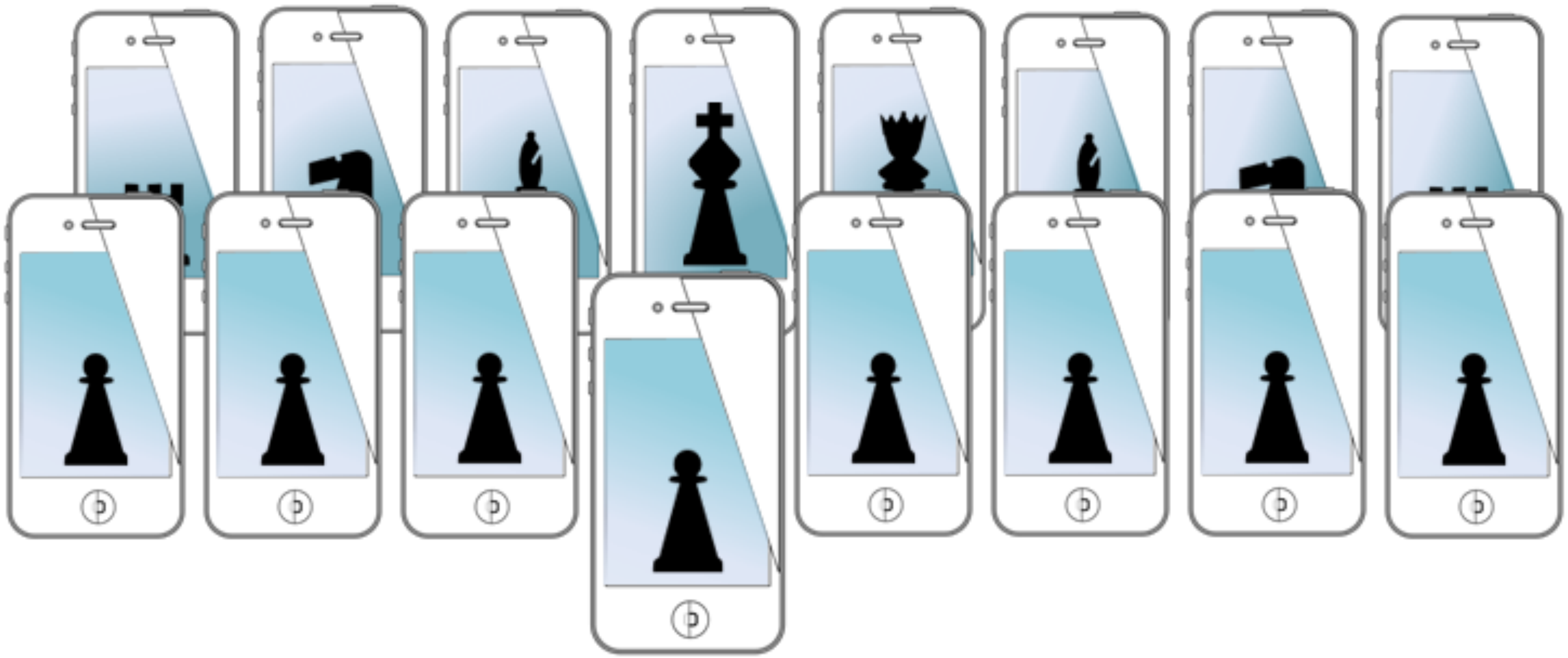
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

activity, pedometers, and motion sensing

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

course logistics

- A2 is due next week
  - everyone okay?

agenda

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

storing persistent defau

- 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}?
```

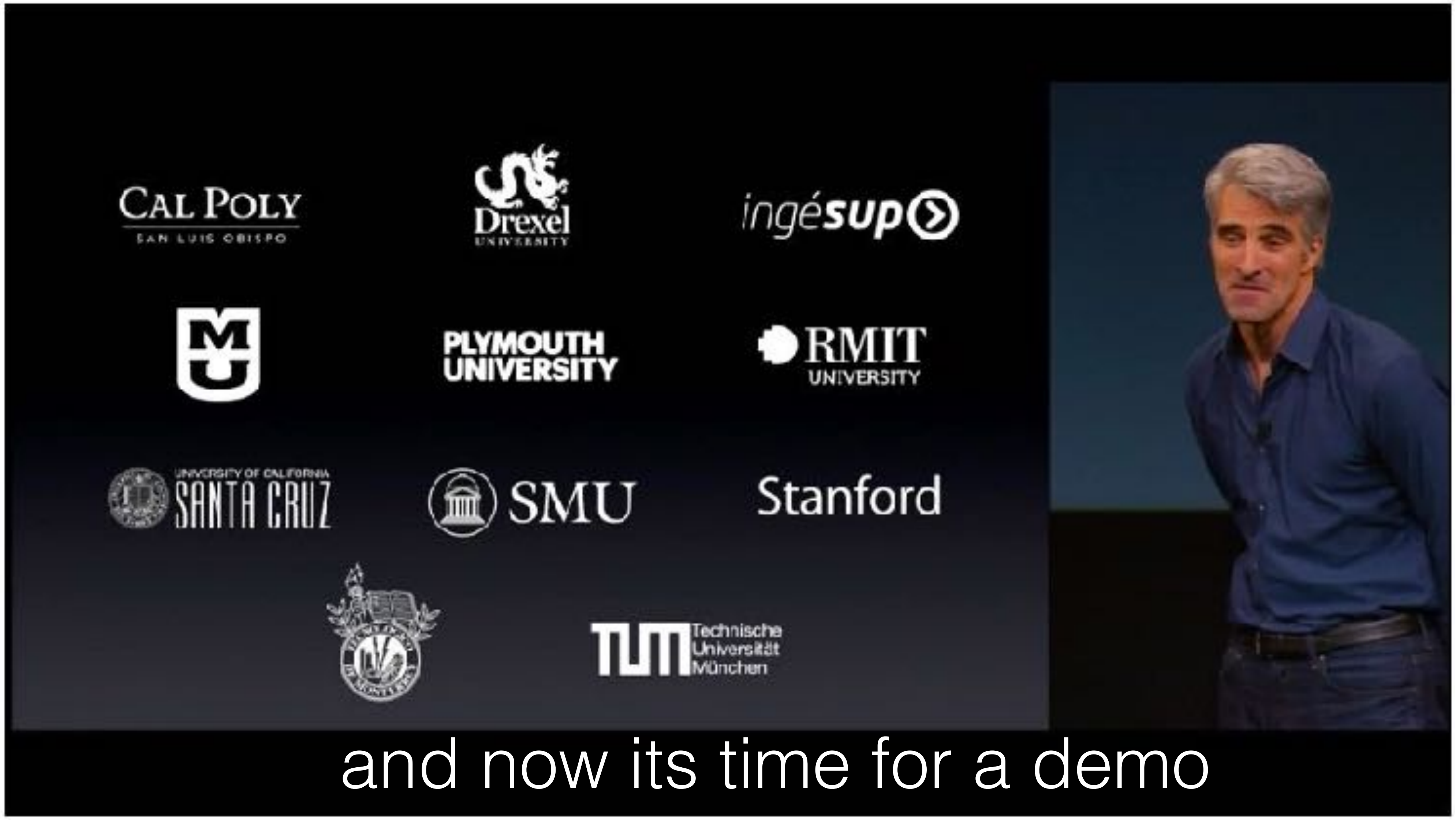
import defaults

primitives

objects

access saved objects

M-# pedometer/activity demo



“continue” 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

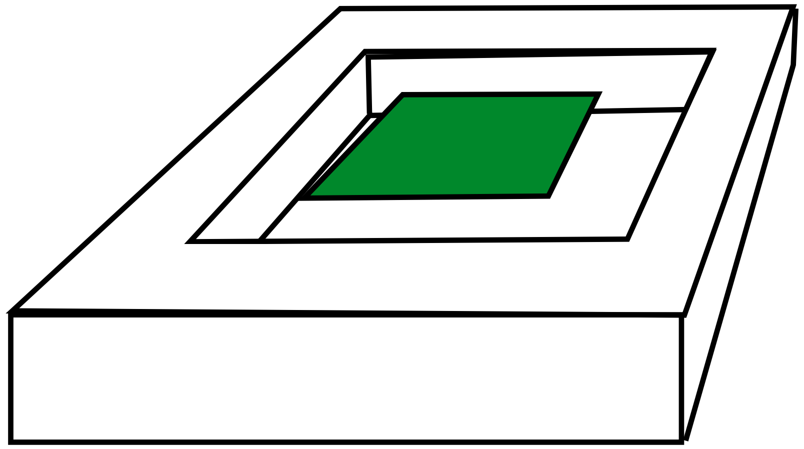
- M-# mediates access to data
- much lower battery consumption

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	~?%	?%

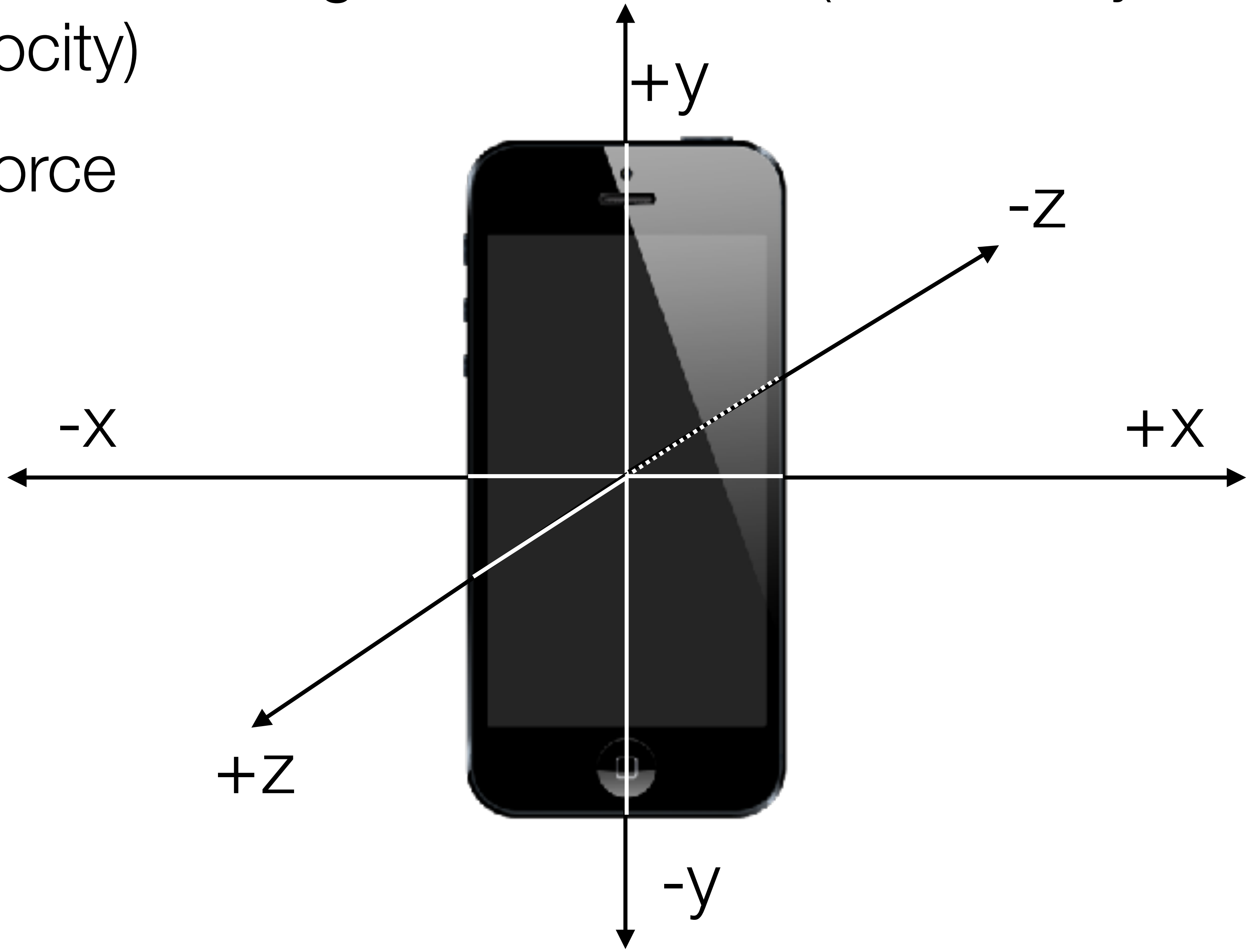
accelerometers

- 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



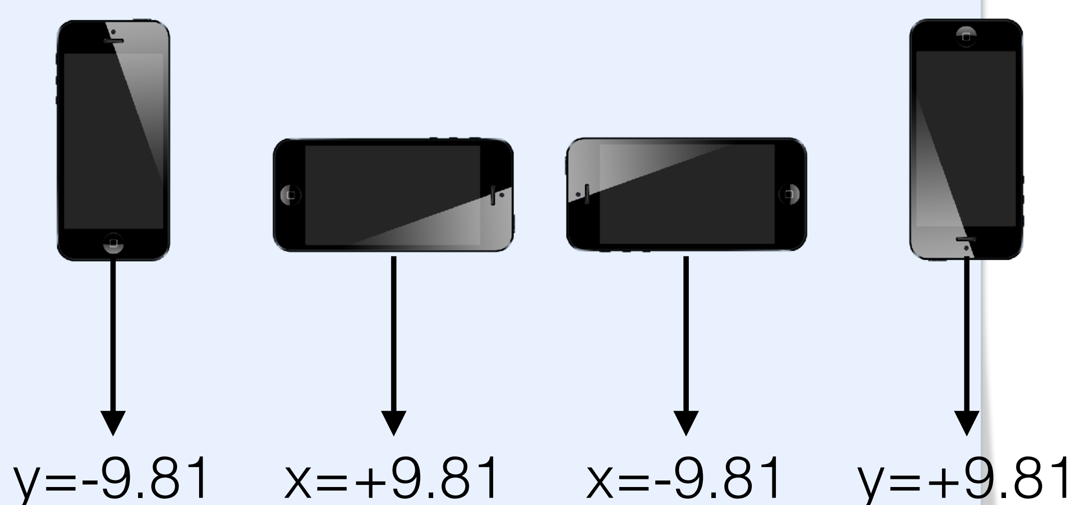
accessing the accelerometer

- 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

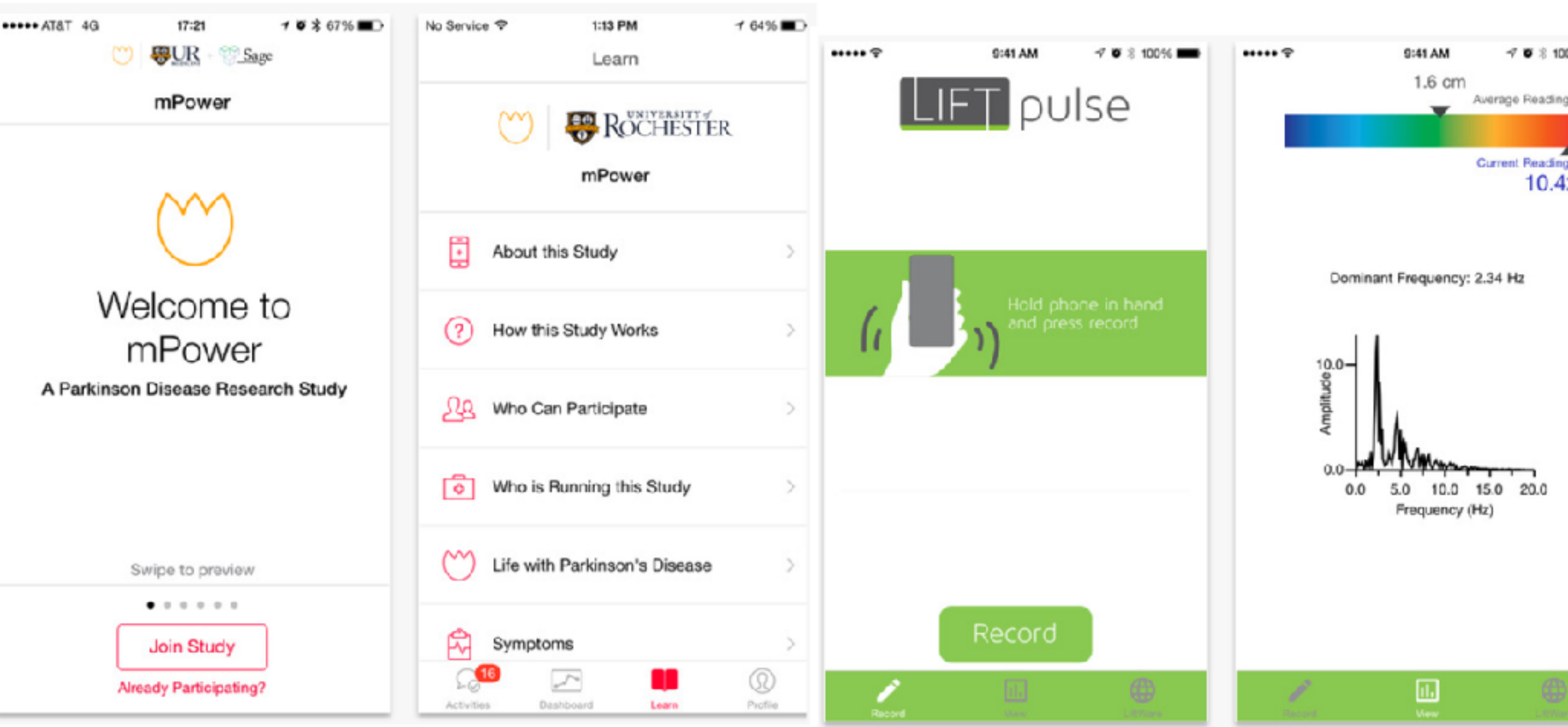
```
CMDeviceMotion *deviceMotion

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



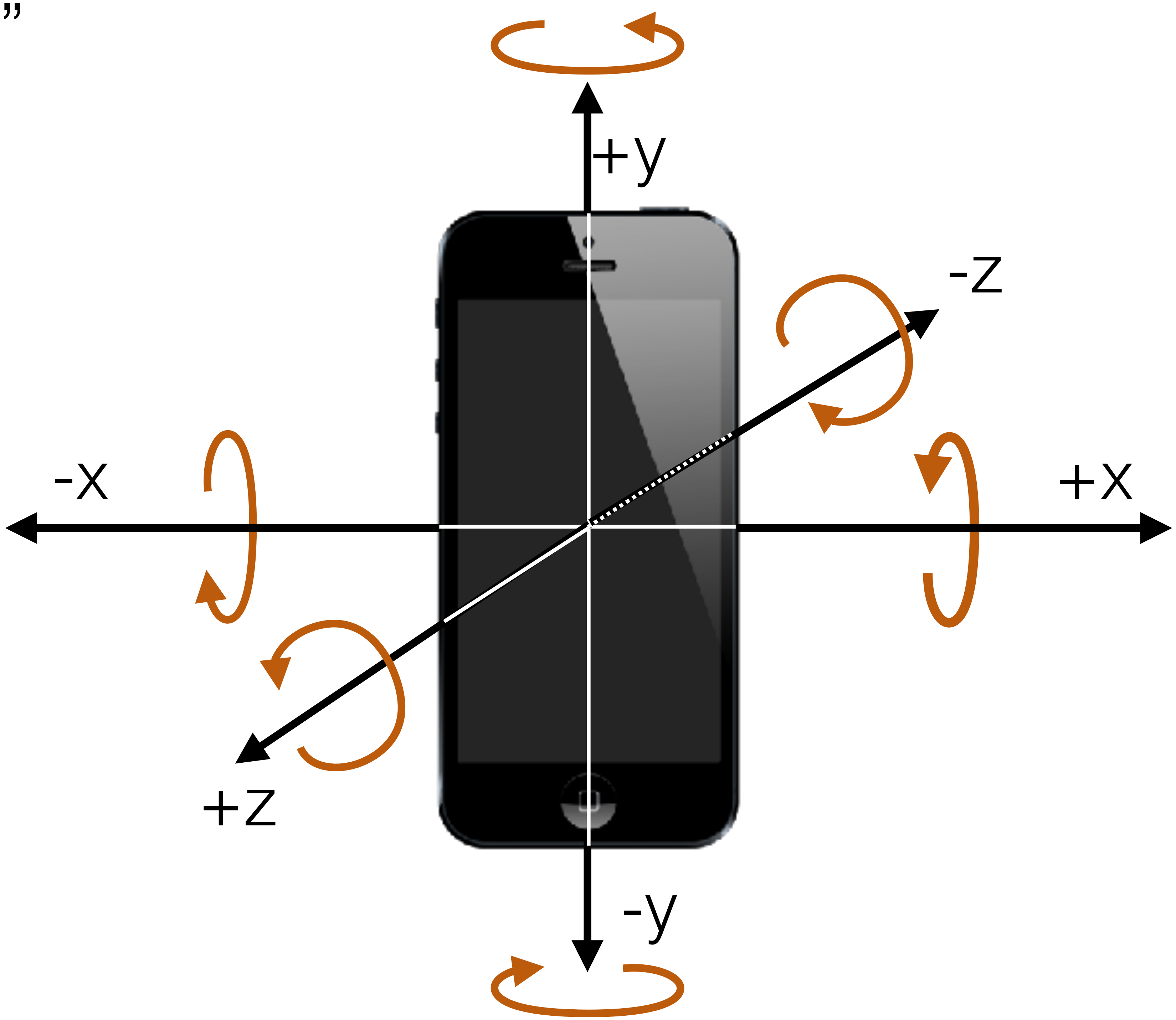
gyroscope

- 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”



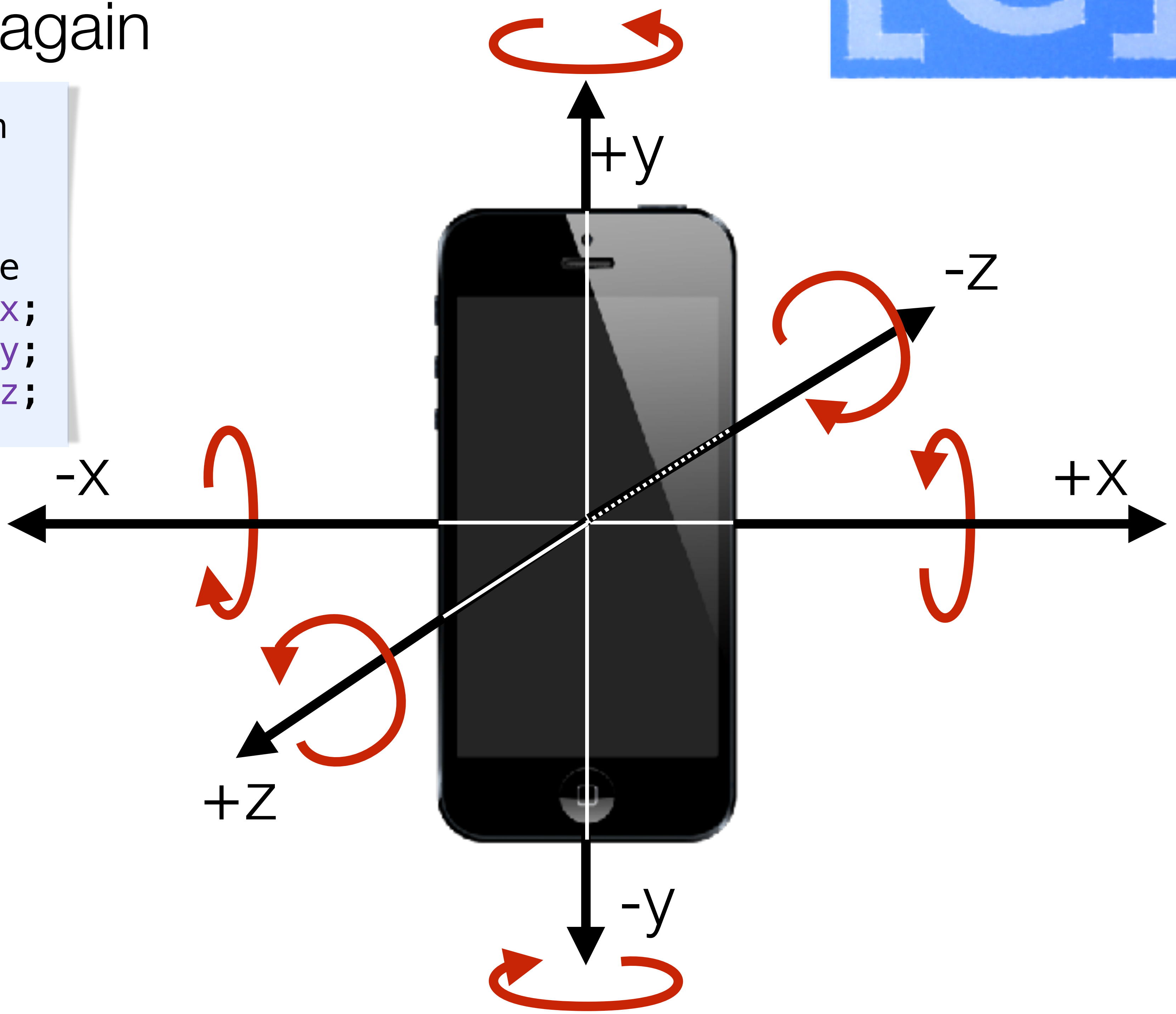
accessing the gyro

- use device motion again

```
CMDeviceMotion *deviceMotion

deviceMotion.rotationRate

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

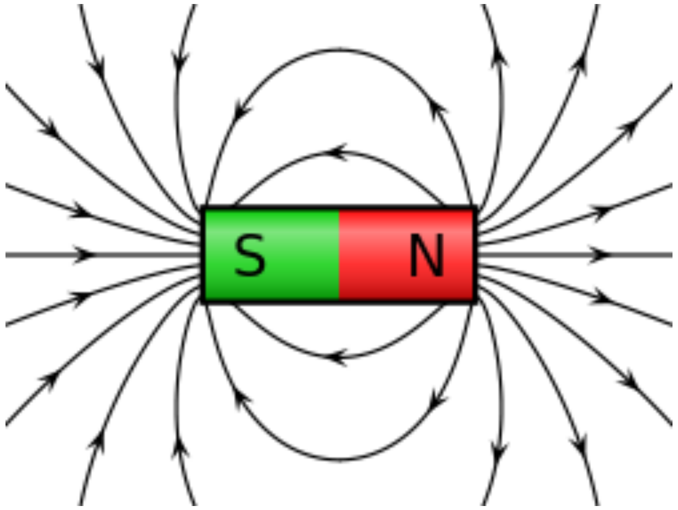
measures **rate** of motion

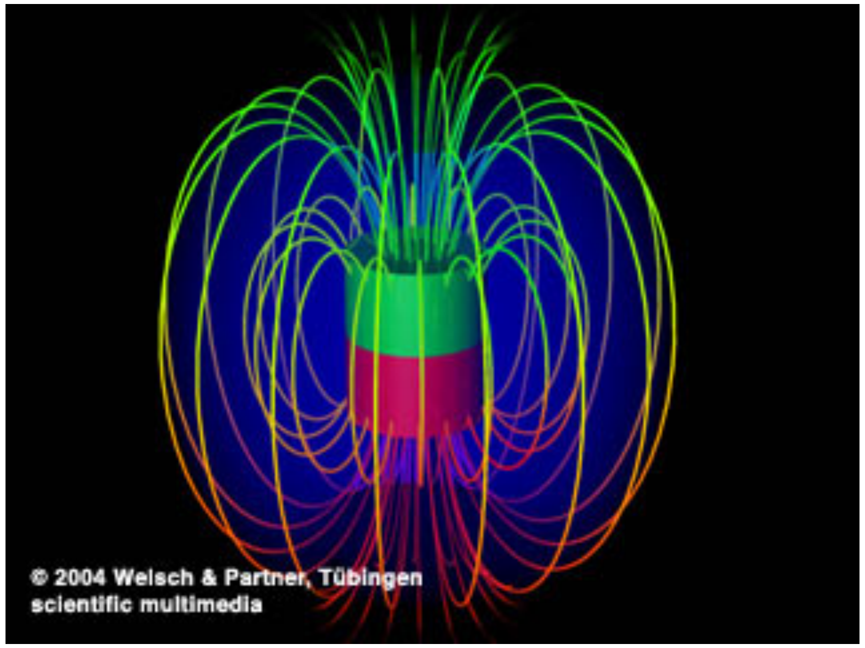
in this example, saves it to array



magnetometers

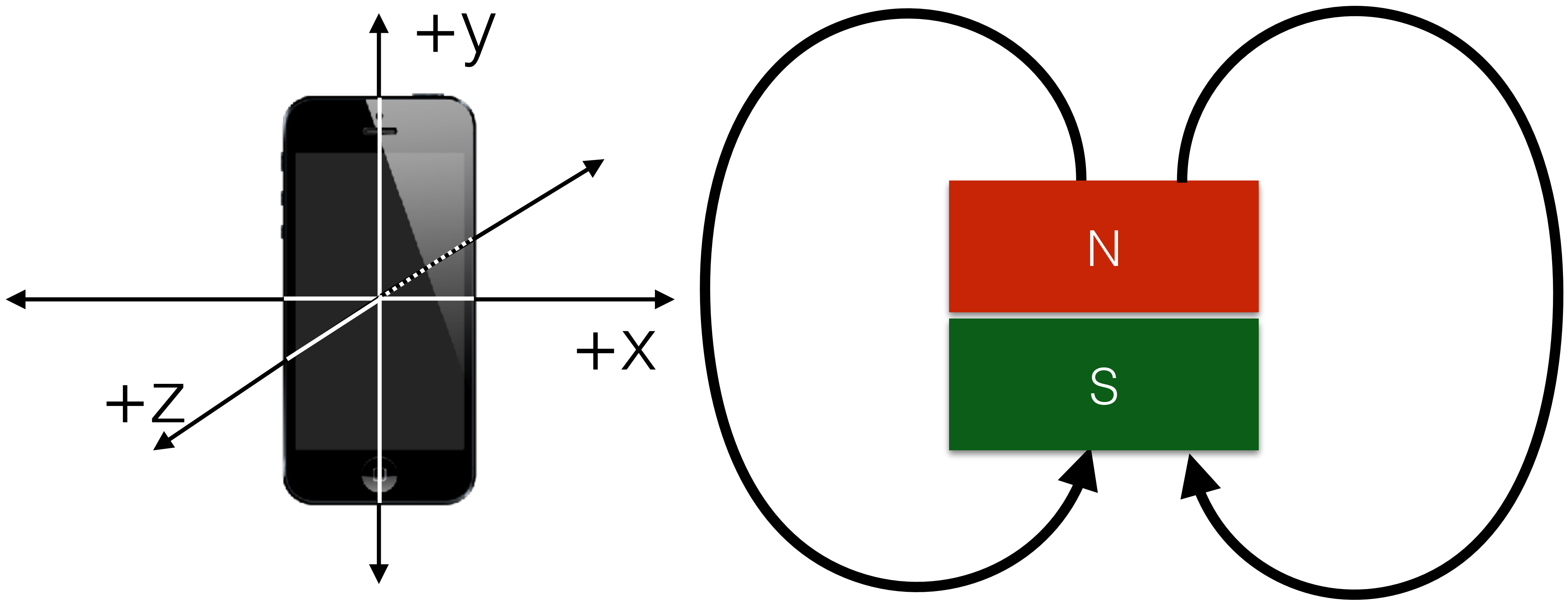
- measure magnetic fields
- magnets are measured in tesla (T)
    - **how**: essentially, there is a tight coupling between electricity flow and magnetic fields
- earth's magnetic field varies, but is around 50 uT
- iPhone can measure up to 1T with a resolution of about 8uT
- magnetic fields have direction!





magnetic fields

- measure magnetic field along axis, towards “south”

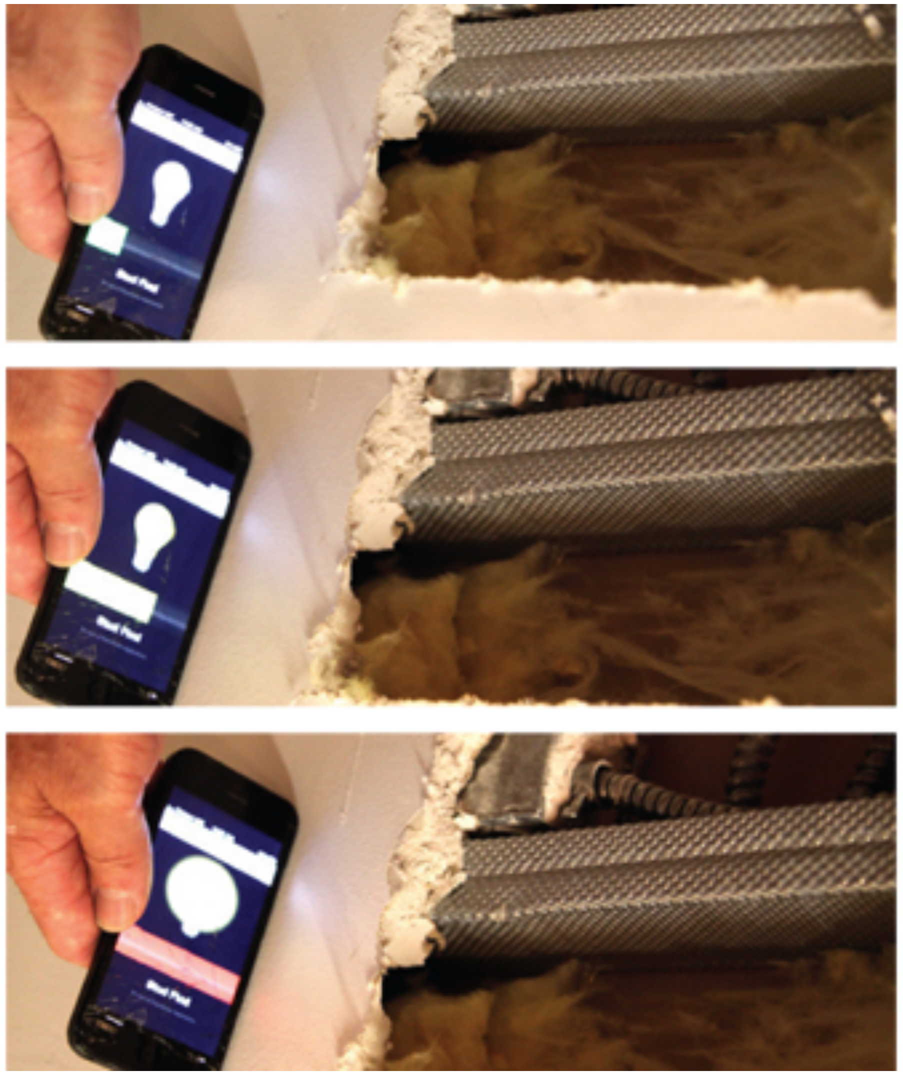


but iPhone has magnetic bias

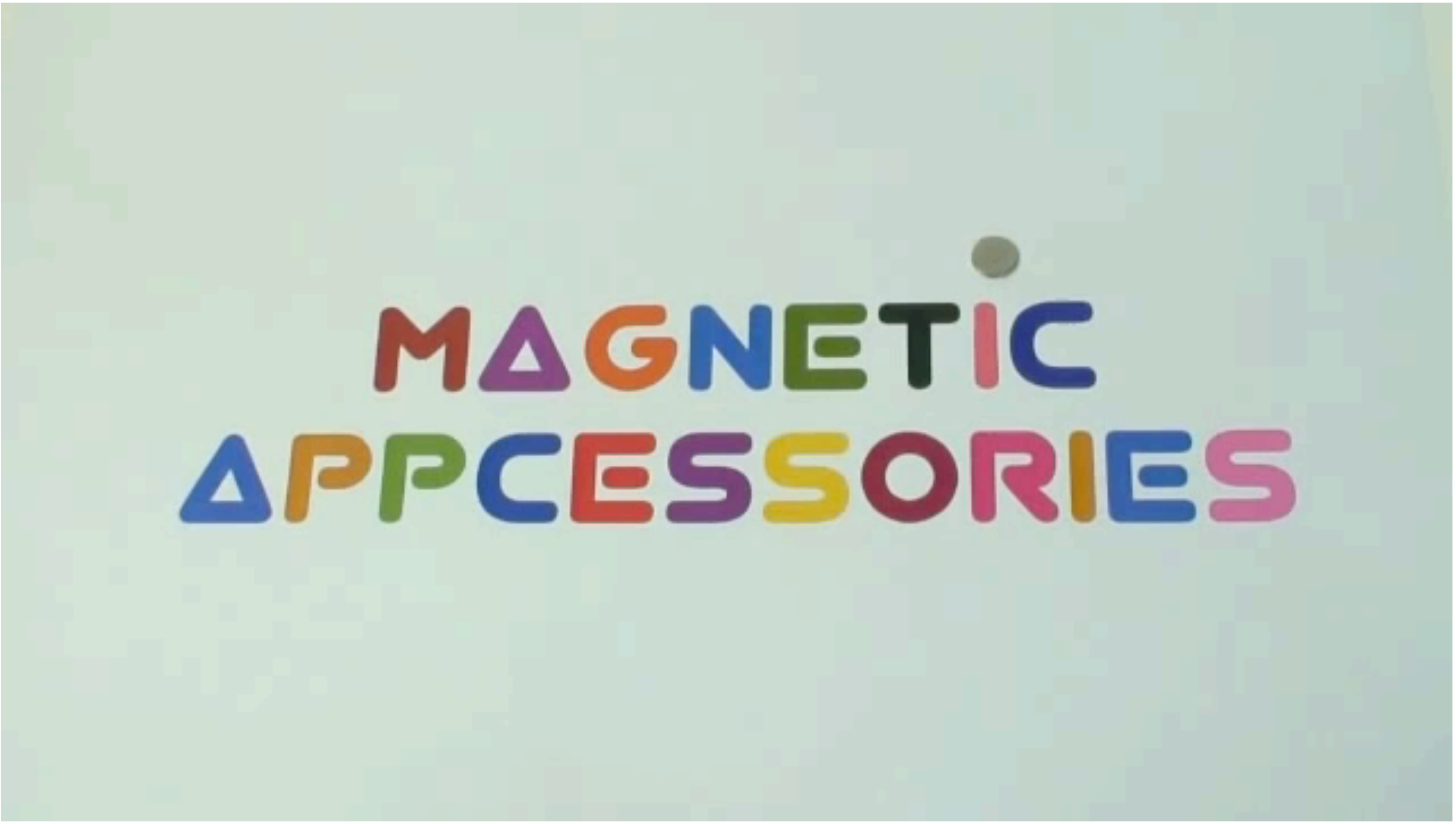
- the phone uses electricity and therefore is a magnet
  - good thing Apple subtracts that out for us!

```
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

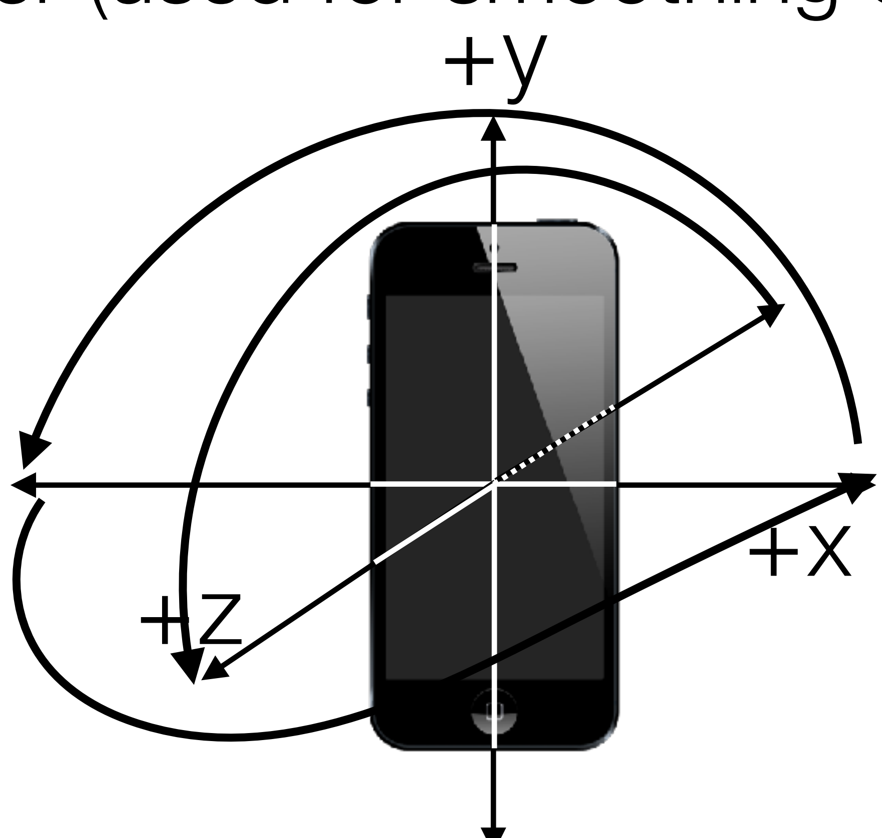


a cool example



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

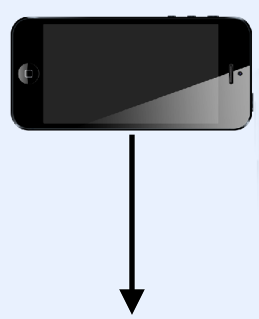
queue to run on

how often to push updates

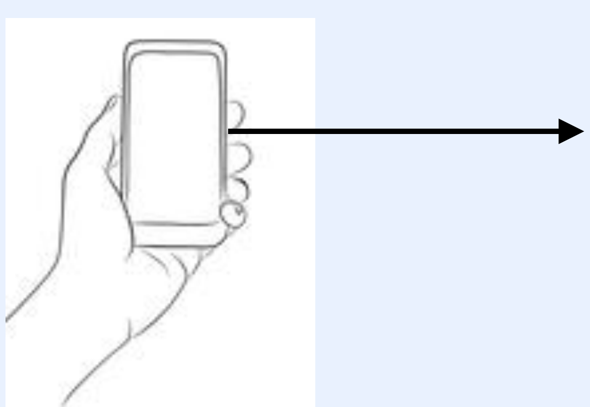
the data

summary

```
CMDeviceMotion *deviceMotion

deviceMotion.gravity
deviceMotion.userAcceleration

CMAcceleration gravity,
CMAcceleration userAcceleration

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

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



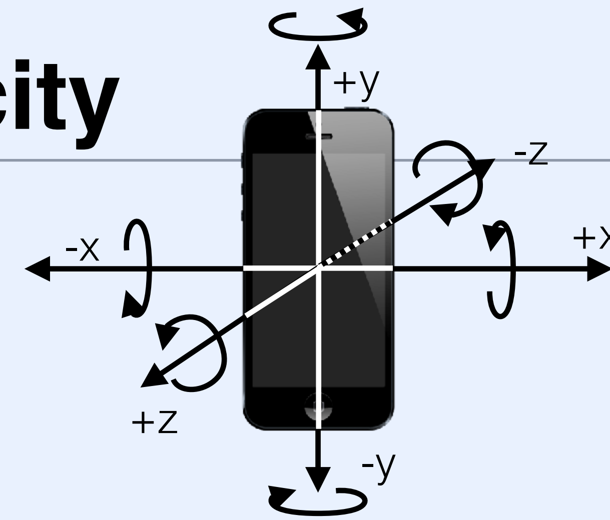
x=+9.81



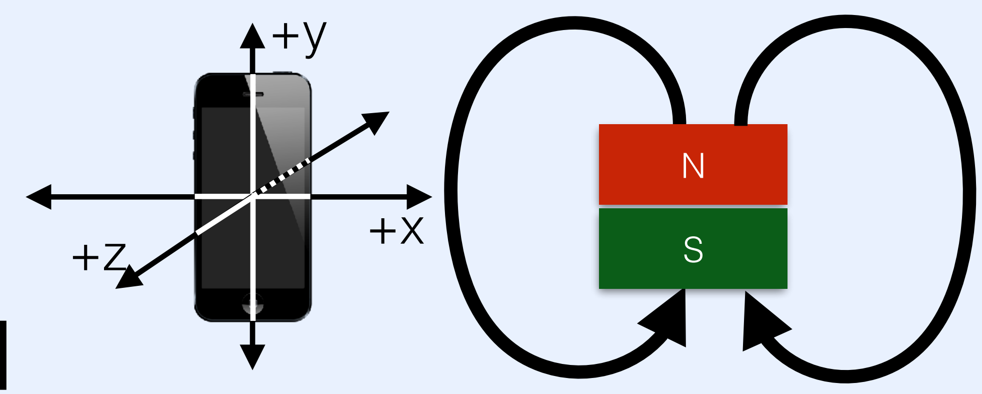
acceleration

rotation velocity

```
deviceMotion.rotationRate
CMRotationRate rotationRate
rotationRate.x;
rotationRate.y;
rotationRate.z;
```

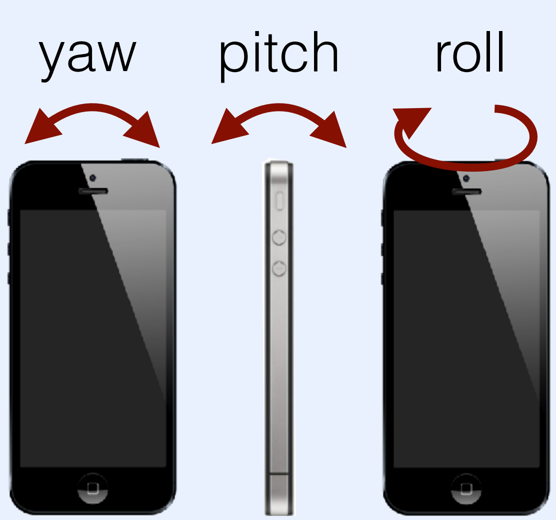



```
deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

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

magneticField.accuracy
```



magnetic field

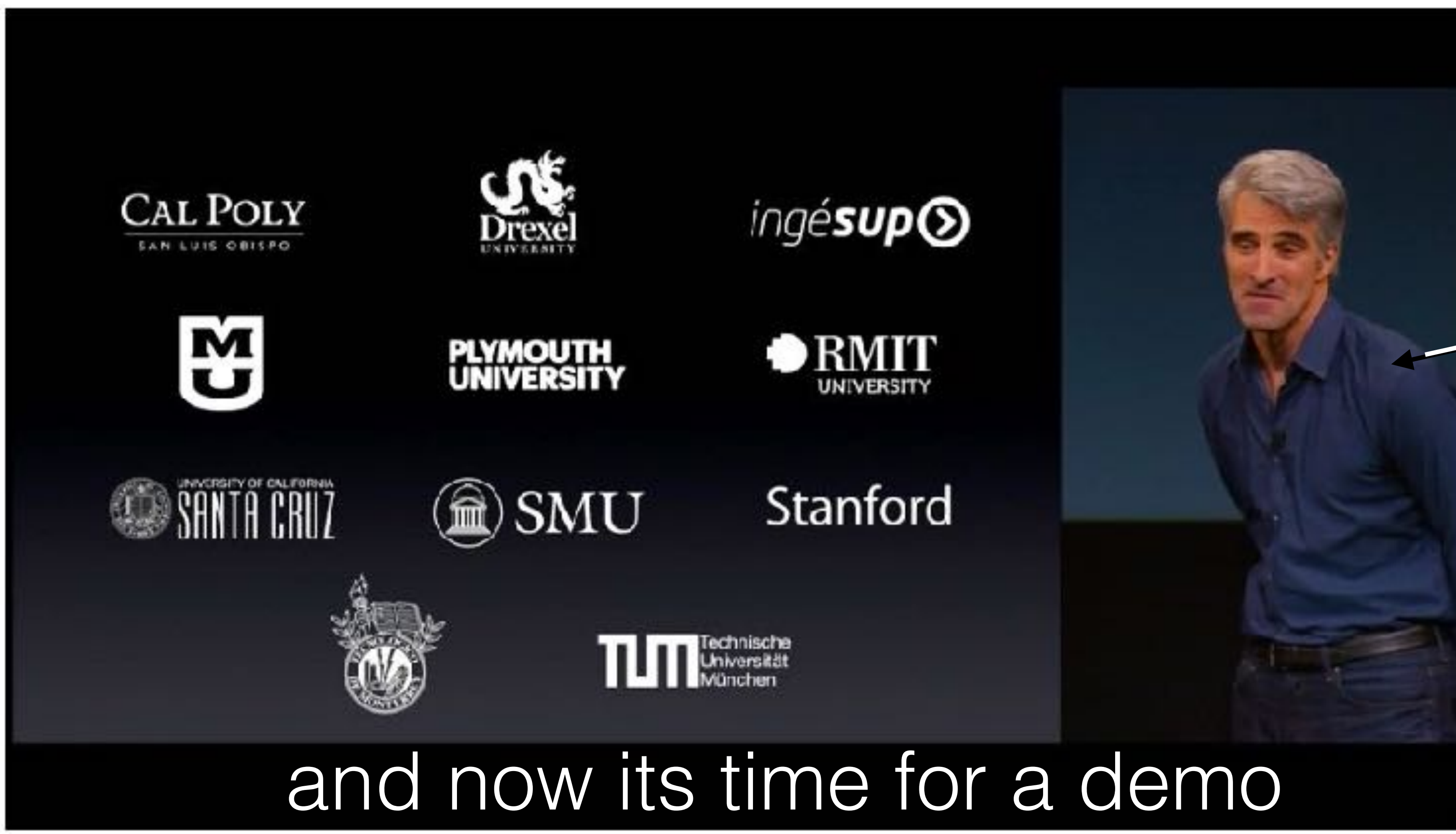
```
deviceMotion.attitude

CMAttitude* attitude

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

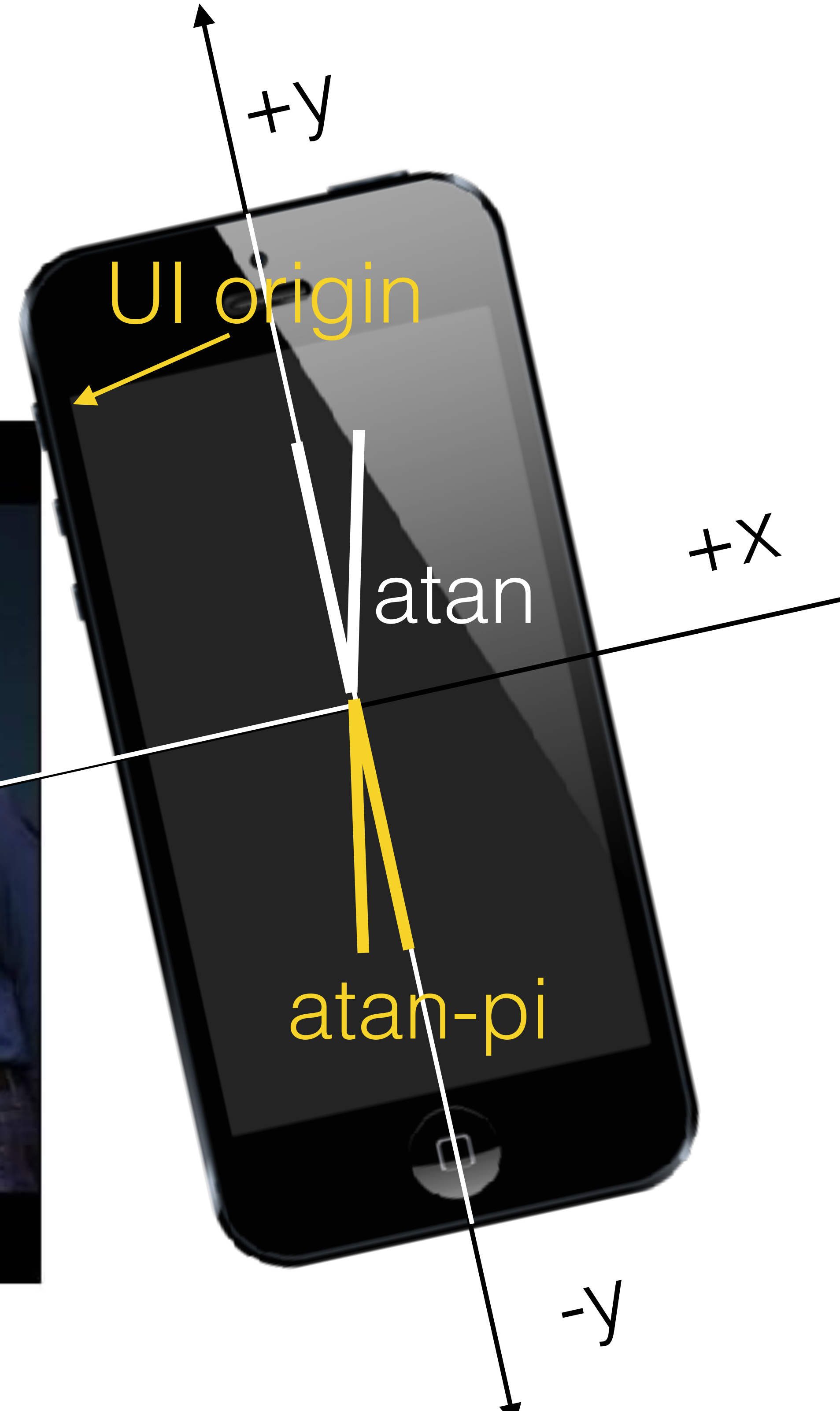


device position

device motion demo

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





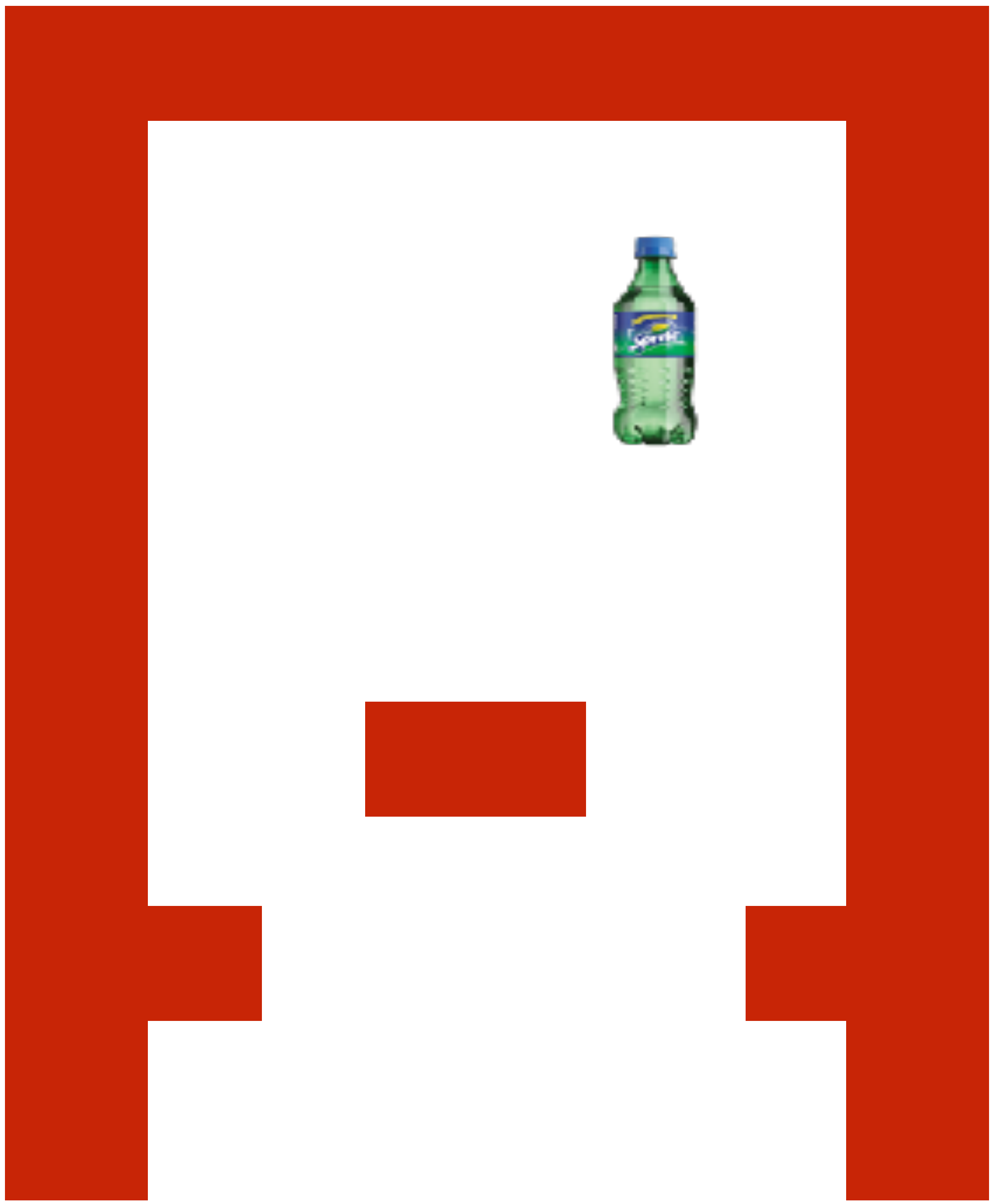
something more?

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

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”

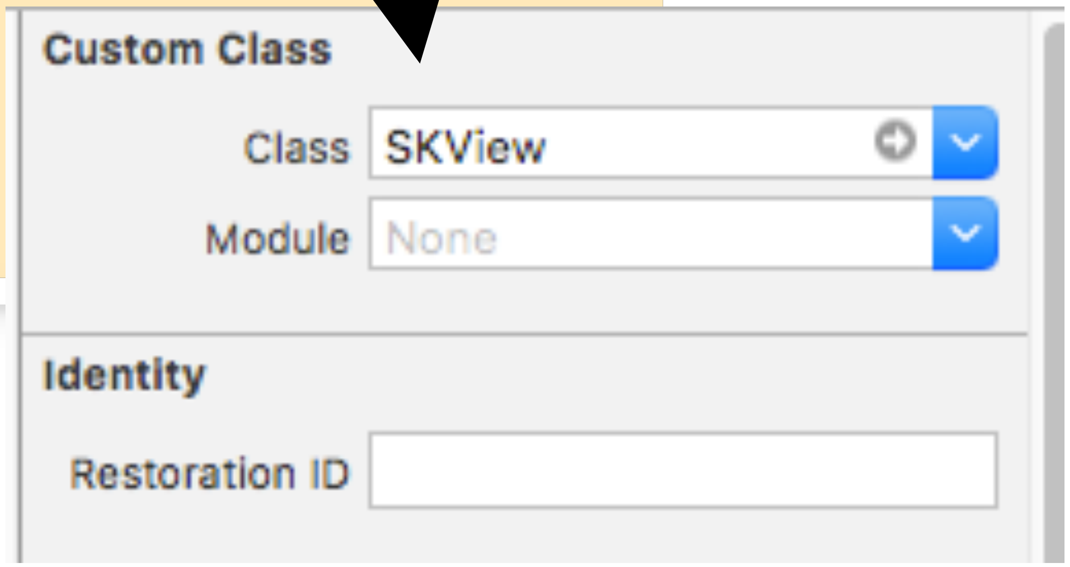
create “bouncy” sprite

make actual gravity
== game gravity

user must move phone
to keep sprite bouncing
on target

setup view controller

```swift
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)
    }
}
```



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

build sprites example

```
func addSpriteBottle(){
        let spriteA = SKSpriteNode(imageNamed: "sprite")

        spriteA.size = CGSize(width:size.width*0.1,height:size.height * 0.1)

        let randNumber = random(min: CGFloat(0.1), max: CGFloat(0.9))
        spriteA.position = CGPoint(x: size.width * randNumber, y: size.height * 0.75)

        spriteA.physicsBody = SKPhysicsBody(rectangleOf:spriteA.size)
        spriteA.physicsBody?.restitution = random(min: CGFloat(1.0), max: CGFloat(1.5))
        spriteA.physicsBody?.isDynamic = true
        spriteA.physicsBody?.contactTestBitMask = 0x00000001
        spriteA.physicsBody?.collisionBitMask = 0x00000001
        spriteA.physicsBody?.categoryBitMask = 0x00000001

        self.addChild(spriteA)
    }
```

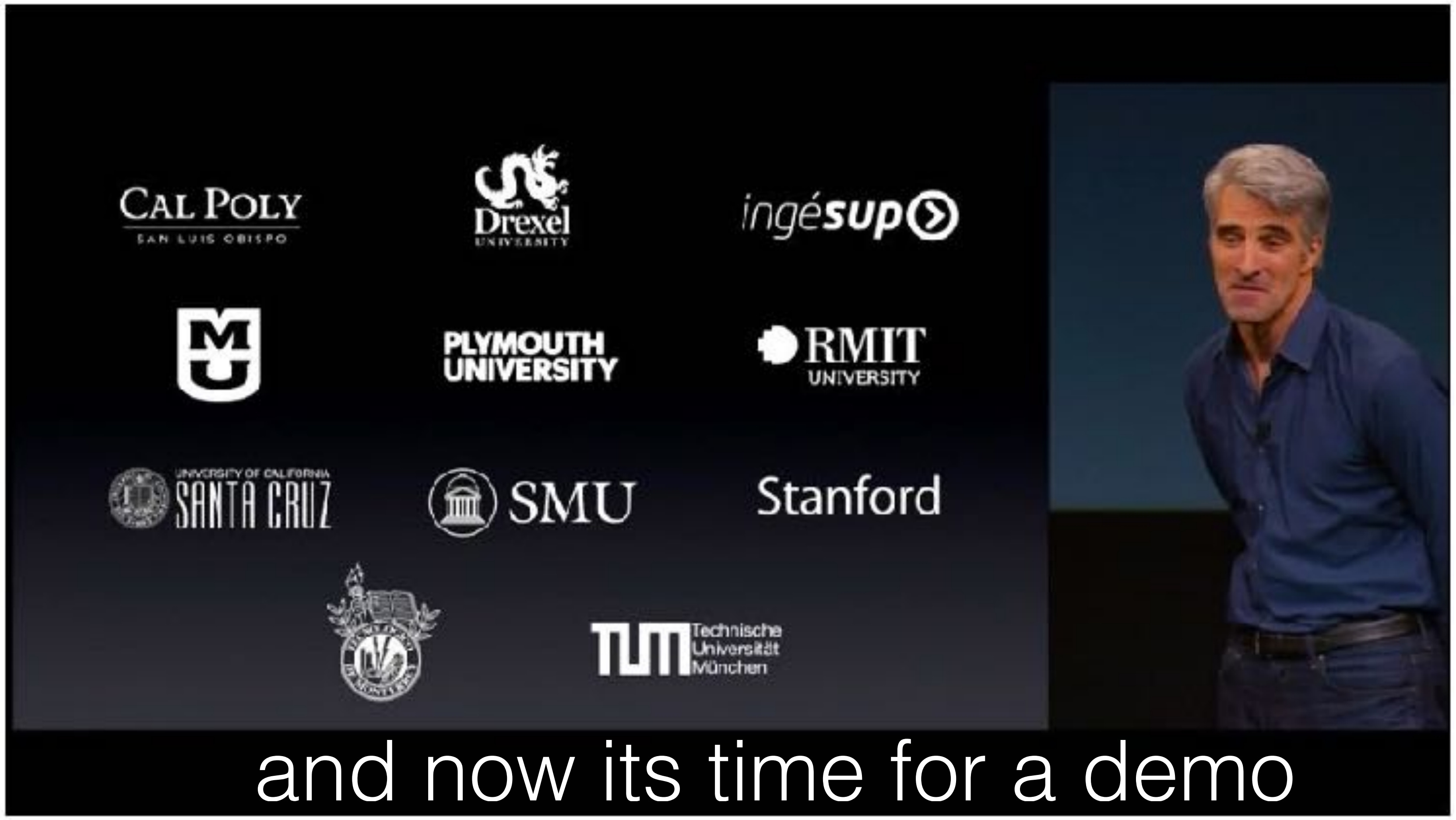
add image texture

interaction physics

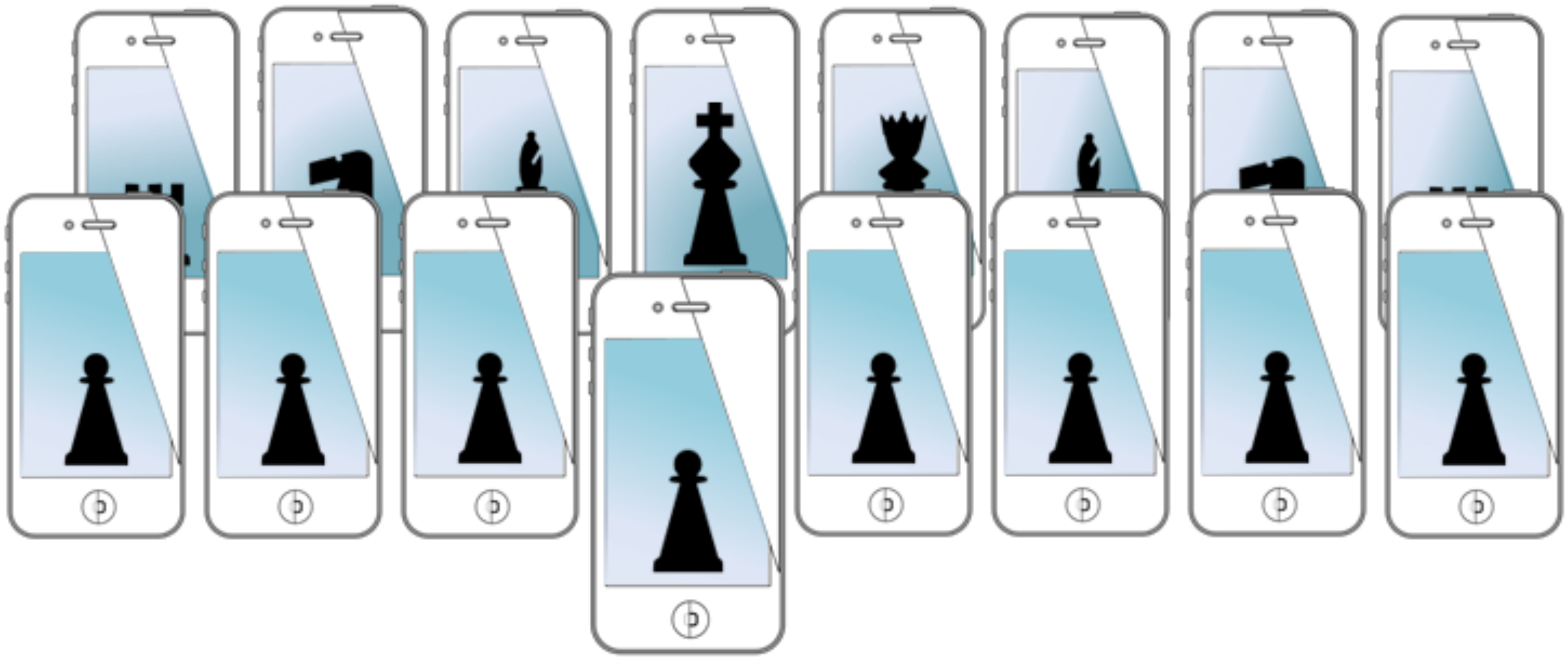
add to scene

device motion demo 2

- lemon lime bounce
- pre-made demo



MOBILE SENSING LEARNING



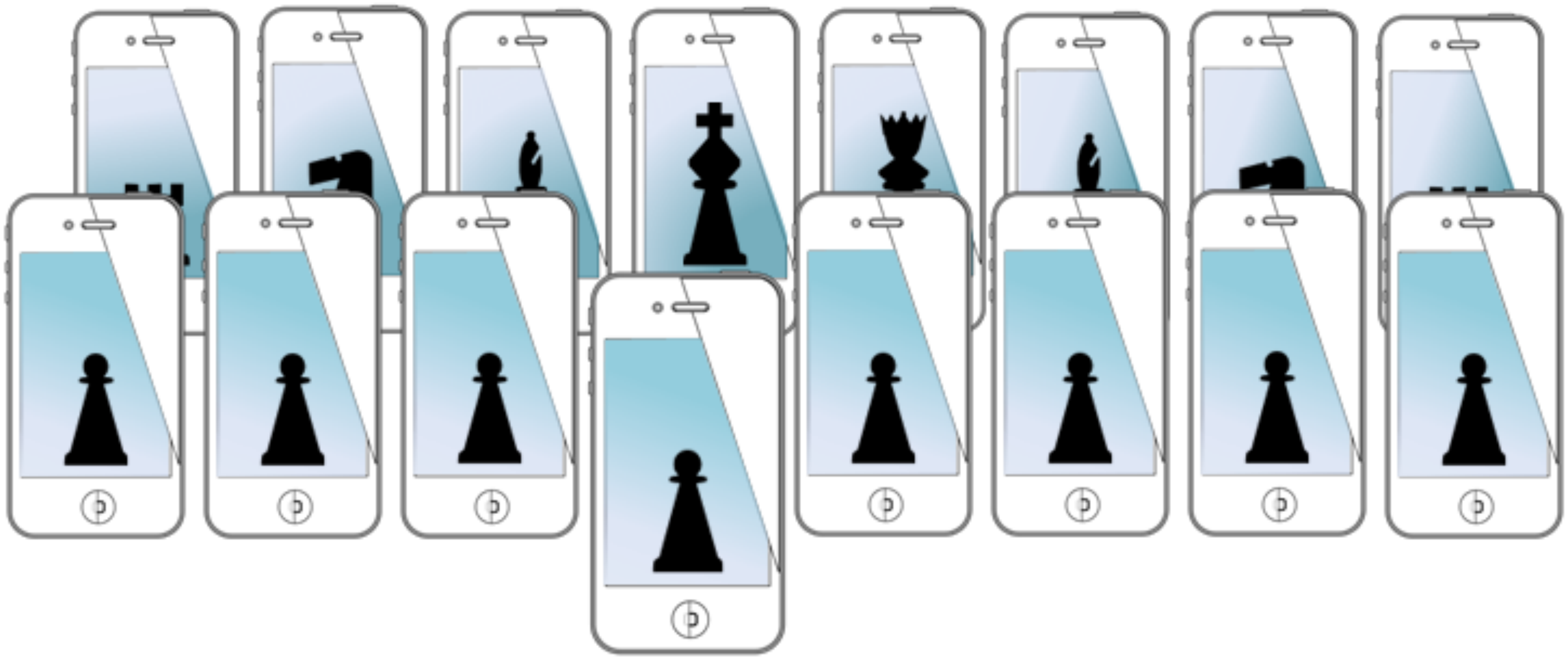
CS5323 & 7323

Mobile Sensing and Learning

activity, pedometers, and motion sensing

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

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

Supplemental Slides: vector trajectory and profiling

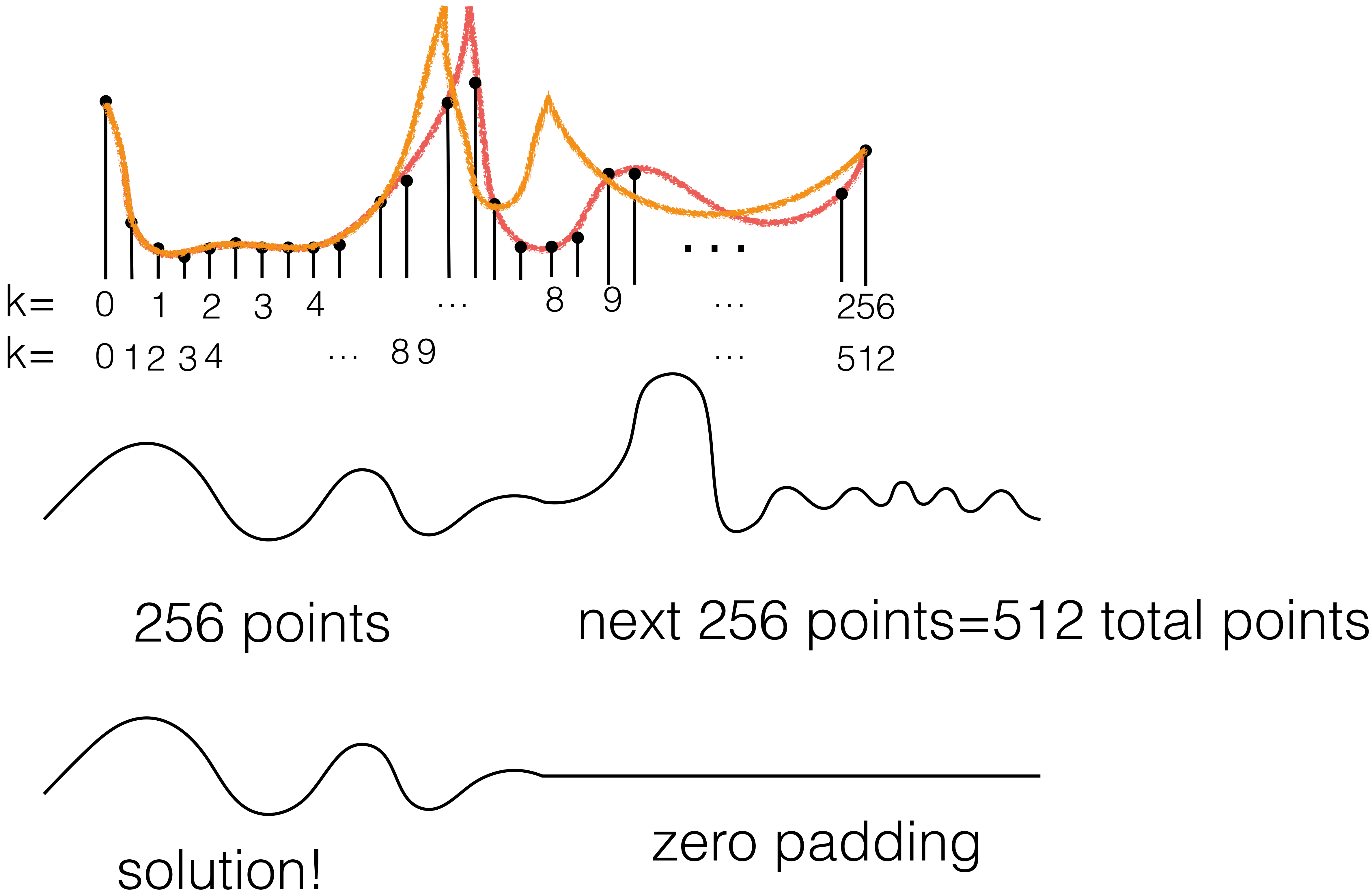
Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University

supplemental slides

- vector trajectory

resolution for the FFT

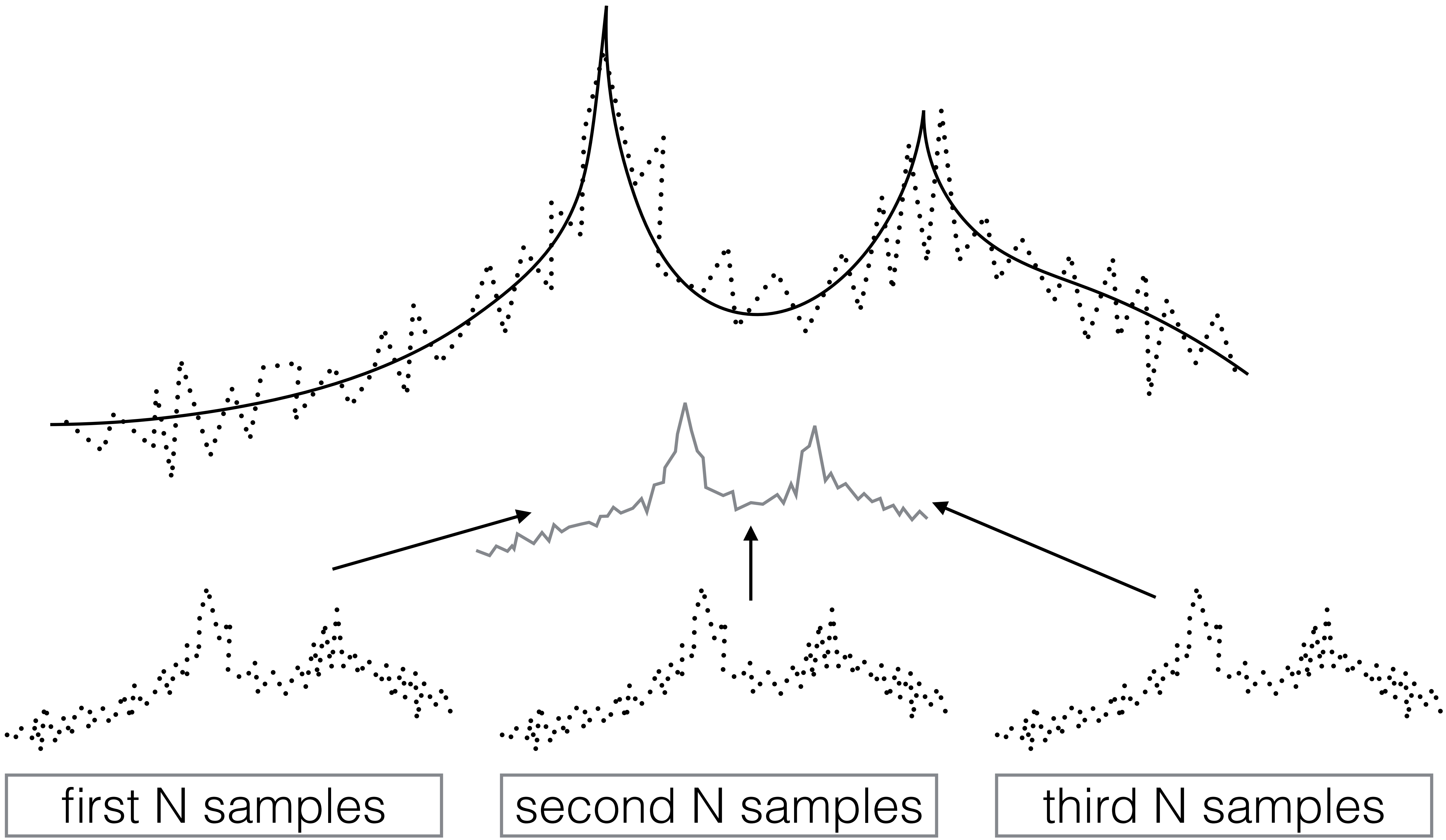
optional



noise in the FFT

optional

- variance around actual magnitude unavoidable



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}
```

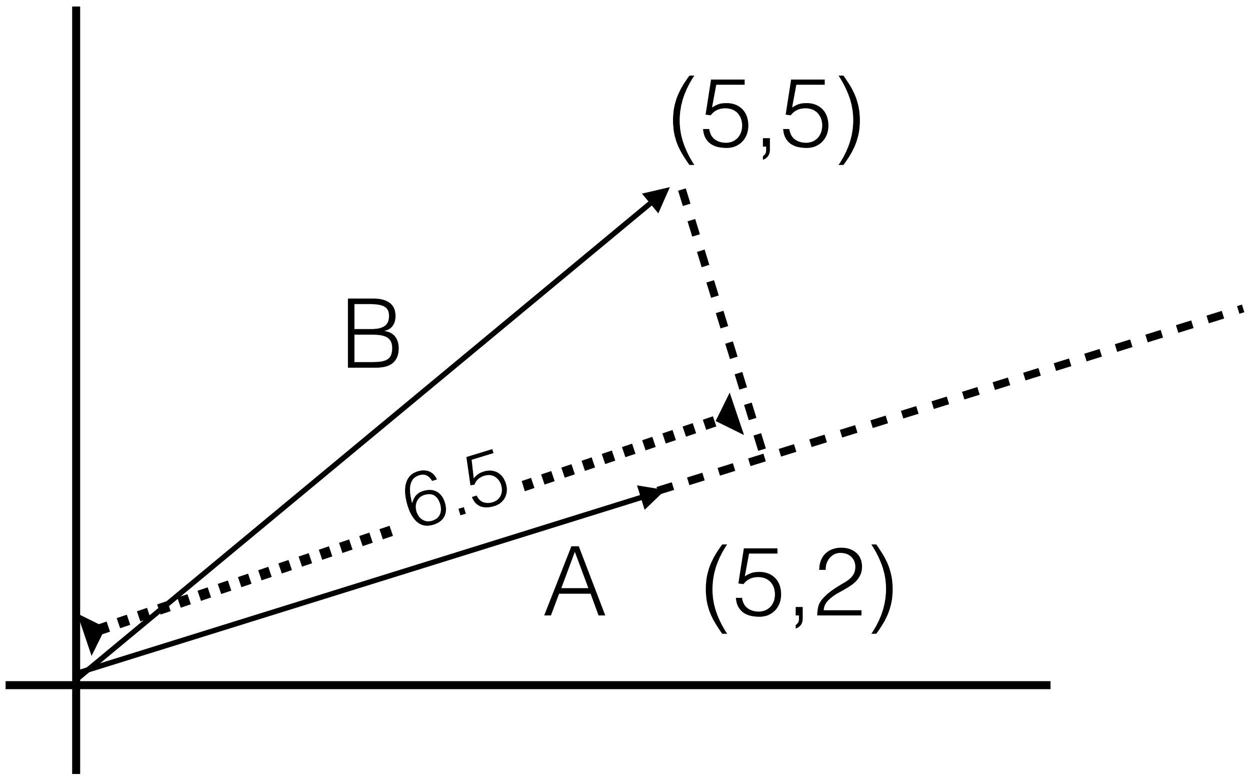
vectors

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