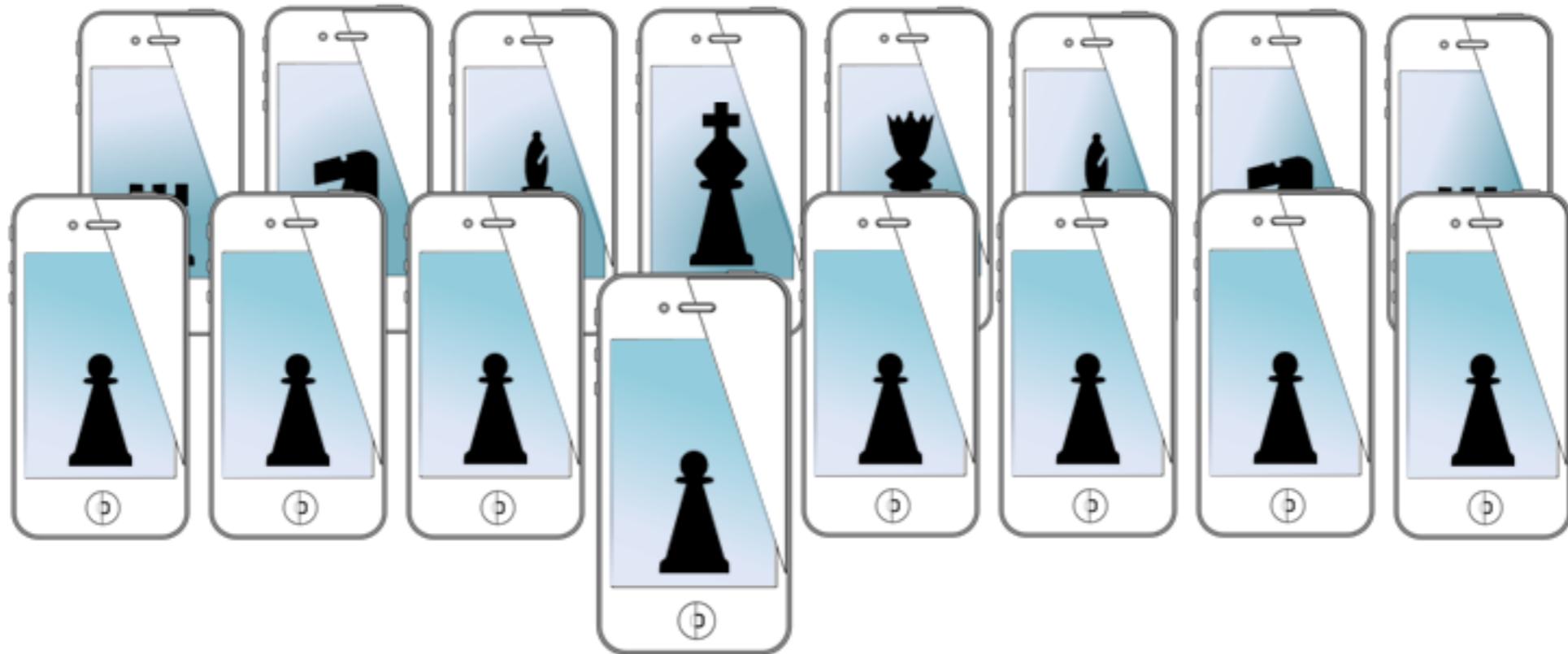


MOBILE SENSING LEARNING



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

- in-class assignment on audio next time!

agenda

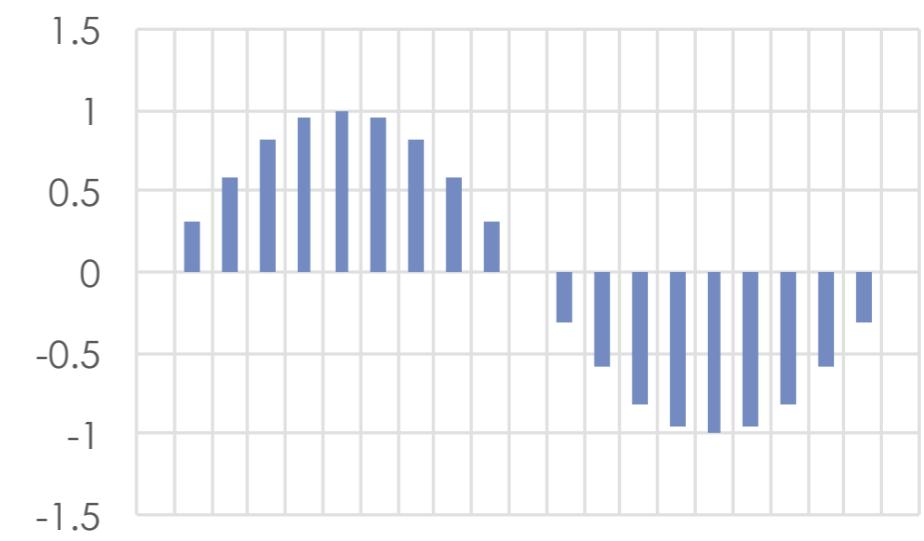
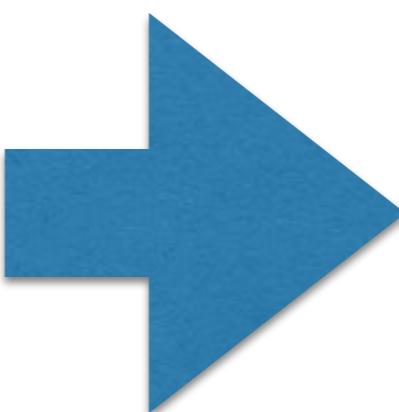
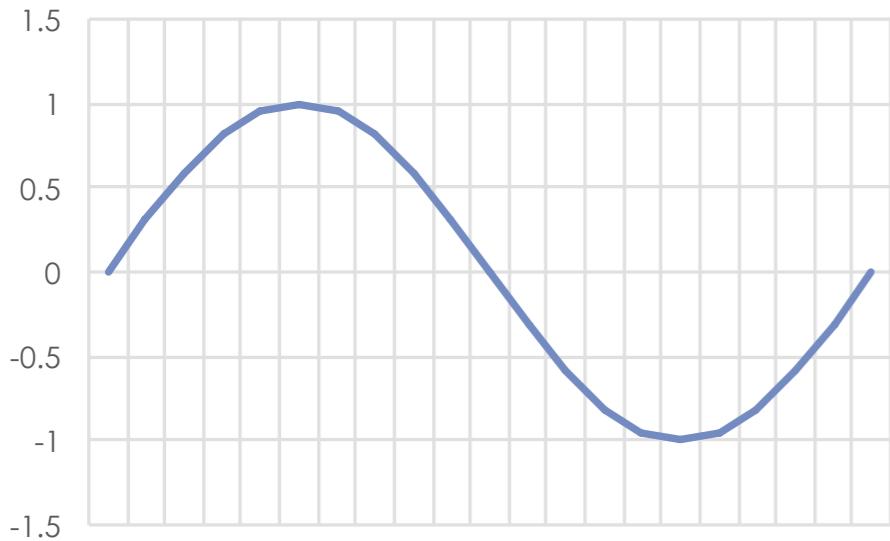
- dealing with sampled data
- the accelerate framework
 - massive digital signal processing library
- graphing audio fast (well, graphing anything)
 - must use lowest level graphing, OpenGL

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

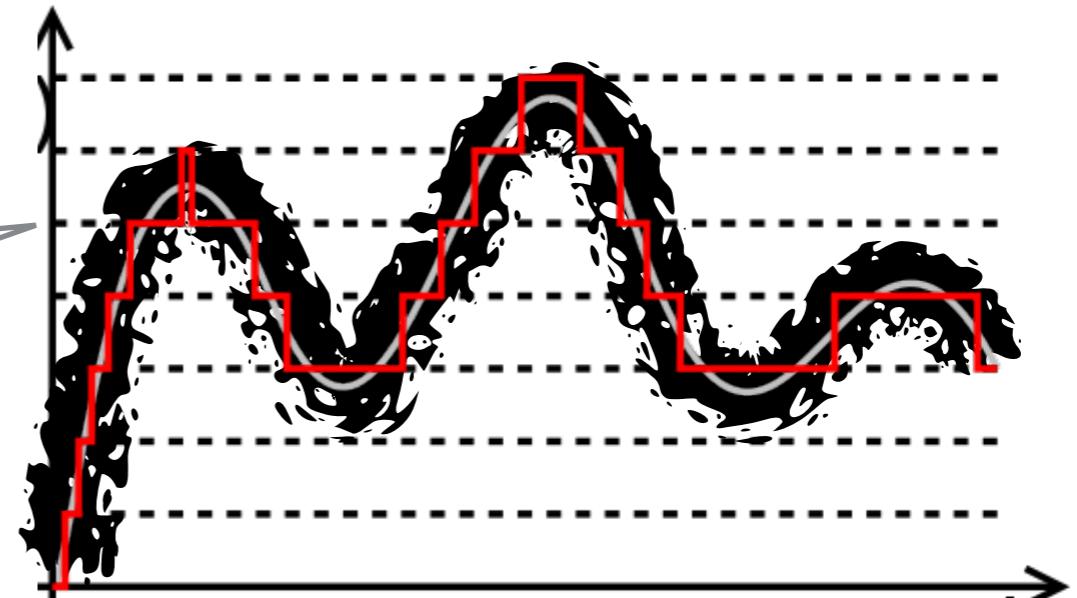
- physical processes are continuous
 - to process with computers, we must digitize it
 - digitization can change how we understand the signal
- digitization occurs in time and amplitude
 - time: sampling
 - amplitude: quantization



sampled data

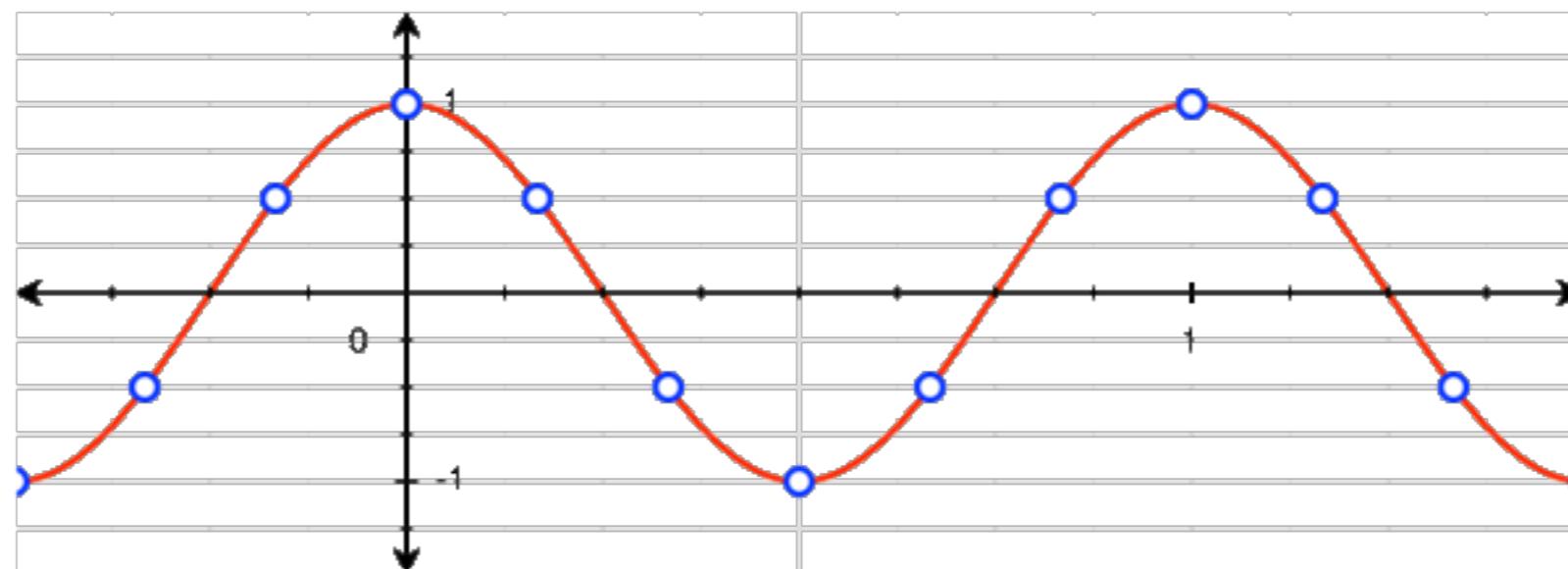
- quantization (amplitude)
 - introduces error in estimating amplitude of a signal
 - error can be reduced by adding more “bits per sample”
- most ADCs are 16 bits, considered “good enough”
- sufficient for most uses
 - not for others!

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?

making a sine wave

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

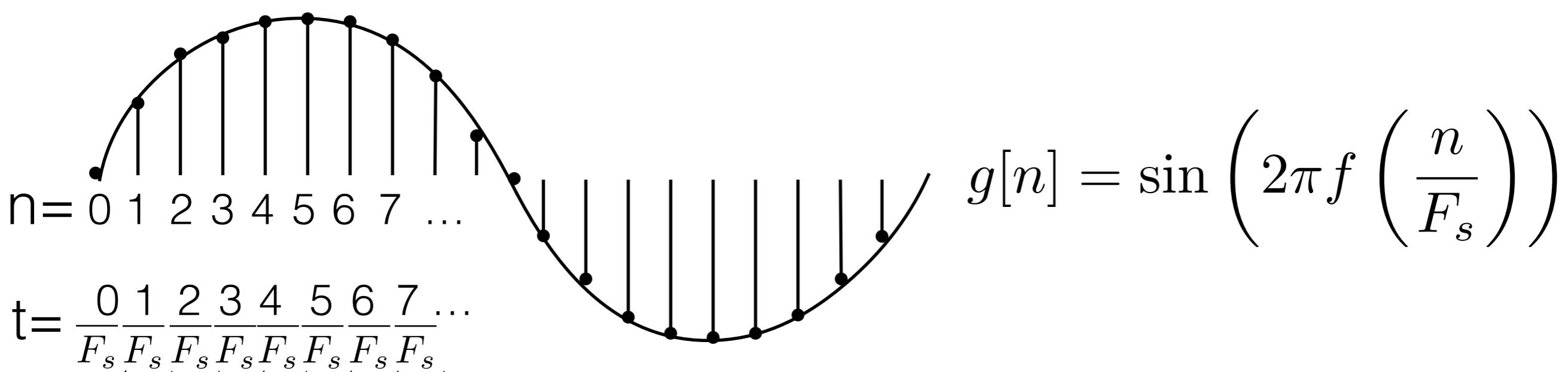
$$g(t) = \sin(2\pi ft)$$

equation for sine wave

frequency in Hz

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?

```
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

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

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

[audioManager setOutputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
{
    double phaseIncrement = 2*M_PI*frequency/samplingRate;
    double sineWaveRepeatMax = 2*M_PI;
    for (int i=0; i < numFrames; ++i)
    {
        data[i] = sin(phase);

        phase += phaseIncrement;

        if (phase >= sineWaveRepeatMax) phase -= sineWaveRepeatMax;
    }
}];
```

sample from the mic

- 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 for the complete API
- provides mathematics for performing fast DSP

```
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

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

```
[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);  
  
}];
```

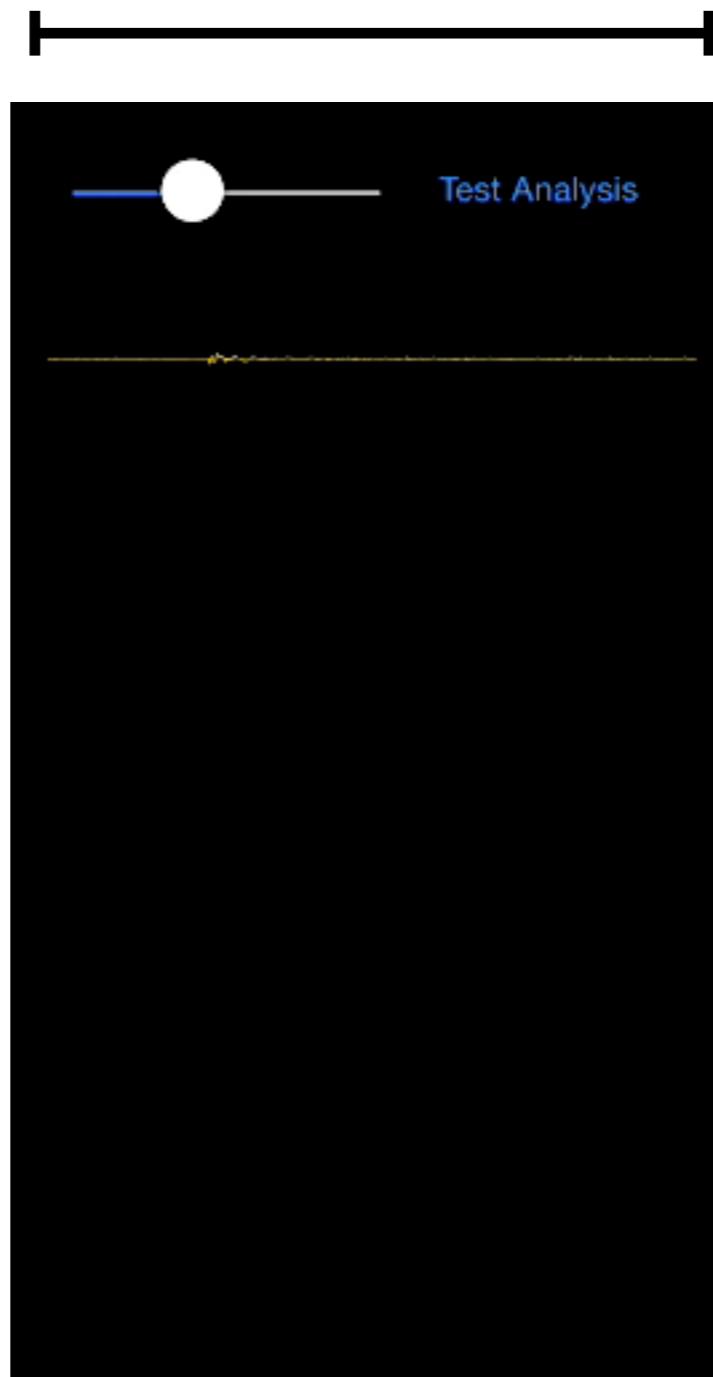
```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)  
  
    vDSP_vsq(data, 1, data, 1, numFrames*numChannels);  
    float meanVal = 0.0;  
    vDSP_meanv(data, 1, &meanVal, numFrames*numChannels);  
}];
```

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

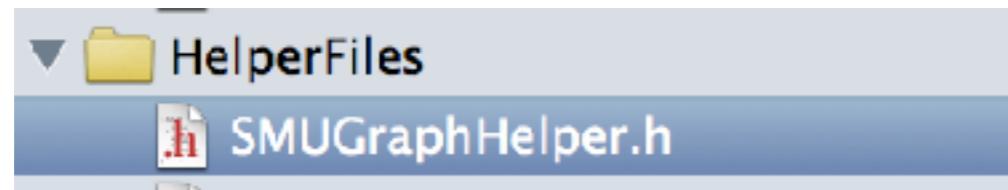
what if we want
lots of graphs?

solution

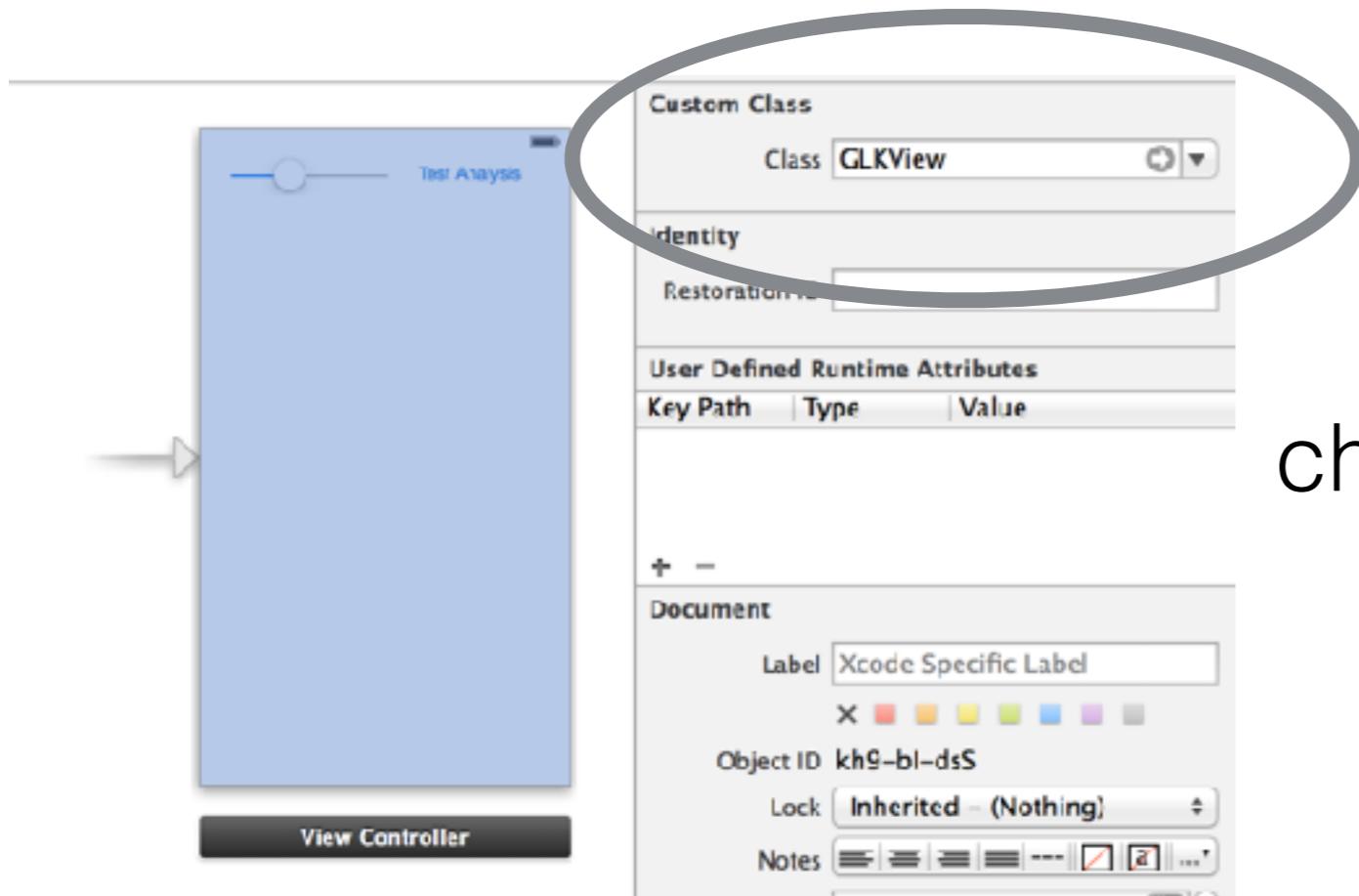
- use the GPU
- set vectors of data on a 2D plane
- let the renderer perform scaling, anti-aliasing, and bit blitting to screen
- ...this is not a graphics course

easy solution

- use graph helper, which uses GLKView and GLKViewController



add .h and .m to project



change view to GLK View

the graph helper

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

When setting up:

```
// start animating the graph
_graphHelper = [[SMUGraphHelper alloc] initWithController:self
                                                    preferredFramesPerSecond:15
                                                       numGraphs:1
                                              plotStyle:PlotStyleSeparated
                                         maxPointsPerGraph:BUFFER_SIZE];
```

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

In view did load:

```
[self.graphHelper setScreenBoundsBottomHalf];
[self.graphHelper setScreenBoundsTopHalf];
[self.graphHelper setFullScreenBounds];
[self.graphHelper setBoundsWithTop:(float) bottom:(float) left:(float) right:(float)];
```

inherit from OpenGL

declare property

setup GLKViewController

bounds for screen
different options

setting data

```
// override the GLKView draw function, from OpenGL ES
- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect {
    [self.graphHelper draw]; // draw the graph
}
```

called for each draw to screen

```
-(void) setGraphData:(float*)
   WithDataLength:(int)
    forGraphIndex:(int)
    withNormalization:(float)
    withZeroValue:(float)
```

```
-(void) setGraphData:(float*)
   WithDataLength:(int)
    forGraphIndex:(int)
```

prototypes for setting scatter data

```
// override the GLKViewController update function, from OpenGL ES
- (void)update{
    // just plot the audio stream
```

get data (shown here: from buffer)

```
// get audio stream data
float* arrayData = malloc(sizeof(float)*BUFFER_SIZE);
[self.buffer fetchFreshData:arrayData withNumSamples:BUFFER_SIZE];
```

```
//send off for graphing
[self.graphHelper setGraphData:arrayData
   WithDataLength:BUFFER_SIZE
    forGraphIndex:0];
```

set data for 0th graph
no normalization

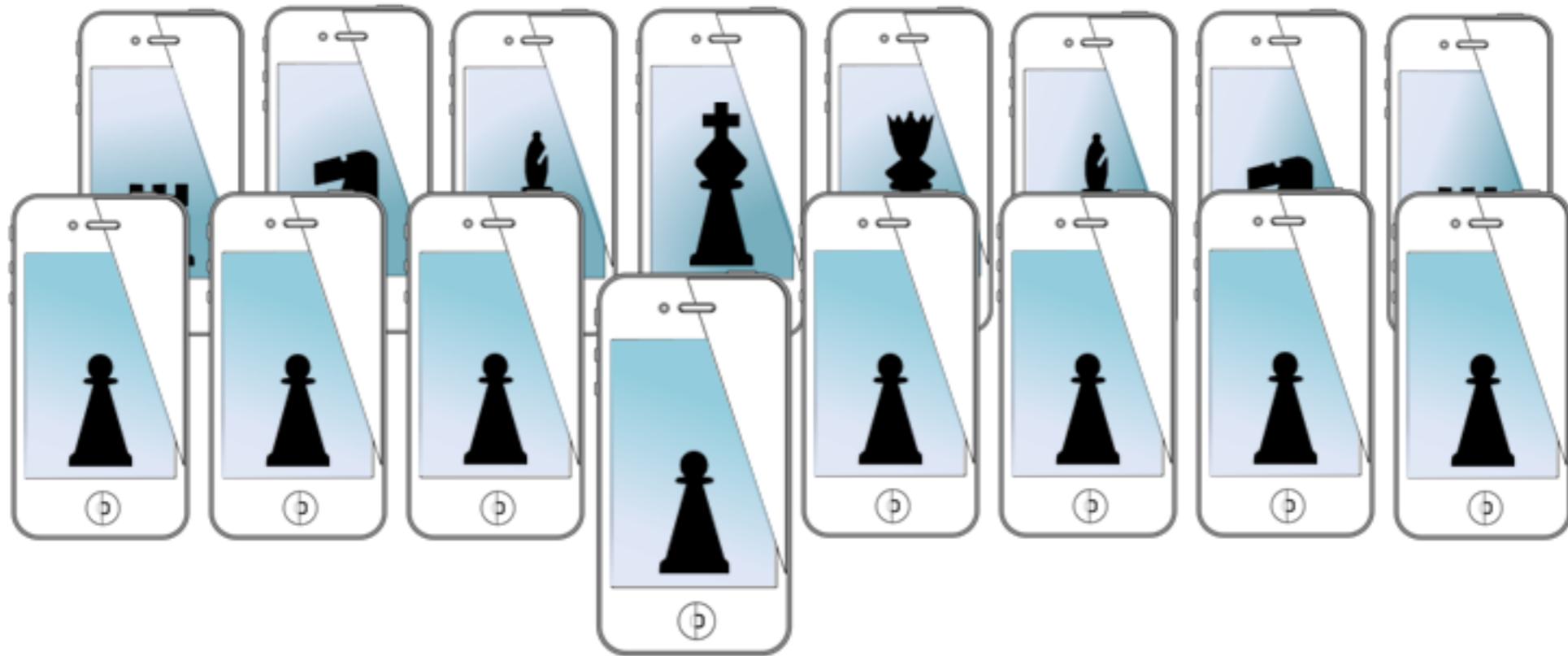
```
[self.graphHelper update]; // update the graph
free(arrayData);
```

update render state and free memory

audio graphing demo!



MOBILE SENSING LEARNING



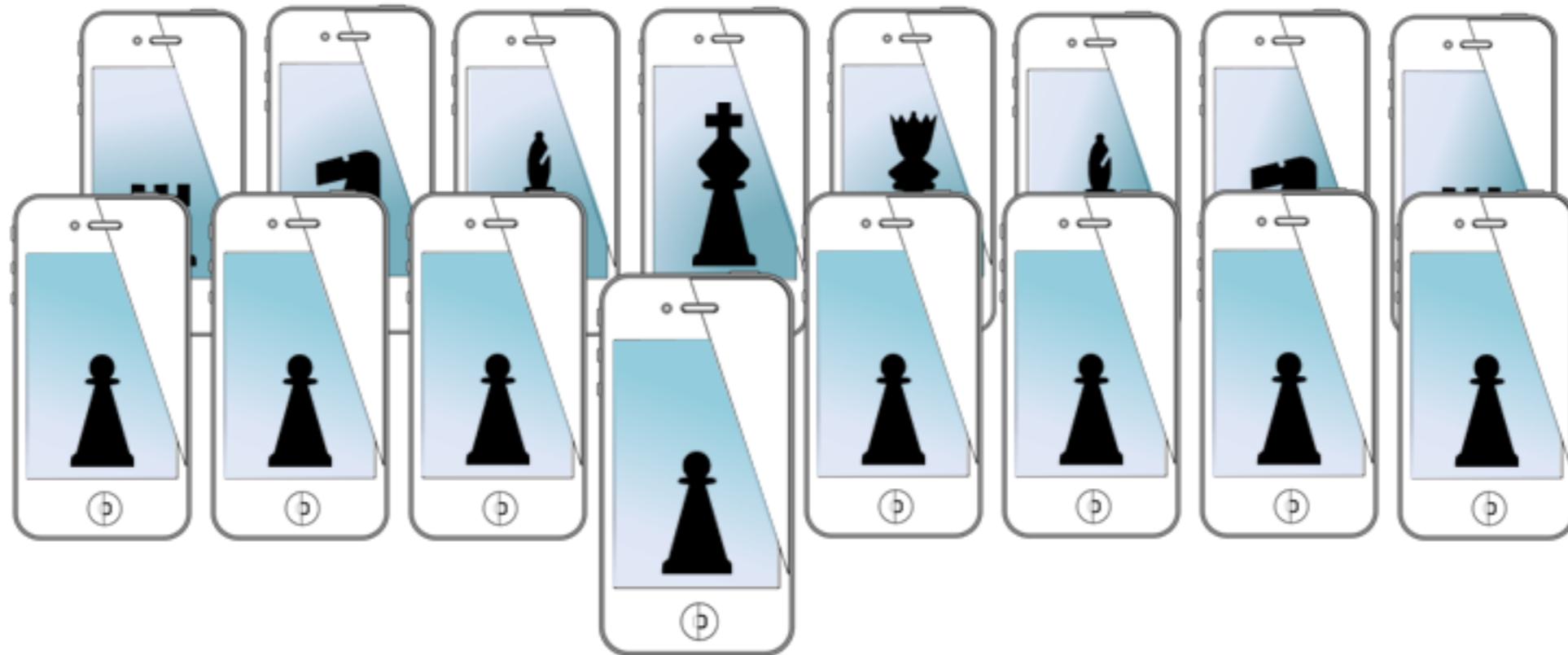
CS5323 & 7323

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

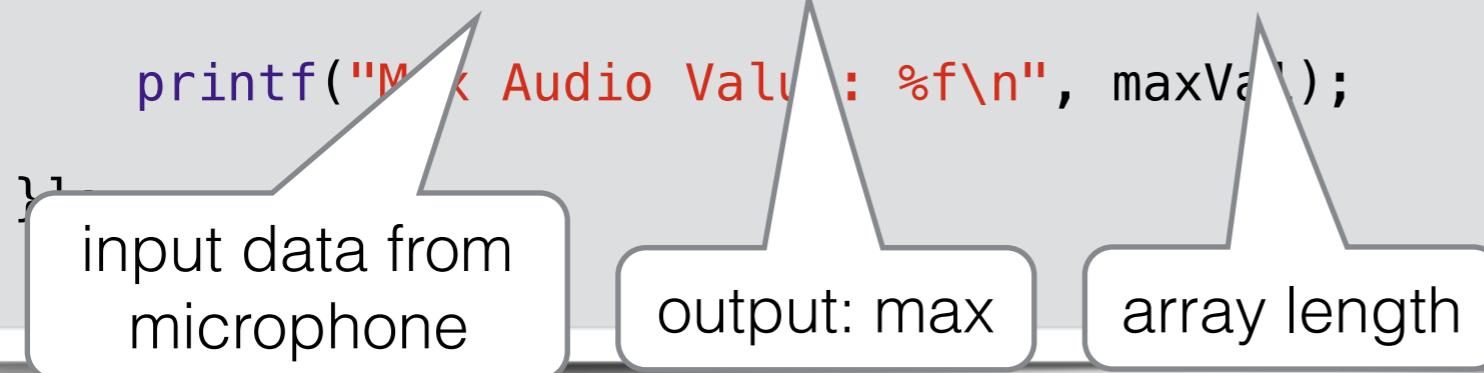
processing audio

- 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);  
};
```



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 mathematician
Worker
Submitted
in 1807

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**.

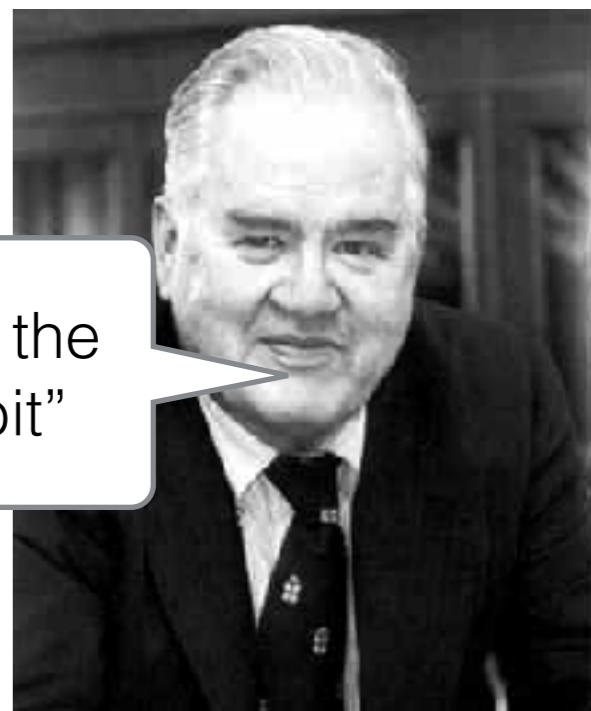
The transform was eventually called The Fourier Transform

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



Cooley

I coined the term “bit”



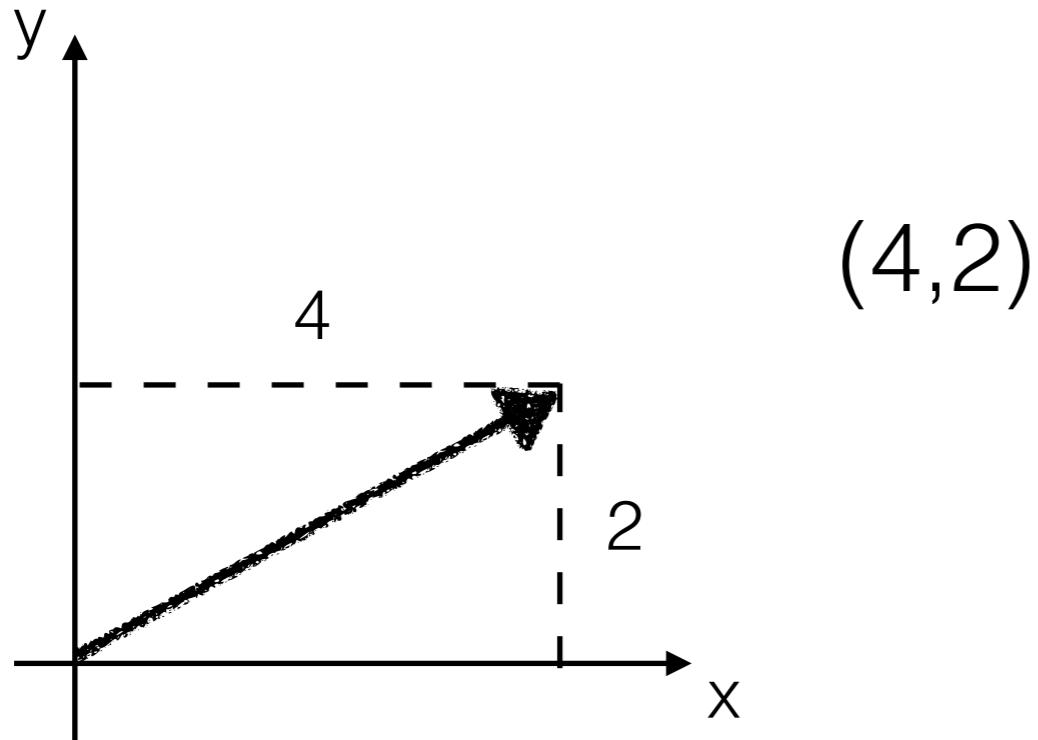
Tukey

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?

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

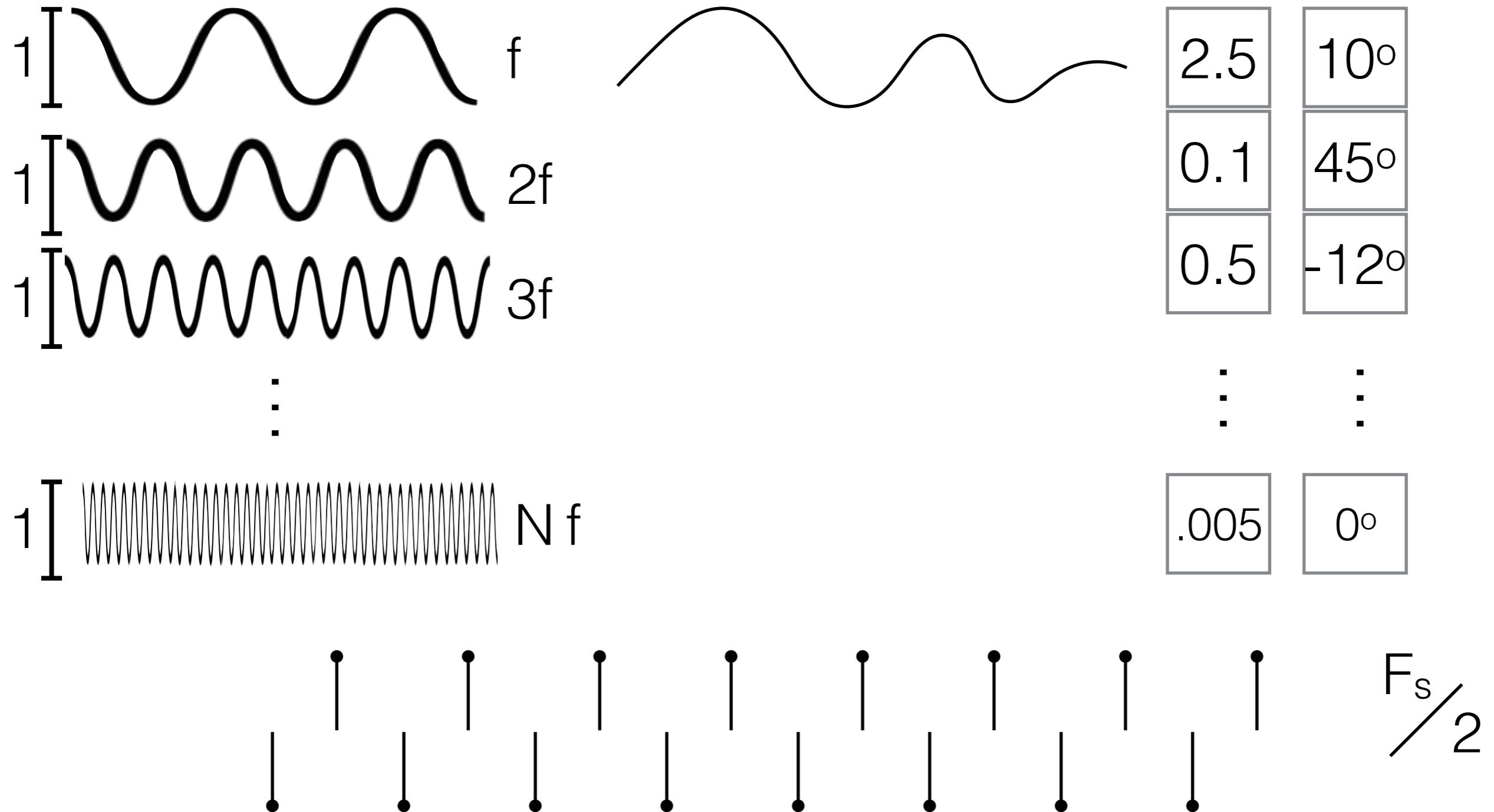
point by point multiplication and add it up!

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

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

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)

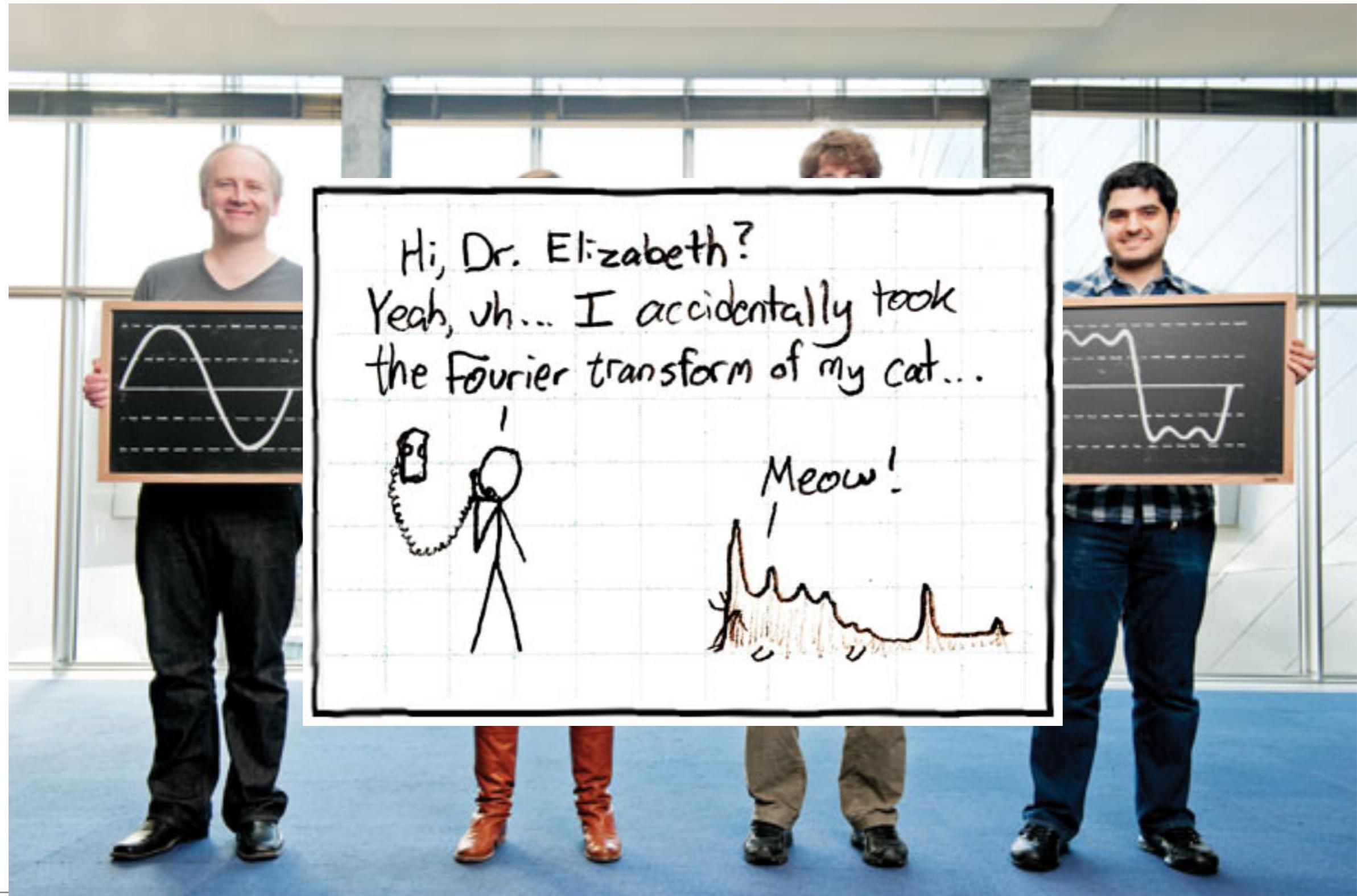
FT by video

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



FT by hilarity

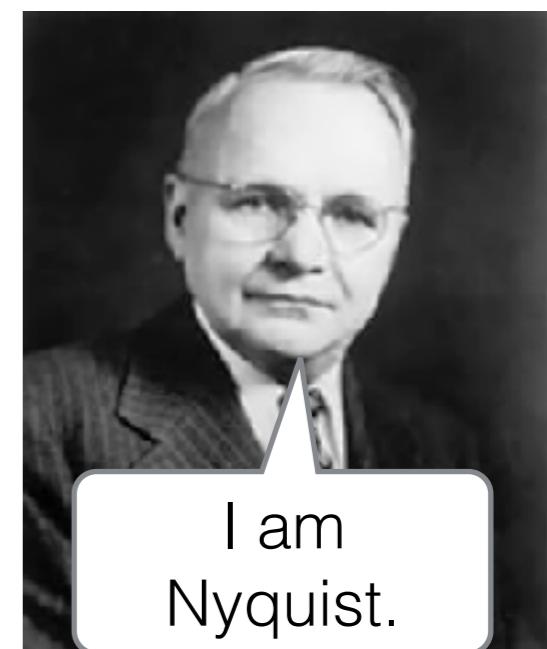
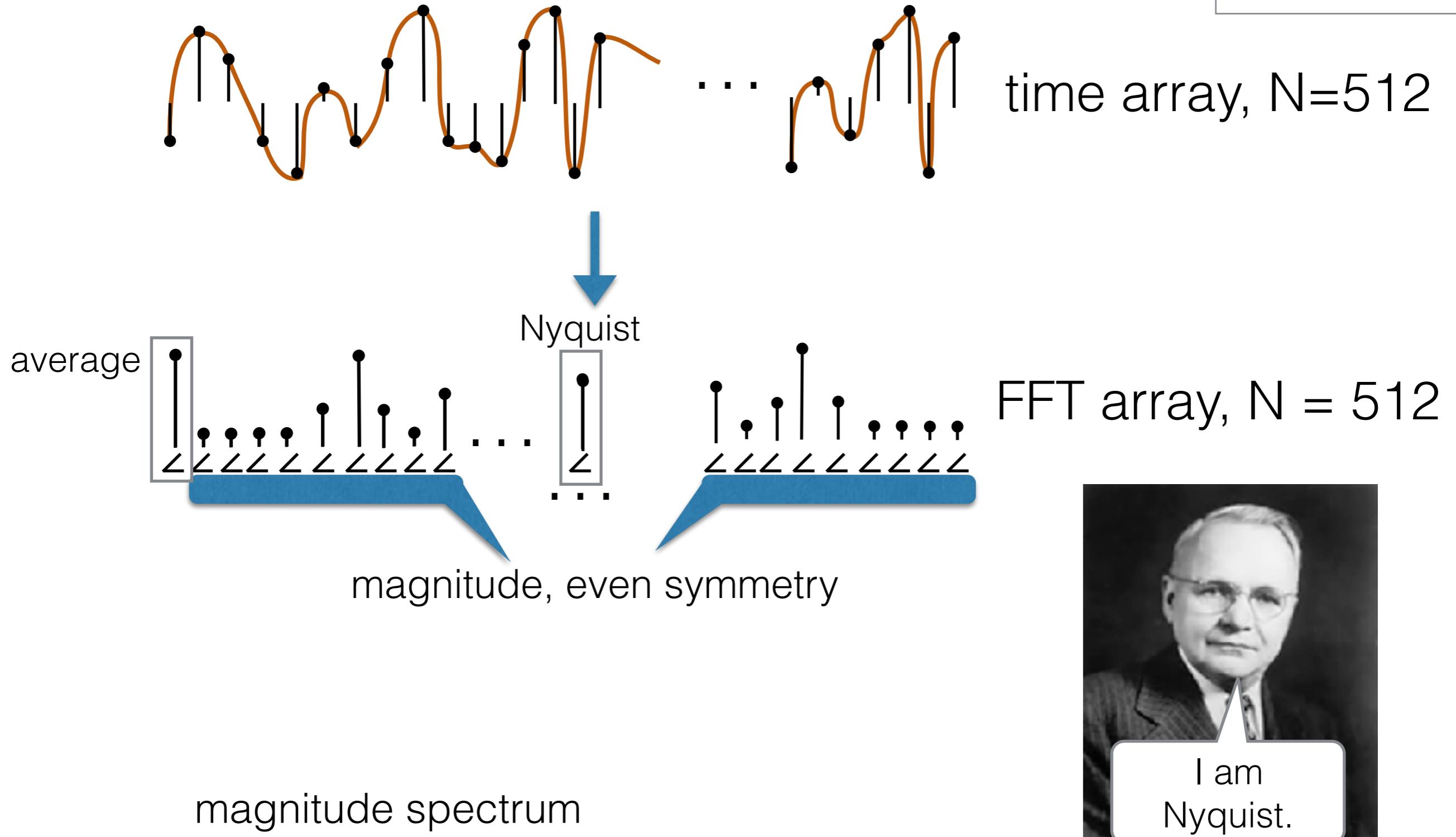
<http://www.preposterousuniverse.com/blog/2014/11/27/thanksgiving-9/>



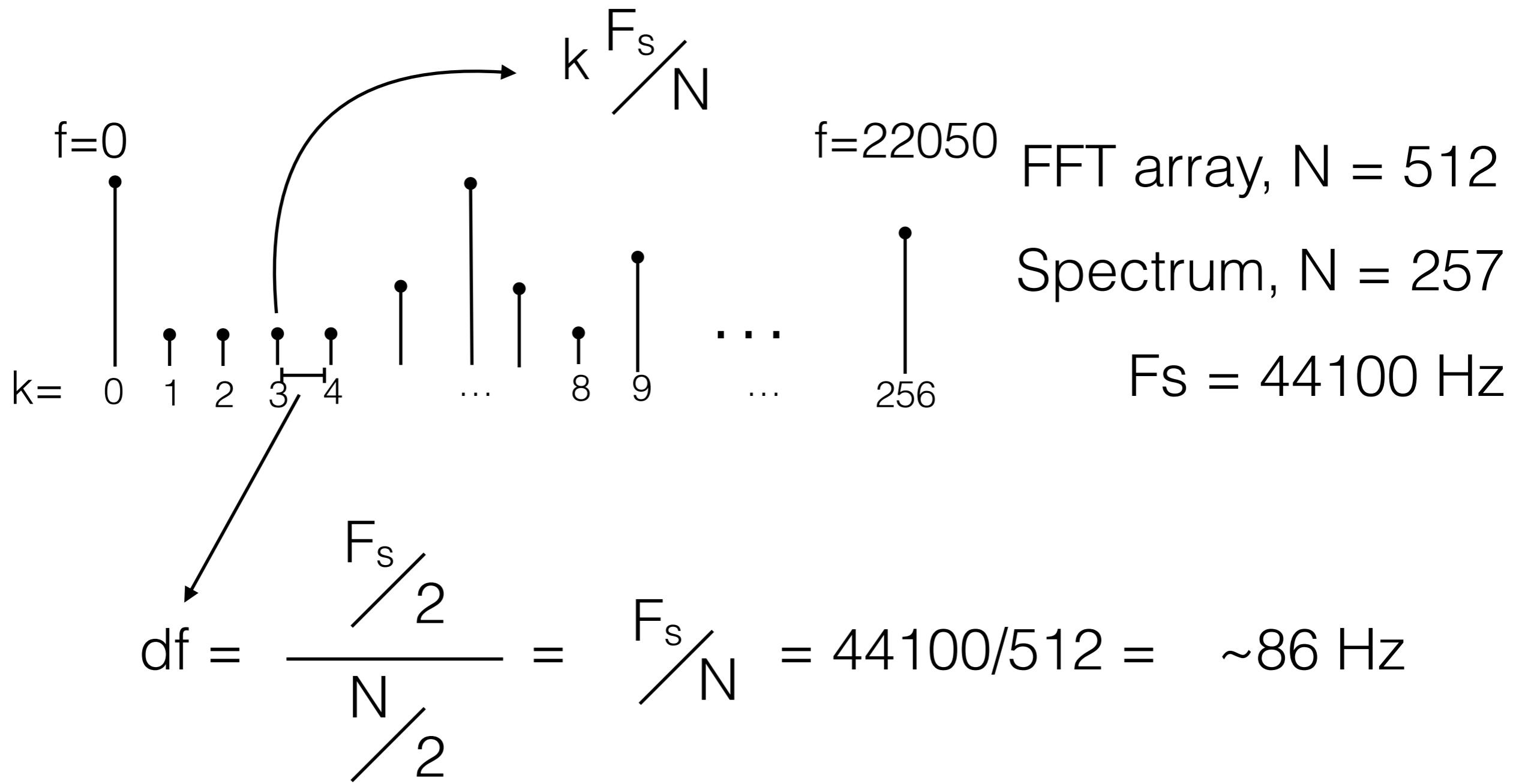
the FFT

- 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

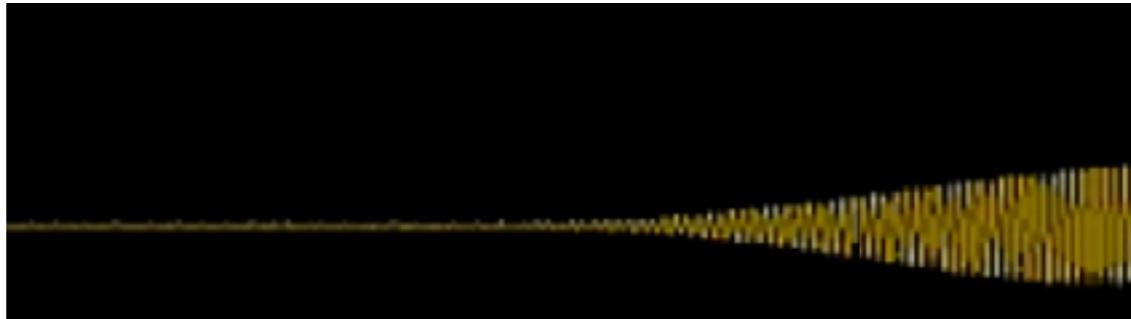


time and frequency



using the FFT

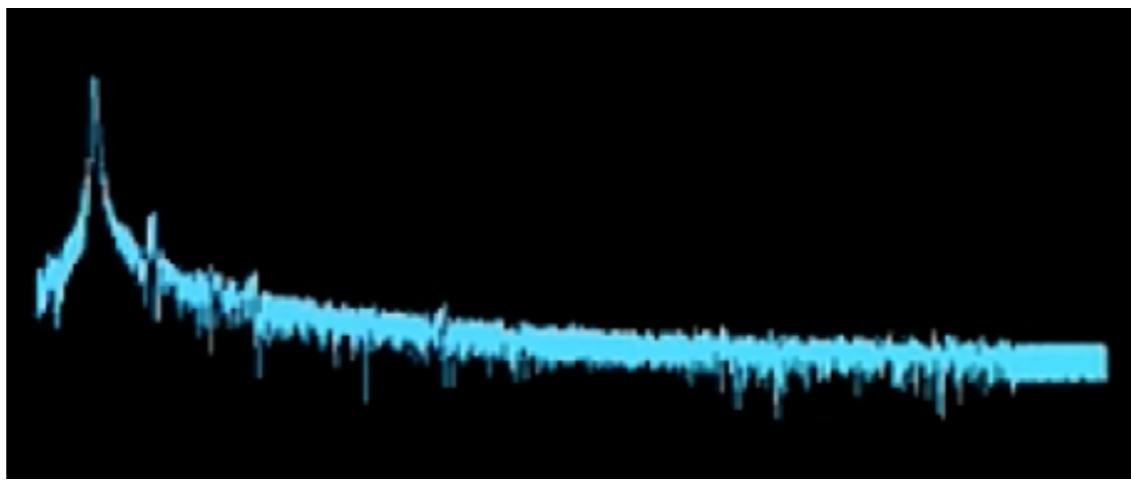
raw audio



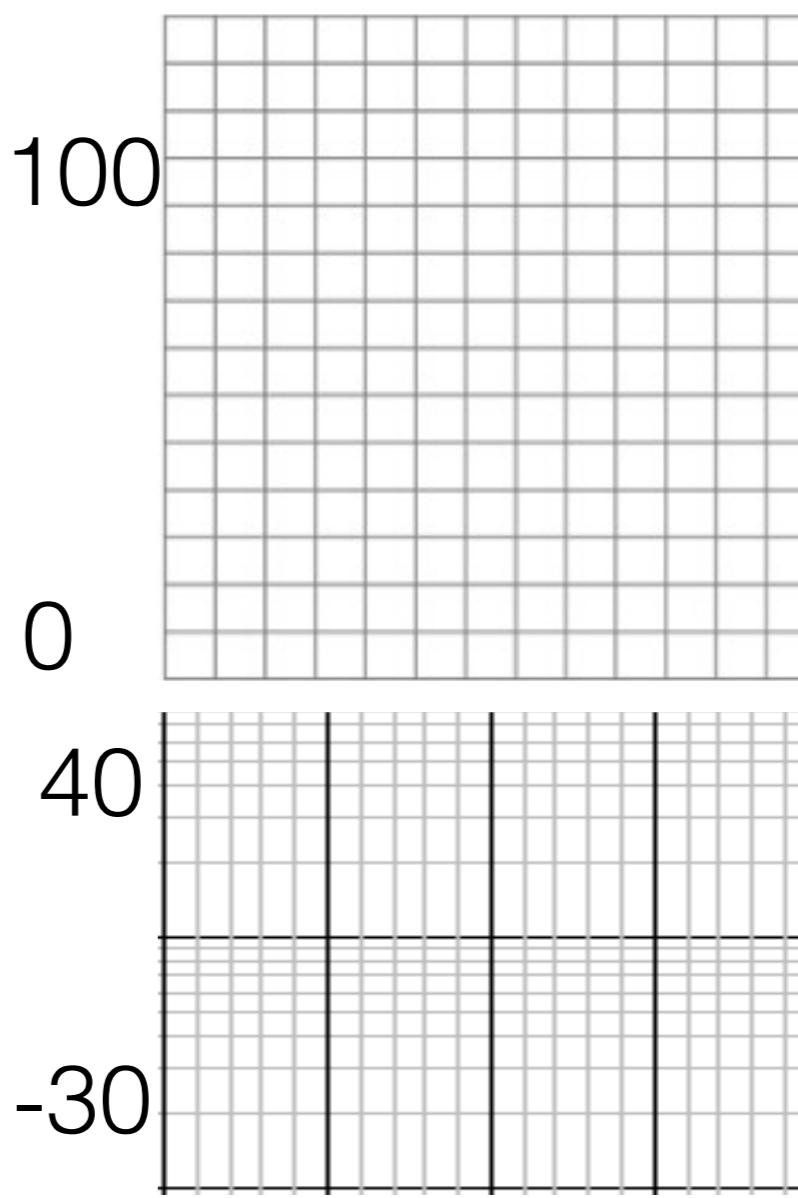
magnitude FFT



magnitude FFT
in dB



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



some fft examples

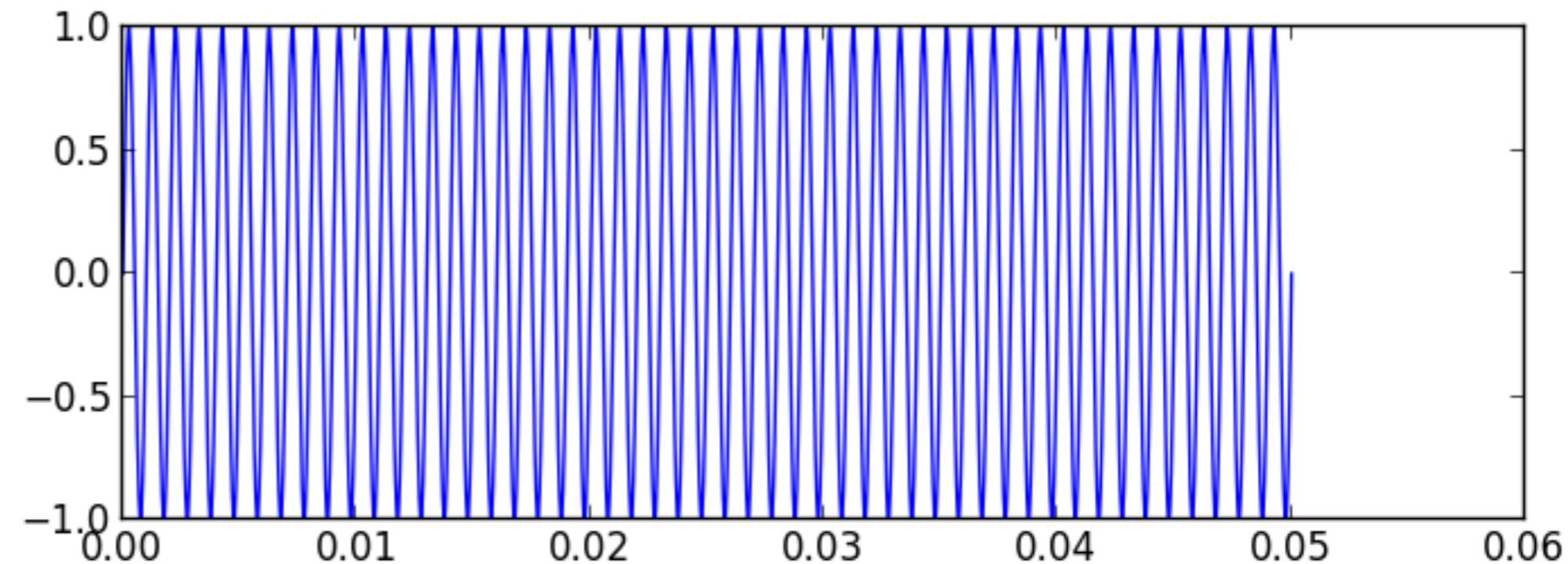
this is just a plot of
an array of values
sampled

**over equal time
intervals**

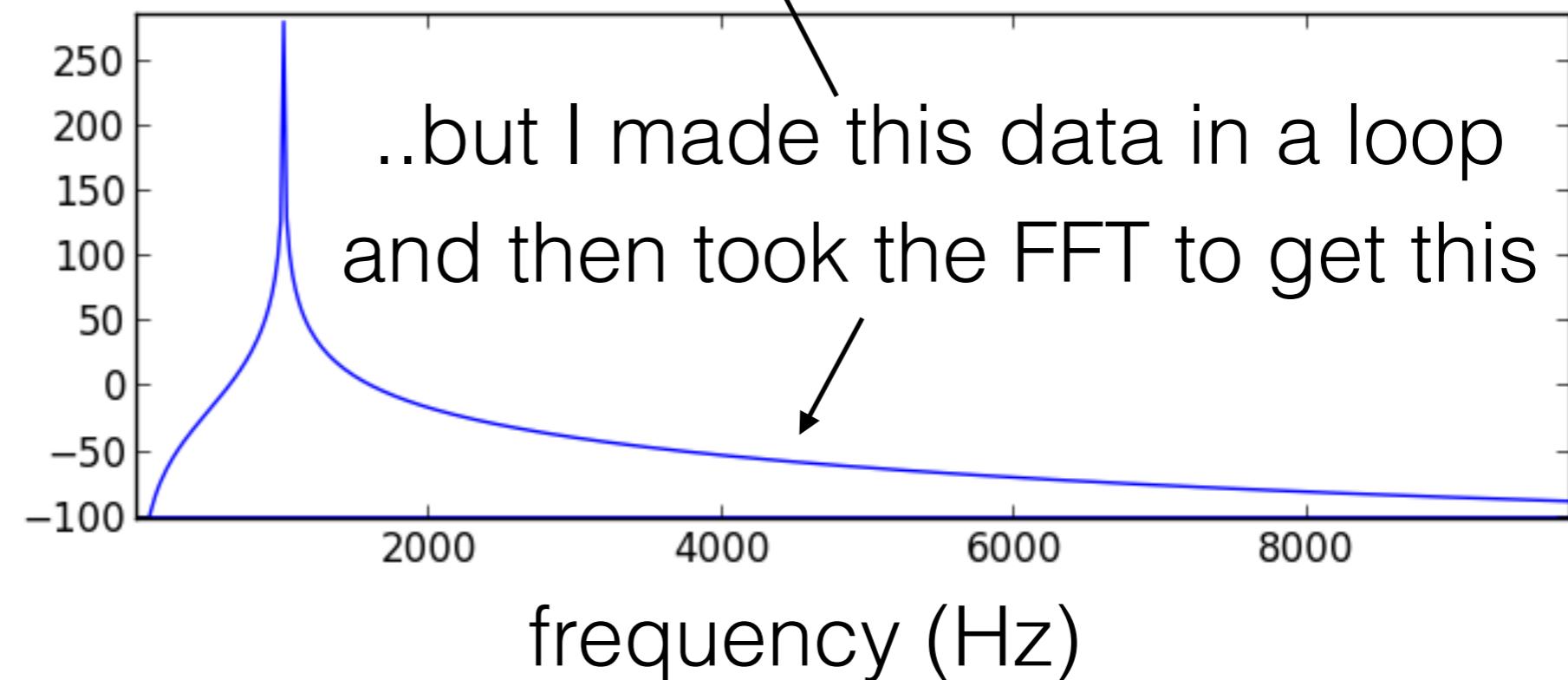
this is just a plot of
an array of FFT
values

**its indices are
equally sampled
over frequency**

1kHz sine wave

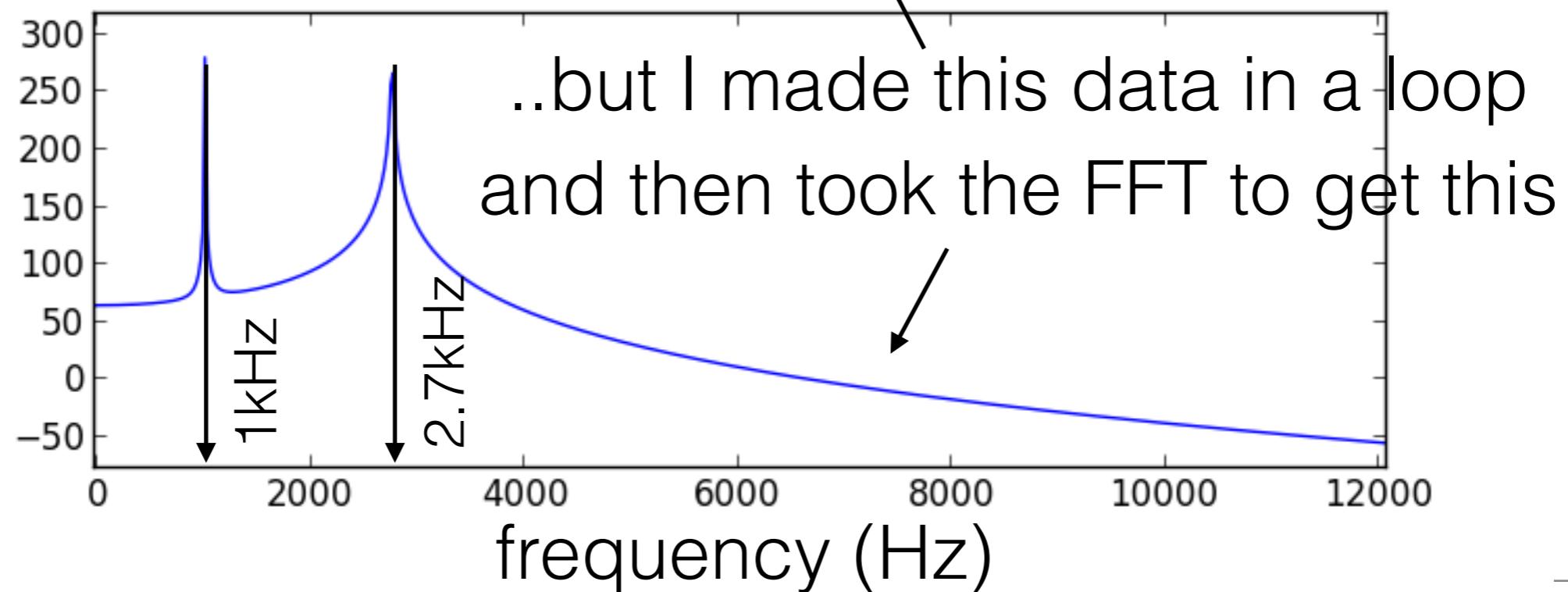
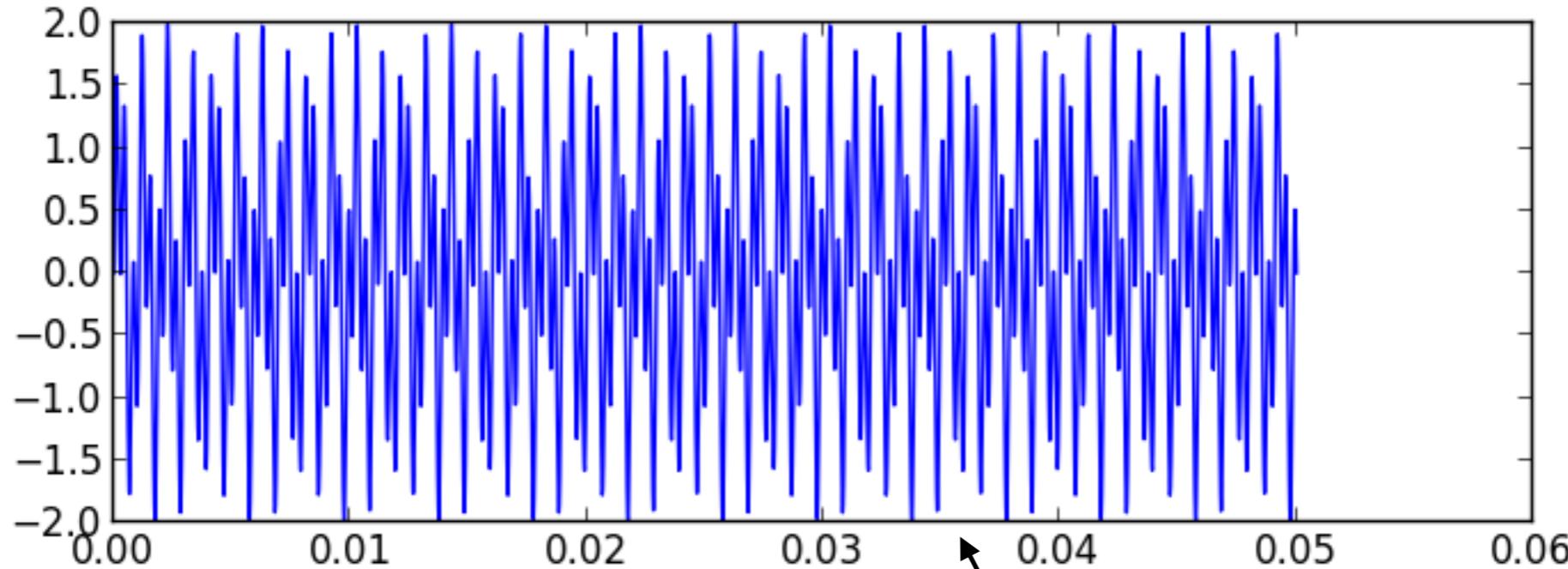


..but I made this data in a loop
and then took the FFT to get this



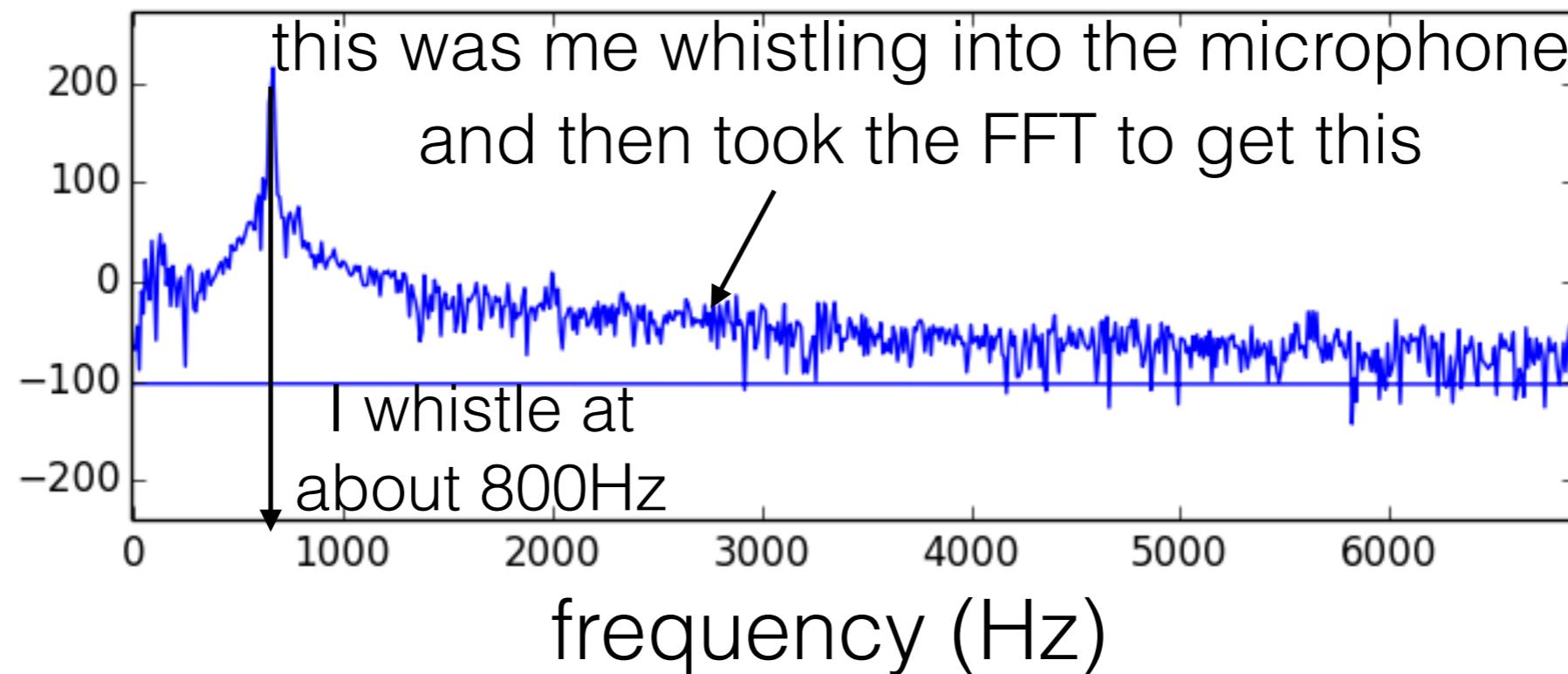
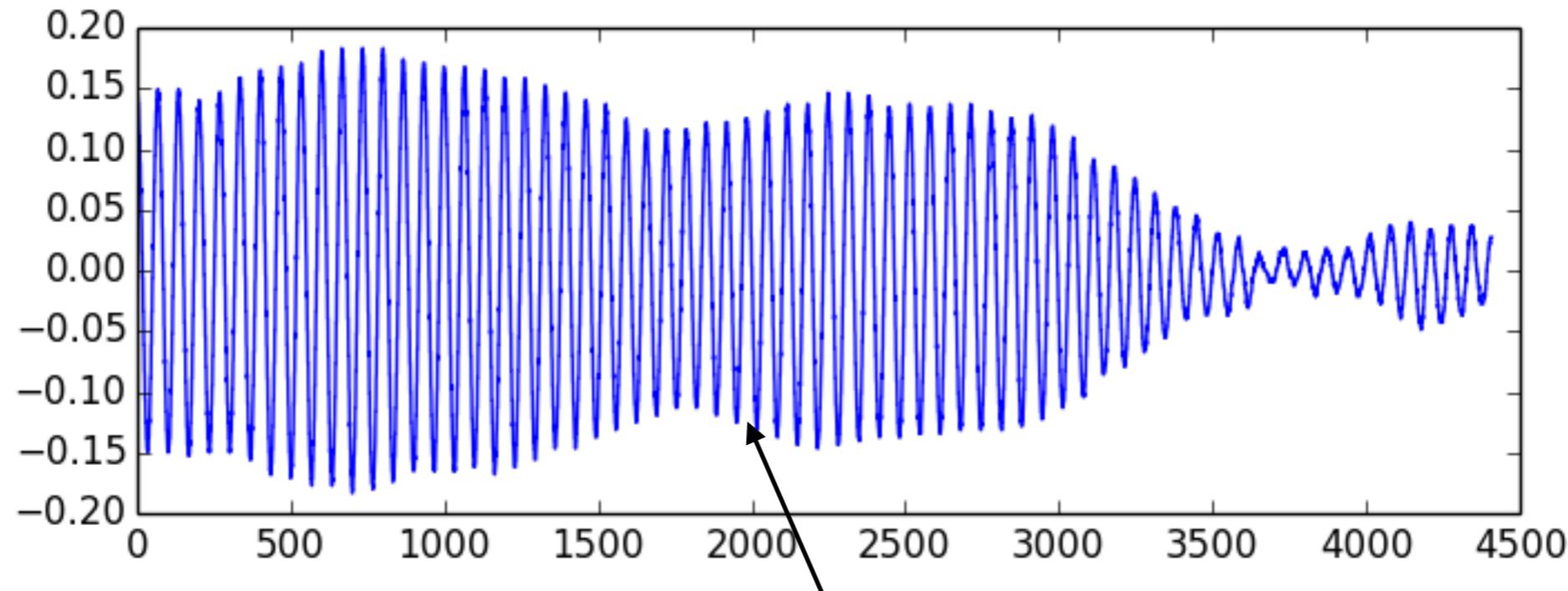
some fft examples

1kHz sine wave + 2.7kHz sine wave



some fft examples

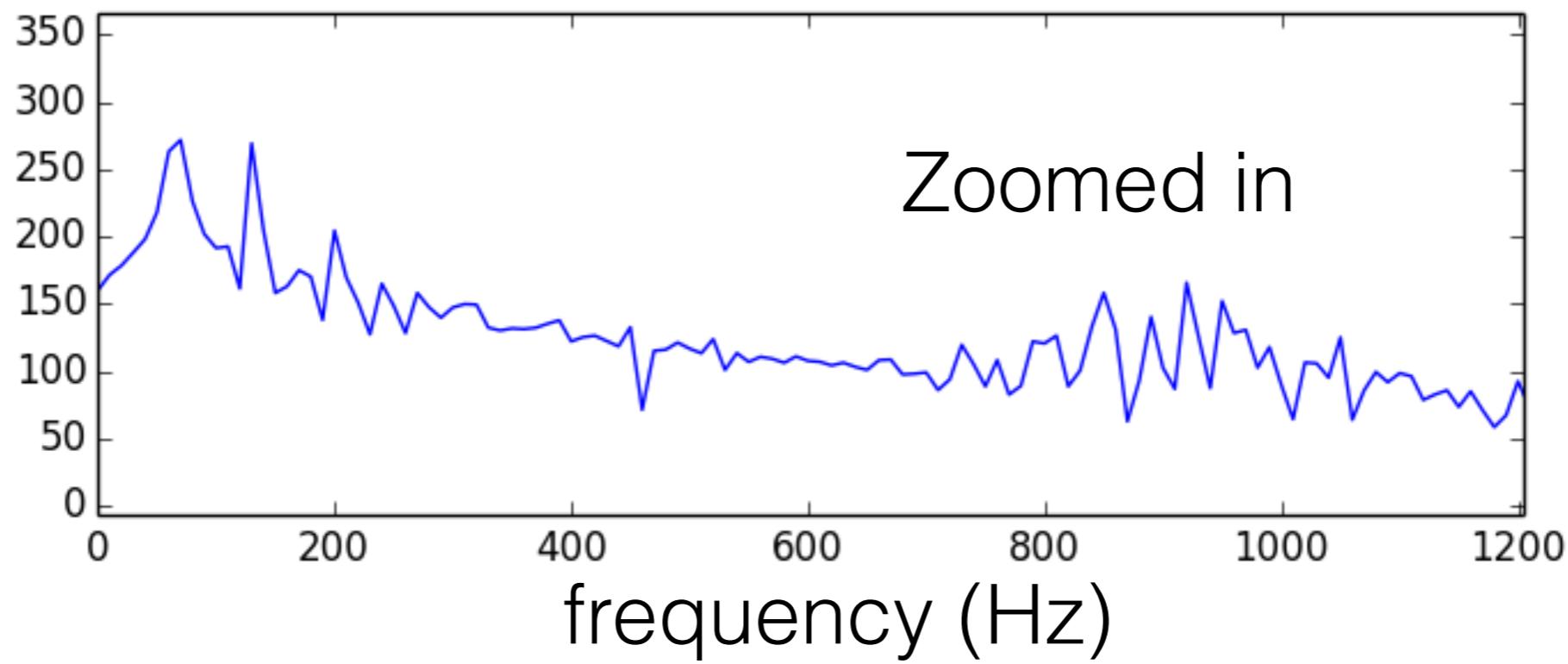
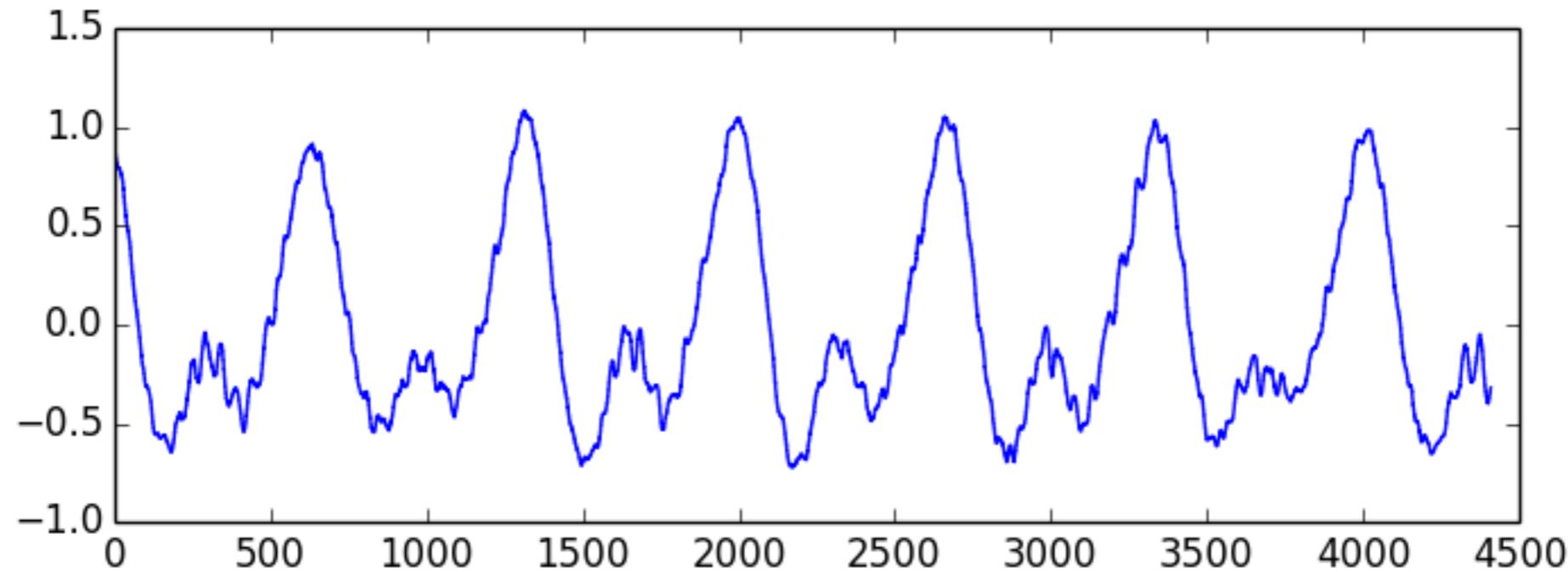
a person whistling



some fft examples

humming

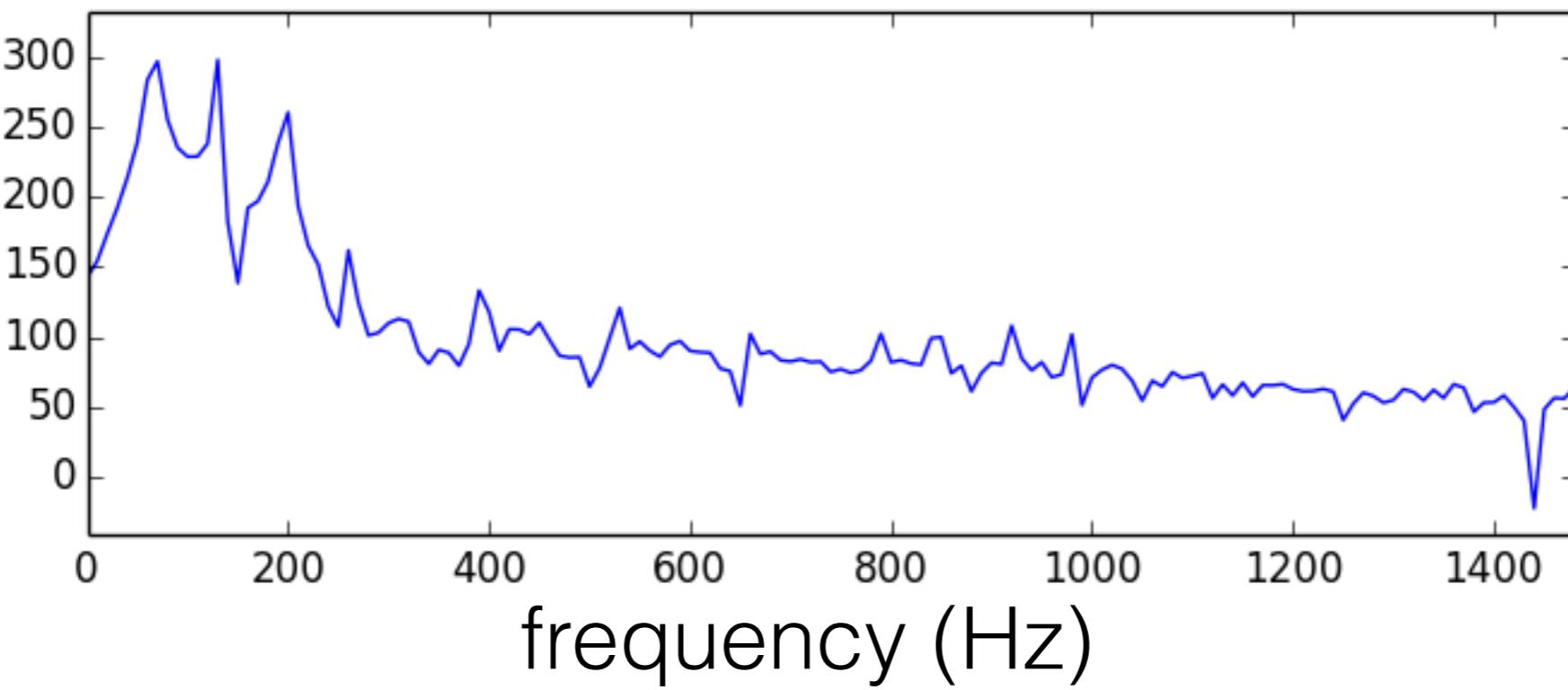
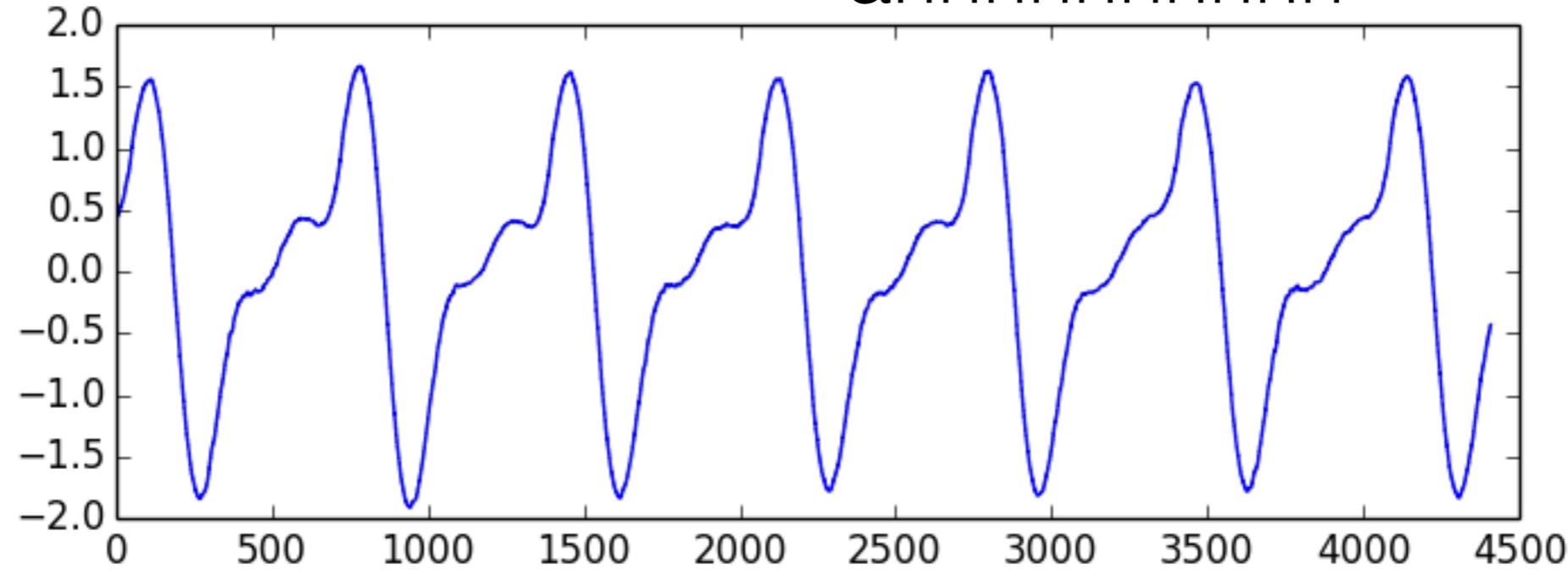
oooooooooooooo



some fft examples

humming again

ahhhhhhhhhh



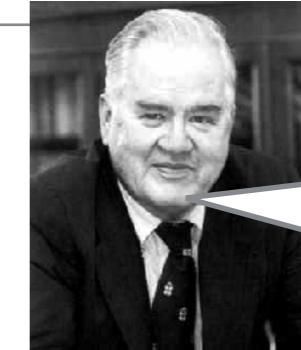
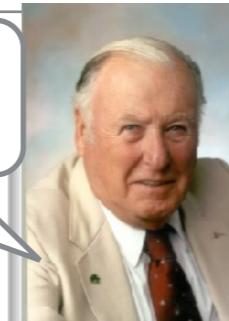
frequency (Hz)

programming the FFT

```
#import "SMUFFTHelper.h"
```

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

Yeah!



This is how to
use our stuff

```
//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

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);
```

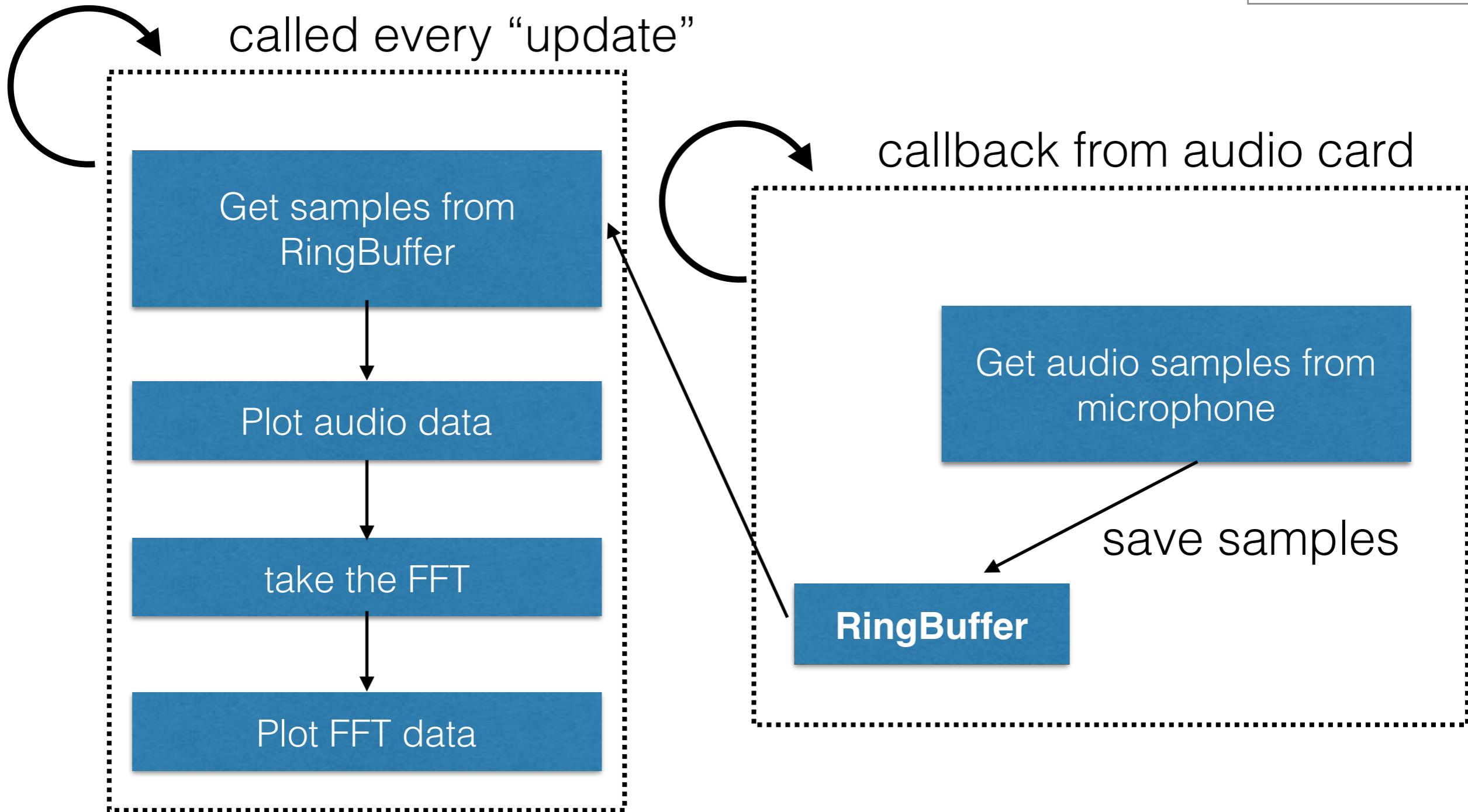
input array
N=window size

magnitude out
N=half window size

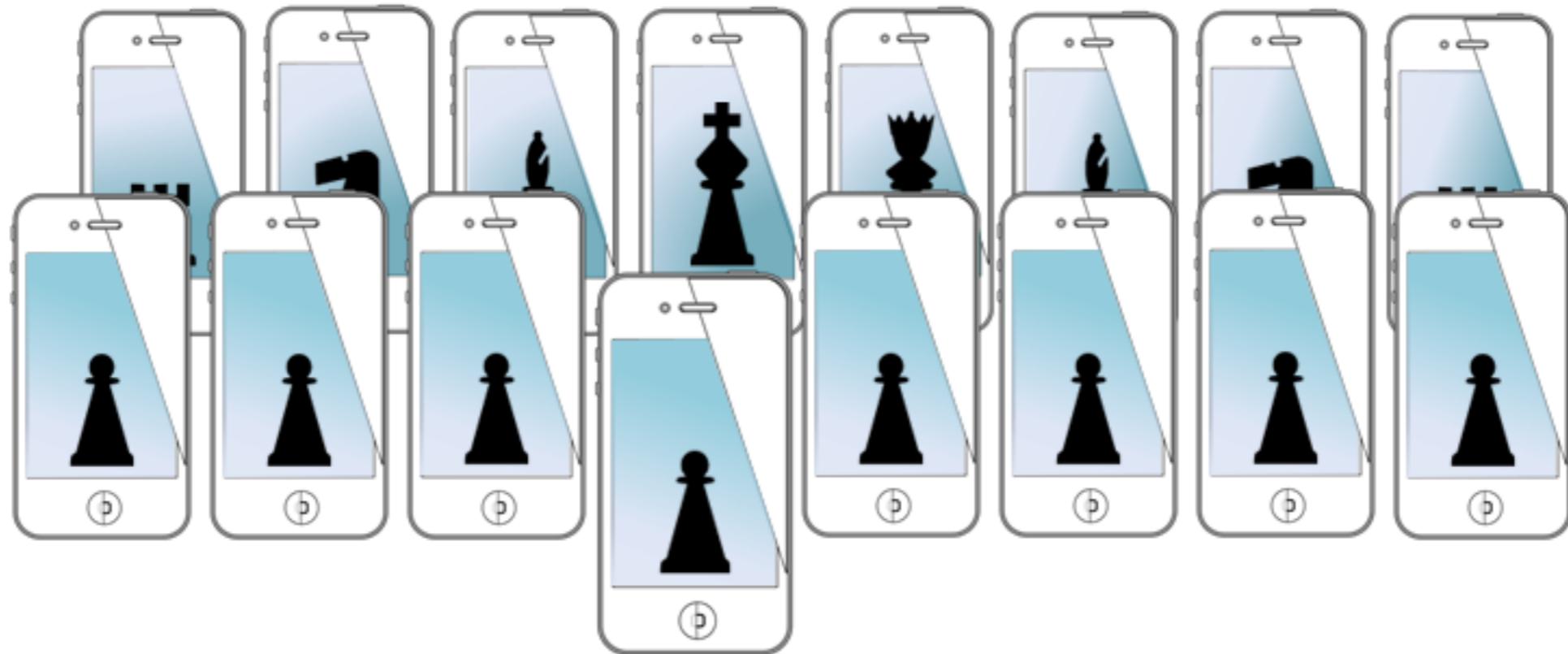
FFT tradeoffs

- 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 6 Hz accuracy?

The program



MOBILE SENSING LEARNING

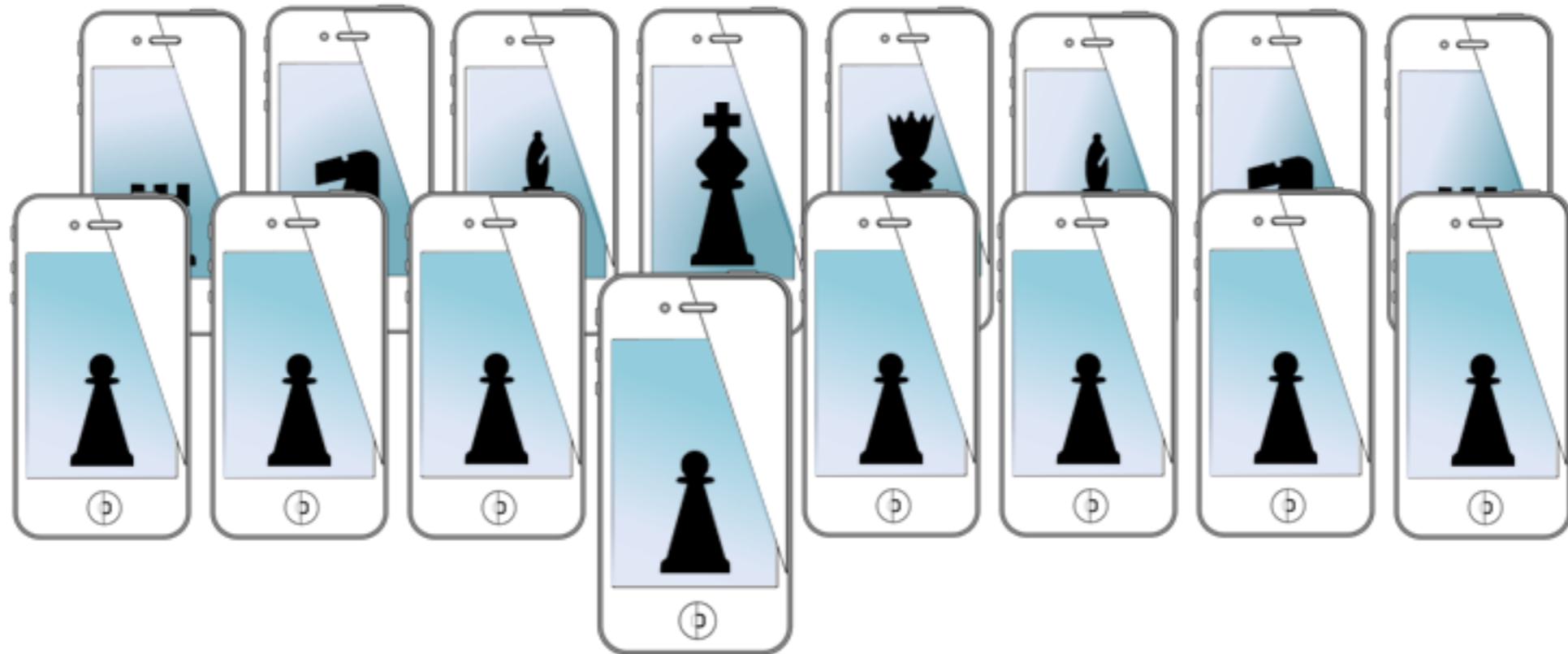


CS5323 & 7323
Mobile Sensing and Learning

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 $y[n]$ ”

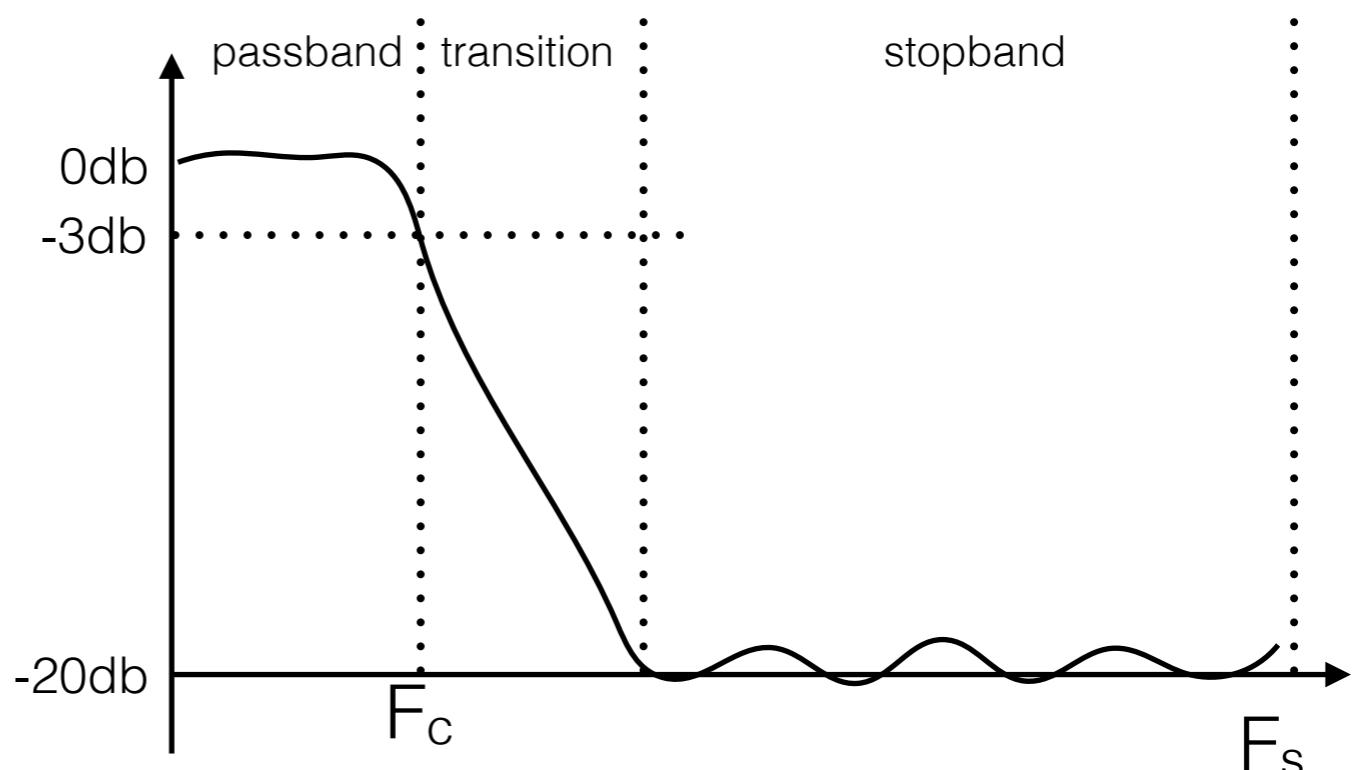


filters

- 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

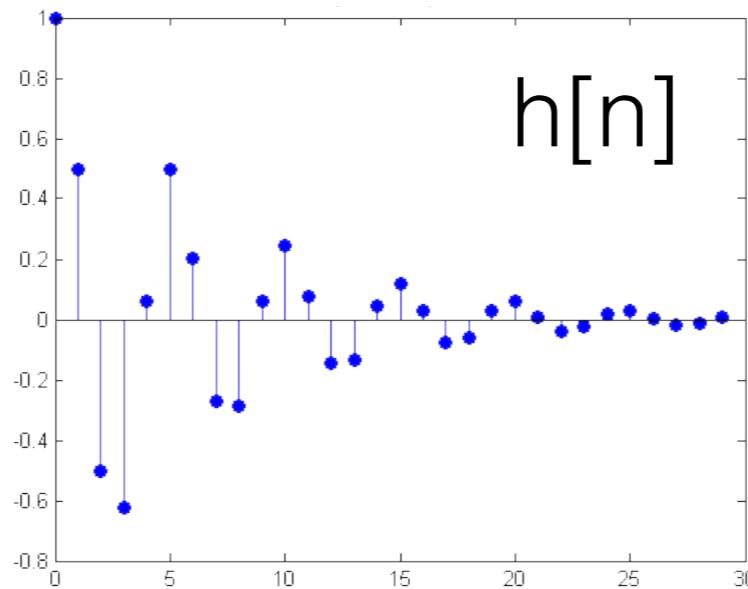
filters in frequency

- 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!

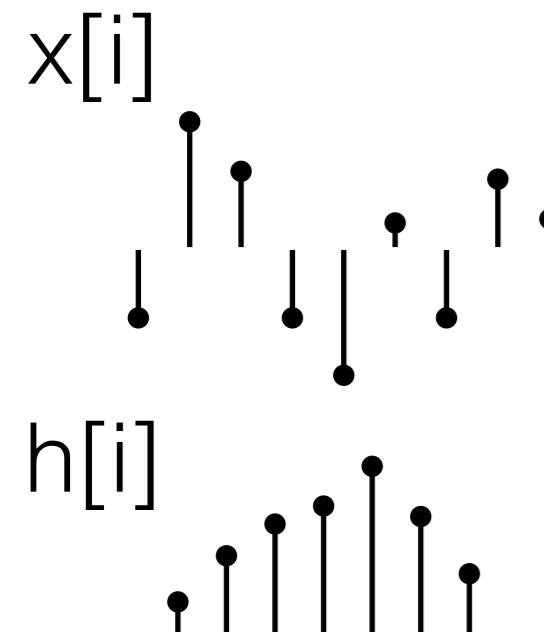
```
from pylab import *
import scipy.signal as signal
n = 61
a = signal.firwin(n, cutoff = 0.3,
                  window = "hamming")
```

filtering by convolution

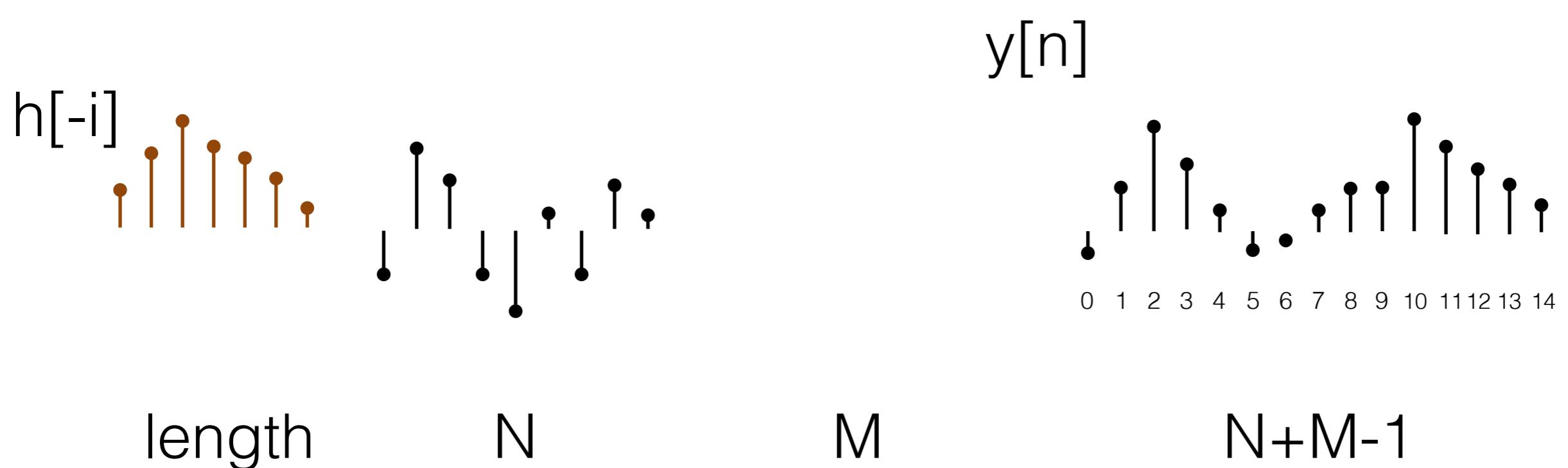
- we apply a filter using **convolution**
 - convolution allows us to combine frequency properties of two signals without taking an FFT
- basic principle:
 - convolution in time is multiplication in frequency
 - so the filter's frequency response will be multiplied by the frequency response of the signal

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

convolution

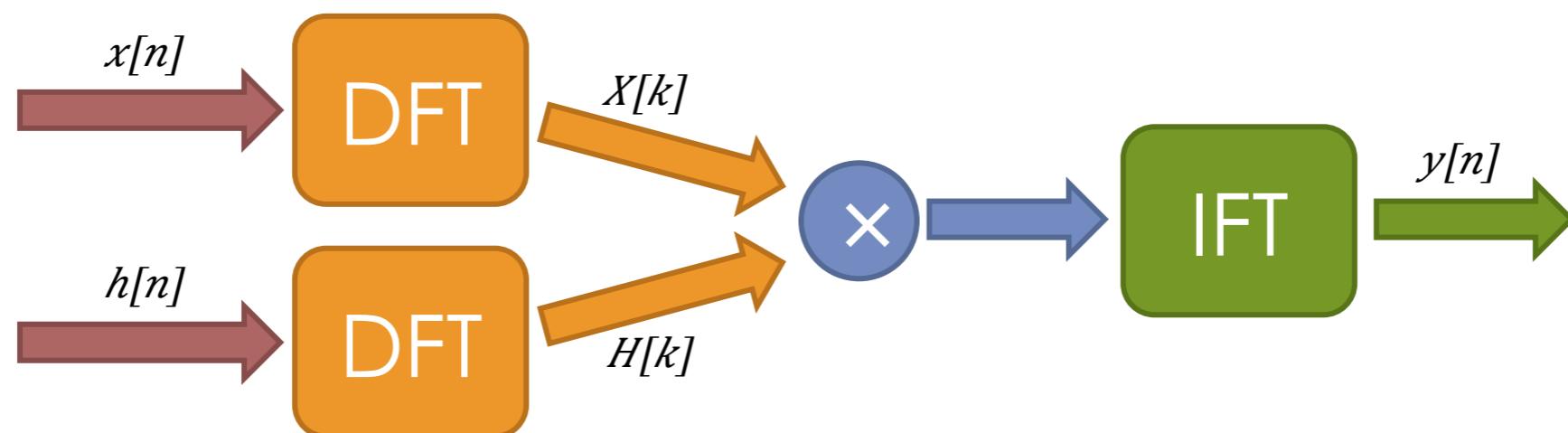


$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$



convolution efficiency

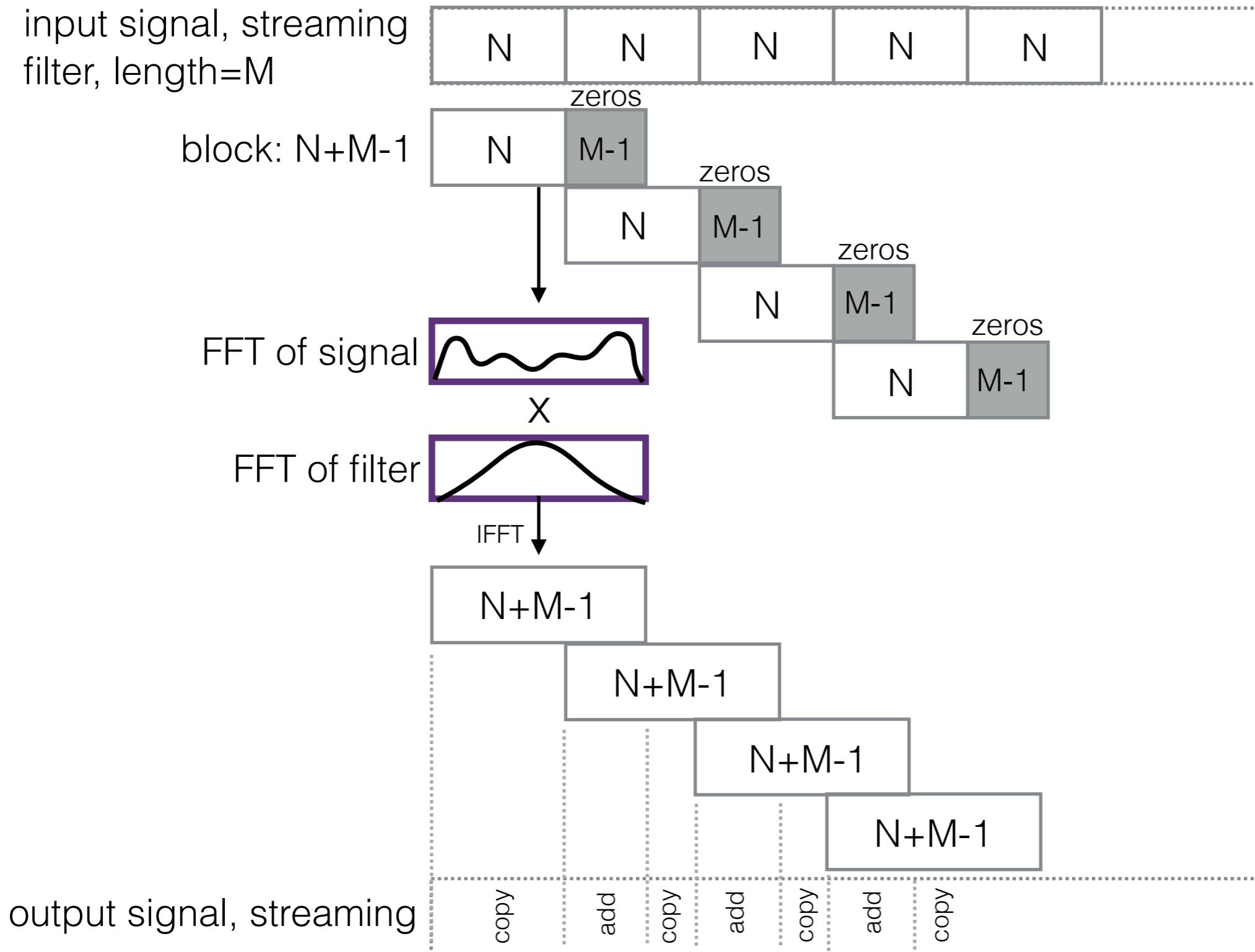
- algorithmic complexity
 - convolution is not particularly efficient, $O(N \times M)$
- to convolve faster, use that Fourier property:
 - “convolution in time is multiplication in frequency”
- why not just multiply to begin with!



its circular

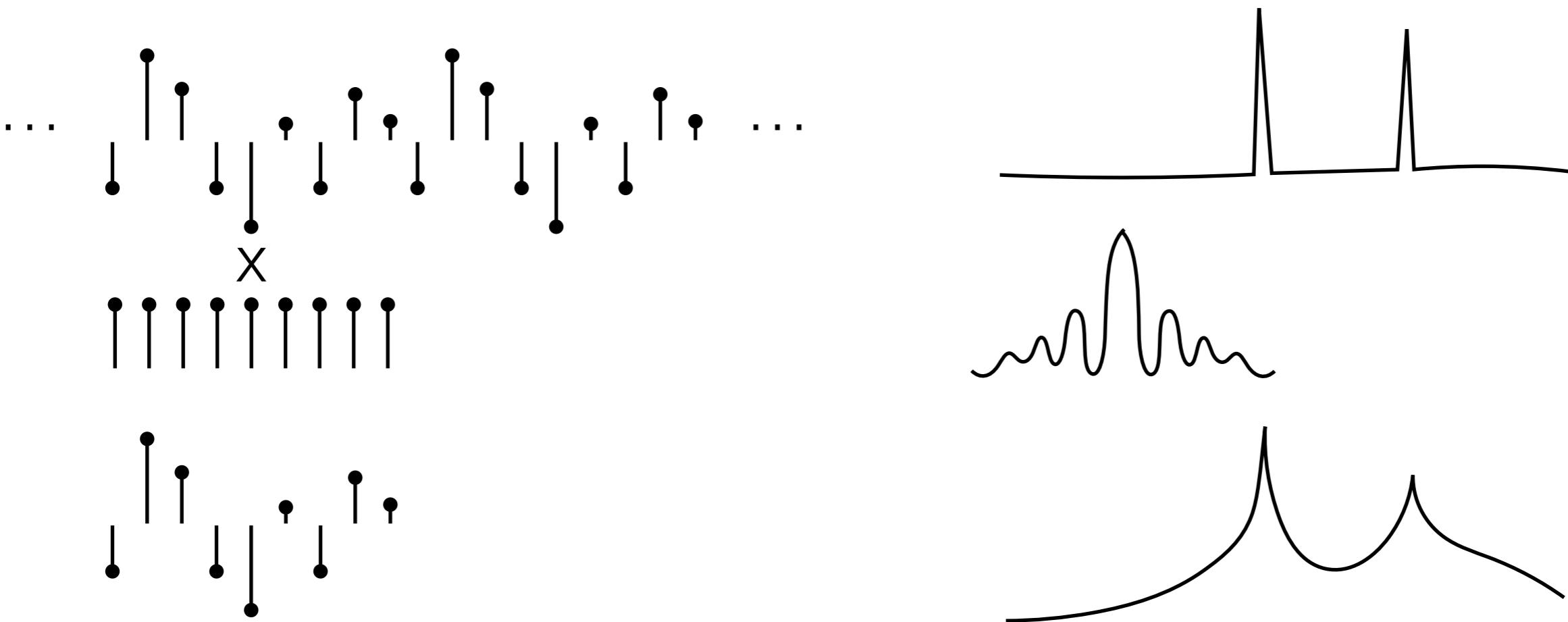
- just using N point FFT performs “Circular Convolution”
 - which is not linear convolution
 - causes the tail end of the convolution to “wrap around” to the beginning
 - FFT assumes the function is periodic (we did not talk about this)
- be aware of circularity when filtering your signal with the FFT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

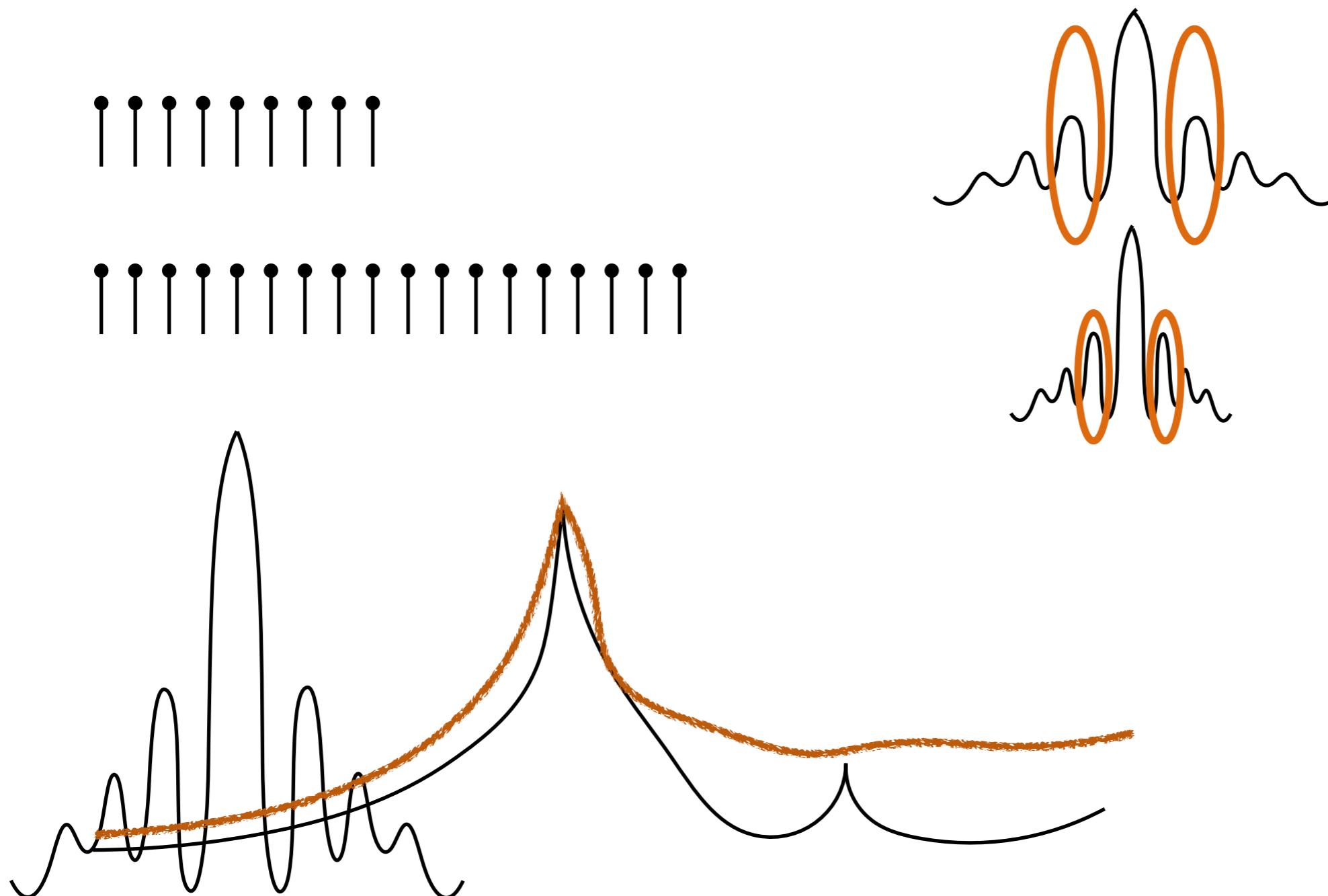


windowing: spectral BW widening

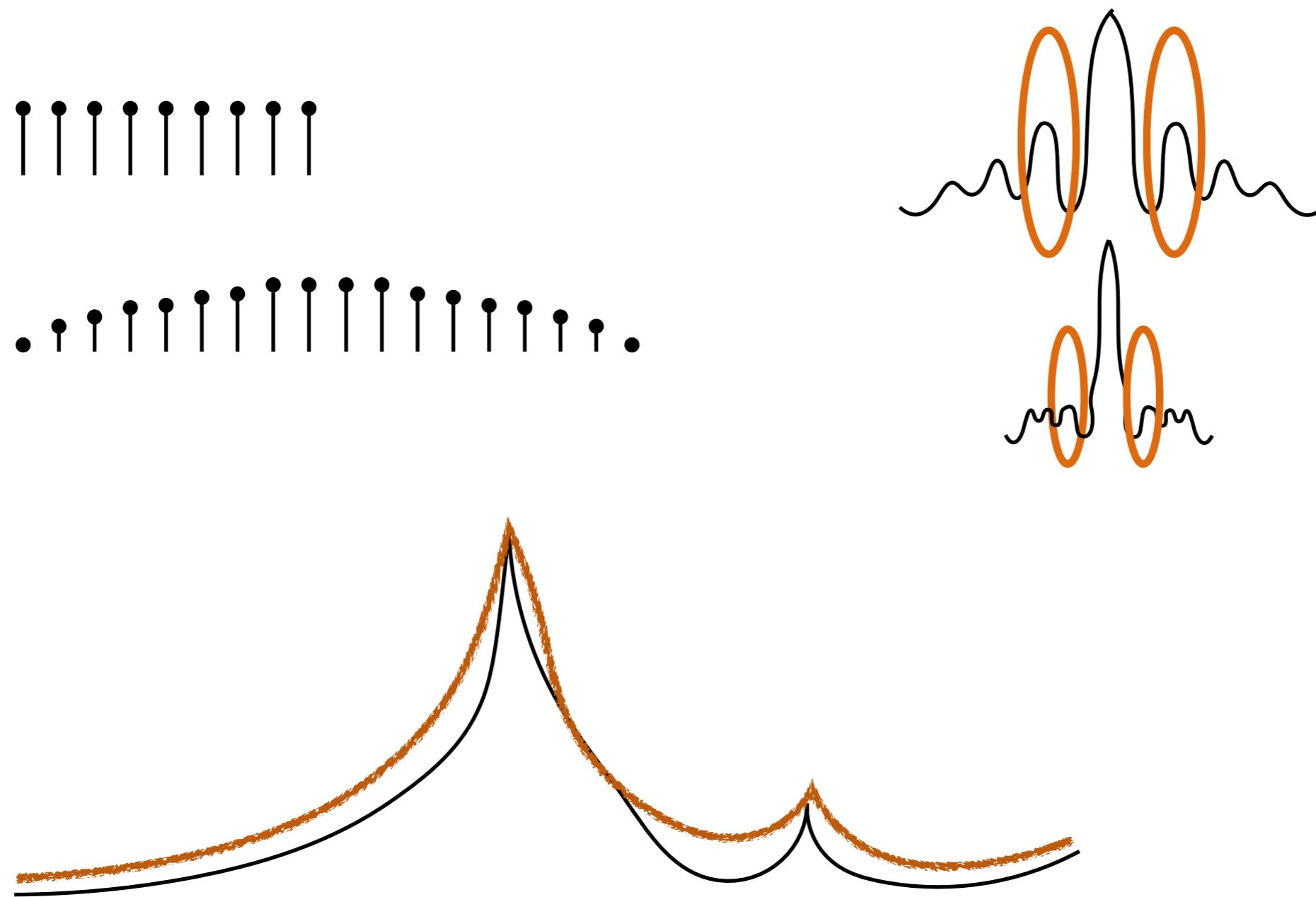
- multiplication in time is convolution in frequency
- a window is something we multiply in time with our signal
- windowing is unavoidable
 - why? we cannot take an infinite FFT...



windowing: spectral leakage



windowing: spectral leakage



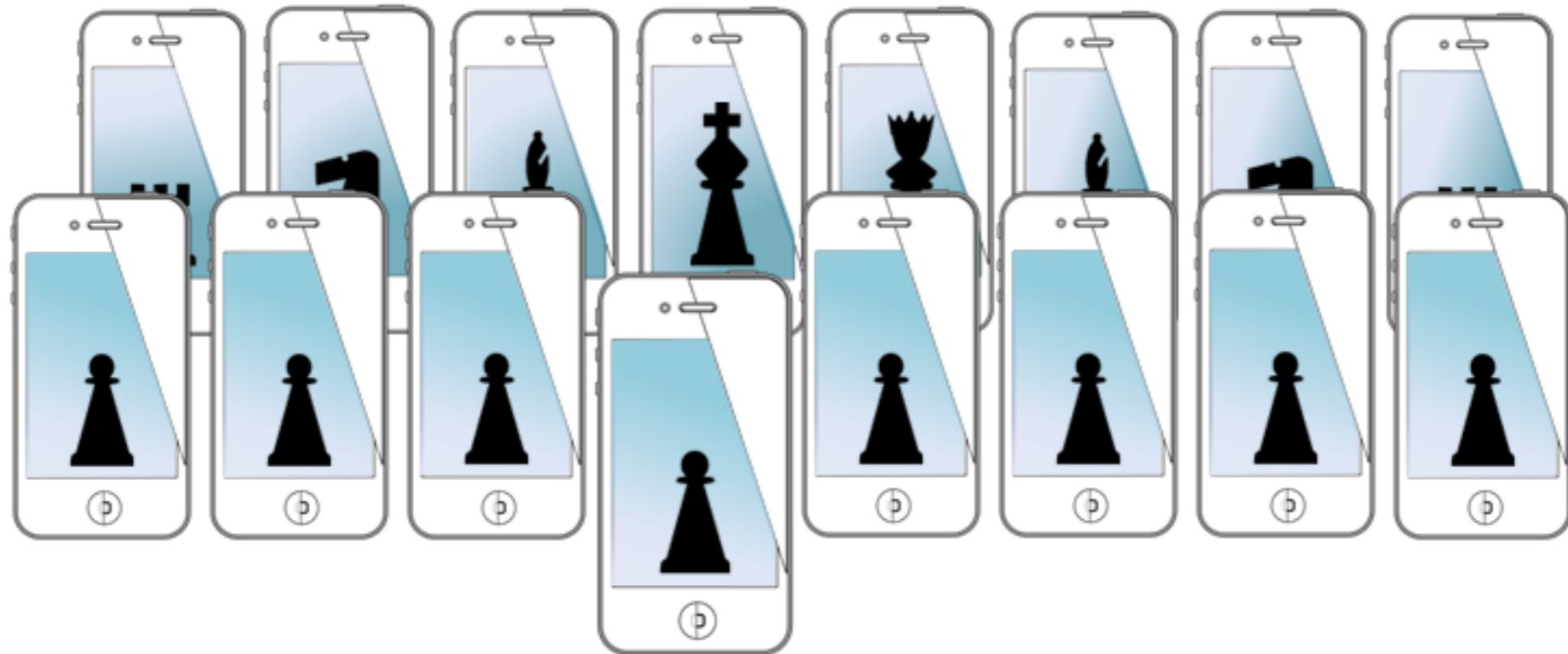
which window to use?

- 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