Dynamic **L**anguage **R**untime

Dynamic in a World of Static

Why dynamic?

- There is a lot of interest in accessing the highly dynamic object model of HTML DOM
- COM is heavily used and the inter-op code could be easier to read and write
- Dynamic languages are becoming increasingly popular

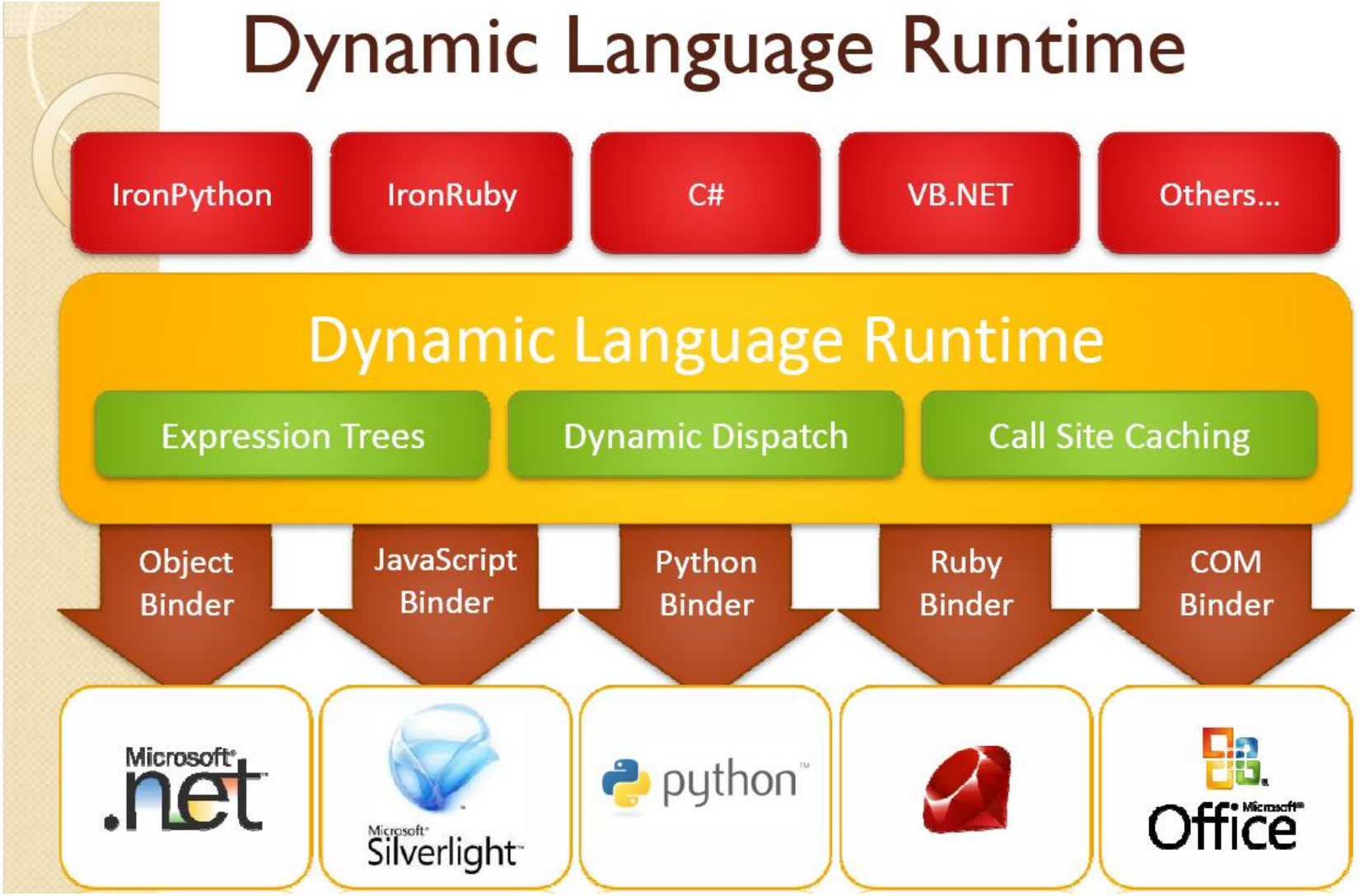
- We need a unified way to work with all of the above

Dynamic in a World of Static

- Common Language Runtime (CLR)
  - Common implementation platform for static languages
- Dynamic Language Runtime (DLR)
  - Common implementation platform for dynamic languages

- Why C# dynamic?
  - Embrace dynamic world
    - Build on DLR opportunity
    - Use code from dynamic languages
    - Use other dynamic object models
    - **Better COM interop**

Dynamic Language Runtime



Terminology

- *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

DLR adds a set of services to the CLR

- Expression trees
  - By this we can express language semantics in form of AST (Abstract syntax tree).
  - The DLR dynamically generates code using the AST which can be executed by the CLR.
- Call site caching
  - When you make method calls to dynamic objects the DLR caches information about those method calls (the binding).
  - For the other subsequent calls to the method the DLR uses the cache history information for fast dispatch.
- Dynamic object interoperability
  - The DLR provides a set of classes and interfaces that represent dynamic objects and operations.
  - E.g.
    - DynamicObject
    - ExpandoObject.

C# Syntax

- Statically typed to be dynamic ☺

```
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

- Dynamic means
  "use my runtime type for binding"

Dynamic = Easy Code

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

C#
When type is known

```
object calc = GetCalculator();
Type calcType = calc.GetType();
object res = calcType.InvokeMember("Add",
         BindingFlags.InvokeMethod, null, calc
         new object[] { 10, 20 });
int sum = Convert.ToInt(res);
```

C# 3.0
When type is unknown

C# 4.0
When type is unknown

Statically typed to be dynamic

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

Dynamic conversion

Dynamic method invocation

Performance?

- Dynamic is slow compared to static typing
  - Don’t use Dynamic if static typing is available
  - But faster then use of reflection
    - Due to call site catching
- If Dynamic is the only option,
  - Then performance is not really the issue

Dynamic COM Interop

- We can use dynamic to access **COM** objects:

```
Type fwMgrType = Type.GetTypeFromProgID("HNetCfg.FwMgr");
dynamic fwMgr = Activator.CreateInstance(fwMgrType);

bool isFwEnabled = fwMgr.LocalPolicy.CurrentProfile.FirewallEnabled;
```

```
dynamic comInterop =
        Activator.CreateInstance(Type.GetTypeFromProgID("MyCOM.Object.Name"));
var result = comInterop.MethodCall(parameter);
```