The Reactive Framework

Agenda

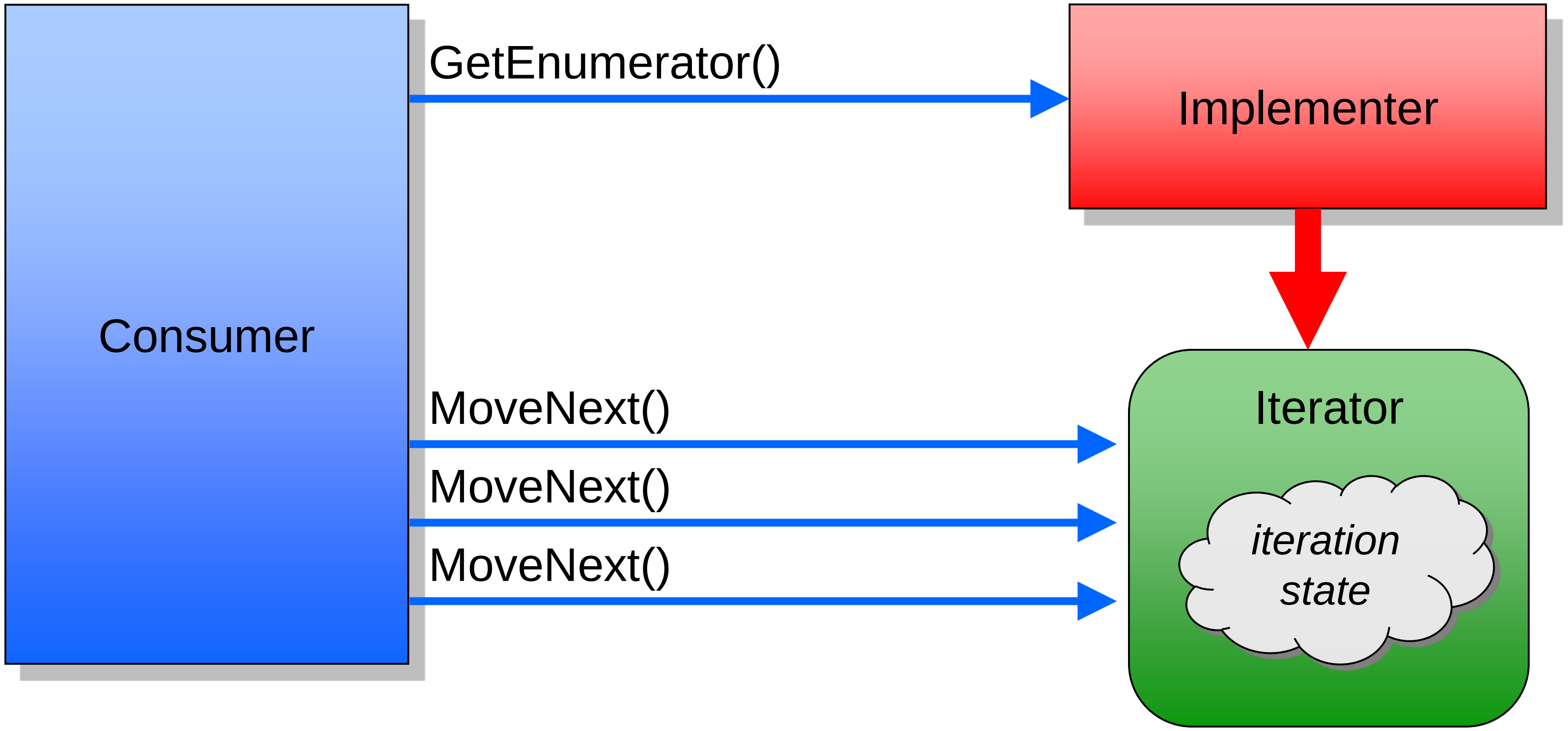
- Iteration
- IEnumerable/IEnumerator
- IObservable/IObserver
- Reactive Framework Extensions
- Example uses

Iteration

- **The ability to consume a series of “things”**
  - Process all items in a collection
  - Read set of rows from a database table
- **Can layer iteration model on top of event streams**
  - Hardware events
  - Stock pricing data streams
  - yield return makes this straightforward
- **Two models**
  - Pull mode
  - Push mode

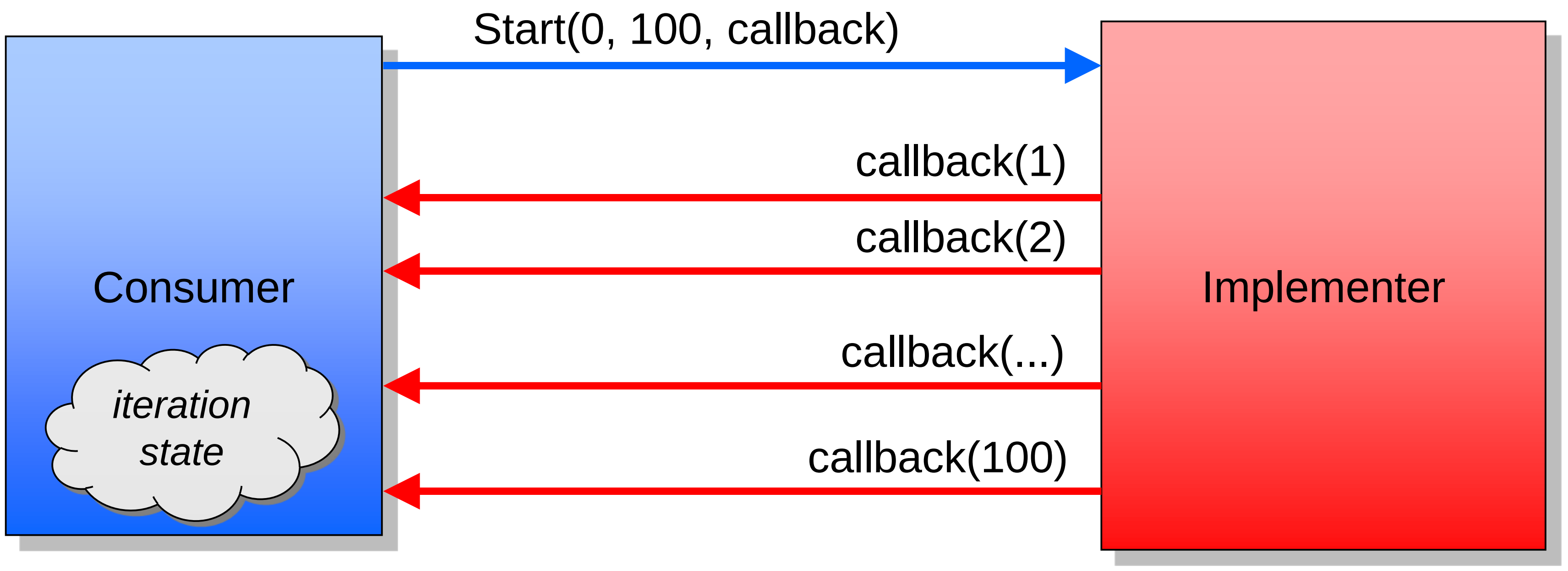
Pull Mode Iteration

- **Could roll our own**
- **Standard model with IEnumerable<T>**
  - Allows rich functionality to be layered on top e.g. LINQ



Ad-hoc Push Mode Iteration

- **Consumer sends implementor a callback**
  - Implemented using delegates
- **Cannot add functionality to ad-hoc implementations**



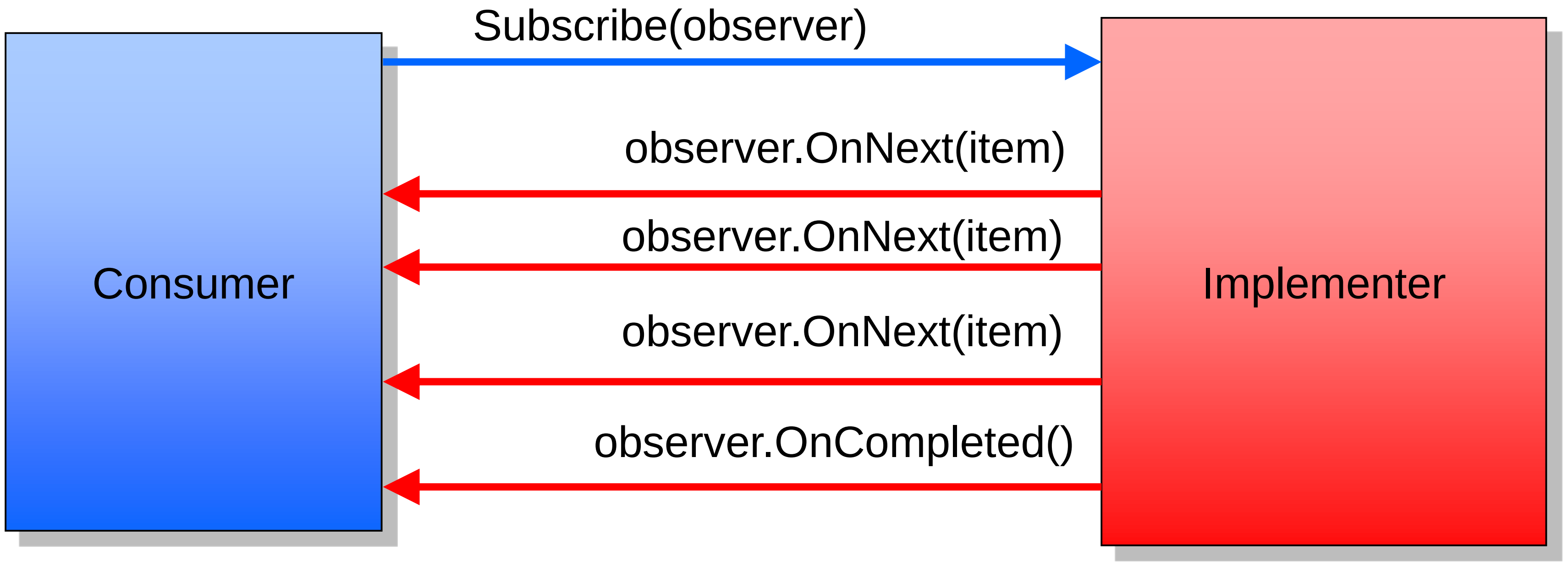
Formalizing Push Mode Iteration

- **.NET 4.0 introduces push mode equivalent of IEnumerable<T>**
  - IObservable<T>
  - IObserver<T>
- **No implementations in 4.0 framework**

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

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

IObservable in Action



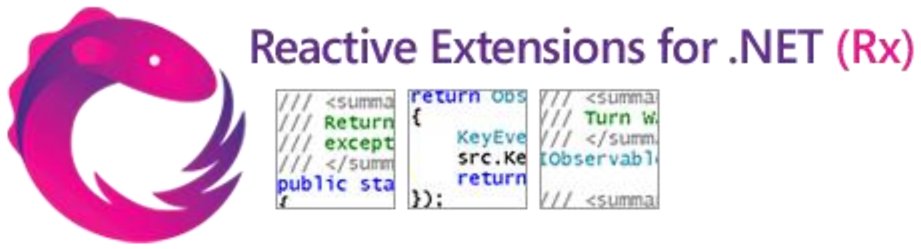
IEnumerable to IObservable Adapter

```
class EnumObserverable<T> : IObservable<T>
{
    IEnumerable<T> source;
    public EnumObserverable(IEnumerable<T> source)
    {
        this.source = source;
    }

    public IDisposable Subscribe(IObserver<T> observer)
    {
        foreach (T item in source)
        {
            observer.OnNext(item);
        }

        observer.OnCompleted();
        return new NullDisposable();
    }
}
```

```
public static IObservable<T> ToObservable<T>(this IEnumerable<T> source)
{
    return new EnumObserverable<T>(source);
}
```

Reactive Framework Extensions (Rx)

- **Library based around IObservable for async programming**
  - Supports .NET 3.5 and 4.0
  - Currently a devlabs project
  - http://msdn.microsoft.com/en-us/devlabs/ee794896.aspx
- **Shipped in 3 assemblies**
  - System.CoreEx: core primitives
  - System.Reactive: IObservable framework
  - System.Interactive: LINQ to Objects extensions



Wrapping IEnumerable<T>

- **Rx provides ToObservable**
- **Subscribe method takes IObserver<T> or delegates**

```
List<int> primes = new List<int>() { 1, 2, 3, 5, 7, 11, 13, 17, 19 };

IObservable<int> ob = primes.ToObservable();

ob.Subscribe(p => Console.WriteLine(p),
             x => Console.WriteLine("Error: {0}", x),
             () => Console.WriteLine("Completed"));
```

Wrapping Tasks

- **Rx fundamentally about async**
  - Task API integration natural extension
- **Observable.Start**
  - Spins up new Task
- **Observable.FromAsyncPattern**
  - Wraps Begin/End async pattern

```
IObservable<Unit> obs = Observable.Start(() =>
{
    Console.WriteLine("async started");
    Thread.Sleep(1000);
    Console.WriteLine("async ended");
});

obs.Subscribe(u => Console.WriteLine("OnNext"), () => Console.WriteLine("Completed"));
```

Wrapping Events

- **Can wrap events**
  - Overloads for EventHandler based events

```
var obs = Observable.FromEvent<MouseEventArgs>(this, "MouseMove");

obs.Subscribe(ie => info.Text = ie.EventArgs.GetPosition(this).X.ToString());
```

Layering in Services

- **With standard model for push based iteration can layer in services**
  - LINQ to Rx

```
var obs = Observable.FromEvent<MouseEventArgs>(this, "MouseMove");

var query = from p in obs
            where p.EventArgs.GetPosition(this).X > 300
            select p;

query.Subscribe(ie => info.Text = ie.EventArgs.GetPosition(this).X.ToString());
```

Composing Observables

```
IObservable<int> obs1 = Observable.Start<int>(() =>
{
    Console.WriteLine("async 1 started");
    Thread.Sleep(1000);
    Console.WriteLine("async 1 ended");
    return 42;
});

IObservable<int> obs2 = Observable.Start<int>(() =>
{
    Console.WriteLine("async 2 started");
    Thread.Sleep(2000);
    Console.WriteLine("async 2 ended");
    return 21;
});

IObservable<int> query = from i1 in obs1
                         from i2 in obs2
                         select i1 + i2;

query.Subscribe(i => Console.WriteLine("Result is {0}", i));
```

Recording Time Intervals

- **TimeInterval extension adds timestamp on to each item**
  - Useful for recording multiple events within timeframe

```
var obs = Observable.FromEvent<RoutedEventArgs>(clickMe, "Click");

obs.TimeInterval().Subscribe(item => MessageBox.Show(item.Interval.ToString()));
```

Buffering

- **Can buffer items by time or count**
  - Processing of dense event streams
  - Grouping sets of events

```
dataStream.BufferWithTime(TimeSpan.FromMilliseconds(500)).Subscribe(ints =>
{
    Console.WriteLine(ints.Count);
});
```

Sampling

- **Sampling used when trends are more important than values**
  - Takes items from data stream periodically for processing

```
sensor.GetTemperatures().ToObservable().Sample(new TimeSpan(0, 0, 0, 0, 100))
                                       .BufferWithTime(new TimeSpan(0, 0, 5));
```

Conclusion

- IObservable useful when data is asynchronous
- Rx gives a framework for working with async data
- LINQ integration and extension methods provides rich model over standard interface