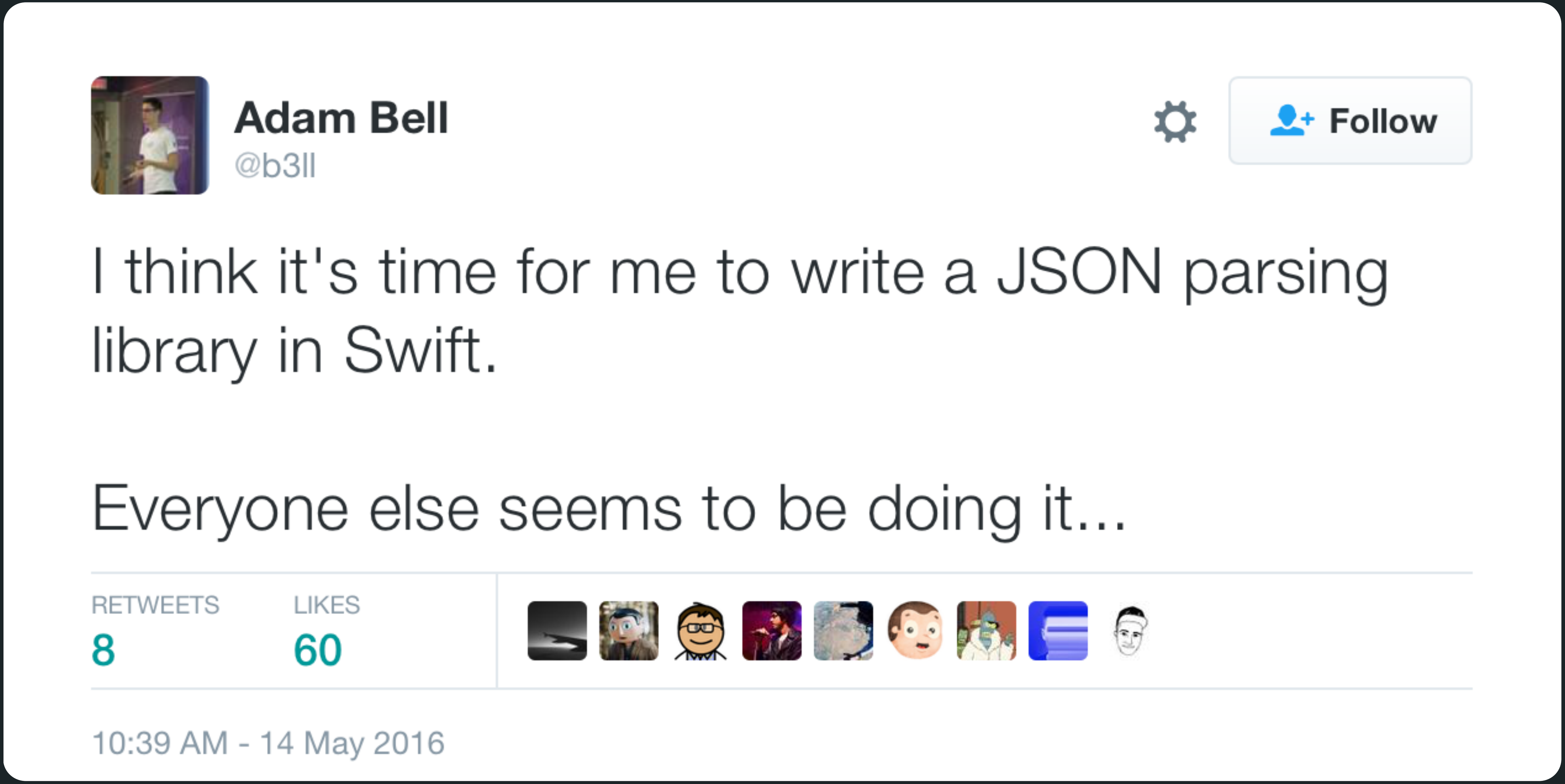
ELIEZER TALON

WHAT NOBODY WILL TELL YOU ABOUT JSON



Adam Bell
@b3ll

Follow

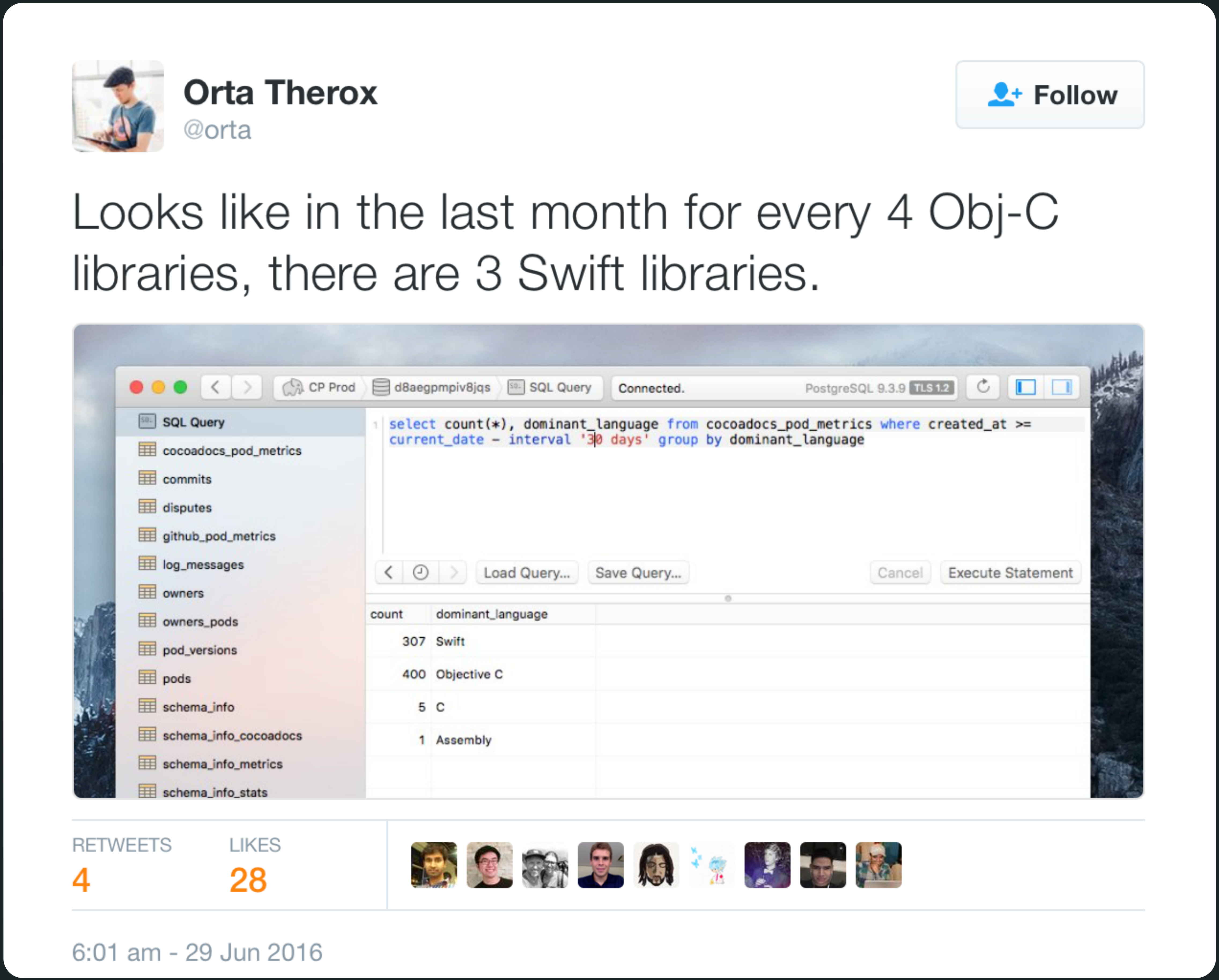
I think it's time for me to write a JSON parsing library in Swift.

Everyone else seems to be doing it...

RETWEETS 8 LIKES 60

10:39 AM - 14 May 2016

OBJECTIVE-C	SWIFT
1586	919



Orta Therox
@orta

Follow

Looks like in the last month for every 4 Obj-C libraries, there are 3 Swift libraries.

```
select count(*), dominant_language from cocoadocs_pod_metrics where created_at >=
current_date - interval '30 days' group by dominant_language
```

count	dominant_language
307	Swift
400	Objective C
5	C
1	Assembly

RETWEETS 4
LIKES 28

6:01 am - 29 Jun 2016

JSON

SUSTAINABLE DEVELOPMENT

&

THIRD-PARTY LIBRARIES

A SHORT SUMMARY

JSON DEMYSTIFIED

2001

DOUGLAS CROCKFORD

Based on ECMA-262

Replacement for XML

Language-independent

Simple specification

JSON.ORG

JSON.ORG

No version number

“If JSON doesn’t fit a task, we will not extend JSON. We will replace JSON.”

–DOUGLAS CROCKFORD

JSON STANDARD

- 2002 json.org, defines the syntax

JSON STANDARD

- 2002 json.org, defines the syntax
- 2006 RFC 4627, adds operational aspects

JSON STANDARD

- 2002 json.org, defines the syntax
- 2006 RFC 4627, adds operational aspects
- 2013 ECMA-404, official standard

JSON STANDARD

- 2002 json.org, defines the syntax
- 2006 RFC 4627, adds operational aspects
- 2013 ECMA-404, official standard
- 2014 RFC 7159, revisits operational aspects

LOOKING FOR

ALTERNATIVES?

XML

YAML

MESSAGE PACK

FLAT BUFFERS

XML

YAML

MessagePack

Flat Buffers

SUPPORT IN APPLE PLATFORMS

NSJSONSERIALIZATION

```
class func JSONObjectWithData(_:options:)

class func dataWithJSONObject(_:options:)
```

“The top level object is either an `NSArray` or an `NSDictionary`”

“Other rules may apply. Calling ***`isValidJSONObject:`*** *or attempting a conversion are the definitive ways to tell if a given object can be converted to JSON data.”*

"Other rules may apply. Calling **`isValidJSONObject:`** *or attempting a conversion are the definitive ways to tell if a given object can be converted to JSON data."*

PARSING JSON

SO WHAT?

#1

Limited support in Swift

2

NSJSONSerialization is not fully compliant

3

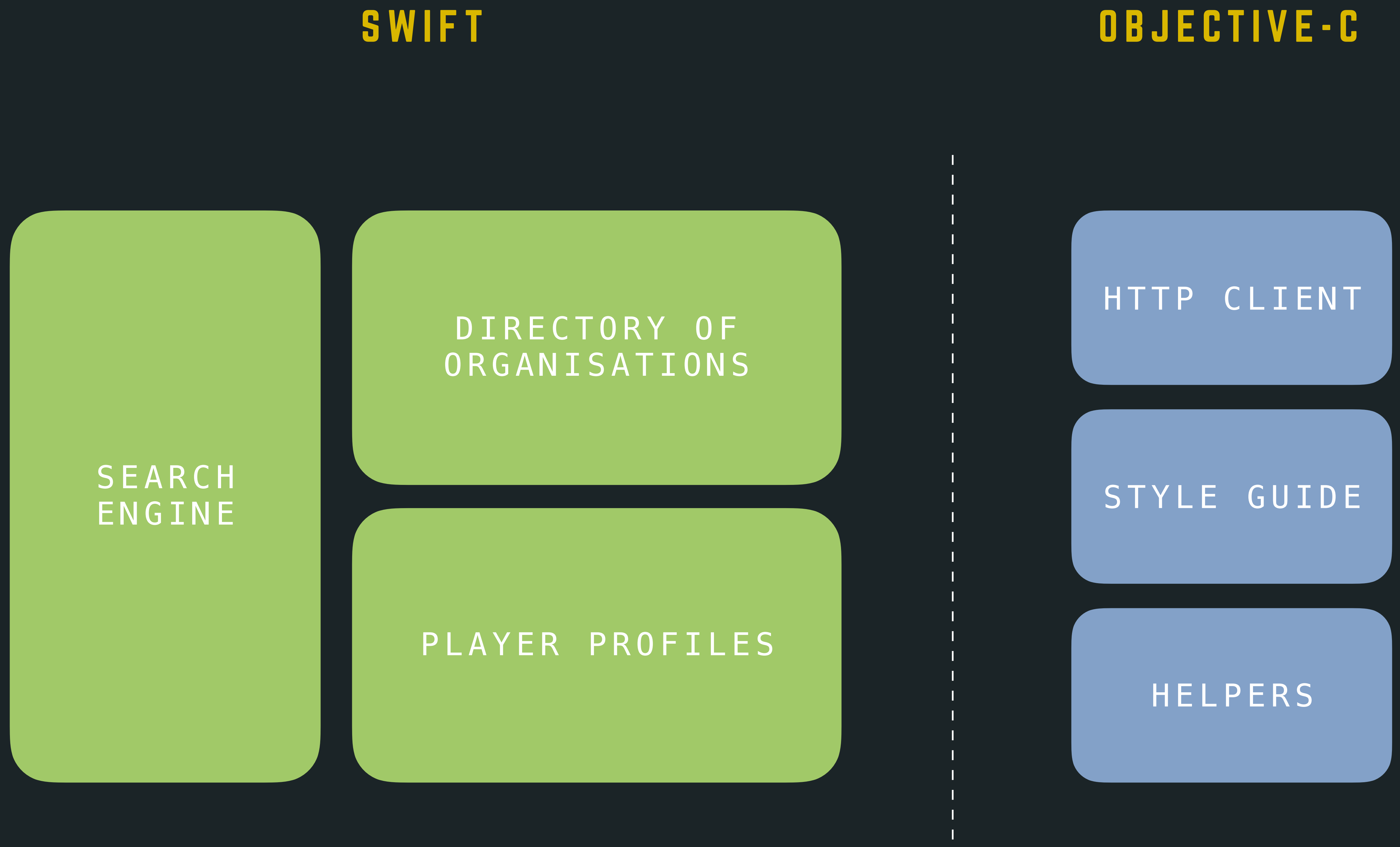
Learning experience

4

Bragging!

PARSING JSON

A PERSONAL JOURNEY


SWIFT
OBJECTIVE-C
SEARCH ENGINE
DIRECTORY OF ORGANISATIONS
PLAYER PROFILES
HTTP CLIENT
STYLE GUIDE
HELPERS

FREDDY – BIG NERD RANCH

```swift
import Freddy

struct Person {

    let name: String
    let age: Int

}

extension Person: JSONDecodable {

    init(json value: JSON) throws {
        name = try value.string("name")
        age = try value.int("age")
    }

}

func parse() {
    let data: NSData = getSomeData()
    do {
        let json = try JSON(data: data)
        let _ = try json.bool("success")
    } catch {
        // do something with `error`
    }
}
```

FREDDY – BIG NERD RANCH

```
import Freddy

struct Person {

    let name: String
    let age: Int

}

extension Person: JSONDecodable {

    init(json value: JSON) throws {
        name = try value.string("name")
        age = try value.int("age")
    }

}

func parse() {
    let data: NSData = getSomeData()
    do {
        let json = try JSON(data: data)
        let _ = try json.bool("success")
    } catch {
        // do something with `error`
    }
}
```

~~NSJSONSerialization~~

FREDDY – BIG NERD RANCH

```swift
import Freddy

struct Person {

    let name: String
    let age: Int

}

extension Person: JSONDecodable {

    init(json value: JSON) throws {
        name = try value.string("name")
        age = try value.int("age")
    }

}

func parse() {
    let data: NSData = getSomeData()
    do {
        let json = try JSON(data: data)
        let _ = try json.bool("success")
    } catch {
        // do something with `error`
    }
}
```

~~NSJSONSerialization~~

FREDDY – BIG NERD RANCH

```
import Freddy

struct Person {

    let name: String
    let age: Int

}

extension Person: JSONDecodable {

    init(json value: JSON) throws {
        name = try value.string("name")
        age = try value.int("age")
    }

}

func parse() {
    let data: NSData = getSomeData()
    do {
        let json = try JSON(data: data)
        let _ = try json.bool("success")
    } catch {
        // do something with `error`
    }
}
```

ARGO – THOUGHTBOT

```
import Argo
import Curry

struct Person {

    let name: String
    let age: Int

}

extension Person: Decodable {

    static func decode(j: JSON) -> Decoded<Person> {
        return curry(Person.init)
            <^> j <| "name"
            <*> j <| "age"
    }

}

func parse() {
    let json = try? NSJSONSerialization.JSONObjectWithData(data, options: [])

    if let j: AnyObject = json {
        let person: Person? = decode(j)
    }
}
```

ARGO – THOUGHTBOT

```
import Argo
import Curry

struct Person {

    let name: String
    let age: Int

}

extension Person: Decodable {

    static func decode(j: JSON) -> Decoded<Person> {
        return curry(Person.init)
            <^> j <| "name"
            <*> j <| "age"
    }

}

func parse() {
    let json = try? NSJSONSerialization.JSONObjectWithData(data, options: [])

    if let j: AnyObject = json {
        let person: Person? = decode(j)
    }
}
```

Generics

ARGO – THOUGHTBOT

```
import Argo
import Curry

struct Person {

    let name: String
    let age: Int

}

extension Person: Decodable {

    static func decode(j: JSON) -> Decoded<Person> {
        return curry(Person.init)
            <^> j <| "name"
            <*> j <| "age"
    }

}

func parse() {
    let json = try? NSJSONSerialization.JSONObjectWithData(data, options: [])

    if let j: AnyObject = json {
        let person: Person? = decode(j)
    }
}
```

3 custom operators

ARGO – THOUGHTBOT

```
import Argo
import Curry

struct Person {

    let name: String
    let age: Int

}

extension Person: Decodable {

    static func decode(j: JSON) -> Decoded<Person> {
        return curry(Person.init)
            <^> j <| "name"
            <*> j <| "age"
    }

}

func parse() {
    let json = try? NSJSONSerialization.JSONObjectWithData(data, options: [])

    if let j: AnyObject = json {
        let person: Person? = decode(j)
    }
}
```

Currying

ARGO – THOUGHTBOT

```
import Argo
import Curry

struct Person {

    let name: String
    let age: Int

}

extension Person: Decodable {

    static func decode(j: JSON) -> Decoded<Person> {
        return curry(Person.init)
            <^> j <| "name"
            <*> j <| "age"
    }

}

func parse() {
    let json = try? NSJSONSerialization.JSONObjectWithData(data, options: [])

    if let j: AnyObject = json {
        let person: Person? = decode(j)
    }
}
```

3 additional types

“We feel that these patterns greatly reduce the pain felt in trying to use JSON with Swift “

“We feel that these patterns greatly reduce the pain felt in trying to use JSON with Swift ”

“Not every piece of clever code is a great pattern.”

–NICK O’NEILL

OBJECT MAPPER – HEARST

```swift
import ObjectMapper

struct Person {

    var name: String
    var age: Int

}

extension Person: Mappable {

    init?(_ map: Map) {
        guard let name = map.JSONDictionary["name"] as? String else {
            return nil
        }
        self.name = name
    }

    mutating func mapping(map: Map) {
        name <- map["name"]
        age  <- map["age"]
    }

}

let person = Mapper<Person>().map(json)
```

OBJECT MAPPER – HEARST

```swift
import ObjectMapper

struct Person {

    var name: String
    var age: Int

}

extension Person: Mappable {

    init?(_ map: Map) {
        guard let name = map.JSONDictionary["name"] as? String else {
            return nil
        }
        self.name = name
    }

    mutating func mapping(map: Map) {
        name <- map["name"]
        age  <- map["age"]
    }

}

let person = Mapper<Person>().map(json)
```

OBJECT MAPPER – HEARST

```
import ObjectMapper

struct Person {

    var name: String
    var age: Int

}

extension Person: Mappable {

    init?(_ map: Map) {
        guard let name = map.JSONDictionary["name"] as? String else {
            return nil
        }
        self.name = name
    }

    mutating func mapping(map: Map) {
        name <- map["name"]
        age  <- map["age"]
    }

}

let person = Mapper<Person>().map(json)
```

SWIFTY JSON

```
import SwiftyJSON

struct Person {

    let name: String
    let age: Int

}

extension Person {

    init?(response: [String: AnyObject]) {
        let json = JSON(response)

        guard let name = json["name"].string else {
            return nil
        }

        guard let age = json["age"].int else {
            return nil
        }

        self.name = name
        self.age = age
    }

}
```

SWIFTY JSON

```swift
import SwiftyJSON

struct Person {

    let name: String
    let age: Int

}

extension Person {

    init?(response: [String: AnyObject]) {
        let json = JSON(response)

        guard let name = json["name"].string else {
            return nil
        }

        guard let age = json["age"].int else {
            return nil
        }

        self.name = name
        self.age = age
    }

}
```

SWIFTY JSON

```
import SwiftyJSON

struct Person {

    let name: String
    let age: Int

}

extension Person {

    init?(response: [String: AnyObject]) {
        let json = JSON(response)

        guard let name = json["name"].string else {
            return nil
        }

        guard let age = json["age"].int else {
            return nil
        }

        self.name = name
        self.age = age
    }

}
```

UNBOX – JOHN SUNDELL

```
import Unbox

struct Person {

    let name: String
    let age: Int

}

extension Person: Unboxable {

    init(unboxer: Unboxer) {
        name = unboxer.unbox("name")
        age = unboxer.unbox("age")
    }

}

let person: Person = try Unbox(dictionary)
```

UNBOX – JOHN SUNDELL

```swift
import Unbox

struct Person {

    let name: String
    let age: Int

}

extension Person: Unboxable {

    init(unboxer: Unboxer) {
        name = unboxer.unbox("name")
        age = unboxer.unbox("age")
    }

}

let person: Person = try Unbox(dictionary)
```

UNBOX – JOHN SUNDELL

```swift
import Unbox

struct Person {

    let name: String
    let age: Int

}

extension Person: Unboxable {

    init(unboxer: Unboxer) {
        name = unboxer.unbox("name")
        age = unboxer.unbox("age")
    }

}

let person: Person = try Unbox(dictionary)
```

UNBOX – JOHN SUNDELL

```swift
import Unbox

struct Person {

    let name: String
    let age: Int

}

extension Person: Unboxable {

    init(unboxer: Unboxer) {
        name = unboxer.unbox("name")
        age = unboxer.unbox("age")
    }

}

let person: Person = try Unbox(dictionary)
```

UNBOX – JOHN SUNDELL

```swift
import Unbox

struct Person {

    let name: String
    let age: Int

}

extension Person: Unboxable {

    init(unboxer: Unboxer) {
        name = unboxer.unbox("name")
        age = unboxer.unbox("age")
    }

}

let person: Person = try Unbox(dictionary)
```

PARSER REQUIREMENTS

PARSER REQUIREMENTS

- Avoiding explicit types by using type inference.

PARSER REQUIREMENTS

- Avoiding explicit types by using type inference.
- Handling the existence/absence of keys by wrapping the native `Dictionary` type.

PARSER REQUIREMENTS

- Avoiding explicit types by using type inference.
- Handling the existence/absence of keys by wrapping the native `Dictionary` type.
- Keeping error handling simple by using `do/catch`.

OUR OWN PARSER

```
struct Person {

    let name: String
    let age: Int

}

extension Person: Deserializable {

    init?(response: [String: AnyObject]) {
        let parser = JSONParser(response)

        do {
            name = try parser.fetch("name")
            age = try parser.fetch("age")
        } catch {
            return nil
        }
    }

}
```

OUR OWN PARSER

```
struct Person {

    let name: String
    let age: Int

}

extension Person: Deserializable {

    init?(response: [String: AnyObject]) {
        let parser = JSONParser(response)

        do {
            name = try parser.fetch("name")
            age = try parser.fetch("age")
        } catch {
            return nil
        }
    }

}
```

OUR OWN PARSER

```swift
struct Person {

    let name: String
    let age: Int

}

extension Person: Deserializable {

    init?(response: [String: AnyObject]) {
        let parser = JSONParser(response)

        do {
            name = try parser.fetch("name")
            age = try parser.fetch("age")
        } catch {
            return nil
        }
    }

}
```

OUR OWN PARSER

```
struct Person {

    let name: String
    let age: Int

}

extension Person: Deserializable {

    init?(response: [String: AnyObject]) {
        let parser = JSONParser(response)

        do {
            name = try parser.fetch("name")
            age = try parser.fetch("age")
        } catch {
            return nil
        }
    }

}
```

OUR OWN PARSER

```swift
struct Person {

    let name: String
    let age: Int

}

extension Person: Deserializable {

    init?(response: [String: AnyObject]) {
        let parser = JSONParser(response)

        do {
            name = try parser.fetch("name")
            age = try parser.fetch("age")
        } catch {
            return nil
        }
    }

}
```

OUR OWN PARSER

```
struct JSONParser {

    private let dictionary: [String: AnyObject]

    init(_ dictionary: [String: AnyObject]) {
        self.dictionary = dictionary
    }

    func fetch<T>(key: String) throws -> T {
        guard let object = dictionary[key] else  {
            throw ParseError.KeyNotFound(key)
        }

        guard let value = object as? T else {
            throw ParseError.TypeMismatch
        }

        return value
    }

}
```

OUR OWN PARSER

```
struct JSONParser {

    private let dictionary: [String: AnyObject]

    init(_ dictionary: [String: AnyObject]) {
        self.dictionary = dictionary
    }

    func fetch<T>(key: String) throws -> T {
        guard let object = dictionary[key] else  {
            throw ParseError.KeyNotFound(key)
        }

        guard let value = object as? T else {
            throw ParseError.TypeMismatch
        }

        return value
    }

}
```

Step 1

OUR OWN PARSER

```
struct JSONParser {

    private let dictionary: [String: AnyObject]

    init(_ dictionary: [String: AnyObject]) {
        self.dictionary = dictionary
    }

    func fetch<T>(key: String) throws -> T {
        guard let object = dictionary[key] else  {
            throw ParseError.KeyNotFound(key)
        }

        guard let value = object as? T else {
            throw ParseError.TypeMismatch
        }

        return value
    }

}
```

Step 2

OUR OWN PARSER

```
struct JSONParser {

    private let dictionary: [String: AnyObject]

    init(_ dictionary: [String: AnyObject]) {
        self.dictionary = dictionary
    }

    func fetch<T>(key: String) throws -> T {
        guard let object = dictionary[key] else  {
            throw ParseError.KeyNotFound(key)
        }

        guard let value = object as? T else {
            throw ParseError.TypeMismatch
        }

        return value
    }

}
```

Step 3

OUR OWN PARSER

```swift
struct JSONParser {

    private let dictionary: [String: AnyObject]

    init(_ dictionary: [String: AnyObject]) {
        self.dictionary = dictionary
    }

    func fetch<T>(key: String) throws -> T {
        guard let object = dictionary?[key] else  {
            throw ParseError.KeyNotFound(key)
        }

        guard let value = object as? T else {
            throw ParseError.TypeMismatch
        }

        return value
    }

}
```

Step 1

Step 2

Step 3

CHOOSING A THIRD-PARTY LIBRARY

WHAT CAN WE LEARN?

SUSTAINABLE DEVELOPMENT

REASONS TO AVOID A THIRD-PARTY LIBRARY

#1

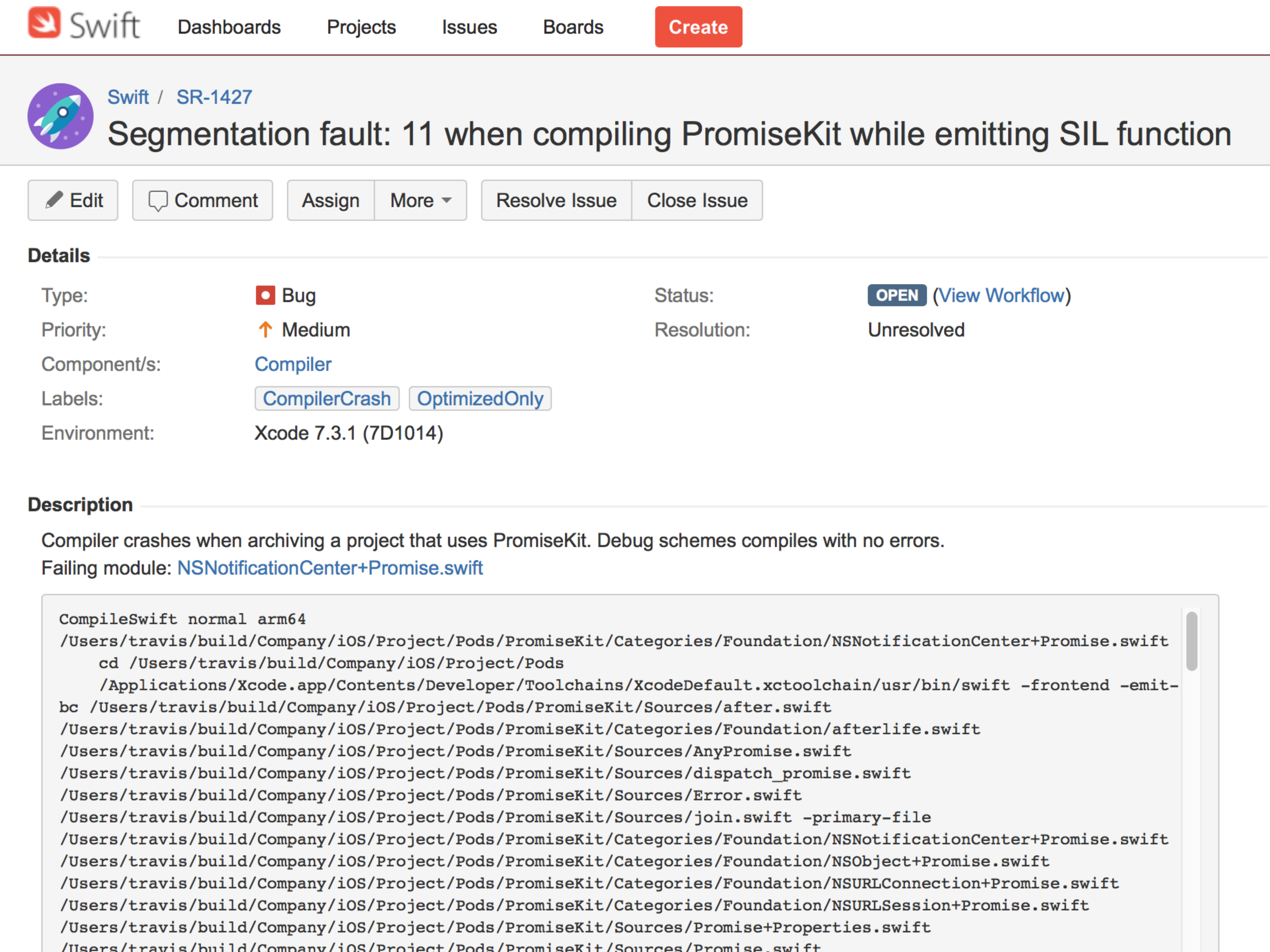
Make sure your problem is not somewhere else

2

Strive for simple solutions to simple problems

3

Stay as close as possible to an evolving platform



Swift Dashboards Projects Issues Boards Create

Swift / SR-1427

Segmentation fault: 11 when compiling PromiseKit while emitting SIL function

Edit | Comment | Assign | More | Resolve Issue | Close Issue

Details

Type:	🟥 Bug	Status:	OPEN (View Workflow)
Priority:	↑ Medium	Resolution:	Unresolved
Component/s:	Compiler		
Labels:	CompilerCrash OptimizedOnly		
Environment:	Xcode 7.3.1 (7D1014)		

Description

Compiler crashes when archiving a project that uses PromiseKit. Debug schemes compiles with no errors.
Failing module: NSNotificationCenter+Promise.swift

```
CompileSwift normal arm64
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/NSNotificationCenter+Promise.swift
    cd /Users/travis/build/Company/iOS/Project/Pods
    /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/swift -frontend -emit-
bc /Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/after.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/afterlife.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/AnyPromise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/dispatch_promise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/Error.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/join.swift -primary-file
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/NSNotificationCenter+Promise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/NSObject+Promise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/NSURLConnection+Promise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Categories/Foundation/NSURLSession+Promise.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/Promise+Properties.swift
/Users/travis/build/Company/iOS/Project/Pods/PromiseKit/Sources/Promise.swift
```

3

Stay as close as possible to an evolving platform

"When faced with two or more alternatives that deliver roughly the same value, take the path that makes future change easier."

–DAVE THOMAS

SUSTAINABLE DEVELOPMENT

REASONS TO USE A THIRD-PARTY LIBRARY

#1

Serious resource constraints or prototyping

2

Sometimes Apple doesn’t just work

3

Objective-C

SUSTAINABLE DEVELOPMENT

LET'S DO IT!