Building Languages With The Dynamic Language Runtime

Martin.Maly@microsoft.com

Dynamic Languages

- Popular
- Powerful
- Simple
- Intuitive
- Interactive
- Inspiring
- Fun

JavaScript

Lua

python™

php

Microsoft® Visual Basic®

Dynamic Language Runtime

- Platform for building dynamic languages
- Built on top of Microsoft .NET Framework
  - Garbage collector
  - Just-in-time compiler (JIT)
  - Rich libraries
  - Tools

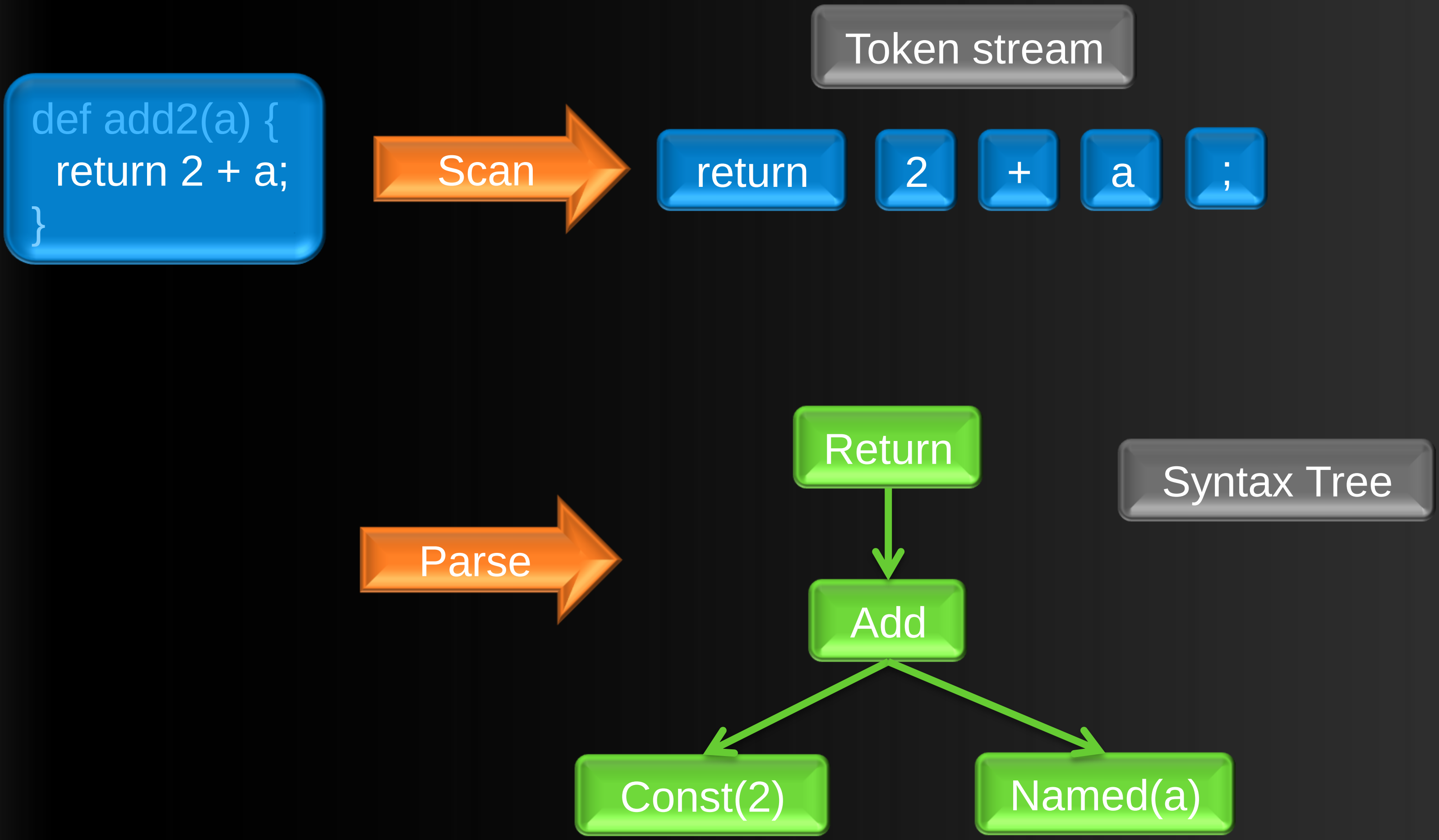
http://www.codeplex.com/IronPython

Demo

ToyScript

Compiler Overview

Frontend



```
def add2(a) {
  return 2 + a;
}
```

Scan

Token stream

return | 2 | + | a | ;

Parse

Syntax Tree

Return

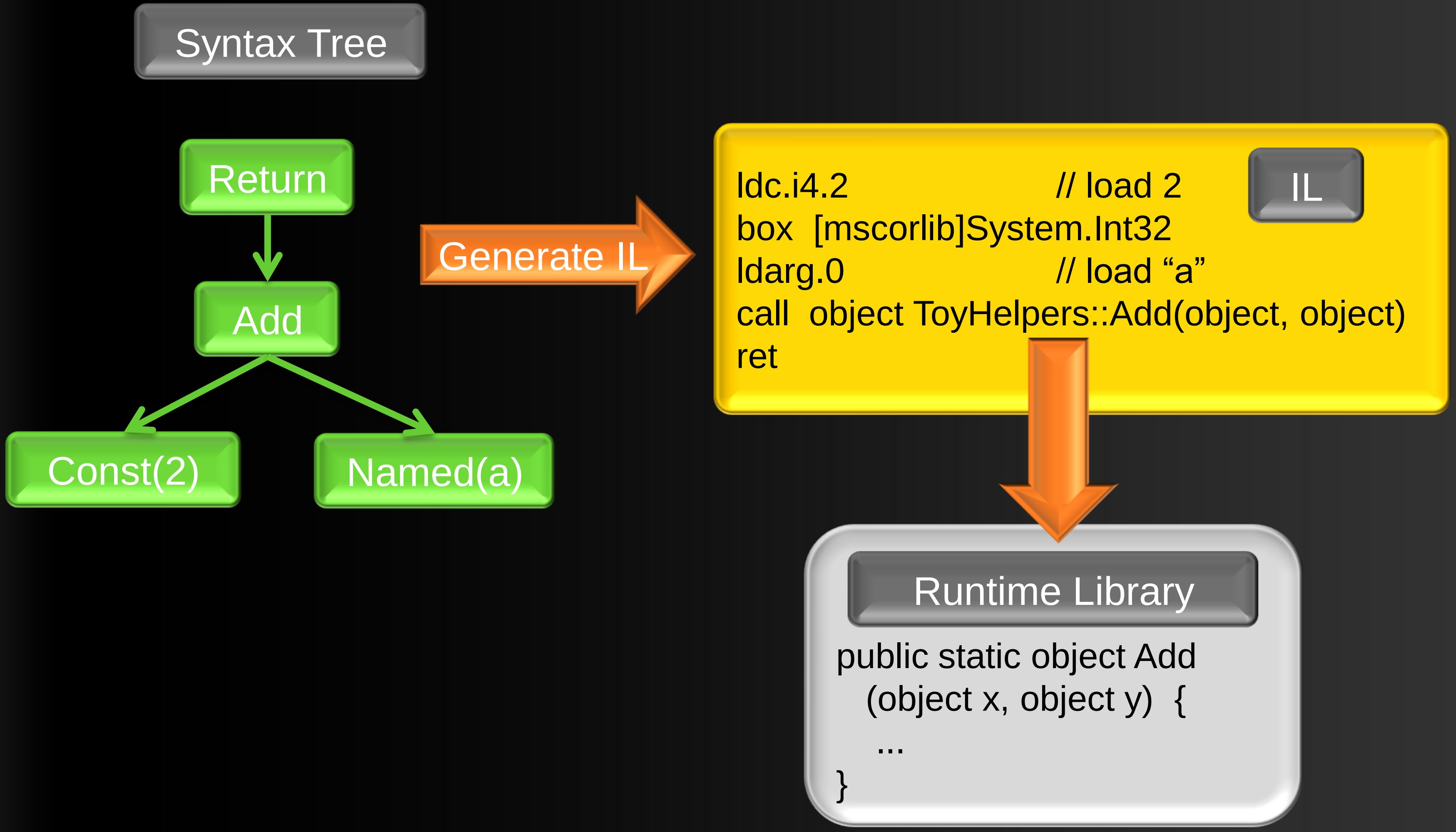
Add

Const(2)

Named(a)

Compiler Overview

Traditional Backend

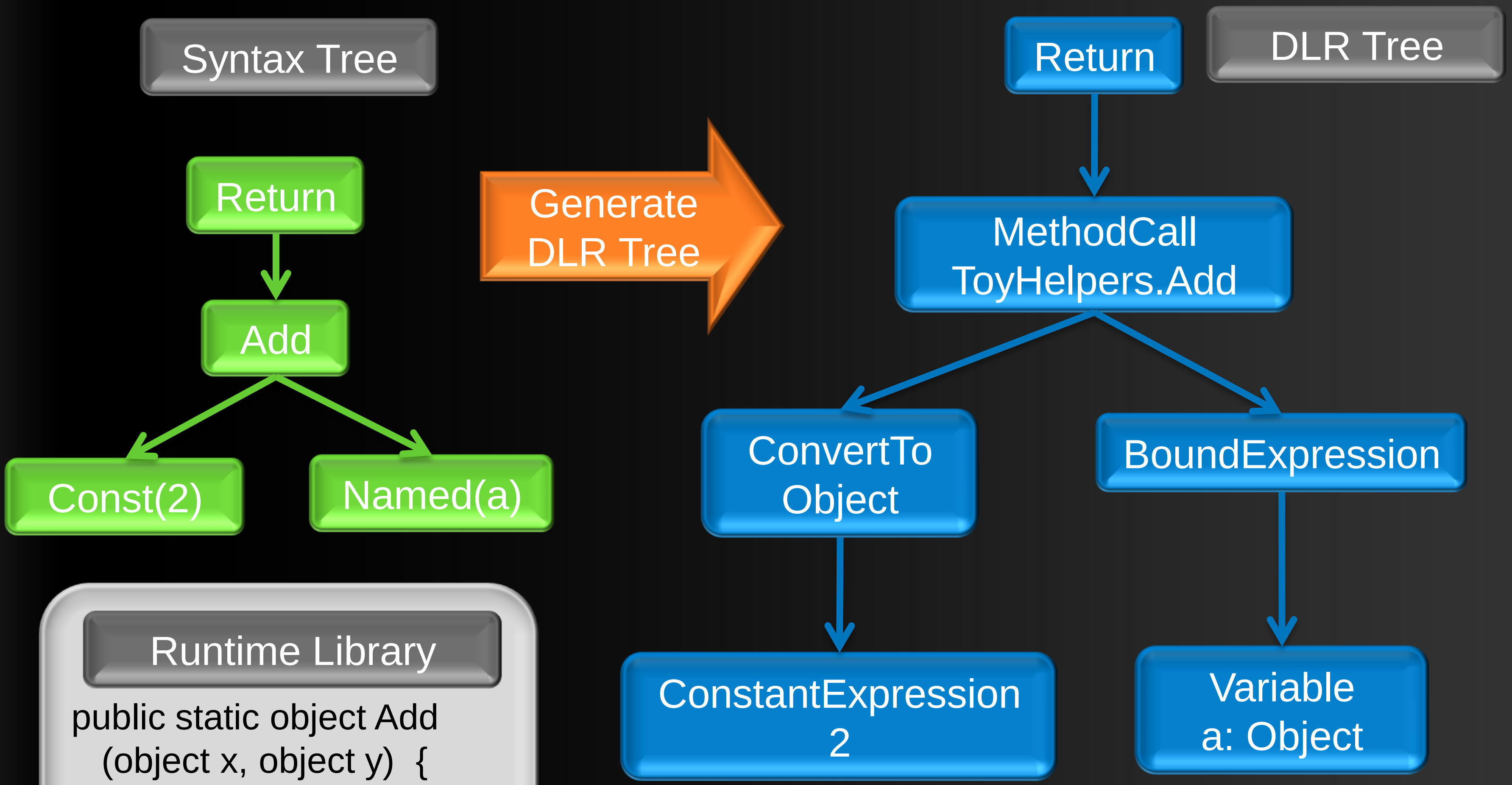


```
ldc.i4.2                    // load 2
box  [mscorlib]System.Int32
ldarg.0                     // load “a”
call  object ToyHelpers::Add(object, object)
ret
```

```
public static object Add
   (object x, object y)  {
     ...
}
```

Compiler Overview

DLR Backend



Runtime Library

```
public static object Add
   (object x, object y)  {
     ...
}
```

Why DLR?

- Focus on your language
  - Scanner
  - Parser
  - Runtime semantics
- DLR
  - Code generation
  - Dynamic operations
  - Extension methods for .NET Type customization
  - Common hosting for all DLR languages

DLR Trees

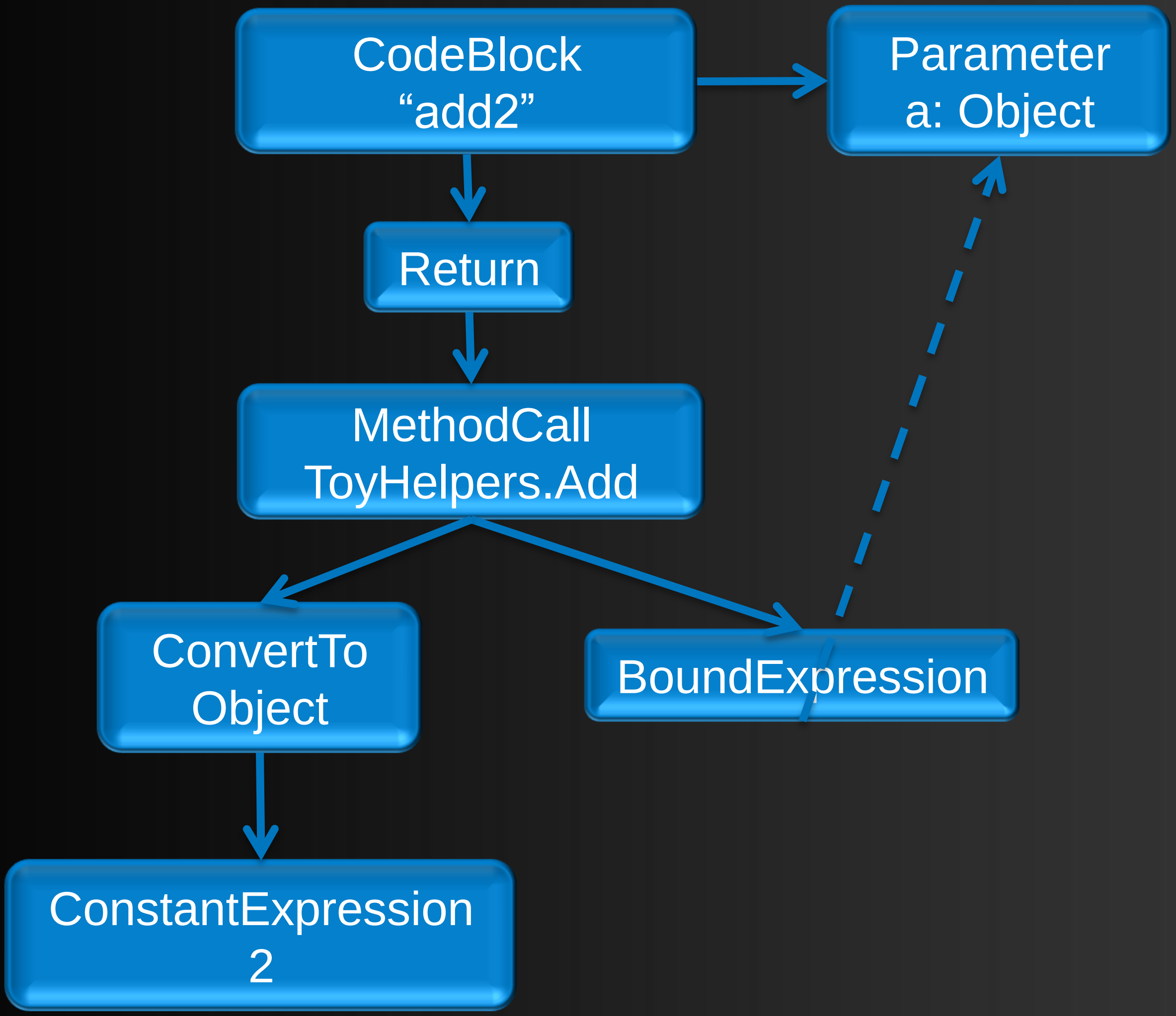
- DLR representation of programs
  - Similar to LINQ expression trees
- Expressions:
  Constant , Unary, Binary, Method call, Property value, Field value, Assignment, …
- Statement support:
  If, While, Try, Return, Switch, Throw, …
- Dynamic behavior support:
  ActionExpression
- Factory methods (Ast.…)

Generating DLR Trees

```
def add2(a) {
    return 2 + a;
}
```

Runtime Library

```
public static object Add
  (object x, object y)  {
   ...
}
```



CodeBlock
“add2”

Parameter
a: Object

Return

MethodCall
ToyHelpers.Add

ConvertTo
Object

BoundExpression

ConstantExpression
2

Generating DLR Trees

```
CodeBlock cb = Ast.CodeBlock("add2");

Variable a = cb.CreateParameter(
   SymbolTable.StringToId( "a" ),
   typeof(object)
);

cb.Body = Ast.Return(
   Ast.Call(
      typeof(ToyHelpers).GetMethod("Add"),
      Ast.Convert(Ast.Constant(2), typeof(object)),
      Ast.Read(a)
   )
);
```

Demo

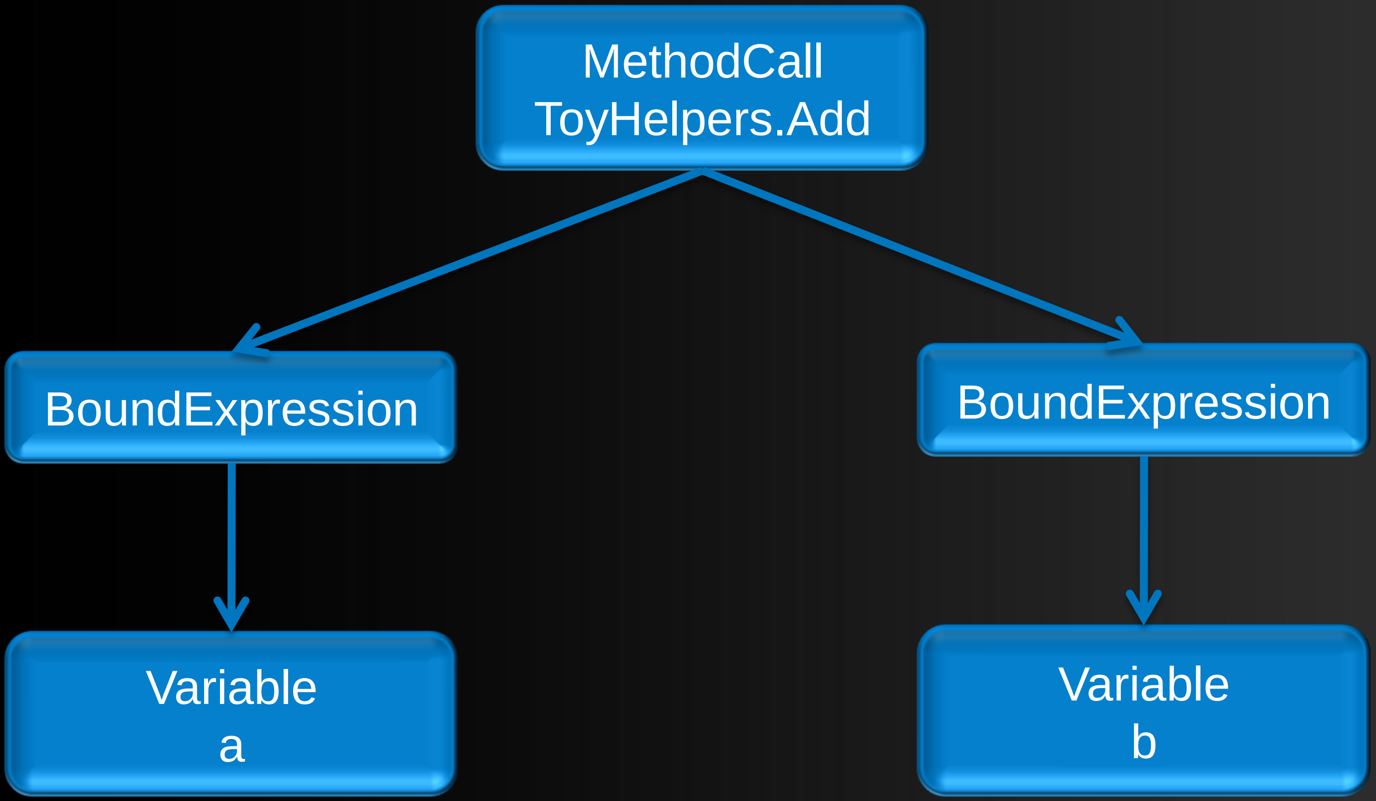
DLR Trees

$a + b$

Operators

Method Call

```
ToyHelpers.Add(a, b)
```



MethodCall
ToyHelpers.Add

BoundExpression

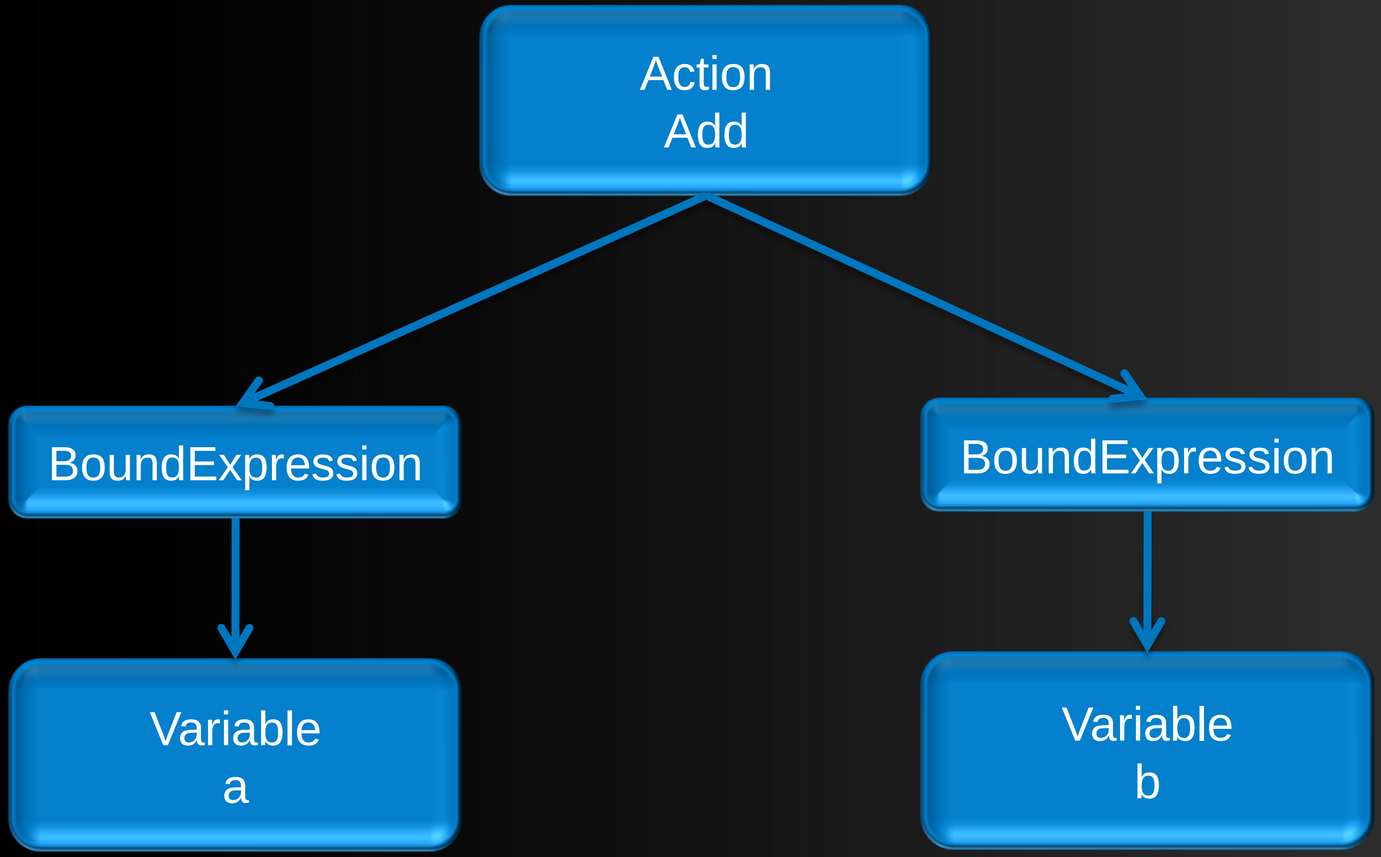
BoundExpression

Variable
a

Variable
b

Operators

Action Expression



Dynamic Actions

Call Site

```
static DynamicSite a_plus_b_site =
       new DynamicSite(Add);

// a + b
a_plus_b_site.Invoke(a, b)

// DynamicSite.Invoke
object Invoke(object a0, object a1) {
   this._handler(this, a0, a1);
}
```

Dynamic Actions

Target Delegate

```
object Handler(DynamicSite s, object a0, object a1)
{








   // HELP !!!
   return s.UpdateMe(a0, a1);
}
```

Dynamic Actions

Updated Target Delegate - Int

```
object Handler(DynamicSite s, object a0, object a1)
{
   if (a0 is int && a1 is int) {
      return (int)a0 + (int)a1;
   }





   // HELP !!!
   return s.UpdateMe(a0, a1);
}
```

Dynamic Actions

Updated Target Delegate - Double

```
object Handler(DynamicSite s, object a0, object a1)
{
   if (a0 is int && a1 is int) {
      return (int)a0 + (int)a1;
   }
   if (a0 is double && a1 is double) {
      return (double)a0 + (double)a1;
   }
   // HELP !!!
   return s.UpdateMe(a0, a1);
}
```

Rule

Rules

- Rule = Test + Target
- Test
  - Condition examining the arguments
- Target
  - An operation to perform if the test succeeds
- Who makes the rules ???
  - The Language
  - The DLR

Rules

Language Action Binder

- DLR requests:
  - “Tell me how to perform this operation with these arguments!”

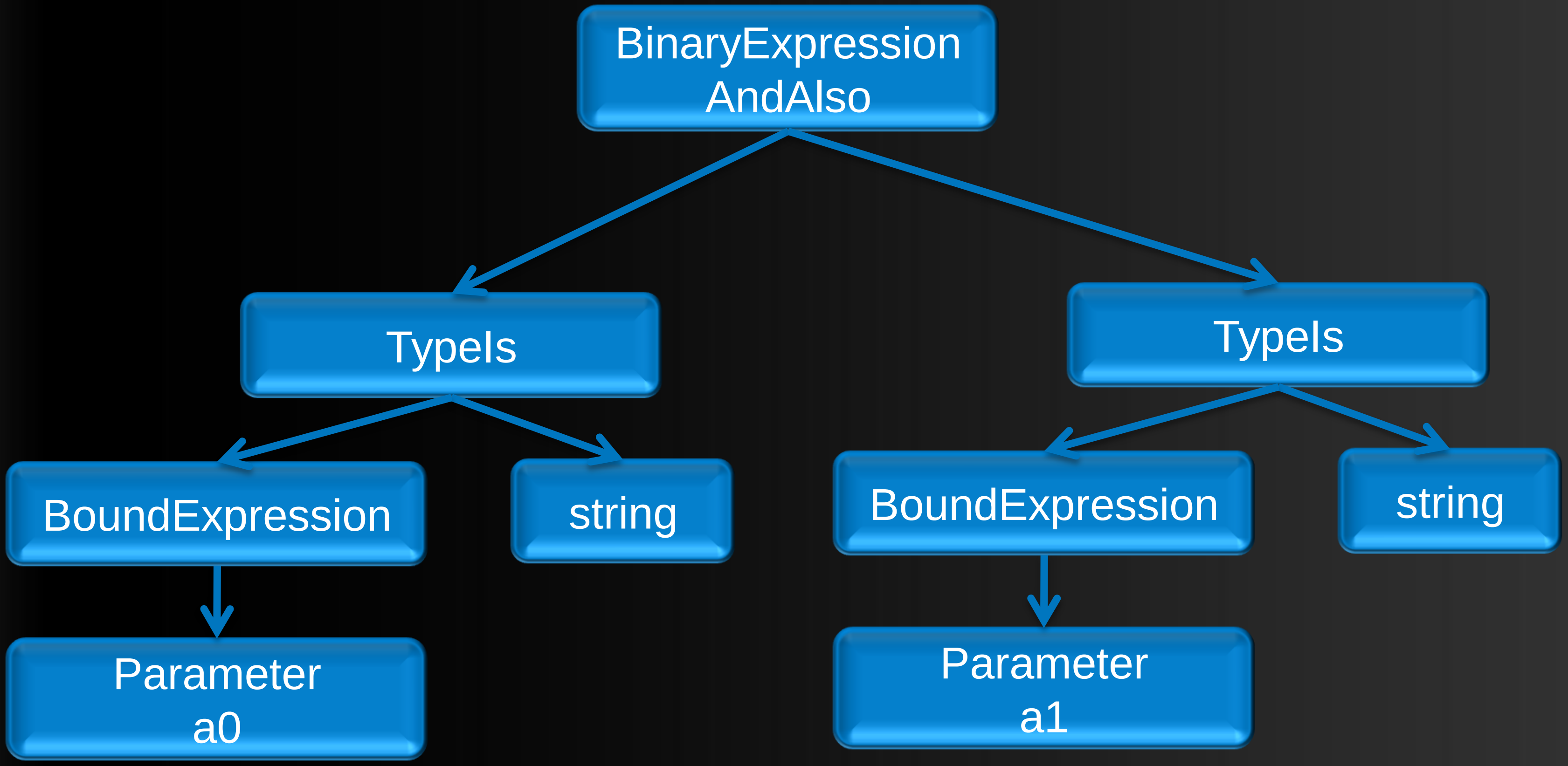
- KEY: “Tell me how!” NOT: “Do it!”

- Language responds:
  - “Here is the Tree”
  - “I don’t know”
    (DLR tries its own built-in behaviors)

Rules

Adding Strings - Test

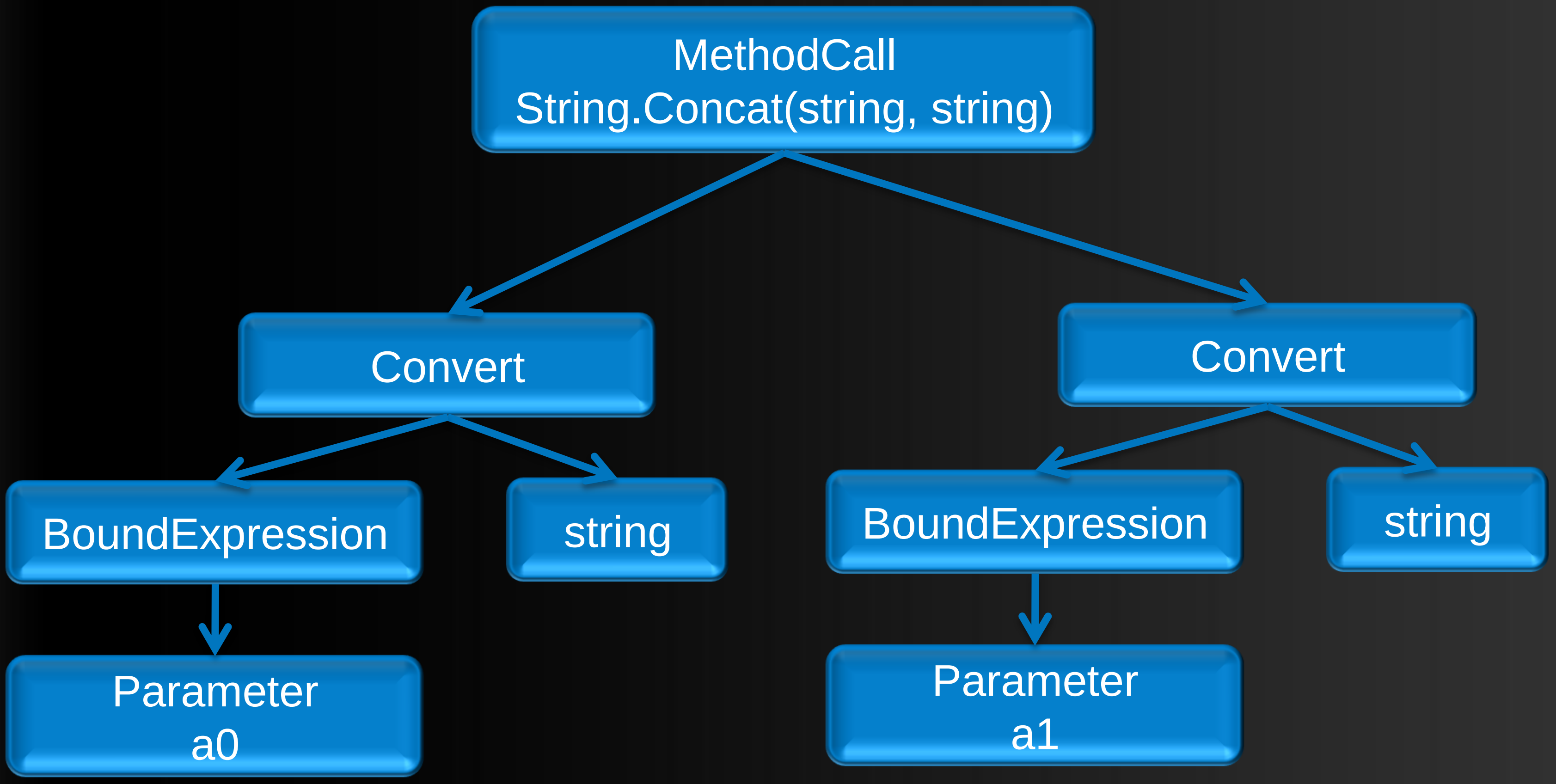
```
(a0 is string) && (a1 is string)
```



Rules

Adding Strings - Target

```
String.Concat((string)a0, (string)a1)
```



Demo

Actions

Targeting the DLR

- Implement scanner and parser
- Translate your AST to the DLR Tree

- Implement your custom types
- Implement customization to .NET types
  - Via extension methods

- Tune performance
  - Runtime library
  - Dynamic types

Next Steps

http://www.codeplex.com/IronPython

http://blogs.msdn.com/mmaly

dlr@microsoft.com

Questions?