

FFT analysis in practice

Perception & Multimedia Computing
Lecture 13

Rebecca Fiebrink

Lecturer, Department of Computing
Goldsmiths, University of London



Last Week

- Review of complex numbers: rectangular and polar representations
- The complex exponential
- The Fourier Series and Discrete Fourier Transform (DFT)
- The Fast Fourier Transform (FFT)
- Lab:
 - Understanding convolution and systems through hands-on practice
 - Signals and convolution in R



Today

- Brief lab discussion (more tomorrow)
- Brief coursework discussion
- Using FFT in practice
 - Choosing parameters and interpreting output
 - Short-time Fourier Transform
 - Example applications
 - Variants of Fourier transform

Lab discussion

- Convolution by hand: Examples
- $h=[3,2,4]$:
 - same as
 $[3, 0, 0] + [0, 2, 0] + [0, 0, 4]$
 - same as
 $3[1] + 2[0, 1] + 4[0, 0, 1]$
 - same as
 $3[1] + 2T_1\{[1]\} + 4T_2\{[1]\}$
- $x * h = x * 3[1] + x * 2T_1\{[1]\} + x * 4T_2\{[1]\}$
- Example on board

The FFT in Practice



Fast Fourier Transform (FFT)

Review

- Given a signal, what is its frequency content?
 - Helps us understand audio content (pitch, timbre, melody, rhythm, genre, speech, ...)
- Also a building block for designing and understanding effects (filters, equalization, reverb, echo)
- One of the most powerful and useful techniques for working with audio, image, and video!

Fast Fourier Transform (FFT)

Review

- Equation:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$$

- Essentially, dot-product multiply our signal x with complex exponentials with periods of N , $N/2$, $N/3$, ... 2 samples (i.e., frequencies of $1/N$, $2/N$, $3/N$, ... $1/2$ oscillations per sample), as well as DC component

Fast Fourier Transform (FFT)

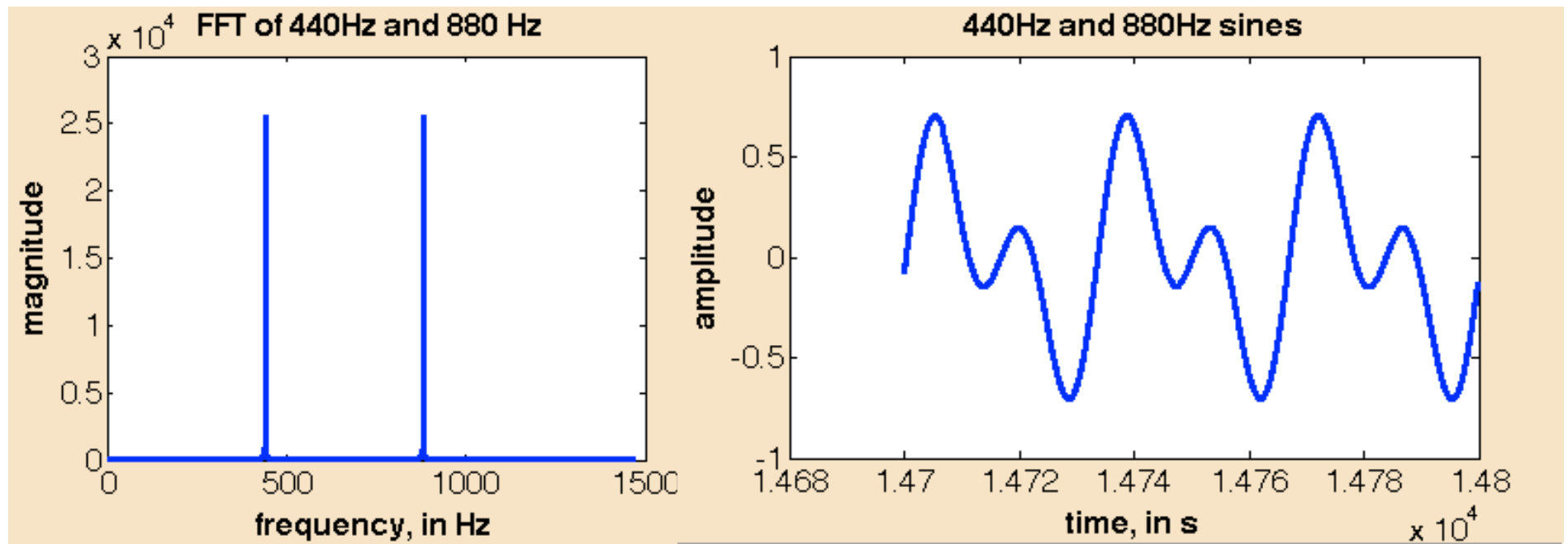
Review

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$$

- Each X_k is a complex number (e.g., $10+5i$, or $3\angle\pi/2$)
- If the k^{th} frequency is present in the signal, X_k will have non-zero magnitude, and its magnitude and phase will tell us how much of that frequency is present and at what phase (though not directly)

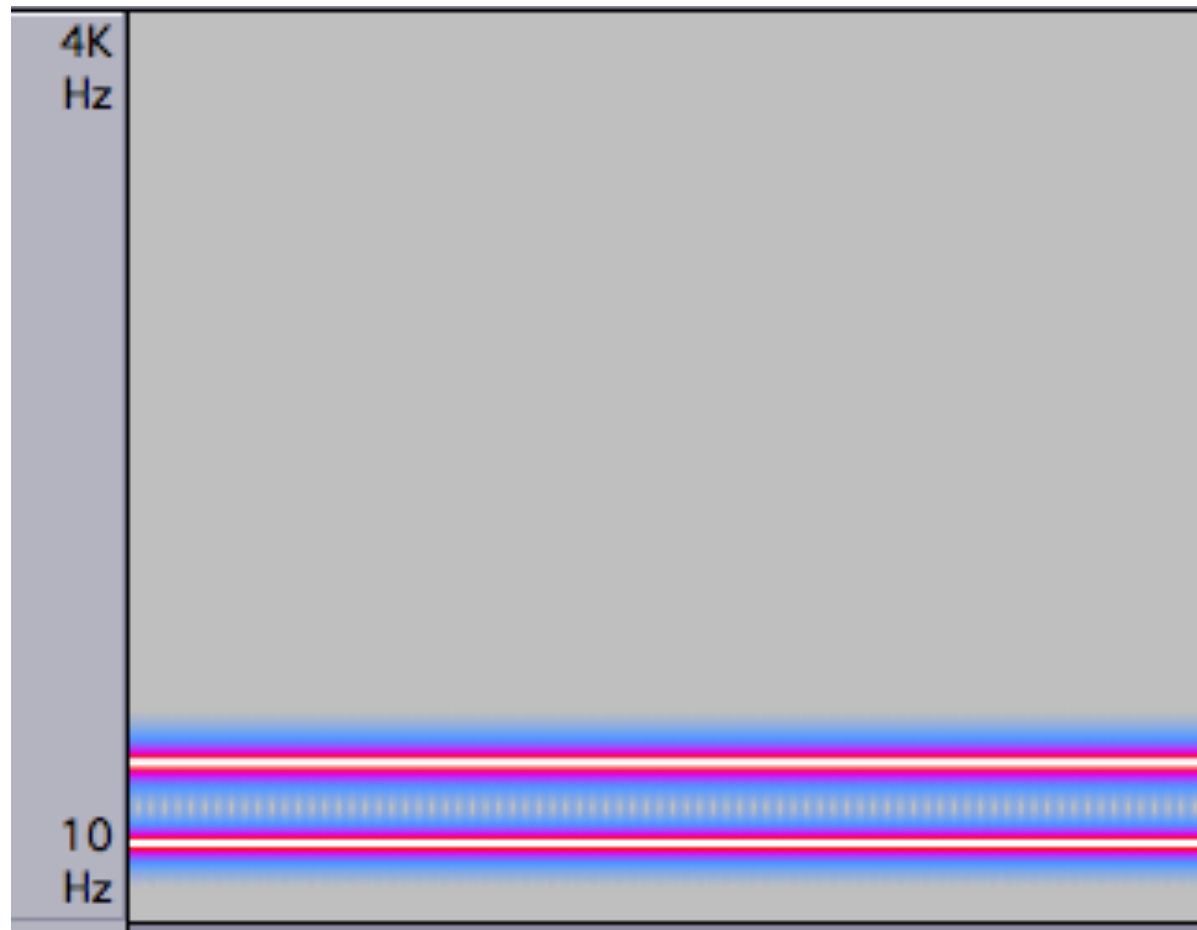
Viewing FFT output

1) Spectrum

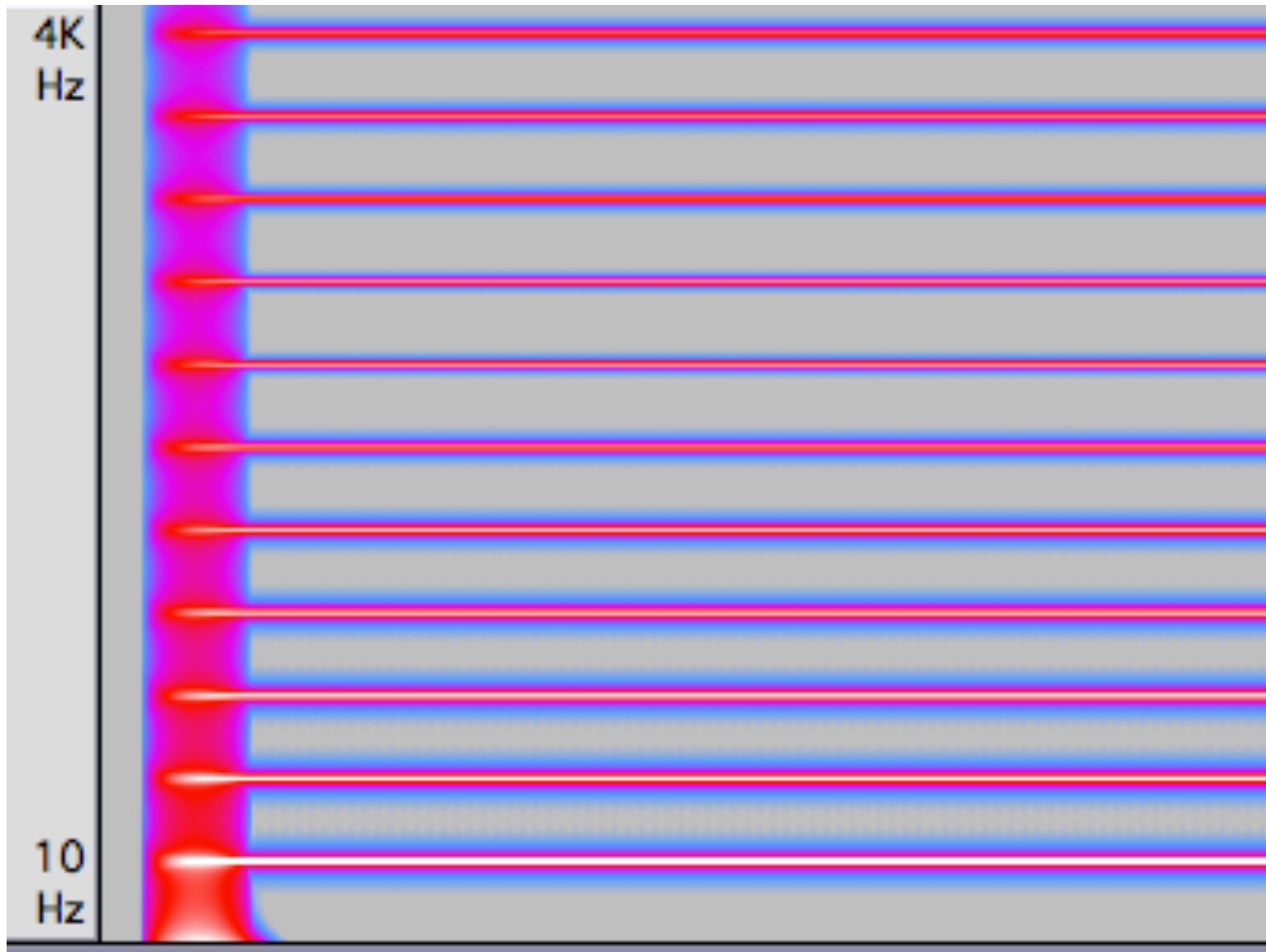


Viewing FFT output

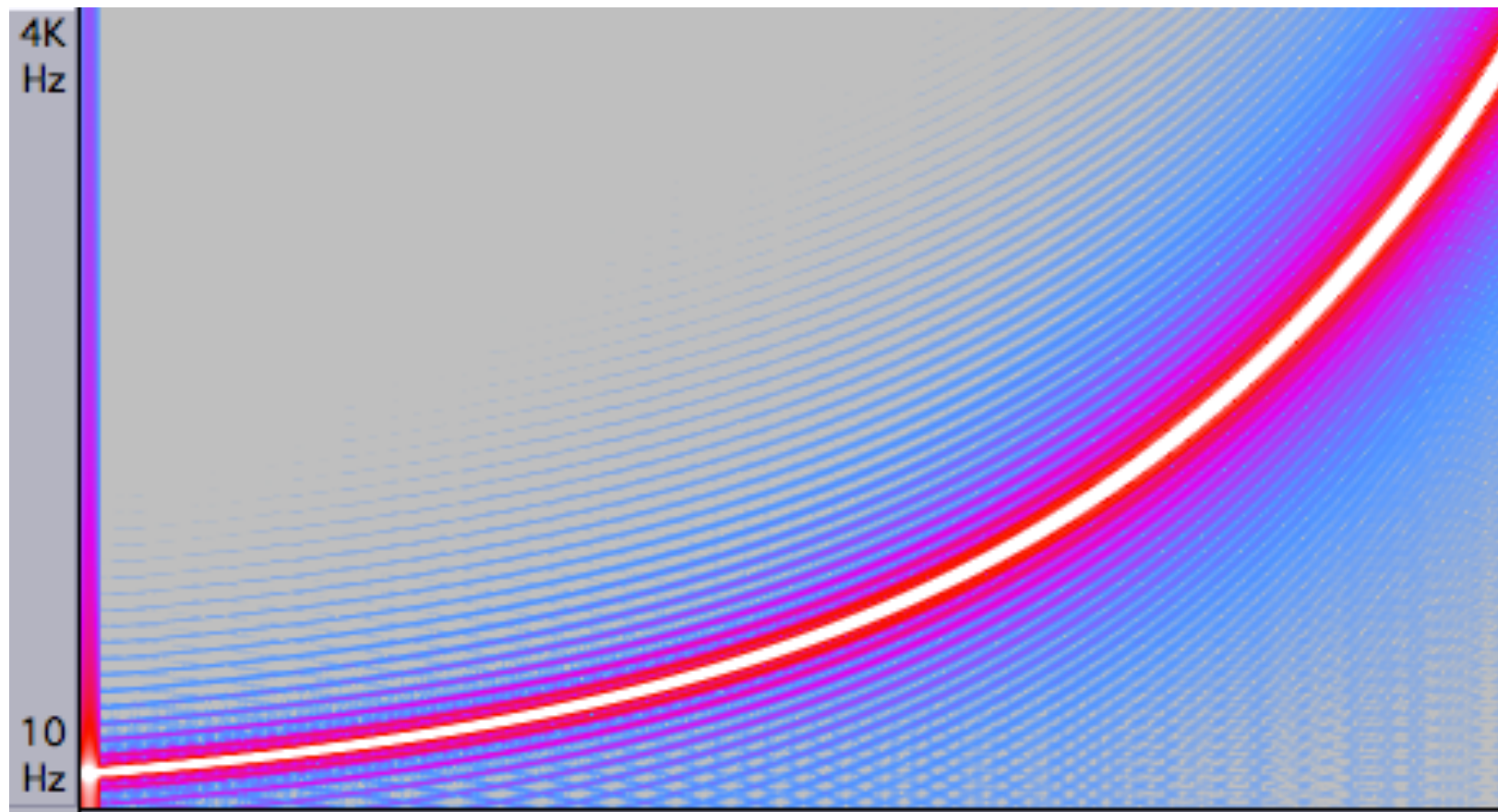
2) Spectrogram



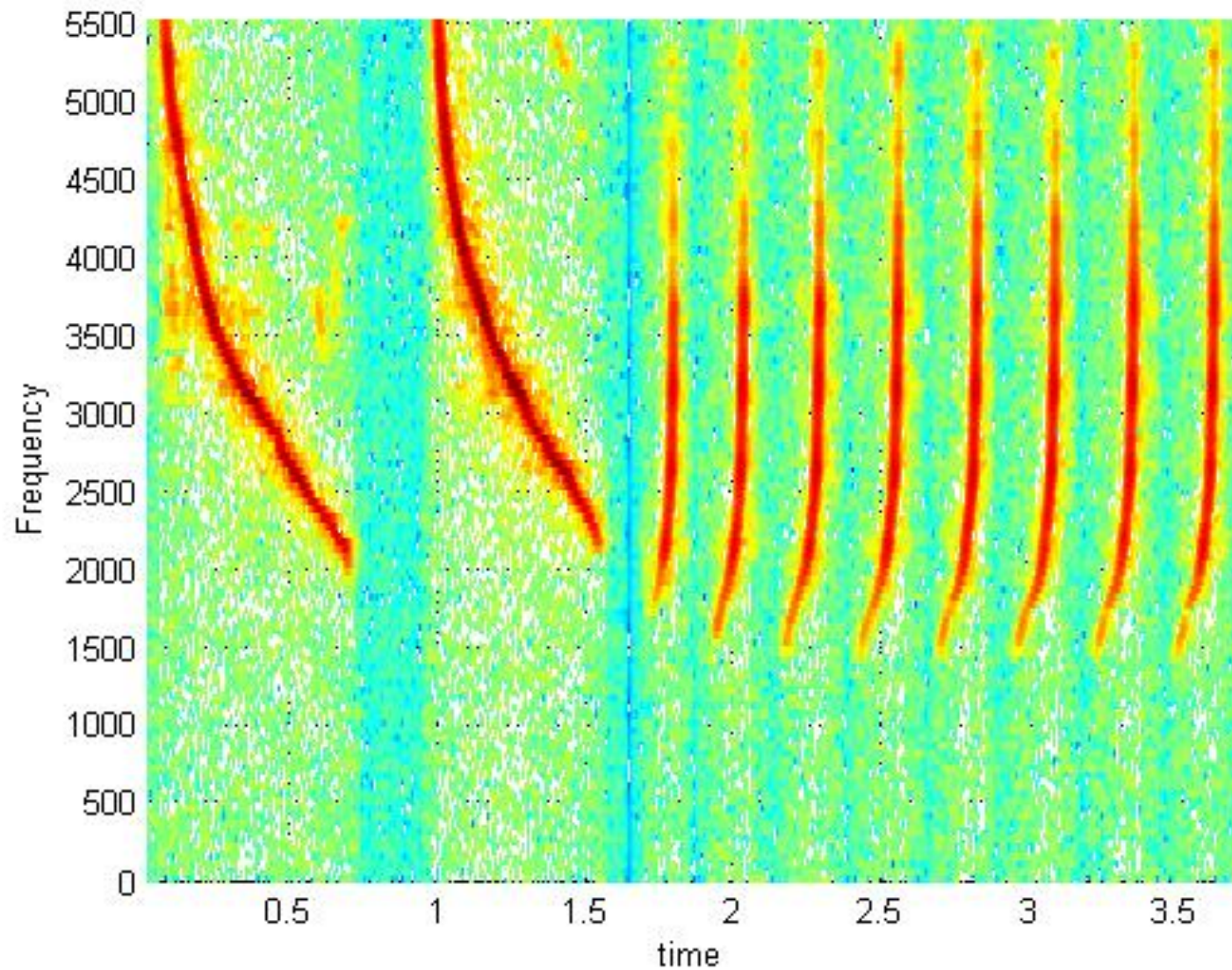
What will you hear?




What will you hear?



What will you hear?

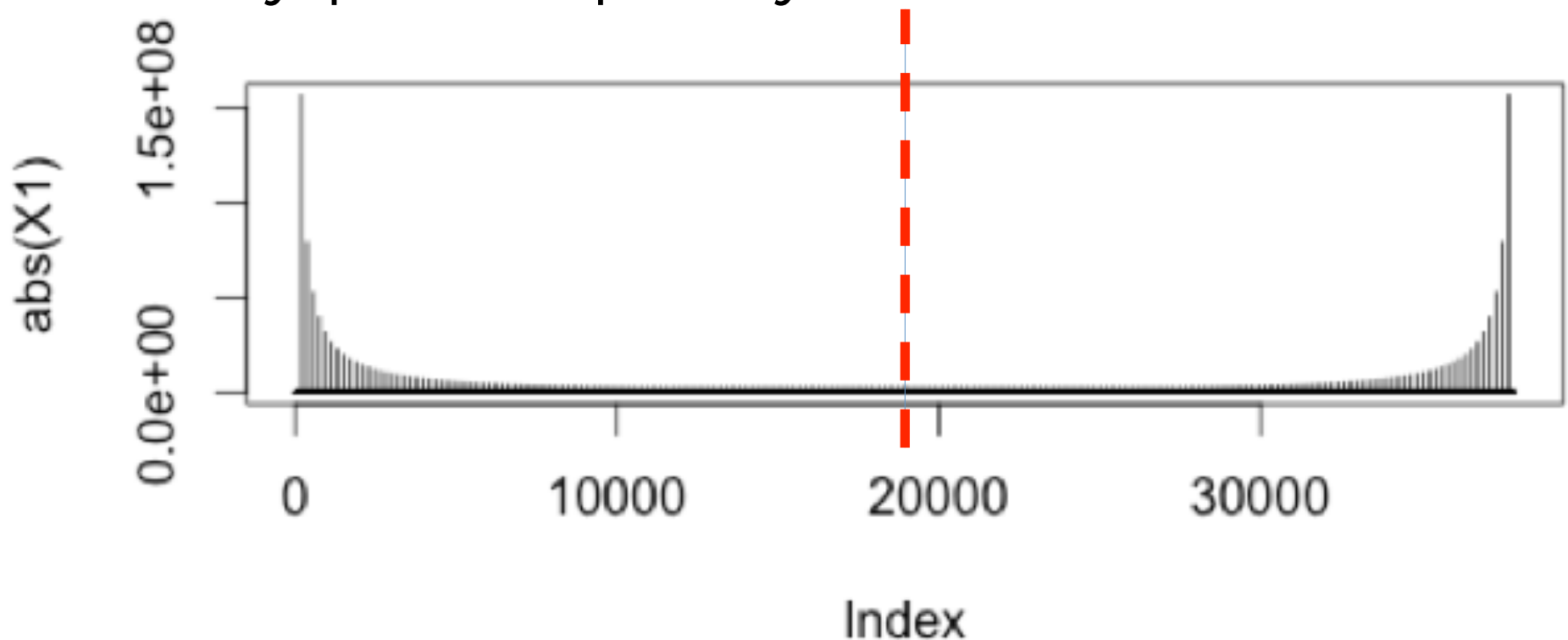


What is *really* output

- N-point FFT computes **N complex values**
- X_0 to X_{N-1} , representing frequencies of 0Hz to $(N-1)/N * \text{SampleRate}$
 $0\text{Hz}, (1/N)*\text{SR}, (2/N)*\text{SR}, \dots (N/2)/N*\text{SR}, \dots (N-1)/N*\text{SR}$
 **$=1/2*\text{SR}$ (Nyquist)**
- These frequencies often called “bins” of FFT
- Note that adjacent bins are $(1/N)*\text{SR}$ apart

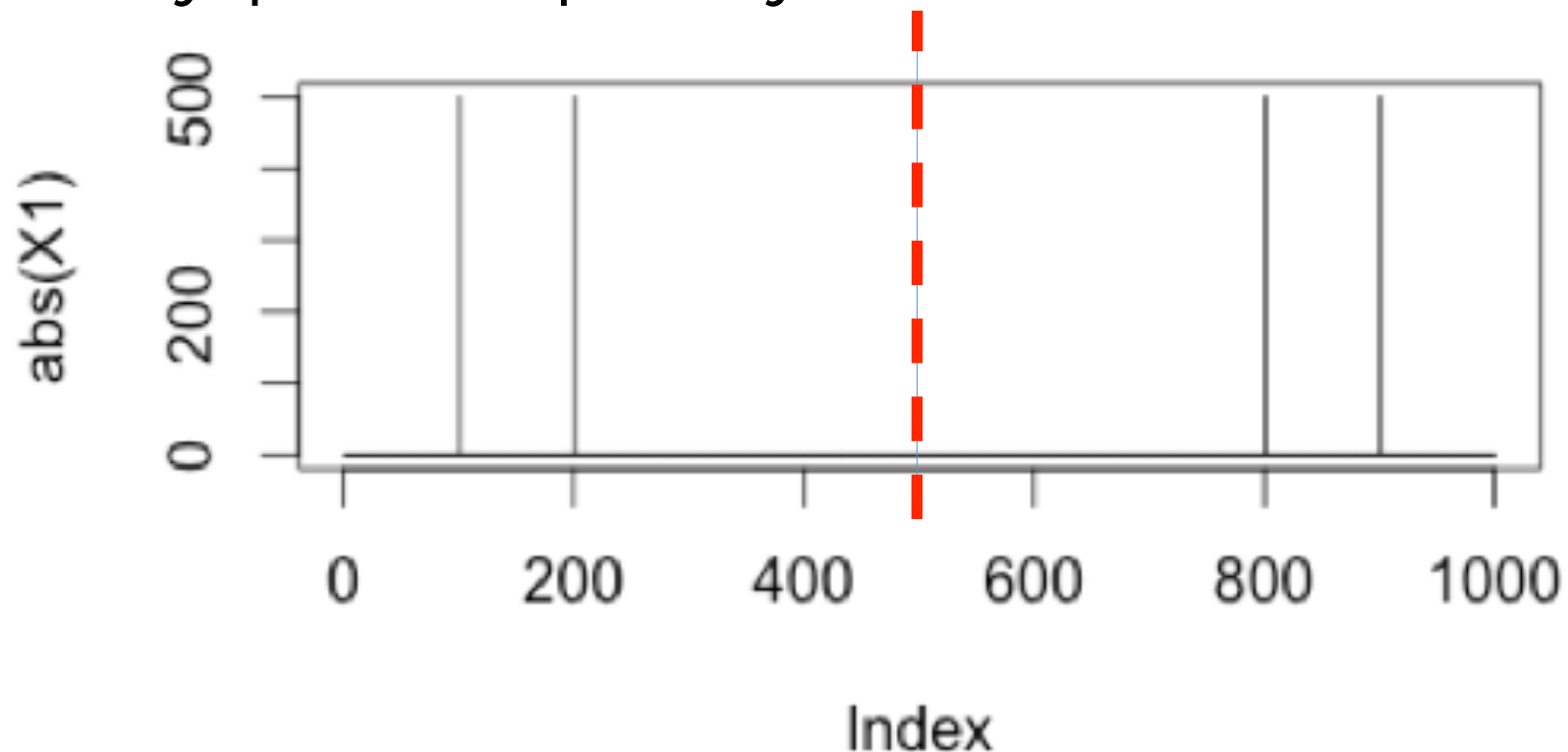
Bins above Nyquist are redundant

Magnitude spectrum is symmetric around the Nyquist frequency:



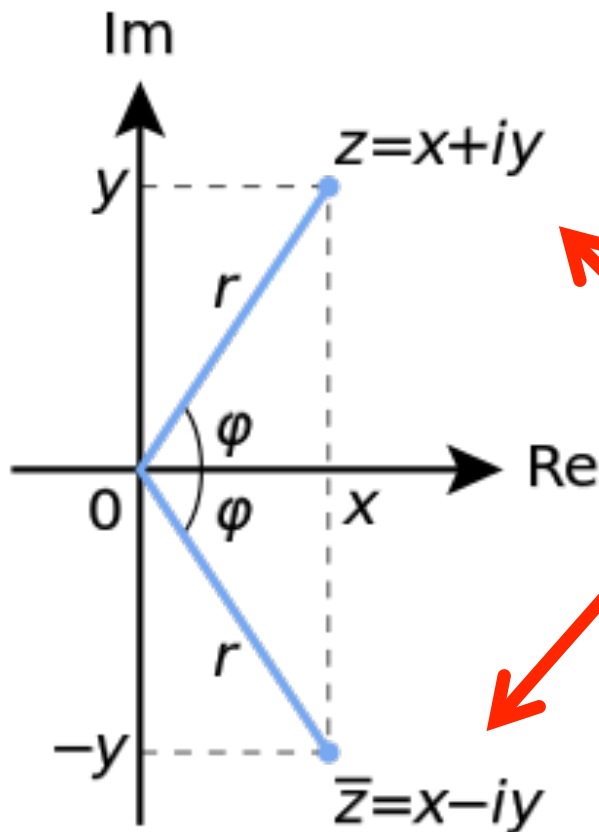
Bins above Nyquist are redundant

Magnitude spectrum is symmetric around the Nyquist frequency:



Bins above Nyquist are redundant

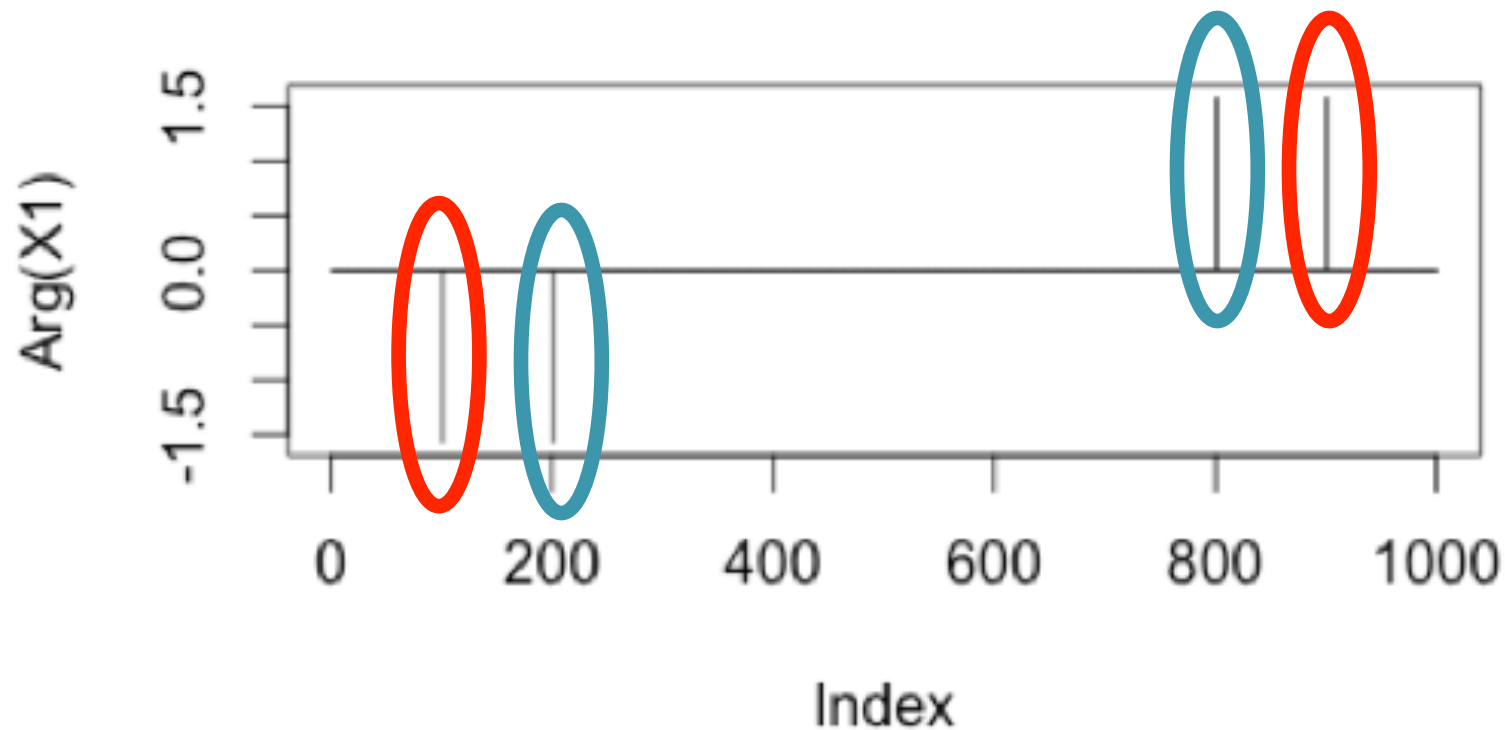
Bin k is complex conjugate of bin $N-k$:



Complex conjugates
(equal in magnitude,
opposite in phase)

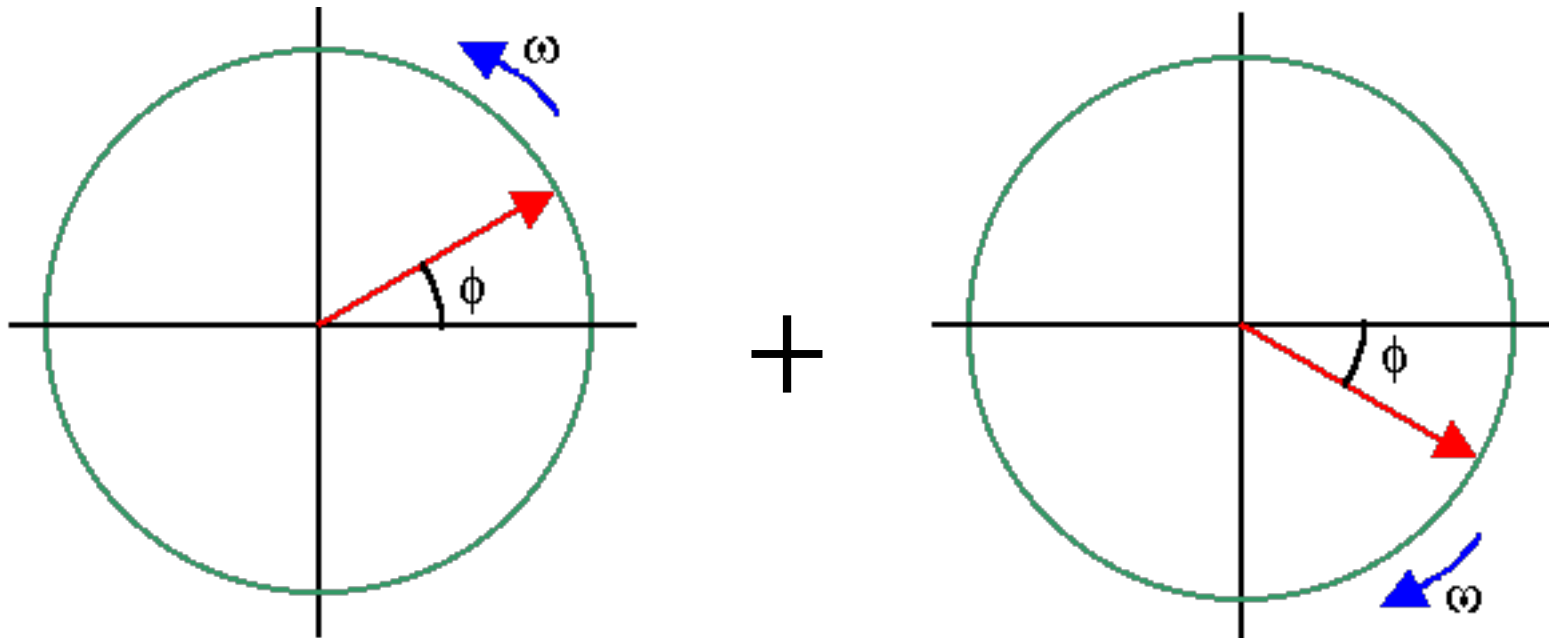
Bins above Nyquist are redundant

Bin k is complex conjugate of bin $N-k$:
phases of these bins are flipped



Why???

If your input is a real-valued sinusoid, FFT decomposes it into one phasor rotating clockwise and one rotating counterclockwise, at the same frequency.






Practical takeaway so far:

You only need to use bins 0 to $N/2$ for analysis, assuming your input signal is real-valued (and not complex-valued: always true for audio)

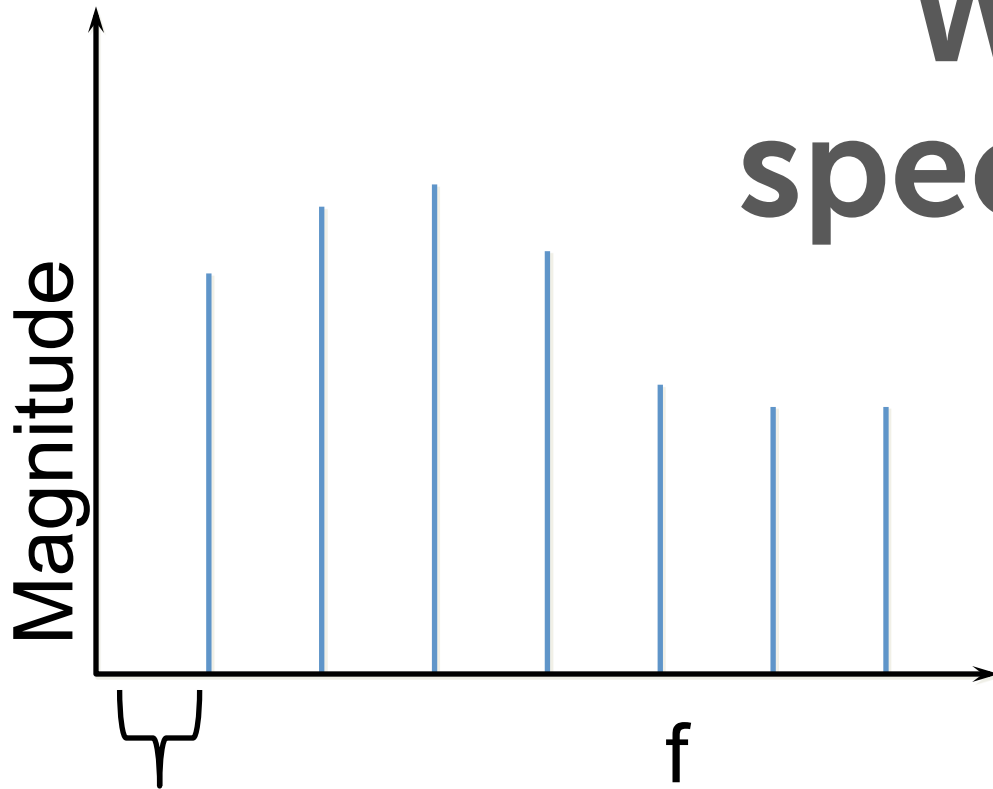
There are specific, simple relationships between magnitudes & phases of these first $N/2+1$ bins and the rest of the bins.



Converting from bin # to frequency in Hz

- N bins of FFT evenly divide frequencies from 0 Hz to $(N-1)/N * SR$
 - Why not up to sample rate itself?
 - SR indistinguishable from 0Hz!
- We're chopping frequencies from 0 up to (but not including) the sample rate into N bins, SO consecutive bins are $(1/N)*SR$ apart

Width of spectrum bins



$$\Delta f = f_{\max} / N = \textit{SampleRate} / N$$



Example

I take an FFT of 128 samples; my sample rate is 1000Hz.

$N = 128$; I have 128 "bins".

Bin 0 is? (*assuming indexing starting w/ 0*)

0 Hz

Bin 1 is?

$$(1/128) * 1000 \approx 7.8 \text{ Hz}$$

Bin 2 is?

$$(2/128) * 1000 \approx 15.6 \text{ Hz}$$

Example

I take an FFT of 128 samples; my sample rate is 1000Hz.

N = 128; I have 128 "bins".

14th bin is?

$$(14/128) * 1000 \approx 109 \text{ Hz}$$

Bins nearest to 300 Hz are?


$$(b/128) * 1000 = 300 \rightarrow b = 38.4$$

bins 38 and 39 are closest

Last bin I care about is?

$$\text{Nyquist: } (b/128) * 1000 = 500 \rightarrow b = 64$$

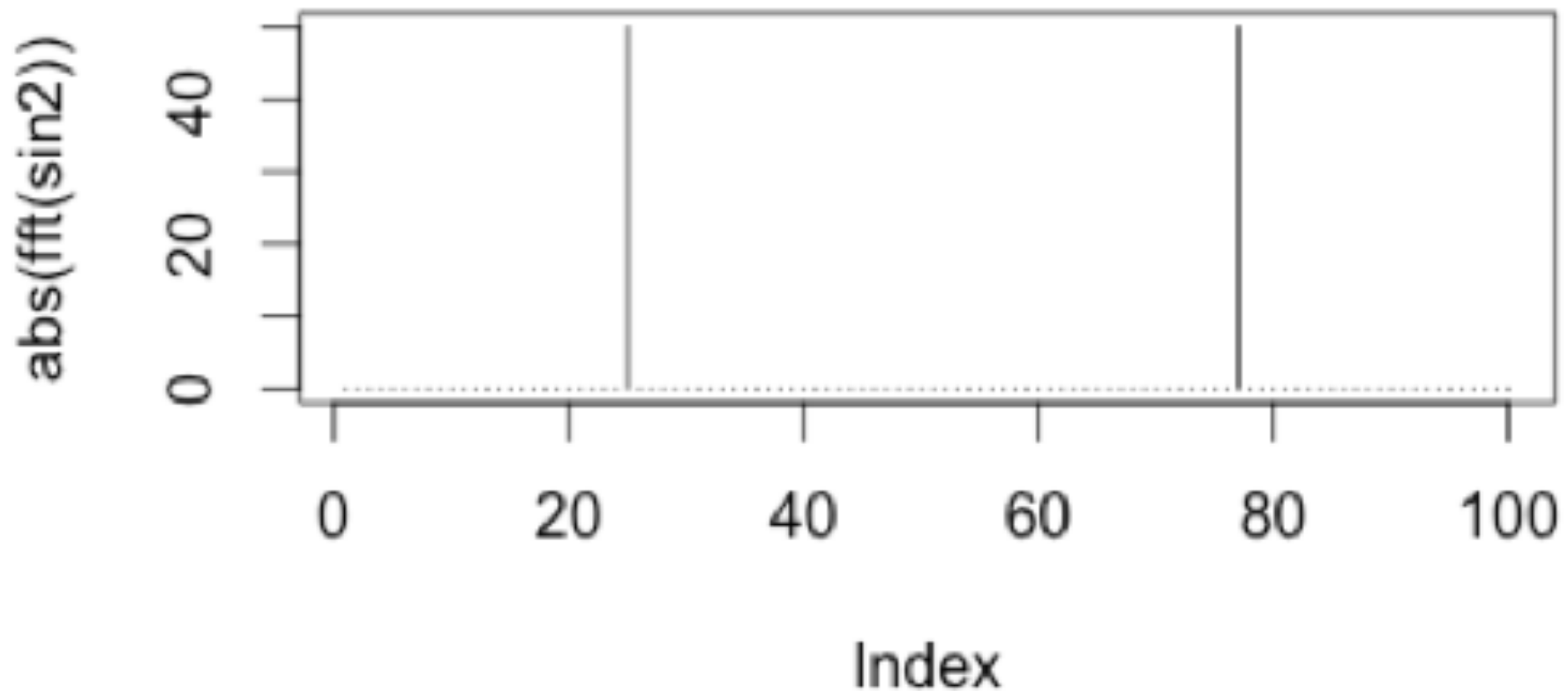
(equivalently, equal to N/2)



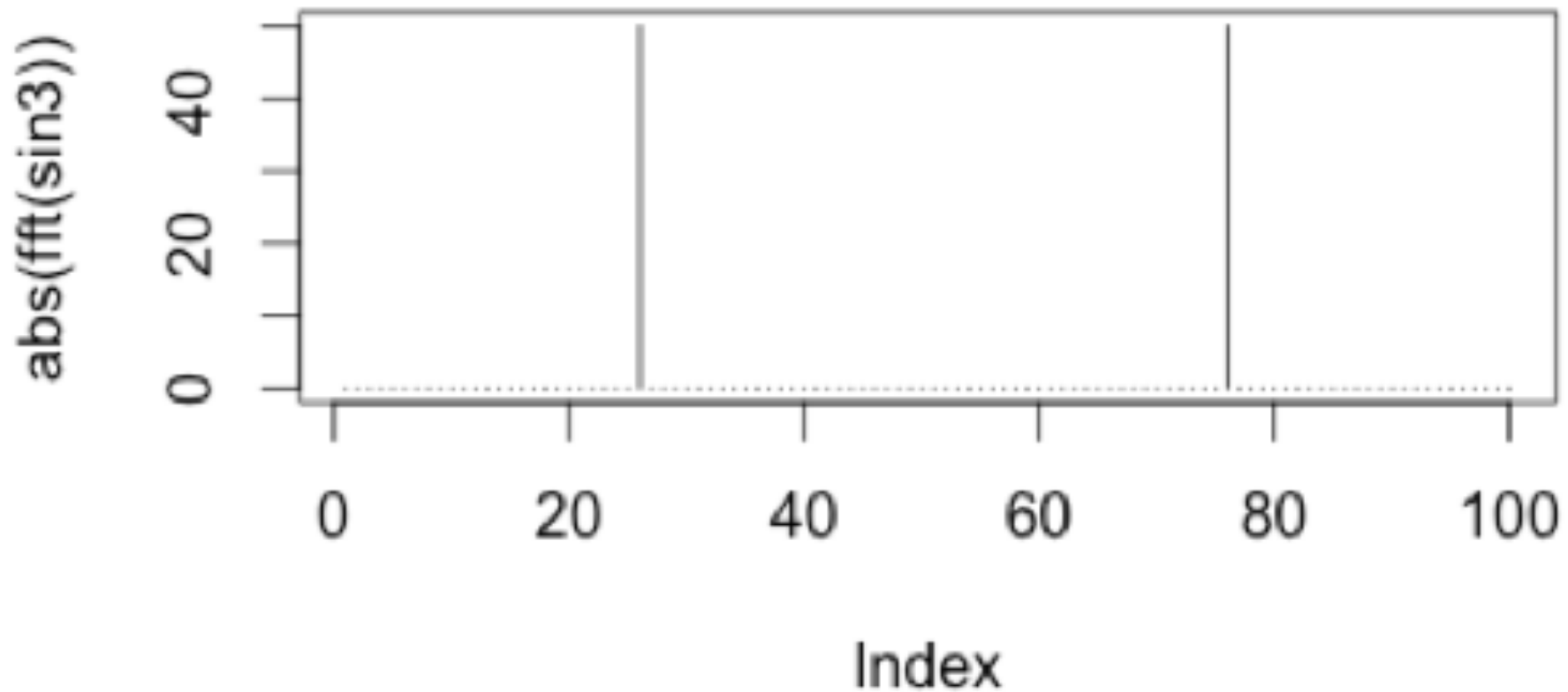
What happens if my signal contains a frequency that's not exactly equal to the center frequency of a bin?

This frequency will “leak” into nearby bins.

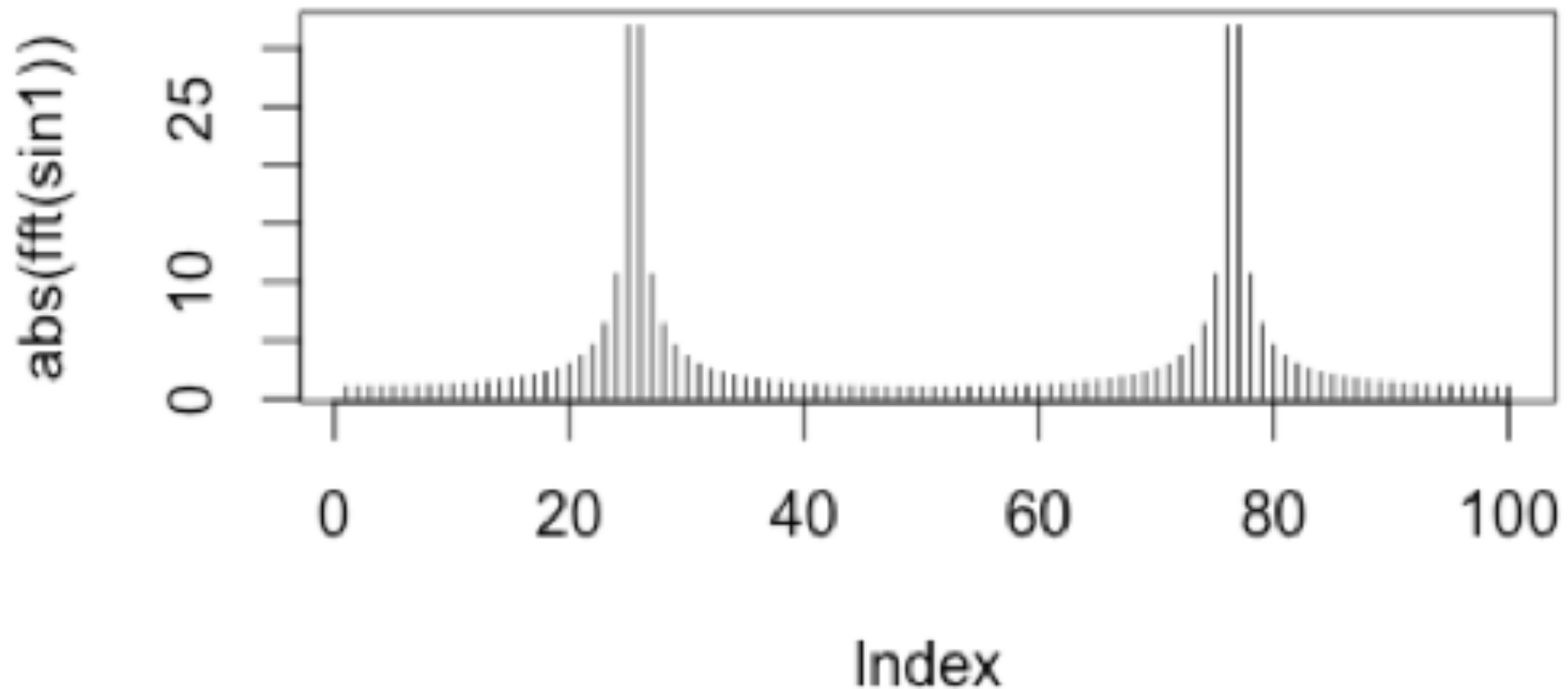
SR = 100Hz, sine at 24 Hz



SR = 100Hz, sine at 25 Hz



SR = 100Hz, sine at 24.5 Hz





How many bins to use? (What should N be?)

More bins?

- Better frequency resolution

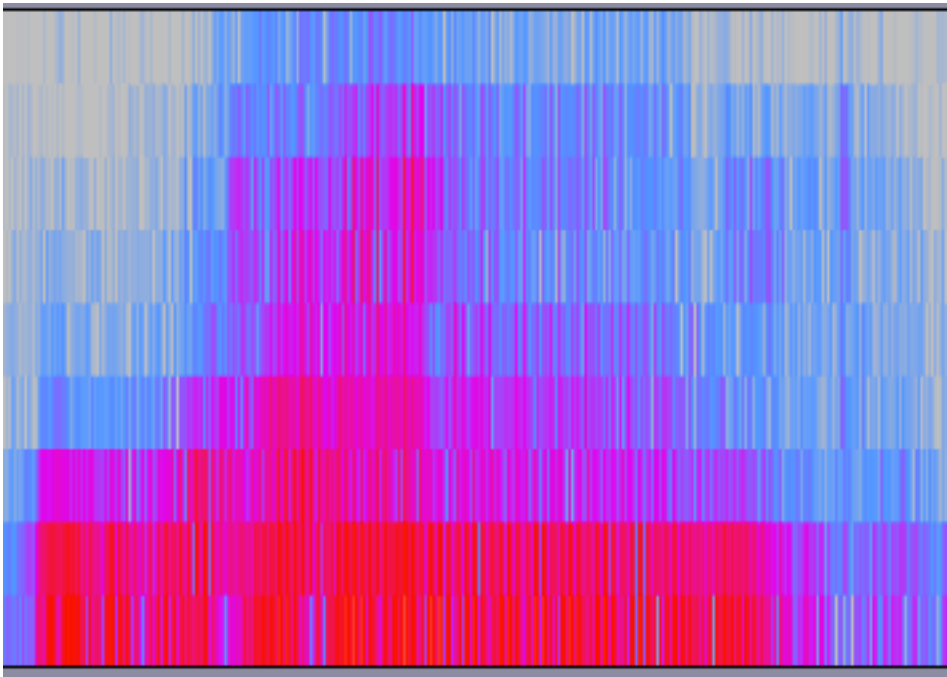
- Worse time resolution (FFT can't detect changes within the analysis frame)

Fewer bins?

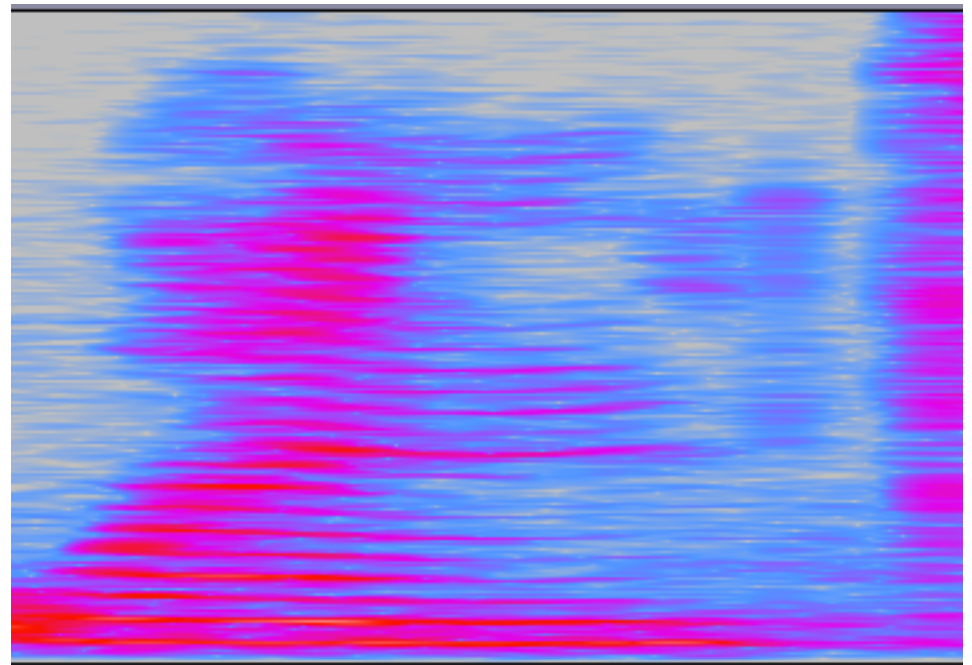
- Worse frequency resolution

- Better time resolution

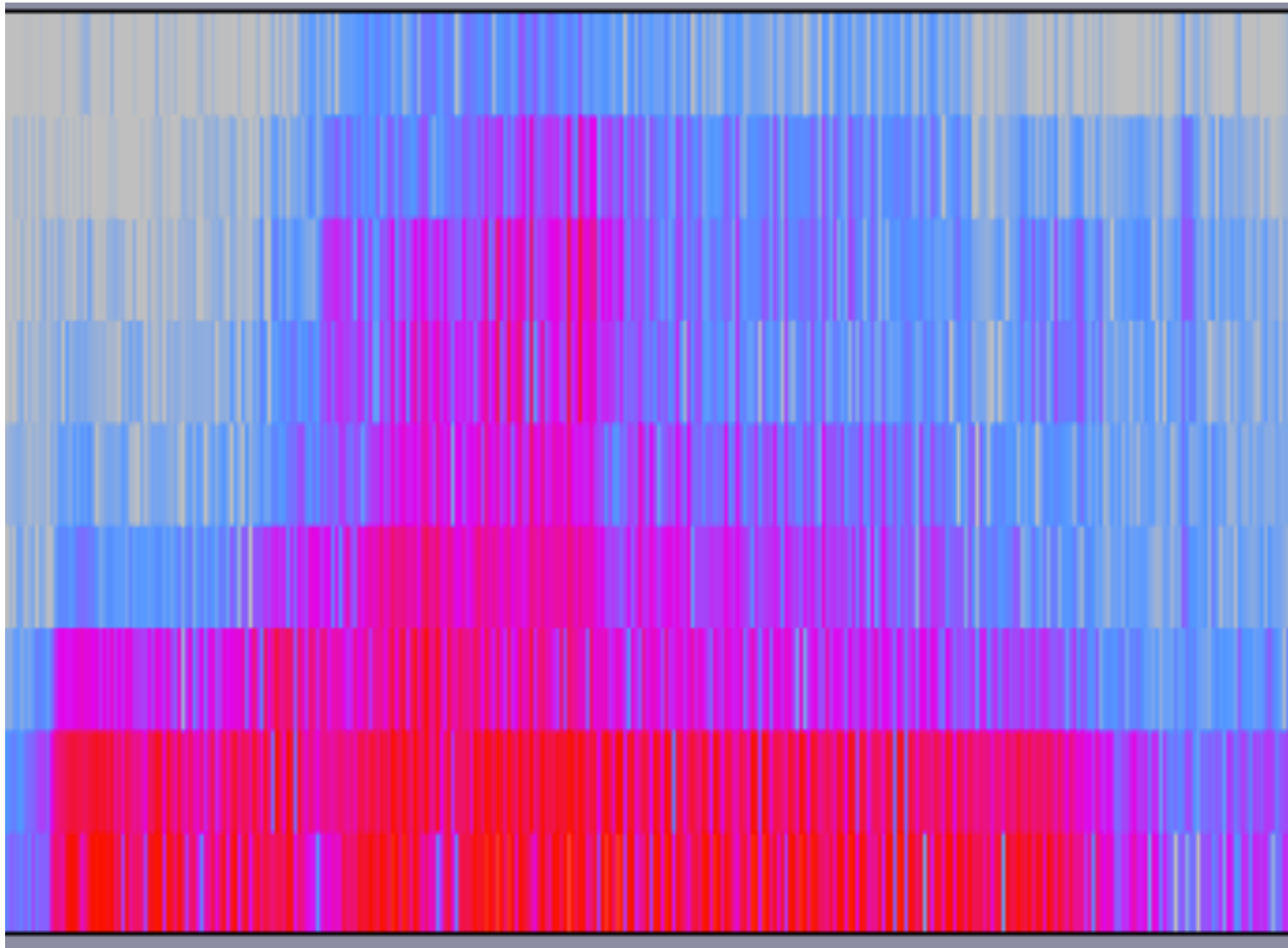
Time/Frequency tradeoff

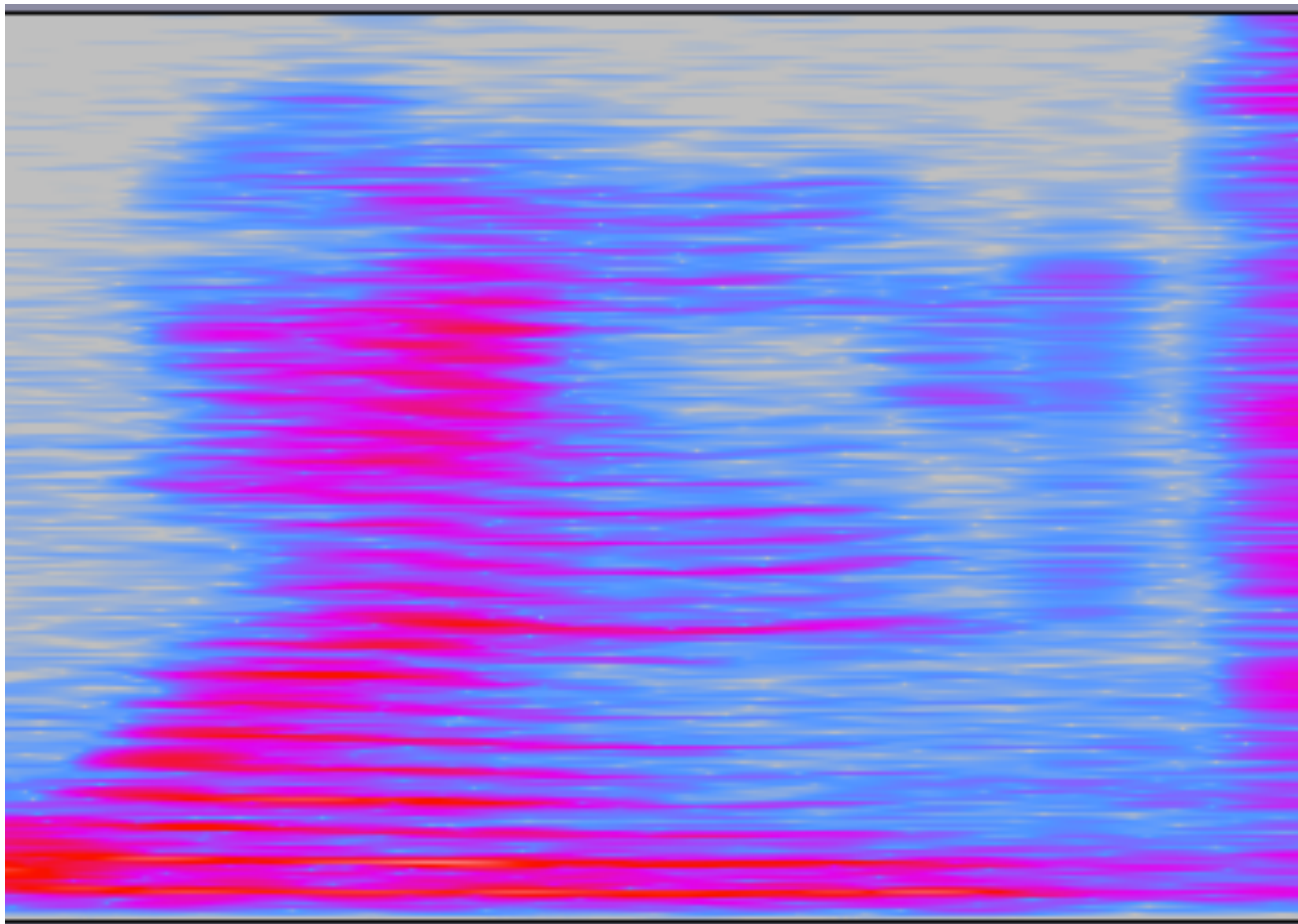


N=64

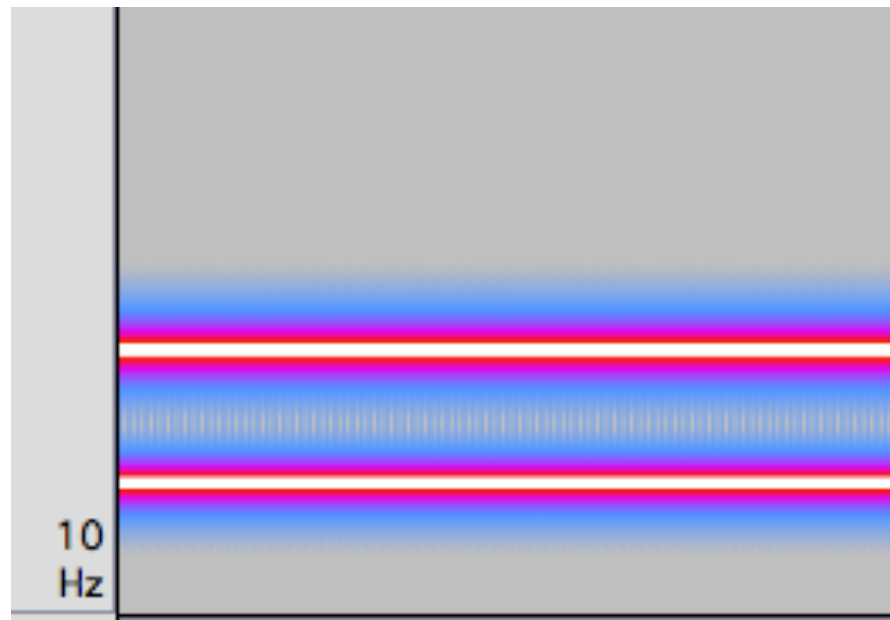
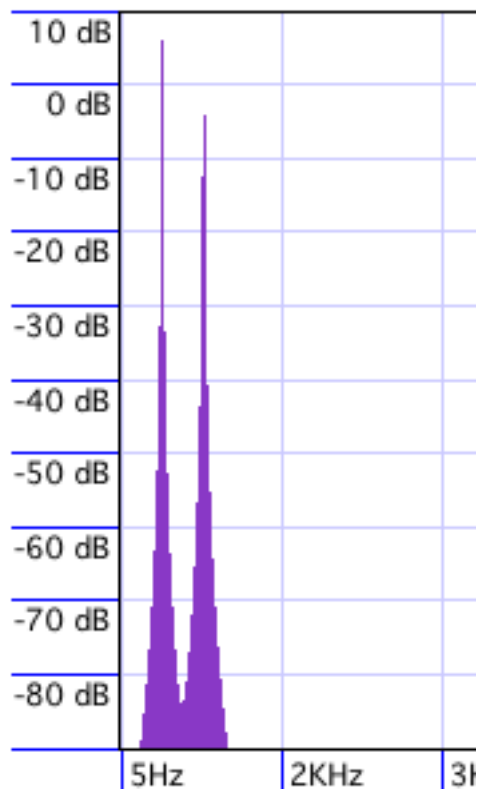


N=4096





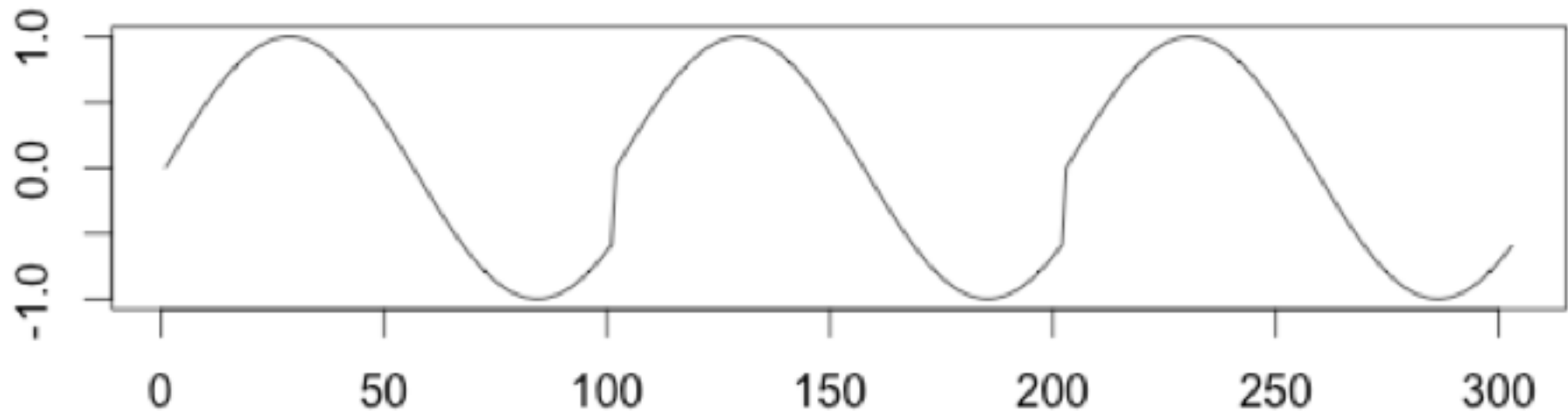
What's all that extra stuff in the spectrum?



33 Not just clean peaks at frequencies and 0 elsewhere...

Reasons for this “stuff”

FFT treats your analysis frame as one period of an infinite, periodic signal.



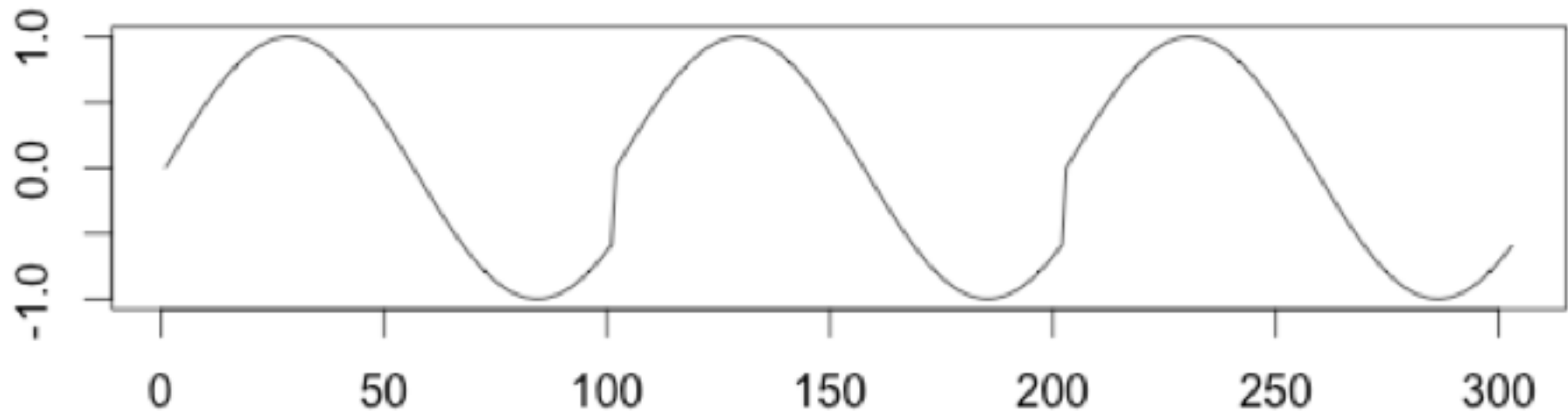
Signal doesn't have an integer # of periods in frame?

→ Contains frequency components

other than 0 , $(1/N)*SR$, $(2/N)*SR$, ... $SR/2$.

Reasons for this “stuff”

FFT treats your analysis frame as one period of an infinite, periodic signal.



“periodic” signal may have discontinuities

→ only representable with high frequency content

Practice: Pitch tracking




Q: How many bins should we use?

Q: Algorithm to determine pitch?



Example R code

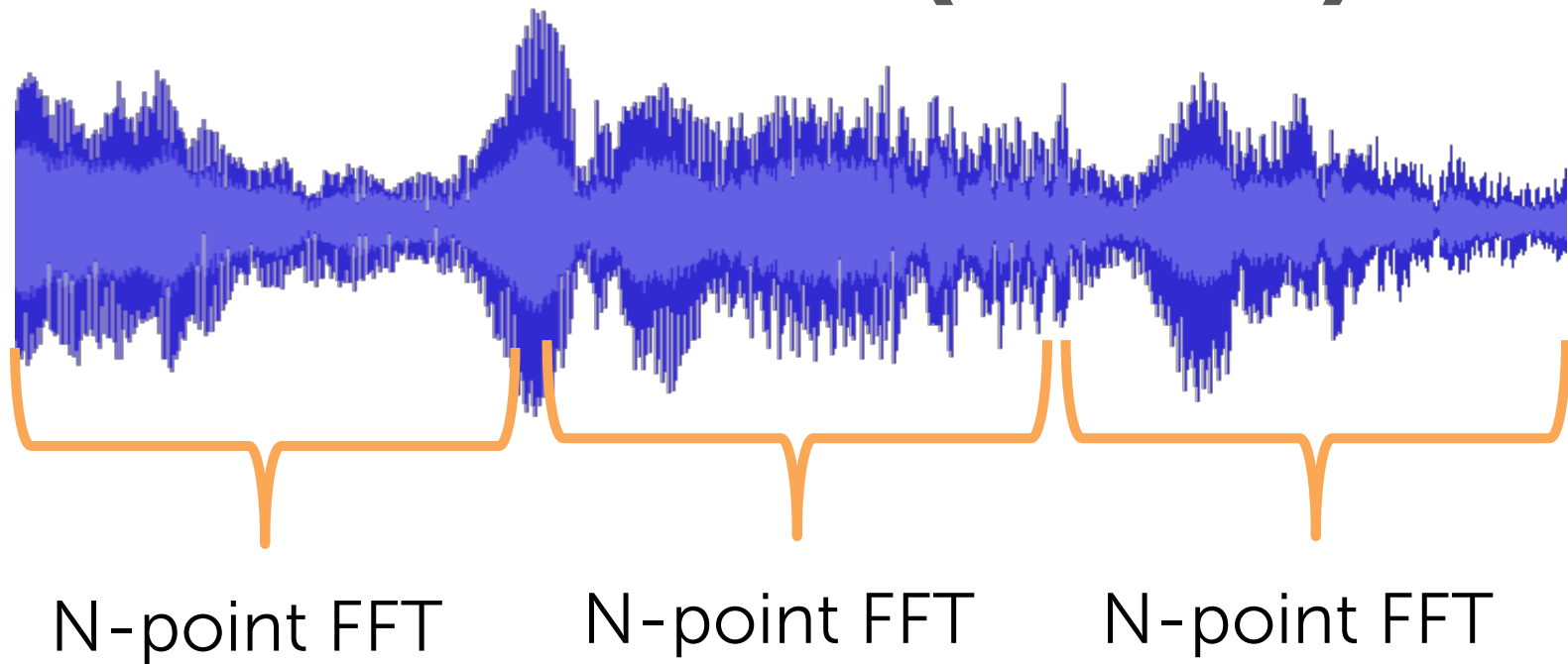
```
saw <- readWave("sawtooth.wav")  
  
X <- fft(saw@left[1:2048]) #saw@left gives  
                           # us left channel samples  
  
plot(abs(X)[1:1025], type="h")  
  
maxbin <- which.max(abs(X)[1:1025])  
  
maxfreq <- (maxbin-1)/2048*44100  
           #assuming 44100 SR
```



How to deal with music that changes over time?

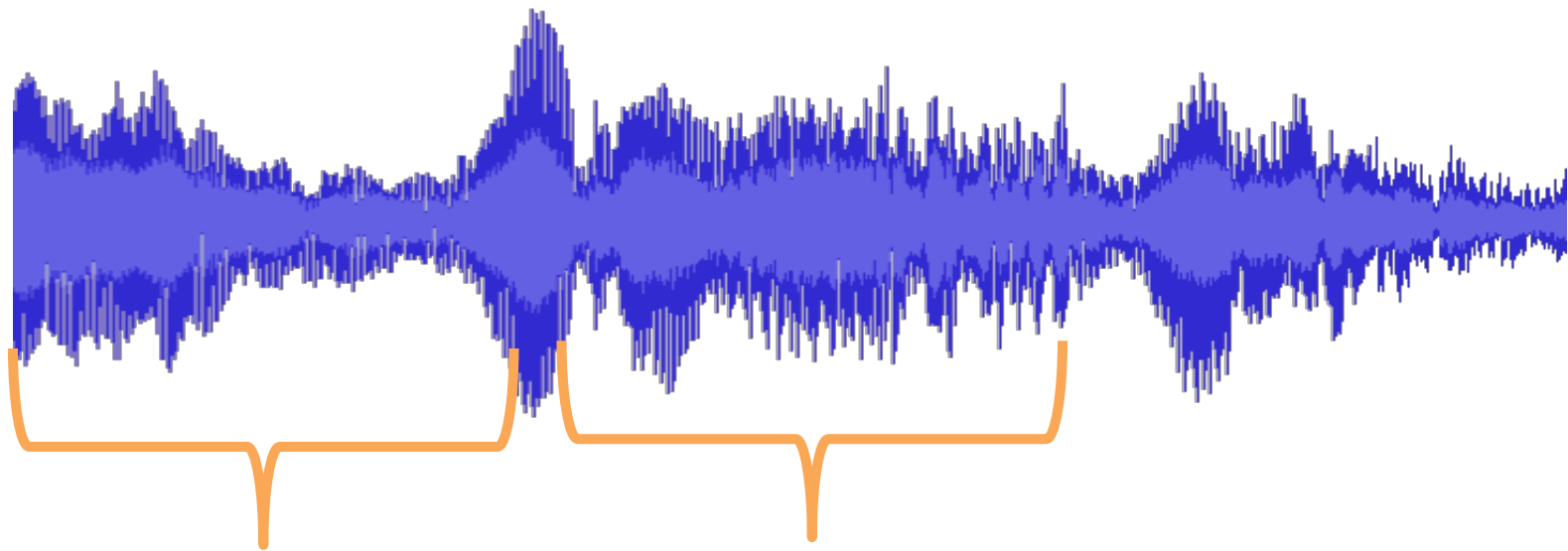
Compute FFT at many points in time.

"Short-time Fourier Transform" (STFT)



STFT hop size


of samples between beginning of one frame and the next



N-point FFT

N-point FFT

Equivalently talk about “overlap” between adjacent frames.
Adjust based on application needs.



Example applications of STFT?

Pitch tracking over time (melody extraction)

Onset detection (for rhythm/tempo analysis?)

Audio fingerprinting

More discussion on these in a few weeks



Practical FFT Questions

- $N = ?$ (Frame length)
 - Balances time & frequency resolutions
 - FFT or STFT?
 - Is frequency content changing over time?
 - If STFT, choose hop size based on granularity of analysis needed
 - Do I care about magnitude, phase, or both?
 - Magnitude alone useful for basic timbre analysis, instrument identification, many other things; phase required for reconstruction of waveform
- ***Plus a few other things: revisiting this at end of lecture****



Converting from FFT back into sound

Option 1: Take magnitude and phase of each bin (including second half of bins), compute a sinusoid at appropriate magnitude, frequency, and phase...

Option 2 (MUCH BETTER): Use inverse FFT (i.e., the IFFT)

The Inverse Discrete Fourier Transform (IDFT)

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N}$$

Compare to DFT:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$$

IDFT is just like DFT, but 1) has $1/N$ factor and positive exponent; 2) converts from complex into real (assuming original signal was real-valued)



The IFFT in practice

Compute IDFT using the IFFT

N FFT bins \rightarrow N IFFT samples

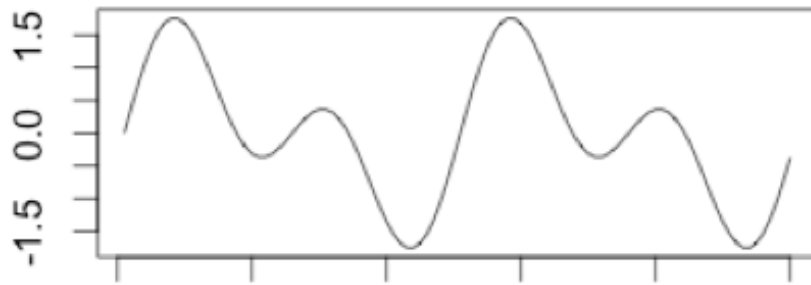
In R, with `signal` library:

```
x <- abs(ifft(X))
```

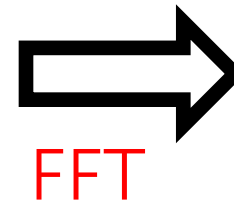
(abs enforces reasonable assumption of real valued elements of x)

A possible application of IFFT?

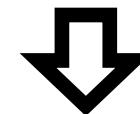
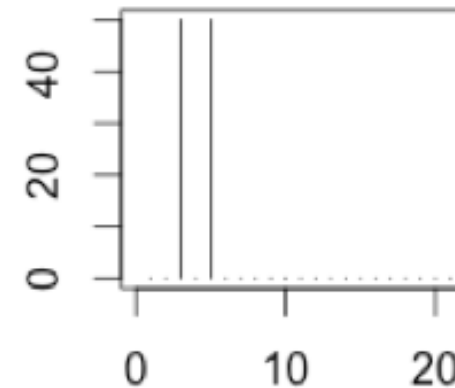
Modify a sound by manipulating its spectrum:



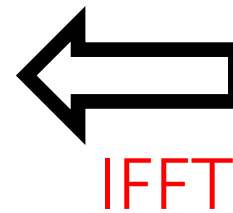
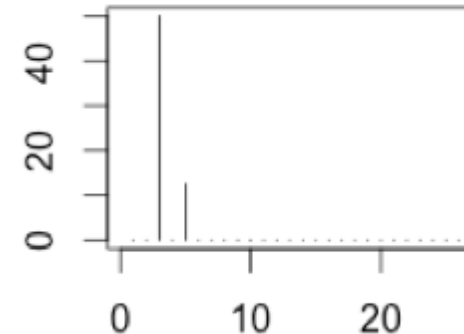
Original signal



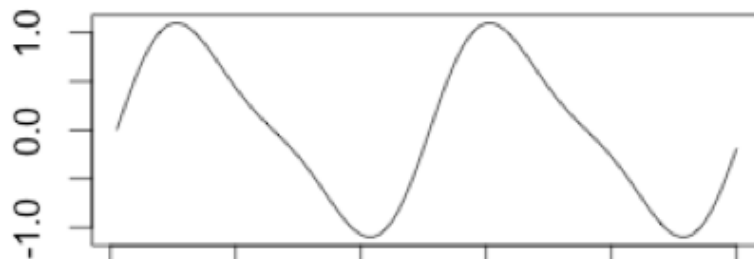
FFT



Multiply 4th
bin by 0.25



IFFT

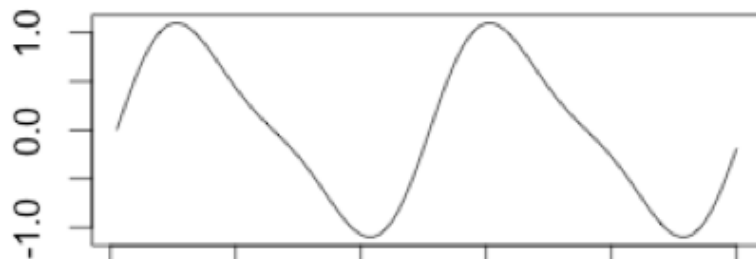


Modified signal

A possible application of IFFT?

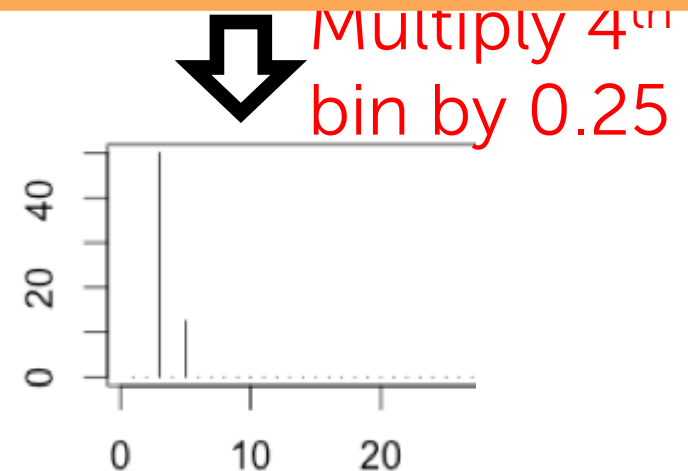
Modify a sound by manipulating its spectrum:

There are better ways of doing this...



Modified signal

IFFT



Why so many versions of Fourier analysis?

	Continuous Time	Discrete Time
Aperiodic / unbounded time, continuous frequency	Fourier Transform	Discrete-time Fourier Transform (DTFT)
Periodic or bounded time, discrete frequency	Fourier Series	Discrete Fourier Transform (DFT) (FFT used here)

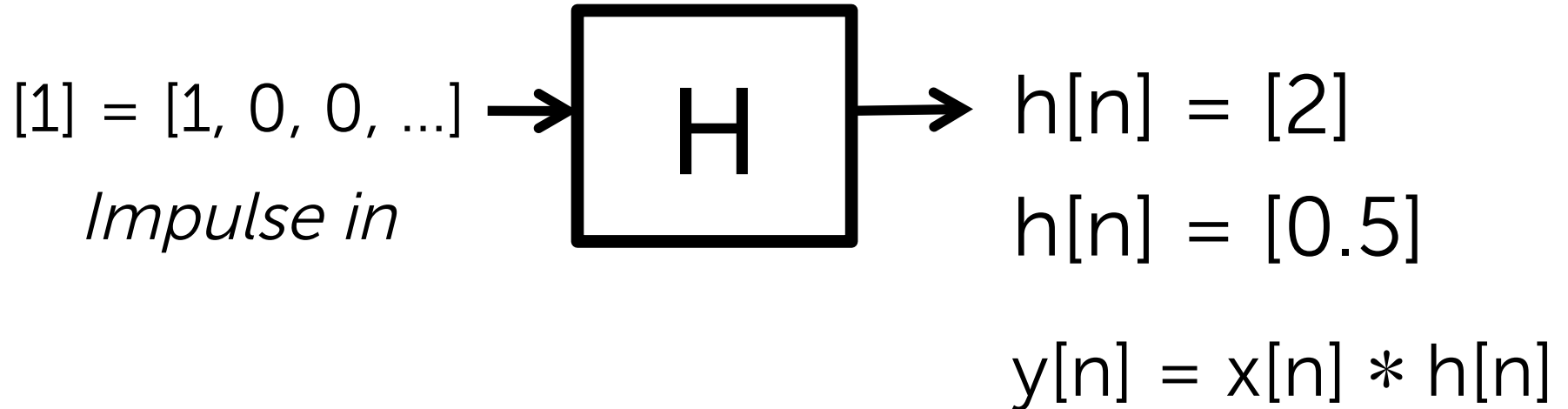
- Each of these also has an inverse.
- You'll mainly care about the FFT (the fast algorithm for computing the DFT).



How to build useful systems?

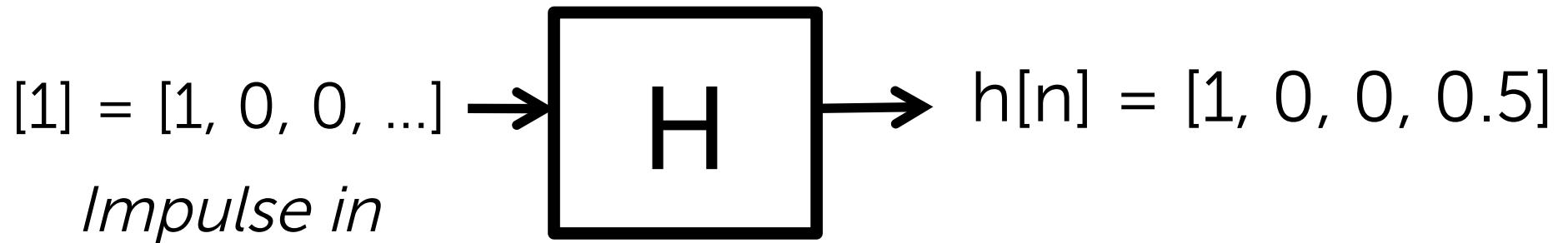
Method 1) Design a useful impulse response.

A very simple system



Volume control!

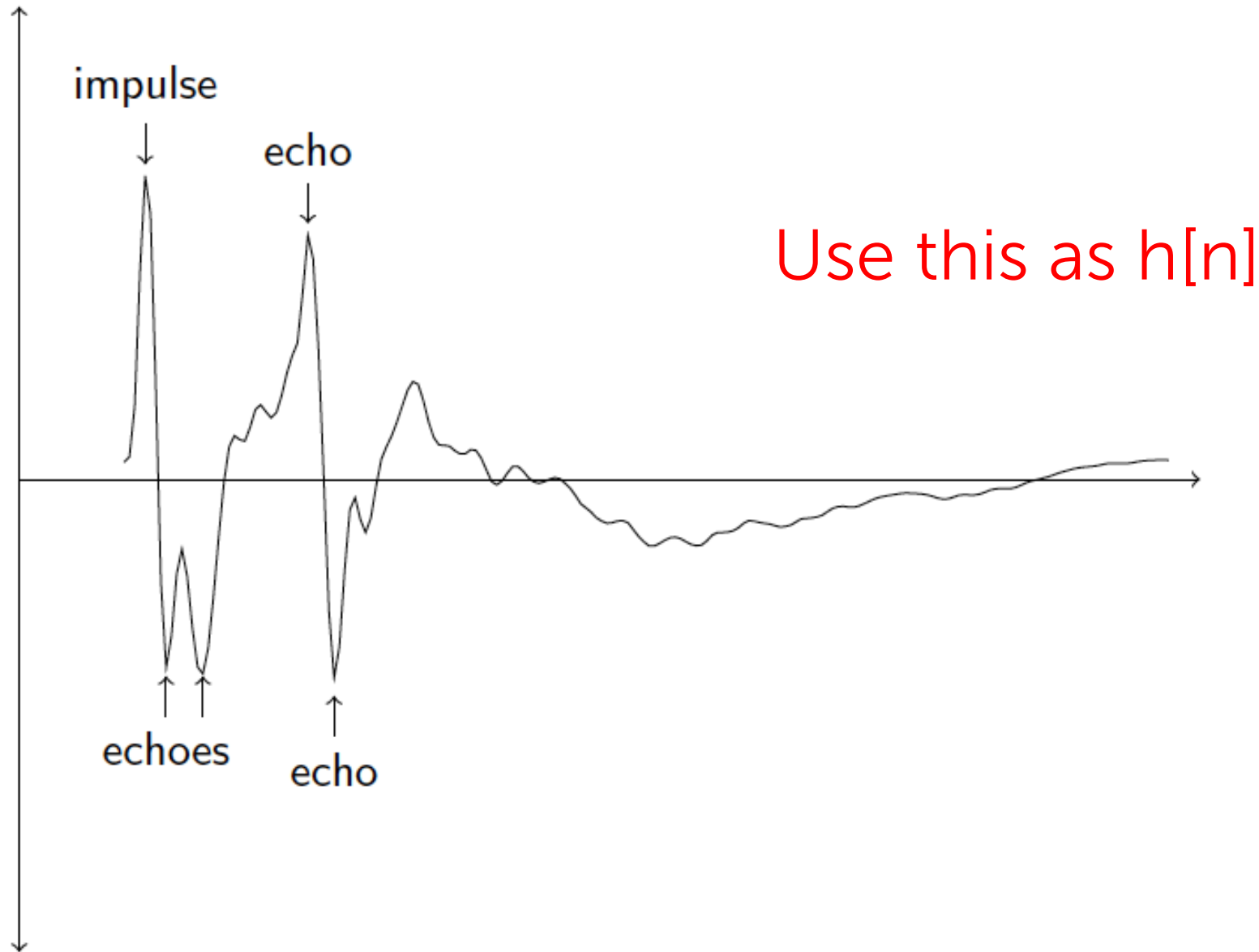
Another very simple system



$$y[n] = x[n] * h[n]$$

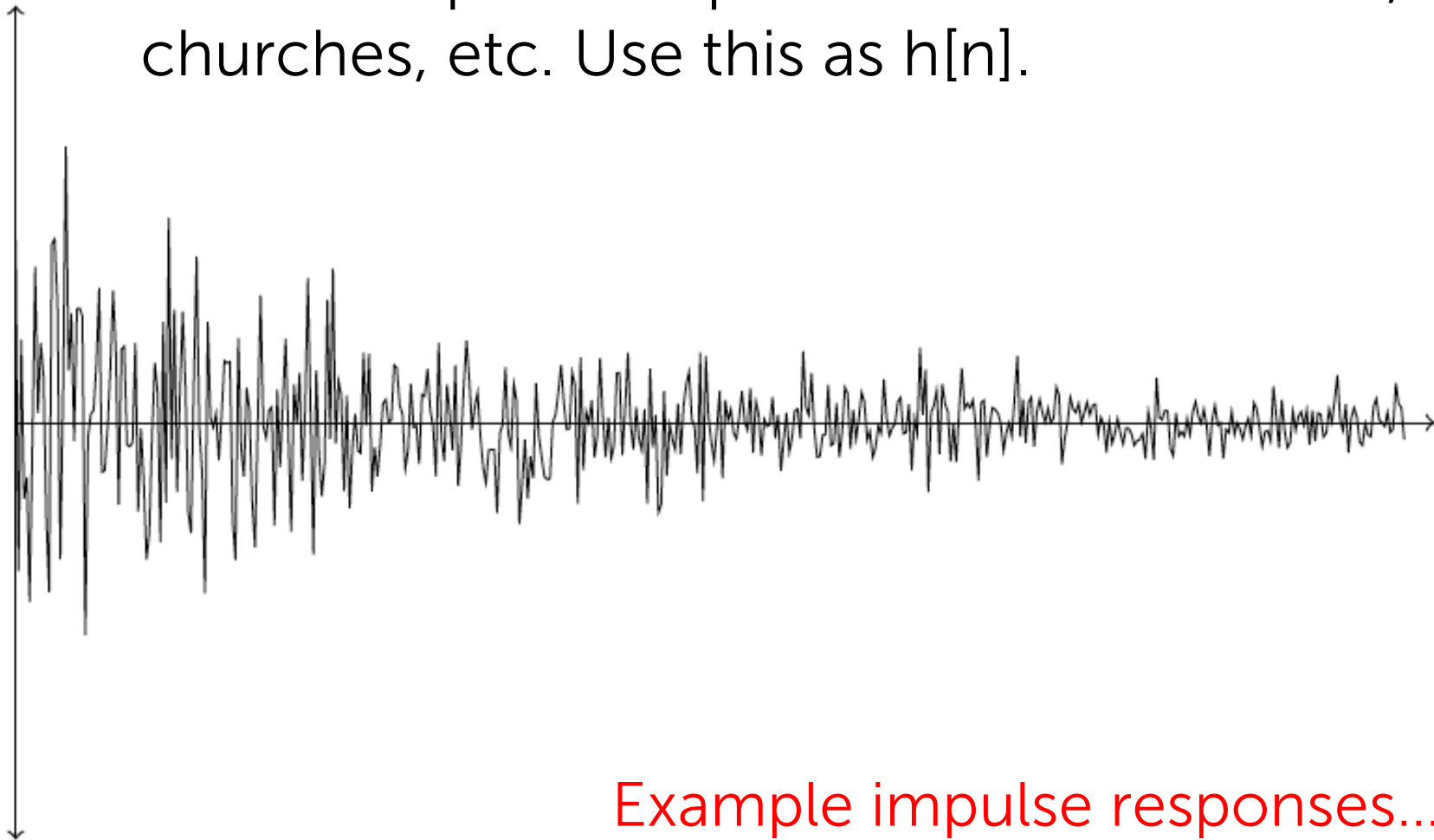
[very simple] echo

More realistic echo



Convolution reverb

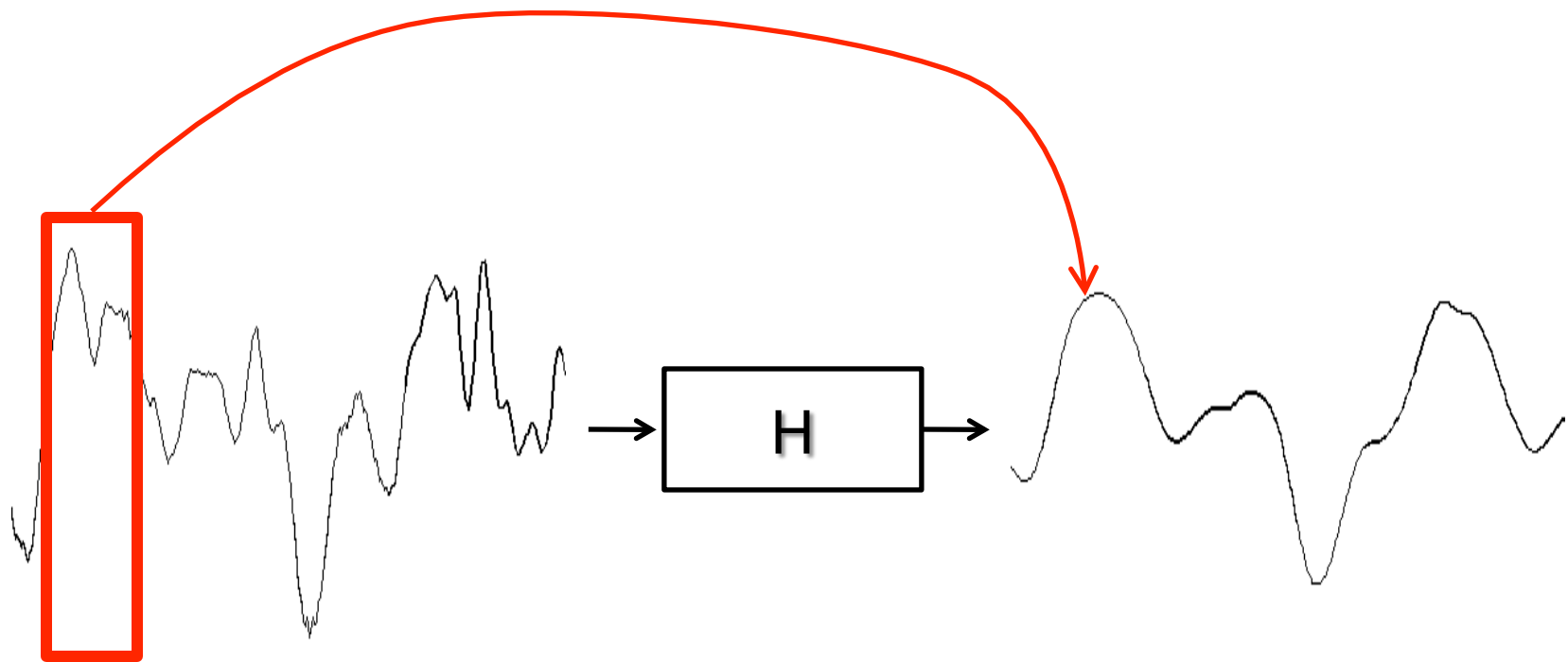
Record impulse response for concert halls, churches, etc. Use this as $h[n]$.



Example impulse responses...

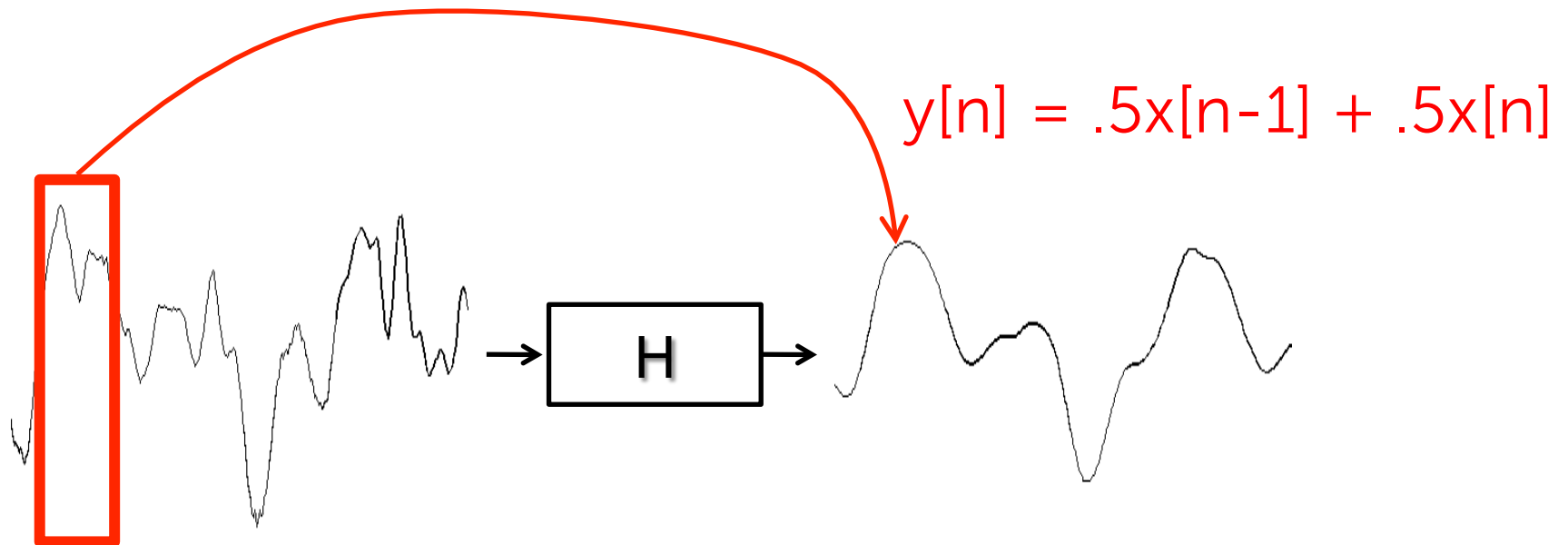
A simple smoothing system

Take average of nearby points:



A simple smoothing system

$$[1] = [1, 0, 0, \dots] \rightarrow \boxed{H} \rightarrow h[n] = [0.5, 0.5]$$

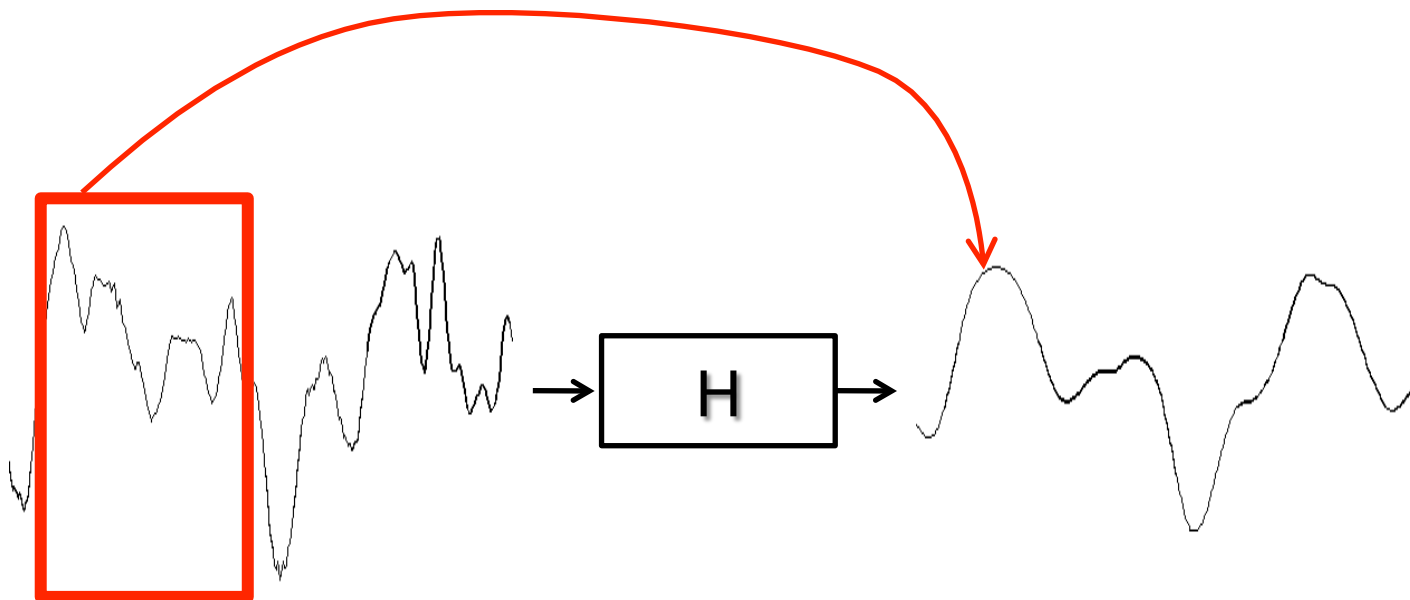


How to improve this?

Can use $h=[0.25, 0.25, 0.25, 0.25]$,
 $h=[0.1, 0.1, \dots 0.1]$ to make signal even smoother

But there's a better way...

"smoother" = "less high-frequency content"





How to build useful systems?

Method 1) Design a useful impulse response

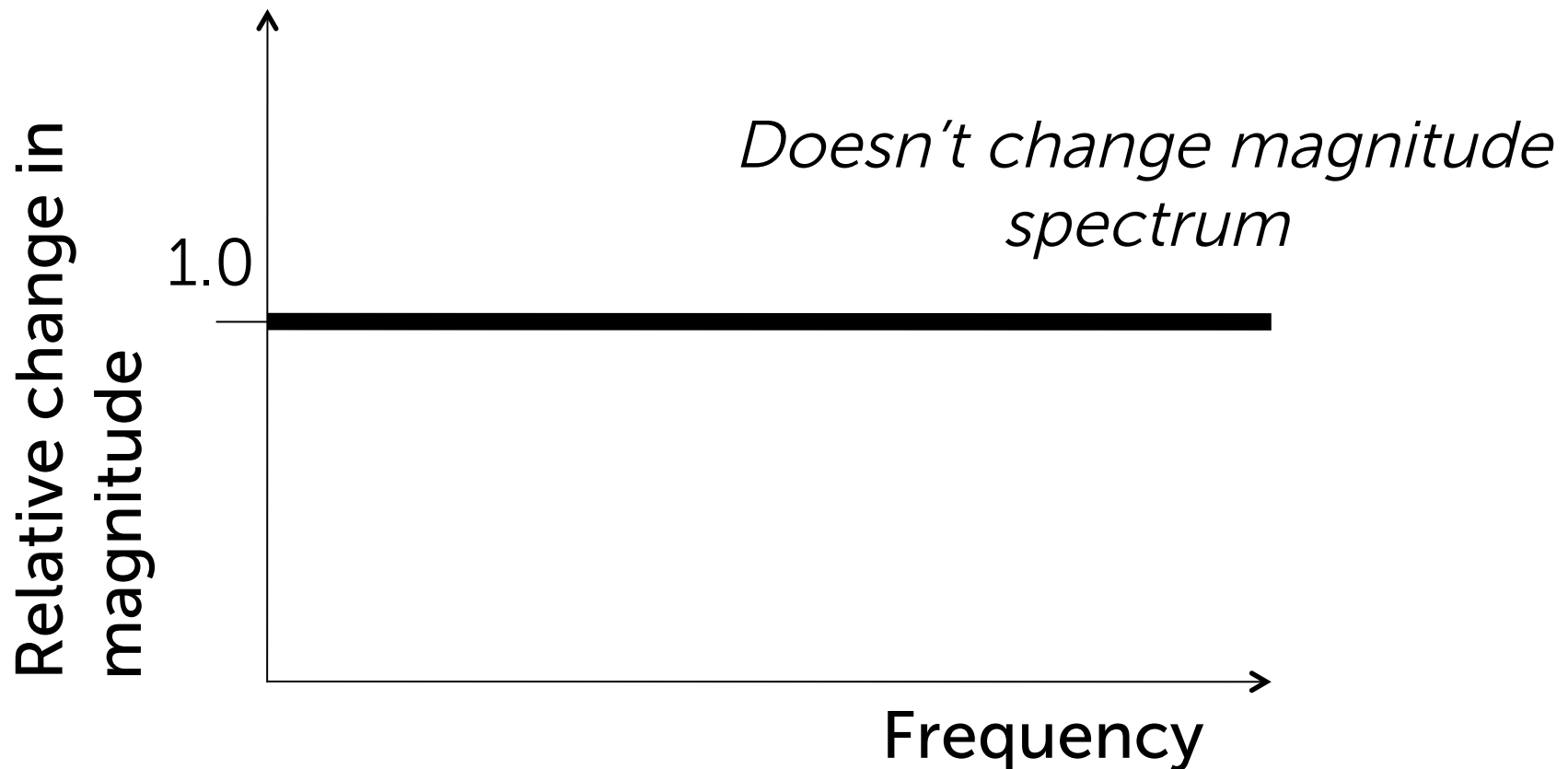
We have to know how we want the time-domain sound signal to be changed by the system.

Method 2) Design a useful **frequency response**

Instead, we can decide how we want the **spectrum** of the sound to be changed by the system.

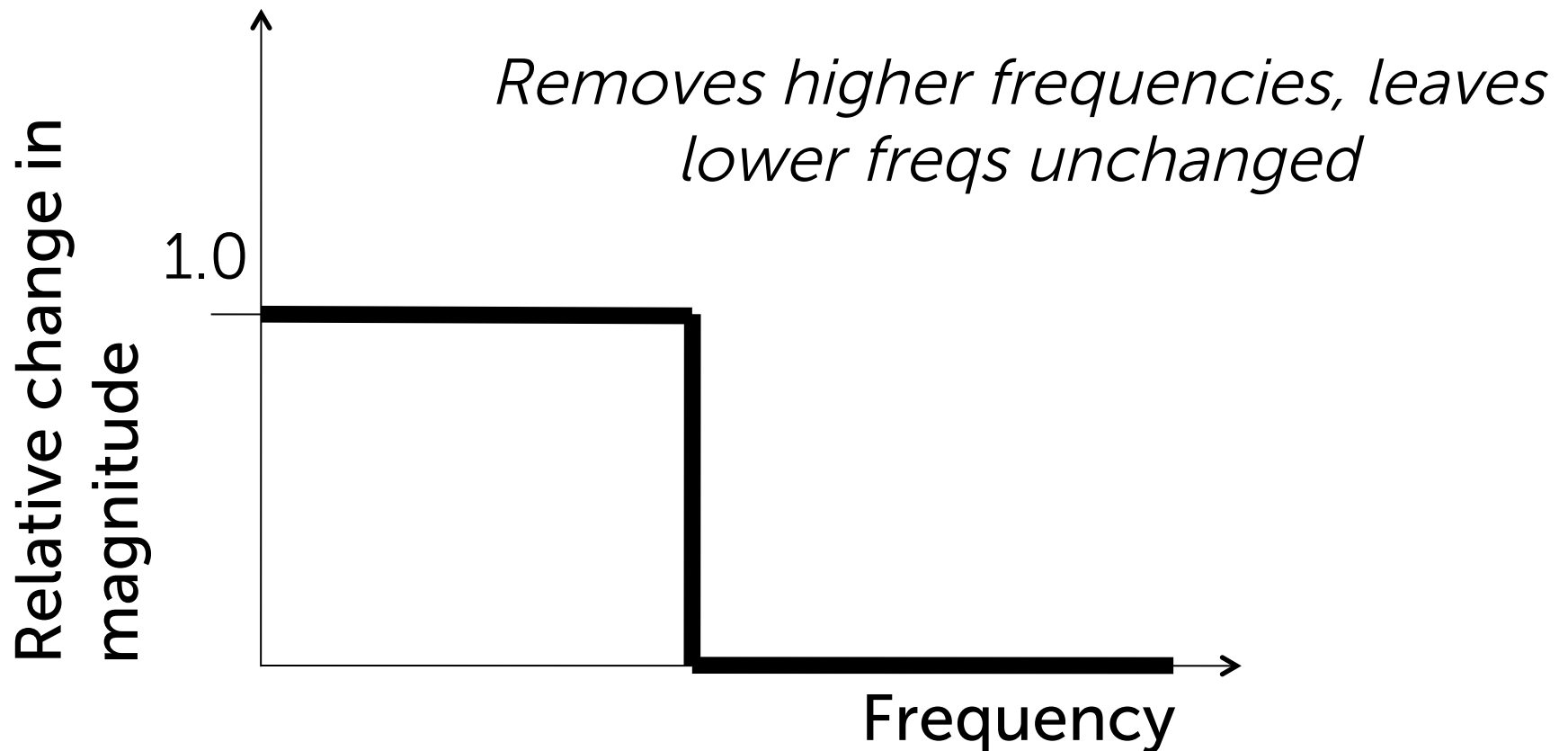
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



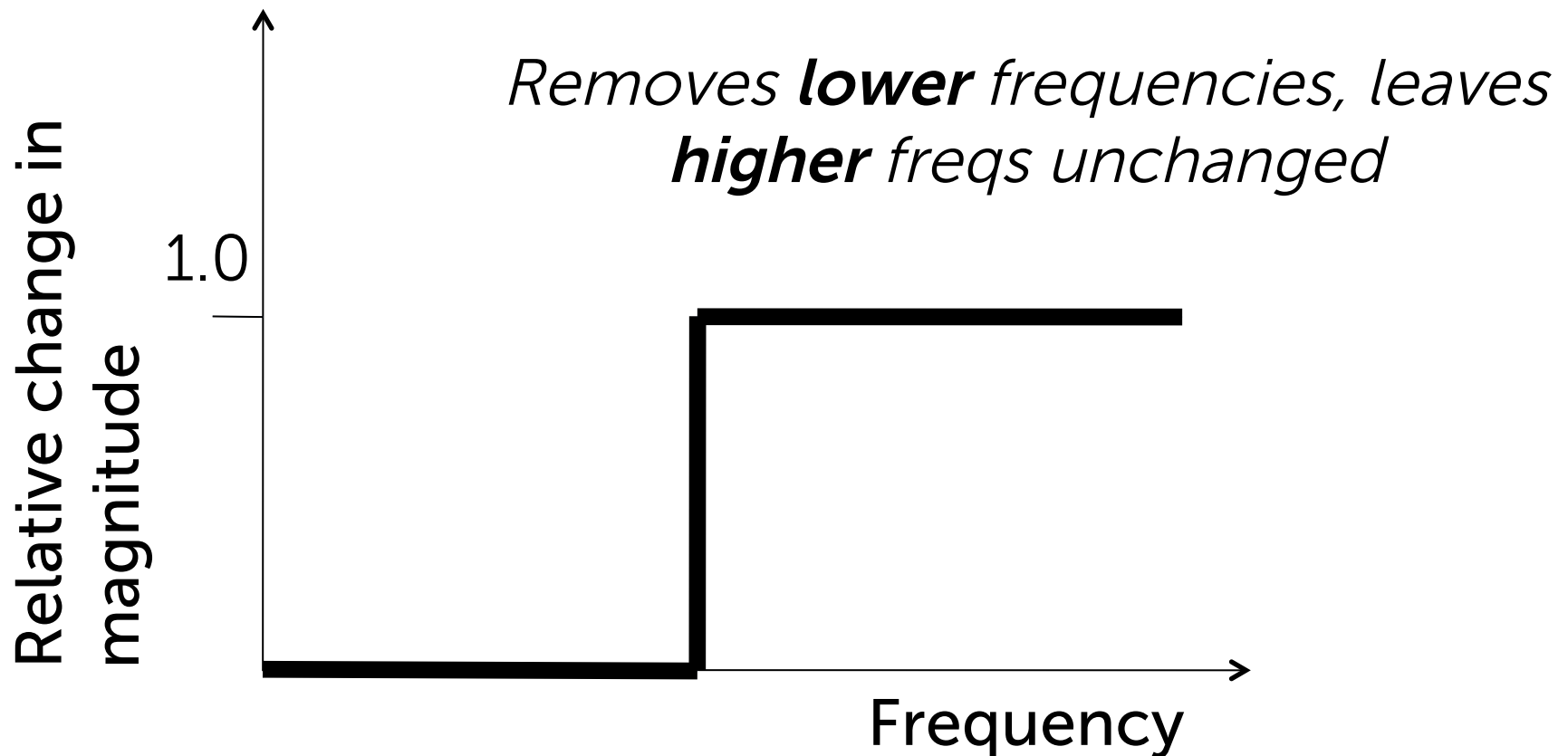
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



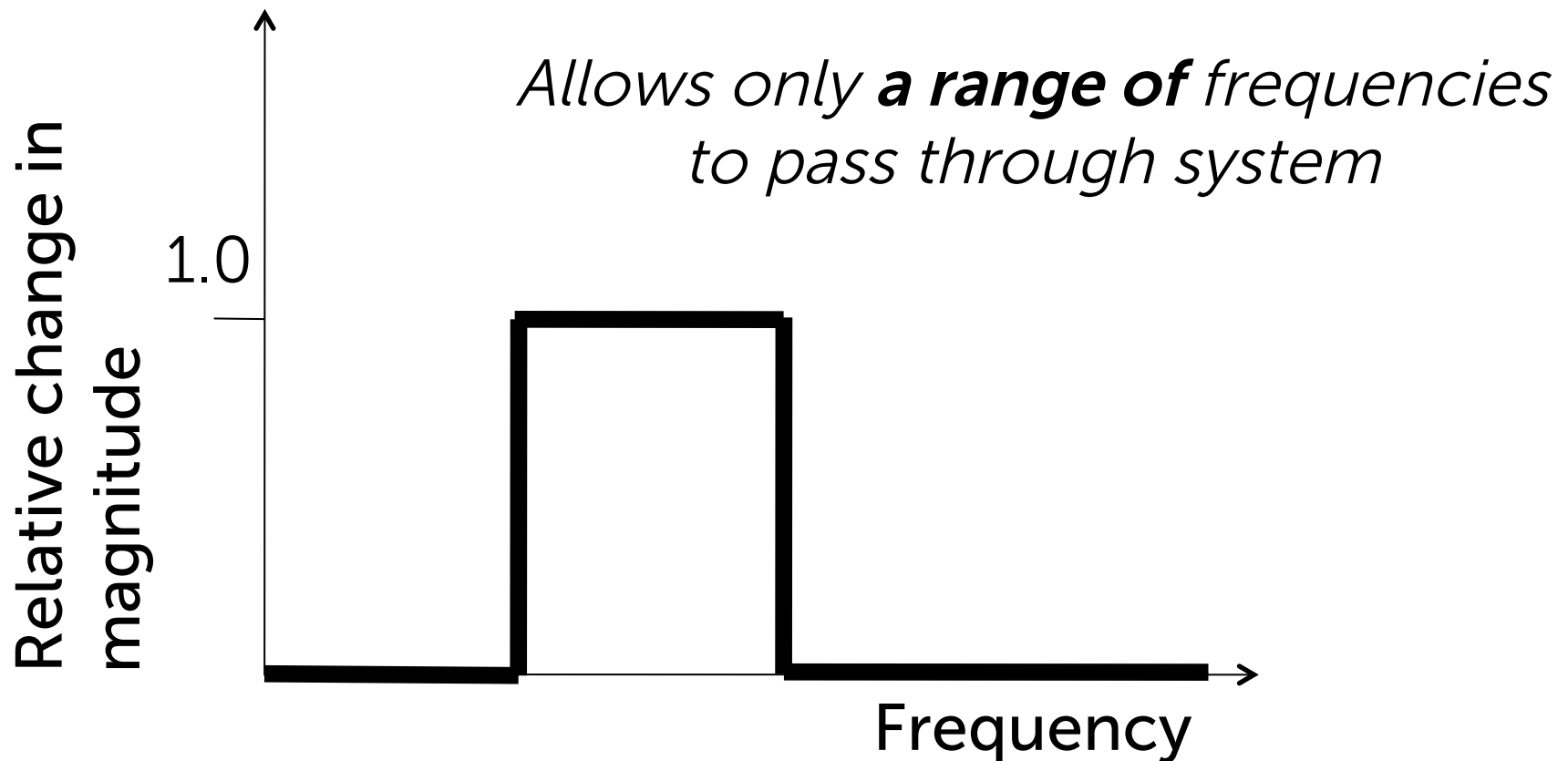
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



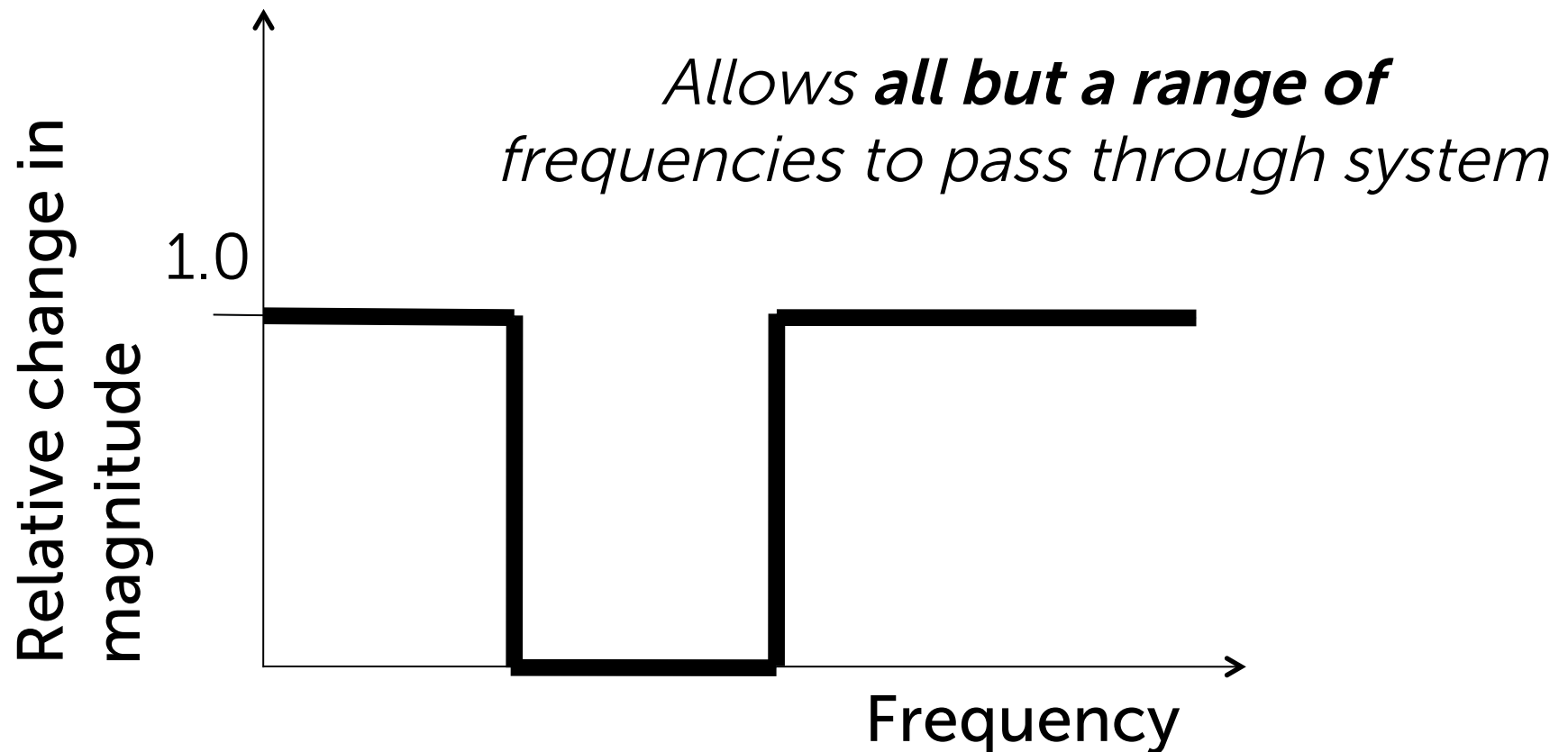
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



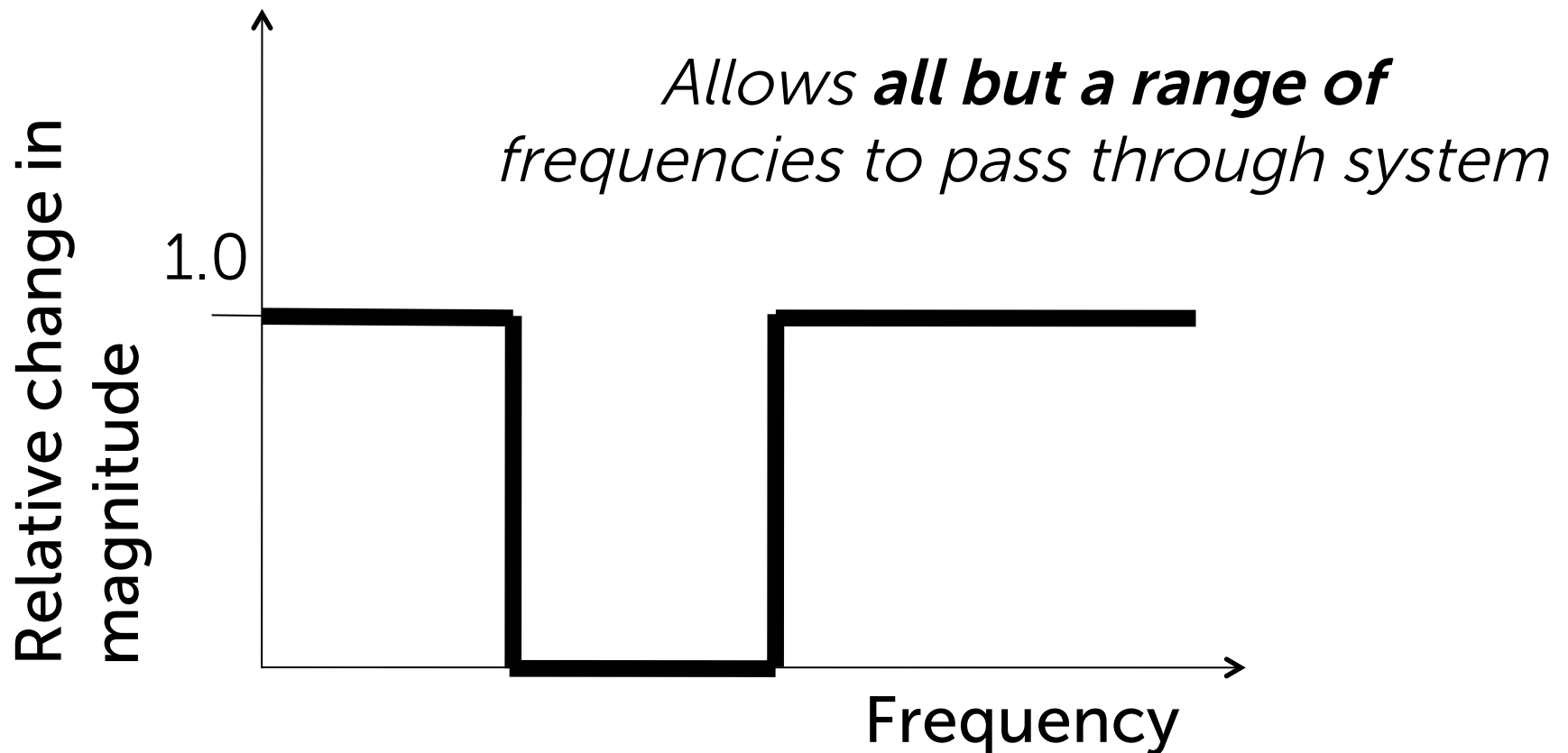
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



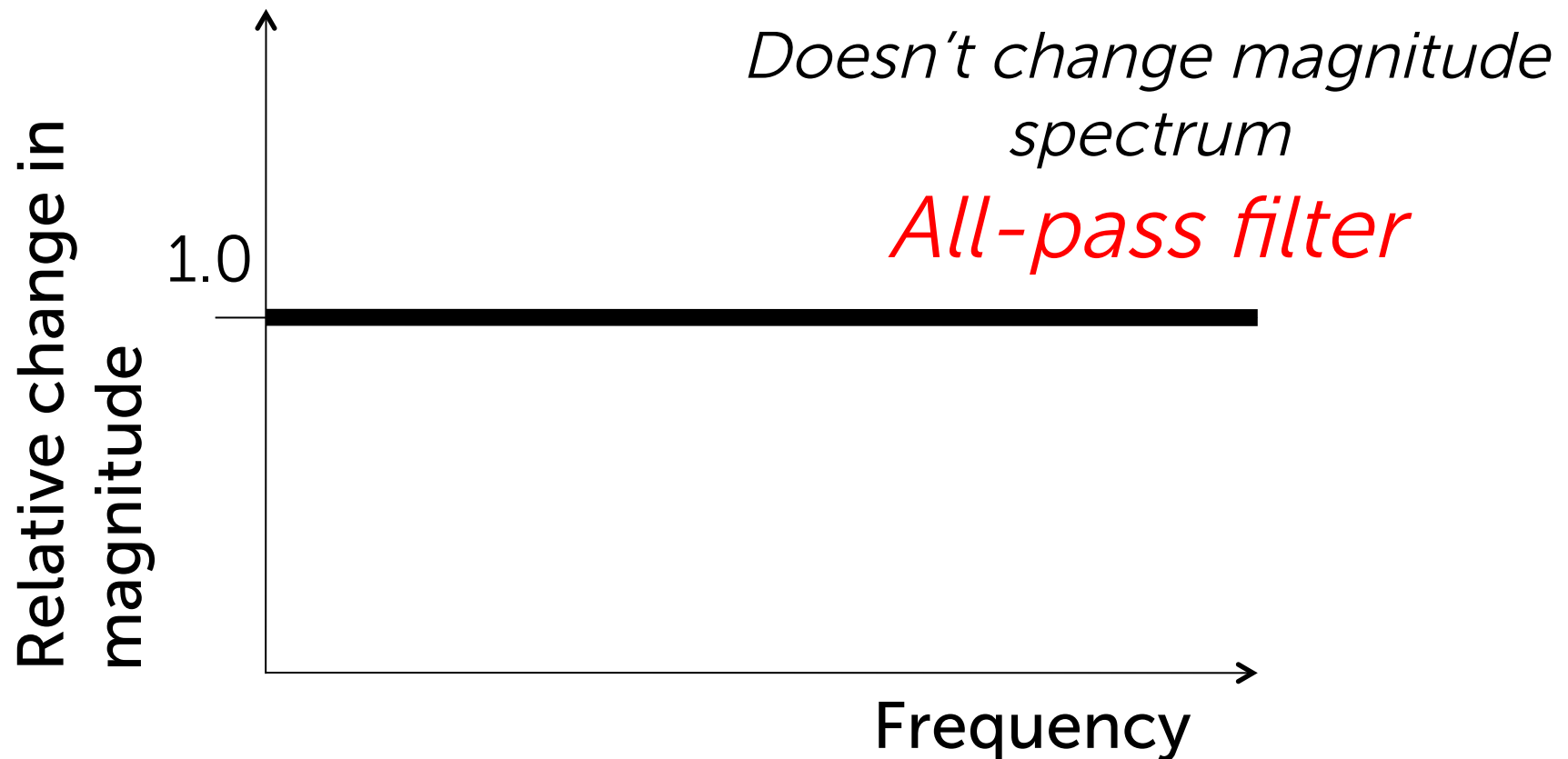


Filters

Each of these systems is an example of a common type of audio filter.

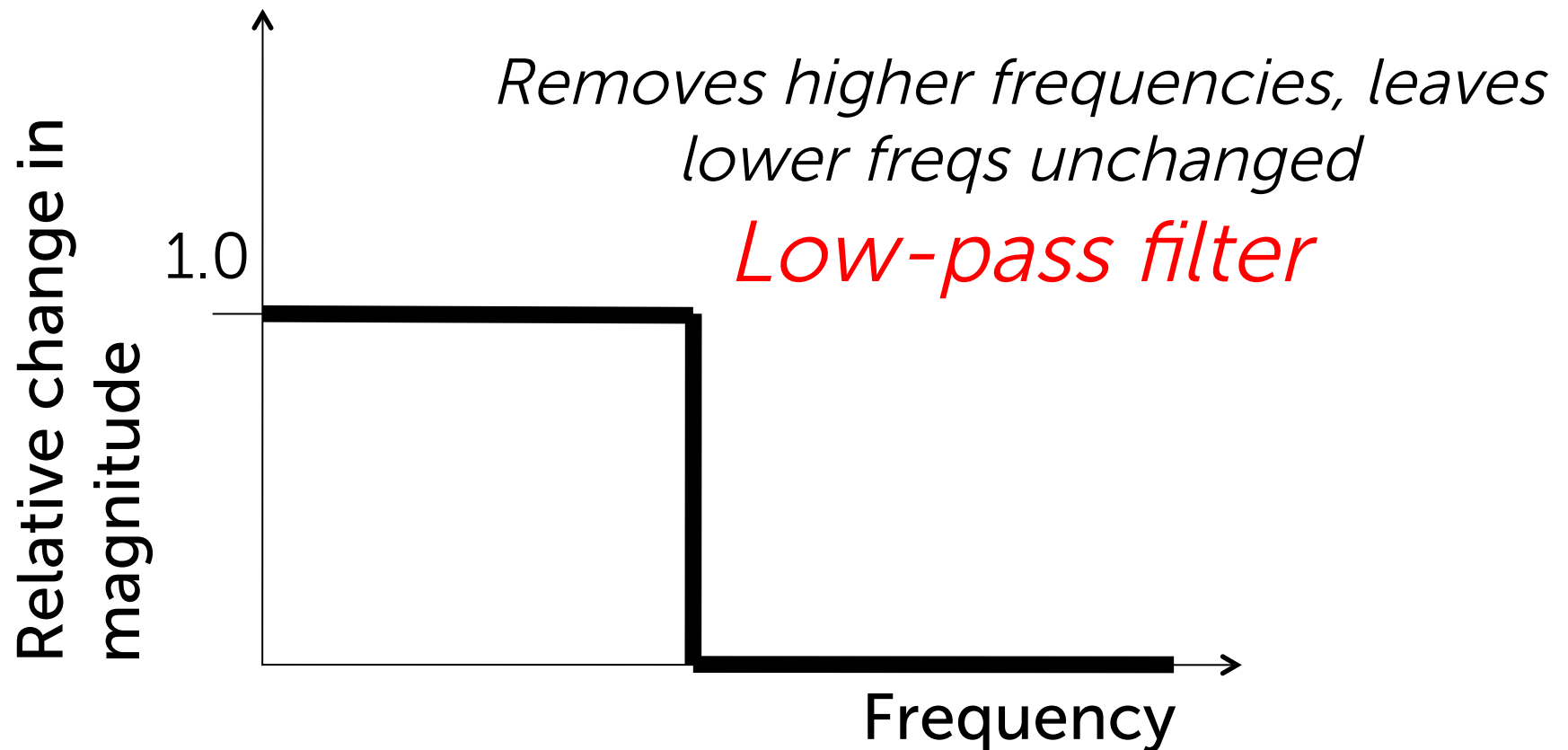
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



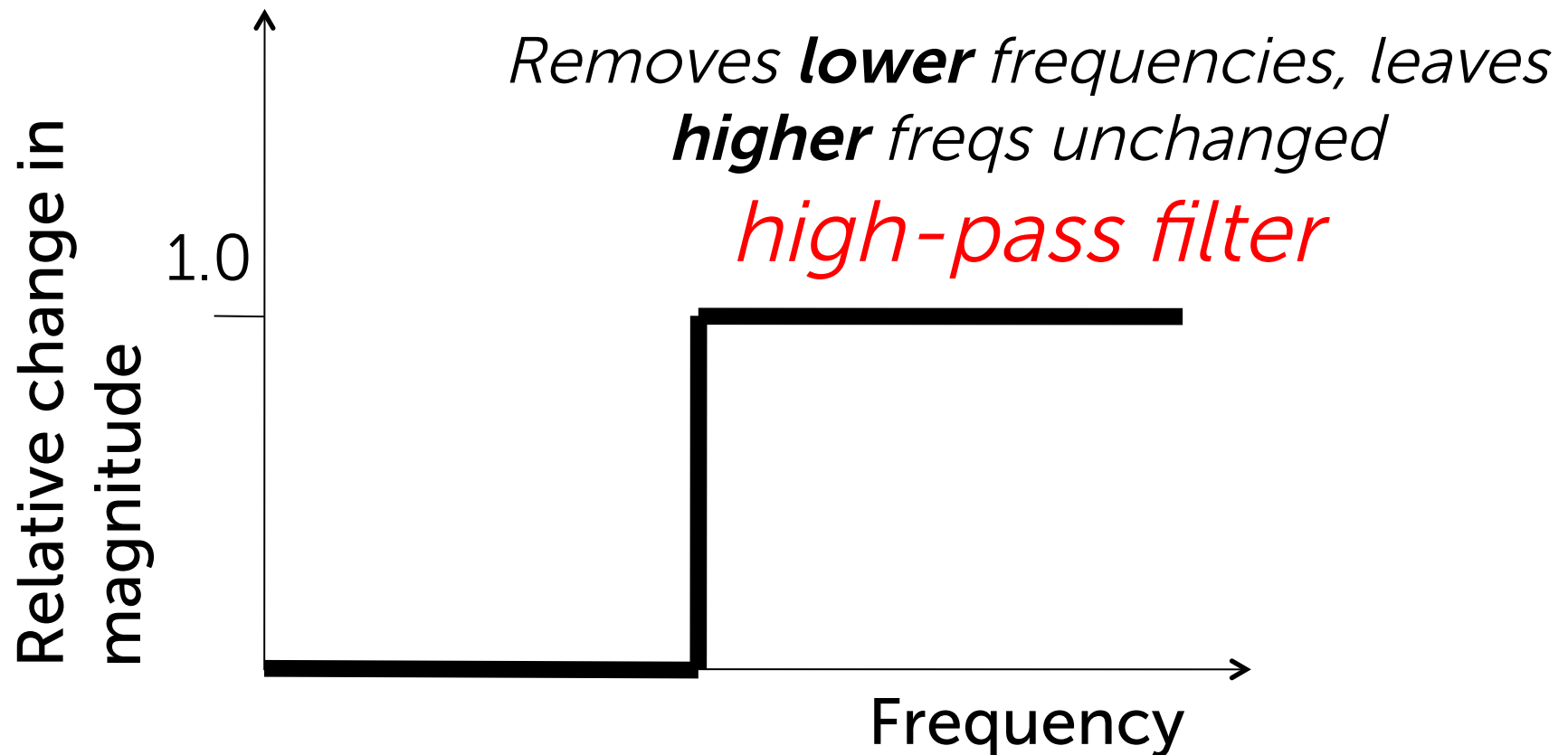
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



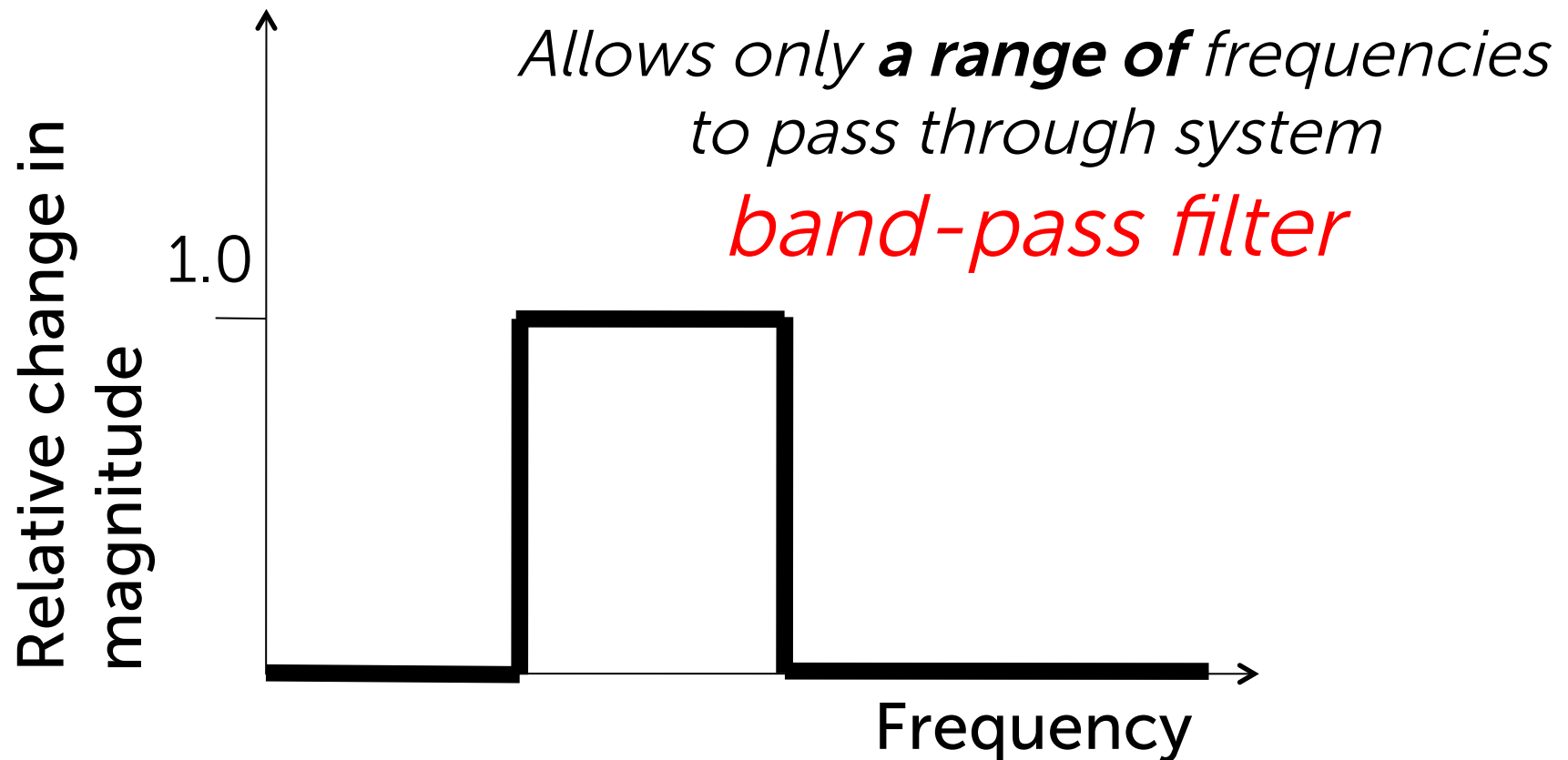
Frequency response

Any LTI system has the ability to change the **spectrum** of a sound



Frequency response

Any LTI system has the ability to change the **spectrum** of a sound

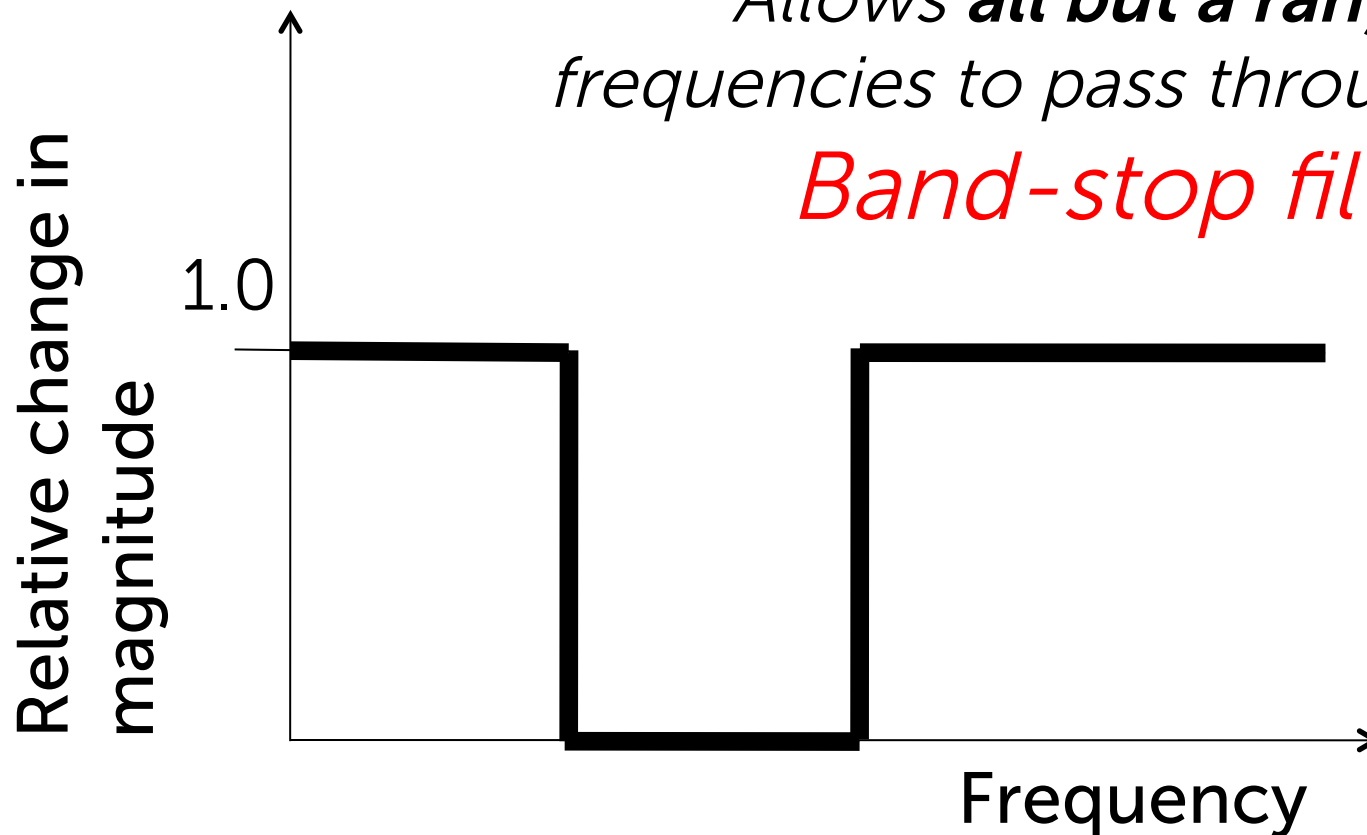


Frequency response

Any LTI system has the ability to change the **spectrum** of a sound

*Allows **all but a range of** frequencies to pass through system*

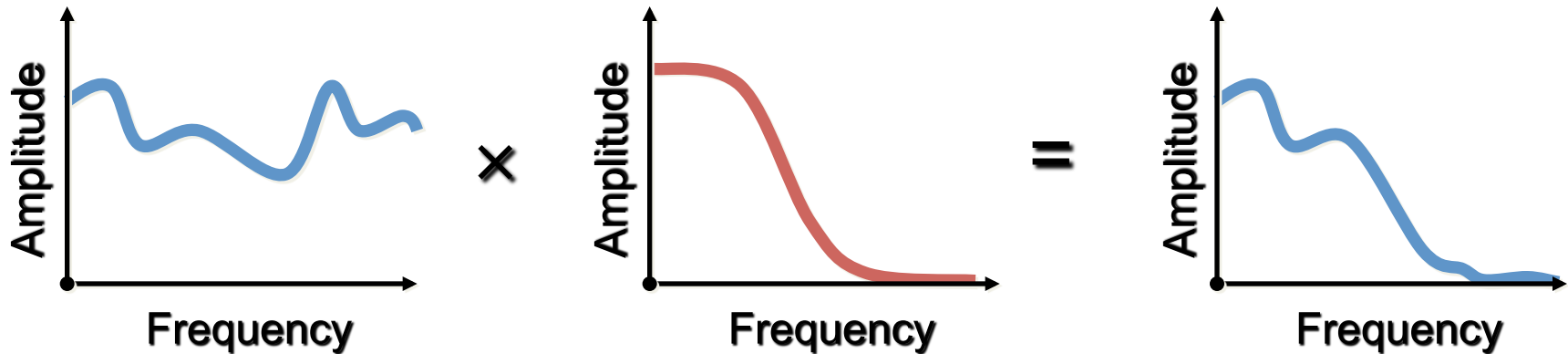
Band-stop filter



The frequency response

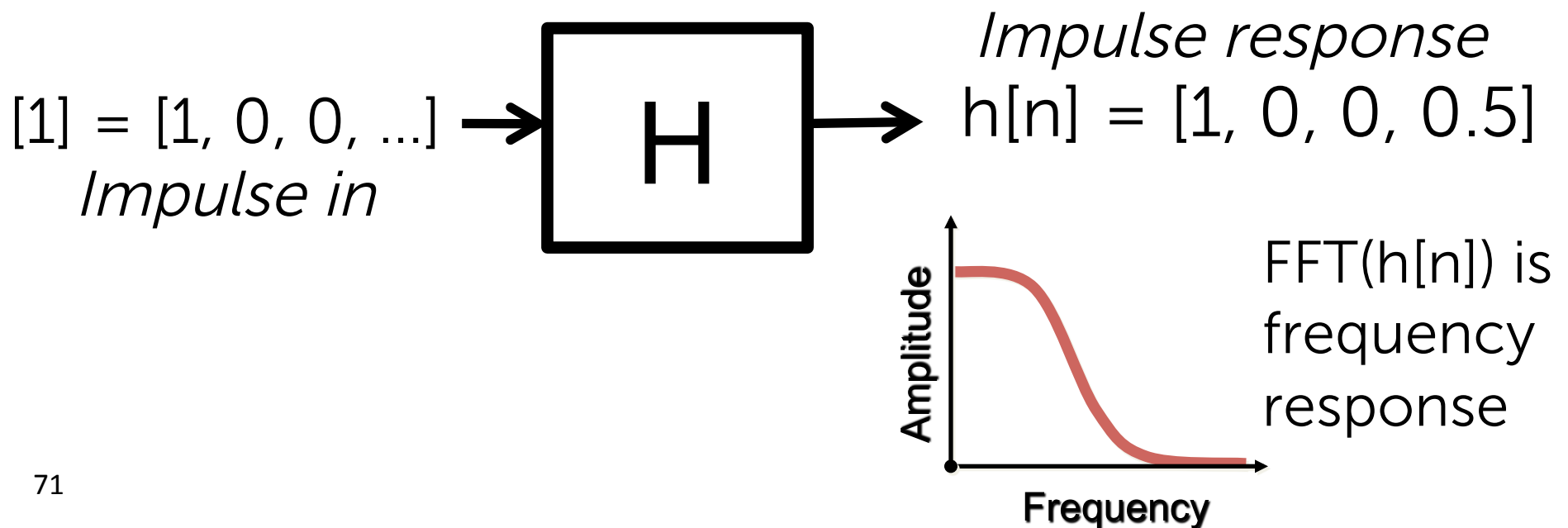
The effect of a system on a signal can be understood as multiplying the signal's spectrum by the frequency response.

*n th bin in input \times n th bin in frequency response
= n th bin in output*



Relationship of frequency response & impulse response

If $h[n]$ is a system's impulse response then the spectrum of $h[n]$ ($\text{FFT}(h[n])$) is the frequency response!





Consequences

- 1) Can take the FFT of $h[n]$ to understand what an arbitrary system with known $h[n]$ will do to a spectrum



Point-wise multiplication in spectral domain = convolution in time domain:

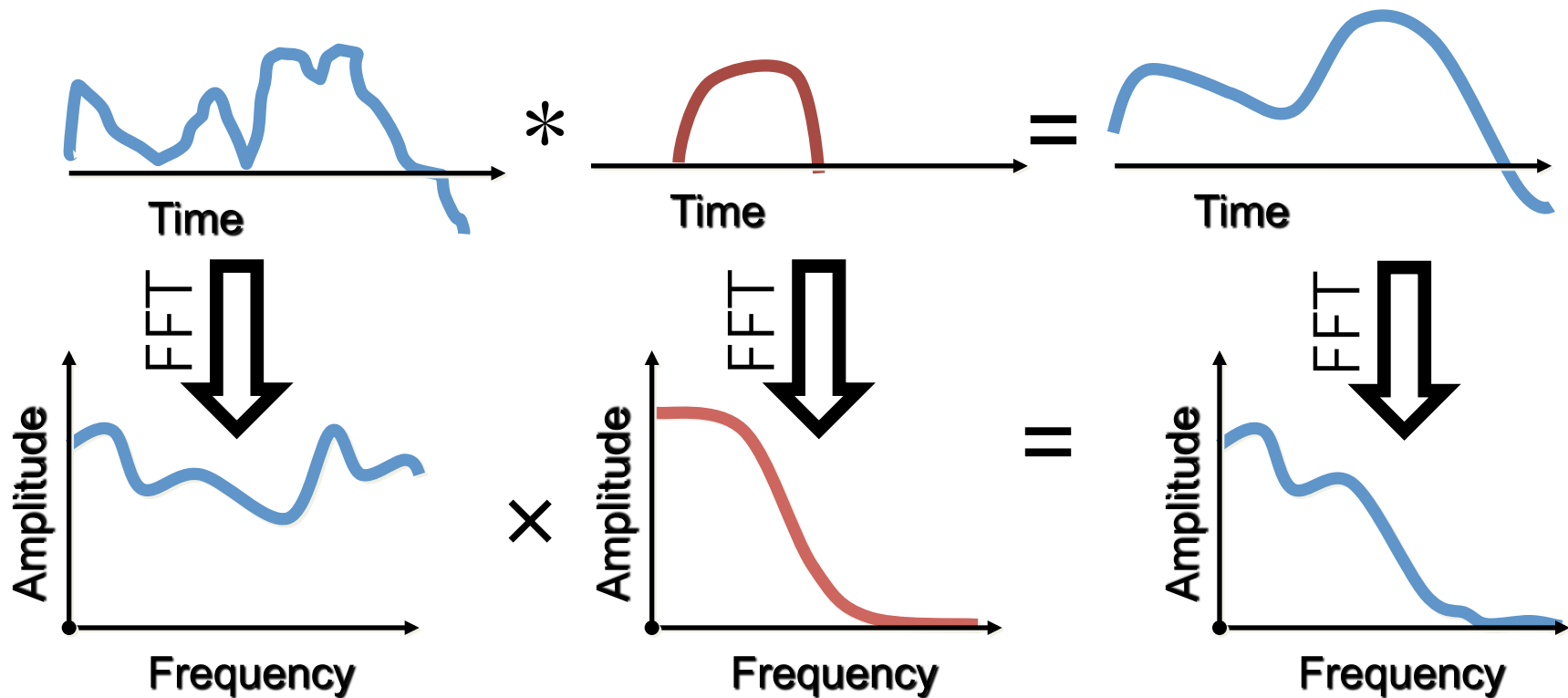
$$a[n] * b[n] \quad \longleftrightarrow \quad A_k \times B_k$$

Point-wise multiplication in time domain = convolution in spectral domain:

$$a[n] \times b[n] \quad \longleftrightarrow \quad A_k * B_k$$

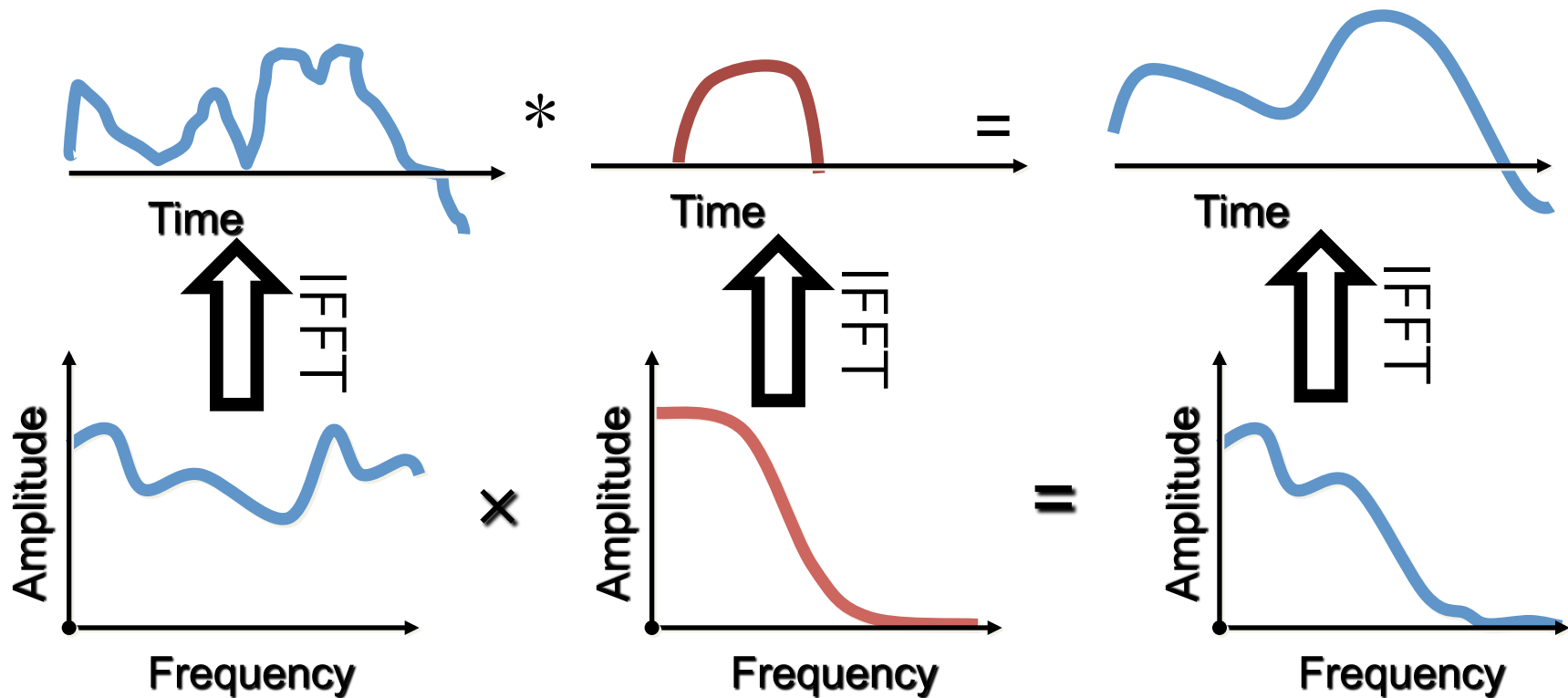
Convolution & Multiplication

Convolving in the time-domain ($x[n] * h[n]$) is equivalent to multiplication in the frequency domain ($X_k \cdot H_k$).



Convolution & Multiplication

Convolving in the time-domain ($x[n] * h[n]$) is equivalent to multiplication in the frequency domain ($X_k \cdot H_k$).



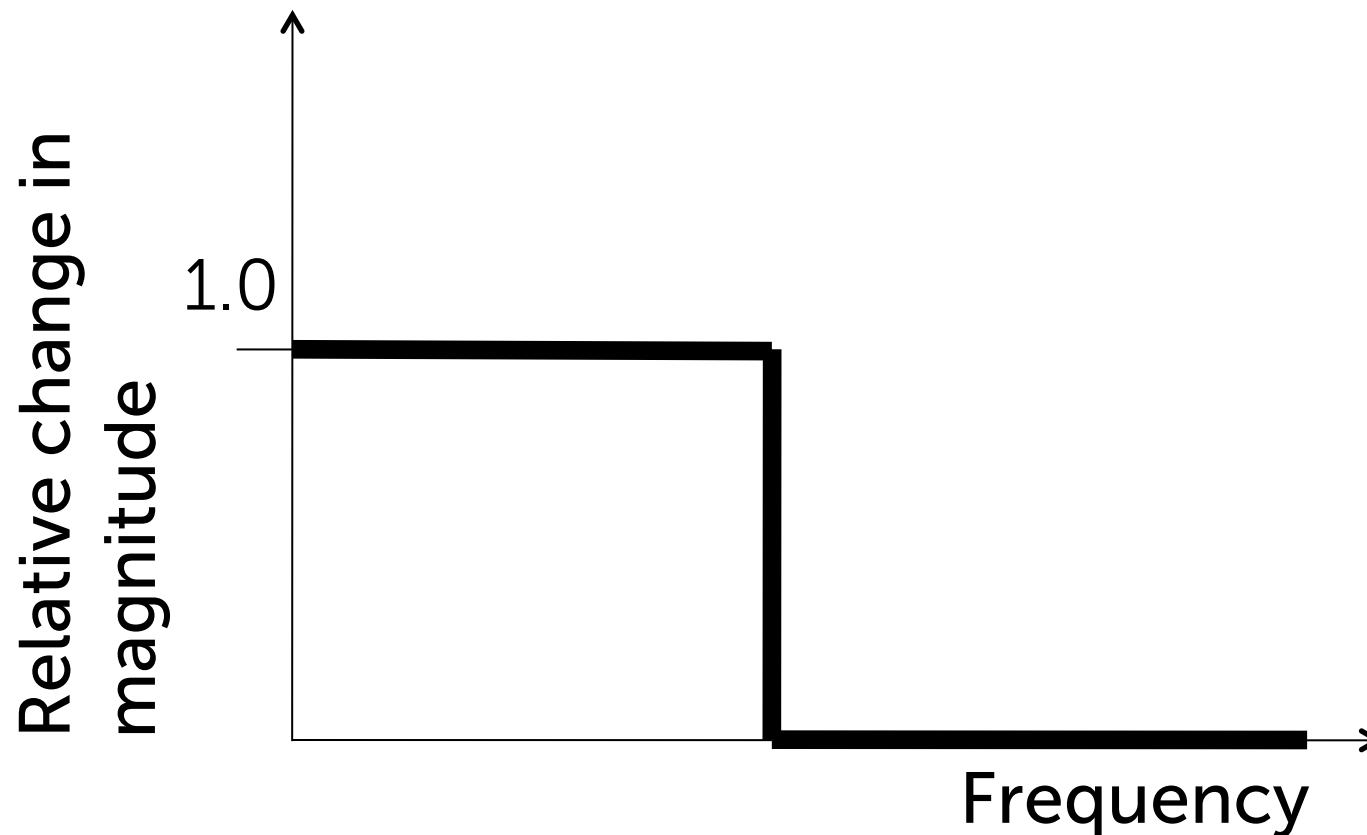


Very important principles

- Convolution in the time-domain ($x[n]$ $h[n]$) is equivalent to multiplication in the frequency domain!
- Also, multiplying in the time domain is equivalent to convolution in the frequency domain.

One big problem...

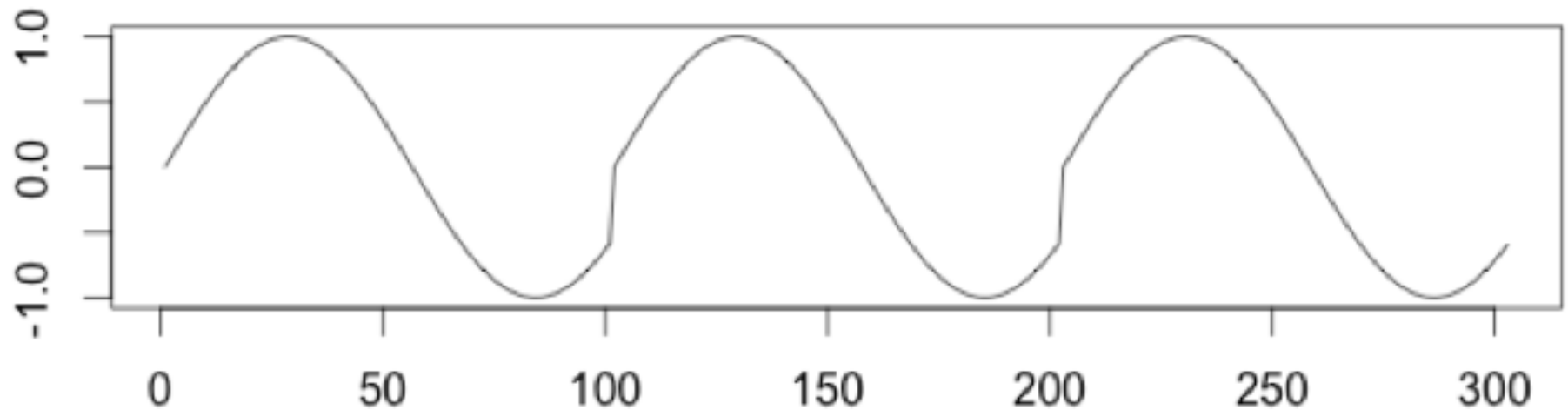
Filters like this are undesirable.



More practical FFT advice

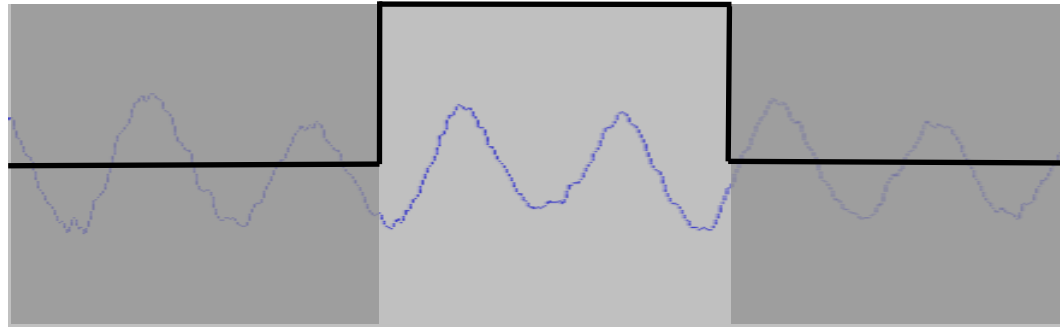
Windowing: Motivation

A problem:



Windowing

“Selecting” N time-domain samples is like point-by-point multiplication with a rectangular function (“window”):



Windowing

A rectangular signal has a very “messy” spectrum!

Signal:

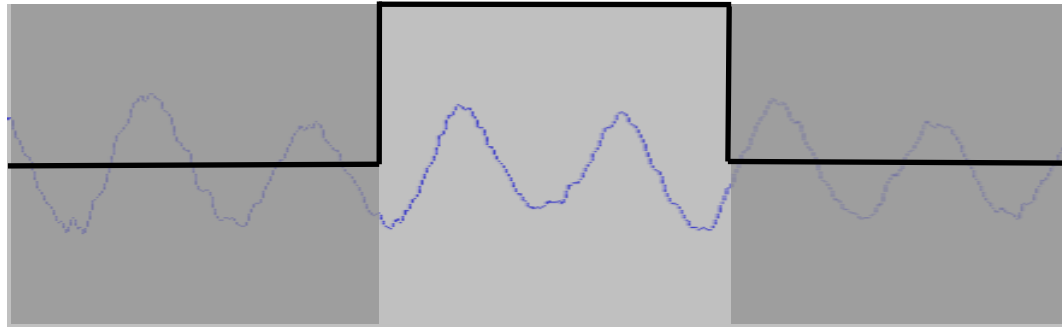


Spectrum:



Windowing

Multiplying a signal by a rectangle in time...



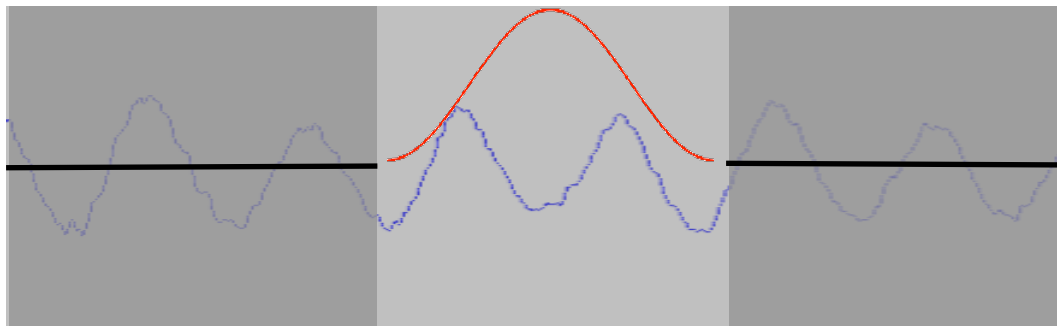
Is equivalent to convolving their spectra!



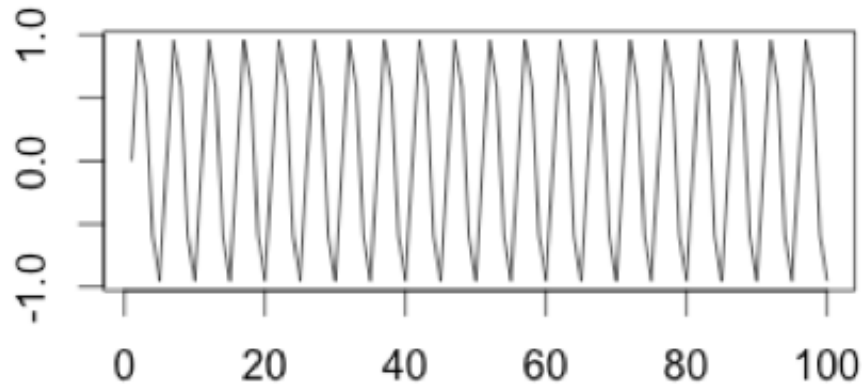
Solution: Apply a smoother window

Before taking FFT, multiply the signal with a smooth window with a “nicer” spectrum

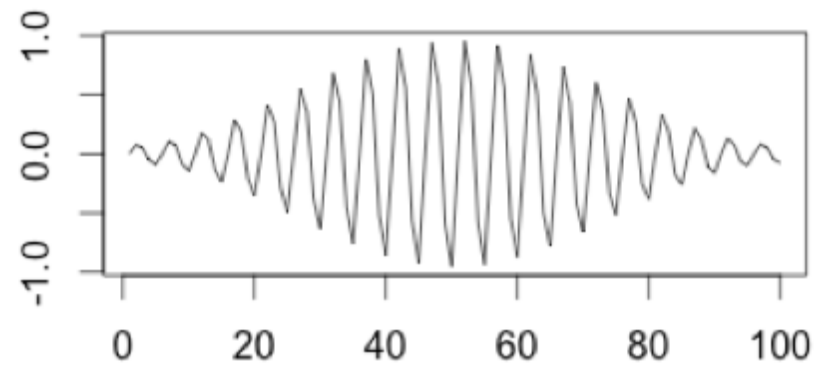
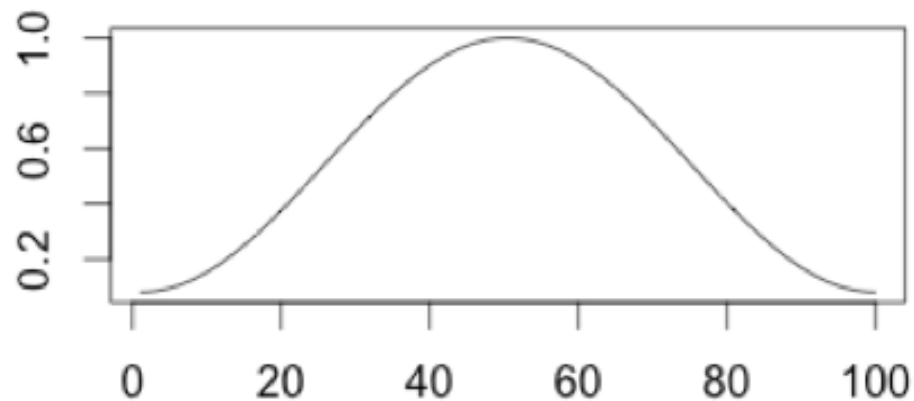
(Equivalently, something that will get rid of sharp edges at either end of analysis frame)



Windowing process

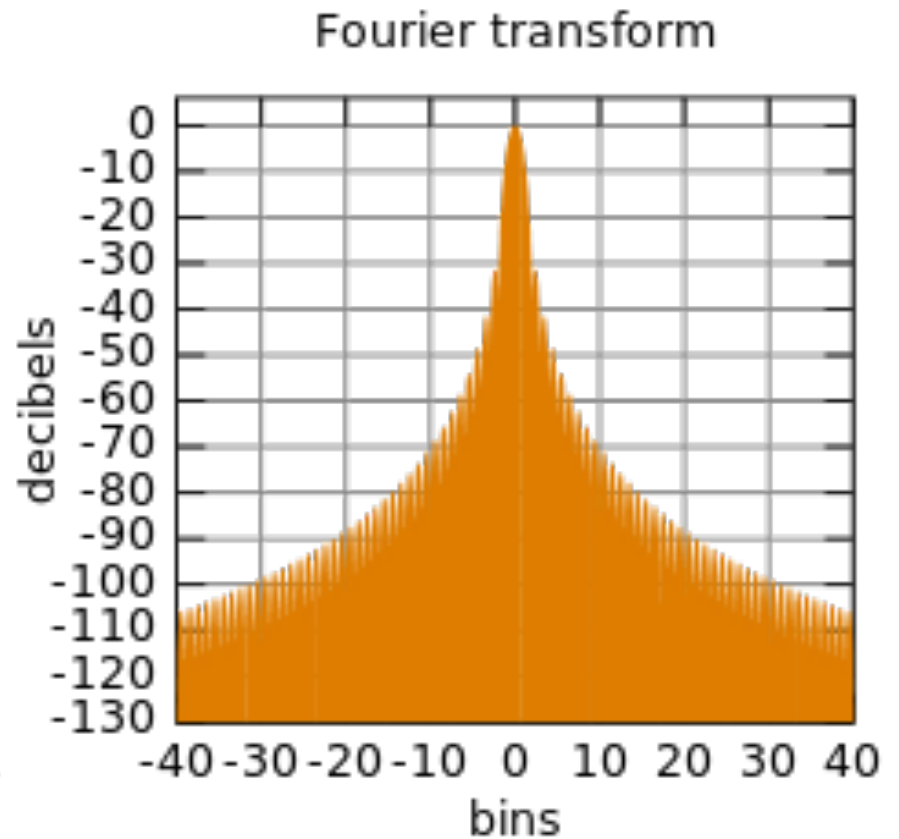
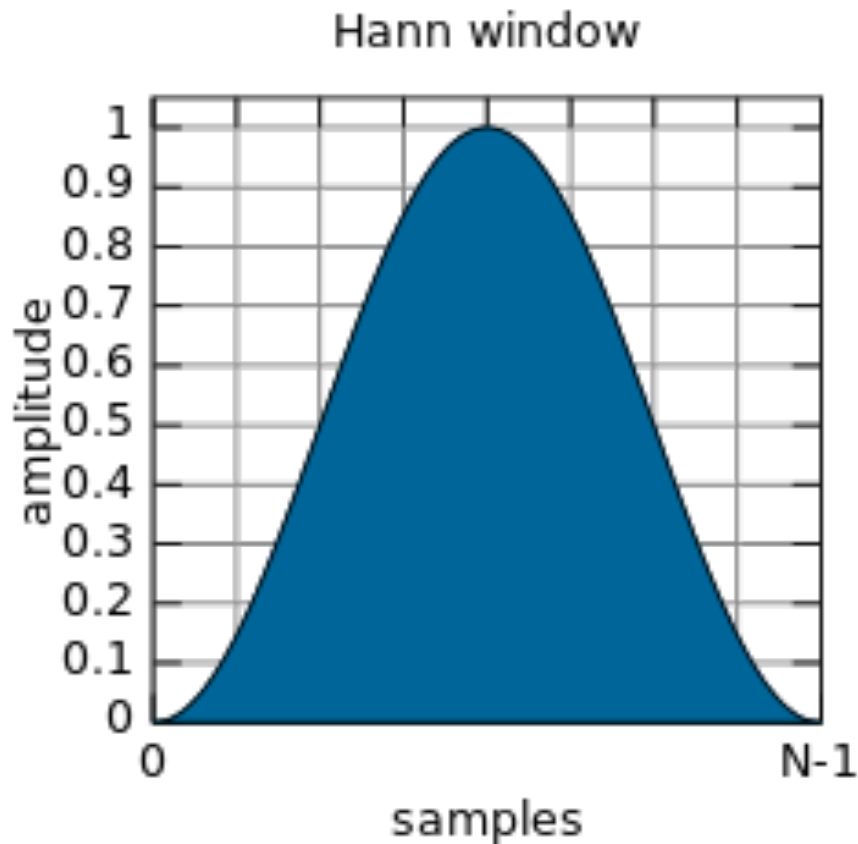


point-wise multiply with
window:



Result
(apply FFT to this)

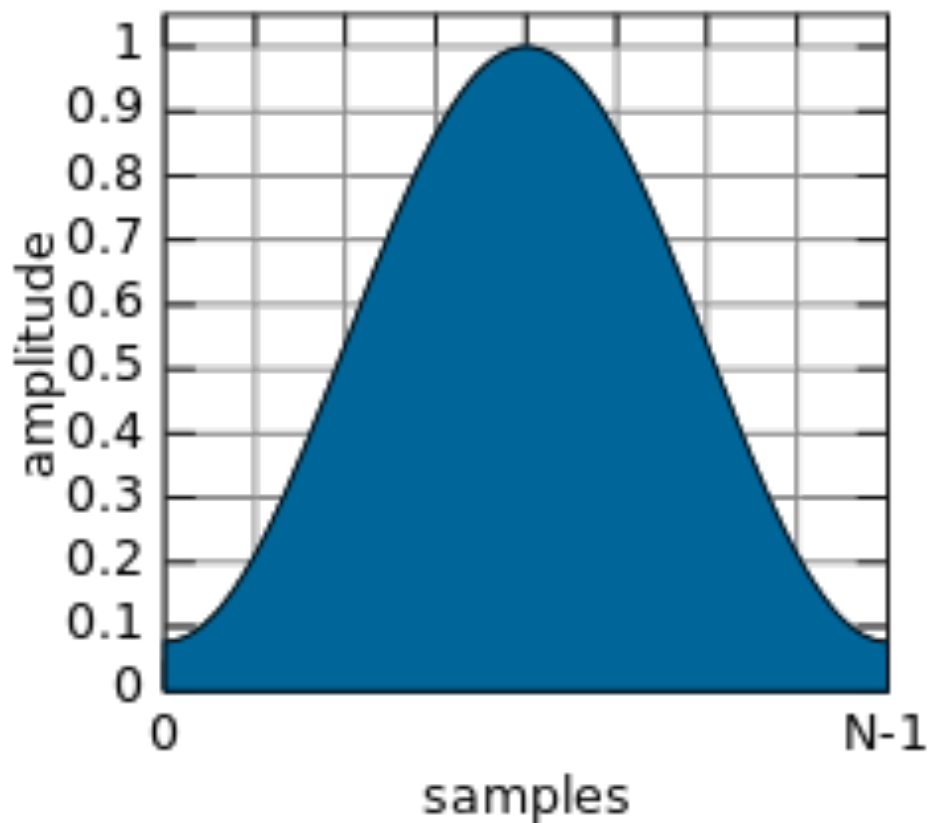
Example windows



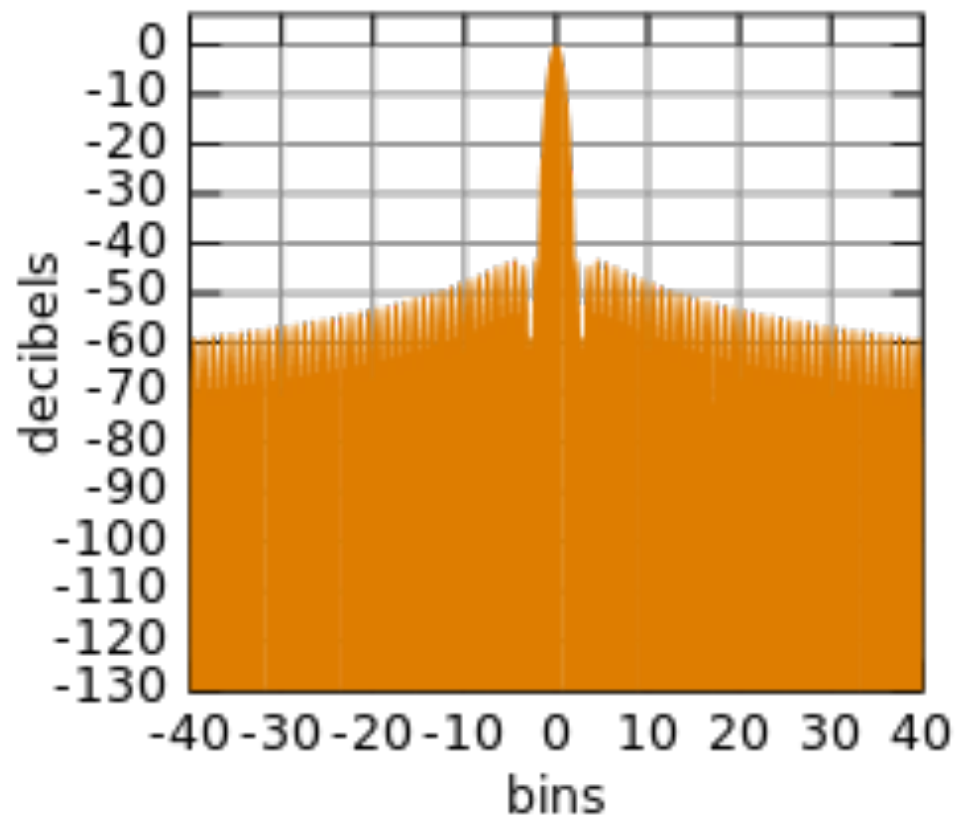
From http://en.wikipedia.org/wiki/Window_function

Example windows

Hamming window ($\alpha = 0.53836$)



Fourier transform



From http://en.wikipedia.org/wiki/Window_function