Reactive Extensions (Rx)

Rafał Łopatka

Reactive Extensions

The ReactiveX allows you to treat streams of asynchronous events with the same sort of simple, composable operations that you use for collections of data items like arrays. It frees you from tangled webs of callbacks, and thereby makes your code more readable and less prone to bugs.

Battle tested

ReactiveX Is a Polyglot Implementation

ReactiveX is a collection of open source projects. ReactiveX is currently implemented in a variety of languages, in ways that respect those languages' idioms, and more languages are being added at a rapid clip.

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

Rx.NET

The Reactive Extensions (Rx) is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators. Using Rx, developers represent asynchronous data streams with Observables, query asynchronous data streams using LINQ operators, and parameterize the concurrency in the asynchronous data streams using Schedulers.

Simply put, **Rx = Observables + LINQ + Schedulers.**

Rx.NET MouseMove

```
mouseMoves
    .Select(signal => new
    {
        Position = signal.EventArgs.GetPosition(MouseMoveContainer),
        IsPressed = signal.EventArgs.LeftButton == MouseButtonState.Pressed
    })
    .Buffer(count: 2, skip: 1)
    .Select(buffer => new
    {
        Previous = buffer.First().Position,
        Current = buffer.Last().Position,
        IsPressed = buffer.Last().IsPressed
    })
    .Where(x => x.IsPressed)
    .Subscribe(points =>
    {
        Draw(points.Previous, points.Current);
    });
```

The Observer pattern done right

In ReactiveX an observer subscribes to an Observable. Then that observer reacts to whatever item or sequence of items the Observable emits.

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

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

IObservable<T> & IObserver<T>

```
mouseMoves
    .Select(signal => new
    {
        Position = signal.EventArgs.GetPosition(MouseMoveContainer),
        IsPressed = signal.EventArgs.LeftButton == MouseButtonState.Pressed
    })
    .Buffer(count: 2, skip: 1)
    .Select(buffer => new
    {
        Previous = buffer.First().Position,
        Current = buffer.Last().Position,
        IsPressed = buffer.Last().IsPressed
    })
    .Where(x => x.IsPressed)
    .Subscribe(points =>
    {
        Draw(points.Previous, points.Current);
    });
```

IObservable<T>

IObserver<T>

.NET Observer vs ReactiveX Observer

- In C#, events have a curious interface. Some find the += and -= operators an unnatural way to register a callback
- **Events are difficult to compose**
- **Events don't offer the ability to be easily queried over time**
- Events are a common cause of accidental memory leaks
- **Events do not have a standard pattern for signaling completion**
- **Events provide almost no help for concurrency or multithreaded applications. e.g. To raise an event on a separate thread requires you to do all of the plumbing**

Operators

Creating observables:

λ Create λ Defer λ Empty λ Never λ Throw λ From λ Interval λ Just λ Range λ Repeat λ Start λ Timer

Transforming observables:

λ Buffer λ FlatMap λ GroupBy λ Map λ Scan λ Window

Filtering observables:

λ Debounce λ Distinct λ ElementAt λ Filter λ First λ IgnoreElements λ Last λ Sample λ Skip λ SkipLast λ Take λ TakeLast

Combining observables:

λ And λ Then λ When λ CombineLatest λ Join λ Merge λ StartWith λ Switch λ Zip

Error handling:

λ Catch λ Retry

Utilities:

λ Delay λ Do λ ObserveOn λ Serialize λ Subscribe λ SubscribeOn λ TimeInterval λ Timeout λ Timestamp λ Using

Conditional & Boolean:

λ All λ Amb λ Contains λ DefaultIfEmpty λ SequenceEqual λ SkipUntil λ SkipWhile λ TakeUntil λ TakeWhile

Mathematical and aggregate:

λ Average λ Concat λ Count λ Max λ Min λ Reduce λ Sum

A Decision Tree of Observable Operators

I want to create a new Observable

- …that emits a particular item: Just
  - …that was returned from a function called at subscribe-time: Start
  - …that was returned from an `Action`, `Callable`, `Runnable`, or something of that sort, called at subscribe-time
    : From
  - …after a specified delay: Timer
- …that pulls its emissions from a particular `Array`, `Iterable`, or something like that: From
- …by retrieving it from a Future: Start
- …that obtains its sequence from a Future: From
- …that emits a sequence of items repeatedly: Repeat
- …from scratch, with custom logic: Create
- …for each observer that subscribes: Defer
- …that emits a sequence of integers: Range
  - …at particular intervals of time: Interval
    - …after a specified delay: Timer
- …that completes without emitting items: Empty
- …that does nothing at all: Never

I want to create an Observable by combining other Observables

- …and emitting all of the items from all of the Observables in whatever order they are received: Merge
- …and emitting all of the items from all of the Observables, one Observable at a time: Concat
- …by combining the items from two or more Observables sequentially to come up with new items to emit
  - …whenever *each* of the Observables has emitted a new item: Zip

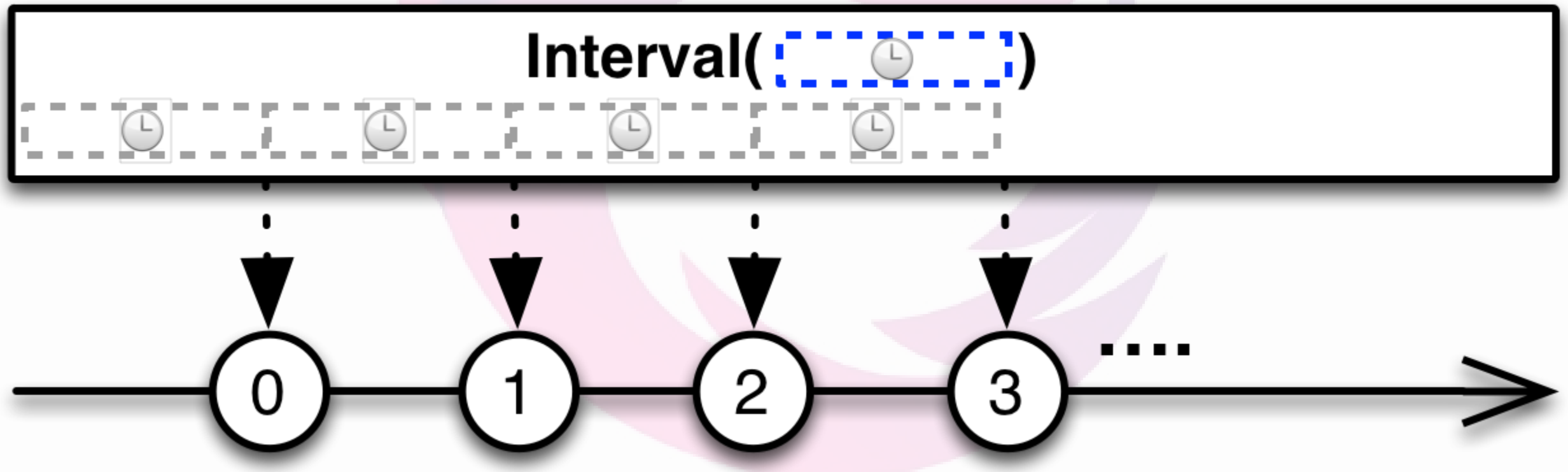
Nuget

```
PM> Install-Package System.Reactive
```

- **System.Reactive**
- System.Reactive.Core
- System.Reactive.Interfaces
- System.Reactive.Linq
- System.Reactive.PlatformServices
- Microsoft.Reactive.Testing

Observable.Interval

create an Observable that emits a sequence of integers spaced by a given time interval

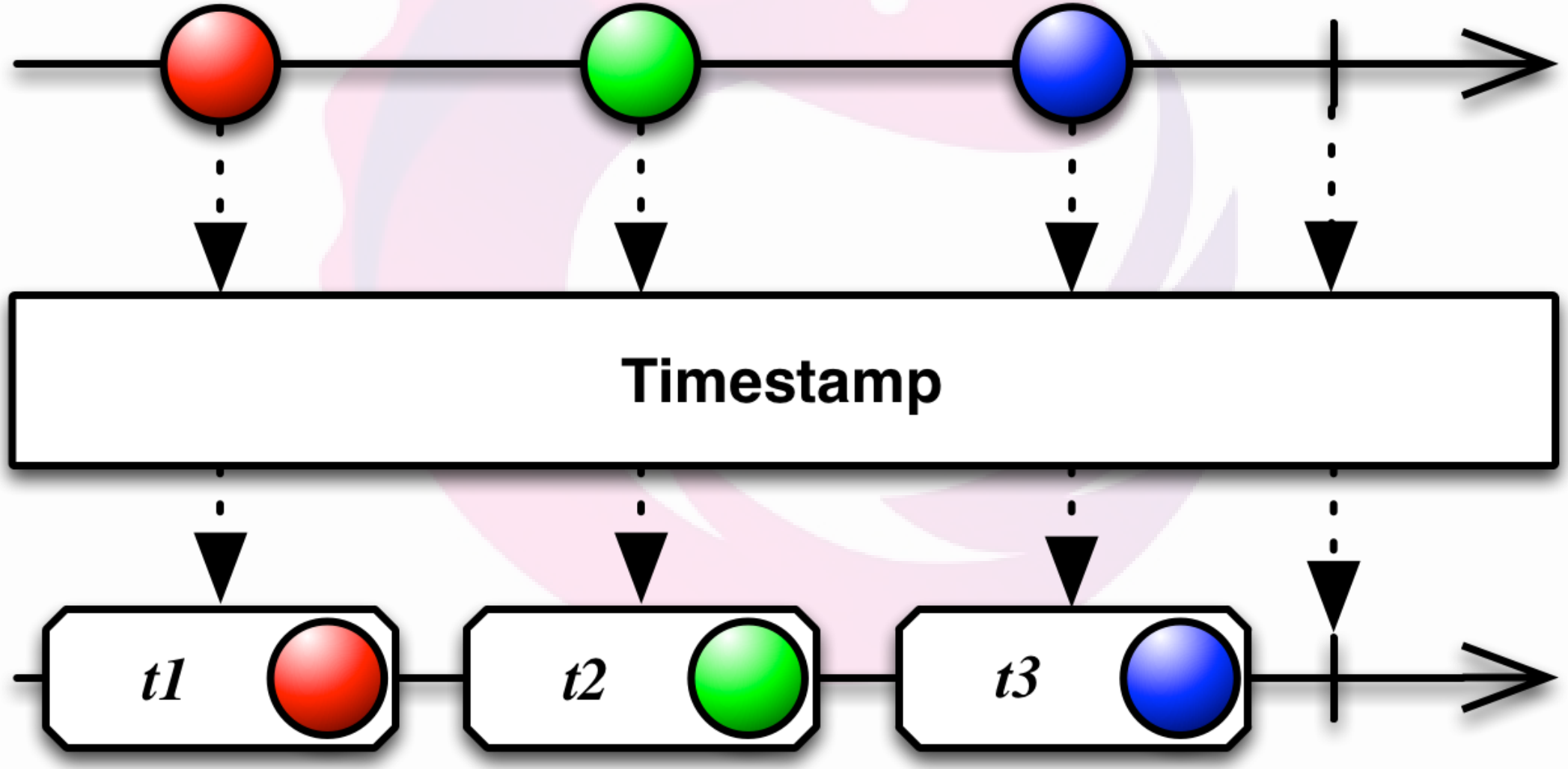


Observable.Interval Example

```
IObservable<long> timeStream = Observable.Interval(TimeSpan.FromSeconds(1));

timeStream
    .Select(signal => new { IsNewContentAvalaible = CheckRemoteContent() })
    .Where(signal => signal.IsNewContentAvalaible)
    .Select(signal => DownloadNewContent())
    .Subscribe(newContent =>
    {
        Console.WriteLine($"New content: {newContent}");
    });
```

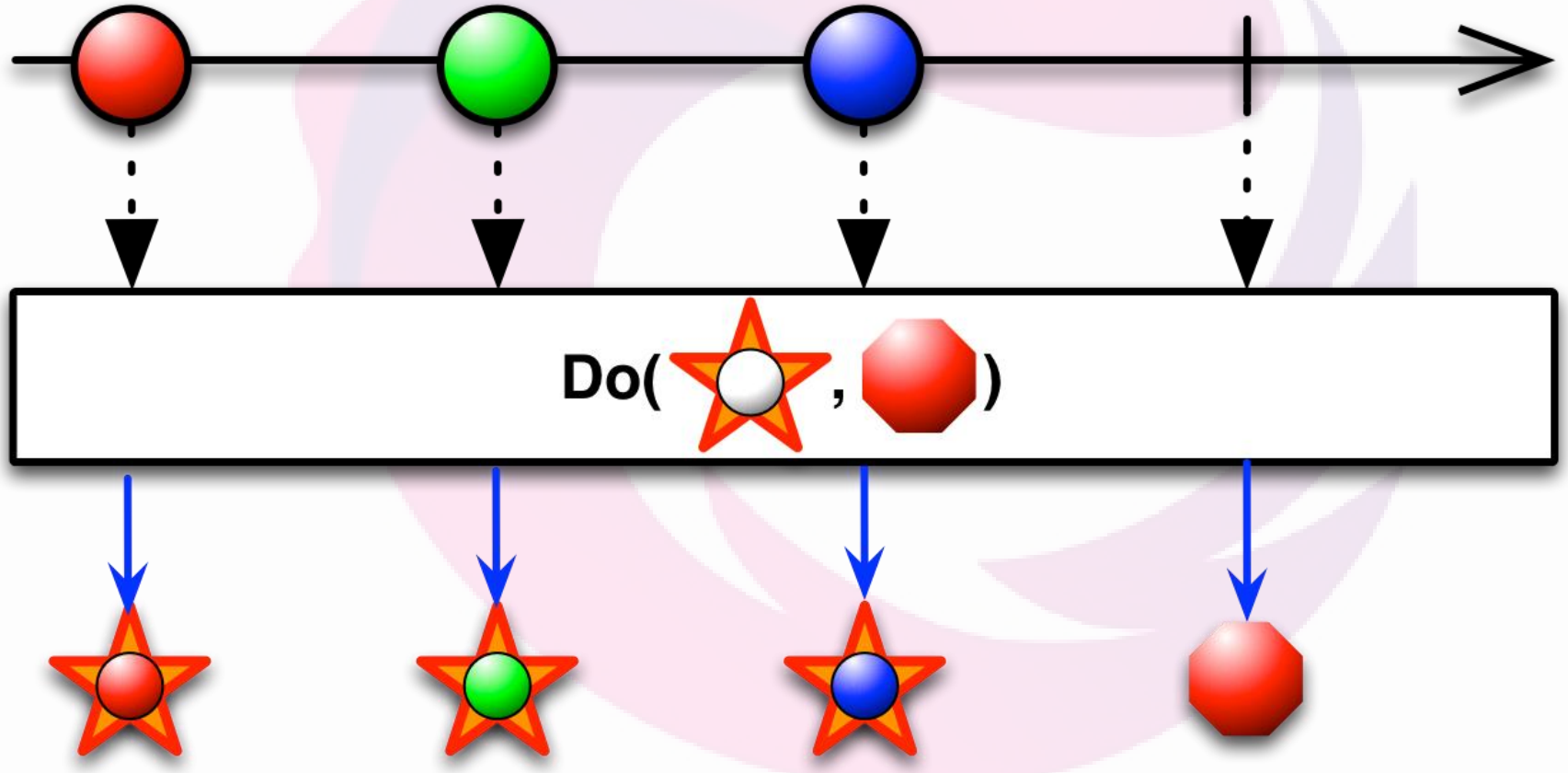
Observable Utilities: TimeStamp

attach a timestamp to each item emitted by an Observable indicating when it was emitted



Observable Utilities: Do

register an action to take upon a variety of Observable lifecycle events



Observable.Interval & Utilities Example

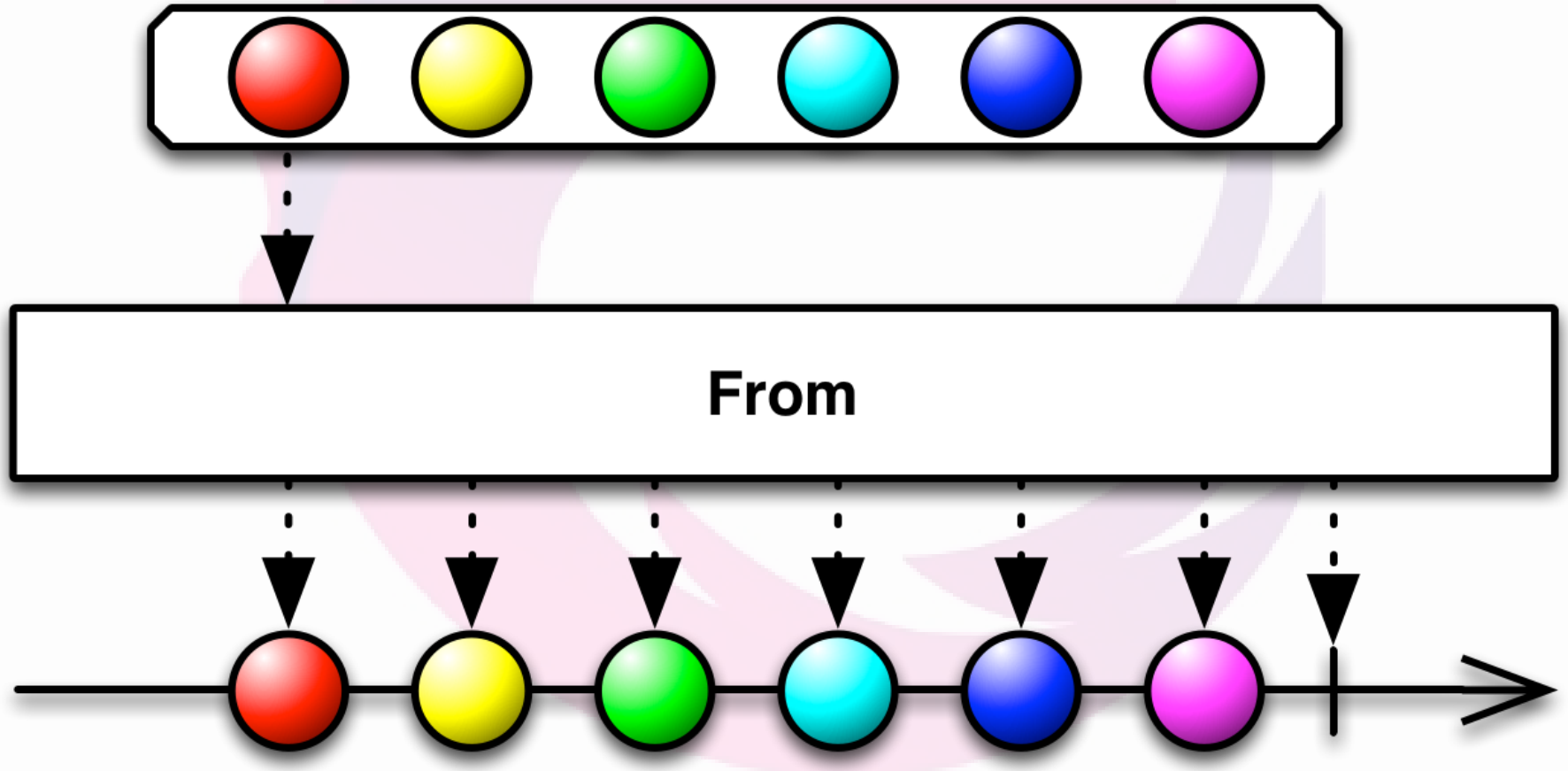
```
IObservable<long> timeStream = Observable.Interval(TimeSpan.FromSeconds(1));
    timeStream
    .Timestamp()
    .Do(signal => Log($"\n[{signal.Value}][{signal.Timestamp}]:\nChecking remote content"))
    .Select(signal => new
    {
        IsNewContentAvalaible = CheckRemoteContent()
    })
    .Do(signal => Log($"\t|Is new content avalaible: {signal.IsNewContentAvalaible}",
                      signal.IsNewContentAvalaible ? ConsoleColor.Green : ConsoleColor.Red))
    .Where(signal => signal.IsNewContentAvalaible)
    .Do(signal => Log($"\t\t|Downloading new content", ConsoleColor.Blue))
    .Select(signal => DownloadNewContent())
    .Subscribe(newContent =>
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine($"\t\t\t|New content: {newContent}");
        Console.ResetColor();
    });
```

Observable.Interval Example Output

```
[0][31.12.2016 21:03:12 +00:00]:
Checking remote content
        |Is new content avalaible: True
                |Downloading new content
                        |New content: 85b53137-5f38-4ffe-b4be-c75fb4cafdac

[1][31.12.2016 21:03:13 +00:00]:
Checking remote content
        |Is new content avalaible: False

[2][31.12.2016 21:03:14 +00:00]:
Checking remote content
        |Is new content avalaible: False

[3][31.12.2016 21:03:15 +00:00]:
Checking remote content
        |Is new content avalaible: False

[4][31.12.2016 21:03:16 +00:00]:
Checking remote content
        |Is new content avalaible: False

[5][31.12.2016 21:03:17 +00:00]:
Checking remote content
        |Is new content avalaible: True
                |Downloading new content
                        |New content: 0834c45d-1582-437b-8fad-a4e66582fd82
```

Observable.From...

convert various other objects and data types into Observables

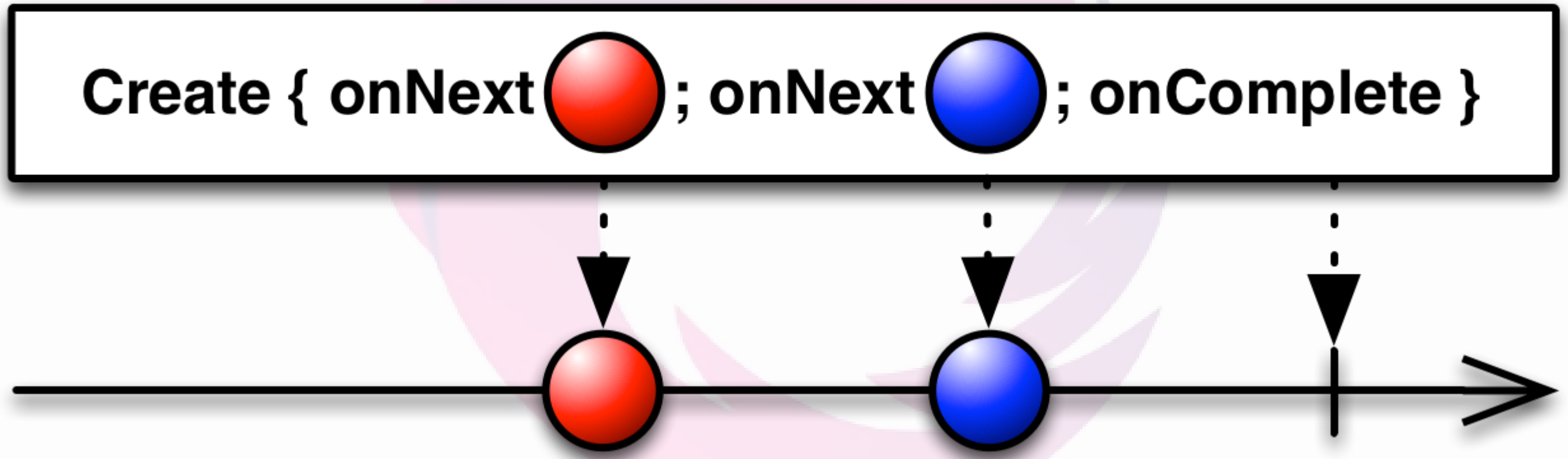


Observable.FromEventPattern Example

```
Observable
    .FromEventPattern<MouseEventArgs>(window, nameof(MouseMoveContainer.MouseMove))
    .Select(signal => new
    {
        Position = signal.EventArgs.GetPosition(MouseMoveContainer),
        IsPressed = signal.EventArgs.LeftButton == MouseButtonState.Pressed
    })
    .Buffer(count: 2, skip: 1)
    .Select(buffer => new
    {
        Previous = buffer.First().Position,
        Current = buffer.Last().Position,
        IsPressed = buffer.Last().IsPressed
    })
    .Where(x => x.IsPressed)
    .Subscribe(points =>
    {
        Draw(points.Previous, points.Current);
    });
```

Observable.Create

create an Observable from scratch by means of a function



Observable.Create Example

```
Observable.Create<long>(observer =>
    {
        int index = 0;
        var timer = new Timer {Interval = 1000};
        ElapsedEventHandler handler = (s, e) => observer.OnNext(index++);
        timer.Elapsed += handler;
        timer.Start();
        return Disposable.Create(() =>
        {
            timer.Elapsed -= handler;
            timer.Dispose();
        });
    })
    .Subscribe(index => Console.WriteLine(index));
```

Observable providers

- Observable.Return
- Observable.Empty
- Observable.Never
- Observable.Throw
- Observable.Create
- Observable.Range
- Observable.Interval
- Observable.Timer
- Observable.Generate
- Observable.Start
- Observable.FromEventPattern
- Task.ToObservable
- Task<T>.ToObservable
- IEnumerable<T>.ToObservable
- Observable.FromAsyncPattern

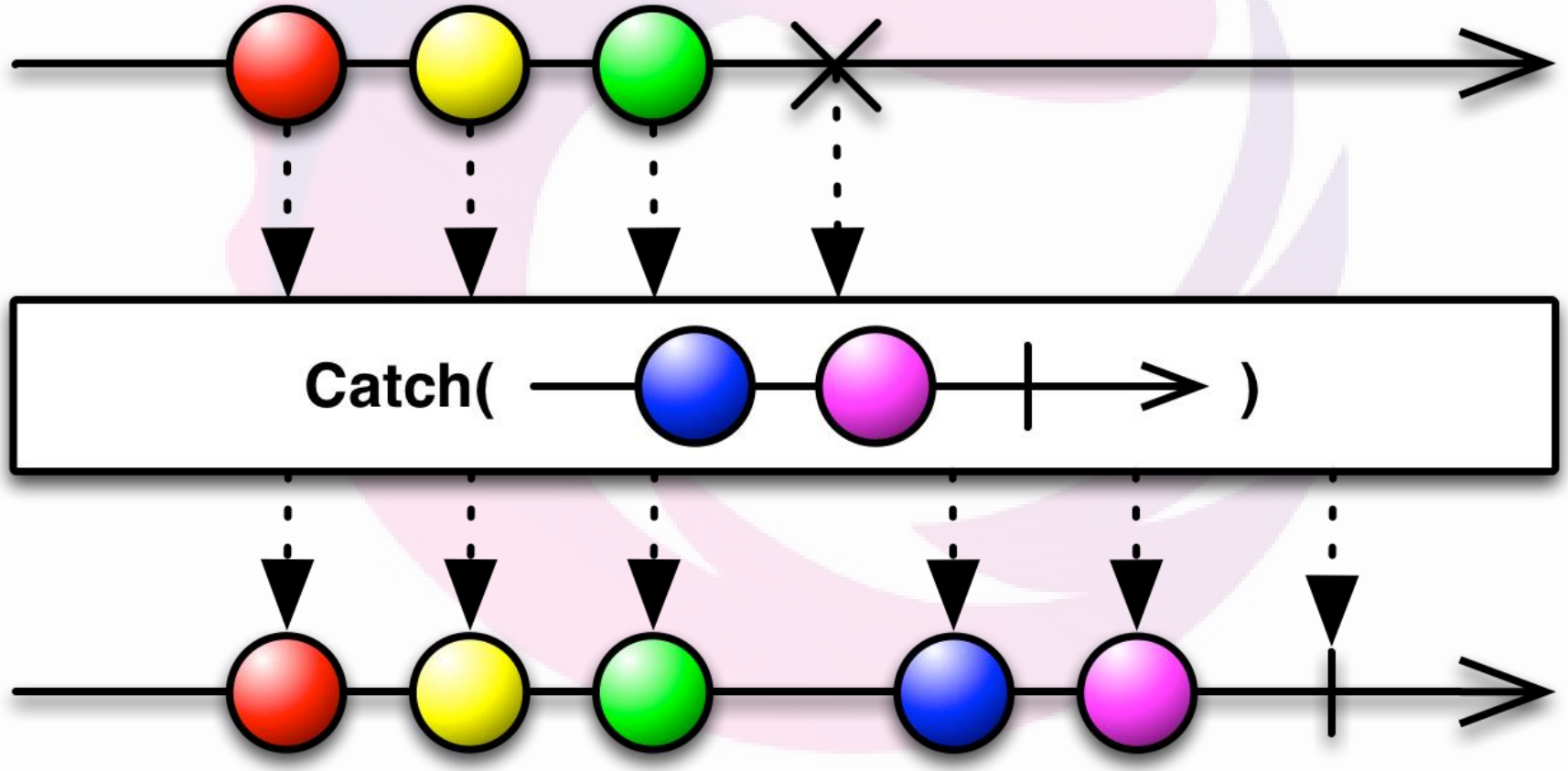
onNext/onError/onCompleted

```
var observable = Observable.Create<int>(observer =>
{
    for (int i = 0; i < 5; i++)
        observer.OnNext(i);

    observer.OnCompleted();
    return Disposable.Empty;
});
observable.Subscribe(onNext: item =>
{
    Console.WriteLine($"OnNext: {item}");
}, onError: exception =>
{
    Console.WriteLine($"OnError: {exception}");
}, onCompleted: () =>
{
    Console.WriteLine("Completed");
});
```

```
OnNext: 0
OnNext: 1
OnNext: 2
OnNext: 3
OnNext: 4
Completed
```

Catch

recover from an onError notification by continuing the sequence without error



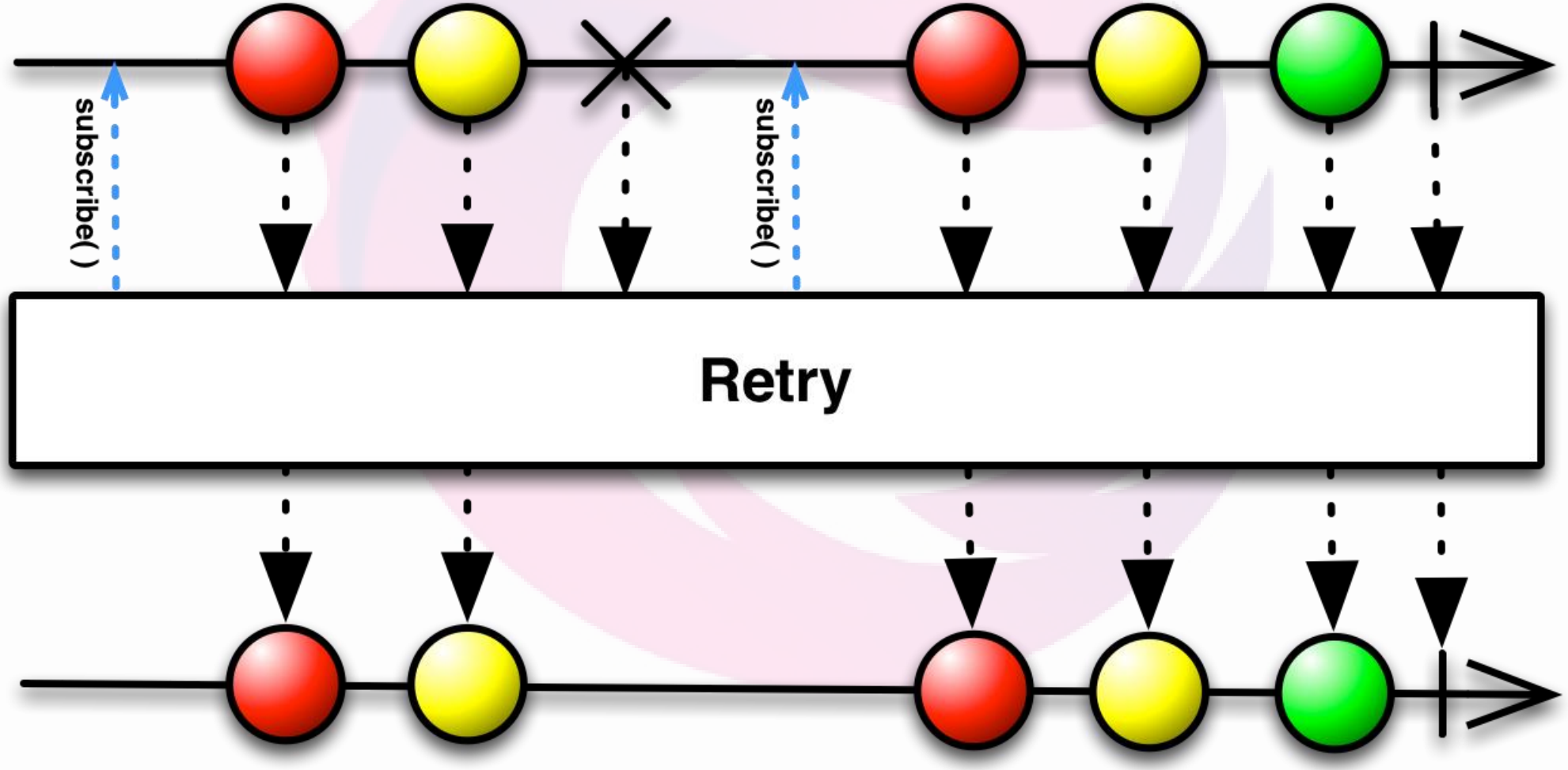
Catch Example

```
OnNext: 1
Exception System.InvalidOperationException
OnNext: -1
Completed
Finally
```

```
Observable.Create<int>(observer =>
{
    observer.OnNext(1);
    throw new InvalidOperationException("InvalidOperationException");
    observer.OnNext(2);
    return Disposable.Empty;
})
.Catch<int, InvalidOperationException>(e =>
{
    Console.WriteLine($"Exception {e.GetType()}");
    return Observable.Return(-1);
})
.Finally(() => Console.WriteLine($"Finally"))
.Subscribe(onNext: item => Console.WriteLine($"OnNext: {item}"),
           onError: exception => Console.WriteLine($"OnError: {exception}"),
           onCompleted: () => Console.WriteLine("Completed"));
```

Retry

if a source Observable emits an error, resubscribe to it in the hopes that it will complete without error



Retry Example

```
OnNext: 1
OnNext: 2
OnNext: 3
OnNext: 1
OnNext: 2
OnNext: 3
OnError: System.InvalidOperationException
```

```
Observable.Create<int>(observer =>
{
    observer.OnNext(1);
    observer.OnNext(2);
    observer.OnNext(3);
    throw new InvalidOperationException("InvalidOperationException");
    return Disposable.Empty;
})
.Retry(2)
.Subscribe(onNext: item => Console.WriteLine($"OnNext: {item}"),
           onError: exception => Console.WriteLine($"OnError: {exception.GetType()}"),
           onCompleted: () => Console.WriteLine("Completed"));
```

Unsubscribe?

```
OnNext 0
OnNext 1
Subscription canceled
```

```
IObservable<long> observable = Observable
    .Interval(TimeSpan.FromSeconds(1));

IDisposable subscription = observable
    .Subscribe(onNext: index => Console.WriteLine($"OnNext {index}"),
        onError: exception => Console.WriteLine("OnError"),
        onCompleted: () => Console.WriteLine("Completed"));

WaitTwoSeconds();
subscription.Dispose();
Console.WriteLine("Subscription canceled");
```

Threading Example #1

Rx is asynchronous and single-threaded by default

```
Console.WriteLine($"[Main ThreadId] {Thread.CurrentThread.ManagedThreadId}");
IObservable<int> observable = Observable.Create<int>(observer =>
{
    for (int i = 0; i < 3; i++)
        observer.OnNext(i);

    observer.OnCompleted();
    return Disposable.Empty;
});
IDisposable subscription = observable
    .Do(x => Console.WriteLine($"[Do ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .Subscribe(x => Console.WriteLine($"[Subscribe ThreadId] {Thread.CurrentThread.ManagedThreadId}"));
```

```
[Main ThreadId] 8
[Do ThreadId] 8
[Subscribe ThreadId] 8
[Do ThreadId] 8
[Subscribe ThreadId] 8
[Do ThreadId] 8
[Subscribe ThreadId] 8
```

Threading Example #2

```
Console.WriteLine($"[Main ThreadId] {Thread.CurrentThread.ManagedThreadId}");
IObservable<int> observable = Observable.Create<int>(observer =>
{
    Task.Run(() =>
    {
        for (int i = 0; i < 3; i++)
            observer.OnNext(i);

        observer.OnCompleted();
    });
    return Disposable.Empty;
});

IDisposable subscription = observable
    .Do(x => Console.WriteLine($"[Do ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .Subscribe(x => Console.WriteLine($"[Subscribe ThreadId] {Thread.CurrentThread.ManagedThreadId}"));
```

```
[Main ThreadId] 9
[Do ThreadId] 11
[Subscribe ThreadId] 11
[Do ThreadId] 11
[Subscribe ThreadId] 11
[Do ThreadId] 11
[Subscribe ThreadId] 11
```

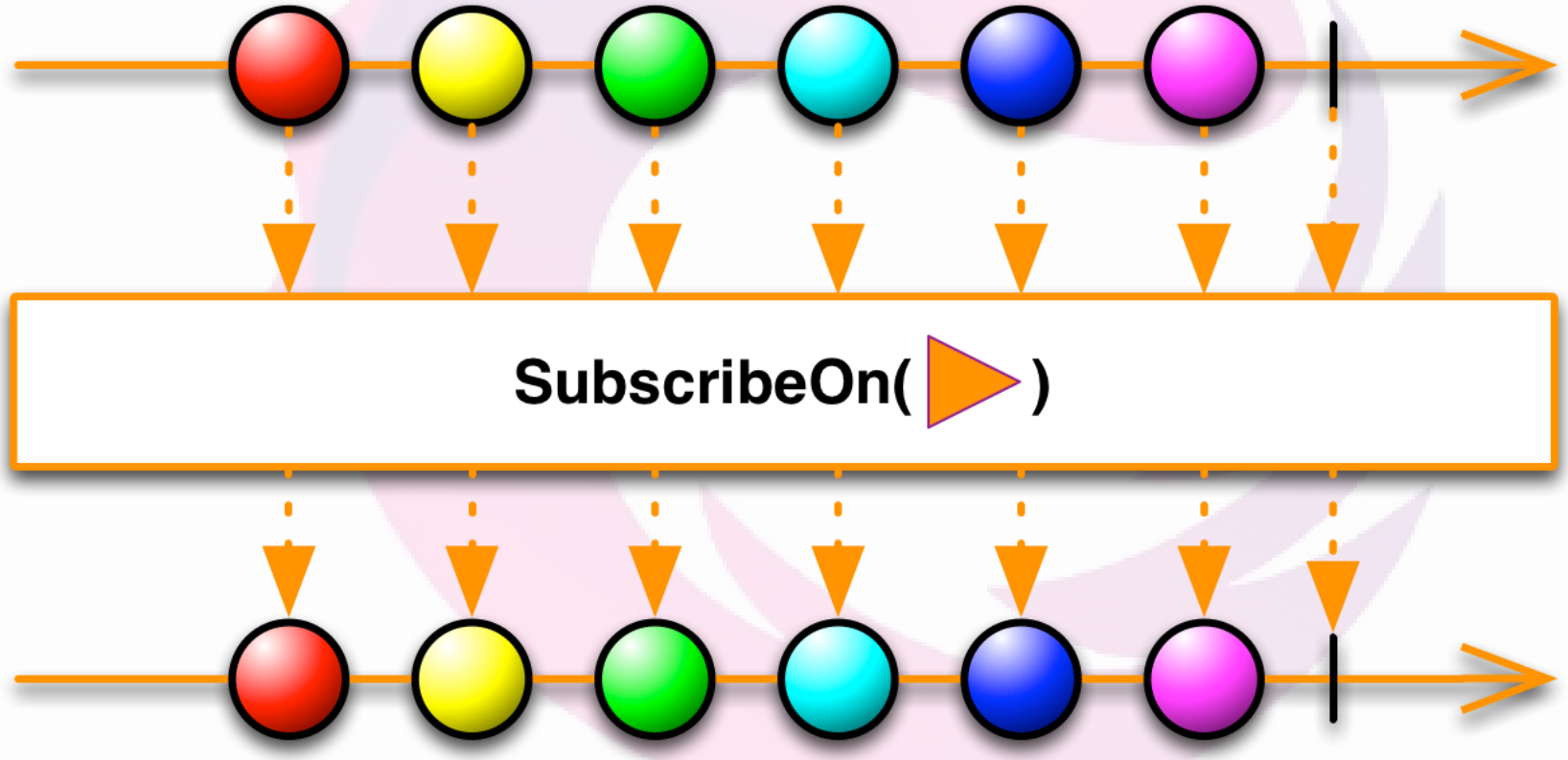
Observable.Interval

```
Console.WriteLine($"[Main ThreadId] {Thread.CurrentThread.ManagedThreadId}");
Observable.Interval(TimeSpan.FromSeconds(1))
    .Do(x => Console.WriteLine($"[Do ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .Take(5)
    .Subscribe(x => Console.WriteLine($"[Subscribe ThreadId] {Thread.CurrentThread.ManagedThreadId}"));
```

```
[Main ThreadId] 9
[Do ThreadId] 11
[Subscribe ThreadId] 11
[Do ThreadId] 12
[Subscribe ThreadId] 12
[Do ThreadId] 11
[Subscribe ThreadId] 11
[Do ThreadId] 12
[Subscribe ThreadId] 12
[Do ThreadId] 11
[Subscribe ThreadId] 11
```

SubscribeOn

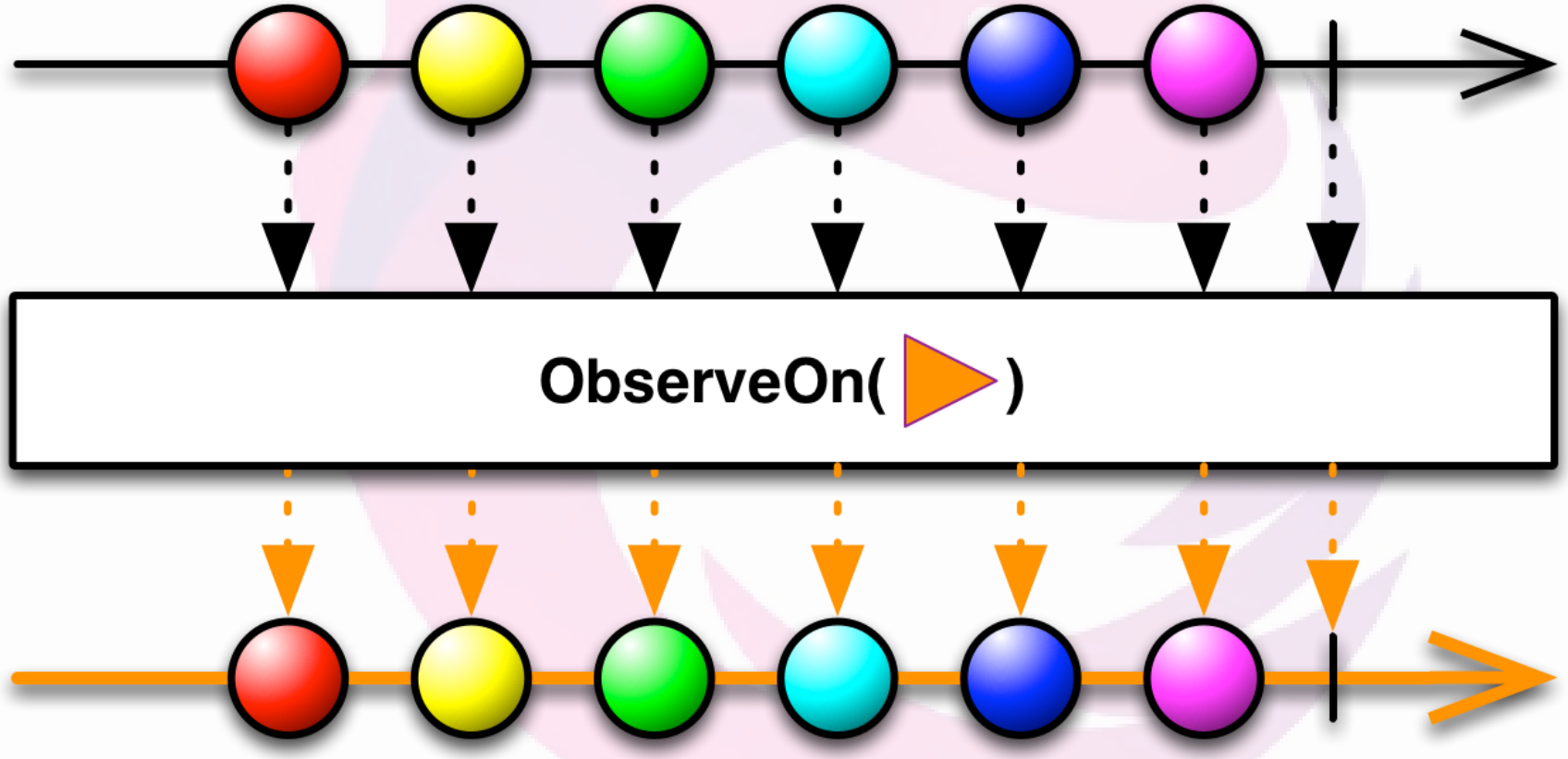
Specify the Scheduler on which an Observable will operate



SubscribeOn operator designates which thread the Observable will begin operating on, no matter at what point in the chain of operators that operator is called

ObserveOn

Specify the Scheduler on which an observer will observe this Observable



ObserveOn affects the thread that the Observable will use below where that operator appears

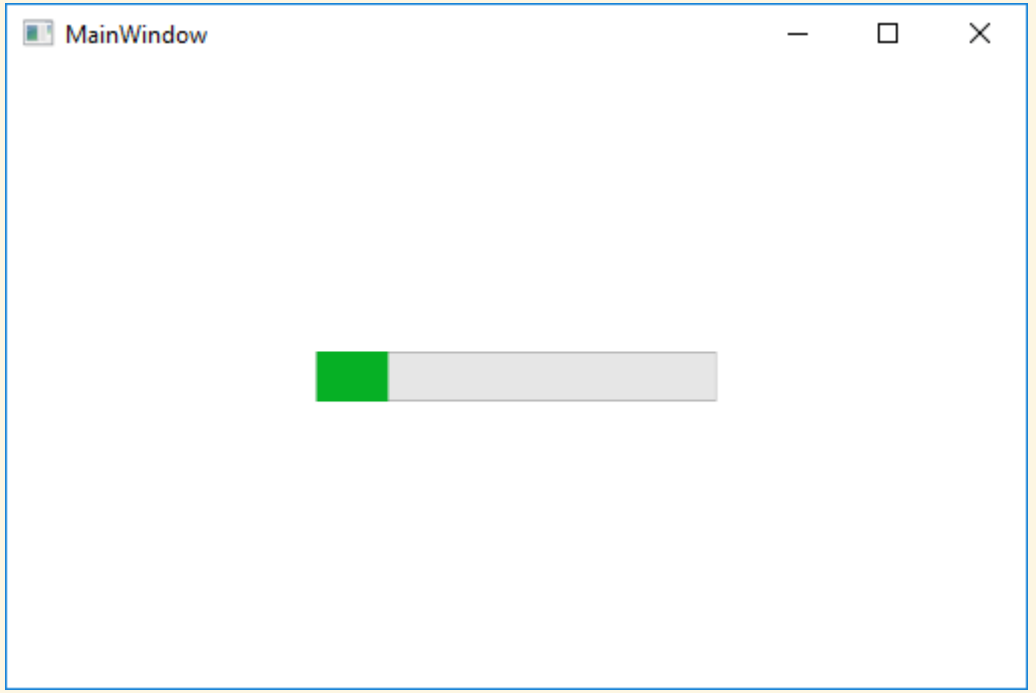
ObserveOn & SubscribeOn Example

```
Console.WriteLine($"[Main ThreadId] {Thread.CurrentThread.ManagedThreadId}");
IObservable<int> observable = Observable.Create<int>(observer =>
{
    Console.WriteLine($"[Start ThreadId] {Thread.CurrentThread.ManagedThreadId}");
    for (int i = 0; i < 3; i++)
        observer.OnNext(i);

    observer.OnCompleted();
    return Disposable.Empty;
});
observable
    .Take(1)
    .SubscribeOn(new NewThreadScheduler())
    .Do(x => Console.WriteLine($"[After SubscribeOn ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .ObserveOn(new NewThreadScheduler())
    .Do(x => Console.WriteLine($"[After ObserveOn #1 ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .ObserveOn(new NewThreadScheduler())
    .Do(x => Console.WriteLine($"[After ObserveOn #2 ThreadId] {Thread.CurrentThread.ManagedThreadId}"))
    .Subscribe(x => Console.WriteLine($"[Subscribe ThreadId] {Thread.CurrentThread.ManagedThreadId}"));
```

```
[Main ThreadId] 1
[Start ThreadId] 4
[After SubscribeOn ThreadId] 4
[After ObserveOn #1 ThreadId] 5
[After ObserveOn #2 ThreadId] 7
[Subscribe ThreadId] 7
```

ObserveOn Dispatcher

```
Observable.Interval(TimeSpan.FromMilliseconds(250))
    .Take(100)
    .ObserveOn(Dispatcher)
    .Subscribe(x =>
    {
        ProgressBar.Value += 1;
    });
```

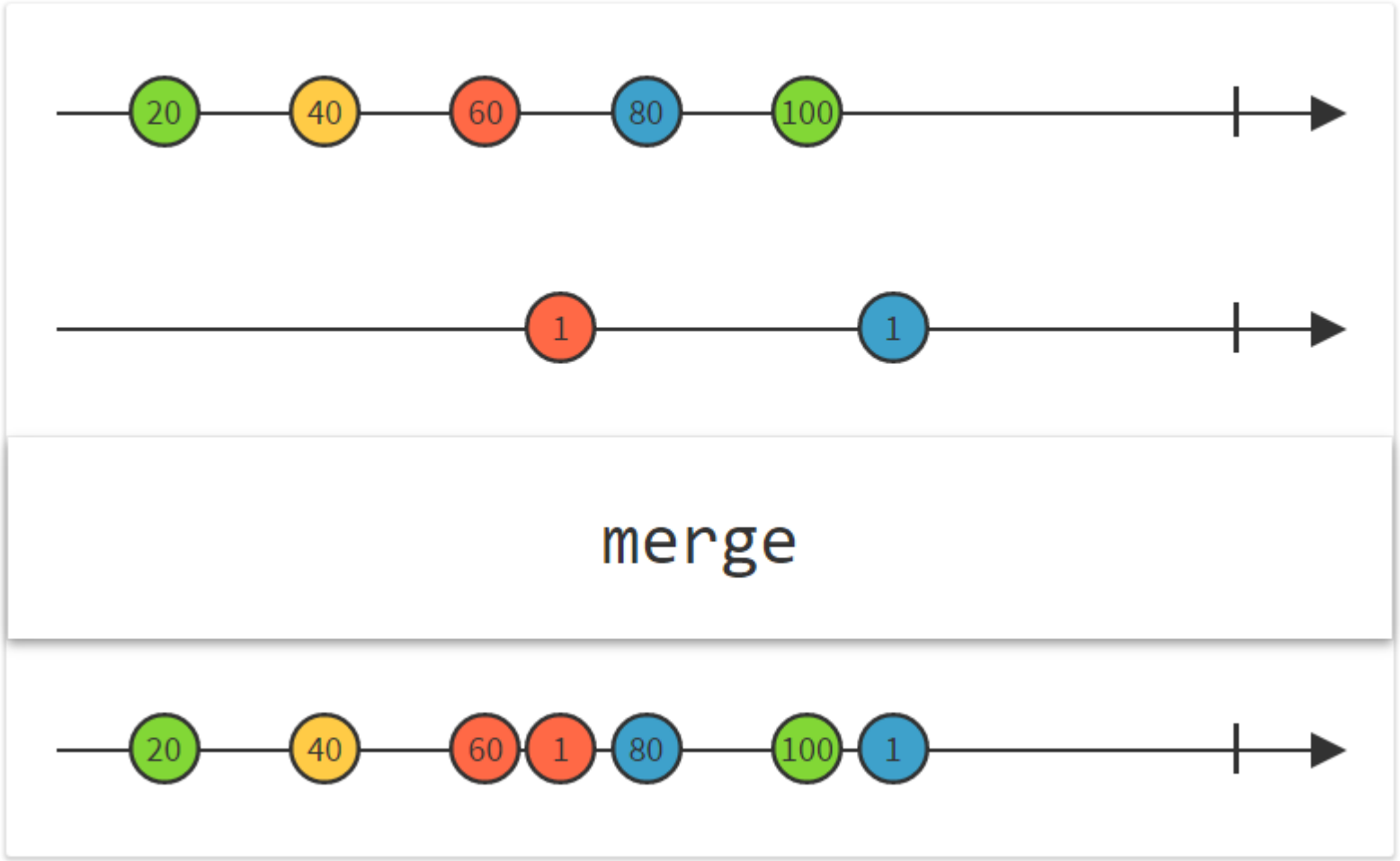


Combining Observables

- And/Then/When
- CombineLatest
- Join
- Merge
- StartWith
- Switch
- Zip

Combining Observables: Merge

Combine multiple Observables into one by merging their emissions



Combining Observables: Merge Example

```
IObservable<string> observable1 = Observable.Interval(TimeSpan.FromSeconds(1))
    .Select(index => $"Observable1: #{index}");

IObservable<string> observable2 = Observable.Interval(TimeSpan.FromSeconds(4))
    .Select(index => $"Observable2: #{index}");

IObservable<string> observable3 = observable1.Merge(observable2);
IDisposable subscription = observable3.Subscribe(Console.WriteLine);
```

```
Observable1: #0
Observable1: #1
Observable1: #2
Observable1: #3
Observable2: #0
Observable1: #4
```

Usage guidelines

- Members that return a sequence should never return null
- Dispose of subscriptions.
- Always provide an OnError handler.
- Break large queries up into parts.
- Name your queries well,
- Avoid creating side effects

Resources & references

- http://introtorx.com
- http://rxwiki.wikidot.com/101samples
- http://reactivex.io/rxjs/
- http://rxmarbles.com
- https://github.com/Reactive-Extensions/Rx.NET/blob/master/Rx.NET/Documentation
- https://github.com/AdaptiveConsulting/ReactiveTrader
- Youtube.com : Reactive Extensions +
  - Erick Mejier
  - Lee Campbell
  - Netflix

Download

github.com/raflop/Rx.NET-Presentation