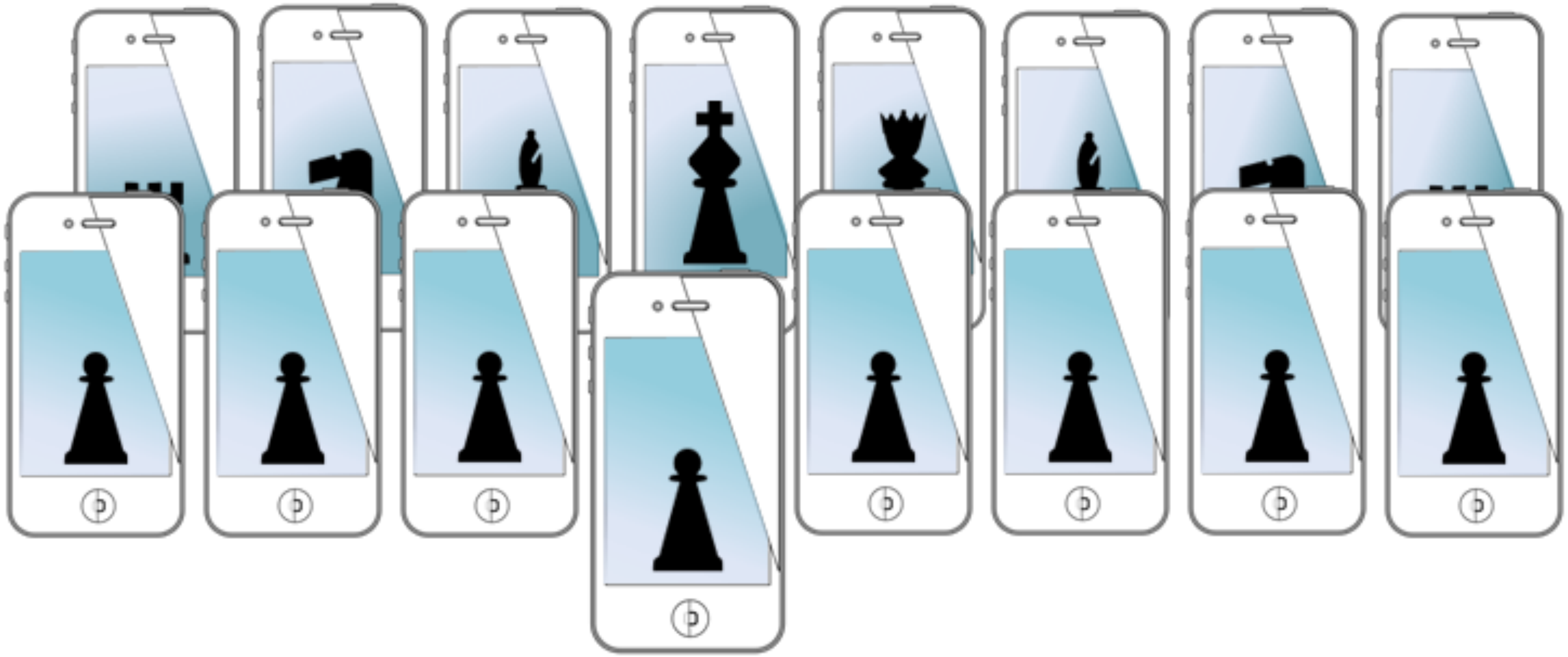
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture eight: audio, profiling, and M7

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

FFT review

- sampling rate
  - dictates the time between each sample, (1 / sampling rate)
  - max frequency we can measure is half of sampling rate

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?

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

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

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”

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!

making a sine wave

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

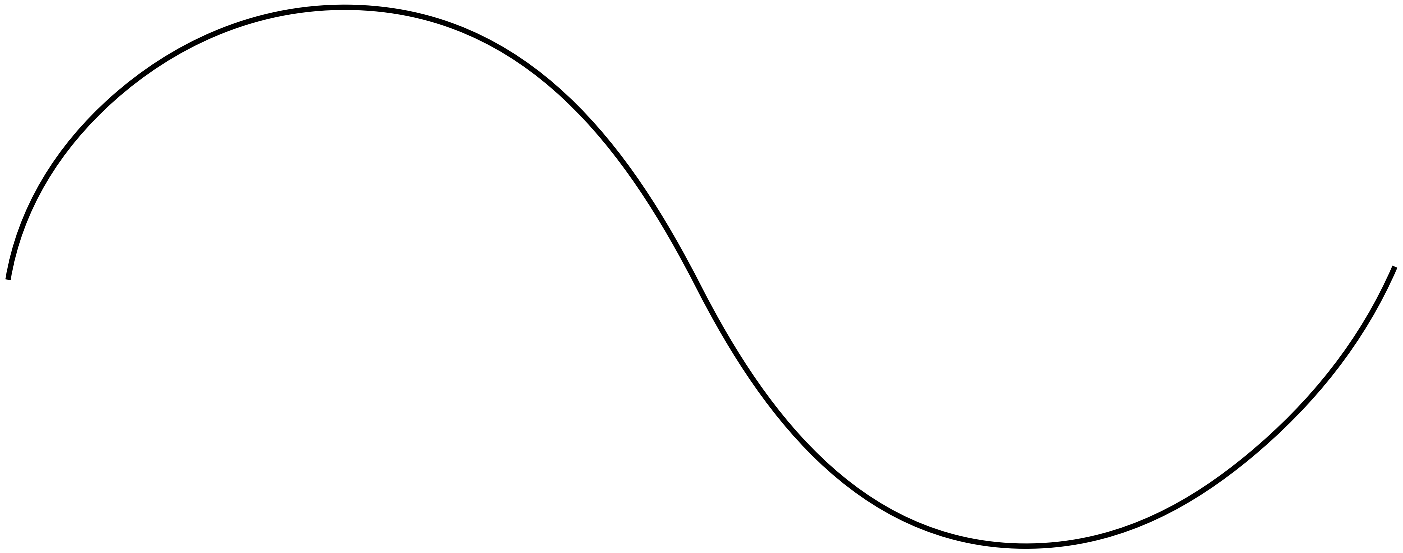
$$g(t) = \sin(2\pi f t)$$

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!



making a sine wave

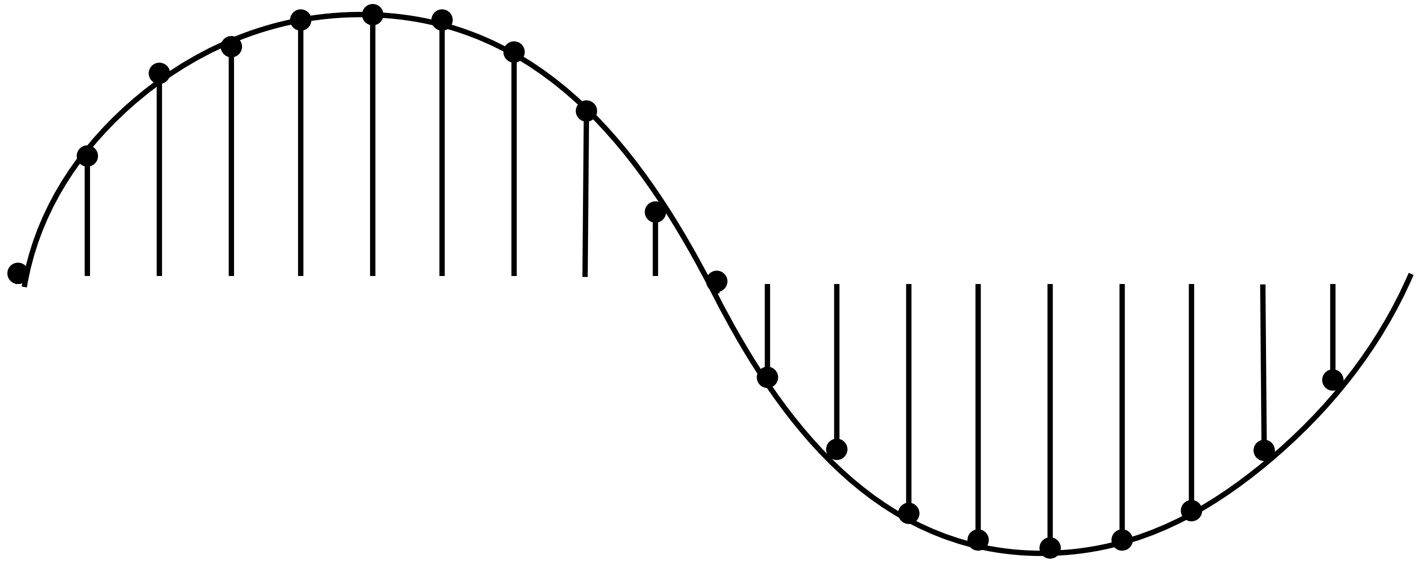
- we want to create a sine wave and play it to the speakers

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



making a sine wave

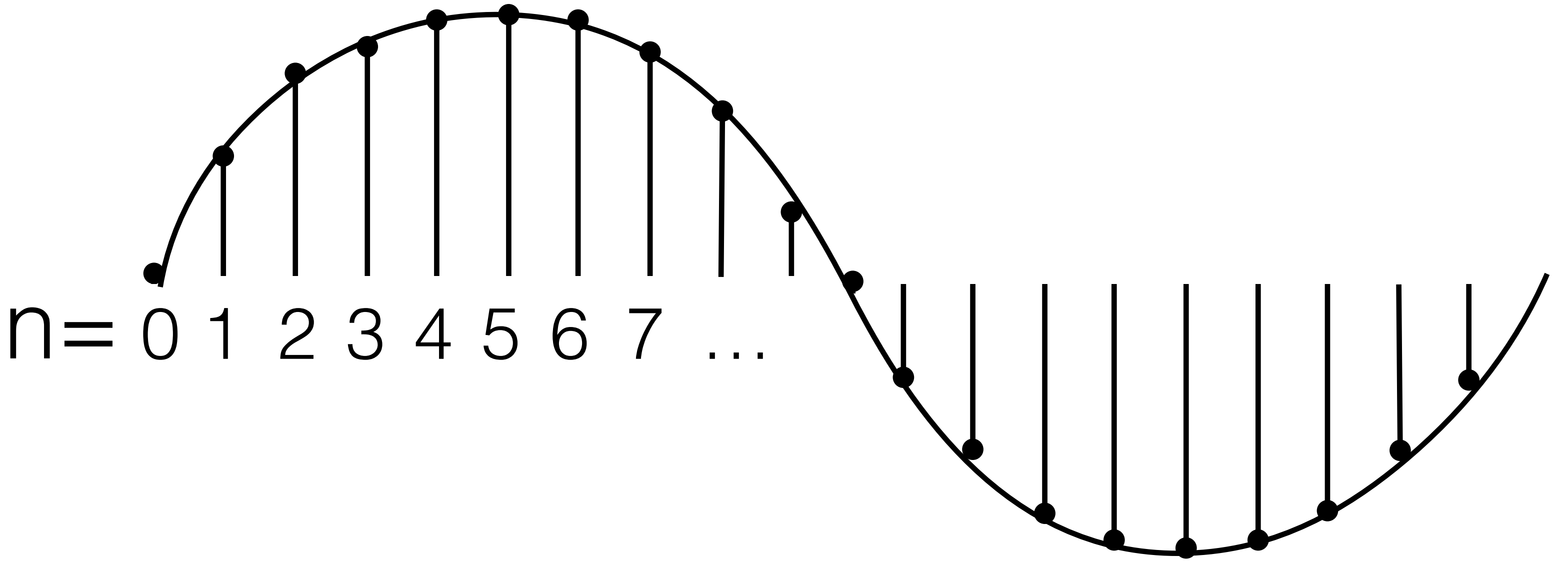
- we want to create a sine wave and play it to the speakers

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



making a sine wave

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

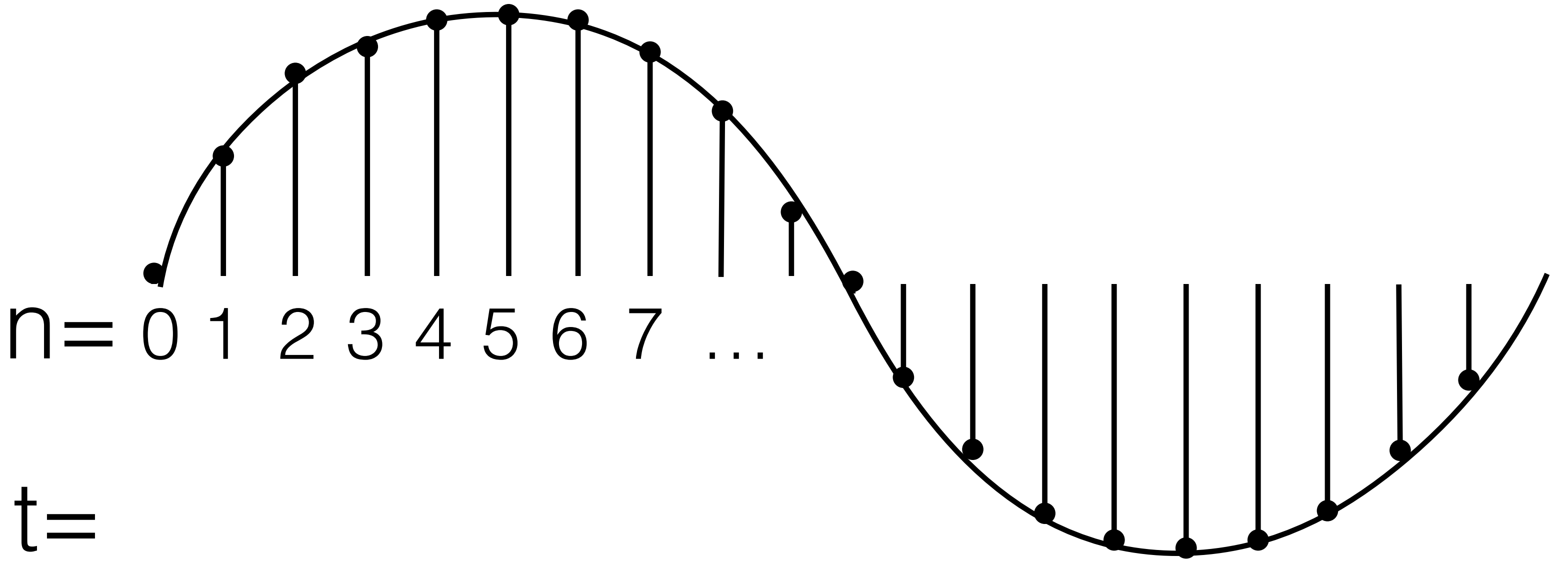
$$g(t) = \sin(2\pi f t)$$

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!



making a sine wave

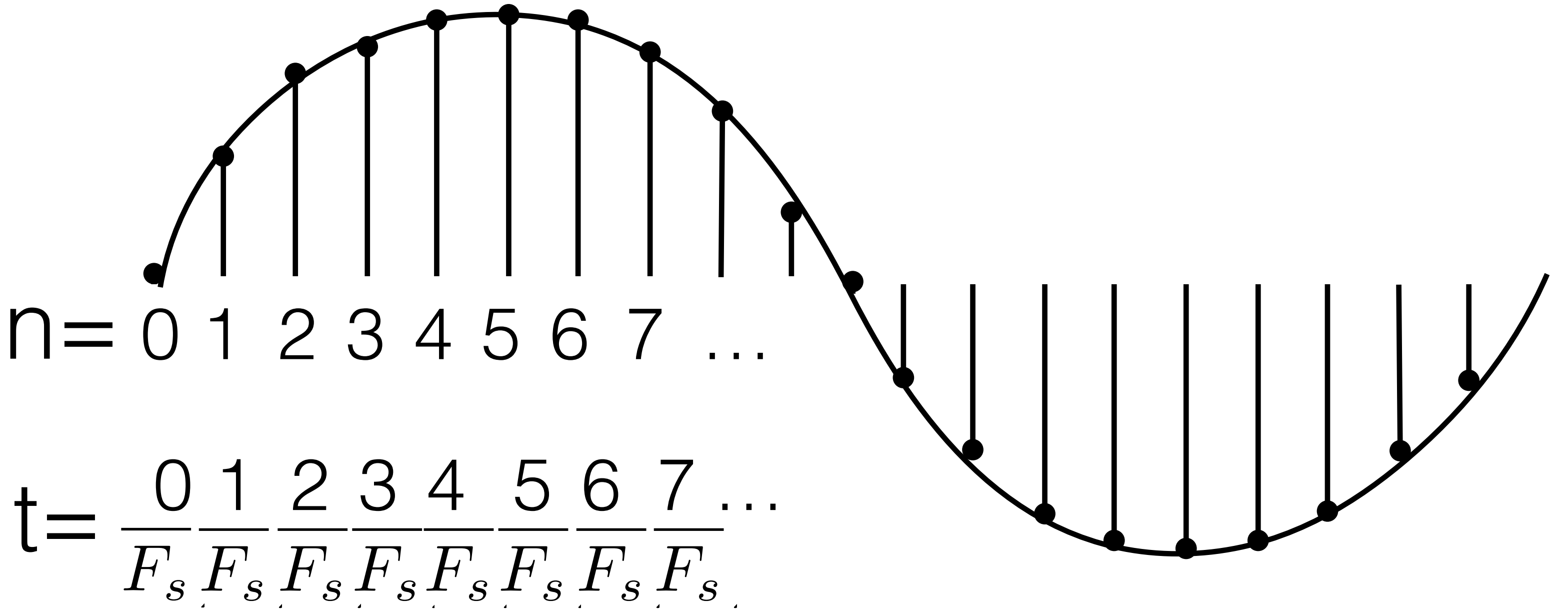
- we want to create a sine wave and play it to the speakers

$$g(t) = \sin(2\pi f t)$$ 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 …

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

making a sine wave

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

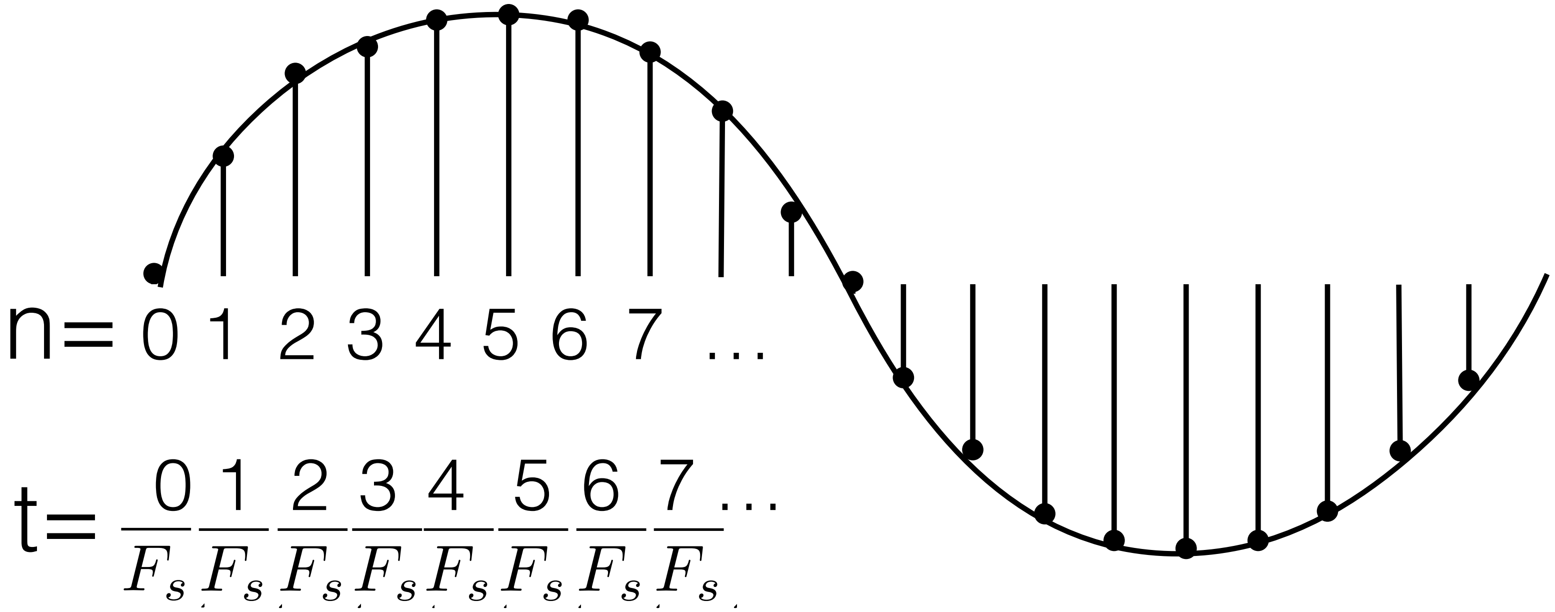
$$g(t) = \sin(2\pi f t)$$

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?

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);
 }
```

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?

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)
 {









 }];
```

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;

     for (int i=0; i < numFrames; ++i)
     {
         data[i] = sin(phase);

         phase += phaseIncrement;


     }

 }];
```

making a sine wave

- bringing it all together

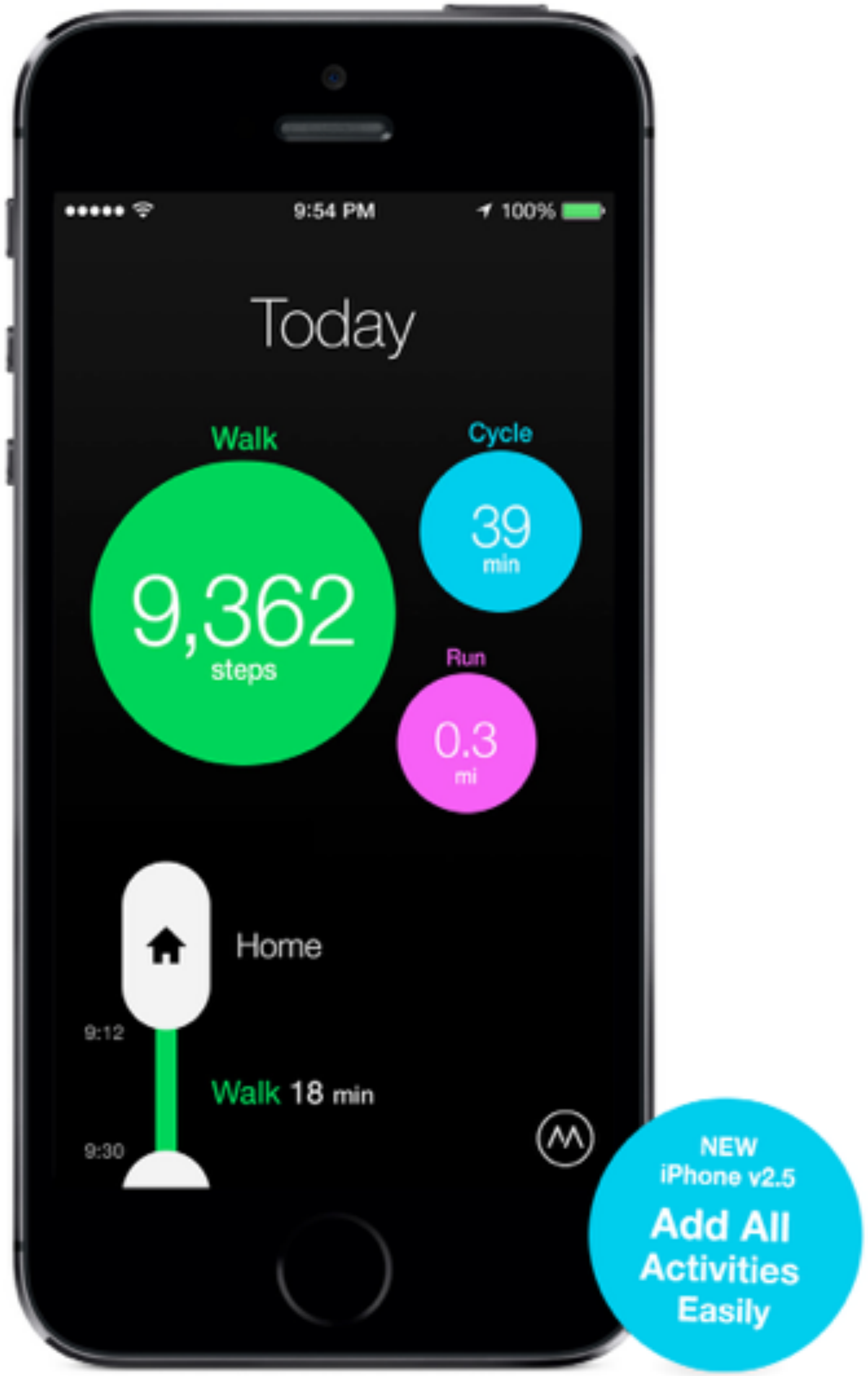
$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$

```
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;
     }

 }];
```

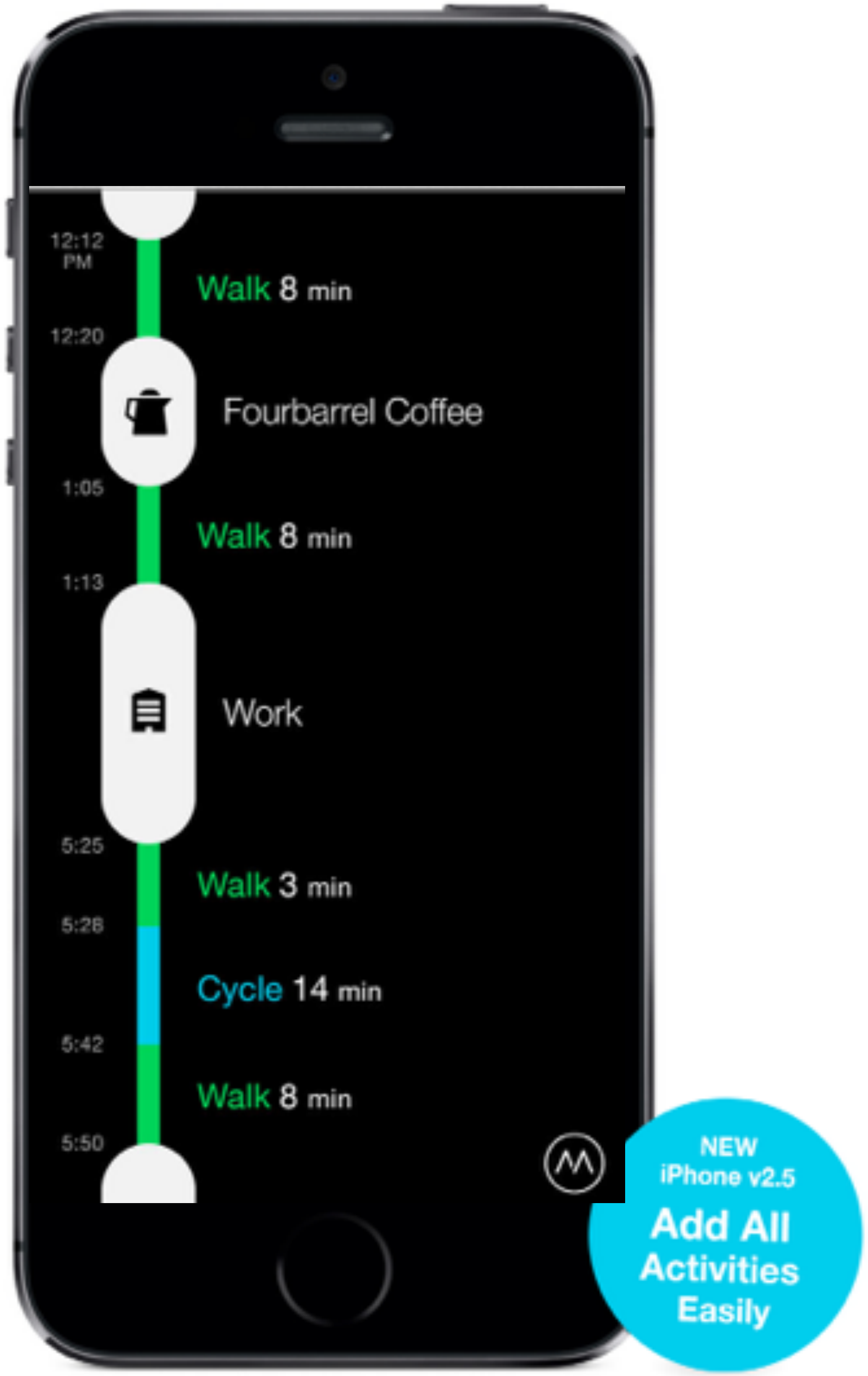
profiling demo

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

a nice example of core motion



a nice example of core motion



12:12 PM

Walk 8 min

12:20

Fourbarrel Coffee

1:05

Walk 8 min

1:13

Work

5:25

Walk 3 min

5:28

Cycle 14 min

5:42

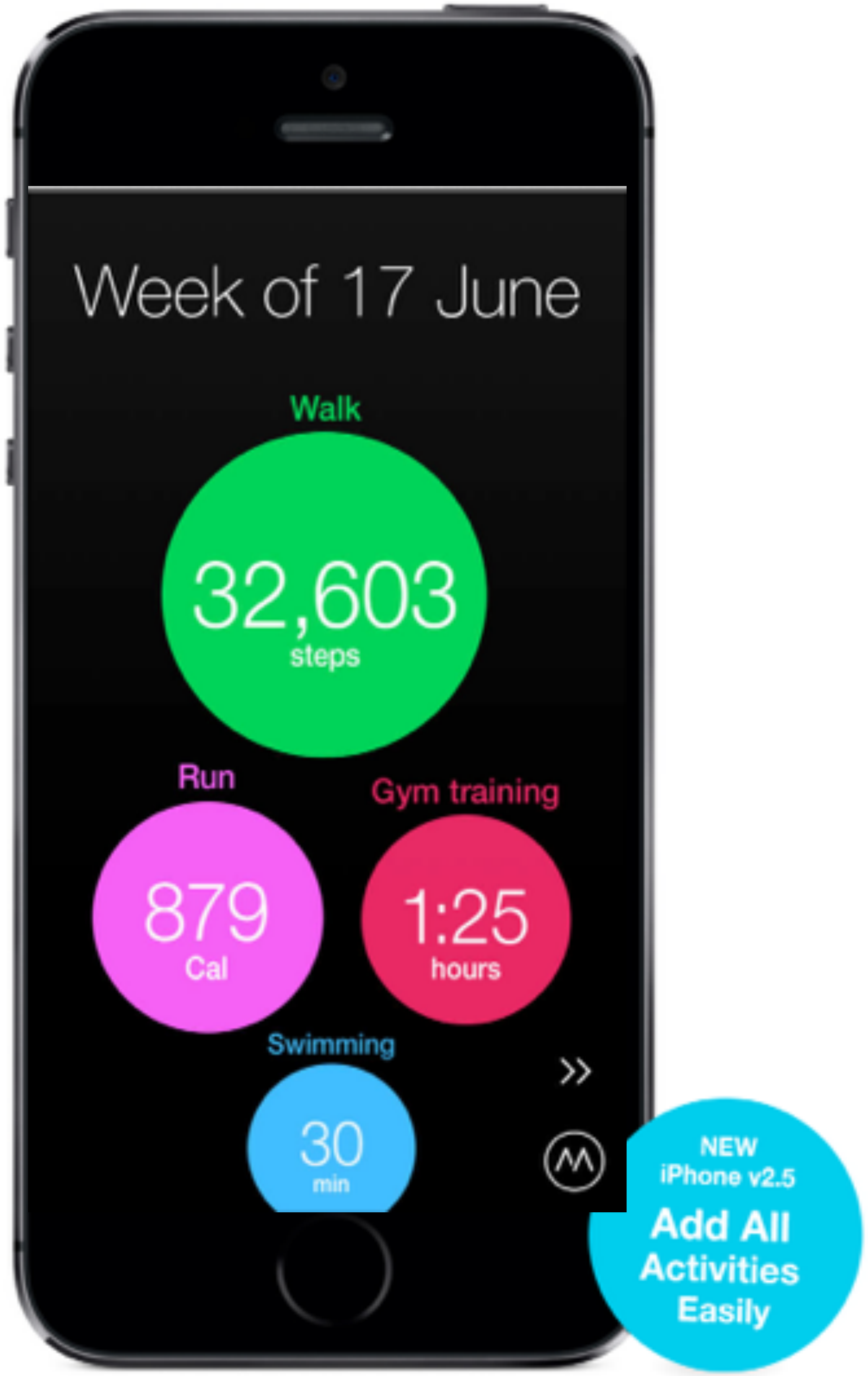
Walk 8 min

5:50

NEW iPhone v2.5

Add All Activities Easily

a nice example of core motion



Week of 17 June

Walk

32,603 steps

Run

879 Cal

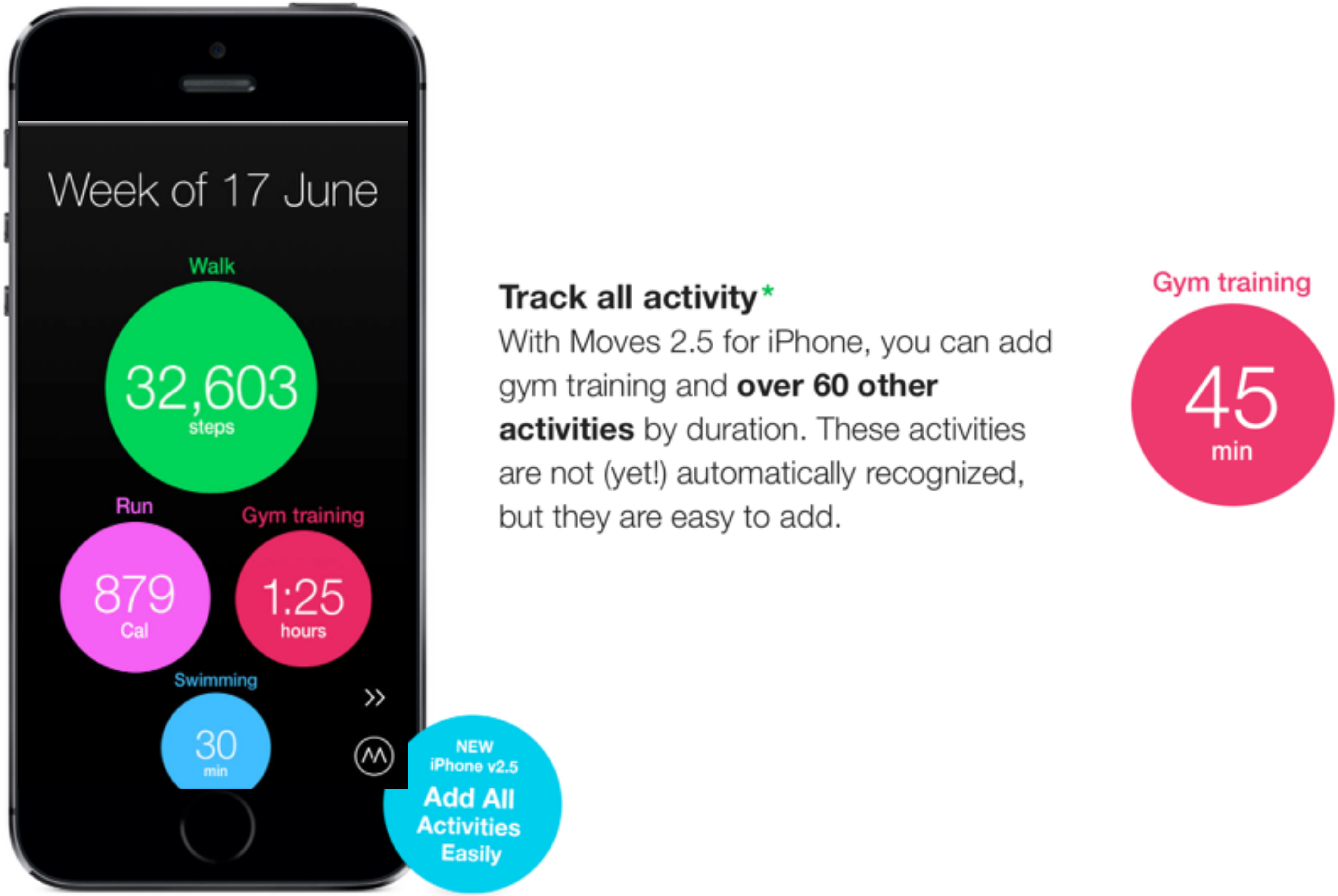
Gym training

1:25 hours

Swimming

30 min

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

the M7 coprocessor

- 150MHz processor that reads all motion data from all "motion" sensors on the phone
  - accelerometer
  - magnetometer (compass)
  - gyroscope

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

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

high level streams

- not just raw data!
  - the M7 does sophisticated analysis of sensor data for you
  - enables naive access to “high level” information

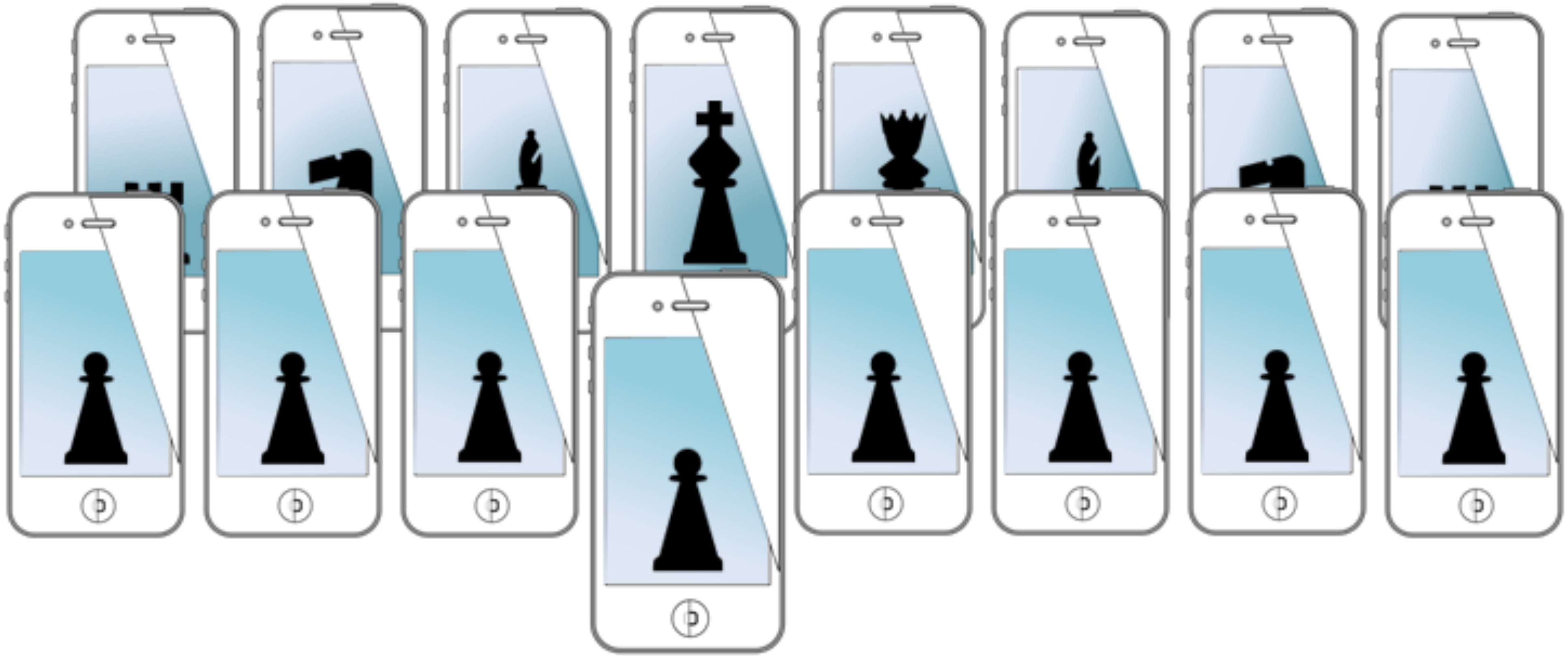
high level streams

- 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

for next time…

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

MOBILE SENSING LEARNING & CONTROL



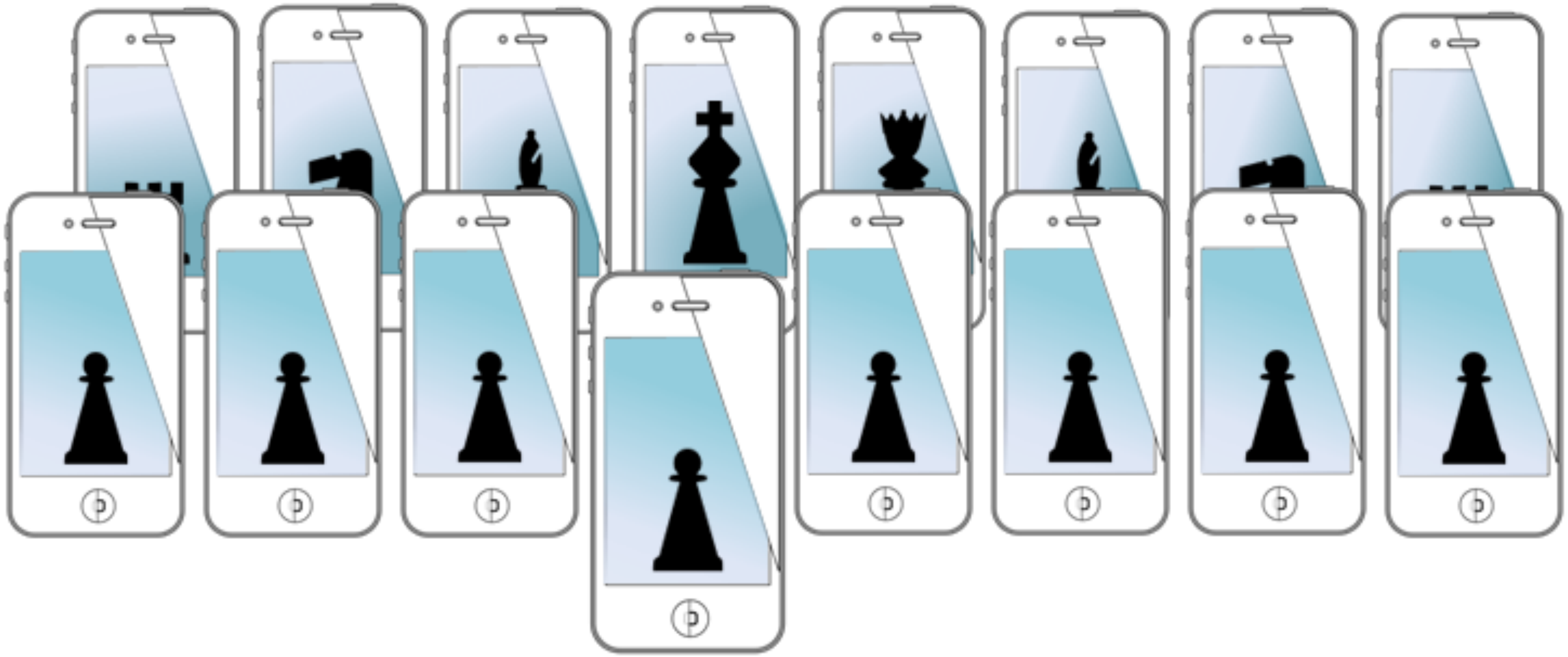
CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture eight: audio, profiling, and M7

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

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture nine: core motion: activity, step counting, and sensor fusion

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

course logistics

course logistics

- A2 is due Friday

course logistics

- A2 is due Friday
  - updates for module A

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart
    - 12Hz accuracy (+-6Hz)

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart
    - 12Hz accuracy (+-6Hz)
- A3 is due the Following Friday!

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart
    - 12Hz accuracy (+-6Hz)
- A3 is due the Following Friday!
  - its a one week assignment

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart
    - 12Hz accuracy (+-6Hz)
- A3 is due the Following Friday!
  - its a one week assignment
  - or is it?

course logistics

- A2 is due Friday
  - updates for module A
    - needs only 100Hz apart
    - 12Hz accuracy (+-6Hz)
- A3 is due the Following Friday!
  - its a one week assignment
  - or is it?
    - better to make due Monday, March 3rd at 6PM?

agenda

- activity
- step counting
- persistence with small data
- more on accelerometers, gyros, and magnetometers
- graphing feedback to users

activity from M7

activity from M7

- uses the “core motion” framework (CM)

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

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];
```

subscribing to activity

- updates are notifications

import framework

```
#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];
```

subscribing to activity

- updates are notifications

import framework

declare activity manager

```
#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];
```

subscribing to activity

- updates are notifications

import framework

declare activity manager

device capable?

```
#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];
```

subscribing to activity

- updates are notifications

import framework

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

declare activity manager

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

device capable?

instantiate

```
   // 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];
```

subscribing to activity

- updates are notifications

import framework

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

declare activity manager

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

device capable?

instantiate

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

subscribe

```
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];
```

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

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

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?

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

what’s in an update?

- updated when any part of activity estimate changes

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

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


   // 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);



    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;
    }
```

example update

inside handler

from notification

```
(CMMotionActivity*) activity
```

```
    // 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);



     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;
     }
```

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;
     }
```

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

past activity

- query for an array of CMMotionActivity activities

```
 // example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];



 [self.motionActivityManager queryActivityStartingFromDate:from
             toDate:now
             toQueue:[NSOperationQueue mainQueue]
   withHandler:^(NSArray *activities, NSError *error) {

     for(CMMotionActivity *cmAct in activities)
     {
         NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
     }

 }];
```

past activity

- query for an array of CMMotionActivity activities

setup date range

```
// example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];



[self.motionActivityManager queryActivityStartingFromDate:from
            toDate:now
            toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSArray *activities, NSError *error) {

    for(CMMotionActivity *cmAct in activities)
    {
        NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
    }

}];
```

past activity

- query for an array of CMMotionActivity activities

setup date range

```
// example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];
```

set dates

```
[self.motionActivityManager queryActivityStartingFromDate:from
            toDate:now
            toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSArray *activities, NSError *error) {

    for(CMMotionActivity *cmAct in activities)
    {
        NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
    }

}];
```

past activity

- query for an array of CMMotionActivity activities

setup date range

```
 // example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];
```

set dates

```
[self.motionActivityManager queryActivityStartingFromDate:from
            toDate:now
            toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSArray *activities, NSError *error) {

    for(CMMotionActivity *cmAct in activities)
    {
        NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
    }

}];
```

set queue

past activity

- query for an array of CMMotionActivity activities

setup date range

```
 // example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];
```

set dates

```
[self.motionActivityManager queryActivityStartingFromDate:from
            toDate:now
            toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSArray *activities, NSError *error) {

    for(CMMotionActivity *cmAct in activities)
    {
        NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
    }

}];
```

set queue

handle output

past activity

- query for an array of CMMotionActivity activities

setup date range

```
 // example of querying from certain dates
NSDate *now  = [NSDate date];
NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];
```

set dates

set queue

```
[self.motionActivityManager queryActivityStartingFromDate:from
            toDate:now
            toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSArray *activities, NSError *error) {

    for(CMMotionActivity *cmAct in activities)
    {
        NSLog(@"At %@, user was walking %d",cmAct.startDate,cmAct.walking);
    }

}];
```

handle error!

handle output

more than activity

- M7 also tracks the number of steps during each activity
- you can get updated by the OS!

step counting

step counting

- special handling from the M7
    - CMStepCounter is the manager
    - updates highly similar to activity manager

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }


   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                              updateOn:1
                 withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
         {
             DO SOMETHING
         }];
```

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

declare and init

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }


   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                          updateOn:1
               withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
        {
            DO SOMETHING
        }];
```

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

declare and init

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }
```

queue to run on

```
   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                              updateOn:1
                 withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
         {
             DO SOMETHING
         }];
```

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

declare and init

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }
```

queue to run on

update interval (preferred)

```
   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                                updateOn:1
                 withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
         {
             DO SOMETHING
         }];
```

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }


   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                                updateOn:1
              withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
        {
            DO SOMETHING
        }];
```

declare and init

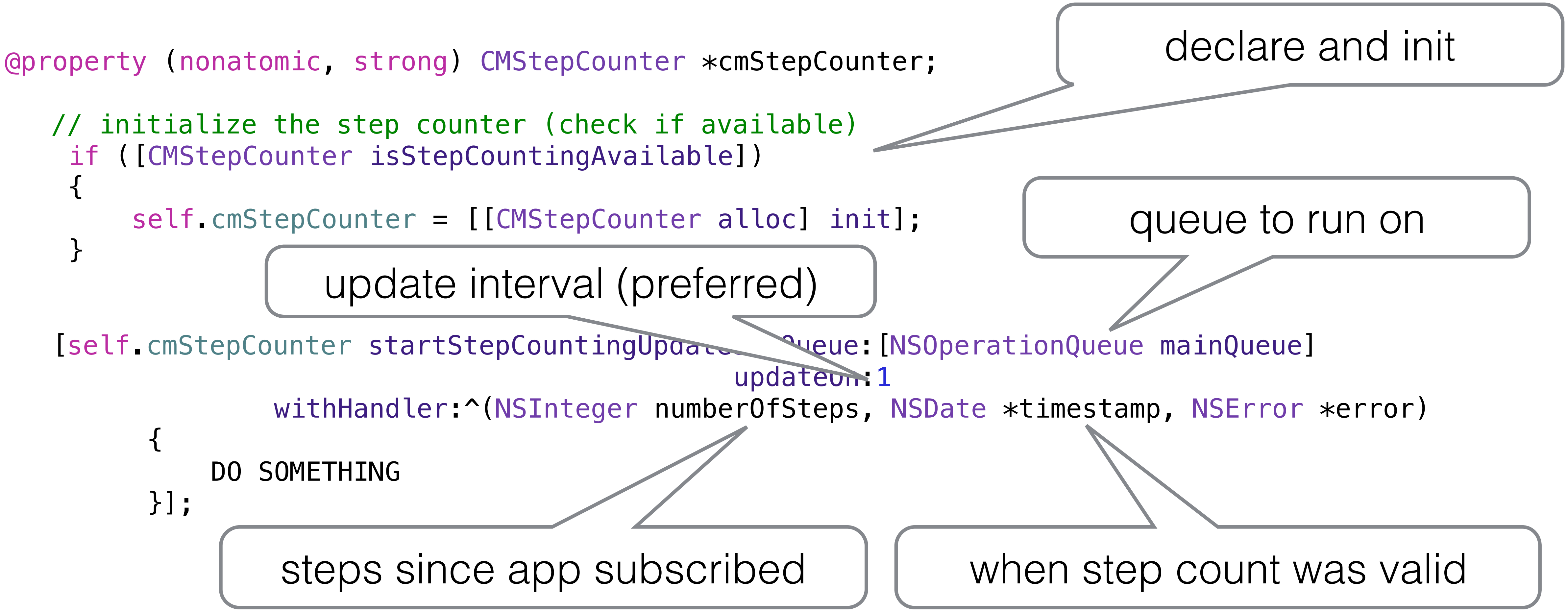
queue to run on

update interval (preferred)

steps since app subscribed

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager



```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

    // initialize the step counter (check if available)
     if ([CMStepCounter isStepCountingAvailable])
     {
         self.cmStepCounter = [[CMStepCounter alloc] init];
     }


    [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                              updateOn:1
                 withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
         {
             DO SOMETHING
         }];
```

declare and init
queue to run on
update interval (preferred)
steps since app subscribed
when step count was valid

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }


   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                                 updateOn:1
              withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
        {
            DO SOMETHING
        }];
```

declare and init

queue to run on

update interval (preferred)

steps since app subscribed

when step count was valid

```
    if ([CMStepCounter isStepCountingAvailable] == YES)
         [self.cmStepCounter stopStepCountingUpdates];
```

step counting

- special handling from the M7
  - CMStepCounter is the manager
  - updates highly similar to activity manager

```
@property (nonatomic, strong) CMStepCounter *cmStepCounter;

   // initialize the step counter (check if available)
    if ([CMStepCounter isStepCountingAvailable])
    {
        self.cmStepCounter = [[CMStepCounter alloc] init];
    }


   [self.cmStepCounter startStepCountingUpdatesToQueue:[NSOperationQueue mainQueue]
                                                updateOn:1
                 withHandler:^(NSInteger numberOfSteps, NSDate *timestamp, NSError *error)
          {
              DO SOMETHING
          }];
```

declare and init

queue to run on

update interval (preferred)

steps since app subscribed

when step count was valid

```
    if ([CMStepCounter isStepCountingAvailable] == YES)
         [self.cmStepCounter stopStepCountingUpdates];
```

unsubscribe

step counting

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120
- iOS: you get the update when we say you do!

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120
- iOS: you get the update when we say you do!
  - which optimizes battery life

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120
- iOS: you get the update when we say you do!
  - which optimizes battery life
  - is not at expense of interaction

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120
- iOS: you get the update when we say you do!
  - which optimizes battery life
  - is not at expense of interaction
  - minimizes bus traffic on chip

step counting

- do not rely on the update to be:
  - reliable
  - what you asked for
  - have any regularity, sometimes 5 steps, sometimes 120
- iOS: you get the update when we say you do!
  - which optimizes battery life
  - is not at expense of interaction
  - minimizes bus traffic on chip
  - and will keep track even if your app is in the background

querying past steps

```
// example of querying steps from certain dates
  NSDate *now = [NSDate date];
  NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];



  [self.cmStepCounter queryStepCountStartingFrom:from
                                              to:now
                                         toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSInteger numberOfSteps, NSError *error) {
     NSLog(@"%ld Steps Taken from %@ to %@",(long)numberOfSteps,from,now);
  }];
```

querying past steps

```
// example of querying steps from certain dates
  NSDate *now = [NSDate date];
  NSDate *from = [NSDate dateWithTimeInterval:-60*60*24 sinceDate:now];



  [self.cmStepCounter queryStepCountStartingFrom:from
                                              to:now
                                         toQueue:[NSOperationQueue mainQueue]
  withHandler:^(NSInteger numberOfSteps, NSError *error) {
     NSLog(@"%ld Steps Taken from %@ to %@",(long)numberOfSteps,from,now);
  }];
```

handle error!

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];




 // synchronize the settings
 [standardUserDefaults synchronize];
```

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

import defaults

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

  // saving an NSInteger
  [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
  [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
  [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

  // saving an object
  [standardUserDefaults setObject:myObject forKey:@"someObject"];




  // synchronize the settings
  [standardUserDefaults synchronize];
```

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

import defaults

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];




 // synchronize the settings
 [standardUserDefaults synchronize];
```

primitives
(nil if not defined)

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

import defaults

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];




 // synchronize the settings
 [standardUserDefaults synchronize];
```

primitives
(nil if not defined)

objects

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];
```

import defaults

primitives
(nil if not defined)

objects

NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary

```
 // synchronize the settings
 [standardUserDefaults synchronize];
```

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];
```

import defaults

primitives
(nil if not defined)

objects

NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary

these are objects!

```
 // synchronize the settings
 [standardUserDefaults synchronize];
```

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

import defaults

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];
```

primitives
(nil if not defined)

objects

NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary

```
 // synchronize the settings
 [standardUserDefaults synchronize];
```

these are objects!

these are property lists, if:
they contain only objects!

storing persistent defaults

- iOS supports NSUserDefaults for primitives and encapsulated data (or lists of)

import defaults

```
// standardUserDefaults variable
 NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 // saving an NSInteger
 [standardUserDefaults setInteger:252 forKey:@"primitiveInteger"];
 [standardUserDefaults setDouble:M_PI forKey:@"primitiveDouble"];
 [standardUserDefaults setFloat:M_PI forKey:@"primitiveFloat"];

 // saving an object
 [standardUserDefaults setObject:myObject forKey:@"someObject"];
```

primitives
(nil if not defined)

objects

`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`

these are objects!

```
 // synchronize the settings
 [standardUserDefaults synchronize];
```

save any changes

these are property lists, if:
they contain only objects!

user defaults

key value behavior for setting and getting!

user defaults

key value behavior for setting and getting!

```
 _dailyStepsGoal = @(50);

NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];
```

user defaults

key value behavior for setting and getting!

```
_dailyStepsGoal = @(50);

NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

NSInteger dailyStepGoalFromUser = [standardUserDefaults
                                      integerForKey:@"dailyStepGoal"];
```

user defaults

key value behavior for setting and getting!

```
 _dailyStepsGoal = @(50);

NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

 NSInteger dailyStepGoalFromUser = [standardUserDefaults
                                     integerForKey:@"dailyStepGoal"];


 if(!dailyStepGoalFromUser){
     [standardUserDefaults setInteger:[_dailyStepsGoal intValue]
                               forKey:@"dailyStepGoal"];
     [standardUserDefaults synchronize];
}
else{
    _dailyStepsGoal = @(dailyStepGoalFromUser);
}
```

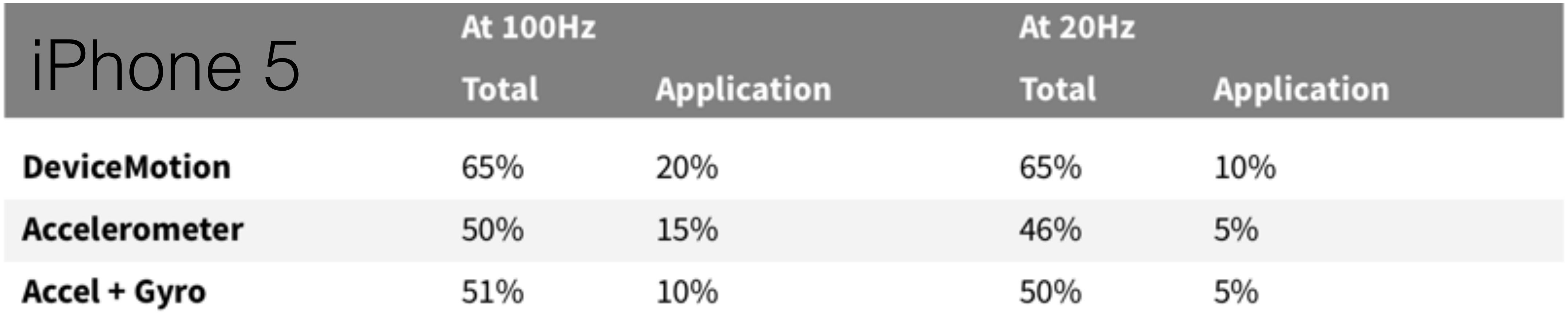
M7 step/activity demo

M7 “raw” motion data

- M7 mediates access to data
- much lower battery consumption

M7 “raw” motion data

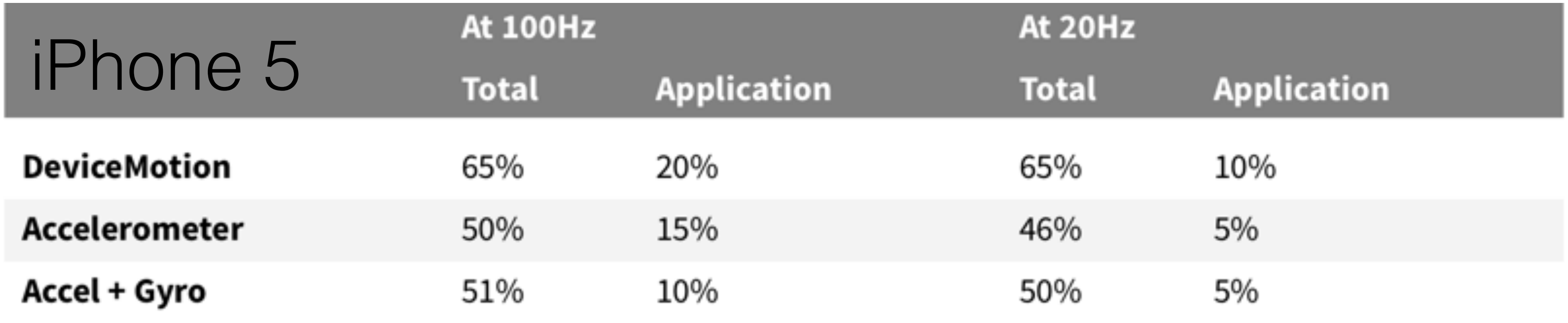
- M7 mediates access to data
- much lower battery consumption



iPhone 5	At 100Hz		At 20Hz	
	Total	Application	Total	Application
DeviceMotion	65%	20%	65%	10%
Accelerometer	50%	15%	46%	5%
Accel + Gyro	51%	10%	50%	5%

M7 “raw” motion data

- M7 mediates access to data
- much lower battery consumption

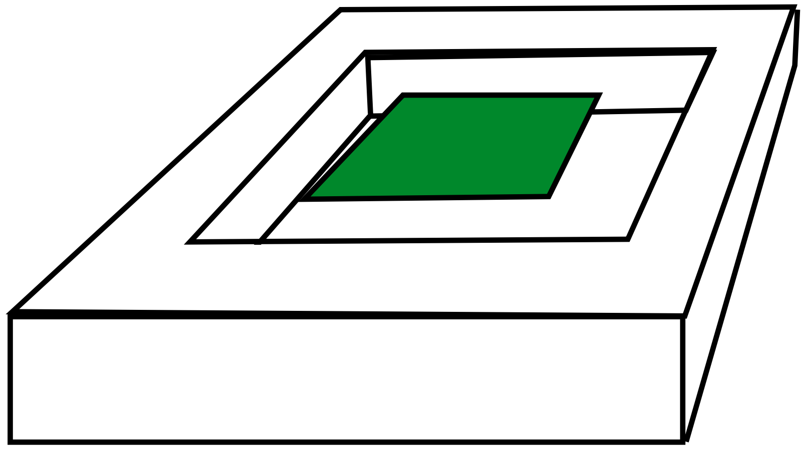


iPhone 5	At 100Hz		At 20Hz	
	Total	Application	Total	Application
DeviceMotion	65%	20%	65%	10%
Accelerometer	50%	15%	46%	5%
Accel + Gyro	51%	10%	50%	5%

iPhone 5s 4% 1%

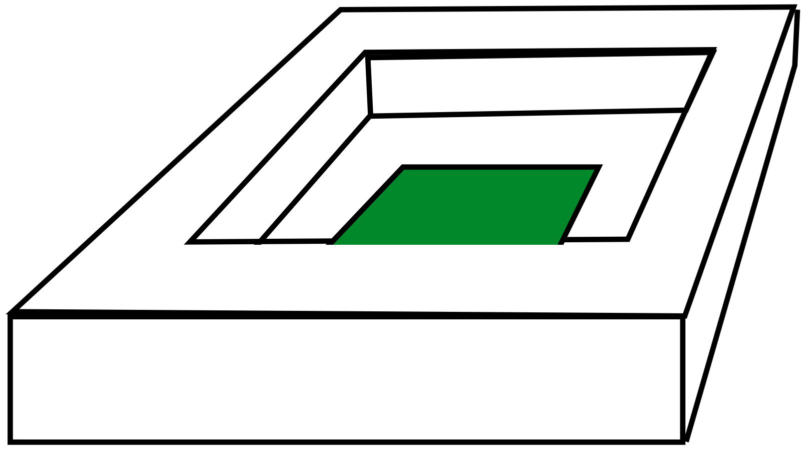
accelerometers

- how does it work?
- solid state device (fabricated on a chip)
- it has specs (not made public by Apple)
  - swing
    - +-8g (force)
  - bias and variance
    - bias can be high, easy to zero out
  - resolution
    - 20 bits or 0.000015g
  - bandwidth
    - 100Hz sampling is highest recommended



accelerometers

- how does it work?
- solid state device (fabricated on a chip)
- it has specs (not made public by Apple)
  - swing
    - +-8g (force)
  - bias and variance
    - bias can be high, easy to zero out
  - resolution
    - 20 bits or 0.000015g
  - bandwidth
    - 100Hz sampling is highest recommended

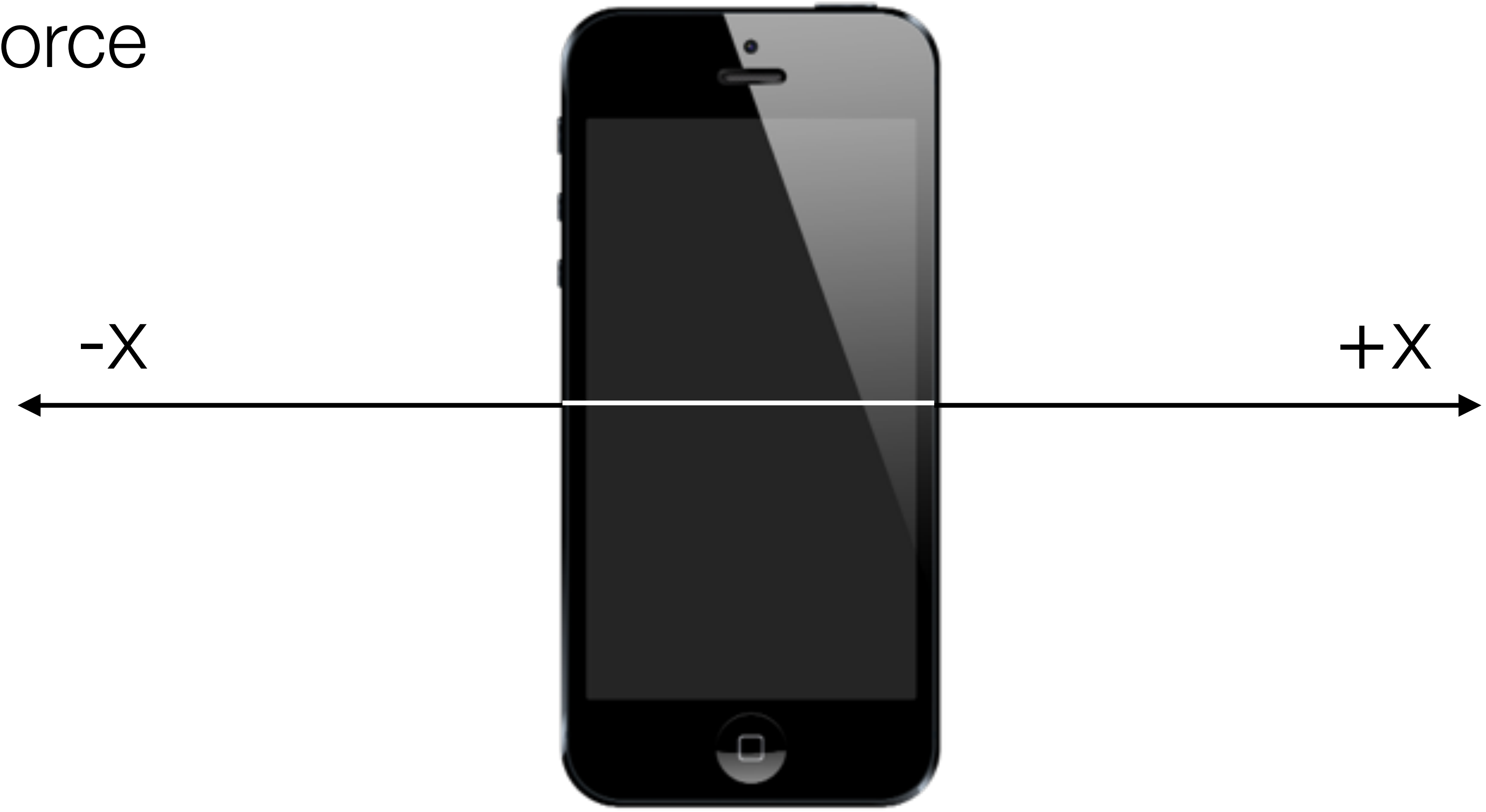


accelerometer

- measures “proper acceleration”
  - due to the weight of the device (not exactly derivative of velocity)
  - g-force

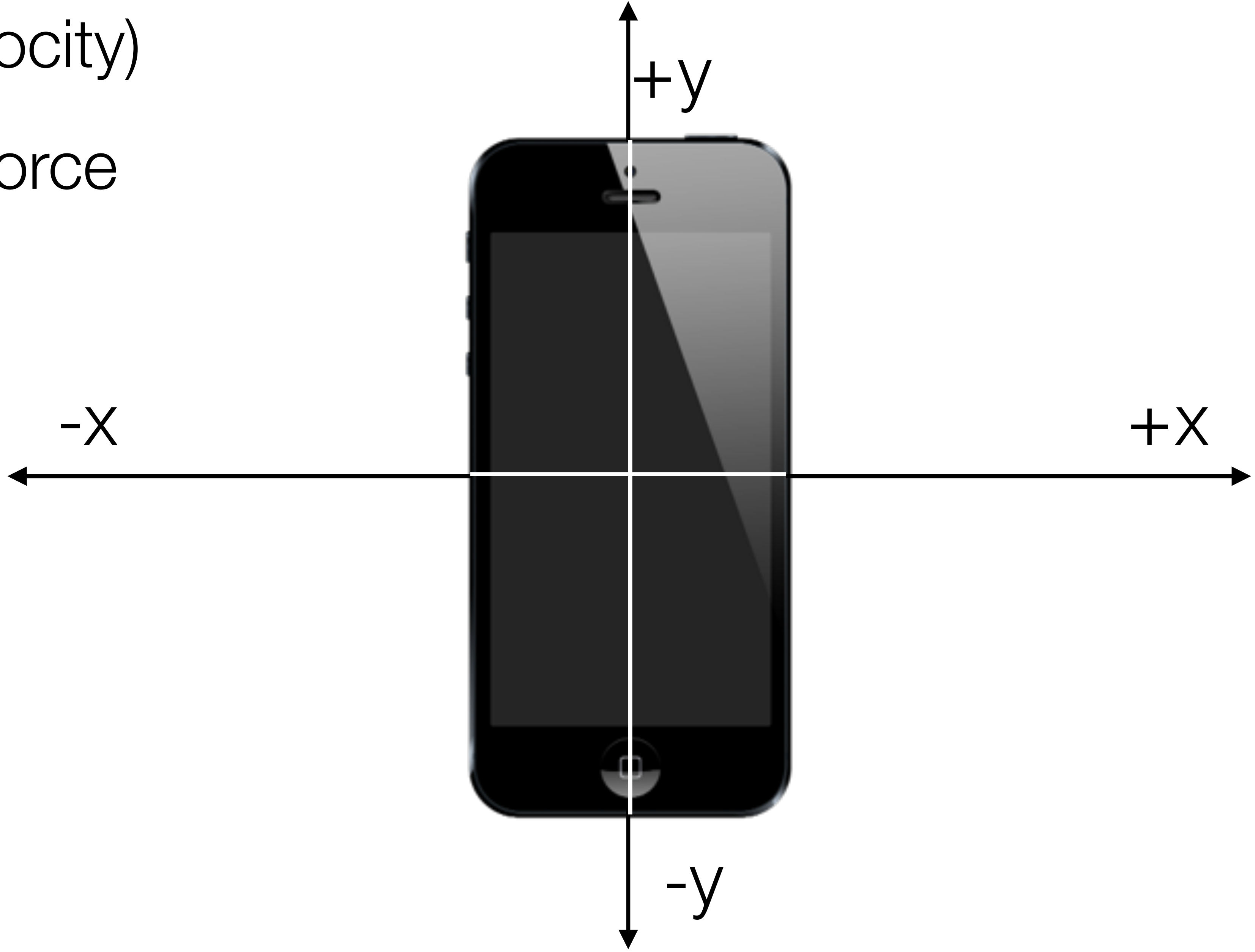
accelerometer

- measures "proper acceleration"
  - due to the weight of the device (not exactly derivative of velocity)
  - g-force



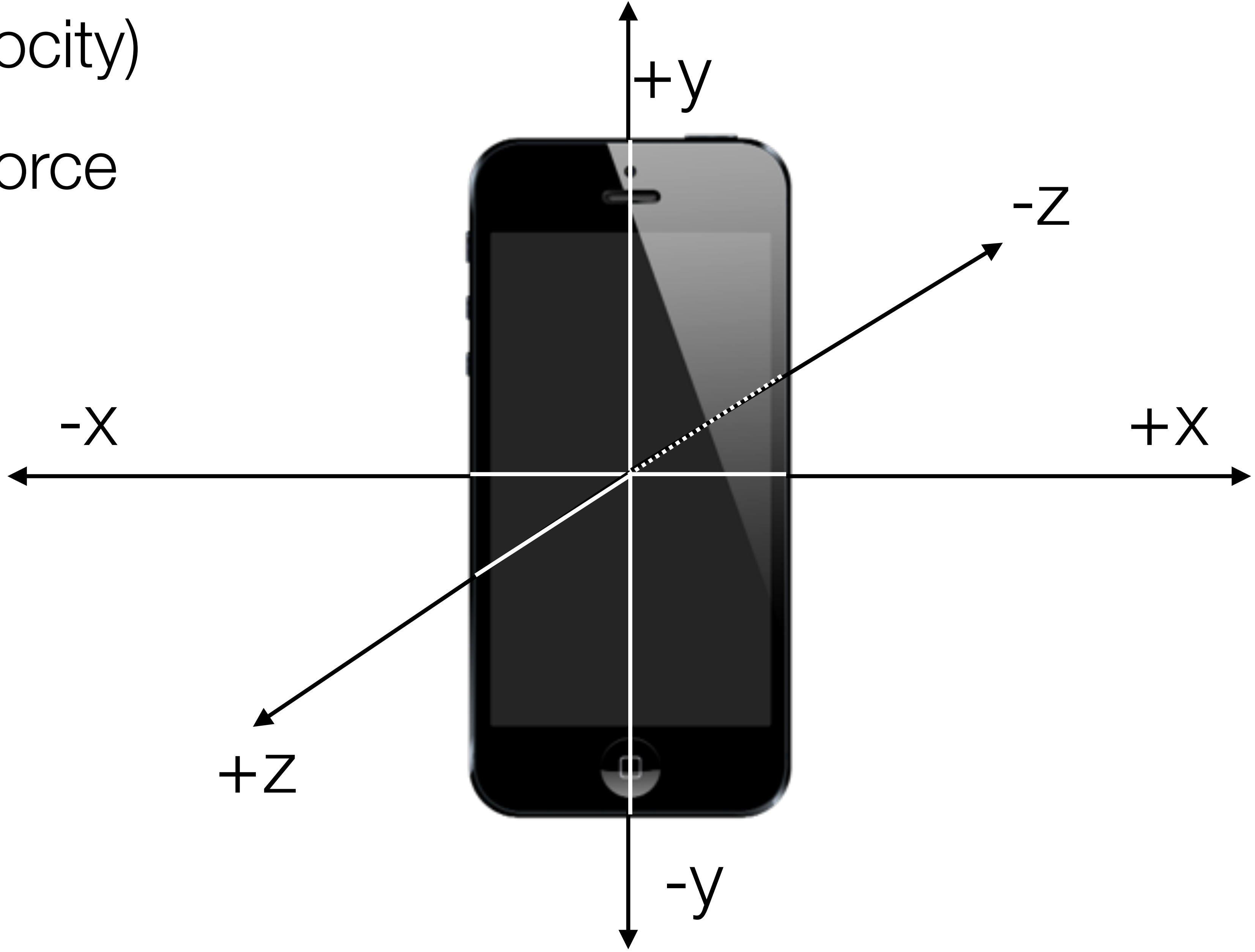
accelerometer

- measures “proper acceleration”
  - due to the weight of the device (not exactly derivative of velocity)
  - g-force



accelerometer

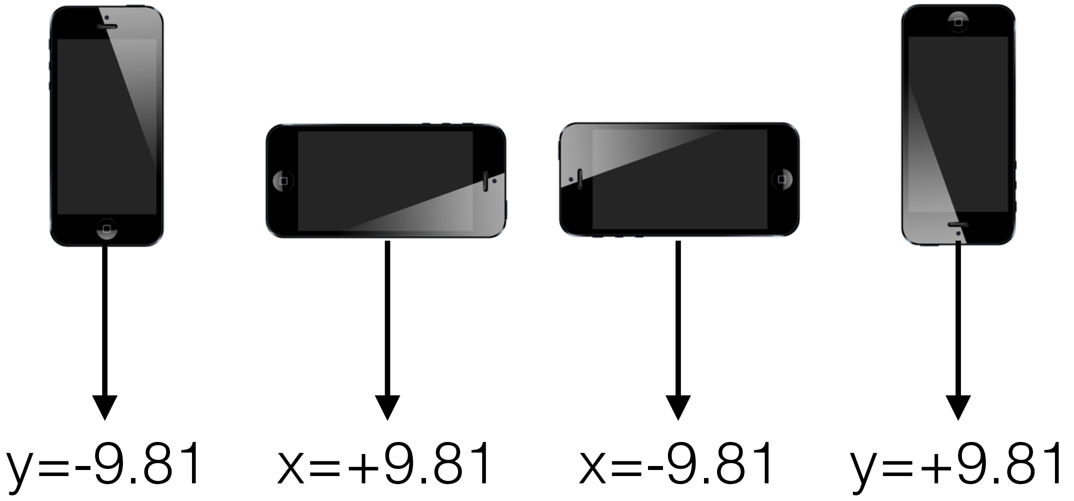
- measures “proper acceleration”
  - due to the weight of the device (not exactly derivative of velocity)
  - g-force



accessing the accelerometer

- usually don’t want the raw accelerometer value
- gravity is always pulling “down” on the device at a constant force of ~9.81g
- the core motion API automatically subtracts gravity from the user acceleration

```
CMDeviceMotion *deviceMotion

deviceMotion.gravity
deviceMotion.userAcceleration

CMAcceleration gravity, CMAcceleration userAcceleration

gravity.x;
gravity.y;
gravity.z;

userAcceleration.x;
userAcceleration.y;
userAcceleration.z;
```



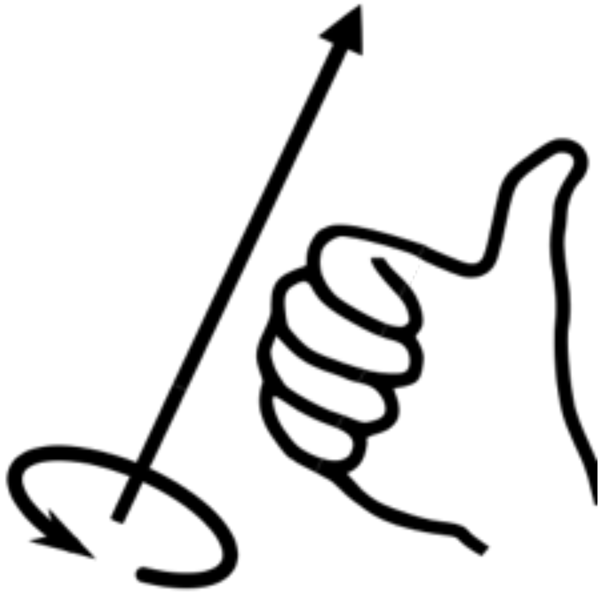
gyroscope

- measures the rate of rotation of the device
- MEMs device
  - essentially a microscopic, vibrating plate that resists motion

so it knows force in any rotating direction

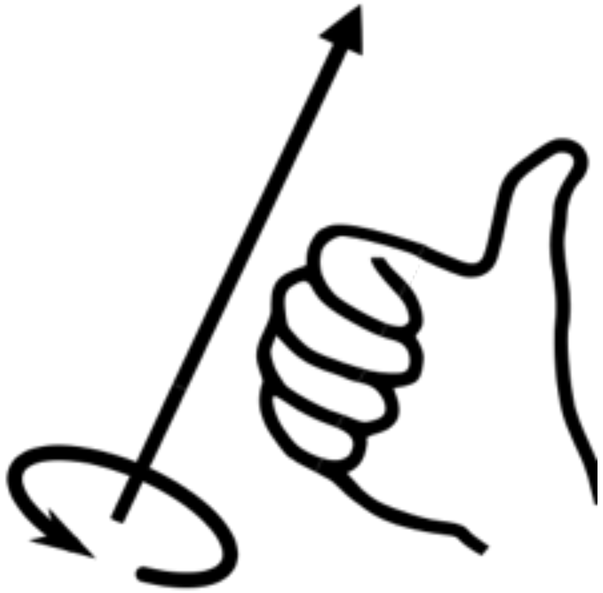
gyroscope

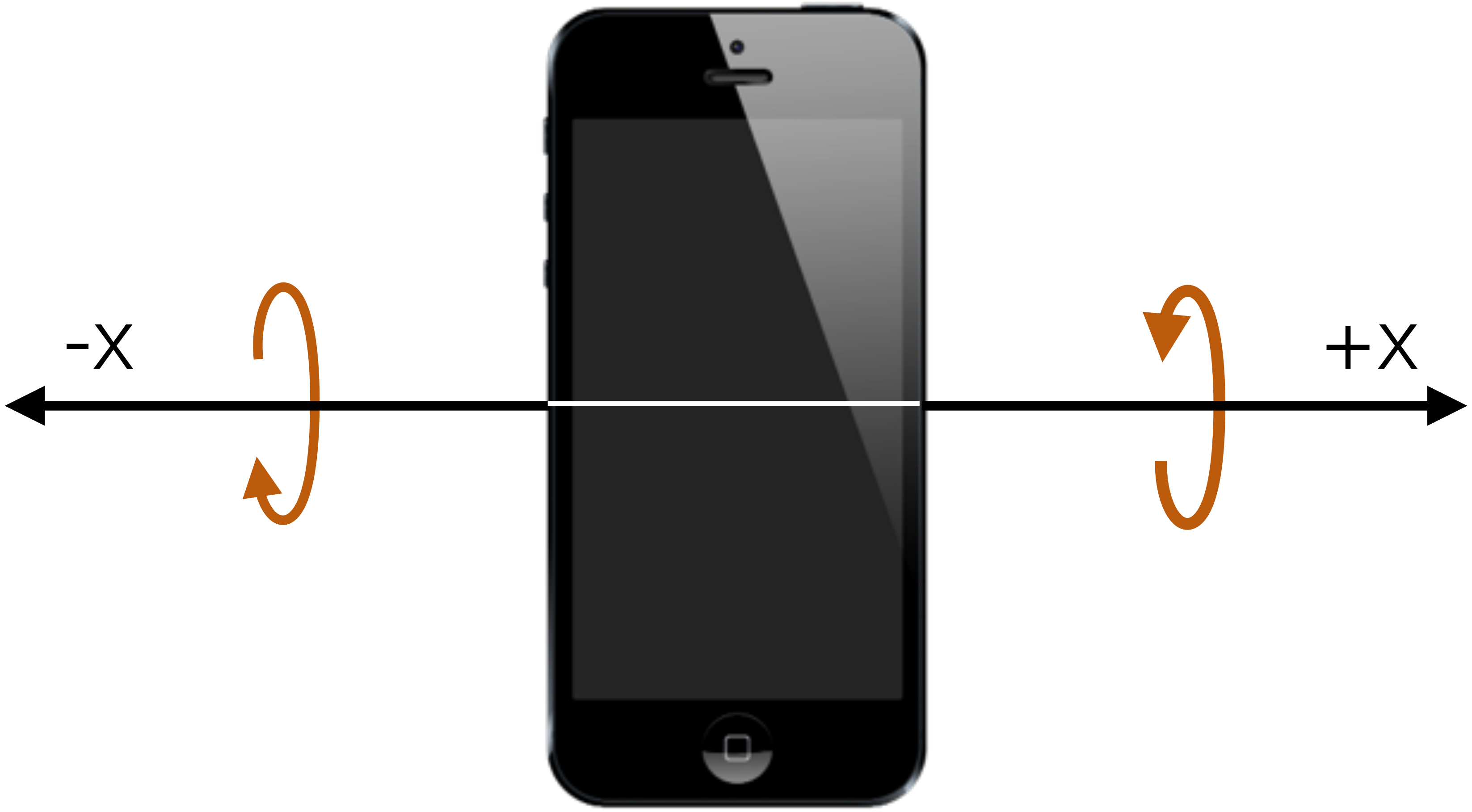
- the “right hand rule”



gyroscope

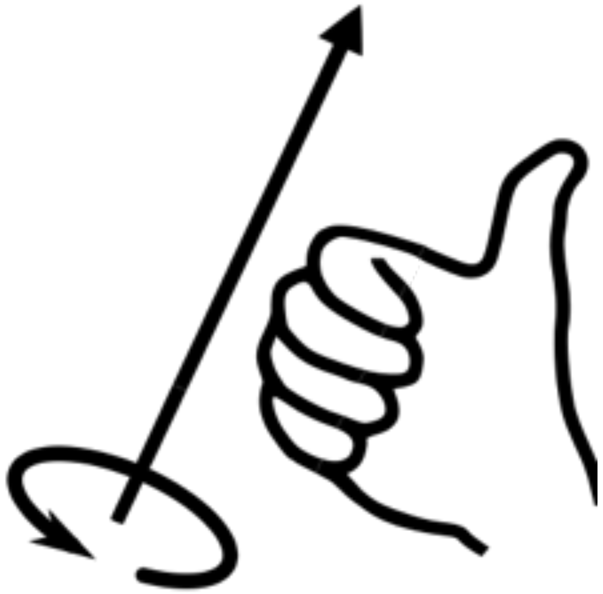
- the “right hand rule”

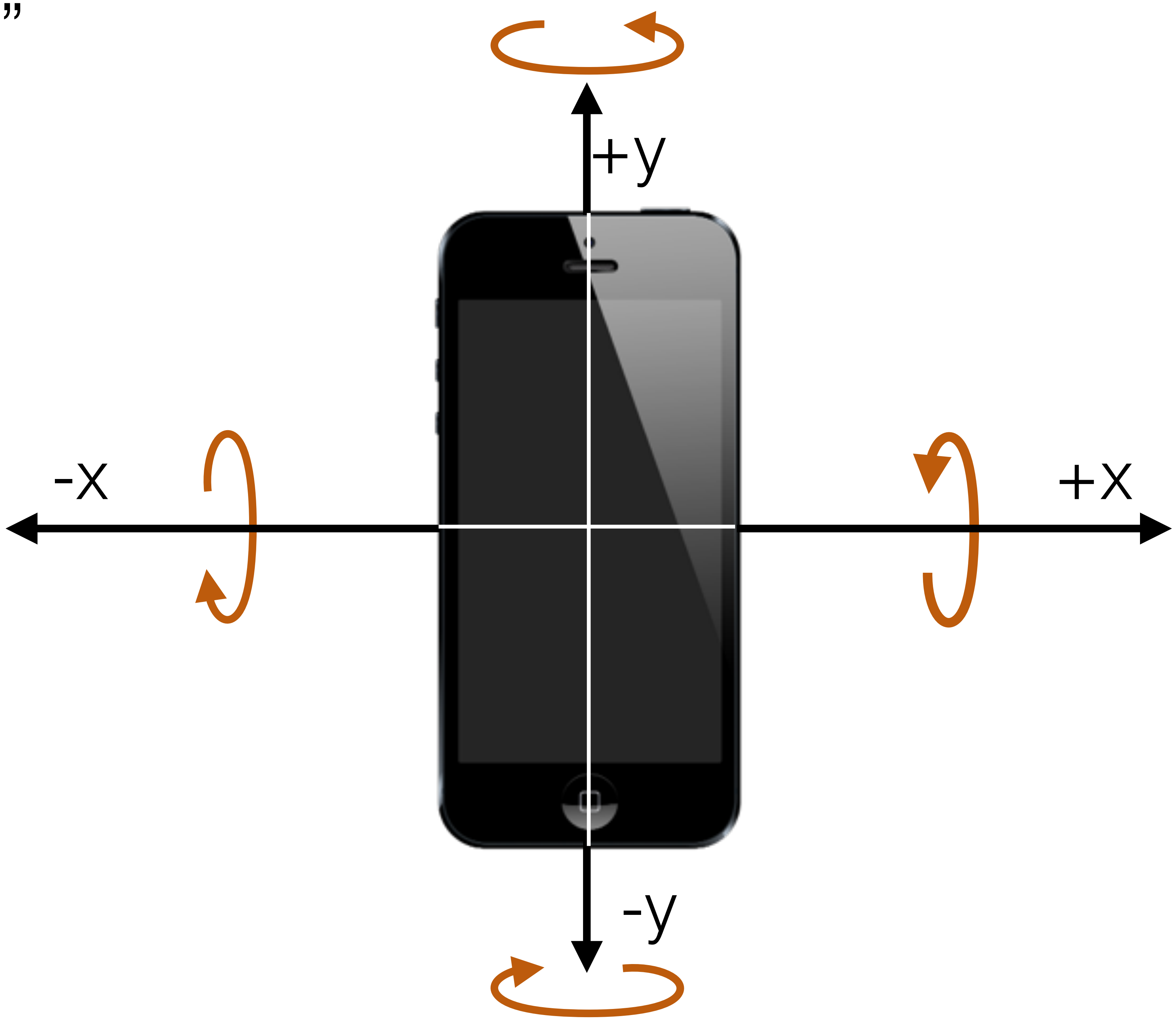




gyroscope

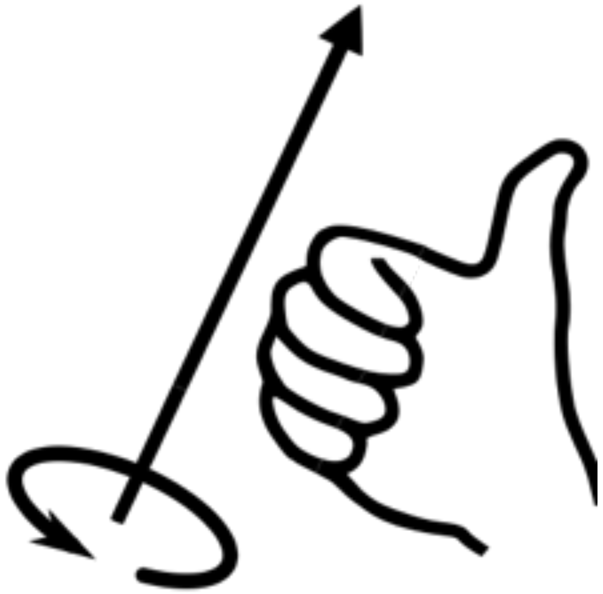
- the “right hand rule”

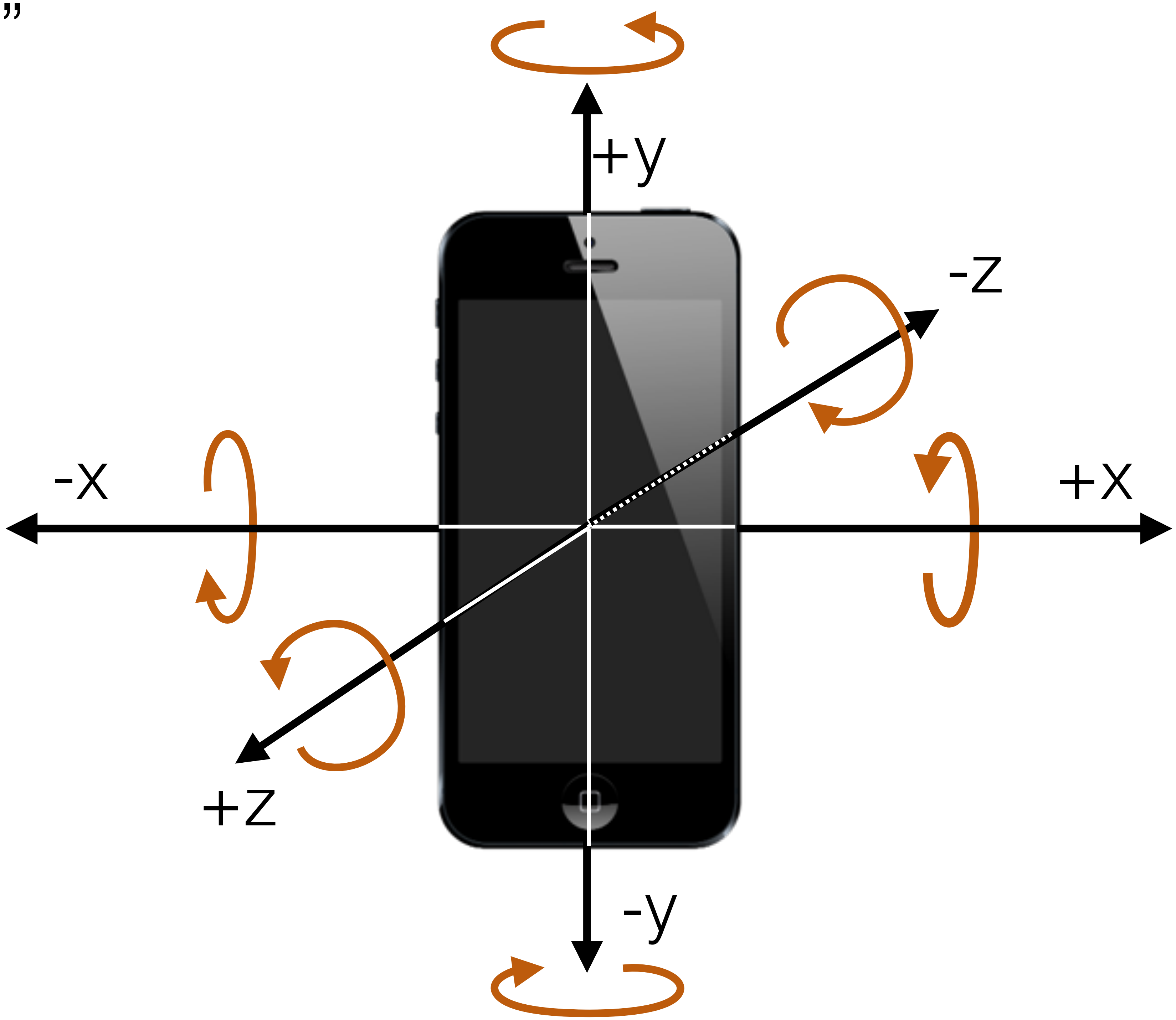




gyroscope

- the "right hand rule"

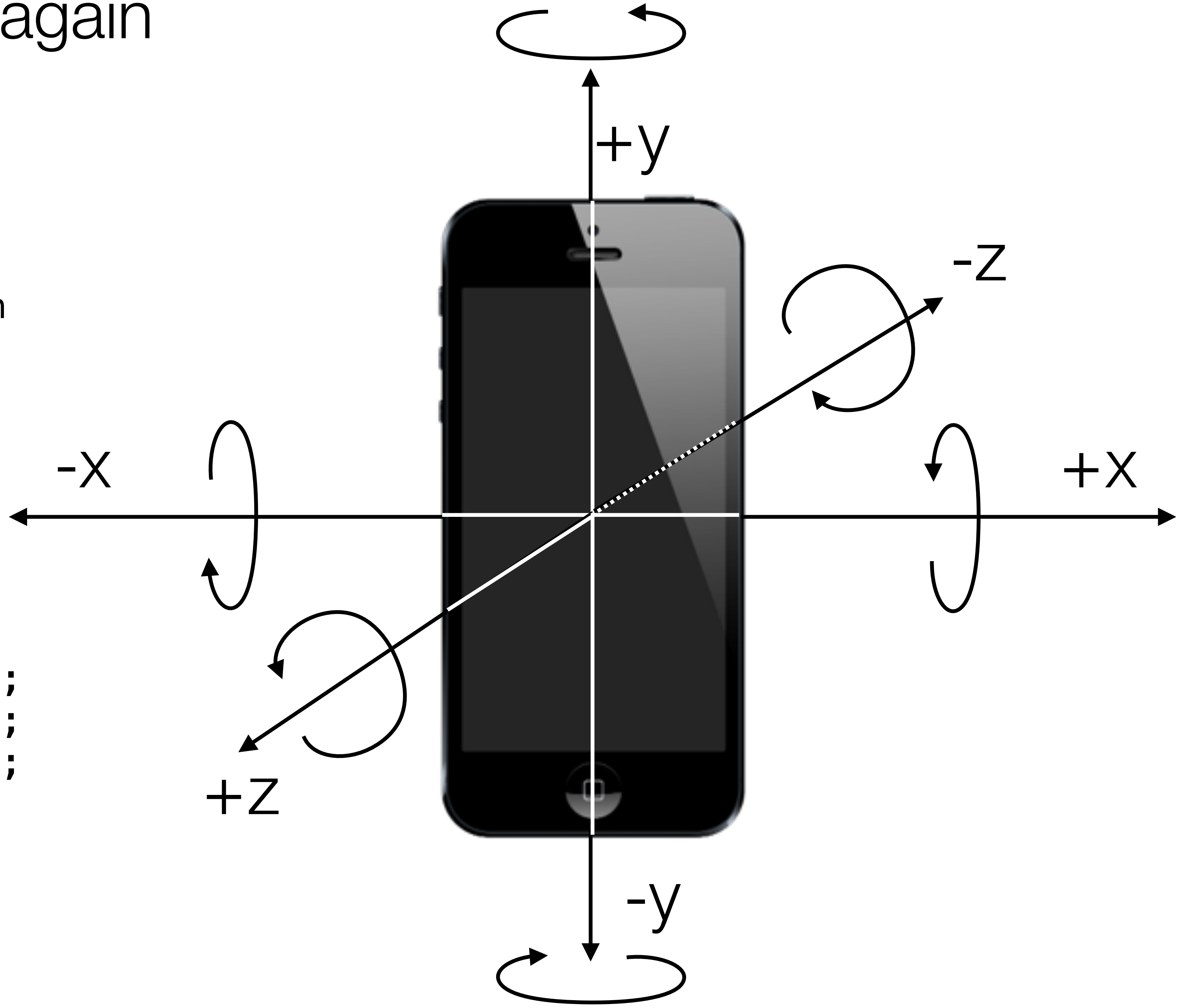




accessing the gyro

- use device motion again

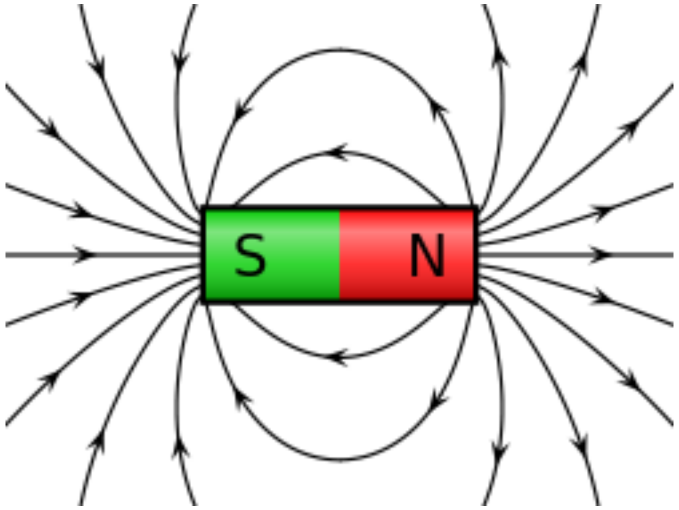
```
CMDeviceMotion *deviceMotion

deviceMotion.rotationRate
```

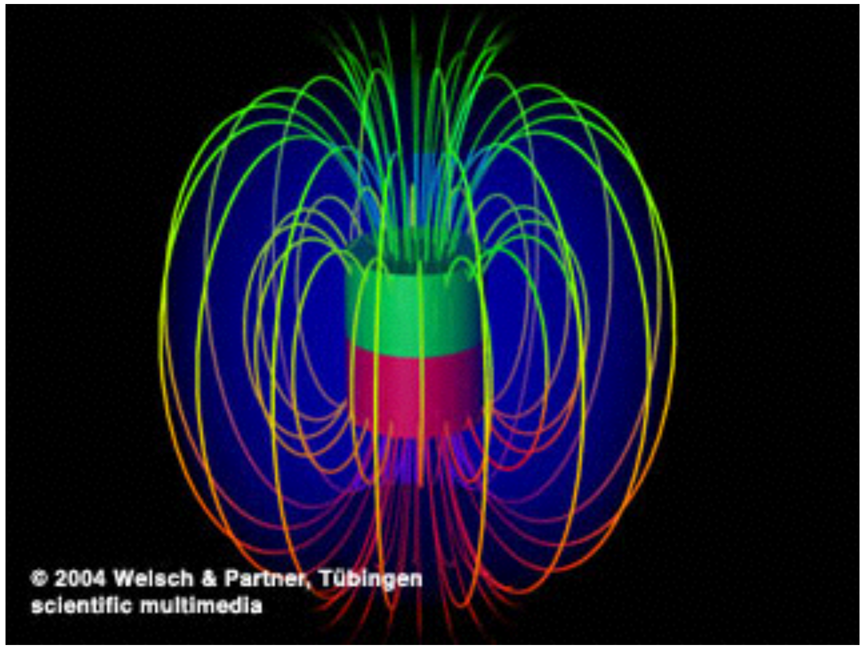


```
CMRotationRate rotationRate
rotX[head] = rotationRate.x;
rotY[head] = rotationRate.y;
rotZ[head] = rotationRate.z;
```

magnetometers

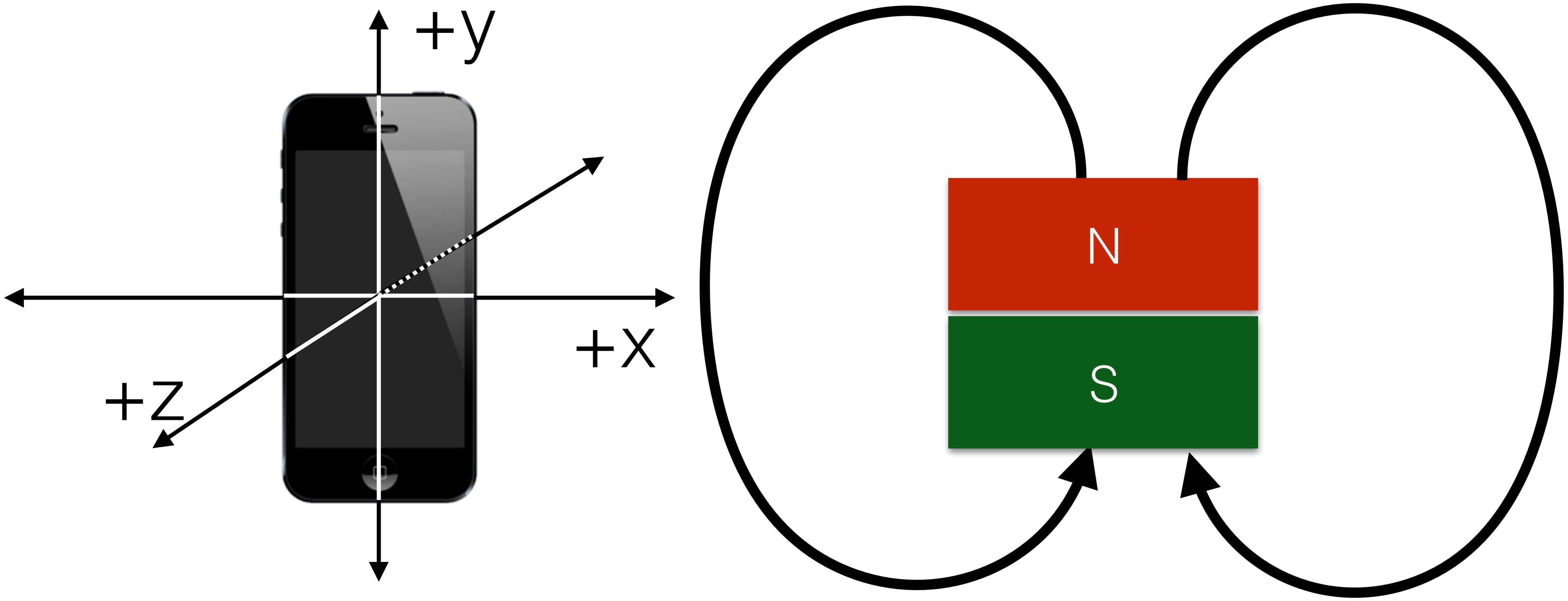
- measure magnetic fields
- magnets are measured in tesla (T)
    - how: essentially, there is a tight coupling between electricity flow and magnetic fields
- earth’s magnetic field varies, but is around 50 uT
- iPhone can measure up to 1T with a resolution of about 8uT
- magnetic fields have direction!





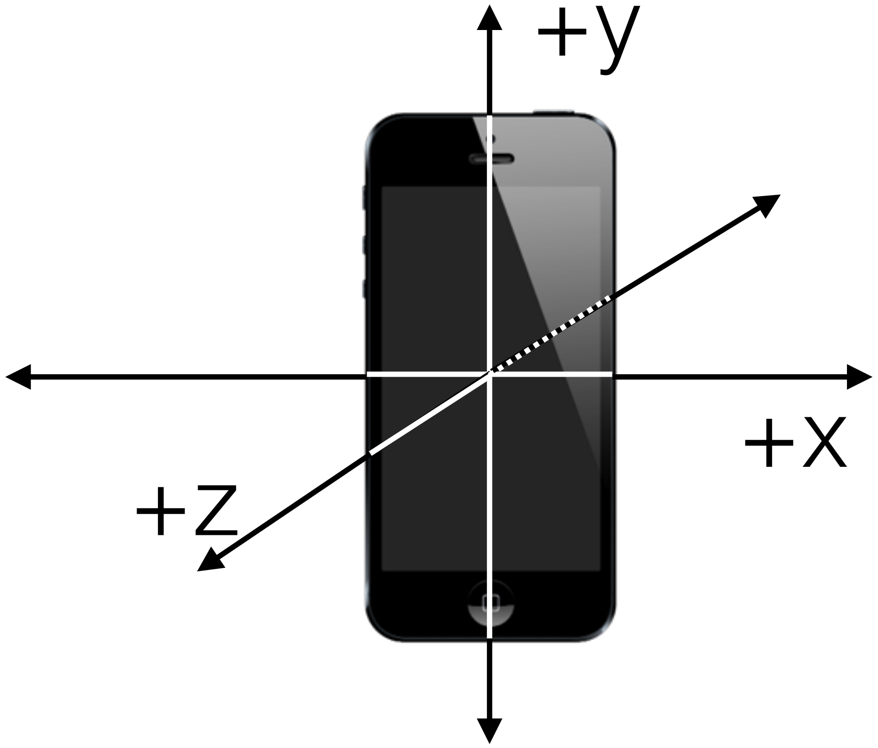
magnetic fields

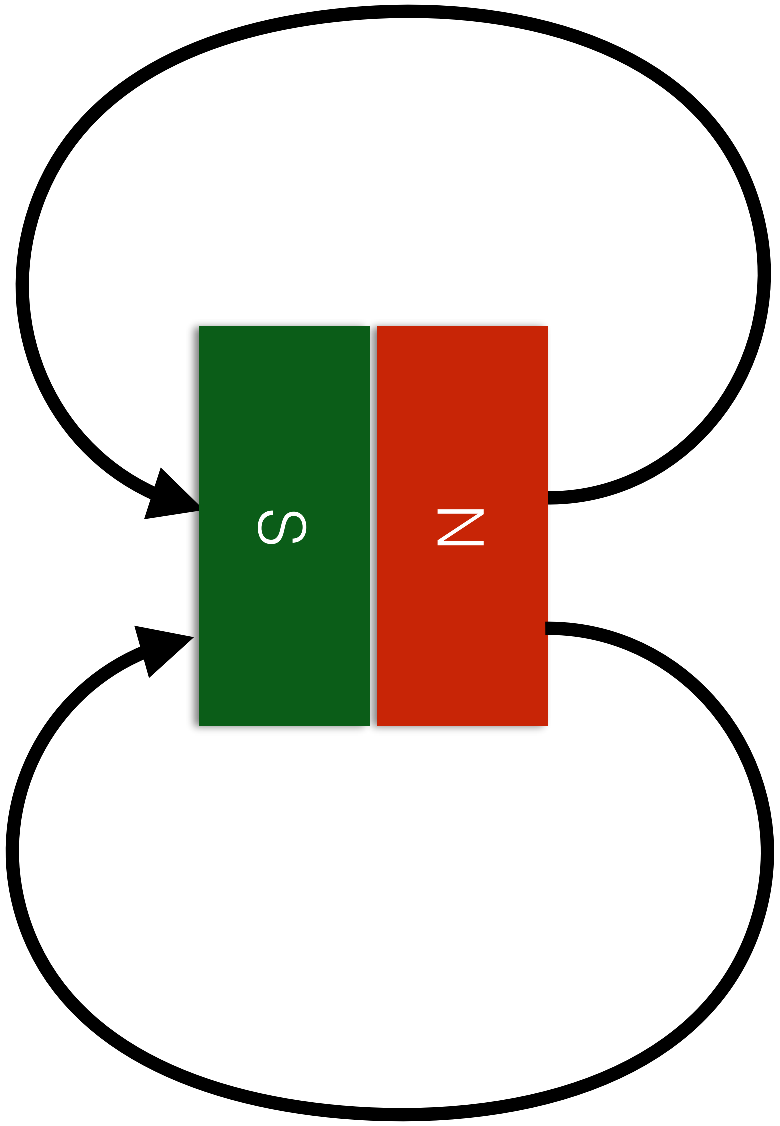
- measure magnetic field along axis, towards “south”



magnetic fields

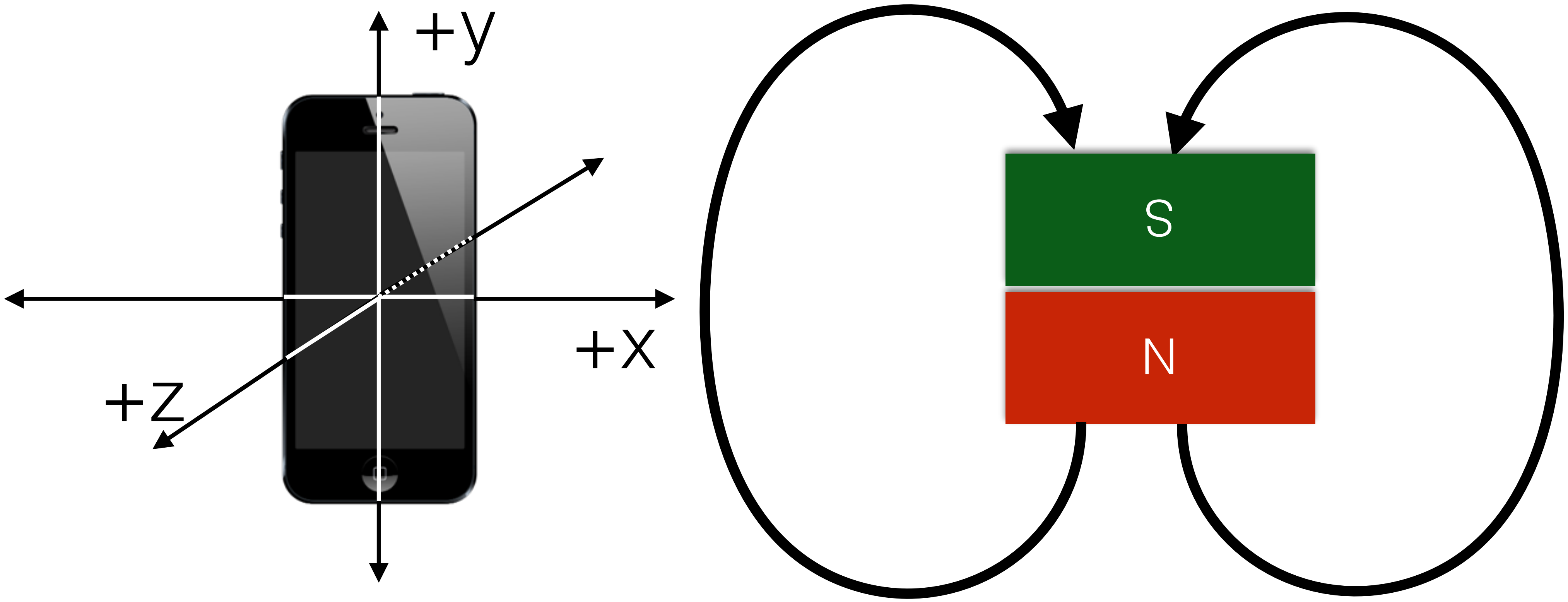
- measure magnetic field along axis, towards "south"





magnetic fields

- measure magnetic field along axis, towards “south”



but iPhone has magnetic bias

- the phone uses electricity and therefore is a magnet
  - good thing Apple subtracts that out for us!

but iPhone has magnetic bias

- the phone uses electricity and therefore is a magnet
  - good thing Apple subtracts that out for us!

```
CMDeviceMotion *deviceMotion

deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

magneticField.field.x
magneticField.field.y
magneticField.field.z

magneticField.accuracy
```

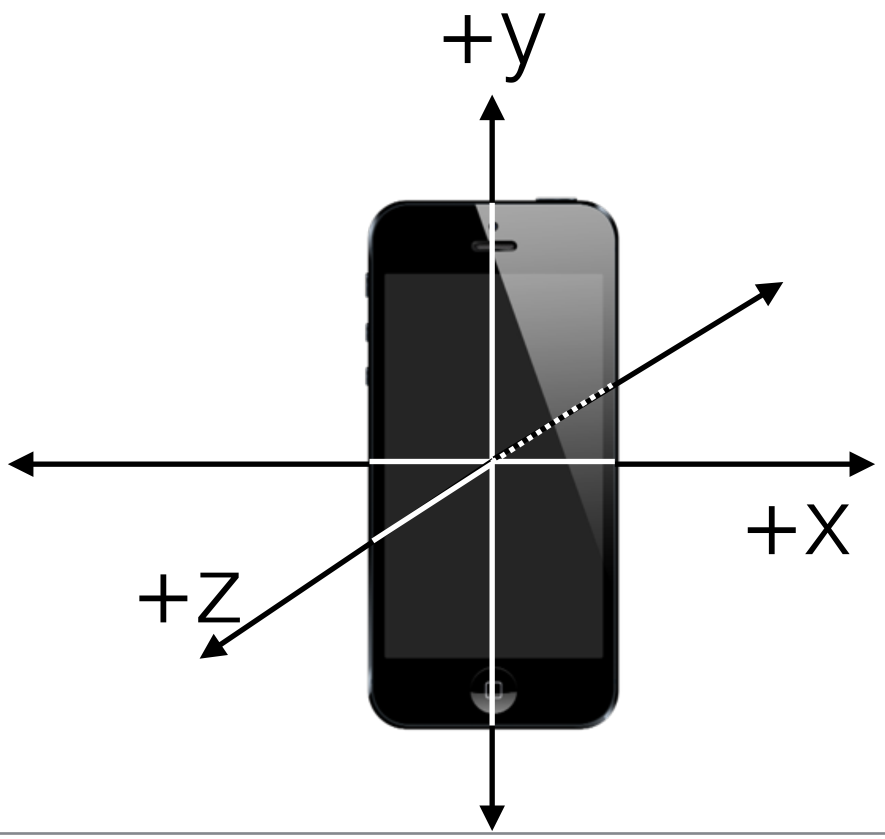
but iPhone has magnetic bias

- the phone uses electricity and therefore is a magnet
  - good thing Apple subtracts that out for us!

```
 CMDeviceMotion *deviceMotion

 deviceMotion.magneticField
 CMCalibratedMagneticField magneticField;

 magneticField.field.x
 magneticField.field.y
 magneticField.field.z

 magneticField.accuracy


CMMagneticFieldCalibrationAccuracyUncalibrated = -1,
    CMMagneticFieldCalibrationAccuracyLow,
    CMMagneticFieldCalibrationAccuracyMedium,
    CMMagneticFieldCalibrationAccuracyHigh
```

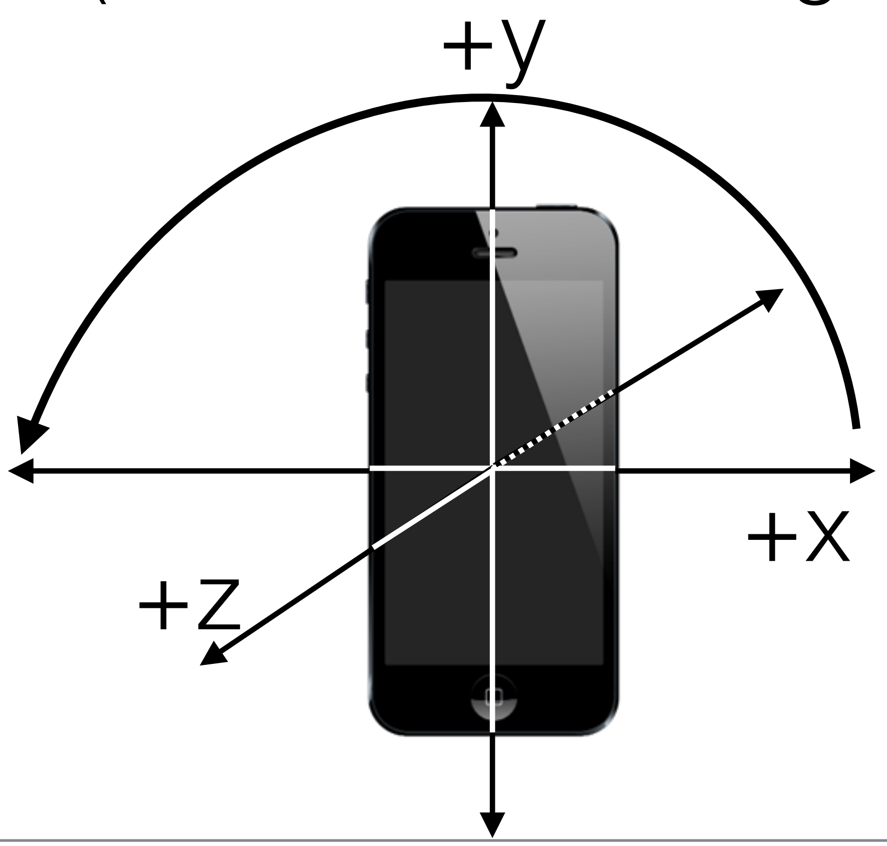
attitude

- attitude is roll, pitch, and yaw
- these are "fused" measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



attitude

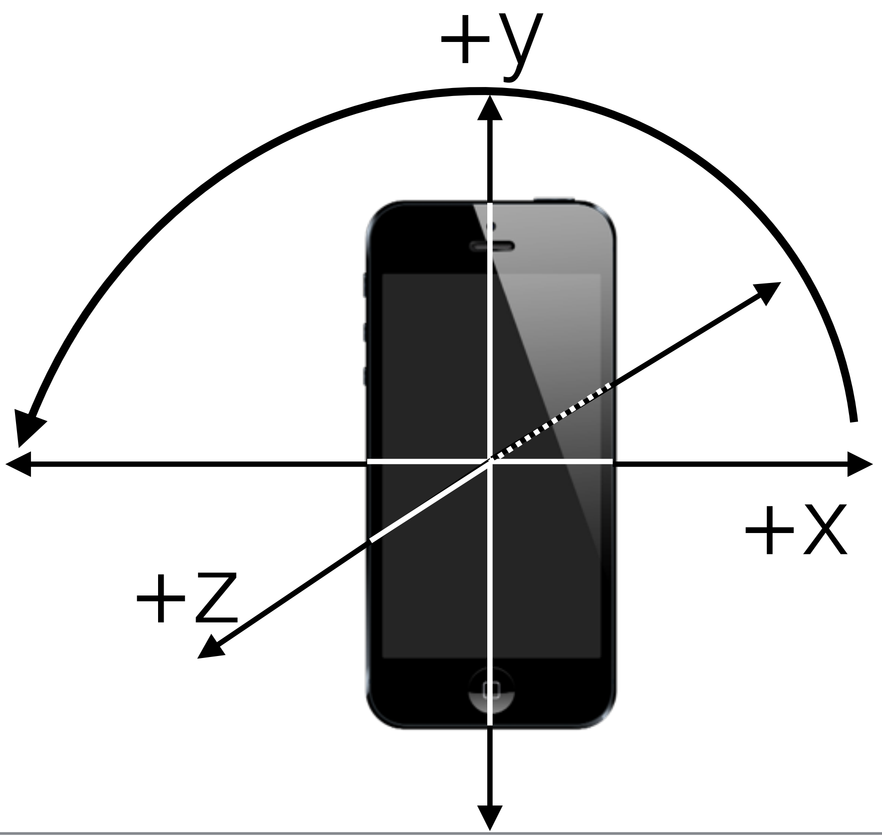
- attitude is roll, pitch, and yaw
- these are “fused” measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane

attitude

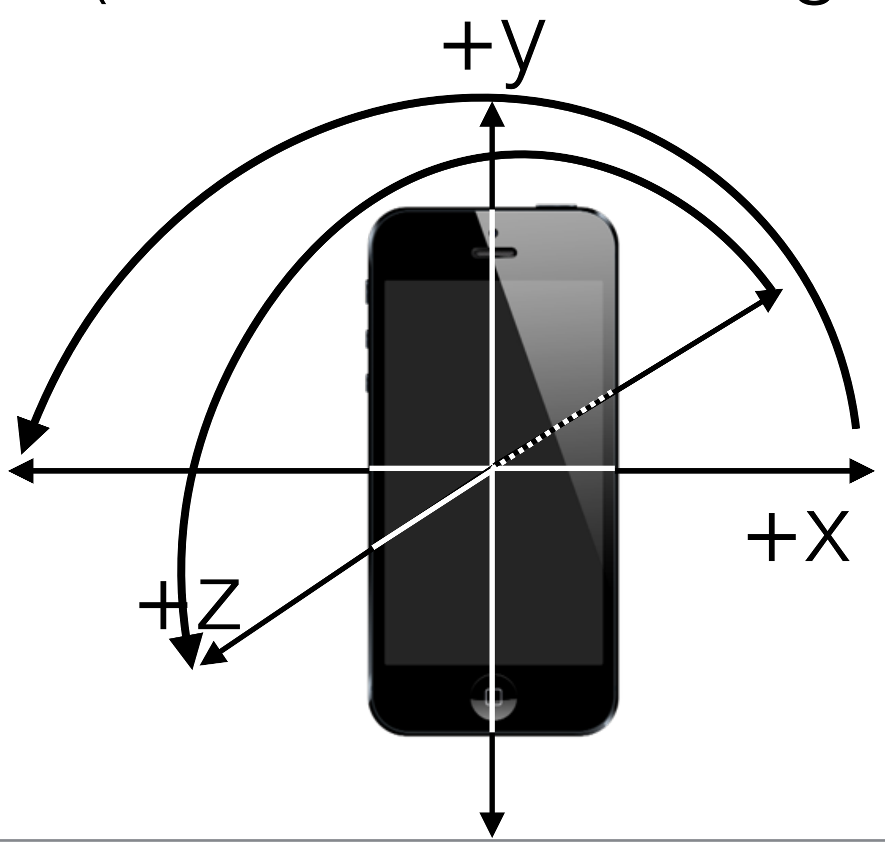
- attitude is roll, pitch, and yaw
- these are “fused” measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane

attitude

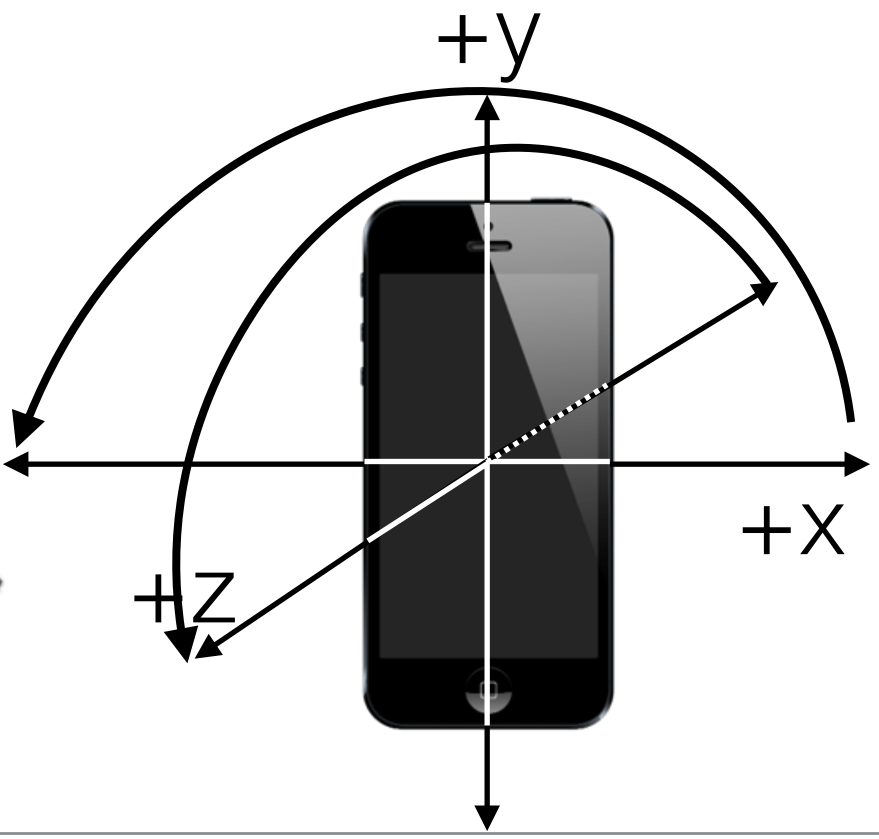
- attitude is roll, pitch, and yaw
- these are “fused” measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane
pitch in y/z plane

attitude

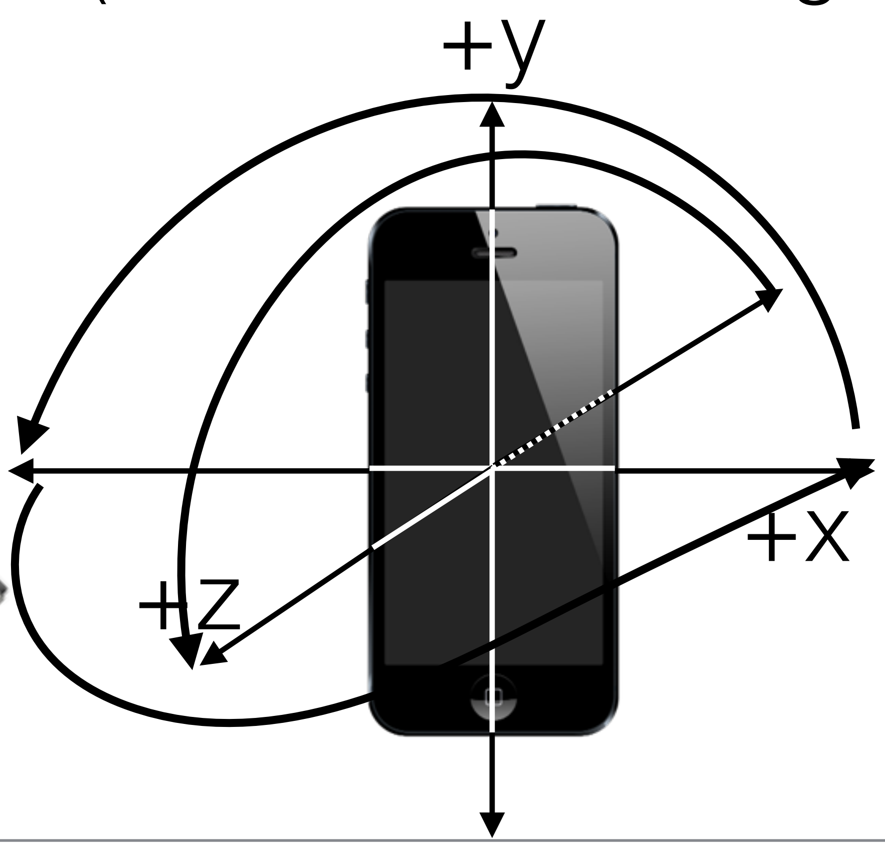
- attitude is roll, pitch, and yaw
- these are “fused” measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane
pitch in y/z plane

attitude

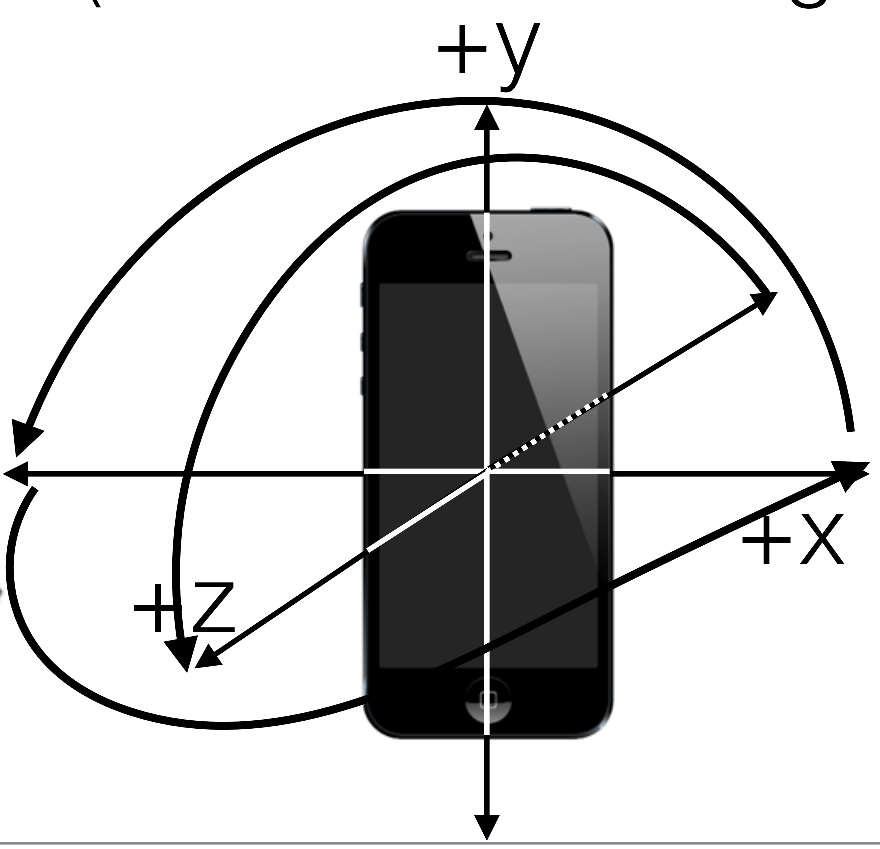
- attitude is roll, pitch, and yaw
- these are "fused" measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane
pitch in y/z plane
roll in x/z plane

attitude

- attitude is roll, pitch, and yaw
- these are “fused” measures of the device from
  - the magnetometer (used as a compass)
  - gyroscope (used for detecting quick rotations)
  - accelerometer (used for smoothing out the gyro)



yaw in x/y plane
pitch in y/z plane
roll in x/z plane

getting updates

```
   // for getting access to the fused motion data (best practice, filtered)
   @property (nonatomic,strong) CMMotionManager *mManager;



   self.mManager = [[CMMotionManager alloc] init];

    if([self.mManager isDeviceMotionAvailable])
    {

        [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
        [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
 withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

           //Access to all the data…
            deviceMotion.attitude,
            deviceMotion.rotationRate,
            deviceMotion.gravity,
            deviceMotion.userAcceleration,
            deviceMotion.magneticField,


        }];
    }
```

getting updates

```
// for getting access to the fused motion data (best practice, filtered)
@property (nonatomic,strong) CMMotionManager *mManager;
```

declare

```
self.mManager = [[CMMotionManager alloc] init];

 if([self.mManager isDeviceMotionAvailable])
 {

     [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
     [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

         //Access to all the data…
          deviceMotion.attitude,
          deviceMotion.rotationRate,
          deviceMotion.gravity,
          deviceMotion.userAcceleration,
          deviceMotion.magneticField,


     }];
 }
```

getting updates

```
   // for getting access to the fused motion data (best practice, filtered)
   @property (nonatomic,strong) CMMotionManager *mManager;
```

declare

instantiate

```
    self.mManager = [[CMMotionManager alloc] init];

     if([self.mManager isDeviceMotionAvailable])
     {

         [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
         [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
 withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

             //Access to all the data…
              deviceMotion.attitude,
              deviceMotion.rotationRate,
              deviceMotion.gravity,
              deviceMotion.userAcceleration,
              deviceMotion.magneticField,


         }];
     }
```

getting updates

```
   // for getting access to the fused motion data (best practice, filtered)
   @property (nonatomic,strong) CMMotionManager *mManager;
```

declare

instantiate

```
    self.mManager = [[CMMotionManager alloc] init];
```

```
     if([self.mManager isDeviceMotionAvailable])
     {
```

if device is capable

```
         [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
         [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
 withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

            //Access to all the data…
             deviceMotion.attitude,
             deviceMotion.rotationRate,
             deviceMotion.gravity,
             deviceMotion.userAcceleration,
             deviceMotion.magneticField,


         }];
     }
```

getting updates

```
// for getting access to the fused motion data (best practice, filtered)
@property (nonatomic,strong) CMMotionManager *mManager;
```

declare

instantiate

```
 self.mManager = [[CMMotionManager alloc] init];

   if([self.mManager isDeviceMotionAvailable])
   {
```

if device is capable

```
         [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
         [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

             //Access to all the data…
              deviceMotion.attitude,
              deviceMotion.rotationRate,
              deviceMotion.gravity,
              deviceMotion.userAcceleration,
              deviceMotion.magneticField,


         }];
   }
```

how often to push updates

getting updates

```
// for getting access to the fused motion data (best practice, filtered)
@property (nonatomic,strong) CMMotionManager *mManager;
```

declare

instantiate

```
 self.mManager = [[CMMotionManager alloc] init];

  if([self.mManager isDeviceMotionAvailable])
  {

      [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
      [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

         //Access to all the data…
          deviceMotion.attitude,
          deviceMotion.rotationRate,
          deviceMotion.gravity,
          deviceMotion.userAcceleration,
          deviceMotion.magneticField,


      }];
  }
```

if device is capable

queue to run on

how often to push updates

getting updates

```
  // for getting access to the fused motion data (best practice, filtered)
  @property (nonatomic,strong) CMMotionManager *mManager;


   self.mManager = [[CMMotionManager alloc] init];

    if([self.mManager isDeviceMotionAvailable])
    {

        [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
        [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
 withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

            //Access to all the data…
             deviceMotion.attitude,
             deviceMotion.rotationRate,
             deviceMotion.gravity,
             deviceMotion.userAcceleration,
             deviceMotion.magneticField,


        }];
    }
```

declare

instantiate

if device is capable

queue to run on

how often to push updates

the data

getting updates

```
// for getting access to the fused motion data (best practice, filtered)
@property (nonatomic,strong) CMMotionManager *mManager;


self.mManager = [[CMMotionManager alloc] init];

 if([self.mManager isDeviceMotionAvailable])
 {

     [self.mManager setDeviceMotionUpdateInterval:yourSamplingIntervalInSeconds];
     [self.mManager startDeviceMotionUpdatesToQueue:[NSOperationQueue mainQueue]
withHandler:^(CMDeviceMotion *deviceMotion, NSError *error) {

          //Access to all the data…
           deviceMotion.attitude,
           deviceMotion.rotationR
     }];
 }
```

declare

instantiate

if device is capable

how often to push updates

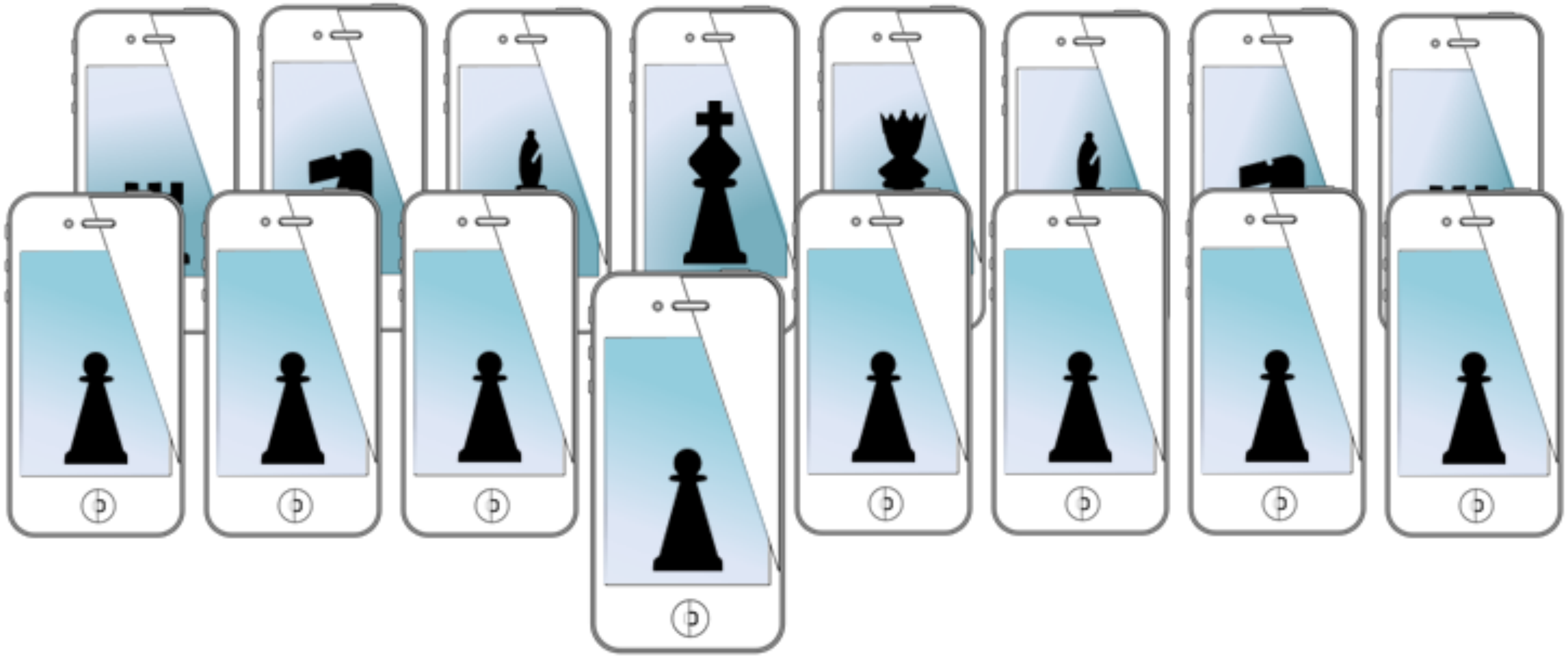
queue to run on

the data

for next time…

- basic image processing with core image

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture nine: core motion: activity, step counting, and sensor fusion

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