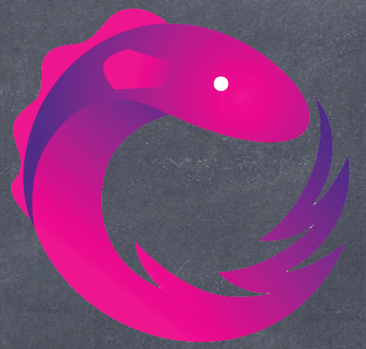
@alspirichev

Introduction to Reactive programming with RxSwift

RxSwift ??

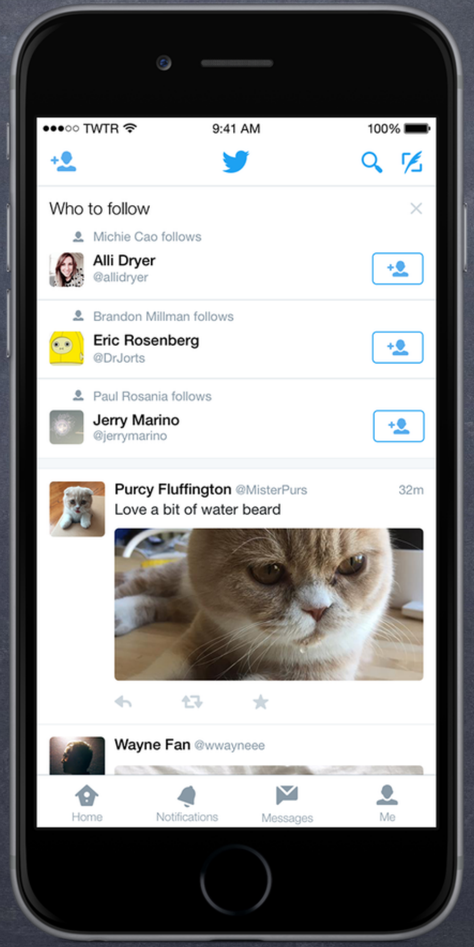


- *Rx is a multi-platform standard*
- *RxSwift - part of the suite of Rx (ReactiveX)*

- *While ReactiveX started as part of the .NET/C# ecosystem, it has grown extremely popular with Rubyists, JavaScripters and particularly Java and Android developers.*

Introduction to asynchronous programming

photo.jpg



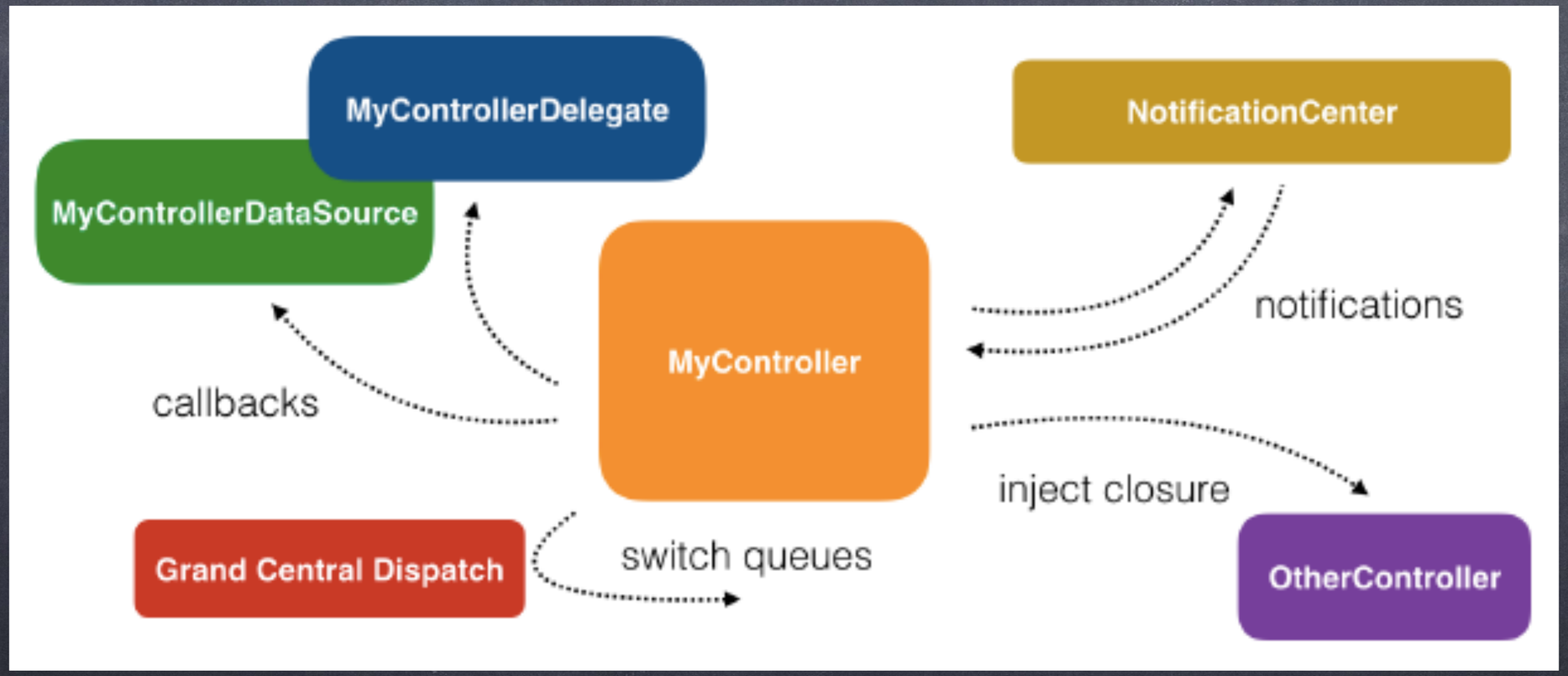
```
@IBAction func didTap {
    ...
}
```

hide/show keyboard

Introduction to asynchronous programming

- ***NotificationCenter***
- ***Delegates***
- ***GCD***
- ***Closures***

Introduction to asynchronous programming



Why use Rx?

- ***State***
- ***Bindings***
- ***Retries***
- ***Powerful operators***
- ***Easy integration***

Why use Rx?

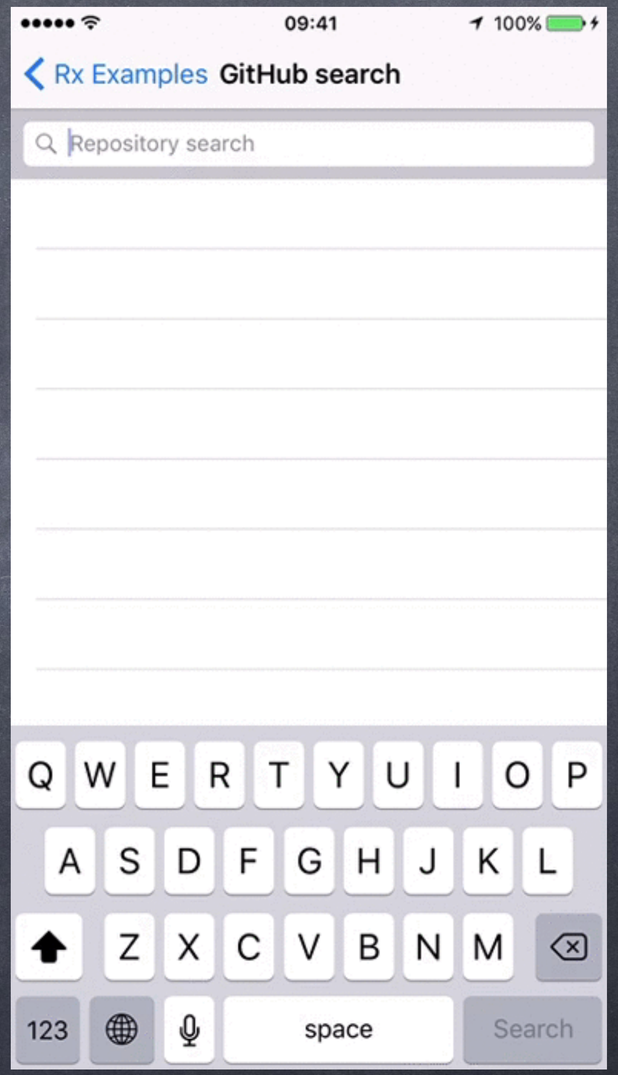
```
// imperative way

a = b + c

b = 5

// 'a' is not up to date
```

vs.

```
// reactive way

Rx enables building apps in
a declarative way.
```

Why use Rx?

```
let searchResults = searchBar.rx
    .text.orEmpty
    .throttle(0.3, scheduler: MainScheduler.instance)
    .distinctUntilChanged()
    .flatMapLatest { query -> Observable<[Repository]> in
        if query.isEmpty {
            return .just([])
        }
        return searchGitHub(query)
            .catchErrorJustReturn([])
    }
    .observeOn(MainScheduler.instance)
```



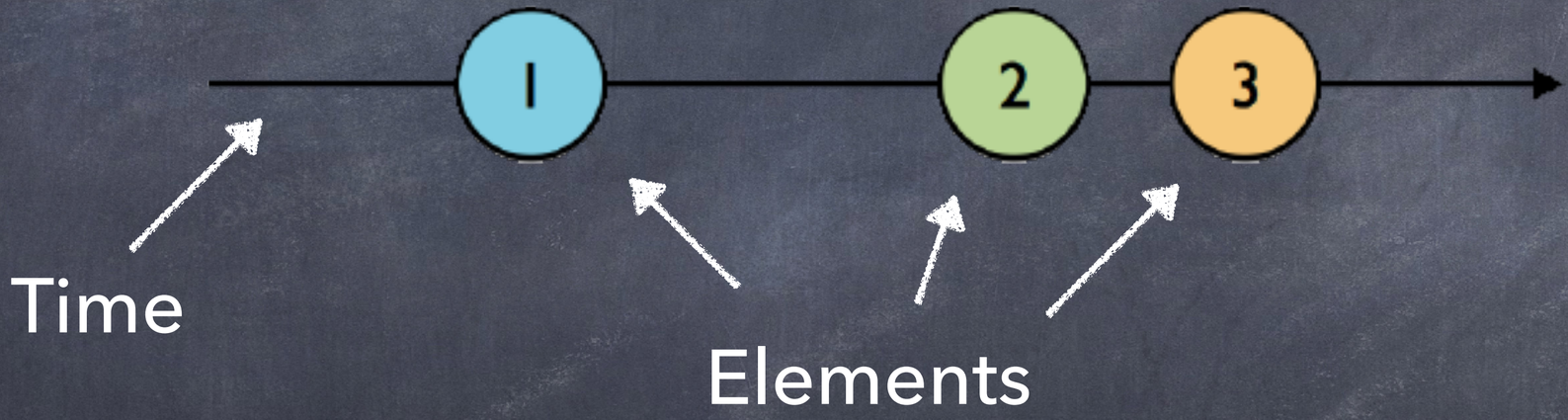
What does RxSwift consist of?

- ***Observables***
- ***Subjects***
- ***Operators***
- ***Schedulers***

Observables

Observables

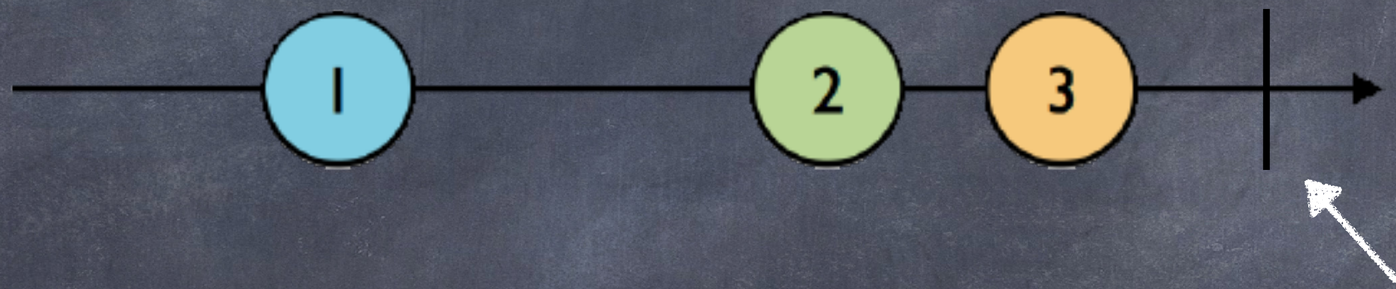
Marble diagram*



* www.rxmarbles.com

Observables

Marble diagram*

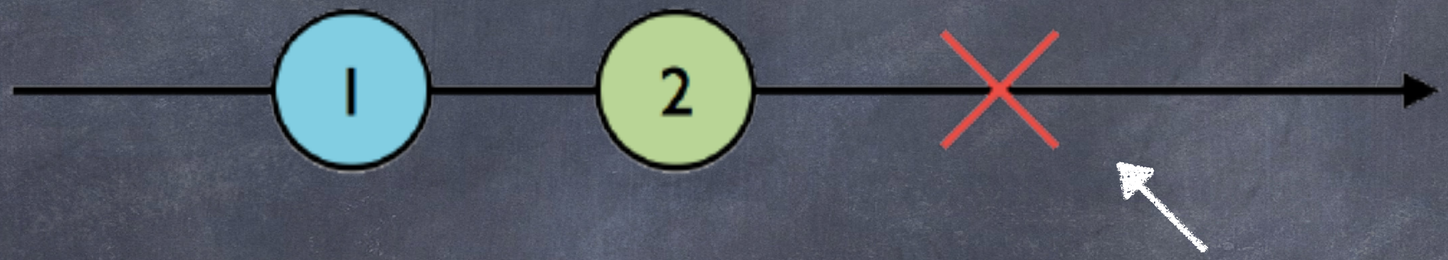


Completed

* www.rxmarbles.com

Observables

Marble diagram*



* www.rxmarbles.com

Observables

Represents a sequence event:

```
public enum Event<Element> {
    /// Next element is produced.
    case next(Element)

    /// Sequence terminated with an error.
    case error(Swift.Error)

    /// Sequence completed successfully.
    case completed
}
```

Observables

Creating observables:

```
let observable1 = Observable.of(1, 2 , 3, 4, 5)

let observable2: Observable<Planet> = Observable.from([.mercury, .earth, .mars])

let observable3 = Observable.just("MERA")
```

Observables

Creating observables:

```
Observable.create { observer in
    observer.onNext(1)
    observer.onNext(2)
    observer.onNext(3)

    observer.onCompleted()

    return Disposables.create()
}
```

Observables

Subscribing to observables:

```
observable1.subscribe { event in
    print(event)
}
```

Observables

Subscribing to observables:

```
observable1.subscribe(onNext: { element in
    print(element)
}, onError: { error in
    print(error.localizedDescription)
}, onCompleted: {
    print("Sequence is completed")

})
```

Observables

"Hot" and "Cold" Observables

- ***Cold observables emit events only when subscribed to.***
- ***Hot observables emit events independent of being subscribed to.***

Observables

Disposing and terminating

```
let observable = Observable.of("A", "B", "C")

let subscription = observable.subscribe { event in
    print(event)
}

// work with subscription

subscription.dispose()
```

Observables

Disposing and terminating

- ***Dispose Bags -*** *used to return ARC like behavior to Rx.*

```
let disposeBag = DisposeBag()

Observable.of("A", "B", "C")
    .subscribe { event in
        // work with event
    }
    .disposed(by: disposeBag)
```

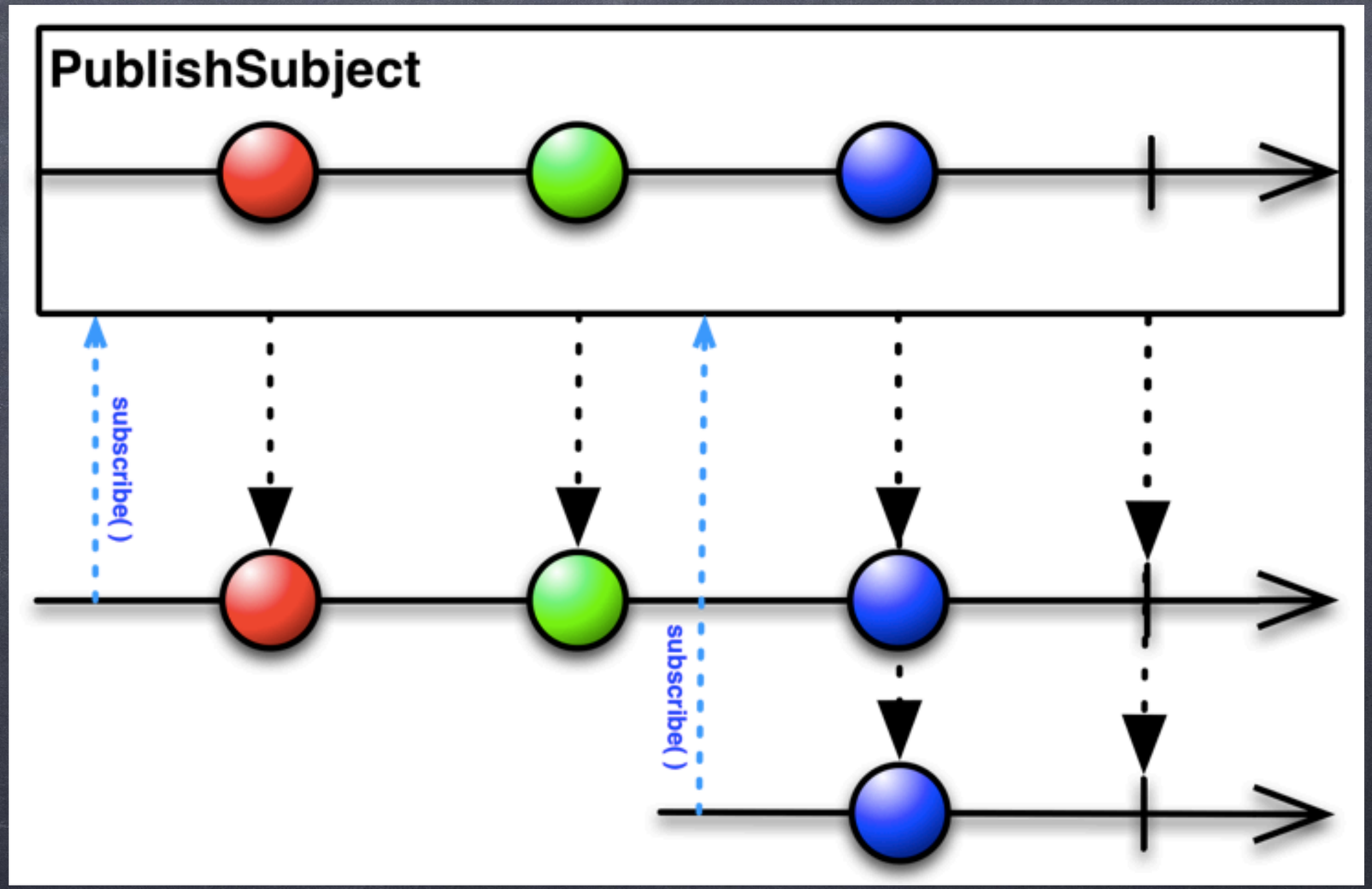
Subjects

Subjects

- ***Subjects act as both an observable and an observer.***
- ***There are four subject types in RxSwift:***
  - ***PublishSubject***
  - ***BehaviorSubject***
  - ***ReplaySubject***
  - ***Variable (will be deprecated)***

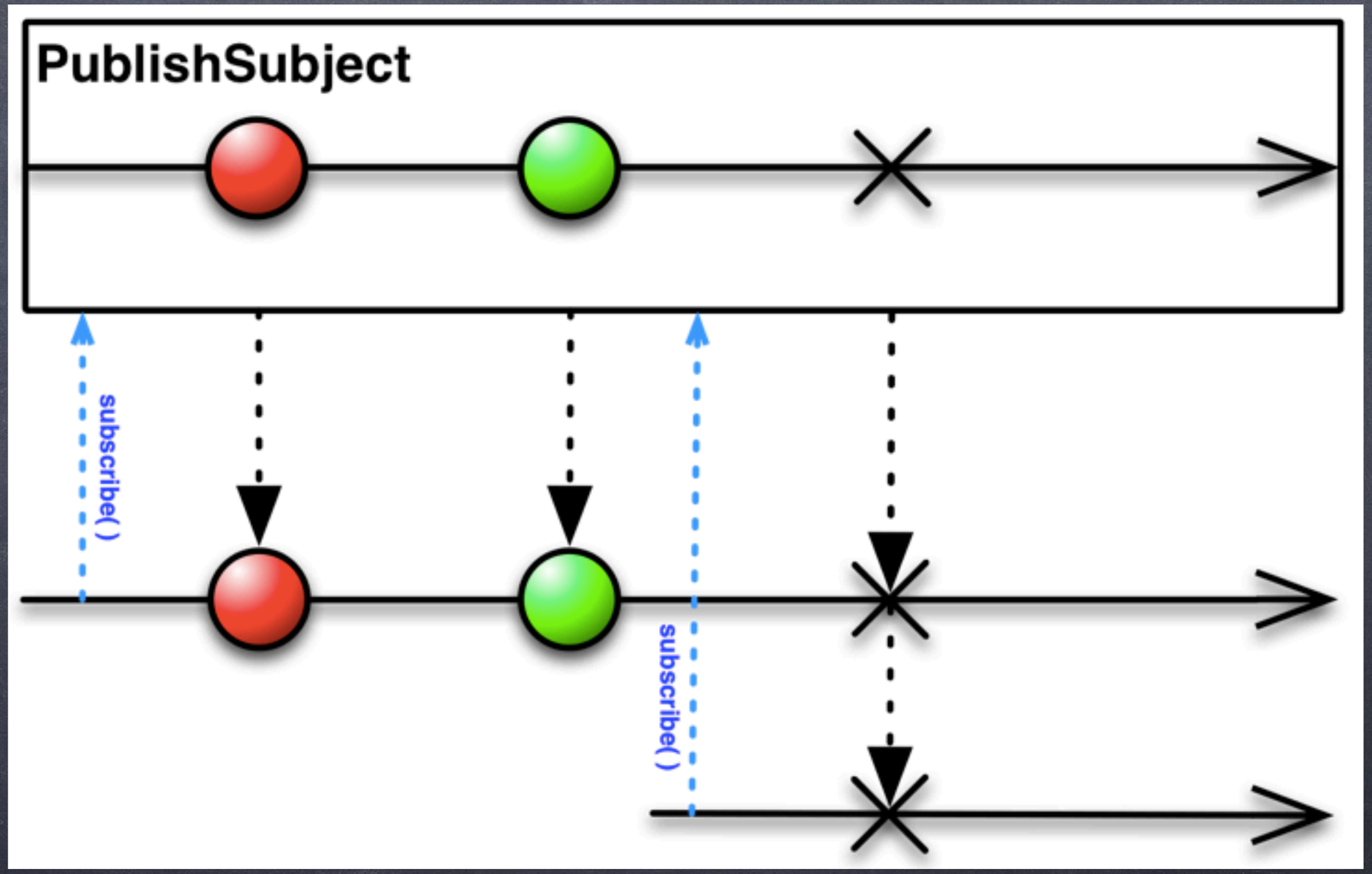
PublishSubject

- ***Starts empty and only emits new elements to subscribers***



PublishSubject

- *If the source Observable terminates with an error, the PublishSubject will not emit any items to subsequent observers, but will simply pass along the error notification from the source Observable.*

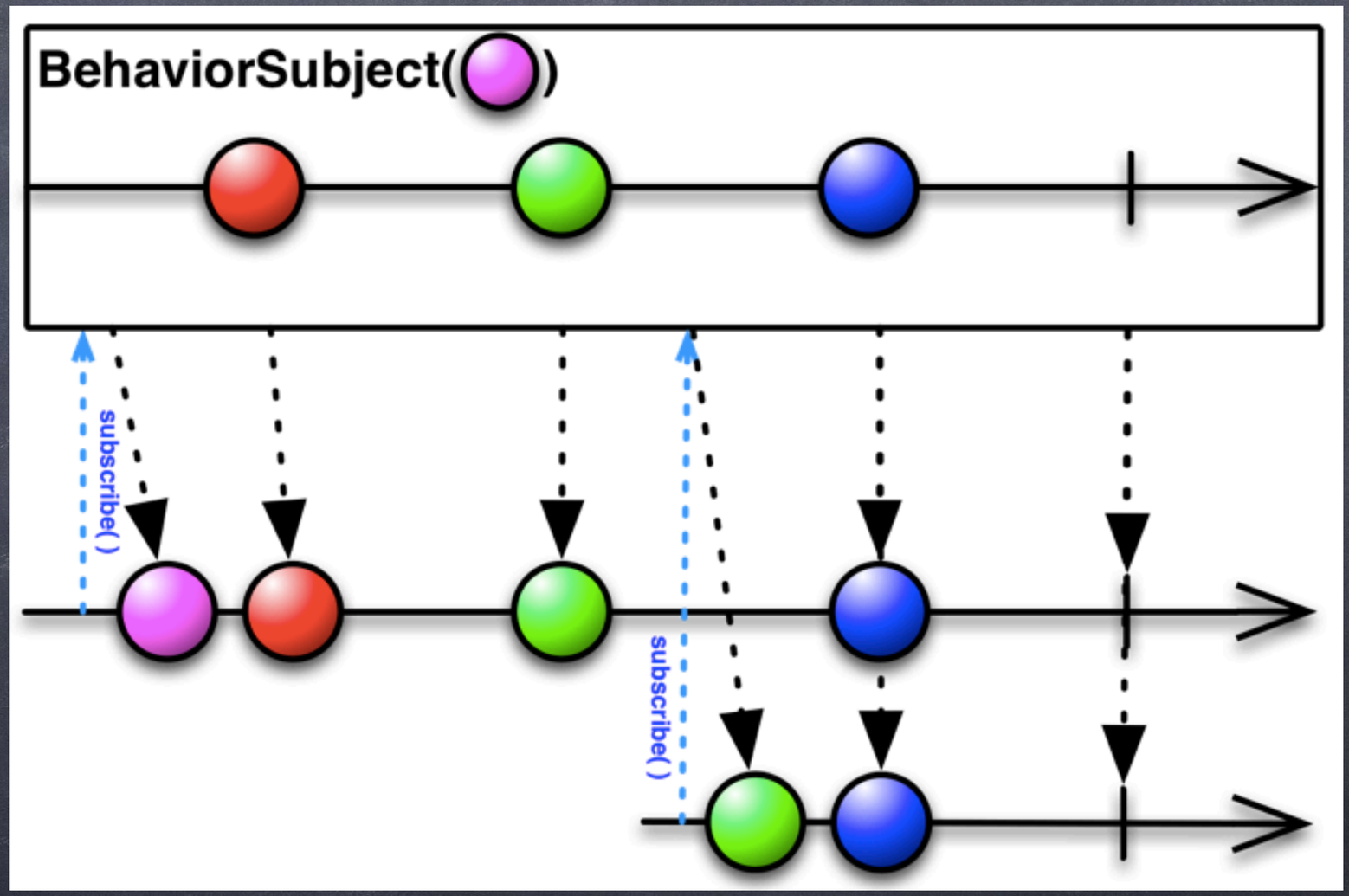


PublishSubject

```
let publishSubject = PublishSubject<String>()

publishSubject.onNext("Hello, ")

publishSubject.subscribe(onNext: { string in
    print(string)
})

publishSubject.onNext("world!")
```

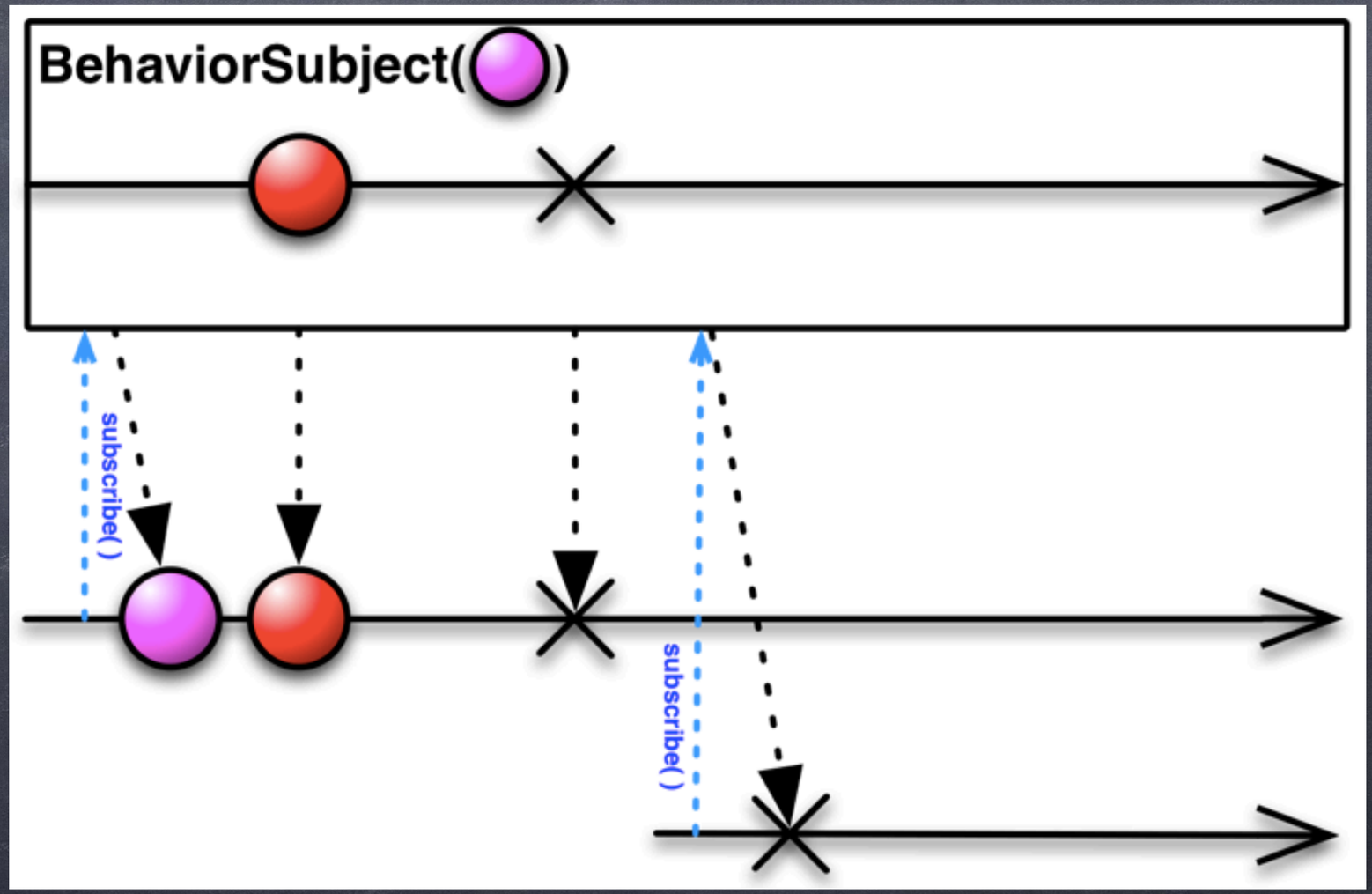
BehaviorSubjects

- ***Behavior subjects replay the latest .next event to new subscribers.***



BehaviorSubjects

- *if the source Observable terminates with an error, the BehaviorSubject will not emit any items to subsequent observers, but will simply pass along the error notification from the source Observable*

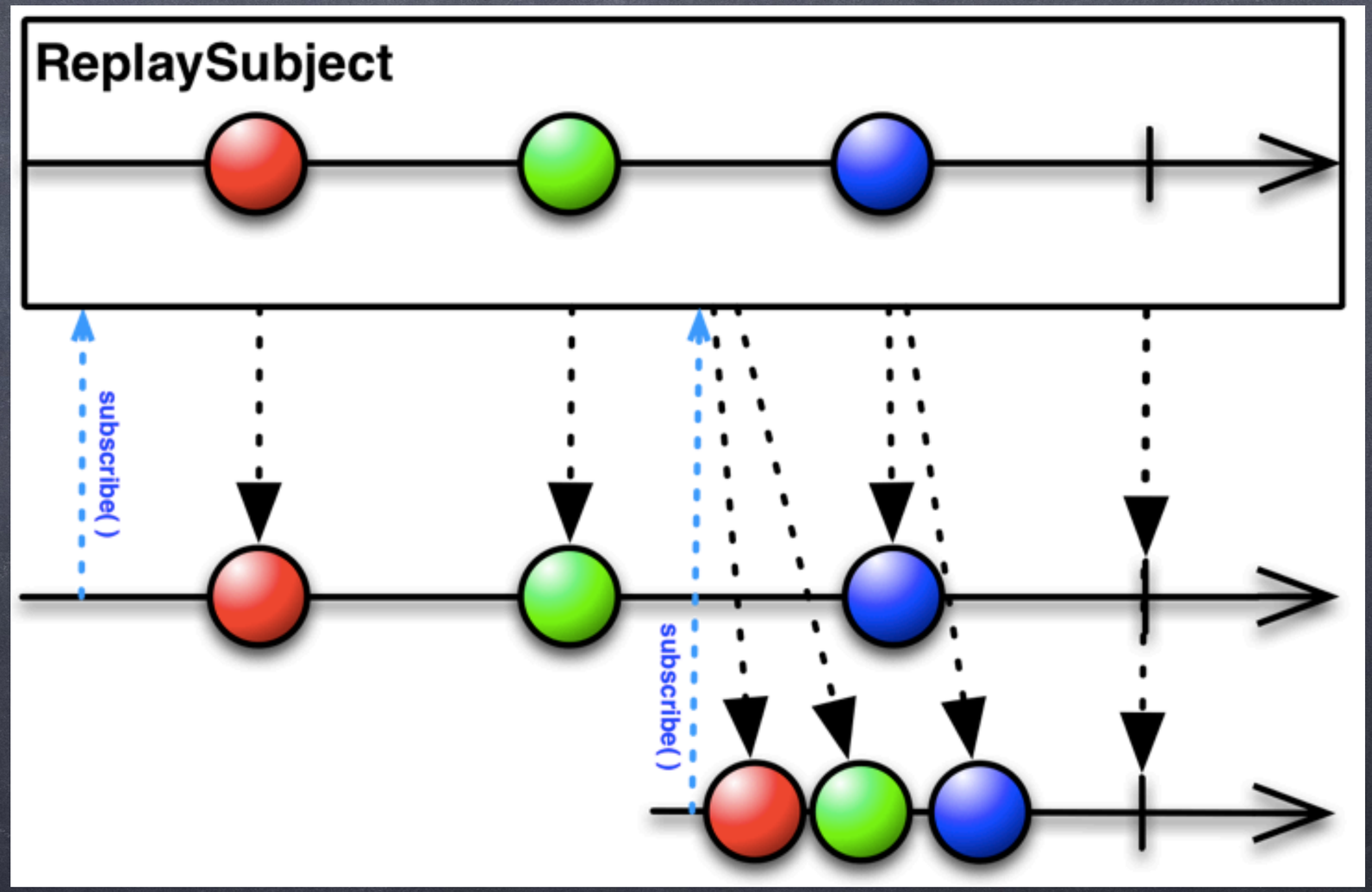


BehaviorSubjects

```
let behaviorSubject = BehaviorSubject(value: 4)

behaviorSubject.onNext(9)

behaviorSubject.subscribe(onNext: { integer in
    print(integer)
})

behaviorSubject.onNext(234)
```

ReplaySubject

- ***Replay subjects will temporarily cache, or buffer, the latest elements it emits, up to a specified size of your choosing.***



ReplaySubject

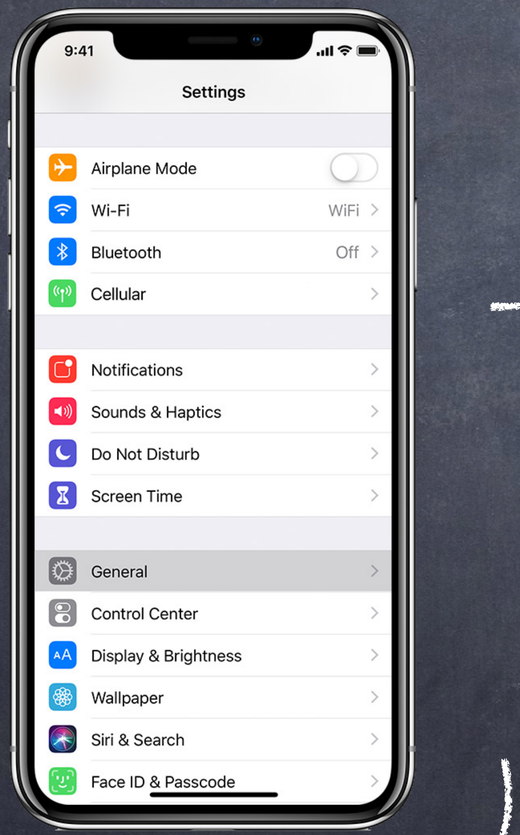
```swift
let relaySubject = ReplaySubject<String>.create(bufferSize: 3)

relaySubject.onNext("RxSwift")
relaySubject.onNext("is the best")

relaySubject.subscribe(onNext: { string in
    print(string)
})

relaySubject.onNext("of the best!")

relaySubject.subscribe(onNext: { string in
    print(string)
})
```

Operators

Operators

- ***Filtering Operators***
- ***Transforming Operators***
- ***Combining Operators***
- ***Time Based Operators***

Operators



.portrait → .landscape → .portrait →

Observable<Orientation>

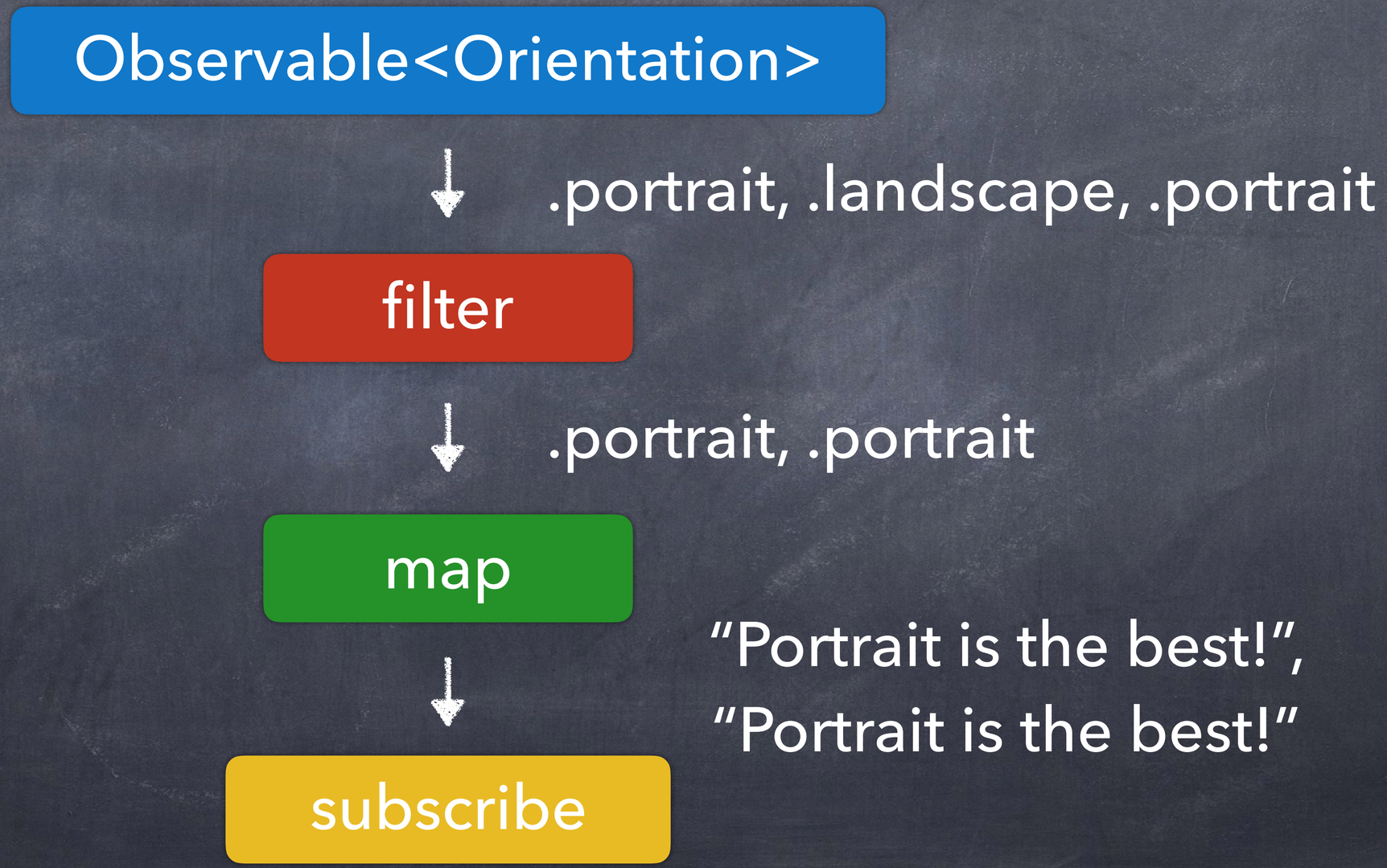
Operators

```
UIDevice.rx.orientation
    .subscribe(onNext: { current in
        switch current {
        case .landscape:
            ... re-arrange UI for landscape
        case .portrait:
            ... re-arrange UI for portrait
        }
})
```

Operators

```
UIDevice.rx.orientation
    .filter { value in
        return value != .landscape
    }
    .map { _ in
        return "Portrait is the best!"
    }
    .subscribe(onNext: { string in
        showAlert(text: string)
    })
```

Operators


Observable<Orientation>
.portrait, .landscape, .portrait
filter
.portrait, .portrait
map
“Portrait is the best!”,
“Portrait is the best!”
subscribe

Filtering Operators

- *filter()*
- *debounce()*
- *first() / last()*
- *takeLast(N)*
- *skip() / skipLast(N)*
- *skipWhile()*

Transforming Operators

- *map()*
- *flatMap()*
- *scan()*
- *groupBy()*

Combining Operators

- *join()*
- *merge()*
- *zip()*
- *switch()*

Time Based Operators

- *replay()*
- *replayAll()*
- *buffer()*
- *delay()*

Schedulers

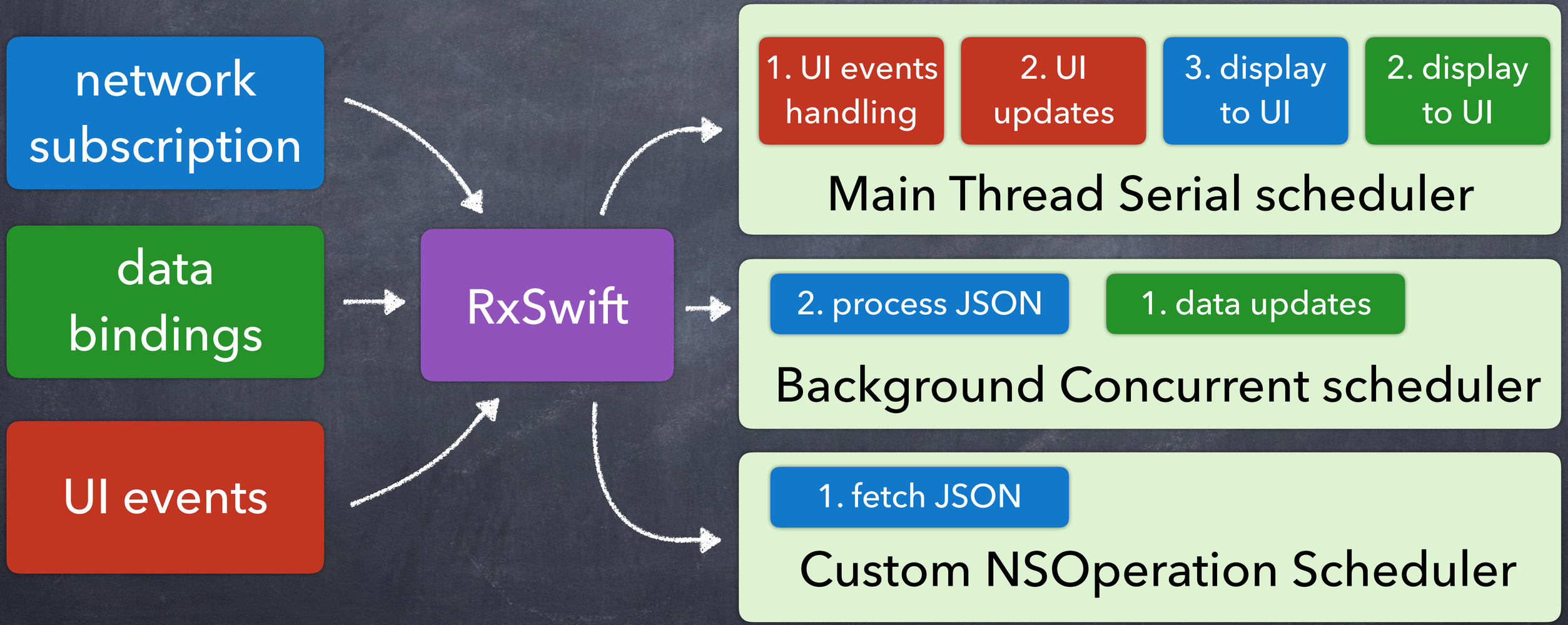
Schedulers

- ***Schedulers abstract away the mechanism for performing work.***
- ***Different mechanisms for performing work include the current thread, dispatch queues, operation queues, new threads, thread pools, and run loops.***
- ***RxSwift comes with a number of predefined schedulers, which cover 99% of use cases.***

Schedulers

- ***MainScheduler (Serial scheduler)***
- ***SerialDispatchQueueScheduler (Serial scheduler)***
- ***ConcurrentDispatchQueueScheduler (Concurrent scheduler)***
- ***OperationQueueScheduler (Concurrent scheduler)***
- ***TestScheduler***

Schedulers

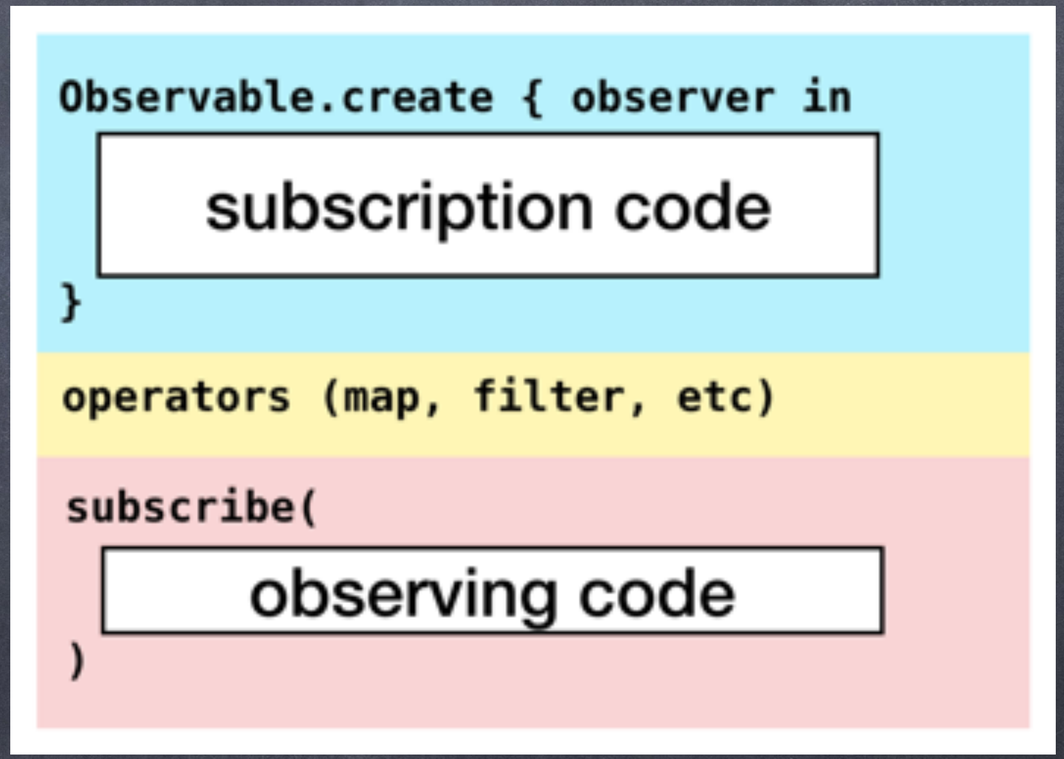

network subscription
data bindings
UI events
RxSwift
1. UI events handling
2. UI updates
3. display to UI
2. display to UI
Main Thread Serial scheduler
2. process JSON
1. data updates
Background Concurrent scheduler
1. fetch JSON
Custom NSOperation Scheduler

Schedulers

1.
```
Observable.create { observer in
    code to do the work
    to produce elements
}
```

2.
```
operators (map, filter, etc)
```

3.
```
subscribe(
  onNext: ...
  onCompleted:...
)
```



Schedulers

- ***subscribeOn() - specify the Scheduler on which an Observable will operate***

```
let globalScheduler = ConcurrentDispatchQueueScheduler(queue: DispatchQueue.global())

Observable.of(1, 2 , 3, 4, 5)
    .subscribeOn(globalScheduler)
    .subscribe { event in
        // work with event
    }
    .disposed(by: disposeBag)
```

Schedulers

- ***observeOn() - specify the Scheduler on which an observer will observe this Observable***

```
let globalScheduler = ConcurrentDispatchQueueScheduler(queue: DispatchQueue.global())

Observable.of(1, 2 , 3, 4, 5)
    .subscribeOn(globalScheduler)
    .observeOn(MainScheduler.instance)
    .subscribe { event in
        // work with event
    }
    .disposed(by: disposeBag)
```

RxCocoa

RxCocoa

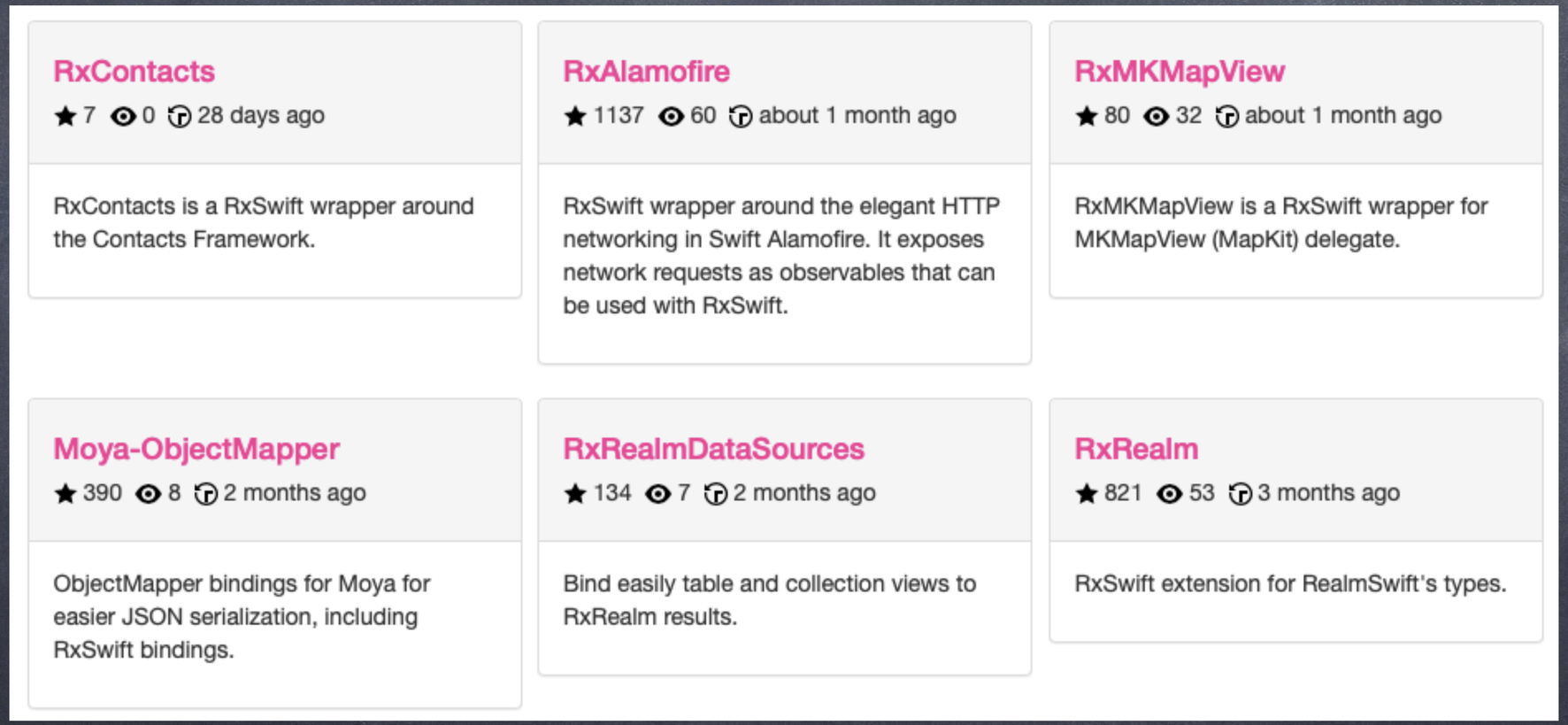
- ***RxSwift is the implementation of the common Rx API.***
- ***RxCocoa is UIKit and Cocoa specific.***

RxCocoa

```
toggleSwitch.rx.isOn
    .subscribe(onNext: { enabled in
        print( enabled ? "it's ON" : "it's OFF" )
    })
    .disposed(by: disposeBag)
```

RxCocoa adds the rx.isOn property (among others) to the UISwitch class so you can subscribe to generally useful event sequences.

RxSwift Community Cookbook



RxGesture

```
view.rx
  .tapGesture()
  .when(.recognized)
  .subscribe(onNext: { _ in
    //react to taps
  })
  .disposed(by: disposeBag)
```

```
view.rx
  .longPressGesture()
  .when(.began)
  .subscribe(onNext: { _ in
      // Do something
  })
  .disposed(by: disposeBag)
```

RxGesture

```
view.rx
  .anyGesture(.tap(), .swipe([.up, .down]))
  .when(.recognized)
  .subscribe(onNext: { _ in
    //dismiss presented photo
  })
  .disposed(by: disposeBag)
```

RxCoreLocation

```
locationManager.rx
    .didUpdateLocations
    .subscribe(onNext: { locationEvent in
        // locationEvent.manager
        // locationEvent.locations
    })
    .disposed(by: disposeBag)
```

```
locationManager.rx
    .placemark
    .subscribe(onNext: { placemark in
        // work with placemark
    })
    .disposed(by: disposeBag)
```

Benefits

- ***Declarative -> Because definitions are immutable and only data changes***
- ***Understandable and concise -> Raising the level of abstraction and removing transient states***
- ***Stable -> Because Rx code is thoroughly unit tested***
- ***Less stateful -> Because you are modeling applications as unidirectional data flows***
- ***Without leaks -> Because resource management is easy***

References

- https://github.com/ReactiveX/RxSwift/tree/master/Documentation
- https://habr.com/ru/post/423603/
- https://swiftbook.ru/post/tutorials/rxswift-in-10-minutes/
- https://www.raywenderlich.com/900-getting-started-with-rxswift-and-rxcocoa

Thank you for your attention!