Understanding Objective-C Inside and Out

CodeMash, January 6th, 2014

Jeff Kelley | @SlaunchaMan

Objective-C is Weird

```
- (BOOL)doYouKnowTheMuffinMan:(TheMuffinMan *)theMuffinMan;
```

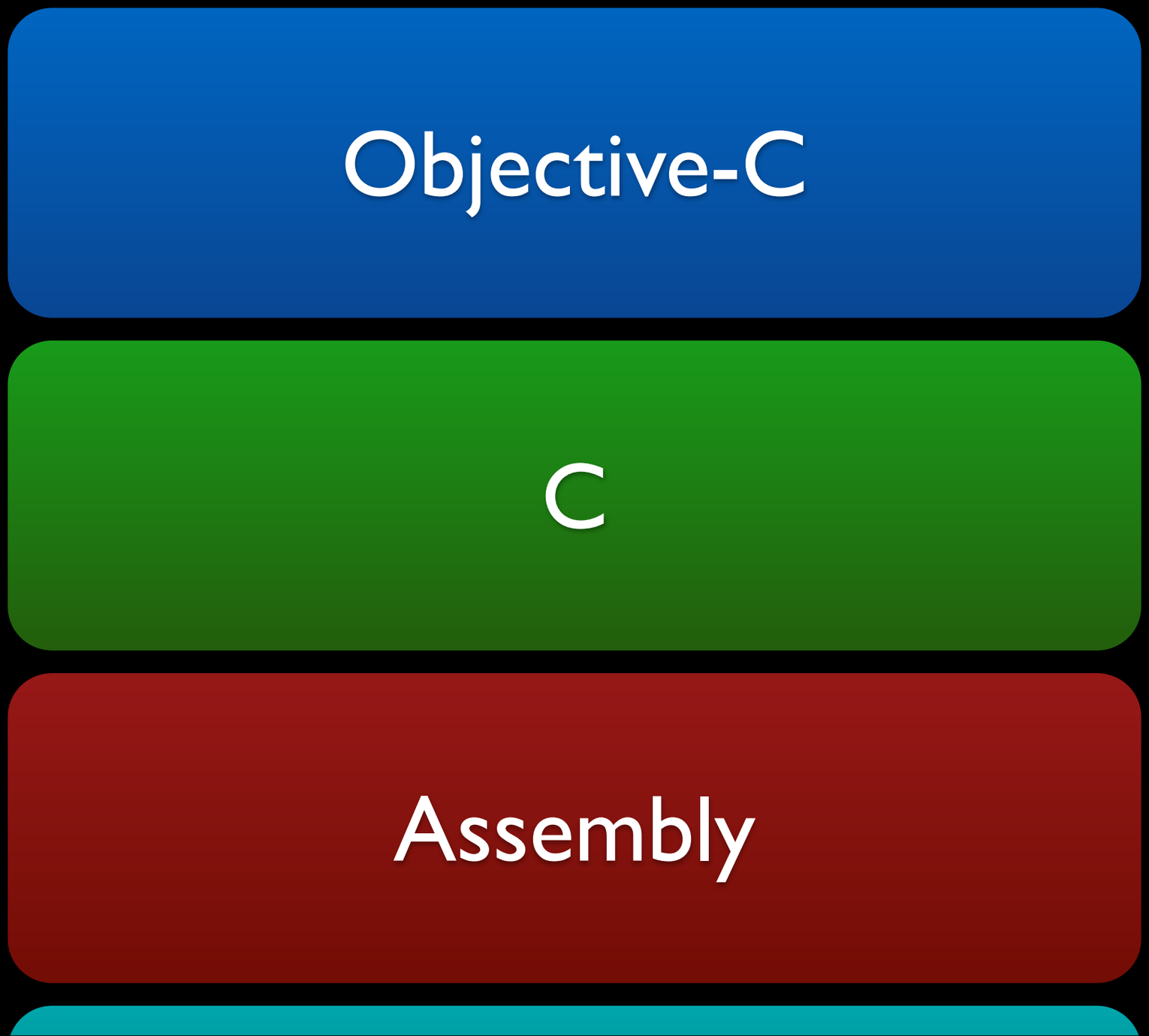
```
- sum:a:b:c:d:e:f:g:h:i:j:k;
```

```
[[[lots of] brackets] are:[literally everywhere]]
```

```
what (^about)(block *syntax)
```

Cargo Cults

Objective-C Foundations



Housekeeping

- Function vs. Method, argument vs. parameter
- This is not a talk about APIs
- This is a talk about everything Objective-C is built on

Objective-C's Beginning

- Developed by Brad Cox and Tom Love in the early 1980s
- Originally OOPC, or Object Oriented Pre-Compiler
  - Was originally a C precompiler!
- Acquired by NeXT in 1995
  - Apple now owns Objective-C rights

Why Objective-C?

- Compatibility with existing C code
  - Originally developed to work alongside telecom C code
- Brings in the object-oriented nature of Smalltalk

Smalltalk

```
| bigNumber |
bigNumber := 42 factorial

'helloWorld' indexOf: $o startingAt: 6
```

Early Objective-C

Early Objective-C

- Tim Berners-Lee wrote the first web browser, WorldWideWeb, on a NeXT Cube in Objective-C in 1989/1990
- Let's look at some source code!

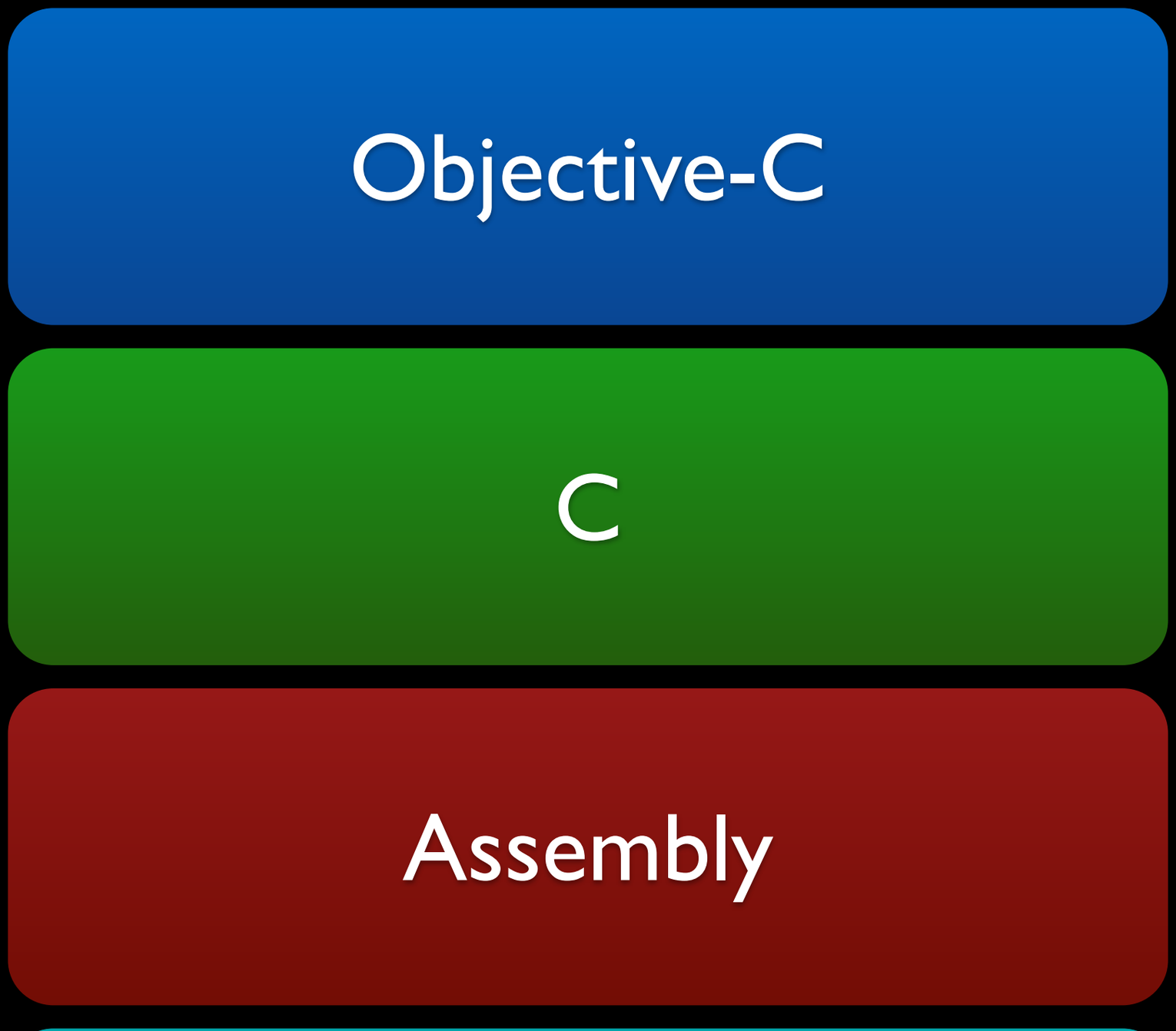
Early Objective-C

```
- readPrintInfo
/*
 * Sets the margin fields from the Application-wide PrintInfo.
 */
{
    id pi;
    float conversion, dummy;
    NXCoord left, right, top, bottom;

    [super readPrintInfo];
    pi = [NXApp printInfo];
    [self convertOldFactor:&conversion newFactor:&dummy];
    [pi getMarginLeft:&left right:&right top:&top bottom:&bottom];
    [leftMargin setFloatValue:left * conversion];
    [rightMargin setFloatValue:right * conversion];
    [topMargin setFloatValue:top * conversion];
    [bottomMargin setFloatValue:bottom * conversion];

    return self;
}
```

Objective-C Foundations



Basic C Example

```
int main(int ac, char *av[])
{
   int a = 10;
   int b = 32;

   return a + b;
}
```

Basic C Function

```
int add(int a, int b)
{
    return a + b;
}
```

```
int main(int ac, char *av[])
{
    int a = 10;
    int b = 32;

    return add(a + b);
}
```

C Functions

Basic C Function

```
int add(int a, int b)
{
    return a + b;
}
```

```
int main(int ac, char *av[])
{
    int a = 10;
    int b = 32;

    return add(a + b);
}
```

C printf Example

```
#include <stdio.h>

int main(int ac, char *av[])
{
   int a = 10;
   int b = 32;

   printf("a + b = %d\n", a + b);
}
```

Objective-C Messages

```
[myArray addObject:theObject]
```

```
objc_msgSend(myArray, @selector(addObject:), theObject);
```

Objective-C Messages

```
[viewController tableView:myTableView
  didSelectRowAtIndexPath:indexPath];
```

```
objc_msgSend(viewController,
             @selector(tableView:didSelectRowAtIndexPath:),
             myTableView,
             indexPath);
```

objc_msgSend()

- Finds the right method to call at runtime, sets everything up, and then runs it
- “crashes a lot”
- “Objective-C is slow”

objc_msgSend()

```
// 44 instruction bytes
_objc_msgSend:
    testq   %rdi, %rdi
    je,pn   NIL
    testb   $1, %dil
    jne,pn  TAGGED
    movq    (%rdi), %r11

    movq    %rsi, %r10
    andl    24(%r11), %r10d
    shlq    $4, %r10
    addq    16(%r11), %r10   fset

    cmpq    (%r10), %rsi
    jne     LOOP

    jmpq    *8(%r10)
```

How does objc_msgSend() work?

Typedef Refresher

- Typedefs let us refer to a type by another name (cue Shakespeare quote)

```
typedef int foo;

foo a = 42;
```

Block Typedefs

- We use blocks a lot, so we use typedefs to clear up the syntax

```
typedef void(^completionHandler)(void);

CompletionHandler handler = ^{ };
```

Function Pointers

```
typedef void(*CompletionHandler_f)(void);

void HandleCompletion(void)
{
    printf("I’m a C function!\n");
}

int main(int argc, const char * argv[])
{
    CompletionHandler_f handler = &HandleCompletion;

    handler();
}
```

objc_msgSend()

- To find the method, it calls other functions

```
IMP class_getMethodImplementation(Class cls,
                                  SEL name);

typedef id (*IMP)(id, SEL, …);

typedef struct objc_selector *SEL;
```

Calling Private API

Let’s call -recursiveDescription on a view.

```
NSString *recursiveDescription =
objc_msgSend(self.window,
             @selector(recursiveDescription));
```

Calling Private API

Let's call -recursiveDescription on a view.

```
SEL selector = @selector(recursiveDescription);

IMP recursiveDescription_IMP =
class_getMethodImplementation([self.window class],
                              selector);

NSString *recursiveDescription =
recursiveDescription_IMP(self.window, selector);
```

Casting `objc_msgSend()`

```
NSArray *myArray = @[ @1, @2, @3 ];
NSUInteger (*unsignedIntegerMessage)(id obj, SEL message) =
(NSUInteger (*)(id, SEL))objc_msgSend;

NSUInteger count = unsignedIntegerMessage(myArray,
@selector(count));
```

objc_msgSend() Family

- `id objc_msgSend(id self, SEL op, ...);`
- `id objc_msgSendSuper(id self, SEL op, ...);`
- `long double objc_msgSend_fpret(id self, SEL op, ...);`
- `void objc_msgSend_stret(id obj, SEL op, ...);`

objc_msgSend() Family

- 64-bit iOS:
- `objc_msgSend(id self, SEL op, ...);`

What is receiving these messages?

Structs

```
typedef struct {
  char *name;
  int number;
  Position position;
} BaseballPlayer;
```

What is an Object?

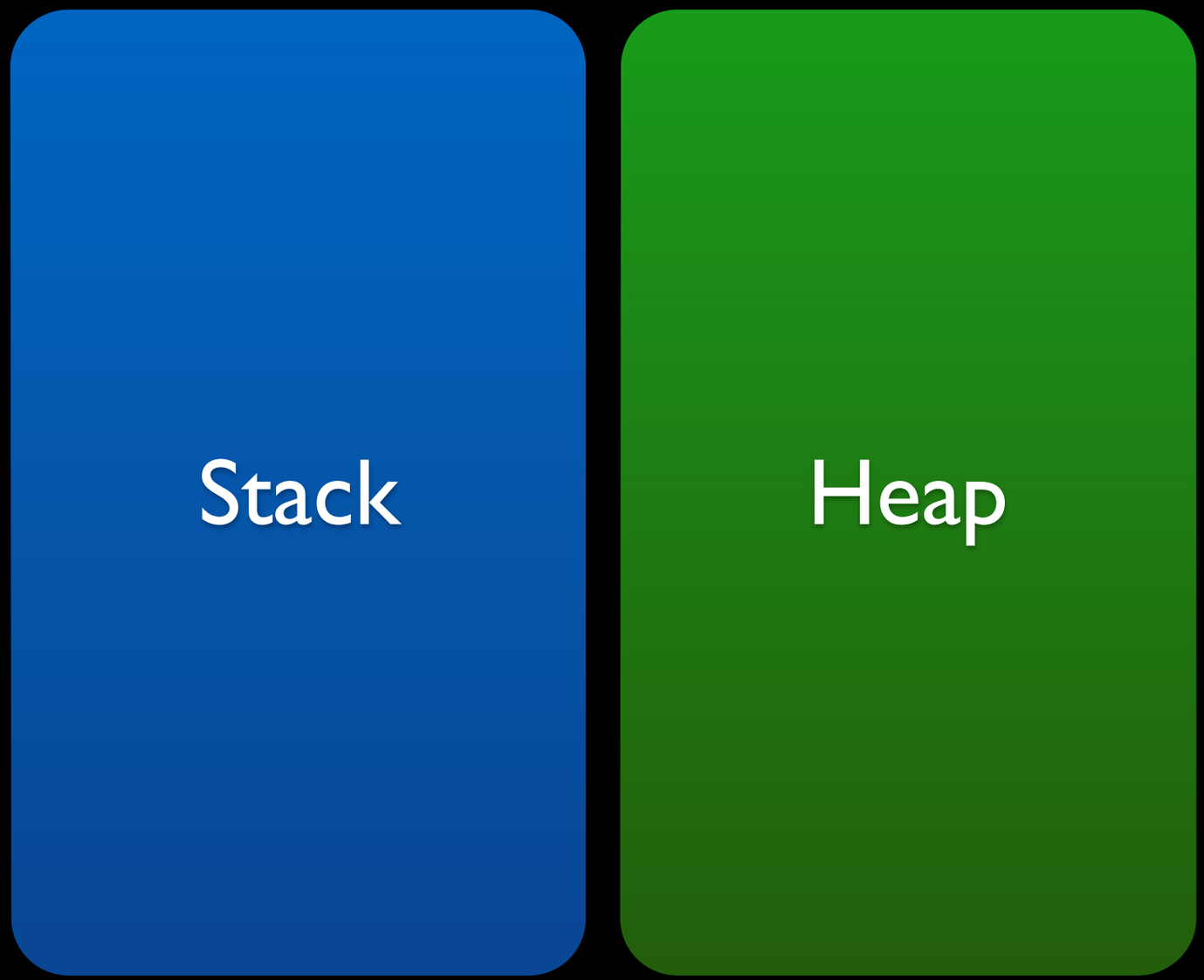
- From `<objc/objc.h>`:

```
typedef struct objc_object {
    Class isa;
} *id;
```

Wait.

I've been writing object-oriented C this entire time!

Memory Management



Memory Management

```
typedef struct {
    char *name;
    int number;
    Position position;
} BaseballPlayer;
```

```
int main(int ac, char *av[]) {
    BaseballPlayer *miggy = malloc(sizeof(BaseballPlayer));

    miggy->name = "Miguel Cabrera”;
    miggy->number = 24;
    miggy->position = thirdBase;

    free(miggy);
}
```

Reference Counting

- We don't want an object to get destroyed while we still need it
- If there is a pointer to it somewhere, we shouldn't delete it
- Retain Count
  - Starts at 1
  - Increment when you store the object's address in a pointer, decrement when you remove it or the pointer falls out of scope
  - When it's 0, the object is destroyed

ARC

- Memory management rules in the compiler

Properties

```
@interface BaseballPlayer : NSObject

@property NSString *name;
@property NSNumber *number;
@property BaseballPosition *position;

@end
```

Back in the day…

```
@interface BaseballPlayer : NSObject {
    NSString *_name;
    NSNumber *_number;
    BaseballPosition *_position;
}

- (NSString *)name;
- (void)setName:(NSString *)name;
- (NSNumber *)number;
- (void)setNumber:(NSNumber *)number;
- (BaseballPosition *)position;
- (void)setBaseballPosition:(BaseballPosition *)baseballPosition;

@end
```

Back in the day…

```
@implementation BaseballPlayer

- (NSString *)name
{
	return _name;
}

- (void)setName:(NSString *)name
{
	if (name != _name) {
		[_name release];
		_name = [name retain];
	}
}

- (NSNumber *)number
{
	return _number;
}
```

Dynamic Classes

- No, really, I need to modify the behavior of an object at runtime.
- Apple already does this with KVO
  - So how do they do it?

KVO in Action

- First, the API creates a subclass of your class at runtime:

```
Class mySubclass =
objc_allocateClassPair([NSObject class],
                       "MySubclass",
                       0);
```

KVO in Action

- Second, the API sets your object to this new class

```
object_setClass(object,
                mySubclass);
```

  - DO NOT set the isa pointer directly
    - `object->isa = mySubclass;`

KVO in Action

- Third, the API implements the method(s) it wants to override:

```
- (void)setFoo:(id)foo
{
  NSLog(@"About to set foo!");
  [super setFoo:foo];
  NSLog(@"Set foo!");
}
```

KVO in Action

- Finally, the API overrides `-class` to hide what it's done
- So `-isMemberOfClass:` still works for the original class

Why would I need to do this?

- Here's an actual, real-life example of dynamic subclassing: Kiwi mocks
- In Kiwi, the BDD test framework, you can stub a method like this:

```
[viewController stub:@selector(view)
           andReturn:thisMockView];
```

Why would I need to do this?

- In production applications, you won't need to do this too often
- But in testing, it's *invaluable*
- Go see Amber Conville's talk this afternoon!

So How is Apple Still Using Objective-C?

- As KVO illustrates, Objective-C is extremely capable of runtime hackery
- As the sole owner, Apple is free to make insane improvements to the language
  - The isa pointer in 64-bit iOS
  - Tagged Pointers
- Excellent tool development in LLVM/Clang

64-Bit iOS isa Pointer

(LSB)

1 bit	`indexed`	`0` is raw isa, `1` is non-pointer isa.
1 bit	`has_assoc`	Object has or once had an associated reference. Object with no associated references can deallocate faster.
1 bit	`has_cxx_dtor`	Object has a C++ or ARC destructor. Objects with no destructor can deallocate faster.
30 bits	`shiftcls`	Class pointer's non-zero bits.
9 bits	`magic`	Equals `0xd2`. Used by the debugger to distinguish real objects from uninitialized junk.
1 bit	`weakly_referenced`	Object is or once was pointed to by an ARC weak variable. Objects not weakly referenced can deallocate faster.
1 bit	`deallocating`	Object is currently deallocating.
1 bit	`has_sidetable_rc`	Object's retain count is too large to store inline.
19 bits	`extra_rc`	Object's retain count above 1. (For example, if `extra_rc` is 5 then the object's real retain count is 6.)

(MSB)

http://www.sealiesoftware.com/blog/

Tagged Pointers

- A pointer to an object will, due to alignment, have some extra bits
- The last bit is always 0… or is it?
- Apple sets the last bit to 1 and uses the remaining bits to store integers
- Looks and acts just like an NSNumber, but needs no additional storage

Tagged Pointers

- Don’t try to set the isa pointer. Bad things will happen.
  - Best-case scenario: compiler error.
- 64-bit iOS means we get tagged pointers
  - Performance/memory gains for free!

LLVM

LLVM/Clang Optimizations

- Link-Time Optimization
  - Applies compiler optimizations across source files, leading to huge potential gains
- New optimization level -Ofast
  - Not suitable for scientific applications that need precise floating-point values

Where is Objective-C Going?

- Recent Objective-C developments have made developing faster
  - Auto-synthesized property accessors
  - ARC
  - Packages
  - Object Literals

Where is Objective-C Going?

- Recent Objective-C developments have reduced the cognitive load of the language
  - ARC
  - Packages
  - Properties

Where is Objective-C Going?

- Future developments will likely continue down these paths
- My suggestion: get rid of pointers
- Other possibilities: automatic number boxing, better JSON parsing, etc.

Q&A

References

- http://en.wikipedia.org/wiki/Objective-C
- http://en.wikipedia.org/wiki/Tim_Berners-Lee
- http://www.mikeash.com/pyblog/friday-qa-2013-09-27-arm64-and-you.html
- http://www.sealiesoftware.com/blog/archive/2013/09/24/objc_explain_Non-pointer_isa.html
- http://sealiesoftware.com/msg/x86-mavericks.html
- http://llvm.org/docs/LinkTimeOptimization.html