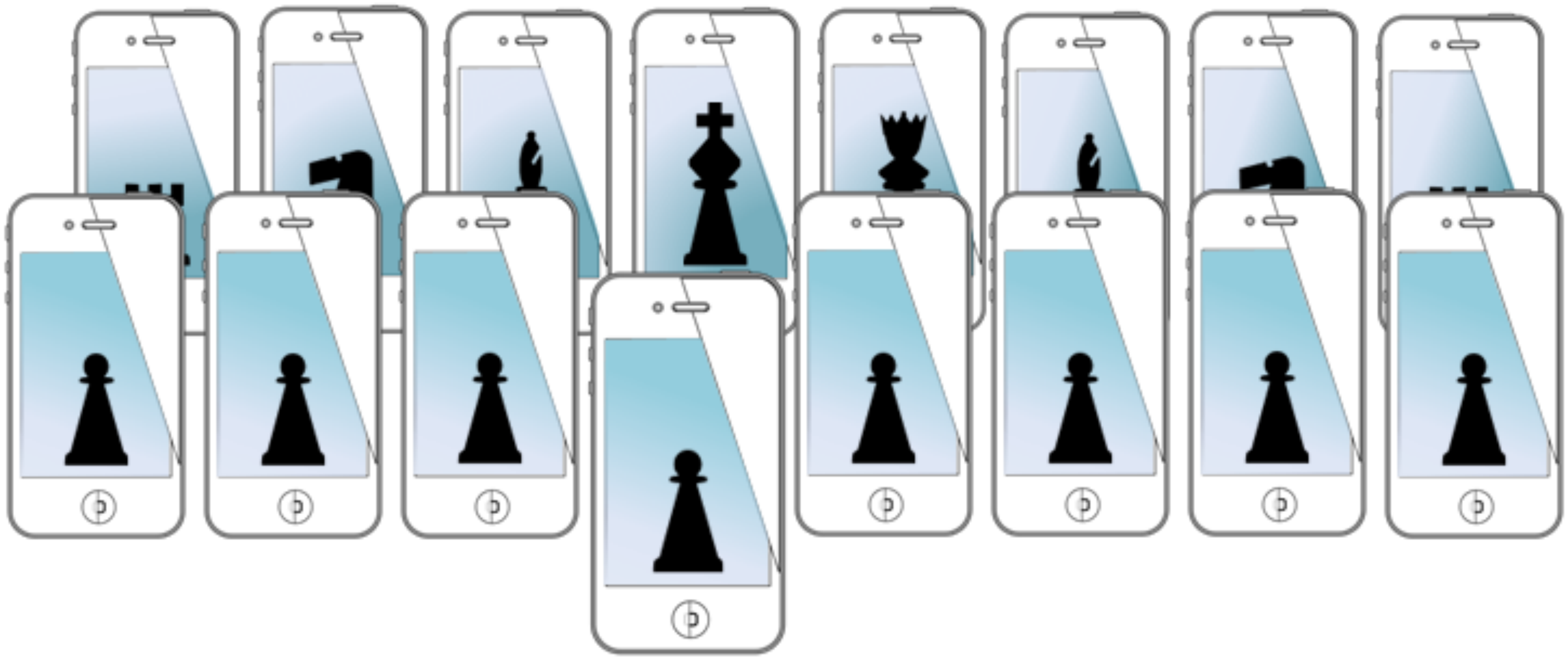
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

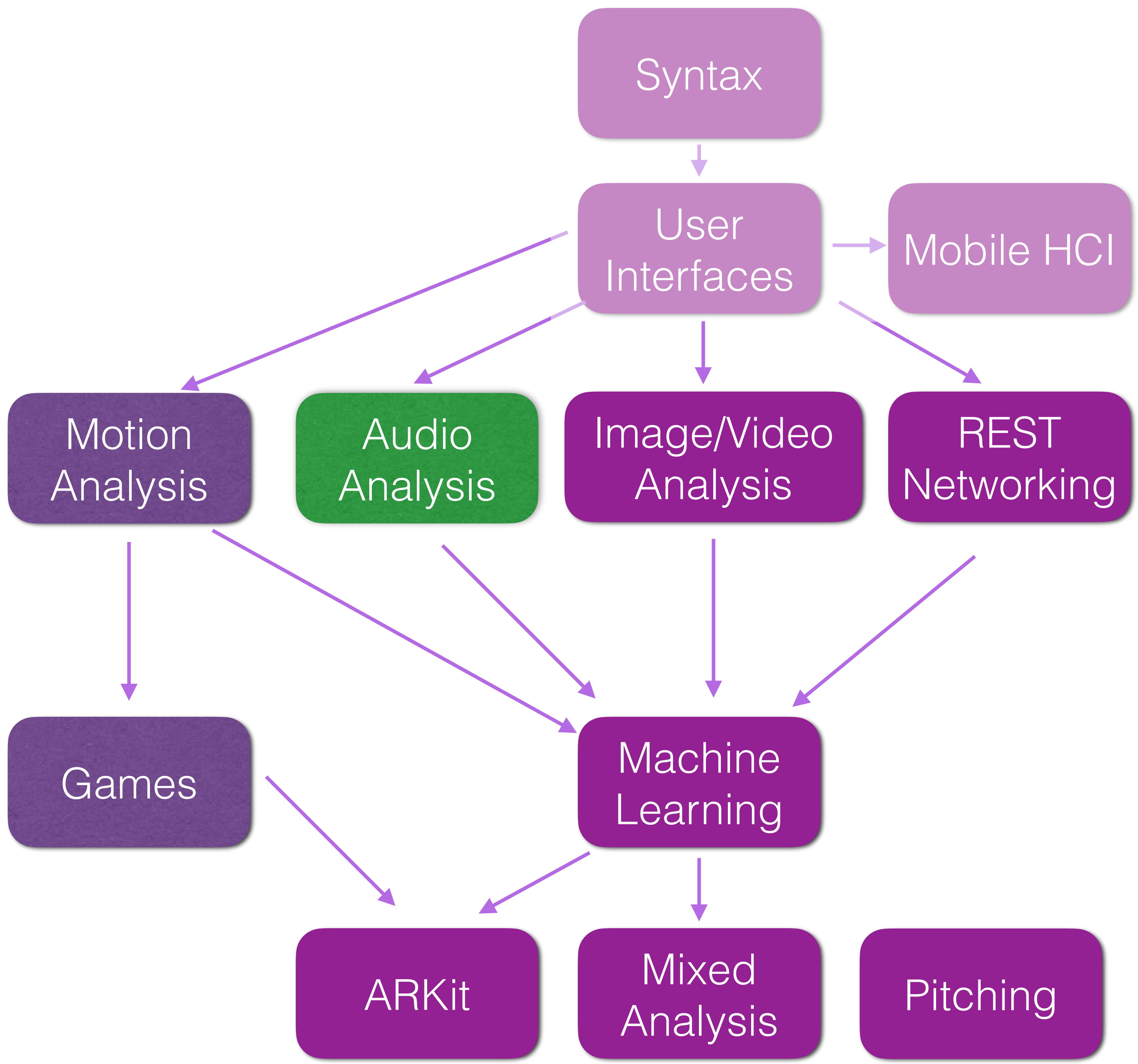
audio graphing, sampled data, & accelerate

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

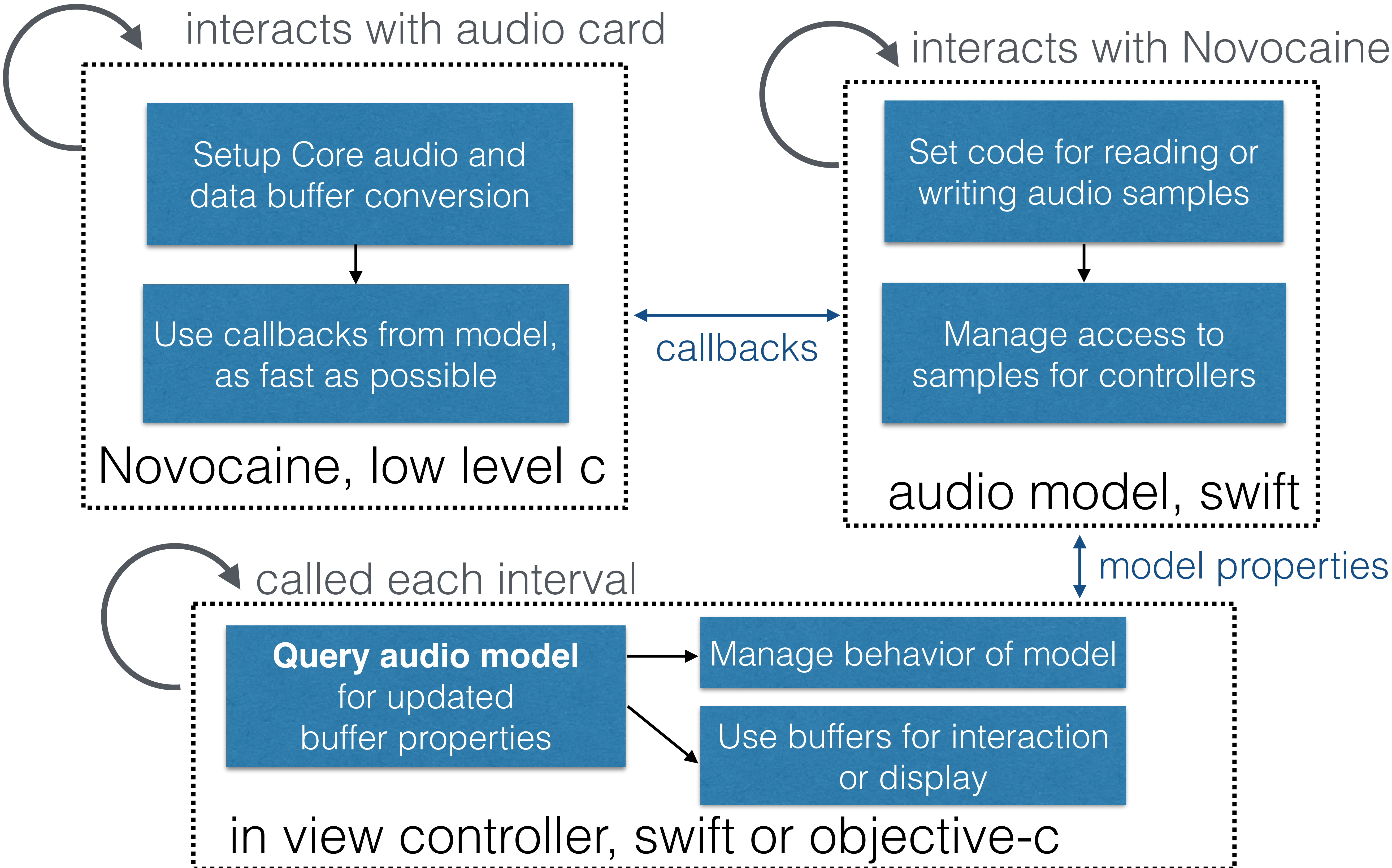
agenda and logistics

- logistics
  - **flipped module on audio next time!**
  - TA Office Hours: See canvas
- 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

class overview



review: MVC with audio



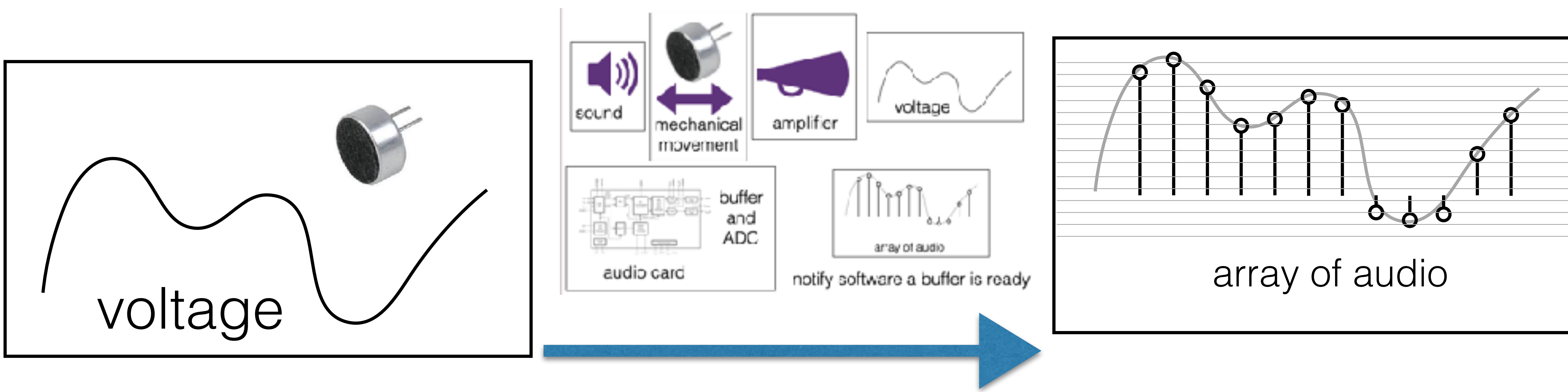
sample from the mic

- recall: data from the microphone on novocaine


```
private func handleMicrophone (data:Optional<UnsafeMutablePointer<Float>>,
                               numFrames:UInt32,
                               numChannels: UInt32) {
        if let arrayData = data{

            // 🎙️ -> 📉 get max element in the buffer
            // bonus: vDSP example (will cover in next lecture)
            // example with iOS accelerate to quickly handle the array
           var max:Float = 0
            vDSP_maxv(arrayData, 1, &max, vDSP_Length(numFrames))
            print(max)

        }

    }
```

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

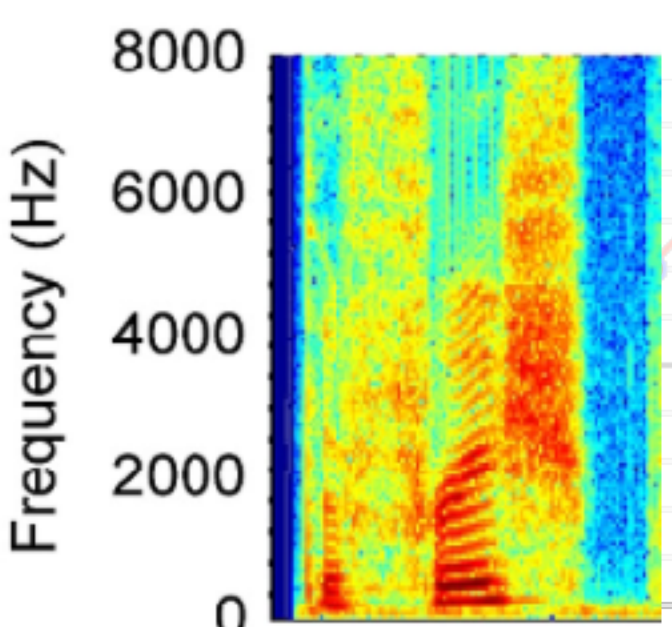


sampled data

- 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
  - **Nyquist**: if capturing an “F”Hz signal, need to sample at least 2 x “F” Hz
  - changing sample rates is complicated



for example, speech

- majority of 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

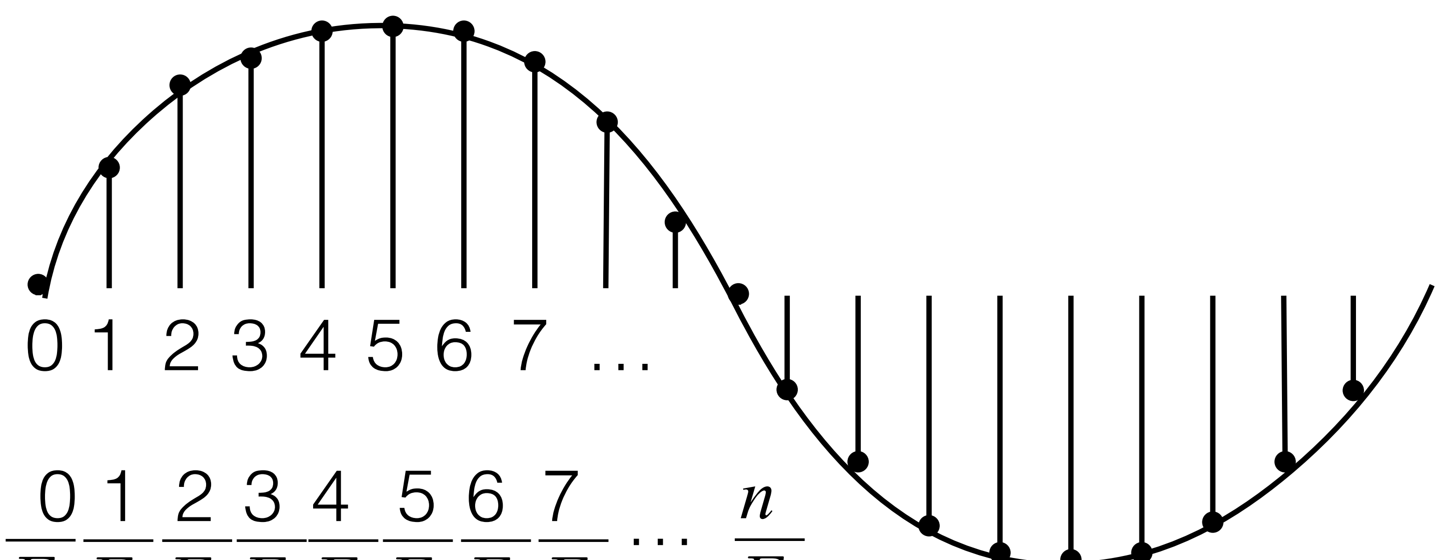
$$g(t) = \sin(2\pi f t)$$ equation for sine wave

f, frequency in Hz

t, time in "seconds"

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

Output Array in Novocaine `g:Optional<UnsafeMutablePointer<Float>>`



$n =$ 0 1 2 3 4 5 6 7 …

$t = \frac{0}{F_s} \frac{1}{F_s} \frac{2}{F_s} \frac{3}{F_s} \frac{4}{F_s} \frac{5}{F_s} \frac{6}{F_s} \frac{7}{F_s} \cdots \frac{n}{F_s}$

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

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?

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;
 }
```

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

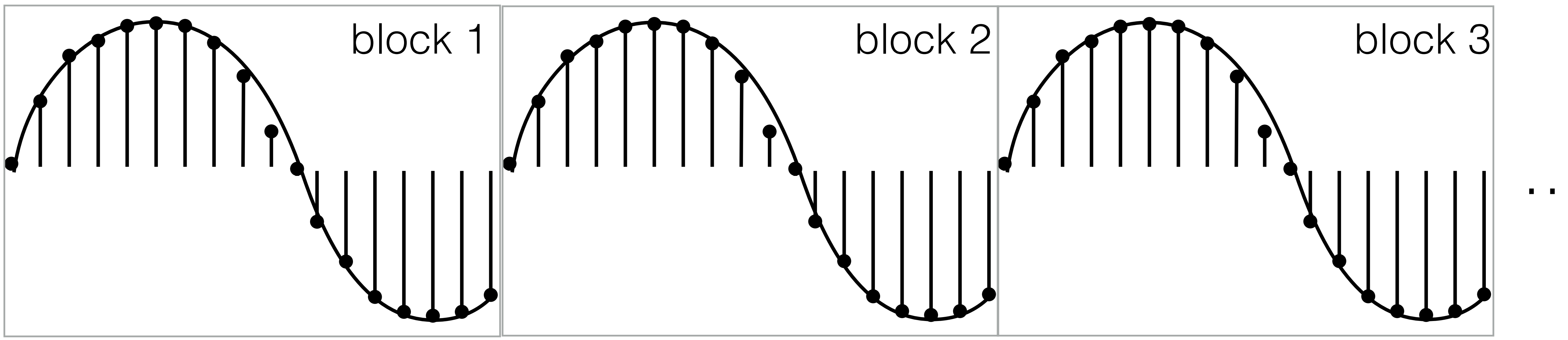
$$\theta \leftarrow \theta + \frac{2\pi f}{F_s}$$

sine wave discontinuity

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

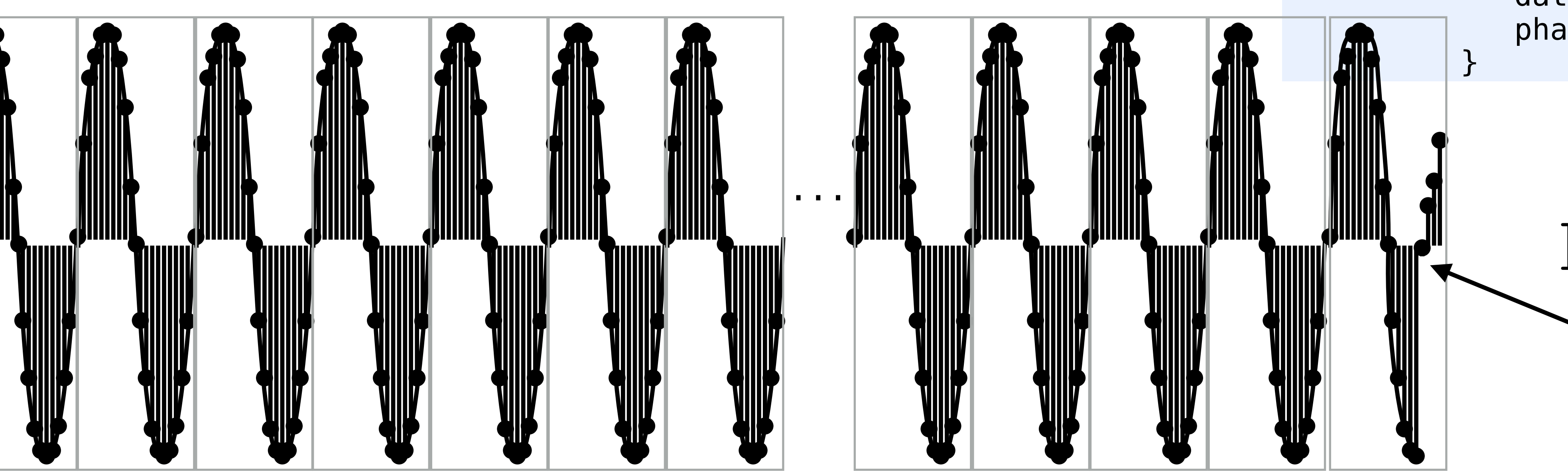
$$\theta \leftarrow \theta + \frac{2\pi f}{F_s}$$

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



`phase = 0` in each output block

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



`phase` variable overflows, discontinuity!

making a sine wave

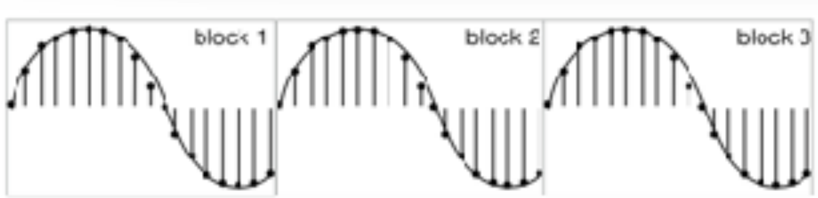
- bringing it all together

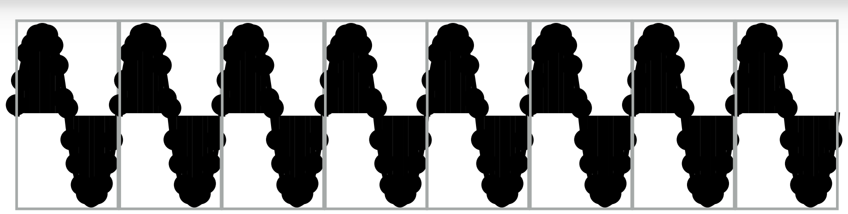
$$\theta \leftarrow \theta + \frac{2\pi f}{F_s} \qquad g[n] = \sin(\theta)$$

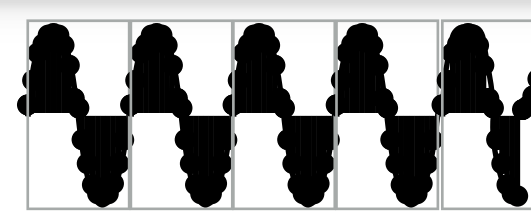
```
var frequency = 18000.0 //starting frequency
var phase = 0.0
var samplingRate = audioManager.samplingRate
let sineWaveRepeatMax = 2*Double.pi

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?







phase overflow

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
  - fast is the name of the game — it uses "c"
  - SIM… ngle Ins… ction, … ple Da…

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
);
```

https://developer.apple.com/documentation/accelerate/1450020-vdsp_vsmul

examples

what do each of these implement?

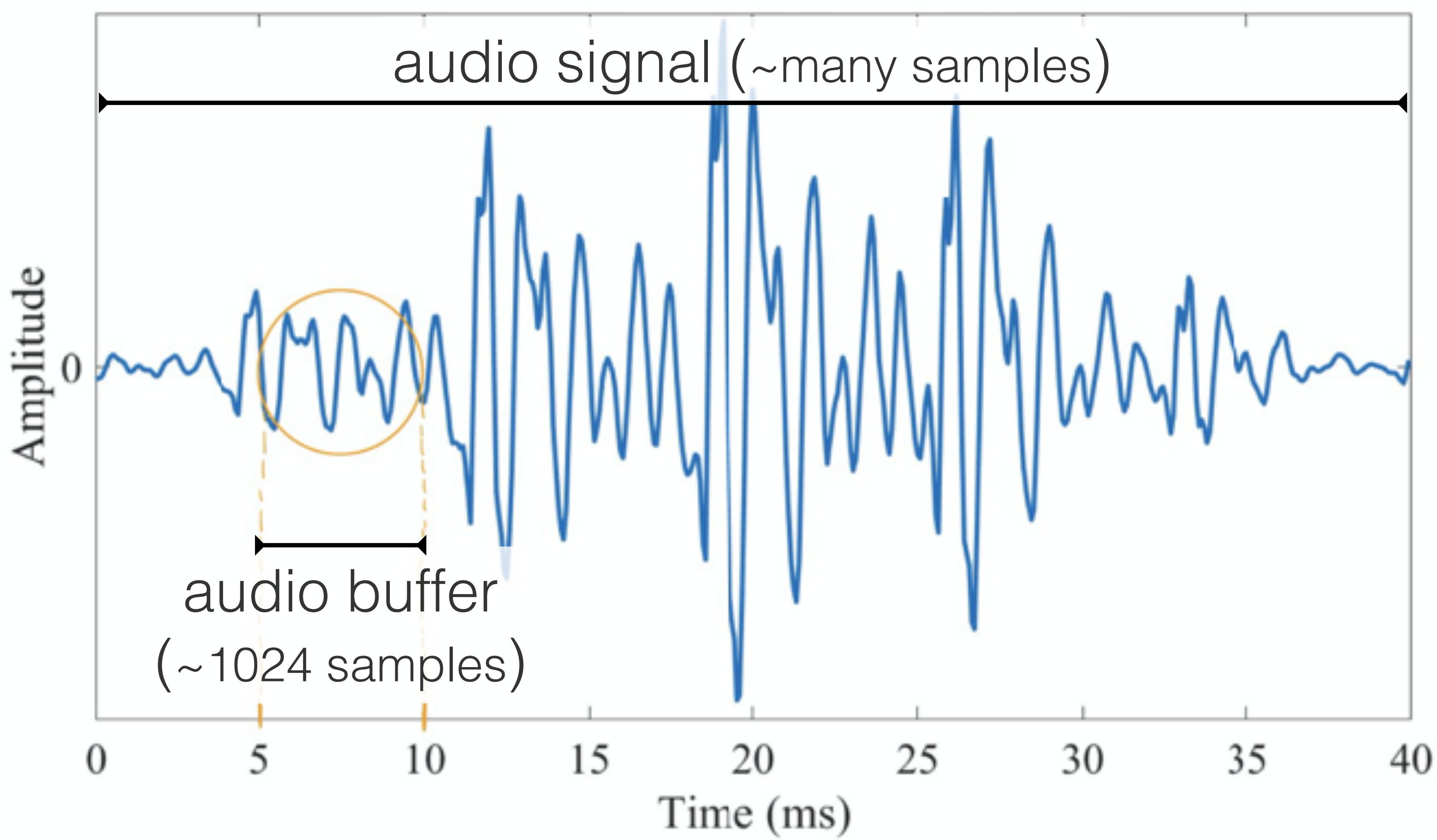
```
outputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32)) {
      ringBuffer.fetchFreshData(data, withNumFrames:numFrames)
      var volume = userSetMultiplyFromSlider;
      vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels)
  }
```

```
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 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

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"
)
```

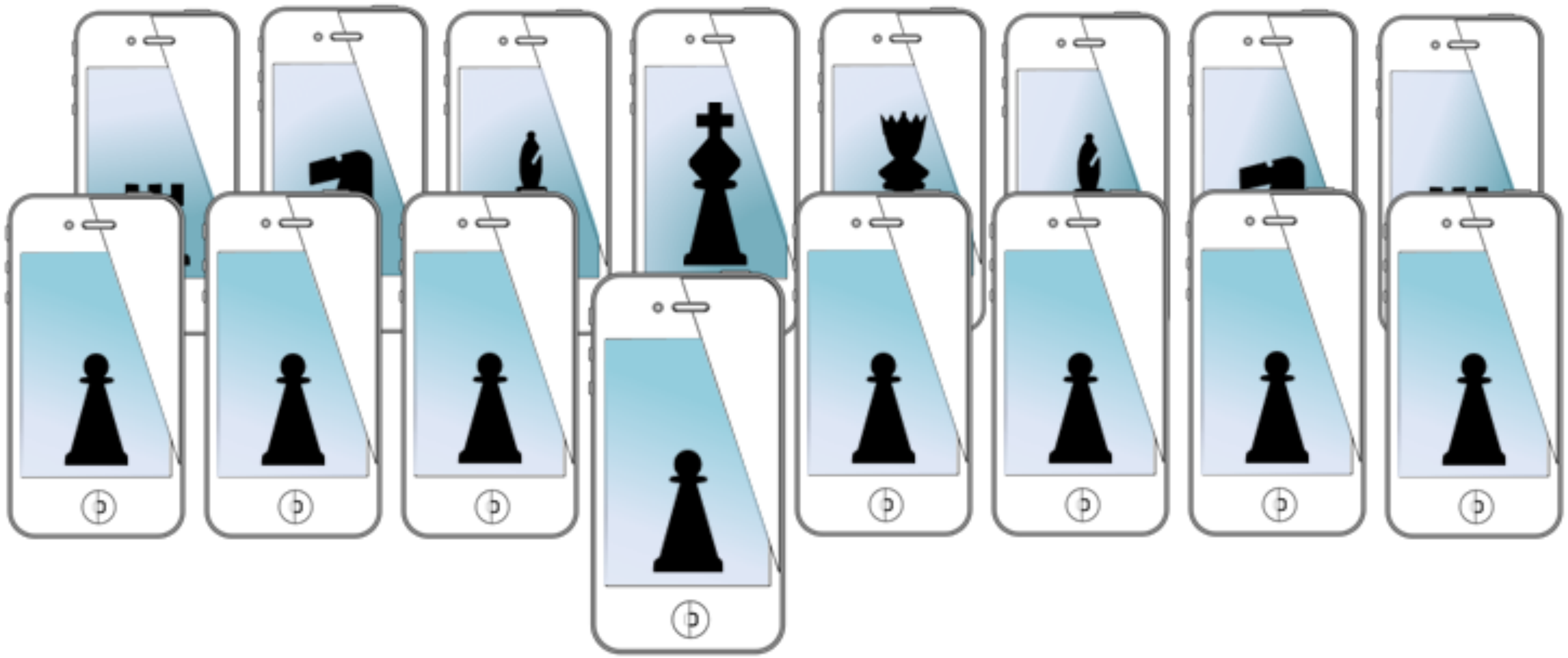
refresh data for each named graph key

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!

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

MOBILE SENSING LEARNING



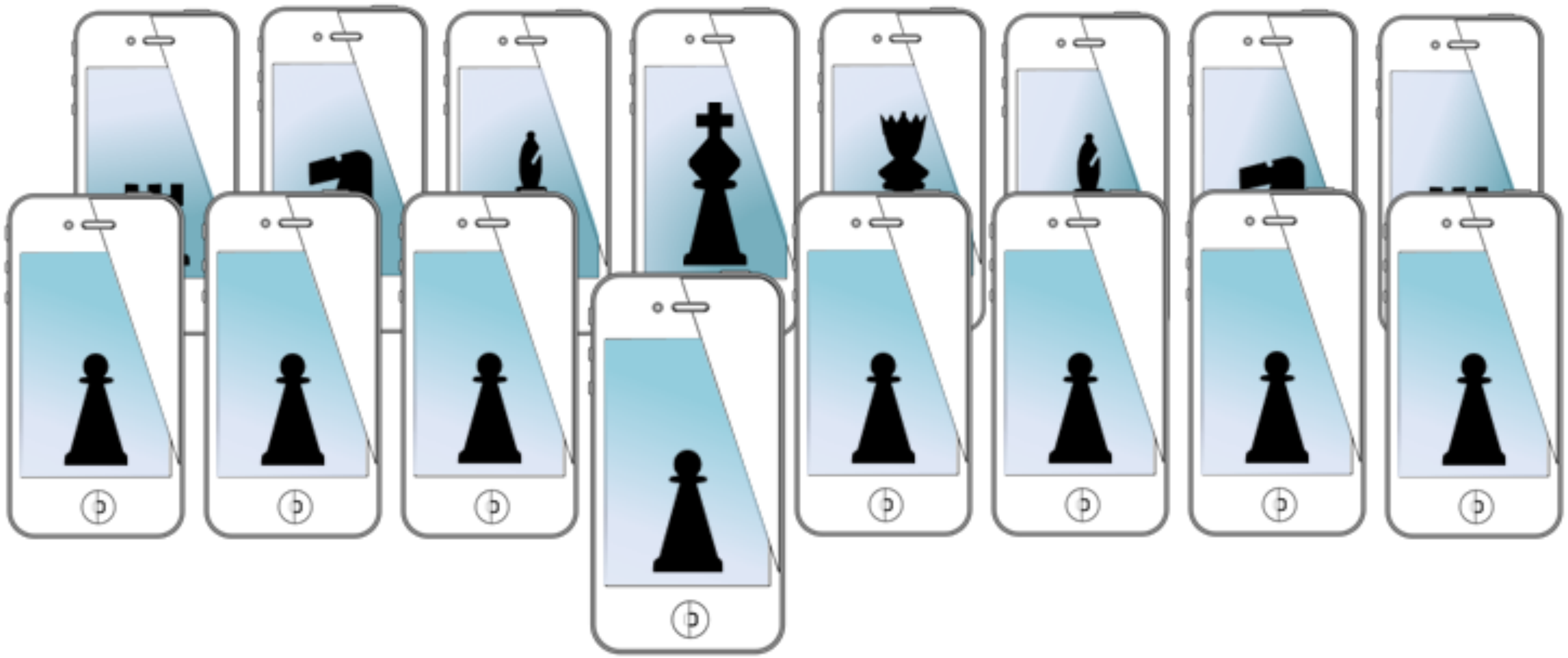
CS5323 & 7323

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

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

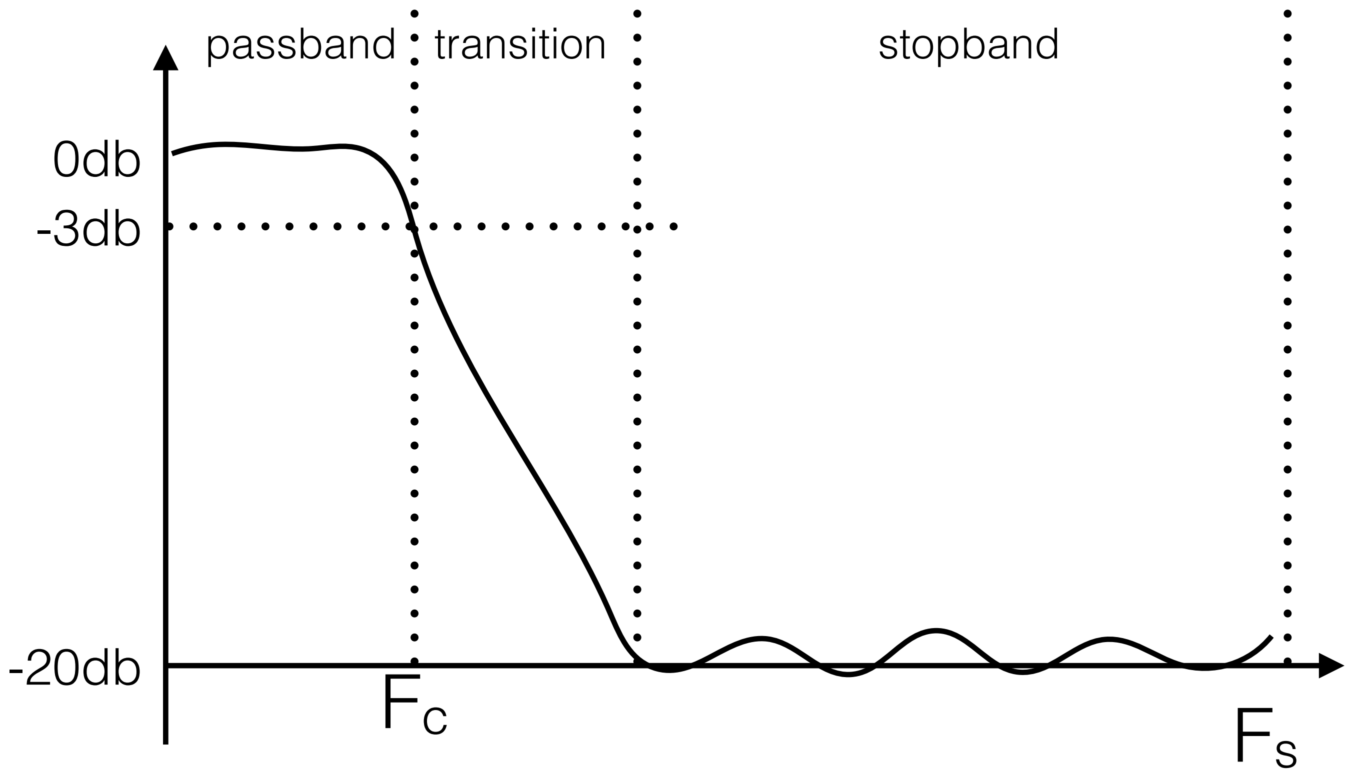
- systems are objects which manipulate signals
  - characterized by their "input/output" relationships
  - we say "x[n] is passed through H, resulting in y[n]"

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

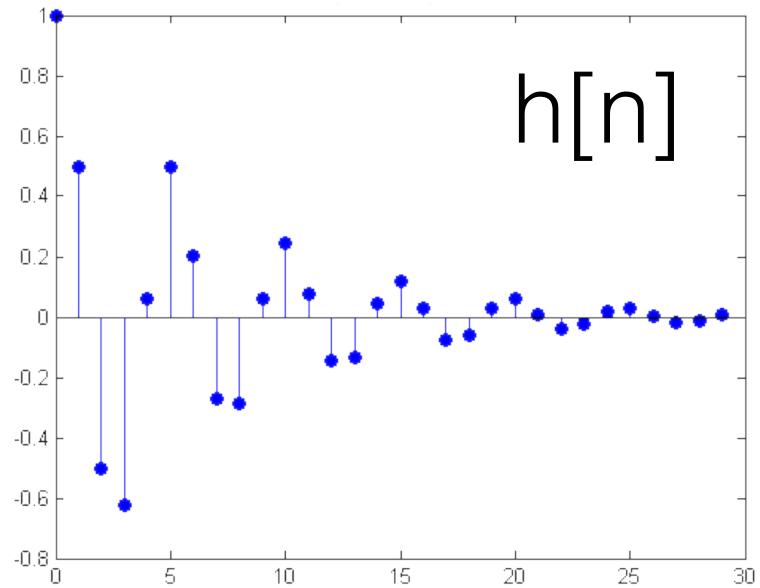
filters in frequency

- 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
- lots of other places!

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

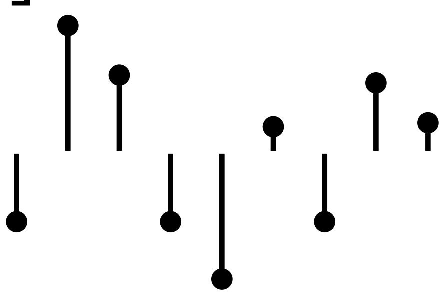
filtering by convolution

- 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

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

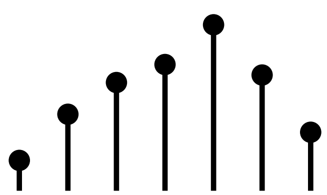
convolution

x[i]

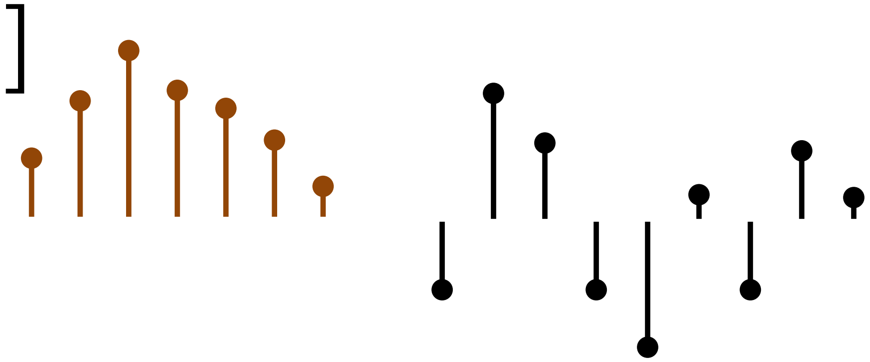


$$y[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

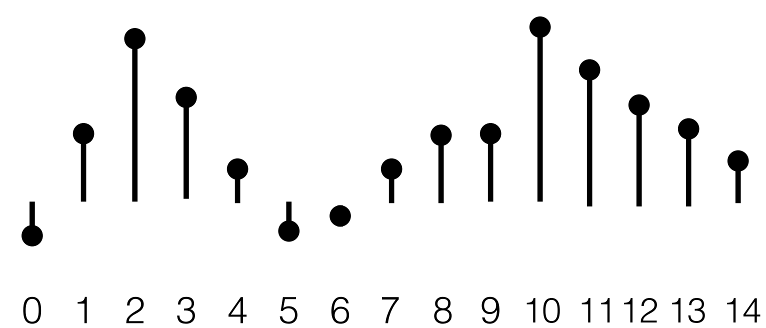
h[i]



h[-i]



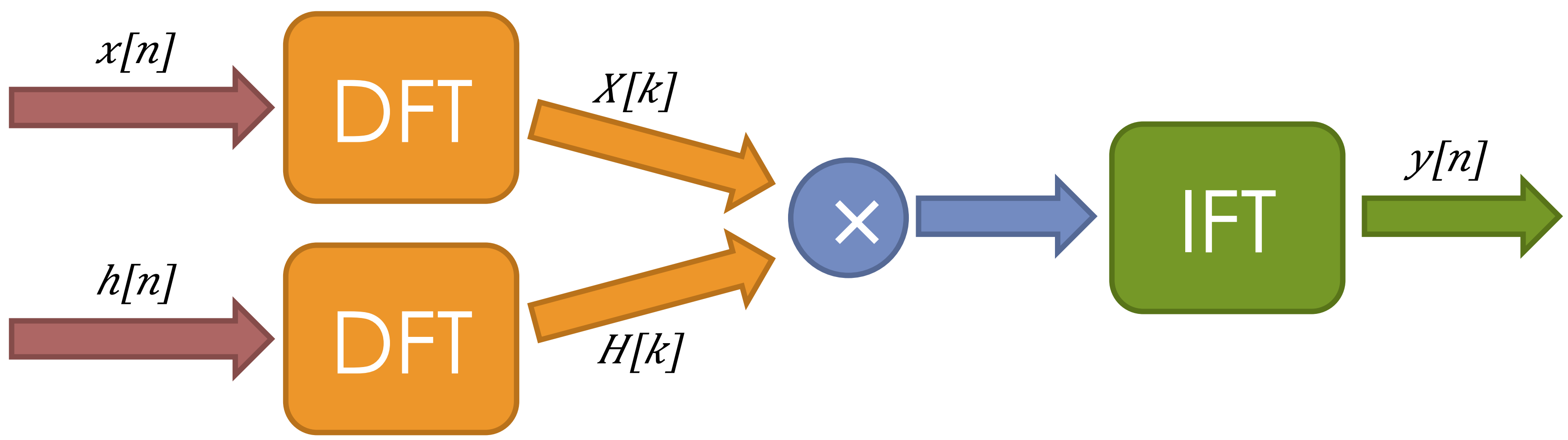
y[n]



length	N	M	N+M-1

convolution efficiency

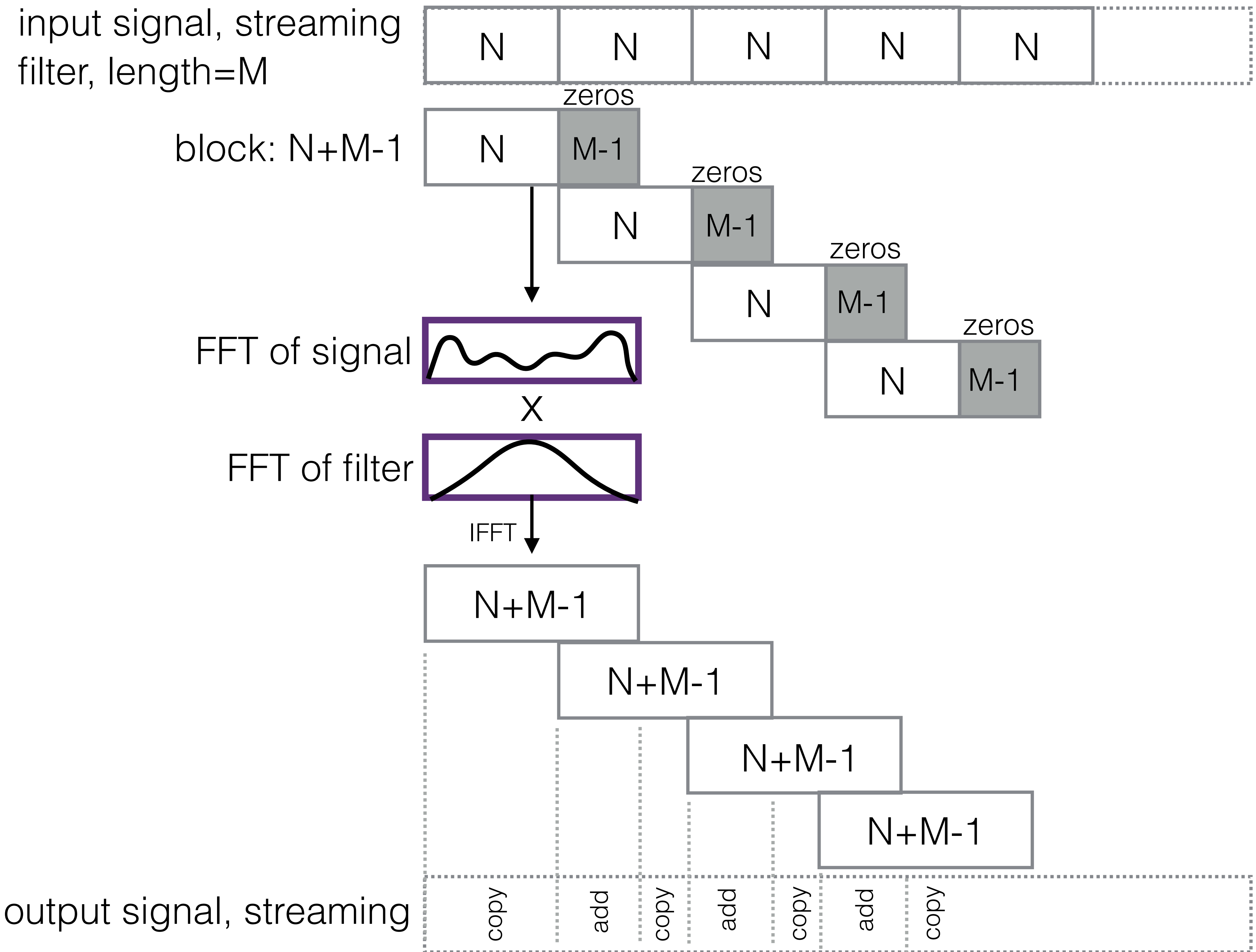
- algorithmic complexity
  - convolution is not particularly efficient, O(NxM)
- to convolve faster, use that Fourier property:
  - “convolution in time is multiplication in frequency”
- why not just multiply to begin with!



its circular

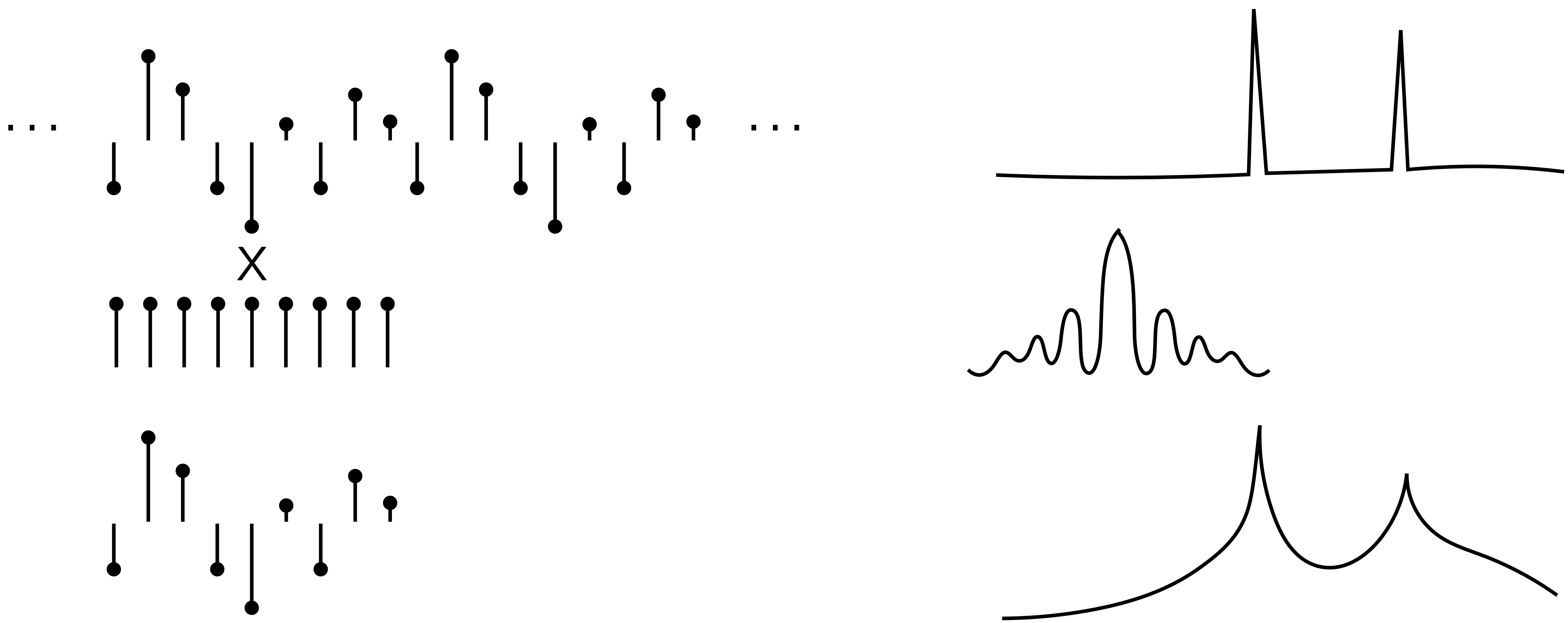
- just using N point FFT performs "Circular Convolution"
  - which is not linear convolution
  - causes the tail end of the convolution to "wrap around" to the beginning
    - FFT assumes the function is periodic (we did not talk about this)
- be aware of circularity when filtering your signal with the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

overlap and add

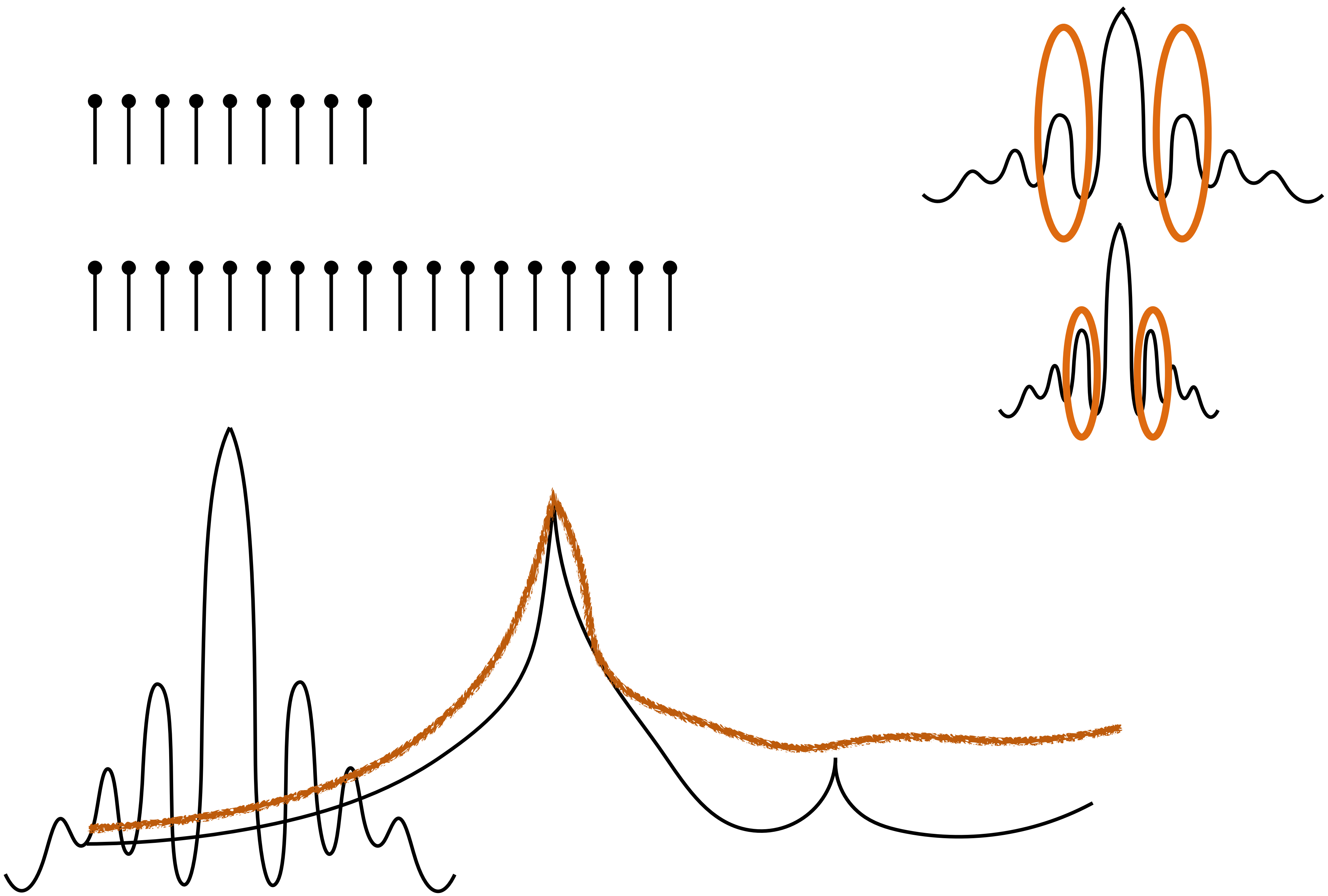


windowing: spectral BW widening

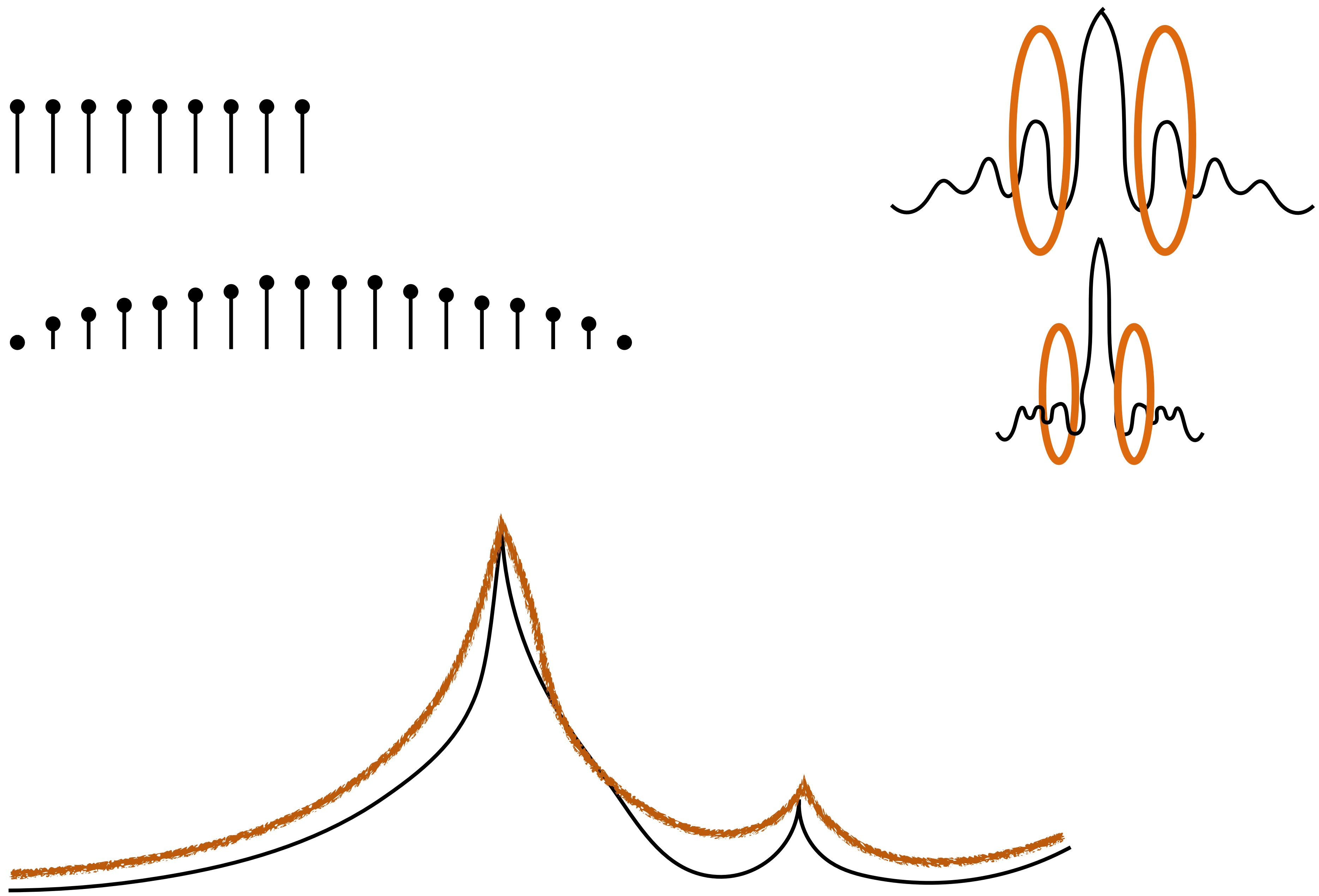
- multiplication in time is convolution in frequency
- a window is something we multiply in time with our signal
- windowing is unavoidable
  - why? we cannot take an infinite FFT…



windowing: spectral leakage



windowing: spectral leakage



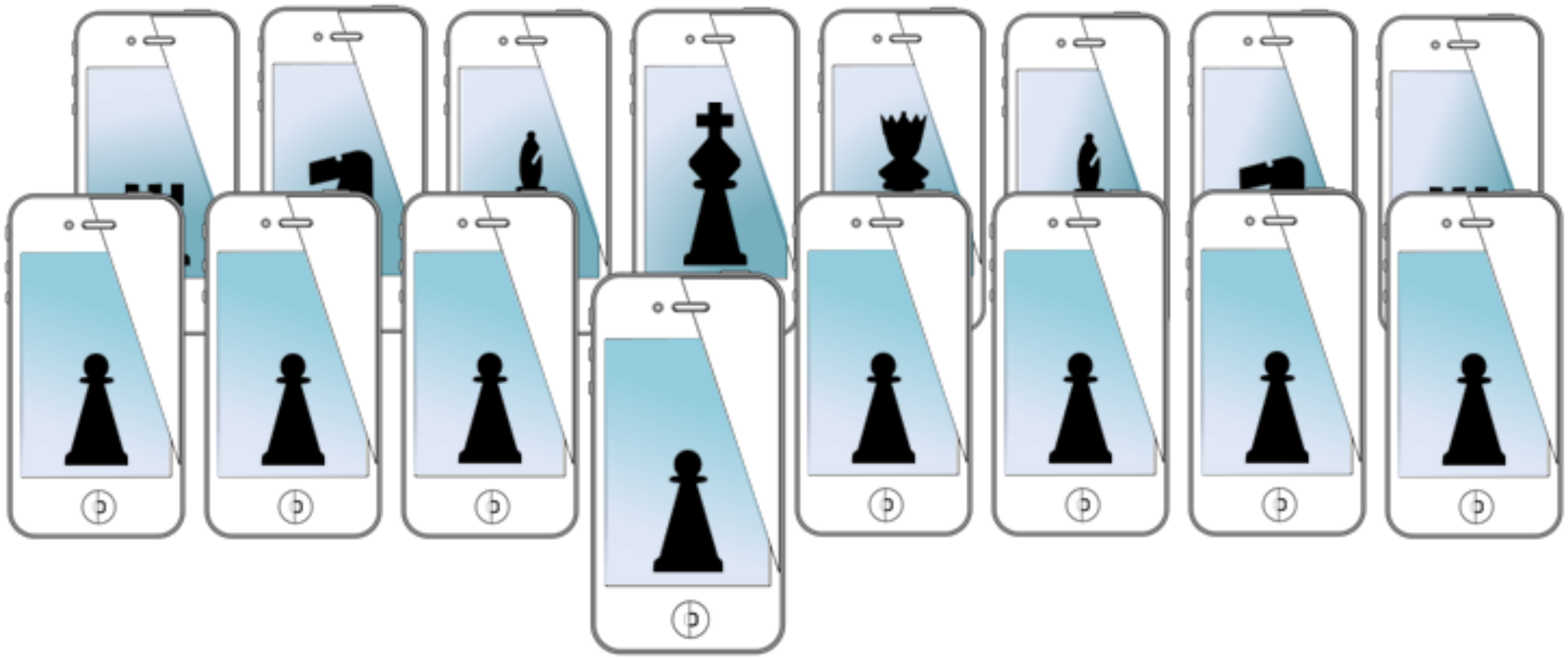
which window to use?

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

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