iOS Development

Adam May
Sam Kirchmeier

Goals

1. Learn iOS
2. Be ambitious
3. Build an app

Build an app

Connects to a web service
Displays hierarchical information
Has custom views
Has custom animations
Uses object persistence
Uses concurrency

PreciouStatus

NY Times
http://developer.nytimes.com/docs

Goodreads
http://www.goodreads.com/api

Instagram
http://instagram.com/developer

Schedule

Tuesdays

6-9 p.m.

Final class is December 17th

iOS TC Hack

Here every other Wednesday

7-9 p.m.

11/13, 11/27, 12/11

Topics

Week 1: Objective-C
Week 2: Cocoa Touch
Week 3: Graphics
Week 4: Networking
Week 5: Performance
Week 6: Deployment

Class Structure

Introductions

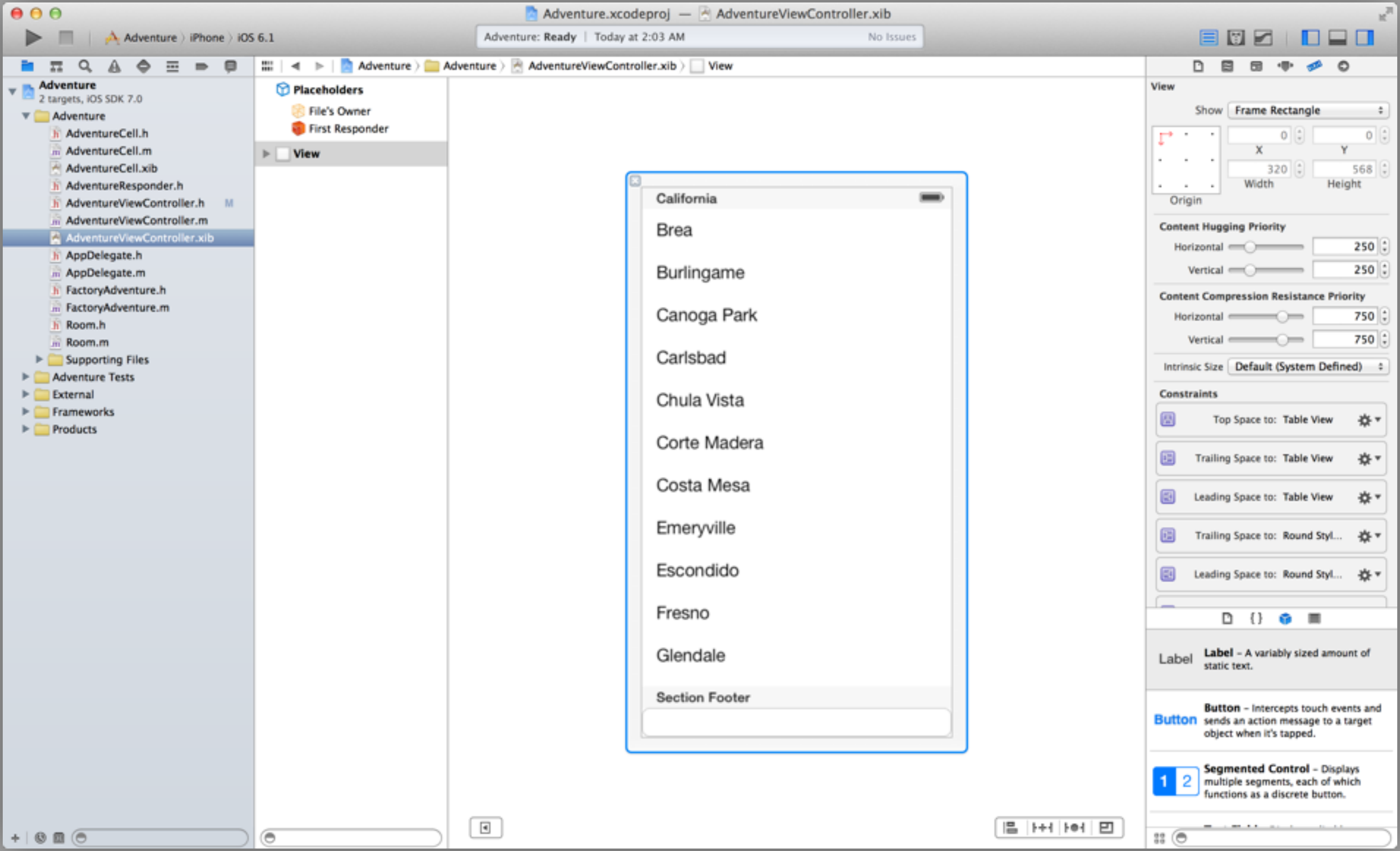
Today

Toolset Overview

Crash Course in Objective-C

Build an Old-School Adventure Game

Toolset


Adventure.xcodeproj — AdventureViewController.xib
Adventure › iPhone › iOS 6.1
Adventure: Ready | Today at 2:03 AM
No Issues
Adventure › Adventure › AdventureViewController.xib › View
Adventure
2 targets, iOS SDK 7.0
Adventure
AdventureCell.h
AdventureCell.m
AdventureCell.xib
AdventureResponder.h
AdventureViewController.h
M
AdventureViewController.m
AdventureViewController.xib
AppDelegate.h
AppDelegate.m
FactoryAdventure.h
FactoryAdventure.m
Room.h
Room.m
Supporting Files
Adventure Tests
External
Frameworks
Products
Placeholders
File's Owner
First Responder
View
California
Brea
Burlingame
Canoga Park
Carlsbad
Chula Vista
Corte Madera
Costa Mesa
Emeryville
Escondido
Fresno
Glendale
Section Footer
View
Origin
Label – A variably sized amount of static text.
Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control – Displays multiple segments, each of which functions as a discrete button.

View

Show: Frame Rectangle

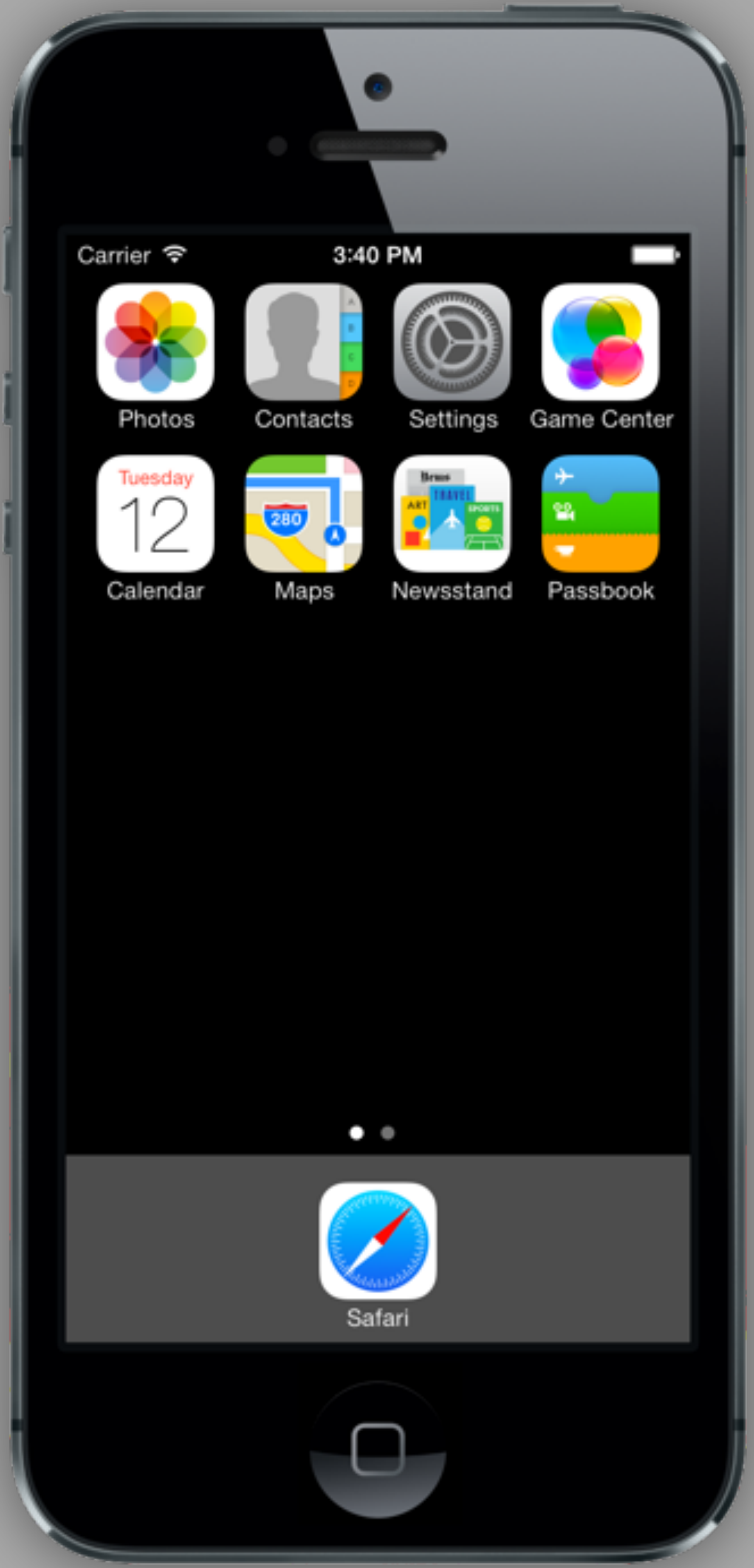
X: 0
Y: 0
Width: 320
Height: 568

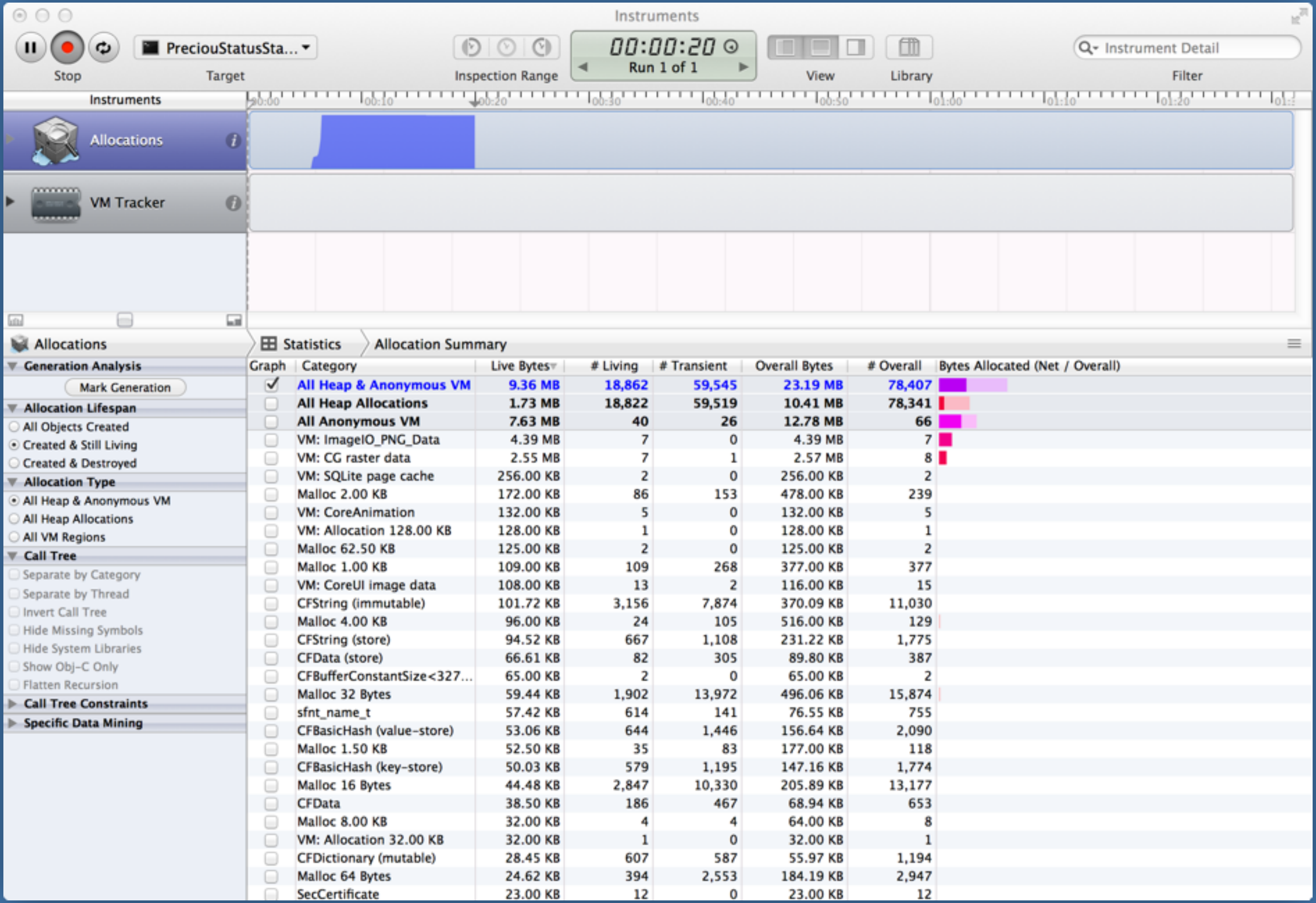
Content Hugging Priority
Horizontal: 250
Vertical: 250

Content Compression Resistance Priority
Horizontal: 750
Vertical: 750

Intrinsic Size: Default (System Defined)

Constraints
Top Space to: Table View
Trailing Space to: Table View
Leading Space to: Table View
Trailing Space to: Round Styl...
Leading Space to: Round Styl...


Carrier
3:40 PM
Photos
Contacts
Settings
Game Center
Tuesday
12
Calendar
Maps
Newsstand
Passbook
Safari



Allocations

Generation Analysis

Mark Generation

Allocation Lifespan

- All Objects Created
- [✓ Created & Still Living]
- Created & Destroyed

Allocation Type

- [✓ All Heap & Anonymous VM]
- All Heap Allocations
- All VM Regions

Call Tree

- Separate by Category
- Separate by Thread
- Invert Call Tree
- Hide Missing Symbols
- Hide System Libraries
- Show Obj-C Only
- Flatten Recursion

Call Tree Constraints

Specific Data Mining

Graph	Category	Live Bytes	# Living	# Transient	Overall Bytes	# Overall	Bytes Allocated (Net / Overall)
✓	All Heap & Anonymous VM	9.36 MB	18,862	59,545	23.19 MB	78,407	
	All Heap Allocations	1.73 MB	18,822	59,519	10.41 MB	78,341	
	All Anonymous VM	7.63 MB	40	26	12.78 MB	66	
	VM: ImageIO_PNG_Data	4.39 MB	7	0	4.39 MB	7	
	VM: CG raster data	2.55 MB	7	1	2.57 MB	8	
	VM: SQLite page cache	256.00 KB	2	0	256.00 KB	2	
	Malloc 2.00 KB	172.00 KB	86	153	478.00 KB	239	
	VM: CoreAnimation	132.00 KB	5	0	132.00 KB	5	
	VM: Allocation 128.00 KB	128.00 KB	1	0	128.00 KB	1	
	Malloc 62.50 KB	125.00 KB	2	0	125.00 KB	2	
	Malloc 1.00 KB	109.00 KB	109	268	377.00 KB	377	
	VM: CoreUI image data	108.00 KB	13	2	116.00 KB	15	
	CFString (immutable)	101.72 KB	3,156	7,874	370.09 KB	11,030	
	Malloc 4.00 KB	96.00 KB	24	105	516.00 KB	129	
	CFString (store)	94.52 KB	667	1,108	231.22 KB	1,775	
	CFData (store)	66.61 KB	82	305	89.80 KB	387	
	CFBufferConstantSize<327...	65.00 KB	2	0	65.00 KB	2	
	Malloc 32 Bytes	59.44 KB	1,902	13,972	496.06 KB	15,874	
	sfnt_name_t	57.42 KB	614	141	76.55 KB	755	
	CFBasicHash (value-store)	53.06 KB	644	1,446	156.64 KB	2,090	
	Malloc 1.50 KB	52.50 KB	35	83	177.00 KB	118	
	CFBasicHash (key-store)	50.03 KB	579	1,195	147.16 KB	1,774	
	Malloc 16 Bytes	44.48 KB	2,847	10,330	205.89 KB	13,177	
	CFData	38.50 KB	186	467	68.94 KB	653	
	Malloc 8.00 KB	32.00 KB	4	4	64.00 KB	8	
	VM: Allocation 32.00 KB	32.00 KB	1	0	32.00 KB	1	
	CFDictionary (mutable)	28.45 KB	607	587	55.97 KB	1,194	
	Malloc 64 Bytes	24.62 KB	394	2,553	184.19 KB	2,947	
	SecCertificate	23.00 KB	12	0	23.00 KB	12	

Objective-C

History

Why Objective-C?

It’s Fast

Object-oriented

Based on C

Dynamism… is that a word?

Apple

Objective-C is a strict superset of C.

Code you write in C just works.

```
int c = 1 + 2;
```

```c
void SFReallyFastStuff()
{
    // Highly optimized C goes here
}

SFReallyFastStuff();
```

Data Types

Primitives

Objects

Primitives

Objects

Integer

char
short
int
long
long long

Floating Point

float
double

```
typedef enum {
  SFColorRed,
  SFColorBlack = 10,
} SFColors;
```

```
typedef struct {
    int units;
    double prices[10];
} SFOrder;
```

```
BOOL shouldWeUseBools = YES;
BOOL c = NO;
```

```
NSInteger a = -42;
NSUInteger b = 42;
```

NSLog

Coding Break

```
[anArray count];
```

```
[anArray objectAtIndex:3];
```

```
[anArray objectAtIndex:3 inRange:range];
```

```
[anArray objectAtIndex:3 inRange:range];

anArray.getObject(3, range);
```

```
[anArray indexOfObject:object inSortedRange:range
options:NSBinarySearchingFirstEqual
usingComparator:comparator];
```

```
[anArray indexOfObject:object inSortedRange:range
options:NSBinarySearchingFirstEqual
usingComparator:comparator];

[anArray indexOfObject:object
         inSortedRange:range
               options:NSBinarySearchingFirstEqual
       usingComparator:comparator];
```

```
[[anArray lastObject] description];
```

```
NSUInteger count = [anArray count];
```

Creating Objects

```objc
NSArray *anArray = [[NSArray alloc] init];
NSArray *anArray = [NSArray array];

NSArray *anotherArray =
    [NSArray arrayWithObject:one];

NSArray *yetAnotherArray =
    [NSArray arrayWithObjects:one, two, three, nil];
```

```
int justAnIntValue;      // Often used
int *aPointerToAnInt;    // Rarely used

NSArray *anArray;        // Always used
NSArray anArray;         // Compiler error
```

Variables

When declaring a local primitive variable, it’s uninitialized.

```
int value;           // Yikes!
```

Best off initializing it, like so:

```
int value = 3;       // OK
```

Pointers are always automatically initialized to nil.

```
NSArray *anArray; // OK
```

nil

`NO` for BOOL return types

`nil` for object return types

`0` for numeric return types

```
NSArray *anArray; // Initialized to nil
NSUInteger count = [anArray count];
// Value of count is 0
```

`nil` is false

```
if (anArray != nil)
{
   id lastObject = [anArray lastObject];
}

if (anArray)
{
   id lastObject = [anArray lastObject];
}
```

```
if ([anArray count] == 0)
{
  // anArray might be nil, might not be
}
```

NSString

```
char *basicString = ”Could I have an array of
chars with a null at the end, please?”;
```

```
NSString *objcString = [[NSString alloc]
   initWithCString:"Awesome string"
   encoding:NSUTF8StringEncoding];
```

Literals

```
NSString *emptyString = [[NSString alloc] init];
NSString *anotherEmptyString = @"";
```

NSNumber

An Objective-C wrapper for scalar C types

```
NSNumber *satScore =
   [[NSNumber alloc] initWithInt:2400];

NSNumber *gpa =
   [[NSNumber alloc] initWithDouble:4.0];

NSNumber *smartyPants =
   [[NSNumber alloc] initWithBool:YES];
```

```
NSNumber *satScore =
   [NSNumber numberWithInt:2400];

NSNumber *gpa =
   [NSNumber numberWithDouble:4.0];

NSNumber *smartyPants =
   [NSNumber numberWithBool:YES];
```

```
NSNumber *satScore = @2400;
NSNumber *gpa = @4.0;
NSNumber *smartyPants = @YES;
```

```
int scalarSatScore = [satScore intValue];
double scalarGpa = [gpa doubleValue];
BOOL scalarSmartyPants = [smartyPants boolValue];
```

```
NSNumber *zero = @0;

if (zero)
{
  // Will this code get executed?
}
else
{
  // Or will this code?
}
```

```
NSNumber *zero = @0;

if ([zero boolValue])
{
    // Will this code get executed?
}
else
{
    // Or will this code?
}
```

NSValue

```
typedef struct {
   double radius;
   double x;
   double y;
} SFCircle;

struct SFCircle aCircle;
aCircle.radius = 10.0;
aCircle.x = 5.0;
aCircle.y = 15.0;

NSValue *circleValue =
    [NSValue value:&aCircle
      withObjCType:@encode(SFCircle)];
```

Collections

NSArray

NSDictionary

NSArray

```
NSArray *subjects =
   [NSArray arrayWithObjects:@"English",
     @"Science", @"Math", nil];
```

```
NSNumber *aNumber = @3;
NSString *aString = @"Three";

NSArray *anArray =
   [NSArray arrayWithObjects:aNumber,
      aString, nil];

NSString *threeAsString =
   [anArray objectWithIndex:3];
```

```
NSArray *subjects =
   @[@”English”, @”Science”, @”Math”];
```

NSDictionary

```
NSDictionary *favoriteColors =
   [NSDictionary dictionaryWithObjectsAndKeys:
      @"Blue", @"Sam",
      @"Green", @"Adam",
      nil];

NSString *samsFavoriteColor =
   [favoriteColors objectForKey:@"Sam"];
```

```
NSDictionary *favoriteColors = @{
    @”Sam” : @”Blue”
   @”Adam” : @”Green”
};
```

Enumeration

```
for (int i = 0; i < anArray.length; i++)
{
   NSString *string = [anArray objectAtIndex:i];
}

for (NSString *string in anArray)
{
   // Optimized
}
```

Subscripting

```
NSArray *rooms = @[@"Office", @"...", @"..."];
NSString *secondRoom = rooms[1];
```

```
NSDictionary *favoriteColors = @{
    @"Sam" : @"Blue"
   @"Adam" : @"Green"
};

NSString *samsFavorite = favoriteColors[@"Sam"];
```

Coding Break

Mutability

NSMutableString
NSMutableArray
NSMutableDictionary
NSMutableSet

Why?

Performance

Limit the scope of mutability

```
NSMutableSet *breweries =
   [NSMutableSet setWithCapacity:3];

[breweries addObject:@"Fulton"];
[breweries addObject:@"Surly"];
[breweries addObject:@"Indeed"];

[breweries removeObject:@"Indeed"];

NSSet *immutableBreweries = [breweries copy];
NSMutableSet *mutableBreweries =
   [immutableBreweries mutableCopy];

NSSet *immutableBreweries =
   [NSSet setWithSet:breweries];
NSMutableSet *mutableBreweries =
   [NSMutableSet setWithSet:immutableBreweries];
```

Equality

`isEqual:` vs. `==`

Adventure

https://github.com/gosmartfactory/ios

Classes

.h

Header
Public Interface

.m

Implementation
Private Interface

Room.h

```
@interface Room : NSObject



@end
```

Room.h

```
@interface Room : NSObject

@property NSString *name;



@end
```

Room.h

```
@interface Room : NSObject

@property (nonatomic, strong) NSString *name;


@end
```

```
@property (nonatomic, strong) NSString *name;
```

```
@property  nonatomic
           atomic
```

```
@property            strong
                     weak
                     assign
                     copy
```

```
NSString *_name;

- (void)setName:(NSString *)name
{
    // Depends on property attributes
}

- (NSString *)name
{
    return _name;
}
```

```
[myRoom setName:@”Kitchen”];
```

```
[myRoom setName:@"Kitchen"];
myRoom.name = @"Kitchen";
```

Methods

Room.h

```
@interface Room : NSObject

@property (nonatomic, strong) NSString *name;


@end
```

Room.h

```
@interface Room : NSObject

@property (nonatomic, strong) NSString *name;

- (id)initWithName:(NSString *)name;

@end
```

```
- (NSUInteger)length;

- (NSComparisonResult)compare:(NSString *)aString
                      options:(NSStringCompareOptions)mask
                        range:(NSRange)range;

- (NSString *)stringByAppendingString:(NSString *)aString;
```

Implementation

Room.m

```
#import ”Room.h”

@implementation Room




@end
```

Room.m

```
#import "Room.h"

@implementation Room

- (id)initWithName:(NSString *)name
{
    // Code for initWithName:
}

@end
```

```
- (id)initWithName:(NSString *)name
{
    self = [super init];

    if (self)
    {
        // Initialize the object here
    }

    return self;
}
```

```
- (id)initWithName:(NSString *)name
{
    self = [super init];

    if (self)
    {
        _name = name;
    }

    return self;
}
```

alloc & init

```
Room *kitchen = [[Room alloc] initWithName:@“Kitchen”];
```

Private Interface

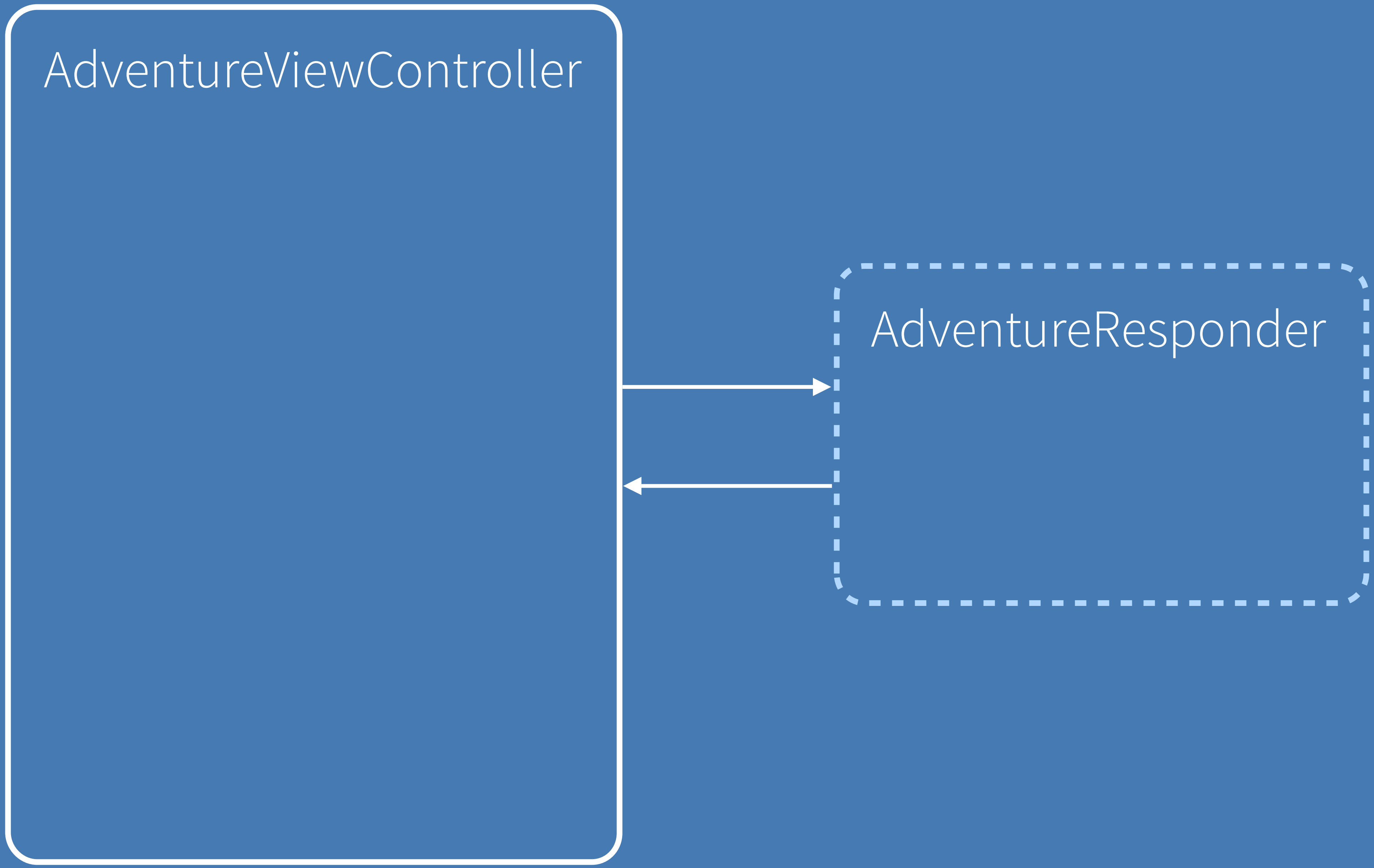
Room.m

```
#import "Room.h"

@interface Room ()

// Private interface goes here

@end

@implementation Room

// Public and private implementation goes here

@end
```

Namespaces

Coding Break

Protocols


AdventureViewController
AdventureResponder

```
@protocol AdventureResponder



@end
```

```
@protocol AdventureResponder


- (NSString *)responseForInput:(NSString *)input;

@end
```

```
@protocol AdventureResponder <NSObject>


- (NSString *)responseForInput:(NSString *)input;

@end
```

```
@protocol AdventureResponder <NSObject>

@optional

- (NSString *)responseForInput:(NSString *)input;

@end
```

.h

```
@interface ZombieAdventure : NSObject <AdventureResponder>



@end
```

.m

```
@implementation ZombieAdventure

- (NSString *)responseForInput:(NSString *)input
{
    return @”You were eaten by zombies. Game over.”;
}

@end
```

Delegates

.h

```
@protocol UITextFieldDelegate;

@interface UITextField : UIControl

@property (nonatomic, weak) id <UITextFieldDelegate> delegate;

@end

@protocol UITextFieldDelegate <NSObject>

- (BOOL)textFieldShouldReturn:(UITextField *)textField;

@end
```

Memory Management

ARC

Automatic retain/release messages

Mix and match by class

Highly optimized

Objects don’t respond to `release`, `retain`, `autorelease` or `retainCount`

No more custom `dealloc` methods

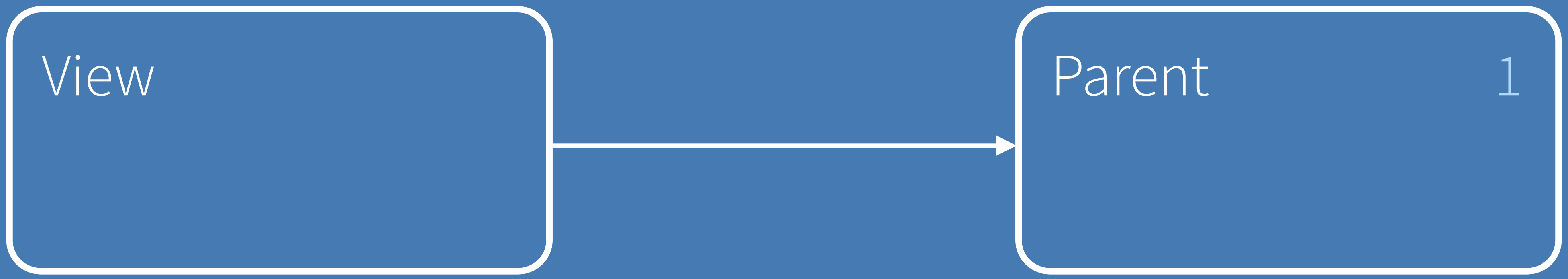
New property attributes `strong` and `weak`

New autorelease blocks

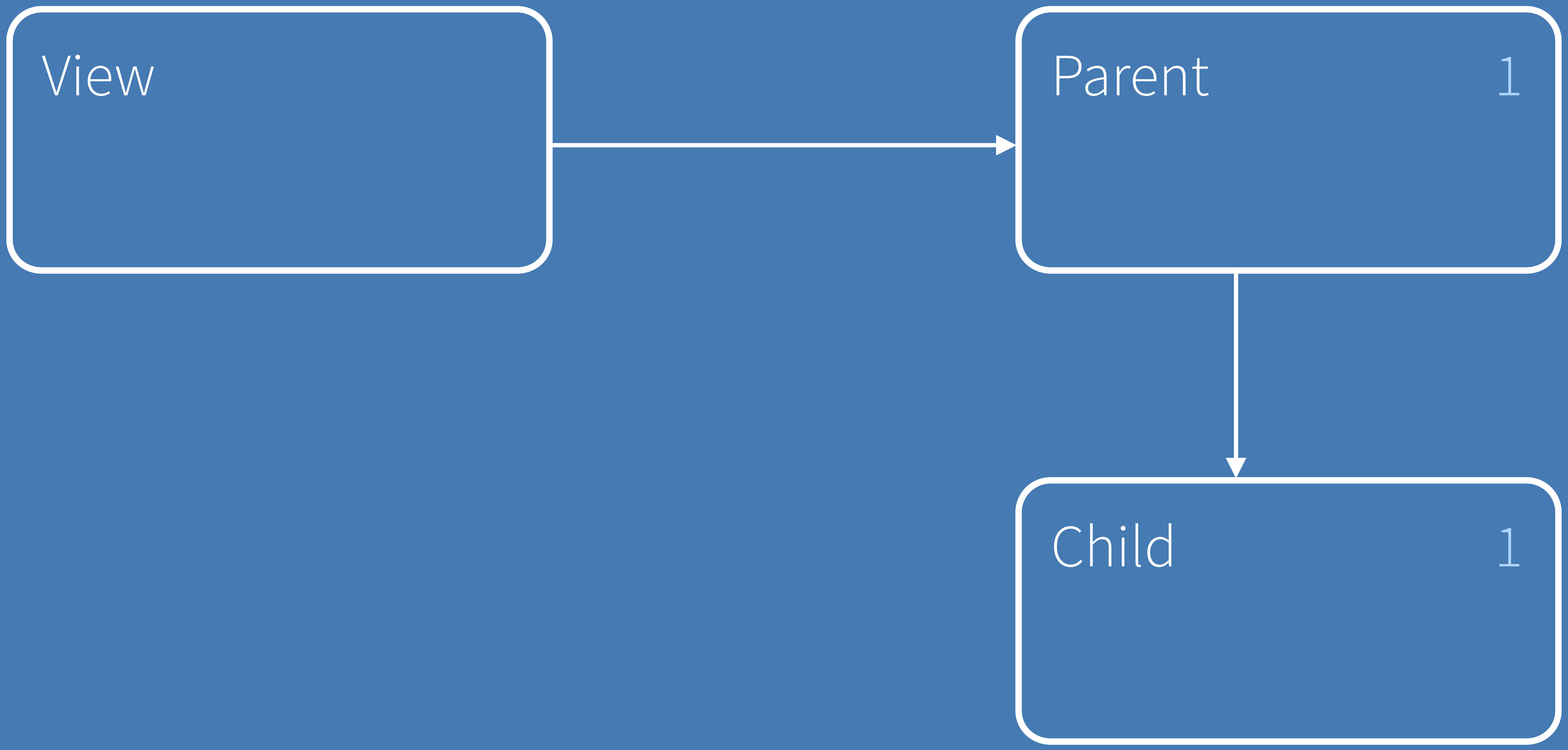
Retain Cycles

Retain Cycle

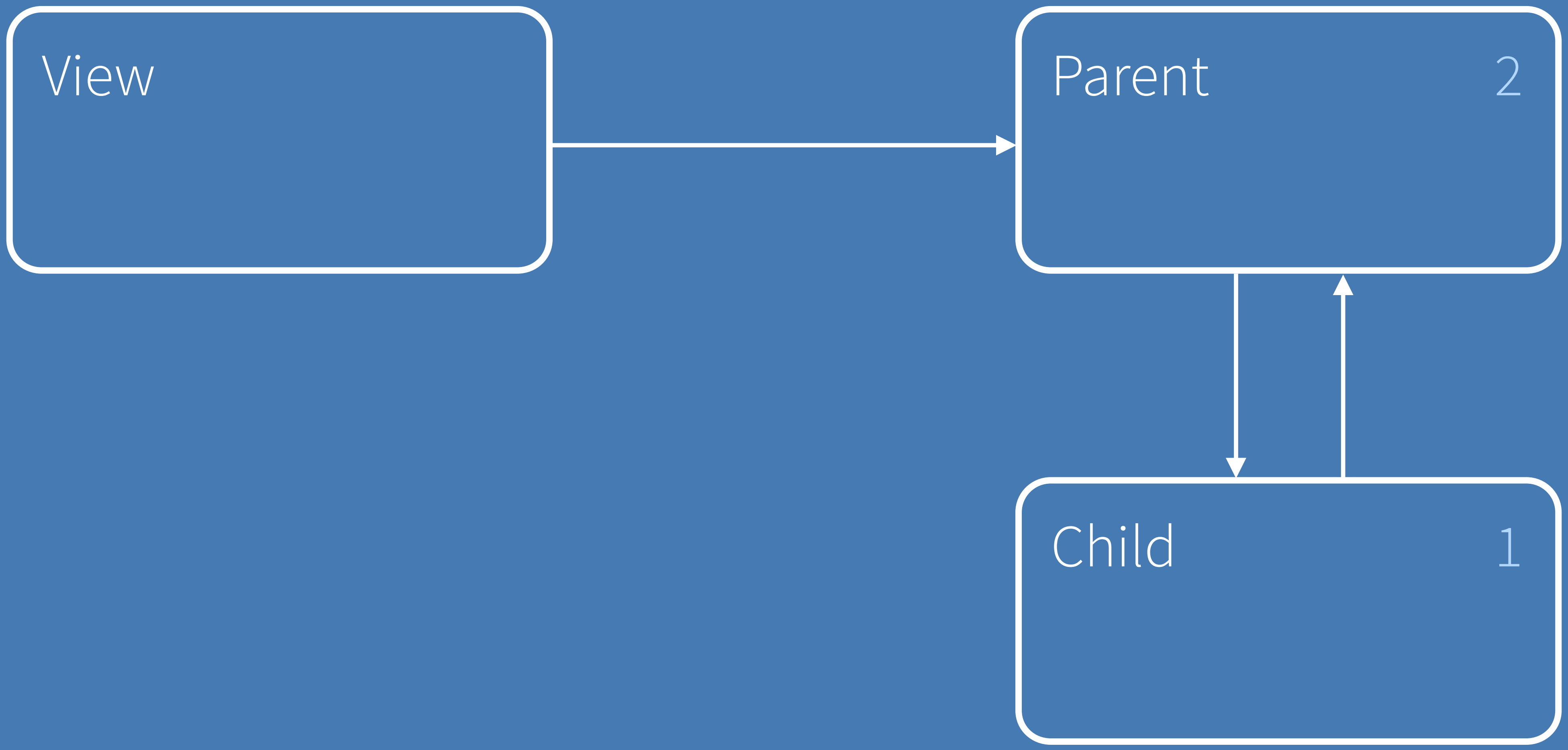
Retain Cycle



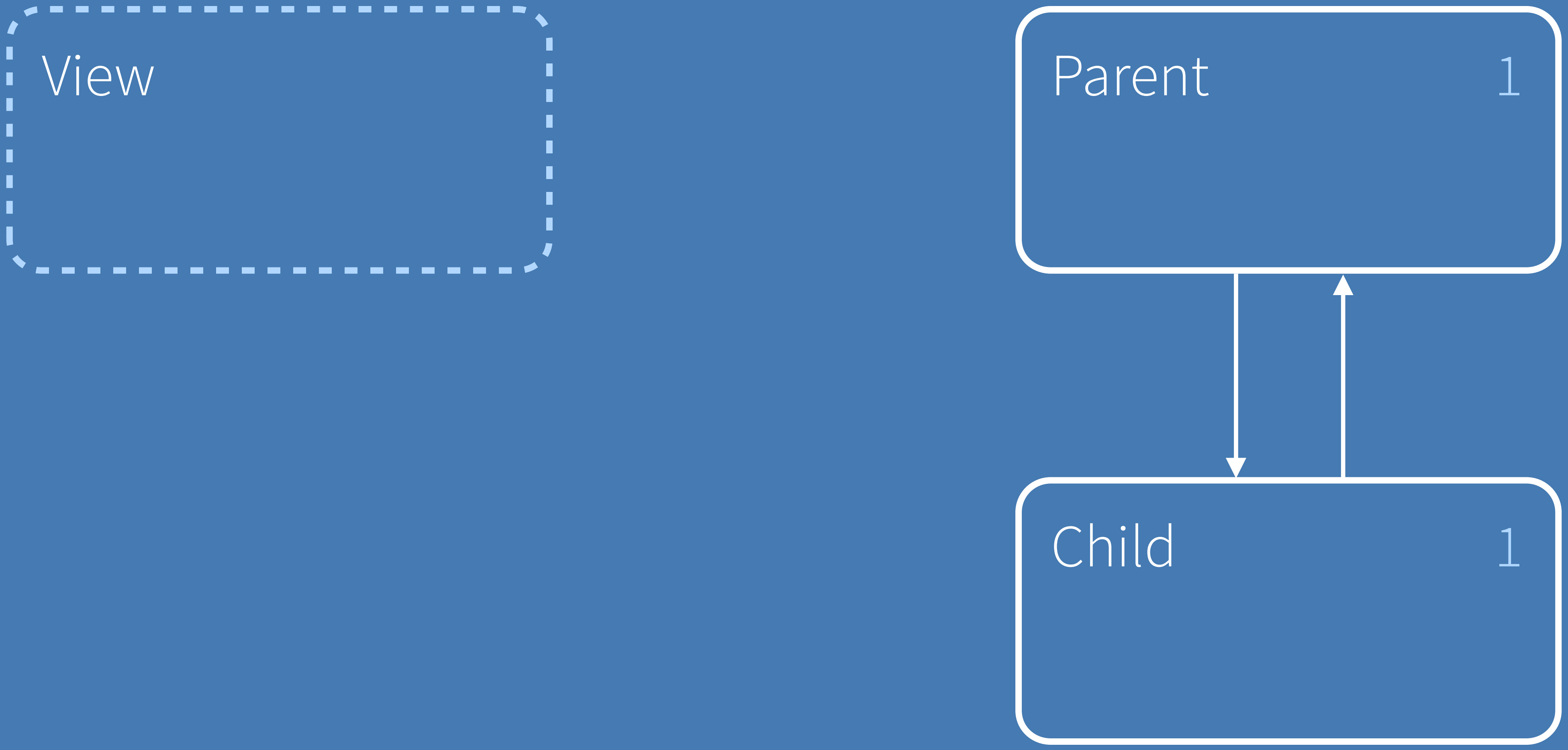
Retain Cycle

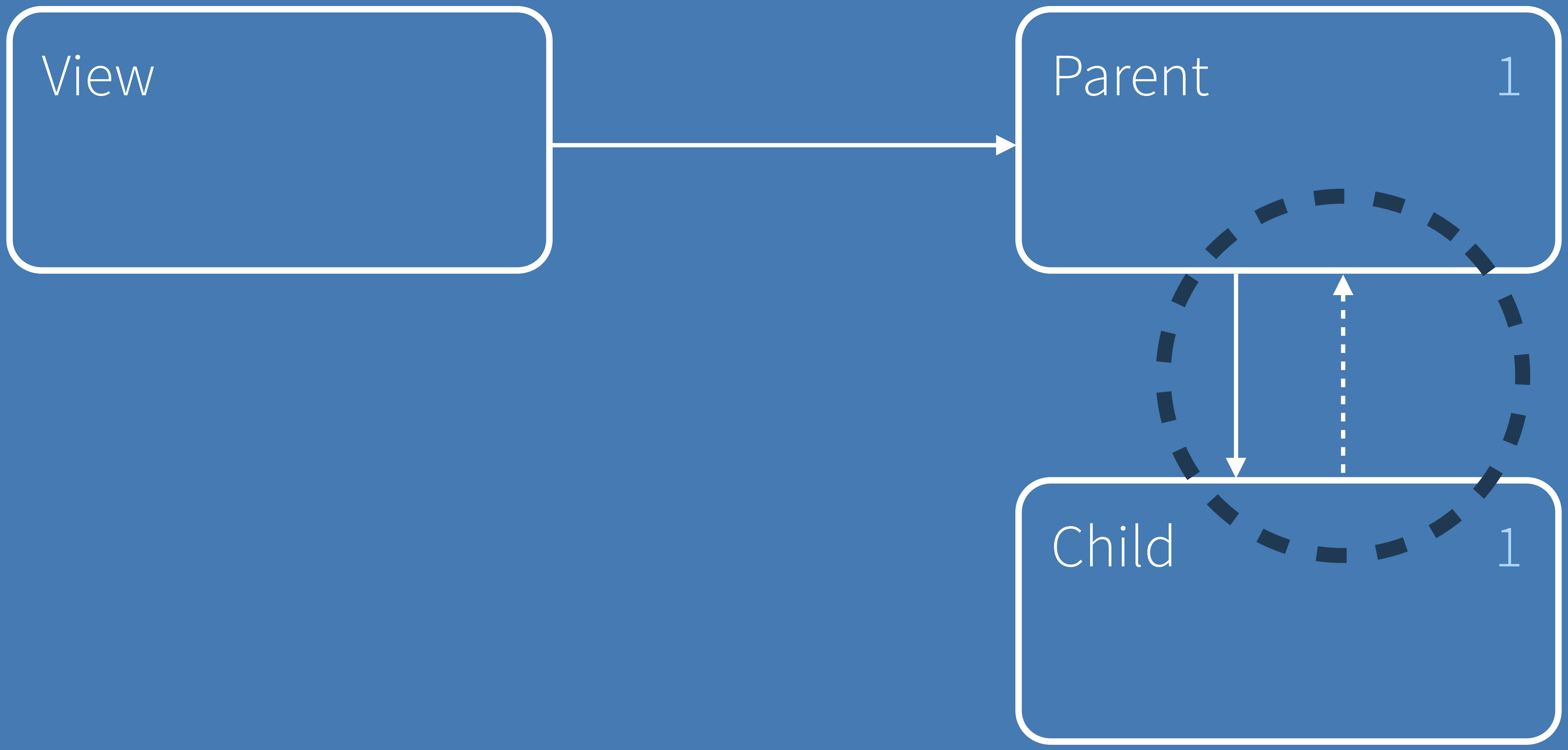


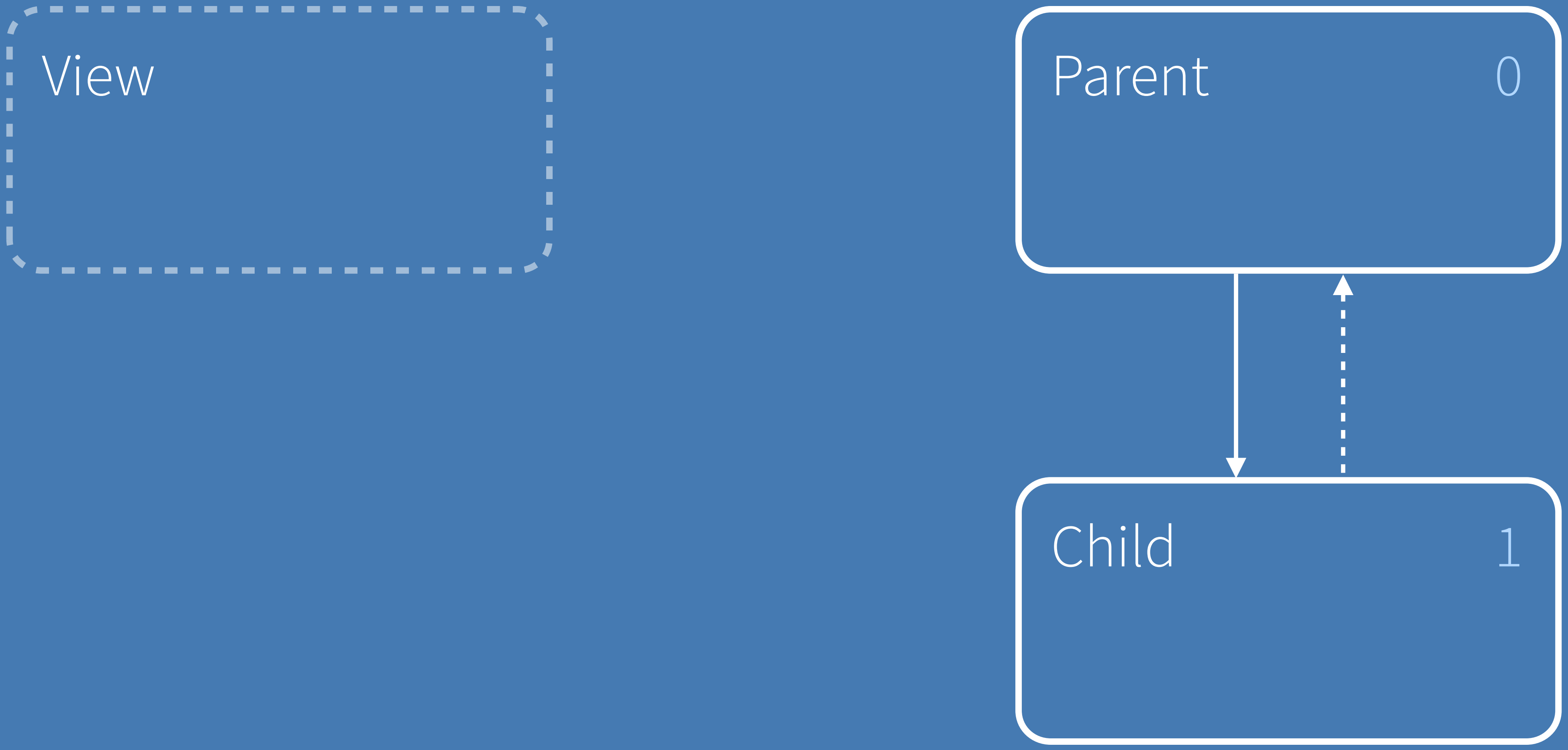
Retain Cycle



Retain Cycle




View
Parent
1
Child
1


View
Parent
0
Child
1

Child 0

Blocks

```
NSArray *trees = @[@"poplar", @"maple", @"birch"];

[trees enumerateObjectsUsingBlock:
   ^(id obj, NSUInteger idx, BOOL *stop)
   {
      NSLog(@"Block visits each tree: %@", obj);
   }];
```

```
NSArray *trees = @[@"poplar", @"maple", @"birch"];
__block NSInteger treeLetters = 0;

[trees enumerateObjectsUsingBlock:
   ^(id obj, NSUInteger idx, BOOL *stop)
   {
      NSString *tree = obj;
      treeLetters += tree.length;
   }];

NSLog(@"Found %d letters.", treeLetters);
```