C# and the CLR

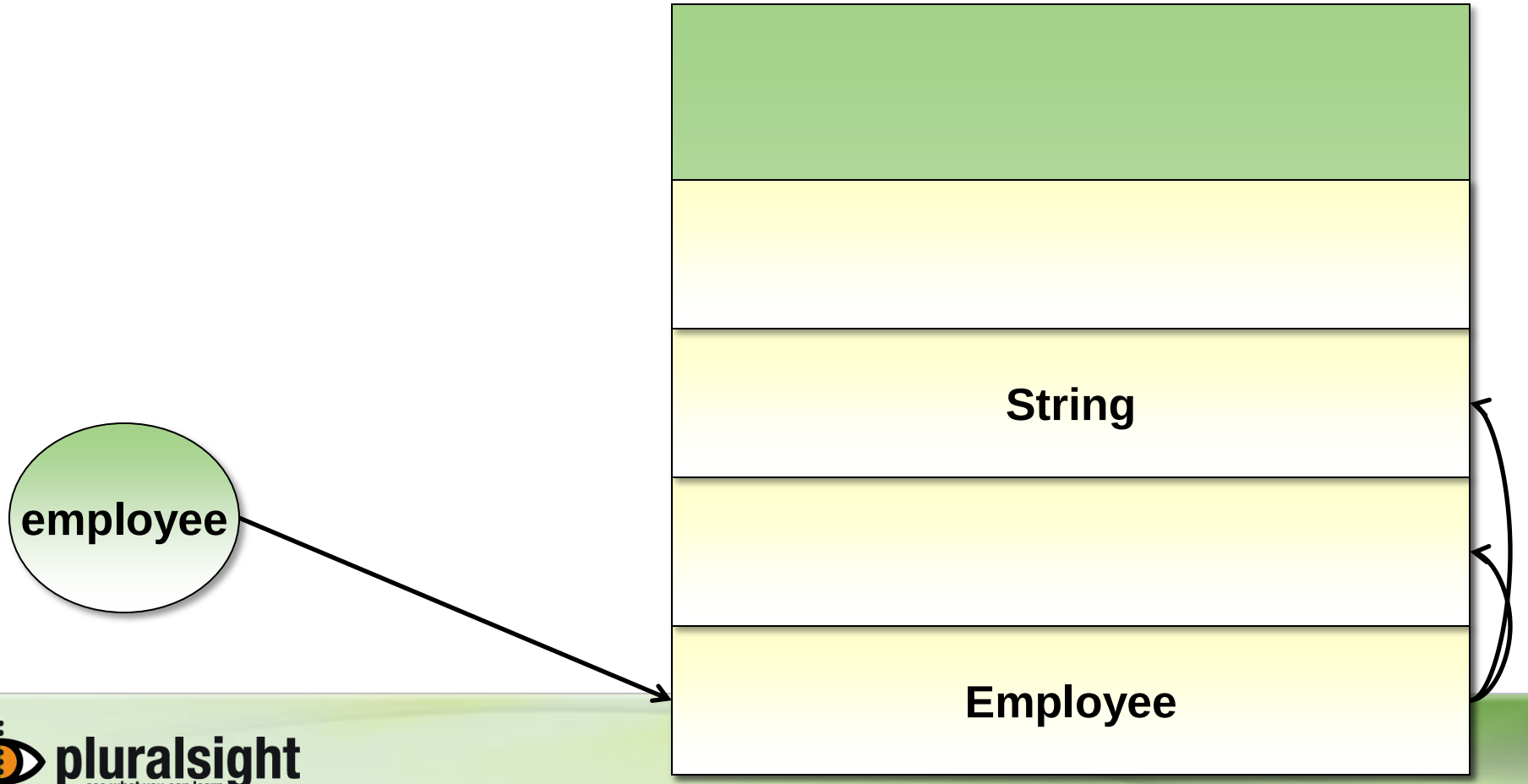
Best Friends Forever

Overview

- JIT compilation and garbage collection
- Threads
- Reflection and metadata
- Processor architecture
- Interoperability

Garbage Collection

- **Garbage collector cleans up unused memory**
  - Visits global variables and local variables to determine what is in use



Threads

- **System.Threading**
  - Low level API for starting, stopping, and joining threads
- **System.Threading.Tasks**
  - High level API for concurrent and asynchronous programming

```
string[] urls = {"http://pluralsight.com",
                 "http://microsoft.com",
                 "http://odetocode.com" };

Parallel.ForEach(urls, url => {
    var client = new WebClient();
    var result = client.DownloadString(url);
    // ...
});
```

Reflection

- CLR provides an API for self-examination
- System.Type is the starting point for reflection

```
Type type = o.GetType();

PropertyInfo[] properties = type.GetProperties();
foreach (var propertyInfo in properties)
{
    Console.WriteLine(propertyInfo.Name);
}
```

Custom metadata

```
[AttributeUsage(AttributeTargets.Property)]
public class DangerZoneAttribute : Attribute
{
    public DangerZoneAttribute(int min, int max)
    {
        Minimum = min;
        Maximum = max;
    }

    public int Minimum { get; set; }
    public int Maximum { get; set; }
}
```

```
[DangerZone(12,18)]
public int Age { get; set; }
```

Invoking Methods and Properties

```
Type type = o.GetType();
PropertyInfo info = type.GetProperty("Length");
int length = (int) info.GetValue(o, null);
```

```
Type type = o.GetType();
MethodInfo info = type.GetMethod("Compute");
int result = (int) info.Invoke(o, new object[] {2, 5});
```


```
dynamic someObject =  o;
int length  = someObject.Length;
int result  = someObject.Compute(2,5);
```

Creating Objects

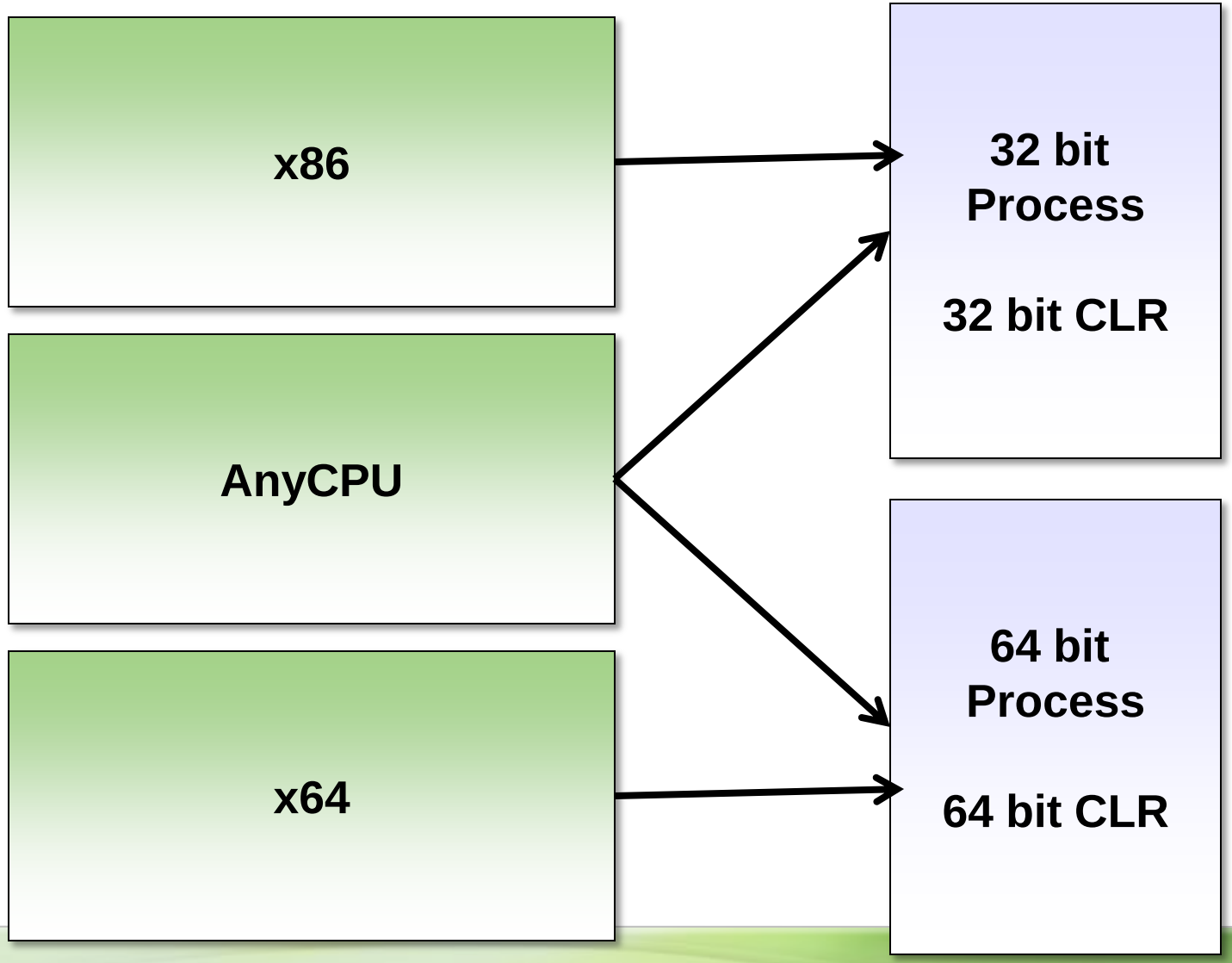
- **Activator class provides static methods to instantiate types**

```
var assembly = Assembly.LoadFrom("plugin.dll");
foreach(var type in assembly.GetTypes())
{
    if(type.GetInterface("ILogger") != null)
    {
        var logger = Activator.CreateInstance(type);
    }
}
```

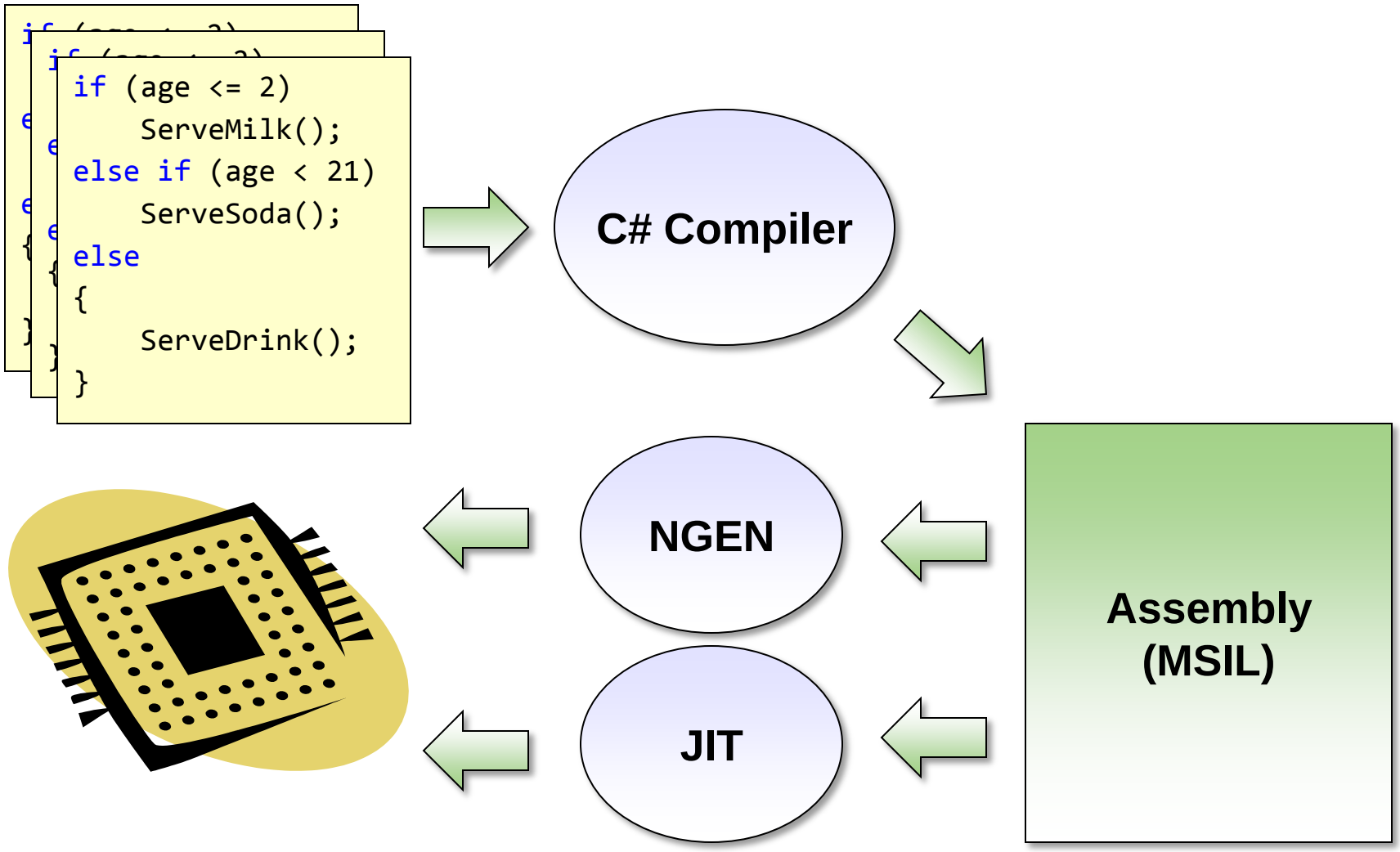
Creating Code

```
MethodInfo methodInfo = typeof (Console).GetMethod("WriteLine",
                        new Type[] {typeof (string)});

DynamicMethod method = new DynamicMethod(
    "HelloWorld", typeof(void), new Type[]{});

ILGenerator il = method.GetILGenerator();
il.Emit(OpCodes.Ldstr, "Hello, world");
il.Emit(OpCodes.Call, methodInfo);
il.Emit(OpCodes.Ret);

Action action = (Action)method.CreateDelegate(typeof (Action));
action();
```

C# on the Metal

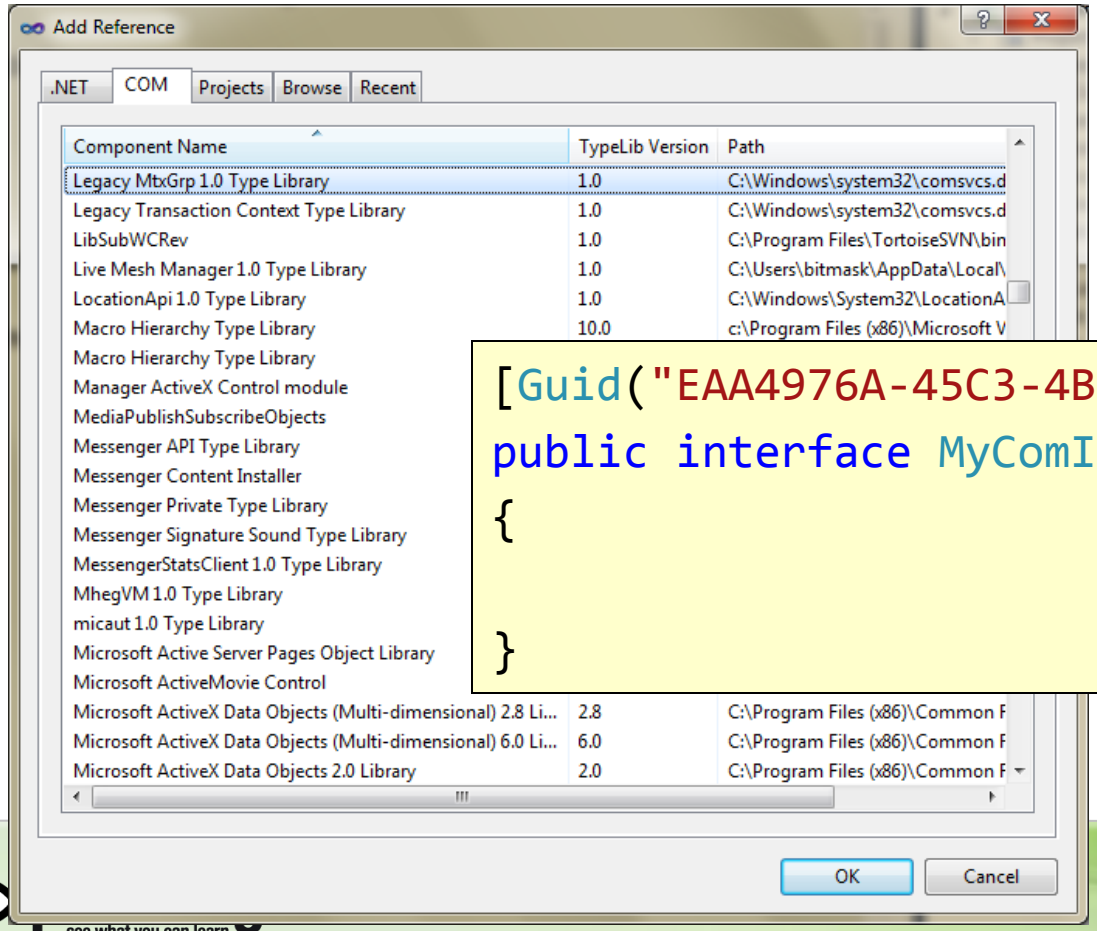


C# - From Source To CPU



COM Interop

- C# can consume COM components
- C# code can package itself as a COM component




```
[Guid("EAA4976A-45C3-4BC5-BC0B-E474F4C3C83F")]
public interface MyComInterface
{


}
```

PInvoke Interop

- **Platform Invoke**
  - Can call into Windows APIs and unmanaged code

```
public static class NativeStuff
{
    public static void Beep()
    {
        if (!MessageBeep(0))
        {
            Int32 err = Marshal.GetLastWin32Error();
            throw new Win32Exception(err);
        }
    }

    [DllImport("User32.dll")]
    static extern Boolean MessageBeep(UInt32 beepType);
}
```

Summary

- C# - tightly integrated with the CLR
  - Can still interop with COM
  - Can still interop with native code
- Metadata drives many features
  - Garbage collection
  - Reflection