



# Async JavaScript at Netflix

Matthew Podwysocki @mattpodwysocki  
[github.com/mattpodwysocki/jsconfuy-2015](https://github.com/mattpodwysocki/jsconfuy-2015)

A black and white aerial photograph of a coastal town. In the center, a large suspension bridge spans a wide river or bay. The town below is built on a hillside, with numerous buildings and roads visible. In the far distance, a range of mountains is visible under a clear sky.

# OR: HOW I LEARNED TO STOP WORRYING ABOUT

**Or "I thought I had a problem. I thought to myself,  
"I know, I'll solve it with promises and events!".  
have Now problems. two I**

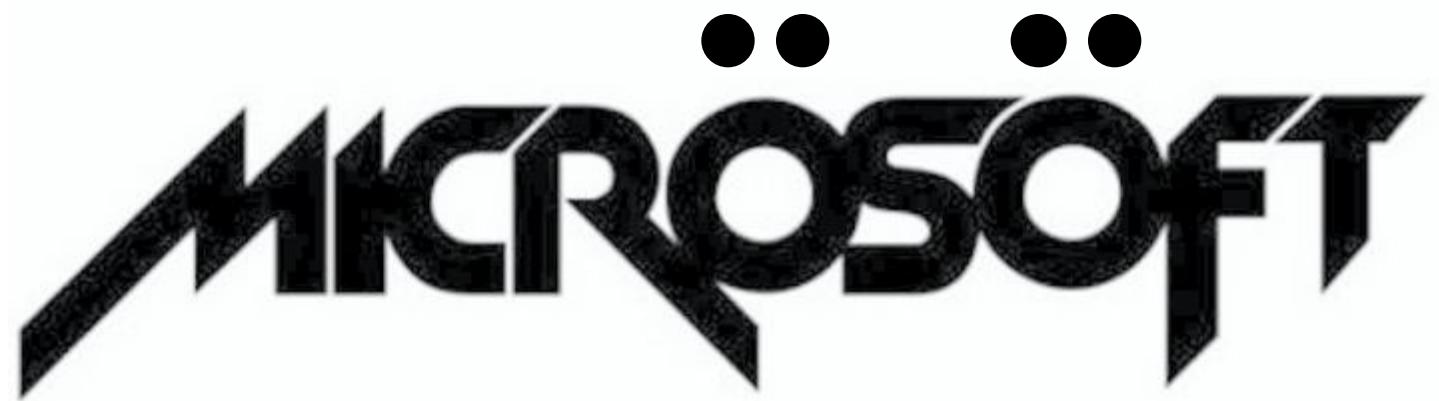


\*\*\*trapd in Monad tutorl  
plz help

**A MONAD IS JUST A  
MONOID IN THE CATEGORY OF  
ENDOFUNCTORS.  
WHAT'S THE PROBLEM ?**

A photograph of a person from the waist up, wearing a dark long-sleeved shirt and a Santa hat. They are holding a small, wrapped gift bag in their hands. The background is a plain, light-colored wall.

**Principal SDE  
Open Sourcerer  
[@mattpodwysocki](https://github.com/mattpodwysocki)  
[github.com/mattpodwysocki](https://github.com/mattpodwysocki)**



The Microsoft logo consists of the word "MICROSOFT" in a bold, black, sans-serif font. The letters are slanted slightly to the right. Above the letter "I", there are two small black dots. Above the letter "O", there are three black dots arranged horizontally.



# Reactive Extensions (Rx)

@ReactiveX  
<http://reactivex.io>

The Netflix logo, featuring the word "NETFLIX" in its signature red, bold, sans-serif font, centered on a white rounded rectangle.

Stream Movies From Any Device

$1/3$  of US Broadband Traffic

This is the story of how Netflix solved

# BIG async problems

by thinking differently about

**Events.**

# The Netflix App is Asynchronous

- 
- 
- 
- 
- 
- /



# Three Years Ago...

•  
•  
• ,



# Today at Netflix

- 
- 0+
- 



# Real-Time is Everywhere...



SPIKE LATHAM

# Let's Face It, Asynchronous Programming is Awful!



**“We choose to go to solve asynchronous  
programming and do the other things,  
not because they are easy, but because  
they are hard”**



# Callback Hell





culturepub.fr

next  
ad ➔

# Events and the Enemy of the State





# First Class Async with Promises



# Breaking the Promise...





# Aborting a fetch #27

[New issue](#)

Open [annevk](#) opened this issue 26 days ago · 182 comments



[annevk](#) commented 26 days ago

Owner

## Goal

Provide developers with a method to abort something initiated with `fetch()` in a way that is not overly complicated.

## Previous discussion

- [#20](#)
- [slightlyoff/ServiceWorker#592](#)
- [slightlyoff/ServiceWorker#625](#)
- [whatwg/streams#297](#)

## Viable solutions

We have two contenders. Either `fetch()` returns an object that is more than a promise going forward or `fetch()` is passed something, either an object or a callback that gets handed an object.

<https://github.com/whatwg/fetch/issues/27>

### Labels

None yet

### Milestone

No milestone

### Assignee

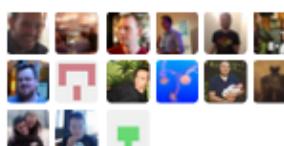
[jakearchibald](#)

### Notifications

[Subscribe](#)

You're not receiving notifications from this thread.

### 15 participants



# 3.0 cancellation overhaul #415

[New issue](#)

Open

petkaantonov opened this issue on Dec 27, 2014 · 79 comments



petkaantonov commented on Dec 27, 2014

Owner

The current cancellation seems to be the most problematic feature due to some annoying limits in the design:

- Awkward API for attaching the callback that cancels the work (through typed catch handler)
- Cannot compose because it is impossible to tell the difference between rejection and cancellation (e.g. `Promise.all(...).cancel()` cannot do the obvious thing and also call `cancel()` on the promises in the array)
- `.cancel()` is asynchronous

Since all consumers and producers must be trusted/"your code", there is no reason to enforce that cancellable promises are single-consumer or create new primitives.

**Edit:** The below design has been scratched, see [#415 \(comment\)](#)

Labels

3.0.0

Milestone

No milestone

Assignee

No one assigned

Notifications

Subscribe

You're not receiving notifications from this discussion

# The Final Countdown...





MAKE GIFS AT GIFOUP.COM



# UNSAFE AT ANY SPEED

The Designed-In Dangers  
of The American Automobile  
By Ralph Nader



**stroom prosesing**

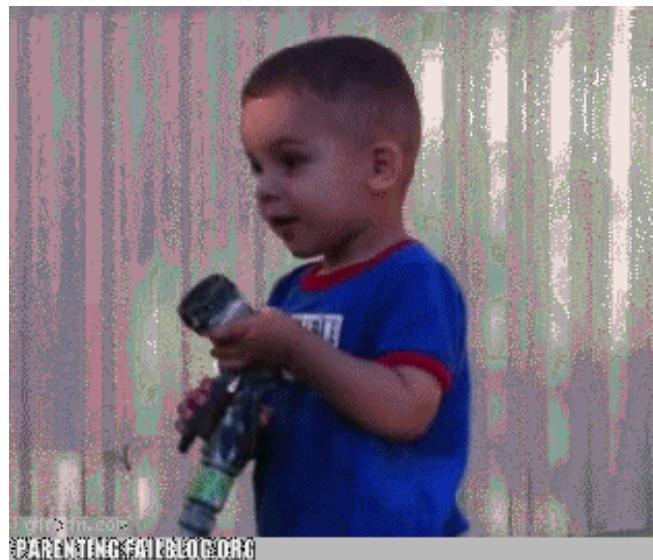


,

1

!

,





1/

0. .

0.11.



# 1994



# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



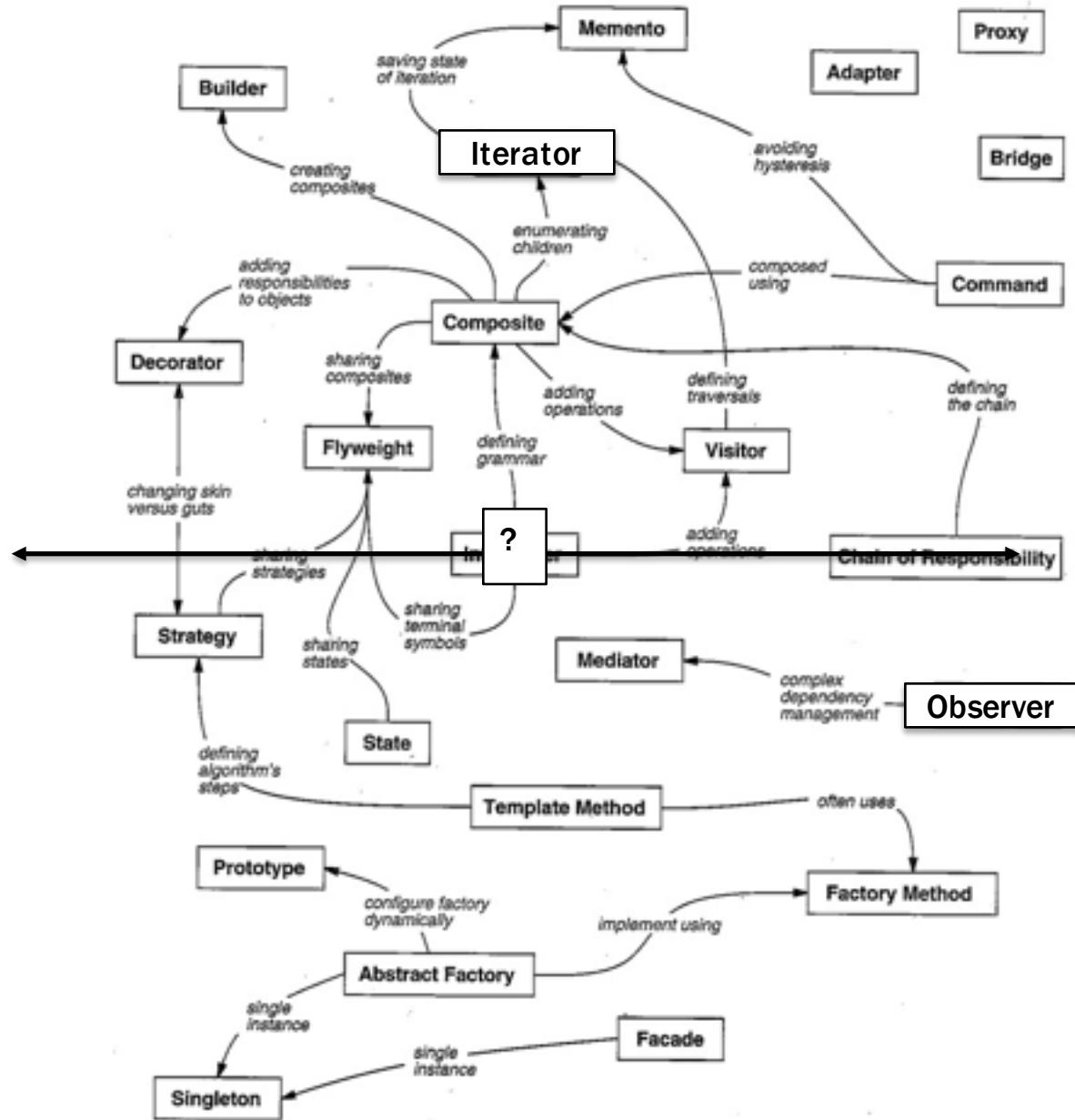
Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



Cover

For



## Design Pattern Relationships

# Iterator Pattern with ES2015

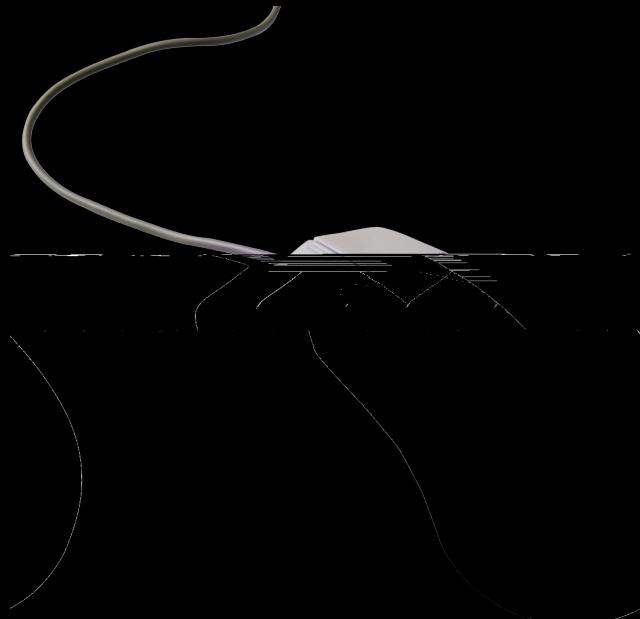
```
> var iterator = getNumbers(); █  
> console.log(iterator.next()); █  
> { value: 1, done: false }  
> █onsole.log(iterator.next()); █  
> { value: 2, done: false }  
> █onsole.log(iterator.next()); █  
> { value: 3, done: false }  
> █onsole.log(iterator.next()); █  
> { done: true }  
> █
```

# Subject/Observer Pattern with the DOM

```
> document.addEventListener(  
  'mousemove',  
  (e) =>  
    console.log(e);  
); ■  
  
> { clientX: 425, clientY: 543 }  
> { clientX: 450, clientY: 558 }  
> { clientX: 455, clientY: 562 }  
> { clientX: 460, clientY: 743 }  
> { clientX: 476, clientY: 760 }
```

“What’s the difference  
between an Array...

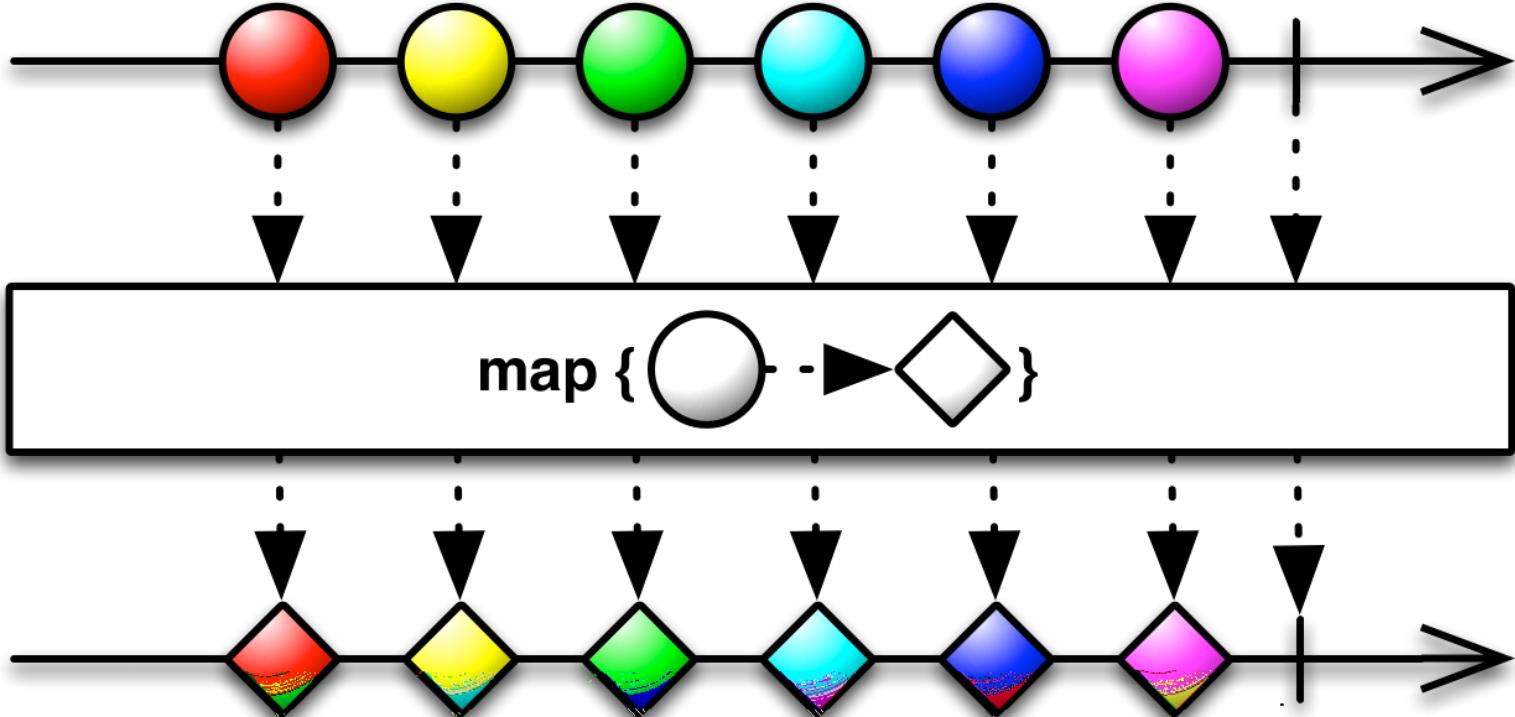
... and an Event?



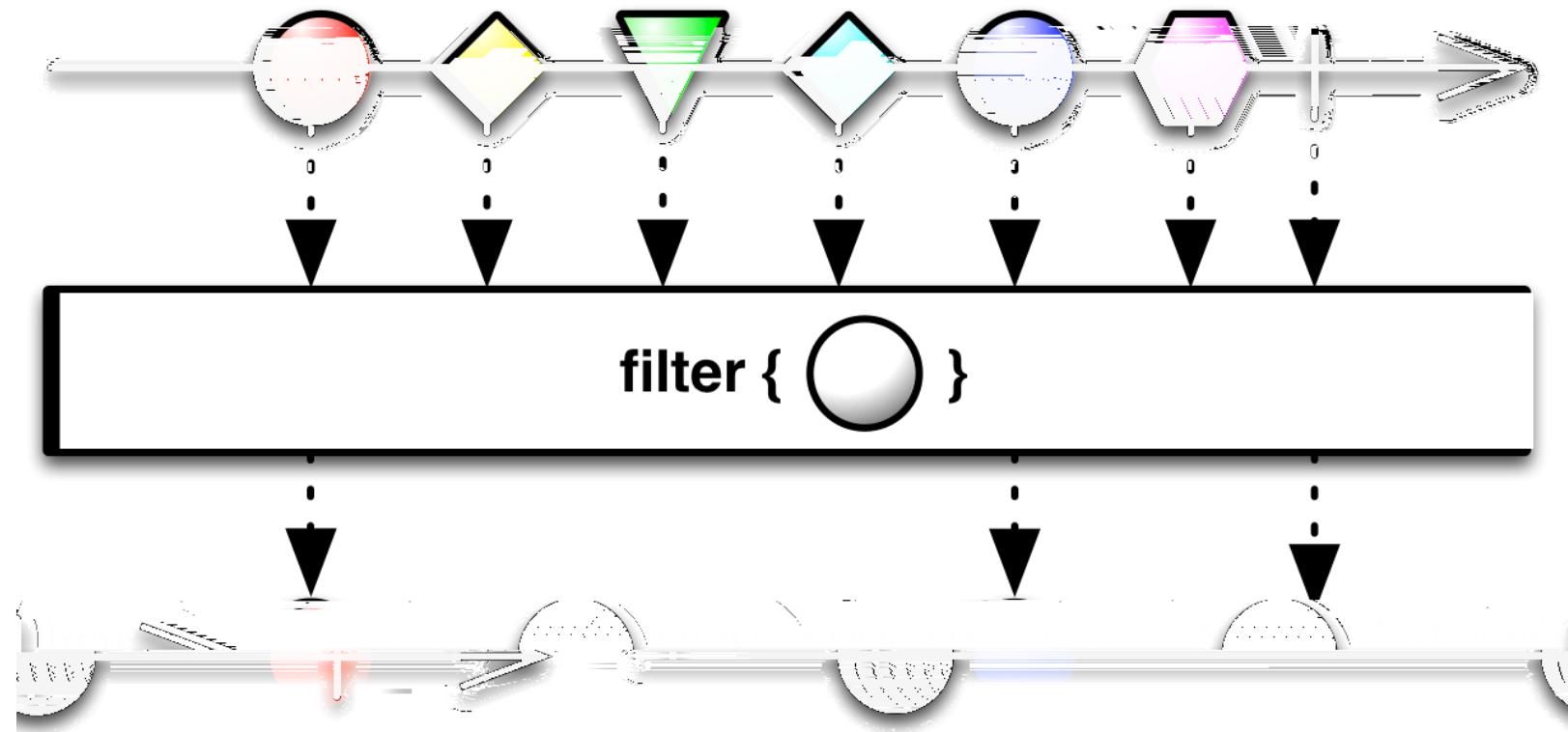
**Events and Arrays are *both*  
collections.**

The majority of Netflix's  
asynchronous code is written with  
just a few ***flexible*** functions.

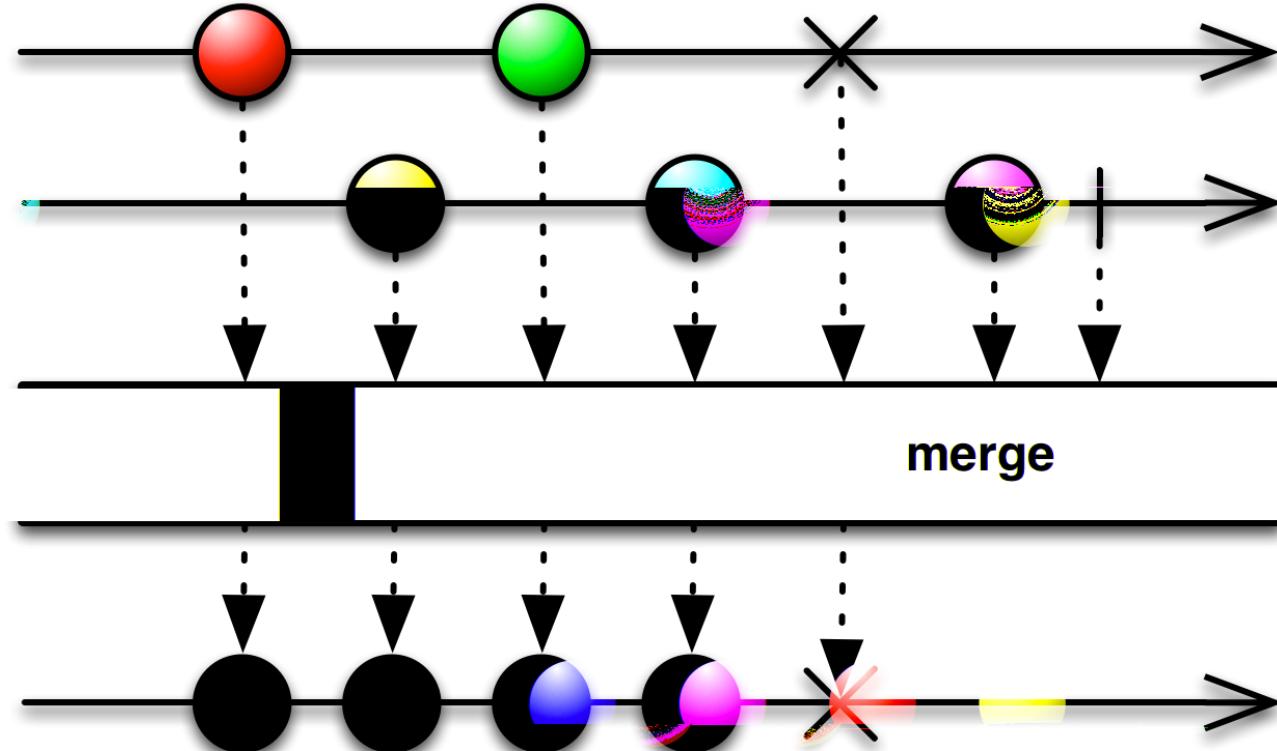
# map()



# filter()

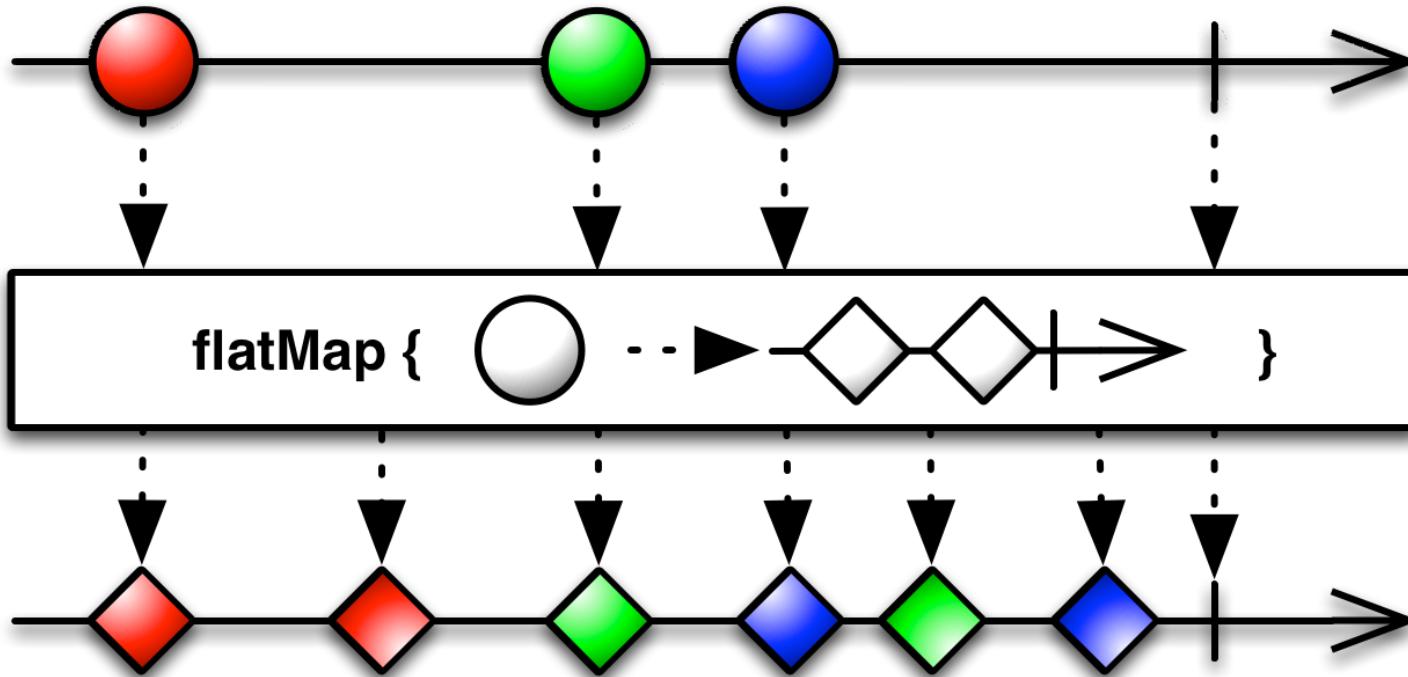


# mergeAll()

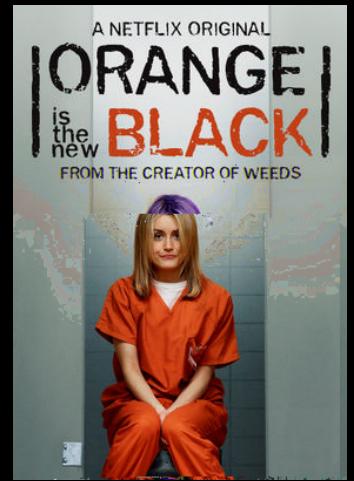


# flatMap()

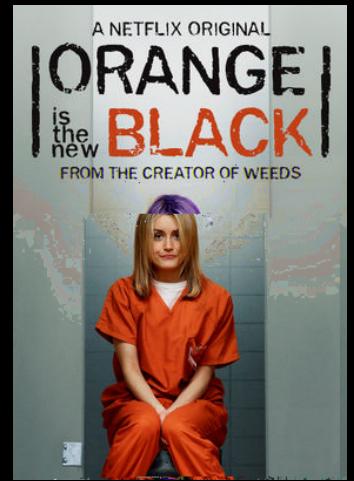
7



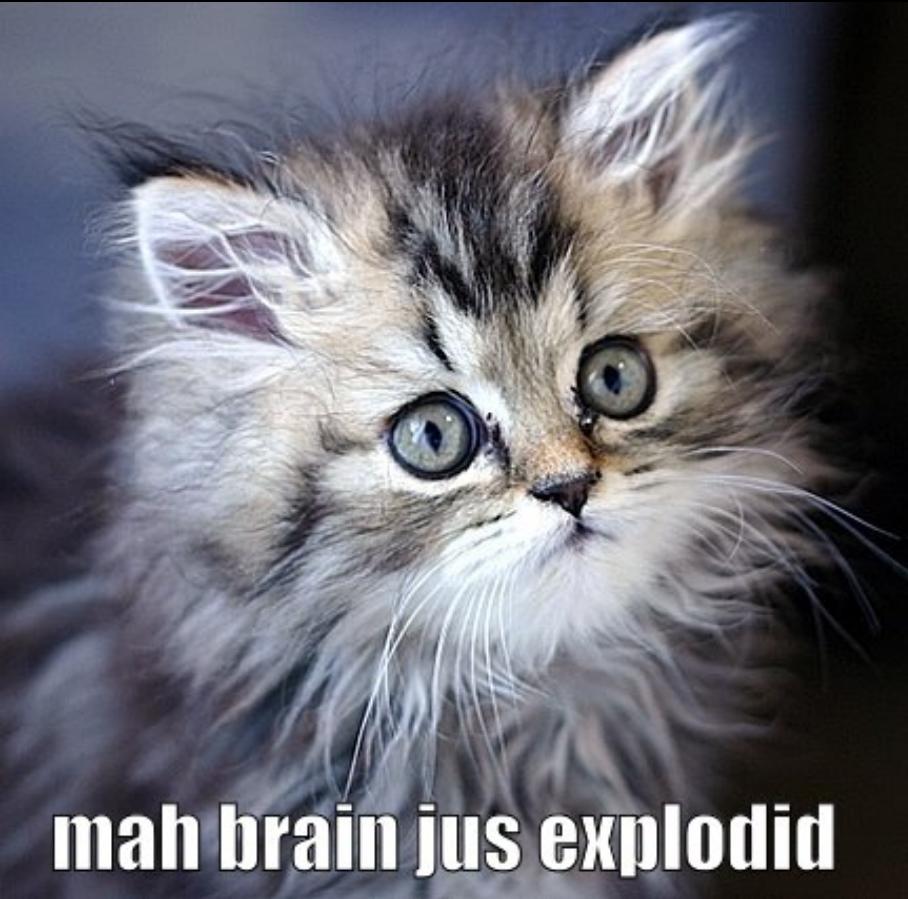
# Top-rated Movies Collection



# Top-rated Movies Collection



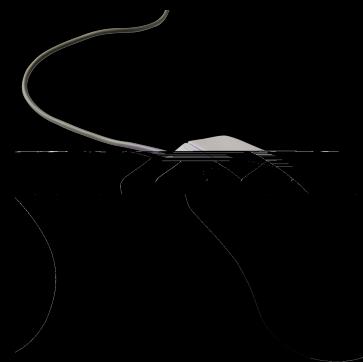
# What if I told you...



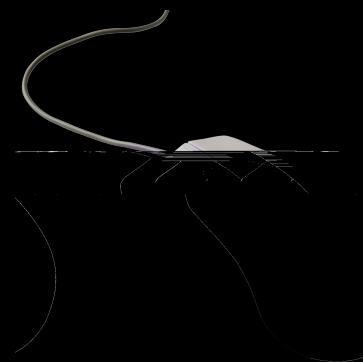
...that you could create a drag event...  
...with the almost the **same code**

**mah brain jus explodid**

# Mouse Drags Collection



# Mouse Drags Collection





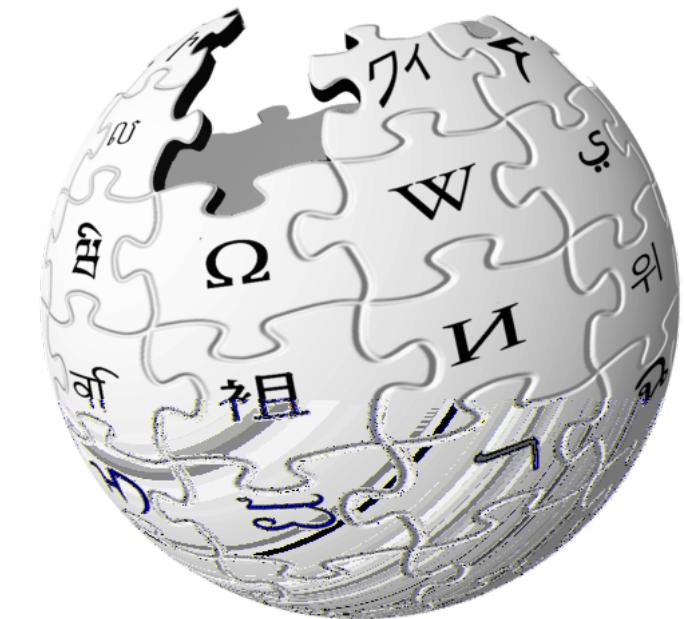
Everything is a stream

# First-Class Asynchronous Values



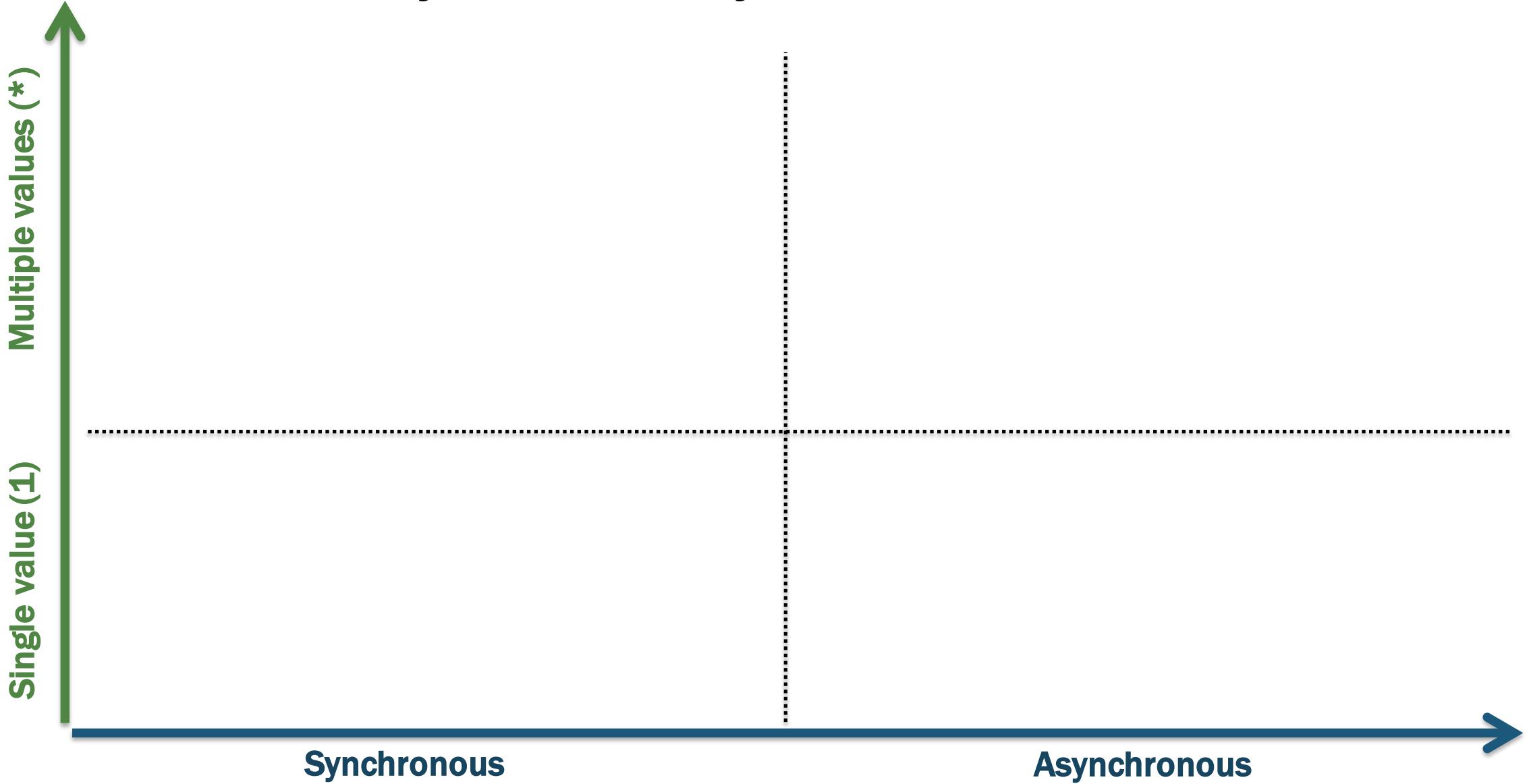
日本語  
Polski  
Русский

- can be stored in **variables** and **data structures**
- can be passed as a parameter to **functions**
- can be returned as the **result** of a function
- can be constructed at **runtime**
- has **intrinsic identity** (independent of its value)



WIKIPEDIA  
*The Free Encyclopedia*

# The General Theory of Reactivity



# What is Reactive Programming Anyhow?

Merriam-Webster defines reactive as “*readily responsive to a stimulus*”, i.e. its components are “active” and always ready to receive events.

**Wanna really know what Reactive Programming Is?**

Real Time Programming: Special Purpose or General Purpose Languages  
Gerard Berry

<http://bit.ly/reactive-paper>



# **Functional Reactive Programming (FRP) is...**

,

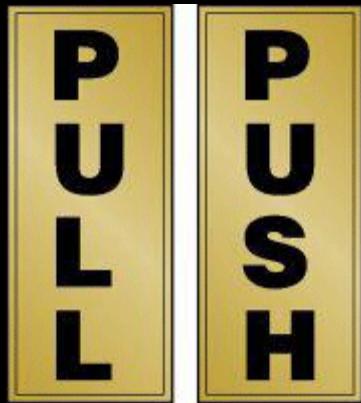
<http://stackoverflow.com/questions/1028250/what-is-functional-reactive-programming>

# Call Us CEP, Compositional Event Processing



THE DUDE

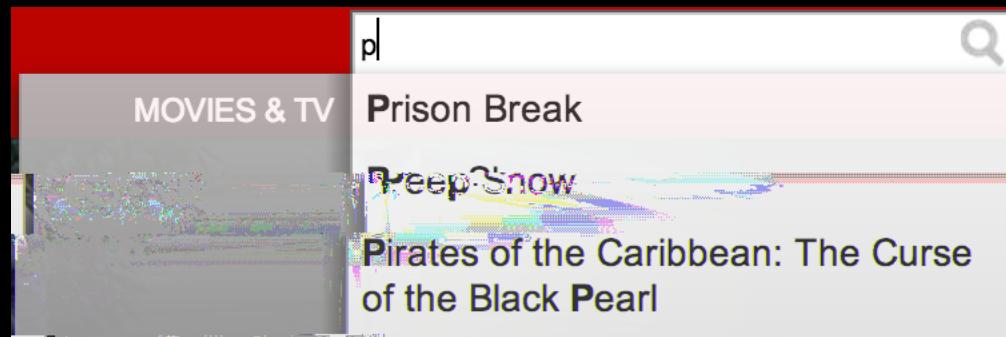
He's the Dude. So that's what you call him. You know, that or, uh, His Dudeness, or uh, Duder, or El Duderino.







# Netflix Search



# Netflix Search with Observables

```
let data =
```

```
  .map(input.value)
  .debounceTime(500)
  .filter()
  .map((term) => search(term))
);
```

Reducing data traffic / volume

DOM events as a sequence of strings

Latest response as movies

```
let subscription = data.forEach((data) => {
  // Bind data to the UI
});
```

Web service call returns single value sequence

Binding results to the UI

# What exactly is Rx?

1.

/

- ( / / )
- / /
- —



# Rx Grammar Police

onNext  \*

Zero or more values

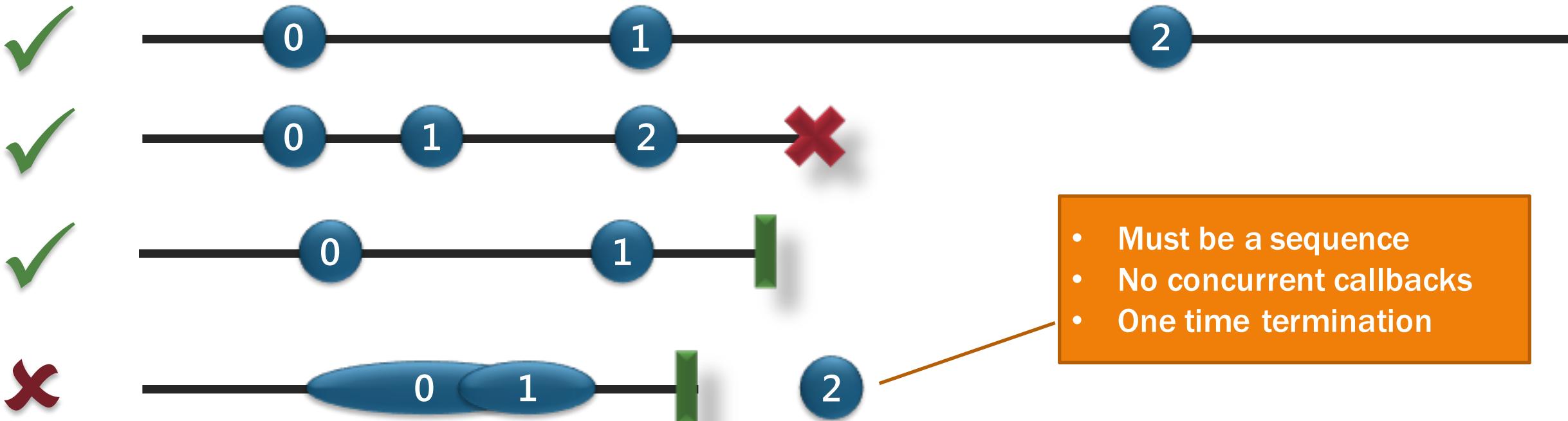
E.g. events are  $\infty$  sequences

( onError 

Calls can fail

onCompleted  ) ?

Resource management  
Sequencing



# What exactly is Rx?

# 1. /

- ( / / )  
▪ / /





# Observables - Querying UI Events

```
let mousedown = mousedown.  
    ((md) => {  
        // calculate offsets when mouse down  
        let startX = md.offsetX,  
            startY = md.offsetY;  
  
    });
```

For each mouse down

1



# Observables - Querying UI Events

```
let mousedrag = mousedown.          ((md) => {  
    // calculate offsets when mouse down  
    let startX = md.offsetX,  
        startY = md.offsetY;  
  
    // calculate diffs until mouse up  
    return mousemove.    ((mm) => {  
        return {  
            left: mm.clientX - startX,  
            top: mm.clientY - startY  
        };  
    })  
});
```

1 For each mouse down

2 Take mouse moves



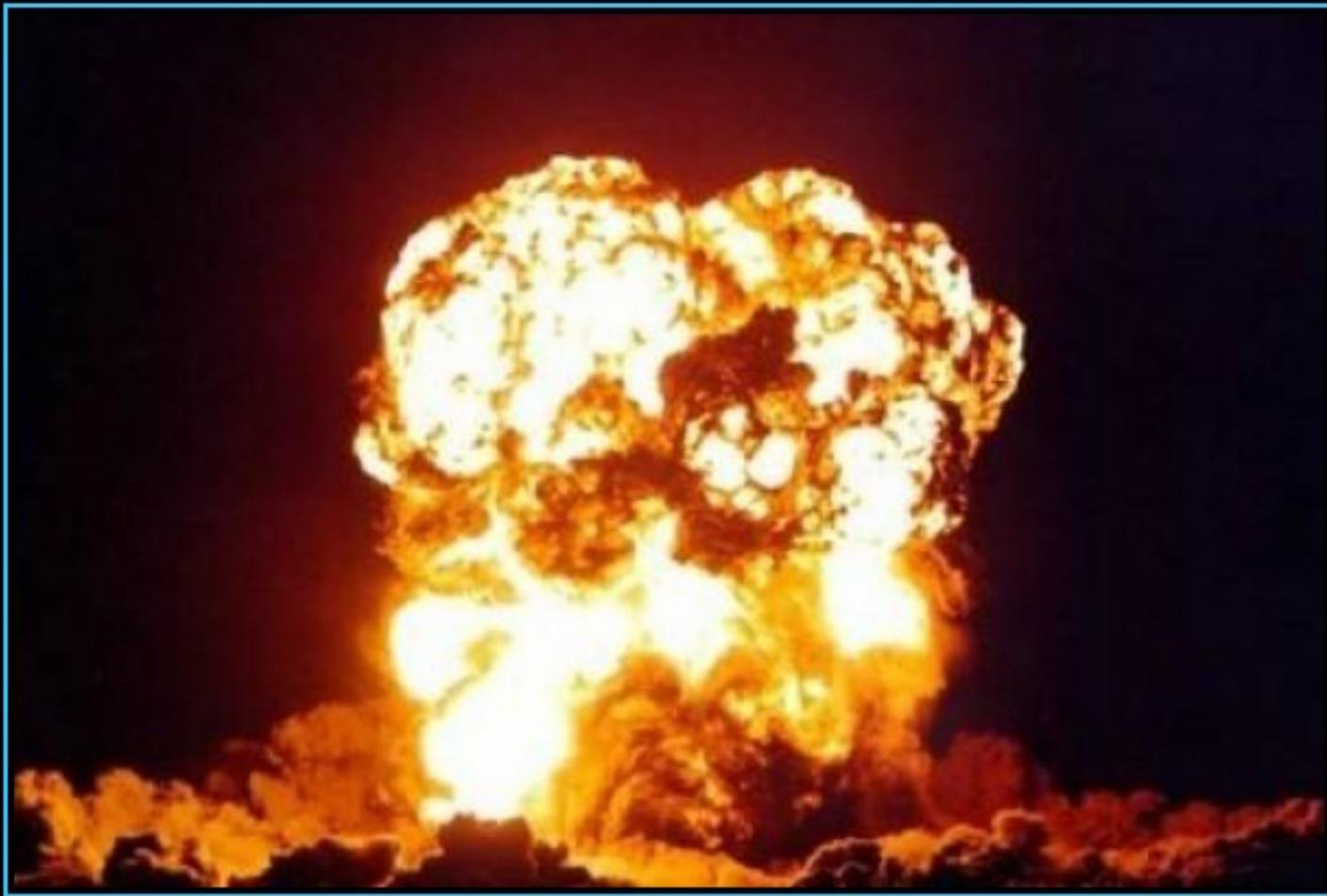
# Observables - Querying UI Events

```
let mousedrag = mousedown.          ((md) => {  
    // calculate offsets when mouse down  
    let startX = md.offsetX,  
        startY = md.offsetY;  
  
    // calculate diffs until mouse up  
    returnmousemove. ((mm) => {  
        return {  
            left: mm.clientX - startX,  
            top: mm.clientY - startY  
        };  
    }).      (mouseup);  
});
```

1 For each mouse down

2 Take mouse moves

3 until mouse up

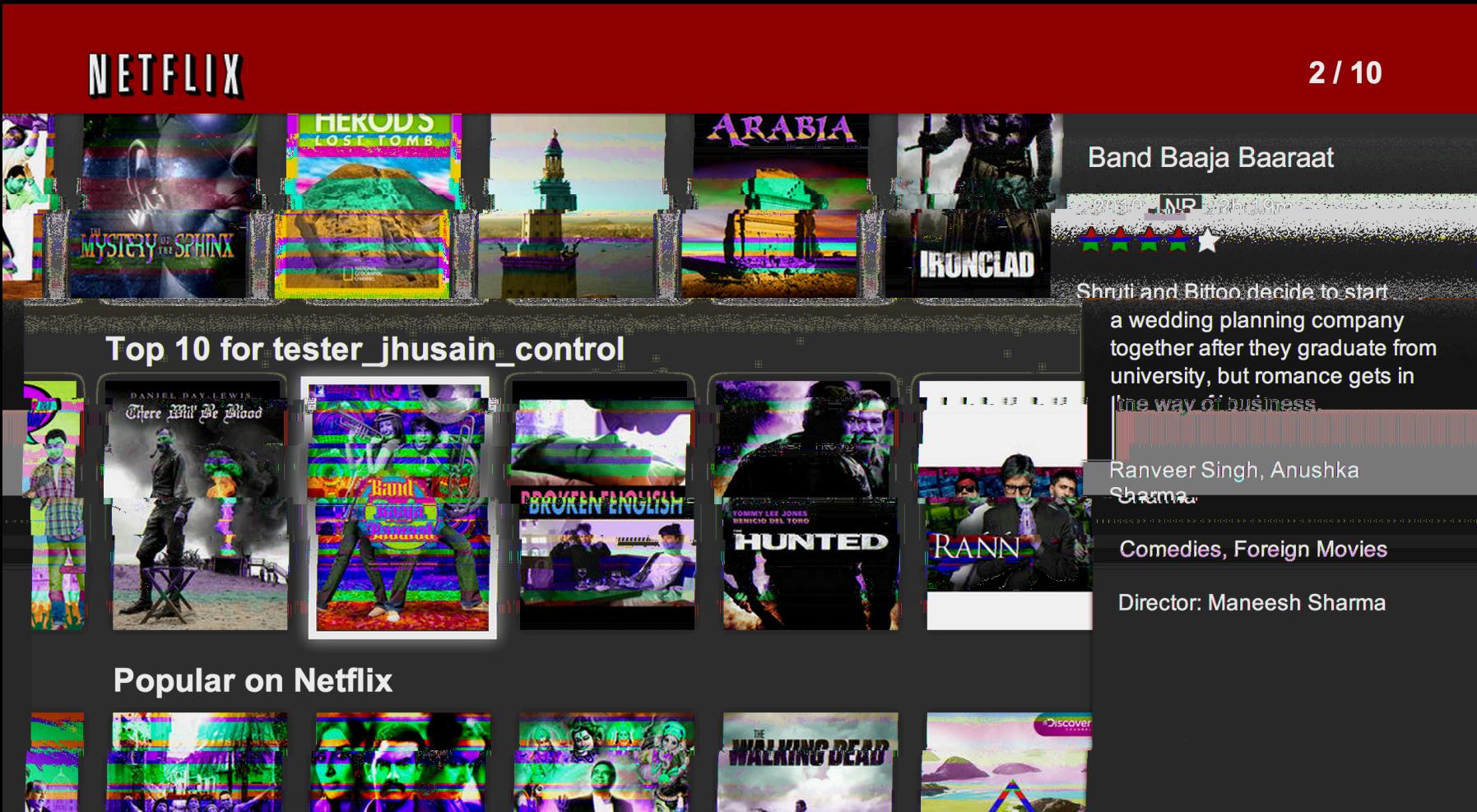


# PROTONIC REVERSAL

You crossed the streams, didn't you?

# Your Netflix Video Lists

## Netflix Row Update Polling



# Client: Polling for Row Updates

,

# Netflix Player



# Player Callback Hell



# Player With Observables



# RxSocketSubject

build passing

A simple web socket wrapper for RxJS

A more advanced

## Install

With bower:

```
bower install
```

With npm:

```
npm install -
```

```
-S rx-socket-subject
```

```
S rx-socket-subject
```

This project is to produce an observable WebSocket implementation that meets a set of common needs for my current project I'm currently working on. RxJS-DOM has a fine WebSocket implementation, which I modelled the initial work on. However, I need something that also does the following:

## Goals

The goals of this project I'm currently working on are to implement a WebSocket implementation that is:



# The Final Countdown...



C:\>DIR A:

Not ready reading drive A  
Abort, Retry, Fail?\_

# What is Rx?

1.

/

.

( / / )

.

/

/

-



# The Role of Schedulers

-  
-  
-  
-  
-

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

# Testing concurrent code: made easy!

```
let scheduler = new TestScheduler();

let input = scheduler.createHotObservable(
    onNext(300, 'JSConf.UY'),
    onNext(400, '2015'),
    onCompleted(500));

let results = scheduler.startWithCreate(() =>
    input.pluck('length')
);

results.messages.assertEqual(
    onNext(300, 9),
    onNext(400, 4),
    onCompleted(500));
```

# Observables and Backpressure

,

- ( , , , )

- ( , , , )



# Async/Await

!

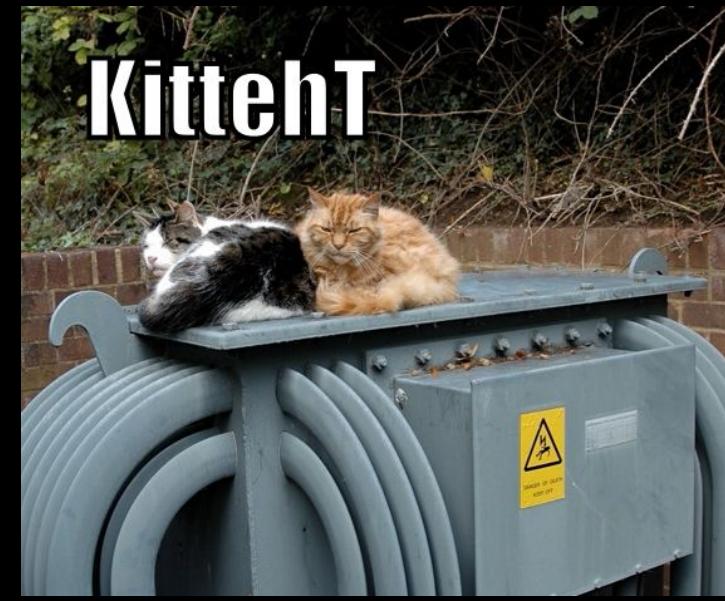
-

1

01

# Async/Await with Observables and Generators...

- /



KittehT

# Async Generators

!

—  
—

01

This is an interactive learning course with exercises you fill out right in the browser. If you just want to browse the content click the button below:

Show all the answers so I can just browse.

# Functional Programming in Javascript

Ever type down some code and realize you wrote the same logic over and over again? Functional programming is a way to write code that's more readable, reusable, and maintainable. In this tutorial you'll learn how to do that by learning five simple functions that you can use to perform any operation you want on collections. These functions are called composable building blocks. You'll be surprised to learn that most of the operations you accomplish with **five simple functions**.

1. map
2. filter
3. concatAll
4. reduce
5. zip

After, more self-descriptive, and more durable. The five functions hold the key to have all the tools you need to easily handle events and AJAX requests. In short, these functions you'll ever learn.

Here's my promise to you: if you learn these 5 functions your code will become shorter, more self-descriptive, and more durable. Also, for reasons that might not be obvious right now, you'll learn that these functions will probably be the most powerful, flexible, and useful functions you'll ever learn.

# RxMarbles

## Interactive diagrams of Rx Observables

### TRANSFORMING OPERATORS

[delay](#)

[delayWithSelector](#)

[findIndex](#)

[map](#)

[scan](#)

[throttle](#)

[throttleWithSelector](#)

### COMBINING OPERATORS

[combineLatest](#)

[concat](#)

[merge](#)

[sample](#)

[startWith](#)

[zip](#)

### FILTERING OPERATORS

[distinct](#)

[distinctUntilChanged](#)

[elementAt](#)

[filter](#)



merge



# SWEETEN YOUR JAVASCRIPT

[Compile](#)[Eval](#)[Step 0](#) readable names auto-compile  
 macro highlighting[vim](#)  
[emacs](#)  
[default](#)

```
1 // Welcome to sweet.js!
2 // sweet.js is a single pass macro system for javascript.
3 // When you edit your code,甜js will automatically compile it to javascript on the right.
4 // This page will also save your code to localstorage on every successful
5 // compile so feel free to close the page and come back later!
6 /*
7 */
8
9
10 // There is a really simple identity macro to test it out.
11
12 // The macro keyword is used to create and name new macros.
13 macro id {
14     rule {
15         // Create the macro rule.
16         // (1) an open paren.
17         // (2) a single token and bind it to $x.
18         // (3) a close paren.
19         ($x)
20     } -> [
21         // Just return the token we bound to, $x.
22         $x
23     ]
24 }
25 id ( 42 );
```

```
1 42;
2 // Note that a single token to sweet.js includes matched
3 // delimiters not just numbers and identifiers. For example,
4 // if you write 1,2,3 it will be interpreted as three tokens.
5 [
6     1,
7     2,
8     3
9 ];
10 // One of the really important things sweet.js does is protect
11 // macros from unintentionally binding or capturing variables
12 // we're not supposed to. This is called hygiene and to enforce
13 // it sweet.js must carefully rename all variable names.
14 var x;
15 var foo = 100;
16 var bar = 200;
17 var tmp = 'my other temporary variable';
18 var tmp$2 = bar;
19 bar = foo;
20 foo = tmp$2;
```

