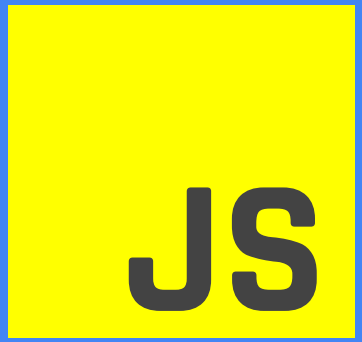
Thinking Reactive with



Lasitha Petthawadu | TL

Yohan Gomez | SSE

Raathigeshan Kugarajan | SSE

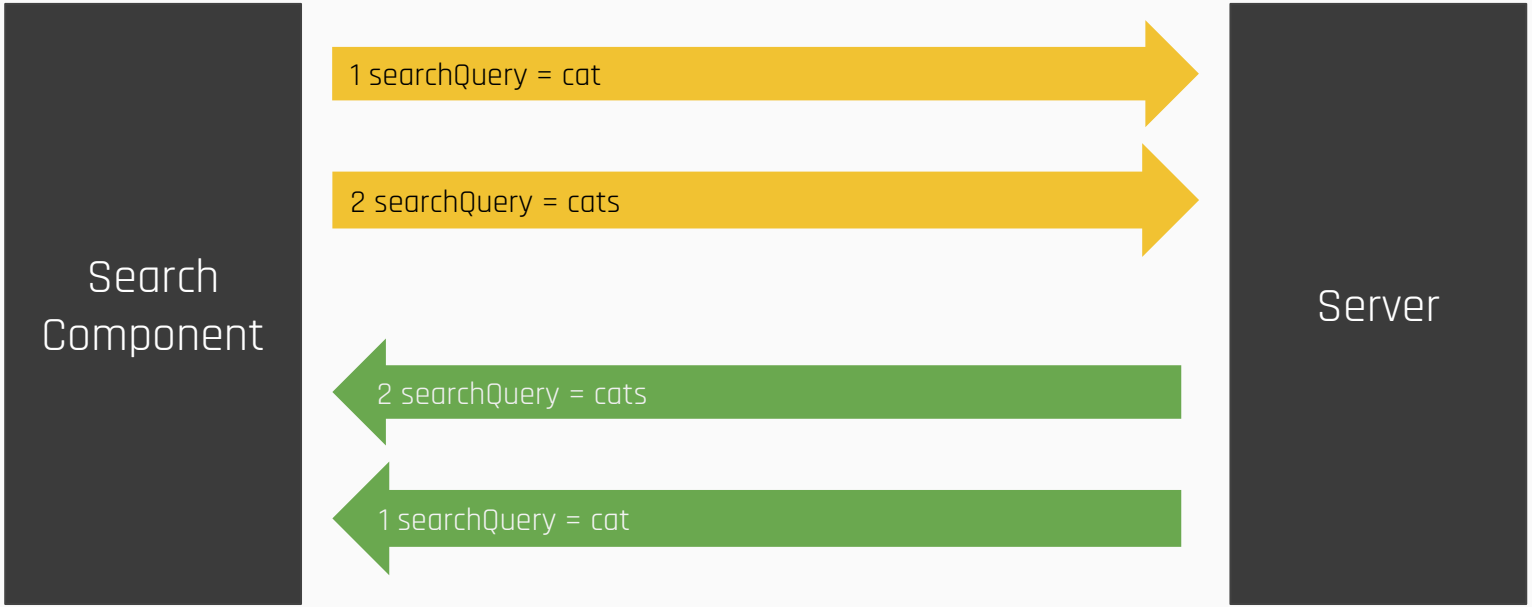
The Ultimate Autocomplete!

- We have two sources to get Movies and TV Series.
- Combine results into one dropdown

Challenges

- Multiple requests in parallel to fetch data
- Return results asynchronously
- Don't listen all key presses like arrow keys
- Limit multiple keypress before sending out request
- Keeping track of event binding is complicated
- Race conditions

Http request order



JQuery Autocomplete

Callback are simple. But...

```
let isMoviesLoaded = false;
let isTvSeriesLoaded = false;
let moviesData = [];
let tvSeriesData = [];

document.getElementById("search").addEventListener("keyup", function(event) {
  let query = event.target.value;
  $.get("http://localhost:3334/movies/" + query, function(movies) {
    isMoviesLoaded = true;
    moviesData = movies;
    if (isTvSeriesLoaded) {
      let results = tvSeriesData.concat(moviesData);
      renderResults(results);
    }
  });

  $.get("http://localhost:3334/tv/" + query, function(series) {
    isTvSeriesLoader = true;
    tvSeriesData = series;
    if (isMoviesLoaded) {
      let results = tvSeriesData.concat(moviesData);
      renderResults(results);
    }
  });
});
```

Fundamental mechanism for handling async in JavaScript

http://jsbin.com/qebunufuxi/1/edit?js,output

Callbacks leads to callback hell

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

The Promise Way

```
document.getElementById("search").addEventListener("keyup", function(event) {
  var query = event.target.value;
  Promise.all([
      $.get("http://localhost:3334/movies/" + query),
      $.get("http://localhost:3334/tv/" + query)
    ])
    .then(function(data) {
      var movies = data[0].data;
      var tv = data[1].data;

      var items = movies.concat(tv);
      console.log(items);
    });
});
```

http://jsbin.com/kuwarulibe/edit?html,js,console,output

Characteristics of Promise

- Guaranteed future
- Immutable
- Single Value
- Caching

But, Solution we have is not perfect...

- It overloads the server with a bunch requests as user types
- Http request order would cause problems (Race conditions)
- Global state is required to handle race conditions
- Leads to memory leaks with event handlers

Introducing Reactive Programming

Pull Based

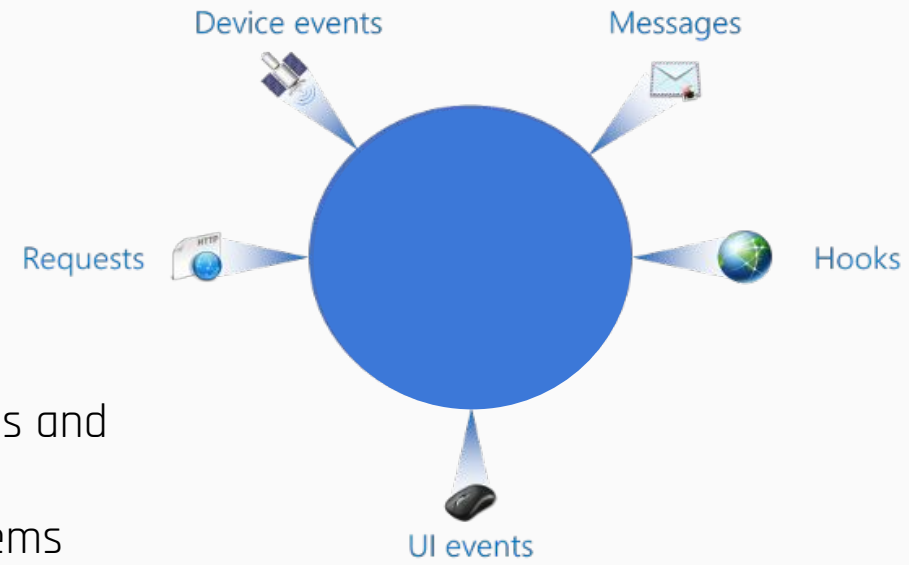
```
function print() {
    // do something here
}

print();
print();
```

Push Based

```
function print() {
    // do something here
}

document.getElementById("myBtn")
.addEventListener("click", print);
```

Data over time

- Mouse clicks
- Key presses in a input box
- Scroll event
- Data through a web socket
- Combining these data sources and coming up with solutions for complex asynchronous problems



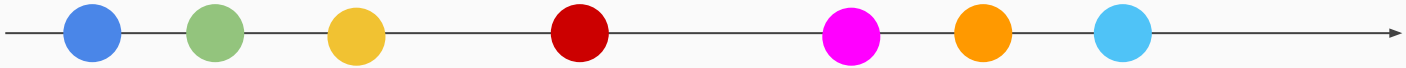
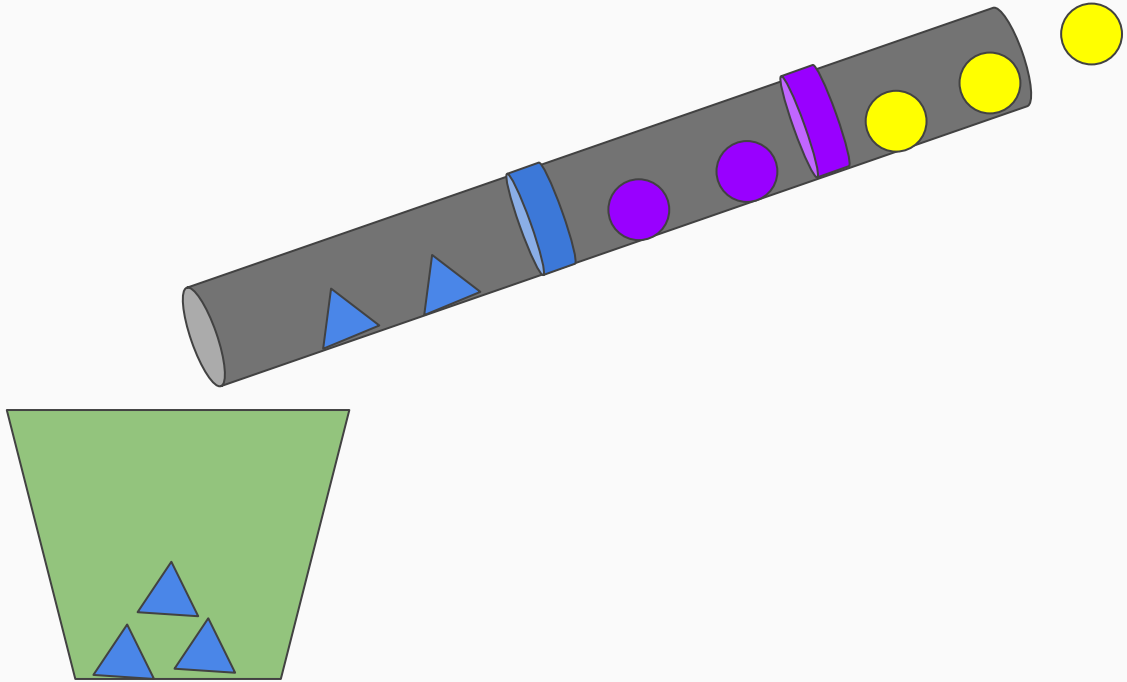
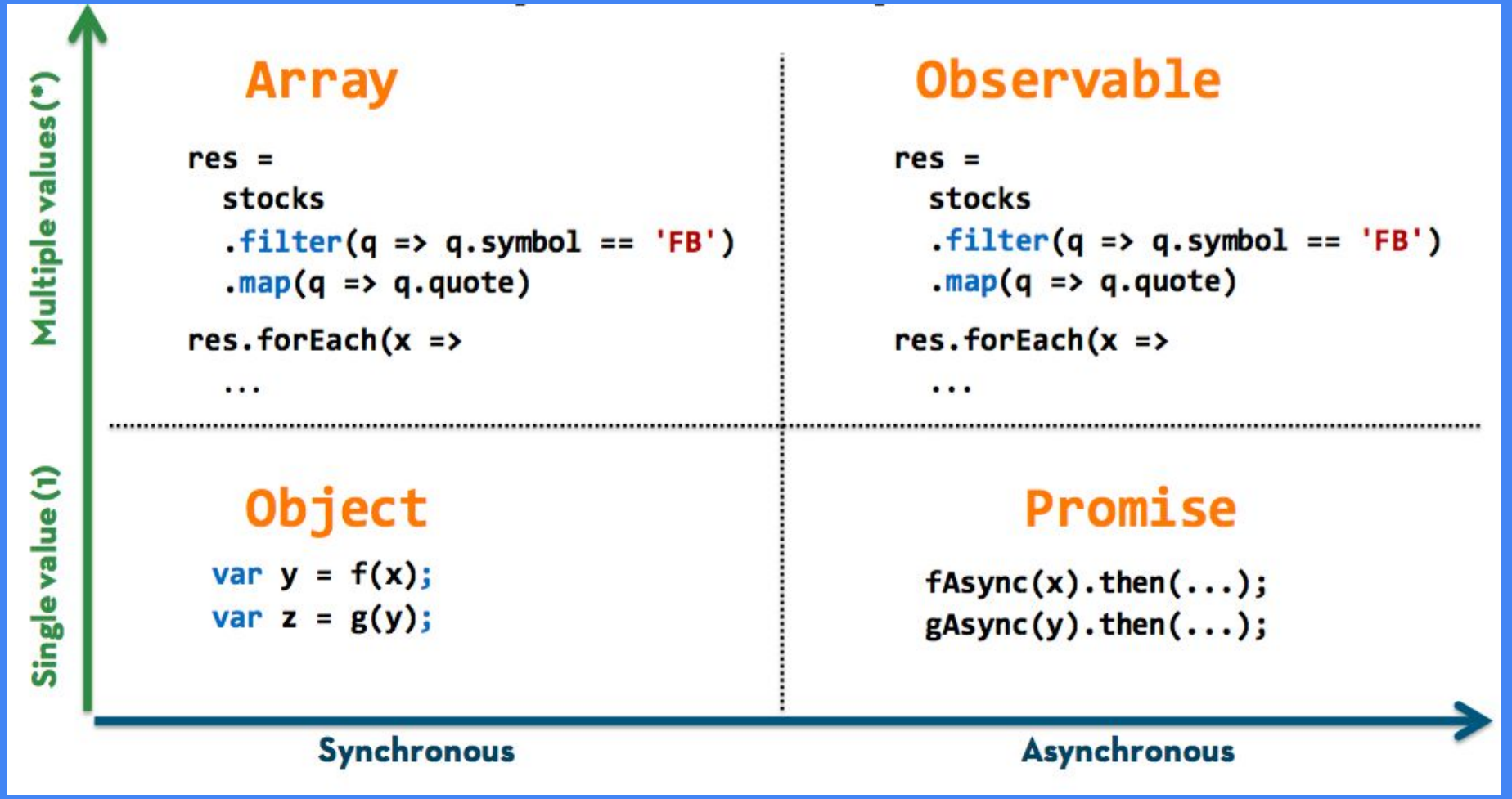


Image Credit: https://www.sitepoint.com/functional-reactive-programming-rxjs/

Observables

1. Collection of zero, one or more values
2. Over any amount of time (Finite and infinite stream)
3. Two types of observables
    a. Finite set of data - An Array of values
    b. Infinite set of data - Mouse move values



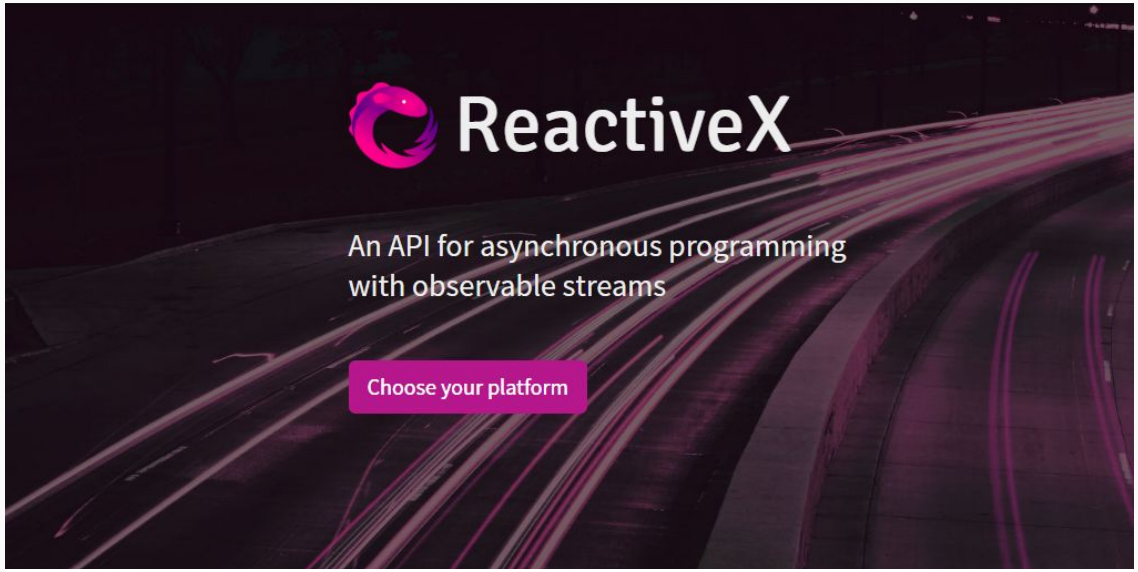

Multiple values (*)
Single value (1)
Array
res =
stocks
.filter(q => q.symbol == 'FB')
.map(q => q.quote)
res.forEach(x =>
...
Observable
res =
stocks
.filter(q => q.symbol == 'FB')
.map(q => q.quote)
res.forEach(x =>
...
Object
var y = f(x);
var z = g(y);
Promise
fAsync(x).then(...);
gAsync(y).then(...);
Synchronous
Asynchronous


Everything is a stream

Reactive Programming Libraries for JavaScript

Introduction to ReactiveX

Rx is a library for composing asynchronous and event-based programs using observable collections.



Ubiquitous

ReactiveX is everywhere, and it's meant for everything.

FRONTEND

Manipulate UI events and API responses, on the Web with RxJS, or on mobile with Rx.NET and RxJava

CROSS-PLATFORM

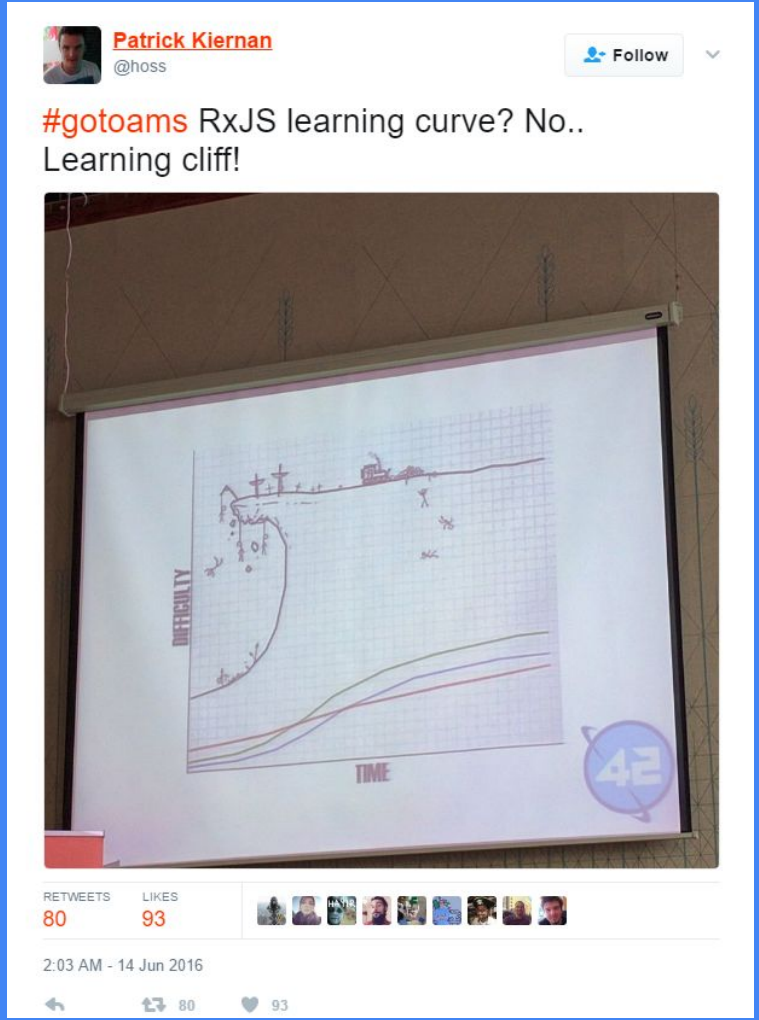
Available for idiomatic Java, Scala, C#, C++, Clojure, JavaScript, Python, Groovy, JRuby, and others

BACKEND

Embrace ReactiveX's asynchronicity, enabling concurrency and implementation independence

RX JS is the Javascript library from ReactiveX Project

RxJS
Learning Cliff



Creating an observables from an array in RxJS

```
var observable = Rx.Observable.from([1,2,3]);

observable.subscribe(
  function(value) {
    console.log(value);
  },
  function(err) {},
  function() {}
);
```

http://jsbin.com/cacozitoze/edit?js,console

RxJS provides helper functions to create observables

Creating an observable from click event

```
var clicks = Rx.Observable.fromEvent(document, 'click');
clicks.subscribe(x => console.log(x));
```

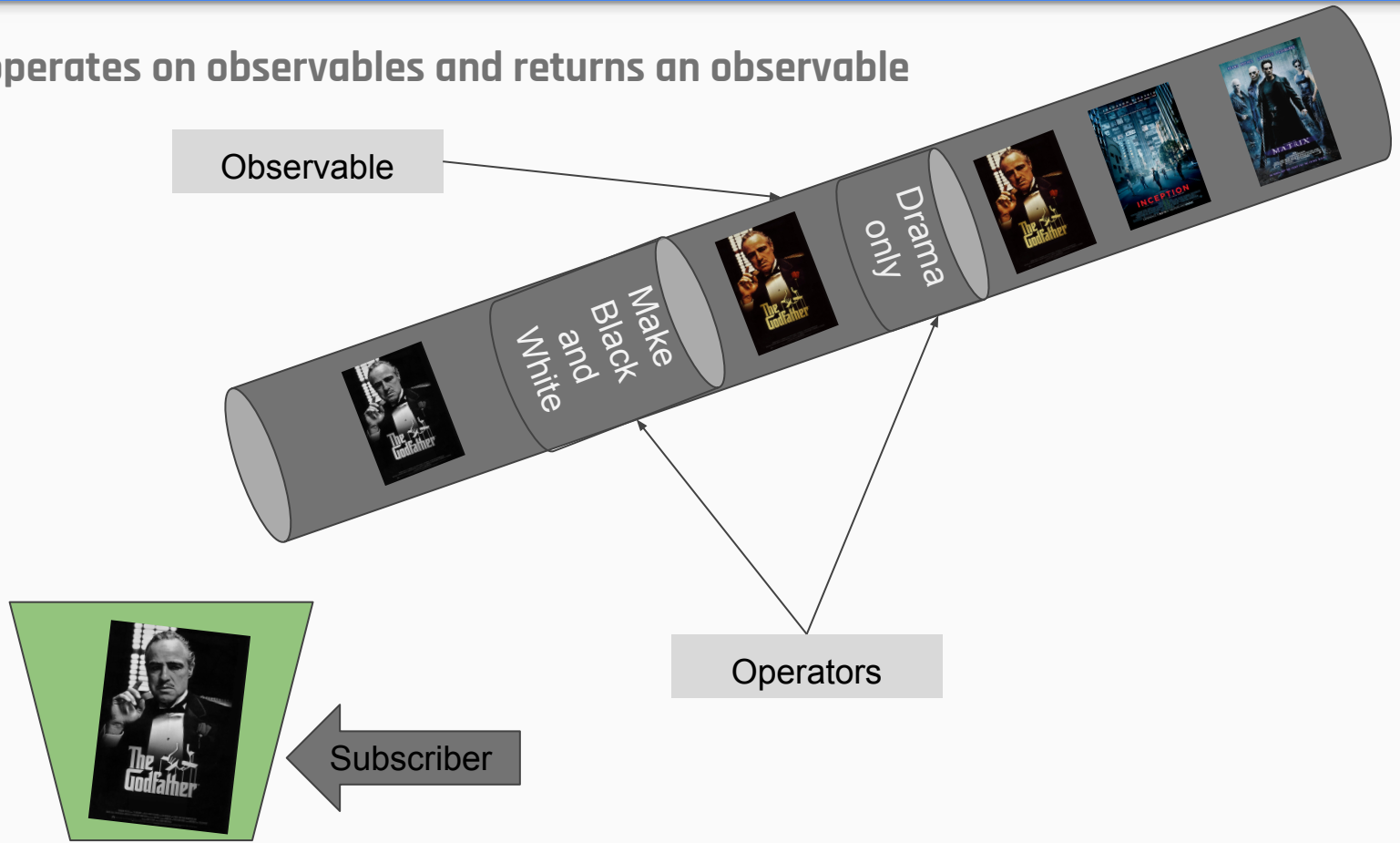
Making an http call into an observable

```
var data = Rx.Observable.fromPromise($.get("http://localhost:3334/user"));
data.subscribe((result) => {
  console.log(result);
});
```

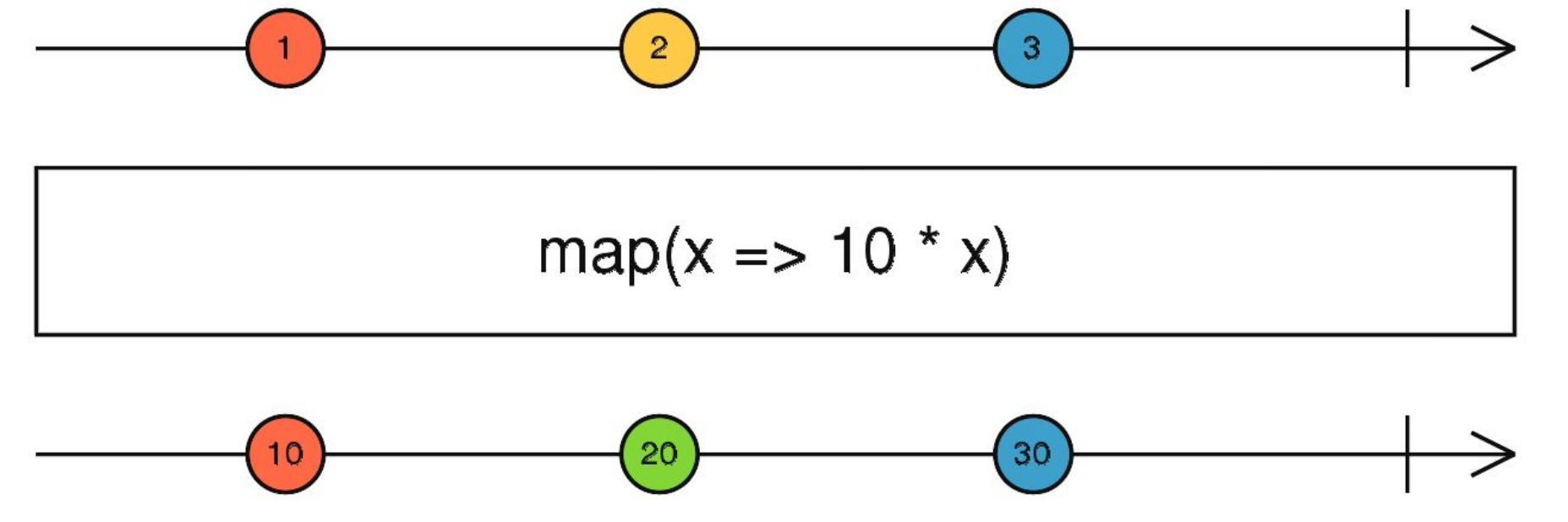
http://jsbin.com/burapuyaya/edit?js,console,output

What are operators in RxJS?

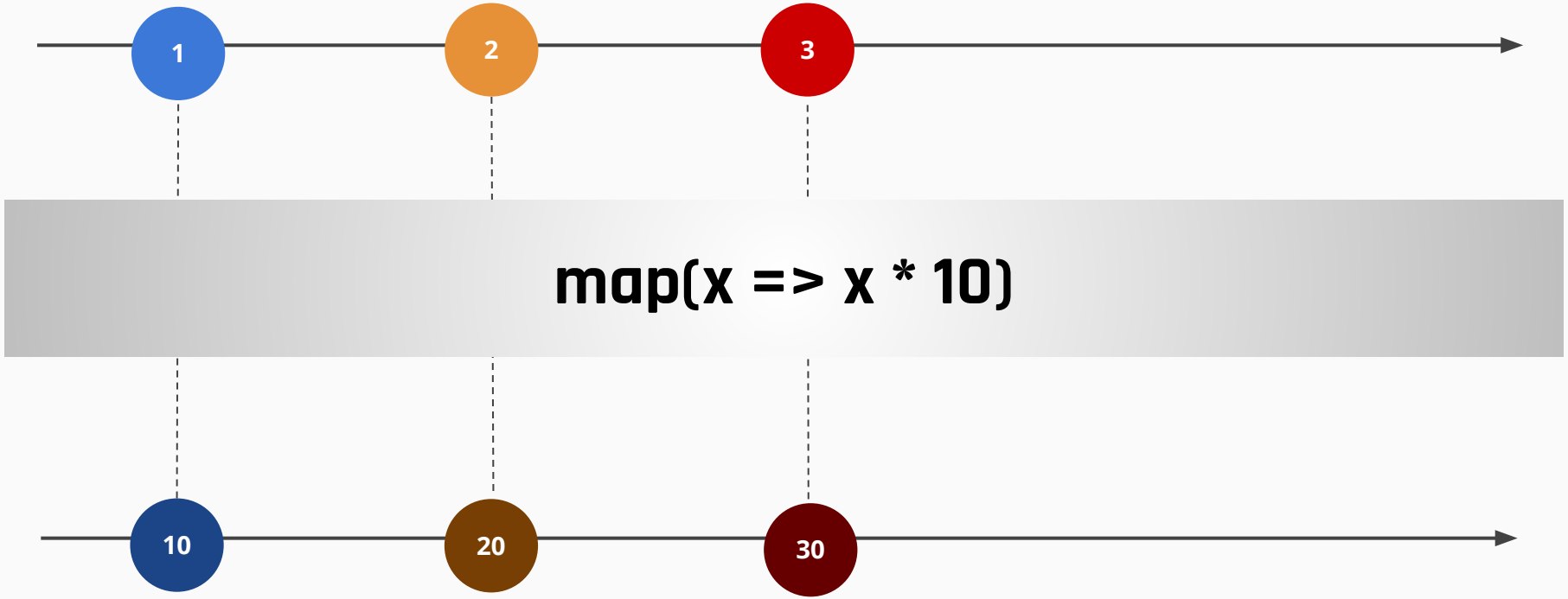
Operators operates on observables and returns an observable



Representing observables in diagram



Map Operator



http://jsbin.com/coxinonufo/1/edit?js,console

Observables in Observable (Higher order observables)

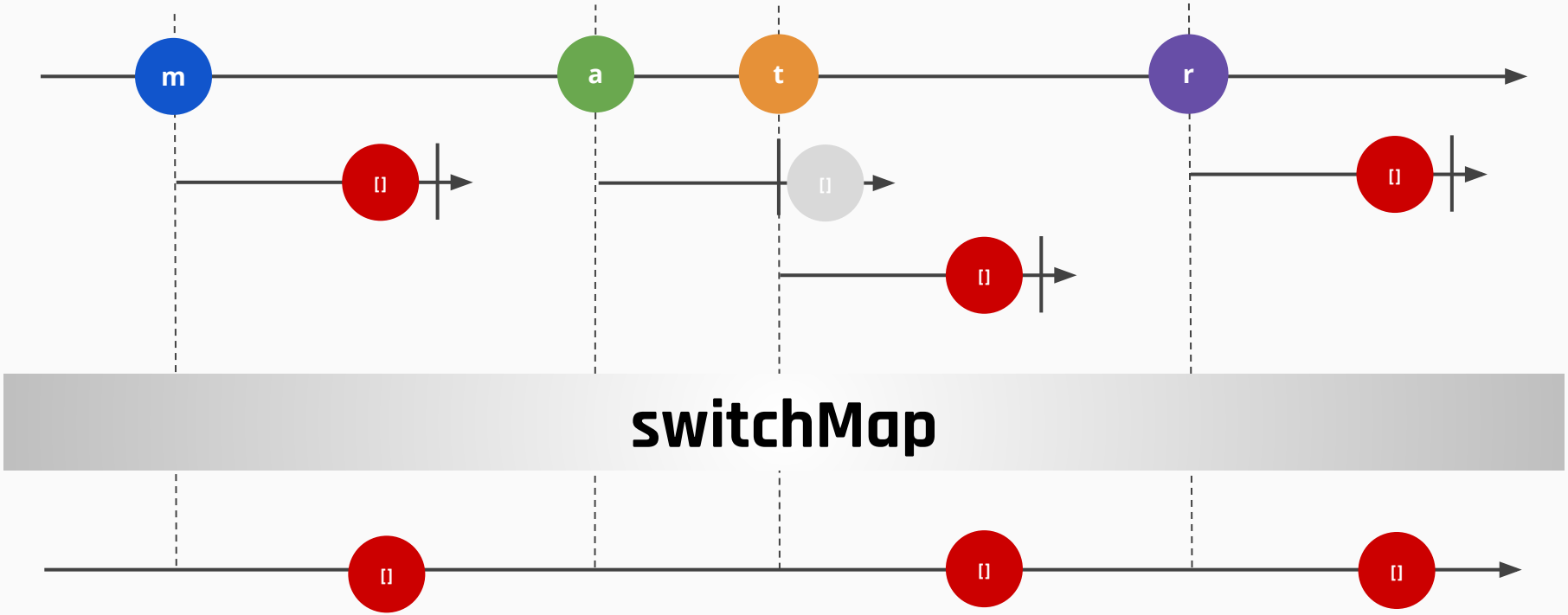
```
var observable = Rx.Observable.from([1,2,3]).map(function(x){
    return Rx.Observable.fromPromise($.get("http://localhost/item/" + x));
});


observable.subscribe(
  function(value) {
  }
);
```

http://jsbin.com/peyavuhojo/edit?js,console

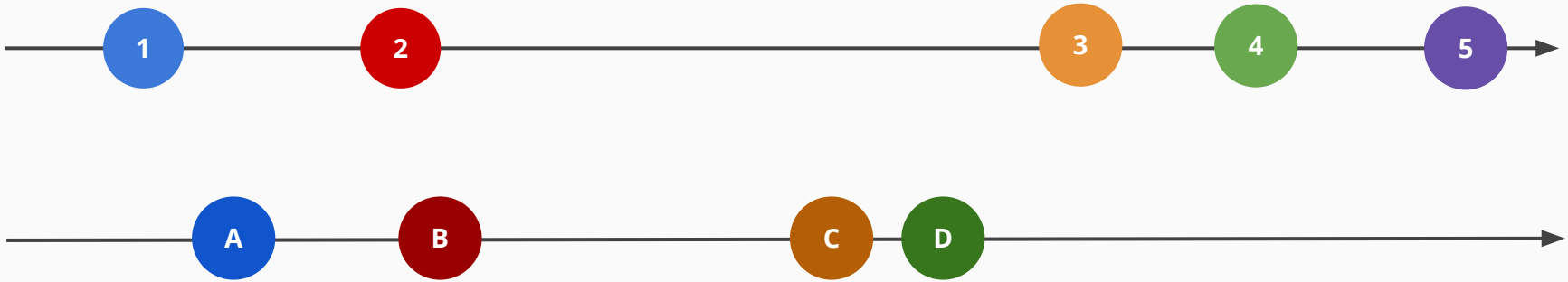
http://jsbin.com/fehojerofa/edit?js,output

Let's do this the Rxjs way

SwitchMap

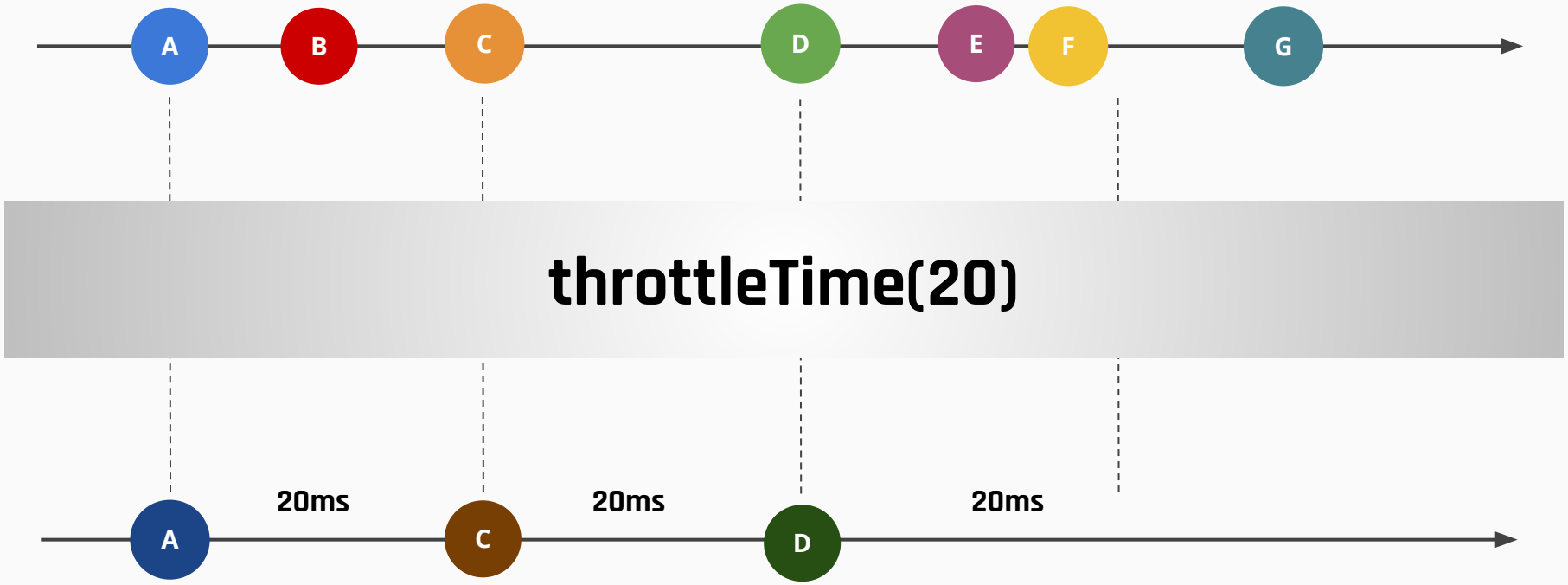


Zip Operator



zip(observable...)

Throttle Time Operator



+120 Operators!

Better Codebases

Functional

Avoid intricate stateful programs, using clean input/output functions over observable streams.

Less is more

ReactiveX's operators often reduce what was once an elaborate challenge into a few lines of code.

Async error handling

Traditional try/catch is powerless for errors in asynchronous computations, but ReactiveX is equiped with proper mechanisms for handling errors.

Concurrency made easy

Observables and Schedulers in ReactiveX allow the programmer to abstract away low-level threading, synchronization, and concurrency issues.

Who uses RxJS

You can write an entire app with RxJS, but

- Use it when you really need to handle complicated async use cases
- Understanding observables and RxJS might be hard and it could impact the new team members.
- Promises are still very much capable for most async scenarios, so don't bring in extra complexity without the actual need. YAGNI!

Learn what you need for now, keep the rest for later

Q&A