

REACTIVECOCOA入門

SIGNALS, SIGNALPRODUCERS AND EVENTS! OH MY!

自己紹介

NICHOLAS MACCHAROLI (ニコラス・マカローリ)

VASILYでiOSエンジニアをしています

OSSとかSWIFT EVOLUTIONに貢献しています

最近SWIFT EVOLUTION SE-0053を提案してSWIFT3.0に
採用されました！

GITHUB.COM/**NIRMA**
@DINOSR

FRP(FUNCTIONAL REACTIVE PROGRAMMING) のメリットは？

- ＞ コードノイズが減らせます
- ＞ @IBACTION, NSNOTIFICATIONCENTERおよびCALLBACK/DELEGATEのパターンよりも、もっとシンプルな実装ができます。

例: リフレッシュボタンのクリックイベント (一般的な実装)

```
@IBAction func refreshButtonClicked(sender: AnyObject) {  
    updateViewForState(.Loading)  
    performNetworkRequest() {  
        updateModel()  
        dispatch_async(dispatch_get_main_queue()) {  
            updateViewForState(.Success)  
        }  
    }  
}
```

refreshButtonClicked: **が呼ばれたら**

1. ボタンの表示を更新
2. ローディングアイコンを表示
3. ネットワークリクエスト
4. リクエストが完了したらCALLBACKでUIとモデルを更新する

一つのメソッドの中に複数の責任の処理が
固まっていて良くない。
シンプルに書き換えたい。

ループ処理だとこんなイメージ

BEFORE:

```
var counter = 0
var animationImages = [UIImage]()

while counter < 10 {
    let imageString = "animation_image_\(counter)"
    if let image = UIImage(named: imageString) {
        animationImages += [image]
    }
}
```

ループ処理だとこんなイメージ

AFTER:

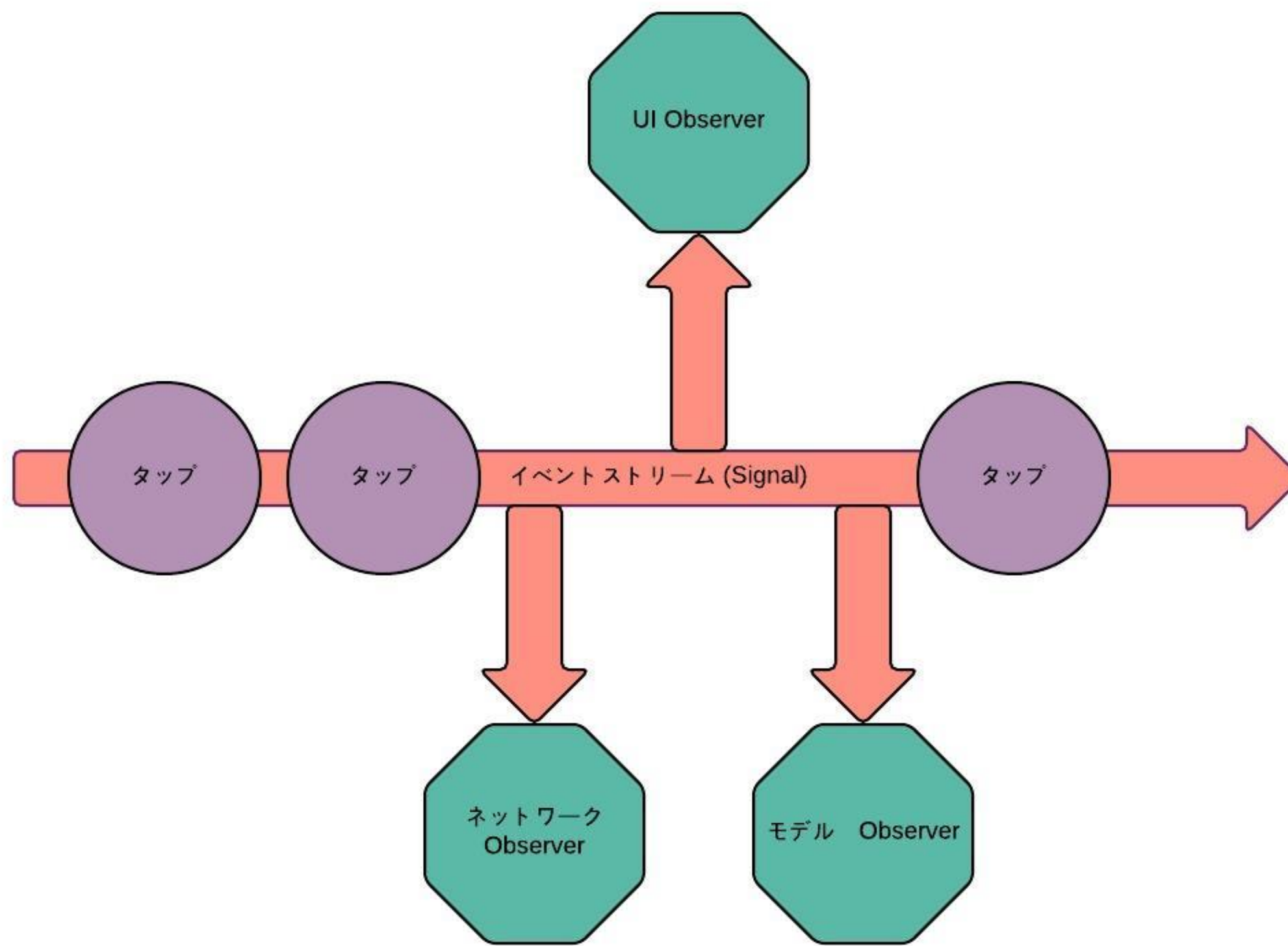
```
(0..<10).flatMap { UIImage(named: "animation_image_\($0)") }
```

RAC

FTW!

REACTIVECOCOA とRXの違い

- NAMING CONVENTION
- RX: HOT OBSERVABLE, COLD OBSERVABLE
- REACTIVECOCOA: SIGNAL, SIGNALPRODUCER
 - COCOA専用API拡張
 - よりシンプルな設計
 - ETC



YES, SIGNALS!

Signalでイベントを受け取ってObserverとか
DisposableでSIGNAL操作ができます。

EVENTS...

```
/// Signals must conform to the grammar:
/// `Next* (Failed | Completed | Interrupted)?`
public enum Event<Value, Error : ErrorType> {
    /// A value provided by the signal.
    case Next(Value)
    /// The signal terminated because of an error. No further events will be
    /// received.
    case Failed(Error)
    /// The signal successfully terminated. No further events will be received.
    case Completed
    /// Event production on the signal has been interrupted. No further events
    /// will be received.
    case Interrupted
    /// Whether this event indicates signal termination (i.e., that no further
    /// events will be received).
}
```

SIGNALS. CHEAP

常に発生しうるイベントストリーム

- ＞ UIコントロール
- ＞ APIリクエスト
- ＞ DB READ / WRITE
- ＞ UINOTIFICATIONCENTER
- ＞ なんでも

SIGNAL作成と使い方

```
let (userNameTextSignal, observer) = Signal<String, NoError>.pipe()
```

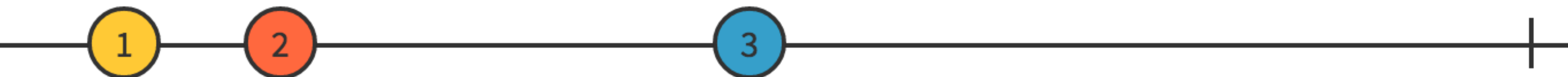
```
userNameTextSignal.observeNext { userName in  
    print("Next Event: \(userName)")  
}
```

```
observer.sendNext("l")           // Next Event: l  
observer.sendNext("la")          // Next Event: la  
observer.sendNext("lat")         // Next Event: lat  
observer.sendNext("latt")        // Next Event: latt  
observer.sendNext("lattn")       // Next Event: lattn  
observer.sendNext("lattne")      // Next Event: lattne  
observer.sendNext("lattner")     // Next Event: lattner  
observer.sendCompleted()
```

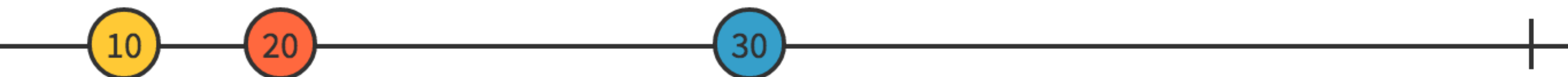
SIGNAL'S FRIENDS MAP, FILTER, REDUCE

RAC MARBLES

diagrams for ReactiveCocoa



`map { x in x * 10 }`



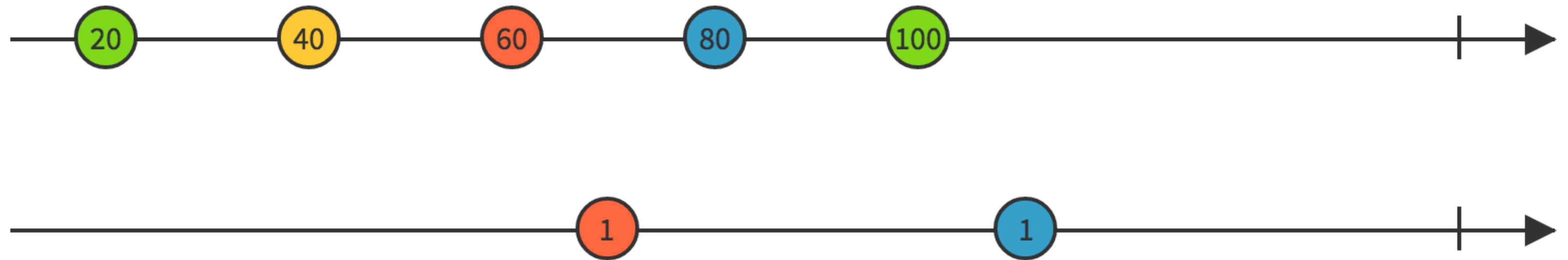
diagrams for ReactiveCocoa



`filter { x in x > 10 }`



Interactive diagrams for ReactiveCocoa



`flatten(.Merge)`



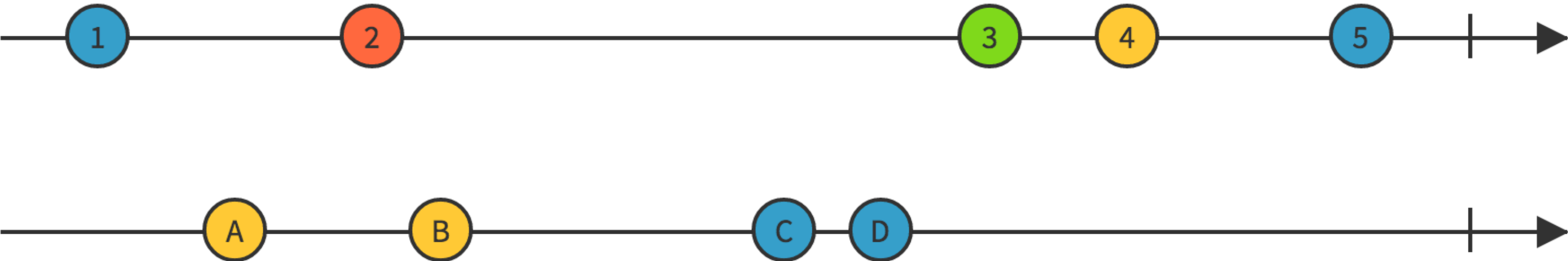
ve diagrams for ReactiveCocoa



reduce { x, y in x + y }



Interactive diagrams for ReactiveCocoa



zip



ve diagrams for ReactiveCocoa



take(2)





takeLast(1)



SIGNAL OPERATORS IN ACTION – FILTER

```
let (userNameTextSignal, observer) = Signal<String, NoError>.pipe()

let improvedUserNameTextSignal = userNameTextSignal.filter { $0.characters.count > 4 }

improvedUserNameTextSignal.observeNext { userName in
    print("Next Event: \(userName)")
}

observer.sendNext("lattn")
observer.sendNext("lattne")
observer.sendNext("lattner")
observer.sendCompleted()
```

既存のSIGNALから新しいSIGNALを作る

```
let (userNameTextSignal, observer) = Signal<String, NoError>.pipe()

let userNameValidSignal = userNameTextSignal.map {
    return $0.characters.count > 5
}

let backgroundColorSignal = userNameValidSignal.map {
    return $0 ? UIColor.greenColor() : UIColor.redColor()
}
```

SIGNALS & END OF LIFE

SWIFTのGeneratorTypeと同じように一回Signalを使い終わったら再利用できない。

SIGNALPRODUCER (COLD SIGNALS)

- SIGNALPRODUCERS は SIGNAL のファクトリー
- 購読しないかぎり、処理は開始されない
 - SIGNAL OPERATORS CAN BE 'LIFTED'

SIGNALPRODUCER – NOTHING

```
var userNameProducer = SignalProducer<String, NoError> { (observer, disposable) in
    ["foo", "bar", "zap", "bin", "fizz"].forEach { userName in
        print("Sending Username: \(userName)")
        observer.sendNext(userName)
    }
}
```

SIGNALPRODUCER – SOMETHING

```
var userNameProducer = SignalProducer<String, NoError> { (observer, disposable) in
    ["foo", "bar", "zap", "bin", "fizz"].forEach { userName in
        print("Sending Username: \(userName)")
        observer.sendNext(userName)
    }
}

userNameProducer.startWithNext { userName in
    print("Received Username: \(userName)")
}
```

OUTPUT

```
Sending Username: foo  
Received Username: foo  
Sending Username: bar  
Received Username: bar  
Sending Username: zap  
Received Username: zap  
Sending Username: bin  
Received Username: bin  
Sending Username: fizz  
Received Username: fizz
```

SIGNALPRODUCER – SIGNALPRODUCERからSIGNALPRODUCER作る

```
var userNameProducer = SignalProducer<String, NoError> { (observer, disposable) in
    ["foo", "bar", "zap", "bin", "fizz"].forEach { userName in
        print("Sending Username: \(userName)")
        observer.sendNext(userName)
    }
}

userNameProducer.startWithNext { userName in
    print("Received Username: \(userName)")
}

let zCounterSignal = userNameProducer.map { $0.containsString("z") }
zCounterSignal.startWithNext {
    if $0 {
        print("Contains z!")
    } else {
        print("No z here.")
    }
}
```

OUTPUT

No z here.

No z here.

Contains z!

No z here.

Contains z!

RAC4.0のAPI変更

- BASED ON SWIFT 2.0
- RAC SIGNALがなくなりました (THIS IS A LIE.)
- RAC3.0からRAC SIGNAL (HOT, COLD) が SIGNAL, SIGNALPRODUCERに変更
- ~~GLOBAL FUNCTIONS~~ Protocol Extensions
- |>(PIPE-LEFT)は普通の.(DOT OPERATOR)に変更

まとめ

- RAC4はいいFRPライブラリとして設計が良い
- コミュニティがヘルシーで進化も早いけど、まだレガシーコードが残っているの
で5/5を上げられないです
- COCOA拡張は便利だけど、ライブラリを肥大化させている