Fachhochschule Nordwestschweiz
Hochschule für Technik

ADVANCED LINQ

Martin Kropp, Yves Senn
University of Applied Sciences Northwestern Switzerland

Learning Target

You

- can solve complex query problems using LINQ
- can explain the difference between fluent notation and query notation
- can formulate LINQ statements in both fluent and query notation and can transform one into the other
- can explain LINQ expression trees

Content

- More LINQ
  - Deferred Execution revisited
  - Subqueries
- LINQ Query Notation
- LINQ Providers
- LINQ Expression Tree

Delegate and Lambda revisited

Just another way to write delegates:

```
static bool SomePredicate(Point p)
{
    return p.X * p.Y > 100000;
}
Predicate<Point> d=SomePredicate;
```

as a full method

```
Predicate<Point> d = delegate(Point p)
{
    return p.X * p.Y > 100000;
};
```

as an anonymous method

```
Predicate<Point> d = p => p.X * p.Y > 100000;
```

as a lambda

Lambda expressions revisited

```
Func<Point,bool> d = p => p.X * p.Y > 100000;
```

Type inferred automatically

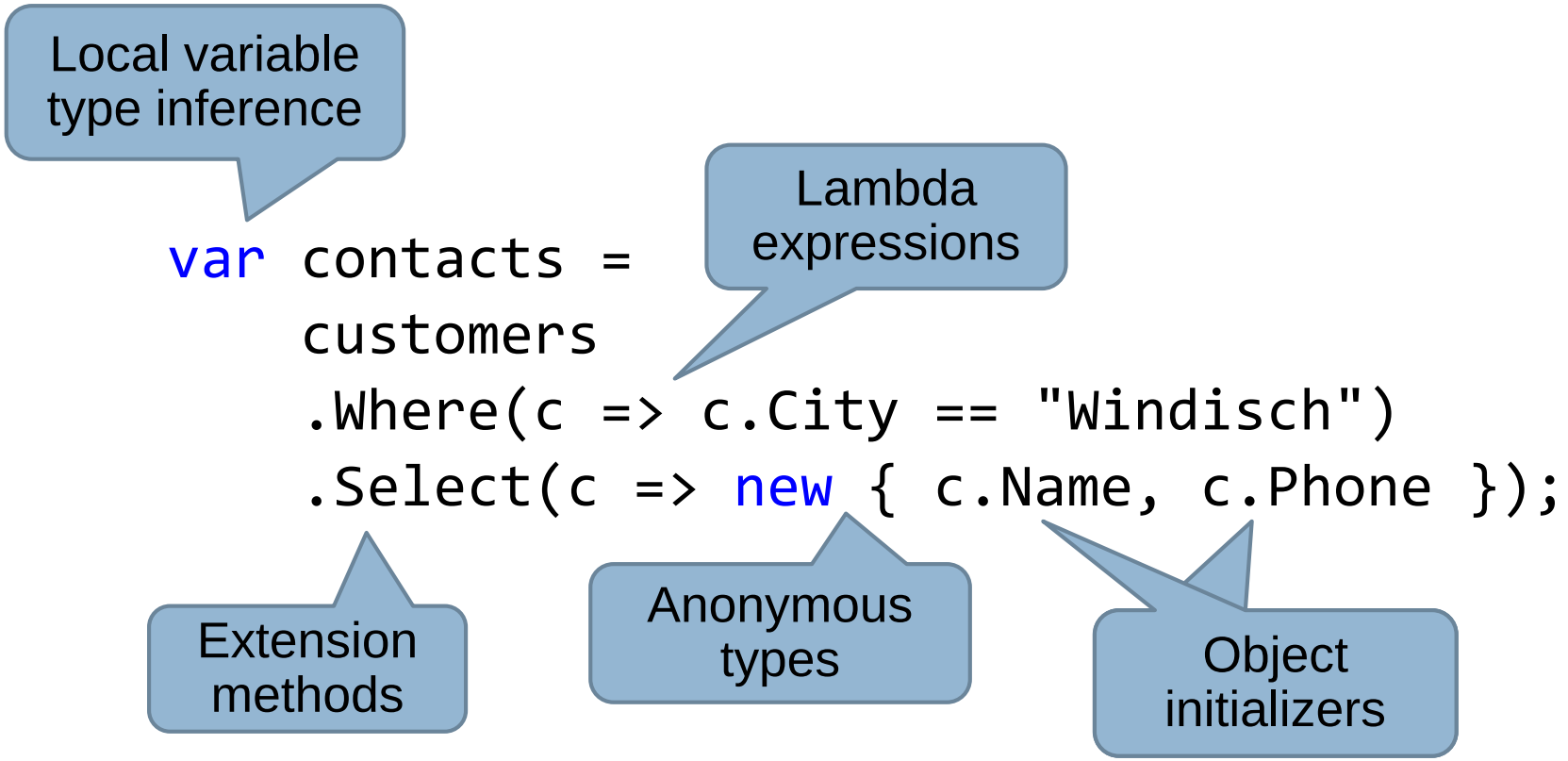
```
Func<Point,bool> d = (Point p) => p.X * p.Y > 100000;
```

explicit typing

```
Func<Point,bool> d = p =>
{
    var r = p.X * p.Y;
    return r > 100000;
};
```

Complex functions

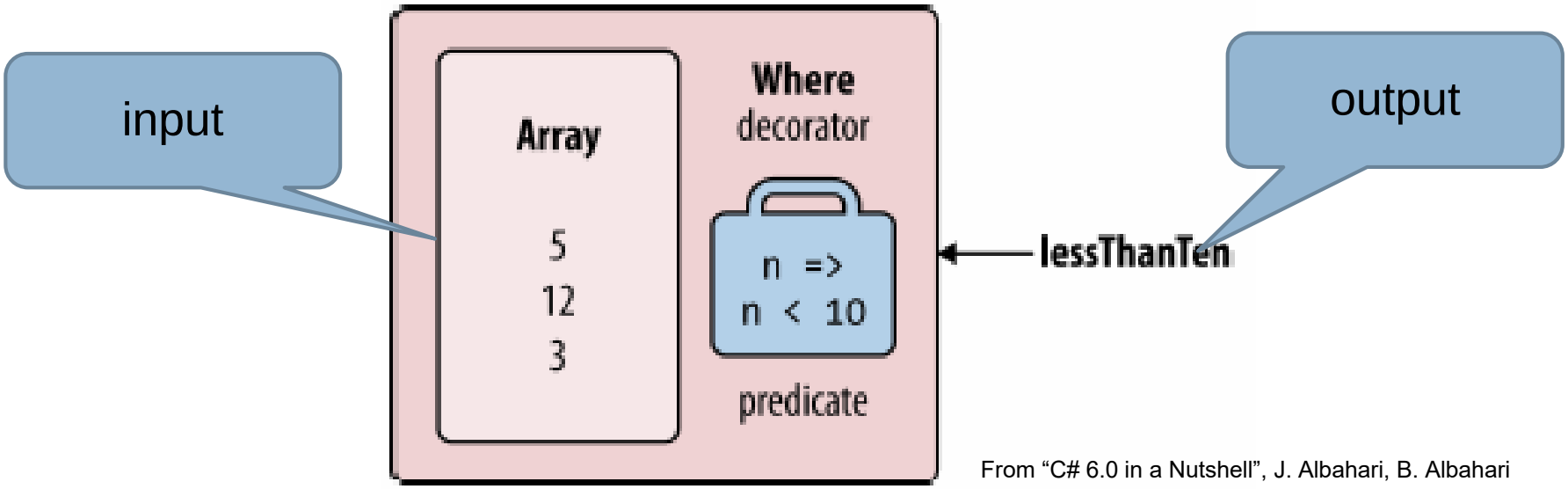
LINQ language features summary



How deferred execution works

Decorator sequence:

```
var lessThanTen = new int[]{5, 12, 3}.Where(n => n < 10);
```



From “C# 6.0 in a Nutshell”, J. Albahari, B. Albahari

https://stackoverflow.com/questions/20962571/what-is-the-use-of-decorator-sequences-in-deferred-execution (second answer from Jon Skeet)

Worksheet – Part 1

Subqueries

A subquery is a query contained within another query's lambda expression:

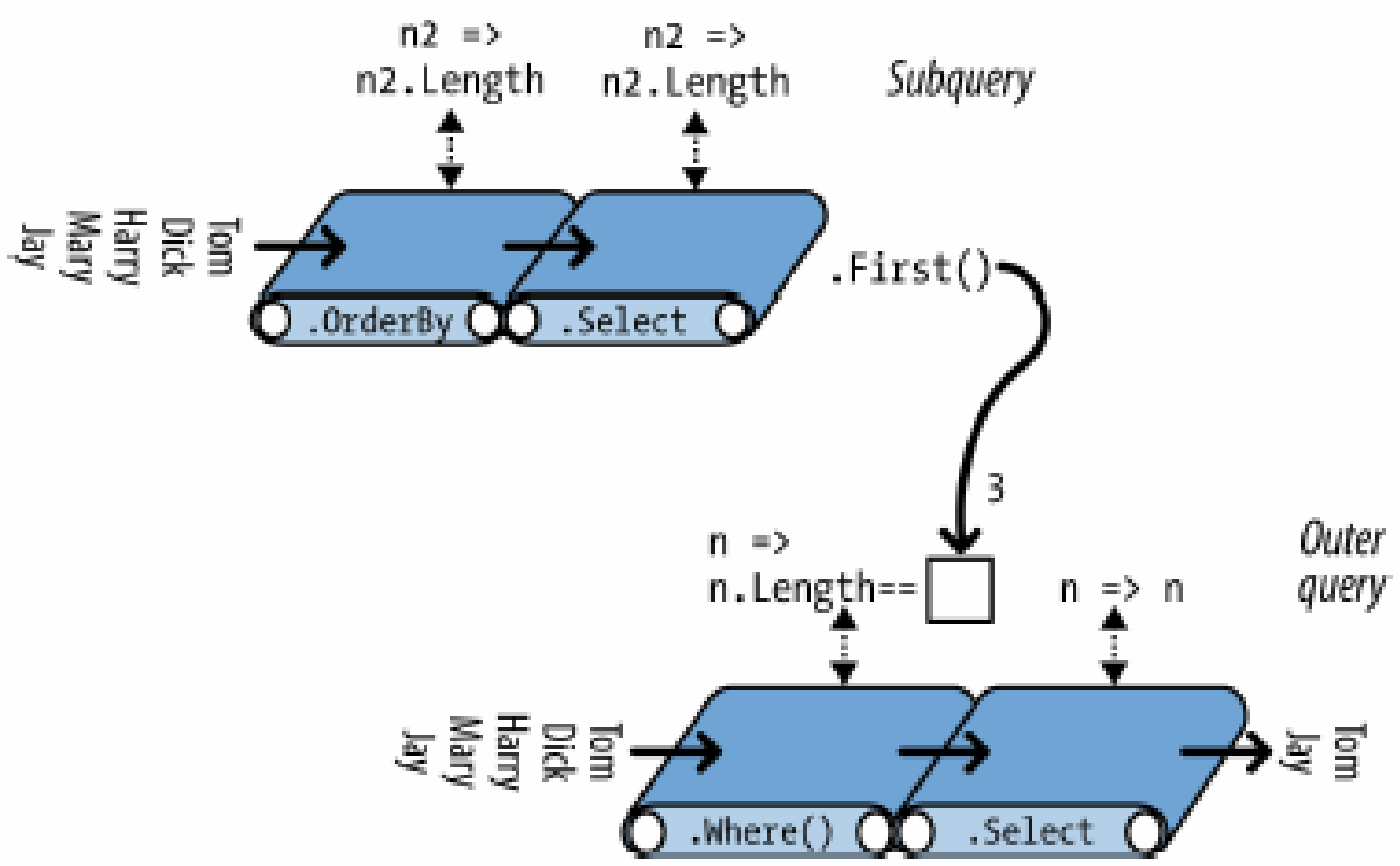
```
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };

var outerQuery = names.Where(n => n.Length ==
                      names.OrderBy(n => n.Length)
                           .Select(n => n.Length)
                           .First());
```

Quiz:

- What's wrong with this query? (Hint: closures)
- Fix it
- What does the fixed query evaluate to?

Subquery composition



From “C# 6.0 in a Nutshell”, J. Albahari, B. Albahari

GroupBy

A GroupBy query groups an enumerable into sub-enumerables by a defined key:

```
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };

var groupBy = names.GroupBy(n => n.Length);
foreach(var group in groupBy) {
        Console.WriteLine(group.Key);
        Console.WriteLine(group.Aggregate((r,n)=>r+","+n));
}
```

Quiz:

- How many times does it iterate?
- What does line 1 print?
- What does line 2 print?

Method syntax vs. query syntax

```
int[] numbers = { 5, 10, 8, 3, 6, 12 };

//"Method" or "fluent" syntax
var numQuery2 = numbers
    .Where(num => num % 2 == 0)
    .OrderBy(n => n);

//"Query comprehension" syntax
var numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;
```

Formulate queries using Lambda expressions, using C# syntax

Formulate queries using a SQL-like syntax

Query Comprehension Syntax

- Query expression consists of set of clauses written in a declarative syntax similar to SQL or XQuery
- Query must
  - begin with **`from`** clause, and
  - end with **`select`** or **`group`** clause

- Between first `from` clause and last `select`/`group` clause, it can contain one or more of the following clauses

```
Where
Orderby
Join
Let
From
Into
```

Hybrid Syntax

You can mix query and method syntax:

```
var results = (from c in Comedians
               select c).Count();
```

This may be necessary because the query syntax alone is not expressive enough.

Worksheet – Part 2

Worksheet – Part 3

“Classic” data query approaches

- Objects using loops and conditions

```
foreach(Customer c in customers)
   if (c.Region == "USA") ...
```

- SQL SELECT from database tables

```
SELECT * FROM Customers WHERE
Region='USA'
```

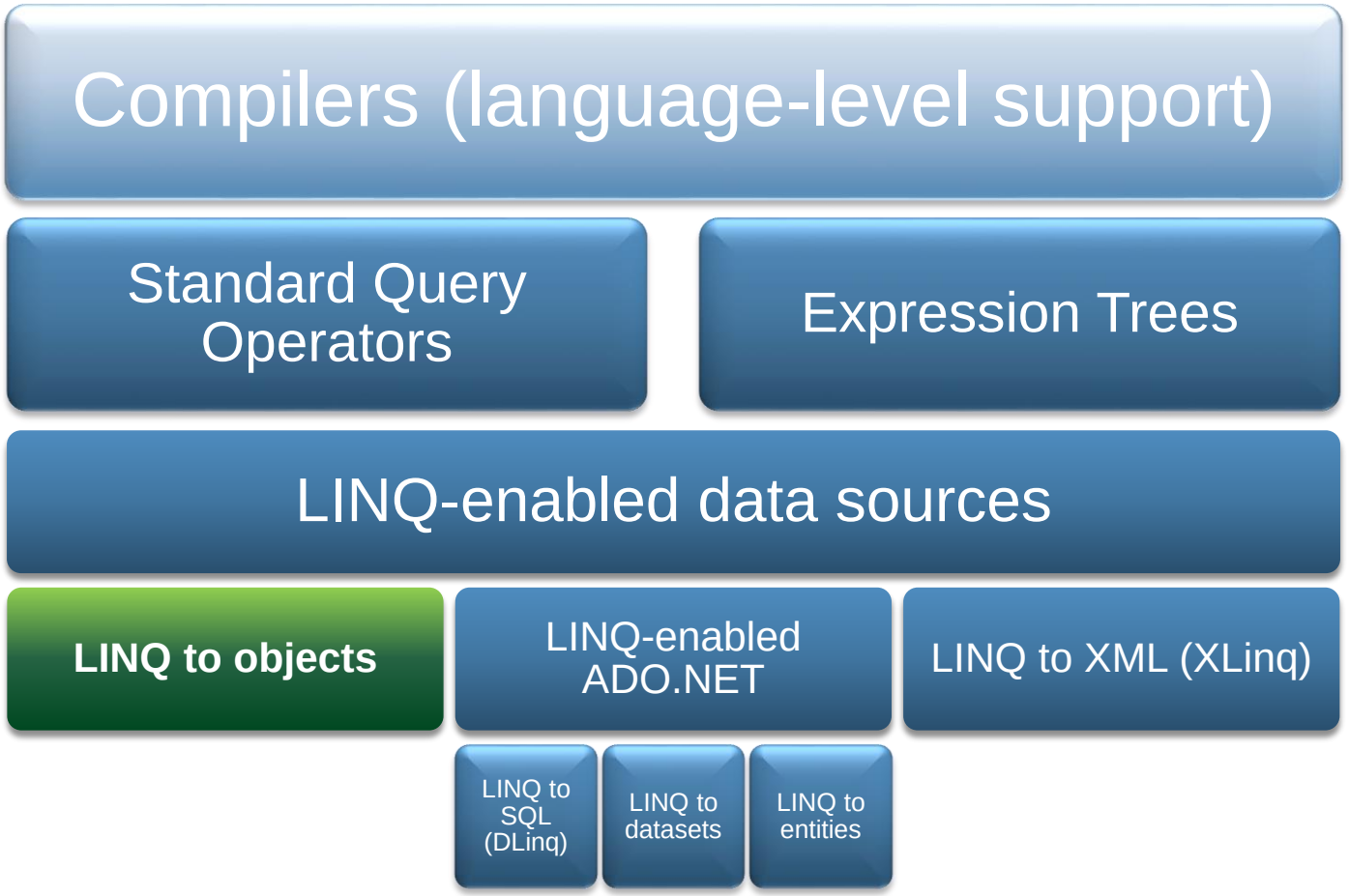
- XPath/XQuery for XML

```
//Customers/Customer[@Region='USA']
```

Problems of classic approaches

- Not type safe
- Error prone
- Different syntax
- Not maintenance-friendly
- Not portable

LINQ architecture



LINQ to Objects

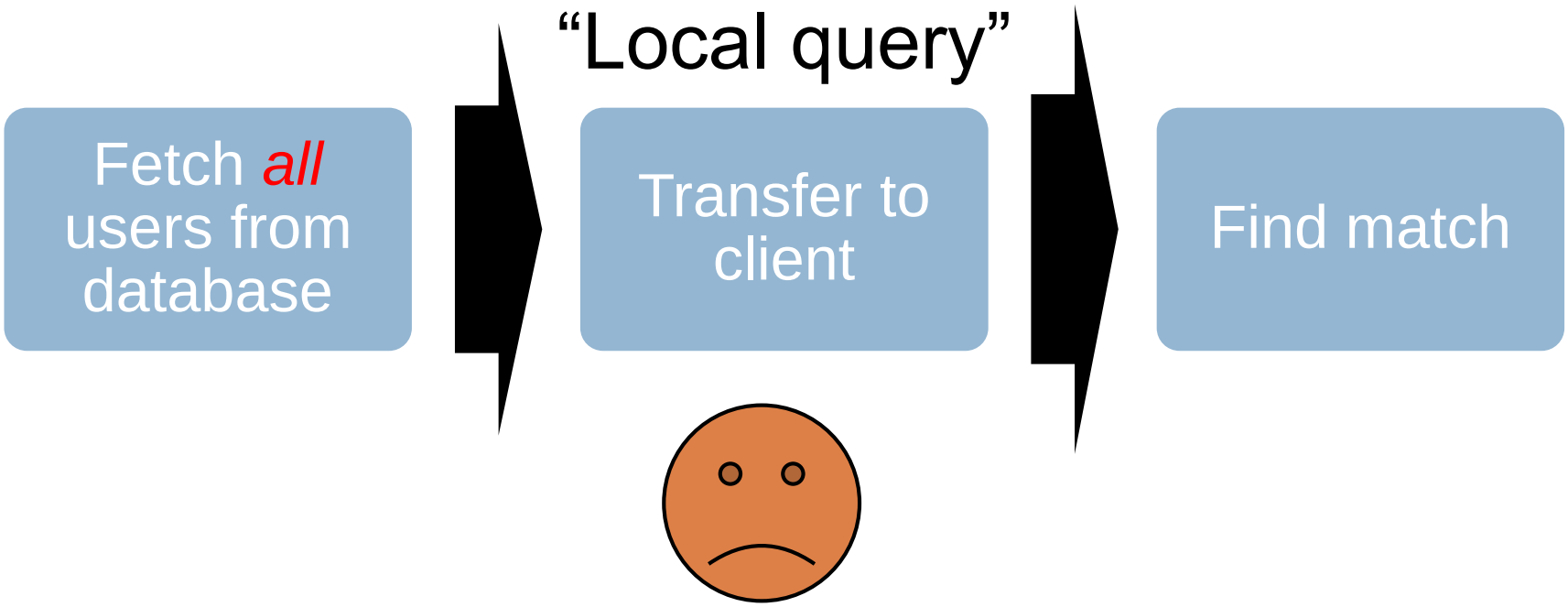
- If an object supports the `IEnumerable` interface, LINQ to Objects enables you to query it. Examples:
  - Arrays
  - Collections (`List`, `Set`, …)
  - Strings
  - …
- Resides in the `System.Linq` namespace
- All examples shown so far have used LINQ to Objects

LINQ providers

- **LINQ to Objects – in-memory data**
- LINQ to SQL – An O/R-Mapper
- LINQ to Entities – Entity Framework, another O/R-Mapper
- LINQ to XML – XML documents
- DryadLINQ – Distributed computing
- LINQ to Hive – Use Hadoop to run code
- LINQ to Twitter – Read Twitter stream
- LINQ to Active Directory – Access AD
- DbLinq – Access MySQL, PostgreSql, …
- …

Using LINQ to query databases

```
var r = Users.Where(u => u.Name == "Peter");
```



Using LINQ to query databases

```
var r = Users.Where(u => u.Name == "Peter");
```

"Interpreted query"/"Remote query"


```
SELECT [t0].[Id], [t0].[Name], [t0].[FirstName], [t0].[Value]
FROM [dbo].[Users] AS [t0]
WHERE ([t0].[Name] = @p0)
-- @p0: Input VarChar (Size = 8000; Prec = 0; Scale = 0) [Peter]
```

Local vs. Interpreted queries

- Local queries (`IEnumerable<T>`)
  - Uses delegates:

```
public static IEnumerable<TSource> Where<TSource>(
        this IEnumerable<TSource> source,
        Func<TSource, bool> predicate)
```

  - Chaining of regular method calls

- Interpreted Queries (`IQueryable<T>`)
  - Compiled into **expression trees** in IL Code:

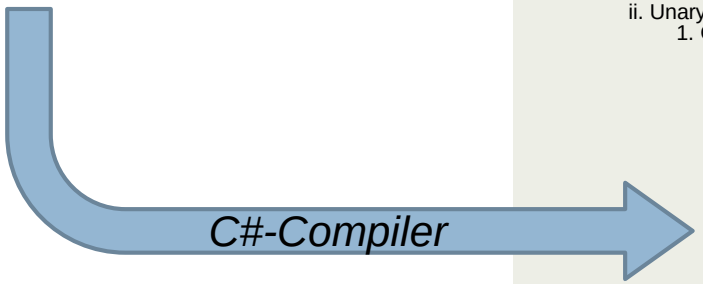
```
public static IQueryable<TSource> Where<TSource>(
        this IQueryable<TSource> source,
        Expression<Func<TSource, bool>> predicate)
```

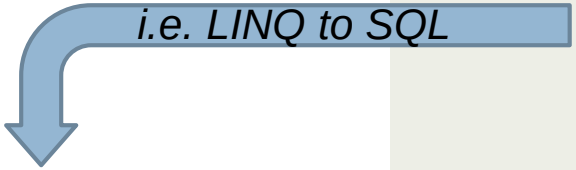
  - «Transformed/compiled» at runtime by the LINQ-provider (e.g. convert C# to SQL for databases)

Expression trees

```
users.Where(user =>
        user.Id==1);
```

C#





SQL

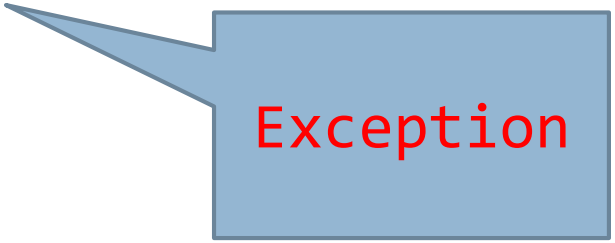
```
SELECT [t0].[Id], [t0].[Name],
[t0].[FirstName], [t0].[Value]
FROM [dbo].[Users] AS [t0]
WHERE ([t0].[Id] = @p0)
-- @p0: Input int [1]
```

1. MethodCallExpression
    a. Method : MethodInfo : "Where"
    b. Arguments : ReadOnlyCollection
        i. ConstantExpression
            1. Value : Object : "LINQConsoleApplication1.User[]"
            2. NodeType : ExpressionType : "Constant"
            3. Type : Type : "EnumerableQuery"
        ii. UnaryExpression
            1. Operand : ExpressionLambda
                a. Expression>
                    i. Body : ExpressionEqual
                        1. BinaryExpression
                            a. Left : ExpressionMemberAccess
                                i. MemberExpression
                        2. Expression : ExpressionParameter
                            a. ParameterExpression
                                i. Name : String : "user"
                                ii. NodeType : ExpressionType : "Parameter"
                                iii. Type : Type : "User"
                        3. Member : MemberInfo : "Int32 Id"
                        4. NodeType : ExpressionType : "MemberAccess"
                        5. Type : Type : "Int32"
                            a. Right : ExpressionConstant
                                i. ConstantExpression
                        6. Value : Object : "1"
                        7. NodeType : ExpressionType : "Constant"
                        8. Type : Type : "Int32"
                            a. Method : MethodInfo : null
                            b. Conversion : LambdaExpression : null
                            c. IsLifted : Boolean : "False"
                            d. IsLiftedToNull : Boolean : "False"
                            e. NodeType : ExpressionType : "Equal"
                            f. Type : Type : "Boolean"
                                i. Parameters : ReadOnlyCollection
                                    1. ParameterExpression
                                        a. Name : String : "user"
                                        b. NodeType : ExpressionType : "Parameter"
                                        c. Type : Type : "User"
                                ii. NodeType : ExpressionType : "Lambda"
                                iii. Type : Type : "Func"
                                    1. Method : MethodInfo : null
                                    2. IsLifted : Boolean : "False"
                                    3. IsLiftedToNull : Boolean : "False"
                                    4. NodeType : ExpressionType : "Quote"
                                    5. Type : Type : "Expression>"
                                        a. NodeType : ExpressionType : "Call"
                                        b. Type : Type : "IQueryable"

Expression Tree

Enforce LINQ execution

```
var u = Users.Where(u => ComplexFunc(u.FirstName) == "Peter");
```



ComplexFunc cannot be compiled to SQL!

```
//for execution as local query
var u = Users.ToList()
      .Where(u => ComplexFunc(u.FirstName) == "Peter");
```

LINQ deferred execution


```
var myquery = Users.Where(u => u.Name == "Kurt");
// Query not yet executing!


var userCount = myquery.Count();
// Query must execute now to evaluate count
                                    //SELECT COUNT(*) FROM users WHERE Name="Kurt"
```

→ Queries are still executed as late as possible

LINQ deferred execution


```
var myquery = Users.Where(u => u.Name == "Kurt");
myquery = myquery.OrderBy(u => u.OrderBy(u.Age));
// not yet executed!!


// enforcing execution options
var list = myquery.ToList();
var list = myquery.ToArray();
var count = myquery.Count();
```

Worksheet – Part 4