

Mother of All Monad Demos

Bodil Stokke

@bodil

Matthew Podwysocki

@mattpodwysocki

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



**trapd in Monad tutorl
plz help**

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/webrebels-2015



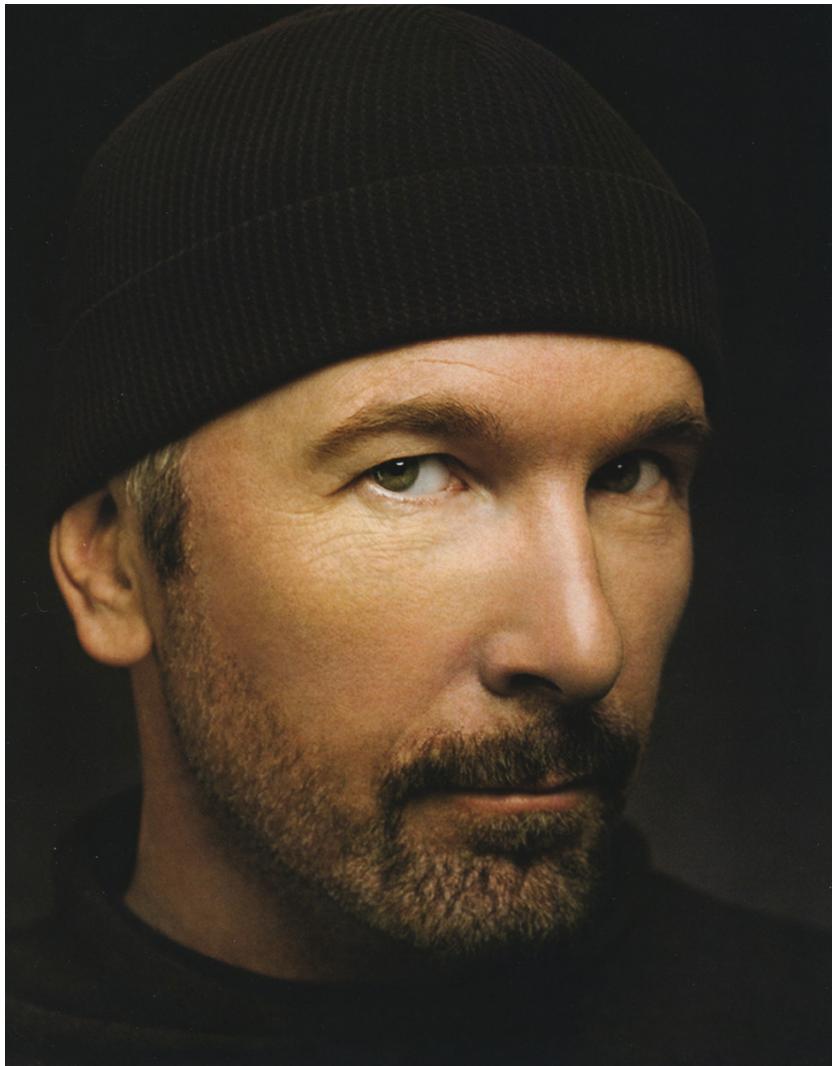
**OR:
HOW I LEARNED TO STOP WORRYING ABOUT
ASYNCHRONOUS PROGRAMMING AND LOVE THE
OBSERVABLE**

**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**



**Principal SDE
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki**

MICROSOFT



**YOU AND ME MAKES
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 within 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**



Asynchronous Nightmares

- **Memory Leaks**
- **Race Conditions**
- **Callback Hell**
- **Complex State Machines**
- **Disjointed Error Handling**



2014

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”**



**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

then

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

Breaking the Promise...

then

```
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

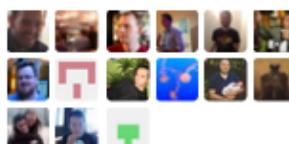
[jakearchibald](#)

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 \(comment\)](#)

Labels

3.0.0

Milestone

No milestone

Assignee

No one assigned

Notifications

Subscribe

You're not receiving notifications from this discussion

The Final Countdown...

then

```
var resource;
try {
  var resource = getResource();
} catch (e) {
  throw e;
} finally {
  resource && resource.dispose();
}

getresource()
  .success(
    function (resource) {
      })
  .catch(
    function (err) {
      });
}

// How do I clean up my resource?
```

then



MAKE GIFS AT GIFOUP.COM



UNSAFE AT ANY SPEED

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

Promises how I would do them

1. No over polymorphic `.then`

Four methods instead: `.map(a -> b)` , `.mapError(a -> b)` , `.flatMap(a -> Promise)` ,
`.flatMapError(a -> Promise)` .

2. No exceptions catching

Make it up to the developer. If it needed (which is very rare), one can do it manually:

```
foo.flatMap(x => {
  try {
    return Promise.resolved(bar(x));
  } catch (e) {
    return Promise.rejected(e);
  }
});
```

3. Ability to unsubscribe

<https://gist.github.com/rpominov/15414fdb166887fcac1b>



stroom prosesing



Let's Face it, Streams¹ 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

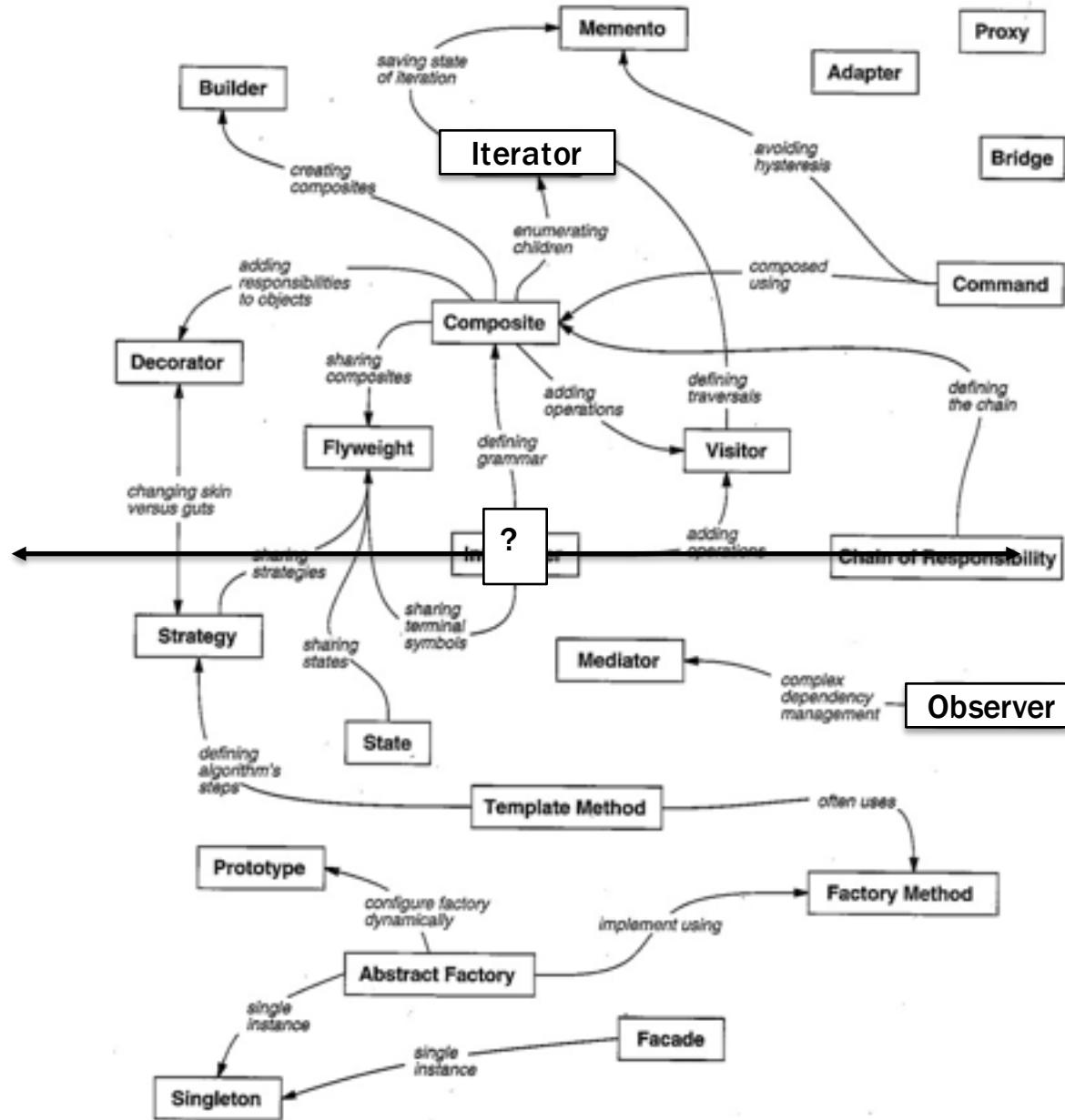


Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES





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}]



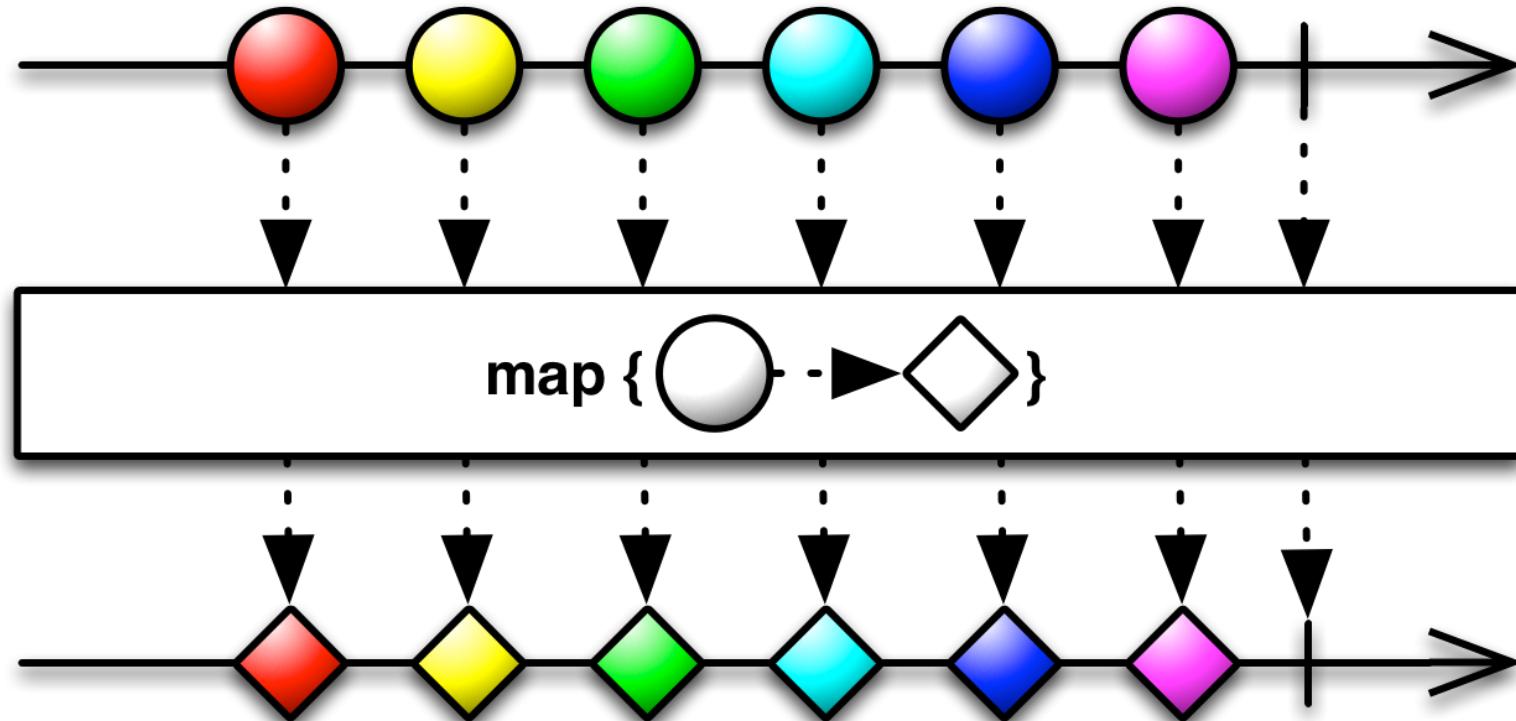
... and an Event?

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

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

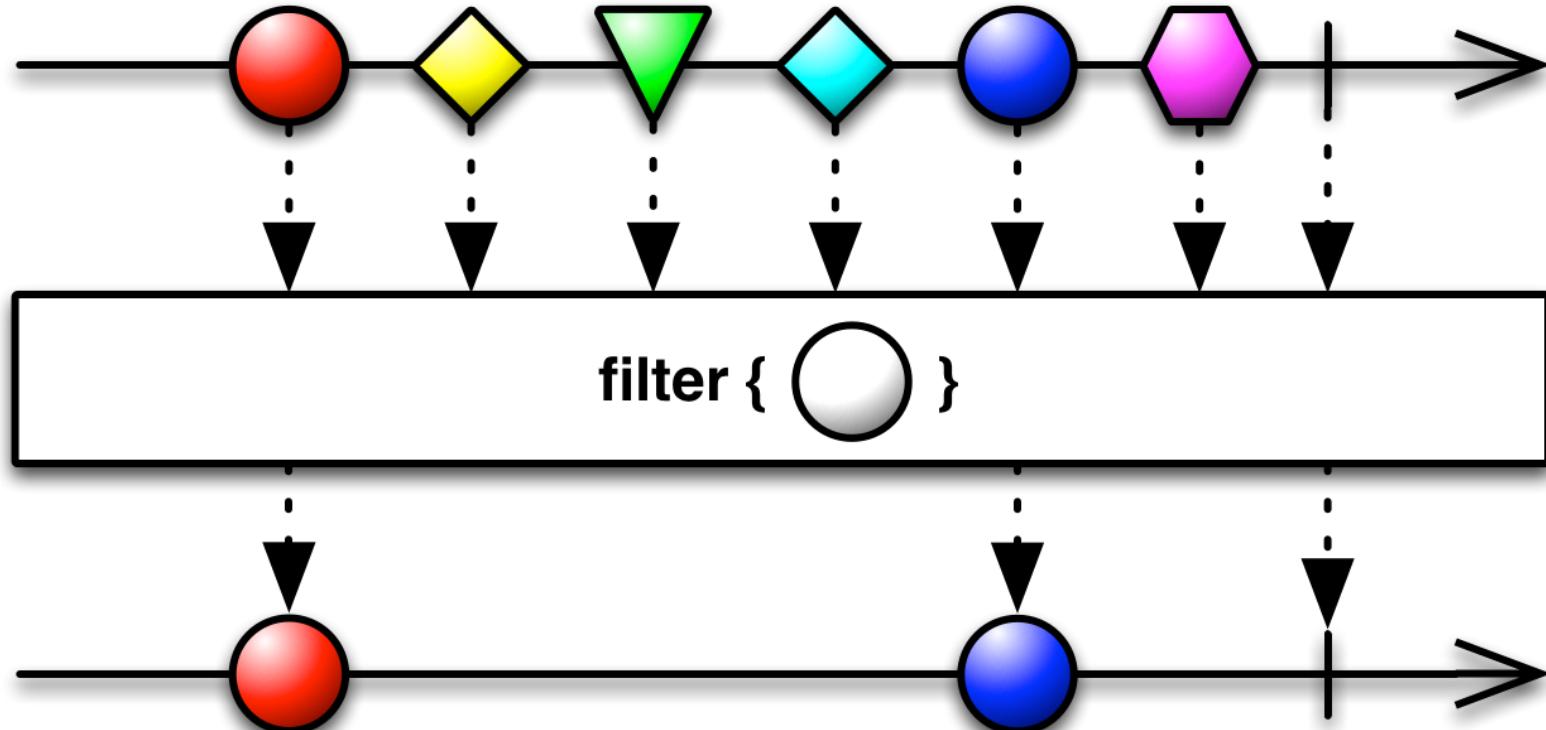
map()

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



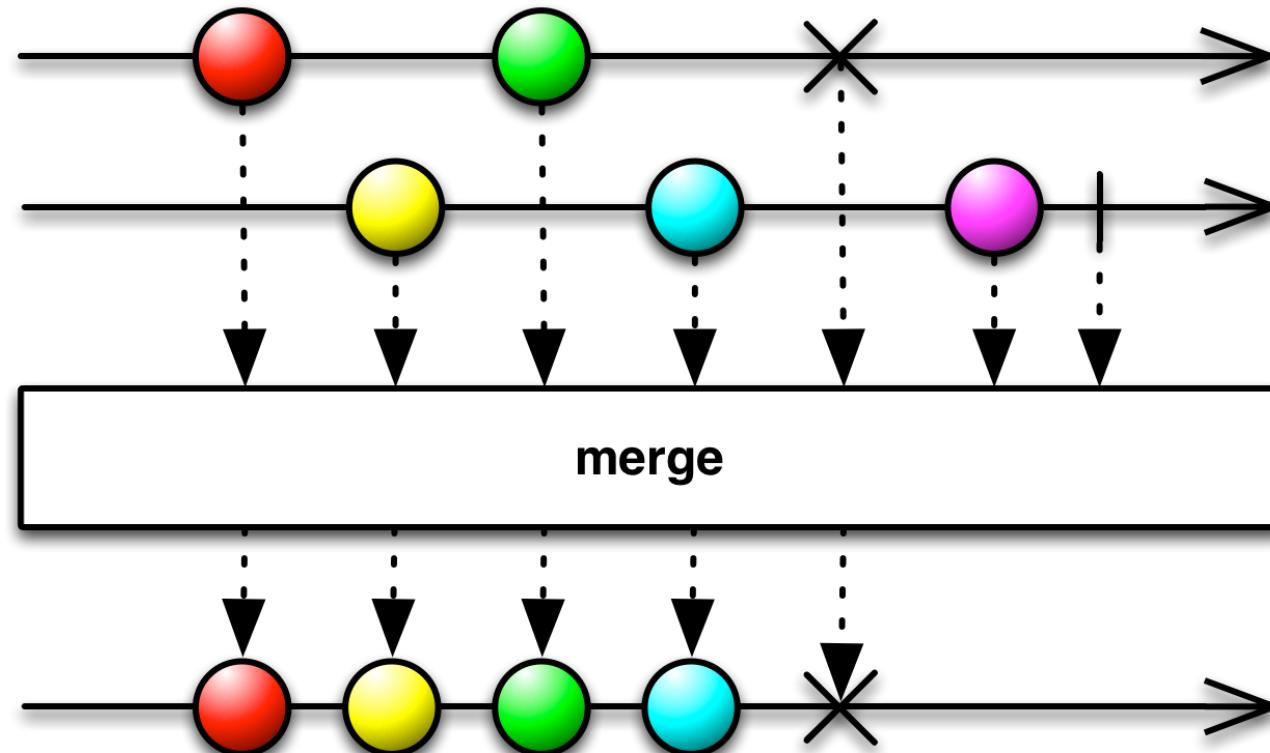
filter()

Filter items emitted by a Collection



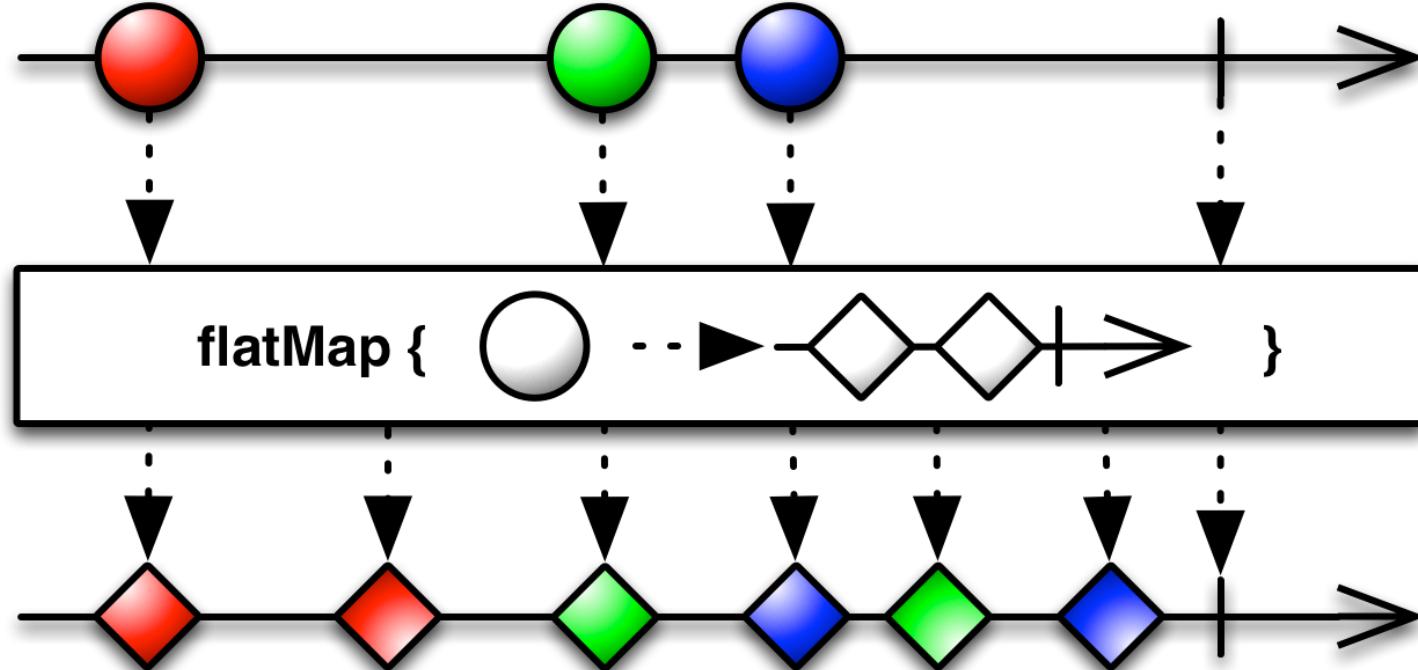
mergeAll()

combine multiple Collections into one by merging their emissions



flatMap()

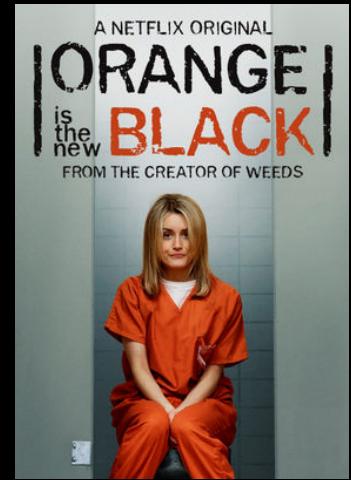
Transform the items emitted by a Collection into Collections, then flatten this into a single Collection



Top-rated Movies Collection

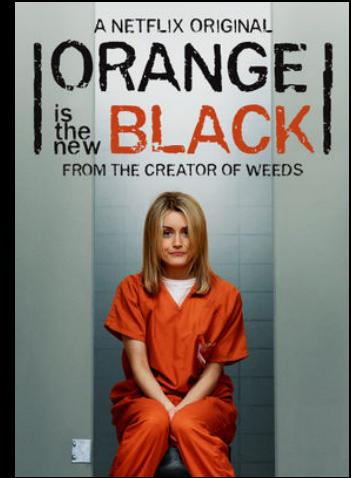
```
let getTopRatedFilms = (user) => {
  user.videoLists
    .map((videoList) =>
      videoList.videos
        .filter((v) => v.rating === 5)
    ).mergeAll();
};

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

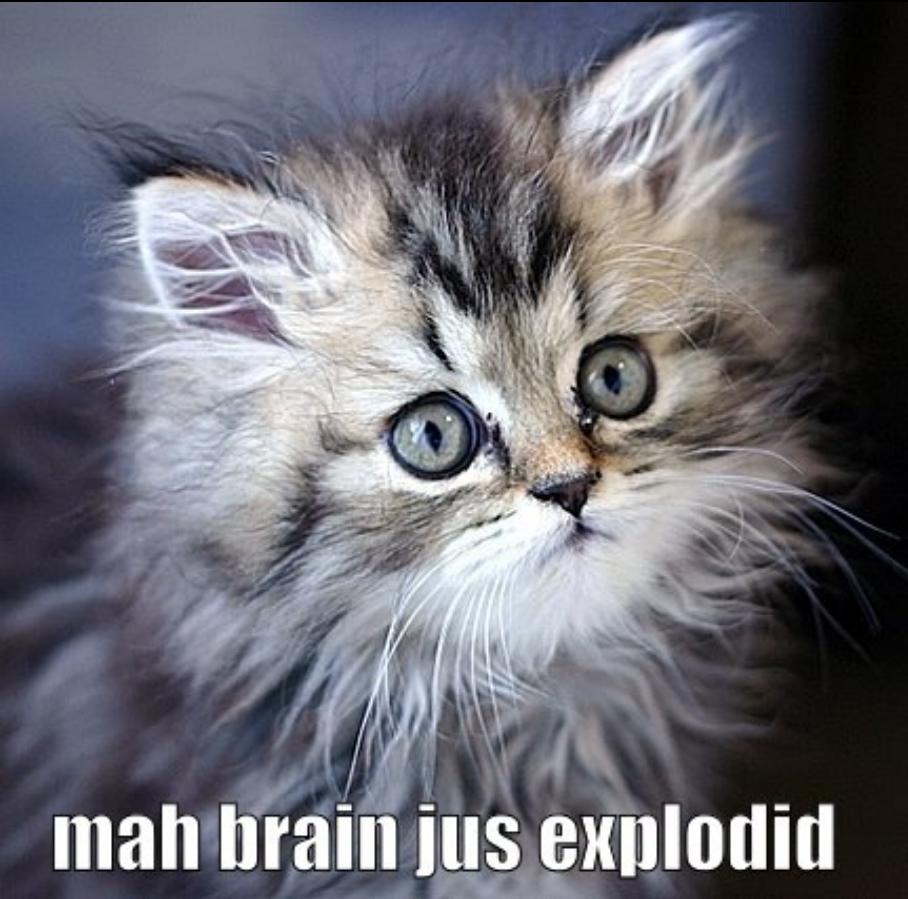


Top-rated Movies Collection

```
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()
};


```

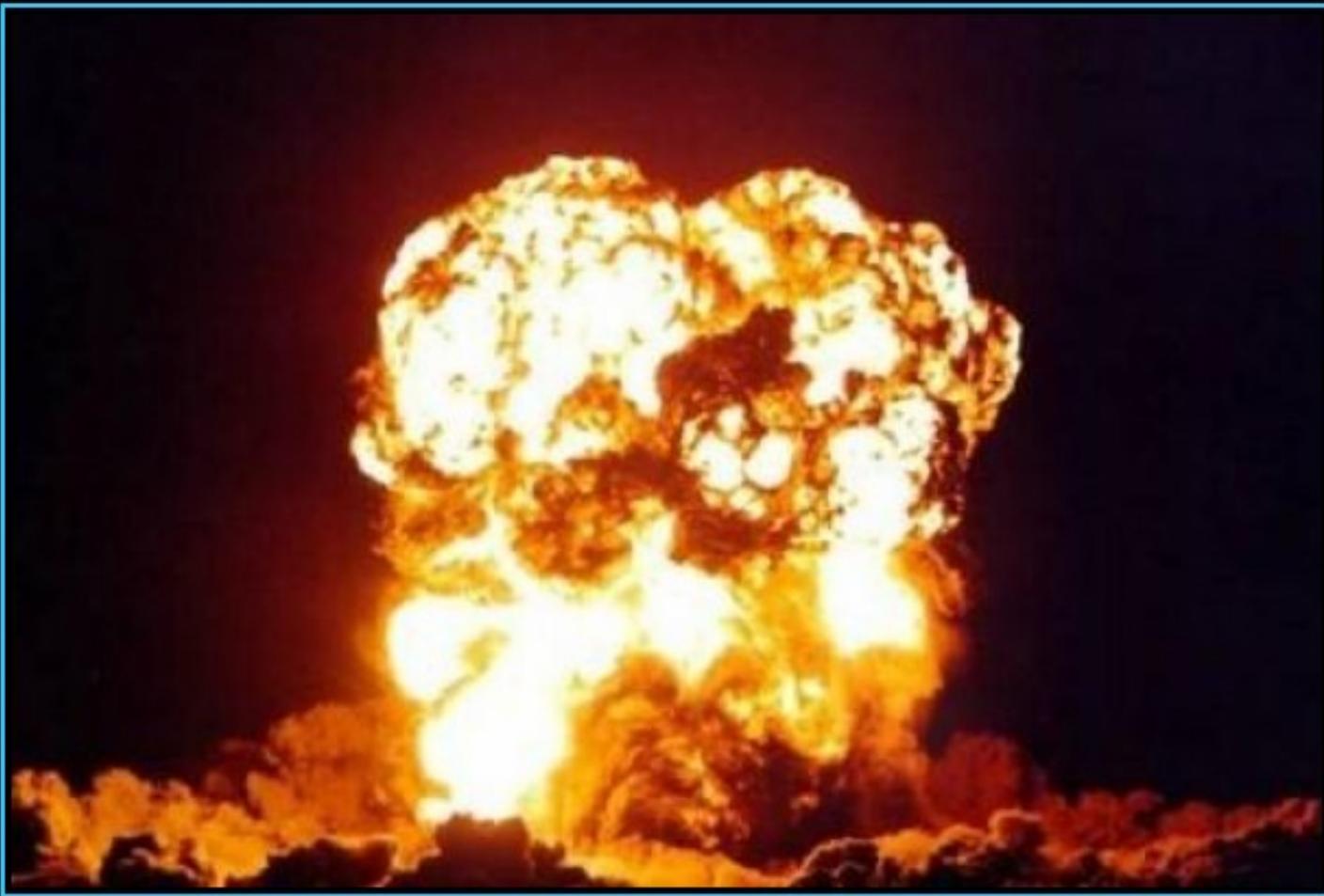
```
getElementDrags(image)
  .forEach(moveImage)
```



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

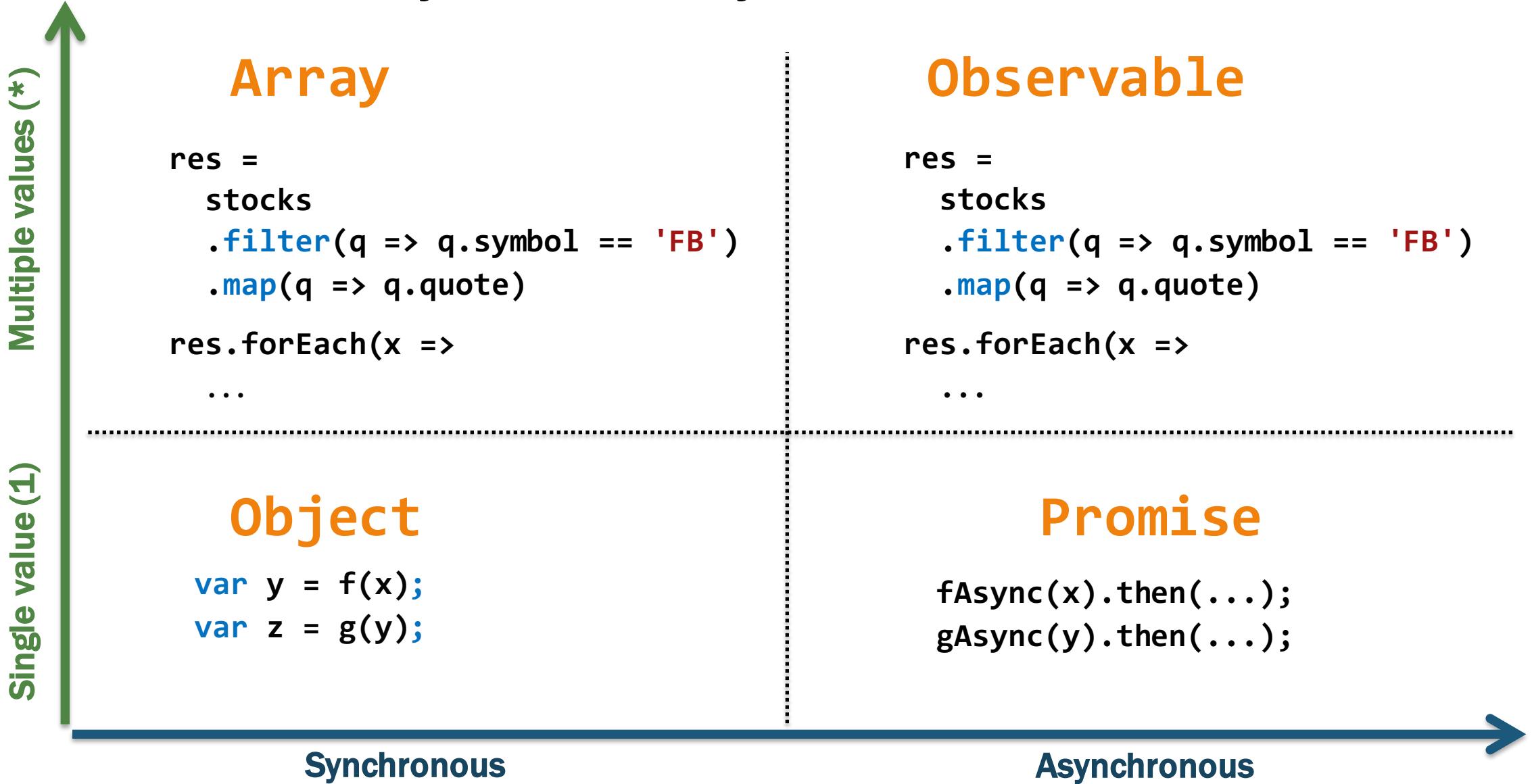
An object is **first-class** when it:^{[4][5]}

- 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)



WIKIPEDIA
The Free Encyclopedia

The General Theory of Reactivity



What is Reactive Programming Anyhow?

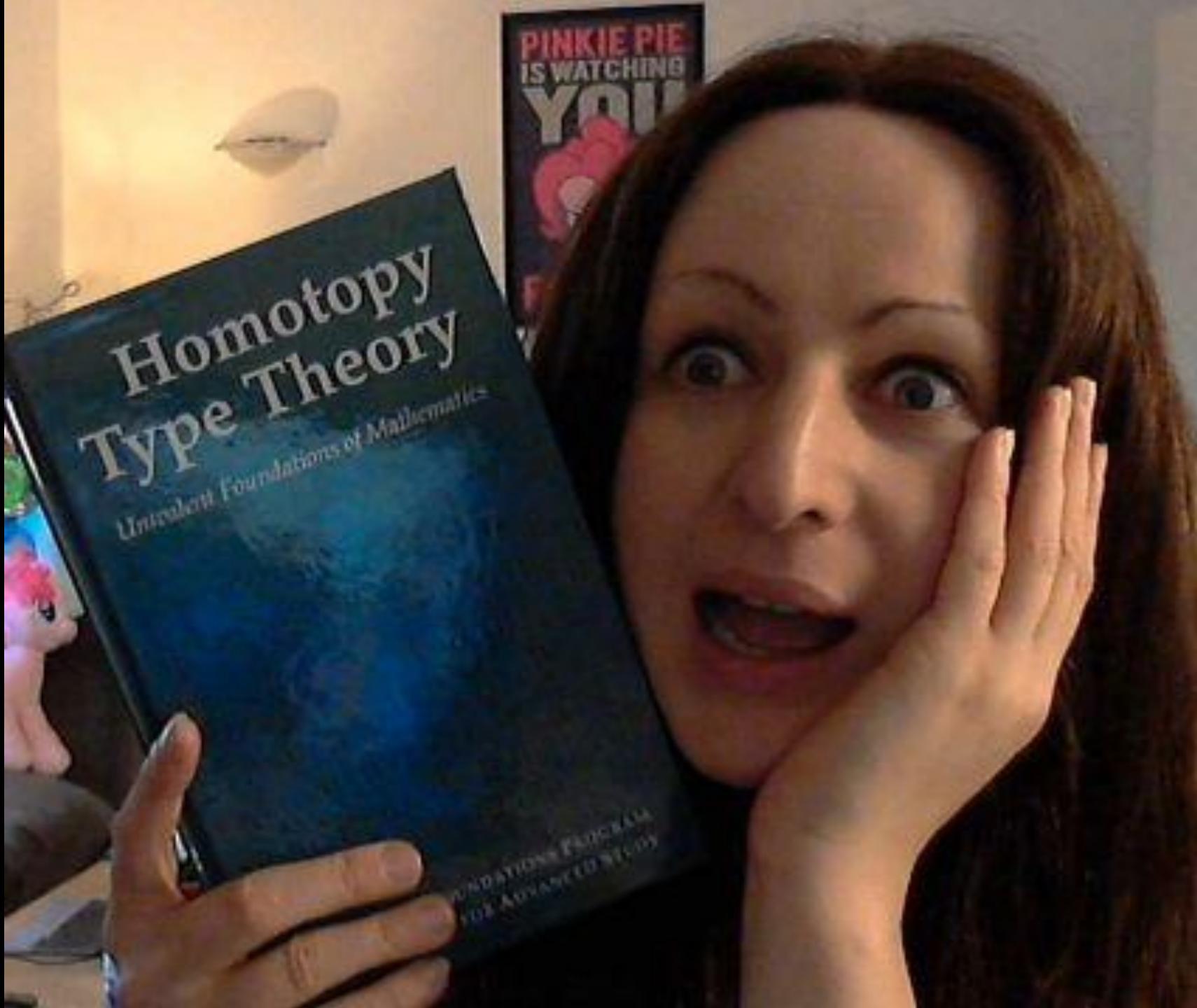
Merriam-Webster defines reactive as “*readily responsive to a stimulus*”, i.e. its components are “active” and always ready to receive events.

Wanna really know what Reactive Programming Is?

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

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





ComputerWissenschaftAkademischesPapierPhobie

Functional Reactive Programming (FRP) is...

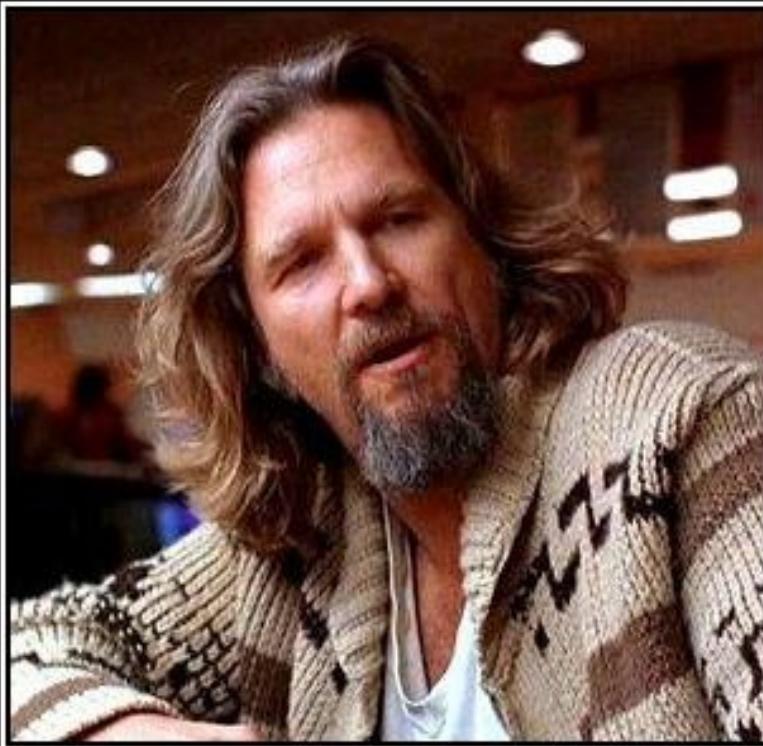
A concept consisting of

- Continuous Time**
- Behaviors: Values over time**
- Events: Discrete phenomena with a value and a time**
- Denotational Semantics**

type Behavior a

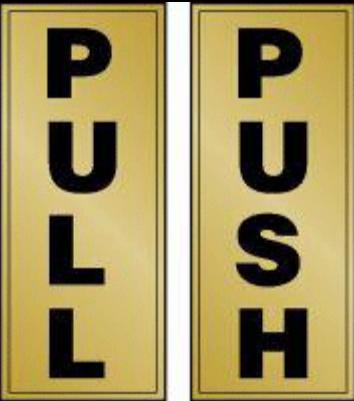
$\mu = \text{Behavior } a \rightarrow (\mathbb{R} \rightarrow a)$

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);
);
```



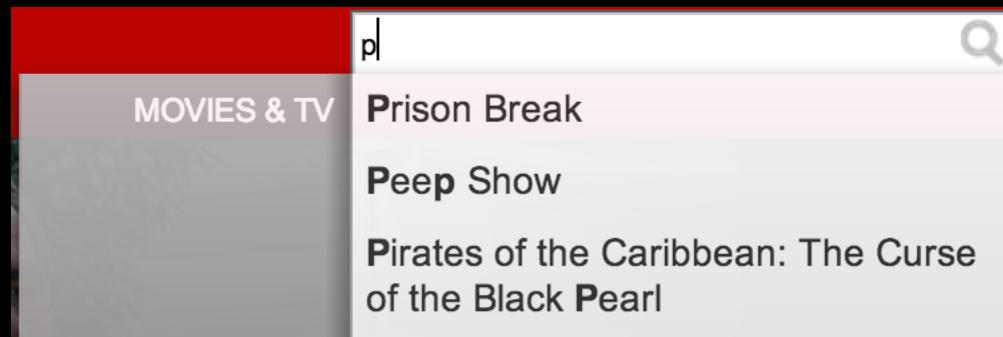
```
function fromEvent(element, event) {
  return Rx.Observable.create(observer => {

    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)
    );
```

```
let subscription = data.forEach((data) => {
    // Bind data 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

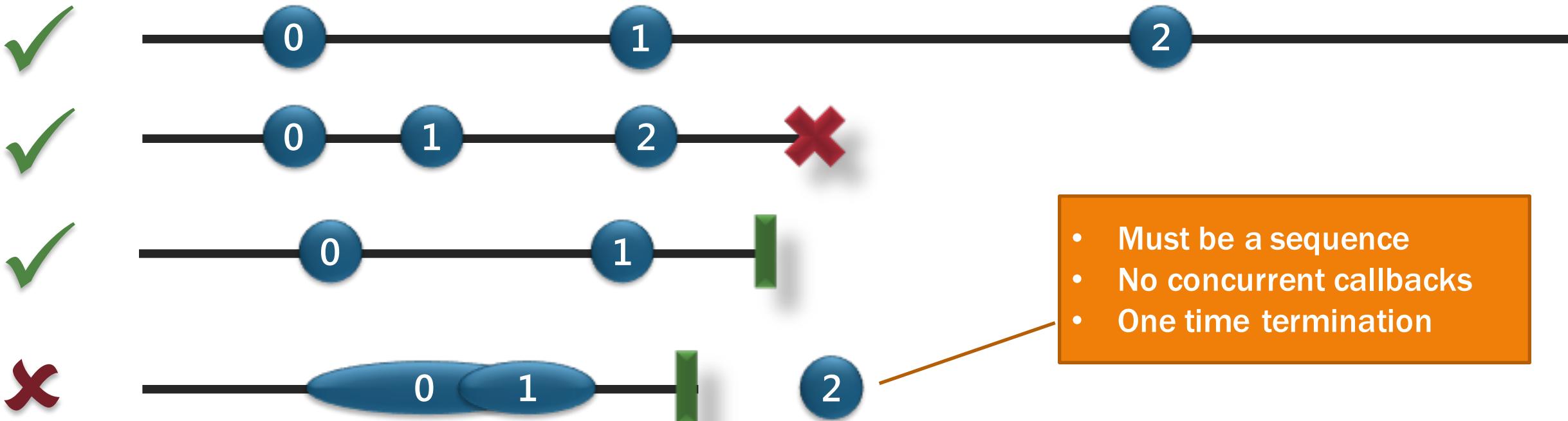
E.g. events are ∞ sequences

(onError 

Calls can fail

onCompleted ) ?

Resource management
Sequencing



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



flip concatMap



An Eye Chart of Operators

amb	if	when	count	extend	jorSort
case	interval	while	debounce	filter	jorSortUntil
catch	just	zip	debounceWithSelector	finally	last
combineLatest	merge	zipArray	defaultIfEmpty	find	lastOrDefault
concat	mergeDelayError	and	delay	findIndex	let
create	never	asObservable	delaySubscription	first	manySelect
defer	of	average	delayWithSelector	firstOrDefault	map
empty	ofWithScheduler	buffer	dematerialize	flatMap	max
for	onErrorResumeNext	bufferWithCount	distinct	flatMapObserver	maxBy
forkJoin	pairs	bufferWithTime	distinctUntilChanged	flatMapLatest	merge
from	range	bufferWithTimeOrCount	do	foreach	mergeAll
fromCallback	repeat	combineLatest	doOnNext	forkJoin	min
fromEvent	return	concat	doOnError	groupBy	minBy
fromEventPattern	start	concatAll	doOnCompleted	groupByUntil	multicast
fromNodeCallback	startAsync	concatMap	doWhile	groupJoin	observeOn
fromPromise	throw	concatMapObserver	elementAt	ignoreElements	onErrorResumeNext
generate	timer	connect	elementAtOrDefault	indexOf	pairwise
generateWithAbsoluteTime	toAsync	includes	every	isEmpty	partition
generateWithRelativeTime	using	controlled	expand	join	pausable

```
Observable.prototype.map = (selector, thisArg) => {
  let source = this;
  return Rx.Observable.create(o => {
    let i = 0;
    return source.subscribe(x => {
      try {
        var result = selector.call(thisArg, x, i++, source);
      } catch (e) {
        return o.onError(e);
      }
      o.onNext(result);
    }, o.onError.bind(o), o.onCompleted.bind(o));
  });
}
```



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

The screenshot shows a Netflix mobile application interface. At the top, there's a red header bar with the word "NETFLIX" on the left and "2 / 10" on the right. Below the header, there's a row of six movie thumbnails: "Band Baaja Baaraat", "The Mystery of the Sphinx", "HEROD'S LOST TOMB", "ARABIA", and "IRONCLAD". To the right of these thumbnails is a movie detail card for "Band Baaja Baaraat". The card includes the title, release year (2010), rating (NR), runtime (2h 19m), and a four-star rating icon. Below the title, the plot summary reads: "Shruti and Bittoo decide to start a wedding planning company together after they graduate from university, but romance gets in the way of business." Further down, the cast (Ranveer Singh, Anushka Sharma) and genre (Comedies, Foreign Movies) are listed, along with the director (Maneesh Sharma). At the bottom of the screen, there are three sections: "Top 10 for tester_jhusain_control" (with thumbnails for "There Will Be Blood", "Band Baaja Baaraat", "BROKEN ENGLISH", "THE HUNTED", and "RAÑÍ"), "Popular on Netflix" (with thumbnails for "The Great Indian Adventure", "The Great Indian Adventure", "The Great Indian Adventure", "The Great Indian Adventure", "The Walking Dead", and "The Great Indian Adventure"), and a small "Discover" section.

NETFLIX

2 / 10

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

Top 10 for tester_jhusain_control

BAND BAAJA BAARAAT

DANIEL DAY-LEWIS There Will Be Blood

Band Baaja Baaraat

BROKEN ENGLISH

THE HUNTED

RAÑÍ

Popular on Netflix

THE WALKING DEAD

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 more advanced web socket wrapper for RxJS

Install

With bower:

```
bower install -S rx-socket-subject
```

With npm:

```
npm install -S rx-socket-subject
```

Goals

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

<https://github.com/blesh/RxSocketSubject>

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));
```

Observables and Backpressure

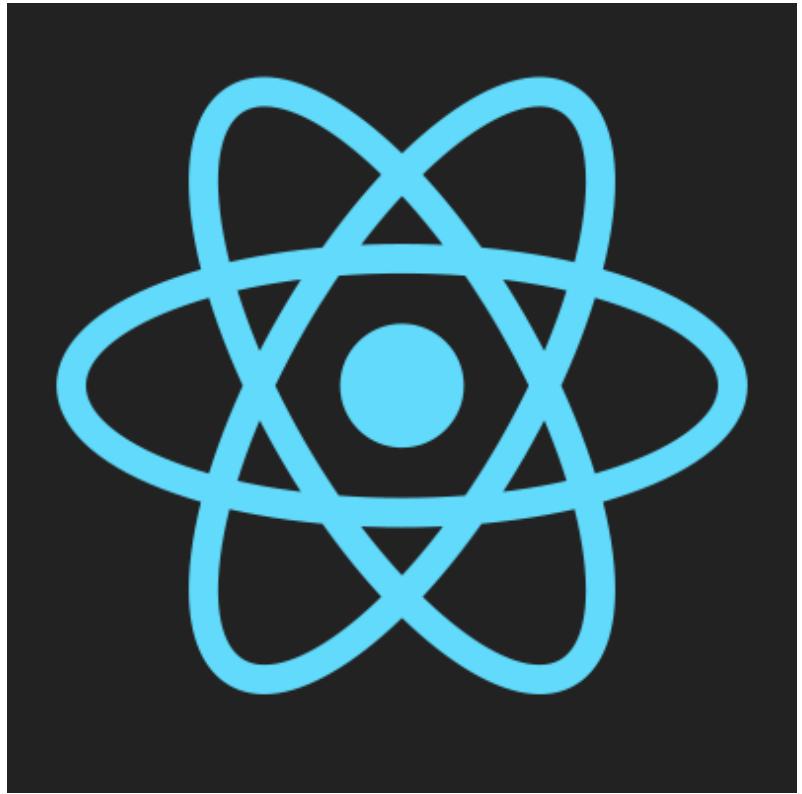
Yes, Observables can have backpressure

- 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);
```

Winning Friends And Influences Others



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





Schedulers...

When?

Where?

How?

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

Testing concurrent code: made easy!

```
let scheduler = new TestScheduler();

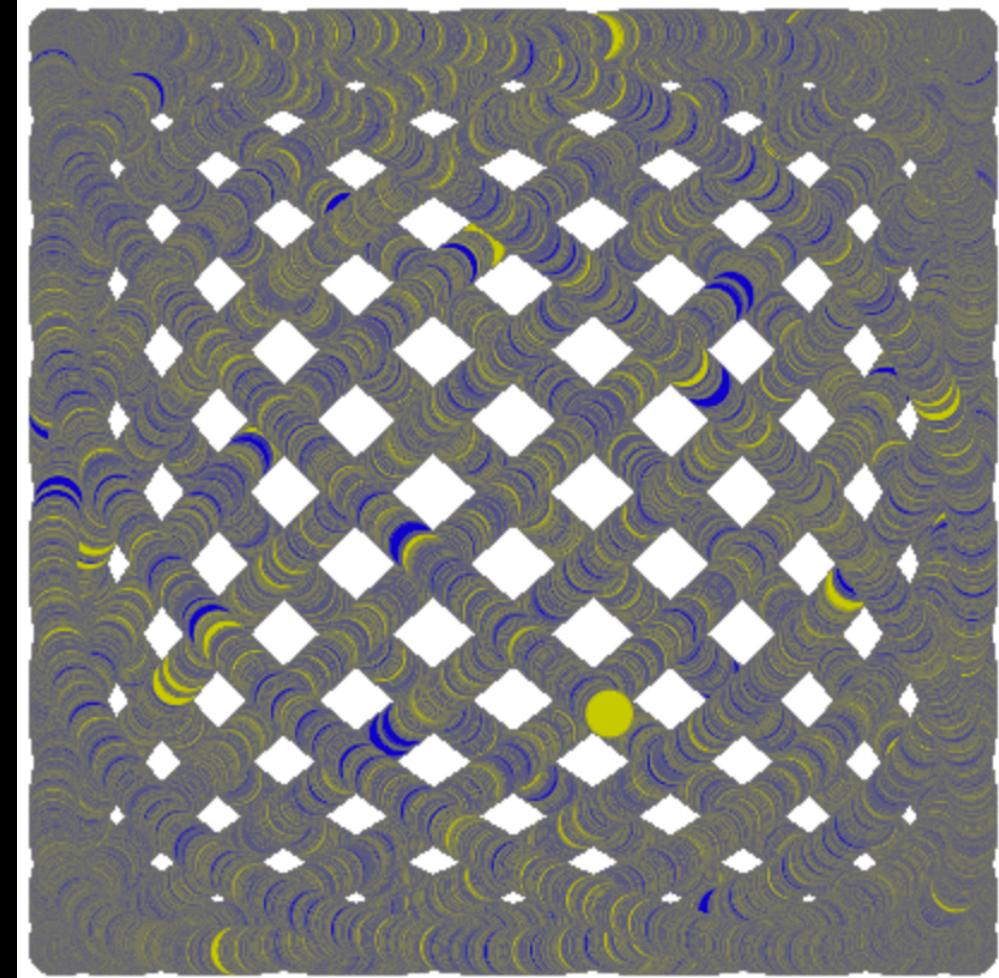
let input = scheduler.createHotObservable(
    onNext(300, 'WebRebels'),
    onNext(400, '2015'),
    onCompleted(500));

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

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

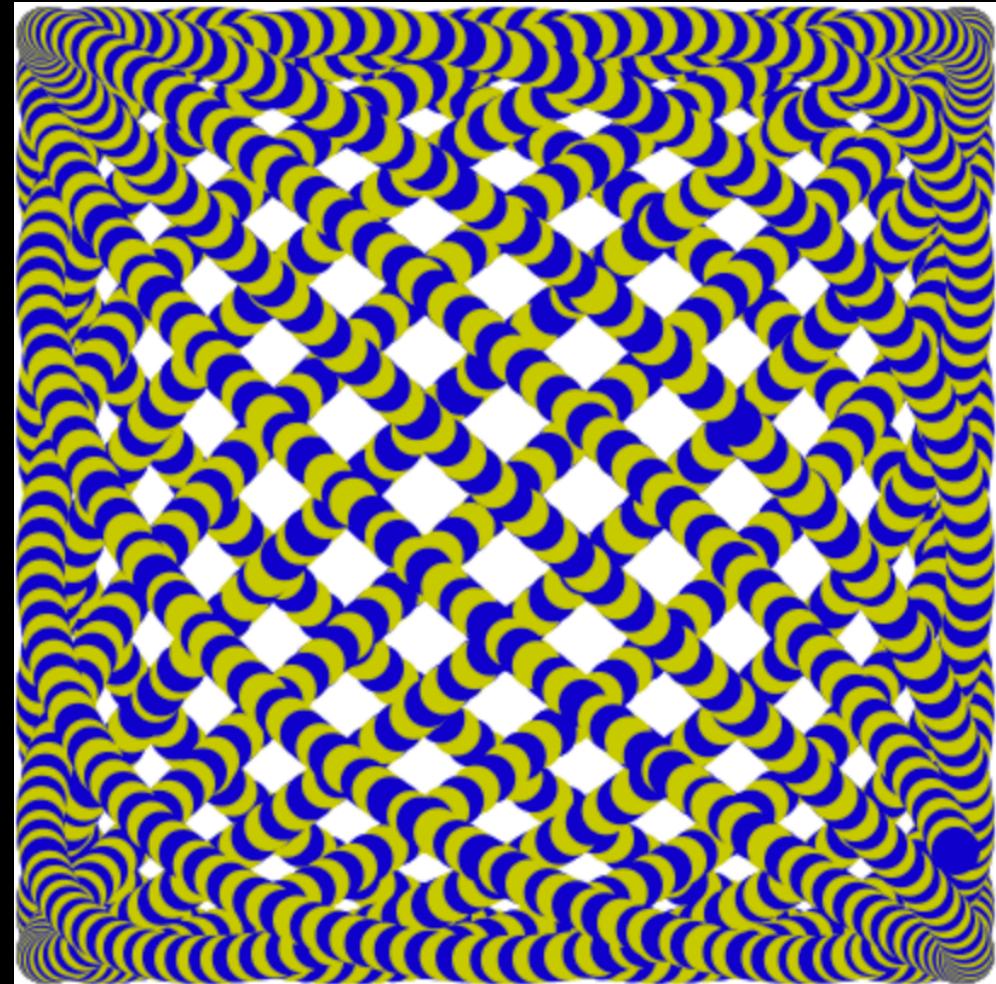
Schedulers Matter!

```
const subscription = Rx.Observable.generate(  
  0,  
  x => true,  
  x => x + 1,  
  x => x,  
  Rx.Scheduler.default  
)  
.timestamp()  
.subscribe(draw);
```



Schedulers Matter!

```
const subscription = Rx.Observable.generate(  
  0,  
  x => true,  
  x => x + 1,  
  x => x,  
  Rx.Scheduler.requestAnimationFrame  
)  
.timestamp()  
.subscribe(draw);
```



Ur Kitteh of Death

Awaits

Async/Await

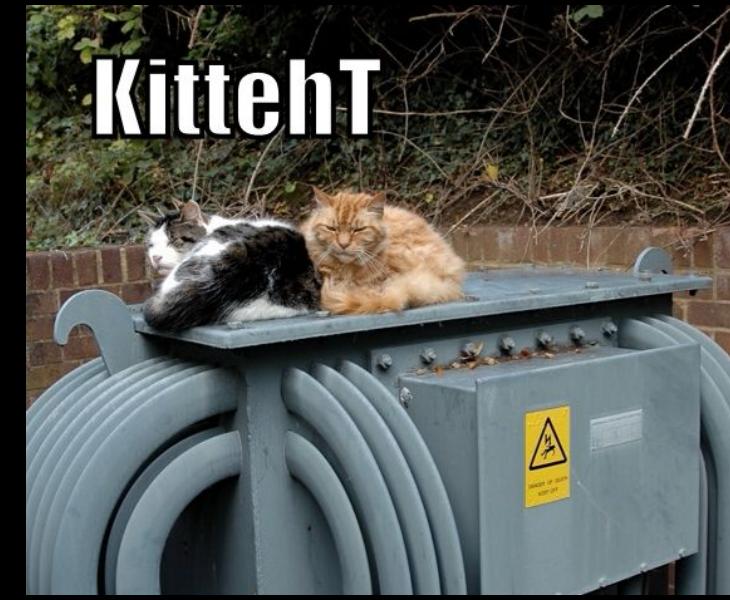
Coming to a JavaScript Engine Near You!

- 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;  
}
```

Async/Await with Observables and Generators...

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



Async Generators in ES7/2016???

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

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

Functional programming provides developers with the tools to abstract common collection operations into reusable, composable building blocks. You'll be surprised to learn that most of the operations you perform on collections can be accomplished with **five simple functions**:

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

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 five functions hold the key to simplifying asynchronous programming. Once you've finished this tutorial you'll also have all the tools you need to easily avoid race conditions, propagate and handle asynchronous errors, and sequence events and AJAX requests. In short, **these 5 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 /*  
2  * welcome to sweet.js!  
3  
4  You can play around with macro writing here on the left side and  
5  your code will automatically be compiled on the right. This page  
6  will also save your code to localStorage on every successful  
7  compile so feel free to close the page and come back later!  
8 */  
9  
10 // Here is a really simple identity macro to get started.  
11  
12 // The `macro` keyword is used to create and name new macros.  
13 macro id {  
14     rule {  
15         // after the macro name, match:  
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
```

```
1 42;  
2 // Note that a single token to sweet.js includes matched  
3 // delimiters not just numbers and identifiers. For example,  
4 // an array with all of its elements counts as one token:  
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 they  
12 // weren't supposed to. This is called hygiene and to enforce hygiene  
13 // 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;
```

