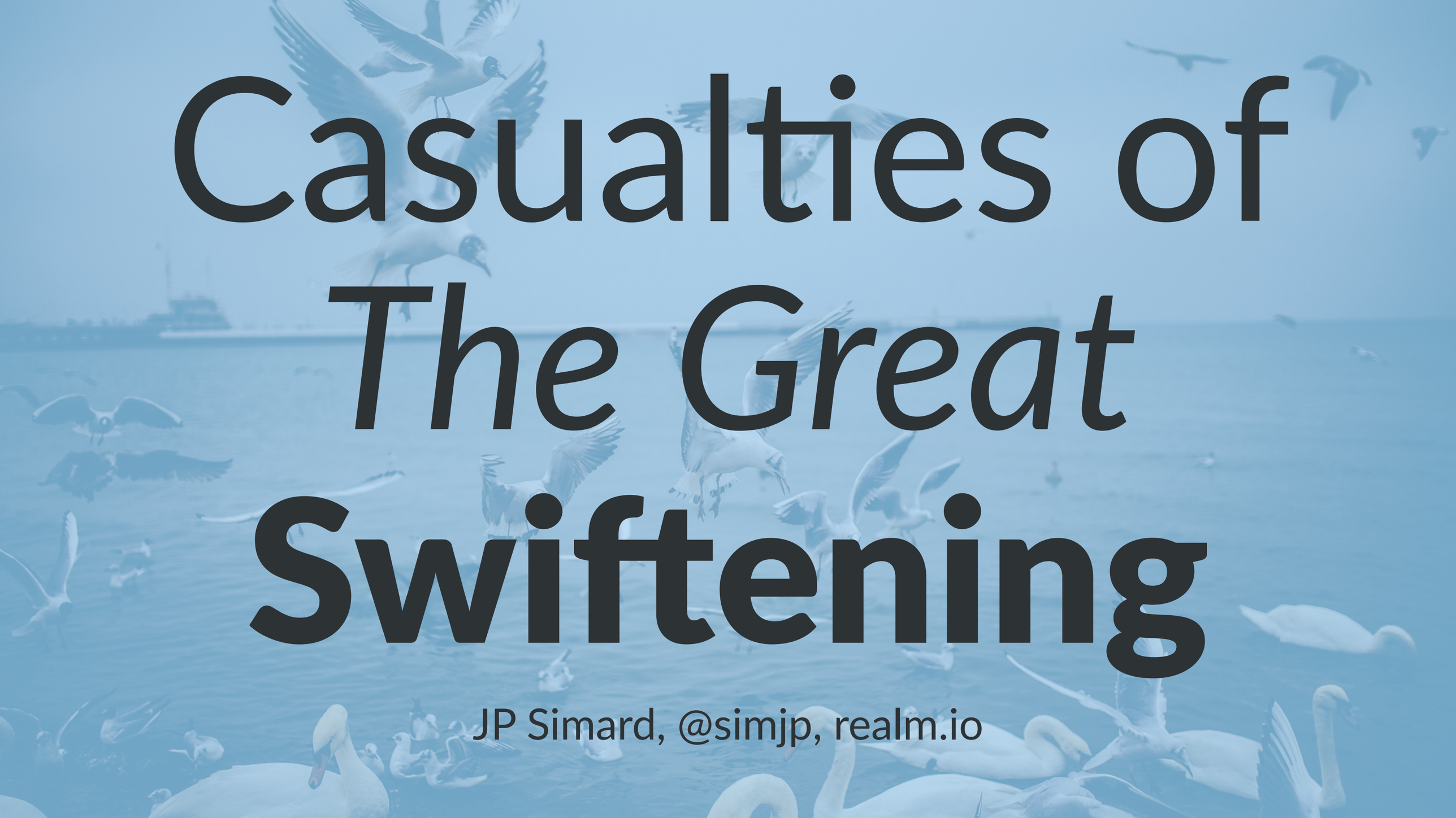


Casualties of *The Great* **Swiftening**

JP Simard, @simjp, realm.io

realm / realm-cocoa

Realm is a mobile database: a replacement for Core Data & SQLite http://realm.io

5,497 commits 101 branches 72 releases 42 contributors

branch: master realm-cocoa / +

Merge pull request #1686 from realm/seg-swift-performance ...

segiddins authored 18 hours ago latest commit c8d1d5dbd9

Realm.xcodeproj	Update performance baselines	10 days ago
Realm	Merge pull request #1686 from realm/seg-swift-performance	18 hours ago
RealmSwift.xcodeproj	[SwiftPerformanceTests] Update baseline for enumerateAndMutateAll sin…	19 hours ago
RealmSwift	Merge pull request #1686 from realm/seg-swift-performance	18 hours ago
docs	fix dash & xcode icons in html docs (fixes #1189)	4 months ago
examples	Fix entitlement file name for case-sensitive file system	10 days ago
plugin	Recombined file and class template installation in install_templates.…	4 months ago
scripts	[Results] import Foundation.NSDate	18 days ago
tools/RealmBrowser	[Browser] Display seconds for dates	a month ago
.dir-locals.el	Project specific Emacs settings updated: Always show trailing white-s…	2 years ago
.gitignore	[build.sh] Run jazzy to generate RealmSwift docs	18 days ago
CHANGELOG.md	[CHANGELOG] Add entry for RLMCollection KVC operations	12 days ago

HTTPS clone URL

https://github.com/realm/

You can clone with HTTPS or Subversion.

Clone in Desktop

Download ZIP

Functional patterns

Protocols and extensions on structs

Pattern matching

Concise syntax

Closures

Generics

Fast iteration

Native collections

Optional types

Operator overloading

Object orientation

Namespaces

Tuples

Type inference

Clear mutability syntax

Read-Eval-Print-Loop (REPL)

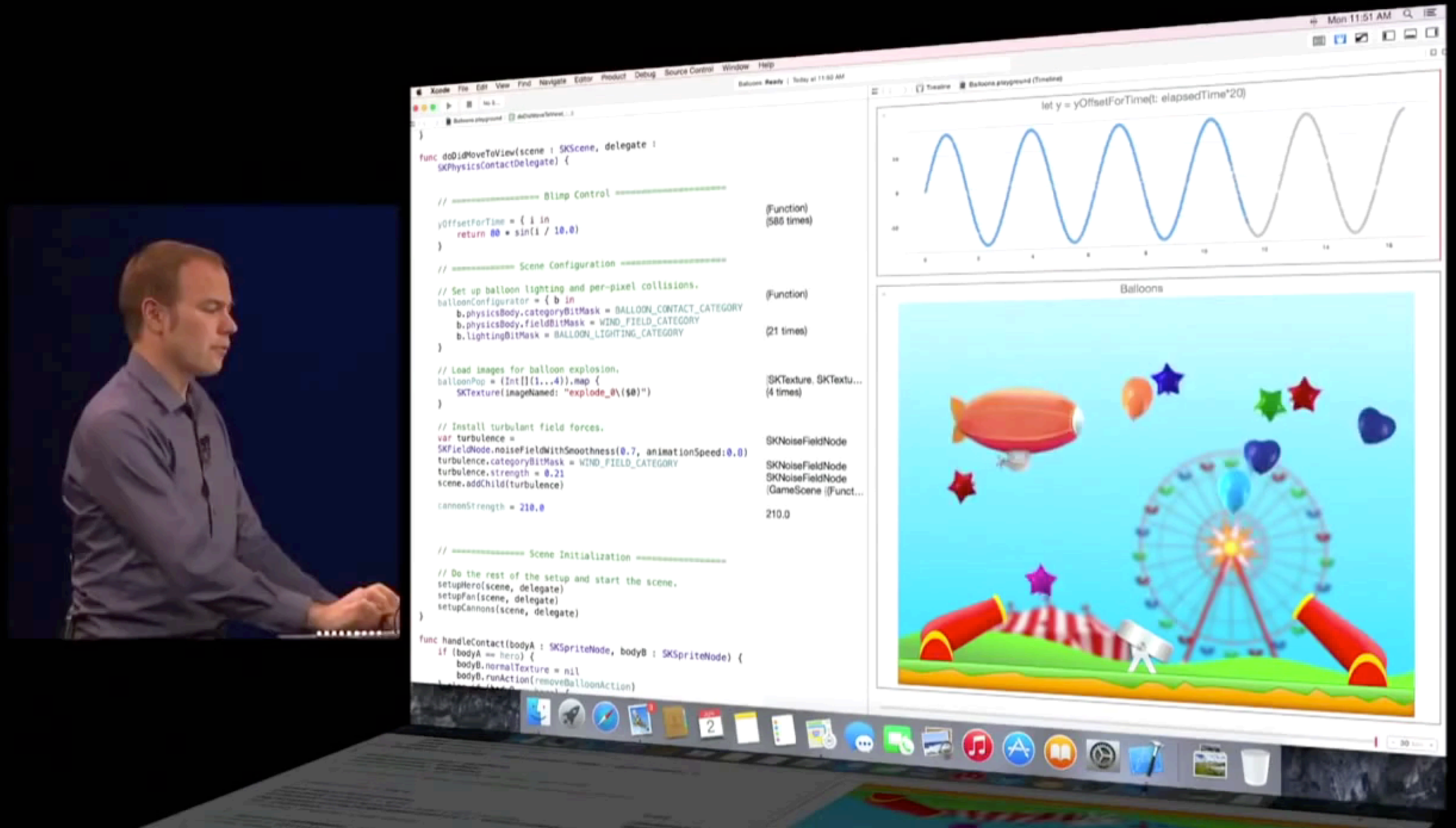
Interactive playground

Multiple return types

Compile to native code

```
if (myDelegate != nil) {
    if ([myDelegate respondsToSelector:
        @selector(scrollViewDidScroll:)]) {
            [myDelegate scrollViewDidScroll:myScrollView];
    }
}
```

```
myDelegate?.scrollViewDidScroll?(myScrollView)
```


Mon 11:51 AM
Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Balloons playground
Balloons: Ready | Today at 11:50 AM
Timeline
Balloons playground (Timeline)
}
func doDidMoveToView(scene : SKScene, delegate :
SKPhysicsContactDelegate) {
// ================== Blimp Control ====================
yOffsetForTime = { i in
return 80 * sin(i / 10.0)
}
// ============== Scene Configuration ==================
// Set up balloon lighting and per-pixel collisions.
balloonConfigurator = { b in
b.physicsBody.categoryBitMask = BALLOON_CONTACT_CATEGORY
b.physicsBody.fieldBitMask = WIND_FIELD_CATEGORY
b.lightingBitMask = BALLOON_LIGHTING_CATEGORY
}
// Load images for balloon explosion.
balloonPop = (Int[](1...4)).map {
SKTexture(imageNamed: "explode_0\($0)")
}
// Install turbulant field forces.
var turbulence =
SKFieldNode.noiseFieldWithSmoothness(0.7, animationSpeed:0.0)
turbulence.categoryBitMask = WIND_FIELD_CATEGORY
turbulence.strength = 0.21
scene.addChild(turbulence)
cannonStrength = 210.0
// ============== Scene Initialization ================
// Do the rest of the setup and start the scene.
setupHero(scene, delegate)
setupFan(scene, delegate)
setupCannons(scene, delegate)
}
func handleContact(bodyA : SKSpriteNode, bodyB : SKSpriteNode) {
if (bodyA == hero) {
bodyB.normalTexture = nil
bodyB.runAction(removeBalloonAction)
(Function)
(586 times)
(Function)
(21 times)
[SKTexture, SKTextu...
(4 times)
SKNoiseFieldNode
SKNoiseFieldNode
SKNoiseFieldNode
(GameScene ((Funct...
210.0
let y = yOffsetForTime(t: elapsedTime*20)
Balloons

Conspicuously Missing Features

1. Runtime Introspection
2. Method Swizzling
3. Key-Value Observing (KVO)
4. Key-Value Coding (KVC)
5. Runtime Manipulation of Methods & Classes
6. Invocations
7. Message proxying

Translated ObjC

vs

Swift ML


Boston Dynamics

1. Runtime Introspection

Your six options for runtime introspection

1. Avoid it. (Stick with compile-time types & constraints)
2. Apply dynamic casting
3. Leverage Swift's **`MirrorType`**
4. Abuse Objective-C's runtime
5. Use private functions
6. Resort to inspecting memory layout

The six degrees of **evil**

1. Avoid it. (Stick with compile-time types & constraints) – 👍
2. Apply dynamic casting – 👌
3. Leverage Swift's **`MirrorType`** – 😕
4. Abuse Objective-C's runtime – 😬
5. Use private functions – 😨
6. Resort to inspecting memory layout – 😱

Compile-time
Types & Constraints

QueryKit

```
struct MyStruct {
  let intProp: Int

  struct Attributes {
    static let intProp = Attribute<Int>("intProp")
  }
}

// Usage
let intProp = MyStruct.Attributes.intProp
intProp > 0 // NSPredicate("intProp > 0"), typesafe
```

Argo

```
extension User: JSONDecodable {
  static func create(name: String)(email: String?)(role: Role)
                    (friends: [User]) -> User {
    return User(name: name, email: email, role: role, friends: friends)
  }

  static func decode(j: JSONValue) -> User? {
    return User.create
      <^> j <| "name"
      <*> j <|? "email" // Use ? for parsing optional values
      <*> j <| "role" // Custom types conforming to JSONDecodable work
      <*> j <|| "friends" // parse arrays of objects
  }
}
```

Dynamic Casting

as?

```
protocol XPCConvertible {}

extension Int64: XPCConvertible {}
extension String: XPCConvertible {}

func toXPC(object: XPCConvertible) -> xpc_object_t? {
  switch(object) {
    case let object as Int64:
      return xpc_int64_create(object)
    case let object as String:
      return xpc_string_create(object)
    default:
      fatalError("Unsupported type for object: \(object)")
      return nil
  }
}
```

Official Swift Reflection

```
/// The type returned by `reflect(x)`; supplies an API for runtime reflection on `x`
protocol MirrorType {
    /// The instance being reflected
    var value: Any { get }
    /// Identical to `value.dynamicType`
    var valueType: Any.Type { get }
    /// A unique identifier for `value` if it is a class instance; `nil` otherwise.
    var objectIdentifier: ObjectIdentifier? { get }
    /// The count of `value`\ 's logical children
    var count: Int { get }
    subscript (i: Int) -> (String, MirrorType) { get }
    /// A string description of `value`.
    var summary: String { get }
    /// A rich representation of `value` for an IDE, or `nil` if none is supplied.
    var quickLookObject: QuickLookObject? { get }
    /// How `value` should be presented in an IDE.
    var disposition: MirrorDisposition { get }
}
```

```swift
struct MyStruct {
    let stringProp: String
    let intProp: Int
}

let reflection = reflect(MyStruct(stringProp: "a", intProp: 1))

for i in 0..<reflection.count {
    let propertyName = reflection[i].0
    let value = reflection[i].1.value
    println("\(propertyName) = \(value)")
    if let value = value as? Int {
        println("int")
    } else if let value = value as? String {
        println("string")
    }
}

// stringProp = a
// string
// intProp = 1
// int
```

Objective-C Runtime

```
import Foundation

class objcSub: NSObject {
    let string: String?
    let int: Int?
}

var propCount: UInt32 = 0
let properties = clazz_copyPropertyList(objcSub.self, &propCount)

for i in 0..<Int(propCount) {
    let prop = properties[i]
    String.fromCString(property_getName(prop))! // => "string"
    String.fromCString(property_getAttributes(prop))! // => "T@,N,R,Vstring"
}
```

Using Private Functions

```
nm -a libswiftCore.dylib | grep "stdlib"

> ...
> __TFSs28_stdlib_getDemangledTypeNameU__FQ_SS
> ...
> _swift_stdlib_conformsToProtocol
> _swift_stdlib_demangleName
> _swift_stdlib_dynamicCastToExistential1
> _swift_stdlib_dynamicCastToExistential1Unconditional
> _swift_stdlib_getTypeName
> ...
```

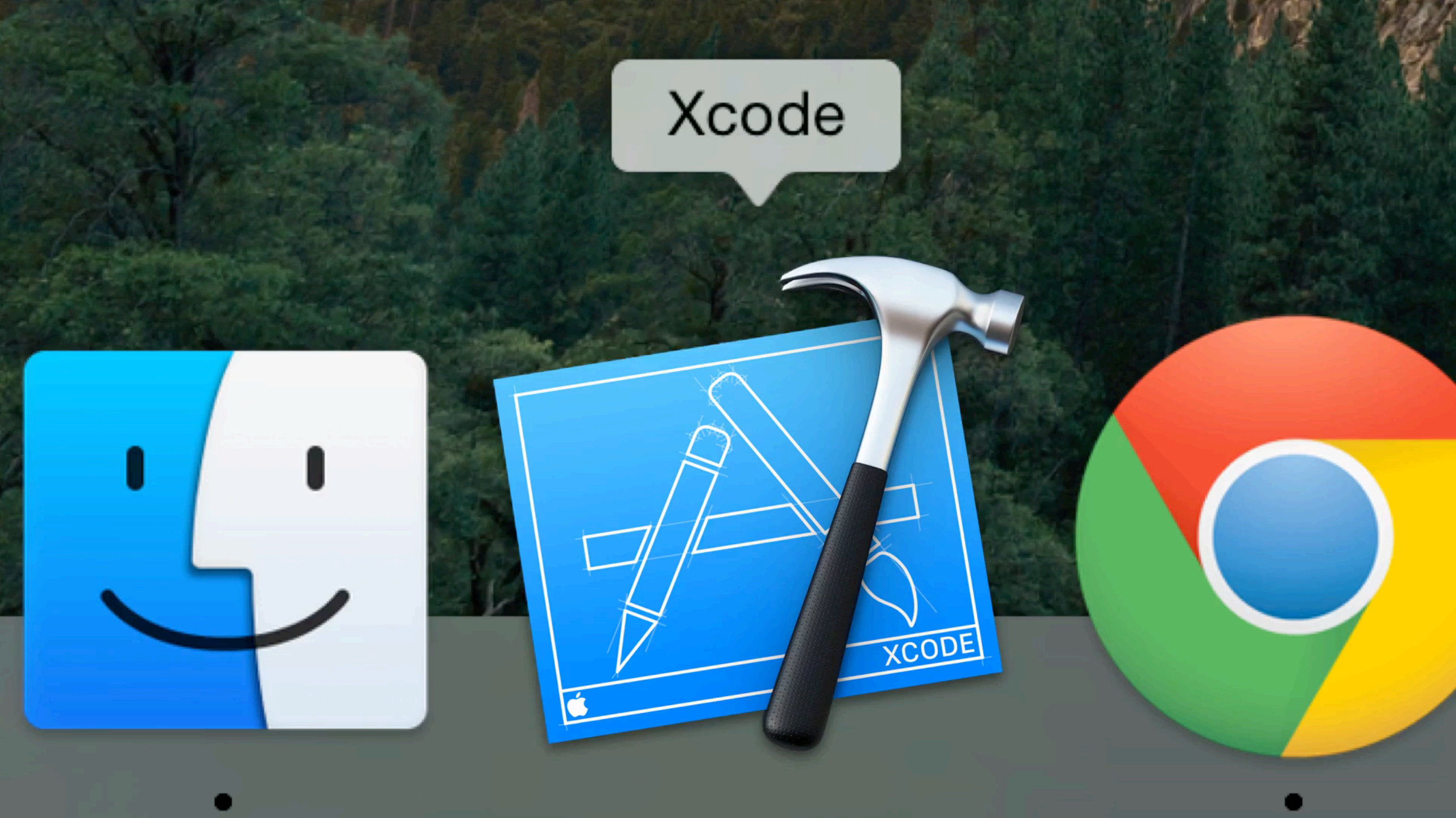
```swift
struct MyStruct {
    let stringProp: String
    let intProp: Int
}

let reflection = reflect(MyStruct(stringProp: "a", intProp: 1))

for i in 0..<reflection.count {
    let propertyName = reflection[i].0
    let value = reflection[i].1.value
    println("\(propertyName) = \(value)")
    println(_stdlib_getDemangledTypeName(value))
}

// stringProp = a
// Swift.String
// intProp = 1
// Swift.Int
```

Inspecting Memory Layout


Xcode
XCODE

```
struct _swift_data {
    unsigned long flags;
    const char *className;
    int fieldcount, flags2;
    const char *ivarNames;
    struct _swift_field **(*get_field_data)();
};

struct _swift_class {
    union {
        Class meta;
        unsigned long flags;
    };
    Class supr;
    void *buckets, *vtable, *pdata;
    int f1, f2; // added for Beta5
    int size, tos, mdsize, eight;
    struct _swift_data *swiftData;
    IMP dispatch[1];
};
```

```
class GenericClass<T> {}

class SimpleClass: NSObject {}

class ParentClass {
    let boolProp: Bool? // Optionals
    let intProp: Int    // Without default value
    var floatProp = 0 as Float // With default value
    var doubleProp = 0.0
    var stringProp = ""
    var simpleProp = SimpleClass()
    var genericProp = GenericClass<String>()
}
```

```
{
  "boolProp": "b",
  "intProp": "i",
  "floatProp": "f",
  "doubleProp": "d",
  "stringProp": "S",
  "simpleProp": "ModuleName.SimpleClass",
  "genericProp": "[Mangled GenericClass]"
}
```

2. Method swizzling

```
var _greet: String -> String = { "Hello, \($0)" }
struct Conf {
    let name: String

    func greet() -> String {
        return _greet(name)
    }
}

let nsnorth = Conf(name: "NSNorth")
nsnorth.greet() // => Hello NSNorth
_greet = { _ in "nope" } // Avoiding Georgia's advice
nsnorth.greet() // => nope
```

3. Key-Value Observing (KVO)

```
struct NSNorth {
    var speakers: [String] {
        didSet {
            println("NSNorth changed their speakers")
        }
    }

    init(speakers: [String]) {
        self.speakers = speakers
    }
}

var nsnorth = NSNorth(speakers: existingSpeakers)
nsnorth.speakers += ["Rob Rix"]
// => prints "NSNorth changed their speakers"
```

4. Key-Value Coding (KVC)

KVC in Objective-C

Dynamic Getter
`-[NSObject valueForKey:]`

Dynamic Setter
`-[NSObject setValue:forKey:]`

Lenses in Swift

```
struct Conference {
    let name: String
    let year: Int
}

struct Lens<A, B> {
    let from: A -> B
    let to: (B, A) -> A
}

let year = Lens(from: { $0.year }, to: {
    Conference(name: $1.name, year: $0)
})

let nsnorth2 = Conference(name: "NSNorth", year: 2014)
let nsnorth3 = year.to(2015, nsnorth2) // => Conference(name: "NSNorth", year: 2015)
```


Giving Up

dynamic

@objc

@NSManaged

These "cheat codes" re-enable Objective-C behaviour

- Dynamic Introspection
- Dynamic Method Dispatch
- KVO
- KVC
- Message Proxying
- Swizzling

Lots more!

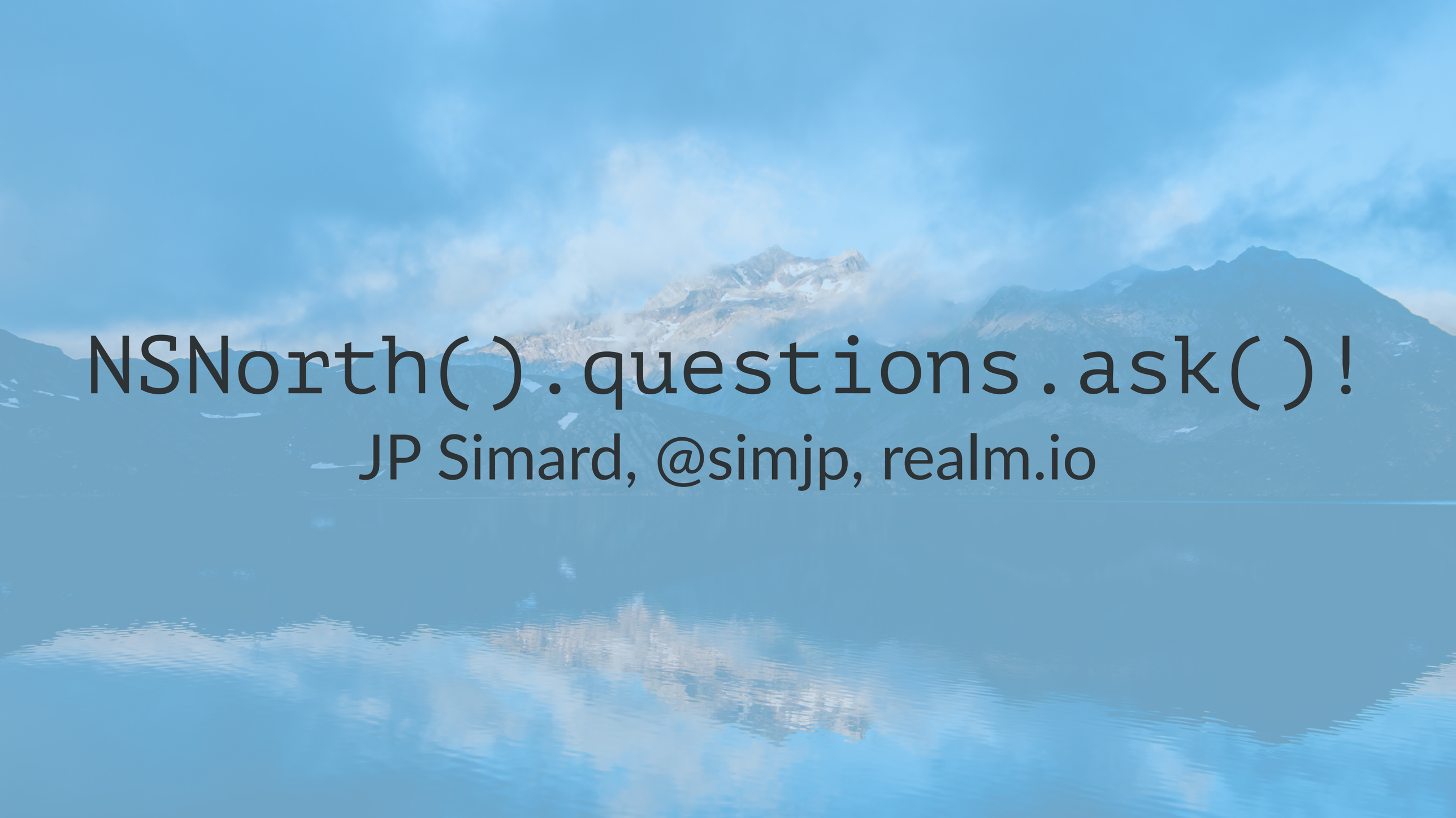
(that you "can't" do)

- C++ interop
- preprocessor
- documentation
- test coverage
- clang format
- other tooling

What do we *do* about this?

Swift is a moving target

Embrace Constraints



NSNorth().questions.ask()!

JP Simard, @simjp, realm.io