

**The Mother of All Demos (1968)**  
- Douglas Engelbart

# Mother of All Monad Demos

**Jon Galloway**

**@jongalloway**

**Matthew Podwysocki**

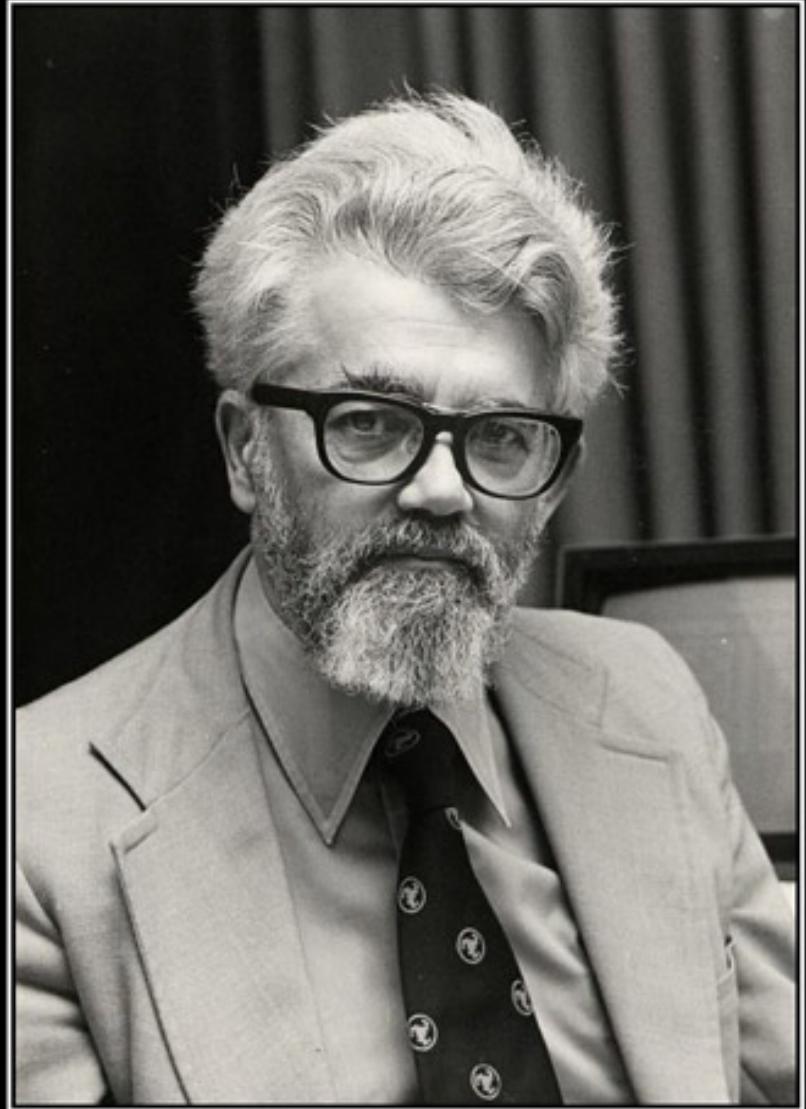
**@mattpodwysocki**

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

WHAT'S THE PROBLEM ?



**trapd in Monad tutorl  
plz help**



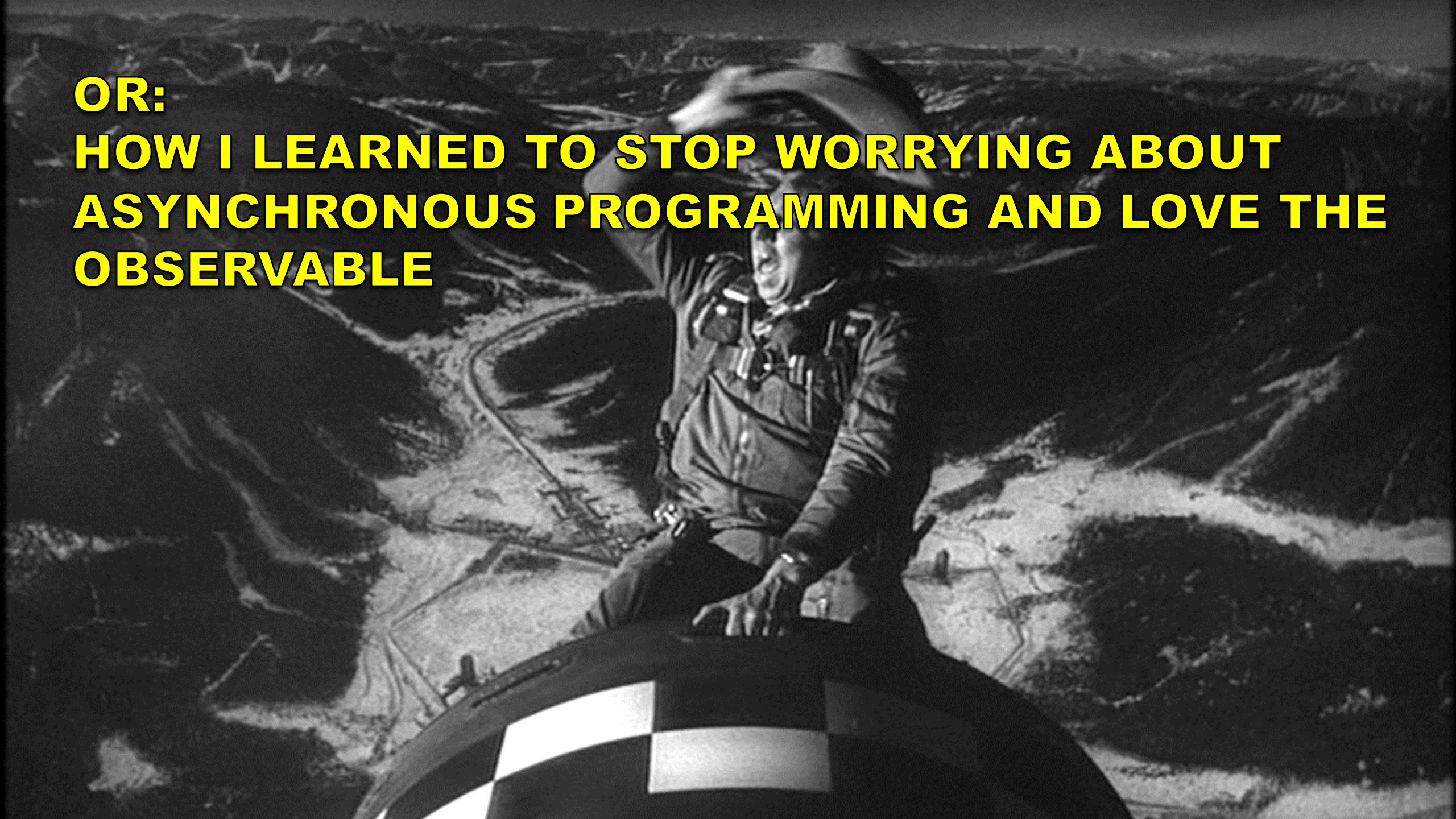
# PROGRAMMING

You're Doing It Completely Wrong.

## Async and Streaming JS Are We doing it wrong?

Matthew Podwysocki @mattpodwysocki

[github.com/mattpodwysocki/ndc-oslo-2015](https://github.com/mattpodwysocki/ndc-oslo-2015)

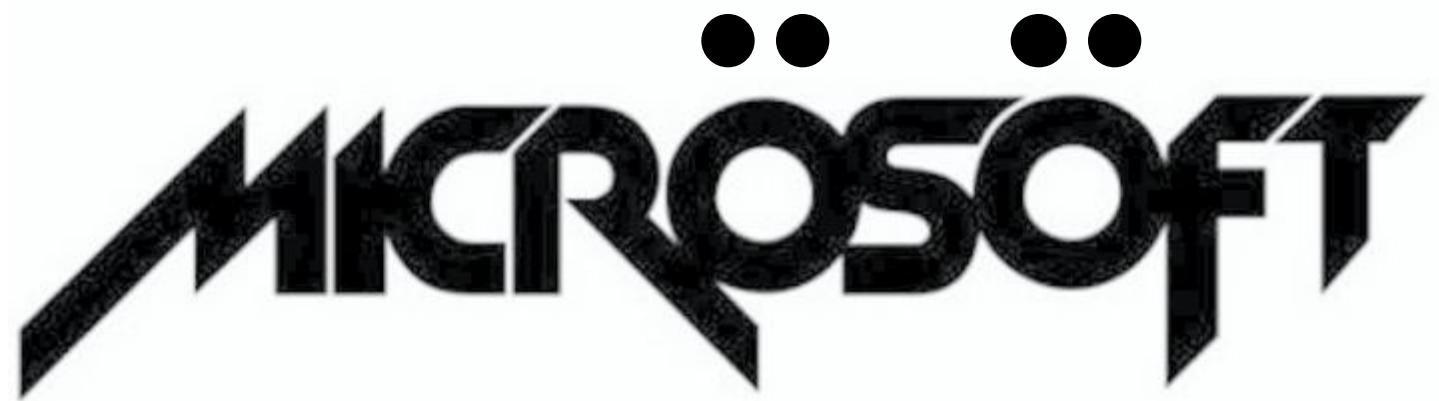


**OR:  
HOW I LEARNED TO STOP WORRYING ABOUT  
ASYNCHRONOUS PROGRAMMING AND LOVE THE  
OBSERVABLE**

**Or "I thought I had a problem. I thought to myself,  
"I know, I'll solve it with callbacks and events!".  
have Now problems. two I**



**Principal Open Source Engineer  
Open Sourcerer  
@mattpodwysocki  
[github.com/mattpodwysocki](https://github.com/mattpodwysocki)**



The Microsoft logo consists of the word "MICROSOFT" in a bold, black, sans-serif font. The letters are slanted slightly to the right. Above the letter "I", there are two small black dots. Above the letter "O", there are three black dots arranged horizontally.



# Thali Project

@thaliproject

<http://thaliproject.org>



# Reactive Extensions (Rx)

@ReactiveX  
<http://reactivex.io>

# Real-Time is Everywhere...



SPIKE LATHAM

# Let's Face It, Asynchronous Programming is Awful!



**“We choose to go to solve asynchronous  
programming and do the other things,  
not because they are easy, but because  
they are hard”**



**Former US President John F. Kennedy - 1962  
[citation needed]**

# Callback Hell

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



culturepub.f

next



# Events and the Enemy of the State

```
var isDown = false, state;

function mousedown (e) {
  isDown = true;
  state = { startX: e.offsetX,
            startY: e.offsetY };
}

function mousemove (e) {
  if (!isDown) { return; }
  var delta = { endX: e.clientX - state.startX,
                endY: e.clientY - state.startY };
  // Now do something with it
}

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

```
function dispose() {
  elem.removeEventListener('mousedown', mousedown, false);
  elem.removeEventListener('mouseup', mouseup, false);
  doc.removeEventListener('mousemove', mousemove, false);
}

elem.addEventListener('mousedown', mousedown, false);
elem.addEventListener('mouseup', mouseup, false);
doc.addEventListener('mousemove', mousemove, false);
```





# First Class Async with Promises

then

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

# Breaking the Promise...

then

```
var promise;

input.addEventListener('keyup', (e) => {

  if (promise) {
    // Um, how do I cancel?
  } else {
    promise = getData(e.target.value).then(populateUI);
  }
}, false);
```



# Aborting a fetch #27

[New issue](#)[Open](#)

annevk opened this issue 26 days ago · 182 comments



annevk commented 26 days ago

Owner

## Goal

Provide developers with a method to abort something initiated with `fetch()` in a way that is not overly complicated.

## Previous discussion

- [#20](#)
- [slightlyoff/ServiceWorker#592](#)
- [slightlyoff/ServiceWorker#625](#)
- [whatwg/streams#297](#)

## Viable solutions

We have two contenders. Either `fetch()` returns an object that is more than a promise going forward or `fetch()` is passed something, either an object or a callback that gets handed an object.

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

### Labels

None yet

### Milestone

No milestone

### Assignee



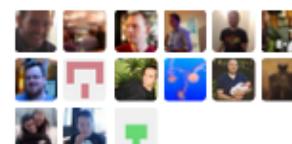
jakearchibald

### Notifications

[Subscribe](#)

You're not receiving notifications from this thread.

### 15 participants



# 3.0 cancellation overhaul #415

[New issue](#)

Open

petkaantonov opened this issue on Dec 27, 2014 · 79 comments



petkaantonov commented on Dec 27, 2014

Owner

The current cancellation seems to be the most problematic feature due to some annoying limits in the design:

- Awkward API for attaching the callback that cancels the work (through typed catch handler)
- Cannot compose because it is impossible to tell the difference between rejection and cancellation (e.g. `Promise.all(...).cancel()` cannot do the obvious thing and also call `cancel()` on the promises in the array)
- `.cancel()` is asynchronous

Since all consumers and producers must be trusted/"your code", there is no reason to enforce that cancellable promises are single-consumer or create new primitives.

**Edit:** The below design has been scratched, see [#415 \(comment\)](#)

Labels

3.0.0

Milestone

No milestone

Assignee

No one assigned

Notifications

Subscribe

You're not receiving notifications from this discussion

# The Final Countdown...

then

```
var resource;  
  
try {  
    var resource = getResource();  
} catch (e) {  
    throw e;  
} finally {  
    resource && resource.dispose();  
}  
  
getresource()  
    .success(  
        function (resource) {  
            })  
    .catch(  
        function (err) {  
            });  
  
// How do I clean up my resource?
```

then



MAKE GIFS AT GIFOUP.COM



## UNSAFE AT ANY SPEED

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



**stroom prosesing**



## Let's Face it, Streams<sup>1</sup> 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





# WHATWG Streams

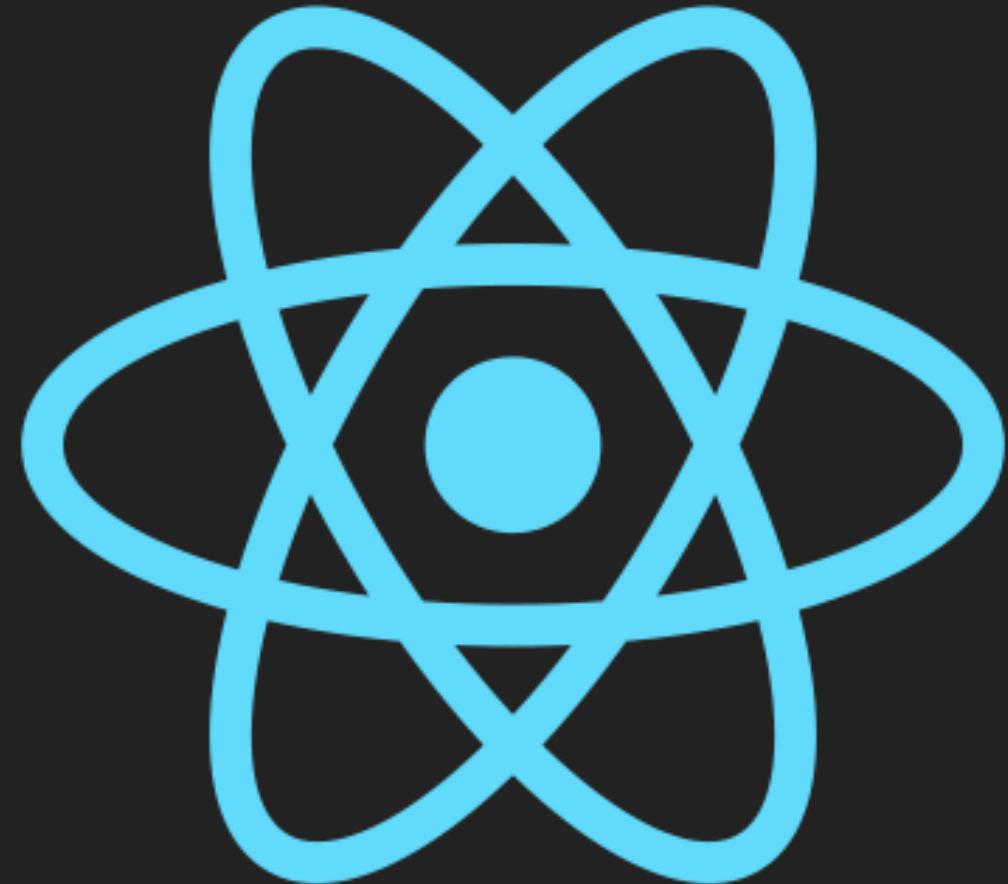
**Specification for creating, composing and consuming streams of data**

- Focused on low-level I/O, not on object mode
- Think Node.js Streams + Promises

```
readableStream.pipeTo(writableStream)
  .then(() => console.log("All data successfully written!"))
  .catch(e => console.error("Something went wrong!", e));
```

# 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

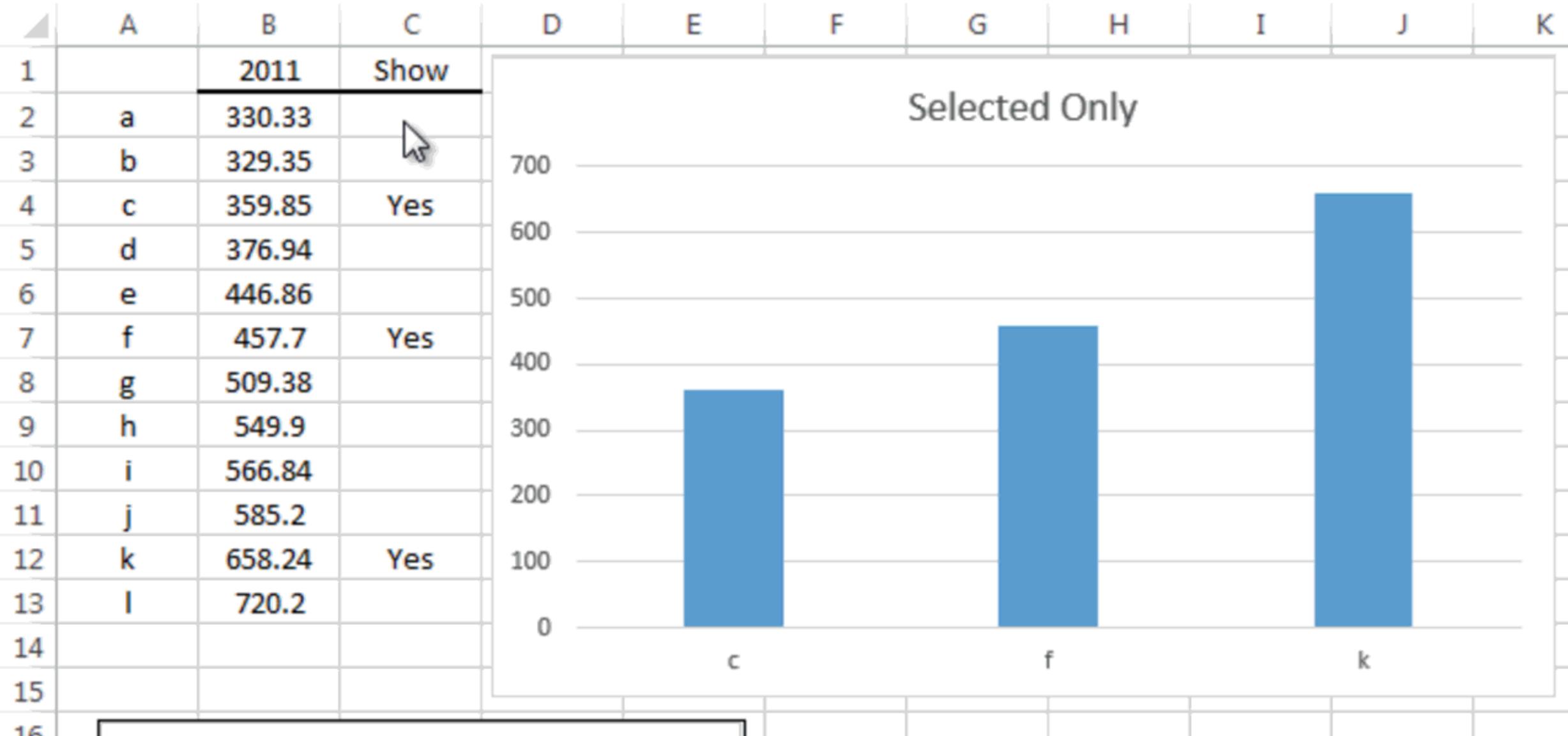
Number

T18

:



fx

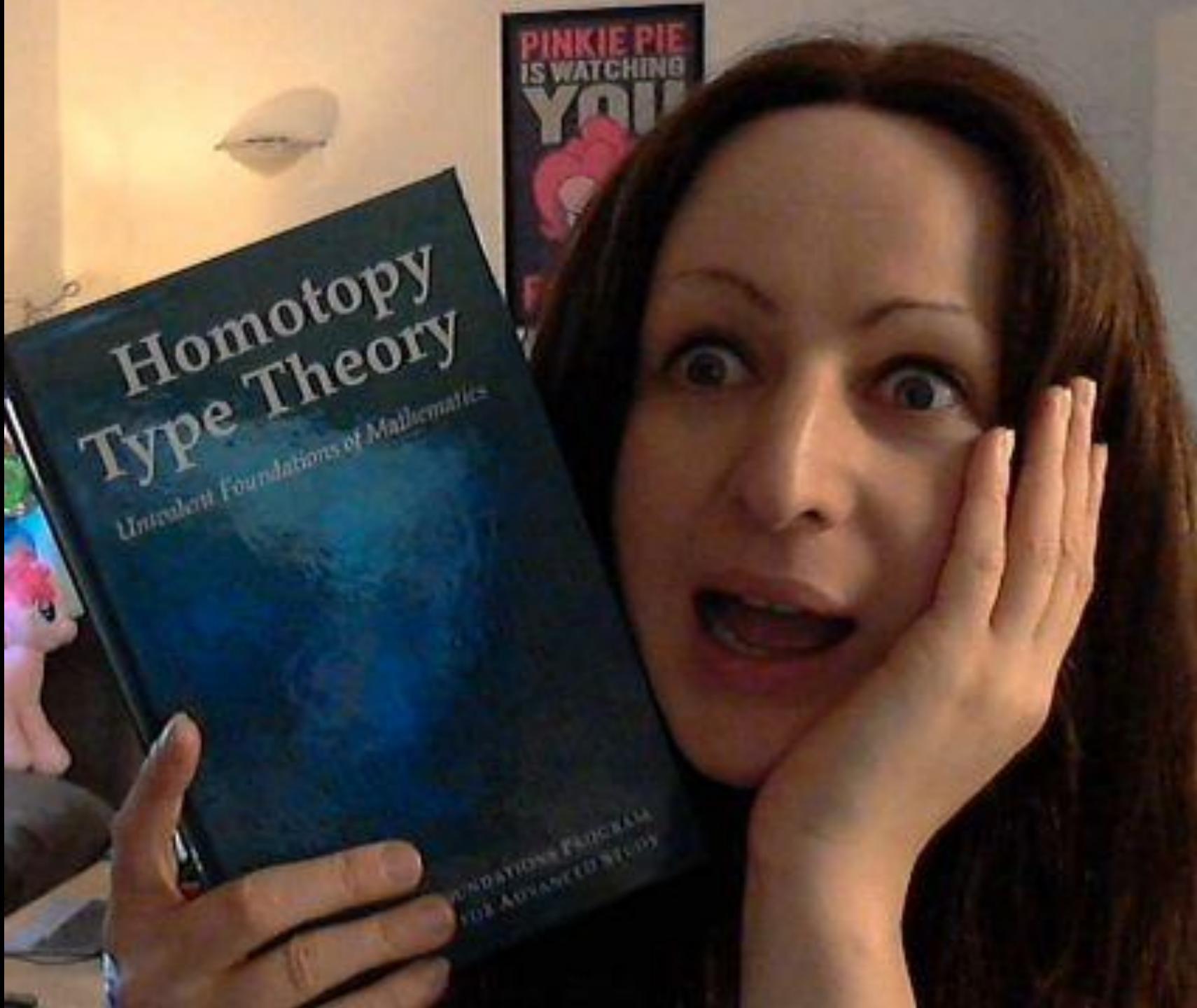


# **Wanna really know what Reactive Programming Is?**

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

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





**ComputerWissenschaftAkademischesPapierPhobie**

# Functional Reactive Programming (FRP) is...

**A concept consisting of**

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

type Behavior a

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

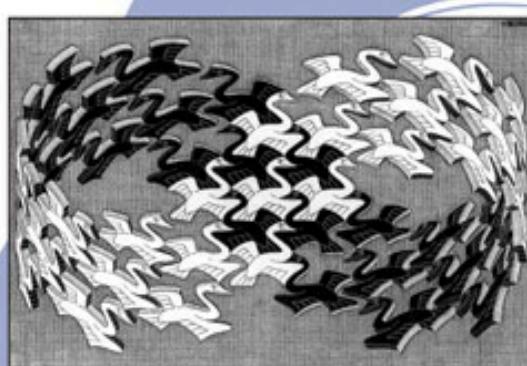
# 1994



# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides

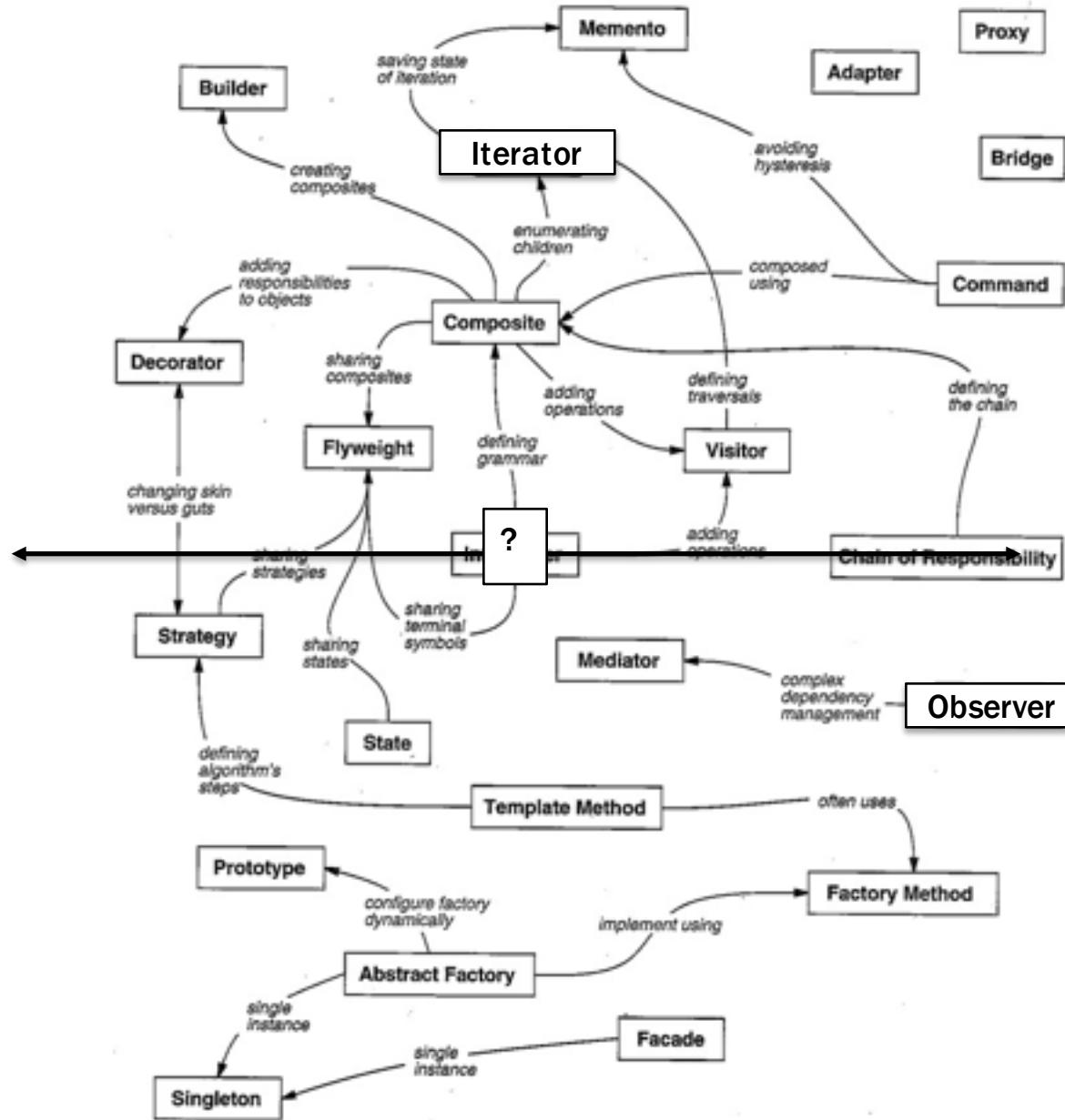


Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES





## Design Pattern Relationships

# Iterator Pattern with ES2015

```
> let iterator = getNumbers(); █  
> console.log(iterator.next()); █  
> { value: 1, done: false }  
> █onsole.log(iterator.next()); █  
> { value: 2, done: false }  
> █onsole.log(iterator.next()); █  
> { value: 3, done: false }  
> █onsole.log(iterator.next()); █  
> { done: true }  
> █
```

# Subject/Observer Pattern with the DOM

```
> document.addEventListener(  
  'mousemove',  
  (e) =>  
    console.log(e);  
); ■  
  
> { clientX: 425, clientY: 543 }  
> { clientX: 450, clientY: 558 }  
> { clientX: 455, clientY: 562 }  
> { clientX: 460, clientY: 743 }  
> { clientX: 476, clientY: 760 }
```



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

Foreword by Grady Booch



# Fundamental Abstractions

## Gang of Four Subject/Observer Pattern

- Notifies Observers on State Change
- Not Very Compositional

```
function handler (e) {  
    // Handle the event  
}
```

```
eventTarget.addEventListener(event, handler, false);
```

```
eventTarget.removeEventListener(event, handler, false);
```



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

Foreword by Grady Booch



# Fundamental Abstractions

## Adapting the Subject/Observer Pattern

- Ensuring duality with the enumerator pattern
- More Compositional Approach

```
interface Observable<T> {  
    subscribe(observer : Observer) : Disposable  
}
```

```
interface Observer<T> {  
    onNext(value : T) : void  
    onError(error : Error) : void  
    onCompleted() : void  
}
```

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

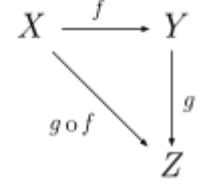
[{x: 23, y: 44}, {x:27, y:55}, {x:27, y:55}]



... and an Event?

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

# Category Theory to the Rescue!



Category theory  
[www.Wikipedia.org](http://www.Wikipedia.org)

## Adapting the Subject/Observer Pattern

- Observable/Observer (push) is dual to Iterable/Iterator (pull)
- Cross-influence from both domains

```
interface Observable<T> {
    subscribe(observer : Observer) : Disposable
}

interface Observer<T> {
    onNext(value : T) : void
    onError(error : Error) : void
    onCompleted() : void
}

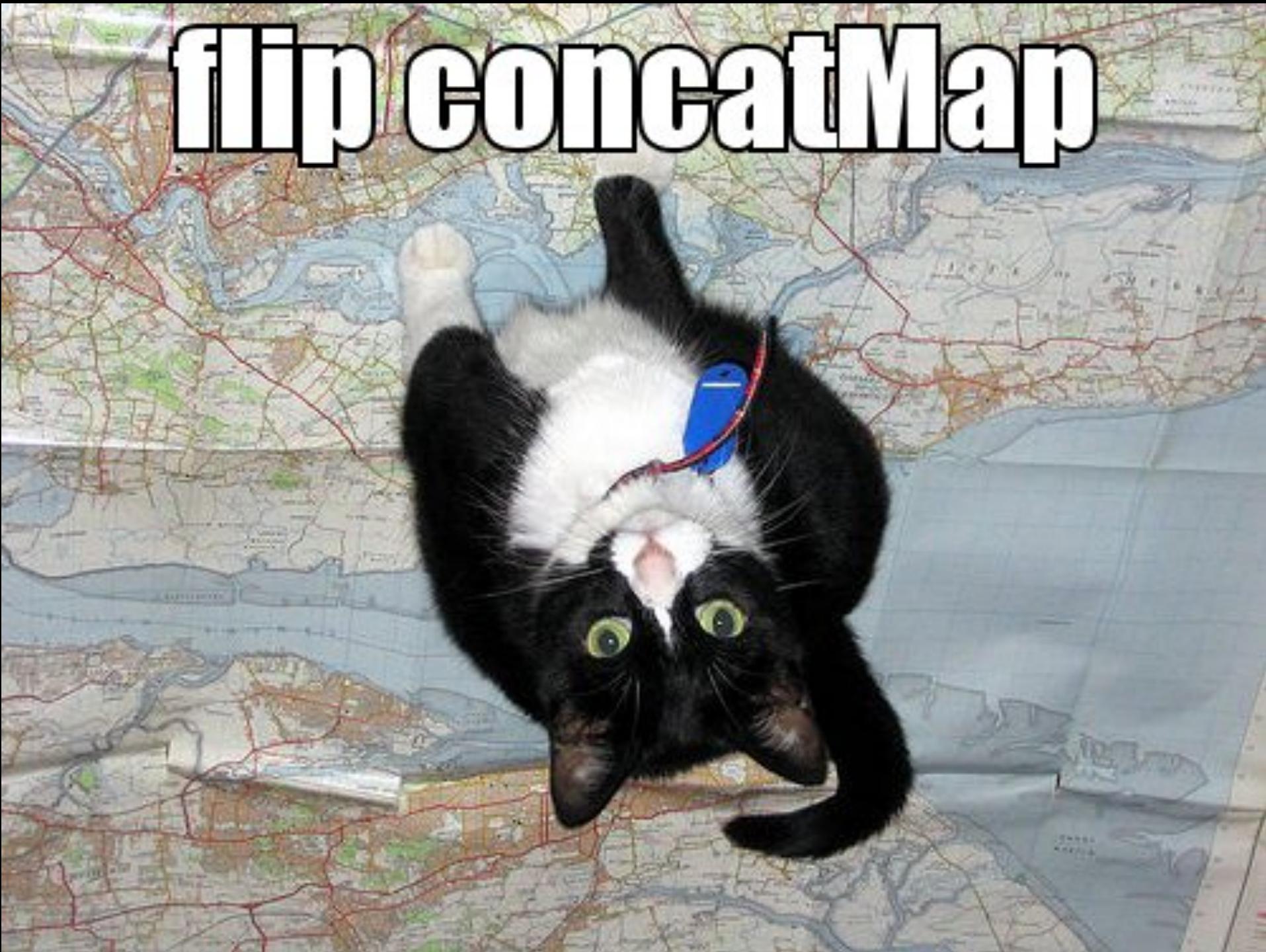
interface Iterable<T> {
    [Symbol.iterator](): Iterator<T>
}

interface Iterator<T> {
    next() : IteratorResult<T> // throws
}

interface IterationResult<T> {
    done: boolean
    value: T
}
```

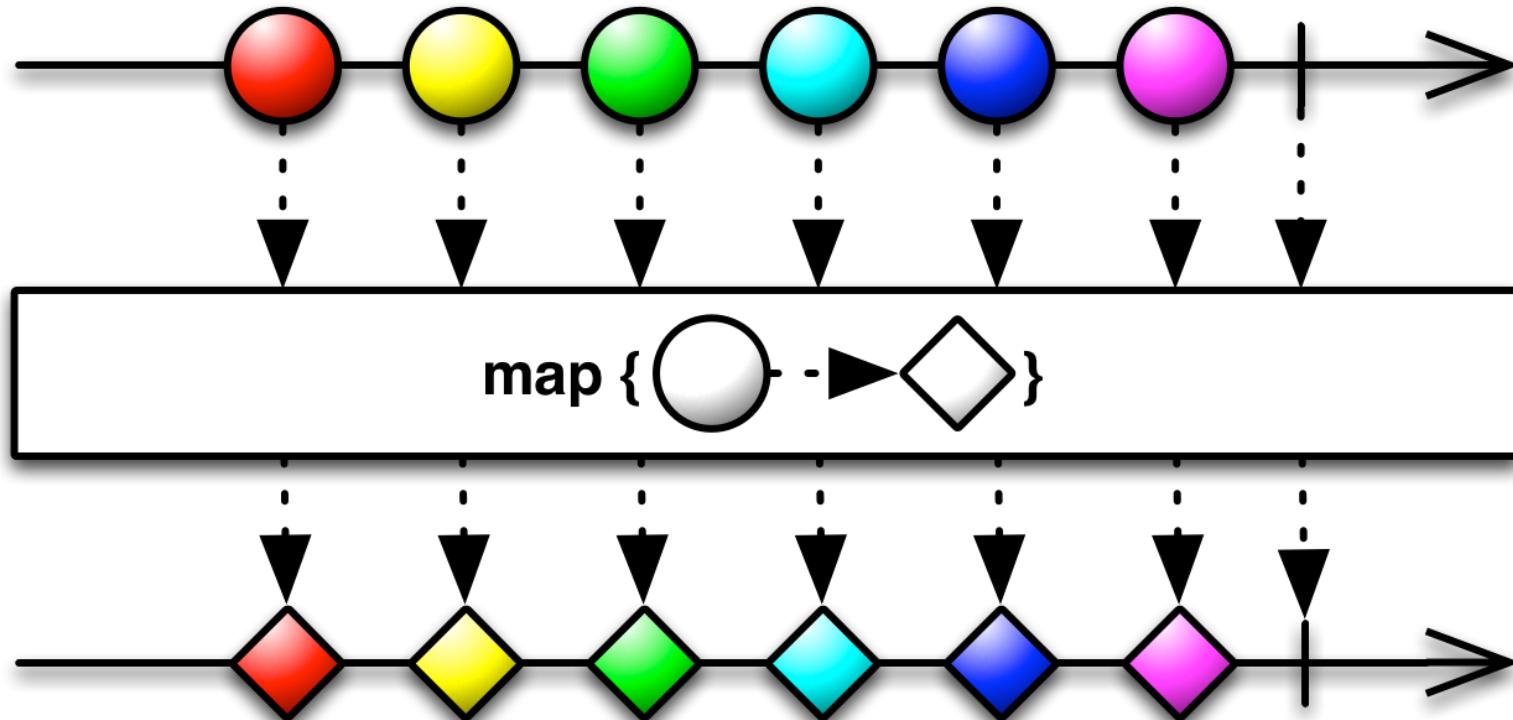
The majority of your asynchronous  
code is written with just a few  
***flexible*** functions.

# **flip concatMap**



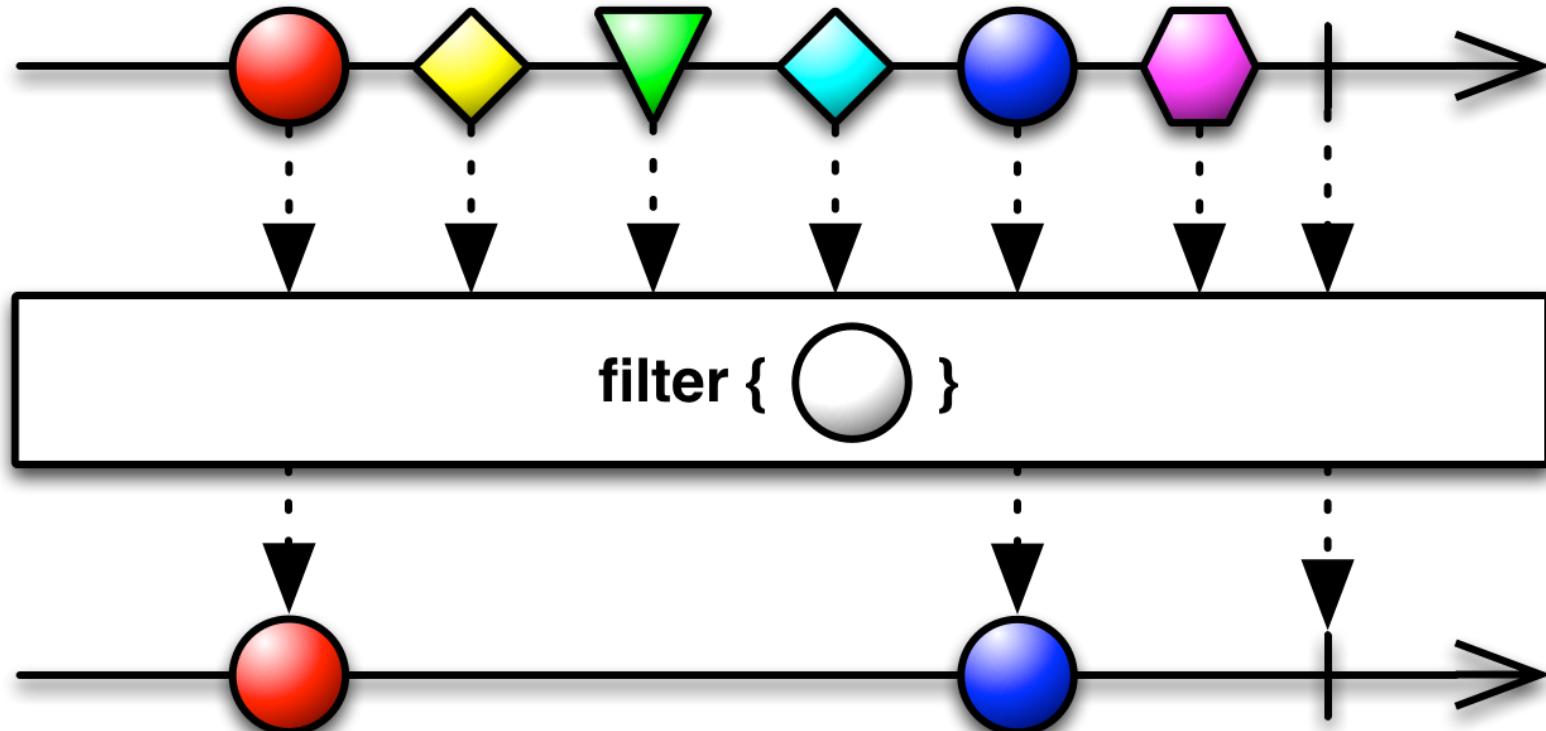
# map()

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



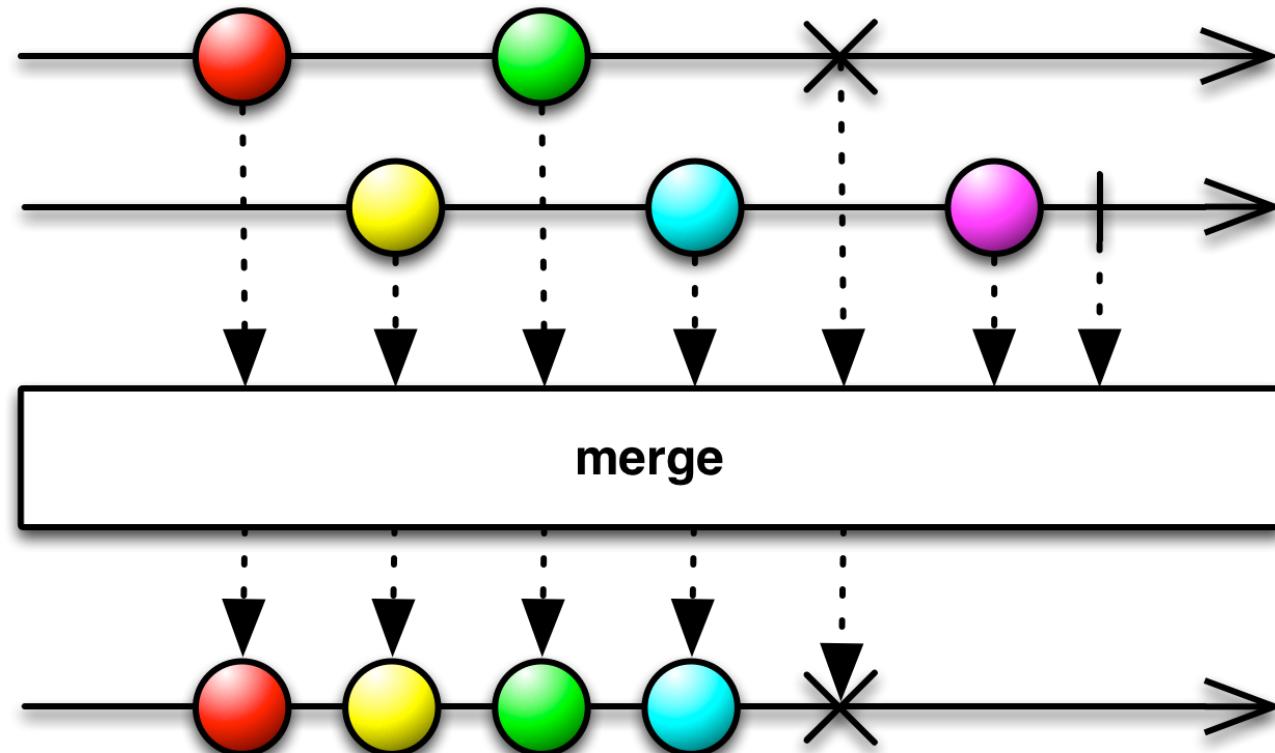
# filter()

Filter items emitted by a Collection



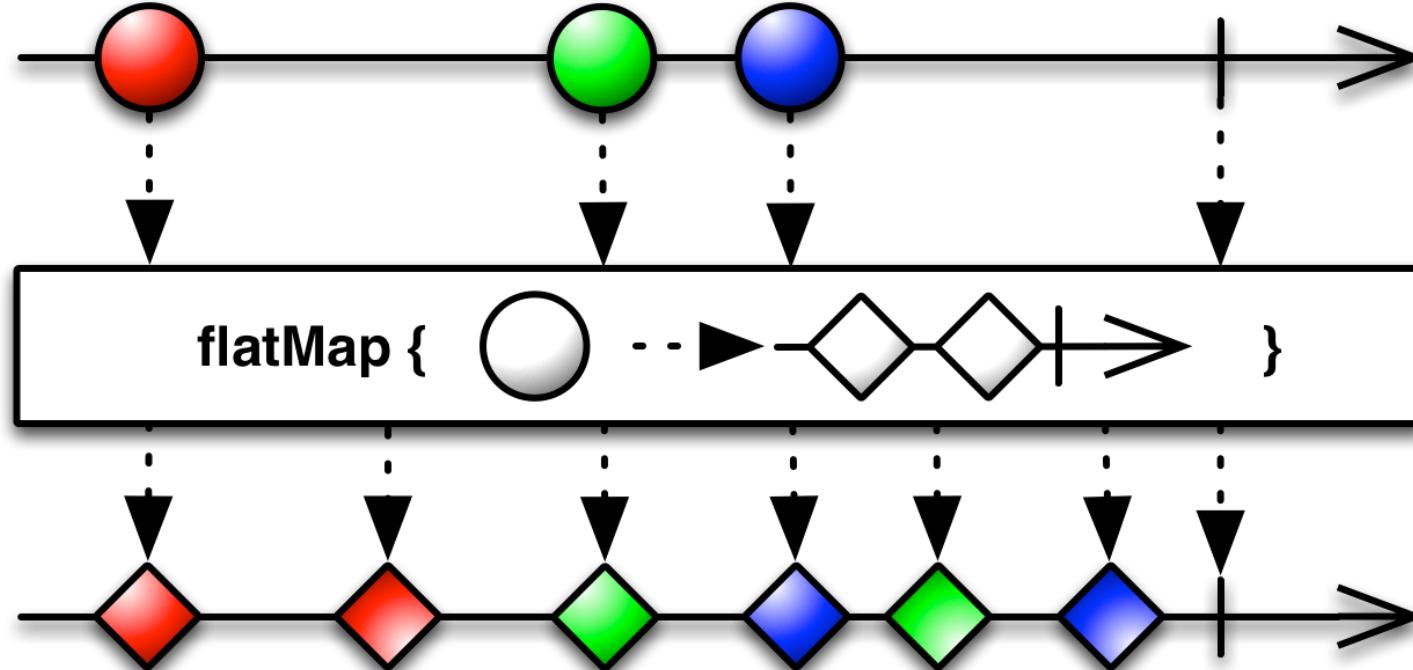
# mergeAll()

combine multiple Collections into one by merging their emissions



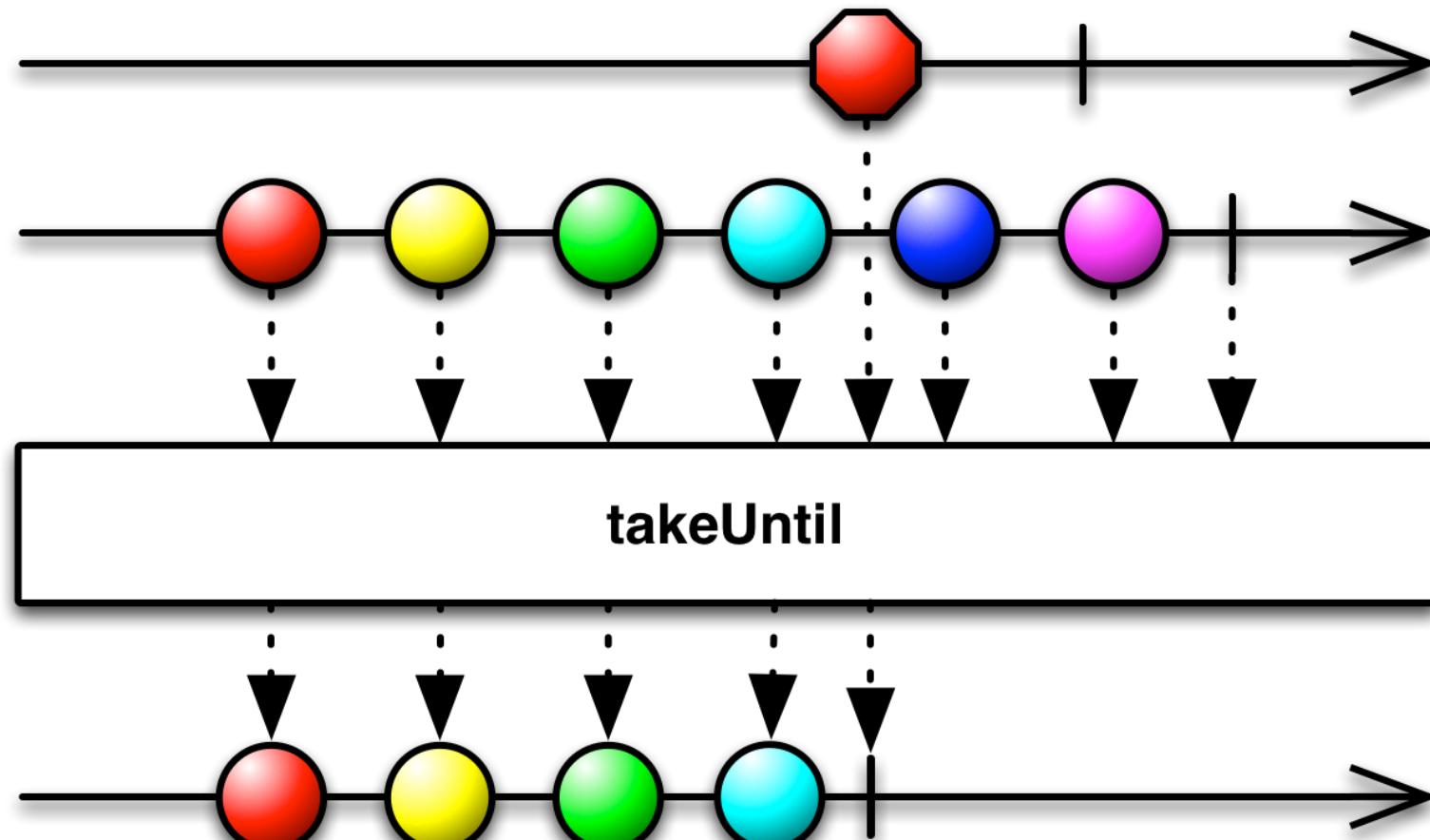
# flatMap()

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



# `takeUntil()`

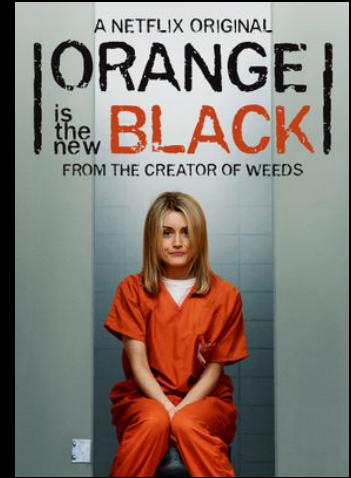
**Discard any items emitted by a Collection after a second Collection emits an item or terminates**



# Top-rated Movies Collection

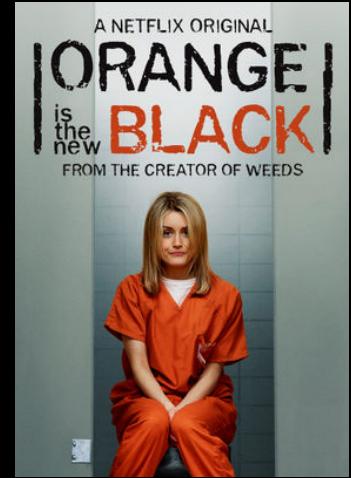
```
const getTopRatedFilms = (user) => {
  user.videoLists
    .map((videoList) =>
      videoList.videos
        .filter((v) => v.rating === 5)
    ).mergeAll();
};
```

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

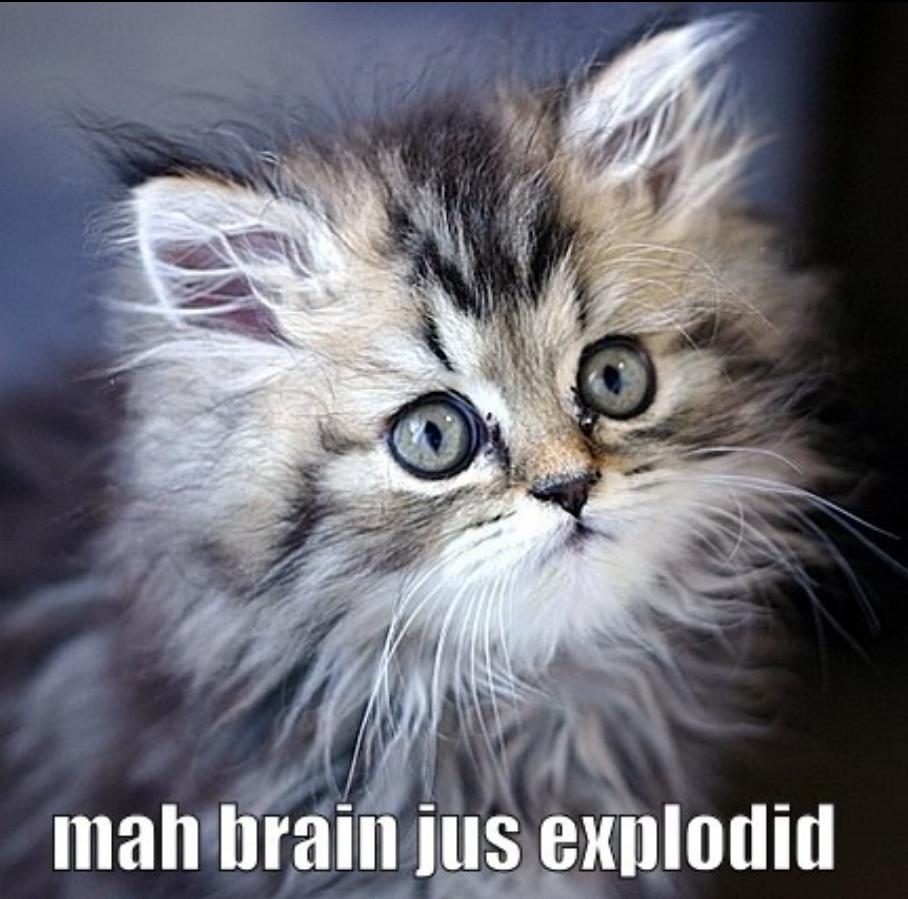


# Top-rated Movies Collection

```
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**

**mah brain jus explodid**

# Mouse Drags Collection

```
let getElementDrags = (elmt) =>
  dom.mousedown(elmt)
    .map((md) =>
      dommousemove(document)
        .filter .takeUntil(dommouseup(elmt)))
    ).mergeAll()
};

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

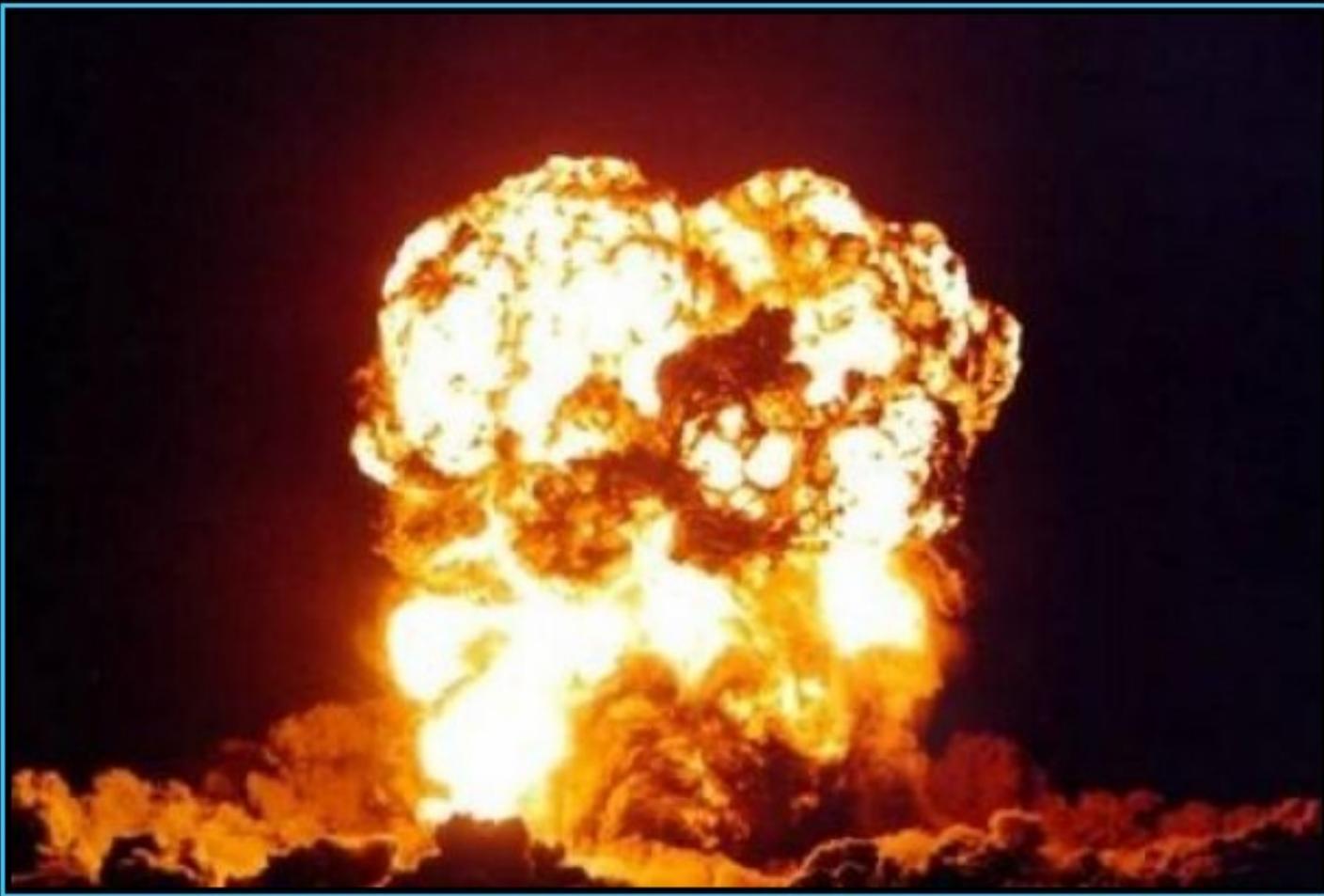


# Mouse Drags Collection

```
let getElementDrags = (elmt) =>
  dom.mousedown(elmt)
    .flatMap((md) =>
      dommousemove(document)
        .filter .takeUntil(dommouseup(elmt)))
    )
};

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





# PROTONIC REVERSAL

You crossed the streams, didn't you?



Everything is a stream

# First-Class Async and Events

## Objects to the rescue

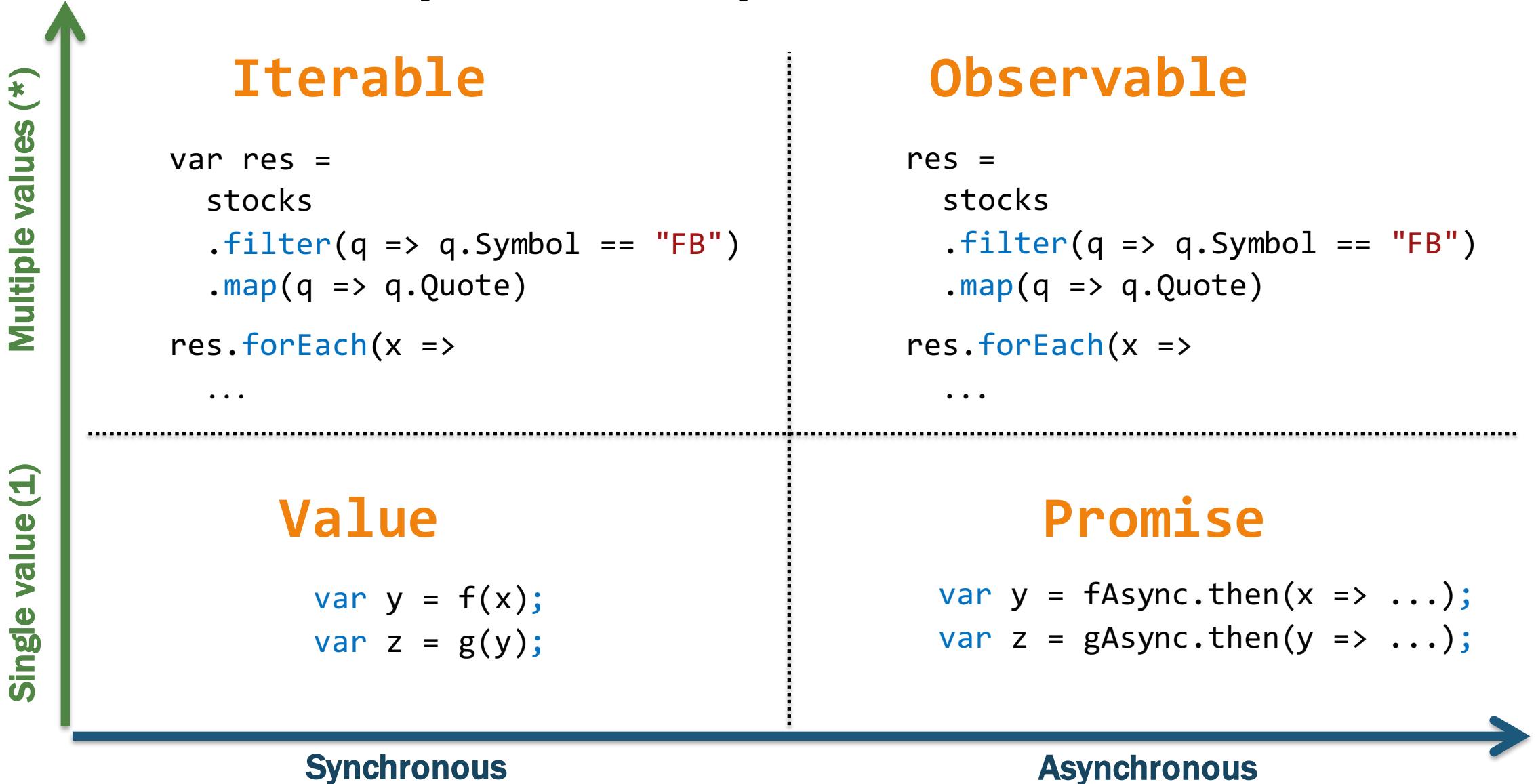
An object is **first-class** when it:<sup>[4][5]</sup>

- 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)

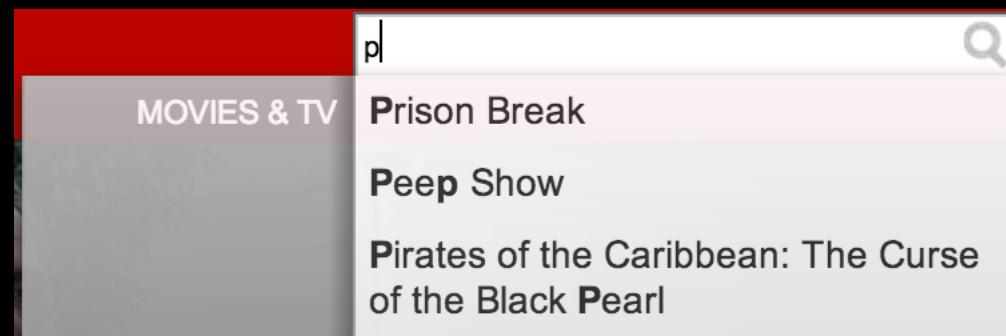


**WIKIPEDIA**  
*The Free Encyclopedia*

# The General Theory of Reactivity



# Type Ahead Search



# Type Ahead Search with Observables

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

```
let subscription = data.subscribe((data) => {
    // Bind data to the UI
});
```

# Your Netflix Video Lists

## Netflix Row Update Polling

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

2 / 10

Band Baaja Baaraat

2010 NR 2h 19m

★★★★★

Shruti and Bittoo decide to start a wedding planning company together after they graduate from university, but romance gets in the way of business.

Ranveer Singh, Anushka Sharma

Comedies, Foreign Movies

Director: Maneesh Sharma

Top 10 for tester\_jhusain\_control

BAND BAAJA BAARAAT

DANIEL DAY-LEWIS There Will Be Blood

Band Baaja Baaraat

BROKEN ENGLISH

THE HUNTED

RAÑÍ

Popular on Netflix

THE WALKING DEAD

# Polling for Row Updates

```
function getRowUpdates(row) {  
  let scrolls = Rx.Observable.fromEvent(document, 'scroll');  
  let rowVisibilities =  
    scrolls.debounce(50)  
      .map((scrollEvent) => row.isVisible(scrollEvent.offset))  
      .distinctUntilChanged()  
      .share();  
  
  let rowShows = rowVisibilities.filter((v) => v);  
  let rowHides = rowVisibilities.filter((v) => !v);  
  
  return rowShows  
    .flatMap(Rx.Observable.interval(10))  
    .flatMap(() => row.getRowData().takeUntil(rowHides))  
    .toArray();  
}
```

# Netflix Player



# Player Callback Hell

```
function play(movieId, cancelButton, callback) {  
    var movieTicket,  
        playError,  
        tryFinish = function() {  
            if (playError) {  
                callback(null, playError);  
            }  
            else if (movieTicket && player.initialized) {  
                callback(null, ticket);  
            }  
        };  
    cancelButton.addEventListener("click", function() { playError = "cancel"; });  
    if (!player.initialized) {  
        player.init(function(error) {  
            playError = error;  
            tryFinish();  
        })  
    }  
    authorizeMovie(movieId, function(error, ticket) {  
        playError = error;  
        movieTicket = ticket;  
        tryFinish();  
    });  
});
```



# Player With Observables

```
let authorizations =  
  player.init().flatMap(() =>  
    playAttempts.flatMap((movieId) =>  
      player.authorize(movieId)  
        .retry(3)  
        .takeUntil(cancels))  
    )  
  );  
let subscription = authorizations.subscribe(  
  (license) => player.play(license),  
  (error) => showDialog('Sorry, can't play right now.'));
```





# The Final Countdown...

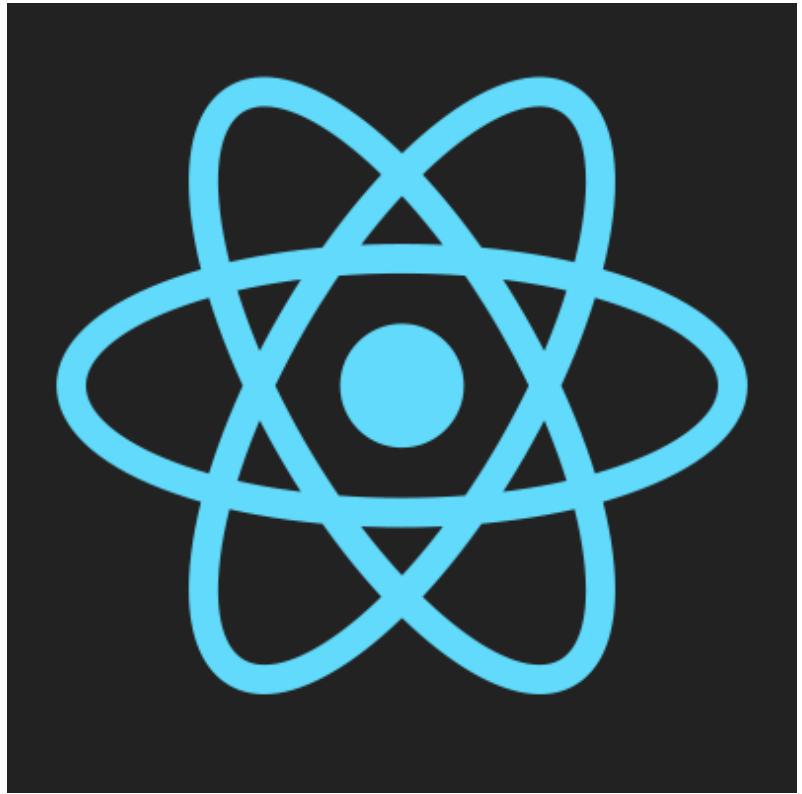


```
try {  
  let resource = getData();  
  for (let data of resource) {  
    processItem(data);  
  }  
} catch (e) {  
  throw e;  
} finally {  
  resource && resource.dispose();  
}
```

```
var d = getData();  
d.retry(3)  
.catch(defaultData)  
.finally(() => d.cleanup())  
.forEach(data =>  
  processItem(data));
```



# 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

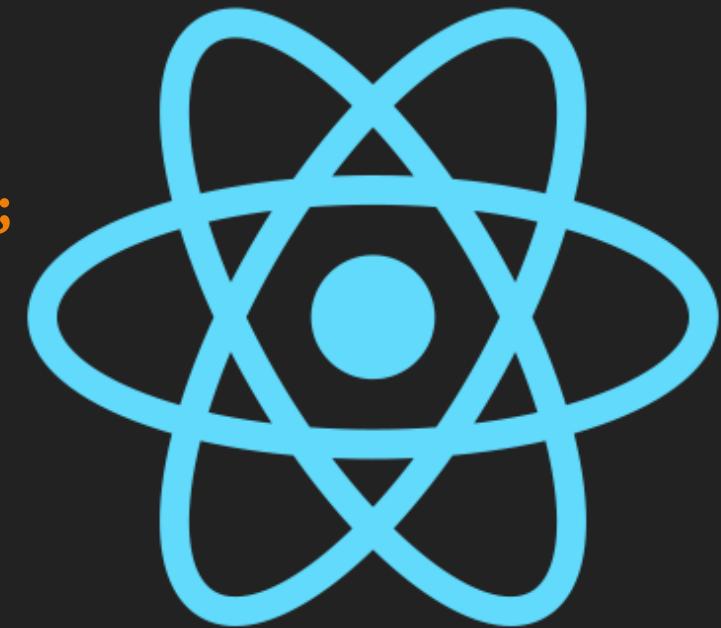
1 parent [363b9ba](#) commit [21568106b114943b59ce37832d82c8fb9a63285b](#)

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

[Unified](#) [Split](#)

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

```
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>
    );
  }
});
```



React

The  
Future  
of

JS

# **Ur Kitteh of Death**

## **Awaits**

# Async/Await

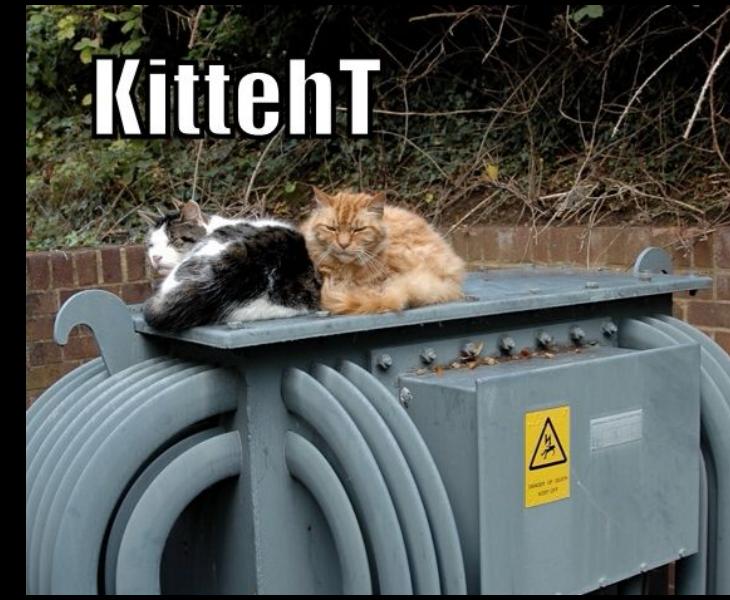
## Coming to a JavaScript Engine Near You!

- Adds `async` and `await` keywords for Promises
- Accepted into Stage 1 of ECMAScript 7 in January 2014

```
async function chainAnimationsAsync(elem, animations) {  
    let ret = null;  
    try {  
        for (let anim of animations) {  
            ret = await anim(elem);  
        }  
    } catch (e) { /* ignore and keep going */ }  
    return ret;  
}
```

# Async/Await with Observables and Generators...

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





## Asynchronous Programming with Futures and Streams

- Futures are like JavaScript Promises
- Streams unify I/O and events
- Mirrors Rx by adding map/filter/reduce, etc

```
Stream<List<int>> stream = new File('quotes.txt').openRead();
stream.transform(UTF8.decoder).listen(print);

querySelector('#myButton').onClick.listen((_) => print('Click.'));
```

# 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) {  
  
    let moveStreams = listen(element, "mousedown").map(e => {  
  
        e.preventDefault();  
  
        return takeUntil(  
            listen(element, "mousemove"),  
            listen(document, "mouseup"));  
    });  
  
    return switchLatest(moveStreams);  
}  
  
let 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>

This is an interactive learning course with exercises you fill out right in the browser. If you just want to browse the content click the button below:

Show all the answers so I can just browse.

# Functional Programming in Javascript

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

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

Here's my promise to you: if you learn these 5 functions your code will become shorter, more self-descriptive, and more durable. Also, for reasons that might not be obvious right now, you'll learn that these five functions hold the key to simplifying asynchronous programming. Once you've finished this tutorial you'll also have all the tools you need to easily avoid race conditions, propagate and handle asynchronous errors, and sequence events and AJAX requests. In short, **these 5 functions will probably be the most powerful, flexible, and useful functions you'll ever learn.**

# RxMarbles

## Interactive diagrams of Rx Observables

### TRANSFORMING OPERATORS

[delay](#)

[delayWithSelector](#)

[findIndex](#)

[map](#)

[scan](#)

[throttle](#)

[throttleWithSelector](#)

### COMBINING OPERATORS

[combineLatest](#)

[concat](#)

[merge](#)

[sample](#)

[startWith](#)

[zip](#)

### FILTERING OPERATORS

[distinct](#)

[distinctUntilChanged](#)

[elementAt](#)

[filter](#)



merge

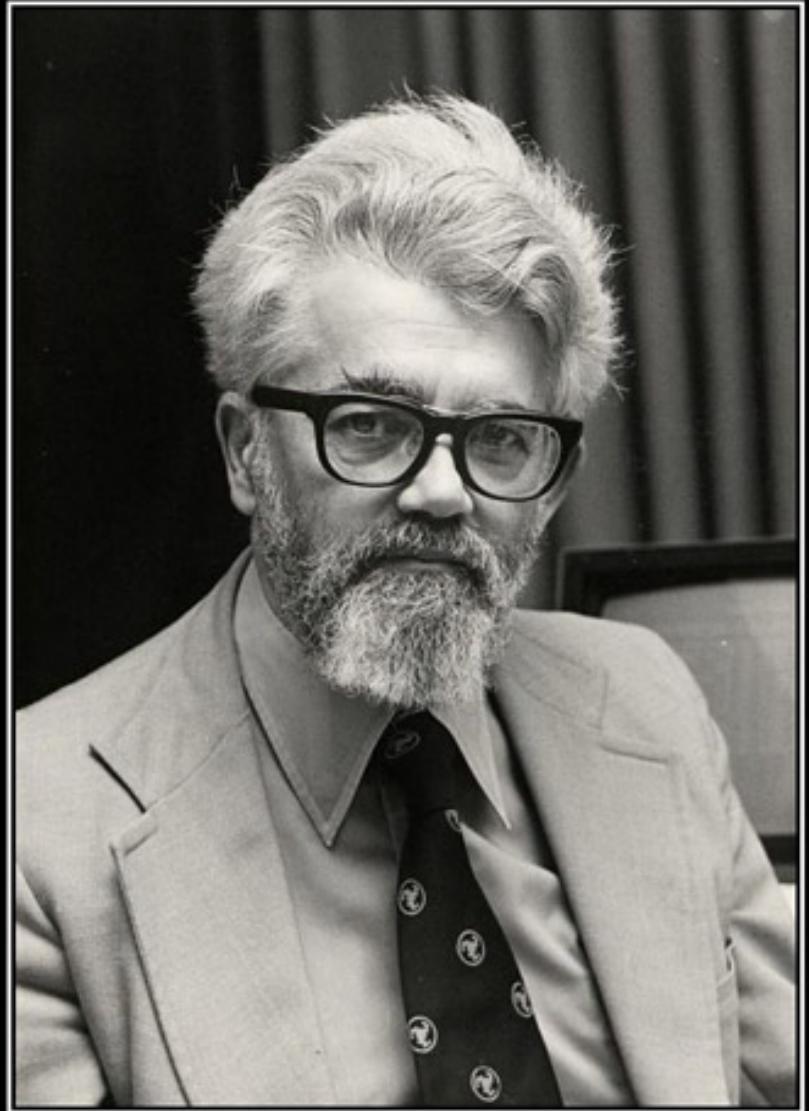


# SWEETEN YOUR JAVASCRIPT

[Compile](#)[Eval](#)[Step 0](#) readable names auto-compile  
 macro highlighting[vim](#)  
[emacs](#)  
[default](#)

```
1 /*  
2  * welcome to sweet.js!  
3  
4  You can play around with macro writing here on the left side and  
5  your code will automatically be compiled on the right. This page  
6  will also save your code to localStorage on every successful  
7  compile so feel free to close the page and come back later!  
8 */  
9  
10 // Here is a really simple identity macro to get started.  
11  
12 // The `macro` keyword is used to create and name new macros.  
13 macro id {  
14     rule {  
15         // after the macro name, match:  
16         // (1) a open paren  
17         // (2) a single token and bind it to '$x'  
18         // (3) a close paren  
19         ($x)  
20     } => {  
21         // just return the token we bound to '$x'  
22         $x  
23     }  
24 }  
25 id ( 42 );  
26  
27 // Note that a single token to sweet.js includes matched  
28
```

```
1 42;  
2 // Note that a single token to sweet.js includes matched  
3 // delimiters not just numbers and identifiers. For example,  
4 // an array with all of its elements counts as one token:  
5 [  
6     1,  
7     2,  
8     3  
9 ];  
10 // one of the really important things sweet.js does is protect  
11 // macros from unintentionally binding or capturing variables they  
12 // weren't supposed to. This is called hygiene and to enforce hygiene  
13 // sweet.js must carefully rename all variable names.  
14 var x;  
15 var foo = 100;  
16 var bar = 200;  
17 var tmp = 'my other temporary variable';  
18 var tmp$2 = bar;  
19 bar = foo;  
20 foo = tmp$2;
```



# PROGRAMMING

YOU'RE DOING IT COMPLETELY WRONG.

## Questions?

Matthew Podwysocki @mattpodwysocki

[github.com/mattpodwysocki/ndc-oslo-2015](https://github.com/mattpodwysocki/ndc-oslo-2015)