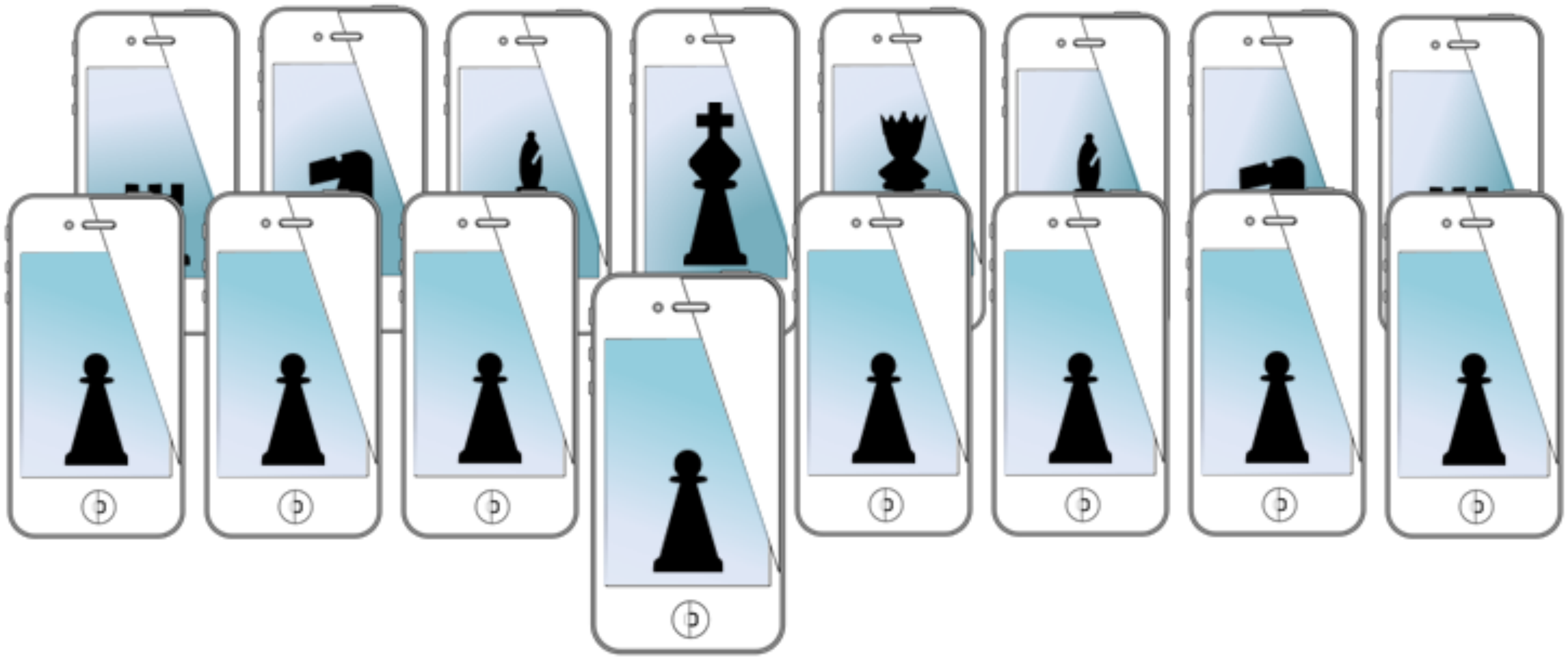
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture six: audio graphing, sampled data, accelerate, & FFT

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

- A1 grades by Wednesday (I hope)
- A2 is up!

Module A
Create an iOS application using the NovocaineExample template that:

- Reads from the microphone
- Takes an FFT of the incoming audio stream
- Displays the frequency of the two loudest tones within 6Hz accuracy
- Is able to distinguish tones as least 25Hz apart, lasting for 100ms or more

The sound source must be external to the phone (i.e., laptop, instrument, another phone, etc.).

Module B
Create an iOS application using the NovocaineExample template that:

- Reads from the microphone
- Plays a settable (via a slider or setter control) inaudible tone to the speakers (15-20kHz)
- Displays the magnitude of the FFT of the microphone data in decibels
- Is able to distinguish when the user is {not gesturing, gestures toward, or gesturing away} from the microphone using Doppler shifts in the frequency

agenda

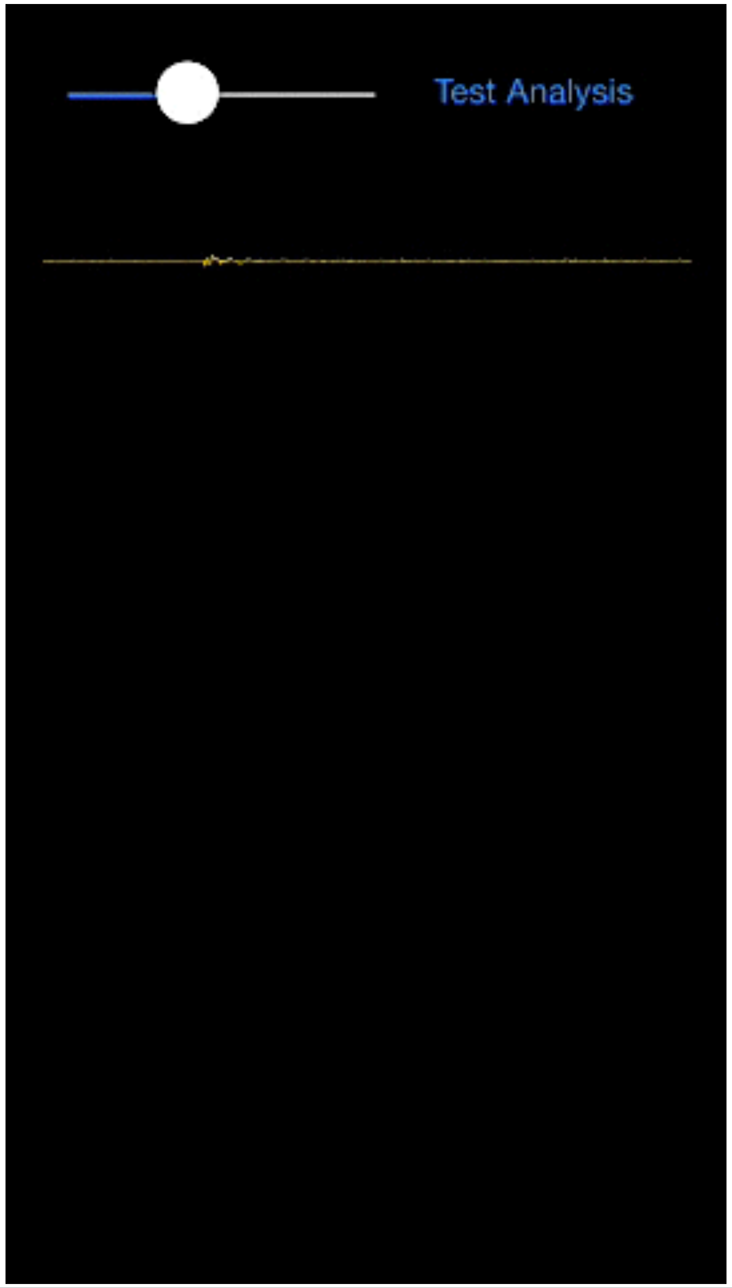
- graphing audio fast (well, graphing anything)
  - must use lowest level graphing, OpenGL
- dealing with sampled data
- the accelerate framework
  - massive digital signal processing library
- the Fourier Transform for spectral analysis

audio graphing

- we want to see the incoming samples
  - good for debugging
  - equalizers
  - oscilloscope type applications

how much data to show?

- sampling at 44.1kHz == 44100 samples per second



0.5 seconds is 22050 samples

display is 640 pixels wide

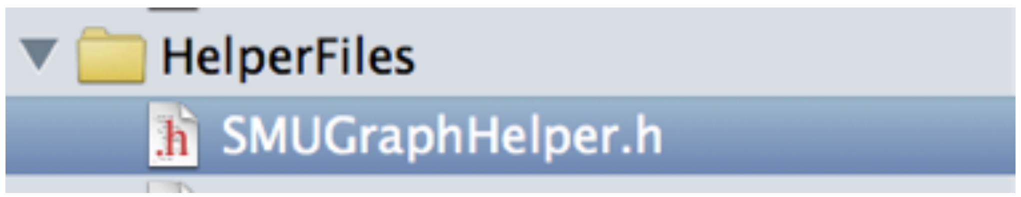
what if we want lots of graphs?

solution

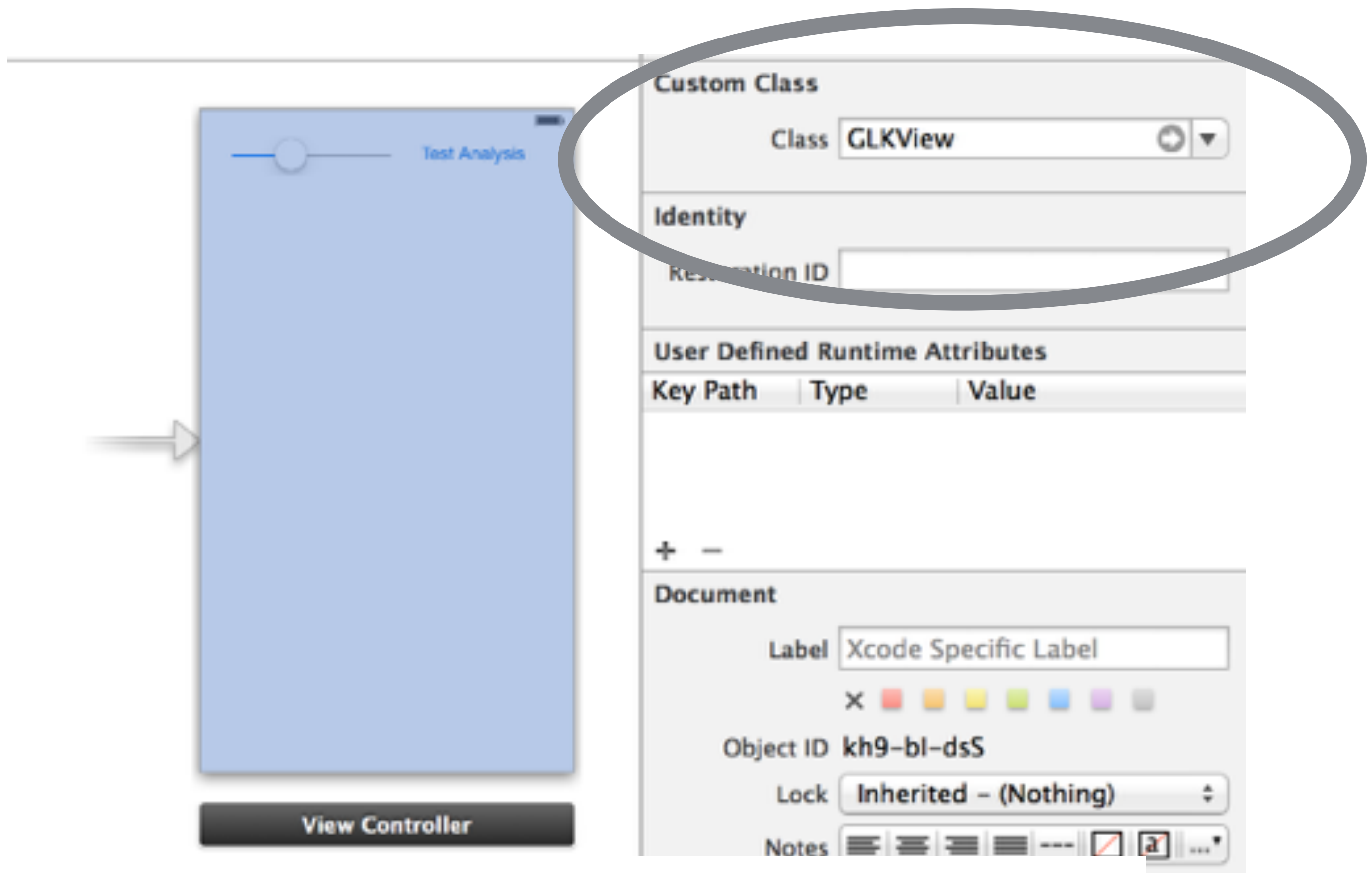
- use the GPU
- set vectors of data on a 2D plane
- let the renderer perform scaling, anti-aliasing, and bit blitting to screen
- …this is not a graphics course

easy solution

- use graph helper, which uses GLKView and GLKViewController



add this to project



add GLKit framework

the graph helper

```
#import <GLKit/GLKit.h>
@interface YourCustomViewController : GLKViewController

GraphHelper     *graphHelper;

In View Did Load
   // start animating the graph
    int framesPerSecond = 15;
    int numDataArraysToGraph = 3;
    graphHelper = new GraphHelper(self,
                                  framesPerSecond,
                                  numDataArraysToGraph,
                                  PlotStyleSeparated);//drawing starts immediately after call

    graphHelper->SetBounds(-0.9,0.9,-0.9,0.9); // bottom, top, left, right,
   //full screen==(-1,1,-1,1)


-(void)dealloc{
    graphHelper->tearDownGL();
    // ARC handles everything else, just clean up what we used c++ for (calloc, malloc, new)
}
```

declare in implementation

inherit from open GL

setup GLKViewController

bounds for screen

```
enum PlotStyle {
    PlotStyleOverlaid,
    PlotStyleSeparated
};
```

setting data

```
//  override the GLKViewController draw function, from OpenGLES
- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect {
    graphHelper->draw(); // draw the graph
}
```

called for each draw to screen

```
void setGraphData(int graphNum, float *data, int dataLength, float normalization = 1.0,
                  float minValue = 0.0)
```

prototype for setting scatter data

```
//  override the GLKView update function, from OpenGLES
- (void)update{

    graphHelper->setGraphData(0,                       //channel index
                              inputAudioDataBuffer,    //data
                              kBufferLength/2.0,       //data length
                              64.0);                   // max value to normalize (==1 if not set)

    // just plot the audio stream
    graphHelper->setGraphData(1,inputAudioDataBuffer,kBufferLength); // set graph channel

    graphHelper->update(); // update the graph
}
```

set data for 0th graph

set data for 1st graph

update render state

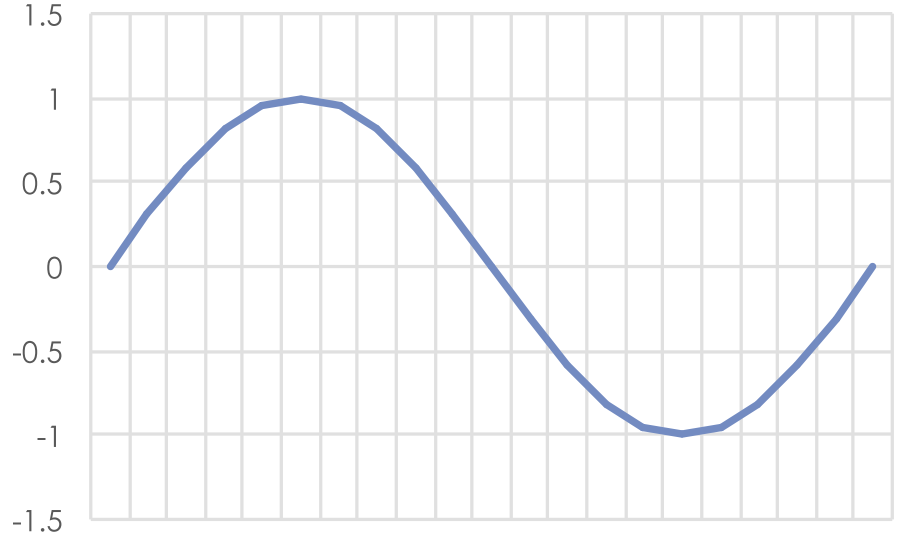
audio graphing demo!

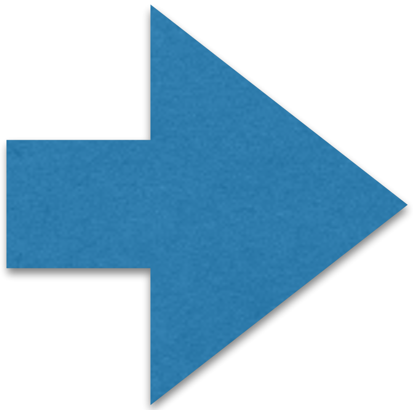
intro to sampled data

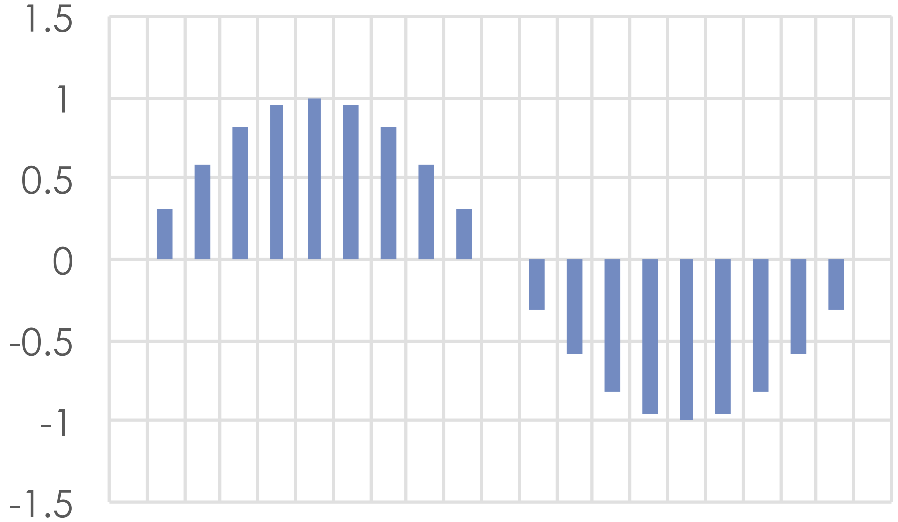
- why is understanding sampled data important?
  - because we'll be dealing with it all semester
  - it's important to understand basic mistakes that can be made
- there are entire courses dedicated to sampled time series
  - actually entire courses on analyzing frequency content
- we'll touch on a few guidelines to help you design your projects better

intro to sampled data

- physical processes are continuous
  - to process with computers, we must digitize it
  - digitization can change how we understand the signal
- digitization occurs in time and amplitude
  - time: sampling
  - amplitude: quantization



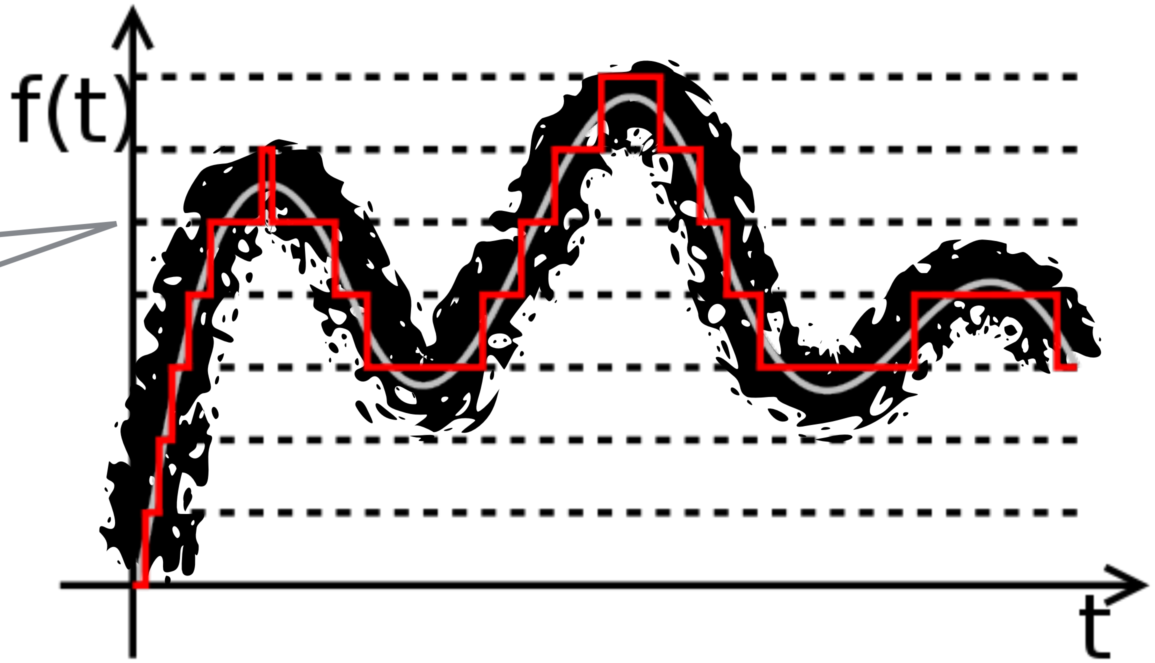




sampled data

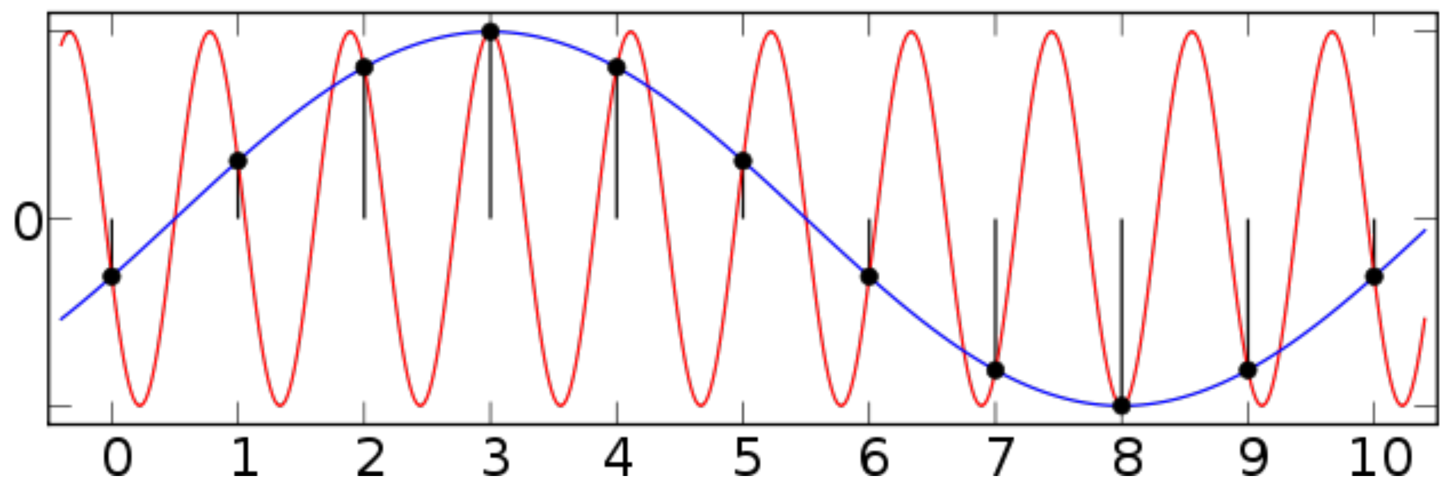
- quantization (amplitude)
    - introduces error in estimating amplitude of a signal
    - error can be reduced by adding more “bits per sample”
- most ADCs are 16 bits, considered “good enough”
- sufficient for most uses
    - not for others!

iPhone uses LPCM 32 bits, Q8.24



sampling errors

- sampling in time
  - introduces errors through ‘aliasing’
  - limits the range of frequencies able to be accurately captured
  - root of most common mistakes with sampled data



so how do I sample?

- heuristics
  - don’t try to sample extremely small increments or values!
  - if capturing an “X”Hz signal, need to sample at least 2“X” Hz
  - changing sample rates is complicated, don’t just drop every other sample
- for example, speech
  - majority of necessary energy in speech is located < 8000Hz
  - phones (for speech) typically capture at 16KHz or lower
  - good enough for speech, not music!

sanity check

- I need to detect an 80Hz signal
  - what sampling rate should we use?

- I want to detect a feather dropping next to the microphone
  - can the sound be detected?

the accelerate framework

- very powerful digital signal processing (DSP) library
  - look at vDSP Programming Guide on developer.apple.com for the complete API
- provides mathematics for performing fast DSP

input data | stride | scalar | output | array length

```
vDSP_vsmul(data, 1, &mult, data, 1, numFrames*numChannels);
```

```
void vDSP_vsmul (
   const float __vDSP_input1[],
   vDSP_Stride __vDSP_stride1,
   const float *__vDSP_input2,
   float __vDSP_result[],
   vDSP_Stride __vDSP_strideResult,
   vDSP_Length __vDSP_size
);
```

an example

```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels) {
        float volume = userSetVolumeFromSlider;
        vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels);
        ringBuffer->AddNewInterleavedFloatData(data, numFrames, numChannels);
    }];
```

```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels) {

        // get the max
        float maxVal = 0.0;
        vDSP_maxv(data, 1, &maxVal, numFrames*numChannels);

        printf("Max Audio Value: %f\n", maxVal);

    }];
```

```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
 {
     vDSP_vsq(data, 1, data, 1, numFrames*numChannels);
     float meanVal = 0.0;
     vDSP_meanv(data, 1, &meanVal, numFrames*numChannels);
 }];
```

processing audio

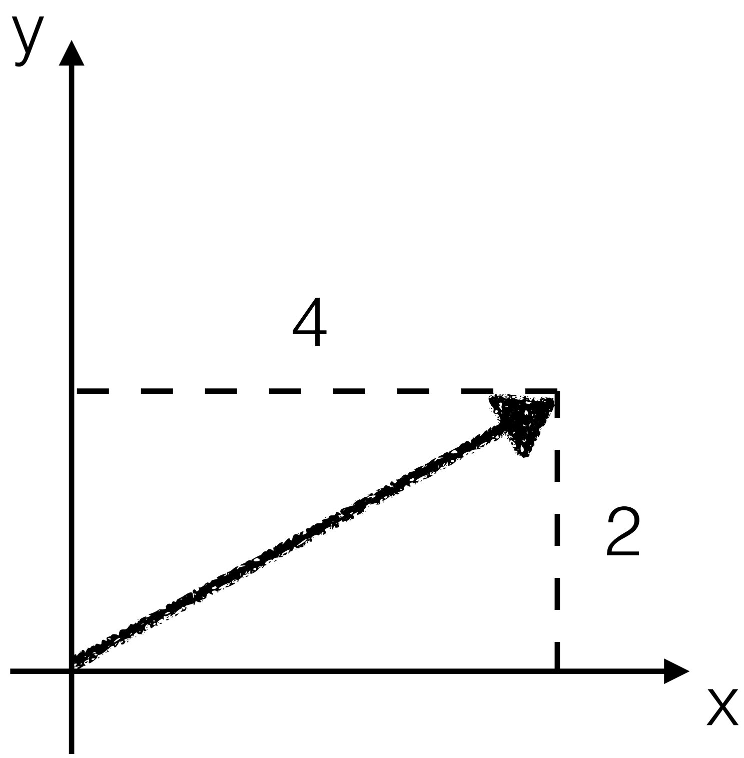
- lots of space to explore
  - great reference: “DSP First” by McClellan, Schafer, Yoder
    - http://www.rose-hulman.edu/DSPFirst/visible3/contents/index.htm
  - filtering
    - manipulate signal: high, low, bandpass
  - analysis
    - analyze characteristics of signal (like speech recognition)
  - synthesis
    - play around with different ideas, and see what sounds good!
    - not just pure synthesis, but also manipulation (like a guitar effect)
- for now, we’re going to stick with analysis
  - specifically, the **Fourier Transform**

the Fourier transform

- extremely useful, not just for signal junkies but also:
  - computer scientists, engineers, physicists, mathematicians, astronomers, oceanographers, health care professionals, etc.
- the Fourier Transform (FT) converts a time series into a frequency spectrum
  - the spectrum is an array of complex numbers which we will represent in polar form (i.e., with magnitude and phase)
  - each complex number represents a sinusoidal wave at a specific frequency
- we will use the FFT in the accelerate framework
  - complexity is $O(N \log_2(N))$ (for radix 2 FFT)

what is the FFT?

think of it as a vector projection (intuitively)



(4,2)

where did these numbers come from?

$\vec{\mathbf{x}}$ $(1,0) \bullet (4,2) = 4$

$\vec{\mathbf{y}}$ $(0,1) \bullet (4,2) = 2$

point by point multiplication and add it up!

$\vec{\mathbf{x}} \bullet \vec{\mathbf{y}} = 0$

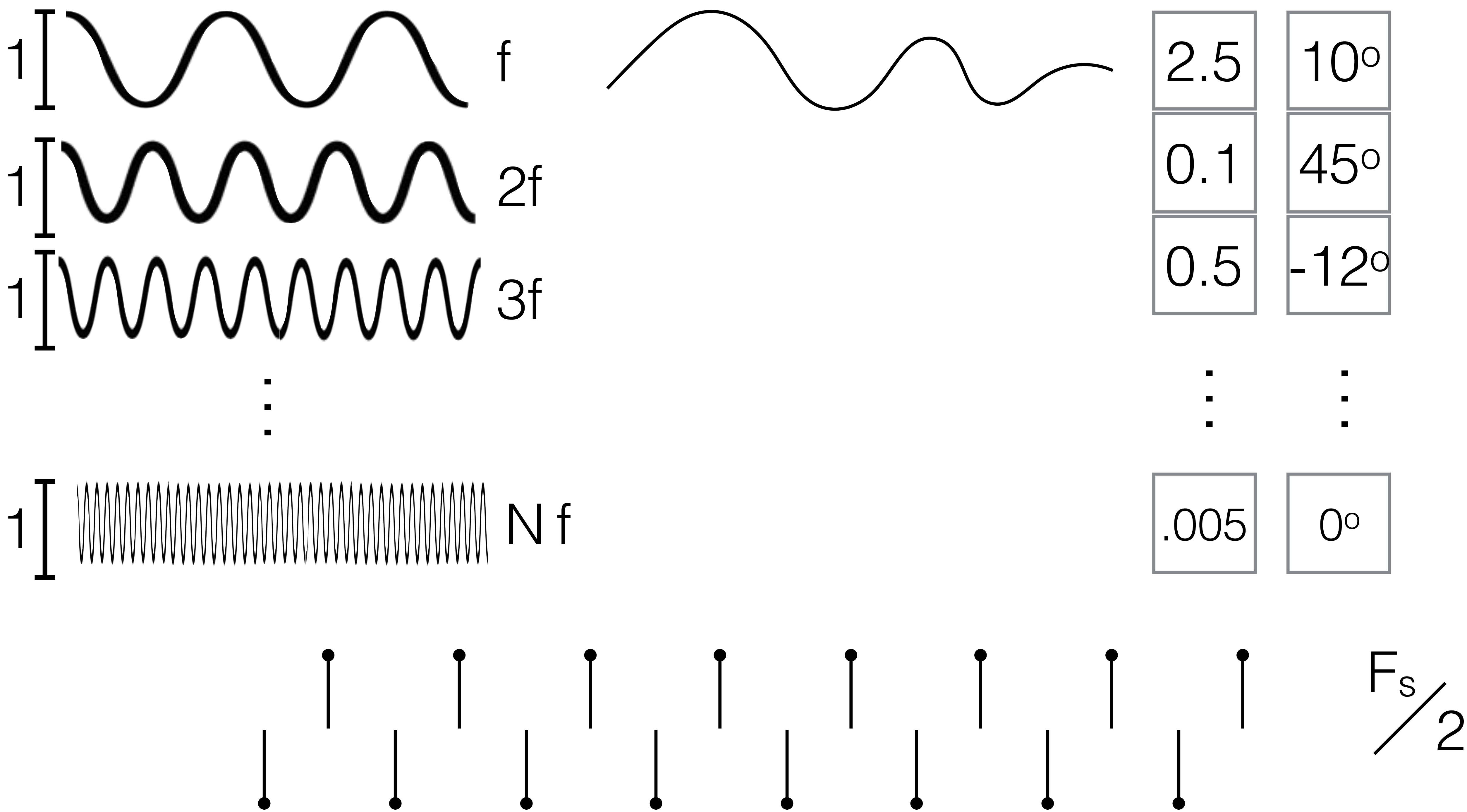
$|\mathbf{x}| = |\mathbf{y}| = 1$

tells us “how much” of the vector $\vec{\mathbf{x}}$

is the results of other orthogonal vectors $\vec{\mathbf{y}}$

what is the FFT?

what if the orthogonal vectors were functions?

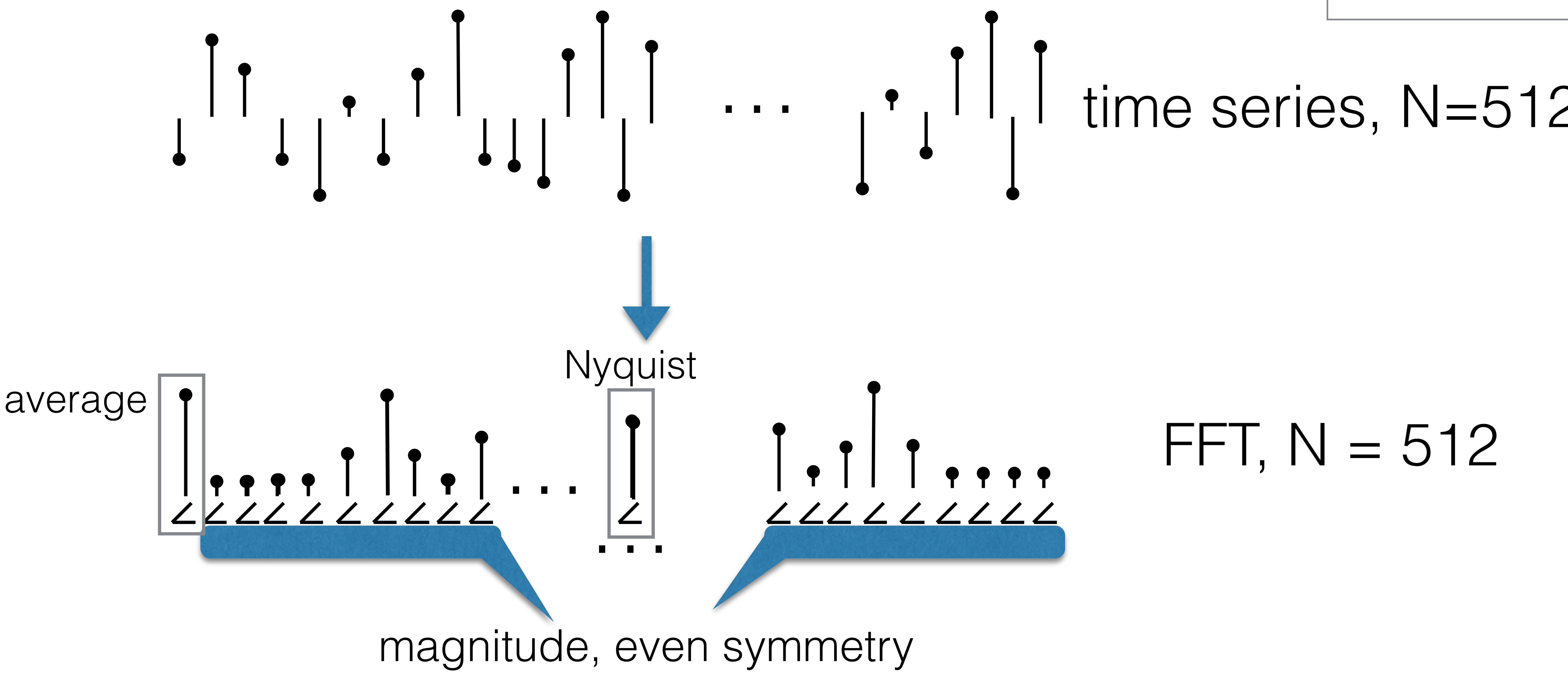


the FFT

frequency content

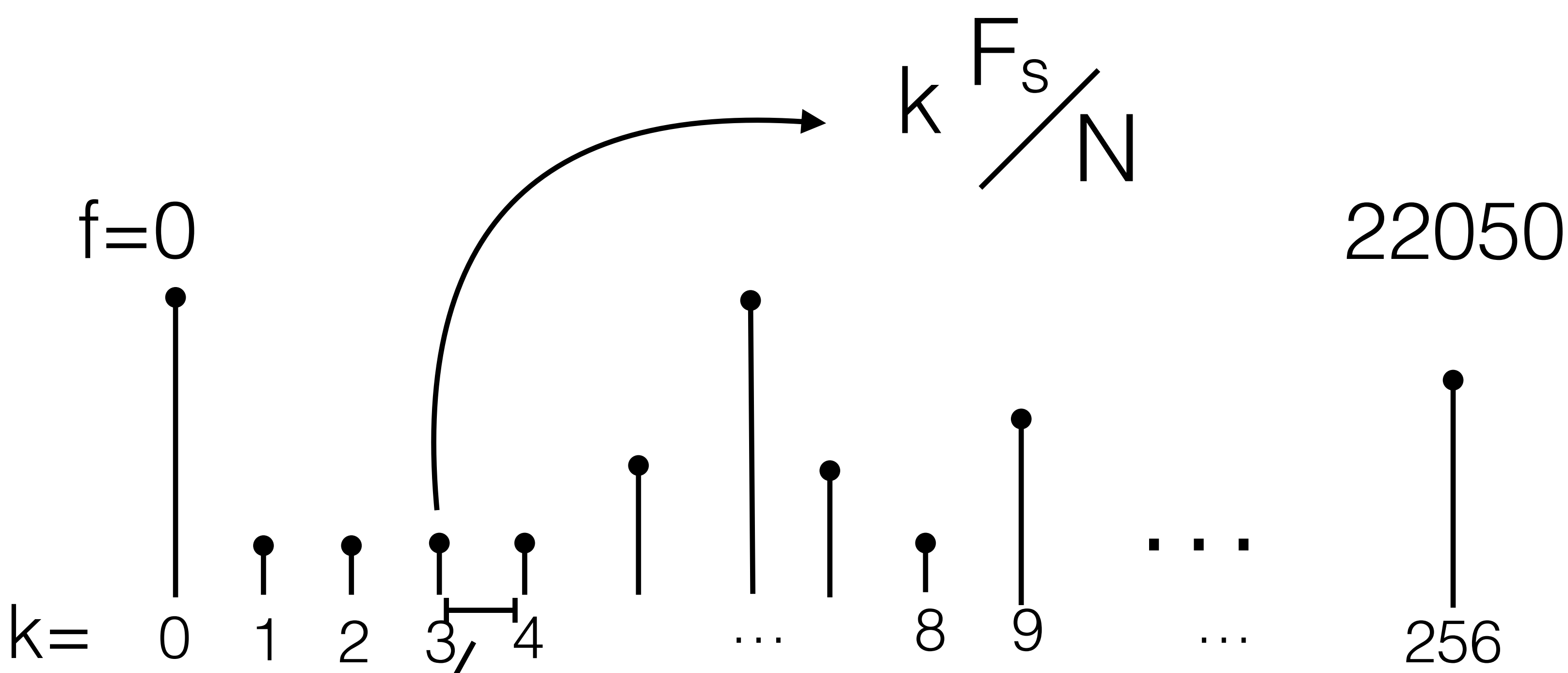
0f	0.7	0^o					=	0.7
1f	2.5	10^o	+	2.5	-10^o	-1f	~	2.5 cos(2*pi* (f) t+10^o)
2f	0.1	45^o	+	0.1	-45^o	-2f	~	0.1 cos(2*pi* (2f) t+45^o)
3f	0.5	-12^o	+	0.5	12^o	-3f	~	0.5 cos(2*pi* (3f) t-12^o)
	⋮	⋮		⋮	⋮			
N f	.005	0^o	+	.005	0^o	-N f	~	0.005 cos(2*pi* (Nf) t)

time and frequency



magnitude spectrum

time and frequency



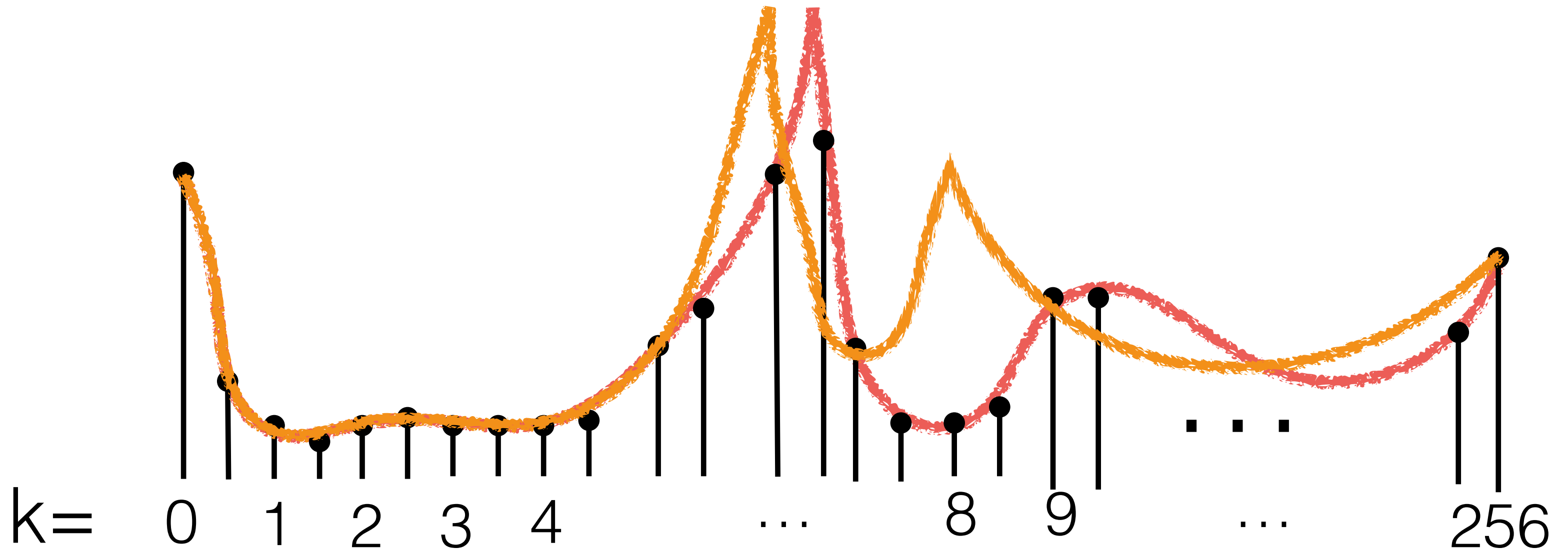
FFT, N = 512

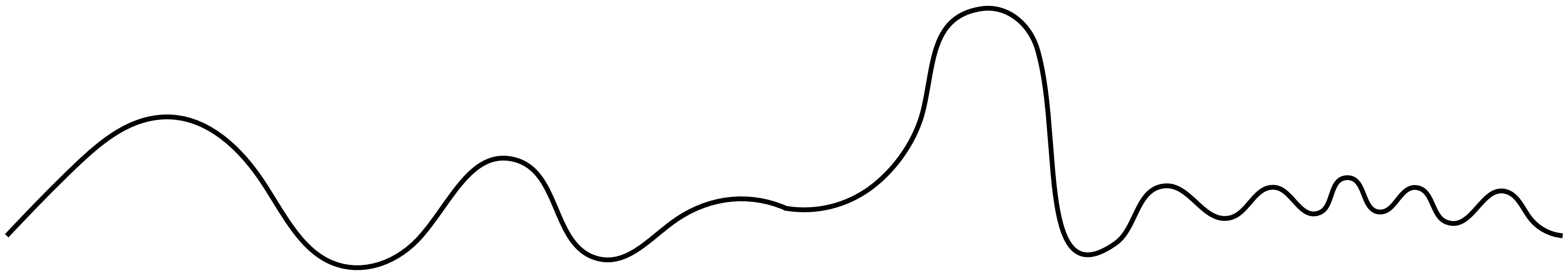
Spectrum, N = 257

Fs = 44100 Hz

$$df = \frac{F_s/2}{N/2} = F_s/N = 22050/512 = 43.06 \text{ Hz}$$

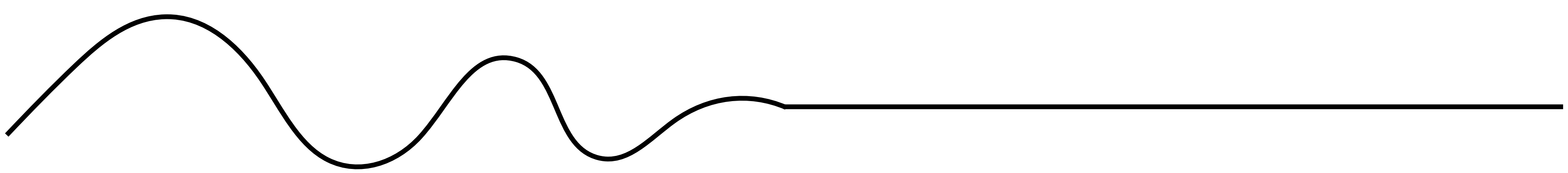
using the FFT





512 points

1024 points



solution!

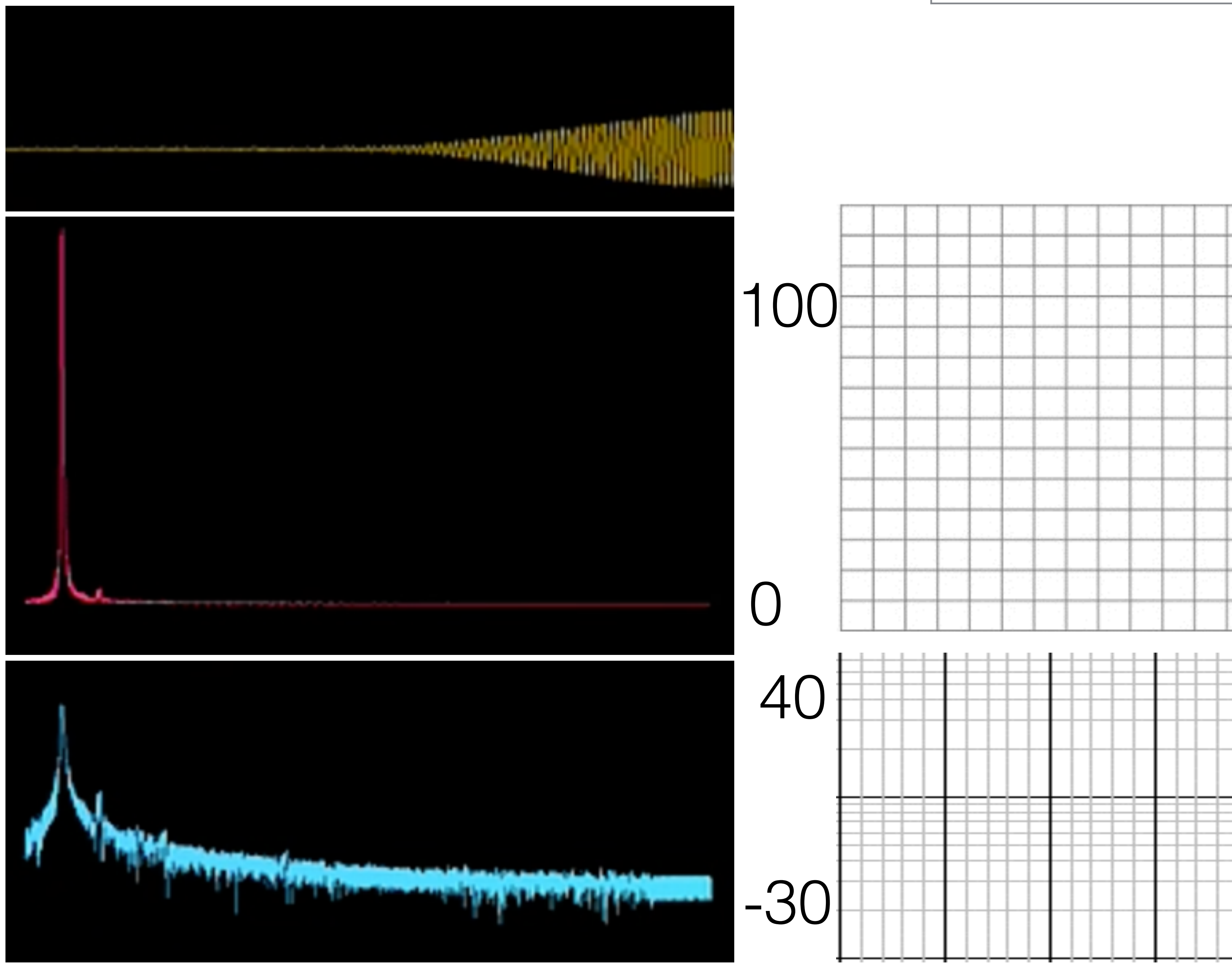
zero padding

using the FFT

raw audio

magnitude FFT

magnitude FFT
in dB
$20 \log_{10}(|FFT|)$



programming the FFT

```
#import "SMUFFTHelper.h"

float           *fftMagnitudeBuffer;
float           *fftPhaseBuffer;
SMUFFTHelper    *fftHelper;
```

fft size | window size | window type

```
//setup the fft
fftHelper = new SMUFFTHelper(kBufferLength,kBufferLength,WindowTypeRect);
fftMagnitudeBuffer = (float *)calloc(kBufferLength/2,sizeof(float));
fftPhaseBuffer     = (float *)calloc(kBufferLength/2,sizeof(float));
```

```
enum WindowType {
    WindowTypeHann,
    WindowTypeHamming,
    WindowTypeRect,
    WindowTypeBlackman,
    };
```

```
free(fftMagnitudeBuffer);
free(fftPhaseBuffer);
delete fftHelper;
```

tear down in dealloc

```
fftHelper->forward(0,inputAudioDataBuffer, fftMagnitudeBuffer, fftPhaseBuffer);
```

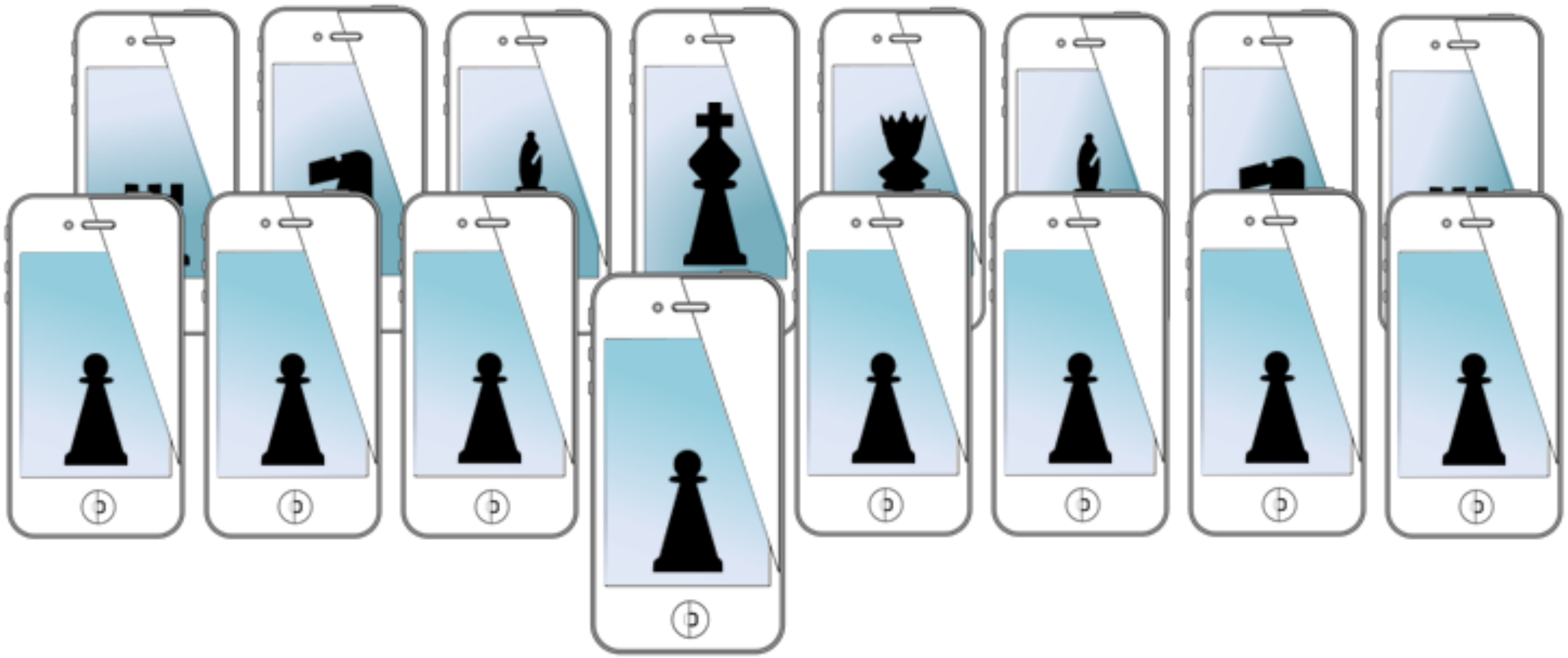
reserved | input array | magnitude out | phase out

highly optimized!! even using tricks we have not discussed

for next time…

- programming the FFT
- filtering
- more frequency analysis with the FFT
    - windowing effects

MOBILE SENSING LEARNING & CONTROL



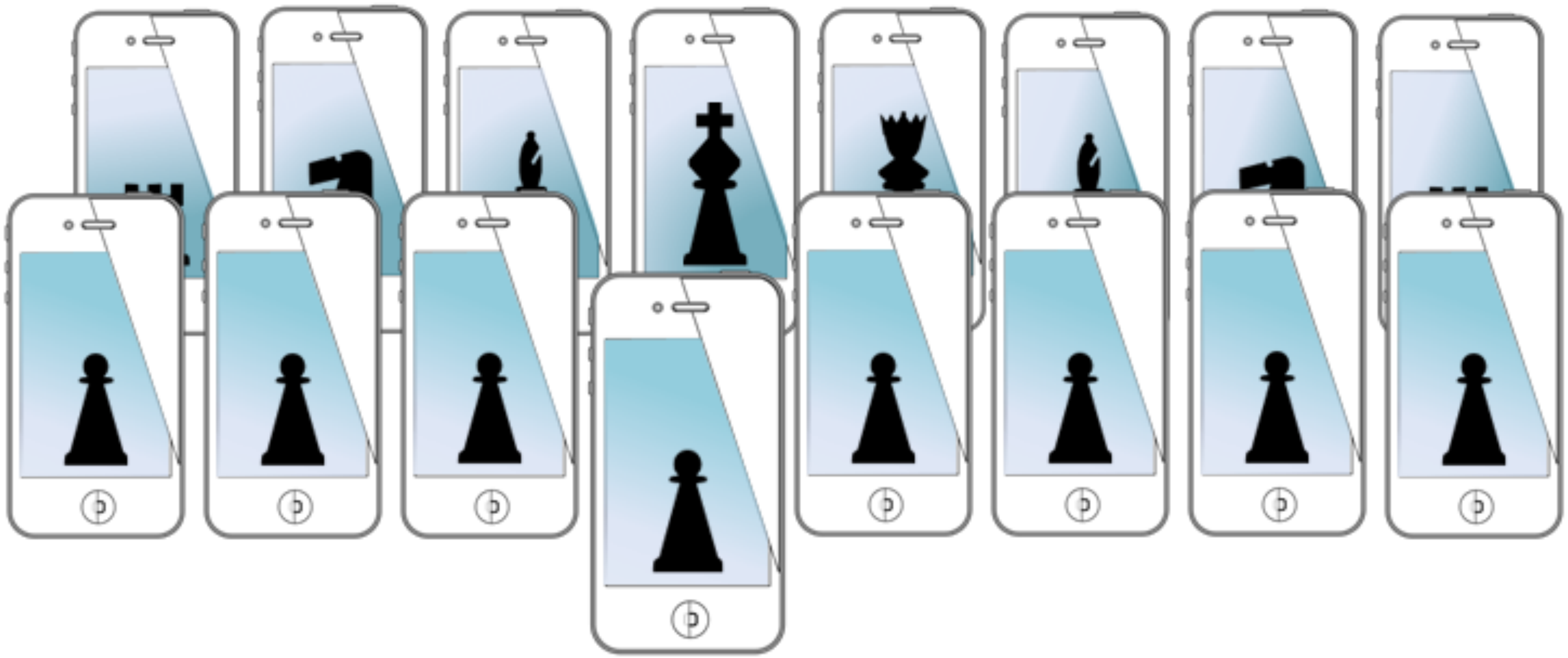
CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture six: audio graphing, sampled data, accelerate, & FFT

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture seven: filtering and windowing

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

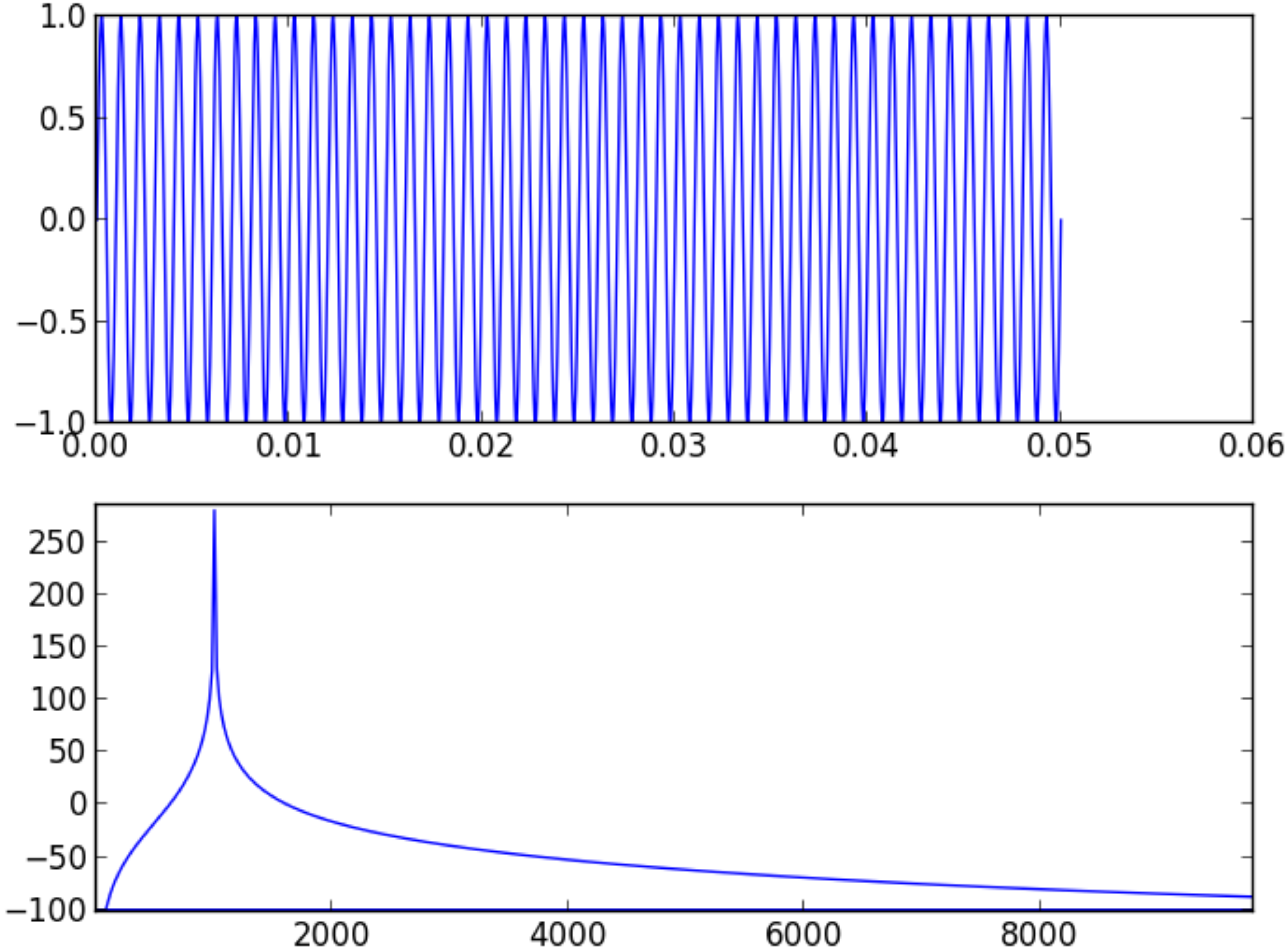
- A1 grades posted
  - feedback from blackboard system
  - team member who turned in A1 has the feedback.
- A2 is due Friday of next week
  - constraints updated for 12Hz accuracy (+-6Hz)

agenda

- using the FFT
- we'll be taking a brief look at filtering
  - an in-depth treatment would take many weeks, we will just do the basics
- how to use a filter (FIR)
  - how to create the most basic of filters
  - two methods of applying the filter to a signal
- We will NOT cover filter design
  - For the curious, the only filter design we'll touch is windowed FIR

some fft examples

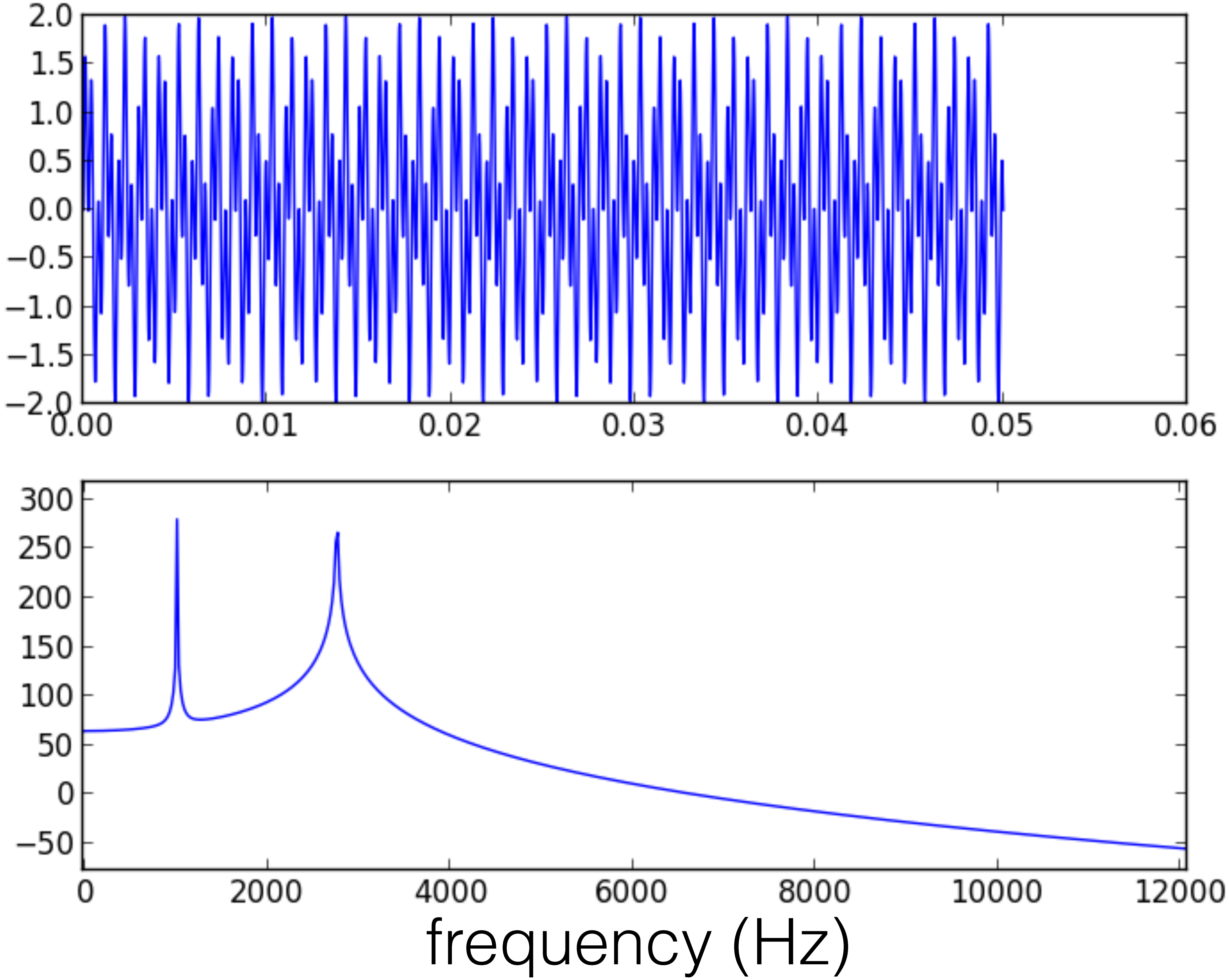
1kHz sine wave



frequency (Hz)

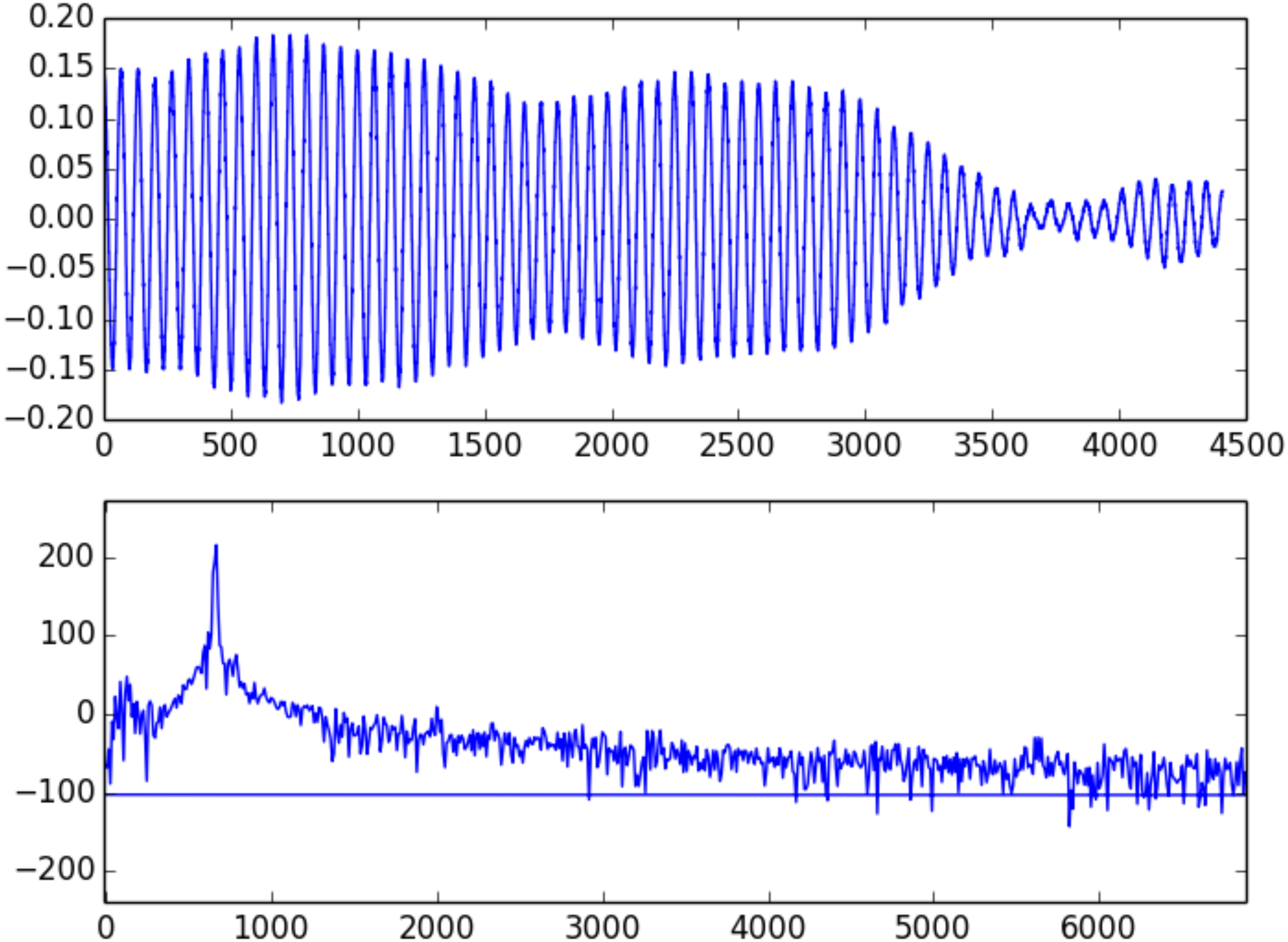
some fft examples

1kHz sine wave +2.7kHz sine wave



some fft examples

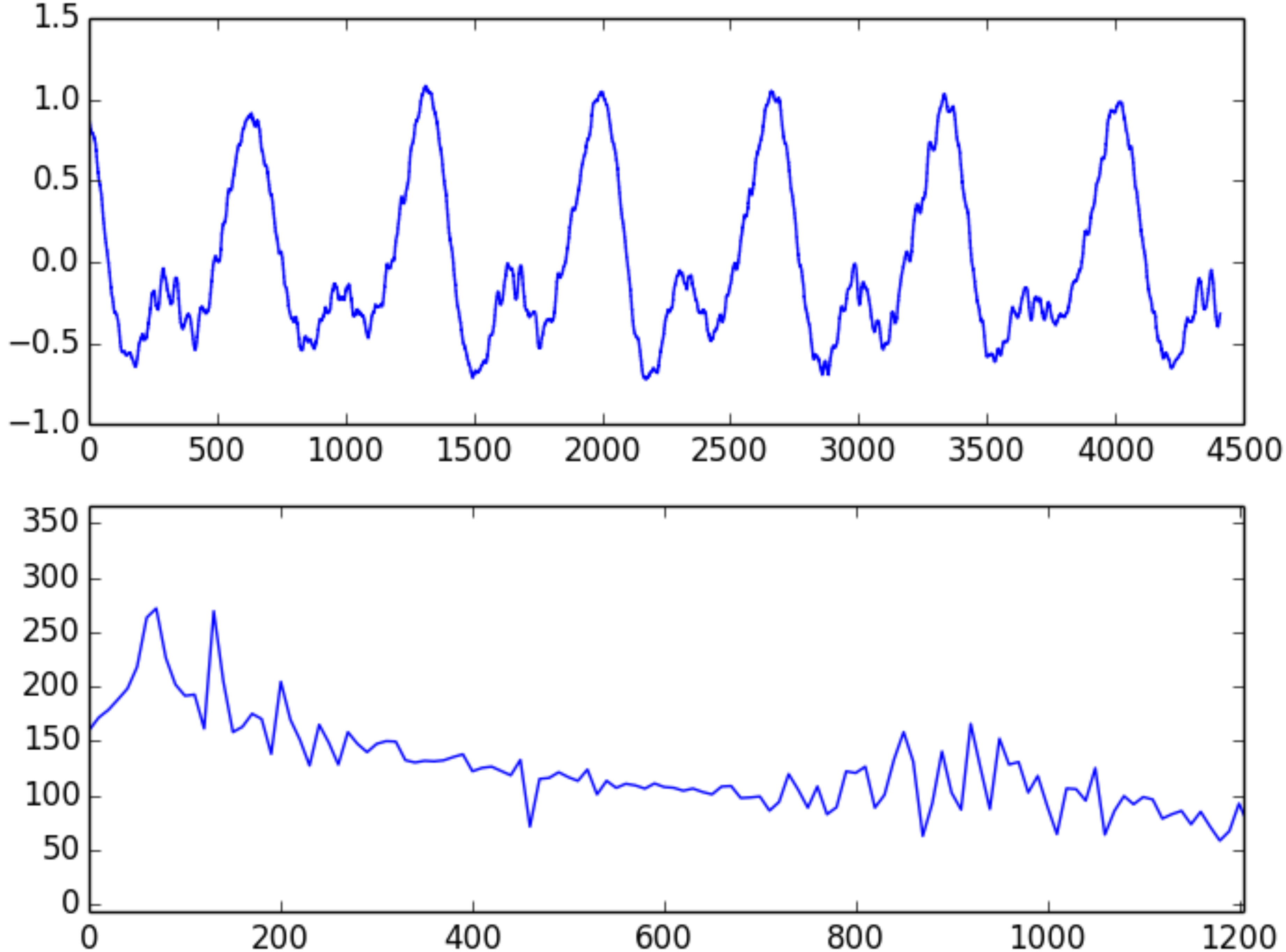
a person whistling



frequency (Hz)

some fft examples

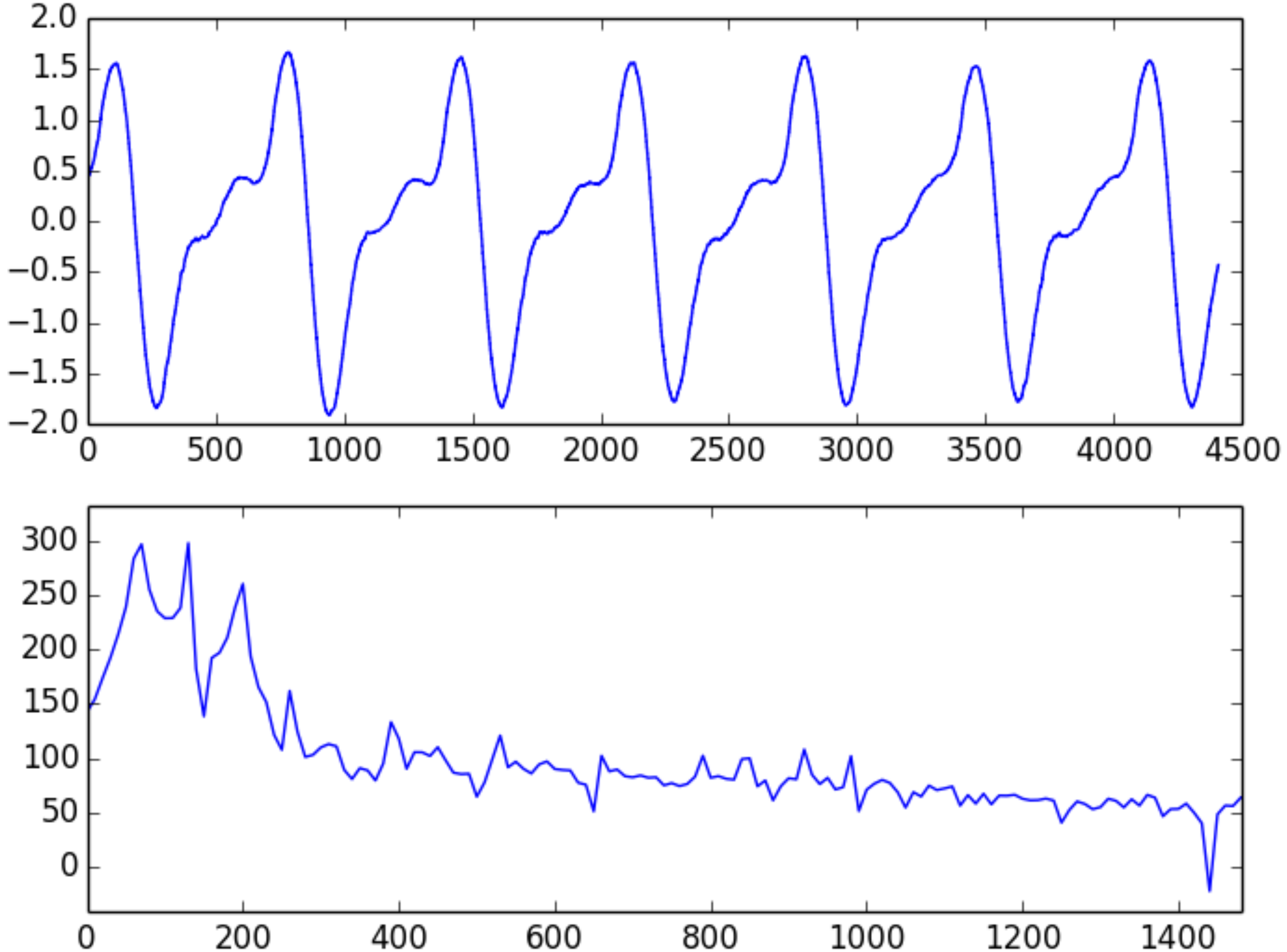
humming



frequency (Hz)

some fft examples

humming again



frequency (Hz)

fft demo!!

```
#import "SMUFFTHelper.h"

float           *fftMagnitudeBuffer;
float           *fftPhaseBuffer;
SMUFFTHelper    *fftHelper;

//setup the fft
fftHelper = new SMUFFTHelper(kBufferLength,kBufferLength,WindowTypeRect);
fftMagnitudeBuffer = (float *)calloc(kBufferLength/2,sizeof(float));
fftPhaseBuffer     = (float *)calloc(kBufferLength/2,sizeof(float));
```

fft size — window size — window type

```
enum WindowType {
    WindowTypeHann,
    WindowTypeHamming,
    WindowTypeRect,
    WindowTypeBlackman,
    };
```

```
free(fftMagnitudeBuffer);
free(fftPhaseBuffer);
delete fftHelper;
```

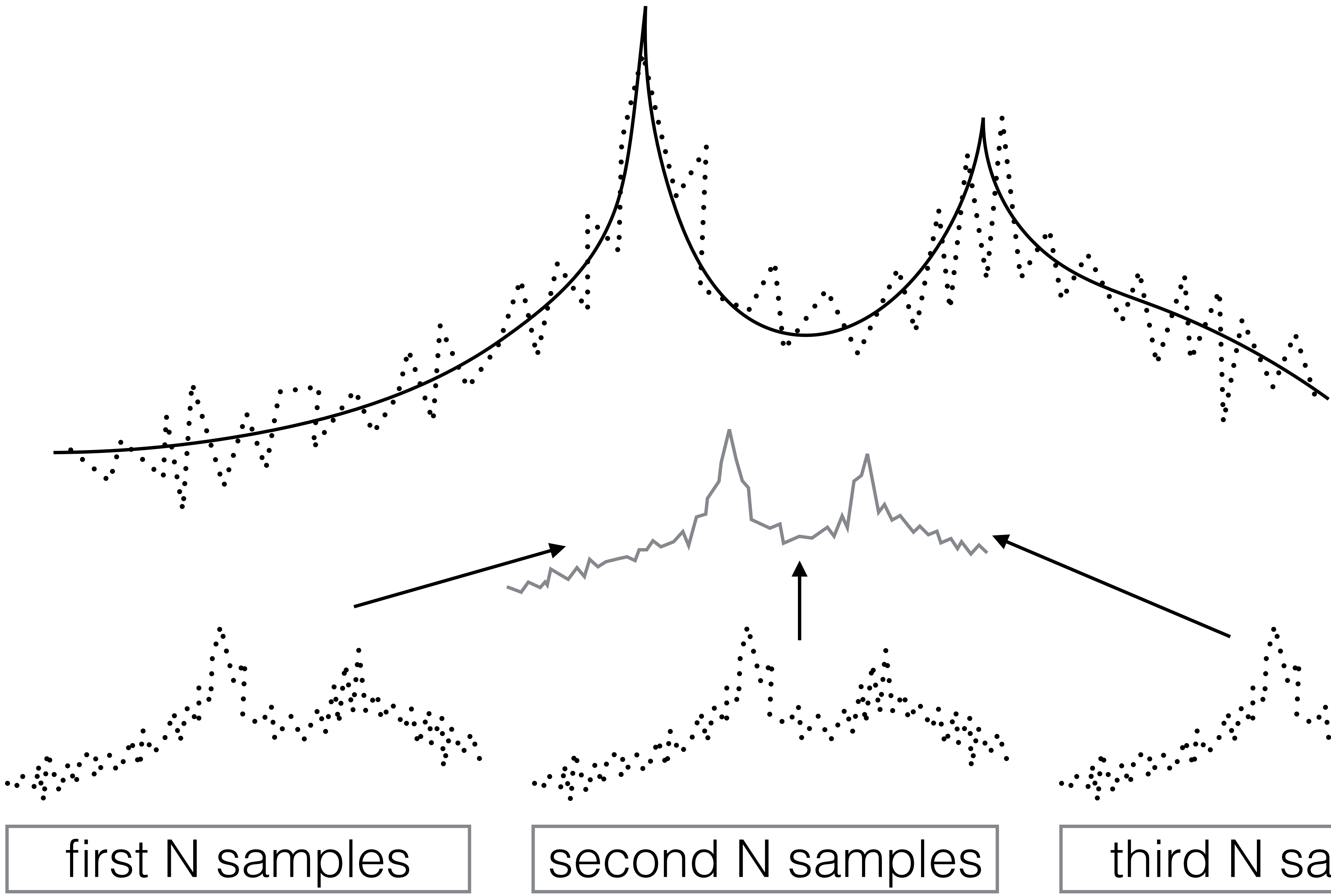
tear down in dealloc

```
fftHelper->forward(0,inputAudioDataBuffer, fftMagnitudeBuffer, fftPhaseBuffer);
```

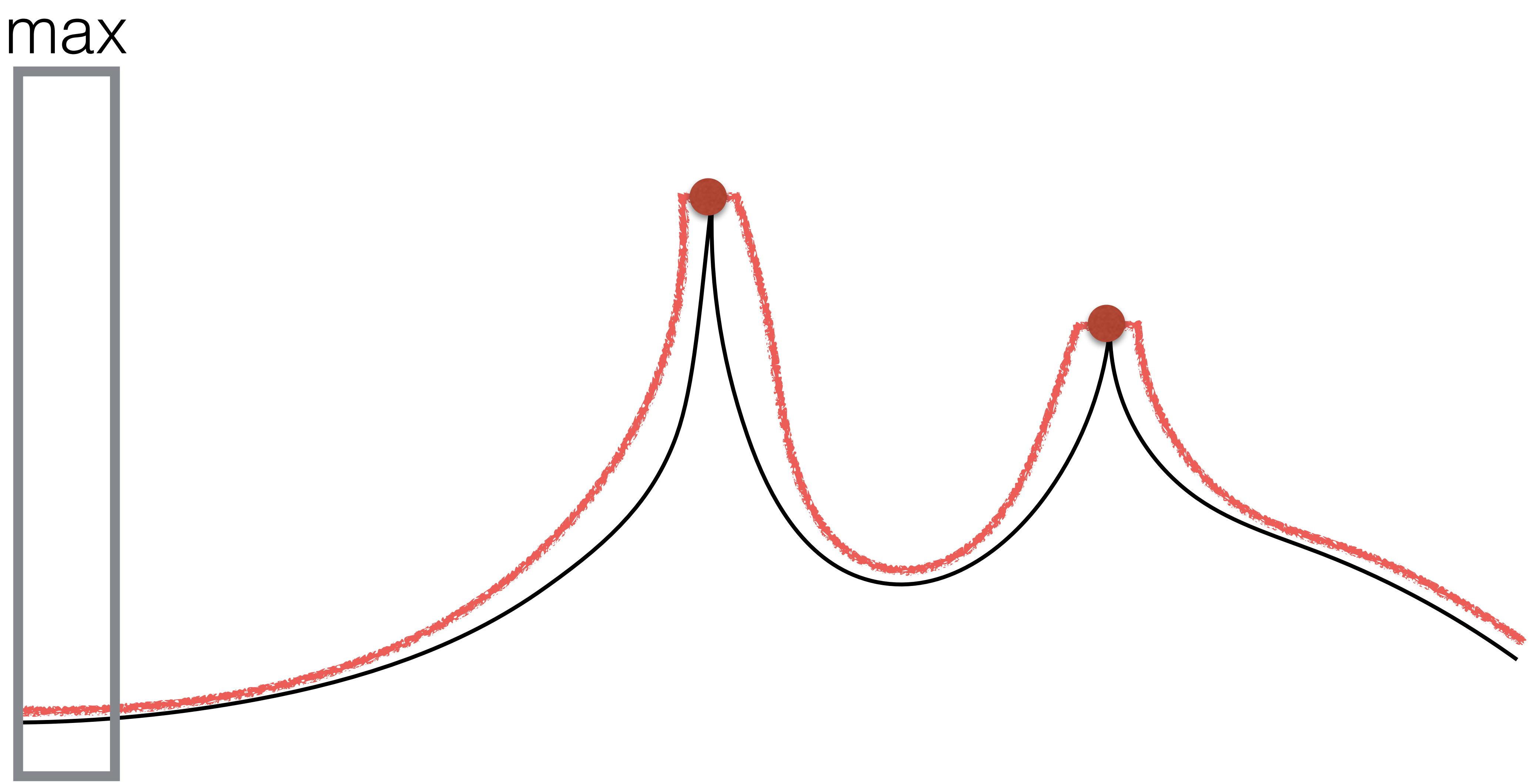
reserved — input array — magnitude out — phase out

noise in the FFT

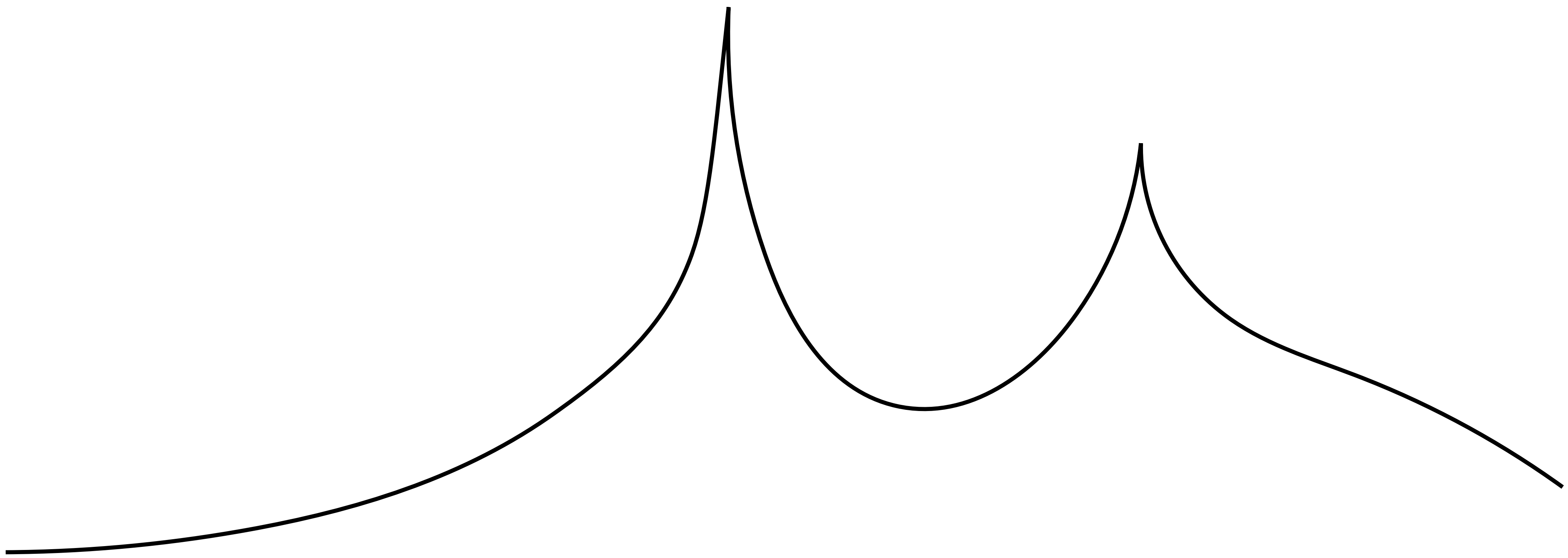
- variance around actual magnitude unavoidable



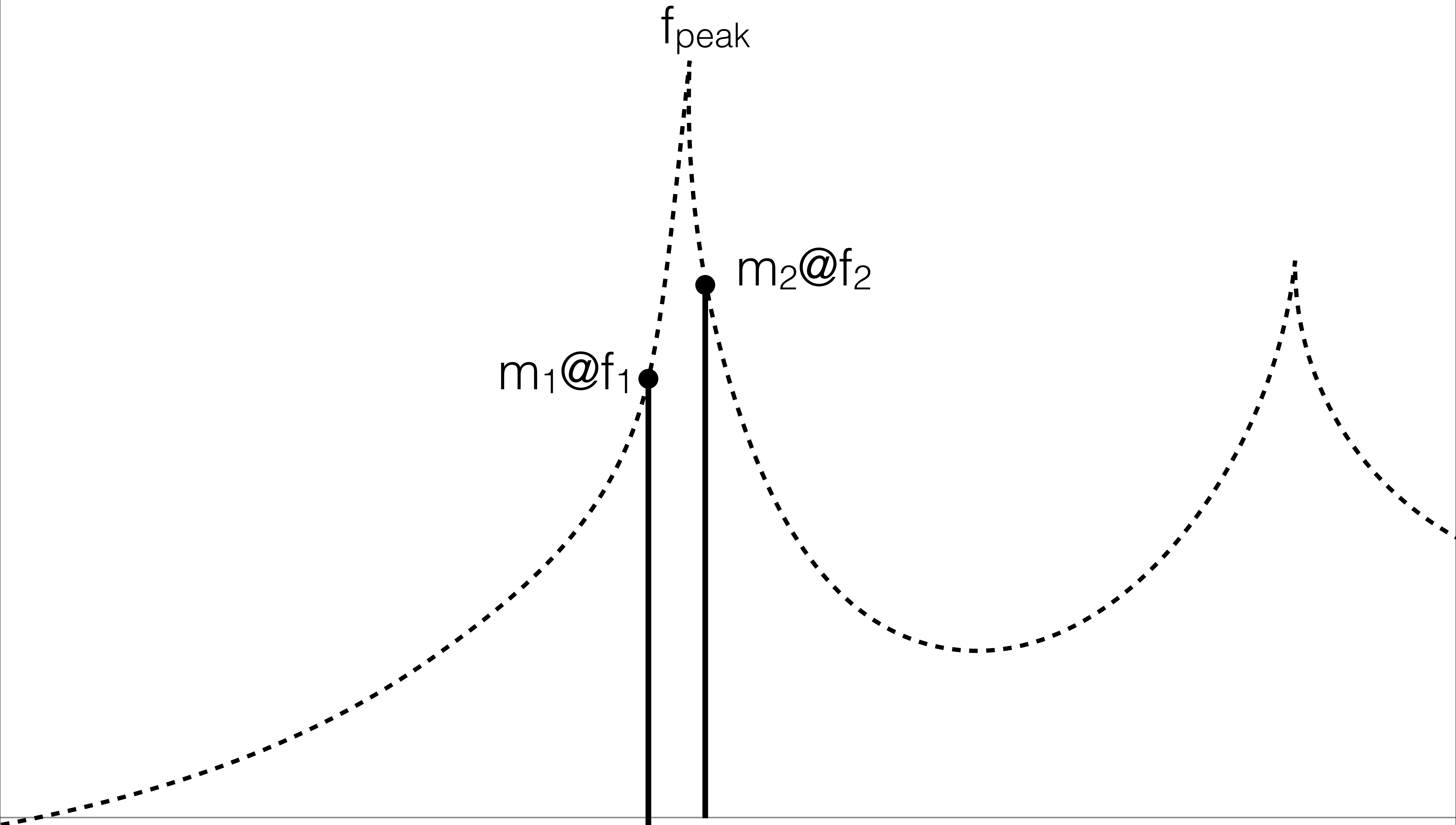
local peak finding



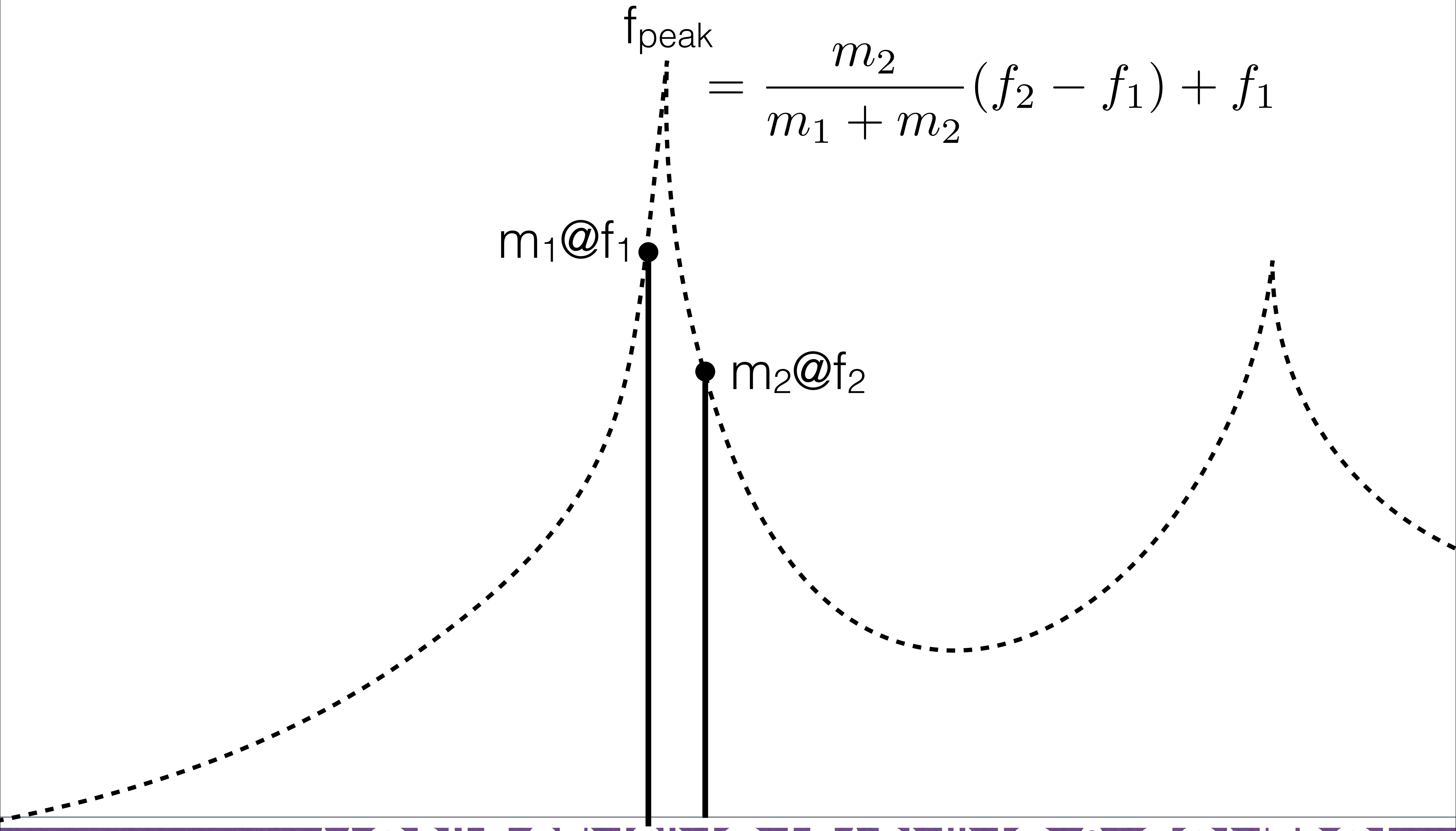
peak interpolation



peak interpolation

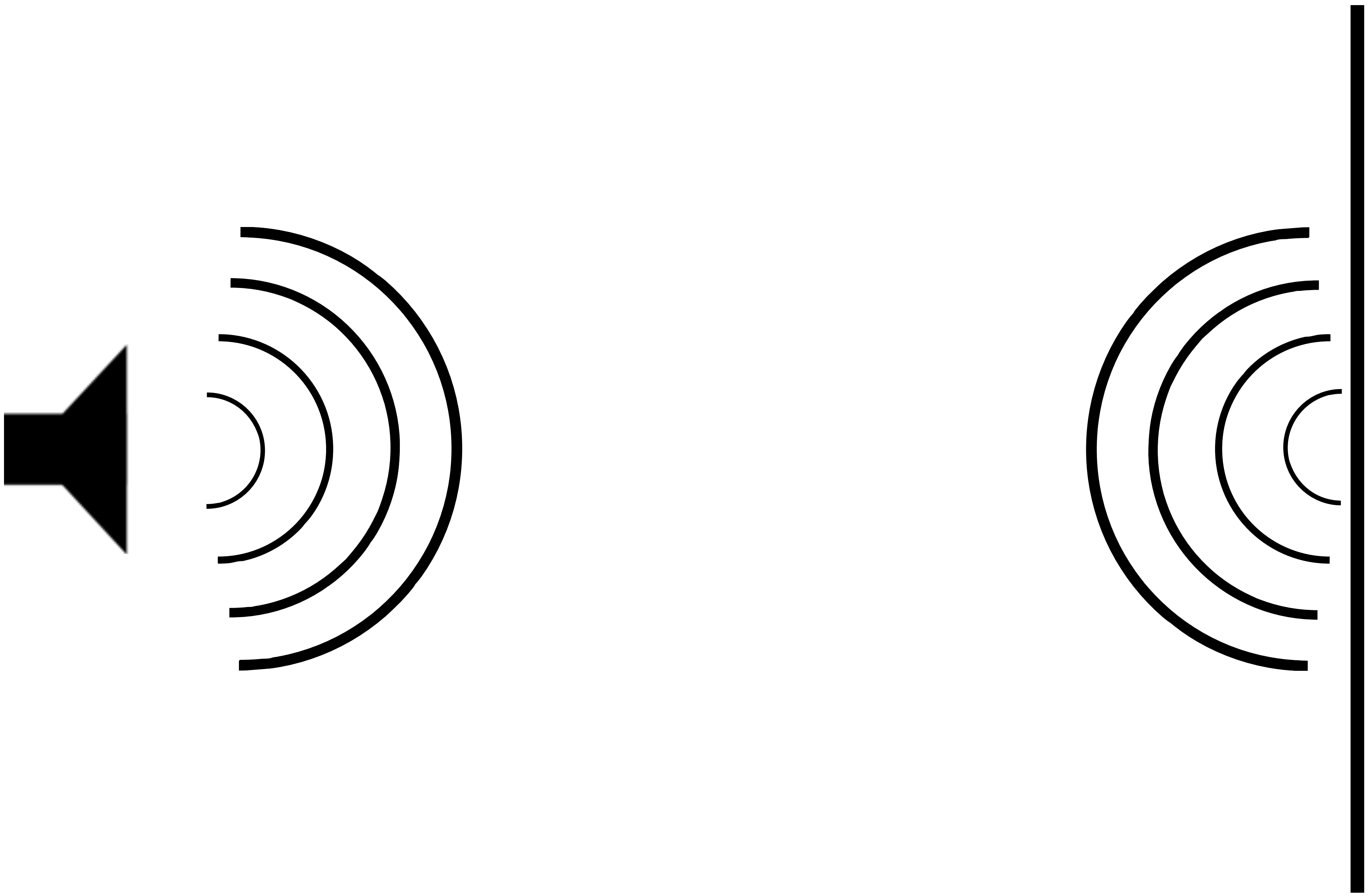


peak interpolation

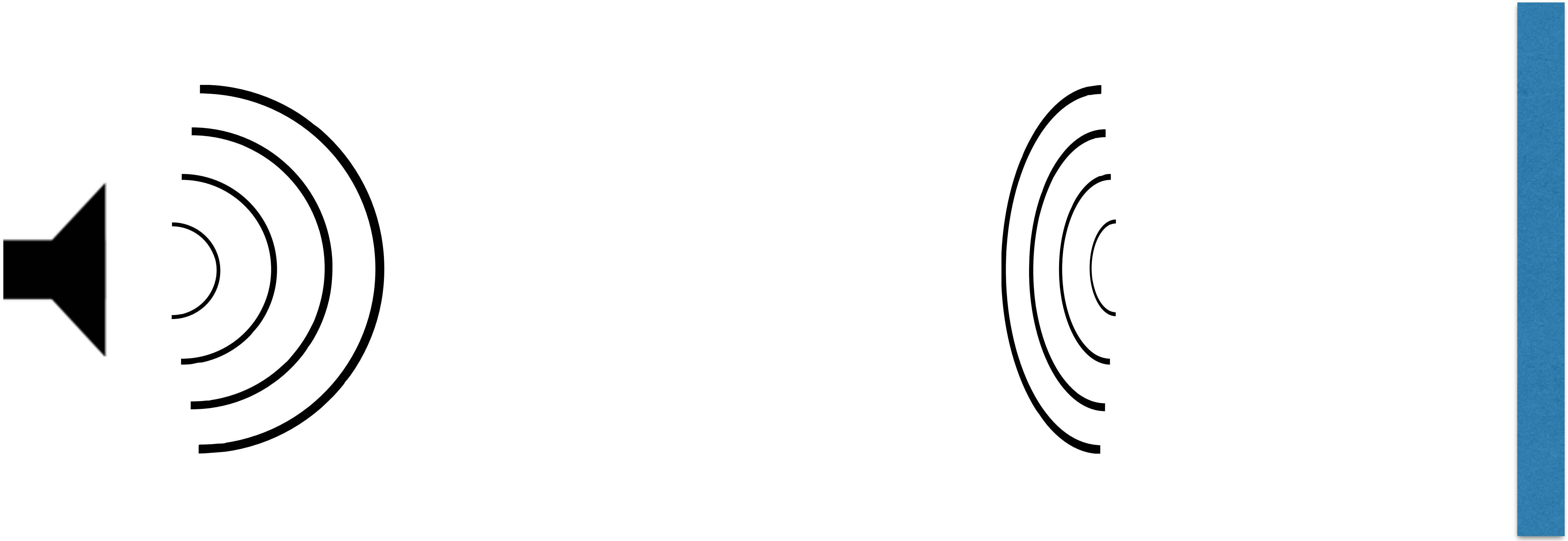


$$f_{peak} = \frac{m_2}{m_1 + m_2}(f_2 - f_1) + f_1$$

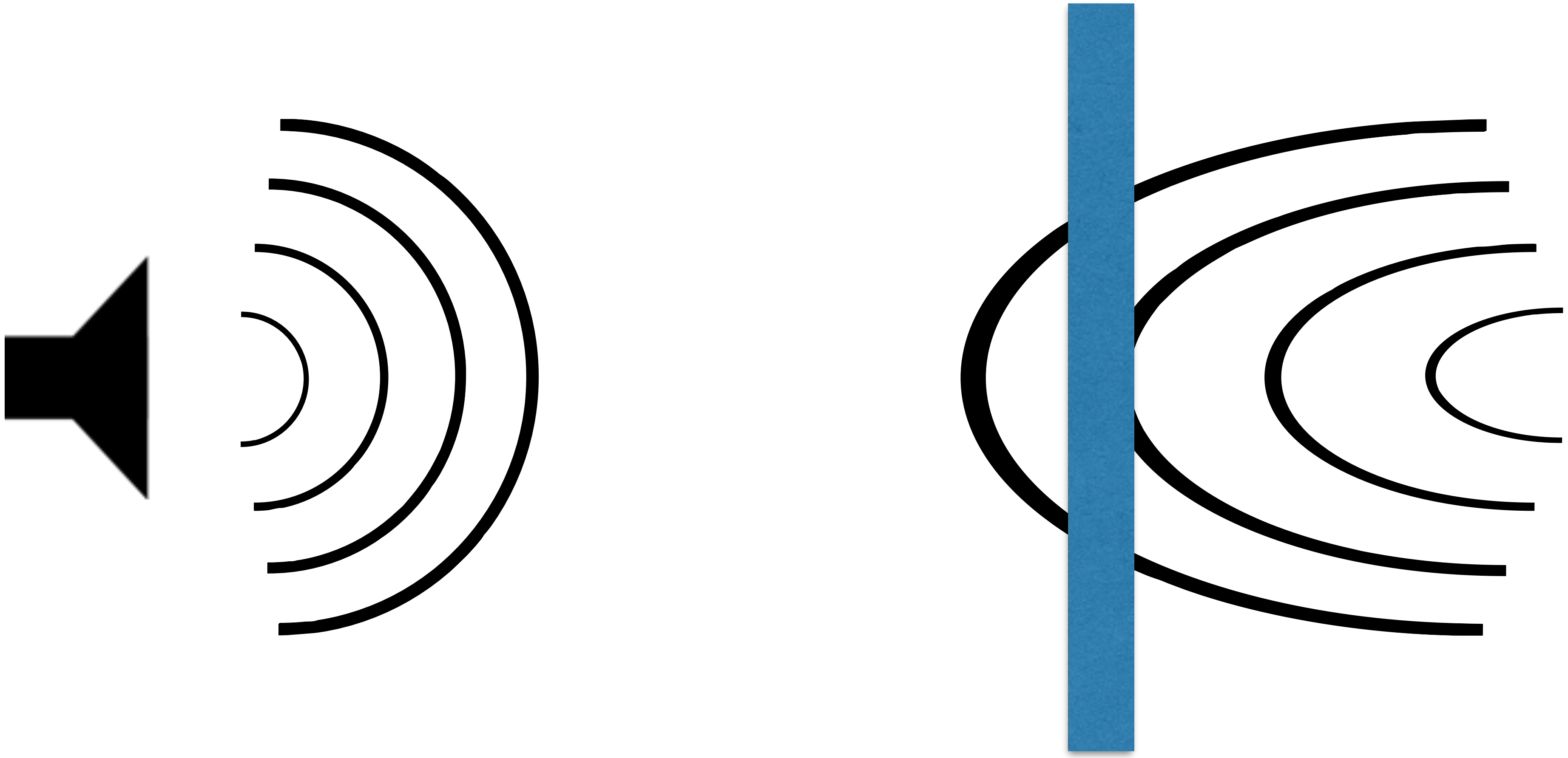
the doppler effect



the doppler effect



the doppler effect



the doppler effect

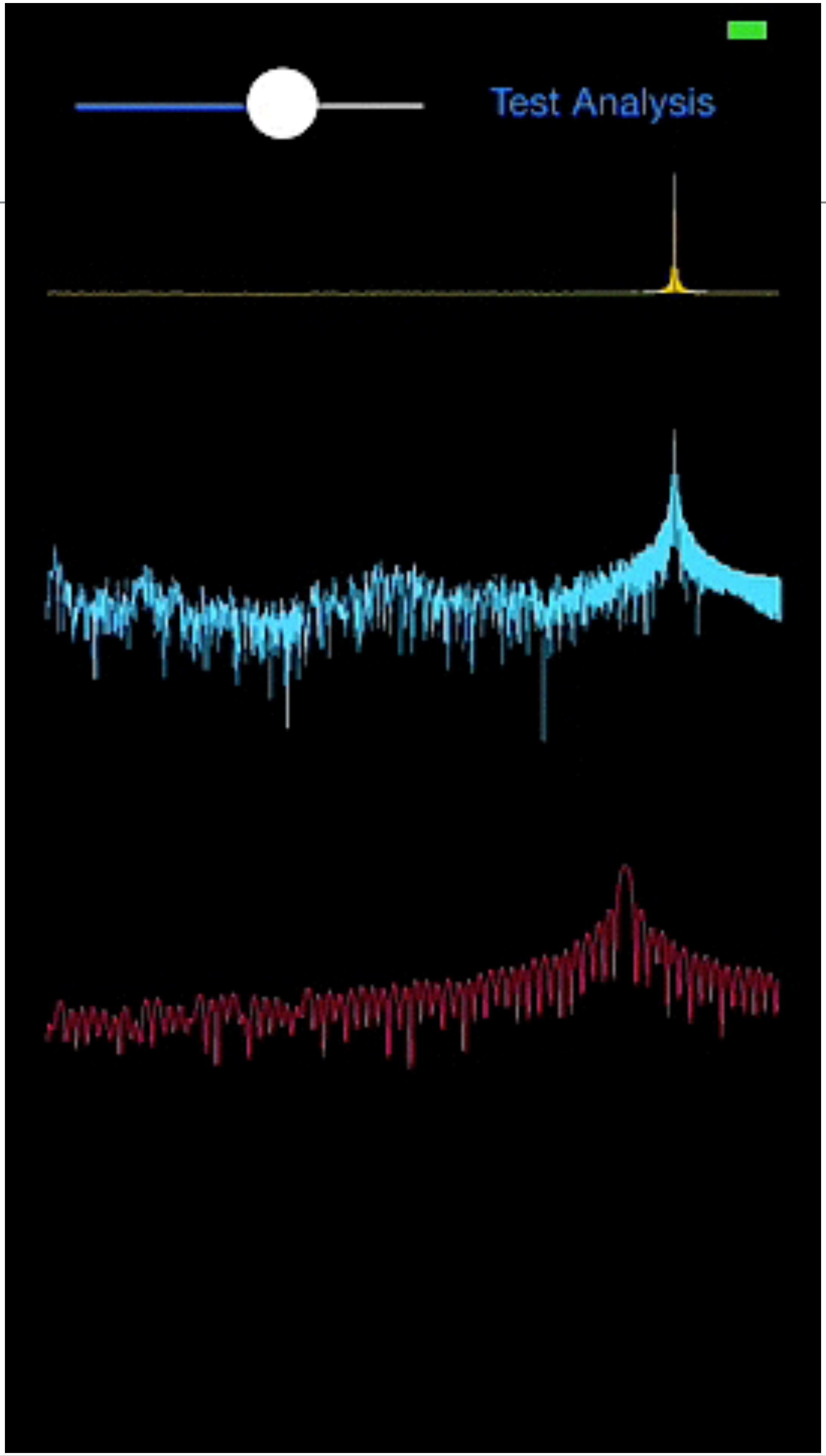
velocity of object

change in frequency

$$\Delta f = \frac{V_{object}}{c} f_0$$

speed of sound

frequency of source



linear

db

db zoomed

filters!

- we will cover what we can…

signals and systems

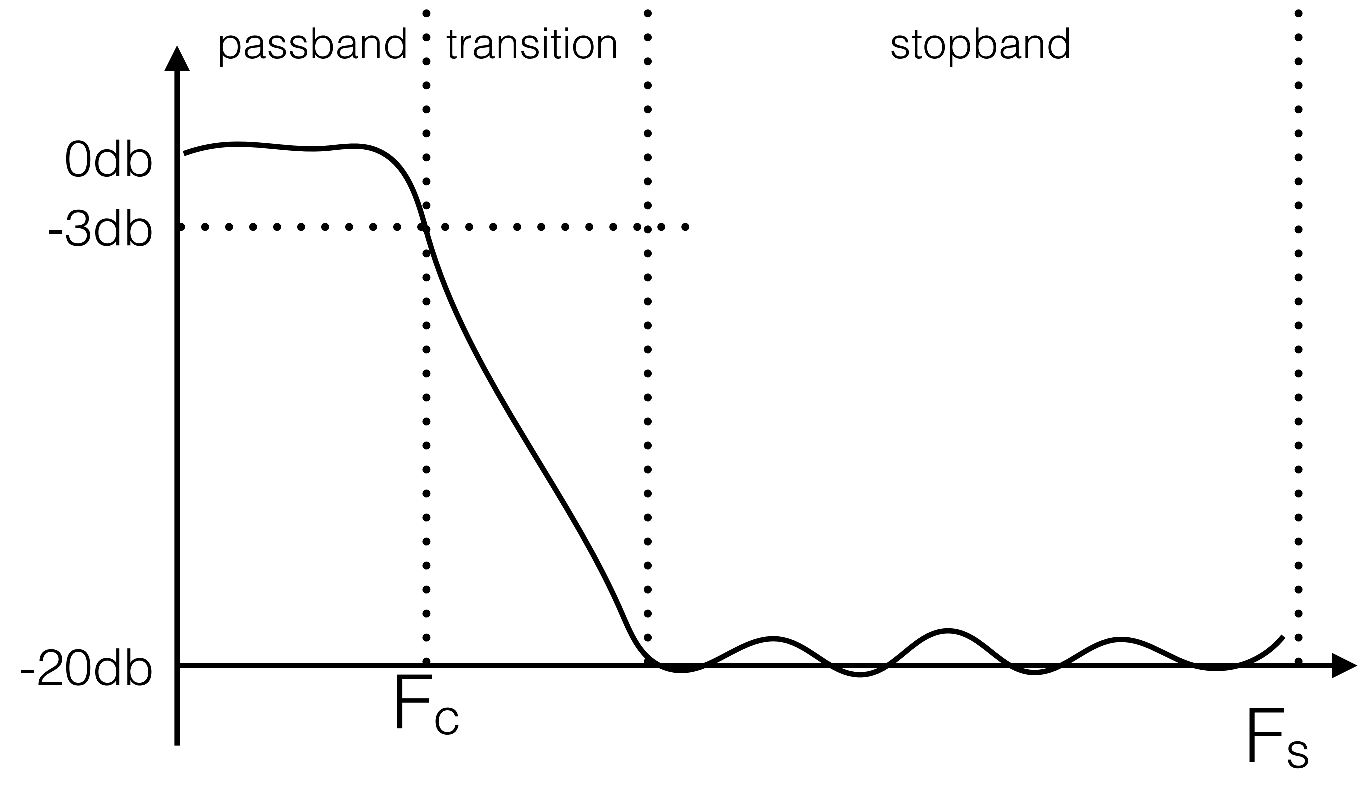
- signals are collections of sampled data (arrays)
  - such as audio, accelerometer, etc.
  - can also be 2D, like images
- systems are objects which manipulate signals
  - characterized by their “input/output” relationships
  - we say “x[n] is passed through H, resulting in y[n]”

filters

- filters are systems which manipulate frequencies
  - certain frequencies to pass through, but not others
  - "lowpass" filter allows low frequencies to pass through
  - "highpass" filter likewise allows high frequencies through
- keep in mind: no filter is perfect!!
  - no filter will pass everything you want while stopping everything you don't
  - everything is a balance between different parameters you can control
- we won't study how to design filters
  - we will study properties of filters and how to use them
  - so we need to know what filters can and cannot do

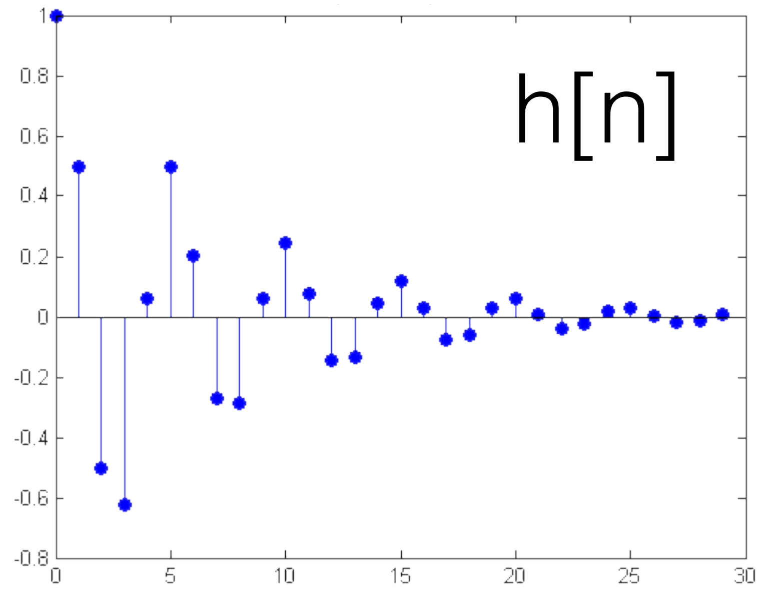
filters in frequency

- filters can be characterized a few different ways
  - let's start by looking at their properties in the frequency domain
- filters have the following frequency-domain attributes:
  - passband gain
  - passband bandwidth
  - stopband attenuation
  - transition bandwidth



filters in time

- filter are also signals (time series)
  - the series is called the “impulse response” of the filter
  - the frequency-domain plots are just Fourier transforms of the impulse response (magnitude)
- the time-domain property we care about is length
  - everything else is best left to a filter design course



so how to design a filter?

- scipy.signal in python (try to use remez)
- decent tutorial:
  - http://mpastell.com/2010/01/18/fir-with-scipy/
- matlab
  - fdatool

```
from pylab import *
import scipy.signal as signal
n = 61
a = signal.firwin(n, cutoff = 0.3,
                  window = "hamming")
```

- lots of other places!

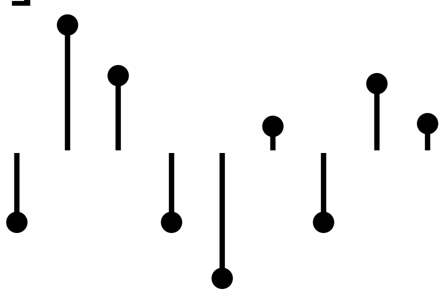
filtering by convolution

- we apply a filter using **convolution**
  - convolution allows us to combine frequency properties of two signals without taking an FFT
- basic principle:
  - convolution in time is multiplication in frequency
  - so the filter's frequency response will be multiplied by the frequency response of the signal

$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

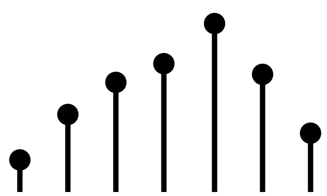
convolution

x[i]

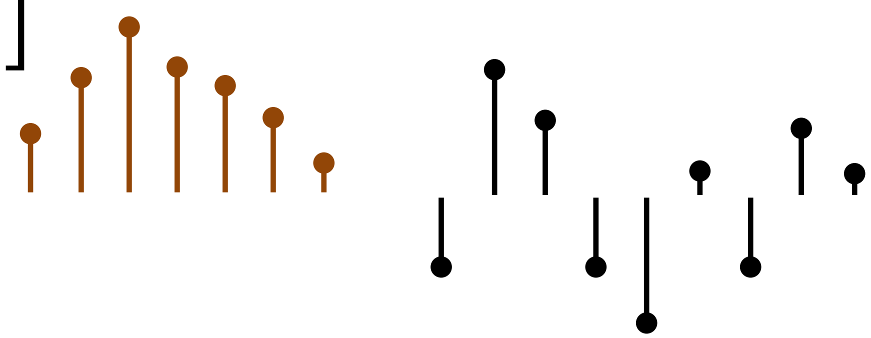


$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

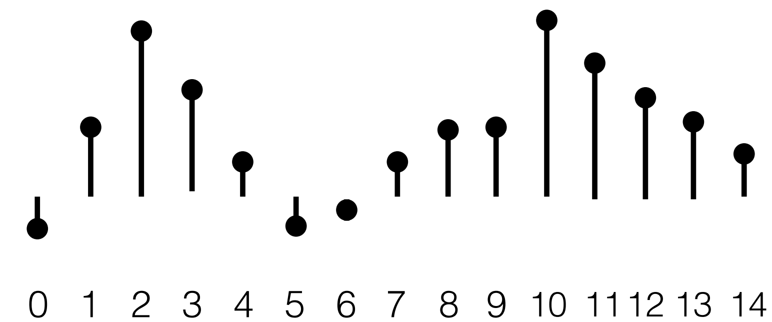
h[i]



h[-i]



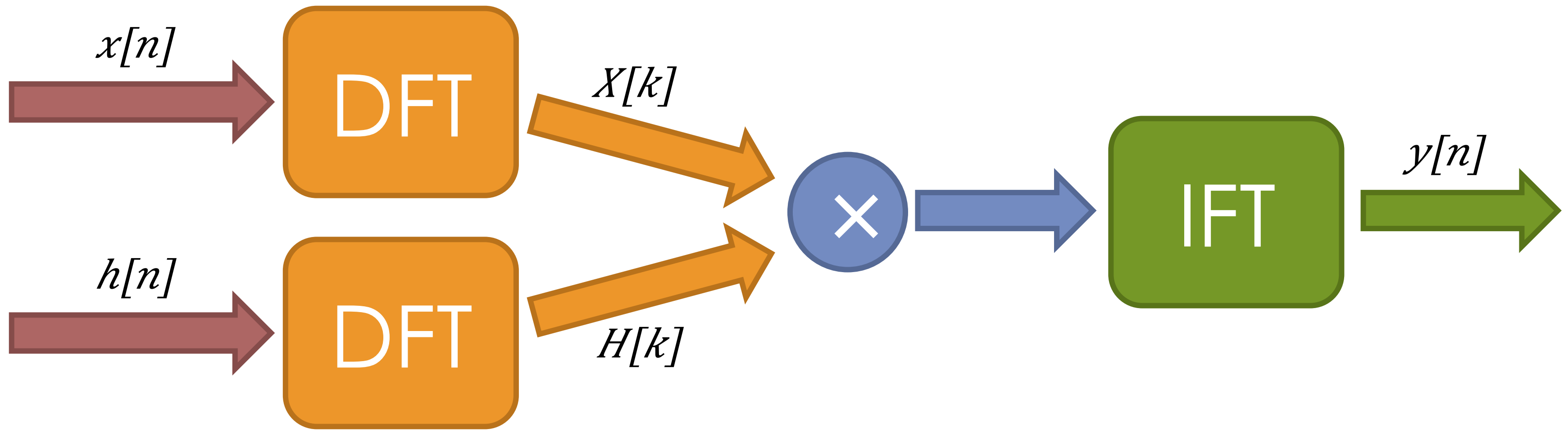
y[n]



length N M N+M-1

convolution efficiency

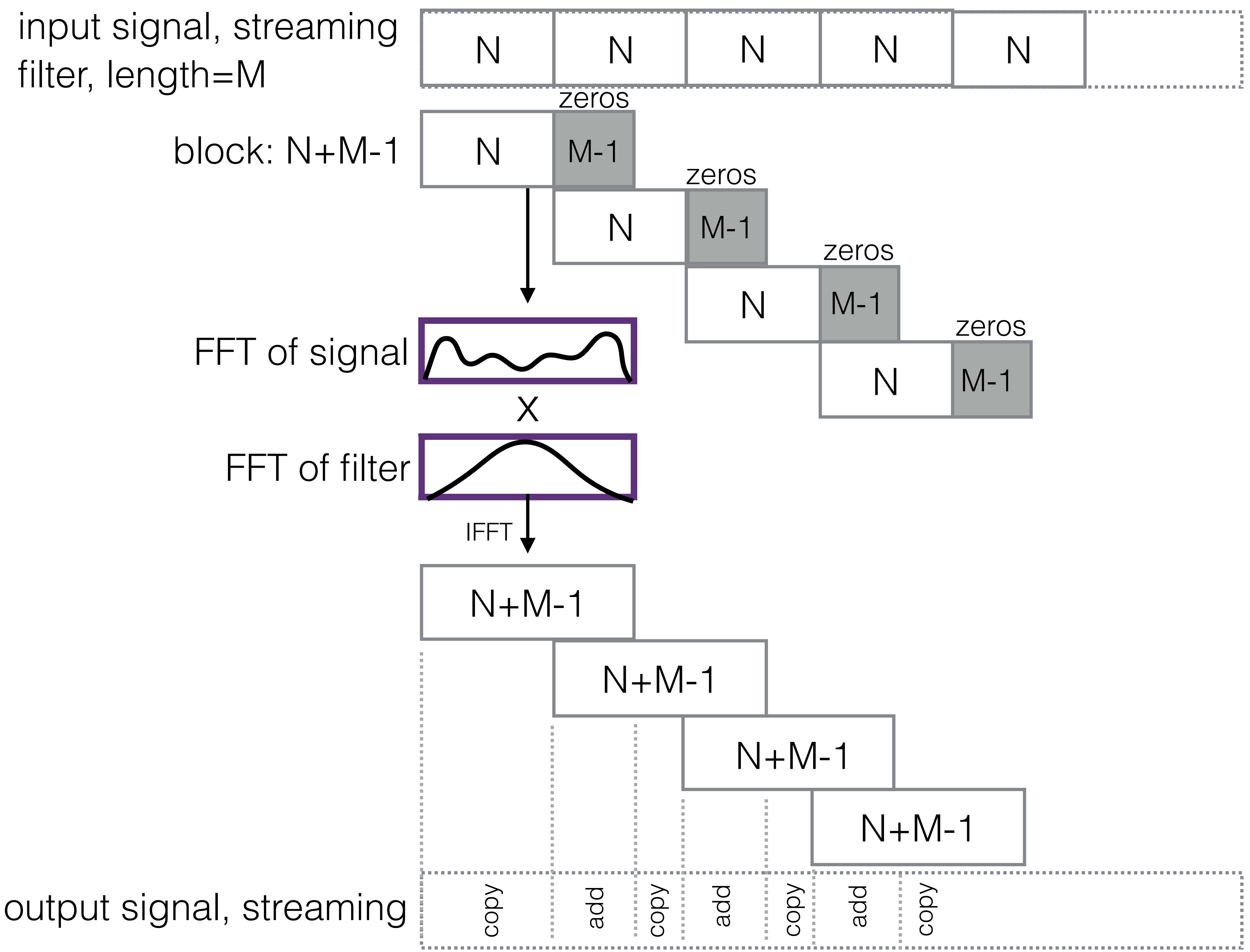
- algorithmic complexity
  - convolution is not particularly efficient, O(NxM)
- to convolve faster, use that Fourier property:
  - “convolution in time is multiplication in frequency”
- why not just multiply to begin with!



its circular

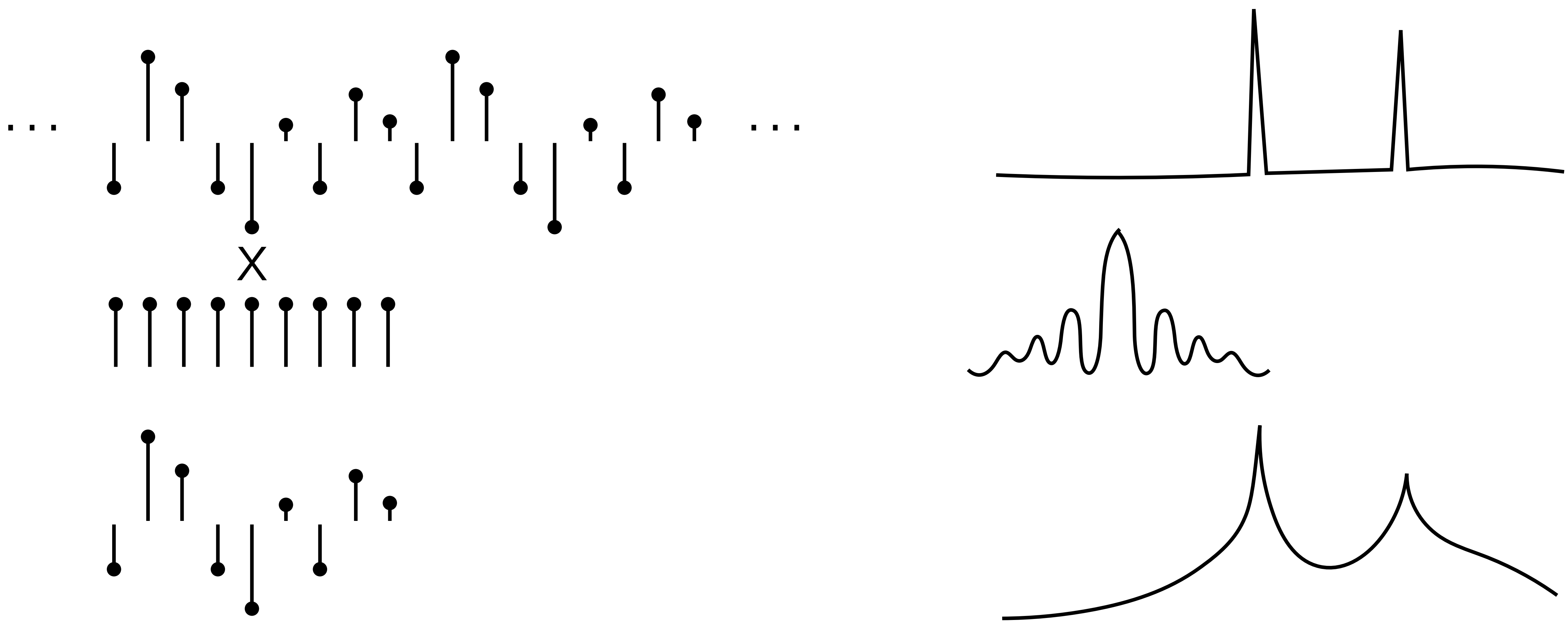
- just using N point FFT performs “Circular Convolution”
  - which is not linear convolution
  - causes the tail end of the convolution to “wrap around” to the beginning
    - FFT assumes the function is periodic (we did not talk about this)
- be aware of circularity when filtering your signal with the FFT
  - zero-padding can solve this for you!
  - zero-pad both signals to a length that will contain the entire convolution, N+M-1
  - for streaming, you must use overlap-and-add!
    - http://en.wikipedia.org/wiki/Overlap–add_method

overlap and add

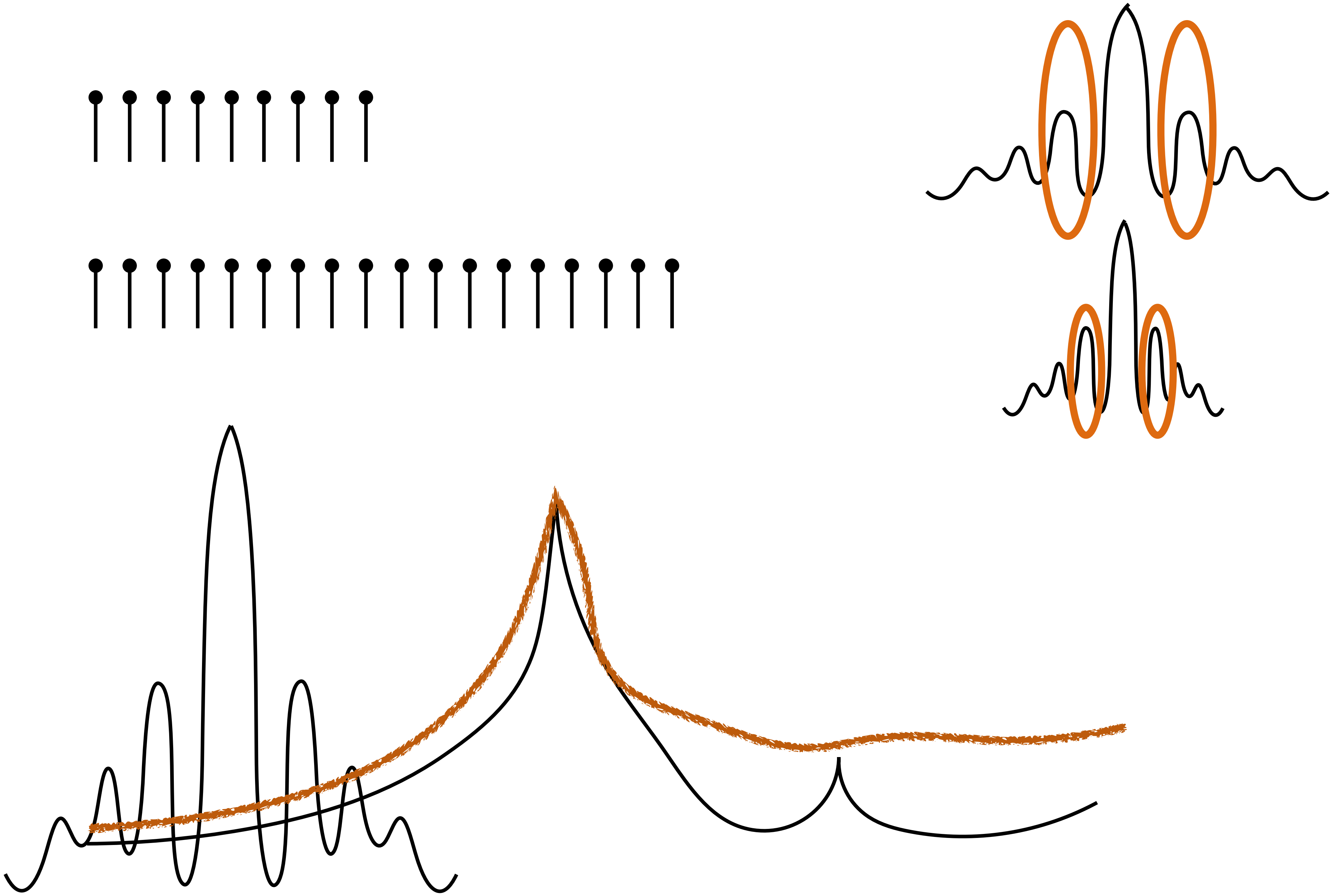


windowing: spectral BW widening

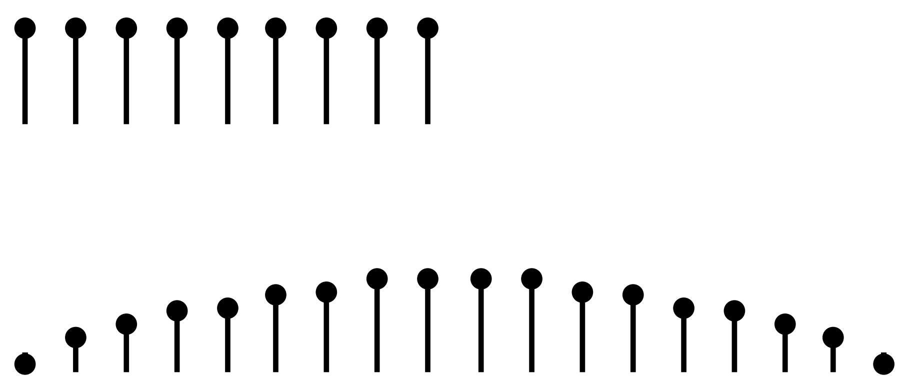
- multiplication in time is convolution in frequency
- a window is something we multiply in time with our signal
- windowing is unavoidable
  - why? we cannot take an infinite FFT…

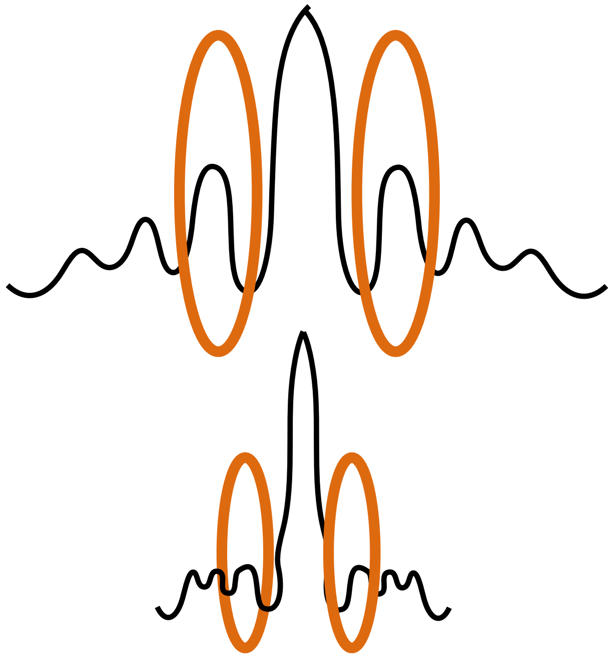


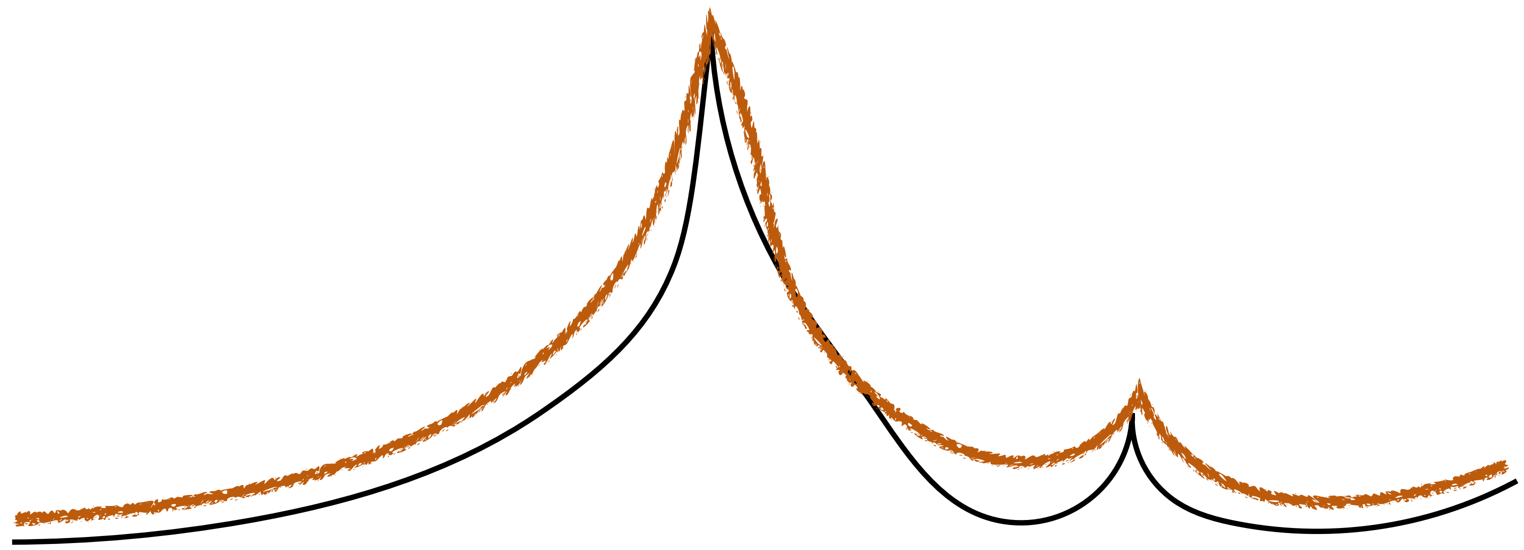
windowing: spectral leakage



windowing: spectral leakage







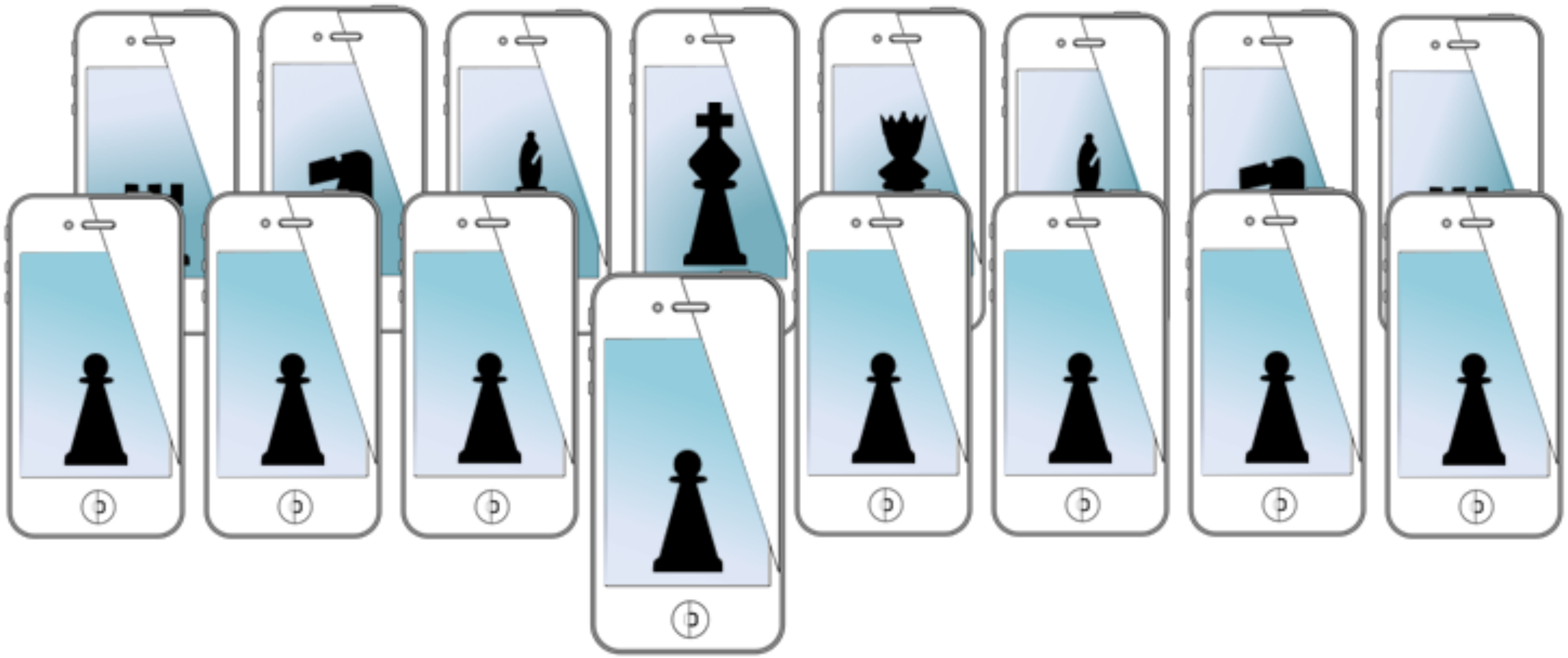
which window to use?

- depends
  - narrowest main lobe: rect
  - good tradeoff: hamming (or Von Hann)
  - optimal tradeoff for a given bandwidth:
    - discrete prolate spheroidal sequence (dpss, Slepian taper)

for next time…

- core motion
  - the M7 co-processor

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture seven: filtering and windowing

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University