Parameterized Model for File System and LINQ

Soonho Kong
Programming Research Laboratory
Seoul National University

Purpose

“*Parameterized Model* provides *efficient* and *effective* solution to the test generation of the program interacting with *environment*.”

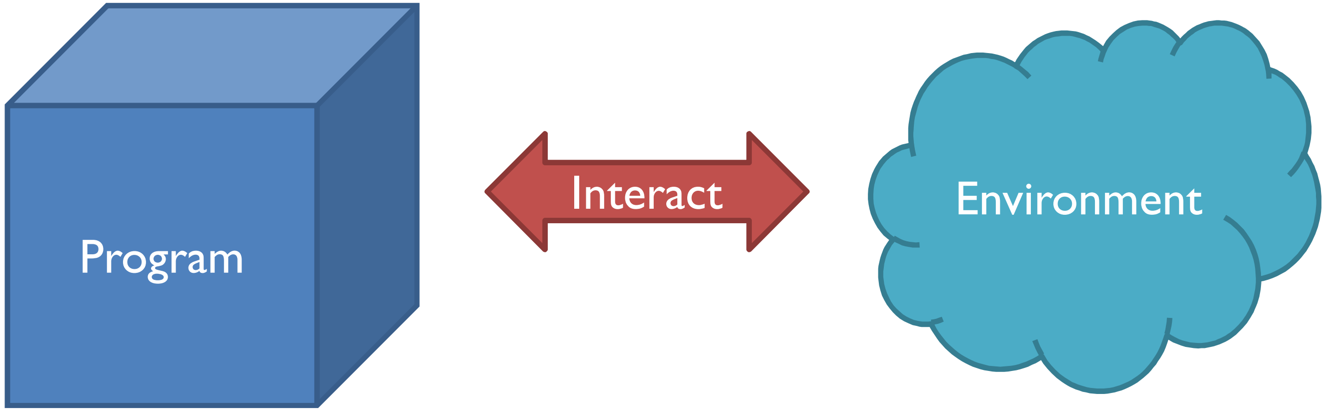
Agenda

1. Concept of Parameterized Model
2. Parameterized Model for File System
3. Parameterized Model for LINQ
4. Future of Parameterized Model

Agenda

1. **Concept of Parameterized Model**
2. Parameterized Model for File System
3. Parameterized Model for LINQ
4. Future of Parameterized Model

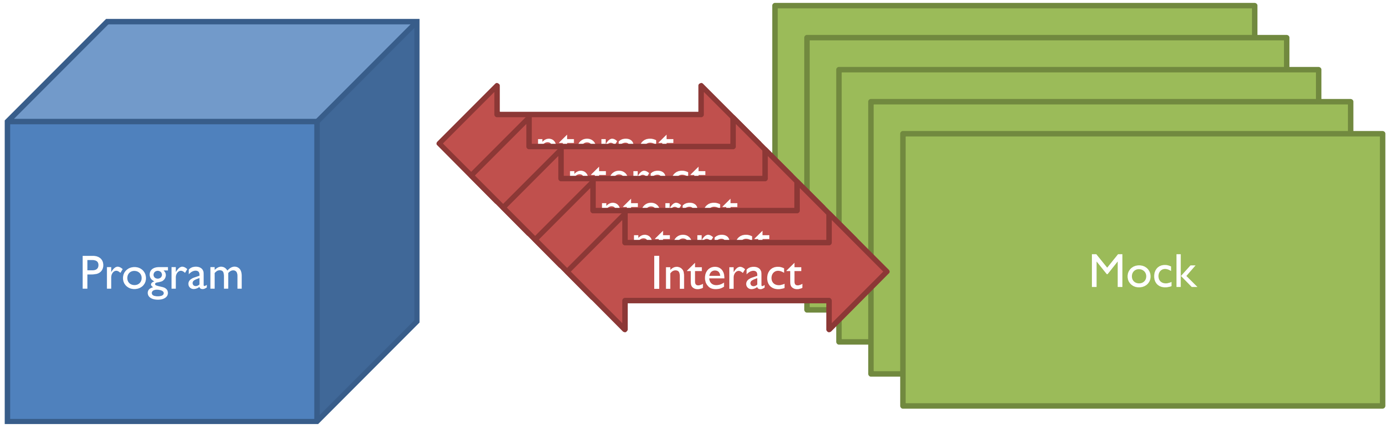
Goal & Problem



Goal : Test a program interacting with Environment

Problem : Fixed with the given Environment

Current Solution : Mocking



Good: We can program the environment.

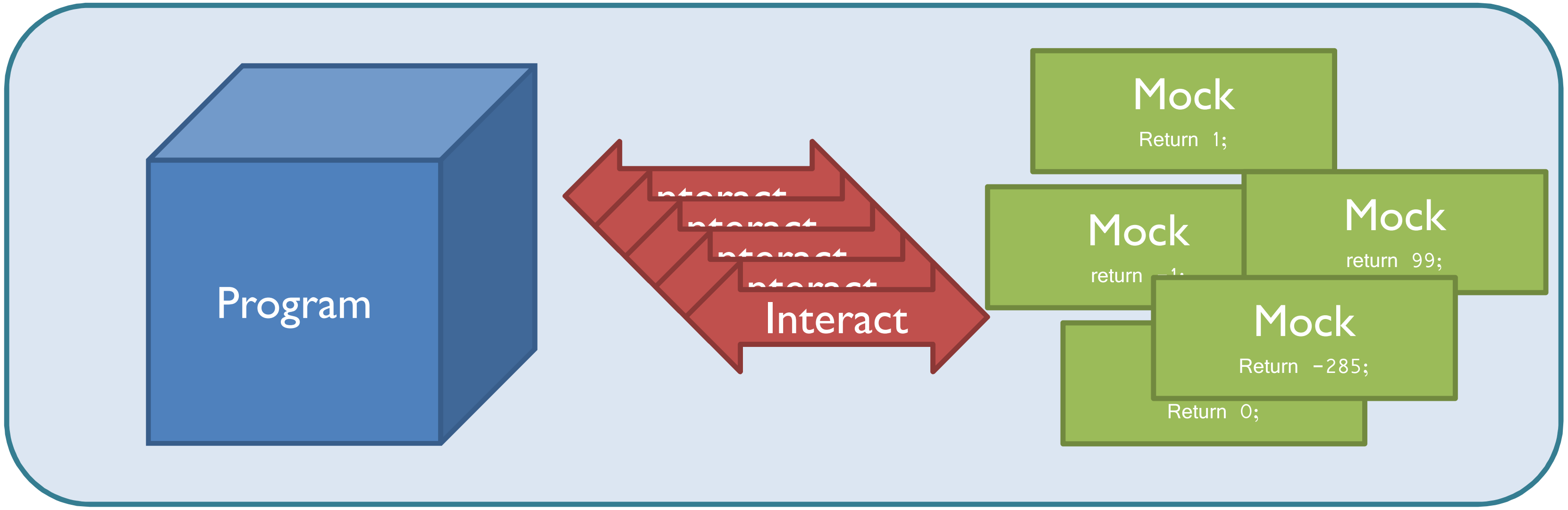
Bad:

1. You have to write it manually
2. One mock provides one behavior
3. You have no idea when you can stop
4. You might miss some corner cases

From Mocks to Models

- Instead of providing fixed behavior, Give a chance to Pex to explore and choose the behavior.
- Write a parameterized model for the environments which are used widely and frequently. So people can just take and use it.

A Model, Once and for All!



Program

Interact

Mock

Return 1;

Mock

return -1;

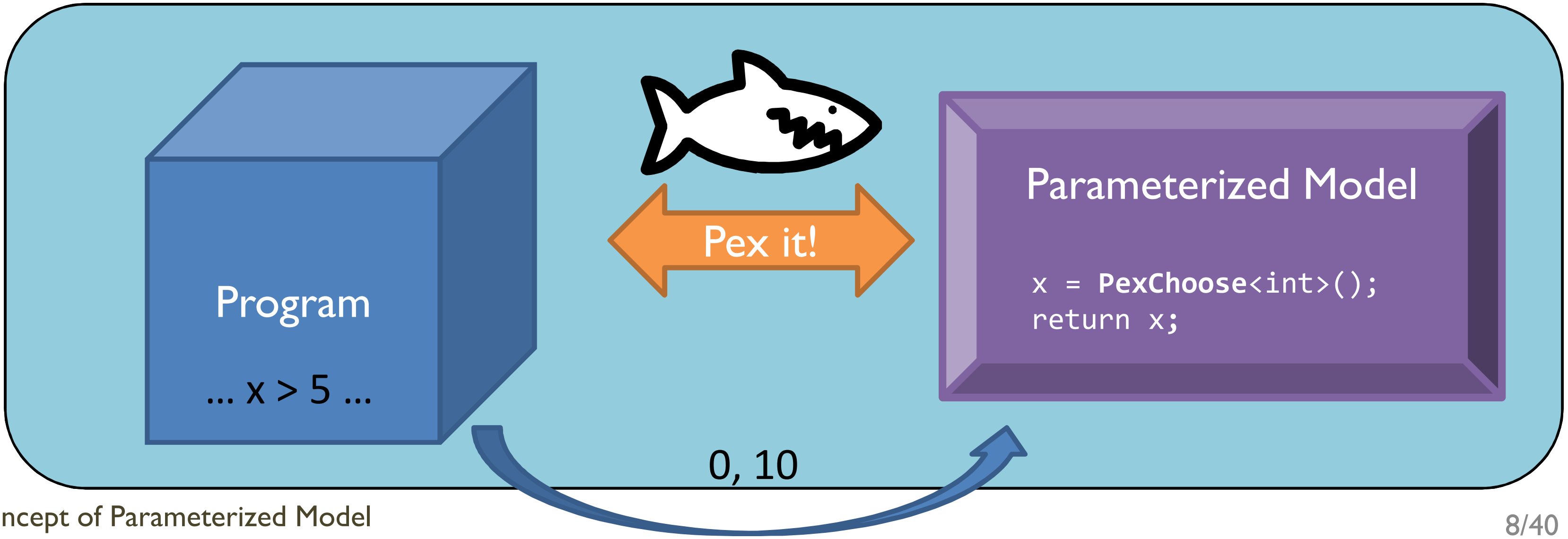
Mock

return 99;

Mock

Return -285;

Return 0;



Program

... x > 5 ...

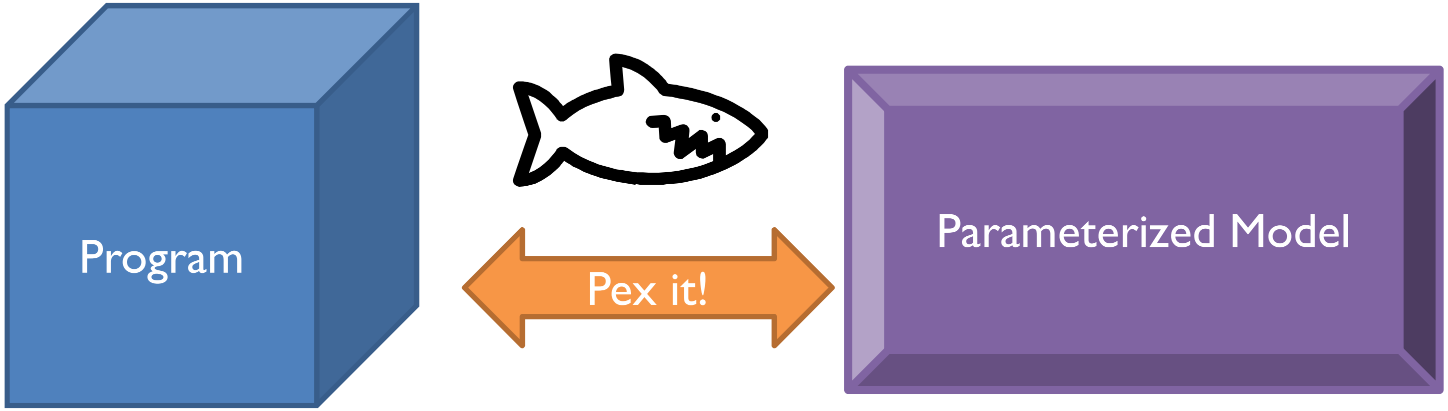
Pex it!

Parameterized Model

```
x = PexChoose<int>();
return x;
```

0, 10

Our Solution : Parameterized Model



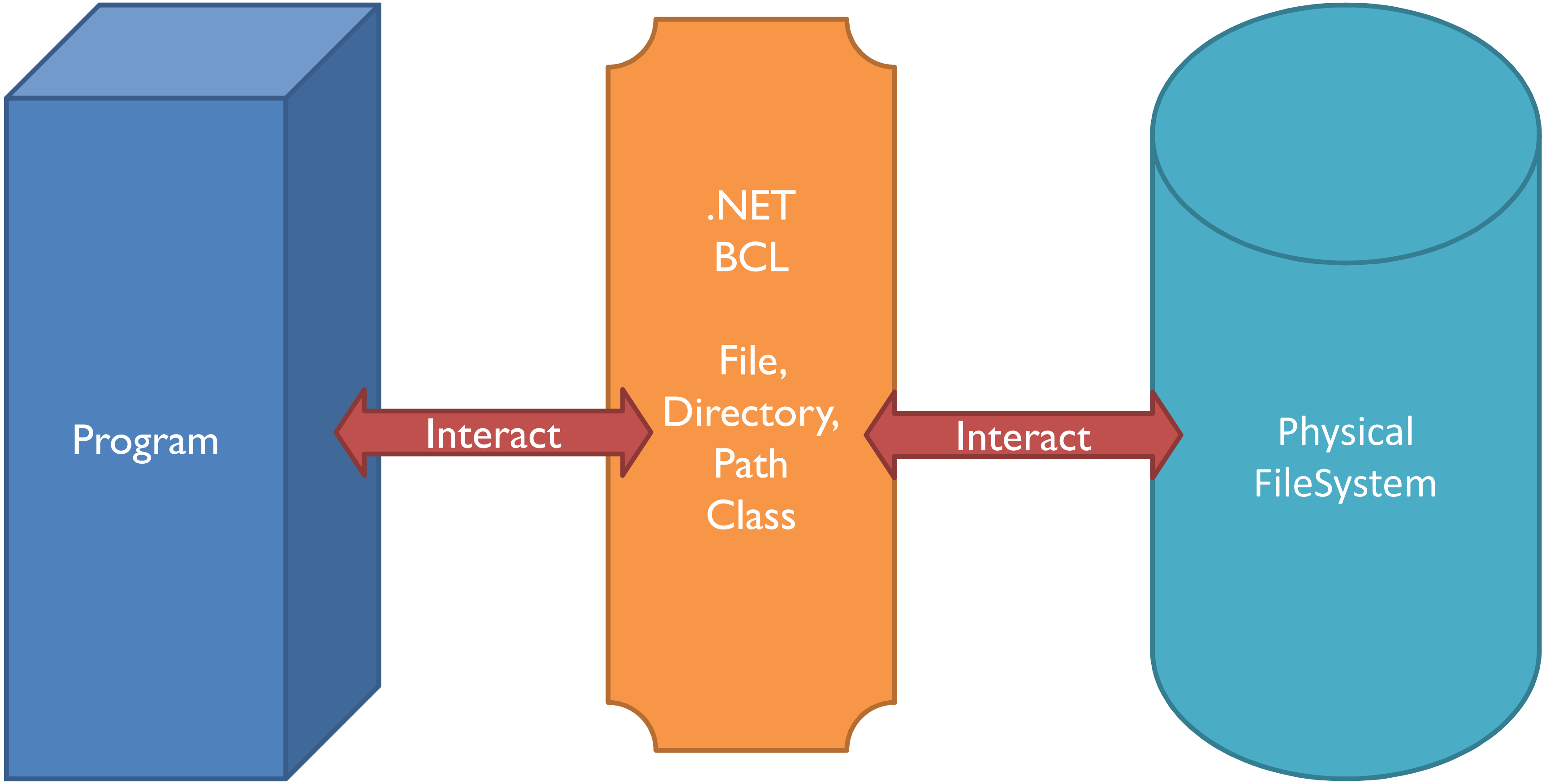
Good:

1. One model provides every possible behavior.
2. You can share the model.
3. Test against model, not against you.

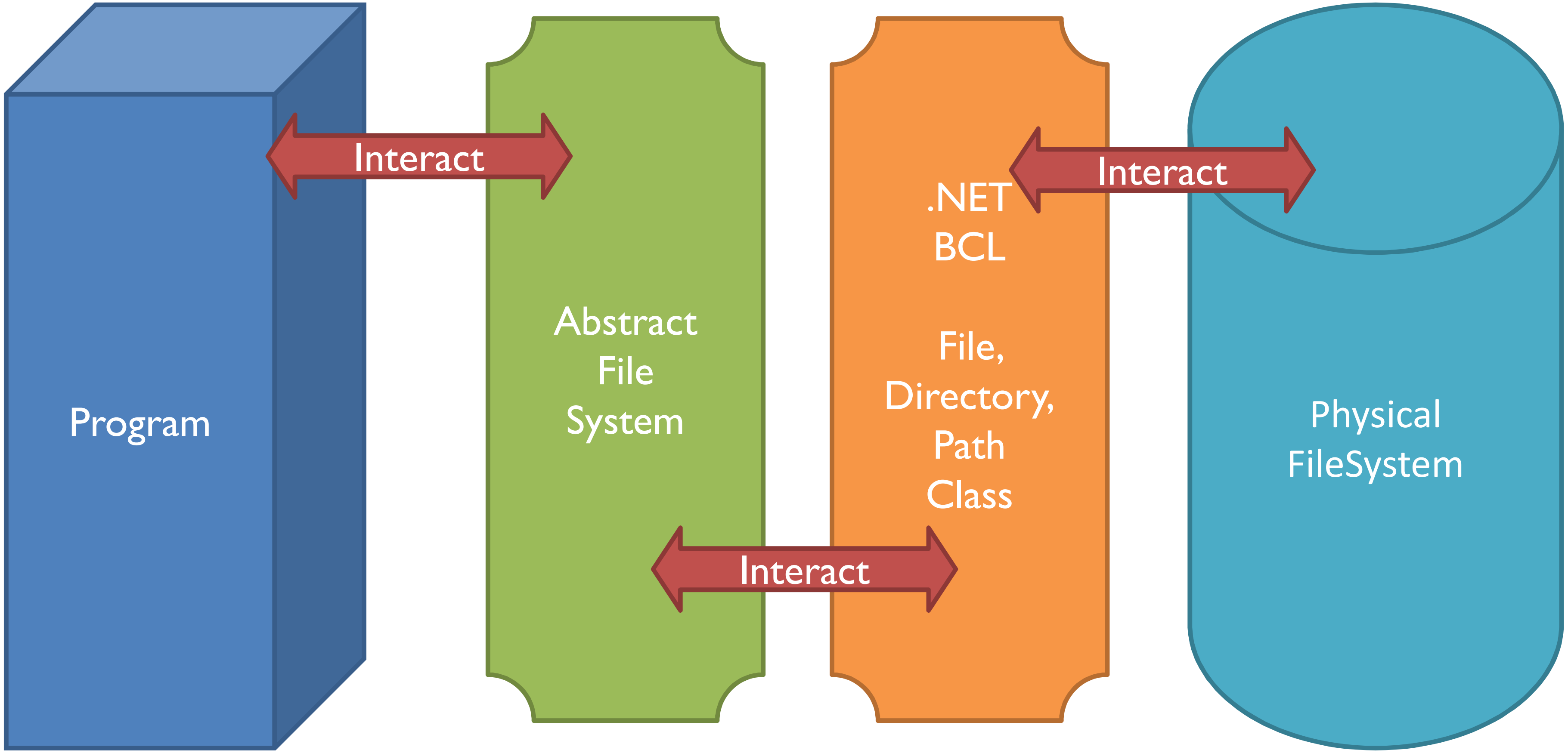
Agenda

1. Concept of Parameterized Model
2. **Parameterized Model for File System**
3. Parameterized Model for LINQ
4. Future of Parameterized Model

Program Interacting with Physical File System



Injecting Dependency



Program

Interact

Abstract File System

Interact

.NET BCL

File, Directory, Path Class

Interact

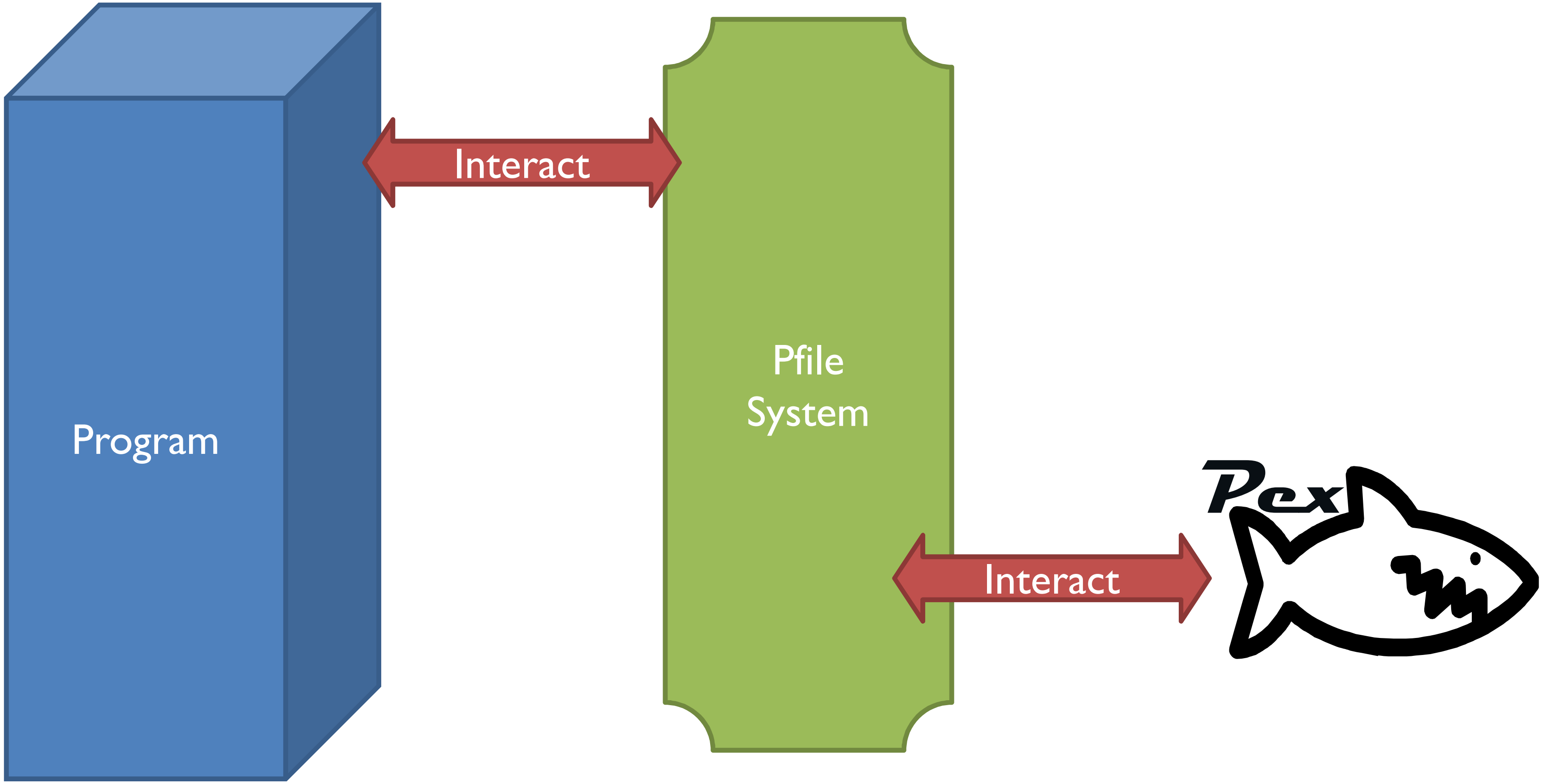
Physical FileSystem

Parameterized Model for File System

IFileSystem Interface

- Abstract layer for the file system
- Taken from CodePlex Open Source Project Community
- Containing 32 methods
  - Create/Delete File
  - Create/Delete Directory
  - Read/Write/Append File Contents
  - Retrieve All the Files/Directories in the Directory
  - Get/Set File/Directory Attributes
  - …

PFileSystem



Parameterized Model for File System

PFileSystem

- Parameterized model for the IFileSystem
- Maintains list of information
  about file system entity – file/directory.
- Gives Pex a chance to choose its behavior.
- 2529 Lines of Code, 5 Classes

PFileSystem – FileExists(path)

- fs.FileExists(@“c:\users\t-sokong\report.txt”)

Instance of PFileSystem

- It would be true, false
  - If true,
    - what is the content of this file?
    - Parent directory “c:\users\t-sokong” must exist.
    - What about the date and attributes of this file?

PFileSystem – FileExists(path)

```
// Create if possible
   if (check) {
     var call = PexChoose.FromCall(this);
     if (call.ChooseValue<bool>("Create File \"" + path + "\" or Not"))  {
       // Ensure path to file
       foreach (var dirPath in dirStack) {
         if (DirectoryExists(dirPath))
           continue;
         CreateSingleDirectory(dirPath, false);
       }
       var fileData = call.ChooseValueNotNull<byte[]>("Contents in file  " + path);
       // Create File
       if (info == null)  {
         info = new PFileInfo(ItemType.File, path, new List<byte>(fileData));
         FileInfos.Add(info);
       }
       else
```

Create only if possible

Ask Pex to Create or Not

Create a path to this file

Ask Pex about
the Content of this file

Ask Pex about
Data,Attributes…

PFileSystem

DEMO

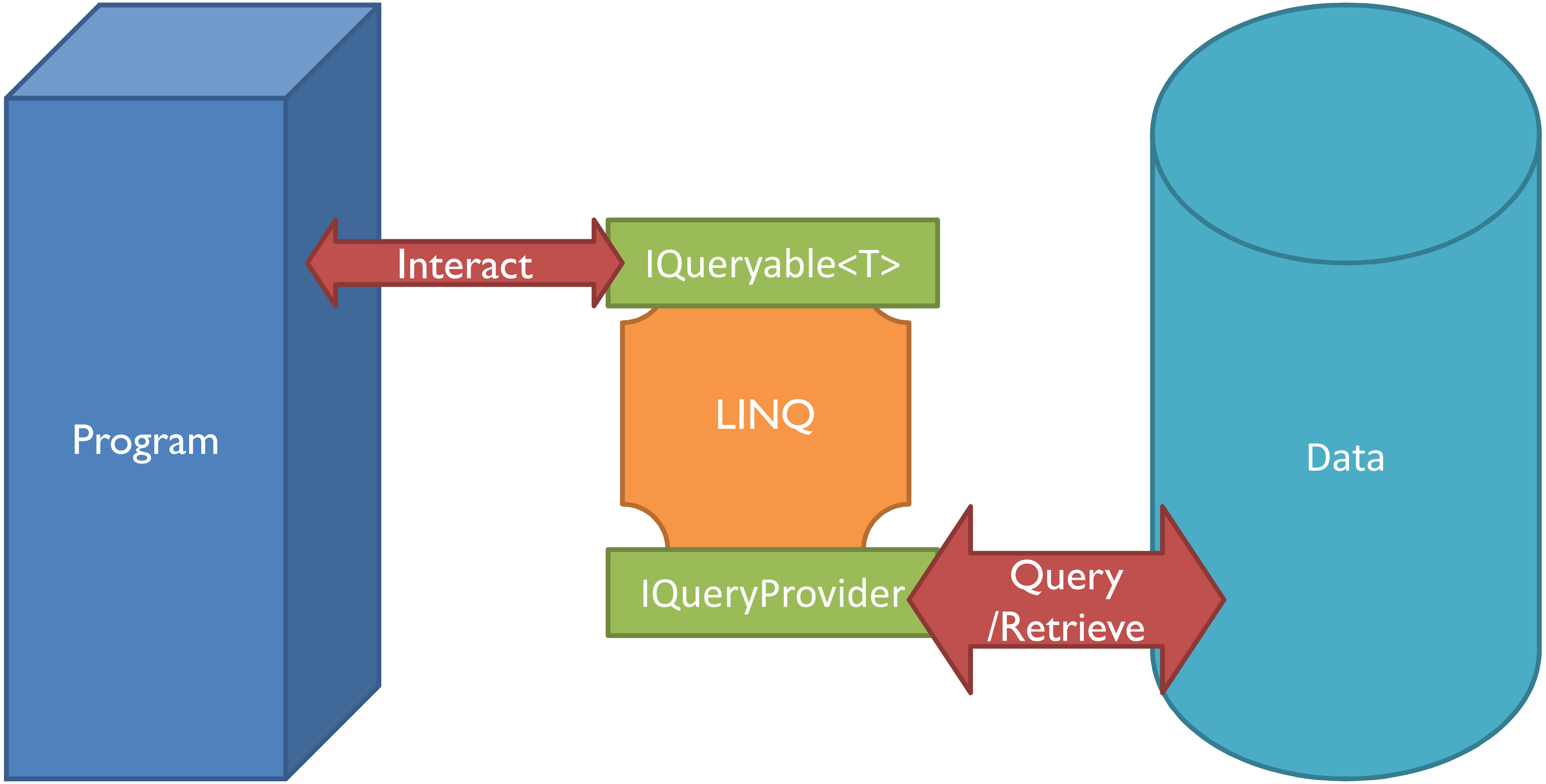
Agenda

1. Concept of Parameterized Model
2. Parameterized Model for File System
3. **Parameterized Model for LINQ**
4. Future of Parameterized Model

The LINQ Project

- Language Integrated Query(LINQ)
- Released as a part of .NET framework 3.5
- Provides a general/unified way to query data

Why We Care About LINQ



Parameterized Model for LINQ

How LINQ Works

- IQueryable<T> Interface
  - ElementType: Type of T
  - Expression: Represents the Query it will perform when executed
  - Provider: Describes how it executes the query

How LINQ Works

- Standard Query Operators (43 Operators)
  - Projection Operators: Select, SelectMany
  - Restriction Operators: Where
  - Grouping Operators: GroupBy
  - Aggregate Operators: Max, Min, Sum, Average, Count, …
  - Quantifier Operators: All, Any, Contains
  - …
- Defined in both Queryable and Enumerable class as an extension method

How LINQ Works

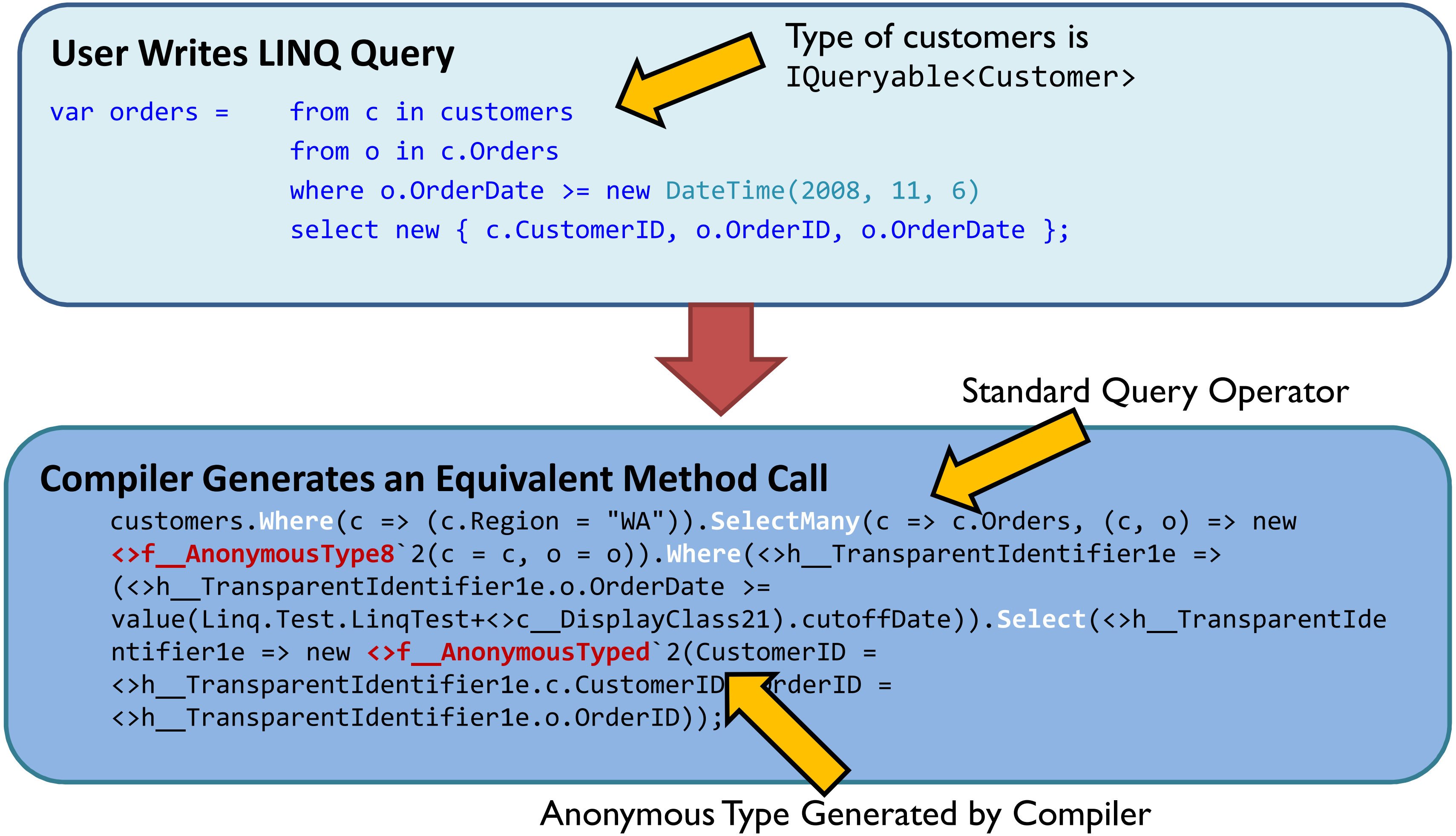
- Two types of Execution
  - Deferred Execution: Return the IQueryable which contains the expression to run. It is executed when it is actually enumerated.

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

  - Immediate Execution: Return the result immediately.

```
int Count<TSource>(this IQueryable<TSource> source)
```

How LINQ Works



User Writes LINQ Query

```
var orders =     from c in customers
                 from o in c.Orders
                 where o.OrderDate >= new DateTime(2008, 11, 6)
                 select new { c.CustomerID, o.OrderID, o.OrderDate };
```

Type of customers is `IQueryable<Customer>`

Compiler Generates an Equivalent Method Call

```
customers.Where(c => (c.Region = "WA")).SelectMany(c => c.Orders, (c, o) => new
<>f__AnonymousType8`2(c = c, o = o)).Where(<>h__TransparentIdentifier1e =>
(<>h__TransparentIdentifier1e.o.OrderDate >=
value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__TransparentIde
ntifier1e => new <>f__AnonymousTyped`2(CustomerID =
<>h__TransparentIdentifier1e.c.CustomerID rderID =
<>h__TransparentIdentifier1e.o.OrderID));
```

Standard Query Operator

Anonymous Type Generated by Compiler

How LINQ Works

Compiler Generates an Equivalent Method Call

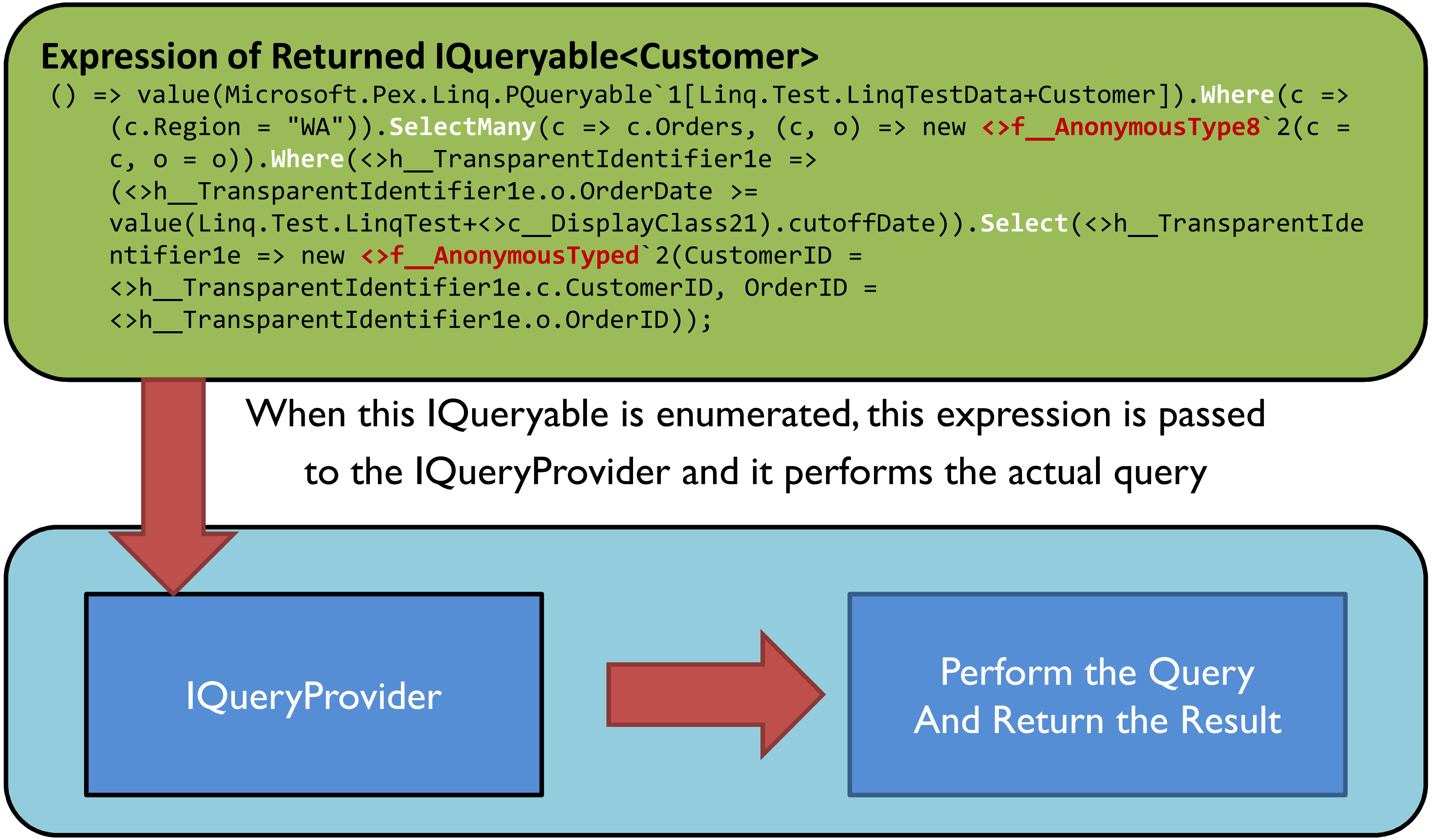
```
customers.Where(c => (c.Region = "WA")).SelectMany(c => c.Orders, (c, o) => new
   <>f__AnonymousType8`2(c = c, o = o)).Where(<>h__TransparentIdentifier1e =>
   (<>h__TransparentIdentifier1e.o.OrderDate >=
   value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__TransparentIde
   ntifier1e => new <>f__AnonymousTyped`2(CustomerID =
   <>h__TransparentIdentifier1e.c.CustomerID, OrderID =
   <>h__TransparentIdentifier1e.o.OrderID));
```

After it is executed, it returns an IQueryable<Customer> whose expression is

Expression of Returned IQueryable<Customer>

```
() => value(Microsoft.Pex.Linq.PQueryable`1[Linq.Test.LinqTestData+Customer]).Where(c => (c.Region = "WA")).SelectMany(c => c.Orders, (c,
   o) => new <>f__AnonymousType8`2(c = c, o = o)).Where(<>h__TransparentIdentifier1e => (<>h__TransparentIdentifier1e.o.OrderDate >=
   value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__TransparentIdentifier1e => new
   <>f__AnonymousTyped`2(CustomerID = <>h__TransparentIdentifier1e.c.CustomerID, OrderID =
   <>h__TransparentIdentifier1e.o.OrderID));
```

How LINQ Works



Expression of Returned IQueryable<Customer>

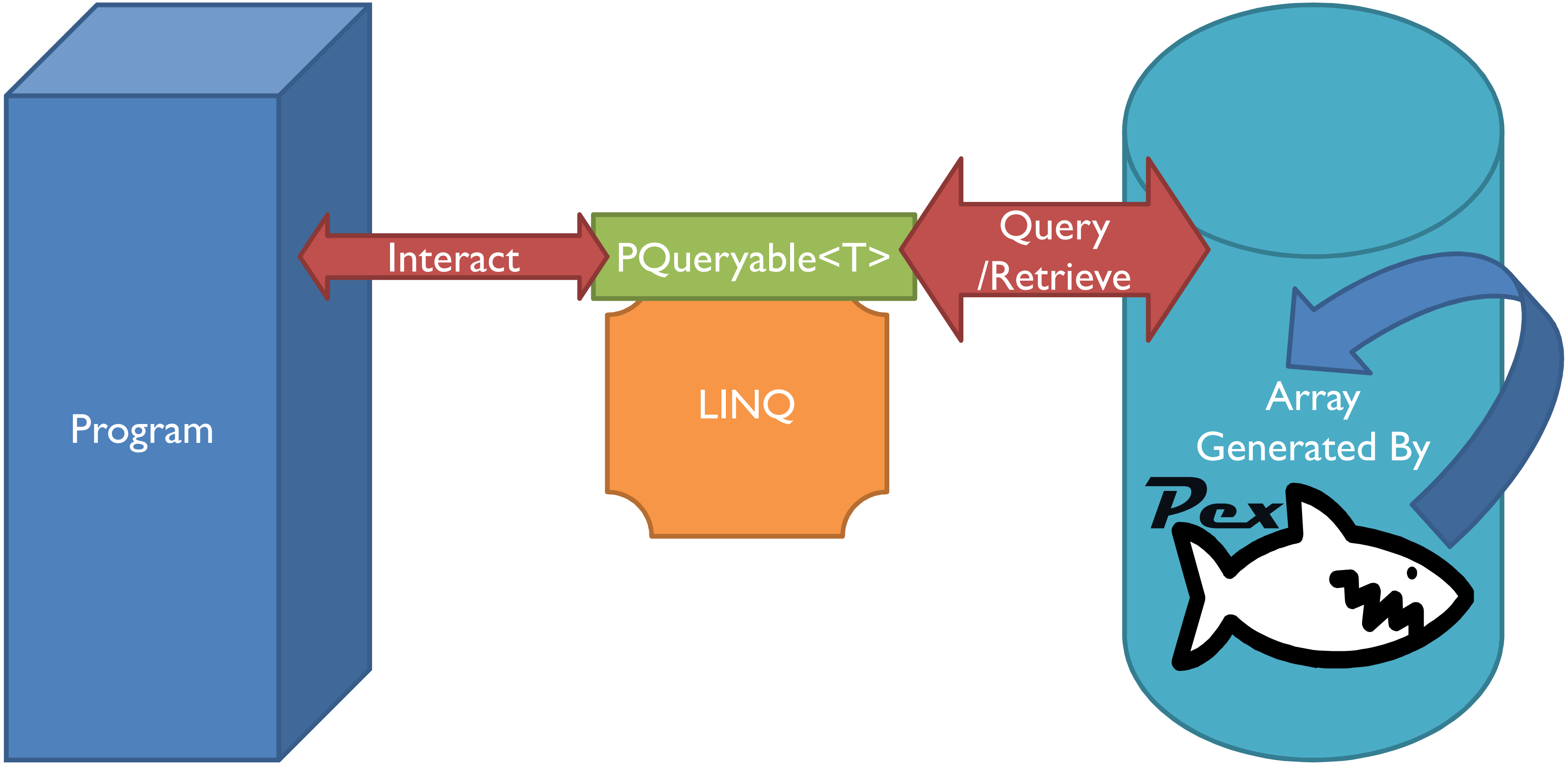
```
() => value(Microsoft.Pex.Linq.PQueryable`1[Linq.Test.LinqTestData+Customer]).Where(c =>
    (c.Region = "WA")).SelectMany(c => c.Orders, (c, o) => new <>f__AnonymousType8`2(c =
    c, o = o)).Where(<>h__TransparentIdentifier1e =>
    (<>h__TransparentIdentifier1e.o.OrderDate >=
    value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__TransparentIde
    ntifier1e => new <>f__AnonymousTyped`2(CustomerID =
    <>h__TransparentIdentifier1e.c.CustomerID, OrderID =
    <>h__TransparentIdentifier1e.o.OrderID));
```

When this IQueryable is enumerated, this expression is passed to the IQueryProvider and it performs the actual query

IQueryProvider

Perform the Query
And Return the Result

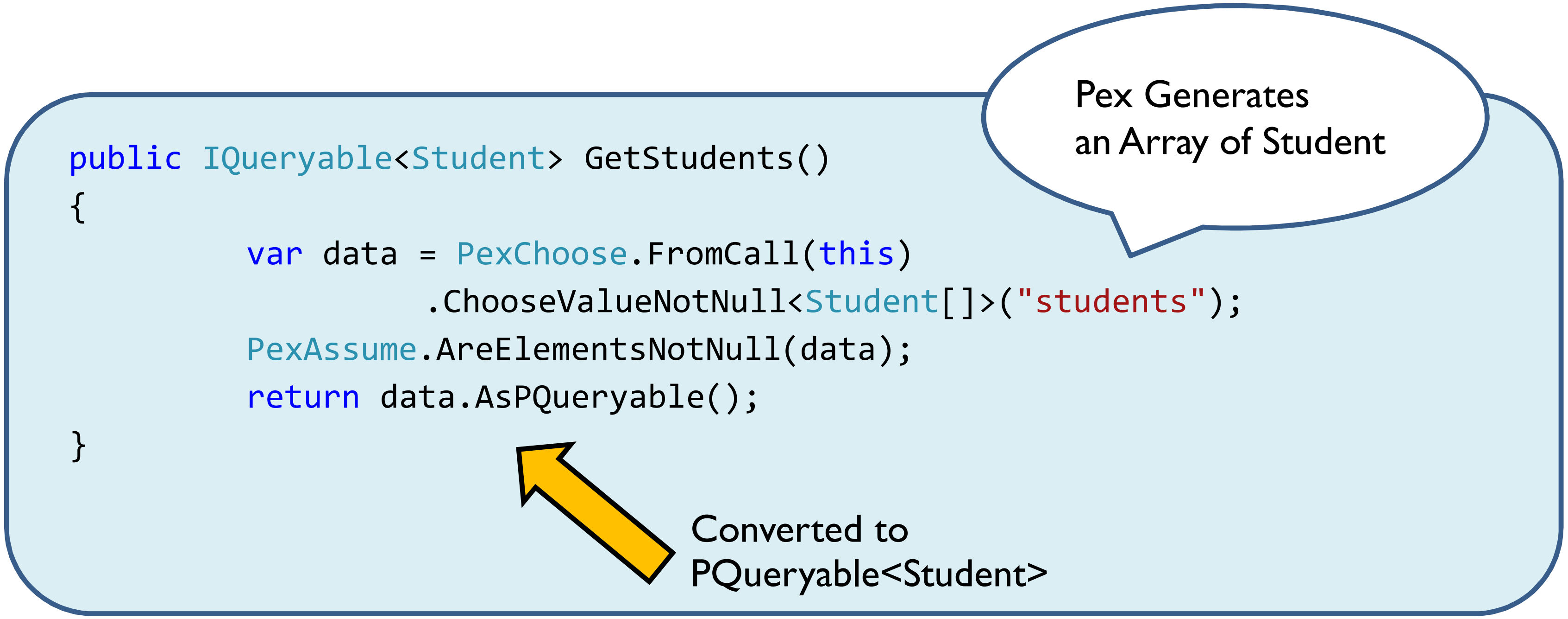
Big Picture: Pex.LINQ



PQueryable<T>

- Implementation of IQueryable<T>
- Instantiated from *AsPQueryable* method

```
IQueryable<TElement> AsPQueryable<TElement>(this IEnumerable<TElement> source)
```



```
public IQueryable<Student> GetStudents()
{
         var data = PexChoose.FromCall(this)
                  .ChooseValueNotNull<Student[]>("students");
         PexAssume.AreElementsNotNull(data);
         return data.AsPQueryable();
}
```

Pex Generates
an Array of Student

Converted to
PQueryable<Student>

PQueryable<T>

- It also Implements IQueryProvider interface
- When executing an Expression,

Expression of Returned IQueryable<Customer>

```
() => value(Microsoft.Pex.Linq.PQueryable`1[Linq.Test.LinqTestData+Customer]).Where(c =>
    (c.Region = "WA")).SelectMany(c => c.Orders, (c, o) => new <>f__AnonymousType8`2(c =
    c, o = o)).Where(<>h__TransparentIdentifier1e =>
    (<>h__TransparentIdentifier1e.o.OrderDate >=
    value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__TransparentIde
    ntifier1e => new <>f__AnonymousTyped`2(CustomerID =
    <>h__TransparentIdentifier1e.c.CustomerID, OrderID =
    <>h__TransparentIdentifier1e.o.OrderID));
```

3. Invoke It!

Issue 1

- **Problem**: ExpressionCompiler uses "Lightweight" Code Generation which Pex cannot monitor and instrument
  - "Limitation" of the CLR – Won't Fix.
- **Solution**: Substitute ExpressionCompiler to create a delegate using "Heavyweight" Code Generation

Issue 2

- **Problem**: With "Lightweight" Code Generation we could skip the "Visibility Check". We cannot skip it when we use "Heavyweight" Code Generation.
- **Solution**: Traverse Expression and change any access to the private class, field, property, method, and constructor into the equivalent method call using reflection.

Example

```
obj.PrivateField → PrivateFieldInfo.GetValue(obj)
```

Issue 3

- **Problem:** Pex iterates dynamic symbolic execution and it leads to repeated creation of the same code generated method. It generates redundant test cases.

Expression of Returned IQueryable<Customer>

```
() => value(Microsoft.Pex.Linq.PQueryable`1[Linq.Test.LinqTestData.Customer]).Where(c =>
    (c.Region = "WA")).SelectMany(c => c.Orders, (c, o) => new <>f__AnonymousType8`2(c =
    c, o = o)).Where(<>h__TransparentIdentifier1e =>
    (<>h__TransparentIdentifier1e.o.OrderDate >=
    value(Linq.Test.LinqTest+<>c__DisplayClass21).cutoffDate)).Select(<>h__Tran
    sparentIdentifier1e => new <>f__AnonymousTyped`2(CustomerID =
    <>h__TransparentIdentifier1e.c.CustomerID, OrderID =
    <>h__TransparentIdentifier1e.o.OrderID));
```

Keep Creating New Code for This

- **Solution:** Implement ExpressionComparer and Create new method only if it is new lambda expression.

Pex.LINQ

- 1832 Lines of Code
- 9 Classes
- Substitution
  - 3 Methods in System.Linq.ExpressionCompiler
  - 1 Method in System.Runtime.CompilerServices.ExecutionScope

Pex.LINQ

DEMO

Evaluation Result

- ExpressionCompilerTest suite
  - Covers every type of LINQ expression.
  - 61 Tests, 155 Generated Tests, 100% dynamic coverage
- LINQ101SampleTest suite
  - LINQ 101 Sample from the official LINQ website
  - More queries from the Standard Query Operators document
  - Covers every type of the Standard Query Operators

Agenda

1. Concept of Parameterized Model
2. Parameterized Model for File System
3. Parameterized Model for LINQ
4. **Future of Parameterized Model**

Future Work

- Modeling more and more environment parts
- Introducing new interfaces that abstract "static" (untestable) APIs
- Investigate how to make it easier to write such models

Future Testing Revolution

- Provide ability to "Save" initial environment model state to reality
  - Then generated tests can not only be executed against model, but also against reality
  - Then PUTs give rise to unit tests and integration tests!
  - Also useful to validate models against reality
- Same Test for unit test and integration test!

Thank You!