Async JavaScript at Netflix

Matthew Podwysocki @mattpodwysocki

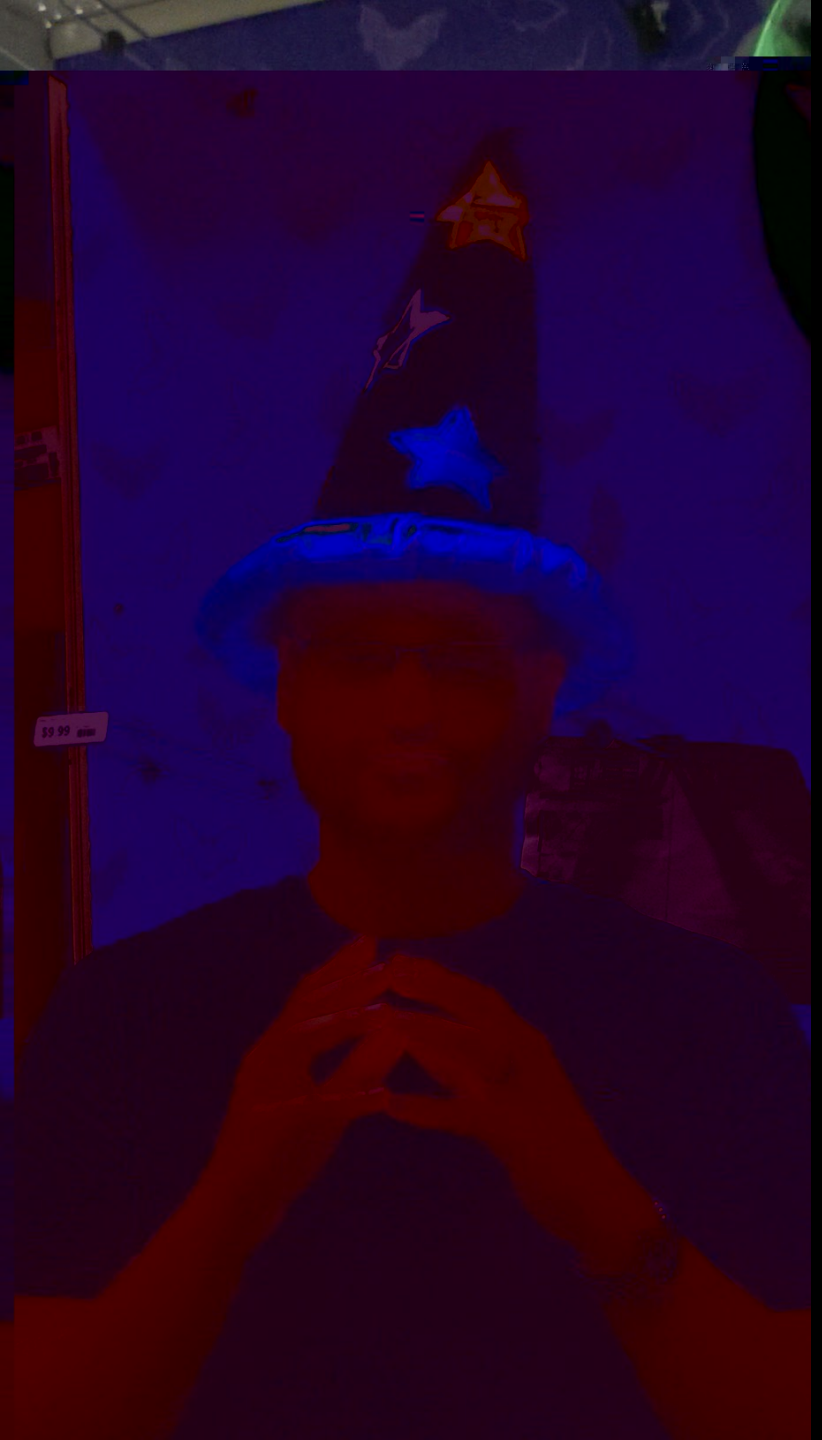
github.com/mattpodwysocki/jsconfuy-2015


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 ?



Principal SDE
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki


MICROSOFT

Reactive Extensions (Rx)

@ReactiveX

http://reactivex.io

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...

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.clienyY - 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);
```

whatwg / fetch

Watch 35 | Star 88 | Fork 11

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

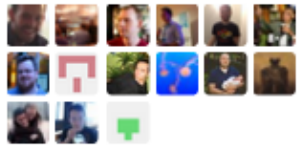
jakearchibald

Notifications

Subscribe

You're not receiving notifications from this thread.

15 participants



petkaantonov / **bluebird**

Watch 182 | Star 6,048 | For

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 thread.

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

The Final Countdown...


```
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

UNSAFE AT ANY SPEED

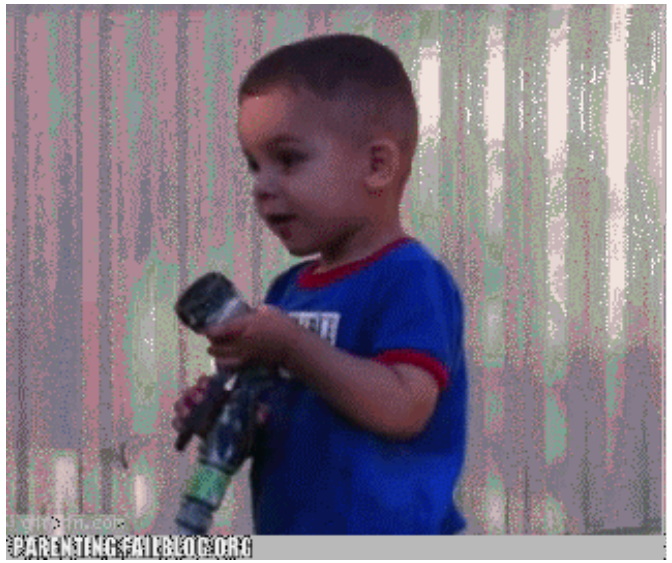
The Designed-In Dangers of The American Automobile

By Ralph Nader


streem 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

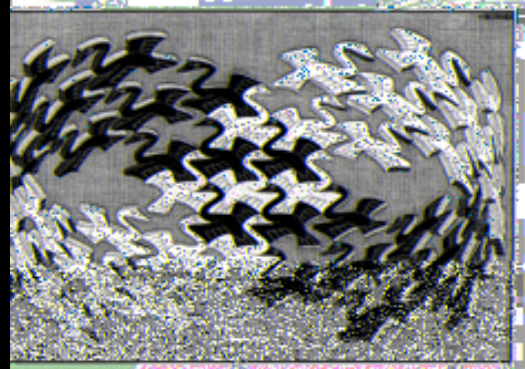
- Landed in 0.11.2
- Adds cork/uncork/_writev


1994
Sports Illustrated
ACE VENTURA
PET DETECTIVE
KEANU REEVES
CLERKS
A very funny look at the over-the-counter culture.
NIRVANA
UNPLUGGED
IN NEW YORK
weezer

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



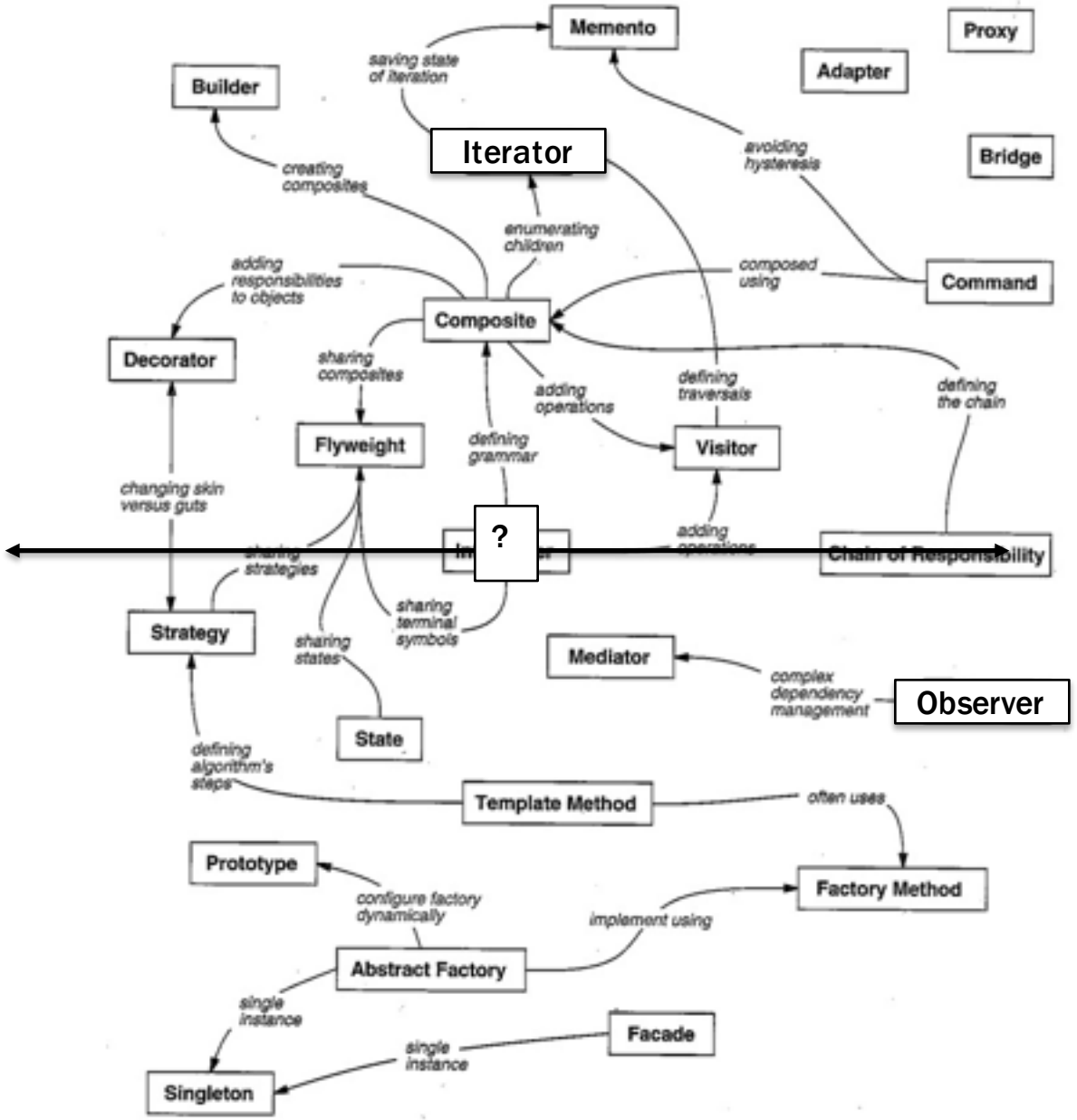
oreword by Grady Booch

ADDISON-WESLEY PROFESSI

ONAL COMPUTING SERIES

Fo

Development



Design Pattern Relationships

Iterator Pattern with ES2015

```
> var iterator = getNumbers();
> console.log(iterator.next());
> { value: 1, done: false }
> console.log(iterator.next());
> { value: 2, done: false }
> console.log(iterator.next());
> { value: 3, done: false }
> console.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 }
> { clientX: 476, clientY: 760 }
> { clientX: 476, clientY: 760 }
> { clientX: 476, clientY: 760 }
```

“What’s the difference between an Array…

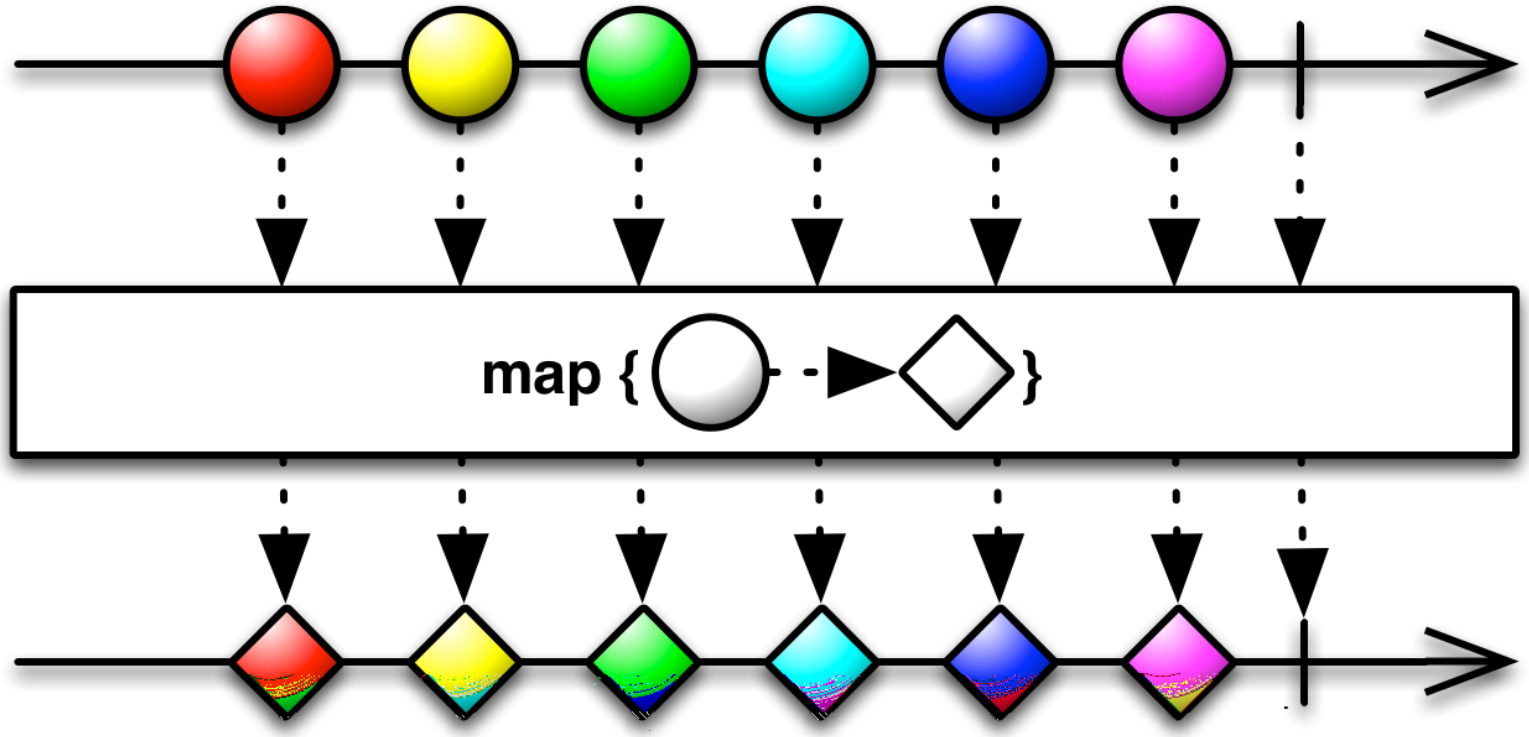
... and an Event?

Events and Arrays are *both* collections.

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

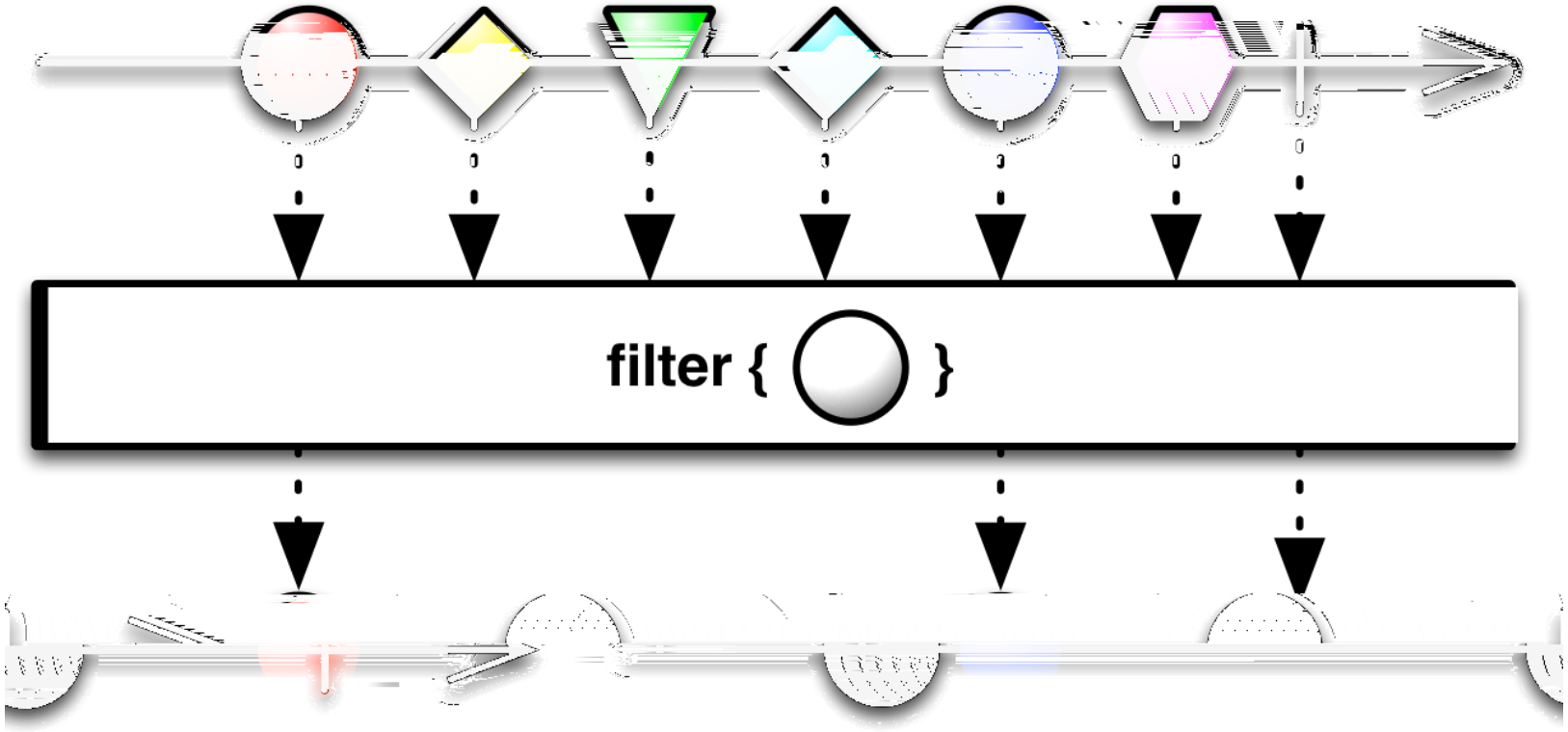
map()

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



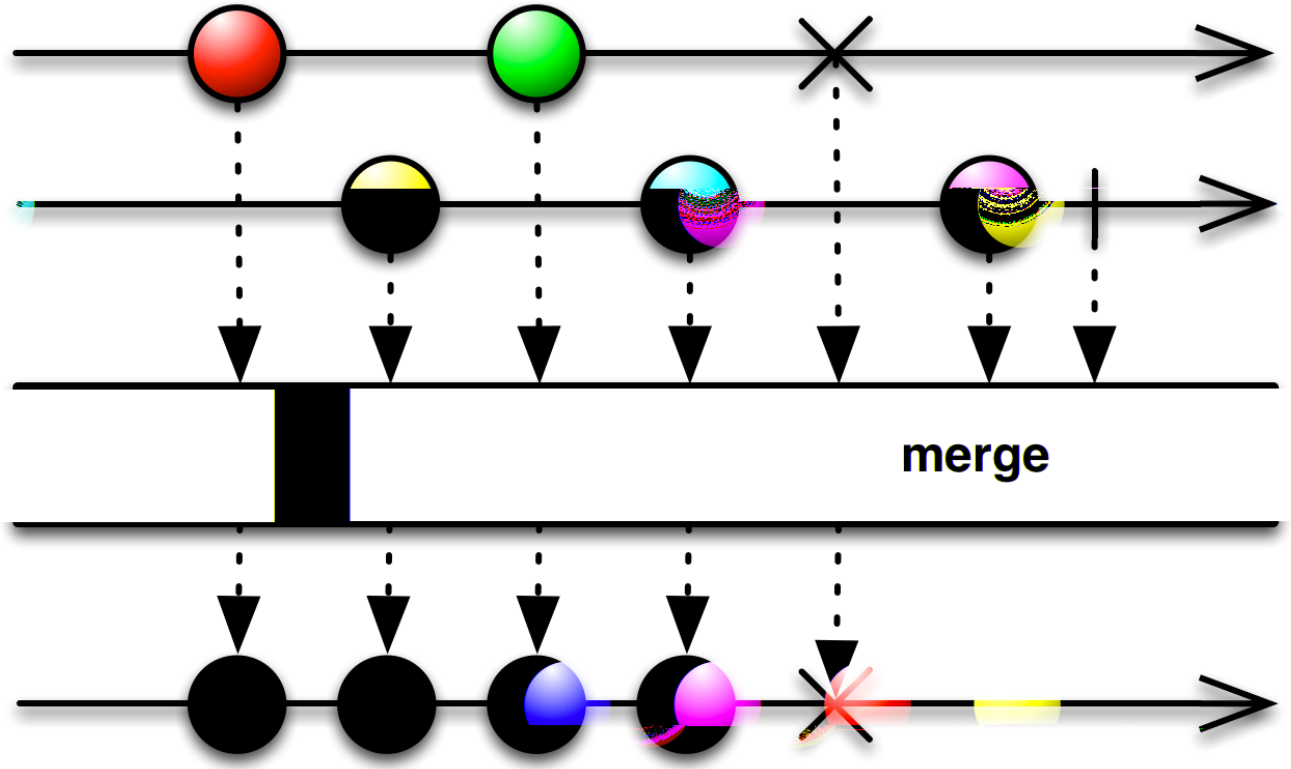
filter()

Filter items emitted by a Collection



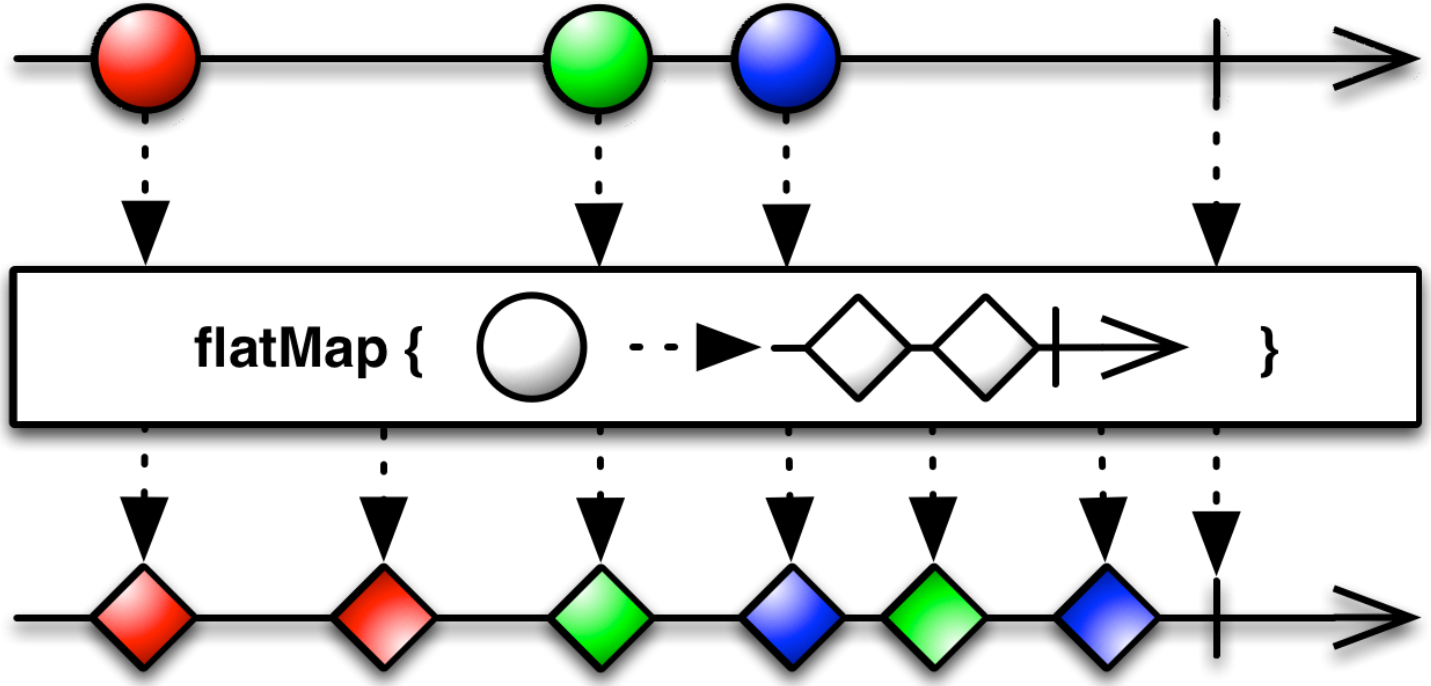
mergeAll()

combine multiple Collections into one by merging their emissions



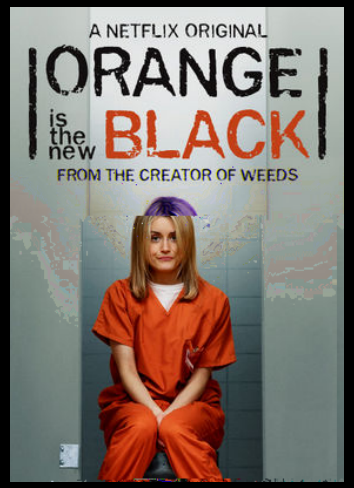
flatMap()

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



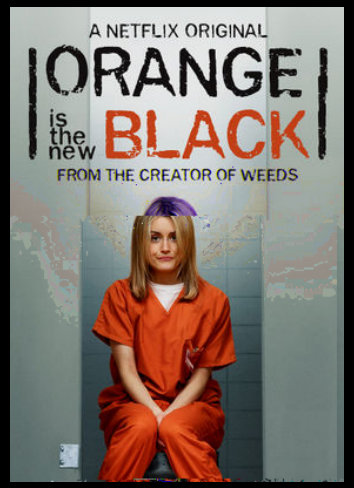
Top-rated Movies Collection

```
let getTopRatedFilms    (user)
  user.videoLists
    .map((videoList)
      videoList.videos
        .filter((v)    v.rating      )
    ).mergeAll();
 ;

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

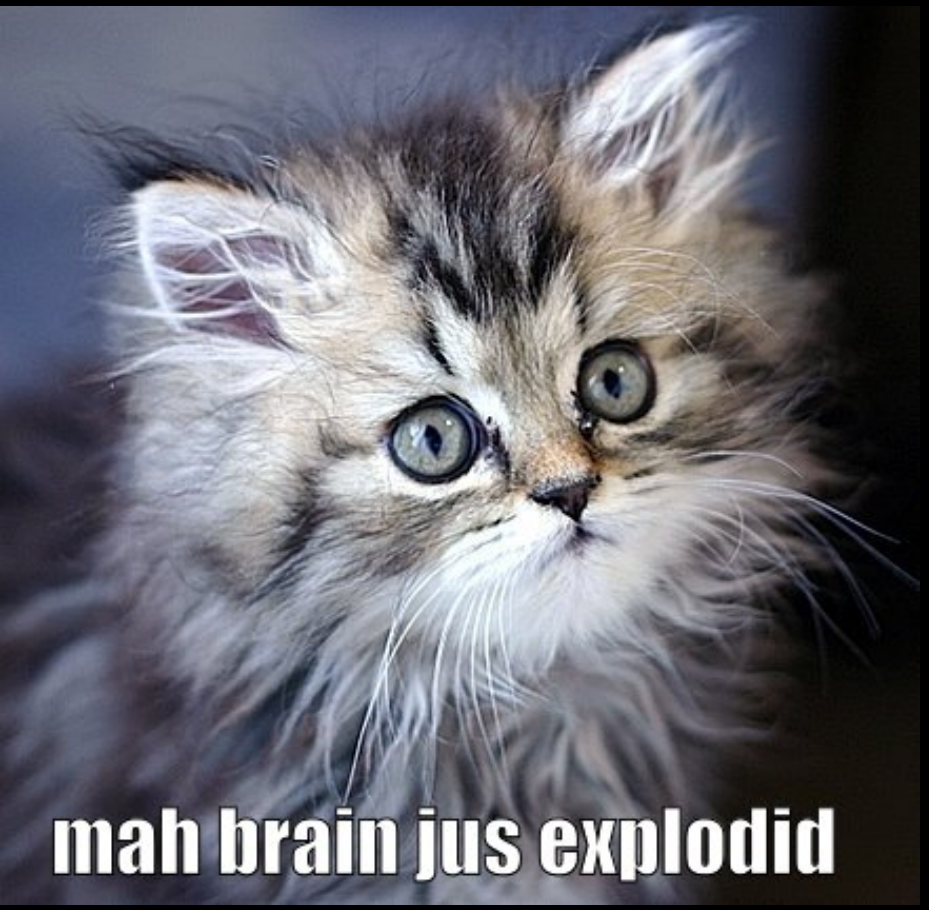


Top-rated Movies Collection

```
let getTopRatedFilms    (user)
  user.videoLists
    .flatMap((videoList)
      videoList.videos
        .filter((v)    v.rating      )
    )
 ;

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



What if I told you…



…that you could create a drag event…

…with the almost the *same code*

Mouse Drags Collection

```
let getElementDrags    (elmt)
  dom.mousedown(elmt)
    .map((md)
      dom.mousemove(document)
        .filter   .takeUntil(dom.mouseup(elmt));
    ).mergeAll()
 ;

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

Mouse Drags Collection

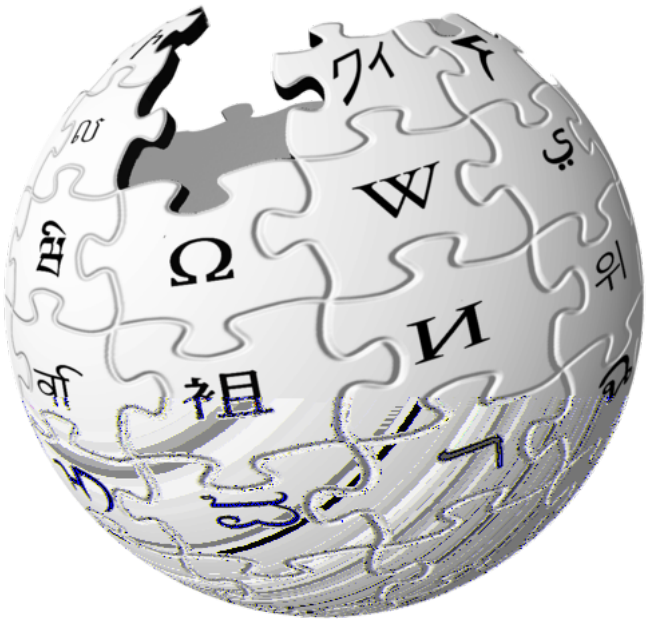
```
let getElementDrags    (elmt)
  dom.mousedown(elmt)
    .flatMap((md)
      dom.mousemove(document)
        .filter   .takeUntil(dom.mouseup(elmt))
    )
 ;

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


Everything is a stream

First-Class Asynchronous Values

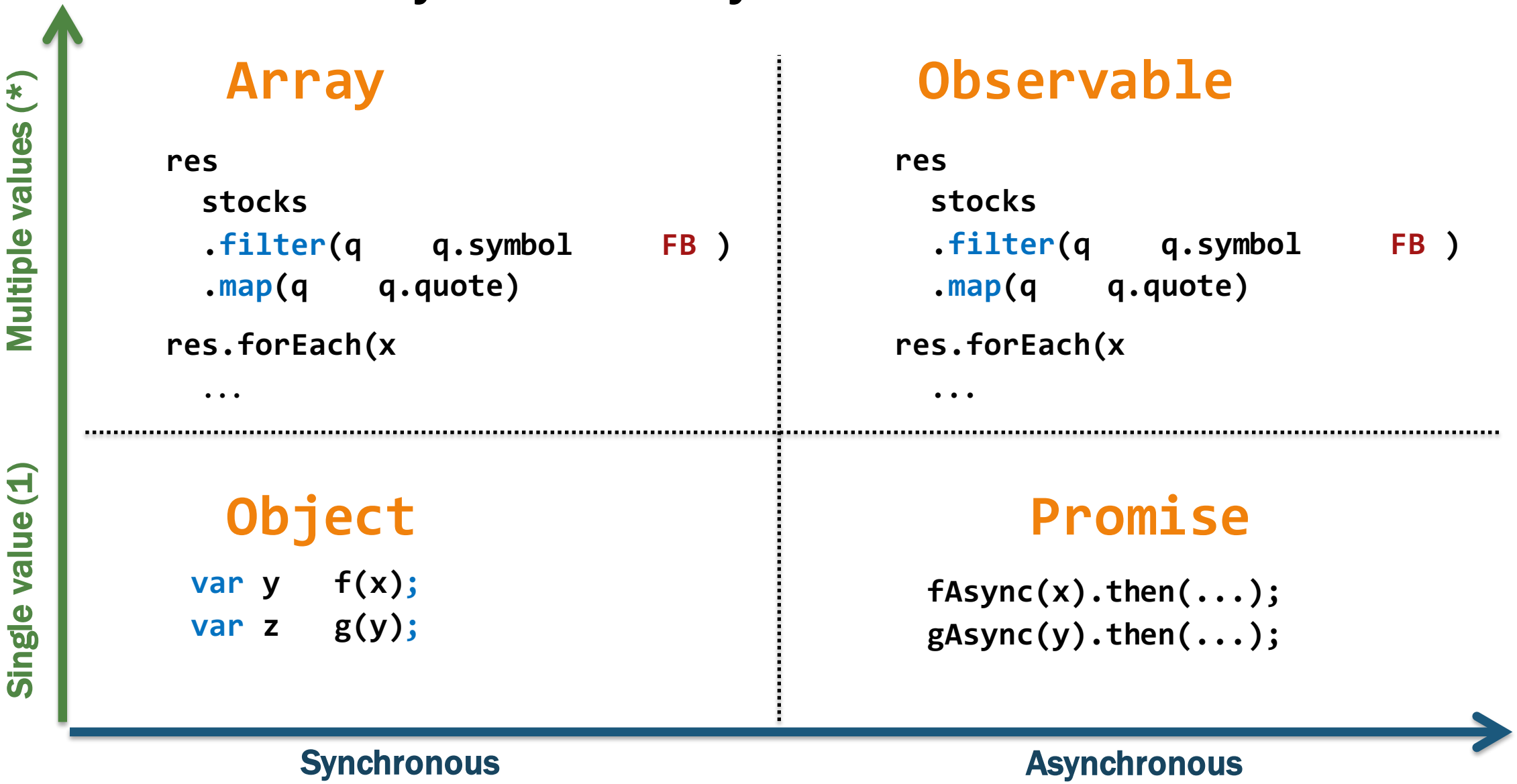
本語
olski
усский

- can be stored in variables and d
- can be passed as a parameter t
- can be returned as the result of
- can be constructed at runtime
- has intrinsic identity (independe



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

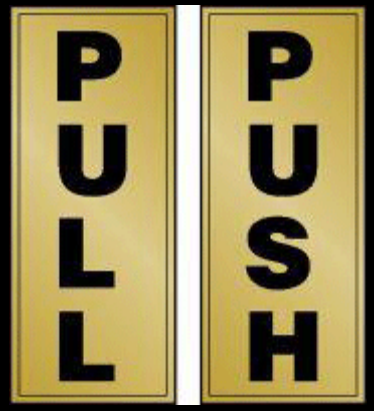
- Continuous Time
- Behaviors: Values over time
- Events: Discrete phenomena with a value and a time
- Compositional behavior for behavior and events

What it is not

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

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

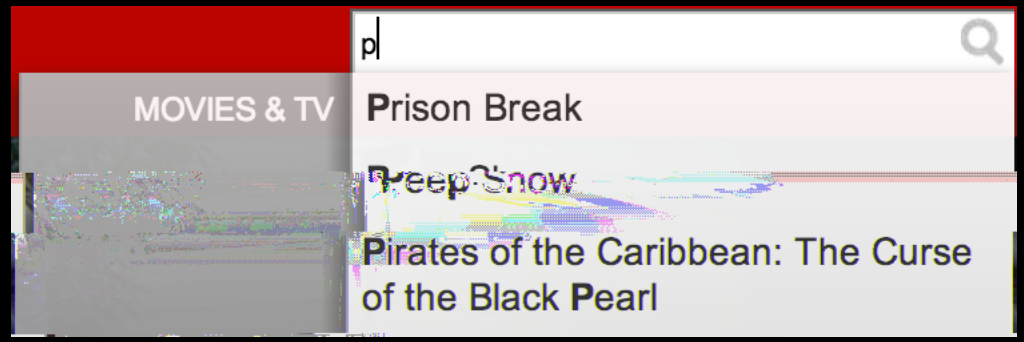
Call Us CEP, Compositional Event Processing




```
let source   getStockData();

source
  .filter((quote)
    quote.price     ;
  )
  .map((quote)
    quote.price;
  )
  .forEach((price)
    console.log(      :      price);
  );
```

```
let source   getStockData();

source
  .filter((quote)
    quote.price     ;
  )
  .map((quote)
    quote.price;
  )
  .forEach((price)
    console.log(      :      price);
  );
```

Netflix Search



Netflix Search with Observables

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

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

DOM events as a sequence of strings

Reducing data traffic / volume

Latest response as movies

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 * (onError | onCompleted)?

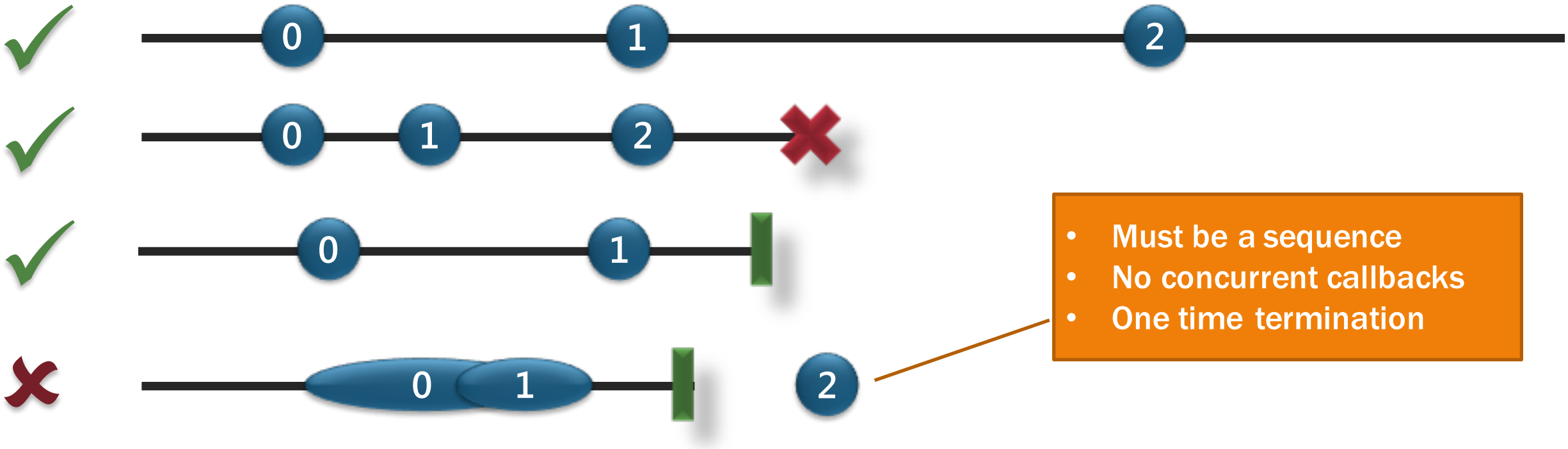
Zero or more values

E.g. events are ∞ sequences

Calls can fail

Resource management

Sequencing



- Must be a sequence
- No concurrent callbacks
- One time termination

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 mousedrag = mousedown.flatMap((md) => {

    // calculate offsets when mouse down
    let startX = md.offsetX,
        startY = md.offsetY;

    // calculate diffs until mouse up
    return mousemove.map((mm) => {
        return {
            left: mm.clientX - startX,
            top:  mm.clientY - startY
        };
    })
});
```

1

For each mouse down

2

Take mouse moves

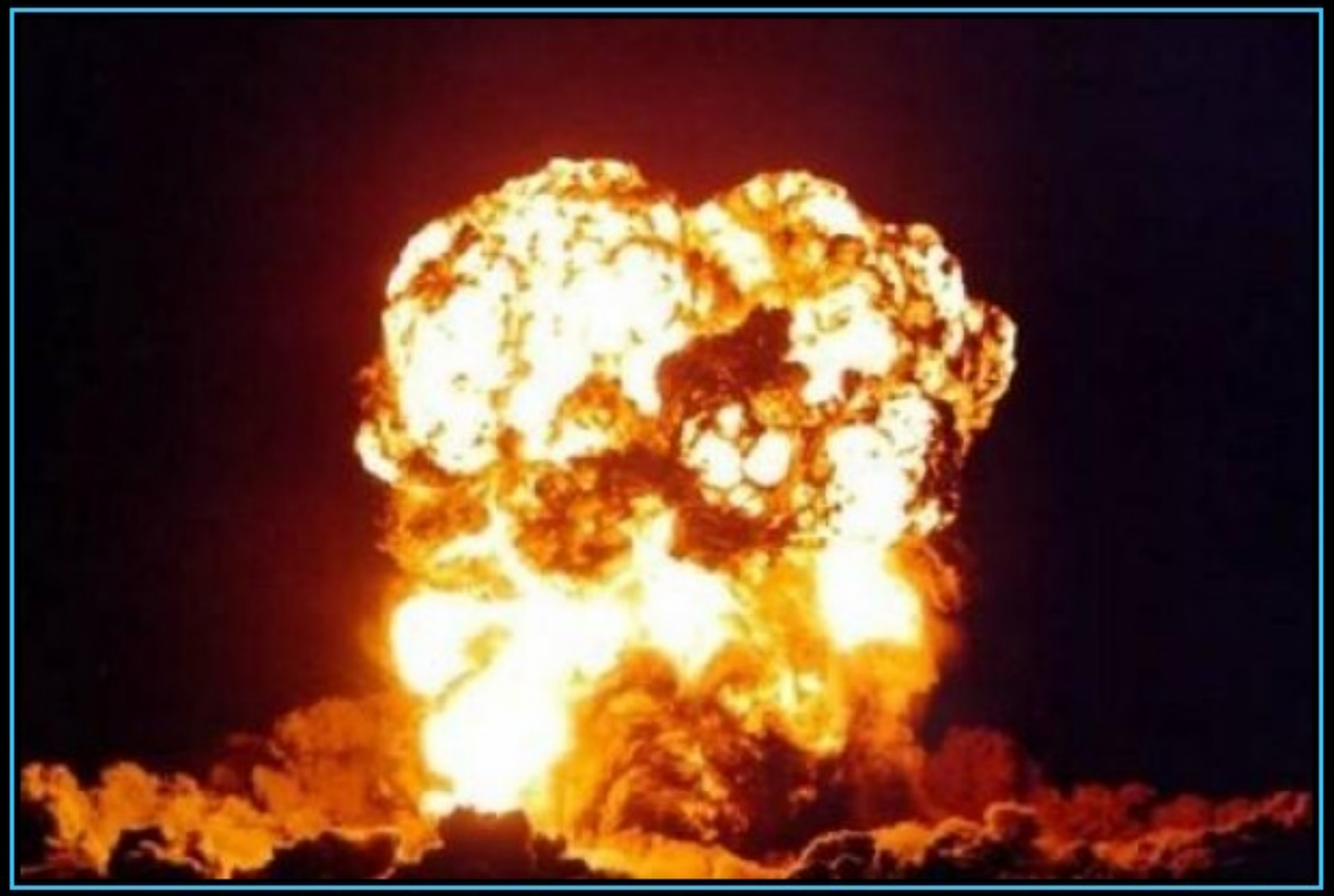
Observables - Querying UI Events

```
let mousedrag = mousedown.flatMap((md) => {

    // calculate offsets when mouse down
    let startX = md.offsetX,
        startY = md.offsetY;

    // calculate diffs until mouse up
    return mousemove.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

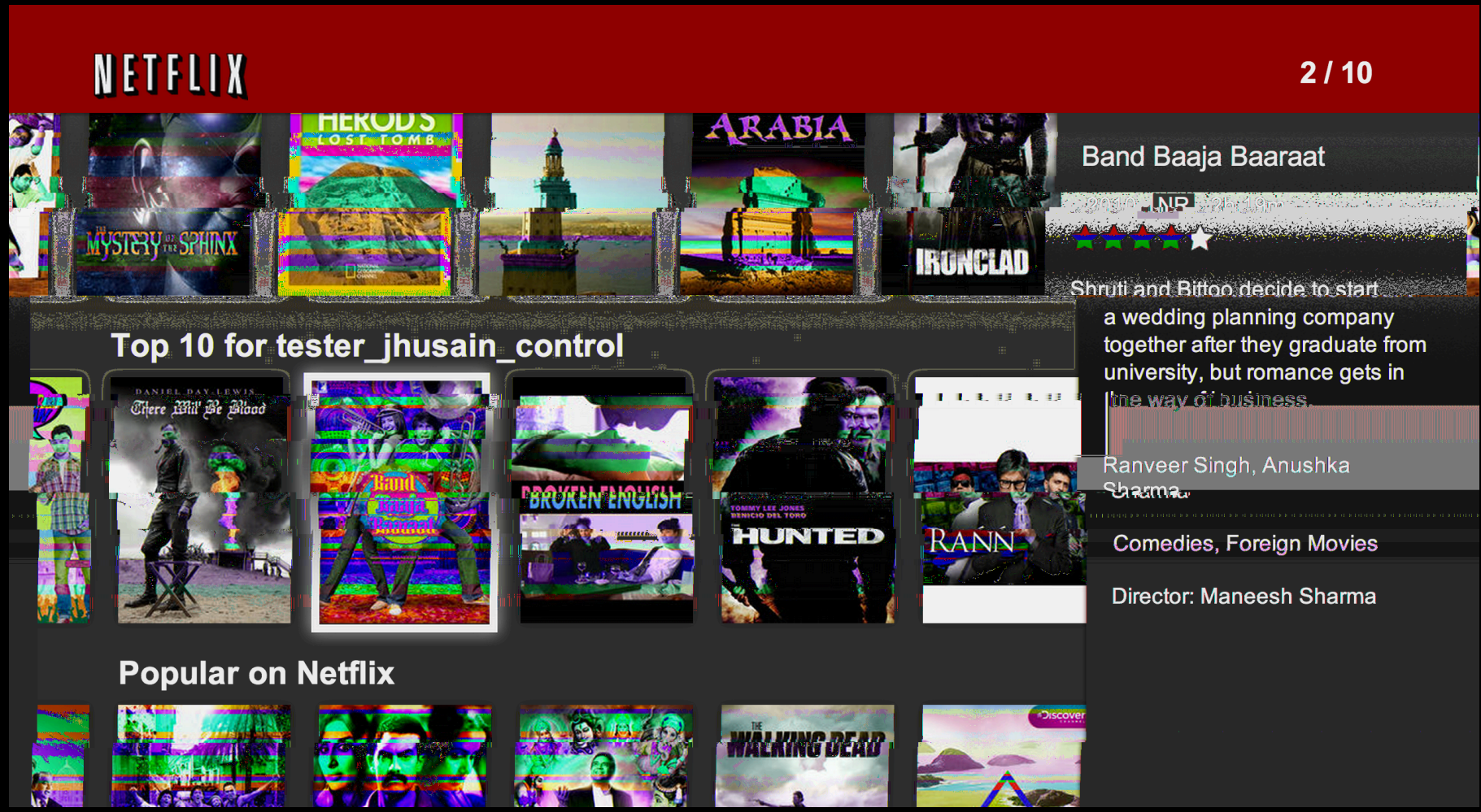


PROTONIC REVERSAL

You crossed the streams, didn't you?

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.debouce(  )
             .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(  ))
       .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));
                )
        );
authorizations.forEach(
    (license) => player.play(license),
    (error) { showDialog("Sorry, can’t play right now."); });
```

RxSocketSubject

ed web socket wrapper for RxJS

A more advance

Install

With bower:

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

```
bower install
```

With npm:

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

```
npm install -
```

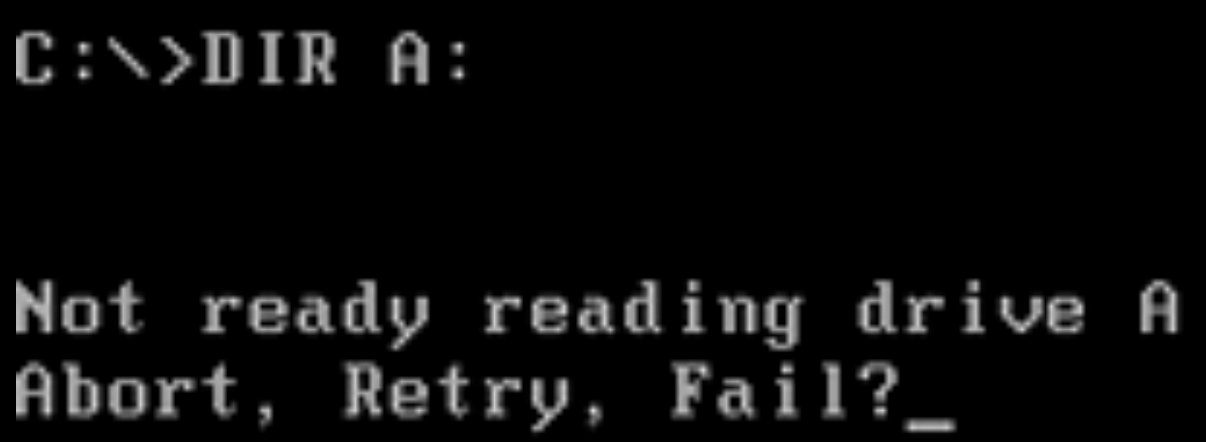
Goals

s project is to produce an observable WebSocket implementation that meets a set of common needs for ently working on. RxJS-DOM has a fine WebSocket implementation, which I modelled the initial work on tion off of. However, I need something that also does the following:

The goals of this projects I'm curr this implementat

```
let socket   RxSocketSubject.create(
  [ ws://netflix.com/socket ,
    ws://netflix.com/socket  ,
    ws://netflix.com/socket  ]);


socket.forEach((e)
  if (e.data      end )
    socket.onCompleted();

  if (e.data      bad data )
    socket.onError(new Error( bad data ));

 );

socket.onNext( some data );
```

```
var d   getData();
                  ,
d.retry( )
    .catch(defaultData)
    .finally(()    d.cleanup())
    .forEach(
        (data)

         ,
        (err)

         );
```



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(  );
```

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;
```

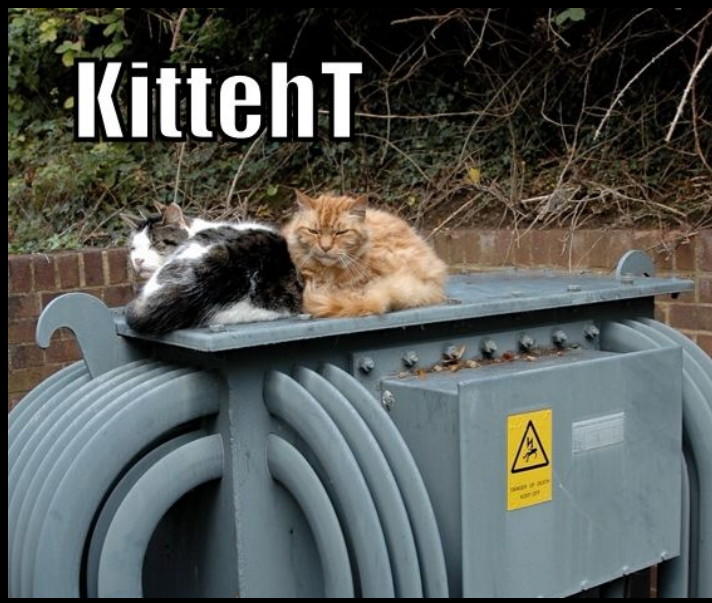
https://github.com/lukehoban/ecmascript-asyncawait

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( )
    .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
      document.mousemove.takeUntil(document.mouseup))
        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

ou perform on collections can be

composable building blocks. You'll be surprised to learn that most of the operations y
accomplished with **five simple functions**...

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

rter, more self-descriptive, and more
e five functions hold the key to
o have all the tools you need to easily
vents and AJAX requests. In short,
ctions you'll ever learn.

Here's my promise to you: if you learn these 5 functions your code will become sho
durable. Also, for reasons that might not be obvious right now, you'll learn that thes
simplifying asynchronous programming. Once you've finished this tutorial you'll also
avoid race conditions, propagate and handle asynchronous errors, and sequence e
these 5 functions will probably be the most powerful, flexible, and useful func

http://jhusain.github.io/learnrx/

RxMarbles

Interactive diagrams of Rx Observables

TRANSFORMING OPERATORS

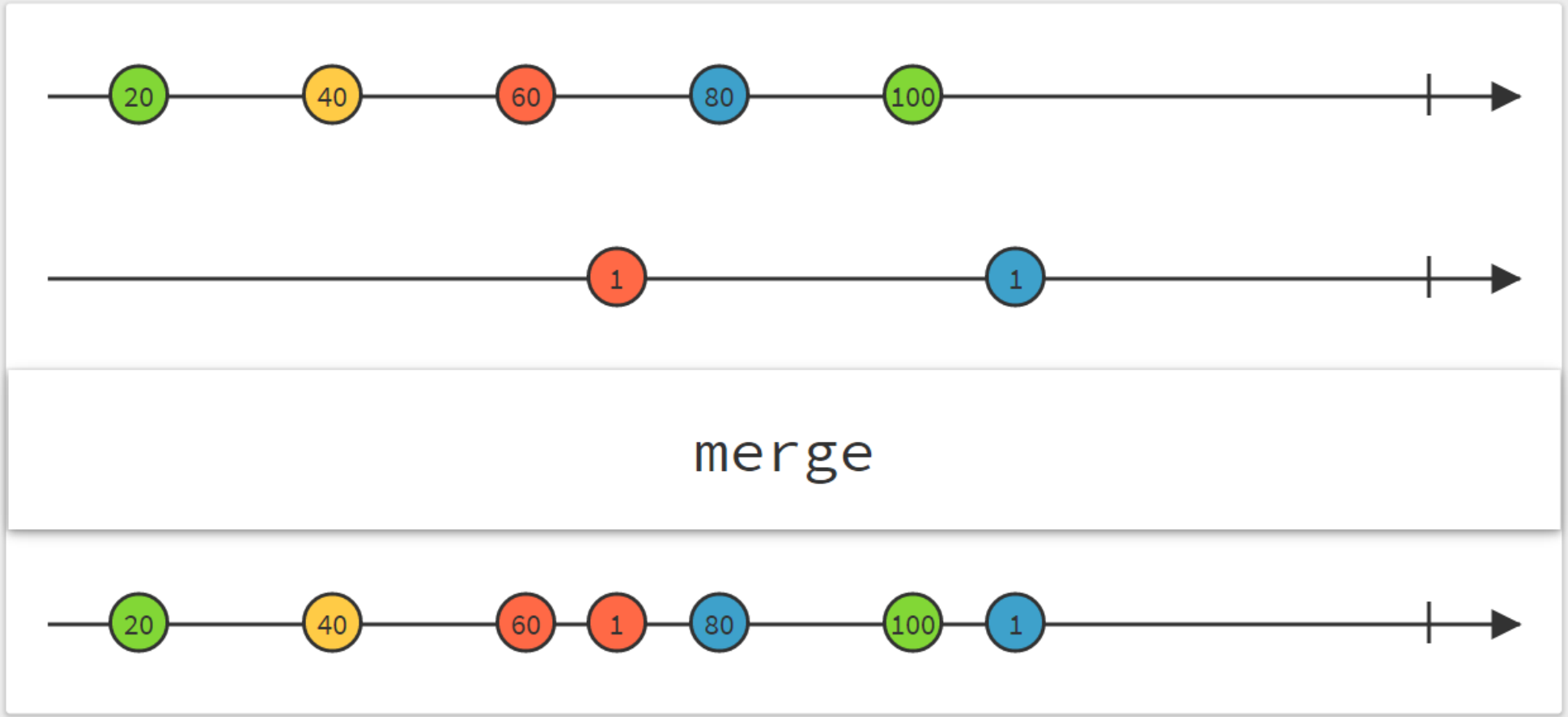
- delay
- delayWithSelector
- findIndex
- map
- scan
- throttle
- throttleWithSelector

COMBINING OPERATORS

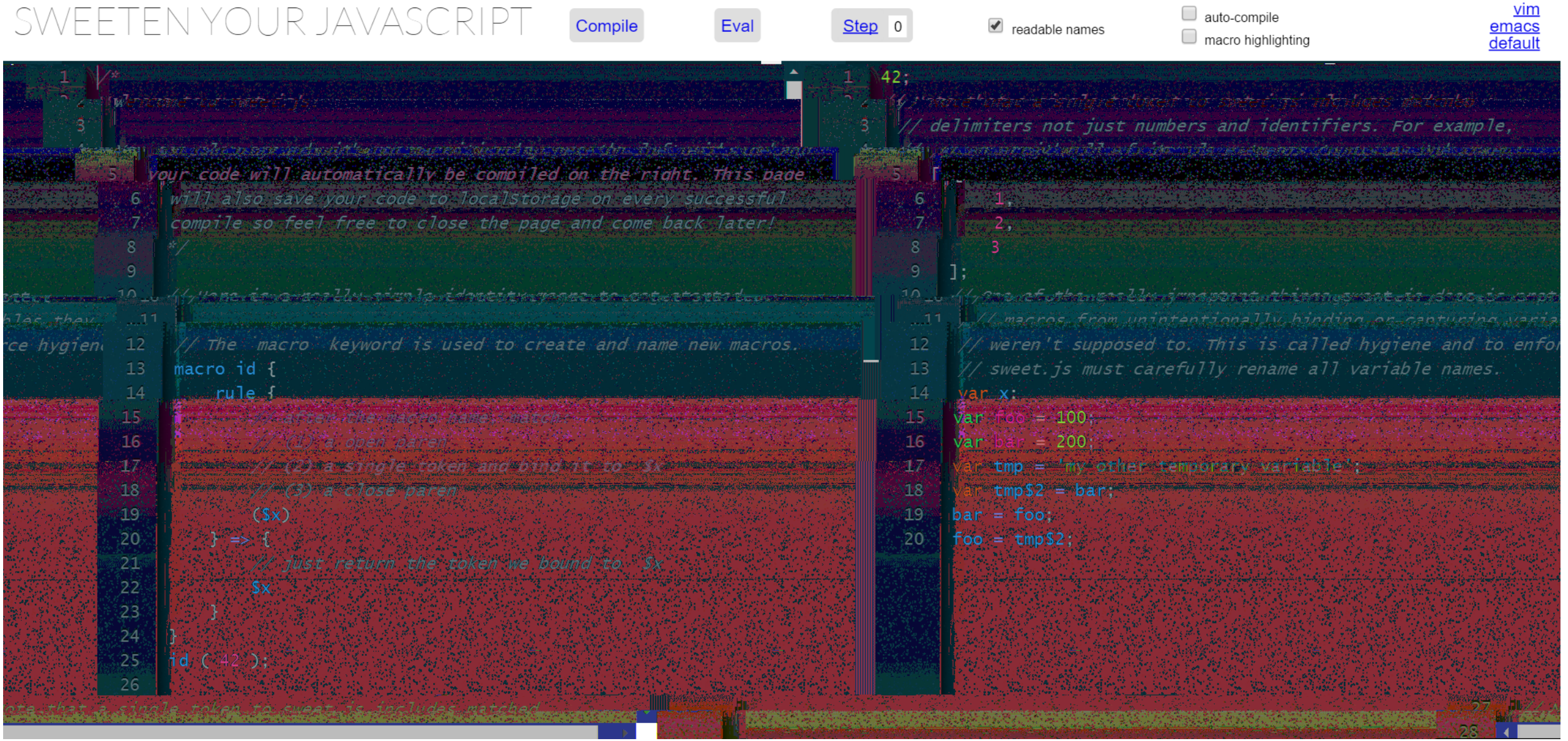
- combineLatest
- concat
- merge
- sample
- startWith
- zip

FILTERING OPERATORS

- distinct
- distinctUntilChanged
- elementAt
- filter



http://www.rxmarbles.com/



http://sweetjs.org/editor.html