# Lessons Learned Wrapping an Objective-C Framework in Swift

## NSLondon

JP Simard, *@simjp*, *realm.io*

# Why?

# Why?

## API Improvements

```
// Go from this:
func objcFunc(a: AnyObject!, b: NSArray!) -> NSString!
// to this:
func swiftFunc(a: ClassA?, b: [Int]) -> String?

// Or using autoclosures, generics,
// default parameter values, making types
// and optionality explicit, etc.
```

# Why?

## To Expose Unmappable Interfaces

```
// Objective-C variable arguments aren't exposed to Swift
+ (RLMResults *)objectsWhere:(NSString *)format, ...;
// neither are #defines
#define NSStringFromBool(b) (b ? @"YES" : @"NO")
// or certain declarations
NSVariableStatusItemLength
```

# Why Not Rewrite in Swift?

1. Lazy
2. No significant gains from the language
3. Existing codebase complex and/or mature
4. Yup, still lazy

# Step 1: Make dynamic objc framework

# Porting an API

```
RLMObject                                  // ObjC
Realm.Object                               // Swift
RLMObject.allObjects()                     // ObjC
objects<T: RLMObject>(type: T.Type)        // Swift
```

# Packaging

1. Dynamic framework, logical
2. Can't statically link Objective-C framework in Swift framework (no modules)

# Exposing Just the Right Interfaces

# Exposing Just the Right Interfaces

## Making sure Objective-C is still import-able

1. Module Map needs to be preserved
2. Must be a dynamic framework

# Step 2: Setup Swift framework target

# Make Sure Your Framework Compiles ...

— when building standalone
— when building a target which links against it

# Exposing Just the Right Interfaces

## Exposing private interfaces just for Swift binding

**Unfortunately there is no private bridging headers for framework targets**

# Exposing Just the Right Interfaces

## Exposing private interfaces just for Swift binding

```
framework module Realm {
    umbrella header "Realm.h"

    export *
    module * { export * }

    explicit module Private {
        header "RLMRealm_Private.h"
    }
}
```

# Exposing Just the Right Interfaces

## Don't subclass

```
import Realm
class Realm: RLMRealm {
    // What's wrong with this?
}
```

# Exposing Just the Right Interfaces

## Don't subclass. Wrap!

```
import Realm
class Realm {
    private var rlmRealm: RLMRealm

    private init(rlmRealm: RLMRealm) {
        self.rlmRealm = rlmRealm
    }
}
```

# Testing

1. Don't test functionality, test interfaces
2. Write equivalency tests

# Documentation

1. Compare Objective-C & Swift interfaces side by side
2. Copy docs from Objective-C
3. Convert doxygen format to ReST

# Publishing

— Carthage
— CocoaPods

# Links

— **This talk:** *github.com/jpsim/talks*
— **Realm:** *github.com/realm/realm-cocoa*
— **RealmSwift:** *realm/realm-cocoa/tree/al-swift*
— **jazzy:** *github.com/realm/jazzy*

# Thank You!

```
NSLondon().questions?.askThem!
```

**JP Simard, *@simjp*, *realm.io***