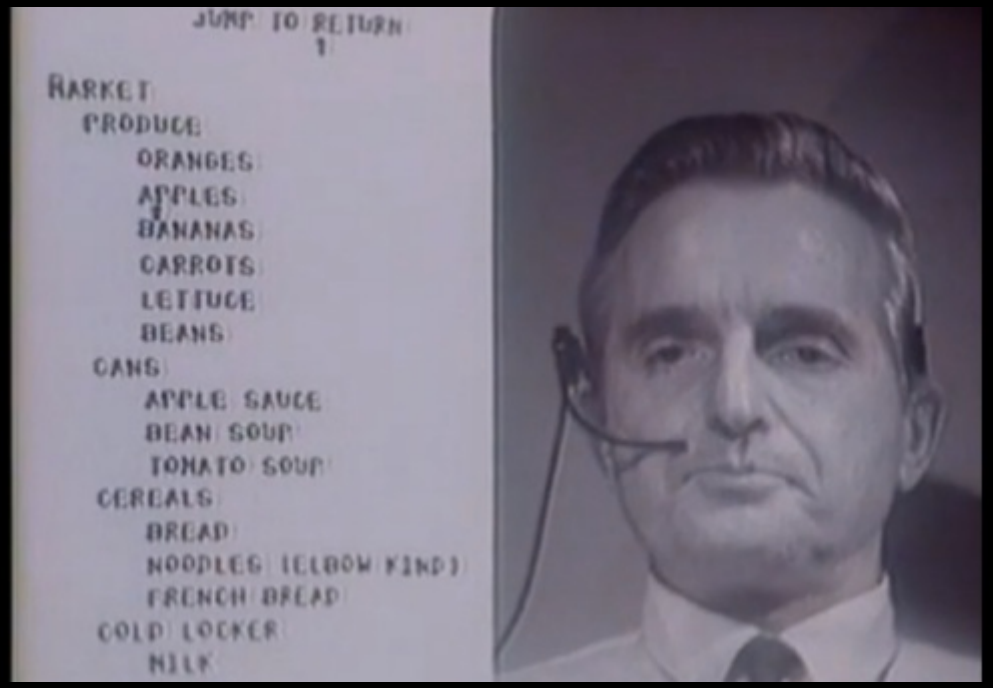
Mother of All Monad Demos



Jenn Schiffer @jennschiffer

Brendan Eich @BrendanEich

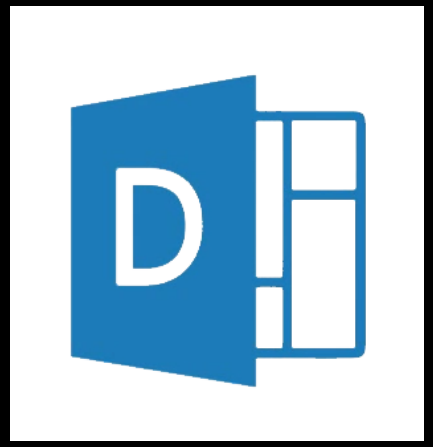
The Mother of All Demos (1968)

- Douglas Engelbart

A MONAD IS JUST A MONOID IN THE CATEGORY OF ENDOFUNCTORS.

WHAT'S THE PROBLEM ?


trapd in Monad tutorl
plz help



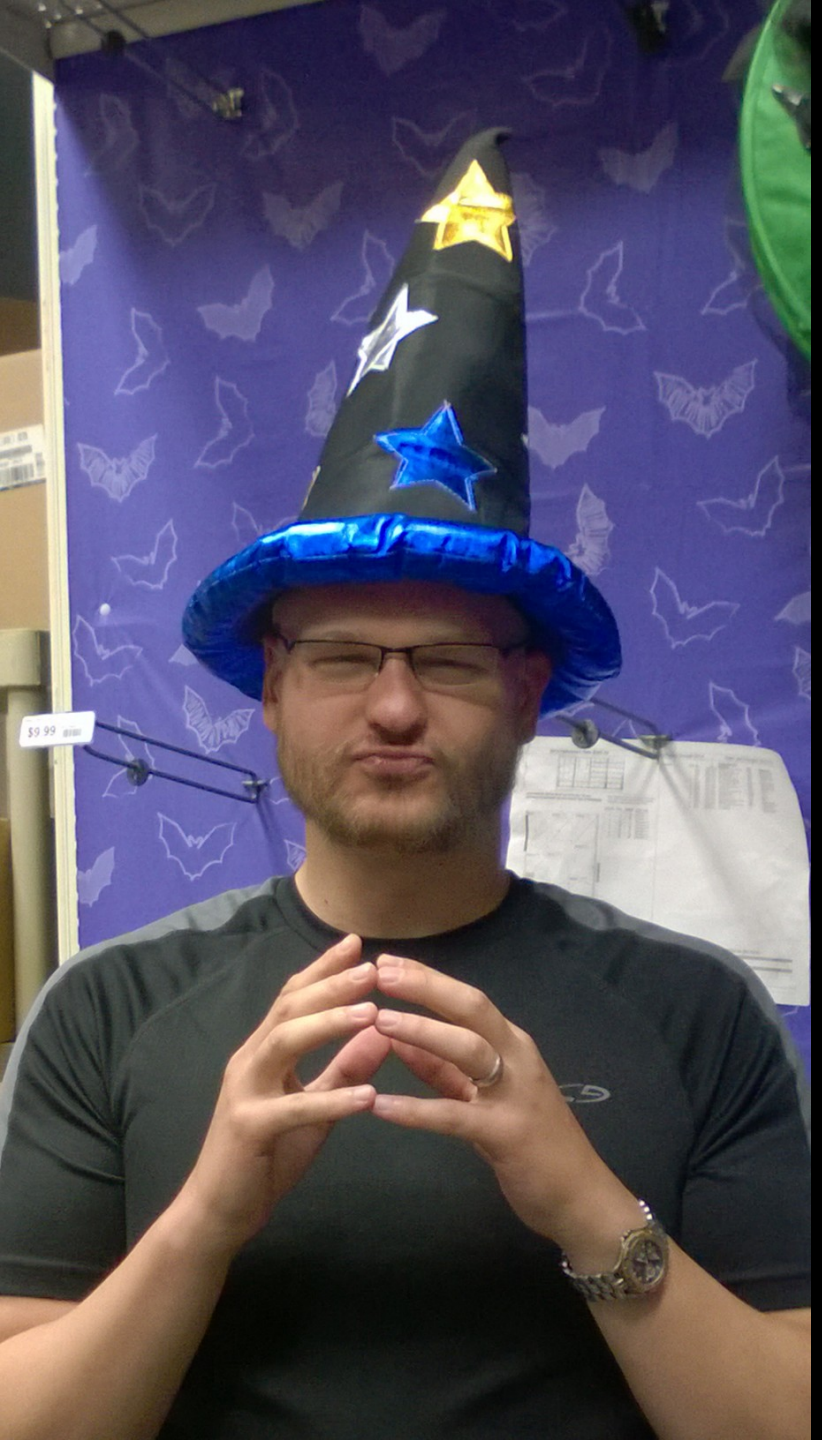
Async JavaScript at Netflix, Microsoft and the World

Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/jsconfco-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

@ThaliProject
github.com/thaliproject


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, Microsoft and others 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...

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


culturepub.fr
next
ad

Events and the Enemy of the State

```
var isDown = false,

function mousedown (
  isDown = true;
  state = { startX:
            startY:
}

function mousemove (
  if (!isDown) { ret
  var delta = { endX
                endY
  // Now do somethi
}

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

```
sedown', mousedown, false);
seup', mouseup, false);
emove', mousemove, false);

n', mousedown, false);
, mouseup, false);
', mousemove, false);
```

First Class Async with Promises

then

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

Breaking the Promise...

then

```
let 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 on Mar 26 · 204 comments

annevk commented on Mar 26 — 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.

A promise-subclass

Labels

None yet

Milestone

No milestone

Assignee

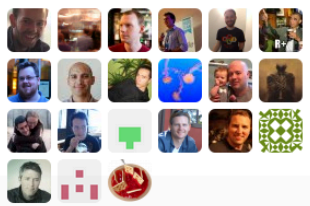

jakearchibald

Notifications

Subscribe

You're not receiving notifications from this thread.

21 participants



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

then

```
var resou

try {                                    e) {
  var res
} catch (
  throw e
} finally
  resourc
}
```

// How do I clean up my resource?

then

UNSAFE AT ANY SPEED

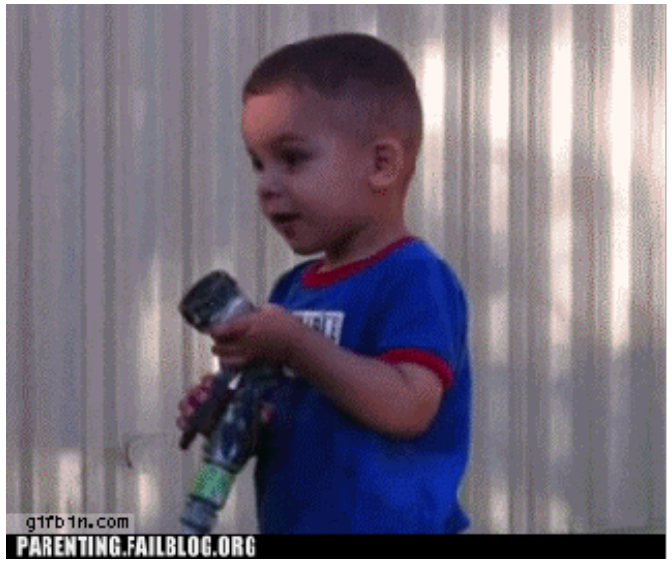
The Designed-In Dangers of The American Automobile

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...



Streams 2 Electric Boogaloo

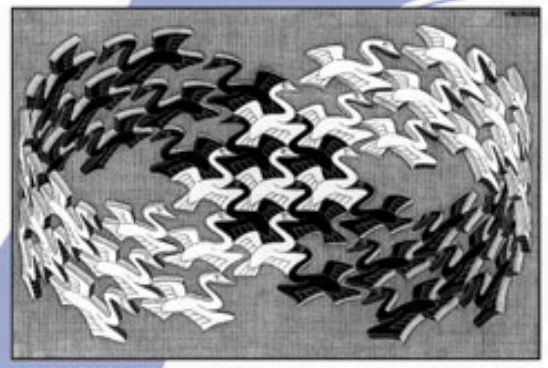
Streams 33 1/3


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

Design Patterns

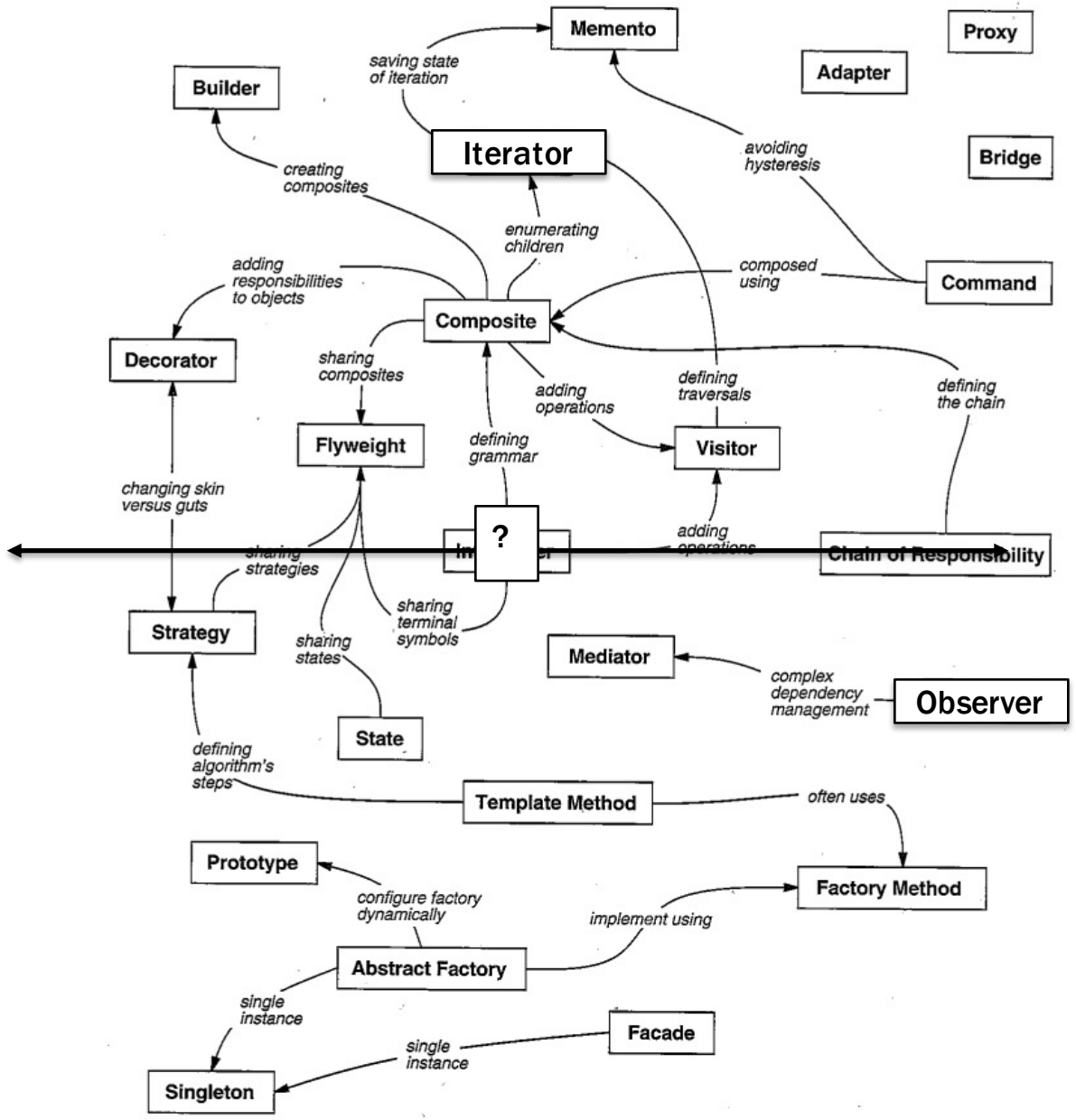
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



Design Pattern Relationships

Iterator Pattern with ES2015

```
> let 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...

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

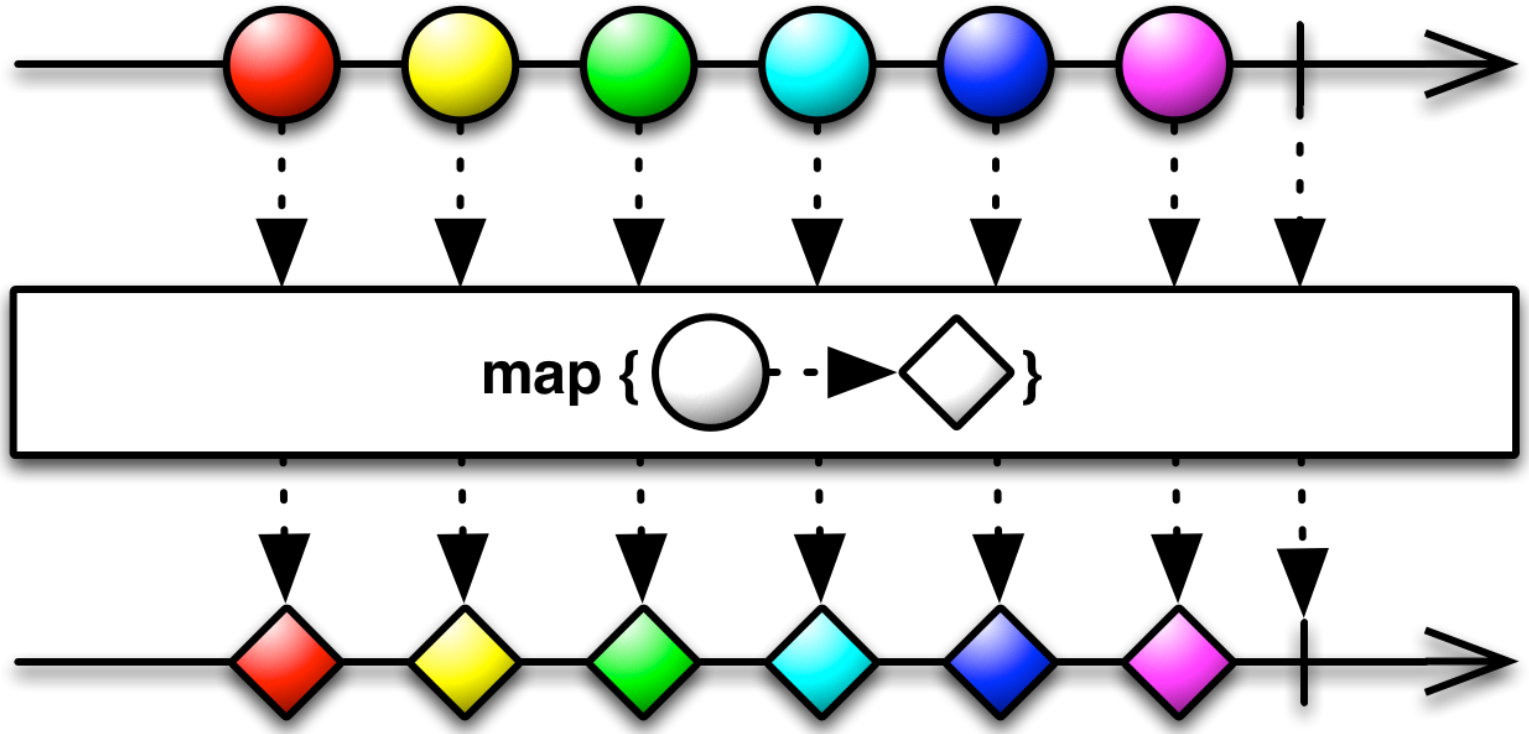
... and an Event?

Events and Arrays are *both* collections.

The majority of asynchronous code from Netflix, Microsoft and others is written with just a few *flexible* functions.

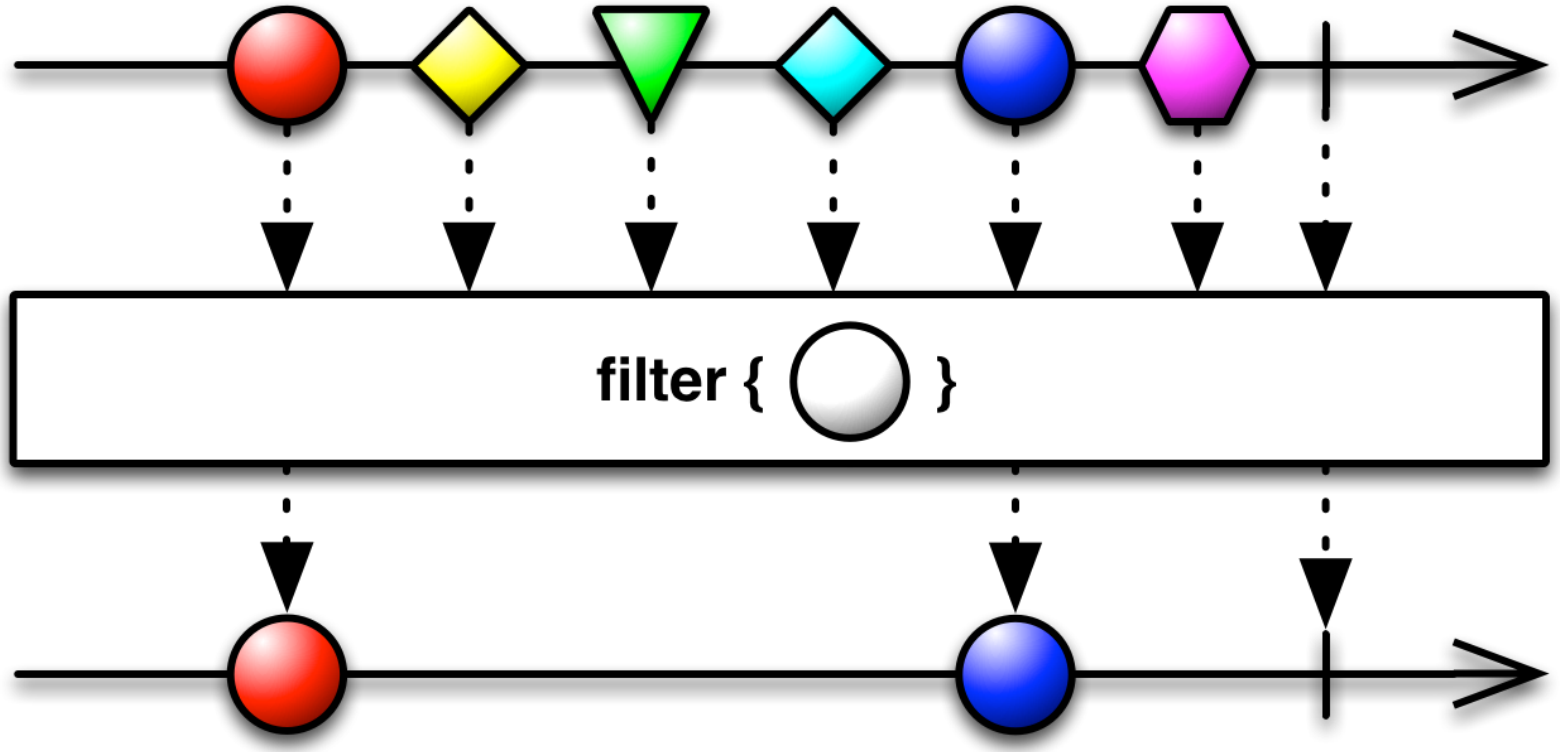
map(selector, thisArg)

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



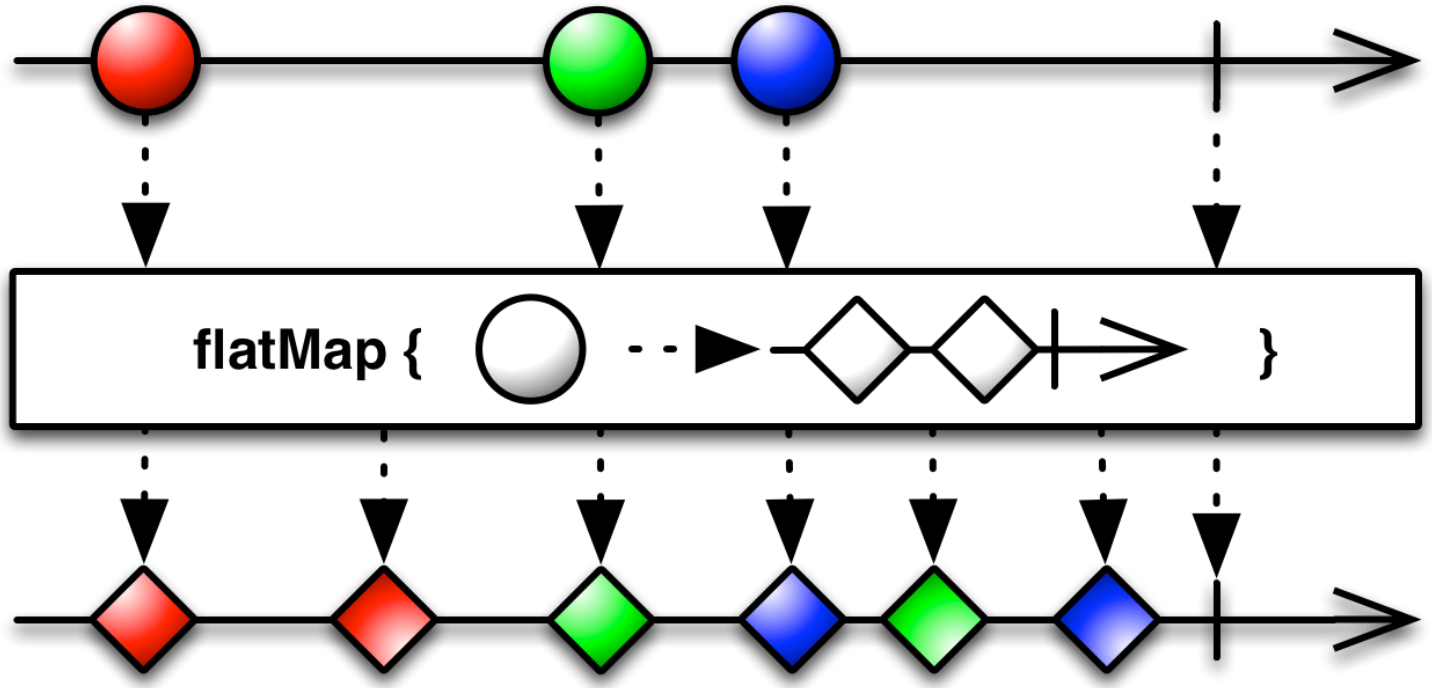
filter(predicate, thisArg)

Filter items emitted by a Collection



flatMap(selector)

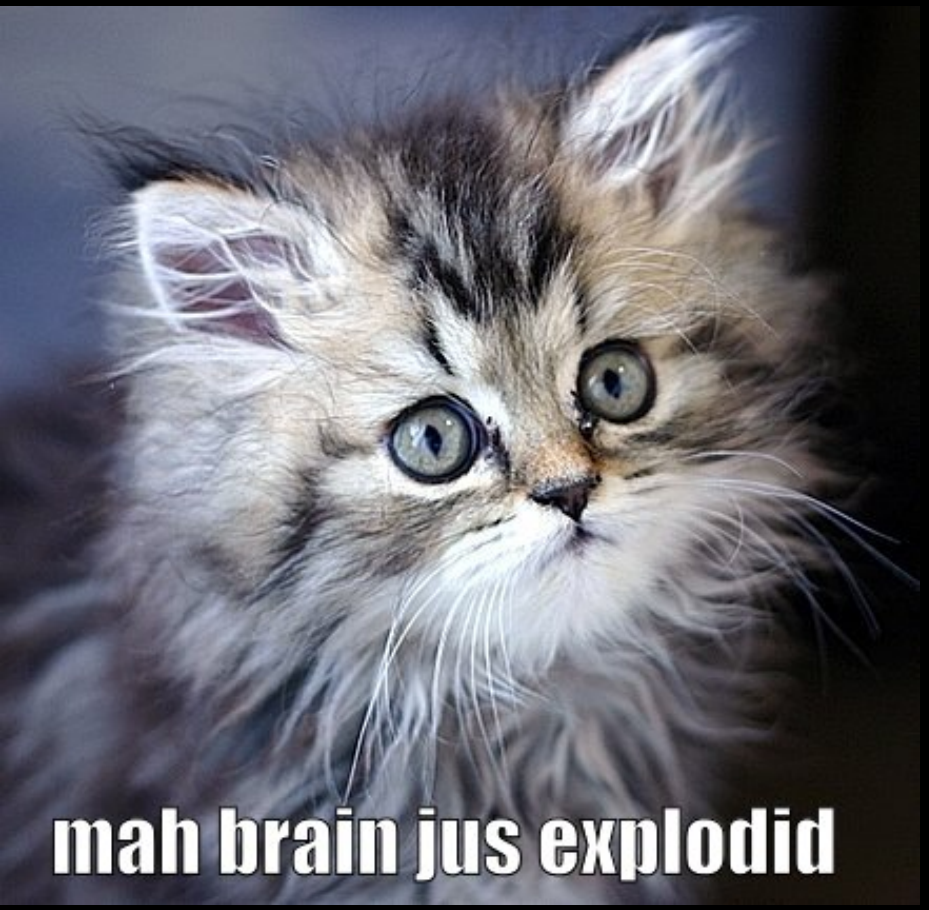
Transform the items emitted by a Collection into Collections, then flatten this into a single Collection



Top-rated Movies Collection

```
const 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*

Mouse Drags Collection

```
const getElementDrags = elmt => {
  DOM.mousedown(elmt)
    .flatMap( md =>
      dom.mousemove(document)
        .filter   .takeUntil(DOM.mouseup(elmt))
    )
};

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


Everything is a stream

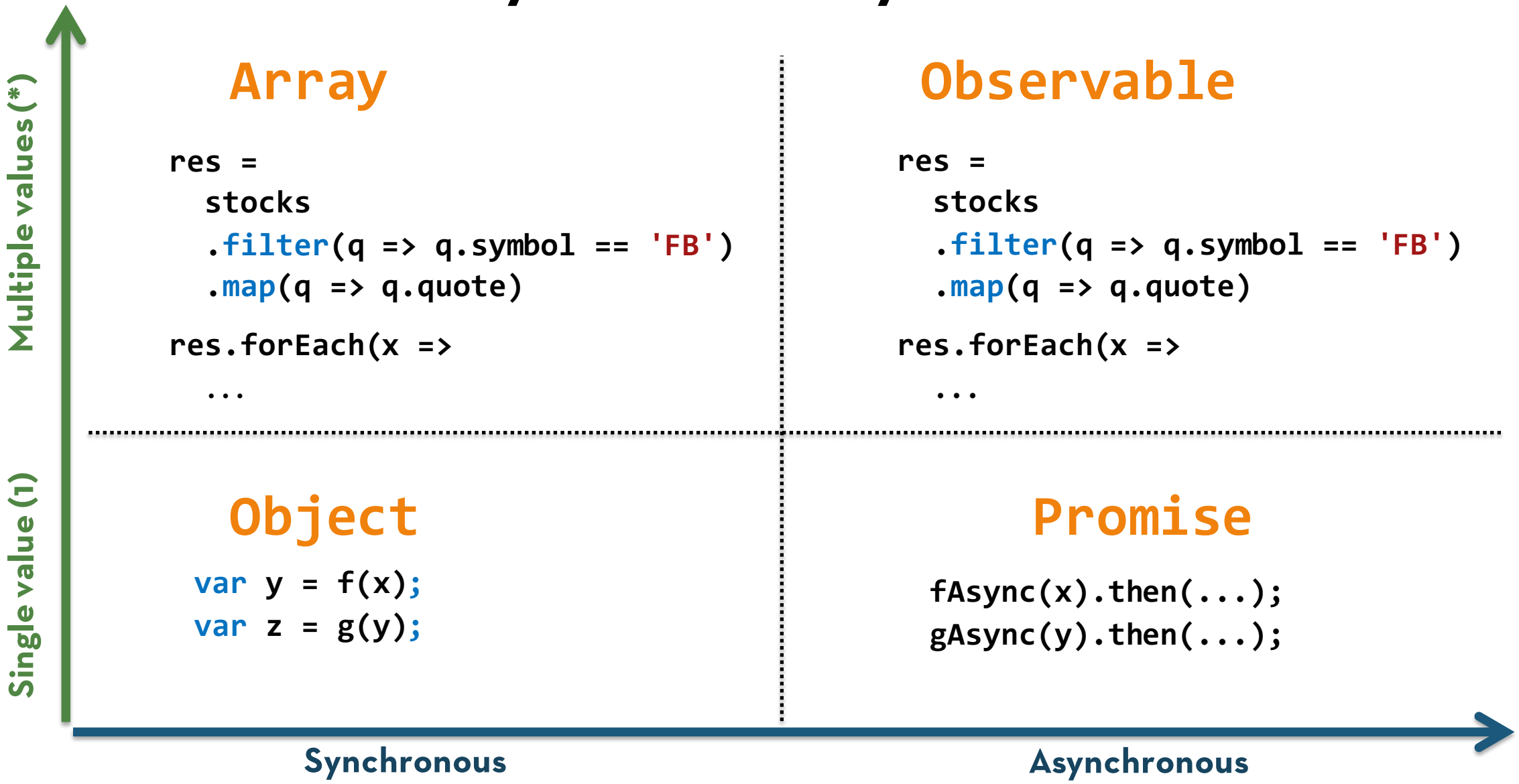
First-Class Asynchronous Values

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



Array

```
res =
  stocks
  .filter(q => q.symbol == 'FB')
  .map(q => q.quote)

res.forEach(x =>
  ...
```

Observable

```
res =
  stocks
  .filter(q => q.symbol == 'FB')
  .map(q => q.quote)

res.forEach(x =>
  ...
```

Object

```
var y = f(x);
var z = g(y);
```

Promise

```
fAsync(x).then(...);
gAsync(y).then(...);
```

Reactive Programming

React

What is Reactive Programming Anyhow?

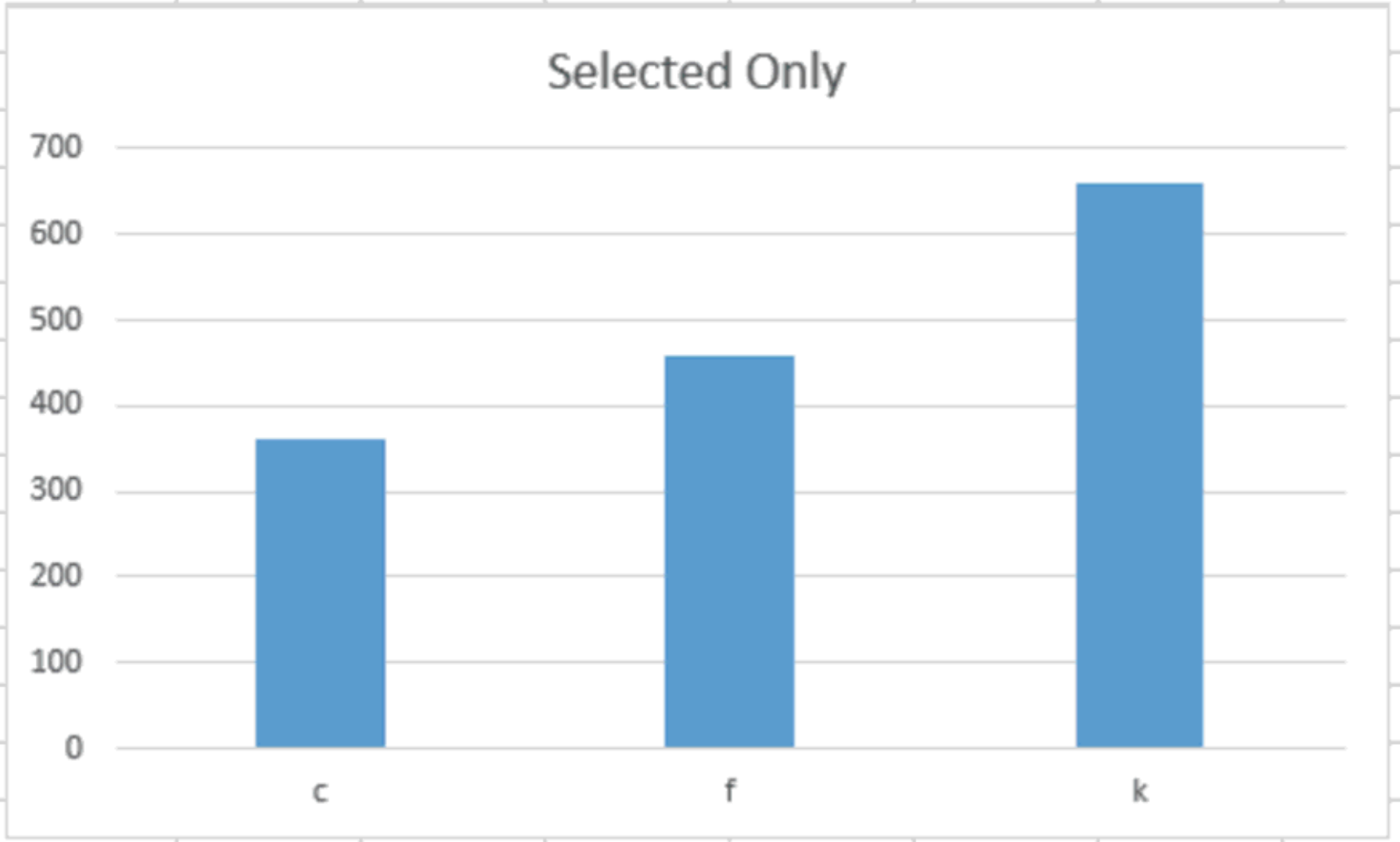
Merriam-Webster defines reactive as *“readily responsive to a stimulus”*, i.e. its components are “active” and always ready to receive events.

```
$( 'p' ).click(function() {
  $( this ).slideUp();
});
```

Clipboard | Font | Alignment | Number

T18

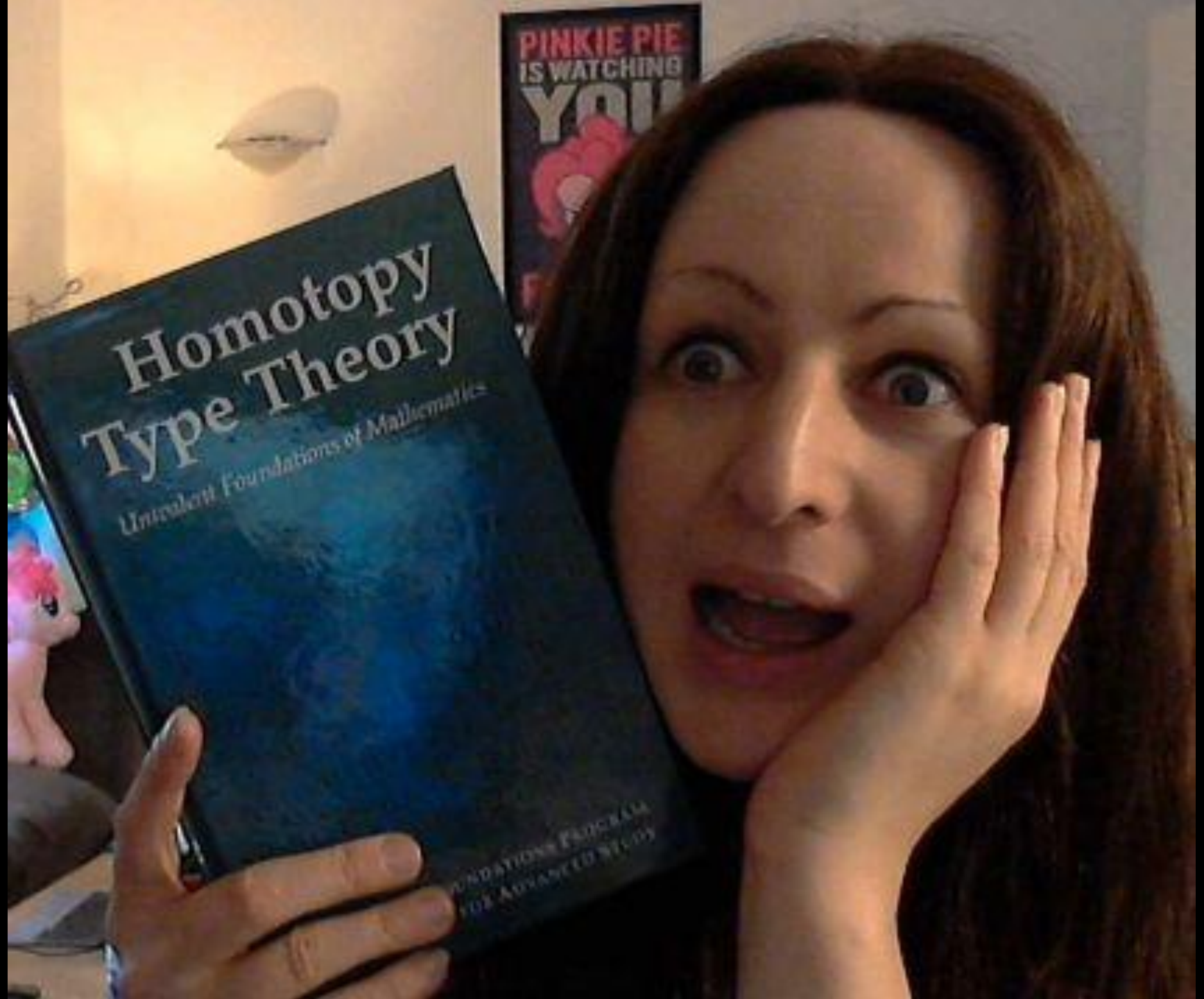
	A	B	C
1		2011	Show
2	a	330.33	
3	b	329.35	
4	c	359.85	Yes
5	d	376.94	
6	e	446.86	
7	f	457.7	Yes
8	g	509.38	
9	h	549.9	
10	i	566.84	
11	j	585.2	
12	k	658.24	Yes
13	l	720.2	
14			
15			
16			



Wanna really know what Reactive Programming Is?

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

http://bit.ly/reactive-paper


PINKIE PIE
IS WATCHING
YOU
Homotopy
Type Theory
Univalent Foundations of Mathematics

ComputerWissenschaftAkademischesPapierPhobie

Functional Reactive Programming (FRP) is...

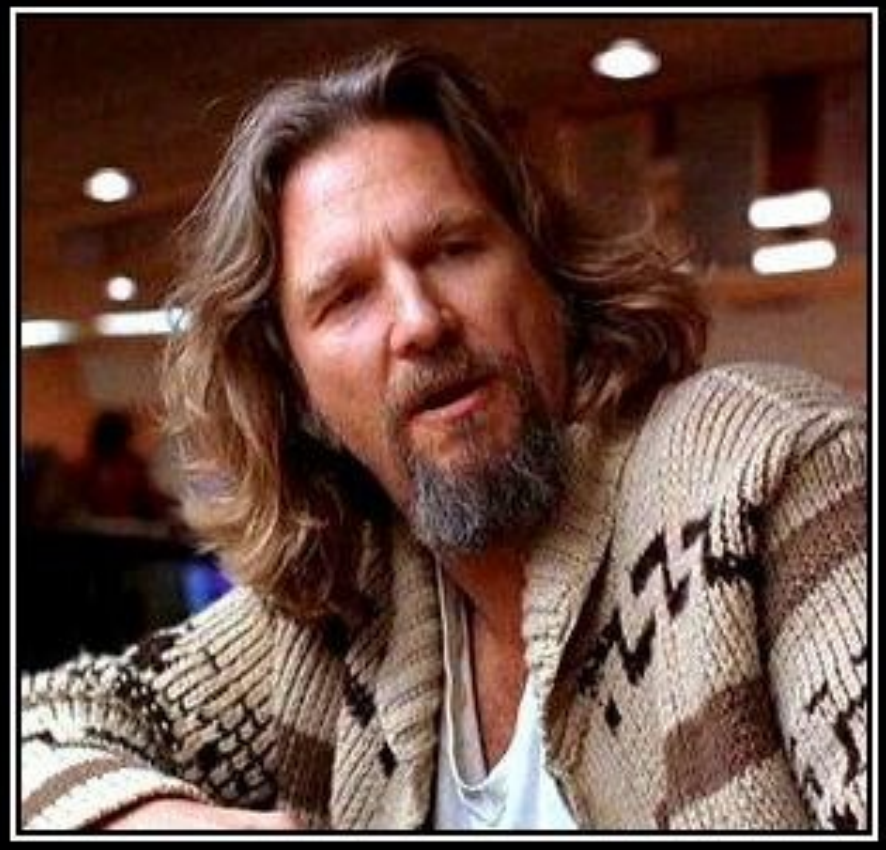
A concept consisting of

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

```
type Behavior a

μ = Behavior a -> (R -> a)
```

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

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.

Rx Grammar Police

onNext * (onError | onCompleted)?

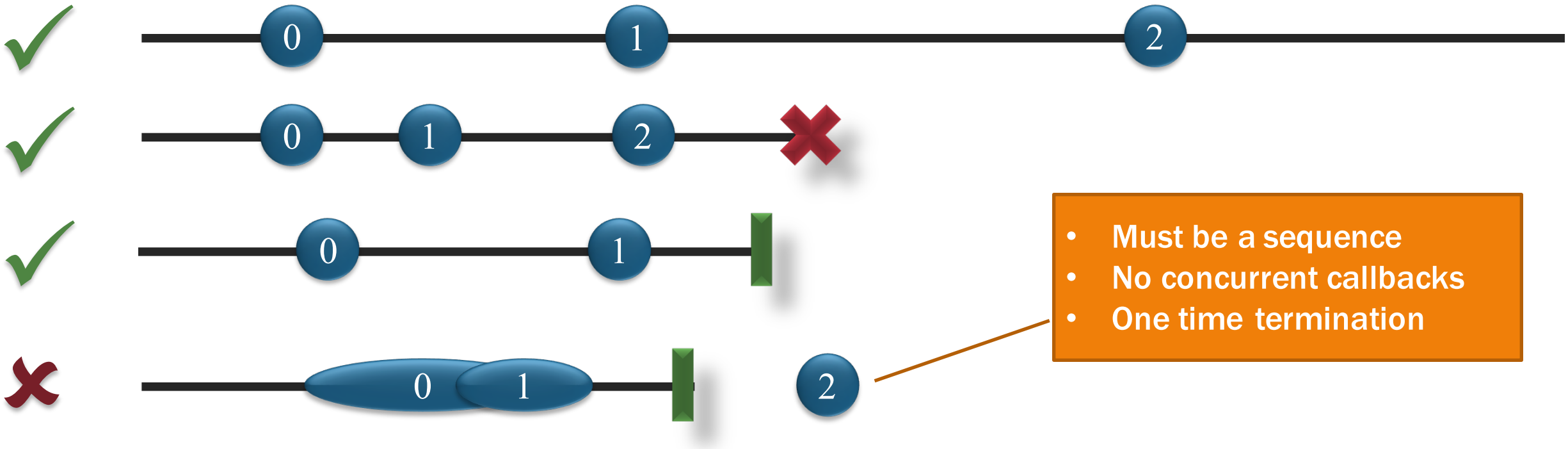
Zero or more values

E.g. events are ∞ sequences

Calls can fail

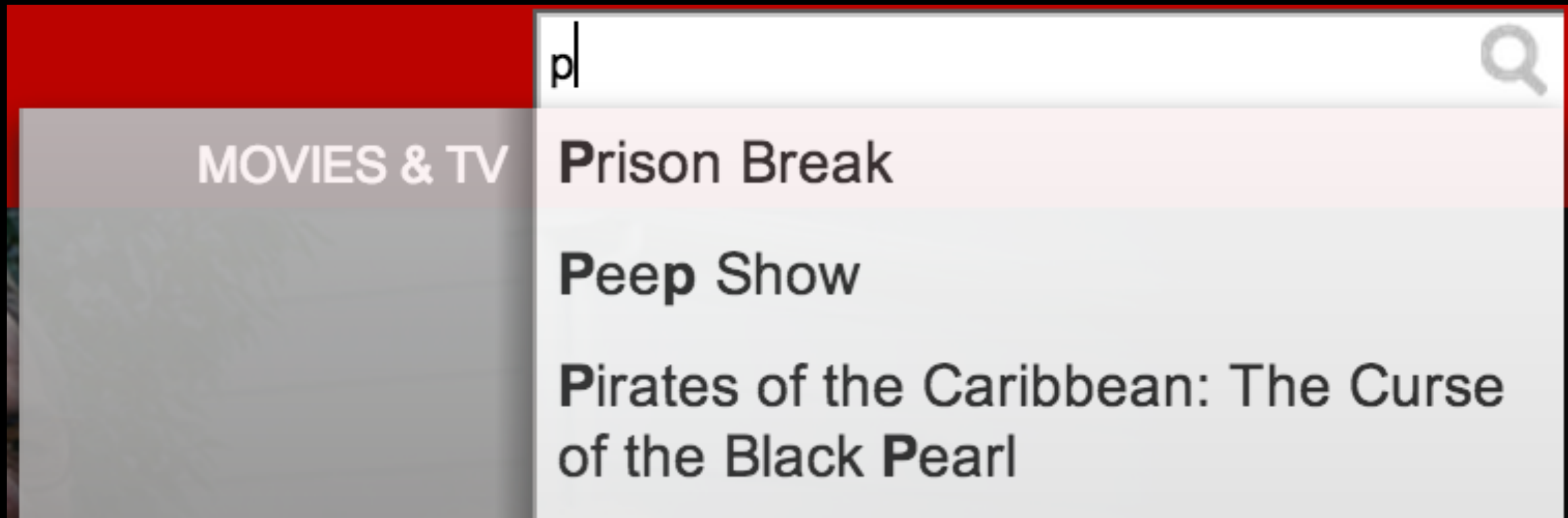
Resource management

Sequencing



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

Type Ahead Search...



Type Ahead Search with just jQuery

```
let $input = $('#textInput'), $results = $('#results');

// Keep track of old state
let jqXHR, currentText = '';

const handler = () => {
  let text = $(this).val();

  // Cancel previous if one already out there
  if (jqXHR && jqXHR.state() === 'pending') {
    jqXHR.reject({ statusText: 'abort' });
  }

  // Now check if text has changed
  if (text !== currentText) {
    currentText = text;
    $results.empty();

    jqXHR = searchWikipedia(currentText, 3).then(
      data => {
        var result = data[1];
        result.forEach( value =>
          $('<li>' + value + '</li>').appendTo($results);
        );
      },
      xhr => {
        if (xhr.statusText !== 'abort') {
          $('<li>' + xhr.statusText + '</li>').appendTo($results);
        }
      }
    );
  }
};

$input.keyup(debounce(handler, 500));
```

```
function searchWikipedia (term, numTimes) {
  var deferred = $.Deferred();

  (function makeRequest(num) {
    $.ajax({
        url: 'http://en.wikipedia.org/w/api.php',
        dataType: 'jsonp',
        data: {
          action: 'opensearch',
          format: 'json',
          search: term
        },
        success: (data, status, xhr) => {
          deferred.resolve(data, status, xhr);
        },
        error: (xhr) => {
          if (xhr.statusText !== 'abort') {
            if (num > 0) {
              return makeRequest(num - 1);
            } else {
              deferred.rejectWith(this, arguments);
            }
          }
        }
    });
  }(numTimes));

  return deferred;
}

function debounce(fn, wait) {
  let id;
  return () => {
    let args = arguments, context = this;
    if (id) { window.clearTimeout(id); }
    window.setTimeout(() => fn.apply(context, args), wait);
  };
}
```

Type Ahead 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

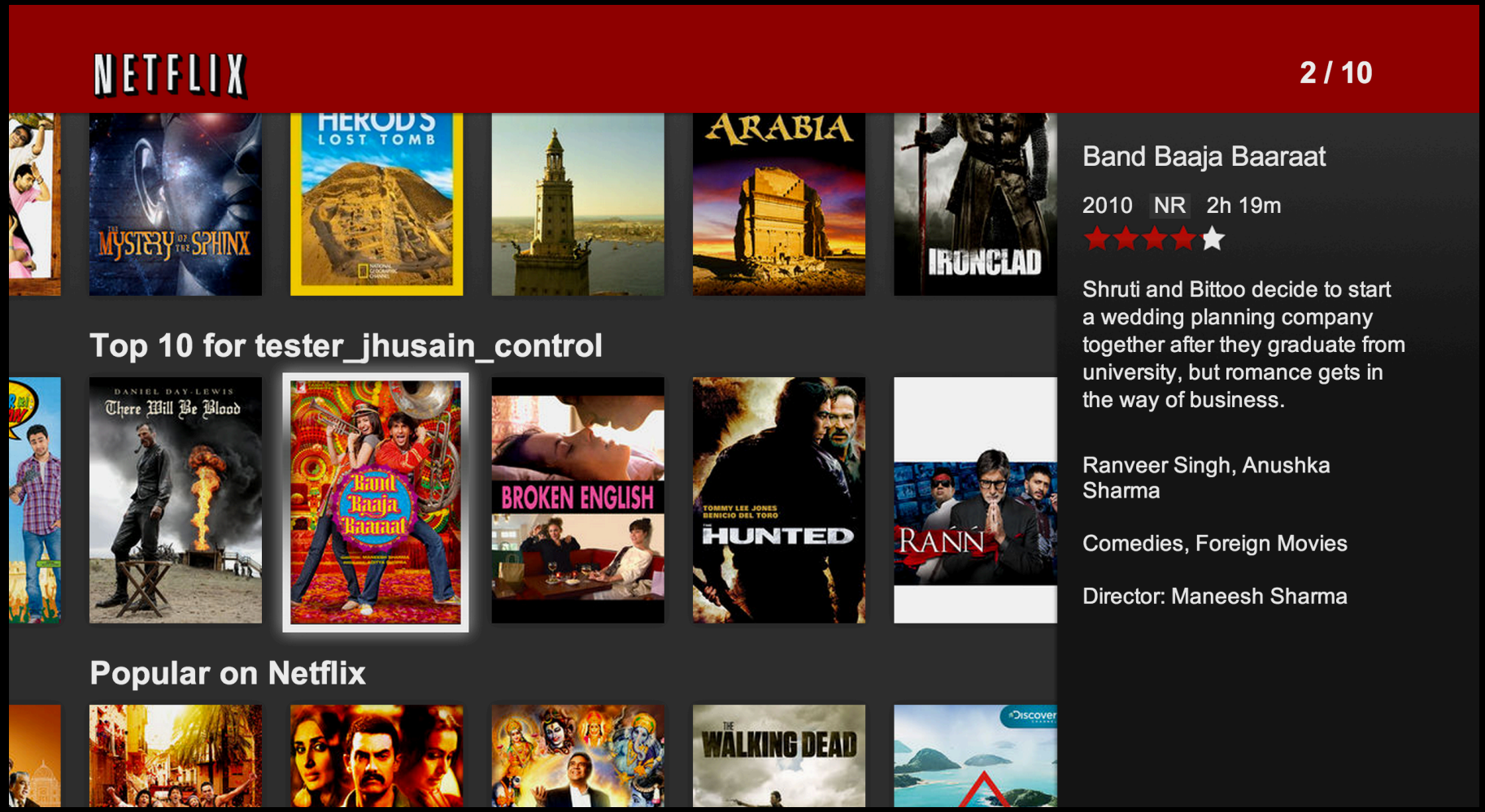
What exactly is Rx?

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

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(50)
            .map( scrollEvent => row.isVisible(scrollEvent.offset))
            .distinctUntilChanged()
            .share();
    let [rowShows, rowHides] = rowVisibilities.partition( 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.');
```


Oven Mitt
GRILLGLOVE
Professional Stunts
Do Not Attempt

Implementing Spell Check in Slack

```
let userStoppedTyping = inputEvent
 .concatMap(() => Rx.Observable.just(true).concat(Rx.Observable.never()))
 .takeUntil(inputEvent.debounce(750))
 .repeat()
 .startWith(true);

let currentKeyboardLanguage = userStoppedTyping
 .map(() => this.getCurrentKeyboardLanguage())
 .distinctUntilChanged()
 .merge(this.overrideKeyboardLanguage)
 .distinctUntilChanged();

let subscription = currentKeyboardLanguage.subscribe( lang => {
```


```
try {
  let res                                    a)
  for (le                                    cleanup())
    proce
  }
} catch (                                    a);
  throw e
} finally
  resourc
}
```

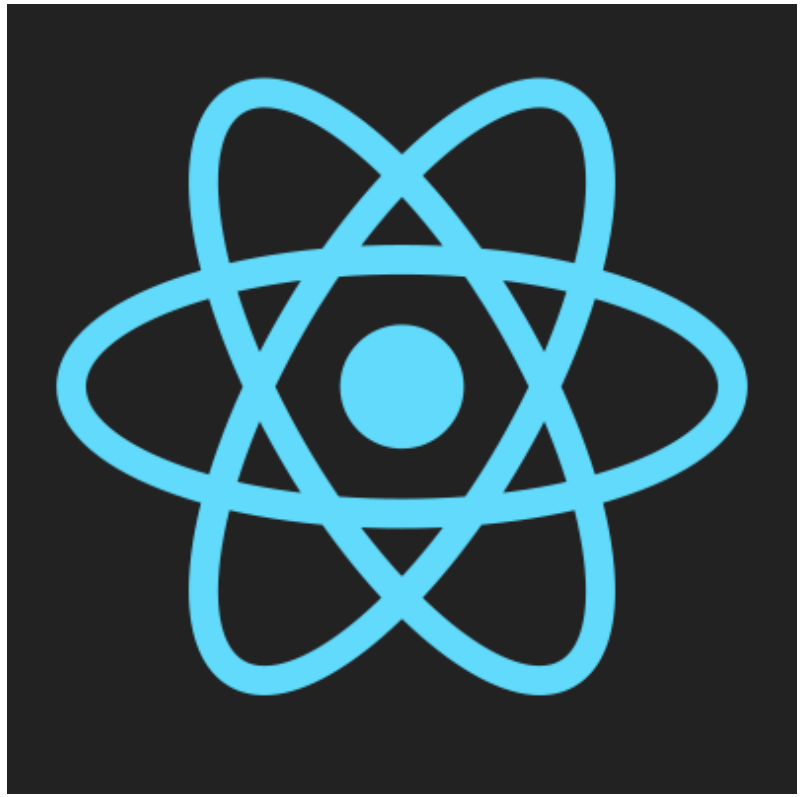
Observables and Backpressure

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



angular / angular

Watch 585 | Star 3,617 | Fork 909

feat(http): add basic http service

Browse files

```
This implementation only works in JavaScript, while the Observable transpilation
story gets worked out. Right now, the service just makes a simple request,
and returns an Observable of Response.

Additional functionality will be captured in separate issues.

Fixes #2028
```

master (#4)

jeffbcross authored on Apr 29 — 1 parent 363b9ba commit 21568106b114943b59ce37832d82c8fb9a63285b

Showing 35 changed files with 1,054 additions and 2 deletions.

Unified | Split

https://github.com/angular/angular/commit/21568106b114943b59ce37832d82c8fb9a63285b

Angular 2 and RxJS, perfect together?

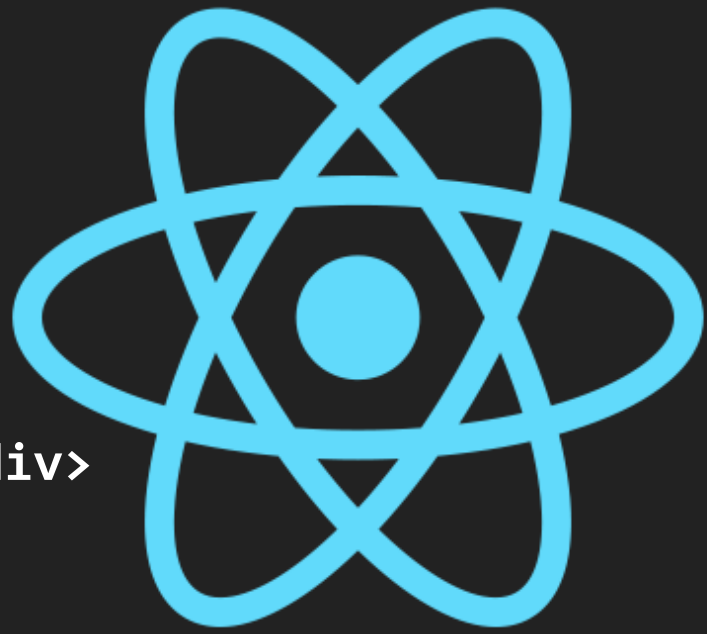
Data Libraries

Select any of the following data-related libraries that you will likely utilize on a majority of your Angular 2 projects: (choose multiple)

RxJS	38.7%	522
Immutable.js	34.7%	469
Firebase	33.3%	449
angular-meteor	23.8%	321
Falcor	14.5%	196
Other	9.1%	123
BaconJS	6.6%	89

http://angularjs.blogspot.com/2015/09/angular-2-survey-results.html

```
const Timer = React.createClass({
  getInitialState: () => {
    return {secondsElapsed: 0};
  },
  componentDidMount: () => {
    this.subscription = Rx.Observable.interval(1000)
      .subscribe(x => this.setState({secondsElapsed: x}));
  },
  componentWillUnmount: () => {
    this.subscription.dispose();
  },
  render: () => {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
});
```



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?

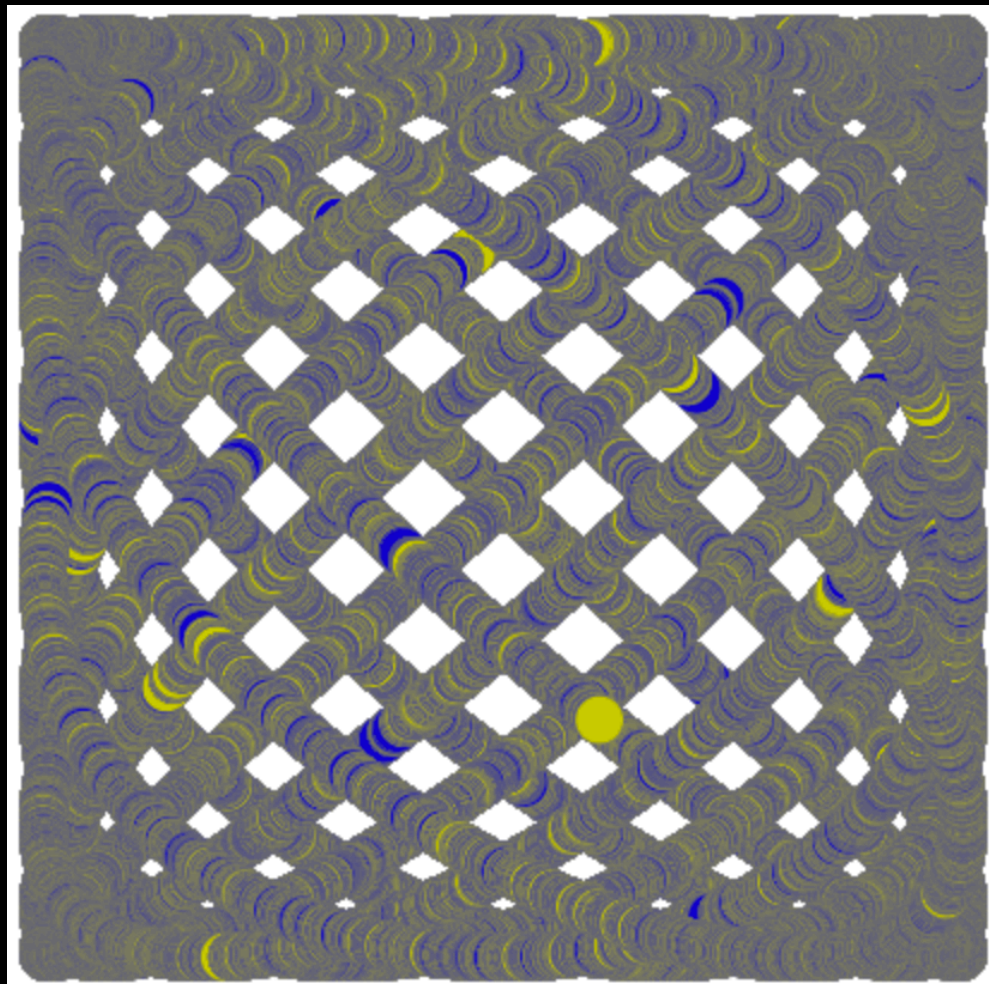
```
let d = scheduler.schedule(state,  () => {
  // Do work async
}, 1000);

d.dispose();
```

Testing async code: made easy!

```
let scheduler = new TestScheduler();


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


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


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

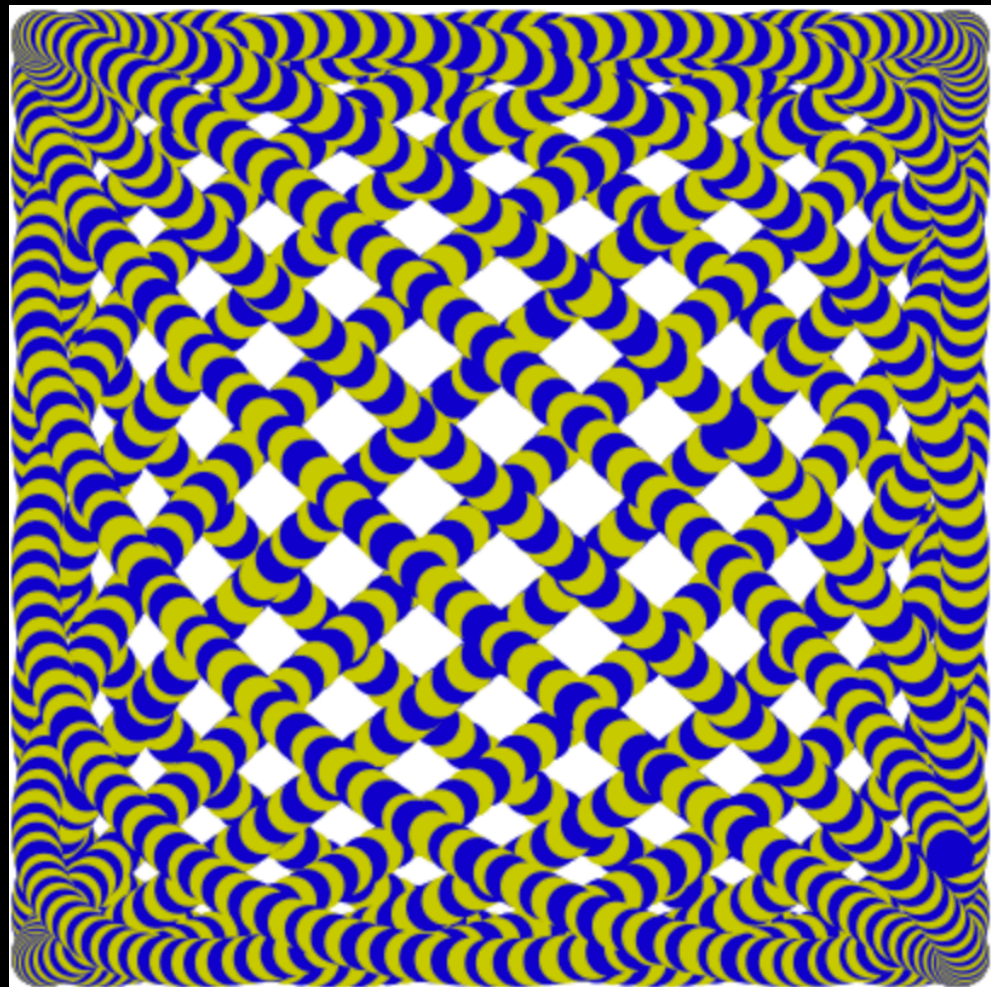
Schedulers Matter!

```
const scheduler = Scheduler.default;

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



No, Really, They Do!

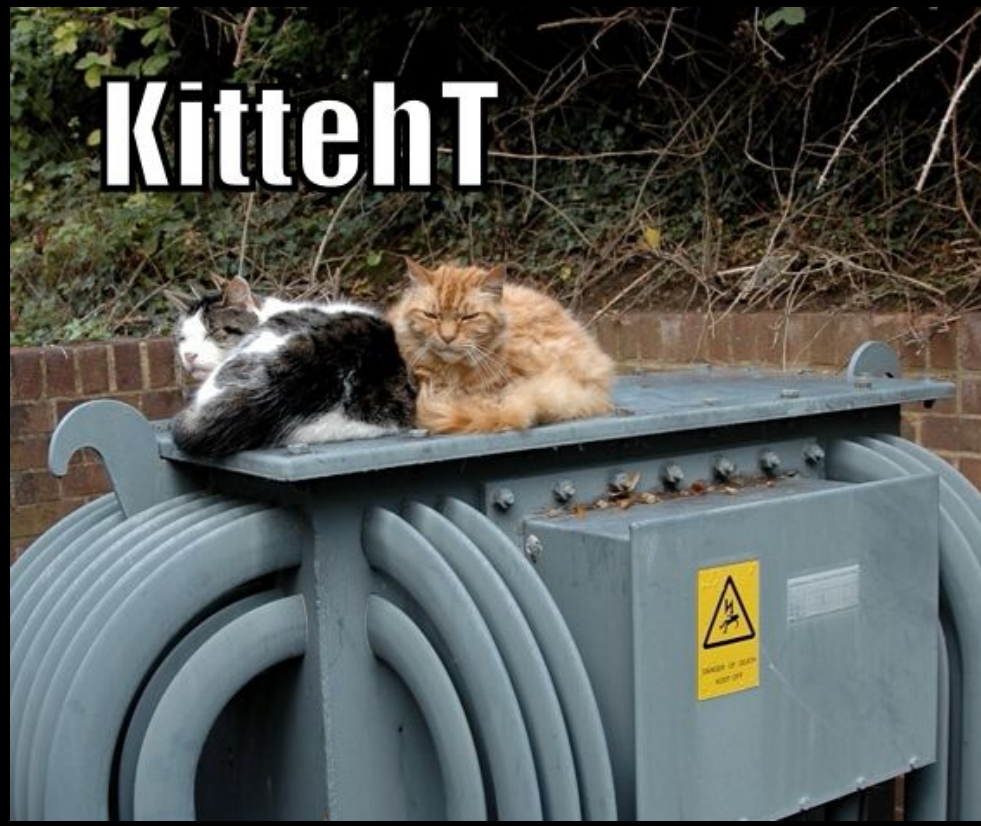
```
const scheduler = Scheduler.requestAnimationFrame;

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



ncawait

Async/Await with Observables and Generators...

```
Rx.Observable.spawn(function* () {
  const result = yield get('http://jsconf.co')
    .retry(3)
    .catch(cachedVersion)
    .finally(doCleanup);

  return result;
}).subscribe(::console.log);
```



Observables Coming to ES2016?!?!

ECMAScript Observable

This proposal introduces an **Observable** type to the ECMAScript standard library. The **Observable** type can be used to model push-based data sources such as DOM events, timer intervals, and sockets. In addition, observables are:

- *Compositional*: Observables can be composed with higher-order combinators.
- *Lazy*: Observables do not start emitting data until an **observer** has subscribed.
- *Integrated with ES6*: Data is sent to consumers using the ES6 generator interface.

> The **Observable** concept comes from *reactive programming*. See http://reactivex.io/ for more information.

Example: Observing Keyboard Events

Using the **Observable** constructor, we can create a function which returns an observable stream of events for an arbitrary DOM element and event type.

```
function listen(element, eventName) {
    return new Observable(sink => {
        // Create an event handler which sends data to the sink
        let handler = event => sink.next(event);

        // Attach the event handler
        element.addEventListener(eventName, handler, true);
```

https://github.com/zenparsing/es-observable

Observables in ES2016

```
function mouseDrags(element) {

    const moveStreams = listen(element, "mousedown").map(e => {

        e.preventDefault();

        return takeUntil(
            listen(element, "mousemove"),
            listen(document, "mouseup"));
    });

    return switchLatest(moveStreams);
}

const subscription = mouseDrags(document.body).subscribe({

    next(e) { console.log(`DRAG: <${ e.x }:${ e.y }>`) },
    throw(x) { console.log(`ERROR: ${ x }`) },
    return(x) { console.log(`COMPLETE: ${ x }`) },
});
```

https://github.com/zenparsing/es-observable/blob/master/demo/mouse-drags.js

http://github.com/Reactive-Extensions/RxJS
http://github.com/ReactiveX/RxJS

The Future of RxJS...