London 2011

Tutorials: March 7-8
Conference: March 9-11

www.qconlondon.com

QCon

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

LINQ, Take Two

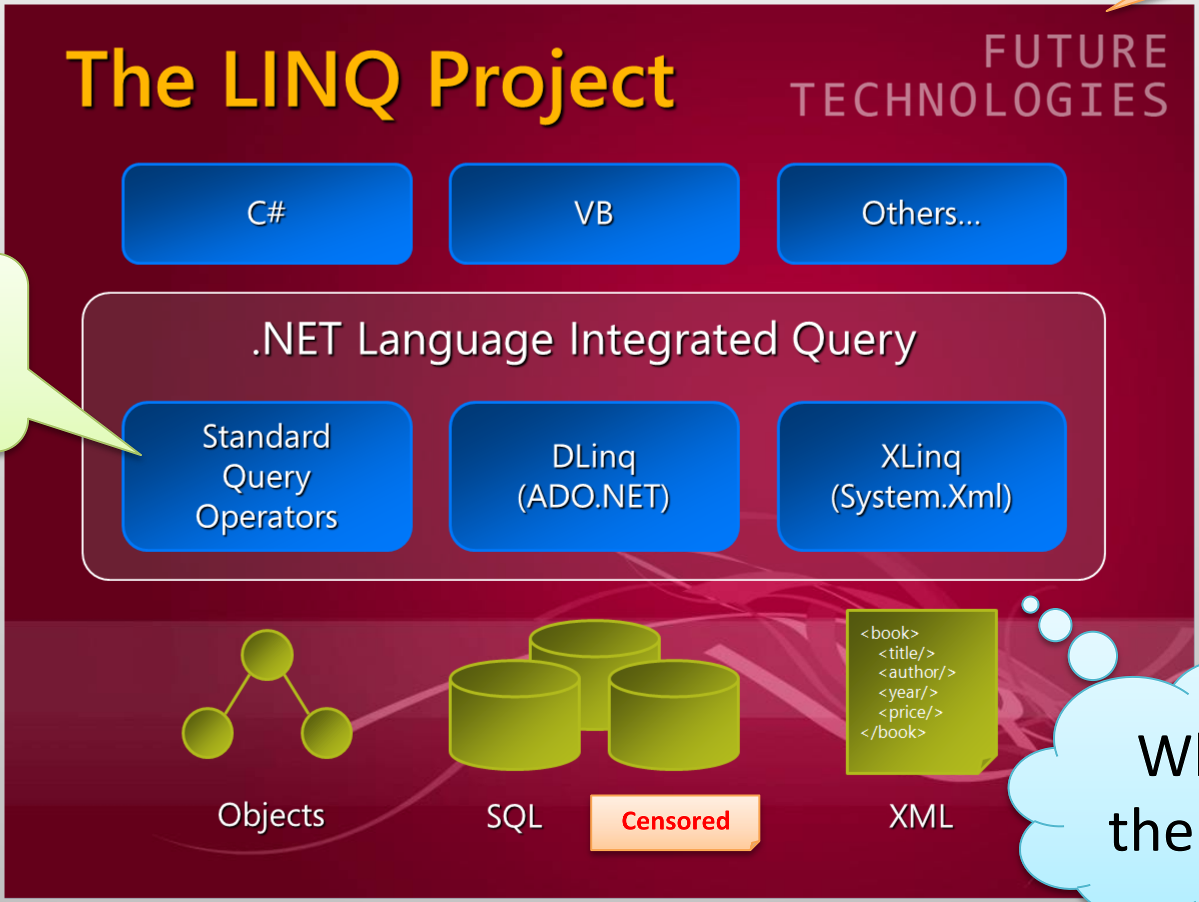
Realizing the LINQ to Everything Dream

Bart J.F. De Smet
Microsoft Corporation
bartde@microsoft.com

What’s LINQ?

A historical perspective

5 years ago



The LINQ Project

FUTURE TECHNOLOGIES

C#

VB

Others…

.NET Language Integrated Query

Standard Query Operators

DLinq (ADO.NET)

XLinq (System.Xml)

Objects

SQL

XML

```
<book>
   <title/>
   <author/>
   <year/>
   <price/>
</book>
```

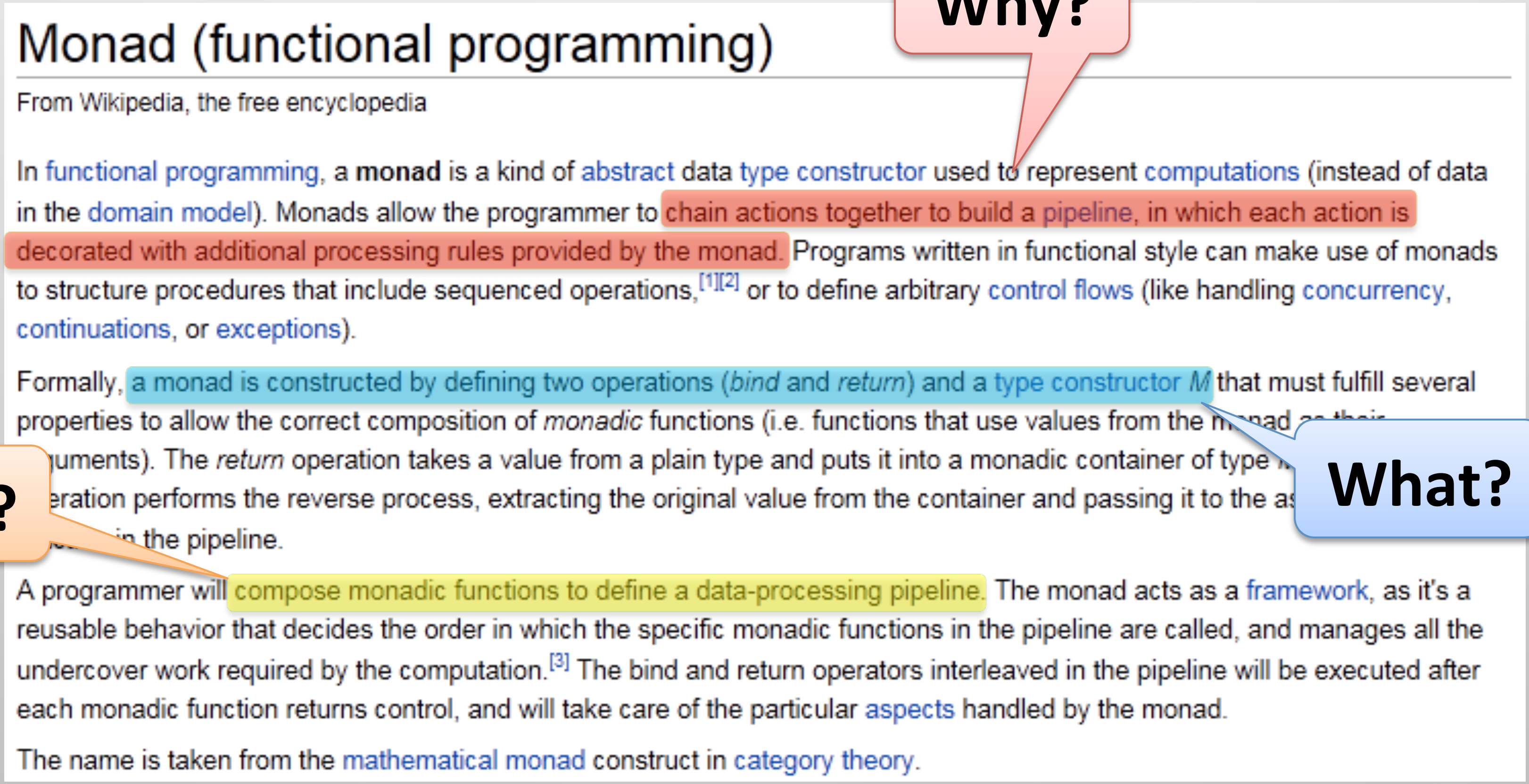
Little recent innovation

Censored

Where’s the ***cloud***?

Essential LINQ

Language Integrated Monads



Why?

What?

How?

Monad (functional programming)

From Wikipedia, the free encyclopedia

In functional programming, a **monad** is a kind of abstract data type constructor used to represent computations (instead of data in the domain model). Monads allow the programmer to chain actions together to build a pipeline, in which each action is decorated with additional processing rules provided by the monad. Programs written in functional style can make use of monads to structure procedures that include sequenced operations,[1][2] or to define arbitrary control flows (like handling concurrency, continuations, or exceptions).

Formally, a monad is constructed by defining two operations (*bind* and *return*) and a type constructor *M* that must fulfill several properties to allow the correct composition of *monadic* functions (i.e. functions that use values from the monad as their arguments). The *return* operation takes a value from a plain type and puts it into a monadic container of type M [illegible] operation performs the reverse process, extracting the original value from the container and passing it to the as[illegible] next function in the pipeline.

A programmer will compose monadic functions to define a data-processing pipeline. The monad acts as a framework, as it's a reusable behavior that decides the order in which the specific monadic functions in the pipeline are called, and manages all the undercover work required by the computation.[3] The bind and return operators interleaved in the pipeline will be executed after each monadic function returns control, and will take care of the particular aspects handled by the monad.

The name is taken from the mathematical monad construct in category theory.

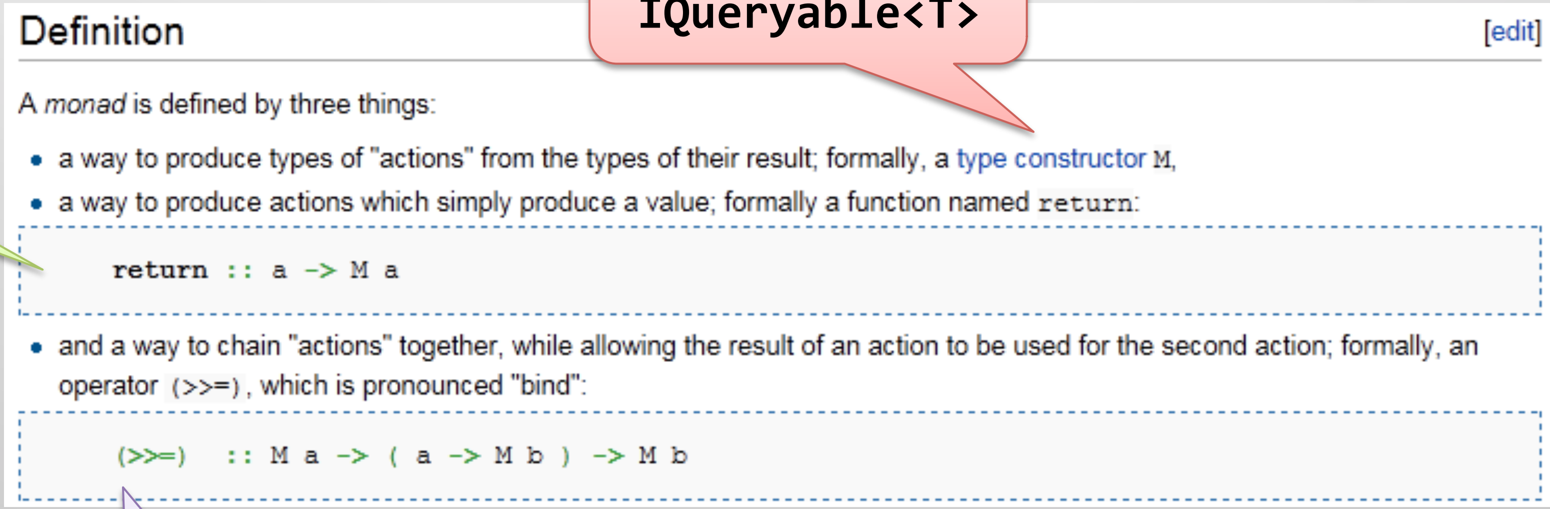
Essential LINQ

The monadic Bind operator

Could there be ***more***?

```
IEnumerable<T>
IQueryable<T>
```

```
new[]
{ 42 }
```



Definition [edit]

A *monad* is defined by three things:

- a way to produce types of "actions" from the types of their result; formally, a type constructor M,
- a way to produce actions which simply produce a value; formally a function named `return`:

```
return :: a -> M a
```

- and a way to chain "actions" together, while allowing the result of an action to be used for the second action; formally, an operator `(>>=)`, which is pronounced "bind":

```
(>>=)  :: M a -> ( a -> M b ) -> M b
```

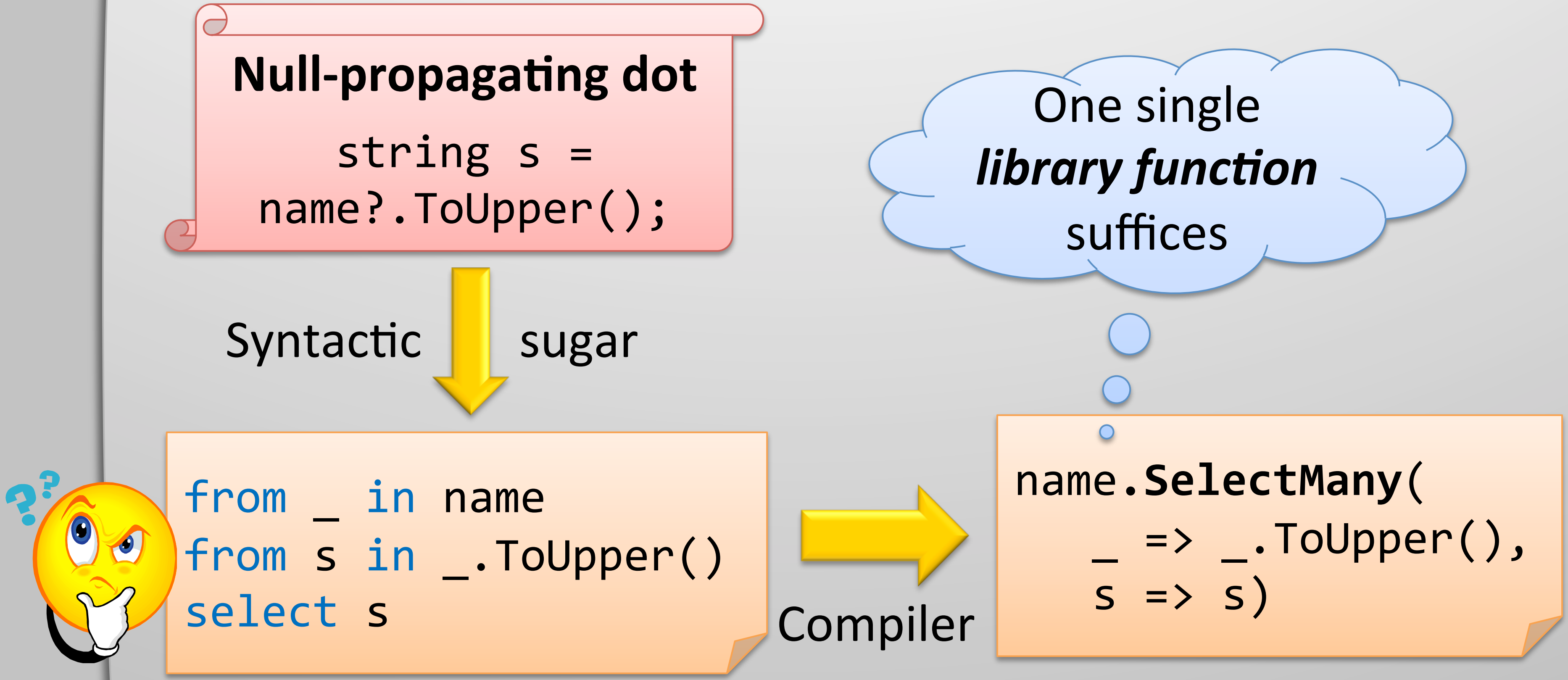
SelectMany

```
IEnumerable<R> SelectMany<T, R>(
    this IEnumerable<T> source,
    Func<T, IEnumerable<R>> selector)
```

Also see www.codeplex.com/LINQSQO for “Project MinLINQ”

Essential LINQ

Maybe monad (for fun and no profit)



Null-propagating dot

```
string s =
name?.ToUpper();
```

Syntactic sugar

```
from _ in name
from s in _.ToUpper()
select s
```

Compiler

```
name.SelectMany(
    _ => _.ToUpper(),
    s => s)
```

One single *library function* suffices

Demo

Query Providers Revisited

Why do we have IQueryable<T>?

Implements **`IQueryable<T>`**

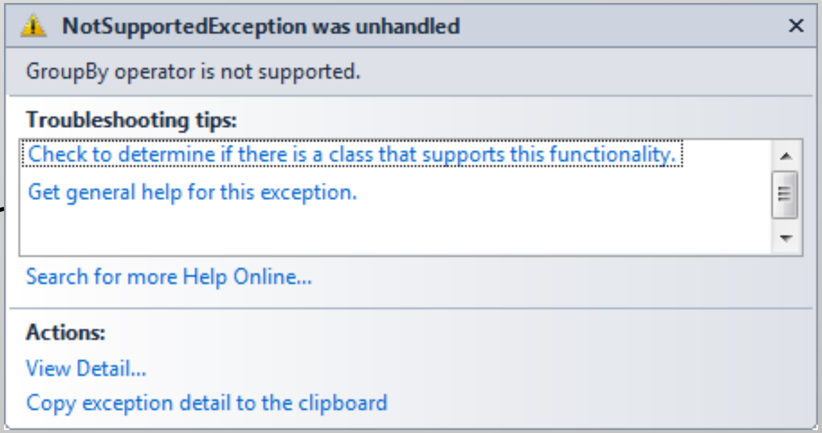
Does it really have to be a runtime check?

```
var src = new Source<Product>();

var res = from p in src
          where p.Price > 100m
          group p by p.Category

foreach (var p in res)
    Console.WriteLine(p);
```

Compiles fine



NotSupportedException was unhandled

GroupBy operator is not supported.

Troubleshooting tips:

Check to determine if there is a class that supports this functionality.

Get general help for this exception.

Search for more Help Online...

Actions:

View Detail...

Copy exception detail to the clipboard

Query Providers Revisited

Leveraging the query pattern

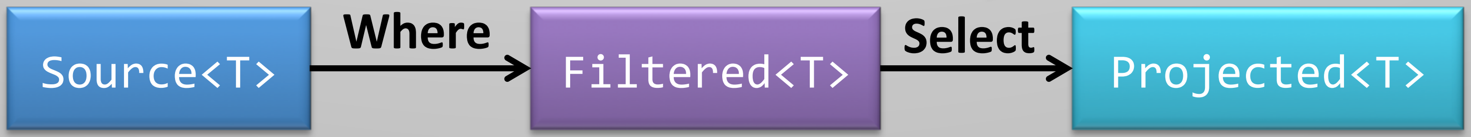
```
var res = from p in src
          where p.Price > 100m
          group p by p.Category;
```

Syntactic sugar

```
var res = src                              ✓
          .Where(p => p.Price > 100m)      ✓
          .GroupBy(p => p.Category);       STOP
```

Can be instance methods

No **GroupBy** “edge”



Query Providers Revisited

Taking it one step further

```
var res = from tweet in twitter
          where tweet.From == "B
          where tweet.About == "
          select tweet;
```

From
About
Location

Query *"learns"*

```
class TweetAboutFromLoc
{
  public FromString From;
  public AboutString About;
  public LocString Location;
}

class FromString
{
  FilterFrom operator ==(
    FromString f, string s)
}
```

"Has a" type

```
class TwitterByFrom
{
    public TwitterByAbout Where(TweetAboutLoc, FilterAbout filter);
    // Other filter methods
}   // Fields with current filters
}
```

Custom *syntax trees*

Demo

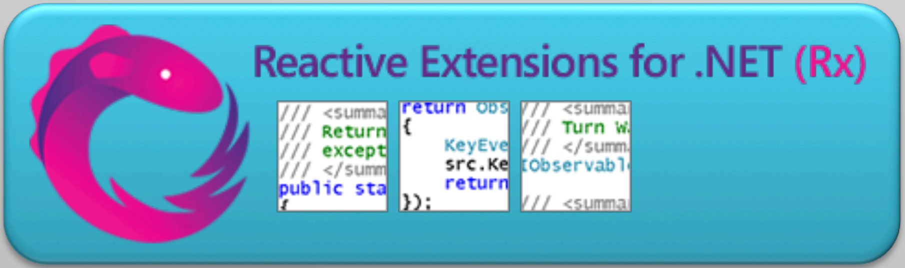
Asynchronous Data Access

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!**

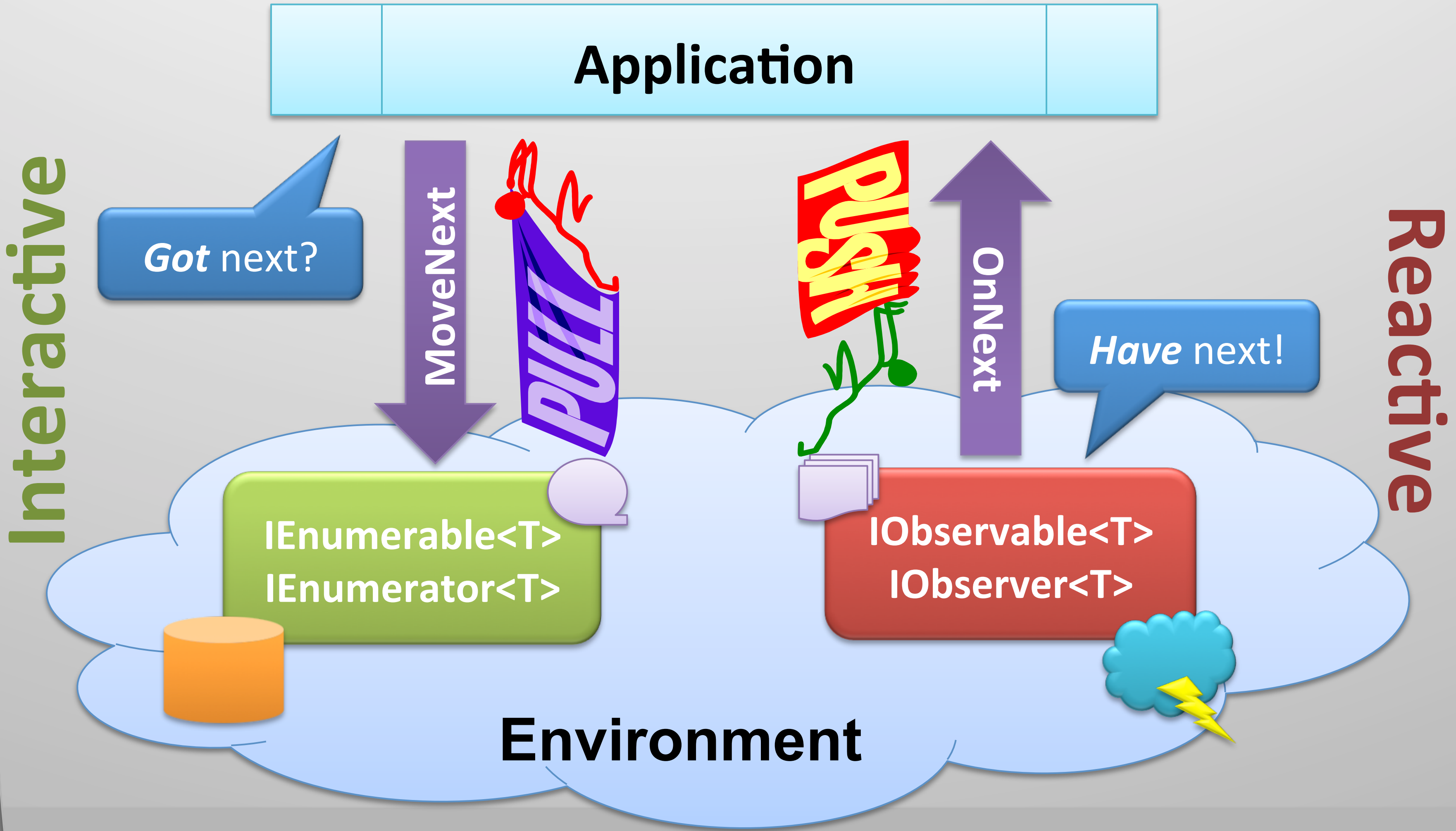


- .NET 3.5 SP1 and 4.0
- Silverlight 3 and 4
- XNA 3.1 for XBOX and Zune
- Windows Phone 7
- JavaScript (RxJS)

Download at **MSDN Data Developer Center**

Asynchronous Data Access

Push-based data retrieval



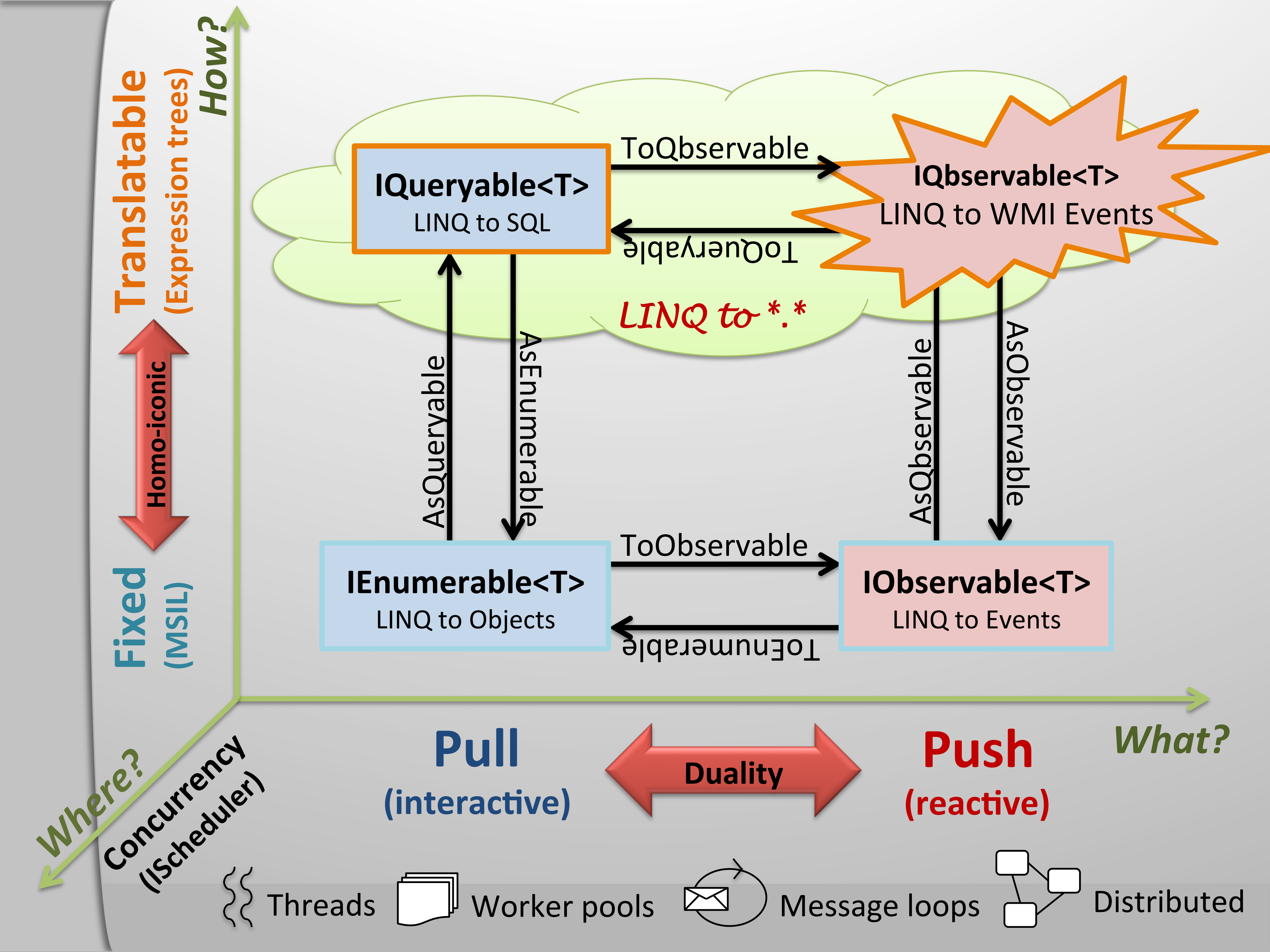
Asynchronous Data Access

Essential interfaces

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

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

Both interfaces ship
in the ***.NET 4 BCL***



How?

Translatable (Expression trees)

Homo-iconic

Fixed (MSIL)

IQueryable<T>
LINQ to SQL

ToQbservable

ToQueryable

IQbservable<T>
LINQ to WMI Events

LINQ to *.*

AsQueryable

AsEnumerable

AsQbservable

AsObservable

IEnumerable<T>
LINQ to Objects

ToObservable

ToEnumerable

IObservable<T>
LINQ to Events

What?

Pull
(interactive)

Duality

Push
(reactive)

Where?

Concurrency
(IScheduler)

Threads

Worker pools

Message loops

Distributed

Asynchronous Data Access

IQbservable<T>

```
interface IQbservable<out T> : IObservable<T>
{
    Expression            Expression  { get; }
    Type                  ElementType { get; }
    IQbservableProvider Provider    { get; }
}
```

We welcome ***semantic*** discussions

Extended role for some operators

```
interface IQbservableProvider
{
    IQbservable<R> CreateQuery<R>(Expression expression);
}
```

No **Execute** method

Demo

Asynchronous Data Access

Future C# and VB “await”

- The essence of the feature

Asynchronously computed value

```
async Task<int> GetLength(Task<string> name)
{
    string n = await name;
    return n.Length;
}
```

Result is async too

Can ***await*** an asynchronous value

- Feature based on the “**await-pattern**”
  - Awaiting Task<T> returns object of type T
  - Awaiting IObservable<T>?
    - Pattern implemented by Rx (cf. GetAwaiter method)
    - Many results, returns IList<T>

Asynchronous Data Access

IAsyncEnumerable<T>

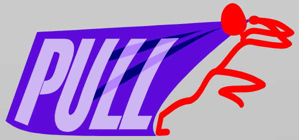
- Await feature is about **sequential code**
  - What about asynchronous **pull-based data** collections?

```
public interface IAsyncEnumerable<T>
{
  IAsyncEnumerator<T> GetEnumerator();
}
```

Rx defines ***Query Operators***

```
public interface IAsyncEnumerator<T> : IDisposable
{
  Current { get; }
  Task<bool> MoveNext();
}
```

Can ***await***



Demo

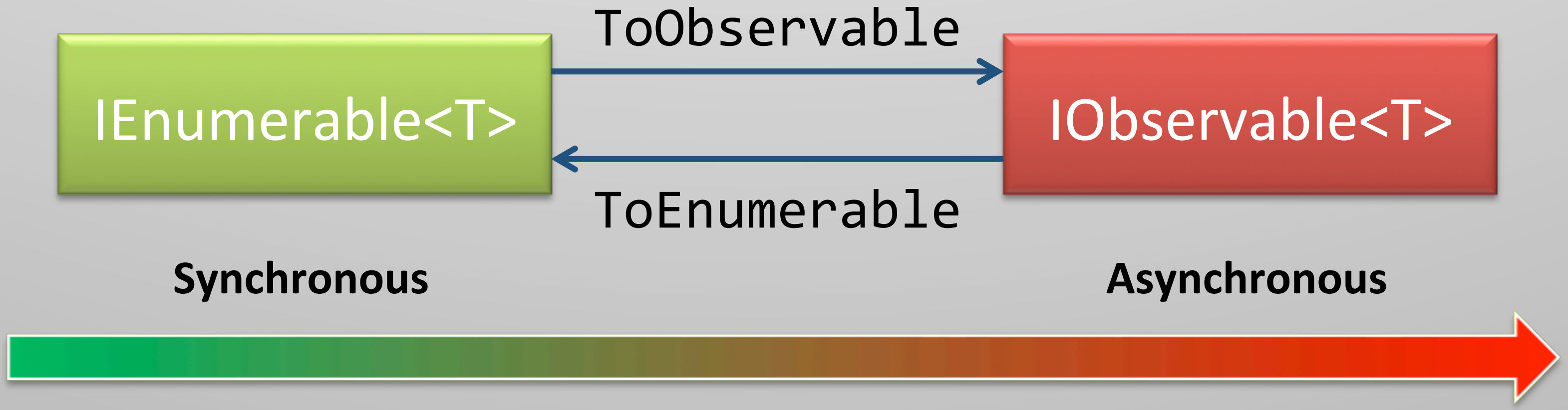
Where Execution Happens

IScheduler interface

Introduction** of **concurrency

System.Concurrency namespace in System.CoreEx

```
interface IScheduler
{
    DateTimeOffset Now { get; }
    IDisposable Schedule(Action action);
    IDisposable Schedule(Action action, TimeSpan dueTime);
}
```



Where Execution Happens

IScheduler specifies “where”

Imported TextBox **TextChanged** event

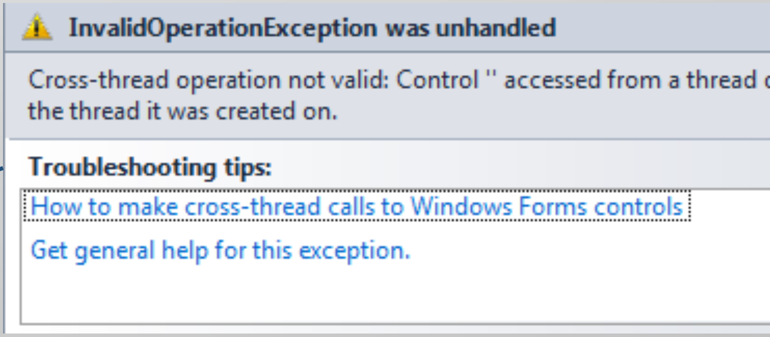
```
var res = from word in input.DistinctUntilChanged()
                            .Throttle(TimeSpan.FromSeconds(0.5))
          from words in lookup(word)
          select words;
```

Asynchronous ***web service call***

Indicates ***where*** things run

Use of scheduler to ***synchronize***

```
res.ObserveOn(new ControlScheduler
res.Subscribe(words =>
{
    lst.Items.Clear();
    lst.Items.AddRange((from word in words
                        select word.Word).ToArray());
});
```



⚠ **InvalidOperationException was unhandled**

Cross-thread operation not valid: Control '' accessed from a thread o
the thread it was created on.

Troubleshooting tips:

How to make cross-thread calls to Windows Forms controls

Get general help for this exception.

Where Execution Happens

IScheduler parameterization

~~***Bak*** [illegible] otion of “[illegible] e”?~~

Entry-point for the ***schema***

```
var ctx = new NorthwindDataContext();
var res = from product in ctx.Products
          where product.Price > 100m
          select product.Name;
```

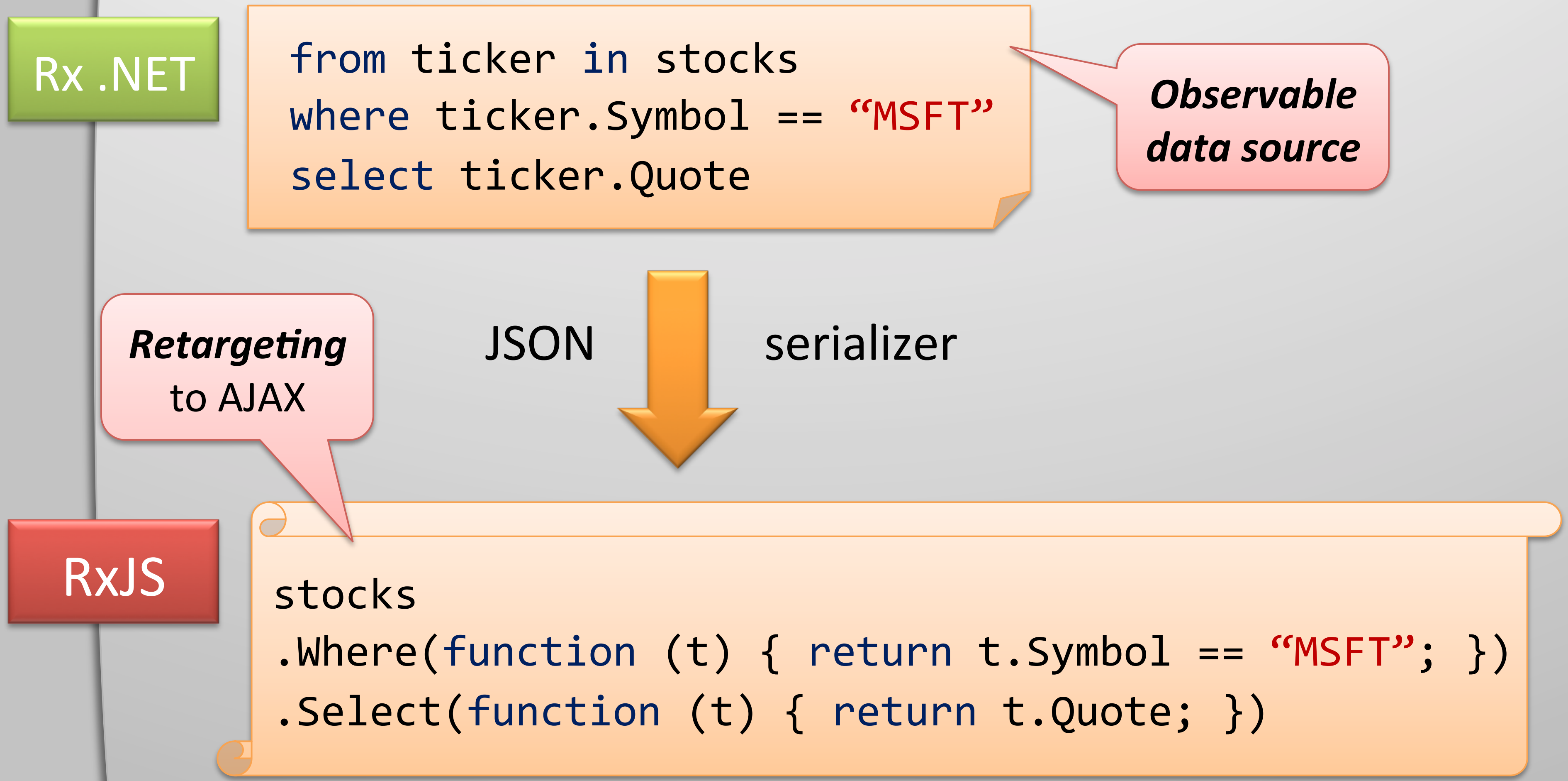
Custom schedulers could be ***very rich*** (e.g. server farm)

Decoupled “what” from “where”

```
foreach (var p in res.RemoteOn(new SqlScheduler(“server”)))
    // Process product
```

Where Execution Happens

Expression tree remoting



Rx .NET

```
from ticker in stocks
where ticker.Symbol == “MSFT”
select ticker.Quote
```

Observable data source

JSON serializer

Retargeting to AJAX

RxJS

```
stocks
.Where(function (t) { return t.Symbol == “MSFT”; })
.Select(function (t) { return t.Quote; })
```

Demo

- Remoting of Query Operators

LINQ to the Unexpected

Microsoft® Solver Foundation

```
Model[
  Decisions[Reals, SA, VZ],
  Goals[
    Minimize[20 * SA + 15 * VZ]
  ],
  Constraints[
    C1 ->    0.3 * SA
           + 0.4 * VZ >= 2000,
    C2 ->    0.4 * SA
           + 0.2 * VZ >= 1500,
    C3 ->    0.2 * SA
           + 0.3 * VZ >= 500,
    C4 -> SA <= 9000,
    C5 -> VZ <= 6000,
    C6 -> SA >= 0,
    C7 -> VZ >= 0
  ]
]
```

Cost / barrel

Refinement %

Max
# barrels

Min
# barrels

Non-persistent

```
from m in ctx.CreateModel(new {
    vz = default(double),
    sa = default(double)
})
where    0.3 * m.sa
       + 0.4 * m.vz >= 2000
    &&   0.4 * m.sa
       + 0.2 * m.vz >= 1500
    &&   0.2 * m.sa
       + 0.3 * m.vz >= 500
where    0 <= m.sa && m.sa <= 9000
    &&   0 <= m.vz && m.vz <= 6000
orderby 20 * m.sa + 15 * m.vz
select m
```

To compute
call **Solve**

LINQ to the Unexpected

Joining with reactive sources

SelectMany

```
from costSA in GetPriceMonitor("SA")
from costVZ in GetPriceMonitor("VZ")

from m in ctx.CreateModel(
                  new { vz = default(double),
                        sa = default(double) })
where 0.3 * m.sa + 0.4 * m.vz >= 2000
   && 0.4 * m.sa + 0.2 * m.vz >= 1500
   && 0.2 * m.sa + 0.3 * m.vz >= 500
where 0 <= m.sa && m.sa <= 9000
   && 0 <= m.vz && m.vz <= 6000

orderby costSA * m.sa + costVZ * m.vz
select m
```

Observable data sources

Subscribe here!

Parameters to decision

Demo

- Theorem Solving using Z3

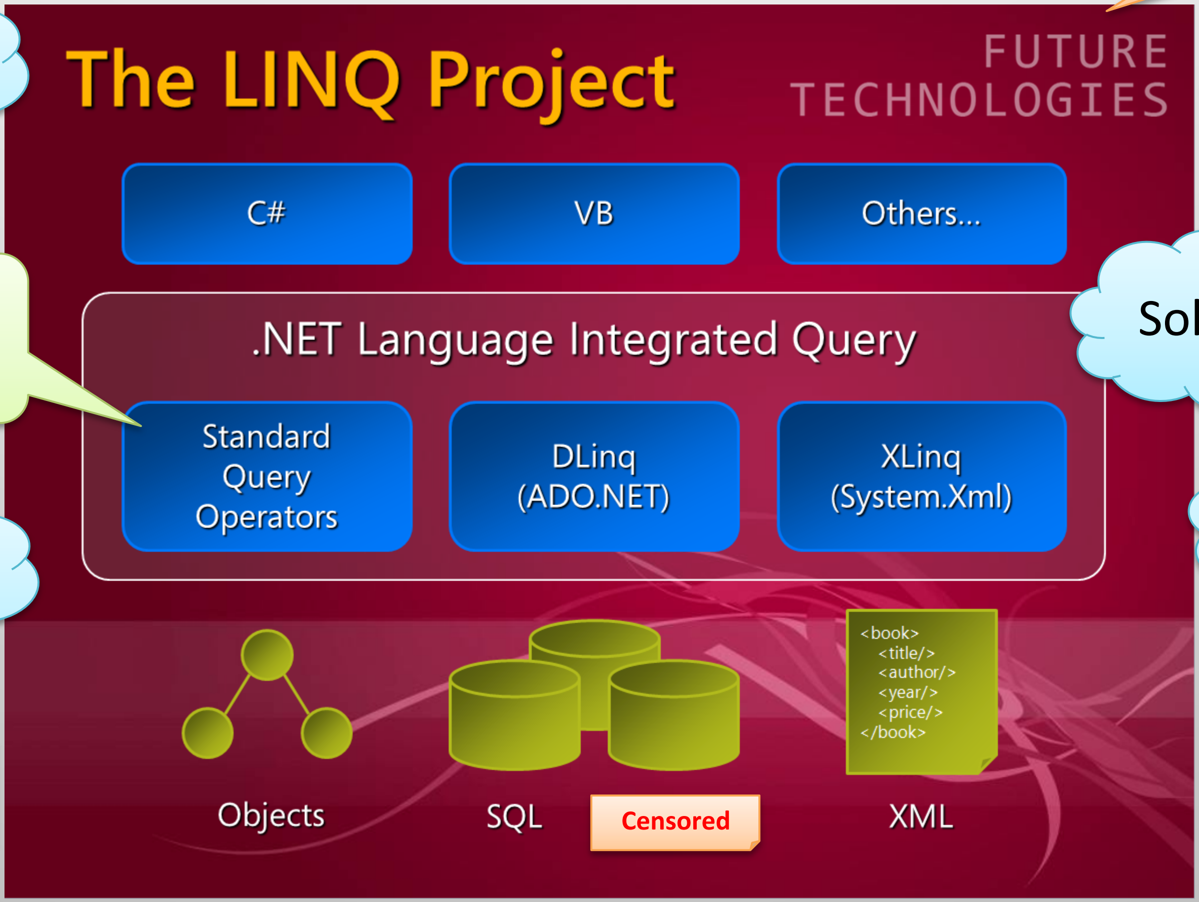
What’s LINQ?

A futuristic perspective

Next 5 years

Remoting

Async



The LINQ Project

FUTURE TECHNOLOGIES

C#

VB

Others…

.NET Language Integrated Query

Standard Query Operators

DLinq (ADO.NET)

XLinq (System.Xml)

```
<book>
  <title/>
  <author/>
  <year/>
  <price/>
</book>
```

Objects

SQL

Censored

XML

Rx and Ix are here

Solvers

Toolkits

Scheduler

Thanks!

Bart J.F. De Smet