Democratizing event processing at all scales and platforms with Reactive Extensions (Rx)

Matthew Podwysocki   @mattpodwysocki

@ReactiveX

https://github.com/Reactive-Extensions/NDC-Oslo-2014

OR:
HOW I LEARNED TO STOP WORRYING ABOUT ASYNCHRONOUS PROGRAMMING AND LOVE THE OBSERVABLE

trapd in Monad tutorl
plz help

streem prosesing

Real-time is everywhere…

Stock tickers

RSS feeds

GPS

Reactive Extensions (Rx)

Social media

UI events

Server management
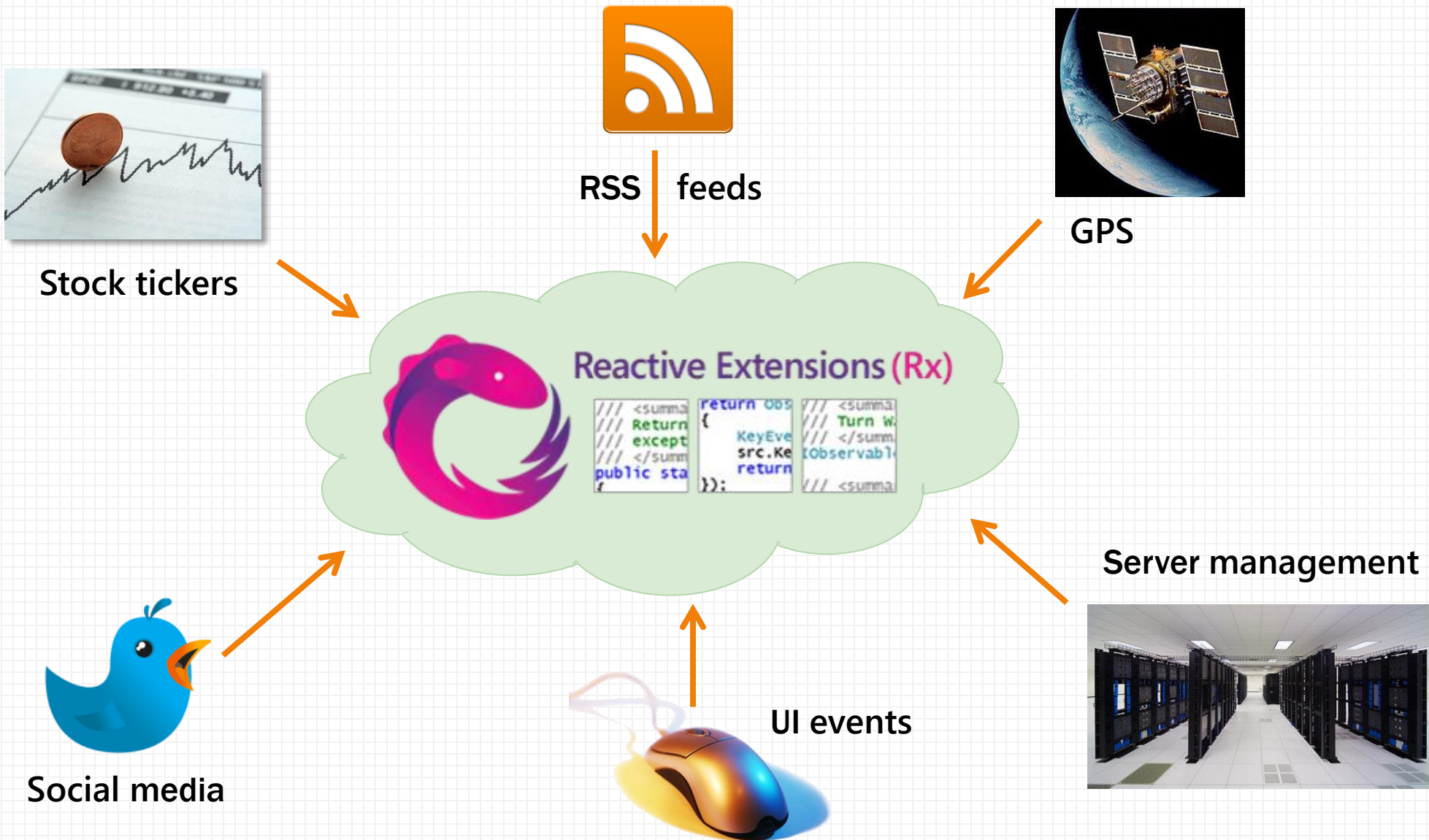
Scalable Abstractions
Powering Cortana

Native RxCPP
Small FootPrint
Processing Sensors

Managed code (Rx.NET)
Across Data Centers
Processing the World's Data

Hi. I'm Cortana.

bing

http://channel9.msdn.com/posts/Scalable-Information-Stream-Processing-by-Bing-in-Support-of-Cortana-Scenarios

Top-rated Movies Collection

```
Func<User, IObservable<Video>> getTopRatedFilms = user =>
    user.VideoLists
        .Select(videoList =>
            videoList.Videos
                .Where(video => video.Rating === 5.0)).
        Merge();

getTopRatedFilms(user)
    .Subscribe(film => Console.WriteLine(film));
```

What if I told you…

…that you could create a drag event…

…with the almost the *same code*?

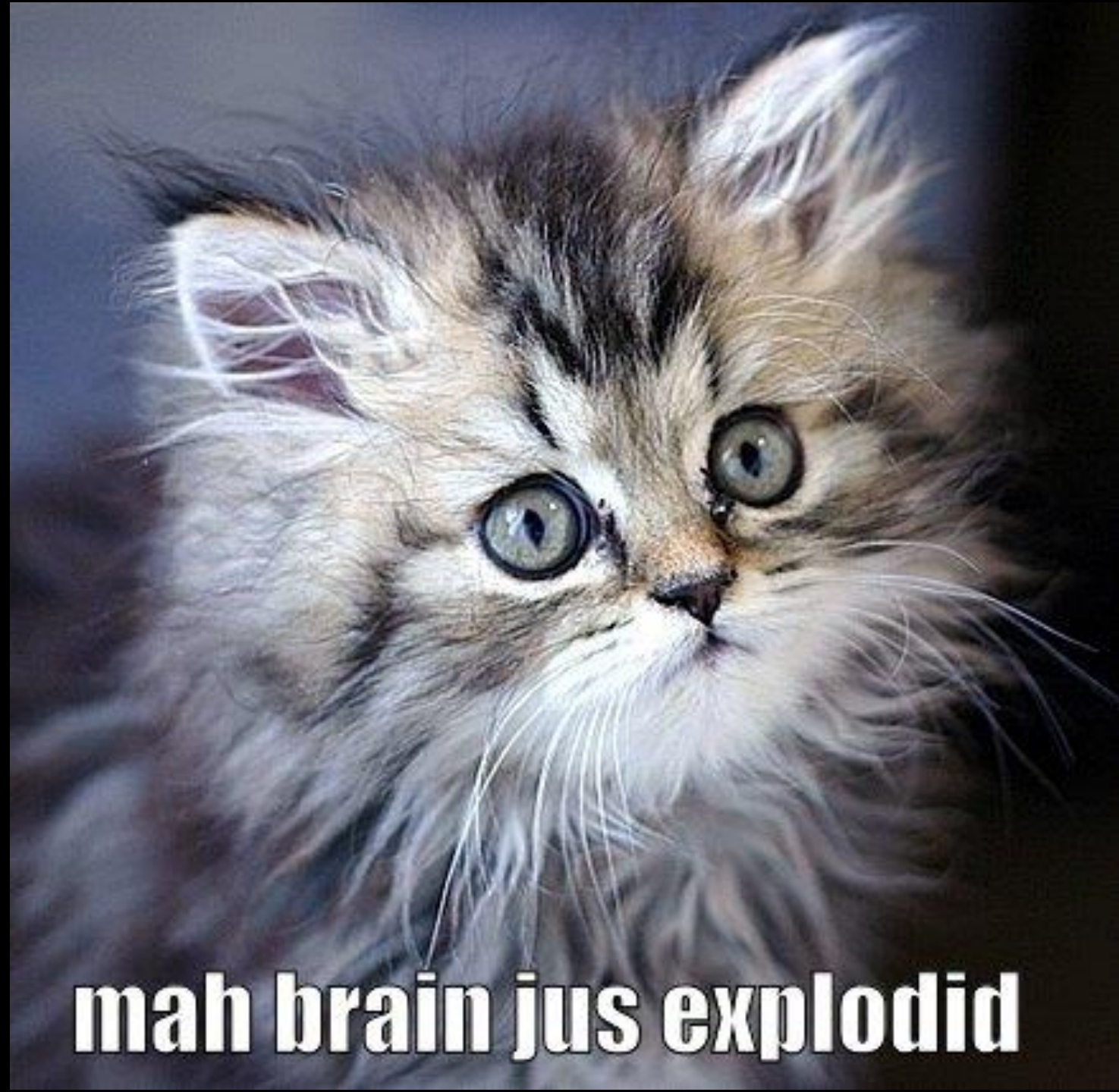# Mouse Drags Collection

```
Func<Element, IObservable<Event<EventArgs>> getElementDrags  = elmt =>
    elmt.MouseDowns()
        .Select(mouseDown =>
            mainWindow.MouseMoves()
                Where .TakeUntil(elmt.MouseUps())))
        .Merge();

getElementDrags(image)
    .Subscribe(pos => image.Position = pos);
```

mah brain jus explodid

Callback Hell

```javascript
function play(movieId, callback) {
    var movieTicket, playError,
        tryFinish = function () {
            if (playError) {
                callback(null, 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();
    });
});
```

Asynchronous Programming is Annoying

Each language has its own way of expressing async/event-based programming

– Java futures are different from JavaScript promises are different from Clojure core.async) are different from .NET Tasks

– Each concept covers only part of the story

Wouldn't it be great to have a unifying concept to generalize how we think about concurrent/reactive programming?

Reactive Manifesto

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

OnNext:
Reactive Applications Demo

Ordinary Interactive Programming

```
try {
    foreach (var item in collection)
        DoSomething();                        ←——————  OnNext(T)
}
catch (Exception e) {
    HandleOrThrow(e);                         ←——————  OnError()
}

DoCleanup();                                  ←——————  OnCompleted()
```
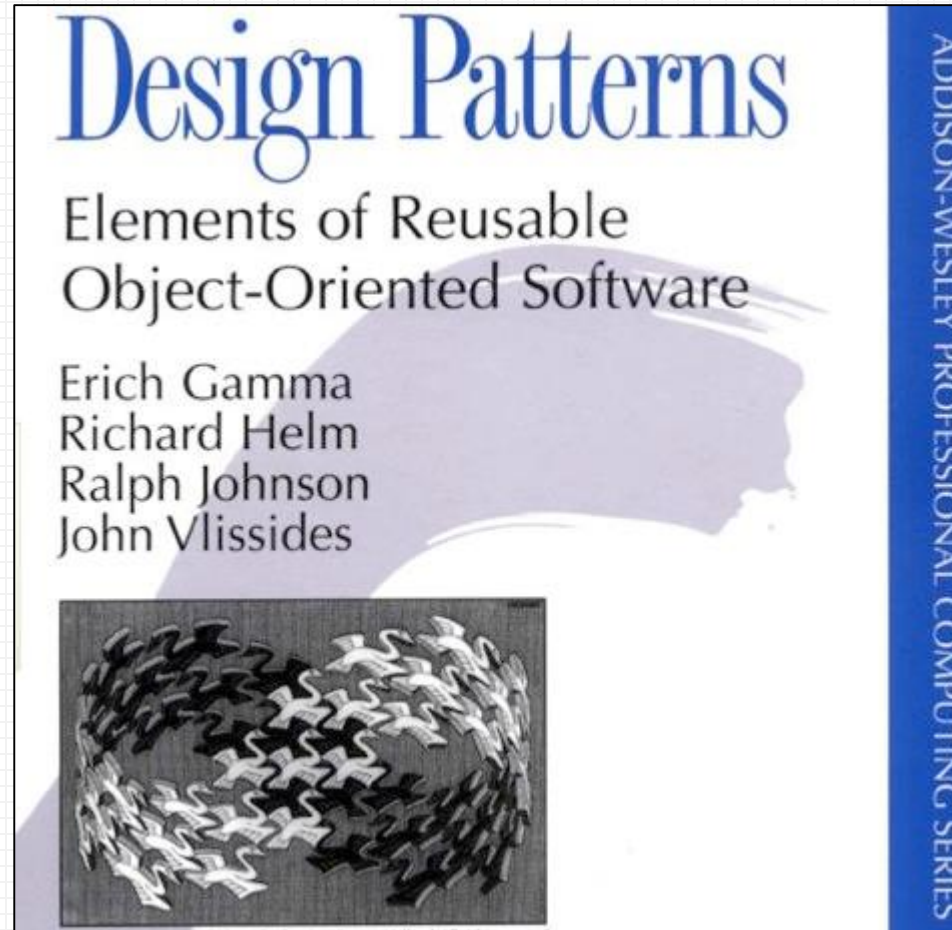
That was the iterator pattern

Making it push-based

```
IObservable<T> collection = ...

var obs = Observer.Create(
            onNext:      x  => DoSomething(x),
            onError:     e  => HandleError(e),
            onCompleted: () => DoCleanup());


var subscription = collection.Subscribe(obs);

// deterministically cleans up all resources
subscription.Dispose();
```

Rx Grammar Police

OnNext ● * (OnError ✖ | OnCompleted ▌) ?

Zero or more values

E.g. events are ∞ sequences

Calls can fail

Resource management

Sequencing

✔ ─── 0 ─── 1 ─── 2 ───

✔ ─── 0 ── 1 ── 2 ── ✖

✔ ─── 0 ─── 1 ──▌

✖ ─── ⬭0 ⬭1 ─▌ 2

- Must be a sequence
- No concurrent callbacks
- One time termination
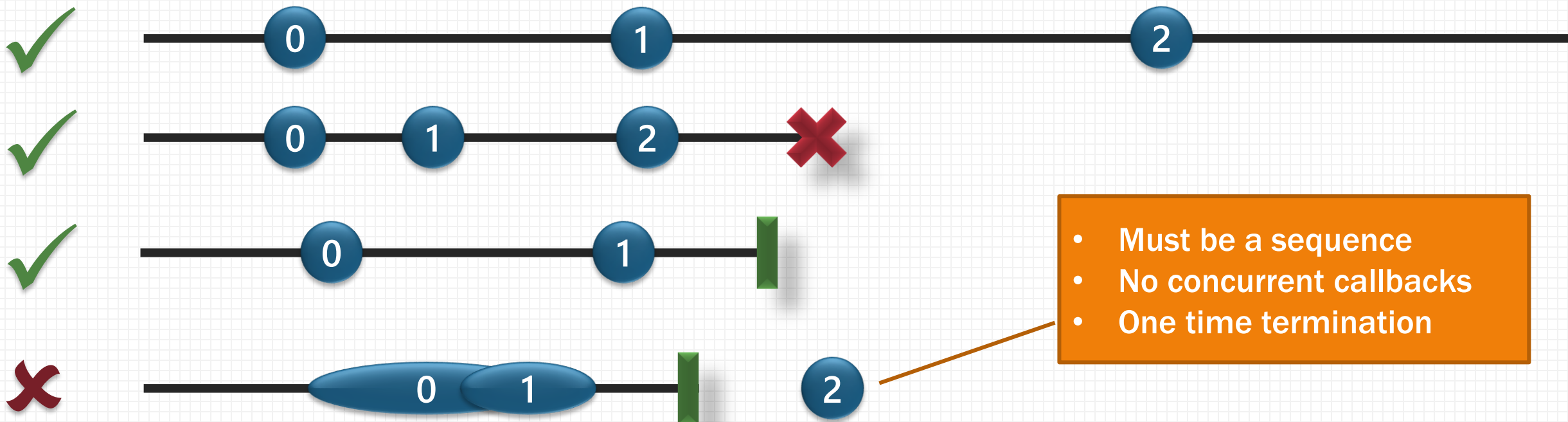
First-Class Events

Objects to the rescue

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)

How about a **query library?**

Or mocking for testing...?
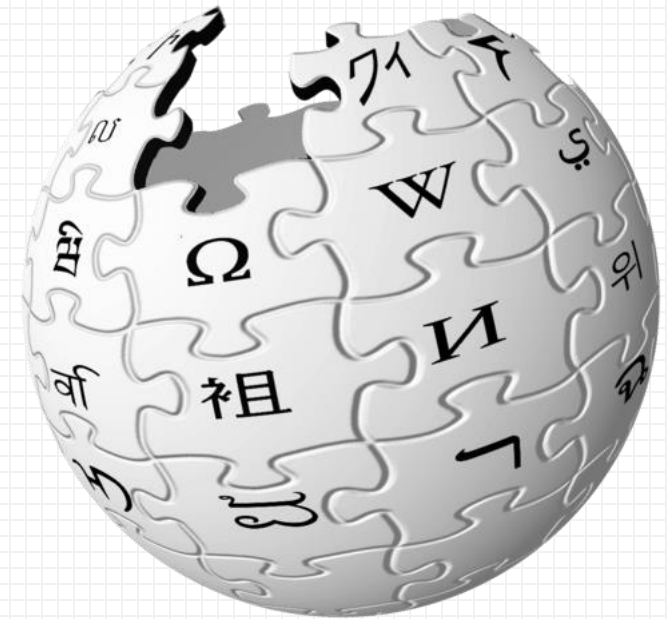
Observables are awaitable...

```csharp
static async Task<int> DoStuff()
{
    var observable = Observable.Return(42);
    return await observable;
}

static async void GetStuff()
{
    var value = await DoStuff();
    Console.WriteLine(value);
}
```

imperative cat

is not amused

Observables are more powerful...

```csharp
async Task<int> GetMovies()
{

    var observable = service.GetMovies().Retry(3)
    return await observable;

}


async void ProcessMovies()
{

    var movies = await GetMovies();
    Console.WriteLine(movies.Count);

}
```

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

Rx is everywhere*

.NET

JavaScript (RxJS)

Java (RxJava)

  + Scala, Groovy, Clojure

Objective-C (ReactiveCocoa)

C++

Python

Ruby

PHP

Dart

Haskell

Swift???

* Varying levels of completeness – YMMV

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

Reactive collections: the dual of iterable collections

IEnumerable	IObservable
pull	push
foreach	Subscribe(IObserver)
T Current, bool MoveNext()	OnNext(T)
throws Exception	OnError(Exception)
returns	OnCompleted()

```
// IEnumerable<Stock>
// Historical stock data

stocks
    .Where(q => q.Symbol == "FB")
    .Select(q => q.Quote)
    .ForEach(Console.WriteLine);
```

```
// IObservable<Stock>
// Incoming stock feed

stocks
    .Where(q => q.Symbol == "FB")
    .Select(q => q.Quote)
    .Subscribe(Console.WriteLine);
```

Marble diagram: filter
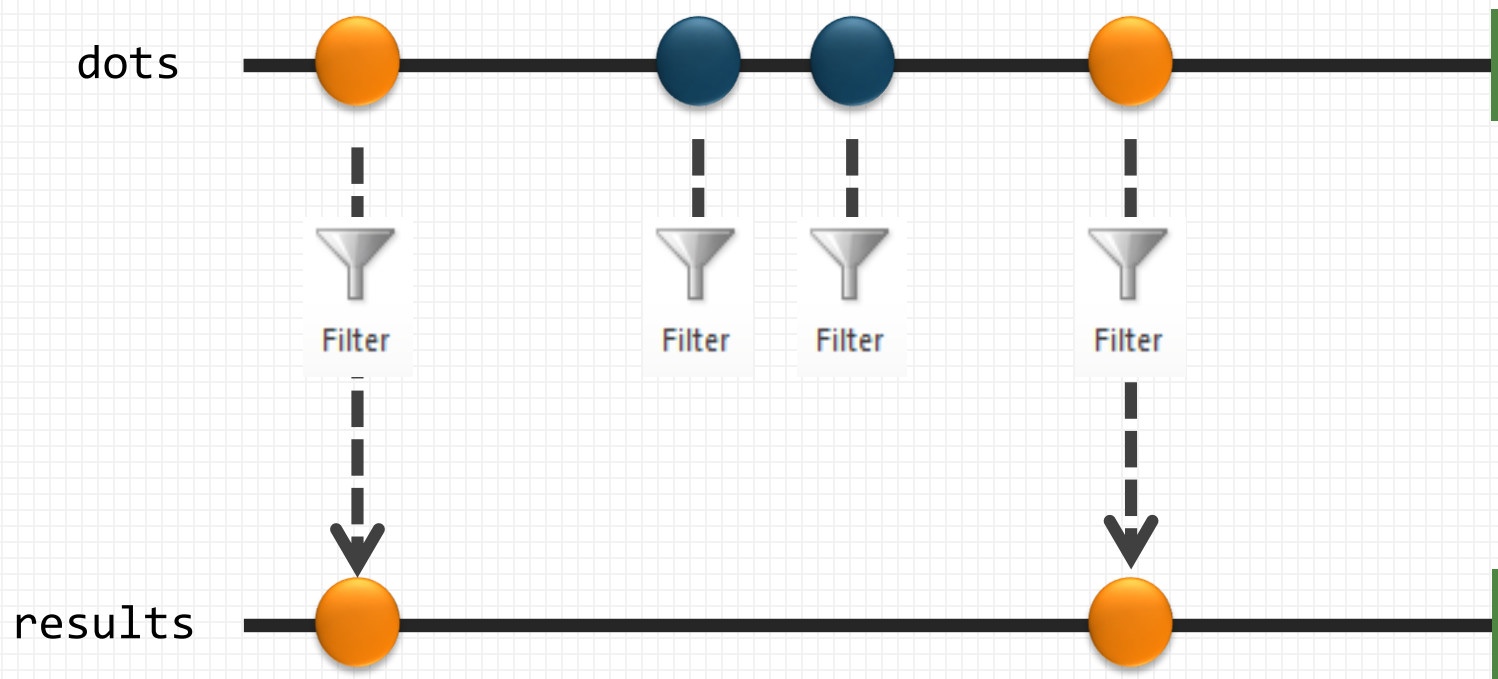

```
.filter(function (dot) {
    return dot.isOrange();
})
```

Marble diagram: map
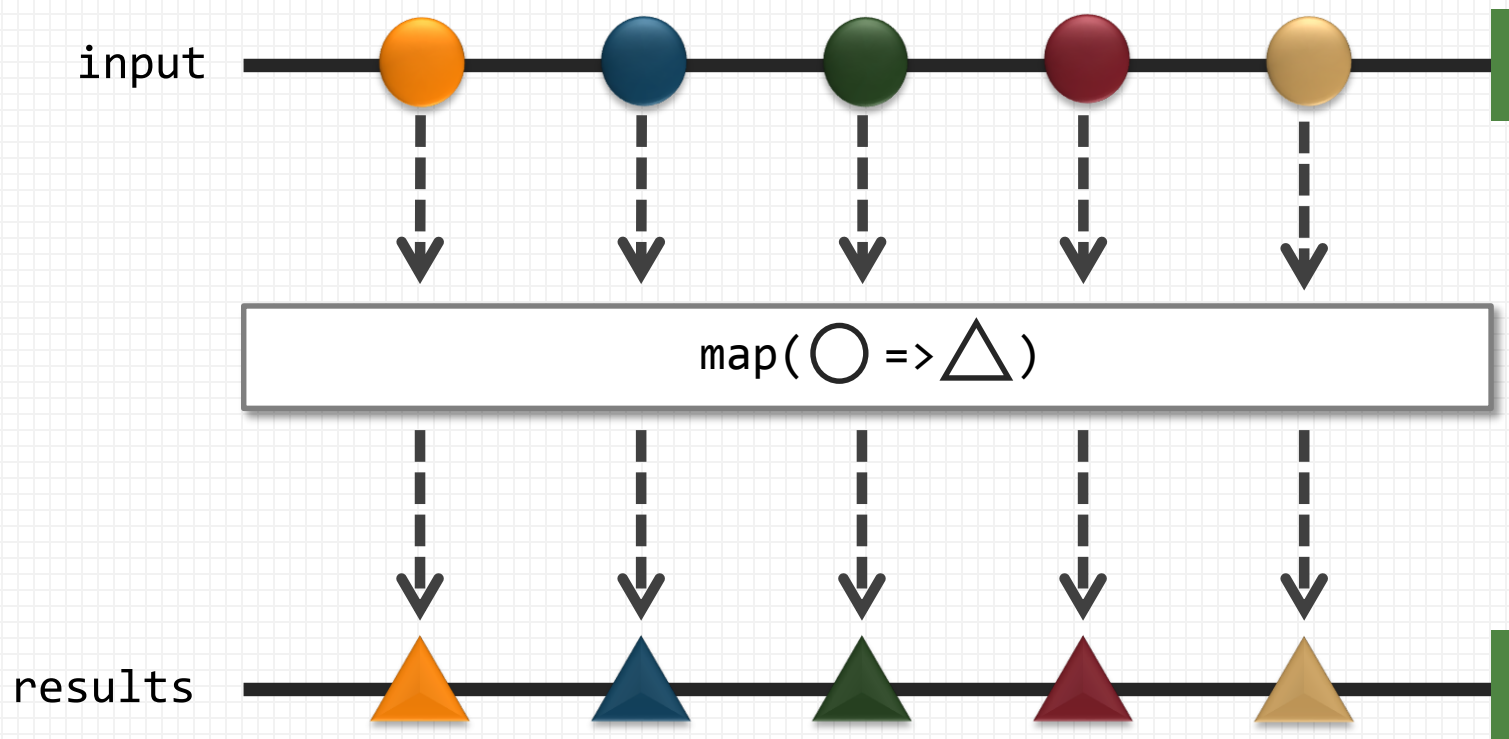

```
.map(function (item) {
    return transform(item);
})
```

Marble diagram: delay

Since Observables are asynchronous, they have a notion of time

input

.delay(1000)

results
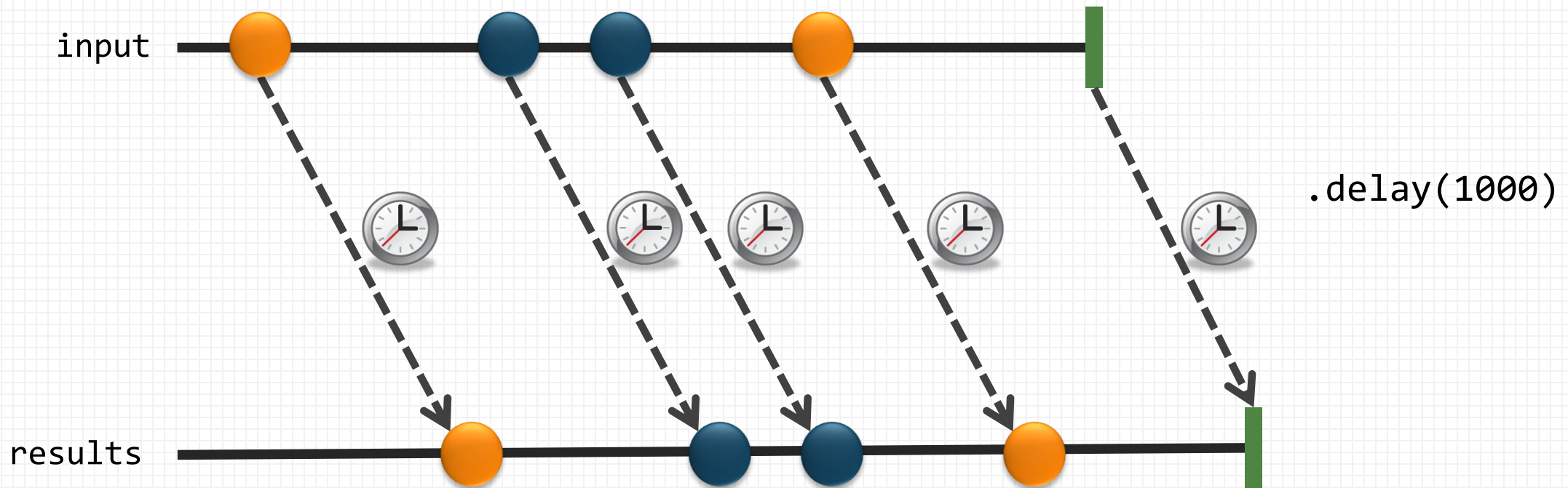
The Asynchronous Programming Landscape
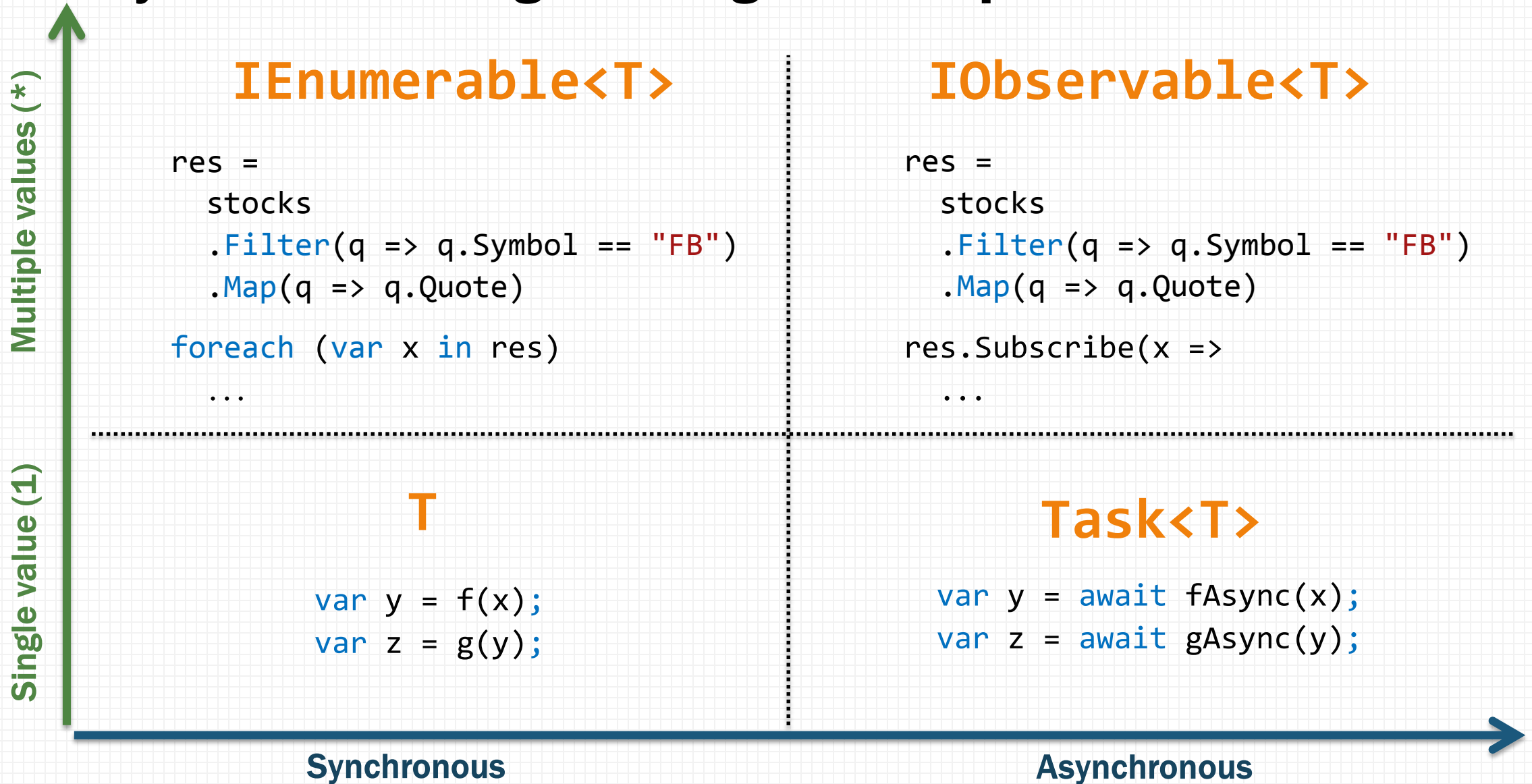
OnNext:
Drag and Drop

Querying UI Events

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

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

For each mouse down **1**

```
});
```

Querying UI Events

```javascript
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

Querying UI Events

```javascript
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

Composing Events and Promises

Observable of strings

XHR web service request

React

keyup

Dictionary
web service

Reaction
Reactive
Reactor

Data binding on HTML page

Observable of arrays of words
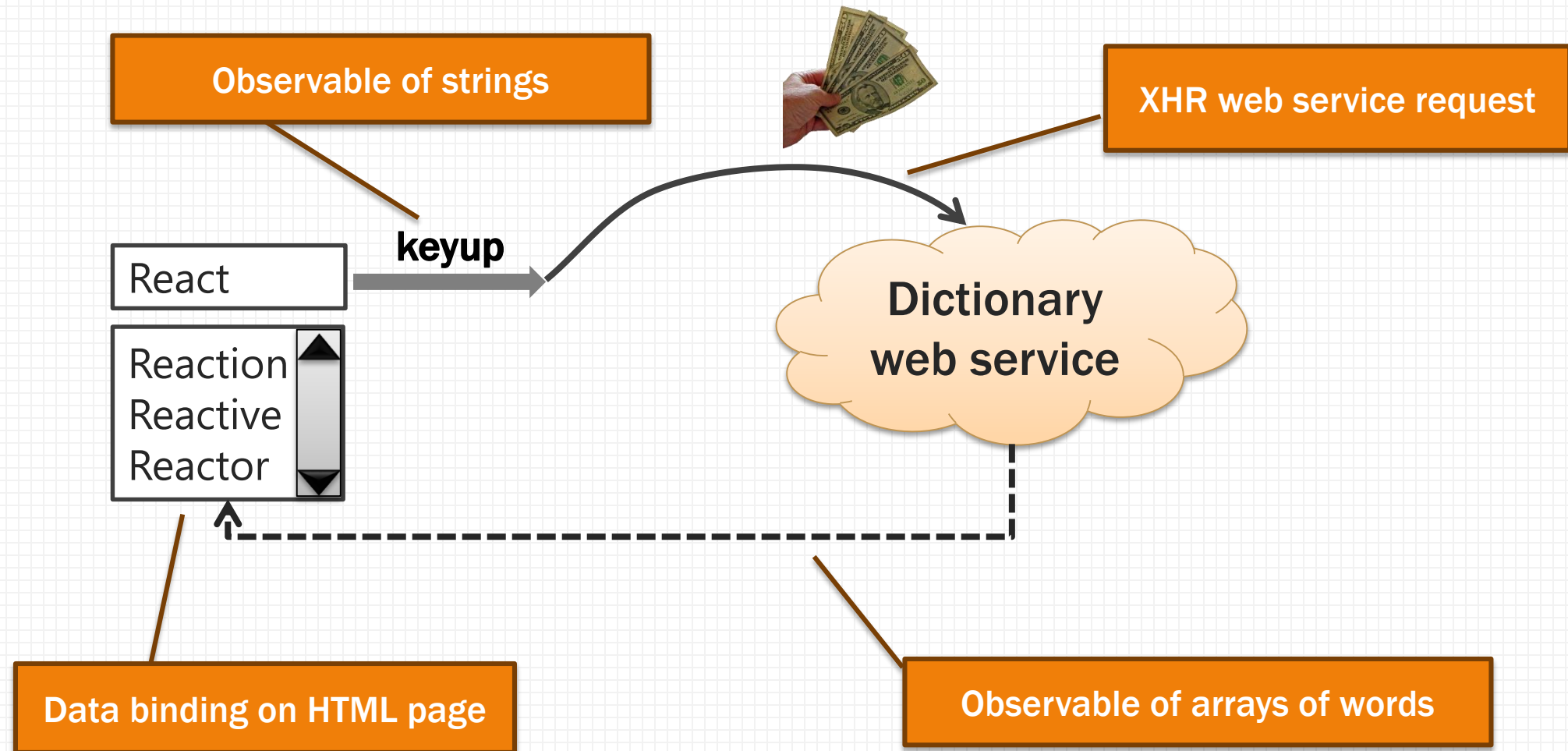
Composing Events and Promises

```
var words = Rx.Observable.fromEvent(
        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

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, "NDC"),
    onNext(400, "OSLO"),
    onCompleted(500));


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


results.messages.assertEqual(
    onNext(300, 3),
    onNext(400, 4),
    onCompleted(500));
```

More about Rx

Open-sourced by MS Open Tech in Nov 2012

– **Rx.NET**

– **RxJS**

– **RxCpp**

Who uses Rx?

– **Netflix ported it to Java (RxJava)**

• **Heavily used in back-end**

• **Use RxJS/Rx.NET on clients**
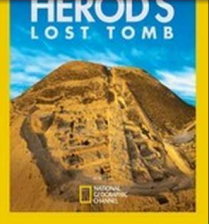
– **GitHub**

• **GitHub for Windows (ReactiveUI + Rx.NET)**

• **GitHub for Mac (ReactiveCocoa)**
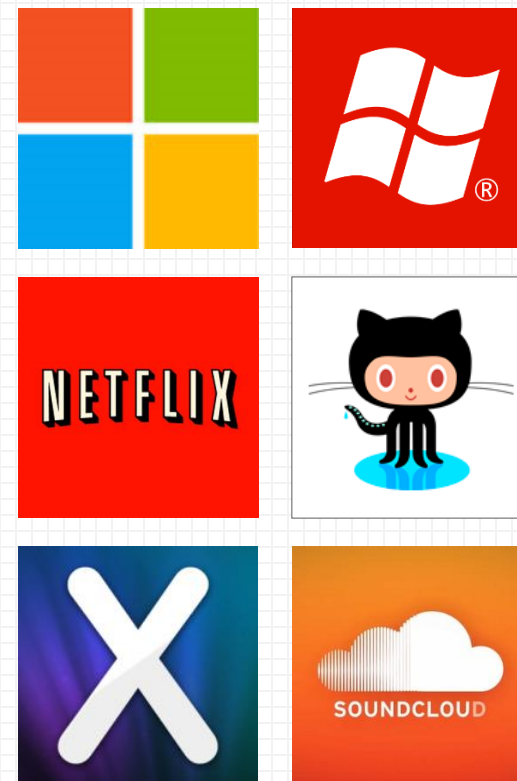
Reactive Programming at Netflix

Netflix

- **Stream movies to your TV, game console, or mobile device**

- **Responsible for 1/3 of US Broadband Traffic**

Two Years Ago...

- **Complex async code**

- **Client and Server developers tightly coupled**

- **Different platforms, different approaches to async**

Today

- **Rx used on server and client**

- **30+ developers using Rx in 5 different languages**

- **Same async model everywhere**
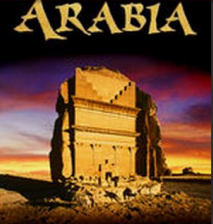
End to End Reactive Programming at Netflix

Your Netflix Video Lists

Netflix Row Update Polling

Client: Polling for Row Updates

```javascript
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();
};
```

Server: Retrieving Social Row

```
IObservable<List<dynamic>> GetSocialRowData() {
    return Observable.RelationalJoin(
        socialService.GetFriends(user),
        messageService.GetNotifications()
            .Where(notification => notification.Video.IsAvailable),
        friend => friend.Id,    // join key selector
        notification => notification.Friend.Id, // join key
        (friend, notification) =>
            new {
                Id = notification.Id,
                Name = notification.Video.Name,
                Message = notification.Message,
                Friend = new { Name = friend.Name, Id = friend.Id }
            }).
        ToList();
}
```

Netflix uses Rx for...

- **Data Binding**

- **Data Access**

- **Animations**

- **Building Complex Events**

The Future Of Rx…

What are the problems we're looking at next?

- **Backpressure**

- **Distributed Rx**

- **Query Expressions in JavaScript**

- **Rx-Lite**

- **Custom RxJS Builds**

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/
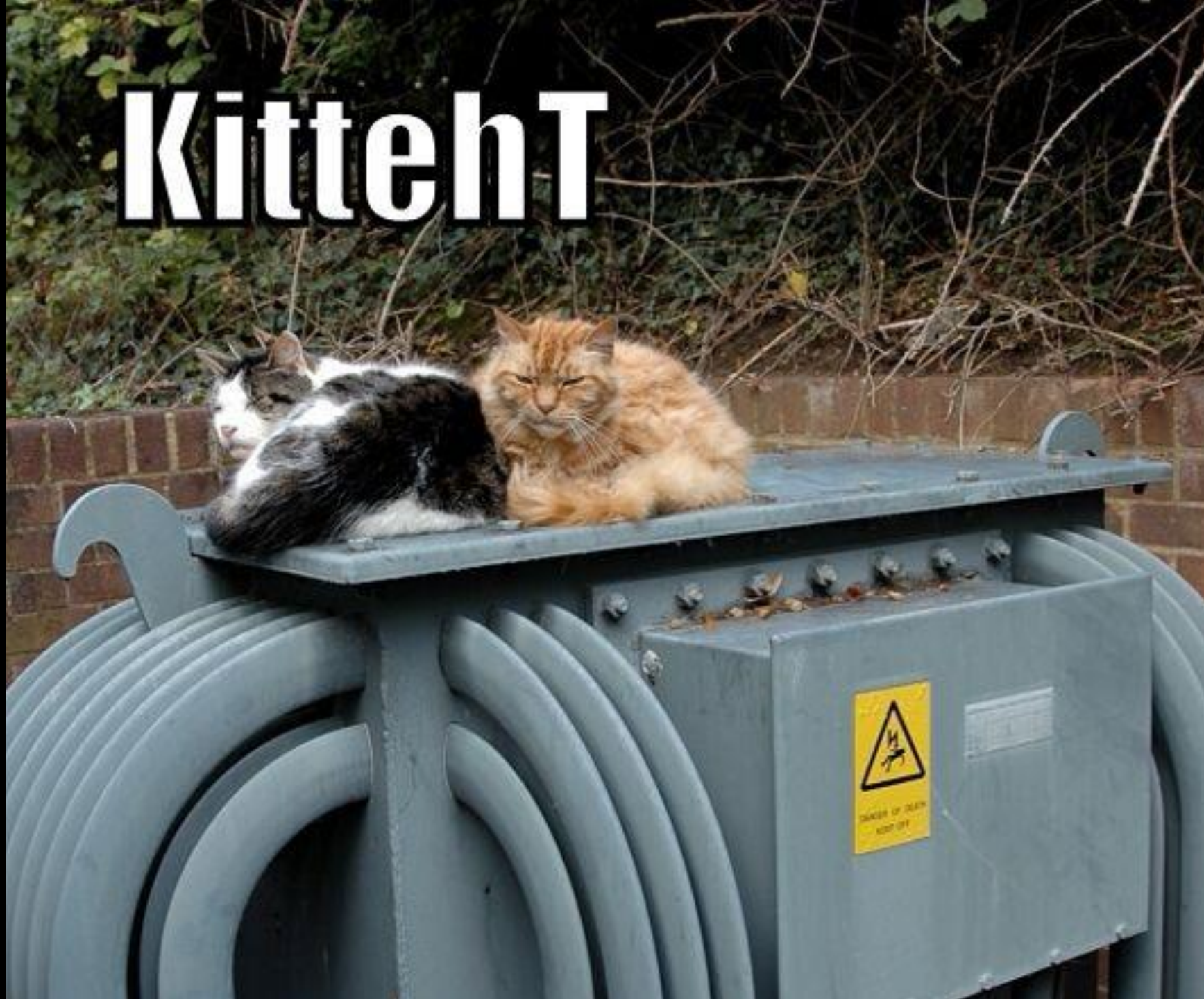
A Glimpse into the Future - RxJS and Generators

```javascript
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);

    try {
        var b = yield get('http://invalidhost');
    } catch (e) {
        console.log(e.code);
    }
});
```

OnCompleted: Rx

Language neutral model with 3 concepts:

1. Observer/Observable

2. Query operations (map/filter/reduce)

3. Schedulers: a set of types to parameterize concurrency

@ReactiveX
github.com/Reactive-Extensions
rx.codeplex.com

https://github.com/Reactive-Extensions/NDC-Oslo-2014