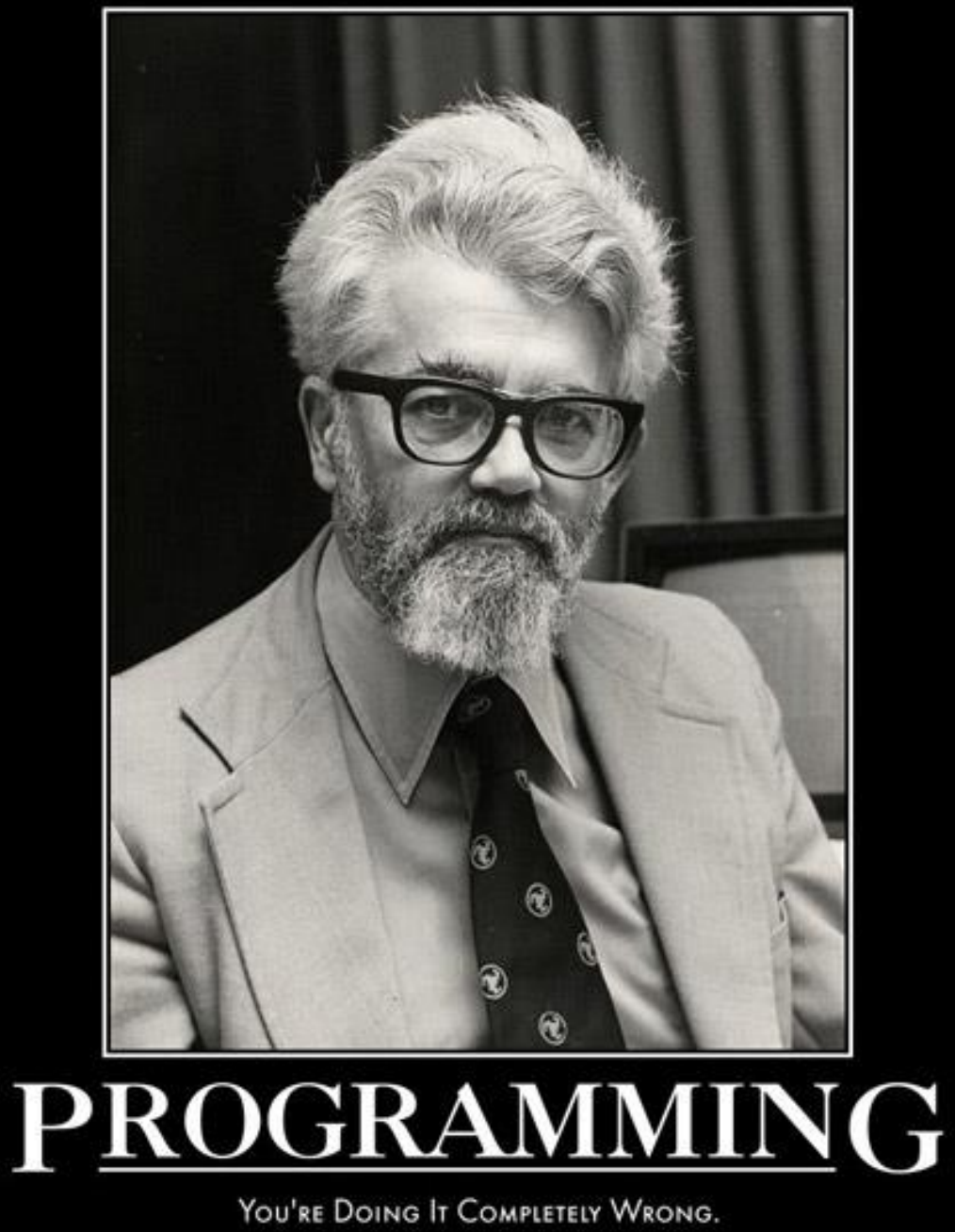


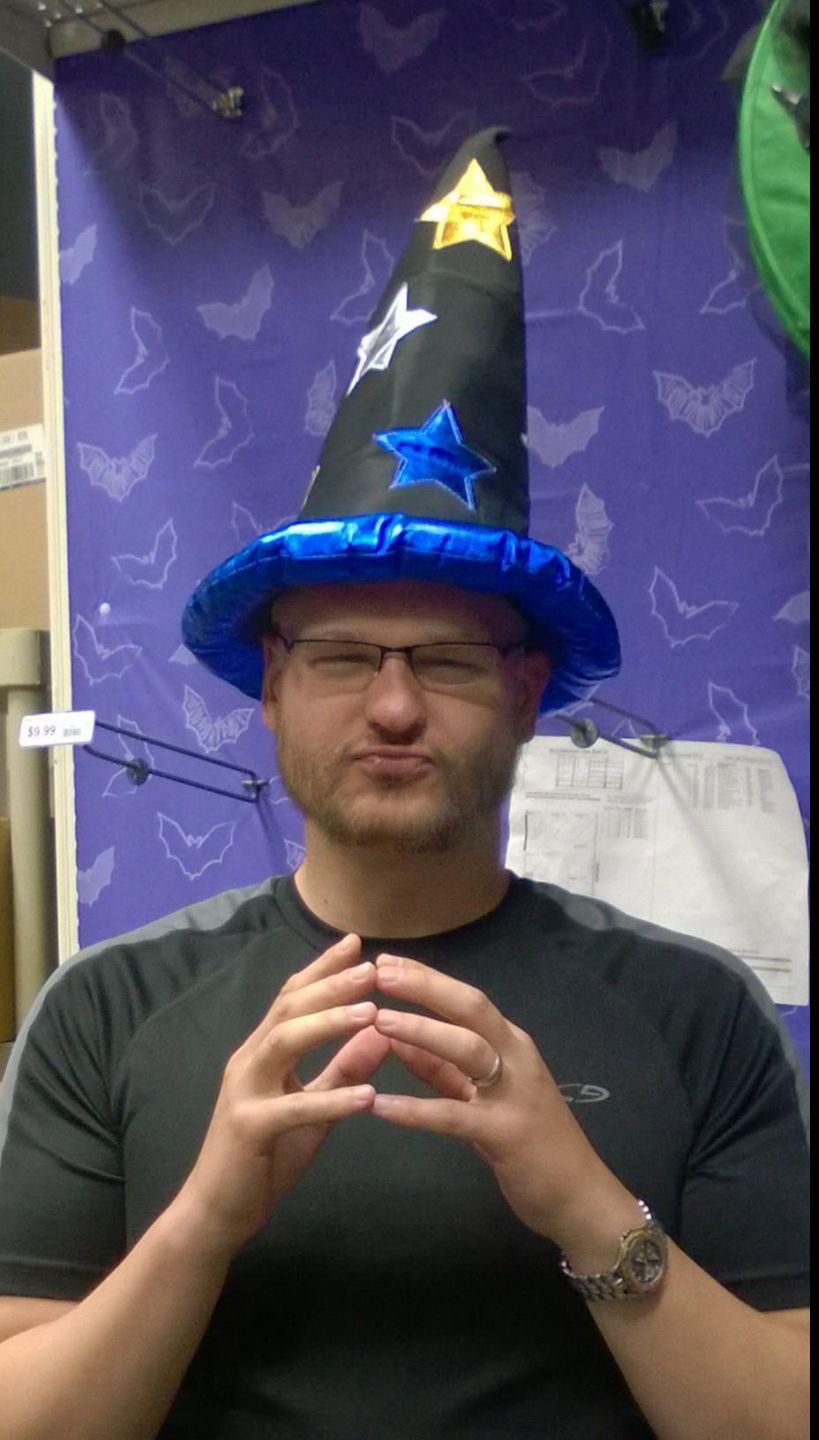
Promises: The Billion Dollar Mistake

Matthew Podwysocki @mattpodwysocki
github.com/mattpodwysocki/buildstuff2014

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



Software Engineer
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki

MICRÖSÖFT

Reactive Extensions (Rx)

@ReactiveX

http://reactivex.io

Null References: The Billion Dollar Mistake

Presented by Tony Hoare on Aug 25, 2009

Community Architecture Topics Language Design Tags QCon , QCon London 2009 Lengt

Share | dz

Summary

Tony Hoare introduced Null r
1965 "simply because it was
Mr. Hoare. He talks about tha
billion-dollar mistake".

Bio

Sir Charles Antony Richard H
Hoare, is a British computer
for the development in 1960
also developed Hoare logic,
Communicating Sequential P
the Occam programming lan

About the conference

http://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

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

DOJO Did it First ™

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 Evolution of Promises…

Promises/A – A Community Standard for Promises

CommonJS

```
.then(fullfilledFn, errorFn, progressFn);
```

What?

- One of three states: unfulfilled, fulfilled, failed
- Always asynchronous with optional progress handler
- Once settled, value is immutable
- .then returns a new Promise

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

http://wiki.commonjs.org/wiki/Promises/A

jQuery

write less, do more.

https://thewayofcode.wordpress.com/2013/01/22/javascript-promises-and-why-jquery-implementation-is-broken/

Promises/A+

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

then

What?

- Removed progress and construction rules from Promises/A
- Three states: pending, fulfilled, rejected
- onFulfilled and onRejected handlers purely optional

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

https://promisesaplus.com/

DOM Promises

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

then

What?

- Put forth via the WHATWG
- Built into the browser runtime
- Handle XHR, Geolocation, IndexedDB, onload

```
var xhr = new XMLHttpRequest();
xhr.open('GET', filename, true);
xhr.send().then(handleSuccess, handleError);
```

https://github.com/slightlyoff/Promises

Incorporate monads and category theory #94

Closed **paulmillr** opened this issue on Apr 10, 2013 · 207 comments

paulmillr commented on Apr 10, 2013

Brian Mckenna criticised current spec. He proposes to use FP approach to achieve much better modularity.

Suggest to read it, really good ideas with just three changes.

http://brianmckenna.org/blog/category_theory_promisesaplus

His proposal is to incorporate into spec three simple apis:

1. `Promise.of(a)` will turn anything into promise.
2. `Promise#then(f)` should take one function, not two.
3. `Promise#onRejected(f)` : move `onRejected` to prototype instead of second arg.

edit: see promises-aplus/constructor-spec#24 for more discussion

https://github.com/promises-aplus/promises-spec/issues/94


trapd in IO monad
plz help

ES6 Promises

then

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

What?

- Approved by TC39 September 2013
- Chrome 32, Opera 19, Firefox 29 & IE Preview
- Defines constructor with resolver with resolve or reject

```
var p = new Promise(function (res, rej) {
  res(42);
});

p.then(function (v) { console.log(v); }
```

https://github.com/domenic/promises-unwrapping

Promises, Promises...

then

Problems with ES6 Promises

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

```
var promise;

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

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

Some things are easily solved...

then

```
var last = require('last');

var smartSearch = last(doSearch);

input.addEventListener('keyup', _.debounce(function (e) {
    smartSearch(e.target.value)
      .then(updateUIWithResults);
  }, 500)
}, false);
```

https://github.com/domenic/last

Others, not so much...

promises-aplus / **cancellation-spec**

Watch

Background #1

Open **ForbesLindesay** opened this issue on Dec 16, 2012 · 38 comments

ForbesLindesay commented on Dec 16, 2012 Collaborator

Cancellation has not yet been put into any of the major JavaScript promise libraries as far as I'm aware.

The only promise library to have any cancellation support is when.js. There has however also been some prior art in the form of C# cancellation tokens. There are a few things to decide upon.

Triggering Cancellation

We need to decide how cancellation is triggered. This is probably as simple as a cancel method on the returned promise. The only problem with doing that is you can't then give a promise (as the result of a function) to multiple mutually suspicious receivers. C# does it with a separate `CancellationTokenSource` object. The relation between `CancellationToken` and `CancellationTokenSource` is analogous to the relationship between promise and resolver.

Handling Cancellation

https://github.com/promises-aplus/cancellation-spec

then

What Can We Learn From Others?

then

java.util.concurrent.Future<V>

- Has .cancel and .isCanceled()
- Does not support chaning like “then”

```
someJavaTask.cancel();
```

System.Threading.Tasks.Task<T>

- Supports cancellation via CancellationToken
- Supports chaining via ContinueWith

```
someTask.ContinueWith(t => DoSomethingElse(), token);
```


streem prosesing

Streaming is Everywhere…

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

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

Why Not Observables?

Creating an Observable

```
var someObservable = Rx.Observable.create(
  function (observer) {

    var task = getData(url, function (d) {
      observer.onNext(d);
      observer.onCompleted();
    });

    return function () {
      if (!task.isDone()) task.abort();
    };
  });
```

Why Not Observables?

Subscribing to an Observable

```
var subscription = someObservable.subscribe(
  function (value) {
    // Do something with the value
  },
  function (error) {
    // Handle the error
  },
  function () {
    // Handle completion
  });

subscription.dispose();  // Deterministic disposal!
```

Autocomplete with Observables

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

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

DOM events as a sequence of strings

Reducing data traffic / volume

Latest response as word arrays

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

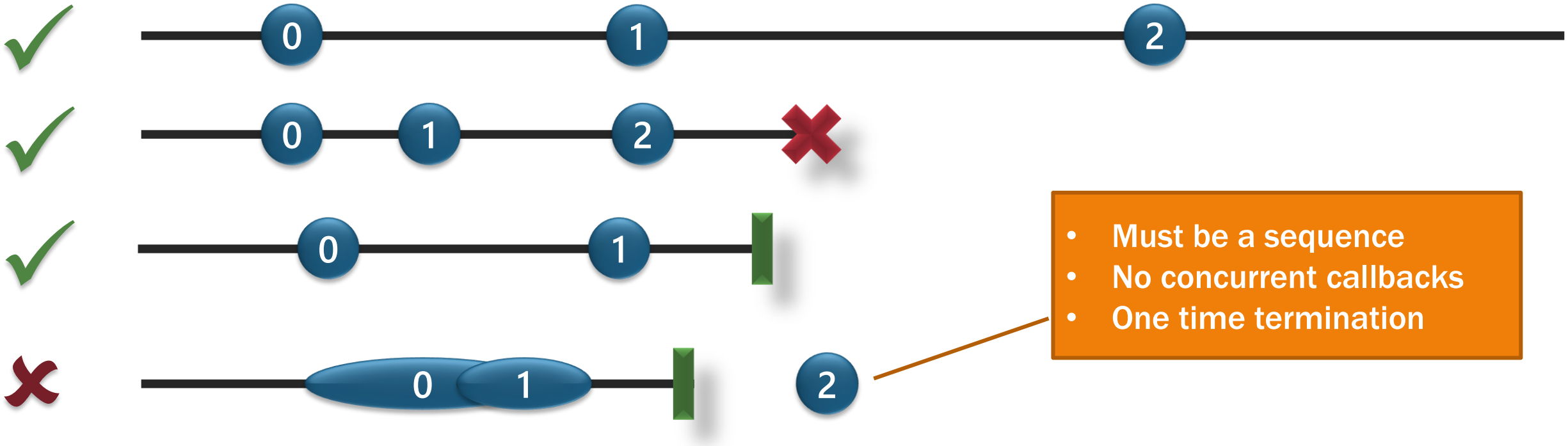
E.g. events are ∞ sequences

(onError | onCompleted) ?

Calls can fail

Resource management

Sequencing



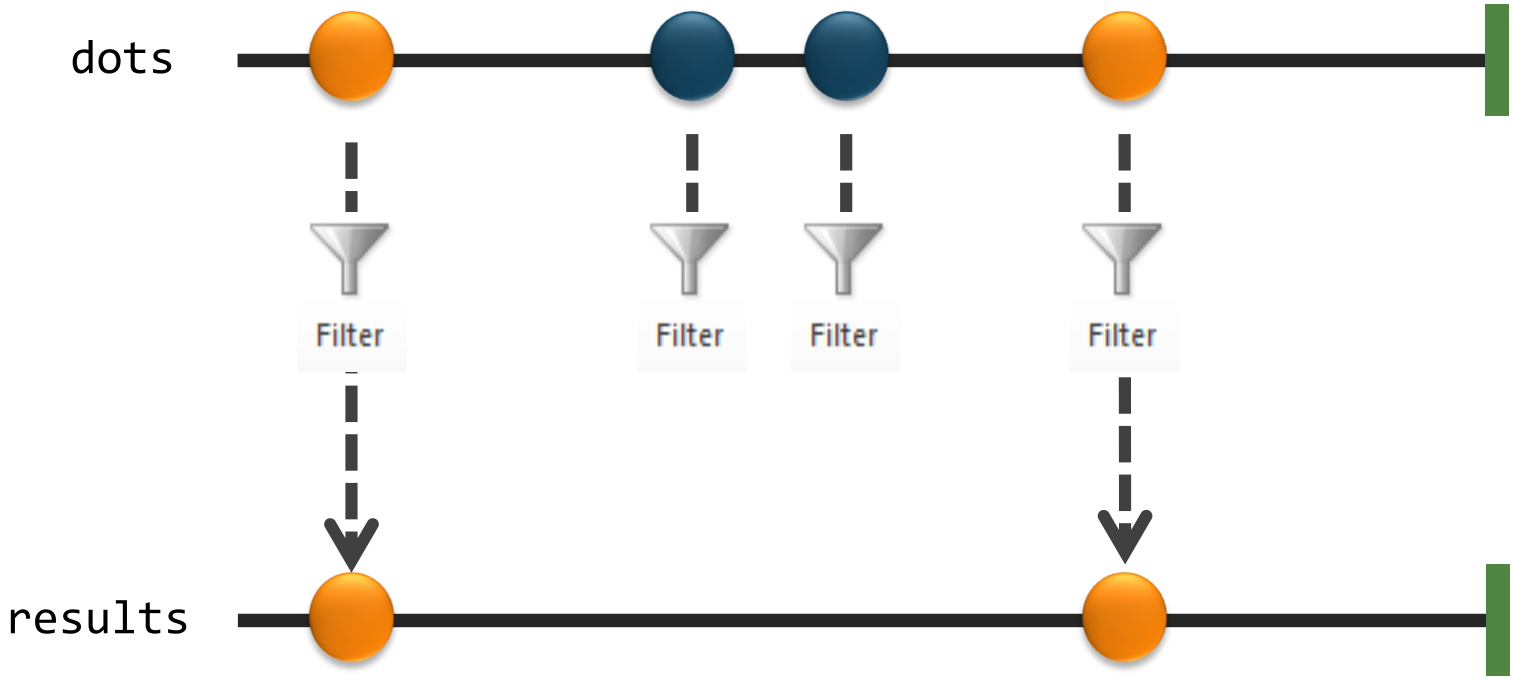
- Must be a sequence
- No concurrent callbacks
- One time termination

What exactly is Rx?

Language neutral model with 3 concepts:

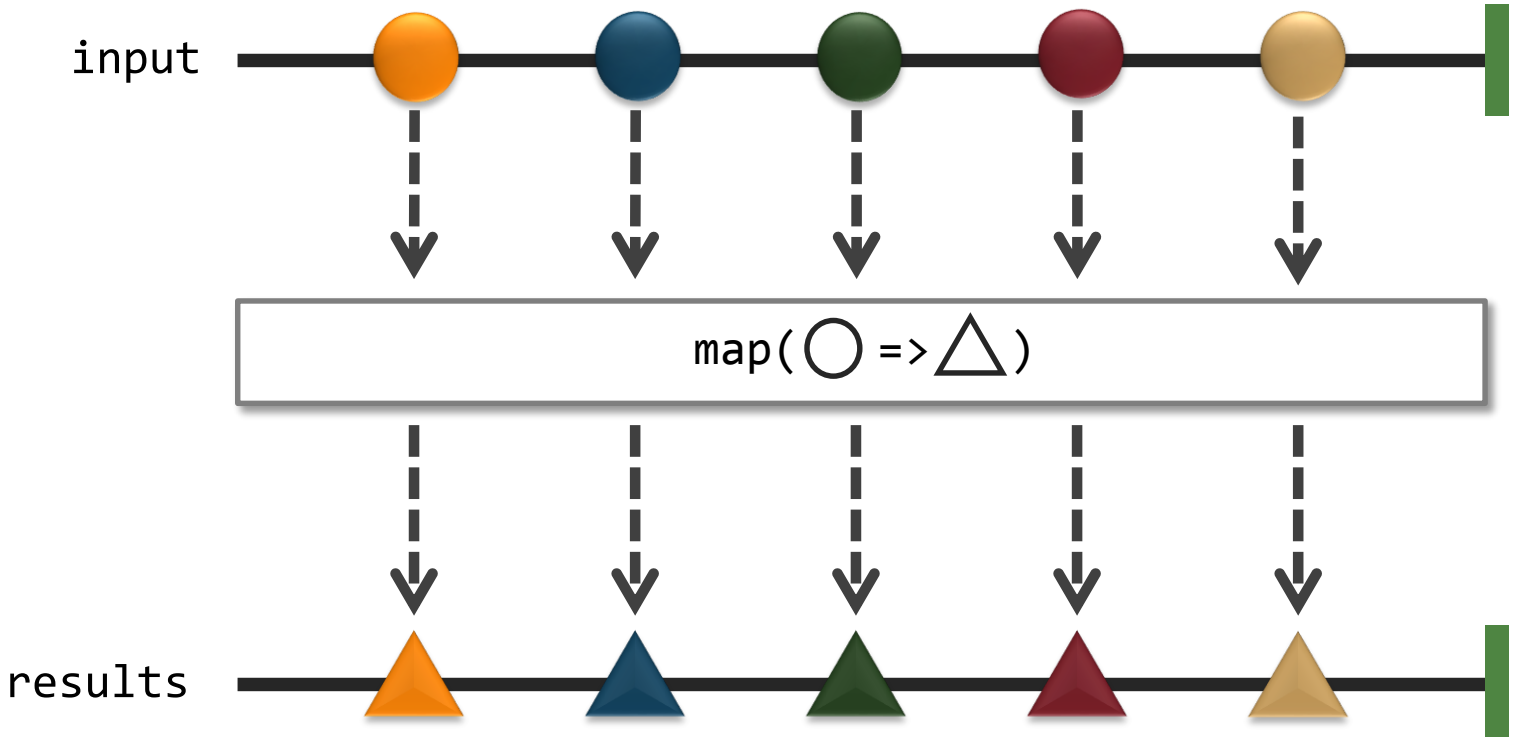
1. Observer/Observable
2. Query operations (map/filter/reduce)
3. How/Where/When
   - Schedulers: a set of types to parameterize concurrency

Marble diagram: filter




```
.filter(function (dot) {
  return dot.isOrange();
})
```

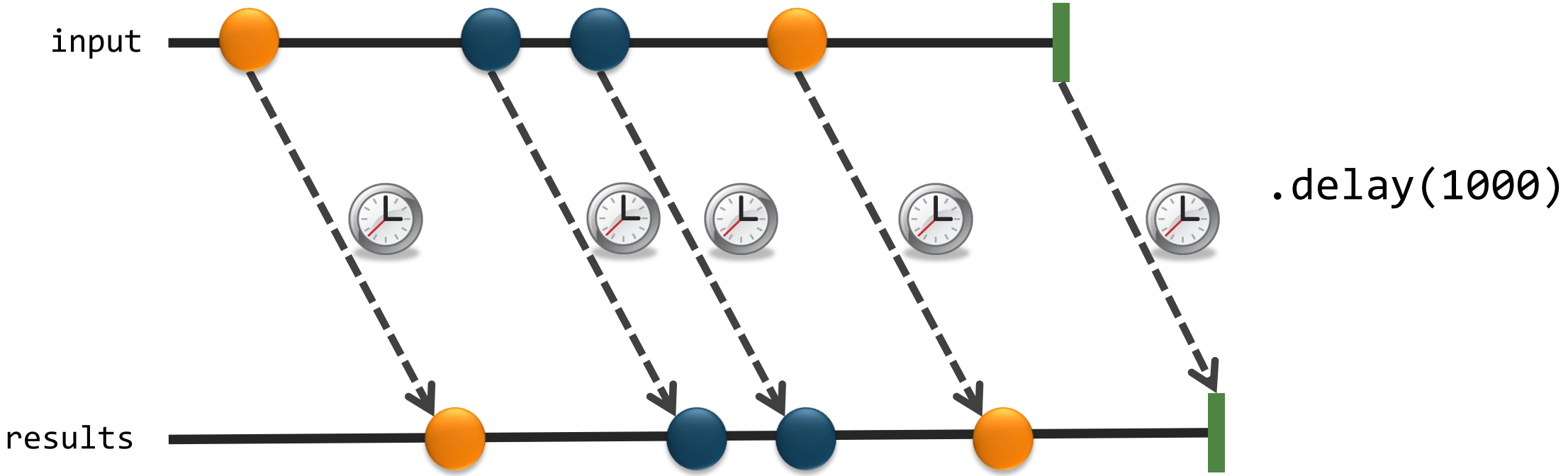
Marble diagram: map




```
.map(function (i) {
  return transform(i);
})
```

Marble diagram: delay

Since Observables are asynchronous, they have a notion of time



RxMarbles

Interactive diagrams of Rx Observables

TRANSFORMING OPERATORS

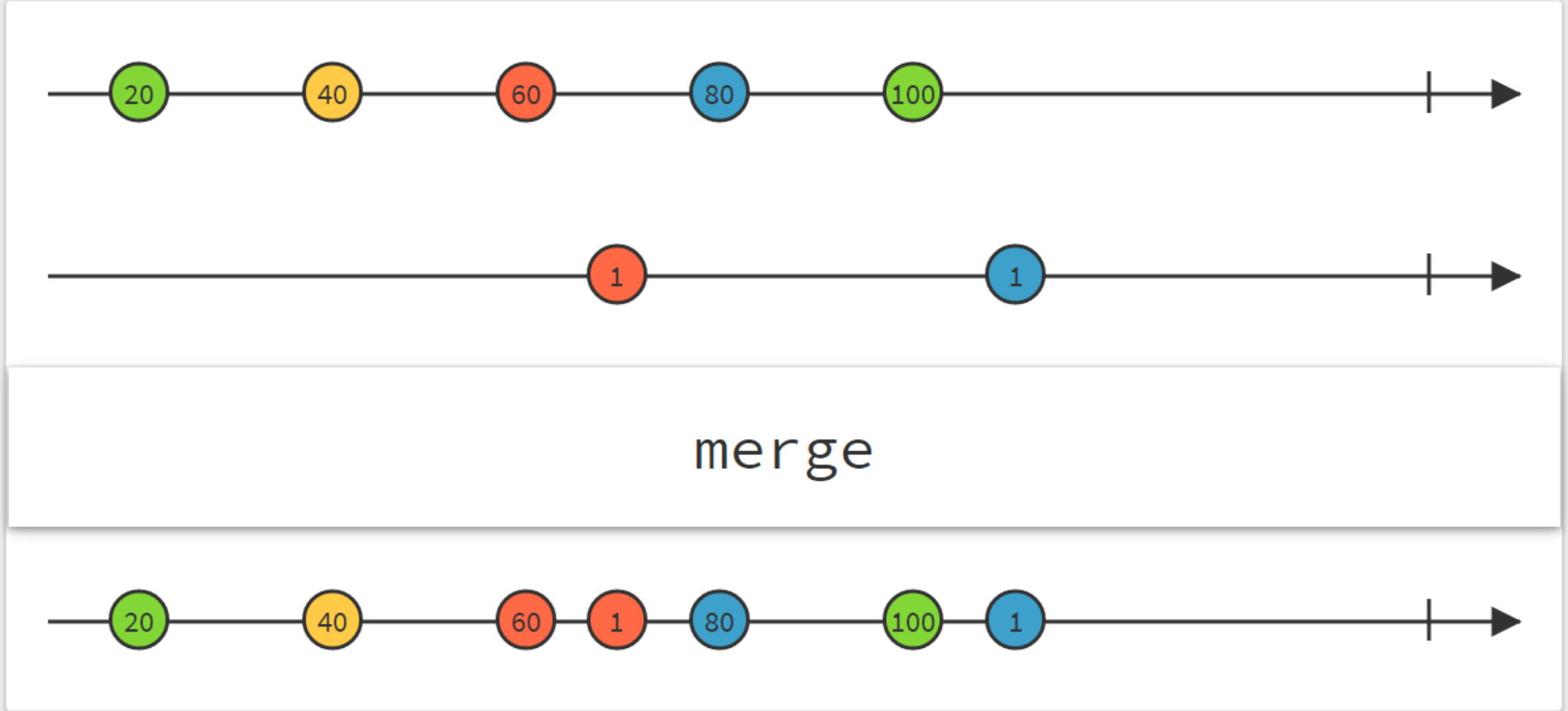
- delay
- delayWithSelector
- findIndex
- map
- scan
- throttle
- throttleWithSelector

COMBINING OPERATORS

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

FILTERING OPERATORS

- distinct
- distinctUntilChanged
- elementAt
- filter



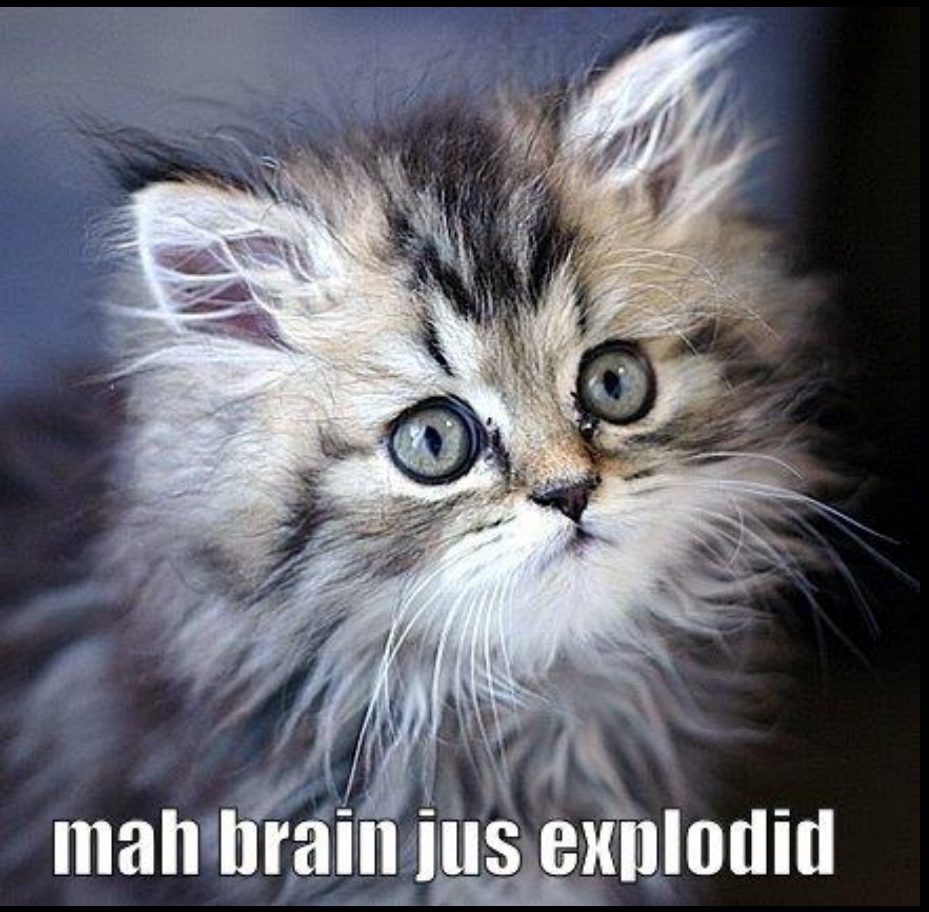
http://www.rxmarbles.com/

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)
    .flatMap(function (md) {
      return dom.mousemove(document)
        .filter  .takeUntil(dom.mouseup(elmt));
    });
};

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


Everything is a stream

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);
  });
```

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);
```

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

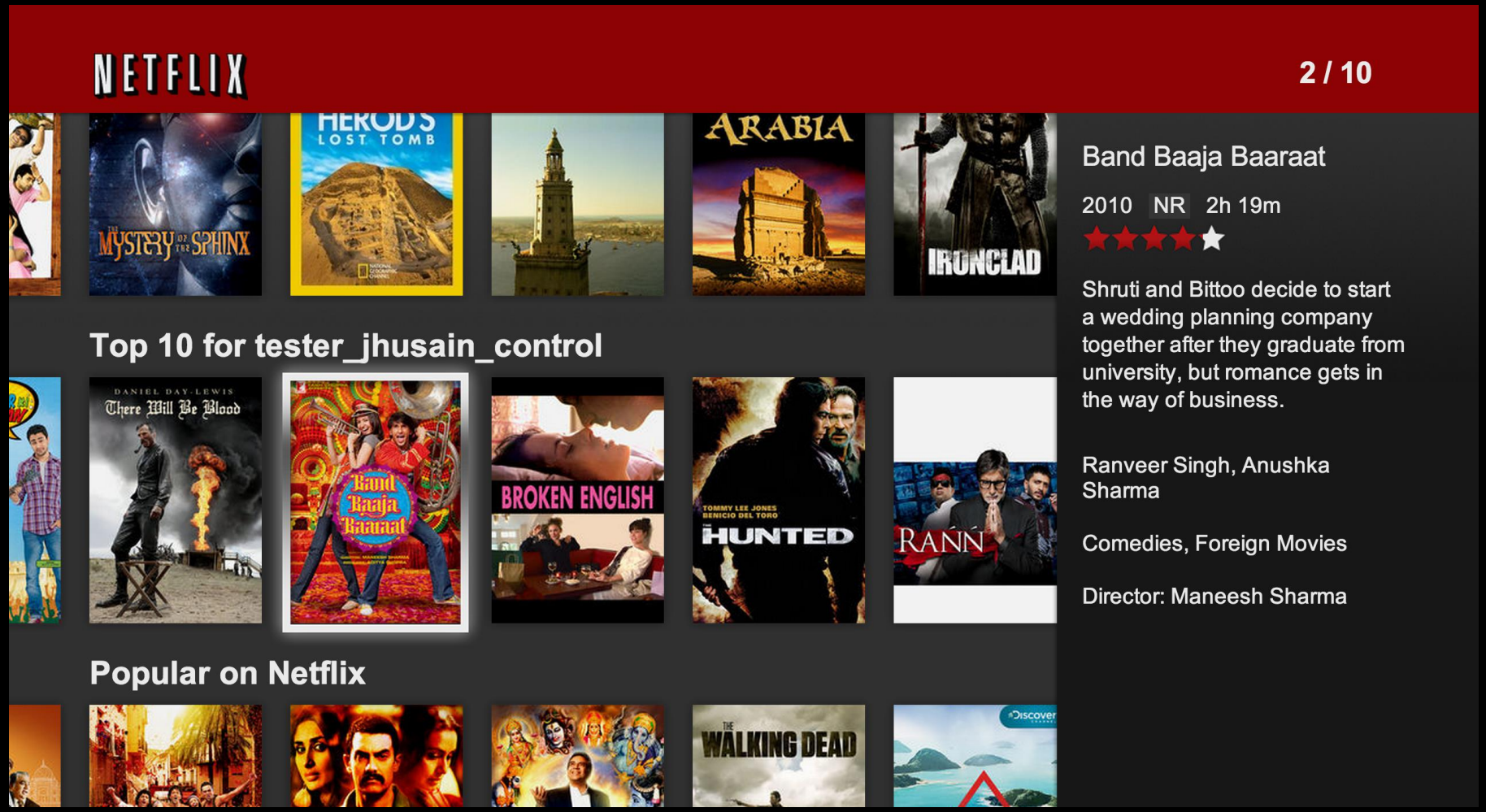
3 until mouse up

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();
};
```

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

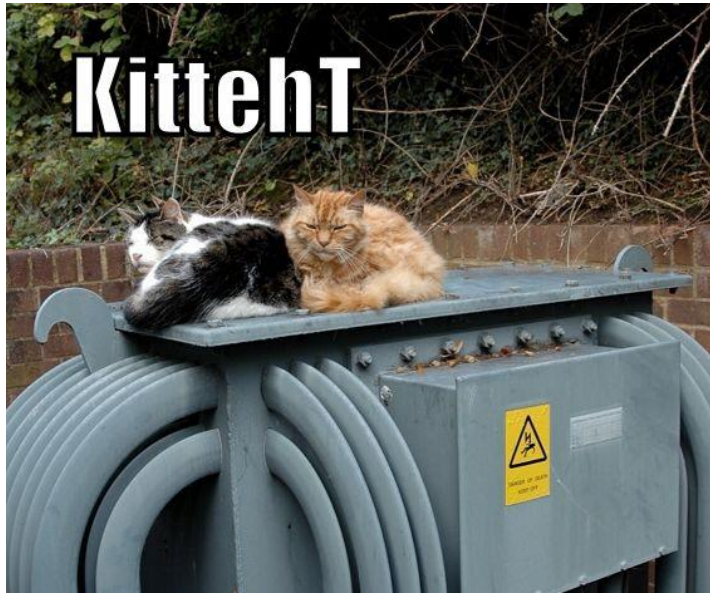
Async/Await with Observables and Generators…

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