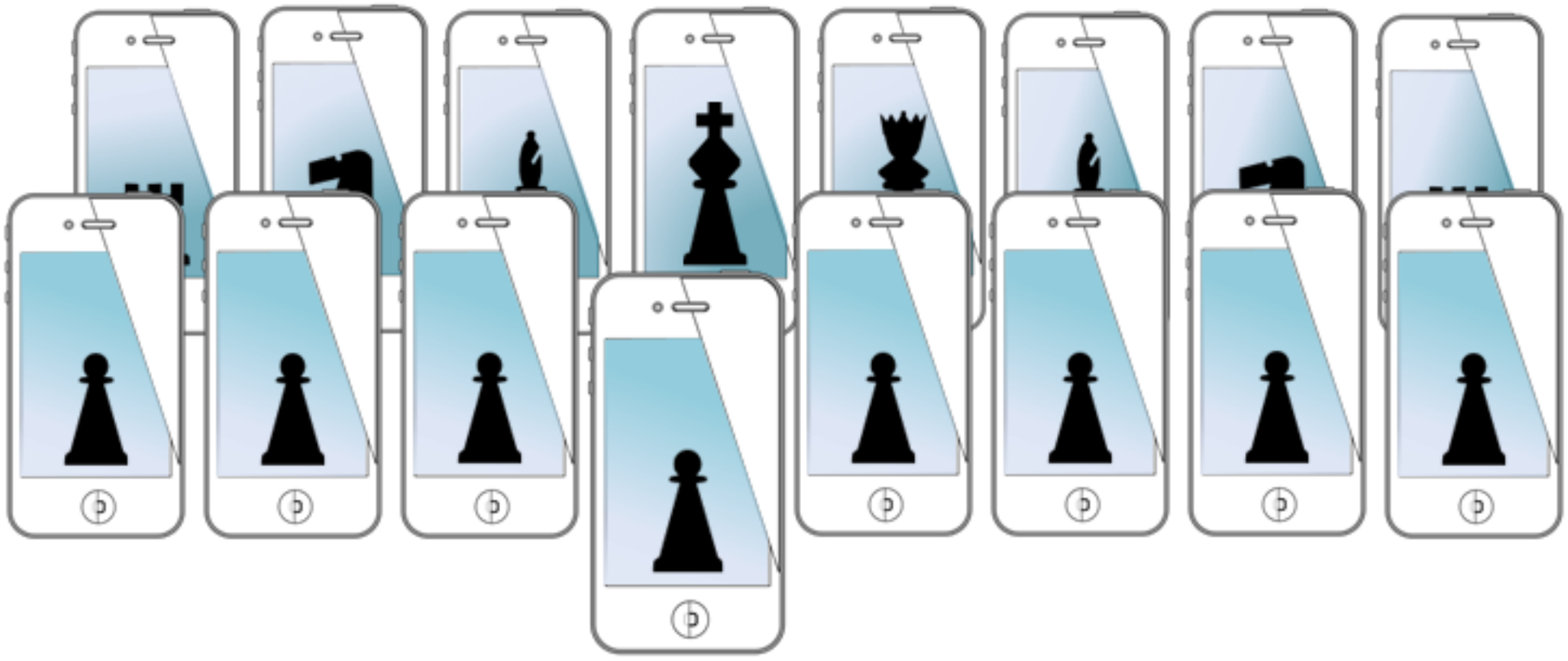
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

audio graphing, sampled data, & accelerate

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

course logistics

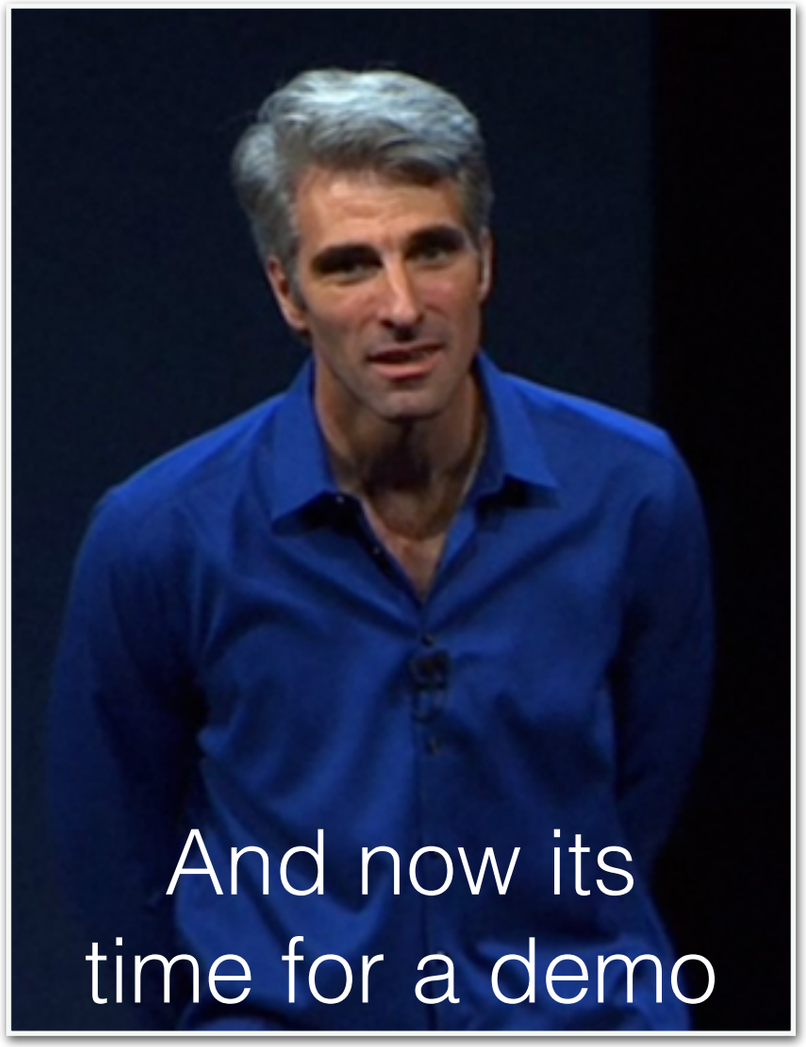
- flipped module on audio next time!

agenda

- dealing with sampled data
- the accelerate framework
  - massive digital signal processing library
- graphing audio fast (well, graphing anything)
  - must use lowest level graphing, Metal

sample from the mic

- recall: data from the microphone on novocaine

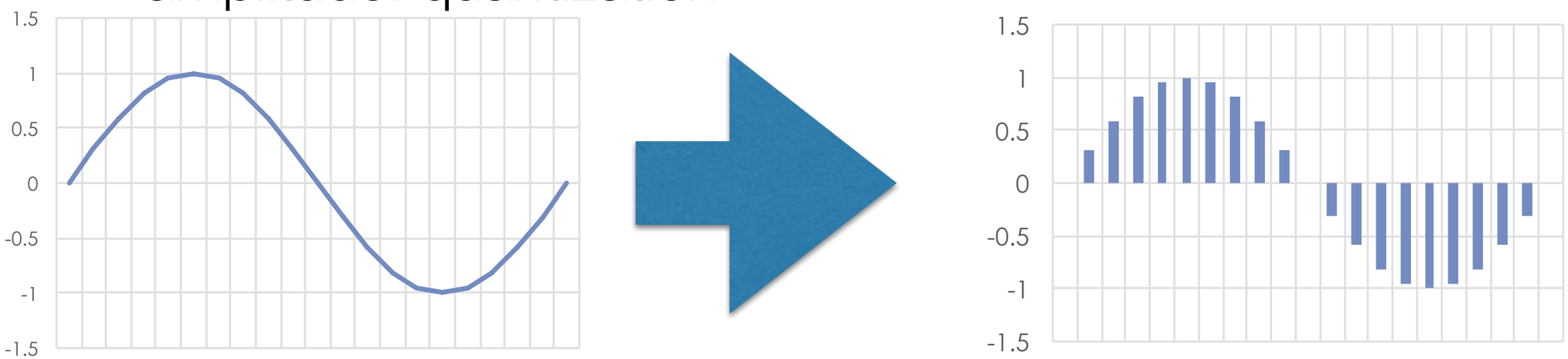


intro to sampled data

- 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

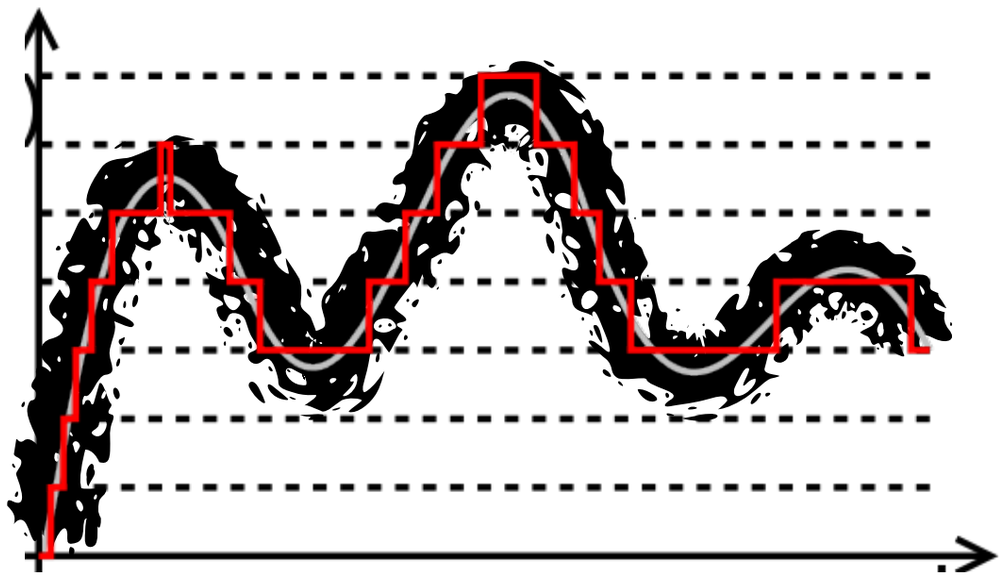
intro to sampled data

- 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



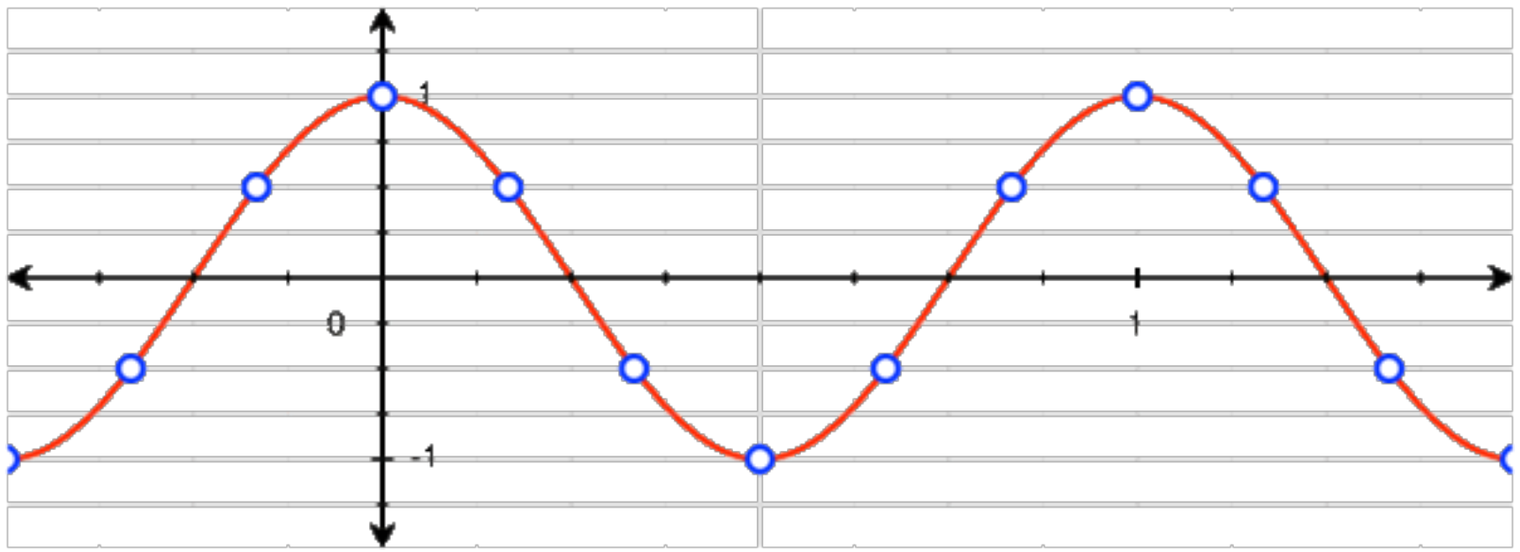
sampled data

- 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!



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?

making a sine wave

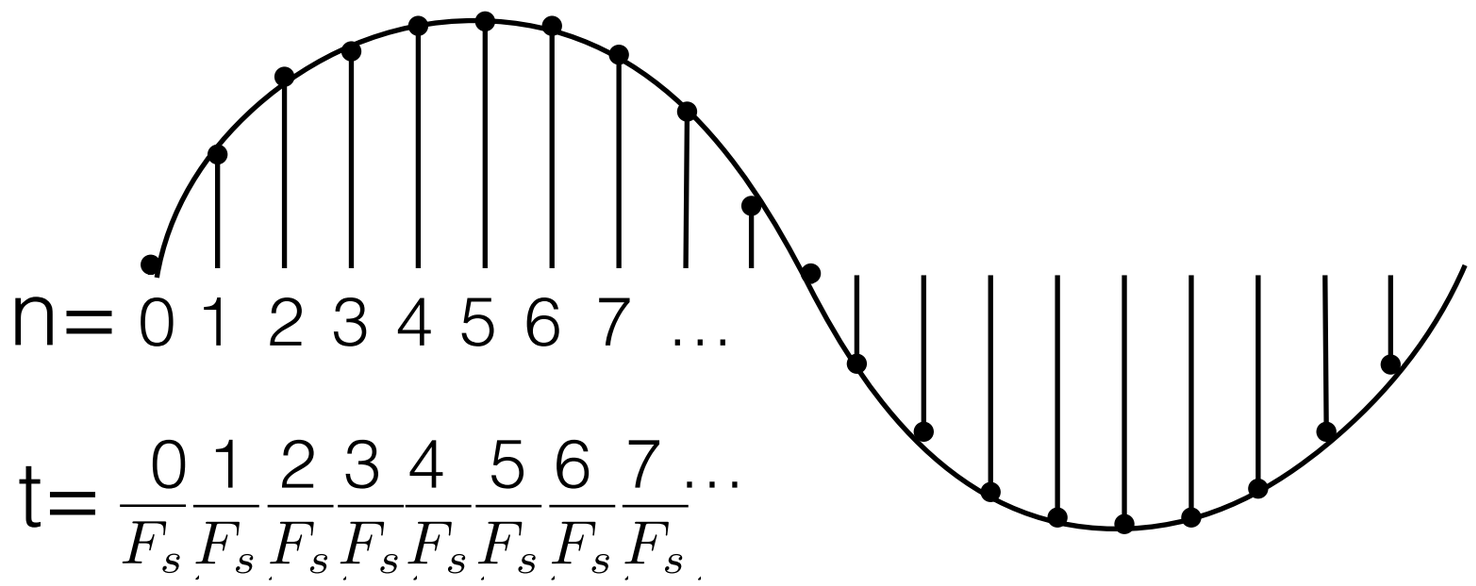
- we want to create a sine wave and play it to the speakers

$$g(t) = \sin(2\pi f t)$$ equation for sine wave

frequency in Hz

time in “seconds”

but we are working digitally, so we have an “index” in an array, not time!



$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$

making a sine wave

$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$ how to program this?

```
for (int n=0; n < numFrames; ++n)
 {
    data[n] = sin(2*M_PI*frequency*n/samplingRate);
 }
```

is this efficient?

```
float phase = 0.0;
double phaseIncrement = 2*M_PI*frequency/samplingRate;
 for (int n=0; n < numFrames; ++n)
 {
    data[n] = sin(phase);
    phase += phaseIncrement;
 }
```

making a sine wave

- bringing it all together $g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$

```
var frequency = 18000.0; //starting frequency
 var phase = 0.0;
var samplingRate = audioManager.samplingRate;

 outputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32))
 {
     var phaseIncrement = 2*Double.pi*frequency/samplingRate
     var i=0;
     var sineWaveRepeatMax = 2*Double.pi;
     while (i < numFrames)
     {
         data![i] = sin(phase);
         i += 1
         phase += phaseIncrement;
         if (phase >= sineWaveRepeatMax) phase -= sineWaveRepeatMax;
     }
 }
```

data: UnsafeMutablePointer<Float>?

play samples to speakers

- demo, play sine wave

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
);
```

examples

what do each of these implement?

```
inputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32)) {
      var volume = userSetMultiplyFromSlider;
      vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels)
      ringBuffer.addNewInterleavedFloatData(data,withNumFrames:numFrames)
  }
```

```
inputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32)) {
      // get the max
      var maxVal = 0.0;
      vDSP_maxv(data, 1, &maxVal, numFrames*numChannels);

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

  }
```

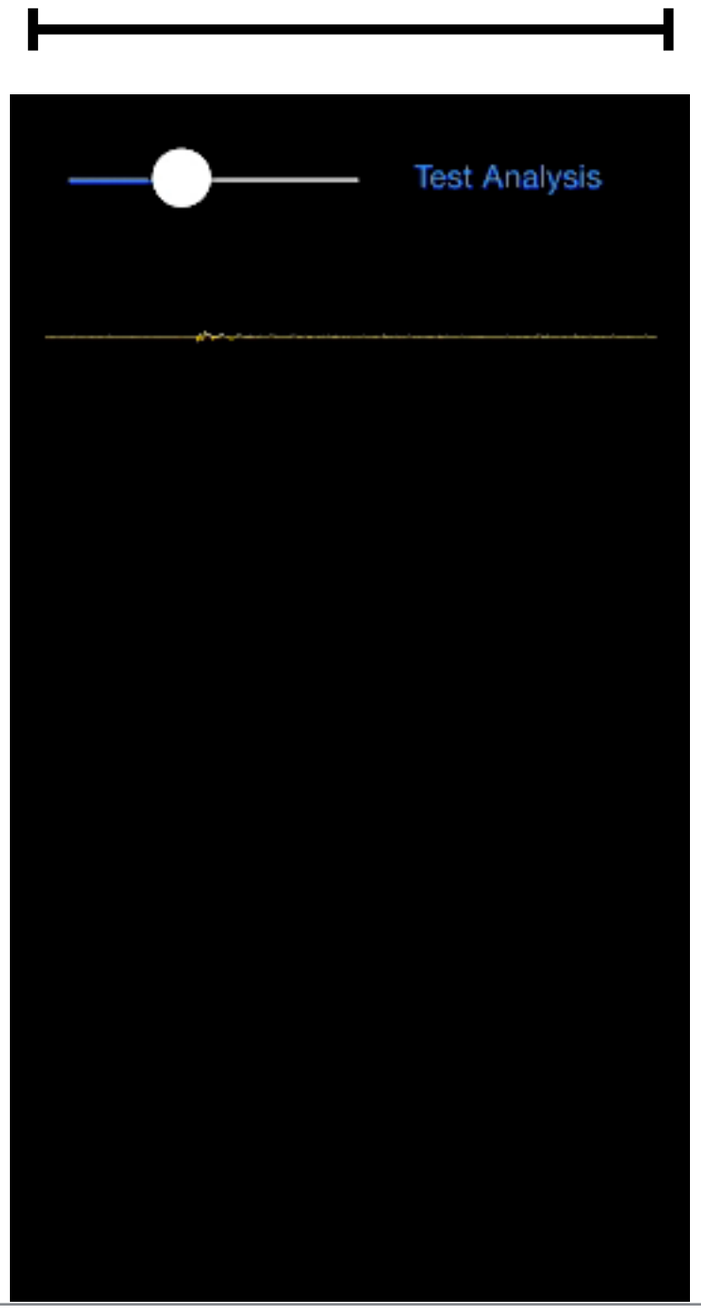
```
inputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32)) {
      vDSP_vsq(data, 1, data, 1, numFrames*numChannels);
      var meanVal = 0.0;
      vDSP_meanv(data, 1, &meanVal, numFrames*numChannels);
  }
```

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

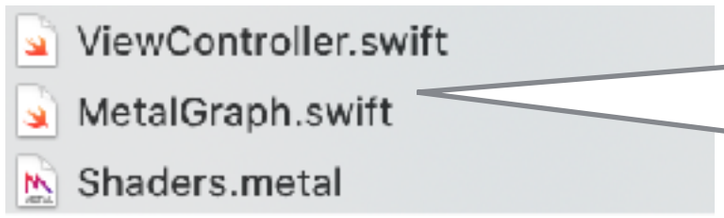
- …this is not a graphics course
- but we need to use the Metal API

Metal

Apple used the mobile multiplayer online battle arena game *Vainglory* to demonstrate Metal's graphics capabilities at the iPhone 6's September 2014 announcement event[1]

Developer(s)	Apple Inc.
Initial release	June 2014; 6 years ago
Stable release	3 / June 2019; 1 year ago
Written in	Shading Language: C++14, Runtime/API: Objective-C
Operating system	iOS, iPadOS, macOS, tvOS
Type	3D graphics and compute API
License	proprietary
Website	developer.apple.com/metal/

the metalGraph class



drag class/shaders
into project, if needed

```
lazy var graph:MetalGraph? = {
        return MetalGraph(mainView: self.view)
    }()
```

declare and init property

```
// add in a graph for displaying the audio
graph?.addGraph(withName: "time",
            shouldNormalize: false,
          numPointsInGraph: AUDIO_BUFFER_SIZE)
```

add graph names to controller
and how many expected points in array

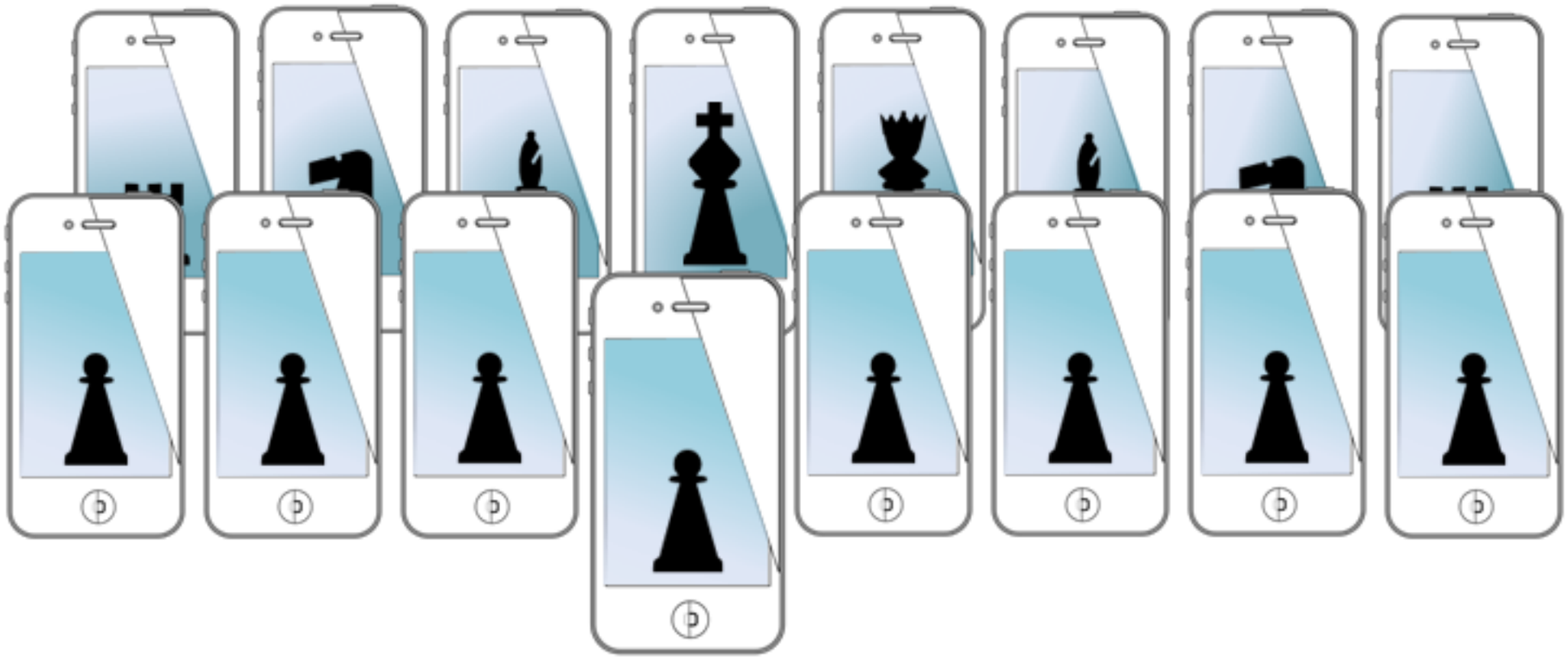
```
// periodically, display the audio data
graph?.updateGraph(
           data:  timeData,
           forKey: "time"
         )
```

refresh data for each
named graph key

Properties: automatic screensize (pixel) downsampling
automatic coloring based in iOS scheme,
efficient memory management through vertex buffers
adding functionality is a pain if you are new to graphics

audio graphing demo!

MOBILE SENSING LEARNING



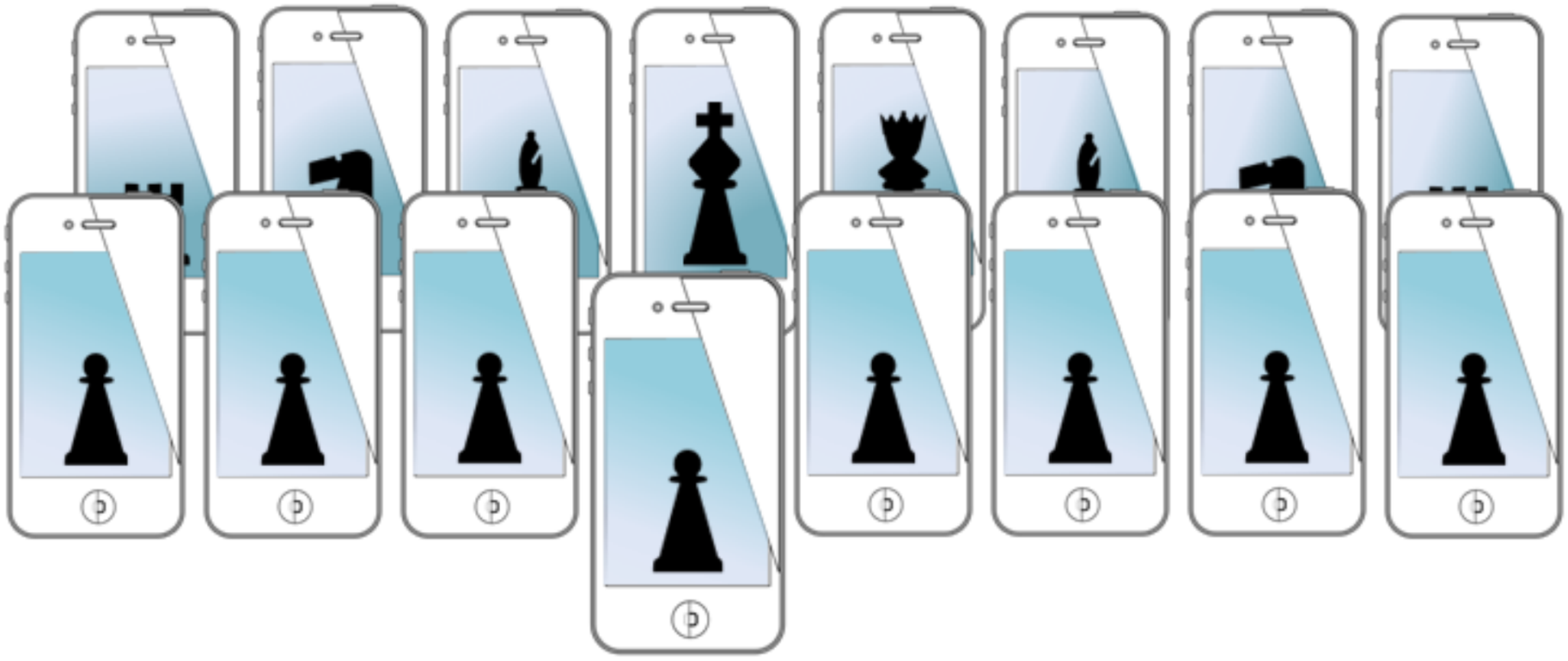
CS5323 & 7323

Mobile Sensing and Learning

audio graphing, sampled data, & accelerate

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

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

video lecture: accelerate & FFT

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

agenda (Video)

- high level: what is audio processing
- audio analysis: the FFT
- how to you use the FFT
- how to program the FFT
- an example: graphing whistles

processing audio

- lots of space to explore, processing signals is big!
  - **great reference**: “DSP First” by McClellan, Schafer, and Yoder
    - http://www.rose-hulman.edu/DSPFirst/visible3/contents/index.htm
  - **a reference for CSE’s**: “Signal Computing” by Stiber, Stiber, and Larson
    - http://faculty.washington.edu/stiber/pubs/Signal-Computing/
  - filtering
    - only let certain frequencies through
  - analysis
    - analyze characteristics of signal (like pitch or amplitude)
  - synthesis
    - play around with different ideas, and see what sounds good!
    - not just pure synthesis, but also manipulation (like guitar effects)

processing audio

- the accelerate framework is great for analyzing audio
- for instance, what is the max value of an audio chunk?

```
[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);
```

input data from microphone

output: max

array length

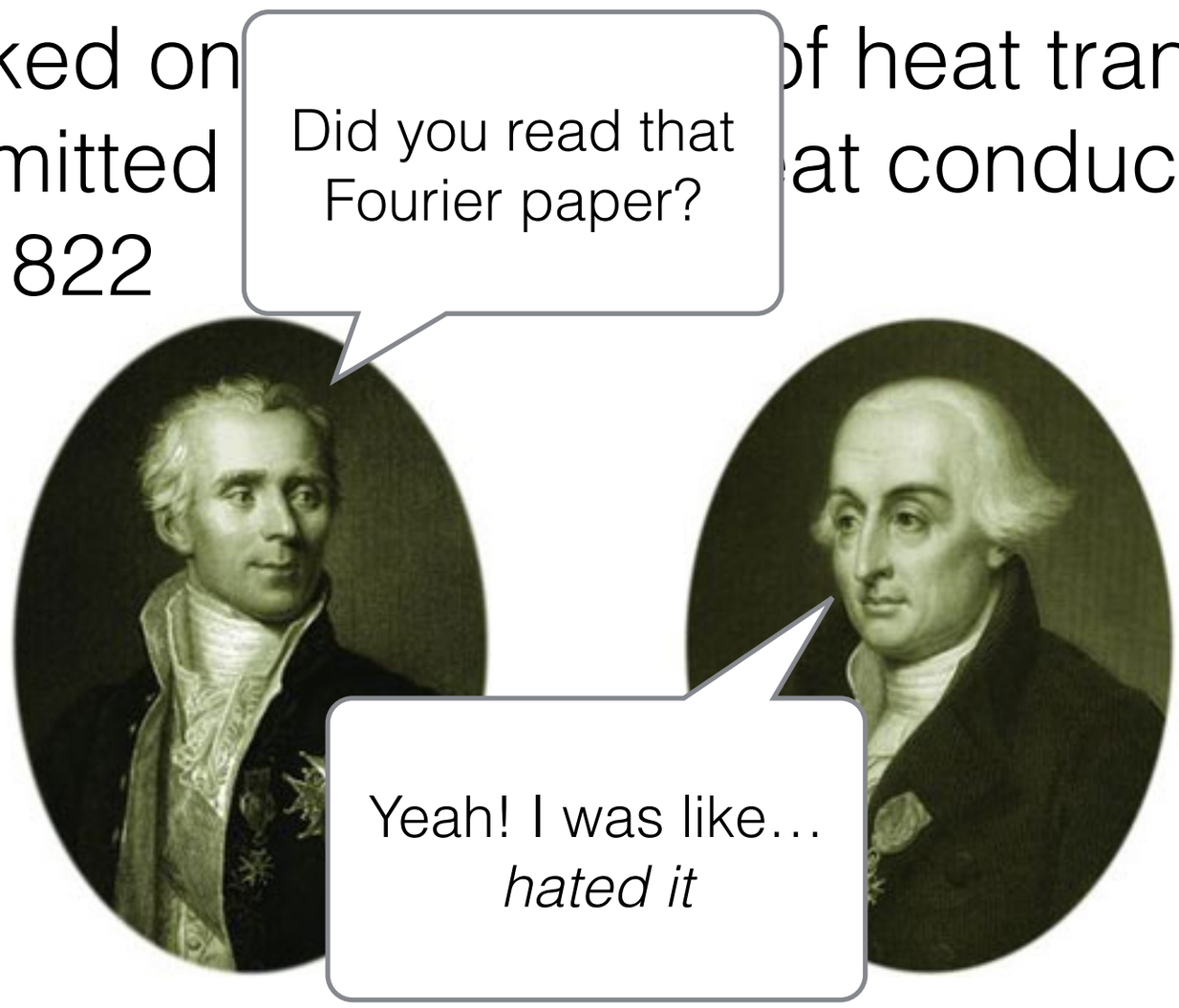
max of audio might be used to detect:

taps next to **phone**

- for now, we're going to stick with analysis
  - specifically, the **Fourier Transform**

Fourier

A mathematician
Worked on of heat transfer
Submitted at conduction
in 1822



Laplace Lagrange

Fourier

A mat
Worke
Subm
in 18

of heat transfer
eat conduction

He said all signals could be defined as a sum of simpler functions

Simpler functions!? Like sine waves? Come on!

Laplace

Lagrange

Fourier

He developed a method that allowed one to transform a signal into a **sum of sine waves**.

The transform was eventually called The Fourier Transform

~150 years later Cooley and Tukey would discover a way to calculate it really fast on a computer

Cooley

I created the FFT to commit espionage against the Russians!

I coined the term “bit”

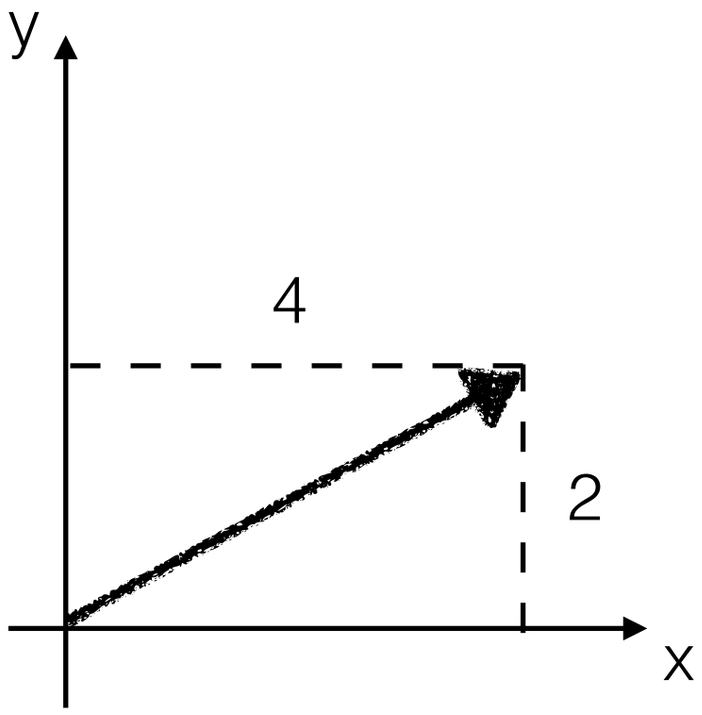
Tukey

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)

FT by vector intuition

think of it as a vector projection (intuitively)



(4,2)

where did these numbers come from?

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

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

point by point multiplication and add it up!

tells us "how much" of the vector is the result of these vectors

$\vec{x}$

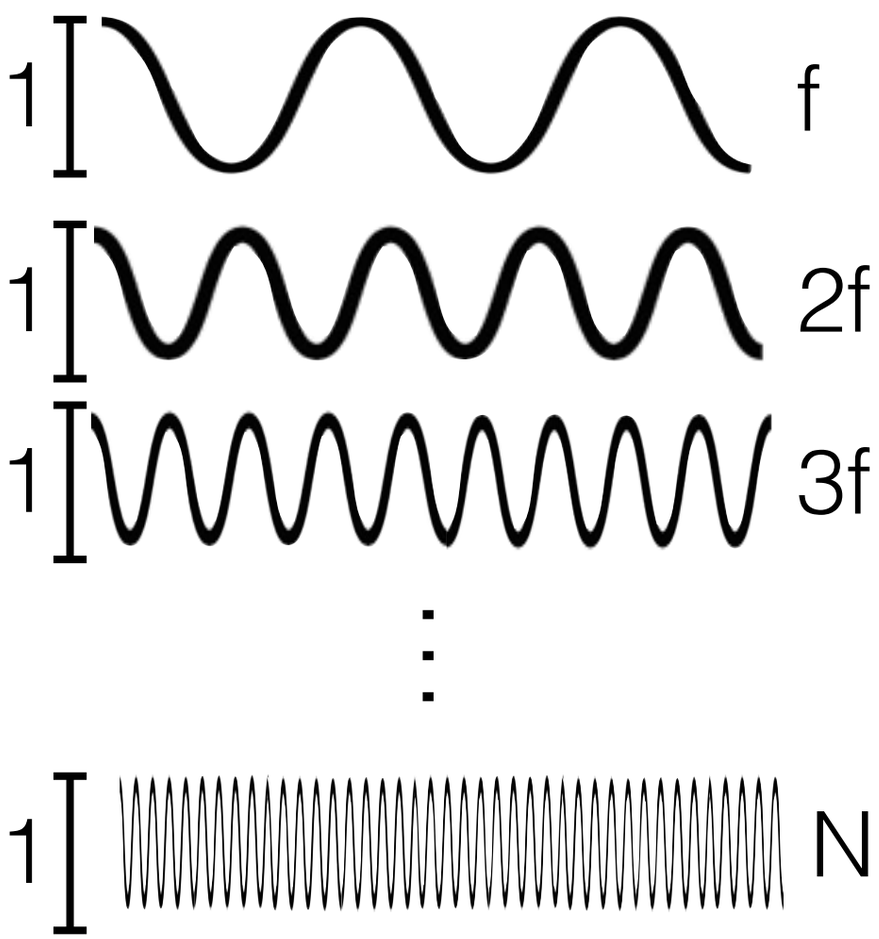
$\vec{y}$

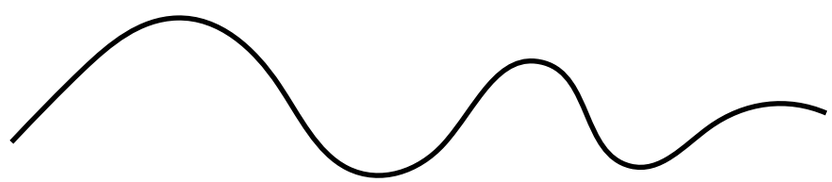
$\vec{x} \bullet \vec{y} = 0$

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

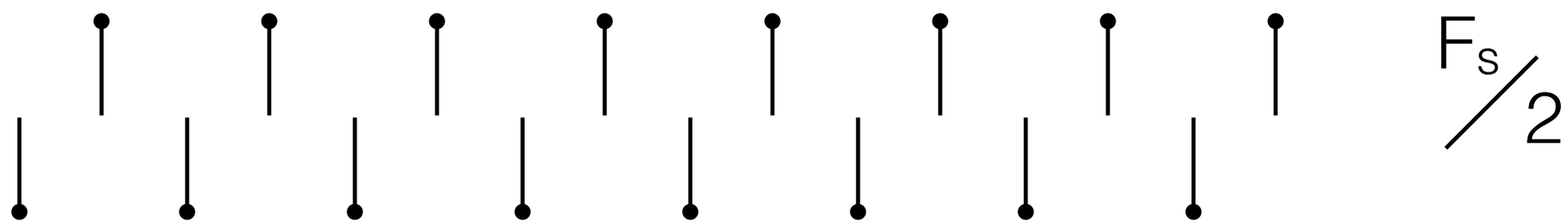
FT by sine waves

what if the orthogonal vectors were functions?





2.5	10°
0.1	45°
0.5	-12°
⋮	⋮
.005	0°



FT by the numbers

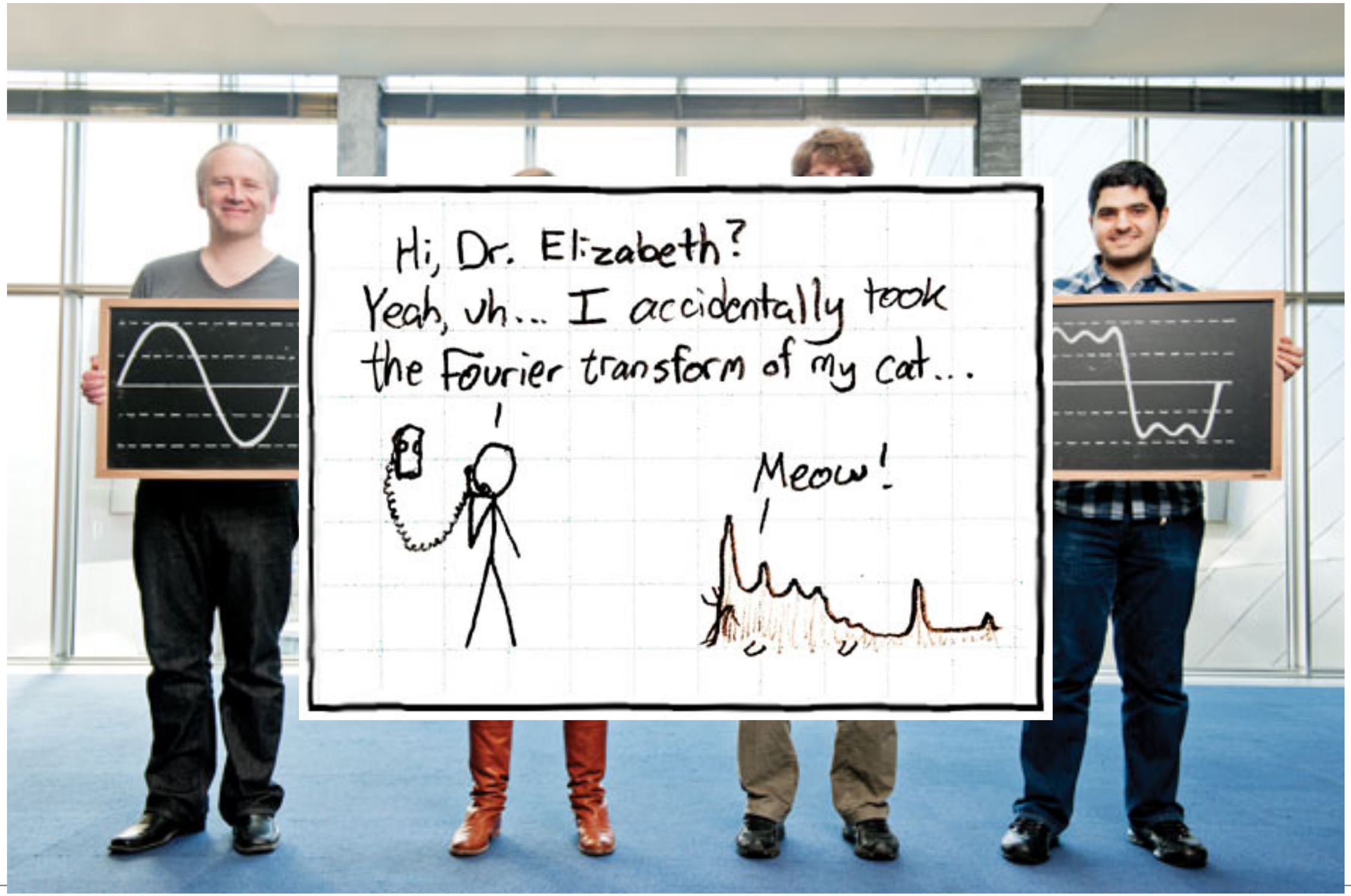
content	frequency									
	0f	0.7	0°						=	0.7
	1f	2.5	10°	+	2.5	-10°	-1f	~	2.5 cos(2*pi* (f) t+10°)	
	2f	0.1	45°	+	0.1	-45°	-2f	~	0.1 cos(2*pi* (2f) t+45°)	
	3f	0.5	-12°	+	0.5	12°	-3f	~	0.5 cos(2*pi* (3f) t-12°)	
	⋮	⋮	⋮		⋮	⋮				
	N f	.005	0°	+	.005	0°	-N f	~	0.005 cos(2*pi* (Nf) t)	

FT by video

http://en.wikipedia.org/wiki/User:LucasVB/Gallery

FT by hilarity

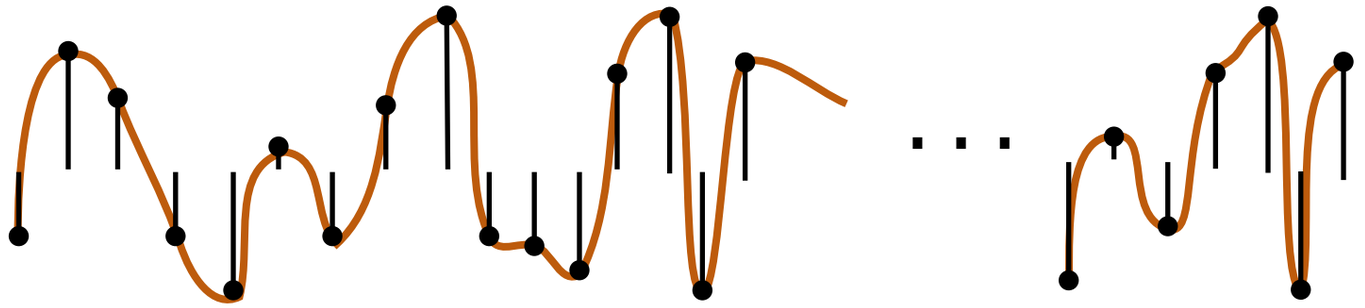
http://www.preposterousuniverse.com/blog/2014/11/27/thanksgiving-9/



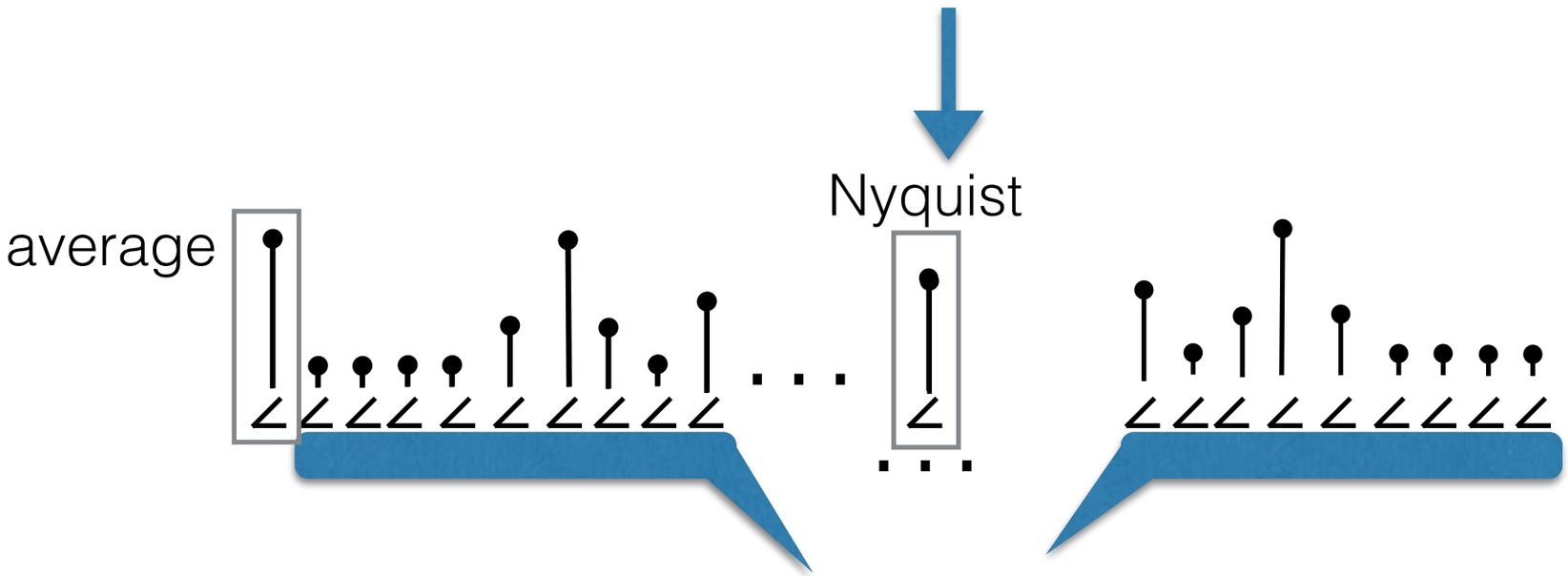
the FFT

- we will use the FFT in the accelerate framework
  - complexity is O(N $\log_2$(N))
- the FFT takes an array of numbers and gives you back an array of complex numbers
  - both input and output arrays are the same length
  - the indices of the input array denote increasing time
  - the indices of the output array denote increasing frequency
- don't quite get the math? so what? understand the output array…

time and frequency



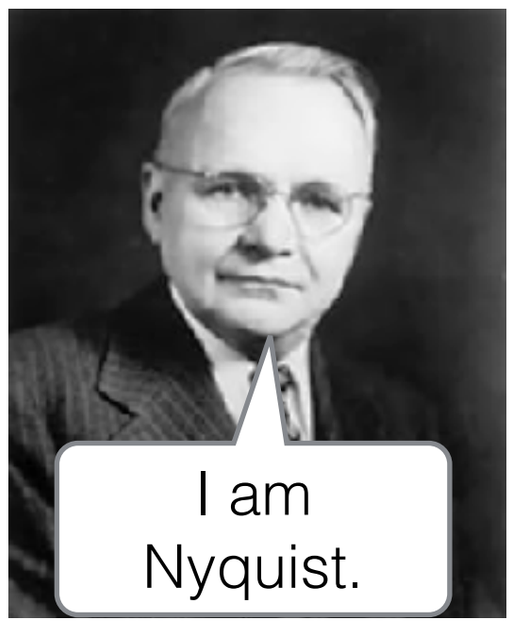
time array, N=512



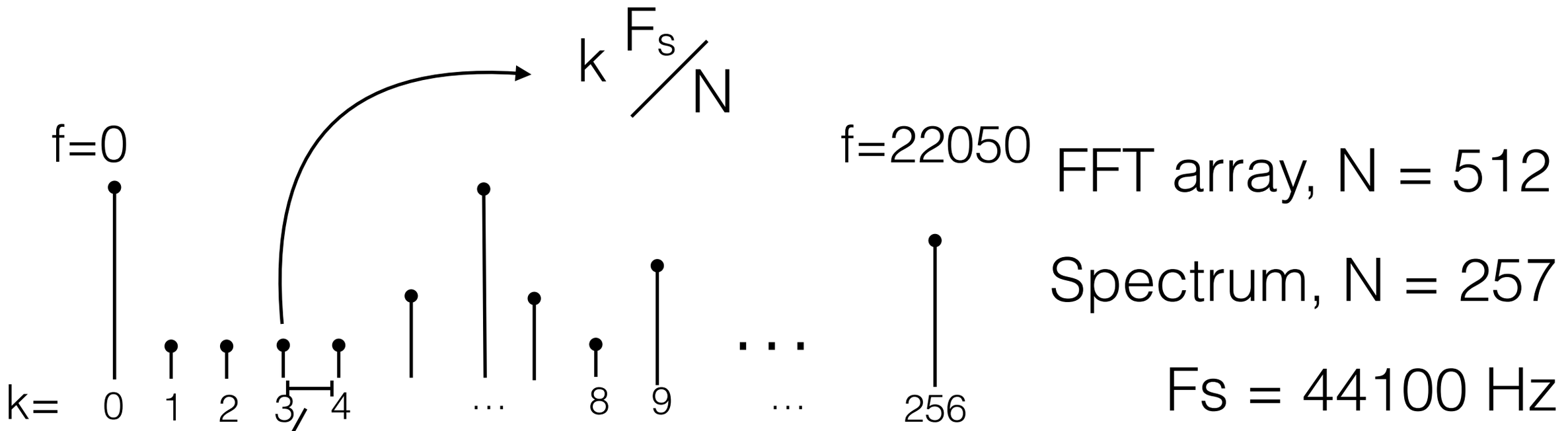
FFT array, N = 512

magnitude, even symmetry

magnitude spectrum



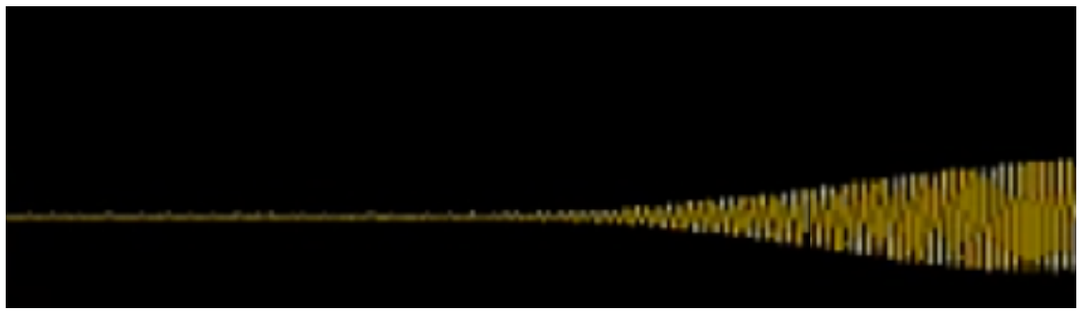
time and frequency



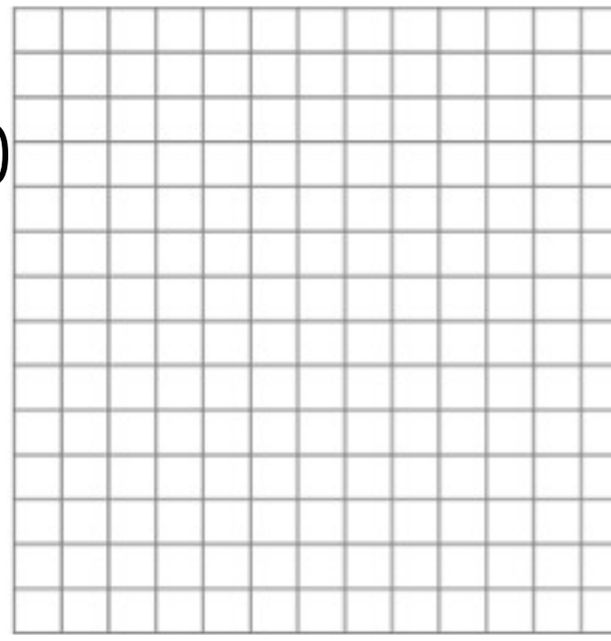
$$df = \frac{F_s/2}{N/2} = F_s/N = 44100/512 = \sim 86 \text{ Hz}$$

using the FFT

raw audio

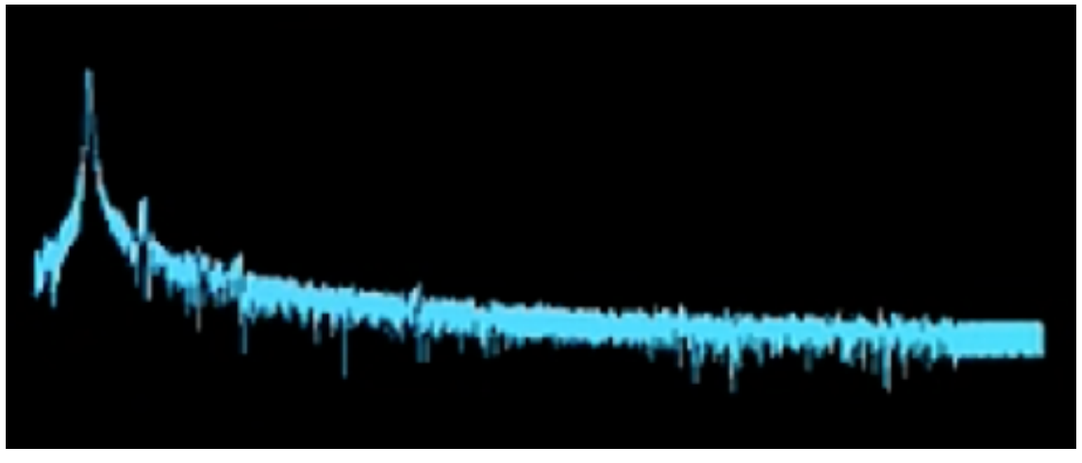


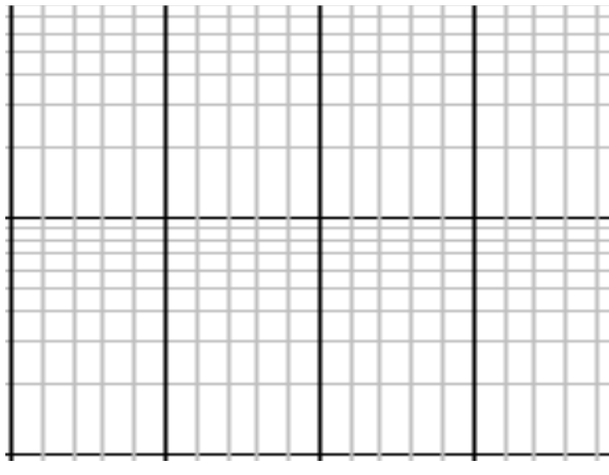
magnitude FFT



magnitude FFT
in dB

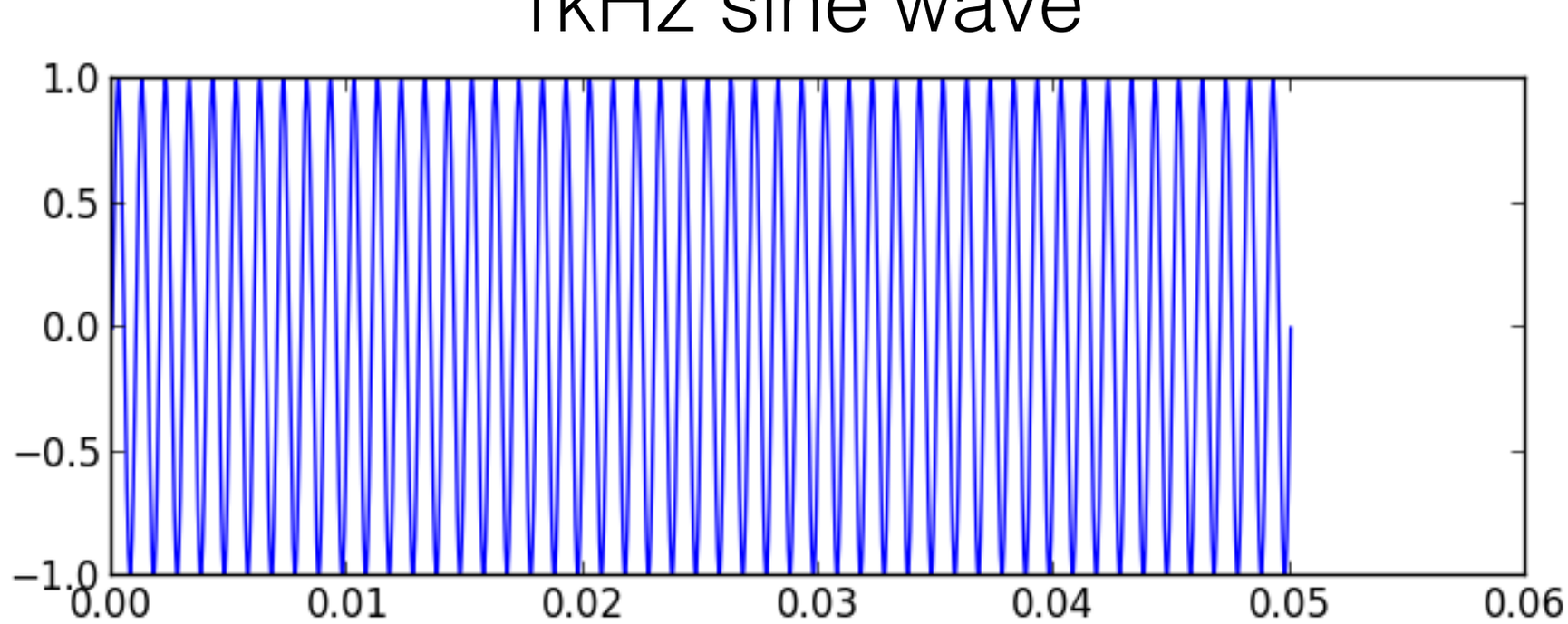
$20 \log_{10}(|FFT|)$



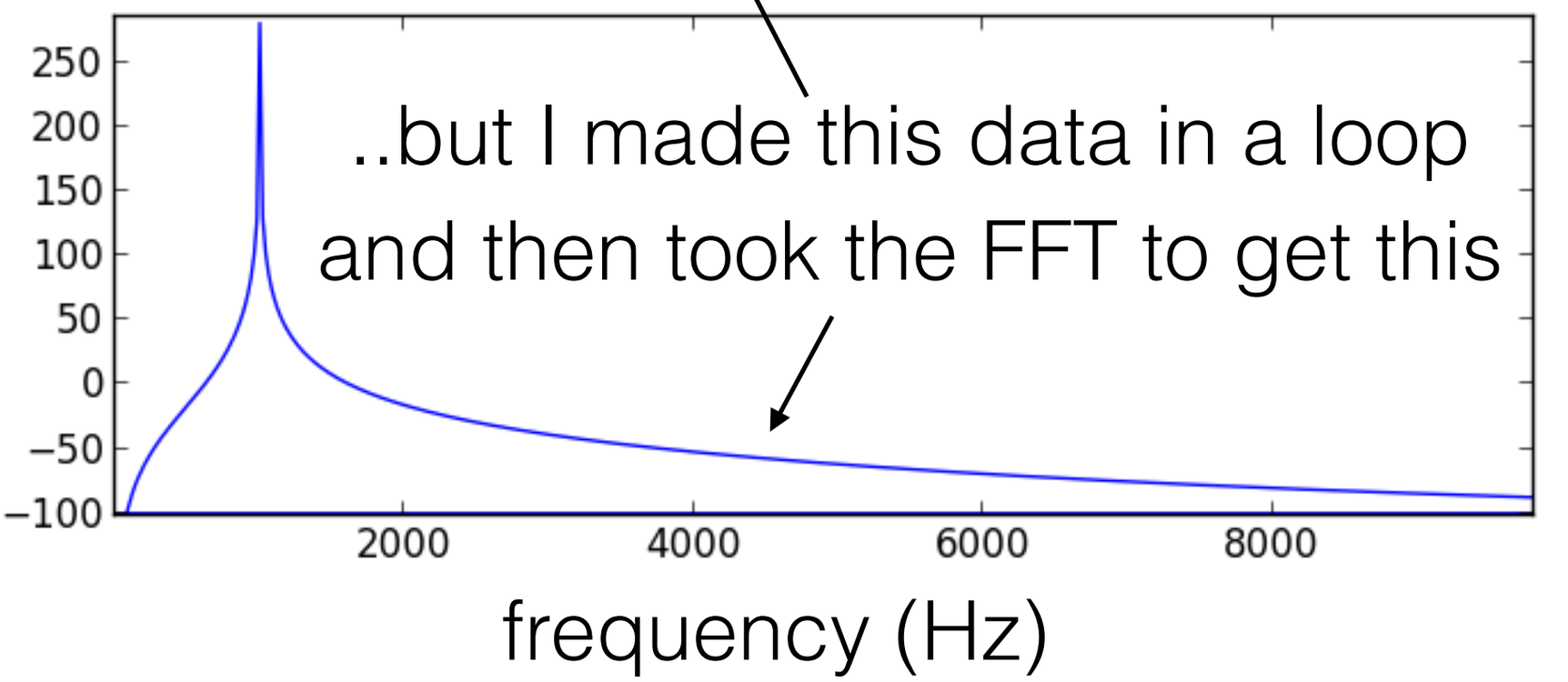


some fft examples

this is just a plot of an array of values sampled **over equal time intervals**

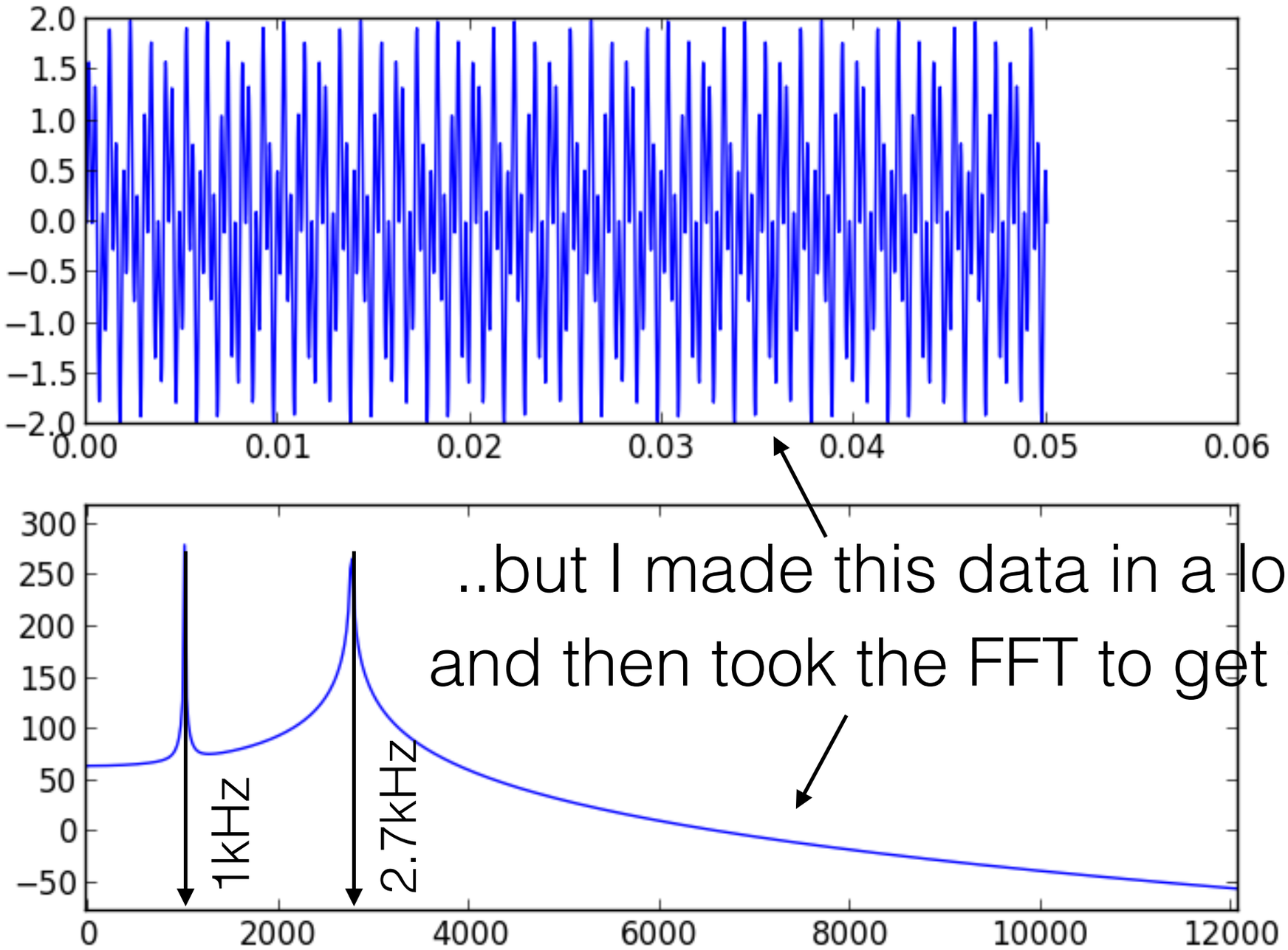


this is just a plot of an array of FFT values **its indices are equally sampled over frequency**



some fft examples

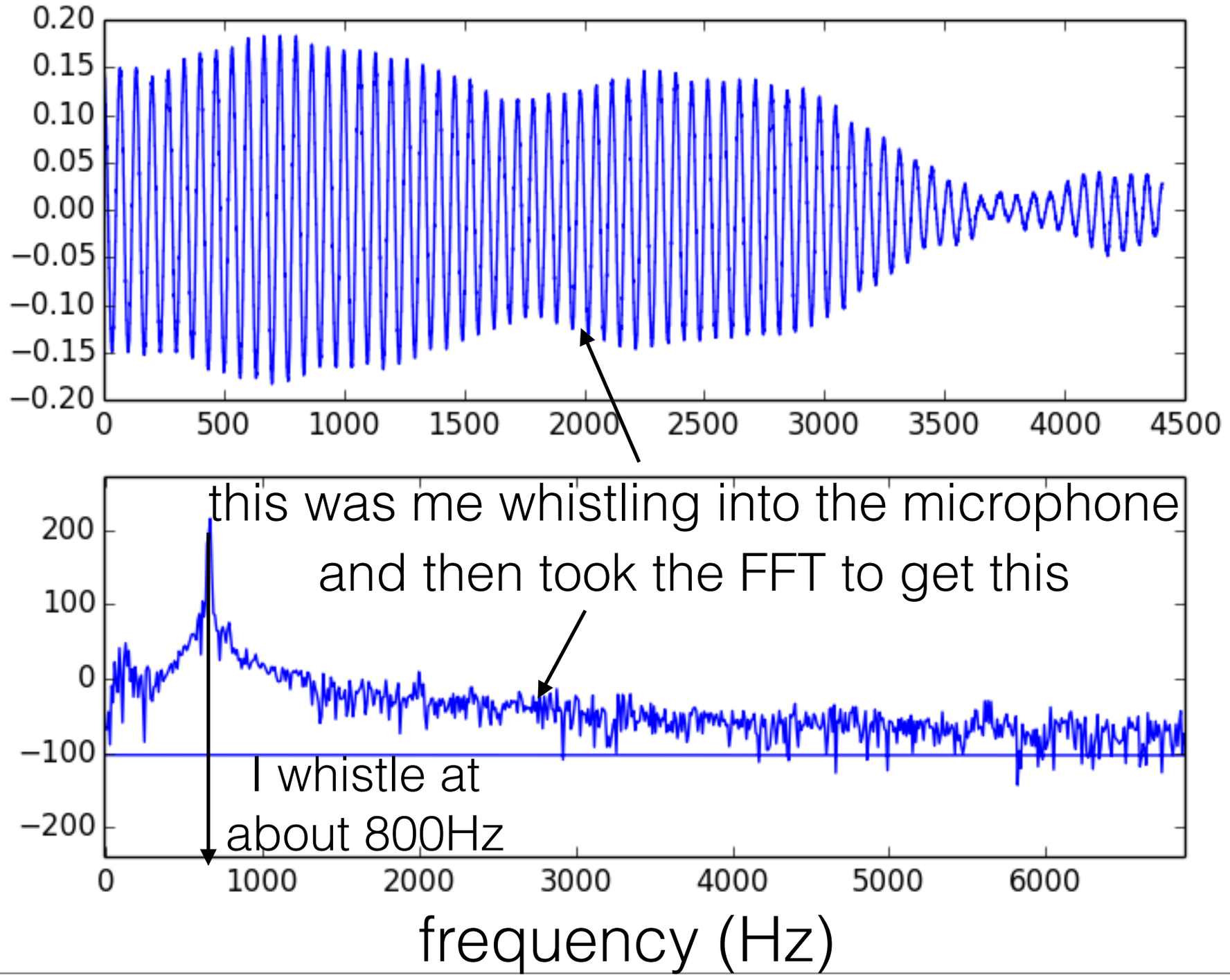
1kHz sine wave +2.7kHz sine wave



..but I made this data in a loop and then took the FFT to get this

some fft examples

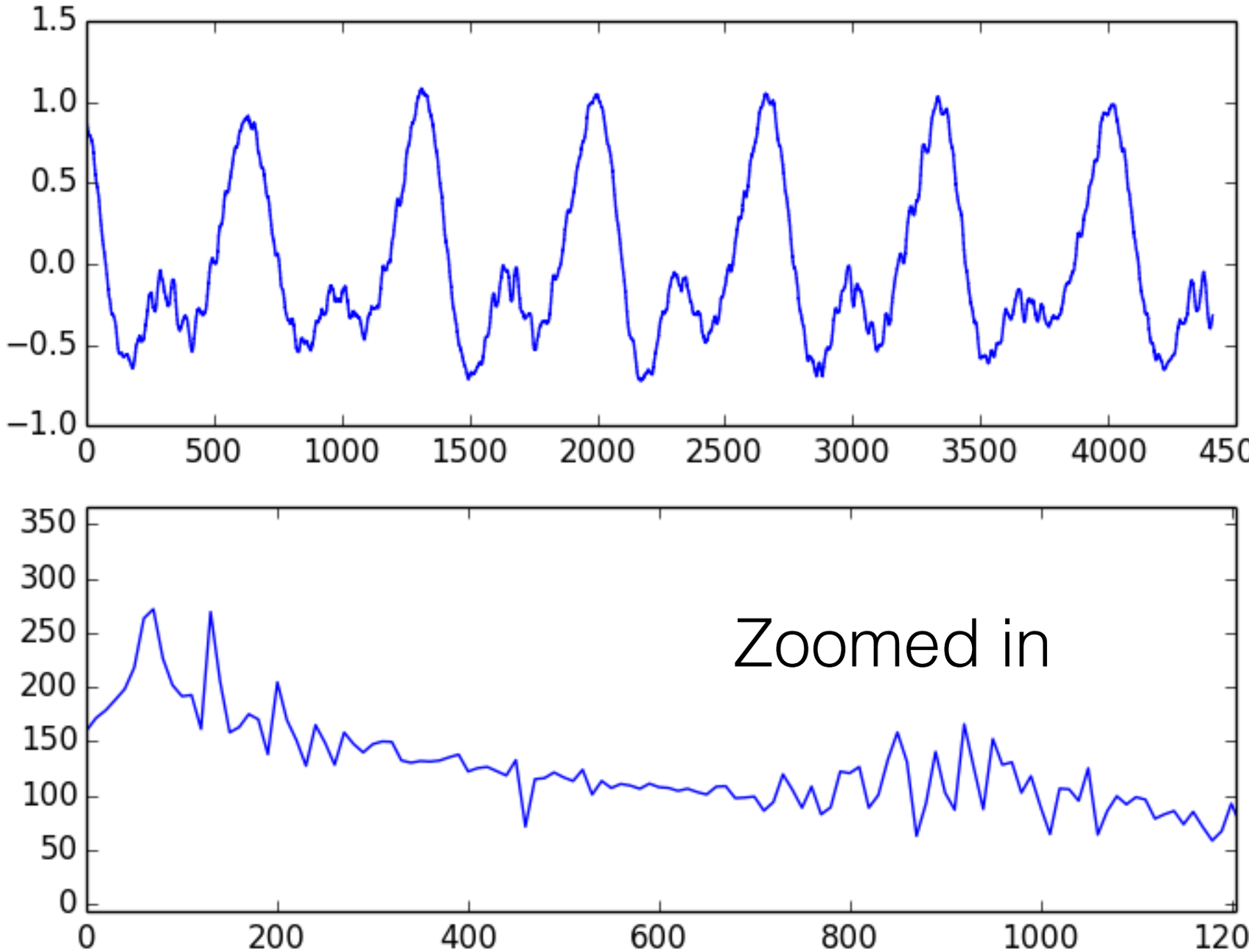
a person whistling



some fft examples

humming

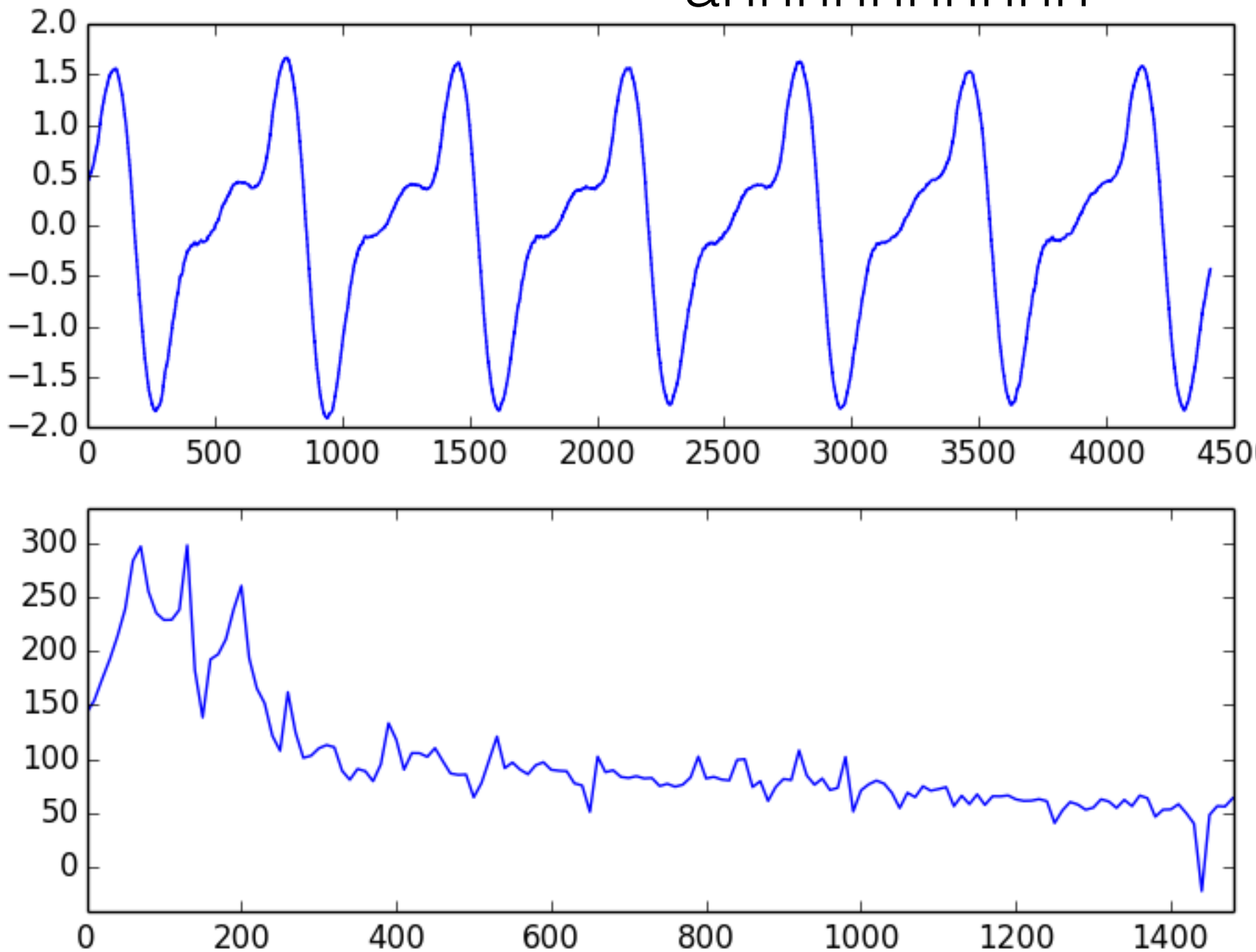
oooooooooooo



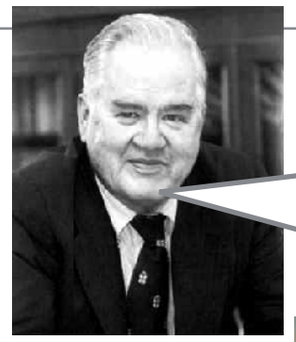
some fft examples

humming again

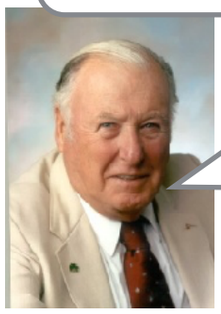
ahhhhhhhhhh



programming the FFT



This is how to use our stuff



Yeah!

```
//setup the fft
private lazy var fftHelper:FFTHelper? = {
        return FFTHelper.init(fftSize: Int32(BUFFER_SIZE))
    }()
```

fft size == buffer size

```
var fftData:[Float]
fftData = Array.init(repeating: 0.0,
                     count: BUFFER_SIZE/2)
```

Magnitude buffer is half the size!

```
// copy time data to swift array
self.inputBuffer!.fetchFreshData(&timeData,
                         withNumSamples: Int64(BUFFER_SIZE))
// now take FFT
fftHelper!.performForwardFFT(withData: &timeData,
           andCopydBMagnitudeToBuffer: &fftData)
```

input array

magnitude out
half window size

highly optimized!! even using tricks we have not discussed

programming the FFT

```
#import "SMUFFTHelper.h"

@property (strong, nonatomic) FFTHelper *fftHelper;
```

```
//setup the fft
_fftHelper = [[FFTHelper alloc]initWithFFTSize:BUFFER_SIZE];
```

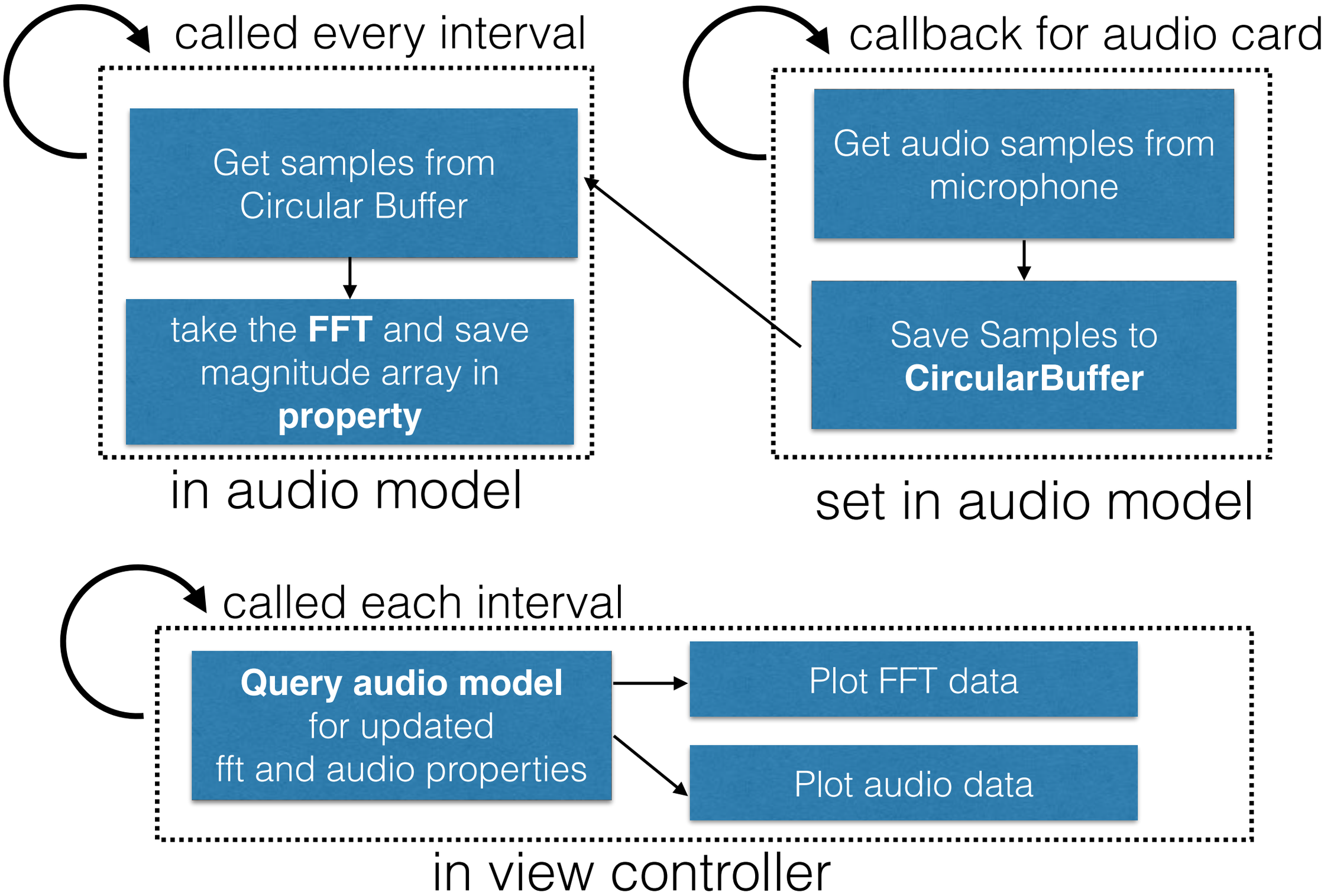
fft size == window size

```
float* fftMagnitude = malloc(sizeof(float)*BUFFER_SIZE/2);

// take forward FFT
[self.fftHelper performForwardFFTWithData:arrayData
               andCopydBMagnitudeToBuffer:fftMagnitude];

free(fftMagnitude);
```

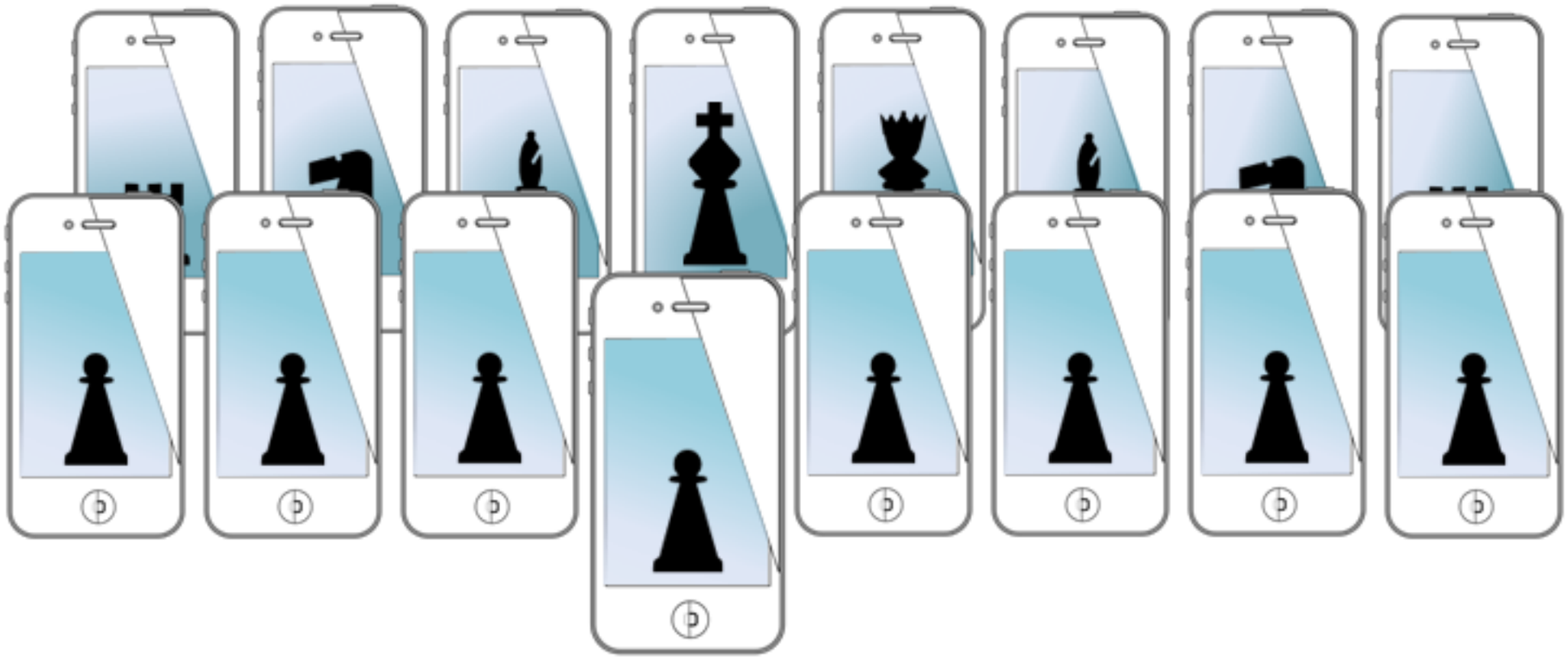
input array
N=window size

magnitude out
N=half window size

The program


called every interval
Get samples from Circular Buffer
take the **FFT** and save magnitude array in **property**
in audio model
callback for audio card
Get audio samples from microphone
Save Samples to **CircularBuffer**
set in audio model
called each interval
Query audio model for updated fft and audio properties
Plot FFT data
Plot audio data
in view controller

MOBILE SENSING LEARNING



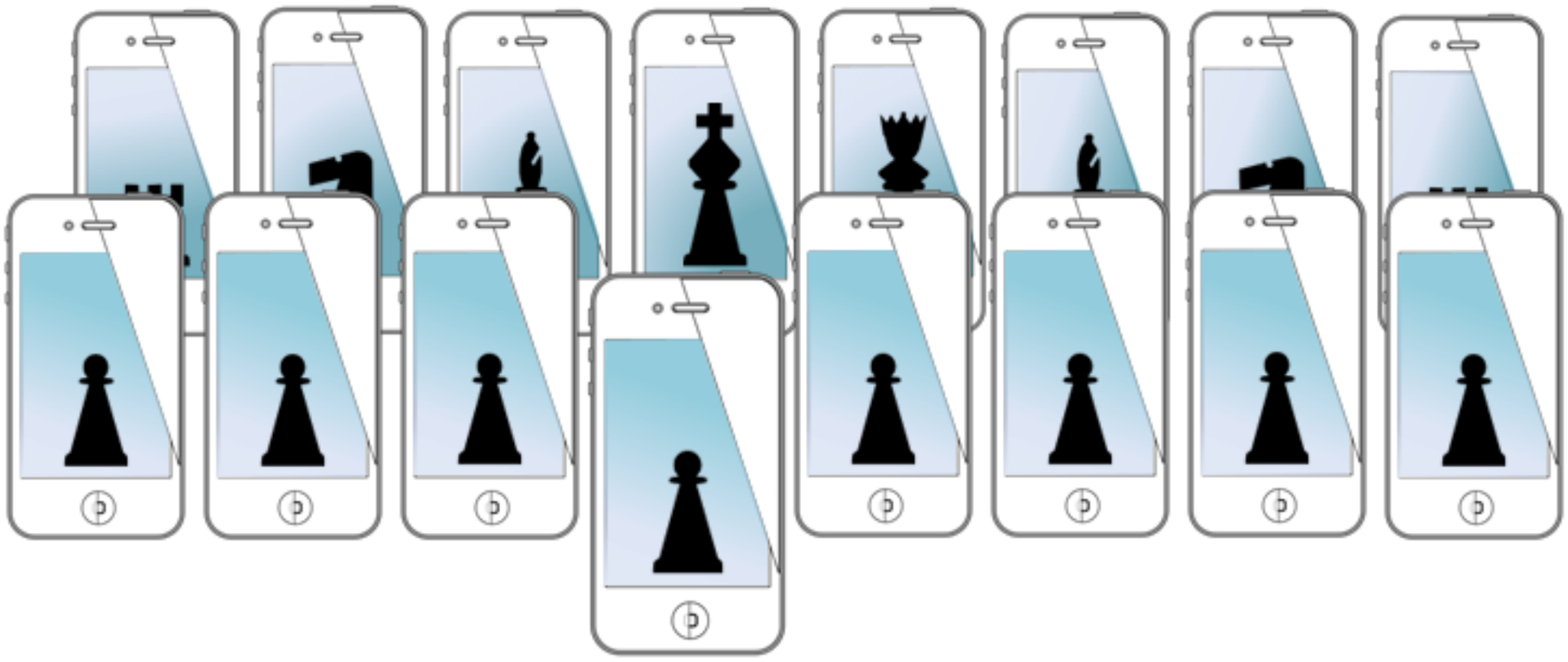
CS5323 & 7323

Mobile Sensing and Learning

video lecture: accelerate & FFT

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

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

Supplemental Slides: filtering and windowing

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

Supplemental Slides

- these slides were removed from the course because of their complexity
- you need a good background in signal representation and time series analysis to really understand these slides

Optional Concepts not Covered in Course Anymore

filters!

- we will cover what we can…

signals and systems

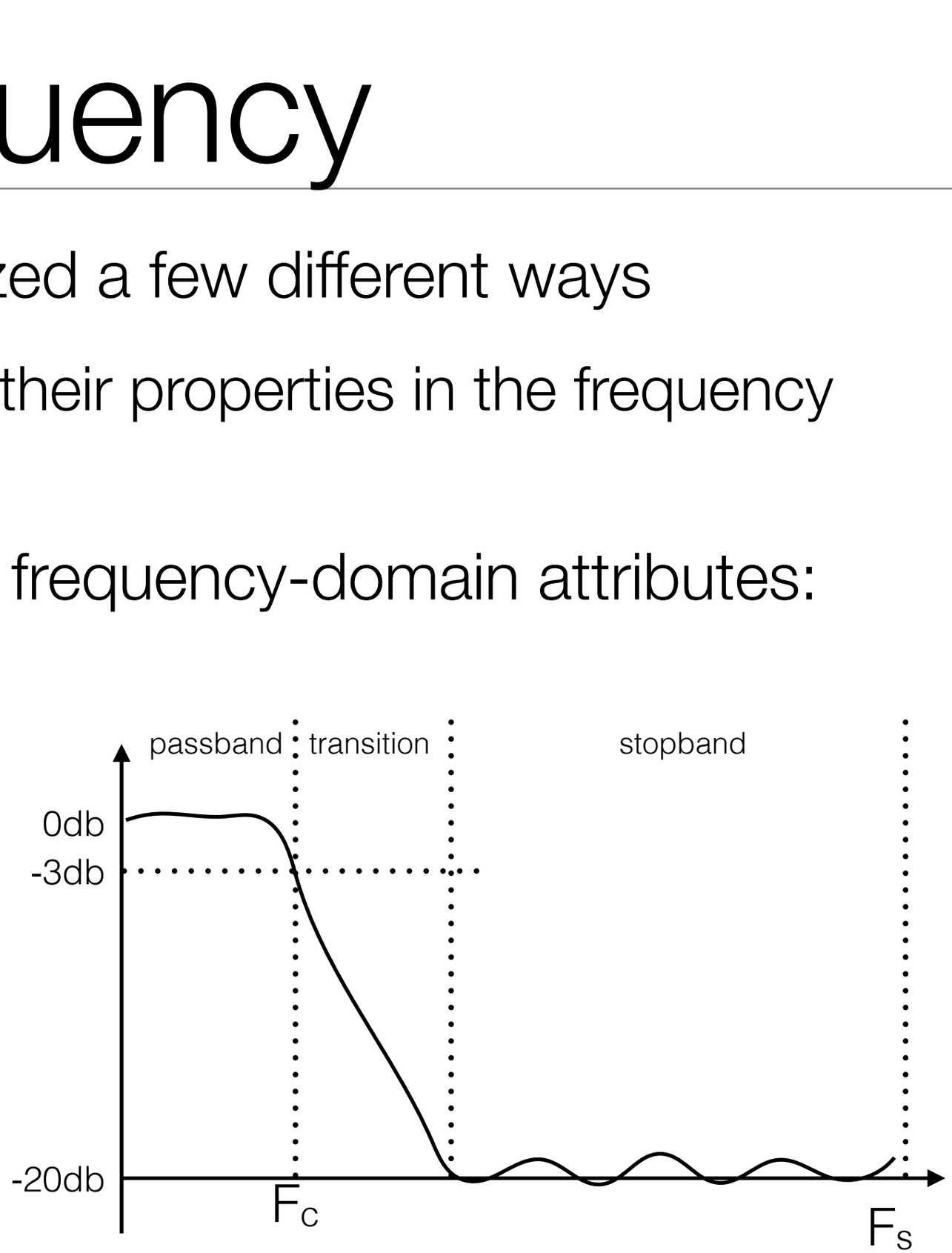
- signals are collections of sampled data (arrays)
  - such as audio, accelerometer, etc.
  - can also be 2D, like images
- systems are objects which manipulate signals
  - characterized by their “input/output” relationships
  - we say “x[n] is passed through H, resulting in y[n]”

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

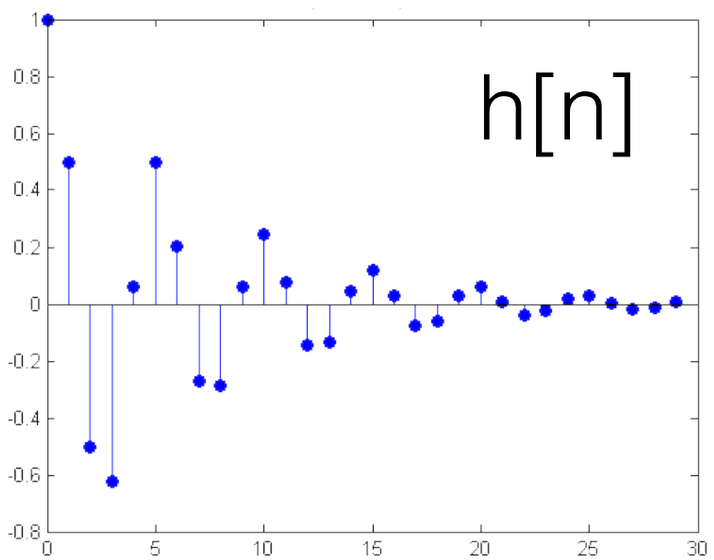
filters in frequency

- 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!

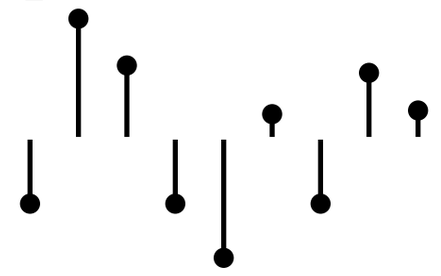
filtering by convolution

- 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

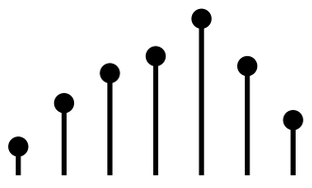
$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

convolution

x[i]

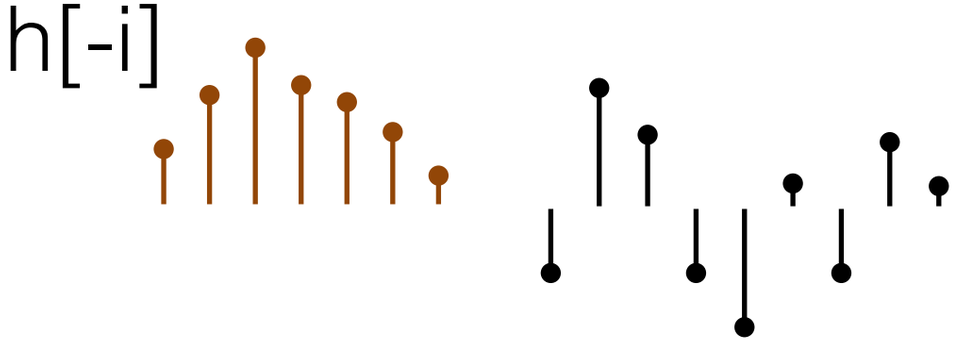


h[i]

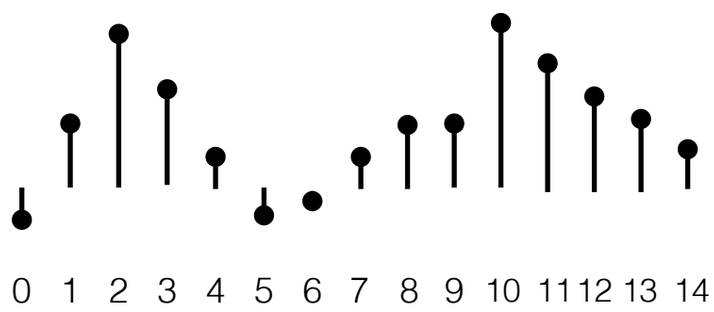


$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

h[-i]



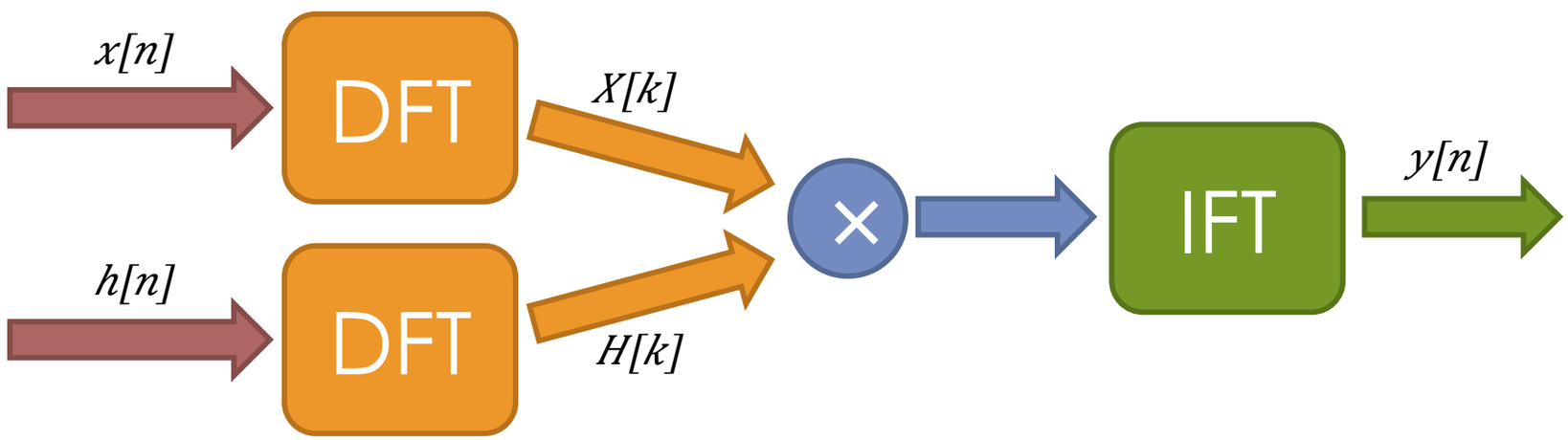
y[n]



length N M N+M-1

convolution efficiency

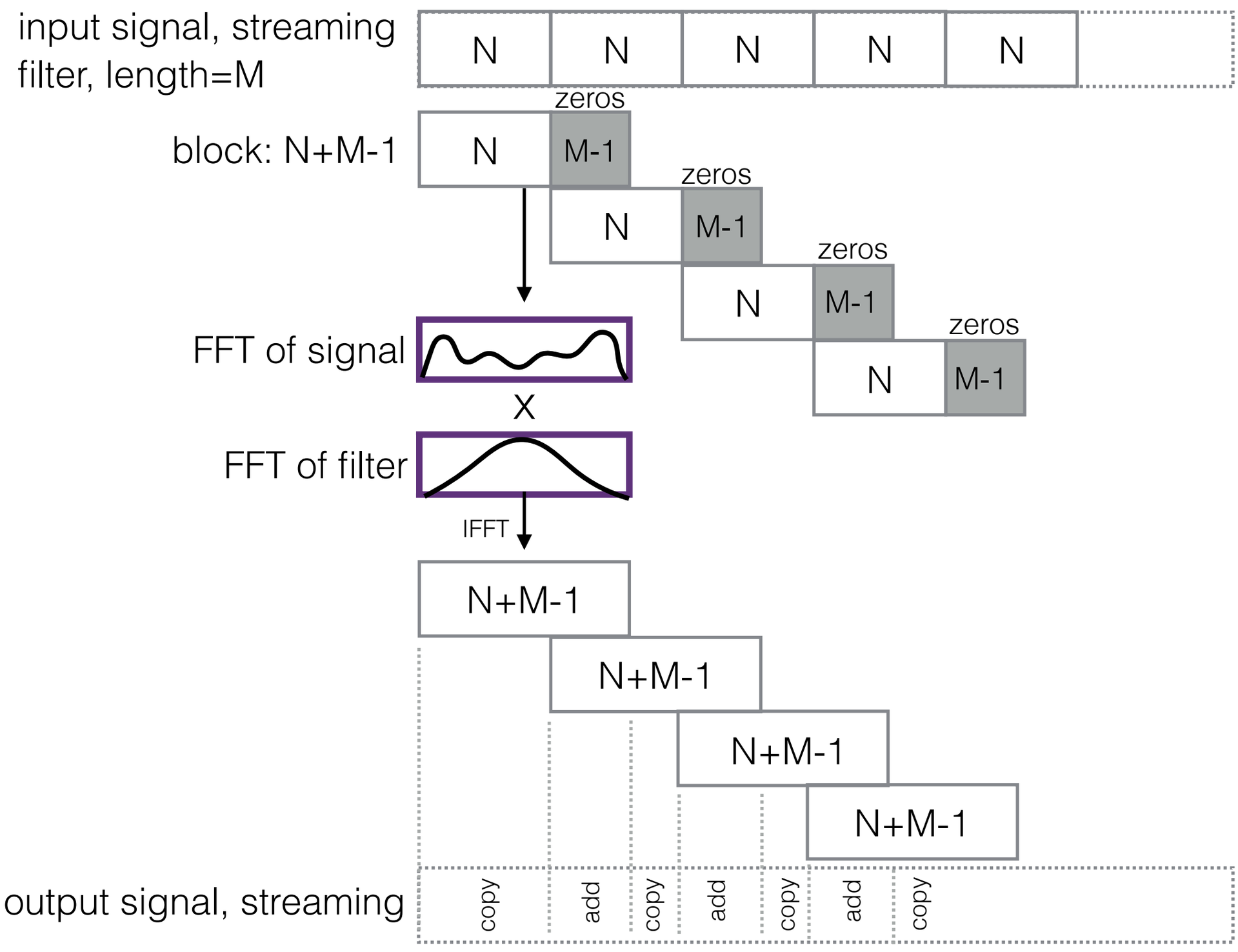
- algorithmic complexity
  - convolution is not particularly efficient, O(NxM)
- to convolve faster, use that Fourier property:
  - “convolution in time is multiplication in frequency”
- why not just multiply to begin with!



its circular

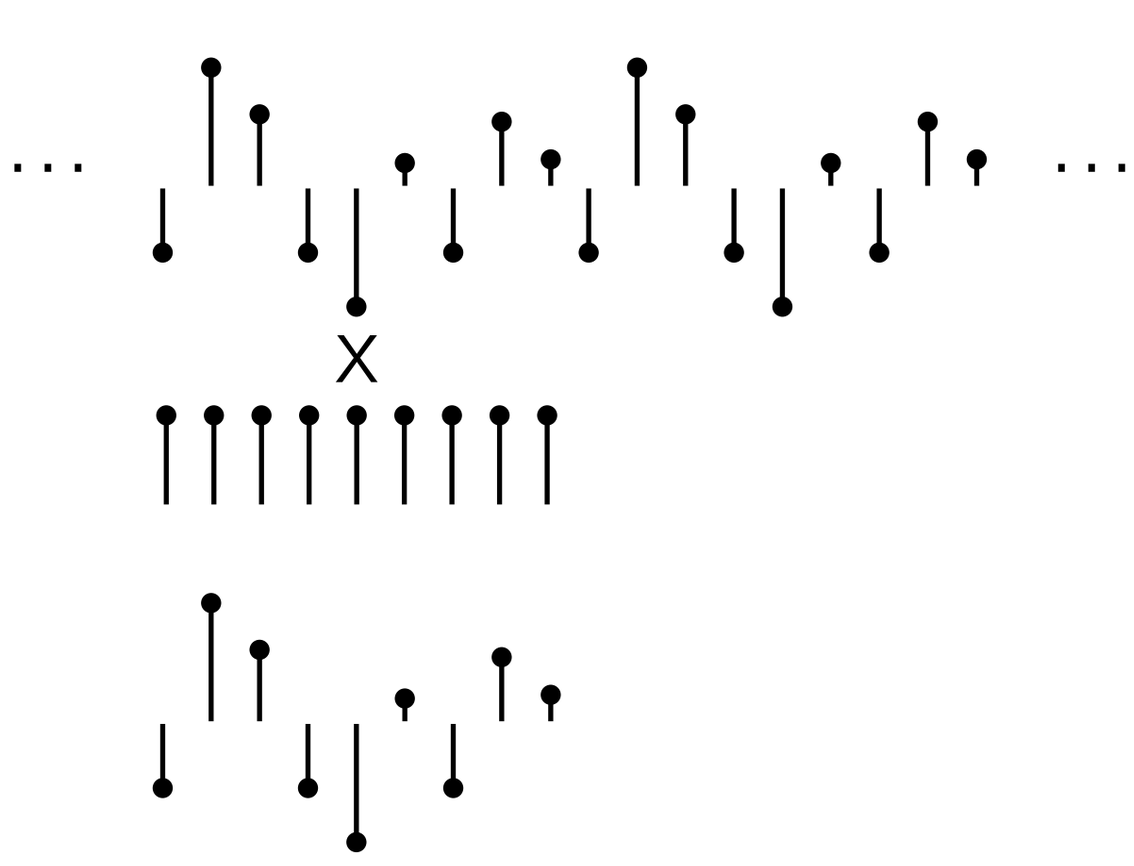
- just using N point FFT performs “Circular Convolution”
  - which is not linear convolution
  - causes the tail end of the convolution to “wrap around” to the beginning
    - FFT assumes the function is periodic (we did not talk about this)
- be aware of circularity when filtering your signal with the 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

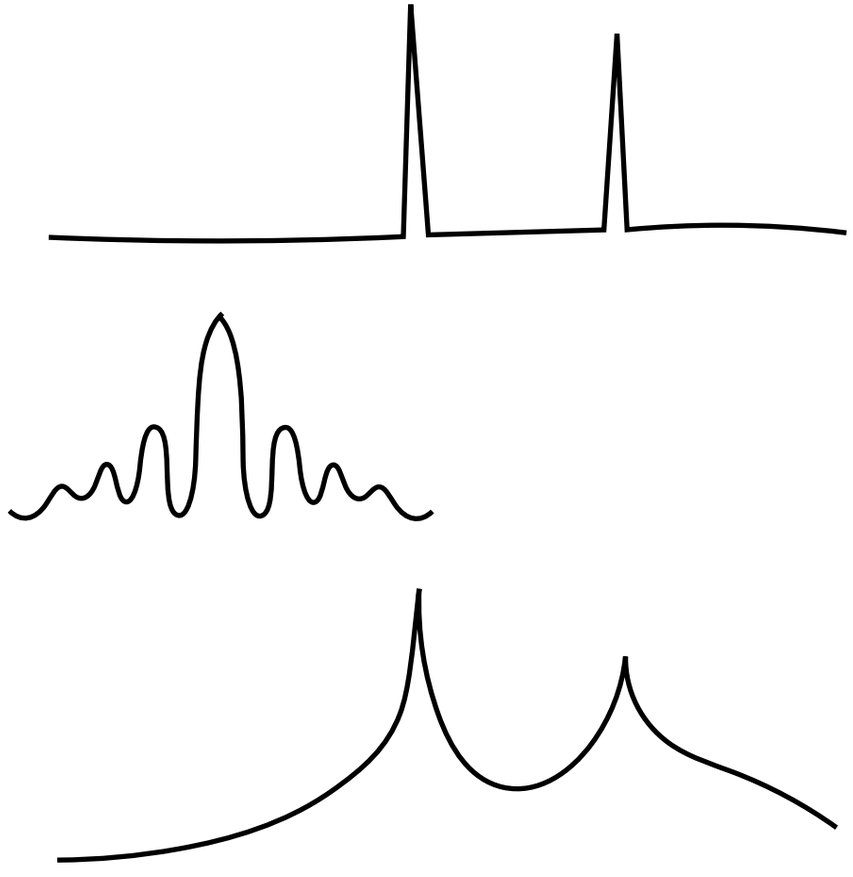
overlap and add



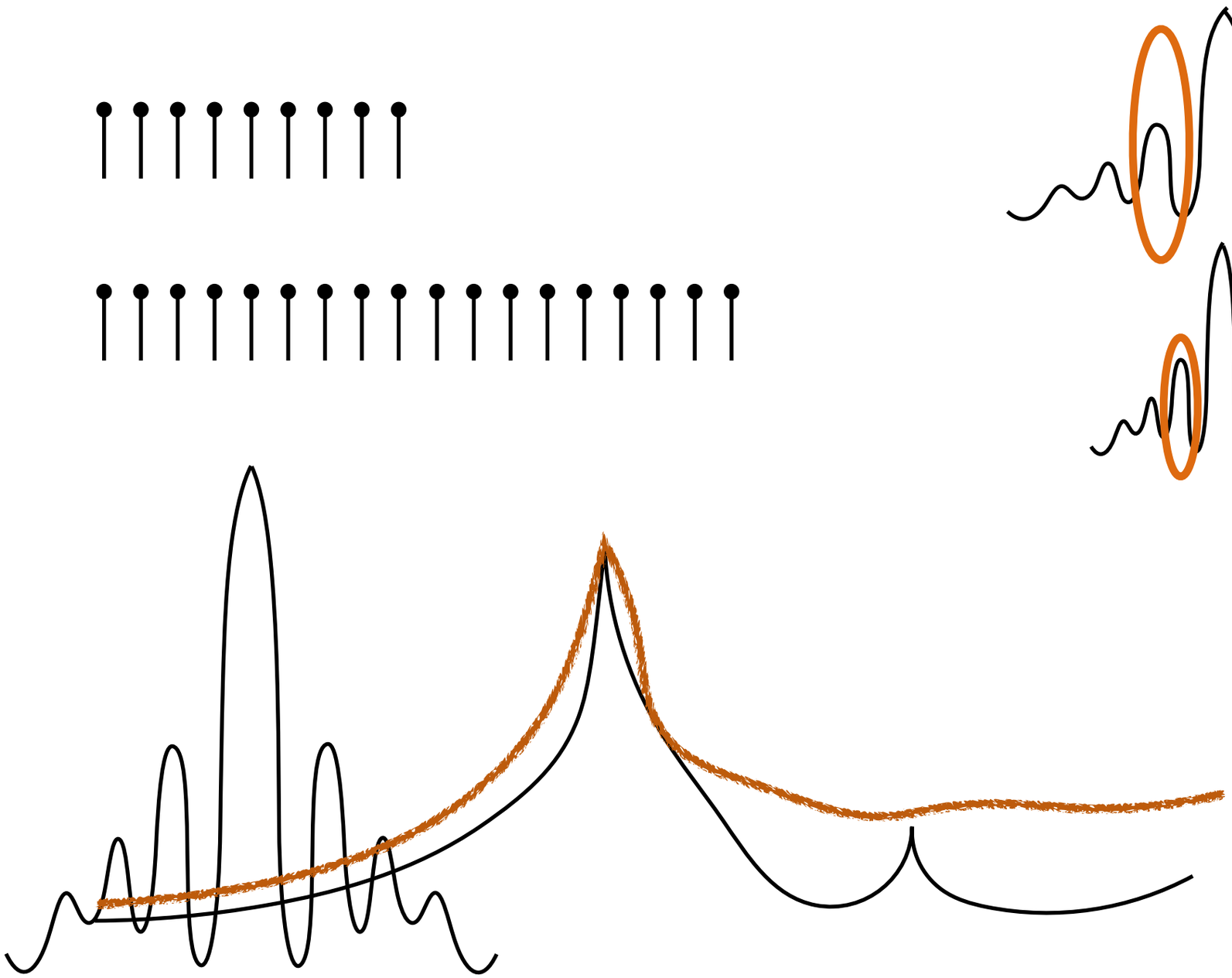
windowing: spectral BW widening

- 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…

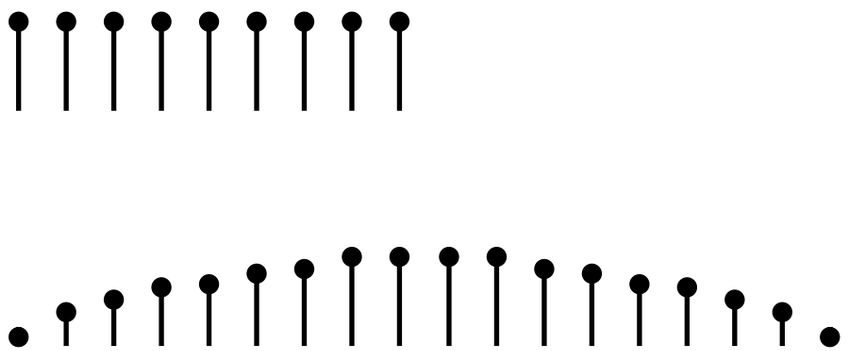


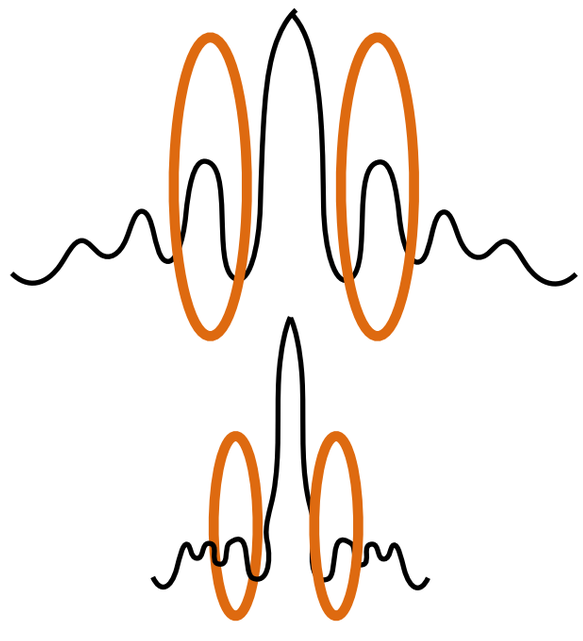


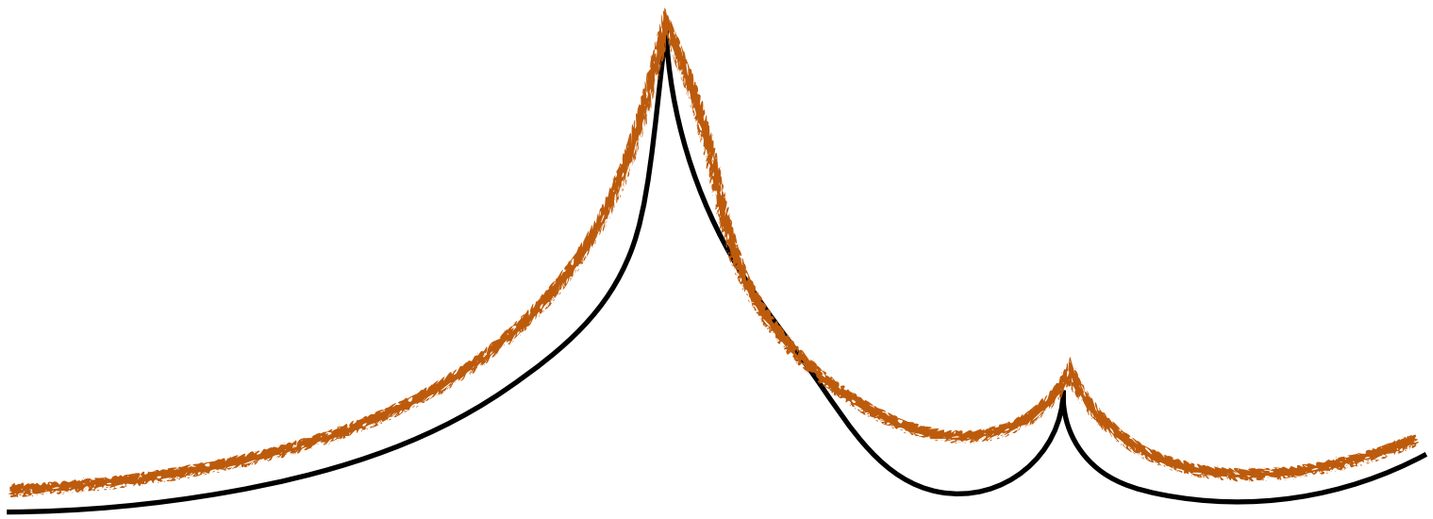
windowing: spectral leakage



windowing: spectral leakage







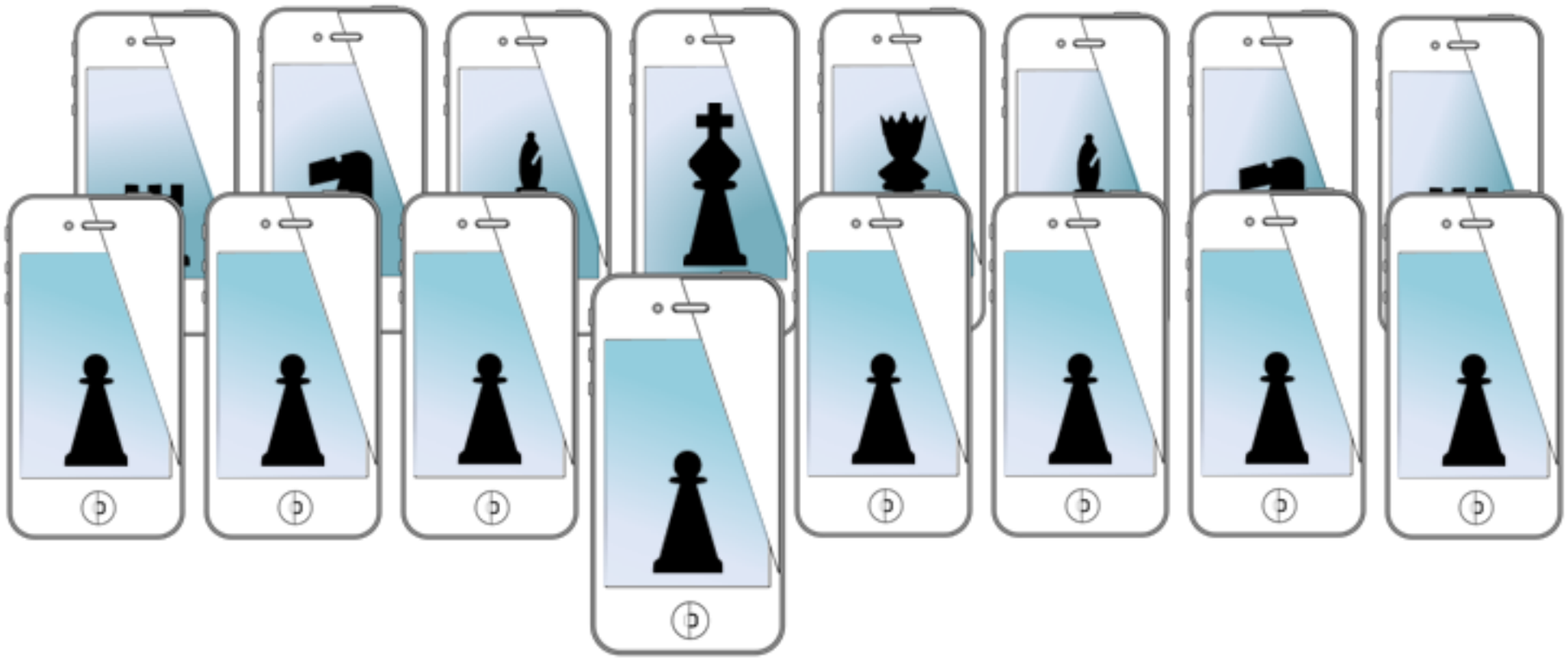
which window to use?

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

FFT review

- sampling rate
  - dictates the time between each sample, (1 / sampling rate)
  - max frequency we can measure is half of sampling rate
- resolution in frequency
  - tradeoff between length of FFT and sampling rate
    - each frequency “bin” is an index in the FFT array
      - each bin represents (Fs / N) Hz
      - what does that mean for 12 Hz accuracy?
- windowing is a result of “convolution” in frequency
  - some windows prevent “leakage” at the cost of frequency resolution

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

Supplemental Slides: filtering and windowing

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