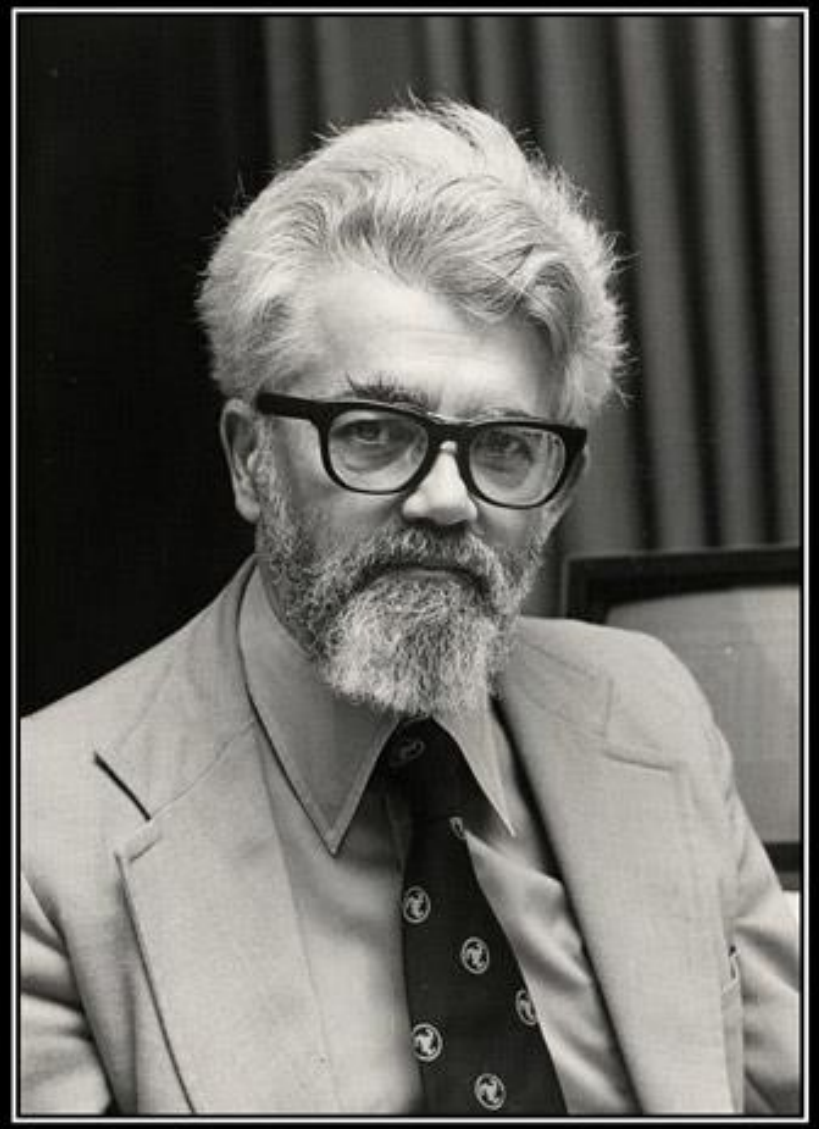


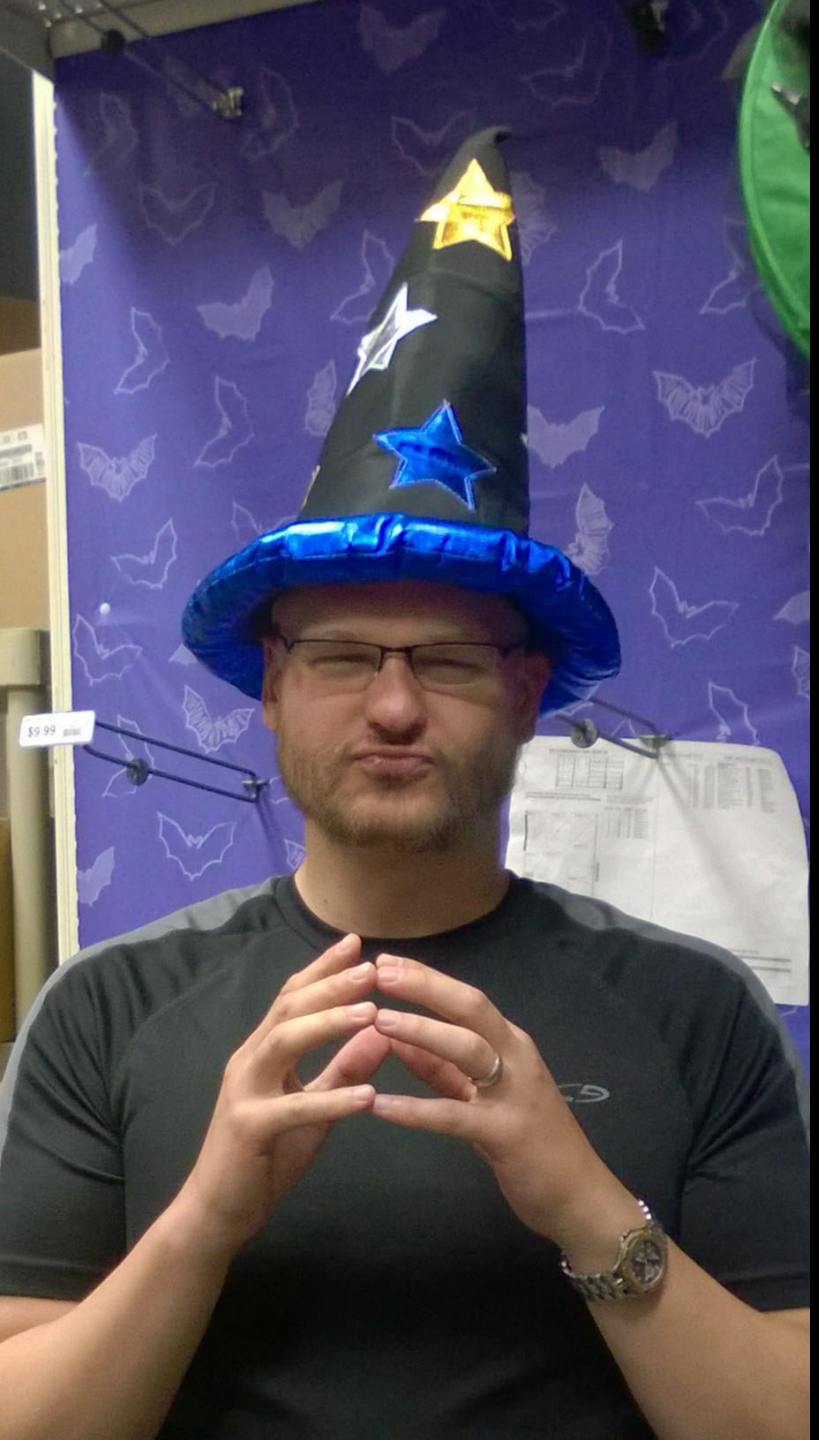
Async and Streaming JS
Are We doing it wrong?

Matthew Podwysocki @mattpodwysocki

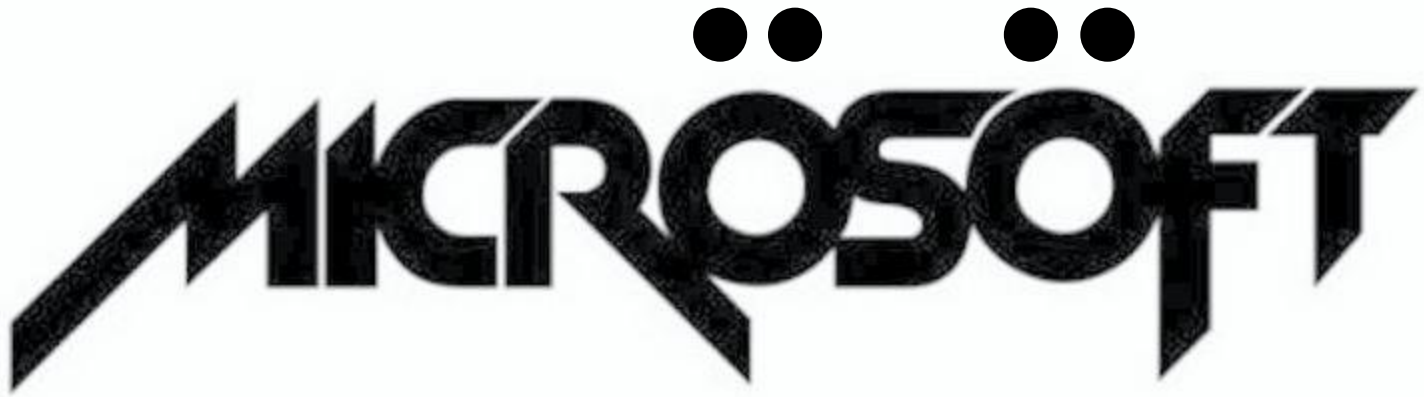
github.com/mattpodwysocki/jsconfeu-2014

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


trapd in Monad tutorl
plz help



Software Engineer
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki


MICROSOFT

Rx

@ReactiveX

http://reactivex.io

“To be conscious that you are ignorant is a great step to knowledge.”

“Ignorance never settles a question.”

“The best way to become acquainted with a subject is to write a book about it.”

Benjamin Disraeli (1804-1881)

Asynchronous Programming is Annoying

Every library has a different way of doing async/events

- Callbacks/Promises are different from events
- Each concept covers only part of the story

Wouldn't it be great to have some concepts to generalize how we think about concurrent/reactive programming?

“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();
    });
});
```

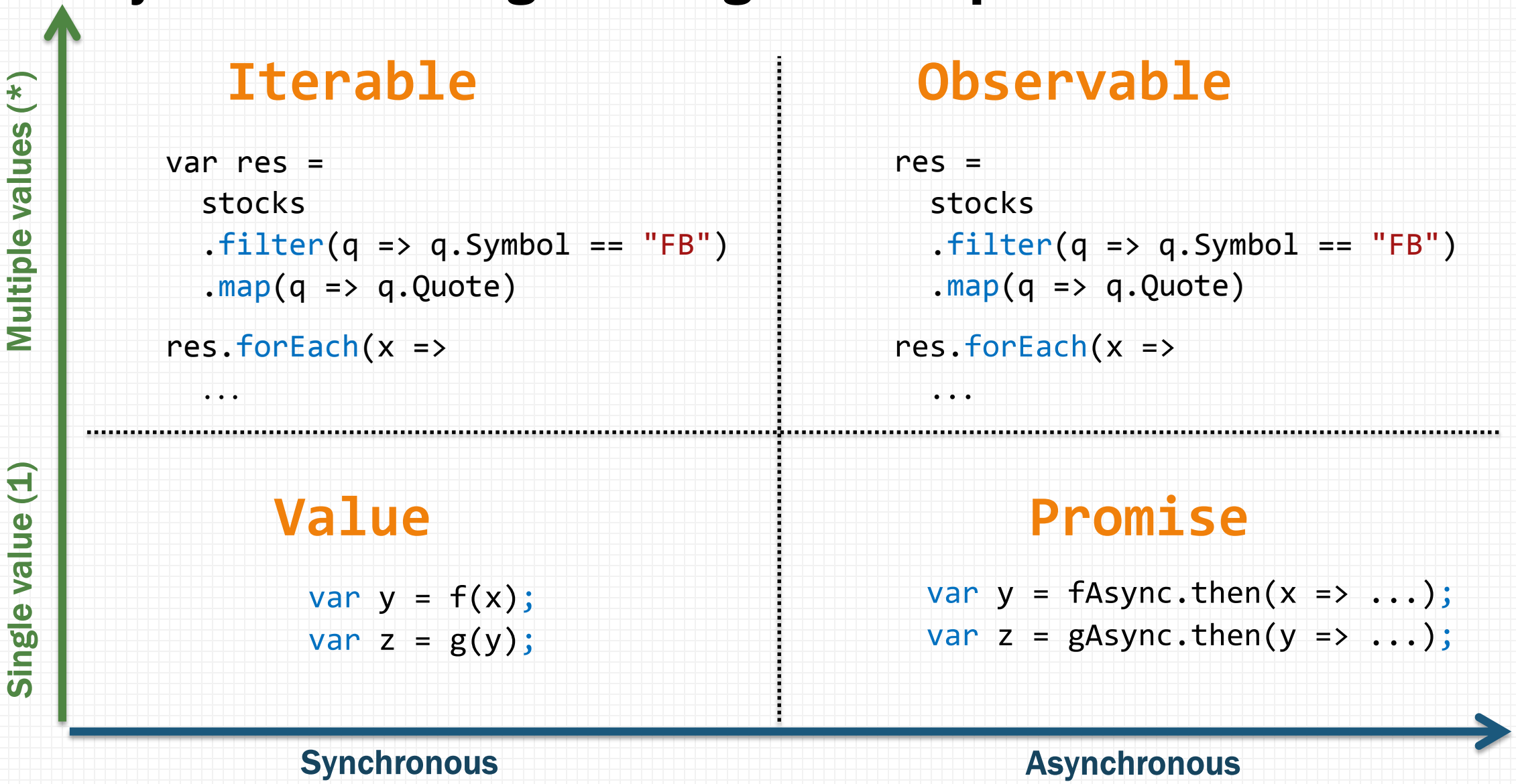

culturepub.fr
next
ad

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);
```

The Asynchronous Programming Landscape



First-Class Async and Events

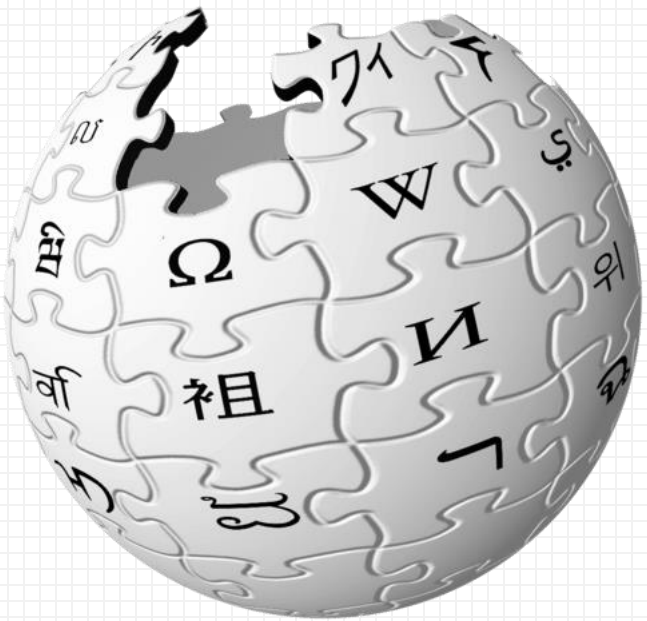
Objects to the rescue

How about a query library?

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)

Or mocking for testing...?



WIKIPEDIA
The Free Encyclopedia

Promises Promises

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

then

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);
```

Reactive Programming

http://www.reactivemanifesto.org

Merriam-Webster defines reactive as *"readily responsive to a stimulus"*, i.e. its components are "active" and always ready to receive events. This definition captures the essence of reactive applications, focusing on systems that:

react to events

the event-driven nature enables the following qualities

react to load

focus on scalability by avoiding contention on shared resources

react to failure

build resilient systems with the ability to recover at all levels

react to users

honor response time guarantees regardless of load

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

Observables - Composing Events and Promises

```
var words = input.keyup
            .map(function() { return input.value; })
            .throttle(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

Observables - Polling for Row Updates

```
function getRowUpdates(row) {
    var scrolls = Rx.DOM.scroll(document);
    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();
};
```


Everything is a stream

Functional Reactive Programming (FRP) is...

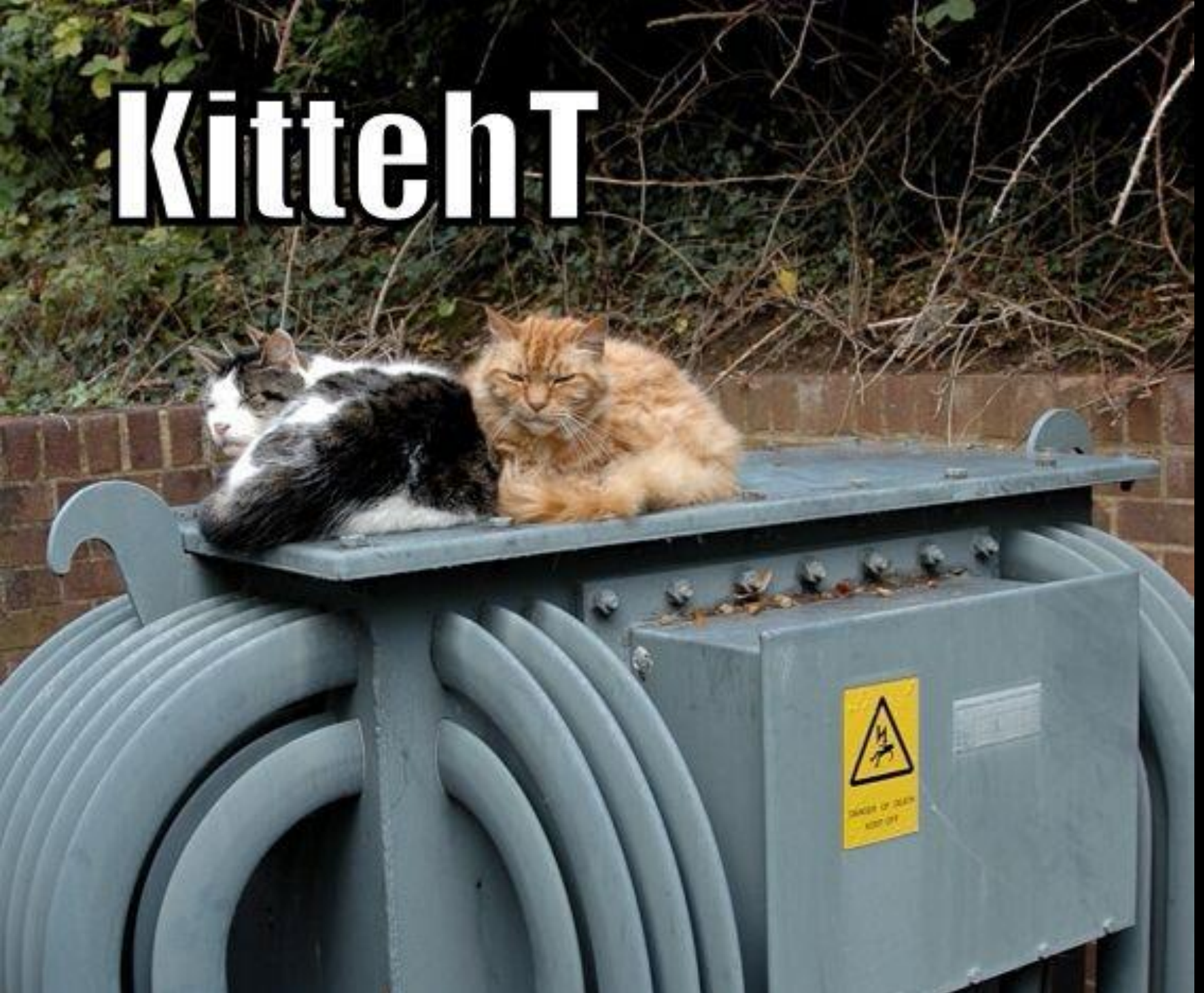
A concept consisting of

- 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


KittehT

Generators

Coming to a runtime near you!

- Yield suspends the execution of the function
- A number of libraries support this like co, Q, and even RxJS

```
var Rx = require('rx');
var request = require('request');

var get = Rx.Observable.fromNodeCallback(request);

Rx.spawn(function* () {
    var a = yield get('http://localhost/stocks1.csv').retry(3);
    console.log(a.length);
});
```

Communicating Sequential Processes (CSP)

Originally from C.A.R Hoare

- Concurrency model for Clojure and Go languages
- Generators make this a much easier win

```
var ch = chan();

go( function* () {
  var val;
  while ((val =
    yield take(ch)) !== csp.CLOSED) {
    console.log(val);
  }
});
```

```
go( function*() {
  yield put(ch, 1);
  yield take(timeout(1000));
  yield put(ch, 2);
  ch.close();
});
```

https://github.com/jlongster/js-csp


imperative cat
is not amused


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


streem prosesing

Let's Face it, Streams1 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...

Streams2 Electric Boogaloo

- Landed in 0.9.4
- Now supported “Object Mode”
- Flowing mode versus non-flowing mode
- Introduced the following Streams classes:
  - Readable
  - Writable
  - Duplex
  - Passthrough

Streams 33 1/3

- Landed in 0.11.2
- Adds cork/uncork/_writev

WHATWG Streams

Specification for creating, composing and consuming streams of data

- Currently in draft
- Focused on low-level I/O, not on object mode
- Unicast in nature only
- Follows Node.js style streams but with Promises
  - Readable
  - Writable

https://whatwg.github.io/streams/

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.'));
```

https://api.dartlang.org/apidocs/channels/stable/dartdoc-viewer/dart-async

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)
- Will be built into the subscriptions starting in 2.4

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

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

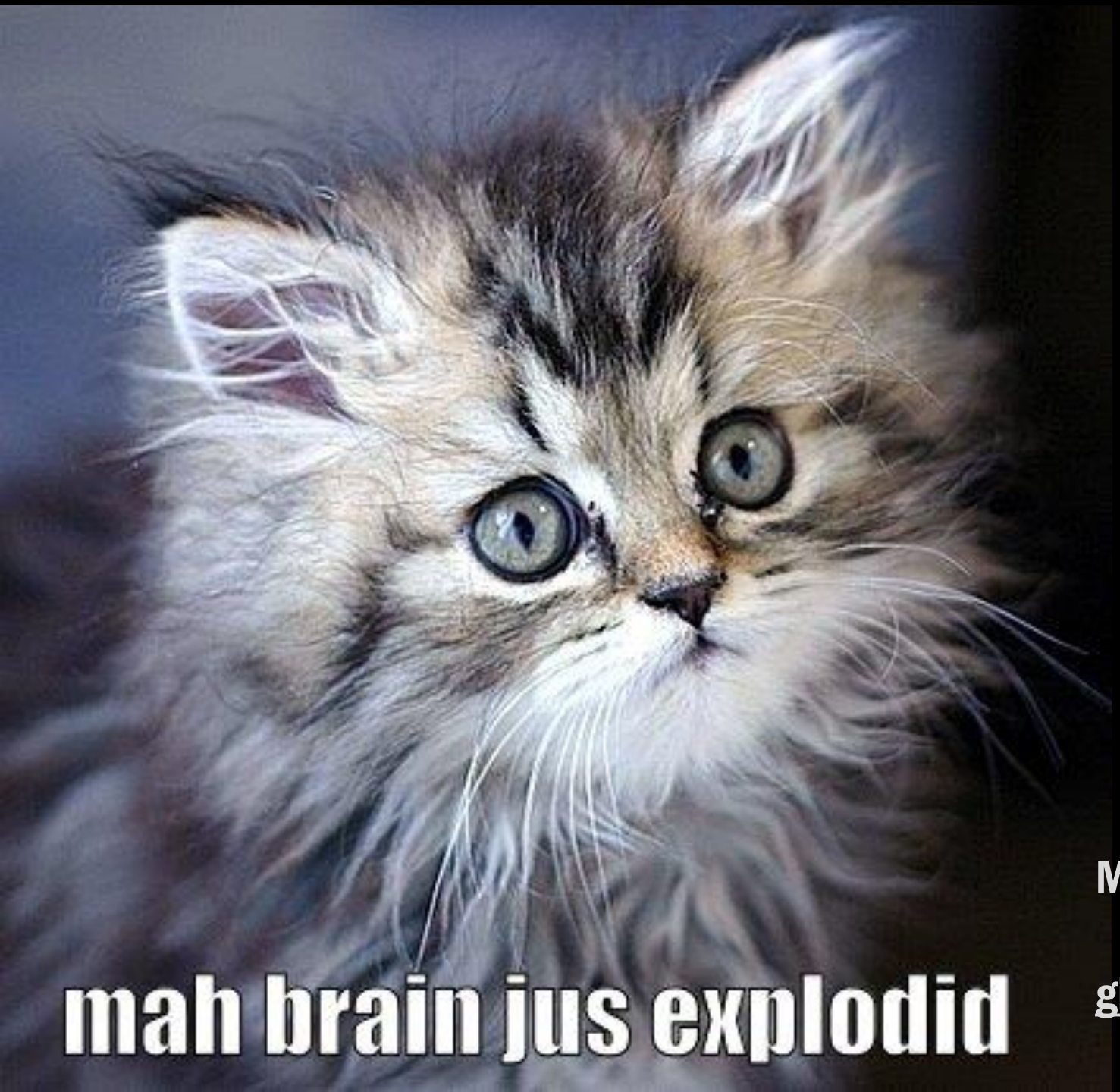
Where do we go from here?

ES7 and Beyond!

- First class events FTW!
- Async Generators!

```
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



Matthew Podwysocki @mattpodwysocki

github.com/mattpodwysocki/jsconfeu-2014