# Dynamic in a World of Static

## Dynamic Binding in C# 4.0

Mads Torgersen, C# Language PM

# The Menu

- The Dynamic Language Runtime
- Dynamic in C#
- Demo

# Background

## Common Language Runtime (CLR):

- Common implementation platform for static languages

# Background

## Common Language Runtime (CLR):

- Common implementation platform for static languages

# Background

## Common Language Runtime (CLR):

- Common implementation platform for static languages
- Good interop

# Background

Dynamic Language Runtime (DLR):

- Common implementation platform for dynamic languages
- Good interop

# Background

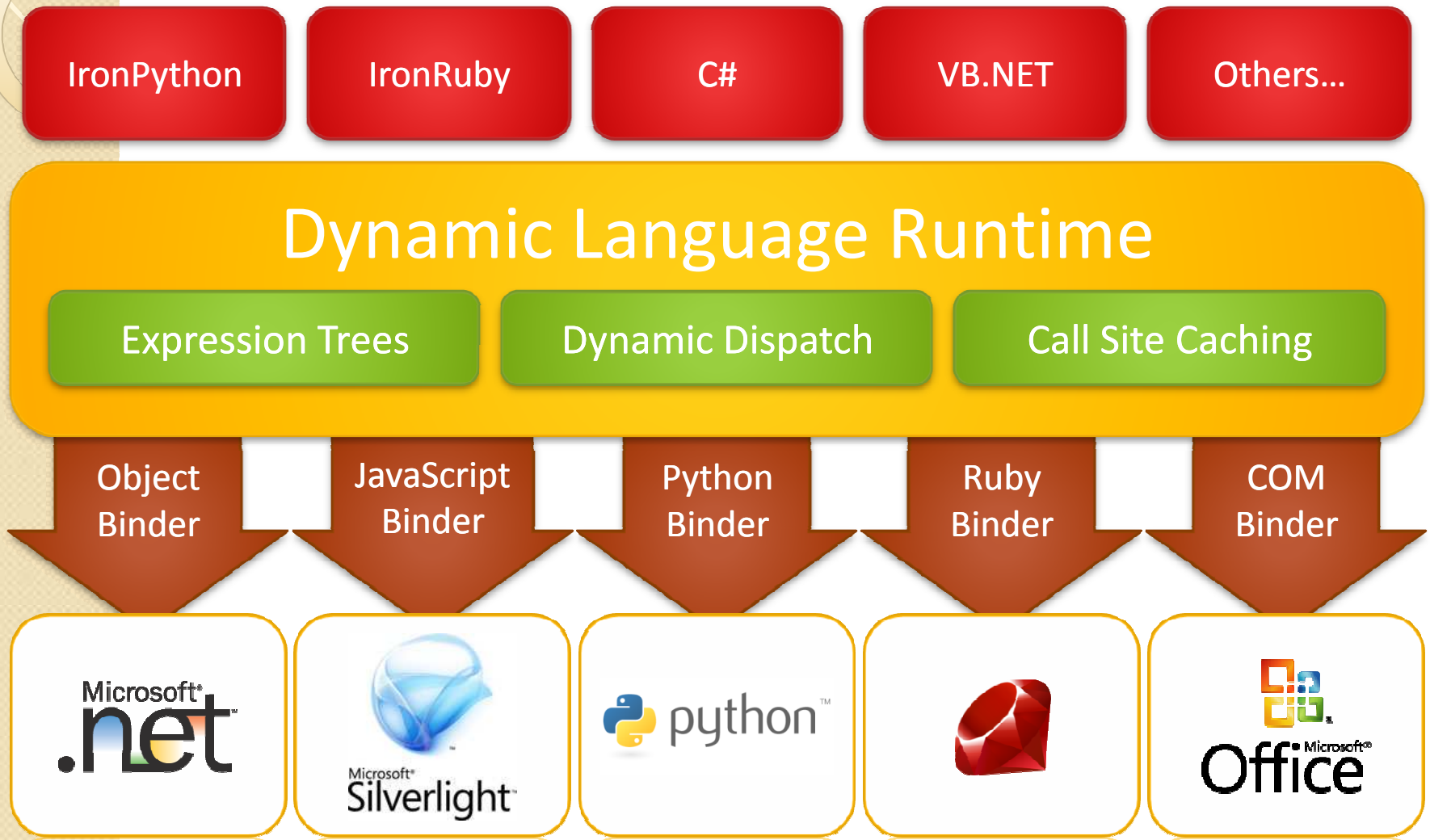
Dynamic Language Runtime (DLR):

- Common implementation platform for dynamic languages
- Good interop
- Enable programmatic dispatch

# Why C# dynamic?

- C# is *not* a dynamic language
  - And will never be
- Embrace dynamic world
  - Build on DLR opportunity
  - Use code from dynamic languages
  - Use other dynamic object models
  - Better COM interop

# Dynamic Language Runtime



# Terminology: Dynamic Binding

- *Binding*:
  Determining the meaning of an operation based on the type of constituents
- *Static binding*:
  Binding is based on compile time (static) types of expressions
- *Dynamic binding*:
  Binding is based on runtime (actual) types of objects

# Binding not typing

- Dynamic *binding* enables programmatic interop
  - Connect different worlds by mapping *actions*, not types
- Dynamic *typing* is a consequence, not a feature

# Syntax

- Knee-jerk: It's got to look different!
  - Safety first
- Secret dream: It's got to look similar!
  - Comfort first

# Syntax

- Explicitly dynamic operations: ☹

```
object d = GetDynamicObject(…);
string result = ~(string)d~[d~.Length ~- 1];
```

- Ugh…

# Syntax

- Dynamic contexts:

```
object d = GetDynamicObject(…);
string result = dynamic((string)d[d.Length - 1]);
```

- *Everything* is dynamic inside
  - A `static` context as well to opt out with?
  - A whole different dialect of C# inside
  - You lose sight of big contexts

# Syntax

- Contagious bit on expressions: ☹

```
object d = GetDynamicObject(…);
string result = (string)d[dynamic(d).Length - 1];
```

- *Something* is dynamic – but what?
  - Rules of propagation?
  - factoring out subexpressions?

# Syntax

- Dynamic type:

```
dynamic d = GetDynamicObject(…);
string result = d[d.Length - 1];
```

- Pro: there's no difference!
  - Easy to see what the code does
- Con: There's no difference!
  - No local indication that it is dynamic

# Why is this OK?

- "Safety" about throwing exceptions
  - Member access already throws exceptions
- You already need types to guess meaning
- This is for making unsafe, error prone, bloated code less so

# Dynamically Typed Objects

```
Calculator calc = GetCalculator();
int sum = calc.Add(10, 20);
```

```
object calc = GetCalculator();
Type calcType = calc.GetType();
object res = calcType.InvokeMember("Add",
    BindingFlags.InvokeMethod, null,
    new object[] { 10,
int sum = Convert.ToIn
```

```
ScriptObject calc = GetCalculator();
object res = calc.Invoke("Add", 10, 20);
int sum = Convert.ToInt32(res);
```

*Statically* typed to be dynamic

```
dynamic calc = GetCalculator();
int sum = calc.Add(10, 20);
```

Dynamic conversion

Dynamic method invocation

# Type or Type Modifier?

- Generality:

```
dynamic Foo d = GetDynamicFoo(…);
```

  - Static binding of Foo's members
  - Dynamic binding of the rest

- Simplicity:

```
dynamic d = GetDynamicFoo(…);
```

  - Dynamic binding of all members
  - Even those on `Object`

# Dynamic binding when?

- When the receiver is dynamic?
  - What to do when arguments are dynamic?

```
dynamic result = Math.Abs((double)d);
```

  - Forces you to choose a type

- When *any* constituent expression is dynamic!

```
dynamic result = Math.Abs(d);
```

# Result type

- Dynamic type:
  - Method call `Math.Abs(d)`
  - Invocation `d("Hello")`
  - Member access `d.Length`
  - Operator application `4 + d`
  - Indexing `d["Hello"]`
- Static type:
  - Conversions `(double)d`
  - Object creation `new Foo(d)`

# Runtime Binding

- C# runtime binder
  - Handles binding of "ordinary objects"
  - Mimics compile time semantics
- IDynamicMetaObjectProvider
  - Implemented by dynamic objects
  - Handle their own binding

# Runtime Binding Semantics

- Constituents typed dynamic:
  - Use their runtime type
- Statically typed constituents:
  - Use their static type
  - Use other static info like literalness
- Principle of least surprise:
  - How dynamic do you want to be today?

`dynamic` means

"use my runtime type for binding"

# Demo…

?