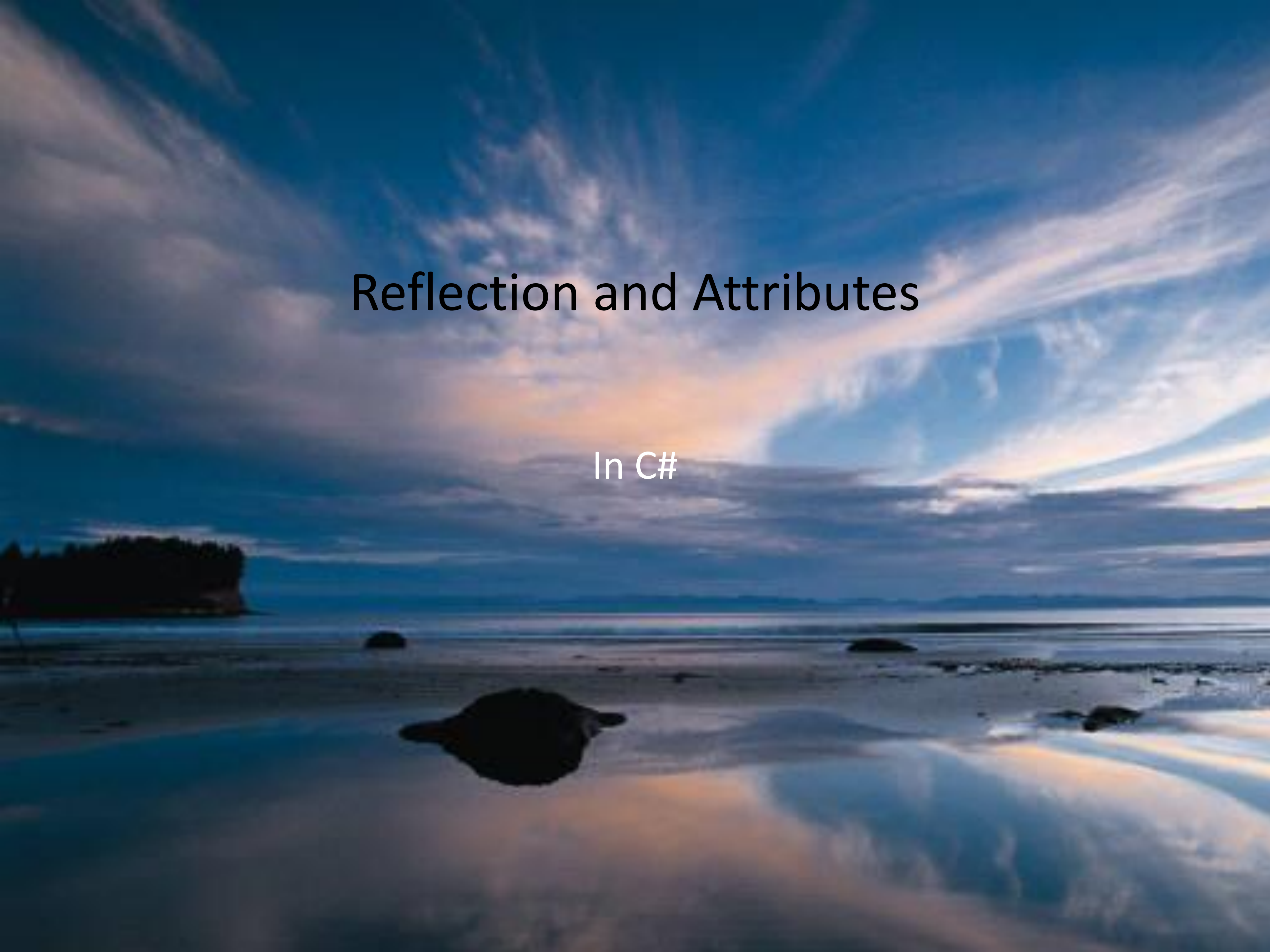
Reflection and Attributes

In C#



Agenda

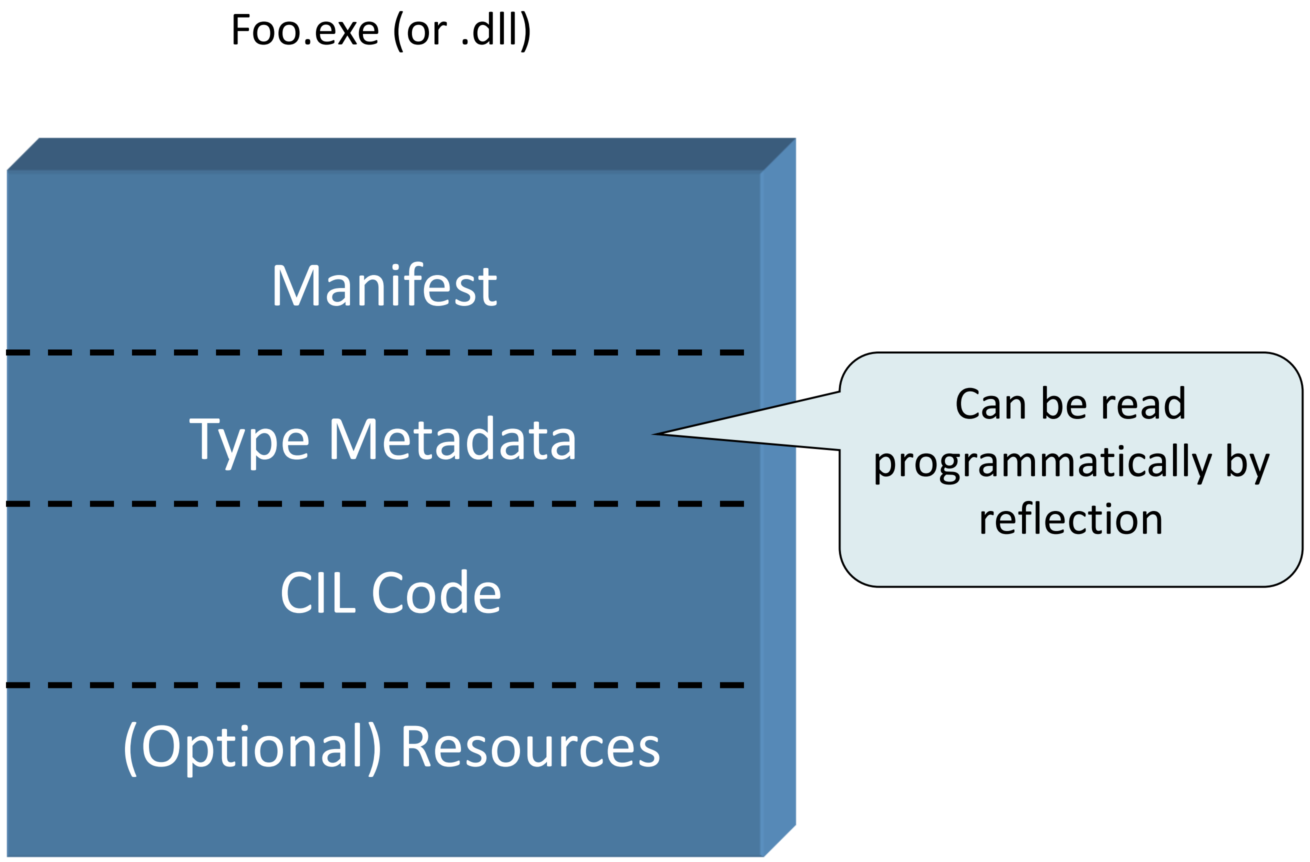
- Reflection
- Attributes

What?

- Reflection is a runtime facility that allows you to write code that can examine data types and metadata at runtime for any variable in the program.

- Metadata
  - Single location for type information
  - Every .NET object can be queried for its type
  - On runtime you can instantiate a type not known at compiletime

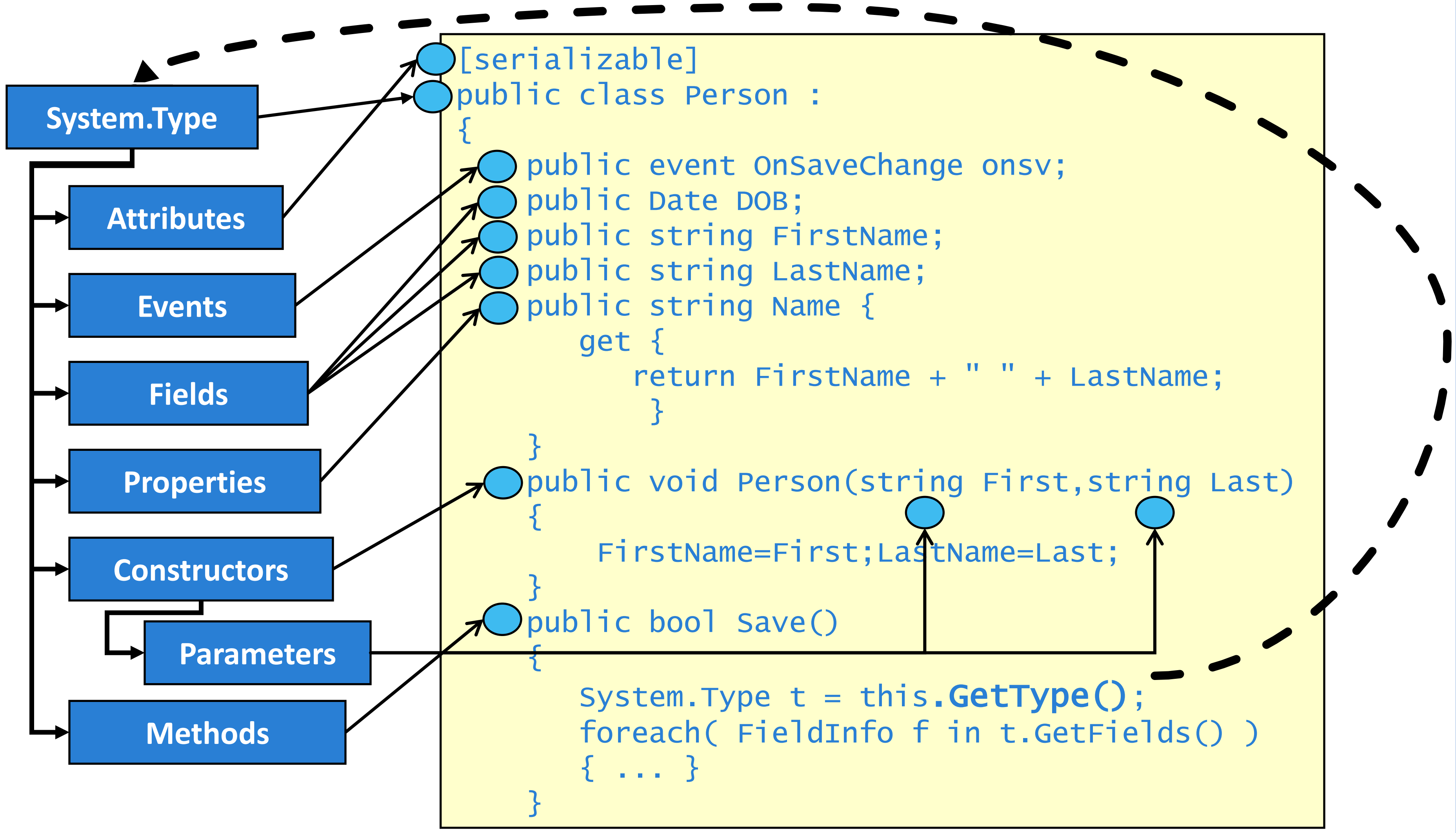
A Single File Assembly



How?

- Every class in .Net (wether build in or custom) inherits the method `GetType()` from `System.Object.`
  - `System.Type type = myObejct.GetType();      //use on objects`
- An alternative approach is to use the **typeof** operator:
  - `System.Type type = typeof(MyClass);         //use on types`
- Or you may use the static method GetType in the Type class:
  - `System.Type type = Type.GetType("MyClass");//use on type names`
- `System.Type`
  - Is the root of the System.Reflection functionality
  - Provides access to metadata for any .NET type
  - Allows drilling down into all facets of a type by use of its members
    - Category: Simple, Enum, Struct or Class
    - Methods and Constructors, Parameters and Return
    - Fields and Properties, Arguments and Attributes
    - Events, Delegates, and Namespaces

Exploring Metadata

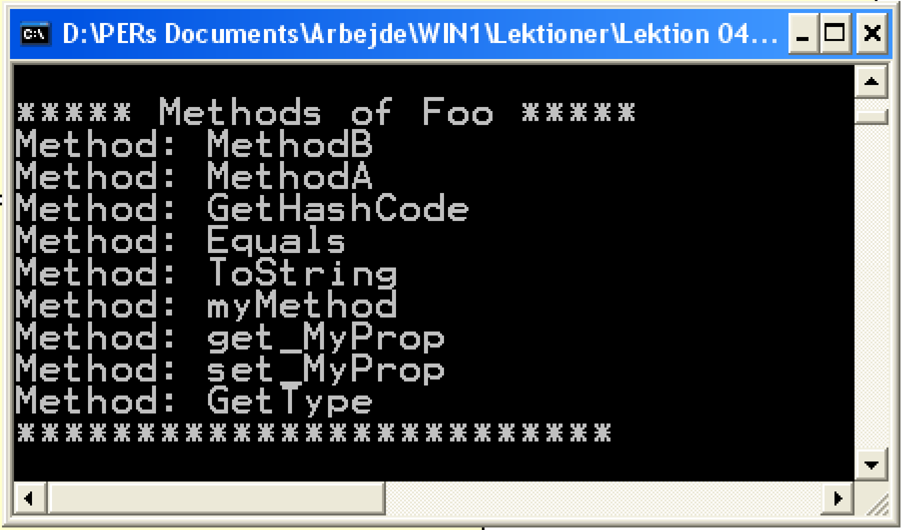


```
[serializable]
public class Person :
{
    public event OnSaveChange onsv;
    public Date DOB;
    public string FirstName;
    public string LastName;
    public string Name {
       get {
           return FirstName + " " + LastName;
        }
    }
    public void Person(string First,string Last)
    {
         FirstName=First;LastName=Last;
    }
    public bool Save()
    {
       System.Type t = this.GetType();
       foreach( FieldInfo f in t.GetFields() )
       { ... }
    }
```

MetaData Samples

```
public class Foo: IFaceOne, IFaceTwo
{
   // Fields.
   public int myIntField;
   public string myStringField;
   // A method.
   public void myMethod(int p1, string p2) { }
   // A property.
   public int MyProp { get {return myIntField;} set {myIntField = value;} }
   // IFaceOne and IFaceTwo impl.
   public void MethodA() {}
   public void MethodB() {}
}
```

```
public static void ListMethods(Foo f)
{
  Console.WriteLine("***** Methods of Foo
  Type t = f.GetType();
  MethodInfo[] mi = t.GetMethods();
  foreach(MethodInfo m in mi)
    Console.WriteLine("Method: {0}", m.Name);
  Console.WriteLine("*************************\n");
}
```



```
***** Methods of Foo *****
Method: MethodB
Method: MethodA
Method: GetHashCode
Method: Equals
Method: ToString
Method: myMethod
Method: get_MyProp
Method: set_MyProp
Method: GetType
*************************
```

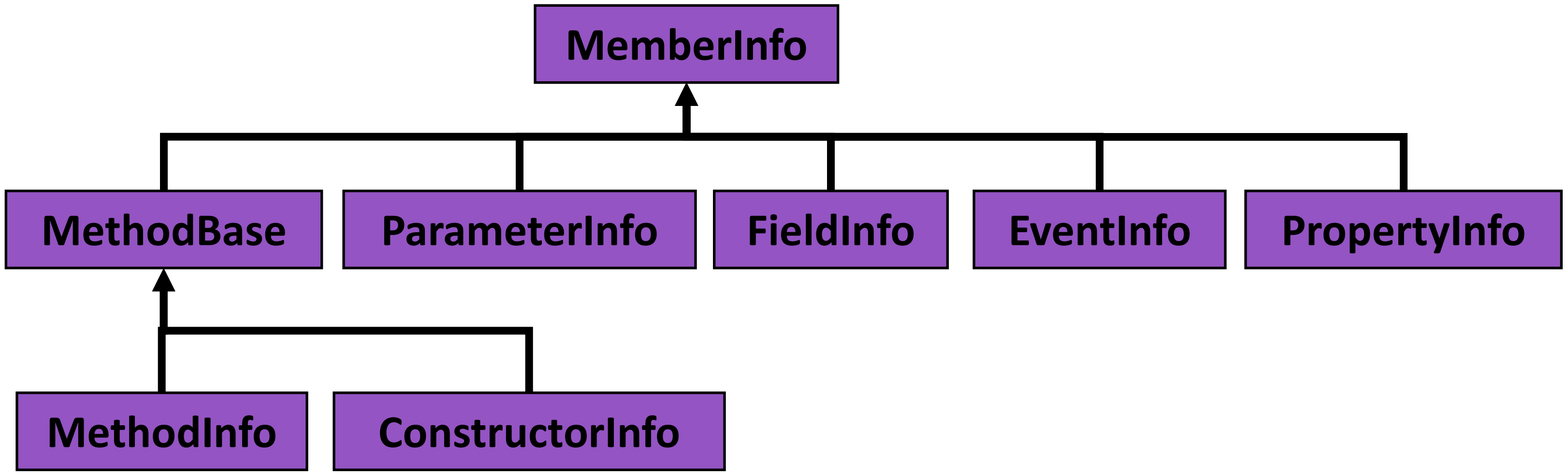
Dynamic Creation Example

- Dynamic Invocation through Reflection
  - Support for late binding

```
Assembly a = null;
a = Assembly.Load("CarLibrary");
// Get the Minivan type.
Type miniVan = a.GetType("CarLibrary.MiniVan");
// Create the Minivan on the fly.
object obj = Activator.CreateInstance(miniVan);
// Create array of params.
object[] paramArray = new object[2];
paramArray[0] = "Fred";
paramArray[1] = 4;
MethodInfo mi = miniVan.GetMethod("TellChildToBeQuiet");
mi.Invoke(obj, paramArray);
```

MemberInfo

- Base class for all "member" element descriptions
  - Fields, Properties, Methods, etc.
- Provides member kind, name, and declaring class



ATTRIBUTES

Attributes

```
[myAttribute]
public class MyClass
{}
```

- Custom attributes are the killer-app for Reflection!
- Attributes enable declarative behavior
- Attributes allow data augmentation

- Any kind of programming structure can be mark by an attribute – not just classes.

Attributes

```
[serializable]
class Person
{
 ...
```

Mark a class as serializable

This attribute is build in .Net

You ask for custom attributes by use of System.Type:

`Type.GetCustomAttributes()`

```
[dbcolumn("Address1")] string Street;
[dbcolumn("Postcode")] string ZIP;
```

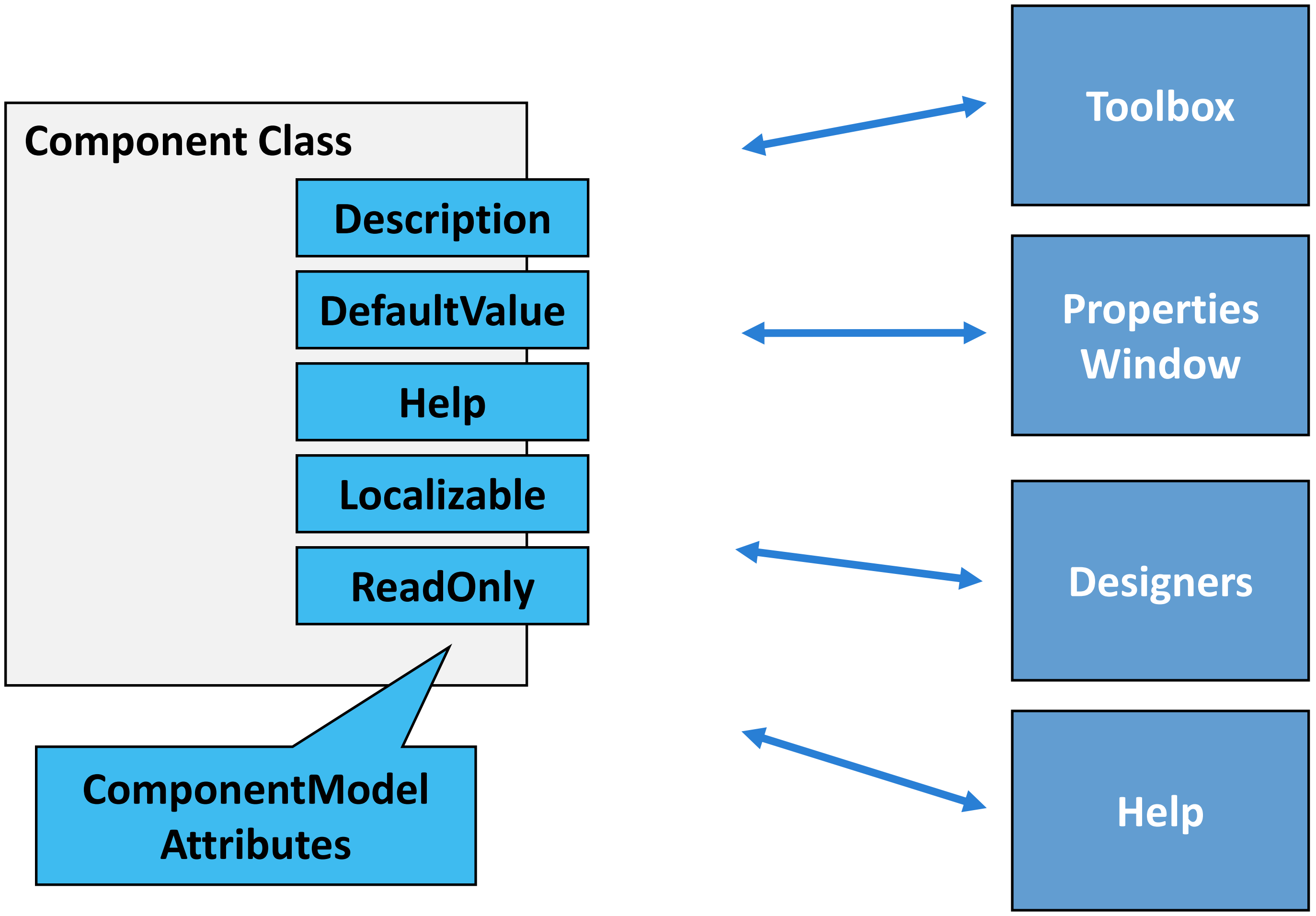
Map fields to database columns with

This is a custom attribute

`FieldInfo.GetCustomAttributes()`

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Visual Studio.NET and Reflection



How to Define Your Own Attribute

- You just create a class that derives from System.Attribute

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public class VehicleDescriptionAttribute : System.Attribute
{
   private string description;
   public string Desc
   {
      get { return description; }
      set { description = value; }
   }

   public VehicleDescriptionAttribute() {}
   public VehicleDescriptionAttribute(string desc)
   { description = desc;}
}
```

How to Use Attibutes

```
[VehicleDescription("A very long, slow but feature rich auto")]
public class Winnebago
{
   public Winnebago()
   {
   }
}
```

```
public static int Main(string[] args)
{
   // Get the Type of winnebago.
   Type t = typeof(Winnebago);

   // Get all attributes for the class.
   object[] customAtts = t.GetCustomAttributes(false);

   // List all info.
   foreach(VehicleDescriptionAttribute v in customAtts)
      Console.WriteLine(v.Desc);
   return 0;
}
```

Summary

- Reflection = System.Type + GetType() or typeof
- You can explore type information at runtime
- Reflection enables attribute-driven programming