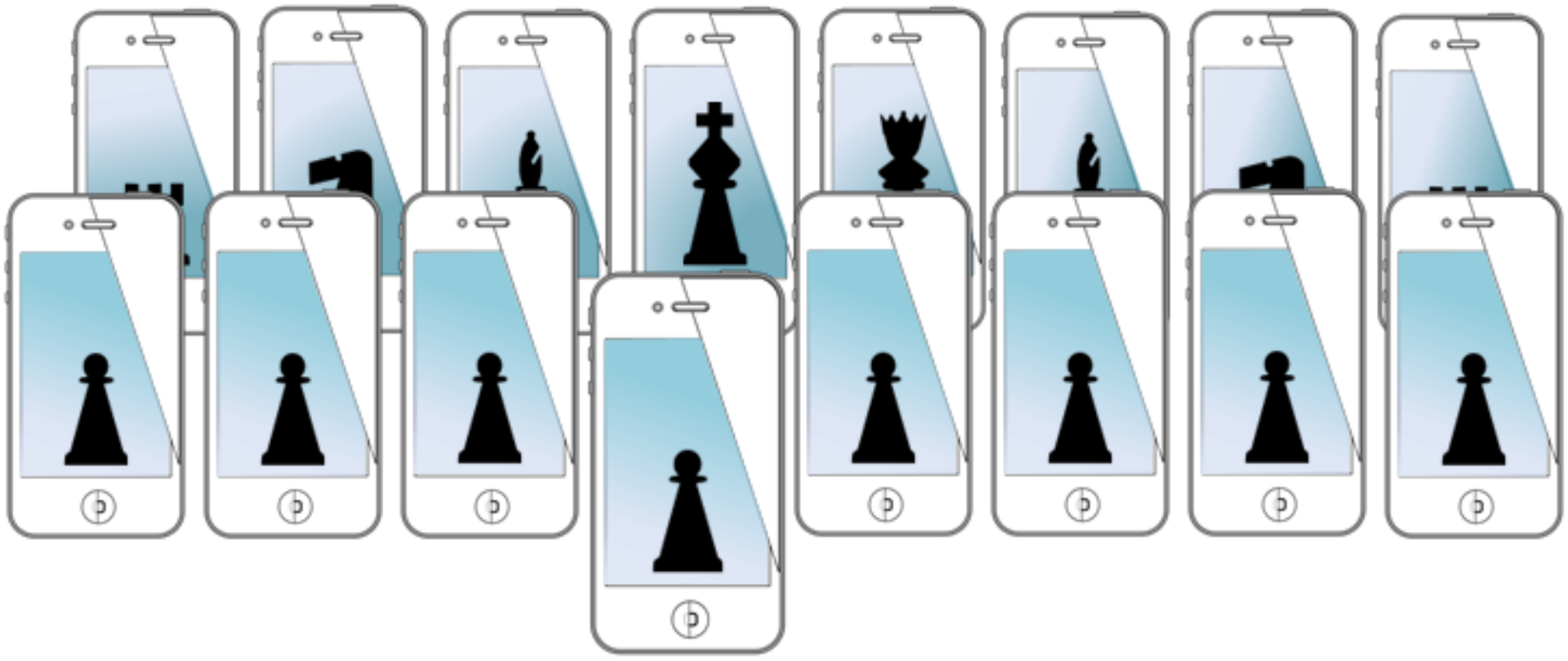
MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing & Learning

UI elements and swift

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

course logistics

- reminder: university developer program!
- **after today, I am out of mac-mini's!!!**
- next Time: flipped assignment, in person or via zoom
  - use headphones if zooming in class!!
- a1 due at the **end of next week**
  - make a video of the app and submit it (YouTube, dropbox, direct upload to canvas, etc.)
  - use quicktime for video (if you don't know what to use)

agenda

- blocks and concurrency
- target action behavior
  - and constraints
- text fields
- gesture recognizers
- timers / segmented control
- **remainder of time**: demo!

blocks and closures

- not callback functions (but similar)
  - created at runtime
  - once created, can be called multiple times
  - can access data from scope when defined
    - syntax is different in swift and objective-c (also slightly different behavior)
- not exactly a lambda (*but similar*)
  - but it acts like an object that can be passed as an argument or created on the fly
- swift uses closures, objective-c uses blocks

block/closure syntax

most common usage is as input into a function

```
^(Parameters) {
    // code
}
```

enumerate with block

```
// here the block is created on the fly for the enumeration
[myArray enumerateObjectsUsingBlock:^(NSNumber *obj, NSUInteger idx, BOOL *stop) {
    // print the value of the NSNumber in a variety of ways
    NSLog(@"Float Value = %.2f, Int Value = %d",[obj floatValue],[obj integerValue]);
}];
```

swift syntax

```
myArray.enumerateObjects({obj, idx, ptr in
    print("\(obj) is at index \(idx)")
})
```

```
{ ( parameters ) -> return type in
    statements
}
```

some semantics

- variables from same scope where block is defined are **read only**

```
NSNumber * valForBlock = @5.0;
```

- Unless you use keyword:

```
__block NSNumber * valForBlock = @5.0;
```

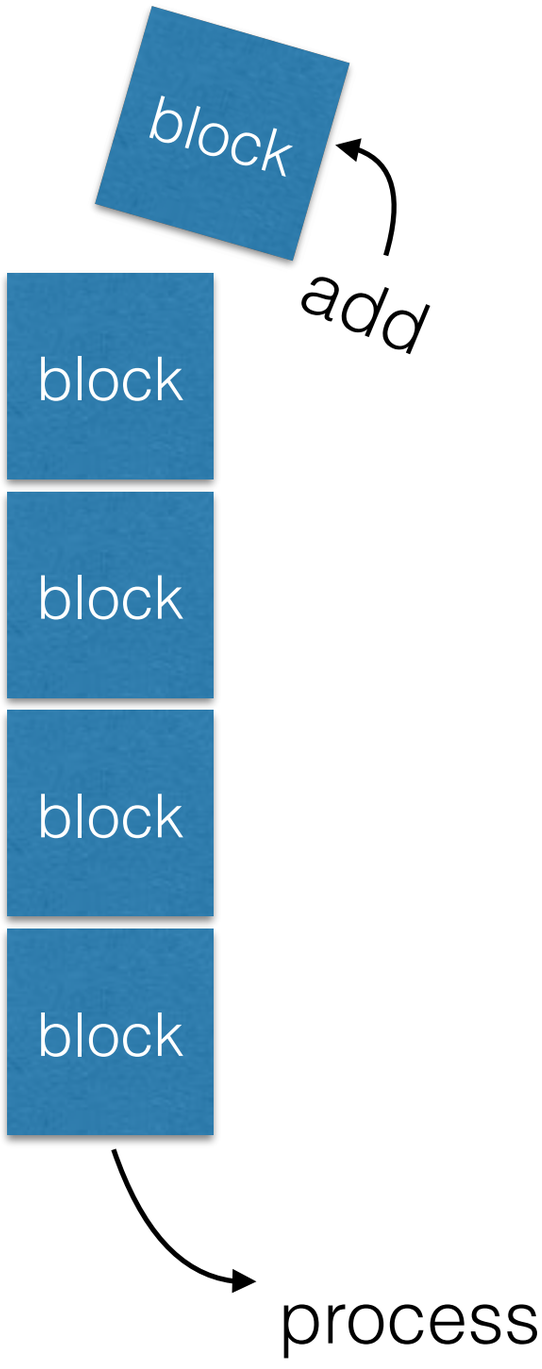
- classes hold a **strong** pointer to blocks they use
- blocks hold a **strong** pointer to __block variables
  - so using “self” would create a retain cycle

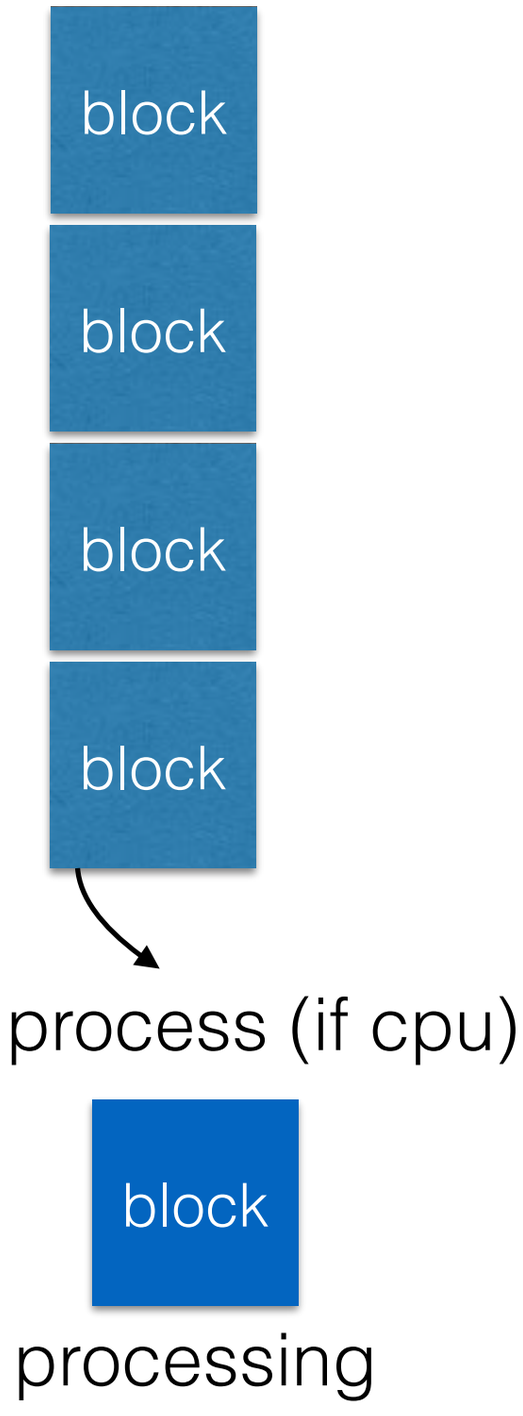
```
self.value = (some function in block)

__block ViewController * __weak  weakSelf = self;

weakSelf.value = (some function in block)
```

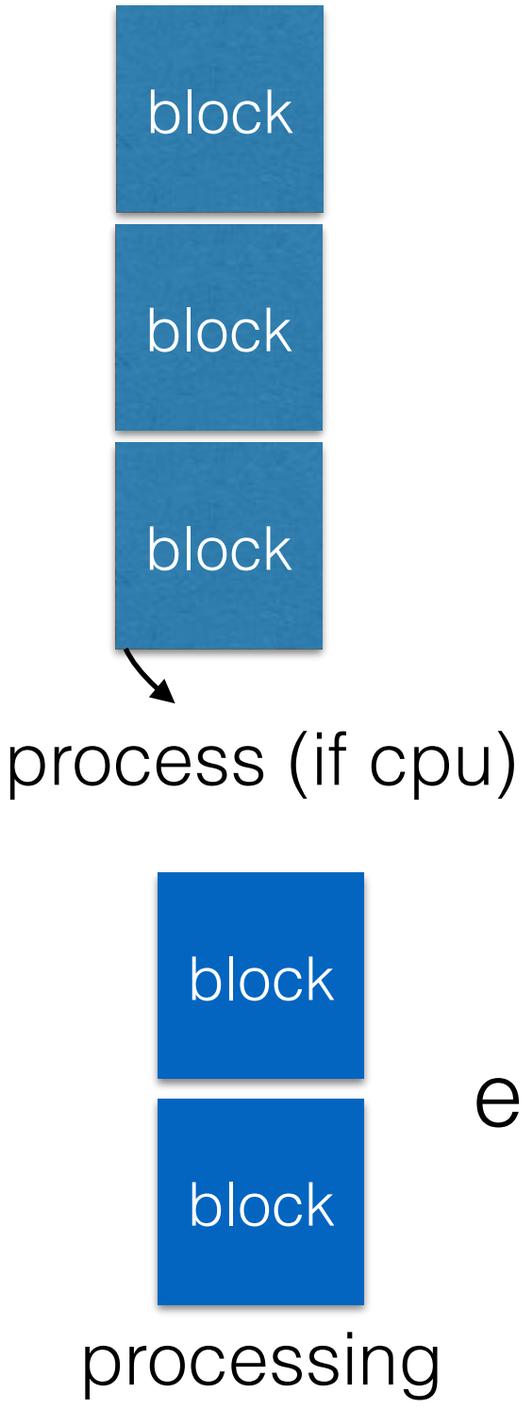
concurrency in iOS

- grand central dispatch (GCD) handles all operations
  - GCD looks at “queues” of **blocks** that need to be run
  - GCD and the Xcode compiler work deep inside the OS, actually in the kernel — they are optimized
  - for a **serial queue** each block is run sequentially
  - for **concurrent queues** the first block is dequeued
    - if CPU is available, then the next block is also dequeued, but could finish any time
- the **main queue handles all UI operations** (and no other queue should generate UI changes!!)
  - so, **no updating of** the views, labels, buttons, (image views*) **except from the main queue**

concurrent queues



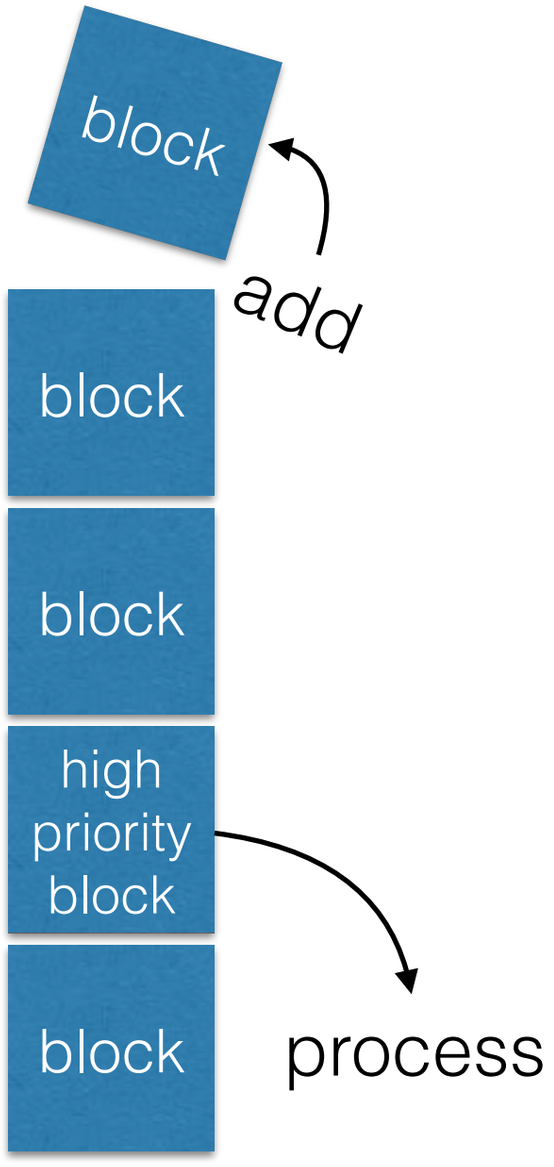


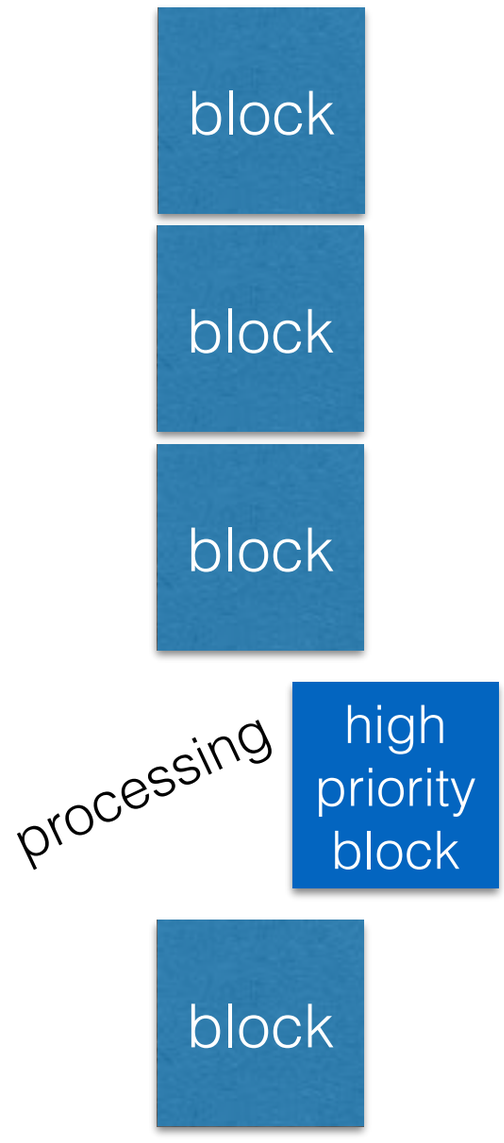


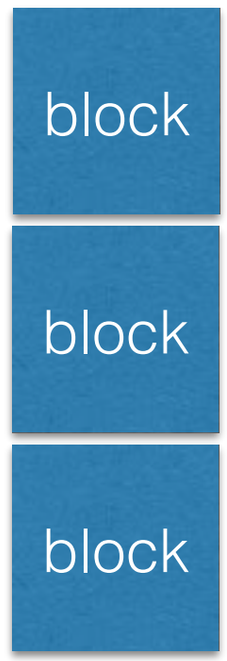
threads?

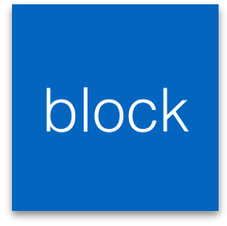
either could finish

the main queue









processing

one thread
UI is in sync

queue syntax

create new queue

define block

update UI, another block

```
NSOperationQueue *newQueue = [[NSOperationQueue alloc] init];
newQueue.name = @"ObjCQueue";
[newQueue addOperationWithBlock:^{
    // your code to execute
    for(int i=0;i<3;i++)
        NSLog(@"I am being executed from a dispatched queue, fro   bjective-c");

    // now I need to set something in the UI, but I am n   in the main thread!
    // call from main thread
    dispatch_async(dispatch_get_main_queue(), ^{
        self.label.text = [NSString stringWithFormat:@"Finished running %d times, Safe",3];
    });

}];
```

```
var queue:DispatchQueue = DispatchQueue(label: "myQueue")
queue.async {
    //code to execture in block
    for _ in 0..<3{
        print(" I am being executed from a default queue")
    }
    // now we go to the main queue
    DispatchQueue.main.async {
        print("Running from main queue!")
    }
}
```

same functionality,
update UI, another block

queue syntax

- using global queues

access a global queue

```
// An example of using already available queues from GCD
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // your code to execute
    for(int i=0;i<3;i++)
        NSLog(@"I am being executed from a global concurrent queue");

    // now I need to set something in the UI, but it must be set in the main thread!

    // call from main thread
    dispatch_async(dispatch_get_main_queue(), ^{
        self.label.text = @"Finished running from GCD global";
    });
```

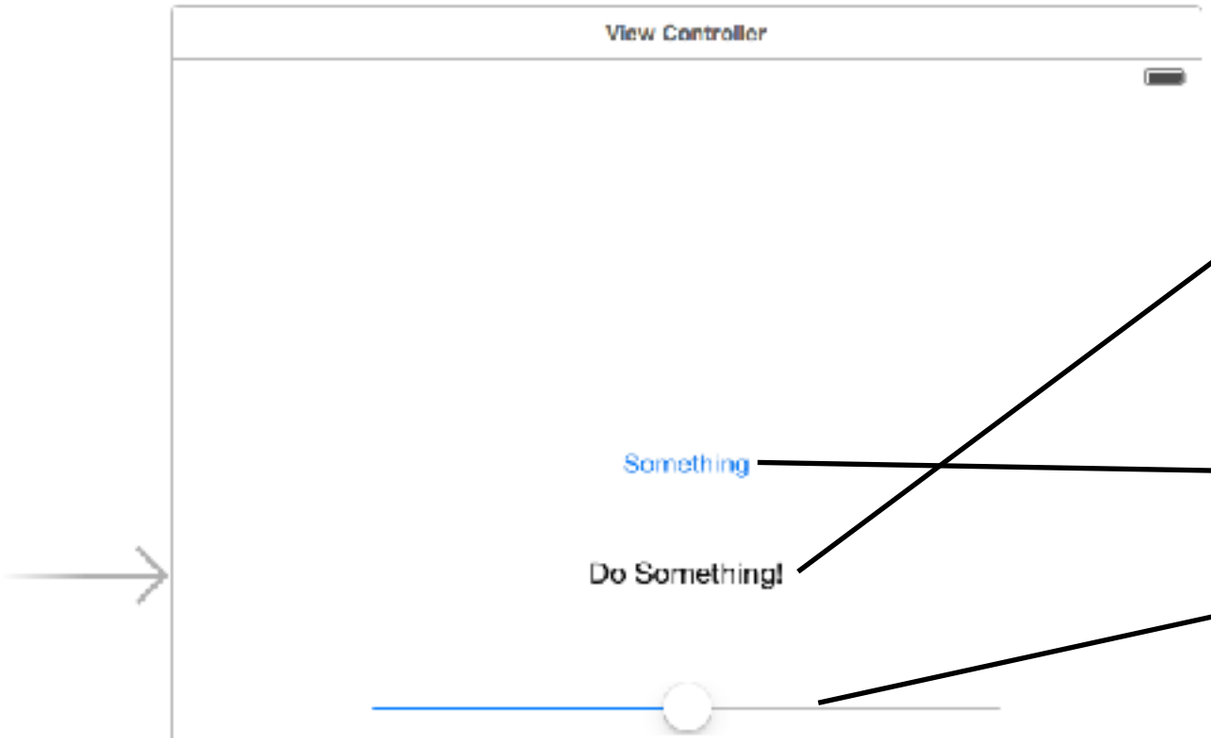
not on main queue!!

main queue!

```
DISPATCH_QUEUE_PRIORITY_LOW
DISPATCH_QUEUE_PRIORITY_DEFAULT
DISPATCH_QUEUE_PRIORITY_HIGH
DISPATCH_QUEUE_PRIORITY_BACKGROUND
```

target and action

- UI elements communicate back to their controllers with **actions**, UI elements are called from the **Main Queue**



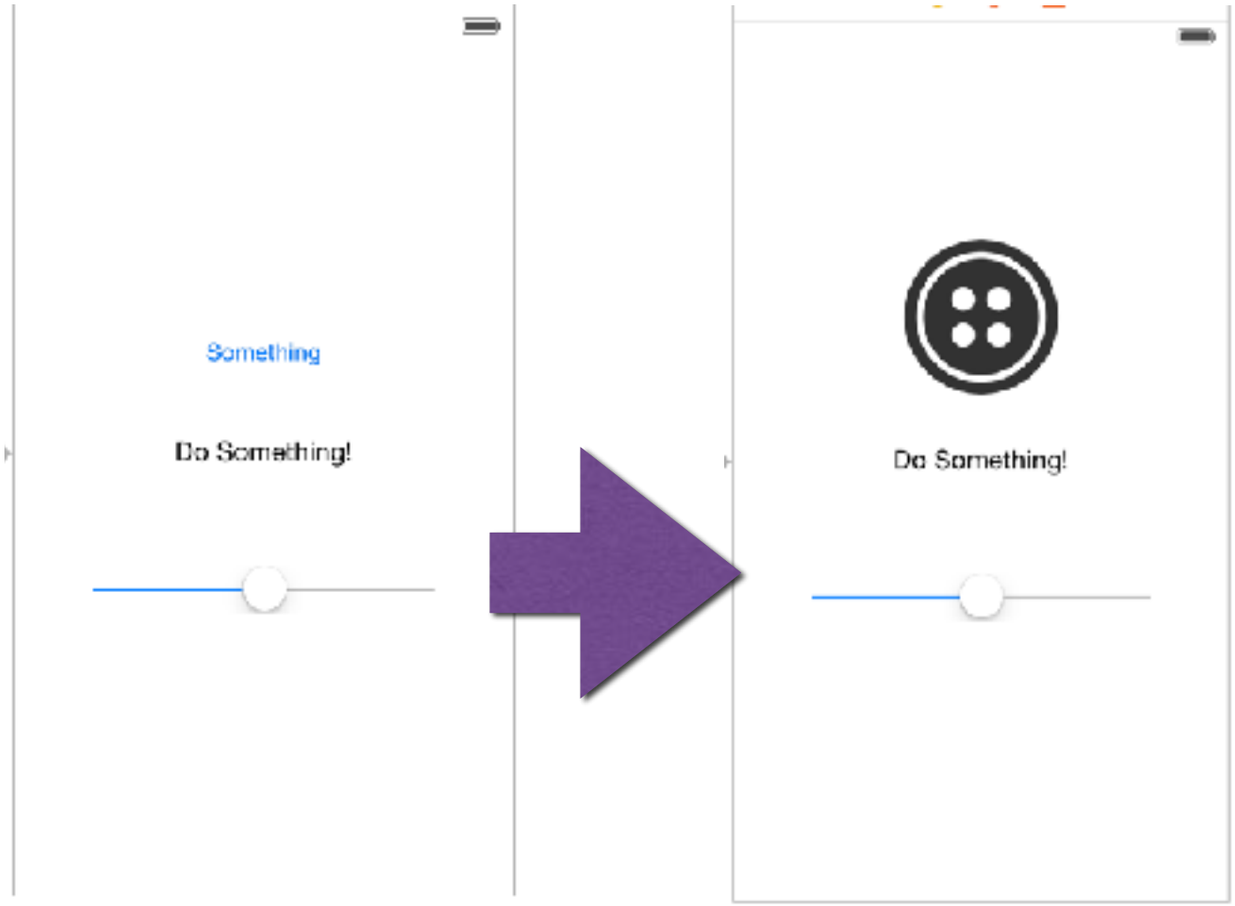
storyboard classes: the views

class: the controller

```
//
#import "ViewController.h"

@interface ViewController ()
@property (weak, nonatomic) IBOutlet UILabel *
    somethingLabel;

@end

@implementation ViewController
- (IBAction)buttonPressed:(UIButton *)sender {
    self.somethingLabel.text = @"Thanks!";
}
- (IBAction)sliderChanged:(UISlider *)sender {
    self.somethingLabel.text = [NSString
        stringWithFormat:@"Value of slider is %.
        2f",sender.value];
}

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the
        view, typically from a nib.
}
```

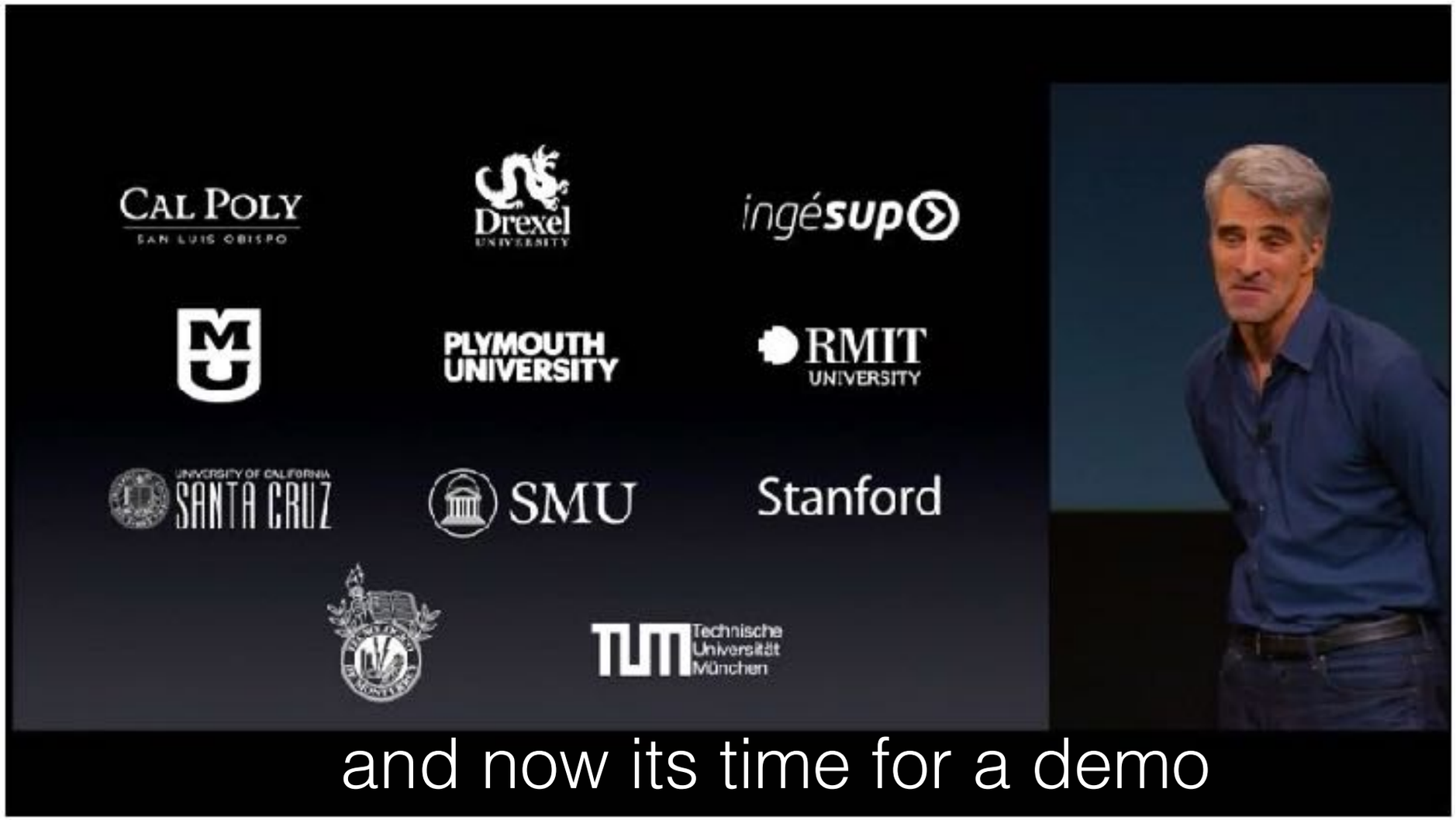
bring your buttons to life

- in many settings you are **given criteria** from a graphic designer
  - but right now, **you** are the graphic designer
- use **images** for more **descriptive** buttons and labels
- **good tip**: make them the right size from the start!

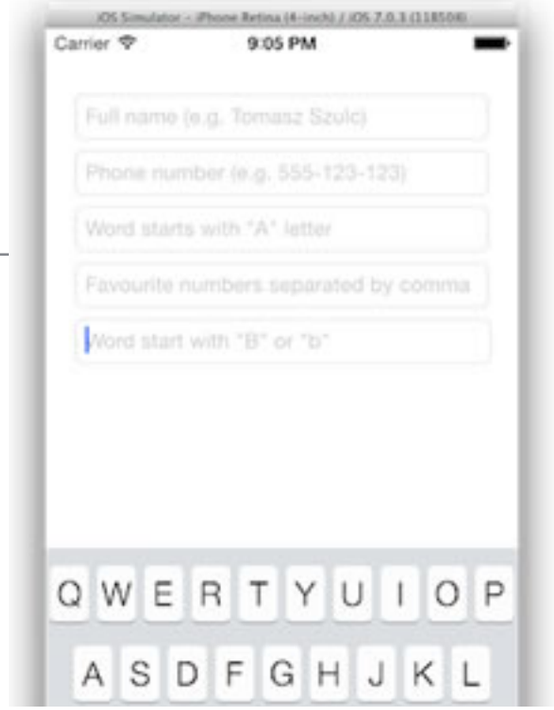


UI basics demo

Guess the Number…



text fields



- text fields are common
- but they require the use of the keyboard!
- so you need **delegate** when events happen
  - say when to dismiss the keyboard
  - define what happens to text that the user entered

outlet, setup from storyboard

```
@interface ViewController () <UITextFieldDelegate>
@property (weak, nonatomic) IBOutlet UITextField *nameTextField;
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    self.nameTextField.delegate = self;
}

-(BOOL)textFieldShouldReturn:(UITextField *)textField{
    [textField resignFirstResponder];
    return YES;
}
```

tell compiler we are delegate

return button pressed

make VC delegate

give up keyboard control

UI text field demo

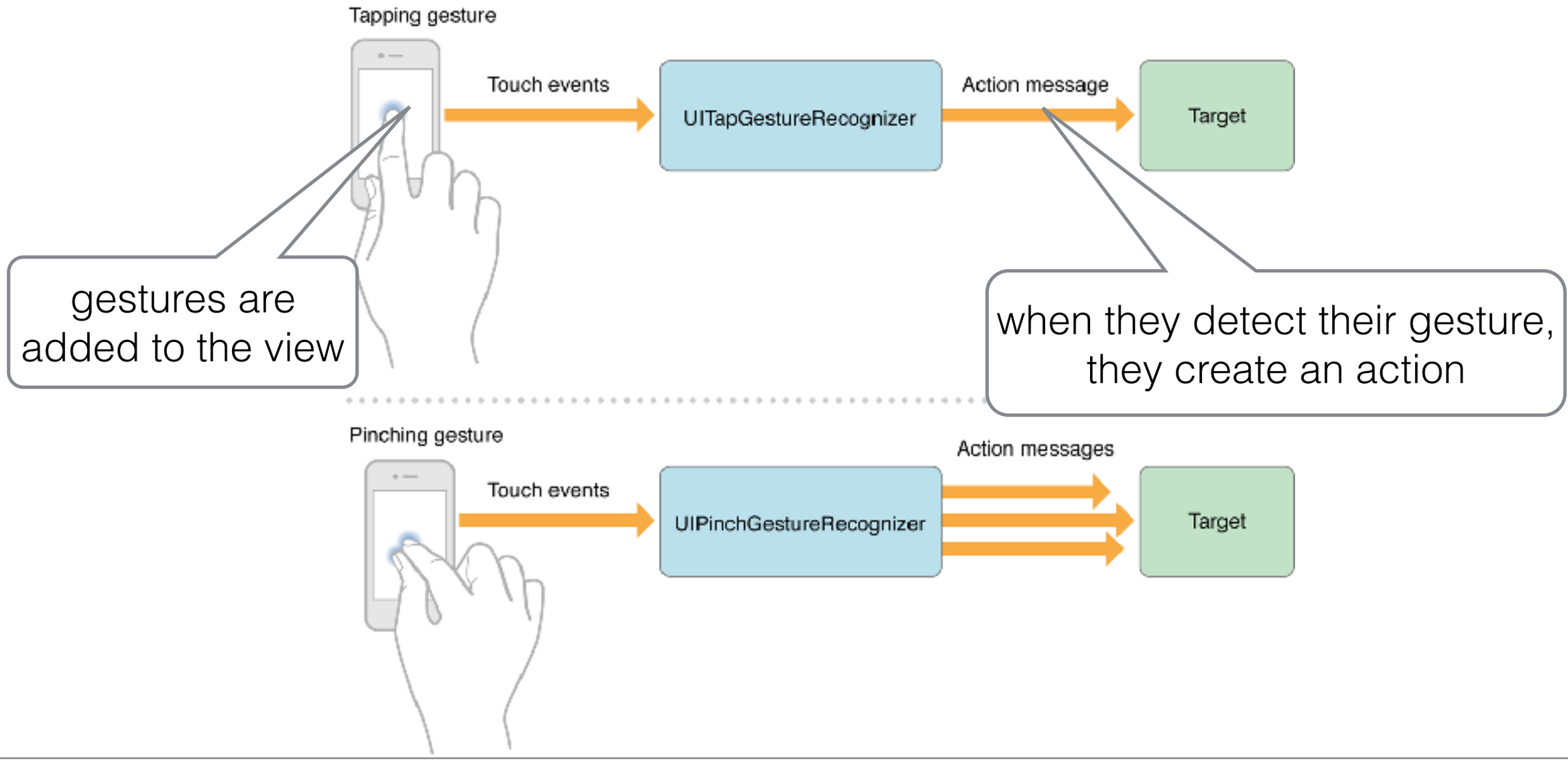
gesture recognition

- the fun part about doing things on the iPhone!
- **the point**: recognize different gestures and then make something happen
- lots of ways to do this
  - **programmatically**: quick and versatile
  - **target-action**: easy
  - **delegation**: more feature rich
- here is the complete documentation:

https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GestureRecognizer_basics/GestureRecognizer_basics.html

gesture recognition

- need a UIGestureRecognizer
  - UITapGestureRecognizer, UIPinchGestureRecognizer, …



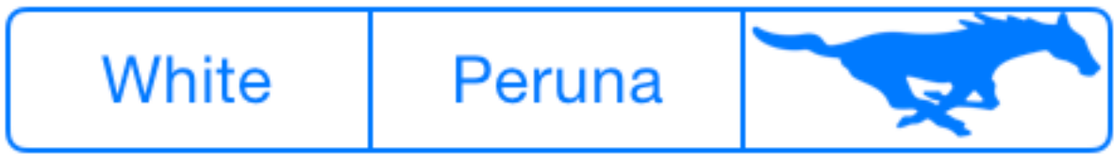
UI gesture demo

timers, segmented control

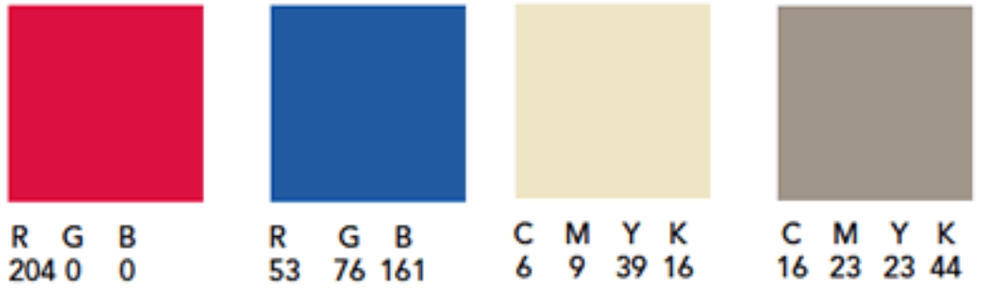
```
- (IBAction)updateFromSegmentedControl:(UISegmentedControl *)sender {

    NSString *selectedText = [sender titleForSegmentAtIndex: [sender selectedSegmentIndex]];

    YOUR_CODE
}
```

get title from control

get value of control



standard SMU colors

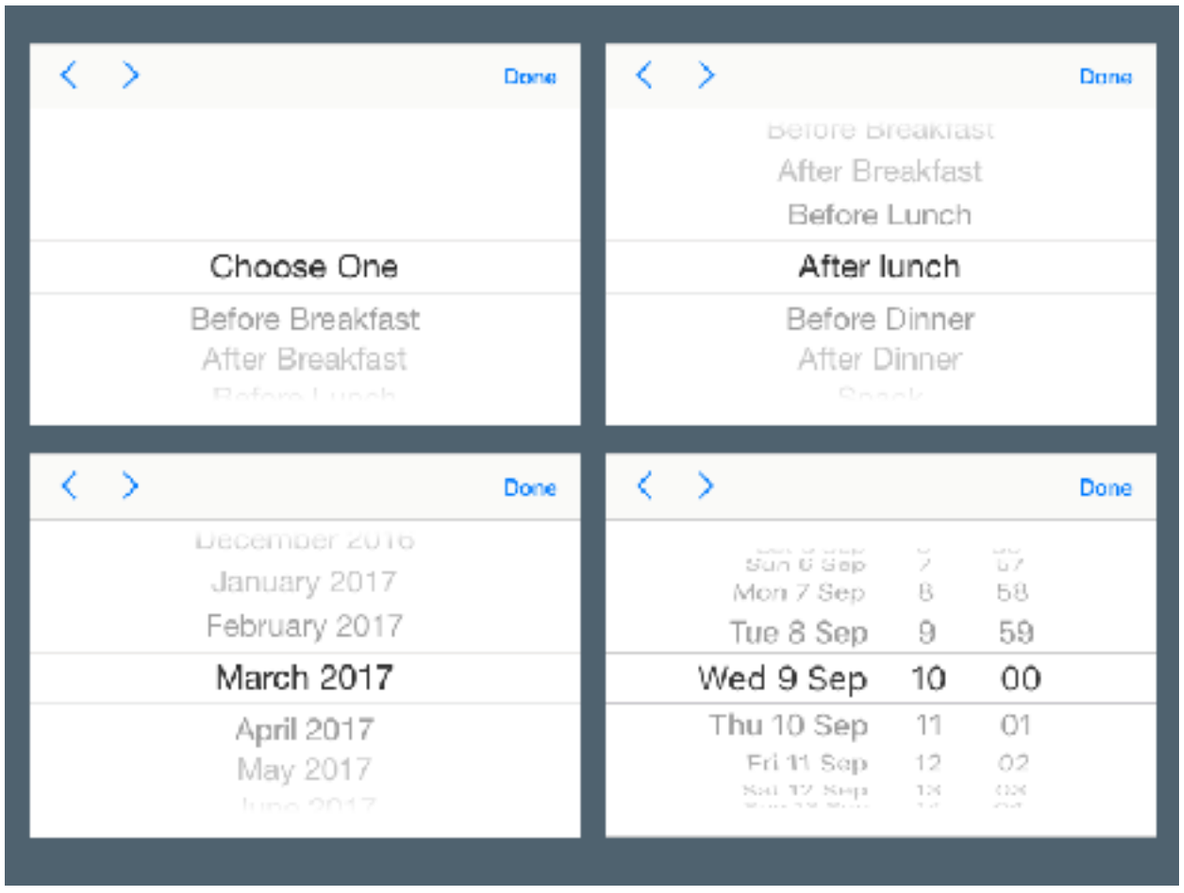



```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:someIntervalInSeconds
                                                  target:self
                                                selector:@selector(someFunction:)
                                                userInfo:nil
                                                 repeats:YES];

// don't get blocked by the main thread
[[NSRunLoop mainRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
```

when should the timer be running? what modes?

pickers

- **look at documentation**: find out how to use a picker view
- you will have all the tools to do it from working with collections and the table view controllers in flipped lecture video!
- you are the data source!



assignment one

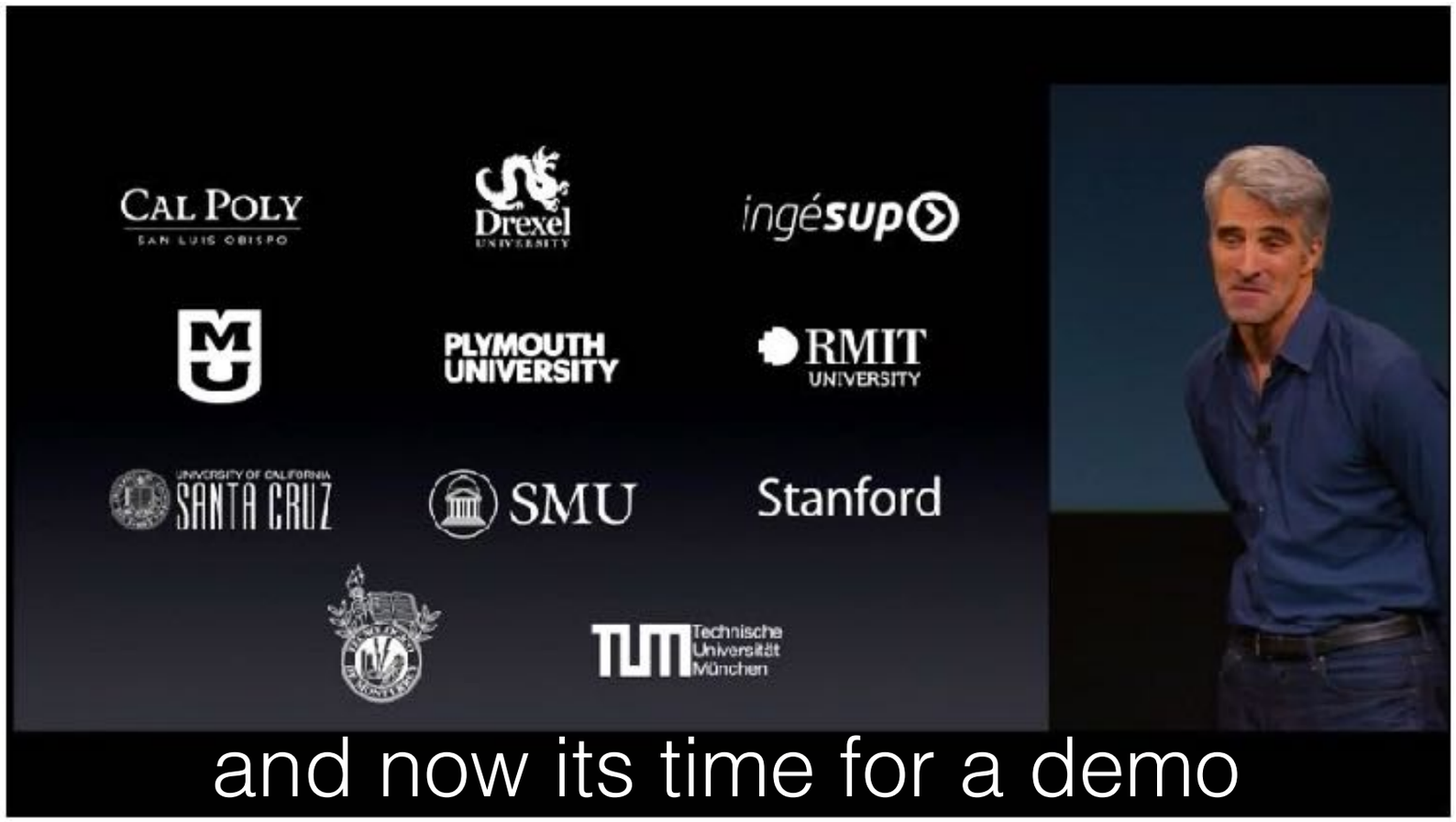
- Posted on Canvas!

before demo… don't forget

- View Controllers in iOS via flipped assignment
  - Watch videos **before class**
  - download GitHub project and having running **before class**
  - come ready to work in teams on an in-class assignment
    - distance students: turn in within a few days of the assignment
- next lectures:
  - mobile HCI
  - audio access

adding functionality

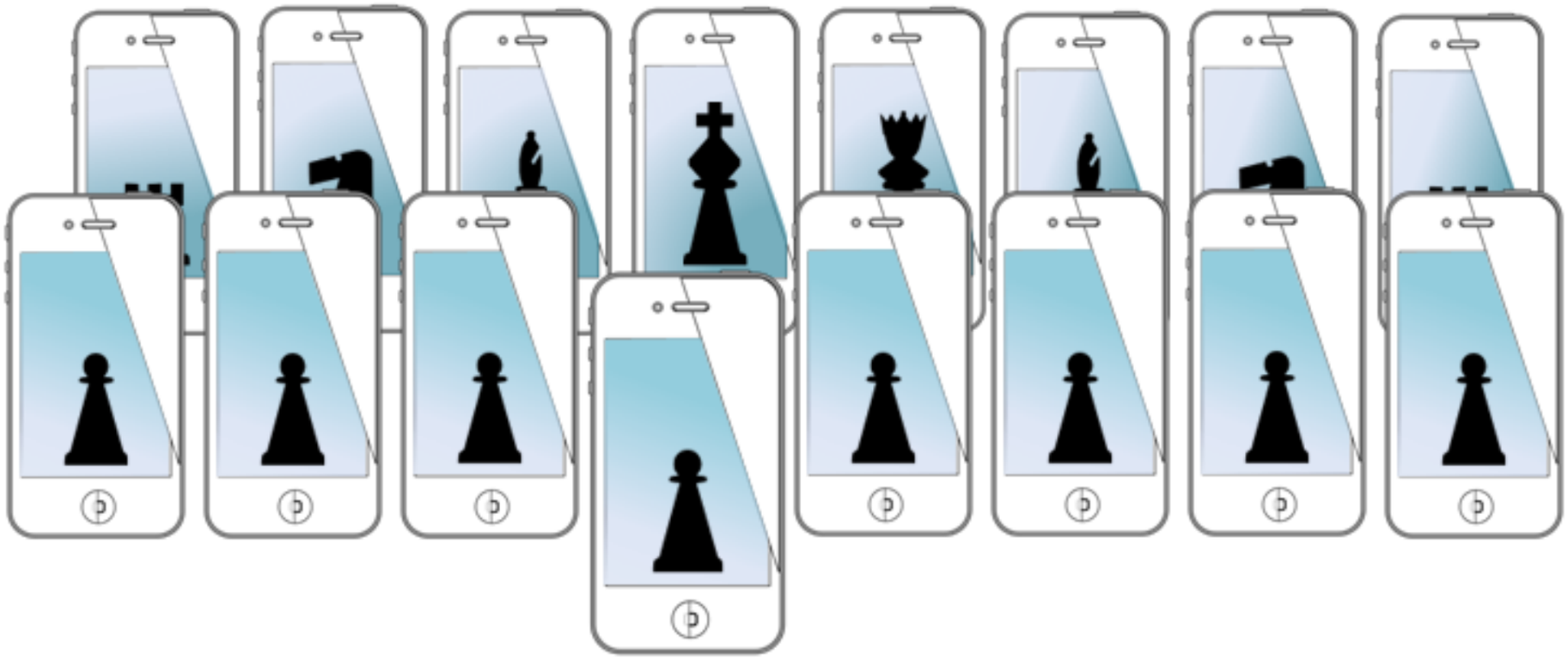
- support for landscape and portrait modes with auto layout
- using the `switch` statement in swift (very powerful)
- `if let`
- mixing objective-c
- and more!



https://developer.apple.com/swift/

https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html

MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing & Learning

UI elements

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