Async JavaScript at Netflix, Microsoft and the World

Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/scotlandjs-2016

'ScotlandJS 2016'.sup();

Take me down to the concurrency city where green pretty is grass the code the and are...

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, Microsoft and others solved

BIG async problems

by thinking differently about

Events.

The Most Apps are Asynchronous

App Startup

Player

Data Access

Animations

View/Model Binding

Asynchronous Nightmares

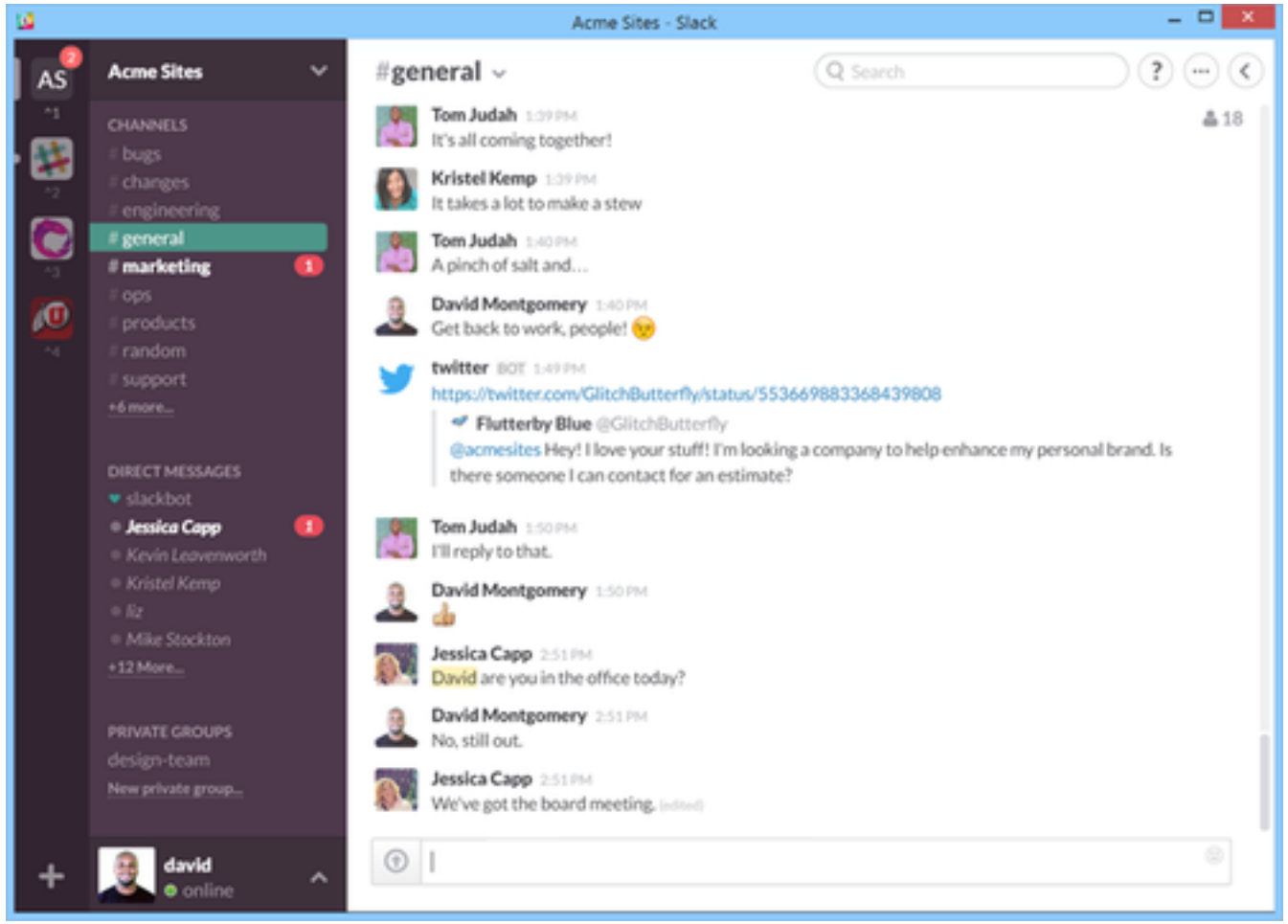
Memory Leaks

Race Conditions

Callback Hell

Complex State Machines

Disjointed Error Handling




2014


Async is
Awful

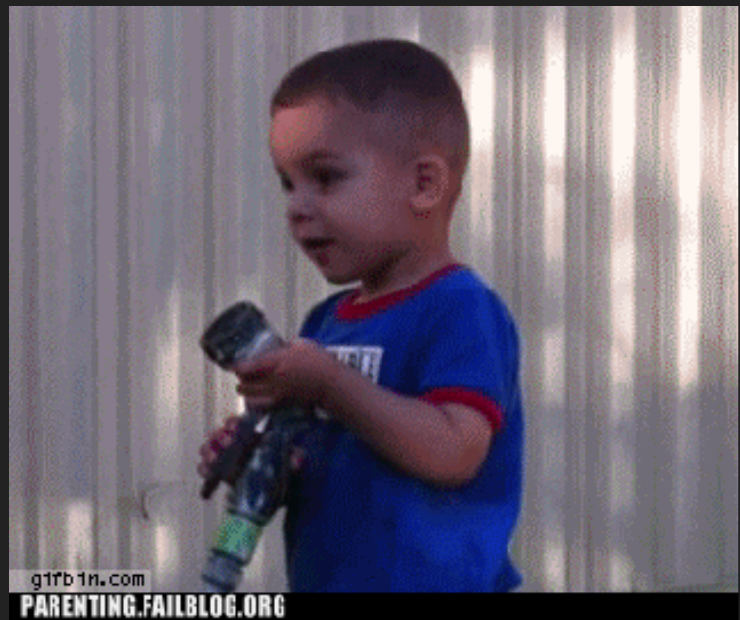
Living in Callback Hell

```
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

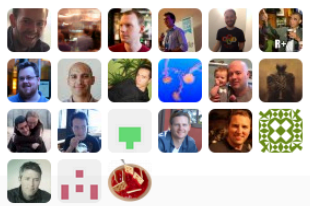

jakearchibald

Notifications

Subscribe

You're not receiving notifications from this thread.

21 participants



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

Canceled as the third state

This document outlines why I think it's important for cancelation to be represented as a third promise state, alongside fulfillment and rejection.

It's important to note that the discussions here are independent of *how* the promise gets canceled, whether via the "cancelable promise" approach or the "cancel token" approach. The question is more about once a promise gets canceled, how that state is represented and propagates down to any derived promises.

Against the alternative

The most popular alternative to the third state approach is to represent cancelation as a special type of rejection. This is very awkward in a language like JavaScript, which does not have typed catch guards (and won't for a long time, since this is blocked on pattern matching syntax which is a large proposal nobody has started yet). It leads to a lot of code like

https://github.com/domenic/cancelable-promise/blob/master/Third%20State.md


It's the final countdown

Promise.prototype.finally

ECMAScript Proposal, specs, and reference implementation for `Promise.prototype.finally`

Spec drafted by @ljharb, following the lead from the cancelable promise proposal.

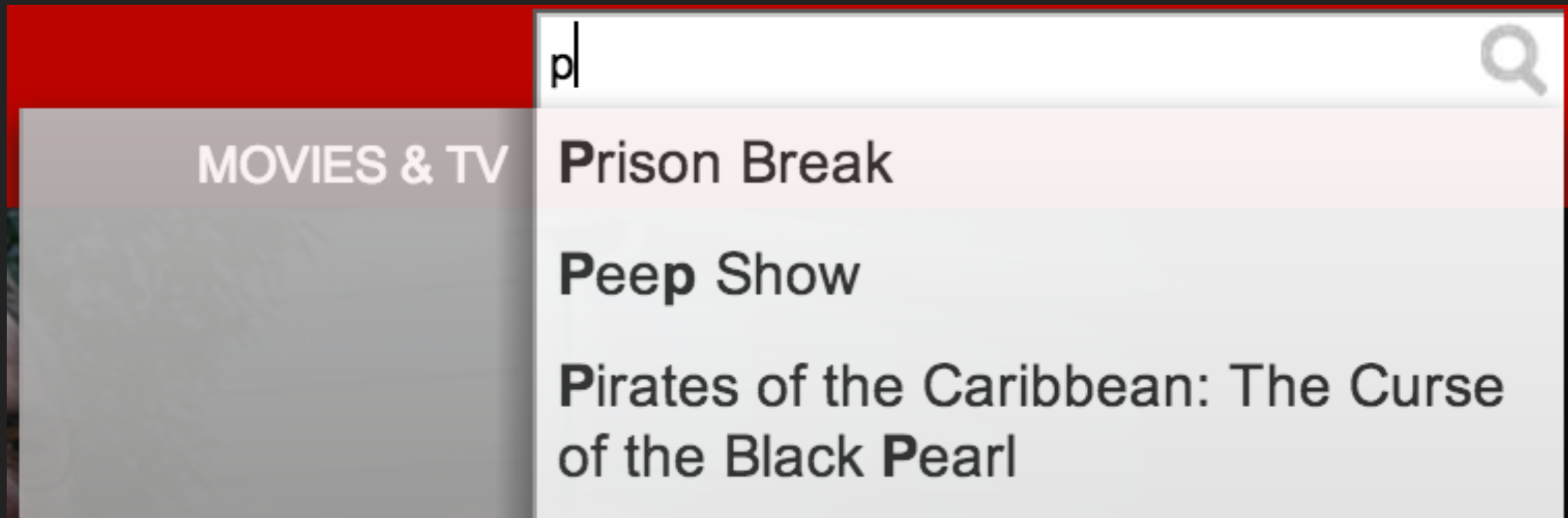
This proposal is currently stage 0 of the process.

Rationale

Many promise libraries have a "finally" method, for registering a callback to be invoked when a promise is settled (either fulfilled, or rejected). The essential use case here is cleanup - I want to hide the "loading" spinner on my AJAX request, or I want to close any file handles I've opened, or I want to log that an operation has completed regardless of whether it succeeded or not.

https://github.com/ljharb/proposal-promise-finally

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?

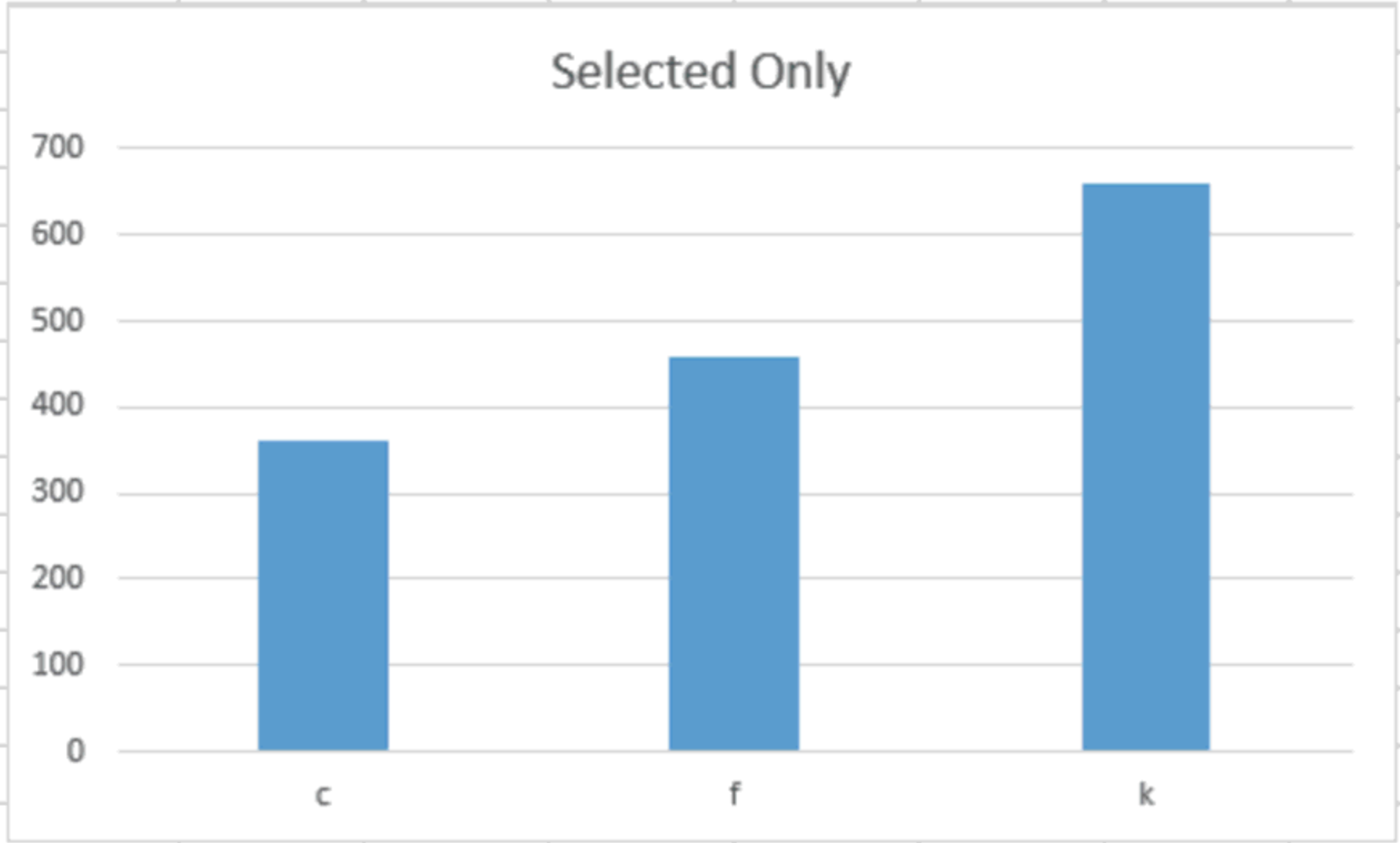
Merriam-Webster defines reactive as *“readily responsive to a stimulus”*, i.e. its components are “active” and always ready to receive events.

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

Clipboard | Font | Alignment | Number

T18

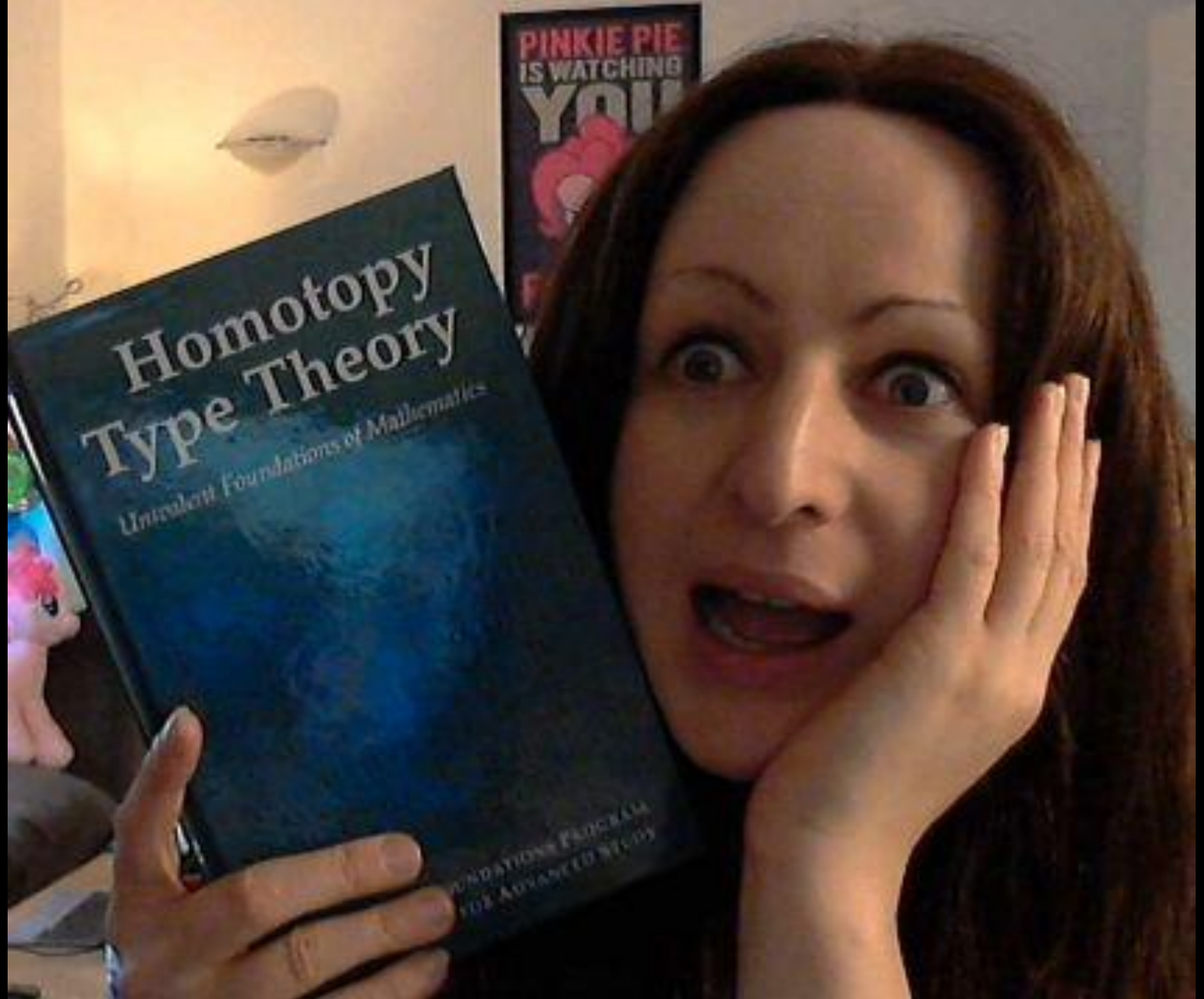
	A	B	C
1		2011	Show
2	a	330.33	
3	b	329.35	
4	c	359.85	Yes
5	d	376.94	
6	e	446.86	
7	f	457.7	Yes
8	g	509.38	
9	h	549.9	
10	i	566.84	
11	j	585.2	
12	k	658.24	Yes
13	l	720.2	
14			
15			



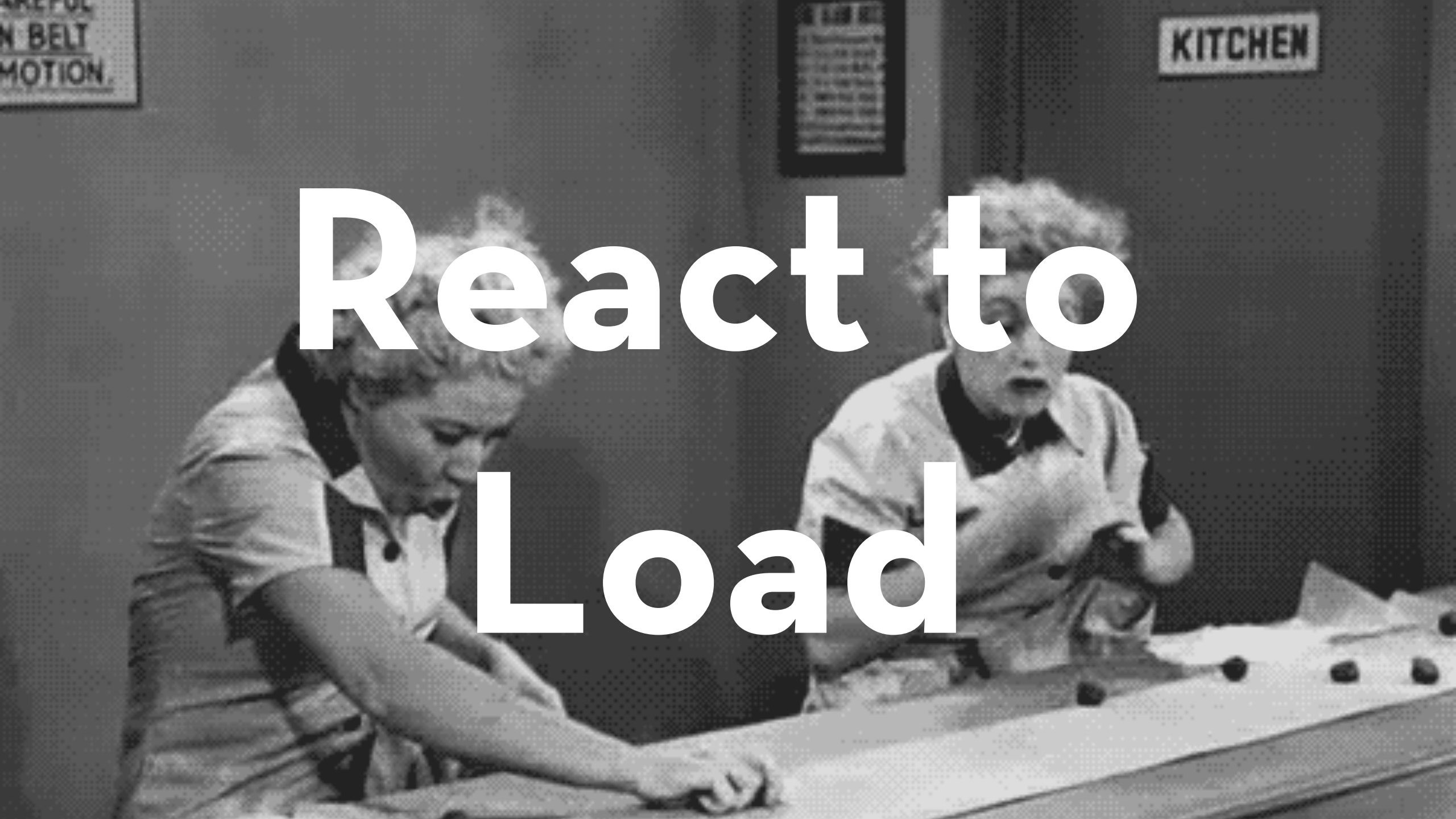
Wanna really know what Reactive Programming Is?

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

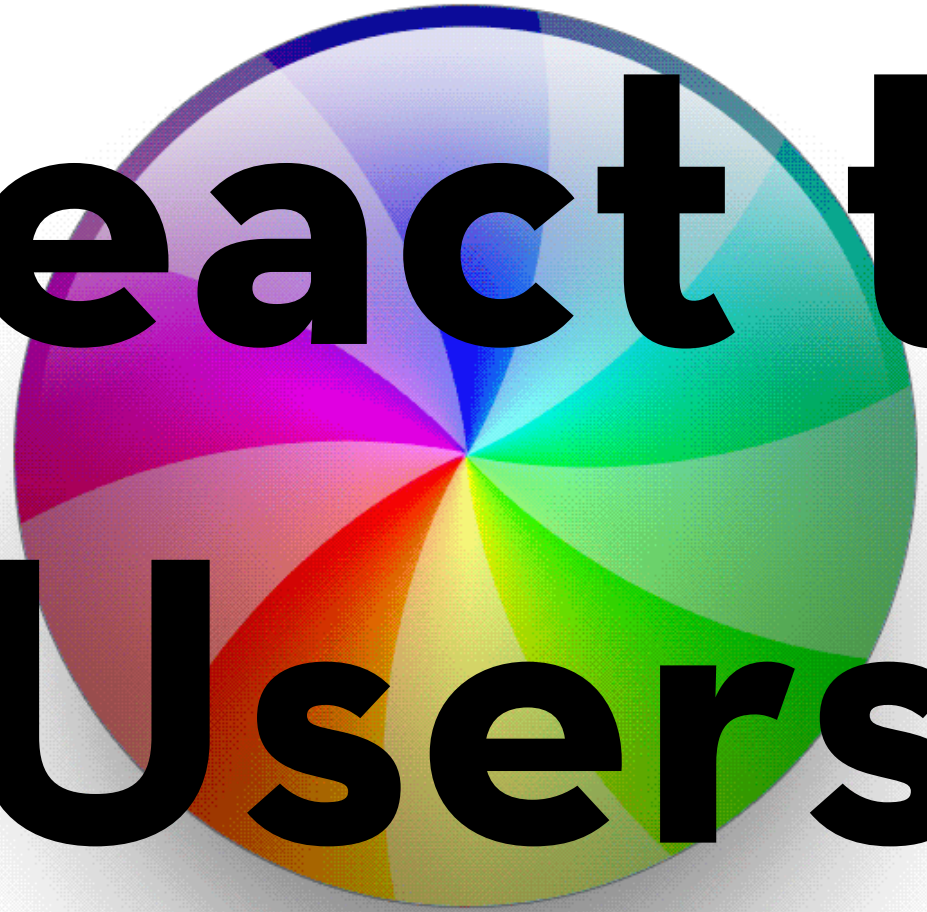
ComputerWissenschaftAkademischesPapierPhobie



React to Load


React to
Failure
next

React to Users

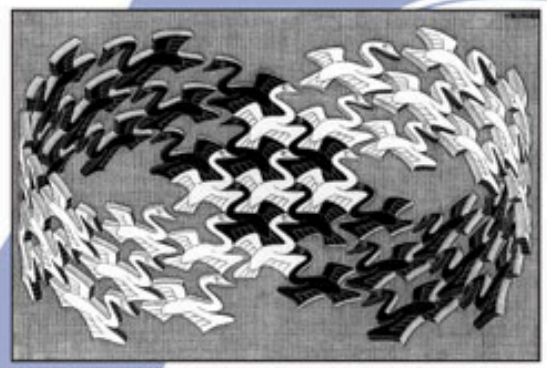



1994
ACE VENTURA
PET DETECTIVE
Sports Illustrated
SUPER METROID
KEANU REEVES
Dookie
CLERKS
A very funny look at the over-the-counter culture.
NIRVANA
UNPLUGGED
IN NEW YORK
weezer
AMERICA
Online

Design Patterns

Elements of Reusable Object-Oriented Software

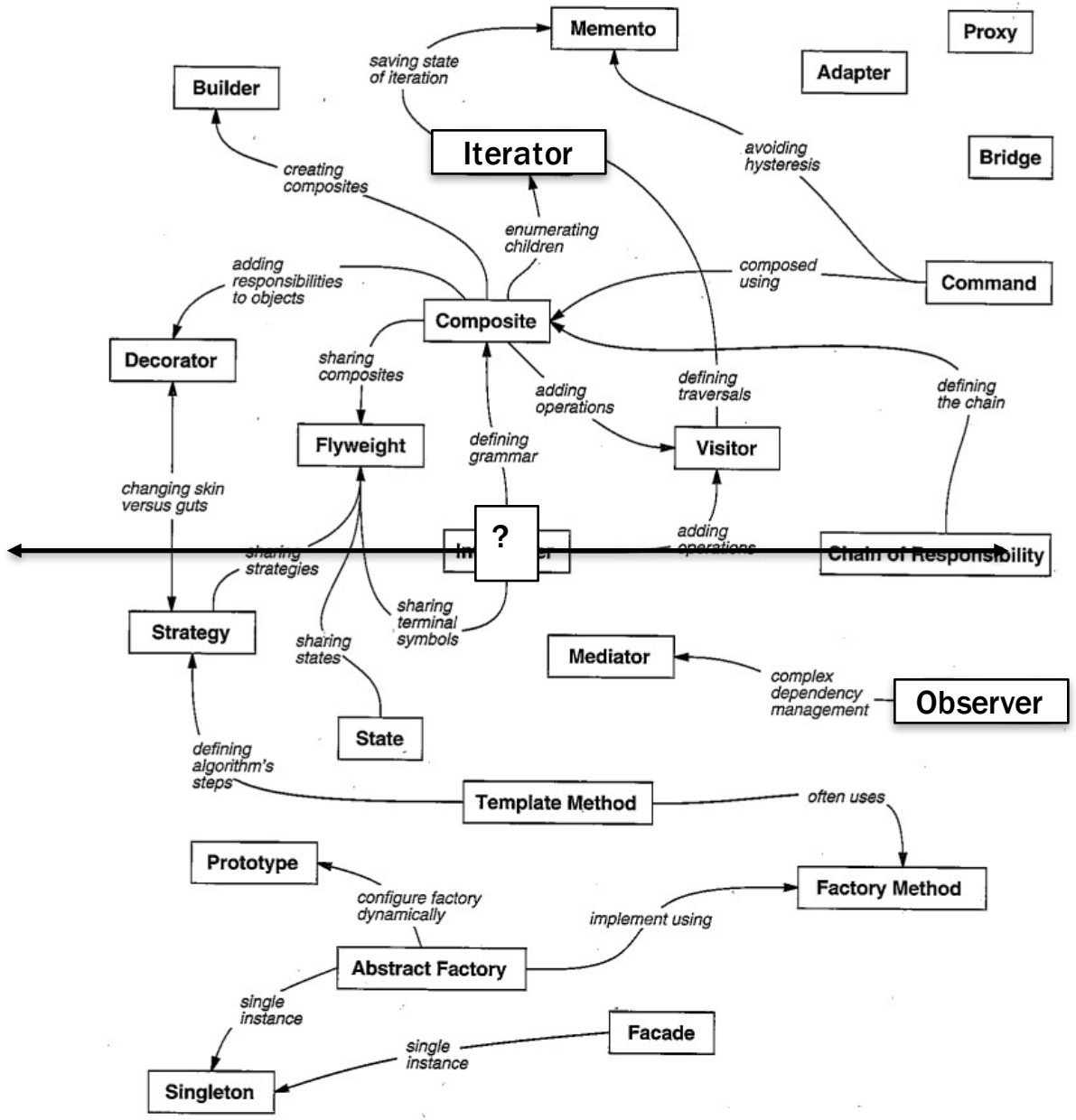
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



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

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

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

... and an Event?

Events and Iterables 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
}
```

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

```
interface Subscription {
  unsubscribe() : void;
}
```

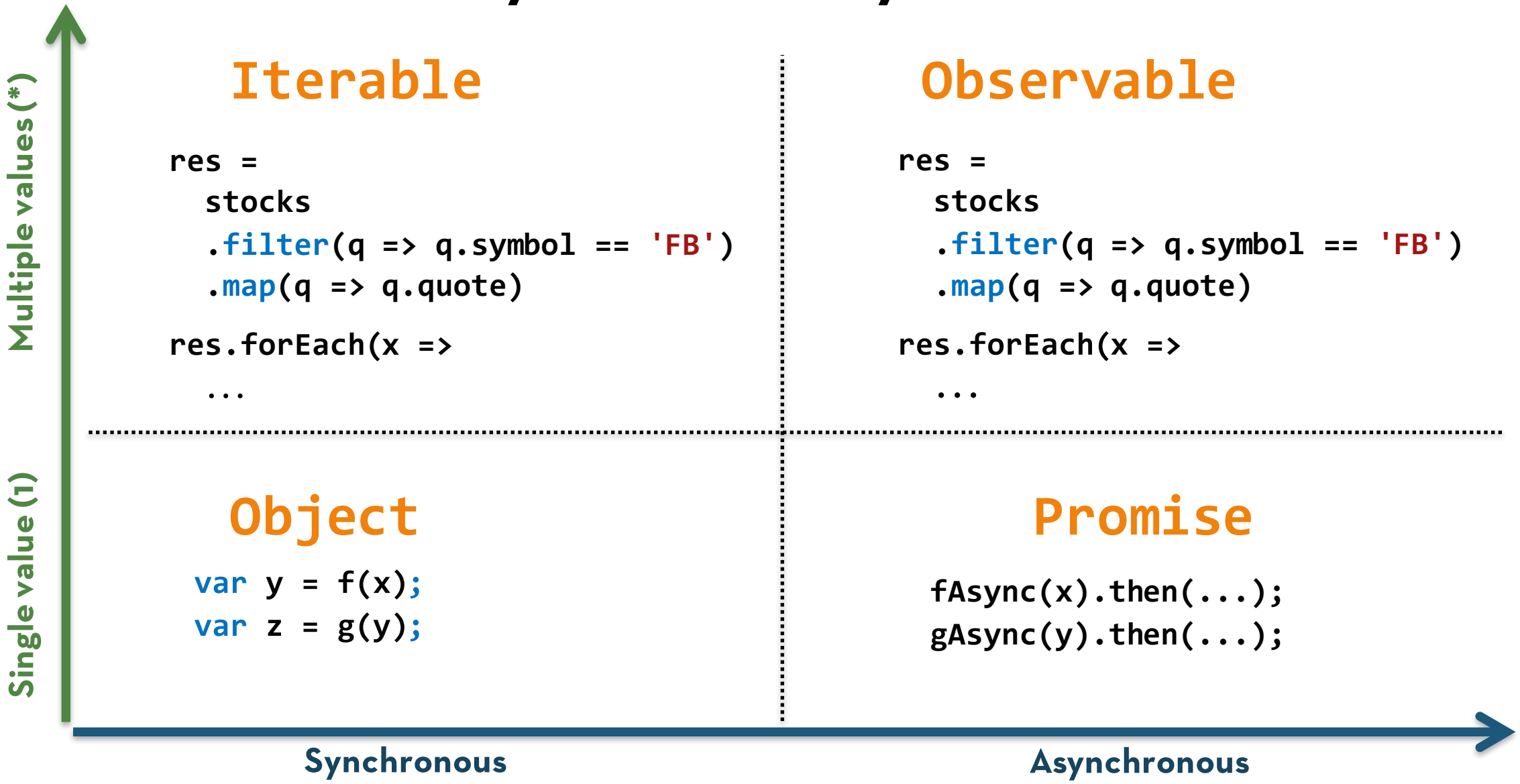
First-Class Asynchronous Values

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)

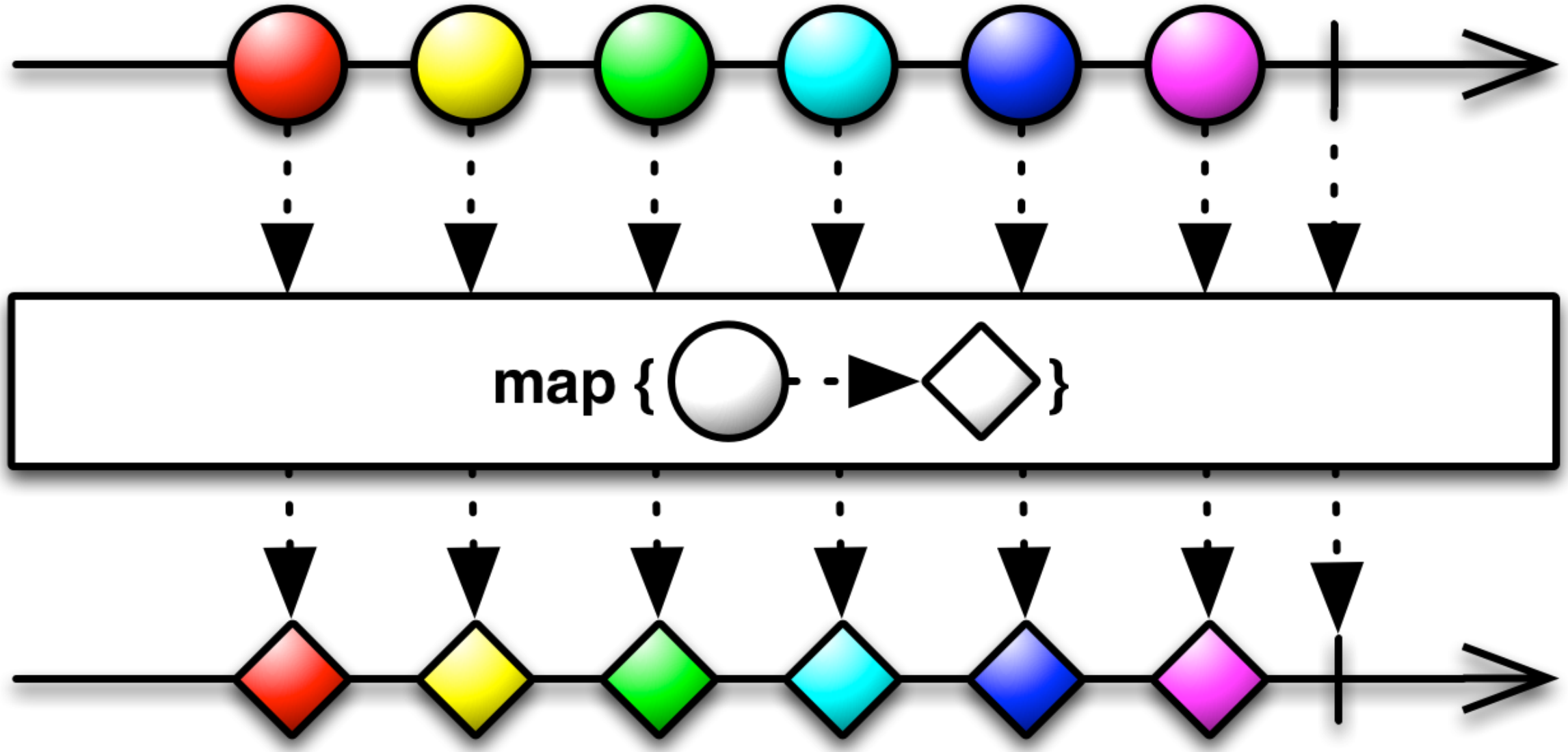
WIKIPEDIA
The Free Encyclopedia

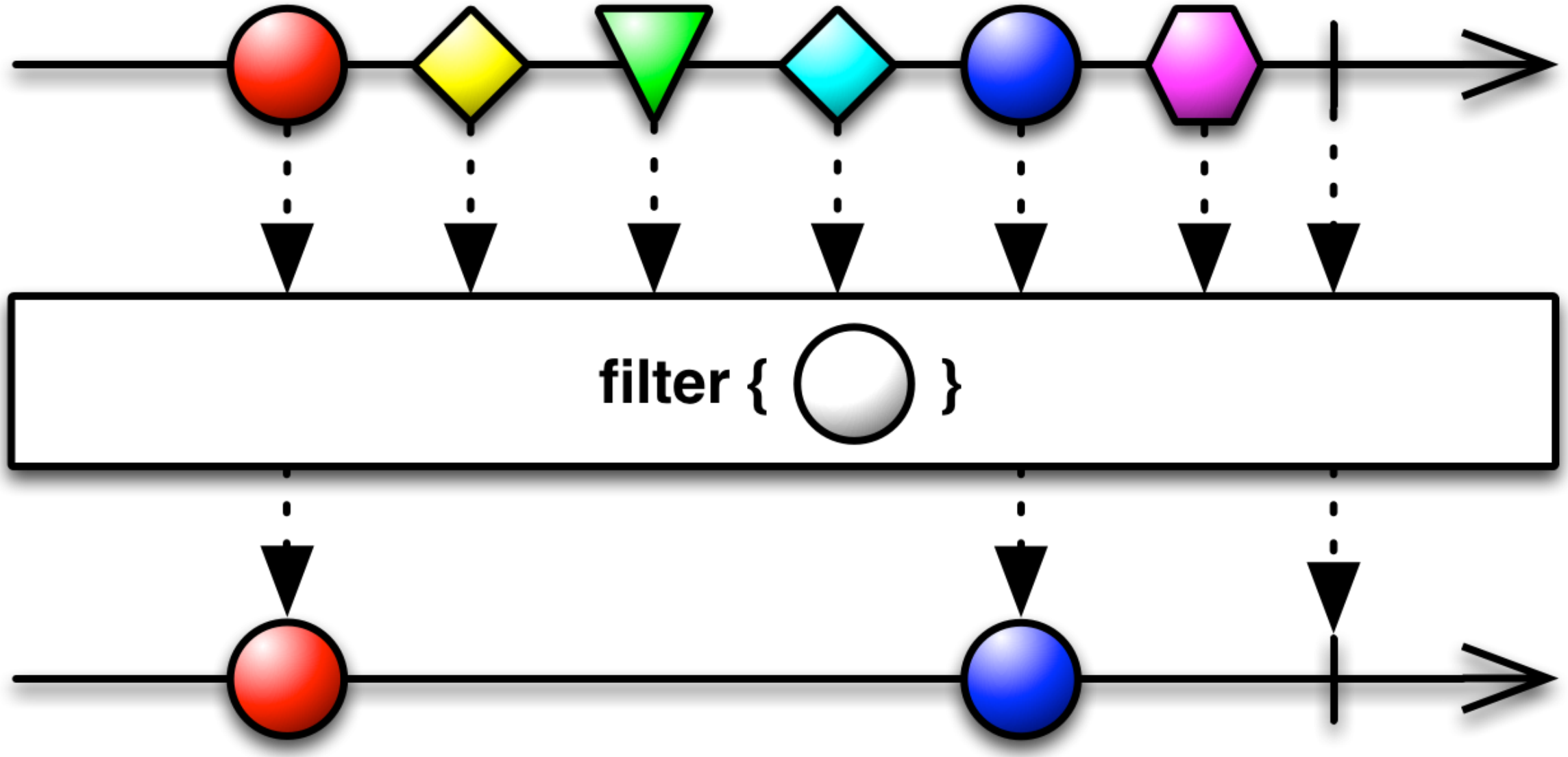
The General Theory of Reactivity

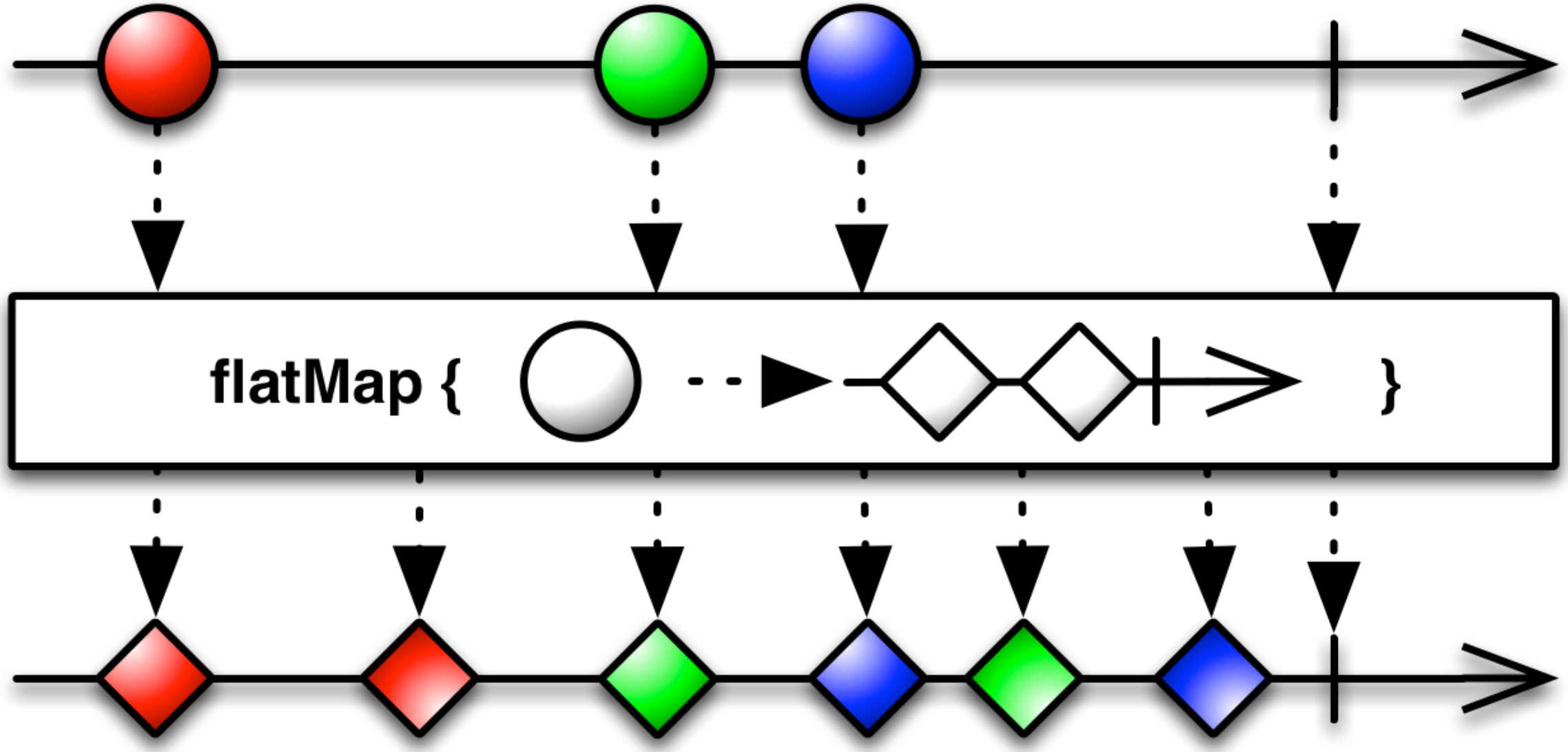



Everything is a stream

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


map {


filter { }


flatMap {
}


You Must
flatMap It!

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)

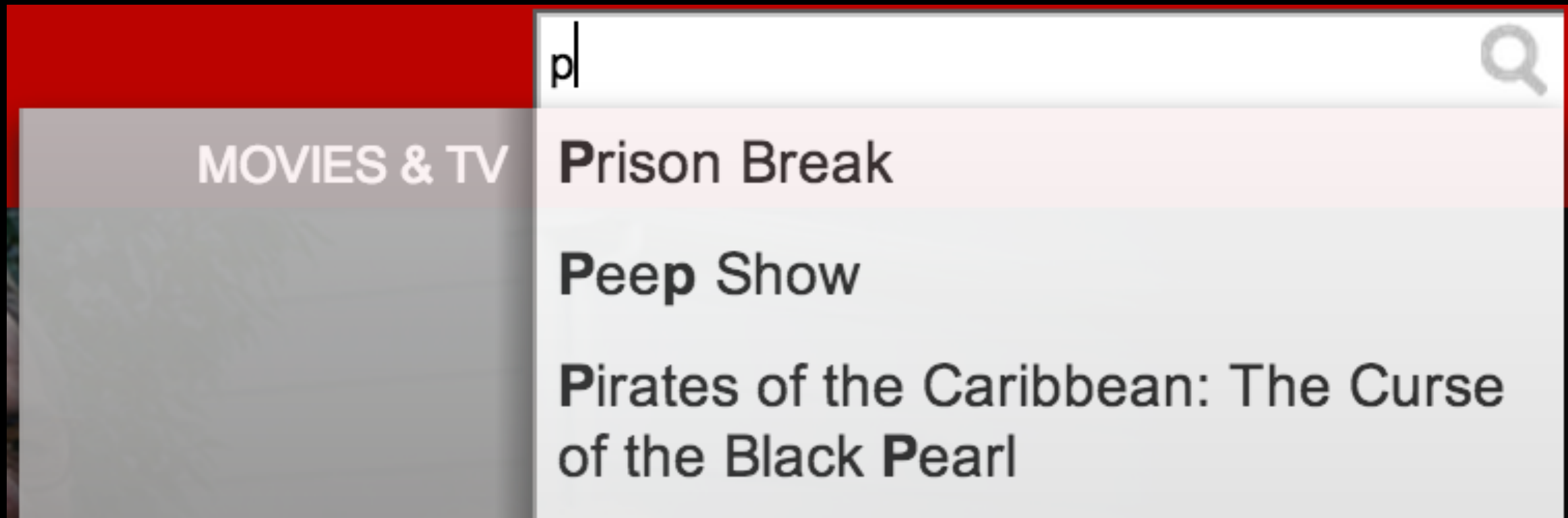
jeffbcross authored on Apr 29 — 1 parent 363b9ba commit 21568106b114943b59ce37832d82c8fb9a63285b

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>
  `
})
```

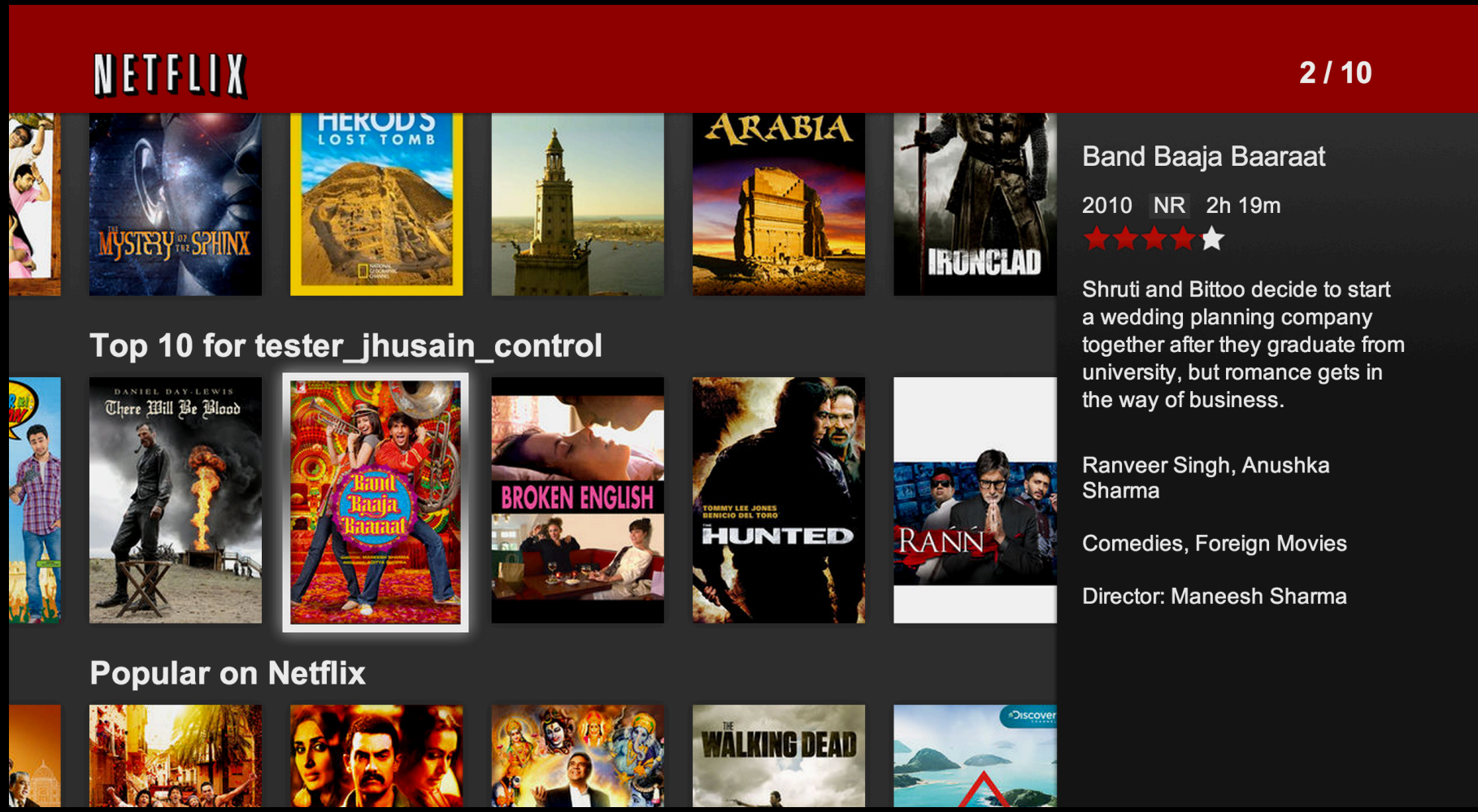
```
import {Injectable} from 'angular2/core';
import {URLSearchParams, Jsonp} from 'angular2/http';

@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));
  }
}
```

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

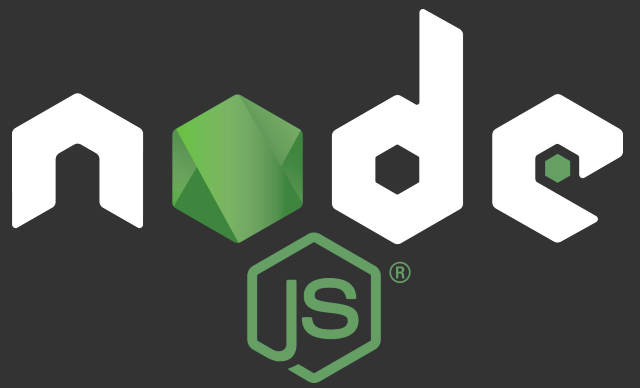
Implementing Spell Check in Slack

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

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

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

The question is: do you really need Redux if you already use Rx? Maybe not. It's not hard to re-implement Redux in Rx. Some say it's a two-liner using Rx `.scan()` method. It may very well be!

If you're in doubt, check out the Redux source code (there isn't much going on there), as well as its ecosystem (for example, the developer tools). If you don't care too much about it and want to go with the reactive data flow all the way, you might want to explore something like Cycle instead, or even combine it with Redux. Let us know how it goes!

```
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 ECMAScript???!!?

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 ECMAScript

```
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();
```

Via the Washington Post. wapo.st/10jJ02P

@ReactiveX

http://reactivex.io

Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/scotlandjs-2016