iPhone for .NET Developers

Ben Scheirman

Director of Development - ChaiONE

@subdigital

What you need

- A Mac
- Xcode
- iPhone SDK (limited to Simulator)
- iPhone Developer Program ($99 /year)

Objective-C

- Based on C
- Object Oriented
- Dynamic
- A little weird
- Powerful

Objective-C Primer

- Calling methods

Objective-C Primer

- Calling methods

```
[someObject someMethod];
```

Objective-C Primer

- Calling methods

```
[someObject someMethod];
```

```
[someObject someMethodWithInput:5];
```

Objective-C Primer

- Calling methods

```
[someObject someMethod];
```

```
[someObject someMethodWithInput:5];
```

```
[dictionary setObject:obj
                forKey:key];
```

Objective-C Primer

- Nesting method calls

Objective-C Primer

- Nesting method calls

```
[NSString stringWithFormat:
    [prefs format]];
```

Objective-C Primer

- Instantiating classes

Objective-C Primer

- Instantiating classes

```
UIView *view = [[UIView alloc] init];
```

Objective-C Primer

- Instantiating classes

```
UIView *view = [[UIView alloc] init];
```

```
NSDate *date = [NSDate date];
```

Objective-C Primer

- Defining Classes

Objective-C Primer

- Defining Classes

```
//Person.h
@interface Person {
  //instance variables
}

//properties & methods

@end
```

Objective-C Primer

- Defining Classes

Objective-C Primer

- Defining Classes

```
//Person.m
#import "Person.h"

@implementation Person

//implement properties & methods

@end
```

Objective-C Primer

- Defining Methods

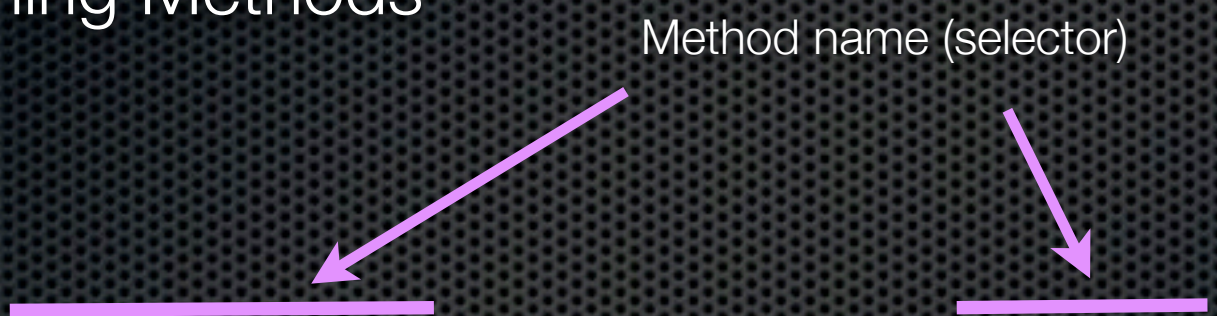
Objective-C Primer

- Defining Methods

```
-(void)showLoadingText:(NSString *)text animated:(BOOL)animated;
```

Objective-C Primer

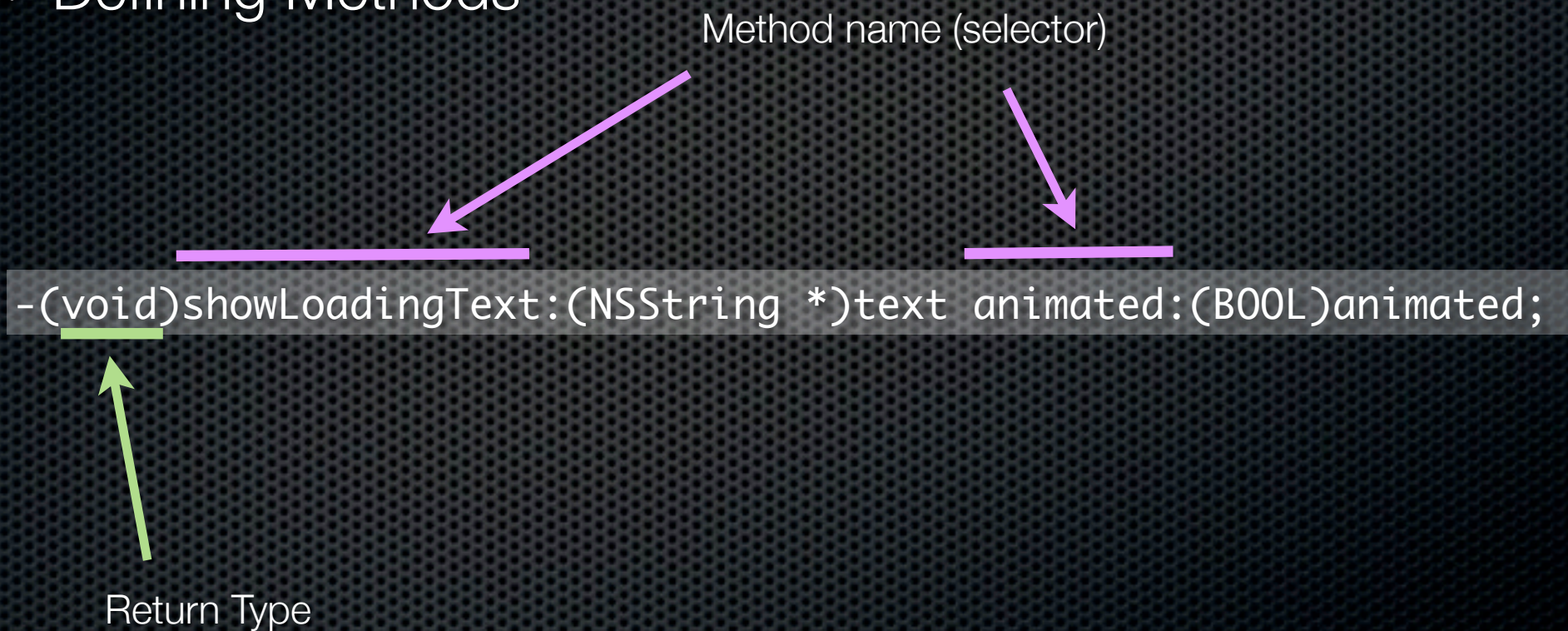
- Defining Methods




```
-(void)showLoadingText:(NSString *)text animated:(BOOL)animated;
```

Objective-C Primer

- Defining Methods

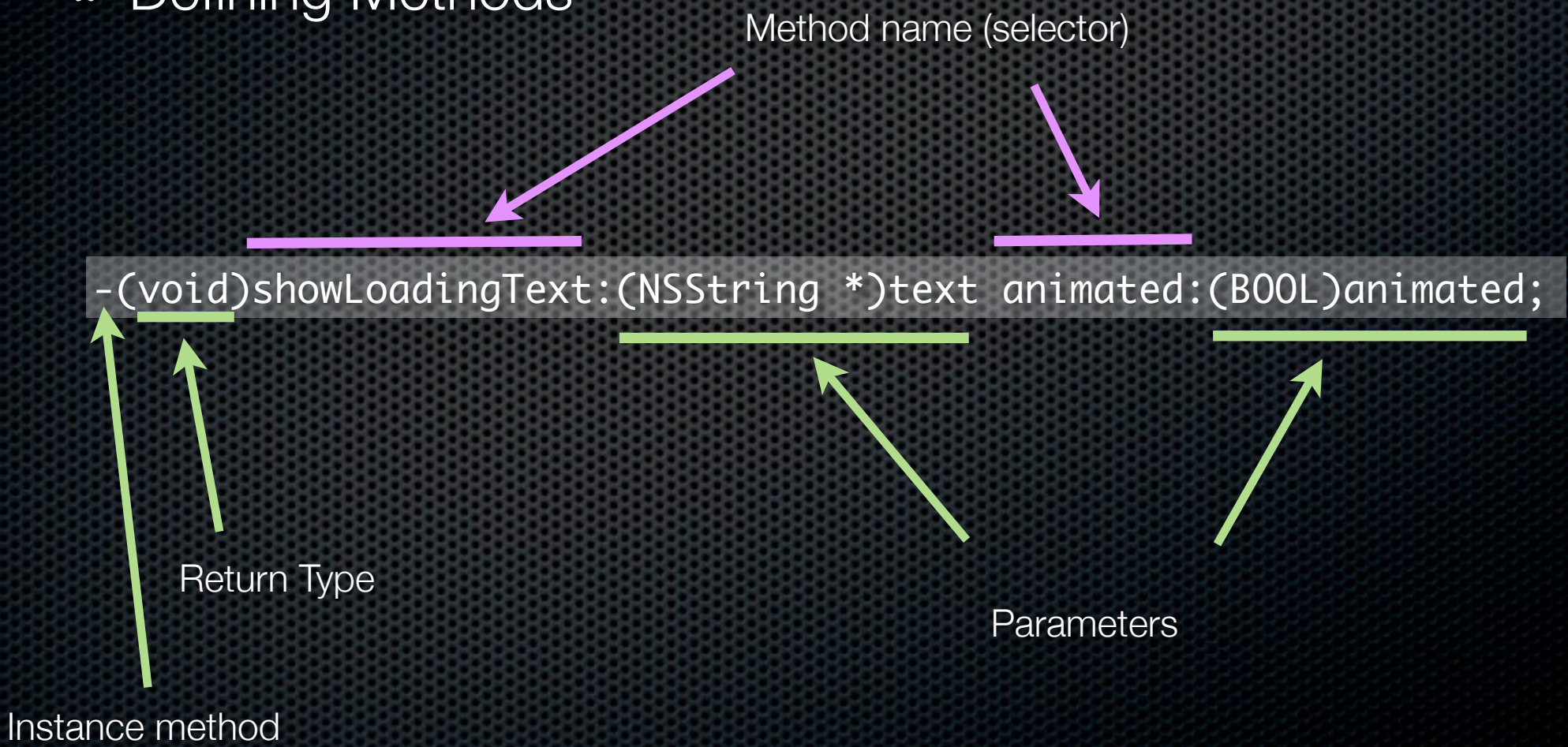


Objective-C Primer

- Defining Methods

Objective-C Primer

- Defining Methods



Memory Management

- No garbage collection on the iPhone
- Retain / Release

Memory Management

- Retain / Release Dance

Memory Management

- Retain / Release Dance

```
Foo *foo = [[Foo alloc] init];
```

1

Memory Management

- Retain / Release Dance

```
Foo *foo = [[Foo alloc] init];
```

1

```
[foo retain];
```

2

Memory Management

- Retain / Release Dance

```
Foo *foo = [[Foo alloc] init];
```

1

```
[foo retain];
```

2

```
[foo release];
```

1

Memory Management

- Retain / Release Dance

```
Foo *foo = [[Foo alloc] init];
```

1

```
[foo retain];
```

2

```
[foo release];
```

1

```
[foo release];
```

0

Memory Management

- Retain / Release Dance

```
Foo *foo = [[Foo alloc] init];
```
1

```
[foo retain];
```
2

```
[foo release];
```
1

```
[foo release];
```
foo is deallocated

0

Getters / Setters

Getters / Setters

```
[foo setBar:@"baz"];
```

Getters / Setters

```
[foo setBar:@"baz"];
```

```
[foo bar]; //returns @"baz"
```

Getters / Setters

```
[foo setBar:@"baz"];
```

```
[foo bar]; //returns @"baz"
```

```
foo.bar = @"gruul";
```

Getters / Setters

```
[foo setBar:@"baz"];
```

```
[foo bar]; //returns @"baz"
```

```
foo.bar = @"gruul";
```

```
foo.bar //returns @"gruul"
```

Implementing setters

Implementing setters

```
-(void)setBar:(id)value {
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
    [bar release];
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
    [bar release];
    bar = nil;
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
    [bar release];
    bar = nil;
  }
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
     [bar release];
     bar = nil;
  }
  if(value != nil)
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
    [bar release];
    bar = nil;
  }
  if(value != nil)
    bar = [value retain];
```

Implementing setters

```
-(void)setBar:(id)value {
  if(bar == value) return;
  if(bar != nil) {
    [bar release];
    bar = nil;
  }
  if(value != nil)
    bar = [value retain];
}
```

No Thanks

Properties

Properties

```
//Foo.h
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;

-(void)dealloc {
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;

-(void)dealloc {

  [image release];
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;

-(void)dealloc {

  [image release];
  [message release];
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;

-(void)dealloc {

  [image release];
  [message release];

  [super dealloc];
```

Properties

```
//Foo.h
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, copy) NSString *message;



//Foo.m
@synthesize image, message;

-(void)dealloc {

  [image release];
  [message release];

  [super dealloc];
}
```

Dot Syntax Dogma

Use dot syntax if you like it

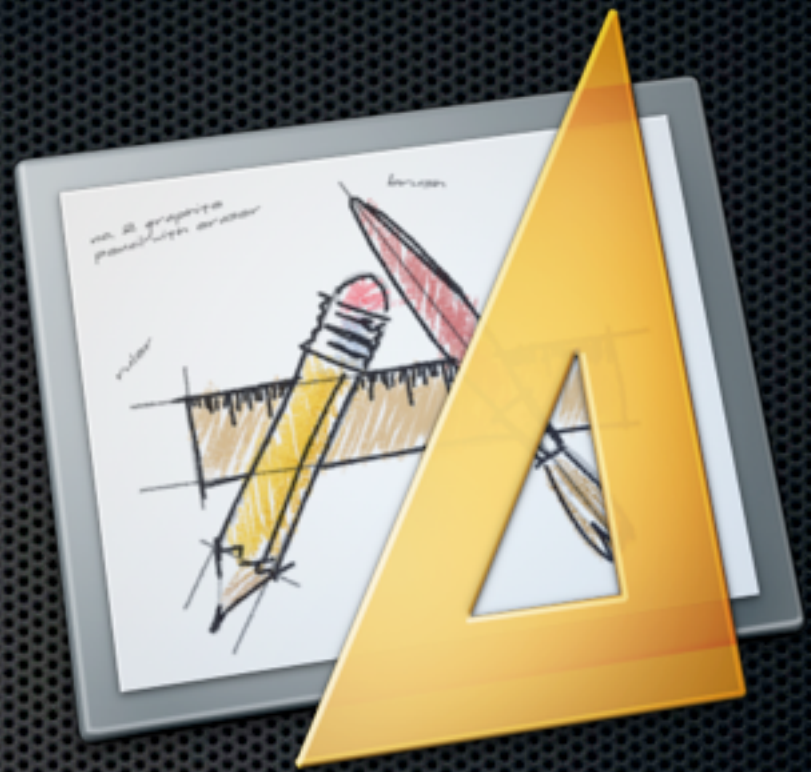
Just be aware of what it's hiding

Xcode

- Your IDE
- Code completion
- Interactive Debugger

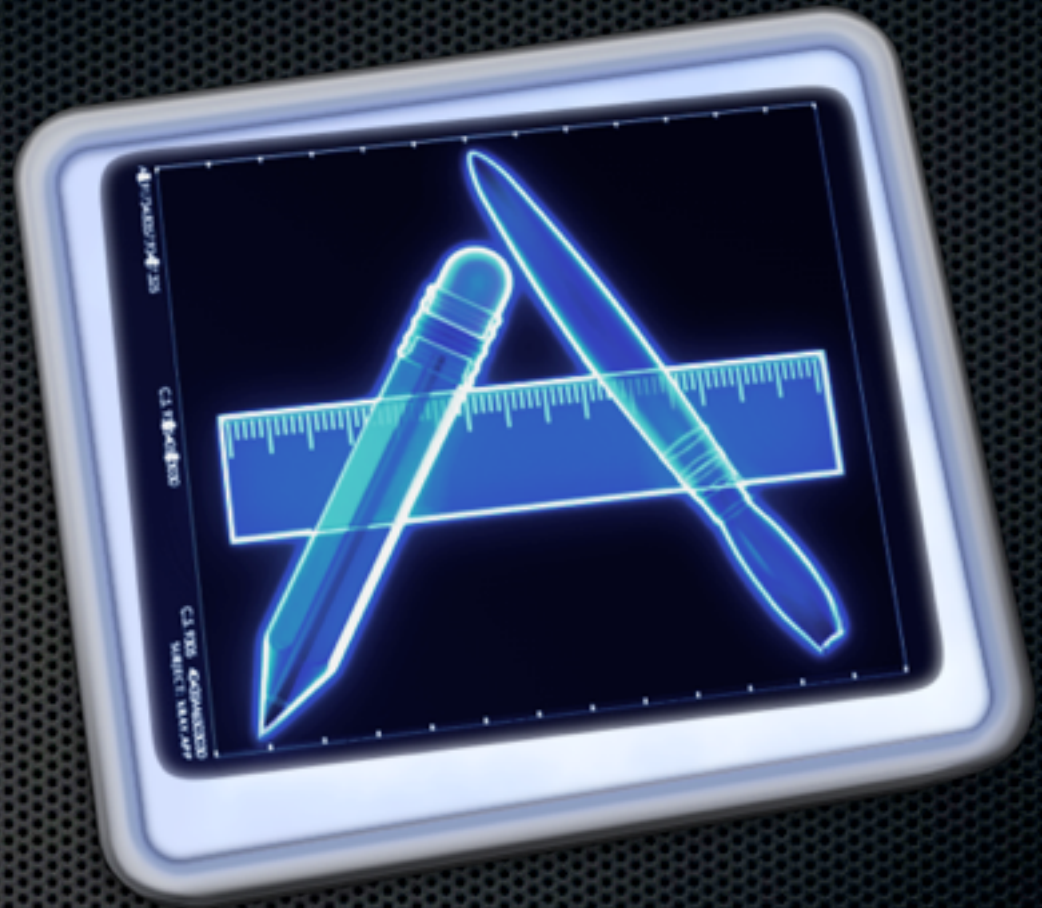
- Lacks good refactoring tools

Interface Builder



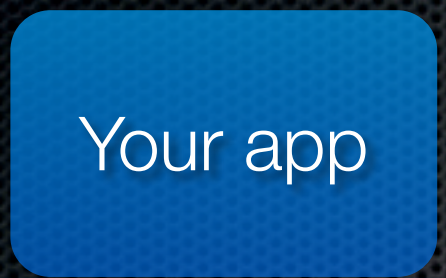
- Drag-n-drop UI building
- Layouts are defined in XIBs (XML representation). Usually called "Nibs"
- "Make connections" with classes defined in Xcode
  - variables --> UI components
  - UI events --> methods

Instruments



- Find Memory Leaks
- Analyze Memory Usage
- Track down slow code

iOS SDK


Your app


UIKit


CoreFoundation

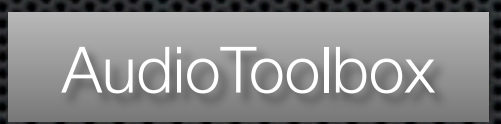

CoreGraphics

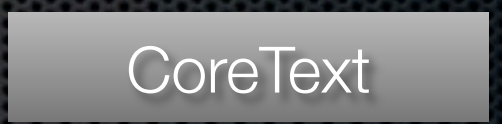

Accelerate

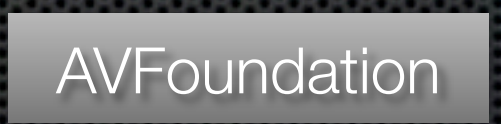

CoreMotion

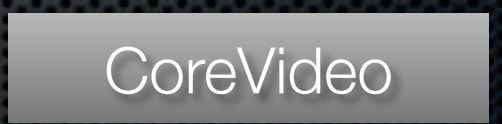

AddressBook


CoreTelephony


AudioToolbox


CoreText


AVFoundation


CoreVideo


CoreAudio


GameKit


CoreData


iAd

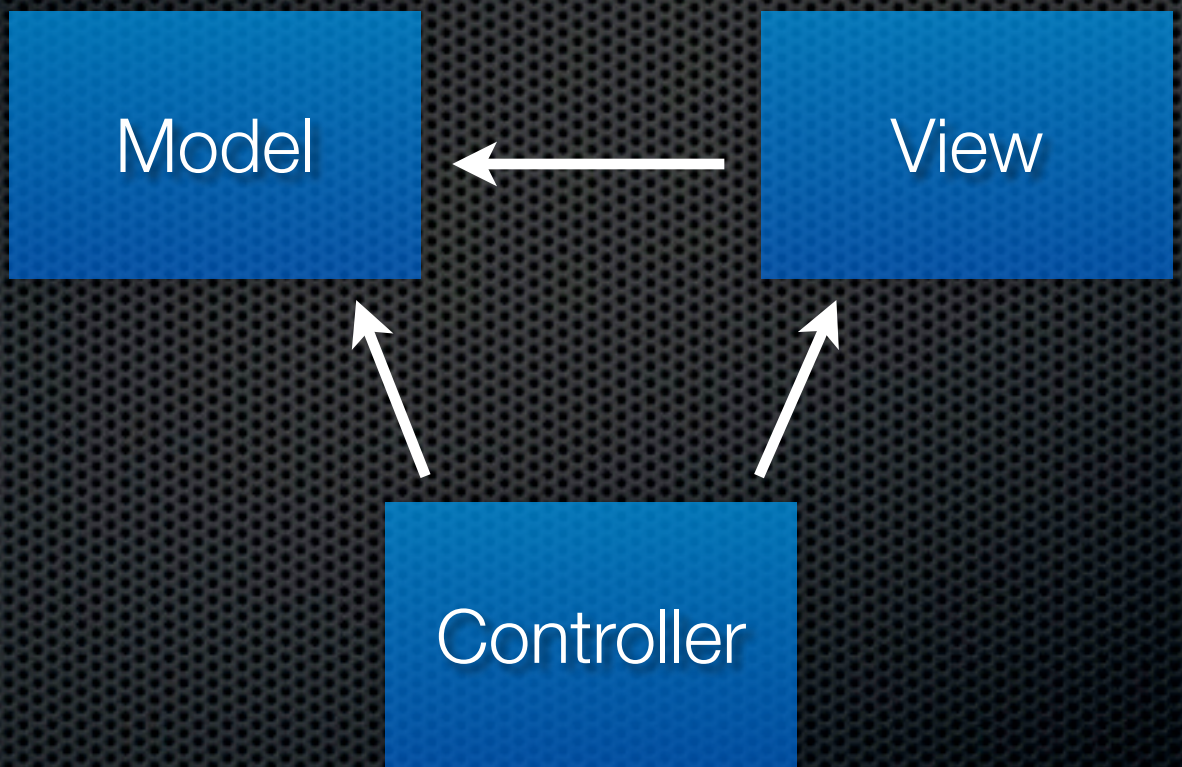

CoreLocation


MapKit


CFNetwork


StoreKit

Model View Controller



The View Controller

- Handles setup logic for a screen
- Handles user input
- Interacts with the model
- Contains 1 or more views

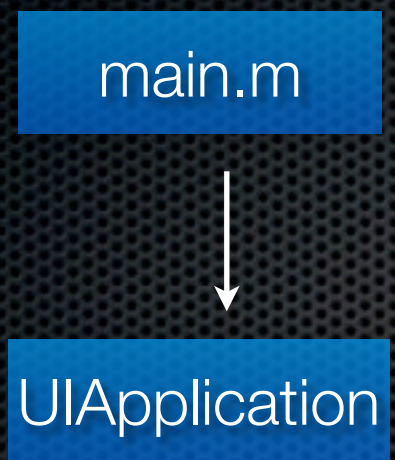
The View

- Visual representation
- Drawing
- Laying out subviews (autorotation)
- May Handle touch events

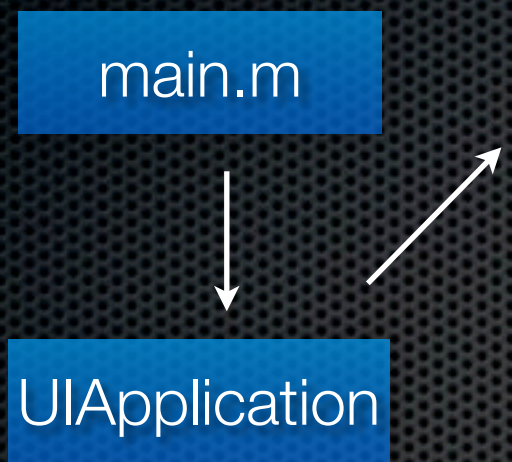
Lifecycle of an App

main.m

Lifecycle of an App

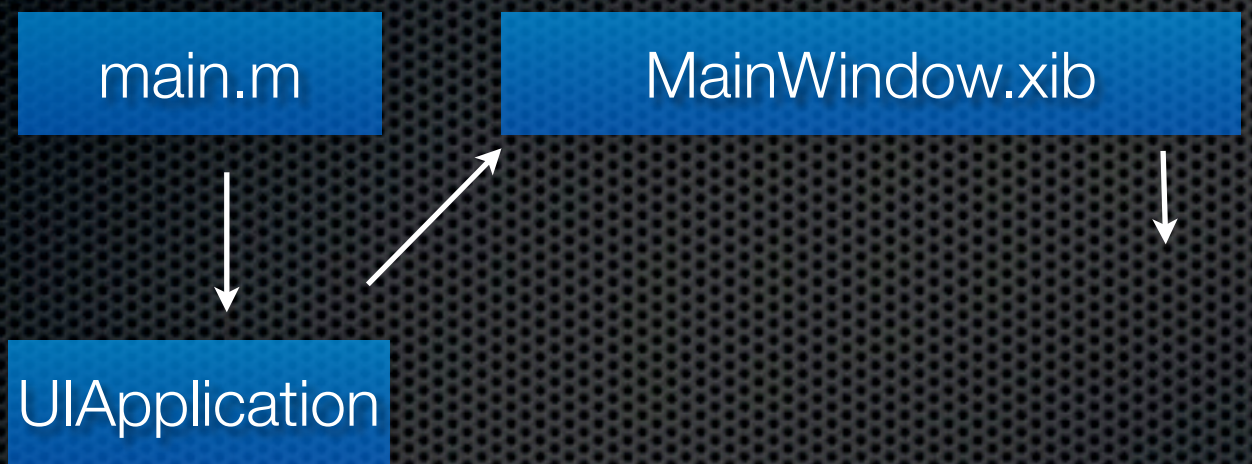


Lifecycle of an App

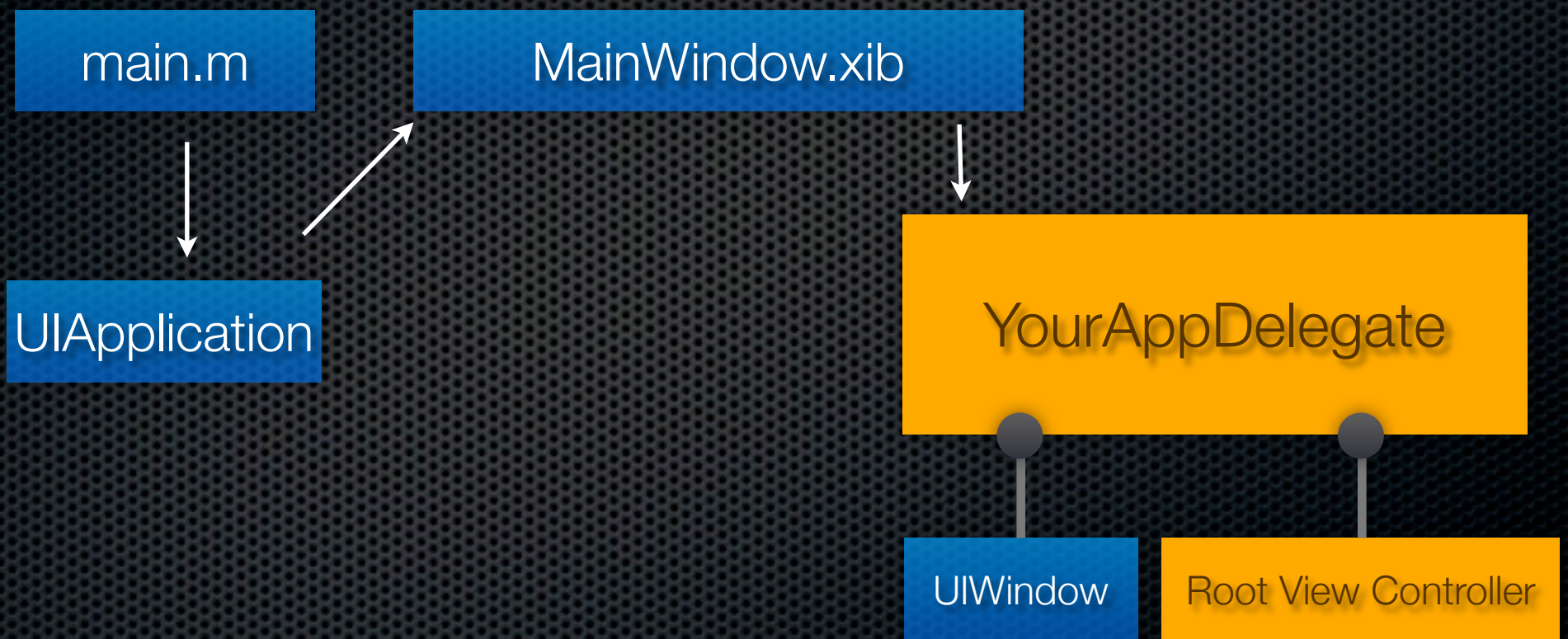


Lifecycle of an App

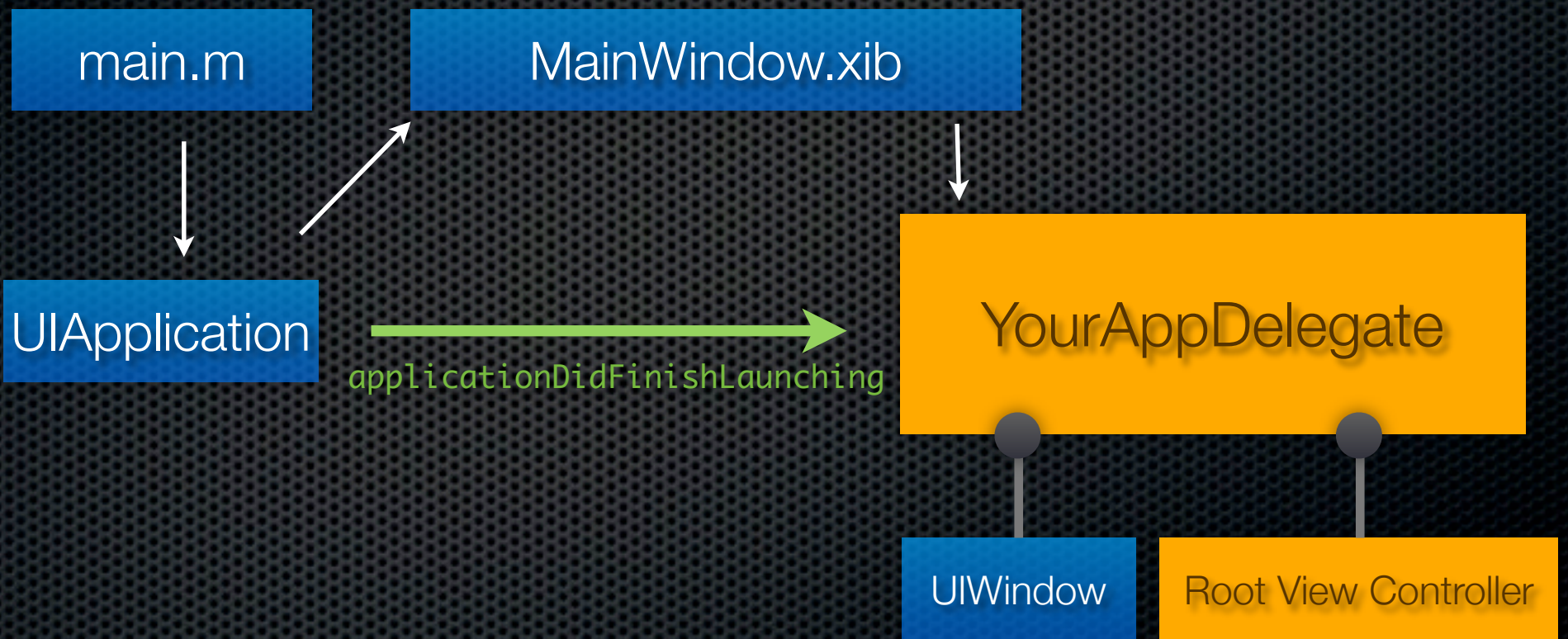
Lifecycle of an App



Lifecycle of an App



Lifecycle of an App



Time to code!