Language Integrated Query (LINQ)

Nguyen Minh Dao
daonm@yahoo.com

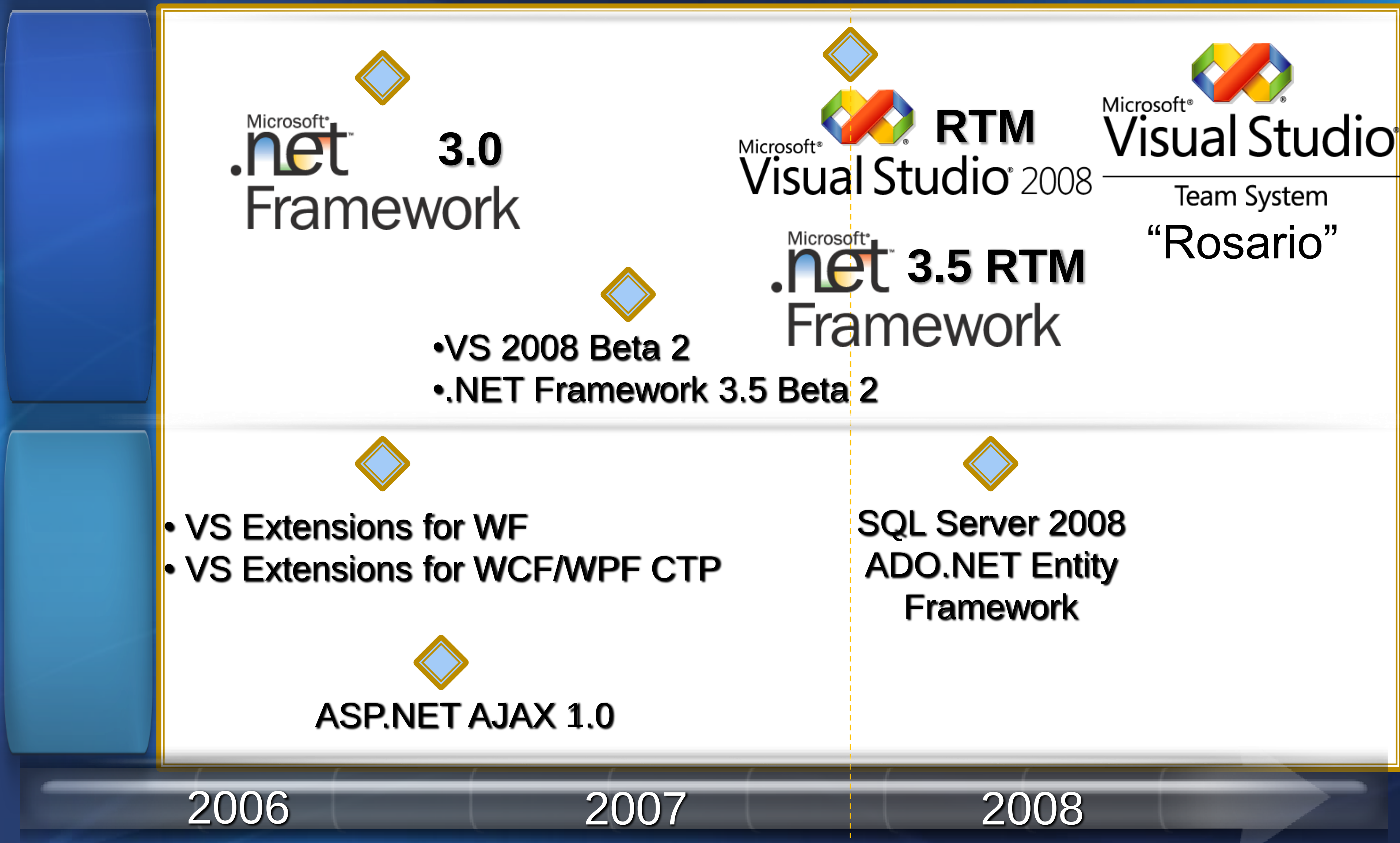
Introduction

- We use many different types of query
  - SQL, XQuery/XPath, DataView row filters, etc.
- Maybe we could enhance productivity by...
  - Deciding on one query expression syntax
  - Enabling compilers to check queries & results
  - Allow extensibility to target all kinds of data

Agenda

- C#3 and VB9 Language Enhancements
  - Building to LINQ to Objects
- LINQ to XML
- LINQ to SQL
- LINQ to DataSets

.NET Framework - VS Roadmap



Microsoft .net Framework 3.0

Microsoft Visual Studio 2008 RTM

Microsoft .net Framework 3.5 RTM

Microsoft Visual Studio Team System "Rosario"

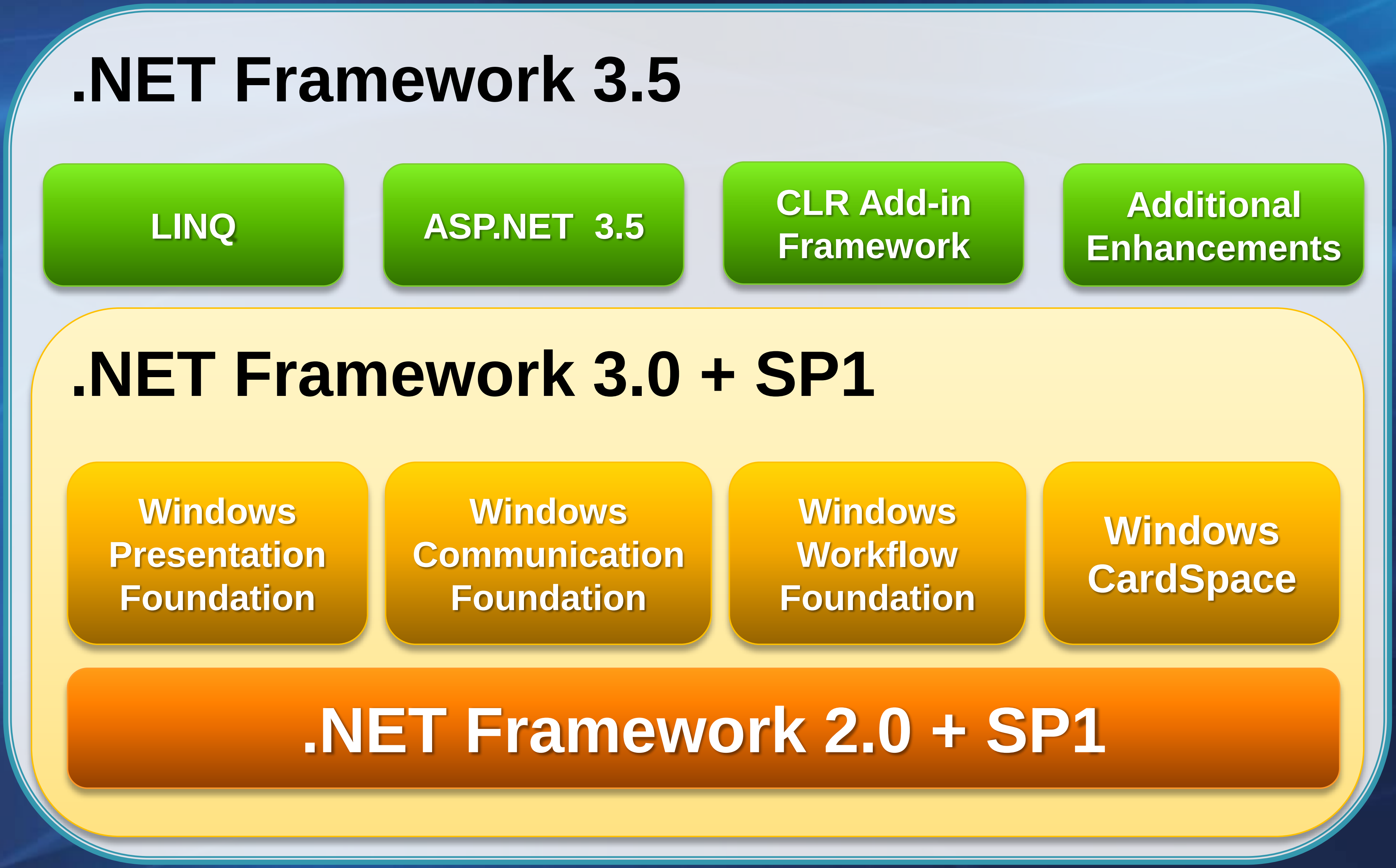
- VS 2008 Beta 2
- .NET Framework 3.5 Beta 2

- VS Extensions for WF
- VS Extensions for WCF/WPF CTP

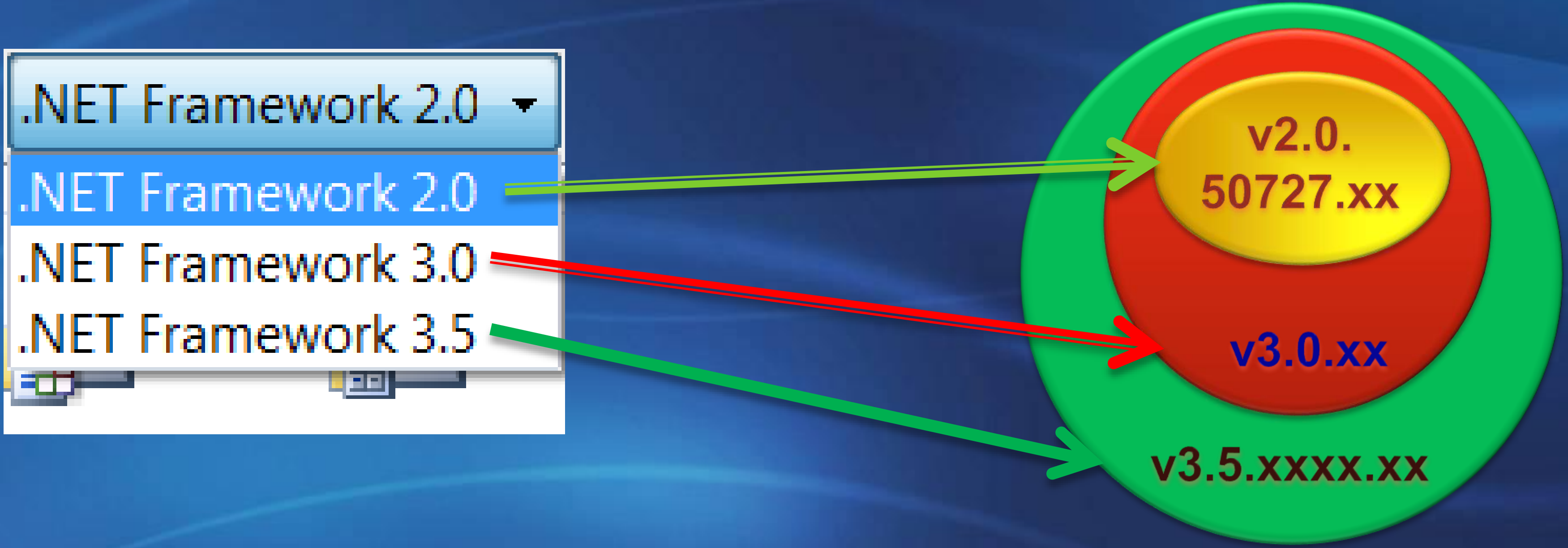
ASP.NET AJAX 1.0

SQL Server 2008
ADO.NET Entity Framework

What is the .NET Framework 3.5?

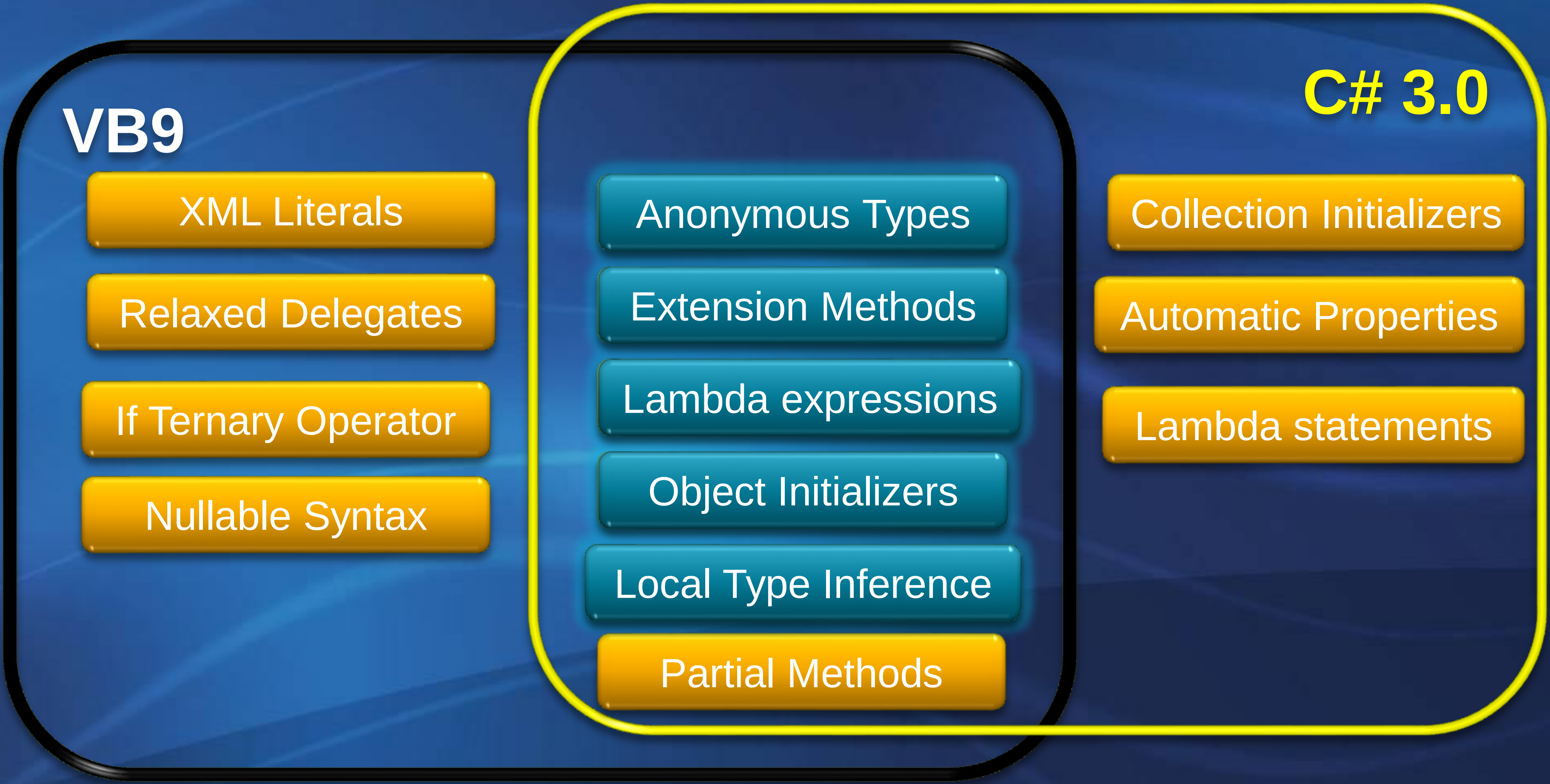


Multi-targeting in Visual Studio 2008



Compiler Features

Most are LINQ enablers



Language Innovations

Query expressions

```
var contacts =
    from c in customers
    where c.City == "Hove"
    select new { c.Name, c.Phone };
```

Local variable type inference

Lambda expressions

```
var contacts =
    customers
    .Where(c => c.City == "Hove")
    .Select(c => new { c.Name, c.Phone });
```

Extension methods

Anonymous types

Object initializers

Extension methods

- Add methods to existing types “virtually”
  - Static method with first “**this**” parameter
  - Scoping using namespaces

```
static class MyExtensions {
   public static string Reverse(this string s) {
      char[] c = s.ToCharArray();
      Array.Reverse(c);
      return new string(c);
   }
}
```

Promoted first parameter

```
string name = “Bart”;
string reversed = name.Reverse();
```

Automatic properties

- Tired of writing properties?
  - Compiler can generate the get/set plumbing

```
class Customer {
   private string _name;

   public string Name
   {
      get { return _name; };
      set { _name = value; };
   }
}
```

Setter required; can be private or internal

```
class Customer {
   public string Name { get; set; }
}
```

Object initializers

- Ever found the constructor of your taste?
  - Insufficient overloads, so pick one
  - Call various property setters

```
class Customer {
   public string Name { get; set; }
   public int Age { get; set; }
}
```

```
Customer c = new Customer();
c.Name = “Bart”;
c.Age = 24;
```

Can be combined with any constructor call

```
var c = new Customer(){ Name = “Bart”, Age = 24};
```

Collection initializers

- Arrays easier to initialize than collections?!

```
int[] ints = new int[] { 1, 2, 3 };
List<int> lst = new List<int>();
lst.Add(1);
lst.Add(2);
lst.Add(3);
```

```
int[] ints = new int[] { 1, 2, 3 };
var lst = new List<int>() { 1, 2, 3 };
```

Works for any ICollection class by calling its Add method

Anonymous types

- Let the compiler cook up a type
  - Can’t be returned from a method
  - Local variable type inference becomes a must
  - Used in LINQ query projections

```
var person = new { Name = “Bart”, Age = 24 };
```

```
var customer = new { Id = id, person.Name };
```

Lambda expressions

- Functional-style anonymous methods

```
delegate R BinOp<A,B,R>(A a, B b);

int Calc(BinOp<int, int, int> f, int a, int b)
{
   return f(a, b)
}
```

```
int result = Calc(
   delegate (int a, int b) { return a + b; },
   1, 2);
```

Parameter types inferred based on the target delegate

```
var result = Calc((a, b) => a + b, 1, 2);
```

Demo

Anonymous types, automatic properties, collection initializers, extension methods

First, A Taste of LINQ

```
using System;
using System.Query;
using System.Collections.Generic;

class app {
  static void Main() {

    string[] names = { "Burke", "Connor",
                       "Frank", "Everett",
                       "Albert", "George",
                       "Harris", "David" };

    var expr = from s in names
               where s.Length == 5
               orderby s
               select s.ToUpper();

    foreach (string item in expr)
      Console.WriteLine(item);
  }
}
```

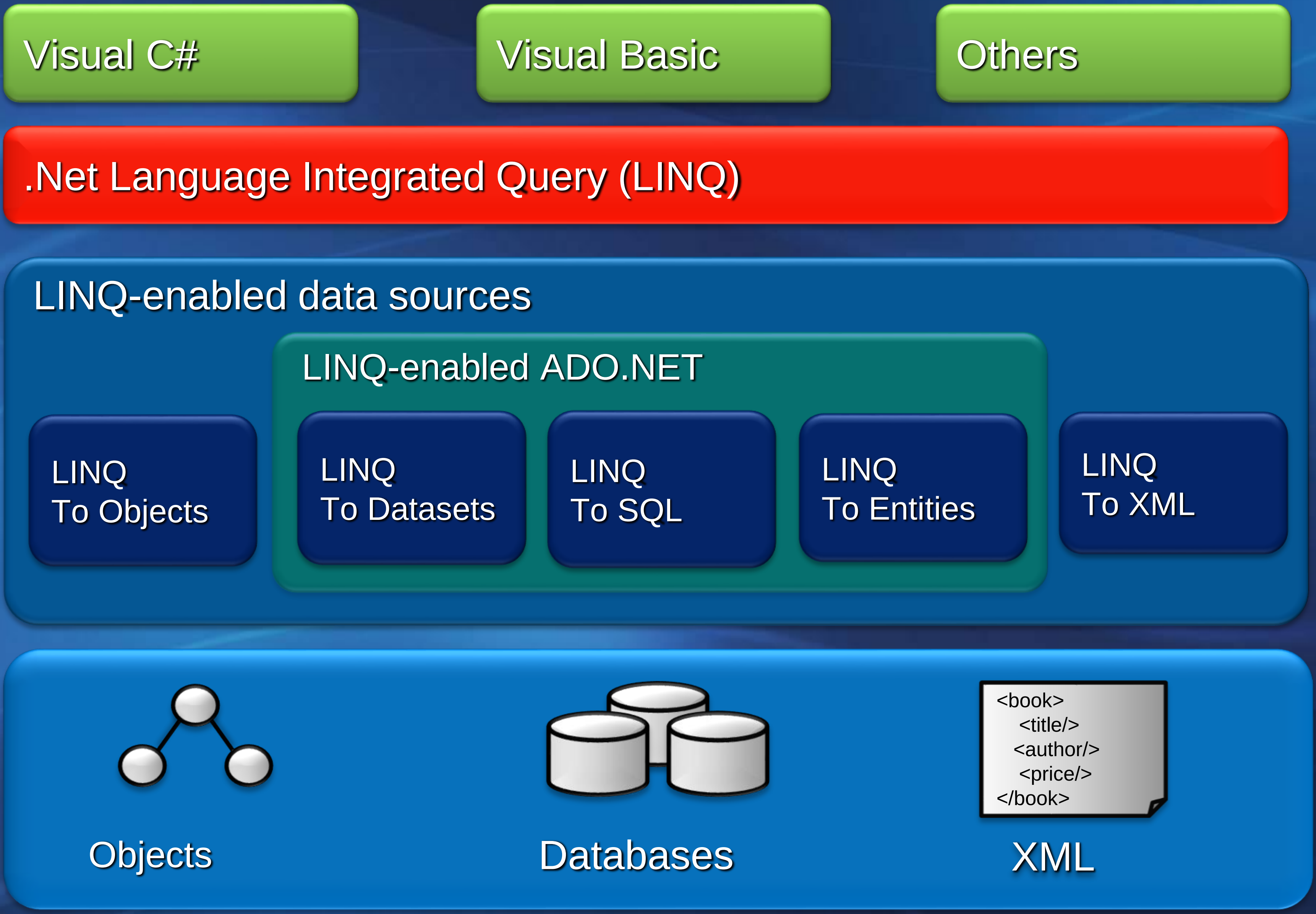
```
BURKE
DAVID
FRANK
```

Query Expressions

- Introduce SQL-Like Syntax to Language
- Compiled to Traditional C# (via Extension Methods)

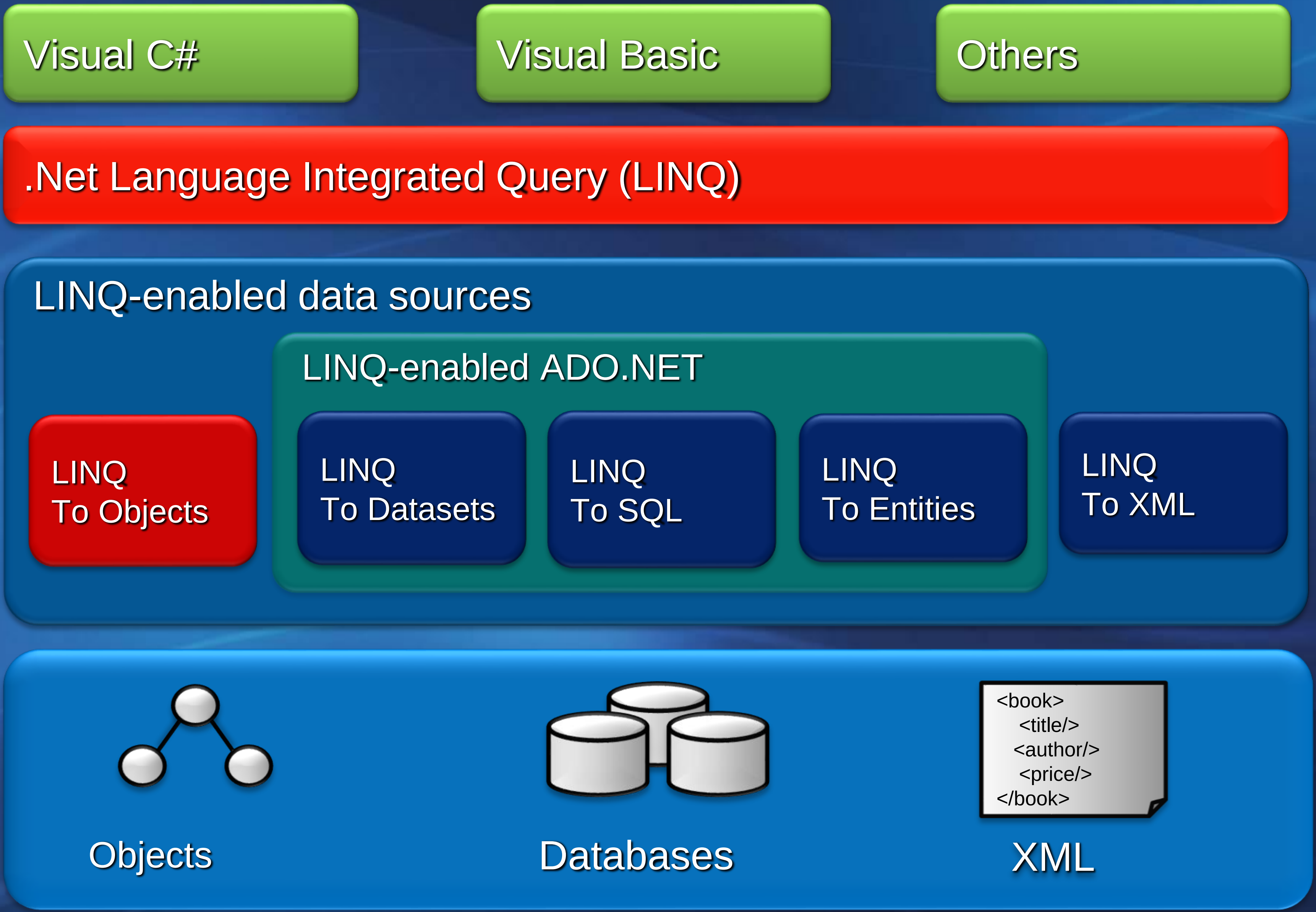
```
from itemName in srcExpr
join itemName in srcExpr on keyExpr equals keyExpr
       (into itemName)?
let itemName = selExpr
where predExpr
orderby (keyExpr (ascending | descending)?)*
select selExpr
group selExpr by keyExpr
into itemName query-body
```

LINQ Architecture



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

LINQ Architecture



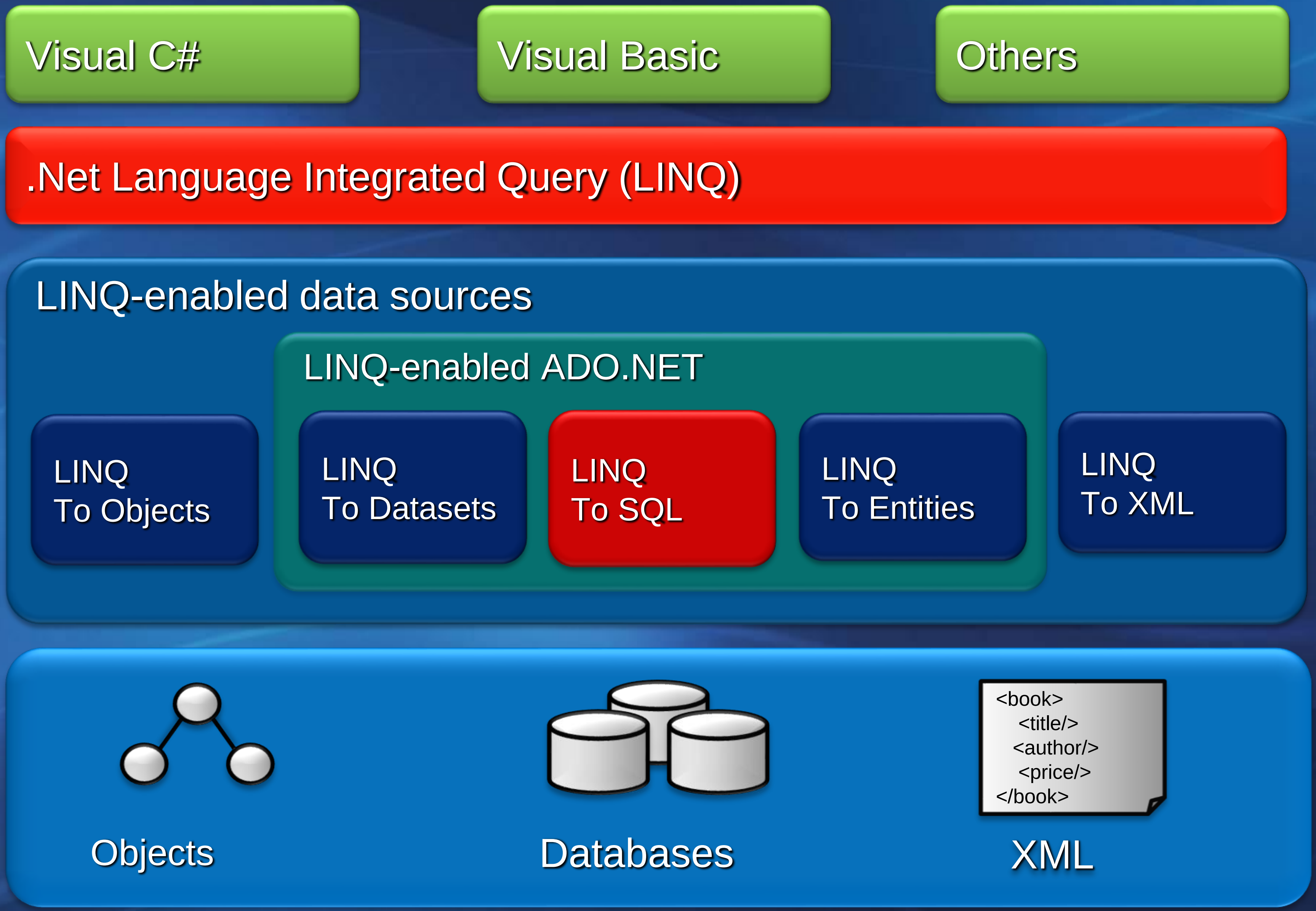
LINQ to Objects

- Native query syntax in C# and VB
  - IntelliSense
  - Autocompletion
- Query Operators can be used against any .NET collection (IEnumerable<T>)
  - Select, Where, GroupBy, Join, etc.
- Deferred Query Evaluation
- Lambda Expressions

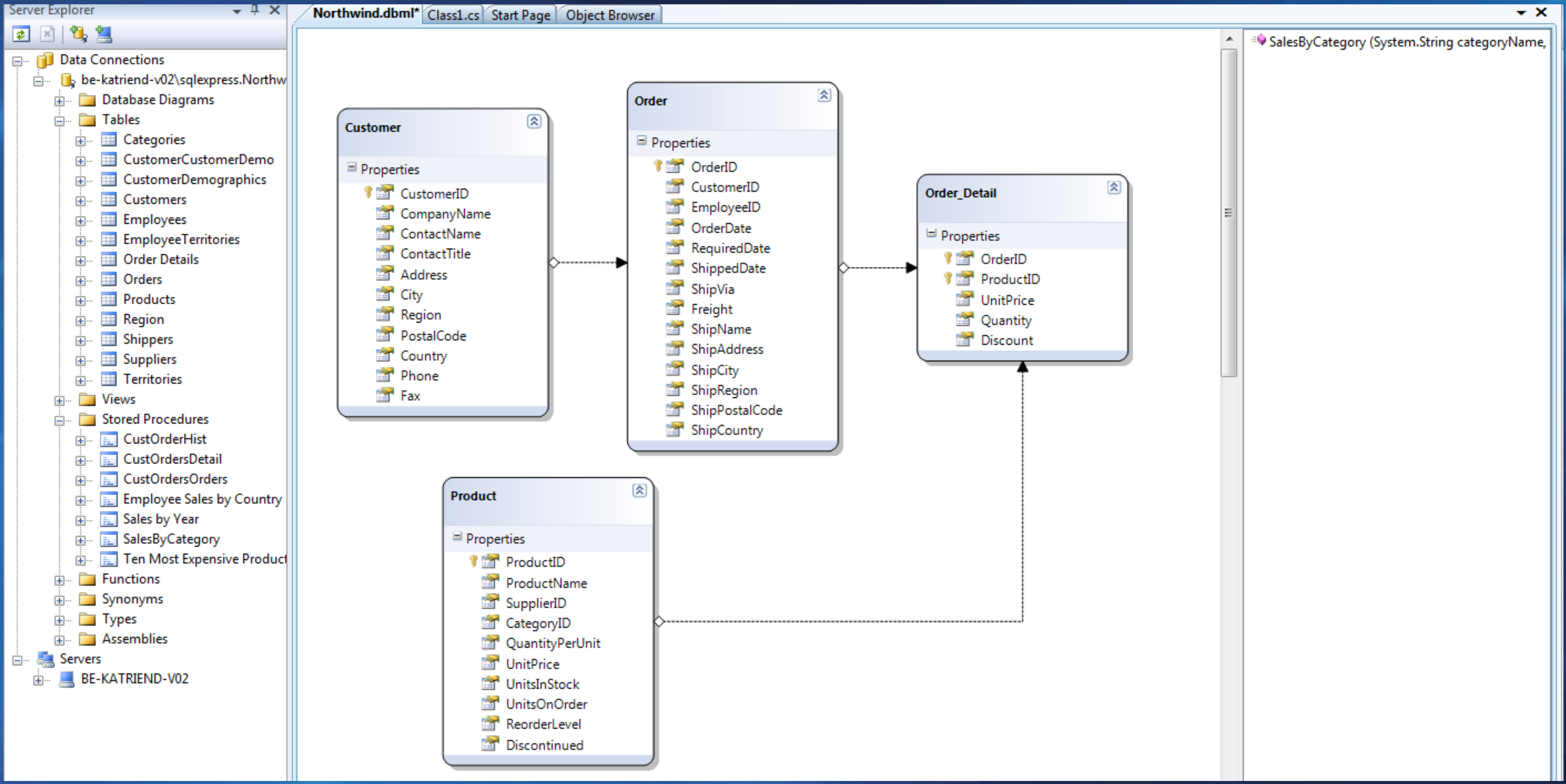
```
using System;
using System.Query;
using System.Collections.Generic;

class app {
  static void Main() {
    string[] names = { "Burke", "Connor",
                 "Frank", "Everett",
                 "Albert", "George",
                 "Harris", "David" };

    IEnumerable<string> expr =
                from s in names
                where s.Length == 5
                orderby s
                select s.ToUpper();

    foreach (string item in expr)
      Console.WriteLine(item);
  }
}
```

```
BURKE
DAVID
FRANK
```

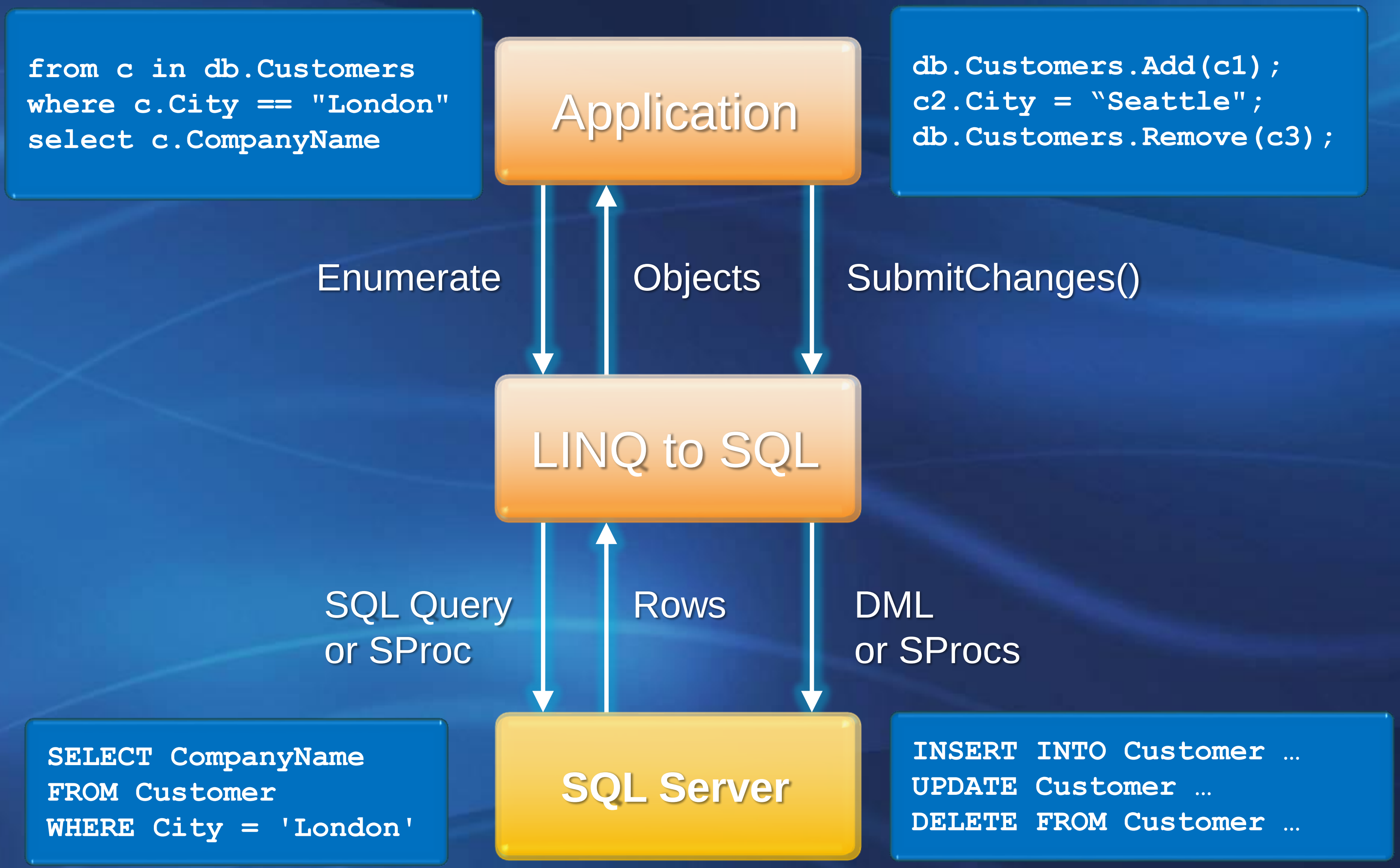
LINQ Architecture



LINQ to SQL Overview



LINQ to SQL Architecture



```
from c in db.Customers
where c.City == "London"
select c.CompanyName
```

```
db.Customers.Add(c1);
c2.City = "Seattle";
db.Customers.Remove(c3);
```

```
SELECT CompanyName
FROM Customer
WHERE City = 'London'
```

```
INSERT INTO Customer …
UPDATE Customer …
DELETE FROM Customer …
```

LINQ to SQL

- DataContext is the central class
- Use code-gen for ORM
- SQL is only submitted when needed
- Parent-child relationships are respected
  - Control of deferred loading
- Can insert/update/delete
  - Transactionally, with concurrency checks

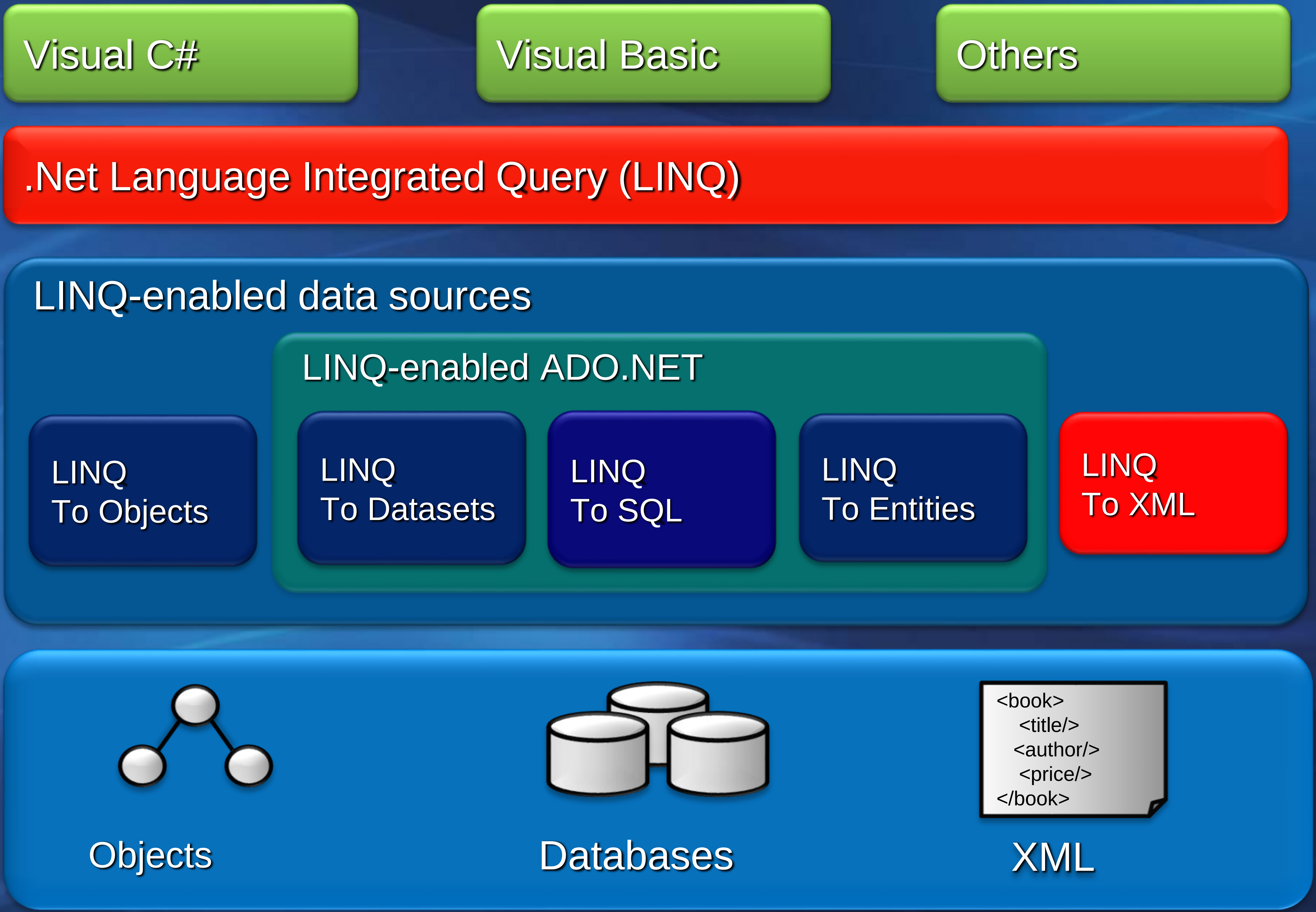
LINQ to DataSet

- Query expressions over in-memory data
- Works with untyped or typed DataSets
- If query returns some kind of DataRow: -
  - Can yield results as a DataView
  - ...and therefore databind to those results

Demo

LINQ to Objects
LINQ to SQL

LINQ Architecture



LINQ to XML

- Creating XML
  - Constructors lend themselves to nesting
  - Can use LINQ (over anything) to build XML
- Querying
  - Use normal *axes* from XML infoset
  - Get full power of query expressions over XML
  - Select, where, group by, etc.
- Xml Namespaces

Demo

LINQ to XML

That’s LINQ

- A combination of new language features, and new fx3.5 classes (with extension methods)
- A common query expression syntax
- Freedom to implement across different kinds of data
- It’s TYPED...
  - The compiler can check your queries
  - The compiler can check your results

MSDN in the UK

Sign up for the MSDN Newsletter
msdn
Register for an MSDN Event
msdn
Download the latest Nuggets
msdn
Meet the UK Developer Community Team
msdn

- Visit http://msdn.co.uk
  - Newsletter
  - Events
  - Nugget Videos
  - Blogs

THANK YOU