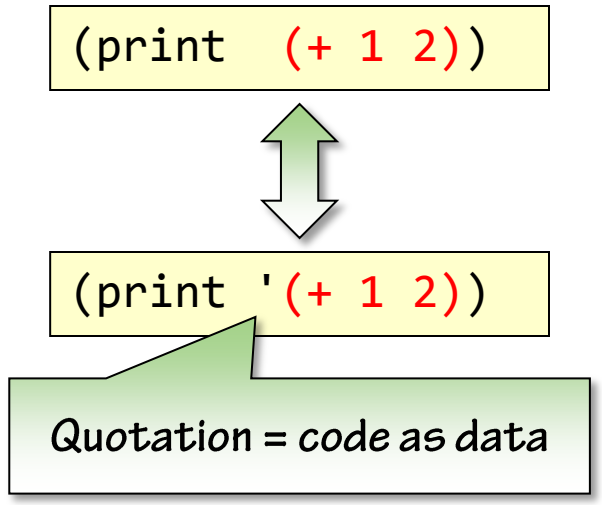
Expression Trees

Bart J.F. De Smet
bartde@outlook.com

pluralsight
hardcore dev and IT training

Representing Code as Data

- **Powerful language concept**
  - Enables meta-programming scenarios
    - Generate code at runtime
    - Rewrite code at runtime
    - Translate code at runtime
  - Sometimes referred to as quotations
    - E.g. LISP (1958)



- **Homo-iconicity**
  - Greek for:
    - Homo = same
    - Icon = representation
  - Same syntax whether
    - Code compiles into an executable form
    - Code compiles into a runtime data representation

.NET Expression Trees

- **Usage in Language Integrated Query (LINQ)**
  - Translation of expressions into query languages, e.g. T-SQL
    - E.g. IQueryable<T> query providers
  - Introduced in C# 3.0, VB 9.0, and .NET 3.5
- **Leveraged by the Dynamic Language Runtime (DLR)**
  - Compilation of dynamic languages to IL at runtime
    - E.g. IronPython translates to expression (statement) trees
  - Power of Just in Time (JIT) compilation brought to dynamic languages
- **Captured via lambda expressions**
  - Assigned to delegate types

```
Func<int, int>  f = x => x * 2;
```

  - Assigned to expression types

```
Expression<Func<int, int>> f = x => x * 2;
```

Homo-iconic

Compilation of Expression Trees

- **Lambda expression assigned to delegate type**

```
Func<int, int> f = x => x * 2;
```

```
Func<int, int> f = delegate (int x) {
  return x * 2;
};
```

Anonymous method (C# 2.0)

- **Lambda expression assigned to expression tree type**

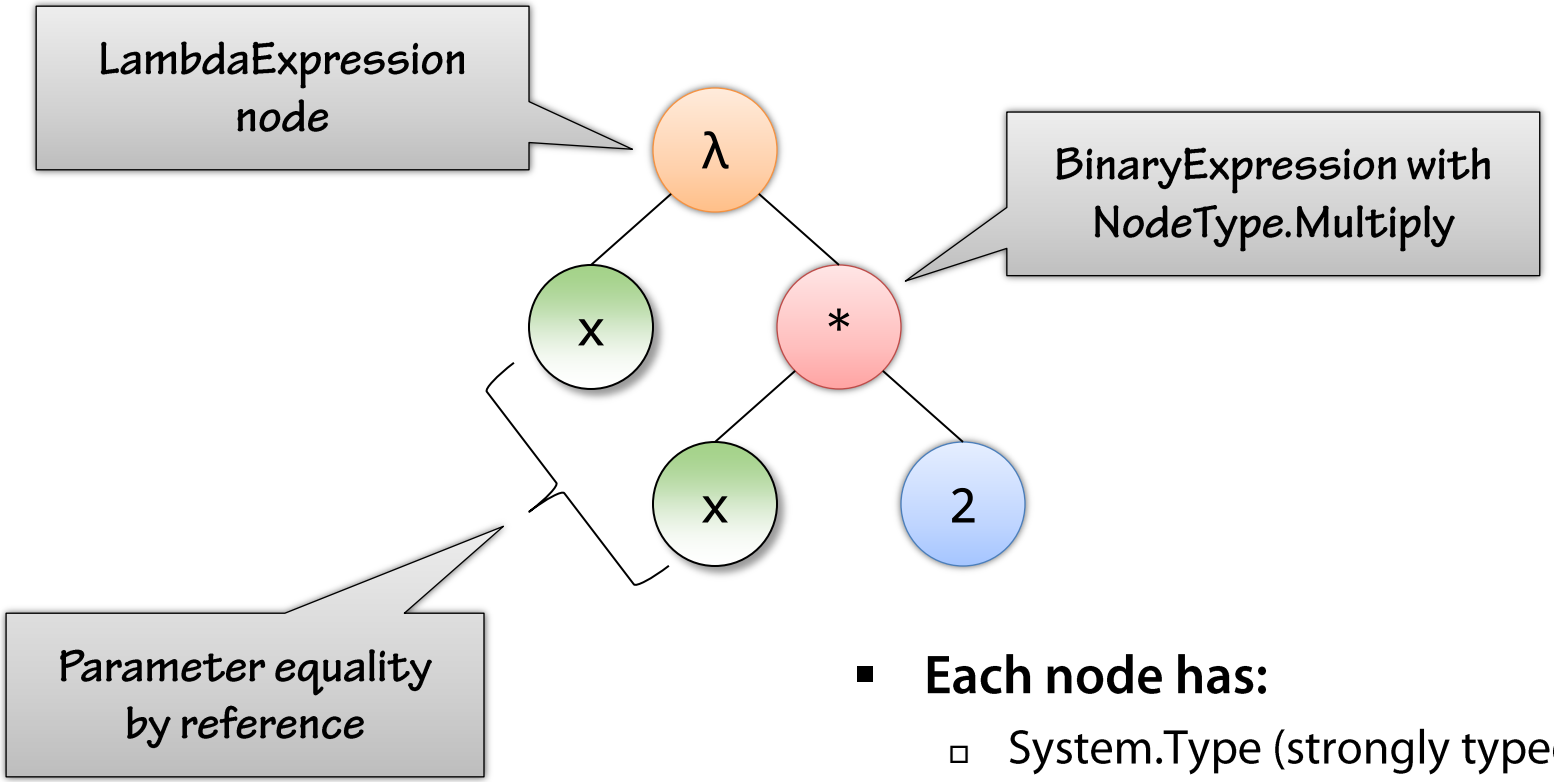
```
Expression<Func<int, int>> f = x => x * 2;
```

```
ParameterExpression x = Expression.Parameter("x", typeof(int));
Expression<Func<int, int>> f = Expression.Lambda<Func<int, int>>(
  Expression.Multiply(x, Expression.Constant(2)),
  x
);
```

Factory methods in BCL

Expression Trees in Pictures

```
Expression<Func<int, int>> f = x => x * 2;
```

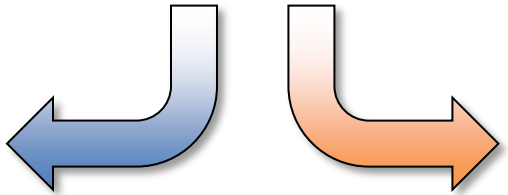


- **Each node has:**
  - System.Type (strongly typed)
  - NodeType (to switch on kind)

Usage in LINQ Query Providers

```
var res = from x in xs where x > 0 select x + 1;
```

```
var res = xs.Where(x => x > 0).Select(x => x + 1);
```



xs is IEnumerable<int>

```
var res =
  Enumerable.Select(
    Enumerable.Where(
      xs,
      x => x > 2
    ),
    x => x + 1
  );
```

LINQ to Objects

LINQ to SQL et al

xs is IQueryable<int>

```
var res =
  Queryable.Select(
    Queryable.Where(
      xs,
      x => x > 2
    ),
    x => x + 1
  );
```

```
static IEnumerable<T> Where<T>(
  this IEnumerable<T> source,
  Func<T, bool> predicate);
```

```
static IQueryable<T> Where<T>(
  this IQueryable<T> source,
  Expression<Func<T, bool>> prd);
```

Usage in LINQ Query Providers

```
static IEnumerable<T> Where<T>(this IEnumerable<T> source,
                               Func<T, bool> predicate) {
  foreach (T item in source)
    if (predicate(item))
      yield return item;
}
```

Executes in memory using iterators

```
static IQueryable<T> Where<T>(this IQueryable<T> source,
                              Expression<Func<T, bool>> predicate) {
  MethodInfo where = (MethodInfo)MethodBase.GetCurrentMethod();
  return source.Provider.CreateQuery<T>(
    Expression.Call(
      where.MakeGenericMethod(typeof(T)),
      source.Expression,
      predicate
    )
  );
}
```

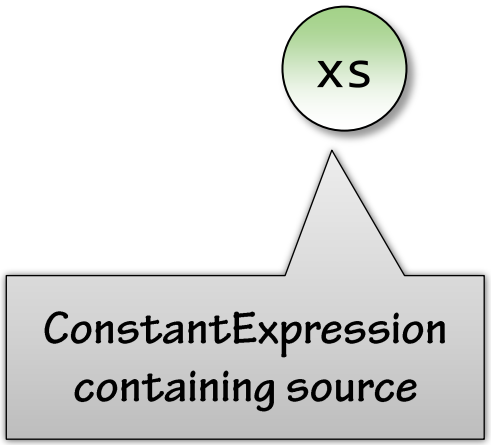
MethodCallExpression describing call to Where

Executes under control of query provider (e.g. SQL)

Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

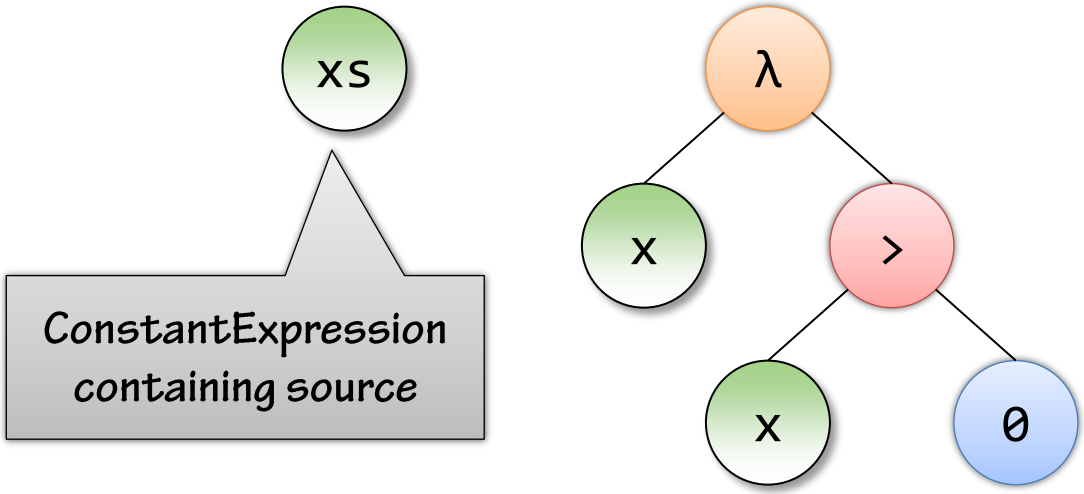
```
xs.Where(x => x > 0).Select(x => x + 1)
```



Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

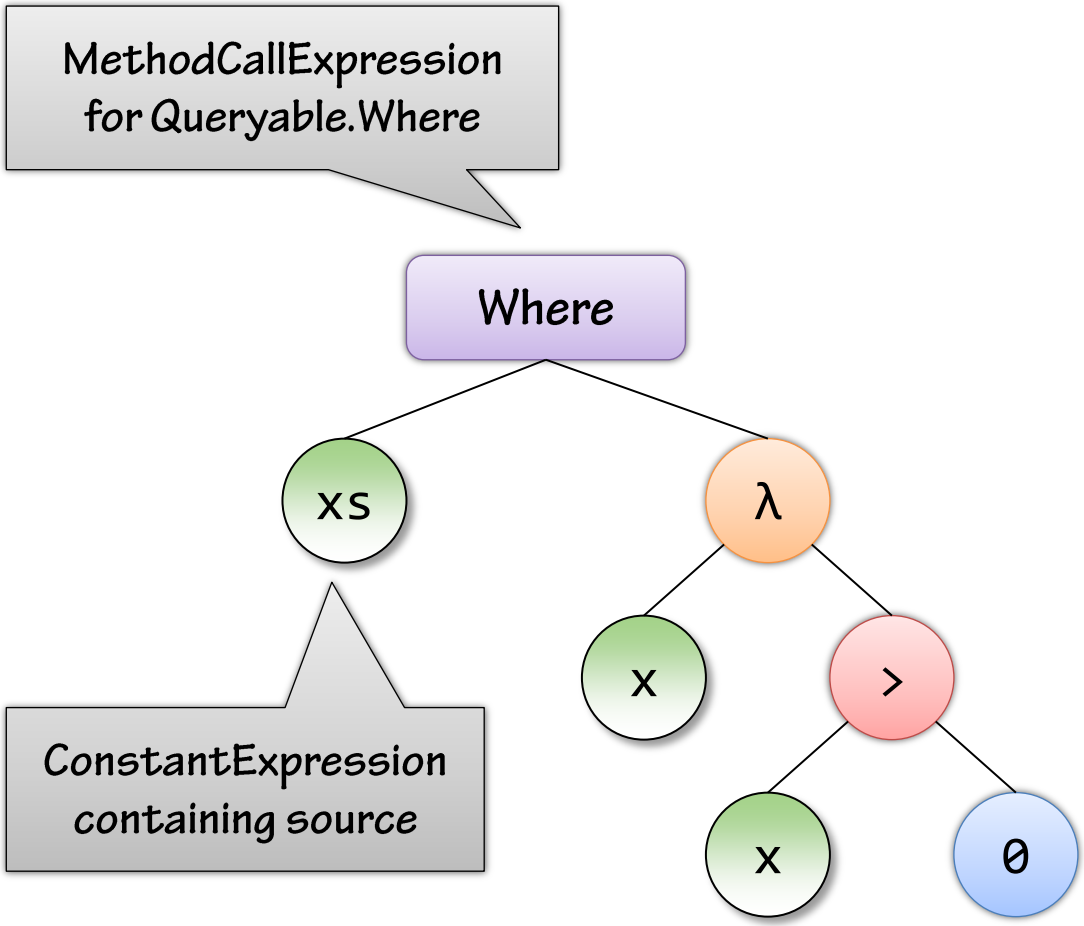
```
xs.Where(x => x > 0).Select(x => x + 1)
```



Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

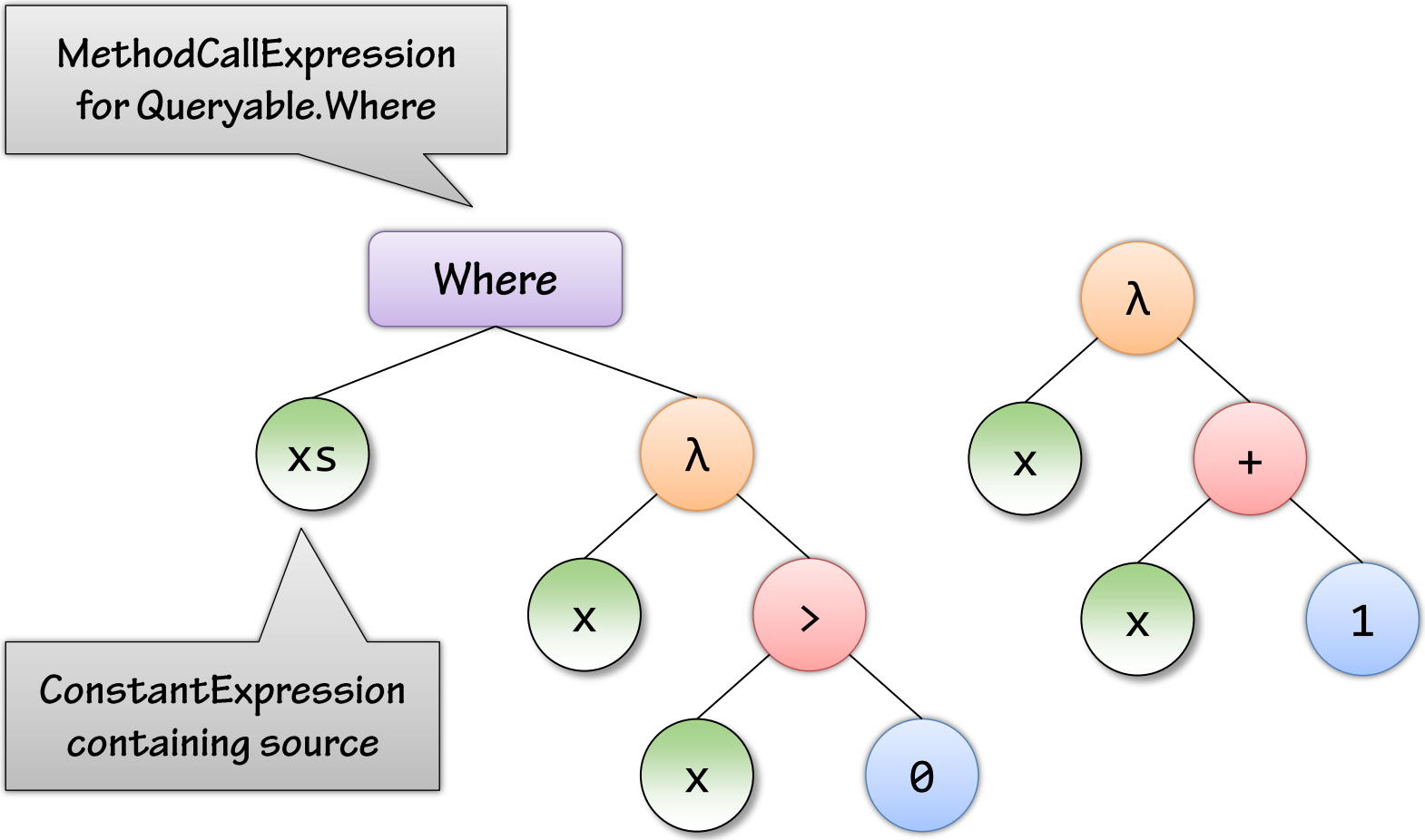
```
xs.Where(x => x > 0).Select(x => x + 1)
```



Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

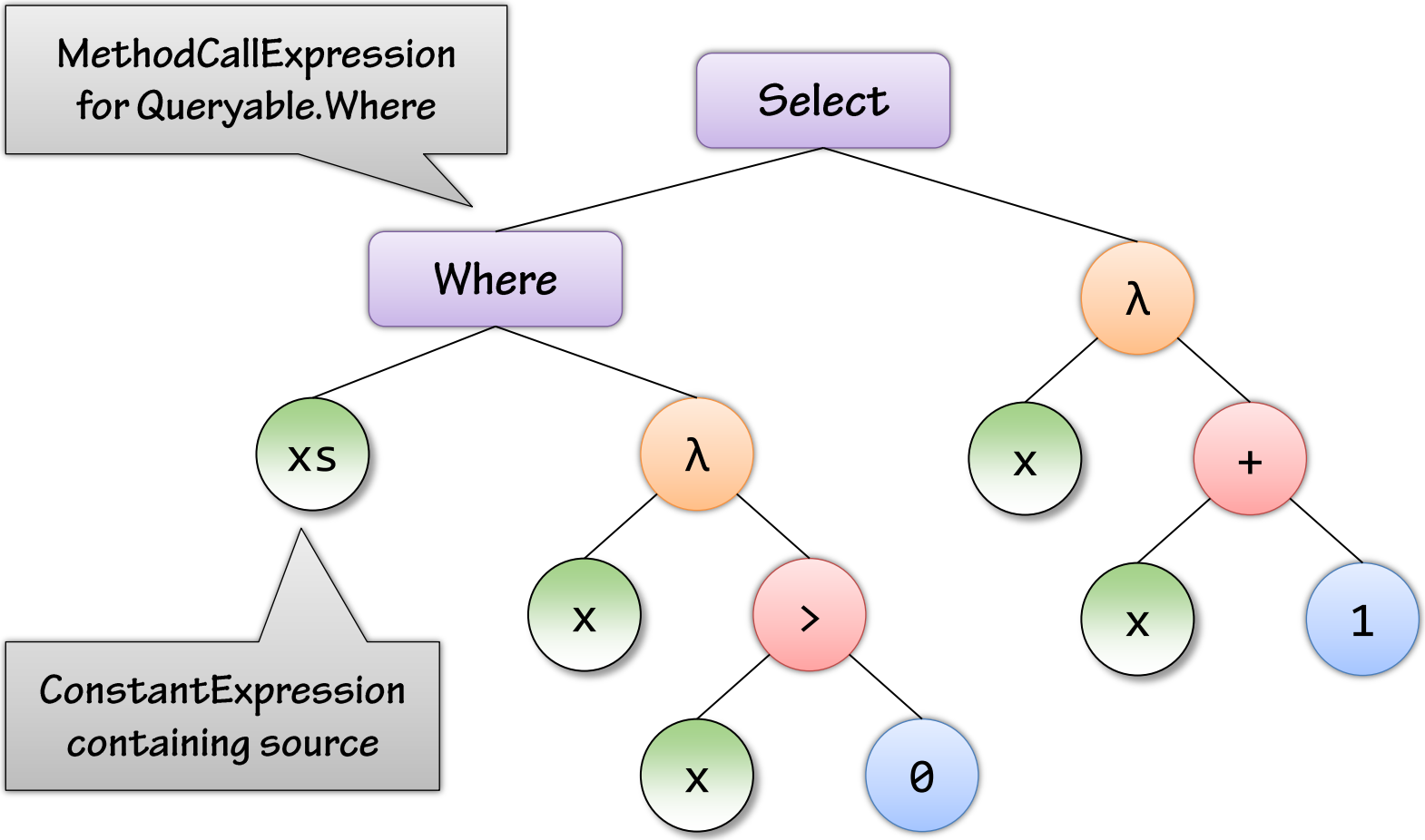
```
xs.Where(x => x > 0).Select(x => x + 1)
```



Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

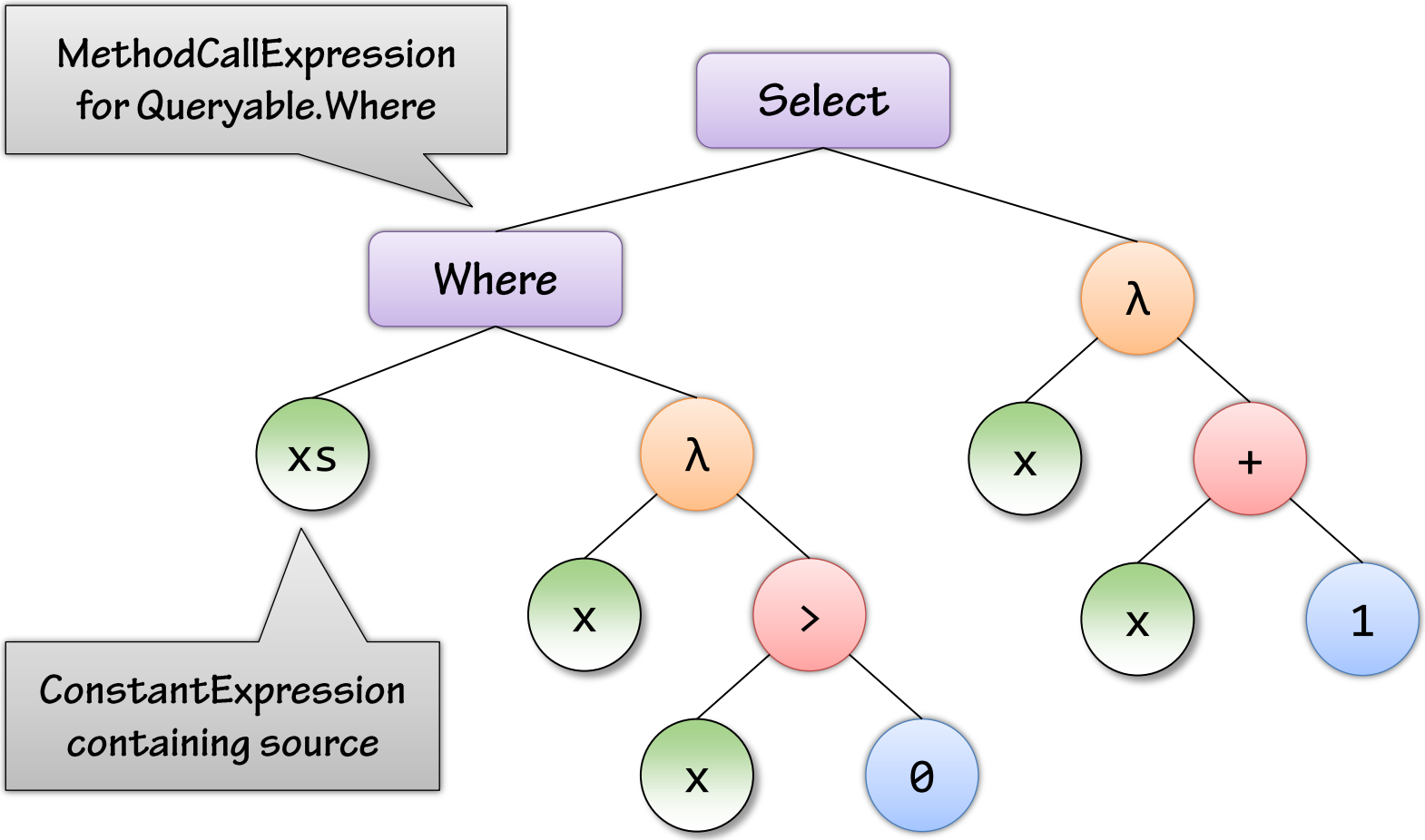
```
xs.Where(x => x > 0).Select(x => x + 1)
```



Query Expression Trees in Pictures

```
from x in xs where x > 0 select x + 1
```

```
xs.Where(x => x > 0).Select(x => x + 1)
```



Lightweight Code Generation

- **Expression trees can be compiled at runtime**
  - Compile method on
    - LambdaExpression returns Delegate
    - Expression<T> returns T
  - Uses System.Reflection.Emit under the hood
    - Generated IL code gets JIT compiled
    - Can be compiled to a MethodBuilder

Can manipulate the expression tree first

```
Expression<Func<int, int>> f = x => x * 2;
          Func<int, int>  g = f.Compile();
```

Can be invoked like any other delegate

  - Example in LINQ to Objects:
    - Using AsQueryable on an IEnumerable<T> collection
    - Enumerating the IQueryable<T> rewrites Queryable.* to Enumerable.* methods

Restrictions on Expression Trees

- **Named parameters**
  - Evaluation order matters!

```
var p = Math.Pow(y: f(), x: g());
```

```
var __t1 = f();
var __t2 = g();
var p = Math.Pow(__t2, __t1);
```

  - Could be done using lambda invocation
    - Call-by-value

- **Async methods and iterators**
  - Await expressions require an async method context
  - Iterators don't have lambda expression support at all (VB has these)

- **Compiler implementation debt**
  - Optional parameters (suffer from versioning issues)
  - Lambda expressions with statement bodies
    - Statement trees introduced in .NET 4.0 for DLR
    - Loop constructs can be quite complex to analyze

The Mythical "infoof" Operator

- **Getting reflection of types and members**
  - ldtoken instruction in IL code
  - E.g. typeof(T)

```
ldtoken T
call Type::GetTypeFromHandle(valuetype RuntimeTypeHandle)
```

- **No built-in operator to get info of method, property, constructor, etc.**
  - Rationale: would require a lot of new syntax (e.g. pick overloads)
  - Expression trees have MethodCallExpression, MemberExpression, etc.

```
static MemberInfo InfoOf<T>(Expression<Func<T>> f) {
  if (f.Body.NodeType == ExpressionType.Call)
    return ((MethodCallExpression)f.Body).Method;
  else if (f.Body.NodeType == Expression.Member)
    return ((MemberExpression)f.Body).Member;
  …
}
```

```
(MethodInfo)InfoOf(() => Console.ReadLine())
```

Summary

- **Code as data**
  - Essence of metaprogramming
  - Homo-iconicity = user convenience
- **LINQ query providers**
  - Translate expression trees into query languages
  - Leverages homo-iconicity with IQueryable<T>
- **Runtime code generation**
  - Simplification over System.Reflection.Emit
  - Generate, rewrite, analyze code at runtime
- **Tips and tricks**
  - Build mini internal domain-specific languages (DSLs)
  - Create mini language constructs as libraries (e.g. InfoOf)