Cellular Networks and Mobile Computing COMS 6998-8, Spring 2012

Instructor: Li Erran Li
(lierranli@cs.columbia.edu)

http://www.cs.columbia.edu/~coms6998-8/

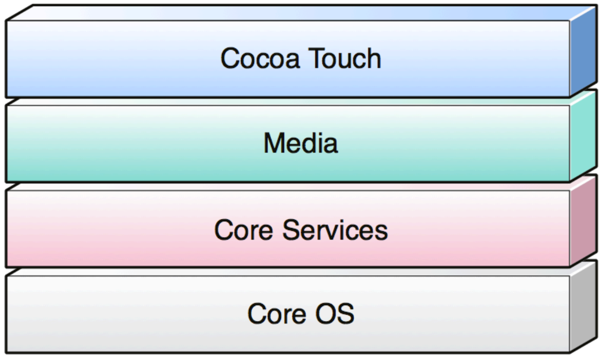
2/6/2012: Introduction to iOS programming

Outline

- iOS Overview
- Objective-C
- Model-View-Controller
- Demo
- Networking
- iCloud

iOS Architecture

- Implemented as a number of layers
- Lower layers provide fundamental services and technologies
- Higher layers provide more sophisticated services
  - Builds upon the functionality provided by the lower layers
  - Provides object-oriented abstractions for lower layer constructs



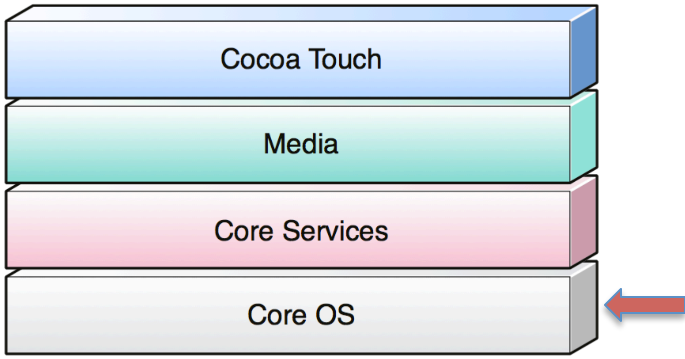
iOS Frameworks

- Frameworks are packages of system interfaces.
  - Each framework contains dynamically shared libraries and associated resources (header files, images, etc)
  - When a framework is used, they need to be linked into the project
    - Standard frameworks such as Foundation and UIKit are linked by default, when a template project is started
- Higher level frameworks often build on lower level frameworks

iOS Overview: CoreOS

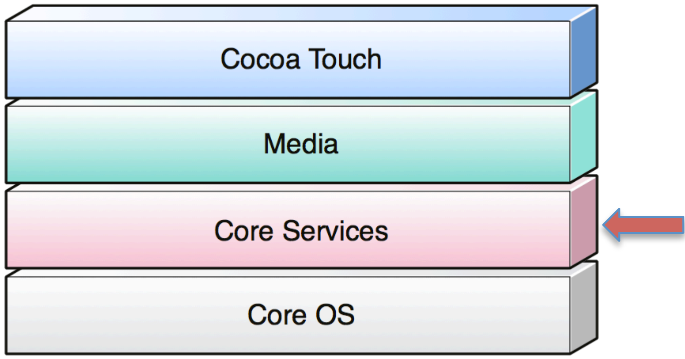
CoreOS is based on Mach

- System Framework
  - Threading (POSIX)
  - Networking (BSD sockets)
  - File system
  - Service discovery (Bonjour & DNS)
  - Memory management
  - Math computations
- External Accessory Framework and Core Bluetooth Framework: support for communicating with hardware accessories
- Security Framework: crypto library and keychain Services (secure storage of passwords, keys, for one or more users)
- Accelerate Framework
  - DSP, linear algebra and image processing optimized for hardware



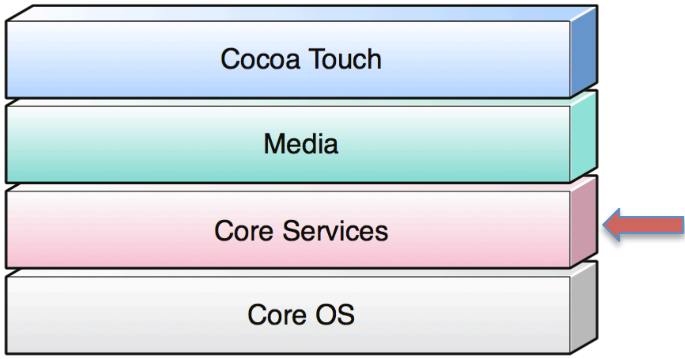
iOS Overview: Core Services

- High level features
  - iCloud storage (iOS5)
  - Automatic reference counting (iOS5)
  - SQLite: lightweight SQL database
  - Grand Central Dispatch (GCD): manage concurrent execution of tasks
    - Thread management code moved to the system level
    - Tasks specified are added to an appropriate dispatch queue.
  - Block objects: a C-level language construct; an anonymous function and the data (a closure or lambda)
  - In-App purchase: process financial transactions from iTune account
  - XML support



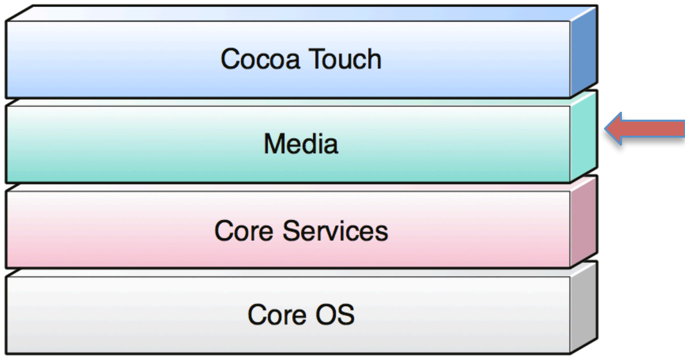
iOS Overview: Core Services (Cont'd)

- CFNetwork Framework: object-oriented abstractions for working with network protocols (DNS, http, ftp, Bonjour services)
- Address Book Framework
- Core Data Framework
- Core Foundation Framework: arrays, sets, string, url, threads
- Foundation Framework: Objective-C wrapper
- Core Media Framework
- Core Location Framework
- Core Telephony Framework
- Newsstand Kit Framework (iOS5): a central place to read newspapers and magazines
- Store Kit Framework: support purchasing from iOS apps
- System Configuration Framework: determine network configuration



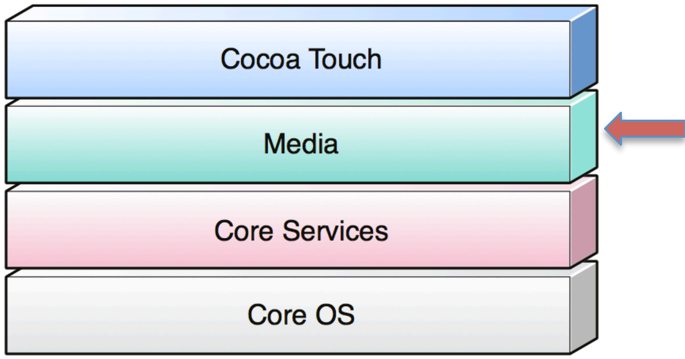
iOS Overview: Media

- High level features
  - Graphics
    - Core graphics
    - Core animation
    - Core image
    - OpenGL ES and GLKit
    - Core text
  - Audio/video
    - Meida player
    - OpenAL
    - Core audio
    - Core media
  - AirPlay: stream audio to Apple TV and to third-party AirPlay receivers



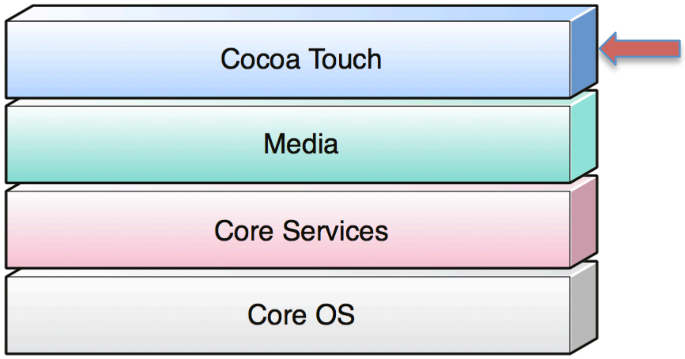
iOS Overview: Media (Cont'd)

- Core Audio Framework
- Core Graphics Framework
- Core Video Framework: provides buffer and buffer pool support for the Core Media framework
- Core MIDI Framework
- Core Image Framework
- Core Text Framework
- Quartz Core Framework: core animation
- AV Foundation Framework: Objective-C classes for playing audio/video content
- Asset Library Framework: query-based interface for retrieving photos and videos from user's device



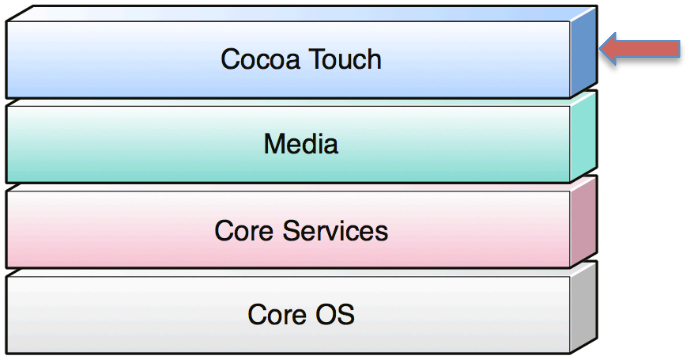
iOS Overview: Cocoa Touch

- High level features
  - Storyboards: supplant nib files as the recommended way to design your application's user interface
  - Document Support: UIDocument class for managing the data associated with user documents
  - Multitasking
  - Printing: support allows applications to send content wirelessly to nearby printers
  - Data protection
  - Push notification
  - Gesture recognizers
  - File-sharing
  - Peer-to-peer services: over Bluetooth, e.g. multi-player games



iOS Overview: Cocoa Touch (Cont'd)

- UIKit Framework: storyboard, multi-touch, cut-copy-paste, multi-tasking, push notification, accelerometer data, built-in camera, battery state information, proximity sensor information
- Event Kit UI Framework: calendar related
- Address Book UI Framework: contact management
- Game Kit Framework
- iAd Framework: deliver banner-based advertisements from your application
- Map Kit Framework: a scrollable map interface
- Message UI Framework: support for composing and queuing email messages in the user's outbox
- Twitter Framework



Outline

- iOS Overview
- Objective-C
- Model-View-Controller
- Demo
- Networking
- iCloud

Objective-C

- A strict superset of ANSI C
- Originally used within NeXT's NEXTSTEP OS (precursor of Mac OS X)
- Single inheritance
- Dynamic runtime: everything is looked up and dispatched at run time
- No garbage collection on iPhone, iTouch and iPad
- New types
  - `id` type: dynamic type to refer to any object
  - Selectors: a message and arguments that will (at some point) trigger the execution of a method

Objective-C

- Introspection
  - An object (class, instance, etc) can be asked at runtime what type it is
    - Can pass anonymous objects to a method, and let it determine what to do based on the object's actual type

`isKindOfClass`: returns whether an object is that kind of class (inheritance included)
`isMemberOfClass`: returns whether an object is that kind of class (no inheritance)
`respondsToSelector:` returns whether an object responds to a given method

Objective-C header file and interface

```
#import <Foundation/Foundation.h>
@interface Stack : NSObject
@property (nonatomic, strong) NSMutableArray *numStack;

-(void) push: (double) num;
-(double) pop;
@end
```

Objective-C stack.h header file

- instance variables are declared as properties
- By default: @protected access
- "-" denotes instance methods

```
define STACKSIZE 10
Class Stack {
private:
    double num[STACKSIZE+1];
    int top;

public:
    Stack();
    void push(double x);
    double  pop();
};
```

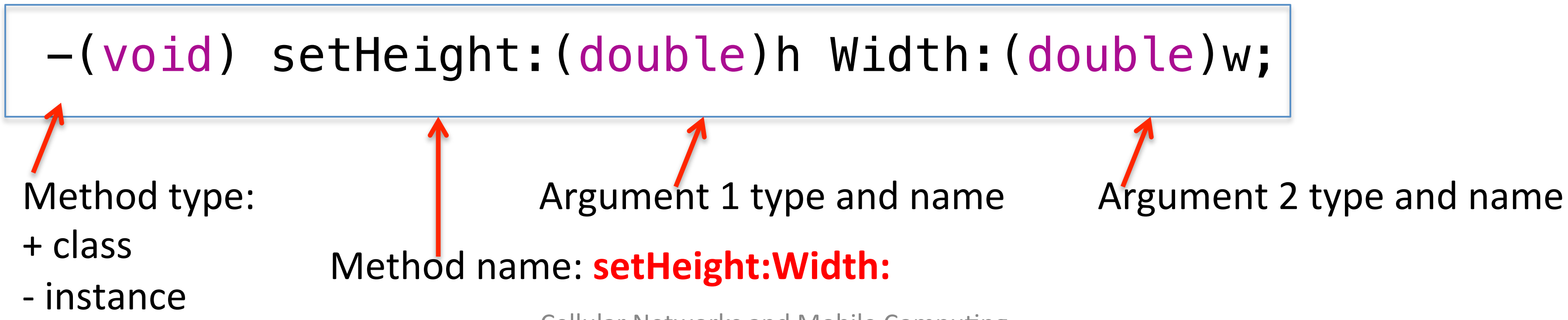
C++ header file

Objective-C Properties

- Provide access to object attributes
  - Shortcut to implementing getter/setter methods
  - Instead of declaring "boilerplate" code, have it generated automatically
- Also allow you to specify:
  - `readonly` versus `readwrite` access memory management policy
  - Memory management: `weak` and `strong`
- Specify `@property` in the header (*.h) file
- Create the accessor methods by `@synthesize` the properties in the implementation (*.m) file

Objective-C Method Declaration

- Each method declaration consists of:
  - A name
  - A return type
  - An optional list of arguments (and their data or object types)
  - An indicator to determine if the method is a class or instance method



```
-(void) setHeight:(double)h Width:(double)w;
```

Objective-C Implementation

```
#import "Stack.h"

@implementation Stack
@synthesize numStack = _numStack;

- (NSMutableArray *) numStack {
    if (_numStack==nil)
        _numStack = [[NSMutableArray alloc] init];
    return _numStack;
}

- (void) push:(double)num {
    [self.numStack addObject:[NSNumber numberWithDouble:num]];
}

- (double) pop {
    NSNumber *numObject = [self.numStack lastObject];
    if(numObject) [self.numStack removeLastObject];
    NSLog(@"poped %@",numObject);
    return [numObject doubleValue];

}
@end
```

Objective-C stack.m file

@synthesize creates getter and setter methods
alloc: a class method

Method syntax
self: the instance itself
dot notation to access setter and getter method

Objective-C Message Syntax

- A square brace syntax

```
[receiver message]
[receiver message:argument]
[receiver message:arg1 :anonymousArg2]
[receiver message:arg1 andArg:arg2]
```

Object receiving the message

Message itself

Main argument

Subsequent named argument

C++ Implementation

```
#include "stack.h"

Stack::Stack()
{
    index = top;
}

void Stack::push(double x)
{
    if(!is_full())
        num[top++] = x;
}

double Stack::pop()
{
    if(!is_empty())
        return num[--top];
    else
        return -1;
}
```

Method syntax

Objective-C Categories and Extensions

- Categories allows new methods to be added to existing class without using subclass
  - category name is listed within parentheses after the class name and the superclass isn't mentioned
- Class extensions are like anonymous categories
  - @interface MyClass ()
  - Methods must be implemented in the main @implementation block for the corresponding class

```
#import <Foundation/Foundation.h>
#import "Stack.h"
@interface Stack (emptyFull)

-(BOOL) isEmpty;
-(BOOL) isFull;
@end
```

StackExt.h

```
#import "StackExt.h"
#define STACK_CAP 100

@implementation Stack(emptyFull)
- (BOOL) isEmpty{
    return ([self.numStack count]==0);
}

- (BOOL) isFull{
    return ([self.numStack count]==STACK_CAP);
}
@end
```

StackExt.m

Objective-C Protocols

- Class and category interfaces declare methods that are associated with a particular class
- protocols declare methods that are independent of any specific class
- Protocols declare methods that can be implemented by any class. Protocols are useful in at least three situations:
  - To declare methods that others are expected to implement
  - To declare the interface to an object while concealing its class
  - To capture similarities among classes that are not hierarchically related

```
@protocol MyXMLSupport
@required
- (void) initFromXMLRepresentation:
(NSXMLElement *)XMLElement;
- (NSXMLElement *)XMLRepresentation;

@optional
- (void)anOptionalMethod;
@end
```

```
@interface aClass <MyXMLSupport>
@end
@interface aClass(categName)<MyXMLSupport>
@end
```

```
@implementation className
…
if (![receiver conformsToProtocol:@protocol
(MyXMLSupport)])
…
@end
```

Objective-C Protocols (Cont'd)

```
#import <UIKit/UIKit.h>
@interface CalculatorAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;
@end
```

CalculatorAppDelegate.h

```
@interface UIApplication (UINewsstand)
- (void)setNewsstandIconImage:(UIImage *)image;
@end

@protocol UIApplicationDelegate<NSObject>
@optional
- (void)applicationDidFinishLaunching:(UIApplication *)application;
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
__OSX_AVAILABLE_STARTING(__MAC_NA,__IPHONE_3_0);

-  (void)applicationDidBecomeActive:(UIApplication *)application;
@end
```

$\uparrow$

UIApplication.h

Objective-C: Associative References

- Associative references
  - Simulate the addition of object instance variables to an existing class
- Fast enumeration
  - The enumeration is considerably more efficient than, for example, using NSEnumerator directly.
  - The syntax is concise.
  - Enumeration is "safe"—the enumerator has a mutation guard so that if you attempt to modify the collection during enumeration, an exception is raised

```
@interface UIView (ObjectTagAdditions)
@property (nonatomic, strong) id objectTag;
- (UIView *)viewWithObjectTag:(id)object;
@end

#import <objc/runtime.h>
static char const * const ObjectTagKey =
"ObjectTag";
@implementation UIView (ObjectTagAdditions)
@dynamic objectTag;

- (id)objectTag {
    return objc_getAssociatedObject(self,
ObjectTagKey);
}

- (void)setObjectTag:(id)newObjectTag {
    objc_setAssociatedObject(self,
ObjectTagKey, newObjectTag,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}
...
@end
```

Objective-C: Fast Enumeration

- The enumeration is considerably more efficient than, for example, using NSEnumerator directly.
- The syntax is concise.
- Enumeration is "safe"—the enumerator has a mutation guard so that if you attempt to modify the collection during enumeration, an exception is raised

```
NSArray *array = [NSArray arrayWithObjects:
                     @"one", @"two", @"three",
@"four", nil];

for (NSString *element in array) {
    NSLog(@"element: %@", element);
}
```

Objective-C: Foundation Framework

- Root class: allocation, initialization and duplication of objects, introspection, object encoding and decoding (for archiving / serialization), message forwarding and message dispatching
  - `NSObject`

- Value objects: encapsulate values of various primitive types
  - `NSNumber`
  - `NSDate`
  - `NSString`
  - `NSData`

- Collections: collections are objects that store other objects
  - `NSArray, NSMutableArray`
  - `NSDictionary, NSMutableDictionary`
  - `NSSet, NSMutableSet`

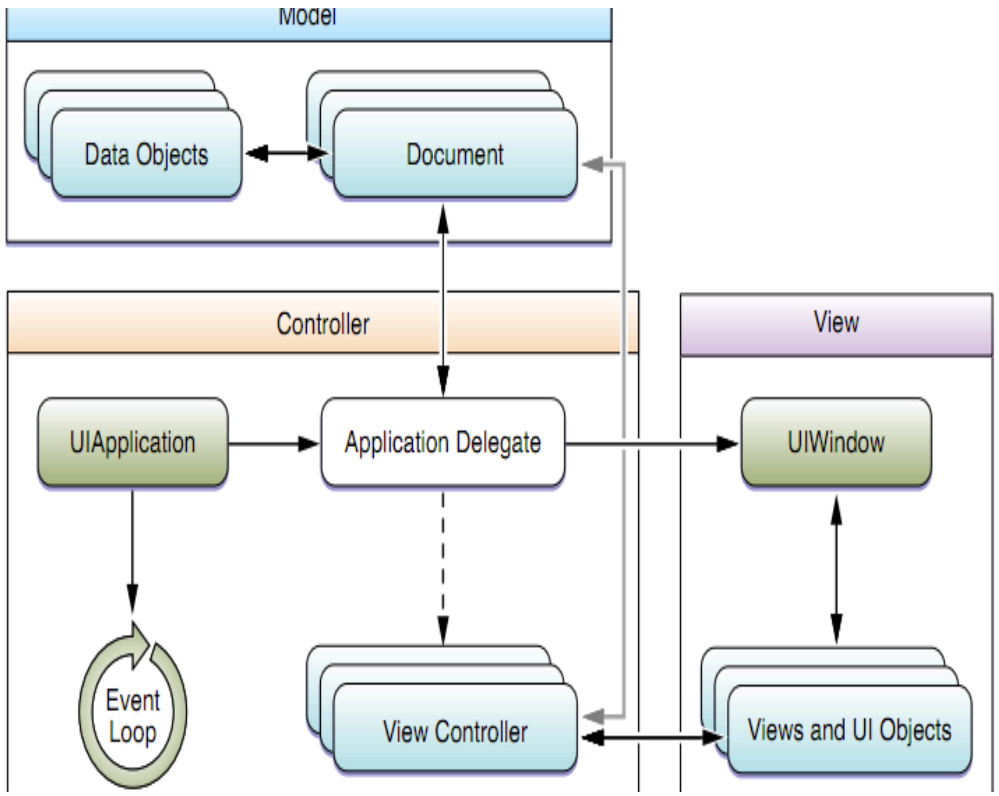
Outline

- iOS Overview
- Objective-C
- Model-View-Controller
- Demo
- Networking
- iCloud

MVC Design Pattern

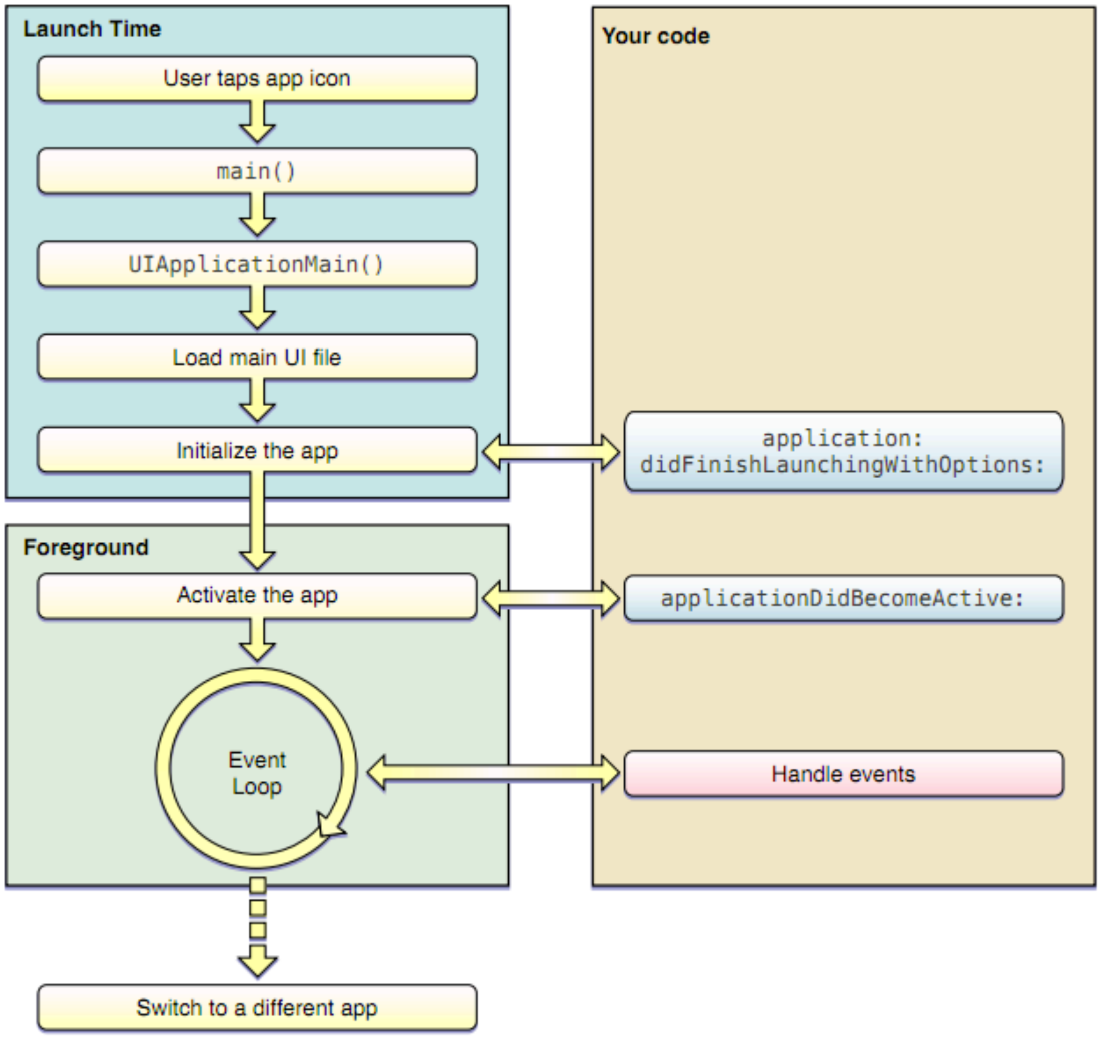
Key objects in iOS apps

- `UIApplication` controller object
  - manages the app event loop
  - coordinates other high-level app behaviors
  - custom app-level logic resides in your app delegate object
- App delegate custom object: created at app launch time, usually by the `UIApplicationMain` function. The primary job of this object is to handle state transitions within the app



MVC Design Pattern (Cont'd)

App launch cycle



MVC: Model

Model: contains the app's underlying data

- Could correspond to an external data source or some current model
  - iTunes database, stored files, internal state of a game
- Actions on the model manage the app data and its state
- Not aware of UI or presentation
  - Leave the interface to the view, and the application logic to the controller
- Models are reusable

MVC: View

View is what you see on screen

- Canvas, interface elements: buttons, labels, table views, etc
- No data stored
  - Model maintains data
  - Updates to model through controller

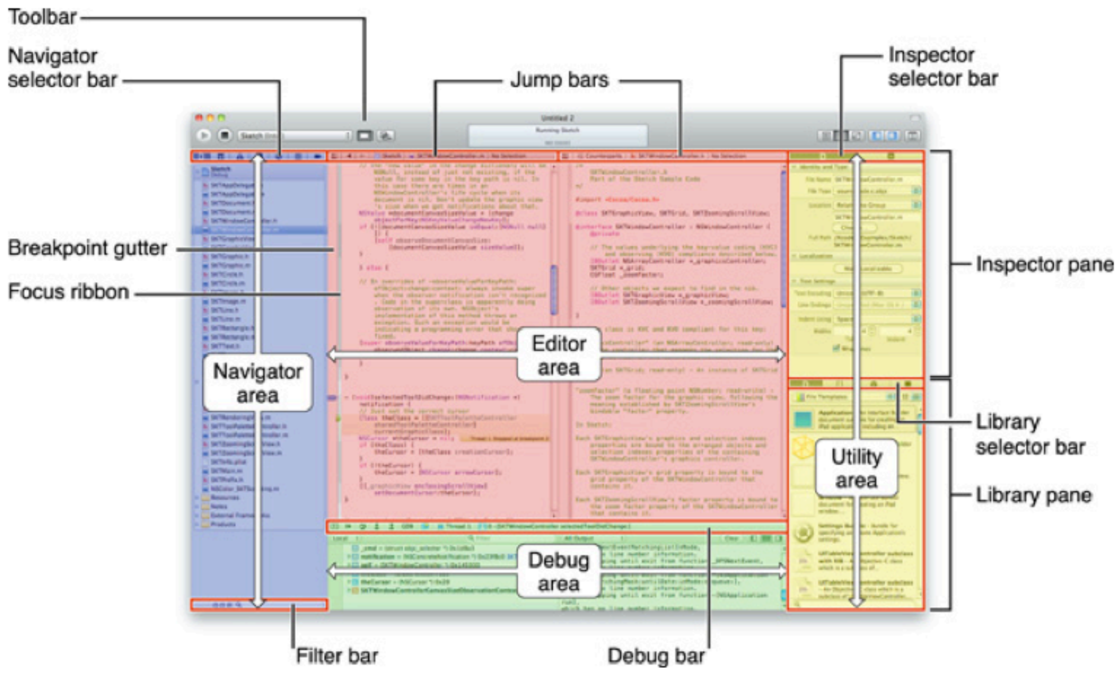
MVC: Controller

Controller

- Knows both model and view
- Acts as a middleman
  - When model changes, inform the view
  - When data manipulated by view, update the model
- Build-in iOS controllers
  - `UIViewController`: managing apps with generic views
  - `UITabBarController`: for tabbed applications (e.g. clock)
  - `UINavigationController`: managing hierarchical data (e.g. email folders)
  - `UITableController`: for lists of data etc (e.g. iTunes tracks)

Xcode4

- The latest IDE for developing MacOSX and iOS applications
  - Single window, supporting multiple workspace
  - Integrated Interface Builder
  - Assistant Editor (split pane that loads related files, such as header files etc)
  - Dynamic syntax checking and alert
  - Version editor with Git or Subversion integration
  - LLVM 2.0 editor with support for C, C++ and Objective-C
  - LLDB debugger



Networking

- CFNetwork: Core Services framework that provides a library of abstractions for network protocols.
  - Working with BSD sockets
  - Creating encrypted connections using SSL or TLS
  - Resolving DNS hosts
  - Working with HTTP, authenticating HTTP and HTTPS servers
  - Working with FTP servers
  - Publishing, resolving and browsing Bonjour services: CFNetServices API provides access to Bonjour through three objects
    - `CFNetService` represents a single service on the network
    - `CFNetServiceBrowser` discovers domains and discover network services within domains.
    - `CFNetServiceMonitor` monitors services for changes to their TXT records

Networking (Cont'd)

- Core Telephony framework: obtain information about a user's home cellular service provider
  - `CTCarrier` object provides information about the user's cellular service provider
  - `CTCall` object provides information about a current call, including a unique identifier and state information—dialing, incoming, connected, or disconnected

iCloud

Fundamentally: nothing more than a URL of a shared directory

- Two storage models
  - iCloud document storage: store user documents and app data in the user's iCloud account
  - iCloud key-value data storage: share small amounts of noncritical configuration data among instances of your app

- iCloud-specific entitlements required
  - Select your app target in Xcode
  - Select the Summary tab
  - In the Entitlements section, enable the Enable Entitlements checkbox

iCloud (Cont'd)

- Check availability: `URLForUbiquityContainerIdentifier:`
- All files and directories stored in iCloud must be managed by a file presenter object, and all changes you make to those files and directories must occur through a file coordinator object. A file presenter is an object that adopts the `NSFilePresenter` protocol
- Explicitly move files to iCloud
- Be prepared to handle version conflicts for a file
- Make use of searches to locate files in iCloud
- Be prepared to handle cases where files are in iCloud but not fully downloaded to the local device; this might require providing the user with feedback
- Use Core Data for storing live databases in iCloud; do not use SQLite

Online Resources

- Client side: iOS
  - Install Xcode 4: http://developer.apple.com/xcode
  - Learning Objective C and iOS development : http://developer.apple.com/devcenter/ios/index.action
  - Stanford iPhone development course(on iTunes): http://www.stanford.edu/class/cs193p/cgi-bin/drupal/

Questions?