Lecture 21

Reactive Programming with Rx

Duality between Push and Pull models, Iterable vs. Observable. Composing two asynchronous services.

Ras Bodik
Shaon Barman
Thibaud Hottelier

Hack Your Language!
CS164: Introduction to Programming Languages and Compilers, Spring 2012
UC Berkeley

One Leftover Topic from L20 on Arrowlets

Demo: Imagine you want to visualize an algorithm.

Example: bubble sort

http://www.cs.umd.edu/projects/PL/arrowlets/example-bubble-sort.xhtml

Ideally, the visualizing program looks like normal sort

- adding visualization commands and animation should not require changes to the algorithm structure

the hooks that perform visualization should be:

- unobtrusive (added into a few program points)
- parameterizable (change the style of visualization)

Example

```
var bubblesortA = function(x) {
    var list = x. list , i = x.i , j = x.j ;
    if ( j + 1 < i) {
        if ( list.get( j ) > list.get( j + 1)) {
            list.swap(j, j + 1);
        }
        return Repeat({ list : list, i : i , j : j + 1 });
    } else if ( i>0 ) {
        return Repeat({ list : list , i : i - 1, j : 0 });
    } else {
        return Done();
    }
}.AsyncA().repeat(100);
/* list is an object with methods get and swap */
bubblesortA.run({list:list , i : list . length , j : 0 });
```

Example: discussion

The good:

- visualization entirely hidden inside get() and swap()
- animation (one iteration per 100ms) is outside the algo

What can be improved:

- the Arrowlets program replaces the two nested bubble sort loops with a single recursion (the two Repeat's)
- ideal solution: give the programmer 'for' loops, which are desugared to arrows composed with Repeat's

Programming with RxJS

RxJS

RxJS is a Microsoft DSL for reactive programming

- implemented as a JavaScript library
- there is also a .NET version, called Rx, statically typed

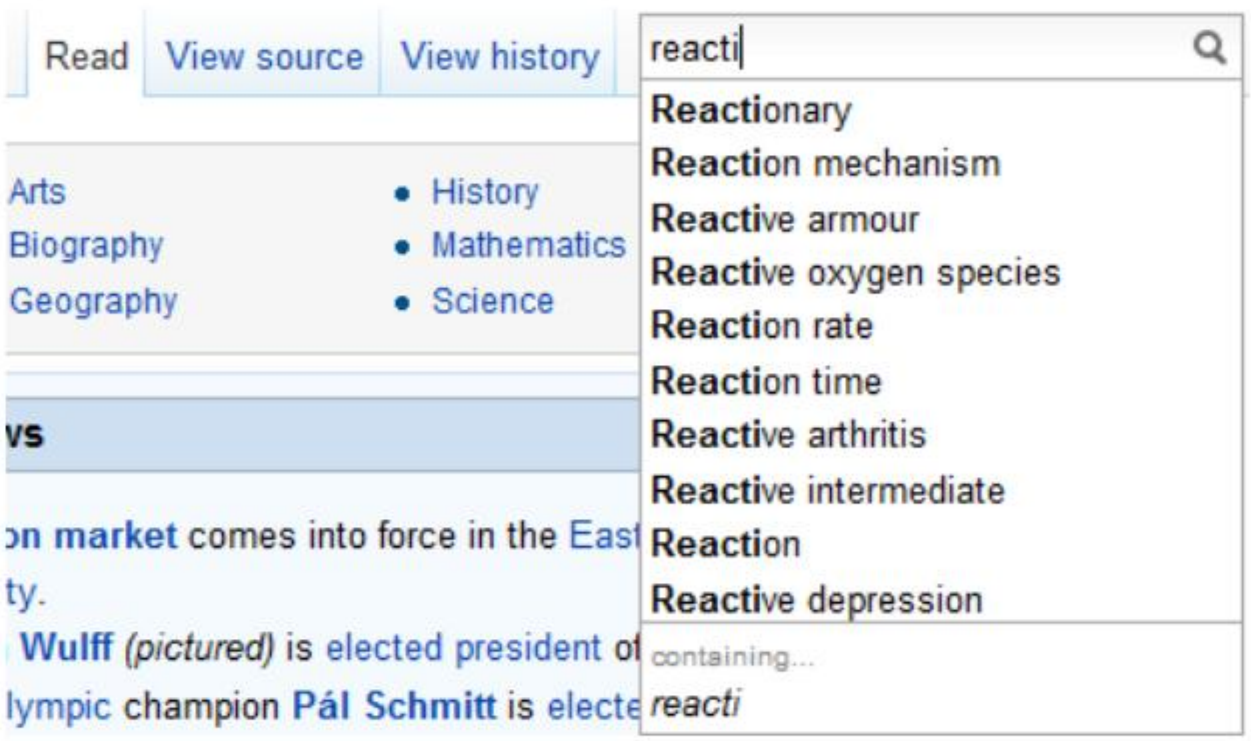
Author is Erik Meijer and team

Erik: Got waaaaaaaaaaaaaaaay to excited while hosting the Lang .Next panel so I bought a wireless Pulse Oximeter (those little things they put on your finger in the the ER) to track my heart rate and oxygen level in real time while dancing around.

Hooked up via Rx of course.

Our running example

Let's implement instant search for Wikipedia:



Reading

These slides are based on http://bit.ly/cAxKPk

This is your assigned reading.

Push-pull duality

Recall Lua lazy iterators

were these push or pull programming model?

In .NET lingo, Iterable vs. Observable:

Iterable: sequence I iterate over and pull elems

Observable: sequence that notifies when a new value is added and pushes the value to observer (listener)

These two are dual

difference in who is the master, and who is the slave

Basics: Observable and Observer

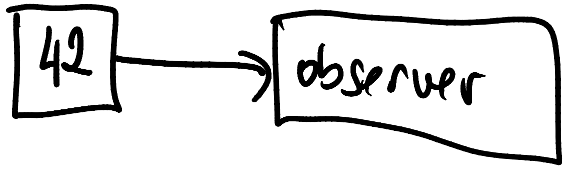
1) Observable and Observer objects

Data source:

```
   var answer = Rx.Observable.Return(42);
```

Listener:

```
   var observer = Rx.Observer.Create(
       function (x) {
           document.write("The answer is " + x);
       }
   );
```



Connecting the two:

```
   answer.Subscribe(observer);
```

The same code packaged in a web page

```
<script type="text/javascript">
     function iExploreRx() {
          var answer = Rx.Observable.Return(42);
          var observer = Rx.Observer.Create(
               function (x) {
                    document.write("The answer is " + x);
               }
          );
          answer.Subscribe(observer);
     }
</script>
<body>
   <button onclick="javascript:iExploreRx()">Tell me
   the answer</button>
</body>
```

2) Observable sequences

First, let’s set up the listener:

```
var source = null;   // we’ll try various sequences here

var subscription = source.Subscribe(
    function (next) {
       $("<p/>").html("OnNext: "+next).appendTo("#content");
    },
    function (exn) {
       $("<p/>").html("OnError: "+exn).appendTo("#content");
    },
    function () {
       $("<p/>").html("OnCompleted").appendTo("#content");
    });
```

Empty sequence

```
var source = Rx.Observable.Empty();
```

Produces the output

```
OnCompleted
```

Sequence with a terminating notification

```
var source = Rx.Observable.Throw("Oops!");
```

Running this code produces the following output:

```
OnError: Oops
```

Single-element sequence

```
var source = Rx.Observable.Return(42);
```

Running this code produces the following output:

```
OnNext: 42
OnCompleted
```

We are now done with trivial sequences

```
var source = Rx.Observable.Range(5, 3);
```

Running this code produces the following output:

```
OnNext: 5
OnNext: 6
OnNext: 7
OnCompleted
```

A for-loop like sequence generator

```
var source = Rx.Observable.GenerateWithTime(
    0, // initial value of iterator variable
    function(i) { return i < 5; },  // test
    function(i) { return i + 1; },  // incr
    function(i) { return i * i; },  // value
    function(i) { return i * 1000; }
);
```

Last function computes how many ms to wait between generated values (here, 1, 2, 3, … seconds)

Events add Asynchrony

3) Events as data sources

In jQuery

```
$(document).ready(function () {
     $(document).mousemove(function (event) {
          $("<p/>").text("X: " + event.pageX+" Y: " + event.pageY)
                   .appendTo("#content");
     });
 });
```

In Rx

```
$(document).ready(function () {
   $(document).toObservable("mousemove").Subscribe(function(event){
       $("<p/>").text("X: " + event.pageX+" Y: " + event.pageY)
                .appendTo("#content");
   });
});
```

4) Projection and filtering

In event-handler programming, you'd write:

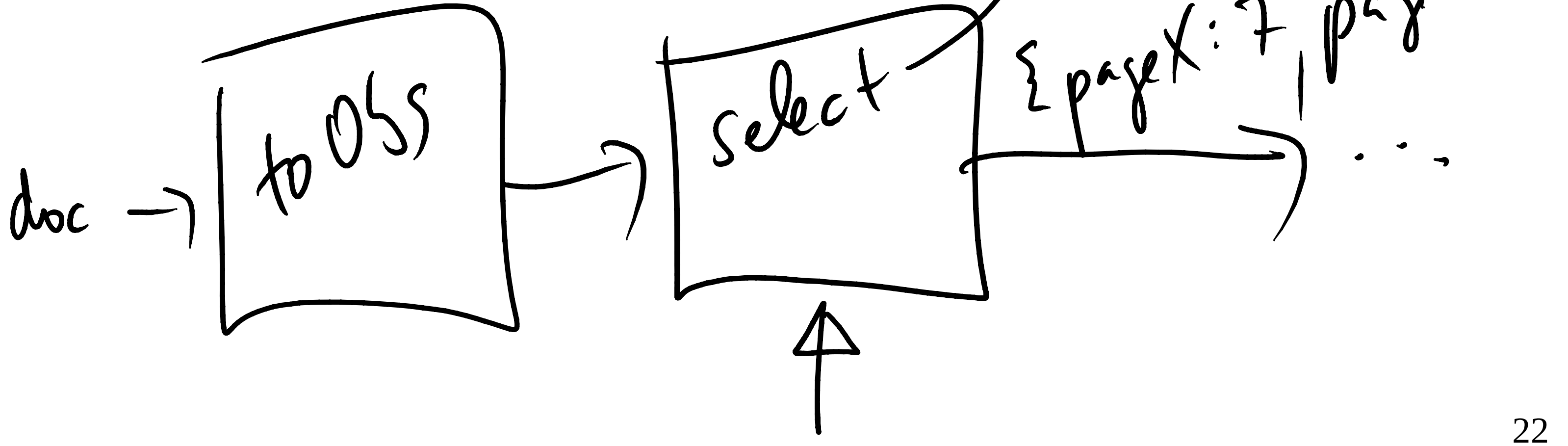
```
function handleMouseMove(event) {   // FILTER some events
        if (event.pageX === event.pageY) {
            // Only respond to events for mouse moves
            // over the first bisector of the page.
        }
}

function handleKeyUp(event) {  // PROJECT the event’s val
        var text = $(event.target).val();
        // And now we can forget about the rest
        // of the event object’s data...
}
```

The same in Rx

```
var moves = $(document).toObservable("mousemove")
      .Select(function(event) {
        return { pageX : event.pageX, pageY : event.pageY };
      });

var input = $("#textbox").toObservable("keyup")
      .Select(function(event) {
            return $(event.target).val();
      });
```



Now we can subscribe to moves and input

```
var movesSubscription = moves.Subscribe(function (pos) {
    $("<p/>").text("X: " + pos.pageX + " Y: " + pos.pageY)
             .appendTo("#content");
});

var inputSubscription = input.Subscribe(function (text) {
        $("<p/>").text("User wrote: " + text)
                 .appendTo("#content");
});
```

Filtering

```
var overFirstBisector = moves
        .Where(function(pos) {
            return pos.pageX === pos.pageY;
        });

var movesSubscription =
     overFirstBisector.Subscribe(function(pos){
        $("<p/>")
           .text("Mouse at: "+pos.pageX+","+pos.pageY)
           .appendTo("#content");
     });
```

Summary: Composition

We were able to compose observable sequences thanks to the first-class nature of sequences

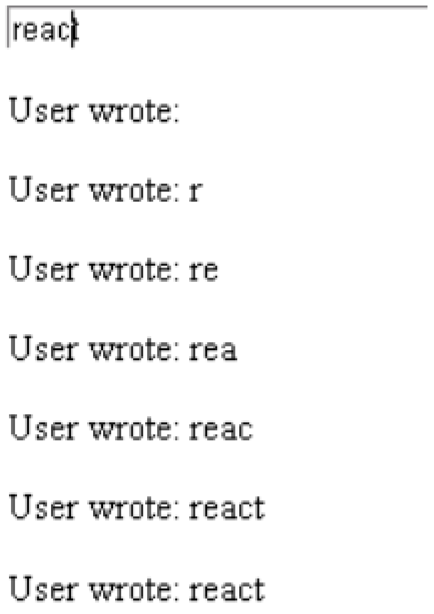
“First-class” means sequences are values which can be stores, passed, and we can define operators for them, such as the filter operator Where.

Manipulating Sequences

Removing duplicate events

DOM raises a keyup event even when the text in textbox does not change. For example, if you select a letter and change it to the same letter. This causes duplicate events:

r e a c t|i v e → (SHIFT-LEFT ARROW) r e a c t i v e → (type t) r e a c t|i v e



```
react

User wrote:

User wrote: r

User wrote: re

User wrote: rea

User wrote: reac

User wrote: react

User wrote: react
```

Rx solution

```
var input = $("#textbox")
  .toObservable("keyup")
  .Select(function (event) { return $(event.target).val();})
  .DistinctUntilChanged();

var inputSubscription = input.Subscribe(function (text) {
        $("<p/>").text("User wrote: " + text)
                 .appendTo("#content");
});
```

Throttle

```
var input = $("#textbox").toObservable("keyup")
        .Select(function (event) {
              return $(event.target).val();
        })
        .Throttle(1000)
        .DistinctUntilChanged();

    var inputSubscription = input.Subscribe(function (text) {
        $("<p/>").text("User wrote: " + text)
                 .appendTo("#content");
    });
```

Throttle

A timer is used to let an incoming message age for the specified duration, after which it can be propagated further on.

If during this timeframe another message comes in, the original message gets dropped on the floor and substituted for the new one that effectively also resets the timer.

Can be used to suppress the number of requests sent to a web service.

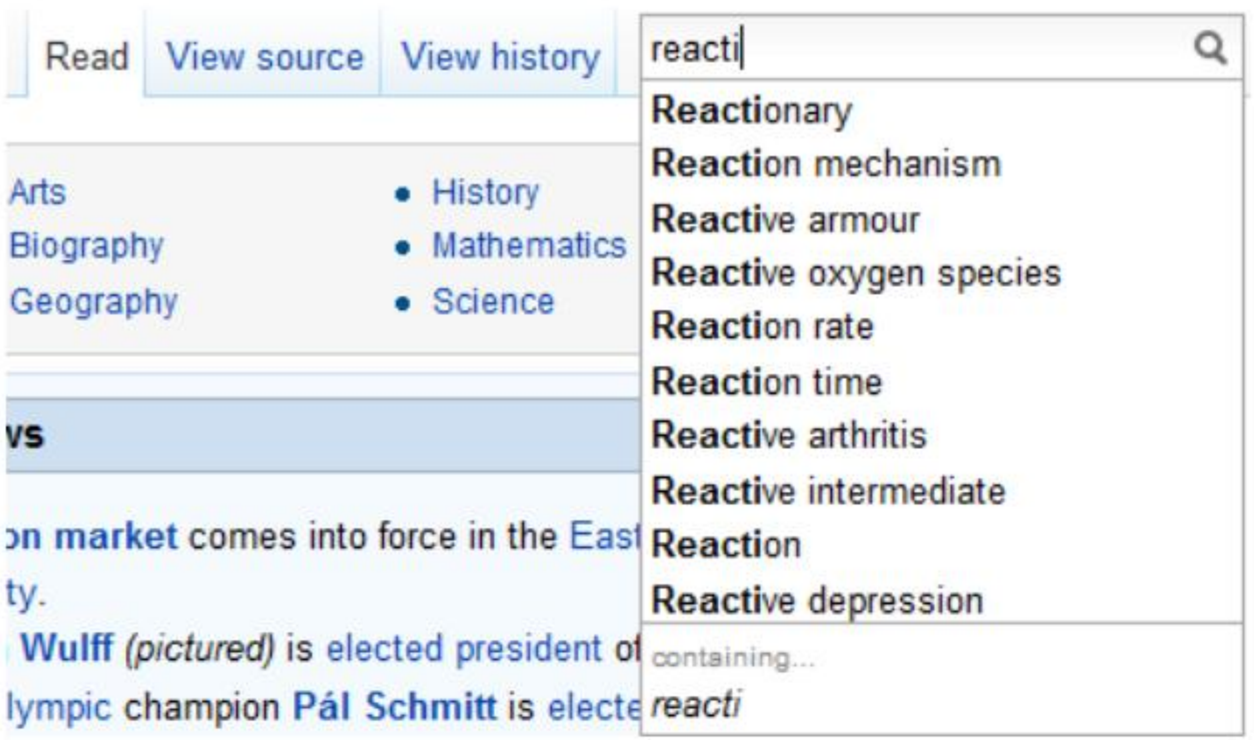
Do and Timestamp

```
var input = $("#textbox").toObservable("keyup")
   .Select(function (event) { return $(event.target).val(); })
   .Timestamp()
   .Do(function(inp) {
         var text = "I: " + inp.Timestamp + "-" + inp.Value;
         $("<p/>").text(text).appendTo("#content");
    })
   .RemoveTimestamp()
   .Throttle(1000)
   .Timestamp()
   .Do(function(inp) {
         var text = "T: " + inp.Timestamp + "-" + inp.Value;
         $("<p/>").text(text).appendTo("#content");
   })
   .RemoveTimestamp()
   .DistinctUntilChanged();
```

Asynchrony with the Server (out-of-order messages)

6) Asynchrony with the server

Let's implement instant search for Wikipedia:



on market comes into force in the East
ty.
Wulff (pictured) is elected president of
lympic champion Pál Schmitt is electe

reacti

- Reactionary
- Reaction mechanism
- Reactive armour
- Reactive oxygen species
- Reaction rate
- Reaction time
- Reactive arthritis
- Reactive intermediate
- Reaction
- Reactive depression

containing...
reacti

Interface to Wikipedia in JS

```
$.ajax({ url: "http://en.wikipedia.org/w/api.php",
         dataType: "jsonp",
         data: { action: "opensearch",
                 search: "react",
                 format: "json" },
         success: function (data, textStatus, xhr) {
             $("#results").empty();
             $.each(data[1], function (_, result) {
                 $("#results").append("<li>"+result+"</li>");
             });
         },
         error: function (xhr, textStatus, errorThrown) {
             $("#error").text(errorThrown);
         }
});
```

Array<String>

The same in Rx

```
$.ajaxAsObservable(
        { url: "http://en.wikipedia.org/w/api.php",
          dataType: "jsonp",
          data: { action: "opensearch",
                  search: "react",
                  format: "json"
                }
        });
```

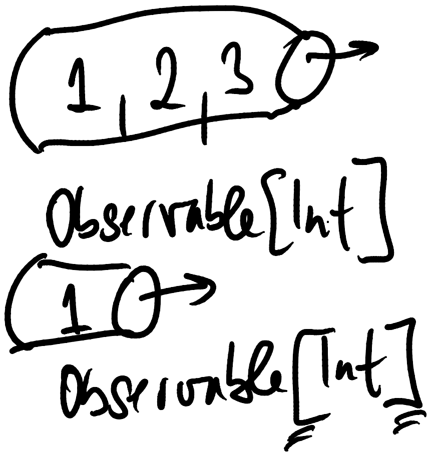
Array[Int]

From a term to observable with search results

A function that creates an observable for a search term:

Search string

```
function searchWikipedia(term) {
        return $.ajaxAsObservable(
            { url: "http://en.wikipedia.org/w/api.php",
              dataType: "jsonp",
              data: { action: "opensearch",
                      search: term,
                      format: "json"
                    }
            })
            .Select(function (d) { return d.data[1]; });
}
```



String $\rightarrow$ Observable[Array[String]]

Print the result of the search

```
var searchObservable = searchWikipedia("react");

var searchSubscription = searchObservable.Subscribe(
        function (results) {
            $("#results").empty();
            $.each(results, function (_, result) {
               $("#results").append("<li>"+result+"</li>");
            });
        },
        function (exn) {
            $("#error").text(error);
        }
);
```

7) Now the complete app

```
<body>
  <input id="searchInput" size="100" type="text"/>
  <ul id="results" />
  <p id="error" />
</body>
```

The header

```
 <head>
    <title>Wikipedia Lookup</title>
    <script type="text/javascript" src="Scripts/jquery-
1.4.1.min.js"></script>
    <script type="text/javascript" src="Scripts/rx.js"></script>
    <script type="text/javascript" src="Scripts/rx.jquery.js">
</script>
```

Wikipedia interface (just like before)

```
function searchWikipedia(term) {
        return $.ajaxAsObservable(
            { url: "http://en.wikipedia.org/w/api.php",
              dataType: "jsonp",
              data: { action: "opensearch",
                      search: term,
                      format: "json"
                    }
            })
            .Select(function (d) { return d.data[1];
});
}
```

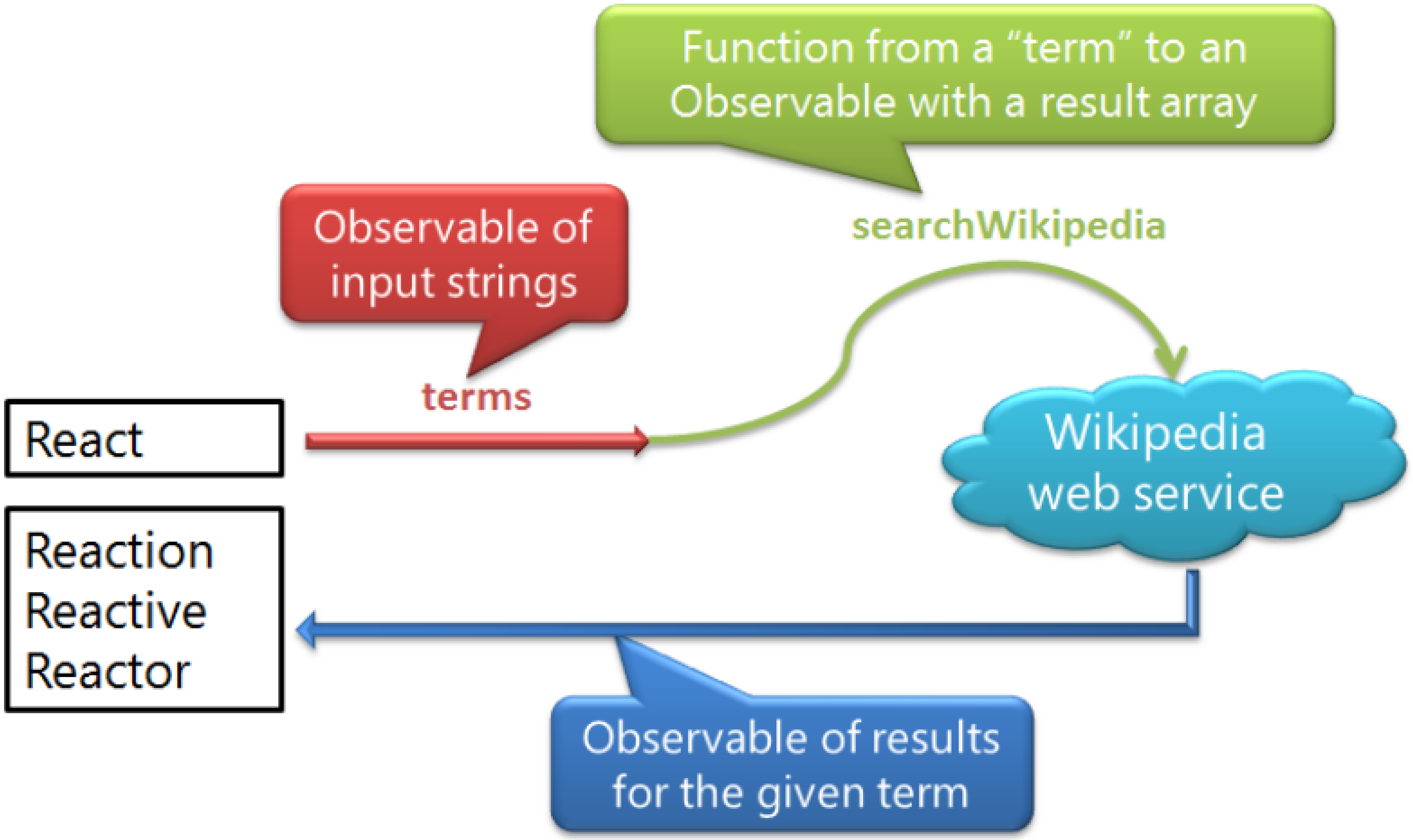
The input, throttled

```
$(document).ready(function () {
   var terms = $("#searchInput").toObservable("keyup")
   .Select(function (event) {
                    return $(event.target).val();
           })
   .Throttle(250);

   // Time to compose stuff here...

 });
```

We want to achieve this composition


Function from a "term" to an Observable with a result array
Observable of input strings
searchWikipedia
terms
React
Wikipedia web service
Reaction
Reactive
Reactor
Observable of results for the given term

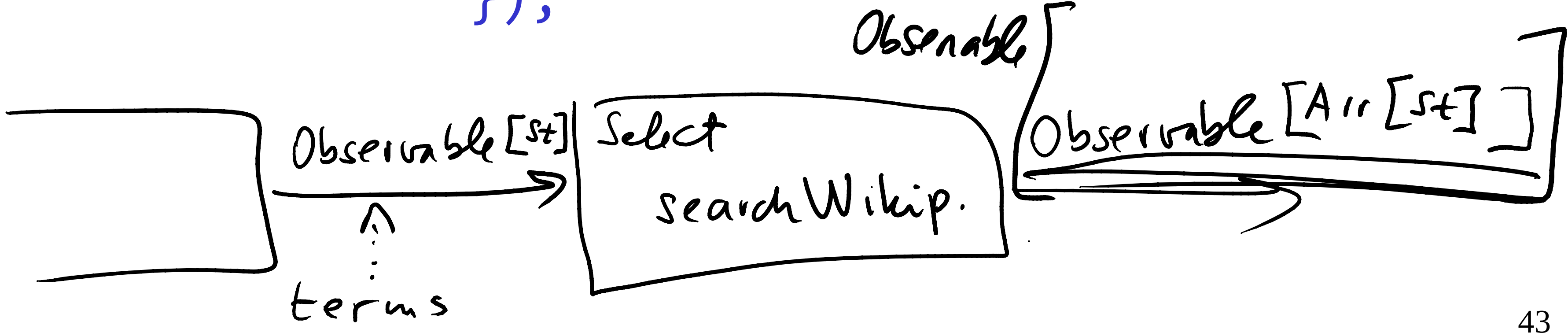
$$\begin{tikzcd}\text{Observable[Int]} \arrow[r] & \boxed{\text{Select(f)}} \arrow[r] & \text{Observable[String]}\end{tikzcd}$$

$f :: \text{Int} \rightarrow \text{String}$

How to compose our two components?

1) Observable sequence of strings
2) Function from a string to an observable sequence that contains an array with the results

```
                 Select
var searchObservable =
   terms.WhatComesHere?(function (term) {
                 return searchWikipedia(term);
              });
```



Why is this solution incorrect?

input type:

    Observable[string]

type of searchWikipedia:

    string → Observable[Array[string]]

hence Select’s result type:

    Observable[Observable[Array[string]]

while we want

    Observable[Array[string]]

hence we need to flatten the inner observable sequences

Correct solution

SelectMany: projects each value of an observable sequence to an observable sequence and flattens the resulting observable sequences into one sequence.

```
    var searchObservable =
            terms
            .SelectMany(function (term) {
                 return searchWikipedia(term);
             });
```

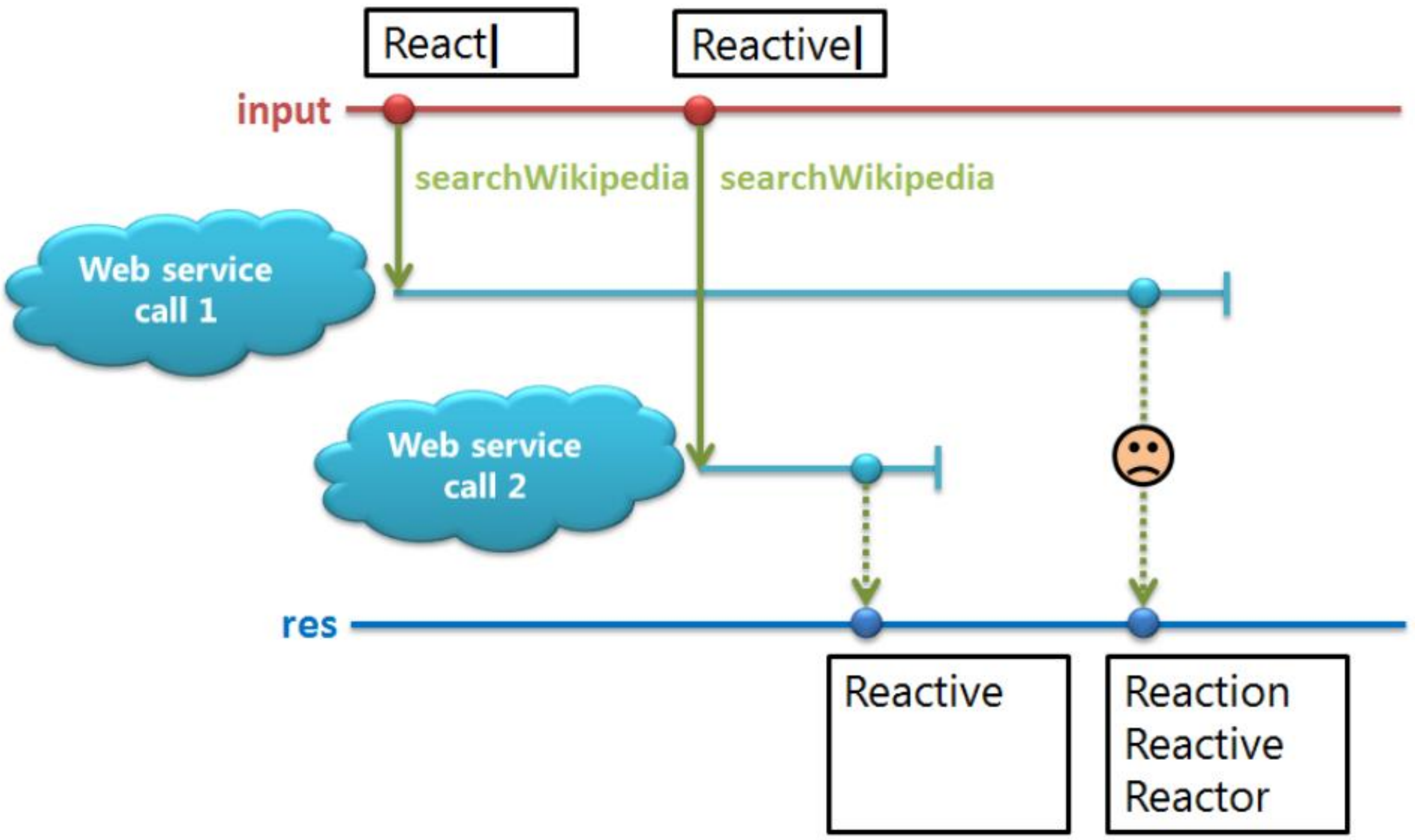
same as

terms.SelectMany(searchWikipedia)

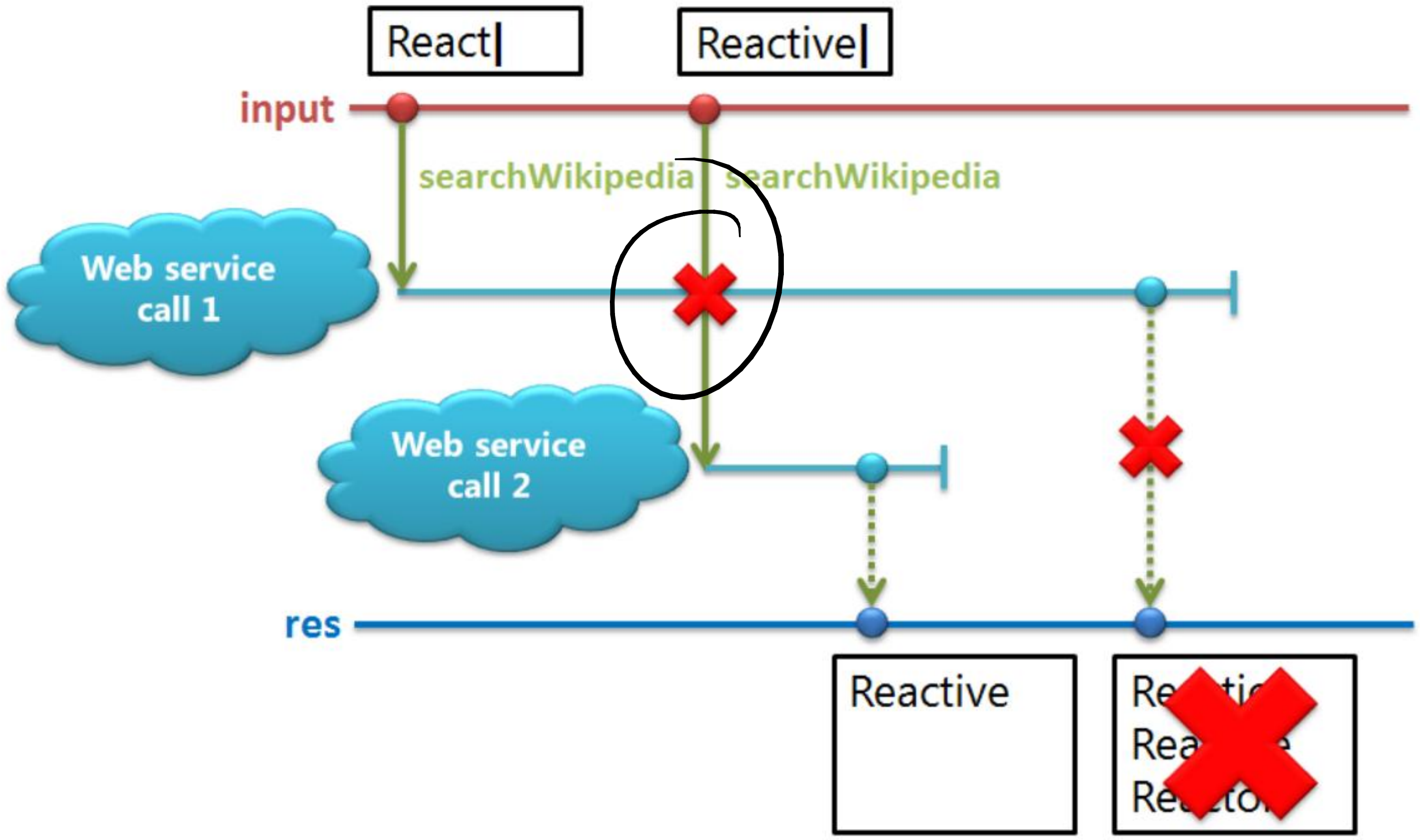
Bind results to a UI

As in Section 6.

Out of order sequences



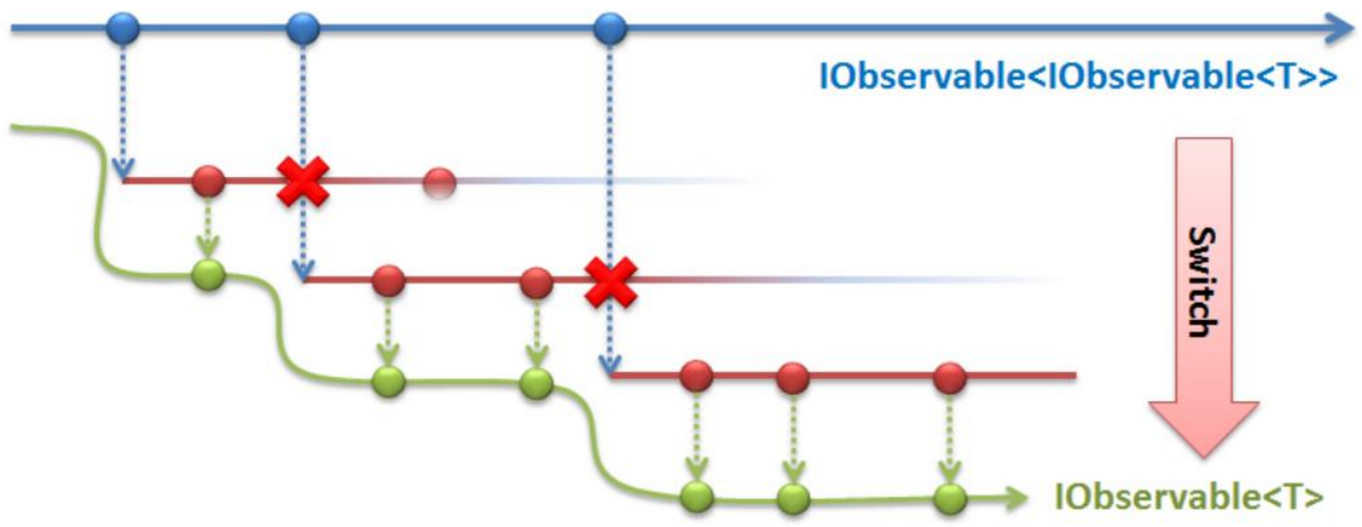
Must suppress the older request



.Select().Switch()

```
var searchObservable =
     terms
     .Select(searchWikipedia);
     // Observable[Observable[…]]
     .Switch();
     // Switch subscribes to latest sequence
```

Semantics of Switch



Summary

From non-composable event handlers to composable observable sequences (streams)

Read the full tutorial, which contains one more interesting section, at http://bit.ly/cAxKPk

- start reading on page 7 (Exercise 2)