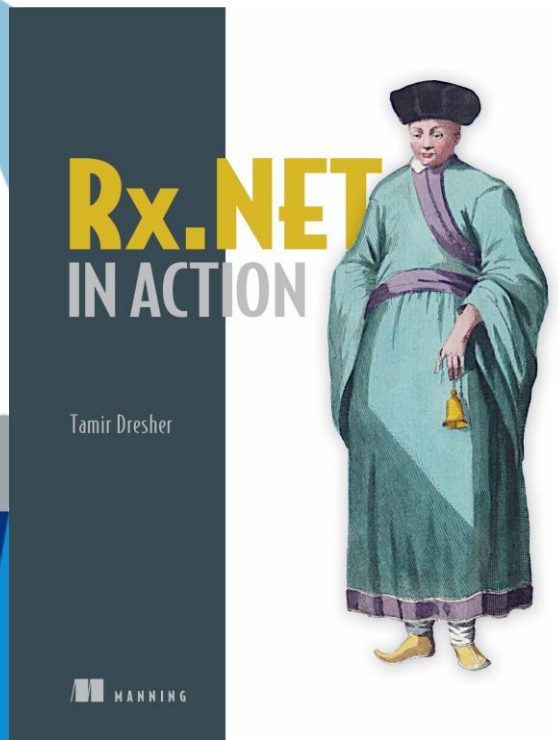
Reactive Extensions (Rx) 101

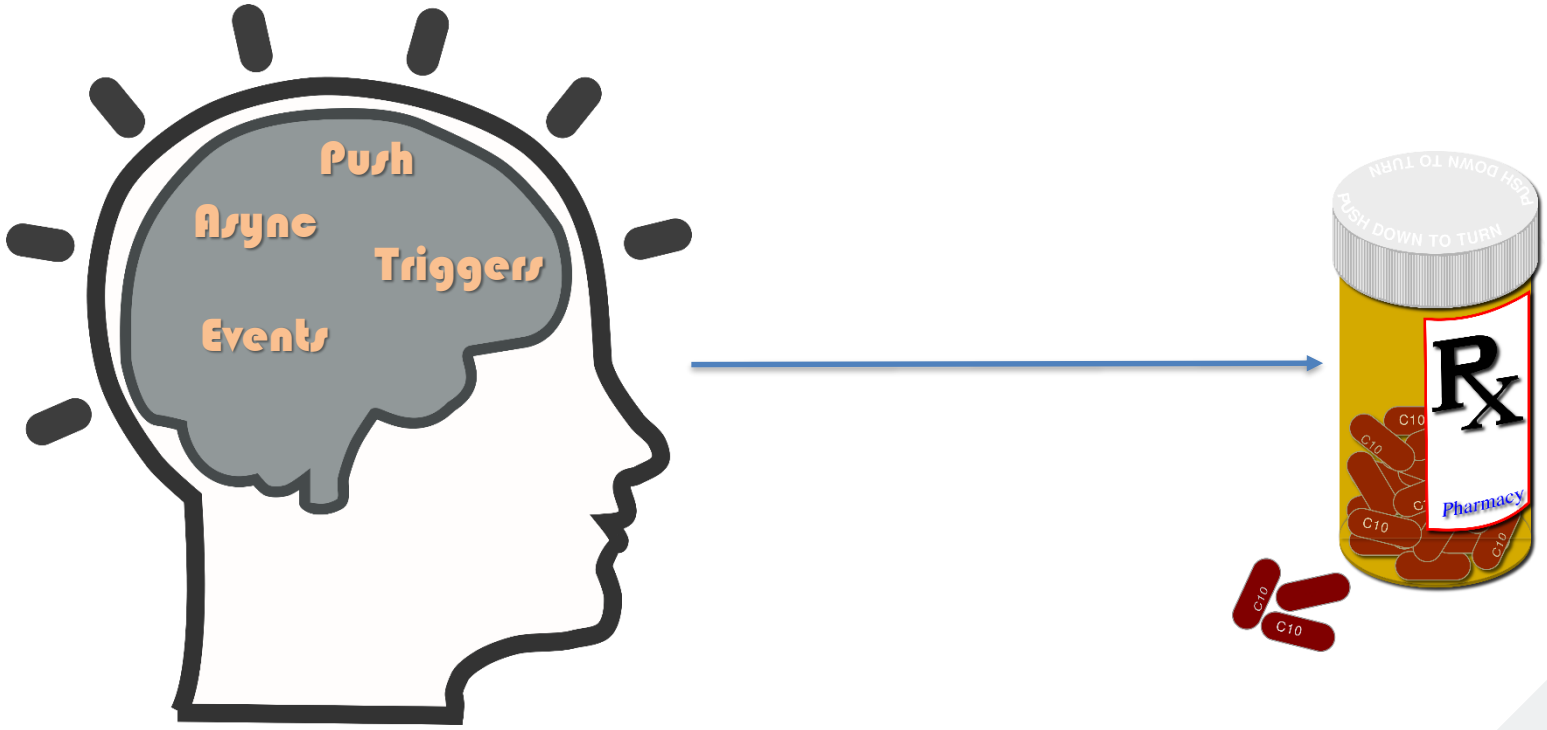
Tamir Dresher (@tamir_dresher)

Cloud Division Leader @ CodeValue

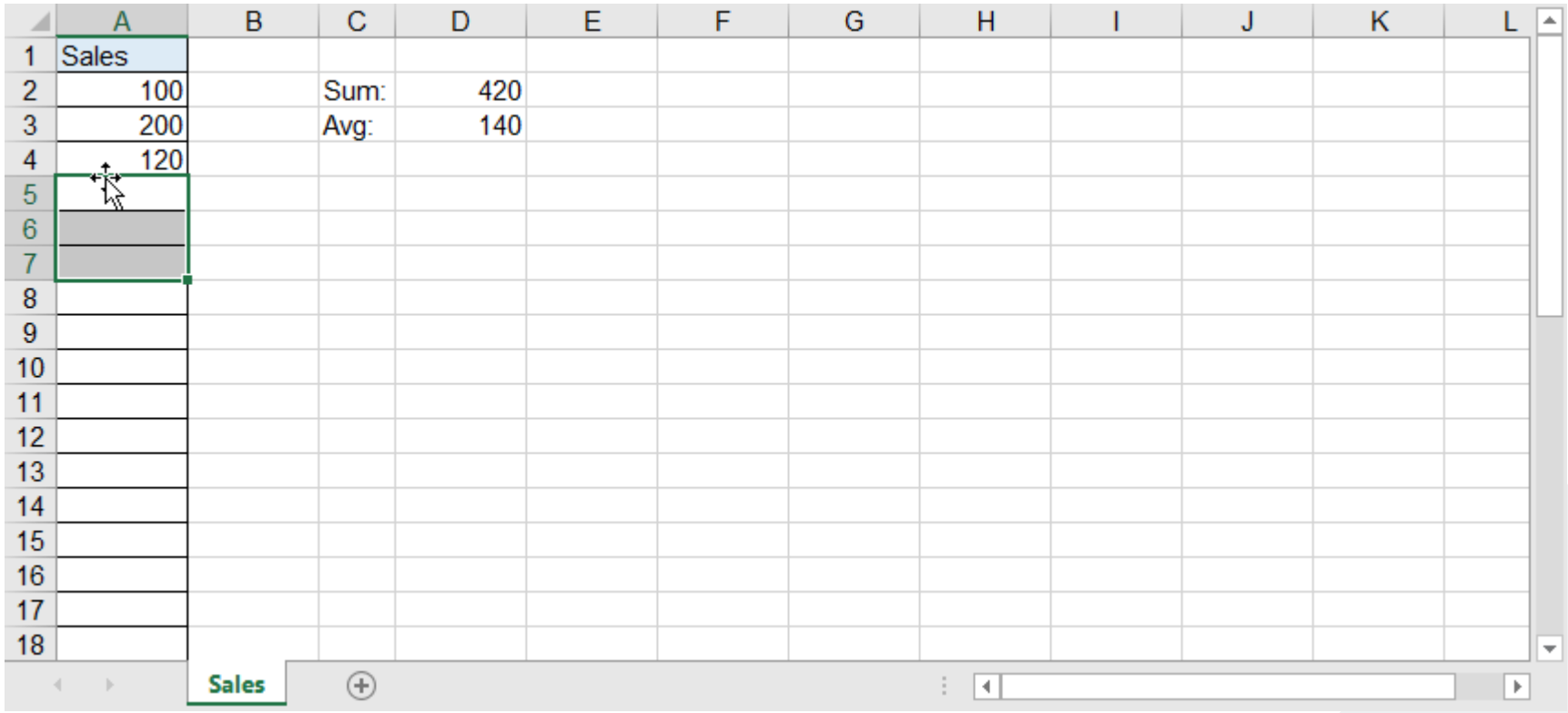


Reactive Extensions

Your headache relief pill to Asynchronous Event based applications



Reacting to changes

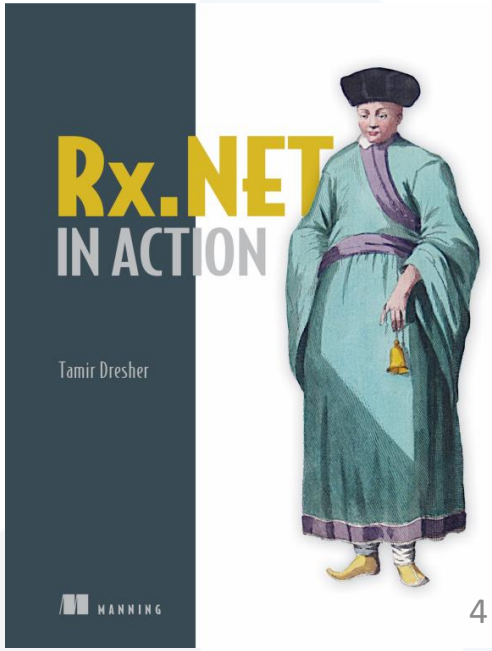


About Me

@tamir_dresher
tamirdr@codevalue.net
http://www.TamirDresher.com.

OzCode Your Road to Magical Debugging

- Cloud Division Leader @ CodeValue, Microsoft MVP
- Software architect, consultant and instructor
- Software Engineering Lecturer @ Ruppin Academic Center
- Author of *Rx.NET in Action* (Manning)



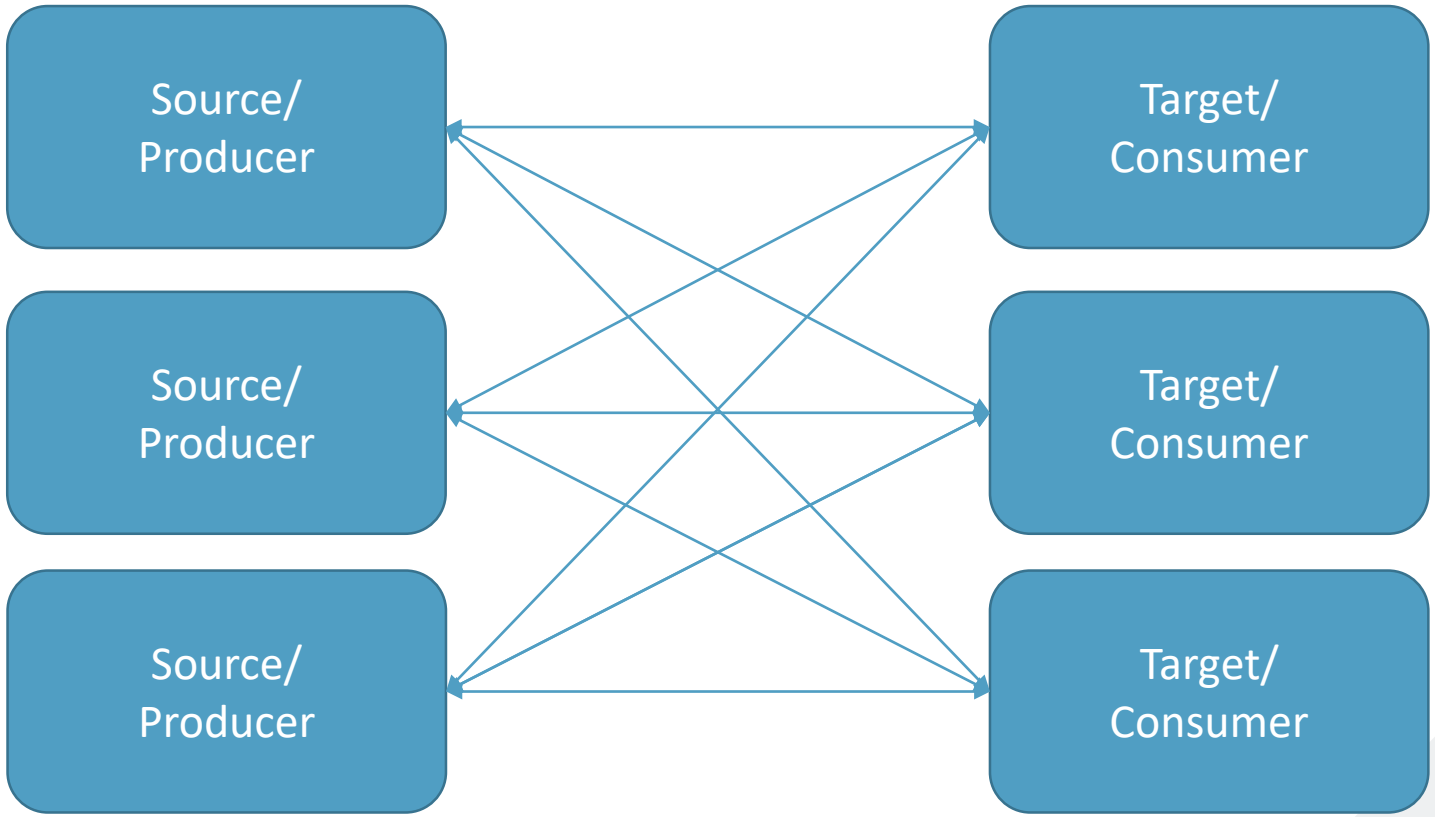
MVP Microsoft® Most Valuable Professional

Agenda

- Rx building blocks – Observables and Observers
- Creating Observables
- Rx queries with Rx operators
- Managing Concurrency with Rx Schedulers

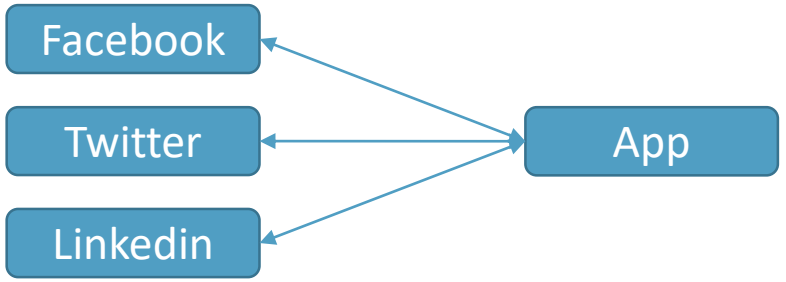
Producer-Consumer

Producers-Consumers



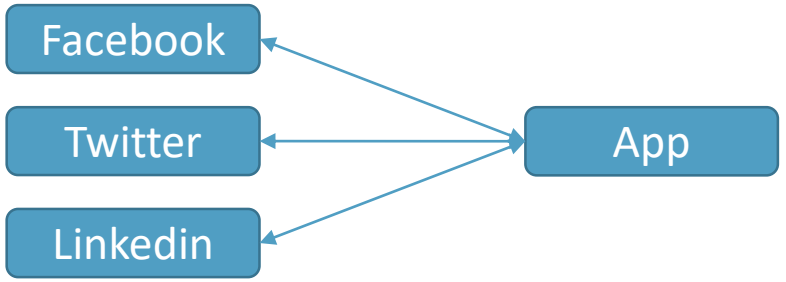
Pull Model

```
IEnumerable<Message> LoadMessages(string hashtag)
{
    var statuses = facebook.Search(hashtag);
    var tweets = twitter.Search(hashtag);
    var updates = linkedin.Search(hashtag);
    return statuses.Concat(tweets).Concat(updates);
}
```



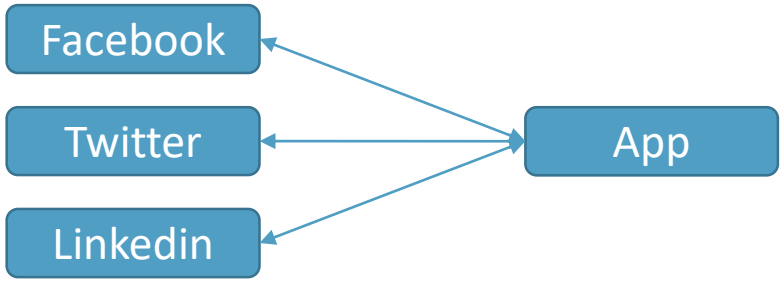
Pull Model

```
async Task<IEnumerable<Message>> LoadMessagesAsync(string hashtag)
{
    var statuses = await facebook.Search(hashtag);
    var tweets = await twitter.Search(hashtag);
    var updates = await linkedin.Search(hashtag);
    return statuses.Concat(tweets).Concat(updates);
}
```



Pull Model

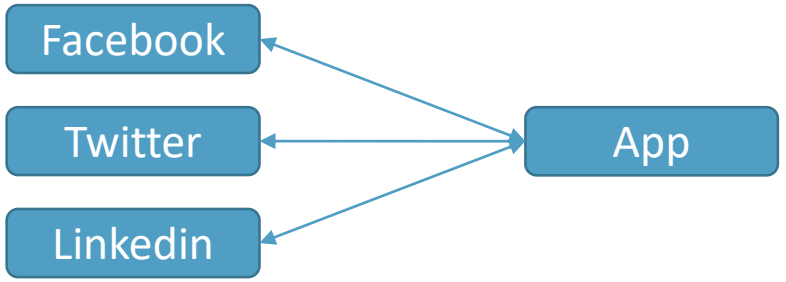
```
async Task<IEnumerable<Message>> LoadMessagesAsync(string hashtag)
{
    ...
    await Task.WhenAll(facebookSearchTask,
                       twitterSearchTask,
                       linkedinSearchTask);
    return ...;
}
```



Pull Model

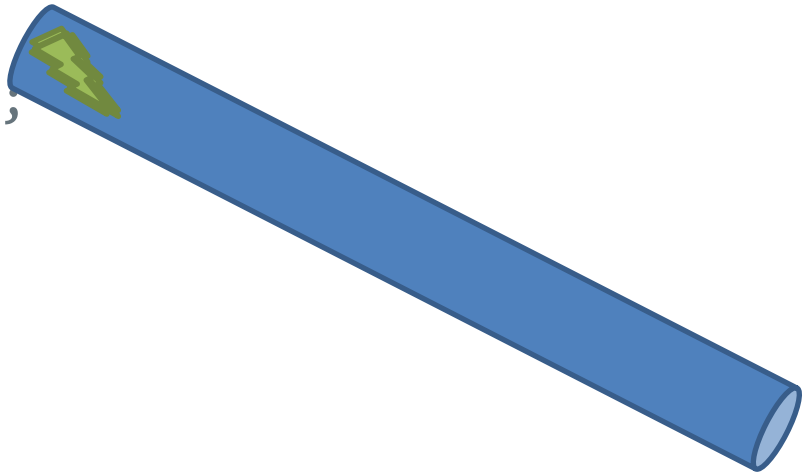
```
IEnumerable<Message> LoadMessages(string hashtag)
{
    var statuses = facebook.Search(hashtag);
    foreach(var s in statuses) yield return s;
    ...

}
```



Push Model

```
???? LoadMessages(string hashtag)
{
    facebook.Search(hashtag);
    twitter.Search(hashtag);
    linkedin.Search(hashtag);
}
```



DoSomething(msg)

Push model with Events

```
class SocialNetworksManager
{
    //members

    public event EventHandler<Message> MessageAvailable;
    public void LoadMessages(string hashtag)
    {
        var statuses = _facebook.Search(hashtag);
        NotifyMessages(statuses);
        :
    }
}
```

can't send as argument

```
var mgr = new SocialNetworksManager();
mgr.MessageAvailable += (sender, msg) => {//do something};
mgr.LoadMessages("Rx");
```

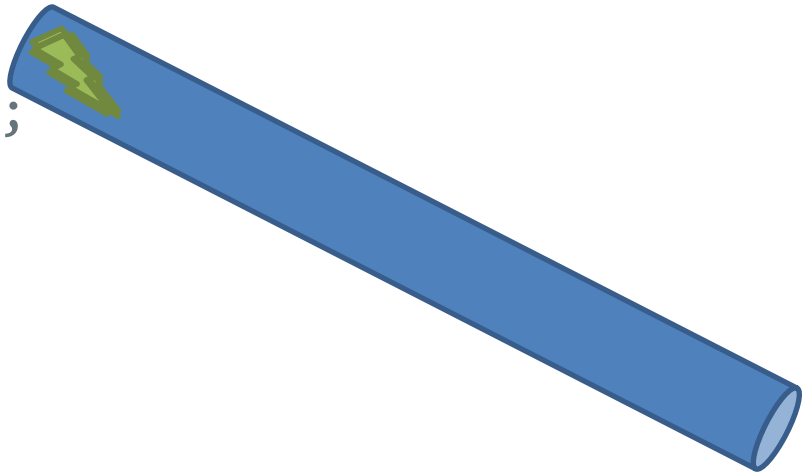
no isolation

hard to unsubscribe

non-composable

Push Model

```
???? LoadMessages(string hashtag)
{
        facebook.Search(hashtag);
        twitter.Search(hashtag);
        linkedin.Search(hashtag);
}
```



DoSomething(msg)

Interfaces

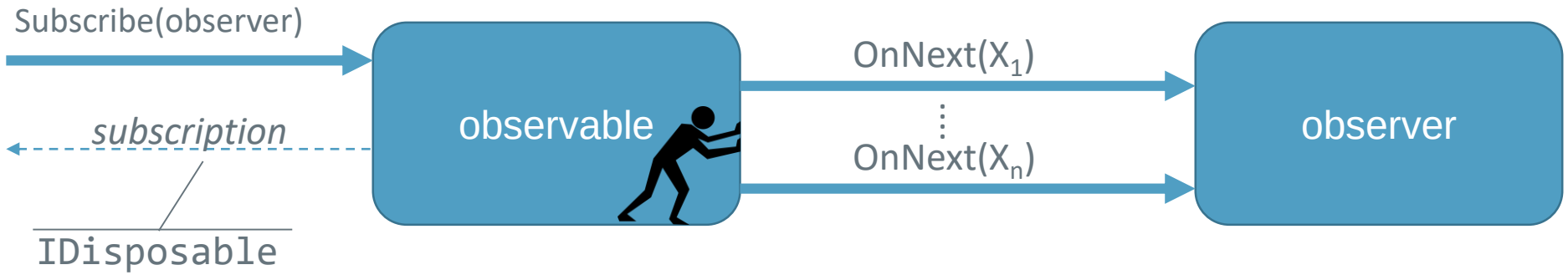
```
namespace System
{
    public interface IObservable<out T>
    {
        IDisposable Subscribe(IObserver<T> observer);
    }

    public interface IObserver<in T>
    {
        void OnNext(T value);
        void OnError(Exception error);
        void OnCompleted();
    }
}
```

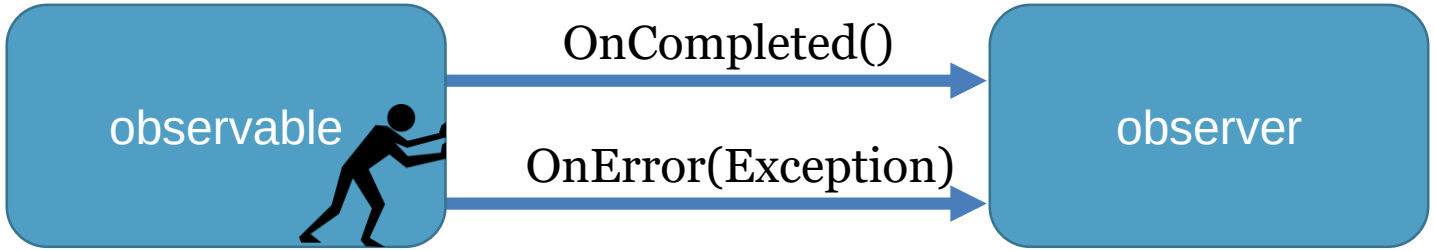
Producer

Consumer

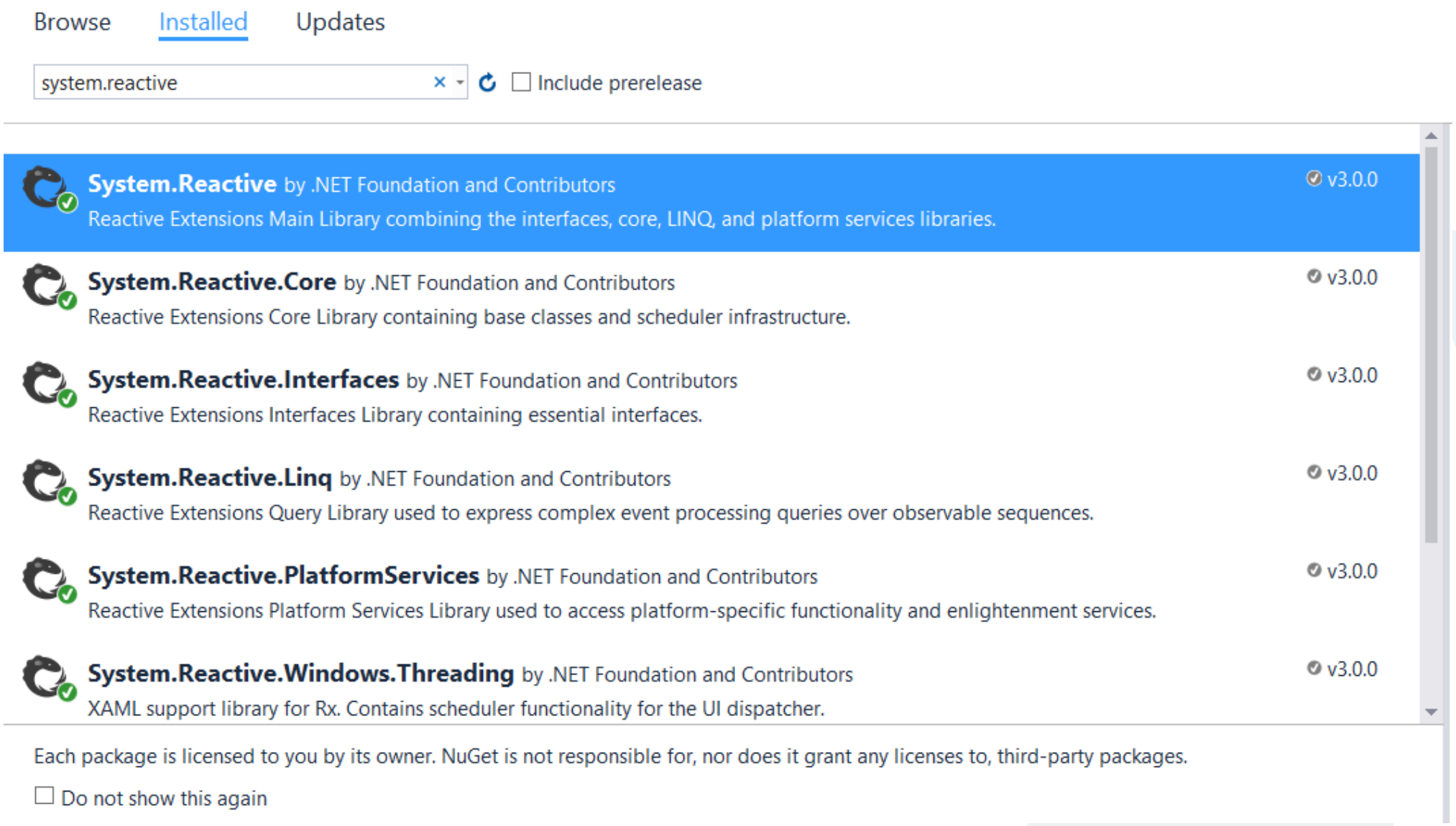
Observables and Observers



Observables and Observers



Rx packages


Browse
Installed
Updates
system.reactive
Include prerelease
System.Reactive by .NET Foundation and Contributors
Reactive Extensions Main Library combining the interfaces, core, LINQ, and platform services libraries.
v3.0.0
System.Reactive.Core by .NET Foundation and Contributors
Reactive Extensions Core Library containing base classes and scheduler infrastructure.
v3.0.0
System.Reactive.Interfaces by .NET Foundation and Contributors
Reactive Extensions Interfaces Library containing essential interfaces.
v3.0.0
System.Reactive.Linq by .NET Foundation and Contributors
Reactive Extensions Query Library used to express complex event processing queries over observable sequences.
v3.0.0
System.Reactive.PlatformServices by .NET Foundation and Contributors
Reactive Extensions Platform Services Library used to access platform-specific functionality and enlightenment services.
v3.0.0
System.Reactive.Windows.Threading by .NET Foundation and Contributors
XAML support library for Rx. Contains scheduler functionality for the UI dispatcher.
v3.0.0
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.
Do not show this again

Push Model with Rx Observables

```
class ReactiveSocialNetworksManager
{
    //members

    public IObservable<Message> ObserveMessages(string hashtag)
    {
        :
    }
}
```

```
var mgr = new ReactiveSocialNetworksManager();
IDisposable subscription =
    mgr.ObserveMessages("Rx")
        .Subscribe(
            msg => Console.WriteLine($"Observed:{msg} \t"),
            ex => { /*OnError*/ },
            () => { /*OnCompleted*/ });
```

Creating Observables

Pull To Push

FacebookClient:

```
IObservable<Message> ObserveMessages(string hashtag)
{
    return Observable.Create( async (observer) =>
    {
        var messages = await GetMessagesAsync(hashtag);
        foreach(var m in messages)
        {
            observer.OnNext(m);
        }
        observer.OnCompleted();

    });
}
```

Subscribe function

Built-in Converters

```
IObservable<Message> ObserveMessages(string hashtag)
{

        var messages = await GetMessagesAsync(hashtag);
        return messages.ToObservable();

}
```

Note: All observers will get the same emitted messages

Deferring the observable creation

```
IObservable<Message> ObserveMessages(string hashtag)
{
    return Observable.Defer( async () =>
    {
        var messages = await GetMessagesAsync(hashtag);
        return messages.ToObservable();

    });
}
```

Note: Every observer will get the "fresh" emitted messages

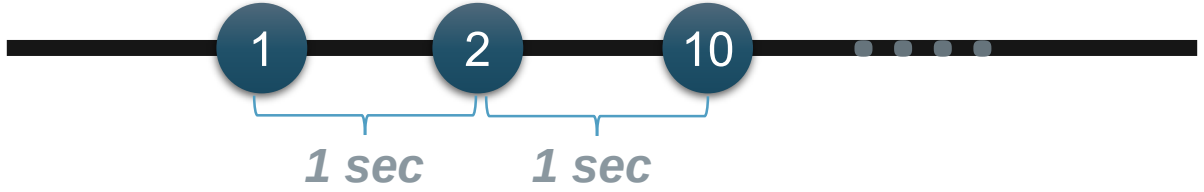
Built-in Combinators (combining operators)

```
Observable.Merge(
    facebook.ObserveMessages(hashtag),
    twitter.ObserveMessages(hashtag),
    linkedin.ObserveMessages(hashtag)
);
```

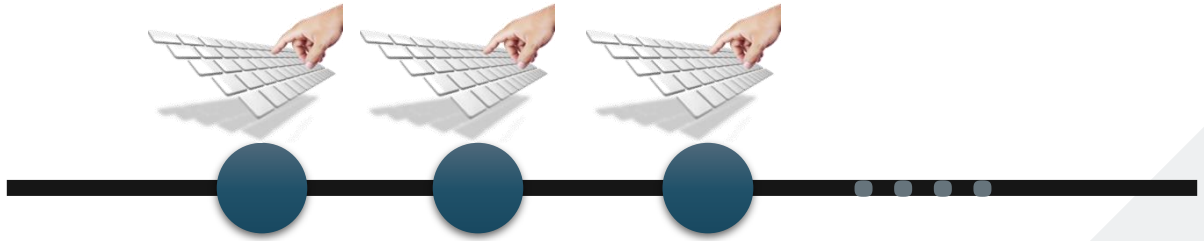
Observables Factories

```
Observable.Range(1, 10)
    .Subscribe(x => Console.WriteLine(x));
```


```
Observable.Interval(TimeSpan.FromSeconds(1))
    .Subscribe(x => Console.WriteLine(x));
```

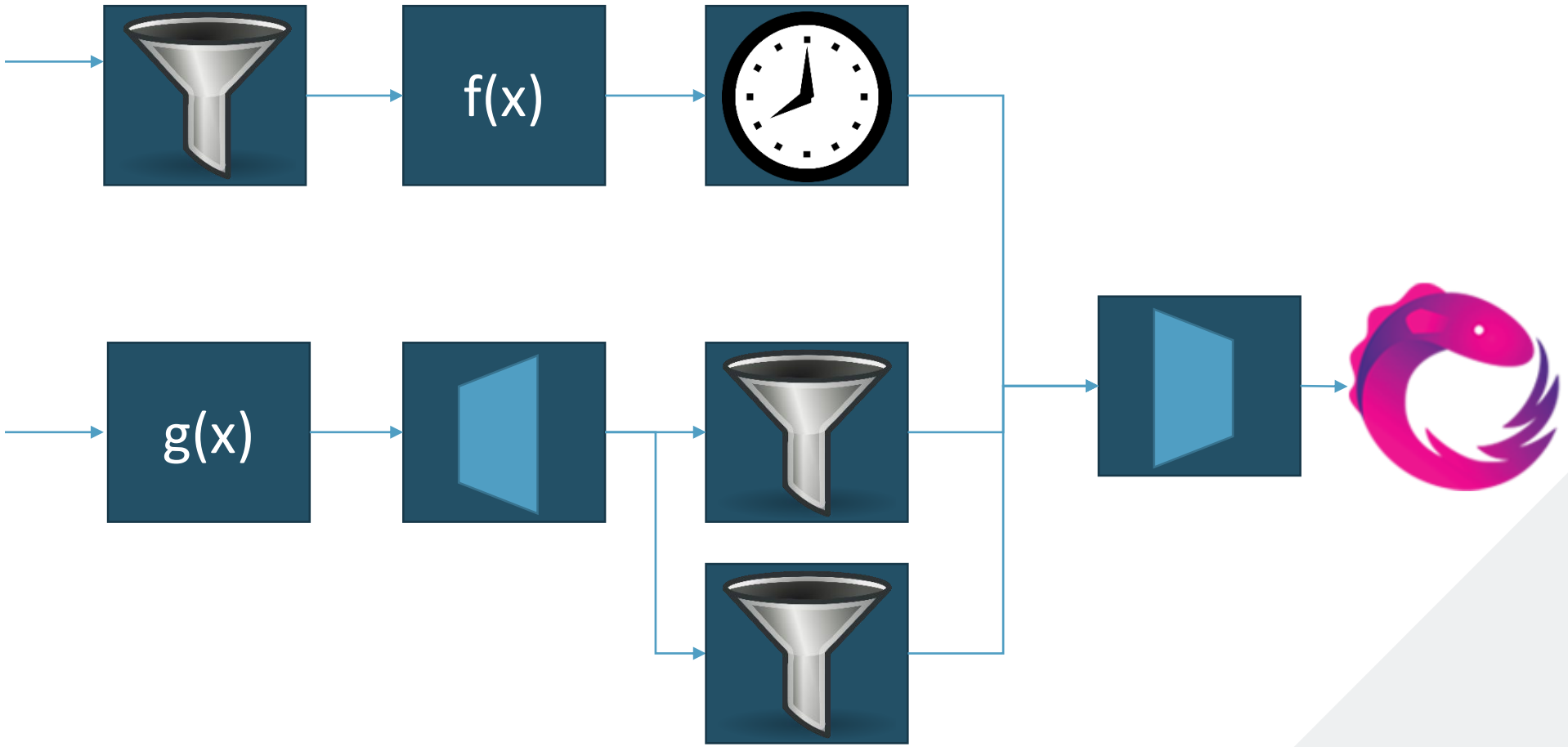



```
Observable.FromEventPattern(SearchBox, "TextChanged")
```

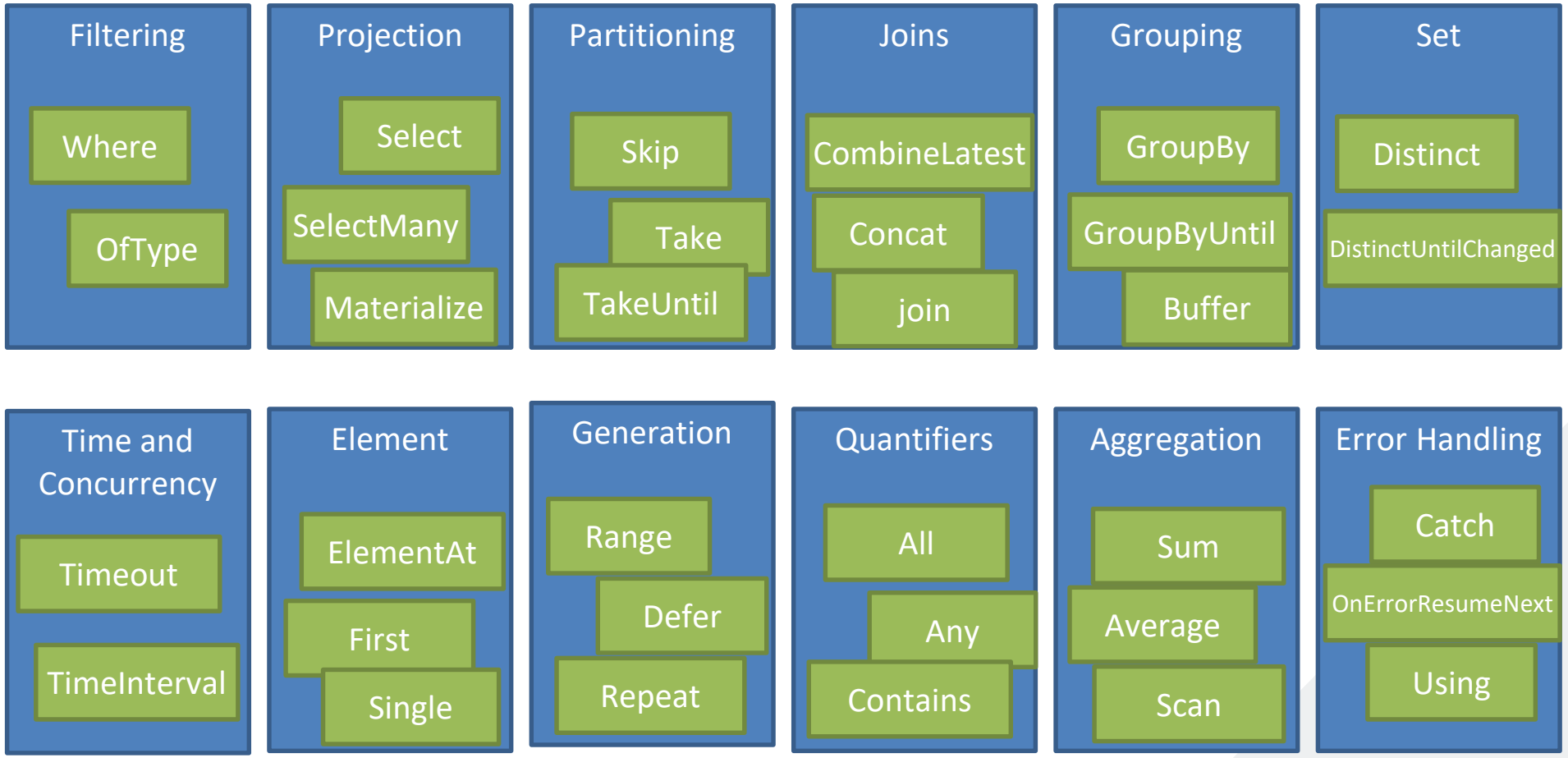


Observable Queries (Rx Operators)

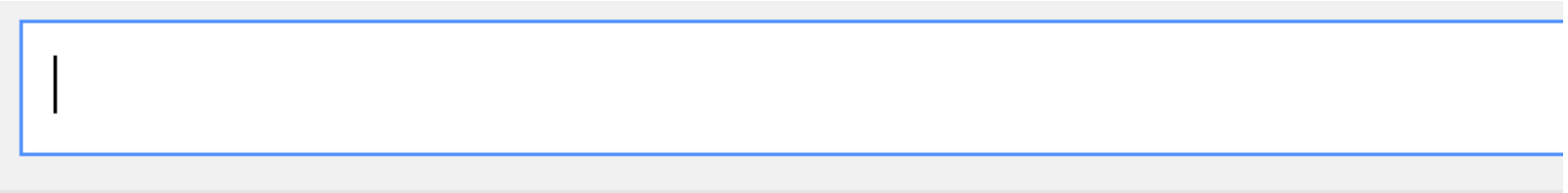
Observable Query Pipeline



Rx operators



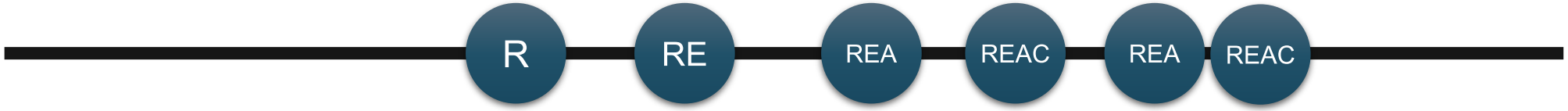
Reactive Search

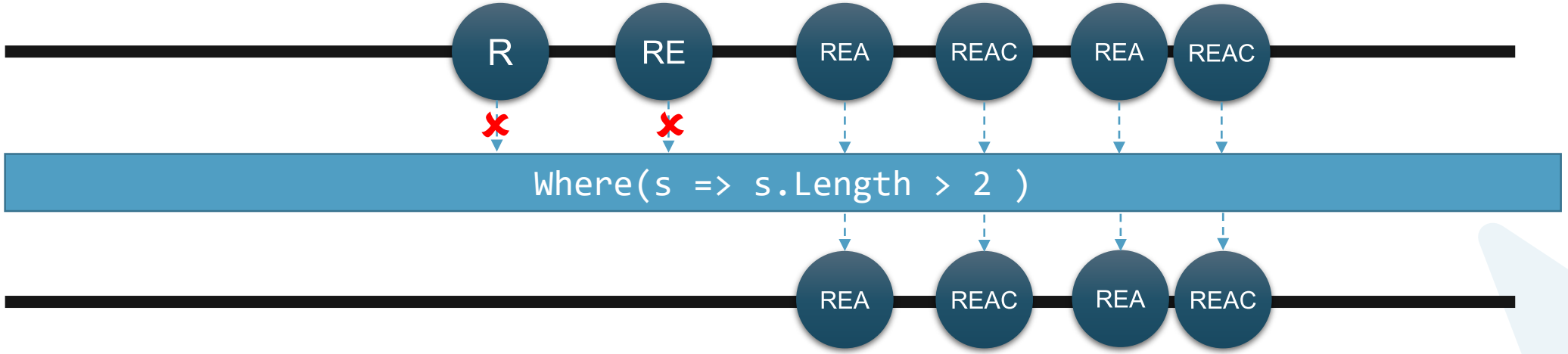


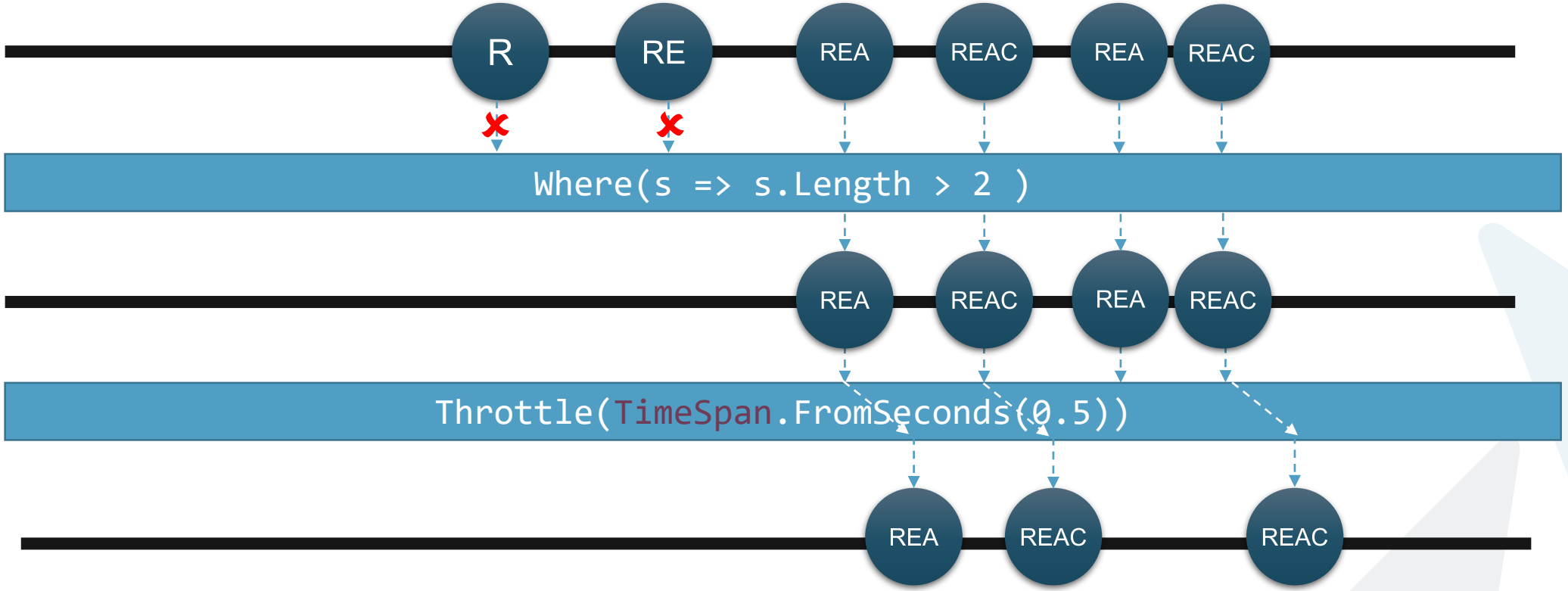
Reactive Search - Rules

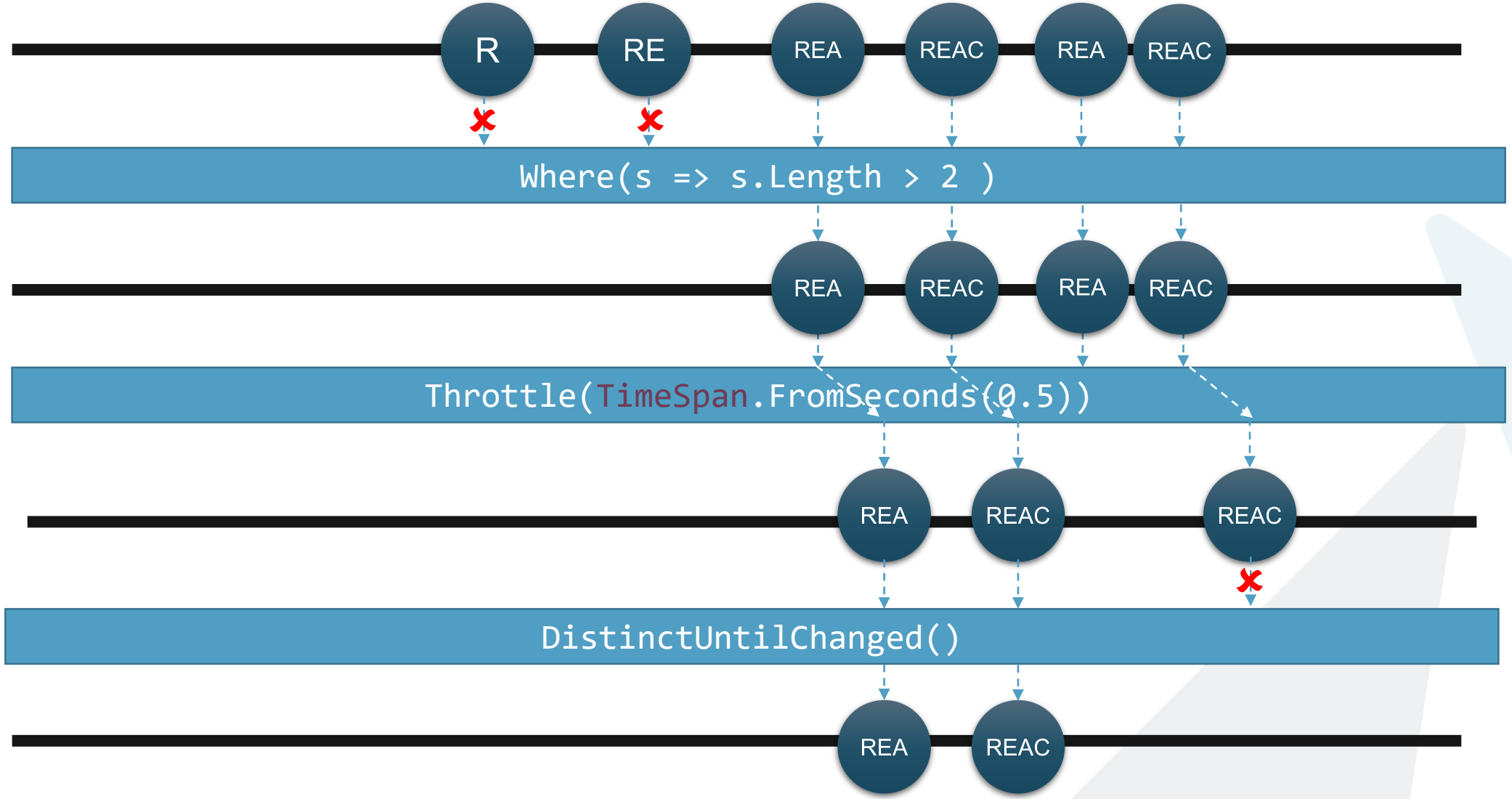
1. At least 3 characters
2. Don’t overflow server (0.5 sec delay)
3. Don’t send the same string again
4. Discard results if another search was requested

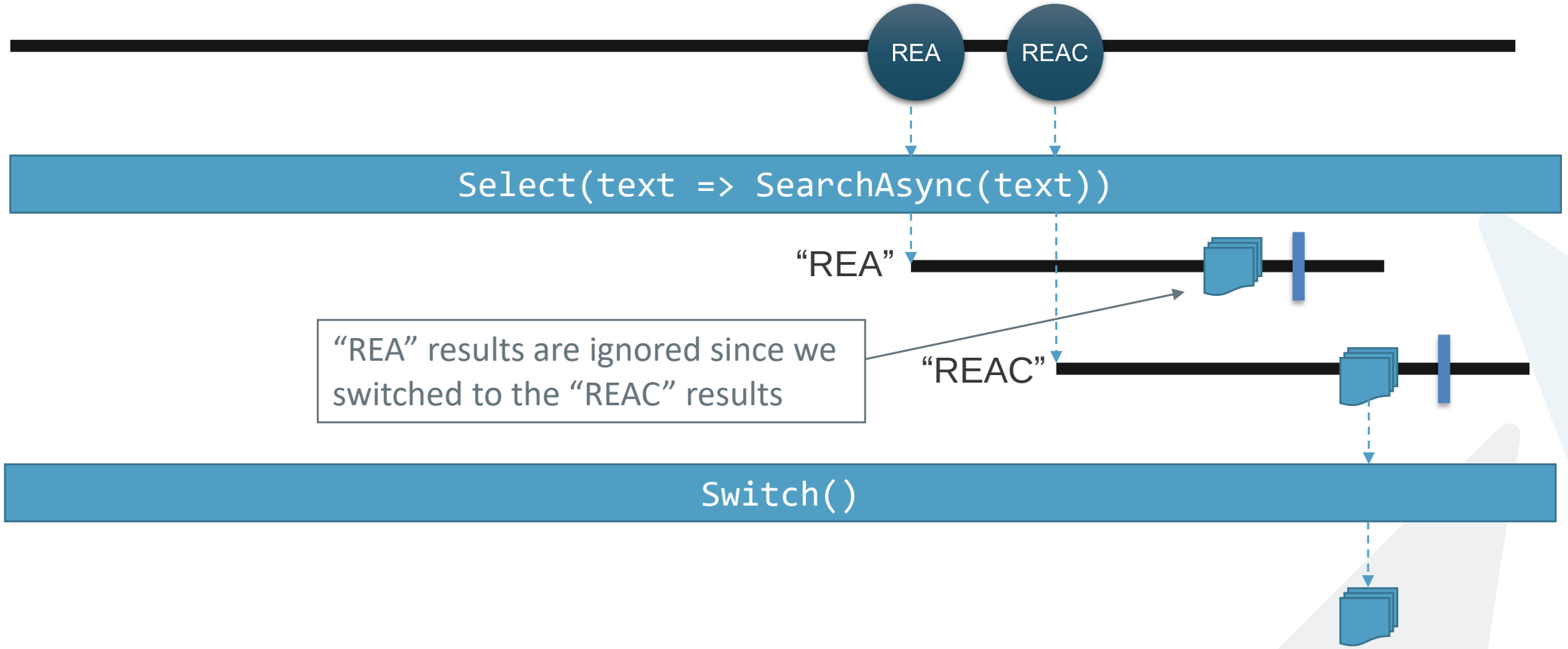
Demo


R
RE
REA
REAC
REA
REAC


R
RE
REA
REAC
REA
REAC
Where(s => s.Length > 2)
REA
REAC
REA
REAC

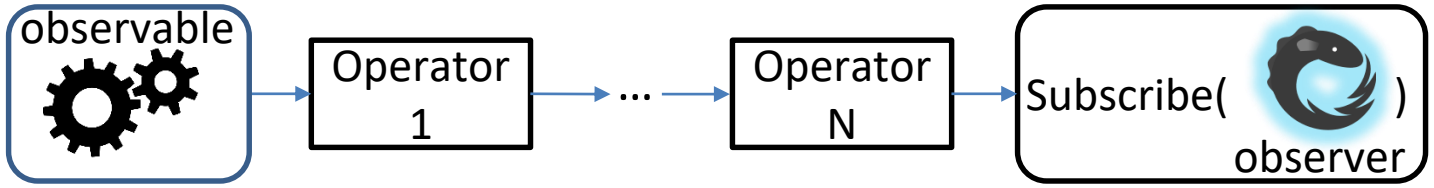

R
RE
REA
REAC
REA
REAC
Where(s => s.Length > 2)
REA
REAC
REA
REAC
Throttle(TimeSpan.FromSeconds(0.5))
REA
REAC
REAC

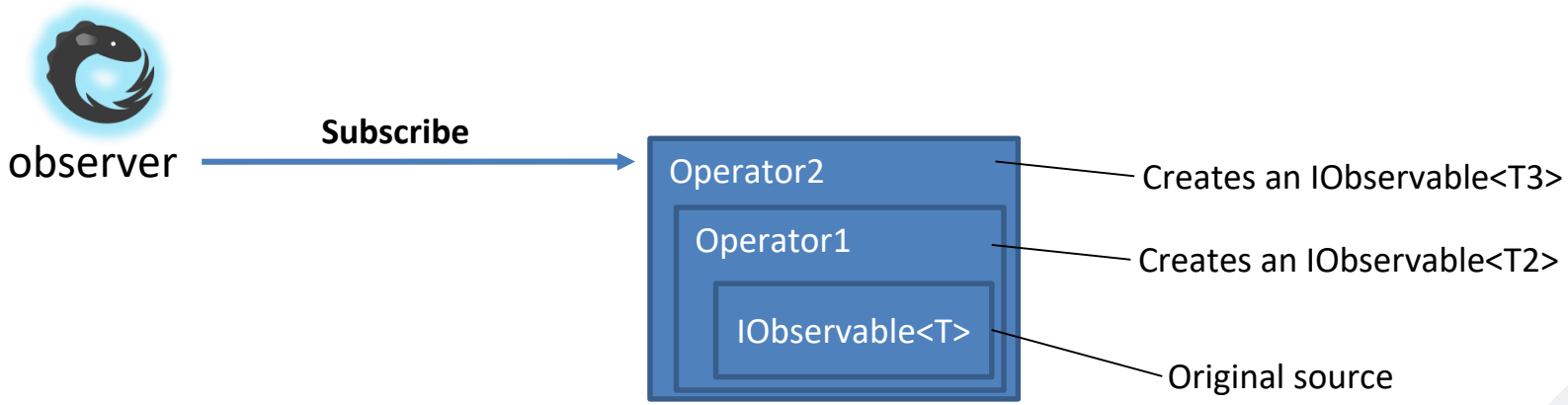

R
RE
REA
REAC
REA
REAC
Where(s => s.Length > 2)
REA
REAC
REA
REAC
Throttle(TimeSpan.FromSeconds(0.5))
REA
REAC
REAC
DistinctUntilChanged()
REA
REAC

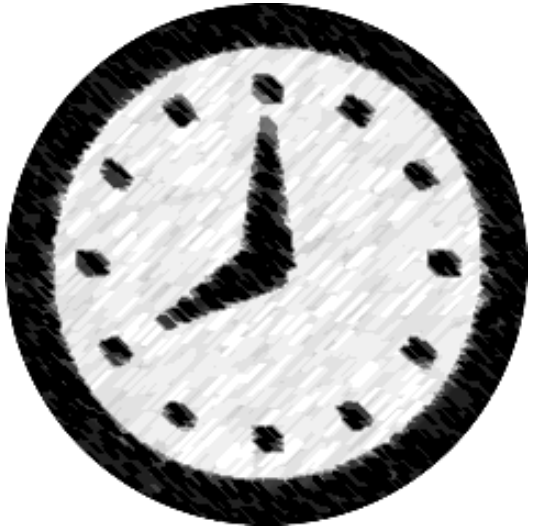

REA
REAC
Select(text => SearchAsync(text))
“REA”
“REA” results are ignored since we switched to the “REAC” results
“REAC”
Switch()

Composability

```
IObservable<T2> Operator<T1>(this IObservable<T1> source, parameters);
```







Abstracting Time and Concurrency

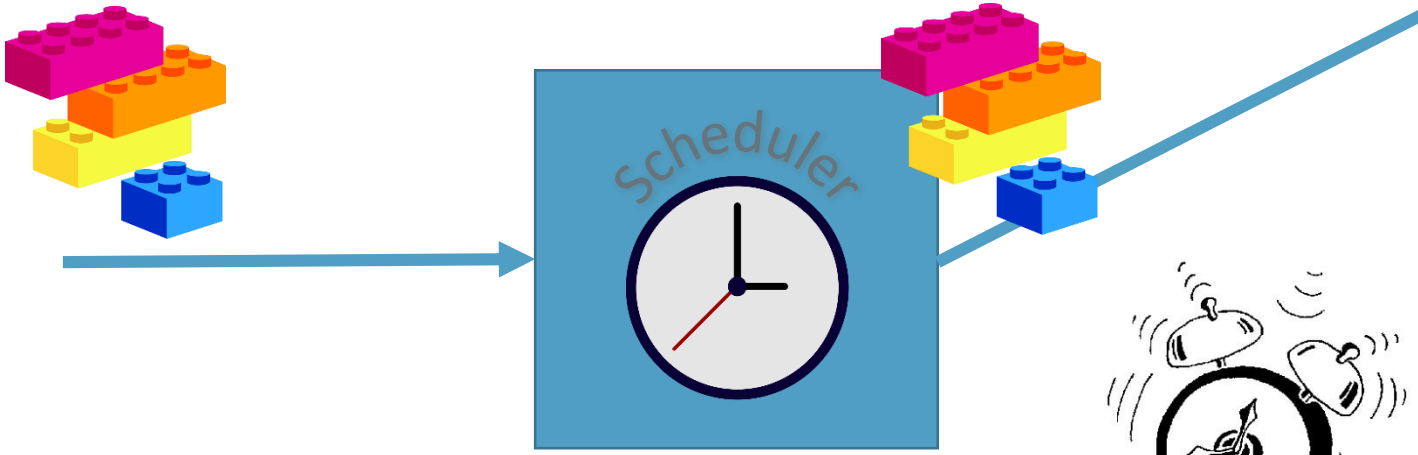
What will be the output?

```
var subscription =
    Observable.Range(1, 5)
        .Repeat()
        .Subscribe(x => Console.WriteLine(x));

subscription.Dispose();
```

- The ***Repeat*** operator resubscribes the observer when the observable completes
- Most developers **falsely** believe that the program will immediately dispose the subscription and nothing will be written to the console
- Rx **is not** concurrent unless it explicitly told to
  - The example above writes the sequence 1-5 indefinitely to the console

Schedulers



Thread Pool

Task Scheduler

other

```
public interface IScheduler
{
    DateTimeOffset Now { get; }

    IDisposable Schedule<TState>(  TState state,
      Func<IScheduler, TState, IDisposable> action);

    IDisposable Schedule<TState>(TimeSpan dueTime,
      TState state,
      Func<IScheduler, TState, IDisposable> action);

    IDisposable Schedule<TState>(DateTimeOffset dueTime,
      TState state,
      Func<IScheduler, TState, IDisposable> action);
}
```

Parameterizing Concurrency

```
//
// Runs a timer on the default scheduler
//
IObservable<T> Throttle<T>(TimeSpan dueTime);

//
// Every operator that introduces concurrency
// has an overload with an IScheduler
//
IObservable<T> Throttle<T>(TimeSpan dueTime,
                           IScheduler scheduler);
```

Parameterizing Concurrency

```
_subscription =
    Observable.FromEventPattern(SearchBox, nameof(SearchBox.TextChanged))
        .Select(_ => SearchBox.Text)
        .Where(t => t.Length >= minTextLength)
        .Throttle(TimeSpan.FromSeconds(0.5), NewThreadScheduler.Default)
        .DistinctUntilChanged()
        .Select(t => _client.SearchAsync(t))
        .Switch()
        .ObserveOnDispatcher()
        .Subscribe(
            x => SearchResults.ItemsSource = x,
            ex=> Debug.Write(ex));
```

Changing Execution Context

```
//
// runs the observer callbacks on the specified
// scheduler.
//
IObservable<T> ObserveOn<T>(IScheduler);

//
// runs the observer subscription and unsubsciption on
// the specified scheduler.
//
IObservable<T> SubscribeOn<T>(IScheduler)
```

Changing Execution Context

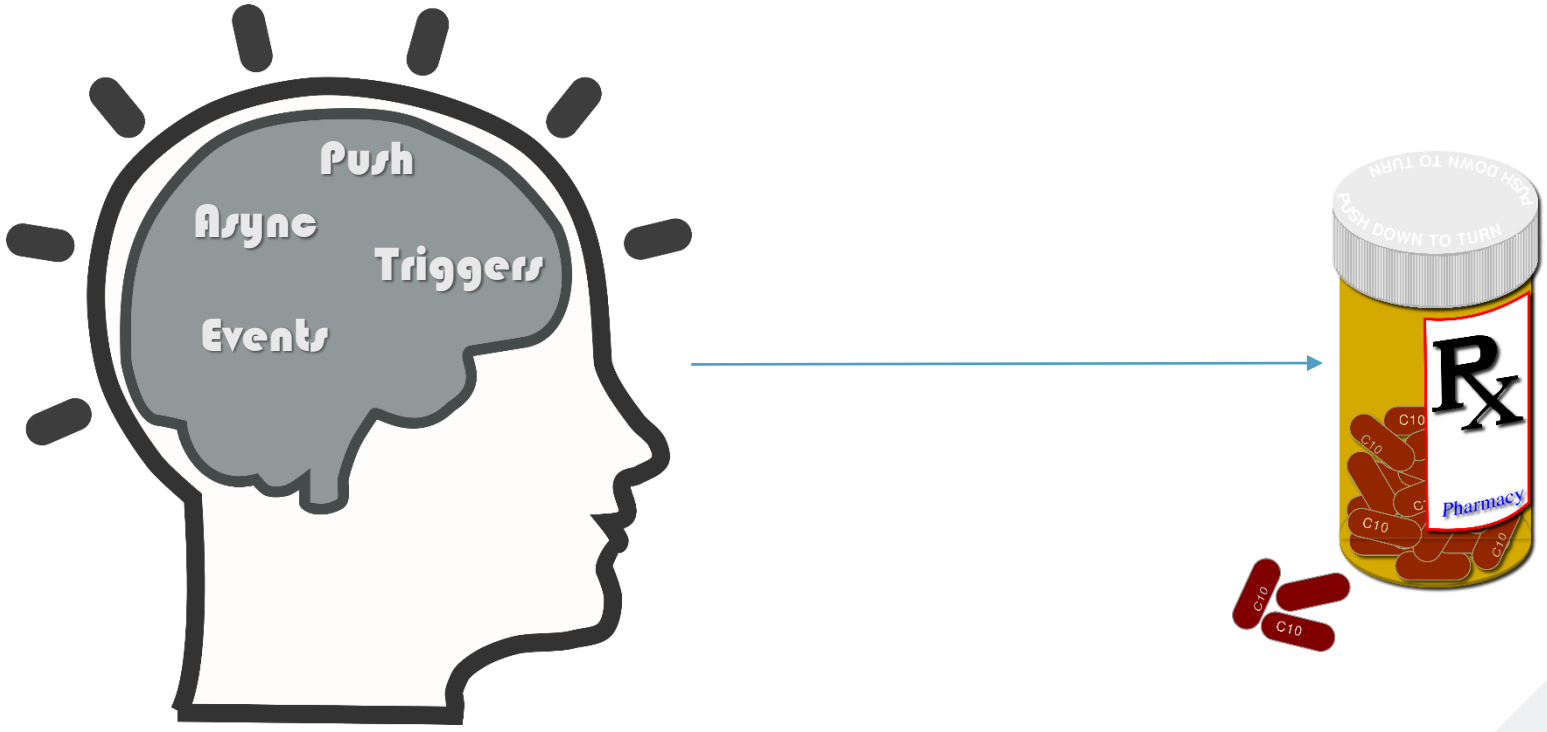
```
_subscription =
    Observable.FromEventPattern(SearchBox, nameof(SearchBox.TextChanged))
        .Select(_ => SearchBox.Text)
        .Where(t => t.Length >= minTextLength)
        .Throttle(TimeSpan.FromSeconds(0.5))
        .DistinctUntilChanged()
        .Select(t => _client.SearchAsync(t))
        .Switch()
        .ObserveOn(DispatcherScheduler.Current)
        .Subscribe(
            x => SearchResults.ItemsSource = x,
            ex=> Debug.Write(ex));
```

Changing Execution Context

```
_subscription =
    Observable.FromEventPattern(SearchBox, nameof(SearchBox.TextChanged))
        .Select(_ => SearchBox.Text)
        .Where(t => t.Length >= minTextLength)
        .Throttle(TimeSpan.FromSeconds(0.5))
        .DistinctUntilChanged()
        .Select(t => _client.SearchAsync(t))
        .Switch()
        .ObserveOnDispatcher()
        .Subscribe(
            x => SearchResults.ItemsSource = x,
            ex=> Debug.Write(ex));
```

Reactive Extensions

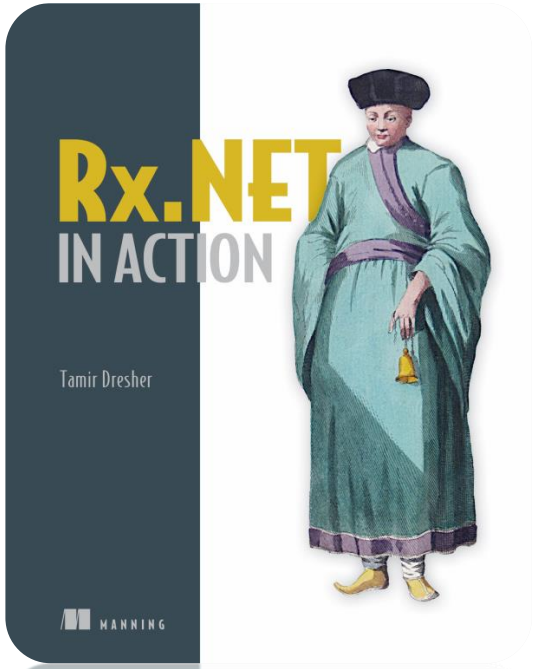
Your headache relief pill to Asynchronous and Event based applications



Summary

- Rx building blocks – Observables and Observers
  - Pull vs. Push
- Creating Observables
  - `Observable.Create()`
  - `Observable.Defer()`
- Rx queries with Rx operators
  - LINQ to Events
- Managing Concurrency with Rx Schedulers
  - Parameterized Concurrency
  - `SubscribeOn()`
  - `ObserverOn()`

Thank You



www.manning.com/dresher

www.reactivex.io

github.com/Reactive-Extensions

https://reactiveui.net/slack

gitter.im/Reactive-Extensions/Rx.NET

@tamir_dresher