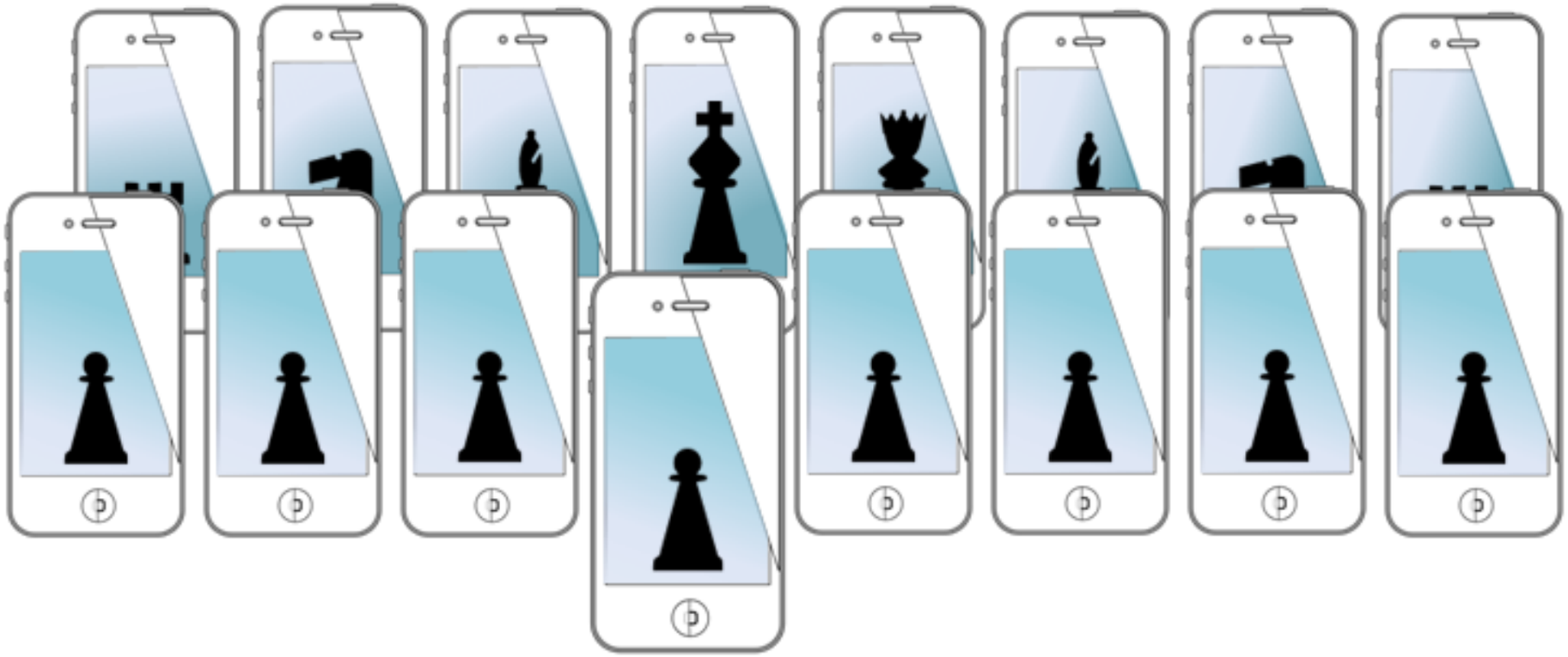
MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing & Learning

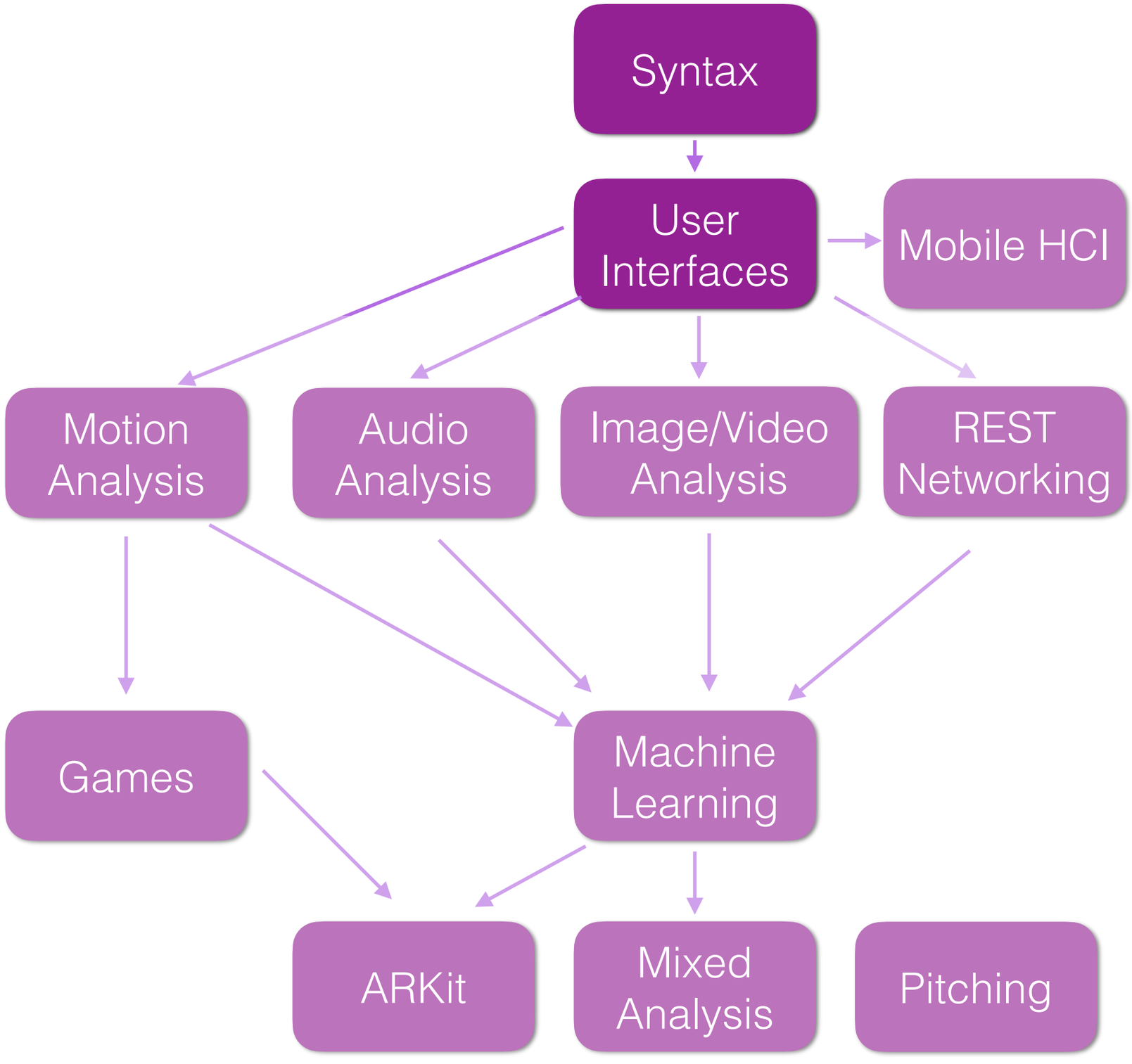
UI elements

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

course logistics

- reminder: university developer program!
- next time: **flipped assignment**, in person/distance
- 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)

class progression



agenda

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

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

```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

blocks and closures

- a block of code that you want to run at another time and perhaps pass to other classes to run
  - created at runtime
  - acts like an object that can be passed as an argument or created on the fly
  - once created, can be called repeatedly
  - can access variables from scope where defined
  - syntax is slightly different in swift and objective-c
  - common to define when calling a method that uses block
- swift calls these **closures**, objective-c says **blocks**

block/closure syntax

most common usage is as input into a function

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

this variable is in scope of block!

```
NSNumber *objInScope = @(32)
// 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]);
    NSLog(@“Scope Variable = %.2f",[objInScope floatValue]);
}];
```

swift syntax

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

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

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

Also valid if closure is last input

some semantics

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

```
NSNumber * objInScope = @5.0;
```

- Unless you use keyword (now mutable):

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

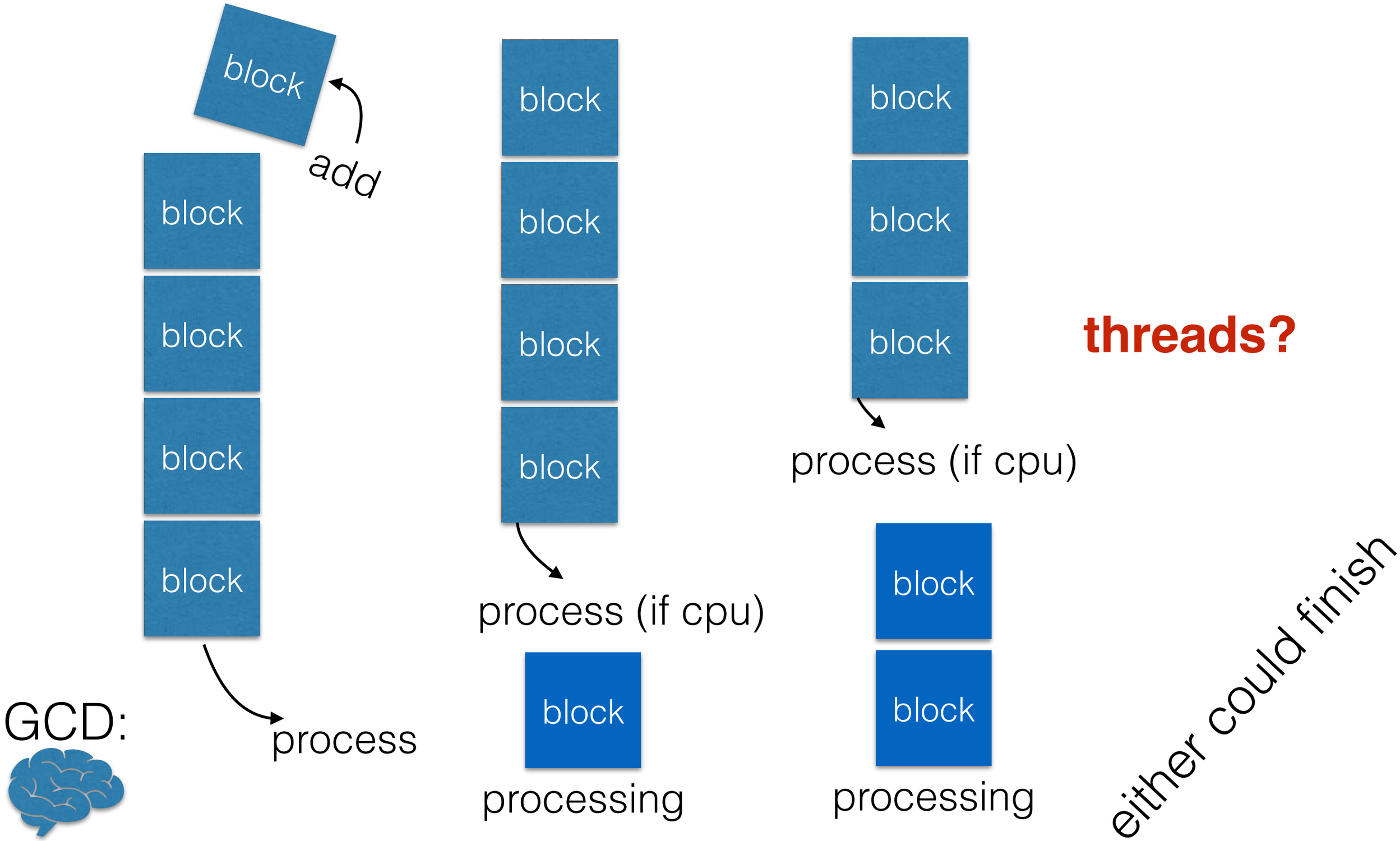
- classes hold a **strong** pointer to blocks they use
- blocks hold a **strong** pointer to `__block` variables
  - so using “self” would create a retain cycle

```
self.value = (some function in block)
__block ViewController * __weak  weakSelf = self;
weakSelf.value = (some function in block)
```

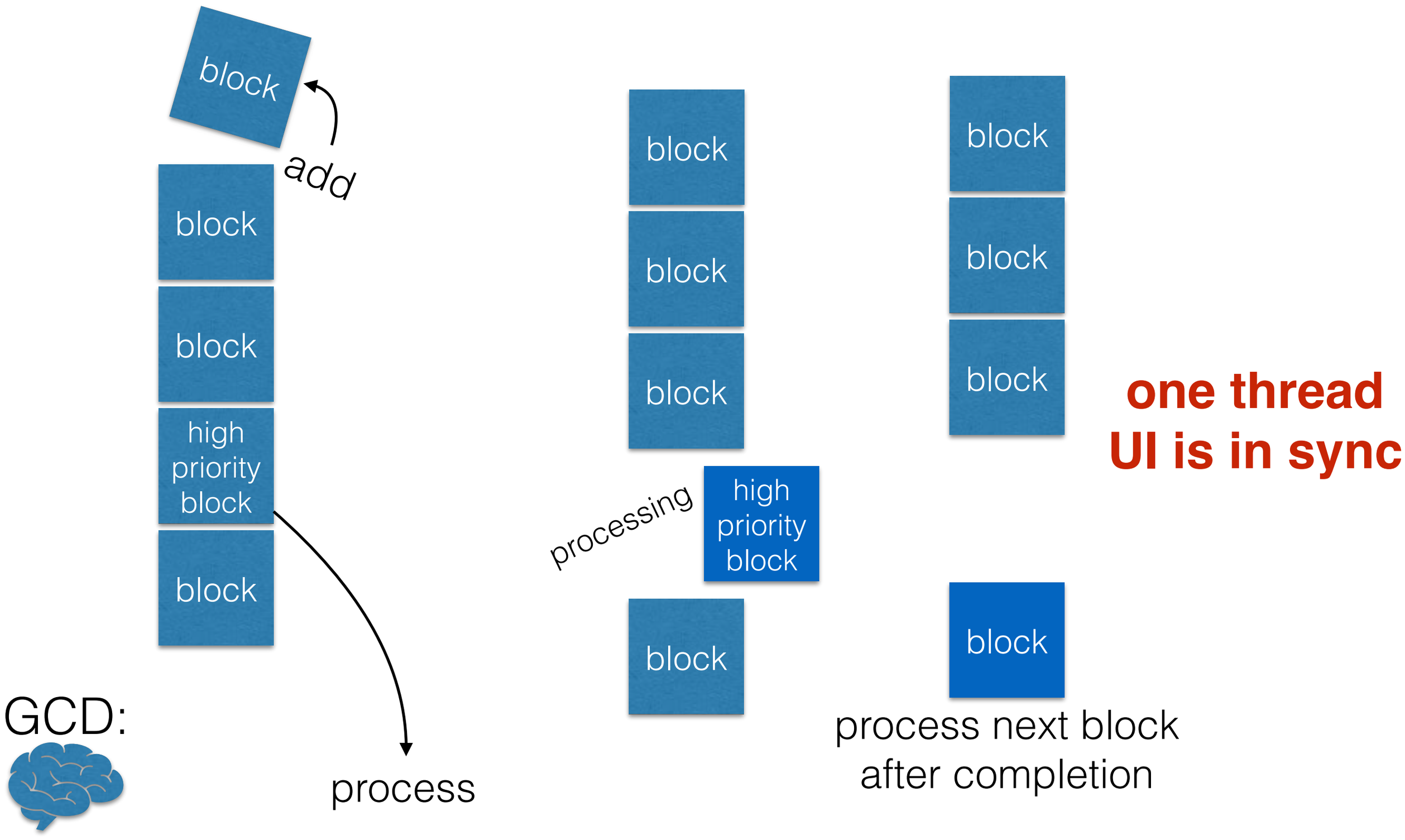
concurrency in iOS

- 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



the main queue



create your own queue!

```
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, from objectives, but
        imagine I am doing something time consuming, like loading something from the internet”);

    // now I need to set something in the UI, but I am not 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];
    });

}];
```

create new queue

define block

update UI, another block

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

same functionality,
update UI, another block

common queues

- using global queues

access a global queue

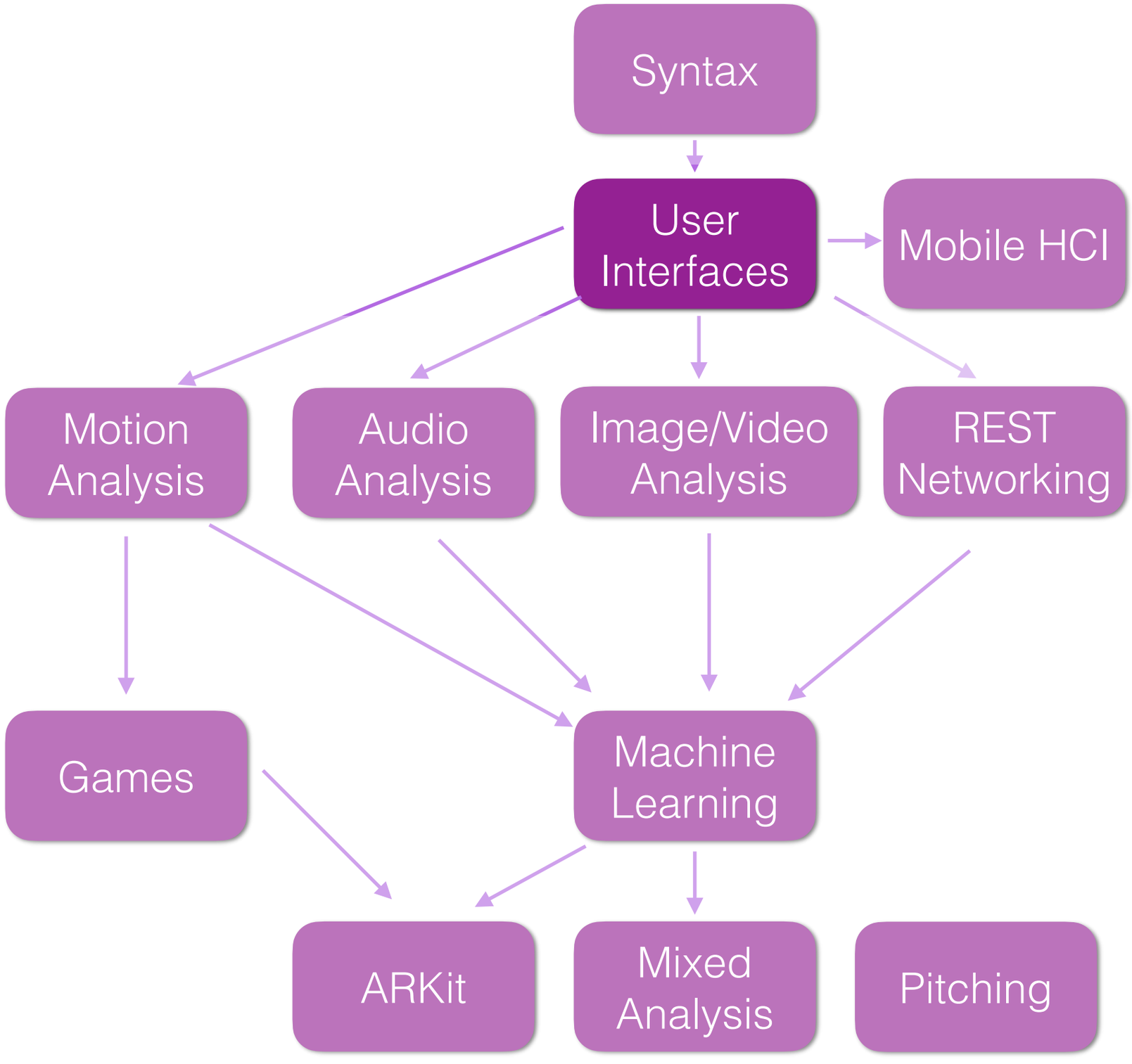
```
// An example of using already available queues from GCD
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // your code to execute
    for(int i=0;i<3;i++)
        NSLog(@"I am being executed from a global concurrent queue");

    // now I need to set something in the UI, but must get 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!

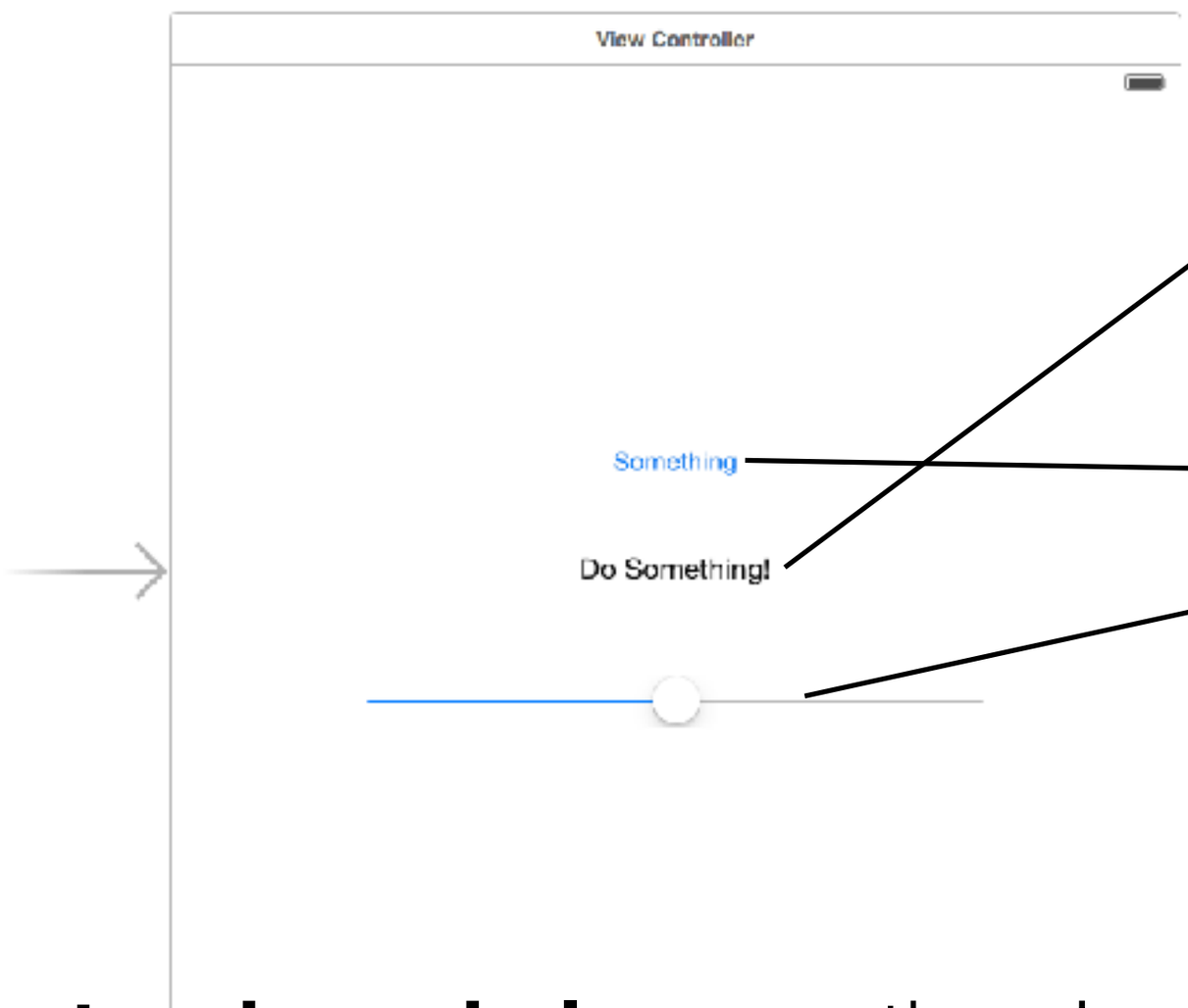
```
DISPATCH_QUEUE_PRIORITY_LOW
DISPATCH_QUEUE_PRIORITY_DEFAULT
DISPATCH_QUEUE_PRIORITY_HIGH
DISPATCH_QUEUE_PRIORITY_BACKGROUND
```

class progression



target and action review

- UI elements can have **outlets** and **actions**, UI actions are called from the **Main Queue**



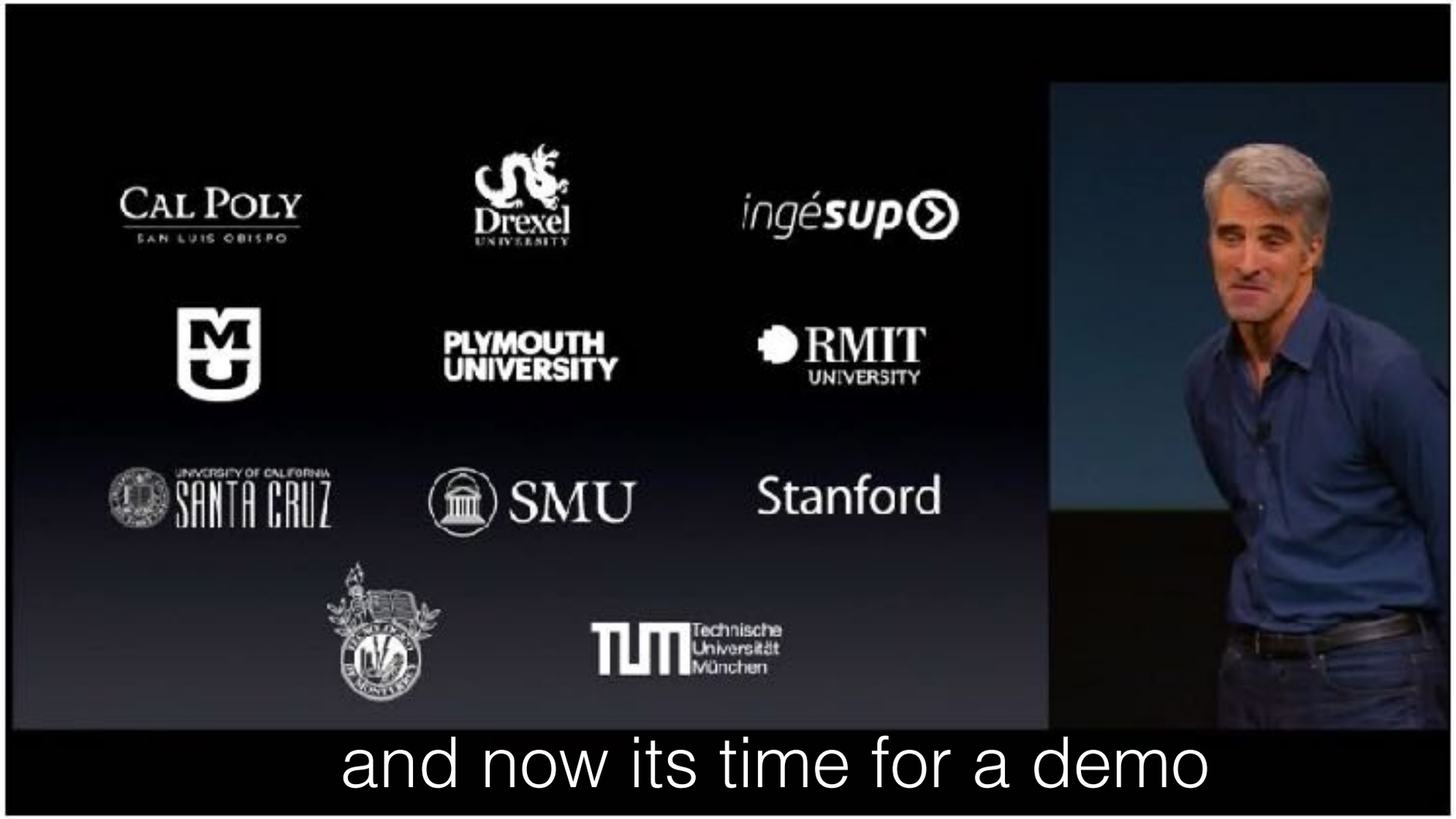
class: the controller

```
//
#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.
}
```

storyboard classes: the views

UI basics demo

Guess the Number…



delegation and protocols

- delegation is an alternative to virtual functions, avoids inheritance
- rather than inheriting functionality, I can declare that I have certain functionalities in my class
  - this allows other classes to call specific methods that I implement!
- if I declare myself as a delegate for a class, it means I agree to implement a protocol
  - a protocol is simply a list of functions that I implement
- delegation is user EVERYWHERE in iOS user interfaces

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

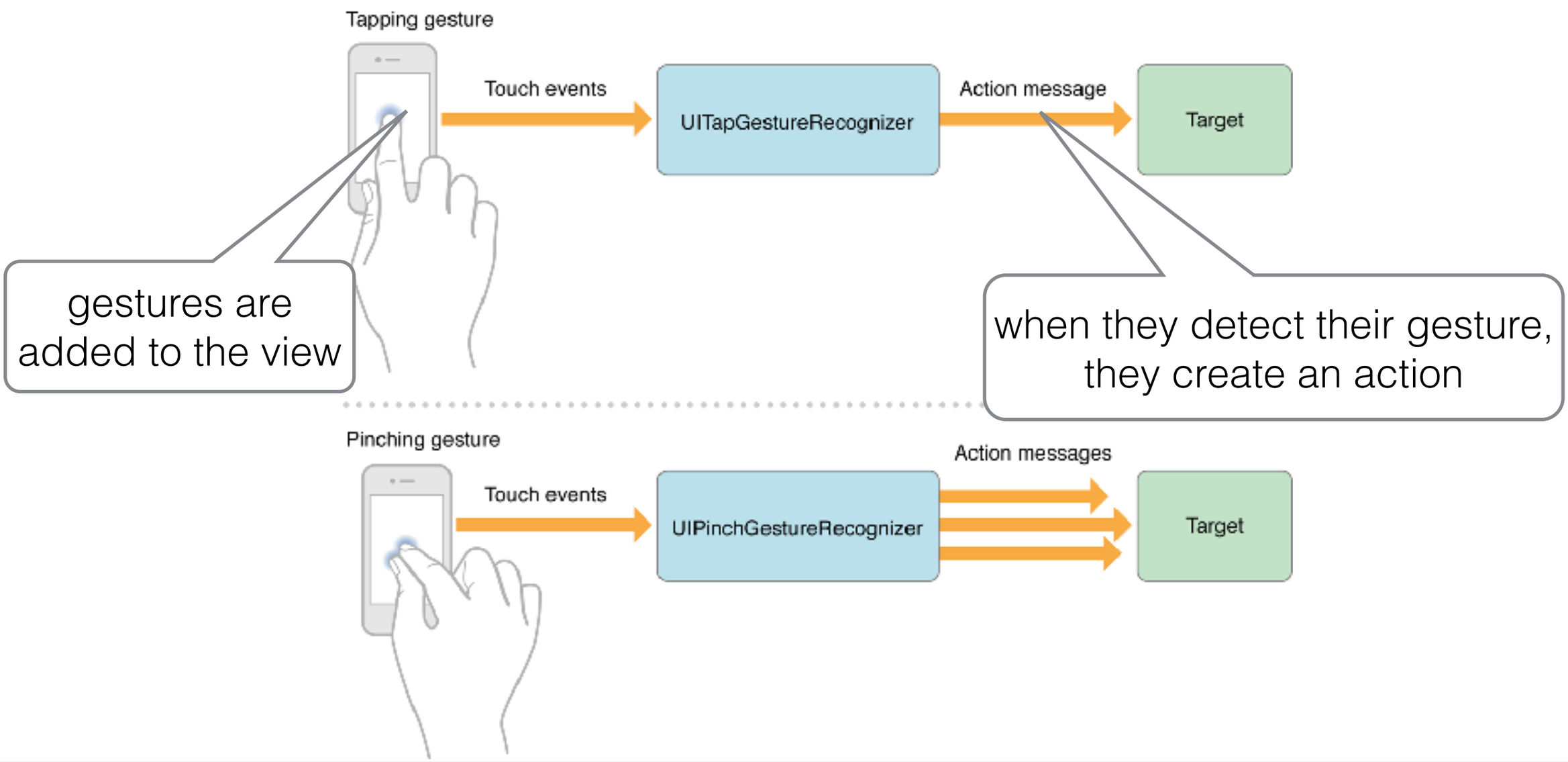
gesture recognition

- 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