



**OR:  
HOW I LEARNED TO STOP WORRYING ABOUT  
ASYNCHRONOUS PROGRAMMING AND LOVE THE  
OBSERVABLE**







**trapd in Monad tutorl  
plz help**





A long-haired brown cat is sitting on a concrete ledge next to a stream. The cat is looking down at the water. The stream is surrounded by green grass and reeds. The water is calm and reflects the sky. The background shows a grassy bank with some trees.

**stream prosesing**

[Download](#)[Docs](#)[Blog](#)[Community](#)[Modules](#)[Resources](#)[Jobs](#)[About](#)[Fork Node](#)

Node.js is a platform built on [Chrome's JavaScript runtime](#) for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Current Version: v0.10.27

**INSTALL**

[Documentation](#)

[API Reference](#)



# WHATWG Streams

## Streams

[soon to become a] **Living Standard** — Last Updated 1 May 2014

### Participate:

Send feedback to [whatwg@whatwg.org](mailto:whatwg@whatwg.org) ([archives](#)) or [file a bug](#) ([open bugs](#))

[IRC: #whatwg on Freenode](#)

### Version History:

<https://github.com/whatwg/streams/commits>

### Editor:

[Domenic Denicola](#) <[domenic@domenicdenicola.com](mailto:domenic@domenicdenicola.com)>



*To the extent possible under law, the editor has waived all copyright and related or neighboring rights to this work.*





# Google Dart

## Introduced Streams as part of dart:async

### Unifies Binary Data

```
Stream<List<int>> stream = new File('quotes.txt').openRead();  
stream.transform(UTF8.decoder).listen(print);
```

### And Events

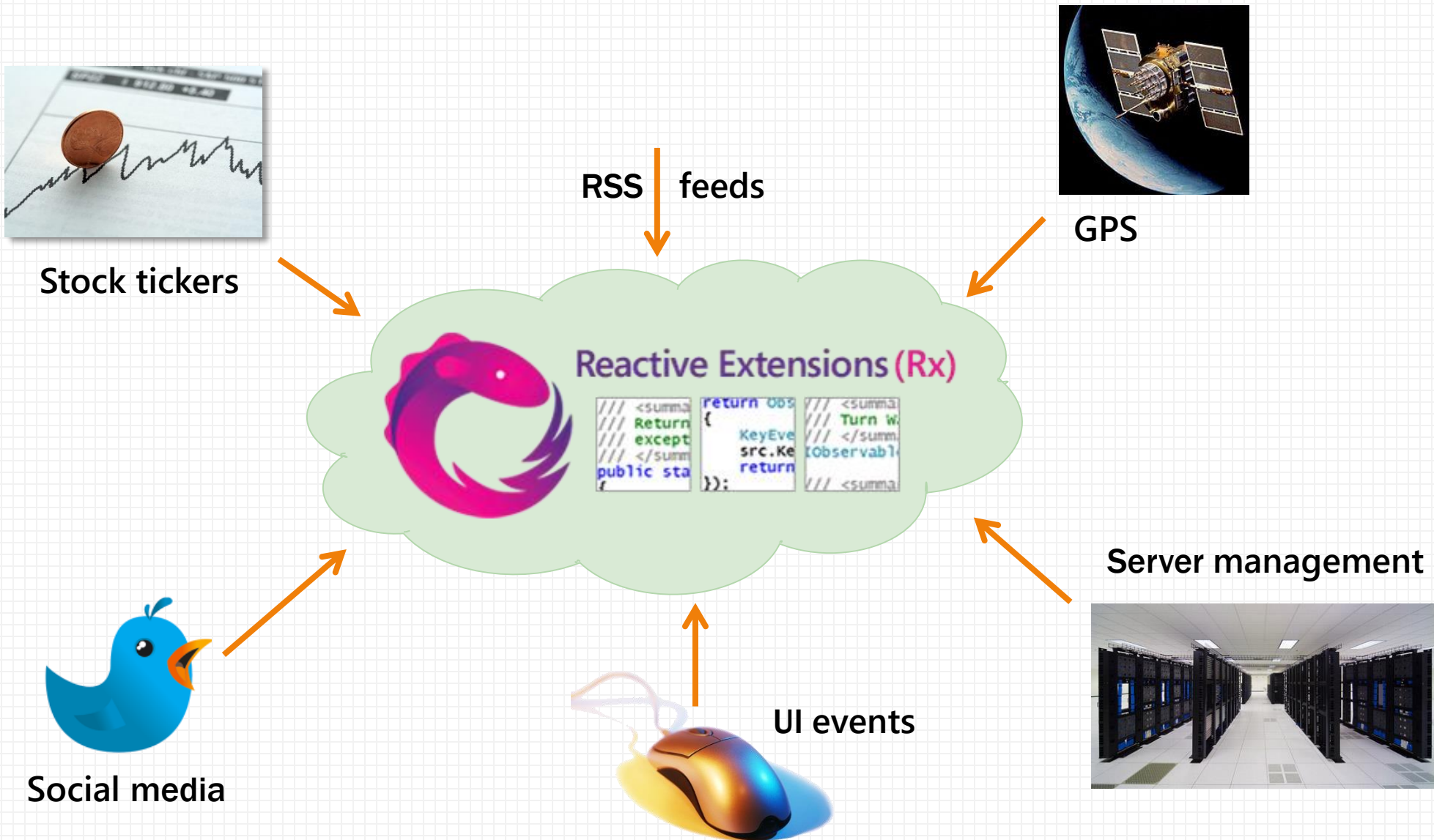
```
querySelector( '#myButton' )  
  .onClick.listen((_) => print( 'Click.' ));
```

<http://news.dartlang.org/2012/11/introducing-new-streams-api.html>





# Real-time is everywhere...



# Top-rated Movies Collection

```
var getTopRatedFilms = function (user) {  
  return user.videoLists  
    .map(function (videoList) {  
      return videoList.videos  
        .filter(function (v) { return v.rating === 5; });  
    })  
    .mergeAll();  
  
getTopRatedFilms(user)  
  .subscribe(function (film) { ... });
```





# What if I told you...

...that you could create a drag event...

...with the almost the *same code*?

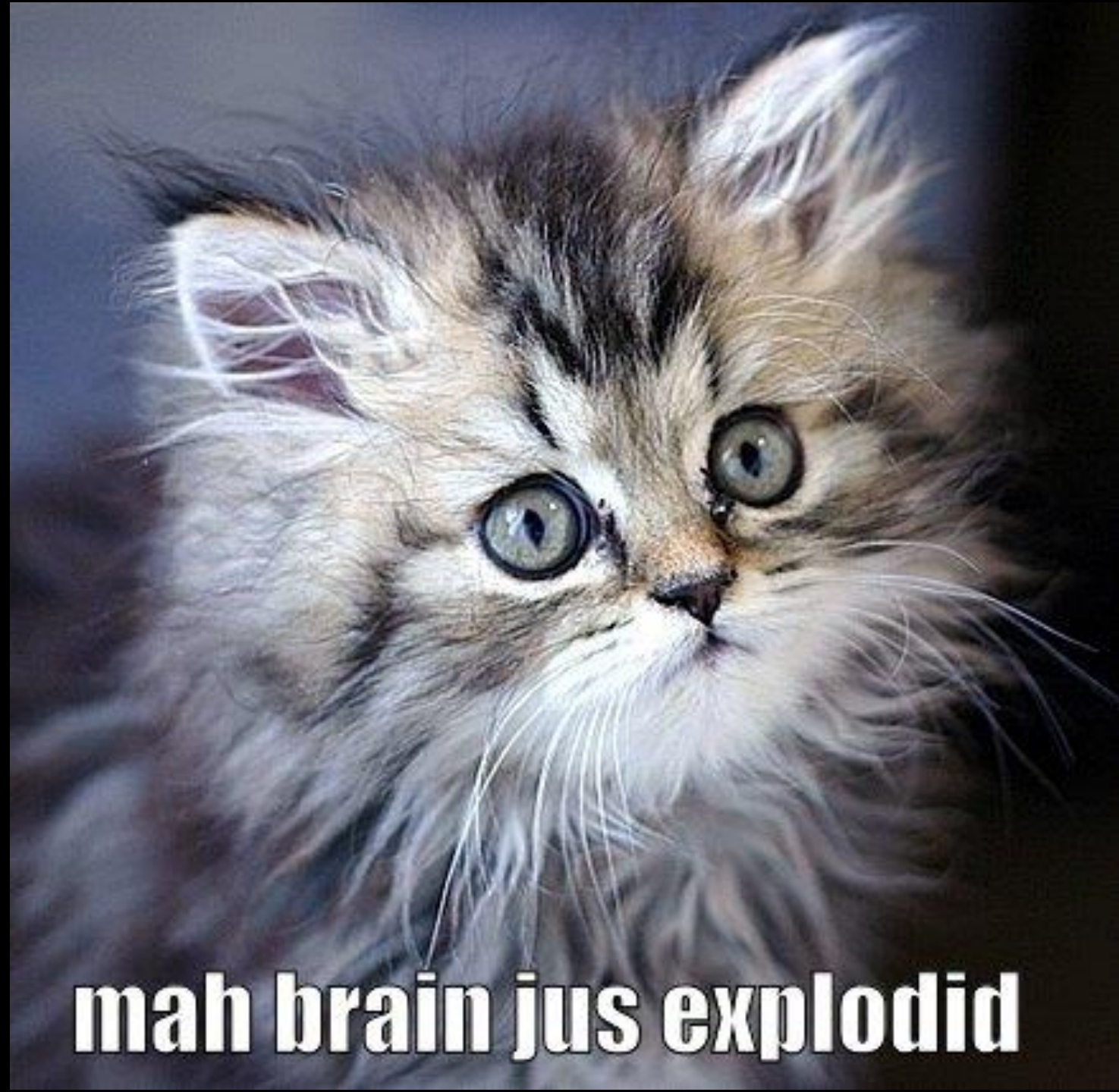




# Mouse Drags Collection

```
var getElementDrags = function (elmt) {  
  return elmt.mouseDowns()  
    .map(function (mouseDown) {  
      return mainWindow.mouseMoves()  
        filter .takeUntil(elmt.mouseUps());  
    })  
    .mergeAll();  
  
getElementDrags(image)  
  .subscribe(updateImagePosition);
```





**mah brain jus explodid**

# Callback Hell

```
function play(movieId, callback) {  
  var movieTicket, playError,  
      tryFinish = function () {  
    if (playError) {  
      callback(null, playError);  
    } else if (movieTicket && player.initialized) {  
      callback(null, ticket);  
    }  
  };  
  if (!player.initialized) {  
    player.init(function (error) {  
      playError = error;  
      tryFinish();  
    })  
  }  
  authorizeMovie(function (error, ticket) {  
    playError = error;  
    movieTicket = ticket;  
    tryFinish();  
  });  
});
```






# Asynchronous Programming is Annoying

**Each framework has its own way of expressing async/event-based programming**

- Node.js has callbacks, then we have Promises, and then events
- Each concept covers only part of the story

**Wouldn't it be great to have a unifying concept to generalize how we think about concurrent/reactive programming?**





# **OnNext:**

## **Reactive Applications Demo**

# Ordinary Interactive Programming

```
try {  
    for (var item in collection)  
        doSomething(item);  
} catch (e) {  
    handleOrThrow(e);  
}  
  
doCleanup();
```

← `OnNext(T)`

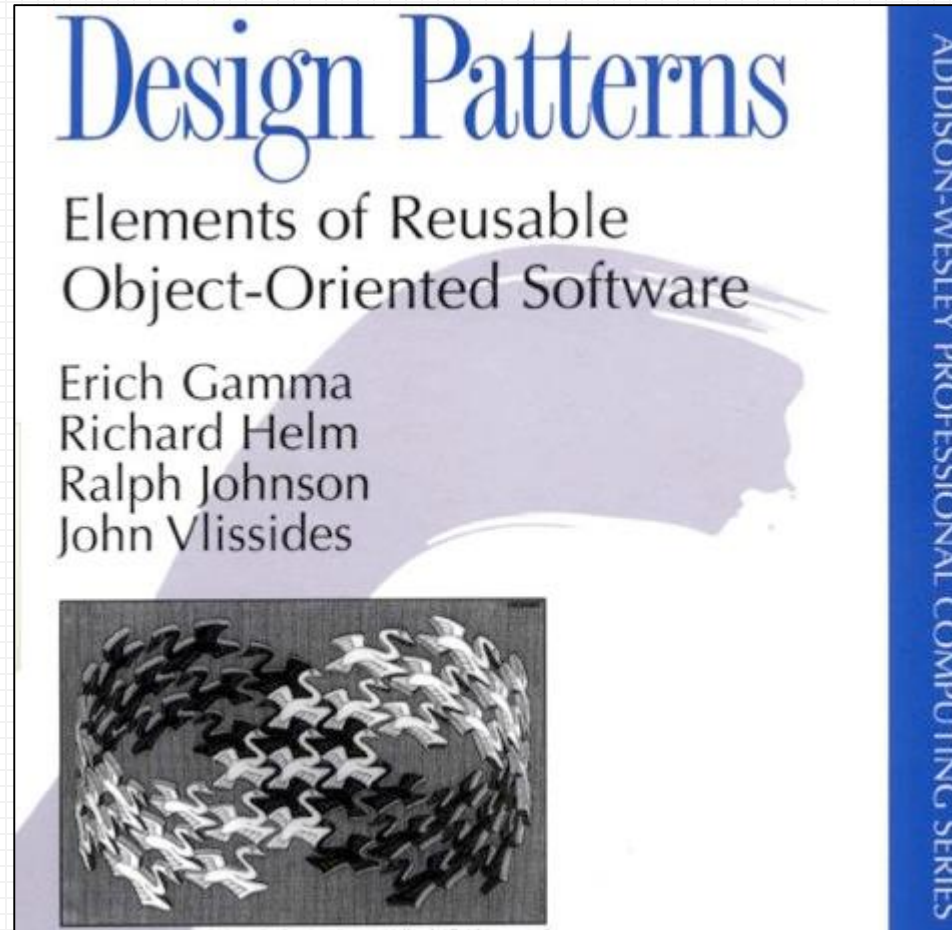
← `OnError()`

← `OnCompleted()`





# That was the iterator pattern



# Making it push-based

```
var collection = Observable.fromEvent(e, 'click');
```

```
var obs = Observer.create(  
    onNext:      x  => doSomething(x),  
    onError:     e  => handleError(e),  
    onCompleted: () => doCleanup());
```

```
var subscription = collection.subscribe(obs);
```

```
// deterministically cleans up all resources  
subscription.dispose();
```

# Rx Grammar Police

OnNext ● \*

Zero or more values

E.g. events are  $\infty$  sequences

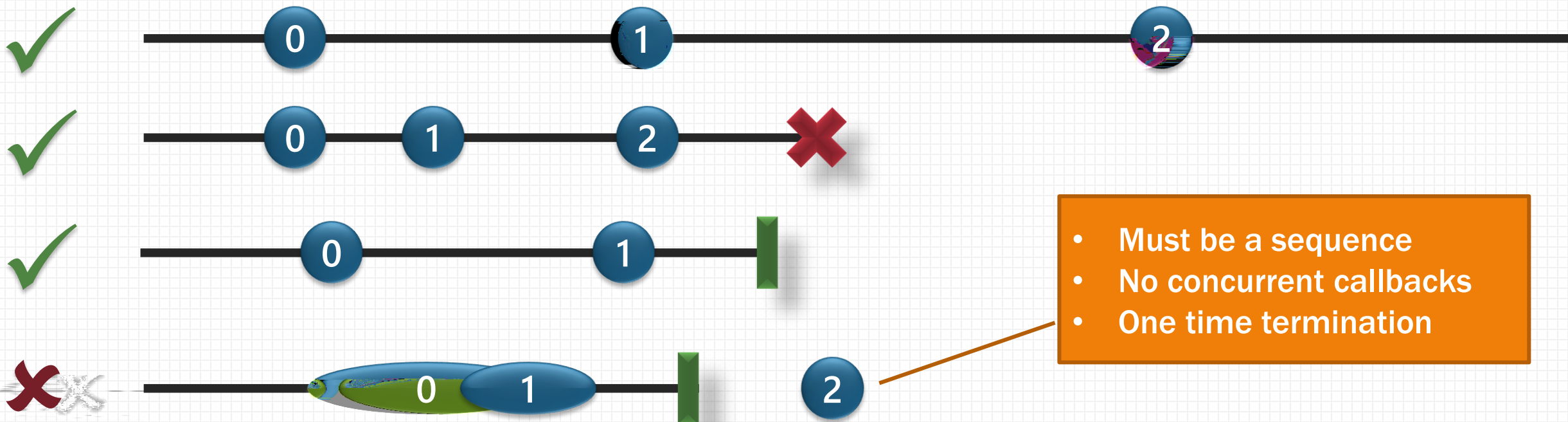
(OnError ✖

Calls can fail

OnCompleted █ ) ?

Resource management

Sequencing





# First-Class Events

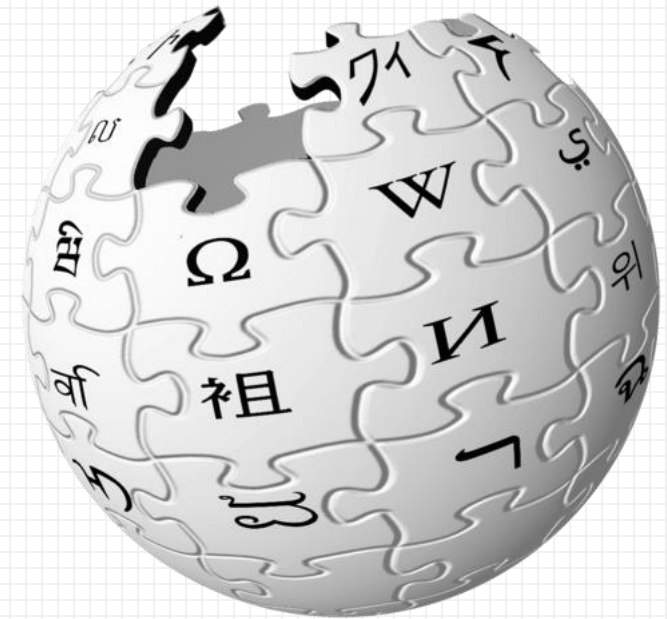
## Objects to the rescue

An object is **first-class** when it:<sup>[4][5]</sup>

- can be stored in variables and data structures
- can be passed as a parameter to a subroutine
- can be returned as the result of a subroutine
- can be constructed at runtime
- has intrinsic identity (independent of any given name)

How about a query library?

Or mocking for testing...?



**WIKIPEDIA**  
*The Free Encyclopedia*

# What is Rx?

**Language neutral model with 3 concepts:**

**1. Observer/Observable**

2.



# Rx is everywhere\*

**.NET**

**JavaScript (RxJS)**

**Java (RxJava)**

**+ Scala, Groovy, Clojure**

**Objective-C (ReactiveCocoa)**

**C++**

**Python**

**Ruby**

**PHP**

**Dart**

**Haskell**

\* Varying levels of completeness – YMMV



# What is Rx?

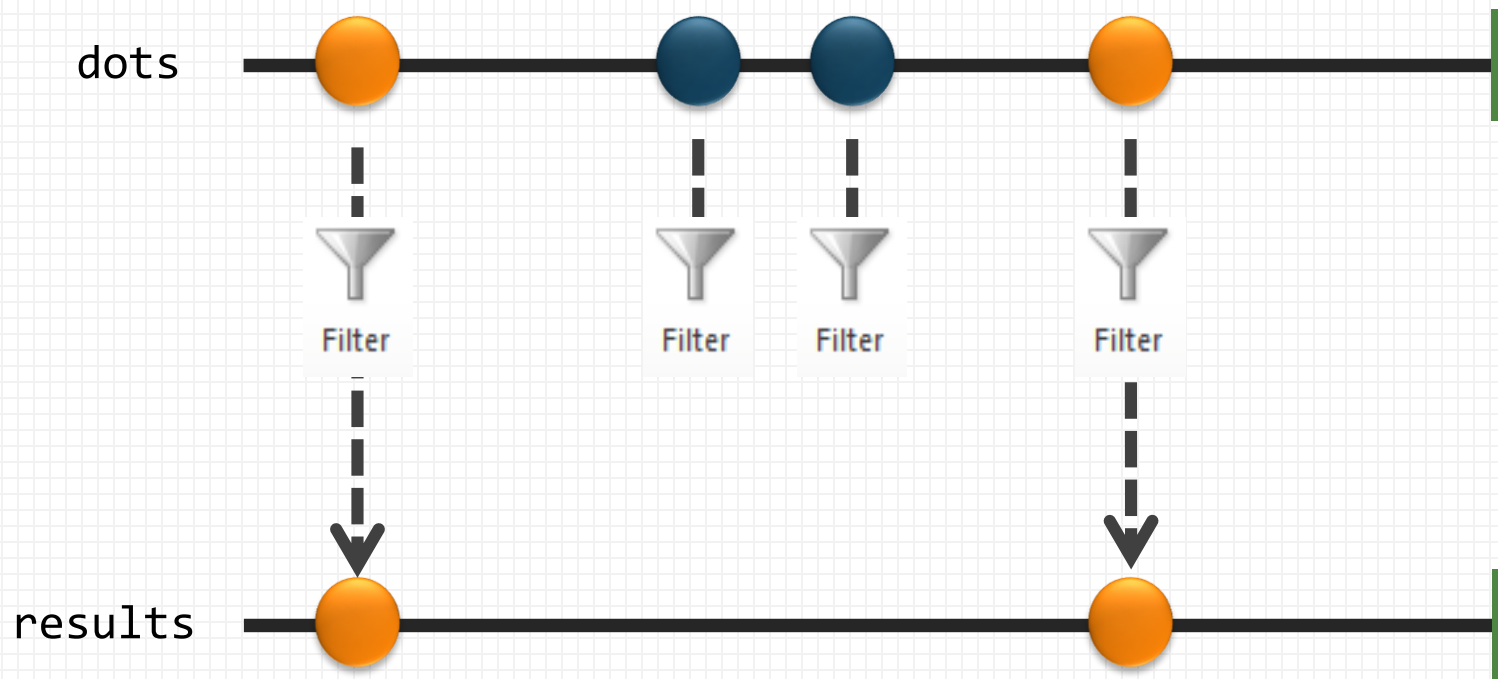
Language neutral model with 3 concepts:

1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
  - Schedulers: a set of types to parameterize concurrency





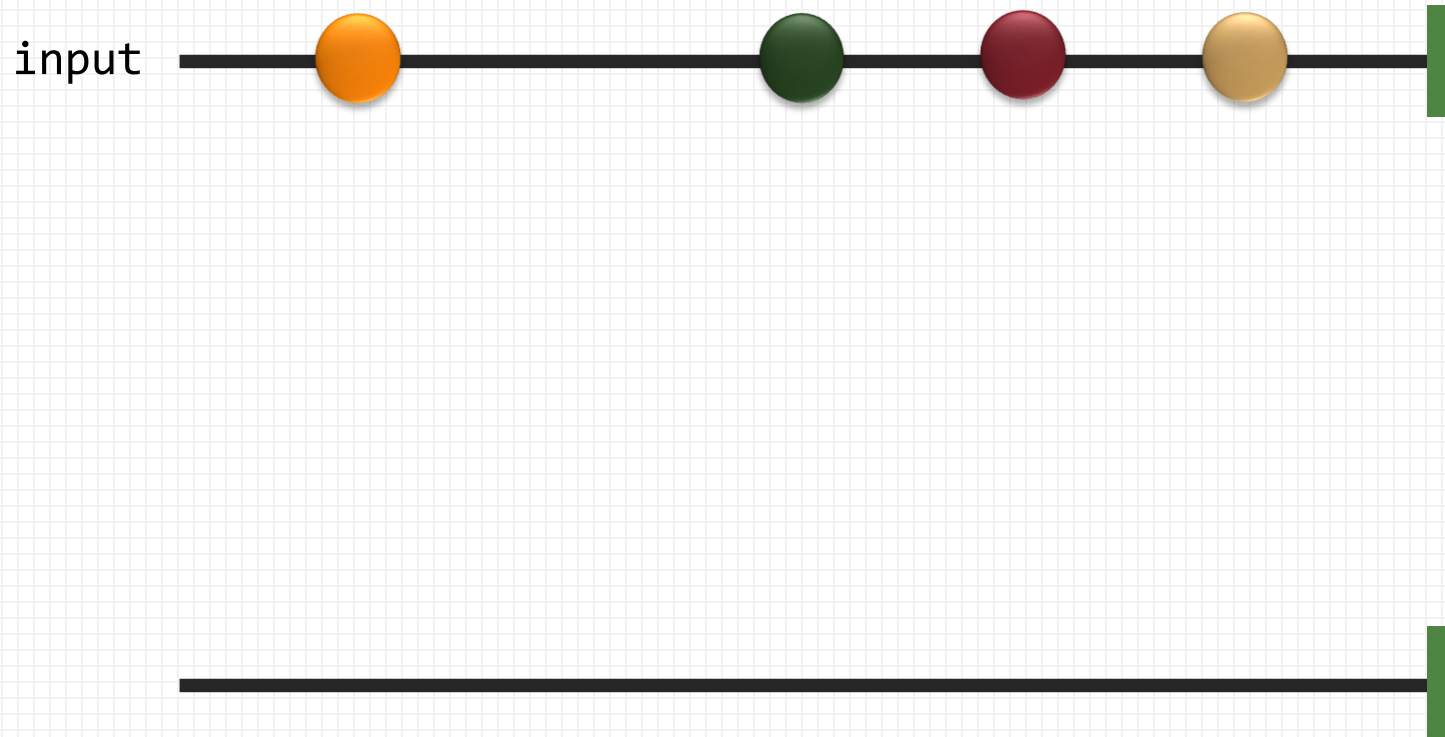
# Marble diagram: filter



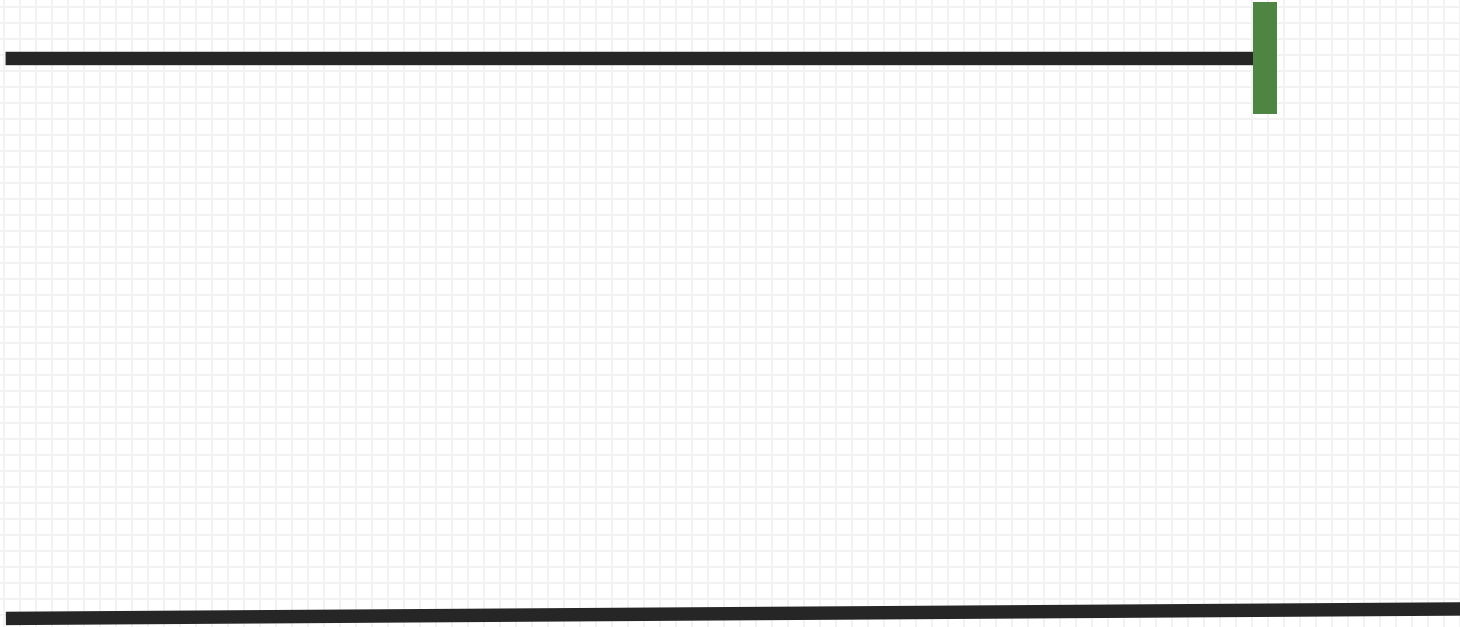
```
.filter(function (dot) {  
  return dot.isOrange();  
})
```



# Marble diagram: map



# Marble diagram: delay





# Querying UI Events



```
var mousedrag = mousedown.flatMap(function (md) {
```

```
    // calculate offsets when mouse down
```

```
    var startX = md.offsetX,  
        startY = md.offsetY;
```

1

For each mouse down

```
});
```



# Querying UI Events



```
var mousedrag = mousedown.flatMap(function (md) {
```

```
    // calculate offsets when mouse down
```

```
    var startX = md.offsetX,  
        startY = md.offsetY;
```

```
    // calculate diffs until mouse up
```

```
    return mousemove.map(function (mm) {
```

```
        return {
```

```
            left: mm.clientX - startX,
```

```
            top:  mm.clientY - startY
```

```
        };
```

```
    });
```

```
});
```

1

For each mouse down

2

Take mouse moves

# Querying UI Events



```
var mousedrag = mousedown.flatMap(function (md) {
```

```
  // calculate offsets when mouse down
```

```
  var startX = md.offsetX,  
      startY = md.offsetY;
```

```
  // calculate diffs until mouse up
```

```
  return mousemove.map(function (mm) {
```

```
    return {
```

```
      left: mm.clientX - startX,
```

```
      top:  mm.clientY - startY
```

```
    };
```

```
  }).takeUntil(mouseup);
```

```
});
```

1

For each mouse down

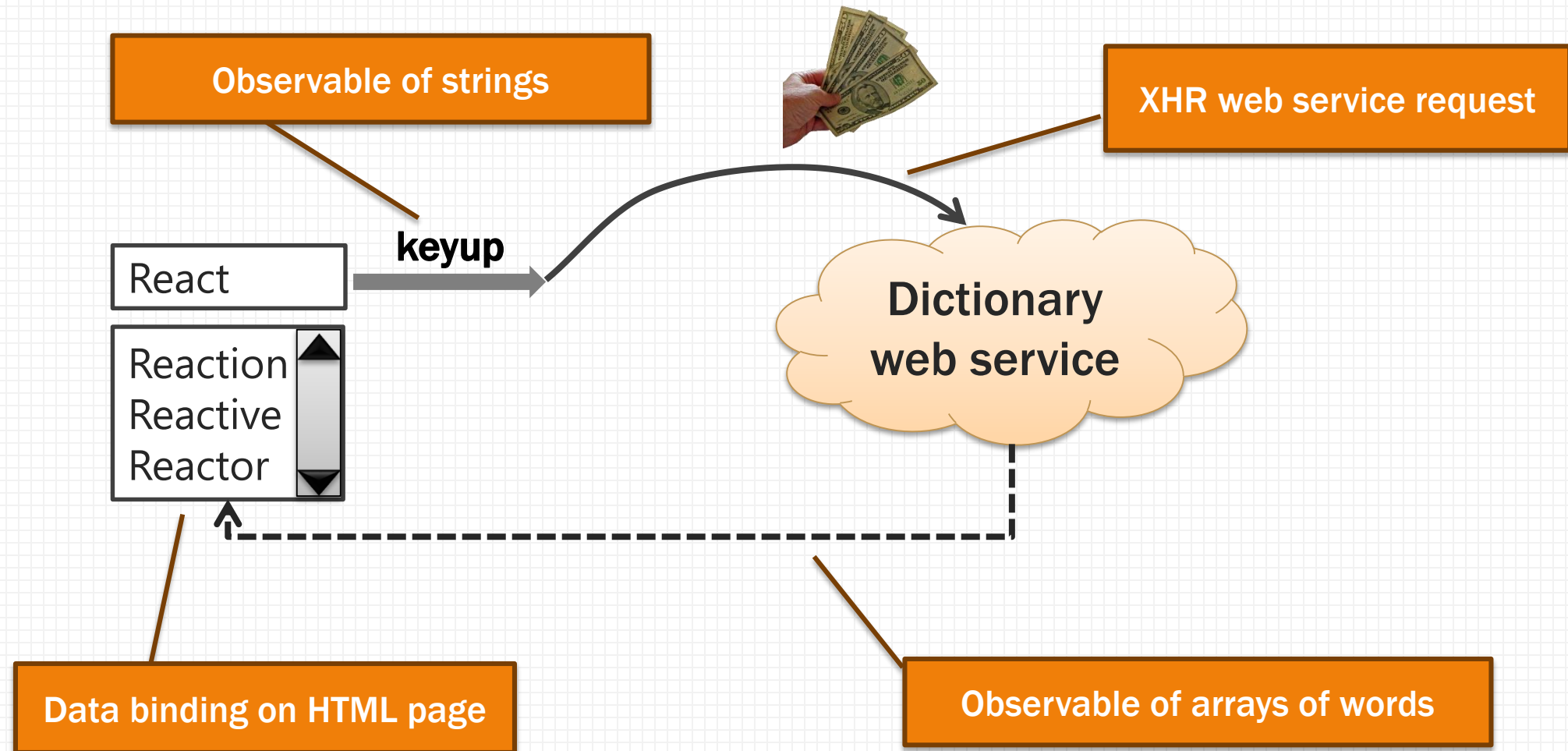
2

Take mouse moves


3

until mouse up

# Composing Events and Promises



# Composing Events and Promises



```
var words = Rx.Observable.fromEvent(  
    input, "keyup")  
    .map(function() { return input.value; })  
    .throttle(500)  
    .distinctUntilChanged()  
    .flatMapLatest(  
        function(term) { return search(term); }  
    );
```

DOM events as a  
sequence of strings

Reducing data  
traffic / volume

Latest response as  
word arrays


```
words.subscribe(function(data) {  
    // Bind data to the UI  
});
```


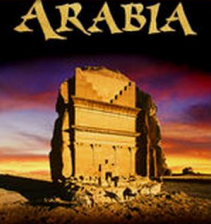

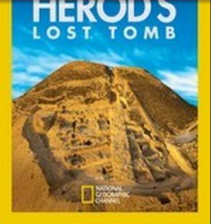

Web service call returns  
single value sequence

Binding results to the UI







# Your Netflix Video Lists

## Netflix Row Update Polling







2 / 10



### Top 10 for tester\_jhusain\_control



### Popular on Netflix



### Band Baaja Baaraat

2010 NR 2h 19m

★★★★★

Shruti and Bittoo decide to start a wedding planning company together after they graduate from university, but romance gets in the way of business.

Ranveer Singh, Anushka Sharma

Comedies, Foreign Movies

Director: Maneesh Sharma





# Client: Polling for Row Updates

```
function getRowUpdates(row) {  
    var scrolls = Rx.Observable.fromEvent(document, "scroll");  
    var rowVisibilities =  
        scrolls.throttle(50)  
            .map(function (scrollEvent) { return row.isVisible(scrollEvent.offset); })  
            .distinctUntilChanged()  
            .publish().refCount();  
    var rowShows = rowVisibilities.filter(function (v) { return v; });  
    var rowHides = rowVisibilities.filter(function (v) { return !v; });  
  
    return rowShows  
        .flatMap(Rx.Observable.interval(10))  
        .flatMap(function () { return row.getRowData().takeUntil(rowHides); })  
        .toArray();  
};
```



# What is Rx?

Language neutral model with 3 concepts:

1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
  - Schedulers: a set of types to parameterize concurrency



# The Role of Schedulers

## Key questions:

- How to run timers?
- Where to produce events?
- Need to synchronize with the UI?

## Schedulers are the answer:

- Schedulers introduce concurrency
- Operators are parameterized by schedulers
- Provides test benefits as well

Cancellation

Many  
implementations

```
d = scheduler.schedule(  
    function () {  
        // Asynchronously  
        // running work  
    },  
    1000);
```

Optional time



# Testing concurrent code: made easy!

```
var scheduler = new TestScheduler();
```

```
var input = scheduler.createColdObservable(  
    onNext(300, "FutureJS"),  
    onNext(400, "2014"),  
    onCompleted(500));
```

```
var results = scheduler.startWithCreate(function () {  
    input.map(function (x) { return x.length; })  
});
```

```
results.messages.assertEqual(  
    onNext(300, 8),  
    onNext(400, 4),  
    onCompleted(500));
```



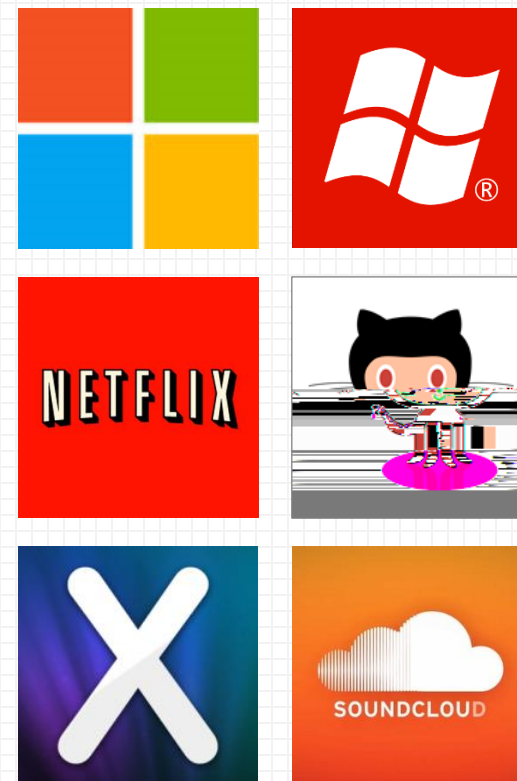
# More about Rx

Open-sourced by MS Open Tech in Nov 2012

- Rx.NET
- RxJS
- RxCpp

## Who uses Rx?

- Netflix ported it to Java (RxJava)
  - Heavily used in back-end
  - Use RxJS/Rx.NET on clients
- GitHub
  - GitHub for Windows (ReactiveUI + Rx.NET)
  - GitHub for Mac (ReactiveCocoa)





# RxJS and the future...

**What are the problems we're looking at next?**

- **Backpressure**
- **Distributed Rx**
- **Query Expressions in JavaScript**
- **Reactive-Streams**
- **Generators**



# Reactive Streams

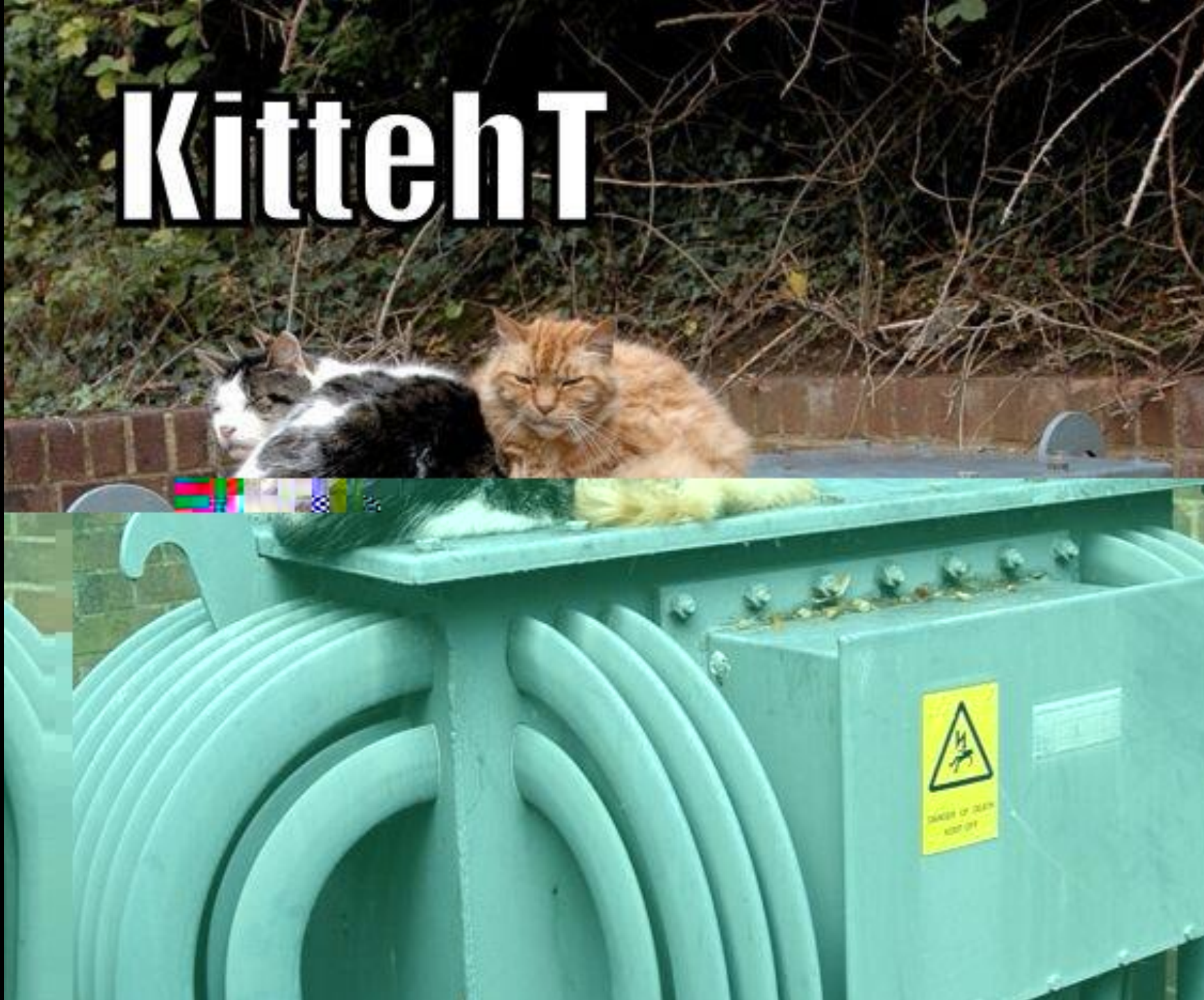
Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure on the JVM.

## The Problem

—Handling streams of data—especially “live” data whose volume is not predetermined—requires special care in an asynchronous system. The most prominent issue is that resource consumption needs to be carefully controlled such that a fast data source does not overwhelm the stream destination. Asynchrony is needed in order to enable the parallel use of computing resources, on collaborating network hosts or multiple CPU cores within a single machine.

<http://www.reactive-streams.org/>

# KitttehT



# RxJS and Generators

```
var Rx = require('rx');
var request = require('request');

var get = Rx.Observable.fromNodeCallback(request);

Rx.spawn(function* () {
  var a = yield get('http://localhost/stocks1.csv').retry(3);
  console.log(a.length);

  try {
    var b = yield get('http://invalidhost');
  } catch (e) {
    console.log(e.code);
  }
});
```



# Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure on the JVM.

## The Problem

—Handling streams of data—especially “live” data whose volume is not predetermined—requires special care in an asynchronous system. The most prominent issue is that resource consumption needs to be carefully controlled such that a fast data source does not overwhelm the stream destination. Asynchrony is needed in order to enable the parallel use of computing resources, on collaborating network hosts or multiple CPU cores within a single machine.

<http://www.reactive-streams.org/>



# Observables and Backpressure

## Yes, Observables can have backpressure

- Can be lossy (pausable, sample, throttle)
- Can be lossless (buffer, pausableBuffered, controlled)
- Will be built into the subscriptions starting in 2.4

```
var pausable = chattyObservable.pausableBuffered();  
pausable.pause();  
pausable.resume();
```

```
var subscription = chattyObservable.subscribe(print);  
subscription.request(10);
```





# Where do we go from here?

## ES7 and Beyond!

- First class events FTW!
- Async Generators!

```
async function* getDrags(element) {  
  for (let mouseDown on element.mouseDowns) {  
    for (let mouseMove on  
      document.mouseMoves.takeUntil(document.mouseUps)) {  
      yield mouseMove;  
    }  
  }  
}
```

<http://esdiscuss.org/notes/2014-06/async%20generators.pdf>



# OnCompleted: Rx

**Language neutral model with 3 concepts:**

- 1. Observer/Observable**
- 2. Query operations (map/filter/reduce)**
- 3. Schedulers: a set of types to parameterize concurrency**



@ReactiveX

[github.com/Reactive-Extensions](https://github.com/Reactive-Extensions)

