ReactiveCocoa in Practice

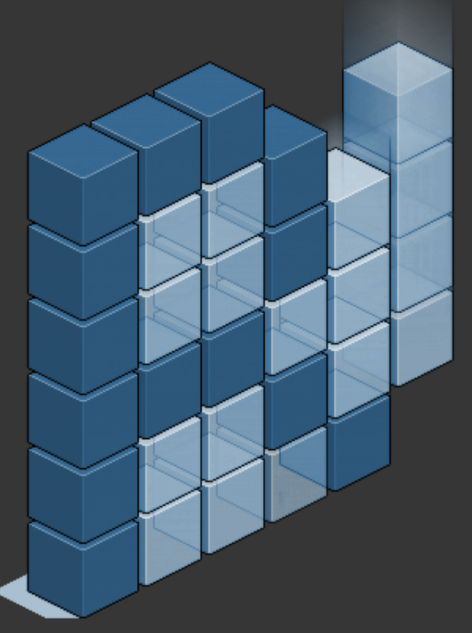
Jeames Bone

@jeamesbone

@outware
www.outware.com.au

Mark Corbyn

@markcorbyn



ReactiveCocoa

What is ReactiveCocoa?

Functional Reactive Programming (FRP) framework

for iOS and OSX applications.

Why ReactiveCocoa?

“Instead of telling a computer **how** to do it’s job, why don’t we just tell it **what** it’s job is, and let it figure the rest out.”

ReactiveCocoa in a Nutshell

The main component in ReactiveCocoa is the Signal.

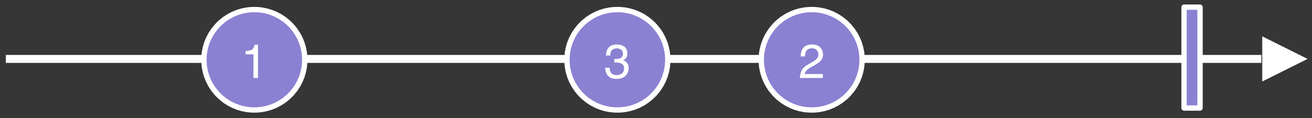
Signals represent streams of values over time.

Signals

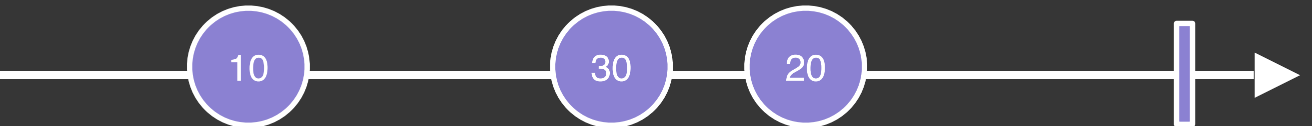
```
[textField.rac_textSignal subscribeNext:^(NSString *text) {
  NSLog(@"text: ‘%@’", text);
}];
```

```
> text: ‘h’
> text: ‘he’
> text: ‘hel’
```

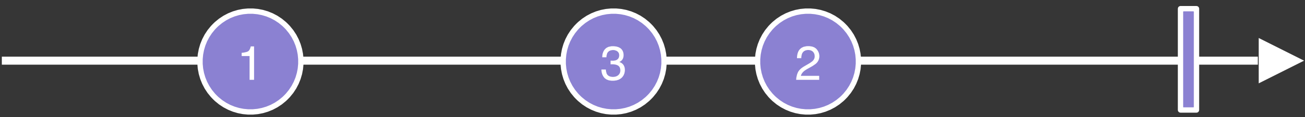
Operators




```
[signalA map:^(NSNumber *num) {
        return num * 10;
}];
```

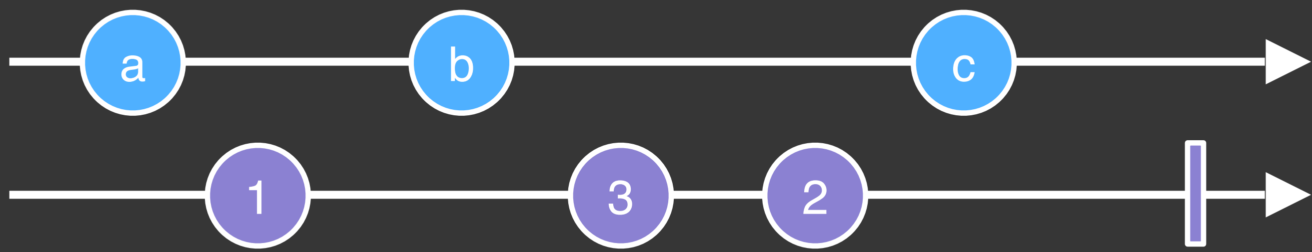


Operators

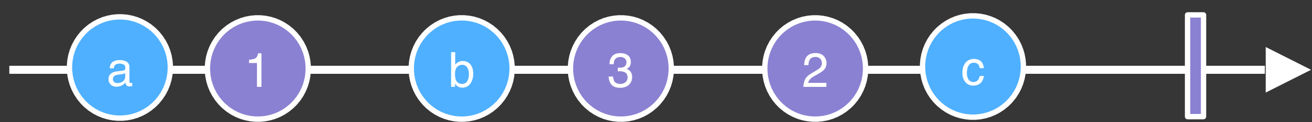



```
[signalA filter:^(NSNumber *num) {
        return num < 3;
}];
```

Operators



[signalA merge:signalB]



What’s Next?

1. Reactive View Controller

2. Reactive Notifications

3. Functional Data Processing

Something Simple

Authentication

```
/// Stores authentication credentials and tells us if we're authenticated.
@protocol AuthStore <NSObject>

@property (nonatomic, readonly) BOOL authenticated;

- (void)storeAccessCredentials:(AccessCredentials *)accessCredentials;
- (nullable AccessCredentials *)retrieveAccessCredentials;
- (void)removeAccessCredentials;

@end
```

Observe Authentication Changes

```
RACSignal *auth = RACObserve(authStore, authenticated);
```

Select the Right View Controller

```
RACSignal *authSignal = RACObserve(authStore, authenticated)

// Pick a view controller
RACSignal *viewControllerSignal =
  [authenticatedSignal map:^(NSNumber *isAuthenticated) {
    if (isAuthenticated.boolValue)
      return [DashboardViewController new];
    } else {
      return [LoginViewController new];
    }
  }];
```

Select the Right View Controller

map { // BOOL to ViewController }

Displaying it on Screen

How do we get the value out?
We have to subscribe, like a callback.

```
- (void)viewDidLoad {
  [super viewDidLoad];

  // Whenever we get a new view controller, PUSH IT
  [[viewControllerSignal deliverOnMainThread]
    subscribeNext:^(UIViewController *viewController) {
      [self showViewController:viewController sender:self];
    }];
}
```

Ok, maybe that’s pushing it

Let’s make a custom view controller container!

Switching to the latest view controller

Manages a signal of view controllers, always displaying the most recent one

```
@interface SwitchingController : UIViewController

- (instancetype)initWithViewControllers:(RACSignal *)viewControllers;

@property (nonatomic, readonly) UIViewController *currentViewController;

@end
```

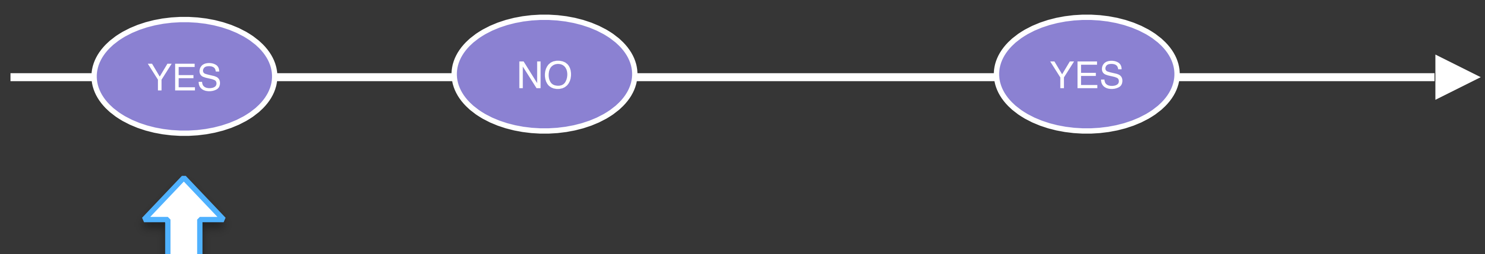
Setting Up

When the view loads, subscribe to our signal

```
- (void)viewDidLoad {
  [super viewDidLoad];

  [[self.viewControllers deliverOnMainThread]
    subscribeNext:^(UIViewController *viewController) {
    [self transitionFrom:self.currentViewController to:viewController];
  }];
}
```

Transitioning

```
- (void)transitionFrom:(UIViewController *)fromViewController
                  to:(UIViewController *)toViewController {
  if (!fromViewController) {
    [self addInitialViewController:toViewController];
    return;
  }

  [self addChildViewController:nextViewController];
  nextViewController.view.frame = self.view.bounds;
  [self.view addSubview:nextViewController.view];
  [previousViewController willMoveToParentViewController:nil];

  [self transitionFromViewController:fromViewController
                  toViewController:toViewController
                          duration:0.5
                           options:UIViewAnimationOptionTransitionCrossDissolve
                        animations:nil
                        completion:^(BOOL finished) {
                          [toViewController didMoveToParentViewController:self];
                          [fromViewController removeFromParentViewController];
                          self.currentViewController = toViewController;
                        }];
}
```



Dashboard

Finishing Up

What happens if the view controller changes rapidly?

Simple Throttling

Only take a new view controller if 0.5 seconds have passed.

```
- (void)viewDidLoad {
  [super viewDidLoad];

  [[[self.viewControllers
    throttle:0.5]
    deliverOnMainThread]
    subscribeNext:^(UIViewController *viewController) {
      [self transitionFrom:self.currentViewController
                        to:viewController];
    }];
}
```

v2: Only throttle while animating

Keep track of when we're animating

```
- (void)transitionFrom:(UIViewController *)fromViewController
                  to:(UIViewController *)toViewController {
  // code
  self.animating = YES;

  [self transitionFromViewController:fromViewController
                  toViewController:toViewController
                          duration:0.5
                           options:UIViewAnimationOptionTransitionCrossDissolve
                        animations:nil
                        completion:^(BOOL finished) {
                          // more code
                          self.animating = NO;
                        }];
}
```

v2: Only throttle while animating

Throttle only if we are animating

```
[[[self.viewControllers
   throttle:0.5 valuesPassingTest:^(id _) {
     return self.isAnimating;
   }]
   deliverOnMainThread]
   subscribeNext:^(UIViewController *viewController) {
     [self transitionFrom:self.currentViewController to:viewController];
   }];
```

What have we learned?

- Signals can represent real world events (Logging in/out)
- We can transform them using operators like map
- We can control timing through operators like throttle

~~1. Reactive View Controller~~

2. Reactive Notifications

3. Functional Data Processing

Hot Signals

- Events happen regardless of any observers.
- Stream of *events* happening in the world.
- e.g. UI interaction, notifications

Cold Signals

- Subscribing starts the stream of events.
- Stream of ***results*** caused by some side effects.
- e.g. network calls, database transactions

Push Notifications

```
- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo {
  // Do something horrible™ in here
}
```

A Better Option

```
typedef NS_ENUM(NSUInteger, NotificationType) {
  NotificationTypeA,
  NotificationTypeB,
  NotificationTypeC,
};

@protocol NotificationProvider

- (RACSignal *)notificationSignalForNotificationType:
    (NotificationType)type;

@end
```

We Want This:

```
@property id<NotificationProvider> notificationProvider;

- (void)viewDidLoad {
  [[[self.notificationProvider
    notificationSignalForNotificationType:NotificationTypeA]
    deliverOnMainThread]
    subscribeNext:^(Notification *notification) {
      [self updateInterfaceWithNotification:notification];
    }]
  }
```

A New Friend!

Lift a selector into the reactive world

```
- (RACSignal *)rac_signalForSelector:(SEL)selector;
```

The returned signal will fire an event every time the method is called.

Let’s Do It!

```
- (RACSignal *)notificationSignalForNotificationType:
    (NotificationType)type {
  return [[[self
    rac_signalForSelector:@selector(application:didReceiveRemoteNotification:)]
    map:^(RACTuple *arguments) {
      return arguments.second;
    }]
    map:^(NSDictionary *userInfo) {
      // Parse our user info dictionary into a model object
      return [self parseNotification:userInfo];
    }]
    filter:^(Notification *notification)
      notification.type = type;
    }];
}
```

```
map { [self parseNotification:userInfo]; }
```

```
filter { notification.type == typeA }
```

Notificati on A

Notificati on A

- We don't want to duplicate our current notification handling.
- Local and remote notifications should have the same effects.

```
RACSignal *remoteNotificationInfo = [[self
  rac_signalForSelector:@selector(application:didReceiveRemoteNotification:)]
  map:^(RACTuple *arguments) {
    return arguments.second
  }];

RACSignal *localNotificationInfo = [[self
  rac_signalForSelector:@selector(application:didReceiveLocalNotification:)]
  map:^(RACTuple *arguments) {
    UILocalNotification *notification = arguments.second;
    return notification.userInfo;
  }];

return [[[remoteNotificationInfo merge:localNotificationInfo]
    map:^(NSDictionary *userInfo) {
      // Parse our user info dictionary into a model object
      return [self parseNotification:userInfo];
    }]
        filter:^(Notification *notification) {
          return notification.type == type;
        }];
```

Problem

- We only get notifications sent *after* we subscribe.
- We can't easily update app state or UI that is created after the notification is sent.

Solution?

[signal replayLast]

Whenever you subscribe to the signal, it will immediately send you the most recent value from the stream.

Replay it

```
return [[remoteNotificationInfo merge:localNotificationInfo]
  map:^(NSDictionary *userInfo) {
    // Parse our user info dictionary into a model object
    return [self parseNotification:userInfo];
  }]
  filter:^(Notification *notification) {
    return notification.type == type;
  }];
```

What have we learned?

- Signals can model complex app behaviour like notifications
- We can combine signals in interesting ways
- Helpers like signalForSelector allow us to lift regular functions into signals

~~1. Reactive View Controller~~

~~2. Reactive Notifications~~

3. Functional Data Processing

Describing an Algorithm with Functions

Example: Finding the best voucher to cover a purchase

- If there are any vouchers with higher value than the purchase, use the lowest valued of those
- Otherwise, use the highest valued voucher available

Imperative Approach

```
NSArray *vouchers = [[self voucherLibrary] vouchers];

NSArray *sortedVouchers = [vouchers
     sortedArrayUsingComparator:^NSComparisonResult(id<Voucher> voucher1,
                                                    id<Voucher> voucher2) {
       return [voucher1 compare:voucher2];
     }];


  id<Voucher> bestVoucher = nil;
  for (id<Voucher> voucher in sortedVouchers) {
    NSDecimalNumber *voucherAmount = voucher.amount;
    if (!voucherAmount) continue;

    if ([voucherAmount isLessThan:purchaseAmount]) {
      bestVoucher = voucher;
    } else if ([voucherAmount isGreaterThanOrEqualTo:purchaseAmount]) {
      bestVoucher = voucher;
      break;
    }
  }

  return bestVoucher;
}
```

Separate the Filter?

```
NSMutableArray *vouchersWithValue = [NSMutableArray array];
for (id<Voucher> voucher in sortedVouchers) {
  if (voucher.amount) {
    [vouchersWithValue addObject:voucher];
  }
}

id<Voucher> bestVoucher = nil;
for (id<Voucher> voucher in vouchersWithValue) {
  NSDecimalNumber *voucherAmount = voucher.amount;
  if ([voucherAmount isLessThen:purchaseAmount]) {
    bestVoucher = voucher;
  } else if ([voucherAmount isGreaterThanOrEqualTo:purchaseAmount]) {
    bestVoucher = voucher;
    break;
  }
}
```

Functional Approach

```
- (id<Voucher>)voucherForPurchaseAmount:(NSDecimalNumber *)purchaseAmount {
    [[[[[[[self voucherLibrary]
    vouchers]
    sortedArrayUsingComparator:^NSComparisonResult(id<Voucher> voucher1,
                                                   id<Voucher> voucher2) {
      return [voucher1 compare:voucher2];
    }]
    rac_sequence]
    filter:^BOOL(id<Voucher> voucher) {
      return voucher.amount != nil;
    }]
    yow_takeUptoBlock:^BOOL(id<Voucher> voucher) {
      return [voucher.amount isGreaterThanOrEqualTo:purchaseAmount];
    }]
    array]
    lastObject];
}
```

```
NSArray *vouchers = [[self voucherLibrary] vouchers];

NSArray *sortedVouchers = [vouchers
    sortedArrayUsingComparator:^NSComparisonResult(id<Voucher> voucher1,
                                                   id<Voucher> voucher2) {
      return [voucher1 compare:voucher2];
    }];


  id<Voucher> bestVoucher = nil;
  for (id<Voucher> voucher in sortedVouchers) {
    NSDecimalNumber *voucherAmount = voucher.amount;
    if (!voucherAmount) continue;

    if ([voucherAmount isLessThan:purchaseAmount]) {
      bestVoucher = voucher;
    } else if ([voucherAmount isGreaterThanOrEqualTo:purchaseAmount]) {
      bestVoucher = voucher;
      break;
    }
  }

  return bestVoucher;
}
```

What have we learned?

- Functional code can be more readable
- It's easy to convert between imperative and functional code using ReactiveCocoa
- Signals are lazy

~~1. Reactive View Controller~~

~~2. Reactive Notifications~~

~~3. Functional Data Processing~~

Resources

- reactivecocoa.io
- ReactiveCocoa on GitHub

- ReactiveCocoa - The Definitive Introduction (Ray Wenderlich)
- A First Look at ReactiveCocoa 3.0 (Scott Logic)

Next Steps

- Try it out!
- Don’t be scared to go all in

Thanks!

Questions?