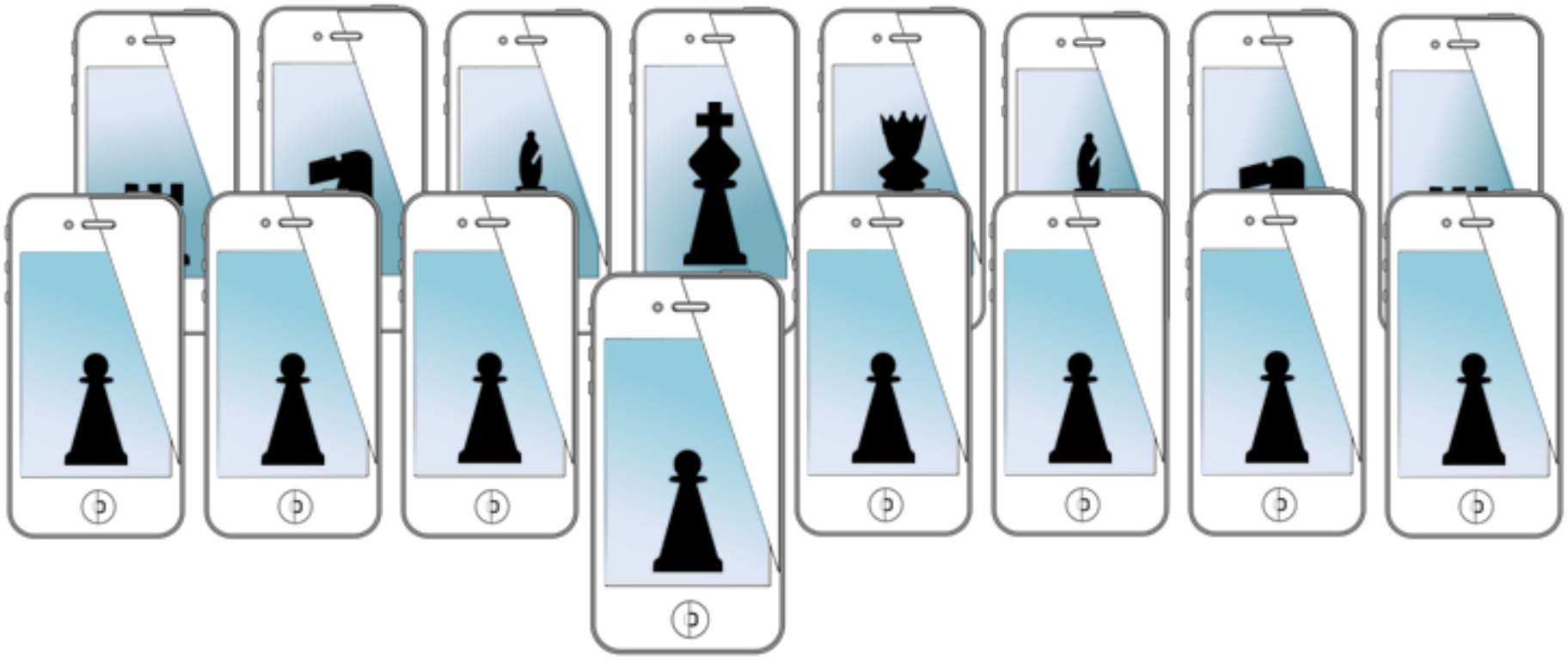
MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week 5, lecture a: doppler and activity monitoring

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

course logistics

- A1 grades coming soon
- A2 is due Friday

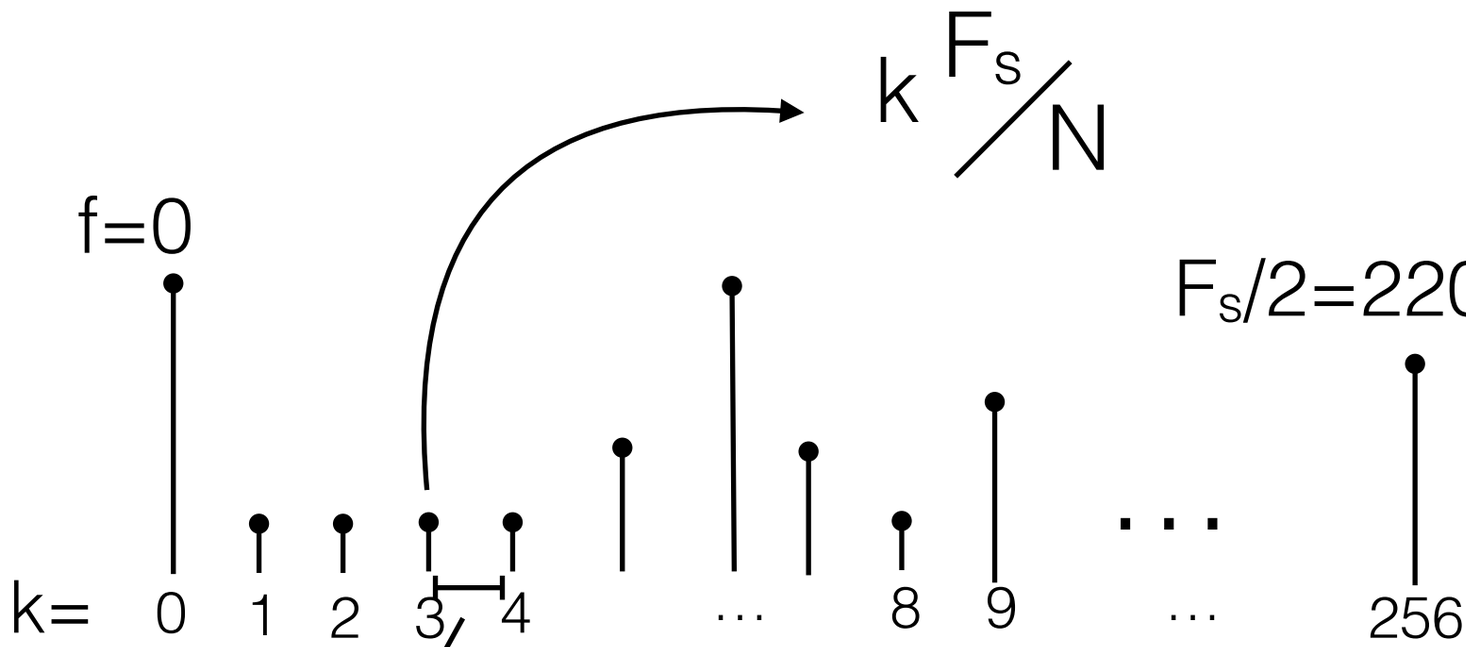
agenda

- general FFT review
- the doppler effect
- A2 explanations
- peak finding
- motion processing

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 6 Hz accuracy?

time and frequency



time array, N = 512

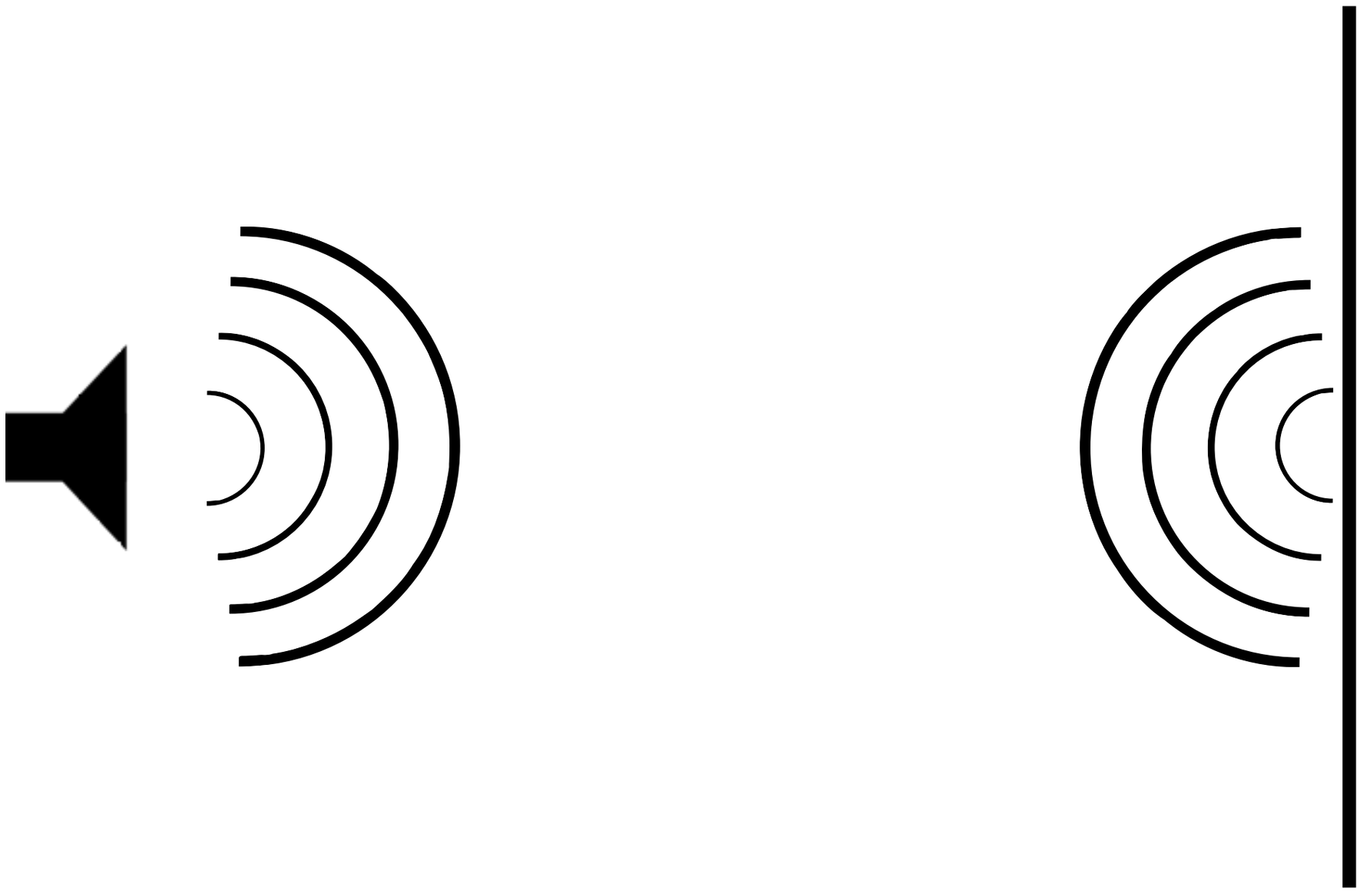
FFT array, N = 512

Spectrum, N = 257

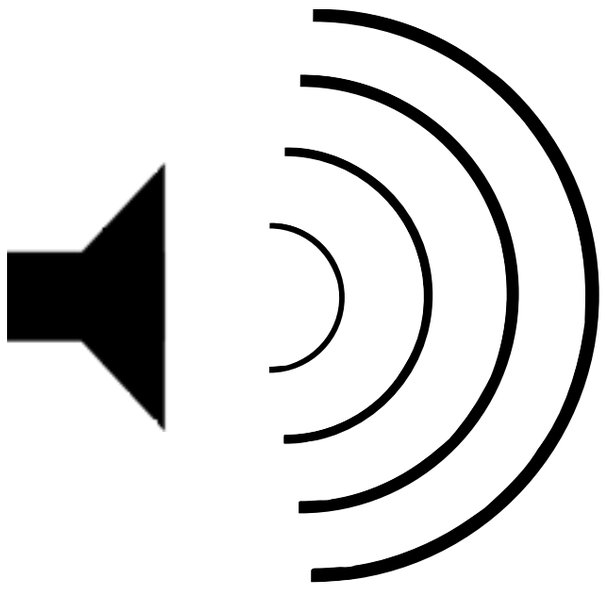
Fs = 44100 Hz

$$df = \frac{Fs/2}{N/2} = Fs/N = 44100/512 = \sim 86\ Hz$$

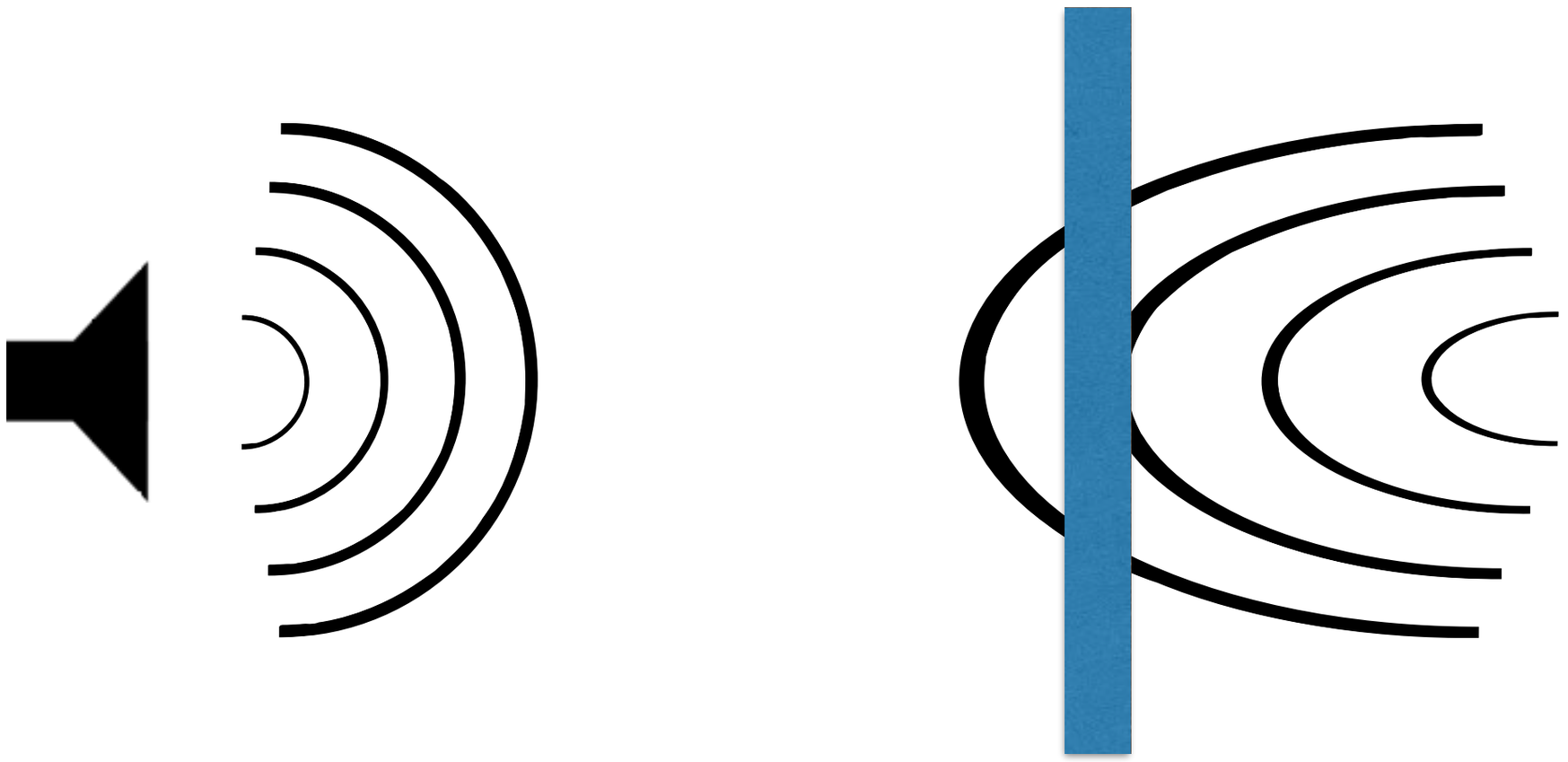
the doppler effect



the doppler effect



the doppler effect



the doppler effect

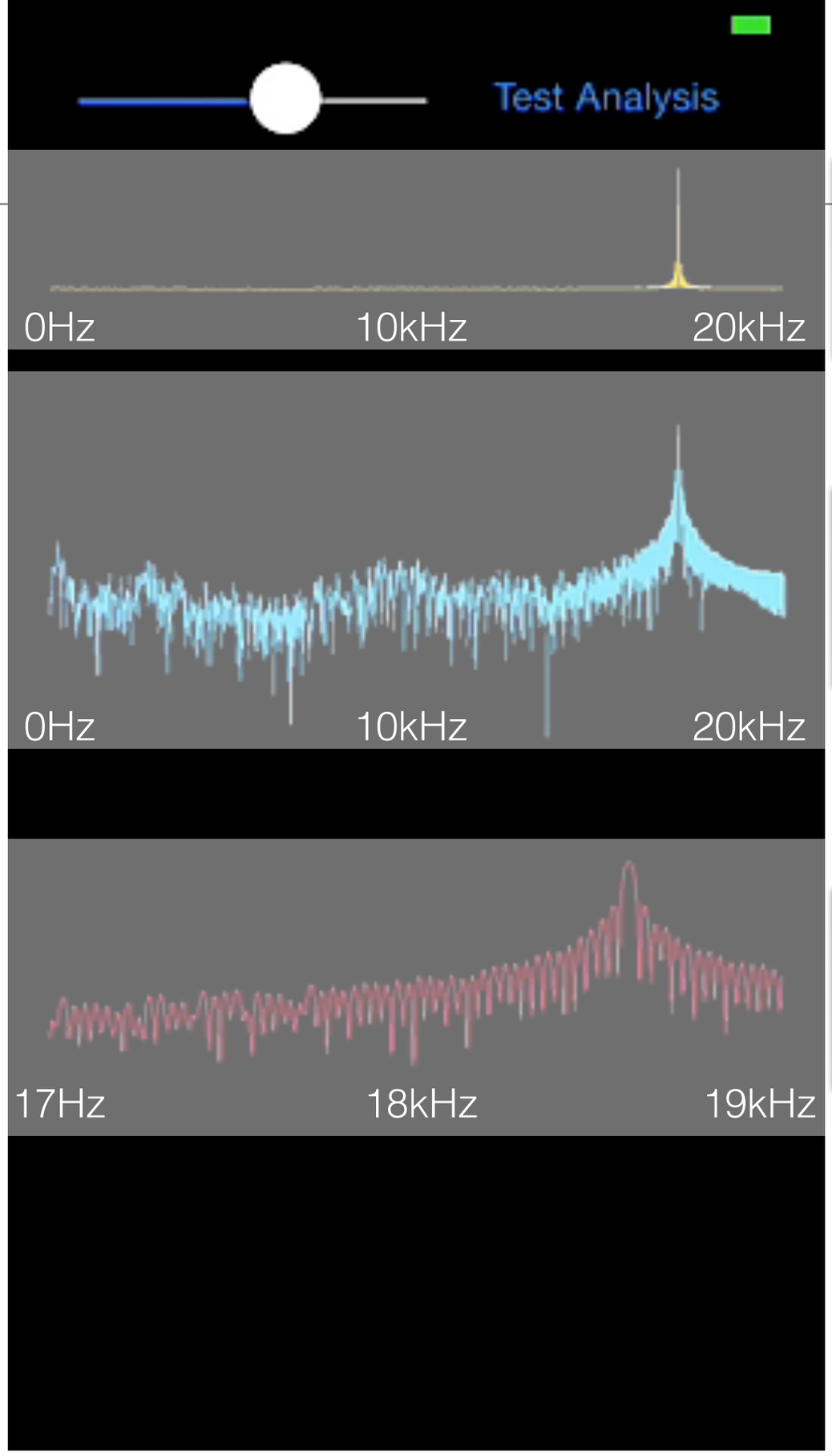
$$\Delta f = \frac{V_{object}}{c} f_0$$

change in frequency

velocity of object

speed of sound

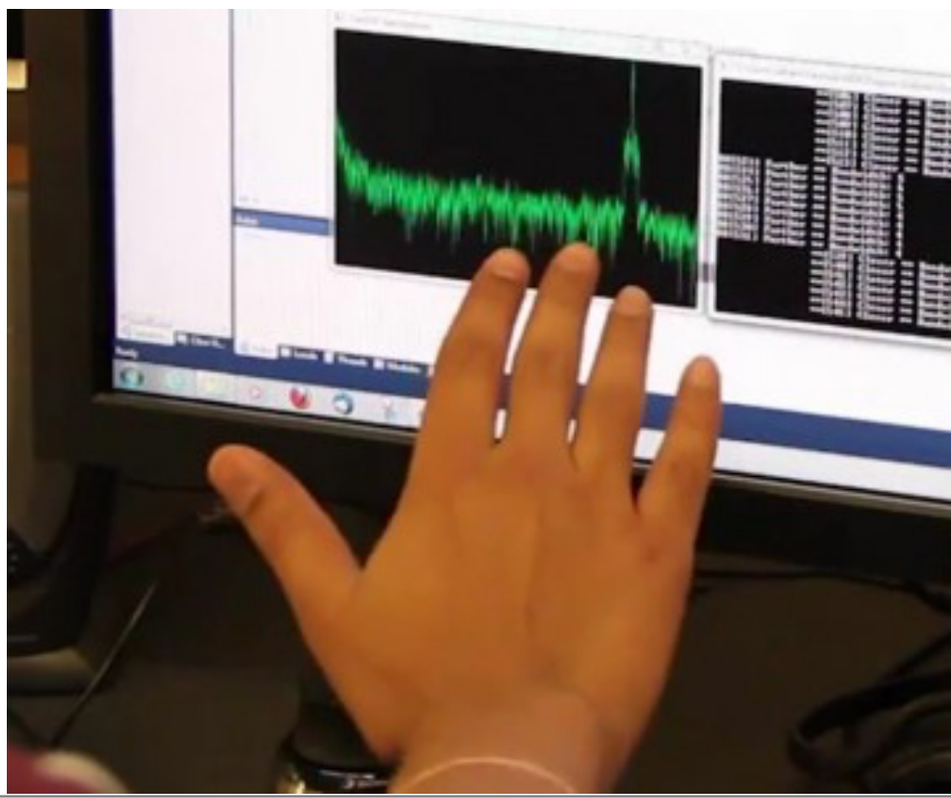
frequency of source



linear

db

db zoomed (freq axis)



A2 specifications

Module A: Create an iOS application using the example template that:

- Reads from the microphone
- Takes an FFT of the incoming audio stream
- Displays the frequency of the two loudest tones within 6Hz accuracy (+-3Hz)
  - Please have a way to "lock in" the last frequencies detected on the display
- Is able to distinguish tones at least 50Hz apart, lasting for 200ms or more
- **Exceptional**: recognize two tones played on a piano (down to one half step apart) and report them by letter (***i.e.***, A4, A#4). Must work at note A2 and above.

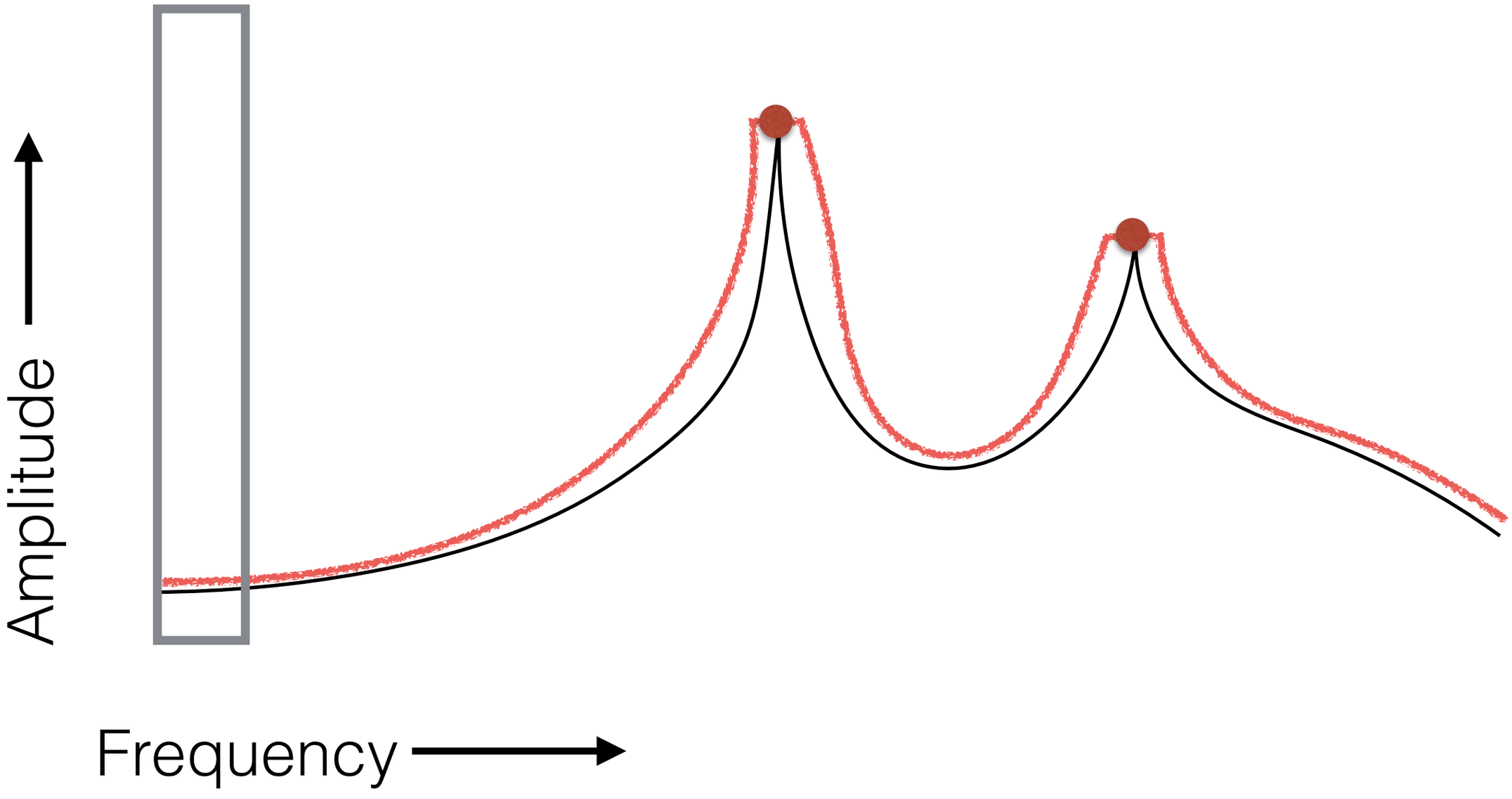
Note: this is harder than just identifying two perfect sine waves!!
The sound source must be external (***i.e.***, laptop, instrument, another phone, ***etc***.).

Module B: Create an iOS application using the example template that:

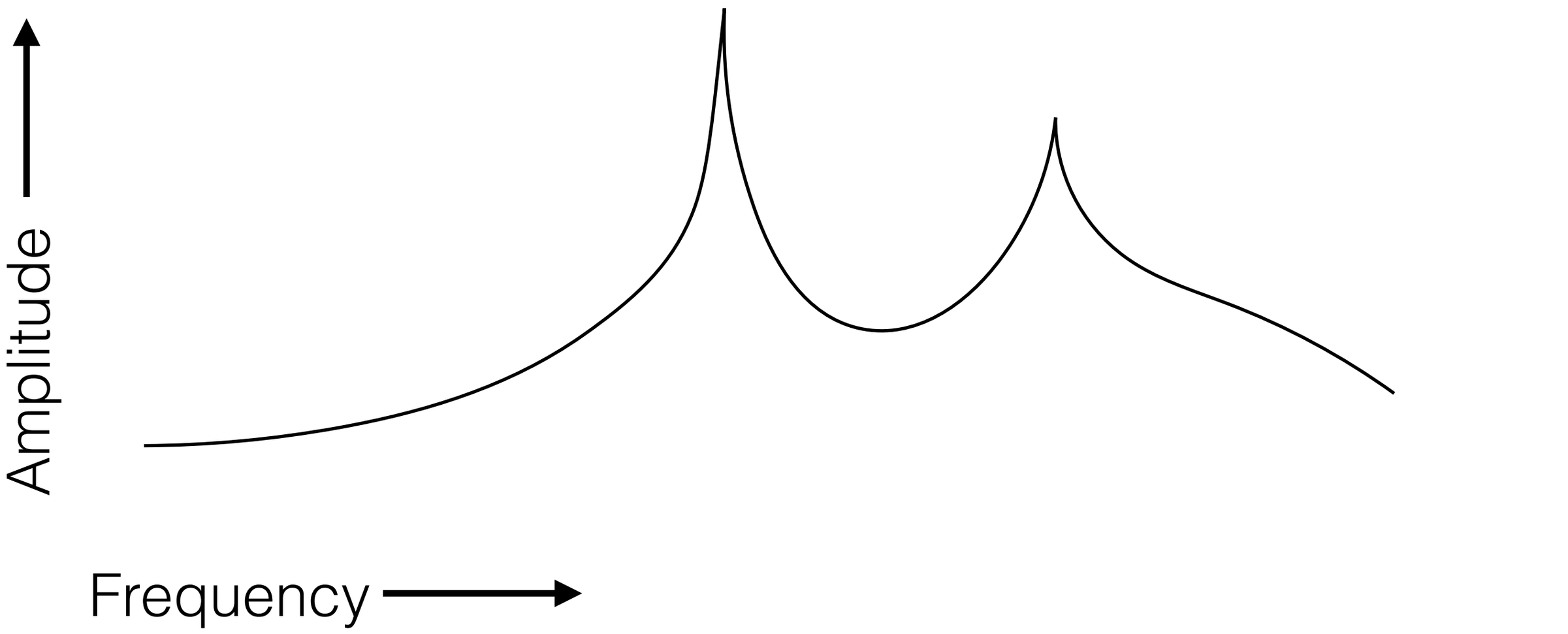
- Reads from the microphone
- Plays a settable (via a slider or setter control) inaudible tone to the speakers (15-20kHz)
- Displays the magnitude of the FFT of the microphone data in decibels
- Is able to distinguish when the user is {**not gesturing**, **gestures toward**, or **gesturing away**} from the microphone using Doppler shifts in the frequency

local peak finding

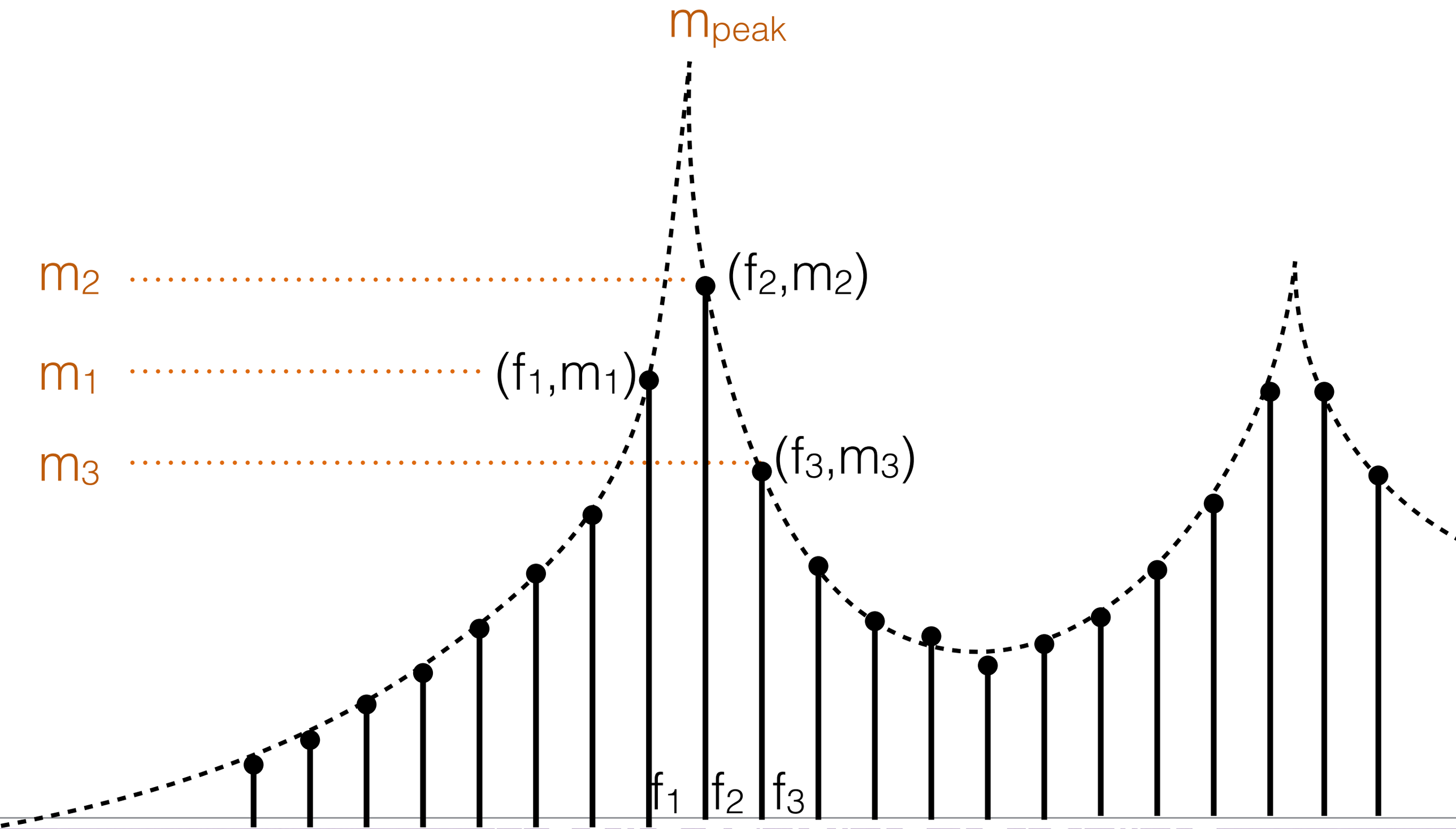
max in window



peak interpolation

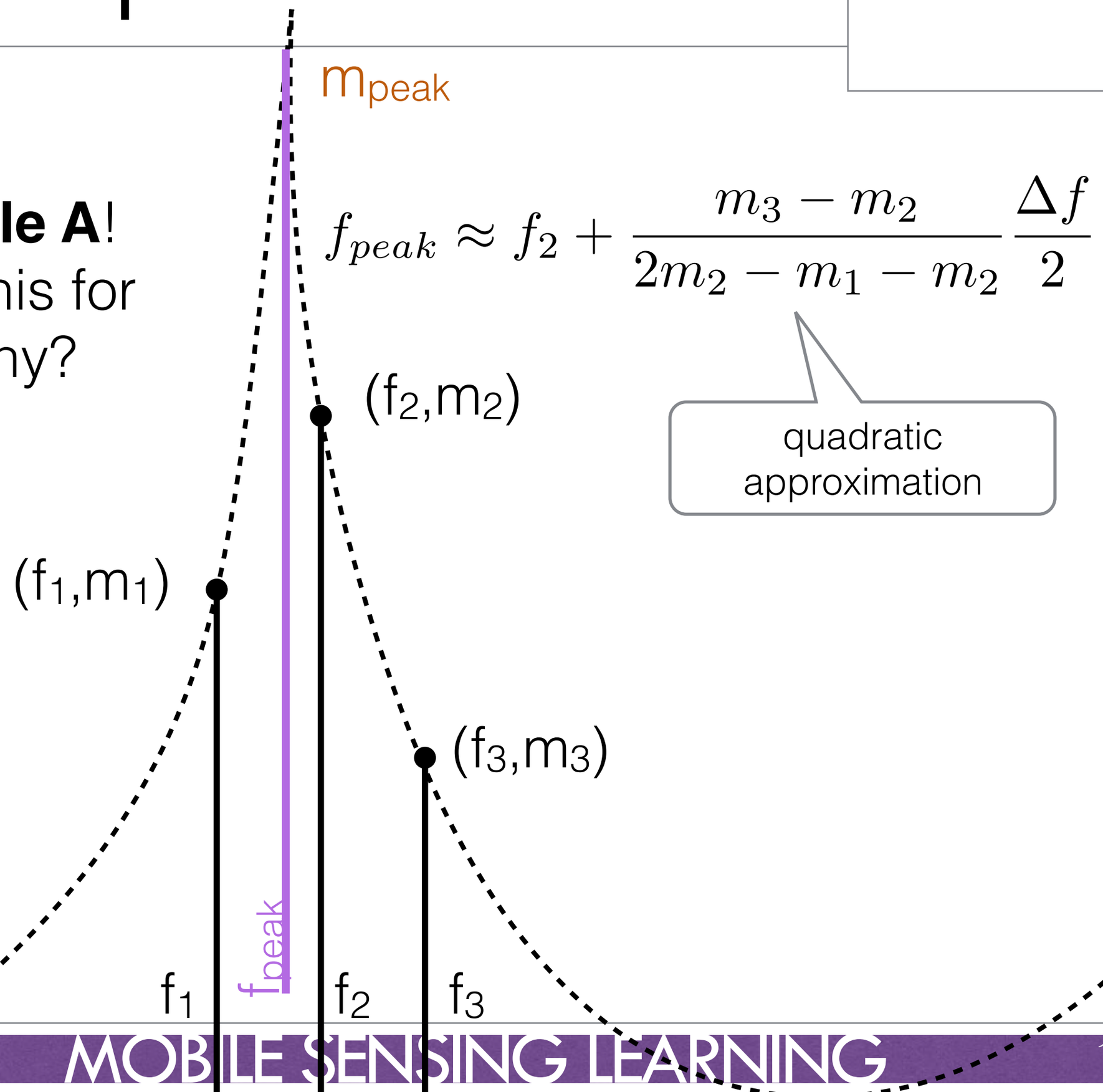


peak interpolation

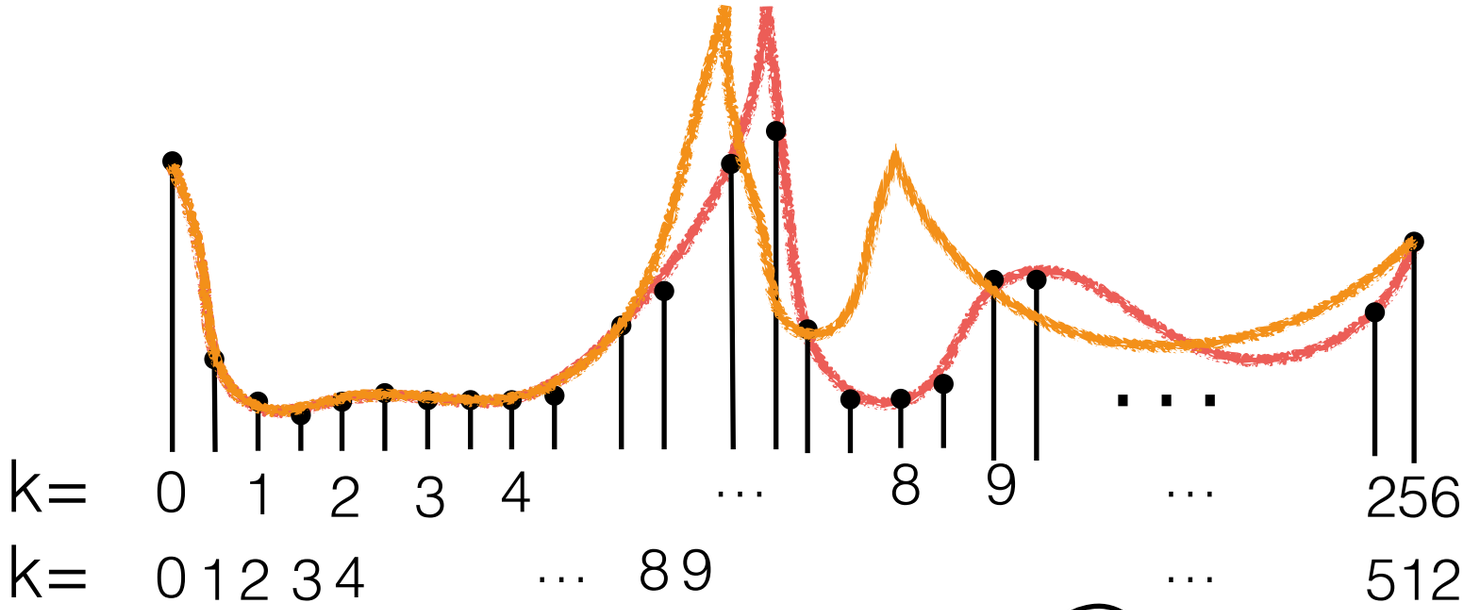


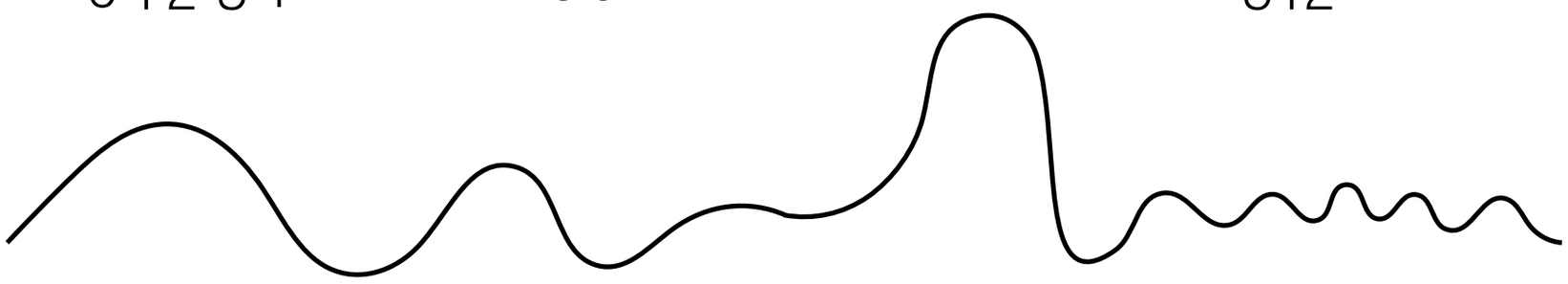
peak interpolation

Great for **module A**!
No need to do this for **module B**, Why?



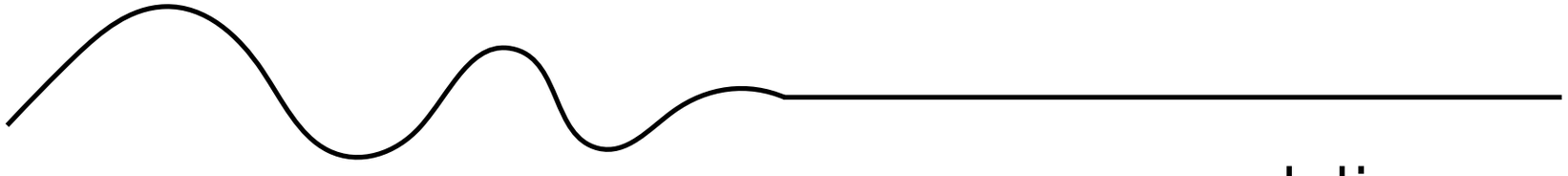
resolution for the FFT





256 points

next 256 points=512 total points

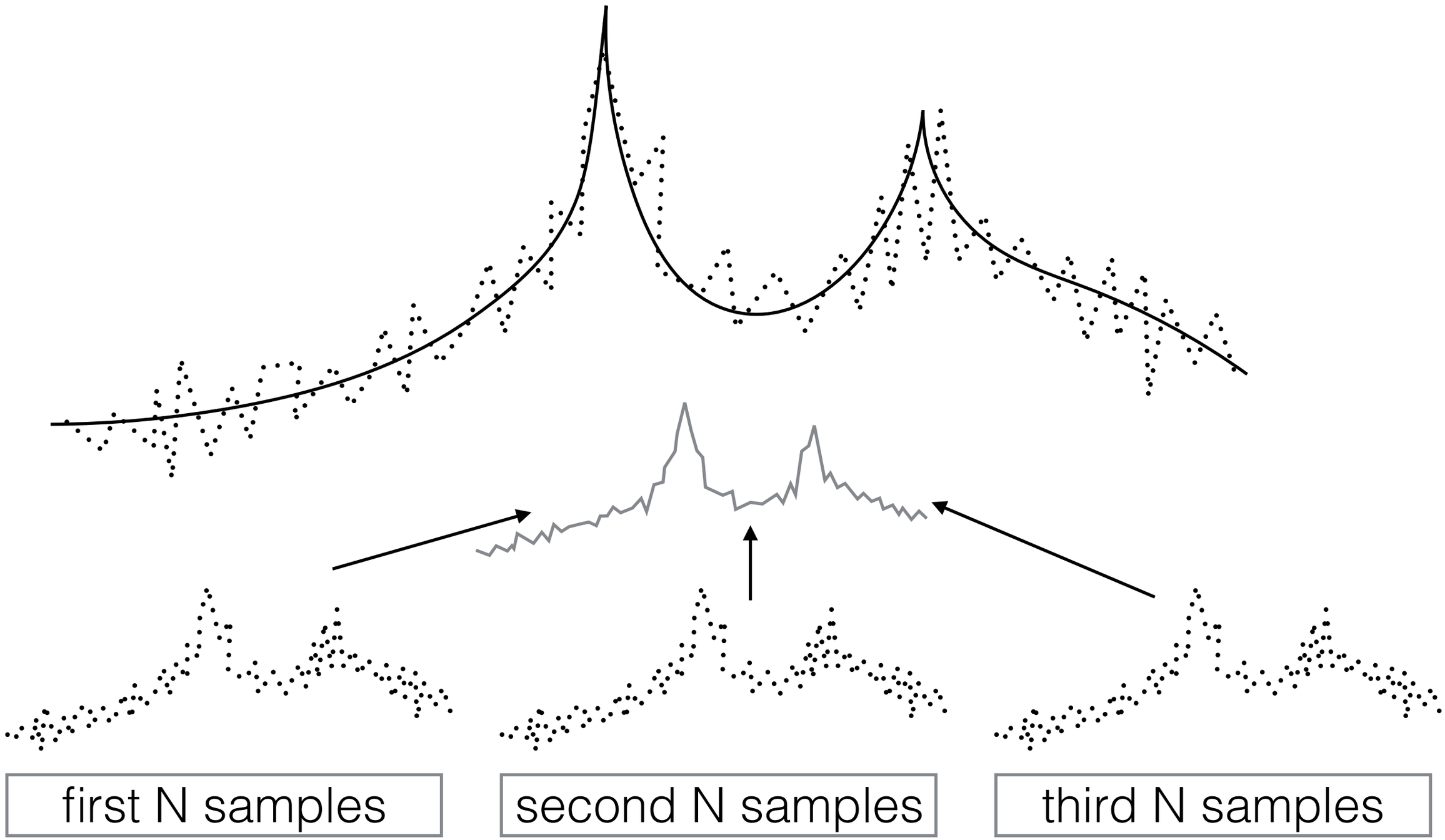


solution!

zero padding

noise in the FFT

- variance around actual magnitude unavoidable



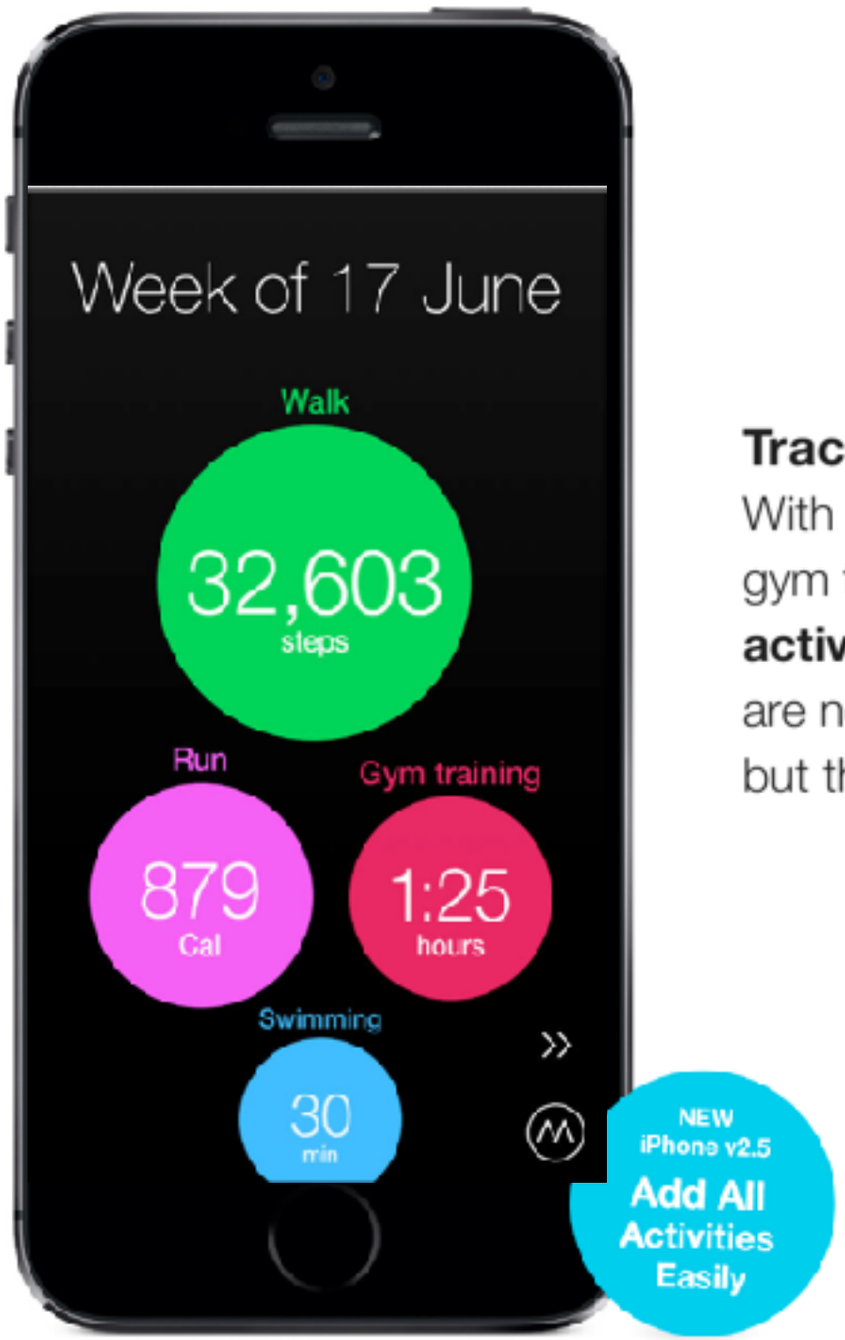
Questions on the FFT

- we are about to move to motion processing…
- so ask now!
- …or later…

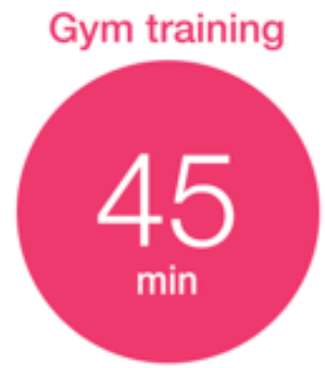
and now…

- no more microphone data!
- let's get data from the other sensors…
- core motion
  - the M# co-processor

a nice example of core motion



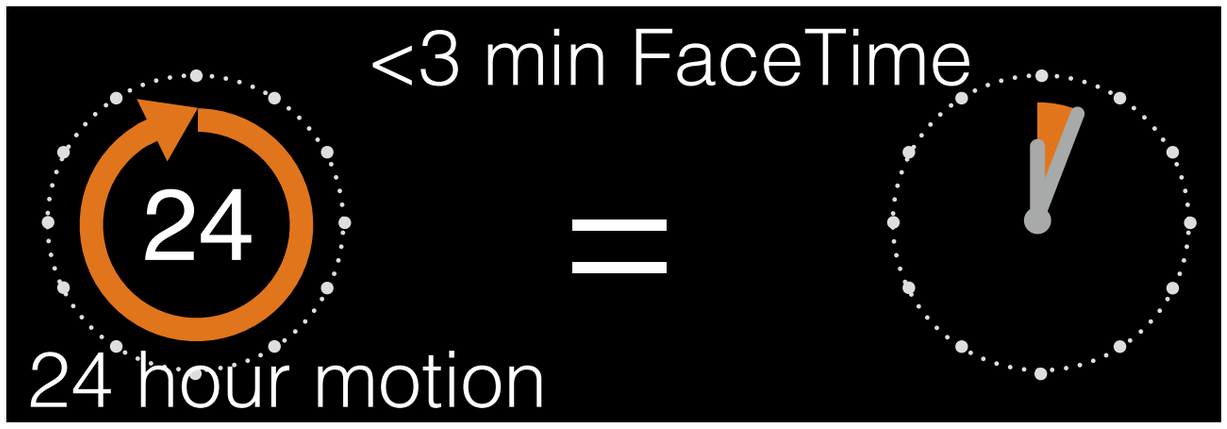
Track all activity*
With Moves 2.5 for iPhone, you can add gym training and **over 60 other activities** by duration. These activities are not (yet!) automatically recognized, but they are easy to add.



the M- coprocessor

- 150MHz+ processor that reads all motion data from all “motion” sensors on the phone
  - accelerometer
  - magnetometer (compass)
  - gyroscope
  - barometer! (M8 and above)

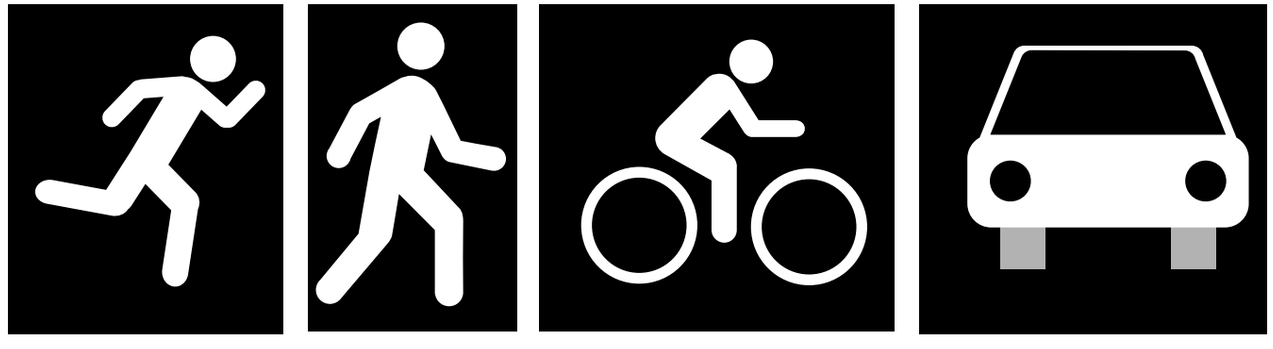
- mediates all access to data
  - battery life++
  - parallel processing++
  - overhead += 0, seriously



- sensor fusion for more accurate analysis, very cool

high level streams

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



activity from M-

- 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

More help: https://developer.apple.com/videos/wwdc/2014/
Navigate to: **Motion Tracking and Core Motion Framework**

subscribing to activity

- updates are notifications

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

// from 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

swift version

```
import CoreMotion


let activityManager = CMMotionActivityManager()
let customQueue = NSOperationQueue() // not the m    queue

 override func viewDidLoad() {
     super.viewDidLoad()

     if CMMotionActivityManager.isActivityAvailable(){
         self.activityManager.startActivityUpdatesToQueue(customQueue)
         { (activity:CMMotionActivity?)  -> Void in
             NSLog("%@",activity!.description)
         }
     }
 }

 override func viewWillDisappear(animated: Bool) {
     if CMMotionActivityManager.isActivityAvailable() {
         self.activityManager.stopActivityUpdates()
     }
     super.viewWillDisappear(animated)
 }
```

import framework

declare activity manager

device capable?

closure to handle updates
(*this one just prints description*)

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}
  - cycling {0, 1}
  - automotive {0,1}
  - unknown {0,1}
  - confidence {Low, Medium, High}

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

```
self.activityManager.startActivityUpdatesToQueue(customQueue)
                { (activity:CMMotionActivity?)  -> Void in
                // do something with the activity info!
            }
```

example update

inside handler

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


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

from notification

access fields easily

look at confidence

what's in an update?

Example Scenarios

Device scenarios	stationary	walking	running	automotive	cycling	unknown
On table	true	false	false	false	false	false
On runner's upper arm	false	false	true	false	false	false
In dash of idling vehicle	true	false	false	true	false	false
In dash of moving vehicle	false	false	false	true	false	false
Passenger checking email	false	false	false	false	false	false
Immediately after reboot	false	false	false	false	false	true
In zumba class	false	false	false	false	false	false

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);
    }

}];
```

setup date range

set dates

set queue

handle error!

handle output

- can you guess what the swift code looks like?

what's in an update?

Motion Activity

Walking

Performance is fairly insensitive to location

- Detection can be suppressed when device is in hand

Relatively low latency

Very accurate, on average

- Expect intermittent transitions into and out of walking state

what's in an update?

Motion Activity

Running

Completely insensitive to location

Shortest latency

Most accurate classification

what's in an update?

Motion Activity

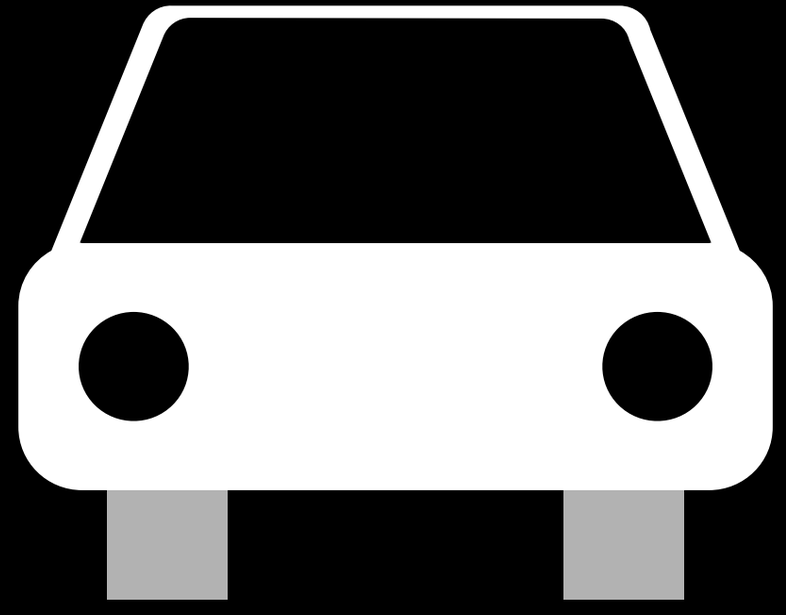
Automotive

Performance is sensitive to location

- Works best if device is mounted, or placed in dash or in cup holder

Variable latency

Relies on other information sources when available



what's in an update?

Motion Activity

Cycling

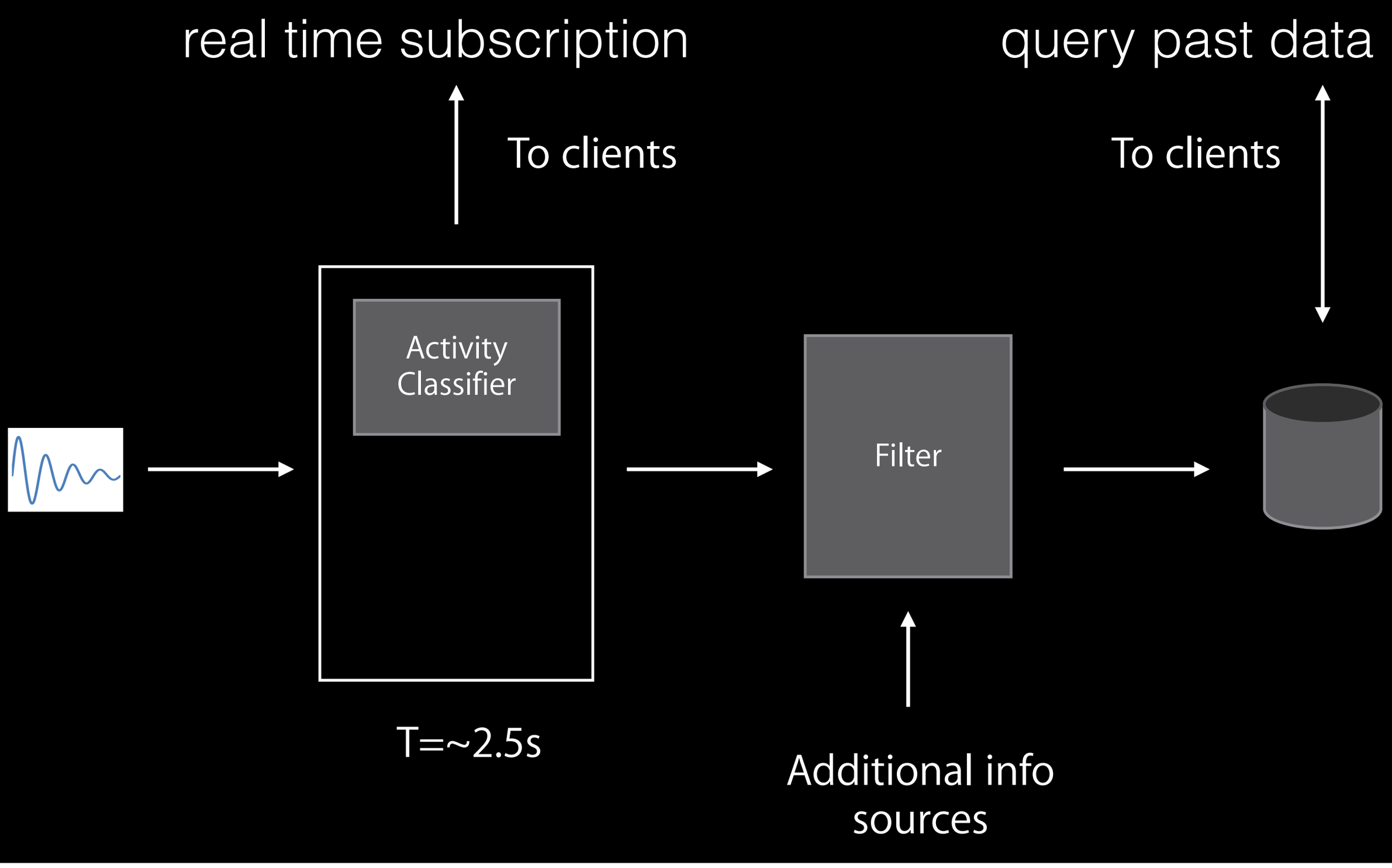
Performance is very sensitive to location

- Works best if device is worn on upper arm

Longest latency

- Best for retrospective use cases

Motion Processing Architecture



more than activity

- M- also tracks pedometer information during each activity
- like activity: setup as a **push** system (subscribe)
- pedometer: special handling from the M?
  - CMStepCounter is deprecated as of 2015!!!!!!! Do not use it…
  - instead, we will use the CMPedometer class

pedometer use

swift

```
let pedometer = CMPedometer()

if CMPedometer.isStepCountingAvailable(){
    pedometer.startPedometerUpdatesFromDate(NSDate())
        { (pedData: CMPedometerData?, error:NSError?) -> Void in
            NSLog("%@",pedData.description)
        }
}

if CMPedometer.isStepCountingAvailable(){
    self.pedometer.stopPedometerUpdates()
```

declare and init

available on this device?

closure handler for updates

unsubscribe

pedometer

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

pedometer use revisiting

declare and init

available on this device?

properties from step counter

```
let pedometer = CMPedometer()

if CMPedometer.isStepCountingAvailable(){
    pedometer.startPedometerUpdatesFromDate(NSDate())
        { (pedData: CMPedometerData?, error:NSError?) -> Void in
            NSLog("%@",pedData.description)
        }
}
```

```
if CMPedometer.isStepCountingAvailable(){
    self.pedometer.stopPedometerUpdates()
```

unsubscribe

```
CMPedometerData,<startDate 2016-09-06 17:13:54
+0000, endDate 2016-09-06 17:14:21 +0000,
steps 16, distance 14.1199999999534,
floorsAscended 0, floorsDescended 0 currentPace
0.328659236431121 8 currentCadence
3.295127630233765>
```

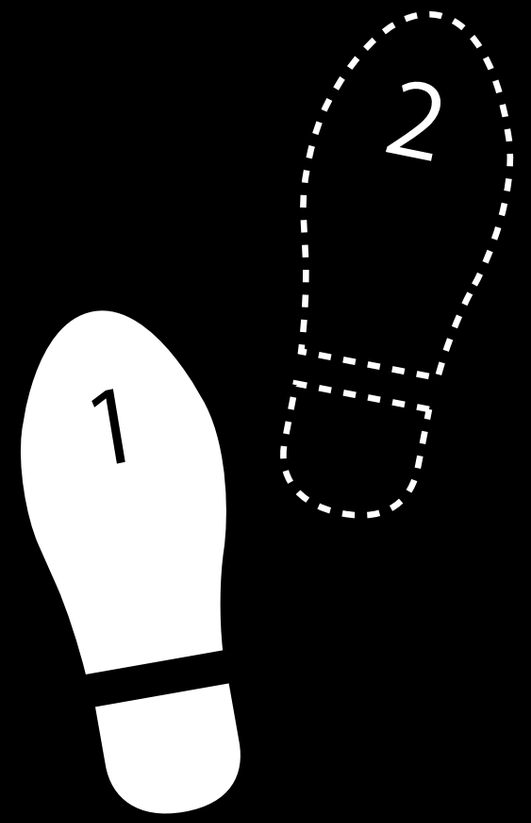
Pedometer

Step counting

Consistent performance across body locations

Extremely accurate

Robust to extraneous motions



Pedometer

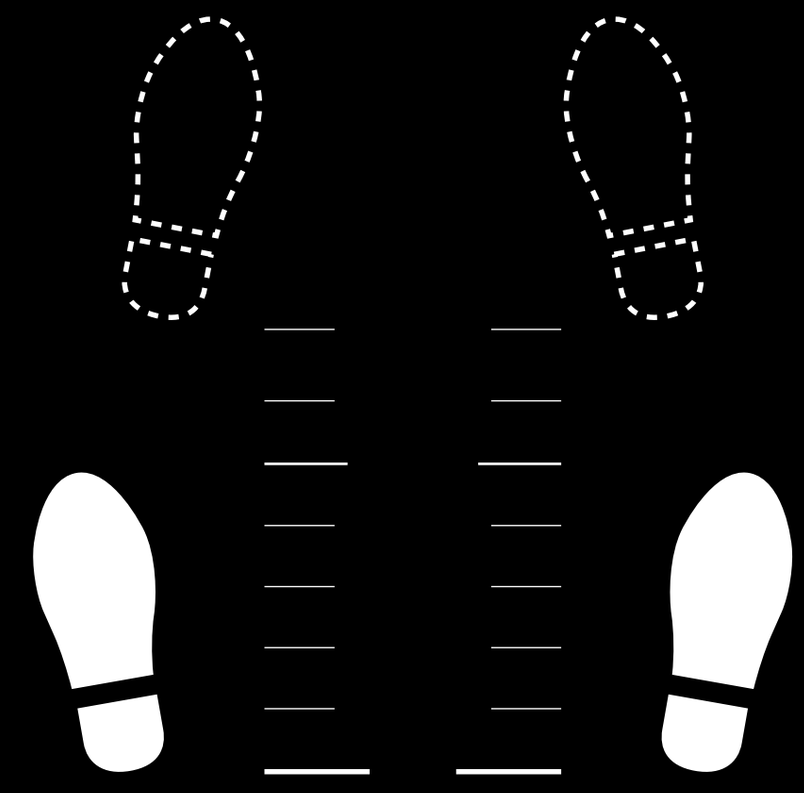
Stride estimation

Consistent performance across body locations

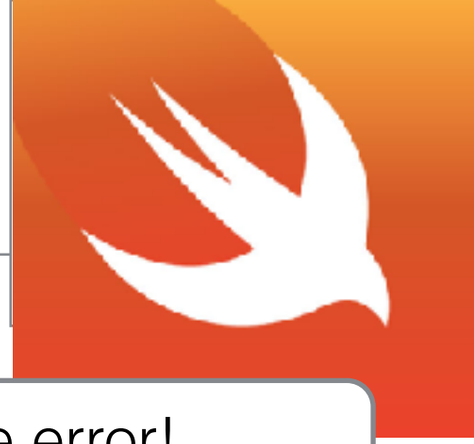
Consistent performance across pace

Extremely accurate

Adapts to the user over time



querying past steps

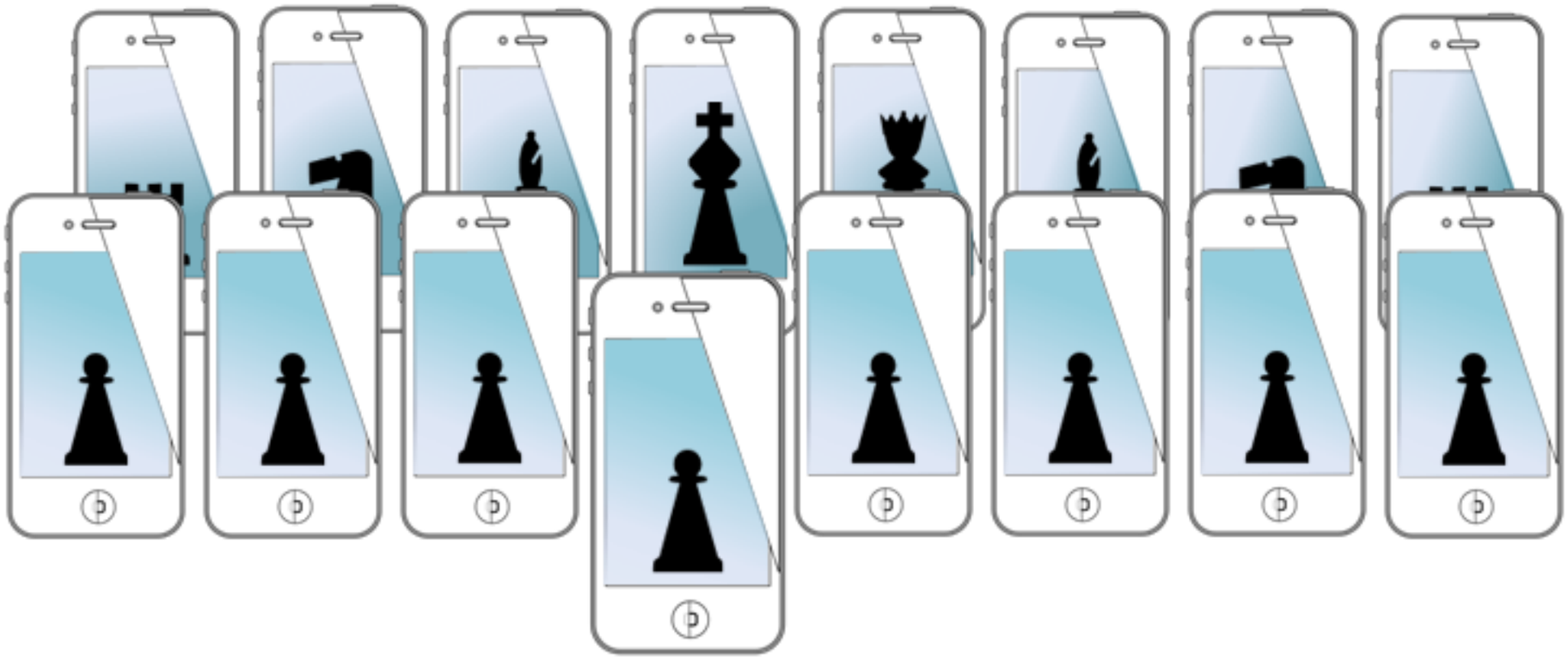


handle error!

```
let now = NSDate()
let from = now.dateByAddingTimeInterval(-60*60*24)

self.pedometer.queryPedometerDataFromDate(from, toDate: now)
{ (pedData: CMPedometerData?, error: NSError?) -> Void in

    let aggregated_string = "Steps: \(pedData.numberOfSteps) \n
            Distance \(pedData.distance) \n
            Floors: \(pedData.floorsAscended.integerValue)"

    dispatch_async(dispatch_get_main_queue()){
        self.activityLabel.text = aggregated_string
    }
}
```

access properties

MOBILE SENSING LEARNING



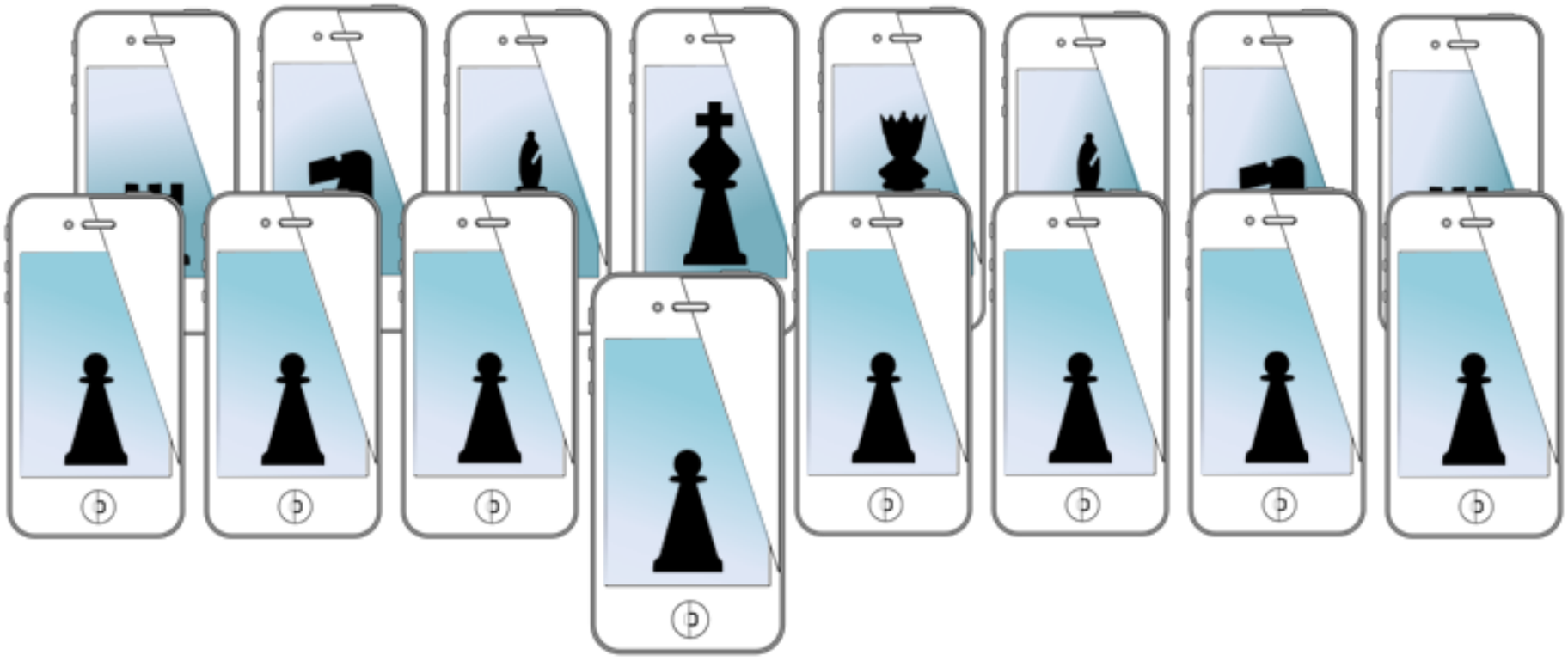
CSE5323 & 7323

Mobile Sensing and Learning

week 5, lecture a: doppler and activity monitoring

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

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week 5 lecture b: activity, pedometers, and motion sensing

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

course logistics

- A2 is due Friday
  - everyone okay?
- A3 is due a week from Friday
  - smaller set of deliverables

Module A

Create an iOS application that:

- Displays the number of steps a user has walked today and yesterday
- Displays a realtime count of the number of steps a users has taken today
- Displays the number of steps until the user reaches a (user settable) daily goal
- Displays the current activity of the user: {unknown, still, walking, running, cycling, driving}

Module B

Create an additional part of the app that, whenever the user meets their step goal for the previous day, allows the playing of a simple game (it can be **very** simple). The game must:

- Uses {acceleration, gyro, magnetometer, OR fused motion} to control some part of the physics of a SpriteKit (or SceneKit) game
- Uses two or more SpriteKit (or SceneKit) objects with dynamic physics
- An idea for exceptional work: use the steps of a user as some type of "currency" in the game to incentivize movement during the day

agenda

- core motion (continued)
  - M- co-processor
- demo
- accelerometers, gyros, and magnetometers
- demo
- SpriteKit
- demo

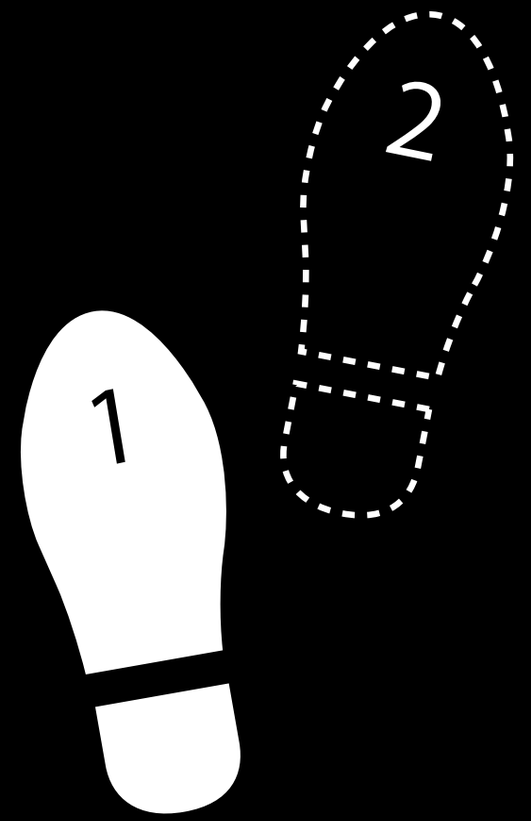
Pedometer

Step counting

Consistent performance across body locations

Extremely accurate

Robust to extraneous motions



Pedometer

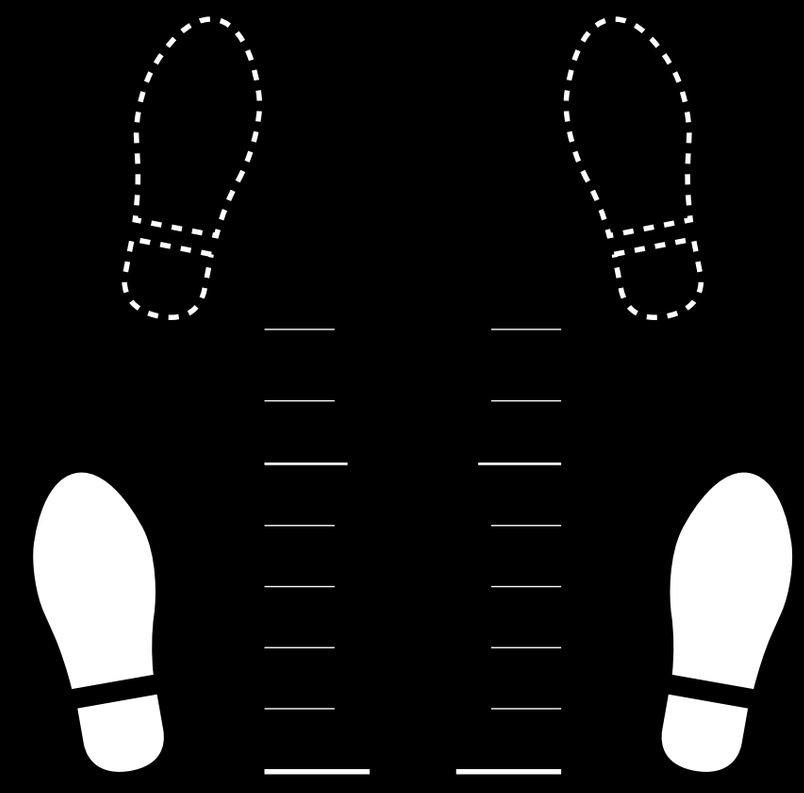
Stride estimation

Consistent performance across body locations

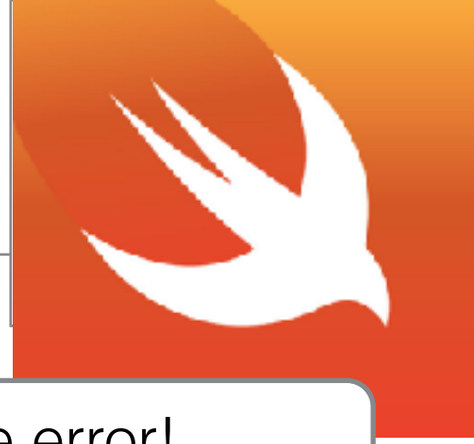
Consistent performance across pace

Extremely accurate

Adapts to the user over time



querying past steps



handle error!

```
let now = NSDate()
let from = now.dateByAddingTimeInterval(-60*60*24)

self.pedometer.queryPedometerDataFromDate(from, toDate: now)
{ (pedData: CMPedometerData?, error: NSError?) -> Void in

    let aggregated_string = "Steps: \(pedData.numberOfSteps) \n
            Distance \(pedData.distance) \n
            Floors: \(pedData.floorsAscended.integerValue)"

    dispatch_async(dispatch_get_main_queue()){
        self.activityLabel.text = aggregated_string
    }
}
```

access properties

storing persistent defaults

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

```
// standardUserDefaults variable
 let defaults = NSUserDefaults.standardUserDefaults()

  // saving
  defaults.setInteger(252, forKey:@“primitiveInteger")
  defaults.setDouble(3.14, forKey:@“primitiveDouble")
  defaults.setFloat
  defaults.setBool
  defaults.setURL

  // saving an object
  defaults.setObject("Coding Explorer", forKey: “userNameKey")

  if let name = defaults.stringForKey("userNameKey") {
      print(name)
  }
  boolForKey         ->  Bool
  integerForKey      ->  Int
  dataForKey         ->  NSData?
  objectForKey       ->  AnyObject?
  arrayForKey        ->  [AnyObject]?
  stringArrayForKey->  [String]?
  dictionaryForKey  ->  {String:AnyObject}?
```

import defaults

primitives

objects

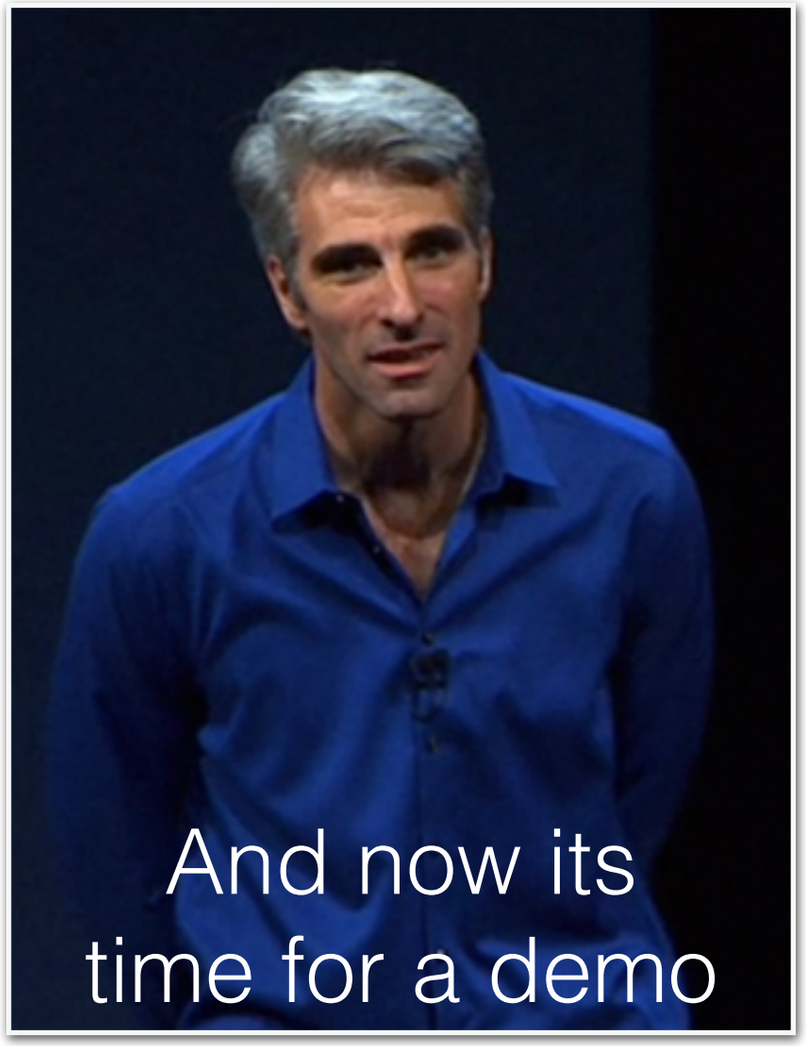
access saved objects

user defaults

key value behavior for setting and getting!

```
_dailyStepsGoal = @(50);

NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

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


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

M- pedometer/activity demo



M- “raw” motion data

Barometer

The barometer senses air pressure to determine your relative elevation. So as you move, you can keep track of the elevation you've gained. It can even measure stairs climbed or hills conquered.

Accelerometer

The accelerometer can measure your distance for walking and running. And by using GPS to calibrate for your running stride, the sensor more accurately captures your movement.

Gyroscope

In addition to knowing whether you're on the move or stationary, M8 works with the gyroscope to detect when you're driving. It also kicks into action when you're taking panoramic photos or playing games that react to your movement.

M- “raw” motion data

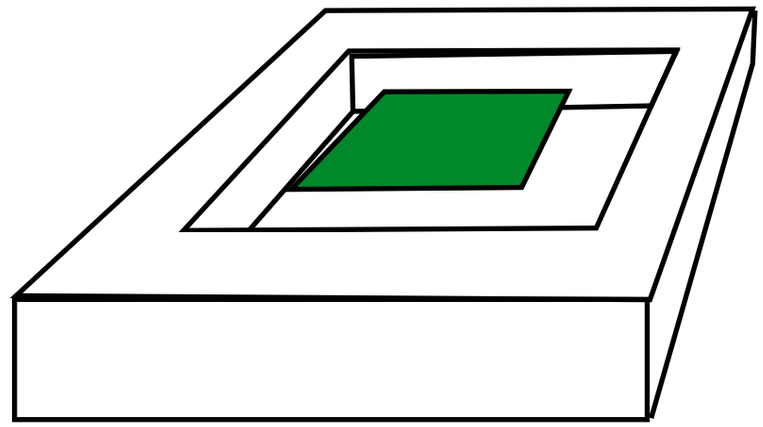
- M- mediates access to data
- much lower battery consumption

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%
iPhone 6, 6S	~2%	1%
iPhone 7	~?%	?%

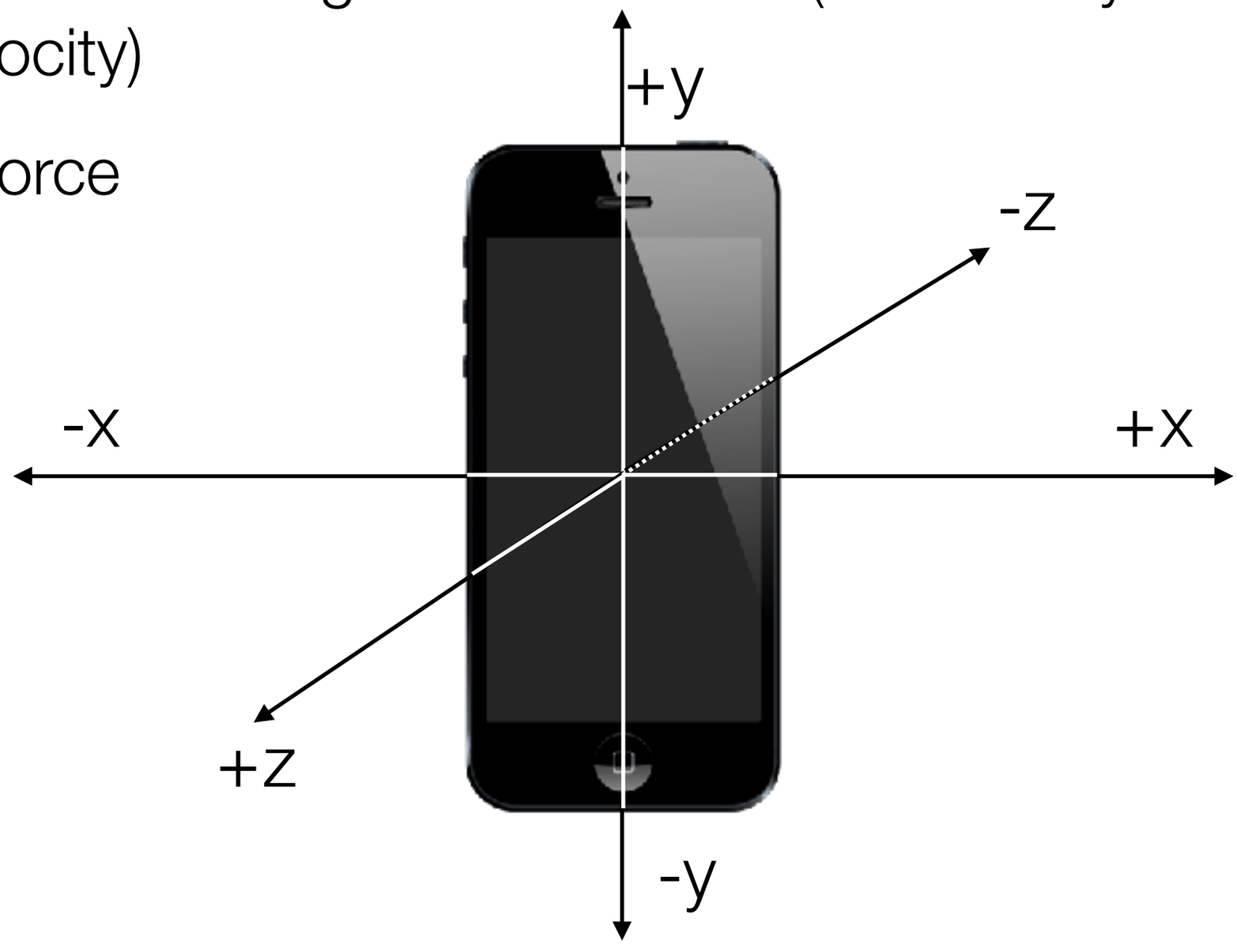
accelerometers

- 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



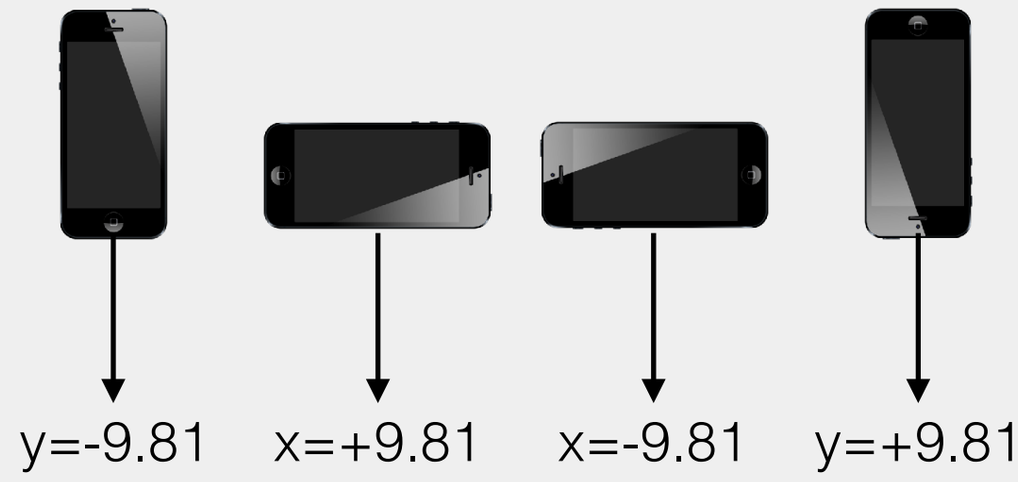
accessing the accelerometer

- 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

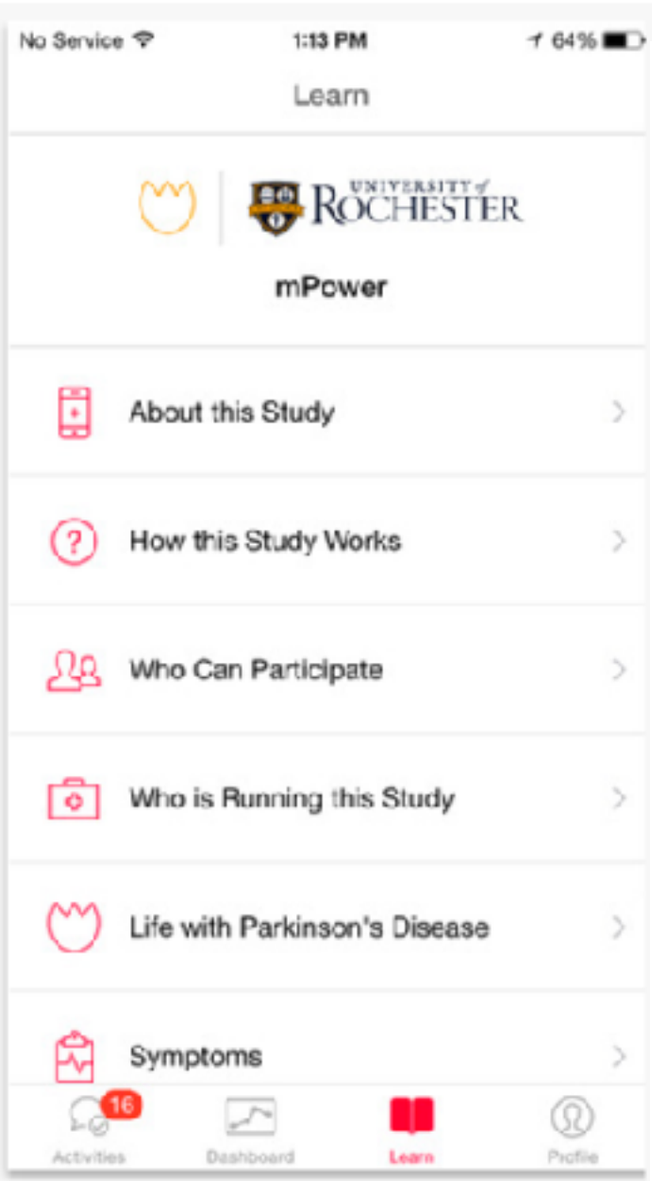
```
CMDeviceMotion *deviceMotion

deviceMotion.gravity
deviceMotion.userAcceleration

CMAcceleration gravity, CMAcceleration userAcceleration

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

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

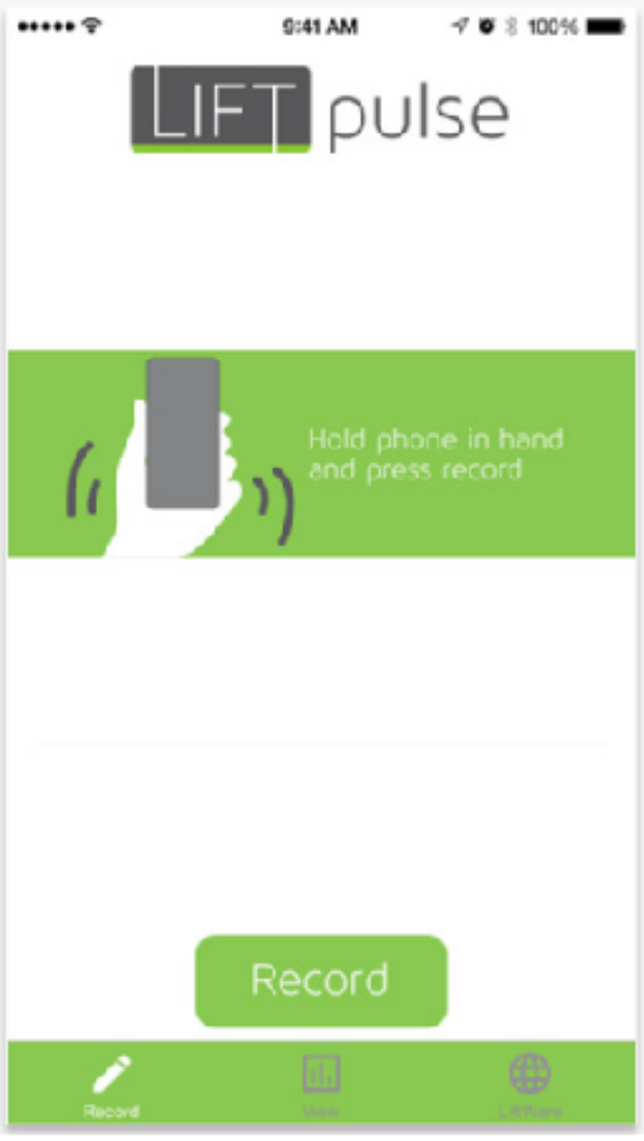
user movement

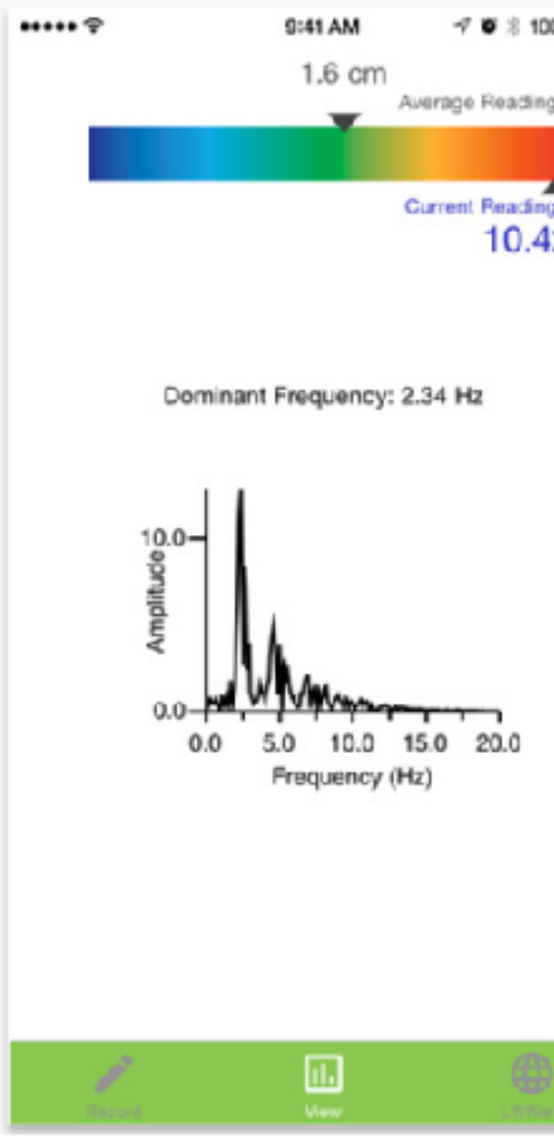
access through a different field!



a cool example







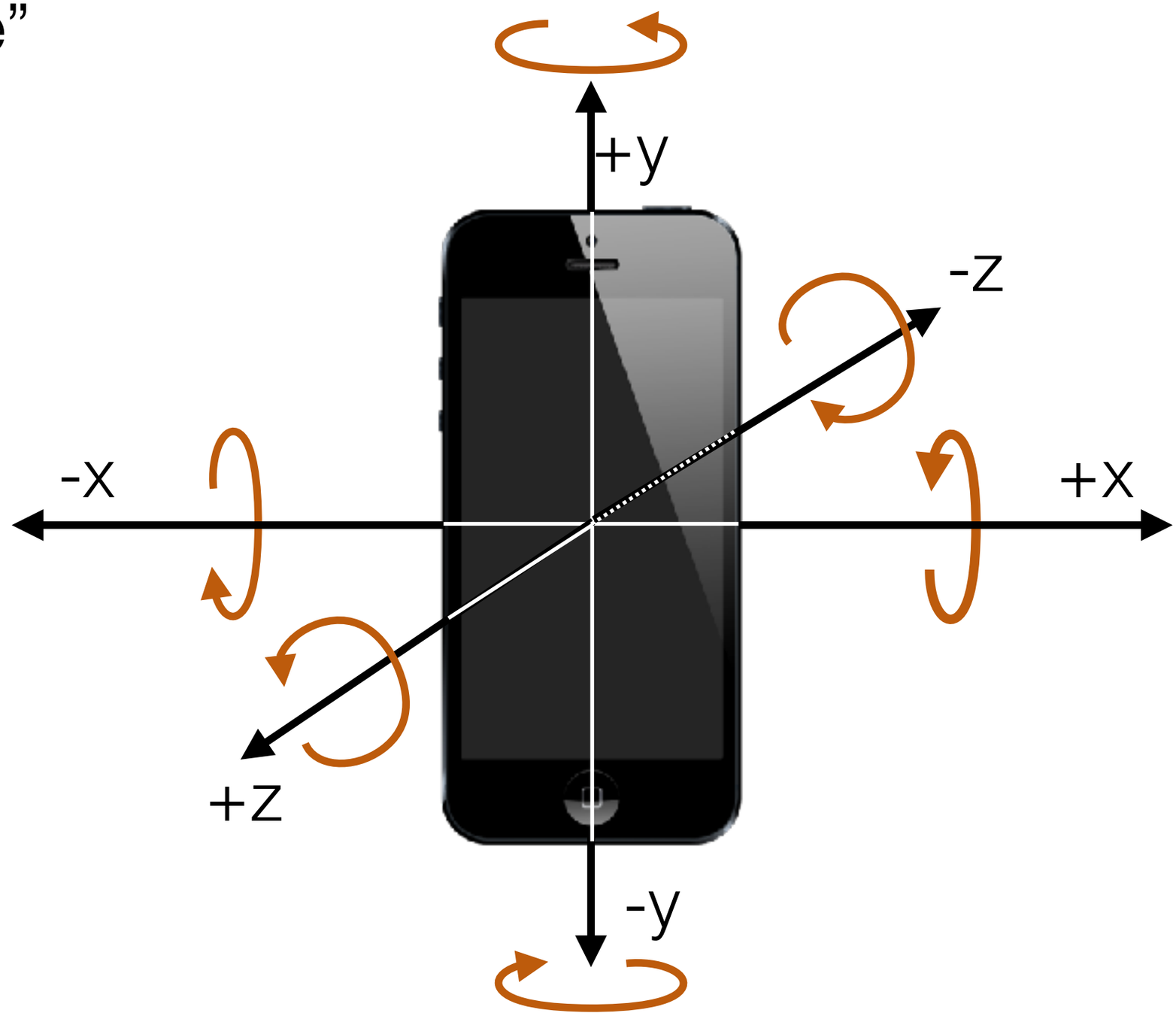
gyroscope

- 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”



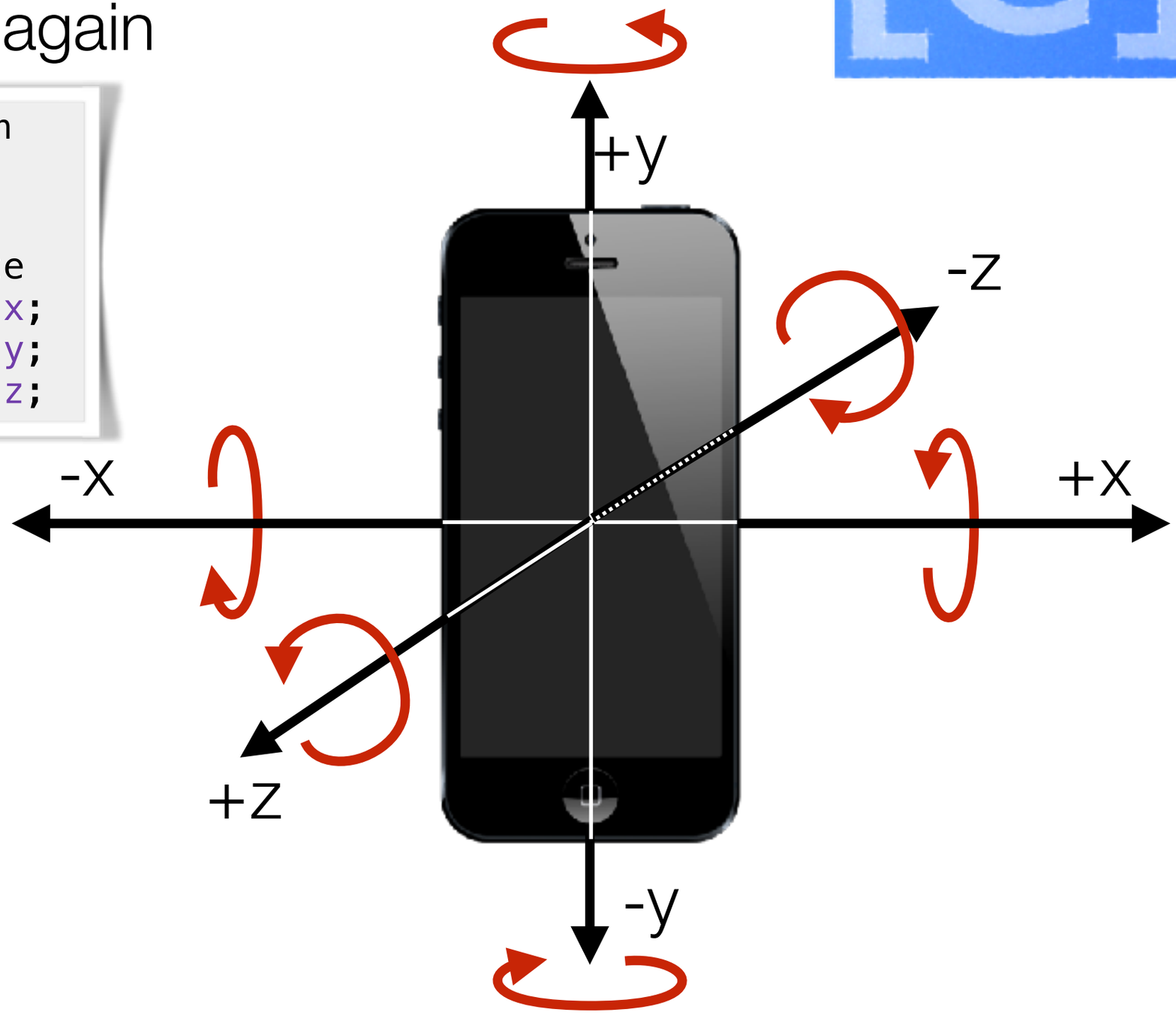
accessing the gyro

- use device motion again

```
CMDeviceMotion *deviceMotion

deviceMotion.rotationRate

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

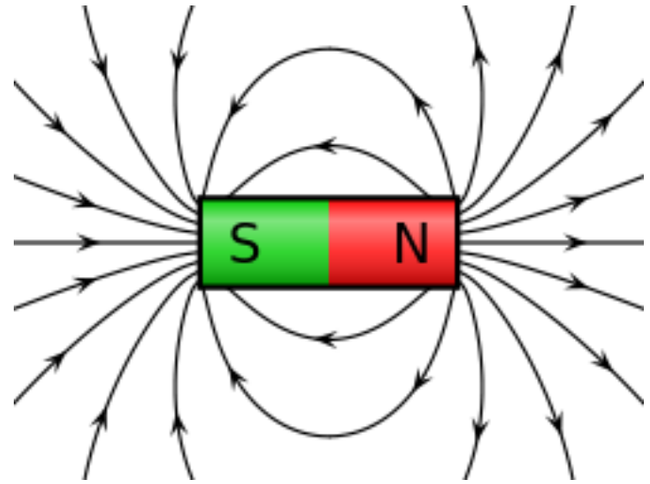
measures **rate** of motion

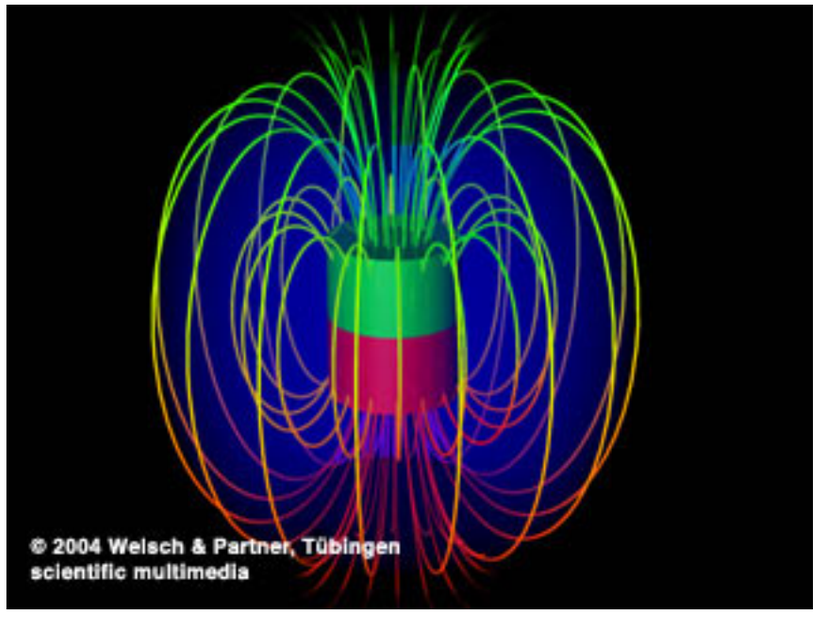
in this example, saves it to array



magnetometers

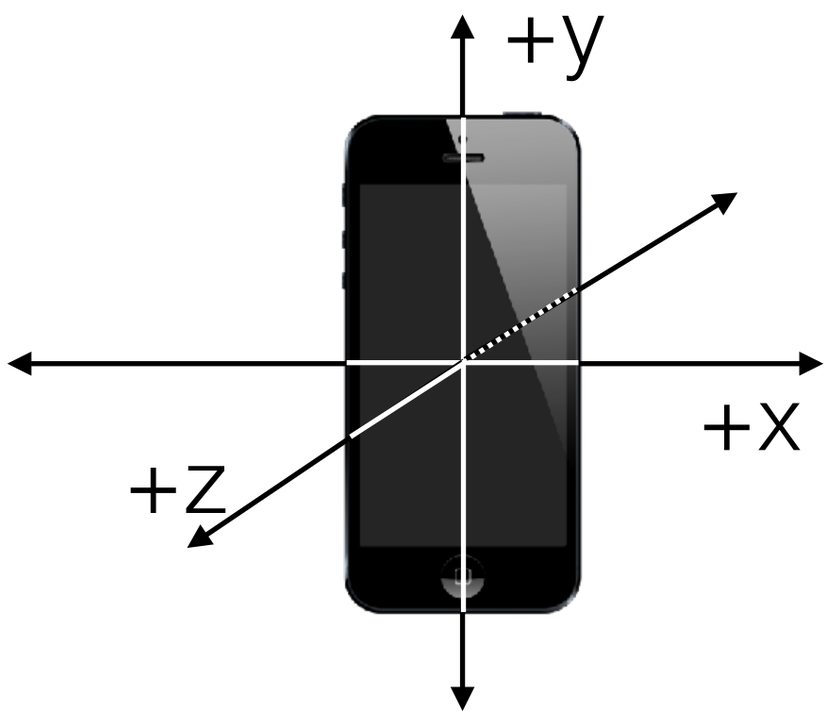
- measure magnetic fields
- magnets are measured in tesla (T)
  - **how**: essentially, there is a tight coupling between electricity flow and magnetic fields
- earth's magnetic field varies, but is around 50 uT
- iPhone can measure up to 1T with a resolution of about 8uT
- magnetic fields have direction!

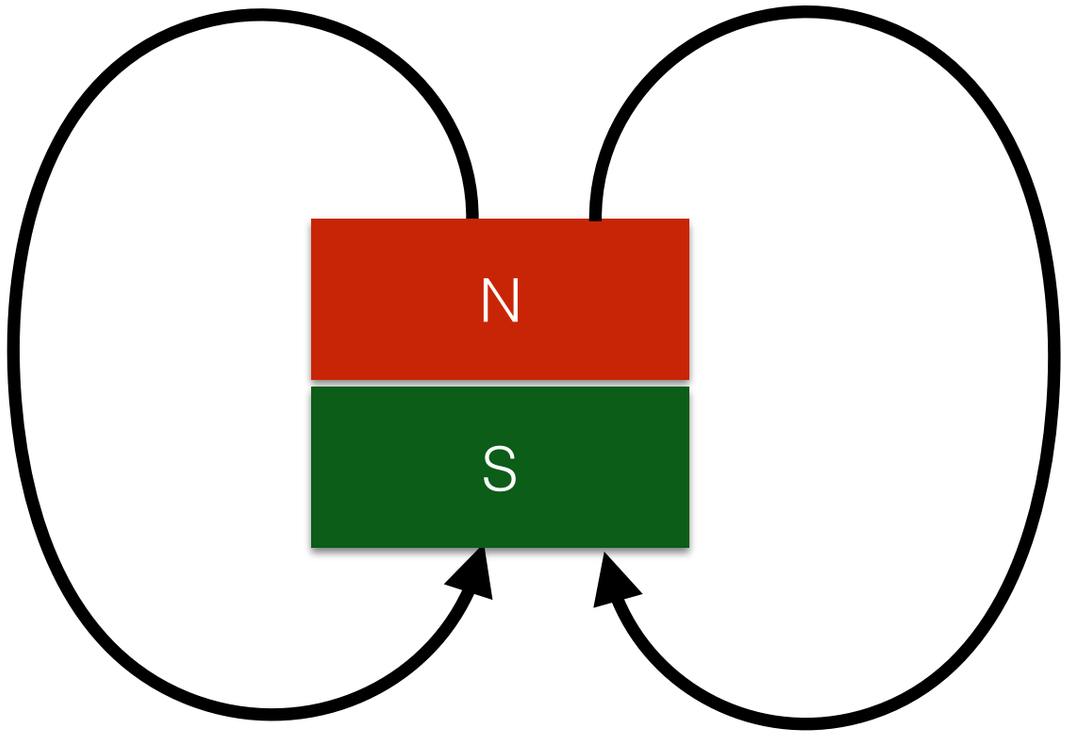




magnetic fields

- measure magnetic field along axis, towards “south”



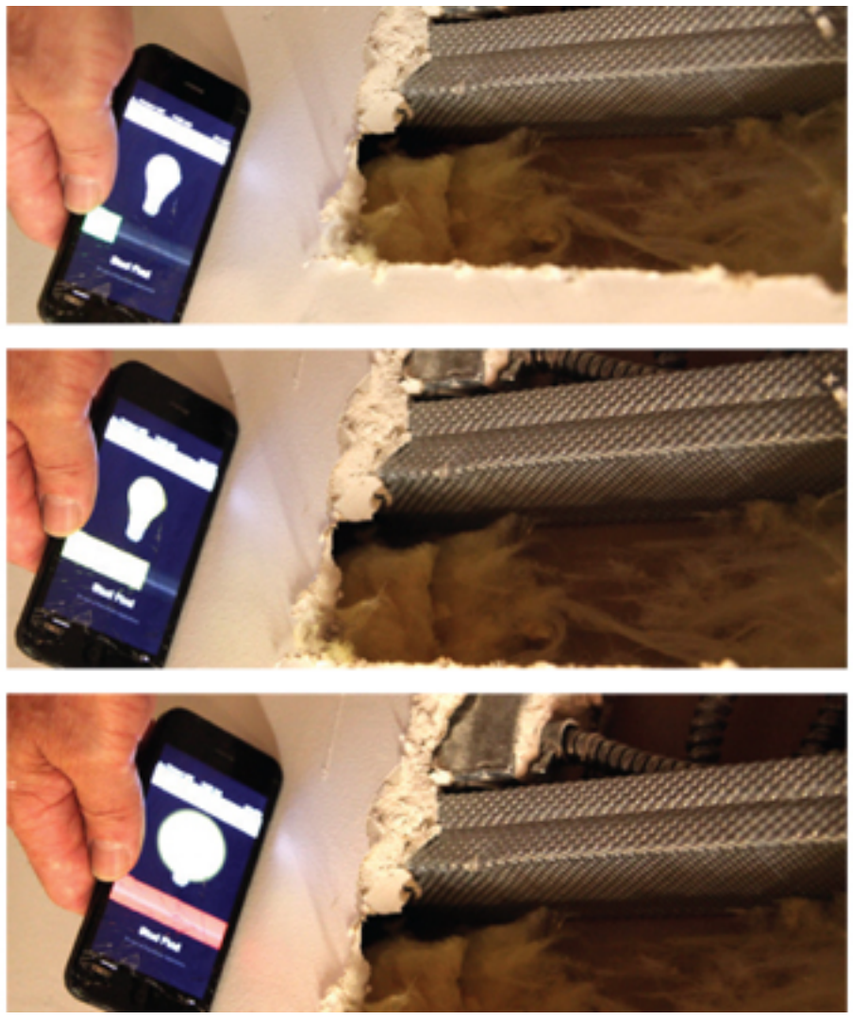


but iPhone has magnetic bias

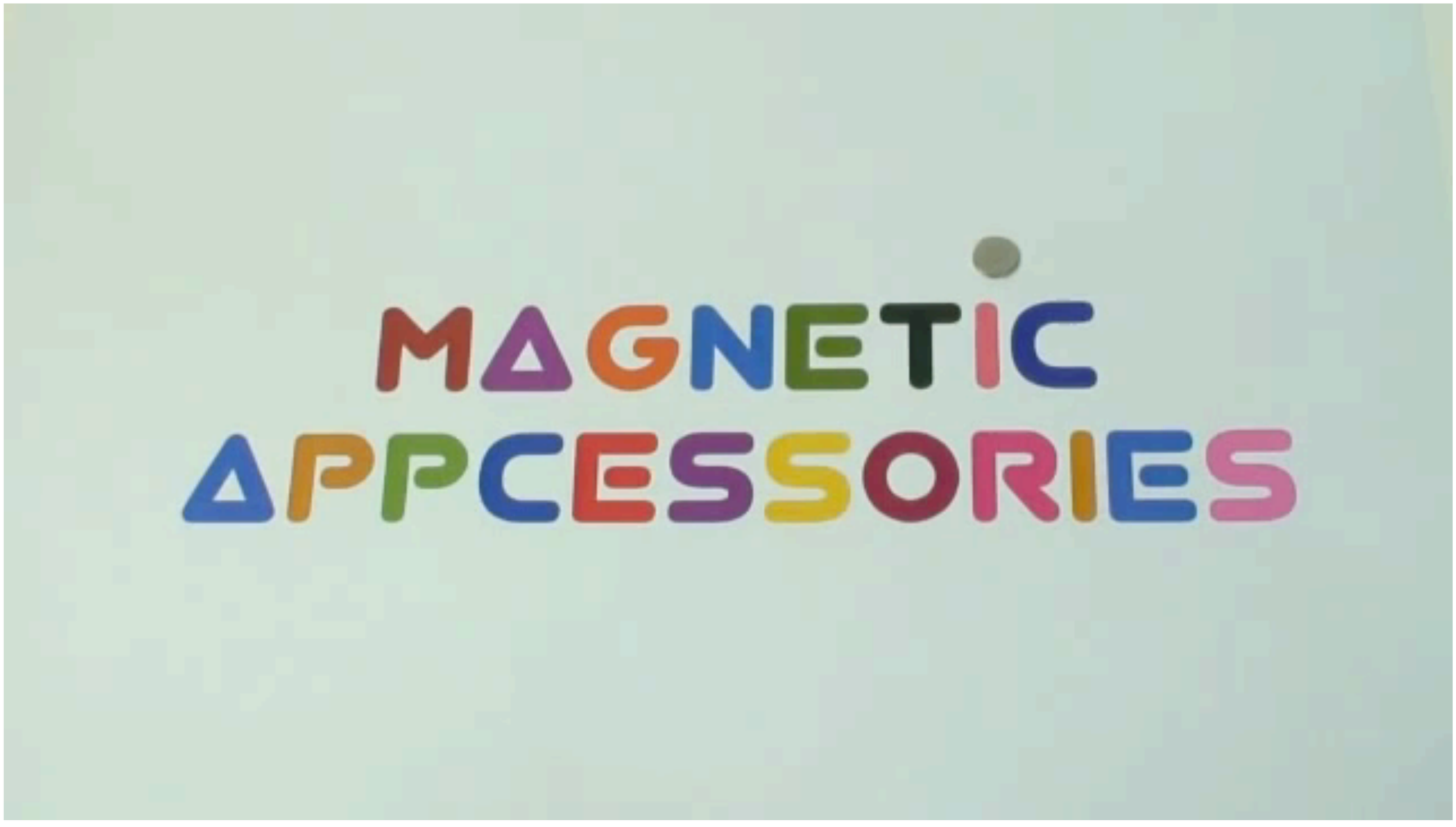
- the phone uses electricity and therefore is a magnet
  - good thing Apple subtracts that out for us!

```
CMDeviceMotion *deviceMotion


deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

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

magneticField.accuracy


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

a cool example

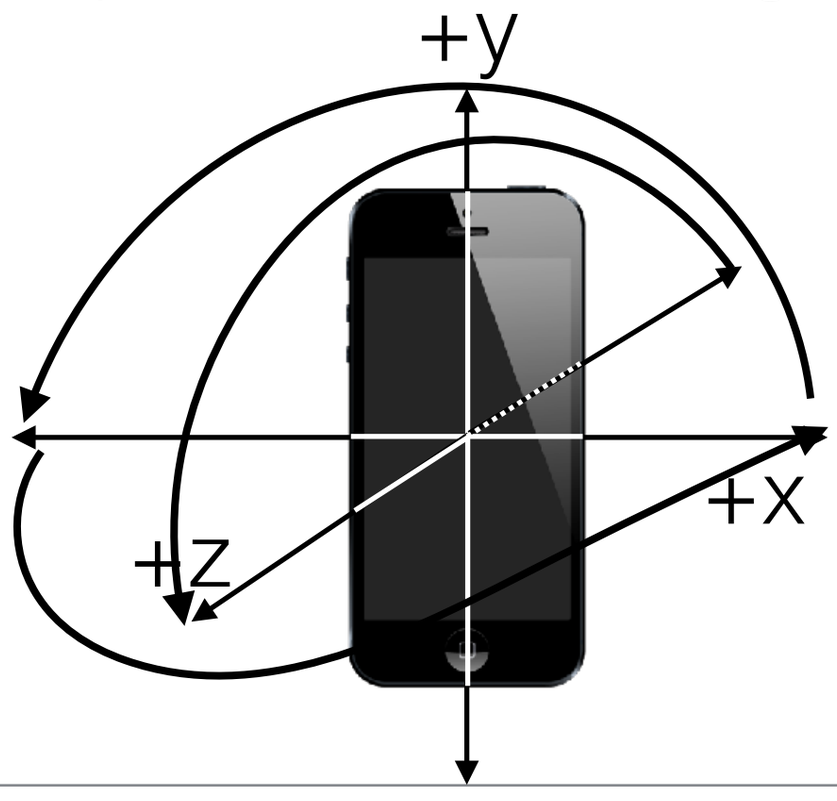


a cool example



attitude

- attitude is roll, pitch, and yaw (position)
- 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,


        }];
    }
```

declare

instantiate

if device is capable

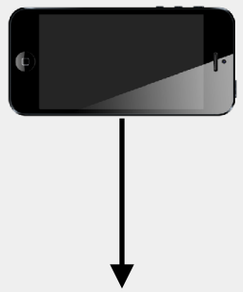
queue to run on

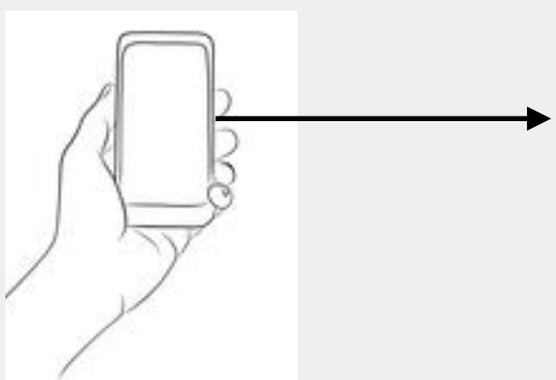
how often to push updates

the data

summary

```
CMDeviceMotion *deviceMotion

deviceMotion.gravity
deviceMotion.userAcceleration

CMAcceleration gravity,
CMAcceleration userAcceleration

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

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

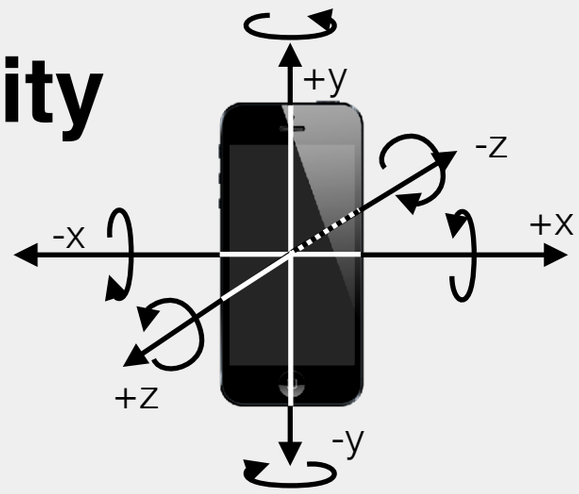




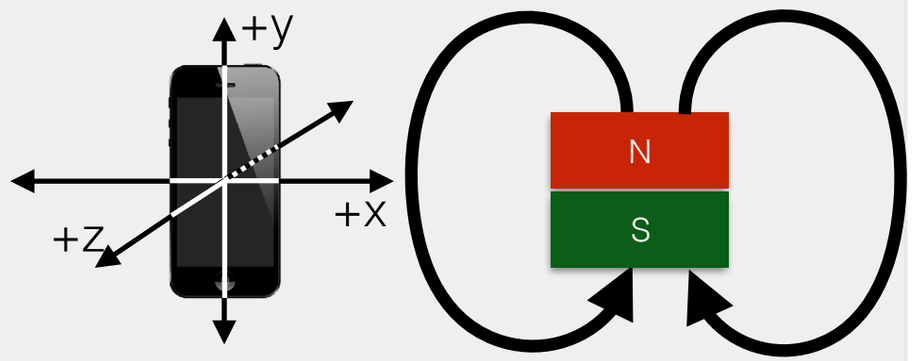
acceleration

rotation velocity

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

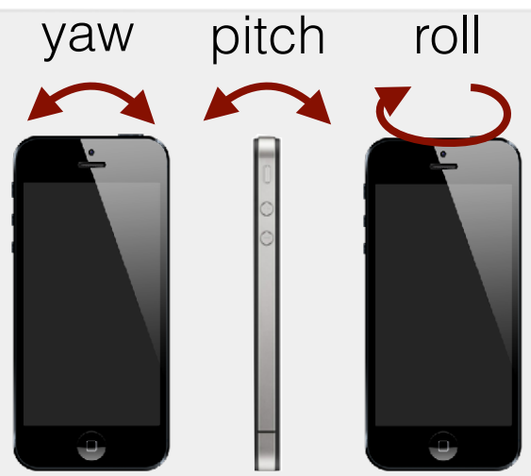



```
deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

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

magneticField.accuracy
```



magnetic field

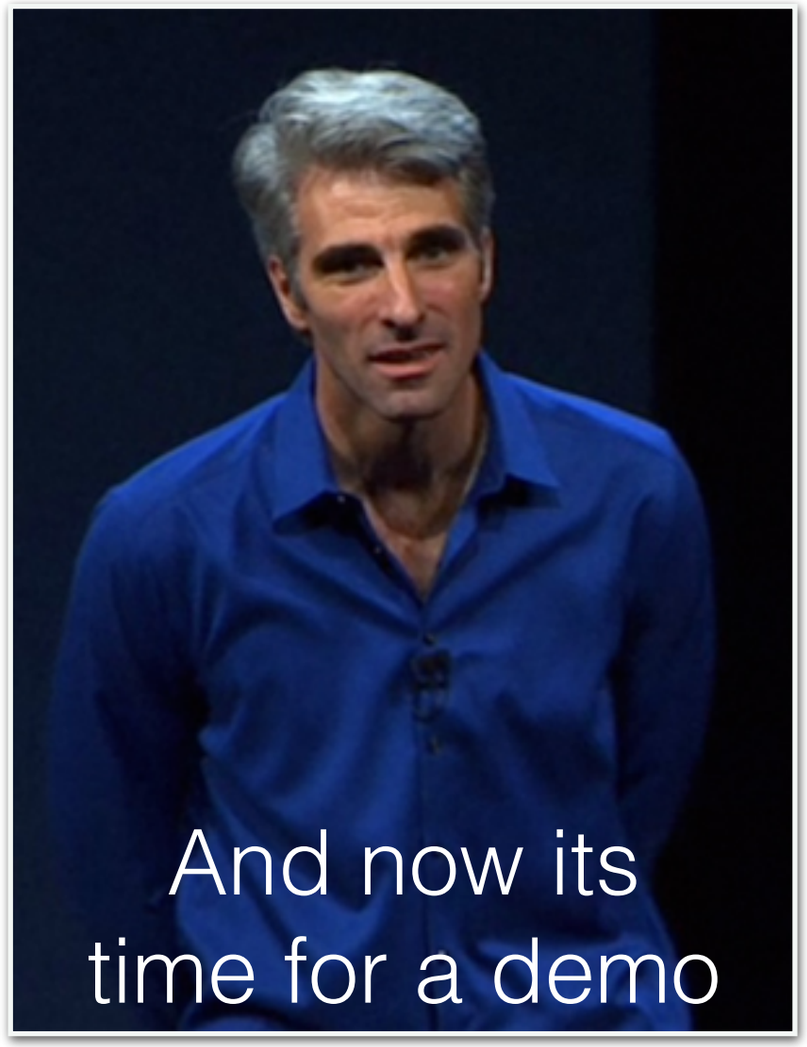
```
deviceMotion.attitude

CMAttitude* attitude

attitude.roll;
attitude.pitch;
attitude.yaw;
```



device position

device motion demo

- lets build something
  - to start: take that gravity!



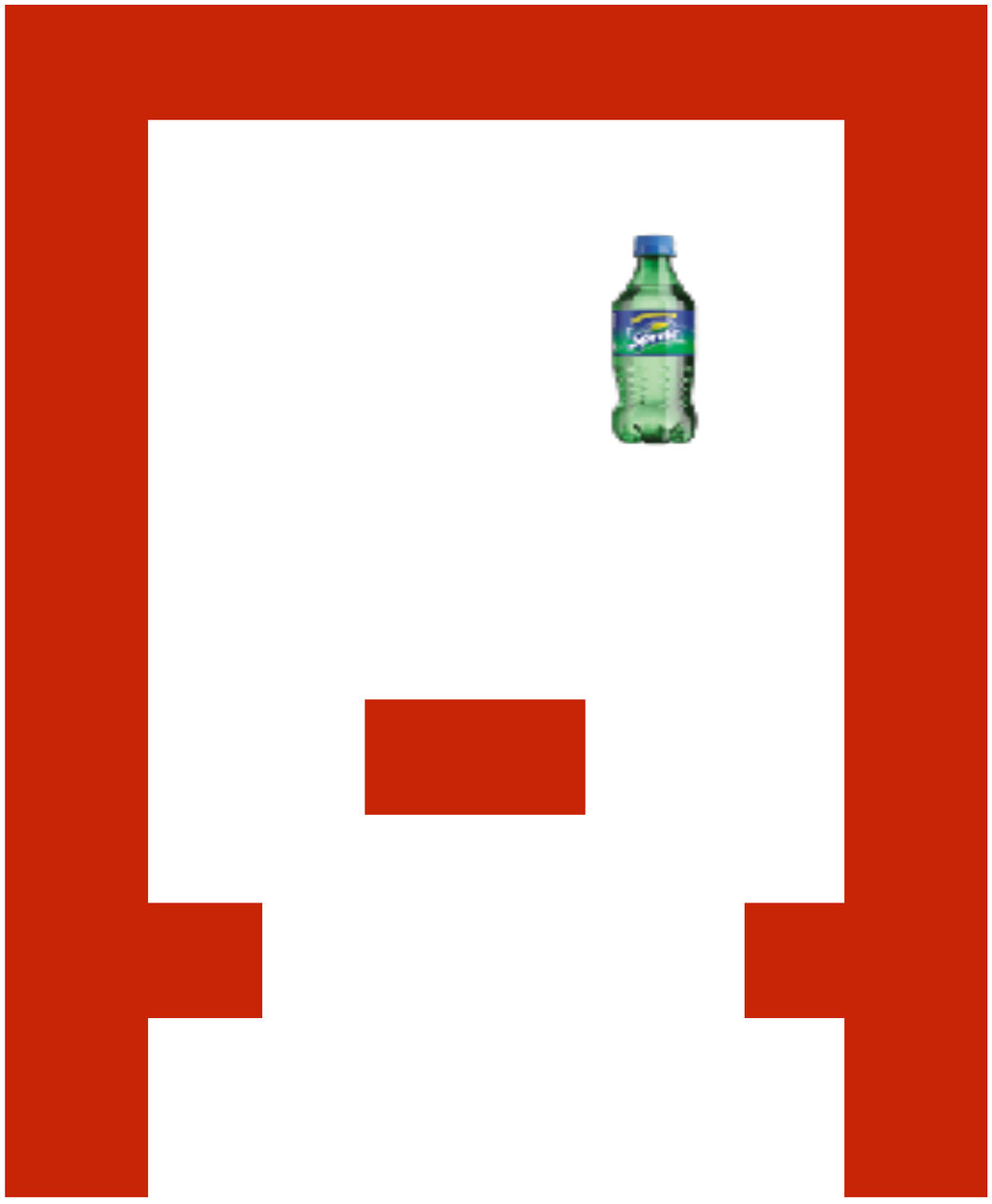
something more?

- how about a 3D physics engine?
  - Enter SceneKit
- 2D Physics? Enter SpriteKit:
  - SK abbreviated
  - real time physics engine for game applications
  - …and 2D games in general

SpriteKit

- setup game scene
- create sprites
  - color/texture
  - physical properties
    - mass
    - restitution
    - friction
    - awesomeness (not really)
- physics updated at 60 Hz

SpriteKit



create “blocks”

create “sides/top”

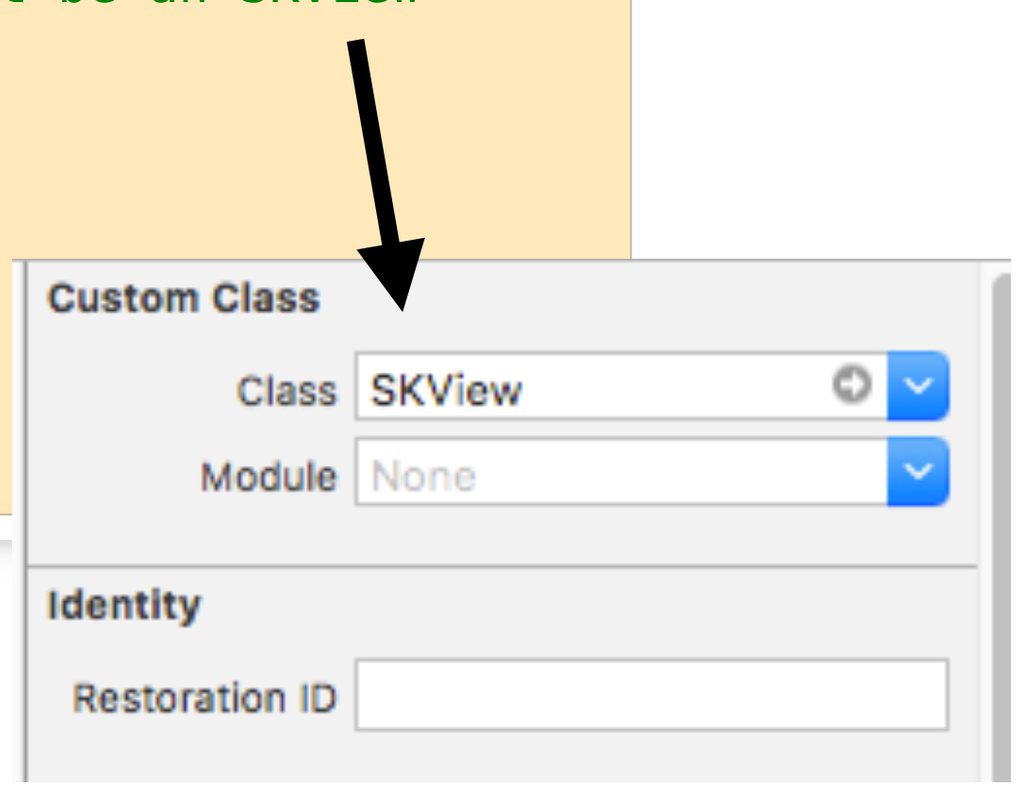
create “bouncy” sprite

make actual gravity == game gravity

user must move phone to keep sprite bouncing on target

setup view controller

```
class GameViewController: UIViewController {


    override func viewDidLoad() {
        super.viewDidLoad()

        //setup game scene
        let scene = GameScene(size: view.bounds.size)
        let skView = view as! SKView // must be an SKView
        skView.showsFPS = true
        skView.showsNodeCount = true
        skView.ignoresSiblingOrder = true
        scene.scaleMode = .ResizeFill
        skView.presentScene(scene)
    }
}
```



build sprites (in GameScene)

```
override func didMoveToView(view: SKView) {

        backgroundColor = SKColor.whiteColor()

        // make sides to the screen
        self.addSidesAndTop()

        // add some stationary blocks
        self.addStaticBlockAtPoint(CGPoint(x: size.width * 0.1, y: size.height * 0.25))
        self.addStaticBlockAtPoint(CGPoint(x: size.width * 0.9, y: size.height * 0.25))

        // add a spinning block
        self.addBlockAtPoint(CGPoint(x: size.width * 0.5, y: size.height * 0.35))

        self.addSprite()
    }
```

make border

make blocks

add bouncy sprite

```
func addSprite(){
        let spriteA = SKSpriteNode(imageNamed: "sprite") // this is a sprite bottle

        spriteA.size = CGSize(width:size.width*0.1,height:size.height * 0.1)
        spriteA.position = CGPoint(x: size.width * 0.1, y: size.height * 0.75)

        spriteA.physicsBody = SKPhysicsBody(rectangleOfSize:spriteA.size)
        spriteA.physicsBody?.restitution = 1.5
        spriteA.physicsBody?.dynamic = true

        self.addChild(spriteA)
    }
```

add image texture

size & position

interaction physics

add to scene

set gravity

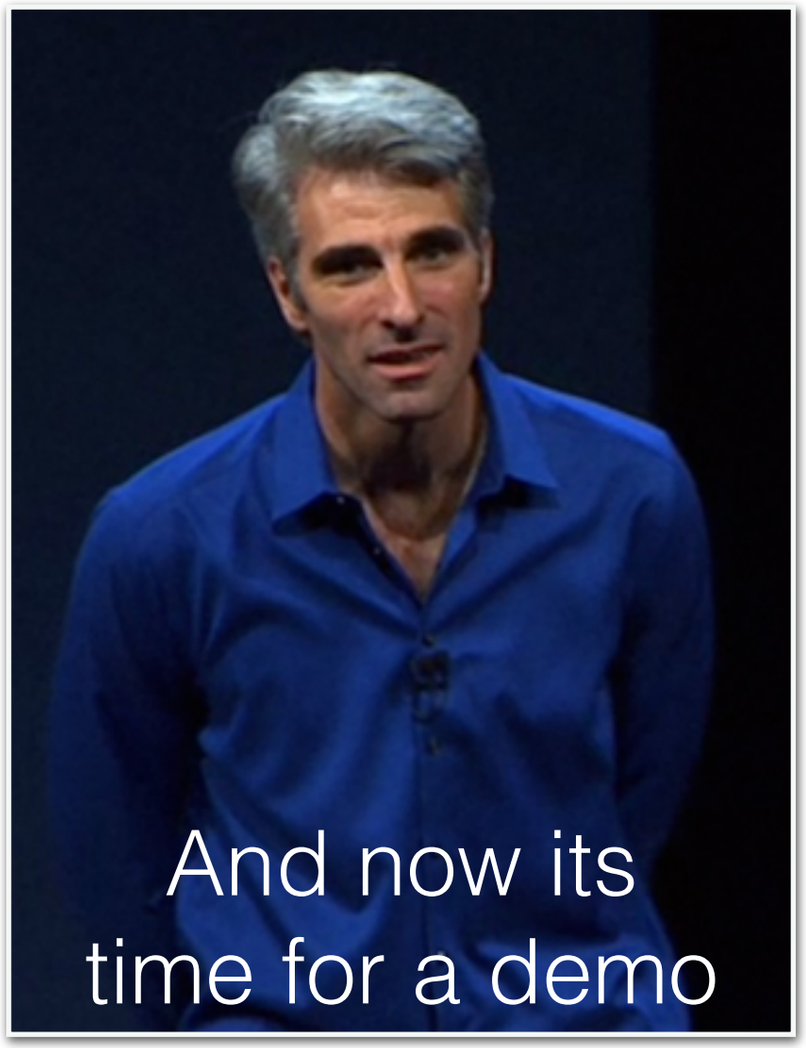
```
let motion = CMMotionManager()
func startMotionUpdates(){
    // some internal inconsistency here:
    // we need to ask the device manager for device

    if self.motion.deviceMotionAvailable{
        self.motion.deviceMotionUpdateInterval = 0.1
        self.motion.startDeviceMotionUpdatesToQueue(NSOperationQueue.mainQueue(),
                                                    withHandler: self.handleMotion)
    }
}

func handleMotion(motionData:CMDeviceMotion?, error:NSError?){
    if let gravity = motionData?.gravity {
        self.physicsWorld.gravity = CGVectorMake(CGFloat(9.8*gravity.x),
                                                 CGFloat(9.8*gravity.y))
    }
}
```

start motion

adjust physics

device motion demo 2

- lemon lime bounce
- pre-made demo



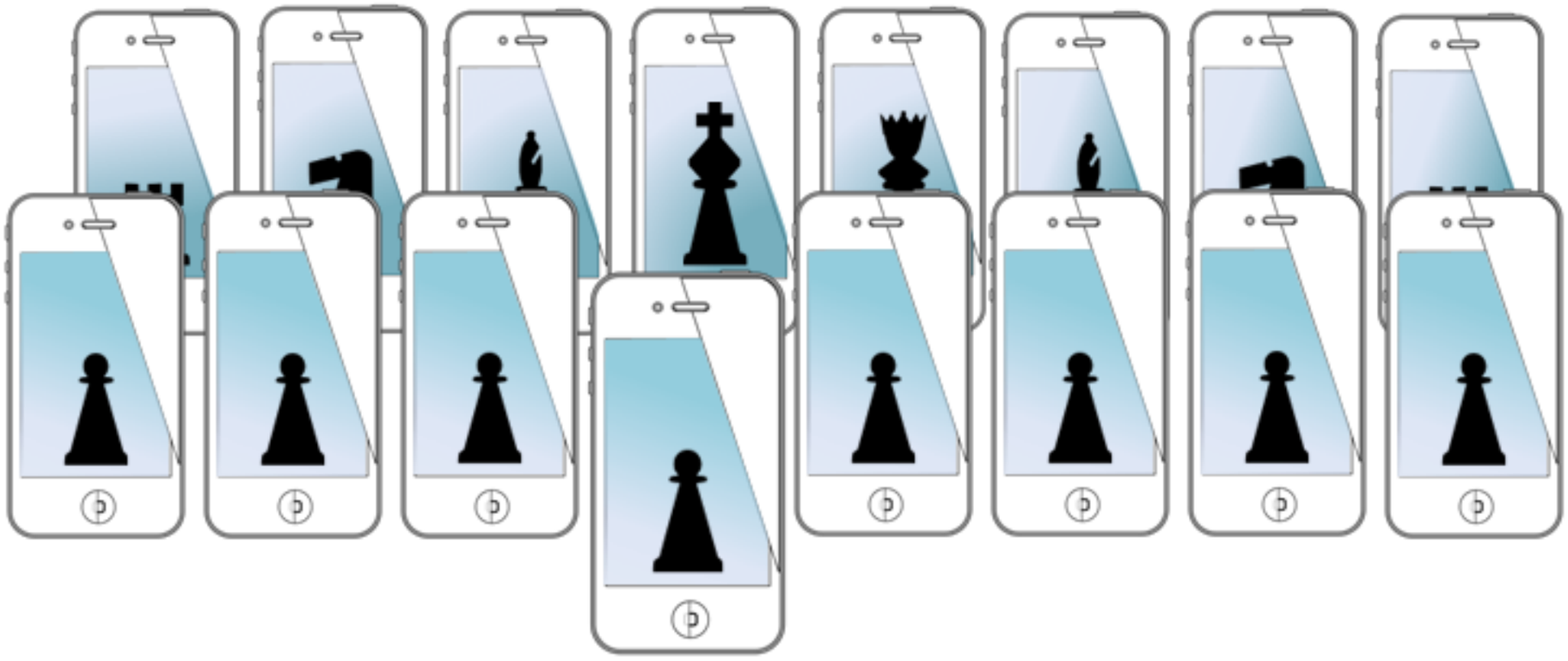
the end of motion…

- before moving on…
- one week lab next time…
- assignment posted

for next time…

- Image processing!

MOBILE SENSING LEARNING



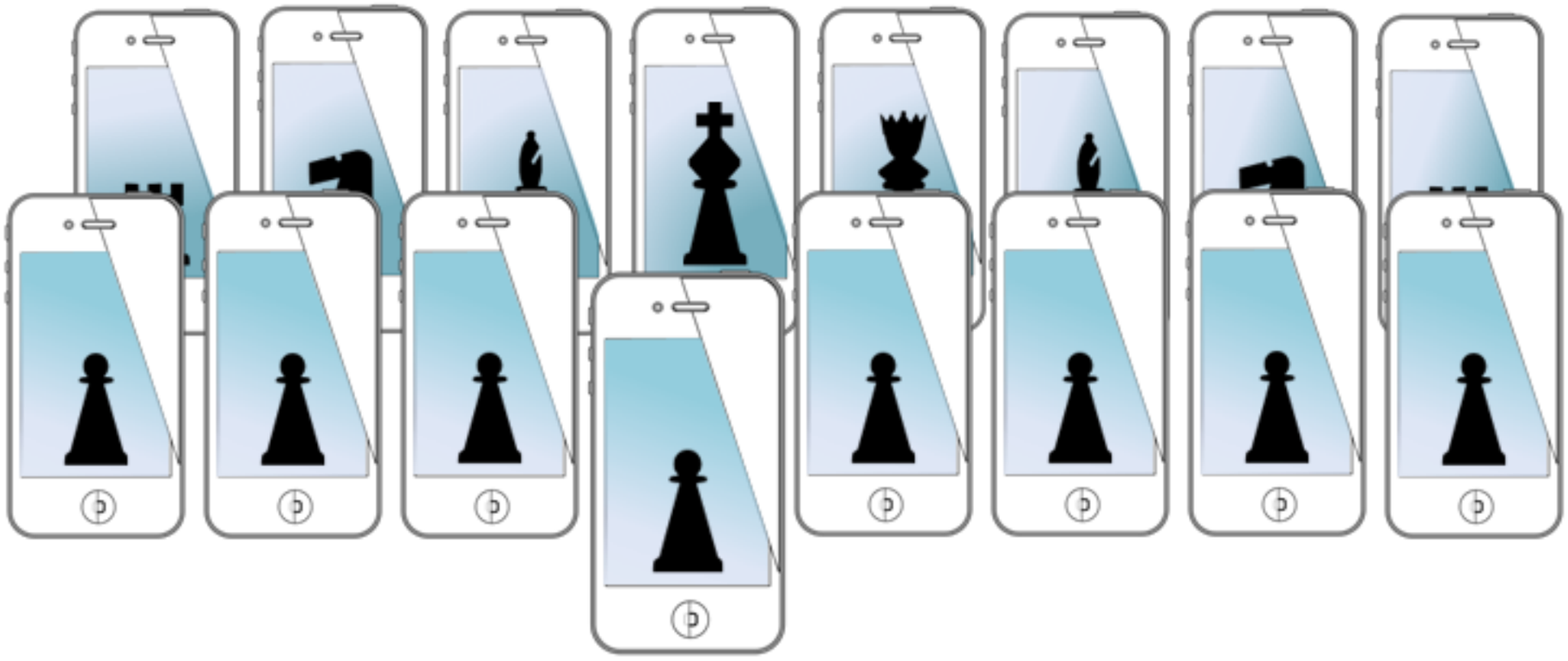
CSE5323 & 7323

Mobile Sensing and Learning

week 5 lecture b: activity, pedometers, and motion sensing

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

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

Supplemental Slides: vector trajectory and profiling

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

supplemental slides

- vector trajectory

phone trajectory

- what direction is the phone (user) headed?
- direction could be:
  - cardinal {N, S, E, W} — GPS and magnetometer
  - altitude {sea level, +30 feet, etc.} — GPS
  - relative altitude {up, down} — motion sensors
  - relative trajectory {left, right, straight} — motion sensors
- how should we sense each of these?

up/down movement

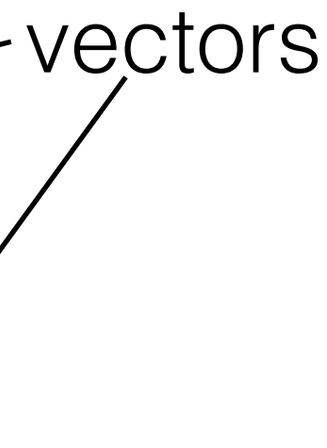
- questions:
  - are we accelerating?
  - in what direction are we accelerating?
  - are we accelerating opposite of gravity?

which way is gravity?

```
deviceMotion.gravity.{x,y,z}
```

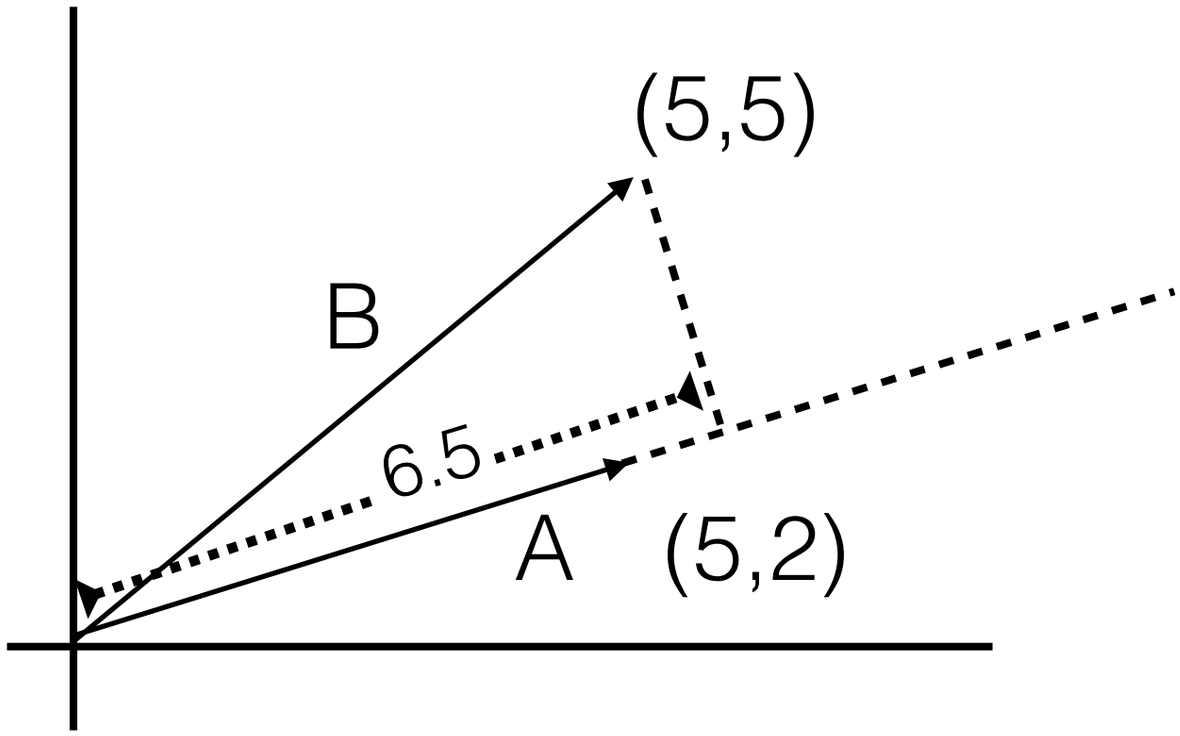
which way is the phone accelerating?

```
deviceMotion.userAcceleration.{x,y,z}
```



vector direction

- how much of one vector is in the direction of another?
- projections



$$\frac{A \bullet B}{|A|}$$

$$\frac{(5,5)\bullet(5,2)}{|(5,2)|}$$

$$\frac{5*5+5*2}{\sqrt{(5^2+2^2)}} = 35/\sqrt{29} \sim 6.5$$

vector direction

- acceleration of the user towards or away from gravity?

```
CMAcceleration gravity, CMAcceleration userAccel

float dotProduct =
  gravity.x*userAccel.x + gravity.y*userAccel.y + gravity.z*userAccel.z;

float normDotProd =
dotProduct / (gravity.x*gravity.x + gravity.y*gravity.y + gravity.z*gravity.z);
```

positive acceleration is speeding up
negative acceleration is slowing down

vector acceleration demo

- don’t drop it!

profiling demo

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