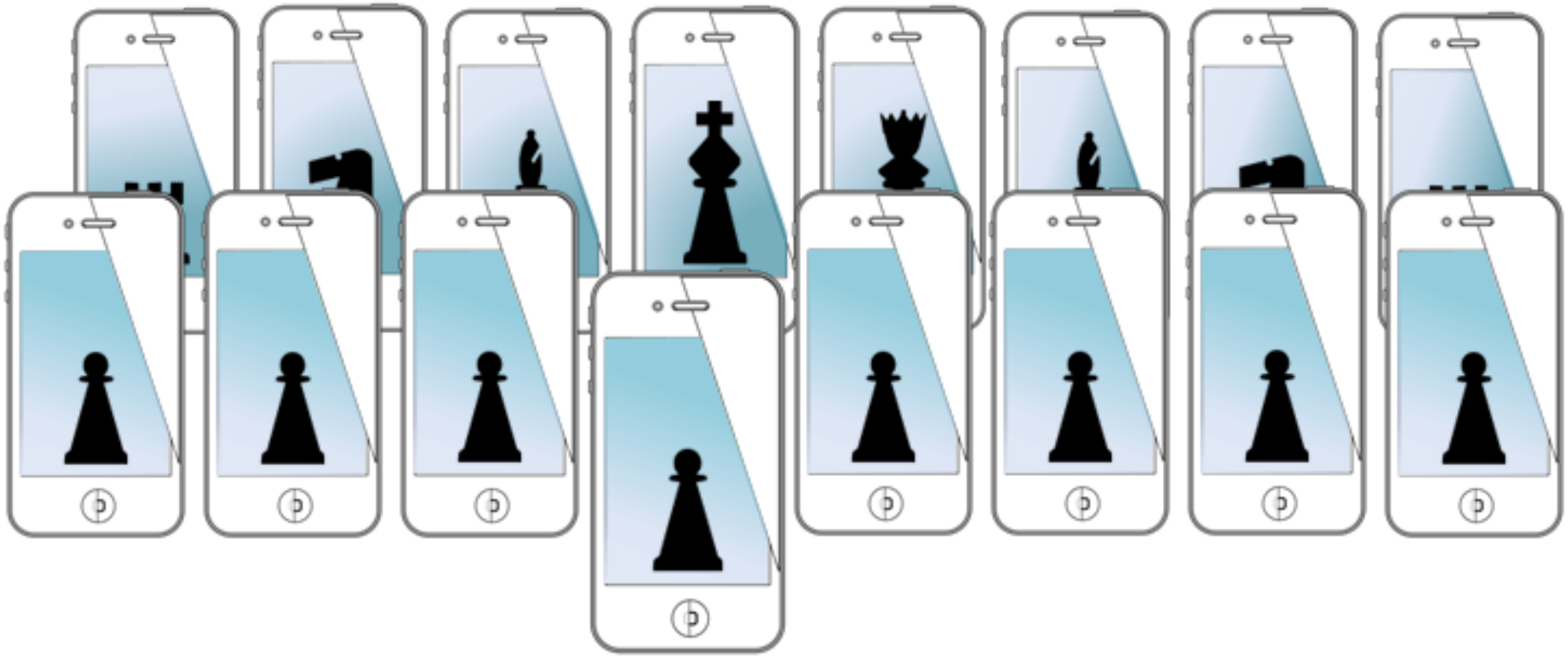


CS5323 & 7323

Mobile Sensing and Learning

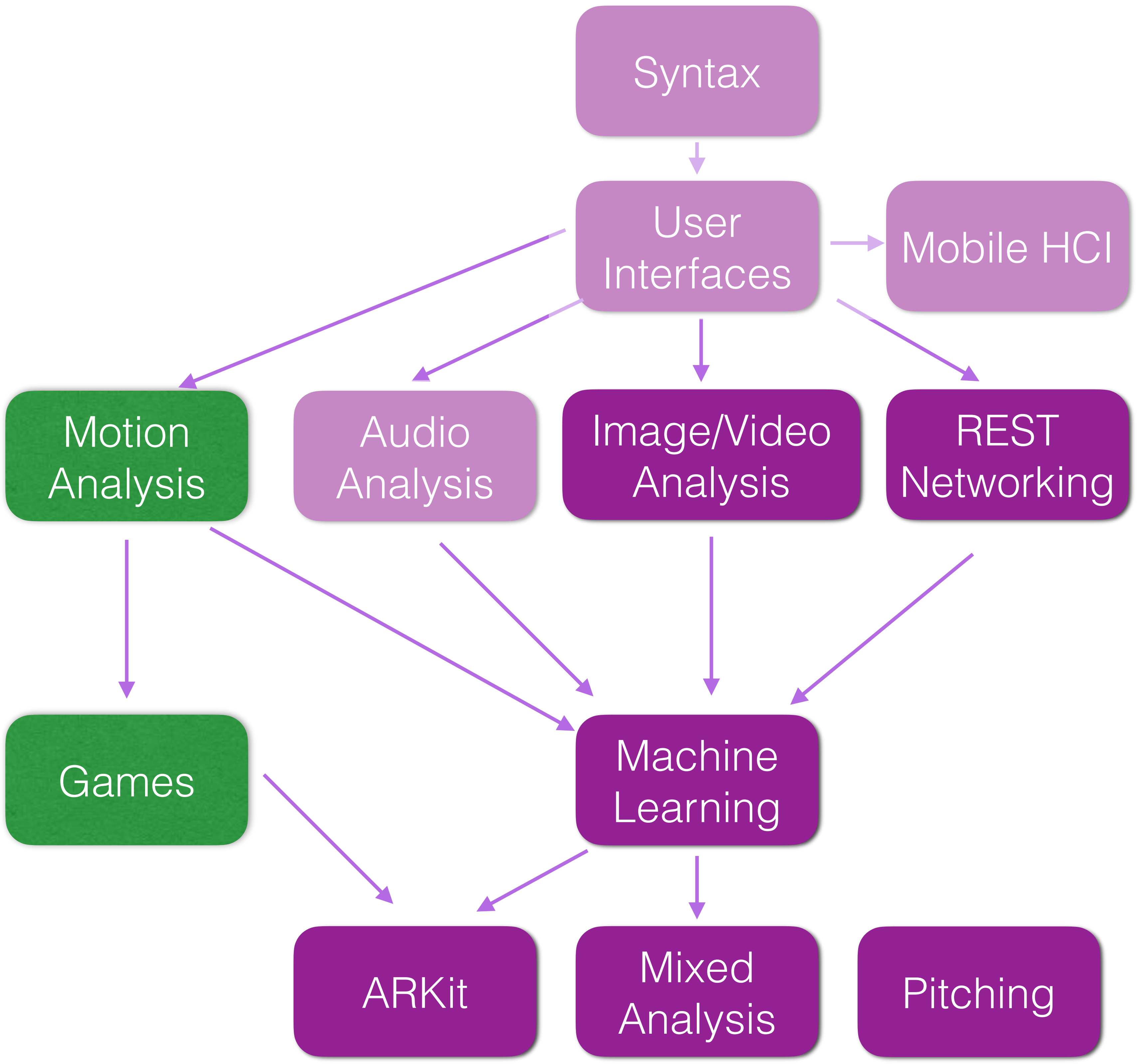
activity, pedometers, and motion sensing

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

logistics and agenda

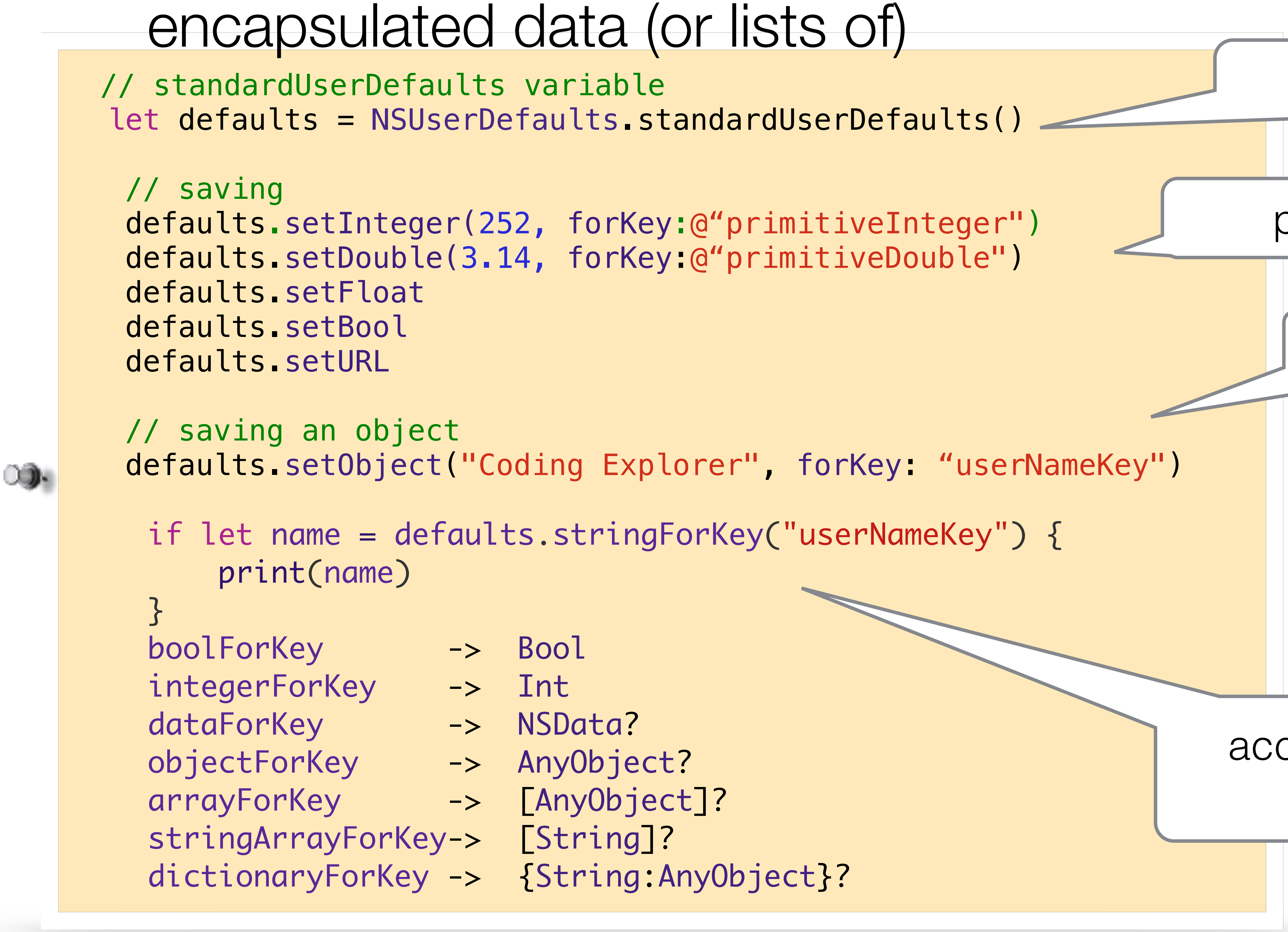
- Logistics:
  - A2 due at end of week
- agenda:
  - core motion (continued)
    - M- co-processor
    - demo
  - accelerometers, gyros, and magnetometers
  - SpriteKit
  - SceneKit

class overview



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}?
```

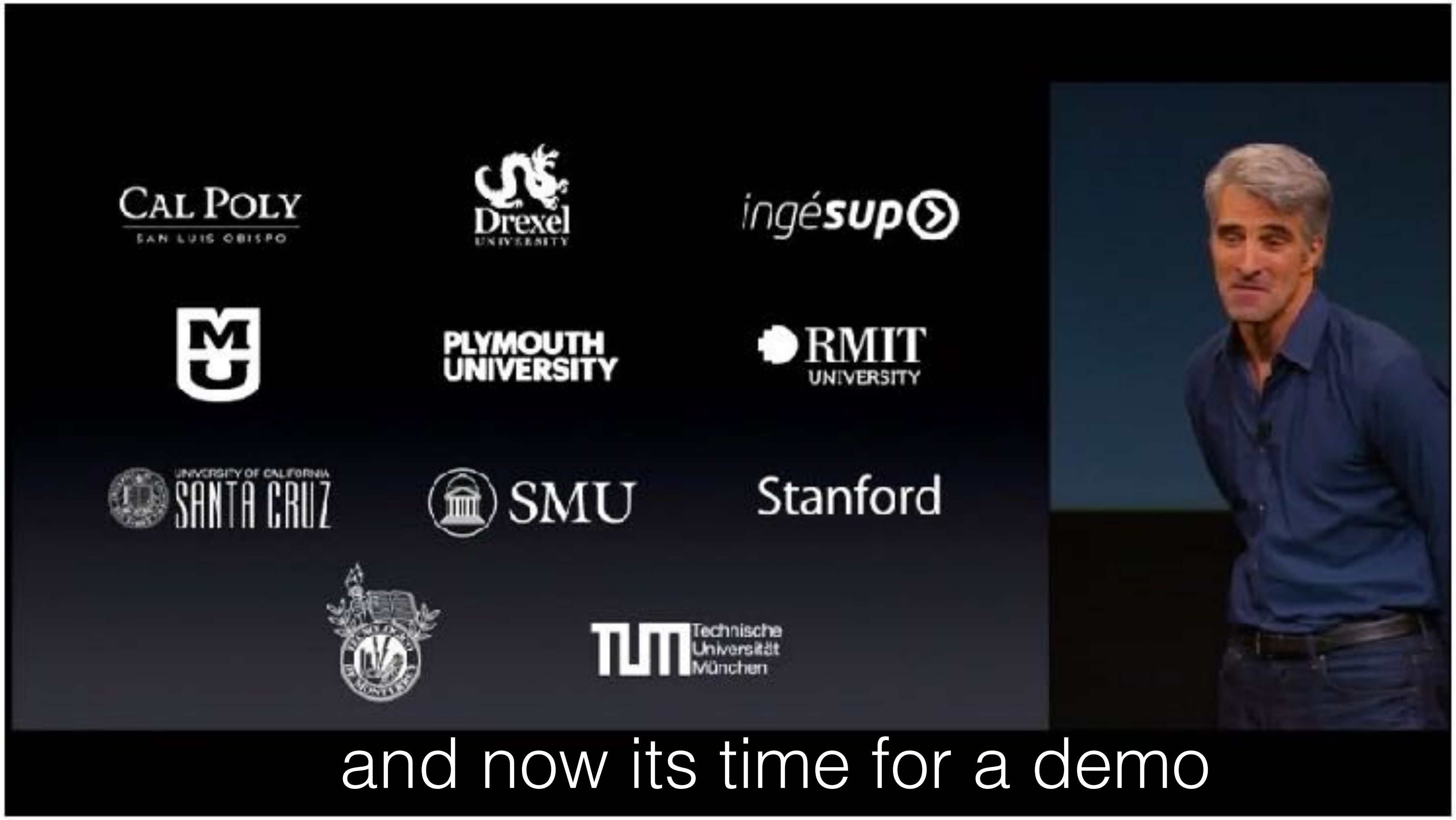
import defaults

primitives

objects

access saved objects

M-# pedometer/activity demo



and now its time for a 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

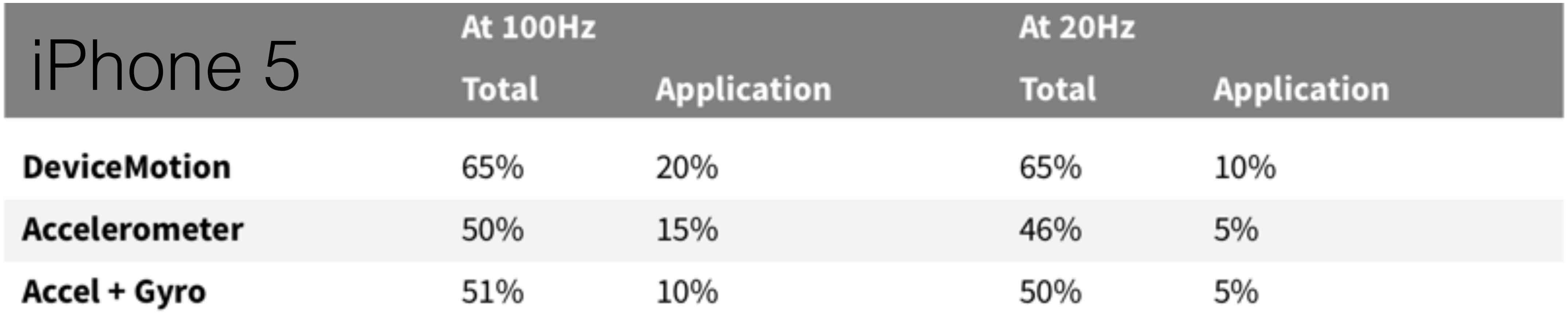
The accelerometer can measure your distance for walking and running. And by using GPS to calibrate for your running stride, the sensor more accurately captures your movement.

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

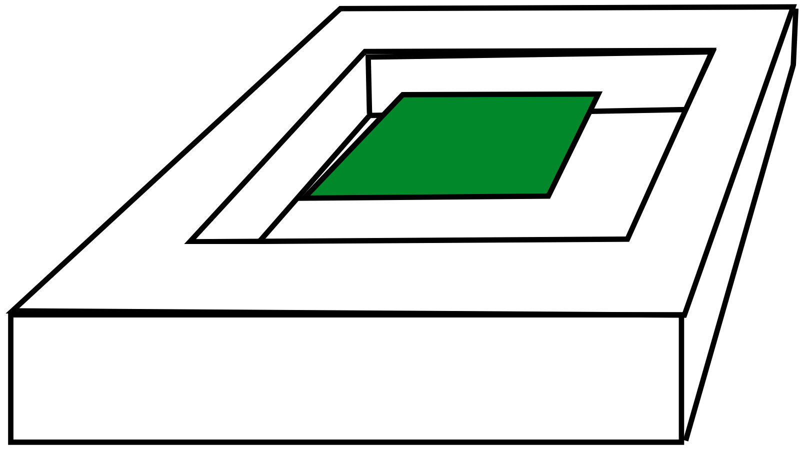


iPhone 5	At 100Hz		At 20Hz	
	Total	Application	Total	Application
DeviceMotion	65%	20%	65%	10%
Accelerometer	50%	15%	46%	5%
Accel + Gyro	51%	10%	50%	5%

iPhone 5s	4%	1%
iPhone 6, 6S	~2%	1%
iPhone 7	~?%	?%

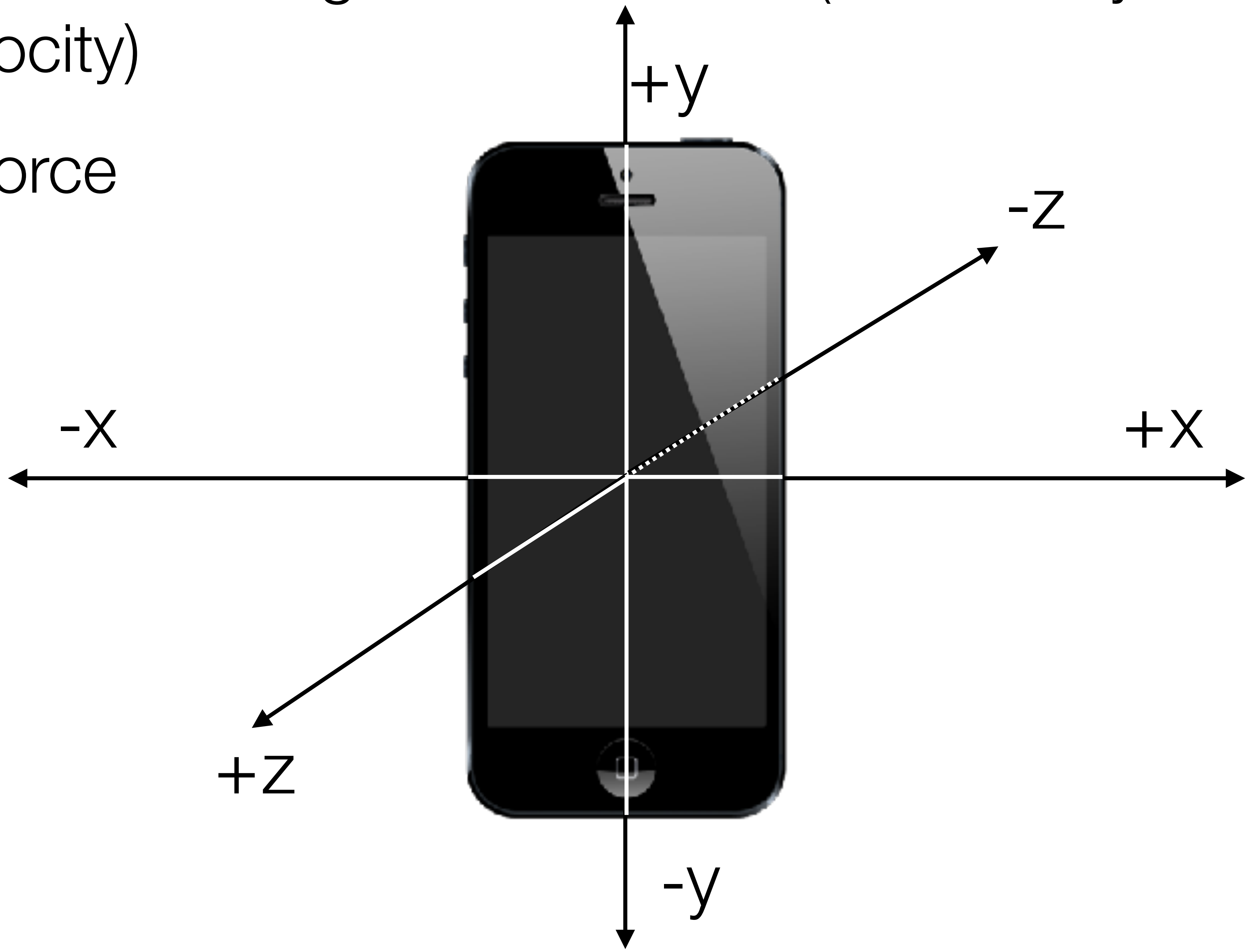
accelerometers

- 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



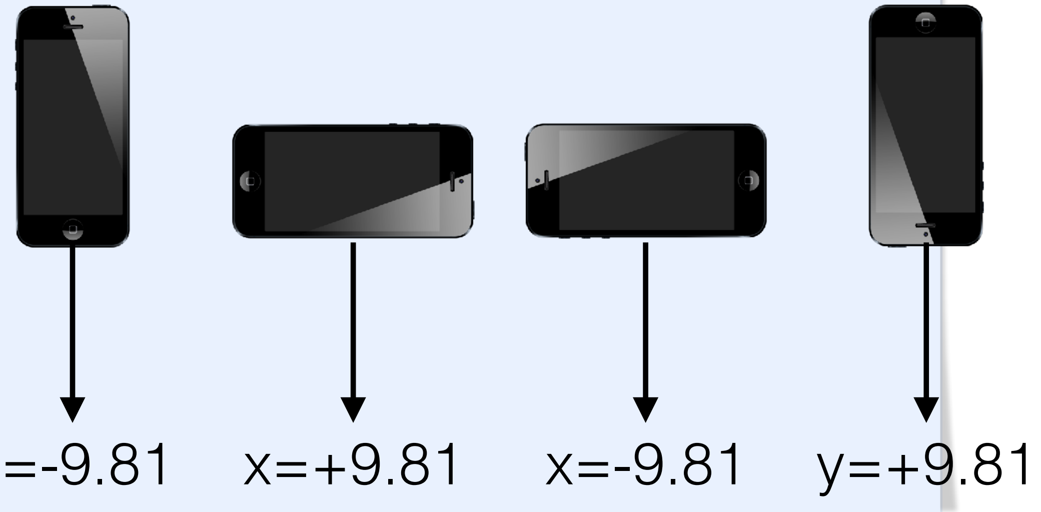
accessing the accelerometer

- 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

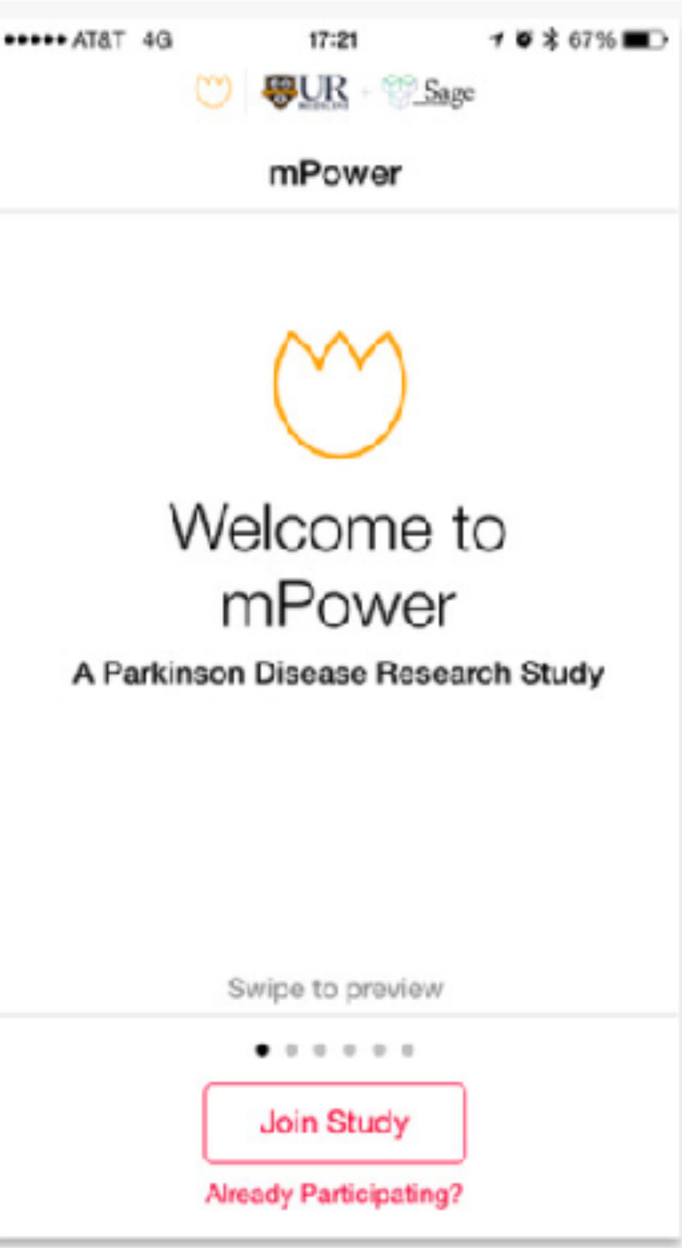
```
CMDeviceMotion *deviceMotion

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!

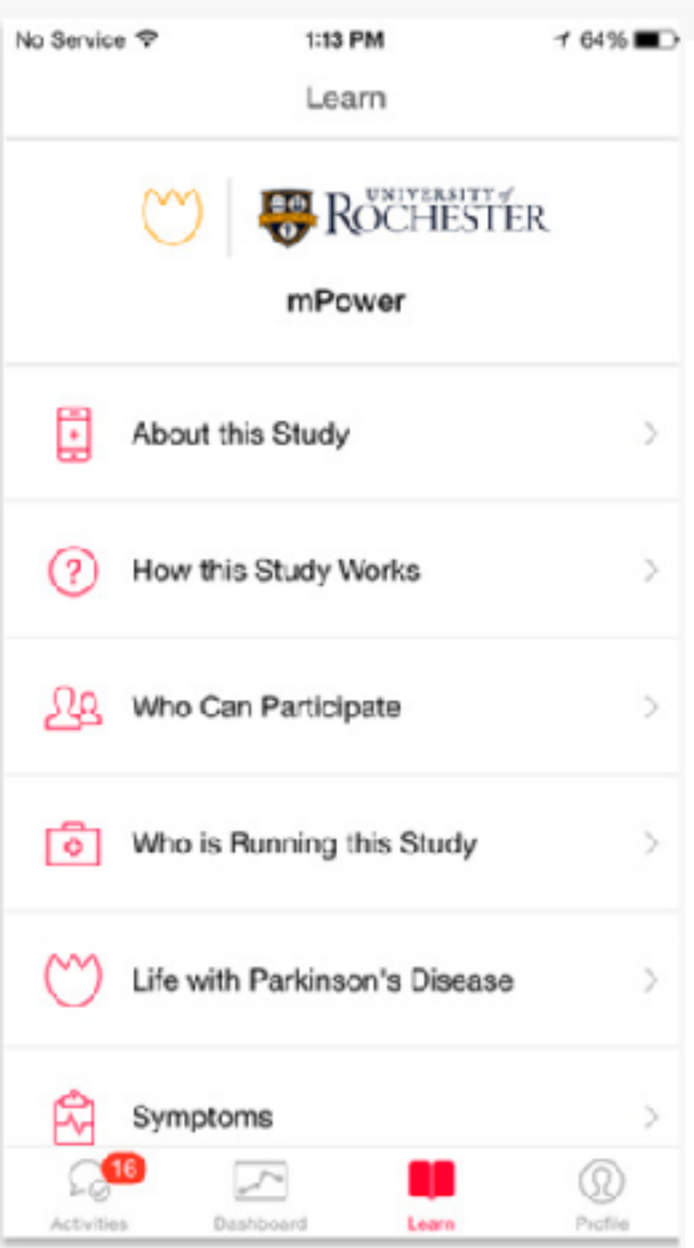


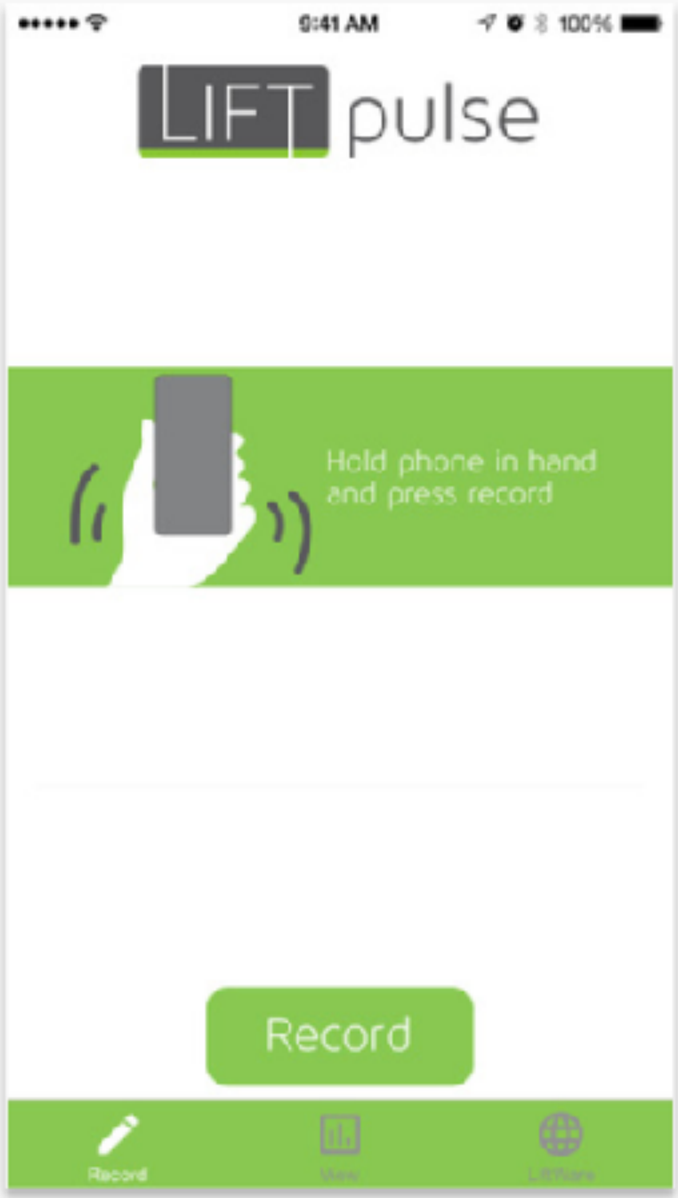
a cool example



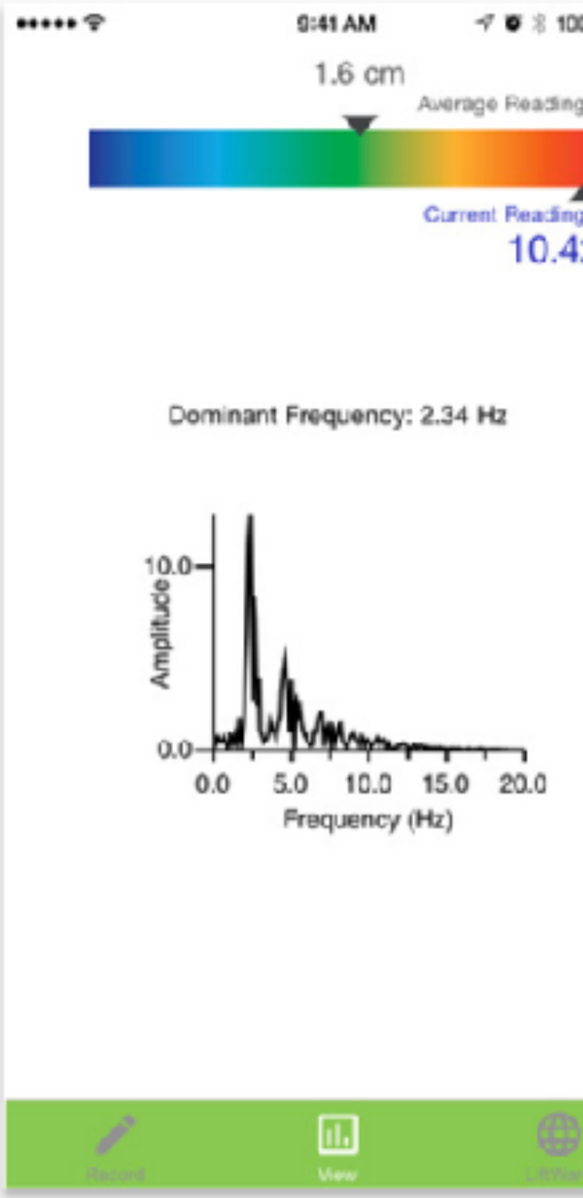
Welcome to mPower

A Parkinson Disease Research Study





Hold phone in hand and press record



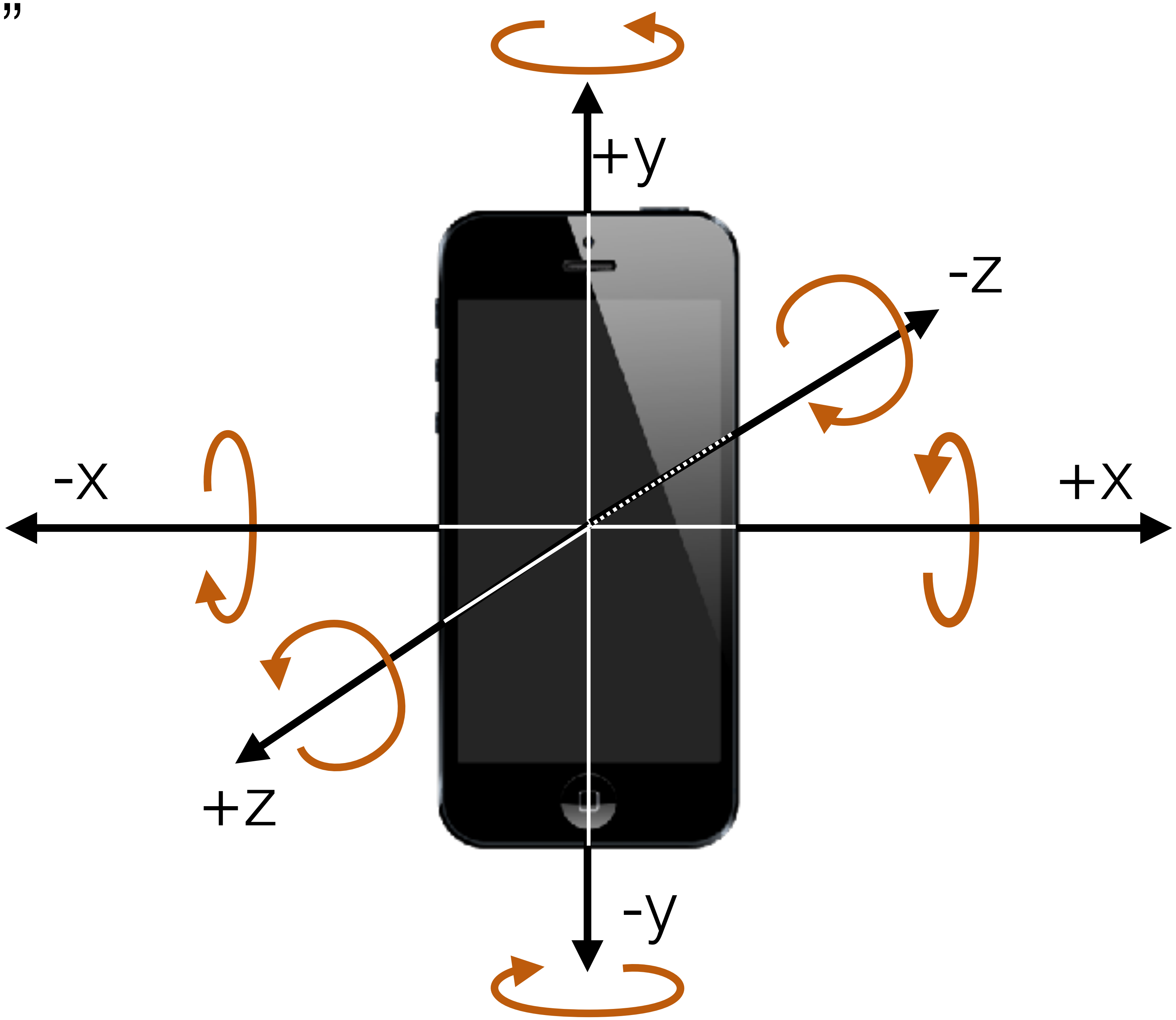
gyroscope

- 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

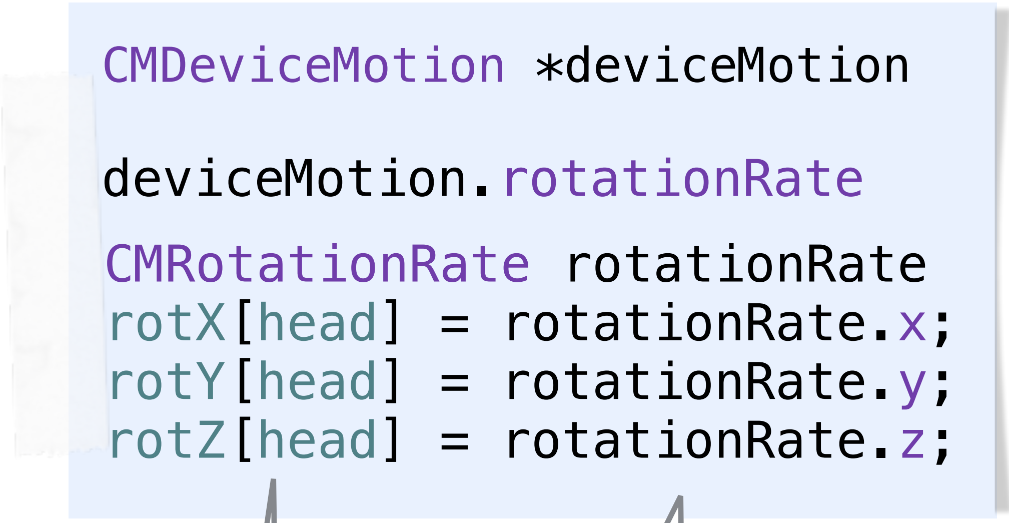
gyroscope

- the “right hand rule”



accessing the gyro

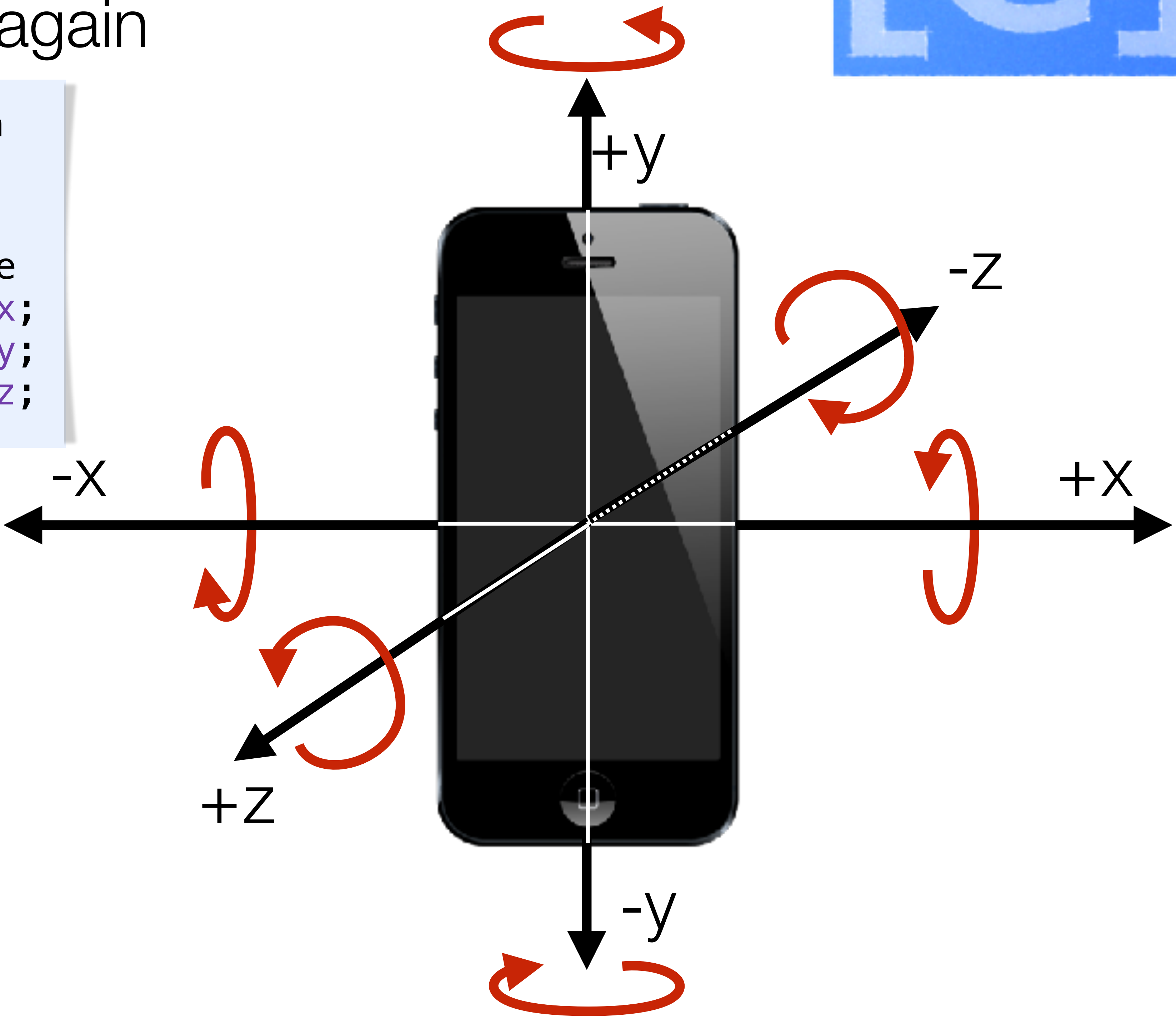
- use device motion again



```
CMDeviceMotion *deviceMotion

deviceMotion.rotationRate

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

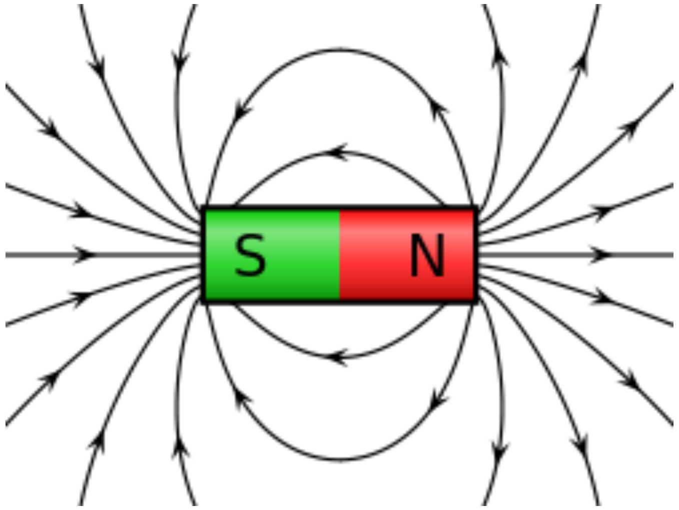
measures **rate** of motion

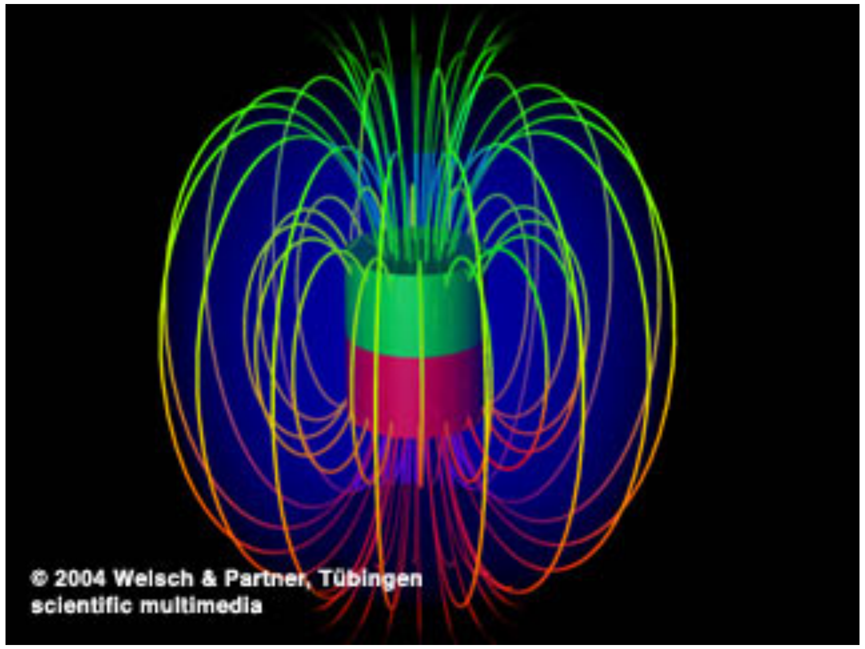
in this example, saves it to array



magnetometers

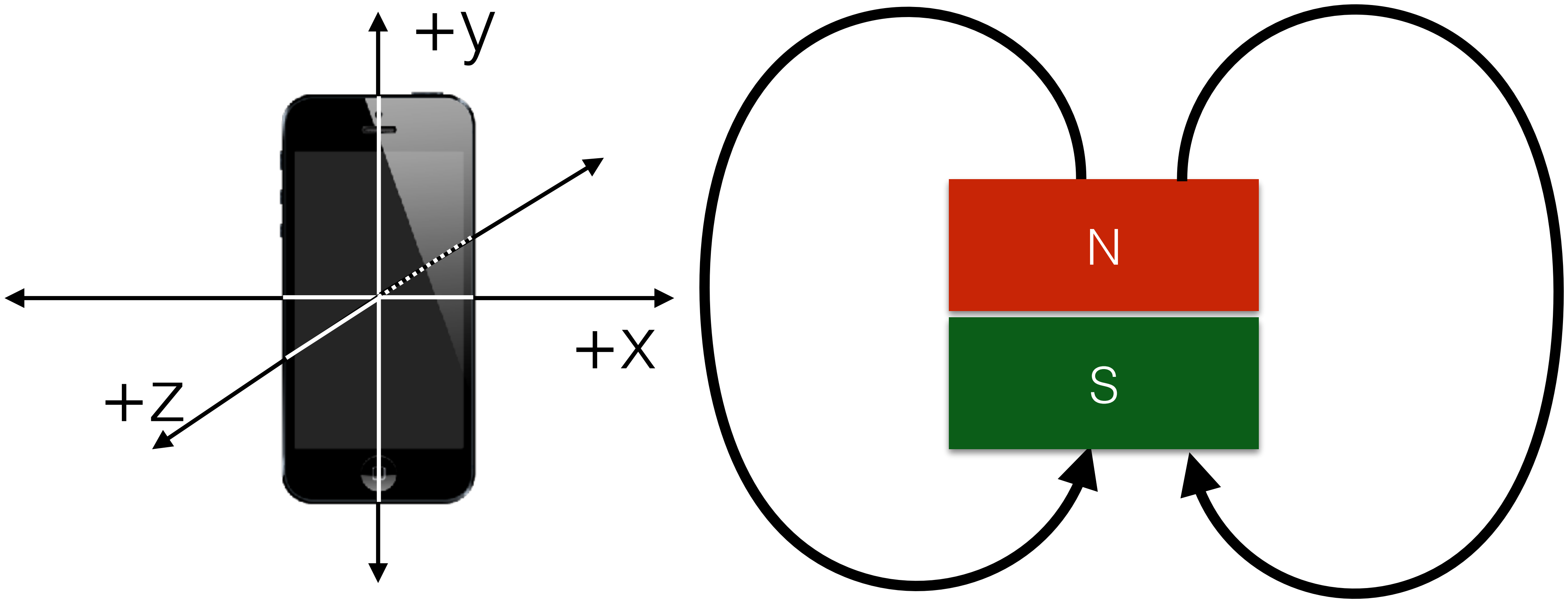
- measure magnetic fields
- magnets are measured in tesla (T)
  - **how**: essentially, there is a tight coupling between electricity flow and magnetic fields
- earth's magnetic field varies, but is around 50 uT
- iPhone can measure up to 1T with a resolution of about 8uT
- magnetic fields have direction!





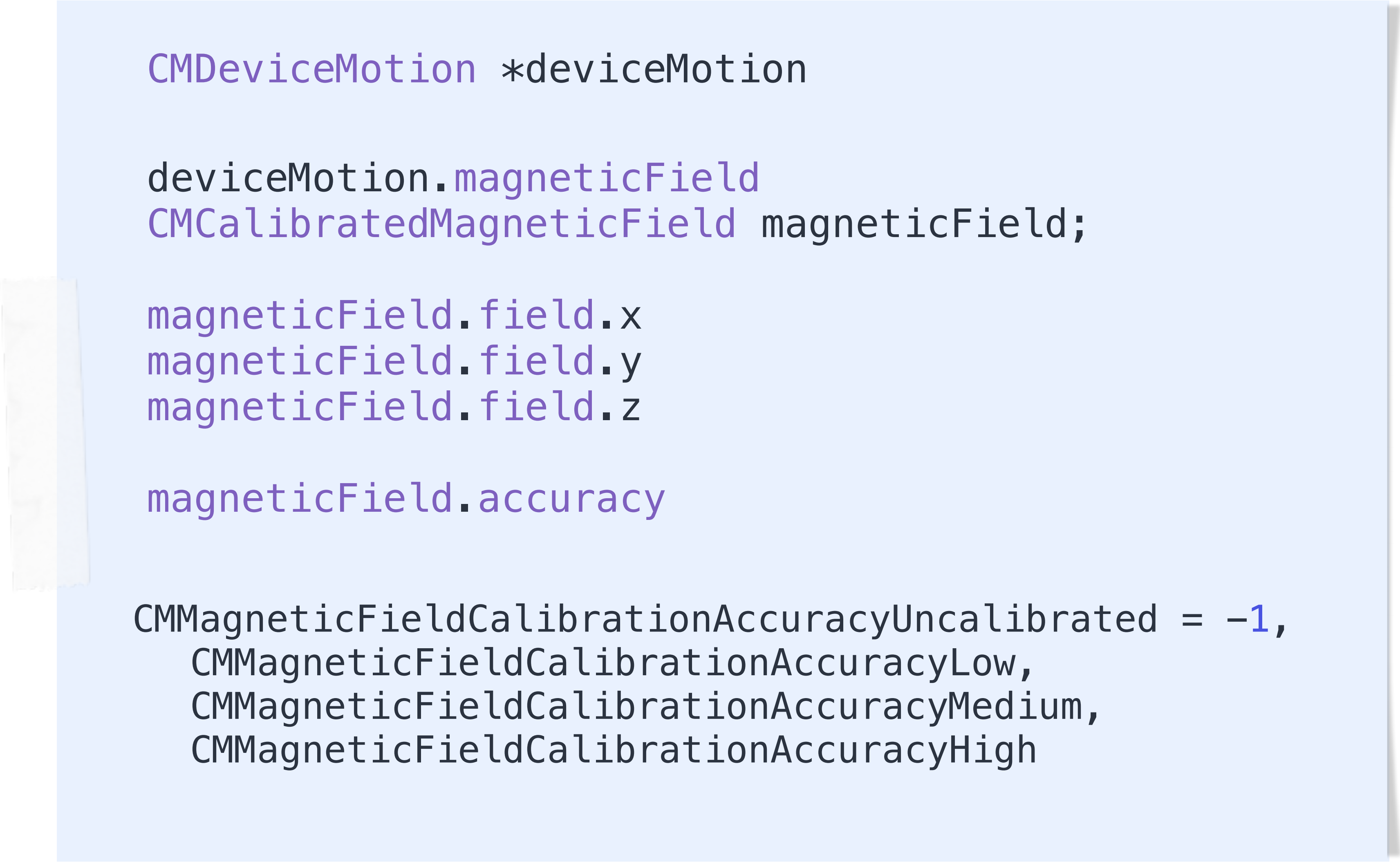
magnetic fields

- measure magnetic field along axis, towards “south”



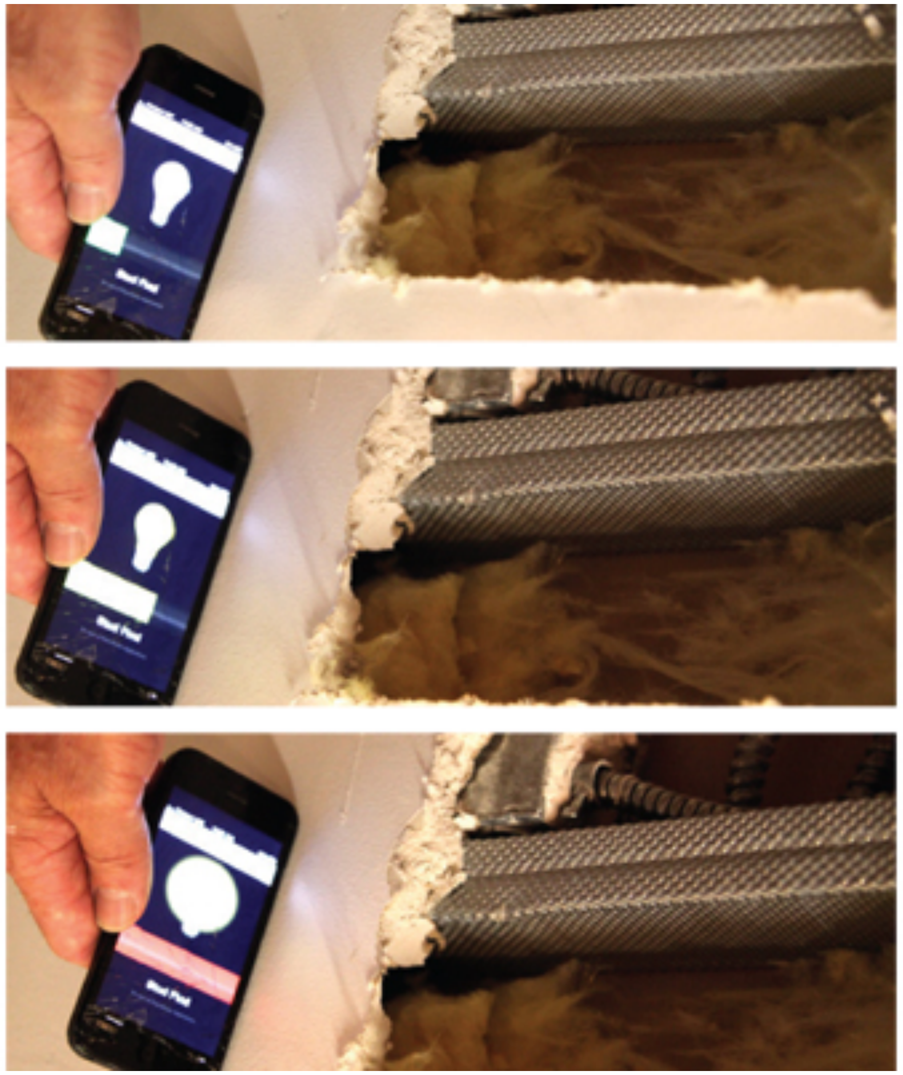
but iPhone has magnetic bias

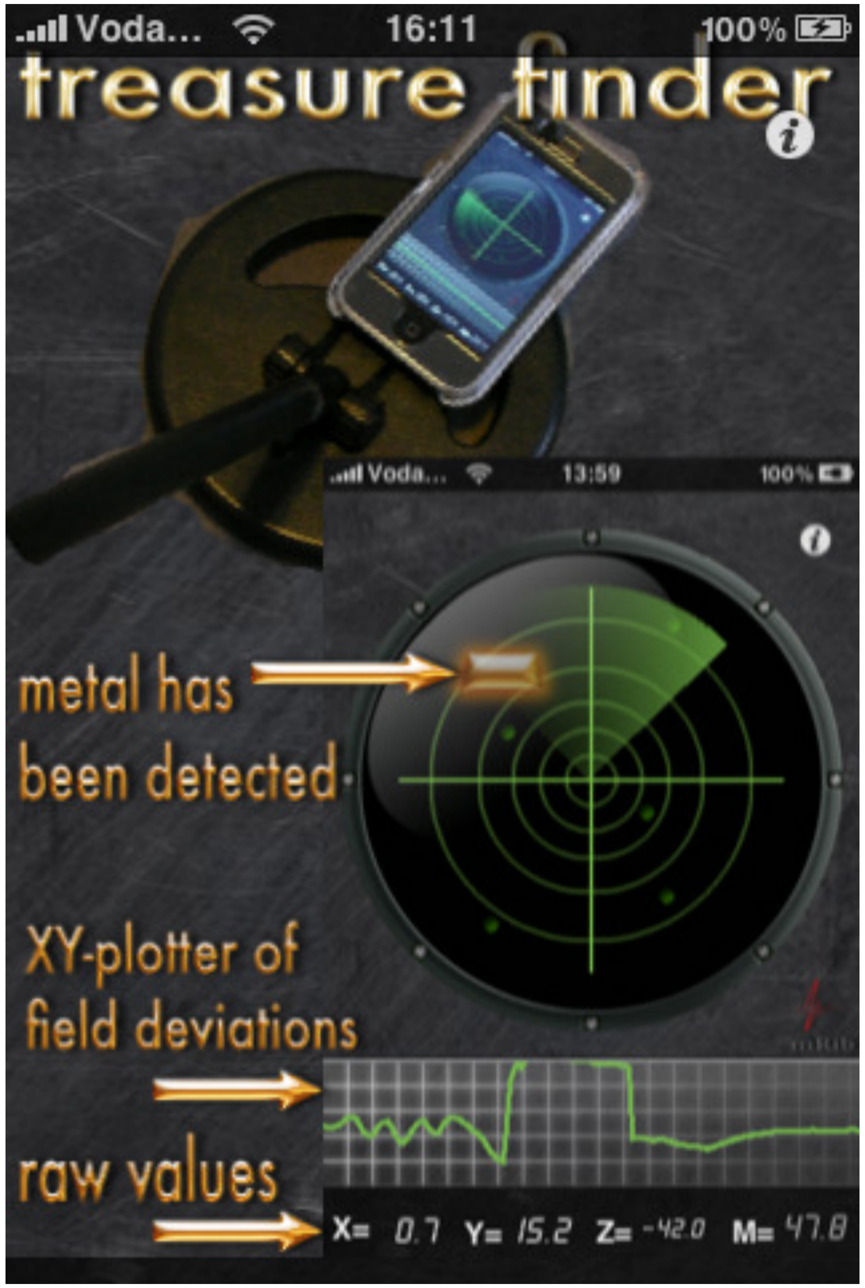
- the phone uses electricity and therefore is a magnet
    - good thing Apple subtracts that out for us!



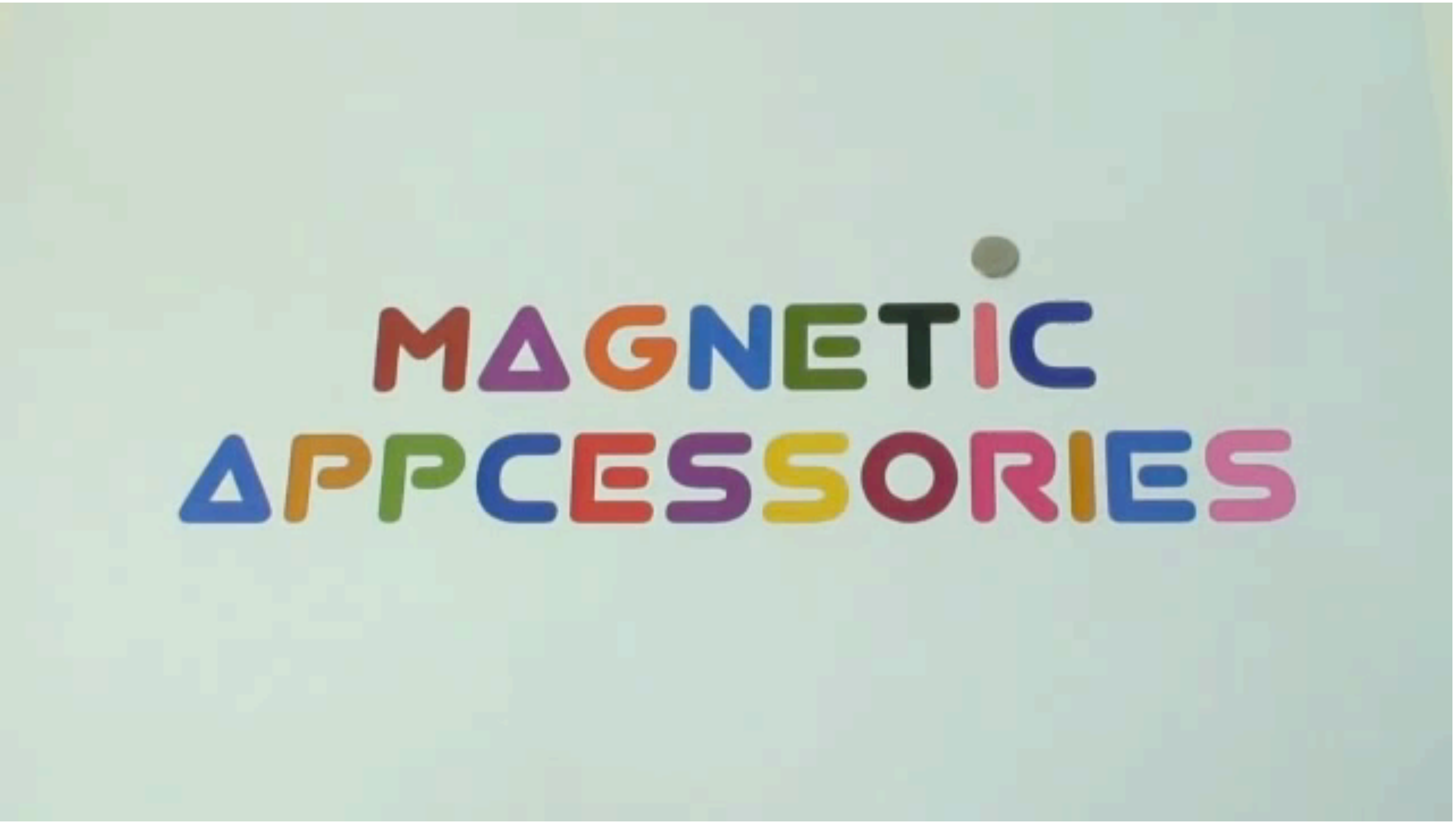
```
CMDeviceMotion *deviceMotion

deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

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

magneticField.accuracy


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

a cool example



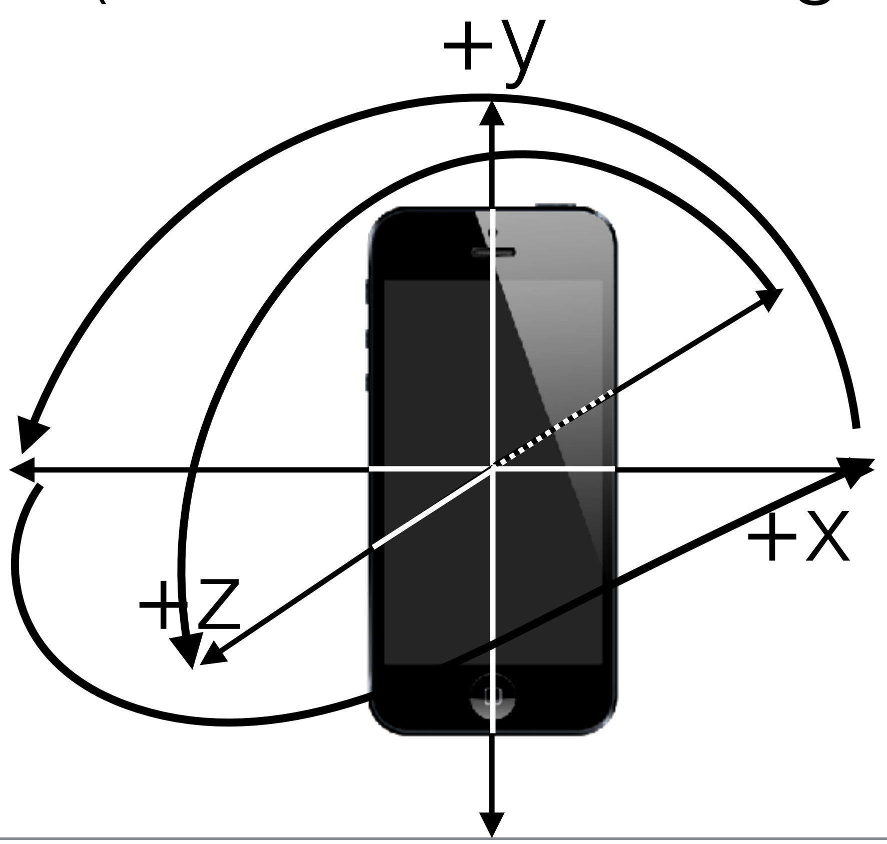


a cool example



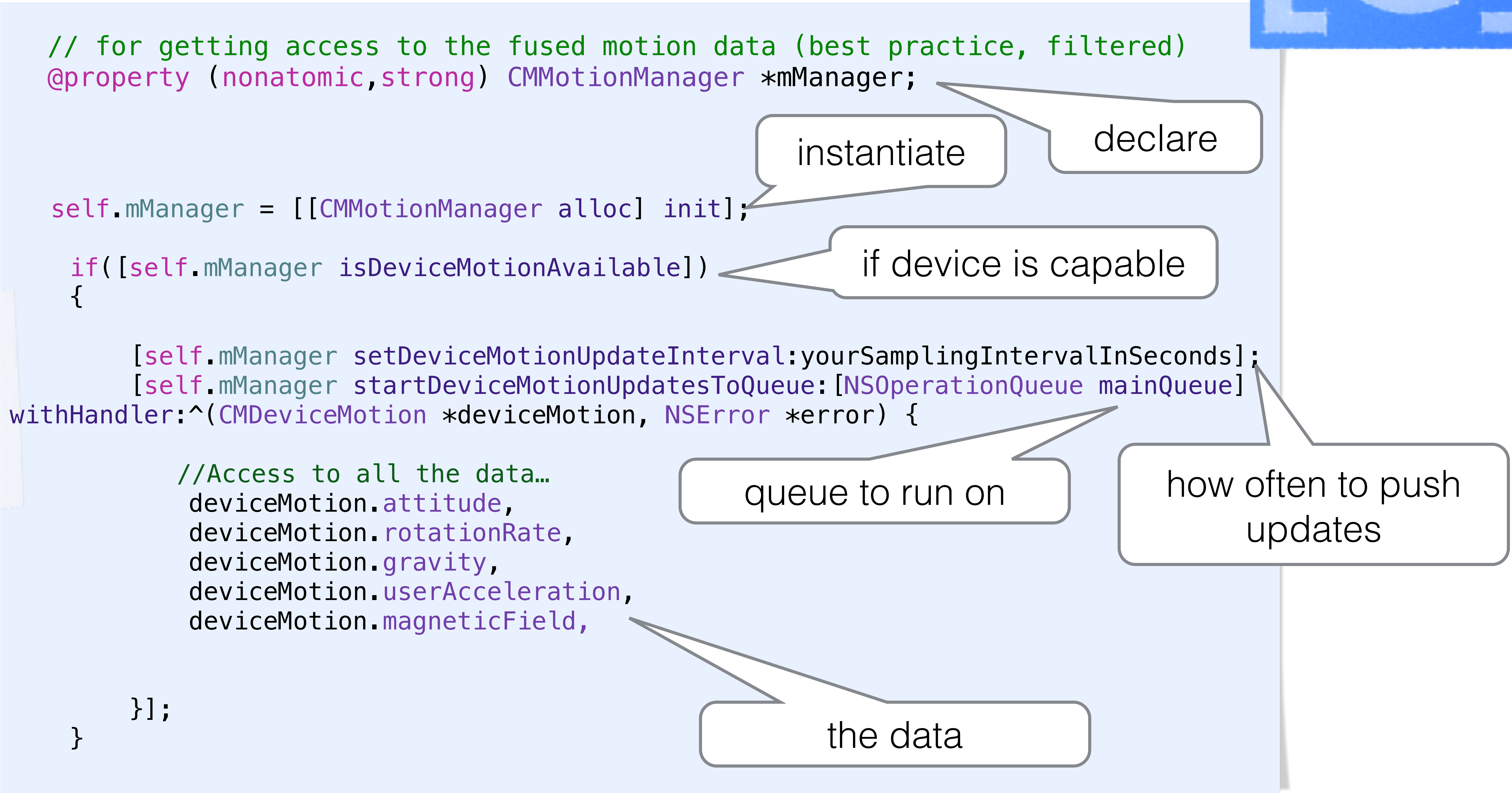
attitude

- 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

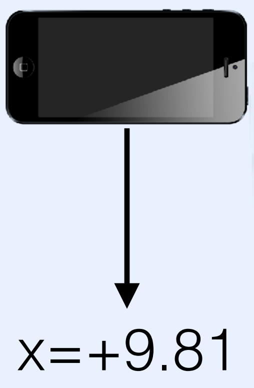
getting updates

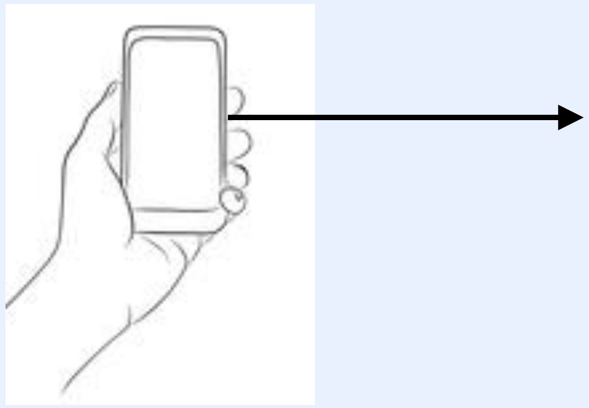


```
  // for getting access to the fused motion data (best practice, filtered)
  @property (nonatomic,strong) CMMotionManager *mManager;



   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,


        }];
    }
```

summary

```
CMDeviceMotion *deviceMotion

deviceMotion.gravity
deviceMotion.userAcceleration

CMAcceleration gravity,
CMAcceleration userAcceleration

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

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

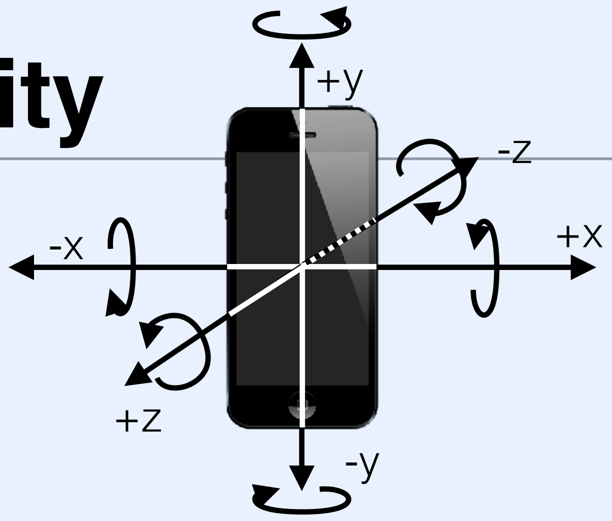




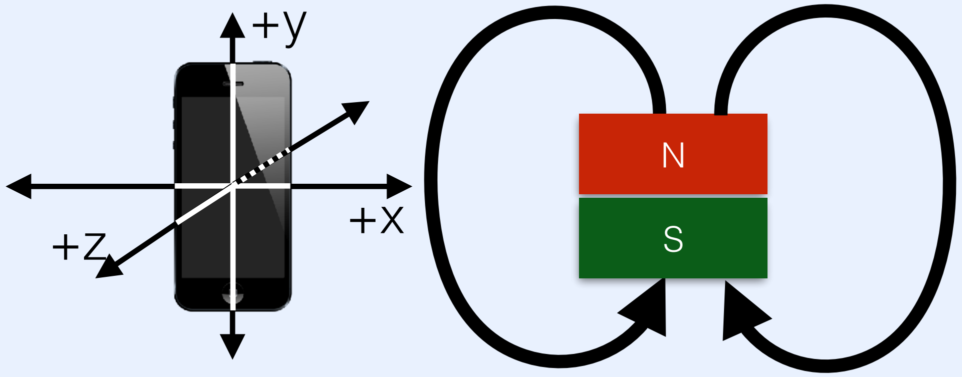
acceleration

rotation velocity

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

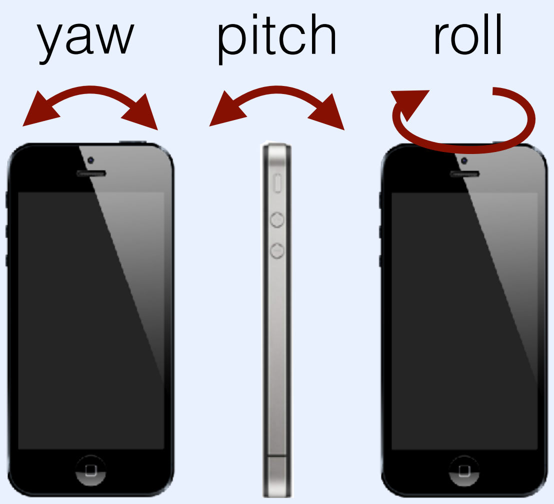


```
deviceMotion.magneticField
CMCalibratedMagneticField magneticField;

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

magneticField.accuracy
```



magnetic field

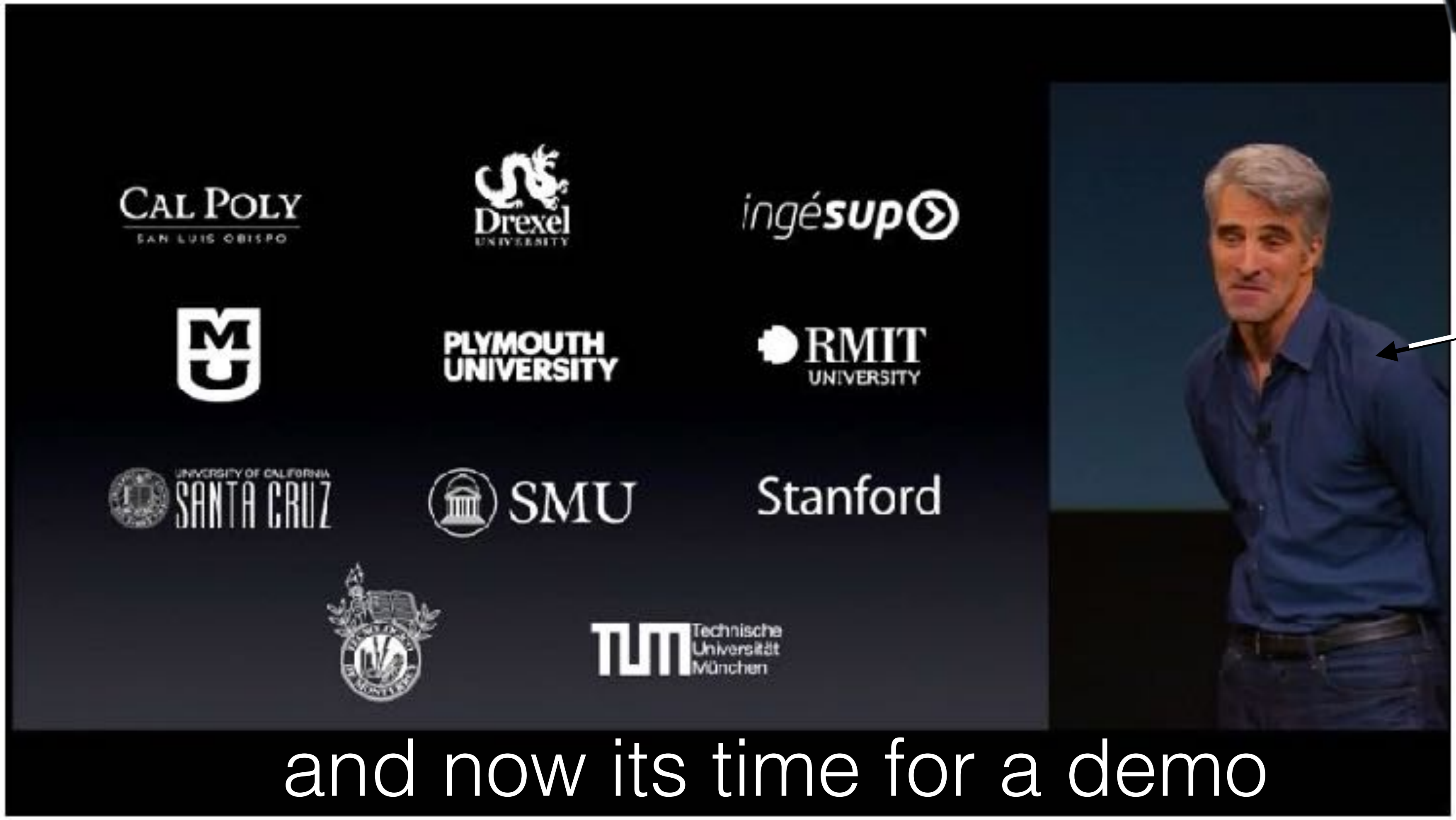
```
deviceMotion.attitude

CMAttitude* attitude

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



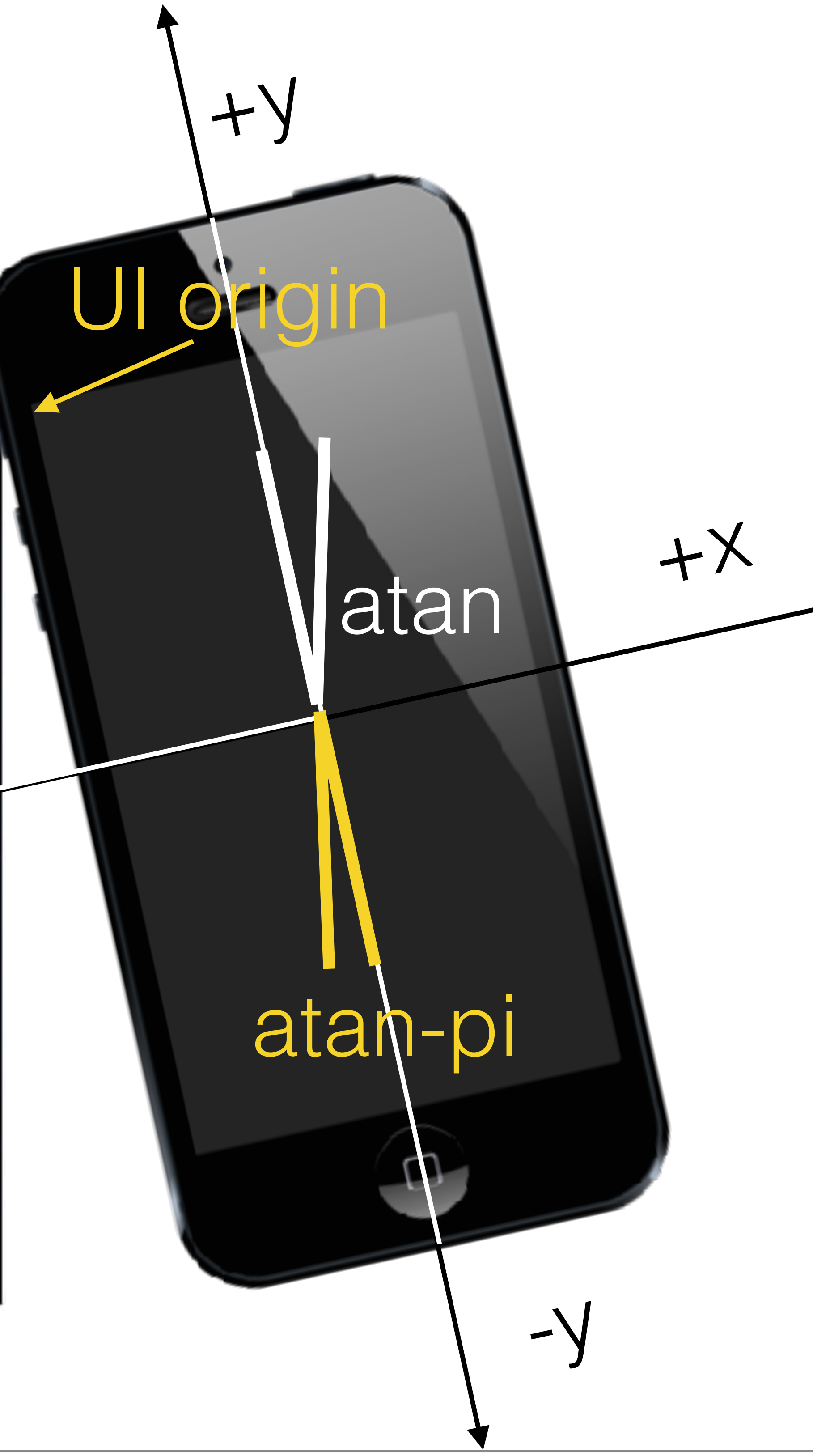
device position

device motion demo

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



and now its time for a demo



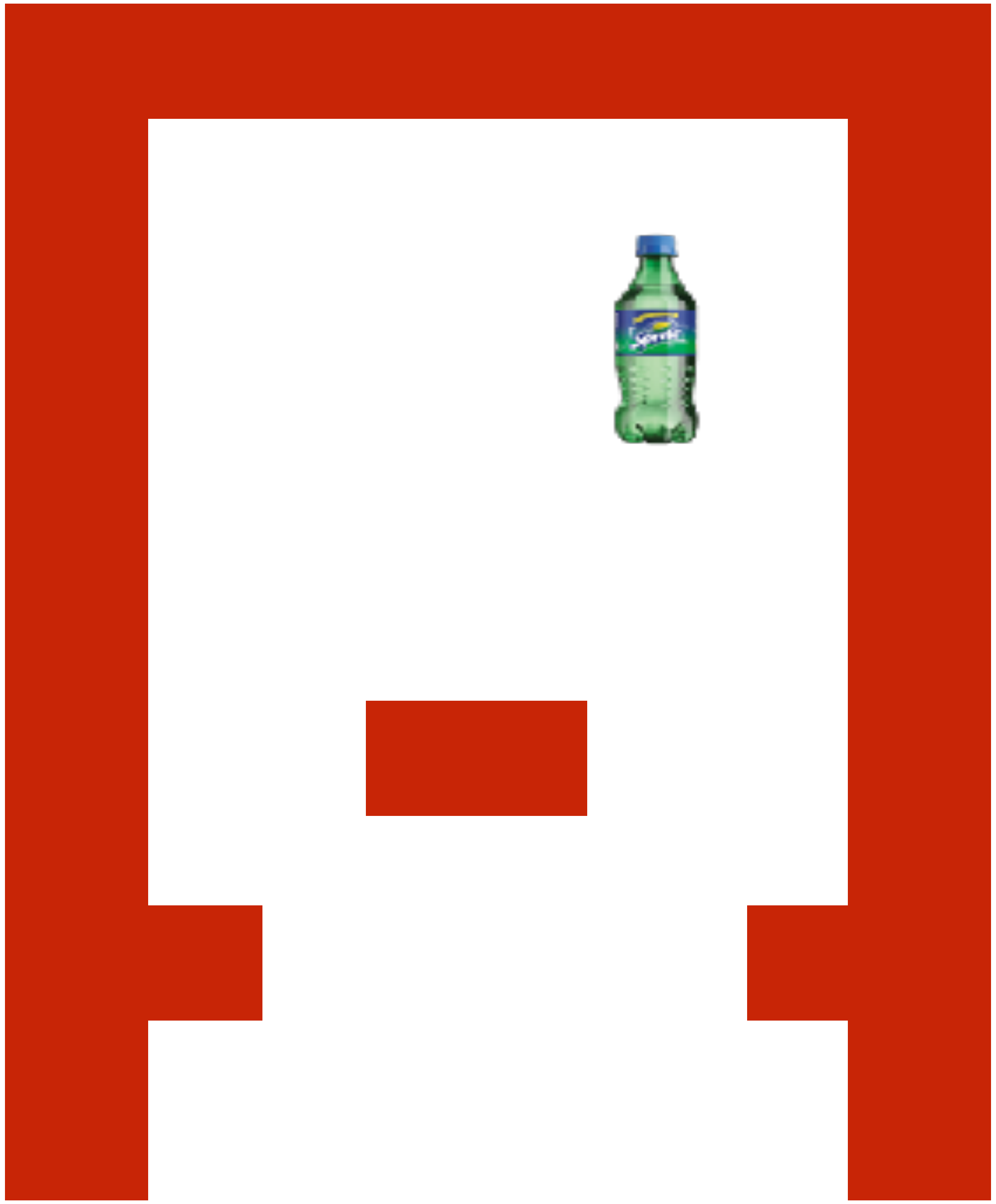
something more?

- 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”

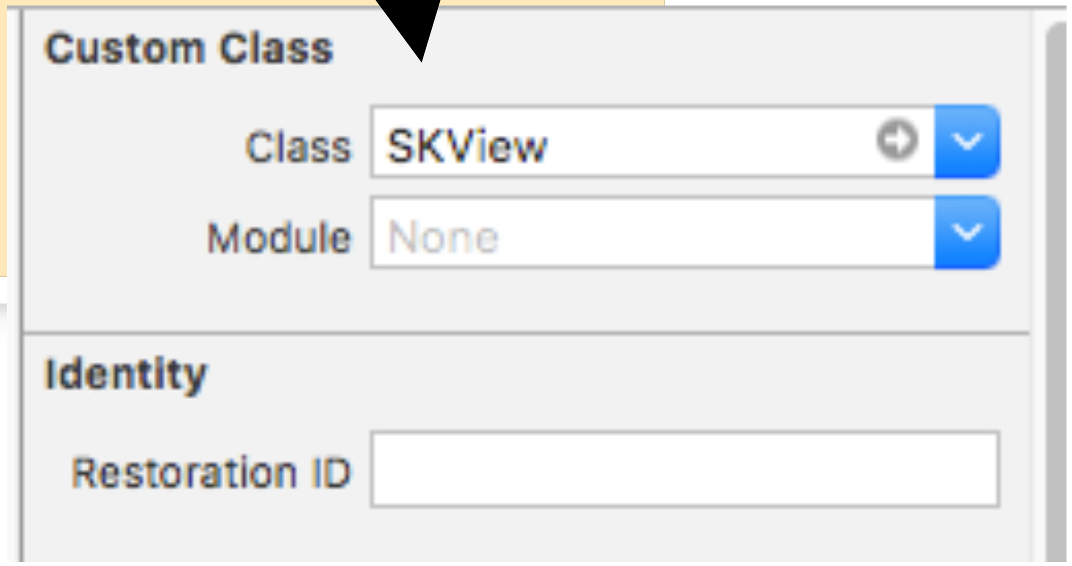
create “bouncy” sprite

make actual gravity
== game gravity

user must move phone
to keep sprite bouncing
on target

setup view controller

```
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)
    }
}
```



Custom Class

Class: SKView

Module: None

Identity

Restoration ID:

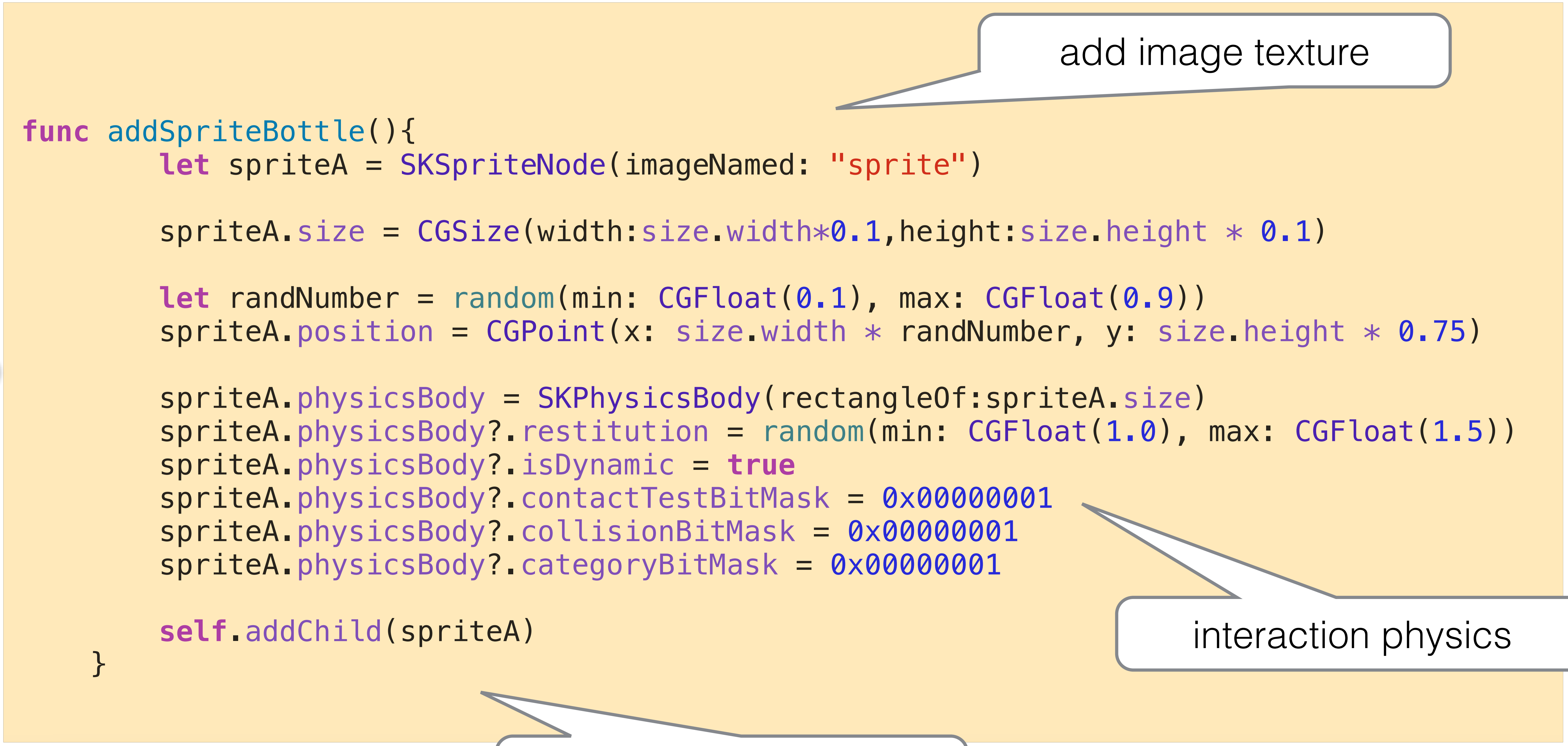
set gravity

```
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)
    }
```

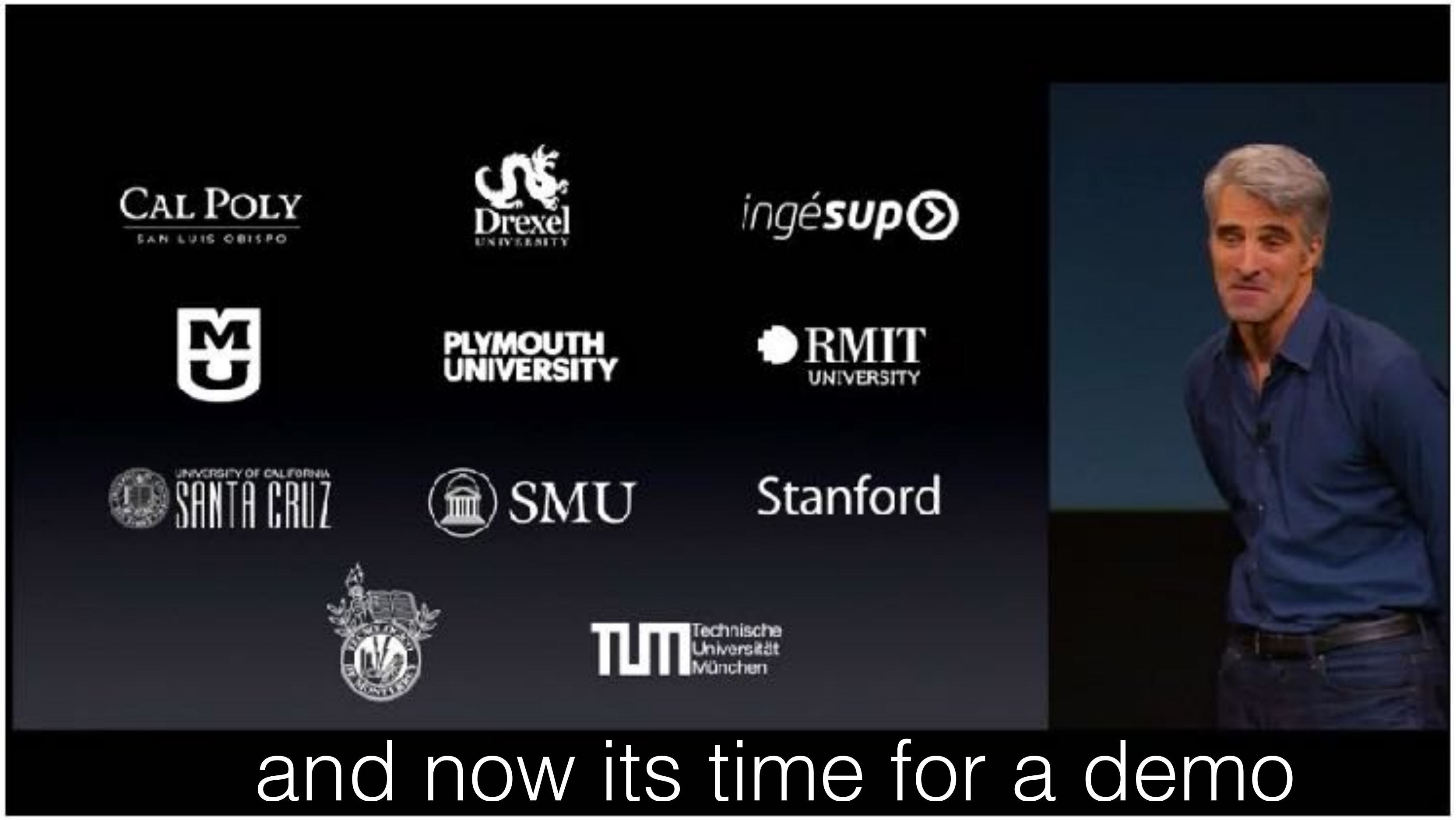
add image texture

interaction physics

add to scene

device motion demo 2

- lemon lime bounce
- pre-made demo

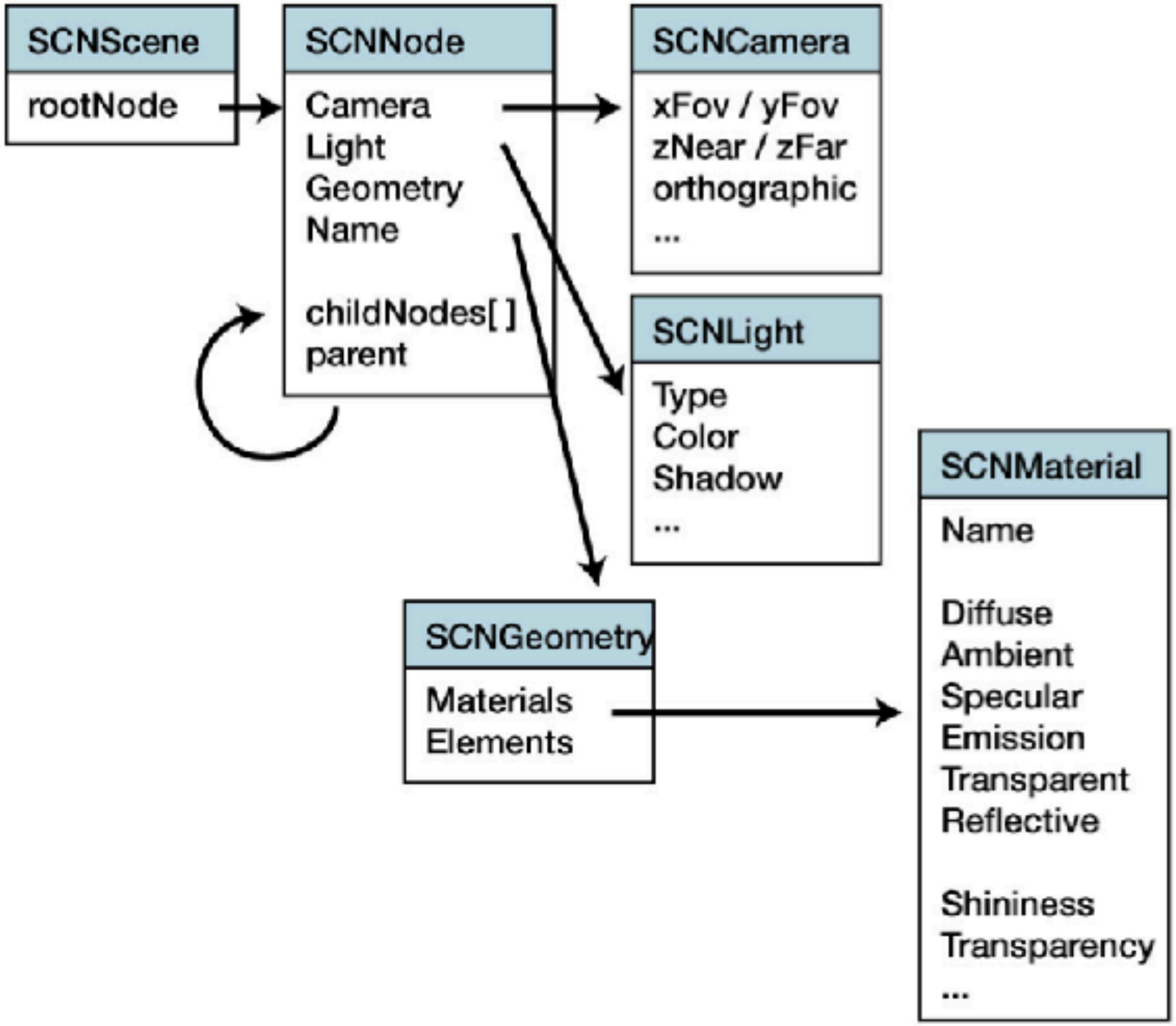


SceneKit: 3D scenes

- SceneKit allows you to create a 3D world and add physics, nodes, lighting, etc.
  - very powerful
- basic workflow:
  - setup world
  - add nodes

work flow in 3D scenes

- \



SCNScene
rootNode

SCNNode
Camera
Light
Geometry
Name

childNodes[ ]
parent

SCNCamera
xFov / yFov
zNear / zFar
orthographic
...

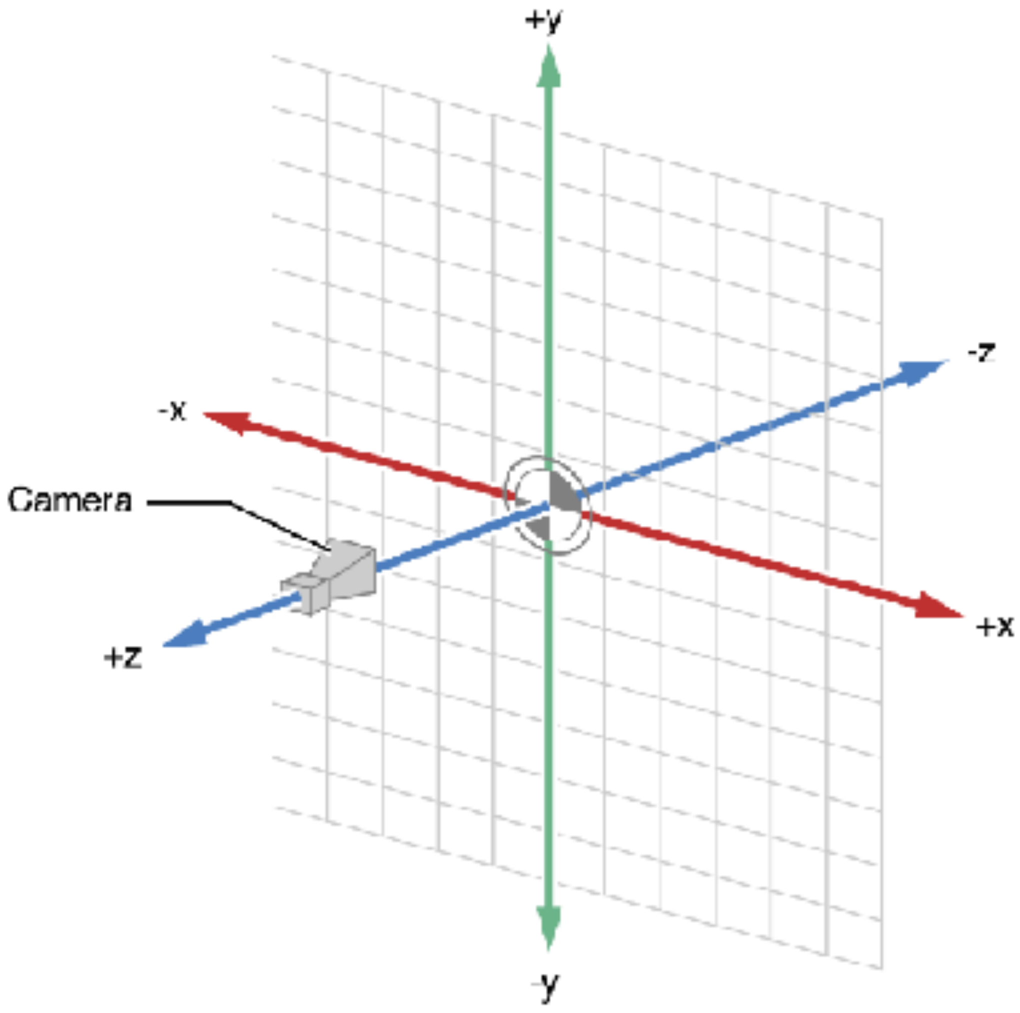
SCNLight
Type
Color
Shadow
...

SCNGeometry
Materials
Elements

SCNMaterial
Name

Diffuse
Ambient
Specular
Emission
Transparent
Reflective

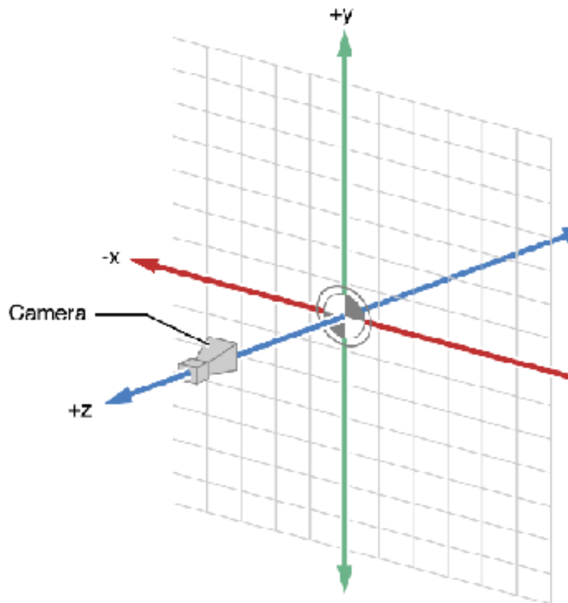
Shininess
Transparency
...



setting up a world

```
// Setup scene
scene = SCNScene()
scene.physicsWorld.speed = 1

// Setup camera position
cameraNode = SCNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(x: 0, y: 0, z: 30)
scene.rootNode.addChildNode(cameraNode)

// add a plane to the view that users must bounce the ball on
//setup the geometry of node (as a plane)
let wall = SCNPlane(width: 10.0, height: 10.0)
wall.firstMaterial?.doubleSided = true
wall.firstMaterial?.diffuse.contents = UIColor.redColor() // make it red!!

// add the plane to the world as a static body (no dynamic physics)
wallNode = SCNNode()
wallNode.geometry = wall
wallNode.physicsBody = SCNPhysicsBody.staticBody()
wallNode.position = SCNVector3(x: 0.0, y: 0.0, z: -5)

scene.rootNode.addChildNode(wallNode)

// Setup view
let view = self.view as SCNView
view.scene = scene
```

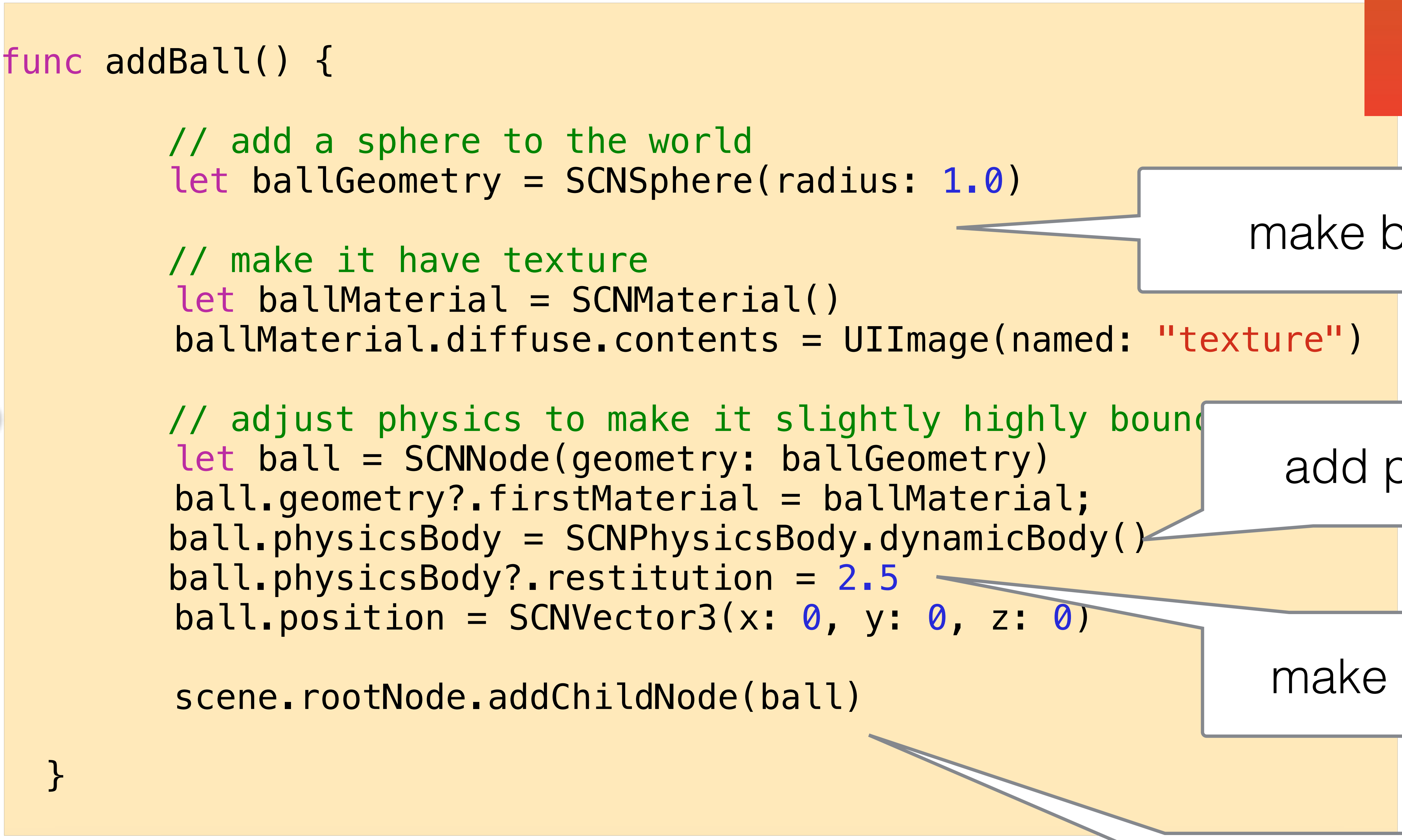
add camera



add plane

make this scene the world

add to world



```swift
func addBall() {

        // add a sphere to the world
        let ballGeometry = SCNSphere(radius: 1.0)

        // make it have texture
        let ballMaterial = SCNMaterial()
        ballMaterial.diffuse.contents = UIImage(named: "texture")

        // adjust physics to make it slightly highly bounc
        let ball = SCNNode(geometry: ballGeometry)
        ball.geometry?.firstMaterial = ballMaterial;
        ball.physicsBody = SCNPhysicsBody.dynamicBody()
        ball.physicsBody?.restitution = 2.5
        ball.position = SCNVector3(x: 0, y: 0, z: 0)

        scene.rootNode.addChildNode(ball)

  }
```

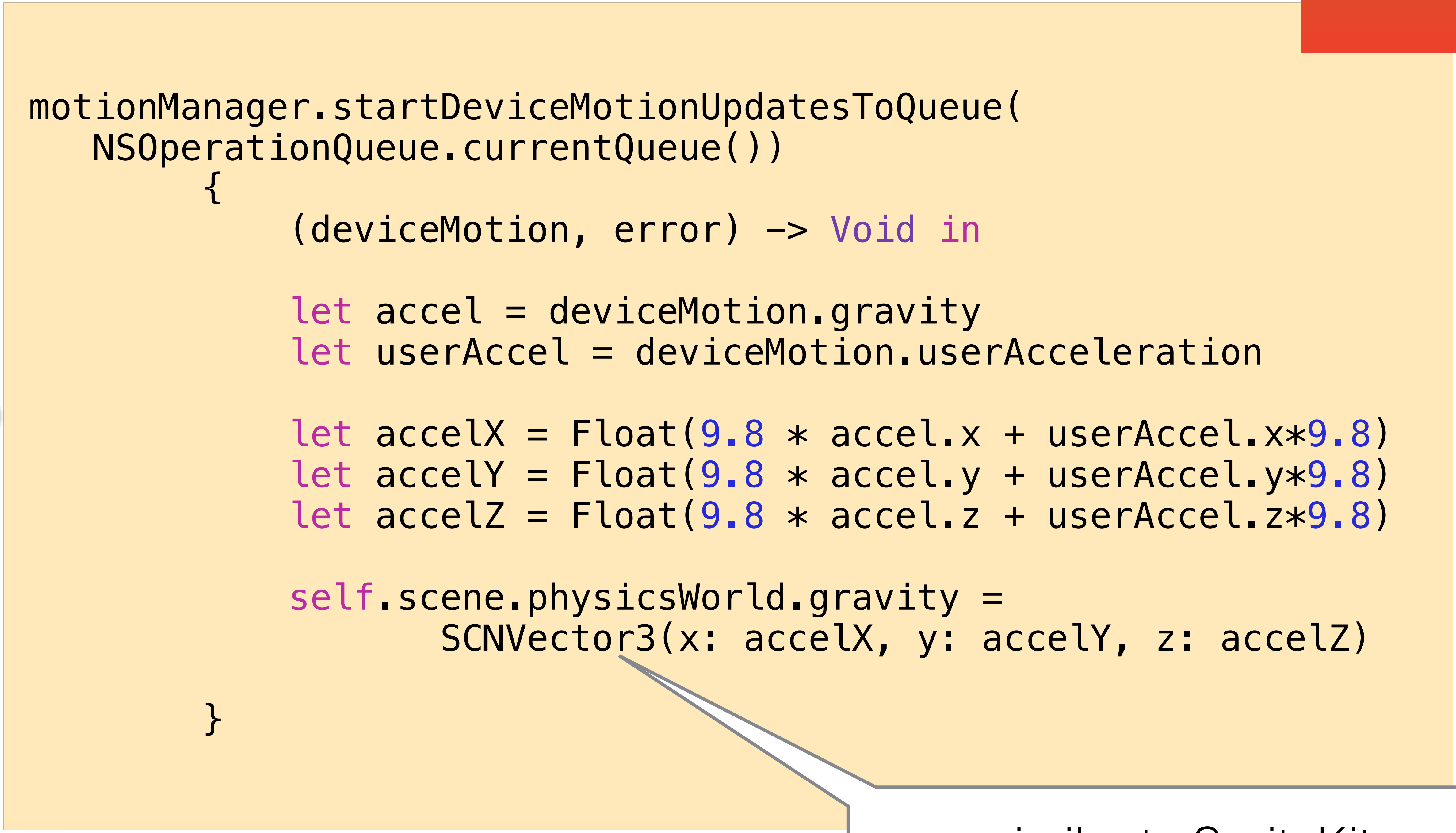
make ball

add physics

make bouncy

add to world

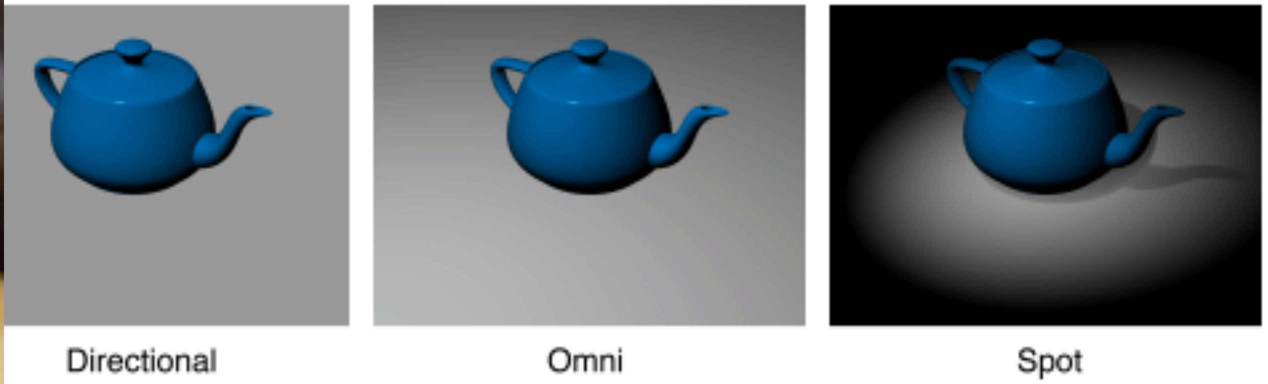
physics in world



```
motionManager.startDeviceMotionUpdatesToQueue(
   NSOperationQueue.currentQueue())
        {
            (deviceMotion, error) -> Void in

            let accel = deviceMotion.gravity
            let userAccel = deviceMotion.userAcceleration

            let accelX = Float(9.8 * accel.x + userAccel.x*9.8)
            let accelY = Float(9.8 * accel.y + userAccel.y*9.8)
            let accelZ = Float(9.8 * accel.z + userAccel.z*9.8)

            self.scene.physicsWorld.gravity =
                   SCNVector3(x: accelX, y: accelY, z: accelZ)

        }
```

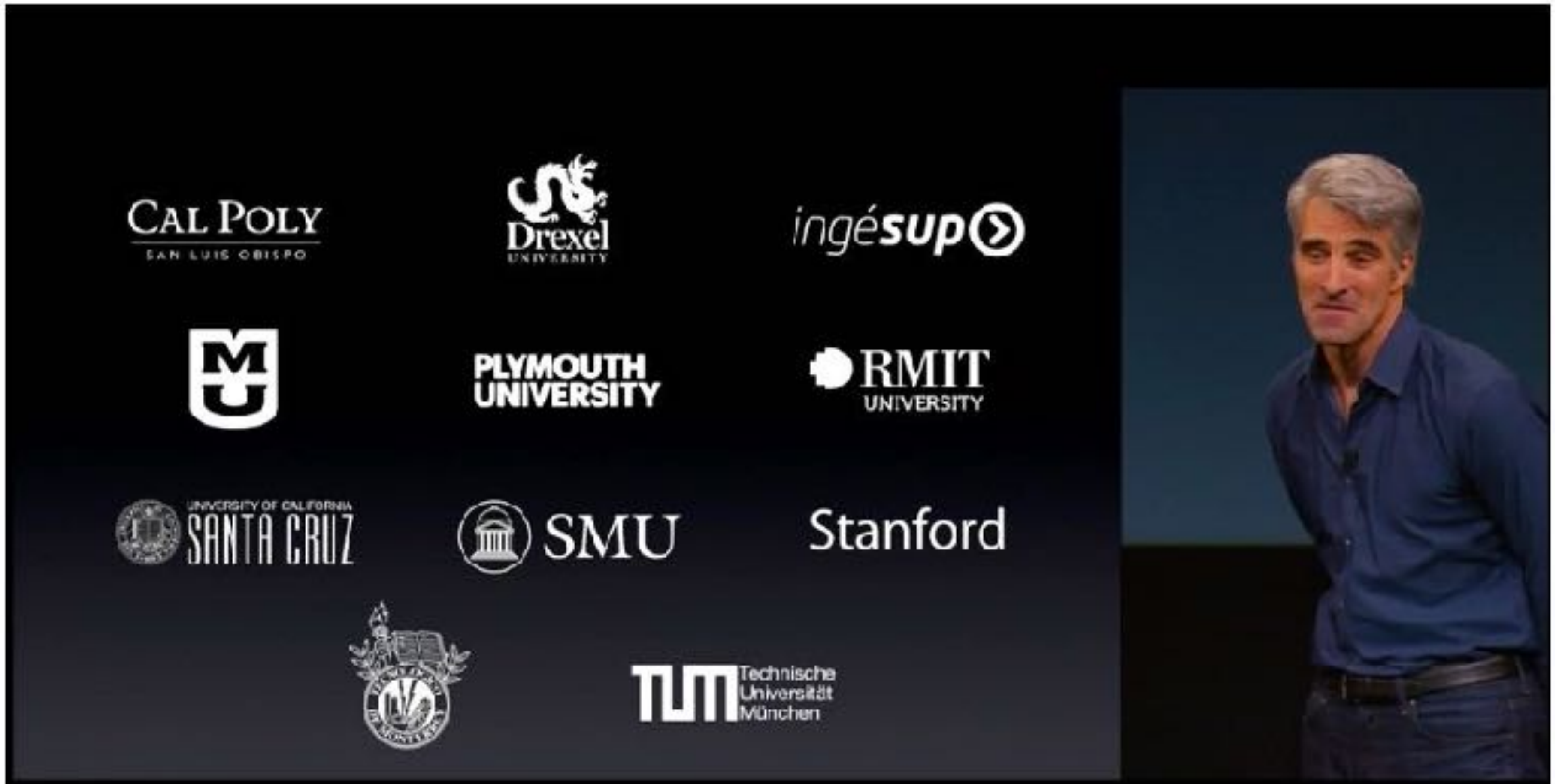
similar to SpriteKit
but in three dimensions!!

When we move to **Augmented Reality**, SceneKit is the engine for adding **Virtual Elements** to the Actual World!



device motion demo 3

- SceneKit VR



and now its time for a demo

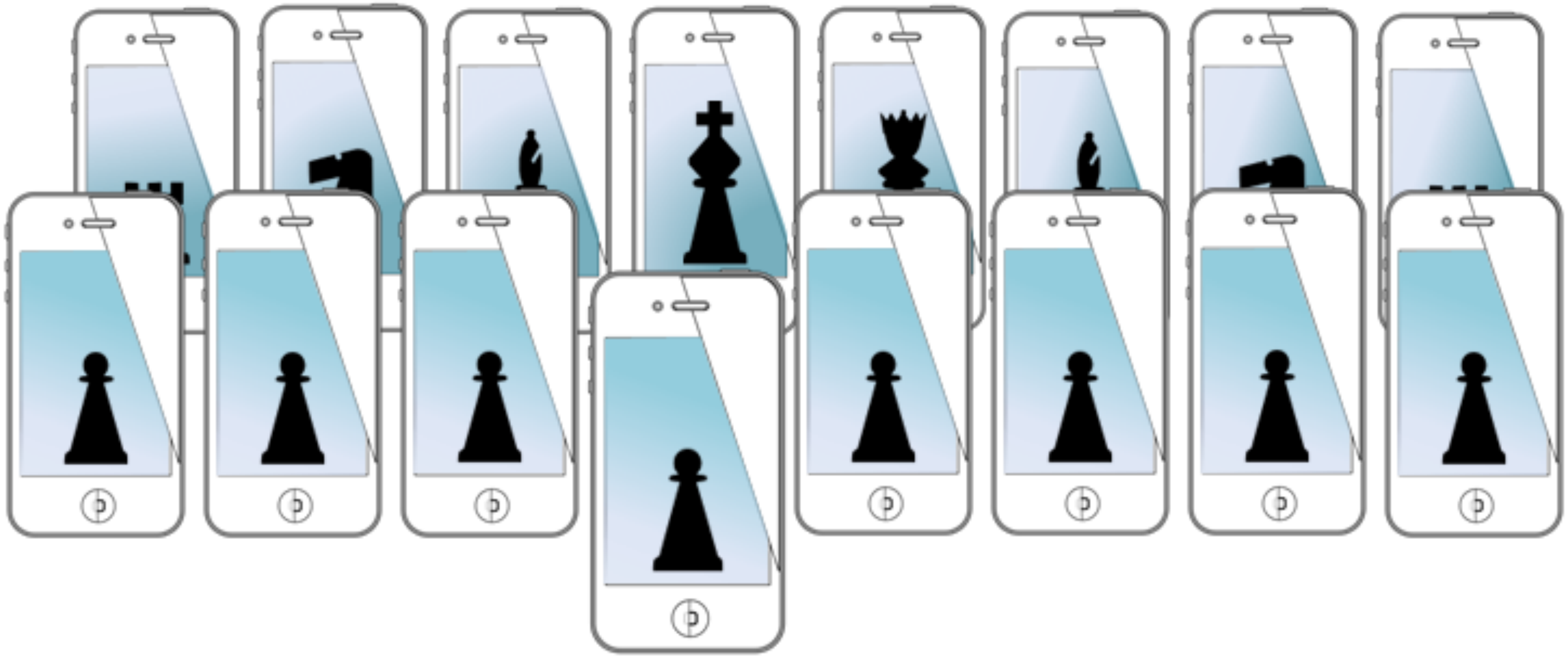
the end of motion…

- before moving on…
- assignment posted

for next time…

- Image processing!

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