

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

**NETFLIX**

# Async JavaScript at Netflix

**Matthew Podwysocki** @mattpodwysocki  
[github.com/mattpodwysocki/jsconfbp-2015](https://github.com/mattpodwysocki/jsconfbp-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

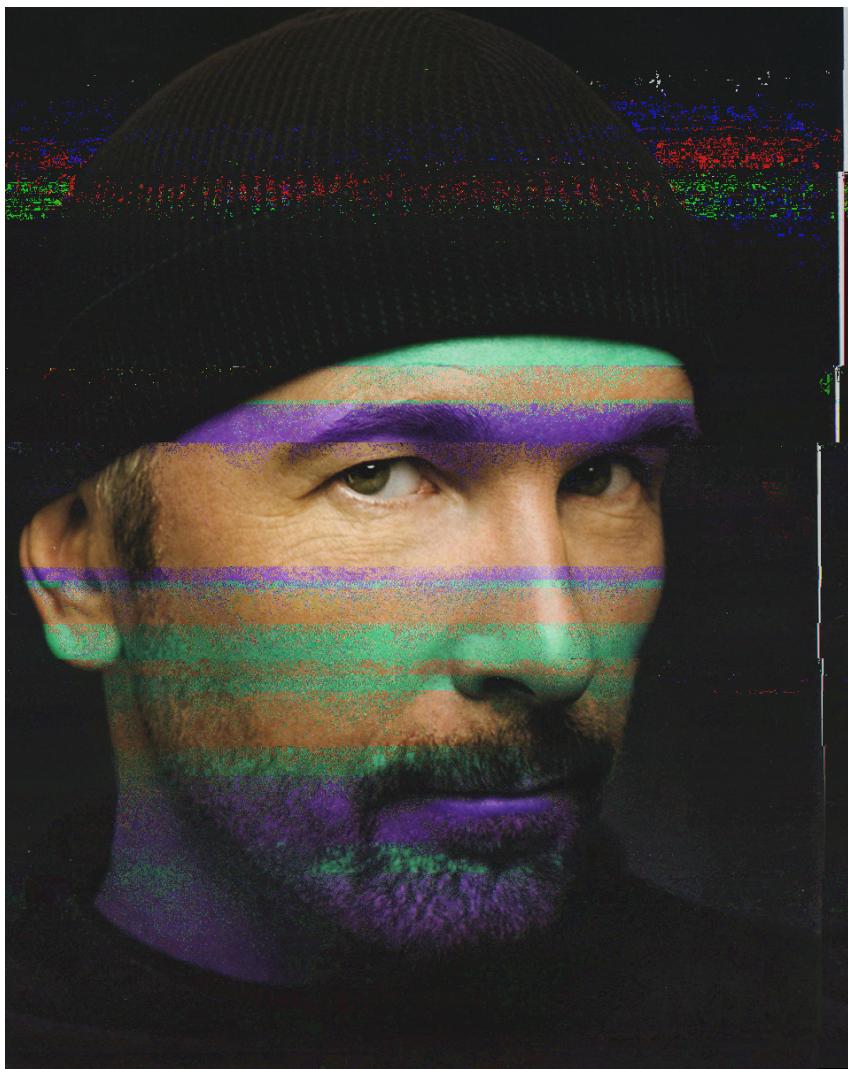
**A MONAD IS JUST A  
MONOID IN THE CATEGORY OF  
ENDOFUNCTORS.  
WHAT'S THE PROBLEM ?**

A photograph of a person from the waist up, wearing a dark long-sleeved shirt. They are wearing a Santa hat with a blue pom-pom and a red and white striped band. They are holding a small, wrapped gift bag with a red ribbon. The background is a dark, out-of-focus indoor setting.

**Principal SDE  
Open Sourcerer  
@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, and above the letter "O", there are three black dots, creating a stylized 'M' character.



**YOU AND ME EQUALS  
2.0000000000000004**

**THANKS JAVASCRIPT AND IEEE 754...**



# 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

- **App Startup**
- **Player**
- **Data Access**
- **Animations**
- **View/Model Binding**



# Three Years Ago...

- Complex asynchronous code
- Client and Server developers tightly coupled
- Different platforms, different approaches to async



# Today at Netflix

- Rx used on server and client
- 30+ developers using Rx in 6 different languages
- Same asynchronous model everywhere



# Real-Time is Everywhere...



SPIKE & SETH

# 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”**



**Former US President John F. Kennedy - 1962  
[citation needed]**

# Callback Hell

```
function play(movieId, callback) {  
  var movieTicket, playError,  
    tryFinish = function () {  
      if (playError) {  
        callback(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();  
  });  
}
```





culturepub.fr

next  
ad ➔

# Events and the Enemy of the State

```
var isDown = false, state;

function mousedown (e) {
  isDown = true;
  state = { startX: e.offsetX,
            startY: e.offsetY };
}

function mousemove (e) {
  if (!isDown) { return; }
  var delta = { endX: e.clientX - state.startX,
                endY: e.clientY - state.startY };
  // Now do something with it
}

function mouseup (e) {
  isDown = false;
  state = null;
}
```

```
function dispose() {
  elem.removeEventListener('mousedown', mousedown, false);
  elem.removeEventListener('mouseup', mouseup, false);
  doc.removeEventListener('mousemove', mousemove, false);
}

elem.addEventListener('mousedown', mousedown, false);
elem.addEventListener('mouseup', mouseup, false);
doc.addEventListener('mousemove', mousemove, false);
```





# First Class Async with Promises



```
player.initialize()  
  .then(authorizeMovie, loginError)  
  .then(playMovie, unauthorizedMovie)
```

# Breaking the Promise...



```
var promise;

input.addEventListener('keyup', (e) => {

  if (promise) {
    // Um, how do I cancel?
  } else {
    promise = getData(e.target.value).then(populateUI);
  }
}, false);
```



# 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

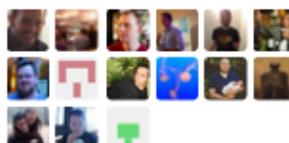
jakearchibald

### 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 \(comment\)](#)

<https://github.com/petkaantonov/bluebird/issues/415>

Labels

3.0.0

Milestone

No milestone

Assignee

No one assigned

Notifications

Subscribe

You're not receiving notifications from this issue

# The Final Countdown...



```
var resource;  
  
try {  
    var resource = getResource();  
} catch (e) {  
    throw e;  
} finally {  
    resource && resource.dispose();  
}  
  
getresource()  
    .success(  
        function (resource) {  
            // How do I clean up my resource?  
        })  
    .catch(  
        function (err) {  
            // How do I clean up my resource?  
        });
```



MAKE GIFS AT GIFOUP.COM



# UNSAFE AT ANY SPEED

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

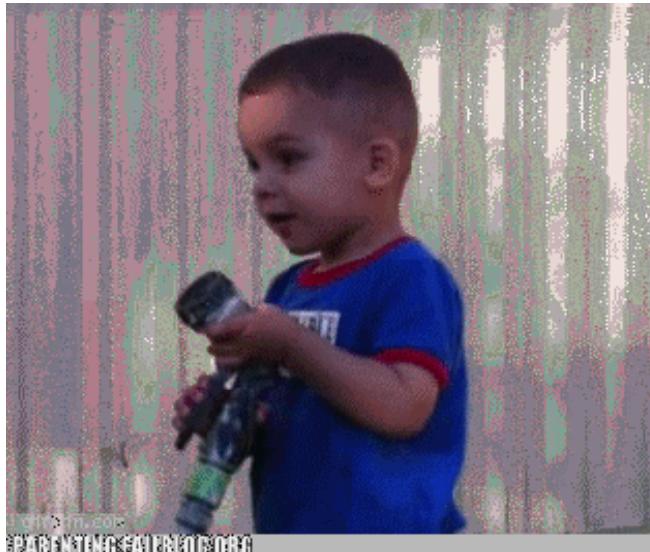


**stroom prosesing**



## Let's Face it, Streams<sup>1</sup> were terrible...

- The pause method didn't
- The 'data' event started immediately, ready or not!
- Can't just consume a specified number of bytes
- Pause and resume were impossible to get right...





## Streams2 Electric Boogaloo

- Landed in 0.9.4
- Now supported “Object Mode”
- Flowing mode versus non-flowing mode
- Introduced the following Streams classes:
  - Readable
  - Writable
  - Duplex
  - Passthrough

## Streams 33 1/3

- Landed in 0.11.2
- Adds cork/uncork/\_writev



# 1994



# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



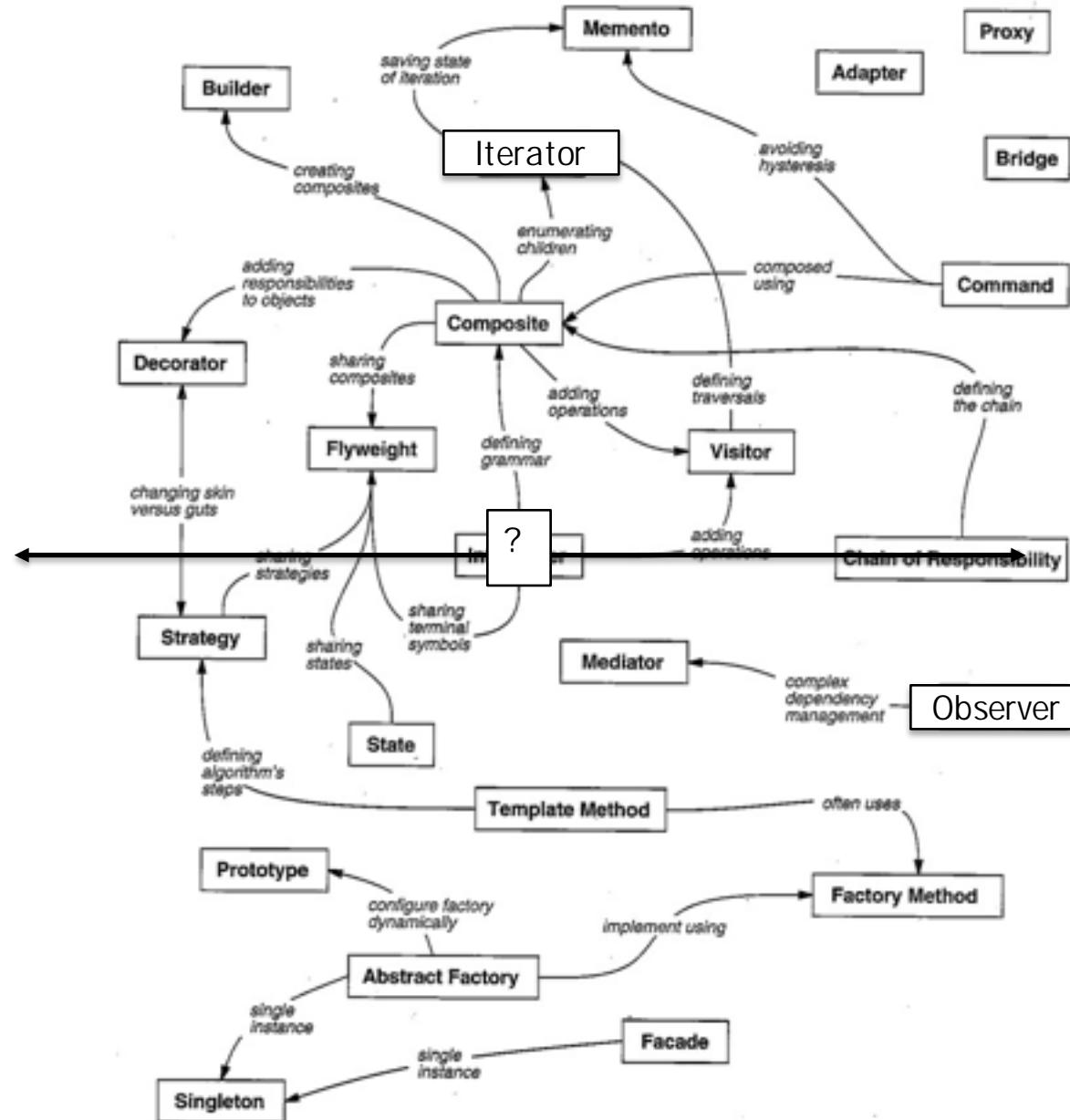
Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



Cover

Fore



Design Pattern Relationships

# Iterator Pattern with ES2015

```
> let 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...

[{x: 23, y: 44}, {x:27, y:55}, {x:27, y:55}]

... and an Event?

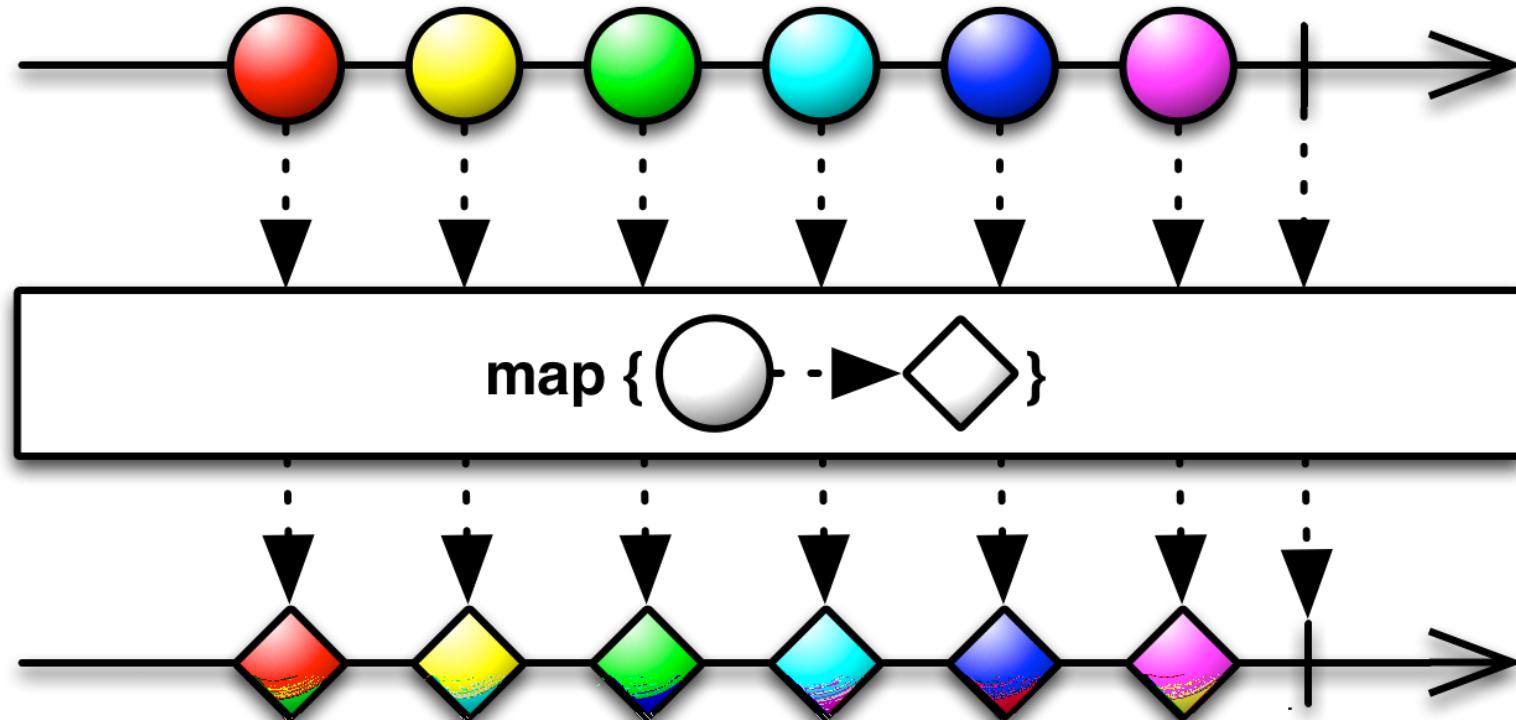


Events and Arrays are *both*  
collections.

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

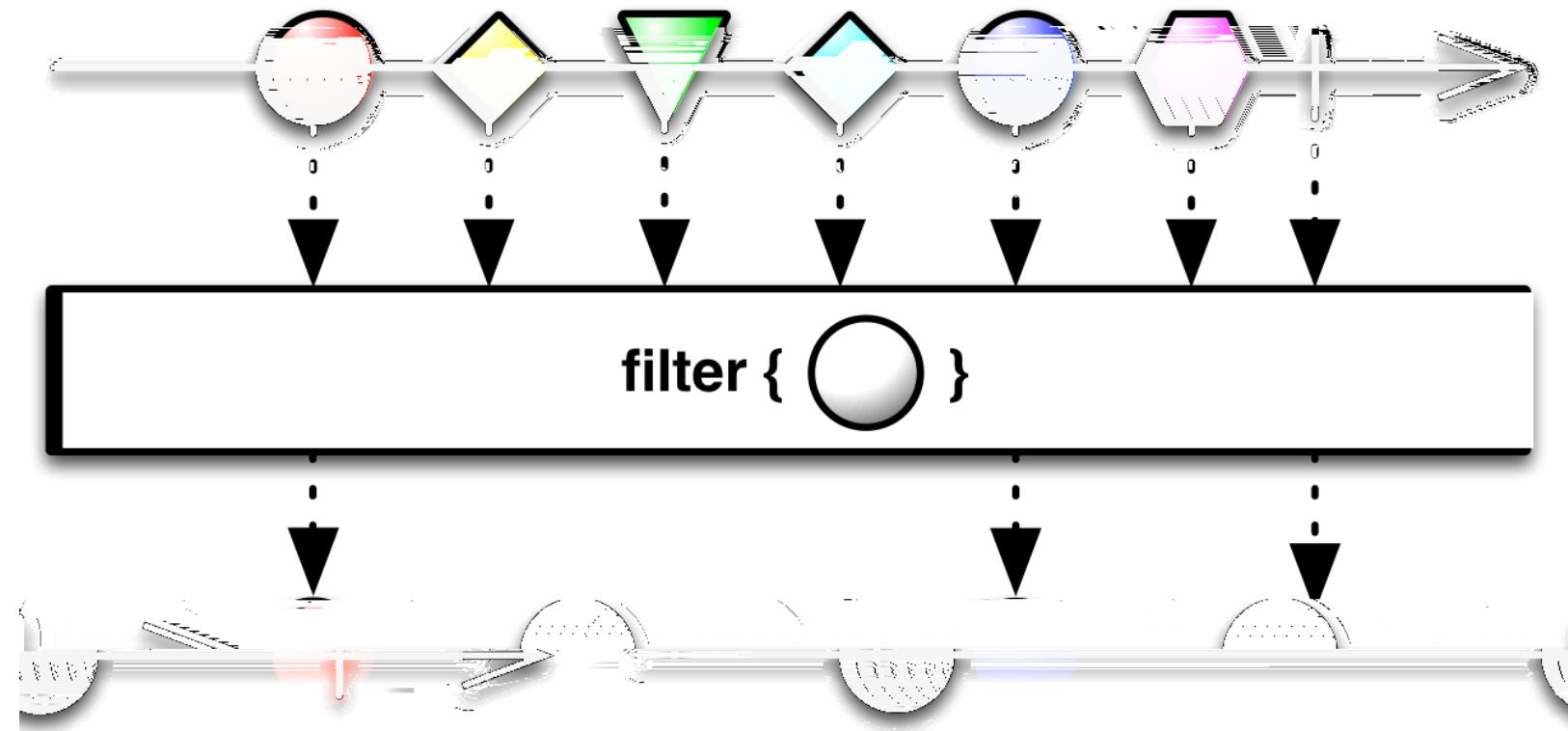
# map()

Transform the items emitted by an Collection by applying a function to each of them



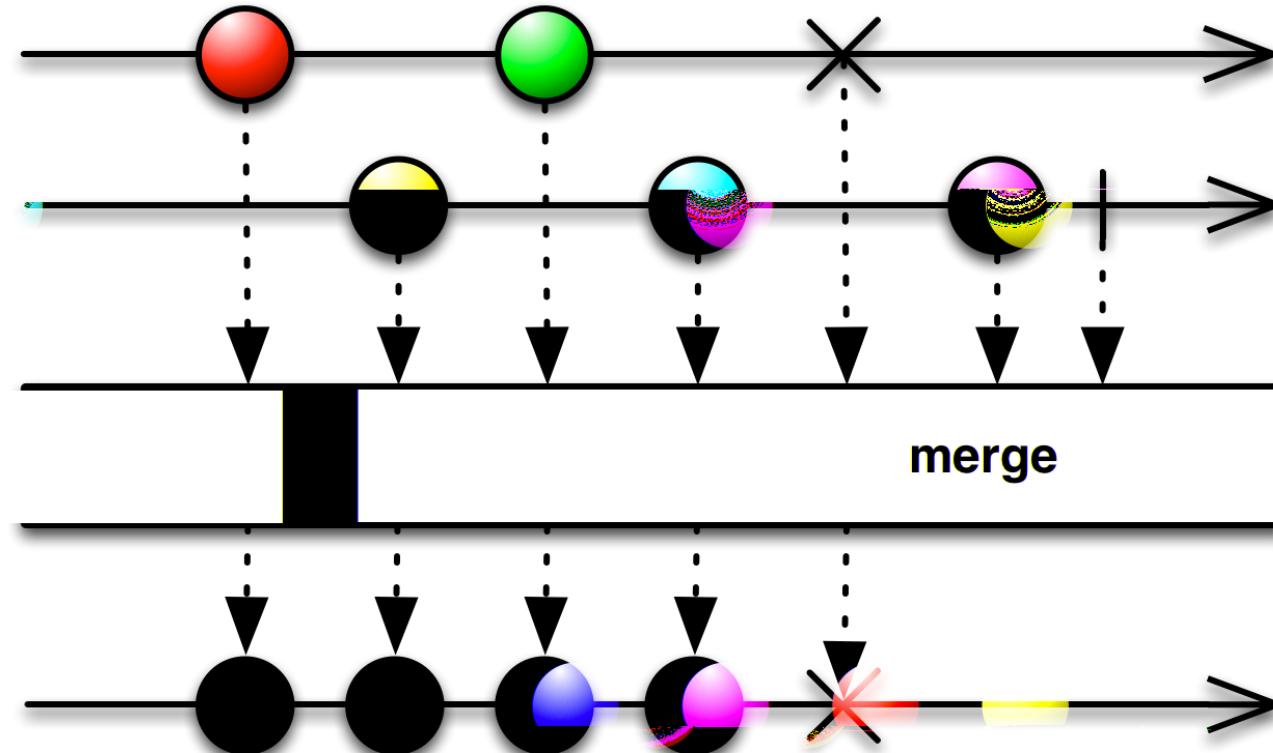
# filter()

Filter items emitted by a Collection



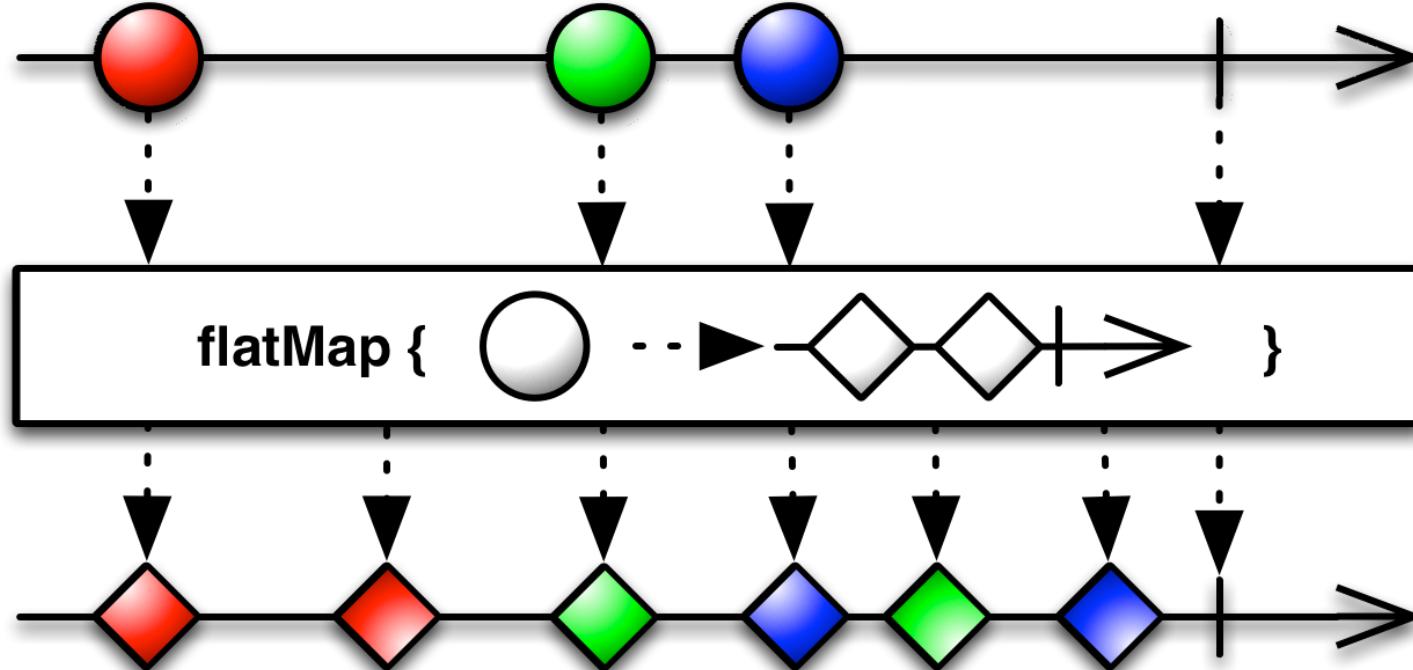
# mergeAll()

combine multiple Collections into one by merging their emissions



# flatMap()

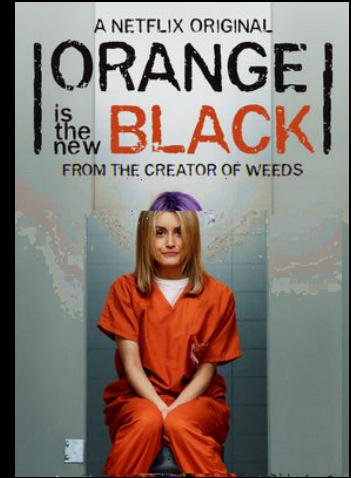
Transform the items emitted by a Collection into Collections, then flatten this into a single Collection



# Top-rated Movies Collection

```
let getTopRatedFilms = (user) => {
  user.videoLists
    .map((videoList) =>
      videoList.videos
        .filter((v) => v.rating === 5)
    ).mergeAll();
};

getTopRatedFilms(me)
  .forEach(displayMovie);
```

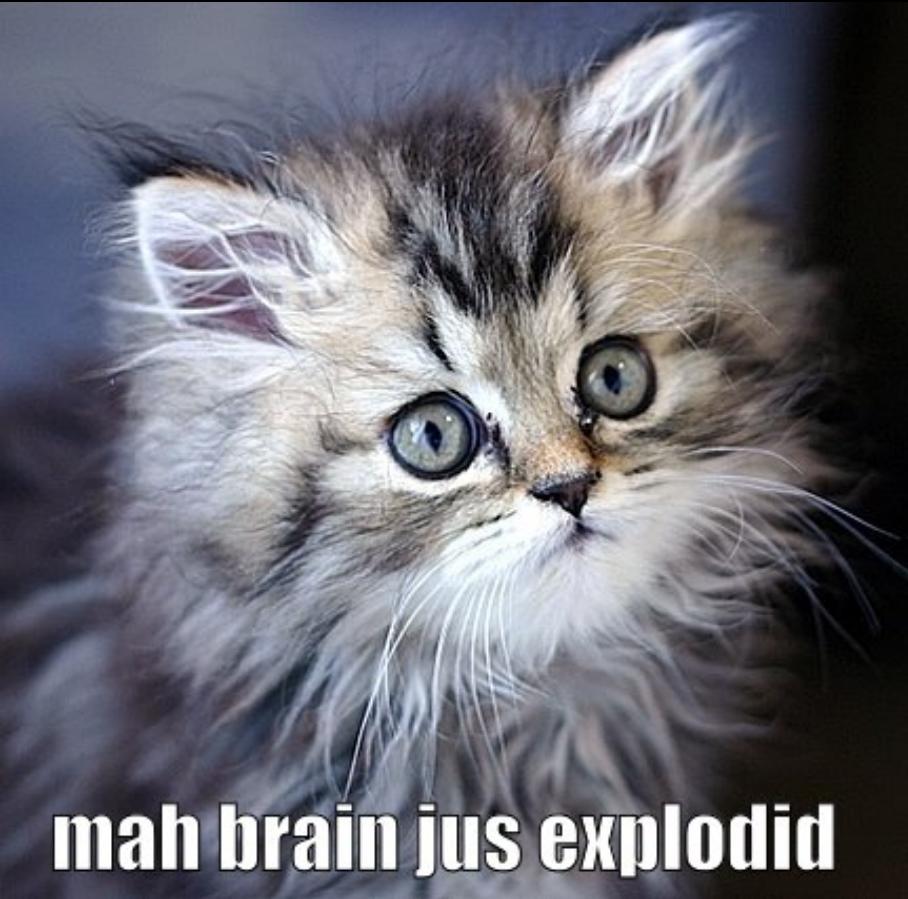


# Top-rated Movies Collection

```
let getTopRatedFilms = (user) =>  
  user.videoLists  
    .flatMap((videoList) =>  
      videoList.videos  
        .filter((v) => v.rating === 5)  
    )  
};  
  
getTopRatedFilms(me)  
  .forEach(displayMovie);
```



# What if I told you...



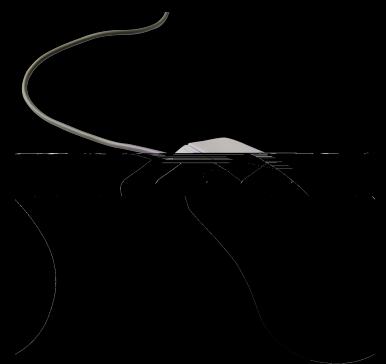
...that you could create a drag event...  
...with the almost the **same code**

**mah brain jus explodid**

# Mouse Drags Collection

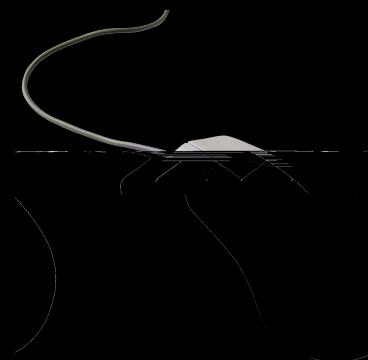
```
let getElementDrags = (elmt) =>  
  dom.mousedown(elmt)  
    .map((md) =>  
      dommousemove(document)  
        .filter .takeUntil(dommouseup(elmt))  
    ).mergeAll()  
};
```

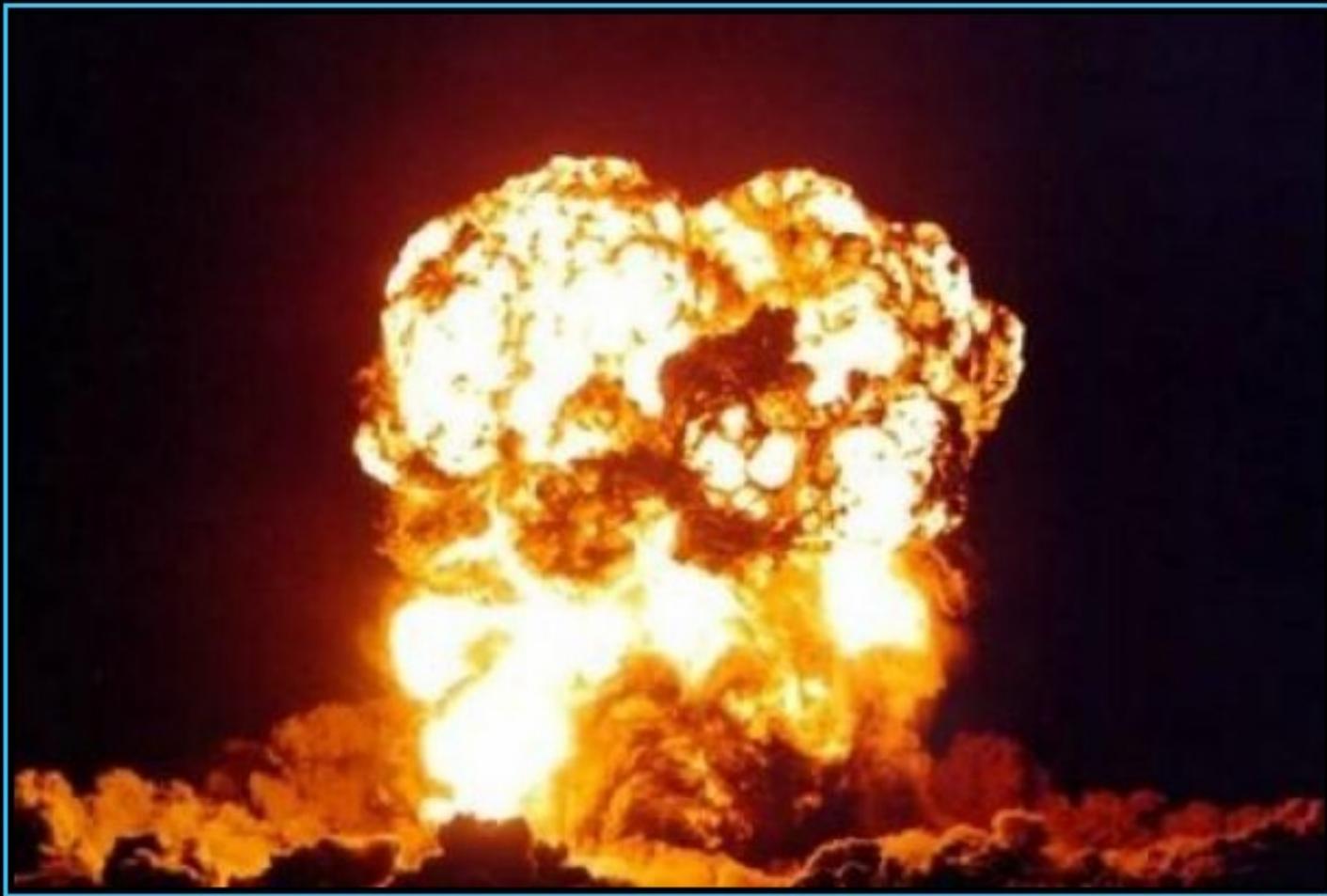
```
getElementDrags(image)  
.forEach(moveImage)
```



# Mouse Drags Collection

```
let getElementDrags = (elmt) =>  
  dom.mousedown(elmt)  
    .flatMap((md) =>  
      dommousemove(document)  
        .filter .takeUntil(dommouseup(elmt))  
    )  
};  
  
getElementDrags(image)  
.forEach(moveImage)
```





# PROTONIC REVERSAL

You crossed the streams, didn't you?



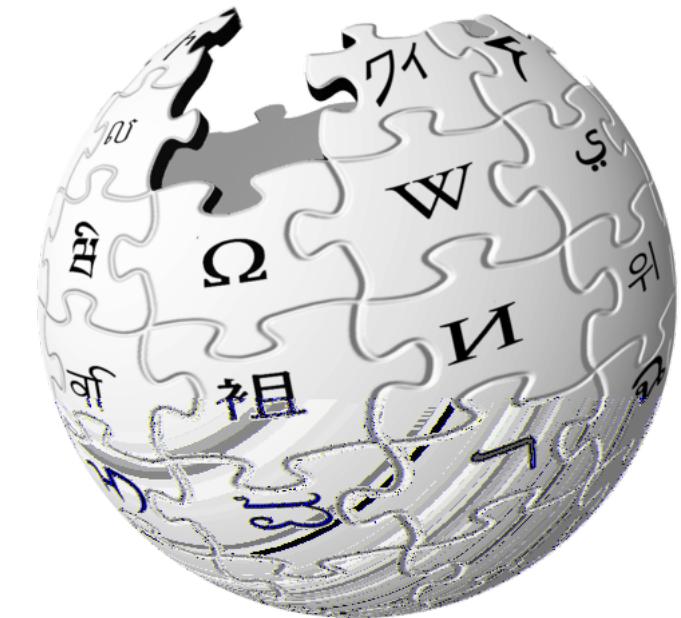
Everything is a stream

# First-Class Asynchronous Values



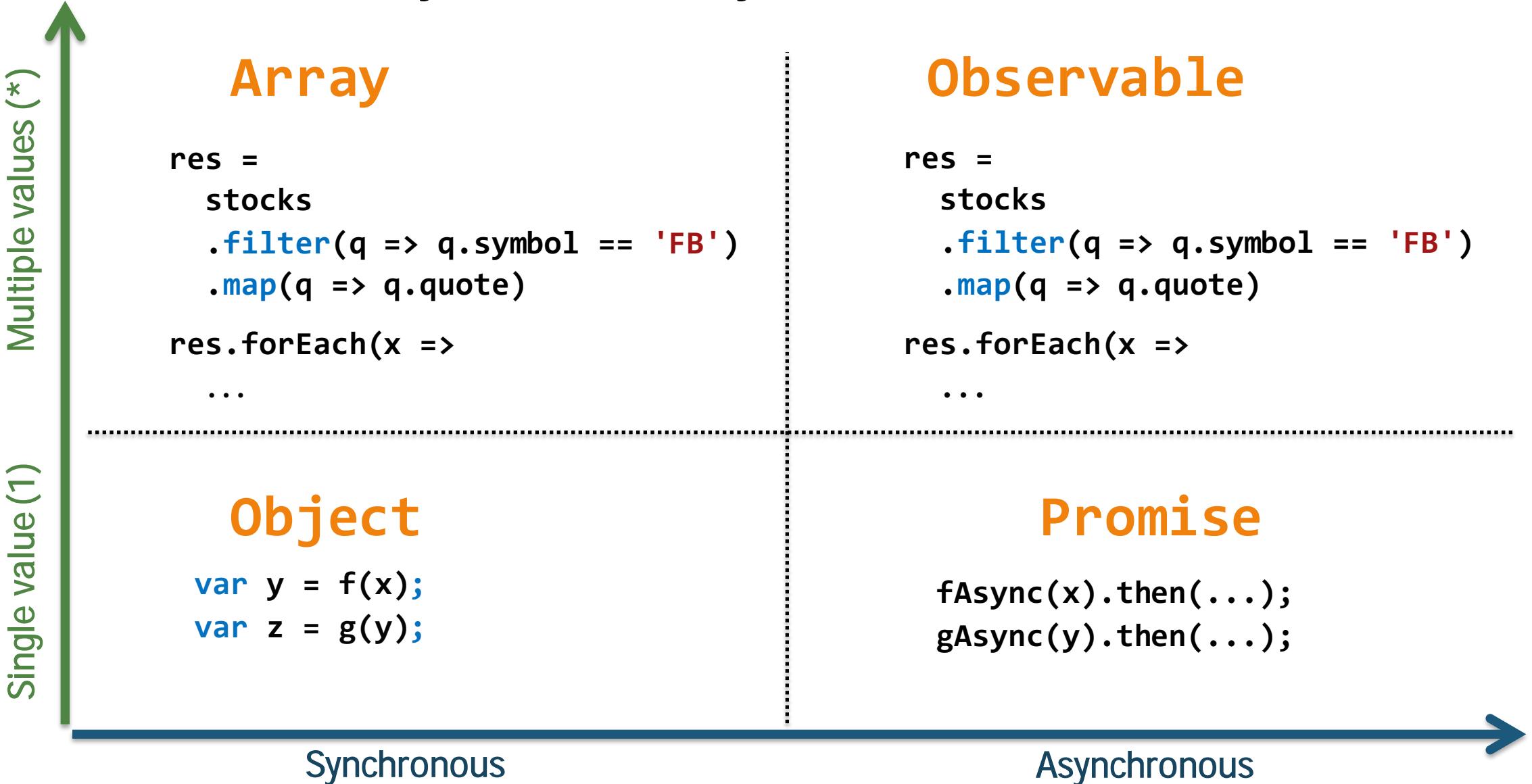
日本語  
Polski  
Русский

- 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



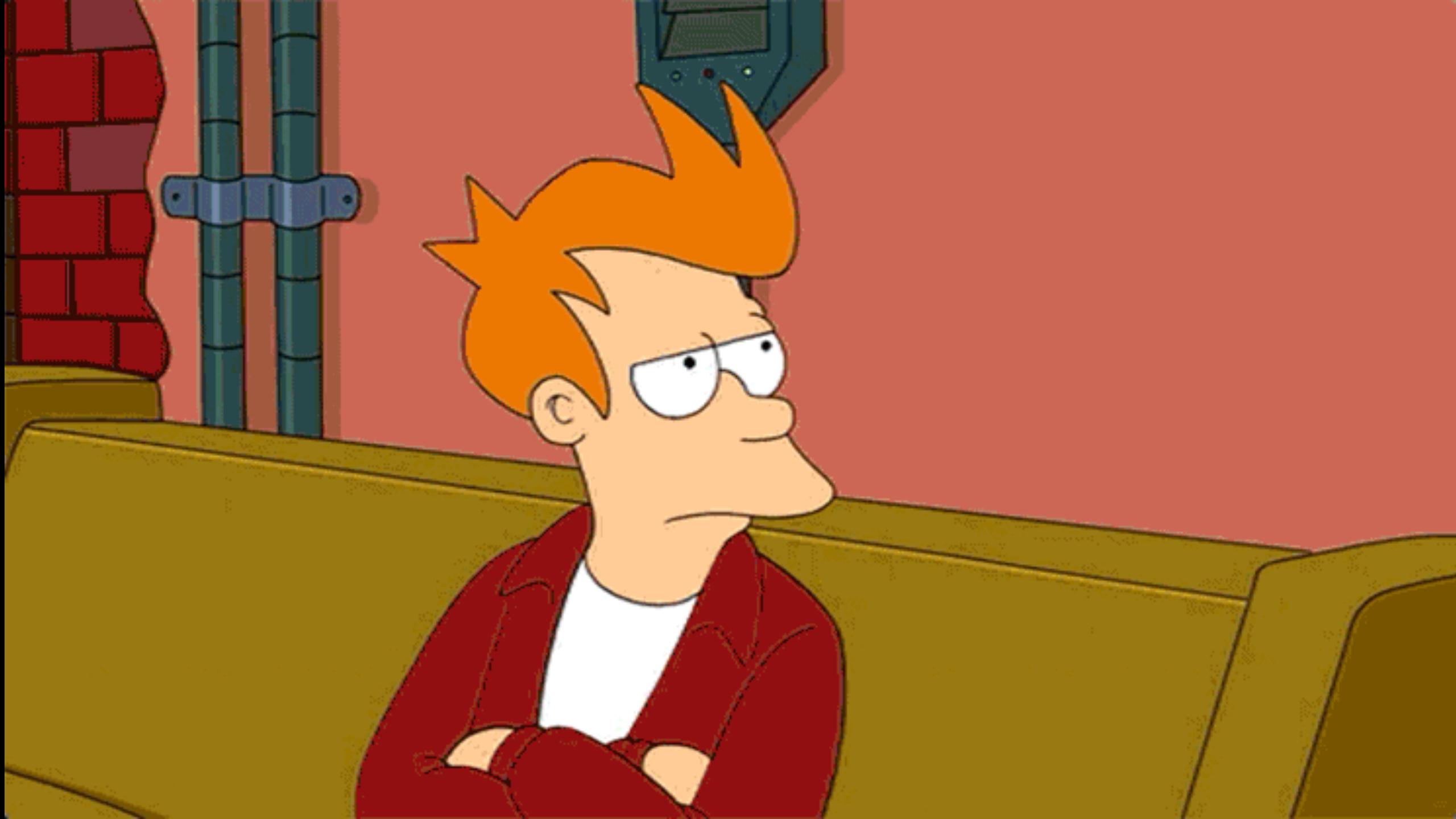
# What is Reactive Programming Anyhow?

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...

**A concept consisting of**

- Continuous Time
- Behaviors: Values over time
- Events: Discrete phenomena with a value and a time
- Compositional behavior for behavior and events

**What it is not**

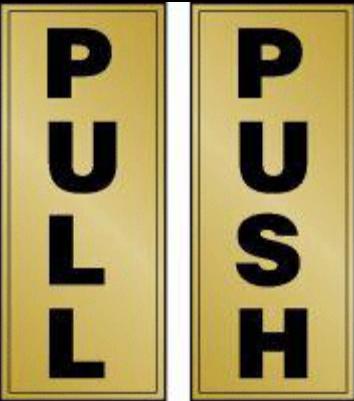
- High order functions on events like map, filter, reduce
- Most so-called FRP libraries out there...

# Call Us RP, 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.



```
let source = getStockData();

source
  .filter((quote) =>
    quote.price > 30;
)
  .map((quote) =>
    quote.price;
)
  .forEach((price) =>
    console.log('> $30: ' + price);
);
```

```
let source = getStockData();

source
  .filter((quote) =>
    quote.price > 30;
)
  .map((quote) =>
    quote.price;
)
  .forEach((price) =>
    console.log('> $30: ' + price);
);
```



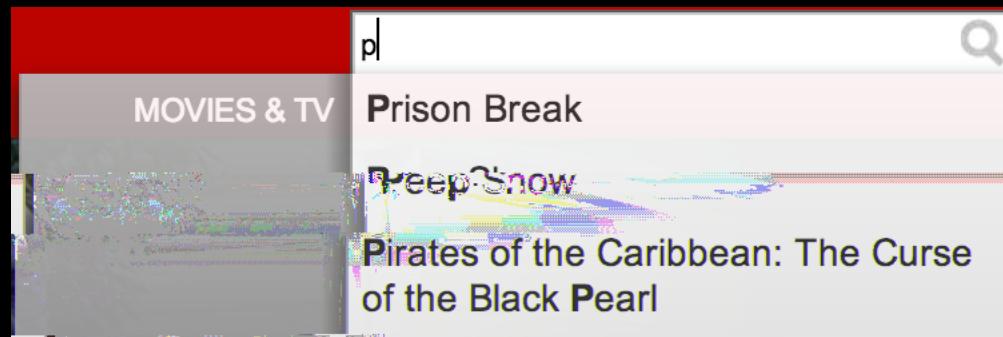
```
function fromEvent(element, event) {
  return Rx.Observable.create(observer => {

    function handler(e) {
      observer.onNext(e);
    }

    element.addEventListener(event, handler, false);

    return Rx.Disposable.create(() => {
      element.removeEventListener(event, handler, false);
    });
  }).share();
}
```

# Netflix Search



# Netflix Search with Observables

```
let data = dom.keyup(input)
    .map(() => input.value)
    .debounce(500)
    .distinctUntilChanged()
    .flatMapLatest(
        (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?

**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**



# Rx Grammar Police

onNext

\*

Zero or more values

E.g. events are  $\infty$  sequences

(onError

Calls can fail

onCompleted

) ?



# What exactly 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





# Observables - Querying UI Events

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

1

For each mouse down



# Observables - Querying UI Events

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

1

For each mouse down

2

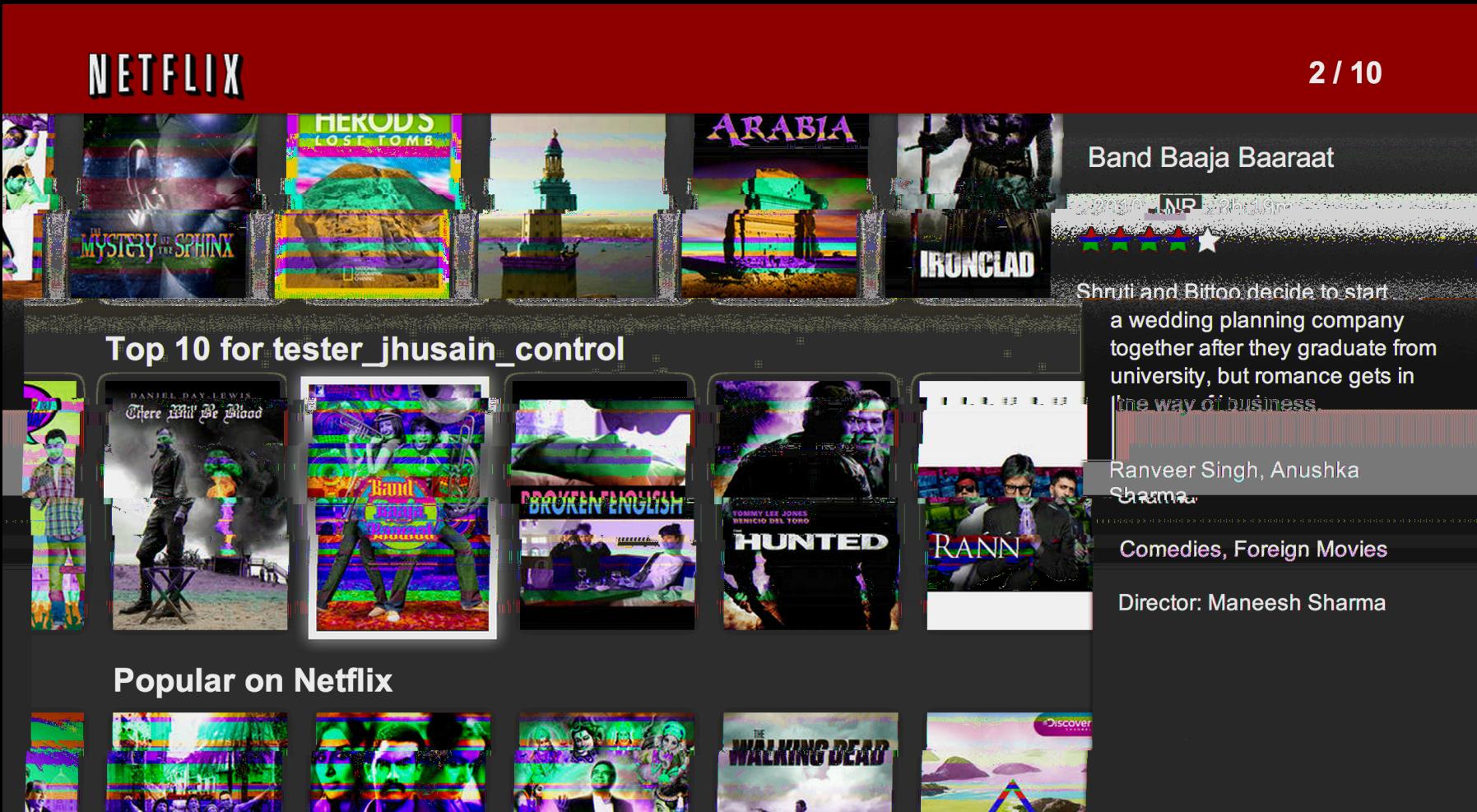
Take mouse moves

3

until mouse up

# Your Netflix Video Lists

## Netflix Row Update Polling



# Client: Polling for Row Updates

```
function getRowUpdates(row) {  
  let scrolls = Rx.Observable.fromEvent(document, 'scroll');  
  let rowVisibilities =  
    scrolls.debounce(50)  
      .map((scrollEvent) => row.isVisible(scrollEvent.offset))  
      .distinctUntilChanged()  
      .share();  
  
  let rowShows = rowVisibilities.filter((v) => v);  
  let rowHides = rowVisibilities.filter((v) => !v);  
  
  return rowShows  
    .flatMap(Rx.Observable.interval(10))  
    .flatMap(() => row.getRowData().takeUntil(rowHides))  
    .toArray();  
}
```

# Netflix Player



# Player Callback Hell

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



# Player With Observables

```
let authorizations =  
  player.init().flatMap(() =>  
    playAttempts.flatMap((movieId) =>  
      player.authorize(movieId)  
        .retry(3)  
        .takeUntil(cancels))  
    )  
  );  
let subscription = authorizations.forEach(  
  (license) => player.play(license),  
  (error) => showDialog('Sorry, can't play right now.'));
```



# RxSocketSubject

build passing

A simple web socket wrapper for RxJS

A more advanced

## Install

With bower:

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

```
bower install
```

With npm:

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

```
npm install -
```

## Goals

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:

The goals of this project I'm currently working on are to implement a WebSocket implementation that is compatible with the RxJS Observable API, and to provide a simple and easy-to-use interface for interacting with WebSockets.

```
let socket = RxSocketSubject.create(  
  [ 'ws://netflix.com/socket1',  
    'ws://netflix.com/socket2',  
    'ws://netflix.com/socket3']);  
  
socket.forEach((e) => {  
  if (e.data === 'end') {  
    socket.onCompleted();  
  }  
  if (e.data === 'bad data') {  
    socket.onError(new Error('bad data'));  
  }  
});  
  
socket.onNext('some data');
```

# The Final Countdown...



```
try {  
  let resource = getData();  
  for (let data of resource) {  
    processItem(data);  
  }  
} catch (e) {  
  throw e;  
} finally {  
  resource && resource.dispose();  
}  
  
var d = getData();  
d.retry(3)  
.catch(defaultData)  
.finally(() => d.cleanup())  
.forEach(data =>  
  processItem(data));
```

# 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

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

Optional time

# 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

**Yes, Observables can have backpressure**

- Can be lossy (pausable, sample, throttle)
- Can be lossless (buffer, pausableBuffered, controlled)

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

```
let controlled = chattyObservable.controlled();
controlled.request(10);
```



# Async/Await

## Coming to a JavaScript Engine Near You!

- Adds `async` and `await` keywords for Promises
- Accepted into Stage 1 of ECMAScript 7 in January 2014

```
async function chainAnimationsAsync(elem, animations) {  
    let ret = null;  
    try {  
        for (let anim of animations) {  
            ret = await anim(elem);  
        }  
    } catch (e) { /* ignore and keep going */ }  
    return ret;  
}
```

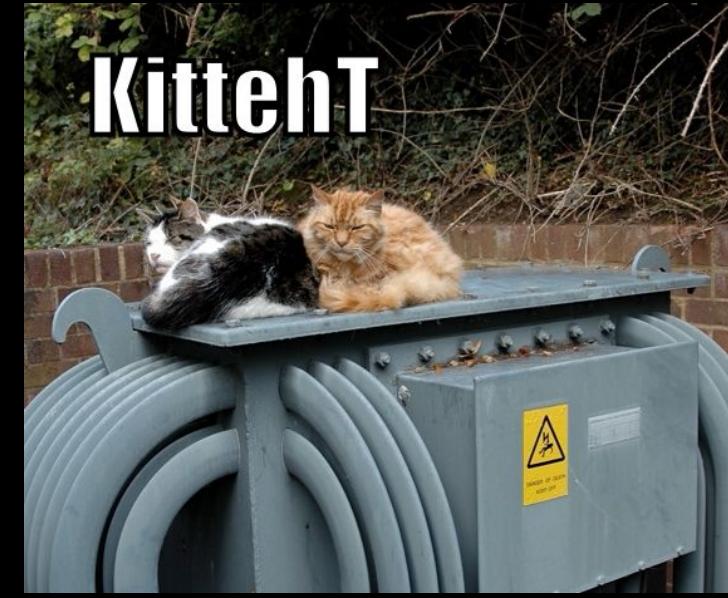
<https://github.com/lukehoban/ecmascript-asyncawait>

# Async/Await with Observables and Generators...

## RxJS and Generators

- Adds `async` /`await` capabilities to single value Observables
- Available in any runtime that has Generators

```
Rx.spawn(function* () {  
  var result = yield get('http://jsconf.uy')  
    .retry(3)  
    .catch(cachedVersion)  
    .finally(doCleanup);  
  
  console.log(result);  
}());
```



# Async Generators

## ES7 and Beyond!

- First class events in the JavaScript runtime
- Proposed in June 2014 at TC39

```
async function* getDrags(element) {  
  for (let mousedown on element.mousedown) {  
    for (let mousemove on  
      documentmousemove.takeUntil(documentmouseup)) {  
      yield mousemove;  
    }  
  }  
}
```

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

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

shorter, more self-descriptive, and more durable. The five functions hold the key to having all the tools you need to easily handle events and AJAX requests. In short, these five 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 submit your code it will be automatically compiled 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 at the end.
11
12 // The macro keyword is used to create and name new macros.
13 macro id {
14     rule {
15         // Just return the macro name - macro
16         // (1) a 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 );
26
27 // Note that a single token to sweet.js includes matched
28 // delimiters not just numbers and identifiers. For example,
29 // if you write 1, 2, 3; it will be converted to
30 // 1, 2, 3;
31
32 // sweet.js imports this generated code so don't
33 // macros from unintentionally binding or capturing variables
34 // that weren't supposed to. This is called hygiene and to enforce
35 // it sweet.js must carefully rename all variable names.
36
37 var x;
38
39 var foo = 100;
40 var bar = 200;
41 var tmp = 'my other temporary variable';
42 var tmp$2 = bar;
43 bar = foo;
44 foo = tmp$2;
```

