ASYNCHRONOUS CODE WITH REACTIVECOCOA

PREVIOUSLY, ON SWIFT SUMMIT...

FLASHBACK

BEFORE

```swift
func loadAvatar(userID: String, completion: (UIImage?, NSError?) -> ()) {
    requestUserInfo(userID) { user, error in
        if let user = user {
            downloadImage(user.avatarURL) { avatar, error in
                if let avatar = avatar {
                    completion(avatar, nil)
                } else {
                    completion(nil, error)
                }
            }
        }
        else { completion(nil, error) }
    }
}
```

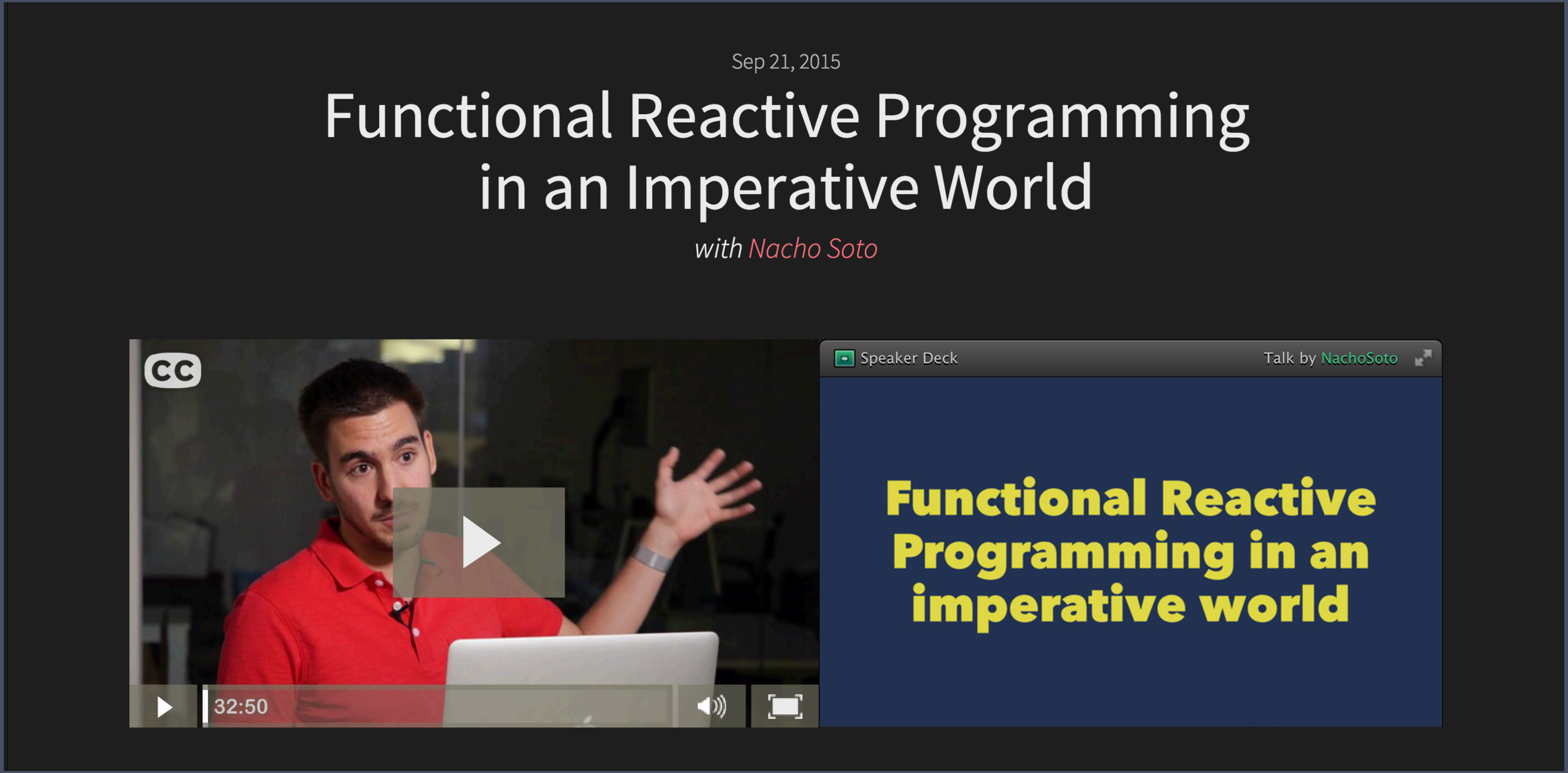
AFTER

```
func requestUserInfo(userID: String) -> Future<User, UserInfoErrorDomain>

func downloadImage(URL: NSURL) -> Future<UIImage, UserInfoErrorDomain>


func loadAvatar(userID: String) -> Future<UIImage, UserInfoErrorDomain> {
    return requestUserInfo(userID)
    .map { $0.avatarURL }
    .andThen(downloadImage)
}
```

WHY?

ASYNCHRONY IN MOBILE APPLICATIONS

- KVO
- USER DATA
- NETWORKING
- GESTURE RECOGNIZERS
- ANIMATIONS
- SENSORS
- MUTABLE STATE
- ...



Sep 21, 2015

Functional Reactive Programming in an Imperative World

with Nacho Soto

TL;DR

ASYNCHRONOUS CODE IS HARD

ASYNCHRONOUS CODE IS HARD

- CANCELATION
- THROTTLING
- ERROR HANDLING
- RETRYING
- THREADING
- ...

"THERE HAS TO BE A BETTER WAY!"

THAT PERFECTION IS UNATTAINABLE IS NO EXCUSE NOT TO STRIVE FOR IT.

– STOLEN FROM NACHO'S TWITTER BIO

REACTIVECOCOA™

REACTIVECOCOA IS HARD

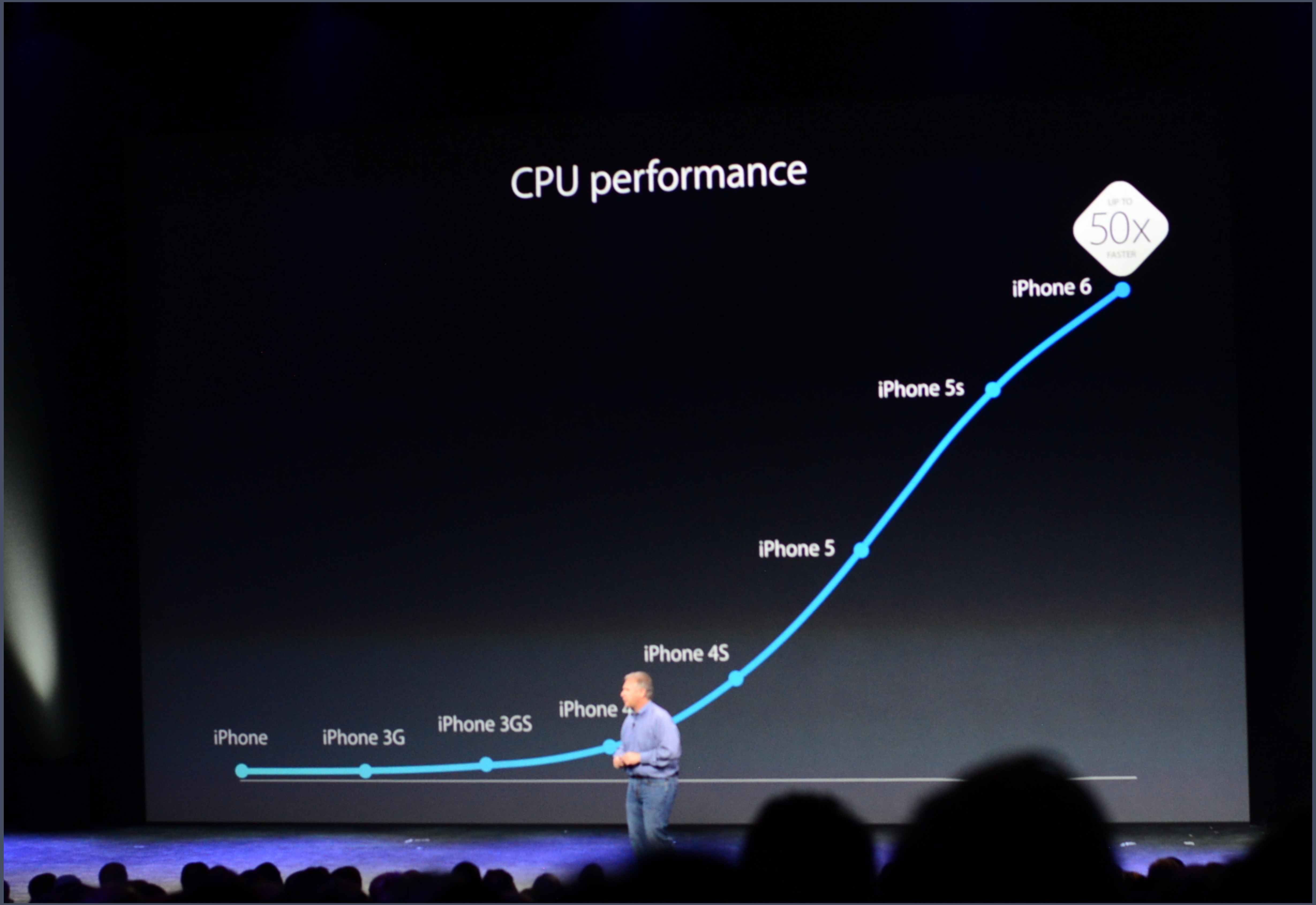
REACTIVECOCOA IS HARD

- SYNTAX IS UNFAMILIAR
- FOREIGN CONCEPTS
- FEELS DIFFERENT TO TRADITIONAL COCOA APIS
- APPLE'S APIS DON'T USE IT.

REACTIVECOCOA IS SIMPLE[1]

- FEW CONCEPTS
- ABSTRACT AWAY COMPLEXITY
- ONE PATTERN FOR ASYNCHRONOUS APIS

[1] "SIMPLE MADE EASY" - RICH HICKEY


CPU performance
UP TO 50x FASTER
iPhone 6
iPhone 5s
iPhone 5
iPhone 4S
iPhone 4
iPhone 3GS
iPhone 3G
iPhone

CONCEPTS IN COCOA INVOLVED IN ASYNCHRONOUS APIS

- DELEGATION
- NSOPERATION
- NSNOTIFICATIONCENTER
- KVO
- TARGET-ACTION
- RESPONDER CHAIN
- CALLBACK BLOCKS

SIGNALS

SIGNALS

- NEXT
- FAILED
- COMPLETED
- INTERRUPTED

`Signal`
AND
`SignalProducer`

Signal VS SignalProducer

```
func doSomethingAndGiveMeTheResult()    -> SignalProducer<Result, Error>


func observeSomeOnGoingWork()           -> Signal<NewValue, Error>
```

OPERATORS

RAC'S OPERATORS: DECLARATIVE VS IMPERATIVE

```
let array = ["one", "two", "three"]

// Imperative
var newArray: [String] = []
for string in array {
    newArray.append(string.uppercaseString)
}

// Declarative
let newArray = array.map { string in return string.uppercaseString }
```

RAC'S OPERATORS: DECLARATIVE VS IMPERATIVE

```
let throttleInterval: NSTimeInterval = 0.5

// Imperative
func search(query: String, completion: ([SearchResult]?, MyErrorType?) -> ())
var lastSearch: NSDate? // <--- State
func didTypeSearchQuery(searchQuery: String) {
    guard (lastSearch?.timeIntervalSinceNow > throttleInterval) ?? false else { return }

    lastSearchDate = NSDate()
    search(searchQuery) { results, error in ... }
}

// Declarative
let searchQuerySignal: Signal<String, NoError>
func search(query: String) -> SignalProducer<[SearchResult], MyErrorType>

searchQuerySignal.throttle(throttleInterval).flatMap(.Latest, search)
```

OPERATORS

- `map`
- `filter`
- `reduce`
- `collect`
- `combineLatest`
- `zip`
- `merge` / `concat` / `switchToLatest`
- `flatMapError` / `mapError`
- `retry`
- `throttle`

KVO

KVO

- CRASH IF OBJECT DEALLOCATES WHILE BEING OBSERVED.
- CRASH IF OBSERVE WRONG KEYPATH (STRINGLY-TYPED API)
- POSSIBLE CRASH WHEN DE-REGISTERING
- EASY TO BREAK PARENT CLASS (`context` OFTEN MISUSED)
- ALL OBSERVATIONS COME THROUGH ONE METHOD
- LOSE CONTRACT: "IS THIS KVO-COMPLIANT?"

PROPERTY

PROPERTY

```
// KVO
class MyClass {
    private(set) dynamic var value: Type
}

let object = MyClass()
object.addObserver(self, forKeyPath: "value", options: [], context: ctx)
func observeValueForKeyPath(keyPath: String?,
    ofObject object: AnyObject?,
     change: [NSObject : AnyObject]?,
     context: UnsafeMutablePointer<Void>) { /* HAVE FUN!! */ }

// PropertyType
class MyClass {
    var value: AnyProperty<Type>
}
let object = MyClass()
object.value.producer.startWithNext { value in ... }
```

MYTH:

"TO USE REACTIVECOCOA, I NEED TO RE-WRITE MY WHOLE APP"

CONCLUSIONS

- OUR TOOLS ARE IMPERFECT. STRIVE TO RECONSIDER PATTERNS, SEEK BETTER ALTERNATIVES.
- THERE'S VALUE IN THESE ABSTRACTIONS.
- REACTIVECOCOA CAN BE ADOPTED SLOWLY.

REFERENCES

- REACTIVECOCOA: HTTPS://GITHUB.COM/REACTIVECOCOA/REACTIVECOCOA
- BACK TO THE FUTURES - ME: HTTPS://REALM.IO/NEWS/SWIFT-SUMMIT-JAVIER-SOTO-FUTURES
- FUNCTIONAL REACTIVE PROGRAMMING IN AN IMPERATIVE WORLD - NACHO SOTO: HTTPS://REALM.IO/NEWS/NACHO-SOTO-FUNCTIONAL-REACTIVE-PROGRAMMING
- "SIMPLE MADE EASY" - RICH HICKEY: HTTP://WWW.INFOQ.COM/PRESENTATIONS/SIMPLE-MADE-EASY

THANK YOU!