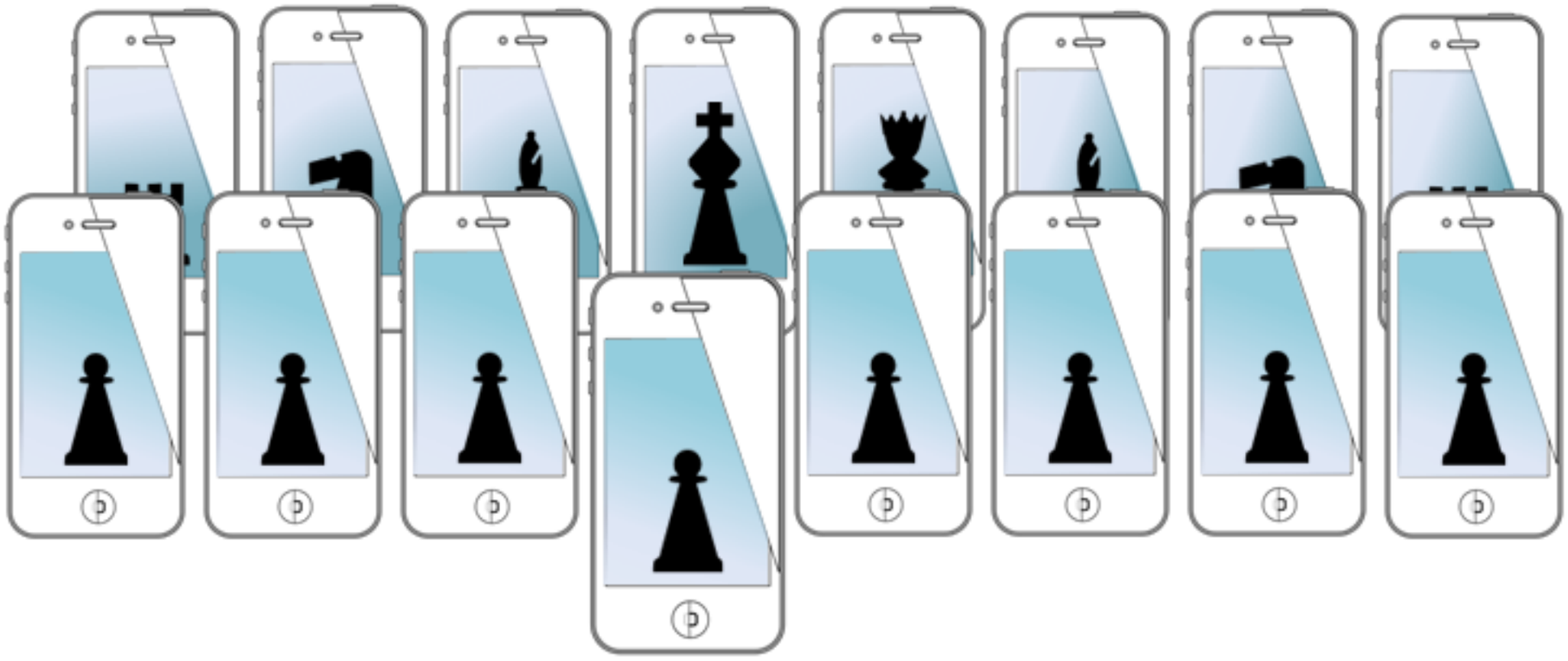
MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

course introduction

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

agenda

- class logistics
- introductions
- what is this mobile sensing course?
  - and what this course is not…
  - course goals
  - syllabus (i.e., how to do well)
    - hardware, lab, grading, MOD
- iOS development platforms

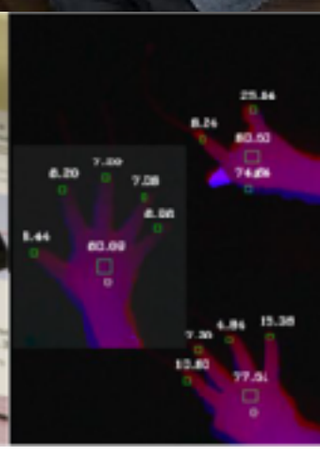
course logistics

- lecture: in class, zoom, recorded
- lab: this class has no lab!
- office hours: Monday 3:30-5PM (Caruth 451, zoom?)
- we will use canvas for managing the course
- and GitHub for managing code:
  - https://github.com/SMU-MSLC
- Zoom étiquette

introductions

- education
  - undergrad and masters from Oklahoma State
  - PhD from the university of Washington, Seattle
- research
  - signal, image, and video processing (mobile)
    - how can combining DSP, machine learning, and sensing make seamless computing?
  - security
    - smartphone side channels
  - mobile health
    - moving outside the clinic: how mobile sensing can help patients and doctors
  - sustainability
    - how technology can increase awareness

http://eclarson.com



SMARTPHONES

The sound of things to come?

SMU research finds new way to snoop: vibration of typing is translatable

By JORDAN WILKERSON
Staff Writer

Smartphones are like living things. With their cameras and microphones, they can see and hear. They can detect the amount of ambient lighting, the air pressure and the temperature — among a host of other aspects about the environment they're in.

Six years ago, less than half of Americans owned a smartphone. Four out of five own one now, says the Pew Research Center. There are millions of people walking around every day with a vast array of these sensors in their pockets.

And smartphones can record all of it.

This has created major concern about how easily ones privacy can be invaded by these sensor-rich devices, with partic-

See RESEARCH Page 4B

introductions (if time)

- me
- about you:
  - name (what you go by)
  - grad/undergrad
  - department
- **something true or false**

what is this course (and not)

- mobile sensing
  - activity recognition **some, yes!**
  - audio analysis **yes!**
  - vision analysis **yes!**
- machine learning **yes! treated as black box**
- microcontroller communication **no, not anymore**
- general iOS development **some basic skills**
- animation and graphics **no, except to display data**
- user interface design **some, all apps rely on user**

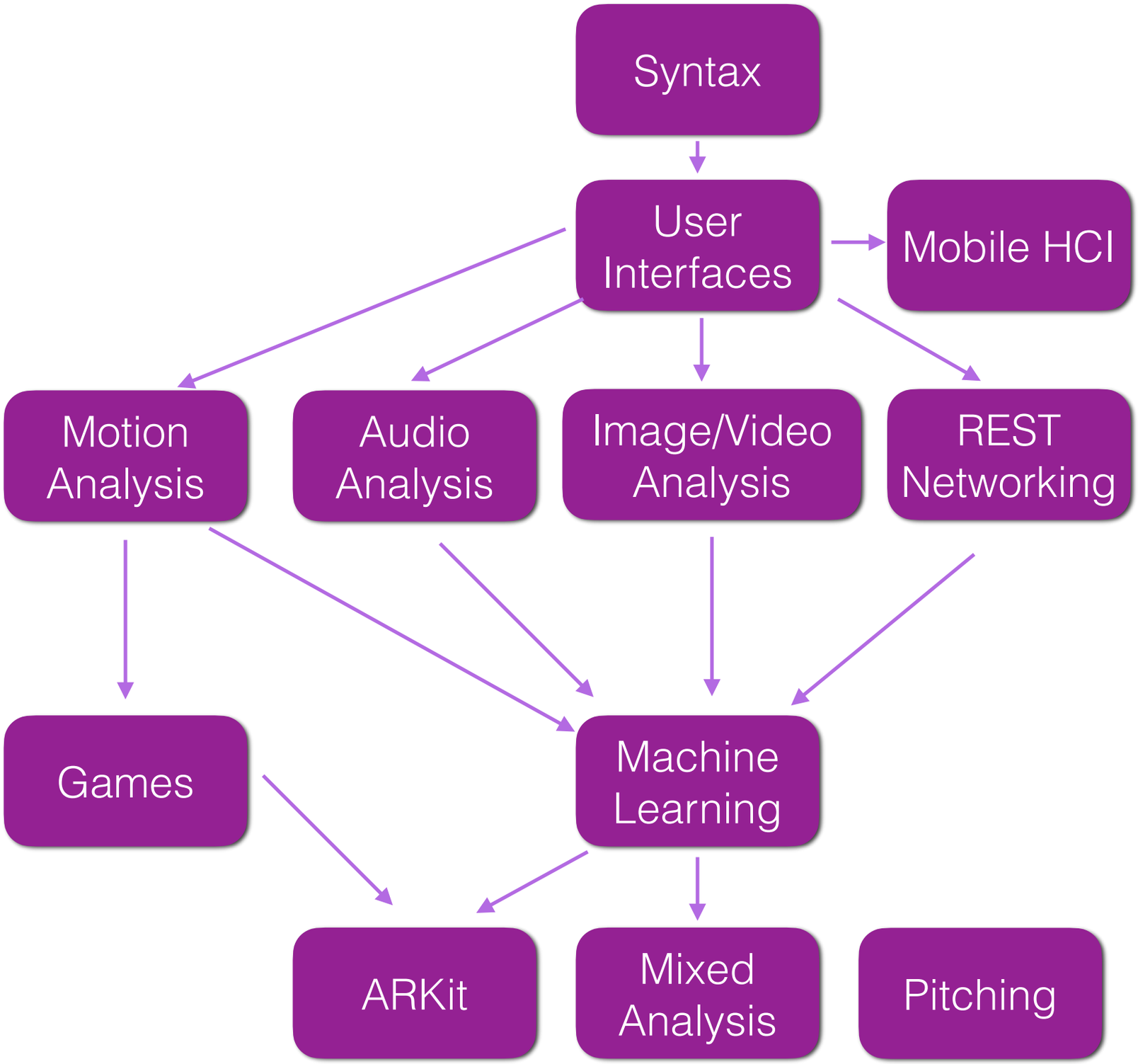
learning to learn

- for what we don't cover: take the free Stanford iOS course!
- prerequisite: model based coding
  - because you will learn at least one new language:
    - objective-c
    - swift

    ← mostly these, **prefer swift**
  - python ← for REST serving and ML training
    - c++
    - objective c++

    ← only for OpenCV

class overview



course goals

- exposure to iOS development, **MVCs** (not MVVM, SwiftUI)
- understand how to **use embedded sensors**
- **exposure** to **machine learning** for mobile sensors
  - use of built-in ML in iOS via coreML
- real time analysis of data streams
    - applications in health, education, security, etc.
- **present** and **pitch** applications

how to do well

- come to class or watch videos
- **start the app assignments early, with your team**
- iterate and test your apps
- use good coding practices, lazy instantiation, recycle classes, use comments

syllabus

- attendance
  - highly recommended, but you can watch video if needed
  - video of classes through Panopto (published after class)
- hardware is needed to develop apps
  - need a team formed (do this before the end of the week)
    - teams are expected to work remotely together
  - iPhones available for checkout, Xcode in library
  - preferable (required?) to use your own Mac
- Now let's head over to canvas

syllabus (via canvas)

- grading
- flipped assignments
- final projects
- MOD

before next class

- look at canvas and GitHub repository (clone first repository)
- get a team together (groups of 1, 2, or 3, no exceptions)
  - contribute **equally**, **everyone** codes, **everyone** designs
  - **pick good members** with different skills than you
  - take turns **coding**
  - use the lab time for coding together
  - **you can change teams throughout semester**
- all assignments are already posted for the semester and all flipped module videos

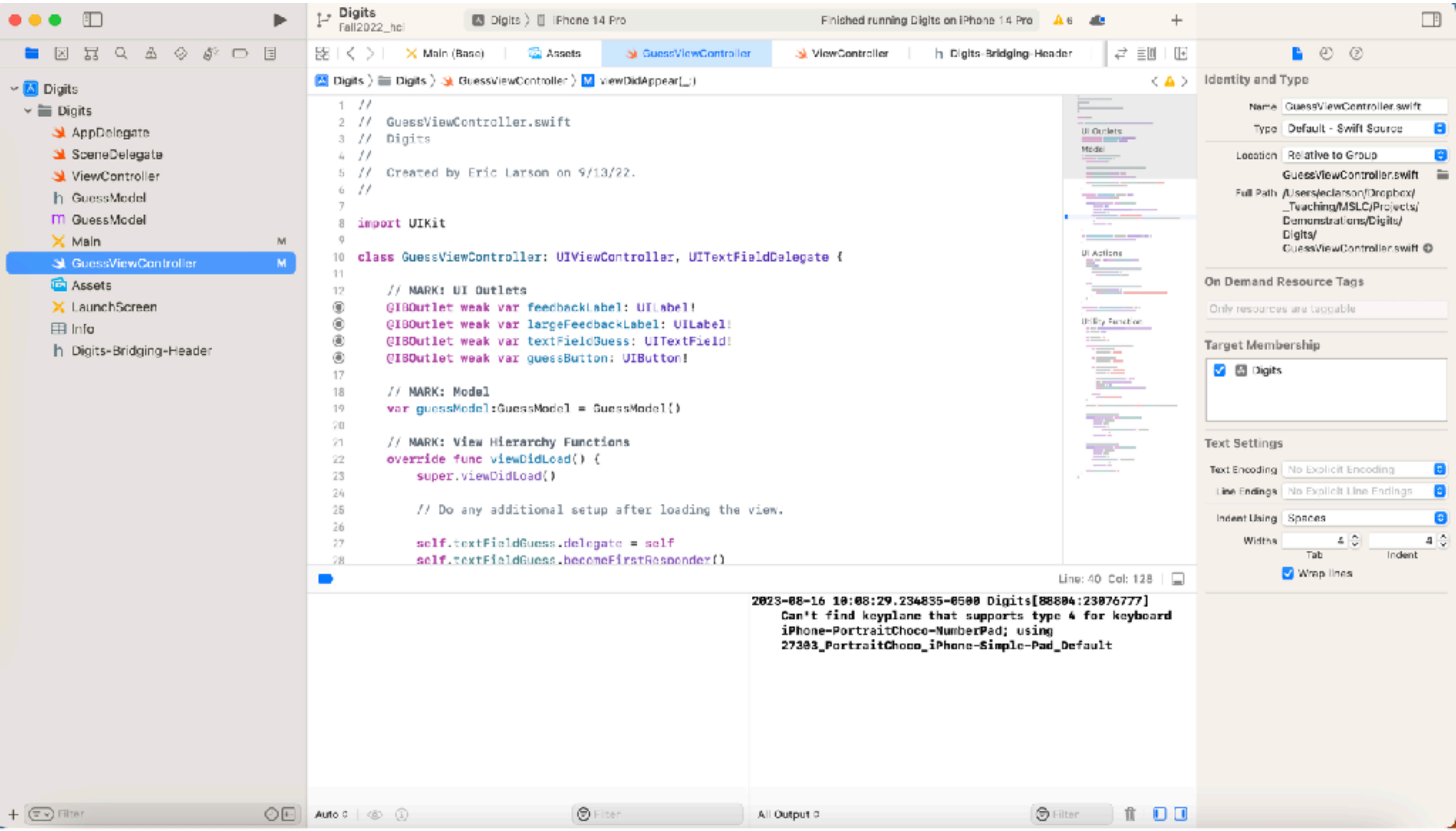
developing in iOS

- **cross platform**
  - React native, flutter, Xamarin (now MAUI), and others…
  - do not always support the latest hardware capability
  - using some phone sensors can be a pain (or slow)
  - works on many different phones
- **native (what we will be using)**
  - Xcode with objective-c and swift
  - limited to Apple, but free
  - packages are well supported, but has learning curve…
  - limited to ONLY iOS (and other Apple OS's)

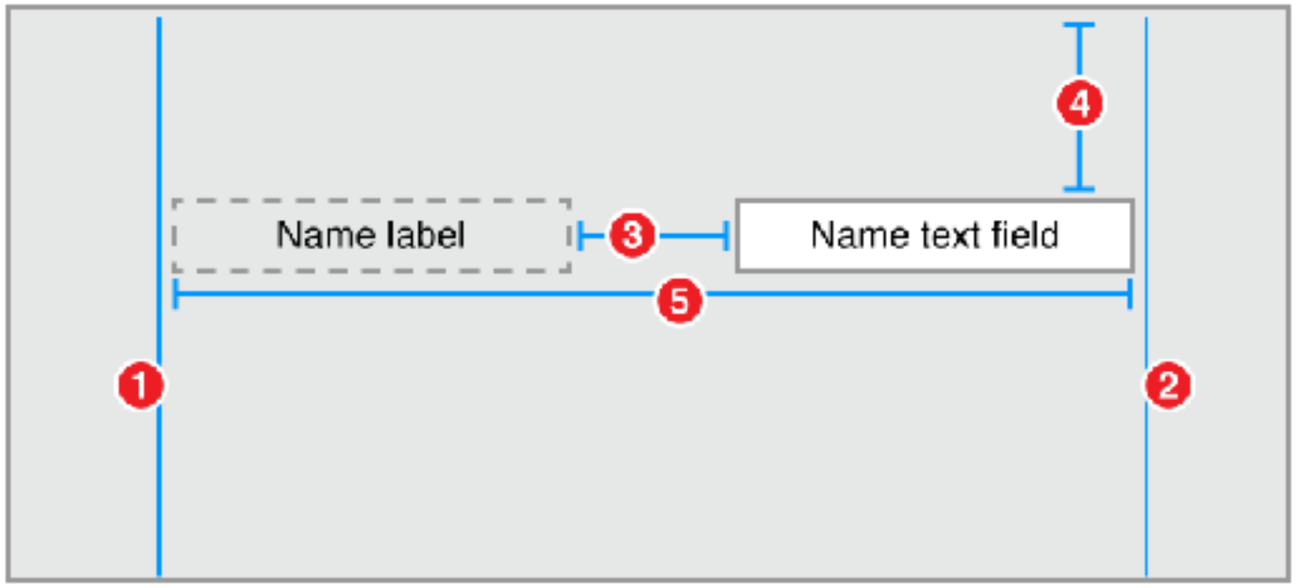
Xcode

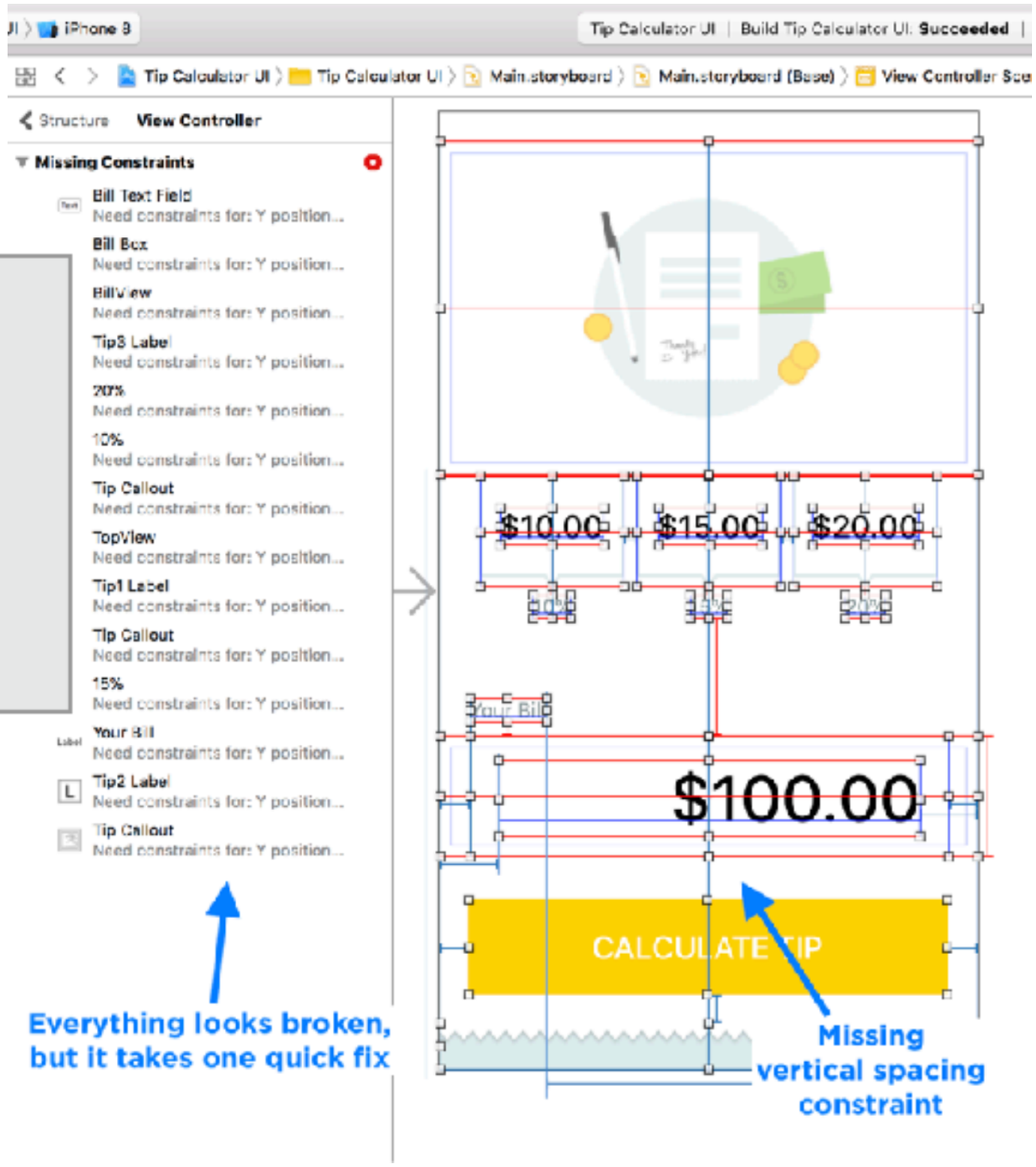
- best shown by example
- a very powerful and complex IDE
- user interface design through **SwiftUI** or **UIKit**
  - we will use UIKit exclusively (easier to get started)
  - storyboards are graphical layout editors that connect to classes for interaction
  - UIKit requires knowledge of auto layout for different sized screens

Xcode overview



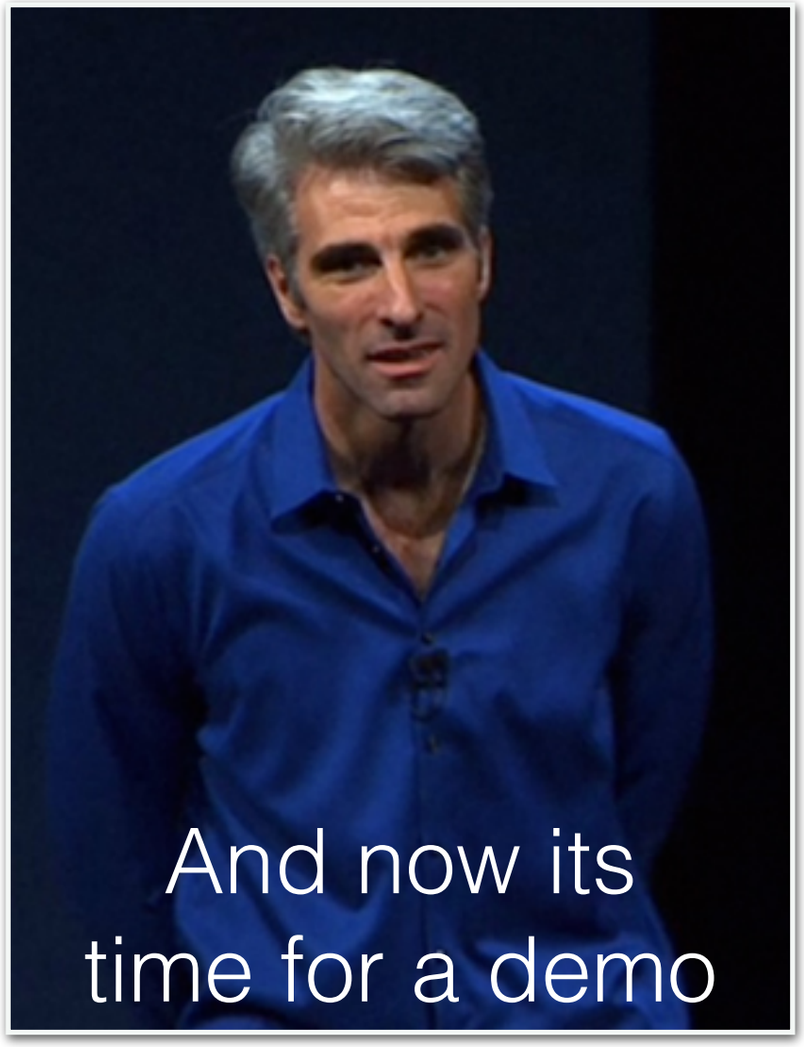
auto layout with storyboard





our first app with Xcode

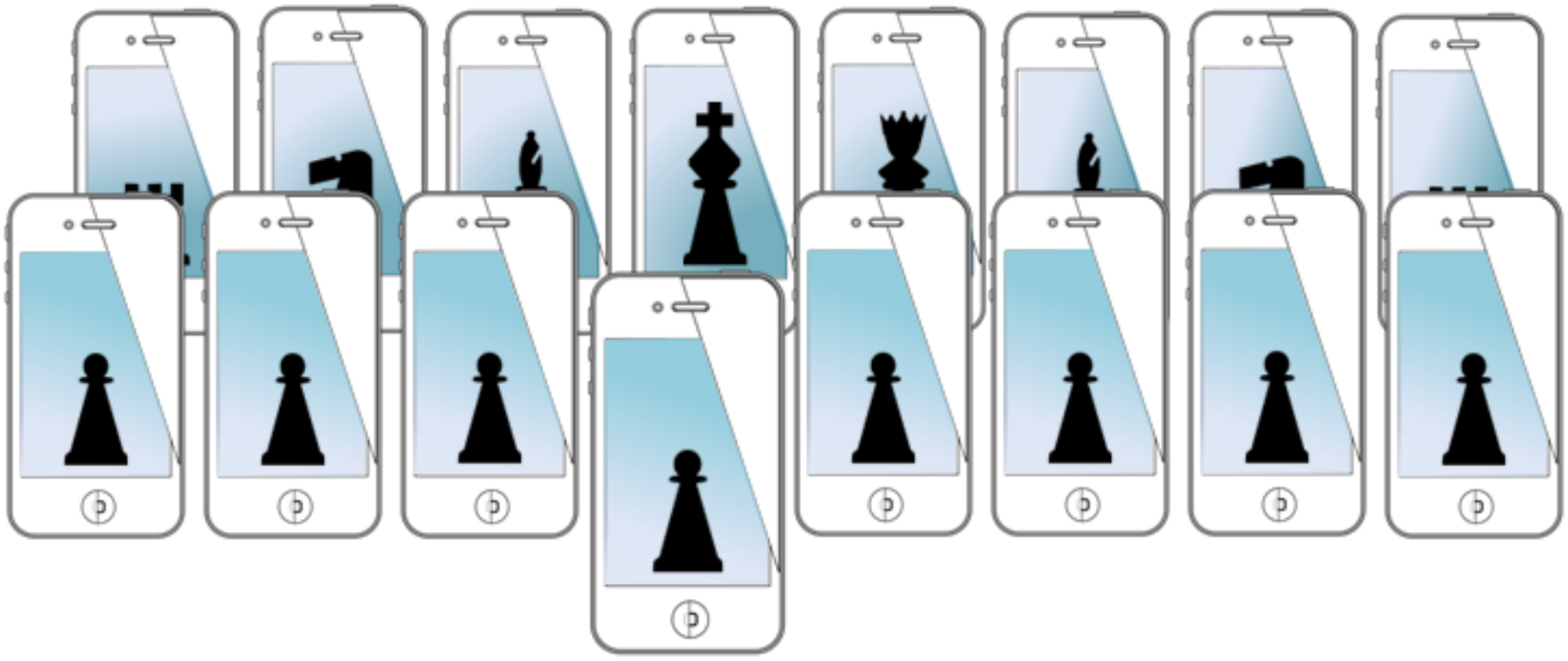
- provides GUI for most git commands
  - commit, branch, push, pull, etc.
- **rarely** is command line needed
- git is great for code but not storyboards
- and some auto layout too!



for next time…

- have teams figured out
- find out how to launch Xcode on your team mac

MOBILE SENSING & LEARNING



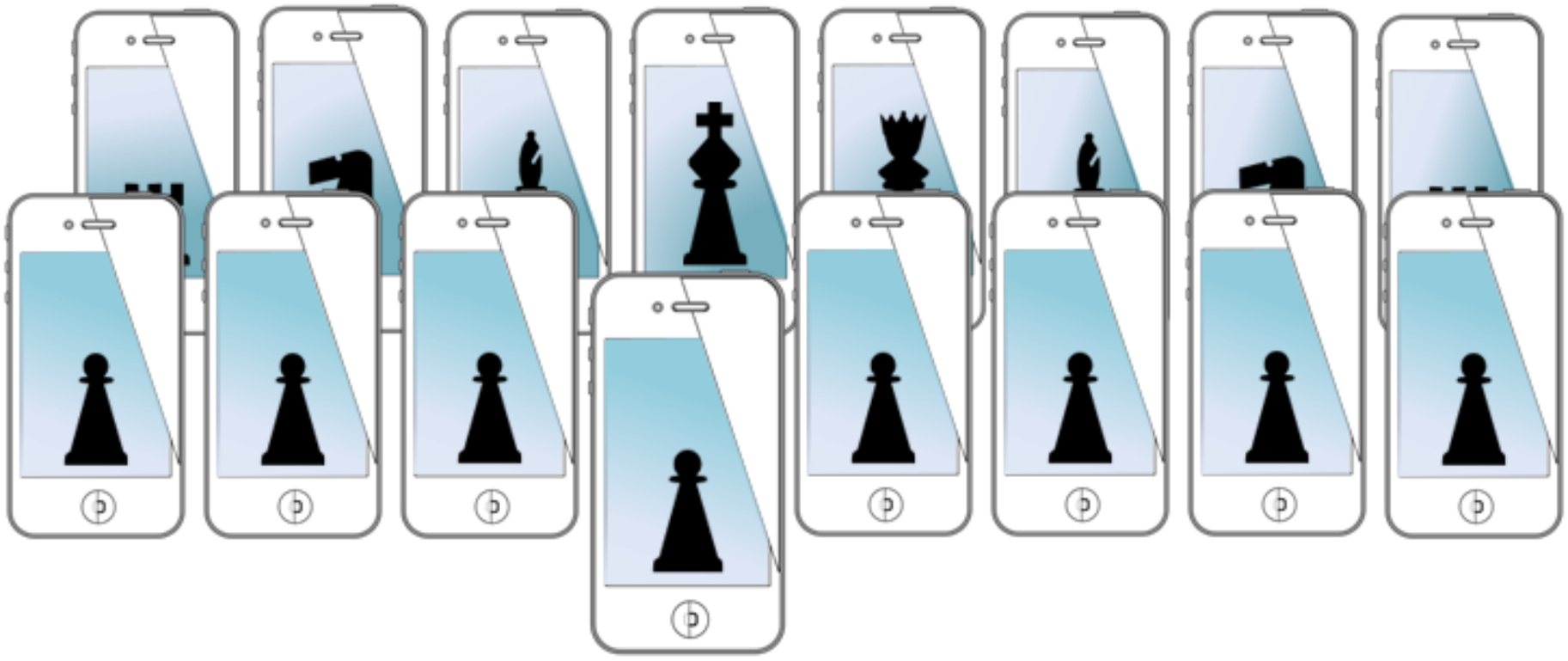
CS5323 & 7323

Mobile Sensing and Learning

course introduction

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

MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

objective-C, swift, and MVC

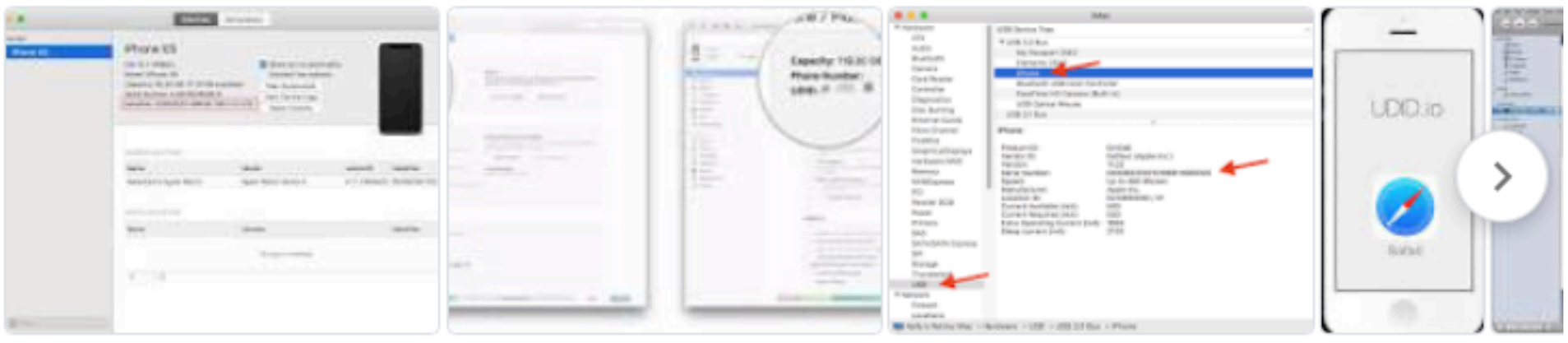
Eric C. Larson, Lyle School of Engineering,
Department of Computer Science, Southern Methodist University

course logistics

- no lab this semester
- teams: **should be on a team now**!
- **equipment checkout**: Phones available
- enrollment in 5000 versus 7000 (ugrad/grad)
- Reminder: Zoom versus in-person and other classes

Apple Developer Program

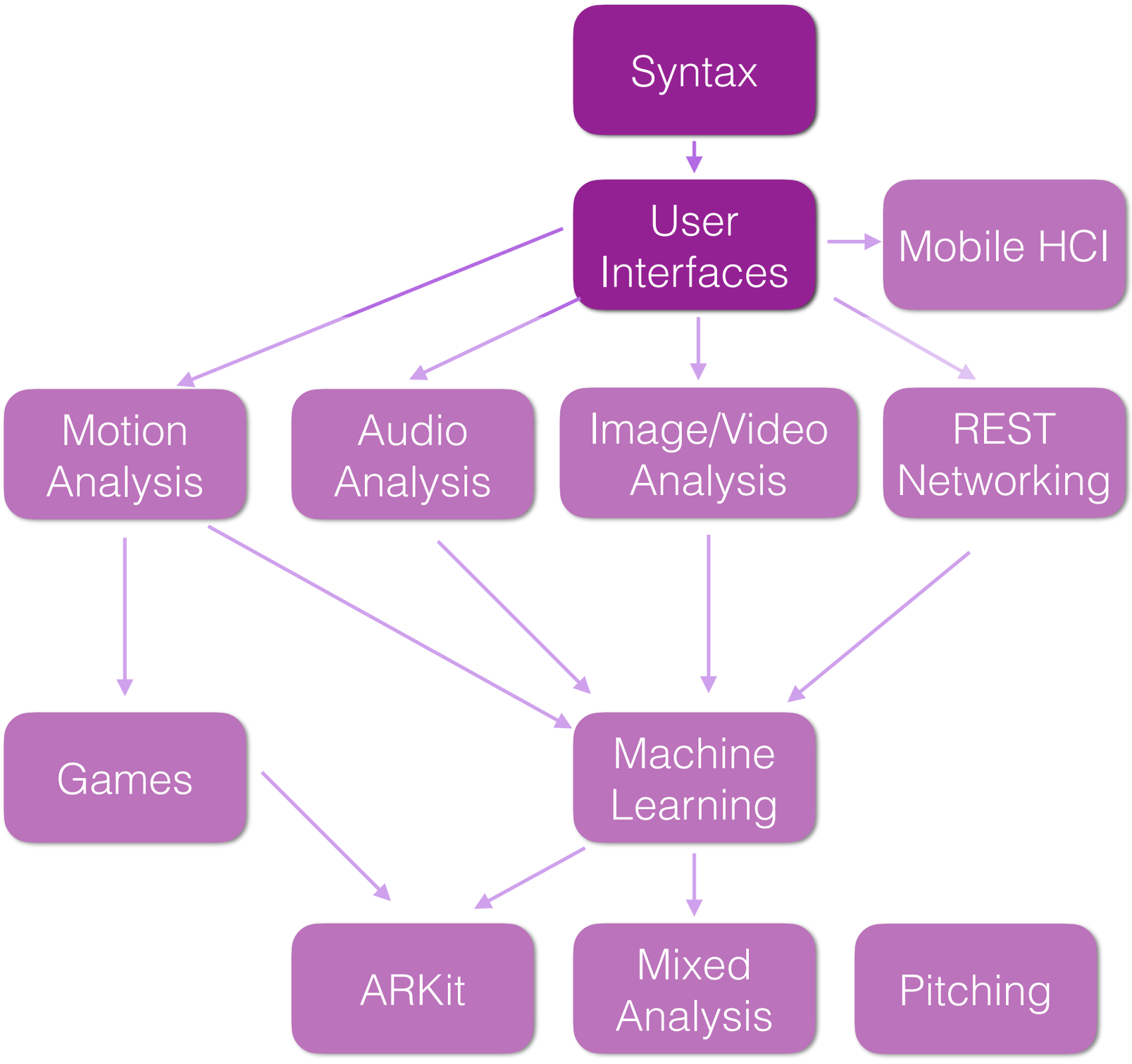
- university developer program: send me an email and I will add you to the program:
    - email that you want invite sent to
    - phone UDID: In Itunes, can also use the Xcode “simulator and devices” window



How To Find Your UDID?

1. Launch iTunes & connect your **iPhone**, iPad or iPod (device). Under Devices, click on your device. Next click on the 'Serial Number' ...
2. Choose 'Edit' and then 'Copy' from the iTunes menu.
3. Paste into your Email, and you should see the **UDID** in your email message.

class progression



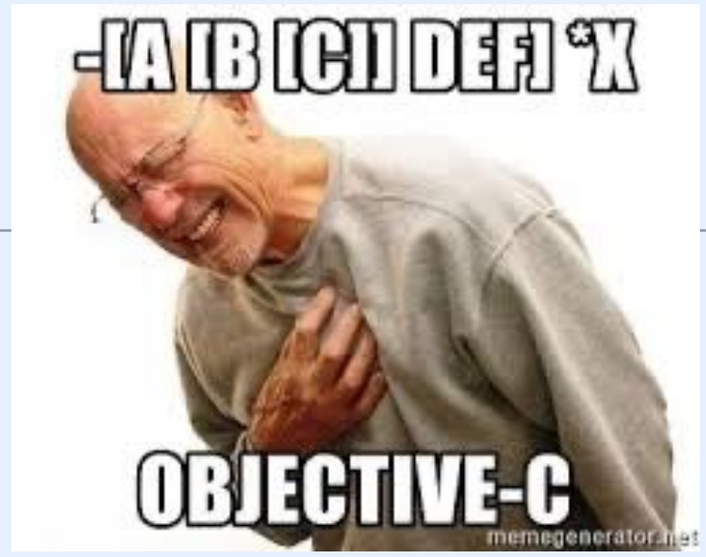
agenda

a big syntax demo…

- **objective-c and swift together**
  - class declaration
  - complex objects
  - common functions
  - encapsulation and primitives
  - memory management

and model view controllers, if time
…also available on flipped module video…

objective c



- strict superset of c
  - but with “messages”
    - so “functions” look very different (i.e., the braces in the logo)

swift

- syntax is nothing like objective-c
- but uses the same libraries…
- similarities with python syntax
  - weakly typed, no need for semicolons

an example class

```objc
@interface SomeViewController ()

@property (strong, nonatomic) NSString *aString;
@property (strong, nonatomic) NSDictionary *aDictionary;

@end

@implementation SomeViewController
@synthesize aString = _aString;

-(NSString *)aString{
    if(!_aString)
        _aString = [NSString stringWithFormat:
                    @“This is a string %d",3];
    return _aString;
}

-(void)setAString:(NSString *)aString{
    _aString = aString;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.aDictionary = @{@"key1":@3,@"key2":@"a string"};
    for(id key in _aDictionary)
        NSLog(@"key=%@, value=%@",key,_aDictionary[key]);


    NSArray *myArray = @[@32,@"a string", self.aString ];
    for(id obj in myArray)
        NSLog(@“Obj=%@",obj);


}
```

```swift
class SomeViewController: UIViewController {

    lazy var aString = {
       return "This is a string \(3)"
    }()

    var aDictionary:[String : Any] = [:]

    override func viewDidLoad() {
        super.viewDidLoad()

        self.aDictionary = ["key1":3, “key2":
                    "String value"] as [String : Any]

        for (_,val) in self.aDictionary {
            print(val)
        }

        let myArray: [Any] = [32,"a string”,
                              self.aString]

        for val in myArray{
            print(val)
        }

    }
}
```

let’s work our way up to understanding both of these examples

variables, pointers, and optionals

```
aString = nil
```

```
aString = nil
```

nil
similar to NULL_POINTER, points to nothing,
can evaluate to “false” in expression

```
double aDouble;
float aFloat;
char aChar;
int aInt;
unsigned int anUnsignedInt;
…
```

Primitives
Direct Access via Stack
CANNOT be nil

```
var aDouble:Double = 0.0
var aFloat:Float = 0.0
var aChar:Character = "c"
var aInt:Int = 0
var unsignedInt:UInt = 0
…
```

Next Step **Encapsulated**
Pointers to the Heap

```
NSString *myString;              @“ ”
NSNumber *myNum;                 @( )
NSArray *myArray;                @[ ]
NSDictionary *myDictionary;      @{ }
NSMutableArray *arrayYouCanMutate;
```

Swift **Optionals**
Pointers to the Heap

```
let myString:String? = "Const"
var myNum:Double? = nil
let myArray:[Any]? = nil
var arrayYouCanMutate:[Any]? = nil
var myDictionary:[String:Any]? = nil
```

classes

class name

inherits from

```
@interface SomeClass  : NSObject

@property (strong, nonatomic) NSString *aPublicStr;

@end
```

if in the **.h** file, it is public

obj-c property:
NOT variables, but they provide *access* to backing variables

```
@interface SomeClass ()

@property (strong, nonatomic) NSString *aPrivateStr;

@end

@implementation SomeClass

   //… implementation stuff…
@end
```

if in the **.m** file, it is private

class name

inherits from

```
class SomeClass : NSObject{

    var aPublicString = "..."
    private var aPrivateString = "..."

    // implementation stuff
}
```

swift defaults to **public** properties

swift property:
special variables
can add functionality through observers and overrides

objective c

class property: access a variable in class

```
@interface SomeClass ()
@property (strong, nonatomic) NSString *aString;
@end
@implementation SomeClass
@synthesize aString = _aString;
```

property declared

backing variable: usually implicit to compiler

setter, **auto** created
`self.aString=val;`

```
-(void)setAString:(NSString *)aString{
    _aString = aString;
}
```

getter, **auto** created
`val=self.aString;`

```
-(NSString *)aString{
  return _aString;
}
```

property `self.aString`
variable `_aString`

getter, **custom overwrites** auto creation

```
-(NSString *)aString{
    if(!_aString)
        _aString = @"This string was not set";
    return _aString;
}
```

lazy instantiation

```
@end
```

objective c

class properties

```
@interface SomeClass ()

  @property (strong, nonatomic) NSString *aString;


@end

@implementation SomeClass
   -(NSString *)aString{
       if(!_aString)
           _aString = @"This string was not set”;
       self.aString = @“Getter Called to set”;
       return _aString;
   }
   -(void)someFunction{

       _aString = @“Direct variable Access, No getter Called”;

       self.aString = @“Getter Called to set”;

   }

@end
```

What does this do?

swift

class properties

```
class SomeClass : NSObject{

    var aPublicString = "..."
    private var aPrivateString = “..."

    var noDefaultVal:Int
    override init() {
        self.noDefaultVal = 0
    }

    lazy var aString = "Default val if not set"
    lazy var aStringAlso = {
        // could do other things here
        return "Value"
    }()


    var watchedVariable:Float = 0.0 {
        willSet(newValue){
            print("setting value to \(newValue)")
        }
        didSet{
            print("\(oldValue) set to \(watchedVariable)")
        }
    }

}
```

property declared in class directly

if no default value, must be setup in `init()`

lazy instantiation, set to values if accessed

property observers: willSet and didSet

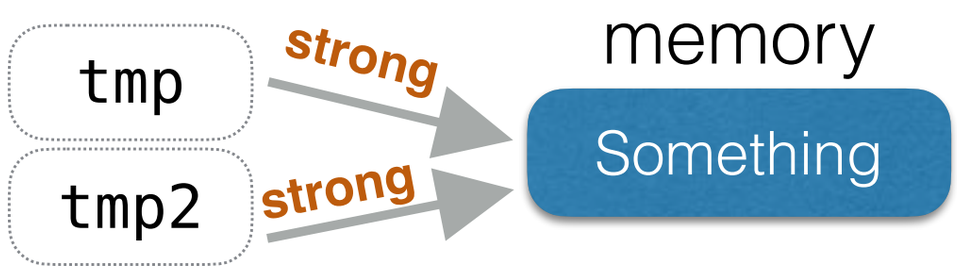
can also override “set” and “get” methods, but this is rare to need

automatic reference counting

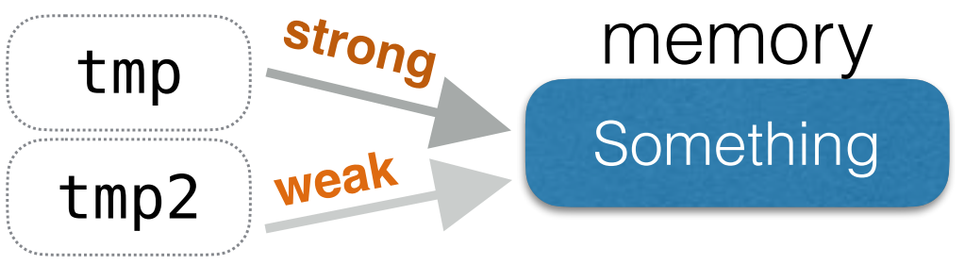
- not garbage collection
- when reference count for variable == 0, trigger event to free memory
  - **strong** pointer adds to reference count
  - **weak** pointer does not add to reference count
  - **unowned** special case of weak, always assumes there is a strong reference with longer lifetime

```
var tmp:String? = "Something"
var tmp2 = tmp
tmp = nil
tmp2 = nil
```

```
NSString* tmp = @"Something";
NSString* tmp2 = tmp;
tmp = nil;
tmp2 = nil;
```

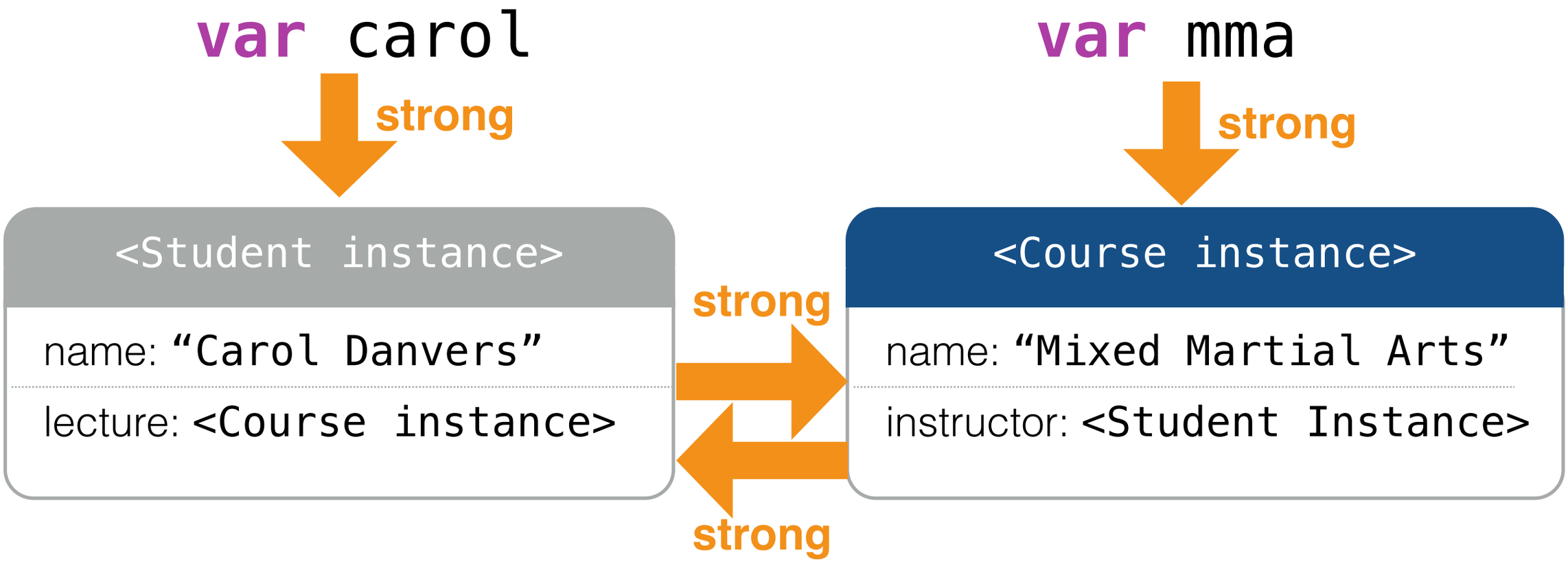


- deallocated after **both references** are `nil`



- deallocated after **strong reference** is `nil`

automatic reference counting

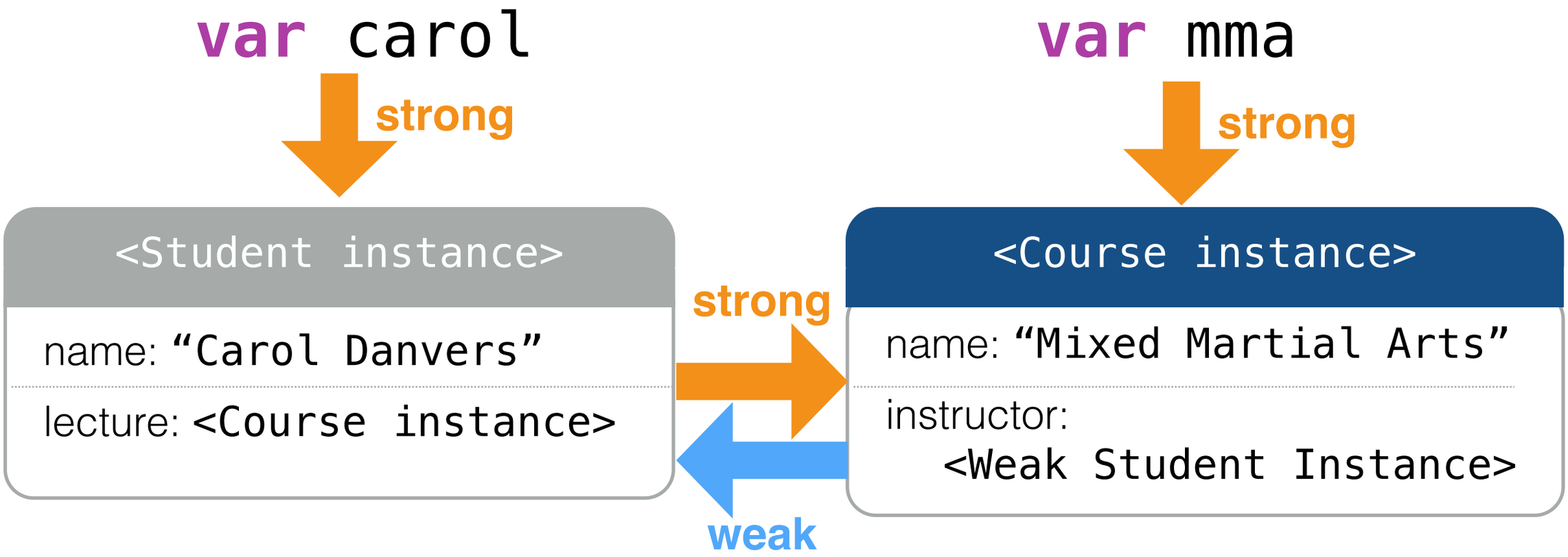



```
carol.lecture = mma
mma.instructor = carol

mma = nil
carol = nil
```

- memory never deallocated because reference cycle
- results in a memory leak if done repeatedly
- solution: weak pointers

automatic reference counting

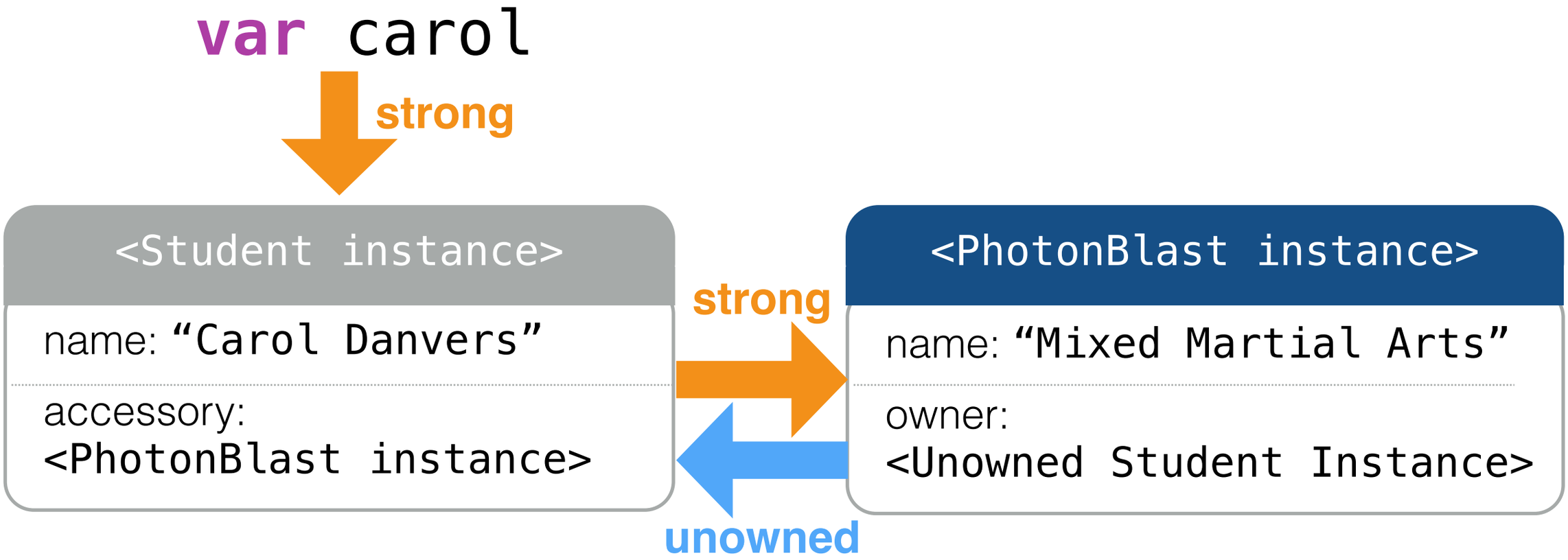



```
carol.lecture = mma
mma.instructor = carol

carol = nil
mma = nil
```

- references to parent instance cascade into properties
- all memory released immediately for use in app

unowned usage



- used primarily when there is no need for referencing a class instance without the parent instance
- typically one-to-one class instances

using strong, weak, unowned

atomic ~ thread safe property access
nonatomic ~ faster access

```
@property (strong, nonatomic) Student *aStudent;
```

strong ~ keep a reference
weak ~ no reference

```
weak var aStudent: Student?
unowned var aStudent: Student?
```

strong by default in swift
weak used when needed

```
self.aStudent = [[Student alloc] init];
```

most common initialization
syntax for obj-c and swift

```
self.aStudent = Student()
```

properties are accessed
through `self` (like c++)

iteration on objects

```
NSArray *myArray = @[@32,@"a string”, [@3.1,@3,@5,@9,@42,@32]];
for(id obj in myArray)
    NSLog(@“Obj=%@",obj);
```

can store any object

loop over an NSArray

```
@interface SomeClass ()
@property (strong, nonatomic) NSDictionary *aDictionary;
@end
```

Dictionary as a class property

Access self

```
self.aDictionary = @{@"key1":@3,@"key2":@"a string"};
for(id key in self.aDictionary)
    NSLog(@"key=%@, value=%@",key,self.aDictionary[key]);
```

```
let myArray: [Any] = [32,"a string”, self.aString]
for val in myArray{
      print(val)
}

self.aDictionary = ["key1":3, “key2":"String value"] as [String : Any]

for (_,val) in self.aDictionary {
      print(val)
}
```

declaration requires specifying **any** if the data is not consistent

Dictionary loops through as tuple **(key, varName)**

mutable and immutable

```
NSArray *myArray = @[@32,@"a string”, [[UILabel alloc]init] ];
```

arrays are **nil** terminated

```
NSMutableArray *anArrayYouCanAddTo = [NSMutableArray arrayWithObjects:aNum,@32, nil];

[anArrayYouCanAddTo addObject:someComplexObject];
```

possible to add objects now

```
NSMutableArray *anotherArray = [@[@32,@"string me"] mutableCopy];
```

```
let myConstArray = [34, 22, 1]
var myArray = [22, 34, 12]
```

more explicit in swift regarding mutability

functions examples

return type | method name | parameter type | parameter name

```
-(NSNumber*) addOneToNumber:(NSNumber *)myNumber {}
```

```
-(NSNumber*)    addOneToNumber:(NSNumber *)myNumber
           withOtherNumber:(NSNumber *)anotherNumber
```

second parameter

receiver class | parameter name/value

```
NSNumber *obj = [self addOneToNumber:@4];
NSNumber *obj = [self addToNumber:@4 withOtherNumber:@67];
```

(+ —) instance versus class method

throwback to **c**

```
float addOneToNumber(float myNum){
    return myNum++;
}

float val = addOneToNumber(3.0);
```

```
func addOneToNumber(myNumber:Float) -> (Float){
    return myNumber+1
}

func addOneToNumber(myNum:Float, withOtherNumber myNum2:Float) -> (Float){
    return myNum+myNum2+1
}

var obj = self.addOneToNumber(myNumber: 3.0)
var obj = self.addOneToNumber(myNum: 3.0, withOtherNumber: 67)
```

(varName:Type) -> (Return Type)

similar named second parameter syntax in swift

common logging functions

function

NSString to format

object to print

```
NSLog(@"The value is: %@",someComplexObject);
NSLog(@"The value is: %d",someInt);
NSLog(@"The value is: %.2f”,someFloatOrDouble);
```

%@ is print for serializable objects

```
someComplexObject = nil;

if(!someComplexObject)
    printf("Wow, printf works!");
```

set to nothing,
subtract from reference count

nil only works for objects!
no primitives, structs, or enums

```
var complexObj:Float? = nil

if let obj = complexObj{
    print("The value is: \(obj)")
}
```

`if let` syntax, **safely unwraps** optional

print variable within string using
`\( varName )`

review

```objc
@interface SomeViewController ()

@property (strong, nonatomic) NSString *aString;
@property (strong, nonatomic) NSDictionary *aDictionary;

@end

@implementation SomeViewController
@synthesize aString = _aString;

-(NSString *)aString{
    if(!_aString)
        _aString = [NSString stringWithFormat:
                    @“This is a string %d",3];
    return _aString;
}

-(void)setAString:(NSString *)aString{
    _aString = aString;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.aDictionary = @{@"key1":@3,@"key2":@"a string"};
    for(id key in _aDictionary)
        NSLog(@"key=%@, value=%@",key,_aDictionary[key]);


    NSArray *myArray = @[@32,@"a string", self.aString ];
    for(id obj in myArray)
        NSLog(@“Obj=%@",obj);


}
```

private properties

backing variable

getter

setter

call from super class

dictionary iteration

array iteration

[C]

```swift
class SomeViewController: UIViewController {

    private lazy var aString = {
       return "This is a string \(3)"
    }()

    private var aDictionary:[String : Any] = [:]



    override func viewDidLoad() {
        super.viewDidLoad()

        self.aDictionary = ["key1":3, “key2":
                    "String value"] as [String : Any]

        for (_,val) in self.aDictionary {
            print(val)
        }



        let myArray: [Any] = [32,"a string”,
                              self.aString]
        for val in myArray{
            print(val)
        }

    }
}
```

private properties

call from super class

dictionary iteration

array iteration

adding to our project

- let's add to our project
  - an objective-c class
  - that uses lazy instantiation

for next time…

- **next time**: more dual language programming
- **one week**: flipped assignment
- **then**: mobile HCI

MVC's

controller has direct connection to view class

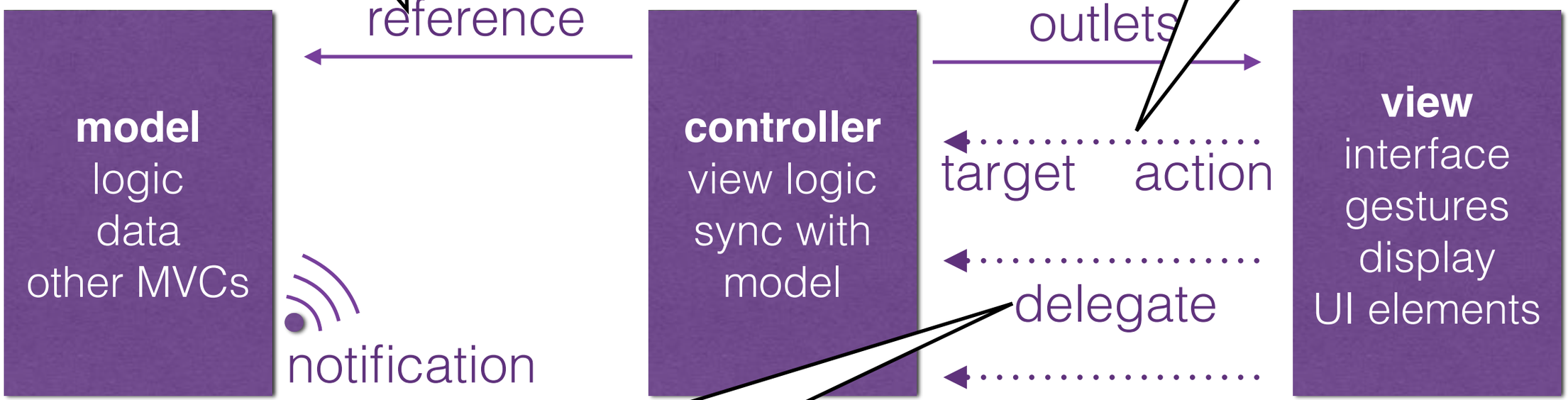
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

controller has direct connection to model class

```
ModelClass *myModel = [get global handle to model]
PhoneNumberStruct * phNumber = [myModel getNumber];
self.phoneNumberLabel.text = phNumber.number;
```

view sends a targeted message

```
- (IBAction)buttonPressed:(id)sender;
- (IBAction)showPhBookPressed:(id)sender;
```




```
MainViewController ()<UITextFieldDelegate>
#pragma  mark - UITextfield Delegate
- (BOOL)textFieldShouldReturn:(UITextField *)textField { …  }
```

controller implements method for view class

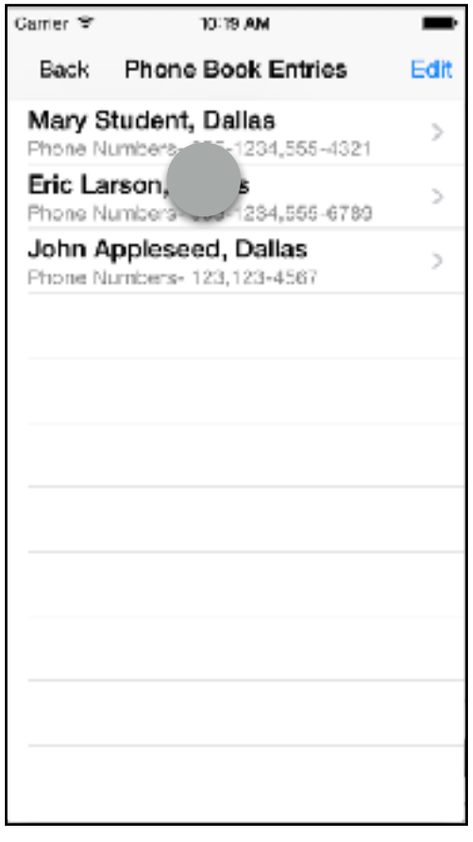
Legend

direct connection
indirect action
general broadcast

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

MVC life cycle

- problem: we need to handoff control of the screen to a new view
- the app itself is handling most of this transition
  - app will “unfreeze” the new view and its class properties
  - **you** need to send information from **source** ViewController to **destination** ViewController



controller life cycle

Source Controller

`prepareForSegue`
prepare to leave the screen

set properties of destination, if needed

Destination Controller

view is unfrozen, property memory allocated

view outlets are ready for interaction

`viewDidLoad`

`viewWillAppear`

`viewDidAppear`

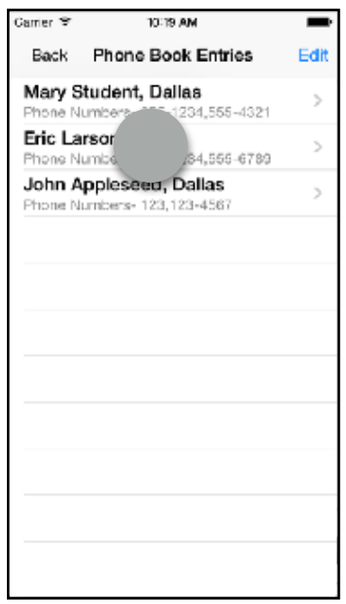
`viewWillDisappear`

`viewDidDisappear`

memory deallocated when app is ready

user

source



destination

MVC's

- sometimes the best way to create a model is through a Singleton

in .h file (so its public)

```
@interface MyCustomClass : NSObject

+ (MyCustomClass*)sharedInstance;

@end
```

+ means its a
class method
don't need instance to call it

in .m file

custom getter

called like a backing variable

```
+ (MyCustomClass*)sharedInstance
{
    static MyCustomClass * _sharedInstance = nil;

    static dispatch_once_t oncePredicate;

    dispatch_once(&oncePredicate, ^{
         _sharedInstance = [[MyCustomClass alloc] init];
     });
     return _sharedInstance;
}
```

don't worry about syntax
until next week…

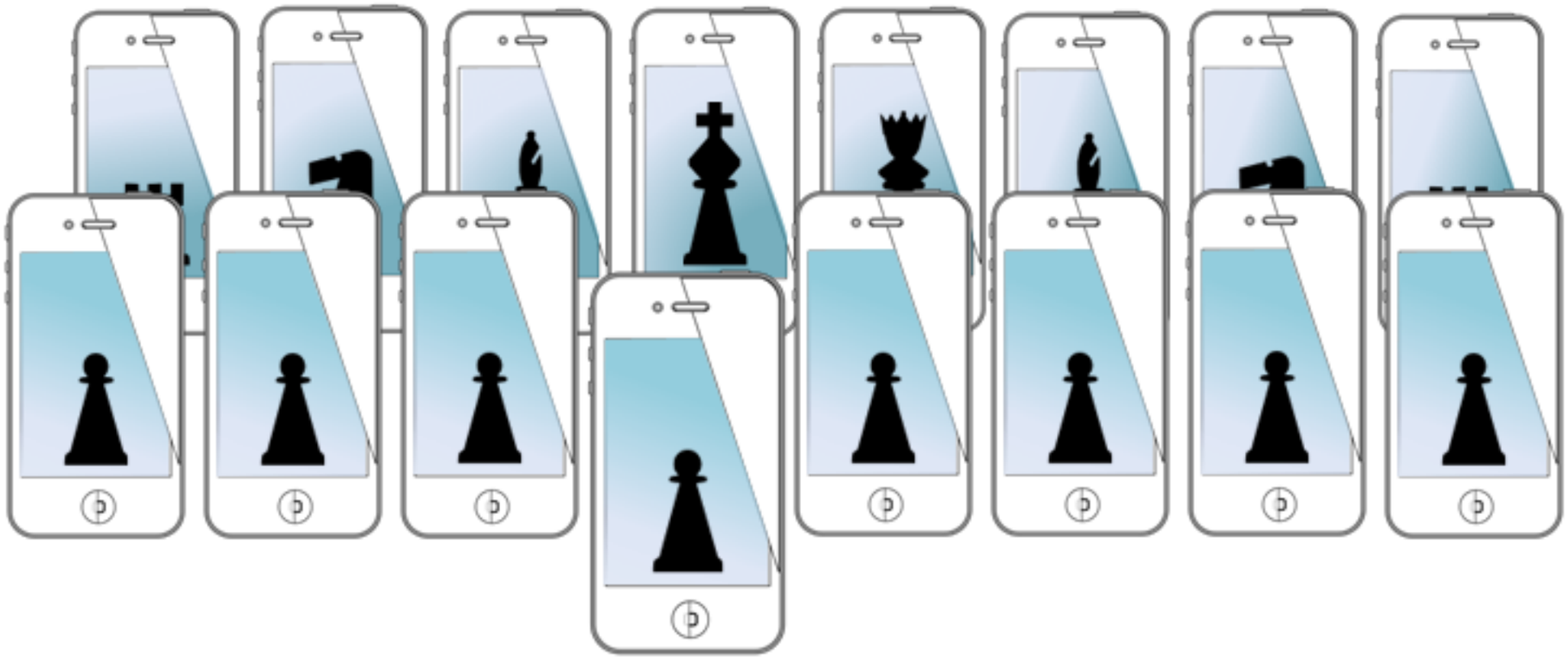
Need more help on MVC's ? Check out Ray Wenderlich:

http://www.raywenderlich.com/46988/ios-design-patterns

for next time…

- Swift
- Mobile HCI

MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

objective-C and MVC

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