



# Advanced Objective-C and Garbage Collection Techniques

**Greg Parker**  
Runtime Wrangler

# What You'll Learn

- The two faces of Objective-C
- Language and Runtime techniques
- Block esoterica
- Optimizing garbage-collected memory

# The Two Faces of Objective-C

# One Language, Two Runtimes

- The Modern runtime
- The Legacy runtime

# Platforms and Architectures

## Legacy

32-bit Mac OS  
iPhone OS Simulator

## Modern

64-bit Mac OS  
iPhone OS devices

# Platforms and Architectures

Legacy

32-bit Mac OS

Modern

64-bit Mac OS  
iPhone OS devices  
iPhone OS Simulator

New

# Why Do You Care?

- Mac OS: 32-bit apps use legacy runtime
  - Must be 64-bit only for some features
- iPhone OS: Simulator previously used legacy runtime
  - May now use new features everywhere

# Language and Runtime Techniques



# Advanced Techniques

- Writing code
- Not writing code
- Not executing code

# Advanced Techniques

- Writing code
  - Class extensions
- Not writing code
  - @synthesized properties
- Not executing code
  - Weak-linked classes

# What Is a Class Extension?

- An additional @interface
- Same @implementation
- Different header file

```
@interface MyClass ()  
-(id)myInternalMethod;  
@property id myInternalProperty;  
@end
```

# Hiding Methods in a Class Extension

PetShopView.h

```
@interface PetShopView : NSView {
    @private
    NSArray *kittens;
    NSArray *puppies;
}

@property (readwrite) int puppyFood;

-(void) feedSnakeWith:(id)food;

@end
```

```
PetShopView *shop;
shop.puppyFood = 0;
[shop feedSnakeWith:shop.kittens];
```

# Hiding Methods in a Class Extension

## PetShopView.h

```
@interface PetShopView : NSView {  
    @private  
    NSArray *kittens;  
    NSArray *puppies;  
}  
  
@property (readonly) int puppyFood;  
  
@end
```

## PetShopView-Private.h

```
@interface PetShopView ()  
  
@property (readwrite) int puppyFood;  
  
-(void) feedSnakeWith:(id)food;  
  
@end
```


# Hiding Methods in a Class Extension

PetShopView.h

```
@interface PetShopView : NSView
@property (readonly) int puppyFood;
@end
```

PetShopView-Private.h

```
@interface PetShopView () {
    @private
    NSArray *kittens;
    NSArray *puppies;
}
@property (readwrite) int puppyFood;
-(void) feedSnakeWith:(id)food;
@end
```



# Ivars in Class Extensions

- @private by default
- Modern runtime only
- LLVM Compiler only
  - Preview: Other C Flags = -Xclang -fobjc-nonfragile-abi2

# @synthesize

## PetShopView.h

```
@interface PetShopView : NSView
{
    @private
    int puppyFood;
}
@property (readwrite) int puppyFood;
@end
```

## PetShopView.m

```
@implementation PetShopView
@synthesize puppyFood;

-(id) init {
    self = [super init];
    self->puppyFood = 10;
    return self;
}

@end
```



# @synthesize by Default



- Modern runtime only
- LLVM Compiler only
  - Preview: Other C Flags = `-Xclang -fobjc-nonfragile-abi2`

# Alternatives to @synthesize

- Write accessor methods by hand
- @dynamic with message forwarding
- @dynamic with dynamic method resolution
- @dynamic with NSObject

# Weak Linking

```
if (NSDrawNinePartImage != NULL) {  
    NSDrawNinePartImage(...);  
} else {  
    // draw something else  
}
```

# Weak Linking with NSStringFromClass

```
Class popoverClass = NSStringFromClass(@"UIPopoverController");  
if (popoverClass) {  
    UIPopoverController *obj = [[popoverClass alloc] init];  
} else {  
    // do something else  
}
```

```
@interface MyController : UIPopoverController  
// crashes  
@end  
  
MyController *obj = [[MyController alloc] init];  
// sorry
```

# Weak Linking Simplified



```
if ([UIPopoverController class]) {
    UIPopoverController *obj = [[UIPopoverController alloc] init];
} else {
    // do something else
}
```

```
@interface MyController : UIPopoverController
// OK
@end

MyController *obj = [[MyController alloc] init];
if (obj) {
    // OK
}
```

# Weak Linking of Objective-C Classes

- Simplify deployment to multiple OS versions
- Implementation forthcoming
- Compiler support
  - GCC and LLVM compilers in Xcode 4
- Runtime support
  - iPhone OS 3.1 and later
  - Not yet on Mac OS
- SDK support
  - Not yet on iPhone OS
  - Not yet on Mac OS

# Block Esoterica

# Block Esoterica

- Block memory in action
- Copying blocks
- `__block` storage variables




# Copying

```
__block int shared;
```

```
int captured = 10;
```

Stack



```
captured = 10;  
block1 = ^{  
    shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block1 copy];
```

Function()

Heap

# Copying

```
__block int shared;
```

```
int captured = 10;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block1 copy];
```

Function()



Heap

# Copying

```
__block int shared;
```

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block1 copy];
```

Function()



Heap

# Copying

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block1 copy];
```

Function()

```
__block int shared;
```

```
block2:  
  const int captured = 10;
```

Heap

# Copying



```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block1 copy];
```

Function()

```
__block int shared;
```

```
block2:  
  const int captured = 10;
```

```
block3:  
  const int captured = 10;
```

Heap



# Copying

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block2 copy];
```

Function()

```
__block int shared;
```

```
block2:  
  const int captured = 10;
```

Heap

# Copying

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block2 copy];
```

Function()

```
__block int shared;
```

```
block2 and block3:  
  const int captured = 10;
```

Heap

# Cleanup

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block2 copy];
```

Function()

```
__block int shared;
```

```
block2 and block3:  
  const int captured = 10;
```

Heap



# Cleanup

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block2 copy];
```

Function()

```
__block int shared;
```

Heap

# Cleanup

```
int captured = 20;
```

```
block1:  
  const int captured = 10;
```

Stack

```
captured = 10;  
block1 = ^{  
  shared += captured;  
};  
captured = 20;  
block2 = [block1 copy];  
block3 = [block2 copy];
```

Function()

```
__block int shared;
```

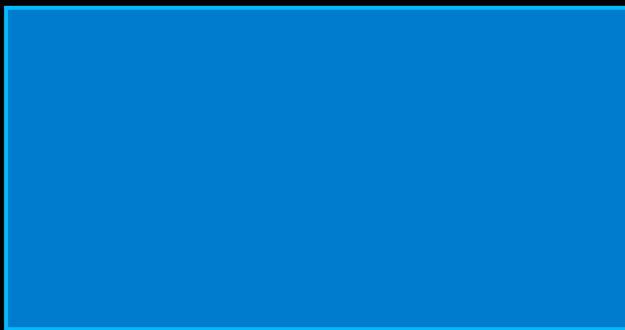
```
block2 and block3:  
  const int captured = 10;
```

Heap

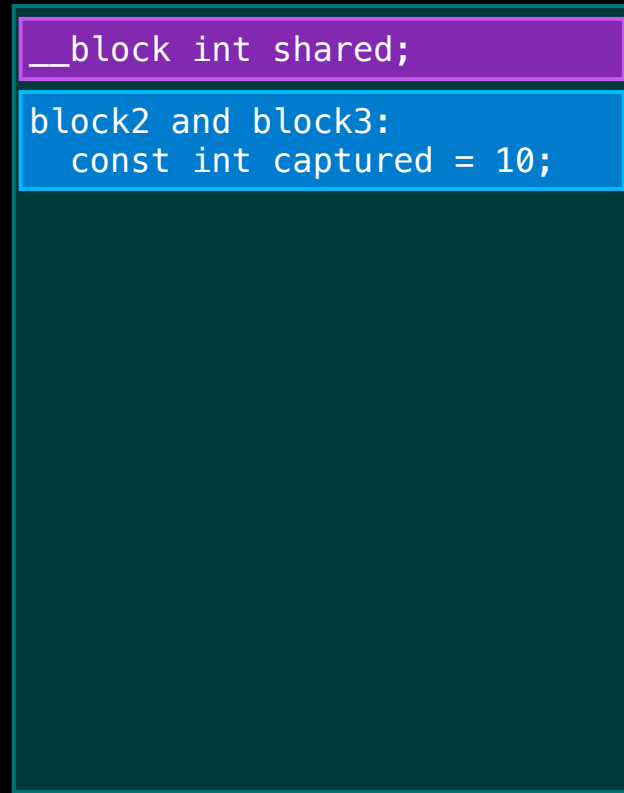
# Cleanup



Stack

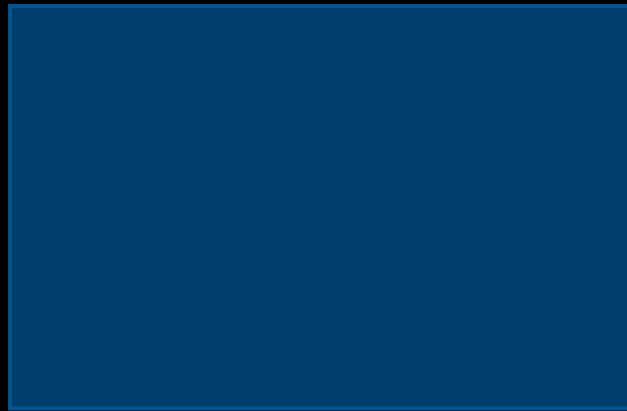


Function()

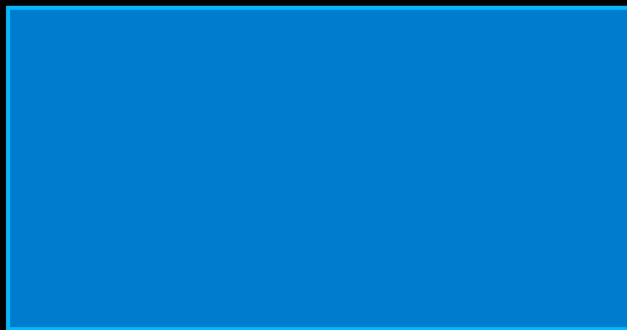


Heap

# Cleanup



Stack



Function()



Heap

# Block Copies

- Outlive the creating function
- Runnable on another thread
  - GC: must copy even if you run it synchronously!

# How to Copy

- [myBlock copy] and [myBlock release]
- Block\_copy(myBlock) and Block\_release(myBlock)
- Prefer the methods, not the functions

# \_\_block Basics

- \_\_block is a storage class
- \_\_block variables are mutable
- \_\_block variables are shared

# \_\_block Basics

- \_\_block variable values are not retained
- \_\_block variables may be copied
- \_\_block arrays are not allowed



## \_\_block Uses

- Send values between different calls to the same block
- Send values to the block's caller
- Beware of thread synchronization

# Optimizing Garbage Collected Memory

# Memory You Don't Want

- Leak: an allocation that is no longer referenced
- Abandoned: an allocation that is referenced, but no longer used

# Leak Detection

- Finds Leaked memory
- Does not find Abandoned memory

# Garbage Collection

- Automatically deallocates Leaked memory
- Does not deallocate Abandoned memory

# Abandoned Memory Examples

- Write-only cache
- Add-only container
- Pointer to current document
- Un-drained autorelease pool

Demo

# Fixing Abandoned Memory

- Limit cache sizes
- Add explicit invalidation protocols
- Use `__weak` pointers
- Add well-placed autorelease pools



# Optimizing Memory

- Use leak detectors to find leaked memory
  - GC: works with unmanaged and collector-disabled memory
- Use Heapshot to find abandoned memory
  - GC: set `AUTO_USE_TLC = NO`

# Summary

- Runtime count reduced on iPhone OS
- Old language features given new twists
- Block objects demystified
- GC memory optimized

# More Information

## Michael Jurewitz

Developer Tools Evangelist  
[jurewitz@apple.com](mailto:jurewitz@apple.com)

## Documentation

The Objective-C Programming Language  
Objective-C Runtime Programming Guide  
<http://developer.apple.com/>

## Apple Developer Forums

<http://devforums.apple.com>

# Related Sessions

Advanced Memory Analysis with Instruments

Presidio  
Thursday 11:30AM

Introducing Blocks and Grand Central Dispatch on iPhone (Repeat)

Pacific Heights  
Friday 2:00PM

# Q&A



