Curing Your Event Processing Blues with Reactive Extensions (Rx)

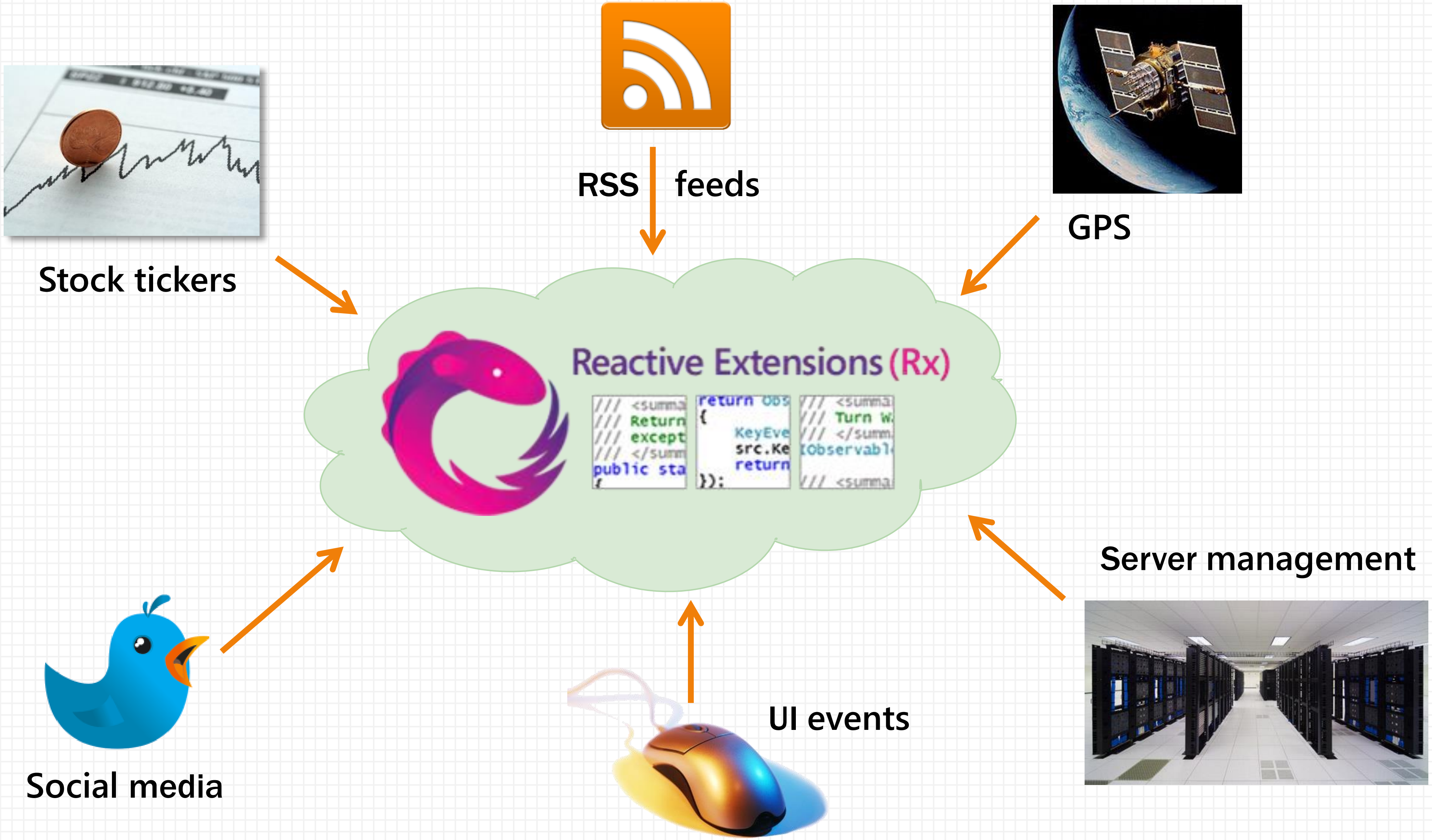
Matthew Podwysocki @mattpodwysocki
Donna Malayeri @lindydonna

Microsoft

@ReactiveX

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

Real-time is everywhere...



Stock tickers

RSS feeds

GPS

Reactive Extensions (Rx)

```
/// <summa
/// Return
/// except
/// </summ
public sta
{
```

```
return Obs
{
    KeyEve
    src.Ke
    return
});
```

```
/// <summa
/// Turn W
/// </summ
IObservabl

/// <summa
```

Server management

UI events

Social media

Asynchronous Programming is Annoying

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

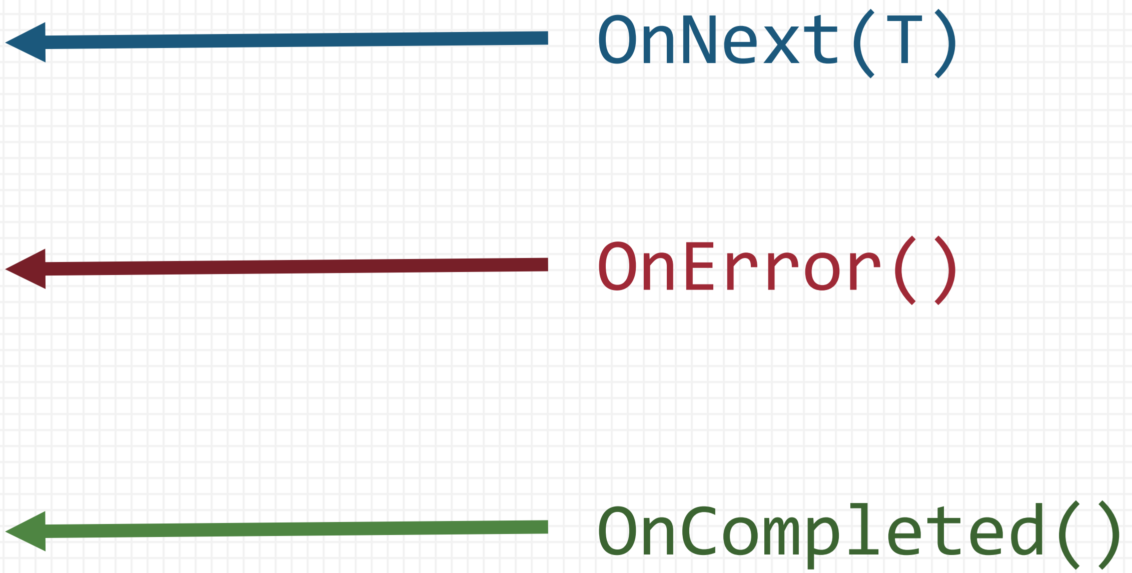
- Java promises are different from JavaScript promises are different from Clojure core.async)
- Each concept covers only part of the story

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

Demo
Reactive Applications

Ordinary Interactive Programming

```
try {
    foreach (var item in collection)
        DoSomething();
}
catch (Exception e) {
    HandleOrThrow(e);
}

DoCleanup();
```

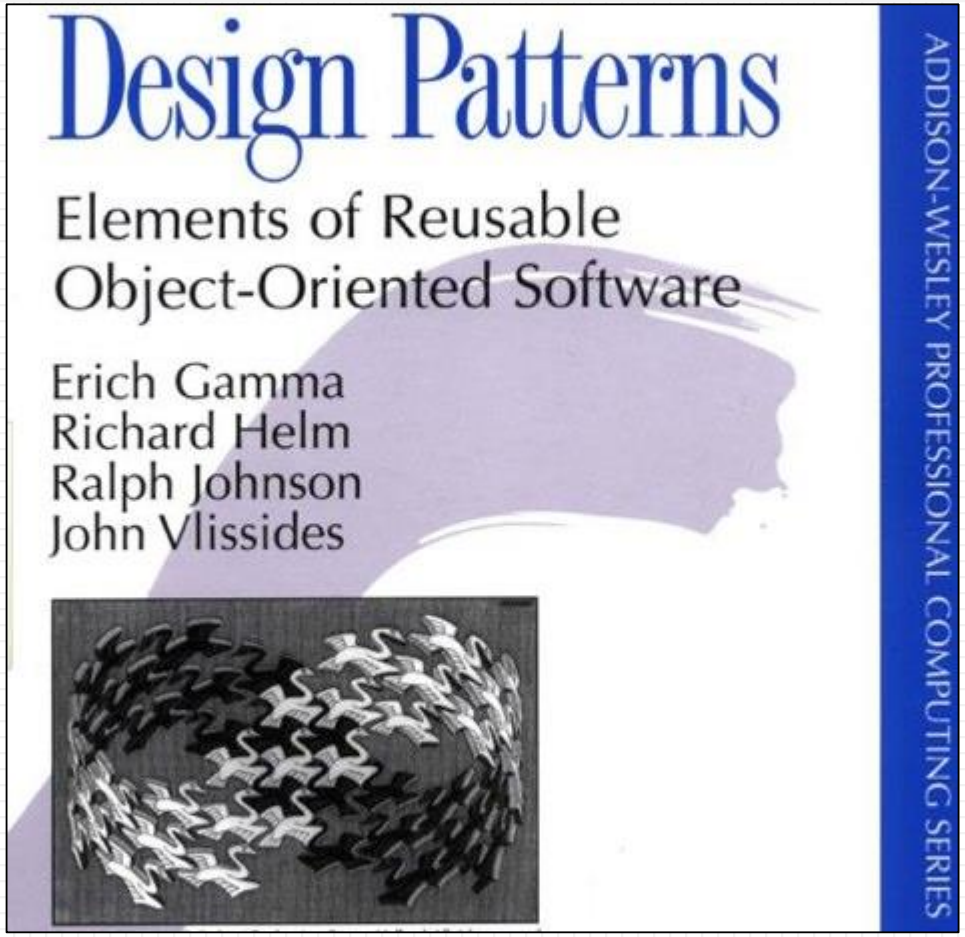


OnNext(T)

OnError()

OnCompleted()

That was the iterator pattern



Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

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 ▮) ?

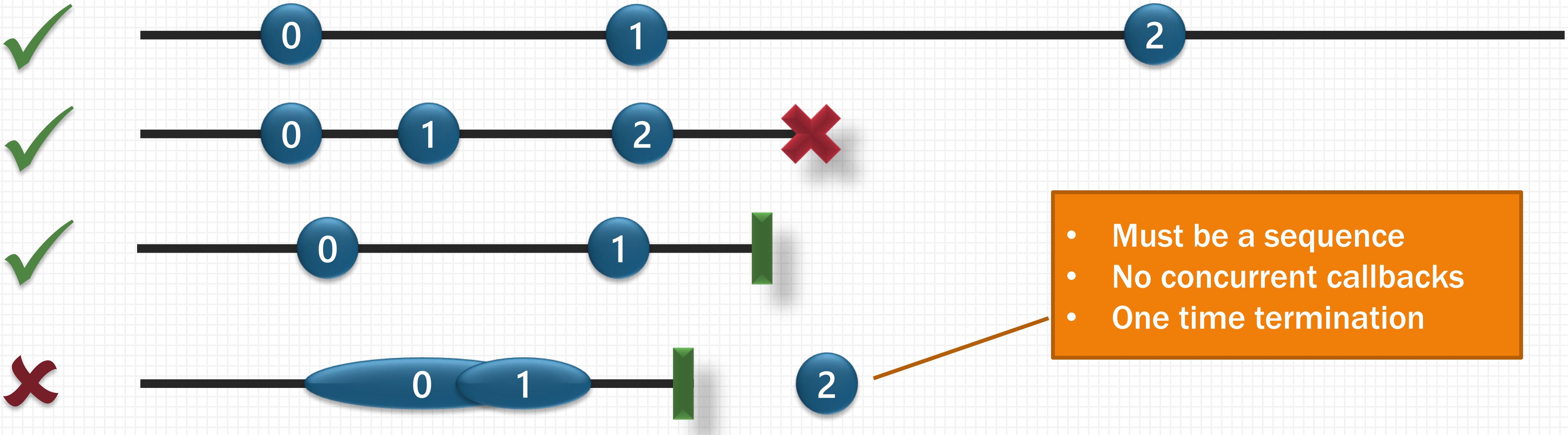
Zero or more values

E.g. events are ∞ sequences

Calls can fail

Resource management

Sequencing



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

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

* 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

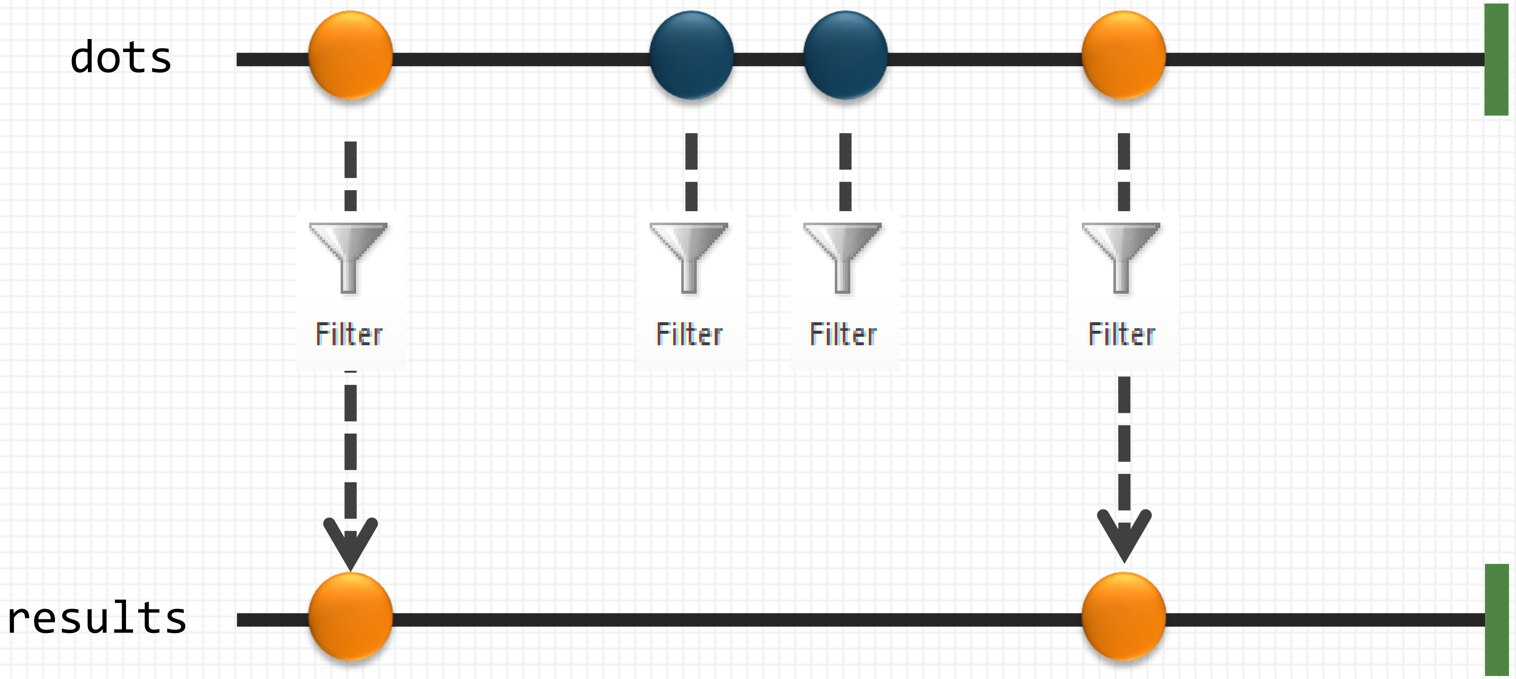
Reactive collections: the dual of iterable collections

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
    .Filter(q => q.Symbol == "FB")
    .Map(q => q.Quote)
    .ForEach(Console.WriteLine);
```

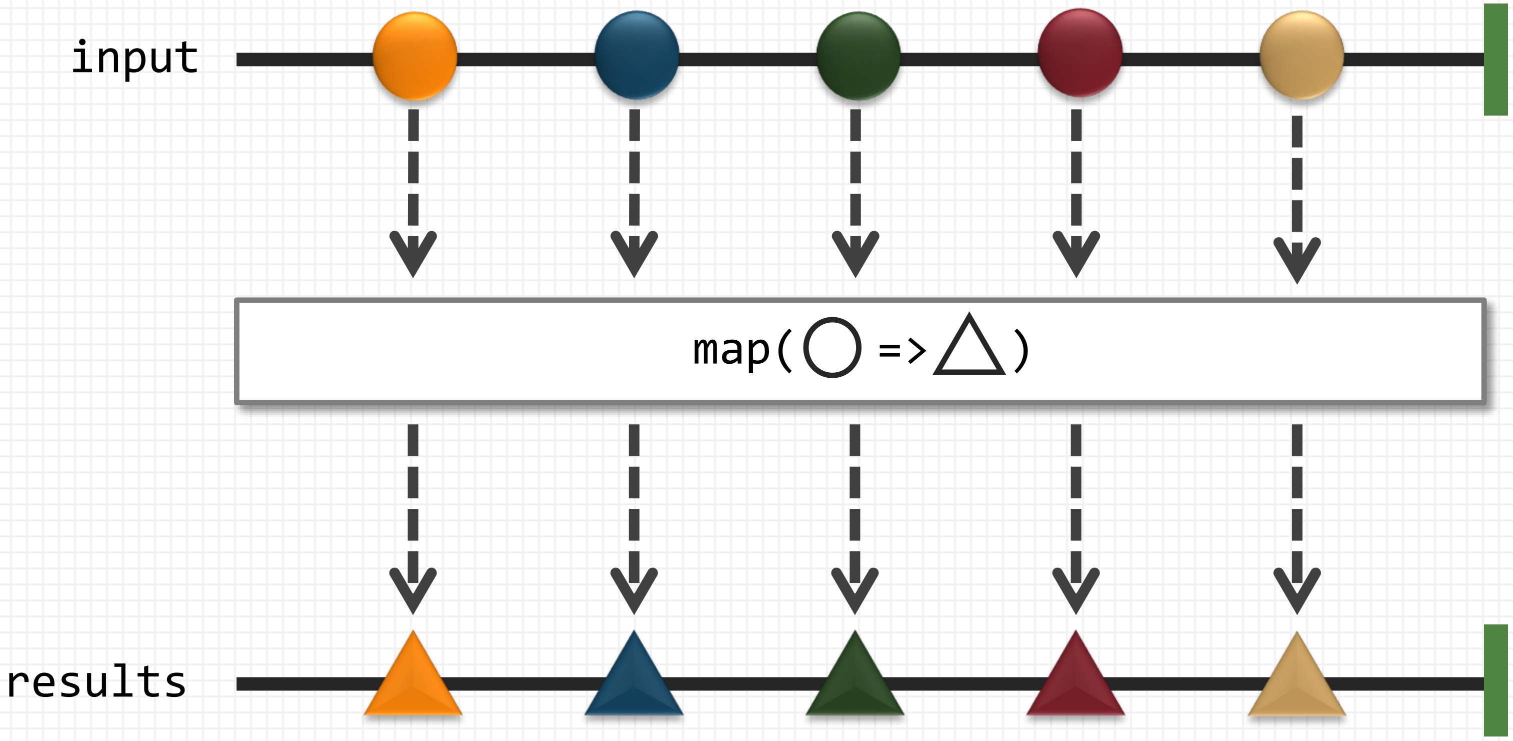
```
// IObservable<Stock>
// Incoming stock feed

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

Marble diagram: filter



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

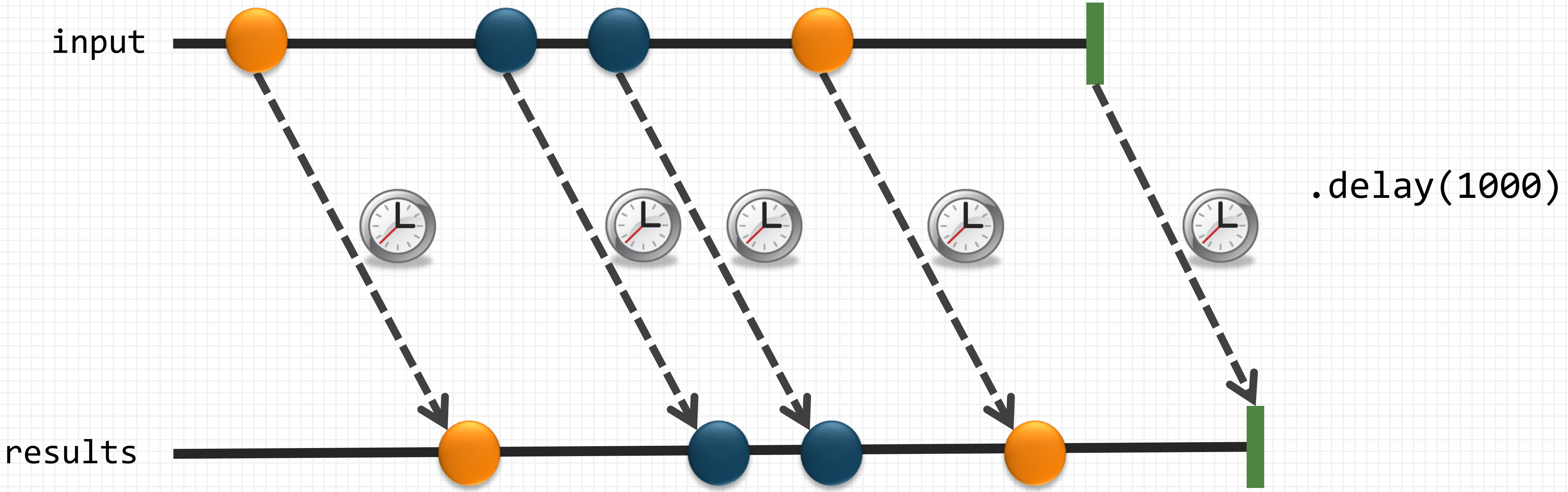
Marble diagram: map



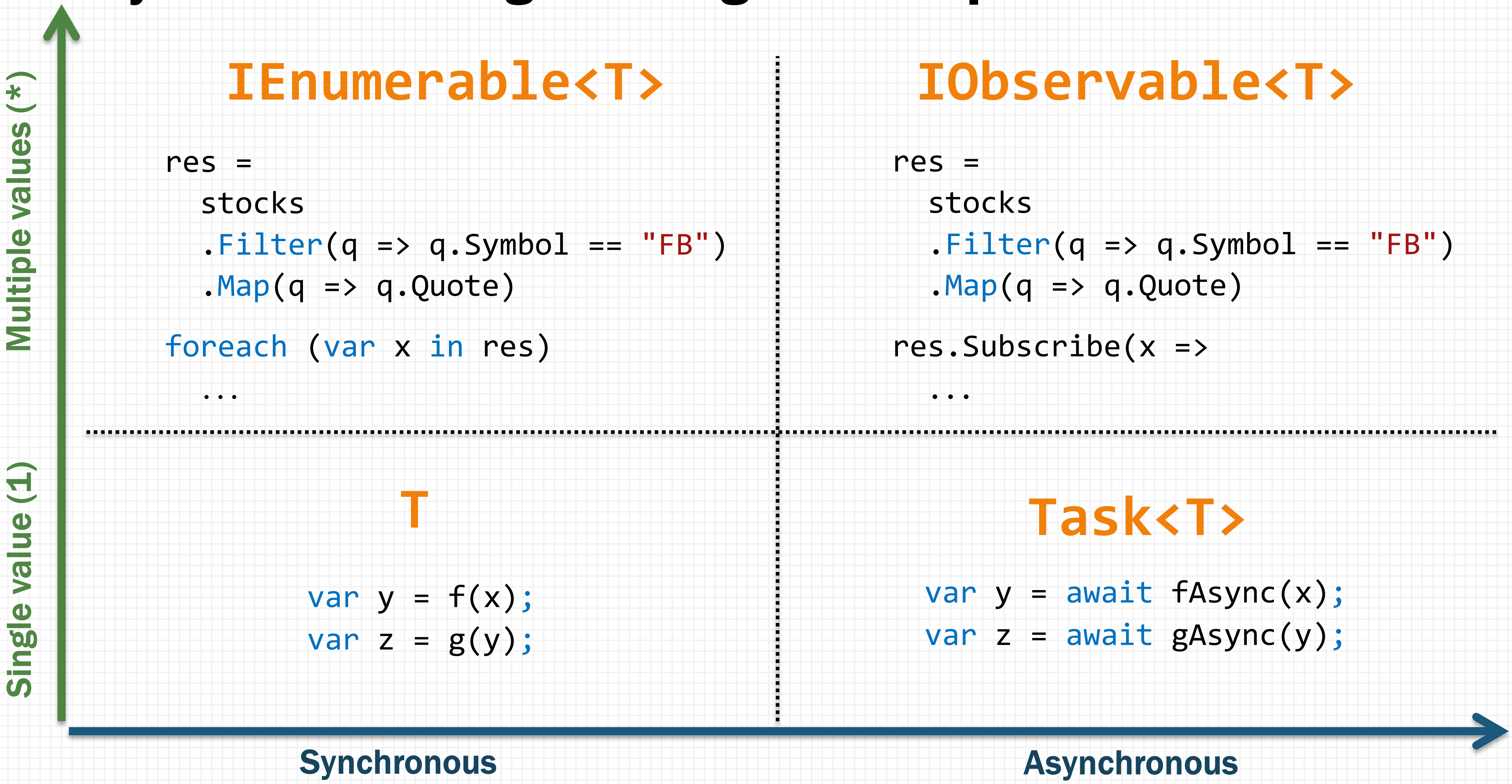
```
.map(function (item) {
  return transform(item);
})
```

Marble diagram: delay

Since Observables are asynchronous, they have a notion of time



The Asynchronous Programming Landscape



IEnumerable<T>

```
res =
  stocks
  .Filter(q => q.Symbol == "FB")
  .Map(q => q.Quote)
foreach (var x in res)
  ...
```

IObservable<T>

```
res =
  stocks
  .Filter(q => q.Symbol == "FB")
  .Map(q => q.Quote)
res.Subscribe(x =>
  ...
```

T

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

Task<T>

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

Demo:
Drag and Drop

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

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

Querying UI Events

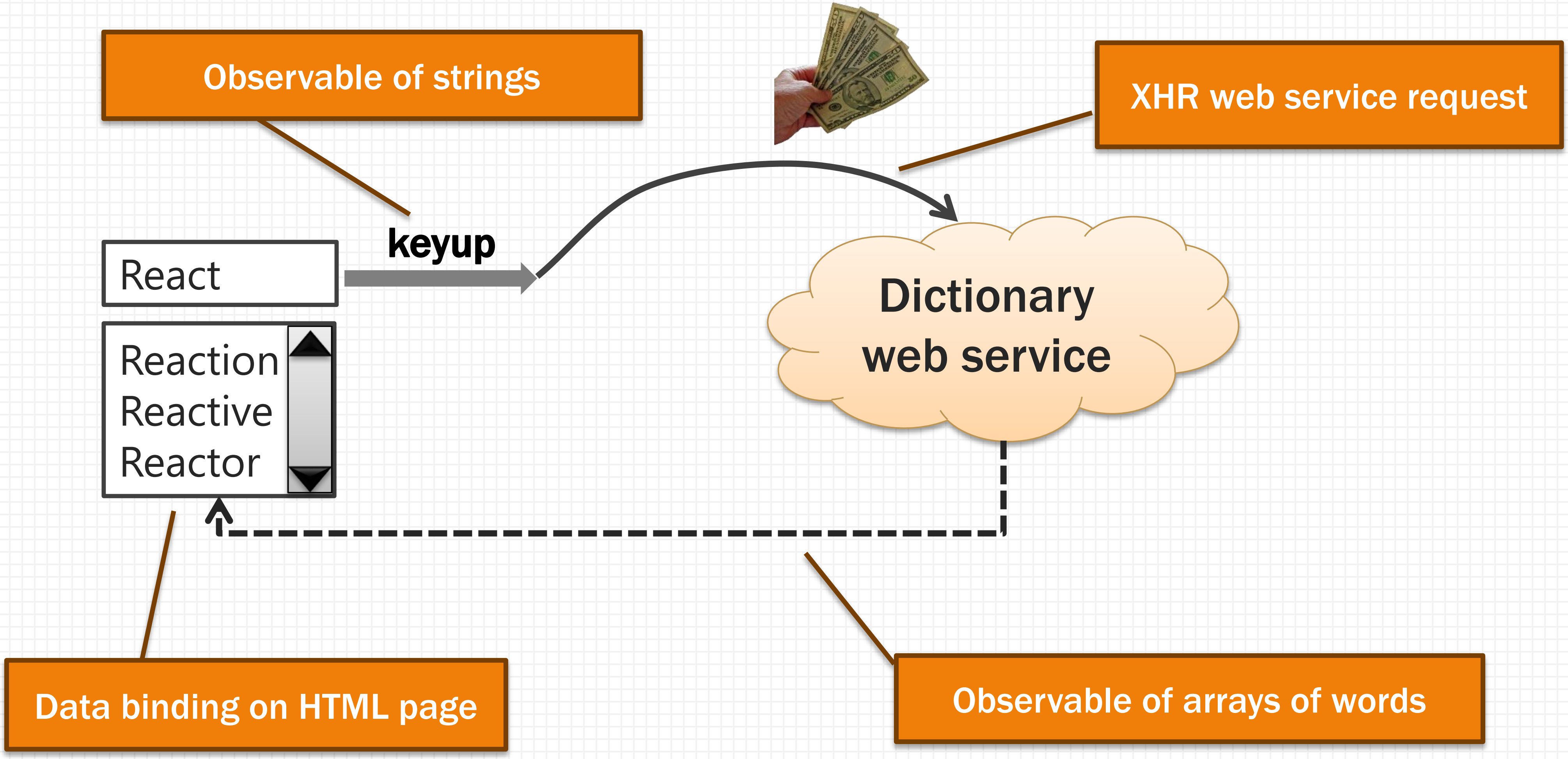
```
var mousedrag = mousedown.flatMap(function (md) {

    // 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

Composing Events and Promises



Observable of strings

XHR web service request

keyup

React

Reaction
Reactive
Reactor

Dictionary web service

Data binding on HTML page

Observable of arrays of words

Composing Events and Promises

```
var words = Rx.DOM.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

Demo:
Controlling a Kinect Sensor with Rx

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

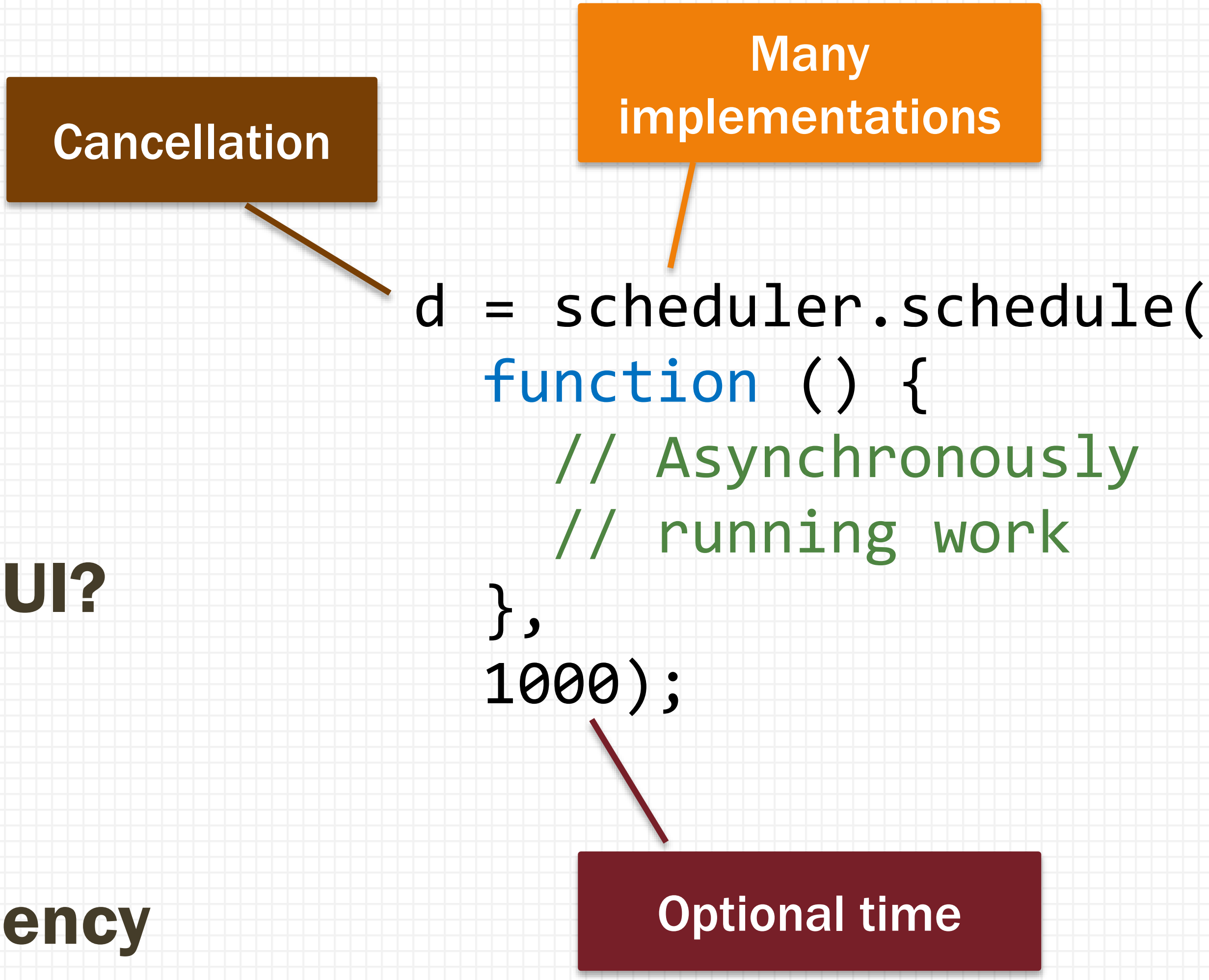
The Role of Schedulers

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



```
d = scheduler.schedule(
  function () {
    // Asynchronously
    // running work
  },
  1000);
```

Cancellation

Many implementations

Optional time

Testing concurrent code: made easy!

```
var scheduler = new TestScheduler();

var input = scheduler.createColdObservable(
    onNext(300, "Strange"),
    onNext(400, "Loop"),
    onCompleted(500));

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

results.messages.assertEqual(
    onNext(300, 7),
    onNext(400, 4),
    onCompleted(500));
```

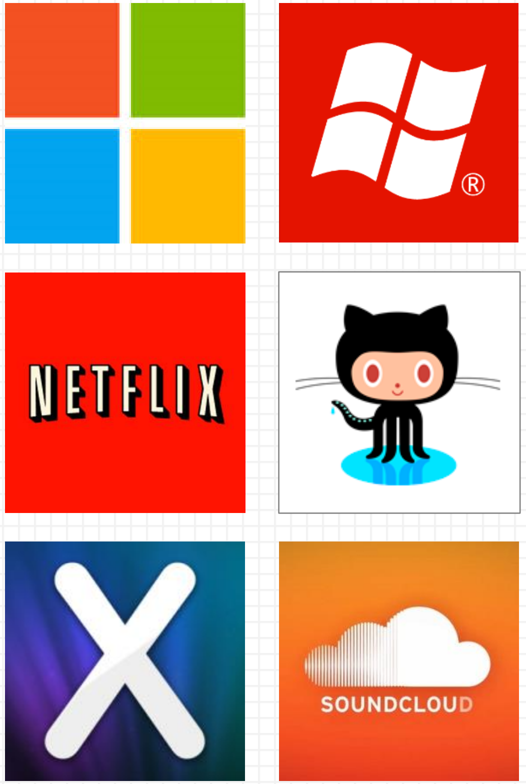
More about Rx

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)



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
rx.codeplex.com
github.com/Reactive-Extensions
```