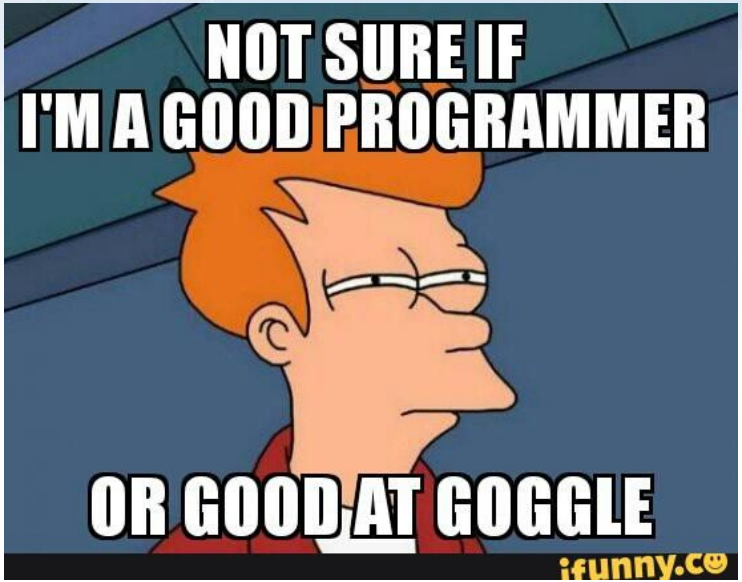
RxJS ALL THINGS!

(Just kidding, please don't do that)

Jeff Beck

About the presenter



What is RxJS?

RxJS is really just a specific implementation of Reactive Extensions or ReactiveX.

ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences.

It extends the observer pattern to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety, concurrent data structures, and non-blocking I/O.

It is sometimes called “functional reactive programming” but this is a misnomer. ReactiveX may be functional, and it may be reactive, but “functional reactive programming” is a different animal. One main point of difference is that functional reactive programming operates on values that change continuously over time, while ReactiveX operates on discrete values that are emitted over time.

TLDR;

- RxJS = Reactive Extensions for JavaScript
- A pattern that allows developers to support sequences of data or events while abstracting away several complexities
- Operates on discrete values that are emitted over time

THIS IS AWESOME!

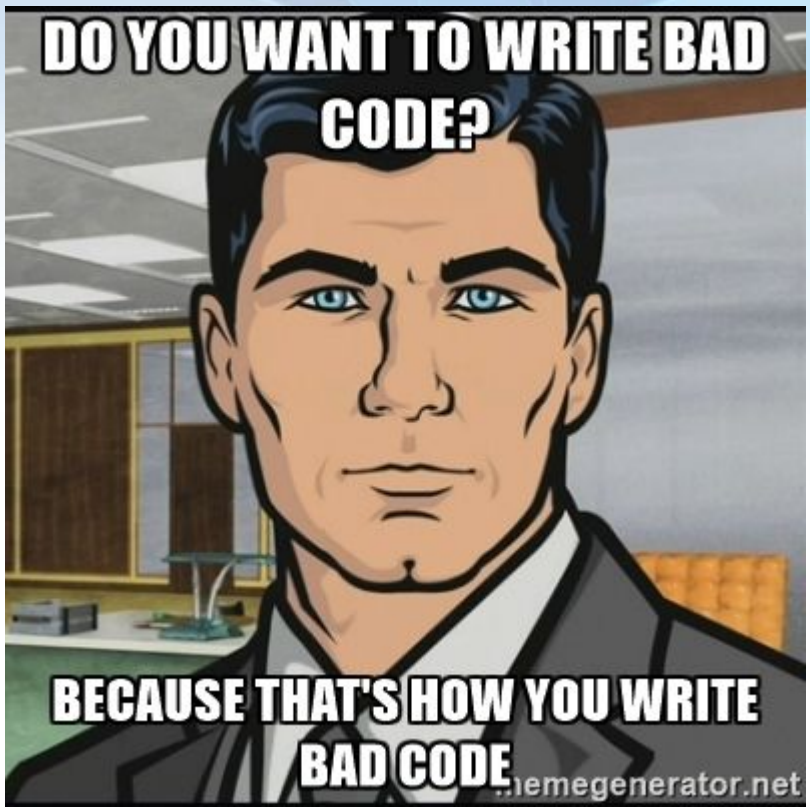
Wait a minute, before we get too excited let's talk about why we shouldn't get too crazy with RxJS.

- Easy to lose track of who or what is listening or emitting to the stream.
- Hot and cold observables.
- Can create an architectural mess.
- Everything… everything is async and needs to be handled from a reactive mindset.

Think twice, code once

Just like anything else in development, think before you type!

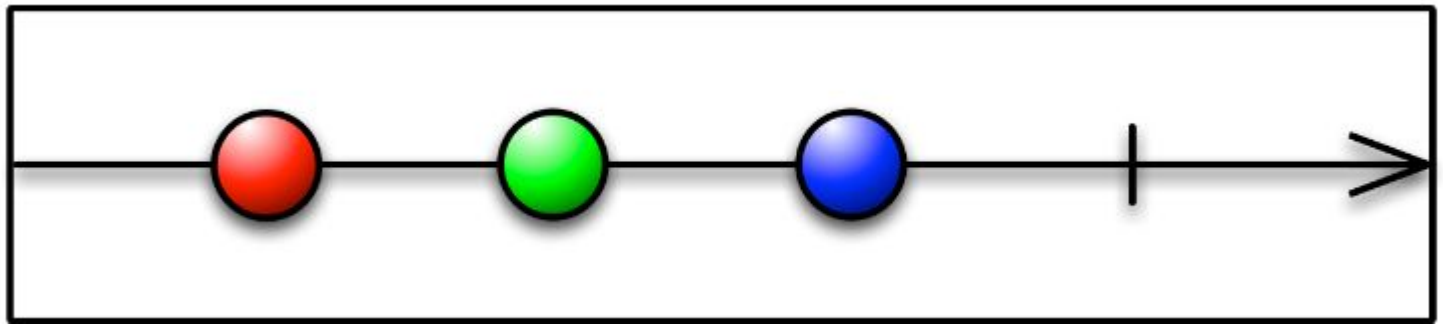
- Don’t pass observables when they are not necessary. Let your structural framework’s change detection handle that.
- Pay close attention to which streams will complete and which will not.
- Place extra caution when architecting. When and where observables are emitted to and subscribed is important. Especially with post or put functionality.
- Don’t always expect data to be there. React to changes to the stream instead.



The Basics

What is an observable?

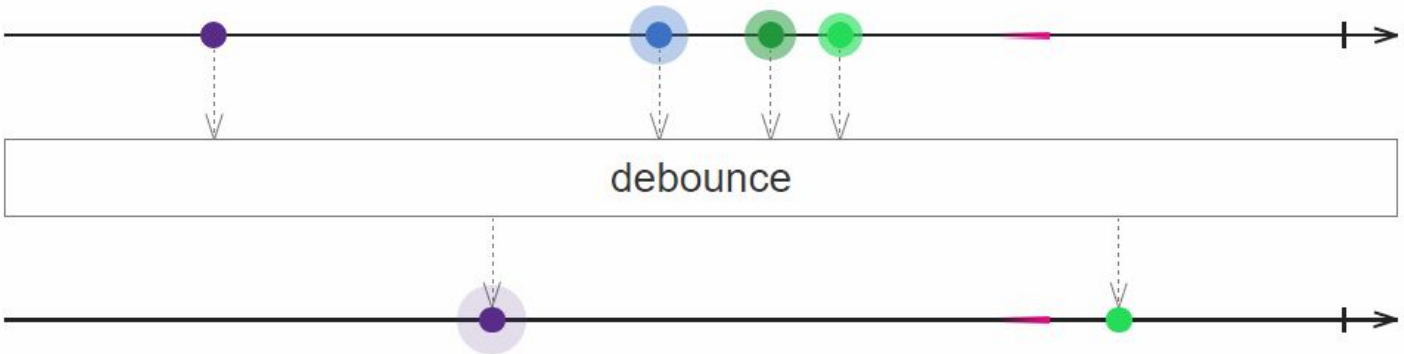
- An observable is a simple stream of asynchronous data.
- It implements a sub set of methods (onNext(), onError(), onComplete()).
- There are two types of observables “Hot” and “Cold”.
  - “Hot” observables begin emitting data as soon as they are created and subscribers may begin observing anywhere in the sequence.
  - “Cold” observables wait until a subscriber subscribes before it begins emitting items. This ensures subscribers receive the entire stream. (ng2 http)
- Observables are just the basics to which we will build upon.



The Basics cont

Ready to Observe?

- You begin to observe items by calling the subscribe method.
- You will often need to use one or more operators to consume the stream.
- An observer can unsubscribe from a observable at any time.



Promise vs Observable

What can a promise do that an observable can't?

NOTHING!

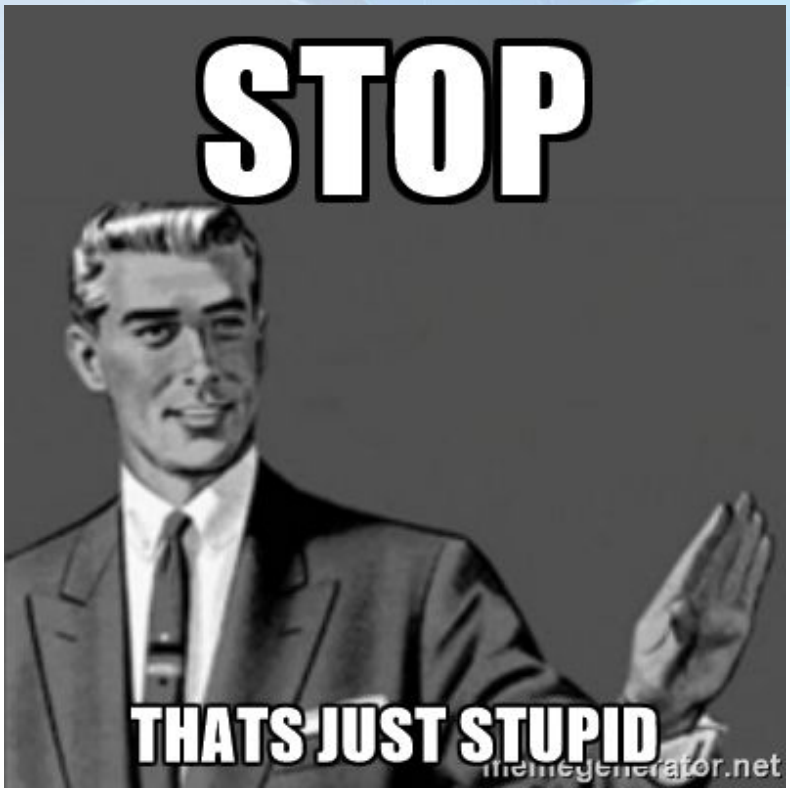
- Observables are lazy.
- Observables are cancelable.
- Observables embody setup and teardown.
- Observables can be retried or reran.
- Promises are done, the logic already ran.
- Several operators that can be leveraged to react to observables.

Let’s get fancy!

So it’s going to hit my endpoint each subscription!?

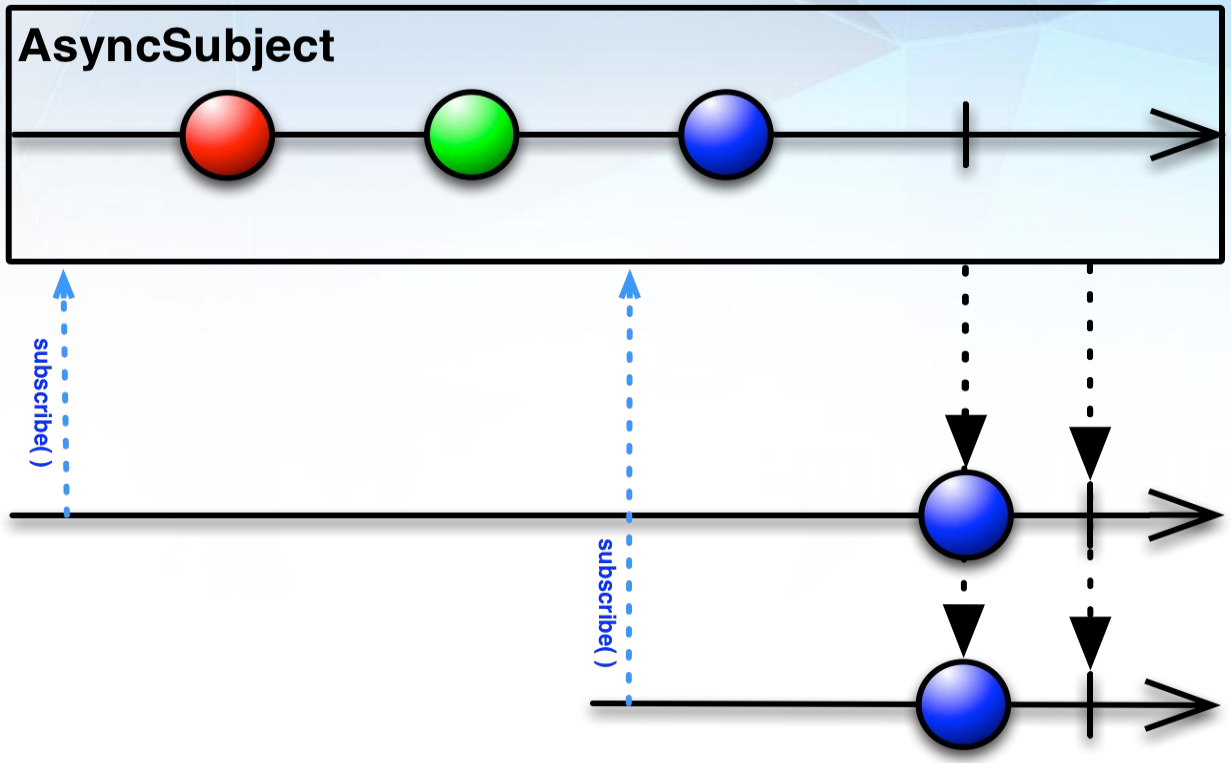
RxJS has several types of **Subjects** for reason like this

- A subject is a bridge or proxy that acts as both observer and an observable.
- They pass past emitted items (sometimes) and current emitted items to the subscriber.
- Subjects can also subscribe to a “cold” observable starting its emission of items. Turning it into a “hot” observable.



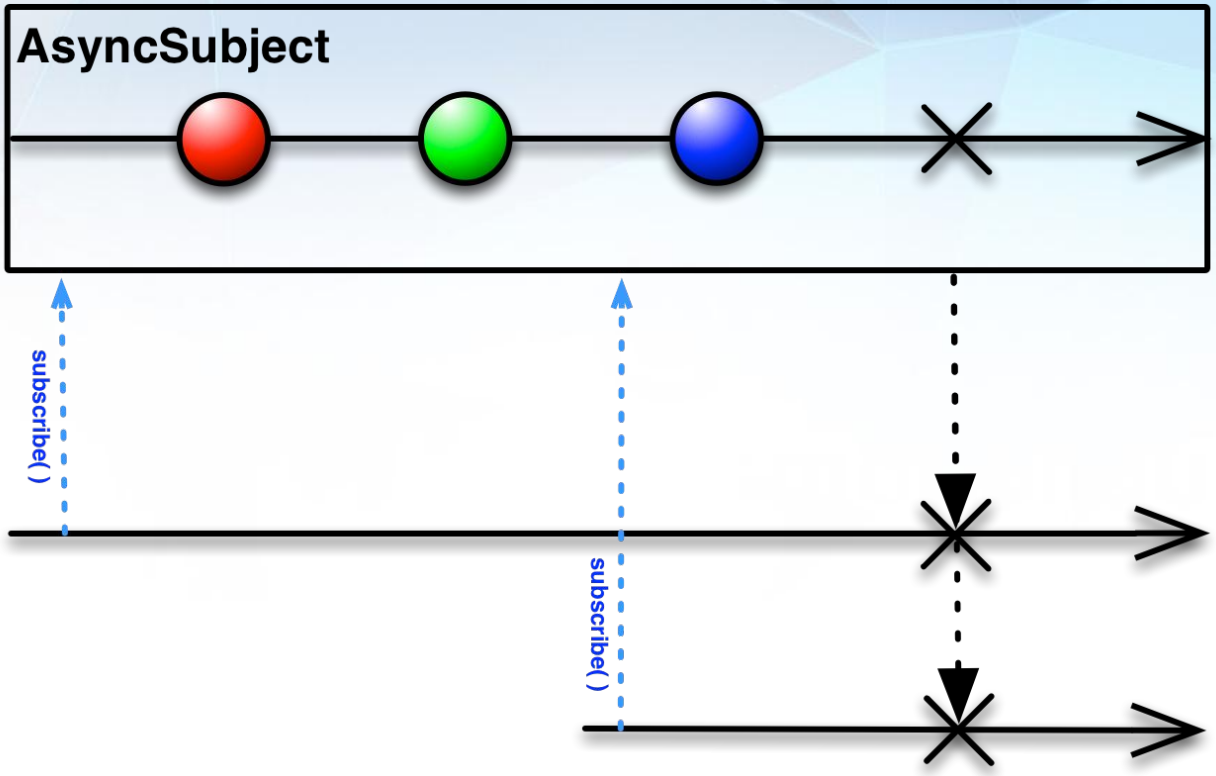
AsyncSubject

An AsyncSubject emits the last value (and only the last value) emitted by the source Observable, and only after that source Observable completes.



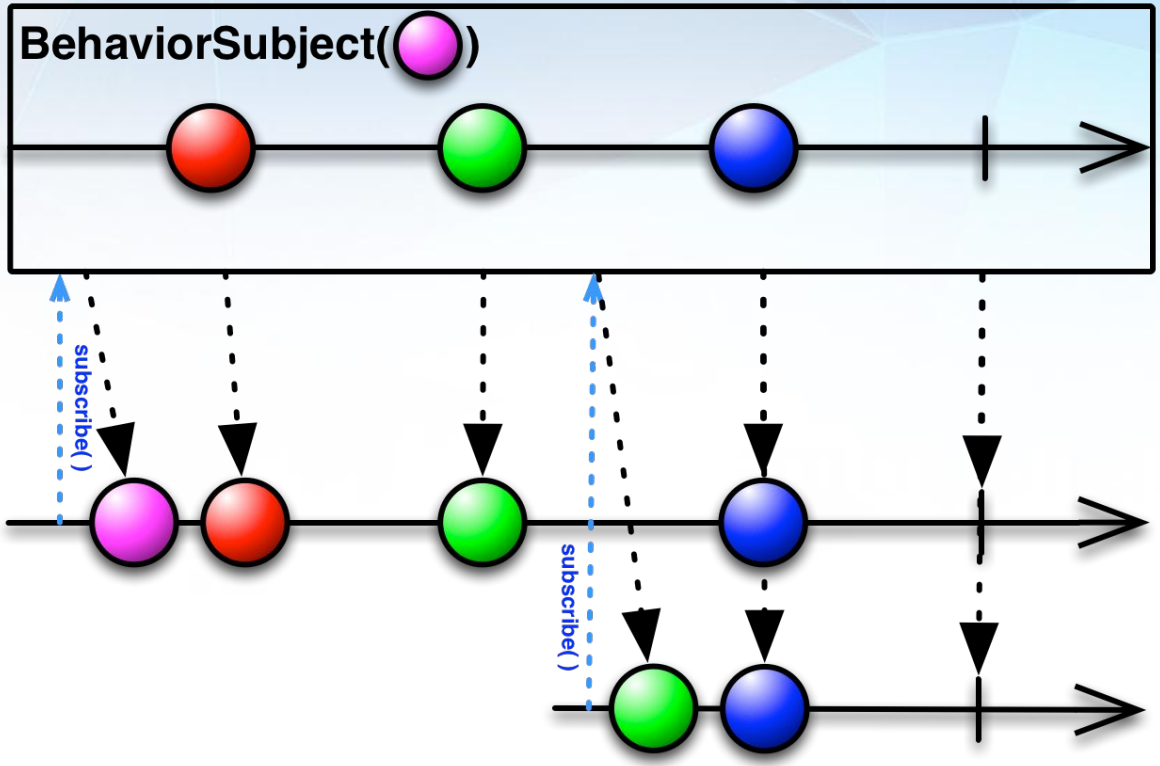
AsyncSubject

If the source Observable terminates with an error, the AsyncSubject will not emit any items, but will simply pass along the error notification from the source Observable.



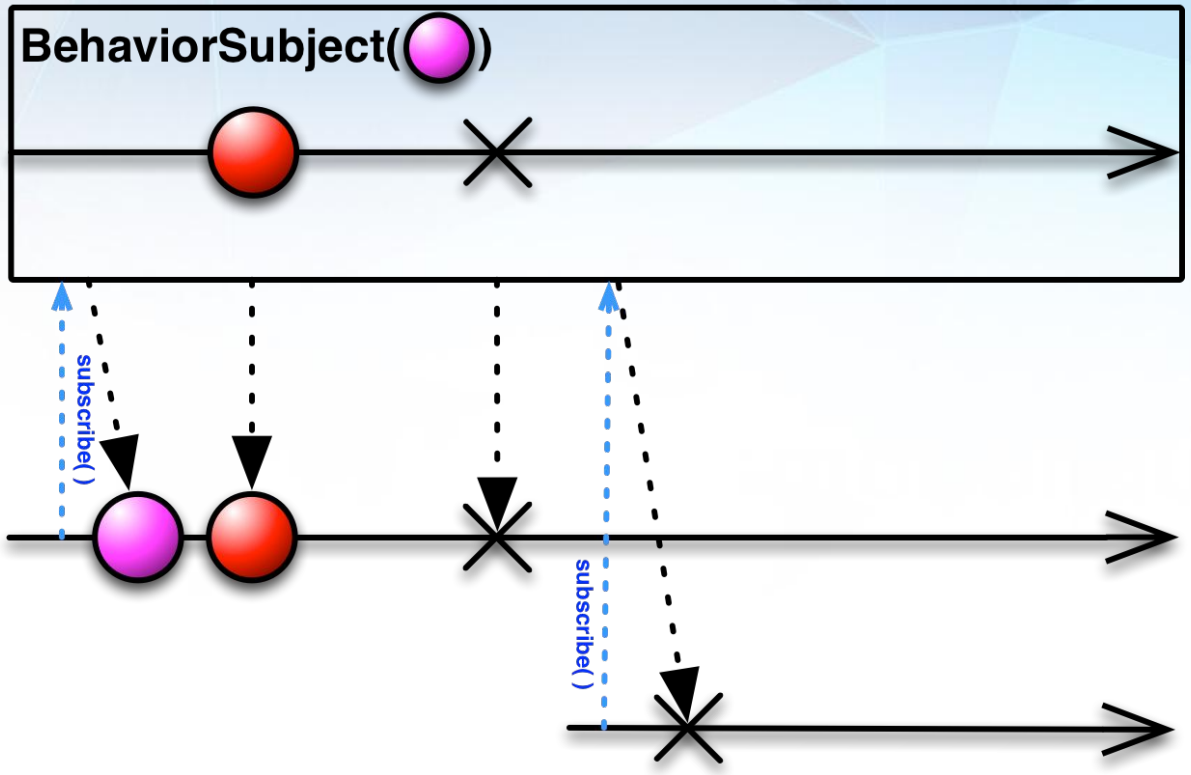
BehaviorSubject

When a BehaviorSubject is subscribed to it will emit the last item it received, or the seed item it was initialized with.



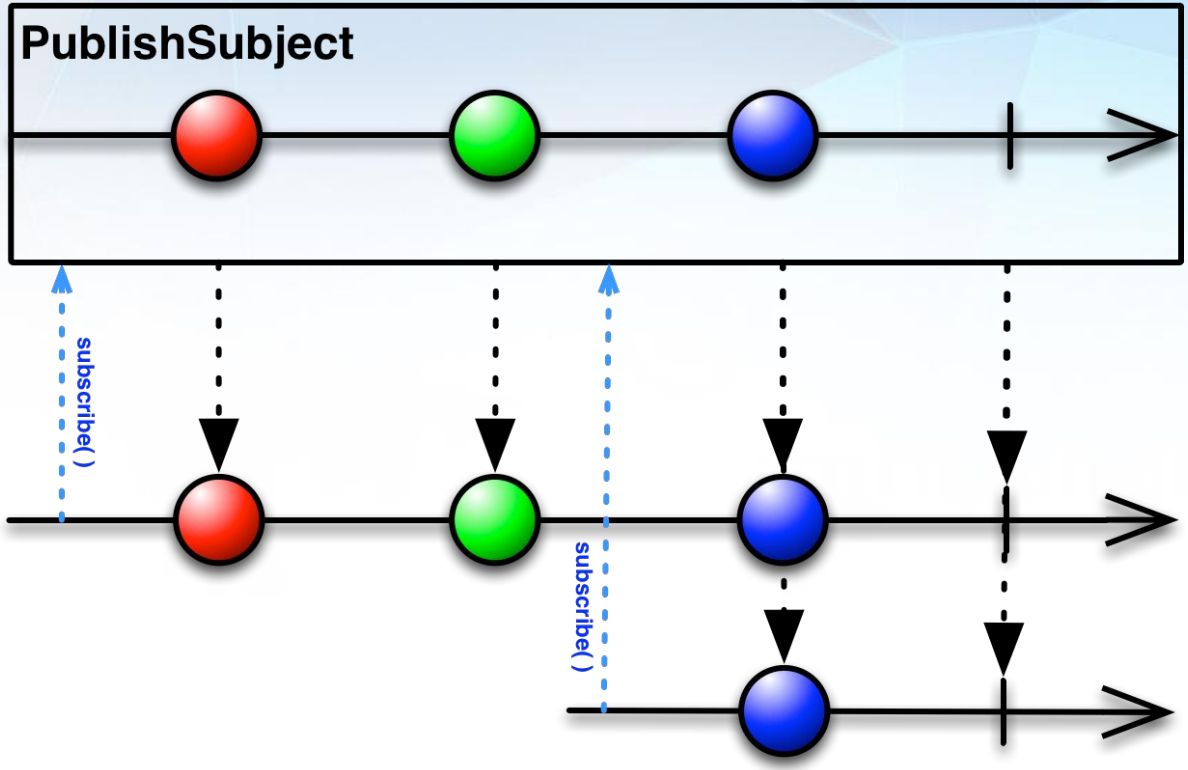
BehaviorSubject

If the source observable terminates with an error, no items will be emitted. It will just pass along the error.



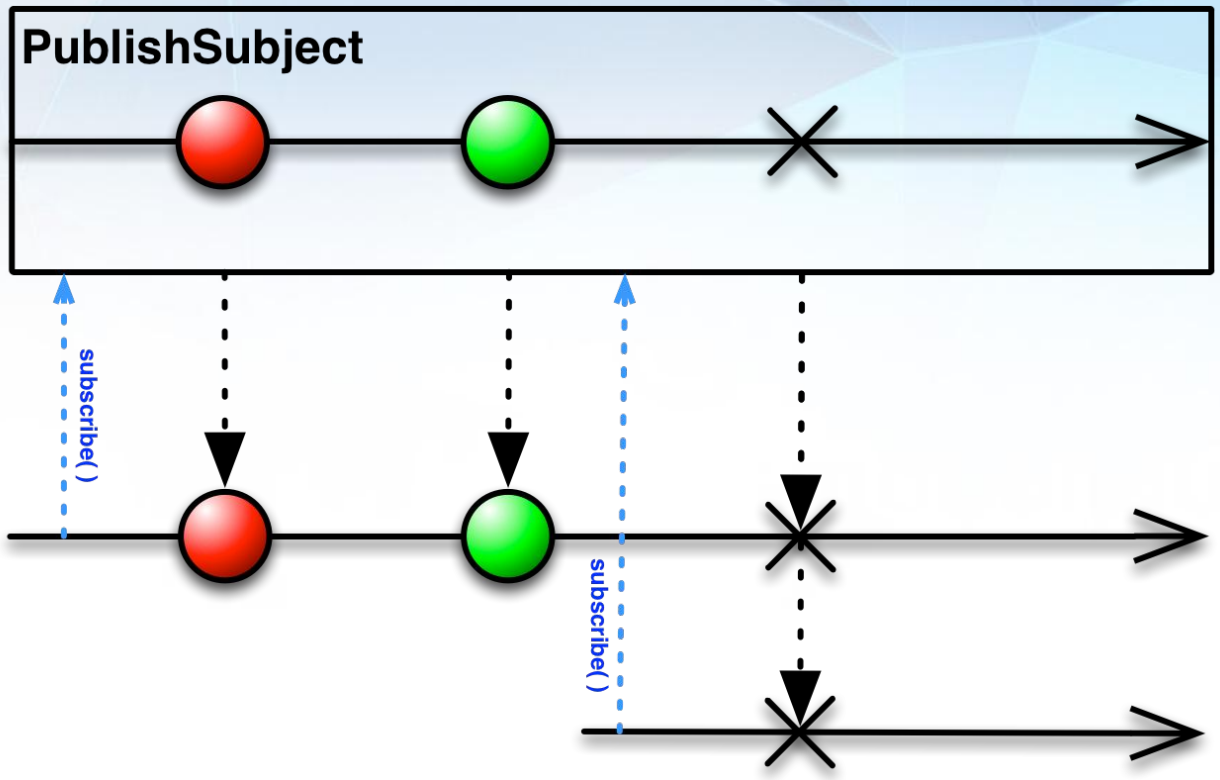
PublishSubject

A PublishSubject emits to the subscriber only those items emitted by the source after the subscriber's subscription.



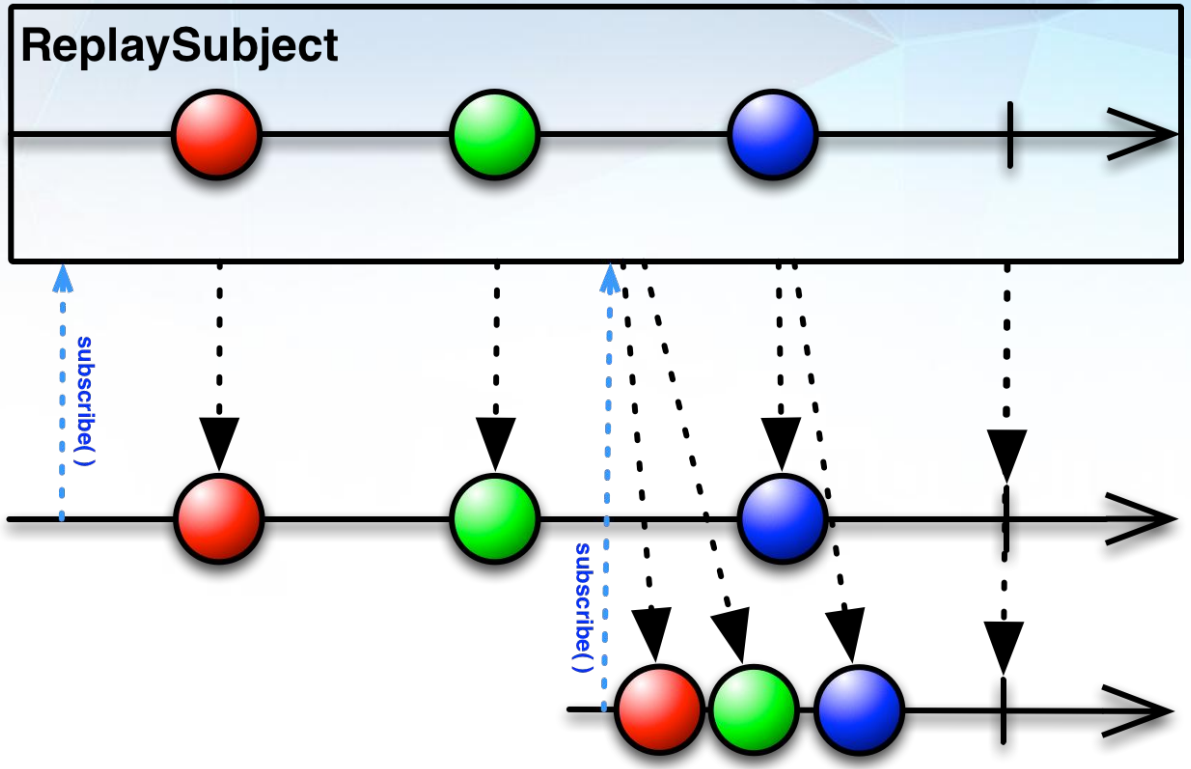
PublishSubject

Since a PublishSubject is data received after the subscriber subscribes it will just pass along the error.



ReplaySubject

A ReplaySubject emits all items received from the sources.



Taming element events

When working with page elements (html inputs and components), you can pass an event to a subject.

Doing this allows you to use RxJS to react to the events using RxJS operators.

Unit testing

```
interface SummaryModelMock extends Subject<any> {
    readOnly$: Subject<boolean>;
    updateStatus: Sinon.SinonSpy;
}

describe('ServiceEventServicePageComponent', (): void => {
    let servicePage: ServiceEventServicePageComponent;
    let serviceModel: Subject<any>;
    let summaryModel: SummaryModelMock;

    beforeEach(() => {
        serviceModel = new Subject();
        summaryModel = (new Subject() as any);
        summaryModel.readOnly$ = new Subject<boolean>();
        summaryModel.updateStatus = sinon.spy();
        const dataServices = new DataServices(new services.dataContracts.ResourceBuilder(<any>{}, <any>{}));

        servicePage = new ServiceEventServicePageComponent(
            <any>serviceModel,
            <any>summaryModel,
            dataServices,
        );

        servicePage.summary = <any>{};
    });
```

Unit testing

```
it('should react to service model changes', () ⇒ {
    // arrange
    const fakeService = { id: 9008, fake: 'service' };

    servicePage.ngOnInit();

    // act
    serviceModel.next(fakeService);

    // assert
    expect(servicePage.service).to.equal(servicePage.service);
});
```

Trivia

What does NPM stand for?

Nothing it is not an acronym.

Thank You

Twitter: @cpt_redbeard81
Gmail: jbeck8176@gmail.com
GitHub: jbeck8176

Special thank you to John Hoover, Brandon Joyce, and my teammates at Renovo Solutions.

Credits

Resources:
http://moduscreate.com/observables-and-promises/
http://reactivex.io/
https://egghead.io/lessons/rxjs-creating-an-observable

https://github.com/jbeck8176/rxjsdemo