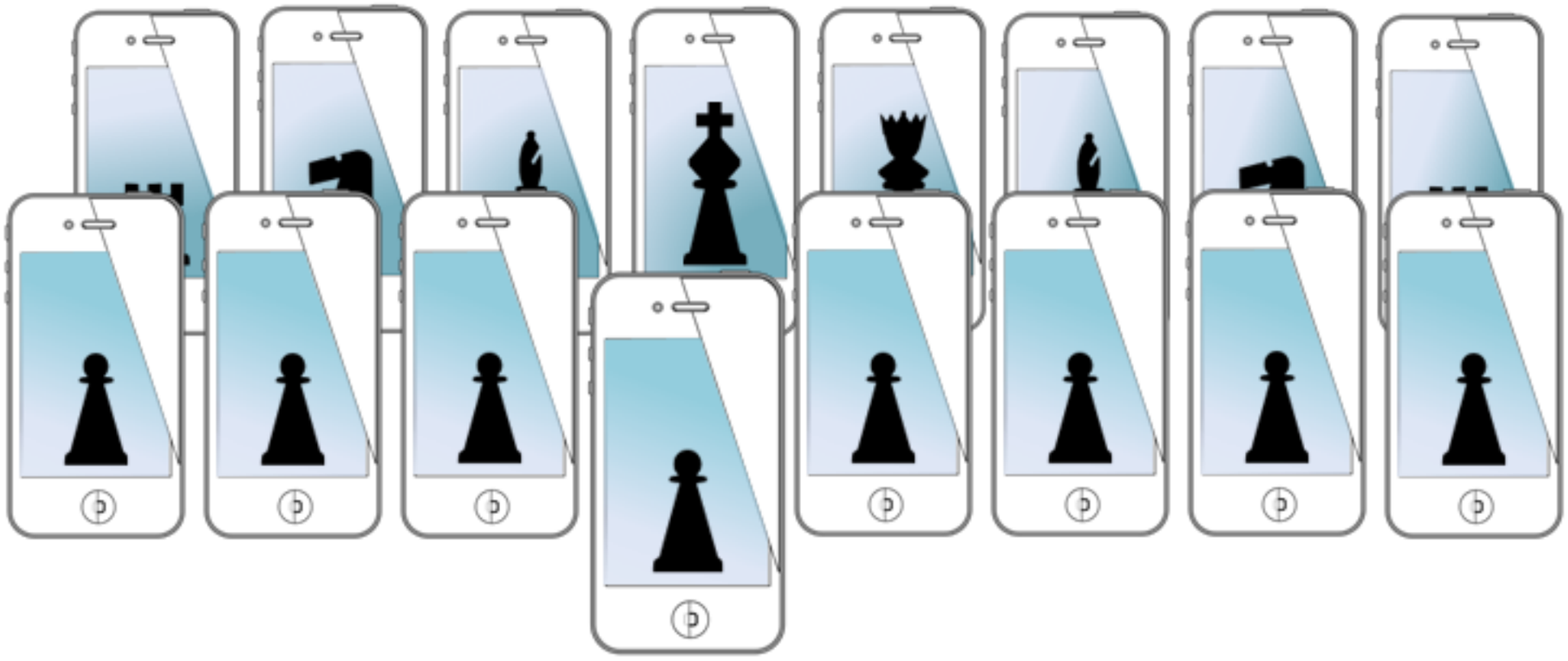
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

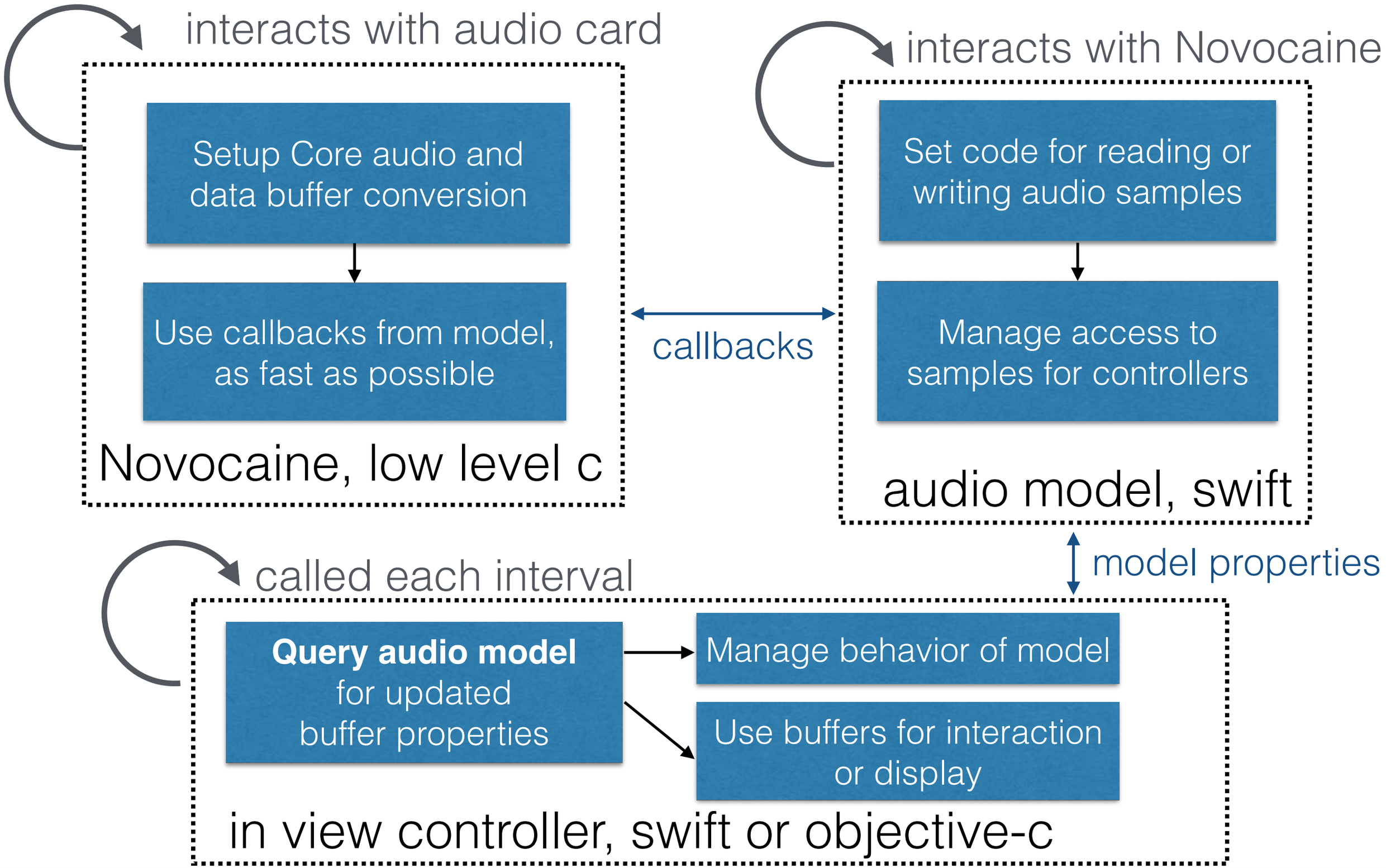
audio graphing, sampled data, & accelerate

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

agenda and logistics

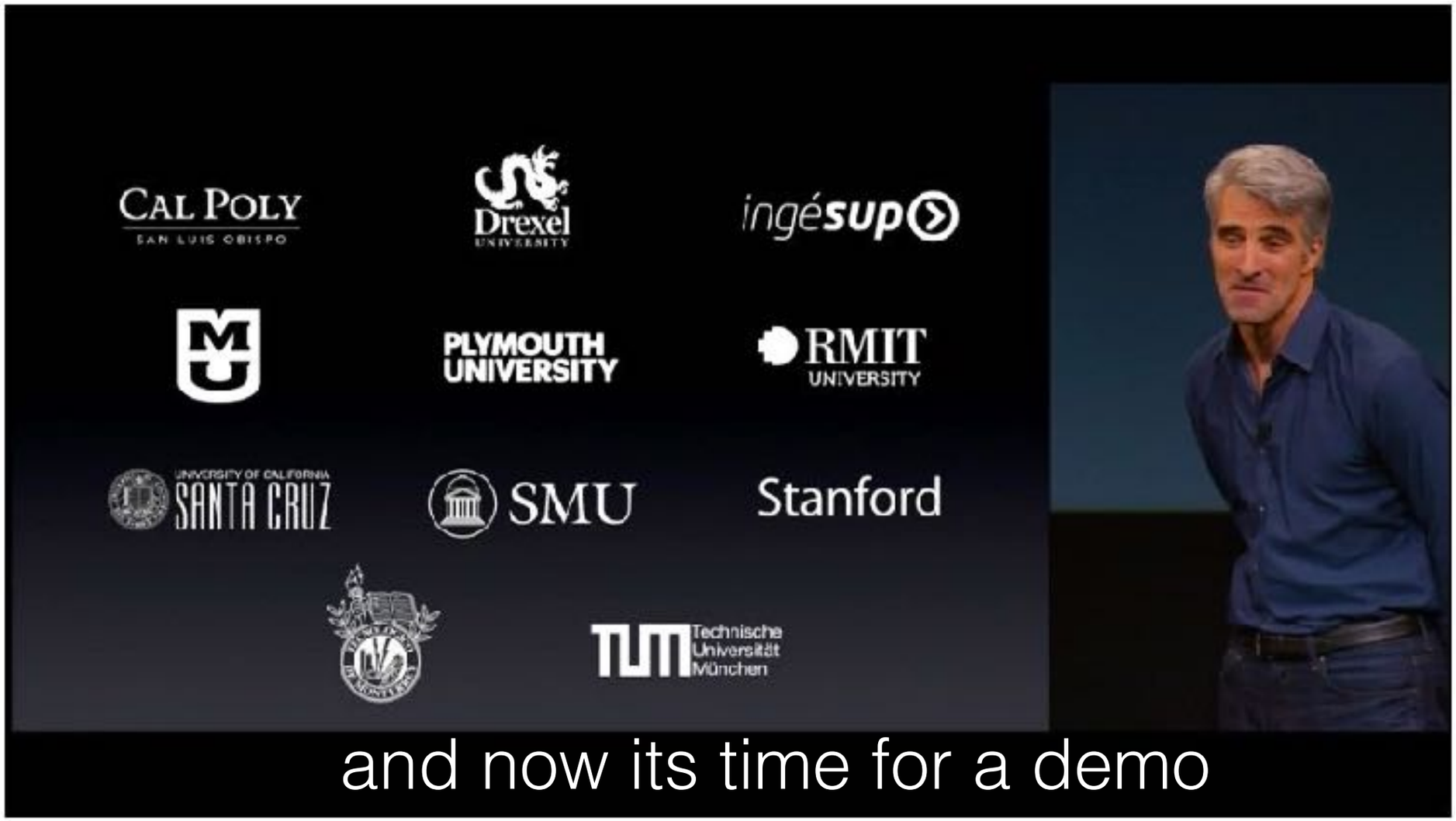
- logistics
  - flipped module on audio next time!
- agenda
  - dealing with sampled data
  - the accelerate framework
    - massive digital signal processing library
  - graphing audio fast (well, graphing anything)
    - must use lowest level graphing, Metal

review: MVC with audio



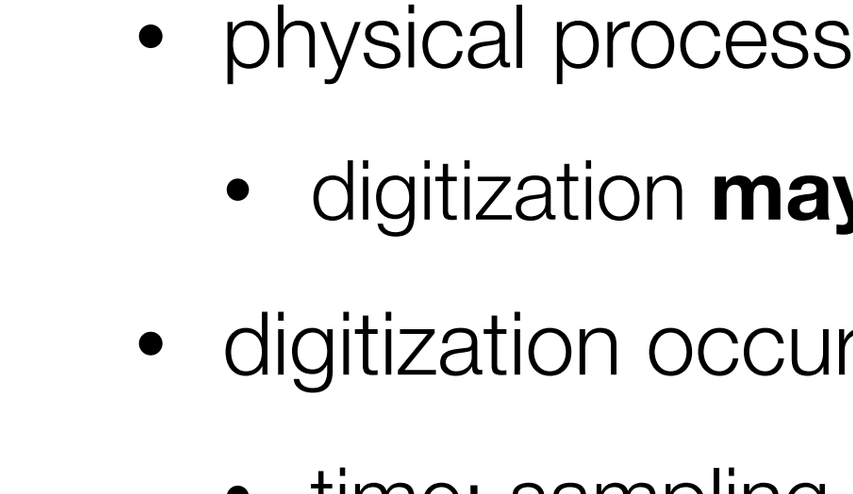
sample from the mic

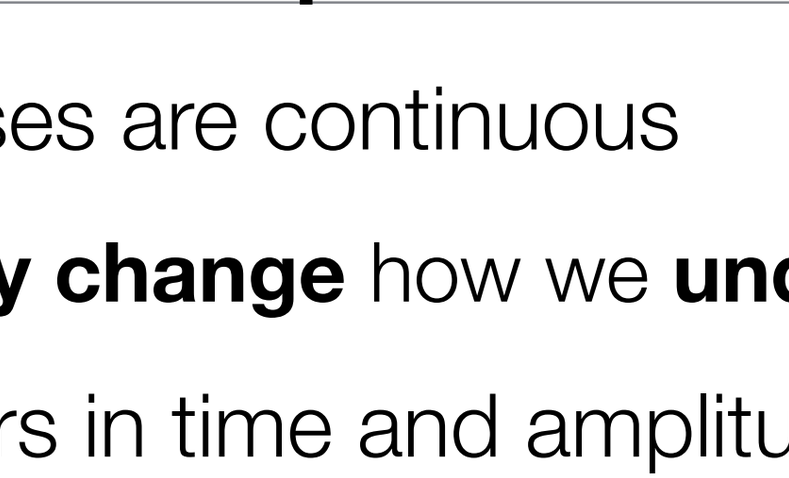
- recall: data from the microphone on novocaine



intro to sampled data

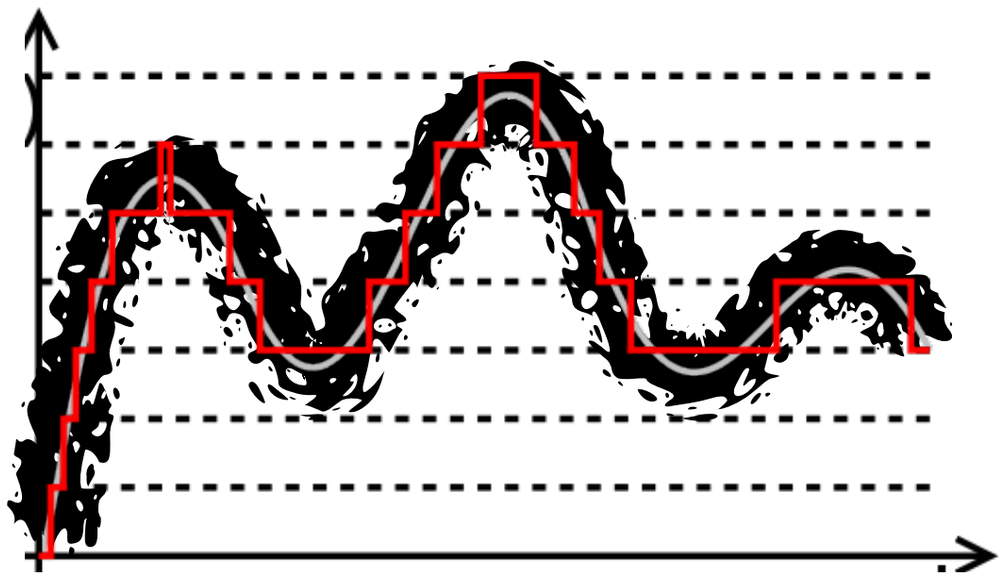
- physical processes are continuous
  - digitization **may change** how we **understand** the signal
- digitization occurs in time and amplitude
  - time: sampling
  - amplitude: quantization





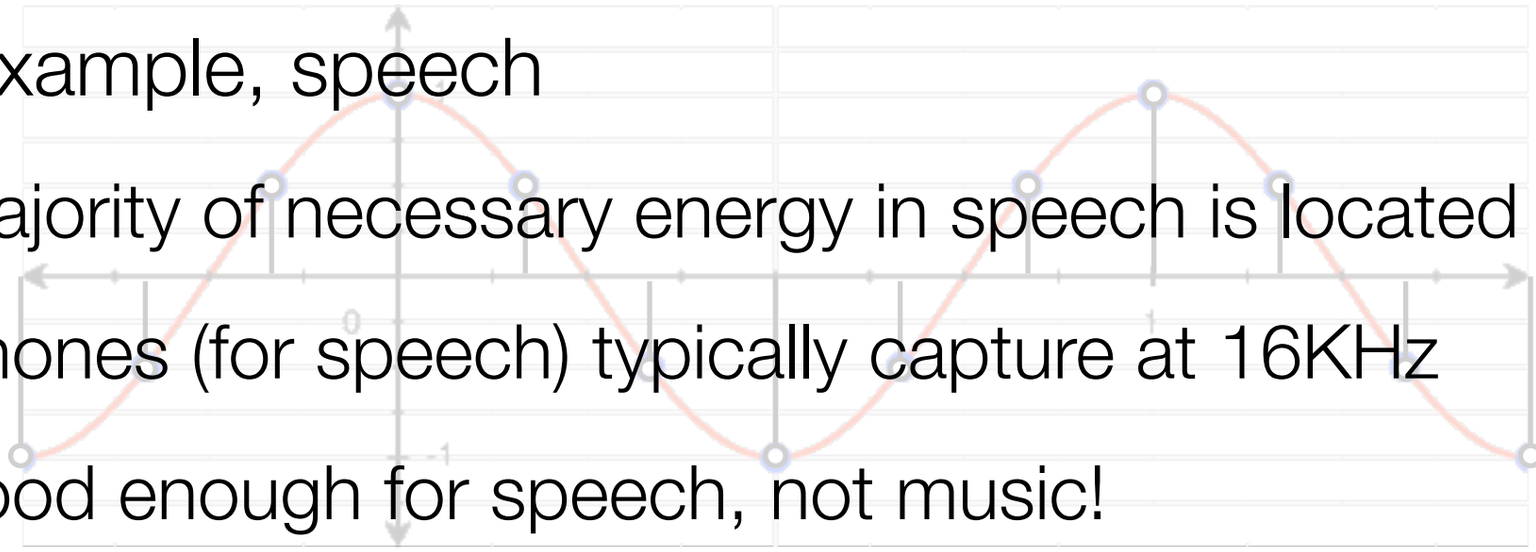
sampled data

- 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

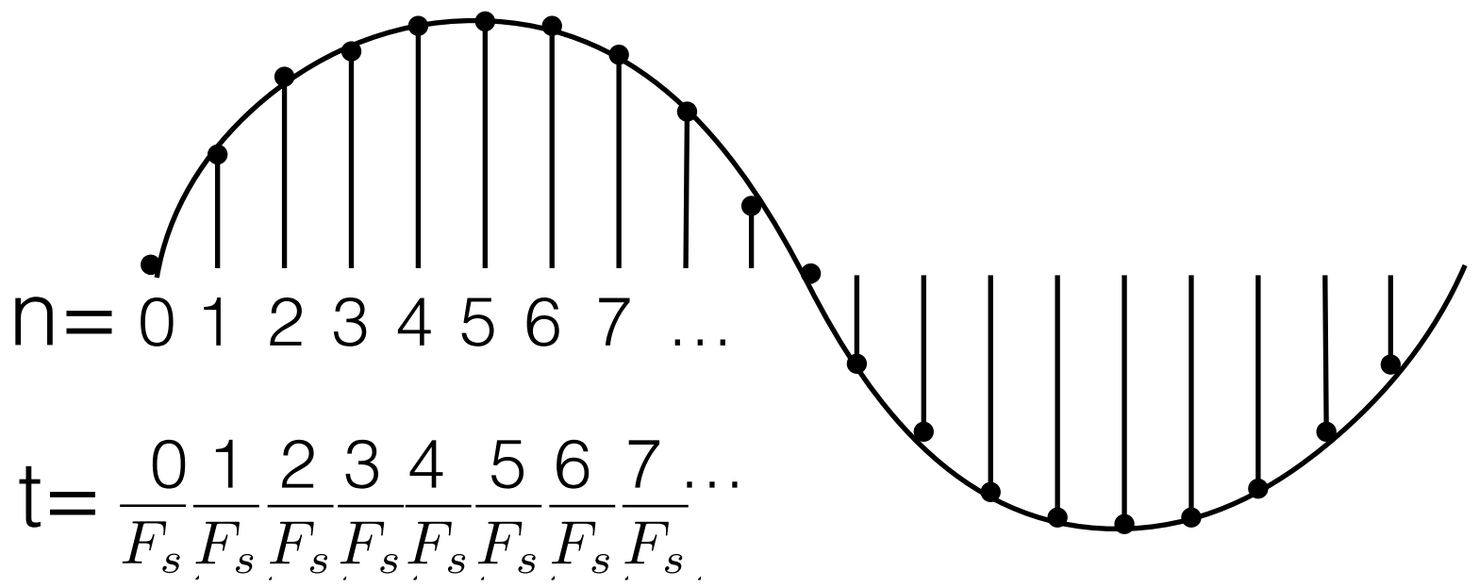
- we want to create a sine wave and play it to the speakers

$$g(t) = \sin(2\pi f t)$$ 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
);
```

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

examples

what do each of these implement?

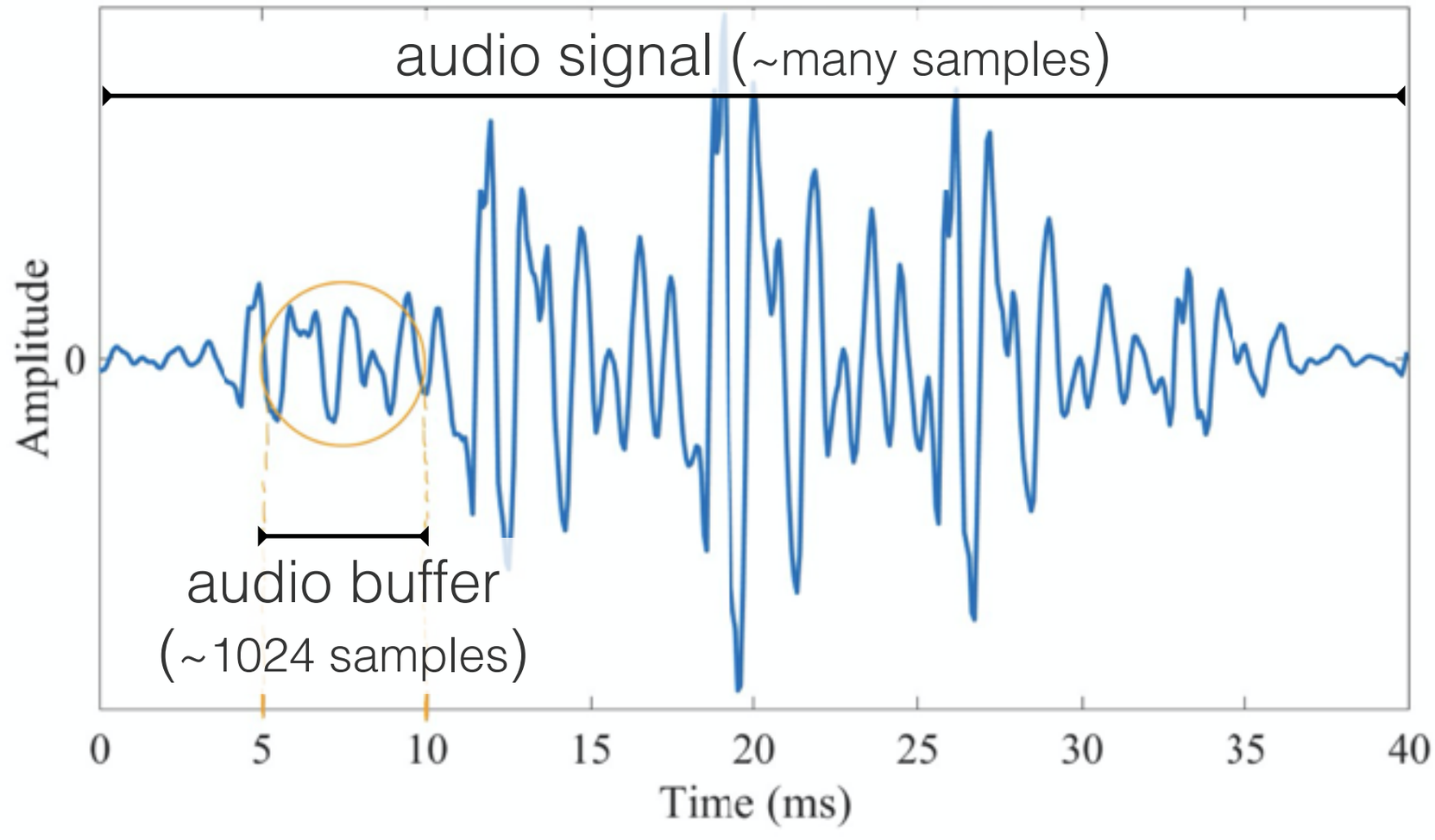
```
outputBlockFunction(data:(…),numFrames:(UInt32),numChannels:(UInt32)) {
      ringBuffer.fetchFreshData(data, withNumFrames:numFrames)
      var volume = userSetMultiplyFromSlider;
      vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels)
  }
```

```
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);
  }
```

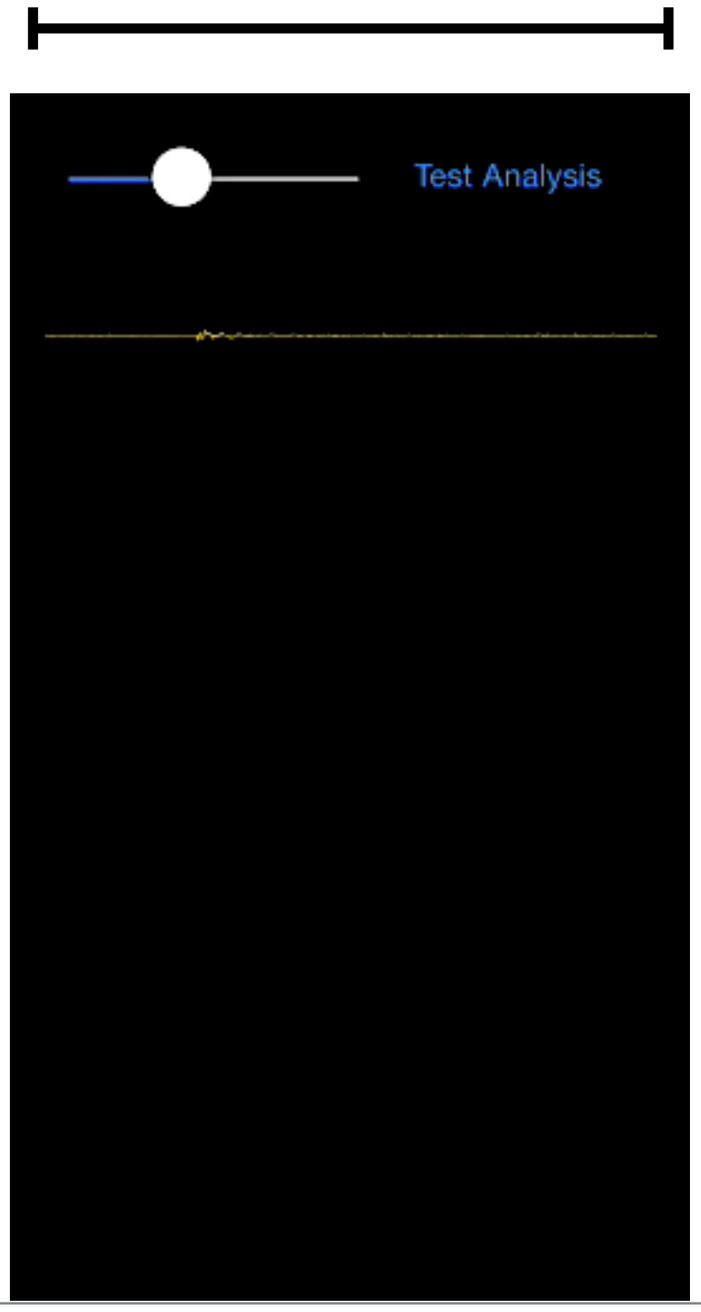
audio graphing

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



how much data to show?

- sampling at 44.1kHz == 44100 samples per second



graph 0.5 second window is:
22050 samples

display is >640 pixels wide

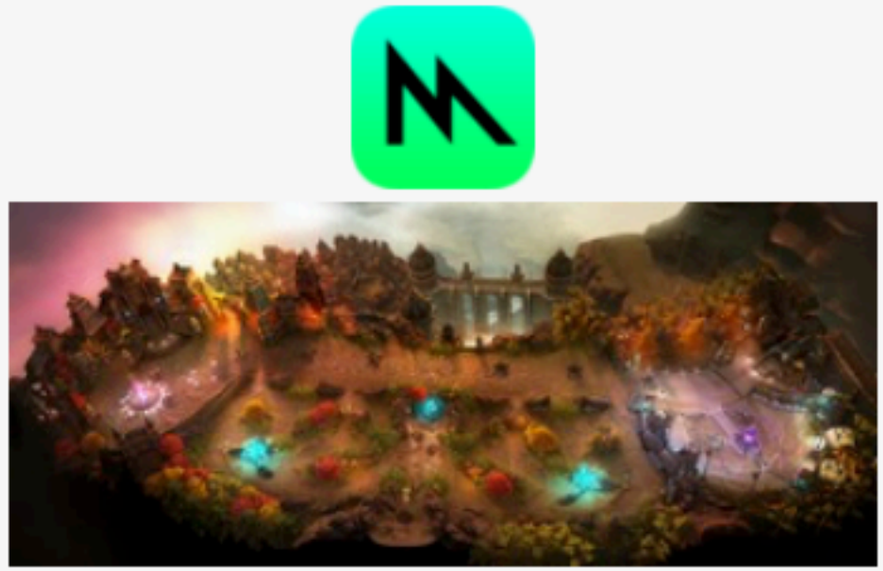
what if we want lots of graphs?

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

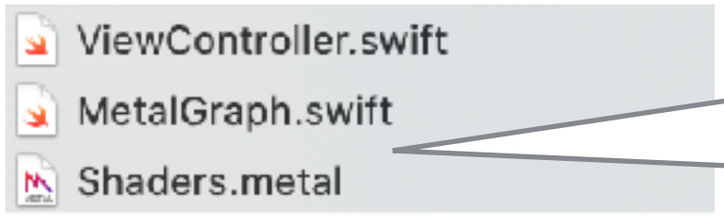
Metal



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

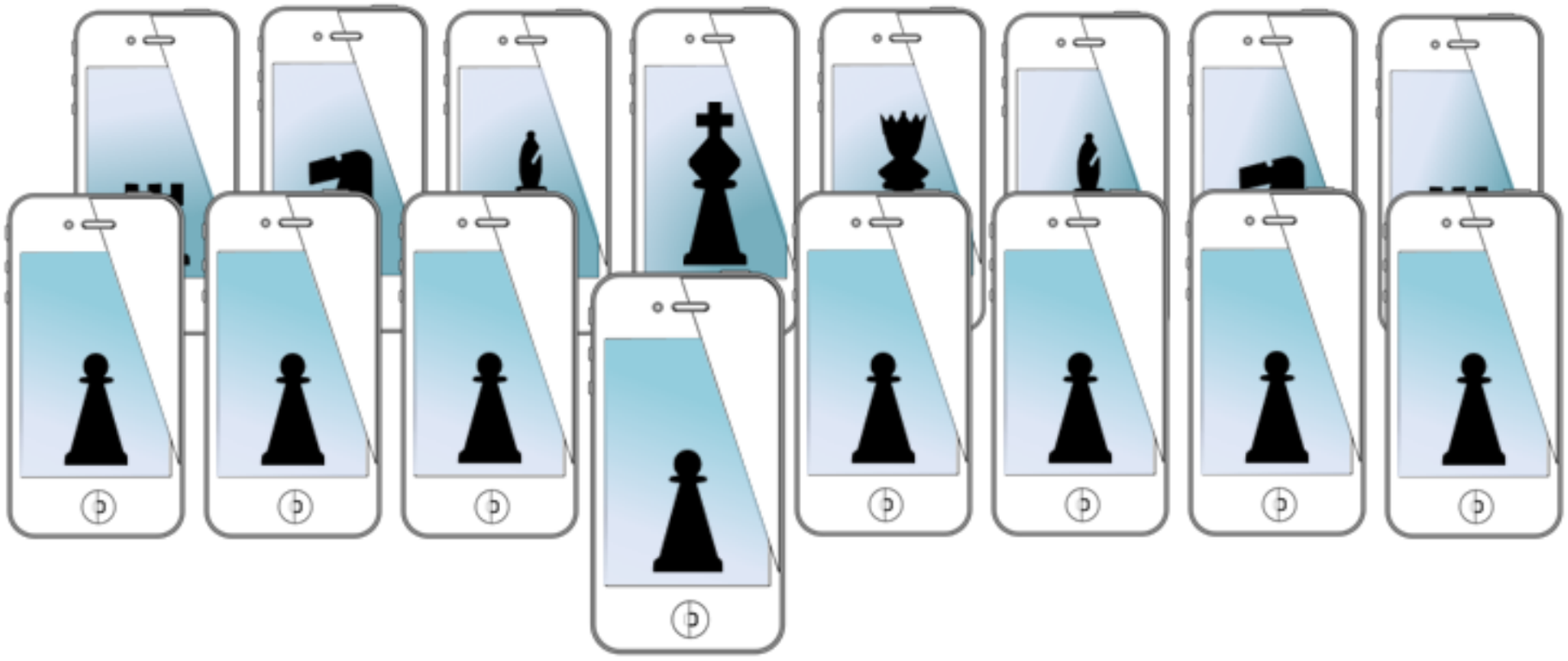
```
// periodically, display the audio data
graph?.updateGraph(
          data:  timeData,
          forKey: "time"
)
```

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