Swift <-> ObjC
Interoperability in Practice

Nikita Lutsenko

@nlutsenko

Facebook, Parse, Banana!

Swift <-> ObjC Interoperability

Swift <-> ObjC

l14y

Agenda

Agenda

→ ObjC -> Swift

Agenda

→ ObjC -> Swift

→ Swift <- ObjC

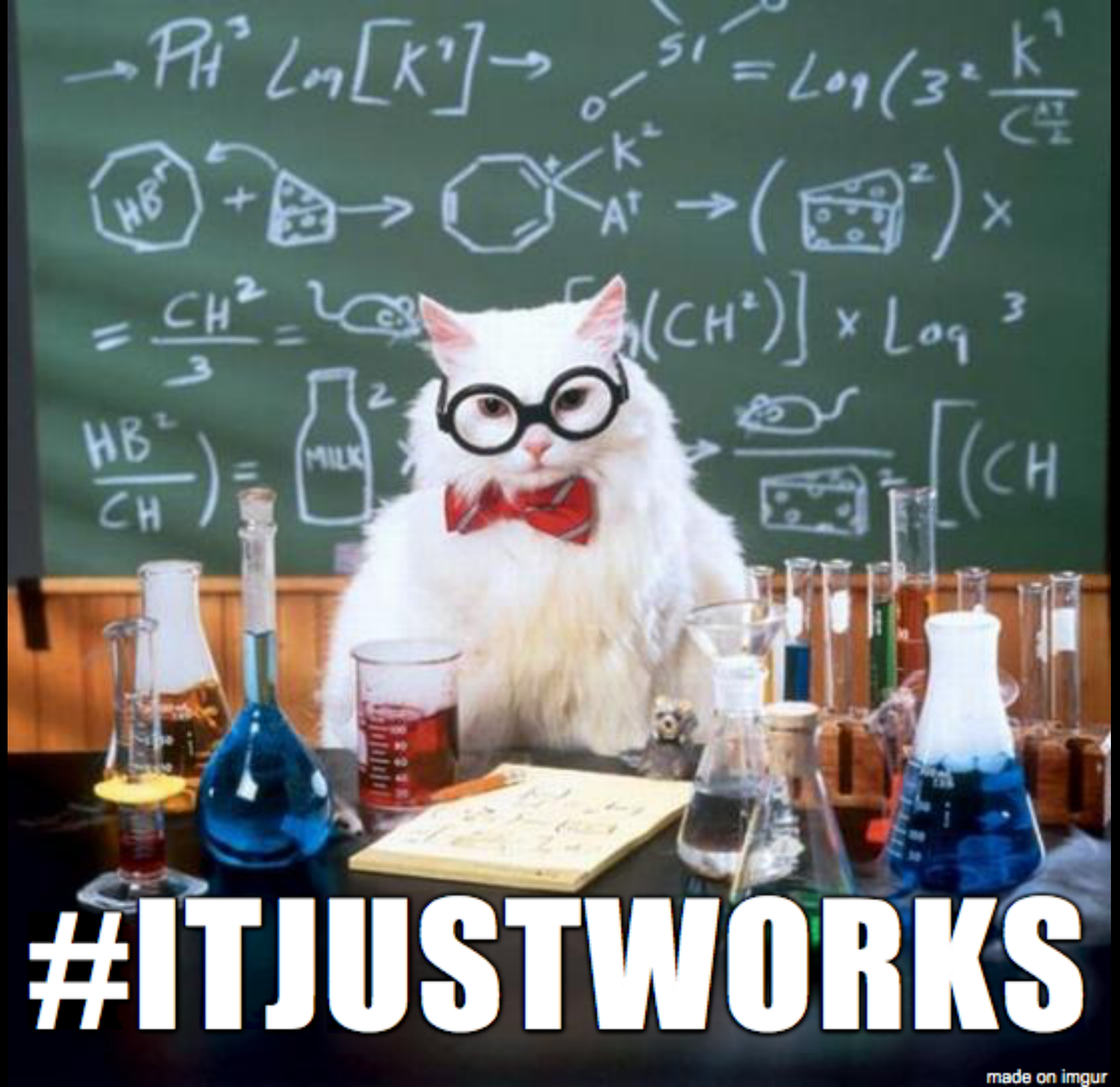
Agenda

→ ObjC -> Swift

→ Swift <- ObjC

→ Swift <-> ObjC

ObjC -> Swift


#ITJUSTWORKS
made on imgur

ObjC -> Swift

ObjC -> Swift

→ #ItJustWorks

ObjC -> Swift

→ #ItJustWorks

→ 100% of API is `@available()`

ObjC -> Swift

→ #ItJustWorks

→ 100% of API is `@available()`

→ Focus on:

→ Swifty Names

→ Type Safety

ObjC -> Swift

→ #ItJustWorks

→ 100% of API is `@available()`

→ Focus on:

→ Swifty Names

→ Type Safety

→ Much better with Swift 3.0

ObjC -> Swift 2.*

ObjC -> Swift 2.*

→ Nullability Annotations

ObjC -> Swift 2.*

→ Nullability Annotations

→ Error Handling

ObjC -> Swift 2.*

→ Nullability Annotations

→ Error Handling

→ `NS_SWIFT_NAME`

ObjC -> Swift 2.*

→ Nullability Annotations

→ Error Handling

→ `NS_SWIFT_NAME`

→ `NS_SWIFT_UNAVAILABLE`

ObjC -> Swift 2.*

→ Nullability Annotations

→ Error Handling

→ `NS_SWIFT_NAME`

→ `NS_SWIFT_UNAVAILABLE`

→ `NS_REFINED_FOR_SWIFT`

ObjC -> Swift 2.*

→ Nullability Annotations

→ Error Handling

→ `NS_SWIFT_NAME`

→ `NS_SWIFT_UNAVAILABLE`

→ `NS_REFINED_FOR_SWIFT`

→ `NS_SWIFT_NOTHROW`

ObjC

```
- (BOOL)performRequest:(NSURLRequest *)request;
- (BOOL)performRequest:(NSURLRequest *)request
                 error:(NSError **)error;
```

Swift 2.*

```
func performRequest(request: NSURLRequest!) -> Bool
func performRequest(request: NSURLRequest!, error: ()) throws
```

ObjC

```objc
NS_ASSUME_NONNULL_BEGIN

- (BOOL)performRequest:(NSURLRequest *)request;
- (BOOL)performRequest:(NSURLRequest *)request
                 error:(NSError **)error;

NS_ASSUME_NONNULL_END
```

Swift 2.*

```swift
public func performRequest(request: NSURLRequest) -> Bool
public func performRequest(request: NSURLRequest, error: ()) throws
```

ObjC

```
- (BOOL)performRequest:(nullable NSURLRequest *)request;

- (BOOL)performRequest:(nullable NSURLRequest *)request
                 error:(NSError **)error;
```

Swift 2.*

```
public func performRequest(request: NSURLRequest?) -> Bool
public func performRequest(request: NSURLRequest?, error: ()) throws
```

ObjC

```
- (BOOL)performRequest:(nullable NSURLRequest *)request
NS_SWIFT_NAME(perform(request:));

- (BOOL)performRequest:(nullable NSURLRequest *)request
                 error:(NSError **)error
NS_SWIFT_NAME(perform(request:));
```

Swift 2.*

```
public func perform(request request: NSURLRequest?) -> Bool
public func perform(request request: NSURLRequest?) throws
```

ObjC

```
- (BOOL)performRequest:(nullable NSURLRequest *)request
NS_SWIFT_UNAVAILABLE("");

- (BOOL)performRequest:(nullable NSURLRequest *)request
                 error:(NSError **)error
NS_SWIFT_NAME(perform(request:));
```

Swift 2.*

```
public func perform(request request: NSURLRequest?) throws
```

ObjC

```
- (BOOL)performRequest:(nullable NSURLRequest *)request
NS_SWIFT_UNAVAILABLE("");

- (BOOL)performRequest:(nullable NSURLRequest *)request
                 error:(NSError **)error;
```

Swift 2.*

```
public func performRequest(request: NSURLRequest?) throws
```

ObjC

```
- (BOOL)performRequest:(nullable NSURLRequest *)request
NS_SWIFT_UNAVAILABLE("");

- (BOOL)performRequest:(nullable NSURLRequest *)request
                 error:(NSError **)error
NS_SWIFT_NAME(perform(request:));
```

Swift 2.*

```
public func perform(request request: NSURLRequest?) throws
```

ObjC

```
- (void)getFirstName:(NSString **)firstName
            lastName:(NSString **)lastName;
```

Swift 2.*

```
func getFirstName(firstName: AutoreleasingUnsafeMutablePointer<NSString?>,
                  lastName: AutoreleasingUnsafeMutablePointer<NSString?>)
```

ObjC

```
- (void)getFirstName:(NSString **)firstName
            lastName:(NSString **)lastName
NS_SWIFT_NAME(get(firstName:lastName:));
```

Swift 2.*

```
func getFirstName(firstName: AutoreleasingUnsafeMutablePointer<NSString?>,
                  lastName: AutoreleasingUnsafeMutablePointer<NSString?>)
```

ObjC

```
- (void)getFirstName:(NSString **)firstName
            lastName:(NSString **)lastName NS_REFINED_FOR_SWIFT;
```

Swift 2.*

```
extension Person {
  var names: (firstName: String?, lastName: String?) {
      var firstName: NSString? = nil
      var lastName: NSString? = nil
      __getFirstName(&firstName, lastName: &lastName)
      return (firstName: firstName as String?, lastName: lastName as String?)
  }
}
```

ObjC -> Swift 3.0

ObjC -> Swift 3.0

→ "The Grand Rename"

ObjC -> Swift 3.0

→ "The Grand Rename"

→ Objective-C Lightweight Generics

ObjC -> Swift 3.0

→ "The Grand Rename"

→ Objective-C Lightweight Generics

→ `NS_NOESCAPE`/`CF_NOESCAPE`

ObjC -> Swift 3.0

→ "The Grand Rename"

→ Objective-C Lightweight Generics

→ `NS_NOESCAPE`/`CF_NOESCAPE`

→ `NS_EXTENSIBLE_STRING_ENUM`

ObjC

```
@interface MyNetworkController<Request: NSURLRequest *> : NSObject

- (void)performRequest:(nullable Request)request;

@end
```

Swift 2.*

```
public class MyNetworkController : NSObject {
    public func performRequest(request: NSURLRequest?)
}
```

ObjC

```objc
@interface MyNetworkController<Request: NSURLRequest *> : NSObject

- (void)performRequest:(nullable Request)request;

@end
```

Swift 3

```swift
public class MyNetworkController<Request : NSURLRequest> : NSObject {
    public func perform(_ request: Request?)
}
```

ObjC

```
- (void)executeActions:(void (^)(void))actions;
```

Swift 3

```
func executeActions(_ actions: () -> Void)
```

ObjC

```
- (void)executeActions:(void (NS_NOESCAPE ^)(void))actions;
```

Swift 3

```
func executeActions(_ actions: @noescape () -> Void)
```

ObjC

```
- (void)executeActions:(void (NS_NOESCAPE ^)(void))actions
NS_SWIFT_NAME(execute(actions:));
```

Swift 3

```
func execute(actions: (@noescape () -> Void))
```

ObjC

```
typedef NSString *NSRunLoopMode;
extern NSRunLoopMode const NSDefaultRunLoopMode;
```

ObjC

```
typedef NSString *NSRunLoopMode NS_EXTENSIBLE_STRING_ENUM;
extern NSRunLoopMode const NSDefaultRunLoopMode;
```

Swift 3

```
struct RunLoopMode : RawRepresentable { }

extension RunLoopMode {
  public static let defaultRunLoopMode: RunLoopMode
}
```

ObjC -> Swift

ObjC -> Swift

→ U

ObjC -> Swift

→ U

→ U

ObjC -> Swift

→ U

→ U

→ R

ObjC -> Swift

→ U

→ U

→ R

→ R

ObjC -> Swift

→ U

→ U

→ R

→ R

→ R

ObjC -> Swift

→ U

→ U

→ R

→ R

→ R

→ R

ObjC -> Swift

ObjC -> Swift

→ Use Nullability Annotations

ObjC -> Swift

→ Use Nullability Annotations

→ Use Objective-C Generics

ObjC -> Swift

→ Use Nullability Annotations

→ Use Objective-C Generics

→ Rename (`NS_SWIFT_NAME`)

ObjC -> Swift

→ Use Nullability Annotations

→ Use Objective-C Generics

→ Rename (`NS_SWIFT_NAME`)

→ Refine (`NS_REFINED_FOR_SWIFT`)

ObjC -> Swift

→ Use Nullability Annotations

→ Use Objective-C Generics

→ Rename (`NS_SWIFT_NAME`)

→ Refine (`NS_REFINED_FOR_SWIFT`)

→ Rinse (`NS_SWIFT_UNAVAILABLE`)

ObjC -> Swift

→ Use Nullability Annotations

→ Use Objective-C Generics

→ Rename (`NS_SWIFT_NAME`)

→ Refine (`NS_REFINED_FOR_SWIFT`)

→ Rinse (`NS_SWIFT_UNAVAILABLE`)

→ Repeat


I LOVE THIS JOB

Swift -> ObjC

Swift -> ObjC

Swift -> ObjC

→ No Tuples

Swift -> ObjC

→ No Tuples

→ No Generics

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

→ No Swift Enums

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

→ No Swift Enums

→ No Swift Structs

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

→ No Swift Enums

→ No Swift Structs

→ No Global Variables

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

→ No Swift Enums

→ No Swift Structs

→ No Global Variables

→ No Free-standing Functions

Swift -> ObjC

→ No Tuples

→ No Generics

→ No Typealiases

→ No Swift Enums

→ No Swift Structs

→ No Global Variables

→ No Free-standing Functions

→ Only ObjC Runtime Compatible


NO FUN ALLOWED?
GOOD.
quickmeme.com

Swift -> ObjC

Swift -> ObjC

→ Subclasses of `NSObject`

Swift -> ObjC

→ Subclasses of `NSObject`

→ Not `private` (`fileprivate`)

Swift -> ObjC

→ Subclasses of `NSObject`

→ Not `private` (`fileprivate`)

→ Not `@nonobjc`

Swift -> ObjC

→ Subclasses of `NSObject`

→ Not `private` (`fileprivate`)

→ Not `@nonobjc`

→ `@objc` protocols

@objc & @nonobjc

`@objc` & `@nonobjc`

→ Subclasses of `NSObject`

`@objc` & `@nonobjc`

→ Subclasses of `NSObject`

→ Explicit export to ObjC

`@objc` & `@nonobjc`

→ Subclasses of `NSObject`

→ Explicit export to ObjC

→ Concrete extensions

`@objc` & `@nonobjc`

→ Subclasses of `NSObject`

→ Explicit export to ObjC

→ Concrete extensions

→ `@objc` protocols

`@objc` & `@nonobjc`

→ Subclasses of `NSObject`

→ Explicit export to ObjC

→ Concrete extensions

→ `@objc` protocols

→ `Int` enums

`@objc` **protocols**

`@objc` **protocols**

→ Special case of protocol

`@objc` **protocols**

→ Special case of protocol

→ Weaker and not-Swifty

`@objc` **protocols**

→ Special case of protocol

→ Weaker and not-Swifty

→ Supports `optional`

`@objc` **protocols**

→ Special case of protocol

→ Weaker and not-Swifty

→ Supports `optional`

→ Extensions are inaccessible from ObjC

Value Types via `_ObjectiveCBridgeable`

Value Types via `_ObjectiveCBridgeable`

→ Powers Foundation

Value Types via `_ObjectiveCBridgeable`

→ Powers Foundation

→ Implementation Detail Protocol

Value Types via `_ObjectiveCBridgeable`

→ Powers Foundation

→ Implementation Detail Protocol

→ ObjC Reference Type <-> Swift Types

_ObjectiveCBridgeable

```
public protocol _ObjectiveCBridgeable {
  associatedtype _ObjectiveCType : AnyObject

  static func _isBridgedToObjectiveC() -> Bool

  func _bridgeToObjectiveC() -> _ObjectiveCType

  static func _forceBridgeFromObjectiveC(_ source: _ObjectiveCType, result: inout Self?)

  static func _conditionallyBridgeFromObjectiveC(
    _ source: _ObjectiveCType,
    result: inout Self?
  ) -> Bool

  static func _unconditionallyBridgeFromObjectiveC(_ source: _ObjectiveCType?) -> Self
}
```

Swift <-> ObjC

Swift <-Pitfalls-> ObjC

Swift <-Pitfalls-> ObjC

Swift <-Pitfalls-> ObjC

→ Type Compatibility

Swift <-Pitfalls-> ObjC

→ Type Compatibility

→ Objective-C Runtime

Swift <-Pitfalls-> ObjC

→ Type Compatibility

→ Objective-C Runtime

→ Closure -> Block Performance

Type Compatibility

```
NSArray<NSString *> *names = @[ @"John", @"Jane" ];
NSArray<id<NSCopying>> *array = names;
```

Type Compatibility

```
let names: [String] = ["John", "Jane"]
let array: [CustomDebugStringConvertible] = names
```

Type Compatibility

```
let names: [String] = ["John", "Jane"]
let array: [CustomDebugStringConvertible] = names
```

```
fatal error: can't unsafeBitCast between types of
different sizes
```

Type Compatibility

```
let names: [String] = ["John", "Jane"]
let array: [CustomDebugStringConvertible] = names.map({
  $0 as CustomDebugStringConvertible
})
```

Objective-C Runtime in Swift

```swift
@objc class Animal: NSObject {
    let name = "Animal"
    override init() {
        super.init()
    }
}
```

```objc
@interface Plant: NSObject
@end

@implementation Plant

- (NSString *)name {
    return @"Plant";
}

@end
```

Objective-C Runtime in Swift

```
// ObjC
SEL selector = @selector(name);

Method originalMethod = class_getInstanceMethod([Animal class], selector);
Method swizzledMethod = class_getInstanceMethod([Plant class], selector);

method_exchangeImplementations(originalMethod, swizzledMethod);

// ObjC
NSLog(@"I am %@!", name);

// Swift
print("I am \(name)!")
```

Objective-C Runtime in Swift

```
// ObjC
SEL selector = @selector(name);

Method originalMethod = class_getInstanceMethod([Animal class], selector);
Method swizzledMethod = class_getInstanceMethod([Plant class], selector);

method_exchangeImplementations(originalMethod, swizzledMethod);

// ObjC
NSLog(@"I am %@!", name); // I am Plant!

// Swift
print("I am \(name)!") // I am Animal!
```


HELLO
YES, THIS IS DOG

Objective-C Runtime in Swift

```swift
@objc class Animal: NSObject {
    let name = "Animal"
    override init() {
        super.init()
    }
}
```

```objc
@interface Plant: NSObject
@end

@implementation Plant

- (NSString *)name {
    return @"Plant";
}

@end
```

Objective-C Runtime in Swift

```
@objc class Animal: NSObject {
    dynamic let name = "Animal"
    override init() {
        super.init()
    }
}
```

```
@interface Plant: NSObject
@end

@implementation Plant

- (NSString *)name {
    return @"Plant";
}

@end
```

Objective-C Runtime in Swift

```
// ObjC
SEL selector = @selector(name);

Method originalMethod = class_getInstanceMethod([Animal class], selector);
Method swizzledMethod = class_getInstanceMethod([Plant class], selector);

method_exchangeImplementations(originalMethod, swizzledMethod);

// ObjC
NSLog(@"I am %@!", name);

// Swift
print("I am \(name)!")
```

Objective-C Runtime in Swift

```
// ObjC
SEL selector = @selector(name);

Method originalMethod = class_getInstanceMethod([Animal class], selector);
Method swizzledMethod = class_getInstanceMethod([Plant class], selector);

method_exchangeImplementations(originalMethod, swizzledMethod);

// ObjC
NSLog(@"I am %@!", name); // I am Plant!

// Swift
print("I am \(name)!") // I am Plant!
```

Yes, this is dog!

Swift <-> ObjC

Swift <-> ObjC

→ You don't need this!

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

→ OpenGL Graphics

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

→ OpenGL Graphics

→ Interacting with C++

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

→ OpenGL Graphics

→ Interacting with C++

→ System calls (server side ftw!)

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

→ OpenGL Graphics

→ Interacting with C++

→ System calls (server side ftw!)

→ Super-über-high-performance

Swift <-> ObjC

→ You don't need this!

→ Migrate to Swift!

→ OpenGL Graphics

→ Interacting with C++

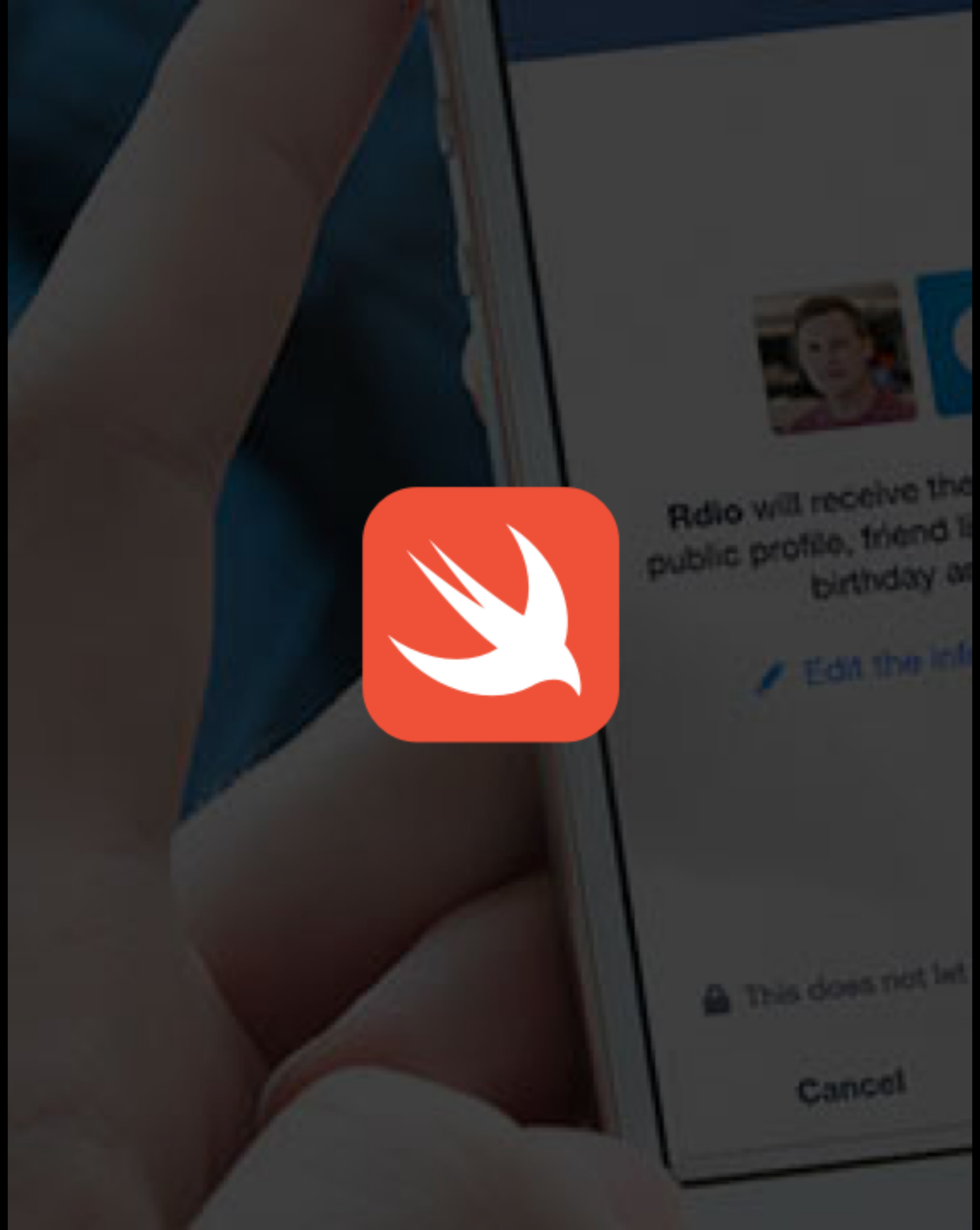
→ System calls (server side ftw!)

→ Super-über-high-performance

→ Support for both languages

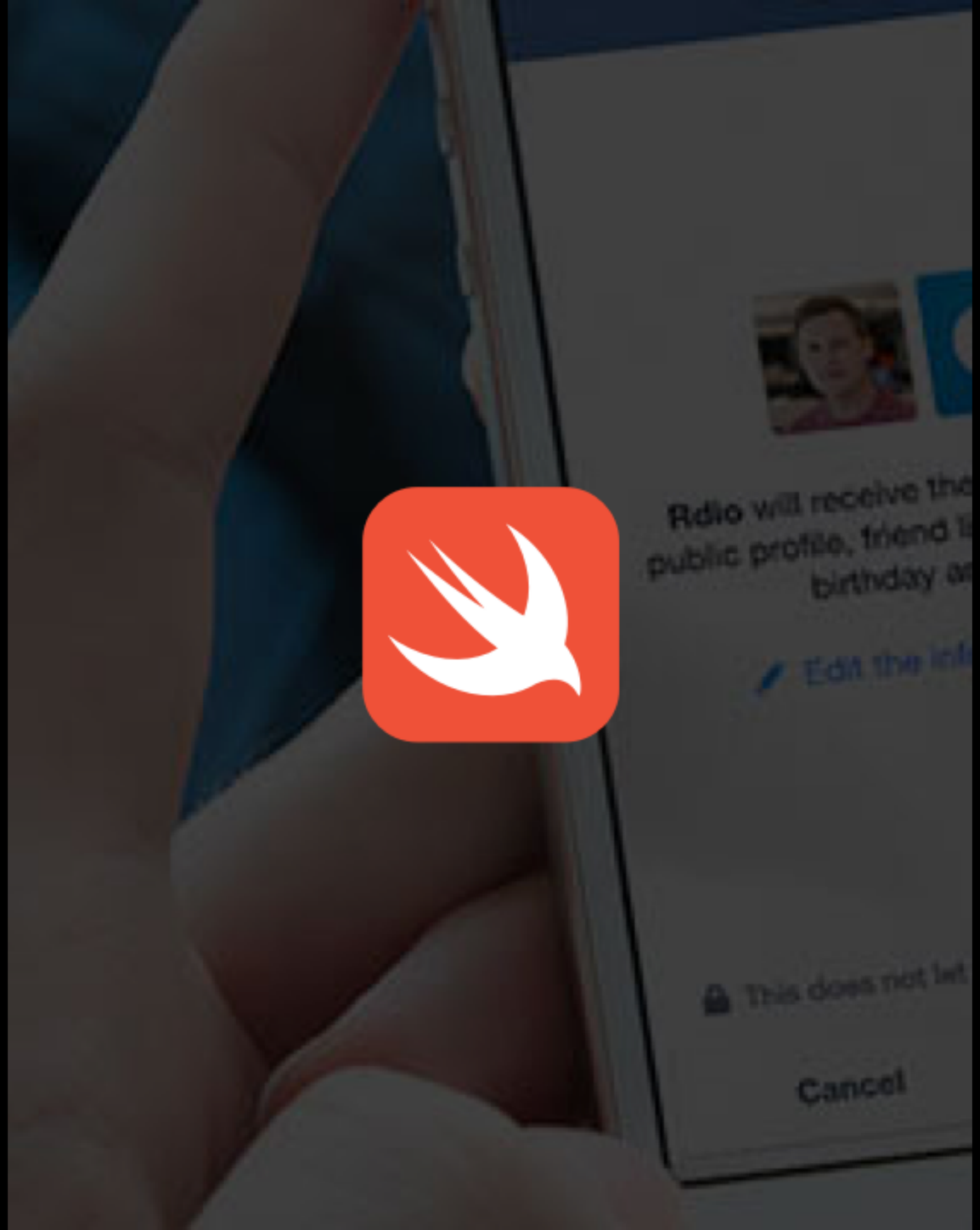
Swift <-> ObjC

@ Facebook



Swift <-> ObjC @ Facebook

→ Swift-taylored Facebook SDK

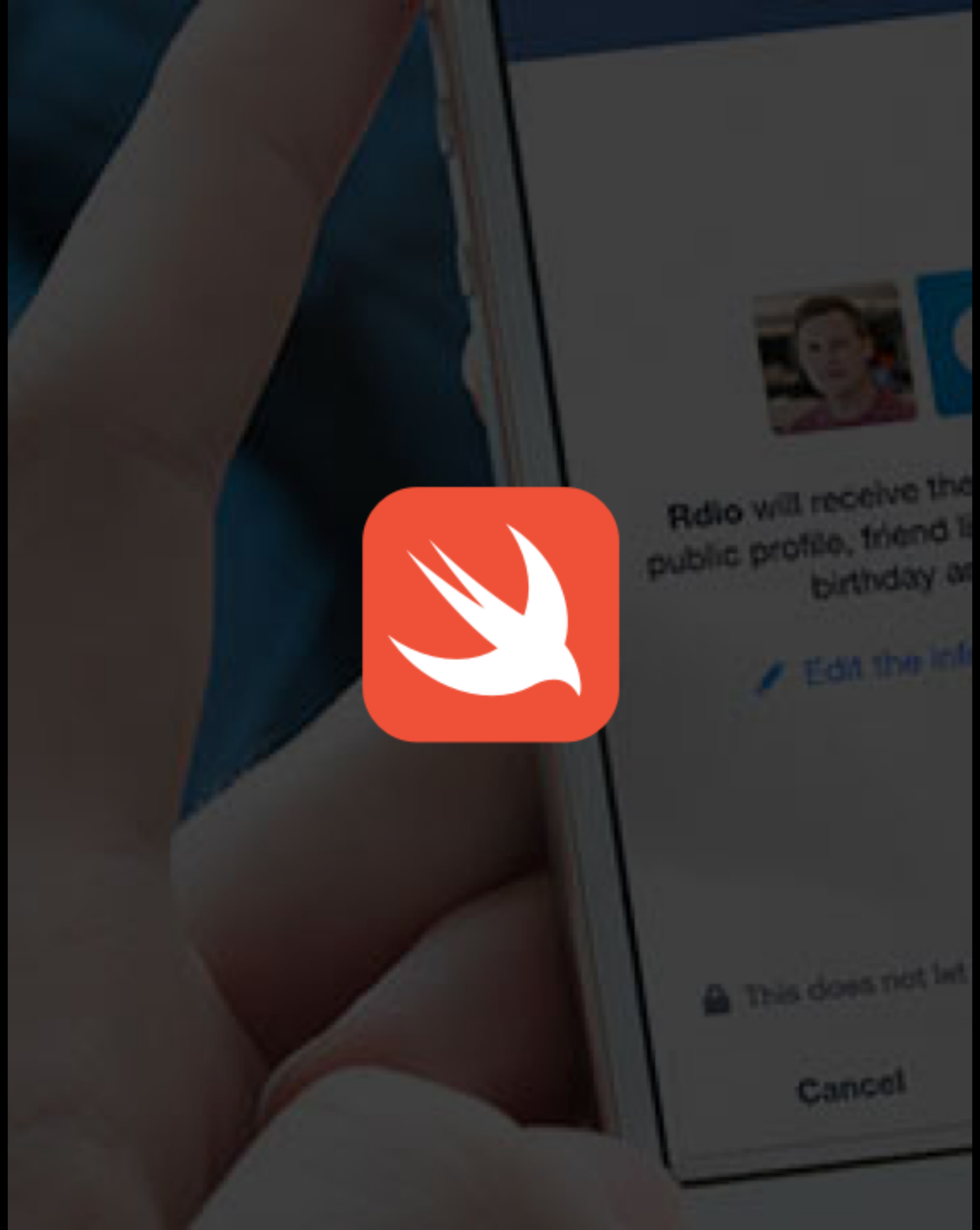


Swift <-> ObjC

@ Facebook

→ Swift-taylored Facebook SDK

→ Builds upon Existing ObjC SDK



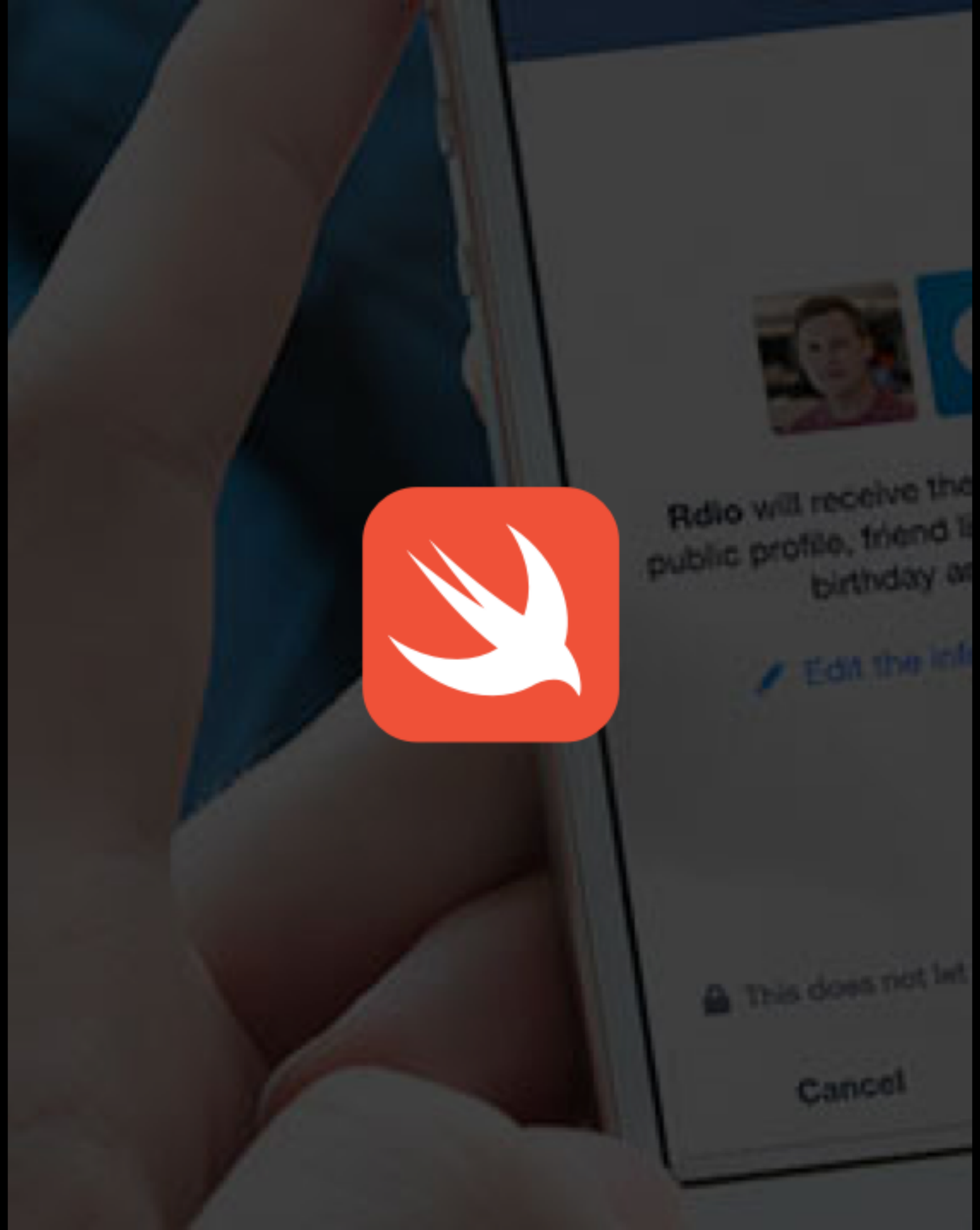
Swift <-> ObjC

@ Facebook

→ Swift-taylored Facebook SDK

→ Builds upon Existing ObjC SDK

→ Native Swift Interface & Types



Swift <-> ObjC

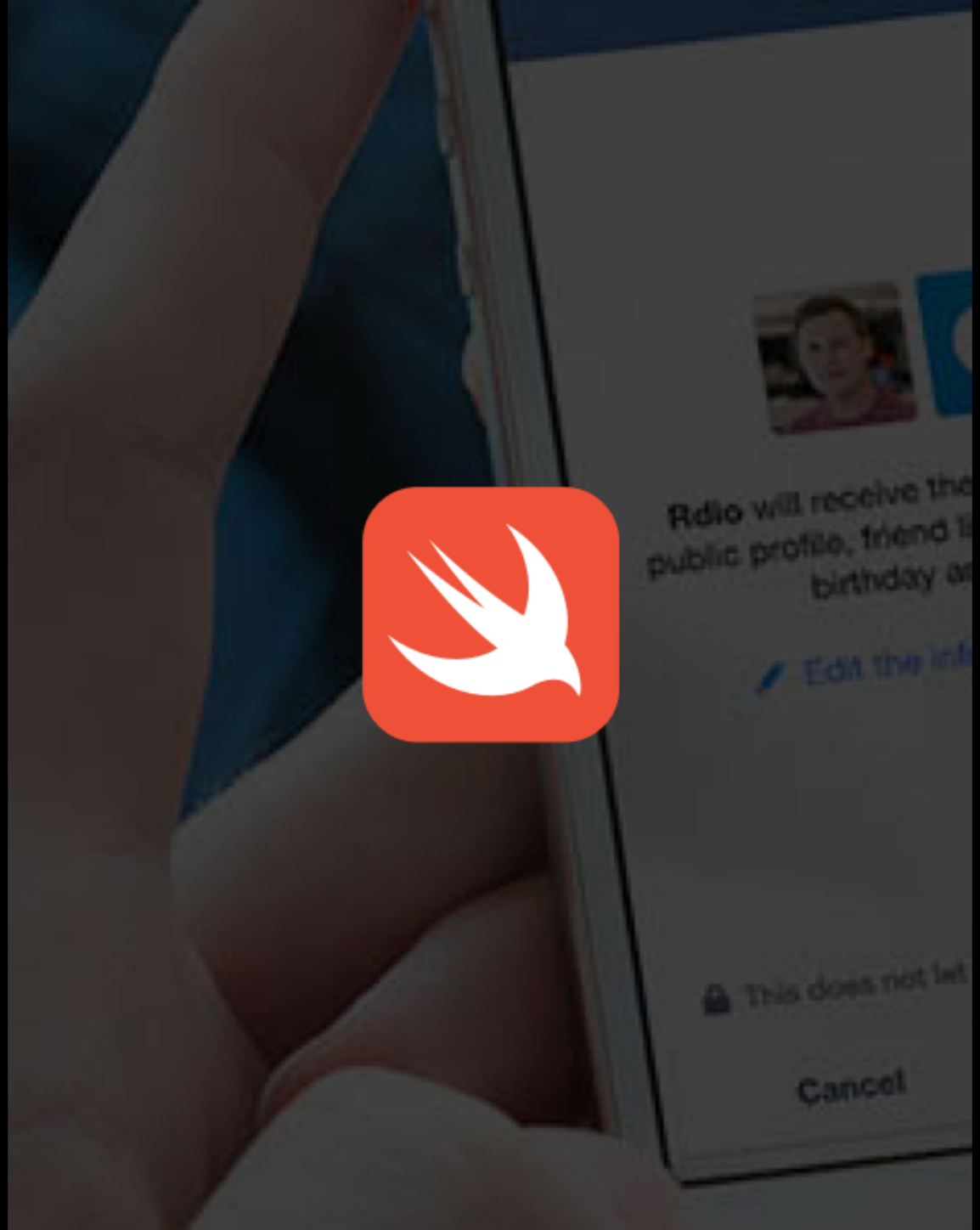
@ Facebook

→ Swift-taylored Facebook SDK

→ Builds upon Existing ObjC SDK

→ Native Swift Interface & Types

→ No exposure to ObjC APIs



Swift <-> ObjC

@ Facebook

Swift <-> ObjC

@ Facebook

→ Structs over Classes

Swift <-> ObjC

@ Facebook

→ Structs over Classes

→ Concrete-typed vs Stringly-typed

Swift <-> ObjC

@ Facebook

→ Structs over Classes

→ Concrete-typed vs Stringly-typed

→ Bridging protocols (`***Delegate`) to Closures

Swift <-> ObjC

@ Facebook

→ Structs over Classes

→ Concrete-typed vs Stringly-typed

→ Bridging protocols (`***Delegate`) to Closures

→ Protocol-oriented (+ default implementation)

Structs over Classes

```
// Create a profile request
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: ["fields" : "name"])
```

Structs over Classes

```
// Create a profile request
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: ["fields" : "name"])

// Start a request
request.startWithCompletionHandler({ _, result, _ in
  // ...
  print(result) // [ 'name': 'Nikita Lutsenko' ]
})
```

Structs over Classes

```
// Create a profile request
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: ["fields" : "name"])

// Start a request
request.startWithCompletionHandler({ _, result, _ in
  // ...
  print(result) // ...
})

request.parameters = ["fields": "id"]
```

Structs over Classes

```
// Create a profile request
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: ["fields" : "name"])

// Start a request
request.startWithCompletionHandler({ _, result, _ in
  // ...
  print(result) // ¯\_(ツ)_/¯
})

request.parameters = ["fields": "id"]
```

Structs over Classes

Structs over Classes

```
// Create a profile request
let request = GraphRequest(graphPath: "me",
                           parameters: ["fields" : "name"])
```

Structs over Classes

```
// Create a profile request
let request = GraphRequest(graphPath: "me",
                           parameters: ["fields" : "name"])

request.start({ _, result in
  // ...
  print(result) // ...
})
```

Structs over Classes

```
// Create a profile request
let request = GraphRequest(graphPath: "me",
                           parameters: ["fields" : "name"])

request.start({ _, result in
  // ...
  print(result) // [ 'name' : 'Nikita Lutsenko' ]
})
```

Structs over Classes

```
// Create a profile request
let request = GraphRequest(graphPath: "me",
                           parameters: ["fields" : "name"])

request.start({ _, result in
  // ...
  print(result) // [ 'name' : 'Nikita Lutsenko' ]
})

request.parameters = ["fields" : "id"]
// ...
```

Structs over Classes

```
// Create a profile request
let request = GraphRequest(graphPath: "me",
                           parameters: ["fields" : "name"])

request.start({ _, result in
  // ...
  print(result) // [ 'name' : 'Nikita Lutsenko' ]
})

request.parameters = ["fields" : "id"]
// Error: Cannot assign to property
```

Structs over Classes

Stringly-typed

Stringly-typed

→ Using `String` (`NSString`) for types

Stringly-typed

→ Using `String` (`NSString`) for types

→ Has limitations on values

Stringly-typed

→ Using `String` (`NSString`) for types

→ Has limitations on values

→ Provides loose runtime guarantees

Stringly-typed

→ Using `String` (`NSString`) for types

→ Has limitations on values

→ Provides loose runtime guarantees

→ ObjC-flavor and JS-taste

Stringly-typed

```
class FBSDKGraphRequest : NSObject {

  var graphPath: String! { get }

  var HTTPMethod: String! { get }

  var apiVersion: String! { get }
}
```

Stringly-typed

```
class FBSDKGraphRequest : NSObject {

  var graphPath: String! { get } // "me"

  var HTTPMethod: String! { get } // GET/POST/DELETE

  var apiVersion: String! { get } // "v2.7"
}
```

Stringly-typed

```
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: [],
                                httpMethod: "GET")

request.startWithCompletionHandler({ _, result, _ in
    print(result) // ...
})
```

Stringly-typed

```
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: [],
                                httpMethod: "GET")

request.startWithCompletionHandler({ _, result, _ in
    print(result) // [ "name" : "Nikita Lutsenko" ]
})
```

Stringly-typed

```
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: [],
                                httpMethod: "GET")

request.startWithCompletionHandler({ _, _, _ in
    print(result) // ...
})
```

Stringly-typed

```
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: [],
                                httpMethod: "YOLO")

request.startWithCompletionHandler({ _, _, _ in
    print(result) // Error!
})
```

Stringly-typed

```
let request = FBSDKGraphRequest(graphPath: "me",
                                parameters: [],
                                httpMethod: "BANANA")

request.startWithCompletionHandler({ _, _, _ in
    print(result) // 🍌
})
```

Stringly-typed

Concrete-typed

```
struct GraphRequest: GraphRequestProtocol {

  let graphPath: String

  let httpMethod: GraphRequestHTTPMethod

  let apiVersion: GraphAPIVersion
}
```

Concrete-typed

```
struct GraphRequest: GraphRequestProtocol {

  let graphPath: String

  let httpMethod: GraphRequestHTTPMethod

  let apiVersion: GraphAPIVersion
}
```

Concrete-typed

```
enum GraphRequestHTTPMethod: String {

  case GET = "GET"
  case POST = "POST"
  case DELETE = "DELETE"

  init?(string: String)
}
```

Concrete-typed

```
struct GraphAPIVersion {

  let stringValue: String

  init(stringLiteral: StringLiteralType)

  init(floatLiteral value: FloatLiteralType)
}
```

Concrete-typed

```
let request = GraphRequest(graphPath: "me",
                           httpMethod: "GET")

request.startWithCompletionHandler({ _, _, _ in
    print(result) // ...
})
```

Concrete-typed

```
let request = GraphRequest(graphPath: "me",
                           httpMethod: "GET") // Error

request.startWithCompletionHandler({ _, _, _ in
    print(result) // ...
})
```

Concrete-typed

```
let request = GraphRequest(graphPath: "me",
                           httpMethod: .GET)

request.startWithCompletionHandler({ _, _, _ in
    print(result) // [ "name" : "Nikita Lutsenko"]
})
```

Concrete-typed

```
let method = GraphRequestHTTPMethod("YOLO") ?? .GET

let request = GraphRequest(graphPath: "me",
                           httpMethod: method)

request.startWithCompletionHandler({ _, _, _ in
    print(result) // [ "name" : "Nikita Lutsenko"]
})
```

Swift <-> ObjC in Practice

Swift <-> ObjC in Practice

→ Large existing codebase

Swift <-> ObjC in Practice

→ Large existing codebase

→ Support for 2 languages

Swift <-> ObjC in Practice

→ Large existing codebase

→ Support for 2 languages

→ Support for latest and greatest

Swift <-> ObjC in Practice

→ Large existing codebase

→ Support for 2 languages

→ Support for latest and greatest

→ ???

Swift <-> ObjC in Practice

→ Large existing codebase

→ Support for 2 languages

→ Support for latest and greatest

→ ???

→ Profit

Thank you!

Questions?

github.com/nlutsenko

twitter.com/nlutsenko

facebook.com/nsunimplemented