(map #(welcome %)

everyone)

Organization for the Understanding of Dynamic Languages

May 1, 2012 - The Box Jelly, Honolulu, HI

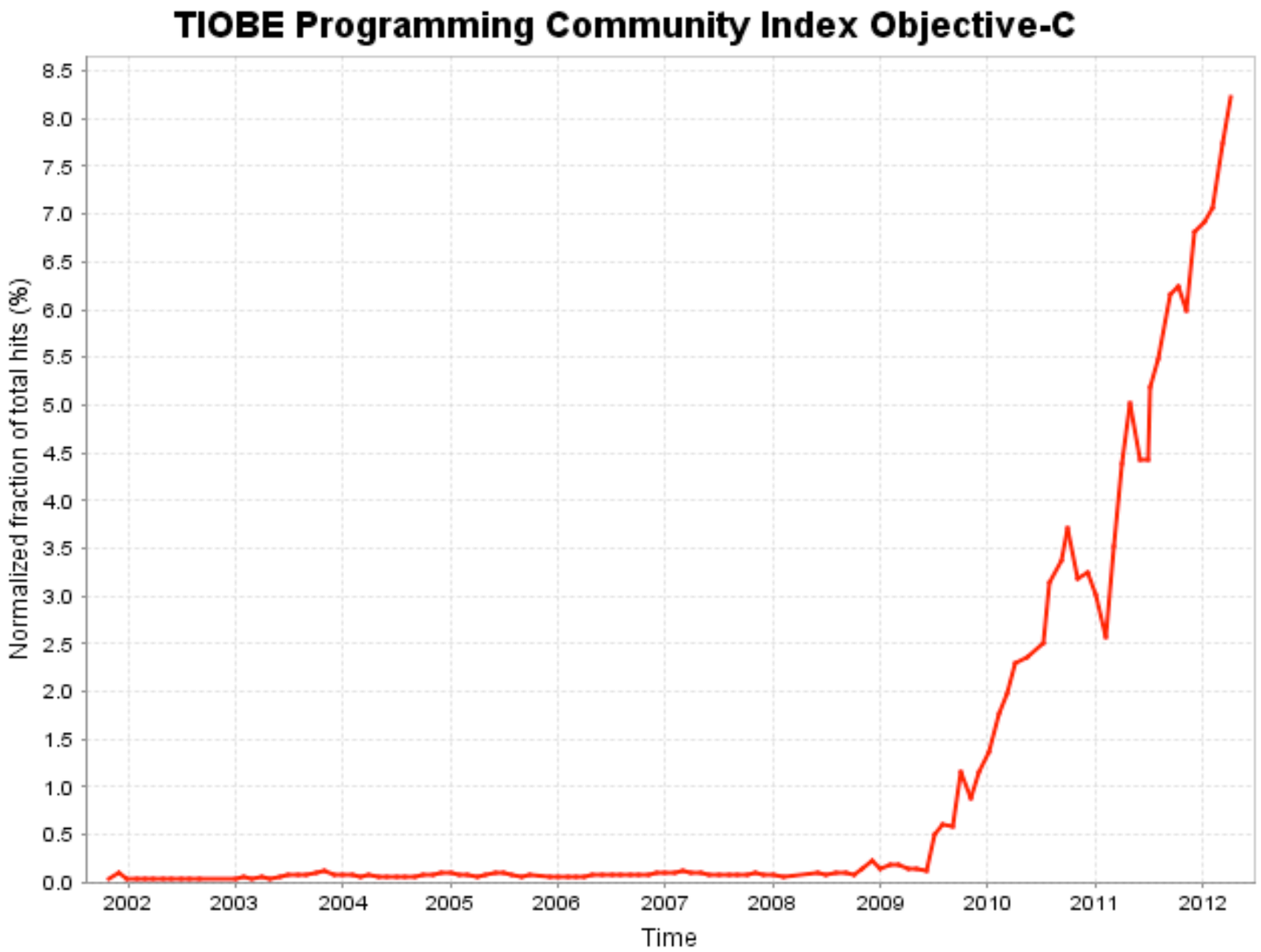
Organization for the Understanding of Dynamic Languages

Welcome to the first meeting.

The purpose of this group is not to make better programmers,

but rather to make Hawai`i a better place for programmers.

And, everyone ~~presents~~ contributes.



Created primarily by Brad Cox & Tom Love in the early 1980s

1988: NeXT licensed ObjC from StepStone (formerly PPI - Cox & Love)

- extended GCC for ObjC, wrote AppKit and FoundationKit

1996: Apple acquired NeXT and uses OpenStep in Mac OS X

- Project Builder (Xcode) and Interface Builder adopted

2001: OS X released (10.0)

2007: iPhone released

- 2008–3G, 2009–3GS, 2010–4, 2011–4S

2010: iPad released

- 2011–v2, 2012–v3

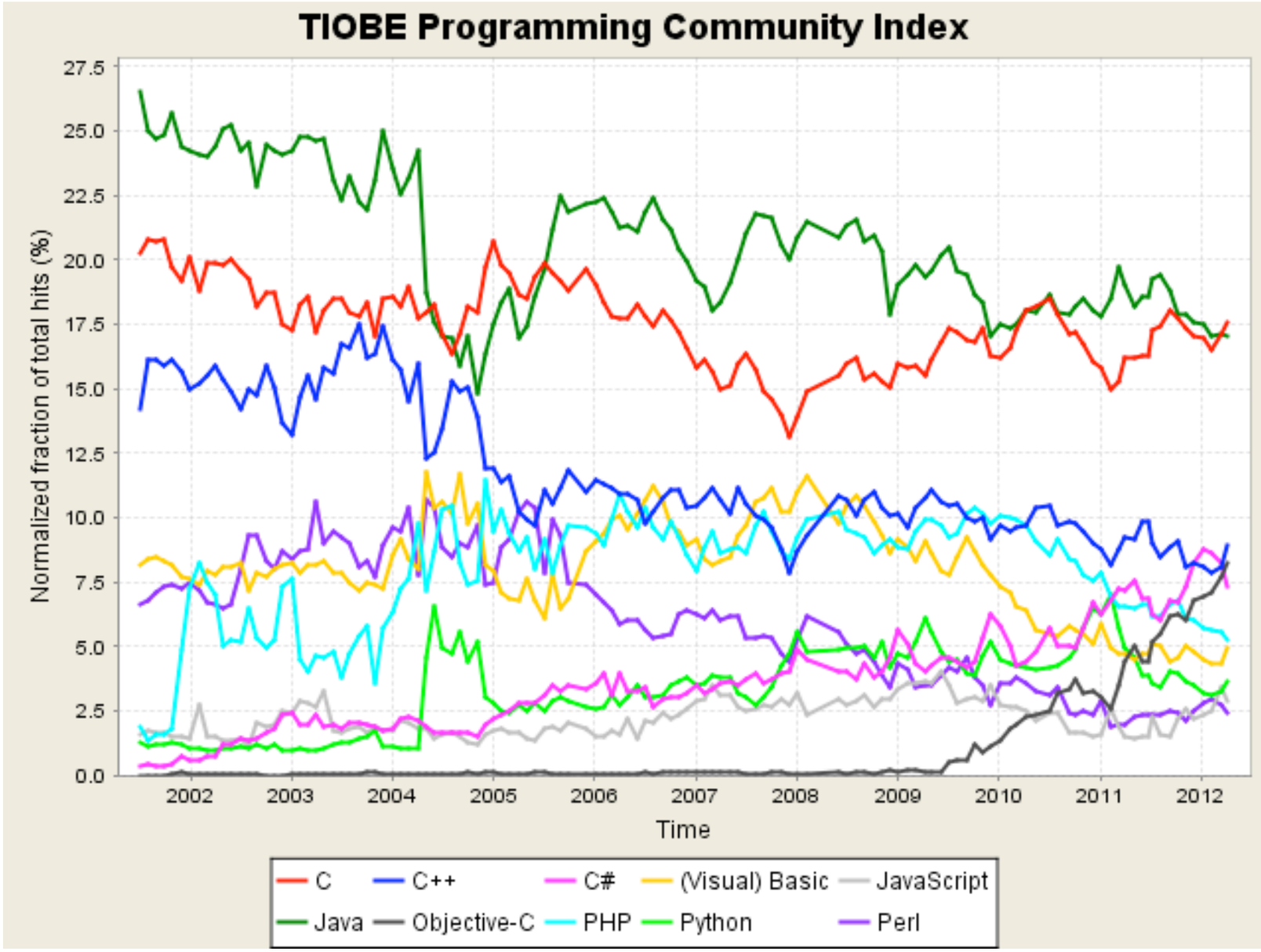
http://www.tiobe.com/index.php/paperinfo/tpci/Objective-C.html
http://en.wikipedia.org/wiki/Objective-C

“Everything popular is wrong.” – Oscar Wilde

Position Apr 2012	Position Apr 2011	Delta in Position	Programming Language	Ratings Apr 2012	Delta Apr 2011	Status
1	2	↑	C	17.555%	+1.39%	A
2	1	↓	Java	17.026%	-2.02%	A
3	3	=	C++	8.896%	-0.33%	A
4	8	↑↑↑↑	Objective-C	8.236%	+3.85%	A
5	4	↓	C#	7.348%	+0.16%	A
6	5	↓	PHP	5.288%	-1.30%	A
7	7	=	(Visual) Basic	4.962%	+0.28%	A
8	6	↓↓	Python	3.665%	-1.27%	A
9	10	↑	JavaScript	2.879%	+1.37%	A
10	9	↓	Perl	2.387%	+0.40%	A
11	11	=	Ruby	1.510%	+0.03%	A
12	24	↑↑↑↑↑↑↑↑↑↑↑↑	PL/SQL	1.373%	+0.92%	A
13	13	=	Delphi/Object Pascal	1.370%	+0.34%	A
14	35	↑↑↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	0.978%	+0.64%	A
15	15	=	Lisp	0.951%	+0.02%	A
16	17	↑	Pascal	0.812%	+0.10%	A
17	16	↓	Ada	0.783%	+0.01%	A--
18	18	=	Transact-SQL	0.760%	+0.18%	A
19	22	↑↑↑	Logo	0.652%	+0.12%	B
20	52	↑↑↑↑↑↑↑↑↑↑↑↑	NXT-G	0.578%	+0.35%	B

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

"... Java's long term downward trend line finally crosses C's stable (almost flat) popularity line."



http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Side Note: The Other Langs

Position	Programming Language	Ratings
21	Lua	0.573%
22	MATLAB	0.573%
23	SAS	0.479%
24	Assembly	0.470%
25	ActionScript	0.412%
26	Fortran	0.405%
27	RPG (OS/400)	0.401%
28	Scheme	0.395%
29	COBOL	0.391%
30	Groovy	0.389%
31	R	0.380%
32	Bash	0.367%
33	ABAP	0.350%
34	cg	0.340%
35	Scratch	0.331%
36	D	0.328%
37	Haskell	0.288%
38	Prolog	0.286%
39	F#	0.281%
40	APL	0.248%
41	Smalltalk	0.244%
42	(Visual) FoxPro	0.239%
43	Erlang	0.234%
44	Awk	0.228%
45	Scala	0.225%
46	Forth	0.219%
47	ML	0.218%
48	Alice	0.176%
49	CFML	0.172%
50	VBScript	0.169%

It's worth considering considering...

- Scala just appeared at #45
- Clojure not in the top 50 (but LISP is #15)
- Haskell #37

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Historical Perspective

Programming Language	Position Apr 2012	Position Apr 2007	Position Apr 1997	Position Apr 1987
C	1	2	1	1
Java	2	1	3	-
C++	3	3	2	7
Objective-C	4	42	-	-
C#	5	8	-	-
PHP	6	4	-	-
(Visual) Basic	7	5	4	5
Python	8	7	22	-
JavaScript	9	9	18	-
Perl	10	6	6	-
Lisp	15	17	16	3
Ada	17	16	11	2

Largest Increase

Year	Winner
2011	Objective-C
2010	Python
2009	Go
2008	C
2007	Python
2006	Ruby
2005	Java
2004	PHP
2003	C++

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Our Planned Talks

Position Apr 2012	Position Apr 2011	Delta in Position	Programming Language	Ratings Apr 2012	Delta Apr 2011	Status
1	2	↑	C	17.555%	+1.39%	A
2	1	↓	Java	17.026%	-2.02%	A
3	3	=	C++	8.896%	-0.33%	A
4	8	↑↑↑↑	Objective-C	8.236%	+3.85%	A
5	4	↓	C#	7.348%	+0.16%	A
6	5	↓	PHP	5.288%	-1.30%	A
7	7	=	(Visual) Basic	4.962%	+0.28%	A
8	6	↓↓	Python	3.665%	-1.27%	A
9	10	↑	JavaScript	2.879%	+1.37%	A
10	9	↓	Perl	2.387%	+0.40%	A
11	11	=	Ruby	1.510%	+0.03%	A
12	24	↑↑↑↑↑↑↑↑↑↑↑↑	PL/SQL	1.373%	+0.92%	A
13	13	=	Delphi/Object Pascal	1.370%	+0.34%	A
14	35	↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	0.978%	+0.64%	A
15	15	=	Lisp	0.951%	+0.02%	A
16	17	↑	Pascal	0.812%	+0.10%	A
17	16	↓	Ada	0.783%	+0.01%	A--
18	18	=	Transact-SQL	0.760%	+0.18%	A
19	22	↑↑↑	Logo	0.652%	+0.12%	B
20	52	↑↑↑↑↑↑↑↑↑↑↑↑	NXT-G	0.578%	+0.35%	B

Position	Programming Language	Ratings
21	Lua	0.573%
22	MATLAB	0.573%
23	SAS	0.479%
24	Assembly	0.470%
25	ActionScript	0.412%
26	Fortran	0.405%
27	RPG (OS/400)	0.401%
28	Scheme	0.395%
29	COBOL	0.391%
30	Groovy	0.389%
31	R	0.380%
32	Bash	0.367%
33	ABAP	0.350%
34	cg	0.340%
35	Scratch	0.331%

36	D	0.328%
37	Haskell	0.288%
38	Prolog	0.286%
39	F#	0.281%
40	APL	0.248%
41	Smalltalk	0.244%
42	(Visual) FoxPro	0.239%
43	Erlang	0.234%
44	Awk	0.228%
45	Scala	0.225%
46	Forth	0.219%
47	ML	0.218%
48	Alice	0.176%
49	CFML	0.172%
50	VBScript	0.169%

May 1: Objective-C
May 24: Haskell
June: Python
June?: Clojure (not ranked, LISP #15)

Next?

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Dynamic Objective-C For Fun and Profit

Dynamic Languages: They're not just for Rubyists (& Pythonistas)

Kyle Oba
https://github.com/mudphone/ObjcPlayground
@mudphone

HICAP
hicapacity.org

THE BOX JELLY

Thank you!

"The Objective-C language defers as many decisions as it can from compile time and link time to runtime. Whenever possible, it does things dynamically. This means that the language requires not just a compiler, but also a runtime system to execute the compiled code. The runtime system acts as a kind of operating system for the Objective-C language; it's what makes the language work."

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html

"You should read this document to gain an understanding of how the Objective-C runtime system works and how you can take advantage of it. Typically, though, there should be little reason for you to need to know and understand this material to write a Cocoa application."

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html

Runtime Interaction

How?

- Through written and compiled Objective-C source
- NSObject methods, allowing introspection:
  - class, isMemberOfClass:, isKindOfClass:, respondsToSelector:, conformsToProtocol:, methodForSelector:
- Direct calls to runtime functions
  - Most Objective-C boils down to these functions, allowing you to write similar C code

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html

Agenda

A Primer

- Object & Class Structure, Syntax, ~~ARC~~

Messaging

- Methods, Messages & Selectors, objc_msgSend()

Swizzling

- Methods and ISA

Categories

@dynamic

KVO

Objective-C Primer

Objective-C...

- a reflective, object-oriented programming language that adds Smalltalk-style messaging to the C programming language.
- is used on OSX and iOS (both derived from the OpenStep standard).
- is the primary language of Apple's Cocoa API.
- was the primary language of the NeXTSTEP OS.
- can be run (without Apple libraries) on any system supporting GCC or Clang.

http://en.wikipedia.org/wiki/Objective-C

Objective-C Primer

Learning Objective-C: A Primer

- Google: Apple Objective-C Primer

Objective-C is a superset of C

- .h == Header files: class, type, function & constants declared
- .m == Objective-C & C source files
- .mm == Objective-C & C++ source files

http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/_index.html

Objective-C Primer

Like C++...

- No multiple inheritance
- No operator overloading

All Objects are C Structs

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

http://en.wikipedia.org/wiki/Objective-C

Objective-C Primer

Sending Messages

- Smalltalk-style
- This is not “method calling”
- One “sends a message” to a receiver
- Target of message is resolved at runtime
- More on this...

```
// C++
obj->do(arg);

// Objective-C
[obj doWithArg:arg];
```

http://en.wikipedia.org/wiki/Objective-C

Objective-C Primer

Classes

- Allows strong and weak typing
  - MyClass *thing; // <= Strong typing
  - id something; // <= Weak typing

Protocols

- Interface, with required and optional methods
- Frequently used to define delegate interface

http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/_index.html

Categories

Categories add methods to a class at runtime

- Break methods up into groups, stored in separate files
- Used to be used as an interface, until optional protocol methods were introduced
- Useful for adding utility methods to existing classes (even if you don't own them)
- Can add methods, properties, & protocols
- Can *NOT* add ivars... why?

Example Category Usage...

Category Usage

```
//  NSArray+Clojureizer.h
//  ObjcPlayground
#import <Foundation/Foundation.h>

@interface NSArray (Clojureizer)

- (NSArray *)map:(id (^)(id obj))f;

@end
```

Category Usage

```objc
//  NSArray+Clojureizer.m
//  ObjcPlayground
#import "NSArray+Clojureizer.h"

@implementation NSArray (Clojureizer)

- (NSArray *)map:(id (^)(id obj))f
{
    NSMutableArray *array = [NSMutableArray arrayWithCapacity:[self count]];
    for (id item in self) {
        [array addObject:f(item)];
    }
    return [NSArray arrayWithArray:array];
}

@end
```

WARNING: No checks on return values

Category Usage

```
NSArray *numbers = [NSArray arrayWithObjects:
                    [NSNumber numberWithInt:0],
                    [NSNumber numberWithInt:1],
                    [NSNumber numberWithInt:5], nil];

NSArray *tripled = [numbers map:^id(id item) {
    return [NSNumber numberWithInt:([item intValue] * 3)];
}];

// results == [0, 3, 15]
```

Category Pros & Cons

Pros

- Replace/add methods in any class, without access to the source code
- Full access to class's private variables

Cons

- Can't call "super" method implementation
- If two categories implement same method name, no way to determine which "wins"
- No compile-time checking of implementation (unless class extensions are used, which are compiled)

Categories

So, why can't we add those ivars?

[How does this all work?]

Object Structure

All Objective-C objects are C structs

- ISA pointer - defines the object's class
- Root class's ivars, penultimate class's ivars, ... superclass's ivars, class's ivars

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

typedef struct objc_class *Class;
```

Class Structure (old)

```
struct objc_class {
    Class isa;

#if !__OBJC2__
    Class super_class                                       OBJC2_UNAVAILABLE;
    const char *name                                        OBJC2_UNAVAILABLE;
    long version                                            OBJC2_UNAVAILABLE;
    long info                                               OBJC2_UNAVAILABLE;
    long instance_size                                      OBJC2_UNAVAILABLE;
    struct objc_ivar_list *ivars                            OBJC2_UNAVAILABLE;
    struct objc_method_list **methodLists                   OBJC2_UNAVAILABLE;
    struct objc_cache *cache                                OBJC2_UNAVAILABLE;
    struct objc_protocol_list *protocols                    OBJC2_UNAVAILABLE;
#endif

} OBJC2_UNAVAILABLE;
```

runtime.h http://opensource.apple.com/release/mac-os-x-1073/

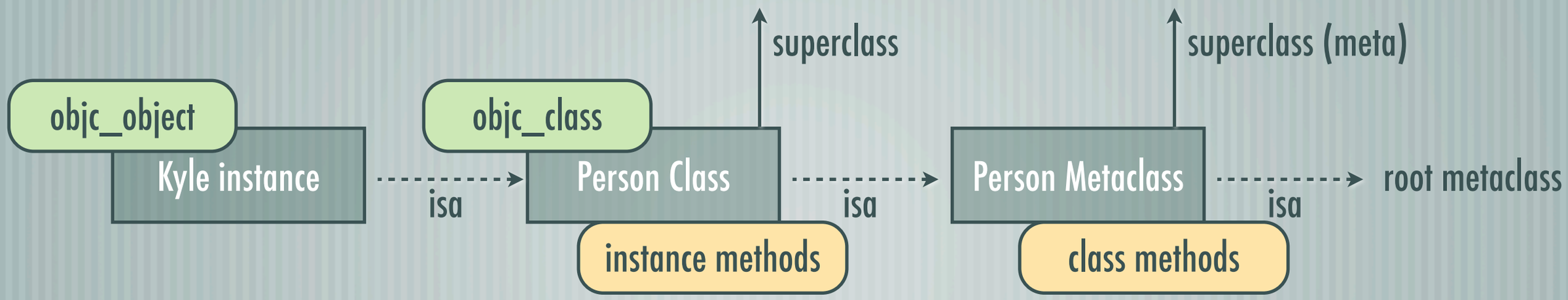
Classes & Metaclasses

Instance methods are defined in the class instance

- Superclass pointer creates hierarchy of class

A Class is like an Object

- [MyClass foo]; // => Send message to Class instance
- How does this work?



Classes and Metaclasses

Metaclasses

- Class methods are stored on the metaclass
  - ISA pointer of Class instance points to the metaclass
- Metaclasses are instances of root class's metaclass
  - Which is an instance of itself (ending in a cycle)
- Root metaclass's superclass is the root class
  - => Class objects respond to root class's instance methods
- Hidden from you and rarely accessed
  - [NSObject class]; //=> [NSObject self];

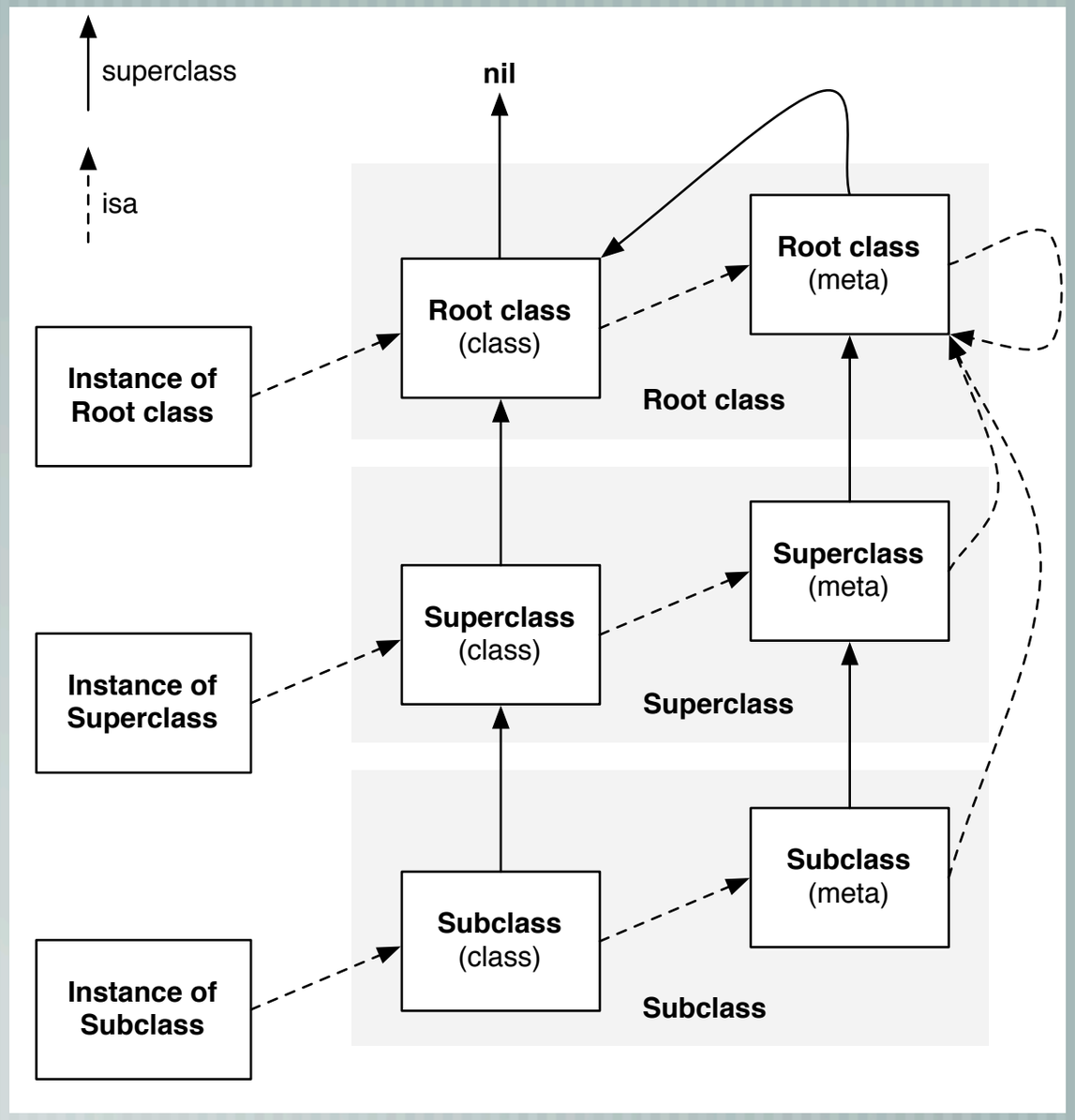


Diagram: http://www.sealiesoftware.com/blog/archive/2009/04/14/objc_explain_Classes_and_metaclasses.html


Picture of a cat

Class Structure (new)

Class structure in OS X (64) and iOS

- Broken up into read-only and writable parts
- Name and ivars stored in read-only portion => Categories can't change/add ivars

```
typedef struct class_ro_t {
   ...
   const char * name;
   ...
   const ivar_list_t * ivars;
   ...
} class_ro_t;
```

Class Structure (new)

Methods, properties, protocols define what a class can do

- Stored in writable portion of class definition
- Categories FTW
- Can be changed at runtime

```
typedef struct class_rw_t {
    ...
    const class_ro_t *ro;

    method_list_t **methods;
    ...
    struct class_t *firstSubclass;
    struct class_t *nextSiblingClass;
} class_rw_t;
```

iOS 5 Programming, Pushing the Limits, by Rob Napier & Mugunth Kumar http://iosptl.com/

Runtime Changes

Object ISA pointer is not const

- Class can be changed at runtime

Class superclass pointer is not const

- Hierarchy can be modified at runtime
- ISA Swizzling (more later)

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

typedef struct class_t {
    struct class_t *isa;
    struct class_t *superclass;
    Cache cache;
    ...
} class_t;
```

Primer Complete

Agenda

A Primer

- Object & Class Structure, Syntax

Messaging

- Methods, Messages & Selectors, objc_msgSend()

Swizzling

- Methods and ISA

Categories

@dynamic

KVO

Messaging Basics

Definitions

- METHOD: An actual piece of code associated with a class, and given a particular name
- MESSAGE: A name and parameters sent to an object
- SELECTOR: A particular way of representing the name of a message or method
- Sending a message: Taking a message and finding and executing the correct method

Types

- SEL == Selector, or name of a method
- IMP == The function itself, which accepts a object pointer (the receiver) and a selector

http://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html

What is a method?

```
// For example, this method:
- (int)foo:(NSString *)str { ...

// Is really:
int SomeClass_method_foo_(SomeClass *self,
                          SEL _cmd,
                          NSString *str) { ...
```

Methods are just C functions.

http://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html

What is a message?

```
// And this:
int result = [obj foo:@"hello"];

// Is really:
int result = ((int (*)(id, SEL, NSString *))objc_msgSend)(obj, @selector(foo:), @"hello");
```

A message send is a call to objc_msgSend().

http://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html

What does this do?

```
typedef struct class_rw_t {
    ...
    method_list_t **methods;
    ...
} class_rw_t;
```

```
struct method_t {
    SEL name;
    const char *types;
    IMP imp;
... };
```

objc_msgSend() looks up the method in the class (or superclass).

method_t is what the runtime considers a method to be:

- a name (SEL)
- argument and return types
- and a function pointer (IMP)

http://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html

Messaging Basics

Sending Messages (a review)

- Smalltalk-style
- One “sends a message” to a receiver
  - Target of message is resolved at runtime

And...

- Methods are identified by a SEL, and then...
- Resolved to a C method pointer IMP implementing it

http://en.wikipedia.org/wiki/Objective-C

Messaging Comparison

Cons

- Slower execution without compile-time binding
- Three times slower than C++ virtual method call
- Does not easily support multiple-inheritance

Pros

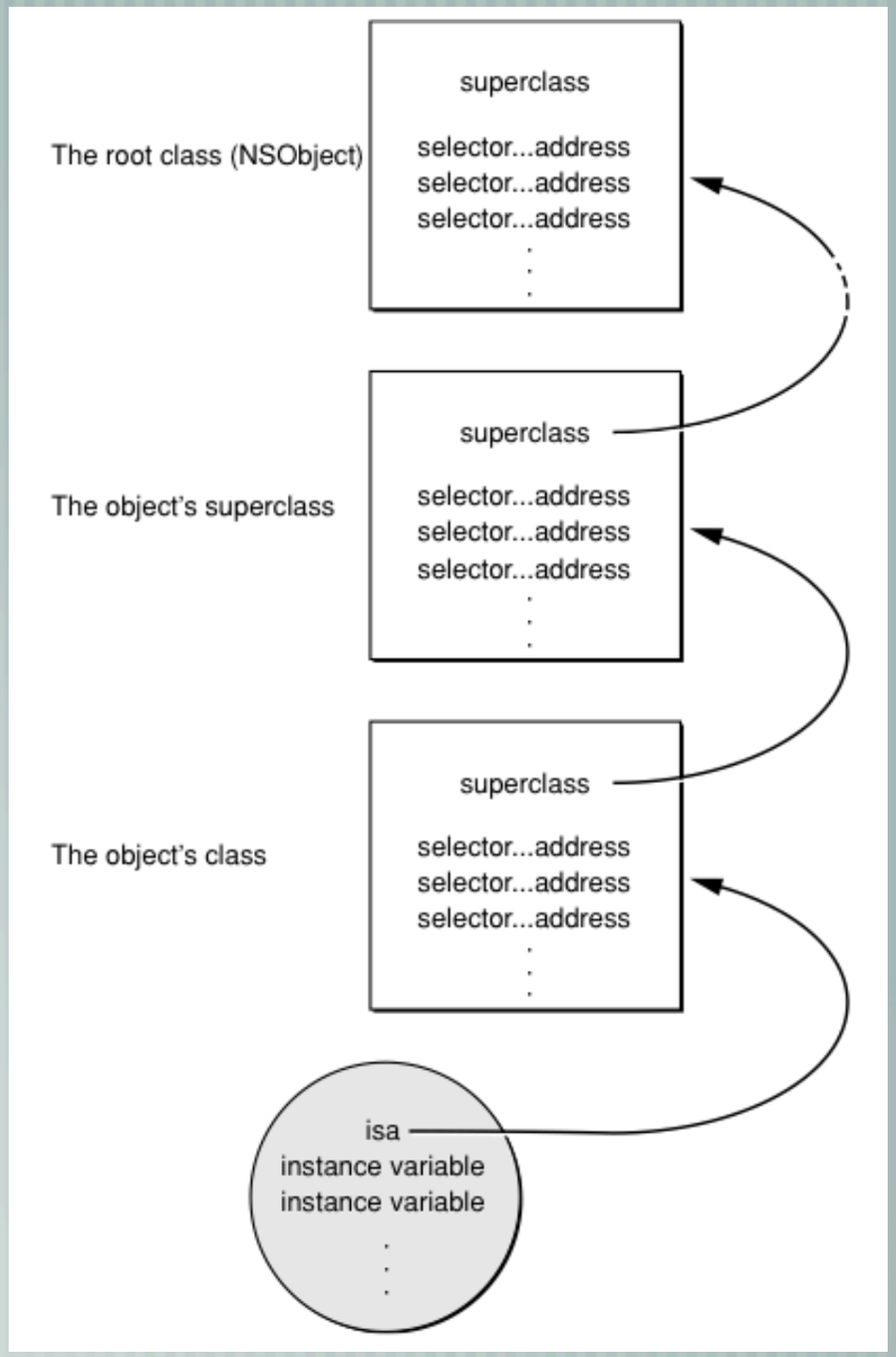
- Dynamic binding
- Methods do not have to be implemented at compile time, or at all
- Subsequent calls to method are from an IMP cache, which can be faster than C++ virtual method calls

http://en.wikipedia.org/wiki/Objective-C

Messaging in Detail

What does objc_msgSend() do?

- This is how methods are dynamically bound to messages
  - == method implementations chosen at runtime
- Looks up class of a given object
  - by dereferencing it and grabbing ISA pointer
- Looks at method list of class, search for selector
- If not found, move to superclass and do the same
- When found, jump to IMP
- Finally, there is also a method cache
  - to speed up this lookup process for future messages



https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html

Messaging in Detail

```
// For example, this method:
- (int)foo:(NSString *)str { ...

// Is really:
int SomeClass_method_foo_(SomeClass *self, SEL _cmd, NSString *str)
{ ...
```

Hidden Arguments

- "self" and "_cmd" are considered hidden, because they are not declared in source code
- But, you can still refer to them in code

Caveat (on implementation lookup)

- We could use "methodForSelector:", but this is provided by Cocoa, and so wouldn't be any fun

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html

Pictures of many cats

Messaging in Detail

I'm not going to go into the ~~details~~ opcode...

```
/********************************************************************
 * id      objc_msgSend(id    self,
 *         SEL op,
 *         ...)
 *
 * On entry: a1 is the message receiver,
 *           a2 is the selector
 ********************************************************************/

    ENTRY objc_msgSend
# check whether receiver is nil
    teq     a1, #0
    itt eq
    moveq   a2, #0
    bxeq    lr

# save registers and load receiver's class for CacheLookup
    stmfd   sp!, {a4,v1}
    ldr     v1, [a1, #ISA]

# receiver is non-nil: search the cache
    CacheLookup a2, v1, LMsgSendCacheMiss
```

Objective-C Runtime, objc4-493.11 http://opensource.apple.com/release/mac-os-x-1073/

A Deep Dive: objc_msgSend

Translation: Order of Operations

- Check if the receiver is nil, if nil -> nil-handler
- If garbage collection available, short circuit on special selectors
  - retain, release, autorelease, retainCount –> self
- Check class' cache for implementation, call it
- Compare requested selector to selectors defined in class, call it
- Compare to superclass, and up the chain
- …

```
/********************************
 * id      objc_msgSend(id    se
 *         SEL op,
 *         ...)
 *
 * On entry: a1 is the message r
 *           a2 is the selector
 ********************************

    ENTRY objc_msgSend
# check whether receiver is nil
    teq     a1, #0
    itteq
    moveq   a2, #0
    bxeq    lr

# save registers and load receiv
    stmfd   sp!, {a4,v1}
    ldr     v1, [a1, #ISA]

# receiver is non-nil: search th
    CacheLookup a2, v1, LMsgSend
```

Objective-C Runtime, objc4-493.11 http://opensource.apple.com/release/mac-os-x-1073/

A Deep Dive: objc_msgSend

Translation: Order of Operations (continued...)

- LAZY Method Resolution
  - Call resolveInstanceMethod: (or resolveClassMethod:), if returns YES, start over
  - YES: Assumes method has been added
- FAST Forwardin
  - Call forwardingTargetForSelector:, if returns non-nil, send message to object
  - Can't forward to self, starts over with new target
- NORMAL Forwarding
  - Call methodSignatureForSelector:, if returns non-nil, create and pass NSInvocation to forwardInvocation:
- Forwarding FAILURE
  - Call doesNotRecognizeSelector: –> throws exception by default

```
/*******
 * id
 *
 *
 *
 * On en
 *
 *******

    ENTR
# check
    teq
    itte
    move
    bxeq

# save r
    stmf
    ldr

# receiv
    Cach
```

Objective-C Runtime, objc4-493.11 http://opensource.apple.com/release/mac-os-x-1073/

How can I use this?

Dynamic Method Resolution

What can we do?

- @dynamic
- Fast Forwarding
- Normal Forwarding

@dynamic synthesis of properties

- As with Core Data NSManagedObjects
- Uses resolveInstanceMethod: and resolveClassMethod:
- Person.h/m dynamic getters and setters example

Dynamic Method Resolution

Fast Forwarding

- forwardingTargetForSelector: useful for proxy objects, or objects which augment another object
- CacheProxy.h/m example

Normal Forwarding

- Slower, but more flexible
- forwardInvocation:
- NSInvocation encapsulates target, selector, method signature, and arguments
  - common pattern is to change the target and invoke
- CacheProxy.h/m example

Dynamic Method Resolution

Forwarding Failure

- doesNotRecognizeSelector:
- Raises NSInvalidArgumentException by default, but you can override this
- This is the end of the line.

So, what is the value of all this?

- Core Data + @dynamic = WIN
- Forwarding mimics multiple inheritance
  - Provides features of multiple inheritance, but provides abilities in smaller objects (rather than single object)
  - Abilities are grouped in a way that is transparent to the message sender

Dynamic Method Resolution

Forwarding Failure

- doesNotRecognizeSelector:
- Raises NSInvalidArgumentException by default, but you can override this
- This is the end of the line.

So, what is the value of all this?

- Core Data + @dynamic = WIN
- Forwarding mimics multiple inheritance
  - Provides features of multiple inheritance, but provides abilities in smaller objects (rather than single object)
  - Abilities are grouped in a way that is transparent to the message sender

Dynamic Method Resolution

HOM - Higher Order Messaging

- messages passed to other messages
- [[arrayOfStrings map] stringByAppendingString:@"suffix"];
  - map returns a proxy object that forwards invocation (forwardInvocation:) to each item of the array.

http://www.mikeash.com/pyblog/friday-qa-2009-04-24-code-generation-with-llvm-part-2-fast-objective-c-forwarding.html

A Deeper Dive: objc_msgSend

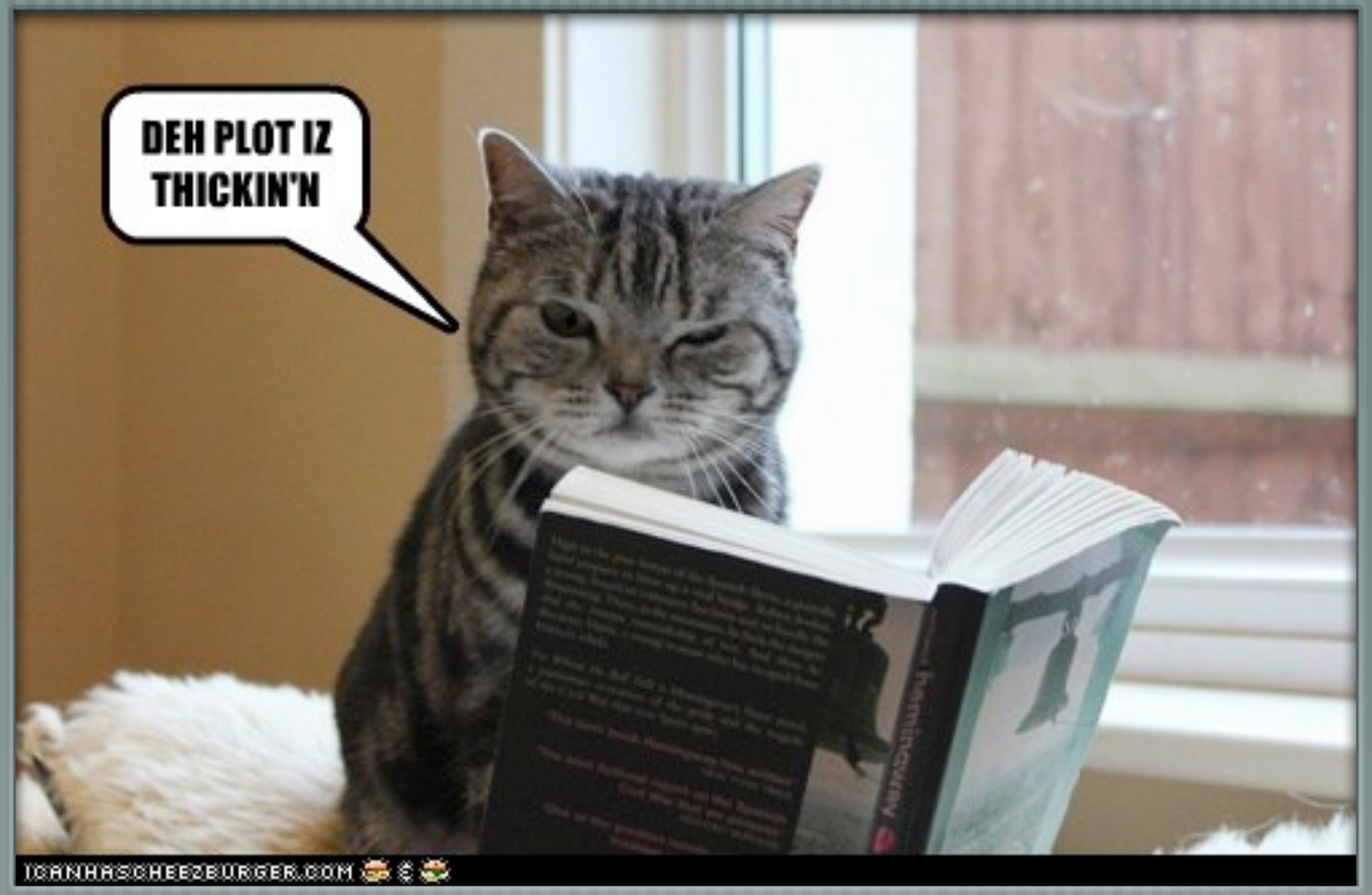
Now we get crazy...

- objc_msgSend() is really a family of functions
- "each written to handle the calling conventions required to deal with various return types under the x86_64 ABI (Application Binary Interface)..."
- Called 10s of millions of times during just the launch of your app
  - That's why it's written in assembly

Now we can look at that opcode...

Objective-C Runtime, objc4-493.11 http://opensource.apple.com/release/mac-os-x-1073/

All 4 parts of http://www.friday.com/bbum/2009/12/18/objc_msgsend-part-1-the-road-map/



SRSLY, no more messaging.

Agenda

A Primer

- Object & Class Structure, Syntax

Messaging

- Methods, Messages & Selectors, objc_msgSend()

Swizzling

- Methods and ISA

Categories

@dynamic

KVO

Swizzling

Transparently replacing one thing with another at runtime

- On iOS, usually this is methods, but can also be done with classes
- Allows you to change the behavior of Apple frameworks

WARNING: May cause app rejection.

Method Swizzling

Why not just use a category?

- Category methods with the same name as an original method, replace the original method
  - There is no way to call the original method from the category method
- Original method you with to replace might have been implemented by a category
  - If two categories have the same method, there is no way to determine which category method “wins”

Method Swizzling

Option 1: The more bad option – method_exchangeImplementations()

- Modifies selector, thus can break things
- Pseudo-recursive call can be misleading (at minimum it's confusing to read)
- You should probably the function pointer approach in RNSwizzle instead (more on this later)

If you're still interested, read:

http://mikeash.com/pyblog/friday-qa-2010-01-29-method-replacement-for-fun-and-profit.html

http://robnapier.net/blog/hijacking-methodexchangeimplementations-502

Method Swizzling

And, if you're *still* interested... NSNotificationCenter (RNHijack)

http://robnapier.net/blog/hijacking-methodexchangeimplementations-502

Method Swizzling

Option 2: The less bad option – RNSwizzle (a category on NSObject)

```
// main.m


int main(int argc, char *argv[])
{
  int retVal = 0;
  @autoreleasepool {
    [NSNotificationCenter swizzleAddObserver];
    Observer *observer = [[Observer alloc] init];
    [[NSNotificationCenter defaultCenter] addObserver:observer
                                             selector:@selector(somthingHappened:)
                                                 name:@"SomethingHappenedNotification"
                                               object:nil];

  }
  return retVal;
}
```

Method Swizzling

The Observer, for completeness...

```
@interface Observer : NSObject
@end

@implementation Observer

- (void)somthingHappened:(NSNotification*)note {
  NSLog(@"Something happened");
}
@end
```

```objc
@implementation NSNotificationCenter (RNSwizzle)

static IMP sOrigAddObserver = NULL;

static void MYAddObserver(id self, SEL _cmd, id observer,
                          SEL selector, NSString *name, id sender) {
  NSLog(@"Adding observer: %@", observer);

  // Call the old implementation
  NSAssert(sOrigAddObserver,
           @"Original addObserver: method not found.");
  if (sOrigAddObserver) {
    sOrigAddObserver(self, _cmd, observer, selector, name, sender);
  }
}

+ (void)swizzleAddObserver {
  NSAssert(! sOrigAddObserver,
           @"Only call swizzleAddObserver once.");
  SEL sel = @selector(addObserver:selector:name:object:);
   sOrigAddObserver = (void *)[self swizzleSelector:sel
                                          withIMP:(IMP)MYAddObserver];
}
@end
```

```objc
@implementation NSObject (RNSwizzle)

+ (IMP)swizzleSelector:(SEL)origSelector
               withIMP:(IMP)newIMP {
  Class class = [self class];
  Method origMethod = class_getInstanceMethod(class,
                                              origSelector);
  IMP origIMP = method_getImplementation(origMethod);

  if(!class_addMethod(self, origSelector, newIMP,
                      method_getTypeEncoding(origMethod)))
  {
    method_setImplementation(origMethod, newIMP);
  }

  return origIMP;
}
@end
```

Remember the structs?

```
typedef struct class_rw_t {
    ...
    const class_ro_t *ro;

    method_list_t **methods;
    ...
    struct class_t *firstSubclass;
    struct class_t *nextSiblingClass;
} class_rw_t;
```

Method Swizzle here

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

ISA Swizzle here

ISA Swizzling

Somethings to consider...

- The ISA pointer defines the object's class
- It is possible to modify an object's class at runtime
- One could add a setClass: method to NSObject to accomplish the same goal as in RNSwizzle
  - This would swap in a new class (kind of like a subclass), instead of using method swizzling
- This is how Key-Value-Observing (KVO) works
  - This is also why there is no overhead to *not* using KVO
  - It allows frameworks to inject code into your classes
  - Now you can do the reverse

ISA Swizzling

Some Caveats...

- Instance before and after the swizzle should have the same size
  - If not, clobbering of memory after the object could occur (which might include the ISA pointer of other object)
  - This creates a difficult situation to debug
  - The example has a check for this – See the NSAssert on class_getInstanceSize()

ISA Swizzling

main.m

```
int main(int argc, char *argv[])
{
  int retVal = 0;
  @autoreleasepool {
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc setClass:[MYNotificationCenter class]];
    Observer *observer = [[Observer alloc] init];
    [[NSNotificationCenter defaultCenter] addObserver:observer
                                             selector:@selector(somthingHappened:)
                                                 name:@"SomethingHappenedNotification"
                                               object:nil];

  }
  return retVal;
}
```

ISA Swizzling

MYNotificationCenter.m

```
@implementation MYNotificationCenter
- (void)addObserver:(id)observer selector:(SEL)aSelector
                name:(NSString *)aName object:(id)anObject
{
  NSLog(@"Adding observer: %@", observer);
  [super addObserver:observer selector:aSelector name:aName
              object:anObject];
}
@end
```

ISA Swizzling

NSObject+SetClass.m

```
@implementation NSObject (SetClass)

- (void)setClass:(Class)aClass {
  NSAssert(
    class_getInstanceSize([self class]) == class_getInstanceSize(aClass),
    @"Classes must be the same size to swizzle.");

  object_setClass(self, aClass);
}
@end
```

Method vs. ISA Swizzling

Method Swizzling

- Changes all instances of a class
- Objects remain the same class
- Difficult / confusing implementations

ISA Swizzling

- Only changes one instance at a time
- Class changes (by definition)
- Overridden methods are just like subclass methods

Resources

Buy These...

- iOS 5 Programming Pushing the Limits, by Rob Napier, Mugunth Kumar
  - http://iosptl.com/
- The Complete Friday Q&A: Volume 1, by Michael Ash
  - http://www.mikeash.com/book.html

Read These...

Basics

- Objective-C Primer
  - http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/_index.html
- Quick intro to C structs
  - http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/PointersI.html
- Objective-C Primer
  - http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/_index.html
- Runtime Source
  - http://opensource.apple.com/release/mac-os-x-1073/

Materials...

Where I got stuff from...

- Metaclass hierarchy diagram
  - http://www.sealiesoftware.com/blog/archive/2009/04/14/objc_explain_Classes_and_metaclasses.html
- Class structure
  - http://cocoawithlove.com/2010/01/getting-subclasses-of-objective-c-class.html
- Messaging examples
  - http://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html
- objc_msgSend details
  - http://www.friday.com/bbum/2009/12/18/objc_msgsend-part-1-the-road-map/

Materials...

Where I got *even more* stuff from...

- Higher Order Messaging
  - http://www.mikeash.com/pyblog/friday-qa-2009-04-24-code-generation-with-llvm-part-2-fast-objective-c-forwarding.html
  - http://cocoadev.com/index.pl?HigherOrderMessaging
- KVO
  - http://www.mikeash.com/pyblog/friday-qa-2012-03-02-key-value-observing-done-right-take-2.html
- Method swizzling
  - http://robnapier.net/blog/hijacking-methodexchangeimplementations-502

HICAP
hicapacity.org

THE BOX JELLY

Thanks Again