CS193P - Lecture 13

iPhone Application Development

Debugging Tips
Searching
Notifications
KVC/KVO

Announcements

- This Friday: Jessica Kahn, Tapulous

- Presence 3 was due last night at midnight
  - Ask if you're not sure how many late days you've used
- Presence 4 is out today, due Tuesday 5/19

The Next Few Weeks

- Work on your final projects!
  - **Due on Sunday 6/7 at 11:59PM**
- Upcoming lectures
  - Multitouch
  - Device APIs: Accelerometer, Location
  - Audio/Video/Web Views
  - More...

Today's Topics

- Exceptions
- Searching
- Notifications
- Key-Value Coding & Observing
- Presence 4

Exceptions

Exceptions in Objective-C

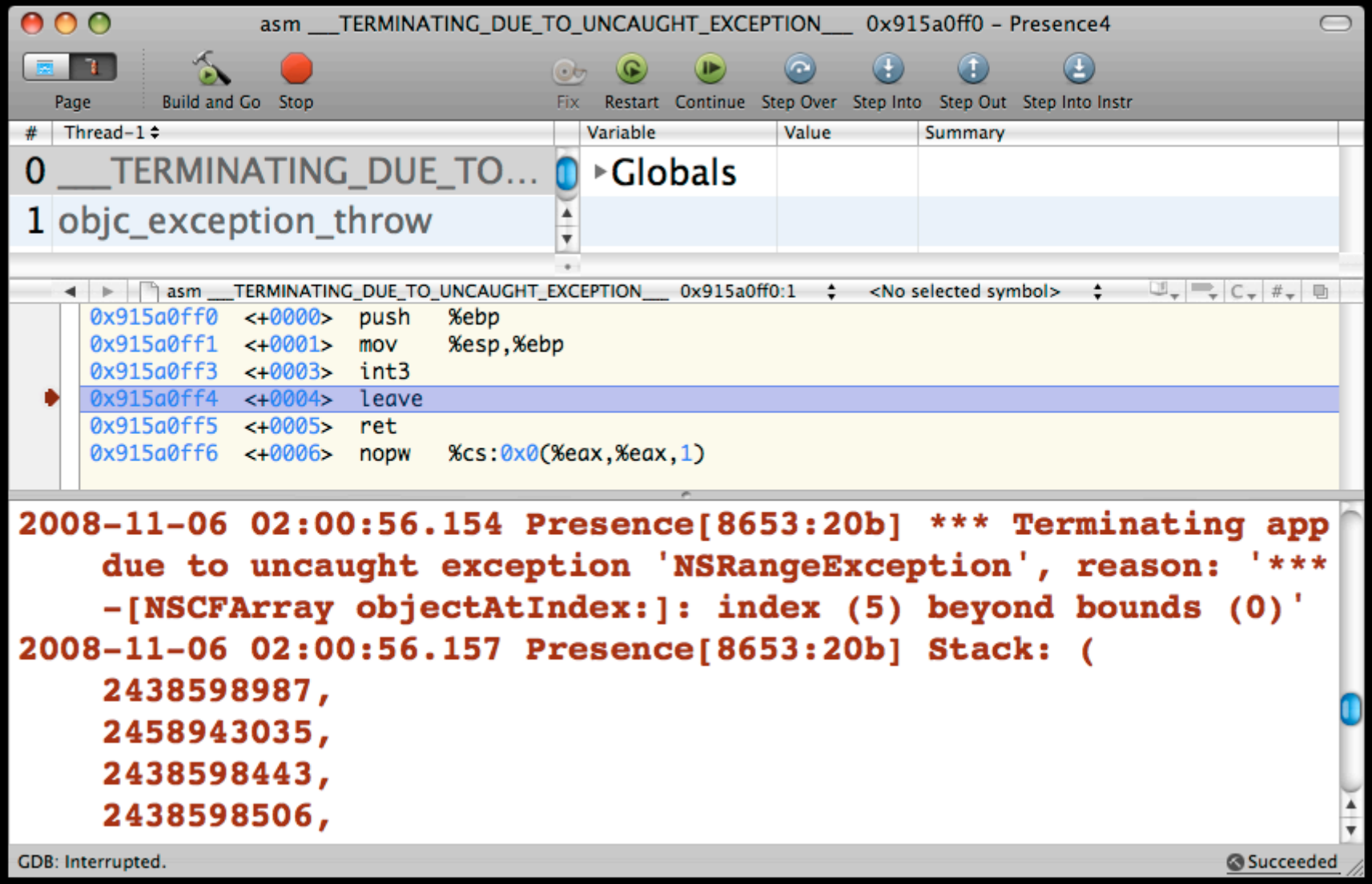
- **Not used for control flow** as in some other languages
- Intended for **truly "exceptional" conditions**
  - Programmer errors
  - An uncaught exception will **crash** a running program

Examples of raised exceptions

- Accessing outside the bounds of an array
- Calling undefined methods on an object
- Setting/Getting undefined key for an object

Debugging Exceptions

- If your program crashes suddenly, look in the Xcode console



Debugging Exceptions

- Console output will often help you find the source
- If not immediately identifiable, though, the crash point shown by default for an unhandled exception isn't of much use
- Add a breakpoint on **objc_exception_throw** to capture it

Demo:
Debugging Unhandled Exceptions

Searching

Searching in Your App

- Allow the user to sift through a large quantity of data
- Local or remote
  - Consider performance, responsiveness
- For large amounts of local data, pairs nicely with SQLite

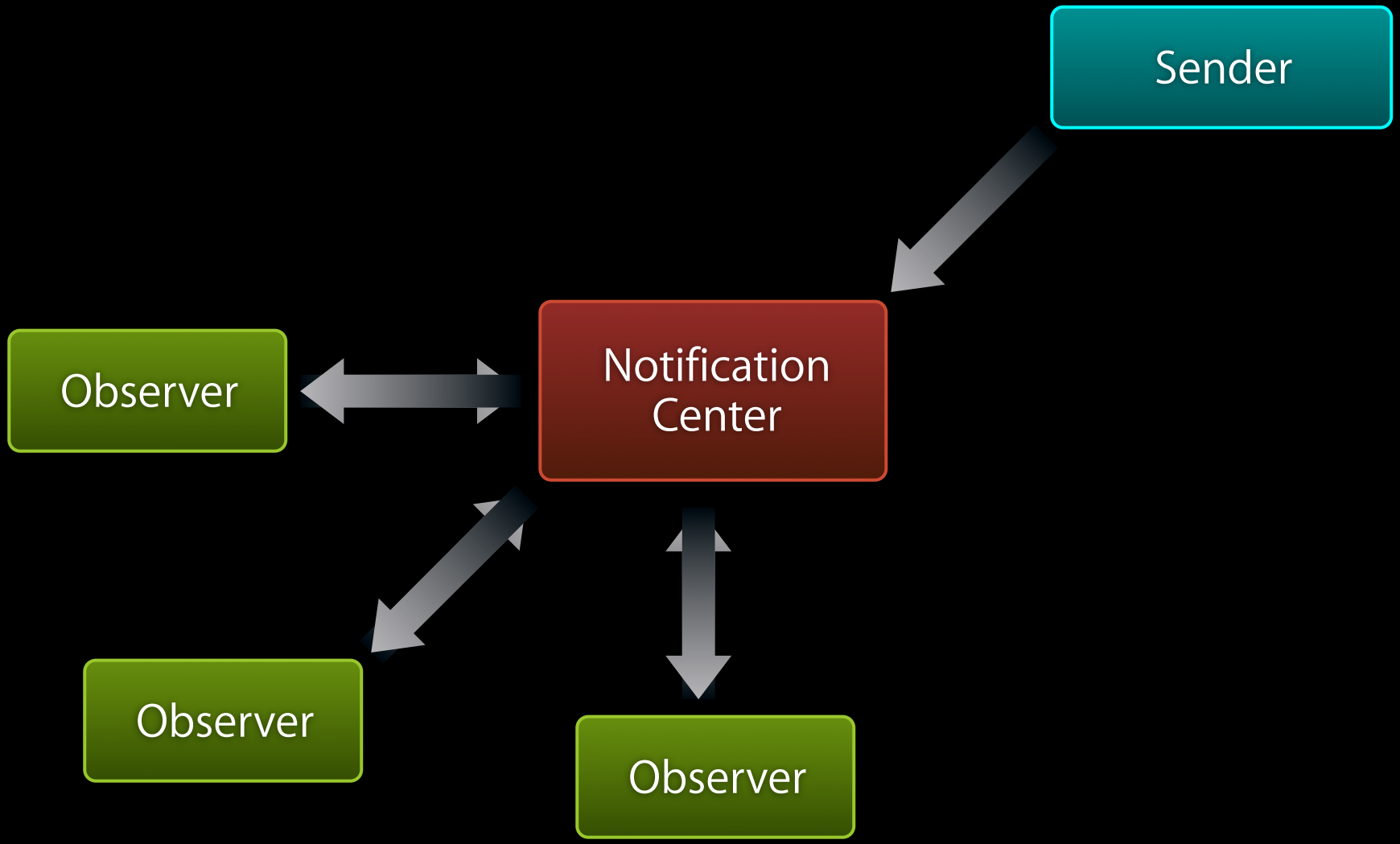
UISearchBar

- Interface element for entering a search query
- Used at the **top of the screen**
  - Often as a table header view
- Adopts some (but not all) text input traits
  - Keyboard automatically uses Search return key
- Bookmark button, cancel button
- Delegate callbacks when Search or other buttons pressed

Demo:
Searching with a Table View

Notifications

How Notifications Work



When to Use Notifications

- Potentially **many interested observers**
- Much more loosely coupled than delegation
  - Single, well-defined observer versus many unknown observers
- Flexible and powerful, but **can be too blunt**
  - Difficult to debug code that overuses notifications

Registering for a Notification

- Specific notification name sent by a specific object

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                     selector:@selector(fooHappened:)
                                     name:MyFooNotification
                                     object:foo];
```

- Specific notification name sent by **any** object

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                     selector:@selector(anyFooHappened:)
                                     name:MyFooNotification
                                     object:nil];
```

Receiving a Notification

- Callback method receives a single parameter
- Object contains notification name, object and user info
  - User info is dictionary of arbitrary key-value pairs

```
- (void)fooHappened:(NSNotification *)note
{
  NSLog(@“%@ from %@”, [note name], [note object]);

  NSDictionary *userInfo = [note userInfo];
  for (NSString *key in userInfo) {
    NSLog(@“%@ = %@”, key, [userInfo objectForKey:key];
  }
}
```

- Notifications are **received on the same thread where posted**

Unregistering for Notifications

- Unregister when no longer interested
- Especially **important when an object is deallocated**
  - Otherwise, notification center may message a deallocated object

```
- (void)dealloc
{
    // Stop listening to all notifications
    [[NSNotificationCenter defaultCenter] removeObserver:self];

    ...

    [super dealloc];
}
```

Posting a Notification

- Remember, don't rely on notifications too heavily
  - Learning which pattern to use will take practice

```
// In the header for this class
extern NSString *const MyFooNotification;

...

// In the implementation
NSString *const MyFooNotification = @"MyFooNotification";

- (void)foo
{
  [[NSNotificationCenter defaultCenter]
          postNotification:MyFooNotification
          object:self
          userInfo:[NSDictionary ...]];
}
```

Interesting Notifications in UIKit

- Application will terminate
- Significant time change
- Memory warnings
- Keyboard showing and hiding

Demo:
Keyboard Notifications

Key-Value Coding & Observing

aka: KVC/KVO

Key-Value Coding (KVC)

- Access object values
  - `NSString *name = person.name;`
  - `NSString *name = [person name];`
  - `NSString *name = [person valueForKey:@“name”];`

- Set object values:
  - `[person setName:@“Pee-Wee Herman”];`
  - `person.name = @“Pee-Wee Herman”;`
  - `[person setValue:@“Pee-Wee Herman” forKey:@“name”];`

Key-Value Coding (KVC)

- Get/set a value on an object by key (a string)
- First attempts to access via KVC-Compliant getters/setters
- If that fails, attempts to get to value directly

Key Paths

- Traverse objects using dot-separated keys
- Ex: @"person.address.street"
- Must use "keyPath" methods, instead of "key" methods to automatically parse the string

```
- (id)valueForKeyPath:(NSString *)keyPath;
```

```
- (void)setValue:(id)value forKeyPath:(NSString *)keyPath;
```

Accessing Undefined Keys

- What if you try to access a key that is undefined?
  - NSUndefinedKeyException

- But you can override!

```
-(id)valueForUndefinedKey:(NSString *)key;

-(void)setValue:(id)value forUndefinedKey:(NSString *)key;
```

Demo:
Key-Value Coding

Key-Value Observing (KVO)

- Listen for changes to an object's KVC-compliant values
- NSObject automatically broadcasts changes to observers
- No changes required to object being listened to

Key-Value Observing (KVO)

- To listen for changes:

```
-(void)addObserver:(NSObject *)anObserver
     forKeyPath:(NSString *)keyPath
     options:(NSKeyValueObservingOptions)options
     context:(void *)context;
```

- To stop listening for changes:

```
-(void)removeObserver:(NSObject *)anObserver
     forKeyPath:(NSString *)keyPath;
```

Key-Value Observing (KVO)

- Observing a value change:

```
-(void)observeValueForKeyPath:(NSString *)keyPath
      ofObject:(id)object
      change:(NSDictionary *)change
      context:(void *)context
      {
        if (context == myContext)
        {
          // Do something
        }
        else
        {
          // Don't forget to call super
        }
      }
```

Demo:
Key-Value Observing

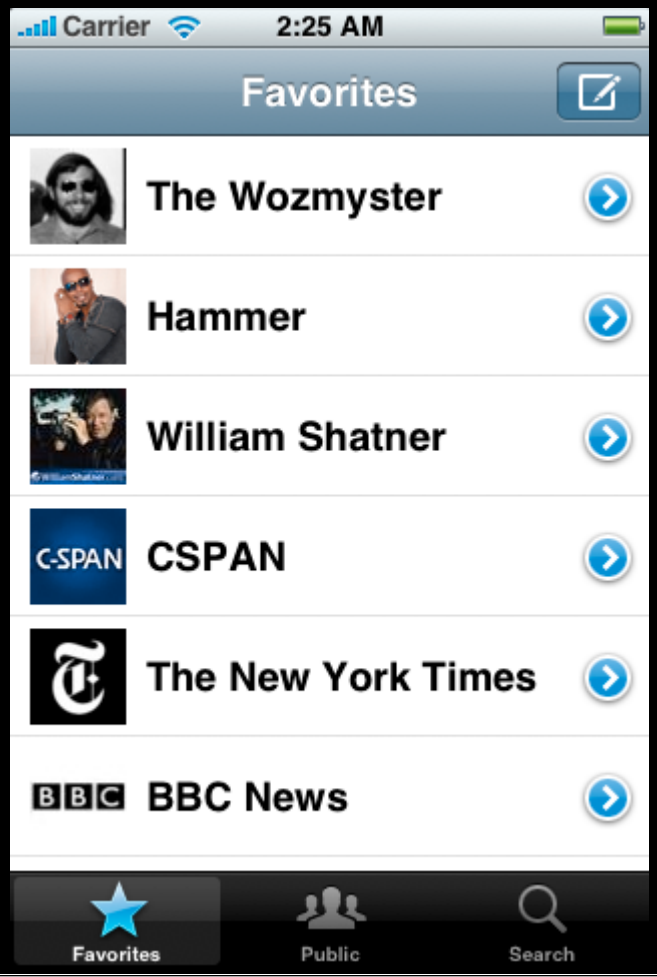
Documentation

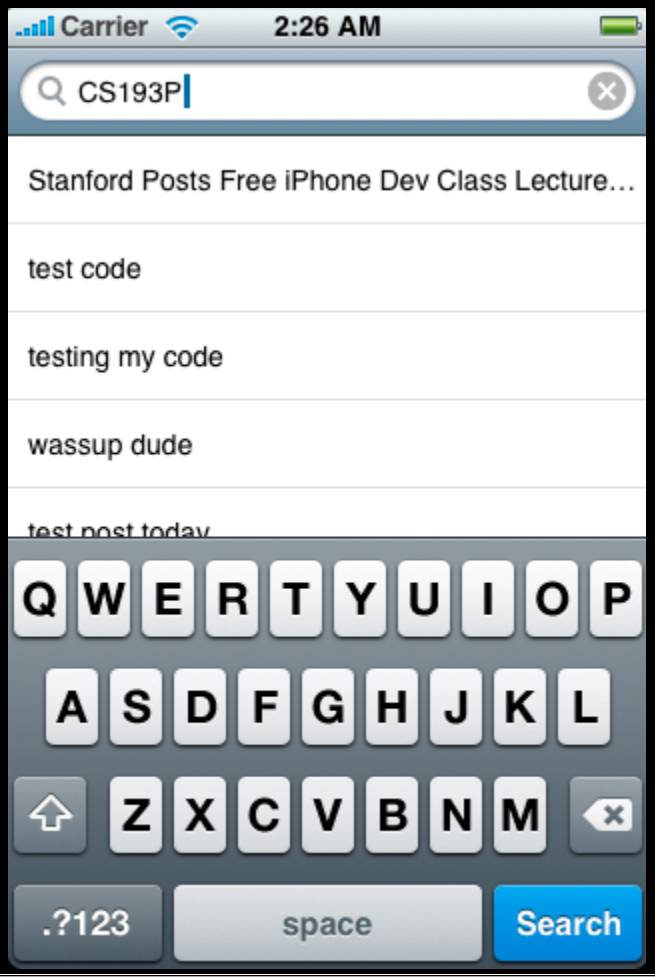
- KVC:
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/KeyValueCoding/
- KVO
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/KeyValueObserving/

Presence - Part 4

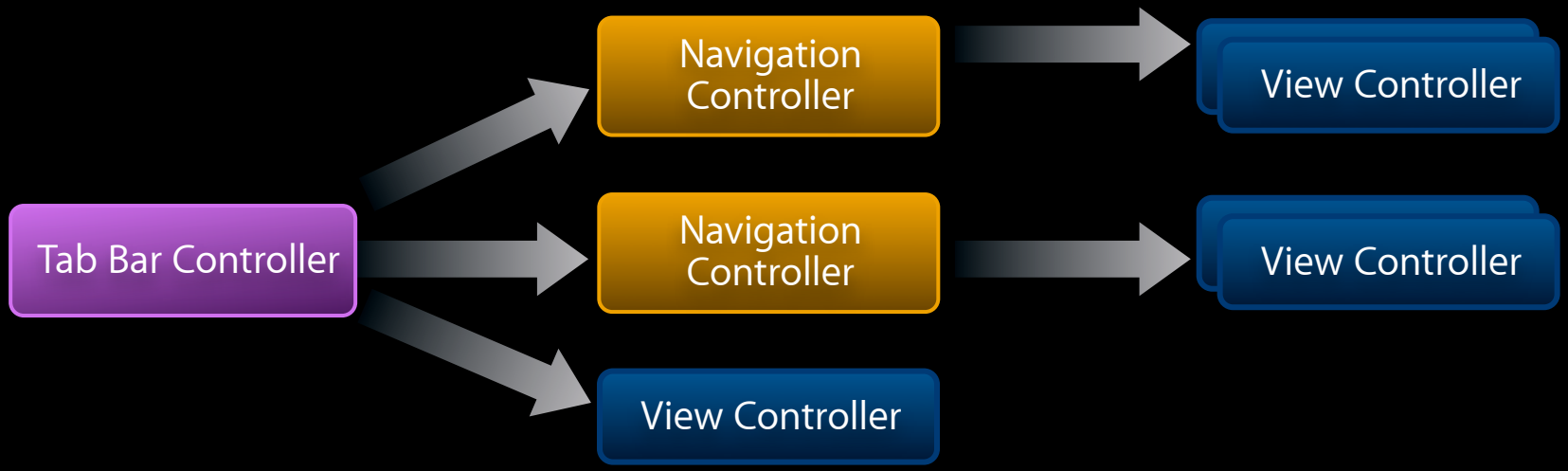
Presence 4 Goals

- Multiple modes with tab bar + navigation
- Address Book integration
- Search





Tab Bar + Navigation Controllers

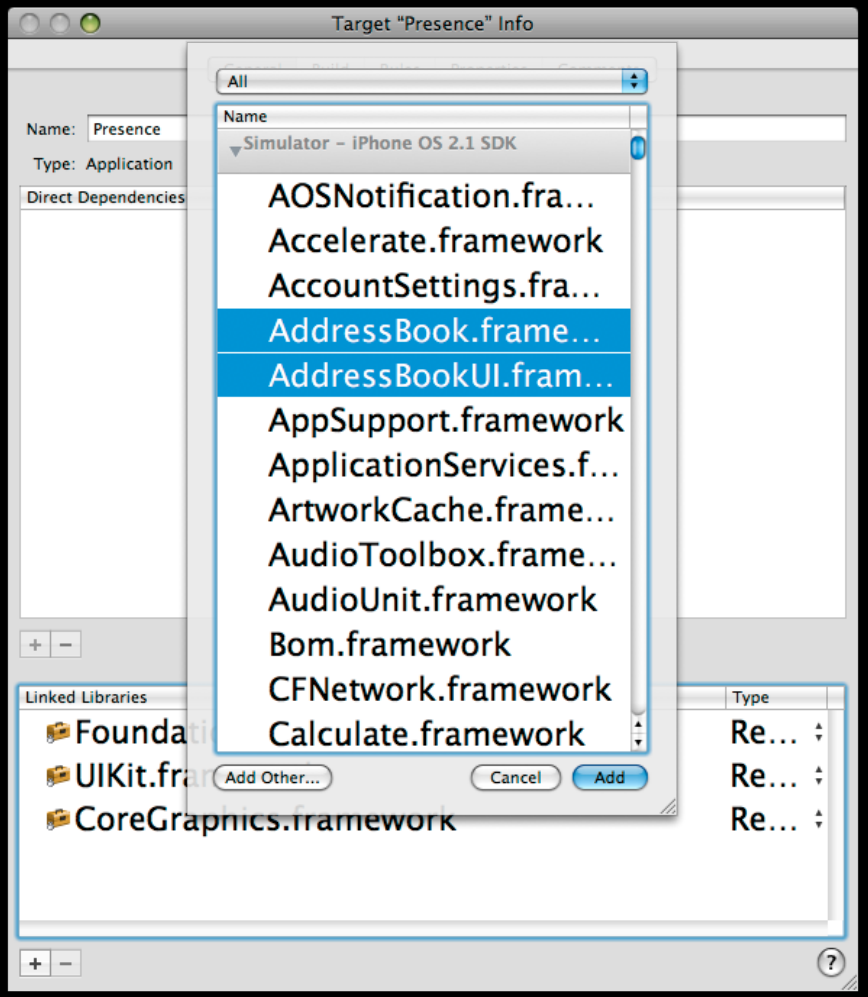


Demo:
Presence 4

Linking Against Address Book

Adding a Framework

- For Presence 4, you'll be using Address Book frameworks
- Your **application target** doesn't link against these by default



Demo:
Adding a Framework

Questions?