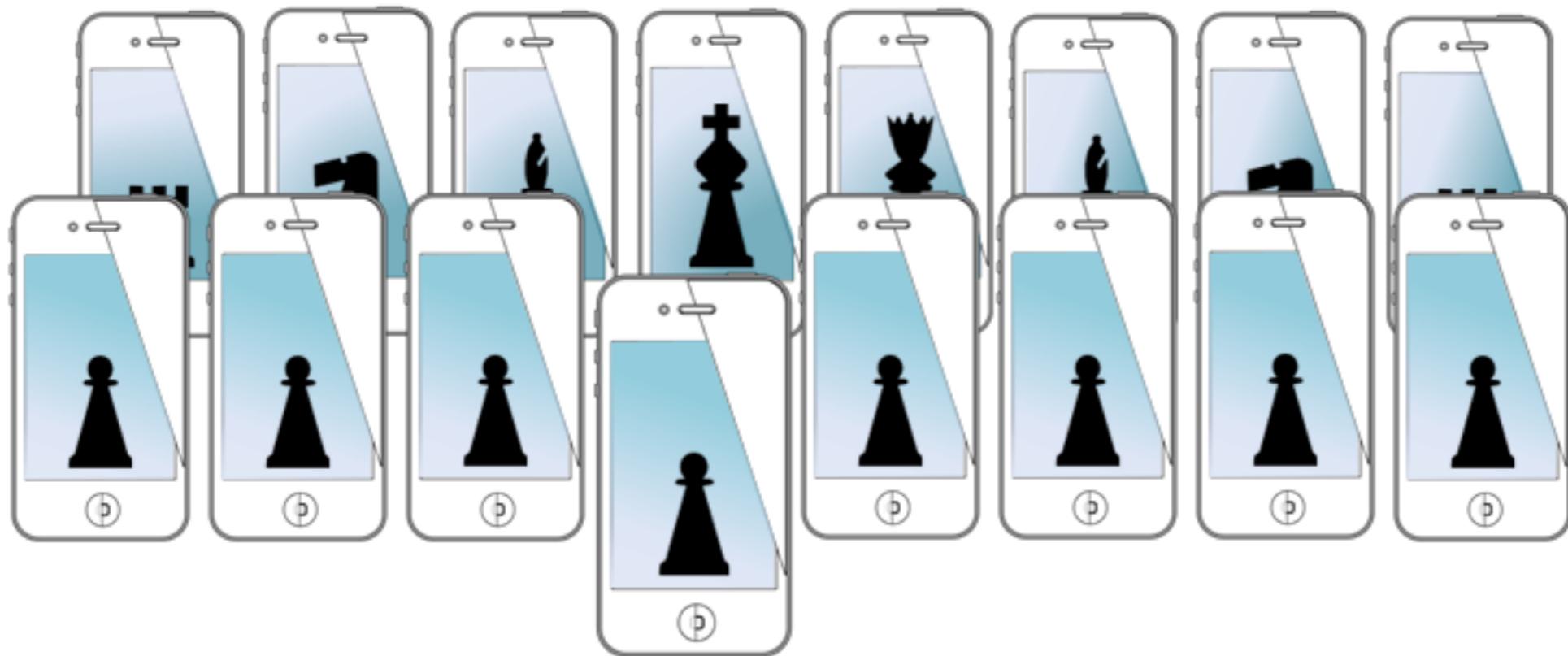


# MOBILE SENSING LEARNING



## CS5323 & 7323

Mobile Sensing and Learning

audio graphing, sampled data, & accelerate

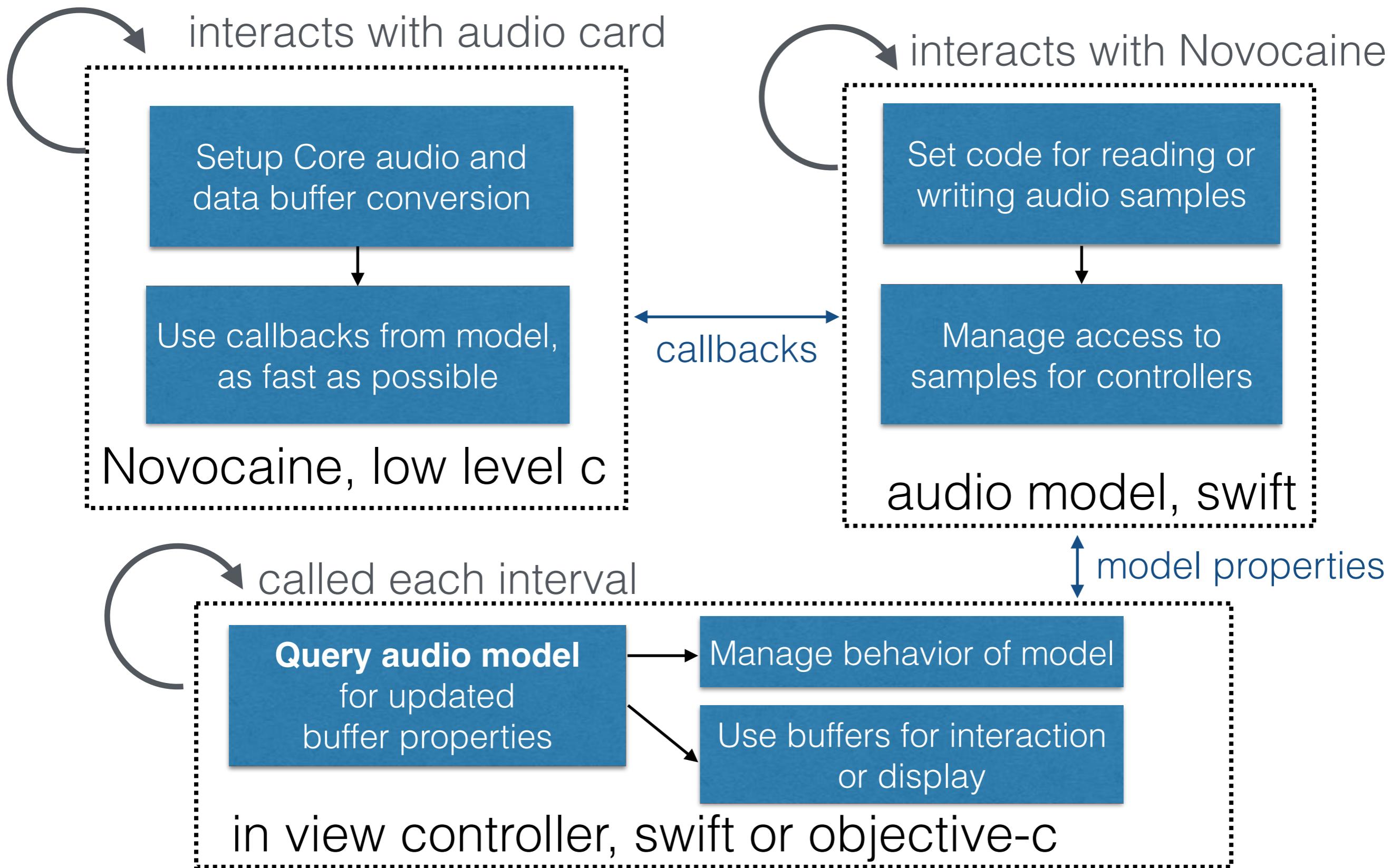
Eric C. Larson, Lyle School of Engineering,  
Computer Science, Southern Methodist University

# agenda and logistics

---

- logistics
  - flipped module on audio next time!
  - TA Office Hours: See canvas
  - **Monday: need to cancel lecture**
- 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

# review: MVC with audio



# sample from the mic

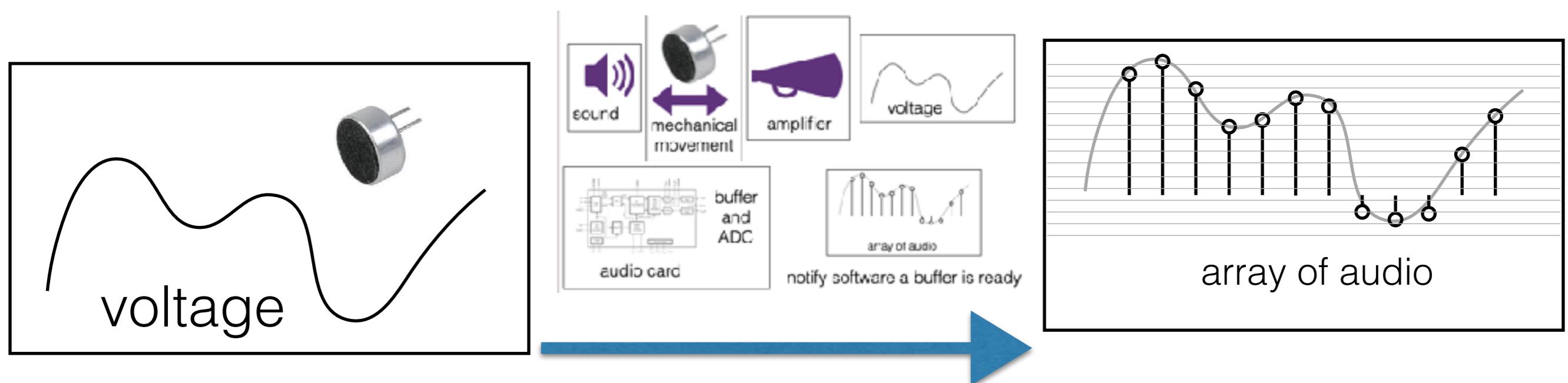
- recall: data from the microphone on novocaine



and now its time for a demo

# intro to sampled data

- physical processes are continuous
  - digitization **may change** how we **understand** the signal
- digitization occurs in time and amplitude
  - time: sampling
  - amplitude: quantization



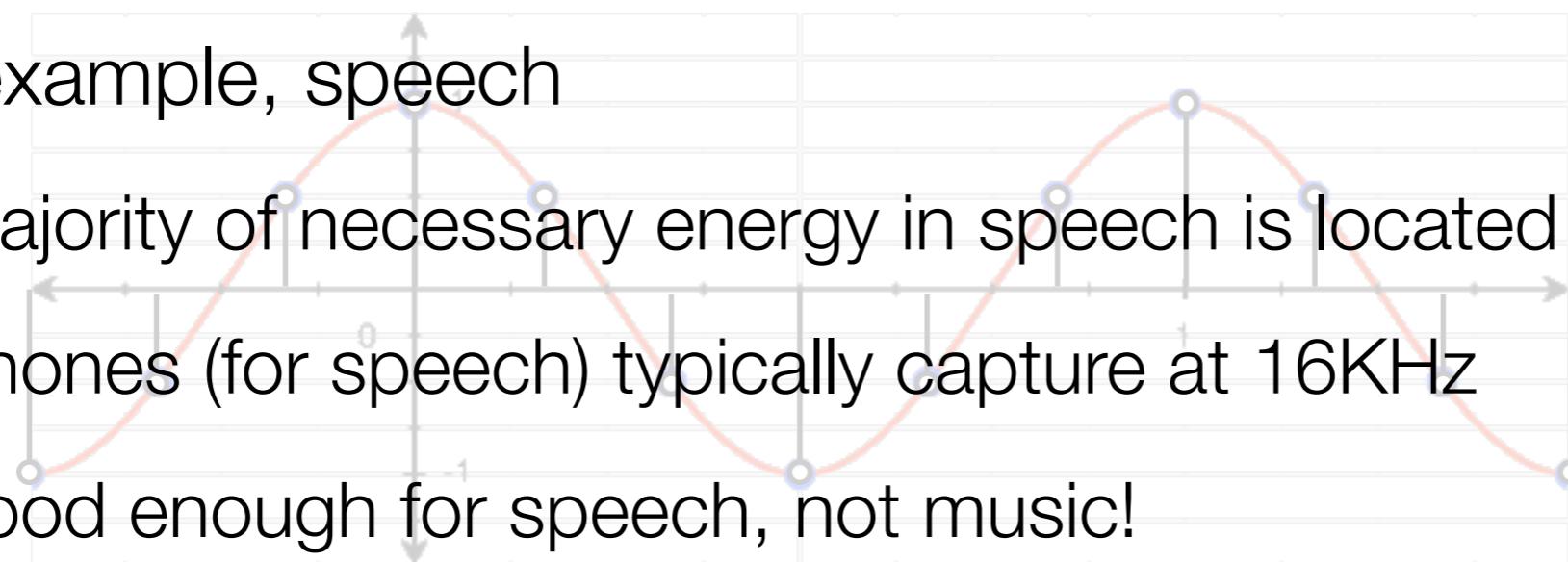
# sampled data

- 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
- 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
- for example, speech
  - majority of necessary energy in speech is located < 8000Hz
  - phones (for speech) typically capture at 16KHz
  - 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

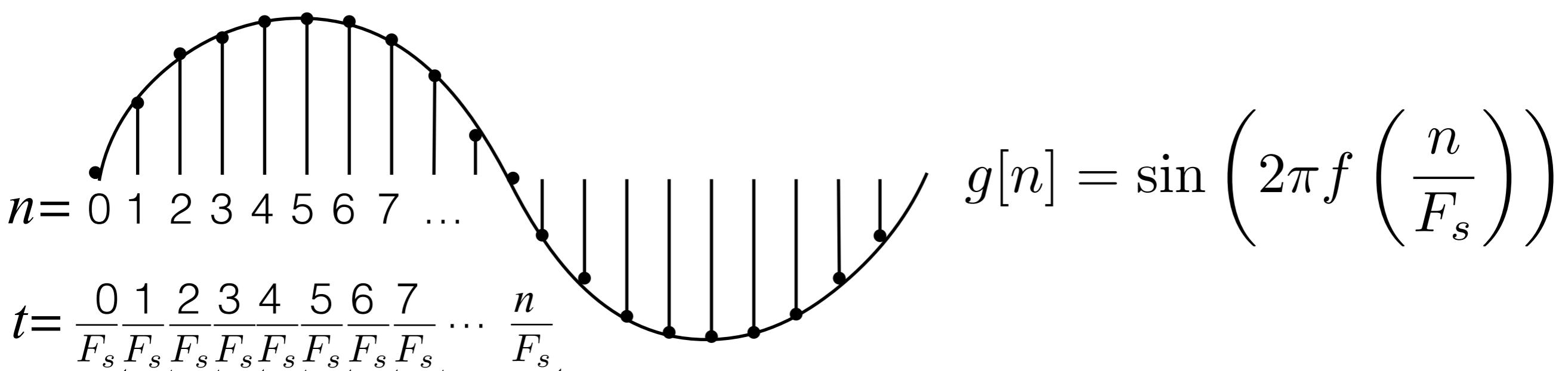
$$g(t) = \sin(2\pi ft)$$

equation for sine wave

$f$ , frequency in Hz

$t$ , time in “seconds”

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



# 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?

should this be initialized inside the audio callback?

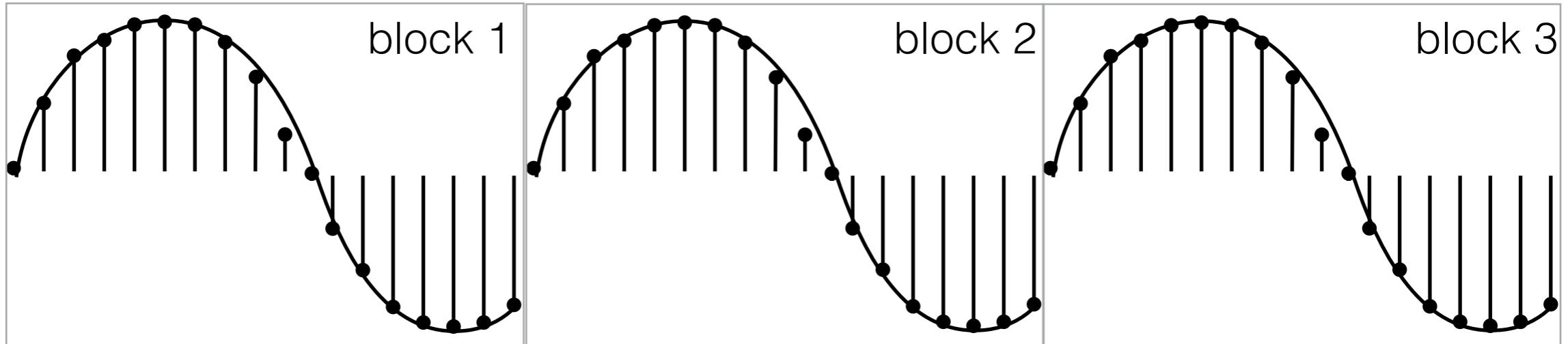
```
float phase = 0.0;
double phaseIncrement = 2*M_PI*frequency/samplingRate;
for (int n=0; n < numFrames; ++n)
{
    data[n] = sin(phase);
    phase += phaseIncrement;
}
```

# sine wave logistics

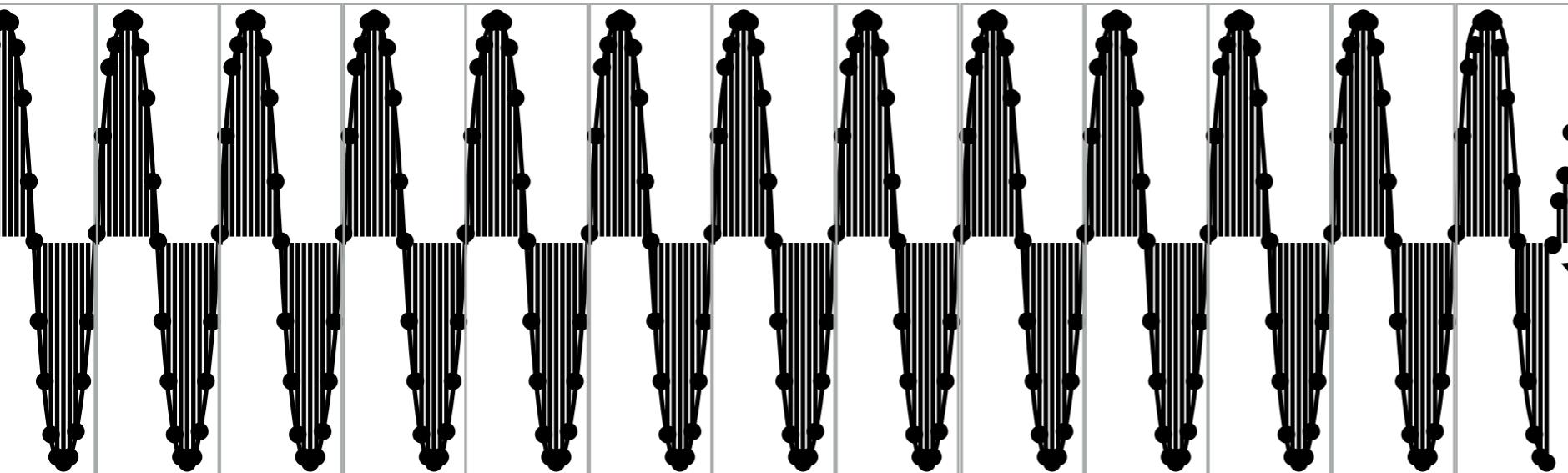
$$g[n] = \sin\left(2\pi f \left(\frac{n}{F_s}\right)\right)$$

$$\theta \leftarrow \theta + \frac{2\pi f}{F_s}$$

$$g[n] = \sin(\theta)$$



phase = 0 in each output block



phase variable  
overflows,  
discontinuity!

# making a sine wave

- bringing it all together

$$\theta \leftarrow \theta + \frac{2\pi f}{F_s} \quad g[n] = \sin(\theta)$$

```
var frequency = 18000.0 //starting frequency
var phase = 0.0
var samplingRate = audioManager.samplingRate
let sineWaveRepeatMax = 2*Double.pi

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

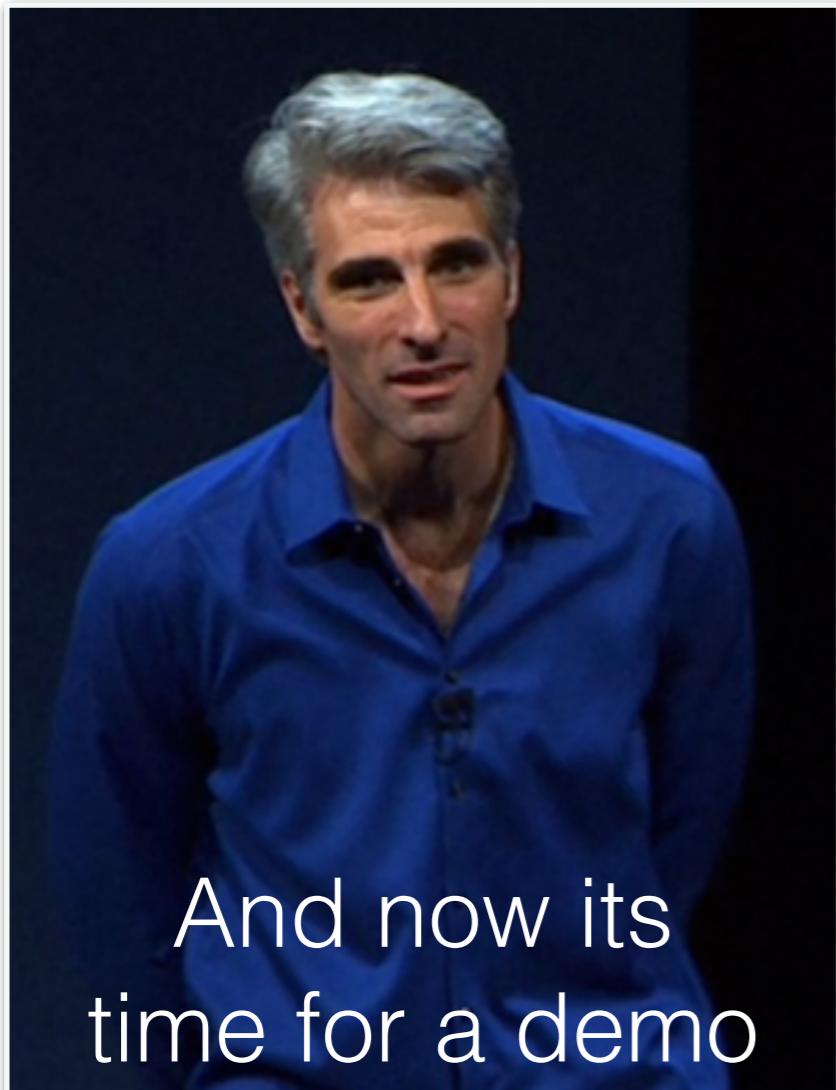
what if more than one channel?

what if the frequency changes?

data: <Optional>UnsafeMutablePointer<Float>

# play samples to speakers

- demo, play sine wave



And now its  
time for a demo



# the accelerate framework

- very powerful digital signal processing (DSP) library
  - look at vDSP Programming Guide on [developer.apple.com](https://developer.apple.com) for the complete API
- provides mathematics for performing fast DSP
  - fast is the name of the game – it uses “c”
    - SIMD single Instruction, Multiple Data

```
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
);
```

[https://developer.apple.com/documentation/accelerate/1450020-vdsp\\_vsmul](https://developer.apple.com/documentation/accelerate/1450020-vdsp_vsmul)

# examples

what do each of these implement?

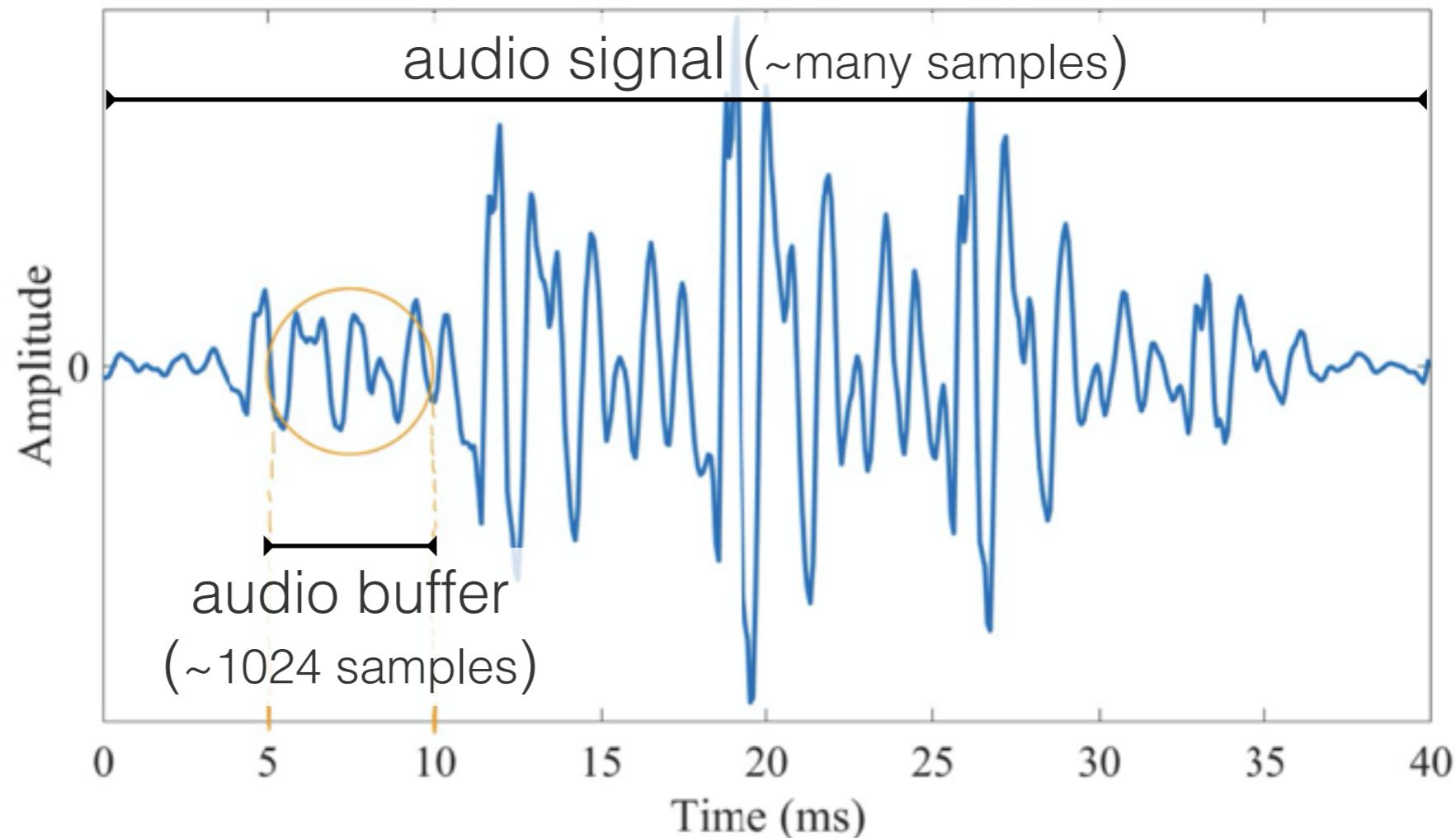
```
outputBlockFunction(data:(...),numFrames:(UInt32),numChannels:(UInt32)) {  
    ringBuffer.fetchFreshData(data, withNumFrames:numFrames)  
    var volume = userSetMultiplyFromSlider;  
    vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels)  
}
```

```
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, etc.



# how much data to show?

- sampling at 48kHz == 48000 samples per second



graph 0.5 second  
window is:  
24000 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 <a href="#">multiplayer online battle arena</a> game <a href="#">Vainglory</a> to demonstrate Metal's graphics capabilities at the <a href="#">iPhone 6's September 2014 announcement event</a> <sup>[1]</sup> |
| <a href="#">Developer(s)</a>     | Apple Inc.   |
| <a href="#">Initial release</a>  | June 2014; 6 years ago   |
| <a href="#">Stable release</a>   | 3 / June 2019; 1 year ago  |
| <a href="#">Written in</a>       | Shading Language: <a href="#">C++14</a> , Runtime/API: <a href="#">Objective-C</a>   |
| <a href="#">Operating system</a> | iOS, iPadOS, macOS, tvOS   |
| <a href="#">Type</a>             | 3D graphics and compute API  |
| <a href="#">License</a>          | proprietary  |
| <a href="#">Website</a>          | <a href="https://developer.apple.com/metal/">developer.apple.com/metal/</a> ↗  |

# the MetalGraph class

ViewController.swift  
MetalGraph.swift  
Shaders.metal

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  
normalize: (default) assume **data is between -1.0 to 1.0**

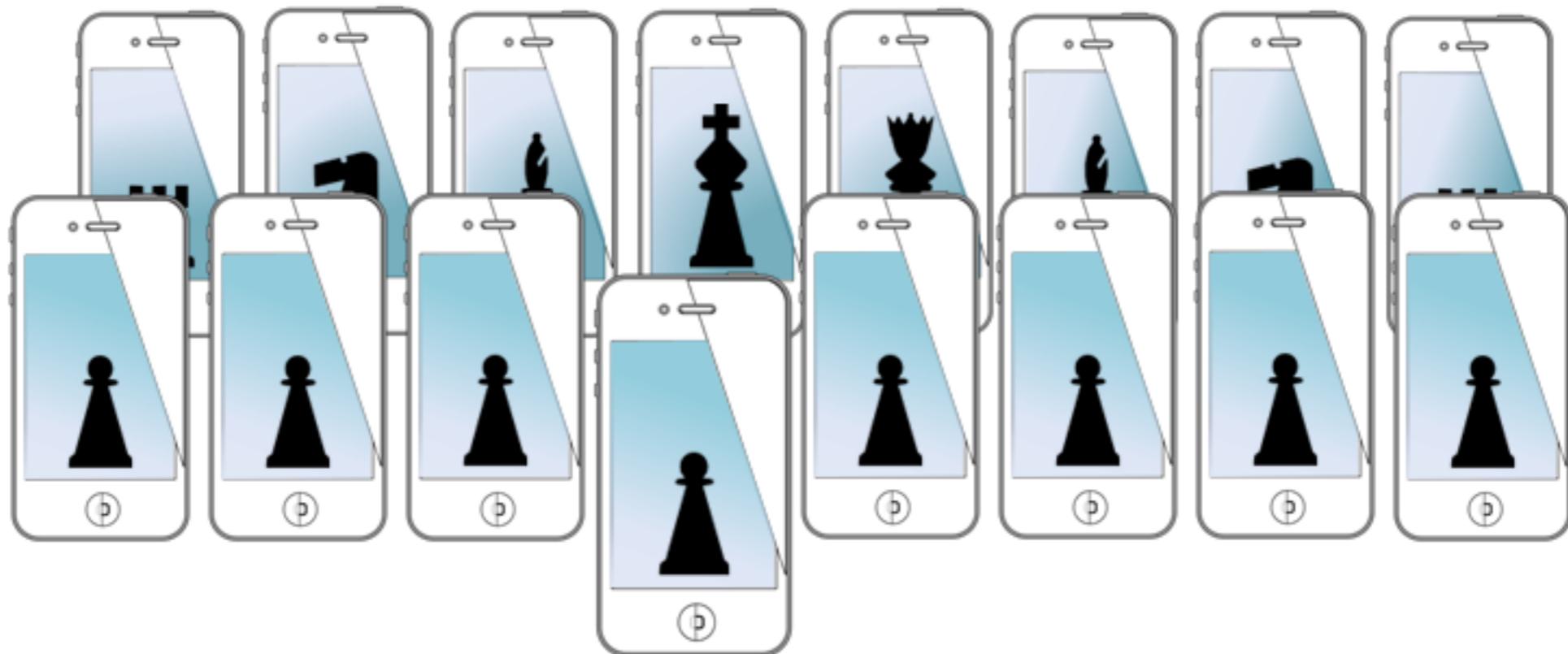
# audio graphing demo!



and now its time for a demo

- if time: (1) add another graph
- (2) preview of FFT graphing

# MOBILE SENSING LEARNING



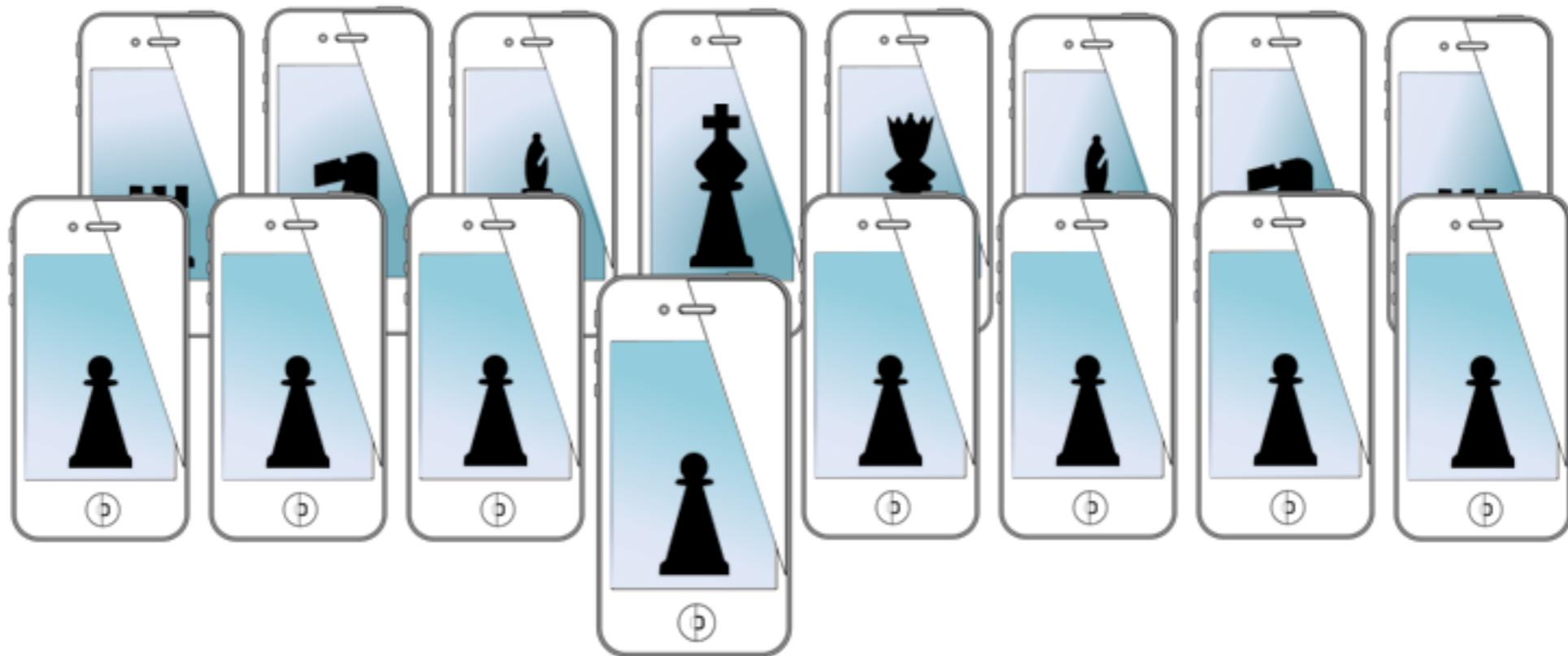
## CS5323 & 7323

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**

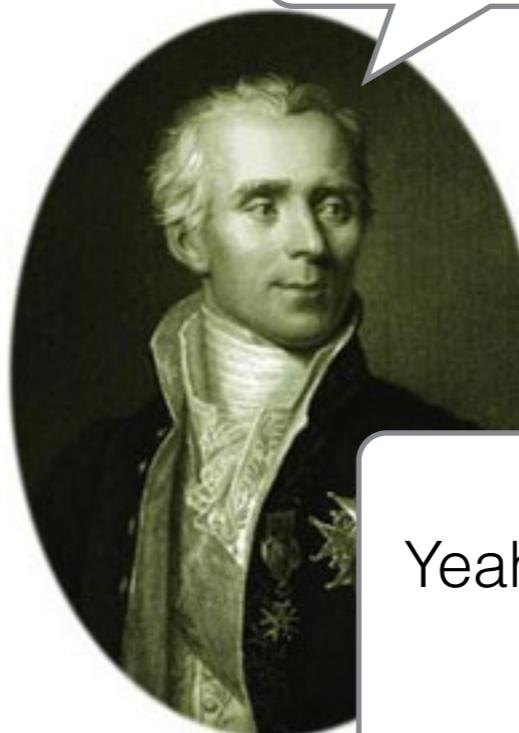
- for now, we're going to stick with analysis
  - specifically, the **Fourier Transform**

# Fourier



A mathematician  
Worked on  
Submitted  
in 1822

Did you read that  
Fourier paper?



Laplace



Lagrange

Yeah! I was like...  
*hated it*

# Fourier



A math  
Worker  
Subm  
in 18

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



Laplace



Lagrange

Simpler functions!? Like  
sine waves? Come on!

# Fourier



I created the FFT to commit espionage against the Russians!

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

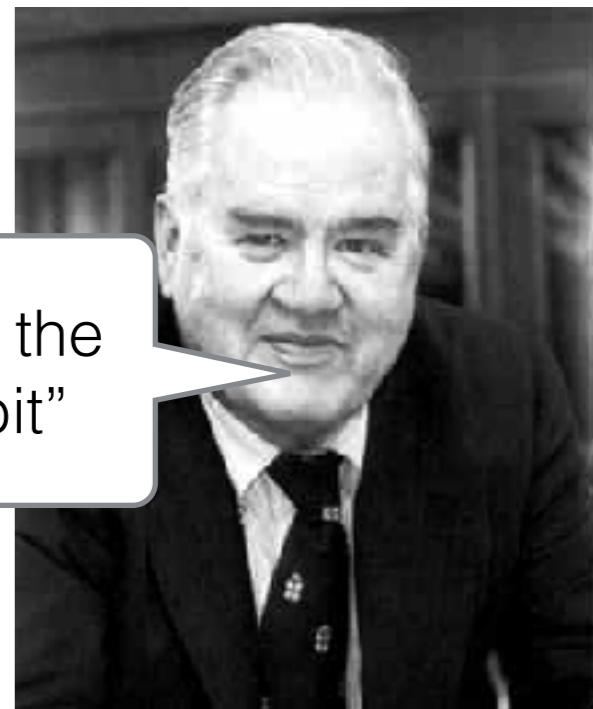
The transform was eventually called The Fourier Transform

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



Cooley

I coined the term “bit”



Tukey

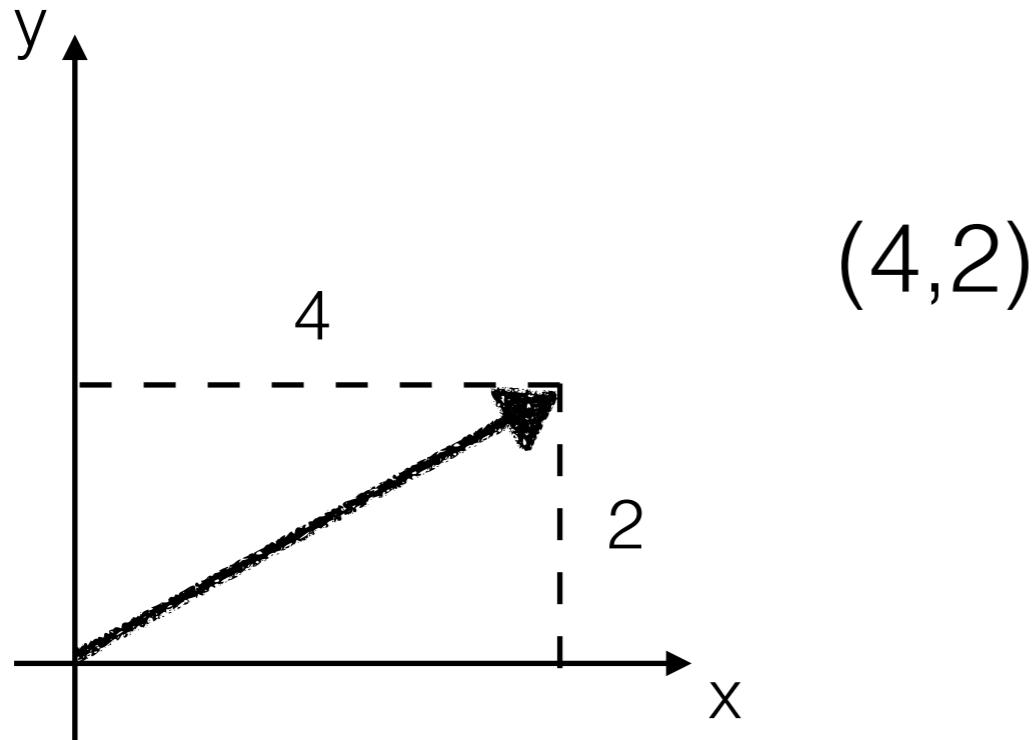
# the Fourier transform

---

- 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)



where did these numbers come from?

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

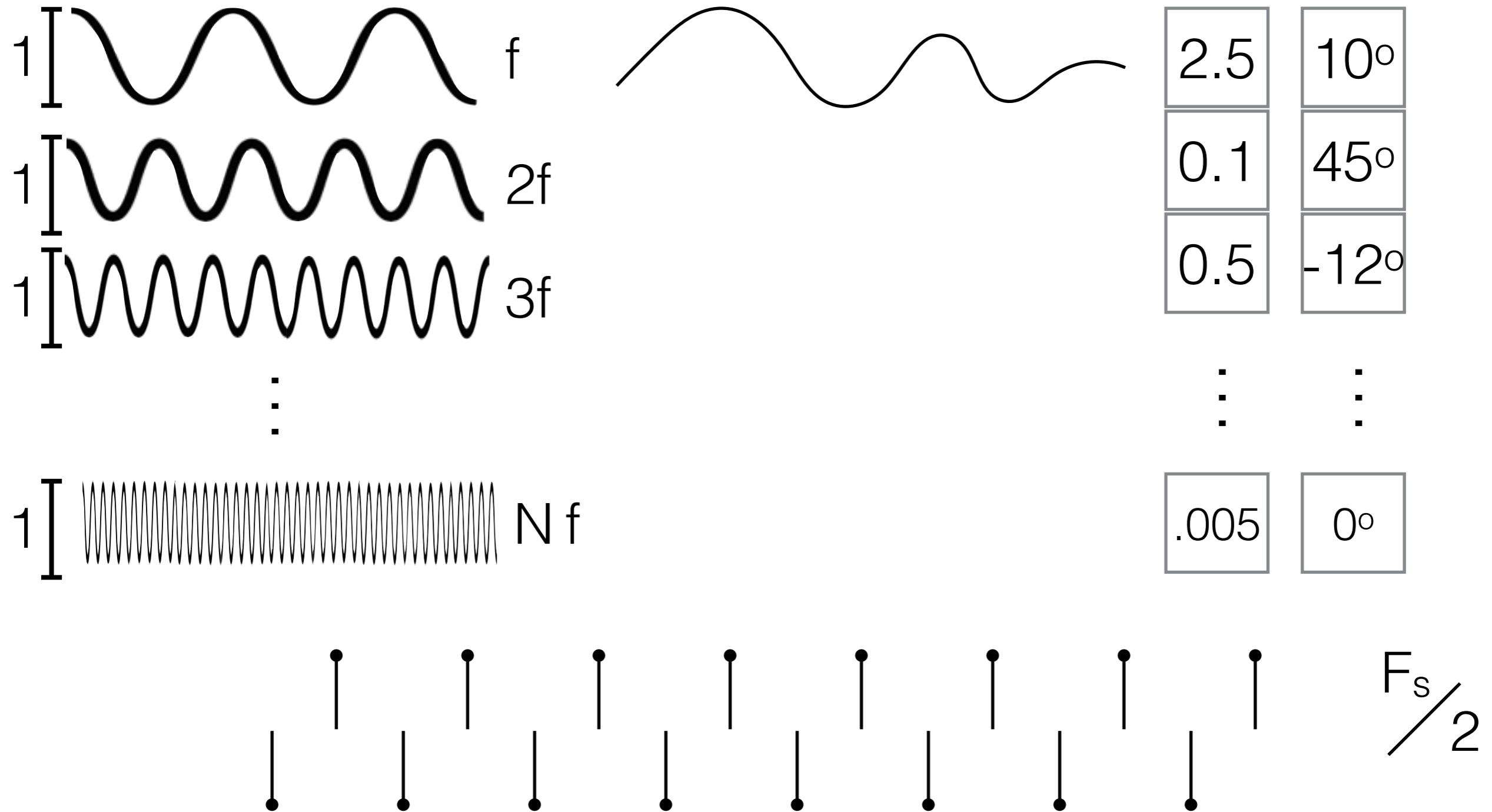
point by point multiplication and add it up!

tells us “how much” of the vector is the result of these vectors

$$\vec{x} \quad \vec{x} \bullet \vec{y} = 0$$
$$\vec{y} \quad |\vec{x}| = |\vec{y}| = 1$$

# FT by sine waves

what if the orthogonal vectors were functions?



# FT by the numbers

frequency content

|     |        |        |          |                               |
|-----|--------|--------|----------|-------------------------------|
| 0f  | [0.7]  | [0°]   | =        | 0.7                           |
| 1f  | [2.5]  | [10°]  | + [2.5]  | -1f ~ 2.5 cos(2pi (f) t+10°)  |
| 2f  | [0.1]  | [45°]  | + [0.1]  | -2f ~ 0.1 cos(2pi (2f) t+45°) |
| 3f  | [0.5]  | [-12°] | + [0.5]  | -3f ~ 0.5 cos(2pi (3f) t-12°) |
| :   | :      |        | :        |                               |
| N f | [.005] | [0°]   | + [.005] | -N f ~ 0.005 cos(2pi (Nf) t)  |

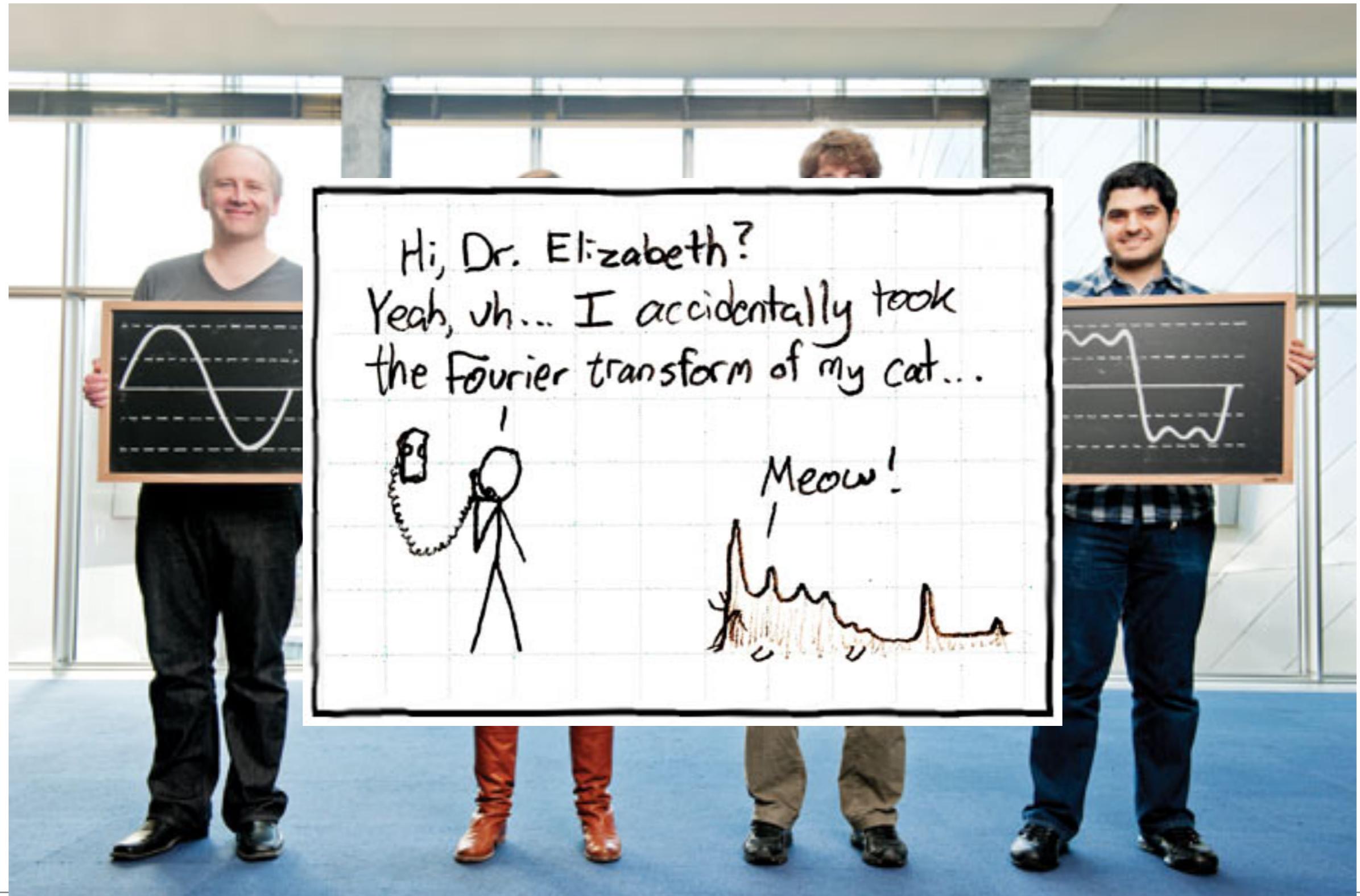
# FT by video

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



# FT by hilarity

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

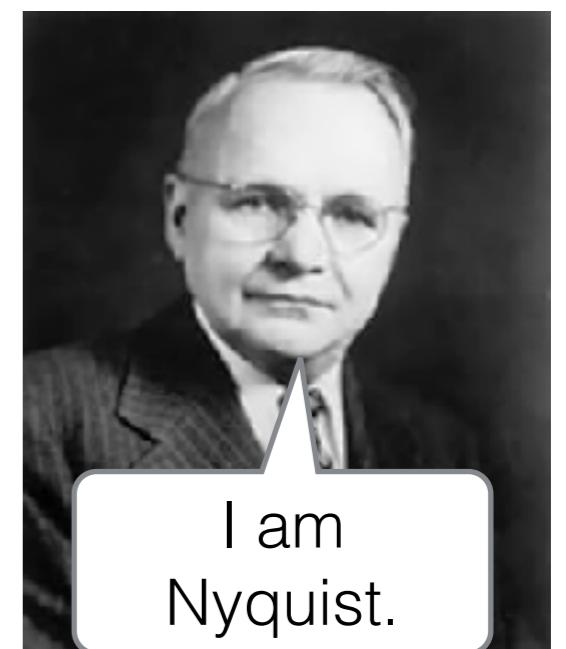
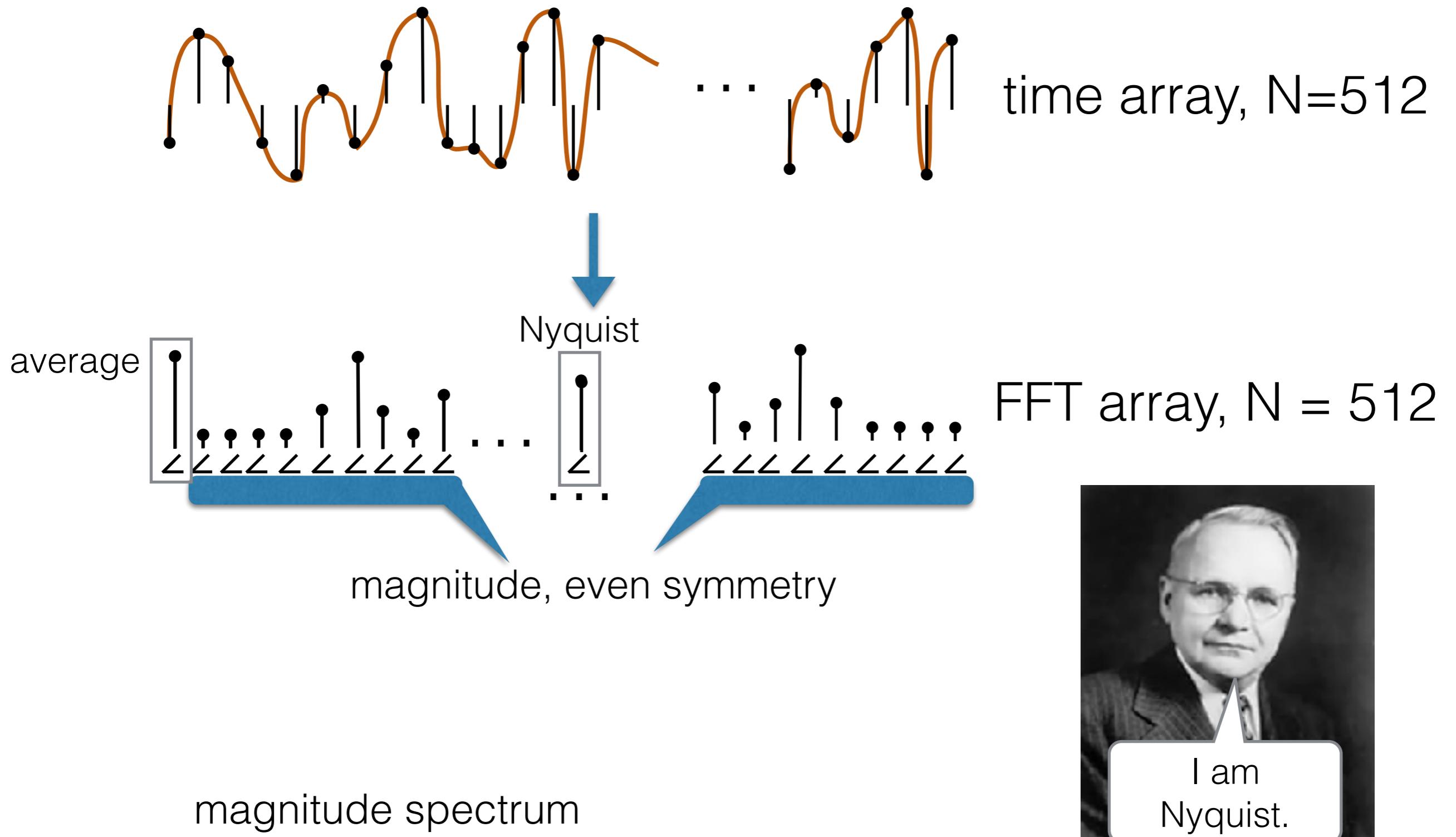


# the FFT

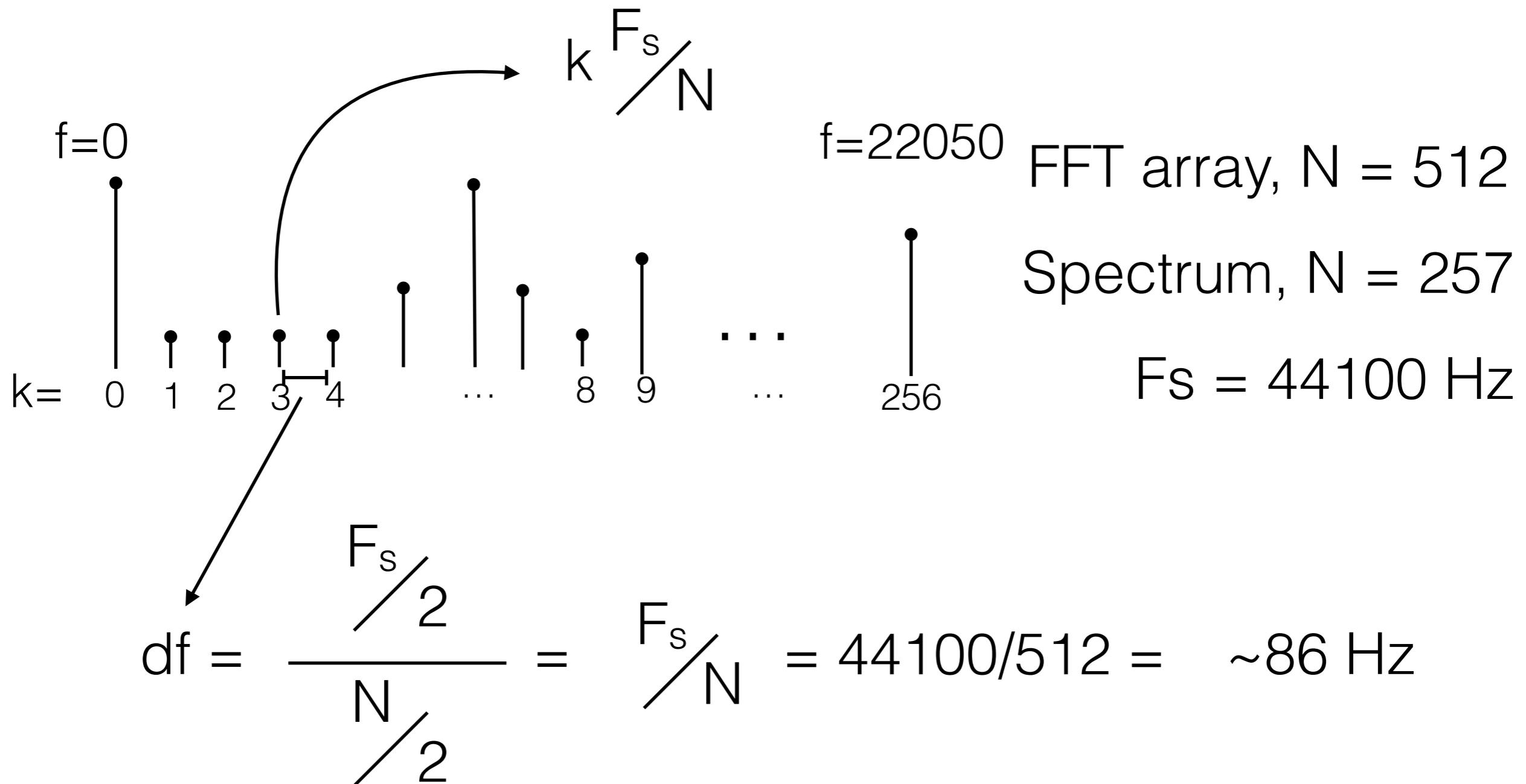
---

- 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

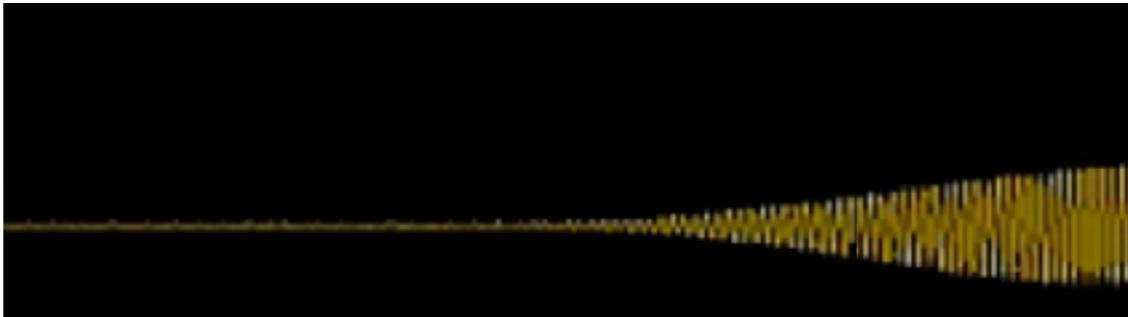


# time and frequency

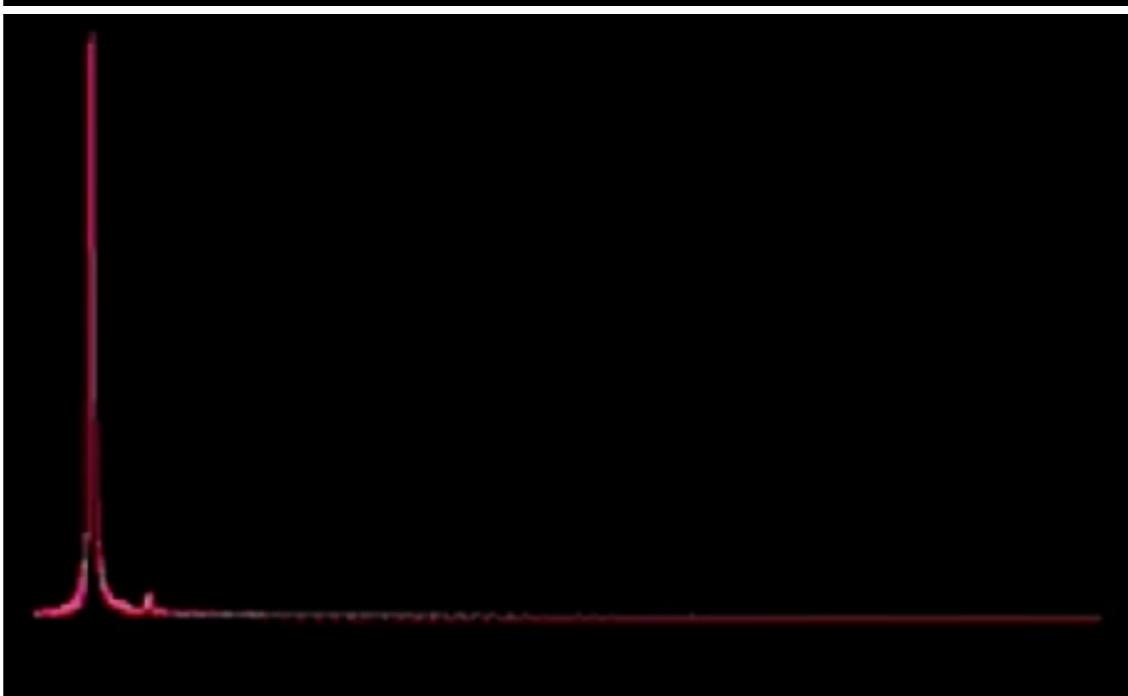


# using the FFT

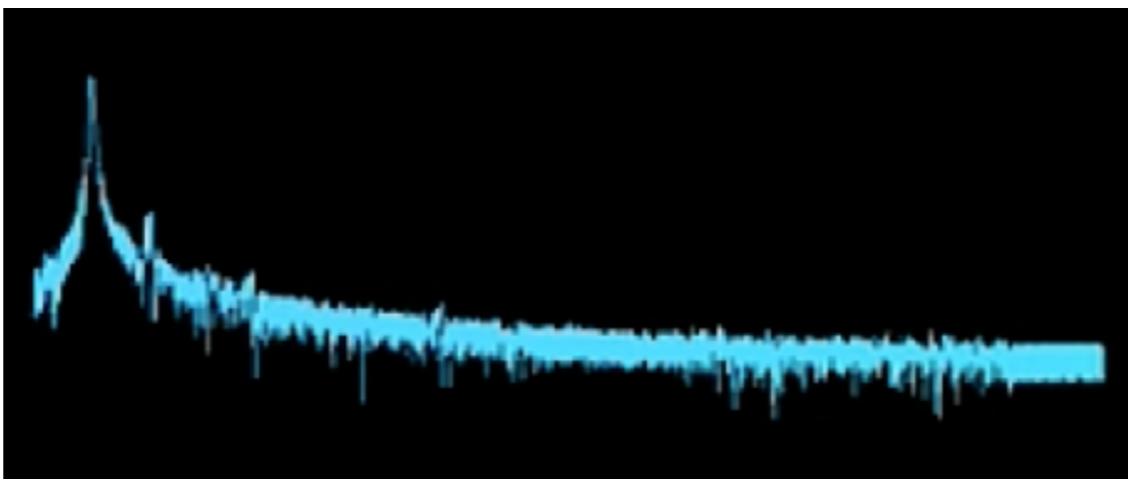
raw audio



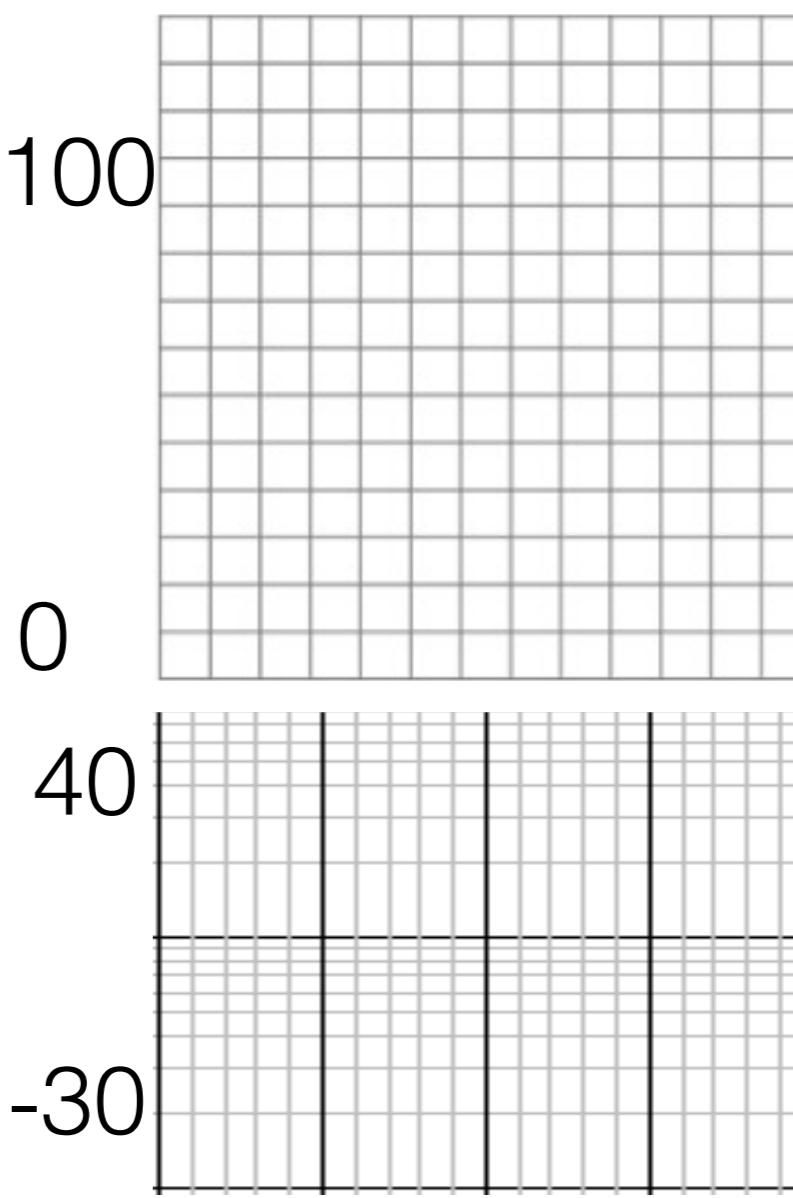
magnitude FFT



magnitude FFT  
in dB



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



# some fft examples

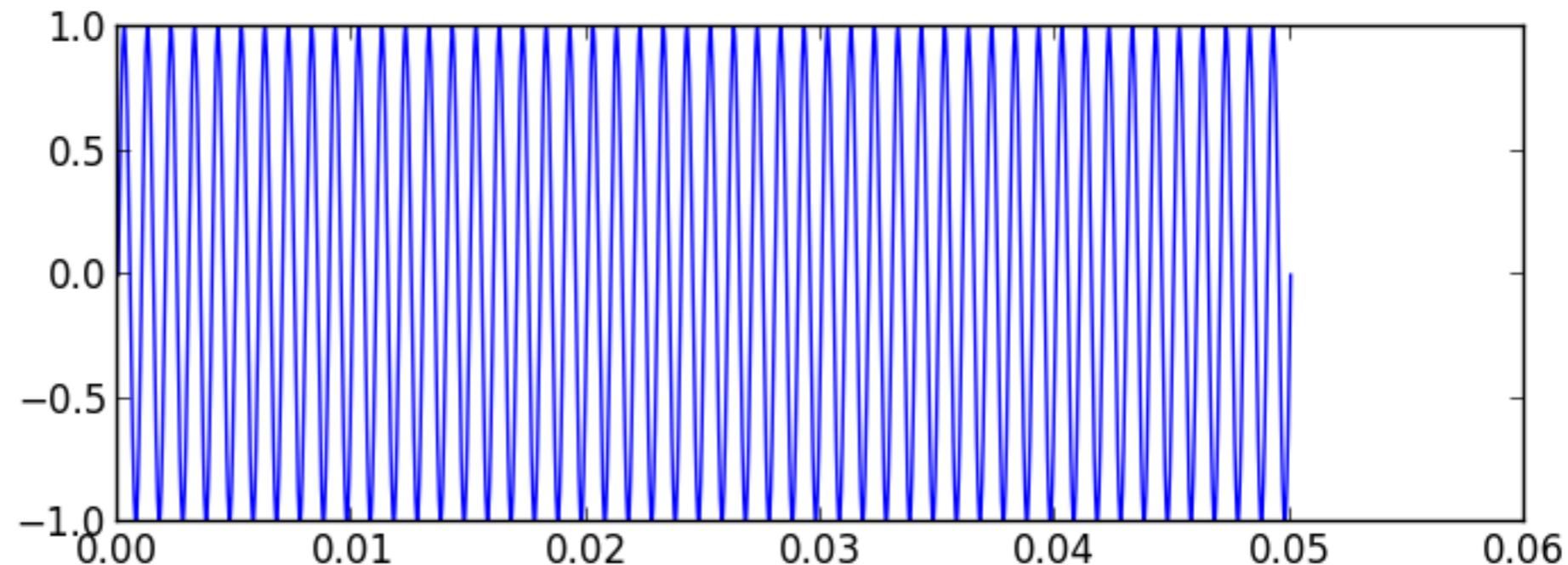
this is just a plot of  
an array of values  
sampled

**over equal time  
intervals**

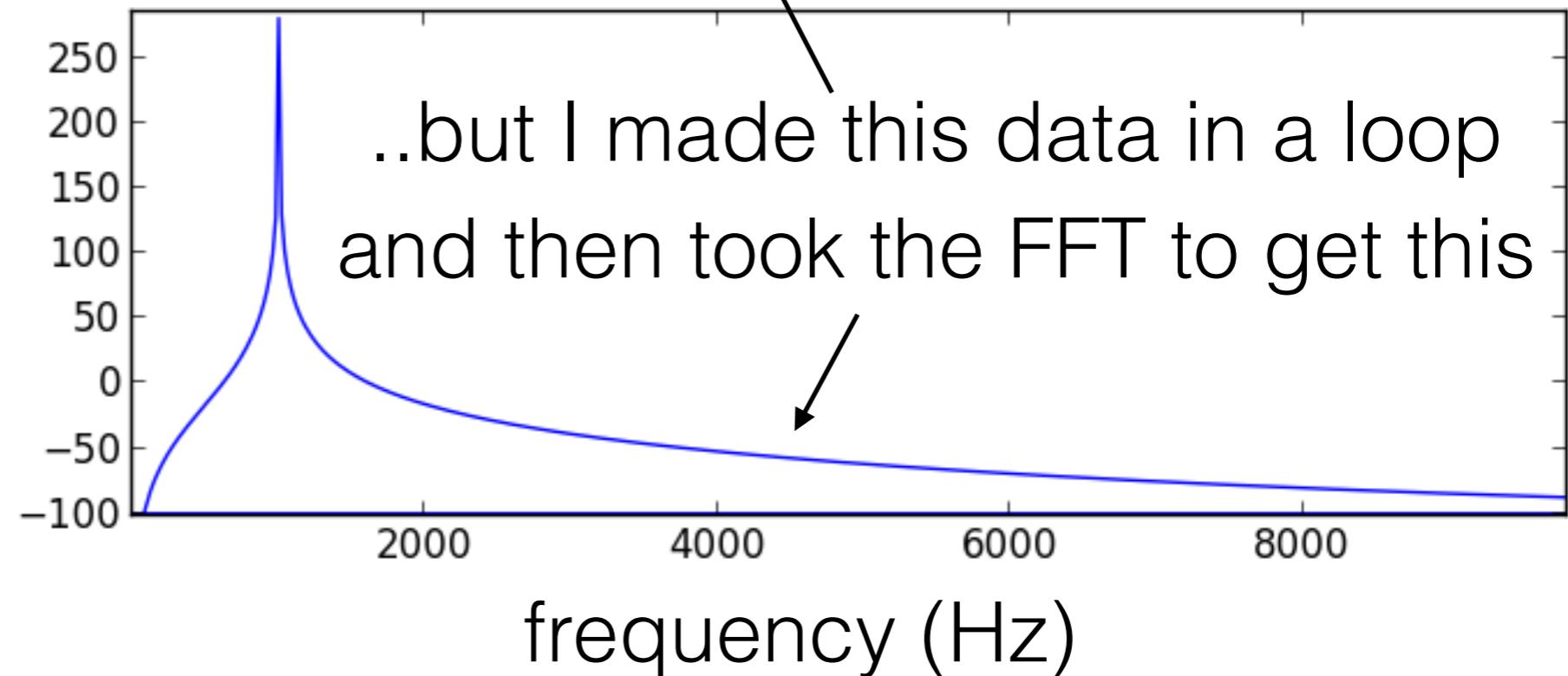
this is just a plot of  
an array of FFT  
values

**its indices are  
equally sampled  
over frequency**

1kHz sine wave

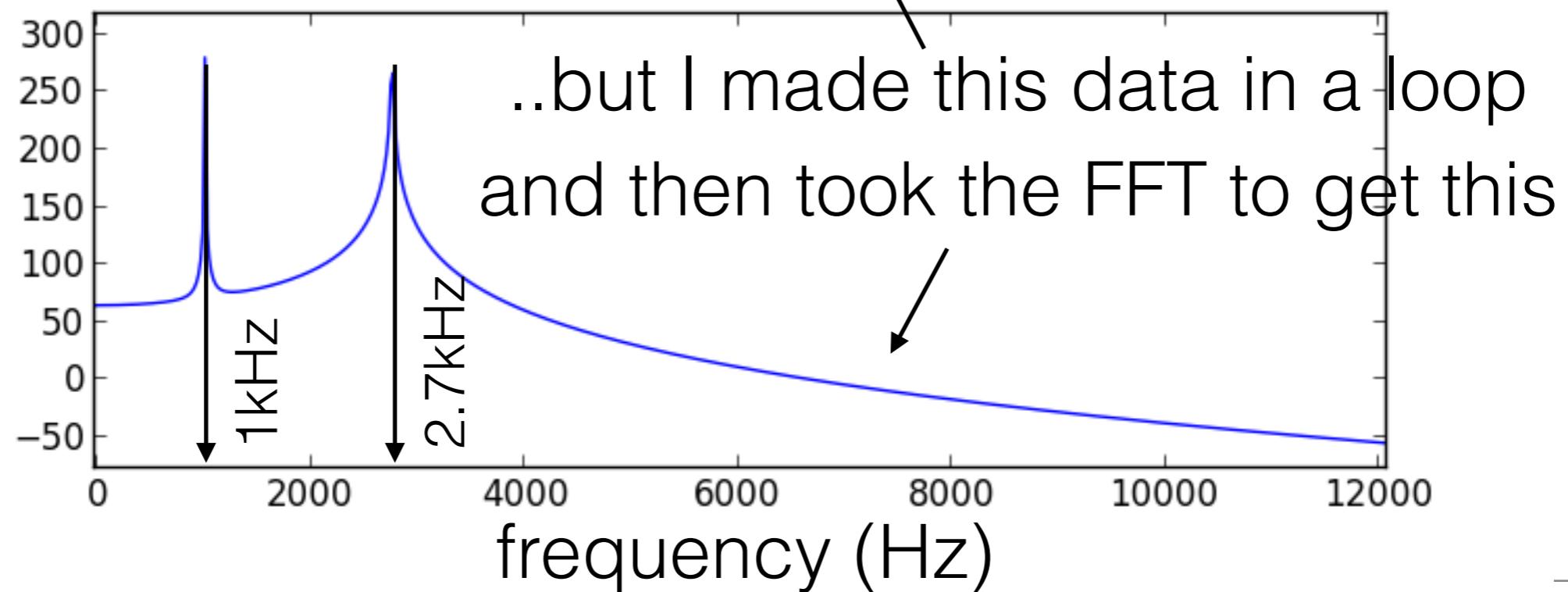
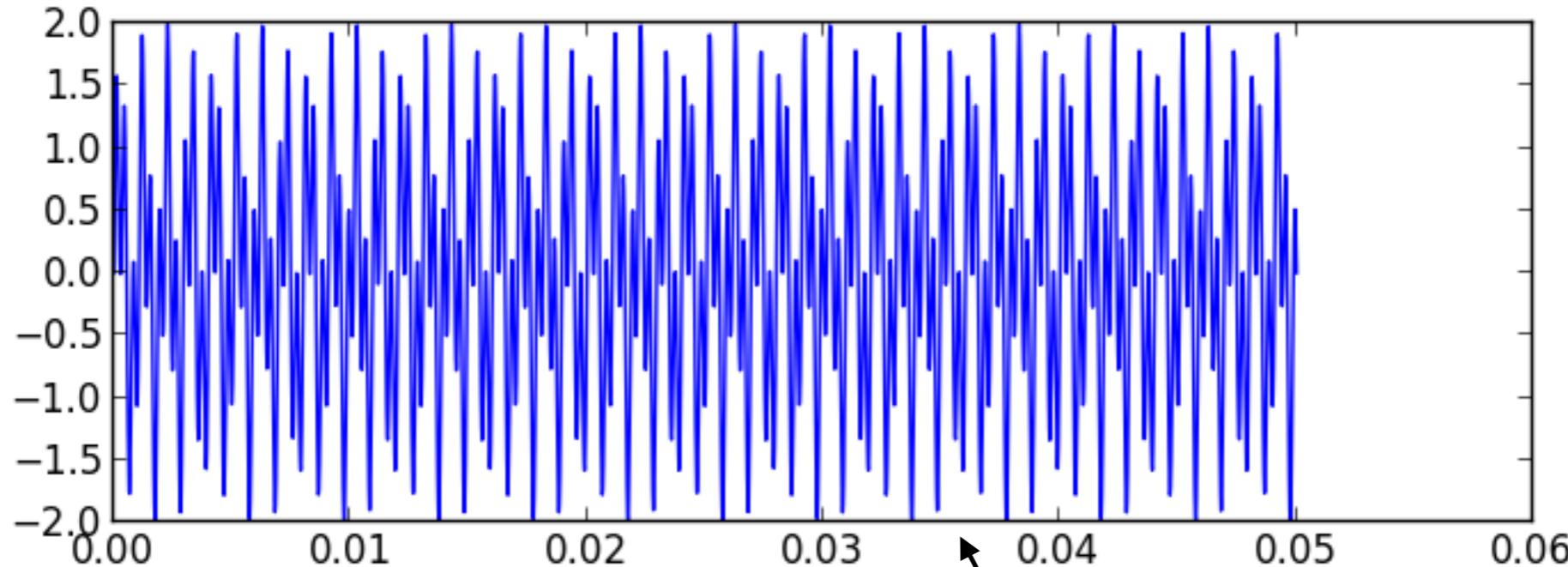


..but I made this data in a loop  
and then took the FFT to get this



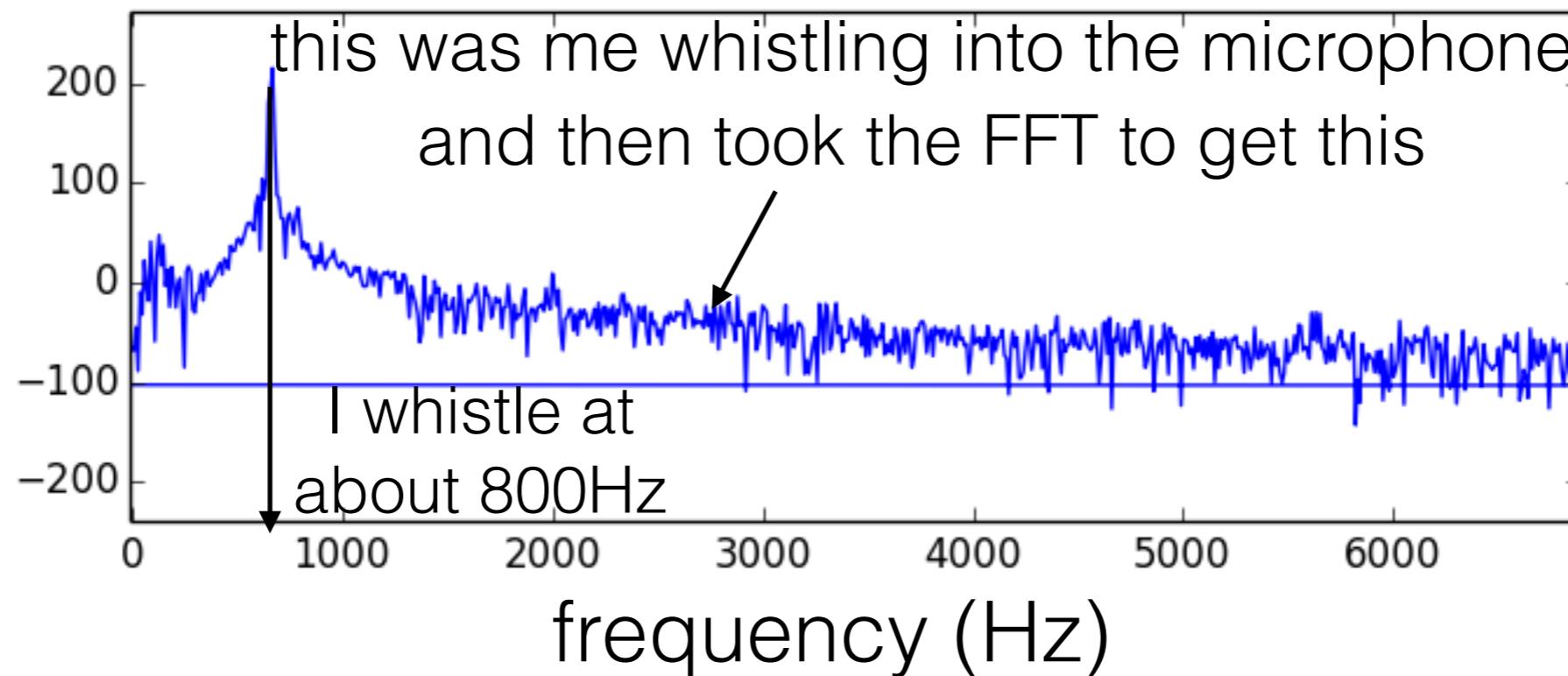
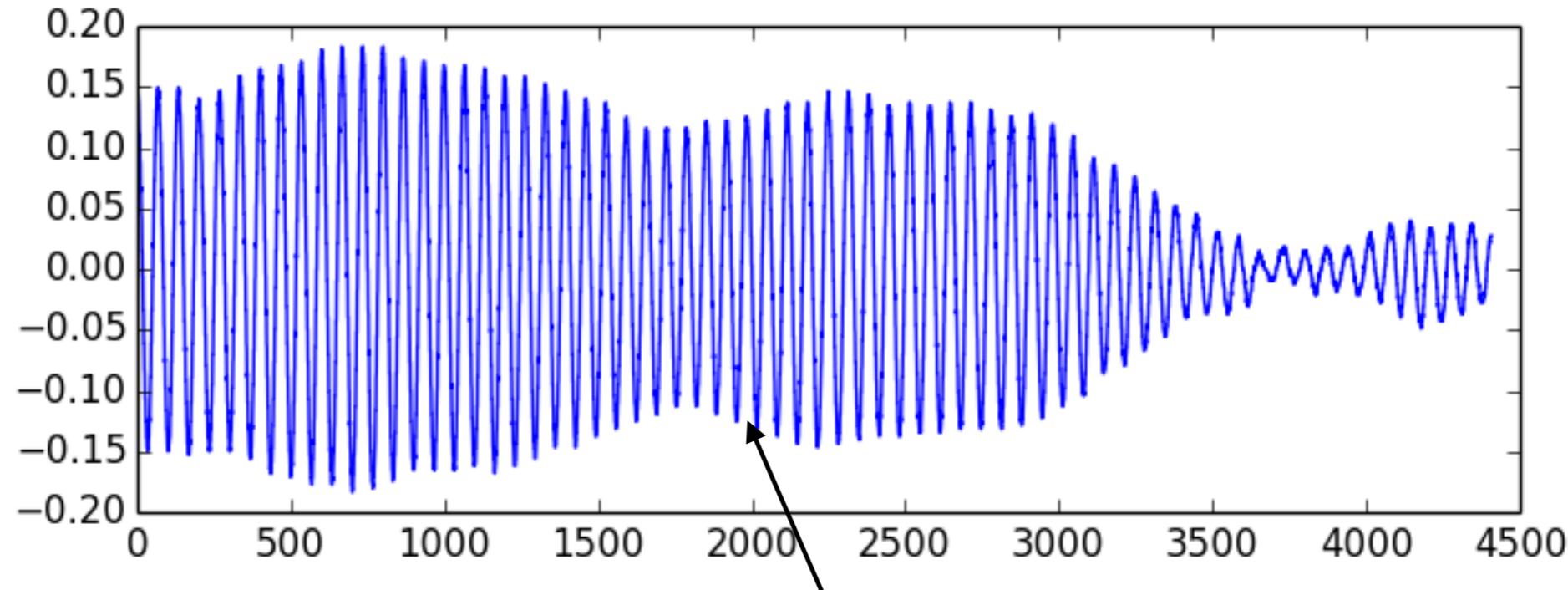
# some fft examples

1kHz sine wave + 2.7kHz sine wave



# some fft examples

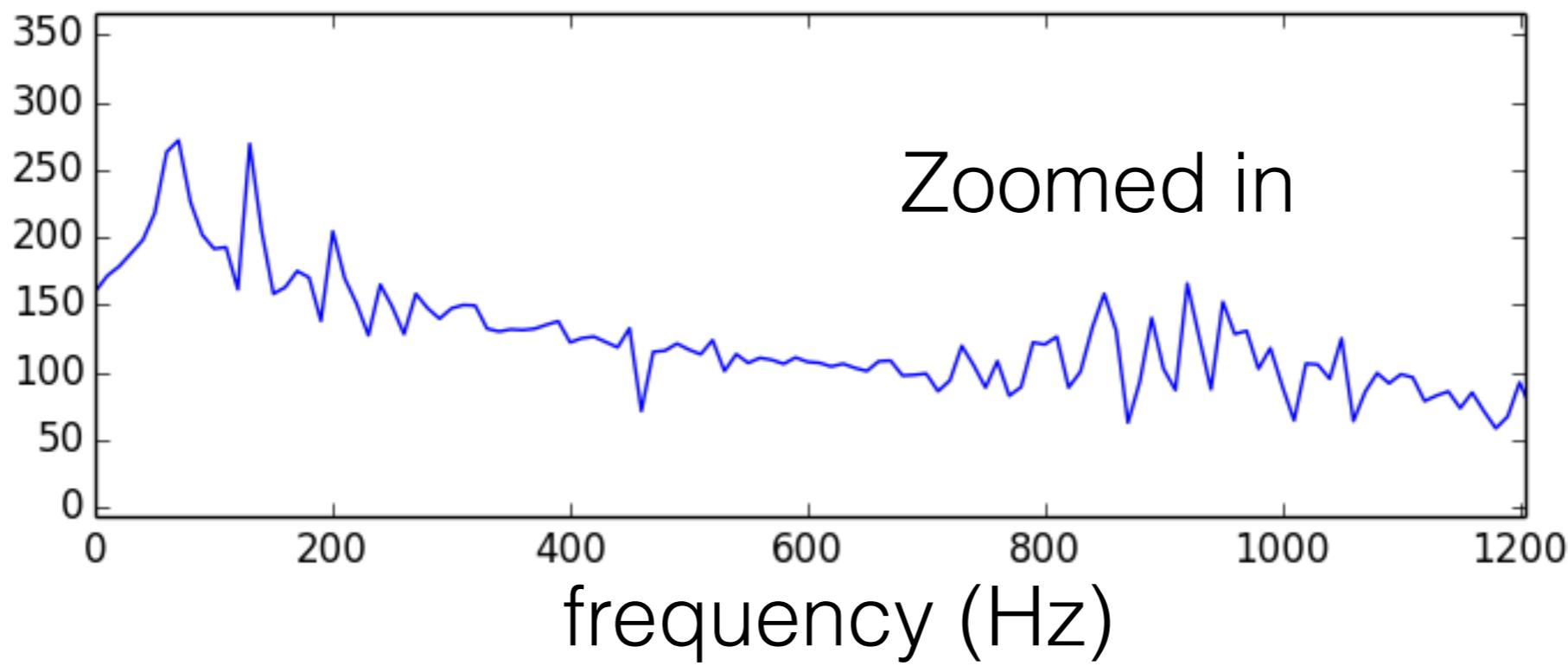
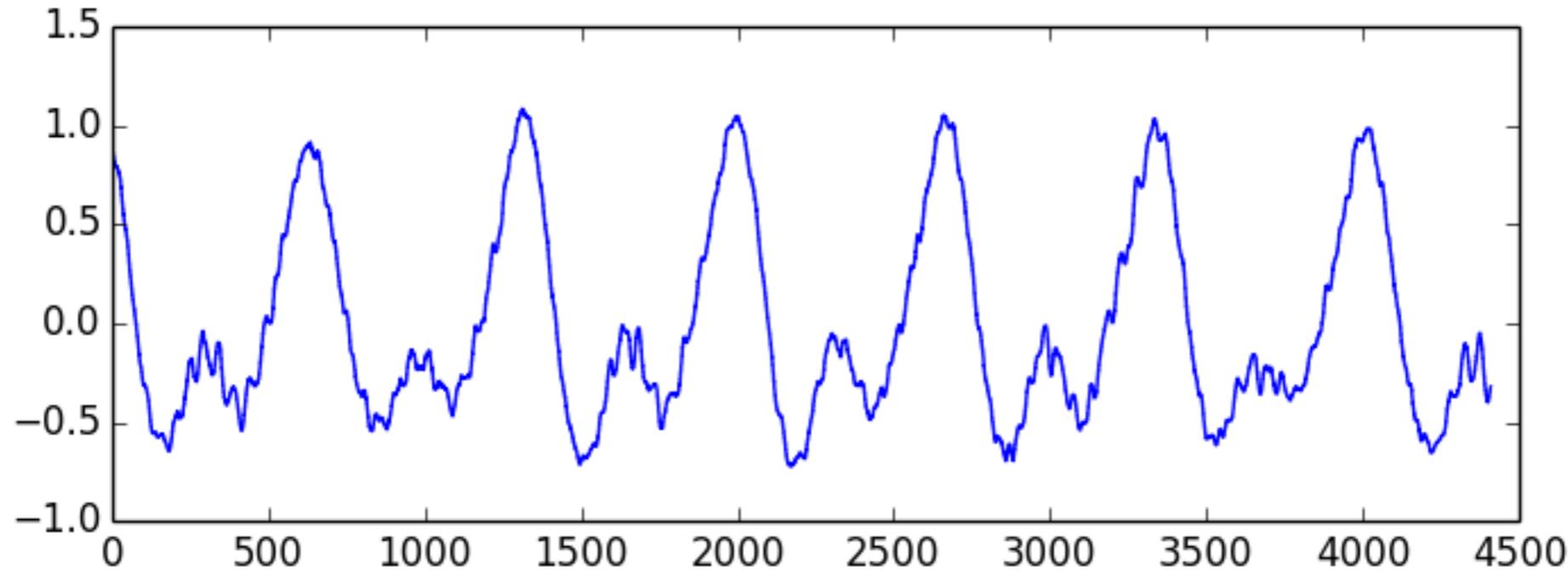
a person whistling



# some fft examples

humming

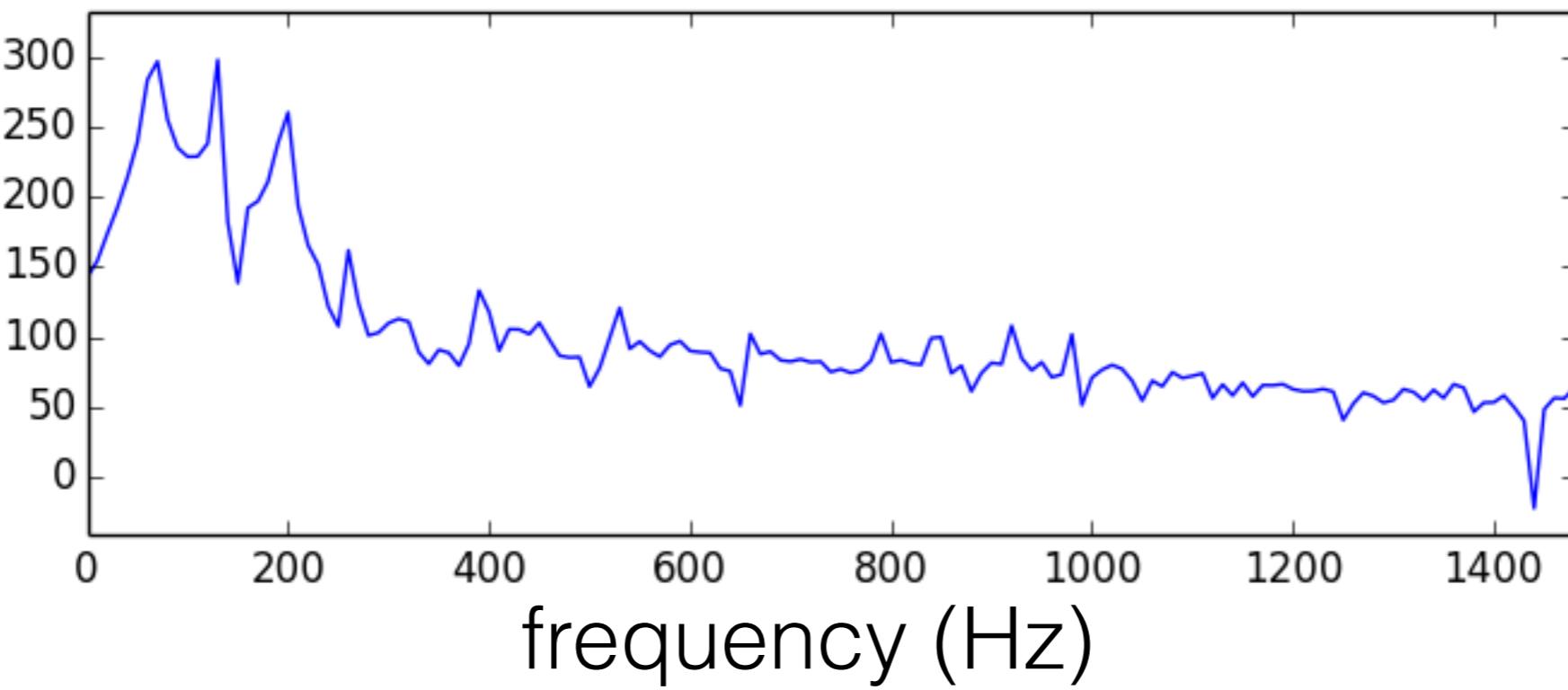
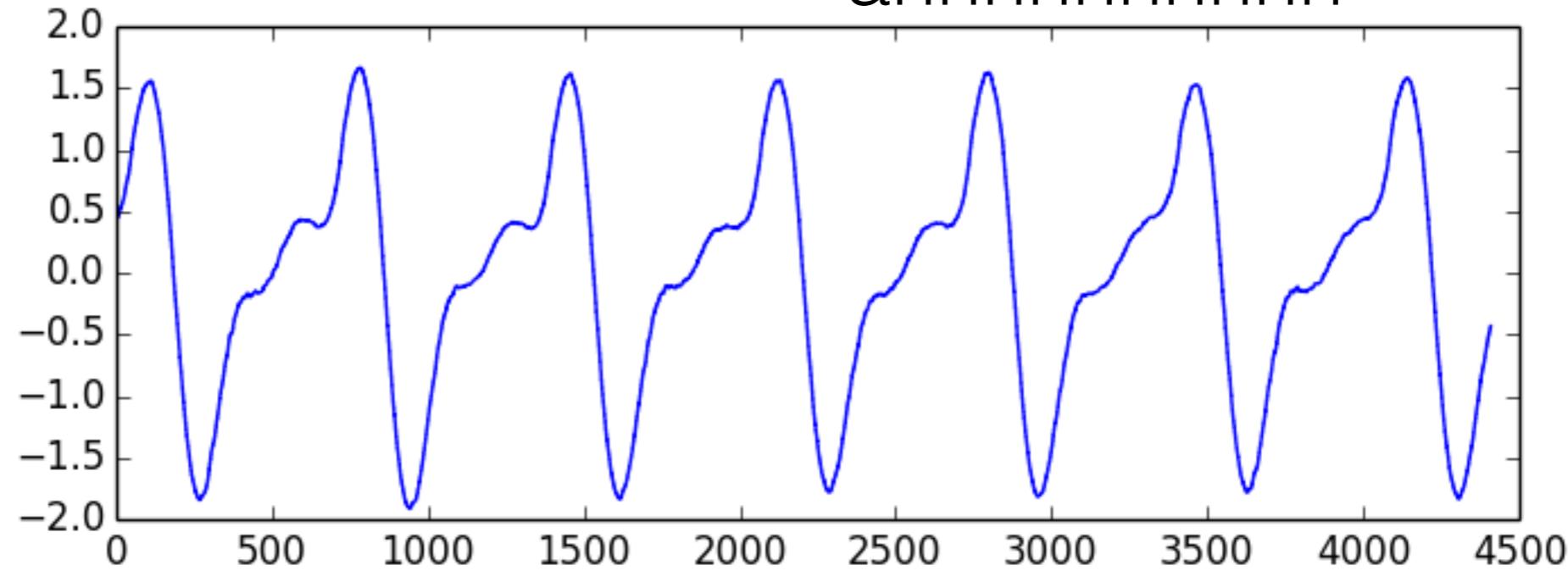
oooooooooooooo



# some fft examples

humming again

ahhhhhhhhhh

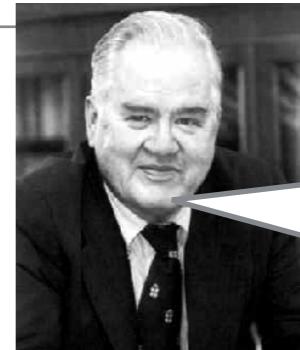


frequency (Hz)



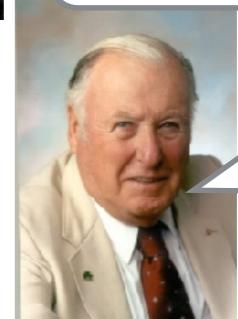
# programming the FFT

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



This is how to use our stuff

fft size == buffer size



Yeah!

```
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;
```

fft size == window size

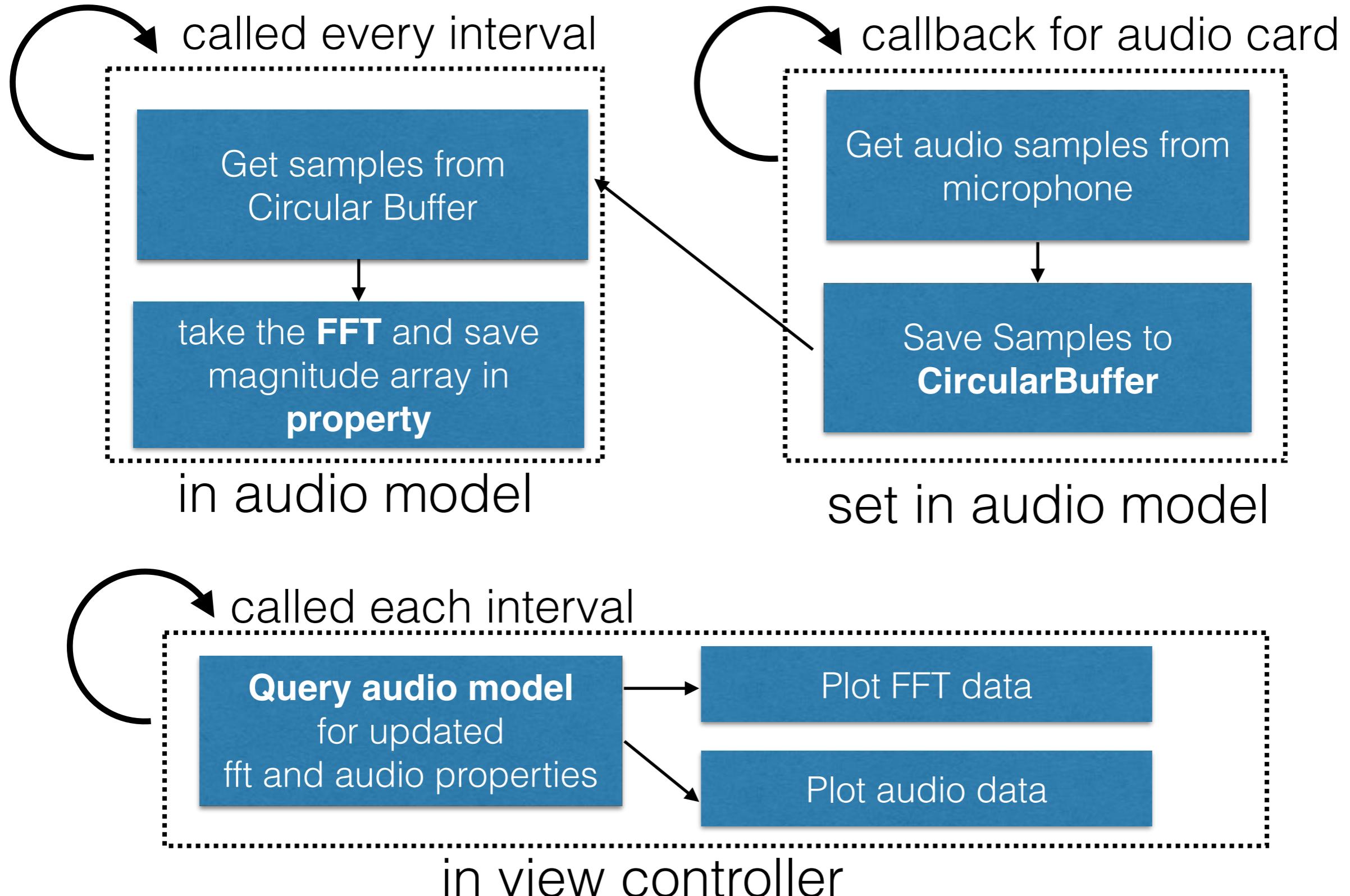
```
//setup the fft  
_fftHelper = [[FFTHelper alloc] initWithFFTSize:BUFFER_SIZE];
```

```
float* fftMagnitude = malloc(sizeof(float)*BUFFER_SIZE/2);  
  
// take forward FFT  
[self.fftHelper performForwardFFTWithData:arrayData  
    andCopydBMagnitudeToBuffer:fftMagnitude];  
  
free(fftMagnitude);
```

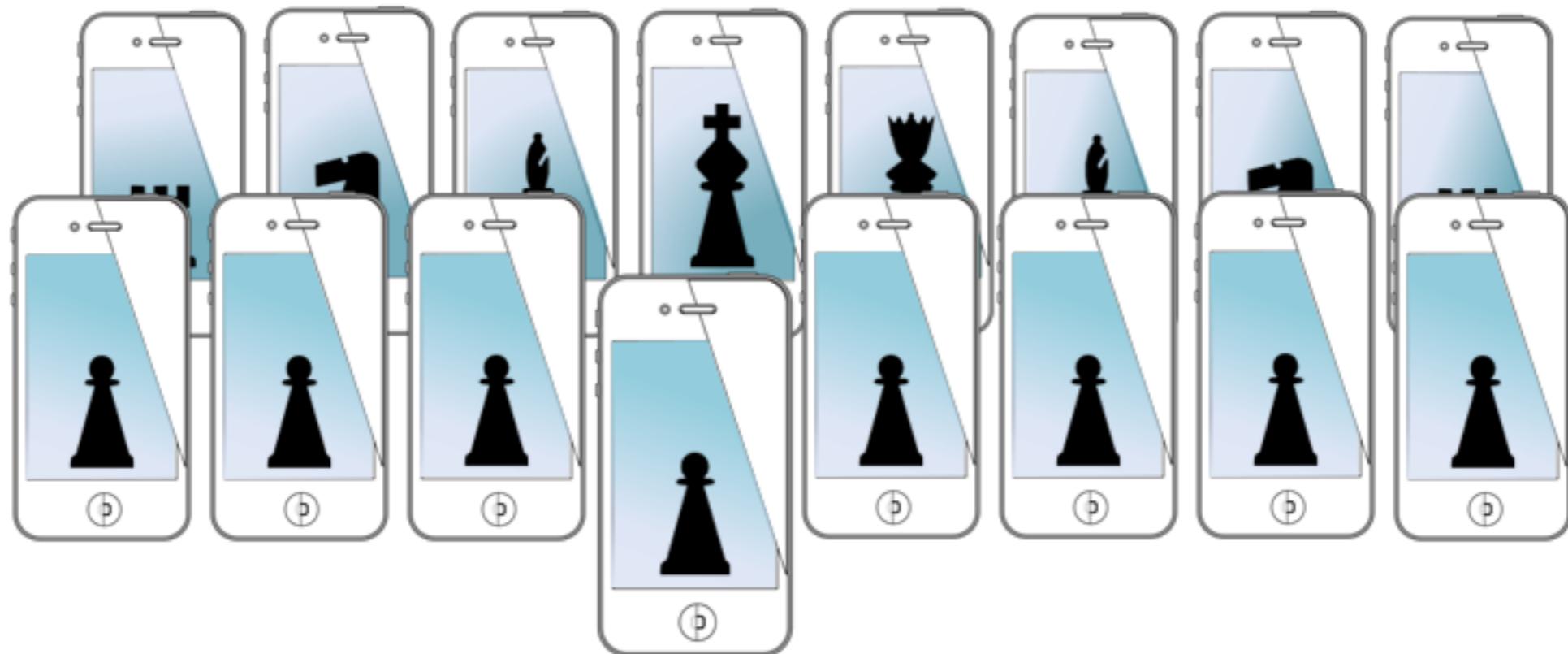
input array  
N=window size

magnitude out  
N=half window size

# The program



# MOBILE SENSING LEARNING

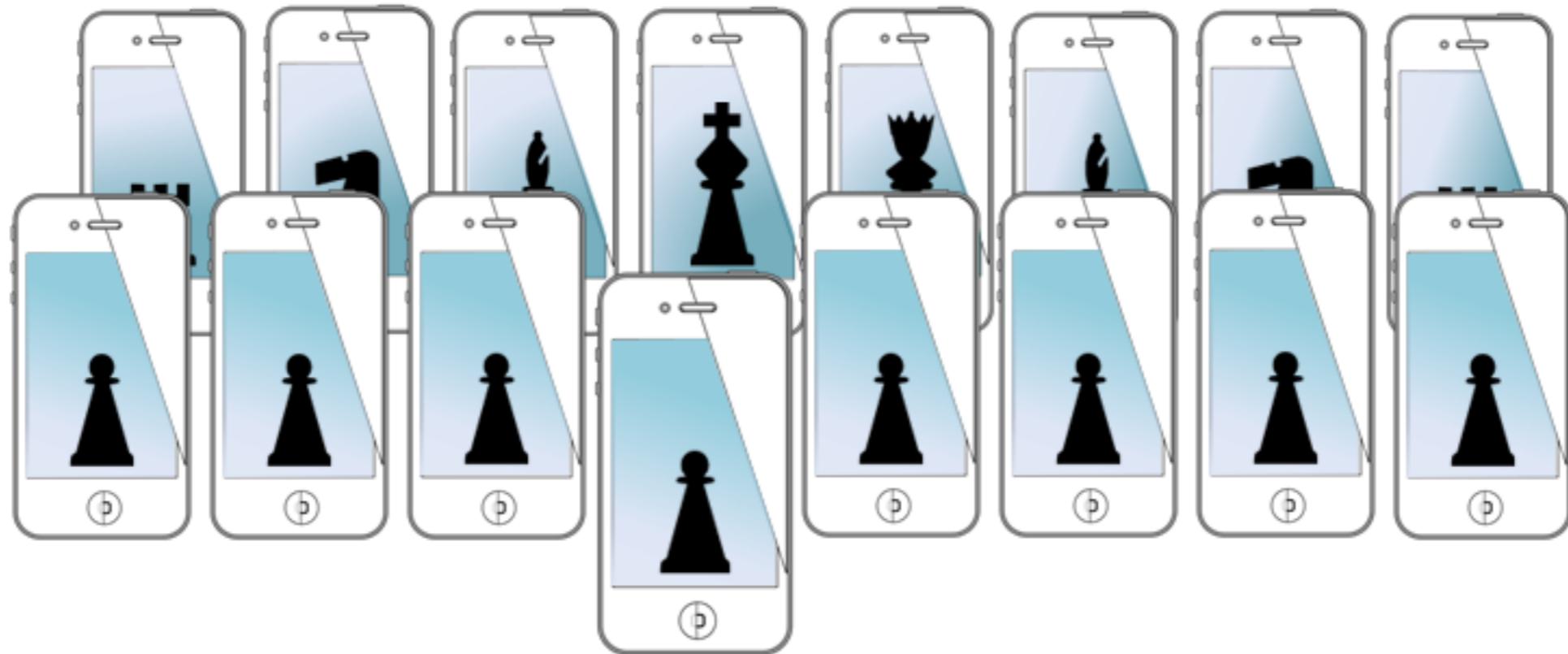


**CS5323 & 7323**  
Mobile Sensing and Learning

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  $y[n]$ ”



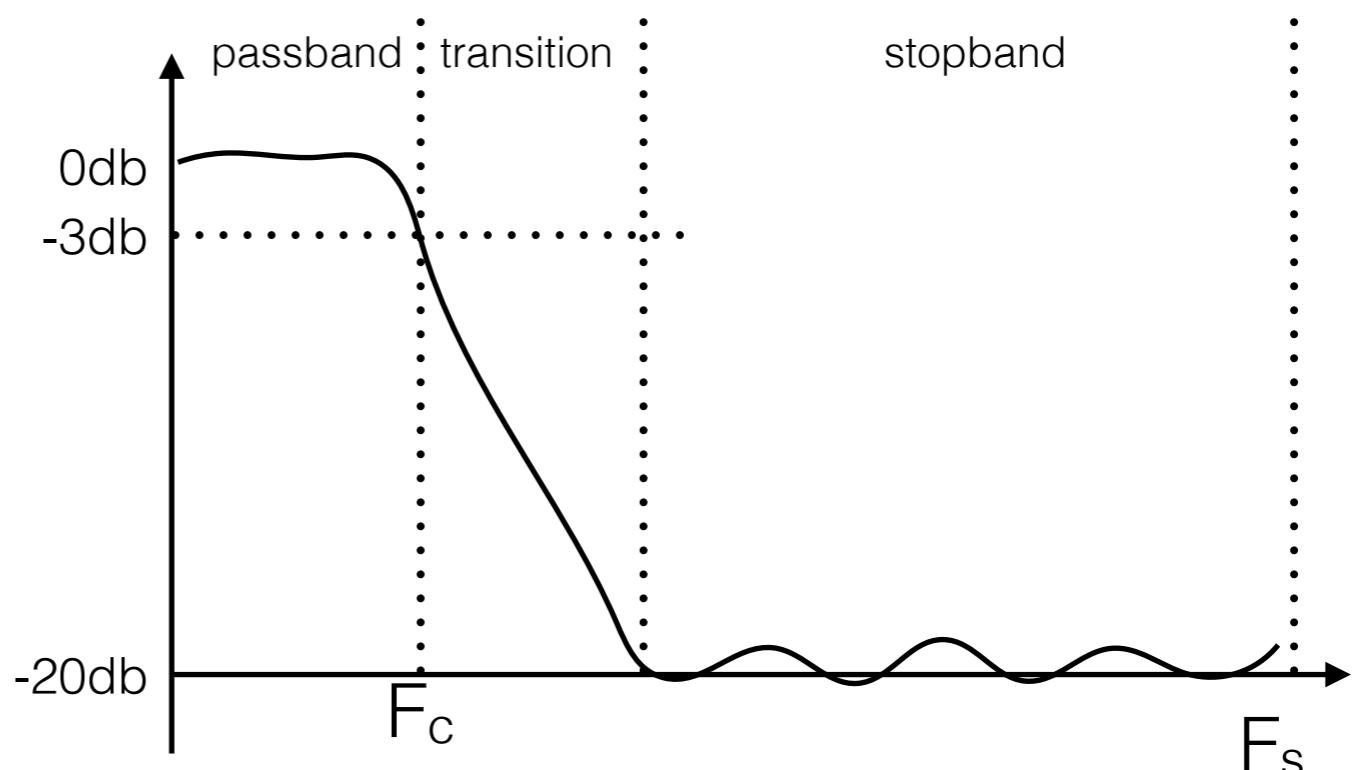
# filters

---

- 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

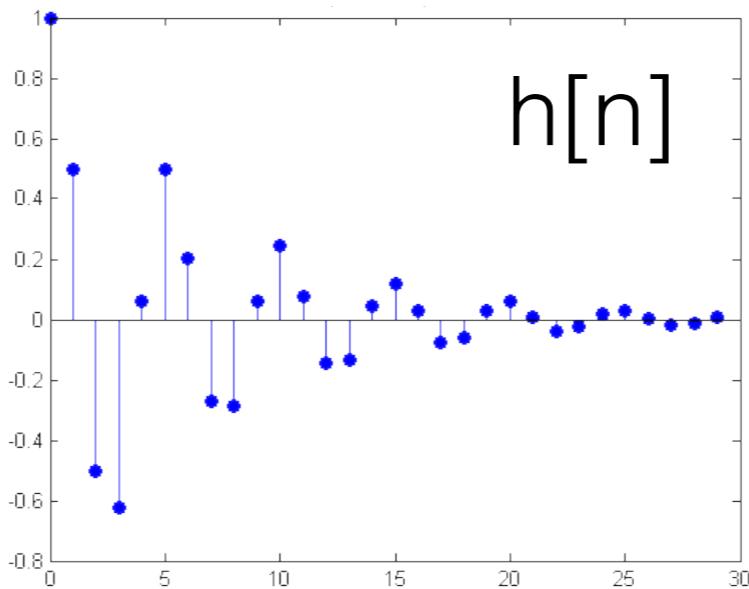
# filters in frequency

- 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

- filters 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
- lots of other places!

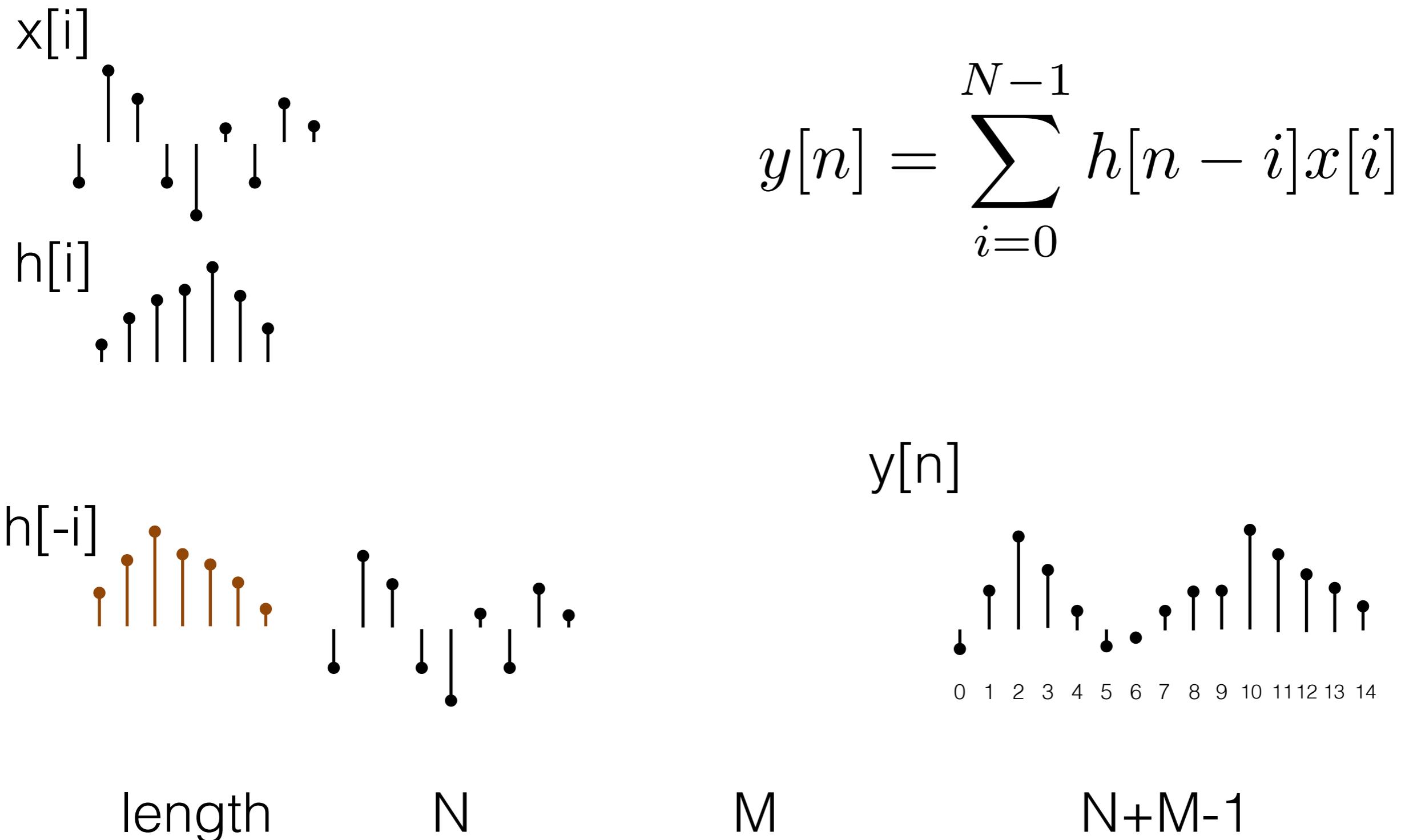
```
from pylab import *
import scipy.signal as signal
n = 61
a = signal.firwin(n, cutoff = 0.3,
                  window = "hamming")
```

# 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

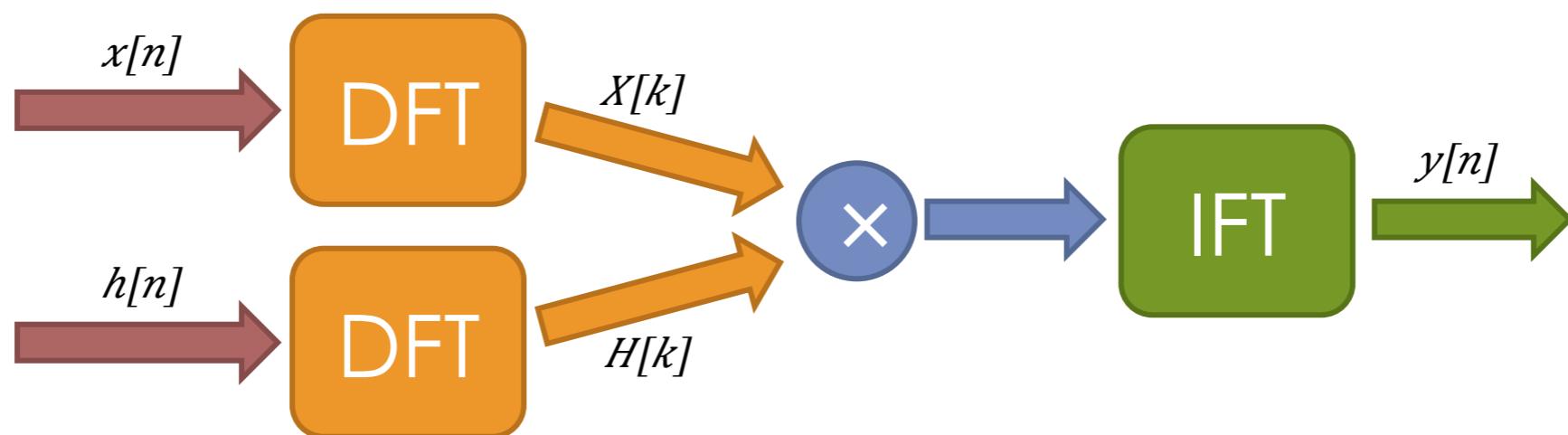
$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

# convolution



# convolution efficiency

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

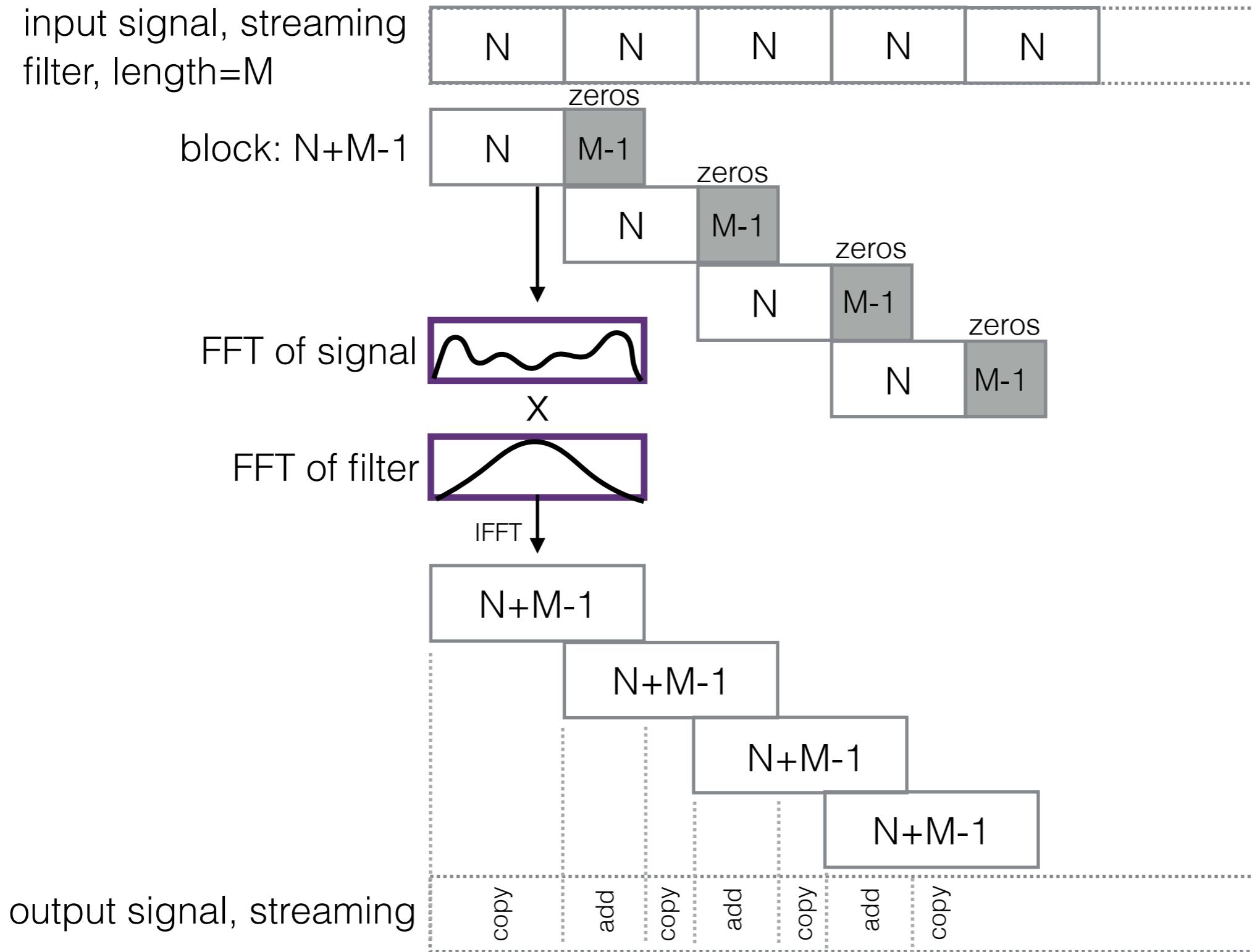


# its circular

---

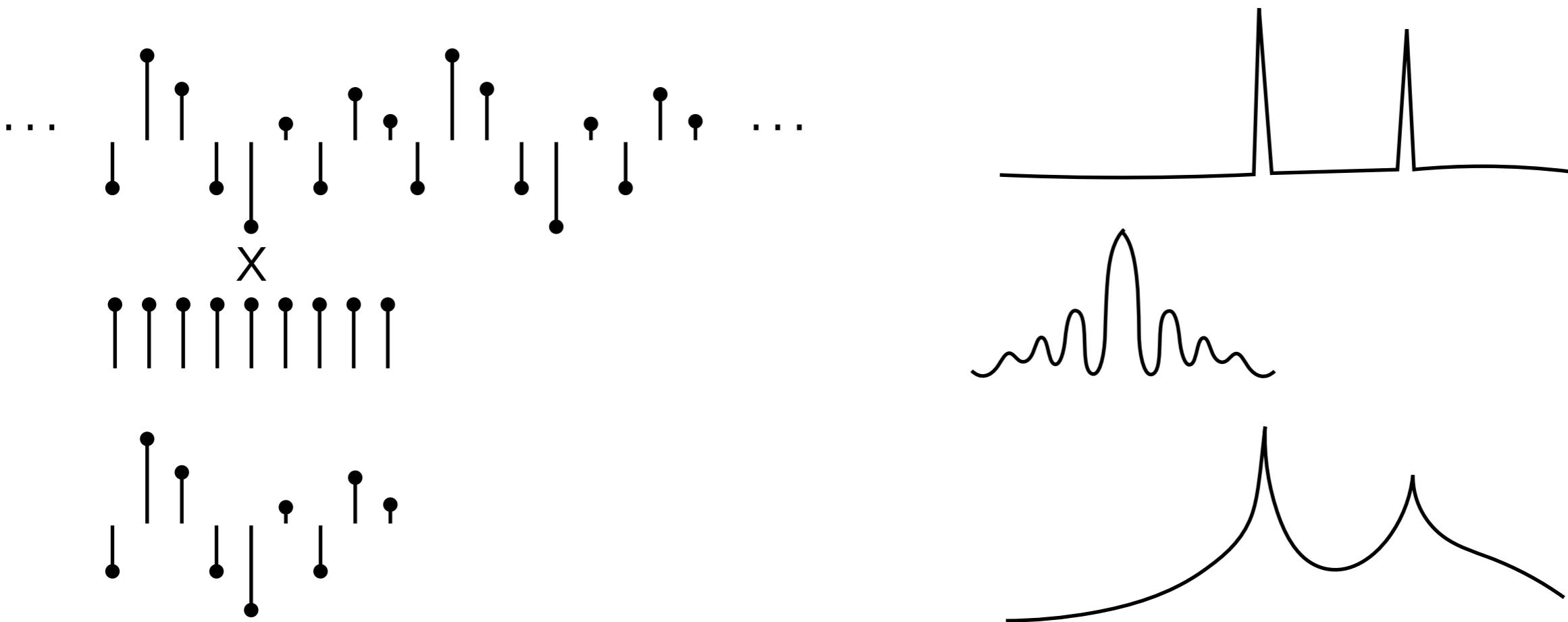
- just using N point FFT performs “Circular Convolution”
  - which is not linear convolution
  - causes the tail end of the convolution to “wrap around” to the beginning
    - FFT assumes the function is periodic (we did not talk about this)
- be aware of circularity when filtering your signal with the 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](http://en.wikipedia.org/wiki/Overlap–add_method)

# overlap and add

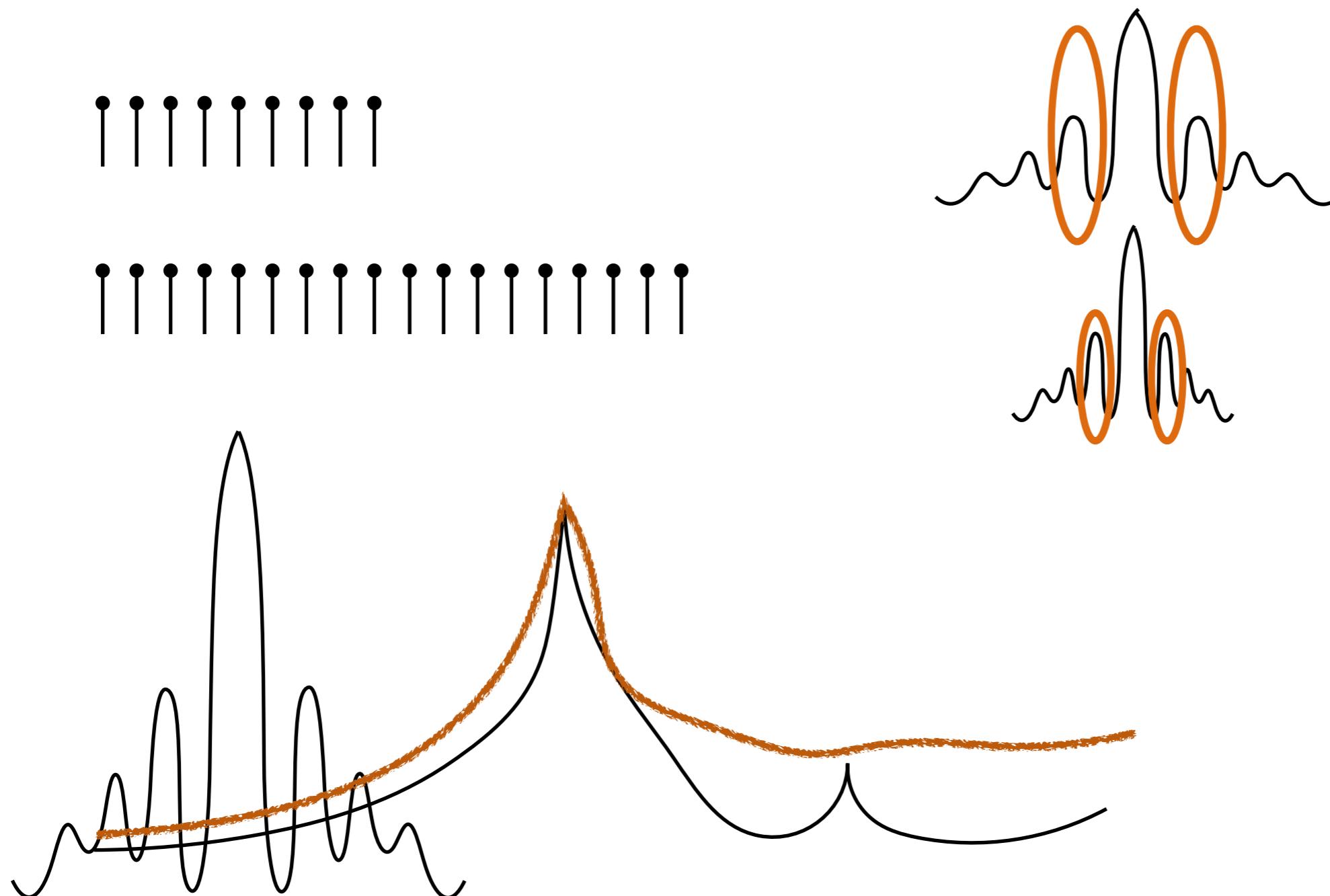


# windowing: spectral BW widening

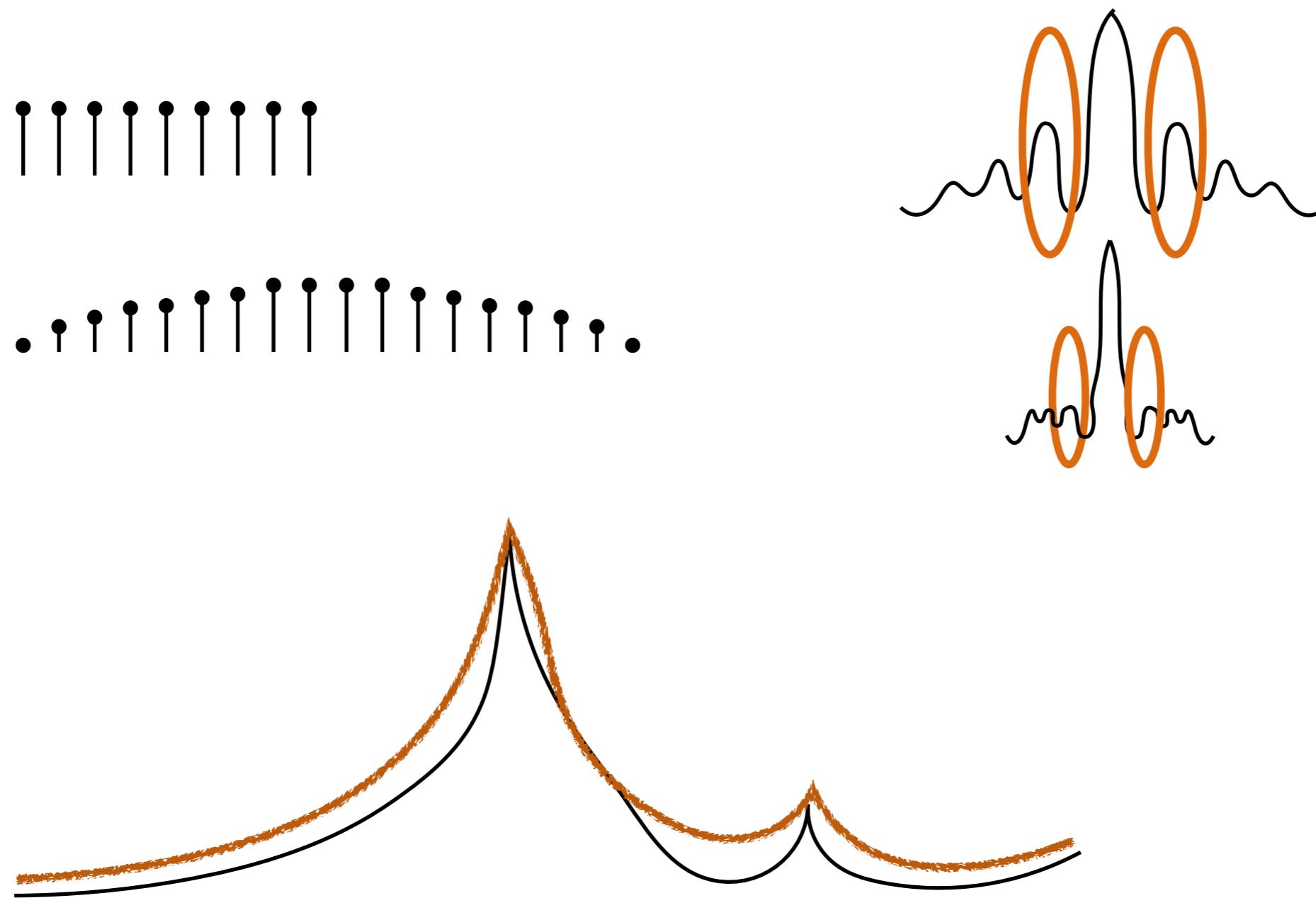
- multiplication in time is convolution in frequency
- a window is something we multiply in time with our signal
- windowing is unavoidable
  - why? we cannot take an infinite FFT...



# windowing: spectral leakage



# windowing: spectral leakage



# which window to use?

---

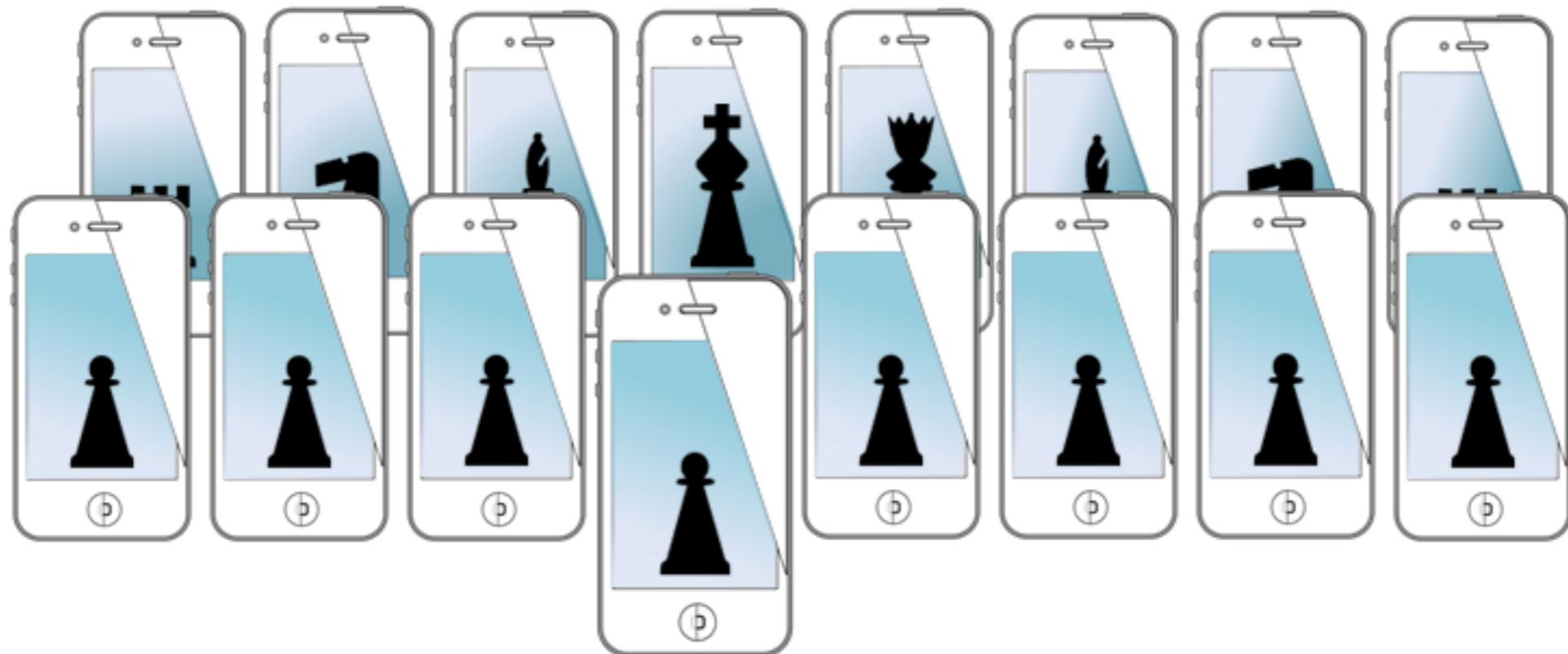
- depends
  - narrowest main lobe: rect
  - good tradeoff: hamming (or Von Hann)
  - optimal tradeoff for a given bandwidth:
    - discrete prolate spheroidal sequence (dpss, Slepian taper)

# FFT review

---

- sampling rate
  - dictates the time between each sample, ( $1 / \text{sampling rate}$ )
  - 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  $(F_s / N)$  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