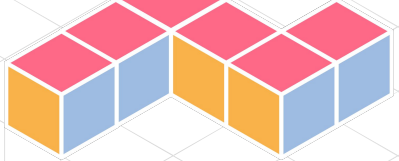
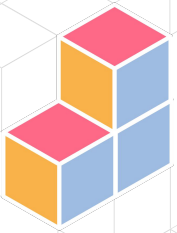
MONGODB WORLD'18
BUILD GIANT IDEAS





A Swift Introduction to Swift

Kaitlin Mahar

What makes Swift Swift?

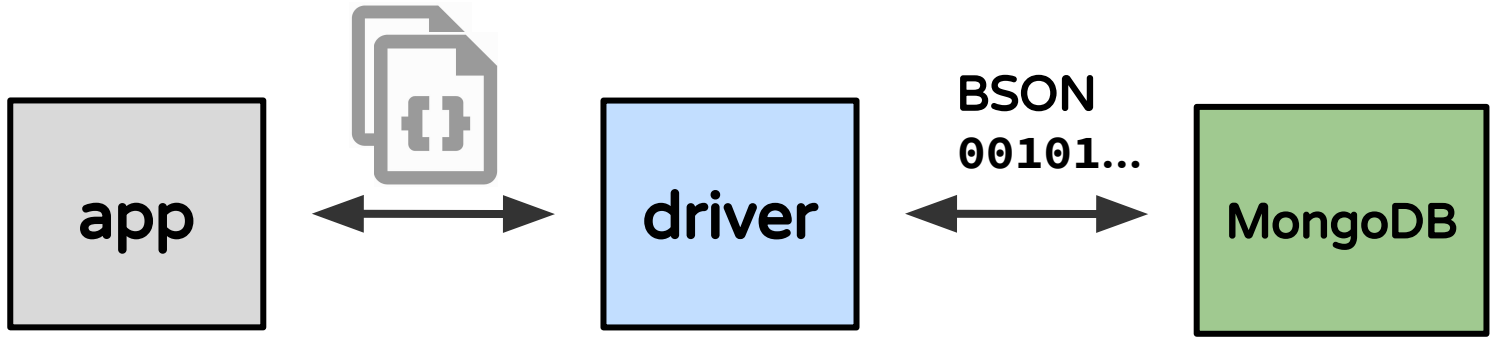
Kaitlin Mahar

Drivers Engineer @ MongoDB

@k__mahar

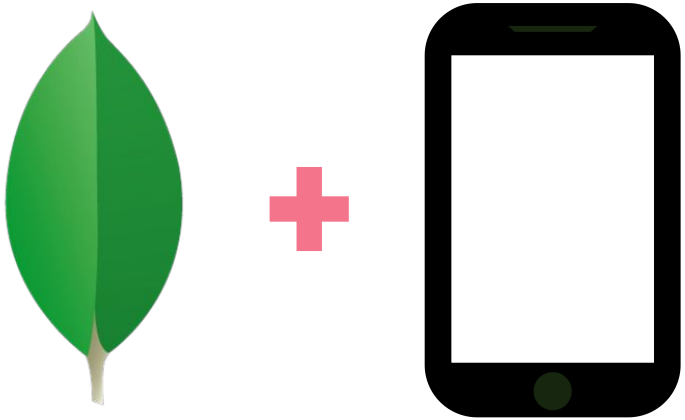
@kmahar

What is a MongoDB driver?



...  !!

Why build a Swift driver?



MongoDB Mobile!

What is Swift, anyway?

What is Swift?

- released in 2014, open-sourced in 2015
- up to version 4.1.2

Why did Apple create Swift?

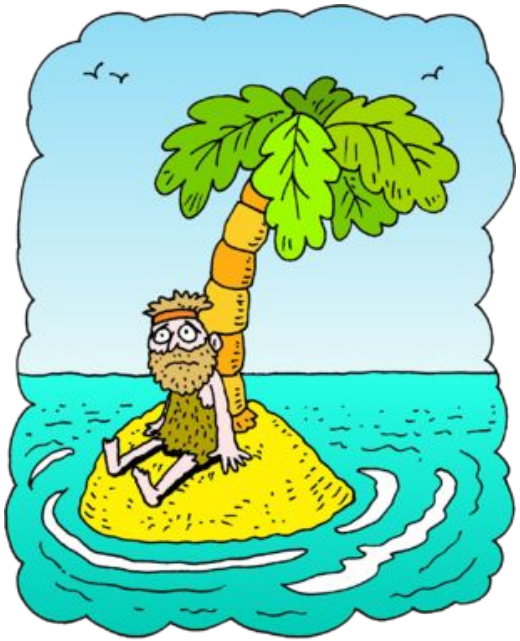
"Objective-C without the C"

```
NSString *str = @"hello,";
str = [str stringByAppendingString:@" world"];
```

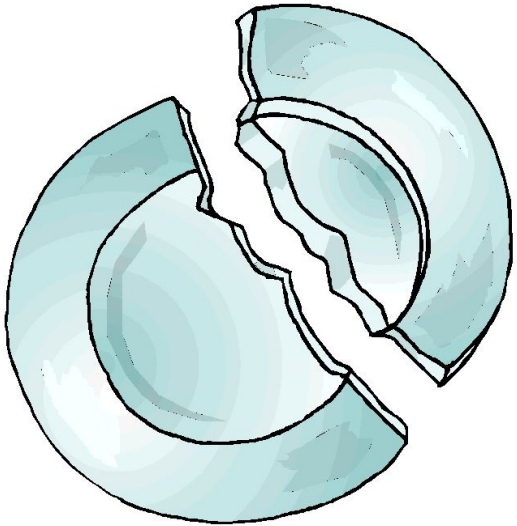
```
var str = "hello,"
str += " world"
```

So why do people still use Objective-C?

young language



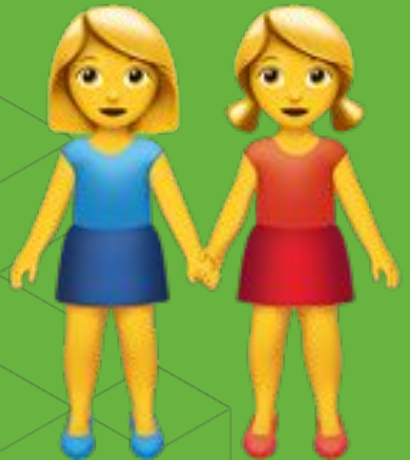
small community



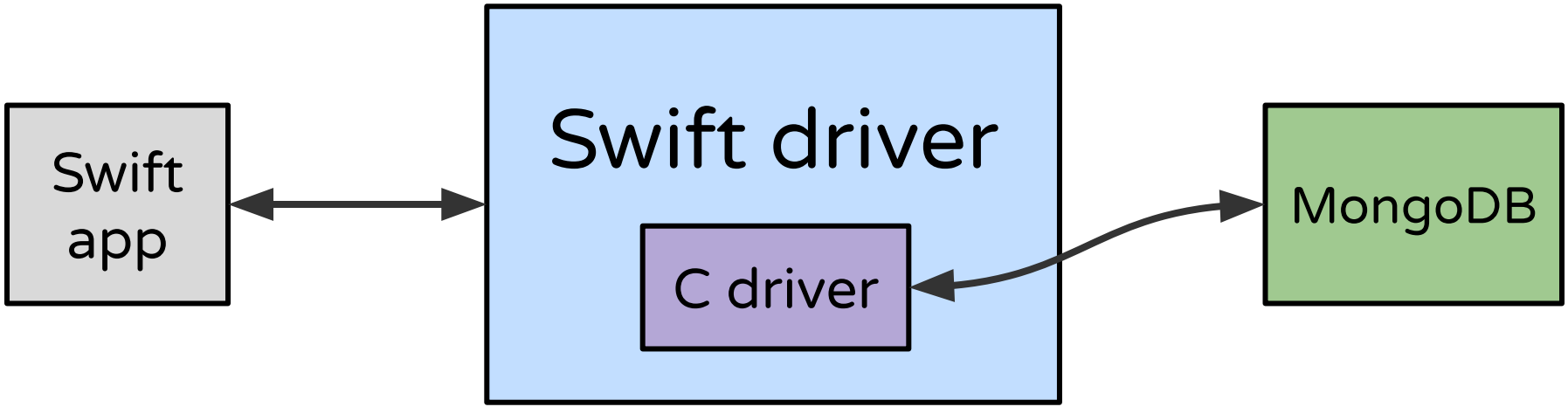
breaking changes

So why *should* I use Swift?

Swift makes working with C libraries easy.



The Swift driver wraps the C driver.



Descriptive Pointer Types

```
OpaquePointer
UnsafePointer<T>
UnsafeMutablePointer<T>
UnsafeBufferPointer<T>
UnsafeMutableBufferPointer<T>
UnsafeRawPointer
UnsafeMutableRawPointer
UnsafeRawBufferPointer
UnsafeMutableRawBufferPointer
```

Opaque (or not)

<T> or **Raw**

Mutable (or not)

Buffer (or not)

Using OpaquePointers in MongoSwift

```
public class MongoClient {
    internal var _client: OpaquePointer?
    // ...
}
```

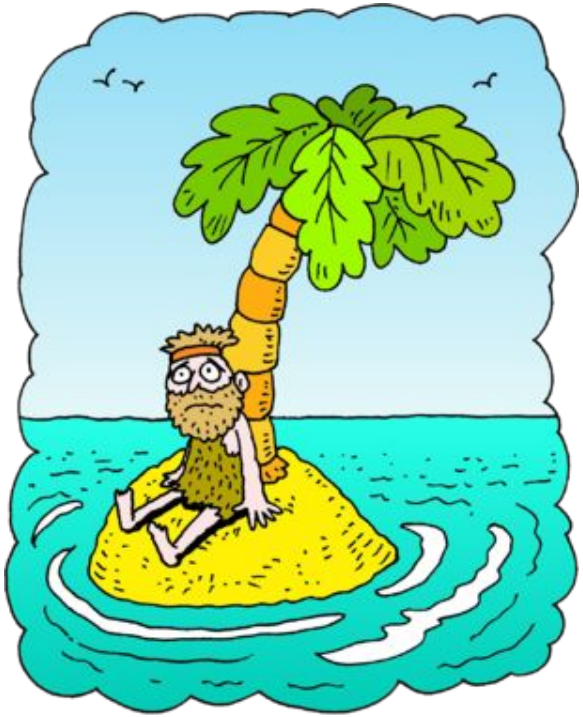
```
public class MongoDatabase {
    private var _database: OpaquePointer?
    // ...
}
```

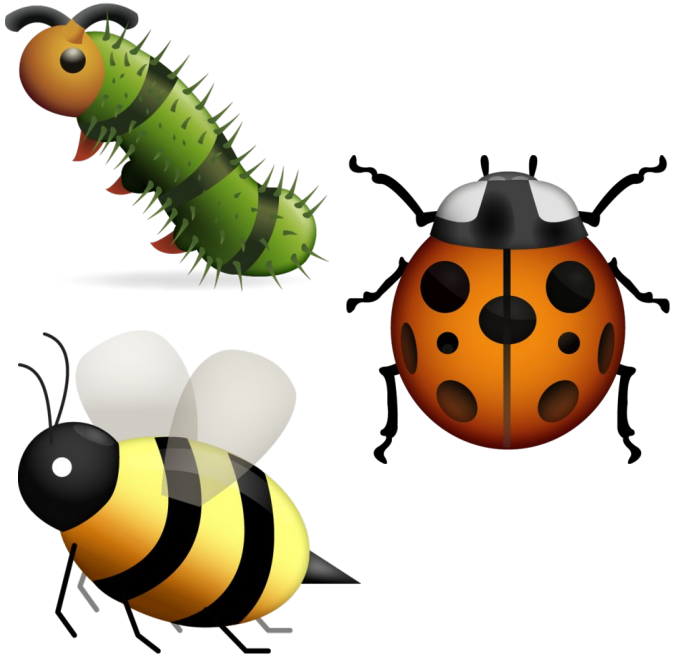
```
public class MongoCollection {
    private var _collection: OpaquePointer?
    // ...
}
```

Using UnsafePointers in MongoSwift

```
internal class DocumentStorage {
    internal var pointer: UnsafeMutablePointer<bson_t>!

    init() {
        self.pointer = bson_new()
    }

    deinit {
        bson_destroy(pointer)
        self.pointer = nil
    }
}
```

Challenges of wrapping C





Swift has a powerful type system.

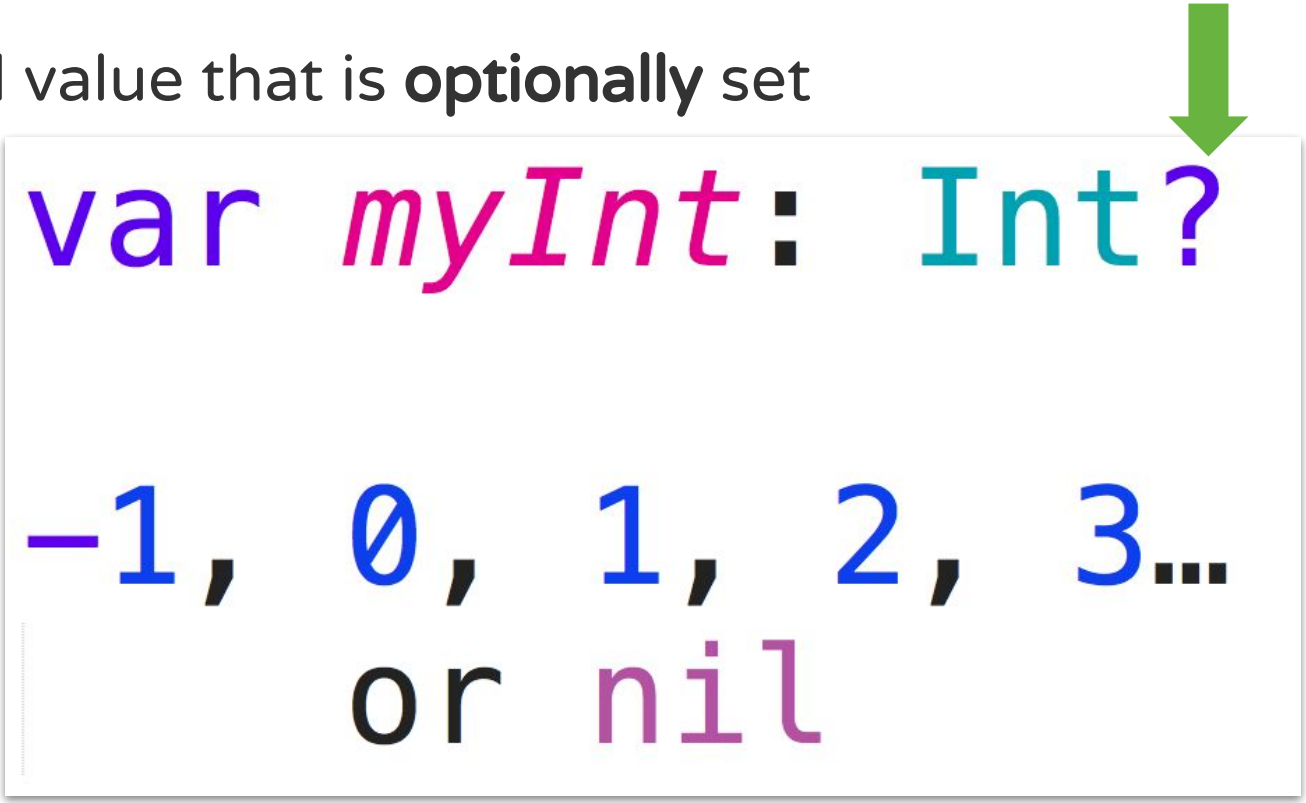
Built-in Optional Support

What is an optional?

```
var myInt: Int

-1, 0, 1, 2, 3…
```

What is an optional?

A typed value that is **optionally** set



Using Optionals in MongoSwift

```
public struct CollectionOptions {
    let readConcern: ReadConcern?
}
```

```
public class MongoDatabase {
    public func collection(_ name: String,
                           options: CollectionOptions? = nil)
                           throws -> MongoCollection {
        // ...
        if let rc = options?.readConcern {
            mongoc_collection_set_read_concern(...)
        }
    }
}
```

Using Optionals in MongoSwift

```
public struct CollectionOptions {
    let readConcern: ReadConcern?
}
```

```
public class MongoDatabase {
    public func collection(  name: String,
                             options: CollectionOptions? = nil)
                             throws -> MongoCollection {
        // ...
        if let rc = options?.readConcern {
            mongoc_collection_set_read_concern(...)
        }
    }
}
```

Using Optionals in MongoSwift

```
public struct CollectionOptions {
    let readConcern: ReadConcern?
}
```

```
public class MongoDatabase {
    public func collection(_ name: String,
                           options: CollectionOptions? = nil)
                           throws -> MongoCollection {
        // ...
        if let rc = options?.readConcern {
            mongoc_collection_set_read_concern(...)
        }
    }
}
```

Extensions

Extensions

Add new functionality to **existing** types

```
extension Date {

    public init(msSinceEpoch: Int64) {
        self.init(timeIntervalSince1970: Double(msSinceEpoch / 1000))
    }

    public var msSinceEpoch: Int64 {
        return Int64(self.timeIntervalSince1970 * 1000)
    }
}
```

Extensions

Add new functionality to **existing** types

```
extension Date {

    public init(msSinceEpoch: Int64) {
        self.init(timeIntervalSince1970: Double(msSinceEpoch / 1000))
    }

    public var msSinceEpoch: Int64 {
        return Int64(self.timeIntervalSince1970 * 1000)
    }
}
```

Extensions

Add new functionality to **existing** types

```
extension Date {

    public init(msSinceEpoch: Int64) {
        self.init(timeIntervalSince1970: Double(msSinceEpoch / 1000))
    }

    public var msSinceEpoch: Int64 {
        return Int64(self.timeIntervalSince1970 * 1000)
    }
}
```

Protocols

Protocols

- Similar to an **interface** in other languages - specify a required set of properties and methods
- A type that implement those requirements *conforms* to the protocol
- Classes, structs, and enums can all conform

ExpressibleByDictionaryLiteral

```
["a": 1, "b": 2]
```

```
public protocol ExpressibleByDictionaryLiteral {
  associatedtype Key
  associatedtype Value

  init(dictionaryLiteral elements: (Key, Value)...)
}
```

ExpressibleByDictionaryLiteral

```
public struct Document: ExpressibleByDictionaryLiteral {
    public init(dictionaryLiteral doc: (String, BsonValue?)...) {
        self.storage = DocumentStorage()
        for (key, value) in doc {
            self[key] = value
        }
    }
}
```

ExpressibleByDictionaryLiteral

```
let myDoc: Document = ["myInt": 2,
                       "myBool": false,
                       "myArray": [1, 2],
                       "myDate": Date()]
```

How to represent BSON values?

`Double`	`String`
`Document`	`Array`
`Binary`	`ObjectId`
`Boolean`	`Date`
`Regex`	`CodeWithScope`
`Int32`	`Timestamp`
`Int64`	`Decimal128`
`MinKey`	`MaxKey`

How to represent BSON values?

Double	String
Document	Array
Binary	ObjectId
Boolean	Date
Regex	CodeWithScope
Int32	Timestamp
Int64	Decimal128
MinKey	MaxKey

Custom protocols

```
public protocol BsonValue {

    var bsonType: BsonType { get }

    func encode(to data: UnsafeMutablePointer<bson_t>, forKey key: String) throws

    static func from(iter: inout bson_iter_t) -> BsonValue
}
```

Custom types can conform to protocols

```
public struct ObjectId: BsonValue {

    public let oid: String

    public var bsonType: BsonType { return .objectId }

    public func encode(to data: UnsafeMutablePointer<bson_t>, forKey key: String) throws {
        // ...
    }

    public static func from(iter: inout bson_iter_t) -> BsonValue {
        // ...
    }
}
```

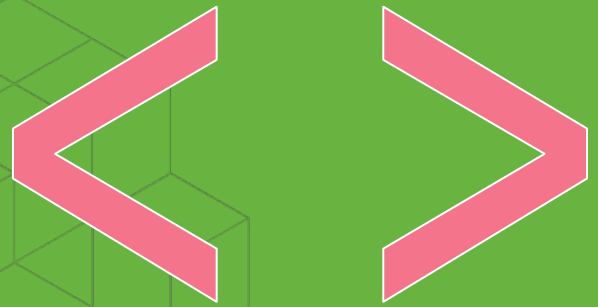
Native types can be *extended* to conform to protocols

```
extension Int32: BsonValue {

    public var bsonType: BsonType { return .int32 }

    public func encode(to data: UnsafeMutablePointer<bson_t>, forKey key: String) throws {
        // ...
    }

    public static func from(iter: inout bson_iter_t) -> BsonValue {
        // ...
    }
}
```

Protocols can be used as types

```
public struct ObjectId: BsonValue {

    public let oid: String

    public var bsonType: BsonType { return .objectId }

    public func encode(to data: UnsafeMutablePointer<bson_t>, forKey key: String) throws {
        // ...
    }

    public static func from(iter: inout bson_iter_t) -> BsonValue {
        // ...
    }
}
```

Generics



Generics

```
struct IntStack {
    var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
}
```

```
struct StringStack {
    var items = [String]()
    mutating func push(_ item: String) {
        items.append(item)
    }
    mutating func pop() -> String {
        return items.removeLast()
    }
}
```

Generics

```
struct Stack<Element> {
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}
```

Generics With Constraints

```
struct NumberStack<Element: Numeric> {
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
    func sum() -> Element {
        var total: Element = 0
        for elt in self.items { total += elt }
        return total
    }
}
```

Using Generics in MongoSwift

```
public class MongoCollection<T: Encodable & Decodable> {
    public func insertOne(_ value: T,
                          options: InsertOneOptions? = nil)
                          throws -> InsertOneResult? {
        let doc = try BsonEncoder().encode(value)
        // insert the doc into the collection
    }
}
```

Challenges of Swift's type system

- Lacking some desired features
  - Auto-generated ==, hashing (new in 4.1)
  - Conditional conformance (new in 4.1.. ish)
  - Default type parameters (someday?)

In summary…

- Swift is new and exciting!
- Swift has a powerful type system: protocols, extensions, generics, optionals, and more.
- Swift interoperates easily with C.
- You should try writing Swift!

What's next for the driver?

- 1.0 release
  - Full CRUD API
  - 4.0 features
  - replica sets, etc.
- Integration with Swift web frameworks
- Replace libbson usage with native Swift

Want to learn more?

Today:

3:30-4pm MongoDB Mobile session in Gibson

On your own:

The Swift Programming Language (by Apple, free)

objc.io videos and books

Swift by Sundell (blog)