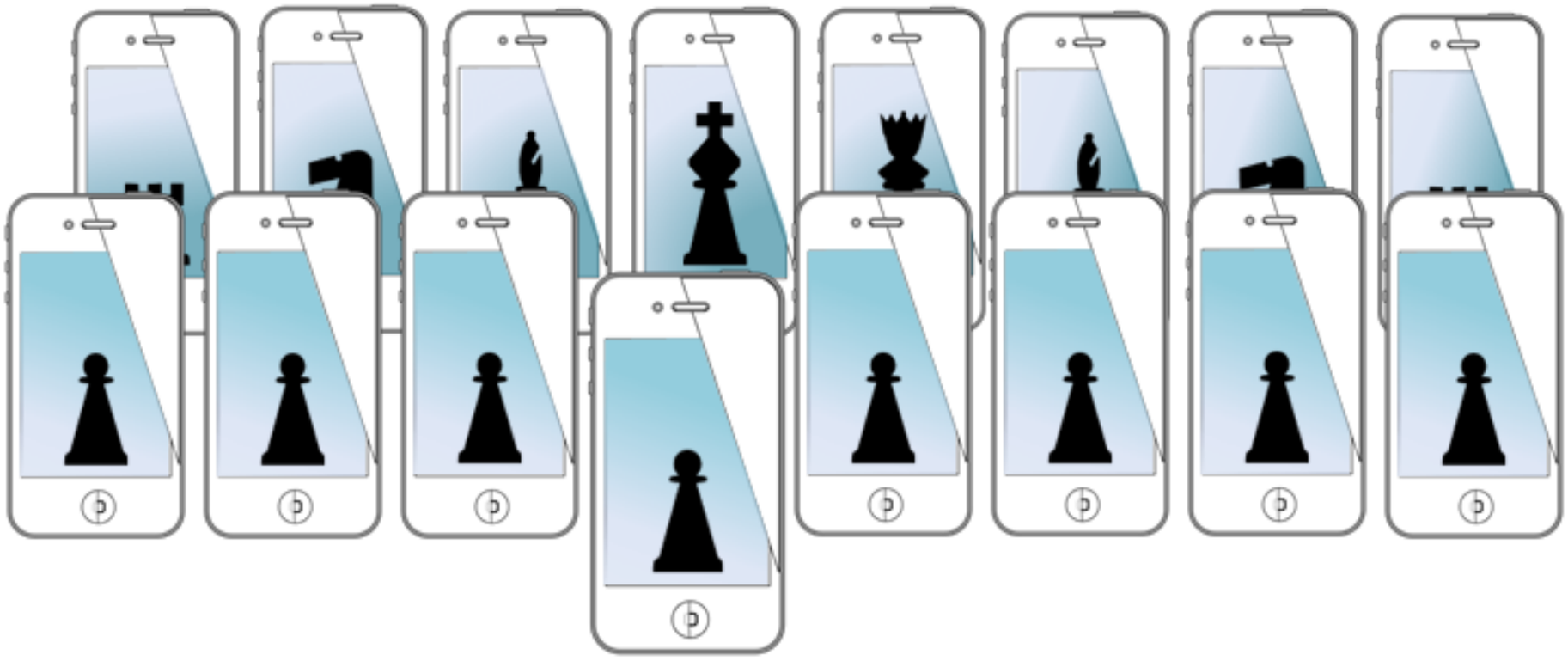
MOBILE SENSING & LEARNING



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
- how to do well
- syllabus
  - hardware, lab access, grading, MOD
- Xcode and git

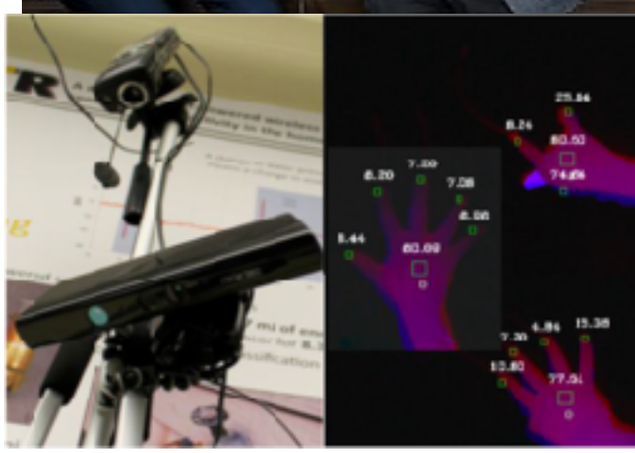
course logistics

- lecture: in this class and via Zoom
- lab: Tues 5-6:30PM (via Zoom only)
- office hours: TBD (via Zoom only)
- 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

- me
  - Eric 👍
  - Dr. Larson 👍
  - Prof. Larson 👍
  - other 👎
- about you:
  - name (what you go by)
  - grad/undergrad
  - department
  - **something true or false**

Choices:

- High School
- College Job
- ML Instructor
- Marvel
- Kids

what is this course (and not)

- mobile sensing
  - activity recognition **some, yes!**
  - audio analysis **yes!**
  - vision analysis **yes!**
- machine learning **some, for inference**
- 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**

for what we do not cover…

- take the free Stanford iOS course!
- prerequisite: model based coding
  - because you will learn at least one new language:
    - objective-c, swift, python, c, c++, objective c++

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
- 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 or attend Zoom!

syllabus

- attendance
  - required for lecture (red/blue cohort)
  - video of classes through panopto (published after class)
- hardware is available for checkout
  - need a team formed (do this before the end of the week)
    - teams are expected to work remotely together
  - mac minis and iPhones available for checkout
  - you can use your own stuff, but will need iPhone 5S or better
- Now let's head over to canvas

syllabus (via canvas)

- grading
- flipped assignments
- final projects
- MOD

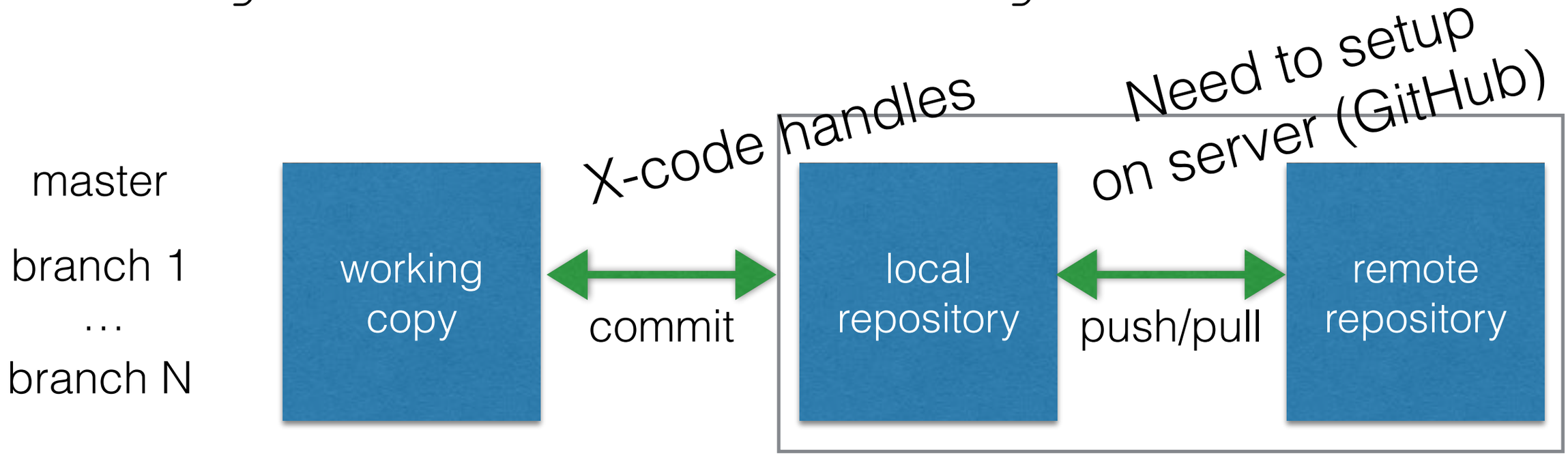
before next class

- 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 in breakout rooms
- assignment 1 is already up!
  - let's check it out…

Xcode and git

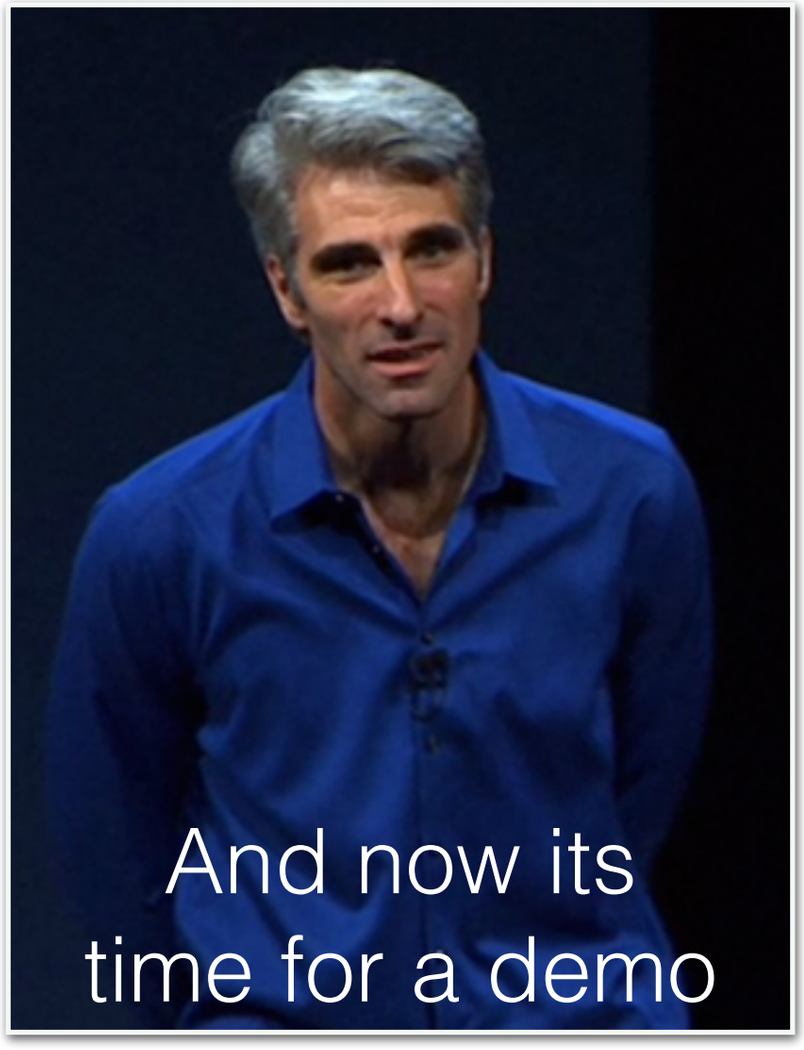
- built into unix (and therefore OSX) and Xcode
- use it when developing with teams or just by yourself
- branching, merging, and all the jazz

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



git with Xcode

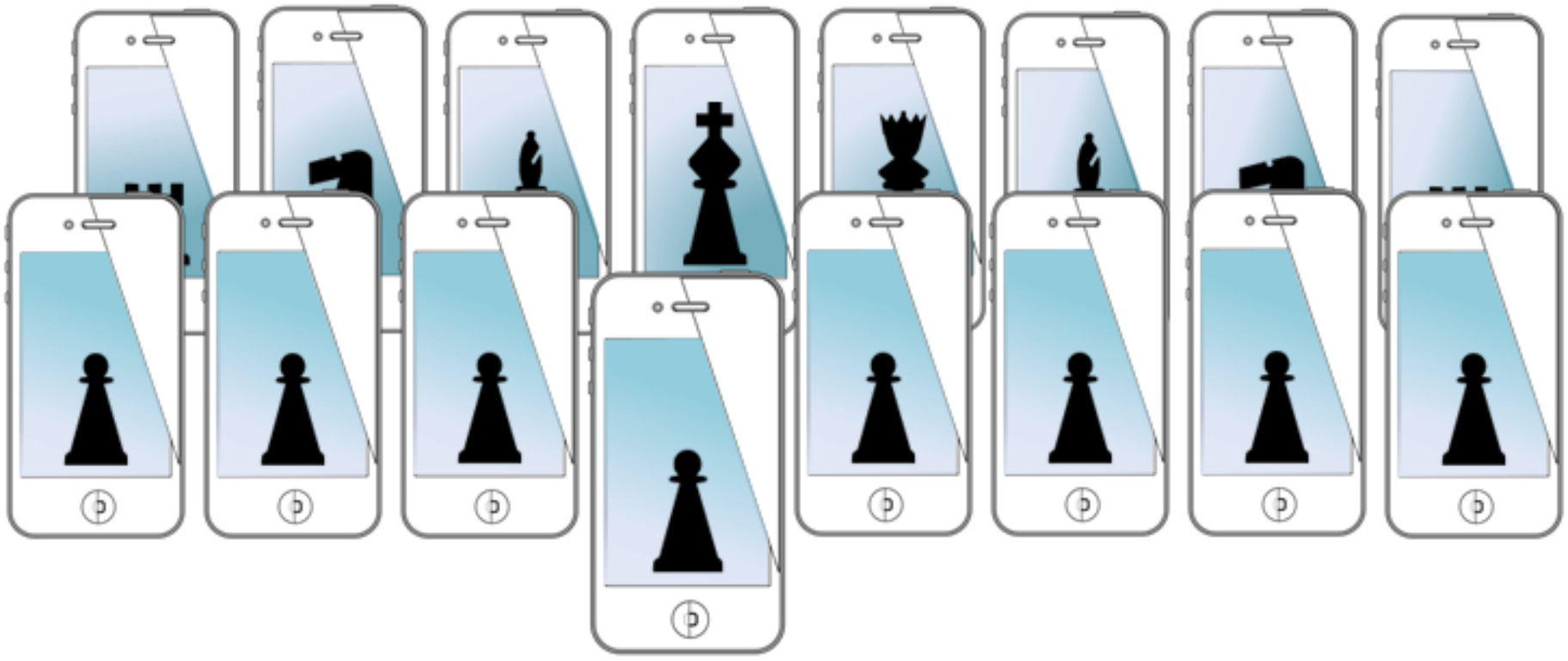
- provides GUI for most git commands
  - commit, branch, push, pull, etc.
- **rarely** is command line needed
- git is great for code!!
- but not great for storyboards …



for next time…

- have teams figured out
- so hardware can be checked out (if you want it)

MOBILE SENSING & LEARNING



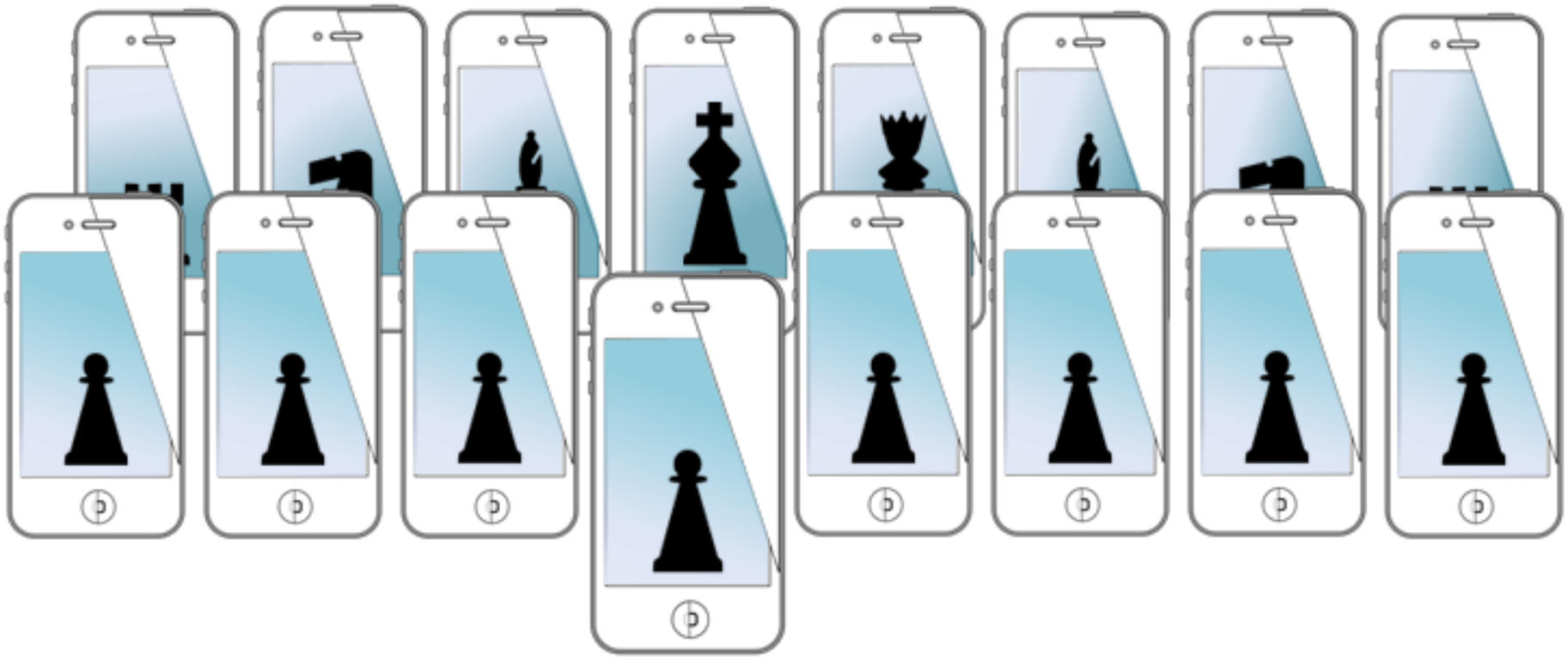
CS5323 & 7323

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 (Zoom)
- class attendance: red/blue cohort
- teams: **should be on a team now**!
- equipment checkout: mac mini's? iPhones?
- 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, click the name of the phone for hidden identifier

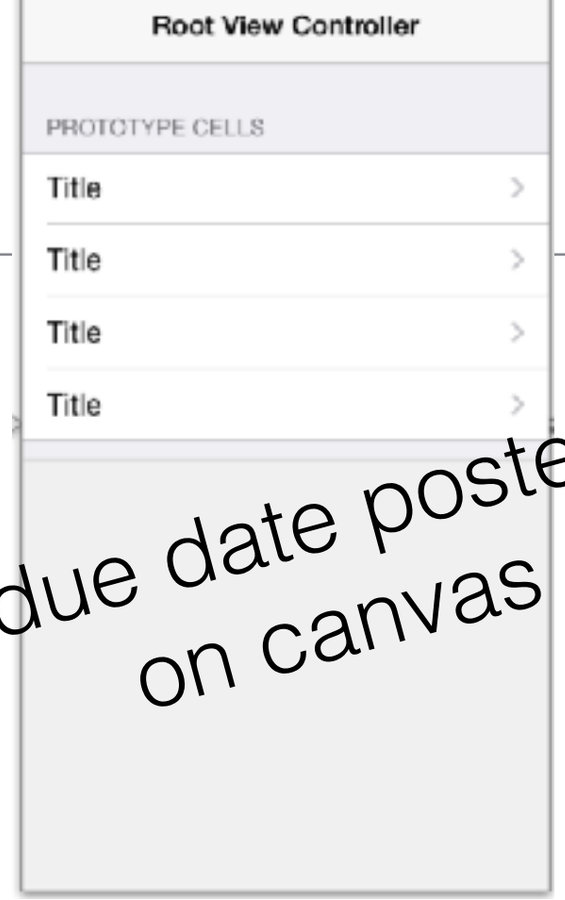
How To Find Your UDID?

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.

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

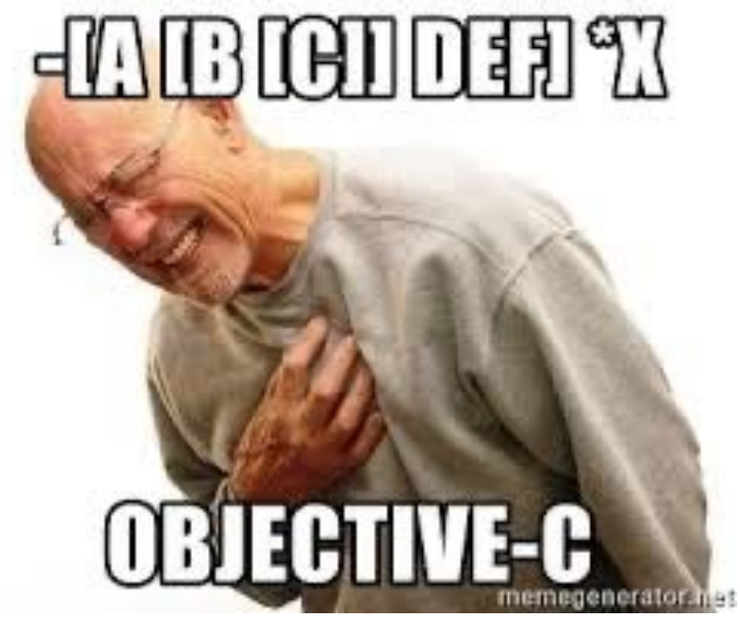
a big syntax demo…

- **objective-c**
  - 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 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
```

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

property

```
@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;
}
```

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 Property Access, No getter Called to Set Var”;

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

 }


@end
```

What does this do?

```
@interface SomeClass ()

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

@implementation SomeClass
@synthesize aString = _aString;
```

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

```
@end
```

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",[UILabel alloc],@3.2];
for(id obj in myArray)
    NSLog(@“Obj=%@",obj);
```

can store any object

loop over an NSArray

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

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

Dictionary as a class property

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);
```

```
someComplexObject = nil;

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

set to nothing,
subtract from reference count

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;

}
```

private properties

backing variable

getter

setter

call from super class

dictionary

dictionary iteration

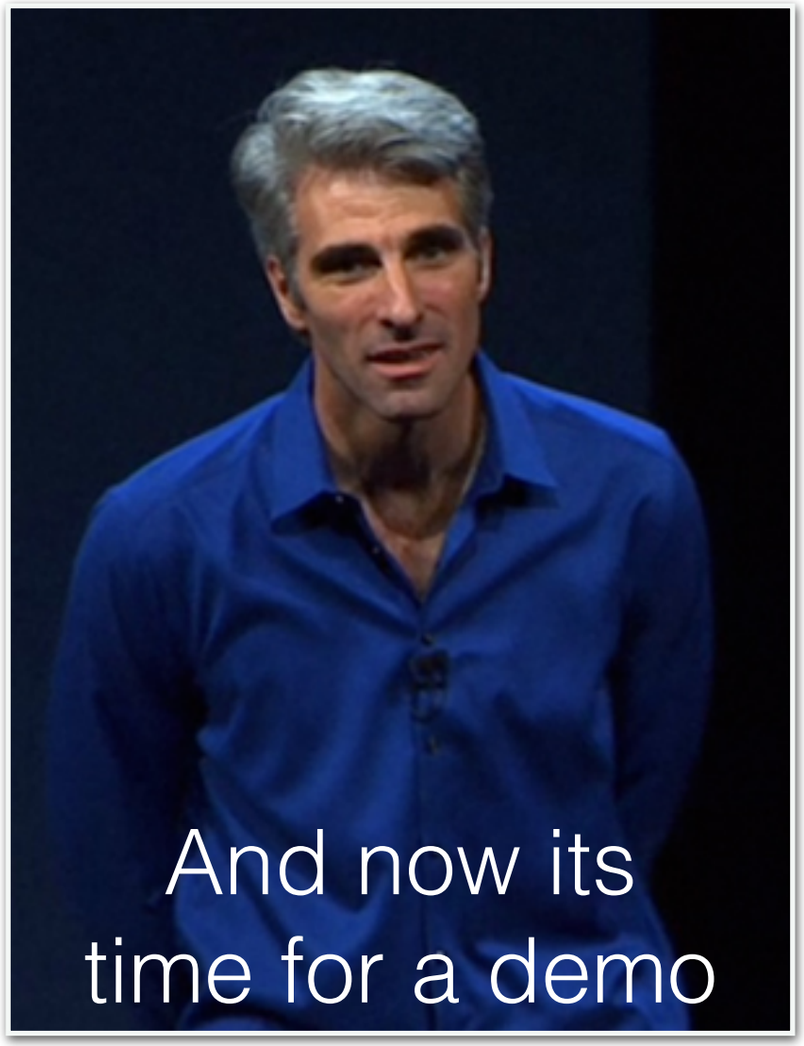
array

array iteration

protected class variable access

adding to our project

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



for next time…

- Swift
- Mobile HCI

MVC's

controller has direct connection to view class

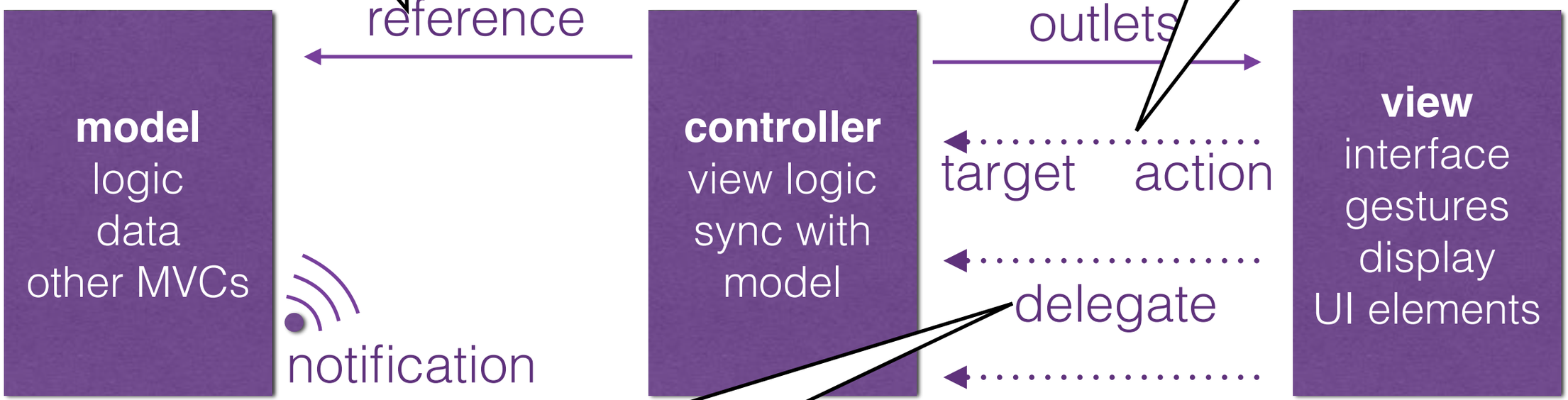
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

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;
```

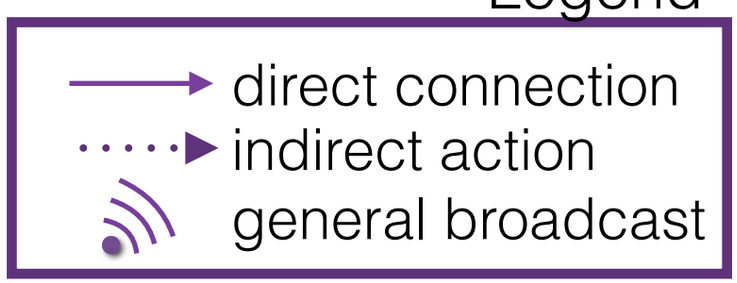



```
MainViewController ()<UITextFieldDelegate>
#pragma  mark - UITextfield Delegate
- (BOOL)textFieldShouldReturn:(UITextField *)textField { …  }
```

controller implements method for view class

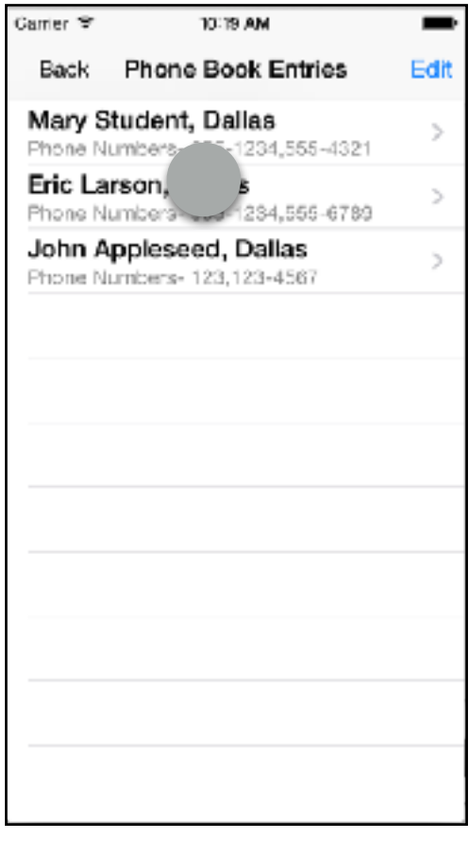
```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

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`

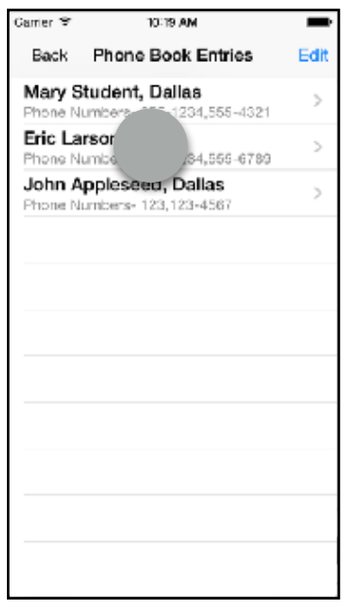
`viewDidAppear`

`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

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…

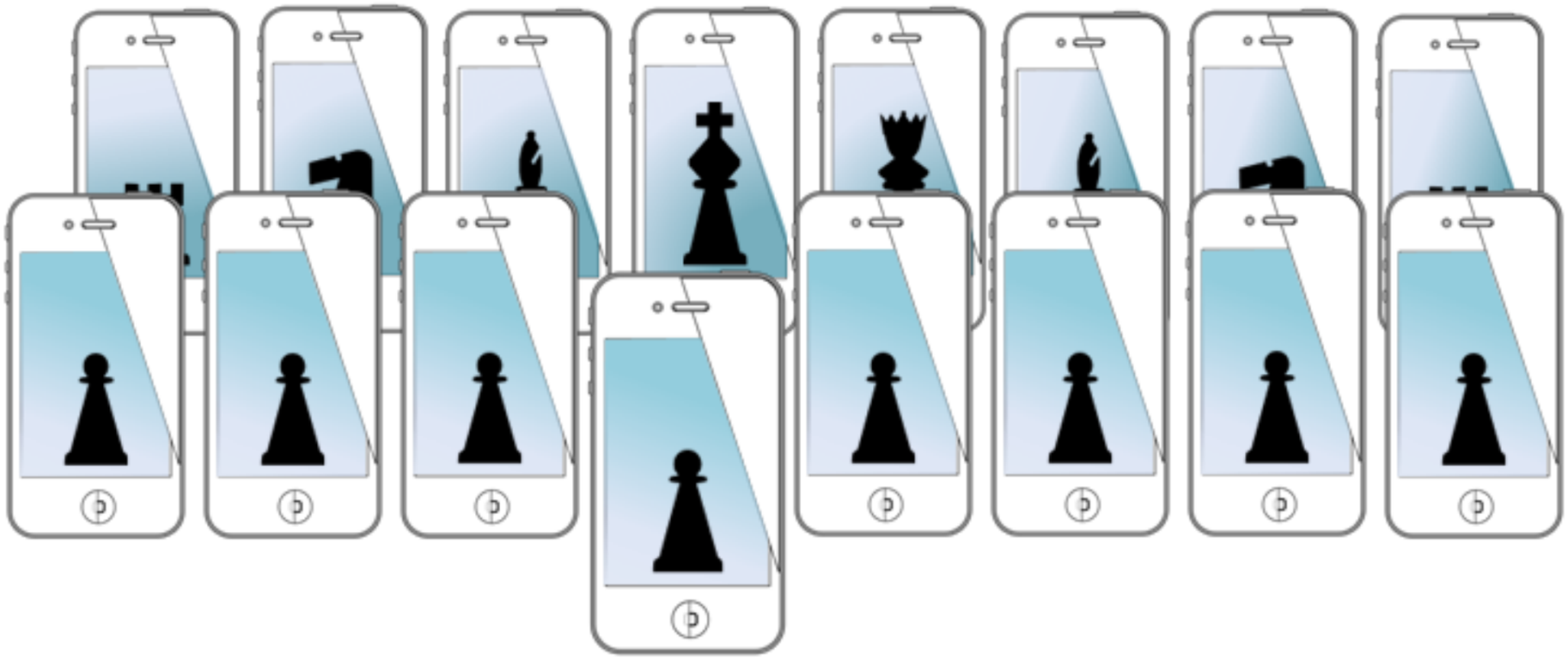
Need more help on MVC's ? Check out Ray Wenderlich:

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