Async JavaScript at Netflix

Matthew Podwysocki @mattpodwysocki
github.com/mattpodwysocki/codemash2015

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

streem prosesing

NETFLIX

Stream Movies From Any Device

1/3 of US Broadband Traffic

This is the story of how Netflix solved

BIG async problems

by thinking differently about

Events.

The Netflix App is Asynchronous

- App Startup
- Player
- Data Access
- Animations
- View/Model Binding

Two Years Ago...

- Complex asynchronous code
- Client and Server developers tightly coupled
- Different platforms, different approaches to asynchrnous

Today at Netflix

- Rx used on server and client
- 30+ developers using Rx in 5 different languages
- Same asynchronous model everywhere

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);
```

“What’s the difference between an Array…

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

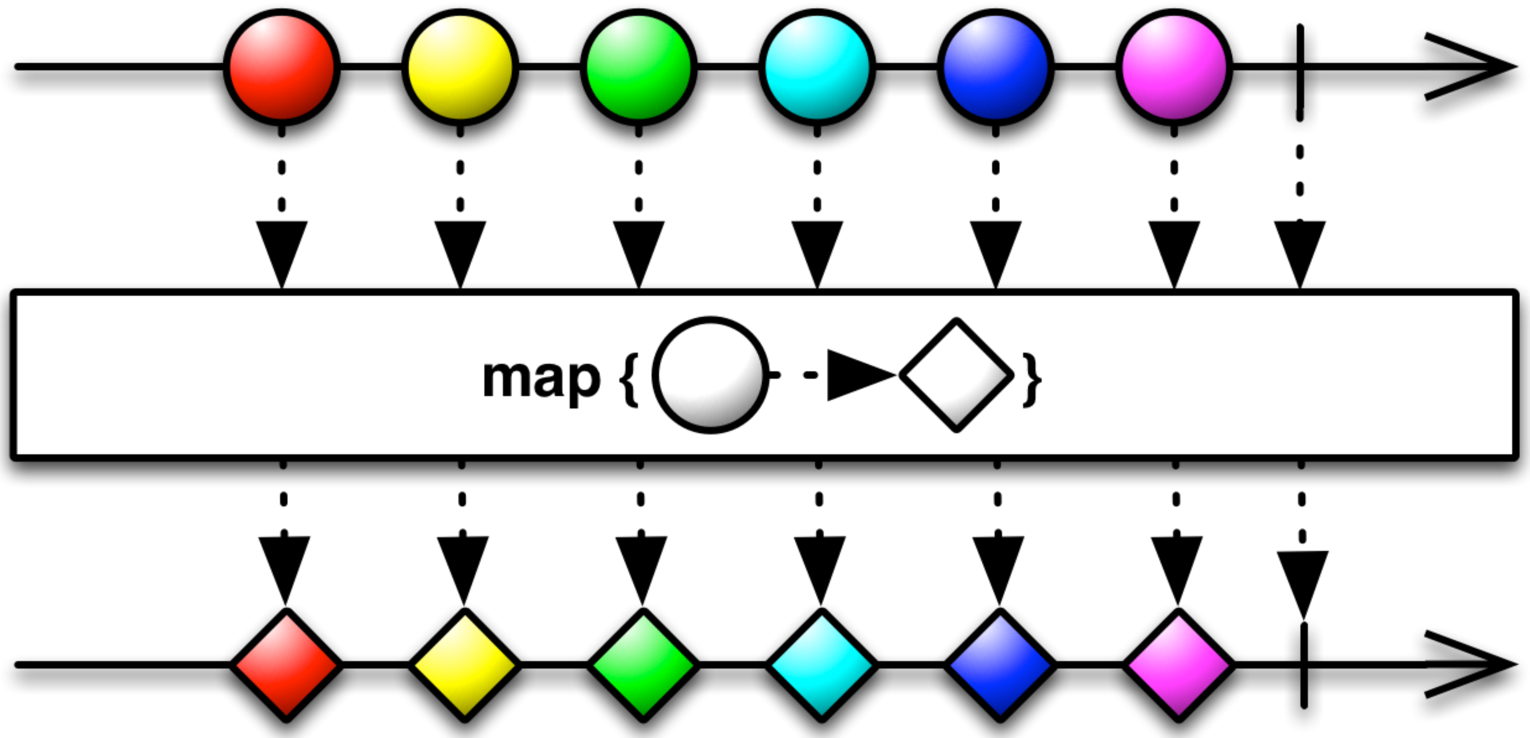
… and an Event?

Events and Arrays are *both* collections.

The majority of Netflix's asynchronous code is written with just a few *flexible* functions.

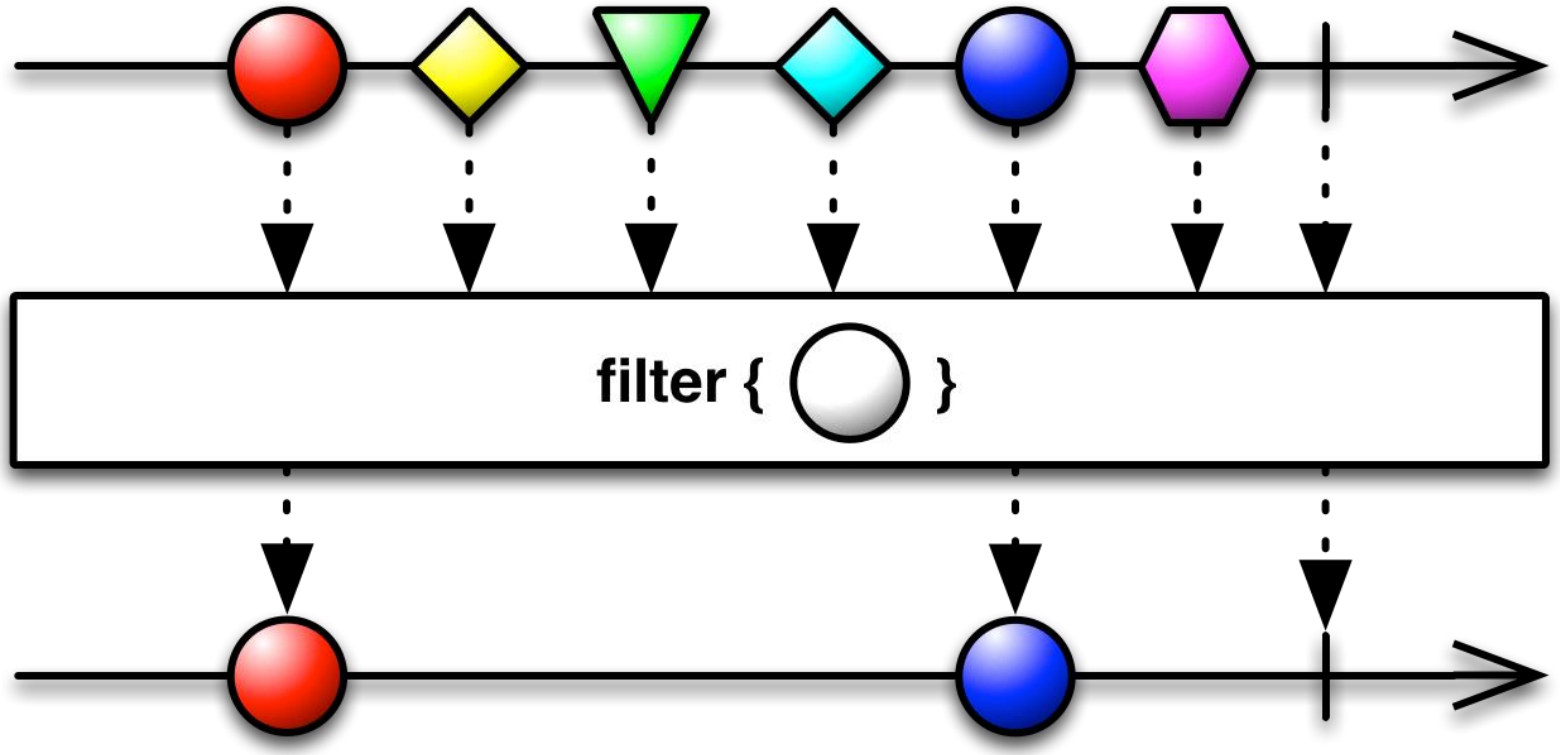
map()

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



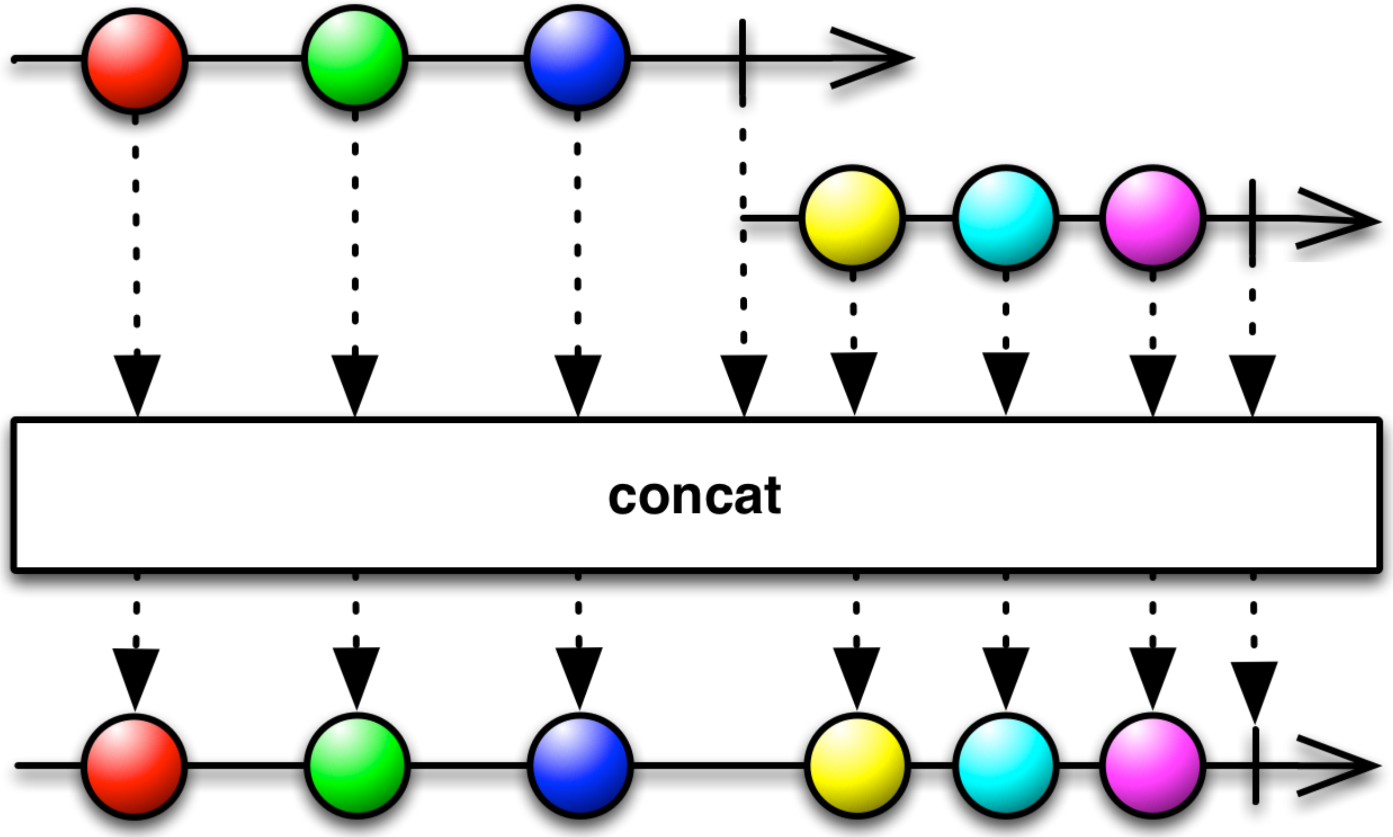
filter()

Filter items emitted by a Collection



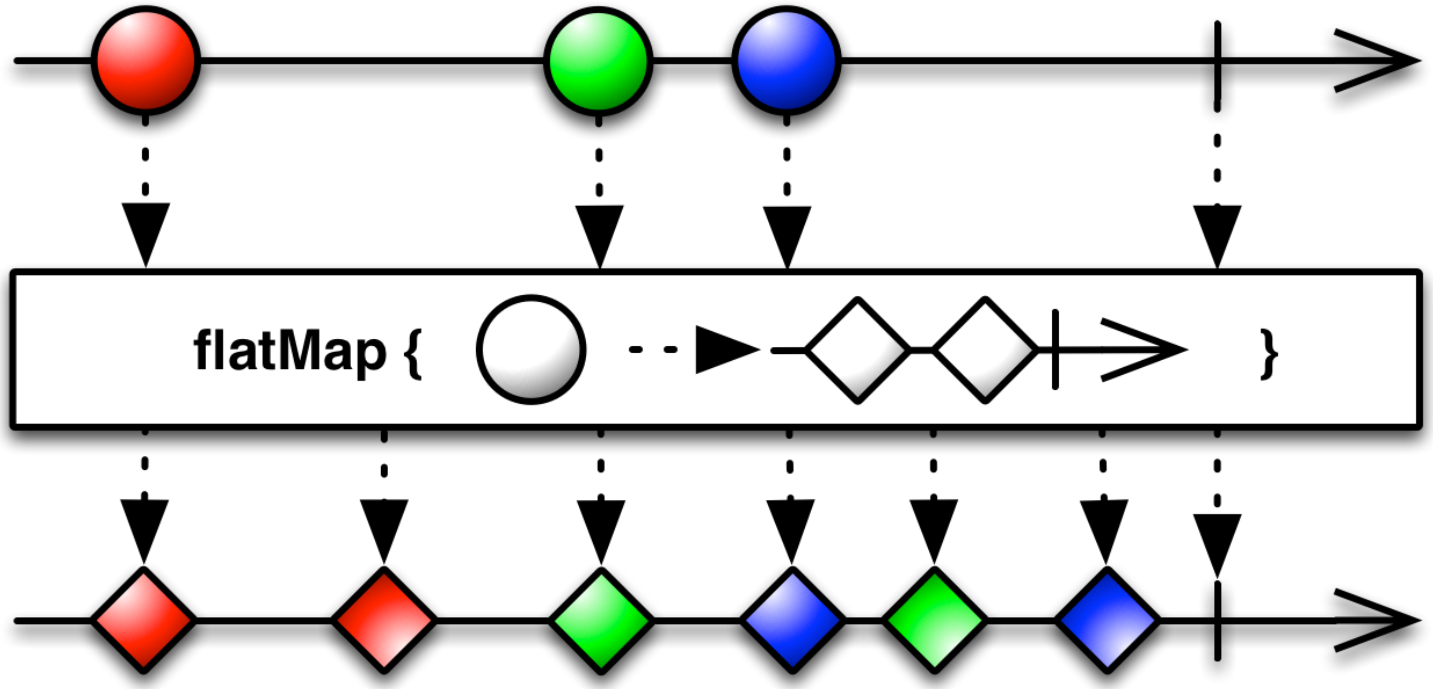
concatAll()

Concatenate two or more Collections sequentially



flatMap()

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



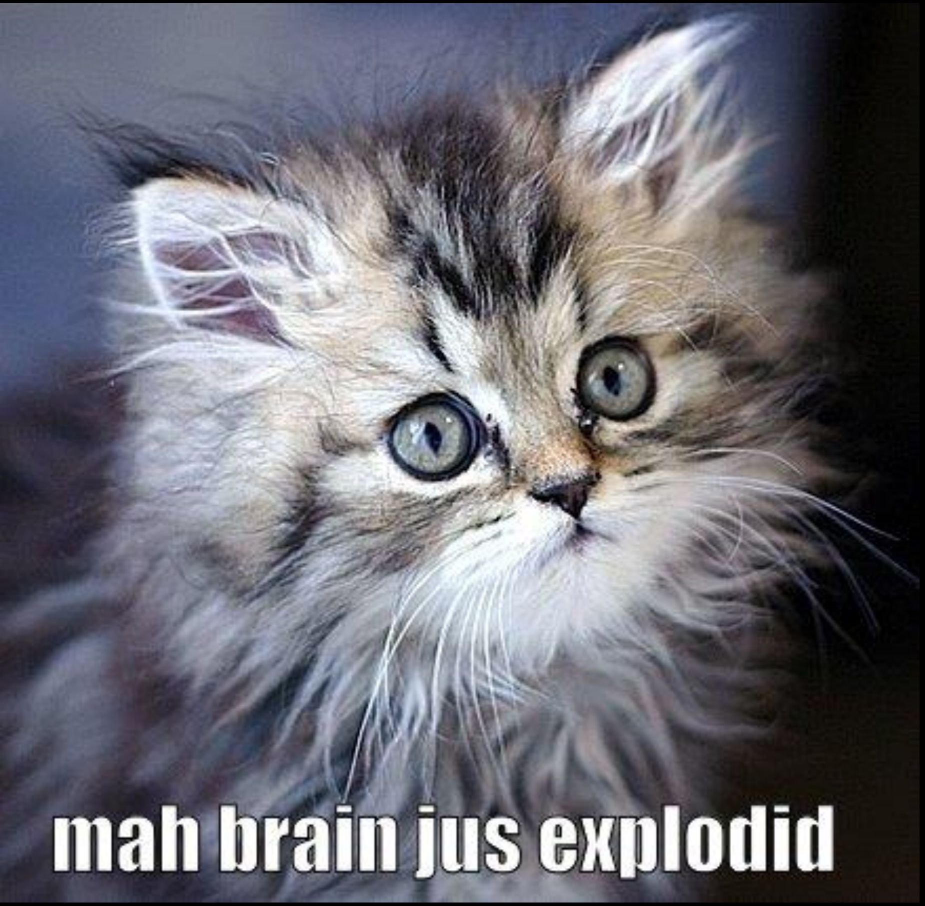
Top-rated Movies Collection

```
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

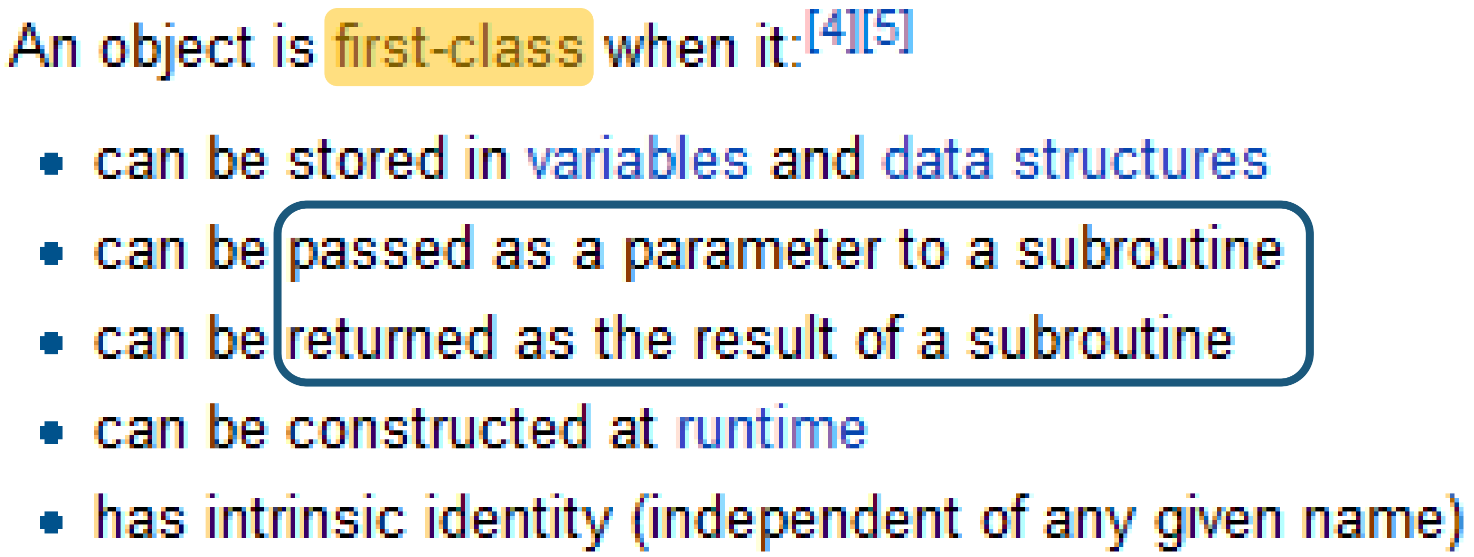
```
var getElementDrags = function (elmt) {
  return dom.mousedown(elmt)
    .map(function (md) {
      return dom.mousemove(document)
        ~~.filter~~  .takeUntil(dom.mouseup(elmt));
    }).concatAll();
};

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

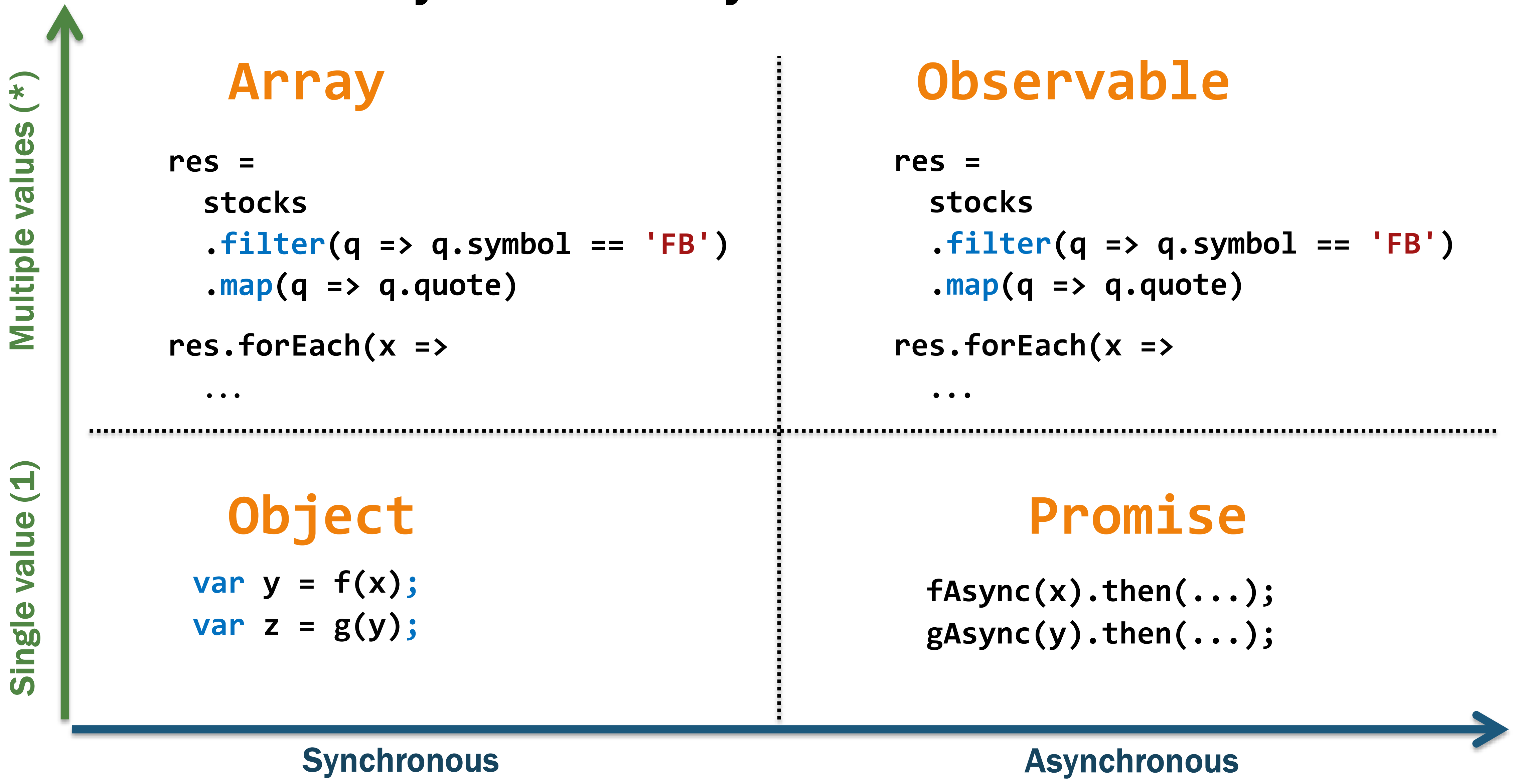
First-Class Asynchronous Values



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)

The General Theory of Reactivity



Array

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

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

Observable

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

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

Object

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

Promise

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

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….

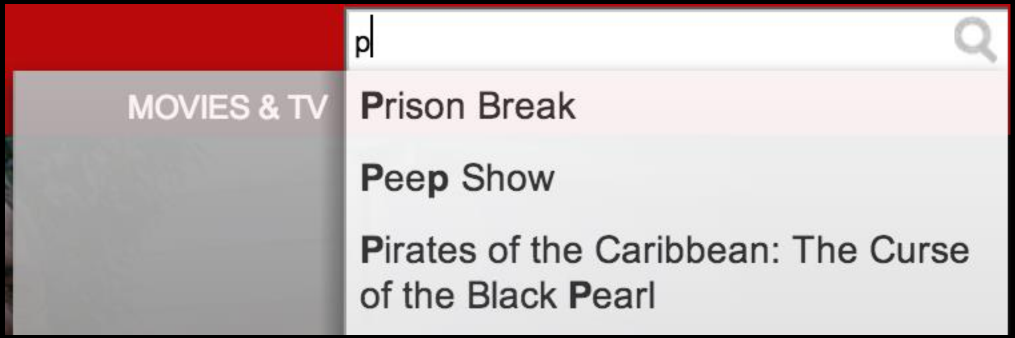
INTERACTIVE

```
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


p
MOVIES & TV
Prison Break
Peep Show
Pirates of the Caribbean: The Curse of the Black **P**earl

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 ● * (onError ✖ | onCompleted ▮) ?

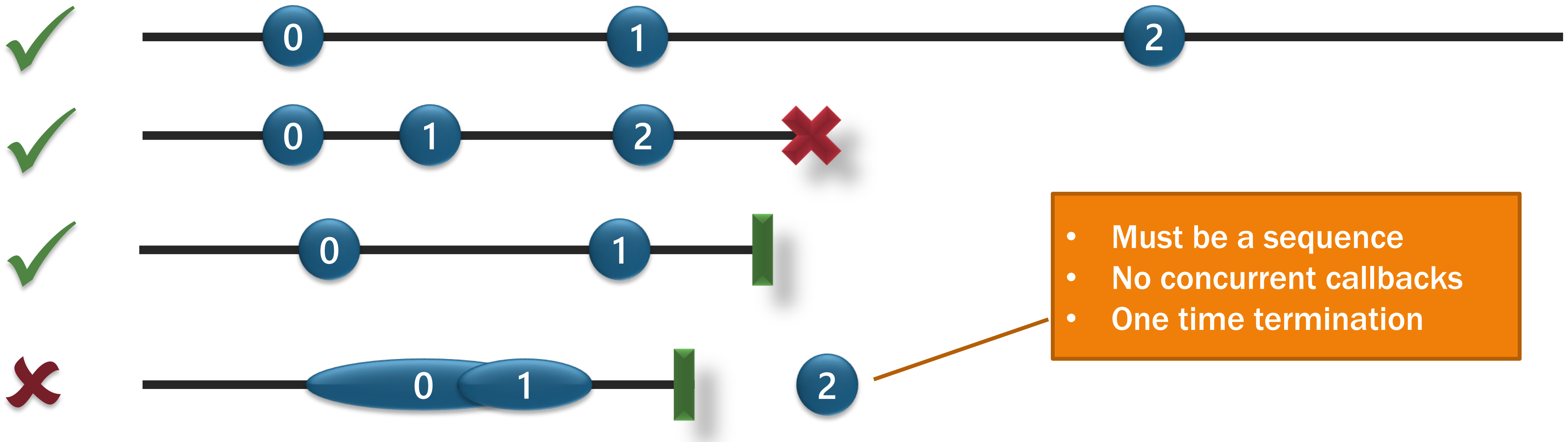
Zero or more values

E.g. events are ∞ sequences

Calls can fail

Resource management

Sequencing



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

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

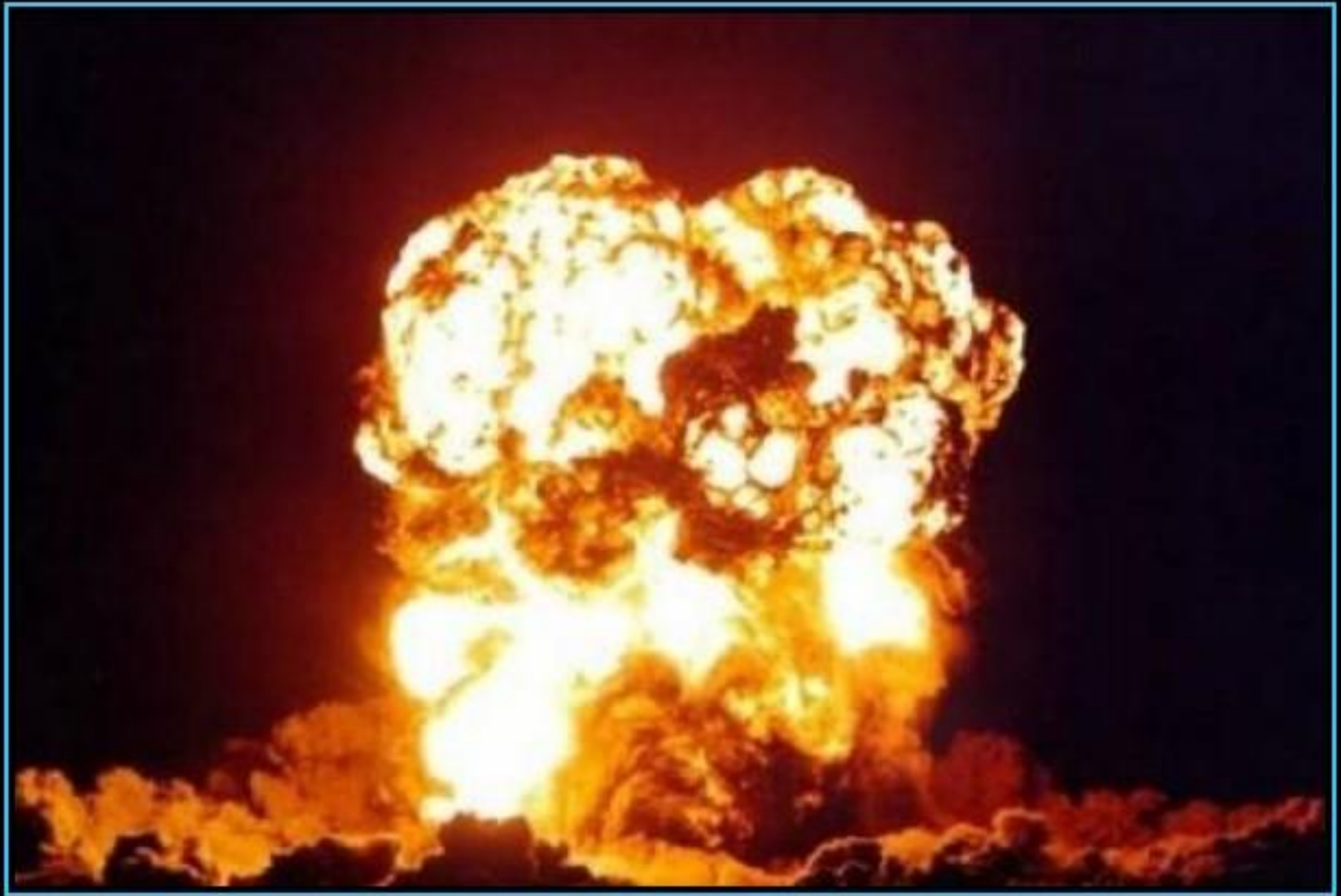
Observables - Querying UI Events

```
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

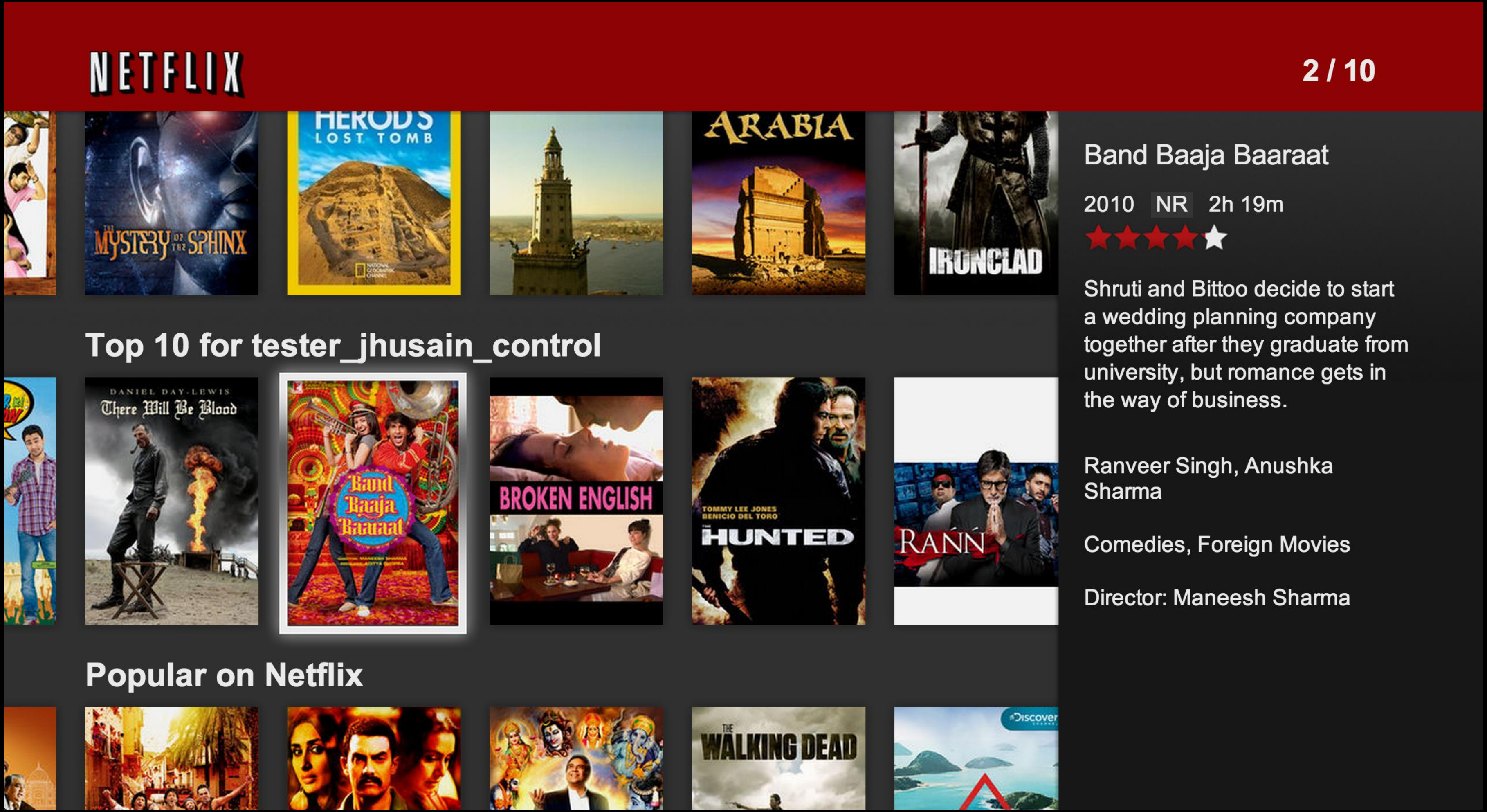


PROTONIC REVERSAL

You crossed the streams, didn't you?

Your Netflix Video Lists

Netflix Row Update Polling



Top 10 for tester_jhusain_control

Popular on Netflix

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

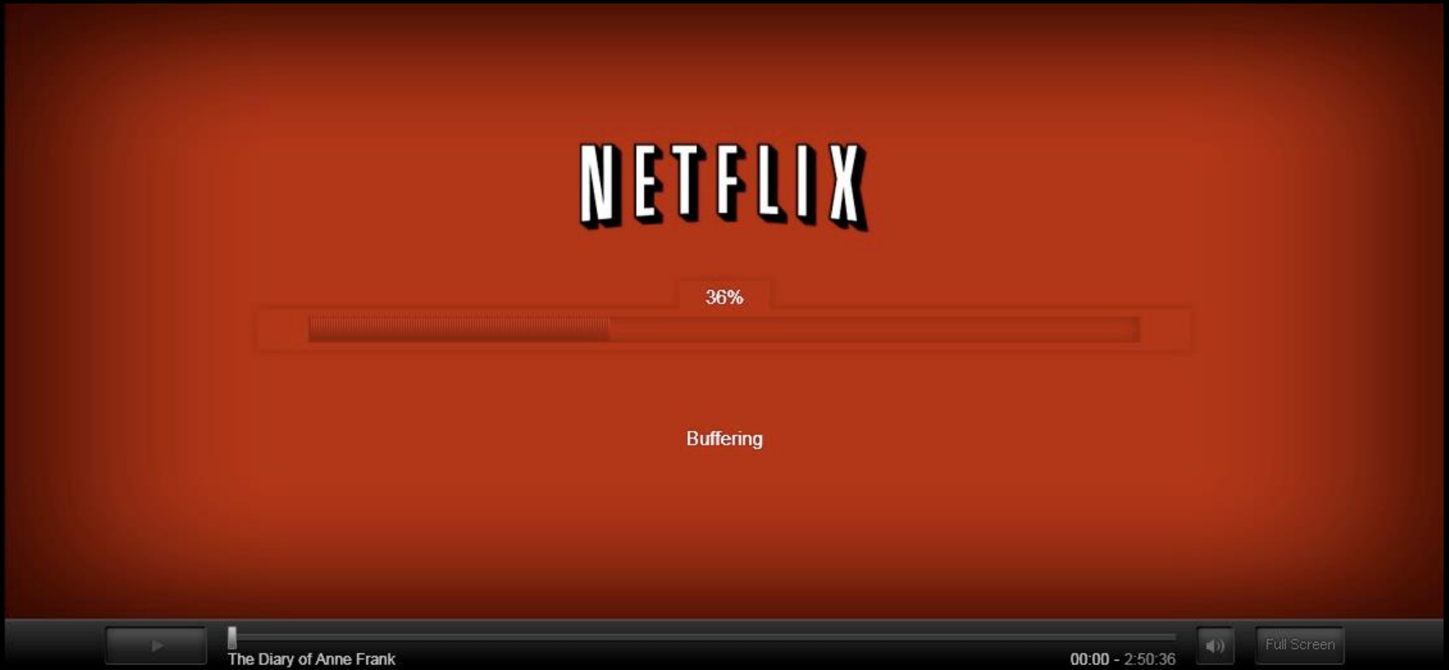
Comedies, Foreign Movies

Director: Maneesh Sharma

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.createColdObservable(
    onNext(300, "BuildStuff"),
    onNext(400, "2014"),
    onCompleted(500));

var results = scheduler.startWithCreate(function () {
    input.map(function (x) { return x.length; })
});

results.messages.assertEqual(
    onNext(300, 10),
    onNext(400, 4),
    onCompleted(500));
```

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;
}
```

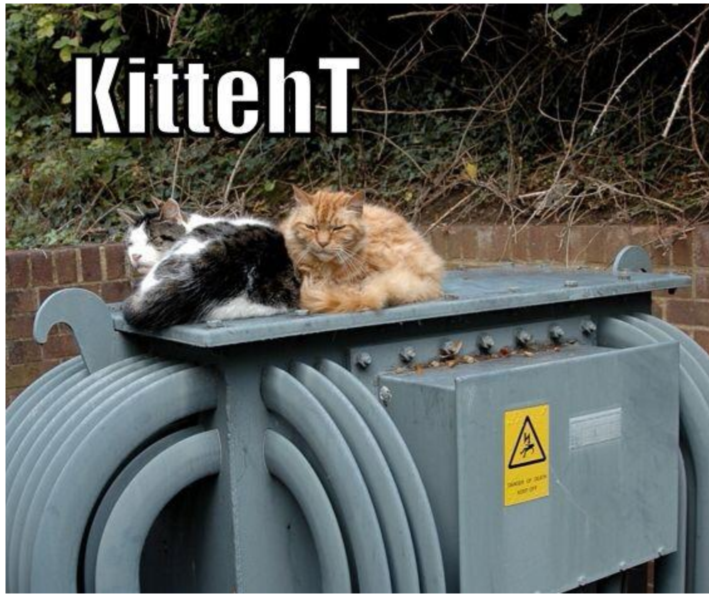
https://github.com/lukehoban/ecmascript-asyncawait

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://buildstuff.lt')
    .retry(3)
    .catch(cachedVersion);

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



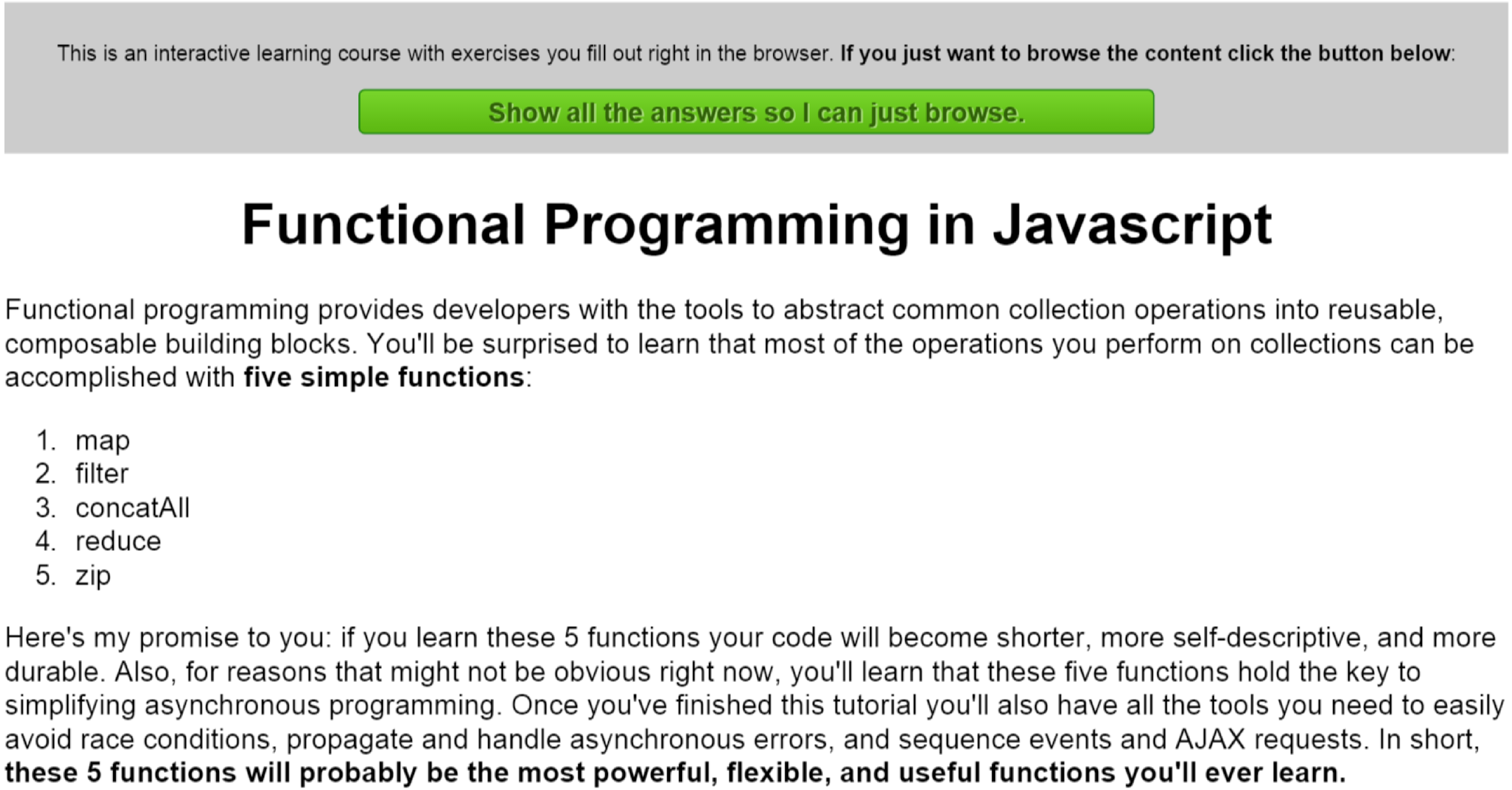
Async Generators

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**:

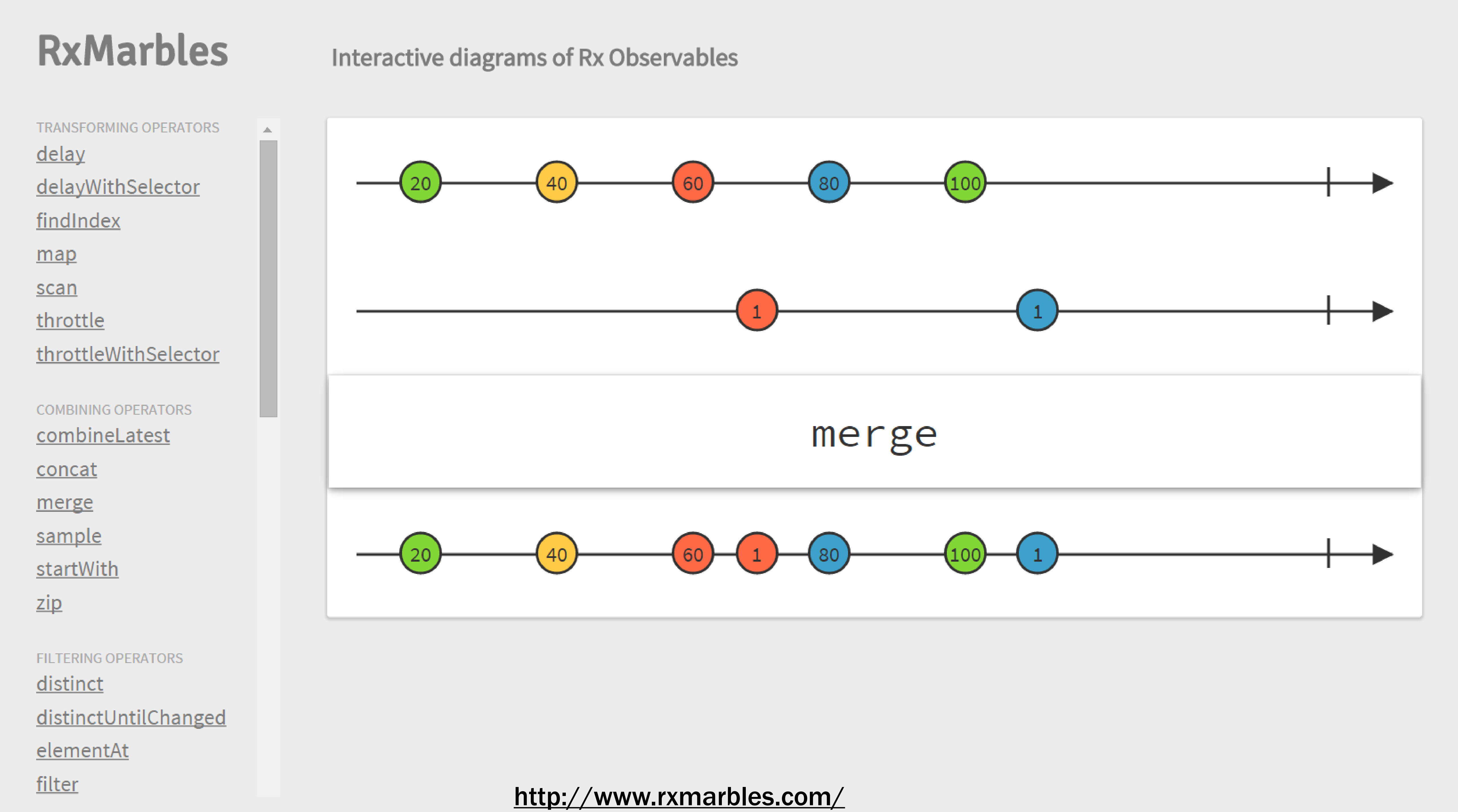
Functional Programming in Javascript

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/