Practical Reflection

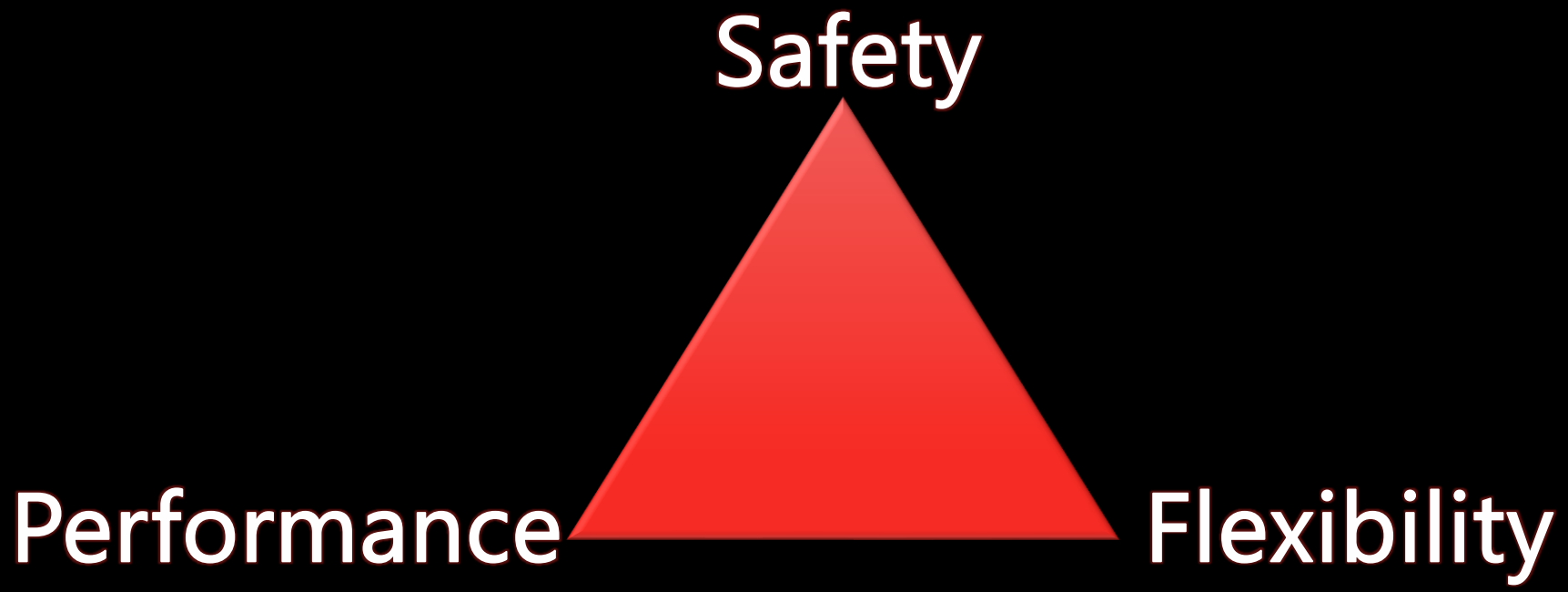
Using reflection in .NET while still keeping your sanity

Jeremy Clark
www.jeremybytes.com
@jeremybytes

Just for Experts?

Goal

Explore the Practical Parts of Reflection

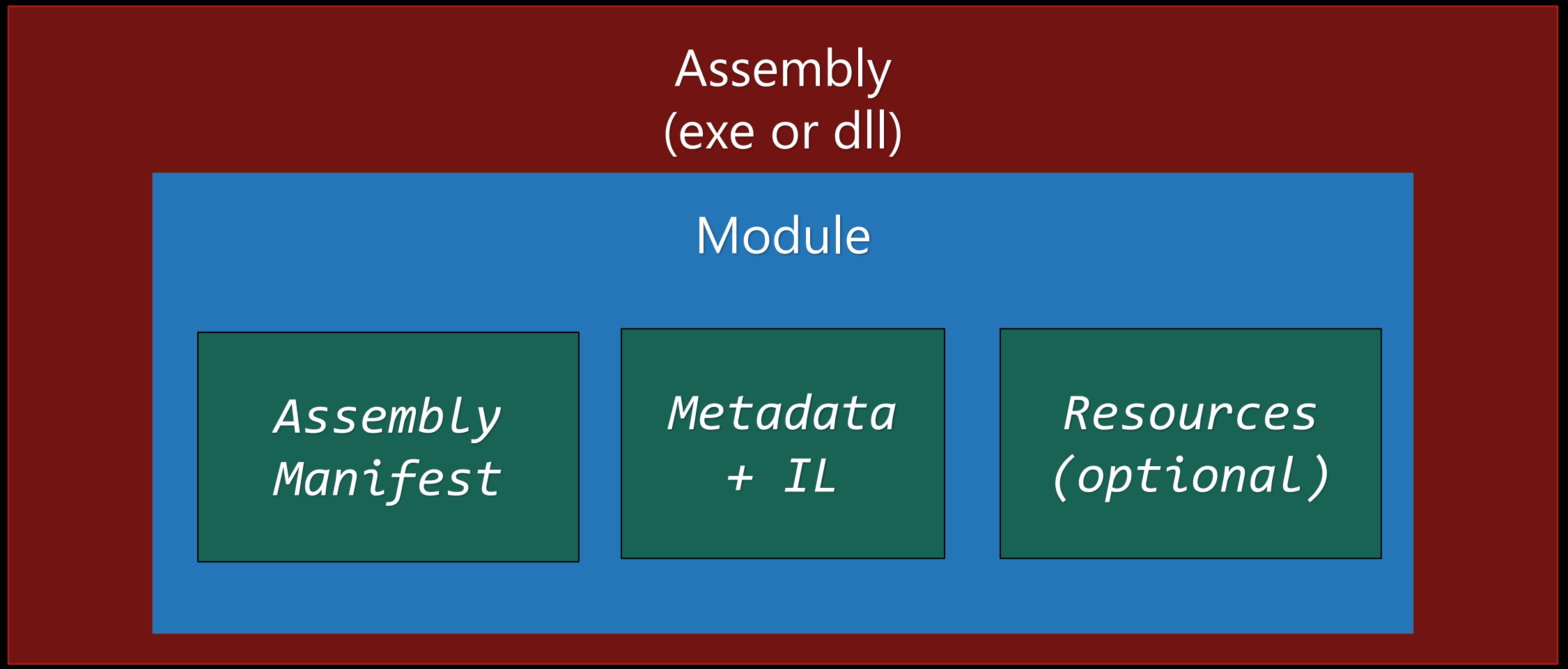


What is Reflection?

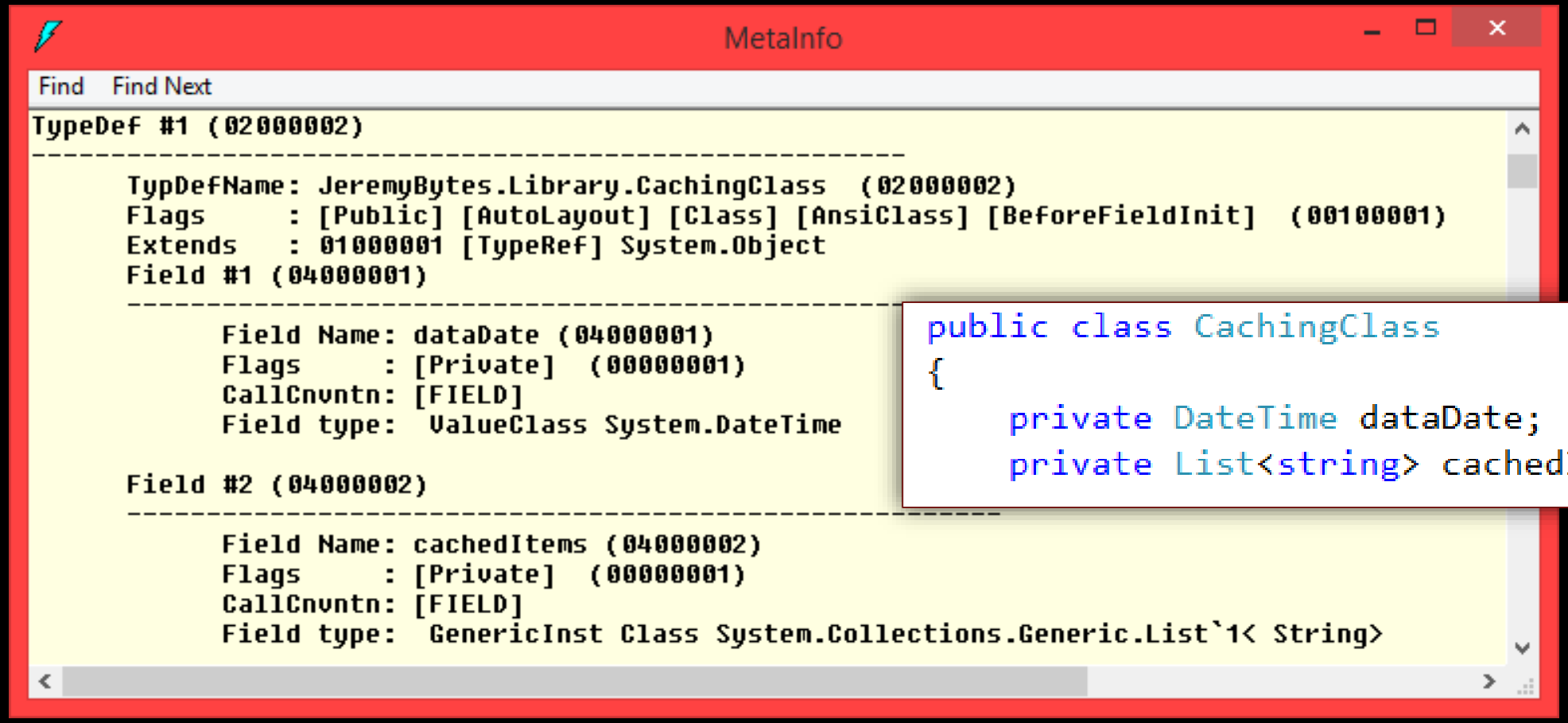
`Inspecting the metadata and compiled code in an assembly.`

- What is an assembly?
- What is metadata?
- How is the code compiled?

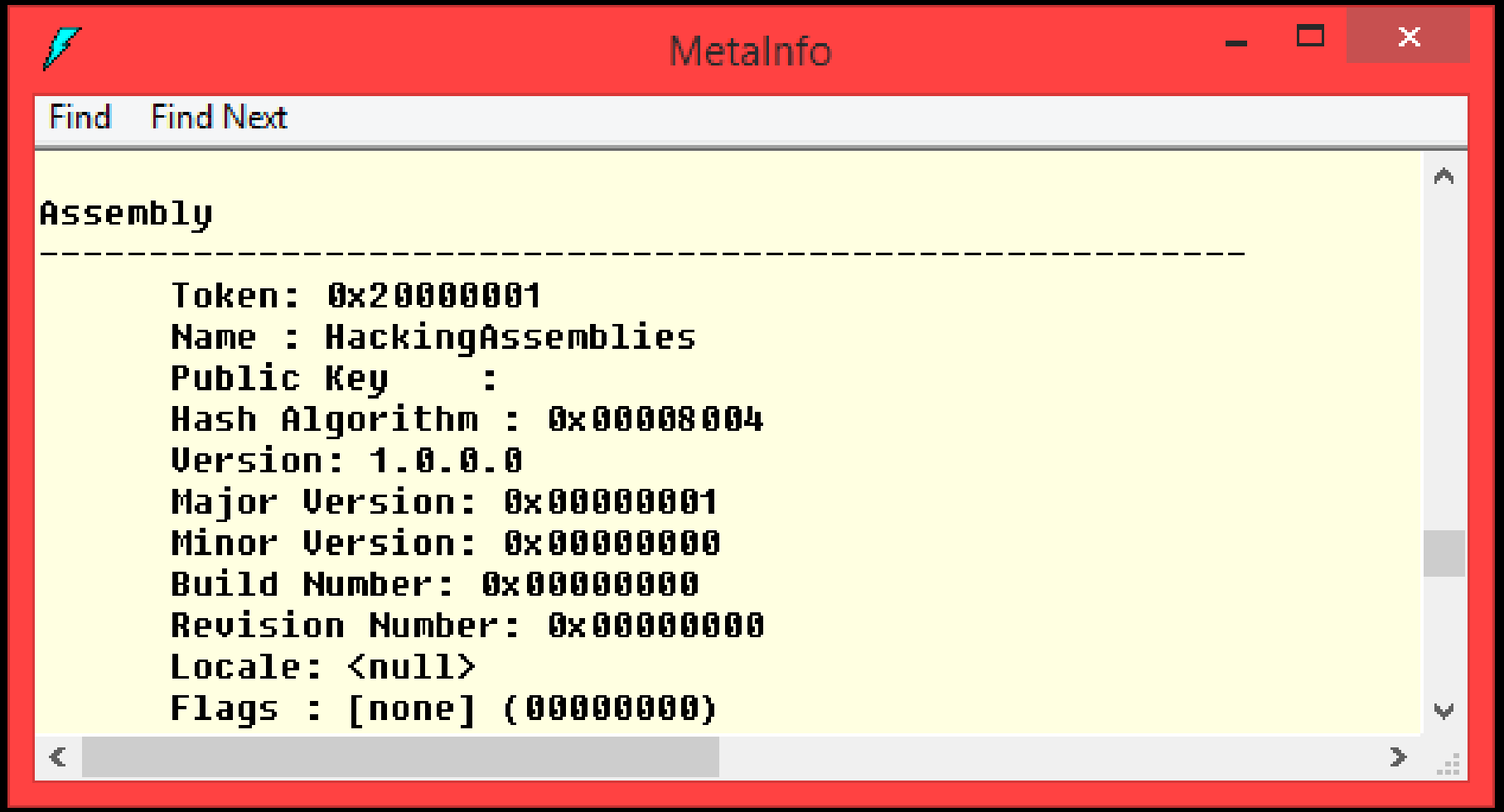
.NET Assemblies



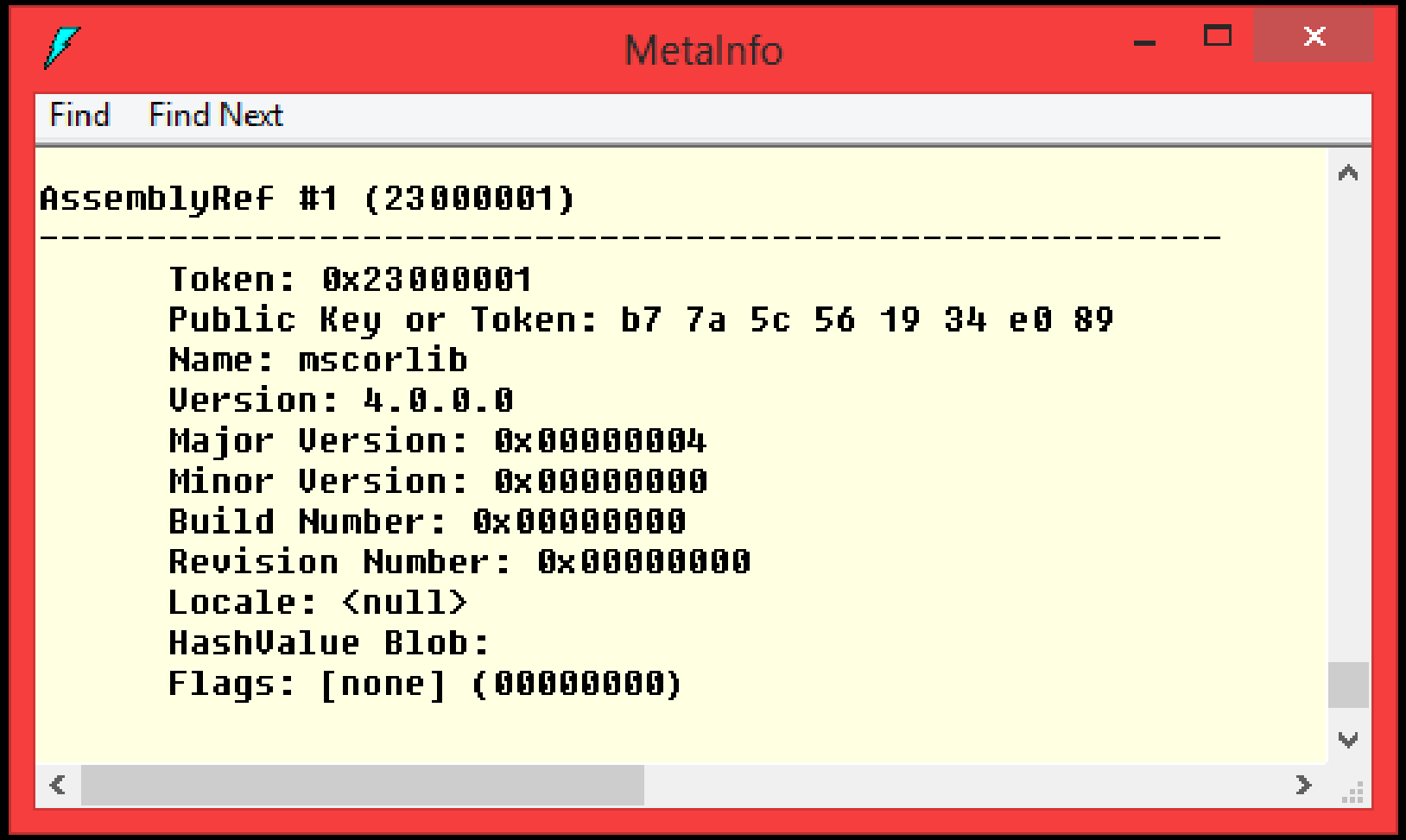
Type Definitions



Assembly Information

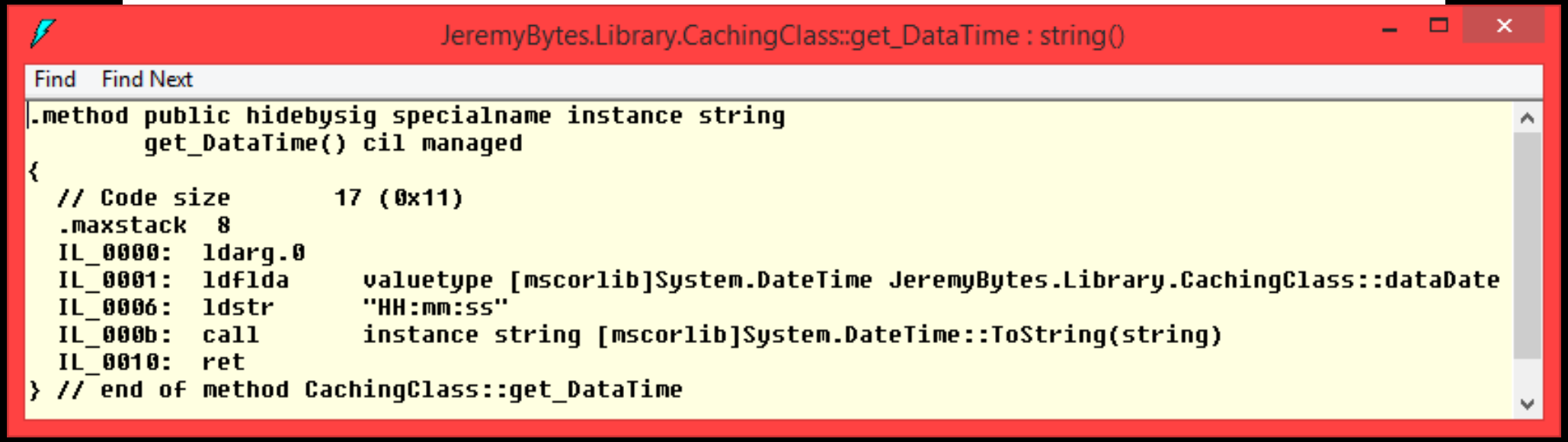


Referenced Assemblies



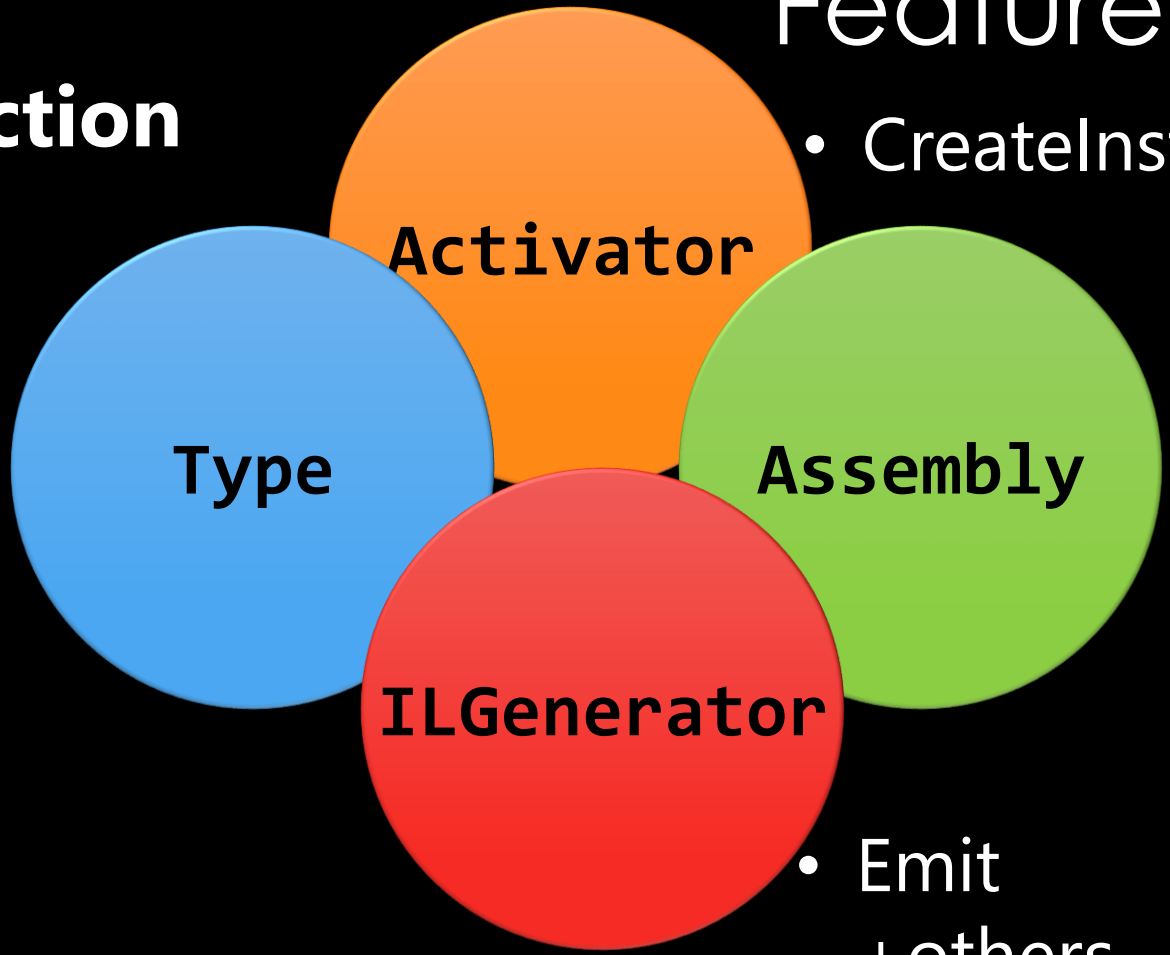
IL (Intermediate Language)

```
public string DataTime
{
    get { return dataDate.ToString("HH:mm:ss"); }
}
```



Feature Overview

System.Reflection



- GetType
- GetMember
- GetMethod
- GetProperty
- GetField

  + many more

- CreateInstance

- Load
- LoadFrom
- GetTypes
- GetName
- GetFiles

  + many more

- Emit

  +others

Things You Can Do

- Reflecting on a Property

```
Type sampleType = typeof(SampleClass);
PropertyInfo cacheProperty = sampleType.GetProperty("CachedItems");
List<string> cacheValue = cacheProperty.GetValue(privateSample)
                          as List<string>;
```

- Useful for interacting with COM objects (pre-.NET 4.0)
- "dynamic" is a better choice for interacting with COM

Things You Can Do

- Reflecting on a Method

```
var list = new List<int>();
Type listType = typeof(List<int>);
Type[] parameterTypes = { typeof(int) };
MethodInfo addMethod = listType.GetMethod("Add", parameterTypes);
addMethod.Invoke(list, new object[] { 7 });
```

- Useful for interacting with COM objects (pre-.NET 4.0)
- "dynamic" is a better choice for interacting with COM

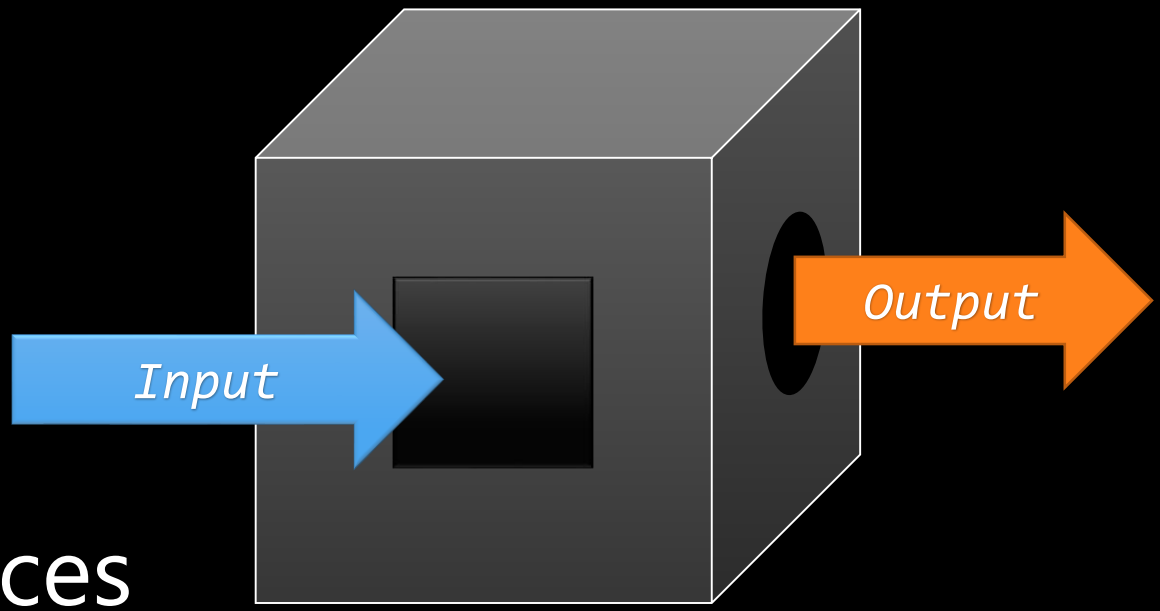
Things You Can Do

- Reflecting on a Private Field

```
Type sampleType = typeof(SampleClass);
FieldInfo cacheField = sampleType.GetField("internalCache",
                       BindingFlags.NonPublic | BindingFlags.Instance);
List<string> cacheValue = cacheField.GetValue(privateSample)
                       as List<string>;
```

- BindingFlags give us access to non-public members
- DANGER DANGER DANGER

Encapsulation



- Use the exposed interfaces
- Don't peek inside the box

Demo

Performance Concerns

Best Practice

Program to an abstraction rather than a concrete type

Practical Reflection Strategy

- **Dynamically Load Assemblies**
  - Happens one time (at start up)
- **Dynamically Load Types**
  - Happens one time (at start up)
- **Cast Types to a Known Interface**
  - All method calls go through the interface
  - No dynamic method calls – no MethodInfo.Invoke
  - Avoid interacting with private members

Various Data Sources

Microsoft SQL Server

MongoDB

CSV

REST Service

WebAPI

Oracle

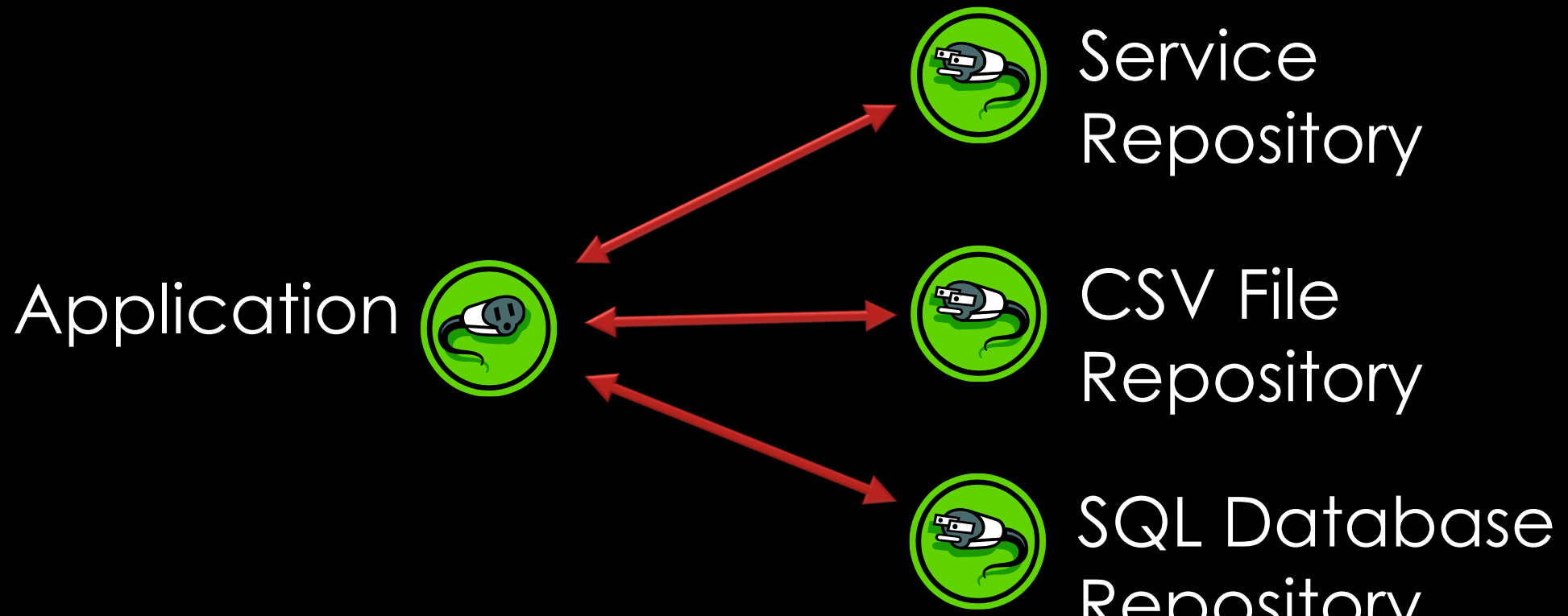
Amazon AWS

JSON

Hadoop

Microsoft Azure

Pluggable Repositories



Demo

Run-Time Binding

Benefits of Dynamic Loading

- Only ship 1 repository assembly
- Remove dependency on concrete repositories
- New repositories can be added without modifying exiting code

Assembly-Qualified Type Name

```
PersonRepository.SQL.SQLRepository,
PersonRepository.SQL,
Version=1.0.0.0,
Culture=neutral,
PublicKeyToken=b77a5c561934e089
```

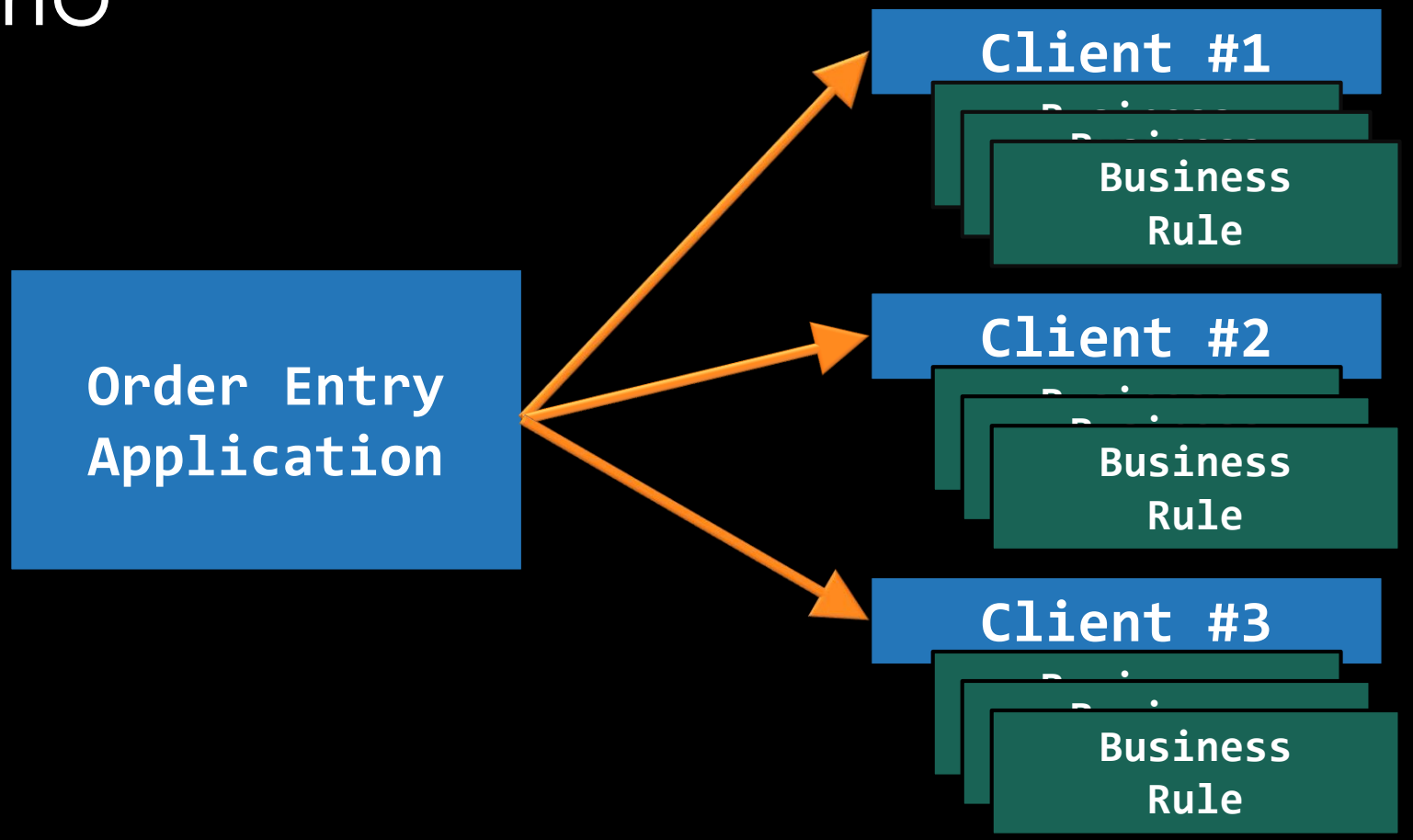
- Fully-qualified type name (namespace and type)
- Assembly Name
- Assembly Version
- Assembly Culture
- Assembly Public Key (for strongly-named assemblies)

Limiting Reflection

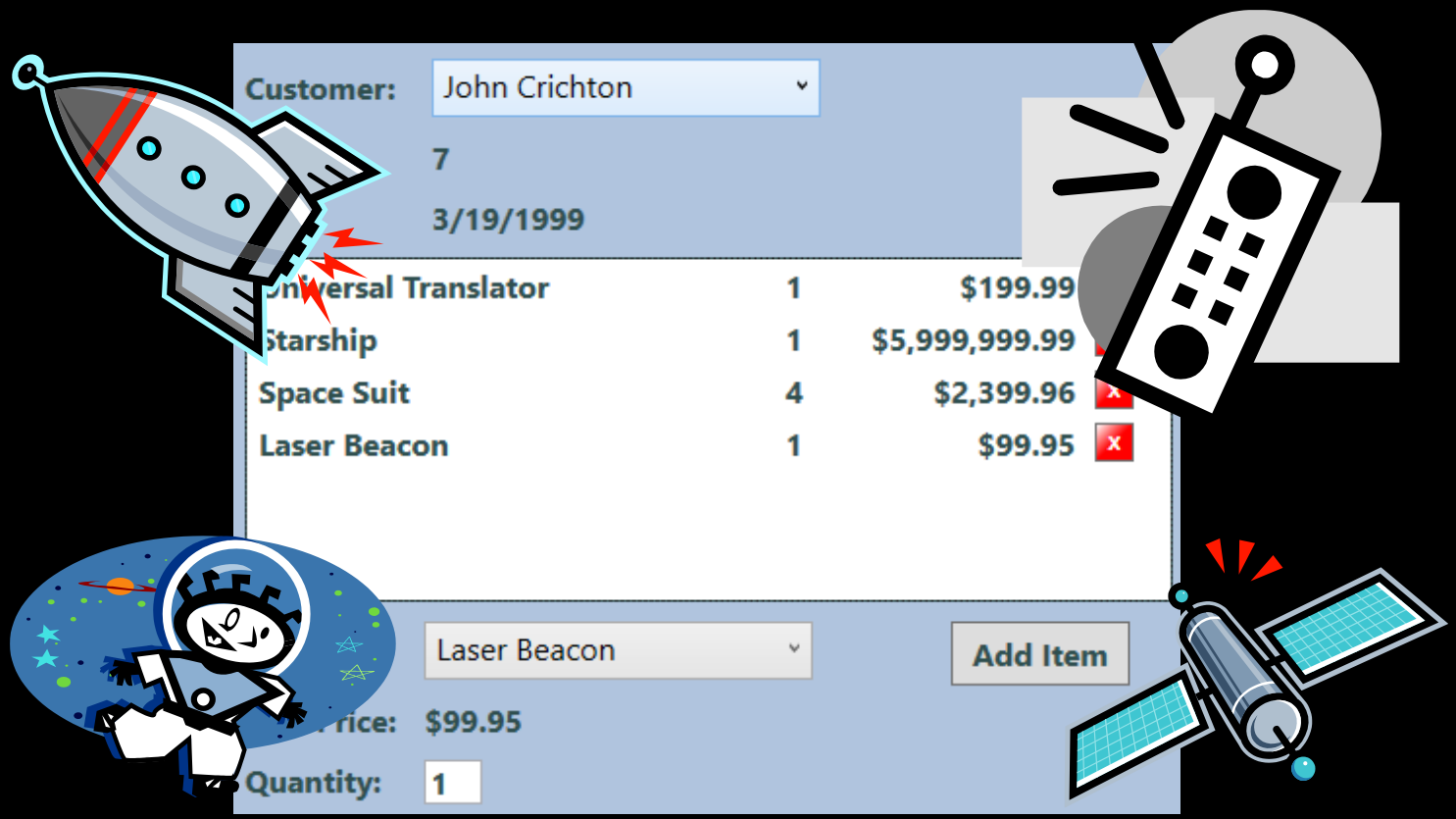
```
private void FetchButton_Click(object sender, EventArgs e)
{
    ClearListBox();

    var people = repository.GetPeople();
    foreach (var person in people)
        PersonListBox.Items.Add(person);

    ShowRepositoryType(repository);
}
```

- No Reflection Here
- Method calls through IPersonRepository

Scenario



Application



Business Rule Interface

```
public interface IOrderRule
{
    string RuleName { get; }
    OrderRuleResult CheckRule(Order order);
}

public class OrderRuleResult
{
    public bool Result { get; set; }
    public string Message { get; set; }

    public OrderRuleResult(bool result,
                           string message) {...}
}
```

Business Rules

Maximum Discount based on Customer Rating

Only 1 Captain's Chair Allowed

Maximum of 1 Starship per Order

Name Badge must match Customer Name

Discovery Process

- Locate all assemblies in the “Rules” folder
- Load each assembly
- Enumerate the types in the assembly
- Check each type to see if it implements our Rule interface
- Create an instance of each Rule and add it to the Rule Catalog

Thank You!

Jeremy Clark

- http://www.jeremybytes.com
- jeremy@jeremybytes.com
- @jeremybytes