iOS Seminar

Michael Saelee <lee@iit.edu>
Senior Lecturer, Department of Computer Science

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Questions for you

- What is your role in your organization?
  - Developer? Manager? HR?
- What are you looking to get out of this?
- Are you looking to build apps yourself?

Agenda

- Developer tools
- GUI-building
- Objective-C
- iOS SDK overview
- UIKit + design patterns

I'll try to answer…

- What is developing for iOS like?
- What are the features of the toolchain?
- What does "iOS code" look like?
- What are the building blocks of an app?
  - How do they fit together?

I won't…

- Try to sell you on the platform
- Cover many small or esoteric details
  - But may provide references if relevant
- Consider testing, backwards compatibility, or third-party solutions

Approach

- Alternate slides with hands-on sessions
- Ideally, 50% of time using iOS dev tools
  - But, impossible to learn language & libraries and be productive so rapidly
    - More *exploration* than live coding

Code repository

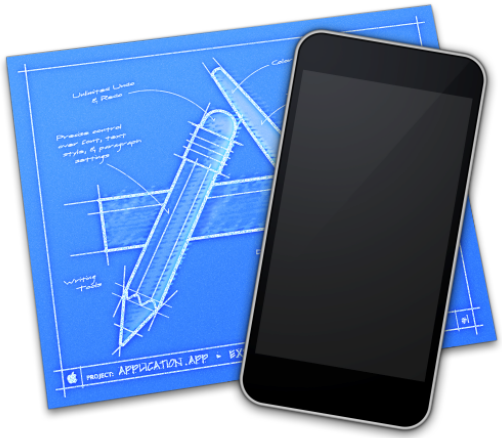
- All sample code at
  http://j.mp/ios1871

Hands-on

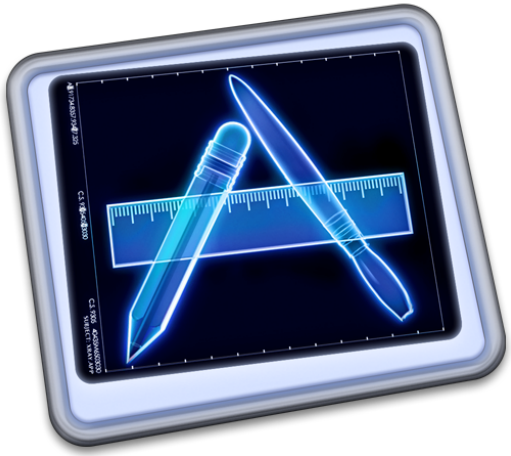
- Downloading the sample code

§Developer Tools

Xcode



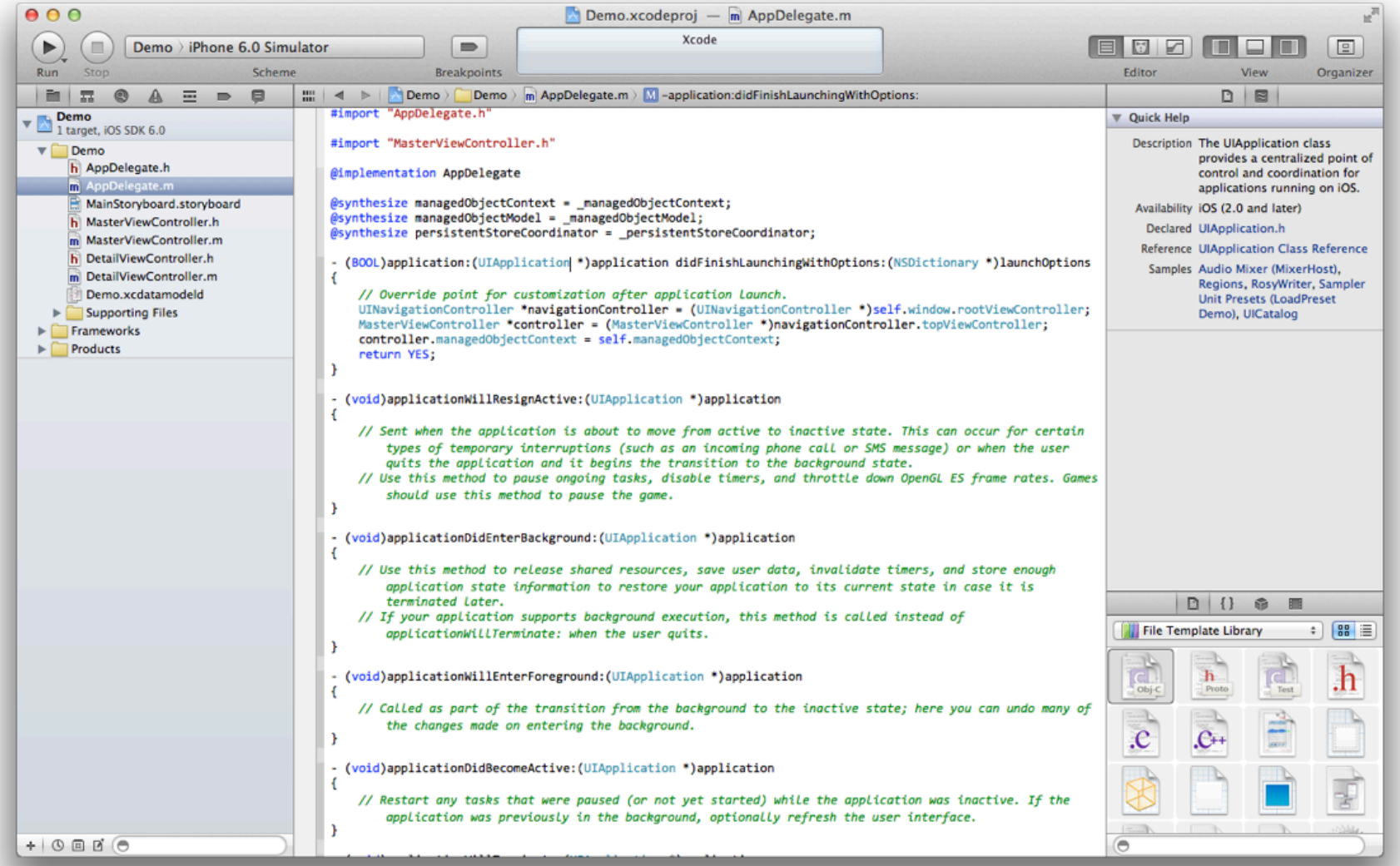
iOS Simulator



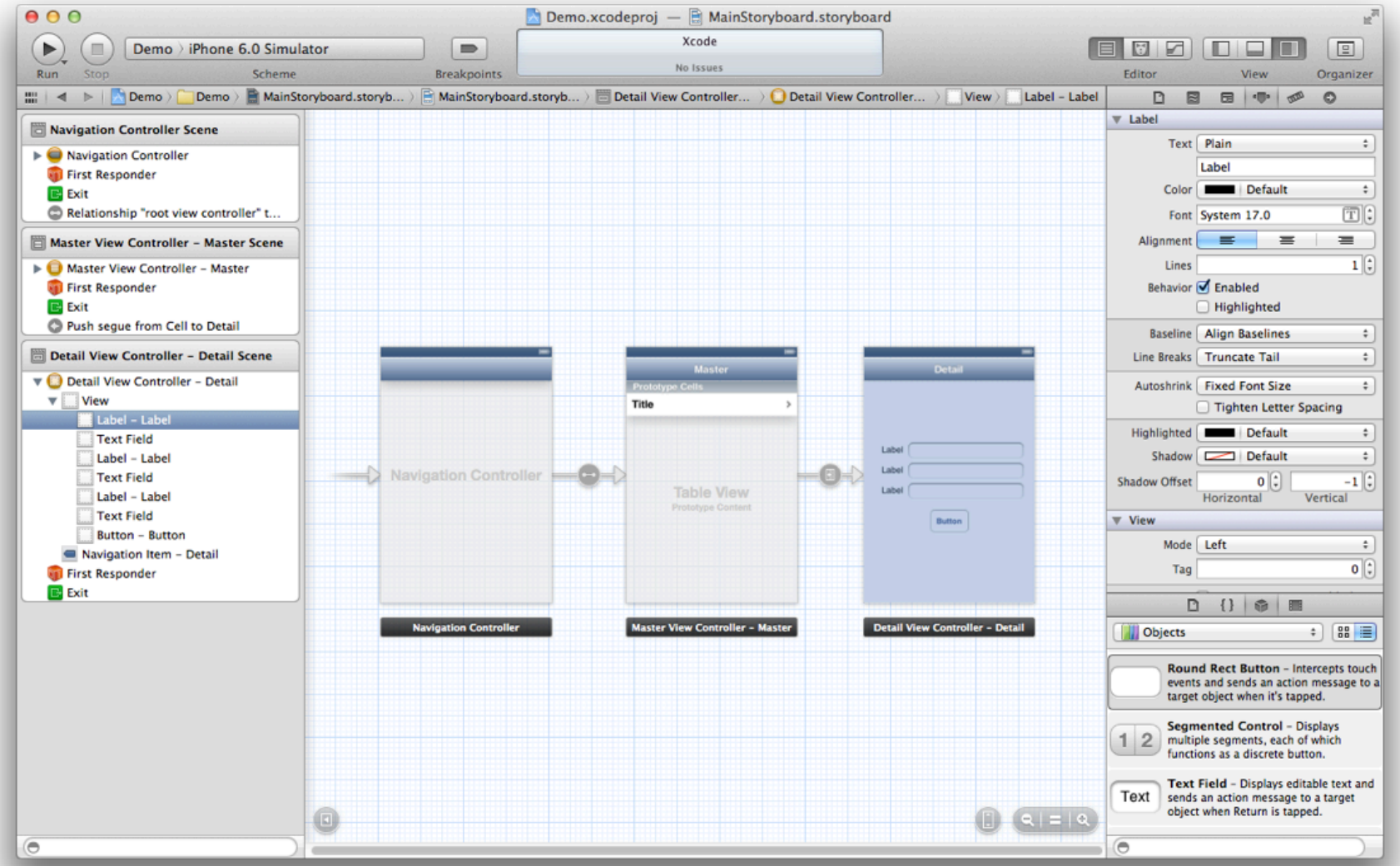
Instruments

Xcode (v4.5)

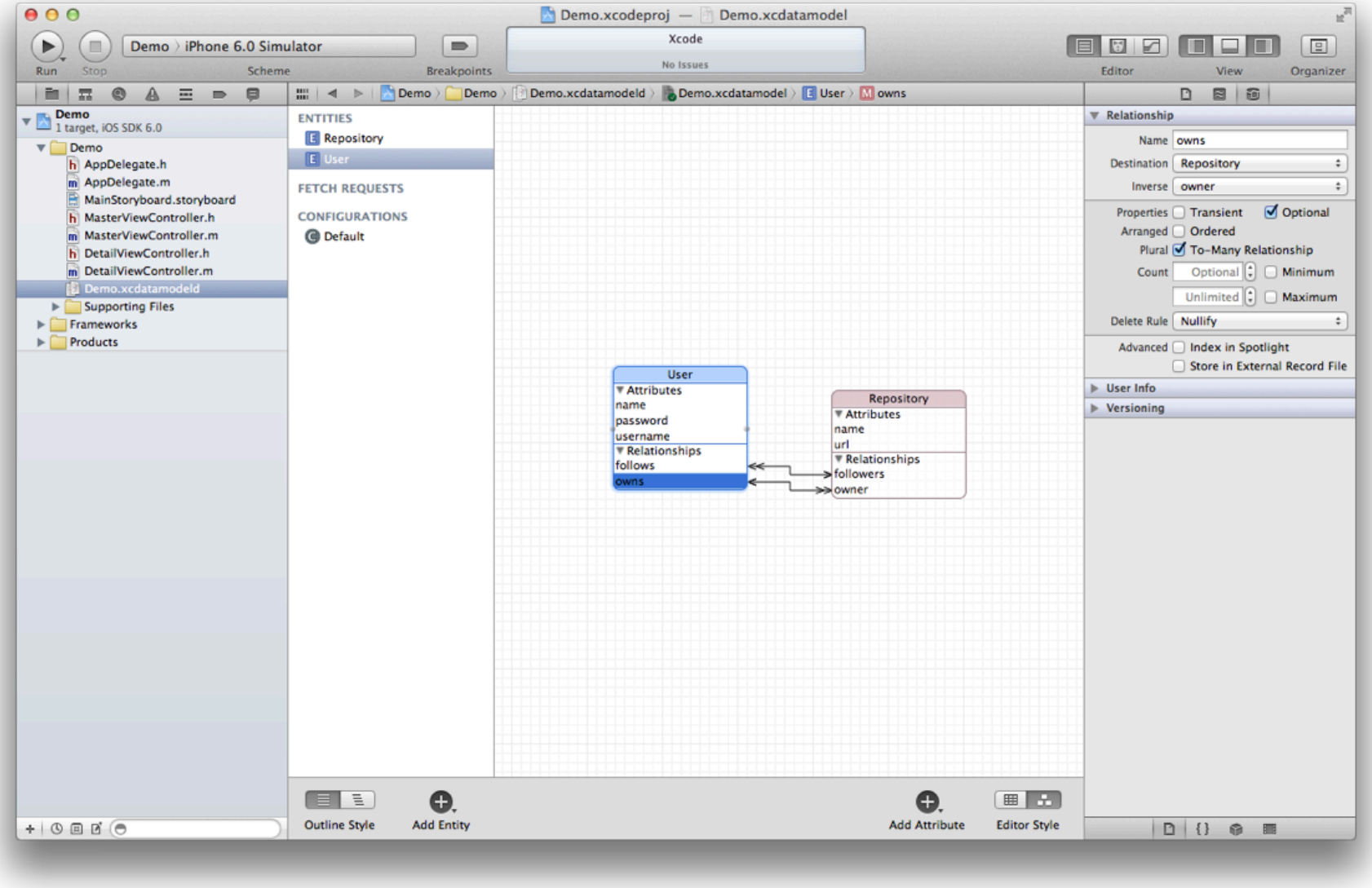
- Apple's IDE for Mac OS X *and* iOS
- Editor, debugger, GUI-builder, data modeler (and more!)


Demo.xcodeproj — AppDelegate.m
Run
Stop
Demo › iPhone 6.0 Simulator
Scheme
Breakpoints
Xcode
Editor
View
Organizer
Demo
1 target, iOS SDK 6.0
Demo
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
MasterViewController.h
MasterViewController.m
DetailViewController.h
DetailViewController.m
Demo.xcdatamodeld
Supporting Files
Frameworks
Products
Demo › Demo › AppDelegate.m › -application:didFinishLaunchingWithOptions:
#import "AppDelegate.h"
#import "MasterViewController.h"
@implementation AppDelegate
@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// Override point for customization after application launch.
UINavigationController *navigationController = (UINavigationController *)self.window.rootViewController;
MasterViewController *controller = (MasterViewController *)navigationController.topViewController;
controller.managedObjectContext = self.managedObjectContext;
return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application
{
// Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
// Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
}
- (void)applicationDidEnterBackground:(UIApplication *)application
{
// Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
// If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
}
- (void)applicationWillEnterForeground:(UIApplication *)application
{
// Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
}
- (void)applicationDidBecomeActive:(UIApplication *)application
{
// Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
}
Quick Help
Description The UIApplication class provides a centralized point of control and coordination for applications running on iOS.
Availability iOS (2.0 and later)
Declared UIApplication.h
Reference UIApplication Class Reference
Samples Audio Mixer (MixerHost), Regions, RosyWriter, Sampler Unit Presets (LoadPreset Demo), UICatalog
File Template Library

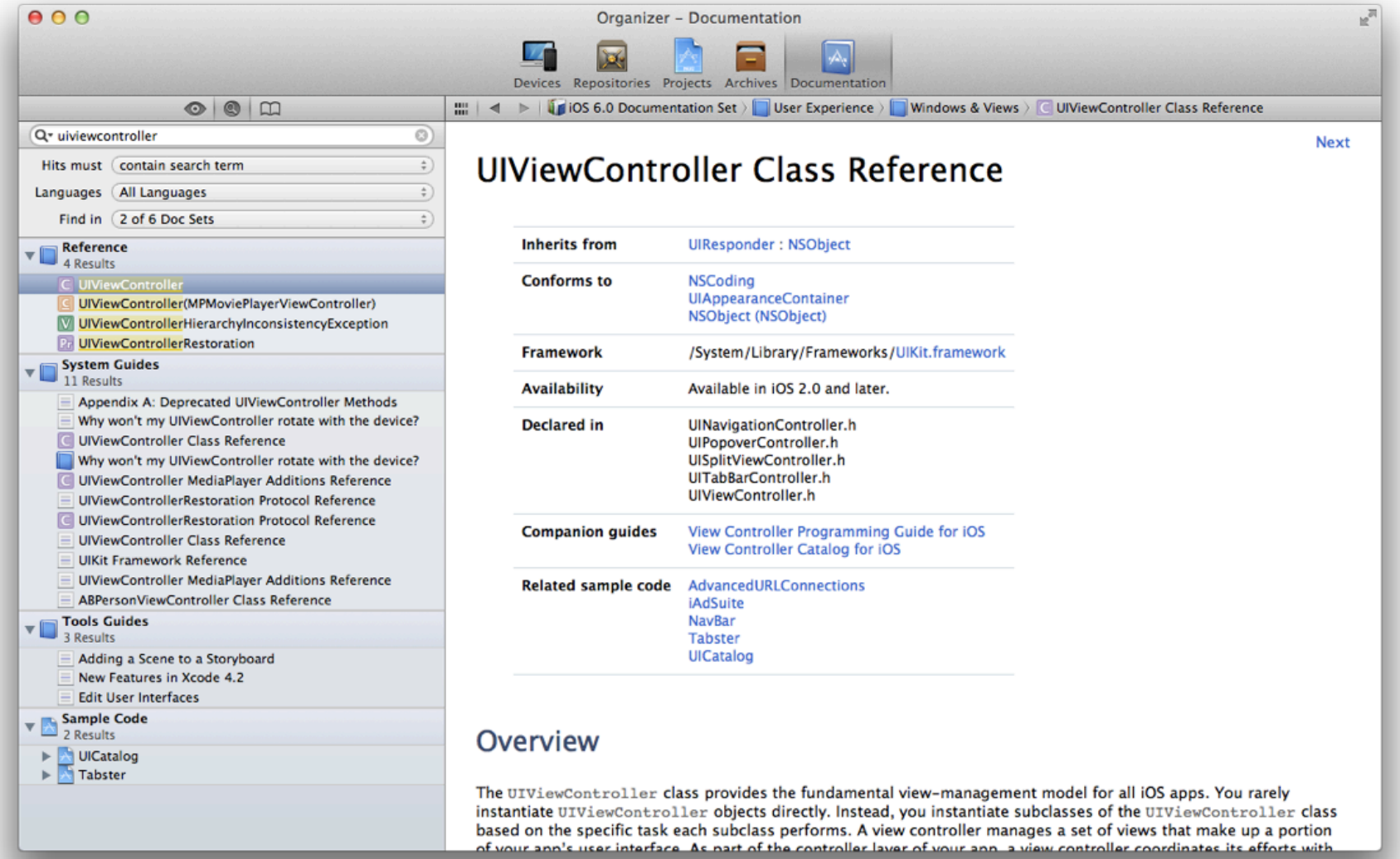
code editor



interface builder (IB)

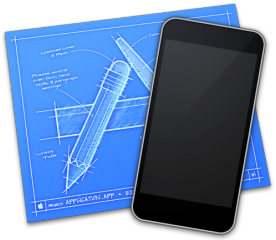


data (ER) modeler



documentation

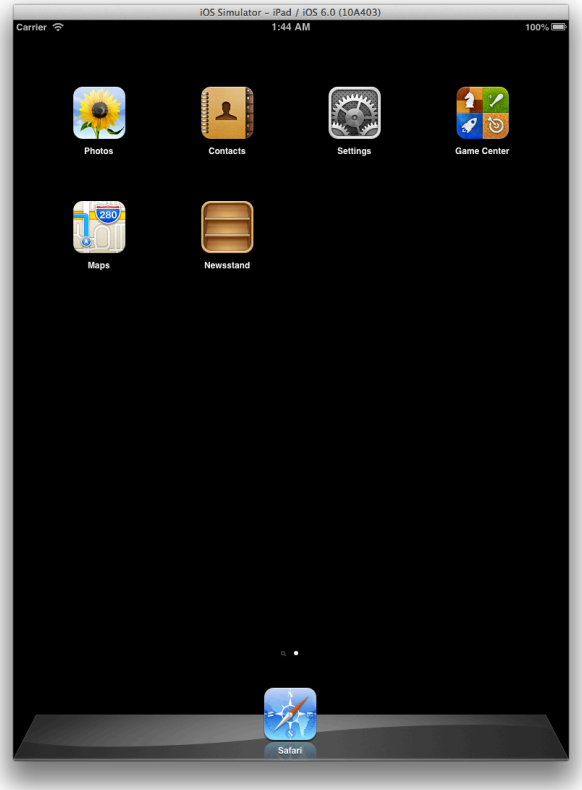
- Prone to typical Apple "radical overhaul" strategy — user base be damned
- Xcode is constantly mutating and evolving
  - … and sprouting new "features"
  - … and irritating a lot of (vocal) devs



iOS Simulator

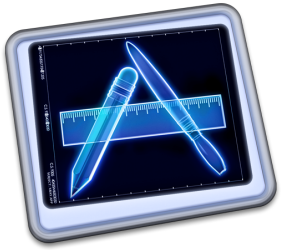
- Simulators for all iOS versions and devices
- Near complete software parity, but hardware features missing (e.g., camera, multi-touch > 2, accelerometer)

- Note: *simulator*, ***not*** *emulator!*
- An emulator very accurately mimics target platform
  - Executes same binary
  - At possible expense of speed (e.g., see Android emulator)



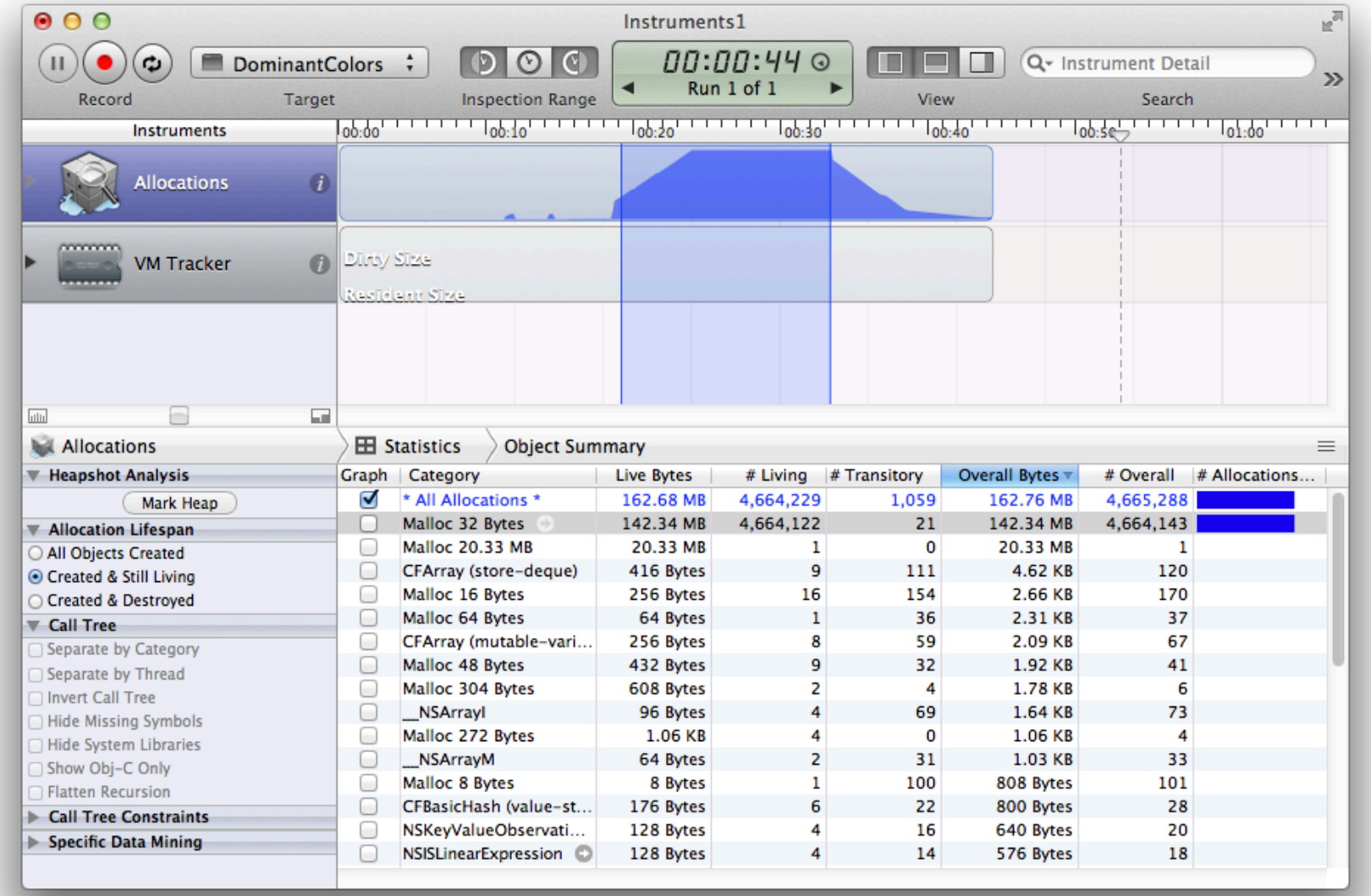
- iOS simulator actually runs Intel-native code (vs. ARM)
  - Fast! (Nice for dev & test)
    - But not indicative of actual speeds
  - Need to build for and test on device separately

- On-device deployment requires iOS dev account
  - = $99
  - http://developer.apple.com

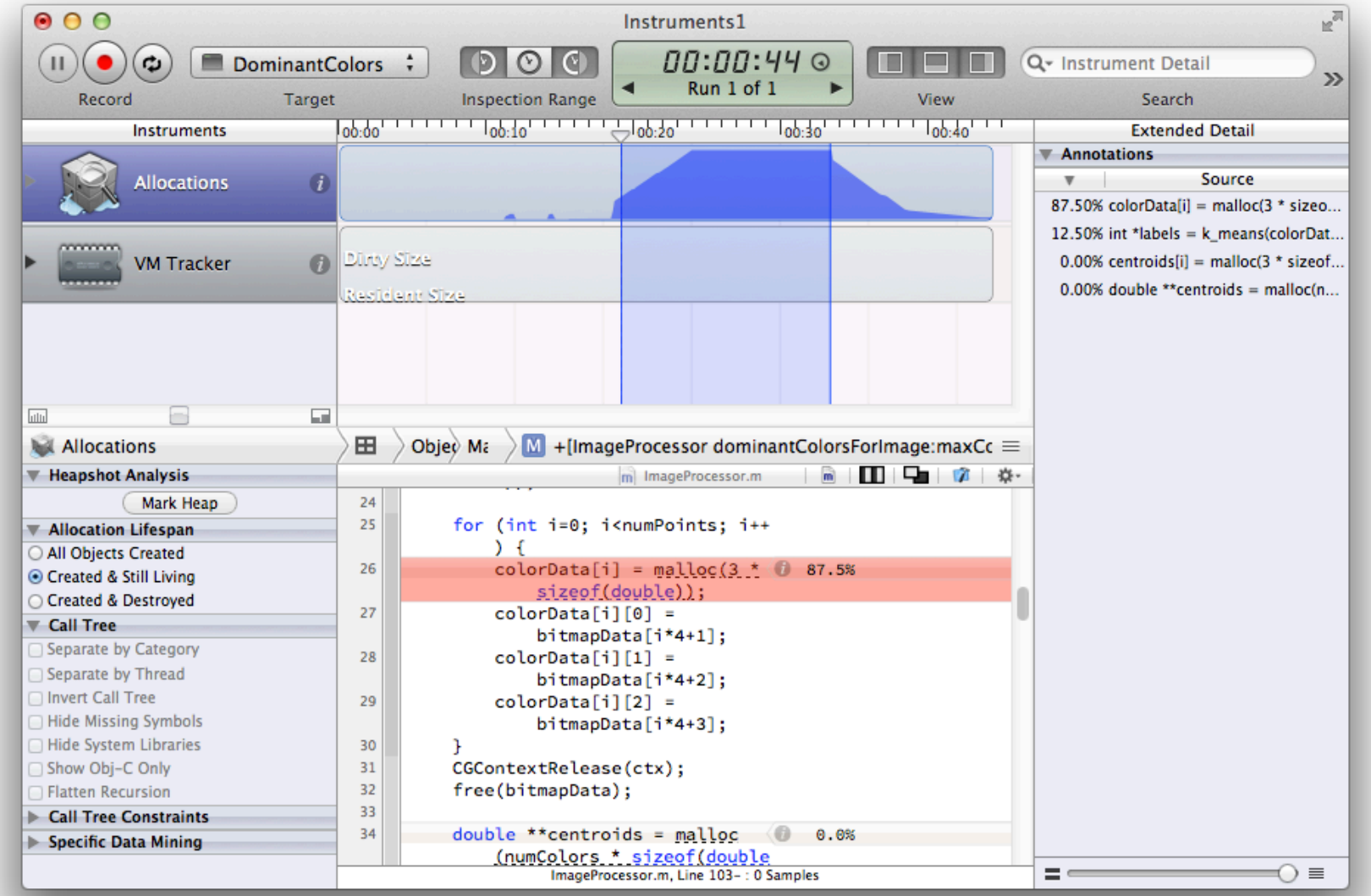


Instruments

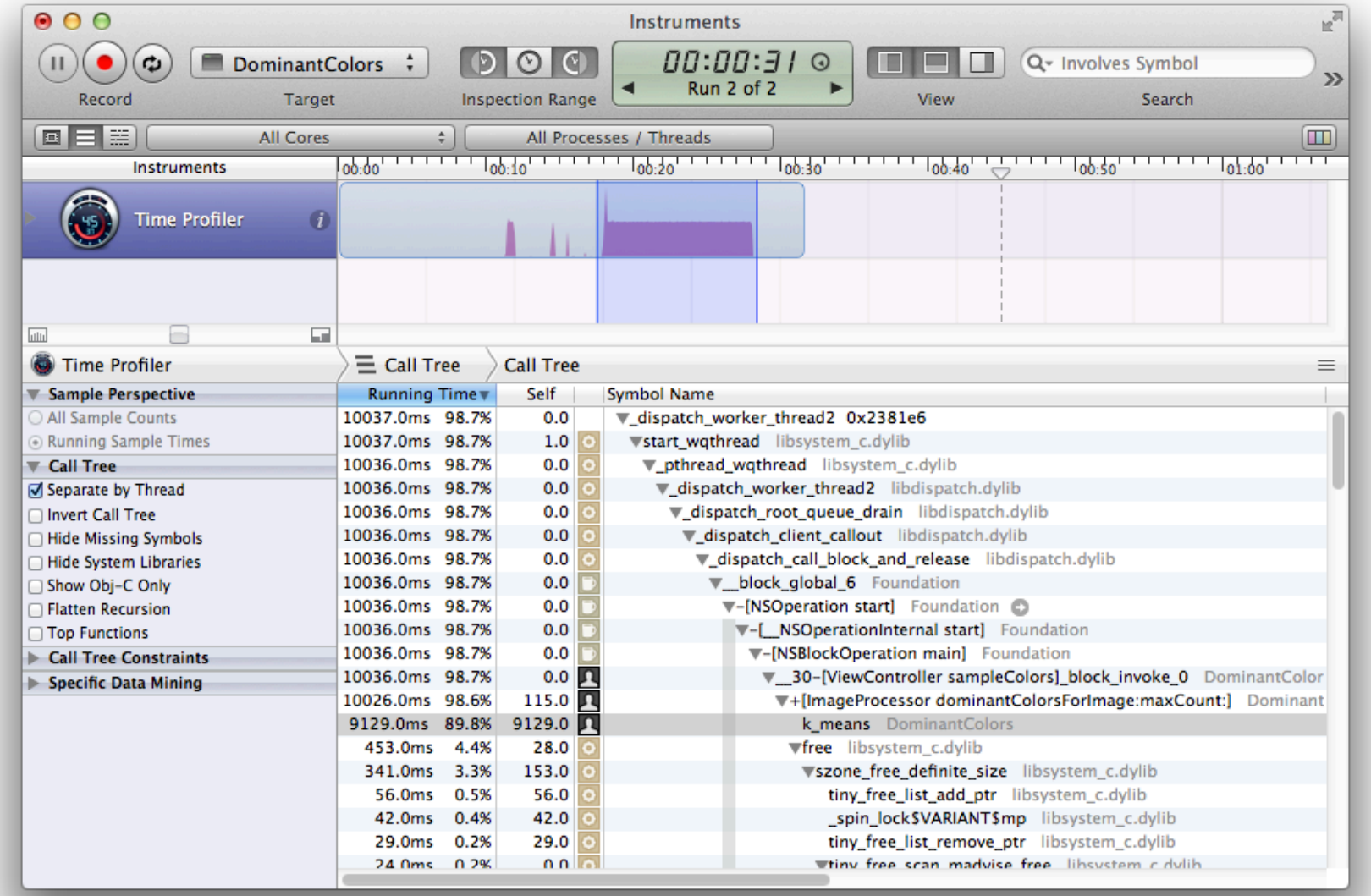
- Performance profiling & optimization tools
- Usage of memory, CPU, energy, network, I/O, graphics, etc.
- Really worth figuring out!



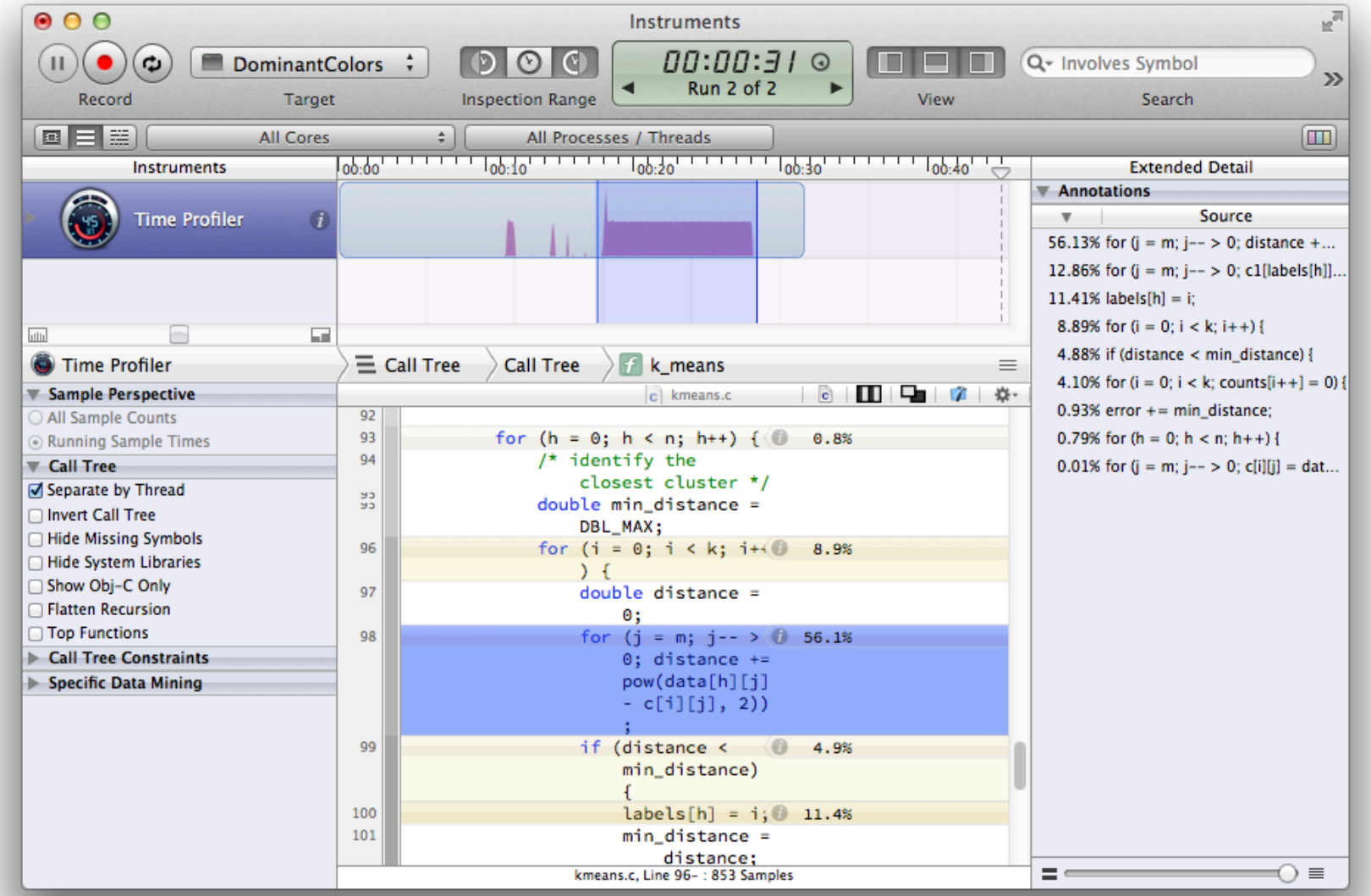
memory profiler



memory profiler (code annotations)



CPU time profiler



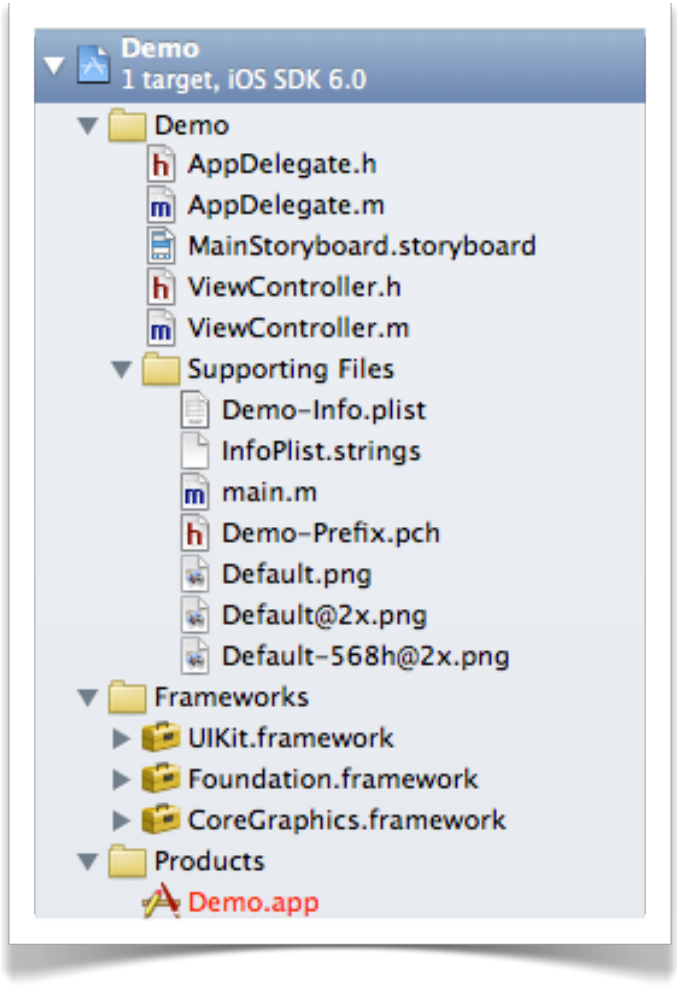
CPU time profiler (code annotations)

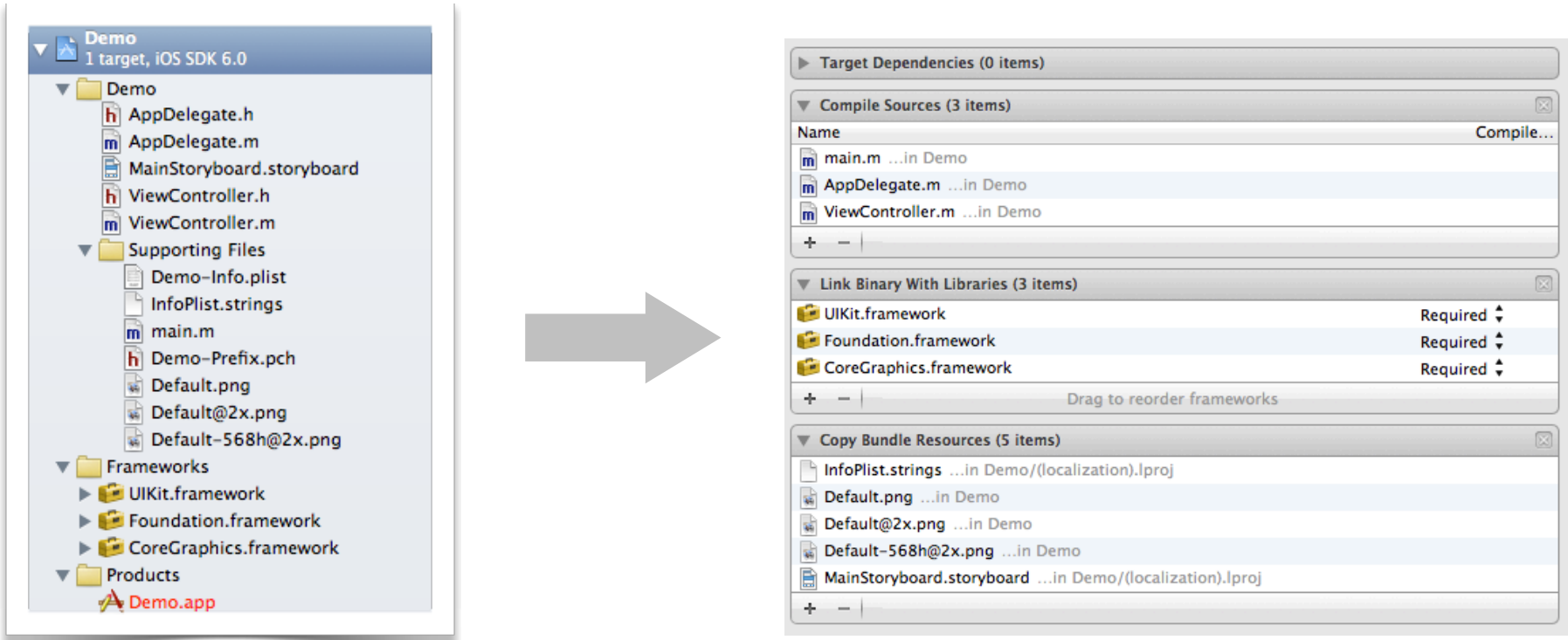
IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Hands-on

- Tour of developer tools
  - Xcode: prefs & workflow
  - Organizer & Documentation
  - iOS simulators
  - Instruments

§What's in an App?


Demo
1 target, iOS SDK 6.0
Demo
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
Demo-Info.plist
InfoPlist.strings
main.m
Demo-Prefix.pch
Default.png
Default@2x.png
Default-568h@2x.png
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
Demo.app

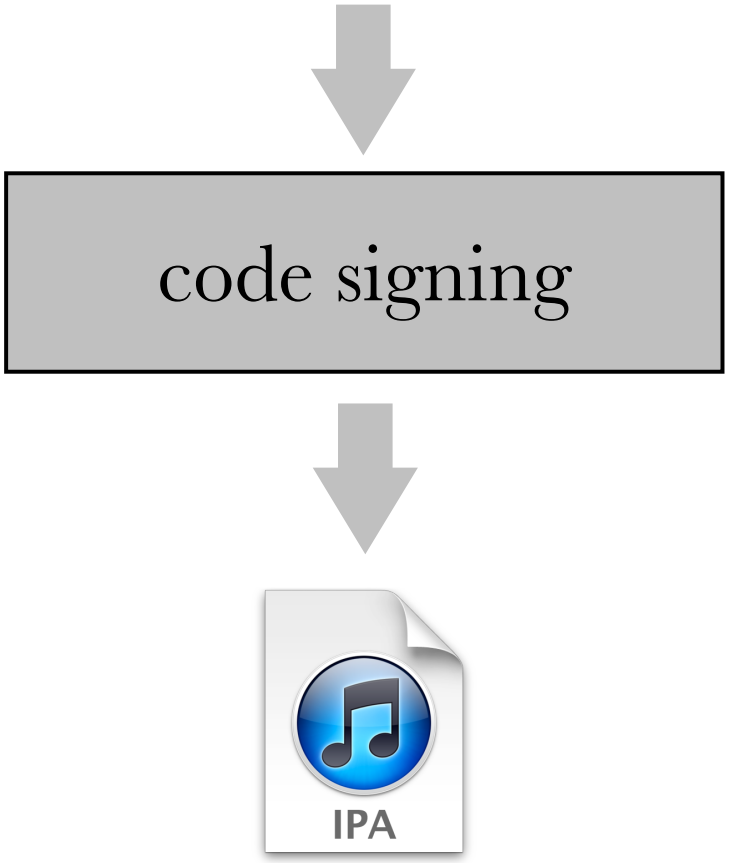


Xcode build settings

"Application Bundle"

{ executable,
property lists / settings,
icons + launch images,
localized strings,
storyboard,
other resources. }

“Application Bundle”



after deployment: Application “sandbox”

{ memory, keychain, file system }

Note: IPA can't be deployed on the simulator! (IPA contain ARM binary)

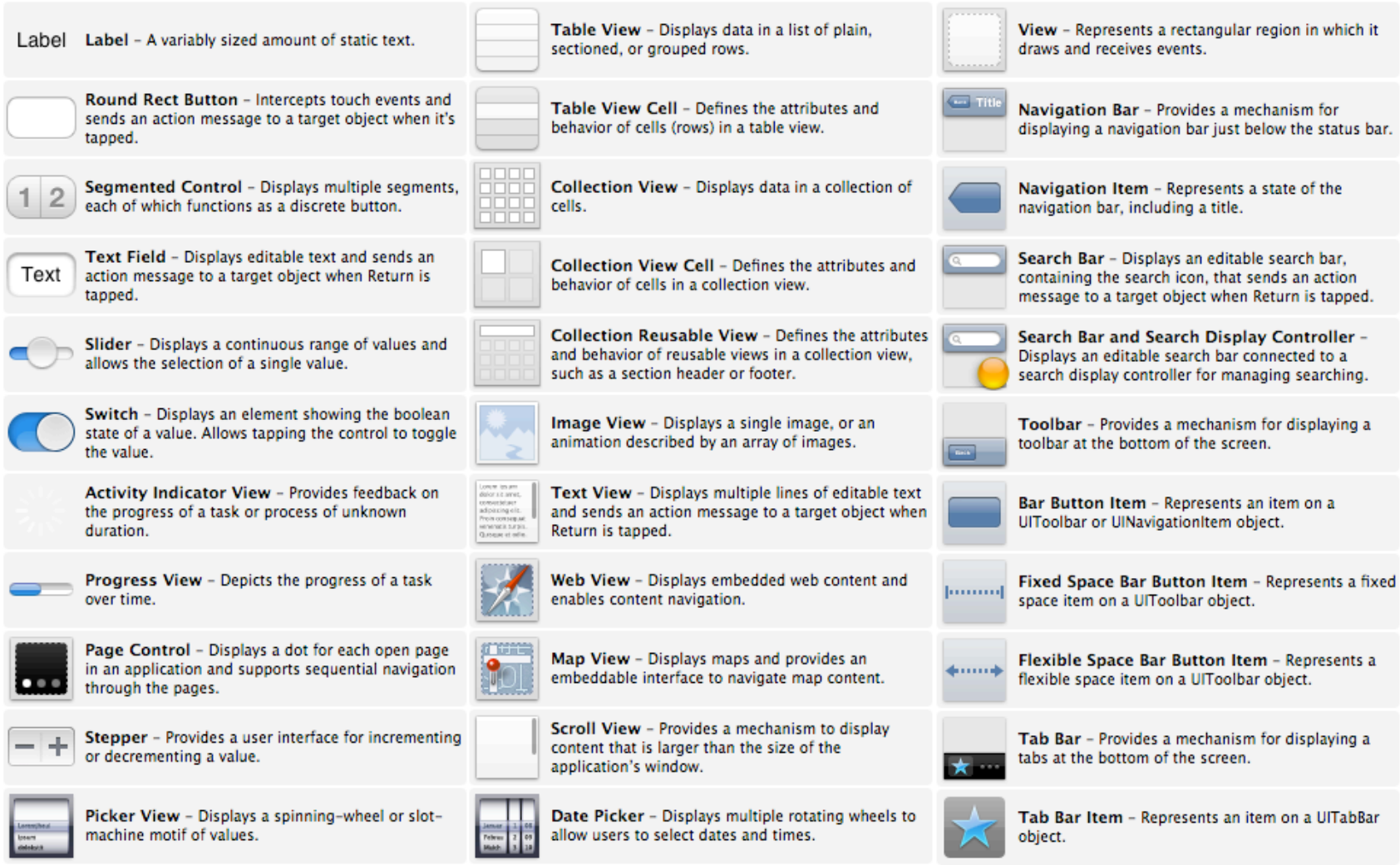
- but can use for on-device testing/ distribution
  - e.g., via TestFlight (testflightapp.com)

§GUI Building

Look ma, no code!

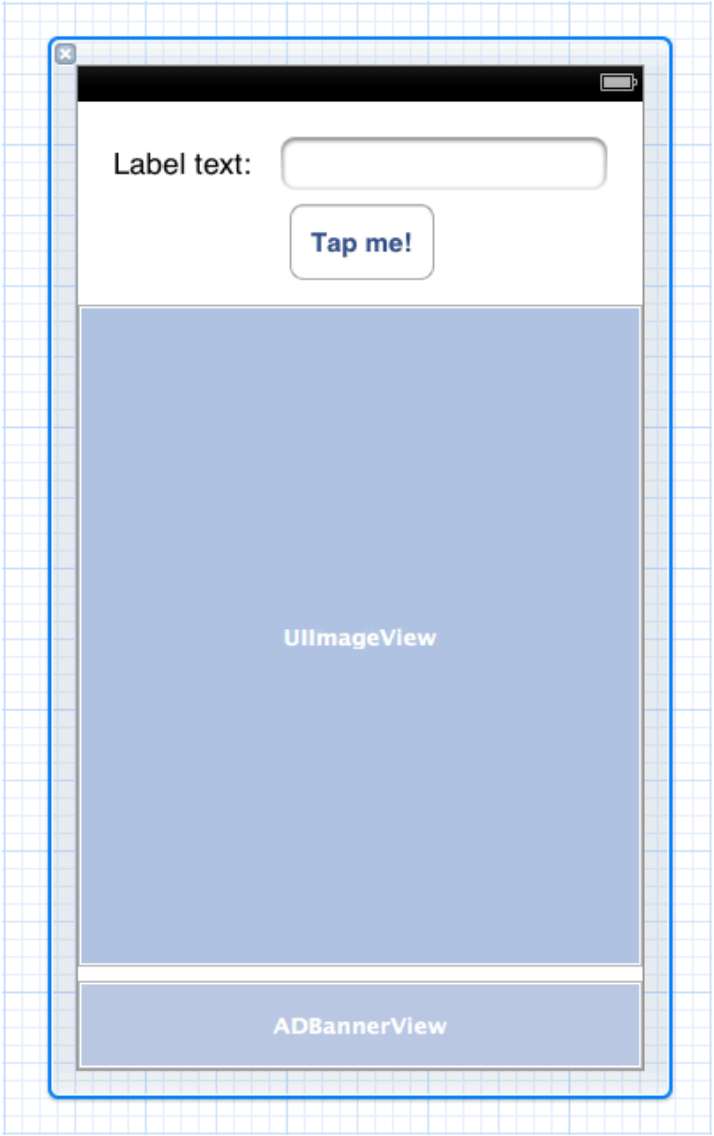
- Interface builder has gotten to where we can do a lot of GUI-creation with no code!
  - Not bad for rapid prototyping
- But non-trivial stuff still requires a lot of "manual" intervention; i.e., code

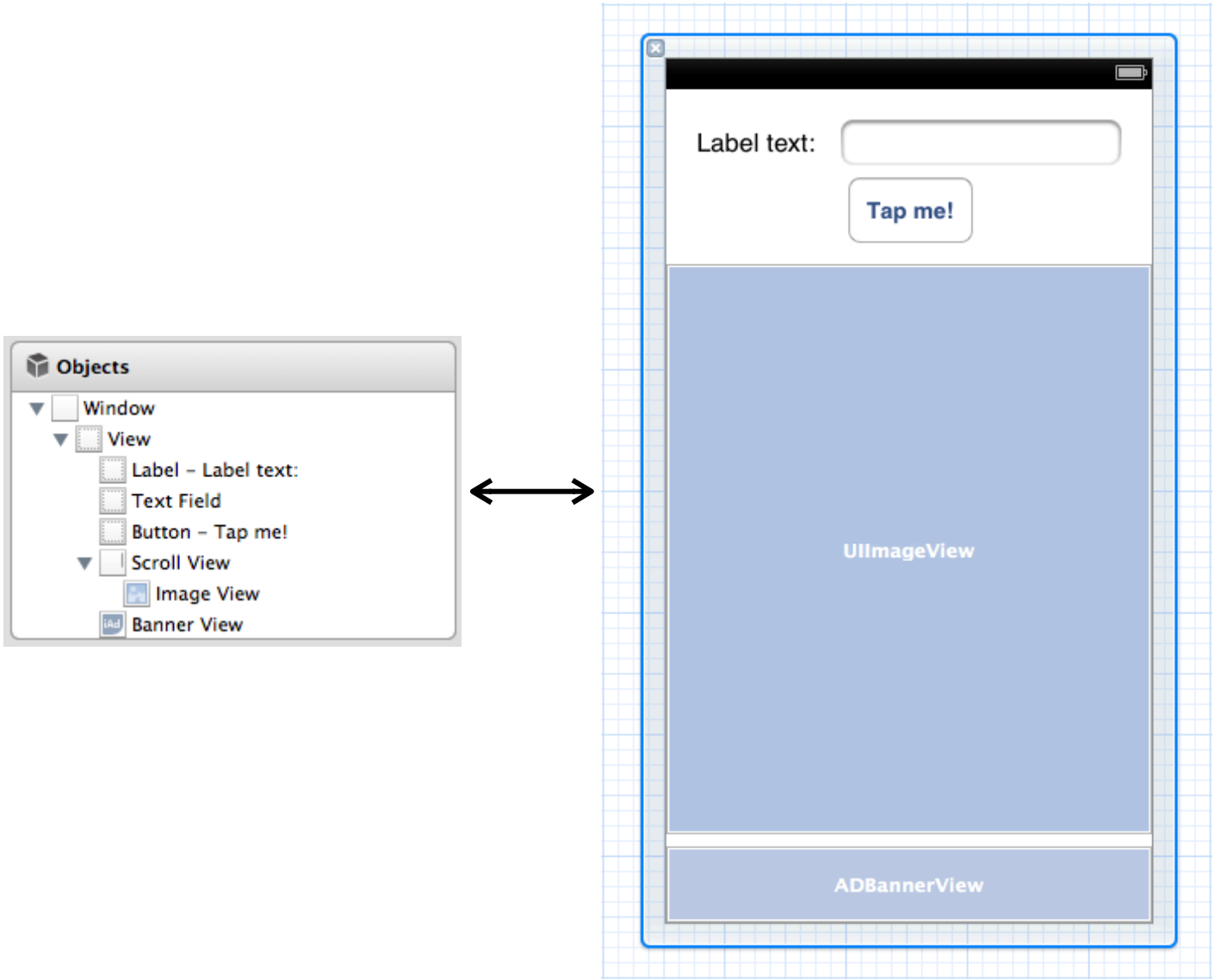
In iOS, a *visible interface element* is termed a ***view***



View types (partial)

Often, a ***scene*** in an app will consist of a single ***root*** *view* and a ***hierarchy*** *of subviews*


Label text:
Tap me!
UIImageView
ADBannerView


Objects
Window
View
Label – Label text:
Text Field
Button – Tap me!
Scroll View
Image View
Banner View
Label text:
Tap me!
UIImageView
ADBannerView

Two ways of defining scenes (without code):

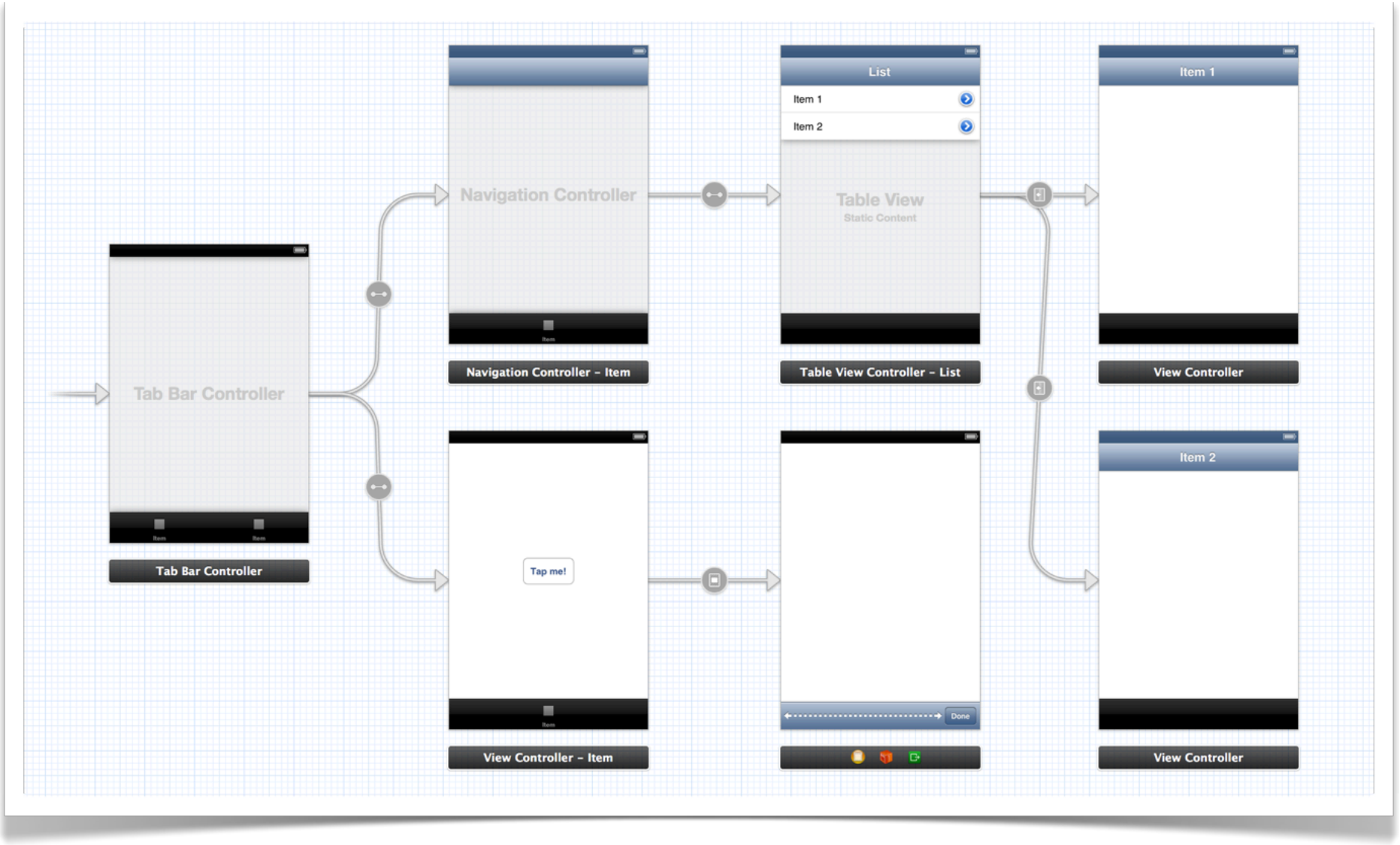
1. NIB files (the old way)
2. Storyboards

The old way: NIB files

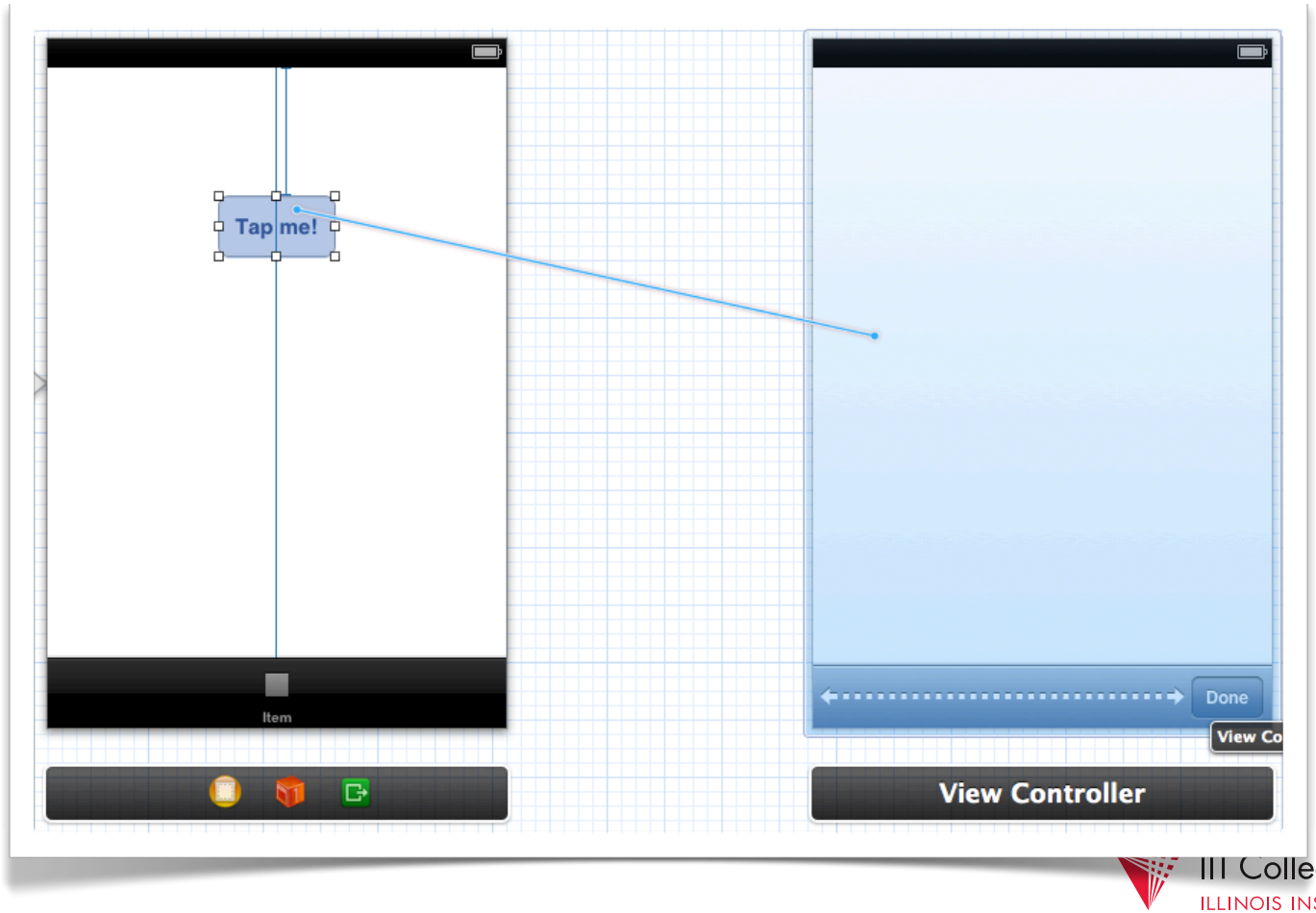
- ***N**ext **I**nterface **B**uilder* files — freeze dried info for a single "scene" in the app
  - Loaded on demand when scene is needed
- Problem: what goes into separate NIB files is determined manually (non-trivial)
  - Also hard to get overview of app

The new: Storyboards

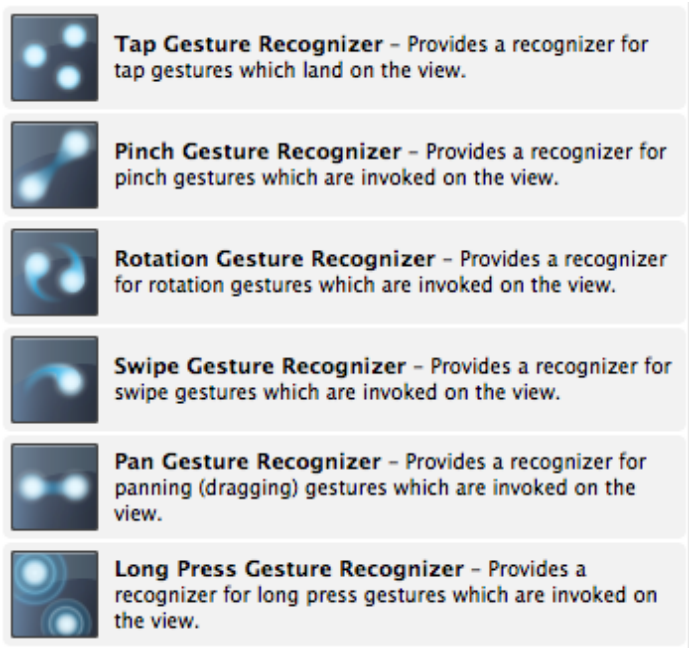
- Contains all scenes (potentially) for an app, and transitions – ***segues*** – between them
  - Individual scene objects are automatically loaded/managed

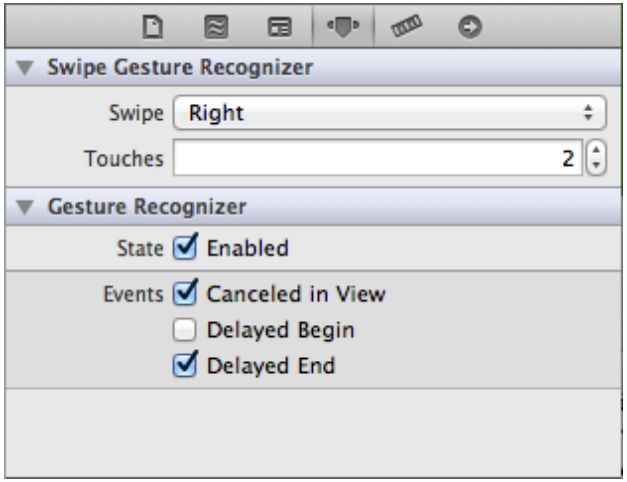

Tab Bar Controller
Tab Bar Controller
Navigation Controller
Navigation Controller – Item
List
Item 1
Item 2
Table View
Static Content
Table View Controller – List
Item 1
View Controller
Tap me!
View Controller – Item
Done
Item 2
View Controller

- Segues can be associated with events in "control" type views (e.g., buttons)



- Segues can also be associated with gestures, directly within IB





- Of course, scenes (and discrete objects) created in IB can be accessed from code
  - Note: segues are objects, too!

Hands-on

- Storyboard-ing
  - Individual scenes
  - Navigation, Tabbed, and List scenes
  - Modal / Push / Relationship segues
  - Gestures

Limitations (of solo IB)

- Not possible to customize event-handling
- Cannot dynamically create/load content
- Cannot define return/exit segues

Another important issue when designing and building GUIs: *layout flexibility / elasticity*

Some display disparities:

1. iPhone ≠ iPad: aspect ratio + size
2. Retina ≠ Non-Retina: pixel density
3. iPhone<5 ≠ iPhone 5: aspect ratio
4. Profile ≠ Landscape: width/height

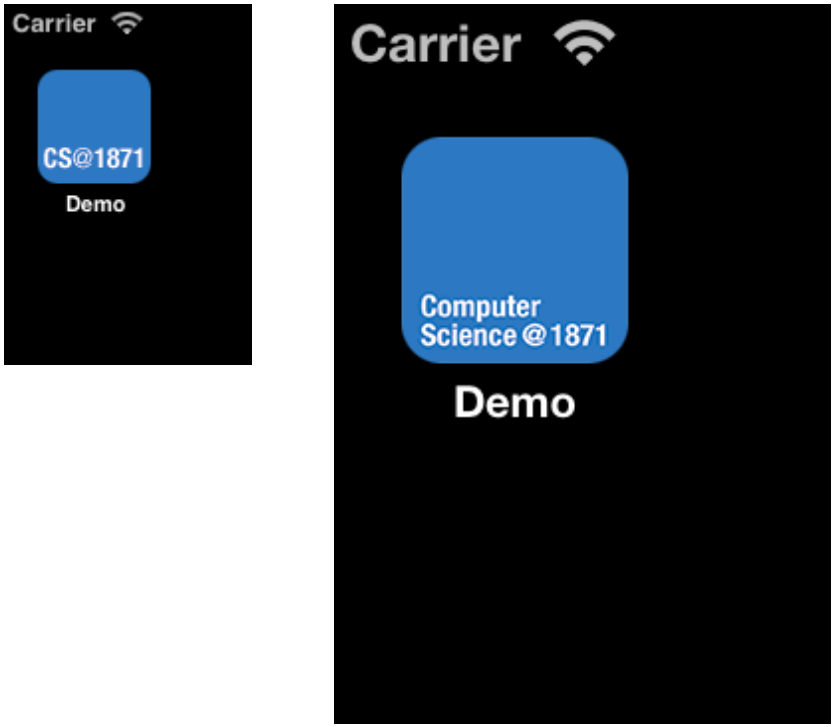
1. iPhone ≠ iPad: aspect ratio + size
   - Usually require entirely different layouts
   - Accommodate different UI conventions


Carrier
3:46 PM
Settings
Bluetooth
Off
General
Privacy
Maps
Safari
Photos & Camera
Twitter
Facebook
Carrier
3:47 PM
100%
Settings
Bluetooth
Off
General
Privacy
Maps
Safari
Photos & Camera
Twitter
Facebook
General
About
Keyboard
International
Accessibility
Reset

2. Retina ≠ Non-Retina: pixel density

- IB layouts are resolution agnostic
- Placement/size is in *points*
  - "Retina point" = 2×2 pixels
- Just need to be careful with bitmaps

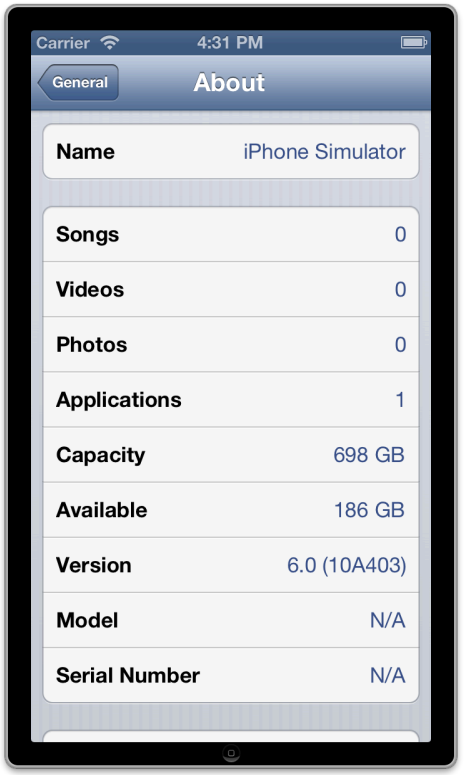
Image functions will automatically choose "@2x" suffixed image resources, if supplied



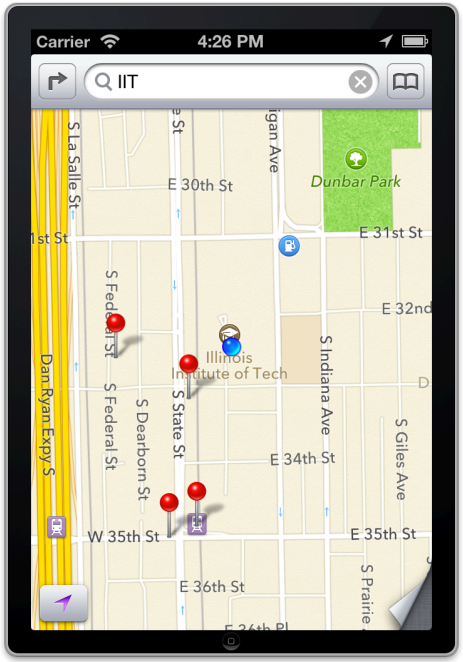
3. iPhone<5 ≠ iPhone 5: aspect ratio

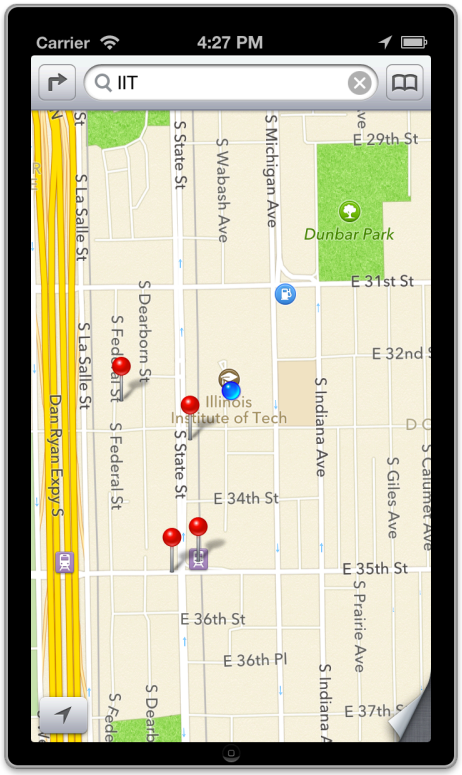
would really prefer *not* to develop separate custom layouts (i.e., separate storyboards) for such similar form factors…

Many built-in views automatically "stretch"



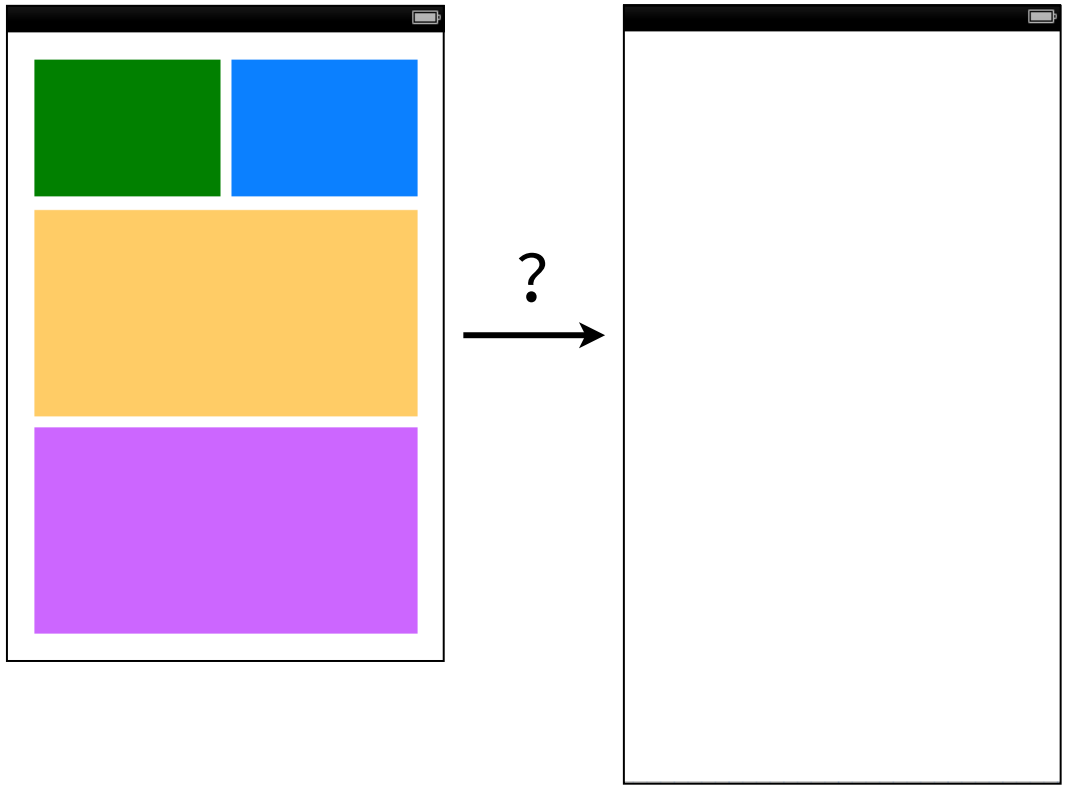
Many built-in views automatically "stretch"





IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

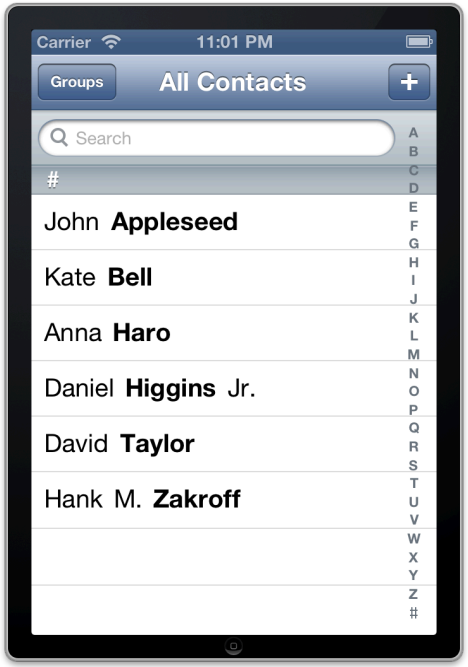
But custom views (e.g., comprised of multiple subviews) require stretching "hints"

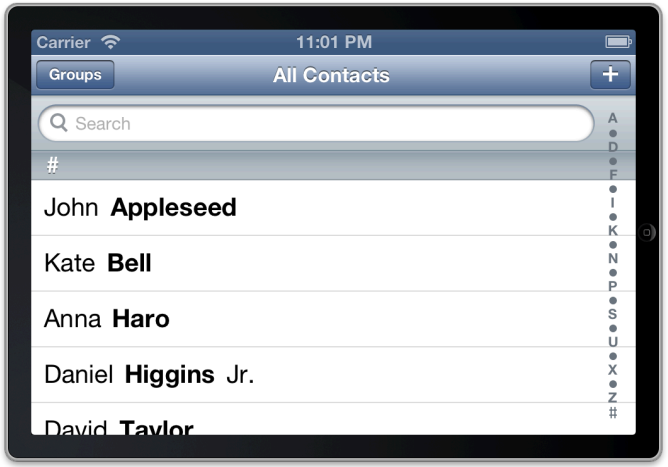


4. Profile ≠ Landscape: width/height

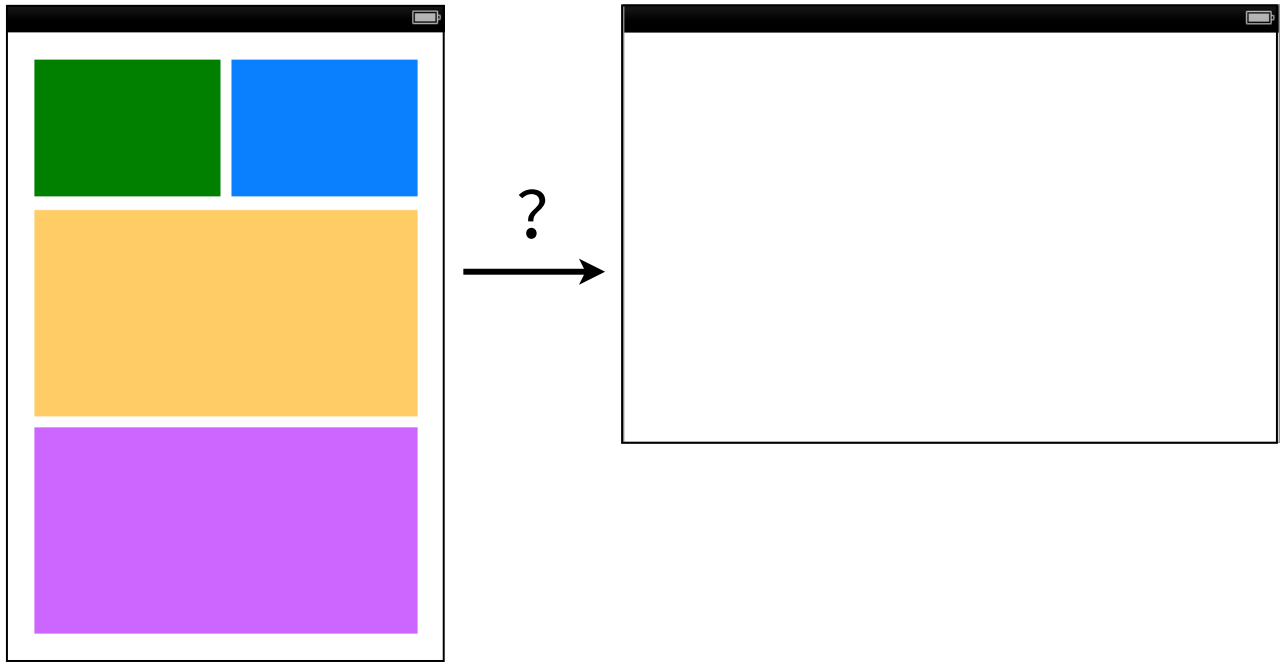
- different perspective, but same screen
- depending on app, may require entirely separate layouts or could *possibly* take advantage of elastic layouts

Again, many built-in views automatically stretch





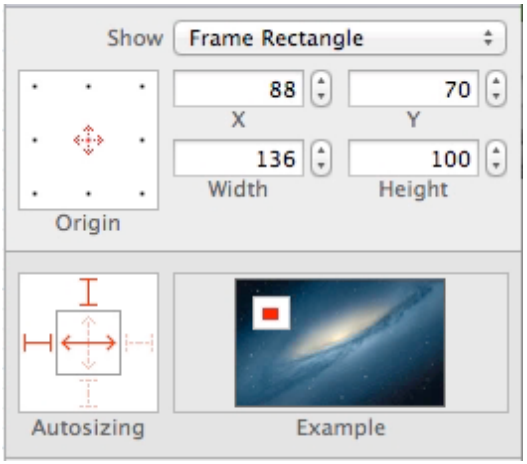
But, custom views?



Flexible layouts (old way): “springs & struts”

- a view can stretch horizontally, vertically, or both (*springs*)
- a view can be anchored (via *struts*) to the edge on any side

e.g., springs & struts

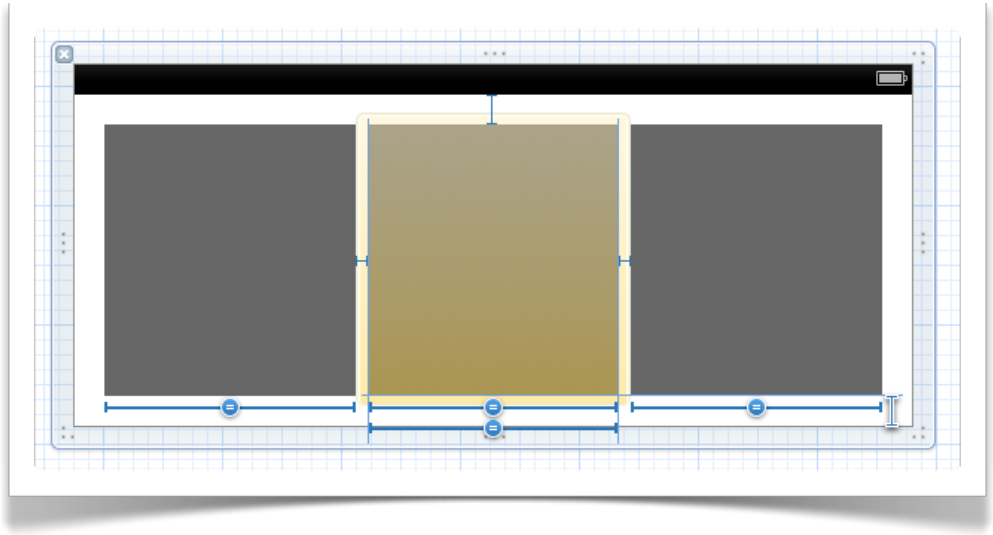


- requires that the view be given an explicit initial size and position (aka “frame”)
  - fine-grained control is hard to achieve

Flexible layouts (new way): Autolayout

- a view can have one or more *constraints* defined on it
  - for *properties* like size/position
  - or for *relationships* between views
    - e.g., equal widths, right-aligned, etc.

e.g., autolayout



- permits "declarative" layouts; i.e., minimal *hardcoding* of explicit sizes/positions

Hands-on

- Project: *LayoutTesting*
- Springs & Struts
- Autolayout

§Objective-C

The Language

- Superset of ANSI C

```c
/* I'm valid Objective-C, too! */

void say_hi()
{
    printf("hello world\n");
}

int main(int argc, const char *argv[])
{
    say_hi();
    return 0;
}
```

The Language

- Object-oriented (à la Smalltalk)

```
// Create and initialize an instance of the "Widget" class
Widget *wdgt = [[Widget alloc] init];

// Set a widget property
wdgt.name = @"thingamajig";

// Ask the widget to do something; i.e., send it a message
[wdgt turnSprocket:@"sprock-xyz" rotations:3.5 speed:10.0];
```

- Separate header (.h) & implementation (.m)

```
@interface Widget : NSObject
@property (strong, nonatomic) NSString *name;

- (NSString *)description;
- (void)turnSprocket:(NSString *)name
           rotations:(CGFloat)deg
               speed:(CGFloat)rpm;
@end
```

```
@implementation Widget
- (NSString *)description
{
    return [NSString stringWithFormat:@"Widget: %@", self.name];
}

- (void)turnSprocket:(NSString *)name
           rotations:(CGFloat)deg
               speed:(CGFloat)rpm
{ /* ... */ }
@end
```

The Language

- Single-inheritance; multi-protocol adoption

```
// this widget conforms to the drilling & welding protocols
Widget<Drilling,Welding> *wdgt = ...;

// so in addition to doing what widgets do ...
[wdgt turnSprocket:@"sprock-abc" rotations:5.0 speed:8.0];

// we can drill with it
[wdgt drillWithBit:2.5 atSpeed:2500];

// and weld with it
[wdgt weldAtTemp:2500];
```

- Protocols declare APIs, not implementation

```
@protocol Drilling
- (void)drillWithBit:(CGFloat)size atSpeed:(CGFloat)rpm;
@end

@protocol Welding
- (void)weldAtTemp:(CGFloat)celsius;
@optional
- (void)coolAtTemp:(CGFloat)celsius;
@end
```

```
@interface Widget : NSObject <Welding, Drilling>
@end
```

```
@implementation Widget
- (void)drillWithBit:(CGFloat)size atSpeed:(CGFloat)rpm
{ /* ... */ }

- (void)weldAtTemp:(CGFloat)celsius
{ /* ... */ }
@end
```

The Language

- Syntactic sugar for standard library objects

```
// string objects
NSString *str = @"hello!";

// array objects
NSArray *widgets = @[ [[Widget alloc] initWithName:@"alligator"],
                      [[Widget alloc] initWithName:@"allen"],
                      [[Widget alloc] initWithName:@"monkey"] ];

// dictionary (a.k.a. map) objects
NSDictionary *aliasMap = @{ @"gator" : widgets[0],
                            @"al"    : widgets[1],
                            @"curly" : widgets[2]  };
```

The Language

- Anonymous functions (closures/lambdas)

```
NSDictionary *aliasMap = @{ @"gator" : widgets[0],
                            @"al"    : widgets[1],
                            @"curly" : widgets[2]  };

// print key/object pairs using a for loop
for (NSString *key in aliasMap) {
    Widget *val = [aliasMap objectForKey:key];
    NSLog(@"%@ => %@", key, val);
}

// print key/object pairs using a block
[aliasMap enumerateKeysAndObjectsUsingBlock:^(id key, id val, BOOL *stop){
    NSLog(@"%@ => %@", key, val);
}];
```

Typical ObjC Code

```
- (void)imagePickerController:(UIImagePickerController *)picker
        didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    UIImage *image = [info objectForKey:UIImagePickerControllerOriginalImage];
    if (image) {
        [_imageView removeFromSuperview];
        _imageView = [[UIImageView alloc] initWithImage:image];
        _scrollView.contentSize = image.size;
        [_scrollView addSubview:_imageView];
        [_queue addOperationWithBlock:^{
            [self processImage:_imageView.image];
        }];
    }
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

The Runtime

- ObjC is very dependent on *runtime library* (written in C) for behavior and features
  - Dynamic message dispatch/forwarding
  - Object introspection
  - Automatic reference counting

The Runtime

- In-depth understanding (typically) of limited practical use
  - But useful for optimization / fine-tuning!

Hands-on

- Project: *ObjCPlaytime*
- Xcode IDE (for code)
  - Building, running, debugging
  - Controlling behavior
  - Completion, snippets, navigation, etc.

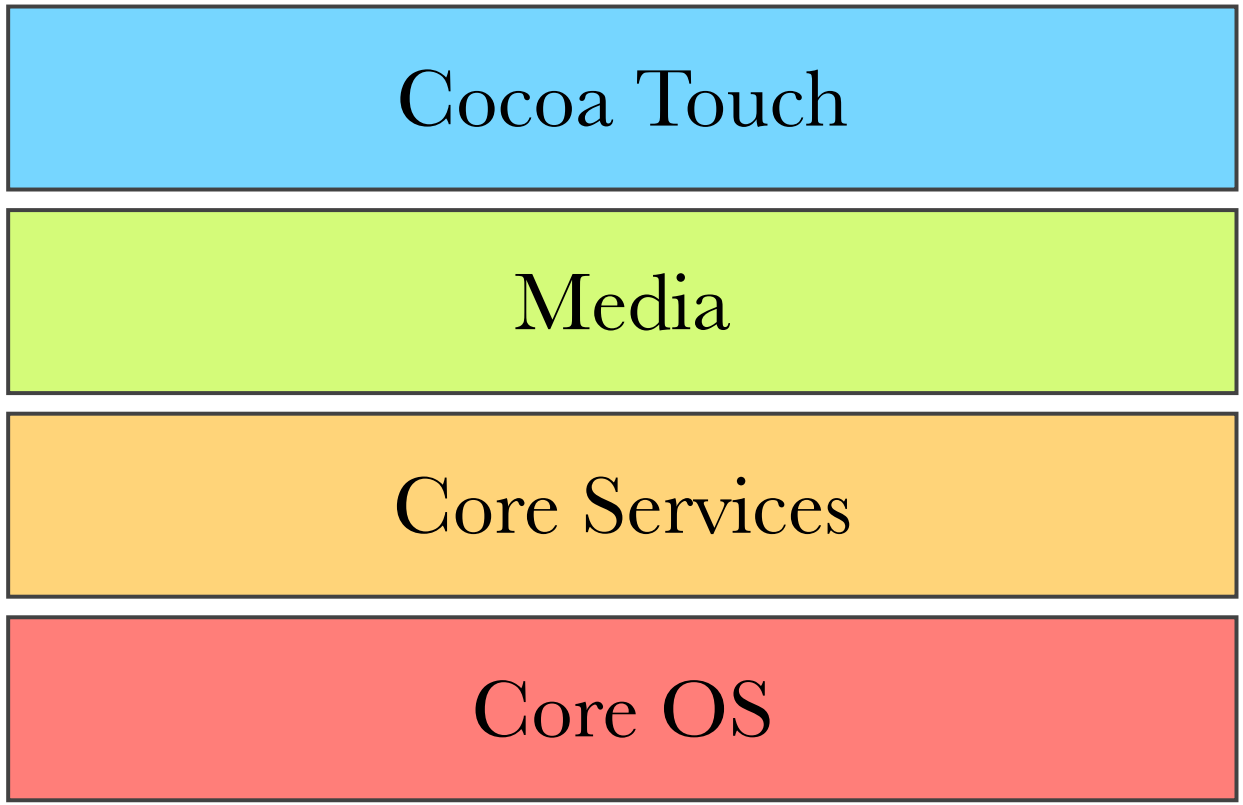
On top of the language & runtime, we also have our *application development libraries*, a.k.a. *frameworks*

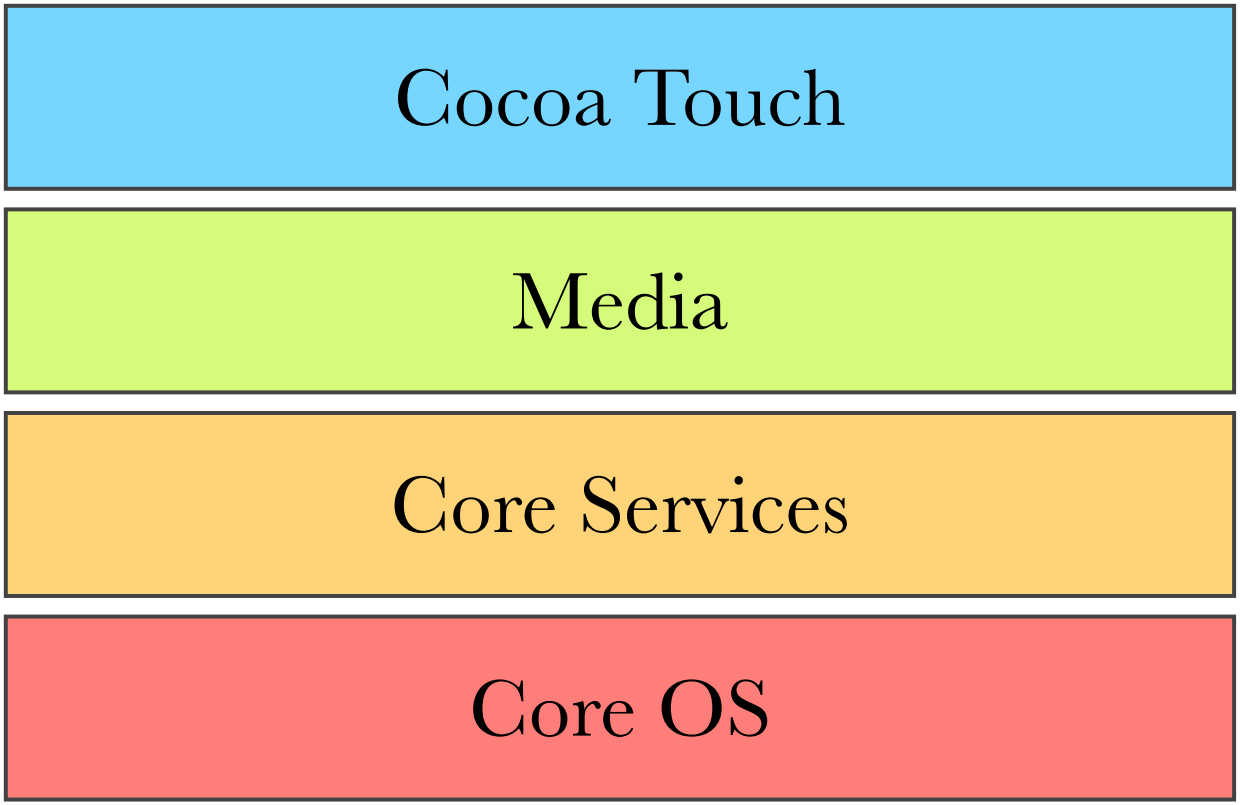
every native app makes heavy use of these!

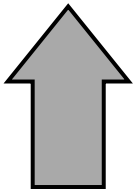
§iOS SDK Overview

Apple-provided frameworks fall into different layers of the iOS *architectural stack*

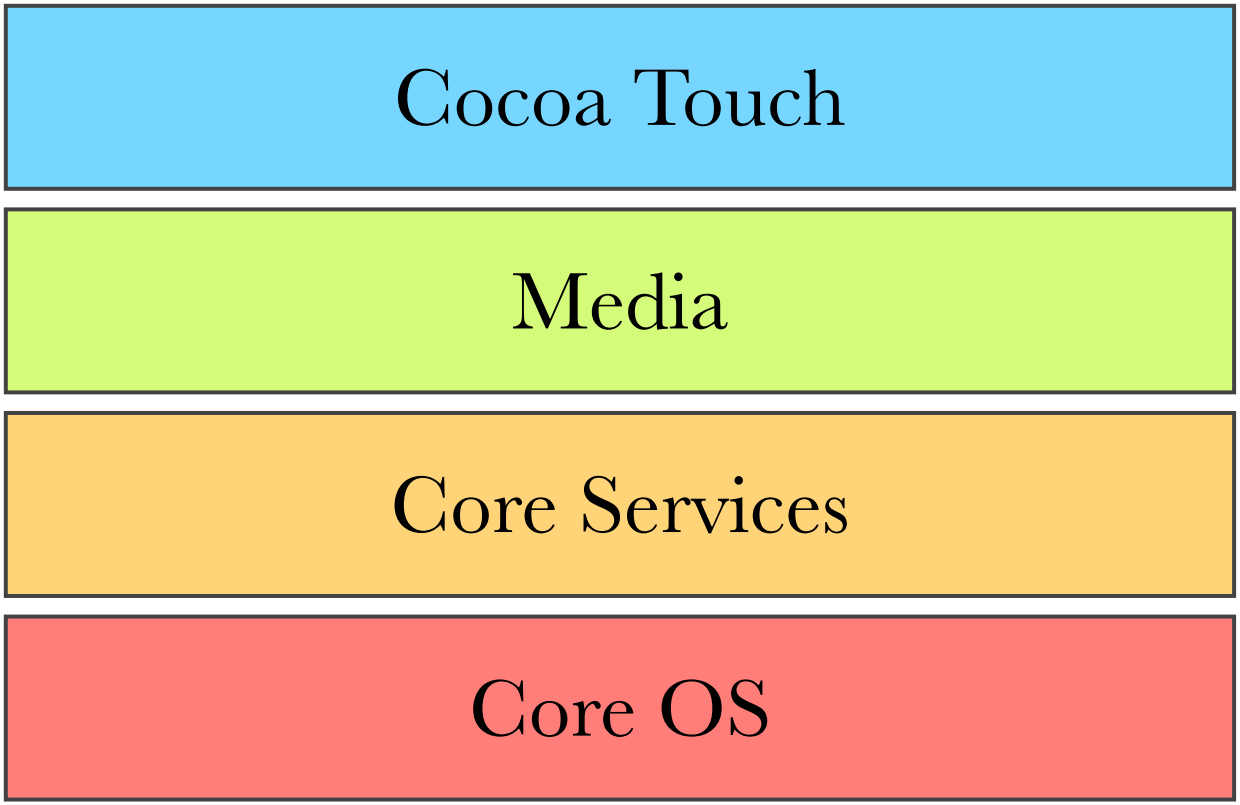
Apple-provided frameworks fall into different layers of the iOS *architectural stack*

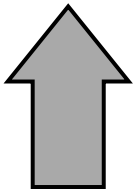




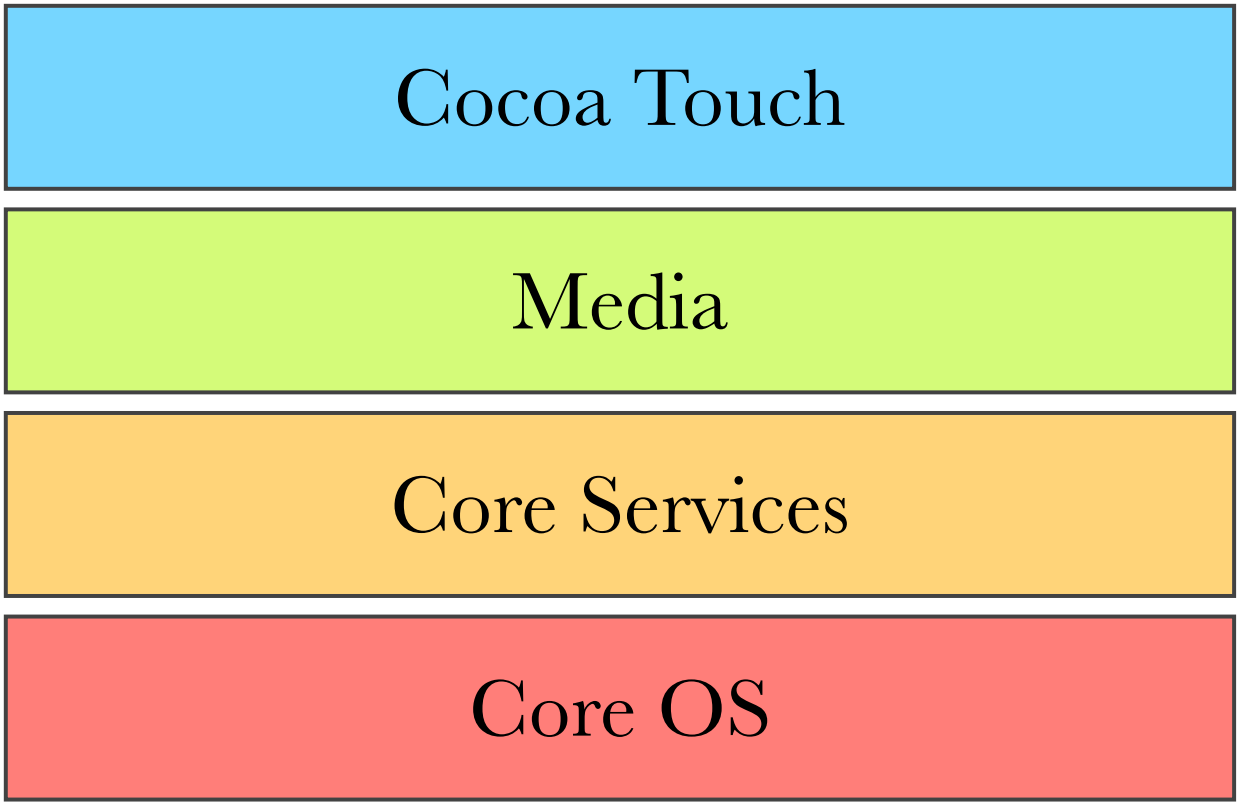


object-oriented,
more abstract,
less code needed

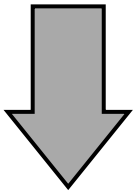


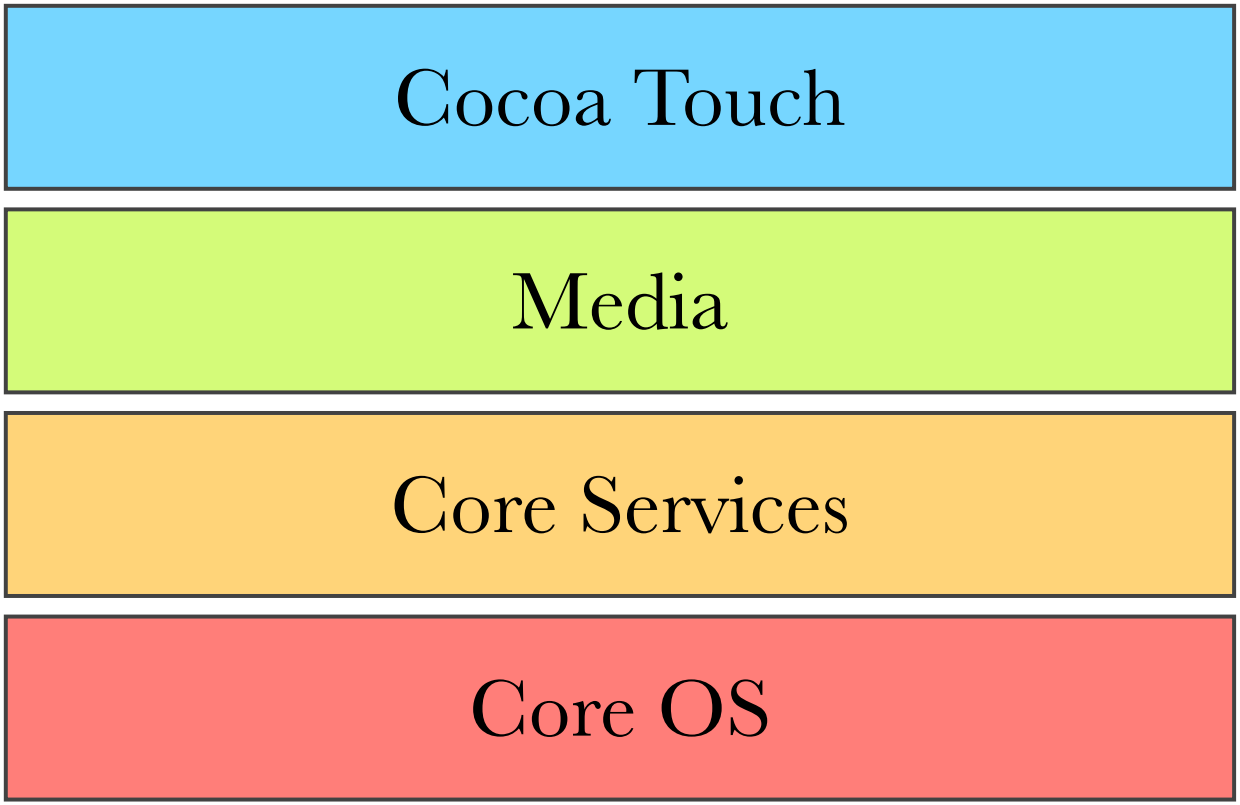


less flexible,
less fine-tunable,
more overhead

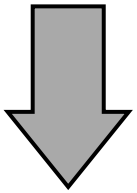


may be procedural, more granular, exposes hardware





complex APIs,
more details …
… more code!

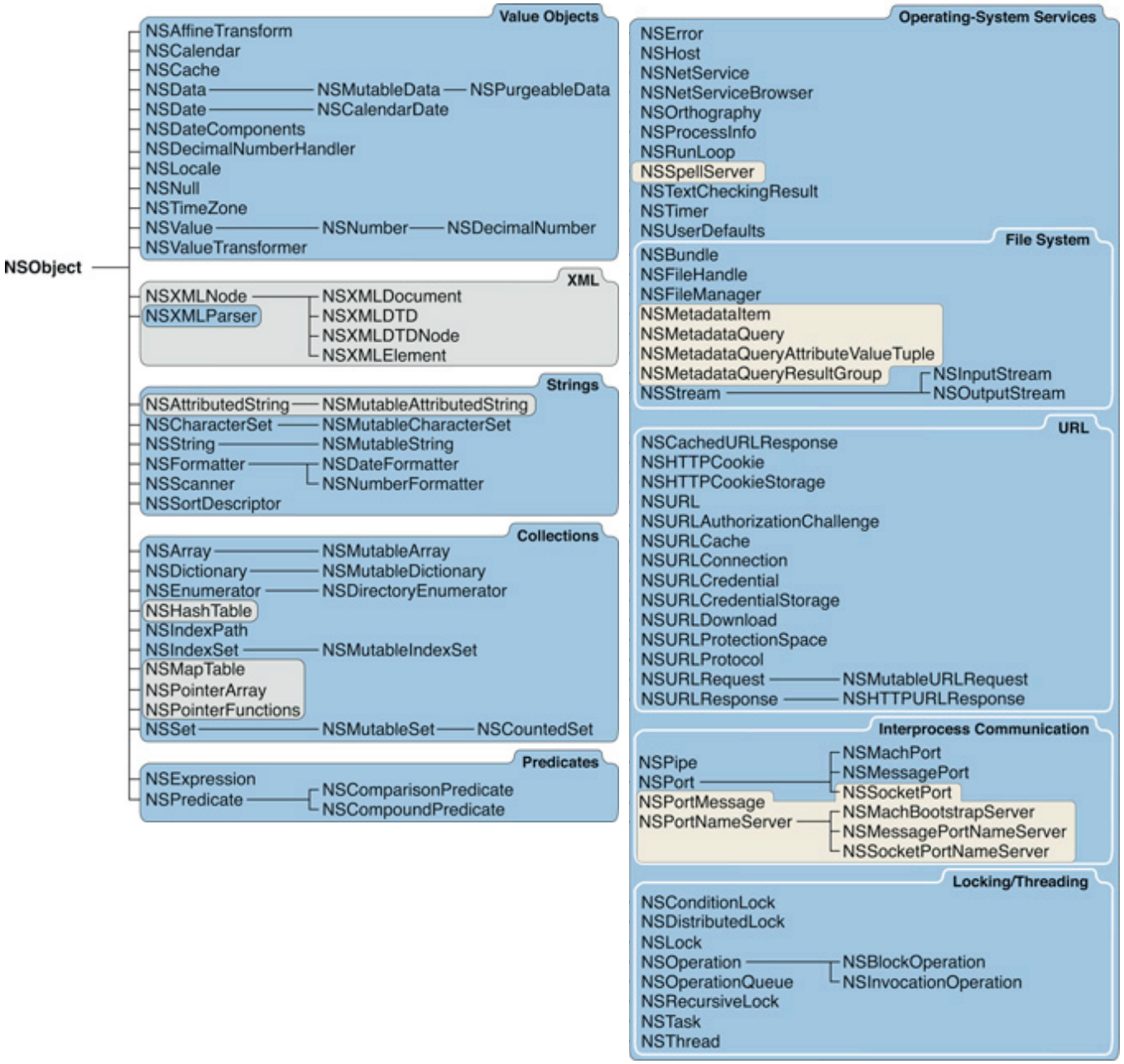


Core OS

- Unlikely to use directly, but used by other layers of iOS stack
- e.g., Security, Bluetooth and System APIs (POSIX / Unix)

Core Services

- "Core" system services for all iOS apps
- Infrastructure: iCloud, In-App Purchase, Newsstand, Social, etc.
- Hardware: Location, Motion, Telephony
- Data structures/management: Core data, **Foundation** framework



(from Apple dev docs)

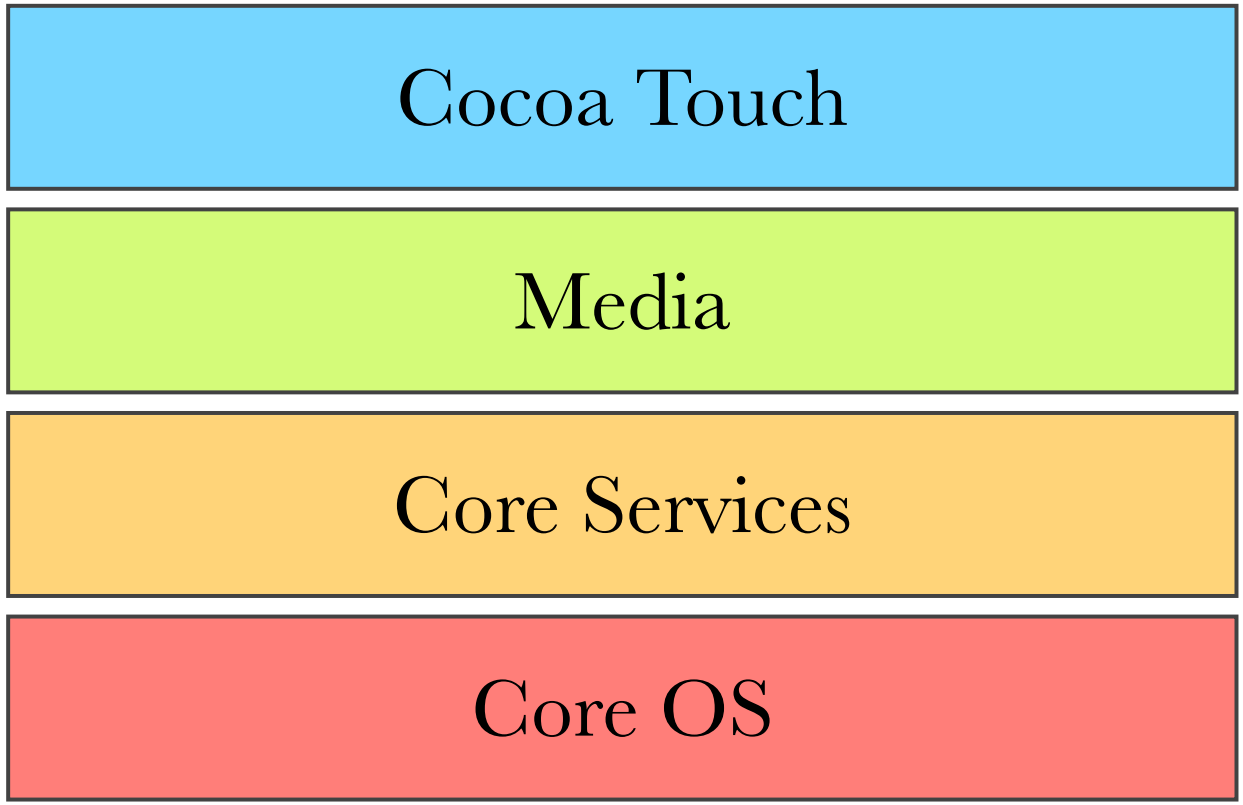
IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Media

- Graphics, Audio, Video APIs
- Core Graphics/Animation/Image/etc.
  - e.g., custom 2D drawing and rendering
- OpenGL ES
  - hardware accelerated 2D/3D graphics

Cocoa Touch

- High level app infrastructure
  - e.g., touch-events, on-screen interface elements, transitions, gestures
- Built-in controllers (e.g., map, photopicker)
- Key framework: **UIKit**


Cocoa Touch
Media
Core Services
Core OS

Typically many ways to accomplish a given task!
(i.e., with frameworks at different levels)

UIKit

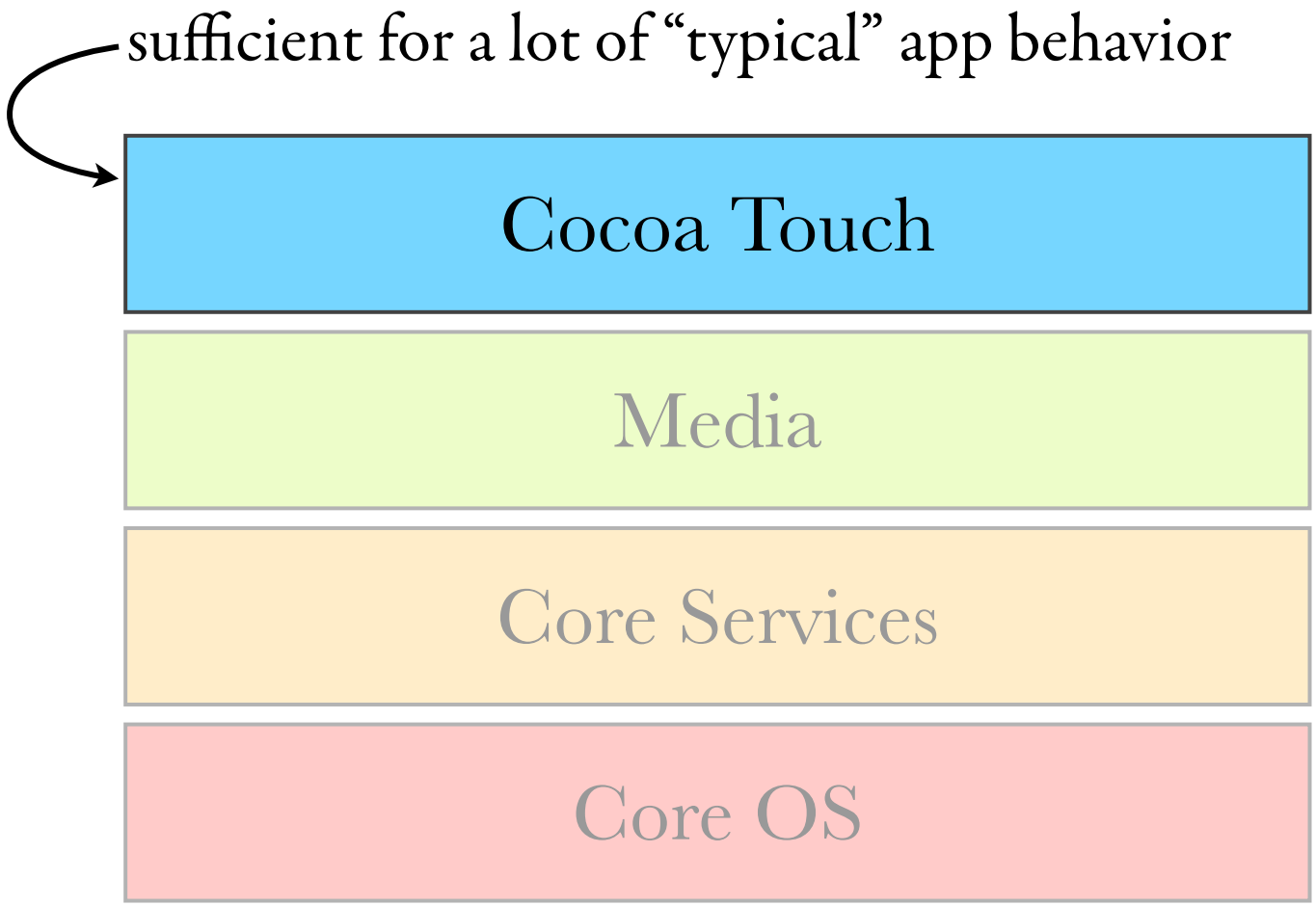
```
// clear with white rectangle
[[UIColor whiteColor] set];
UIRectFill(self.bounds);

// load and draw image at (0,0)
[[UIImage imageNamed:@"image.png"] drawAtPoint:CGPointMake(0, 0)];
```

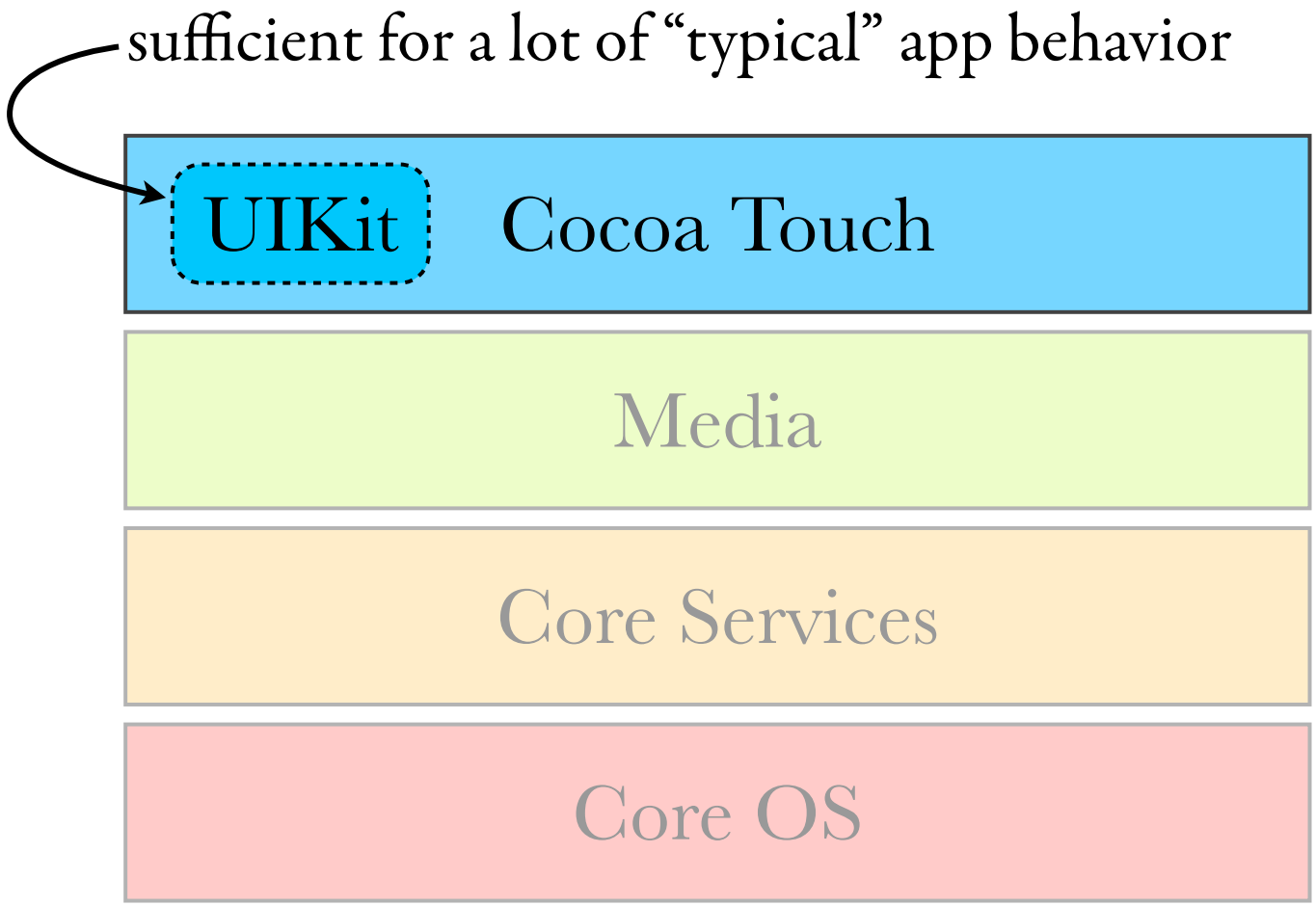
Core Graphics

```
// get current graphics context to draw into
CGContextRef context = UIGraphicsGetCurrentContext();

// clear with white rectangle
CGContextSetRGBFillColor(context, 1.0, 1.0, 1.0, 1.0);
CGContextFillRect(context, self.bounds);

// load image from file
NSString* imageFileName = [[[NSBundle mainBundle] resourcePath]
                           stringByAppendingPathComponent:@"image.png"];
CGDataProviderRef provider = CGDataProviderCreateWithFilename([imageFileName UTF8String]);
CGImageRef image = CGImageCreateWithPNGDataProvider(provider,
                                                    NULL,
                                                    true,
                                                    kCGRenderingIntentDefault);
CGDataProviderRelease(provider);

// draw image at (0,0)
CGContextDrawImage(context,
                   CGRectMake(0, 0, CGImageGetWidth(image), CGImageGetHeight(image)),
                   image);
CGImageRelease(image);
```

General rule of thumb (as with all APIs):

Stay *as high up* as you can. Only use lower levels if *functionality* or *performance* calls for it.

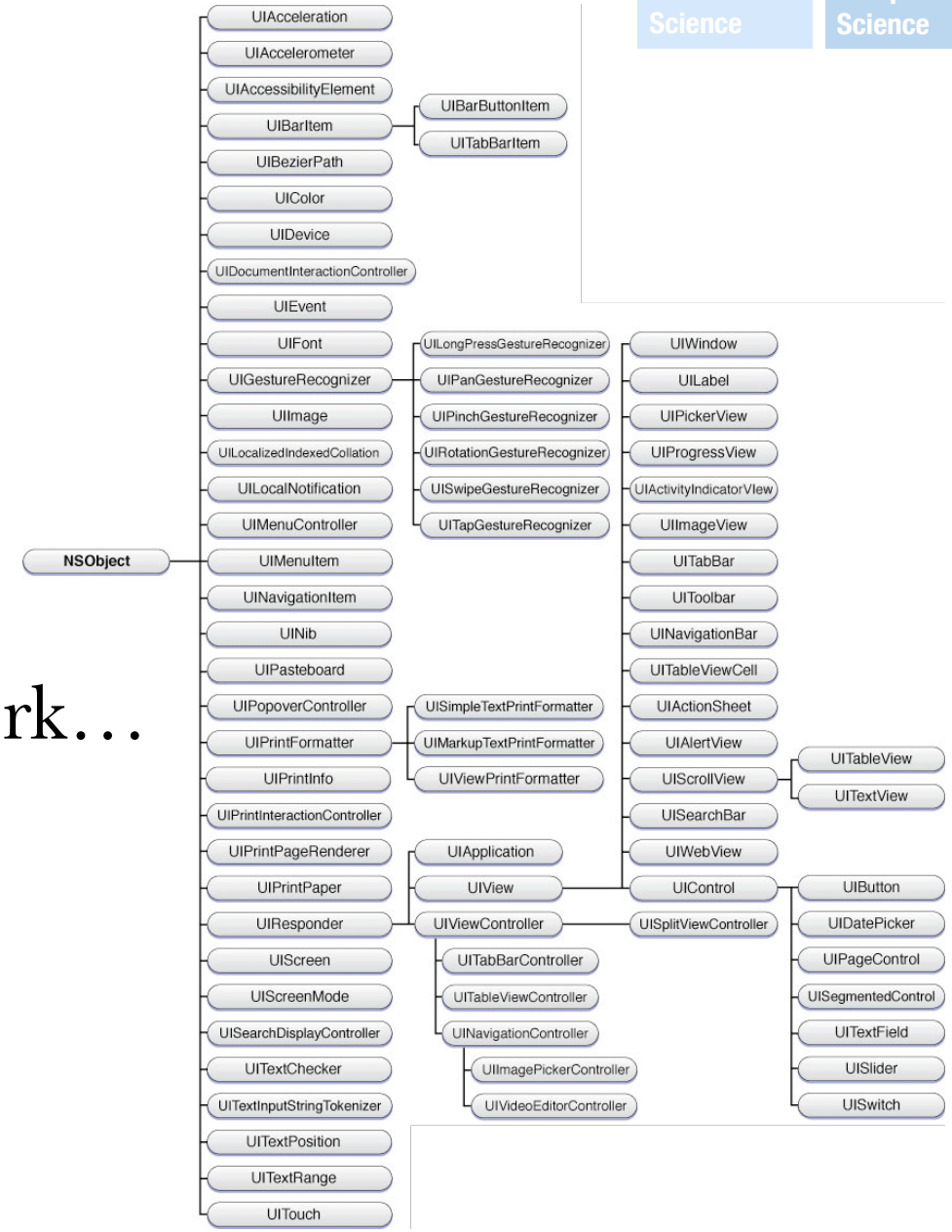

sufficient for a lot of "typical" app behavior
Cocoa Touch
Media
Core Services
Core OS


sufficient for a lot of "typical" app behavior
UIKit
Cocoa Touch
Media
Core Services
Core OS

Bad news (for new devs):

UIKit is a huge framework…

Bad news (for new devs):
UIKit is a huge framework…



IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Bad news (for new devs):

UIKit is a huge framework…

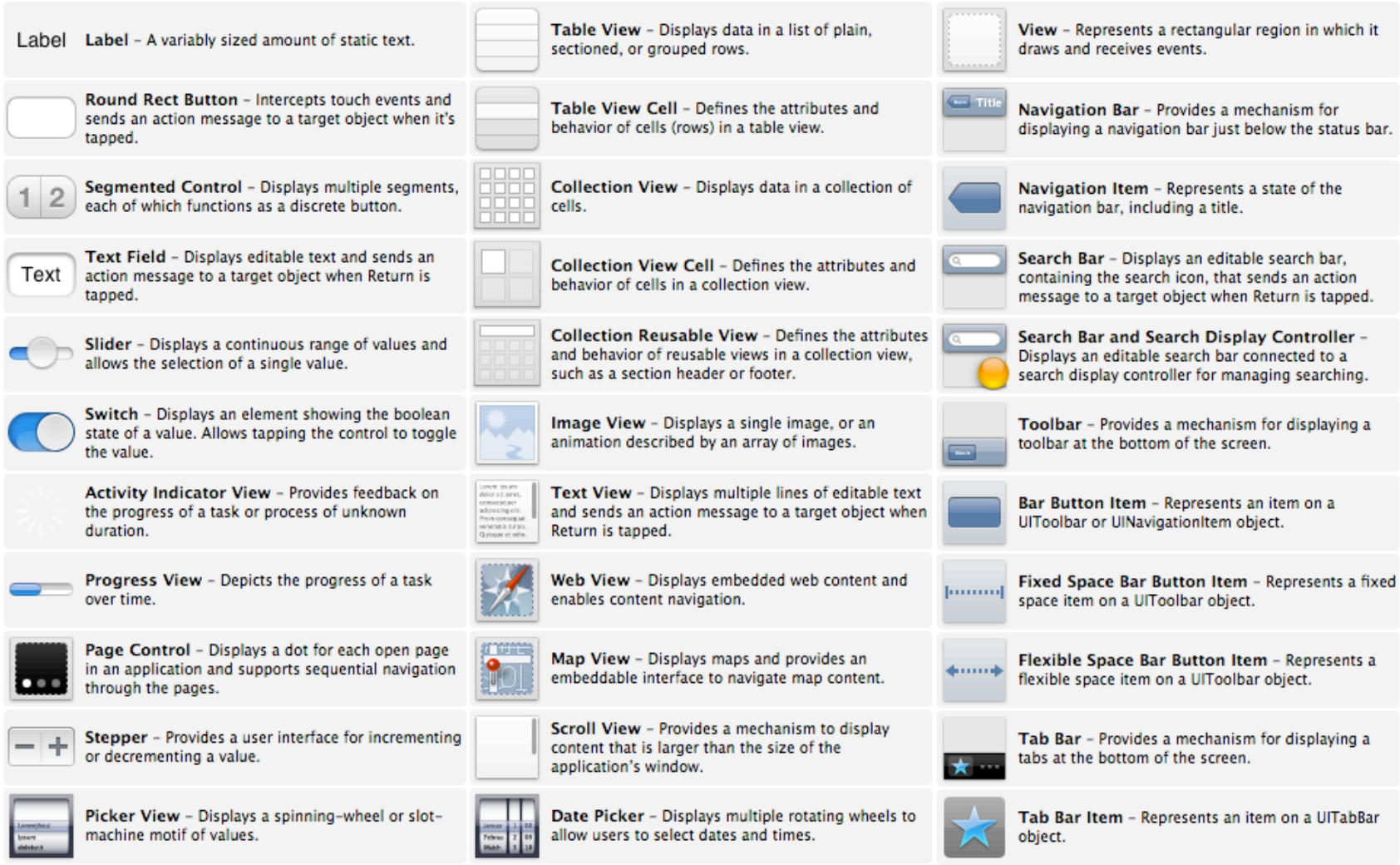
Good news: most UIKit objects can be glued together using a very small bag of tricks

§UIKit + design patterns

First, we distinguish a *visible interface element* from the *invisible code* that manages it

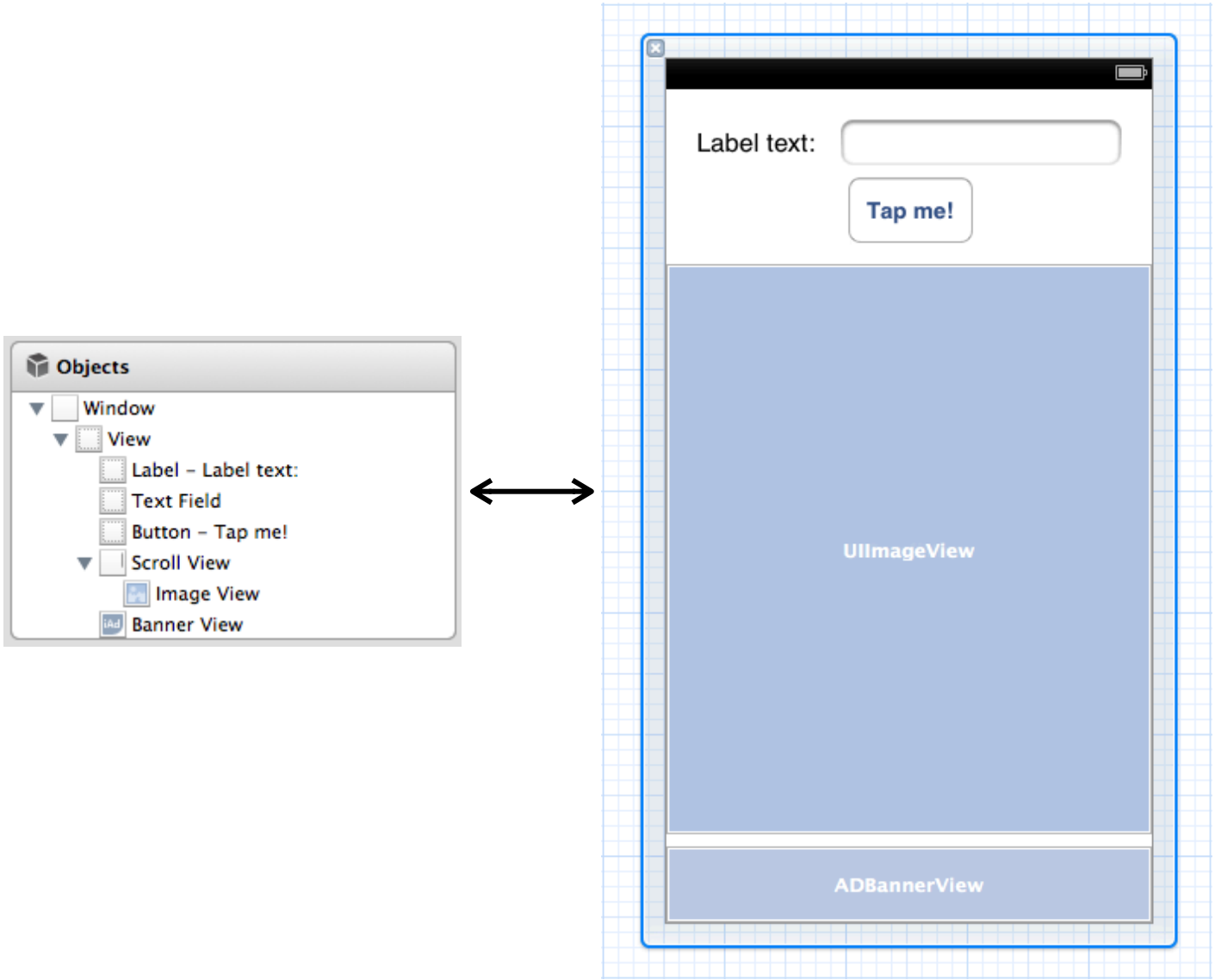
UIKit parlance:

- visible UI elements: *view* objects
  - subclasses of `UIView`
- backing code: *view controller* objects
  - subclasses of `UIViewController`



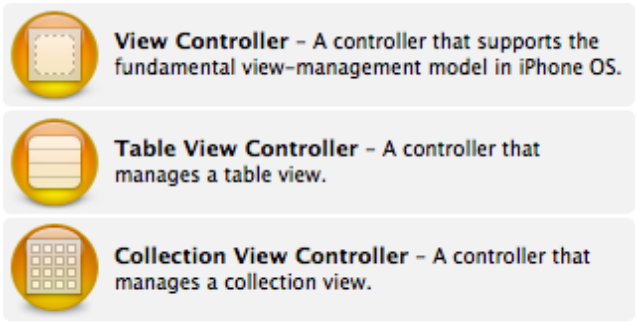
UIKit View Classes (partial)

Recall: a *scene* in an iOS app will consist of a single *root view* and a *hierarchy of subviews*

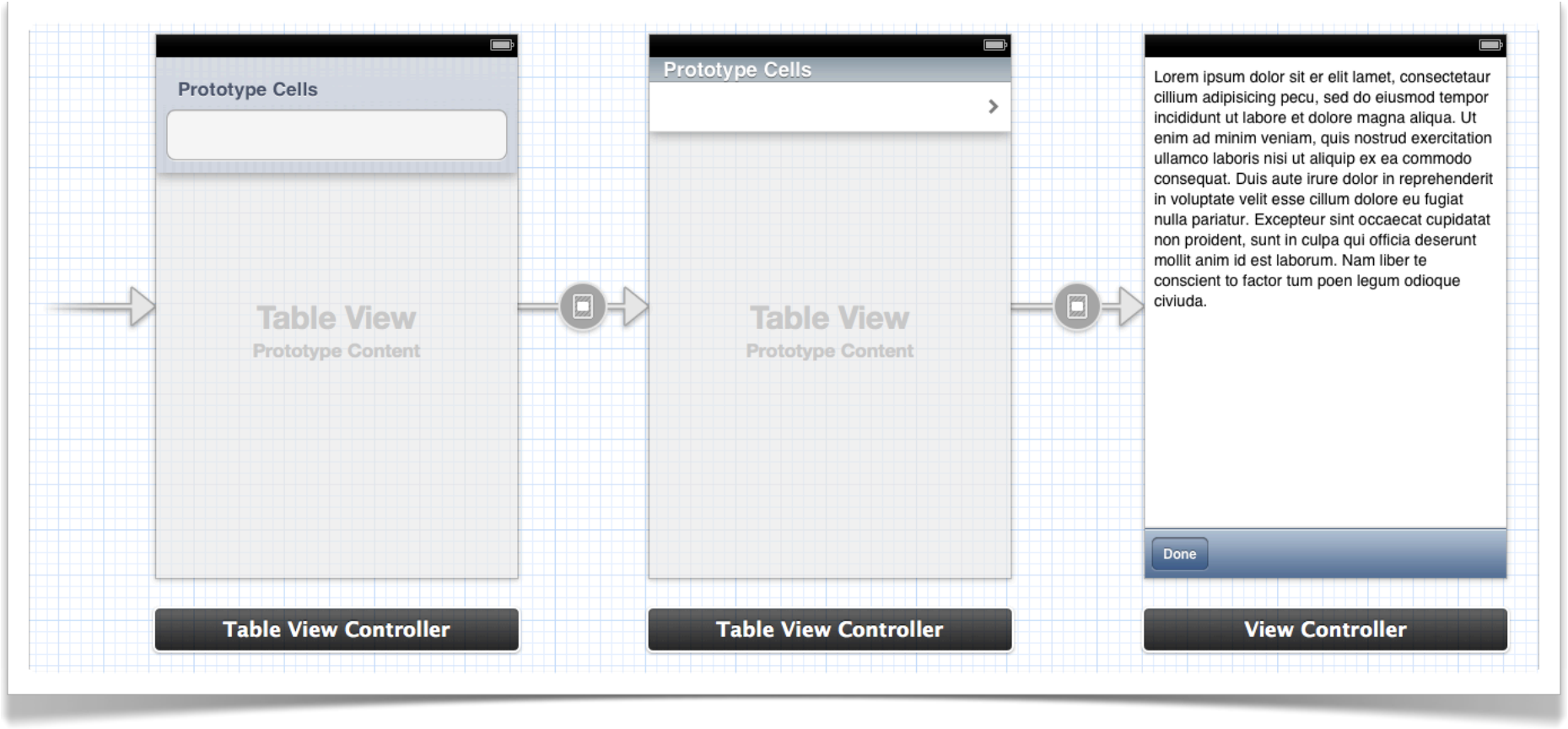

Objects
Window
View
Label – Label text:
Text Field
Button – Tap me!
Scroll View
Image View
Banner View
Label text:
Tap me!
UIImageView
ADBannerView

Each scene (on iPhone) is typically managed by a *single backing view controller*

Each scene (on iPhone) is typically managed by a *single backing view controller*



view controller base classes



view controller per scene

A view controller must know about the state of its associated view (e.g., loaded? visible?)

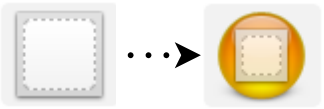
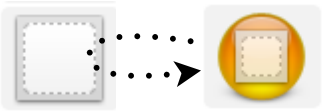
- many *view lifecycle notification* messages
- note: a view controller can still respond to messages if its views are not visible!

Hands-on

- Project: *ControllerDemo*
- Associating Storyboard scenes with custom view controller classes
- View lifecycle notification

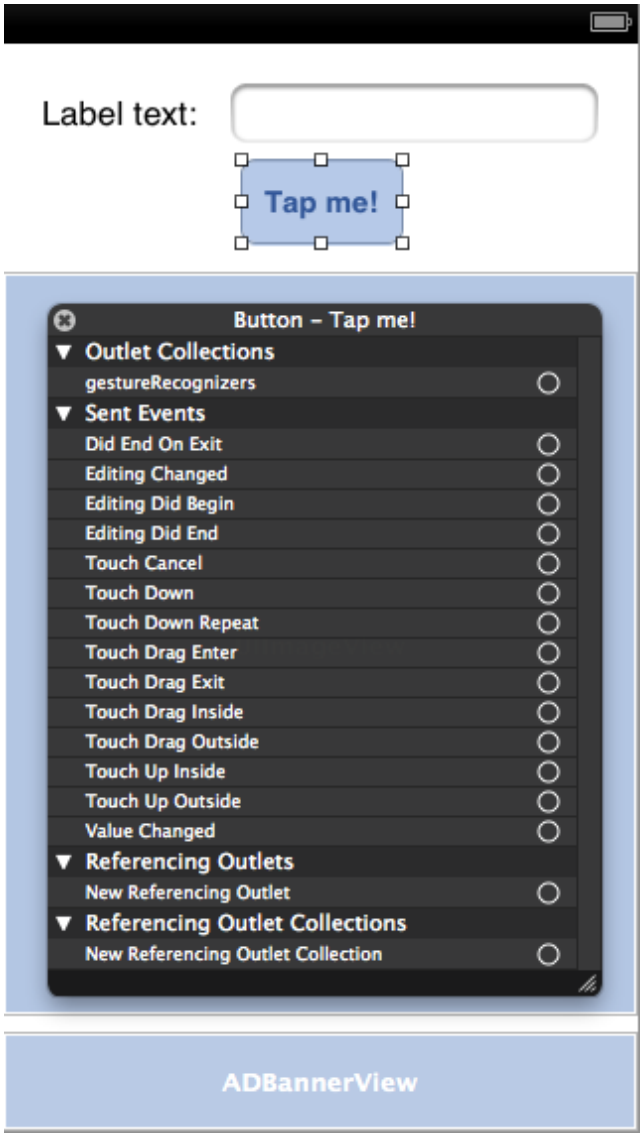
View lifecycle events are useful, but controllers need to interact with views in other ways!

Some basic use cases:

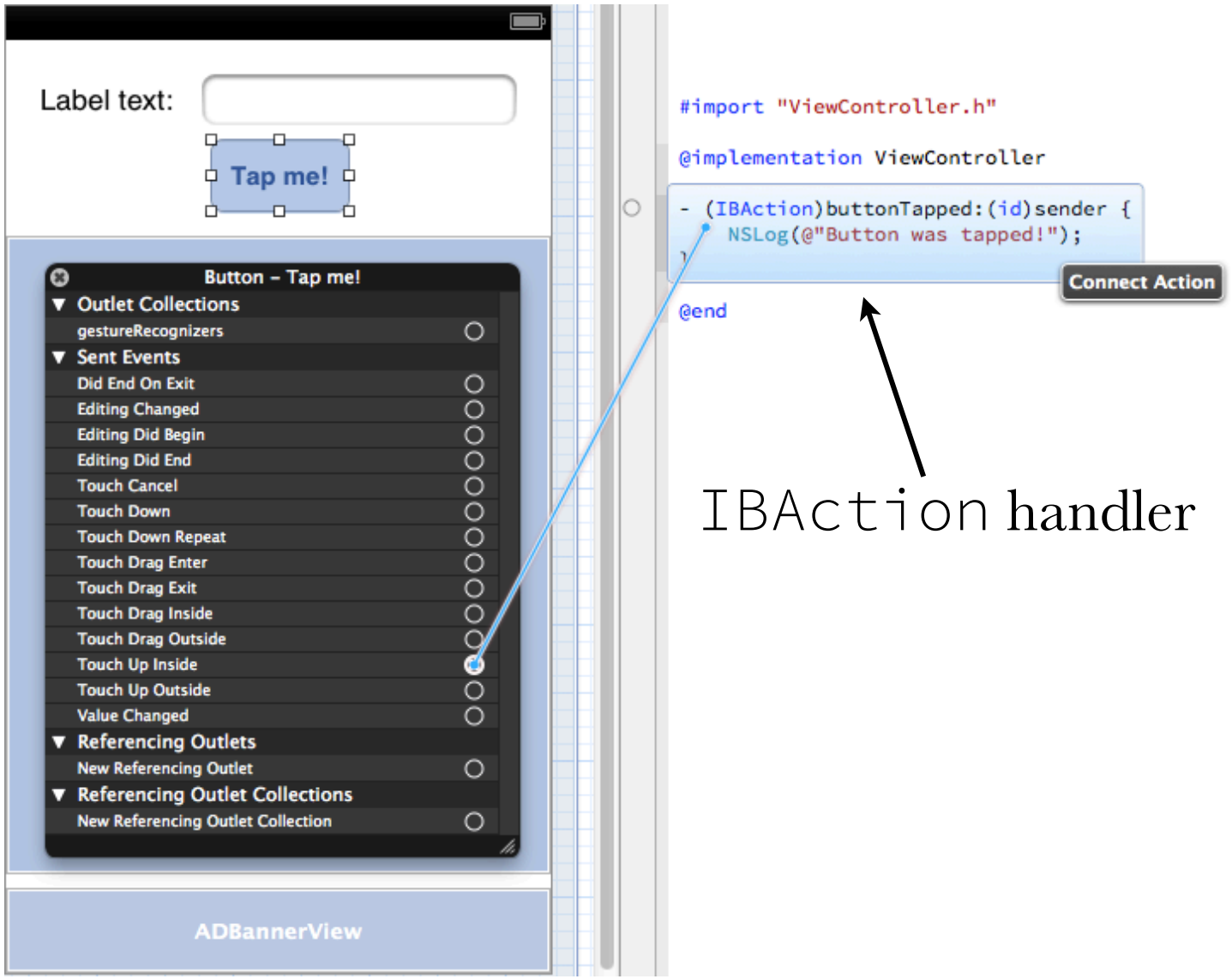
1. *Pushing* events from view to controller 
2. *Pushing* data from controller to view 
3. *Pulling* data from view to controller 

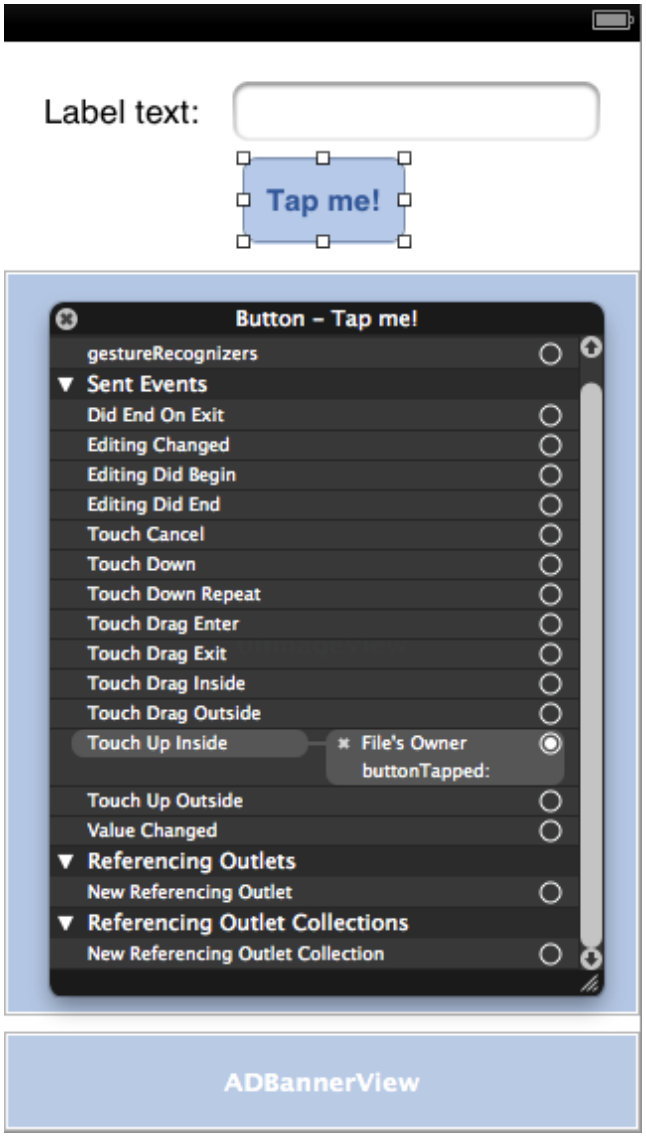
UIControl views have well defined events they can generate (e.g., in response to touches)

e.g., buttons, sliders, switches



UIButton actions

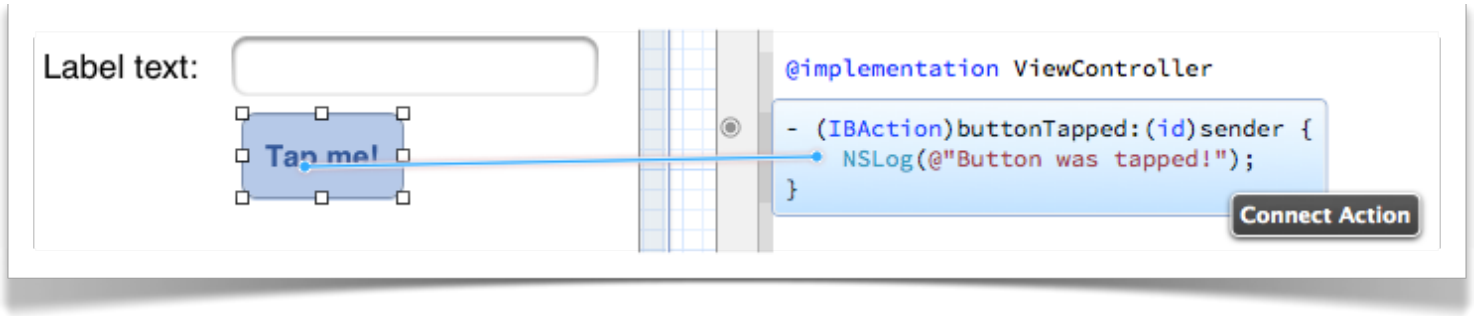

Label text:
Tap me!
Button – Tap me!
Outlet Collections
gestureRecognizers
Sent Events
Did End On Exit
Editing Changed
Editing Did Begin
Editing Did End
Touch Cancel
Touch Down
Touch Down Repeat
Touch Drag Enter
Touch Drag Exit
Touch Drag Inside
Touch Drag Outside
Touch Up Inside
Touch Up Outside
Value Changed
Referencing Outlets
New Referencing Outlet
Referencing Outlet Collections
New Referencing Outlet Collection
ADBannerView
#import "ViewController.h"
@implementation ViewController
- (IBAction)buttonTapped:(id)sender {
NSLog(@"Button was tapped!");
}
Connect Action
@end
IBAction handler



Connection established

Target-Action Pattern

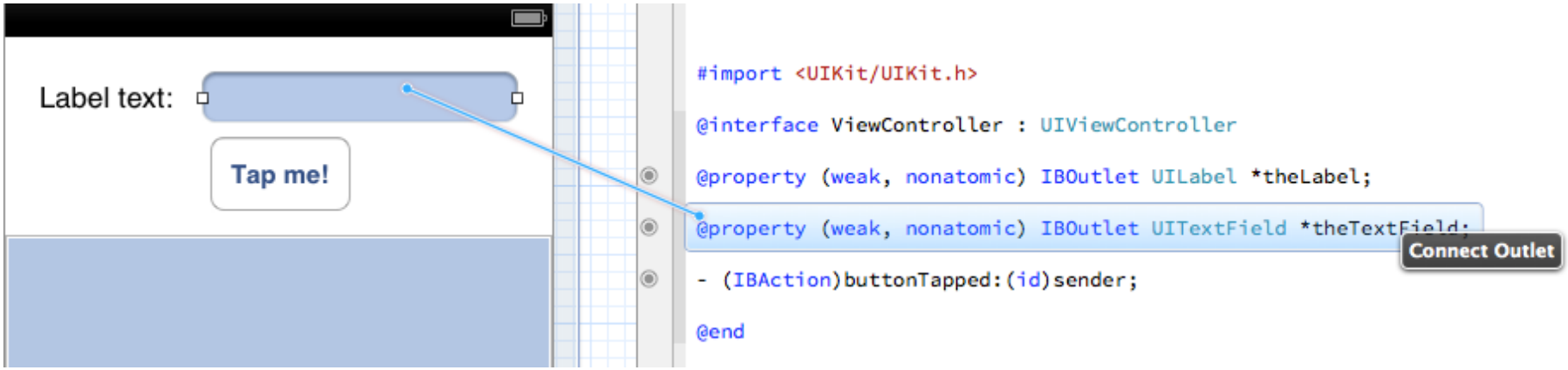
- Key architecture for associating UI events with coded IBAction handlers



Views also have *properties* (read-only or writeable)

E.g., all views: color, size, position, etc.;
text fields: entered text;
"pickers": selected row/column

Views can be connected to controllers as *outlets*



Intuitively: outlets function as devices for controllers to interact with the outside world

```
/* inside the view controller */

self.someOutlet.writeableProperty = ...;
```

```
... = self.someOutlet.propertyName;
```

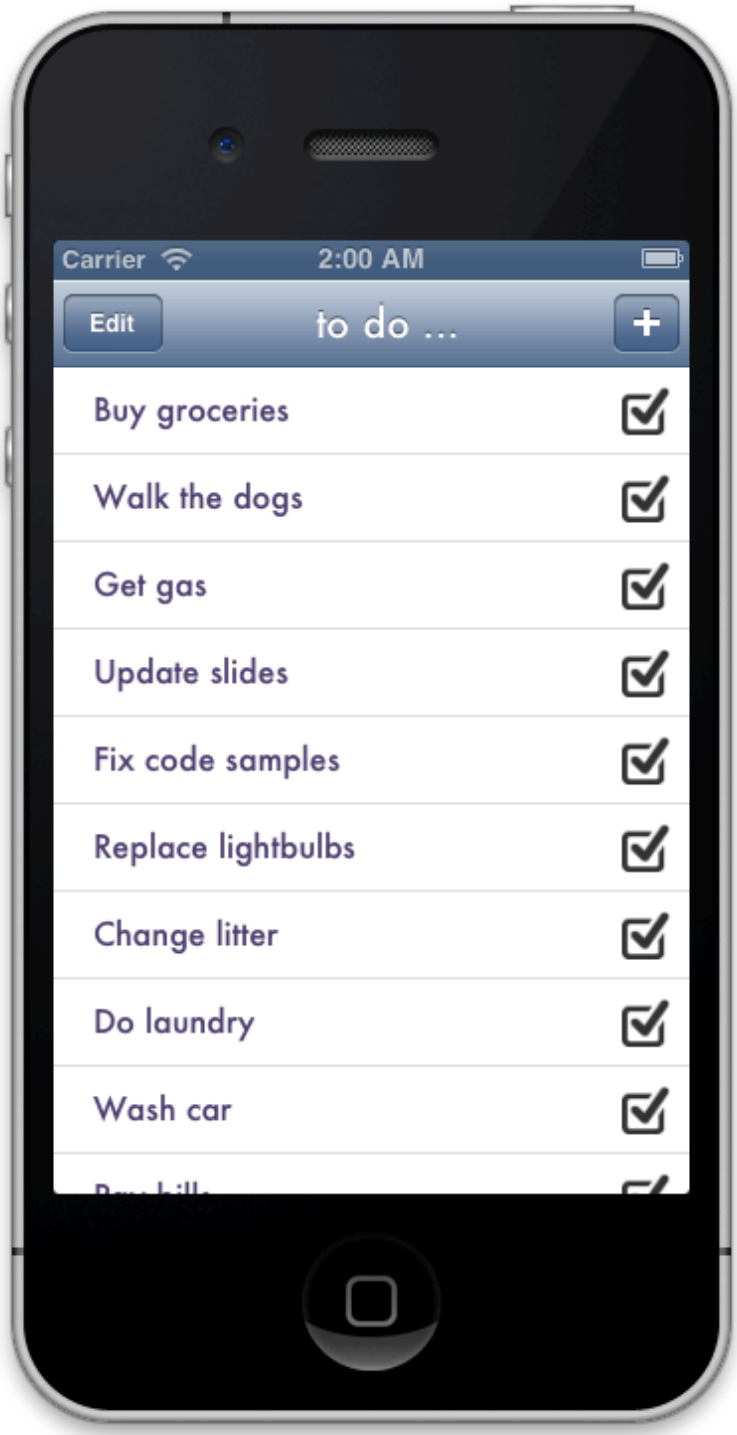
```
- (IBAction)buttonTapped:(id)sender {
    self.theLabel.text = [NSString stringWithFormat:
                          @"%@:",
                          self.theTextField.text];
}
```

Hands-on

- Project: *LabelToFile*
- Hooking up actions & outlets
- Simulator app support directory

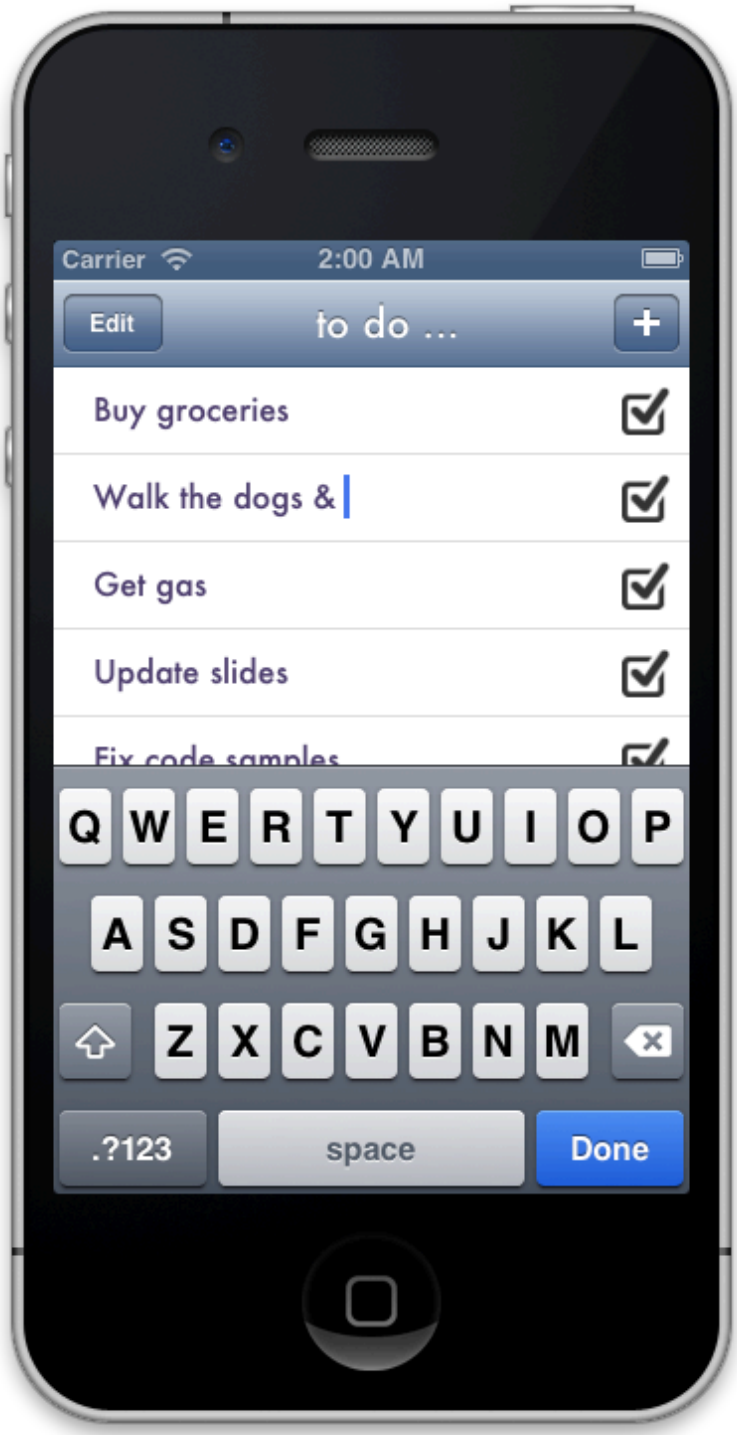
target-action and outlets work well when UI/controller interaction is fairly simple; i.e.,

- # of events controller cares about is limited
- controller always decides when to refresh UI (i.e., controller pushes changes to view)

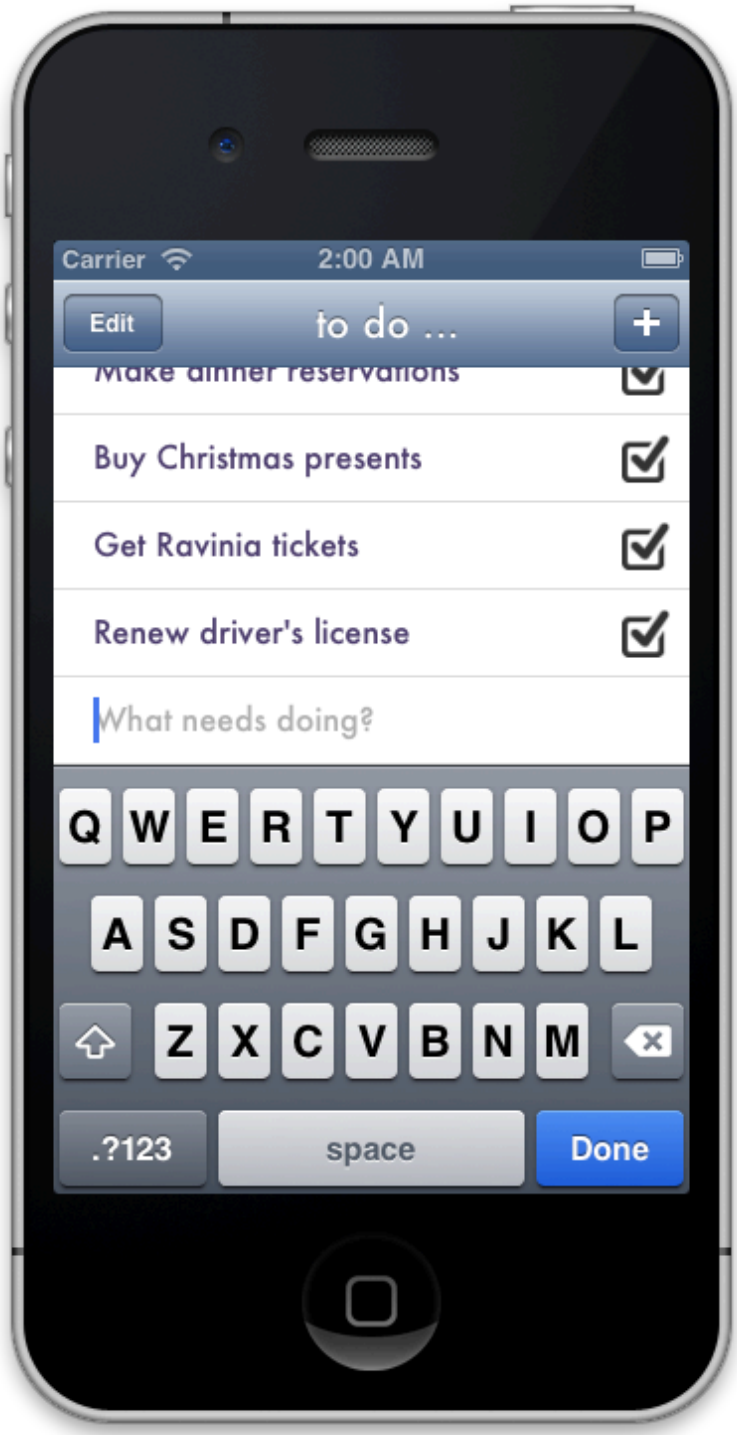


Consider a to-do list app.

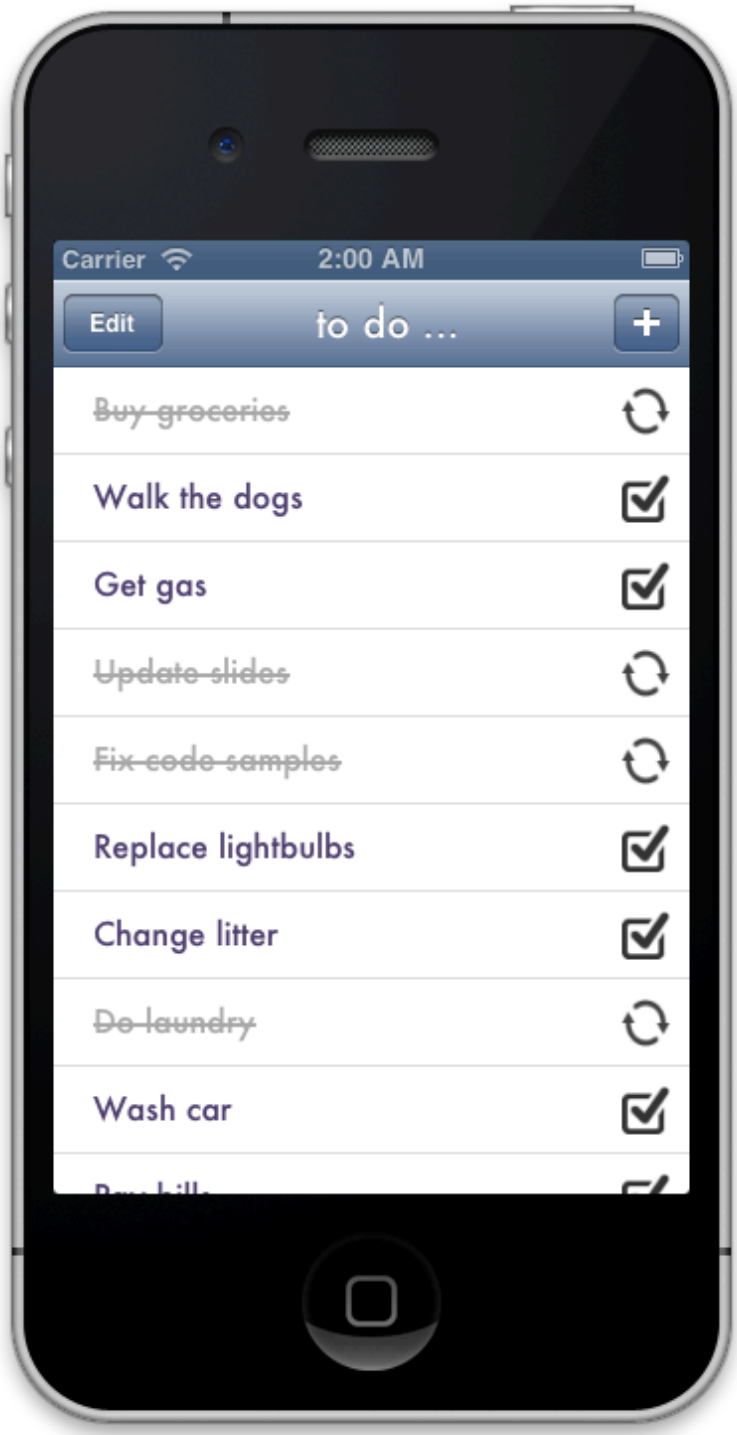
Lots of potentially interesting events.



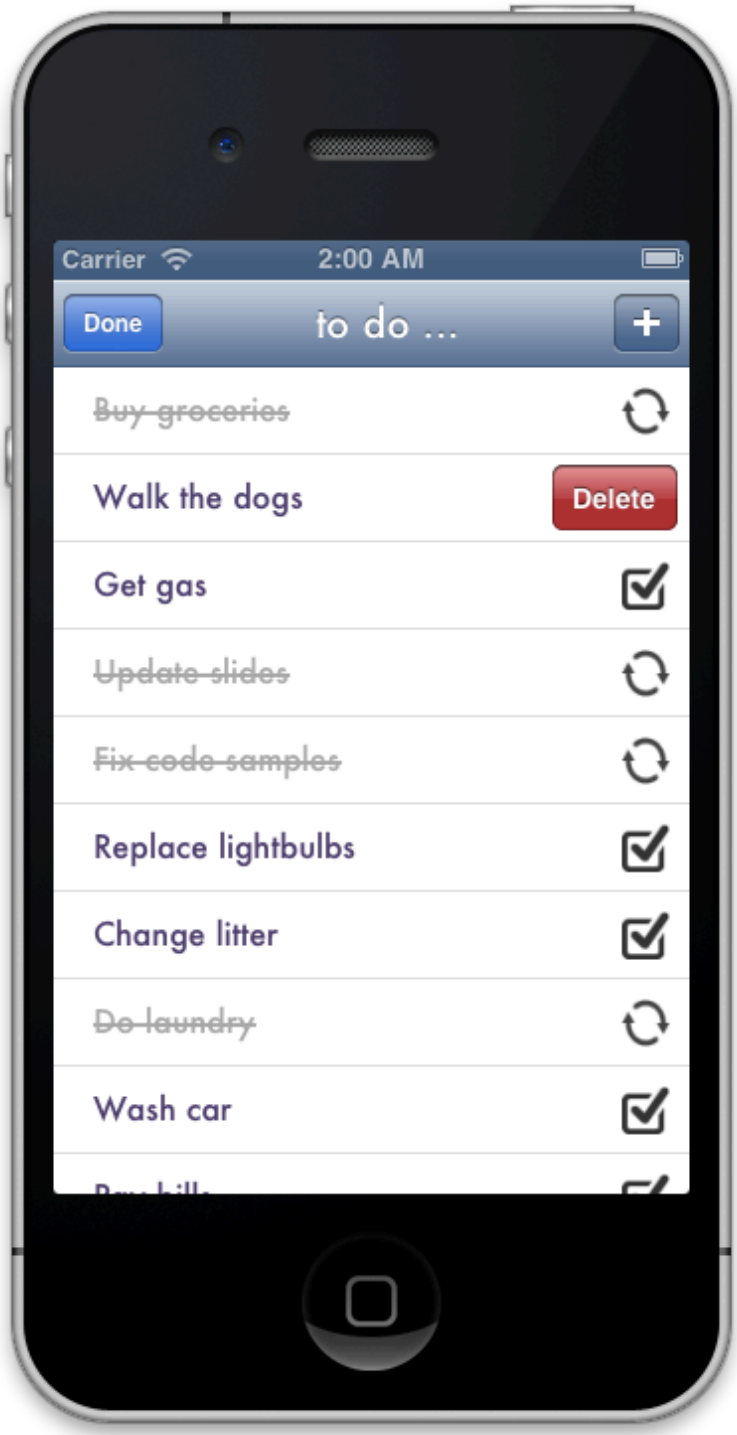
e.g., tapping on a cell
(to start editing)



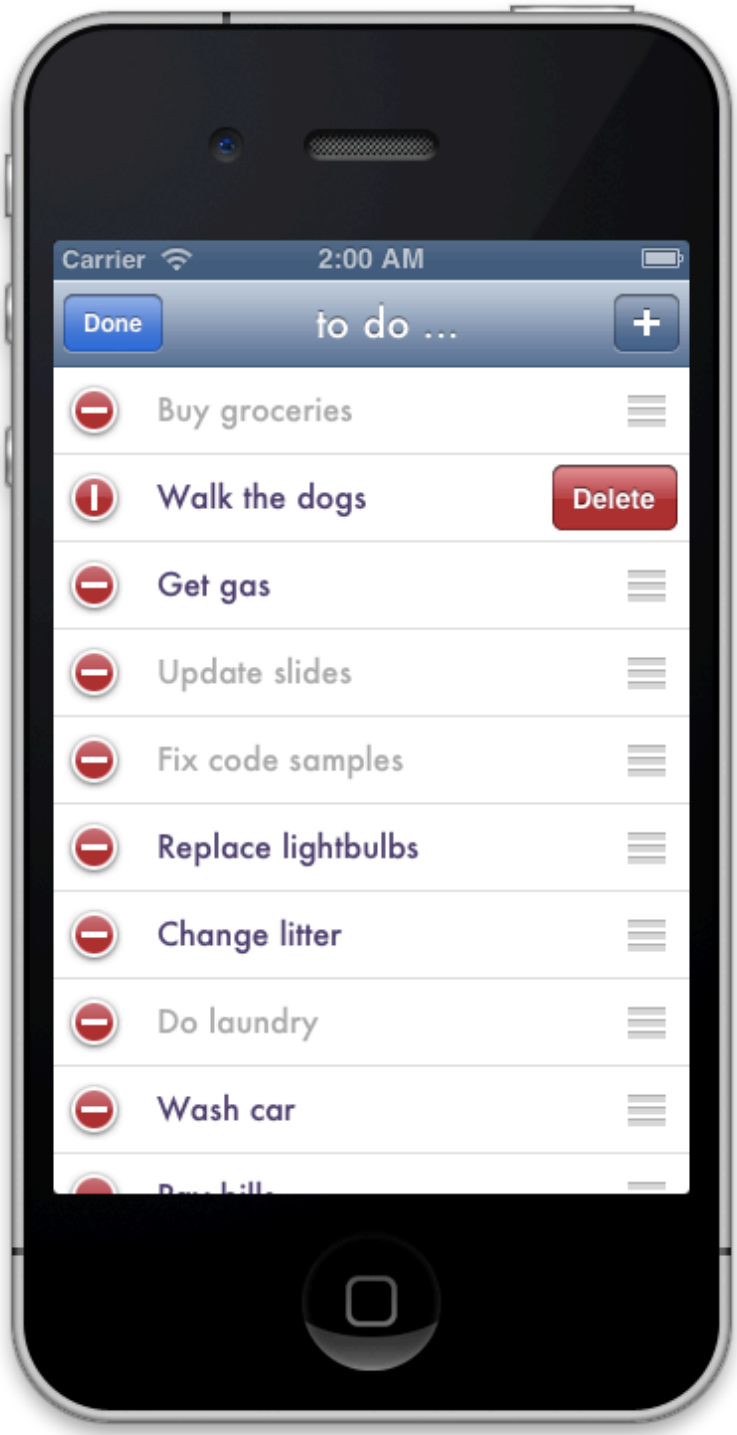
e.g., inserting a new cell



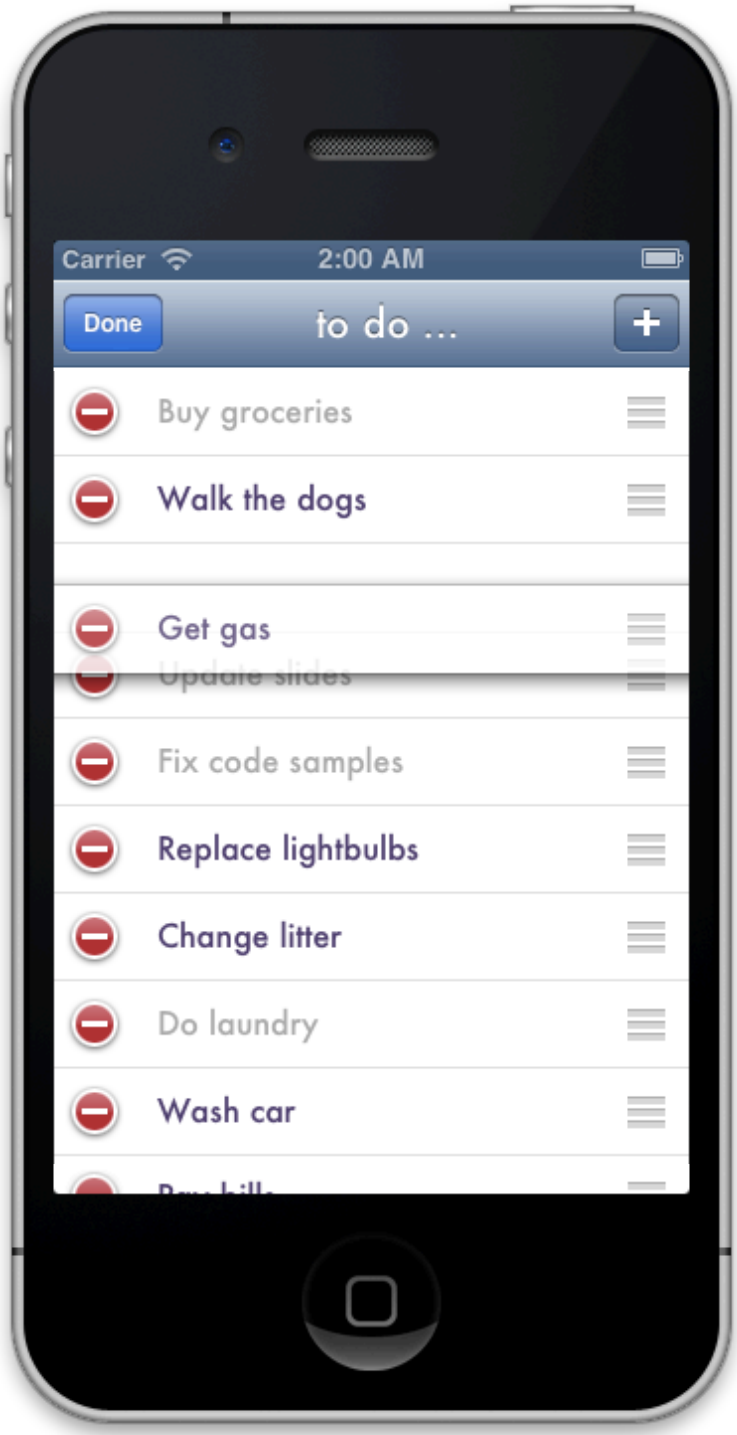
e.g., taps on "accessory" views (to complete)



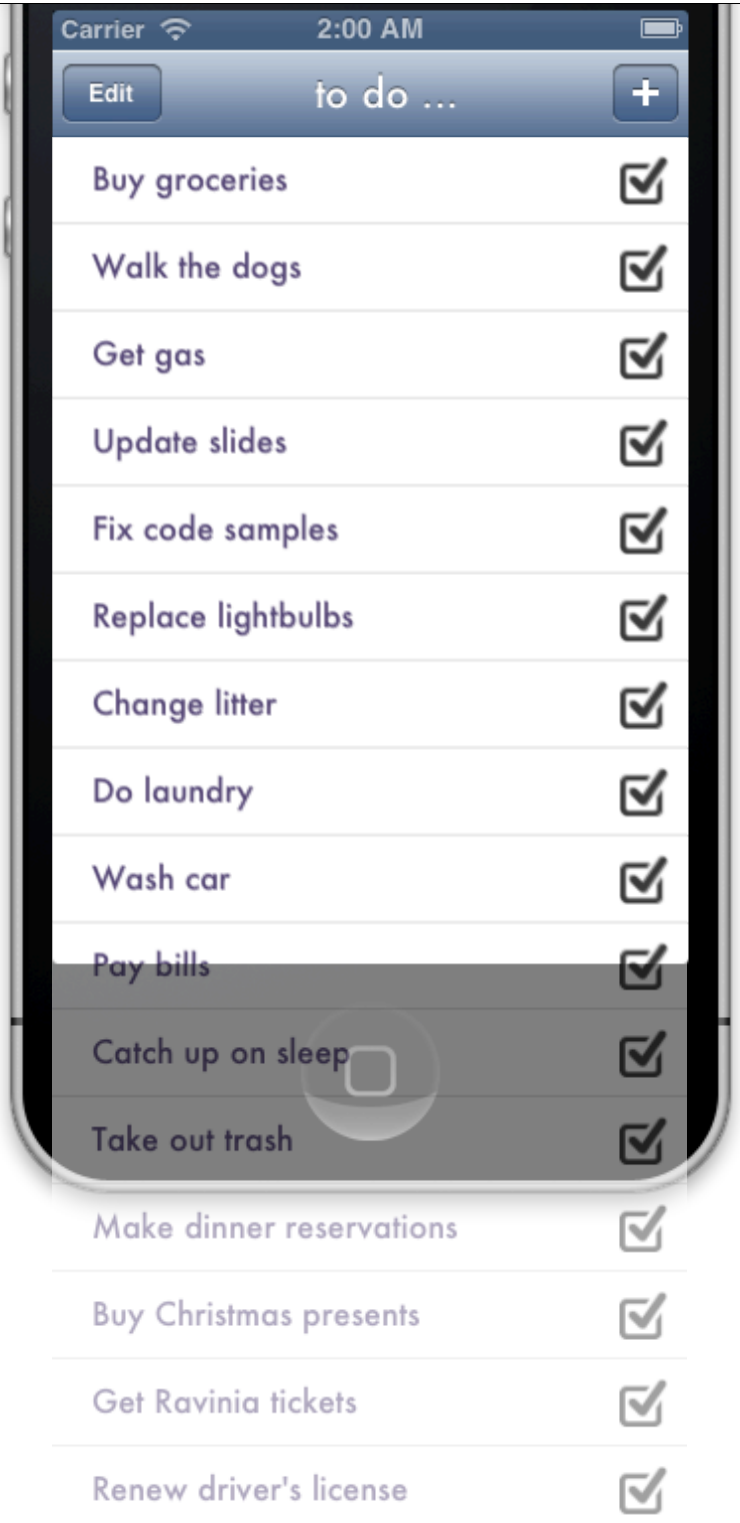
e.g., initiate delete (with a swipe gesture)



e.g., enter editing mode

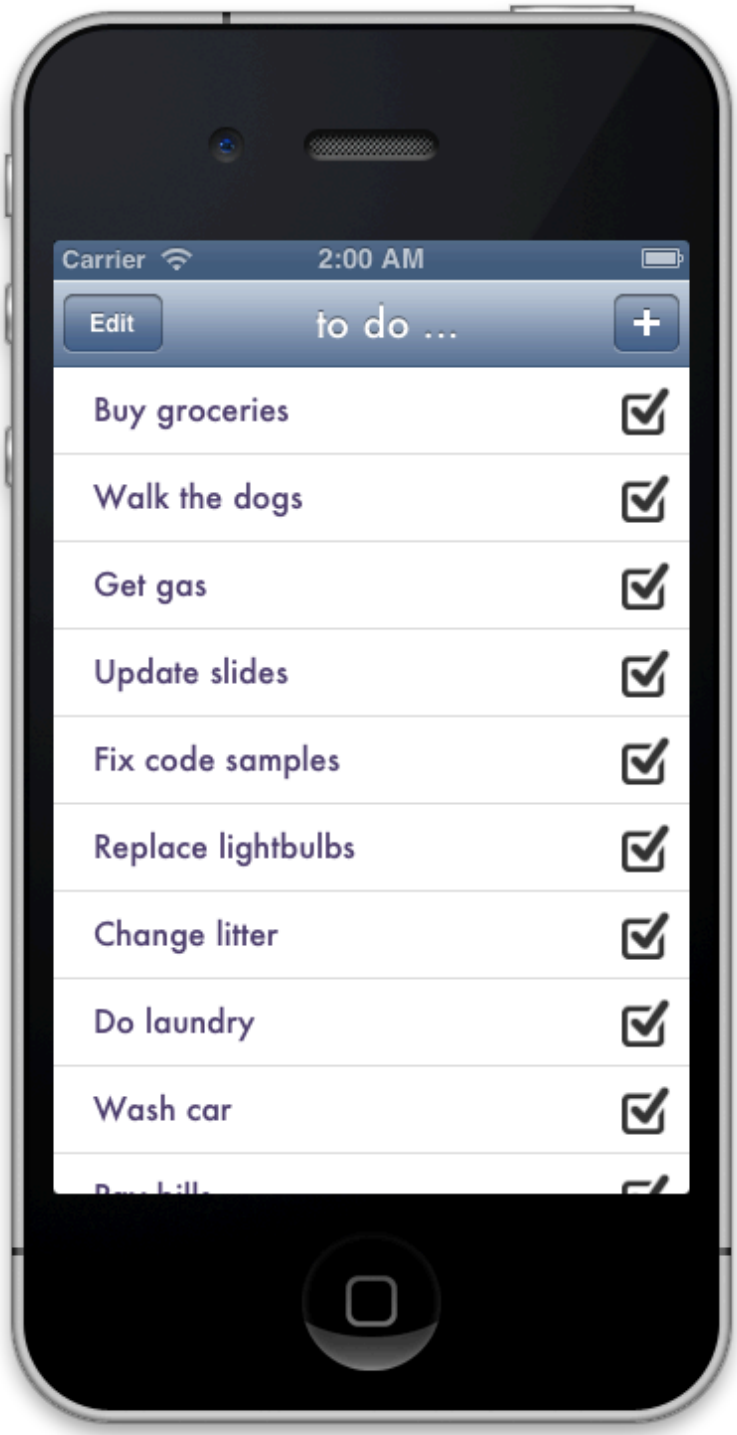


e.g., moving cells
(by dragging)



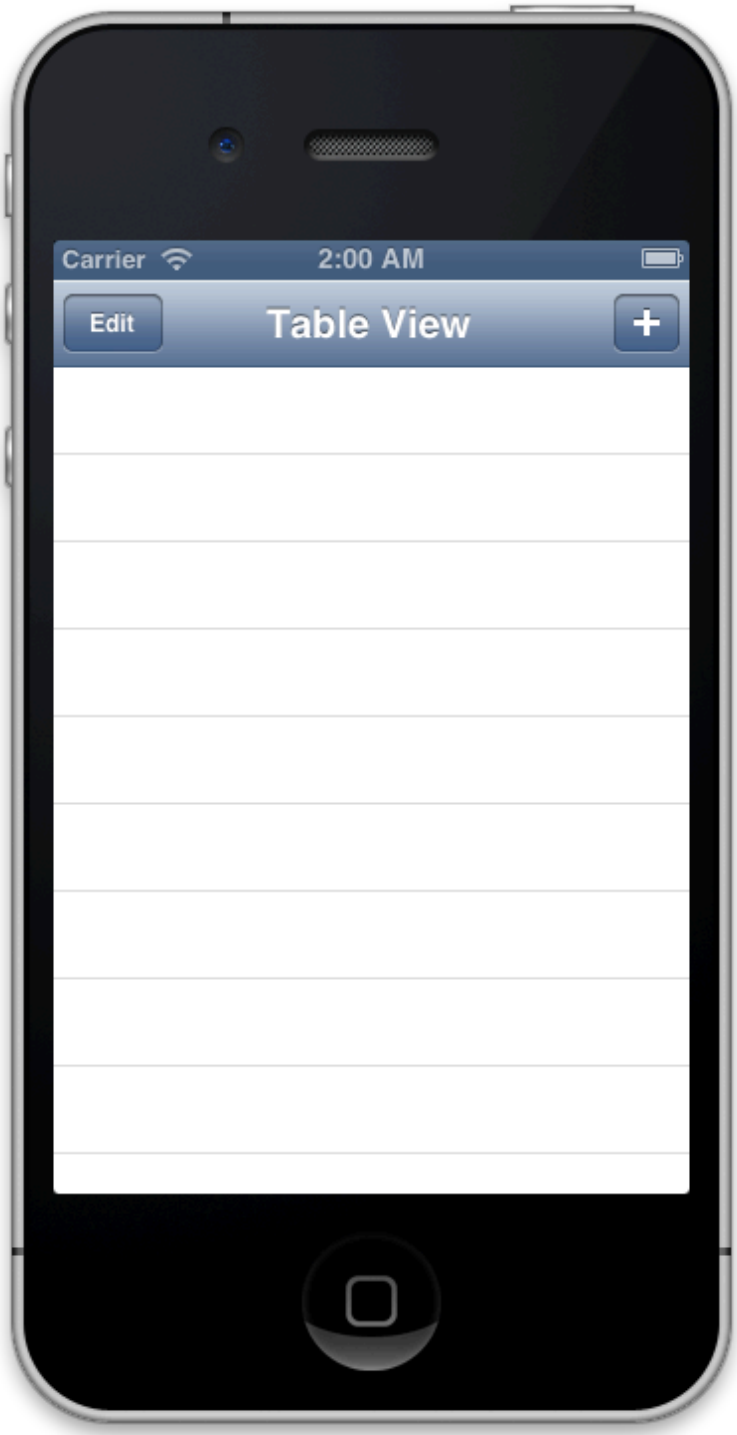
Potentially many more cells than are on screen

- view must be able to quickly *pull in* new data

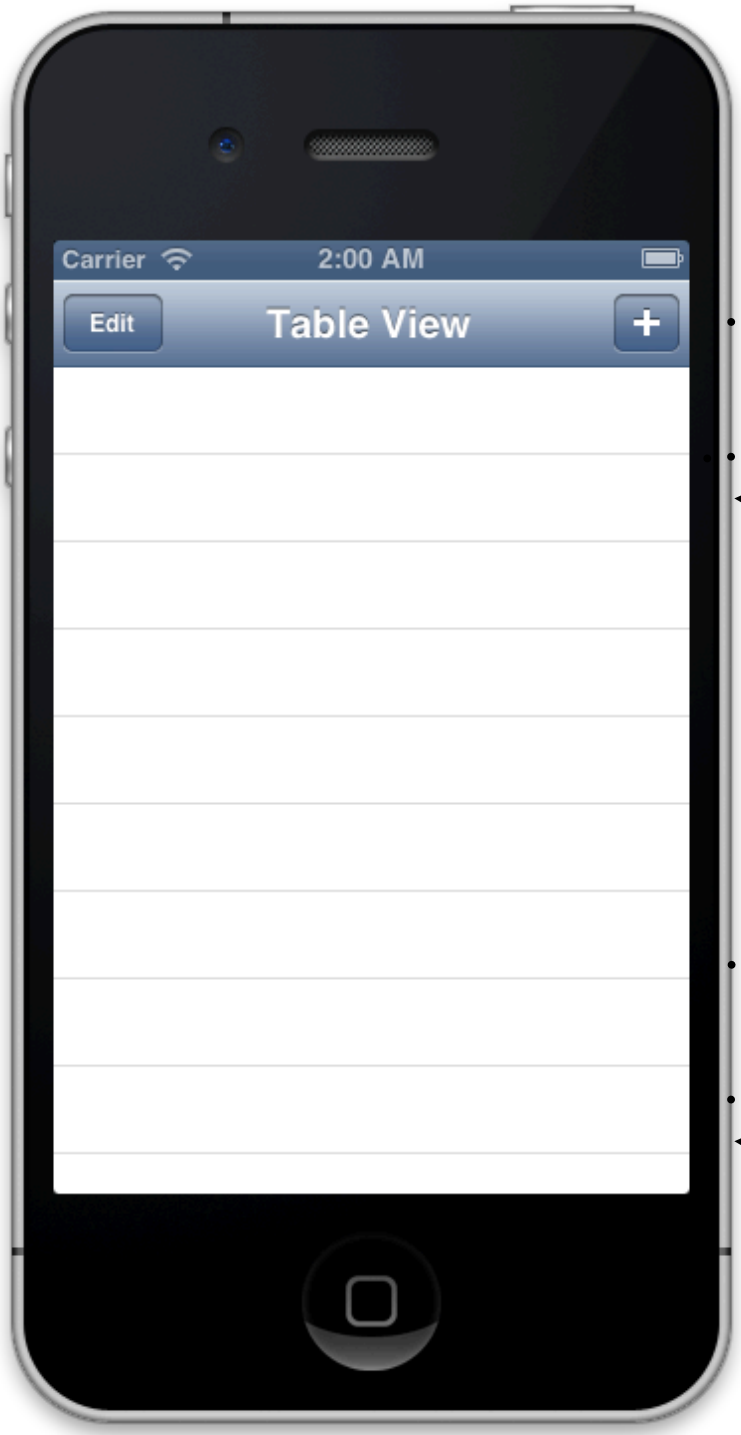


Would be annoying to define *actions* and *outlets* in our code for every event/item!

And how to let the view *pull data* from controller?



New approach: table views define *protocols* for objects that wish to serve as its *delegate* & *data source*



selections,
mode changes

delegate

view configuration
information

insertions, edits,
deletions, moves

data source

rows, # sections,
titles, editability

Delegation Pattern

- Allows complex view reuse by standardizing the protocols they expect *delegates* to adopt
- Typically, the view controller adopts both delegate and datasource protocols

```
@protocol UITableViewDelegate
- (CGFloat)tableView:(UITableView *)tableView
          heightForRowAtIndexPath:(NSIndexPath *)indexPath;
- (void)tableView:(UITableView *)tableView
        accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath;
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath;
@end
```

```
@protocol UITableViewDataSource
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;
- (NSInteger)tableView:(UITableView *)tableView
            numberOfRowsInSection:(NSInteger)section;
- (UITableViewCell *)tableView:(UITableView *)tableView
                    cellForRowAtIndexPath:(NSIndexPath *)indexPath;
- (void)tableView:(UITableView *)tableView
        moveRowAtIndexPath:(NSIndexPath *)sourceIndexPath
        toIndexPath:(NSIndexPath *)destinationIndexPath;
@end
```

```
@interface ListViewController : UIViewController <UITableViewDelegate,
                                                  UITableViewDataSource>
@end
```

```
@implementation ListViewController
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
            numberOfRowsInSection:(NSInteger)section {
    return _data.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"
                                                            forIndexPath:indexPath];
    cell.textLabel.text = _data[indexPath.row];
    return cell;
}
...
@end
```


```
_data = [NSMutableArray arrayWithObjects:
         @"good morning", @"早安", @"おはようございます", @"อรุณสวัสดิ์", @"नमस्ते", nil];
```


```
_data = [NSMutableArray arrayWithObjects:
         @"good morning", @"早安", @"おはようございます", @"อรุณสวัสดิ์", @"नमस्ते", nil];
```

```
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    _data[indexPath.row] = [self reverseString:_data[indexPath.row]];
    [tableView reloadRowsAtIndexPaths:@[indexPath]
                     withRowAnimation:UITableViewRowAnimationAutomatic];
}
```


```
_data = [NSMutableArray arrayWithObjects:
         @"good morning", @"早安", @"おはようございます", @"อรุณสวัสดิ์", @"नमस्ते", nil];
```

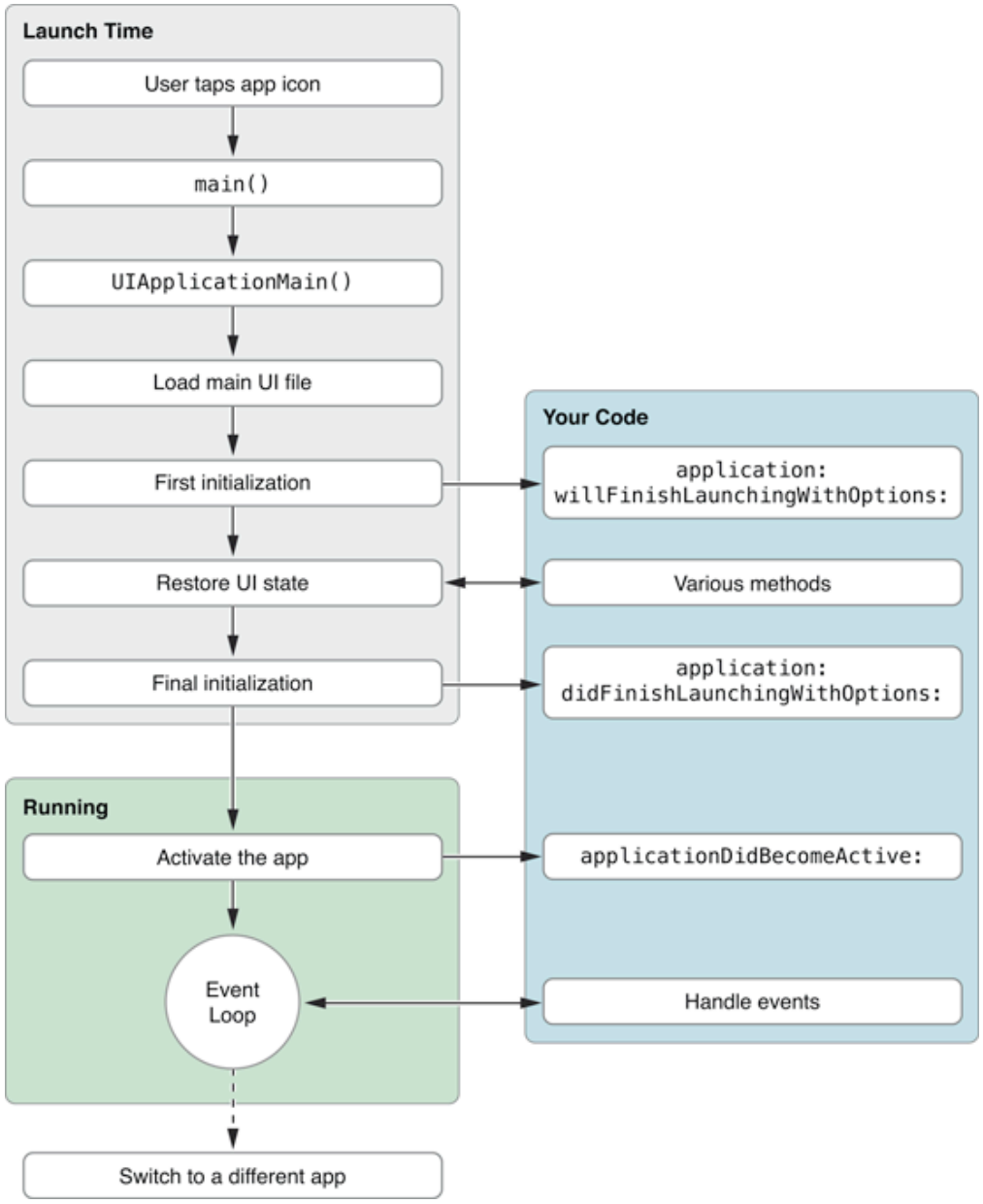
```
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    _data[indexPath.row] = [self reverseString:_data[indexPath.row]];
    [tableView reloadRowsAtIndexPaths:@[indexPath]
                     withRowAnimation:UITableViewRowAnimationAutomatic];
}
```

Hands-on

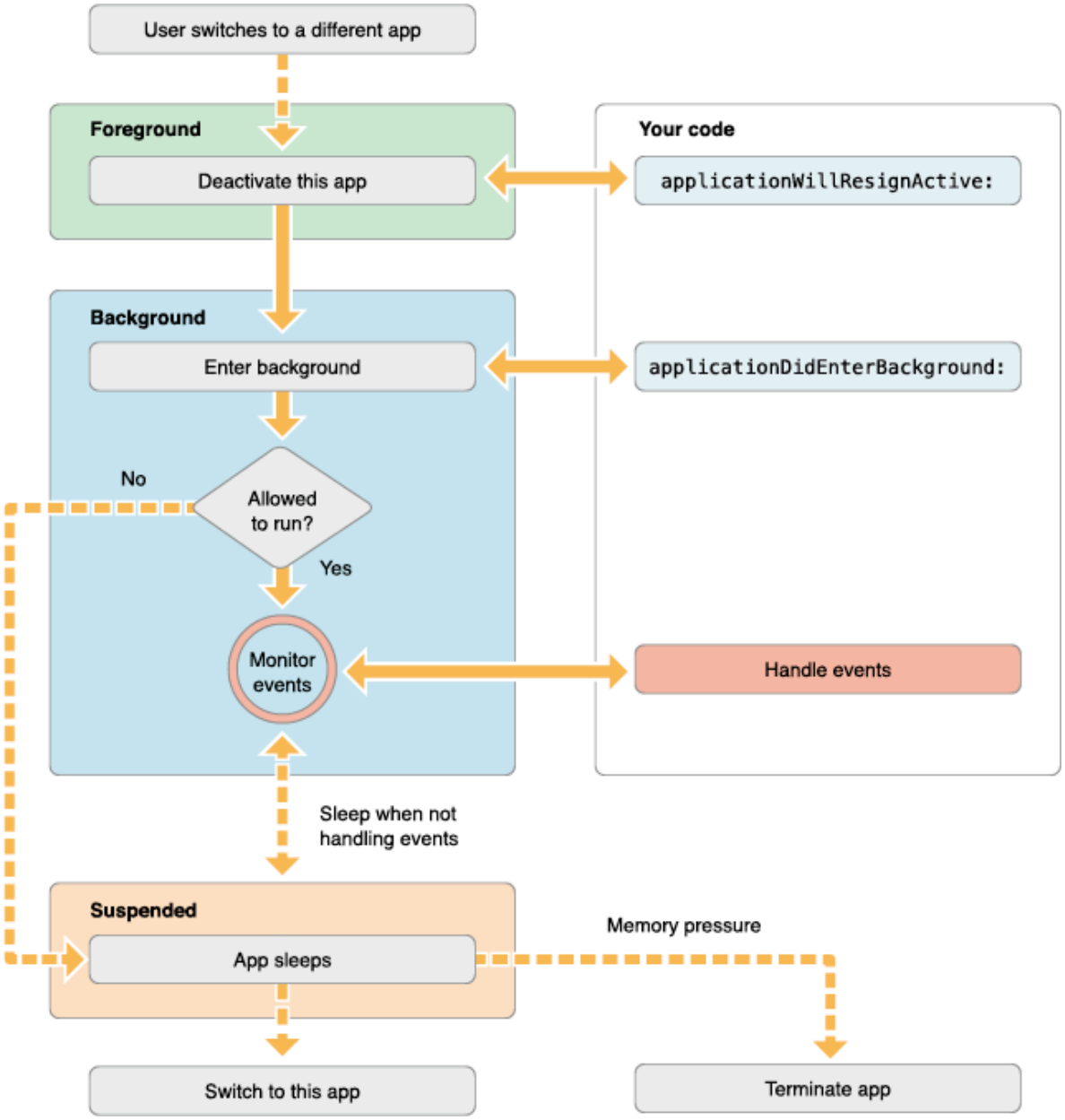
- Projects: *ReversibleGreetings, FontSampler*
- Delegate & Data source protools

AppDelegate

- Delegation isn't just used for managing views
- It is the mechanism for relaying *app lifecycle notifications* to our custom code
  - UIApplication → UIApplicationDelegate
  - Callbacks akin to view event notifications



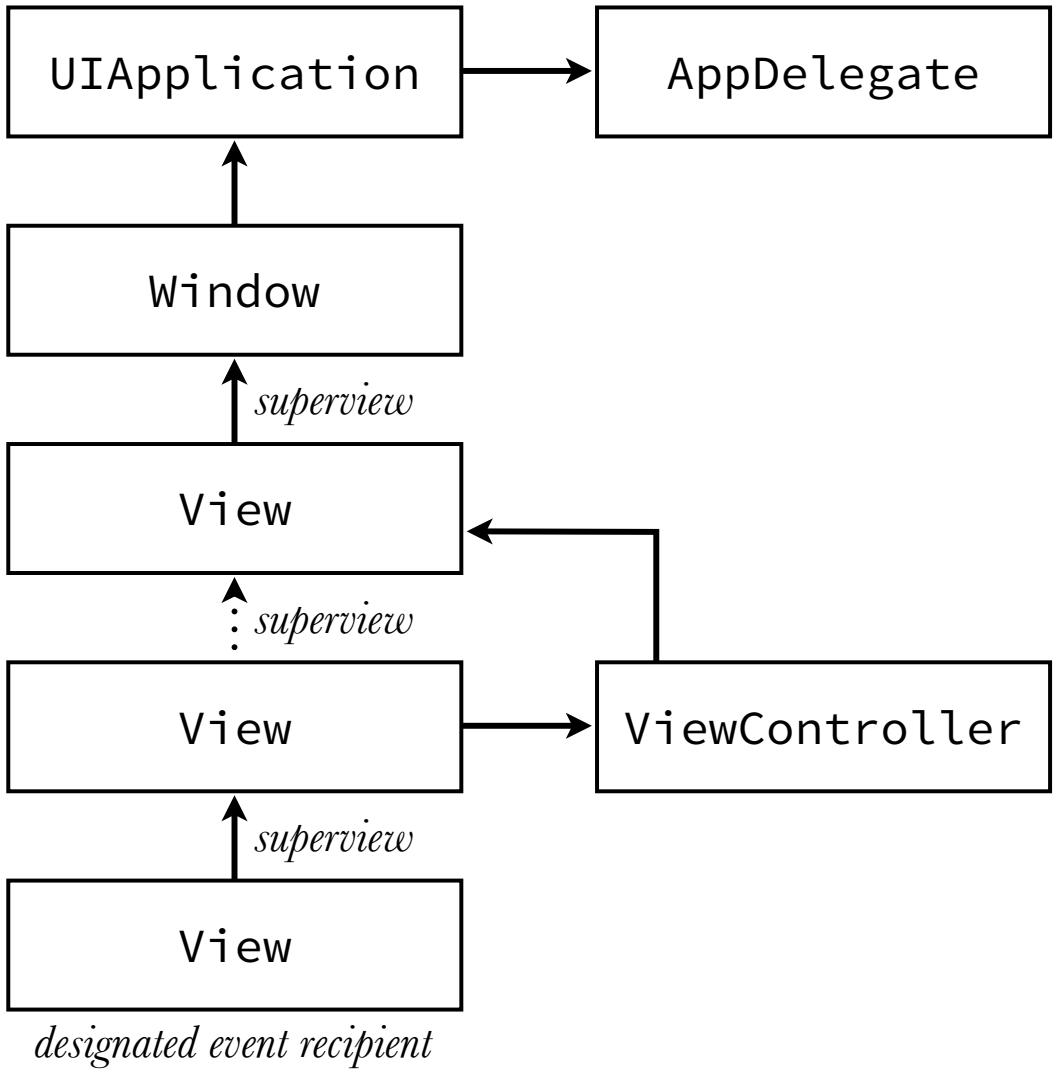
from Apple's *iOS App Programming Guide*



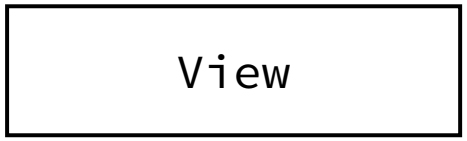
from Apple's *iOS App Programming Guide*

the AppDelegate also receives events that are not handled by views or view controllers

- e.g., low-level touch and motion events
- functions as “catch-all” handler



iOS responder chain



designated event recipient

- two basic ways for an object to be the designated recipient of an event
  1. *hit-testing* for touches
  2. assumes *first responder* status
- note: designated recipient doesn't have to be a view (any subclass of `UIResponder`)

sometimes it's handy to broadcast/receive notifications in non-default ways

- notifications outside our "scope"
- notifications that aren't sent by default
- app-specific notifications

Observer Pattern

- Any object can register to receive notifications of specific events
  - One mechanism: *notification centers*
- Enables *action at a distance* (not necessarily a good thing!)

Hands-on

- Projects: *EventDemo*
- Hit-test & first responders
- Touch & motion event handling
- Event propagation
- Textfields as responders & delegation

Target-Action & **Delegation** both embody the separation of (1) *the thing where events happen* and (2) *the thing that processes events (& supplies data)*

(1) is the **view**

(2) is the **view controller**

But the list of responsibilities is quite lopsided.

(1) just has to map *inputs* to high-level *actions*; e.g., a touch in a table area to cell selection

(2) has to (a) map an action to a *semantic intent*, then (b) actually *execute* that intent

e.g., (2)(a): tapping on a cell means that the user wants to reverse its contents

(2)(b): we need to reach into our data and figure out how to reverse character data (non-trivial for, say, CJK)

```objc
// (2)(a)
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    _data[indexPath.row] = [self reverseString:_data[indexPath.row]];
    [tableView reloadRowsAtIndexPaths:@[indexPath]
                     withRowAnimation:UITableViewRowAnimationAutomatic];
}


// (2)(b)
- (NSString *)reverseString:(NSString *)string
{
    NSMutableString *reversed = [NSMutableString stringWithCapacity:string.length];
    [string enumerateSubstringsInRange:NSMakeRange(0, string.length)
                               options:NSStringEnumerationByComposedCharacterSequences
                                       | NSStringEnumerationReverse
                            usingBlock:^(NSString *str, NSRange strRange,
                                         NSRange enclosingRange, BOOL *stop){
                                [reversed appendString:str];
                            }];
    return reversed;
}
```

```
// (2)(a)
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
```

- mapping action to intent *is* the controller's job

```
}


// (2)(b)
- (NSString *)reverseString:(NSString *)string
{
    NSMutableString *reversed = [NSMutableString stringWithCapacity:string.length];
    [string enumerateSubstringsInRange:NSMakeRange(0, string.length)
                               options:NSStringEnumerationByComposedCharacterSequences
                                       | NSStringEnumerationReverse
                            usingBlock:^(NSString *str, NSRange strRange,
                                         NSRange enclosingRange, BOOL *stop){
                                [reversed appendString:str];
                            }];
    return reversed;
}
```

```
// (2)(a)
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
```

- mapping action to intent *is* the controller's job

```
}

// (2)(b)
- (NSString *)reverseString:(NSString *)string
{
```

- but executing the intent often requires *domain-specific* knowledge/code
- has no place inside a *view controller*!

```
}
```

```
@implementation ListViewController
- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // view controller: ask data to “reverse” itself
    _data[indexPath.row] = [_data[indexPath.row] reversedString];
    [tableView reloadRowsAtIndexPaths:@[indexPath]
                     withRowAnimation:UITableViewRowAnimationAutomatic];
}
@end
```

```
@implementation NSString (Reversing)
// data: apply domain specific knowledge
- (NSString *)reversedString
{
    NSMutableString *string = [NSMutableString stringWithCapacity:self.length];
    [self enumerateSubstringsInRange:NSMakeRange(0, self.length)
                             options:NSStringEnumerationByComposedCharacterSequences
                                     | NSStringEnumerationReverse
                          usingBlock:^(NSString *str, NSRange strRange,
                                       NSRange enclosingRange, BOOL *stop){
                              [string appendString:str];
                          }];
    return string;
}
@end
```

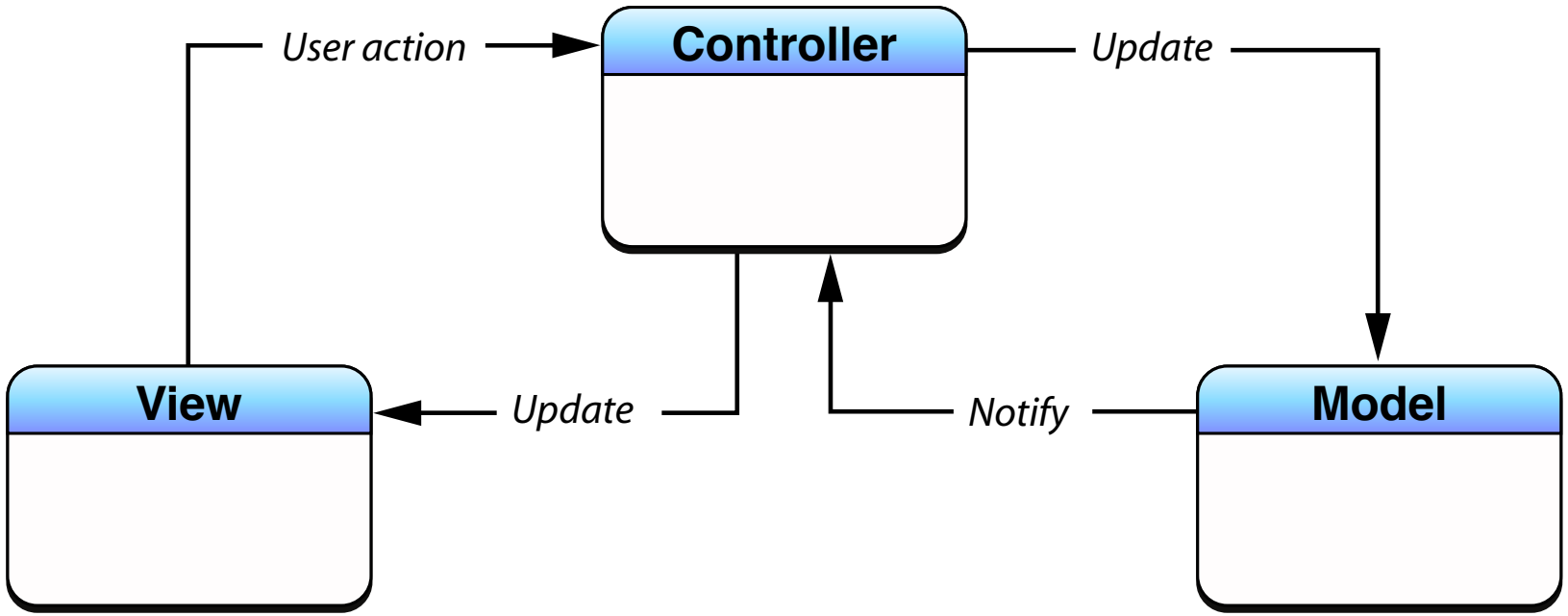
```
@implementation NSString (Reversing)
// data: apply domain specific knowledge
- (NSString *)reversedString
{
    NSMutableString *string = [NSMutableString stringWithCapacity:self.length];
    [self enumerateSubstringsInRange:NSMakeRange(0, self.length)
                             options:NSStringEnumerationByComposedCharacterSequences
                                     | NSStringEnumerationReverse
                          usingBlock:^(NSString *str, NSRange strRange,
                                       NSRange enclosingRange, BOOL *stop){
                              [string appendString:str];
                          }];
    return string;
}
@end
```

- our data classes encapsulate information and domain knowledge
- i.e., not just “data”
  - term them *data models*, or just ***models***

```
@implementation NSString (Reversing)
// data: apply domain specific knowledge
- (NSString *)reversedString
{
    NSMutableString *string = [NSMutableString stringWithCapacity:self.length];
    [self enumerateSubstringsInRange:NSMakeRange(0, self.length)
                             options:NSStringEnumerationByComposedCharacterSequences
                                     | NSStringEnumerationReverse
                          usingBlock:^(NSString *str, NSRange strRange,
                                       NSRange enclosingRange, BOOL *stop){
                              [string appendString:str];
                          }];
    return string;
}
@end
```

- added benefit: we can carry along models we've built from project-to-project
- don't (shouldn't) carry with them any user-interface cruft or assumptions

Model-View-Controller



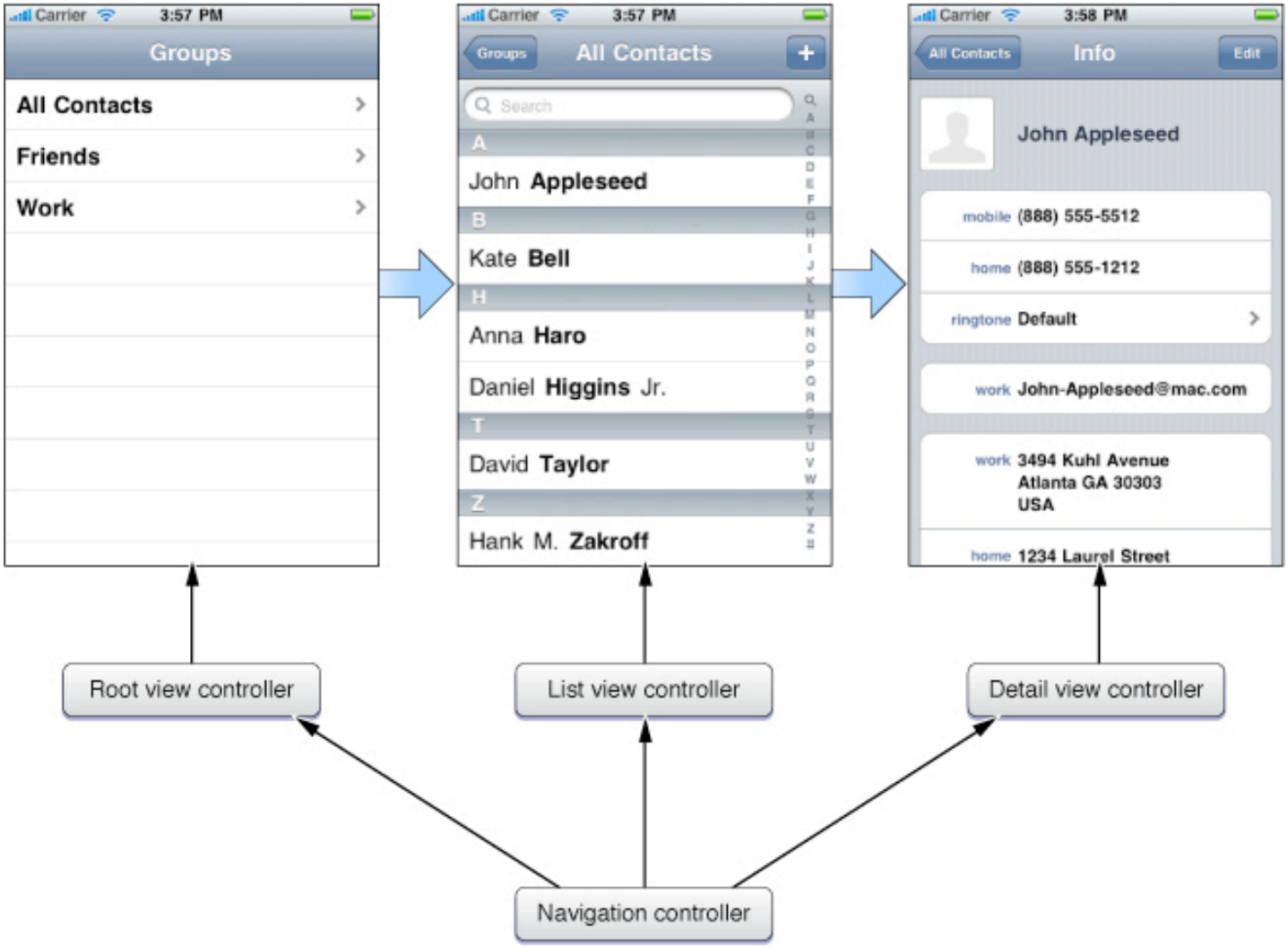
MVC is *central* to the architecture of virtually all native iOS apps, and is found at all levels of the development stack

Hands-on

- Projects: *ReversibleGreetings, SimpleTodo*
- MVC breakdown
- Table view (cell) customization
- A dash of persistence

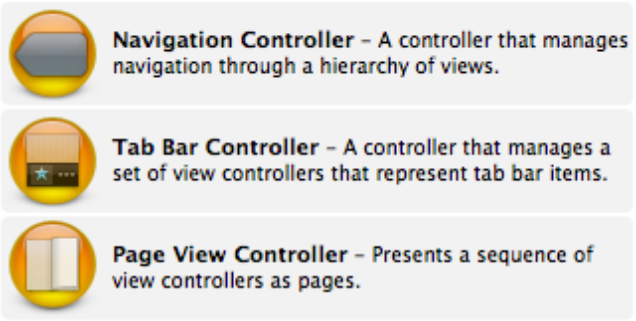
Turns out the one-controller-per-scene tale is not really accurate …

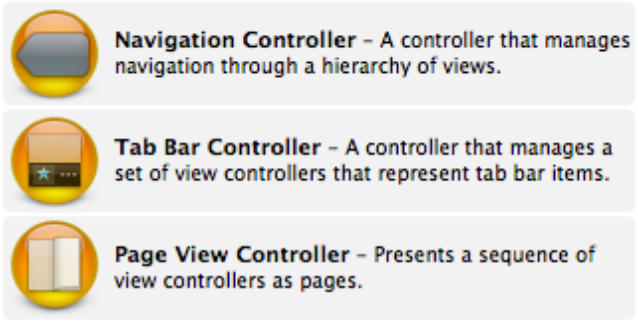
- often need a controller to *manage transitions between* separate, per-screen controllers
- overarching "container" controllers



e.g., Address Book app

Built-in controllers for this!





Good news: intended for use *without subclassing*

- typically drop into Storyboard & customize in code if necessary
- segues are used as to connect controllers — e.g., to pass data
  - (true even without container controllers)

Hands-on

- Project: *Cars*
- Using a navigation controller
- Passing data along with segues
- Plist serialization

</essential-patterns>

<misc-builtins>

System Controllers

- Plenty of additional built-in system view controllers + associated views
  - Use without subclassing
  - Typically present programmatically
    - Obtain result via delegation or block

Hands-on

- Project: *DoesItAll*
- PhotoPicker, E-mail, Twitter, etc.
- Programmatic presentation / dismissal
- Delegation & Block completion handlers

Social Integration

- Built-in “Social” framework
  - Facebook & Twitter
- Automatic account handling & authentication

Hands-on

- Project: *YATC*
- Twitter API access / authentication
- JSON parsing

Web Views

- WebKit-backed view
  - Trivial to integrate an app-local browser
- Also useful for rich-text processing and JavaScript execution

★ Bonus: brand new web inspector debugging

Hands-on

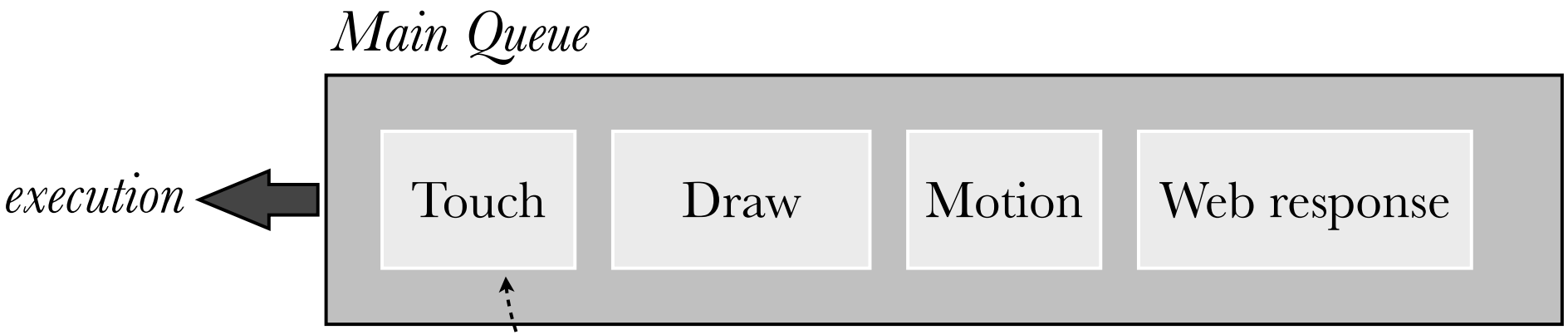
- Projects: *MySafari*, *OnesidedChat*
- WebView usage
- JavaScript execution from ObjC
- Web inspector debugging

Concurrency

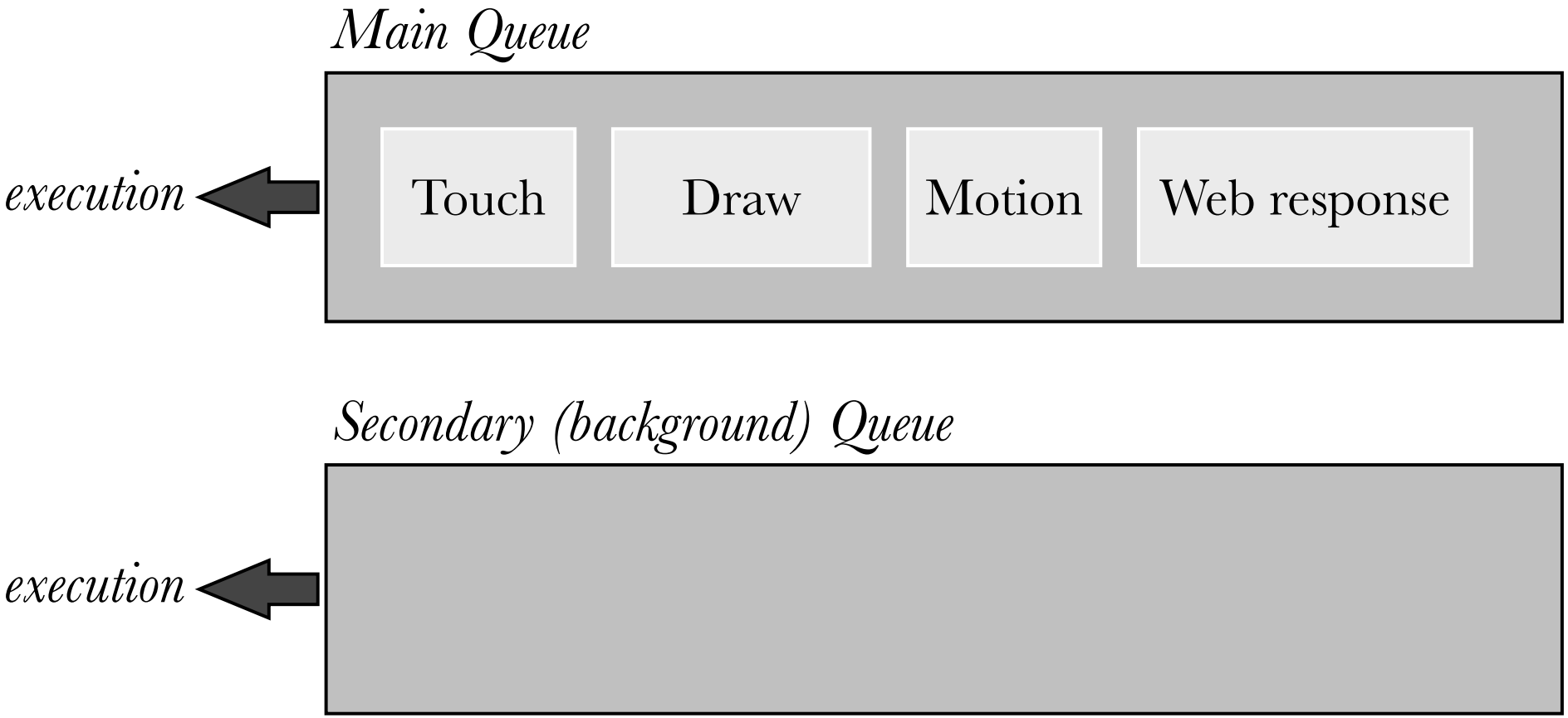
- All UIKit drawing and event dispatching happens in a single thread / *run loop*
- To exploit multi-core phones (and improve responsiveness) need to multi-thread
  - But prefer to *avoid* ***explicit*** *multithreading*

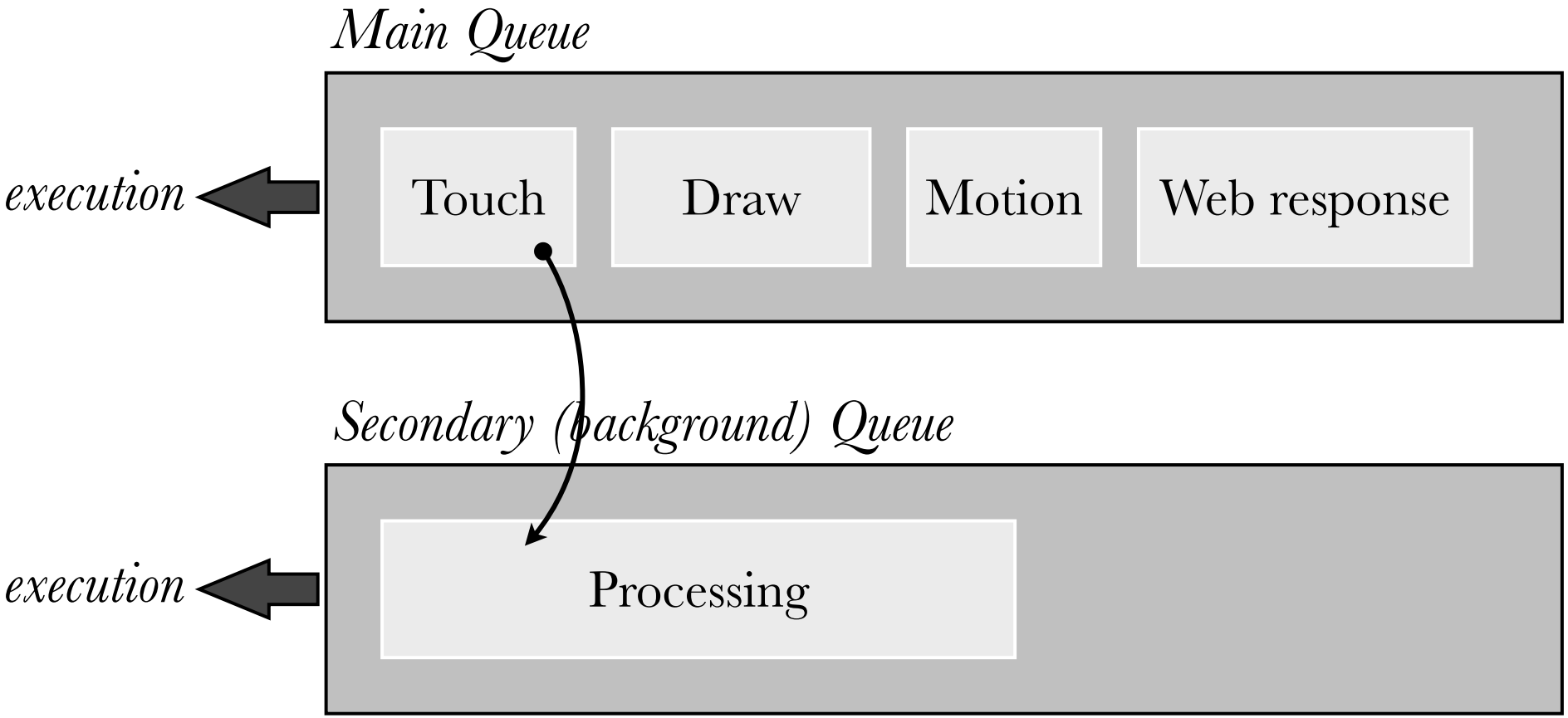
Concurrency

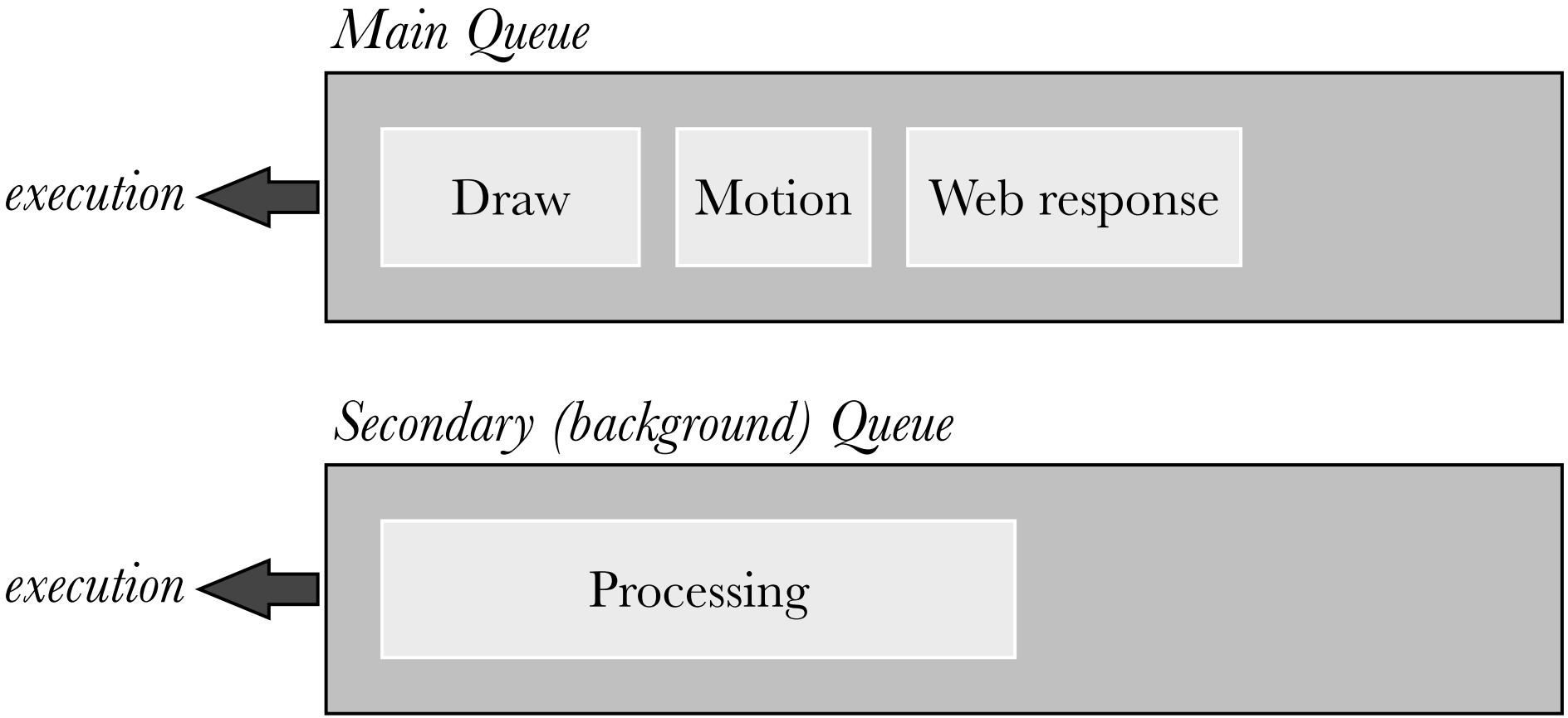
- High level concurrency support via "operation queues"
  - Conceptually, a pool of workers that pulls items of a queue of operations
    - Single worker for the main queue!

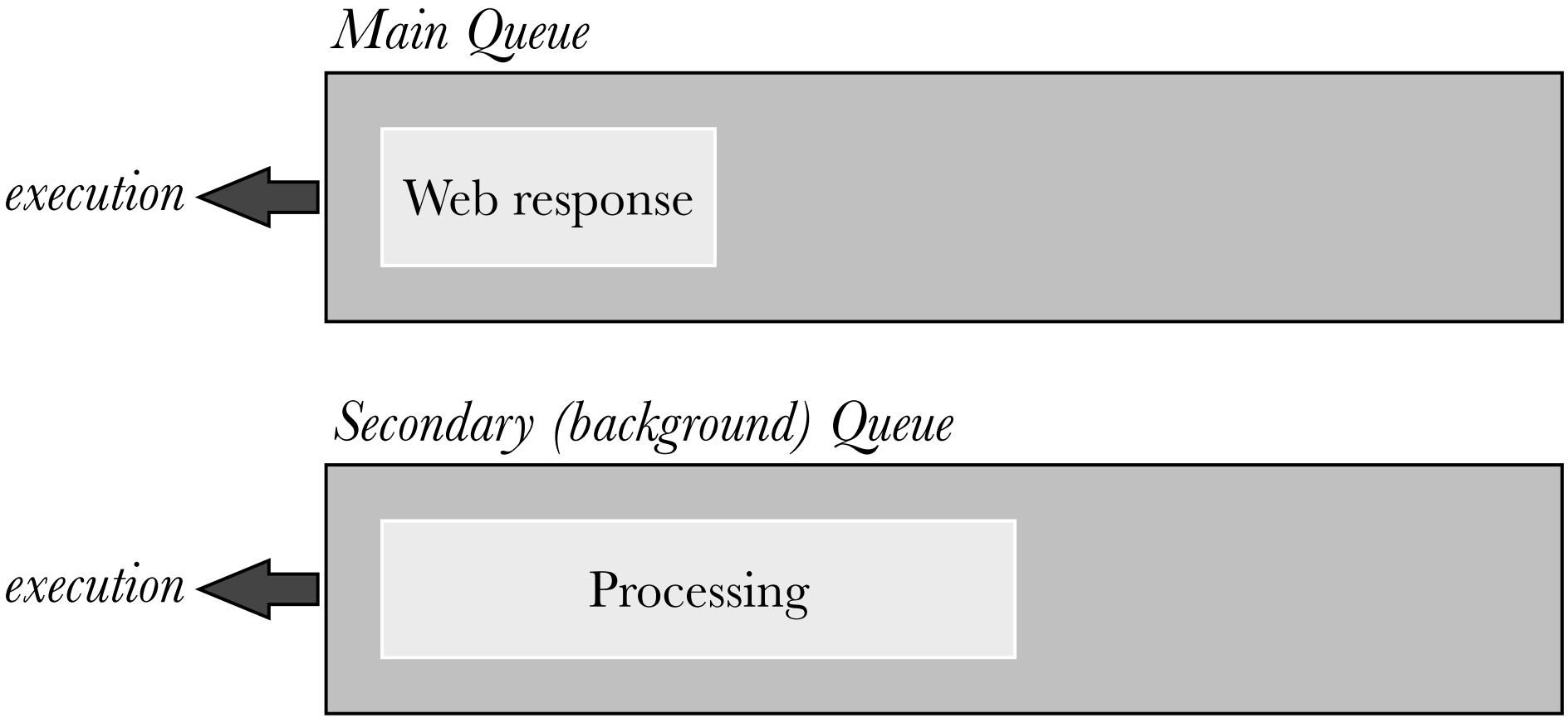


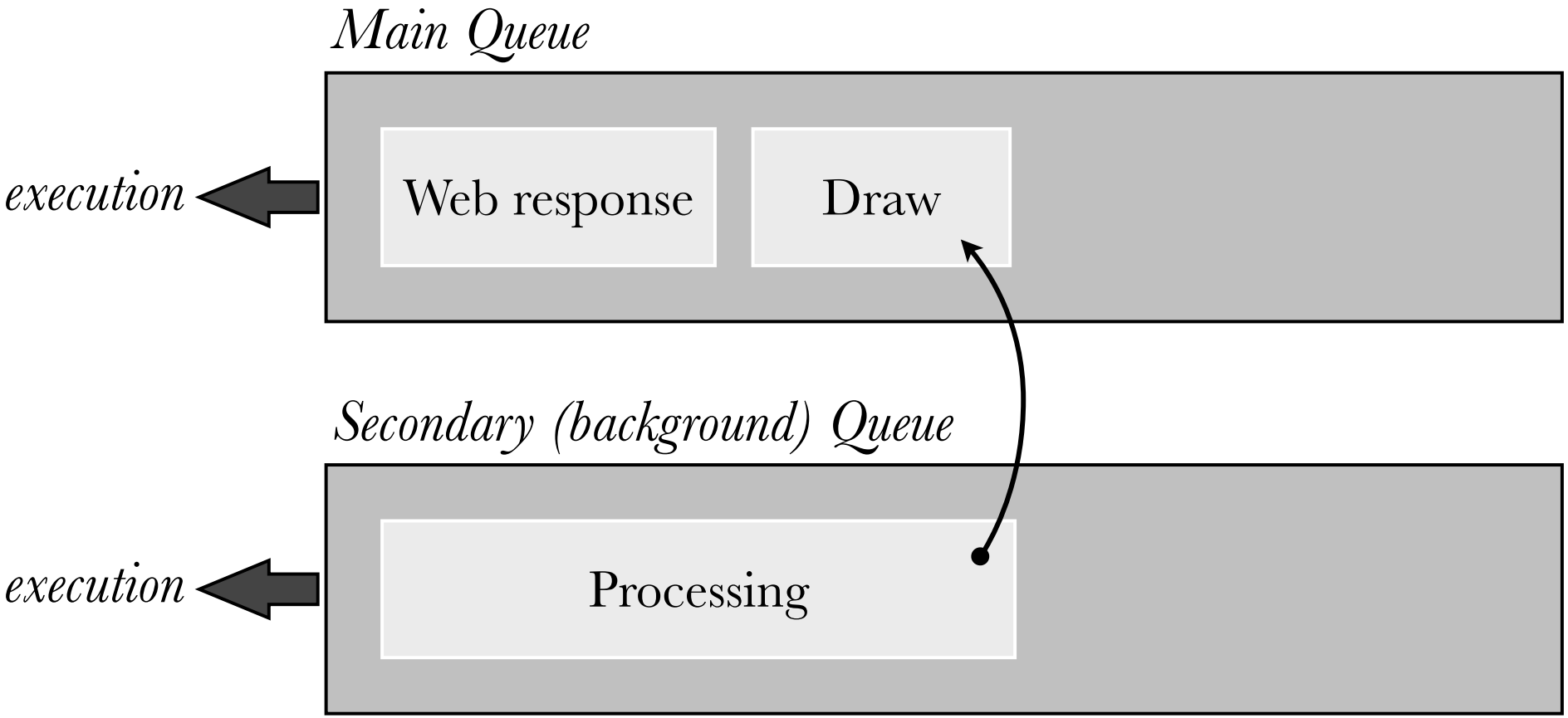
needs to perform lengthy processing … but would hold up the main thread if done here

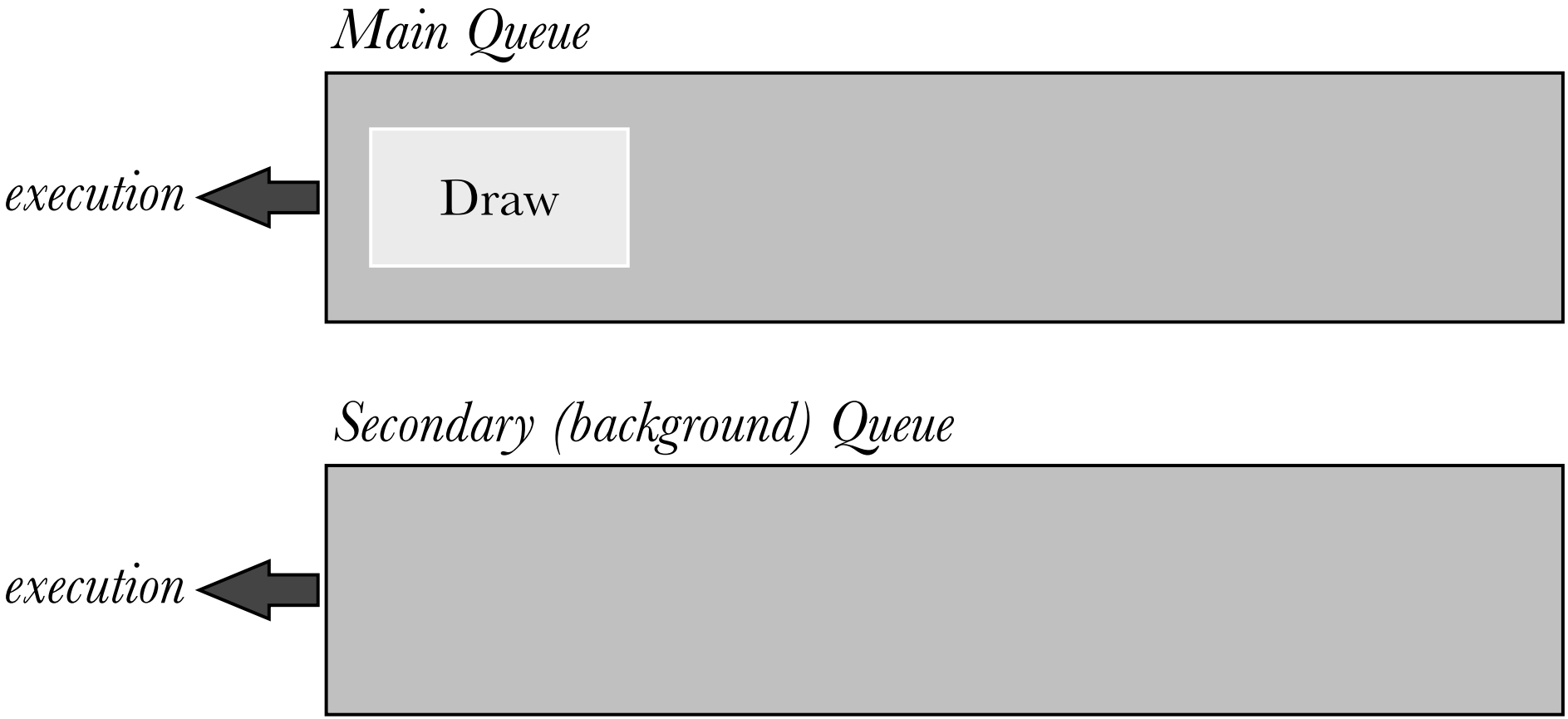

Main Queue
execution
Touch
Draw
Motion
Web response
Secondary (background) Queue
execution


Main Queue
execution
Touch
Draw
Motion
Web response
Secondary (background) Queue
execution
Processing


Main Queue
execution
Draw
Motion
Web response
Secondary (background) Queue
execution
Processing


Main Queue
execution
Web response
Secondary (background) Queue
execution
Processing


Main Queue
execution
Web response
Draw
Secondary (background) Queue
execution
Processing


Main Queue
execution
Draw
Secondary (background) Queue
execution

Hands-on

- Projects: *DominantColors*
- Concurrency with NSOperationQueue
- Incorporating a C library

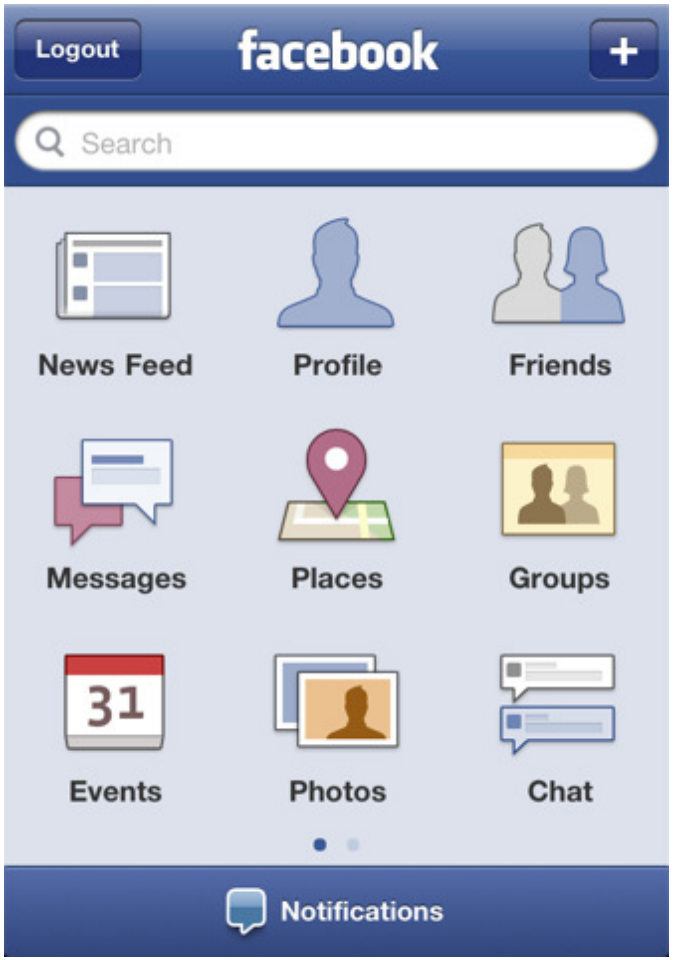
§Alternatives to (Apple's) iOS SDK

there are a *vast* number of alternative/ supplementary frameworks and tools

e.g., for game development, Cocos2D
http://www.cocos2d-iphone.org

- full featured game engine

- implements its own scene management conventions/API



e.g., (now defunct) Three20: filled a need for non-pre-built "grid-view" controllers

caveat emptor: third-party (open-source) frameworks often aren't maintained well, and Apple switches up the iOS SDK frequently (making them unsupported/redundant)

yet another alternative:

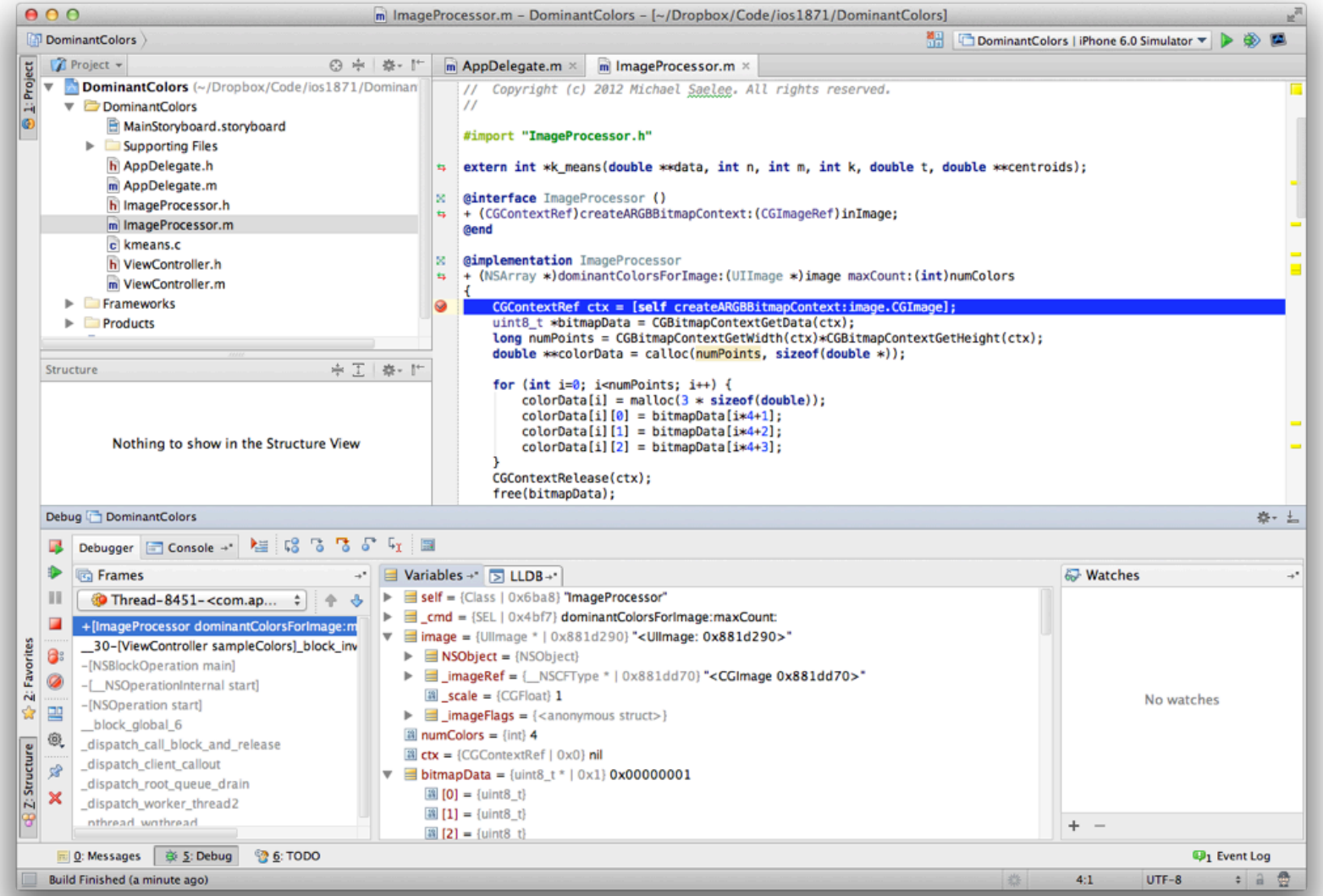
HTML5, CSS, JavaScript based “native” apps via **Apache Cordova**

- packaged as apps, distributed via app store

- runs in local web view

there's even a viable alternative IDE to Xcode!

- JetBrains "AppCode"
- http://www.jetbrains.com/objc/



AppCode screenshot

§Resources

Developer resources

- http://developer.apple.com
  - API documentation
  - Code samples
  - ★ WWDC session videos
  - Developer forums

Classes

- No “certification” available (yet?)
- Our own seminars & semester-long classes
  - Note: seminar content is close to this, but longer for demos + hands-on sessions
- Online Stanford & other iPhone classes

§Questions?

Thanks!

Michael Saelee <lee@iit.edu>
Senior Lecturer, Department of Computer Science

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY