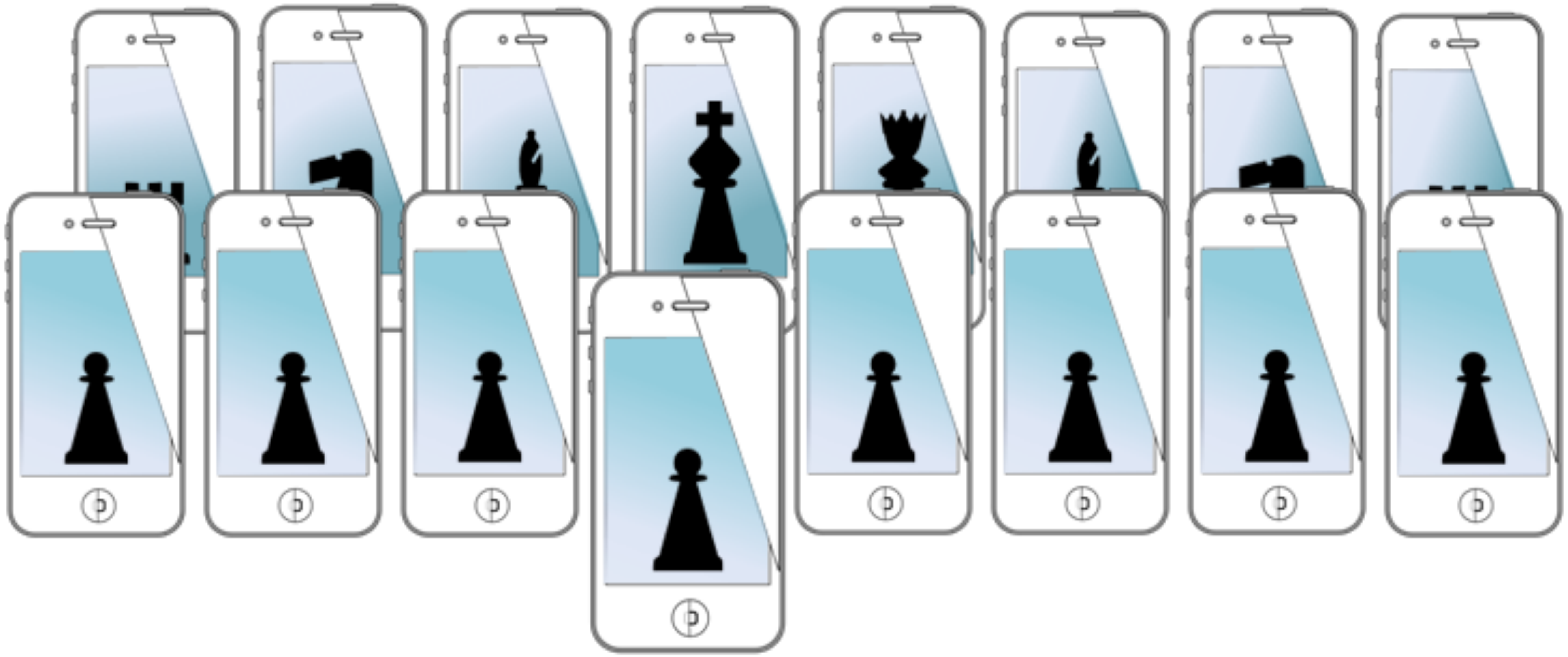
MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

course introduction

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

agenda

- introductions
- class logistics
- what is this mobile sensing course?
  - and what this course is not…
- course goals
- how to do well
- syllabus
  - hardware, lab access, grading, MOD
- Xcode and git

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?
  - natural gestures
    - novel interaction techniques and user interface technology
  - mobile health
    - moving outside the clinic: how mobile sensing can help patients and doctors
  - sustainability
    - how technology can increase awareness

http://eclarson.com

introductions

- about you:
  - name (what you go by)
  - grad/undergrad
  - department
  - something true or false
    - that’s all we have time for…

course logistics

- lab: Tues 5-6:30PM
- we will use canvas for managing the course
- and GitHub for managing code:
  - https://github.com/SMU-MSLC

what is this course (not)

- mobile sensing
  - activity recognition **some, yes!**
  - audio analysis **yes!**
  - vision analysis **yes!**
- machine learning **some, for inference**
- microcontroller communication **yes!**
- general iOS development **some basic skills**
- animation and graphics **no, except to display data**
- user interface design **some, all apps rely on user**

for what we do not cover…

- take the free Stanford iOS course!
- prerequisite: model based coding
- CS students's will find some of this review, EE's will find some of this review — just not at the same time
- creative computation? creativity and design are well rewarded in this class

course goals

- exposure to iOS development, MVCs
- understand how to **use embedded sensors**
- **exposure** to **machine learning** for mobile sensors
  - new: more use of built-in ML in iOS
- real time analysis of data streams
    - applications in mobile health
- **present** and **pitch** applications

how to do well

- complete the lab assignments on time
    - there is no such thing as a late assignment
- start the **lab assignments early, with your team**
- iterate and test your apps
- use good coding practices, lazy instantiation, recycle classes, get on Apple’s developer website for more info
- have fun—seriously
- collaborate, collaborate, collaborate
- and come to class

syllabus

- attendance
  - required for lecture, optional for lab
  - video of classes are not available
- hardware is available for checkout
  - need a team formed (do this before the end of the week)
  - mac minis (password protected, do not change)
  - iPhones (password protected, do not change)
  - you can use your own stuff, but will need iPhone 5S or better
- Now let’s head over to canvas

syllabus

- grading
- in class assignments (flipped assignments)
- final projects
- MOD

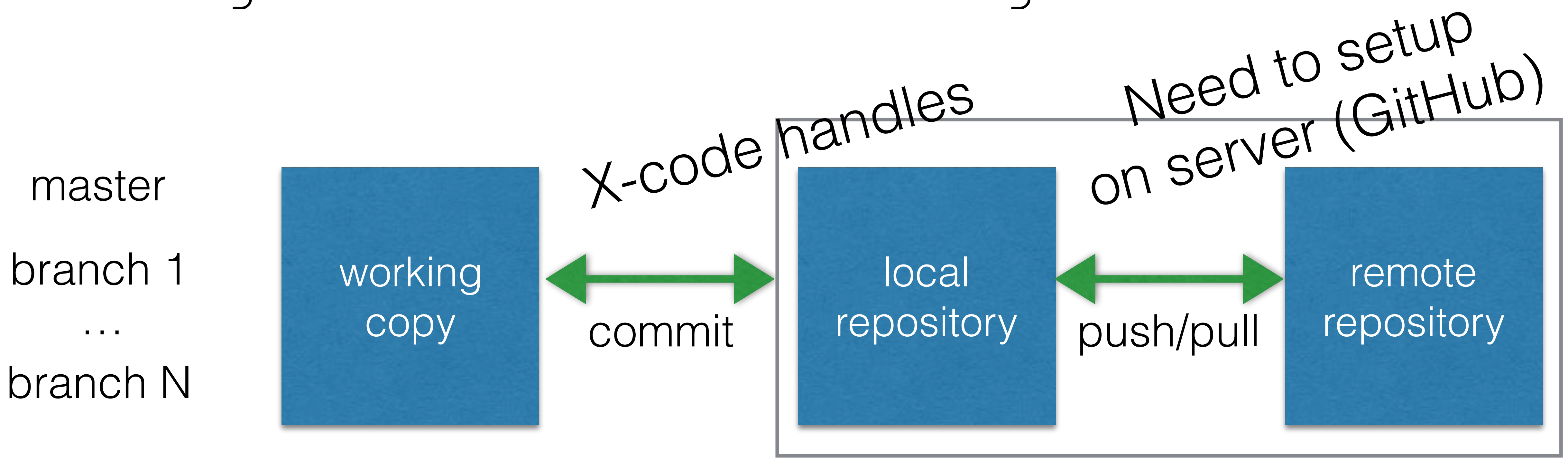
before next class

- look at the class website
- 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 **watching each other code** (I know…)
  - use the lab time for coding together!!!
- assignment 1 is already up!
  - let’s check it out…

Xcode and git

- built into unix (and therefore OSX) and Xcode
- use it when developing with teams or just by yourself
- branching, merging, and all the jazz

```
git init
git add .
git commit -m"starting commit"
```



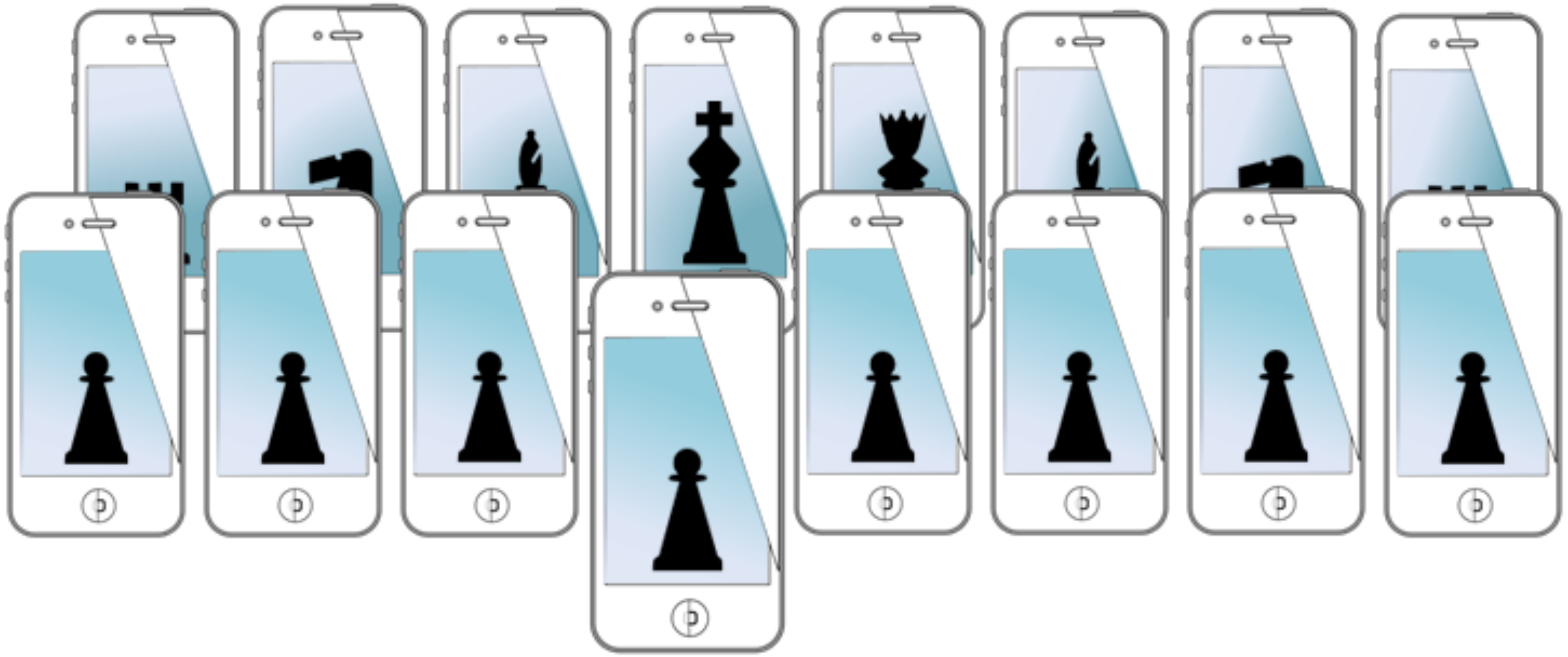
git with Xcode

- provides GUI for most git commands
  - commit, branch, push, pull, etc.
  - plays nicely with submodules
- **rarely** is command line needed
- git is great for code!!
- but not great for storyboards …

for next time…

- have teams figured out
- so hardware can be checked out (to team)

MOBILE SENSING & LEARNING



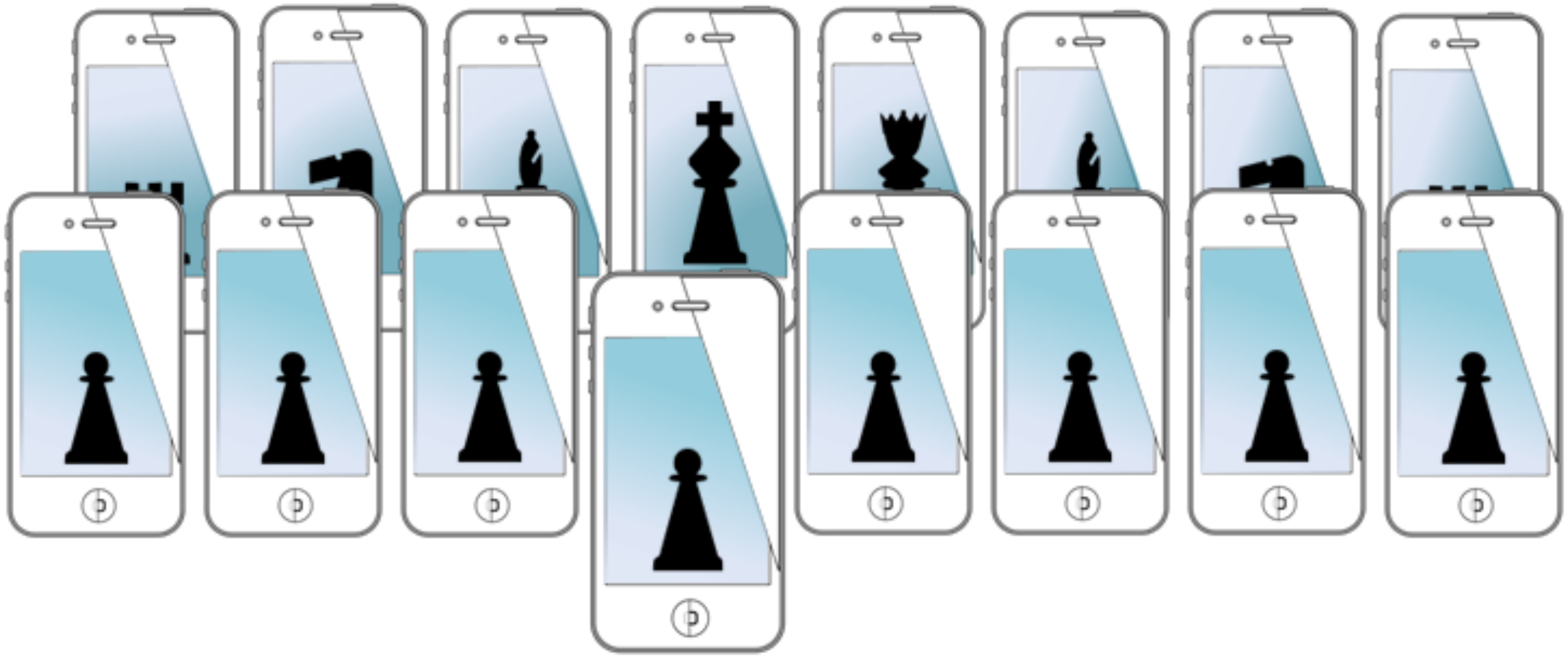
CS5323 & 7323

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 and MVC

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

course logistics

- lab time: Tuesday 5-6:30
- teams: **must be on a team now**!
- equipment checkout:
  - mac mini's? iPhones?
- university developer program:
  - send me an email if you want an invite
    - email that you want invite sent to
    - phone UDID: In Itunes, click the name of the phone for hidden identifier

assignment one

- also posted on Canvas!
  - use a **TableViewController** to load different views
  - TableViewController must implement **three different types of cells and load them dynamically** (cannot use only static cells).
  - View navigation can be hierarchical in any way you want, as long as no loops exist
  - When loading a new view controller your main view controller should hand off information to the controller that is getting created

assignment one

due date posted on canvas

- Automatic Layout
- Buttons, Sliders, and Labels
- Stepper and Switch
- Picker (you must implement picker delegate)
- Segmented Control
- Timer (which should repeat and somehow update the UIView)
- ScrollView (with scrollable, zoomable content)
- Image View
- Navigation Controller
- Collection View Controller
- Table View Controller with dynamic prototype cells
- **Refer to the rubric on canvas for full list of required items**

agenda

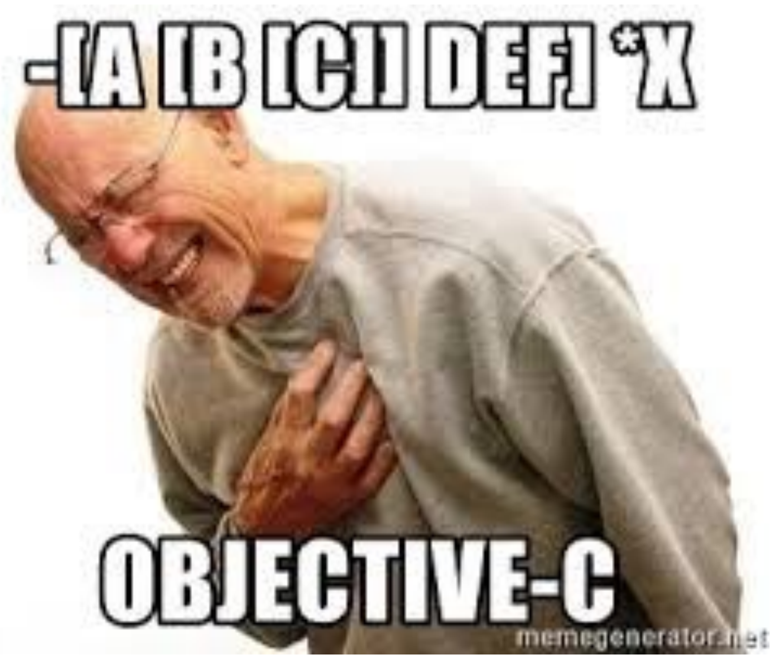
a big syntax demo…

- **objective-c** and (maybe) swift basics
  - class declaration
  - complex objects
  - common functions
  - encapsulation and primitives
  - memory management

and model view controllers for a breather!!!
…also available on video…

objective c

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



objective c variables

```
NSString *aString;
NSNumber *aNum;
NSArray *myArray;
NSDictionary *aDictionary;
NSMutableArray *anArrayYouCanAddTo;
```

Next Step **Encapsulated**
Pointers in the Heap

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

Primitives
Direct Access On the Stack

objective c

classes

interface for class

class name

inherits from

```
@interface SomeClass  : NSObject

@property (strong, nonatomic) NSString *aPublicString;

@end
```

property

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

```
@interface SomeClass ()

@property (strong, nonatomic) NSString *aPrivateString;

@end

@implementation SomeClass

    … implementation stuff…
@end
```

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

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: implicit to compiler

setter, **auto** created

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

getter, **auto** created

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

variable via property	`self.aString`
variable	`_aString`

getter, **custom** overwrites

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

lazy instantiation

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 Property Access, No getter Called to Set Var”;

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

 }


@end
```

What does this do?

objective c

ARC

```
@interface SomeClass ()


@property (strong, nonatomic) NSString *aString;
@end

@implementation SomeClass
@synthesize aString = _aString;




@end
```

atomic ~ thread safe property access
nonatomic ~ faster access

strong ~ keep a reference
weak ~ no reference

automatic reference counting

not garbage collection
when reference count for variable == 0, immediately free memory

strong is usually what you want, else variable is never allocated

weak is used in scenarios where something else holds a reference

objective c

encapsulation

```
NSNumber *aNum = [[NSNumber alloc]init];
aNum = @3;

NSString *aString = [NSString stringWithFormat:@"The time is always %d past %d",42,9];
aString = @"A string";
```

these are PropertyLists: **serializable**, containers for primitive values

Valid Property Lists: NSData, NSDate, NSNumber (int, float, bool)

```
NSArray *myArray = @[@32,@"a string”,@3.2,@3,@[@9,@42,@32]];
                                    [[UILabel alloc]init]];
for(id obj in myArray)
    NSLog(@“Obj=%@",obj);
```

can store any object

loop over an NSArray

An **Array** of **PropertyLists** is also a **PropertyList**

Dictionary as a class property

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

A **Dictionary** of **PropertyLists** is also a **PropertyList**

Access self

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

objective c

mutable and immutable

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

possible to add objects now

all arrays are **nil** terminated

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

[anArrayYouCanAddTo addObject:someComplexObject];

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

objective c

functions examples

return type

method name

parameter name

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

      return @([myNumber floatValue]+1);
}
```

parameter type

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

receiver class

message

parameter value

throwback to **c**

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

float val = addOneToNumber(3.0);
```

second parameter name

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

second parameter

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

objective c

common functions

function

NSString to format

object to print

```
NSLog(@"The value is: %@",someComplexObject);
```

%@ is print for serializable objects

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

set to nothing,
subtract from reference count

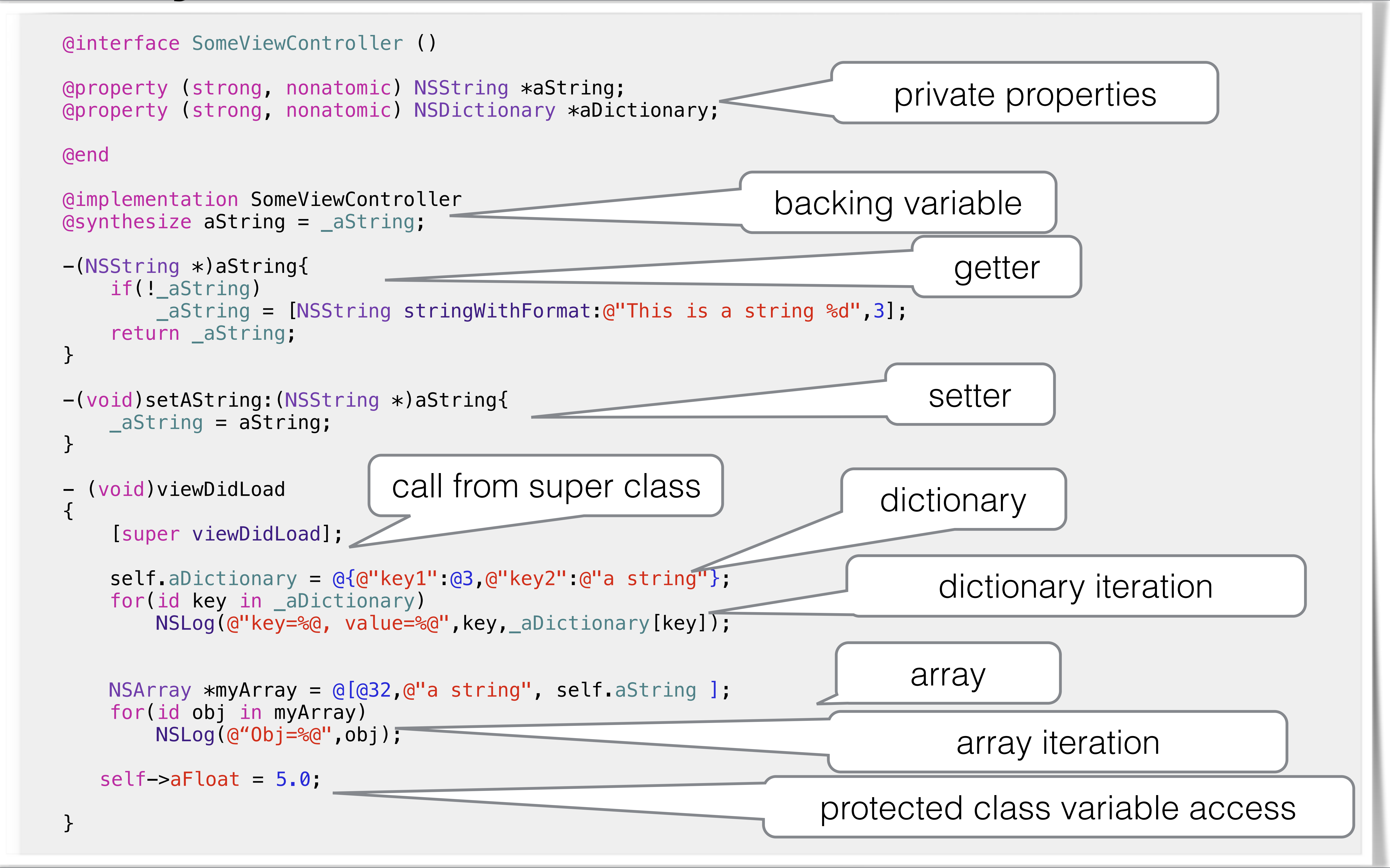
```
    someComplexObject = nil;

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

this means: **if variable is not nil**

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

objective c

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);

    self->aFloat = 5.0;

}
```

adding to our project

- let's add a slider to our project
- and user lazy instantiation
- and some git branching
- and some auto layout

MVC's

controller has direct connection to view class

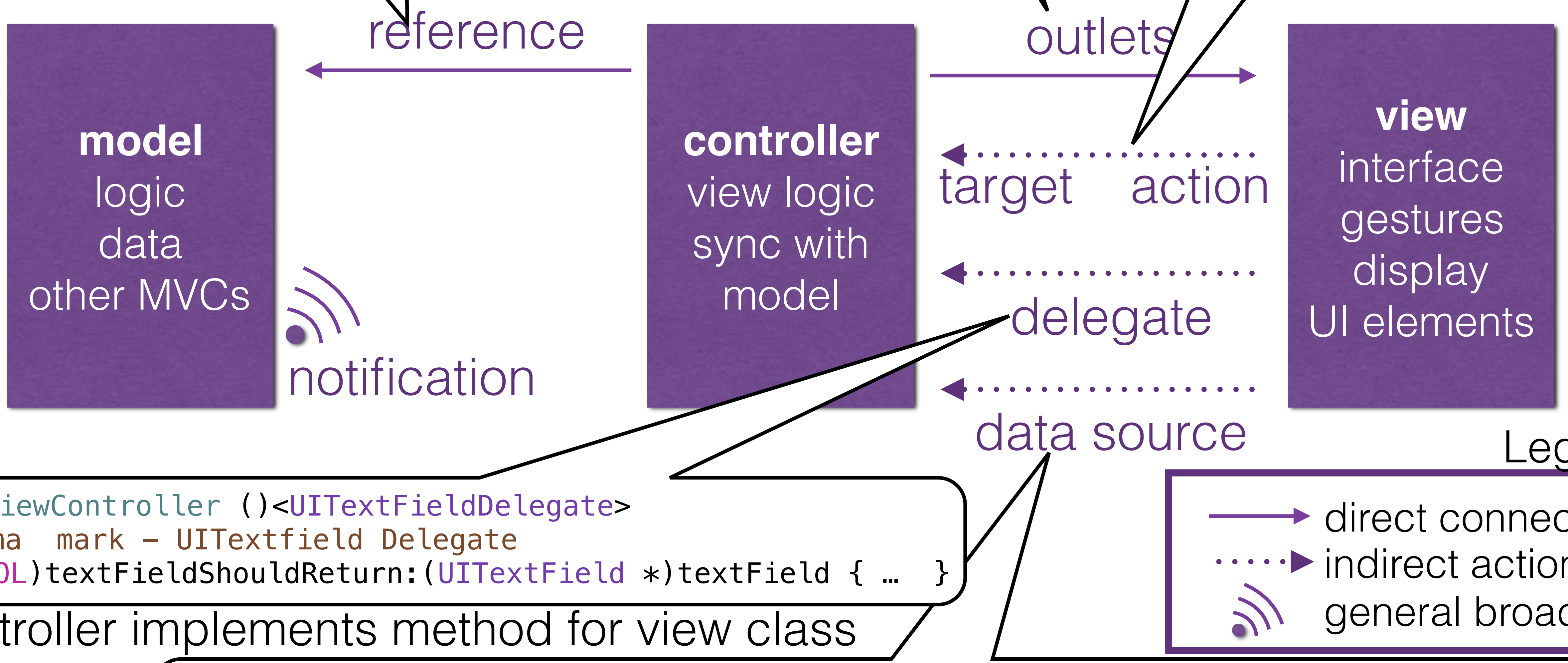
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

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;
```



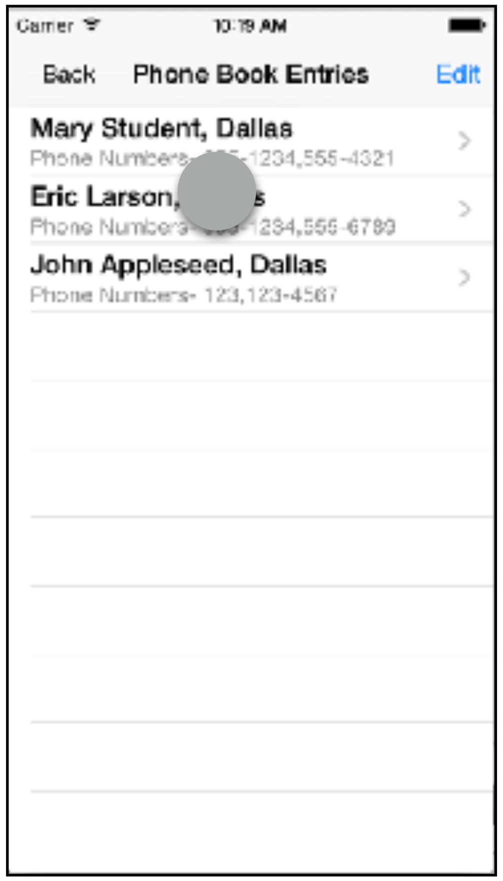
```
MainViewController ()<UITextFieldDelegate>
#pragma  mark - UITextfield Delegate
- (BOOL)textFieldShouldReturn:(UITextField *)textField { …  }
```

controller implements method for view class

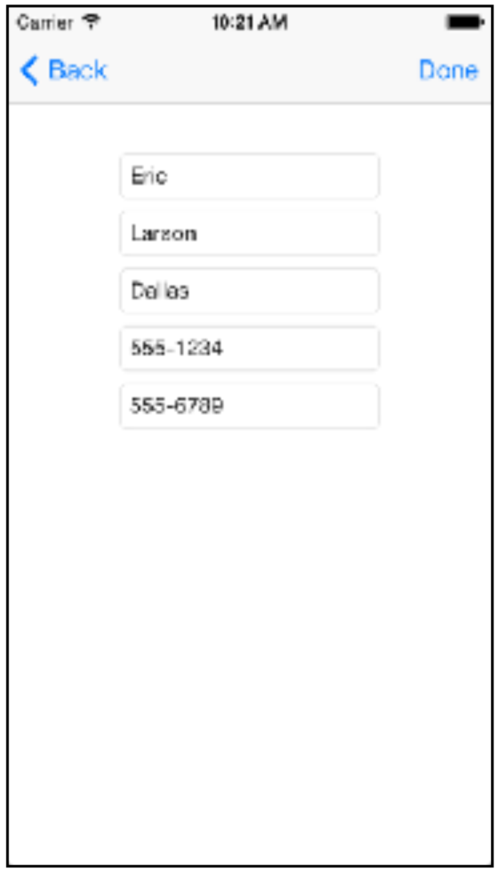
```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

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



Mary Student, Dallas
Phone Numbers- 555-1234,555-4321
Eric Larson, s
Phone Numbers- 1234,555-6789
John Appleseed, Dallas
Phone Numbers- 123,123-4567



Eric
Larson
Dallas
555-1234
555-6789

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`

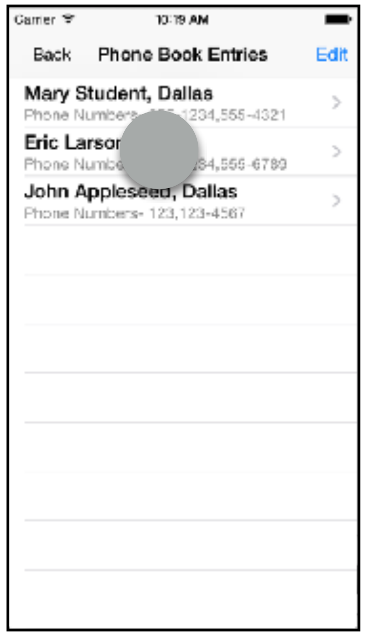
`viewDidAppear`

`viewWillDisappear`

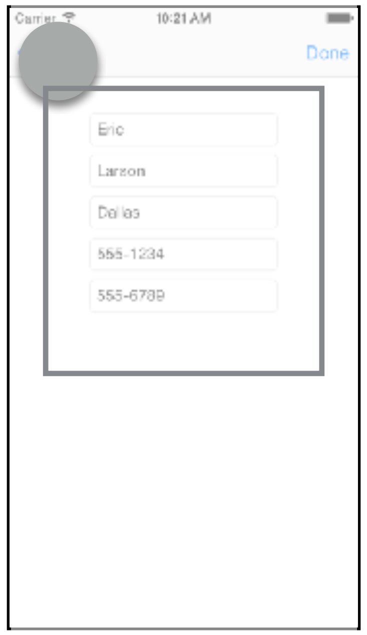
`viewDidDisappear`

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

called like a backing variable

in .m file

```
+ (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…

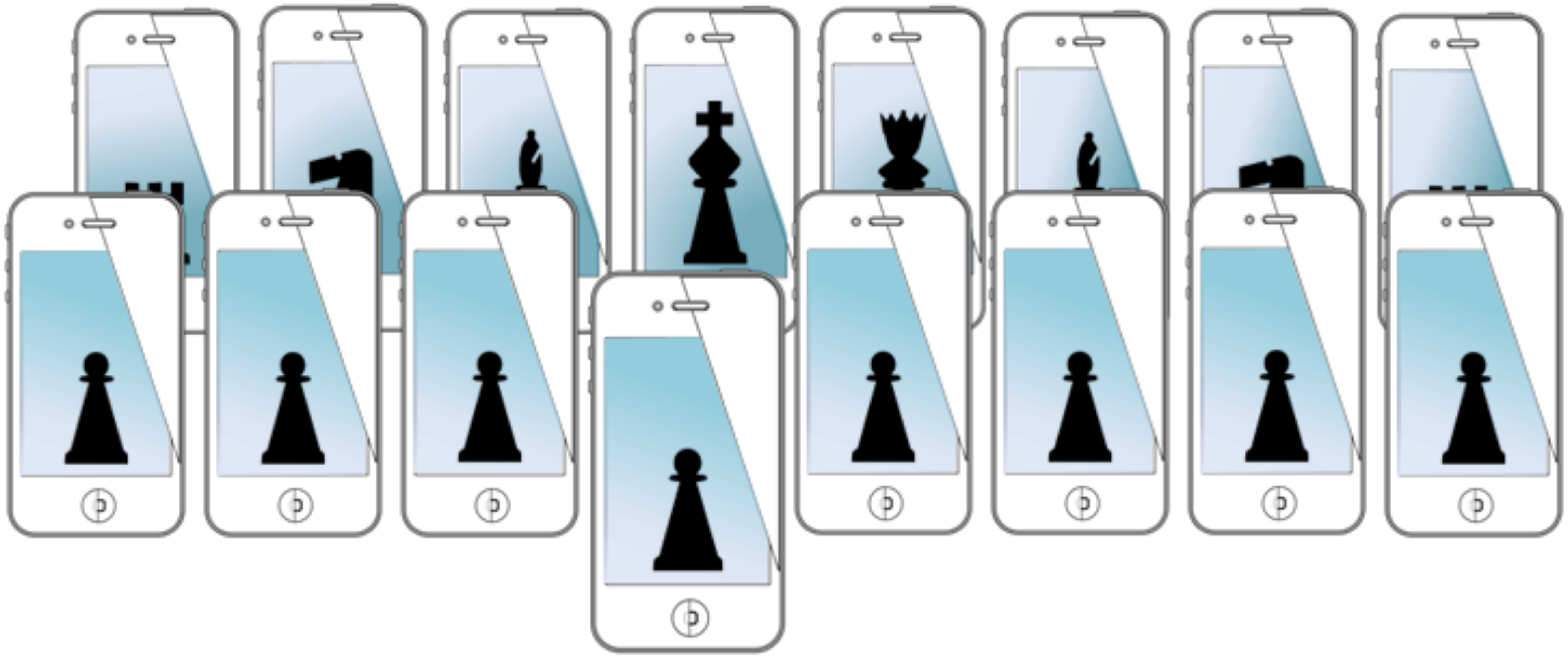
Need more help on MVC’s ? Check out Ray Wenderlich:

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