Objective-C Initialisation

before your app had its morning cup of coffee.

Intro

`objc4` library is open sourced:
http://opensource.apple.com/source/objc4/objc4-646/

Objective-C

- 1970 - Alan Kay and others developed Smalltalk.
- 1980 - Cox and Love developed set of C preprocessing macros to support Smalltalk OOP features.
- 1988 - NeXT licensed Objective-C, added support to GCC.
- 1996 - Apple (NeXT) built Mac OS using Objective-C.
- 2007 - Objective-C 2.0: non-fragile classes (release 2007).
- 2014 - Swift: Objective-C without C.

Initialisation

- Dynamic messaging, associated objects, swizzling, class posing and more - this is where it all begins.
- `_objc_init`
    - Called by `/usr/lib/libSystem`.
    - Encapsulates all initialization steps.

Environment

`environ_init`

- Reads environment variables, prints help.
- Your chance to specify `OBJC_HELP` and other flags.

Locks

`lock_init`

- Sets up locks requires for thread-safe runtime modifications of:
    - Registered selectors.
    - Classes, methods, protocols, caches etc.
    - Temporary list of classes and protocols to call `+load` on.

Exceptions

`exception_init`

- Fairly straightforward exception setup with `std::set_terminate`.
- `_objc_default_uncaught_exception_handler` expected to be overridden by Foundation.

`dyld` handlers

- `unmap_image`
  - Registering first to be prepared if someone unloads image during the `+load`.
  - Finds appropriate headers to unload & removes classes from `loadable_classes` list.
  - Frees classes & their `isa`.

`dyld` handlers

- `map_images` & `load_images`
  - Process the given images which are being mapped in by dyld.
  - All class registration and fixups are performed, and `+load` methods are called.
  - List of headers is in bottom-up order.

Thank you.

One more thing.

```
[obj message];
```

```
objc_msgSend(obj, @selector(message));
```

objc_msgSend

Any message passing can be rewritten using `objc_msgSend`

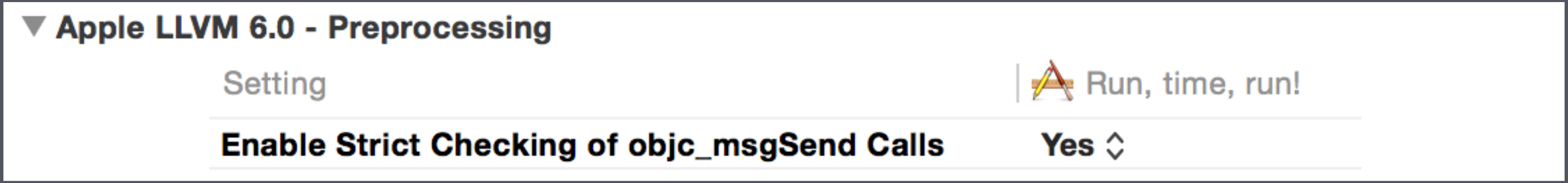
```
// safe release
while ([obj retainCount] > 0) {
    [obj release];
}
```

objc_msgSend

Any message passing can be rewritten using objc_msgSend

```
// fast safe release
while (objc_msgSend(obj, @selector(retainCount)) > 0) {
    (objc_msgSend(obj, @selector(release));
}
```

So you want to call objc_msgSend directly



▼ Apple LLVM 6.0 - Preprocessing

Setting	Run, time, run!
Enable Strict Checking of objc_msgSend Calls	Yes ⌃⌄

```
// fast safe release
typedef NSUInteger (*retain_count_t)(id, SEL);
typedef NSUInteger (*release_t)(id, SEL);

while (((retain_count_t)objc_msgSend)(obj, sel_getUid("retainCount")) > 0) {
    ((release_t)objc_msgSend)(obj, sel_getUid("release"));
}
```

The Objective-C language defers as many decisions as it can from compile time and link time to runtime

-- Objective-C Runtime Reference.

`objc_msgSend`

- Group of methods.
- Declared in `message.h`.
- Implemented in `objc-msg-x86_64.s`. Wait, what?
  - `arm`, `arm64`
  - `i386`, `simulator-i386`
  - `x86_64`, `simulator-x86_64`
  - `win32`

objc_msgSend

```
    ENTRY objc_msgSend
    MESSENGER_START

    cbz r0, LNilReceiver_f

    ldr r9, [r0]        // r9 = self->isa
    CacheLookup NORMAL
    // calls IMP or LCacheMiss

LCacheMiss:
    MESSENGER_END_SLOW
    ldr r9, [r0, #ISA]      // class = receiver->isa
    b   __objc_msgSend_uncached

LNilReceiver:
    mov     r1, #0
    MESSENGER_END_NIL
    bx      lr

LMsgSendExit:
    END_ENTRY objc_msgSend
```

`objc_msgSend`

- Tail optimised
- Dropped support for vtable
- ????

`objc_msgSend`: fast path.

- ~~Check for ignored selectors (GC) and short-circuit.~~
- Check for `nil` target & tagged pointer.
    - ~~Jump to `nil` receiver handler or cleanup and return.~~
    - Return `nil` (or primitive equivalent).
- Search the class's method cache for the method `IMP`.

`objc_msgSend`: slow path

Used once per unique selector per class.

- Method is not cached. Lookup the method `IMP` in the class itself.
- If no `IMP` found, try to:
    - Resolve methods using `+resolveClassMethod` or `+resolveInstanceMethod`.
    - Forward using forwarding mechanism.

⚠️ **Warning**: The Objective-C runtime lets you uncover many private implementation details of system classes. You must not use any of this information in your final product.

-- Technical Note TN2239 "iOS Debugging Magic"

```
-[UIViewController
attentionClassDumpUser:yesItsUsAga
in:althoughSwizzlingAndOverridingP
rivateMethodsIsFun:itWasntMuchFunW
henYourAppStoppedWorking:pleaseRef
rainFromDoingSoInTheFutureOkayThan
ksBye:]
```

`libextobjc`

Runtime and compiler magic at its best.

…extends the dynamism of the Objective-C programming language to support additional patterns present in other programming languages (including those that are not necessarily object-oriented).

-- https://github.com/jspahrsummers/libextobjc.

@keypath

Allows compile-time verification of key paths.

```
@interface MyClass : NSObject
+ (BOOL)classProperty;
@property (nonatomic, assign) NSUInteger someUniqueProperty;
@property (nonatomic, copy) NSArray /*  MyClass */ *collection;
@end

@keypath(MyClass, classProperty);
// @"classProperty"

@collectionKeypath(obj.collection, MyClass.new, someUniqueProperty);
// @"collection.someUniqueProperty"
```

@onExit

Defines some code to be executed when the current scope exits.

```
__block BOOL cleanupBlockRun = NO;
@try {
    @onExit {
        cleanupBlockRun = YES;
    };
    [NSException raise:@"Wild exception" format:@"Absolutely unexpected"];
} @catch (NSException *exception) {
    // your recovery code
} @finally {
    // cleanupBlockRun == YES
}
```

EXTNil

`nil` value you can add to collections.

```
NSDictionary *dictionary = @{@"foo":[EXTNil null]};
dictionary[@"foo"];
// [EXTNil null];

[(NSValue *)dictionary[@"foo"] CGRectValue];
// (CGRect){{0, 0}, {0, 0}}
```

Demo