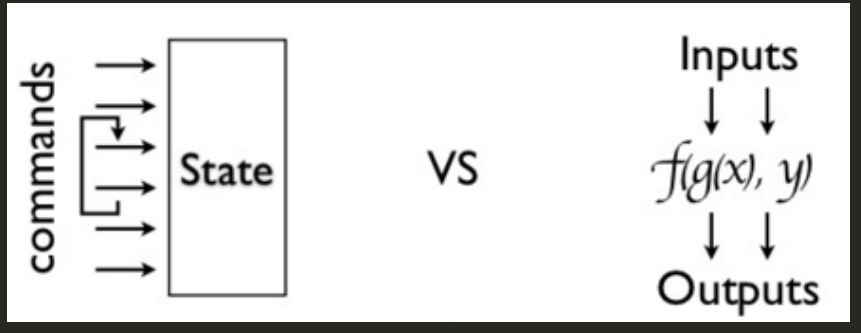
Functional Reactive Programming

What is Functional Programming?


commands
State
VS
Inputs
f(g(x), y)
Outputs

Functional programming is programming without assignment statements.

```
x = x + 1
0 = 1
```

```
x: = x + 1
```

wikipedia.de

What is Reactive?

```
a := b + c
```

```
() -> T
```

Consumer in charge

Eric Mejer youtube.com

`T -> ()`

Producer in charge

Every line of Code we write is executed in reaction to an event

Colin Eberhardt realm.io

But...

…these events come in many forms

Notifications, KVO, Delegates, Callbacks, Target/Selector,…

ReactiveCocoa provides a common interface for all events

ReactiveCocoa

RxSwift

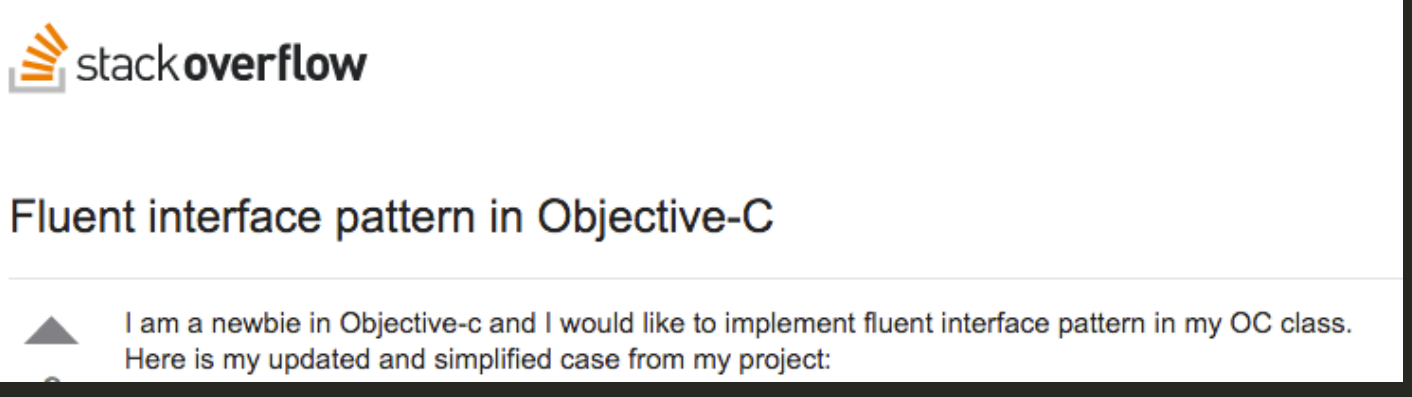
Bolts

PromiseKit

ReactiveKit

...

ReactiveCococa?


stackoverflow
Fluent interface pattern in Objective-C
I am a newbie in Objective-c and I would like to implement fluent interface pattern in my OC class. Here is my updated and simplified case from my project:

```
26  [[[[[[[[[[[[[[[[I dont] think] fluent] interface] patterns] will] be] useful]
    unless] you] want] to] write] methods] like] this];
```
– kennytm Jun 26 '10 at 15:53

swift closures ()->()

```swift
func increment(a: Int) -> Int {
    return a + 1
}

let result = increment(1)
```

```swift
let increment = { (a: Int) -> Int in
    return a + 1
}

let result = increment(1)
```

Type Inference

```
let states = ["California", "New York", "Texas", "Alaska"]

let uppercaseStates = states.map({ (state: String) -> String in
    return state.uppercaseString
})
```

```
let uppercaseStates = states.map({ state in
    return state.uppercaseString
})
```

```
let uppercaseStates = states.map({ state in
    state.uppercaseString
})
```

Shorthand Argument Names

```
let uppercaseStates = states.map({ $0.upppercaseString })
```

Trailing Closures

```
let uppercaseStates = states.map { $0.uppercaseString }
```

Signals

Transformations

Subscribe/Observe(Bind)

Signals send stuff

A signal is a sequence of ongoing events ordered in time

```
-----Pew-------Pew------Pew----------------
```

```
---tap-tap-------tap--->
```

```
---h--e-----l----l---o---->
```

```
--1--2--3--4--5--6--| // it terminates normally
```

```
--a--b--a--a--a---d---X // it terminates with error
```

```
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
}
```

Observe/Subscribe

```
signalProducer
    .startWithNext { next -> () in
        ...
    }
```

```
let usernameTextSignal = usernameField.rac_textSignal().toSignalProducer()

usernameTextSignal
    .startWithNext { next in print(next) }
```

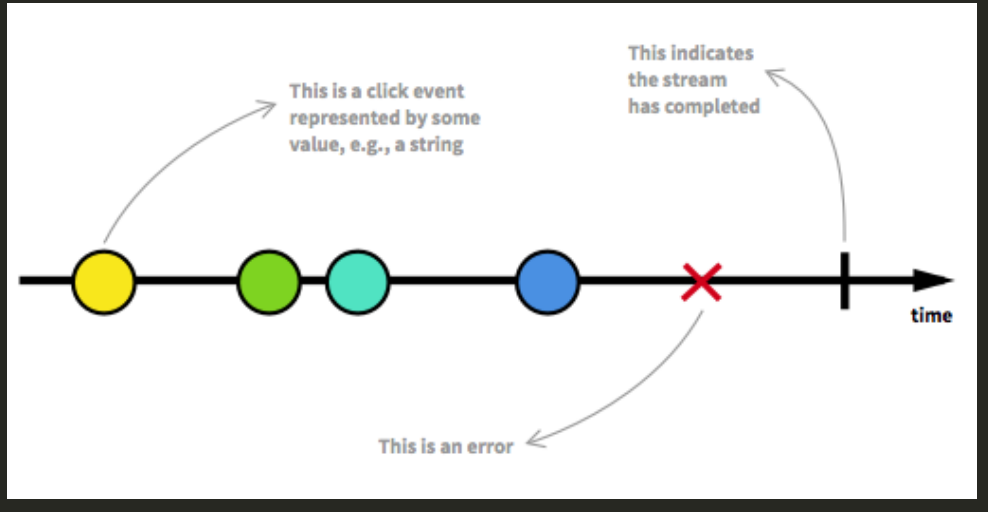
Create Producer

```
let signalProducer = SignalProducer<String, NoError> { observer, disposable in
    observer.sendNext("Test")
    observer.sendCompleted()
}
```

Operators

```
signal
 | |
_| |_
\   / transformation
 \ /
  V
signal //new
```

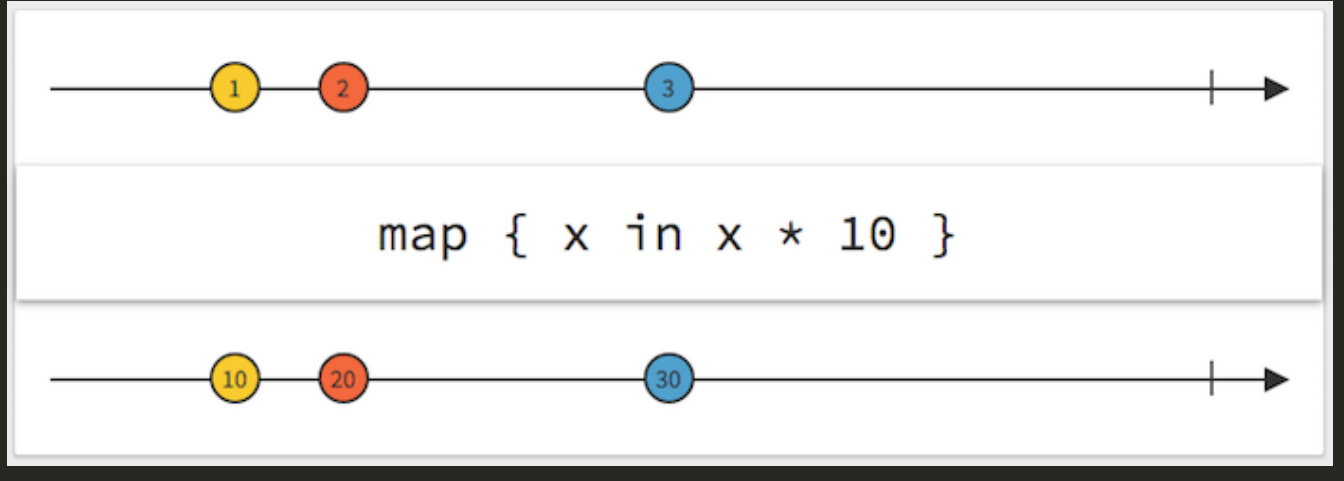
Marbles



rxmarbles.com

Transforming

.map()



```
usernameTextSignal
    .map({ (text: String) -> Bool in
        return characters.count > 3
    })
```

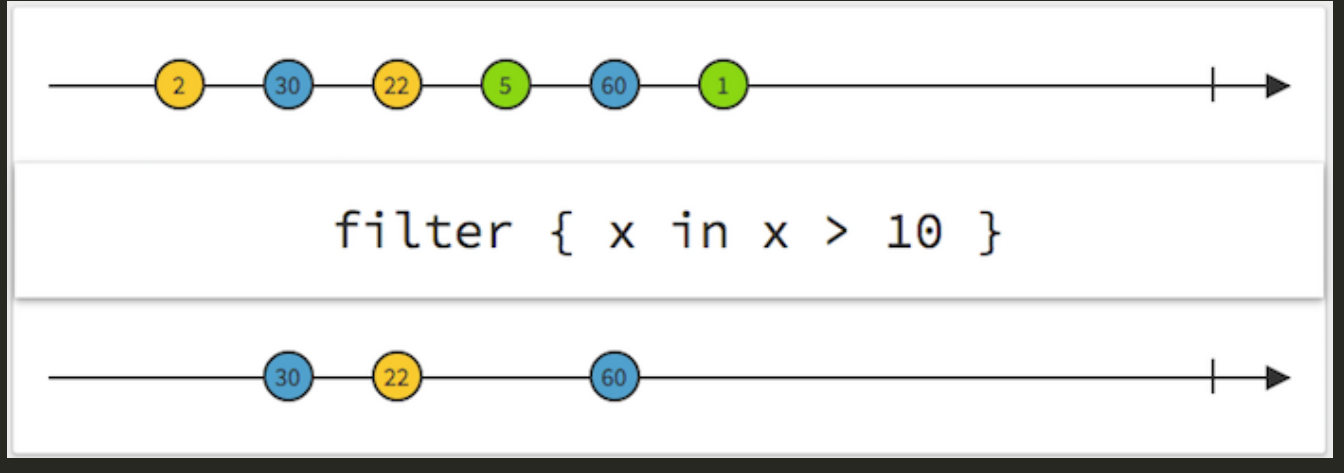
```
usernameTextSignal
    .map { $0.characters.count > 3 }
```

```
---h--he----hel---hell--hello---->
.map()
---f--f-----f-----t-----t-------->
```

```
let usernameValidSignal = usernameTextSignal.map { $0.characters.count > 3 }
```

Transforming

.filter()



```
validUsernameSignal
    .filter { $0 }
```

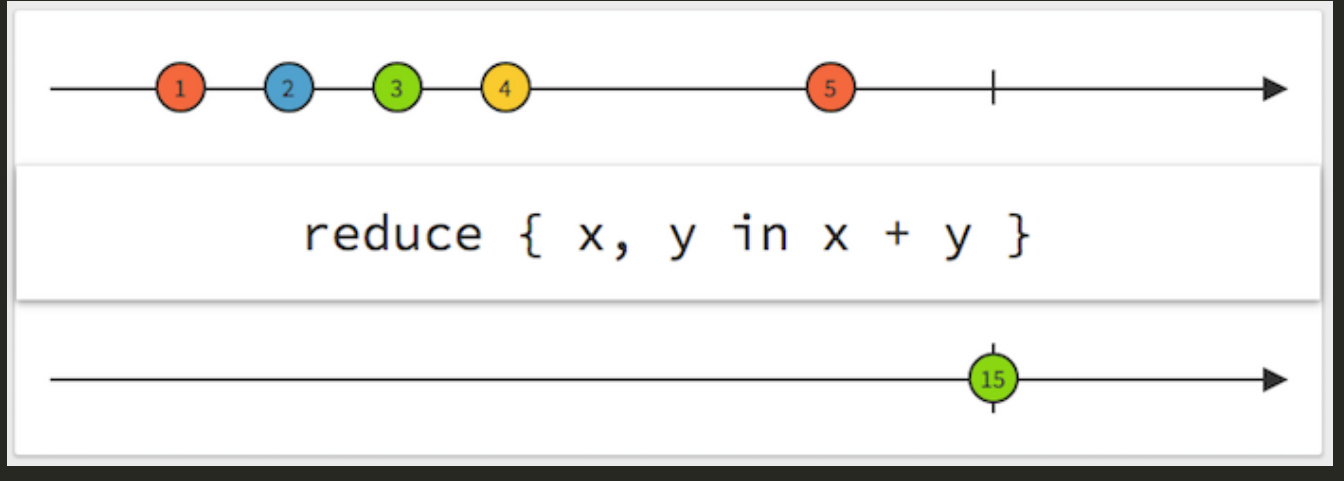
```
---f--f-----f----t-----t----------->

.filter()

-----------------t-----t----------->
```

Transforming

.reduce()

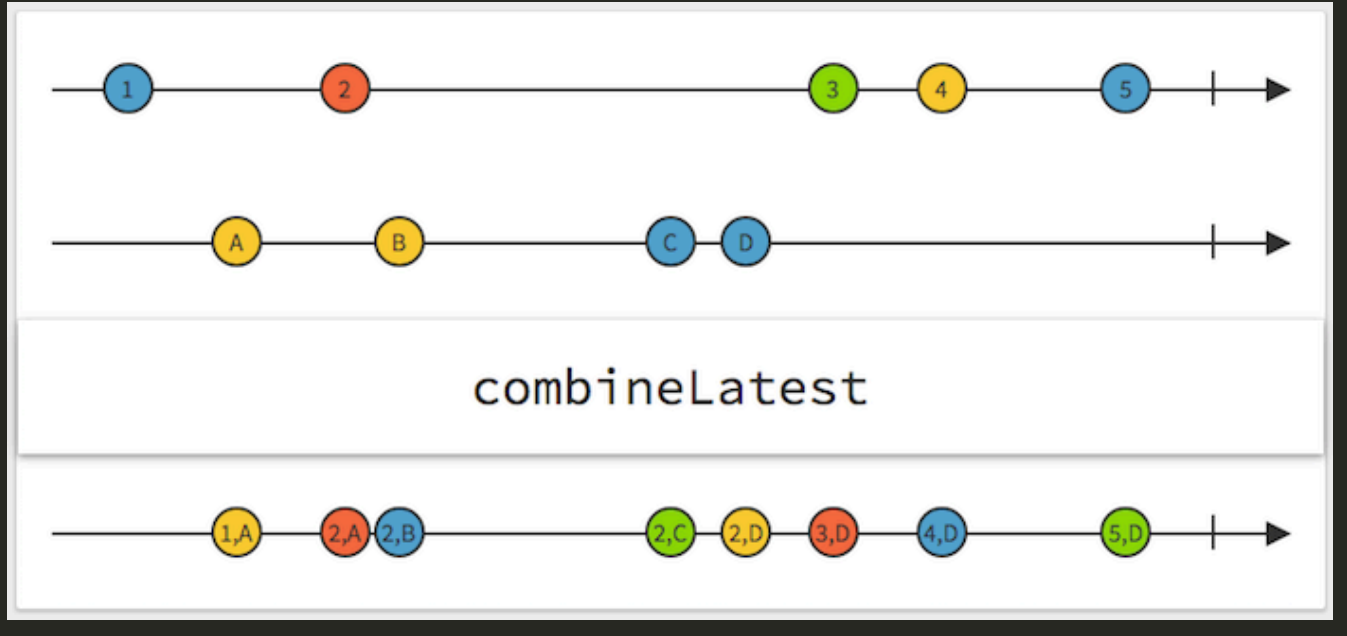


Combining

```
signal + signal
     | |
    _| |_
    \   / combine
     \ /
      V
    signal //new
```

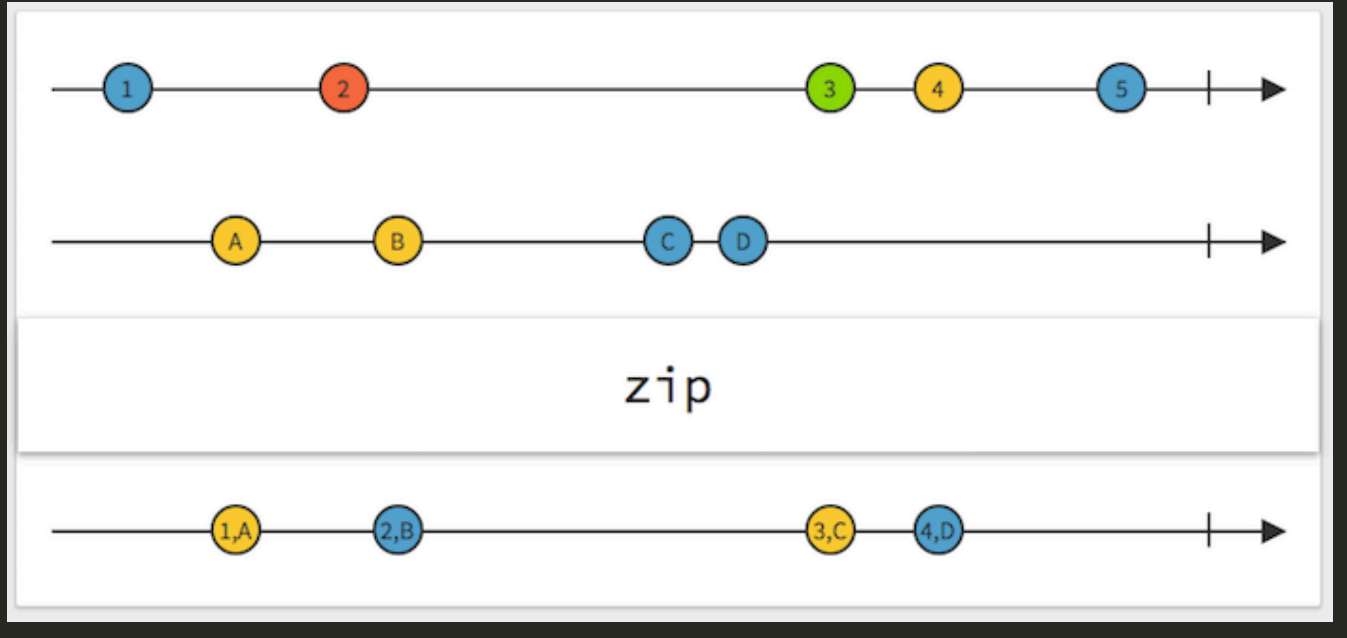
Combining

combineLatest(signalA, signalB)



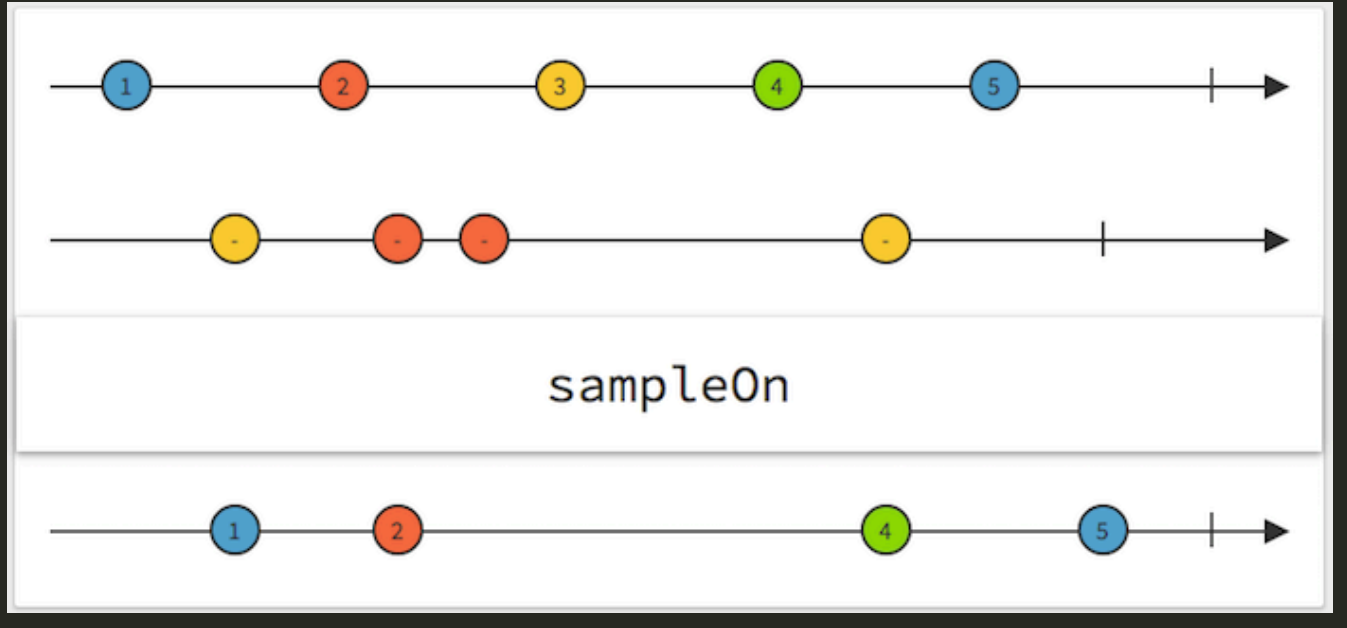
Combining

zip(signalA, signalB)



Combining

.sampleOn(signal)



Handling failures

.retry(count)

// Ignores failures up to count times.

.mapError()

```
signal
    .mapError { (error: NSError) -> CustomError in
        switch error.domain {
        ...
        }
    }
```

.flatMapError()

```
producer
    .flatMapError { _ in SignalProducer<String, NoError>(value: "Default") }
```

Side effects

.on()

// Injects side effects to be performed upon the specified signal events.

```
loginSignal()
    .on(failed: { error -> () in
        print(error)
    })
```

Scheduler

```
loginSignal()
    .observeOn(UIScheduler())
    .on(failed: { error -> () in
        self.activityIndicator.stopAnimating()
    })
```

Combining

Scenario: Search in Dictionary, receive result

Signal $_{t}$

S:----------t----------t-------t-------------t---->

```
.map({ text -> SignalProducer<Result, NSError>
    return self.fetchResultSignal(text)
})
```

Signal $_{\text{FetchResultSignal}\ R}$

S:----------S_R---------S_R-----S_R----------S_R--->

Combining - flatmap

Signal $_{t}$

S:----------t----------t-------t-------------t---->

```
.flatmap(.Latest, { text -> SignalProducer<Result, NSError>
    return self.fetchResultSignal(text)
})
```

FetchResultSignal $_{R}$

S:----------R----------R-------R-------------R---->

Demo

Reference

Talks

Justin Spahr Summers enemy of the state
Colin Eberhardt Swift, ReactiveCocoa…Better Together
Ash Furrow Functional Reactive Awesomeness With Swift
Erik Meijer What does it mean to be Reactive?

Documentation

ReactiveX reactivex.io
ReactiveCococa Documentation github.com

Helpful

Rac Marbles rac-marbles
Rx Marbles rx-marbles
colin eberhardt raywenderlich.com