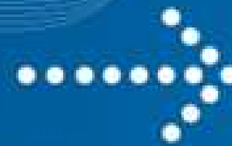
Reactive Extensions (Rx)

Your prescription to cure event processing blues

Bart J.F. De Smet

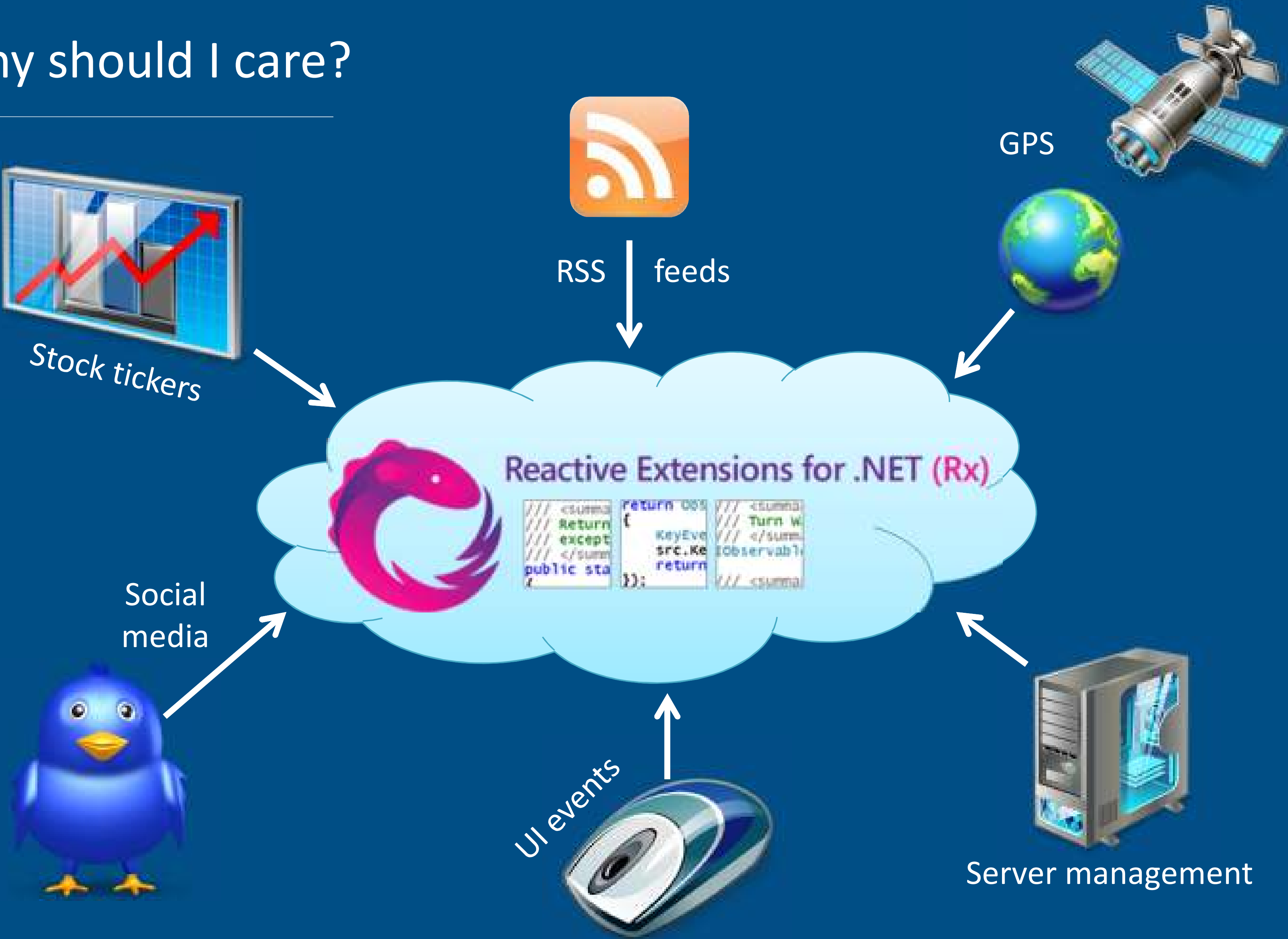
Software Development Engineer

bartde@microsoft.com



PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24

goto; conference


Why should I care?
RSS
feeds
GPS
Stock tickers
Reactive Extensions for .NET (Rx)
Social
media
UI events
Server management

Event Processing Systems

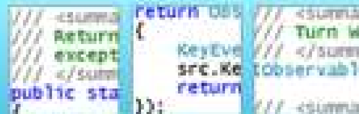
Way ***simpler*** with Rx

$(f \circ g)(x) = f(g(x))$

Rx is a library for composing asynchronous and event-based programs using observable sequences.

Queries! **LINQ!**

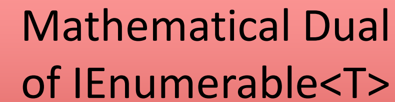
Reactive Extensions for .NET (Rx)



- .NET 3.5 SP1, 4.0, and 4.5
- Silverlight 4, and 5
- Windows Phone 7 and 7.5
- JavaScript (RxJS)

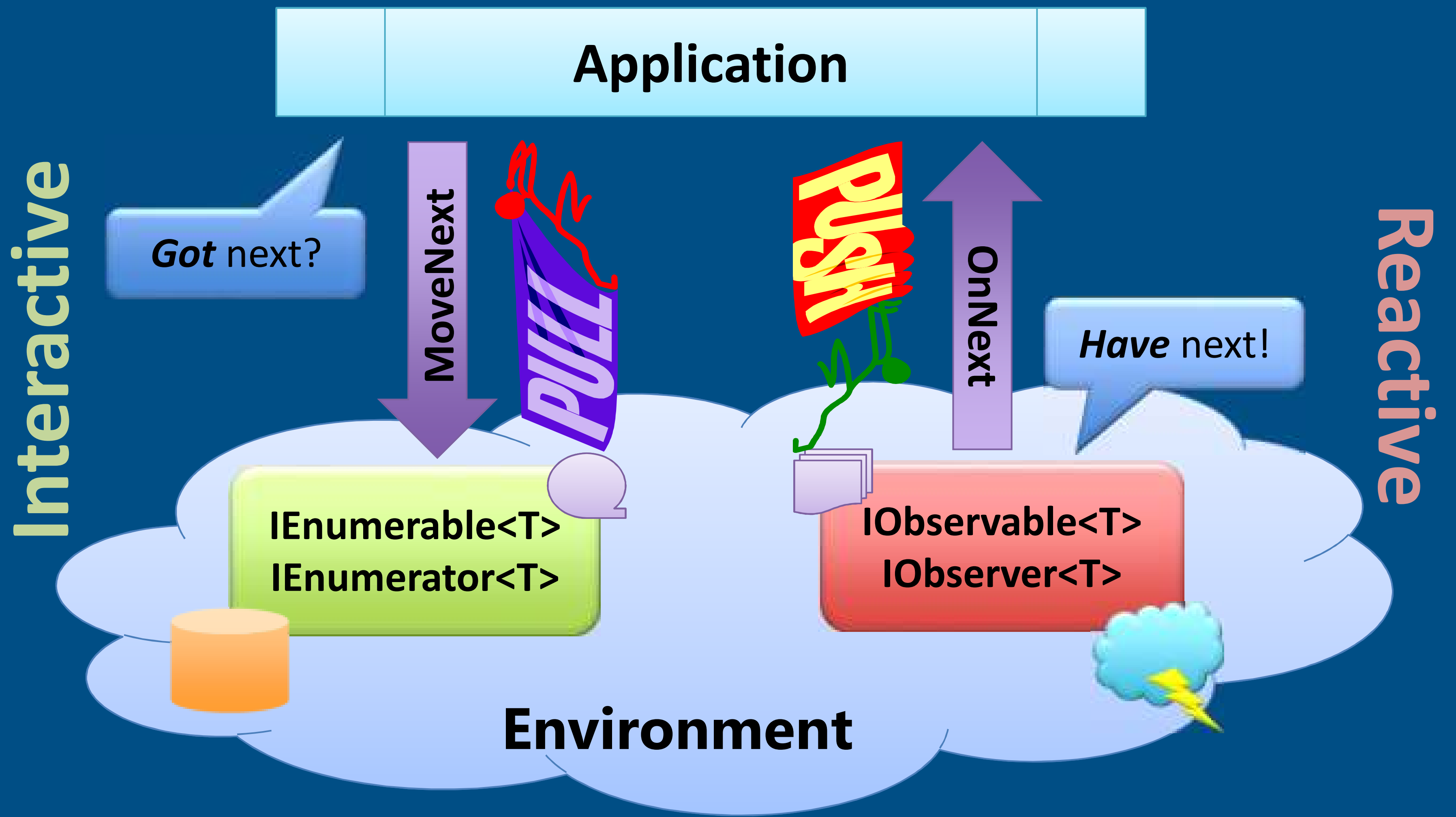
Download at **MSDN Data Developer Center** or use **NuGet**

Observable Sequences




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

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

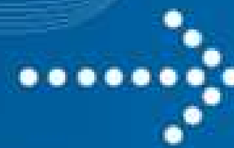
Push-Based Data Retrieval



DEMO

The IObservable<T> interface

PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24



goto; conference

Creating Observable Sequences

OnNext* [OnError | OnCompleted]

```
Observable.Never<int>()
```

```
Observable.Empty<int>()
```

OnCompleted

```
Observable.Return<int>(42)
```

OnNext(42)

```
Observable.Throw<int>(ex)
```

OnError(ex)

Generator Functions

```
var o = Observable.Generate(
    0,
    i => i < 10,
    i => i + 1,
    i => i * i
);
```

Asynchronous

```
o.Subscribe(x => {
    Console.WriteLine(x);
});
```

```
var e = new IEnumerable<int> {
    for (int i = 0;
          i < 10;
          i++)
      yield return i * i;
};
```

Synchronous

```
foreach (var x in e) {
    Console.WriteLine(x);
}
```

The Create operator

```
Observable<int> o = Observable.Create<int>(observer => {
  // Assume we introduce concurrency (see later)…
  observer.OnNext(42);
  observer.OnCompleted();
  return () => { /* unsubscribe action */ };
});

IDisposable subscription = o.Subscribe(
  onNext:      x  => { Console.WriteLine("Next: " + x); },
  onError:     ex => { Console.WriteLine("Oops: " + ex); },
  onCompleted: () => { Console.WriteLine("Done"); }
);
```

C# 4.0 named parameter syntax

C# doesn't have ***anonymous interface implementation****, so we provide various extension methods that take lambdas.*

The Create operator

```
Observable<int> o = Observable.Create<int>(observer => {
  // Assume we introduce concurrency (see later)…
  observer.OnNext(42);
  observer.OnCompleted();
  return () => { /* unsubscribe action */ };
});

IDisposable subscription = o.Subscribe(
  onNext:      x  => { Console.WriteLine("Next: " + x); },
  onError:     ex => { Console.WriteLine("Oops: " + ex); },
  onCompleted: () => { Console.WriteLine("Done"); }
);

Thread.Sleep(30000);
```

F10

The Create operator

```
Observable<int> o = Observable.Create<int>(observer => {
  // Assume we introduce concurrency (see later)…
  observer.OnNext(42);
  observer.OnCompleted();
  return () => { /* unsubscribe action */ };
});
```

```
IDisposable subscription = o.Subscribe(
  onNext:      x  => { Console.WriteLine("Next: " + x); },
  onError:     ex => { Console.WriteLine("Oops: " + ex); },
  onCompleted: () => { Console.WriteLine("Done"); }
);

Thread.Sleep(30000);
```

F10

The Create operator

```
Observable<int> o = Observable.Create<int>(observer => {
  // Assume we introduce concurrency (see later)…
  observer.OnNext(42);
  observer.OnCompleted();
  return () => { /* unsubscribe action */ };
});

IDisposable subscription = o.Subscribe(
  onNext:      x  => { Console.WriteLine("Next: " + x); },
  onError:     ex => { Console.WriteLine("Oops: " + ex); },
  onCompleted: () => { Console.WriteLine("Done"); }
);

Thread.Sleep(30000);
```

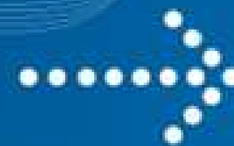
F5

The Create operator

```
Observable<int> o = Observable.Create<int>(observer => {
  // Assume we introduce concurrency (see later)…
  observer.OnNext(42);
  observer.OnCompleted();
  return () => { /* unsubscribe action */ };
});

IDisposable subscription = o.Subscribe(
  onNext:      x  => { Console.WriteLine("Next: " + x); },
  onError:     ex => { Console.WriteLine("Oops: " + ex); },
  onCompleted: () => { Console.WriteLine("Done"); }
);

Thread.Sleep(30000);
```

DEMO

Creating observable sequences



The Trouble with .NET Events

How to pass around?

Hidden data source

```
form1.MouseMove += (sender, args) => {
    if (args.Location.X == args.Location.Y)
        // I'd like to raise another event
};
```

Lack of composition

```
form1.MouseMove -= /* what goes here? */
```

Resource maintenance?

Observable Sequences are First-Class Objects

Objects can be passed

Source of Point values

```
IObservable<Point> mouseMoves =
        Observable.FromEvent(frm, "MouseMove");

var filtered = mouseMoves
                  .Where(pos => pos.X == pos.Y);
```

Can define operators

```
var subscription = filtered.Subscribe(…);
subscription.Dispose();
```

Resource maintenance!

Other Conversions with IObservable<T>

- Asynchronous Programming Model (APM)
  - Legacy support; prefer going through Task<T>

```
Func<int, IObservable<int>> computeAsync =
      Observable.FromAsyncPattern(svc.BeginCompute,
                                  svc.EndCompute);
```

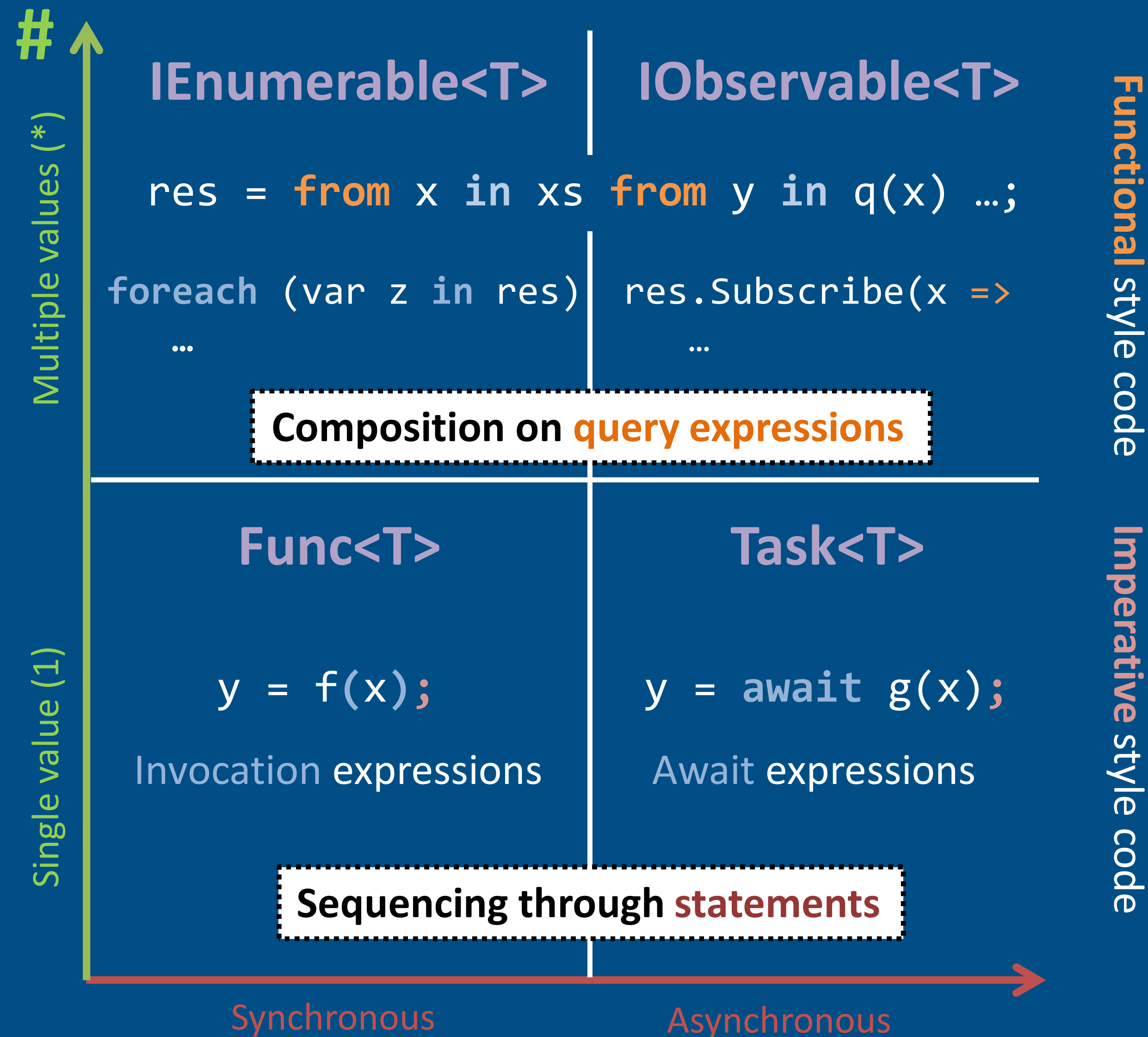
- Task<T>
  - Way to represent single-value asynchrony

```
Task<string> htmlTask =
    webClient.DownloadStringTaskAsync(uri)
    .ToObservable();
```

- IEnumerable<T>
  - Pull-to-push conversion

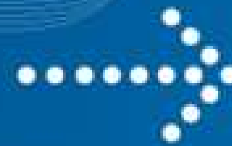
```
IObservable<int> oneToNine =
    Enumerable.Range(0, 10)
    .ToObservable();
```

Asynchronous Data Processing Overview



DEMO

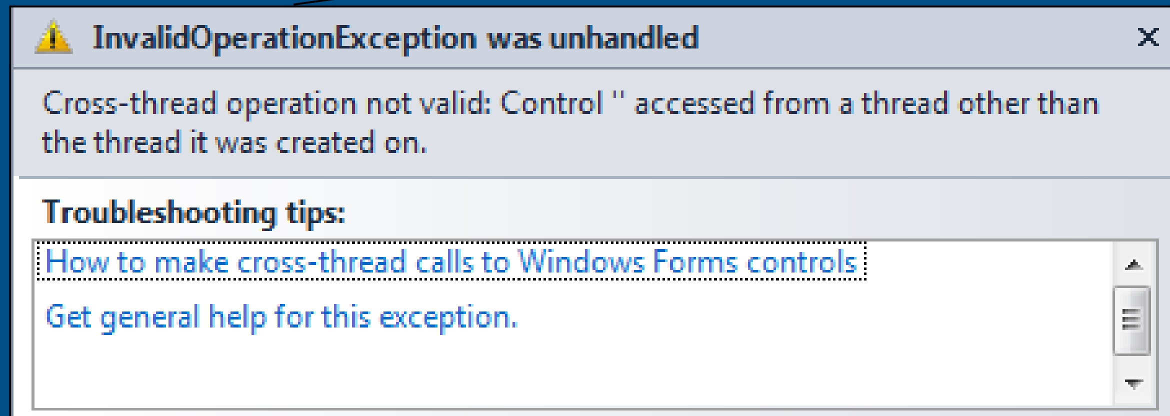
Conversions with IObservable<T>



PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24
goto; conference

Where is the Concurrency?

```
var ticks = Observable.Interval(TimeSpan.FromSeconds(1));
ticks.Subscribe(_ => clock.Text = DateTime.Now.ToString());
```



Schedulers parameterize on **concurrency**

```
var ticks = Observable.Interval(TimeSpan.FromSeconds(1),
                                Scheduler.ThreadPool);

ticks.ObserveOn(new ControlScheduler(clock))
     .Subscribe(_ => clock.Text = DateTime.Now.ToString());
```

What are Schedulers?

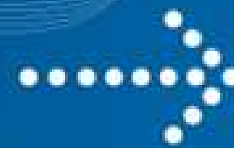
- Single abstraction for **concurrency**
  - `new Thread(() => { … }).Start()`
  - `ThreadPool.QueueUserWorkItem(_ => { … }, null);`
  - `Task.Factory.StartNew(() => { … });`
  - `Dispatcher.BeginInvoke(() => { … });`
- Provide a notion of **time**
  - Contains a clock
  - Allows to schedule work with relative or absolute time

```
interface IScheduler
{
    DateTimeOffset Now { get; }
    IDisposable Schedule<T>(T state,
                            Func<IScheduler, T, IDisposable> action);
    IDisposable Schedule<T>(T state, TimeSpan dueTime,
                            Func<IScheduler, T, IDisposable> action);
    IDisposable Schedule<T>(T state, DateTimeOffset dueTime,
                            Func<IScheduler, T, IDisposable> action);
}
```

DEMO

Using the IScheduler construct

PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24



goto; conference

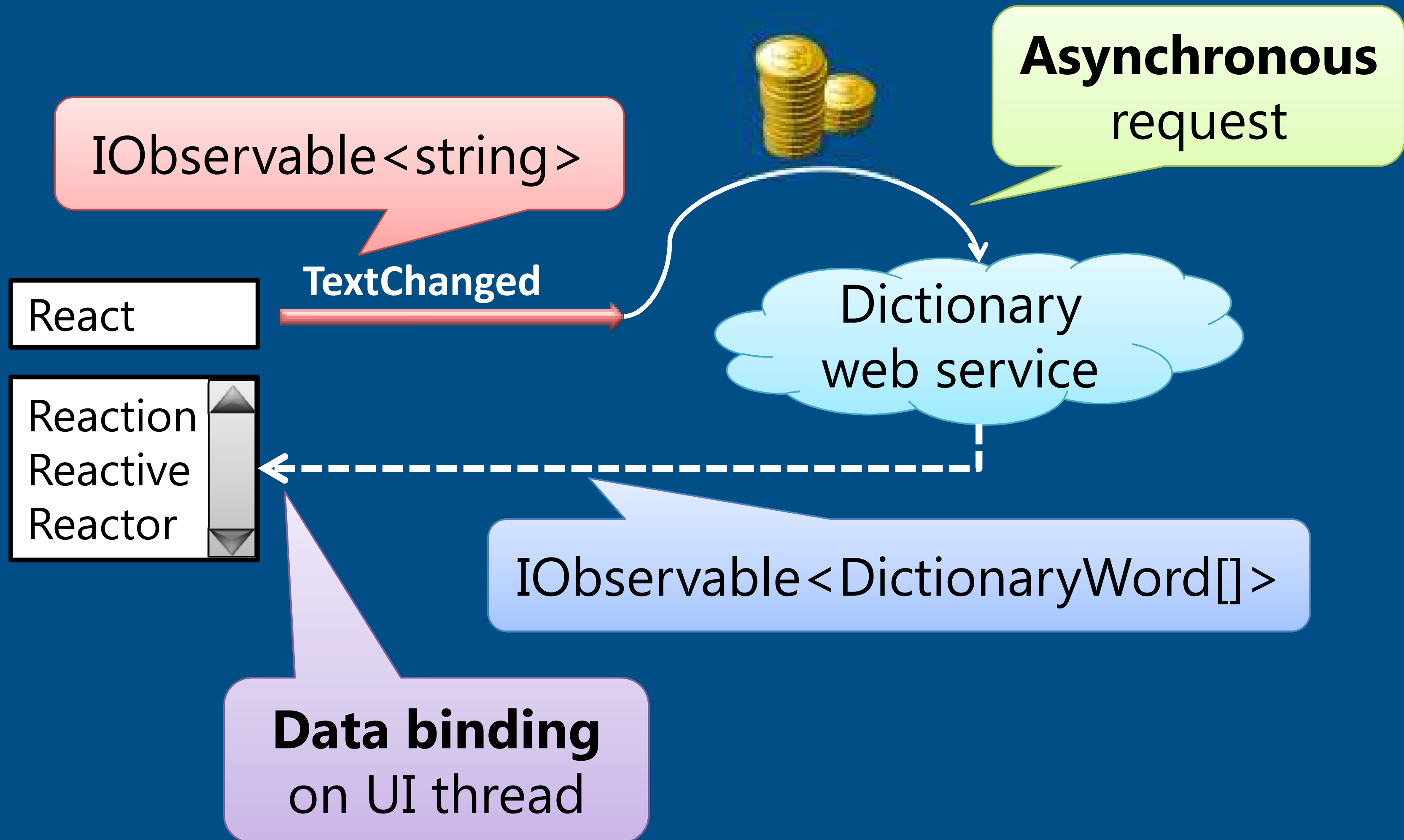
Querying Observable Sequences

- Using LINQ
  - Support for Standard Query Operators

```
IObservable<EventPattern<MouseEventArgs>> moves =
   Observable.FromEventPattern<MouseEventArgs>(frm, “MouseMove”);

IObservable<Point> diagonal = from move in moves
                              let point = move.EventArgs.Location
                              where point.X == point.Y
                              select point;

diagonal.Subscribe(point => {
    // Process the event
});
```

- Time-based operations
  - Natural to the domain of event streams
    - Time-out, delay, throttle
    - Windows with time duration

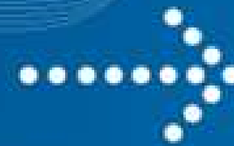
Composition and Querying



DEMO

Querying Observable Sequences using LINQ

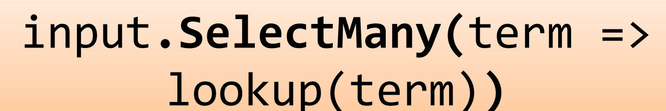
PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24



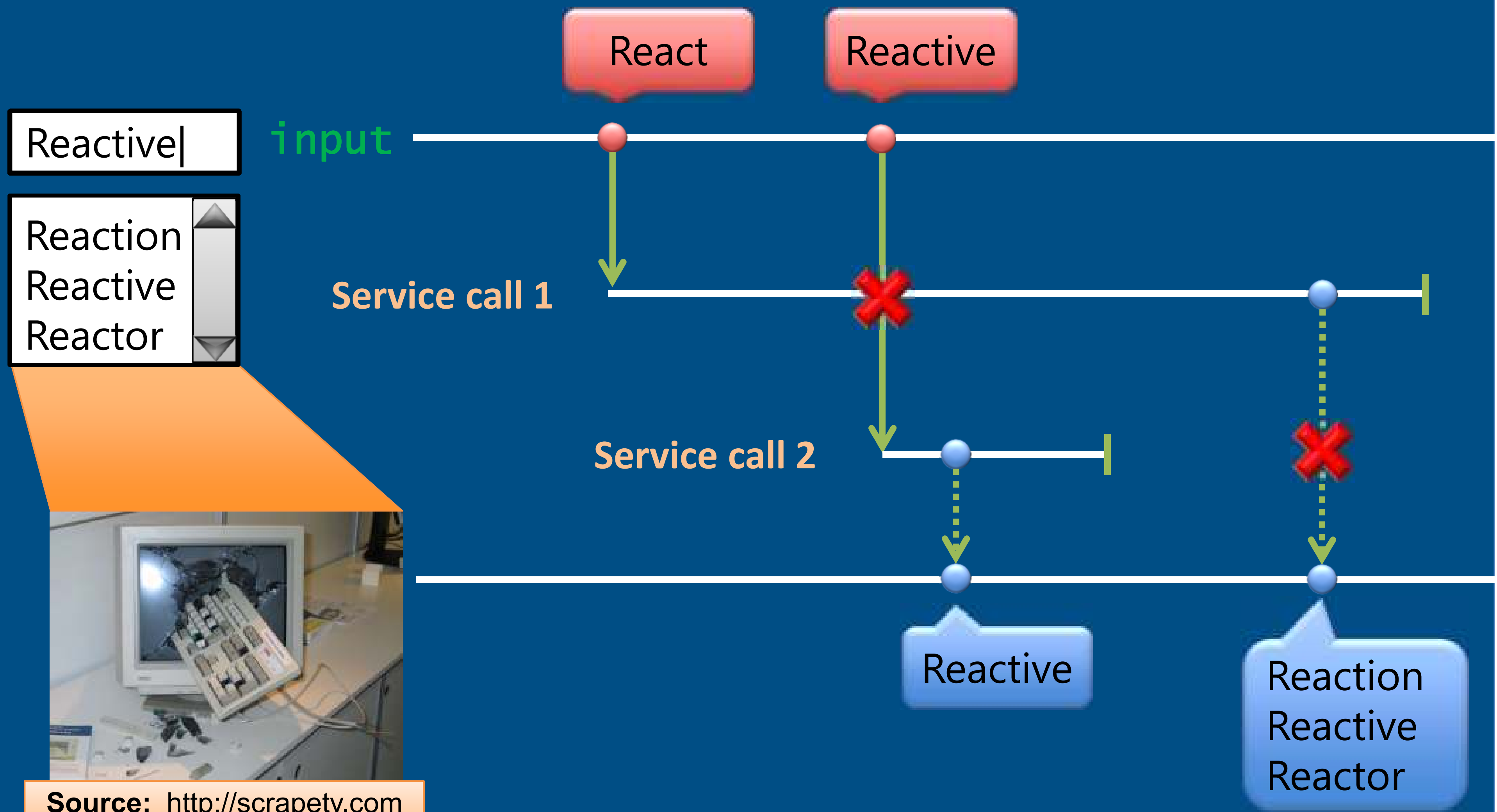
goto; conference

Composition and Querying

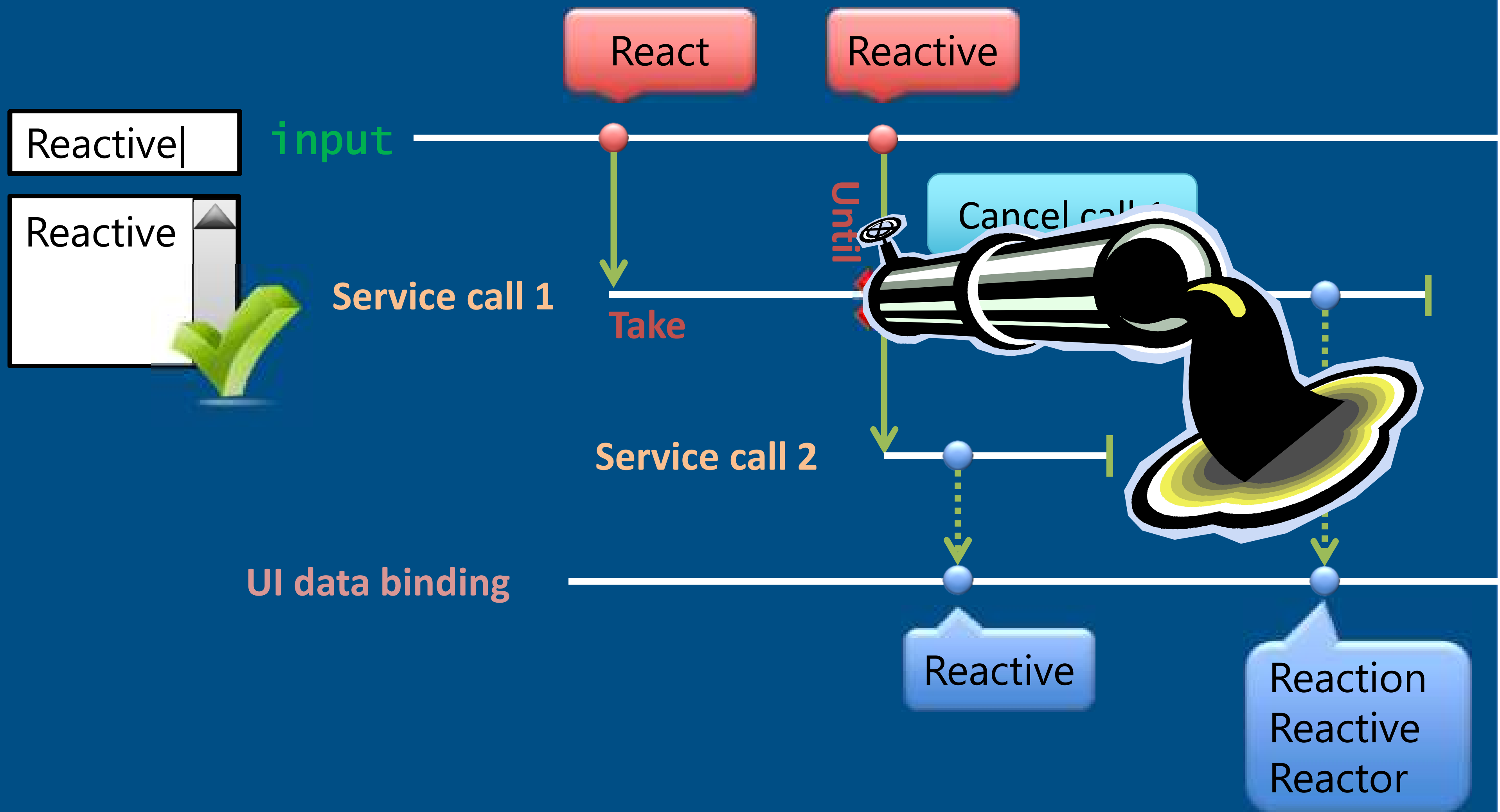
```
// IObservable<string> from TextChanged events
var changed = Observable.FromEvent(txt, "TextChanged");
var input = (from text in changed
             select ((TextBox)text.Sender).Text);
            .DistinctUntilChanged()
            .Throttle(TimeSpan.FromSeconds(1));

// Bridge with the dictionary web service
var svc = new DictServiceSoapClient();
var lookup = Observable.FromAsyncPattern<string, DictionaryWord[]>
                                    (svc.BeginLookup, svc.EndLookup);

// Compose both sources using SelectMany
var res = from term in input
          from words in lookup(term)
          select words;
```



Beware of Concurrent Requests



Beware of Concurrent Requests



Composition to the Rescue

```
// IObservable<string> from TextChanged events
var changed = Observable.FromEvent(txt, "TextChanged");
var input = (from text in changed
             select ((TextBox)text.Sender).Text);
            .DistinctUntilChanged()
            .Throttle(TimeSpan.FromSeconds(1));

// Bridge with the dictionary web service
var svc = new DictServiceSoapClient();
var lookup = Observable.FromAsyncPattern<string, DictionaryWord[]>
                                 (svc.BeginLookup, svc.EndLookup);

// Compose both sources using SelectMany
var res = from term in input
          from words in lookup(term).TakeUntil(input)
          select words;
```

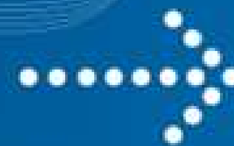
Composition to the Rescue – Alternative Solution

```
// IObservable<string> from TextChanged events
var changed = Observable.FromEvent(txt, "TextChanged");
var input = (from text in changed
             select ((TextBox)text.Sender).Text);
            .DistinctUntilChanged()
            .Throttle(TimeSpan.FromSeconds(1));

// Bridge with the dictionary web service
var svc = new DictServiceSoapClient();
var lookup = Observable.FromAsyncPattern<string, DictionaryWord[]>
                                 (svc.BeginLookup, svc.EndLookup);

// Using Switch to flatten nested sequences
var res = (from term in input
           select lookup(term))
          .Switch();
```

DEMO

Taming the Concurrency Monster

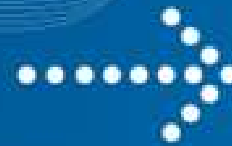
PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24



goto;
conference

THANK YOU

PRAGUE
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Conference : Nov. 22-23 // Training : Nov. 24



goto; conference