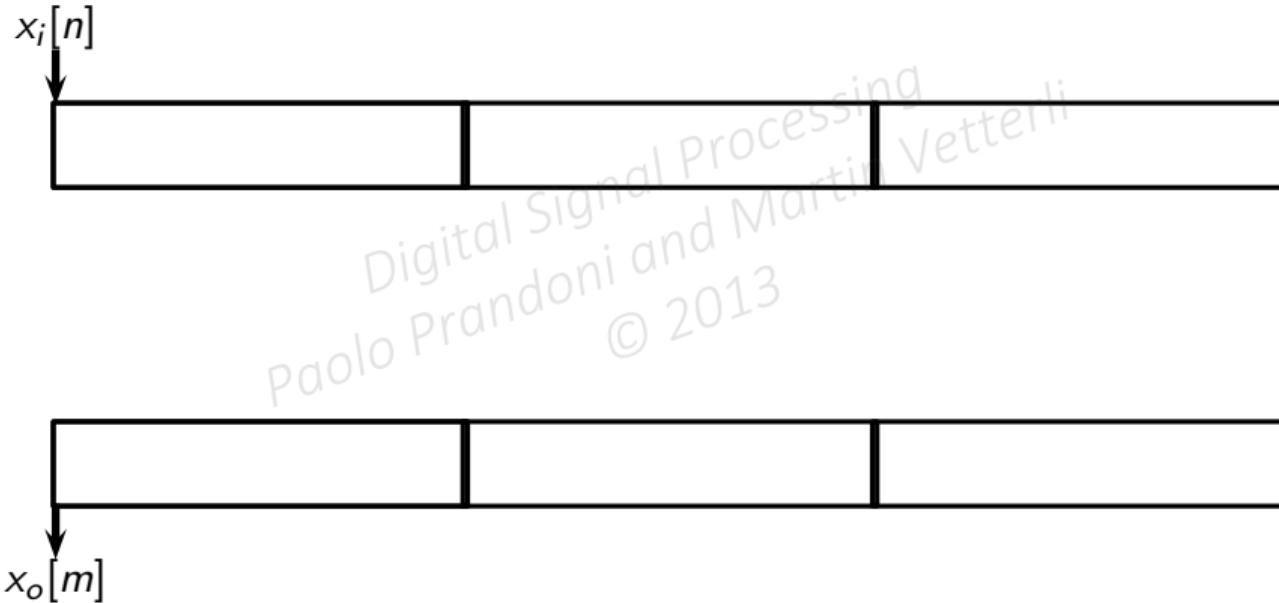
- ▶ multiple input buffers and output buffers
- ▶ equal chunk sizes
- ▶ input IRQ drives processing

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

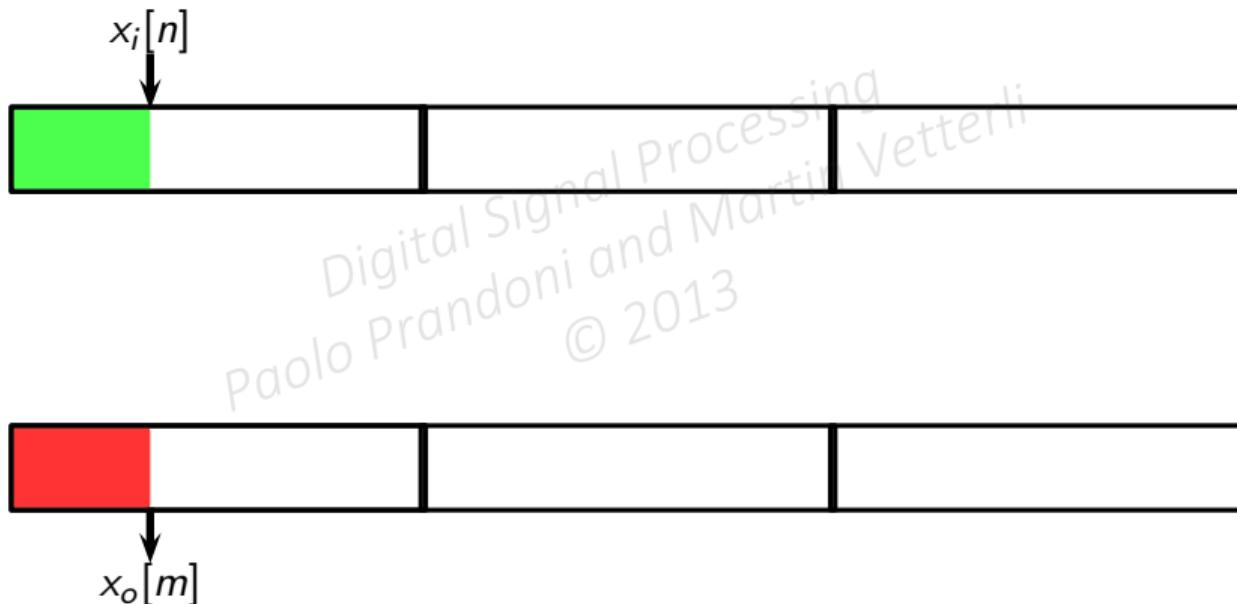
- ▶ multiple input buffers and output buffers
- ▶ equal chunk sizes
- ▶ input IRQ drives processing

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

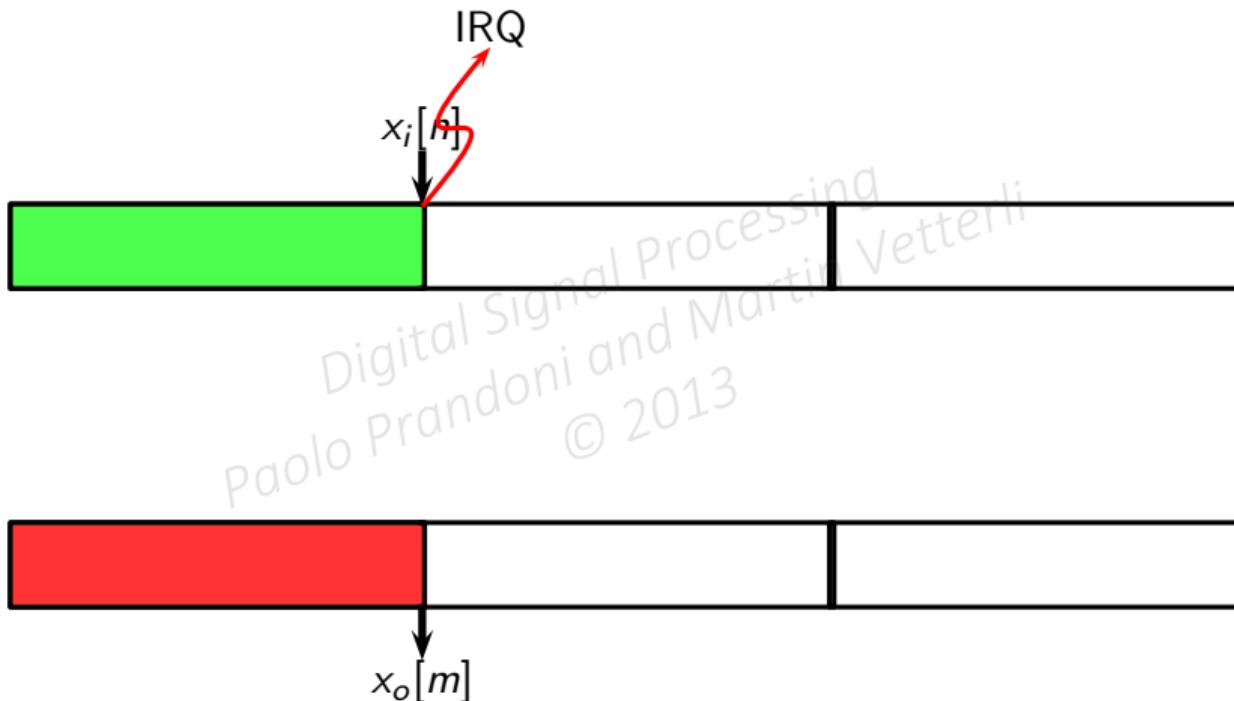
Real-time I/O processing with multiple buffering



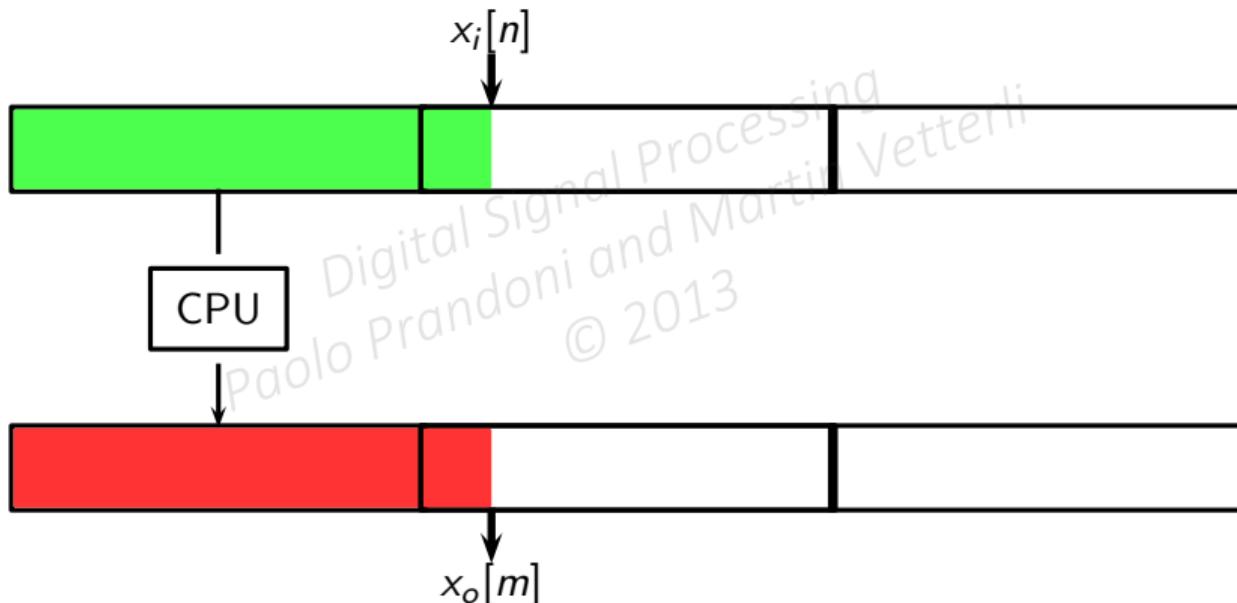
Real-time I/O processing with multiple buffering



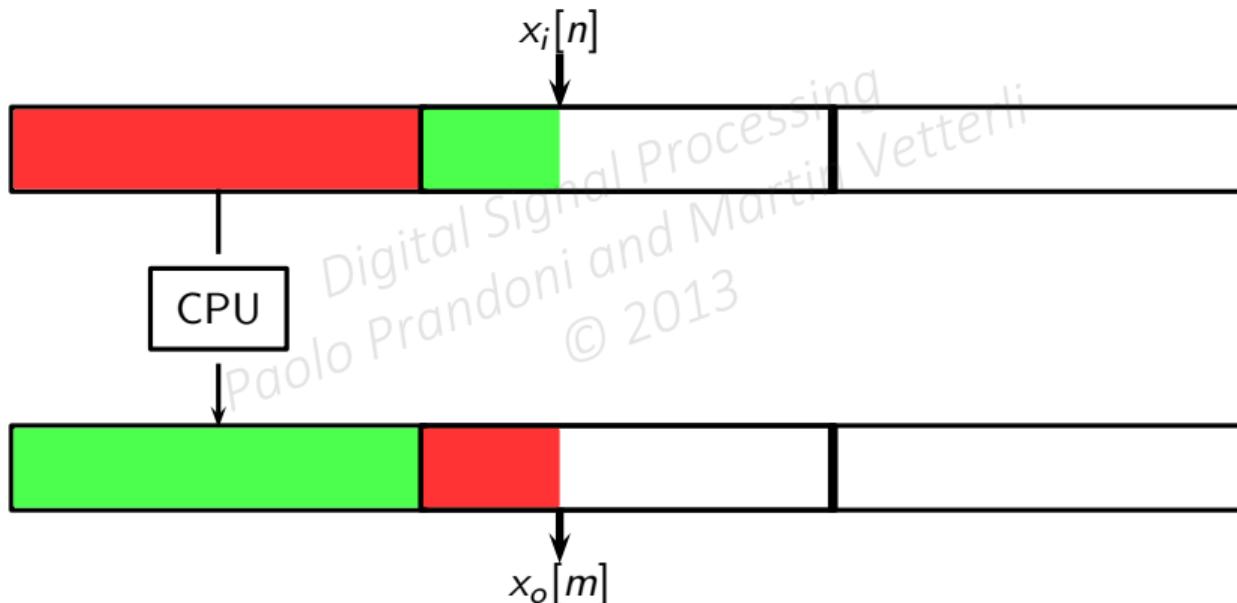
Real-time I/O processing with multiple buffering



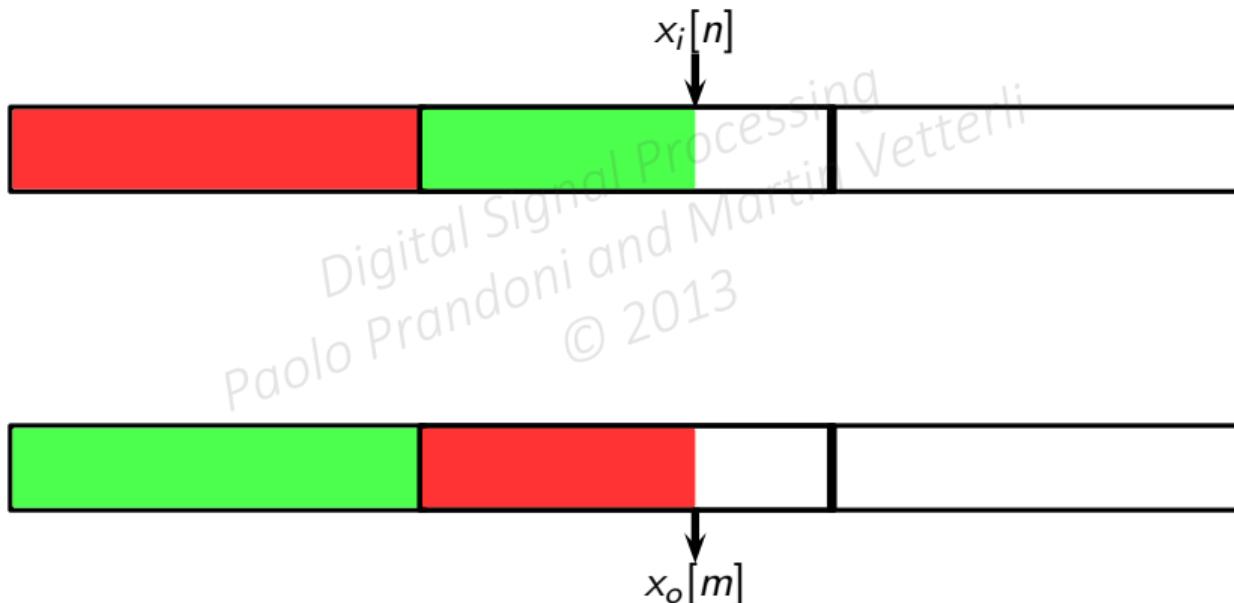
Real-time I/O processing with multiple buffering



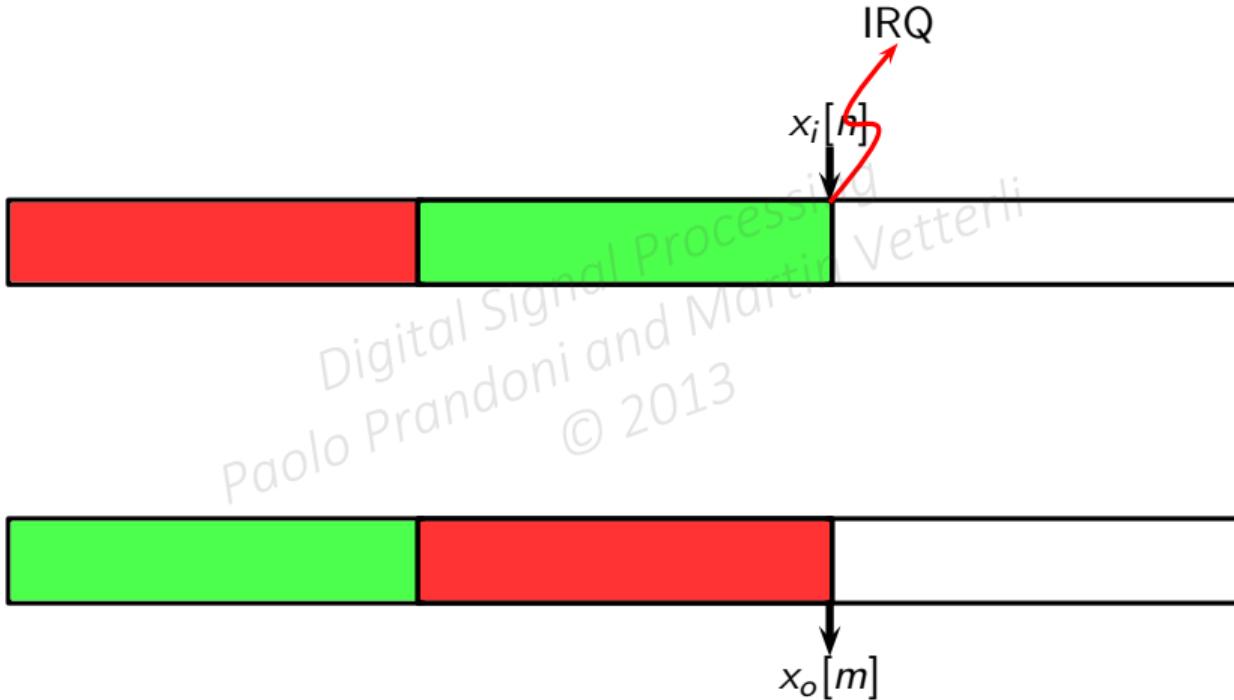
Real-time I/O processing with multiple buffering



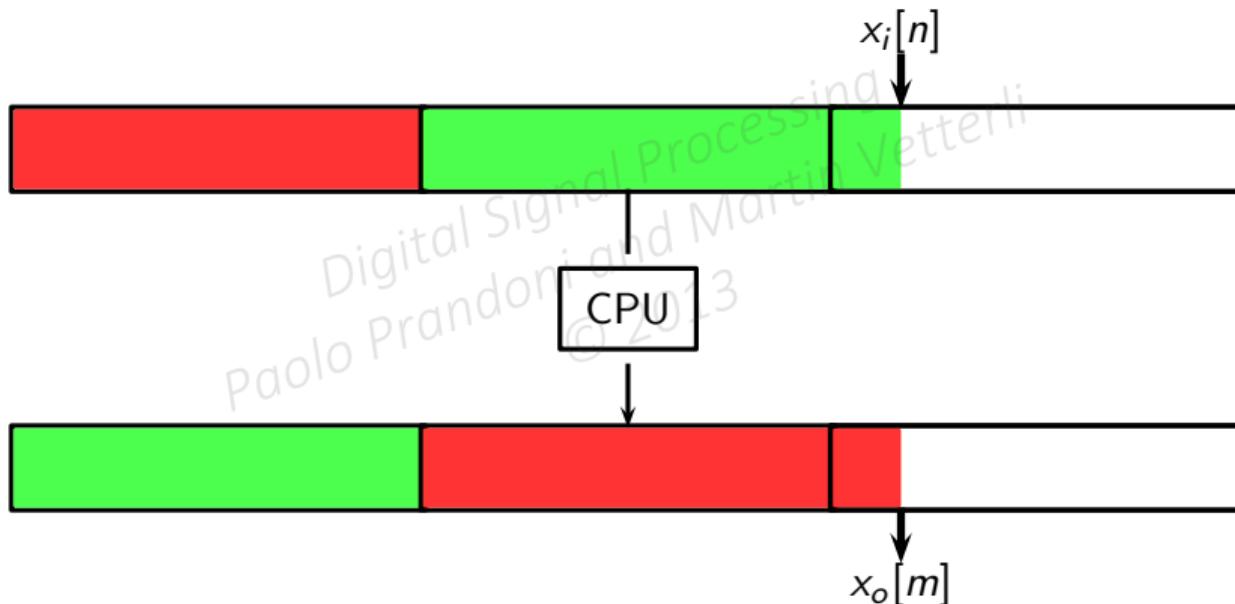
Real-time I/O processing with multiple buffering



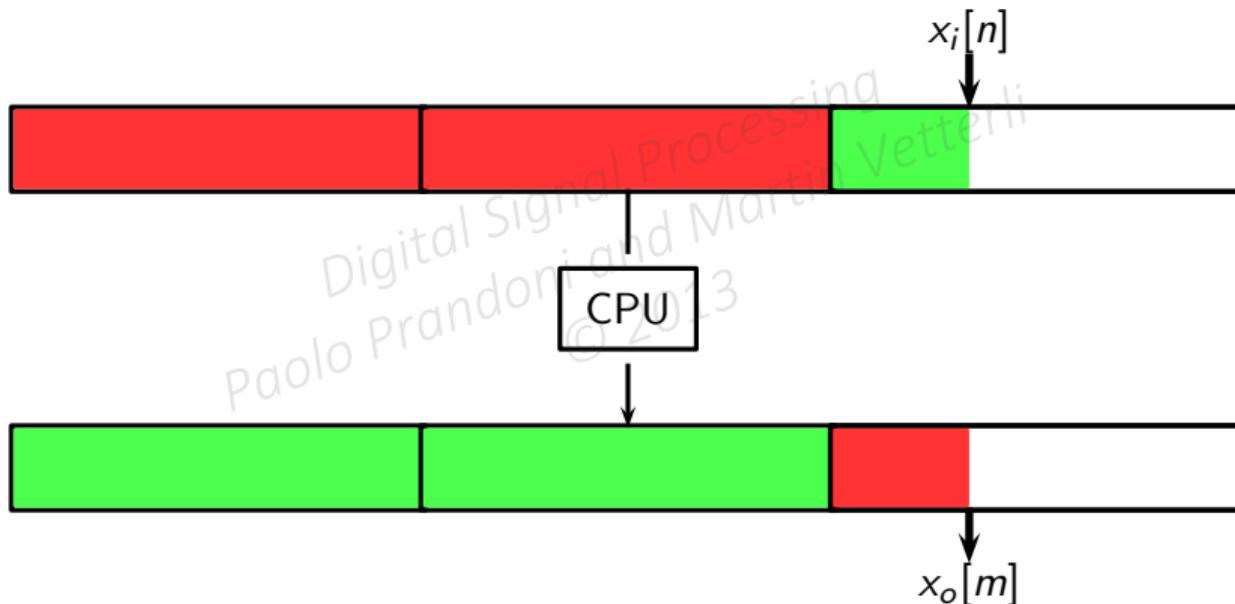
Real-time I/O processing with multiple buffering



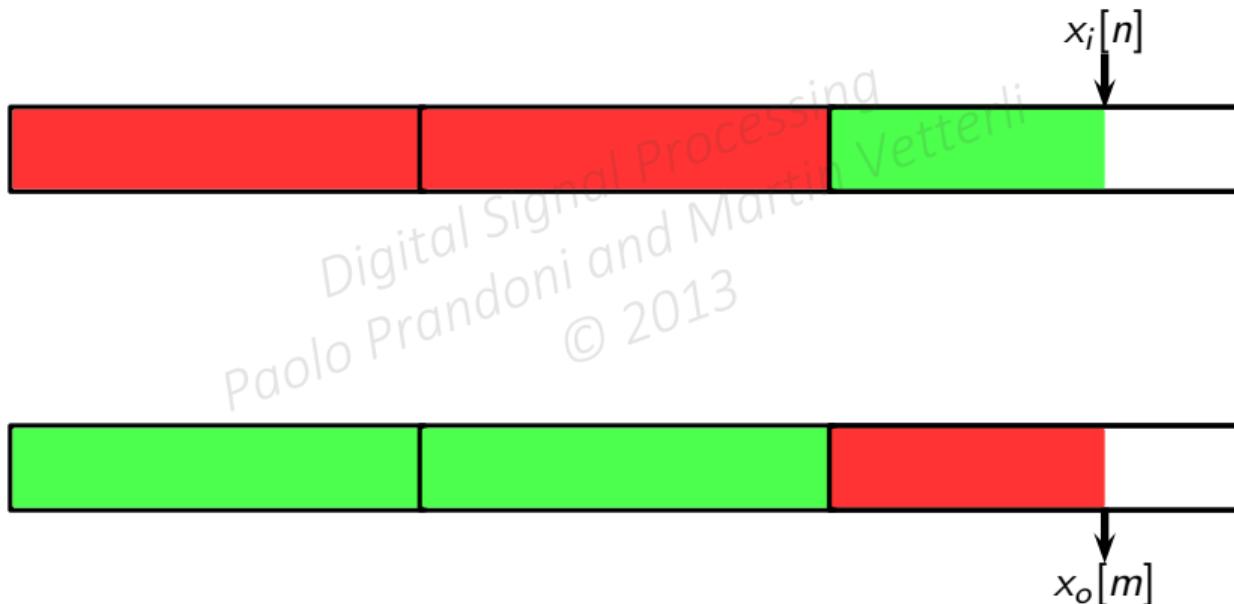
Real-time I/O processing with multiple buffering



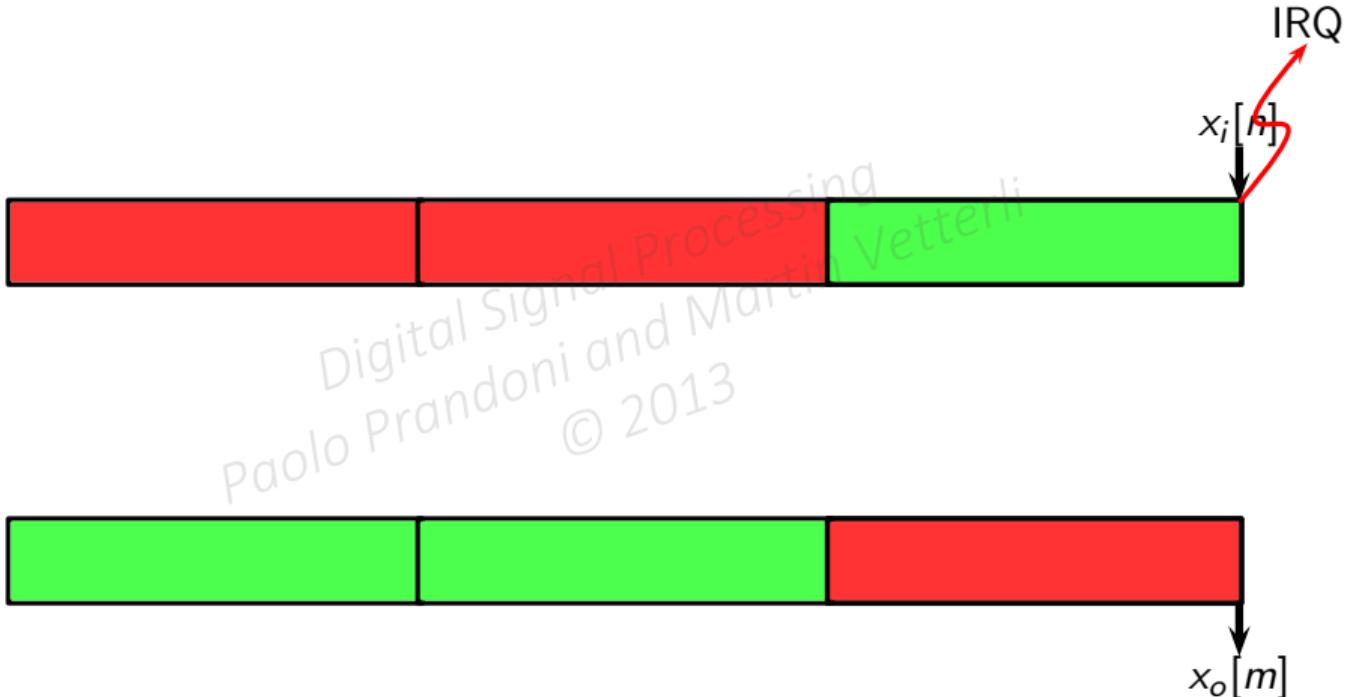
Real-time I/O processing with multiple buffering



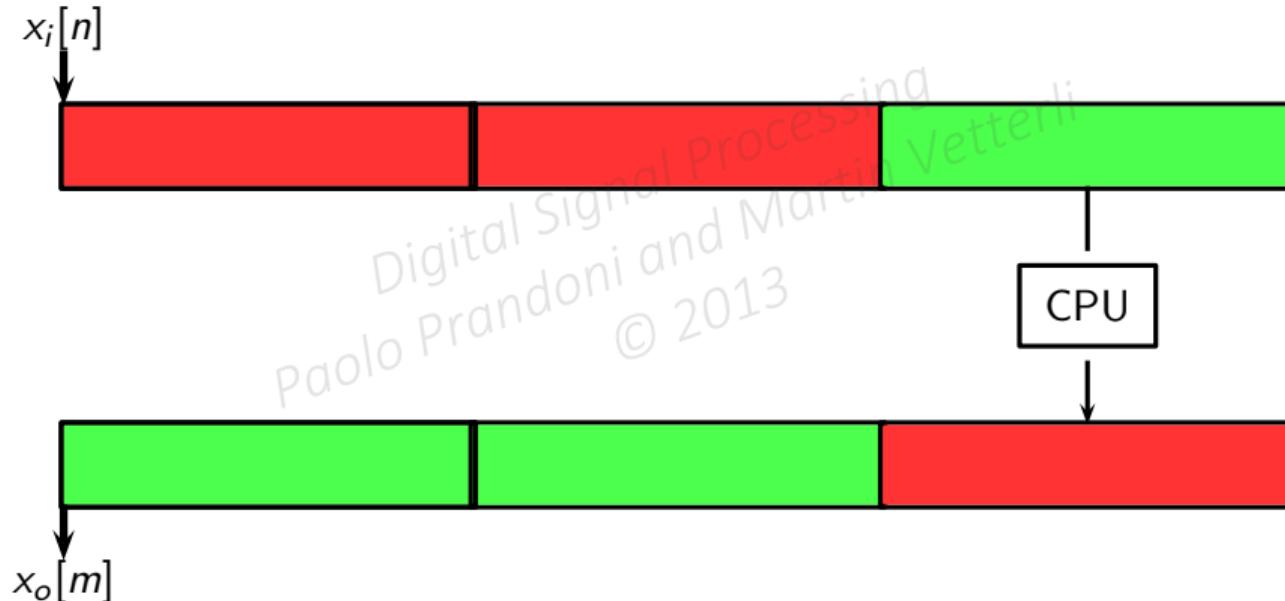
Real-time I/O processing with multiple buffering



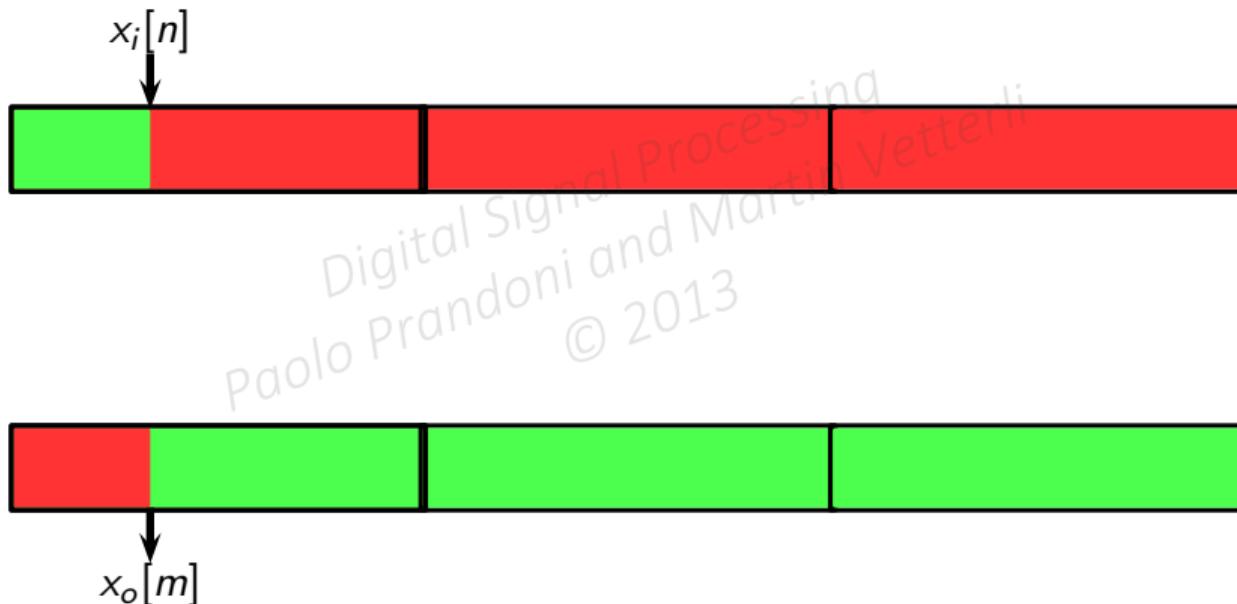
Real-time I/O processing with multiple buffering



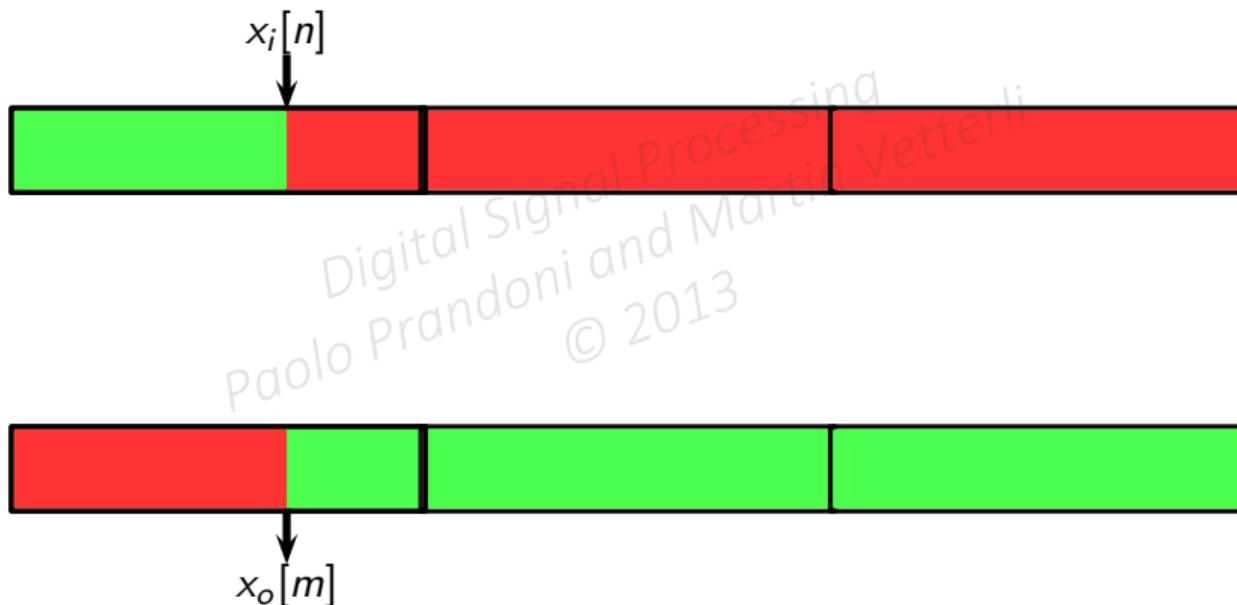
Real-time I/O processing with multiple buffering



Real-time I/O processing with multiple buffering



Real-time I/O processing with multiple buffering



- ▶ total delay $d = T_s \times L$ seconds
- ▶ usually start output process first
- ▶ buffers can be collapsed

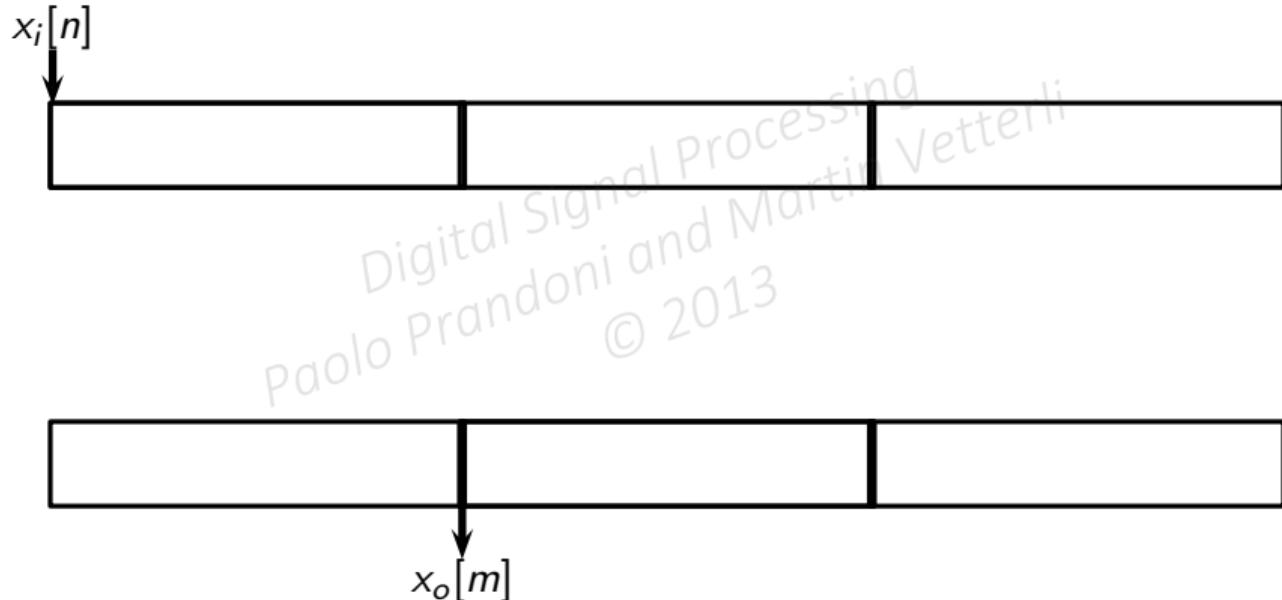
Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ total delay $d = T_s \times L$ seconds
- ▶ usually start output process first
- ▶ buffers can be collapsed

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ total delay $d = T_s \times L$ seconds
- ▶ usually start output process first
- ▶ buffers can be collapsed

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013



Implementation

► low level:

- study soundcard data sheet (each one is different)
- write code to program soundcard via writes to 10 ports
- write an interrupt handler
- write the code to handle the data

► high level:

- choose a good API
- write a callback function to handle the data

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Implementation

► low level:

- study soundcard data sheet (each one is different)
- write code to program soundcard via writes to 16 ports
- write an interrupt handler
- write the code to handle the data

► high level:

- choose a good API
- write a callback function to handle the data

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Implementation

- ▶ low level:
 - study soundcard data sheet (each one is different)
 - write code to program soundcard via writes to IO ports
 - write an interrupt handler
 - write the code to handle the data
- ▶ high level:
 - choose a good API
 - write a callback function to handle the data

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ low level:
 - study soundcard data sheet (each one is different)
 - write code to program soundcard via writes to IO ports
 - write an interrupt handler
 - write the code to handle the data
- ▶ high level:
 - choose a good API
 - write a callback function to handle the data

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

▶ low level:

- study soundcard data sheet (each one is different)
- write code to program soundcard via writes to IO ports
- write an interrupt handler
- write the code to handle the data

▶ high level:

- choose a good API
- write a callback function to handle the data

▶ low level:

- study soundcard data sheet (each one is different)
- write code to program soundcard via writes to IO ports
- write an interrupt handler
- write the code to handle the data

▶ high level:

- choose a good API
- write a callback function to handle the data

▶ low level:

- study soundcard data sheet (each one is different)
- write code to program soundcard via writes to IO ports
- write an interrupt handler
- write the code to handle the data

▶ high level:

- choose a good API
- write a callback function to handle the data

- ▶ low level:
 - study soundcard data sheet (each one is different)
 - write code to program soundcard via writes to IO ports
 - write an interrupt handler
 - write the code to handle the data
- ▶ high level:
 - choose a good API
 - write a callback function to handle the data

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Callback prototype

```
int Callback( const void *input,  
              void *output,  
              unsigned long samples,  
              ...);
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

Callback example

```
int Callback( const void *input,
              void *output,
              unsigned long samples)
{
    float* pIn  = (float *)input;
    float* pOut = (float *)output;

    for (int n = 0; n < samples; n++)
        *pOut++ = Process(*pIn++);
}
```

paolo Prandoni and Martin Vetterli
Digital Signal Processing
© 2013

```
// 10 sec @ 24KHz
enum {BUF_LEN = 0x10000};
enum {BUF_MASK = BUF_LEN -1};

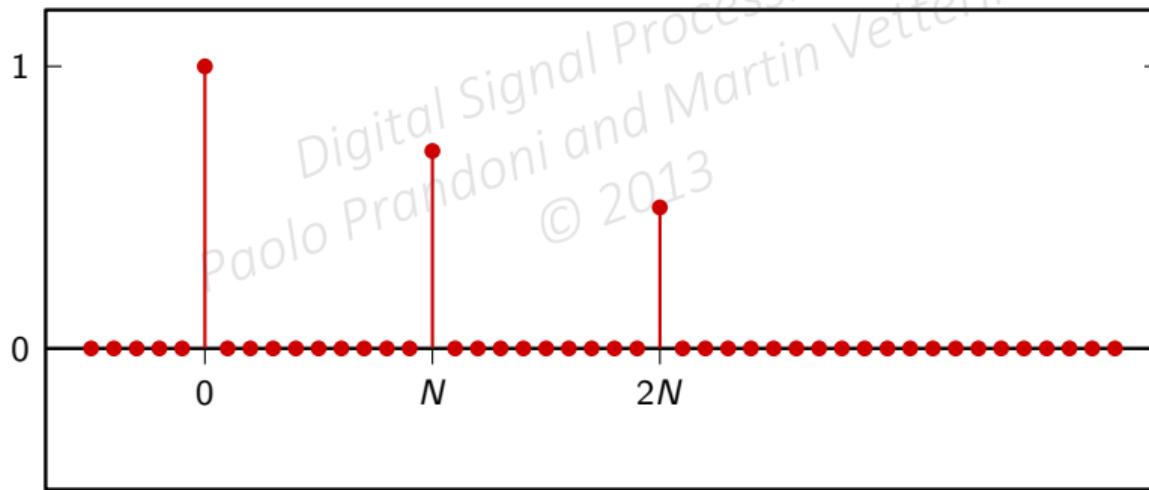
float m_pY[BUF_LEN];
float m_pX[BUF_LEN];
int m_Ix;
int m_Iy;
```

```
float Process(float x)
{
    m_pX[m_Ix] = Sample;
    float y = Effect();
    m_pY[m_Iy] = y;
    m_Ix = (m_Ix + 1) & BUF_MASK;
    m_Iy = (m_Iy + 1) & BUF_MASK;

    return y;
}
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

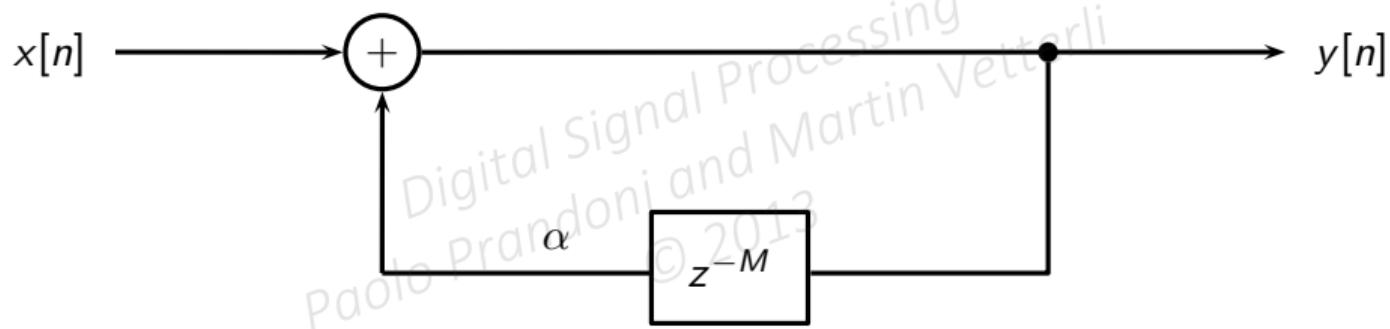
$$y[n] = \frac{a x[n] + b x[n - N] + c x[n - 2N]}{a + b + c}$$



```
float Echo()
{
    static float a = 1;
    static float b = 0.7f;
    static float c = 0.5f;
    static float norm = 1.0f / (a+b+c);
    static int N = (int)(0.3 * m_SR);

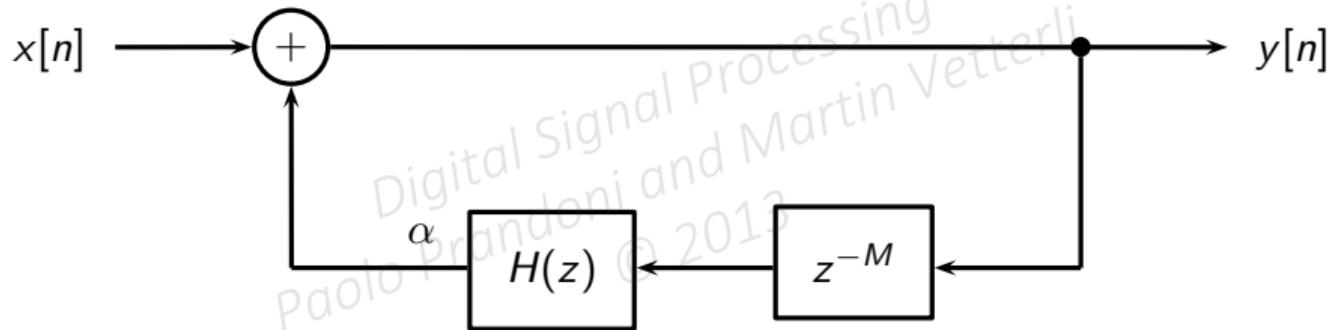
    return norm * ( a * m_pX[m_Ix]
                    + b * m_pX[(m_Ix + BUF_LEN - N) & BUF_MASK]
                    + c * m_pX[(m_Ix + BUF_LEN - 2*N) & BUF_MASK]);
}
```

remember the KS algorithm? it's a sort of IIR echo



$$y[n] = \alpha y[n - M] + x[n]$$

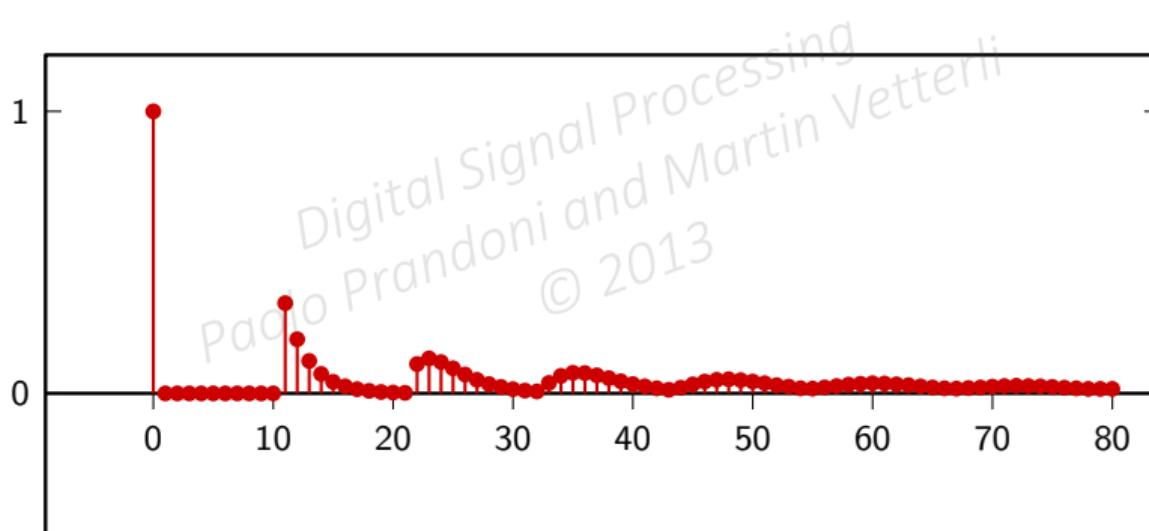
a natural echo has a lowpass characteristic



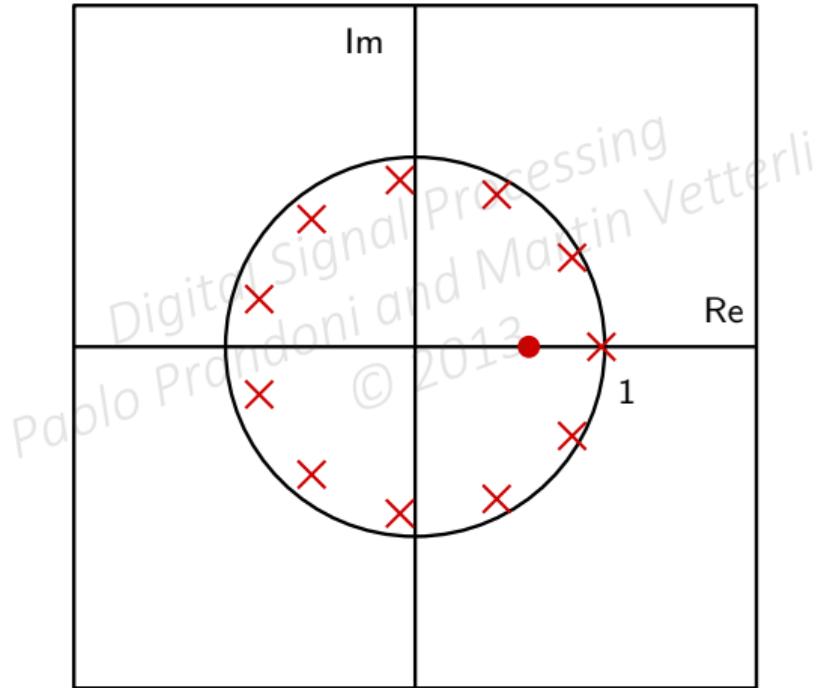
$$y[n] = \alpha(h * y)[n - M] + x[n]$$

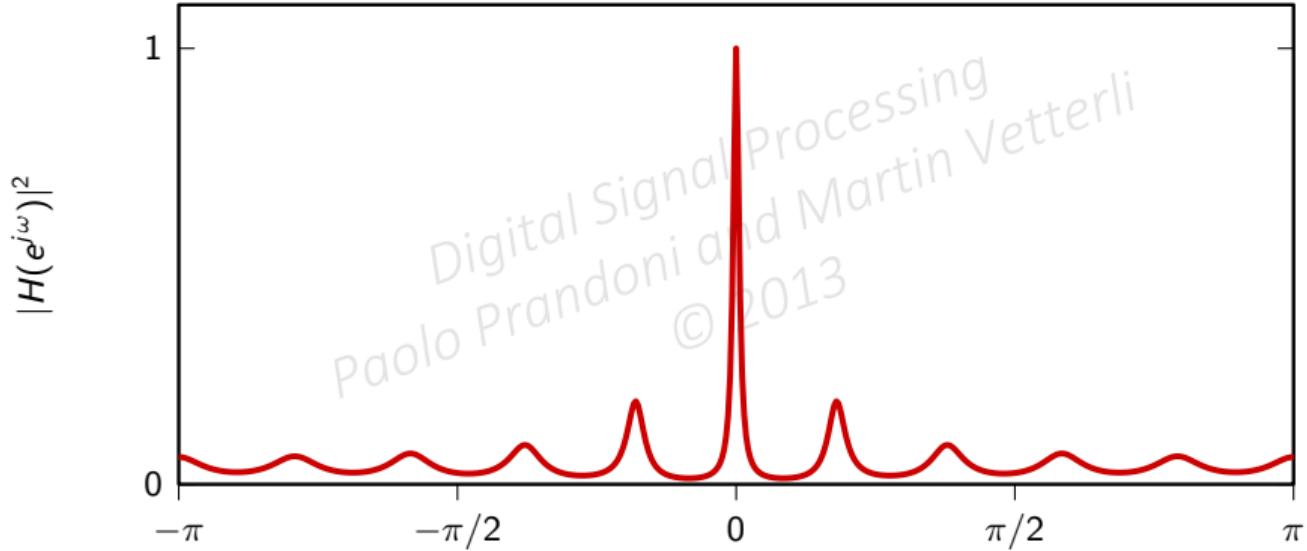
Choose for instance $H(z) = \text{leaky integrator}$:

$$y[n] = x[n] - \lambda x[n-1] + \lambda y[n-1] - \alpha(1-\lambda)y[n-N]$$



$$N = 10, \lambda = 0.6, \alpha = 0.8$$





```
float NaturalEcho()
{
    static float a = 0.7;
    static float L = 0.6;
    static float norm = 1.0f / (1+a);
    static int N = (int)(0.3 * m_SR);

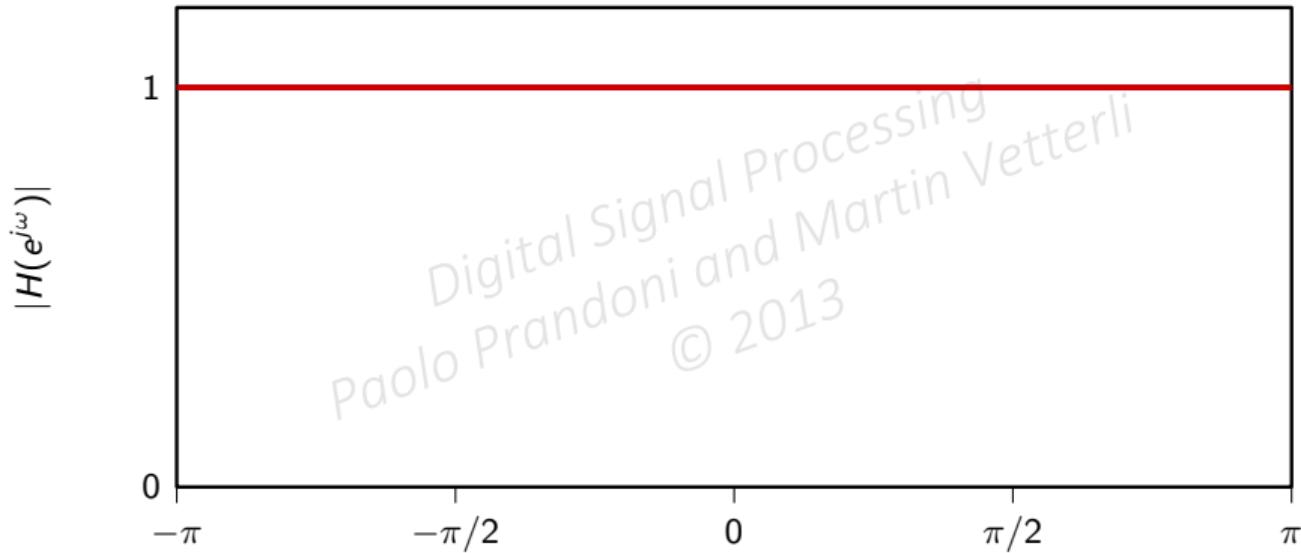
    return norm * (
        m_pX[m_Ix]
        - L *      m_pX[(m_Ix + BUF_LEN - 1) & BUF_MASK]
        + L *      m_pY[(m_Iy + BUF_LEN - 1) & BUF_MASK]
        + a*(1-L) * m_pY[(m_Iy + BUF_LEN - N) & BUF_MASK]);
}
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

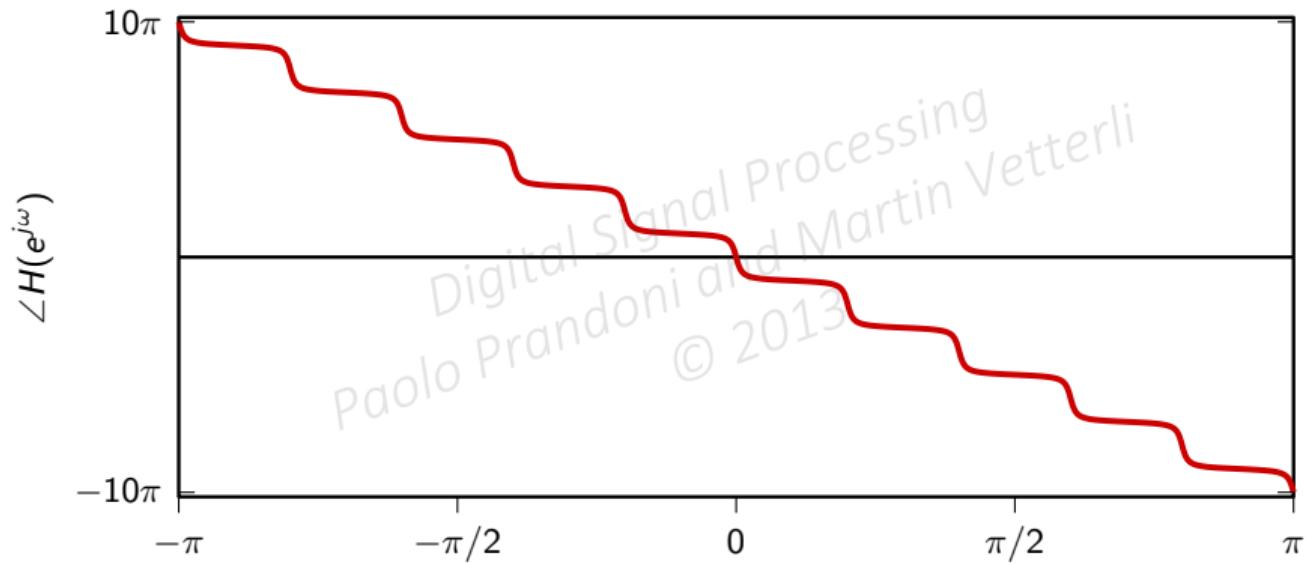
- ▶ reverb is given by the superposition of many many echos with different delays and magnitudes
- ▶ many ways to simulate, always rather costly
- ▶ a cheap alternative is to use an allpass filter

$$H(z) = \frac{-\alpha + z^{-N}}{1 - \alpha z^{-N}}$$

Reverb, magnitude response



Reverb, phase response



```
float Reverb()
{
    static float a = 0.8;
    static int N = (int)(0.006 * m_SR);

    return ( - a * m_pX[m_Ix]
            +      m_pX[(m_Ix + BUF_LEN - N) & BUF_MASK]
            + a * m_pY[(m_Iy + BUF_LEN - N) & BUF_MASK]);
}
```

Digital Signal Processing
Francesco Brandomi and Martin Vetterli
@ 2013

- distortion: clip the signal

$$y[n] = \text{trunc}(ax[n])/a$$

- tremolo: sinusoidal amplitude modulation

$$\text{Digital Signal Processing}$$
$$x[n] = (1 + \cos(\omega_0 n)/G)x[n]$$

- flanger: sinusoidal delay

Pdolo Prandoni and Martin Vetterli
© 2013

$$y[n] = x[n] + x[n - \lfloor d(1 + \cos(\omega_0 n)) \rfloor]$$

- ▶ distortion: clip the signal

$$y[n] = \text{trunc}(ax[n])/a$$

- ▶ tremolo: sinusoidal amplitude modulation

$$y[n] = (1 + \cos(\omega_0 n)/G)x[n]$$

- ▶ flanger: sinusoidal delay

$$y[n] = x[n] + x[n - \lfloor d(1 + \cos(\omega_0 n)) \rfloor]$$

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

- ▶ distortion: clip the signal

$$y[n] = \text{trunc}(ax[n])/a$$

- ▶ tremolo: sinusoidal amplitude modulation

$$y[n] = (1 + \cos(\omega_0 n)/G)x[n]$$

- ▶ flanger: sinusoidal delay

$$y[n] = x[n] + x[n - \lfloor d(1 + \cos(\omega_0 n)) \rfloor]$$

```
float Fuzz()
{
    static float limit = 0.005;
    static float G = 5;

    float y = m_pX[m_Ix];
    if (y > limit)
        y = limit;
    if (y < -limit)
        y = -limit;
    return G*y;
}
```

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

```
float Tremolo()
{
    static double phi = 5 * 2*PI / m_SR; // 5Hz LFO
    static double omega = 0;

    omega = omega + phi;
    return (1 + cos(omega)/4) * m_pX[m_Ix];
}
```

Digital Signal Processing
Pierluigi Prandoni and Martin Vetterli
© 2013

```
float Flanger()
{
    static int N = (int)(0.002 * m_SR);    // max delay
    static double phi = 0.1 * 2*PI / m_SR;   // 0.1Hz LFO
    static double omega = 0;

    int d = (int)(N * (1 + cos(omega)));
    omega = omega + phi;
    return 0.5f + (m_pX[m_Ix] + m_pX[(m_Ix + BUF_LEN - d) & BUF_MASK]);
}
```

END OF MODULE 5.11

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

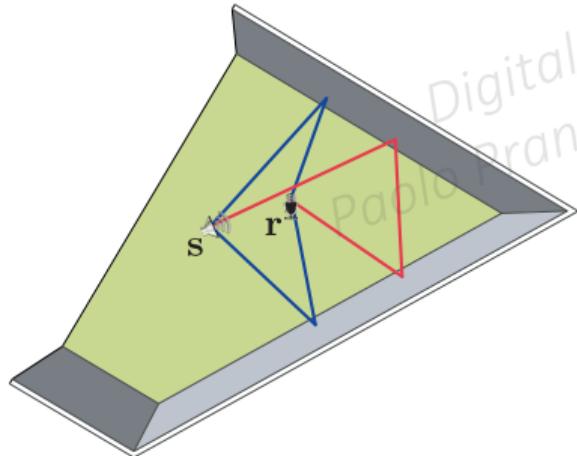
Digital Signal Processing

Module 5.12: Dereverberation and Echo Cancelation

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

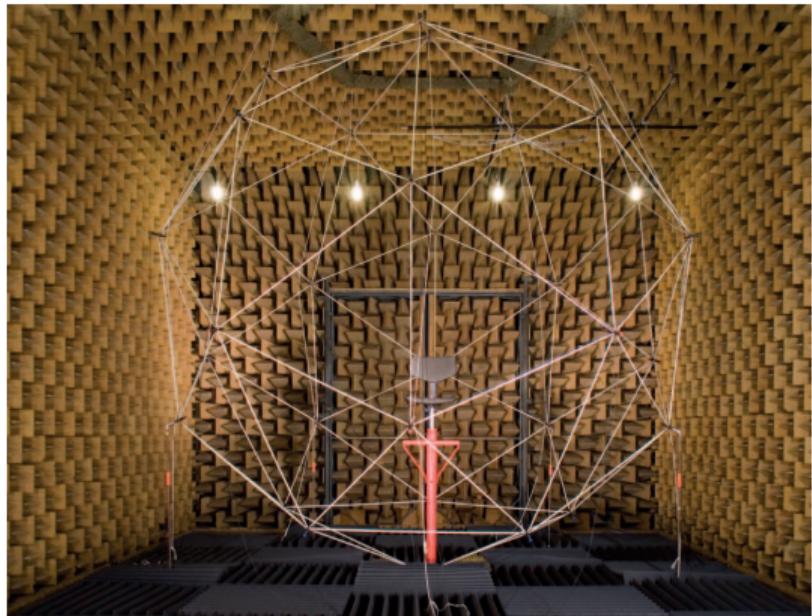
Guest lecture by
Ivan Dokmanić

- ▶ Free space propagation, sound pressure $\sim \frac{1}{\text{distance}}$
- ▶ Every reflection attenuates the sound by $\alpha \in [0, 1]$



- ▶ Room is a linear filter, it acts by convolution
- ▶ $y_{\text{listener}}[n] = x_{\text{source}}[n] * h[n; \text{positions}]$
- ▶ We will suppress the dependence on source and listener positions for simplicity

Anechoic Chamber



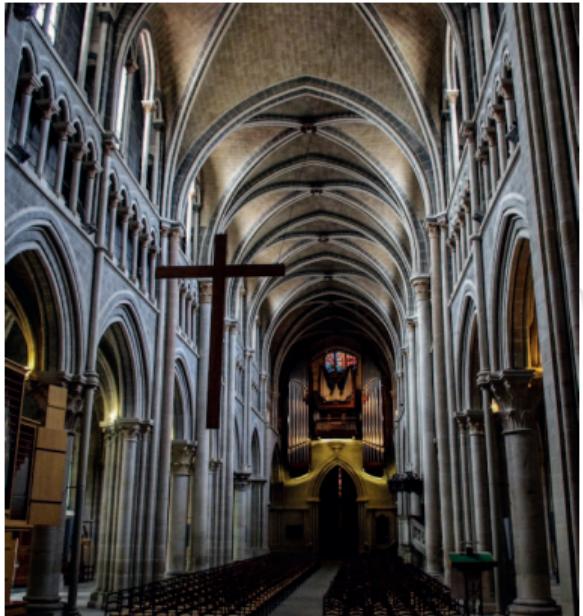
processing
Martin Vetterli
2013

- ▶ Sound in this room



Digital Signal Processing
and Martin Vetterli
2013

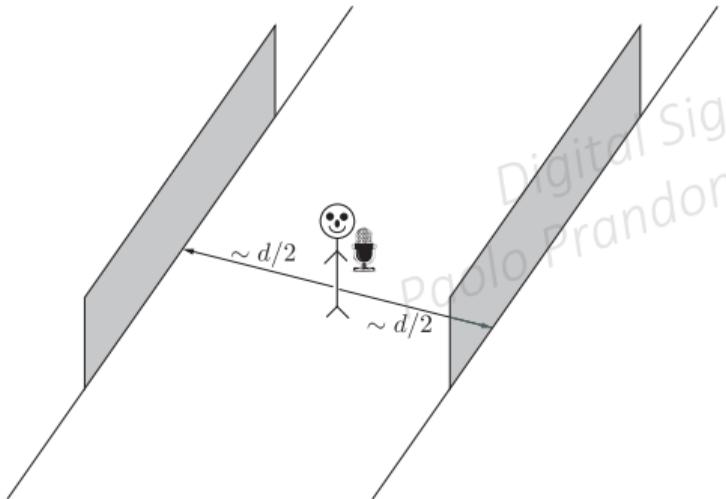
- ▶ Impulse response
- ▶ Sound in this room



nal Processing
i and Martin Vetterli
© 2013

- ▶ Impulse response
- ▶ Sound in this room

- ▶ Consider (almost) colocated speaker and microphone between two walls at distance d



- ▶ The room impulse response is

$$h[n] = \sum_{k=0}^{\infty} \frac{\alpha^k}{B(k + \epsilon)T} \delta[n - kN]$$

where

$$N = \text{round}(T \times F_s) = \text{round}\left(\frac{d}{c}F_s\right)$$

and $c \approx 340\text{m/s}$ is the speed of sound

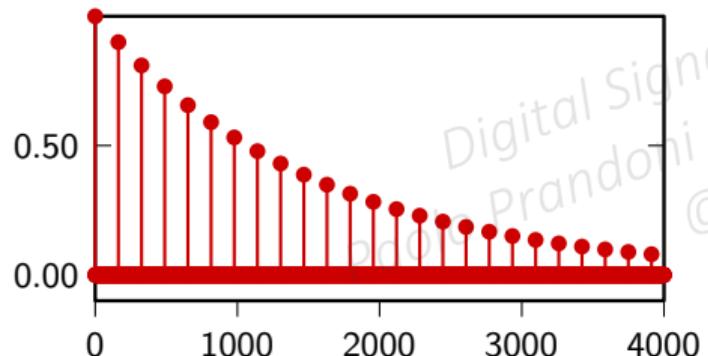
- ▶ $\frac{1}{k}$ term is difficult to handle in the z -domain, so we'll simplify
- ▶ Assume first that the dominant attenuation is due to reflections

$$h_a[n] = \sum_{k=0}^{\infty} \alpha^k \delta[n - kN]$$

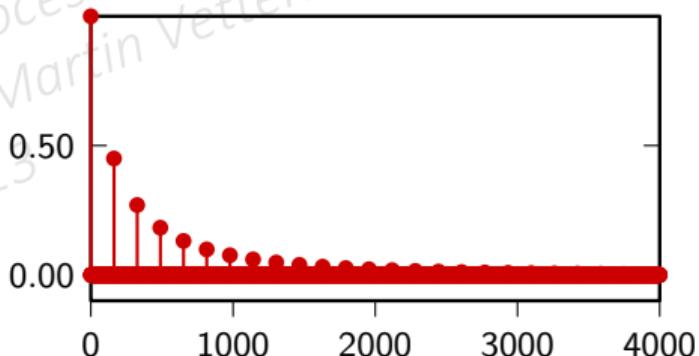
- ▶ This has a simple z -transform!

How do These Sound and Look?

- ▶ Simplified and realistic impulse responses for $F_s = 44100$ Hz, $d = 7$ m, $\alpha = 0.9$
- ▶ Original sound



Simplified room impulse response



Realistic room impulse response

- ▶ Reverberated sound is $y[n] = (h_a * x)[n]$
- ▶ We want to design a filter $h_i[n]$ such that $(h_i * y)[n] = x[n]$, that is

$$x = h_i * y = h_i * (h_a * x) = (h_i * h_a) * x.$$

so that $(h_i * h_a)[n] = \delta[n]$, because $(\delta * x)[n] = x[n]$

- ▶ In the z -domain this becomes $H_i(z)H_a(z) = 1$ and we get that $H_i(z) = \frac{1}{H_a(z)}$

- ▶ Let's get rid of the room! The transfer function is computed as

$$\begin{aligned} H_a(z) &= \sum_{n=0}^{\infty} h_a[n] z^{-n} \\ &= \sum_{n=0}^{\infty} \left(\sum_{k=0}^{\infty} \alpha^k \delta[n - kN] \right) z^{-n} \\ &= \sum_{n=0}^{\infty} \alpha^n z^{-nN} = \frac{1}{1 - \alpha z^{-N}} \end{aligned}$$

Digital signal processing
Paolo Prandoni and Martin Vetterli
© 2013

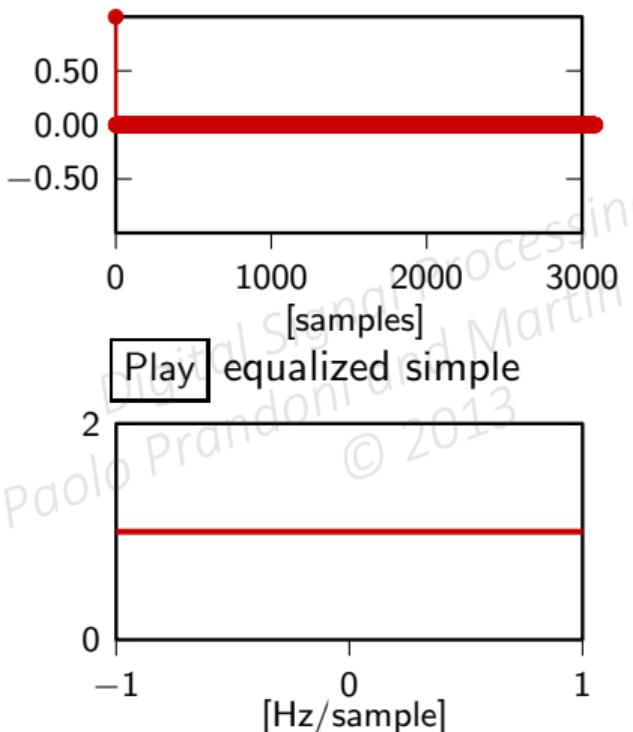
- ▶ The inverse is

$$H_i(z) = H_a(z)^{-1} = 1 - \alpha z^{-N} \leftrightarrow h_i[n] = \delta[n] - \alpha \delta[n - N]$$

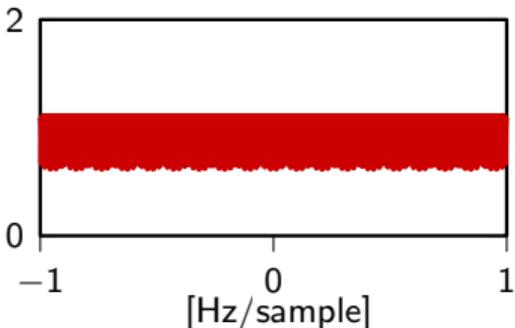
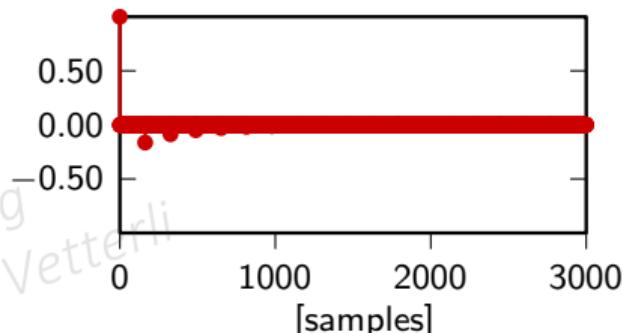
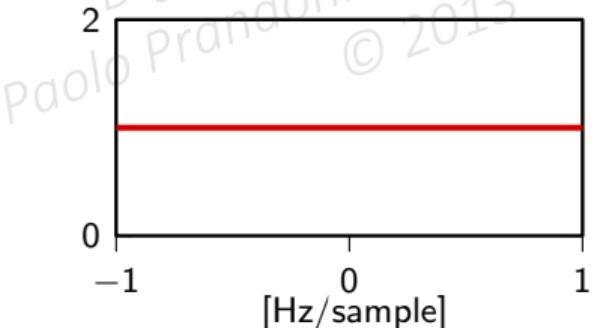
- ▶ But this is a simple FIR filter!
- ▶ For motivated students: observation that exponentials are canceled by simple FIR filters is essential for the so-called Finite Rate of Innovation sampling

Equalized Room, Sounds and Plots

Equalized room
impulse responses

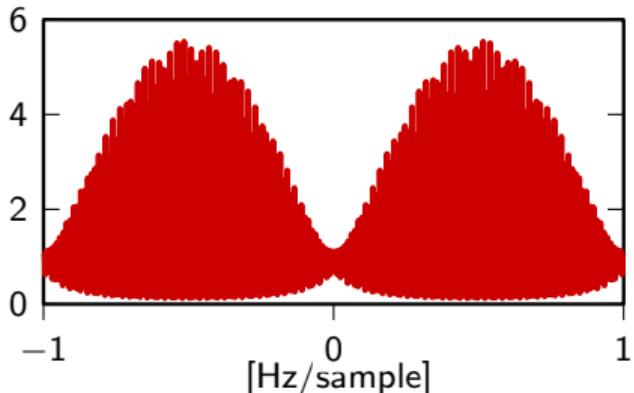
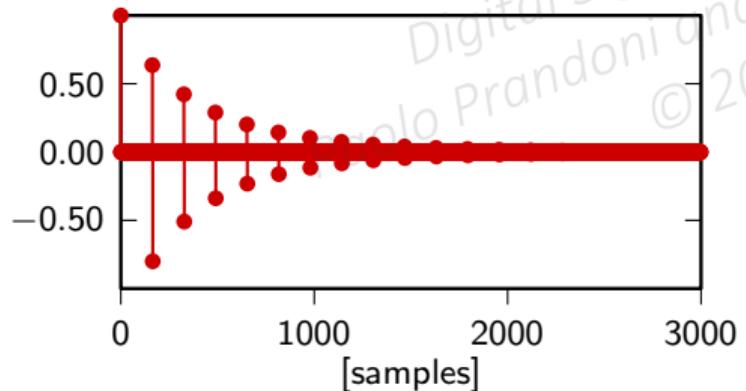


Magnitude of
equalized room
transfer functions



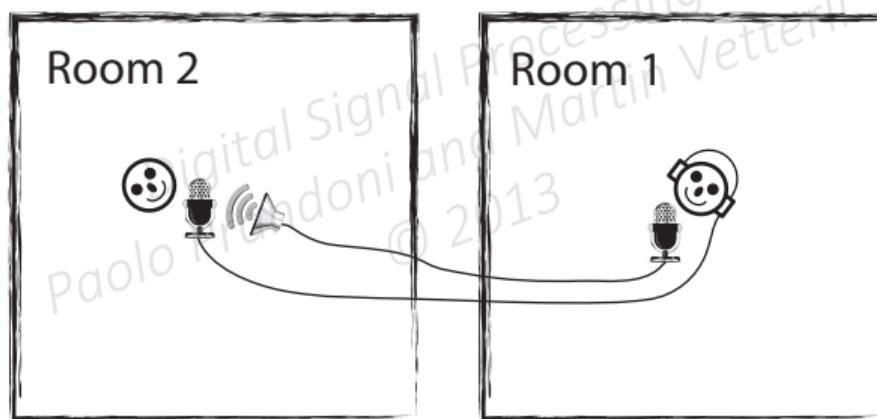
Equalized Room, Model Mismatch

- ▶ Let's hear a different kind of model mismatch: 1% error in room size
- ▶ original sound
- ▶ “equalized”, 1% error in d

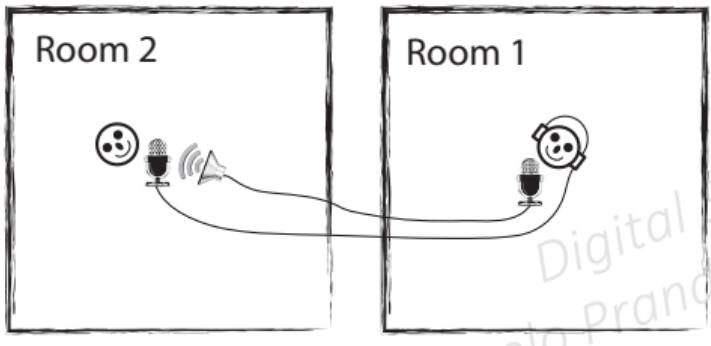


A Teleconference Call

- ▶ Consider now two people talking over a phone, as shown in the figure



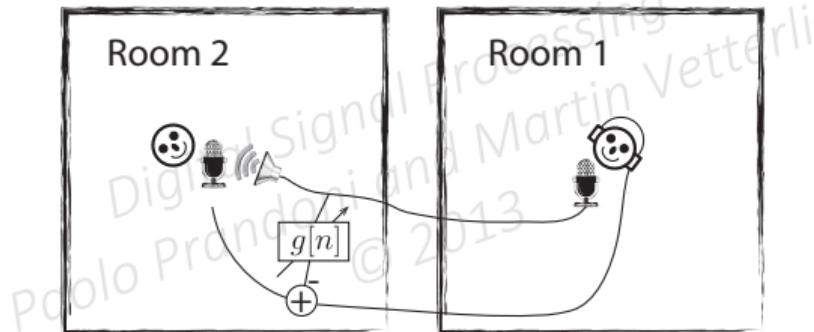
A Teleconference Call



- ▶ Room 1 and Room 2 equal, $d = 7$ m, $\alpha = 0.8$, delay = 0.3 s

- ▶ Person in room one talks into the microphone
- ▶ Signal gets transmitted into another room with a delay and reproduced over a loudspeaker
- ▶ This gets convolved with the room 2, transmitted back, and reproduced over headphones,

- ▶ Must estimate and update the filter $g[n]$ that models Room 2, the loudspeaker and the microphone (not trivial!)



- ▶ Only subtracting the incoming sound (that is, setting $g[n] = \delta[n]$) is not enough,
- ▶ With ideal knowledge of $g[n]$, we get the expected result, only Room 1 remains

Is it Really This Simple?



- ▶ What if you are not halfway between the walls?

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

END OF MODULE 5.12

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013

END OF MODULE 5

Digital Signal Processing
Paolo Prandoni and Martin Vetterli
© 2013