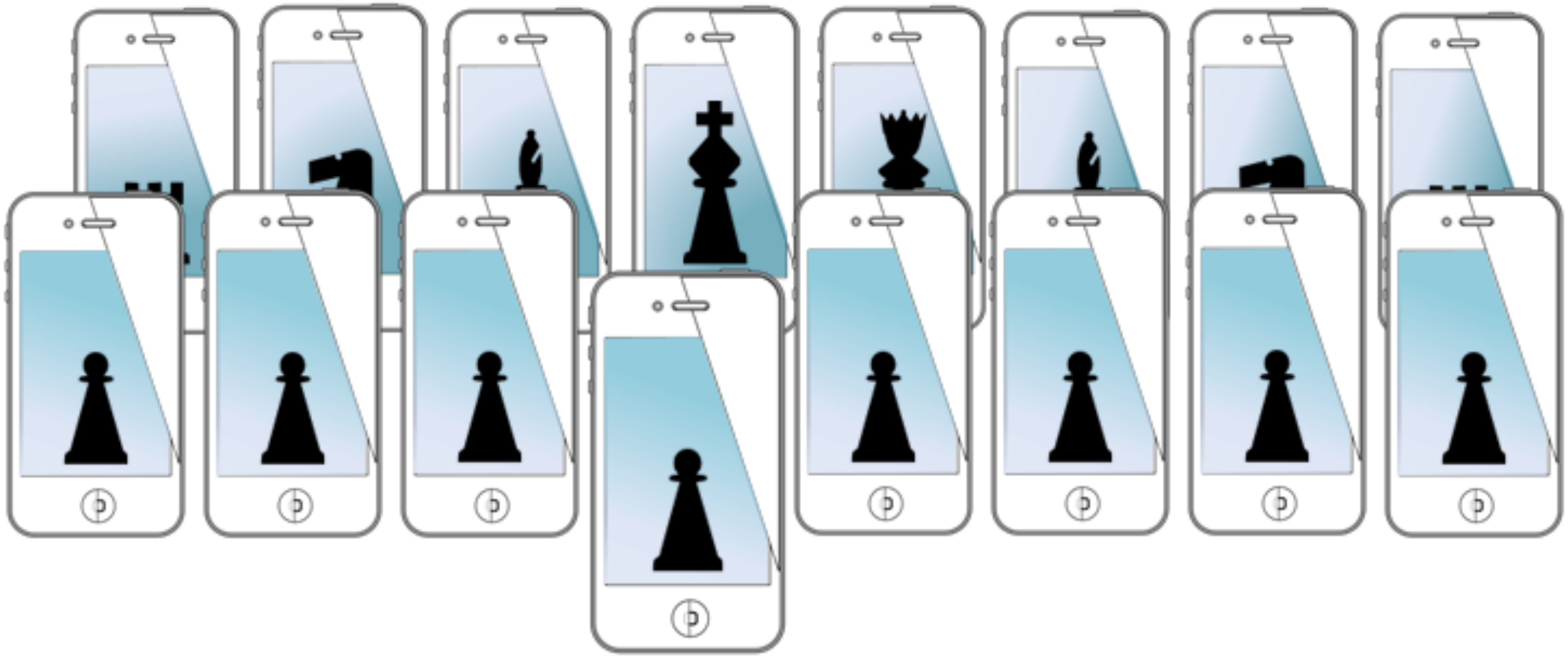
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture eight: audio, profiling, and core motion

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

course logistics

- A2 is due Friday
  - constraints are on the website!
  - feeling lost?

agenda

- FFT review
    - more examples
- profiling and debugging
- core motion
    - M7 co-processor
- accelerometers, gyros, and magnetometers

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

sample from the mic

- demo, switching around PlayRollingStones

making a sine wave

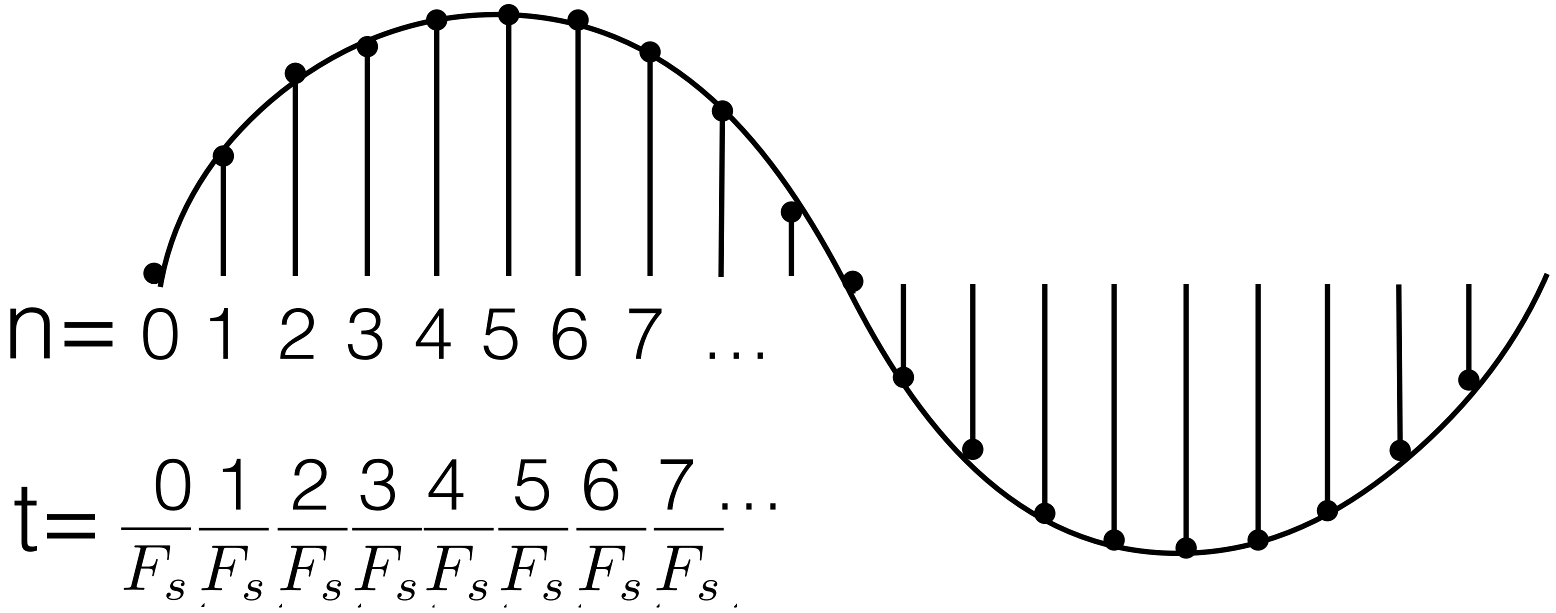
- we want to create a sine wave and play it to the speakers

$g(t) = \sin(2\pi f t)$ 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!


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

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

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

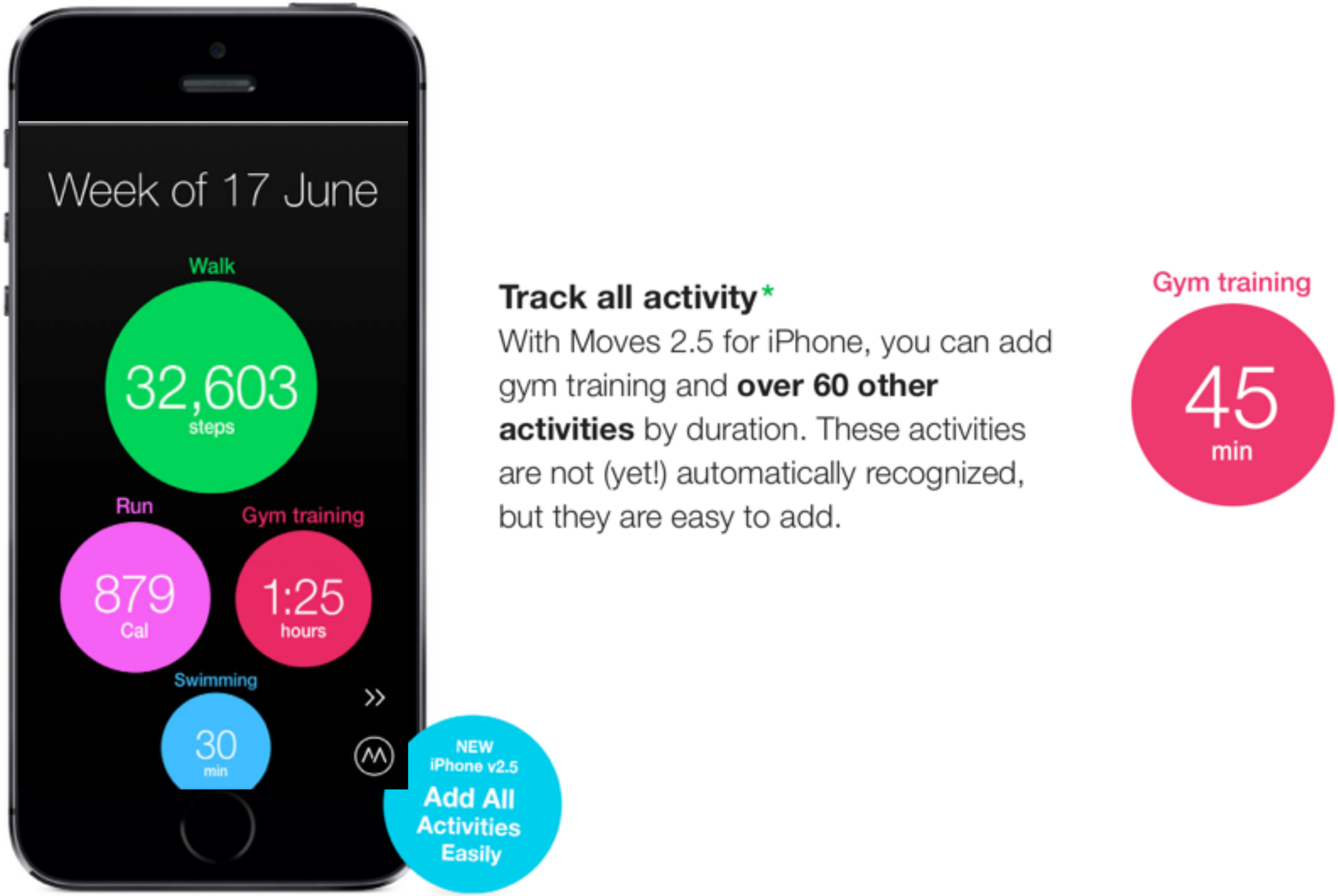
$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$

```
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;
     }

 }];
```

profiling demo

- using the instruments panel in Xcode
  - memory leaks
  - general efficiency
  - excellent integration with iOS

a nice example of core motion



Track all activity*
With Moves 2.5 for iPhone, you can add gym training and **over 60 other activities** by duration. These activities are not (yet!) automatically recognized, but they are easy to add.

the M7 coprocessor

- 150MHz processor that reads all motion data from all “motion” sensors on the phone
  - accelerometer
  - magnetometer (compass)
  - gyroscope
- mediates all access to data
  - battery life++
  - parallel processing++
  - overhead += 0, seriously
- sensor fusion for more accurate analysis, very cool

high level streams

- not just raw data!
  - the M7 does sophisticated analysis of sensor data for you
  - enables naive access to “high level” information
- can register your app to receive “updates” from the M7 unit
  - steps taken (and saved state of steps)
  - some common activity
    - running, walking, still, in car, unknown

activity from M7

- uses the “core motion” framework (CM)
- mediated through the “CMActivityManager”
  - is device capable of activity?
  - query past activities (up to 7 days)
  - subscribe to changes
- interaction completely based on blocks and handlers

subscribing to activity

- updates are notifications

```
#import <CoreMotion/CoreMotion.h>

// from M7 co-processor
@property (nonatomic, strong) CMMotionActivityManager *motionActivityManager;

   // initialize the activity manager (check if available)
    if ([CMMotionActivityManager isActivityAvailable] == YES) {
        self.motionActivityManager = [[CMMotionActivityManager alloc] init];
    }


if ([CMMotionActivityManager isActivityAvailable] == YES) {
        [self.motionActivityManager startActivityUpdatesToQueue:[NSOperationQueue mainQueue]
                                    withHandler:^(CMMotionActivity *activity) {
                                          // do something with the activity info!
                                    }];
        NSLog(@"Activity Manager Running");

    }
    else
        NSLog(@"Cannot start activity manager");


if([CMMotionActivityManager isActivityAvailable] == YES )
        [self.motionActivityManager stopActivityUpdates];
```

import framework

declare activity manager

device capable?

instantiate

subscribe

queue to run on

block to handle updates

end subscription

what’s in an update?

- updated when any part of activity estimate changes
- each update is a CMMotionActivity class instance
  - startDate (down to seconds)
  - walking {0,1}
  - stationary {0,1}
  - running {0,1}
  - automotive {0,1}
  - unknown {0,1}
  - confidence {Low, Medium, High}

```
startActivityUpdatesToQueue:[NSOperationQueue mainQueue]
                withHandler:^(CMMotionActivity *activity) {
                                           // do something
with the activity info!
                                    }];
```

example update

inside handler

```
(CMMotionActivity*) activity
```

from notification

```
    // enum for confidence is 0=low,1=medium,2=high
     NSLog(@" confidence:%ld \n stationary: %d \n walking: %d \n running: %d \n in car: %d",
             activity.confidence,
             activity.stationary,
             activity.walking,
             activity.running,
             activity.automotive);
```

access fields easily

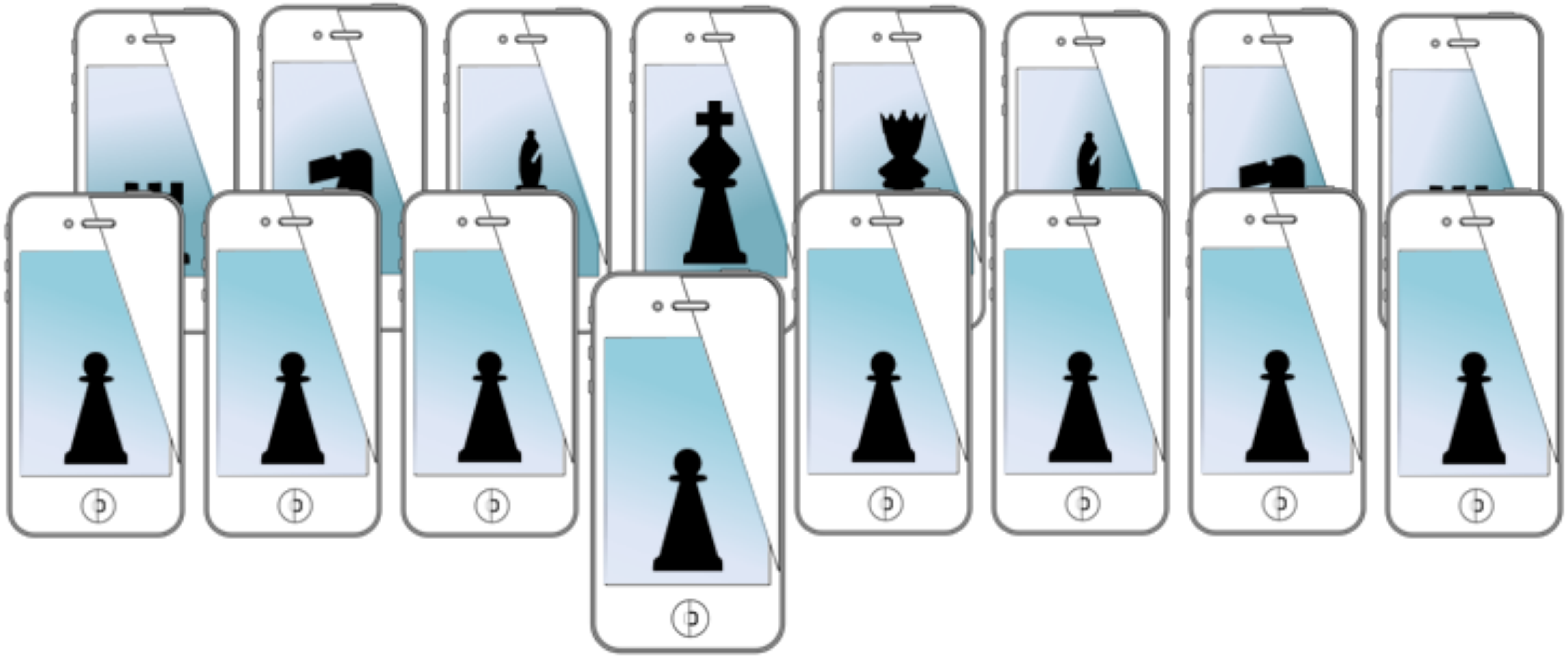
```
     switch (activity.confidence) {
         case CMMotionActivityConfidenceLow:
             self.confidenceLabel.text = @"low";
             break;
         case CMMotionActivityConfidenceMedium:
             self.confidenceLabel.text = @"med.";
             break;
         case CMMotionActivityConfidenceHigh:
             self.confidenceLabel.text = @"high";
             break;
         default:
             break;
     }
```

look at confidence

for next time…

- more on accelerometers, gyros, and magnetometers
- graphing with Apple API

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture eight: audio, profiling, and core motion

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