Intro to Reactive Extensions

Why you want to learn about them

Juanjo Ramos

Agenda

Definitions

" Reactive Extensions is a set of tools allowing imperative programming languages to operate on sequences of data regardless of whether the data is synchronous or asynchronous.

Wikipedia

“The Reactive Extensions (Rx) is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators.

Microsoft

“ Using Rx, developers represent asynchronous data streams with Observables

“ query asynchronous data streams using LINQ operators

“ and parameterize the concurrency in the asynchronous data streams using Schedulers

Microsoft

“ An API for asynchronous programming with observable streams

ReactiveX.io

GCD

Operations

Async / Await

Futures

Promises

Tasks

Coroutines

LET ME DIE

It's all about asynchronous events

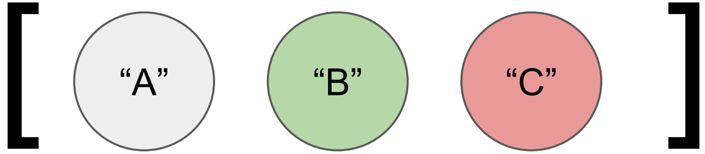
Asynchronous Event Streams

- Event Bus
- Message Broker
- User tapping on a button

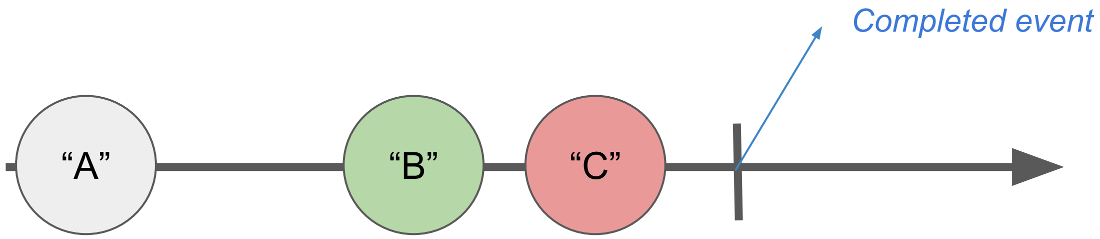
Events can contain

- Value
- Error
- Completed signal

Error and Completed terminates the sequence



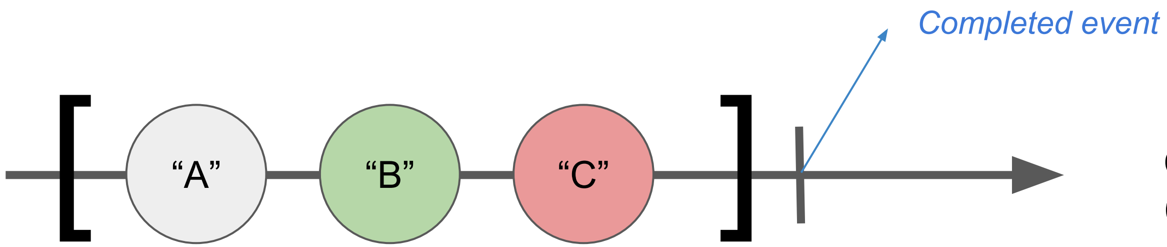
Array of (String) values



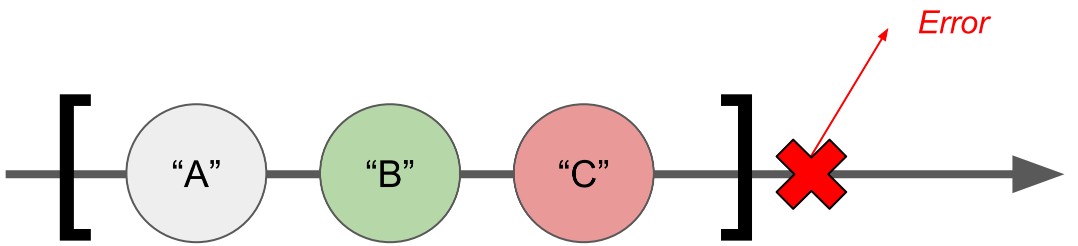
Observable of (String) values

The stream is the *Observable*

Observers subscribe **to *Observables* and *react* to emitted events**



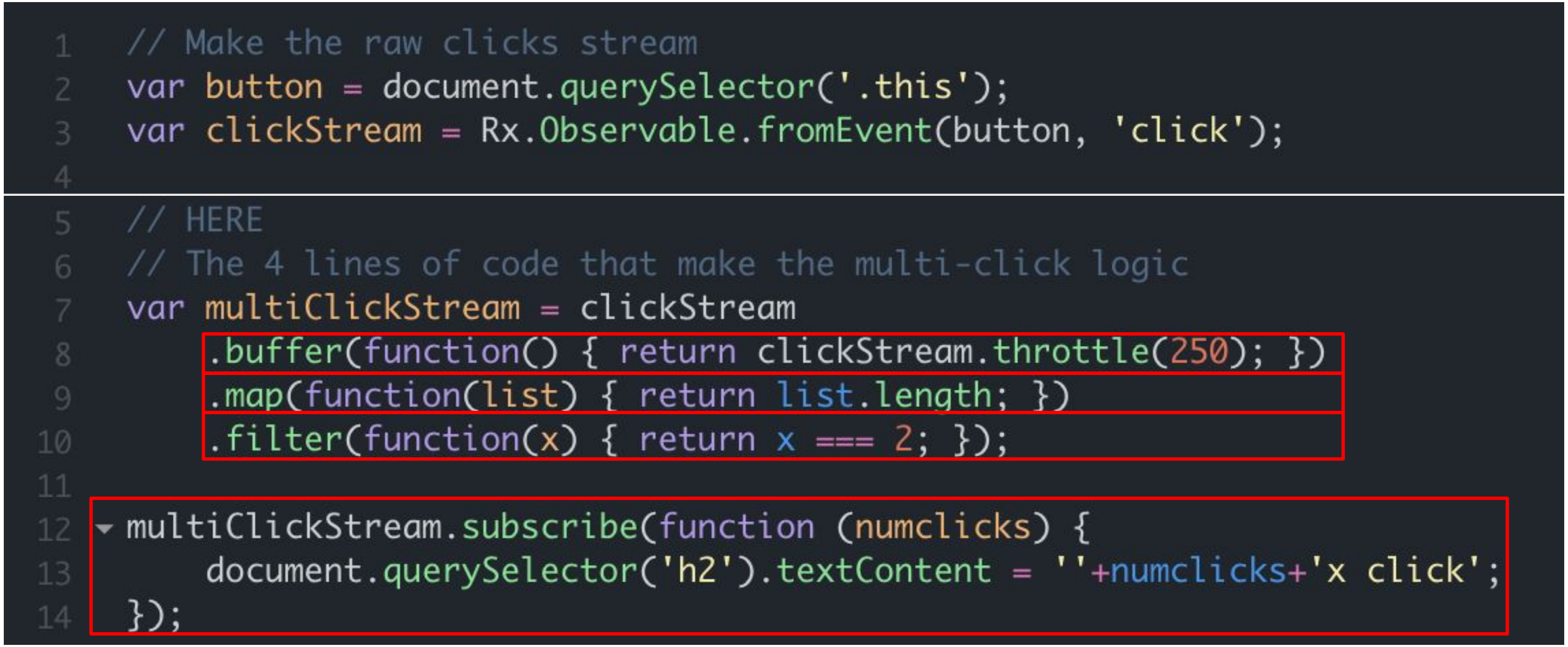
Observable of Array of (String) values



Observable of Array of (String) values

The power of Operators

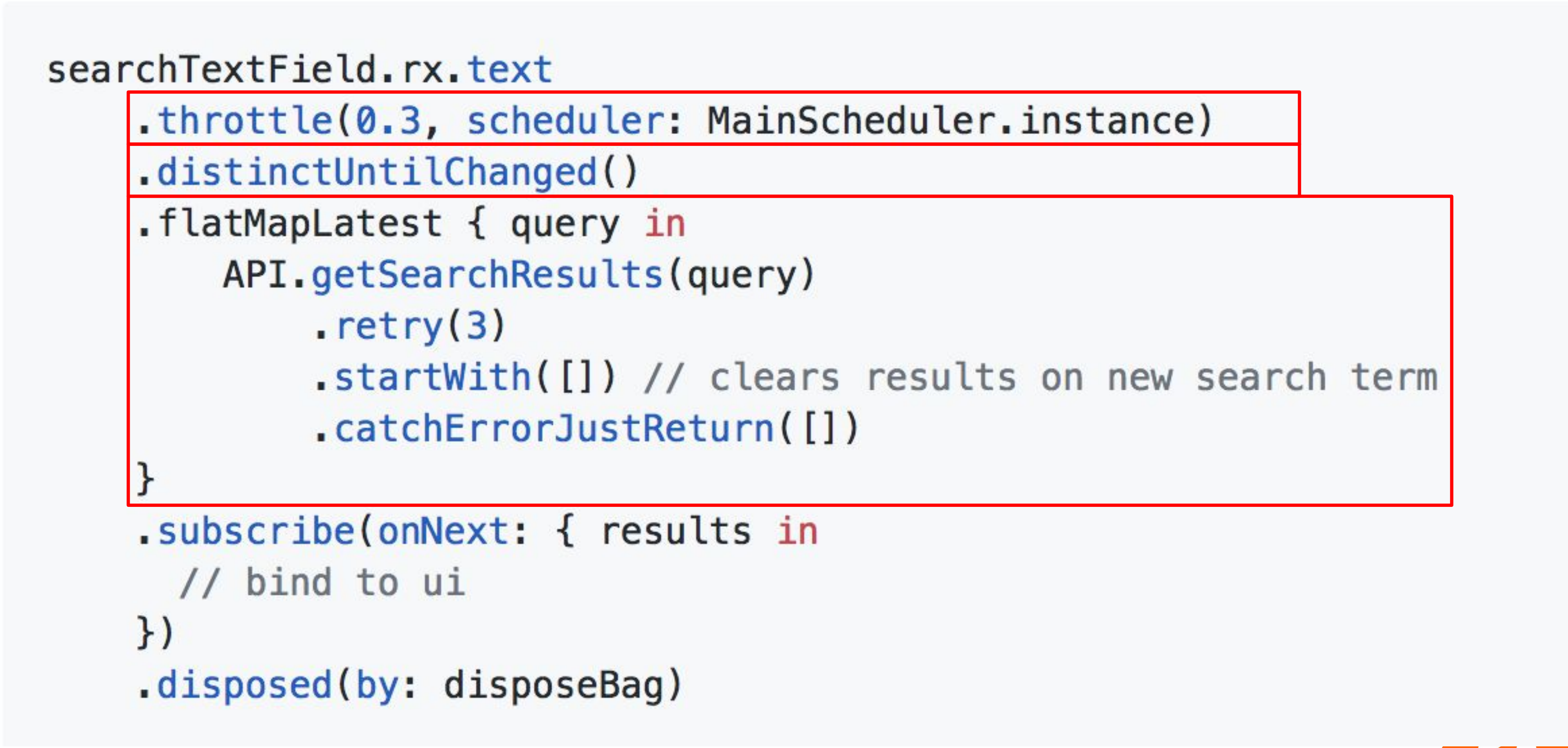
Example: Do some operation after the user double clicks on a button



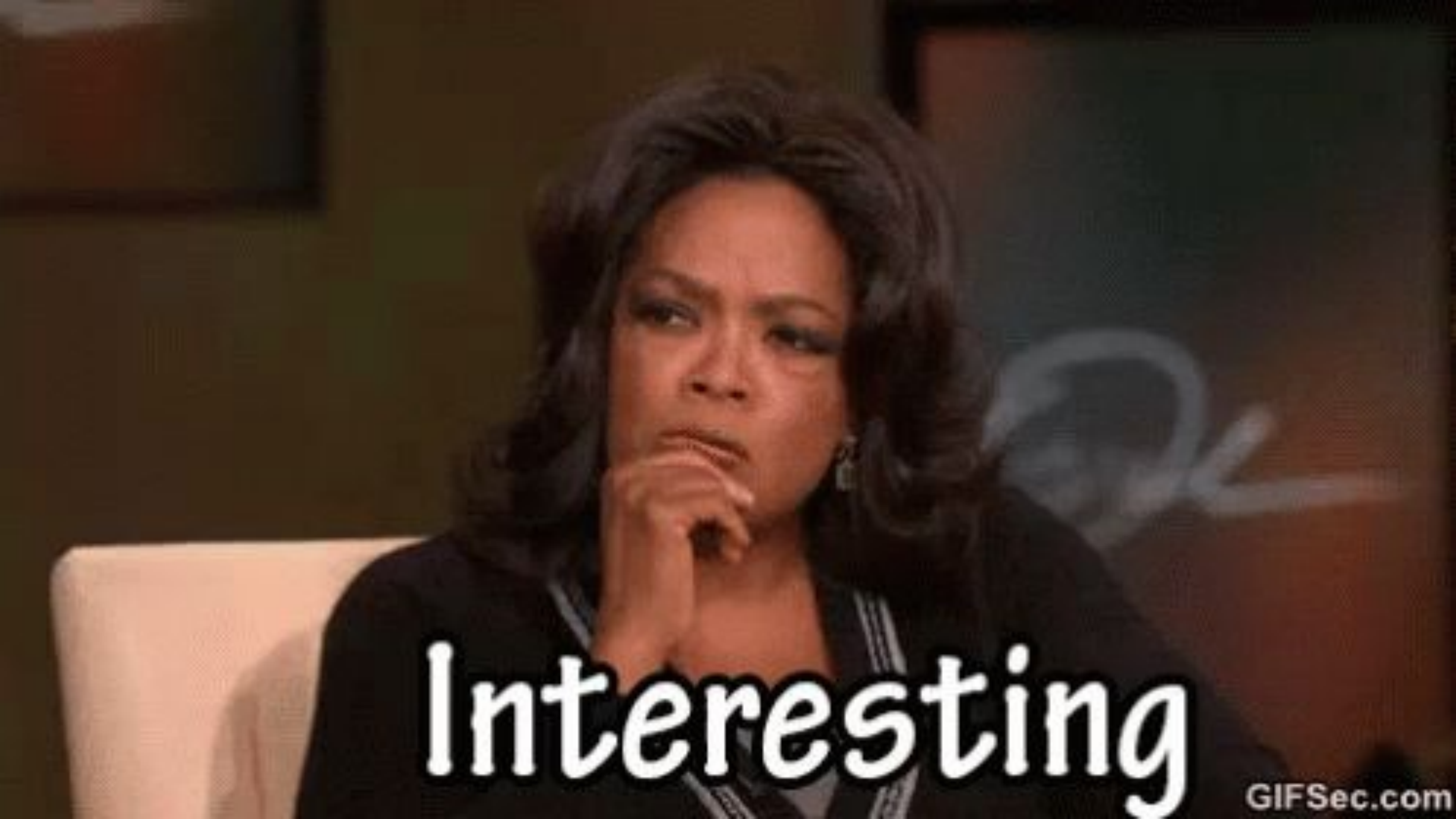
```
// Make the raw clicks stream
var button = document.querySelector('.this');
var clickStream = Rx.Observable.fromEvent(button, 'click');

// HERE
// The 4 lines of code that make the multi-click logic
var multiClickStream = clickStream
    .buffer(function() { return clickStream.throttle(250); })
    .map(function(list) { return list.length; })
    .filter(function(x) { return x === 2; });

multiClickStream.subscribe(function (numclicks) {
    document.querySelector('h2').textContent = ''+numclicks+'x click';
});
```

http://jsfiddle.net/staltz/4gGgs/27/

Example: Autocomplete search box



```
searchTextField.rx.text
    .throttle(0.3, scheduler: MainScheduler.instance)
    .distinctUntilChanged()
    .flatMapLatest { query in
        API.getSearchResults(query)
            .retry(3)
            .startWith([]) // clears results on new search term
            .catchErrorJustReturn([])
    }
    .subscribe(onNext: { results in
      // bind to ui
    })
    .disposed(by: disposeBag)
```


Interesting
GIFSec.com

More operators

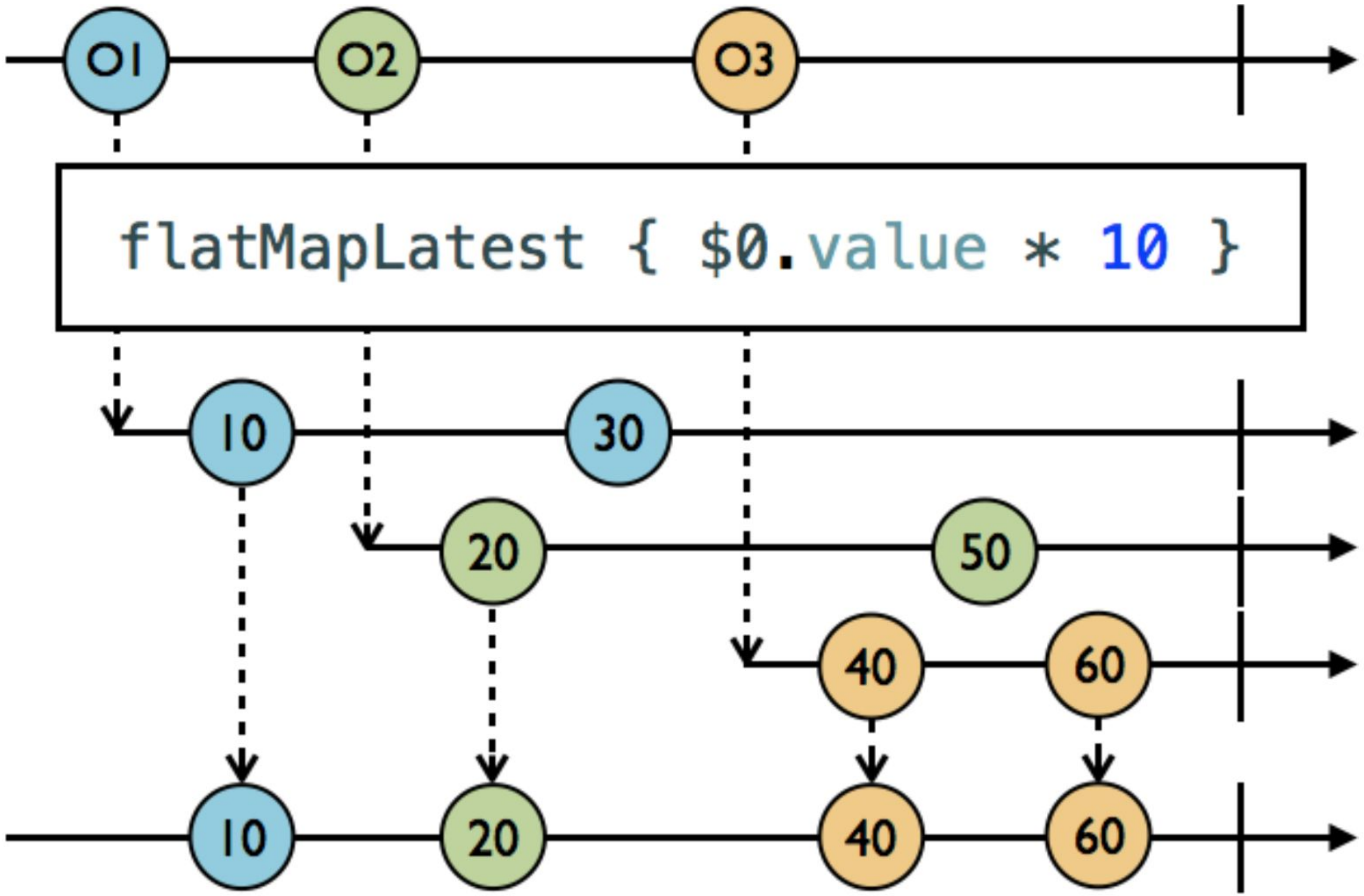
- Zip
- Merge
- Map
- Throttle
- Documentation

FlatMapLatest

*“Projects each element of an observable sequence into a **new sequence** of observable sequences /*

*and then **transforms** an observable sequence of observable sequences into an observable sequence /*

*producing values only from the **most recent** observable sequence.”*


O1
O2
O3
flatMapLatest { $0.value * 10 }
10
30
20
50
40
60
10
20
40
60

Should I learn this?

It makes your life easier

- Retry or throttling -> 1 line of code
- (Multiple) delegation
- Enforces separation of concerns
- Immutability
- Standard mechanism to recover from errors

It *really* makes your life easier

- Decent documentation & support from the community
- Obvious way to compose asynchronous operations
- Implementation in more than 15 programming languages
- Uni-directional data flow
- Fits great with MVVM, MVP, VIPER...

Caveats

- Errors terminate the sequence
- A lot of operators
- Steep learning curve for imperative programmers
- “API for **asynchronous** programming”
- *Reactive Expansion*

Other Topics

- Schedulers
- Disposing Observables
- Hot vs Cold Observable
- Subjects
- Traits

Resources

- Learn the Operators and eat your vegetables
- RxMarbles
- RxSwift -> Playground!
- RxCocoa
- RxSwiftCommunity
- rxswift.slack.com

@JuanjoRamos82

Demo

- Handling errors
- RxSwift playgrounds