# Going Beyond the Observable

Matthew Podwysocki
@mattpodwysocki

github.com/mattpodwysocki/rxjs-live-asia-21

@mattpodwysocki

@mattpodwysocki
@mpodwysocki

BluerThanBlueFalcon

RxJS live! ASIA

I see you're using Promises. Shall we use Observables instead?

OK

## Azure Notification Hubs

**https://azure.microsoft.com/en-us/services/notification-hubs/**

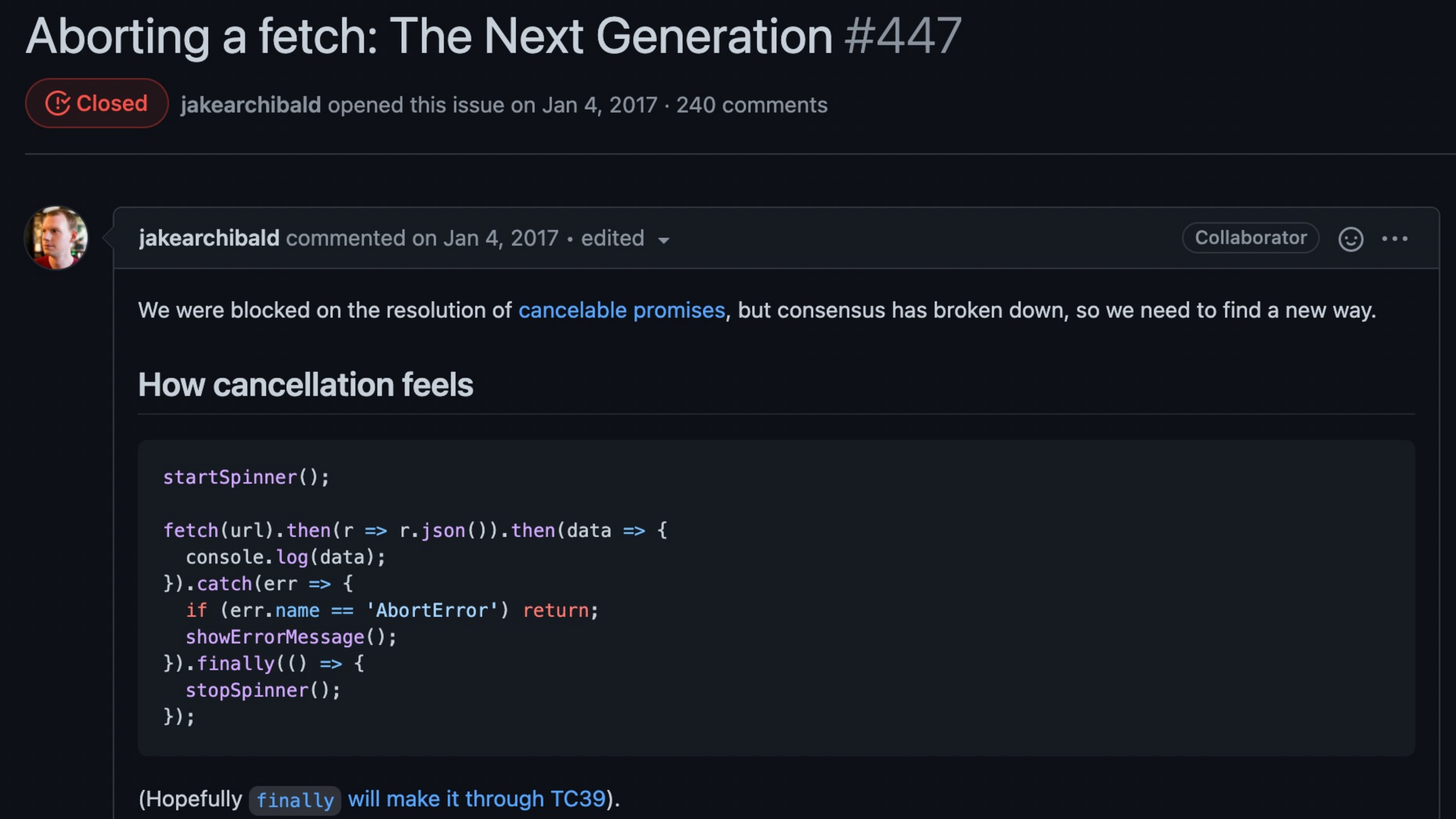
# Async is Awful

## Callbacks are Hell…

```
import { NotificationHubService } from 'azure-sb';

function registerAndSendMessage(token, tags, message, cb) {
  const service = new NotificationHubService(HUB_NAME, CONNECTION_STRING);
  service.apns.createNativeRegistration(token, tags, (err, response) => {
    if (err) {
      cb(err);
      return;
    }

    service.apns.send(tags, message, (error, res) => {
      if (error) {
        cb(error);
        return;
      }

      cb(null, res);
    });
  });
}
```

## Events Don't Compose...

```
let mouseDown = false;
let mouseState = [];

document.addEventListener('mousedown', (e) => {
  mouseDown = true;
});

document.addEventListener('mouseup', (e) => {
  mouseDown = false;
});

document.addEventListener('mousemove', (e) => {
  if (mouseDown) {
    mouseState.push([e.clientX, e.clientY]);
    draw(mouseState);
  } else {
    mouseState = [];
  }
});
```



# Aborting a fetch: The Next Generation #447

Closed jakearchibald opened this issue on Jan 4, 2017 · 240 comments

jakearchibald commented on Jan 4, 2017 • edited Collaborator

We were blocked on the resolution of cancelable promises, but consensus has broken down, so we need to find a new way.

## How cancellation feels

```
startSpinner();

fetch(url).then(r => r.json()).then(data => {
  console.log(data);
}).catch(err => {
  if (err.name == 'AbortError') return;
  showErrorMessage();
}).finally(() => {
  stopSpinner();
});
```

(Hopefully `finally` will make it through TC39).

# Cancelling a Promise…

```js
const controller = new AbortController();
const signal = controller.signal;

startSpinner();

fetch(url, { signal })
  .then(r => r.json())
  .then(response => console.log(response))
  .catch(err => {
    if (err.name === 'AbortError') {
      return;
    }
    showErrorMessage();
  }).finally(() => {
    stopSpinner();
  });
```

```js
const controller = new AbortController();
const signal = controller.signal;

startSpinner();

try {
  const res = await fetch(url, { signal });
  const json = await res.json();
  console.log(json);
} catch (err) {
  if (err.name !== 'AbortError') {
    showErrorMessage();
  }
} finally {
  stopSpinner();
}
```

## Can we do better?

# Let’s Think BIG

# 1973 – Actor Model

```
const delay = (time) => new Promise((res) => setTimeout(res, time));

const ping = spawnStateless(system, async (msg, ctx) =>  {
  console.log(msg.value);
  // ping: Pong is a little slow. So I'm giving myself a little handicap :P
  await delay(500);
  dispatch(msg.sender, { value: ctx.name, sender: ctx.self });
}, 'ping');

const pong = spawnStateless(system, (msg, ctx) =>  {
  console.log(msg.value);
  dispatch(msg.sender, { value: ctx.name, sender: ctx.self });
}, 'pong');

dispatch(ping, { value: 'begin' sender:pong });
```

1978

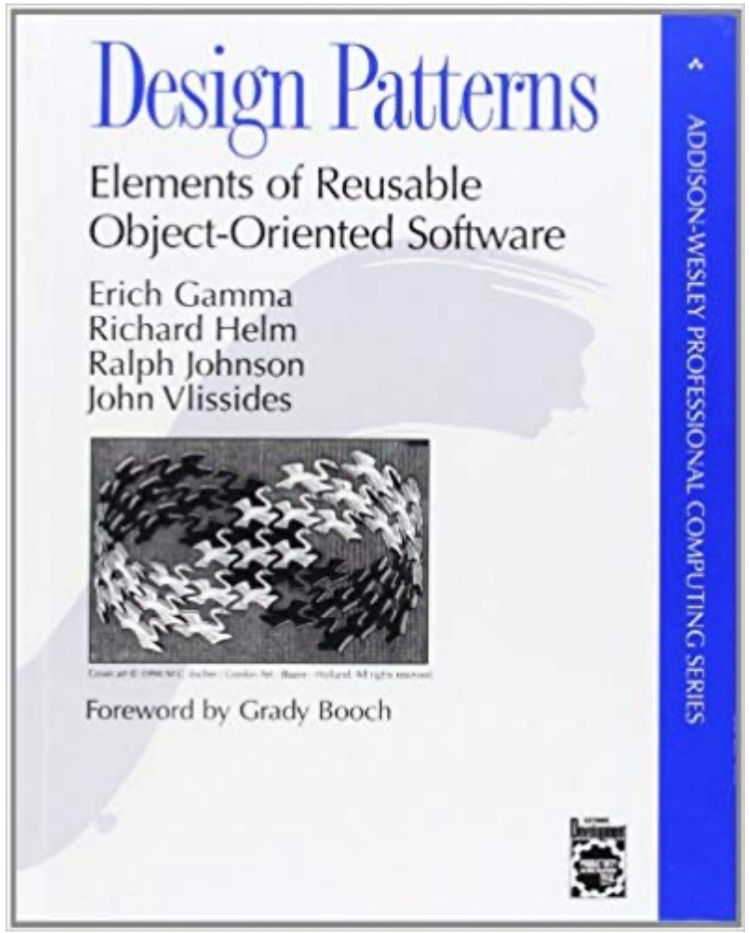
**C**ommunicating

**S**equential

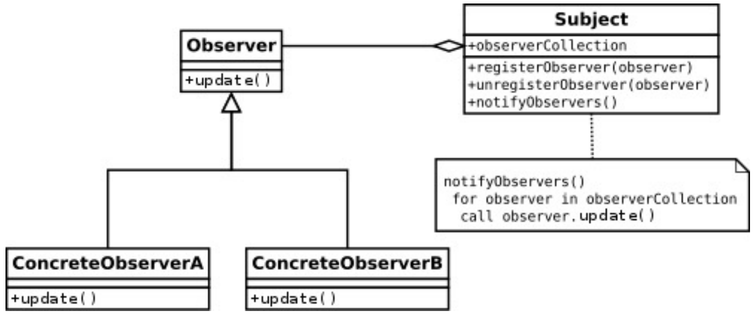
**P**rocesses

```javascript
import {go, chan, take, put} from 'js-csp';

let chA = chan();
let chB = chan();

// Process A
go(function* () {
  const receivedFirst = yield take(chA);
  console.log('A > RECEIVED:', receivedFirst);

  const sending = 'cat';
  console.log('A > SENDING:', sending);
  yield put(chB, sending);
});

// Process B
go(function* () {
  const sendingFirst = 'dog';
  console.log('B > SENDING:', sendingFirst);
  yield put(chA, sendingFirst);

  const received = yield take(chB);
  console.log('B > RECEIVED:', received);
});
```

# 1994 – Gang of Four

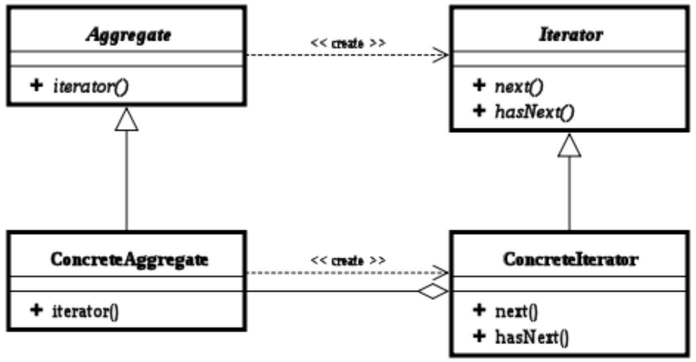

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

**Subject/Observer**



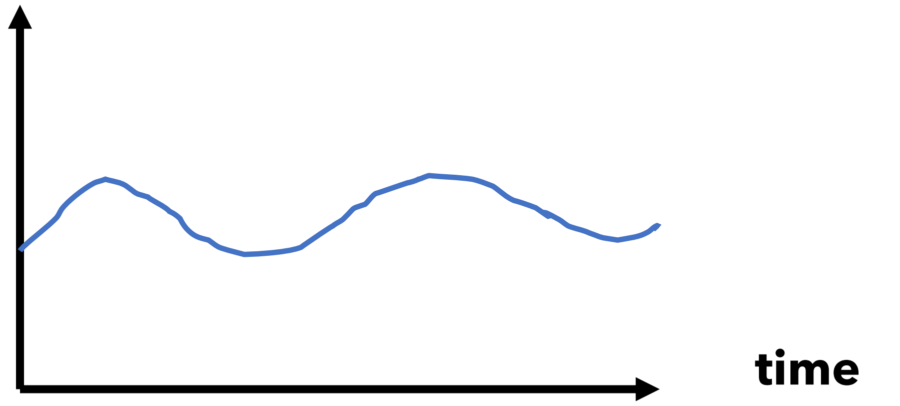
```
notifyObservers()
 for observer in observerCollection
  call observer.update()
```

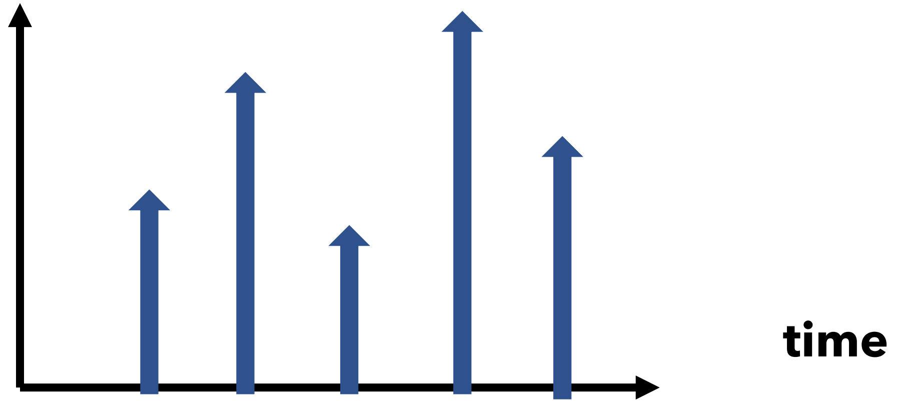
**Iterator**



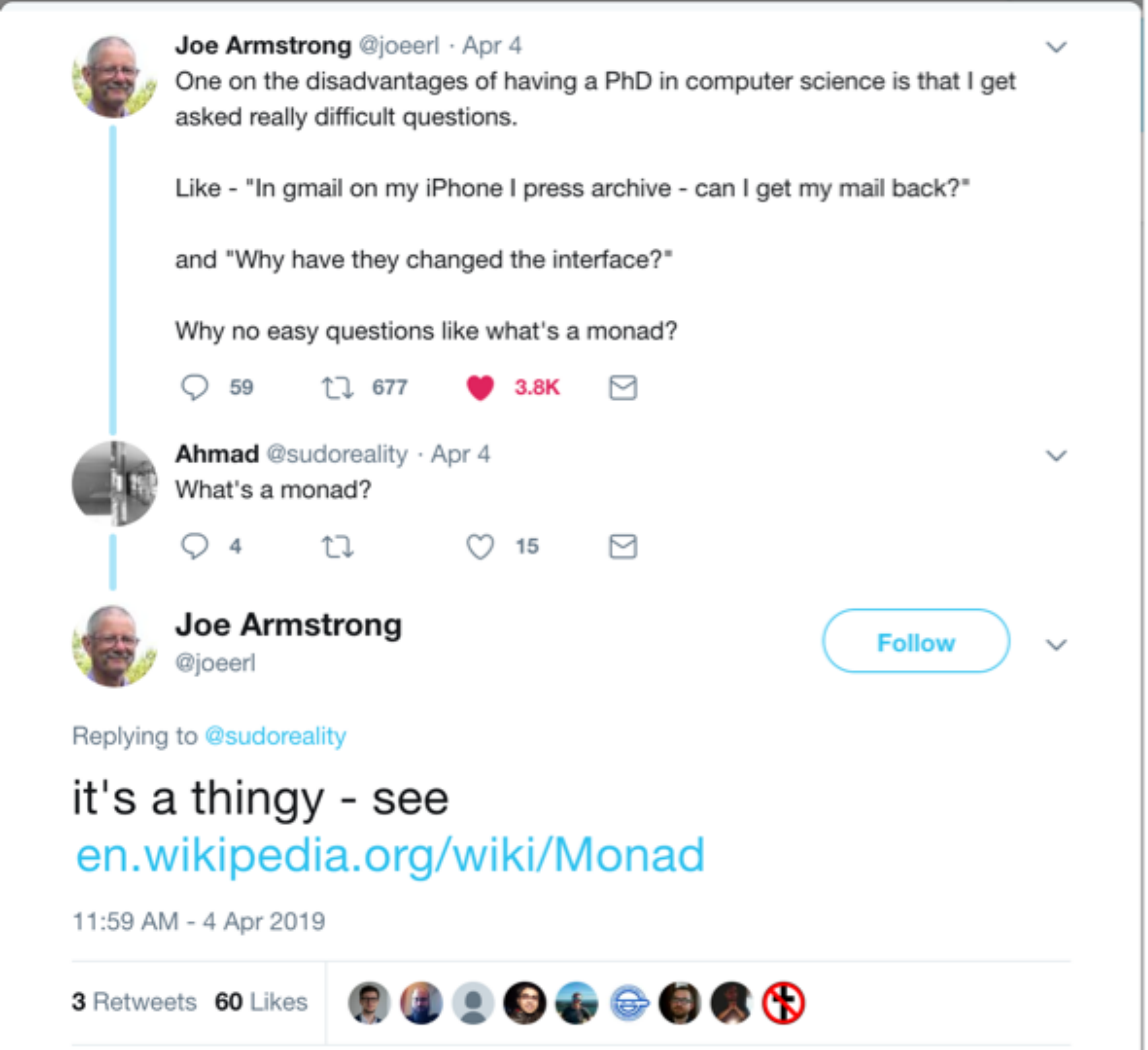
**Duality of Push to Pull Collections**

# 1997 - FRP/FRAN





Functional Reactive Animation
Conal Elliott / Paul Hudak
http://conal.net/papers/icfp97/



Joe Armstrong @joeerl · Apr 4

One on the disadvantages of having a PhD in computer science is that I get asked really difficult questions.

Like - "In gmail on my iPhone I press archive - can I get my mail back?"

and "Why have they changed the interface?"

Why no easy questions like what's a monad?

59 677 3.8K

Ahmad @sudoreality · Apr 4

What's a monad?

4 15

Joe Armstrong
@joeerl

Follow

Replying to @sudoreality

it's a thingy - see en.wikipedia.org/wiki/Monad

11:59 AM - 4 Apr 2019

3 Retweets 60 Likes

## Monads???

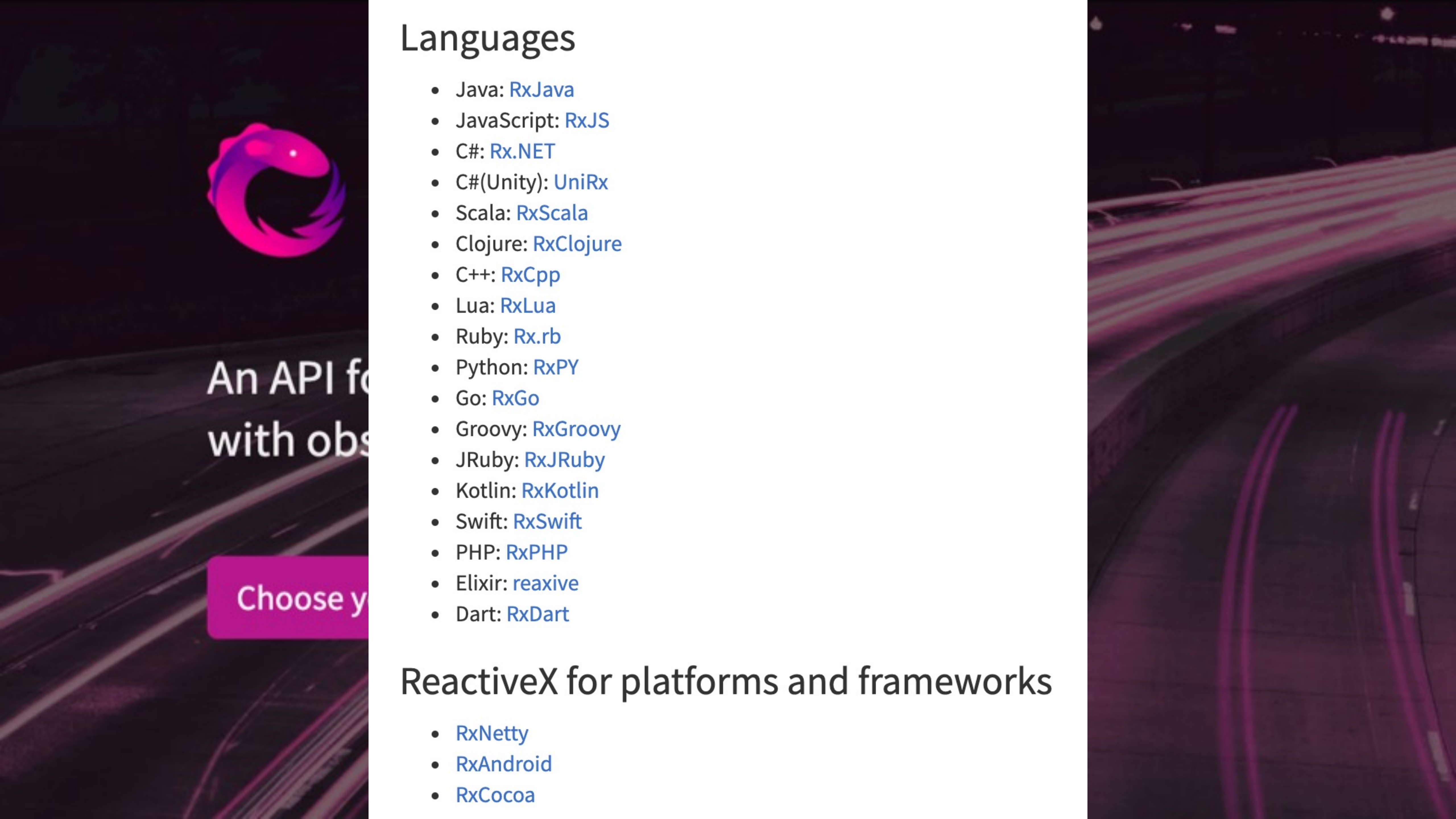
Microsoft Live Labs
Volta

```
public partial class VoltaPage1 : Page
{
    public VoltaPage1()
    {
        var output = new Div();

        var b = new Input();
        b.type = "button";
        b.Value = "Get Message";
        b.Click += () => output.InnerHtml = Handler.GetMessage();

        Document.Body.AppendChild(output);
        Document.Body.AppendChild(b);
    }
}

class Handler
{
    [RunAtOrigin]
    public static string GetMessage()
    {
        return "Hello World";
    }
}
```

## Your Mouse is a Database

Web and mobile applications are increasingly composed of asynchronous and real-time streaming services and push notifications

Erik Meijer
https://queue.acm.org/detail.cfm?id=2169076



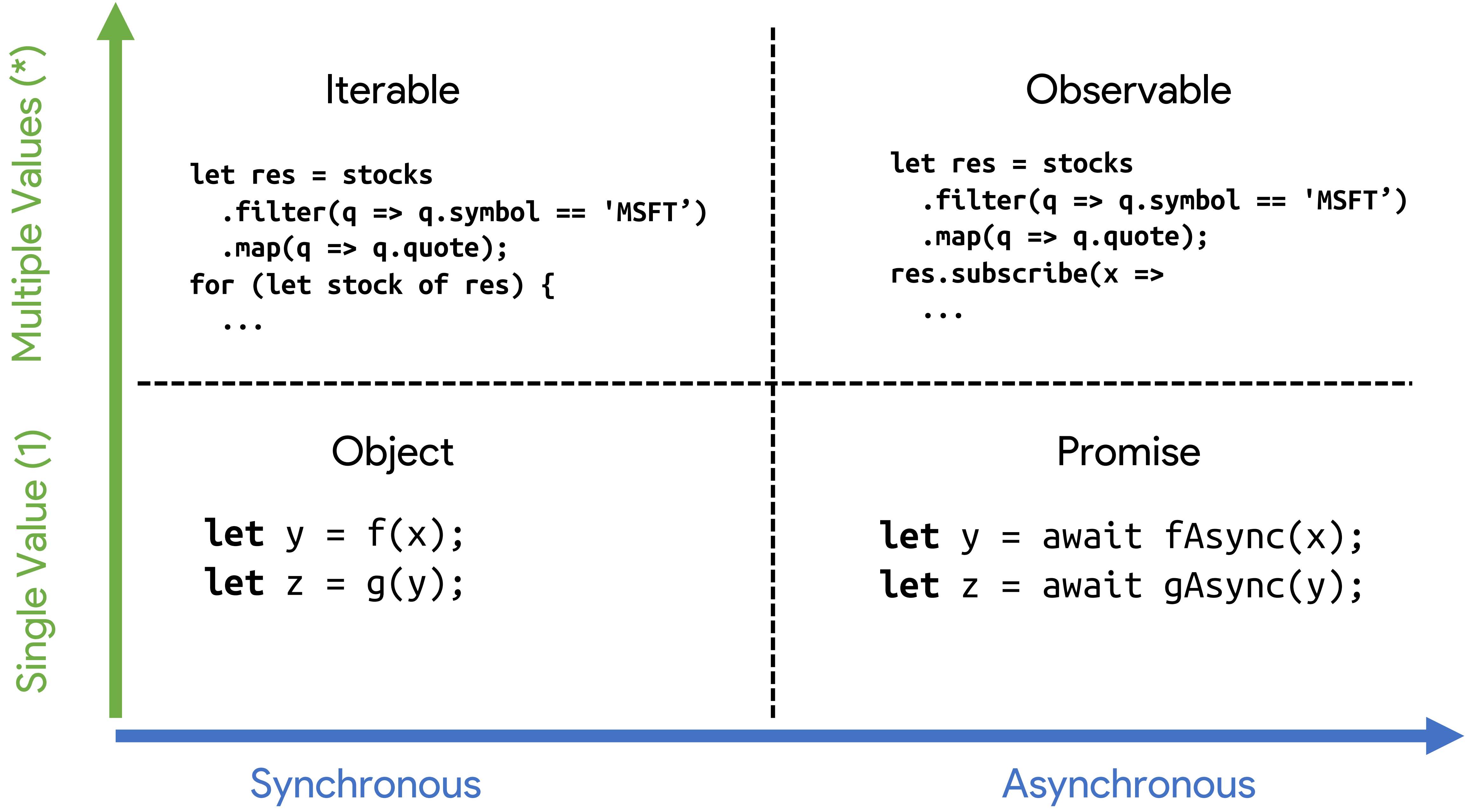
## Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Lua: RxLua
- Ruby: Rx.rb
- Python: RxPY
- Go: RxGo
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift
- PHP: RxPHP
- Elixir: reaxive
- Dart: RxDart

## ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

## General Theory of Reactivity



Iterable

```
let res = stocks
  .filter(q => q.symbol == 'MSFT’)
  .map(q => q.quote);
for (let stock of res) {
  ...
```

Observable

```
let res = stocks
  .filter(q => q.symbol == 'MSFT’)
  .map(q => q.quote);
res.subscribe(x =>
  ...
```

Object

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

Promise

```
let y = await fAsync(x);
let z = await gAsync(y);
```

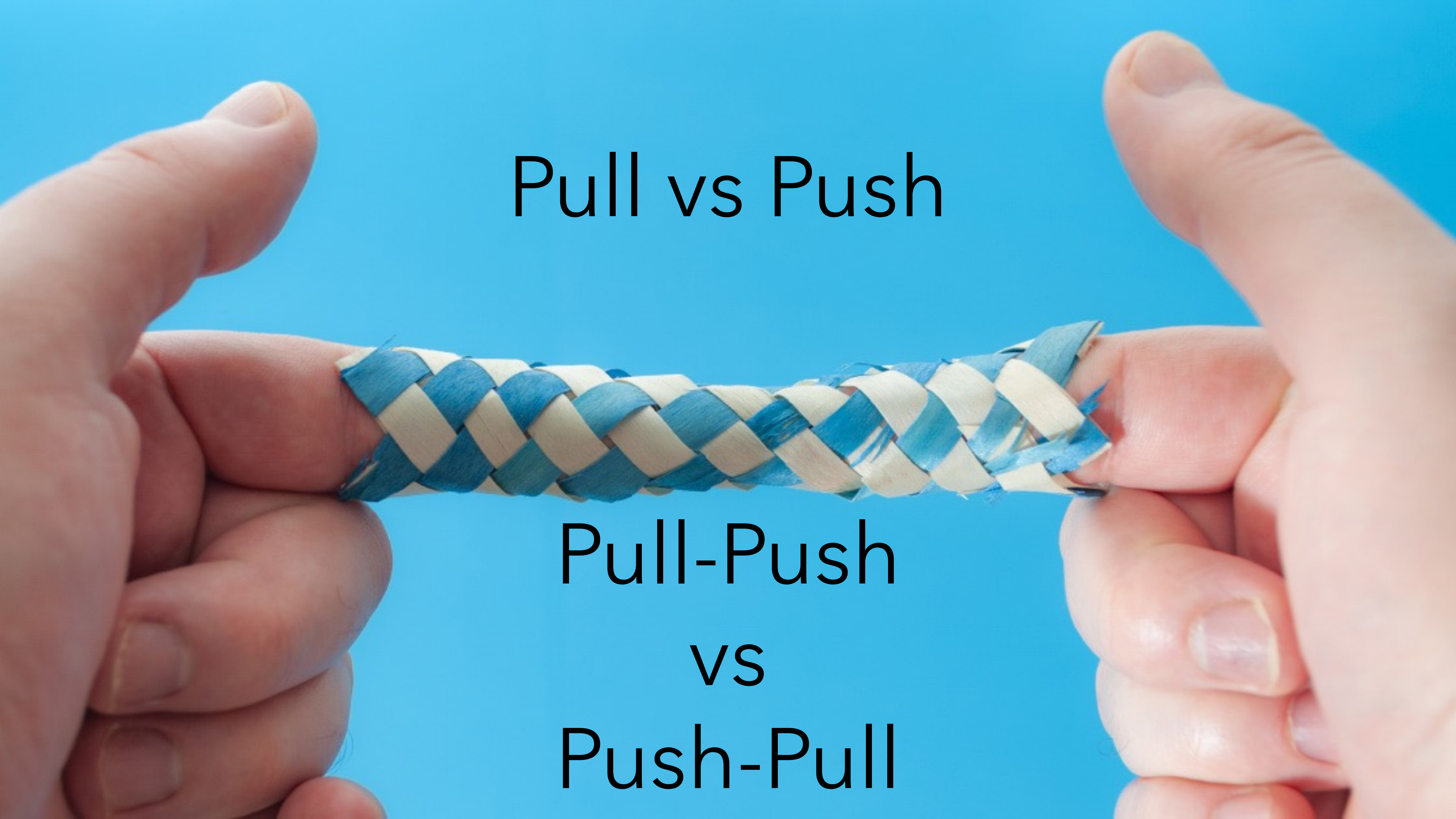
# Things can overload

## Observables get you part of the way there…

**Lossless**
buffer
window

**Lossy**
debounce
sample
throttle

## The Observable Hammer



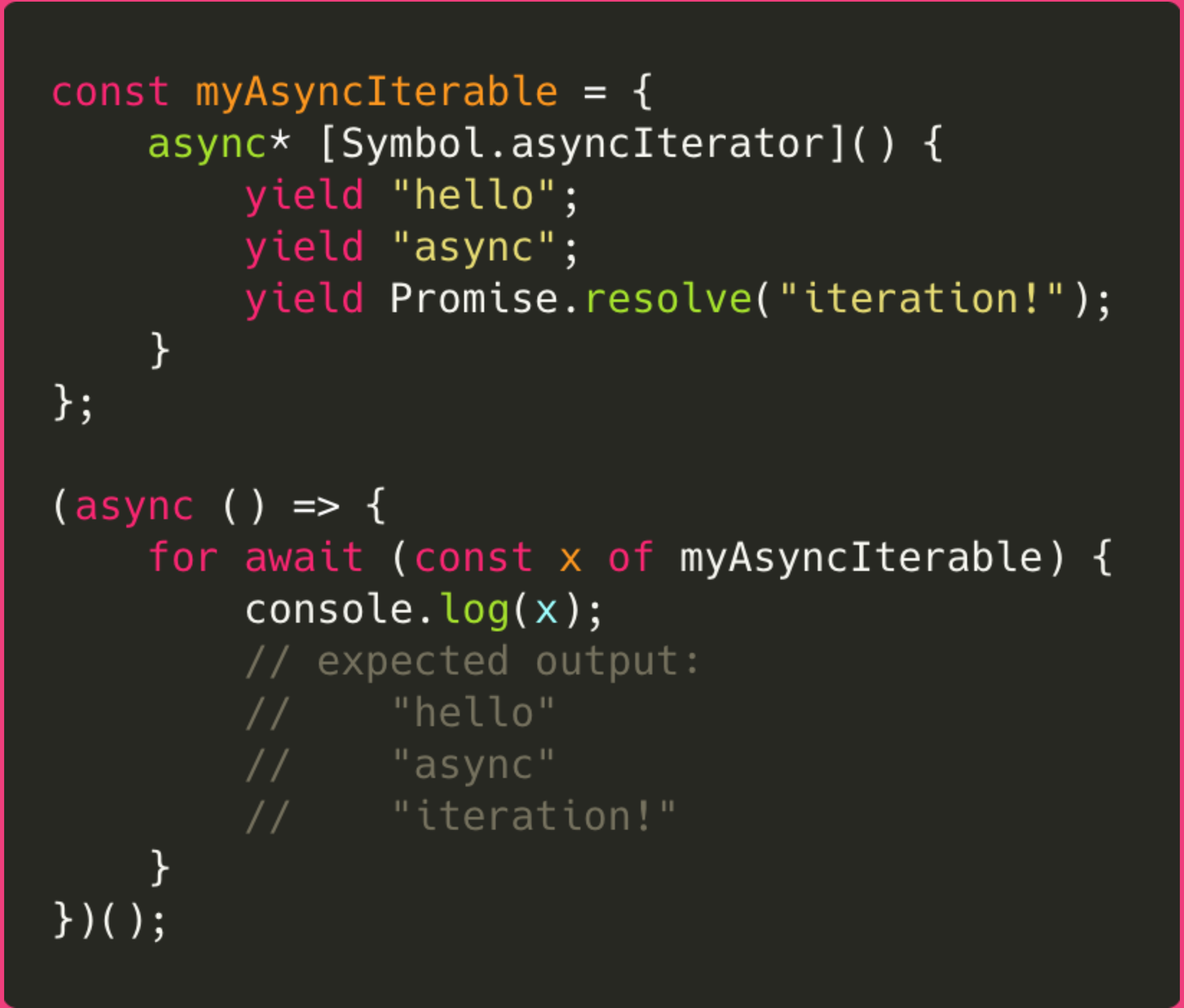
## Pull vs Push

Pull-Push

vs

Push-Pull

## Introducing AsyncIterable



```
const myAsyncIterable = {
    async* [Symbol.asyncIterator]() {
        yield "hello";
        yield "async";
        yield Promise.resolve("iteration!");
    }
};

(async () => {
    for await (const x of myAsyncIterable) {
        console.log(x);
        // expected output:
        //    "hello"
        //    "async"
        //    "iteration!"
    }
})();
```

# Node Streams as AsyncIterable

```
import { createReadStream } from 'fs';

async function printFileToConsole(path) {
  try {
    const readStream = createReadStream(path, { encoding: 'utf-8' });

    for await (const chunk of readStream) {
      console.log(chunk);
    }

    console.log('EOF');
  } catch(error) {
    console.log(error);
  }
}
```

RxJS live! ASIA

# Introducing IxJS AsyncIterable

```
import { as } from 'ix/asynciterable';
import { map } from 'ix/asynciterable/operators';
import { createReadStream } from 'fs';

const stream = as(createReadStream(path, { encoding: 'utf-8' }))
  .pipe(map(transformData))
  .pipe(domEncodeStream);

try {
  for await (const chunk of stream) {
    console.log(chunk);
  }

  console.log('EOF');
} catch (err) {
  console.log(error);
}
```

github.com/reactivex/ixjs

## With AbortController Support

```
import { as } from 'ix/asynciterable';
import { map, withAbort } from 'ix/asynciterable/operators';
import { createReadStream } from 'fs';

const controller = new AbortController();

const stream = as(createReadStream(path, { encoding: 'utf-8' }))
  .pipe(withAbort(controller.signal))
  .pipe(map(transformData))
  .pipe(domEncodeStream);

try {
  for await (const chunk of stream) {
    console.log(chunk);
  }

  console.log('EOF');
} catch (err) {
  if (err.name === 'AbortError') {
    console.log('Aborted');
  } else {
    console.log(error);
  }
}
```

# Enabling cancellation propagation

```
class WithAbortAsyncIterable<TSource> implements AsyncIterable<TSource> {
  private _source: AsyncIterable<TSource>;
  private _signal: AbortSignal;

  constructor(source: AsyncIterable<TSource>, signal: AbortSignal) {
    this._source = source;
    this._signal = signal;
  }

  [Symbol.asyncIterator](): AsyncIterator<TSource> {
    // @ts-ignore
    return this._source[Symbol.asyncIterator](this._signal);
  }
}
```

## Implementing operators

```
export class MapAsyncIterable<TSource, TResult> extends AsyncIterableX<TResult> {
  private _source: AsyncIterable<TSource>;
  private _selector: (value: TSource, signal?: AbortSignal) => Promise<TResult>;
  private _thisArg: any;

  constructor(
    source: AsyncIterable<TSource>,
    selector: (value: TSource, index: number, signal?: AbortSignal) => Promise<TResult>,
  ) {
    super();
    this._source = source;
    this._selector = selector;
  }

  async *[Symbol.asyncIterator](signal?: AbortSignal) {
    throwIfAborted(signal);
    for await (const item of wrapWithAbort(this._source, signal)) {
      const result = await this._selector.call(this._thisArg, item signal);
      yield result;
    }
  }
}
```

## Using Operators

```
import { filter, map, withAbort } from 'ix/asynciterable/operators';

const controller = new AbortController();

async function transformData(term, index, signal) {
  const res = await fetch(buildUrl(term), { signal });
  const json = await res.json();
  return json;
}

async function filterData(term, index, signal) {
  const res = await fetch(buildFilterUrl(term), { signal });
  const json = await res.json();
  return json.contents.length > 0;
}

const result = getTerms()
  .pipe(withAbort())
  .pipe(filter(filterData))
  .pipe(map(transformData));
```

# Introducing IxJS AsyncObservable

```
import { filter, map } from 'ix/asyncobservable/operators';

const controller = new AbortController();

const stream = getData()
  .pipe(map(transformData))
  .pipe(filter(filterData));

const subscription = await stream.subscribeAsync({
  next: async item => await processItem(item),
  error: async err => await processError(err)
}, controller.signal);

await subscription.unsubscribeAsync();
```

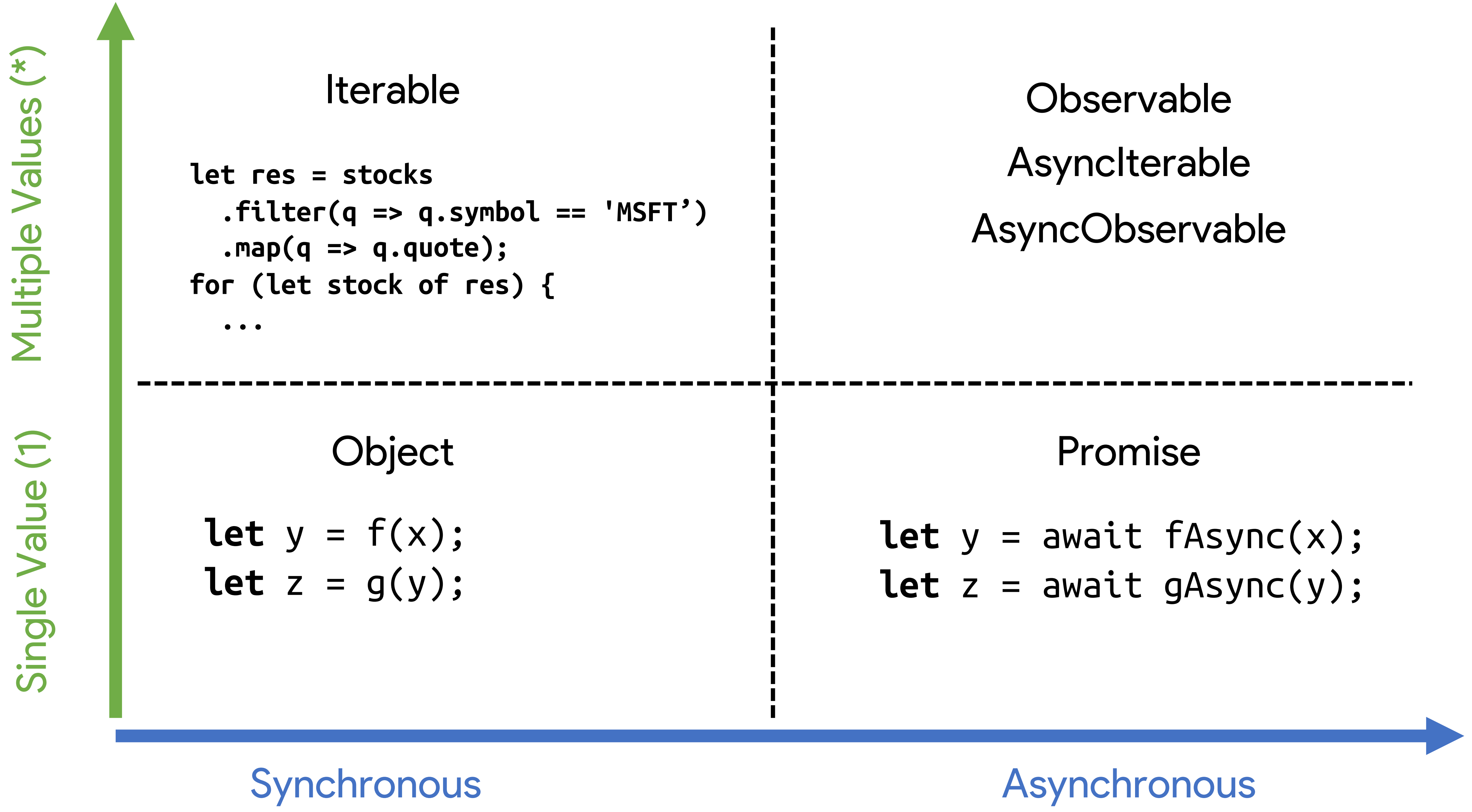
github.com/reactivex/ixjs

# Why IxJS?

**AsyncIterable**
Put Consumer in charge of data flow
Enable integration with Node and DOM Streams
Enable Deep Cancellation within AsyncIterable Streams
Enable Async Projections throughout the chain

**AsyncObservable**
Put Producer in charge of data flow with consumer pulling as needed
Enable Async Projections within AsyncObservable Streams
Enable Async Subscription/Unsubscription from Streams
Enable integration with AbortController APIs

# General Theory of Reactivity

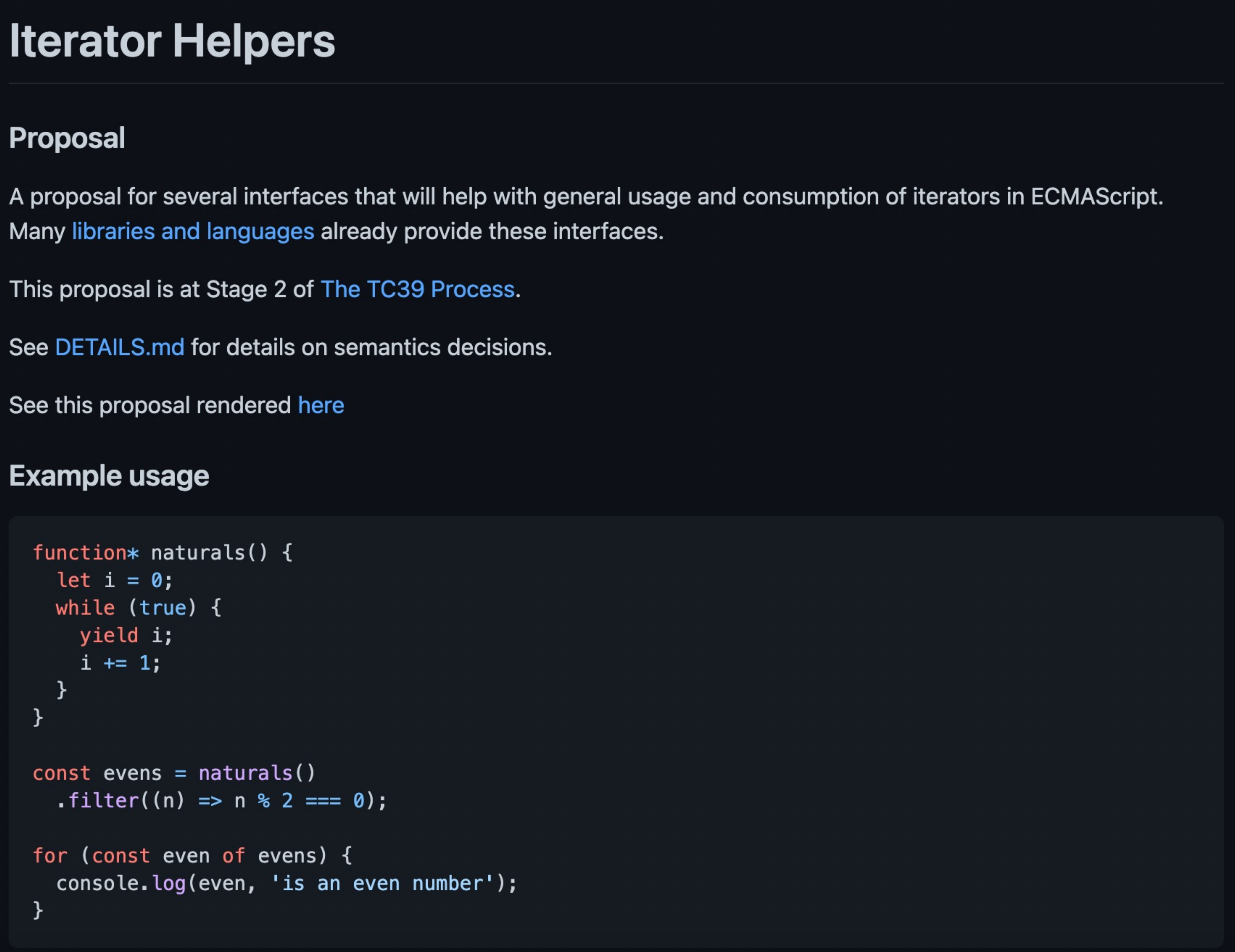


```
let res = stocks
  .filter(q => q.symbol == 'MSFT’)
  .map(q => q.quote);
for (let stock of res) {
  ...
```

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

```
let y = await fAsync(x);
let z = await gAsync(y);
```

Where will we go next?



# Iterator Helpers

## Proposal

A proposal for several interfaces that will help with general usage and consumption of iterators in ECMAScript. Many libraries and languages already provide these interfaces.

This proposal is at Stage 2 of The TC39 Process.

See DETAILS.md for details on semantics decisions.

See this proposal rendered here

## Example usage

```
function* naturals() {
  let i = 0;
  while (true) {
    yield i;
    i += 1;
  }
}

const evens = naturals()
  .filter((n) => n % 2 === 0);

for (const even of evens) {
  console.log(even, 'is an even number');
}
```

https://github.com/tc39/proposal-iterator-helpers

# Let's Build the Future Together!

github.com/mattpodwysocki/rxjs-live-asia-21

Matthew Podwysocki
@mattpodwysocki