Exploring the Reactive Extensions for JavaScript

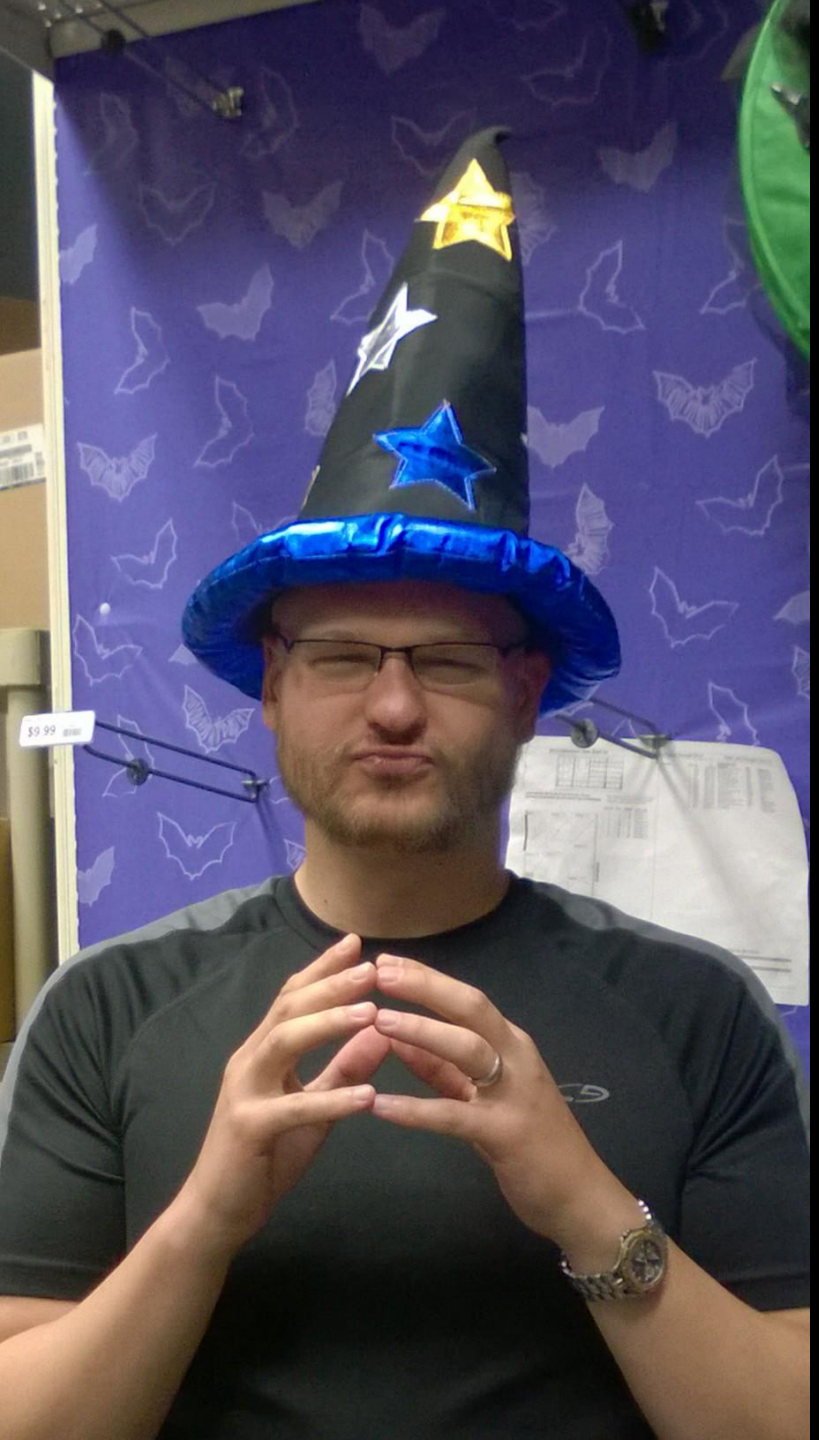
Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/applicative-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


trapd in Monad tutorl
plz help



Principal SDE
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki

MICRÖSÖFT

Reactive Extensions (Rx)

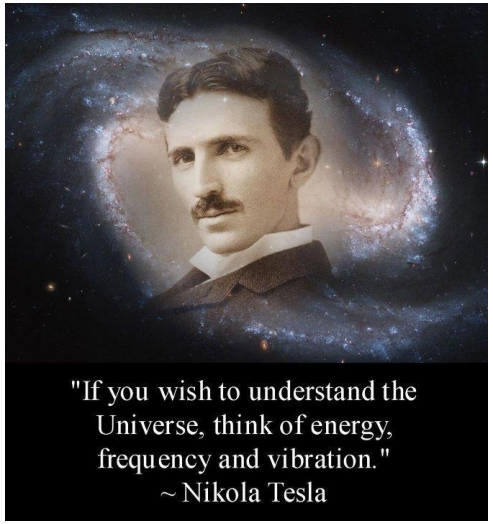
@ReactiveX

http://reactivex.io

An Accidental Discovery

Cloud Programmability Team

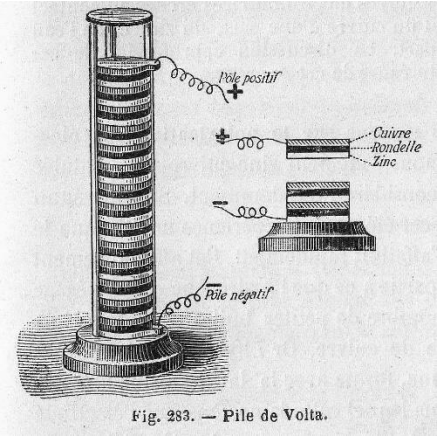
- "Oasis" within Live Labs and later the SQL Server organization
  - Founded by Erik Meijer and Brian Beckman
  - Code-named "Tesla" after Nikola Tesla (not the car)
- Founded in the mid 2000s
  - Making sense of this new thing called "cloud"
- Various projects
  - IL2JS – a compiler from IL to JavaScript
  - Extension to JavaScript with classes, modules, types
  - Embarrassingly distributed build system for the cloud
  - Reactive Extensions aka "LINQ to Events"



Nikola Tesla

www.knowledgeoftoday.org

An Accidental Discovery



www.Wikipedia.org

Project "Volta"

- Tier-splitting of applications
- Write as single-tier .NET application using metadata annotations
  - Attributes like [RunOnClient]
  - Cross-compilation of code to match client capabilities
    - Desktop CLR or Silverlight when available
    - IL to JavaScript when necessary
      - » Even compiling Windows Forms controls to HTML
- No promises or futures
  - async/await with Task<T> was unheard of (C# 5.0)
  - But the web is asynchronous…

An Accidental Discovery

An electric eel…

Project “Volta”

- Dealing with asynchrony across tiers

```
[RunOnClient]
public event EventHandler<MouseEventArgs> MouseMoved;

// Runs in cloud
public void CloudCanvas()
{
    MouseMoved += (o, e) => { /* do stuff */ };
}
```

- Ultimately needs to cross-compile to AJAX
- Events are not first-class objects
  - Can’t transport them across tiers

Making Events First-Class

First class === has object representation

http://en.wikipedia.org/wiki/First-class_citizen

- Methods can be transported using delegates

```
Action a = new Action(Foo); // explicit creation of delegate instance
Action b = Foo;             // method group conversion
Action c = () => { … };     // creates anonymous method

void Foo() { … }
```

- But properties, indexers, and events are metadata citizens

```
event Action Bar  // metadata that refers to …
{
    add    { … }  // add accessor
    remove { … }  // remove accessor
}
```


streem prosesing

Real-Time is Everywhere…

Let’s Face It, Asynchronous Programming is Awful!

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

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

Callback Hell

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


culturepub.fr
next
ad

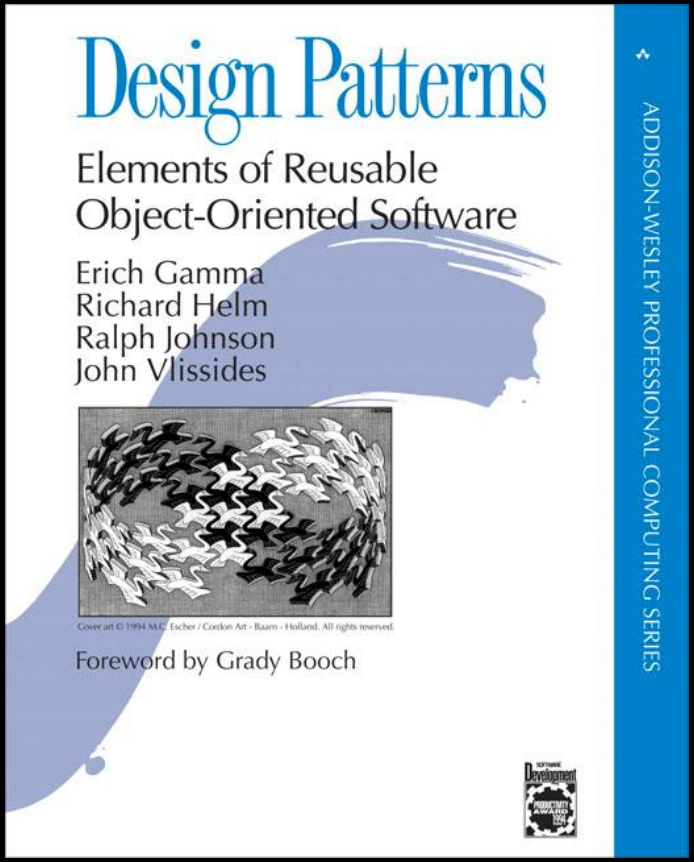
Events and the Enemy of the State

```
var isDown = false, state;

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

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

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

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

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

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

Iterator Pattern

```
> var 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 }
>
```

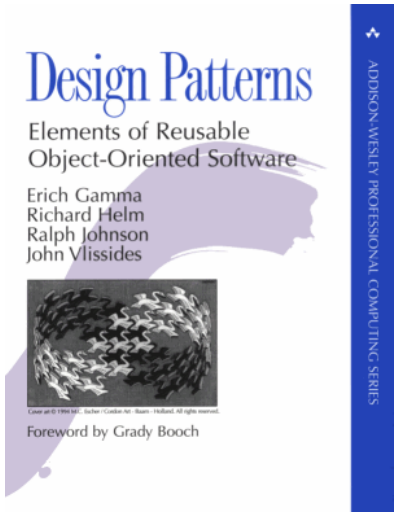
Subject/Observer Pattern

```
> document.addEventListener(
    "mousemove",
    function next(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 }
```

Fundamental Abstractions

Adapting the observer pattern

- Ensuring duality with the enumerator pattern
- More compositional approach



"Gang of four" book
Addison-Wesley

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

The Beauty of Duality

Category theory
www.Wikipedia.org

Category theory to the rescue

- Observable/observer (push) is dual to enumerable/enumerator (pull)
- Cross-influence of both domains

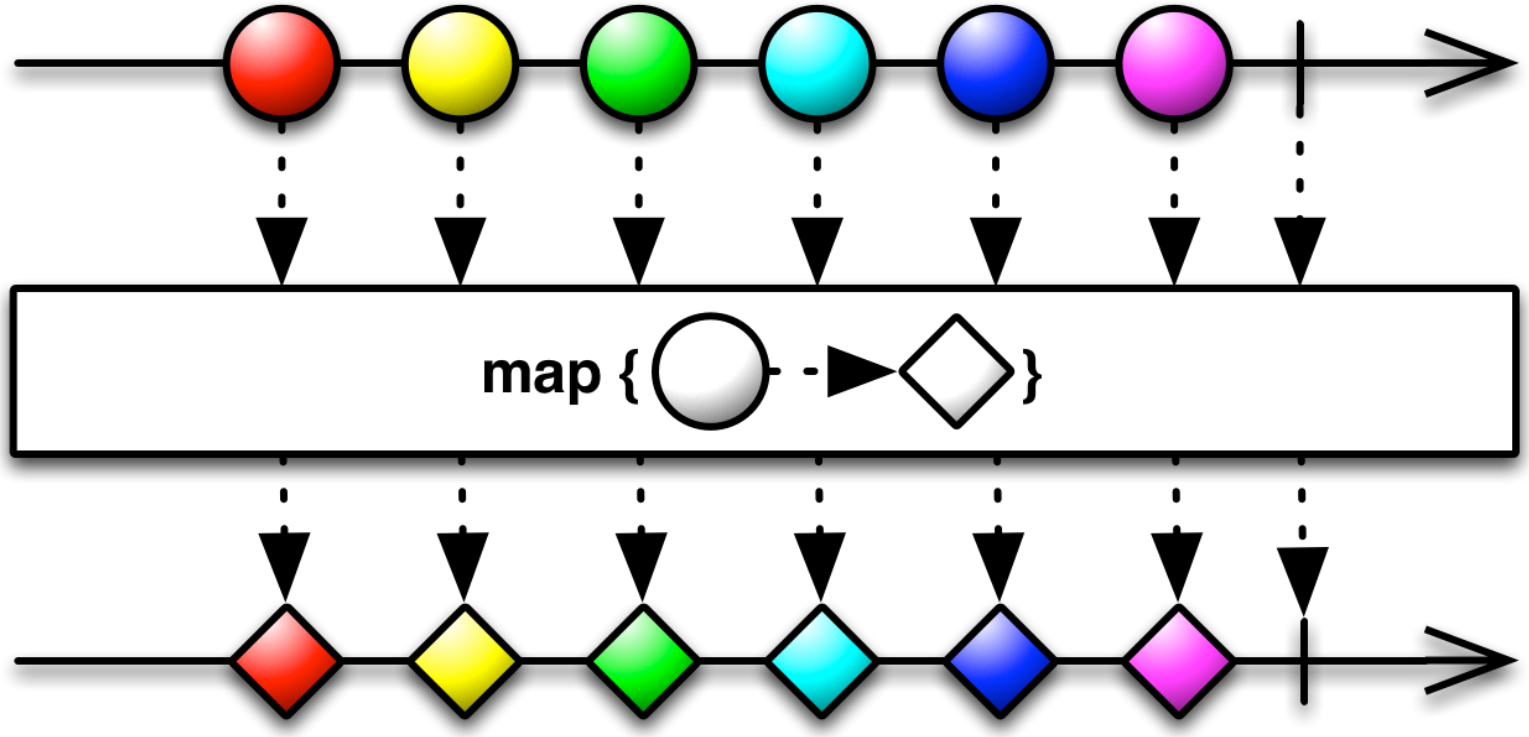
```
interface Observable<T> {
  subscribe(observer : Observer<T>) : Disposable
}

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

```
interface Enumerable<T> {
  [Symbol.iterator]() : Enumerator<T>
}

interface Enumerator<T> {
  next() : EnumeratorValue<T> // throws
}

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

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

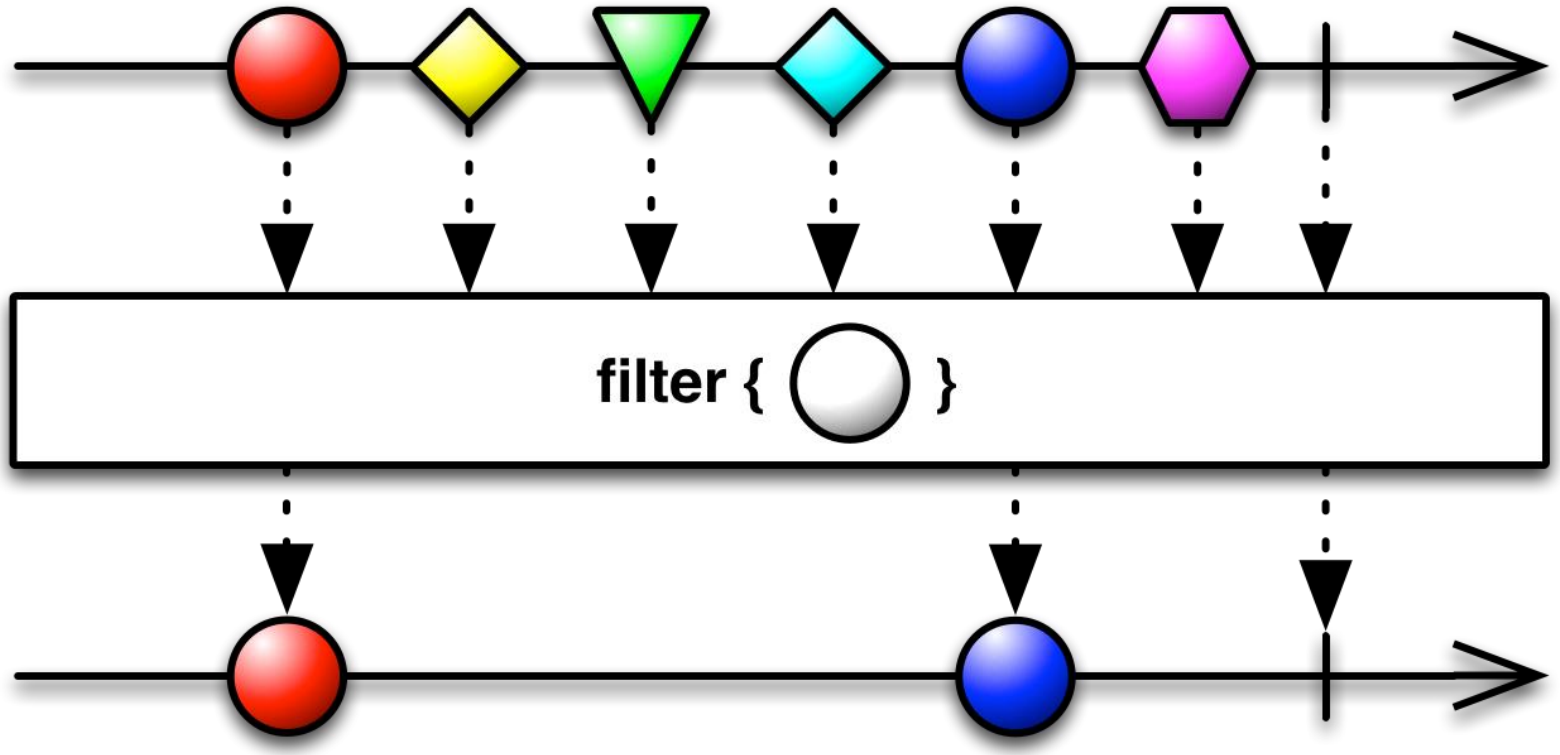
map()

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



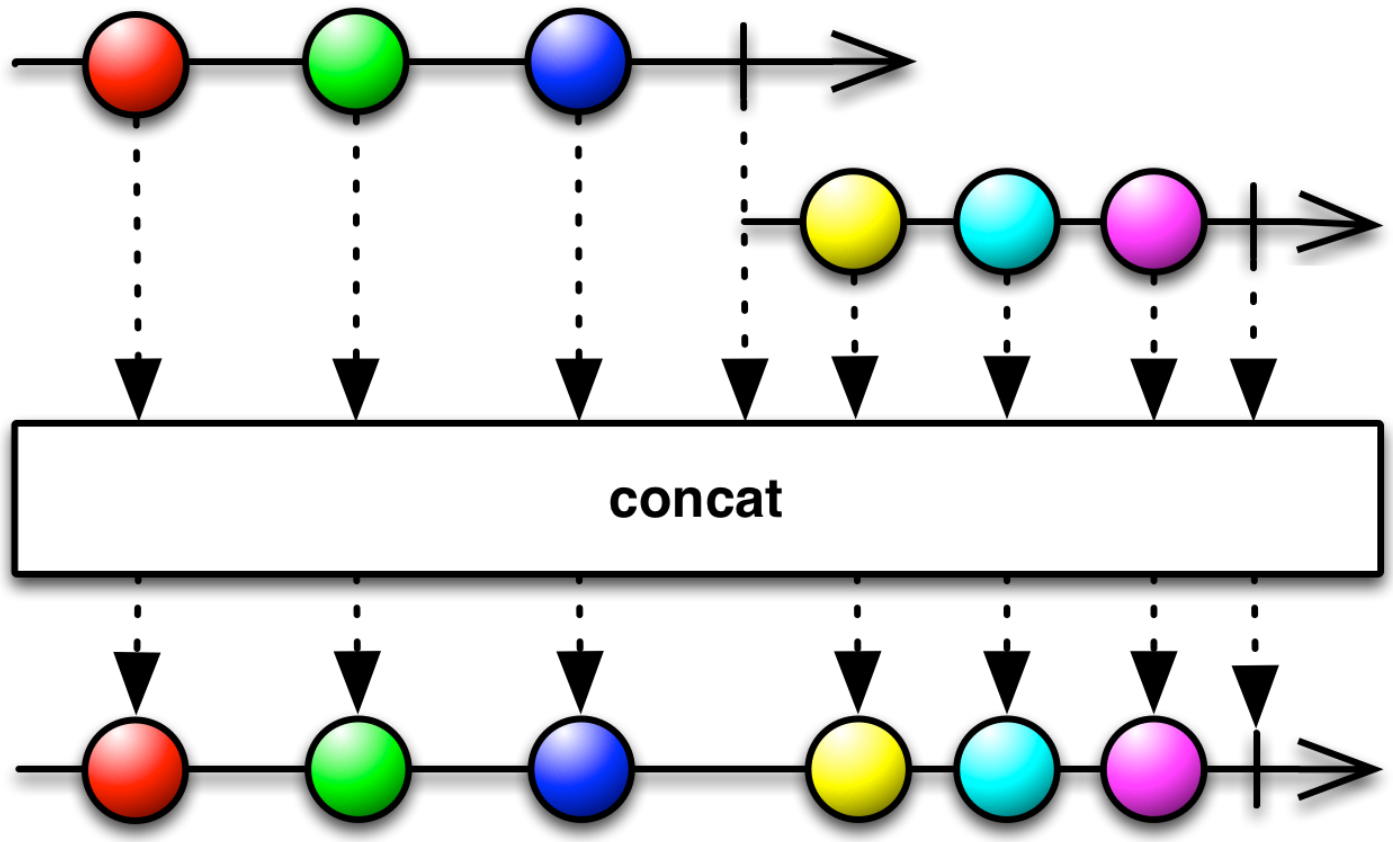
filter()

Filter items emitted by a Collection



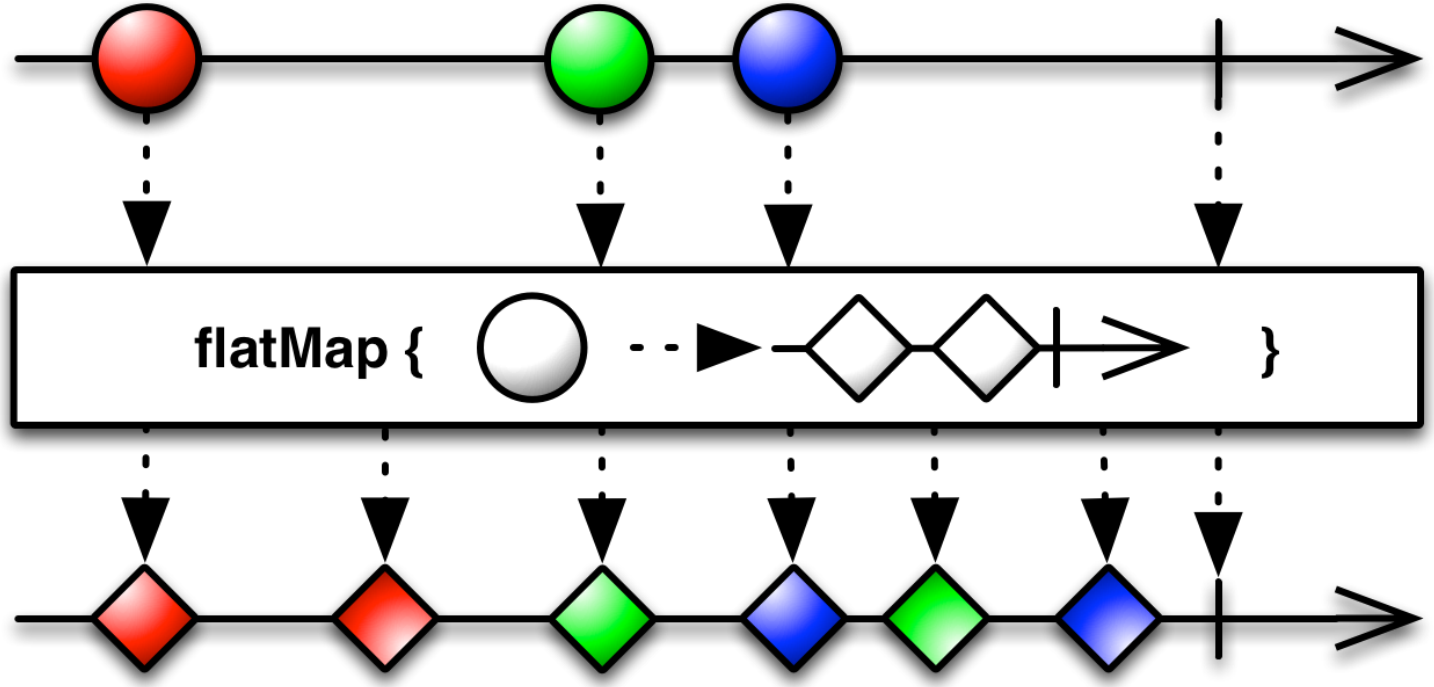
concatAll()

Concatenate two or more Collections sequentially



flatMap()

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



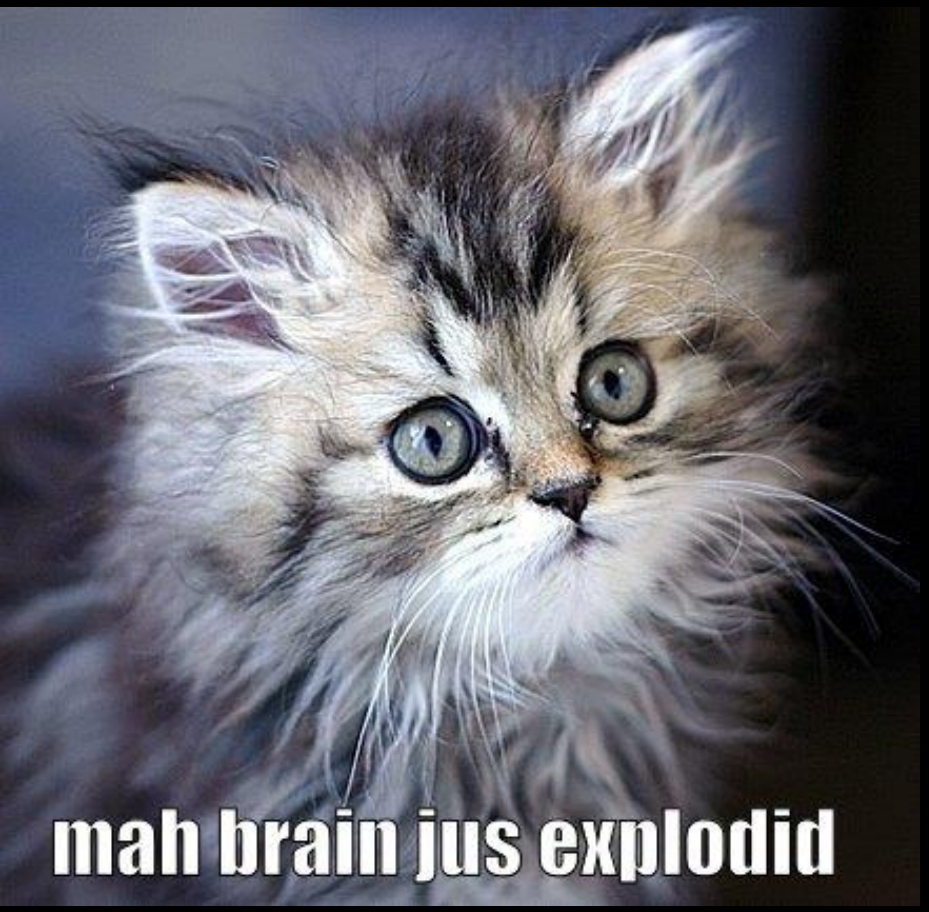
Top-rated Movies Collection

```
var getTopRatedFilms = function (user) {
  return user.videoLists
    .map(function (videoList) {
      return videoList.videos
        .filter(function (v) { return v.rating === 5; });
    }).concatAll();
}

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

Top-rated Movies Collection

```
var getTopRatedFilms = function (user) {
  return user.videoLists
    .flatMap(function (videoList) {
      return videoList.videos
        .filter(function (v) { return 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

```
var getElementDrags = function (elmt) {
  return dom.mousedown(elmt)
    .map(function (md) {
      return dom.mousemove(document)
        .filter  .takeUntil(dom.mouseup(elmt));
    }).concatAll();
};

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

Mouse Drags Collection

```
var getElementDrags = function (elmt) {
  return dom.mousedown(elmt)
    .flatMap(function (md) {
      return 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

Multiple values (*)

Single value (1)

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(...);
```

Synchronous

Asynchronous

Promises Promises

then

```
promise.then(onFulfilled, onRejected);
```

Why?

- Only one callback will be called either onFulfilled or onRejected
- Handlers called asynchronously
- If settled, then calls the handlers once attached

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

Promises Promises

then

Problems in Promiseland

- How do I handle cancellation?
- What if I don't care about the return value ala Autocomplete?

```
var promise;

input.addEventListener('keyup', function (e) {

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

What is Reactive Programming Anyhow?

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

Wanna really know what Reactive Programming Is?

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

http://bit.ly/reactive-paper

Functional Reactive Programming (FRP) is...

A concept consisting of

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

What it is not

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

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

You already know how to do this....

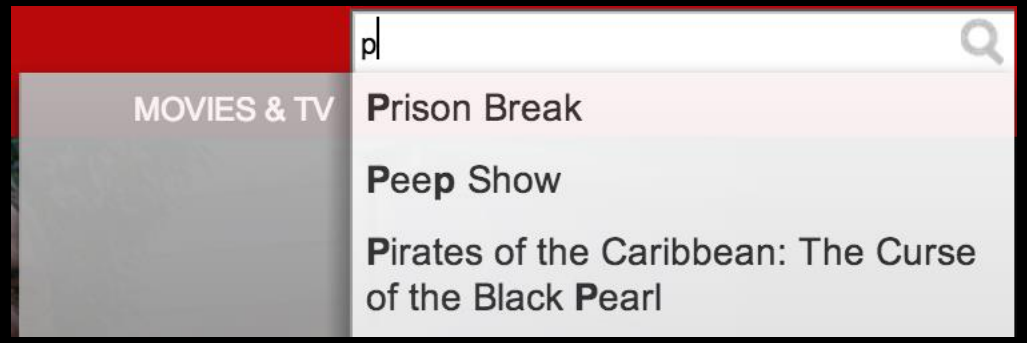
INTERACTIVE

```
var source = getStockData();

source
  .filter(function (quote) {
      return quote.price > 30;
  })
  .map(function (quote) {
      return quote.price;
  })
  .forEach(function (price) {
    console.log(‘Higher than $30: $' + price);
  });
```

REACTIVE

```
var source = getStockData();

source
  .filter(function (quote) {
      return quote.price > 30;
  })
  .map(function (quote) {
      return quote.price;
  })
  .forEach(function (price) {
    console.log(‘Higher than $30: $' + price);
  });
```

Netflix Search



Autocomplete with Observables

```
var data = dom.keyup(input)
           .map(function() { return input.value; })
           .debounce(500)
           .distinctUntilChanged()
           .flatMapLatest(
              function(term) { return search(term); }
           );

data.subscribe(function(data) {
  // Bind data to the UI
});
```

DOM events as a sequence of strings

Reducing data traffic / volume

Latest response as movies

Web service call returns single value sequence

Binding results to the UI

What exactly is Rx?

Language neutral model with 3 concepts:

1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
   - Schedulers: a set of types to parameterize concurrency

Rx Grammar Police

onNext *

Zero or more values

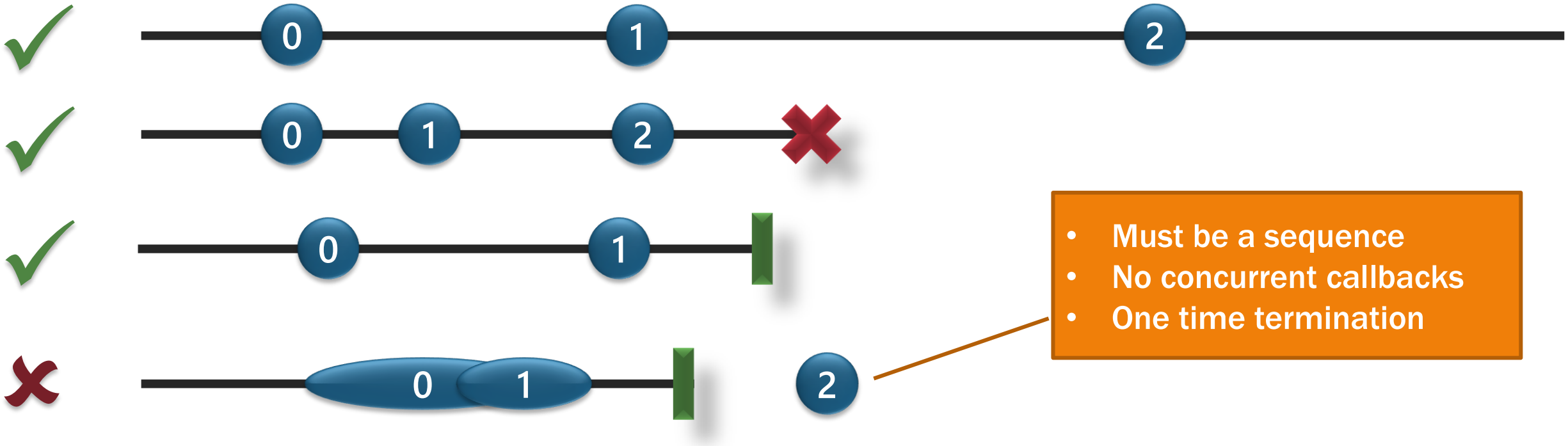
E.g. events are ∞ sequences

(onError | onCompleted) ?

Calls can fail

Resource management

Sequencing



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

What exactly is Rx?

Language neutral model with 3 concepts:

1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
   - Schedulers: a set of types to parameterize concurrency

Observables - Querying UI Events

```
var mousedrag = mousedown.flatMap(function (md) {

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



});
```

1

For each mouse down

Observables - Querying UI Events

```
var mousedrag = mousedown.flatMap(function (md) {

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

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

1 For each mouse down

2 Take mouse moves

Observables - Querying UI Events

```
var mousedrag = mousedown.flatMap(function (md) {

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

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

1 For each mouse down

2 Take mouse moves

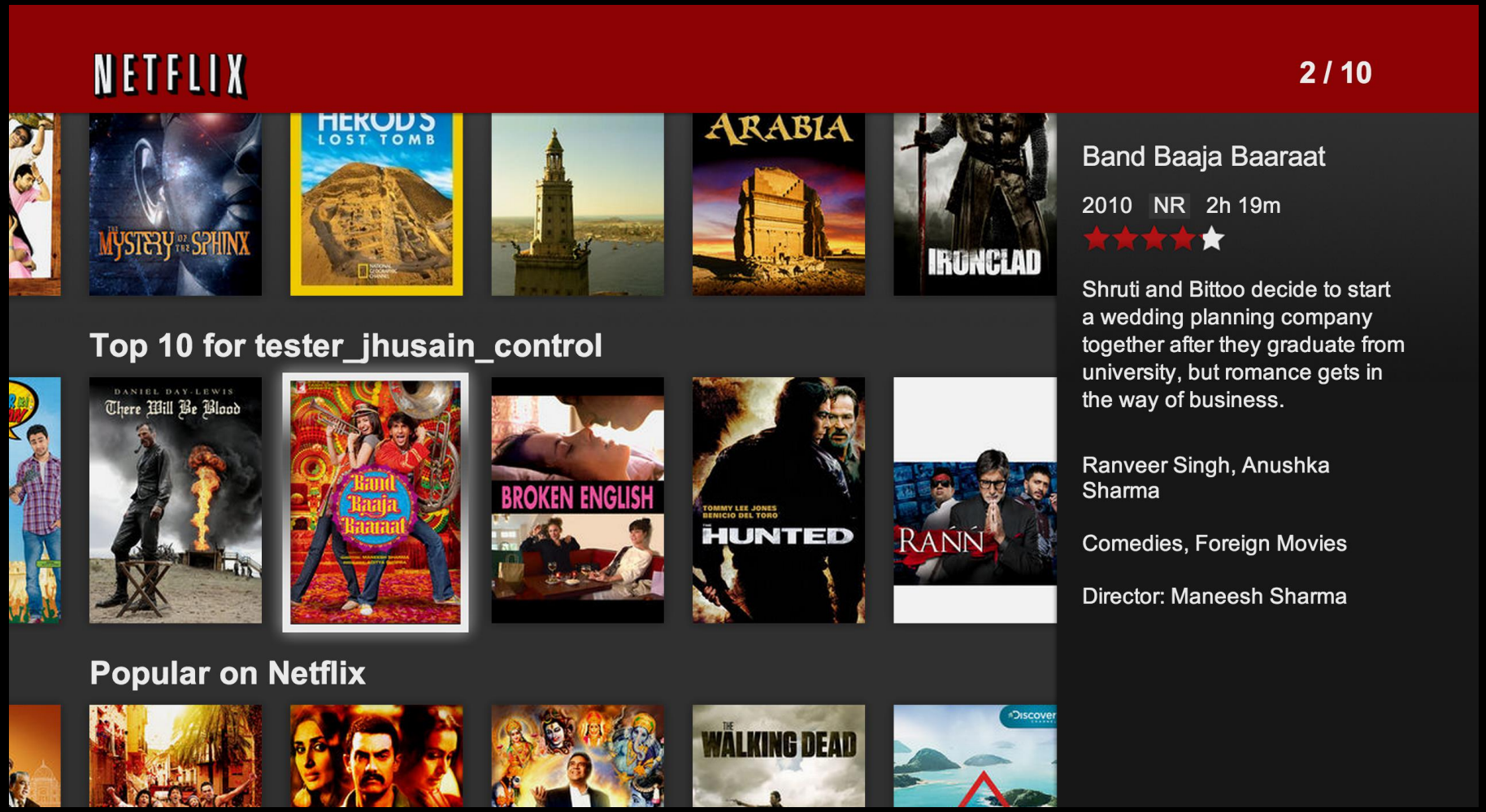
3 until mouse up

PROTONIC REVERSAL

You crossed the streams, didn't you?

Your Netflix Video Lists

Netflix Row Update Polling



Client: Polling for Row Updates

```
function getRowUpdates(row) {
    var scrolls = Rx.Observable.fromEvent(document, "scroll");
    var rowVisibilities =
        scrolls.throttle(50)
             .map(function (scrollEvent) { return row.isVisible(scrollEvent.offset); })
             .distinctUntilChanged()
             .publish().refCount();
    var rowShows = rowVisibilities.filter(function (v) { return v; });
    var rowHides = rowVisibilities.filter(function (v) { return !v) });

    return rowShows
       .flatMap(Rx.Observable.interval(10))
       .flatMap(function () { return 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

```
var authorizations =
   player
      .init()
      .flatMap(function () {
         return playAttempts
            .flatMap(function (movieId) {
               return player.authorize(movieId)
                  .retry(3)
                  .takeUntil(cancels));
            })
      });
authorizations.forEach(
   function (license) { player.play(license); },
   function (error) { showDialog("Sorry, can’t play right now."); });
```

What is Rx?

Language neutral model with 3 concepts:

1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
   - Schedulers: a set of types to parameterize concurrency

The Role of Schedulers

Key questions:

- How to run timers?
- Where to produce events?
- Need to synchronize with the UI?

Schedulers are the answer:

- Schedulers introduce concurrency
- Operators are parameterized by schedulers
- Provides test benefits as well

Cancellation

Many implementations

```
d = scheduler.schedule(
  function () {
    // Asynchronously
    // running work
  },
  1000);
```

Optional time

Testing concurrent code: made easy!

```
var scheduler = new TestScheduler();

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

var results = scheduler.startWithCreate(function () {
    return input.pluck('length');
});

results.messages.assertEqual(
    onNext(300, 11),
    onNext(400, 4),
    onCompleted(500));
```

Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure on the JVM.

The Problem

Handling streams of data—especially "live" data whose volume is not predetermined—requires special care in an asynchronous system. The most prominent issue is that resource consumption needs to be carefully controlled such that a fast data source does not overwhelm the stream destination. Asynchrony is needed in order to enable the parallel use of computing resources, on collaborating network hosts or multiple CPU cores within a single machine.

http://www.reactive-streams.org/

Observables and Backpressure

Yes, Observables can have backpressure

- Can be lossy (pausable, sample, throttle)
- Can be lossless (buffer, pausableBuffered, controlled)

```
var pausable = chattyObservable.pausableBuffered();
pausable.pause();
pausable.resume();

var subscription = chattyObservable.subscribe(print);
subscription.request(10);
```


Ur Kitteh of Death
Awaits
ROFLBOT

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

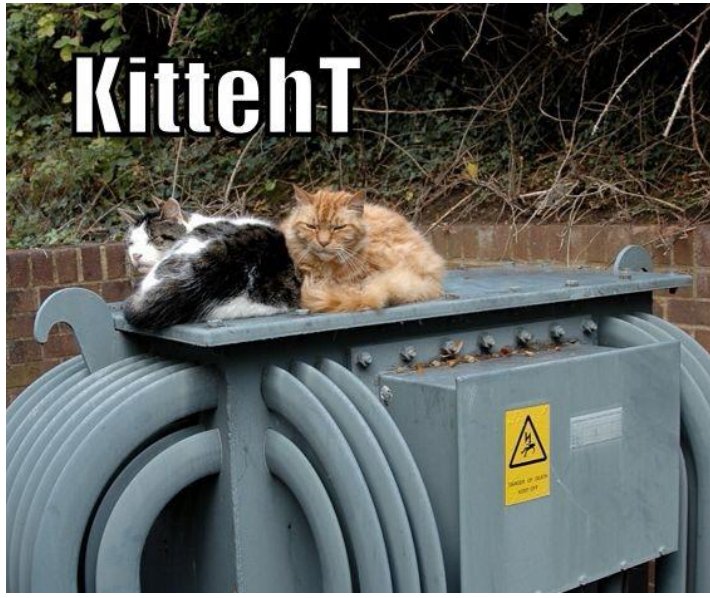
https://github.com/lukehoban/ecmascript-asyncawait

Async/Await with Observables and Generators…

RxJS and Generators

- Adds async /await capabilities to single value Observables
- Available in any runtime that has Generators

```
Rx.spawn(function* () {
  var result = yield get('http://applicative.acm.org/')
    .retry(3)
    .catch(cachedVersion);

  console.log(result);
}());
```



Async Generators

ES7 and Beyond!

- First class events in the JavaScript runtime
- Proposed in June 2014 at TC39

```
async function* getDrags(element) {
  for (let mouseDown on element.mouseDowns) {
    for (let mouseMove on
      document.mouseMoves.takeUntil(document.mouseUps)) {
        yield mouseMove;
    }
  }
}
```

http://esdiscuss.org/notes/2014-06/async%20generators.pdf

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

Show all the answers so I can just browse.

Functional Programming in Javascript

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.**

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

RxMarbles

Interactive diagrams of Rx Observables

TRANSFORMING OPERATORS

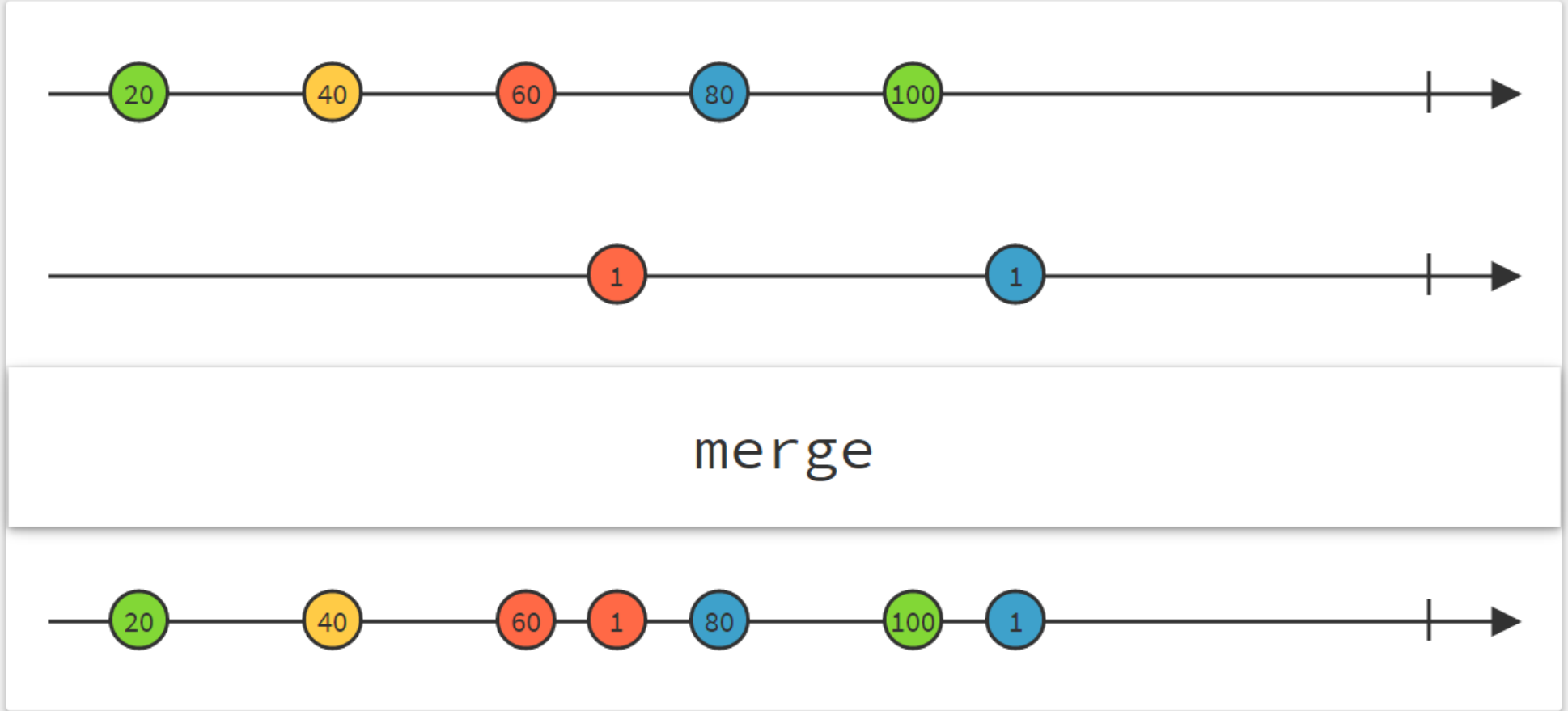
- delay
- delayWithSelector
- findIndex
- map
- scan
- throttle
- throttleWithSelector

COMBINING OPERATORS

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

FILTERING OPERATORS

- distinct
- distinctUntilChanged
- elementAt
- filter



http://www.rxmarbles.com/

RxJS In Action

SWEETEN YOUR JAVASCRIPT

Compile | Eval | Step 0 | readable names | auto-compile | macro highlighting | vim | emacs | default

```
/*
Welcome to sweet.js!

You can play around with macro writing here on the left side and
your code will automatically be compiled on the right. This page
will also save your code to localStorage on every successful
compile so feel free to close the page and come back later!
*/

// Here is a really simple identity macro to get started.

// The `macro` keyword is used to create and name new macros.
macro id {
    rule {
        // after the macro name, match:
        // (1) a open paren
        // (2) a single token and bind it to `$x`
        // (3) a close paren
        ($x)
    } => {
        // just return the token we bound to `$x`
        $x
    }
}
id ( 42 );

// Note that a single token to sweet.js includes matched
```

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

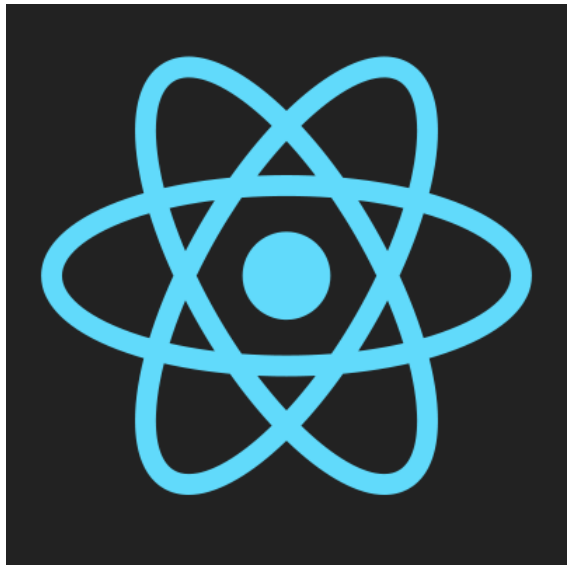
http://sweetjs.org/browser/editor.html

RxJS In Action


```
var documentReadyObs = $(window).readyAsObservable().take(1).publishLast(),
    windowResizeObs  = $(window).resizeAsObservable().startWith(true),

    initEditorObs = documentReadyObs
        .map(_.partial($, "#editor", undefined))
        .map(_.partial(initEditor, getEditorOptions))
        .map(initEditorKeyMap),

    initOutputObs = documentReadyObs
        .map(_.partial($, "#output", undefined))
        .map(_.partial(initEditor, getOutputOptions)),

    // Initialize both CodeMirror instances on document
    // ready, then select them into a list together.
    mirrors = initEditorObs.zip(initOutputObs, concat.bind([])).publish(),
```

https://github.com/mozilla/sweet.js/blob/master/browser/scripts/editor.js

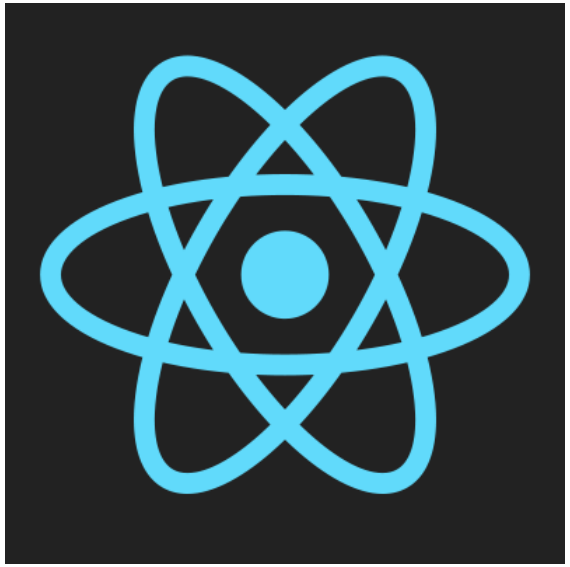
What About My Libraries?



Rx in a Virtual-DOM World

React Bridges

- Rx-React (https://github.com/fdecampredon/rx-react)
- Rx-Flux (https://github.com/fdecampredon/rx-flux)
- AsyncReact (https://github.com/jhusain/asyncreact)
- RxReact (https://github.com/AlexMost/RxReact)



Rx In a Virtual-DOM World

Cycle.js

Model-View-Intent architecture and Virtual-DOM Rendering

```
var Cycle = require('cyclejs');
var h = Cycle.h;

var Model = Cycle.createModel(Intent =>
  ({name$: Intent.get('changeName$').startWith('')})
});

var View = Cycle.createView(Model =>
  ({
    vtree$: Model.get('name$').map(name =>
      h('div', [
        h('label', 'Name:'),
        h('input.field', {attributes: {type: 'text'}}),
        h('h1.header', 'Hello ' + name)
      ])
    )
  })
);
```

RxJS Demos

Reactive Extensions

session.onCompleted()

@ReactiveX

http://reactivex.io