# Building Reactive Architectures

Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/jsday-2016

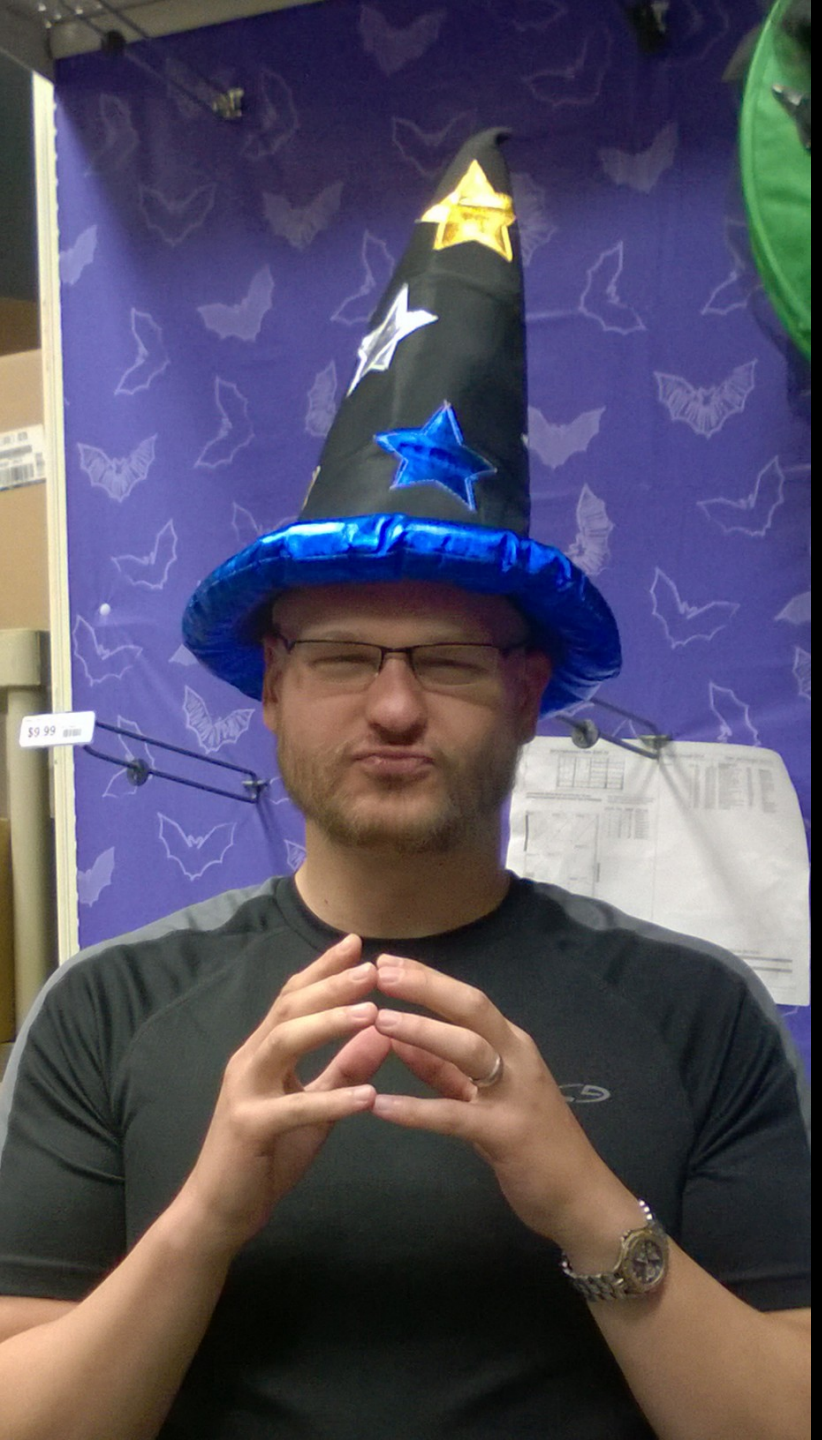
“jsDay 2016”.sup();

**λ Calrissian**
@mattpodwysocki

OH: "take me down to concurrency city where green pretty is grass the girls the and are"

| RETWEETS | LIKES |
| --- | --- |
| 1,137 | 624 |


3:30 PM - 24 Oct 2013



Principal SDE
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki

@ReactWindows

github.com/ReactWindows

# Reactive Extensions (Rx)

@ReactiveX

http://reactivex.io


MICROSOFT


Async is
Awful

# Living in 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();
    });
```

# Events and State, Oh My!

```
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);
```

# Promises are ONLY PART of a solution

then

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

# 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**

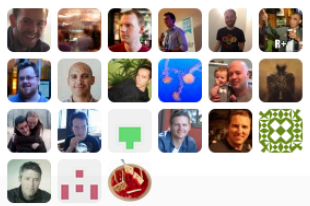

jakearchibald

**Notifications**

Subscribe

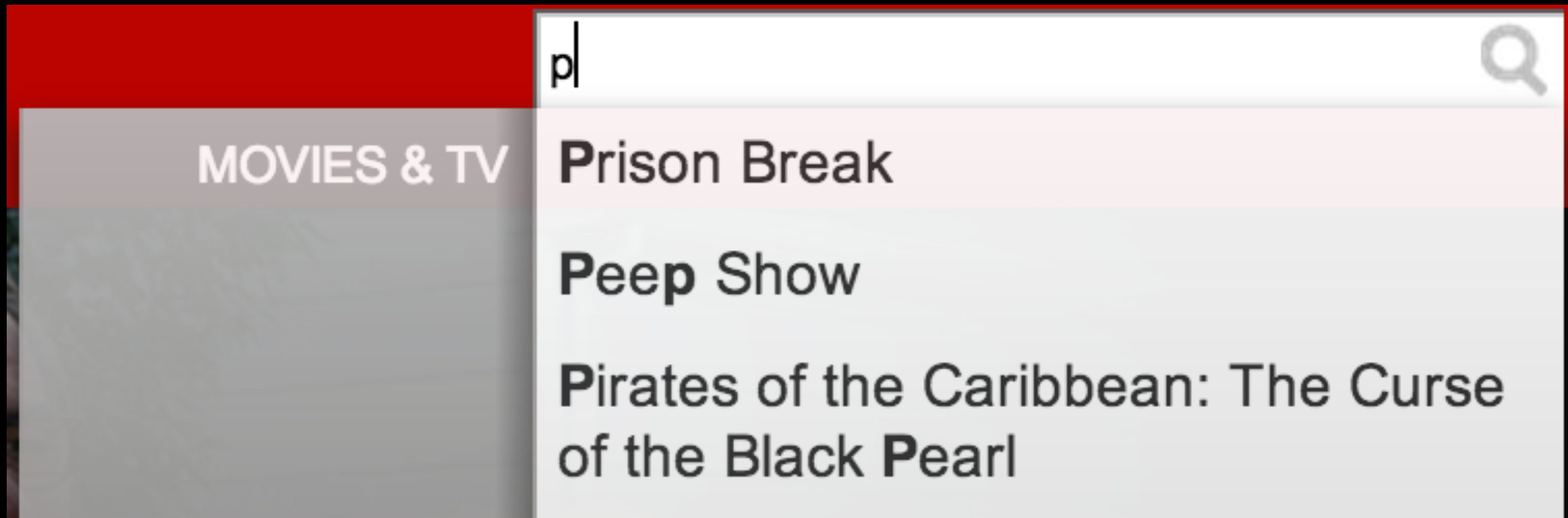
You're not receiving notifications from this thread.

**21 participants**



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

# Type Ahead Search...




jQuery
write less, do more.


Can we do
better?

“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 Solved the Reactive Way!

```
const authorizations =
    player.init().flatMap(() =>
        playAttempts.flatMap( movieId =>
            player.authorize(movieId)
                .retry(3)
                .takeUntil(cancels))
        )
  );

const subscription = authorizations.subscribe(
    license => player.play(license),
    error => showDialog('Sorry, can’t play right now.');
```

# Mouse Drags The Reactive Way

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

const subscription = getElementDrags(image)
  .subscribe(moveImage)
```

# Type Ahead Search The Reactive Way

```
const data = dom.keyup(input)
                .map(e => e.target.value)
                .debounceTime(500)
                .distinctUntilChanged()
                .switchMap(term => search(term).retry(3));

const subscription = data.subscribe(
  data => bindData(data),
  err =>  handleError(err));
```


Oven Mitt
GRILLGLOVE
Much
better!
Professional Stunts
Do Not Attempt

# Reactive Programming

# React

# An update on Object.observe

# Adam Klein (16 days ago) Reply View Original Go to Source Edit

Over three years ago, Rafael Weinstein, Erik Arvidsson, and I set out to design and implement what we believed to be the primitive underlying the data-binding system of MDV ("model-driven views"). We prototyped an implementation in a branch of V8, then got agreement from the V8 team to build a real version upstream, while pushing Object.observe ("O.o") as a part of the upcoming ES7 standard and working with the Polymer team to build their data-binding system on top of O.o.

Three years later, the world has changed in a variety of ways. While other data-binding frameworks (such as Ember and Angular) showed interest, it was difficult to see how they could evolve their existing model to match that of O.o. Polymer rewrote from the ground up for its 1.0 release, and in that rebuilding did not utilize O.o. And React's processing model, which tries to avoid the mutable state inherent in data-binding systems, has become quite popular on the web.

After much discussion with the parties involved, I plan to withdraw the Object.observe proposal from TC39 (where it currently sits at stage 2 in the ES spec process), and hope to remove support from V8 by the end of the year (the feature is used on 0.0169% of Chrome pageviews, according to chromestatus.com).

For developers who have been experimenting with O.o and are seeking a transition path, consider using a polyfill such as MaxArt2501/object-observe or a wrapper library like polymer/observe-js.

**https://esdiscuss.org/topic/an-update-on-object-observe**

# What is Reactive Programming Anyhow?

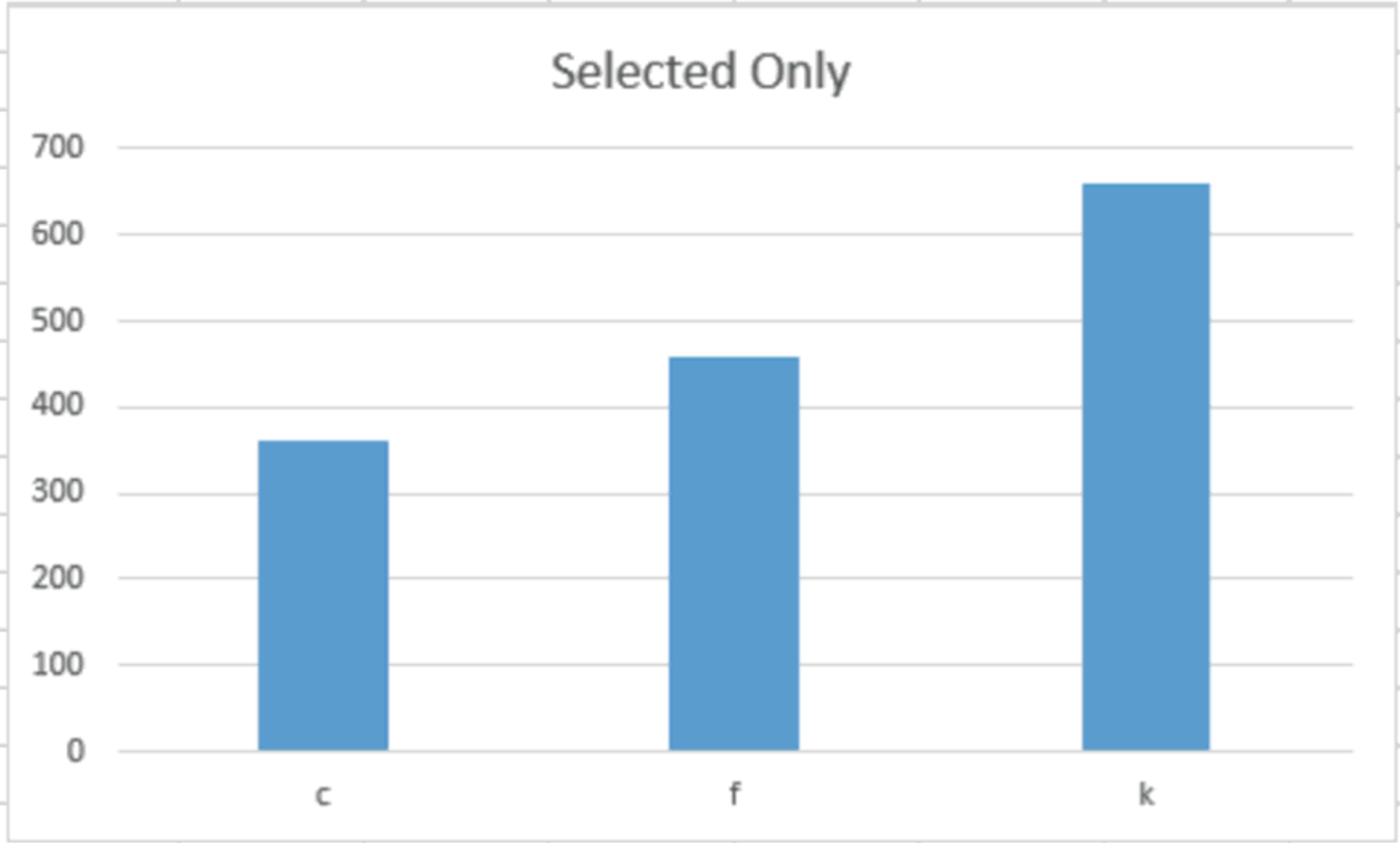
Merriam-Webster defines reactive as *“readily responsive to a stimulus”*, i.e. its components are “active” and always ready to receive events.

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

Clipboard | Font | Alignment | Number

T18

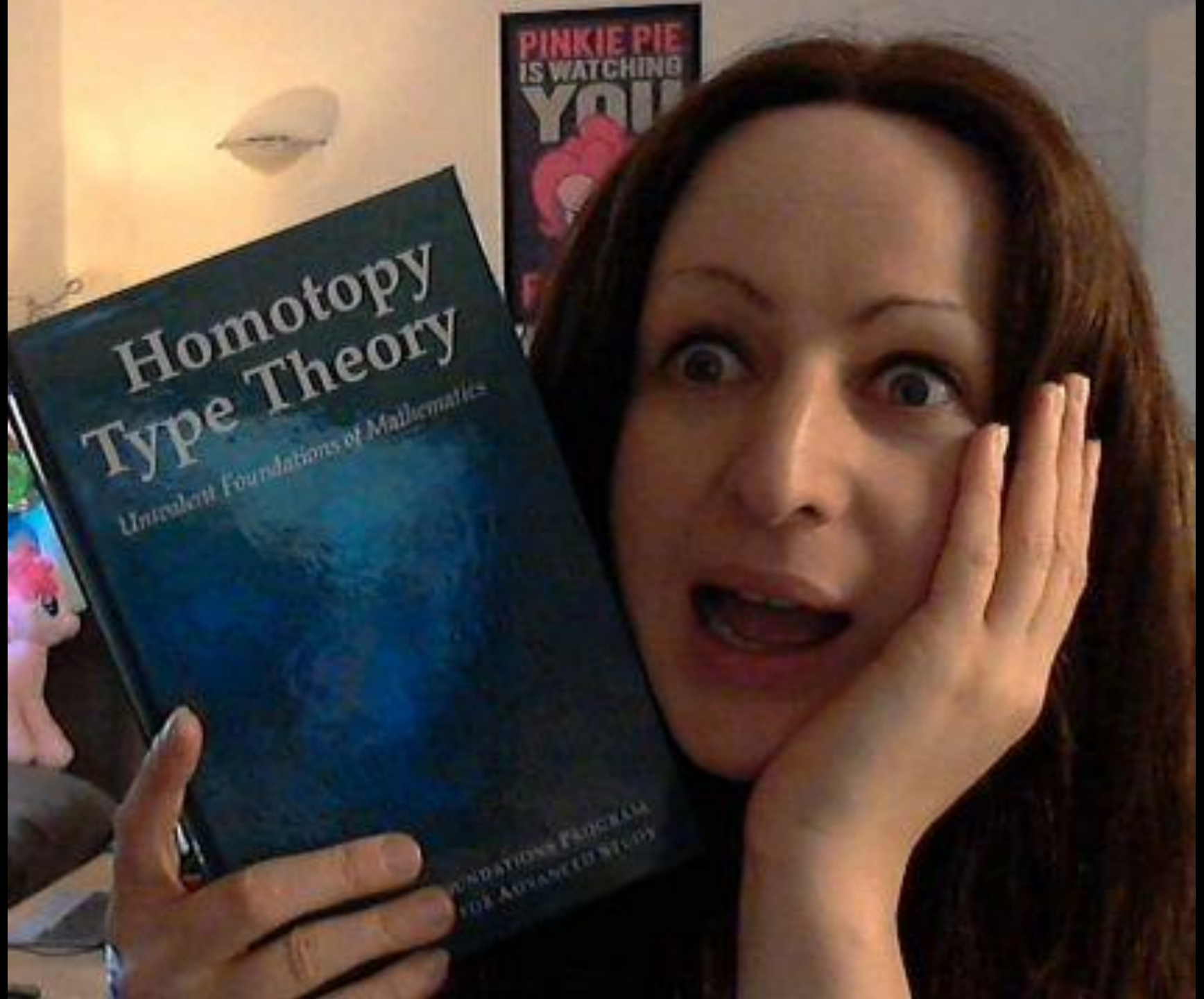
| | A | B | C |
|---|---|---|---|
| 1 | | 2011 | Show |
| 2 | a | 330.33 | |
| 3 | b | 329.35 | |
| 4 | c | 359.85 | Yes |
| 5 | d | 376.94 | |
| 6 | e | 446.86 | |
| 7 | f | 457.7 | Yes |
| 8 | g | 509.38 | |
| 9 | h | 549.9 | |
| 10 | i | 566.84 | |
| 11 | j | 585.2 | |
| 12 | k | 658.24 | Yes |
|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



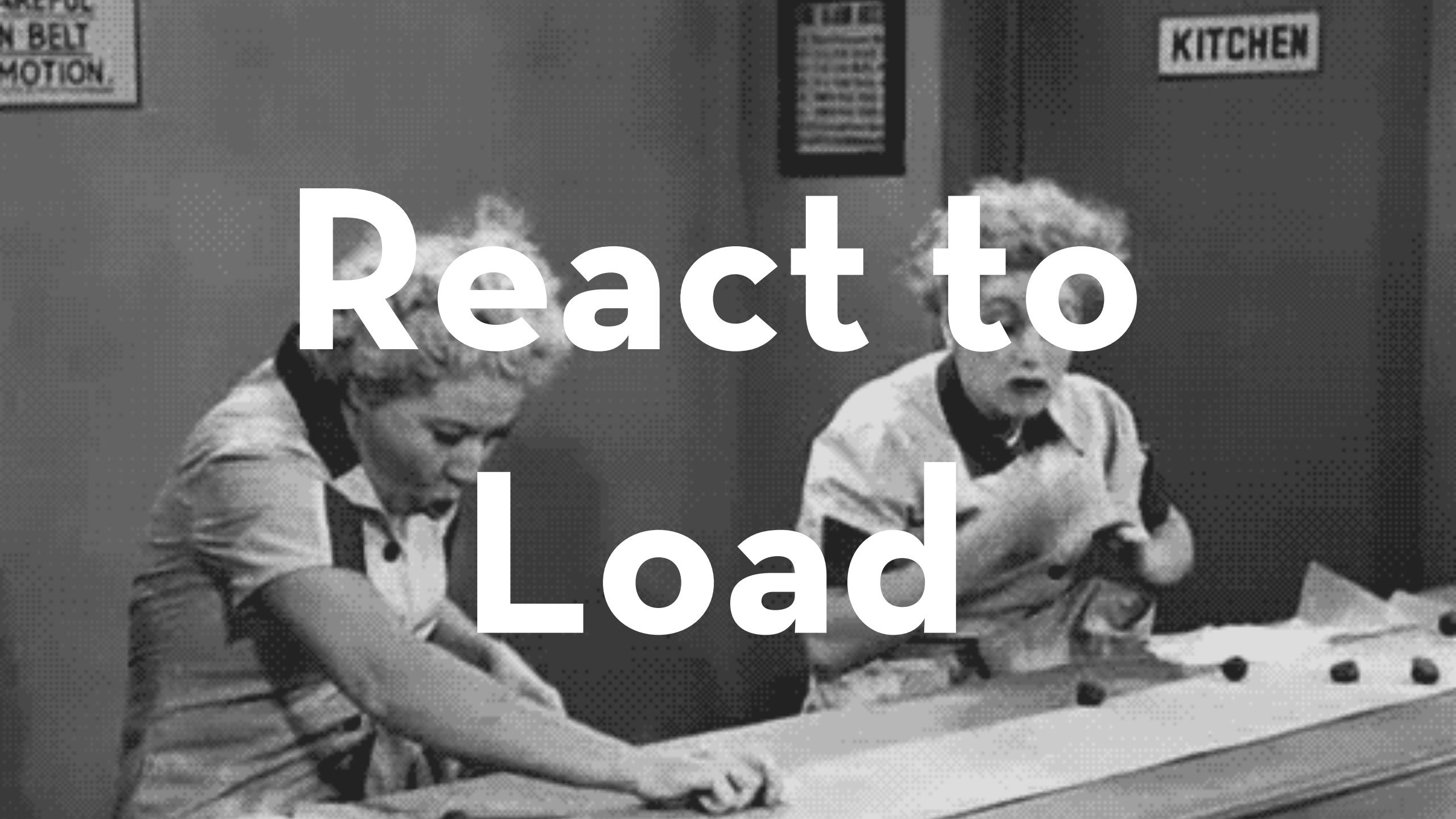
# Wanna really know what Reactive Programming Is?

**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




# React to Load

# Lossy

## Throttling input to a service...

```
const source = dom.keyup(document)
    .map(e => e.target.value)
    .debounceTime(500)
    .distinctUntilChanged()
    .switchMap(queryService);
```

## Sampling data at your own pace...

```
const sampler = randomTickEvent();

const source = tweetStream
   .sample(sampler)
   .groupBy(x => x.userId);
```


Lossless

## Buffering Your Data...

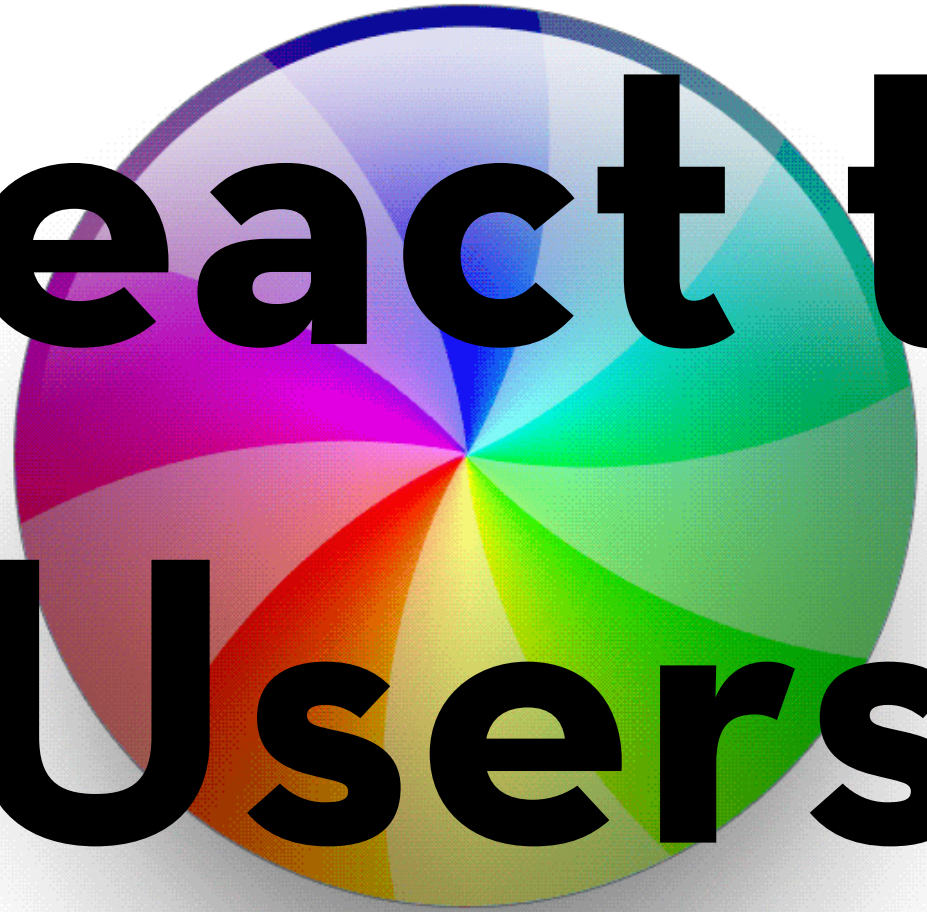
```
const source = tweetStream
  .bufferCount(100)

const source = tweetStream
  .bufferWhen(() =>
    Rx.Observable.interval(
      Math.floor(Math.random() * 400)
));
```


React to
Failure
next

# Handling Errors Gracefully

```
const source = tweetStream
  .retryWhen(a =>
    range(0, 3).zip(a, i => i)
      .flatMap(x => timer(i * 1000)
  )
  .catch(e => handleError(e))
```

# React to Users

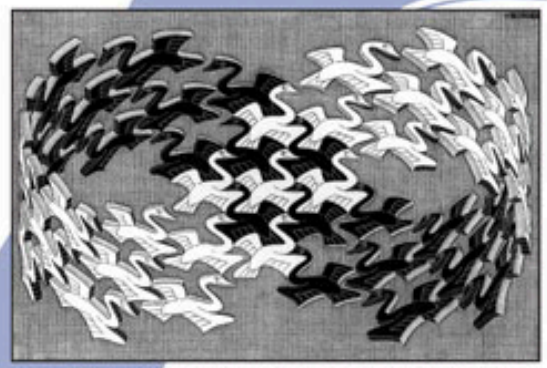



1994
ACE VENTURA
PET DETECTIVE
Sports Illustrated
SUPER METROID
KEANU REEVES
Dookie
CLERKS
A very funny look at the over-the-counter culture.
NIRVANA
UNPLUGGED
IN NEW YORK
weezer
AMERICA
Online

# Design Patterns

## Elements of Reusable Object-Oriented Software

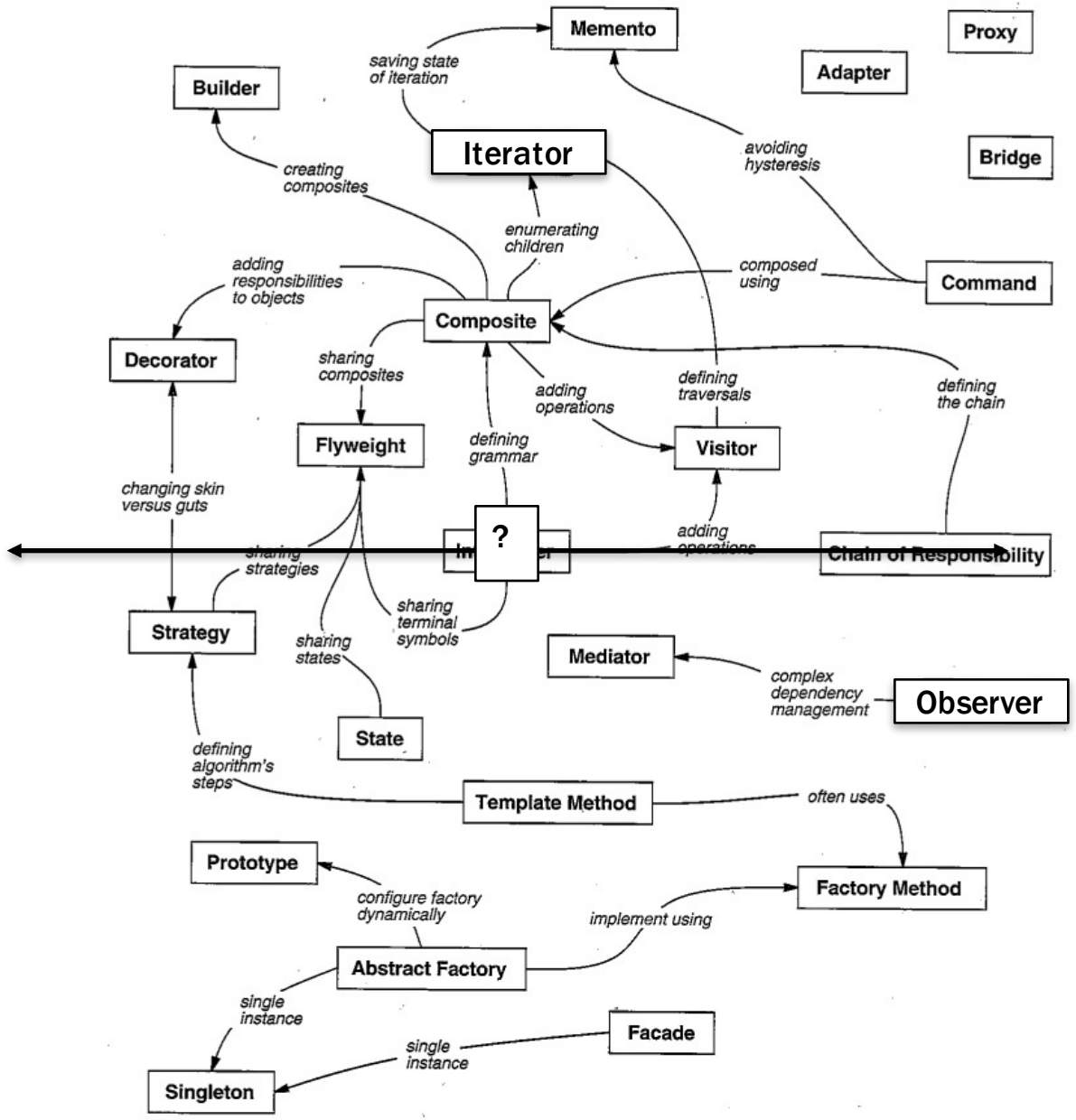
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides





Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES




Design Pattern Relationships

# “What’s the difference between an Iterable...

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

# ... and an Event?

Events and Arrays are *both* collections.

# 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 }
```

# Observable is Subject/Observer Done Right

```
interface Observable<T>
  subscribe(o: Observer<T>) => Subscription<T>
}
```

```
interface Observer<T>
  next: (value: T) => void;
  error: (err: any) => void;
  complete: () => void;
}
```

```
interface Subscription<T> {
  unsubscribe() : void;
}
```

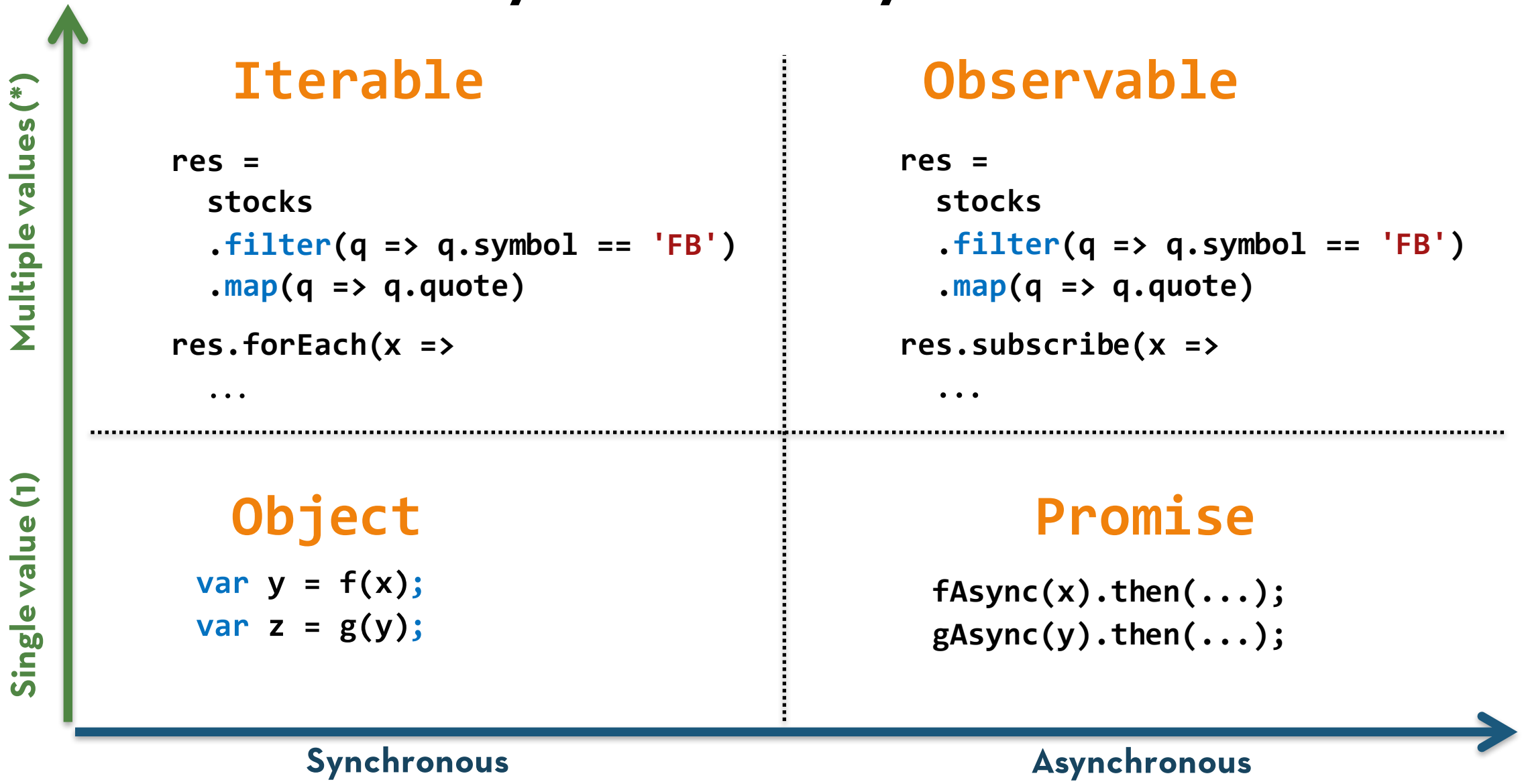
# First-Class Asynchronous Values

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)

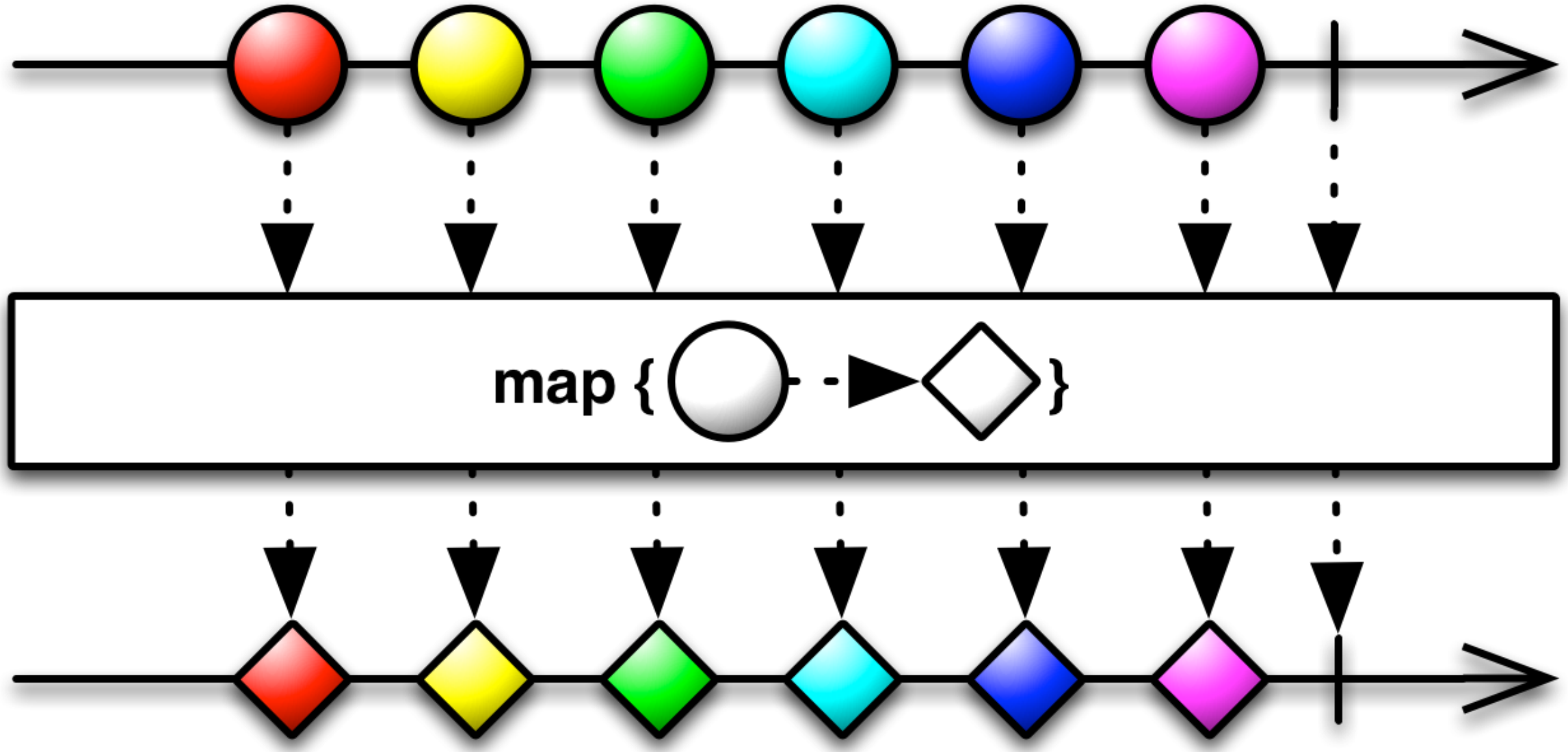
WIKIPEDIA
*The Free Encyclopedia*

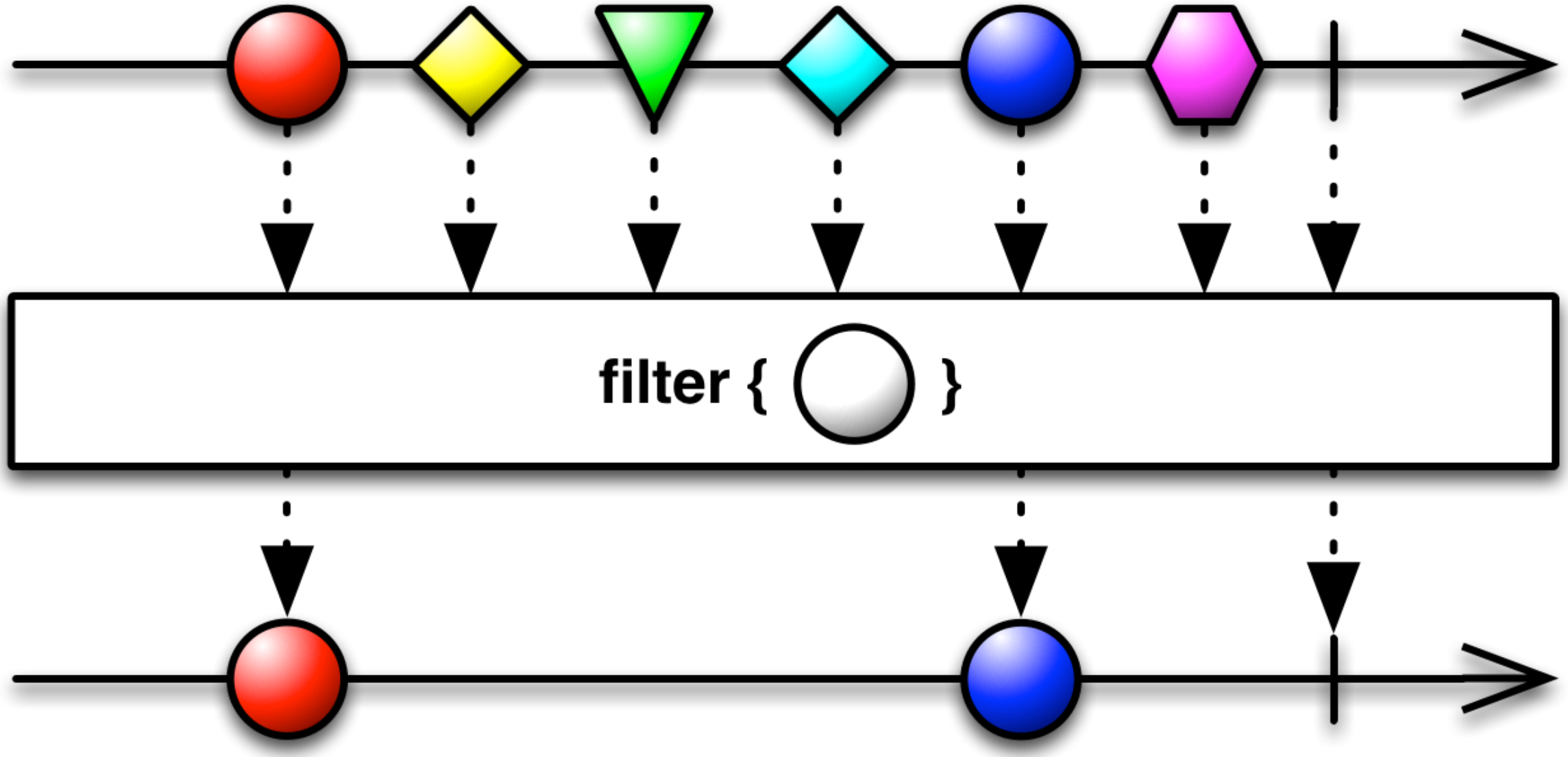
# The General Theory of Reactivity

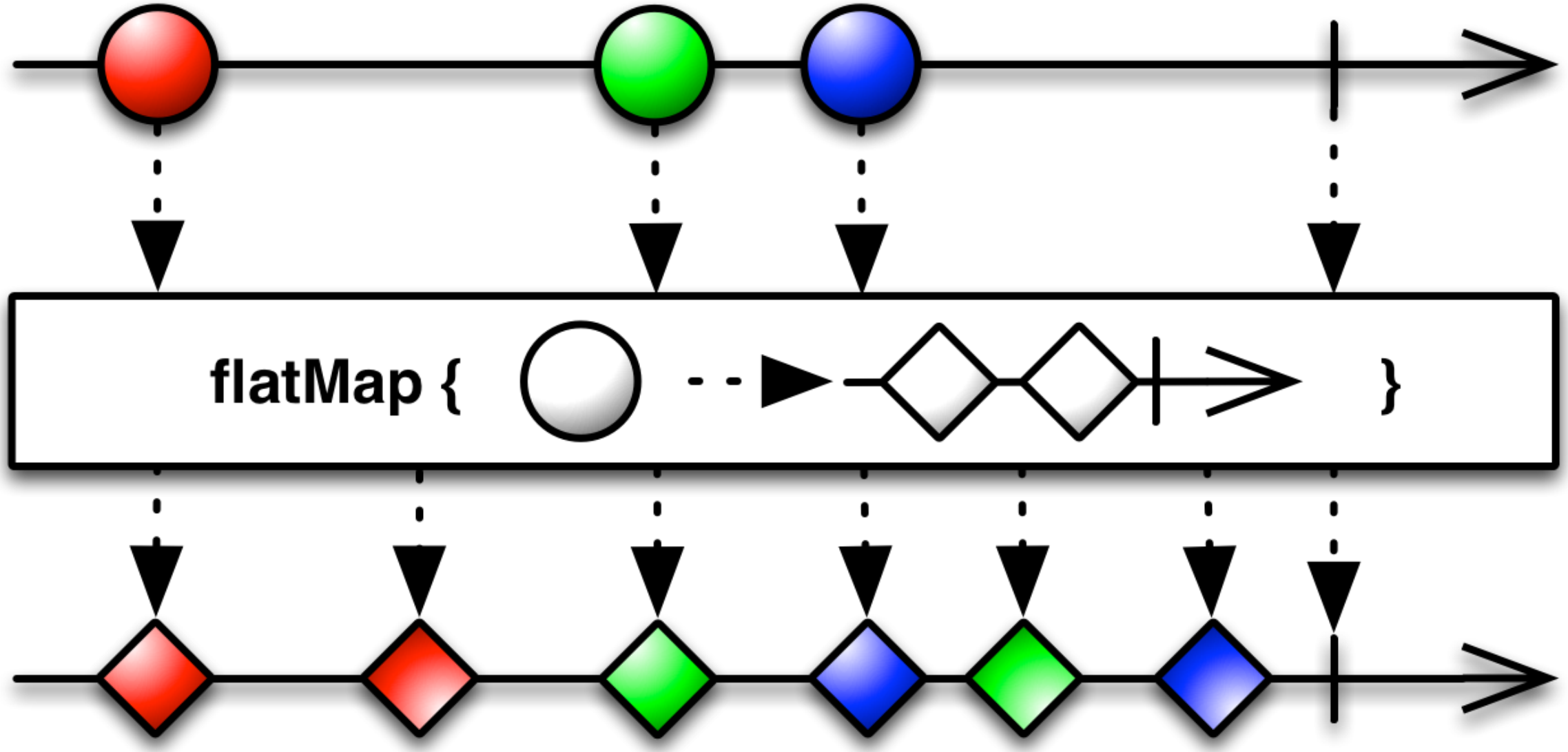



Everything is a stream

The majority of asynchronous code using Observables can be written with just a few *flexible* functions.


map {


filter { }


flatMap {
}


You Must
flatMap It!

# Reactive Programming


jQuery
write less, do more.

I thought to myself, I know, I'll solve this problem with a jQuery plugin. Now I have two problems...


Knockout.


ANGULARJS
by Google

# The Future is Here...


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)

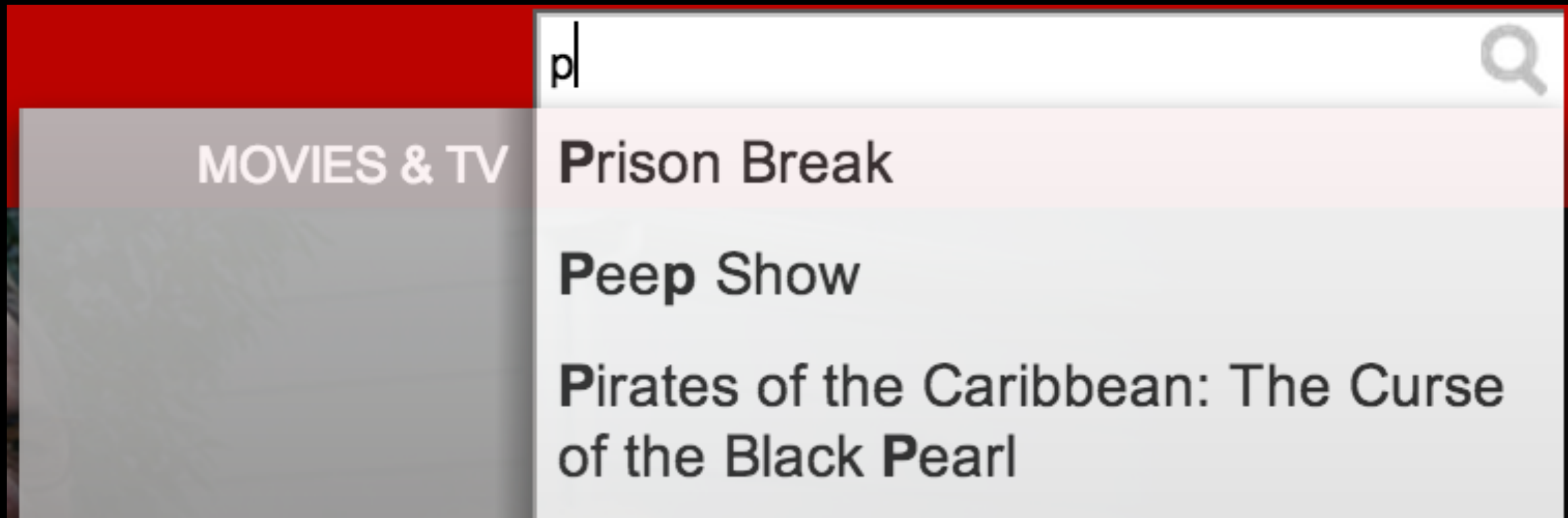
jeffbcross authored on Apr 29 — 1 parent 363b9ba commit 21568106b114943b59ce37832d82c8fb9a63285b

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

Unified | Split

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

# Type Ahead Search...



```
@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Wikipedia Search</h2>
      <input type="text" [ngFormControl]="term"/>
      <ul>
        <li *ngFor="#item of items | async">{{item}}</li>
      </ul>
    </div>
  `
})
```

```
import {Injectable} from 'angular2/core';
import {URLSearchParams, Jsonp} from 'angular2/http';

@Injectable()
export class WikipediaService {
  constructor(private jsonp: Jsonp) {}

  search (term: string) {

  }
}
```

```
search (term: string) {
  var url =
    'http://en.wikipedia.org/w/api.php?callback=JSONP_CALLBACK';
  var search = new URLSearchParams()
  search.set('action', 'opensearch');
  search.set('search', term);
  search.set('format', 'json');

  return this.jsonp
    .get(url, { search })
    .map((response) => response.json()[1]);
}
```

```
export class App {
  items: Observable<Array<string>>;
  term = new Control();
  constructor(private service: WikipediaService) {
    this.items = this.term.valueChanges
                  .debounceTime(400)
                  .distinctUntilChanged()
                  .switchMap(term =>
                    this.service.search(term));
  }
}
```

# ngrx

Reactive Extensions for angular2

Repositories | People 1

Filters | Find a repository…

## devtools

TypeScript ★ 13 3

Developer Tools for ngrx

Updated 16 hours ago

## store

TypeScript ★ 432 43

RxJS powered state management for Angular2 apps, inspired by Redux

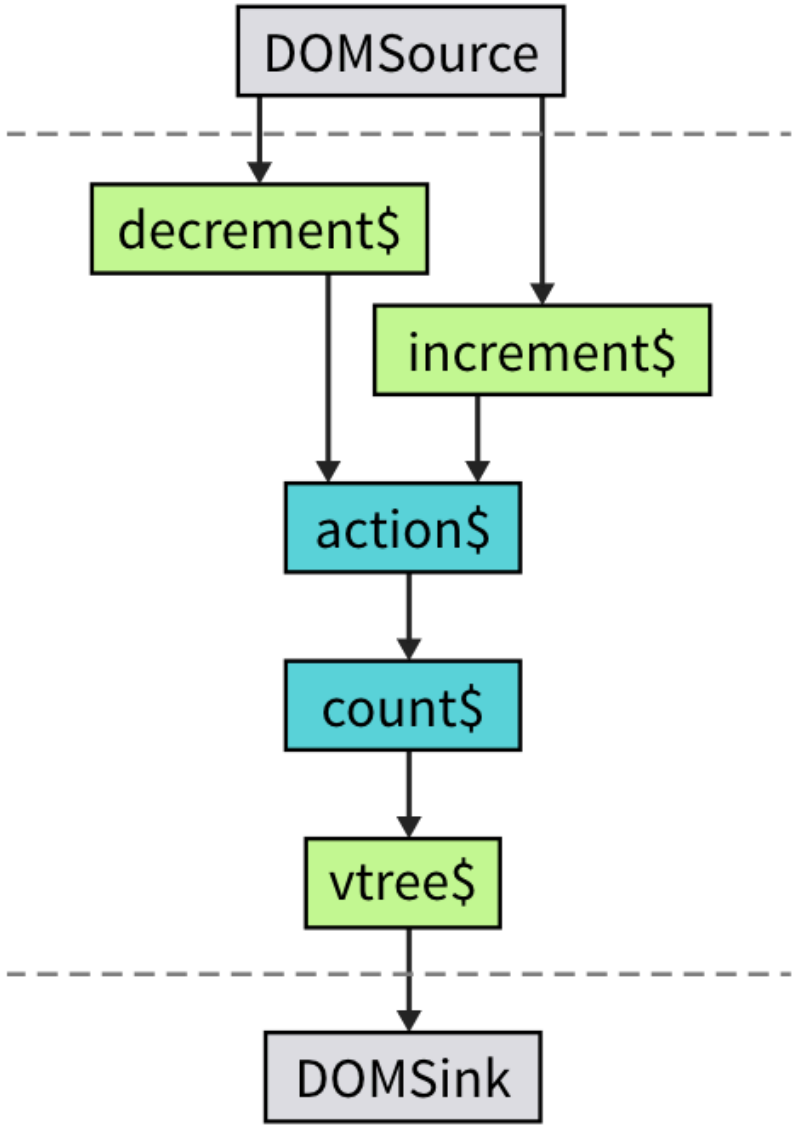
Updated 2 days ago

https://github.com/ngrx

# Cycle.js

A fully reactive JavaScript framework for Human-Computer Interaction

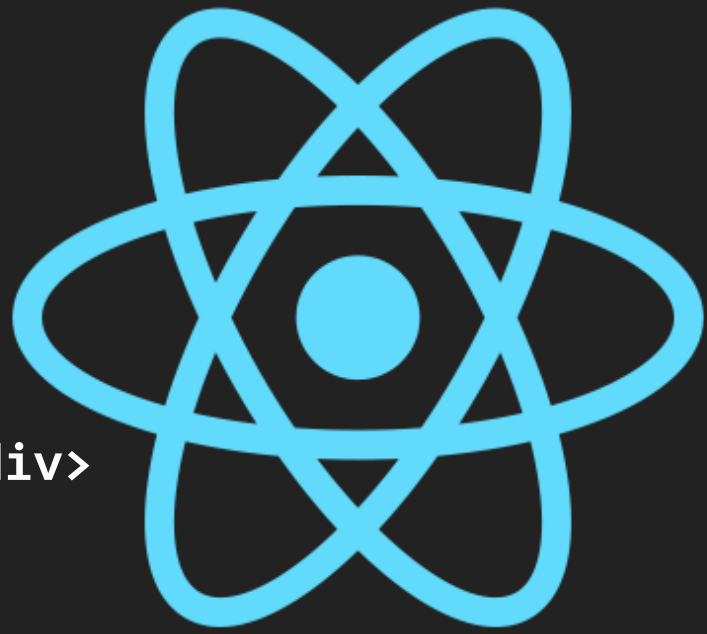
Get started




```
function main(sources) {
  const decrement$ = sources.DOM
    .select('.decrement').events('click').map(ev => -1);

  const increment$ = sources.DOM
    .select('.increment').events('click').map(ev => +1);

  const action$ = Observable.merge(decrement$, increment$);
  const count$ = action$.startWith(0).scan((x,y) => x+y);

  const vtree$ = count$.map(count =>
    div([
      button('.decrement', 'Decrement'),
      button('.increment', 'Increment'),
      p('Counter: ' + count)
    ])
  );
  return { DOM: vtree$ };
}
```

# React

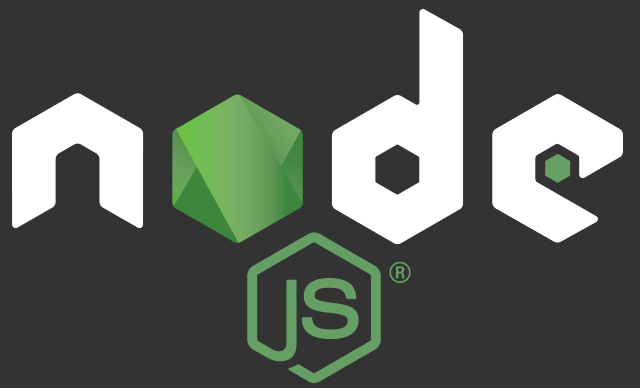
```
const Timer = React.createClass({
  getInitialState: () => {
    return {secondsElapsed: 0};
  },
  componentDidMount: () => {
    this.subscription = Rx.Observable.interval(1000)
      .subscribe(x => this.setState({secondsElapsed: x}));
  },
  componentWillUnmount: () => {
    this.subscription.dispose();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




https://github.com/philpl/fluorine

```
// Create a dispatcher which is our event stream
const dispatcher = createDispatcher();

// Agendas, if fails, all is reverted
const addIfFetchSucceeds = Observable
  .of({ type: 'ADD' })
  .concat(Observable
    .of('/ping')
    .flatMap(fetch)
  );

dispatcher.schedule(addIfFetchSucceeds);
```

```
const requests_ = new Rx.Subject();

function sendHello(e) {
  e.res.writeHead(200, { 'Content-Type': 'text/plain' });
  e.res.end('Hello World\n');
}

requests_
  .do(e => console.log('request to', e.req.url))
  .subscribe(sendHello)

http.createServer((req, res) => {
  requests_.next({ req: req, res: res });
})
```



# Observables Coming to ES2017?!?!

## 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 ES2017

```
function commandKeys(element) {
    const keyCommands = { "38": "up", "40": "down" };

    return listen(element, "keydown")
        .filter(event => event.keyCode in keyCommands)
        .map(event => keyCommands[event.keyCode])
}

const subscription = commandKeys(inputElement).subscribe({
    next(val) { console.log("Received key command: " + val) },
    error(err) { console.log("Received an error: " + err) },
    complete() { console.log("Stream complete") },
});

subscription.unsubscribe();
```


@ReactiveX

http://reactivex.io

Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/ng-conf-2016