

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, and recorded
- **lab: this class has no lab!**
- office hours: Monday 3:30-5PM (Caruth 451, zoom)
- TA currently being on-boarded, will also have office hours
- we will use canvas/GitHub for managing the course
- and GitHub for managing code:
 - <https://github.com/SMU-MSLC>
- Zoom etiquette

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>



Phyn
Smart Water Assistant

SMARTPHONES

The sound of things to come?

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

By JORDAN WILKERSON
Staff Writer

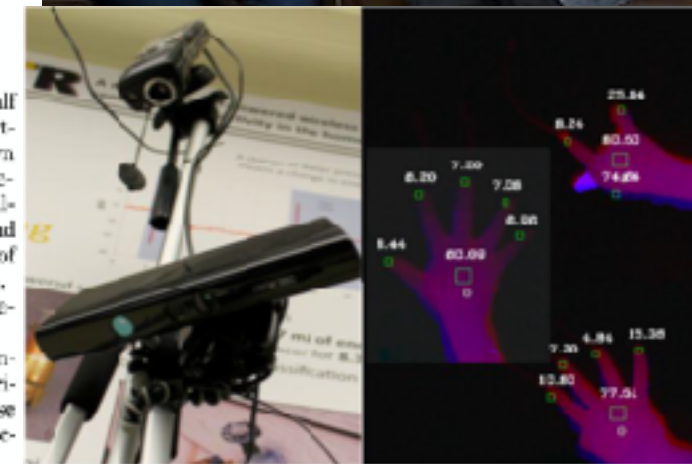
Smartphones are like living things. With their cameras and microphones, they can see and hear. They can detect the amount of ambient lighting, the air pressure and the temperature — among a host of other aspects about the environment they're in.

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 one's privacy can be invaded by these sensor-rich devices, with partic-

See **RESEARCH** Page 4B



introductions (limited)

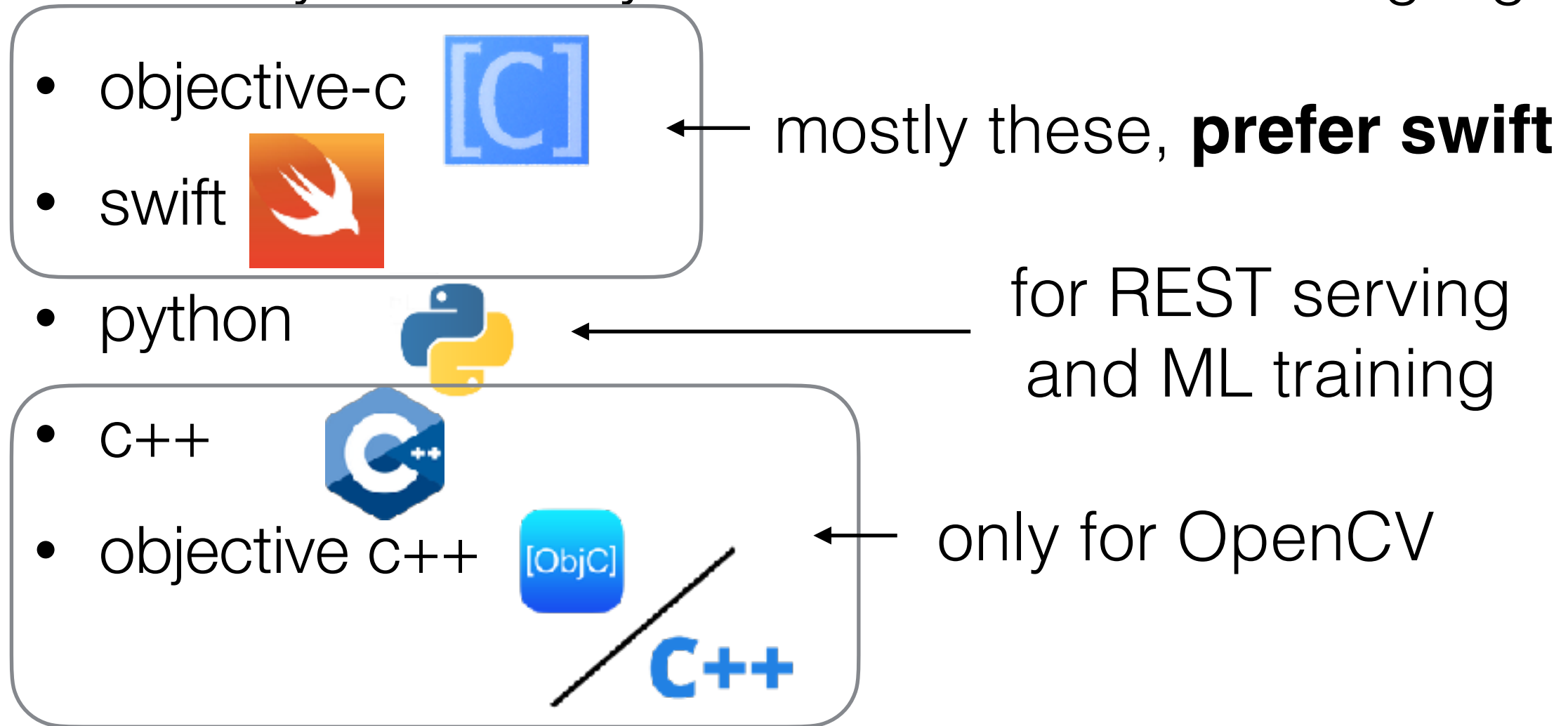
- me: Dr./Professor/ (Eric, if PhD student)
- about you:
 - name (what you go by)
 - grad/undergrad
 - department
 - **something true or false**

what is this course?

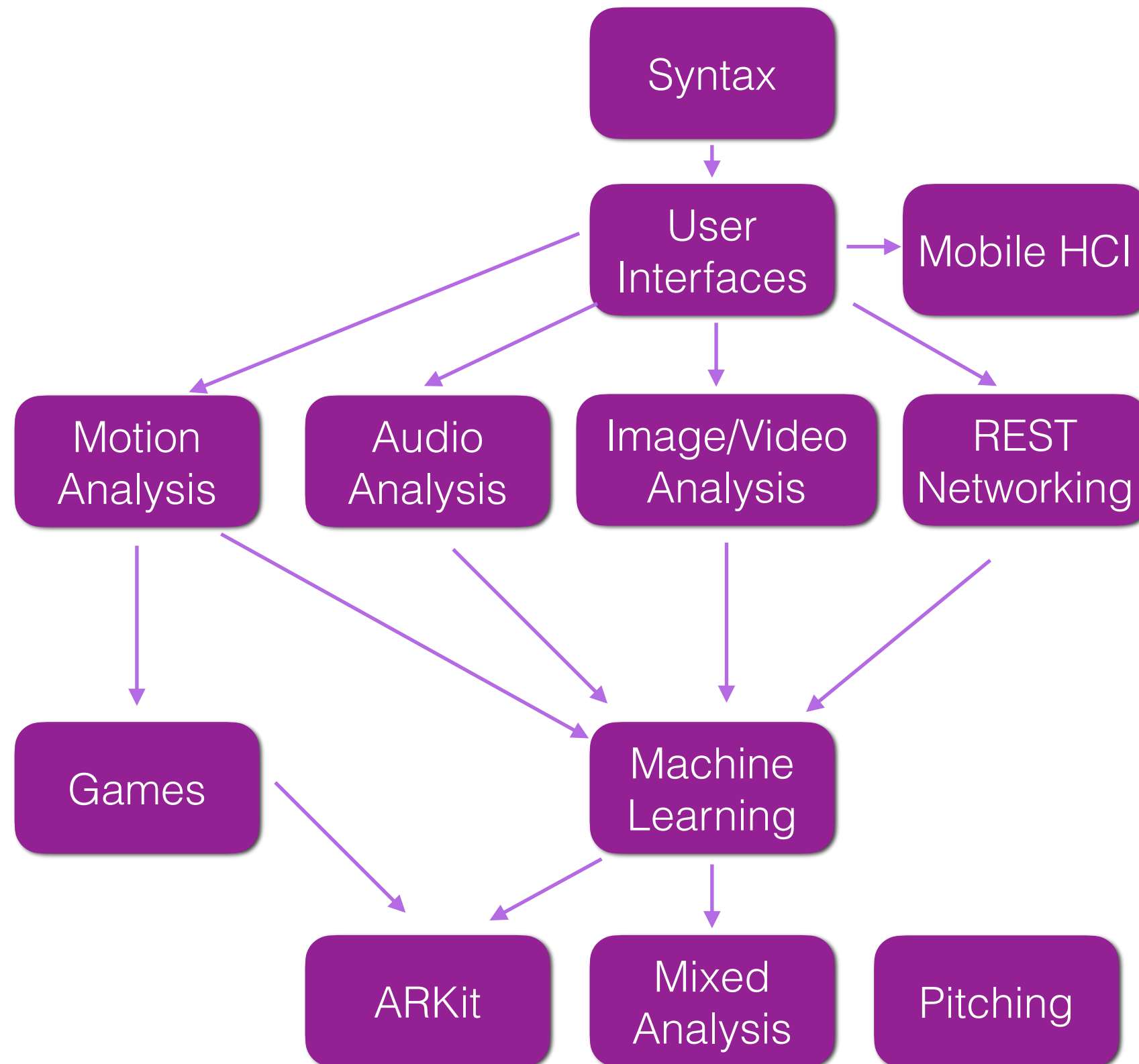
- mobile sensing
 - activity recognition **yes, via developer APIs!**
 - audio analysis **yes, custom sample access!**
 - vision analysis **yes, via multiple APIs!**
- machine learning **yes! treated as black box**
- microcontroller communication **no, archived**
- general iOS development **some but won't get you a job...**
- UI/UX, animation and graphics **not really... but a little**
- VR/AR and games **some, part of bonus lecture content**

learning to learn

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



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** (I grade comments)
- embrace generative coding, but **learn from it**

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)

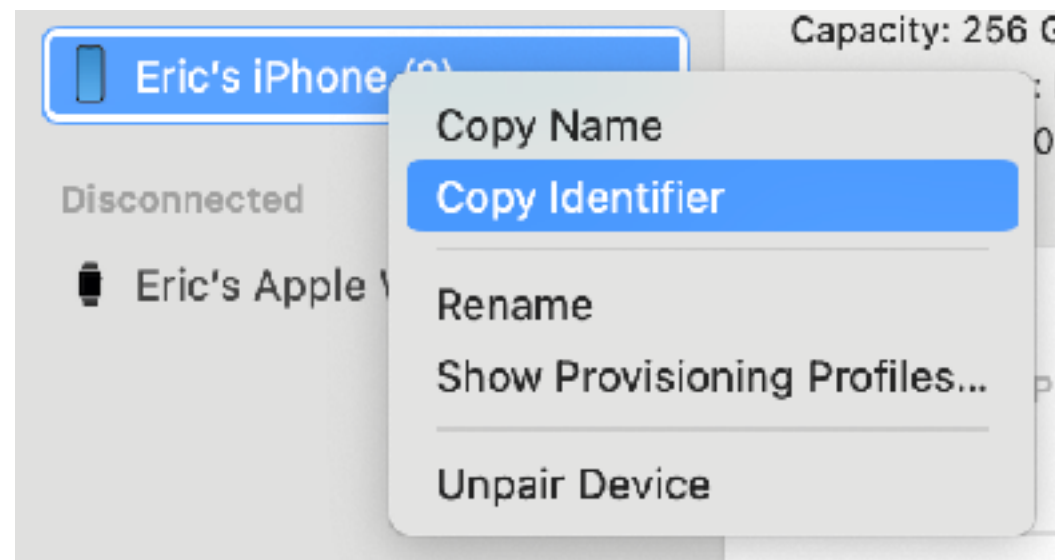
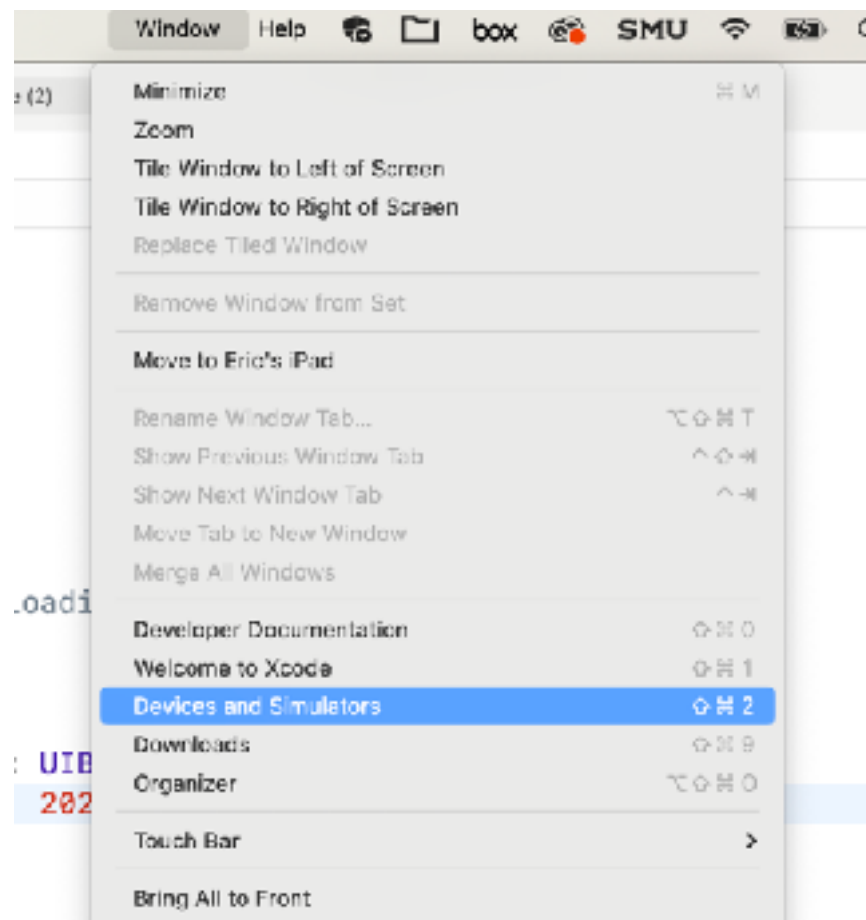
- grading
- flipped assignments
- final projects
- MOD

before next class

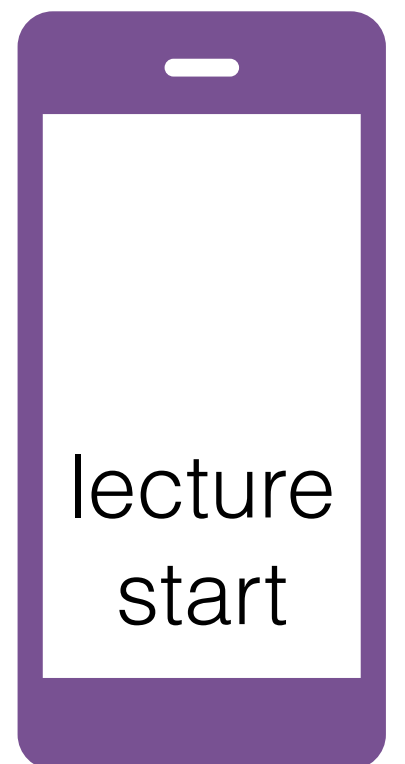
- look at canvas and GitHub repository (clone first repository)
- get a team together (groups of 1-4, no exceptions)
 - contribute **equally**, **everyone** codes, **everyone** designs
 - **pick good members** with different skills than you
 - take turns **coding**
 - **you can change teams throughout semester**
- send me information for the developer program
- all assignments are already posted for the semester and all flipped module videos

Apple Developer Program

- university developer program has been discontinued
- I will use my personal license, so send me:
 - **add user:** email that you want invite sent to (requires sign up)
 - **add device:** identifier, Xcode “Window>Devices and Simulators”



developing in iOS



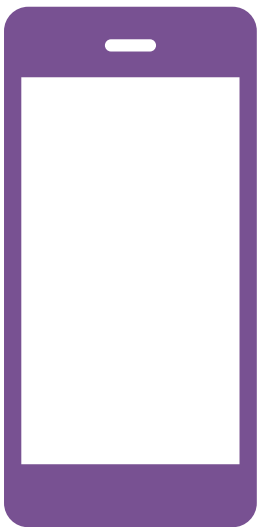
developing in iOS

- **cross platform**

- React native, flutter, 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, using Apple IDE (what we will be using)**

- (-) free to develop, but requires license to run on hardware
- (+) packages are well supported, constantly updated
- (+/-) limited to ONLY Apple (but also optimized for hardware)
- (+/-) Xcode with objective-c and swift



Xcode overview

change view

Tabs Navigation

Apple Intelligence will release during the Semester

Navigation,
Git,
debugging,
building,
profiling

Active Window
(code, storyboard, settings)

Help,
properties,
and
Inspection

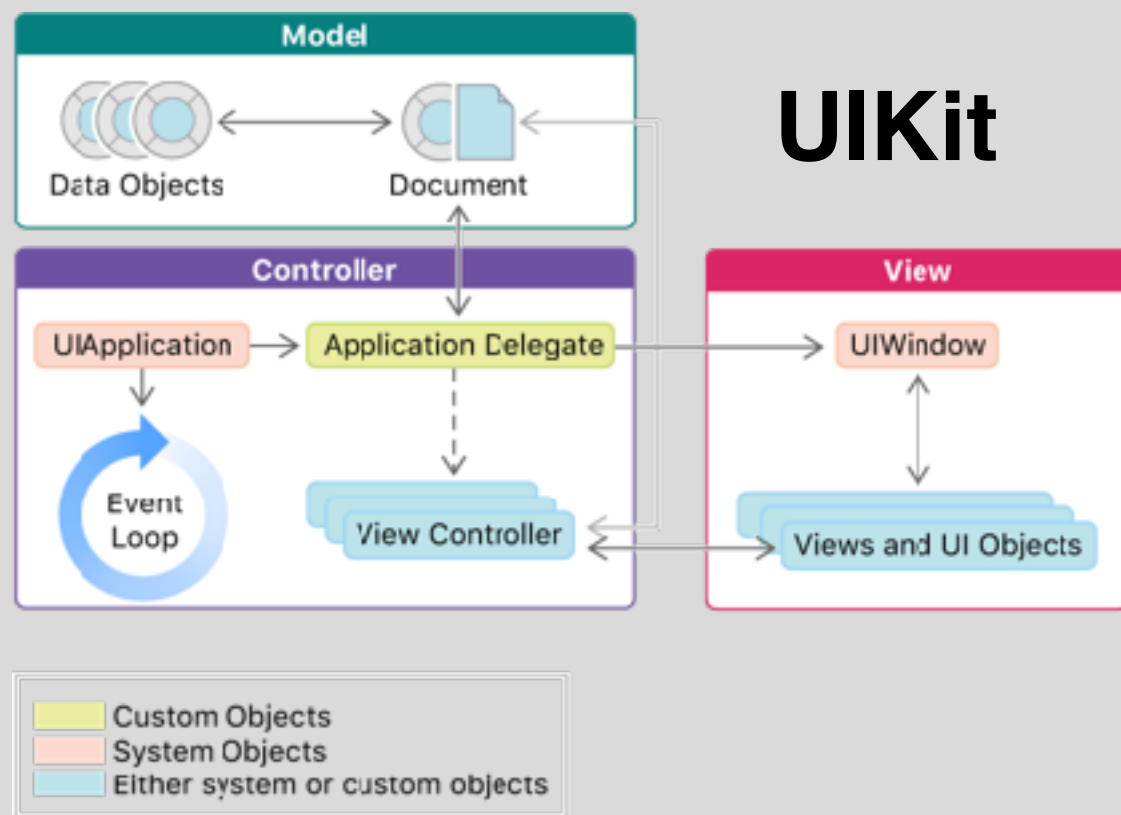
Debugging,
watched variables

Console output
from iPhone or
Simulator

Xcode

- best shown by example
- a very powerful and complex IDE
- if you update iOS, you must update Xcode
 - some Xcode updates require the newest MacOS
- support UI design through **SwiftUI** or **UIKit**
 - my examples will use UIKit exclusively (easier to get started)
 - required use of storyboards and auto layout
 - auto layout is great for apps in one layout

UIKit versus SwiftUI

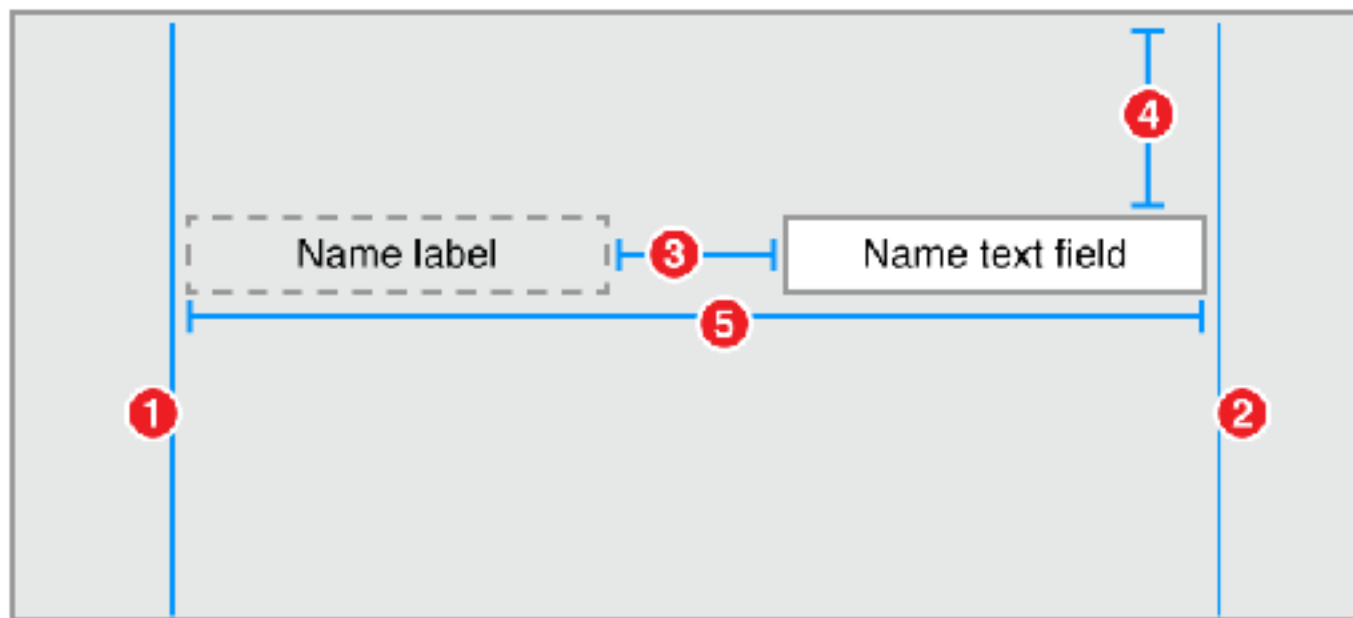


- **UIKit** assumes develop in code and storyboard
- Requires Xcode UI to layout interfaces and link code to view
- Easy to learn and get apps running
- Harder to master for full app development

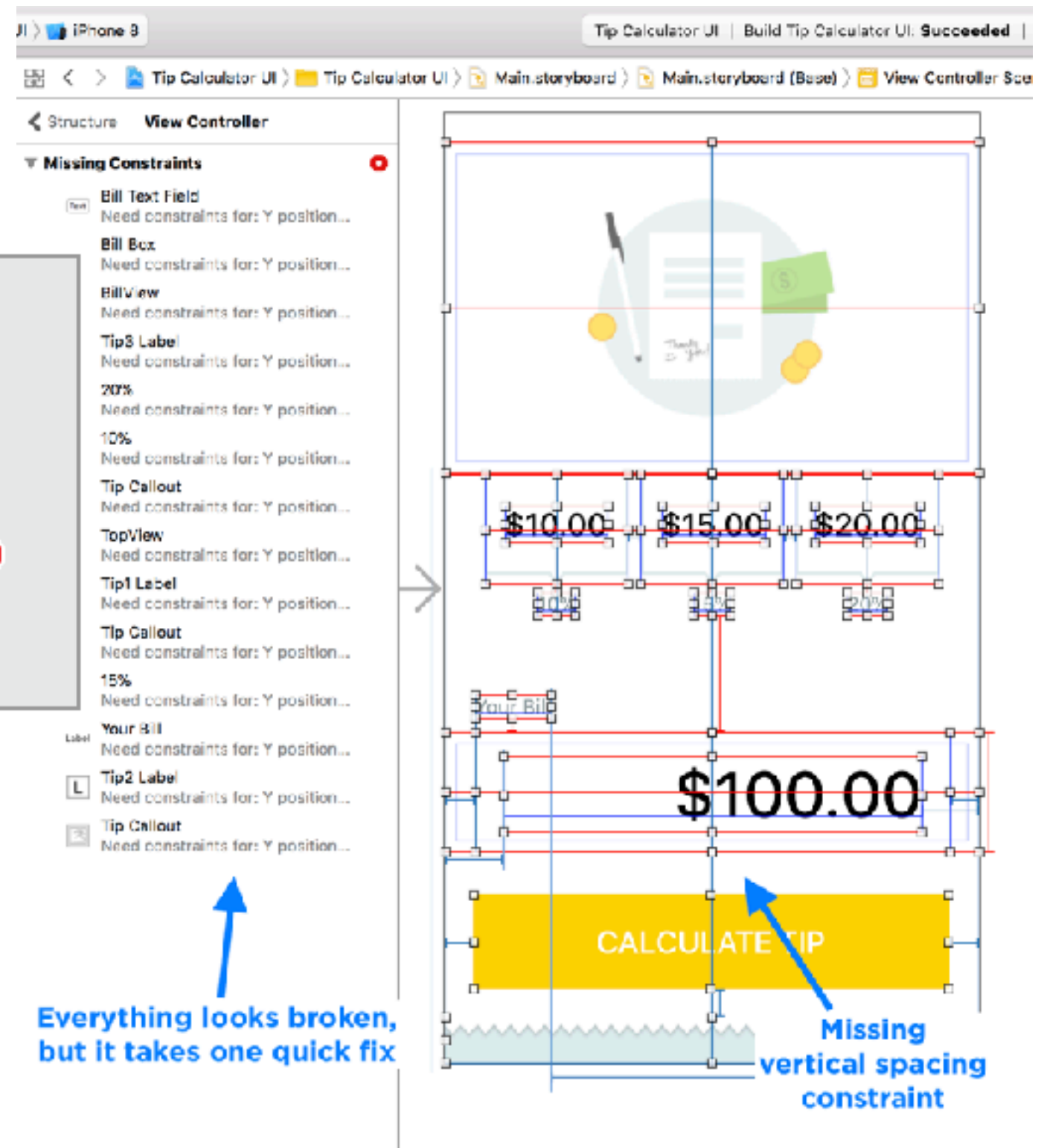


- **SwiftUI** has a more steep learning curve
- Requires learning a new style of coding for laying out views
- Integrates with UIKit and can be mixed

auto layout with storyboard

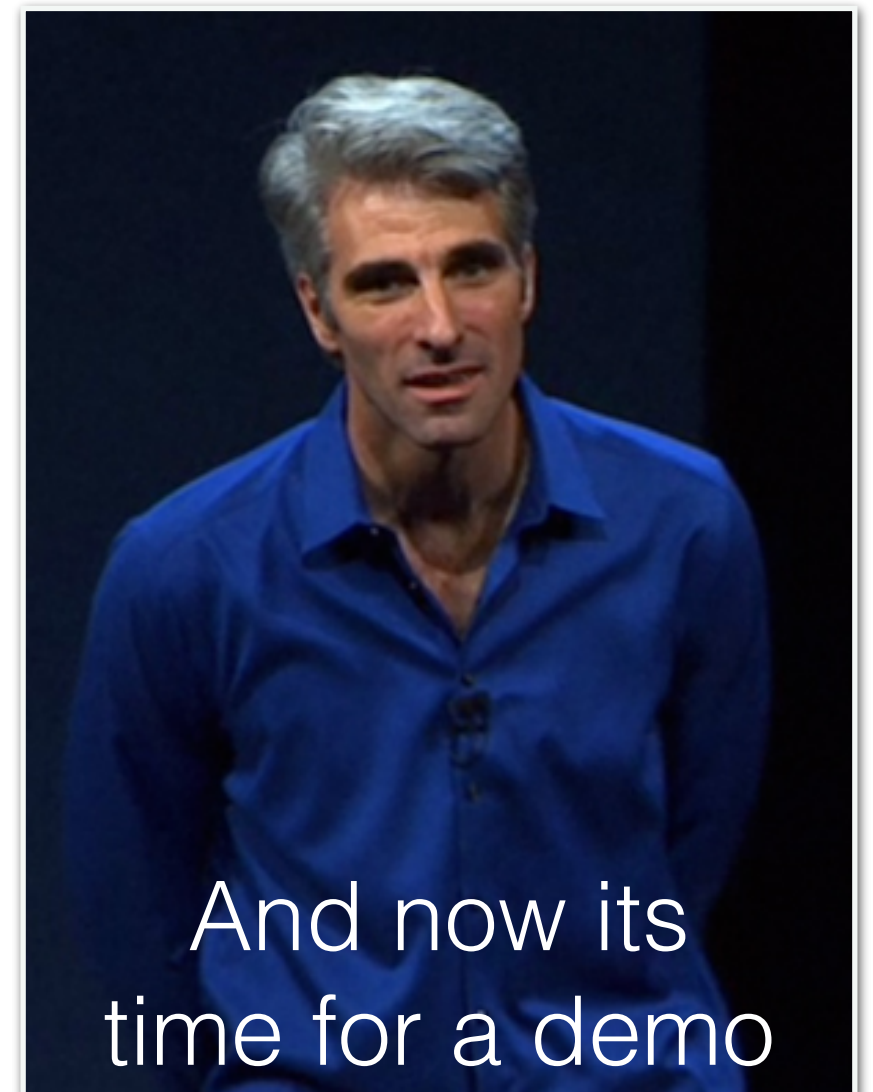


This is the worst part of using UIKit, but necessary



our first app with Xcode

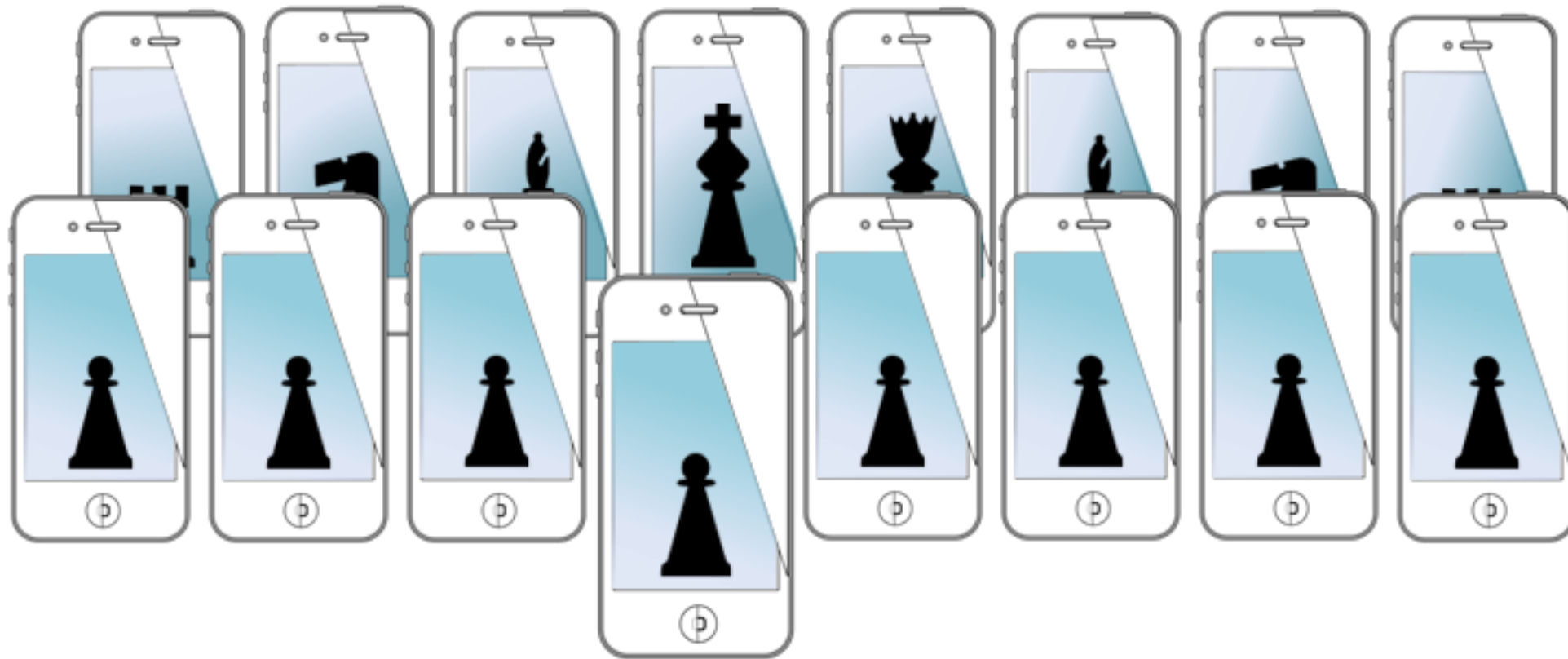
- provides GUI for most git commands
 - commit, branch, push, pull, etc.
- **rarely** is command line needed
- git is great for code but not storyboards
- and some auto layout too!



for next time...

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

MOBILE SENSING & LEARNING



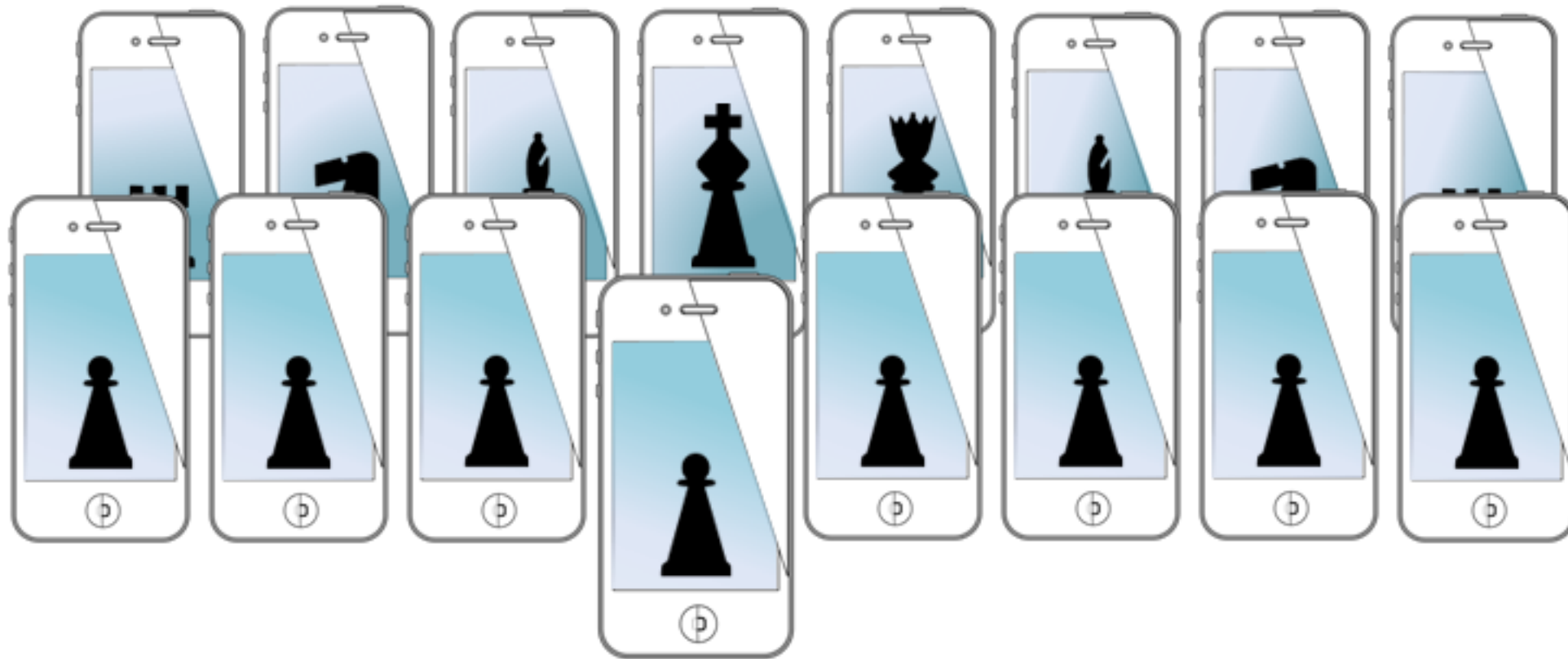
CS5323 & 7323

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

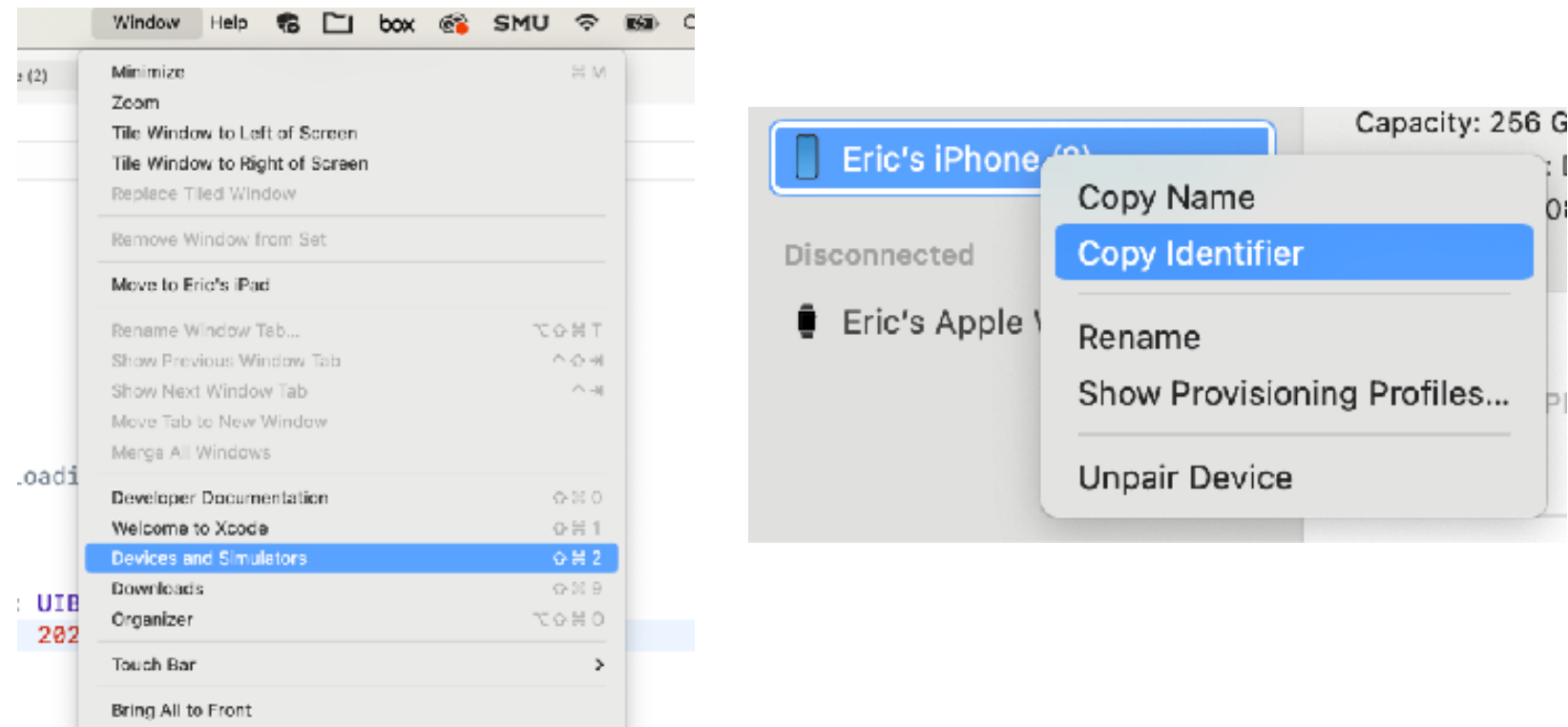
Eric C. Larson, Lyle School of Engineering,
Department of Computer Science, Southern Methodist University

course logistics

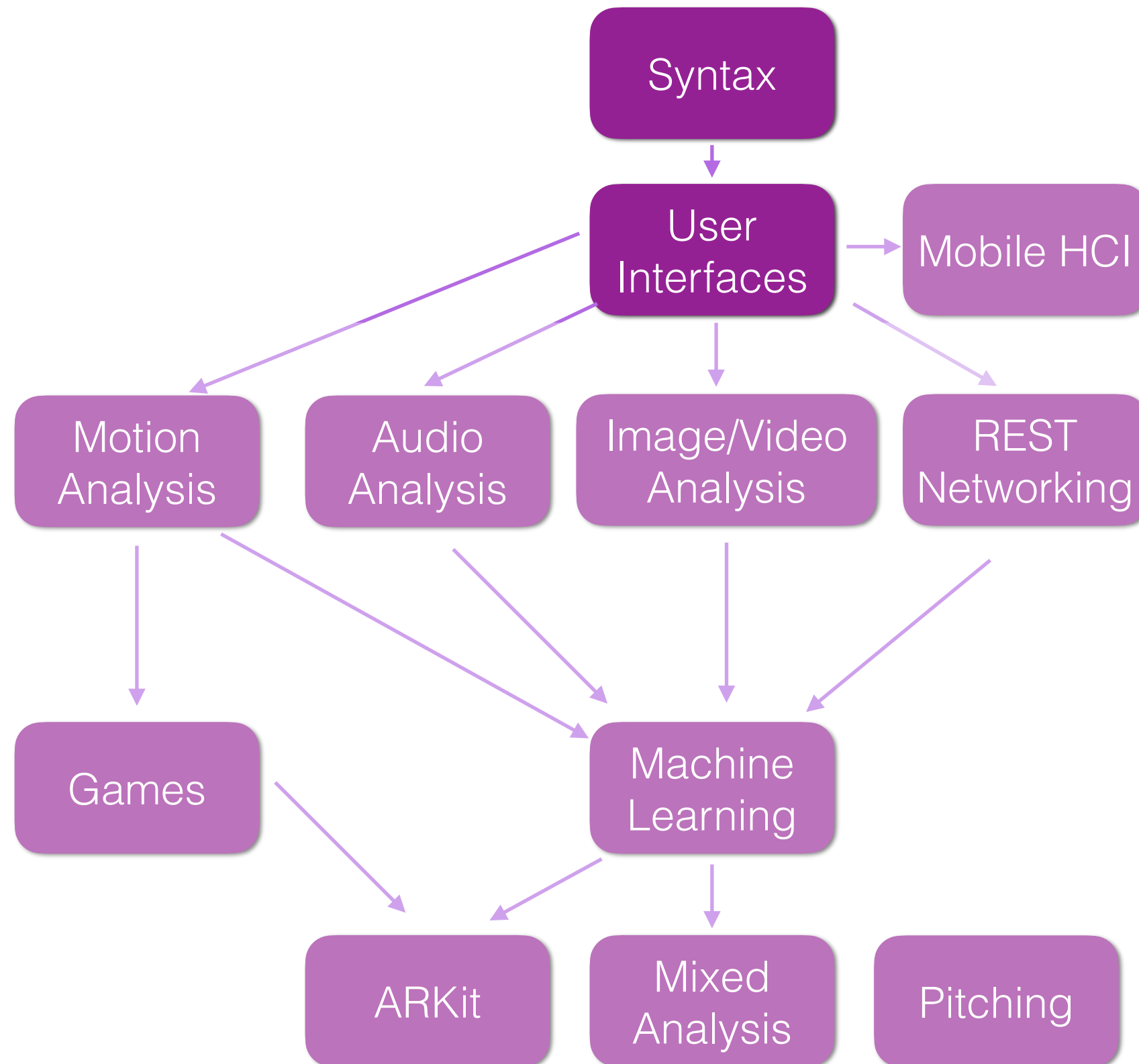
- reminder: no lab this semester
- teams: **get in a team now!** Teams can change throughout semester,
- **equipment checkout:** Phones, macs (must be on a team)
- enrollment in 5000 versus 7000 (ugrad/grad)
- Reminder: Zoom versus in-person and other classes
- Panopto videos access

Apple Developer Program

- if you update iOS, you must update Xcode (maybe MacOS)
- university developer program has been discontinued
- I will use my personal license, so send me:
 - **add user:** email that you want invite sent to (requires sign up)
 - **add device:** identifier, Xcode “Window>Devices and Simulators”



class progression



agenda

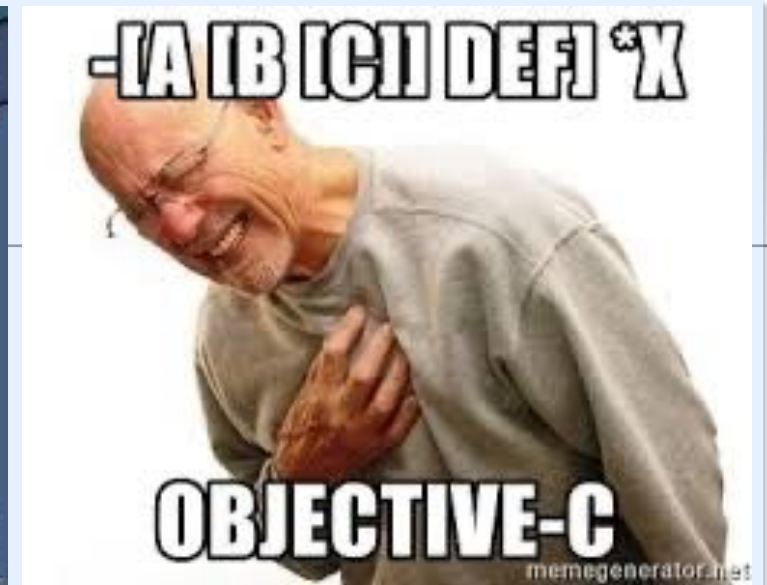
a big syntax demo...

- **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

```
@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);
}
```



```
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

mutable? name:Type = Value

```
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;           shorthand @" "  
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

Declared in the **.swift** file

class name

inherits from

```
class SomeClass : NSObject{
    var aPublicString = "...";
    private var aPrivateString = "...";
    // implementation stuff
}
```

swift defaults to **public properties**
must use **private** keyword

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 = "..."
```

property declared in
class directly

```
    var noDefaultVal: Int
```

```
    override init() {
```

```
        self.noDefaultVal = 0
```

```
    }
```

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

```
    lazy var aString = "Default val if not set"
```

```
    lazy var aStringAlso = {
```

```
        // could do other things here
```

```
        return "Value"
```

```
    }()
```

lazy instantiation,
set to values if accessed

```
    var watchedVariable: Float = 0.0 {
```

```
        willSet(newValue) {
```

```
            print("setting value to \(newValue)")
```

```
        }
```

```
        didSet {
```

```
            print("\(oldValue) set to \(watchedVariable)")
```

```
        }
```

```
    }
```

```
}
```

property observers:
willSet and didSet

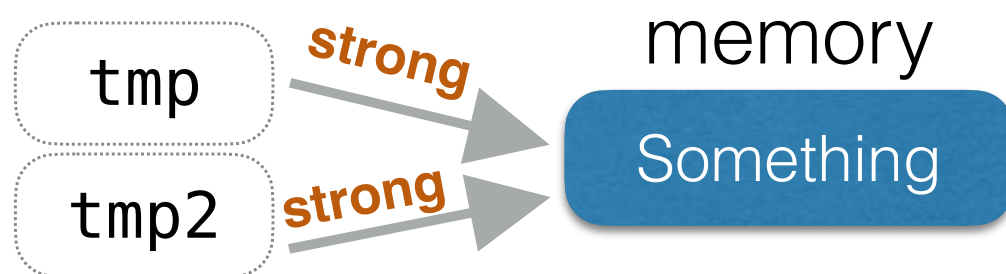
*can also override "set"
and "get" methods, but
this is rare to need*

automatic reference counting

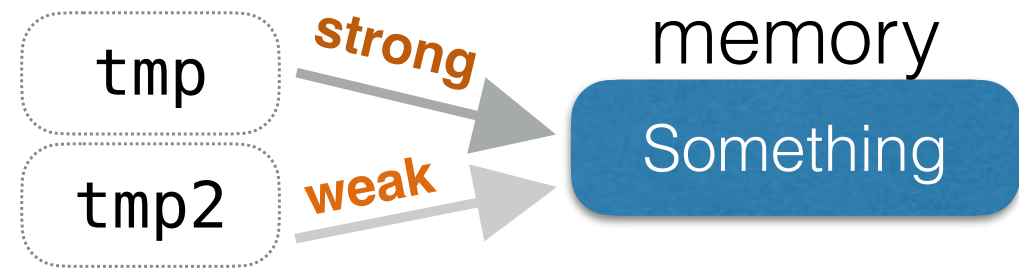
- not garbage collection
- when reference count for variable == 0, trigger event to free memory
 - **strong** pointer adds to reference count
 - **weak** pointer does not add to reference count
 - **unowned** special case of weak, always assumes there is a strong reference with longer lifetime

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

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

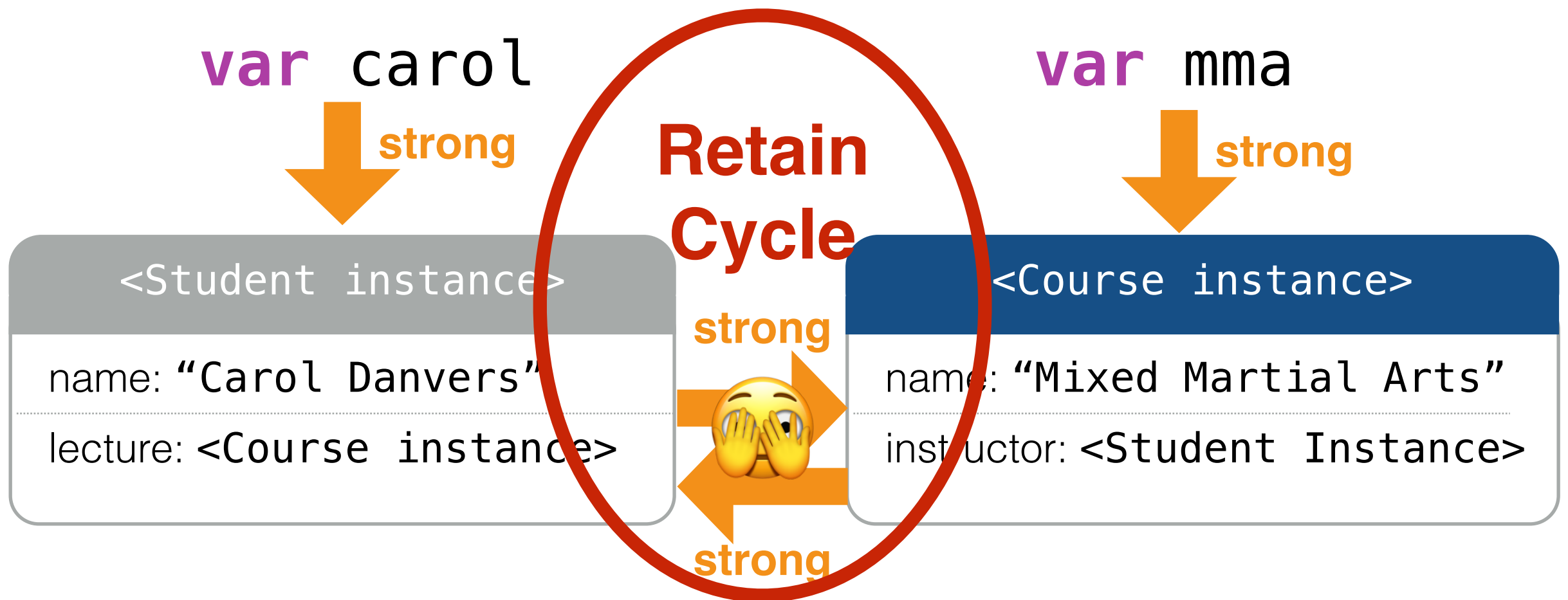


- deallocated after **both references** are nil



- deallocated after **strong reference** is nil

automatic reference counting

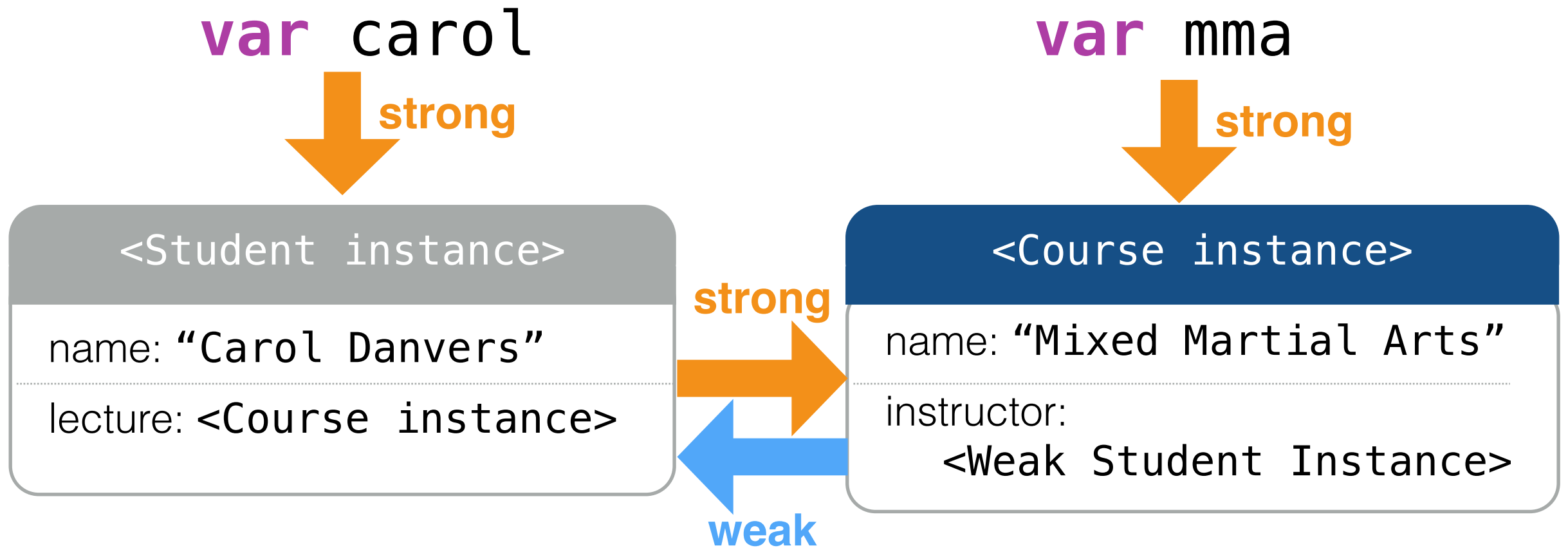


```
carol.lecture = mma  
mma.instructor = carol
```

```
mma = nil  
carol = nil
```

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

automatic reference counting



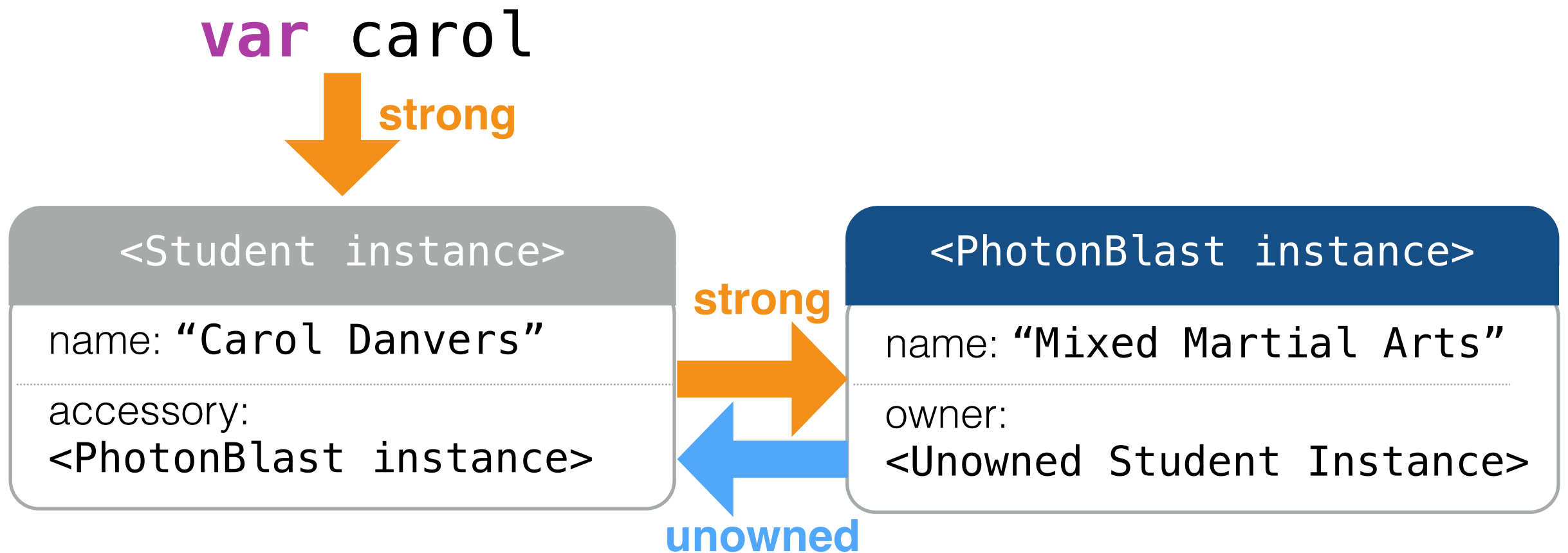
```
carol.lecture = mma  
mma.instructor = carol
```

```
carol = nil  
mma = nil
```

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

```
@IBOutlet weak var someLabel: UILabel!
```

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, 2], [5, 4, 3, 2, 1], 32];  
for(id obj in myArray)  
    NSLog(@"Obj=%@", obj);
```

loop over an NSArray

can store any object

```
@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)  
}
```

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

```
self.aDictionary = ["key1": 3, "key2": "String value"] as [String : Any]  
  
for (_, val) in self.aDictionary {  
    print(val)  
}
```

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
```

receiver class

parameter name/value

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

throwback to **c**

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

```
float val = addOneToNumber(3.0);
```

second (+ —) instance versus class method

```
NSNumber *obj = [NSNumber allocValue:@4];  
[obj addOneToNumber:@4];
```

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

(varName:Type) -> (Return Type)

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

similar named second
parameter syntax in swift

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

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

```
@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



```
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

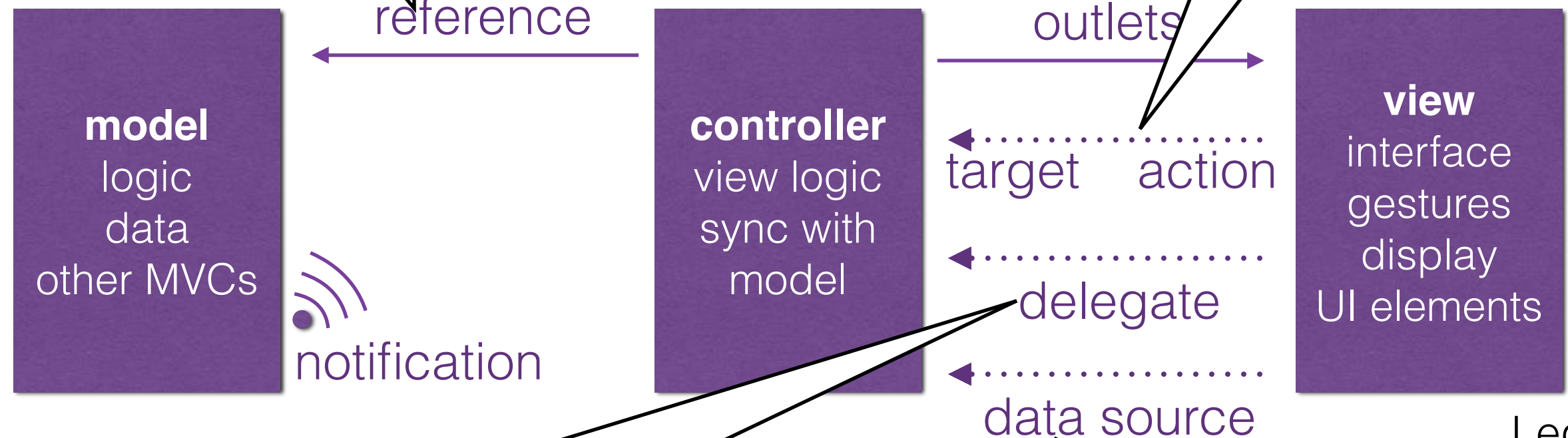
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

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;
```

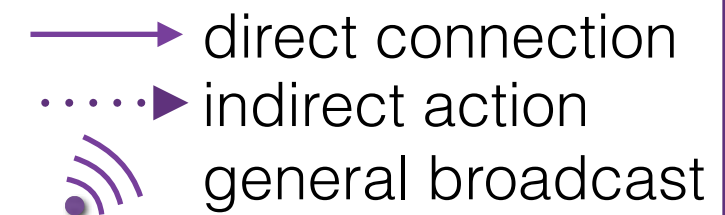


```
MainViewController ()<UITextFieldDelegate>
#pragma mark - UITextField Delegate
- (BOOL)textFieldShouldReturn:(UITextField *)textField { ... }
```

controller implements method for view class

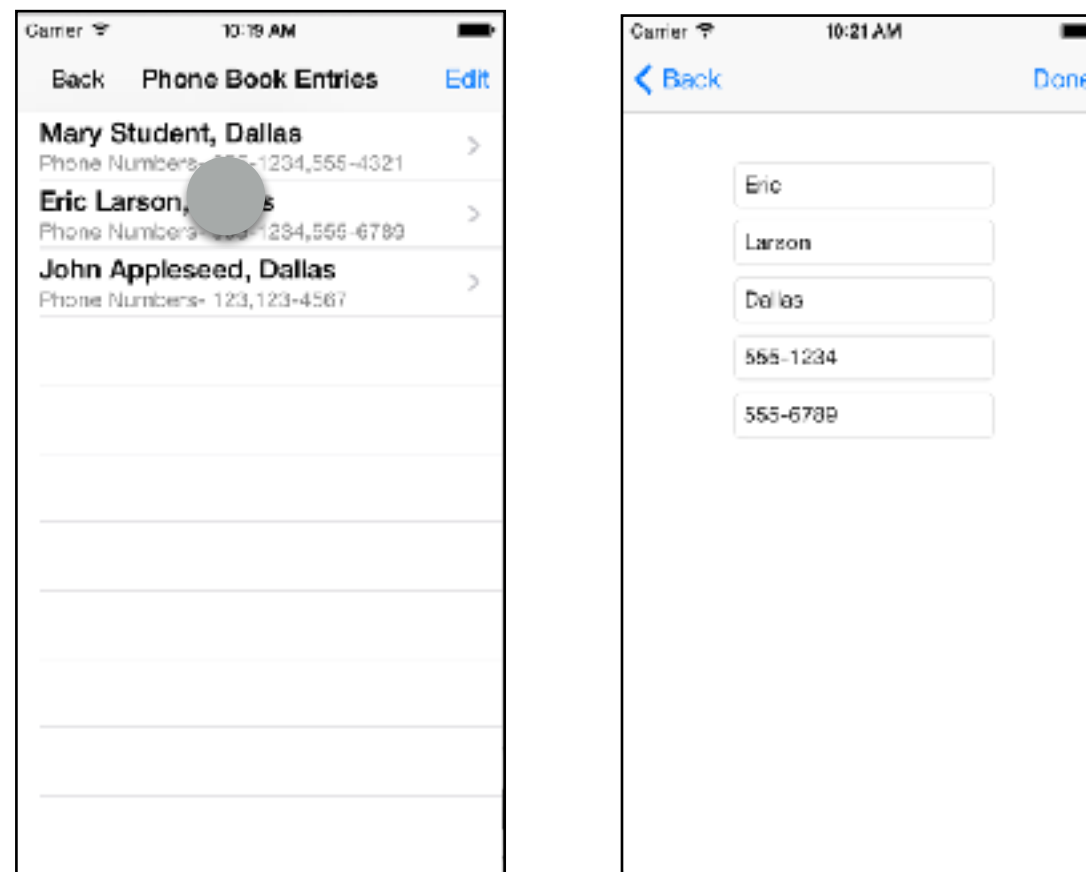
```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSectionSection:(NSInteger)section
```

Legend



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`

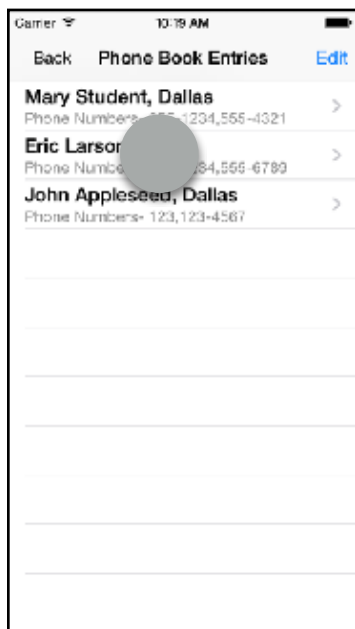
`viewDidAppear`

`viewWillDisappear`

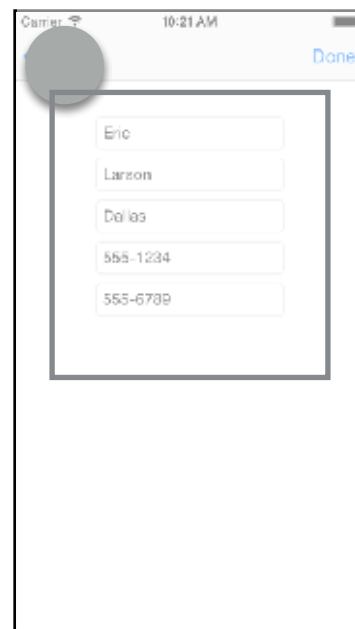
`viewDidDisappear`

memory deallocated when app is ready

source



destination



user

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

custom getter

```
+ (MyCustomClass*)sharedInstance  
{  
    static MyCustomClass * _sharedInstance = nil;  
    static dispatch_once_t oncePredicate;  
    dispatch_once(&oncePredicate, ^{  
        _sharedInstance = [[MyCustomClass alloc] init];  
    });  
    return _sharedInstance;  
}
```

called like a backing variable
in .m file

don't worry about syntax
until next week...

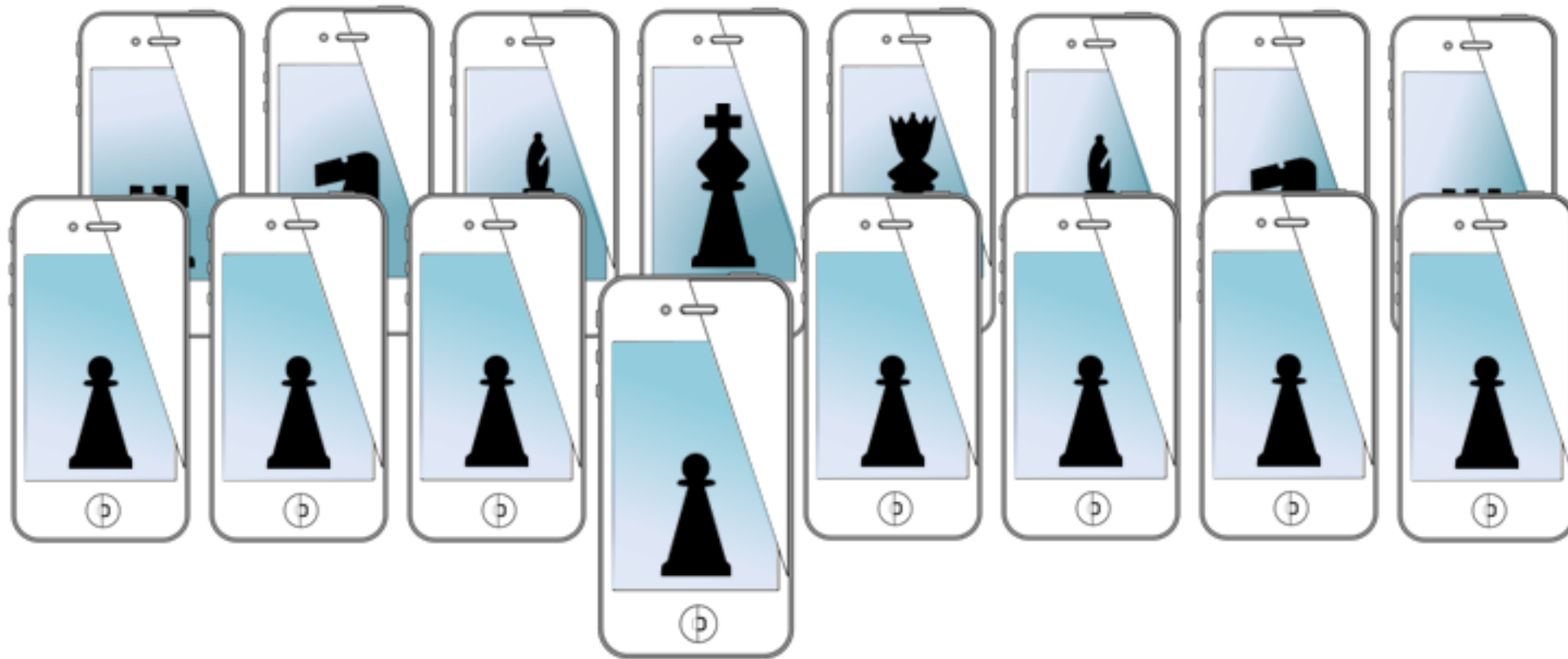
Need more help on MVC's ? Check out Ray Wenderlich:

<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