

Practical Reflection

Using reflection in .NET while still keeping your sanity

Jeremy Clark

jeremybytes.com

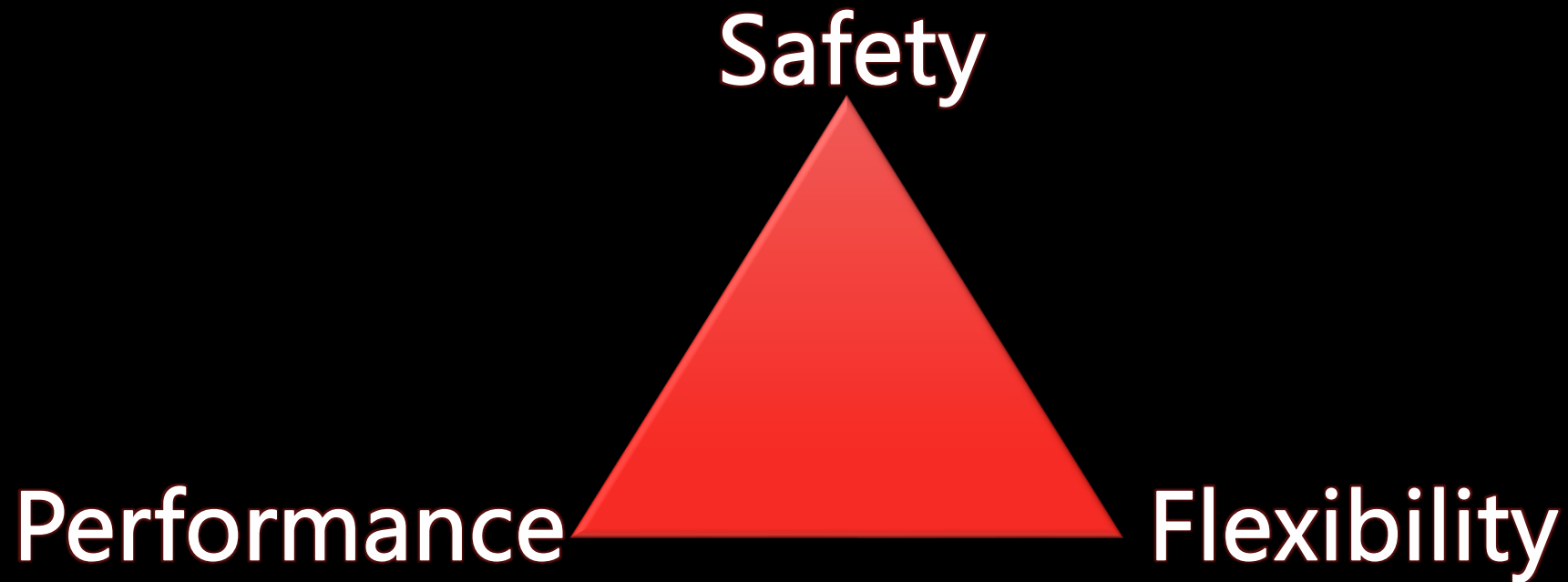
github.com/jeremybytes/sdd-2024

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

Assembly
(exe or dll)

Module

*Assembly
Manifest*

*Metadata
+ IL*

*Resources
(optional)*

Type Definitions

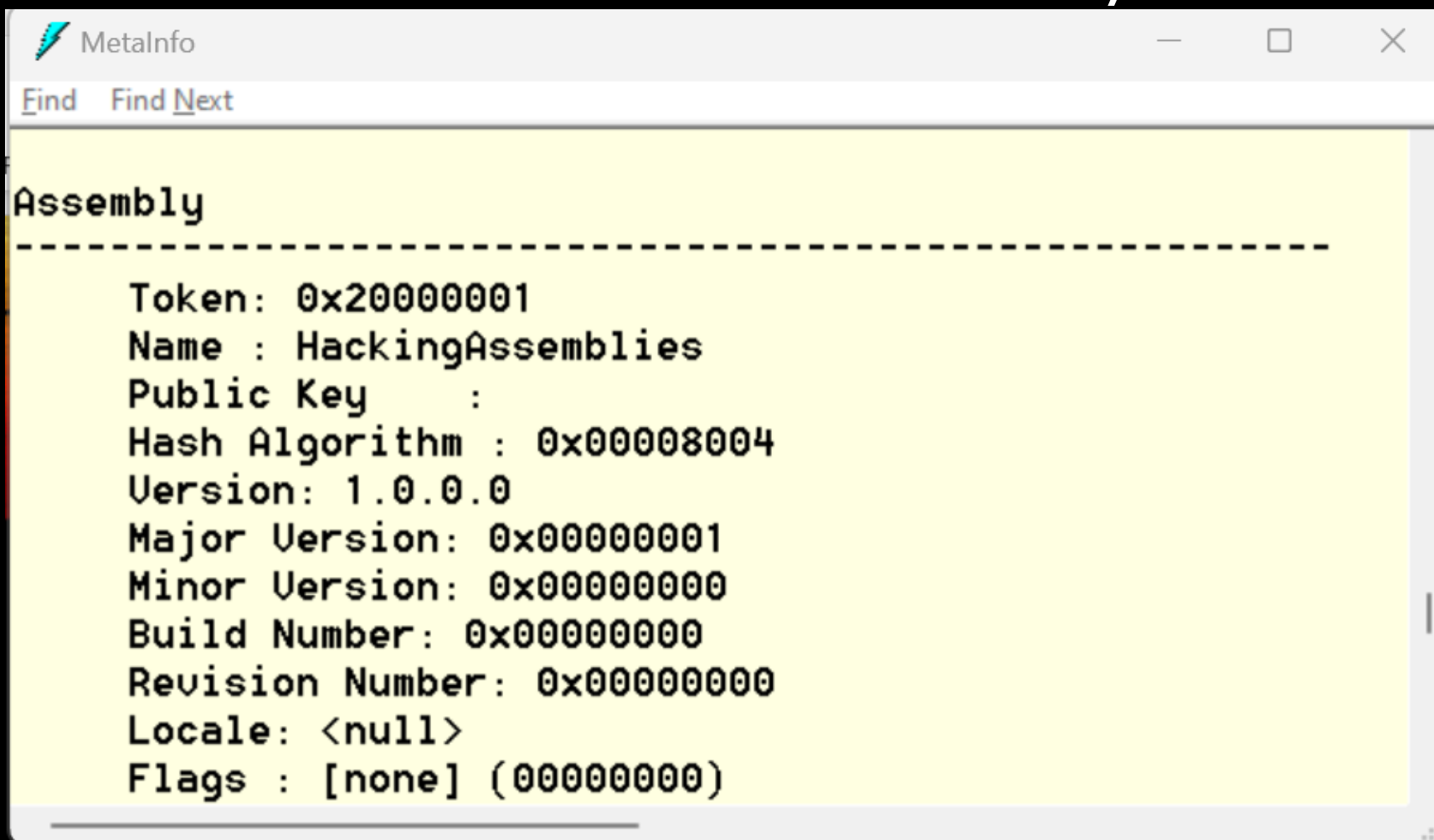
```
MetaInfo
Find Find Next

TypeDef #1 (02000002)
-----
  TypDefName: HackingAssemblies.CachingClass (02000002)
  Flags      : [Public] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit] (00100001)
  Extends    : 0100000F [TypeRef] System.Object
  Field #1 (04000001)
  -----
    Field Name: dataDate (04000001)
    Flags      : [Private] (00000001)
    CallCnvnth: [FIELD]
    Field type: ValueClass System.DateTime

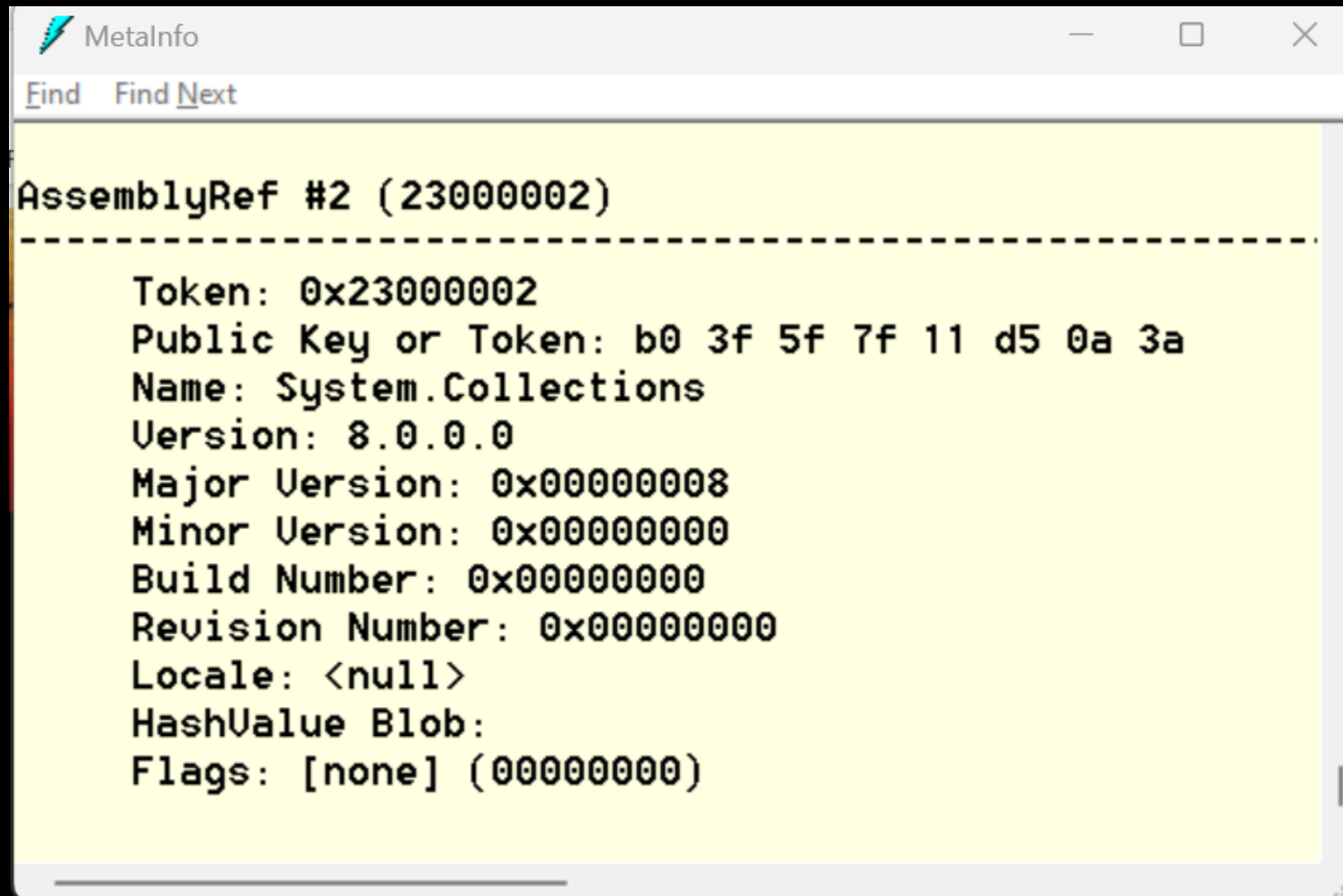
  Field #2 (04000002)
  -----
    Field Name: cachedItems (04000002)
    Flags      : [Private] (00000001)
    CallCnvnth: [FIELD]
    Field type: GenericInst Class System.Collections.Generic.List`1< String>
```

```
public class CachingClass
{
    private DateTime dataDate;
    private List<string> cachedItems;
```


Assembly Information

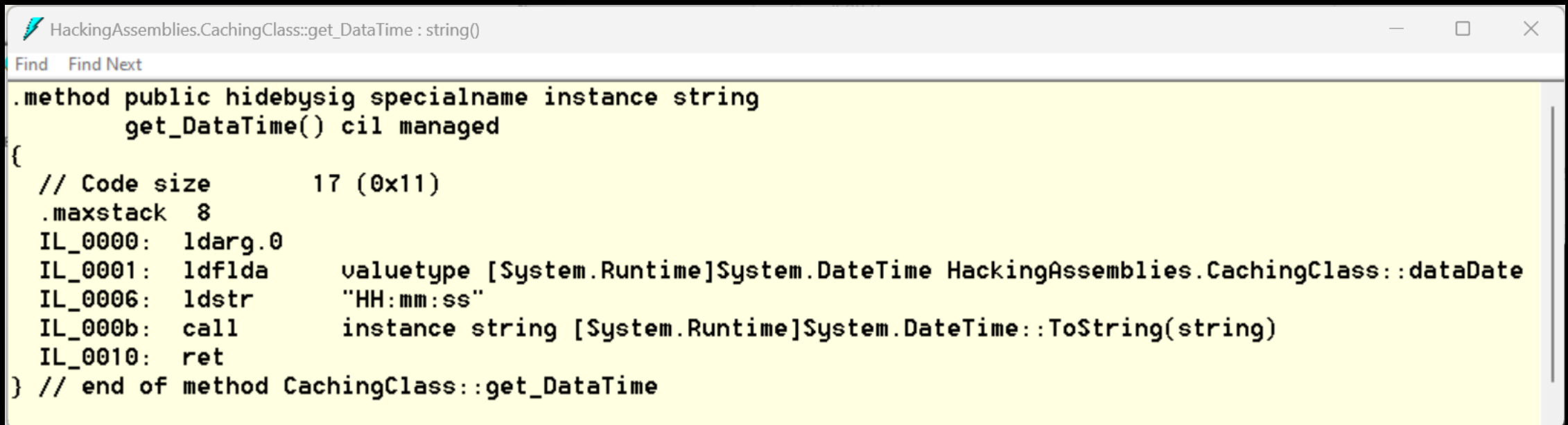


Referenced Assemblies



IL (Intermediate Language)

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



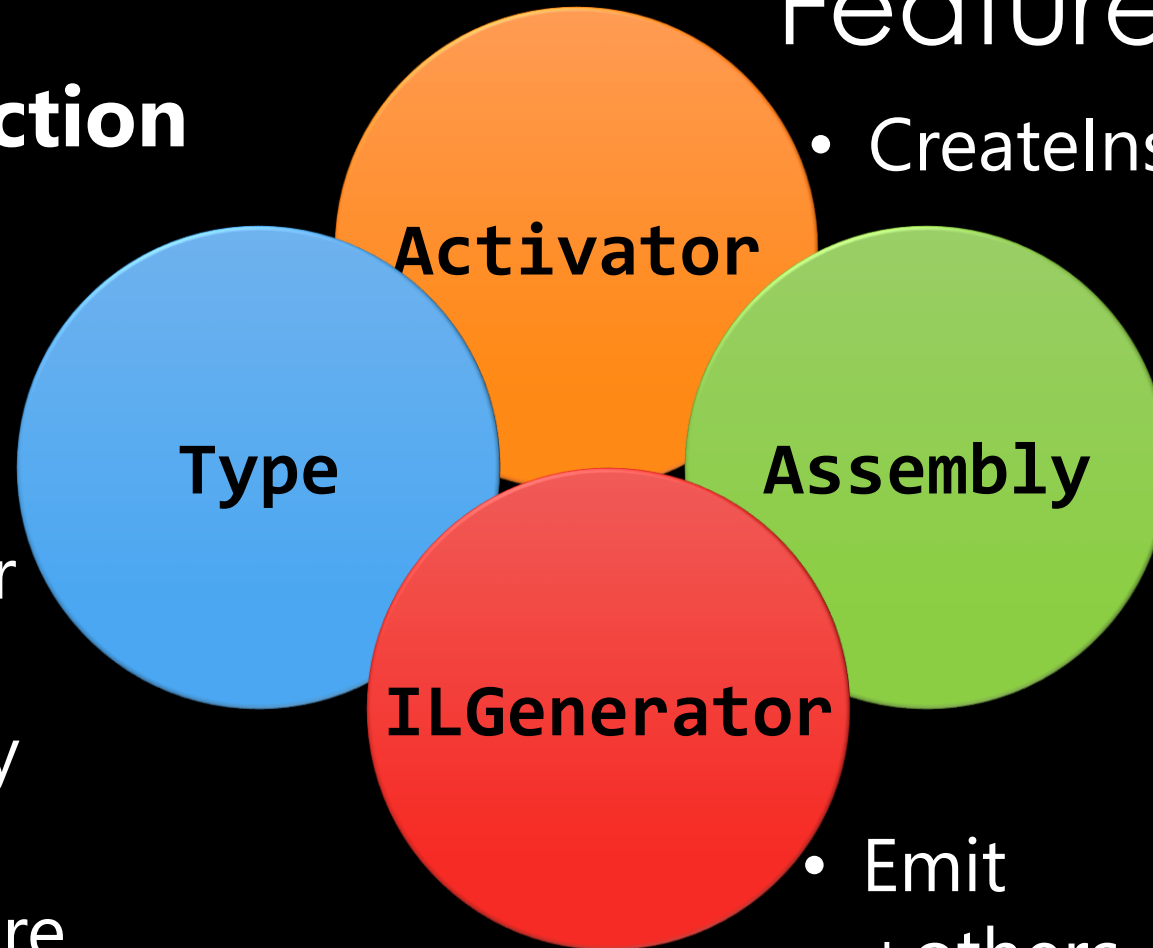
The screenshot shows a debugger window titled "HackingAssemblies.CachingClass::get_DataTime : string()". The window contains a search bar with "Find" and "Find Next" buttons. The main area displays the IL code for the method, which is highlighted in yellow. The code starts with a metadata entry for the method, followed by a block of IL instructions including code size, maxstack, and a sequence of instructions (ldarg.0, ldflda, ldstr, call, ret) that implement the logic of the C# code shown above.

```
.method public hidebysig specialname instance string
    get_DataTime() cil managed
{
    // Code size          17 (0x11)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: ldflda      valuetype [System.Runtime]System.DateTime HackingAssemblies.CachingClass::dataDate
    IL_0006: ldstr      "HH:mm:ss"
    IL_000b: call       instance string [System.Runtime]System.DateTime::ToString(string)
    IL_0010: ret
} // end of method CachingClass::get_DataTime
```

System.Reflection

Feature Overview

- GetType
- GetMember
- GetMethod
- GetProperty
- GetField
- + many more



- CreateInstance

- Emit
- + others

- Load
- LoadFrom
- GetTypes
- GetName
- GetFiles
- + many more

Things You Can Do

- Reflecting on a Property

```
CachingClass safeCacher = new();  
Type cachingType = typeof(CachingClass);  
PropertyInfo? cacheProperty = cachingType.GetProperty("CachedItems");  
List<string> cacheValue = cacheProperty?.GetValue(safeCacher)  
                        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

```
List<int> list = new();  
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

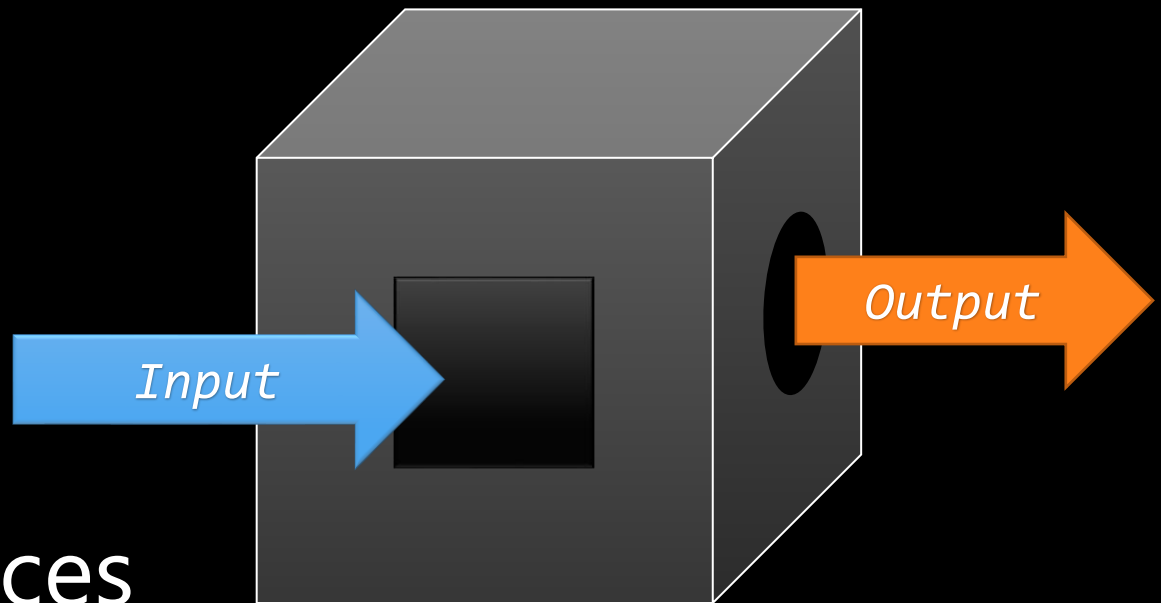
Things You Can Do

- Reflecting on a Private Field

```
CachingClass safeCacher = new();  
Type cachingType = typeof(CachingClass);  
FieldInfo? cacheField = cachingType.GetField("cachedItems",  
                                             BindingFlags.NonPublic | BindingFlags.Instance);  
List<string>? cacheValue = cacheField?.GetValue(dangerousCacher)  
                                             as List<string>;
```

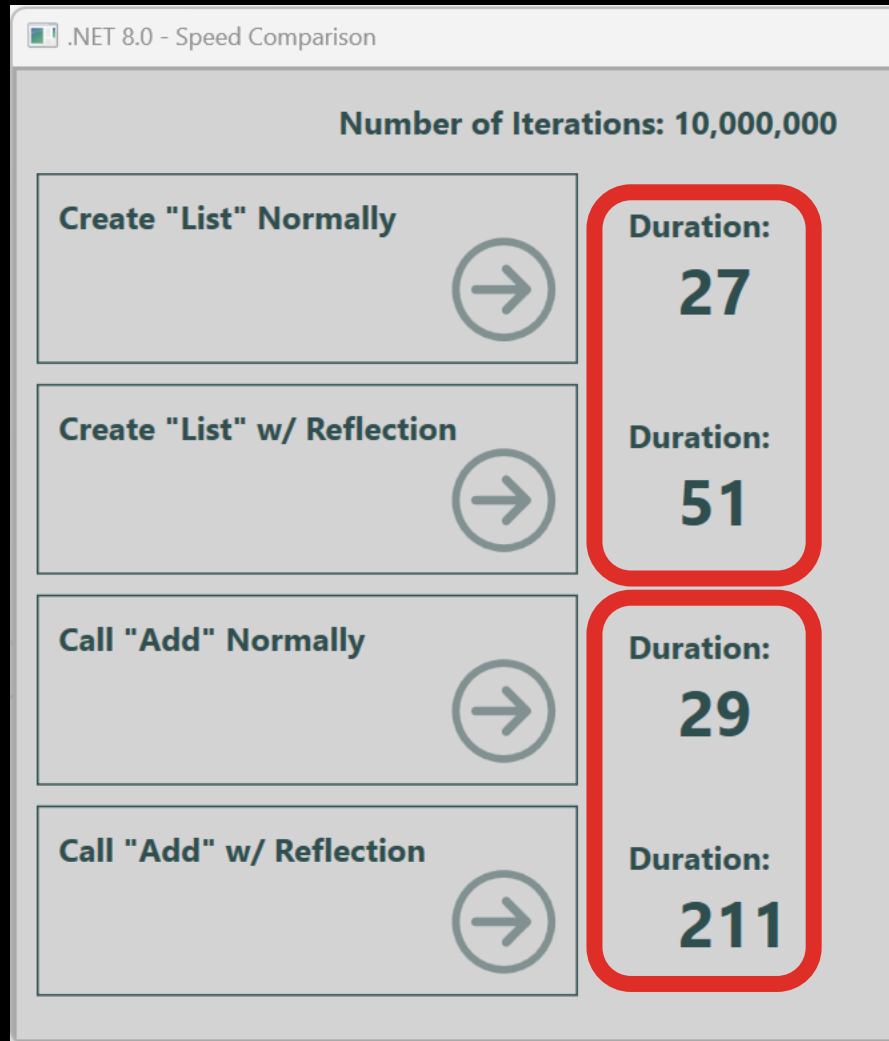
- BindingFlags give us access to non-public members
- DANGER DANGER DANGER

Encapsulation



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

Performance



Reflection 2x slower

Reflection 7x slower

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

Practical Reflection Strategy

- **Sample – Cast to an Interface**

```
private void InterfaceAddButton_Click(object sender, RoutedEventArgs e)
{
    Type listType = typeof(List<int>);
    IList<int>? list = Activator.CreateInstance(listType) as IList<int>;

    list!.Add(7);
}
```

Practical Reflection Strategy





- **Alternate – Create a Delegate**

```
private delegate void ListAddDelegate(List<int> list, int value);

private void DelegateAddButton_Click(object sender, RoutedEventArgs e)
{
    var list = new List<int>();
    Type listType = typeof(List<int>);

    MethodInfo? addMethod = listType.GetMethod("Add");
    var addDelegate = (ListAddDelegate)Delegate.CreateDelegate(
        typeof(ListAddDelegate), addMethod!);
    addDelegate(list, 7);
}
```

Performance

.NET 8.0 - Speed Comparison	
Number of Iterations: 10,000,000	
Call "Add" Normally 	Duration: 29
Call "Add" w/ Reflection 	Duration: 223
Call "Add" w/ Interface 	Duration: 29
Call "Add" w/ Delegate 	Duration: 39

Reflection 7x slower

Interface – no penalty

Delegate – small penalty

Various Data Sources

Microsoft SQL Server

MongoDB

CSV

REST Service

Oracle

WebAPI

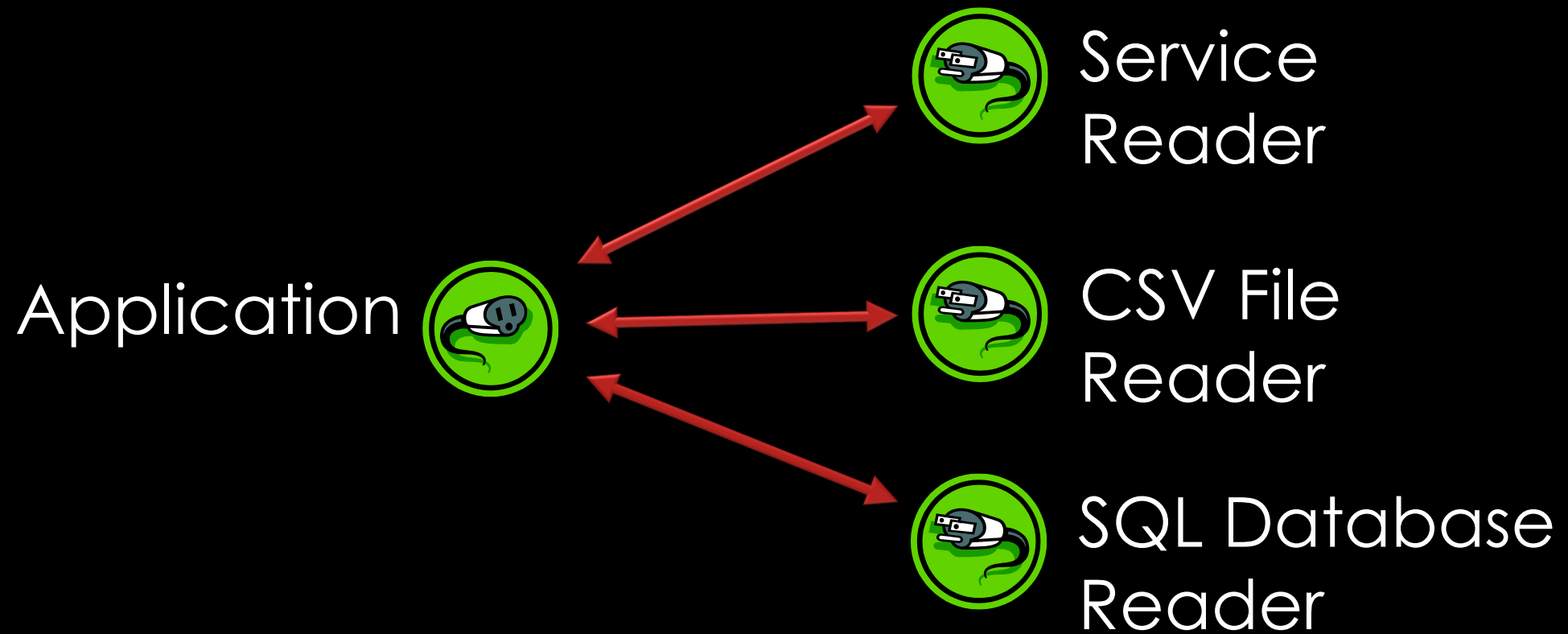
Amazon RDS

JSON

Hadoop

Azure Cosmos DB

Pluggable Data Readers



Benefits of Dynamic Loading

- Only ship 1 data reader assembly
- Remove dependency on concrete data readers
- New data readers can be added without modifying existing code

Configuration

```
<!-- Settings for CSV Reader -->
<appSettings>
  <add key="ReaderAssembly" value="PersonReader.CSV.dll"/>
  <add key="ReaderType" value="PersonReader.CSV.CSVReader"/>

  <add key="CSVFileName" value="People.txt"/>
</appSettings>
```

- Assembly File Name
- Fully-Qualified Type Name
- Other configuration

Configuration

```
<!-- Settings for CSV Reader -->
<appSettings>
  <add key="ReaderAssembly" value="PersonReader.CSV.dll"/>
  <add key="ReaderType" value="PersonReader.CSV.CSVReader"/>

  <add key="CSVFileName" value="People.txt"/>
</appSettings>
```

- Assembly File Name
- Fully-Qualified Type Name
- Other configuration

Steps

- Get assembly file name from configuration
- Create a custom load context
- Load the assembly into the context
- Get data reader type name from configuration
- Reflect into the assembly to get the reader type
- Use the Activator to create an instance of the reader

<https://jeremybytes.blogspot.com/2020/01/dynamically-loading-types-in-net-core.html>

Limiting Reflection

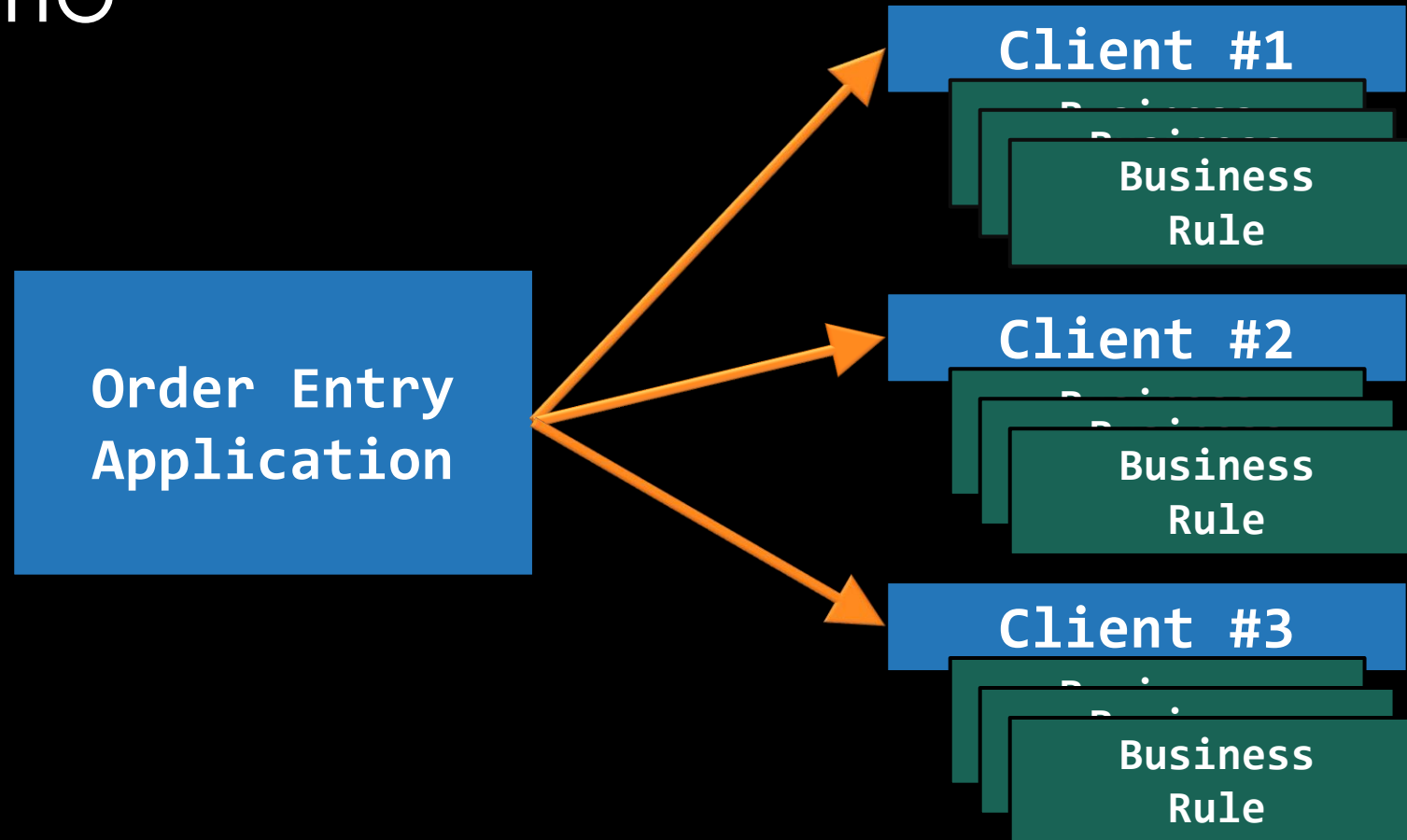
```
private async void FetchButton_Click(object sender, RoutedEventArgs e)
{
    IPersonReader reader = ReaderFactory.GetReader();

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

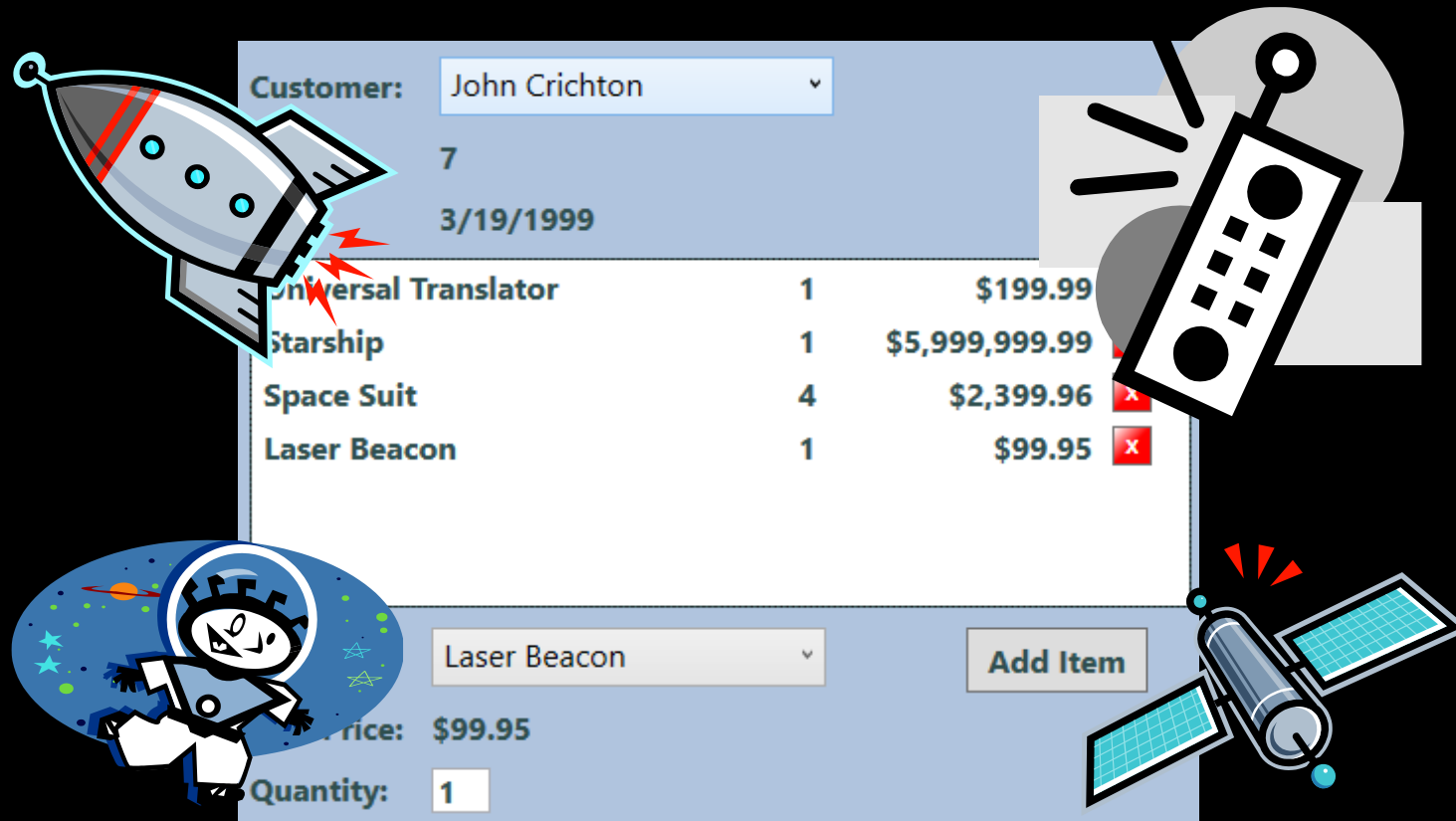
    ShowReaderType(reader);
}
```

- No Reflection Here
- Method calls through IPersonReader

Scenario



Application



Customer: John Crichton

7

3/19/1999

Universal Translator	1	\$199.99	
Starship	1	\$5,999,999.99	
Space Suit	4	\$2,399.96	x
Laser Beacon	1	\$99.95	x

Laser Beacon

Price: \$99.95

Quantity: 1

Add Item

The application interface is decorated with several space-themed illustrations: a rocket ship on the left, a satellite on the right, and an astronaut in the bottom left corner. The interface includes a customer selection dropdown, a list of items with their quantities and prices, and an 'Add Item' button.

Business Rule Interface

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

public record 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 that implement the Rule interface
- Create an instance of each Rule
- Add it to the Rule instance to the Rule Catalog

Summary

- There are lots of things you *can* do
- Some things that are dangerous (such as accessing private members)
- Reflection is slow
 - Limit the amount of reflection
 - Use interfaces or delegates
- Dynamic loading of assemblies is very useful in certain applications



Thank You!

Jeremy Clark

- jeremybytes.com
- youtube.com/jeremybytes
- github.com/jeremybytes/sdd-2024