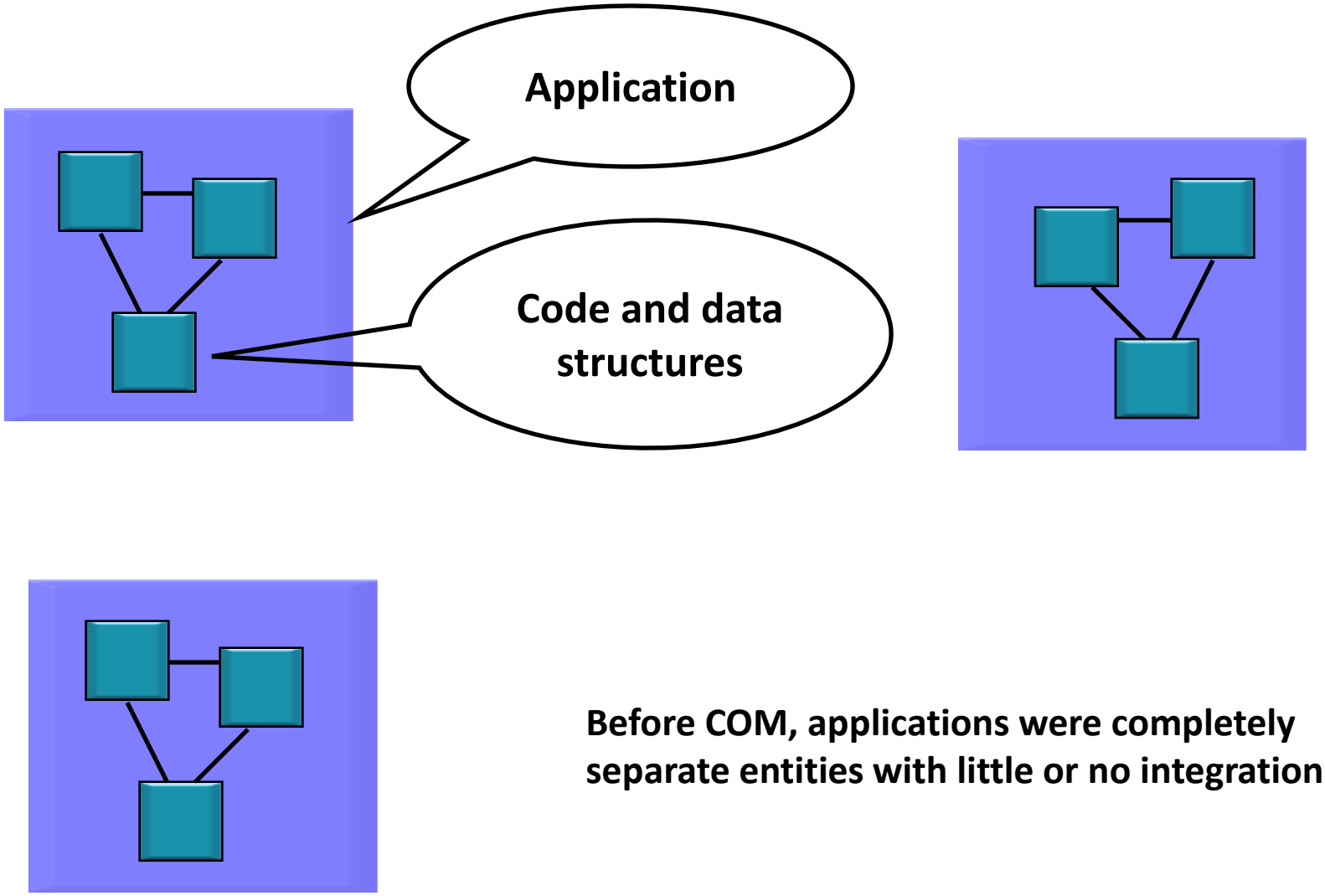
Components in .Net

Agenda

- The .NET Evolution
- .Net's component architecture
  - Overview
  - Structure
  - Version numbers
  - Sharing
- How to use software components
- How to design software components

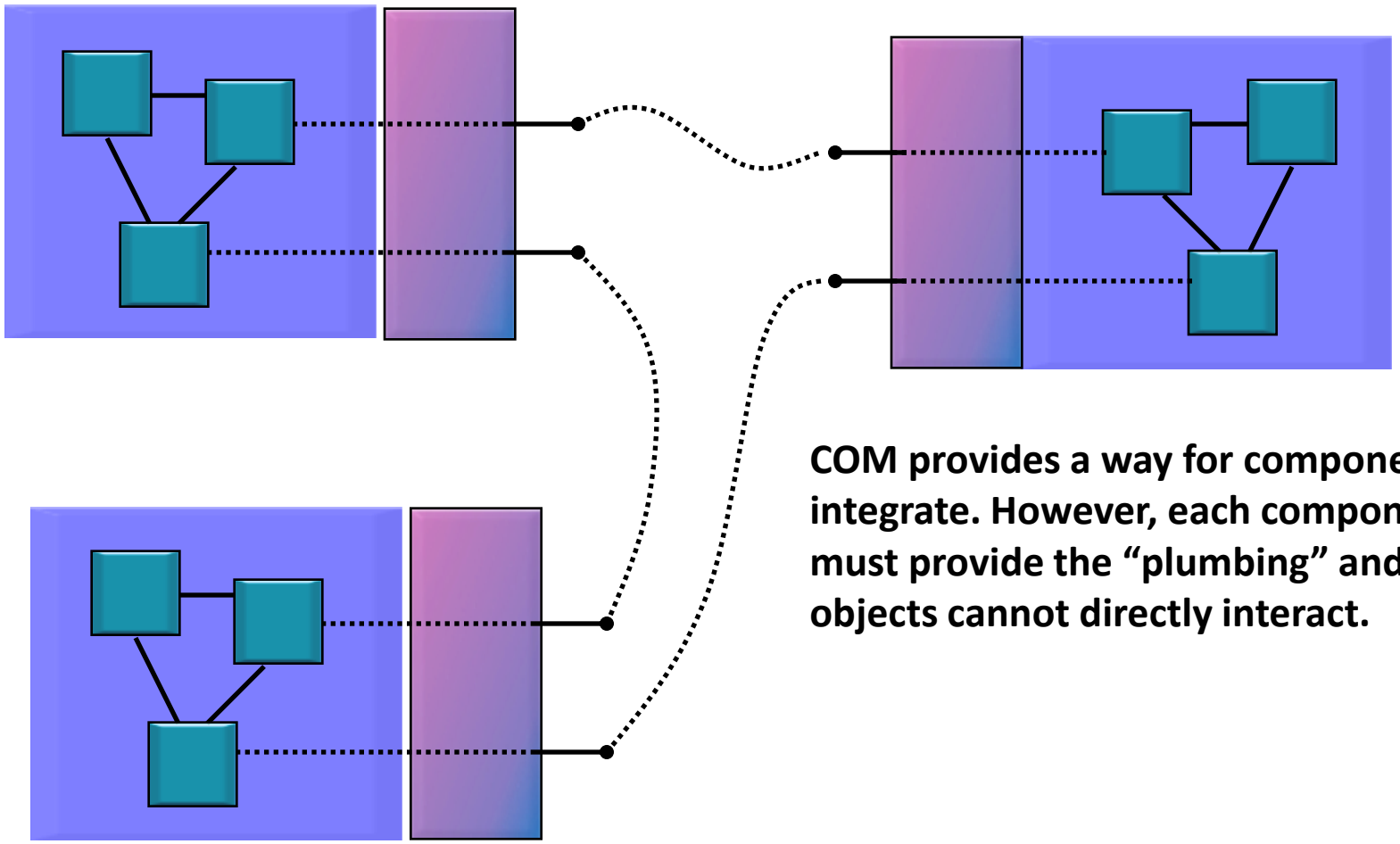
THE .NET EVOLUTION

The .NET Evolution



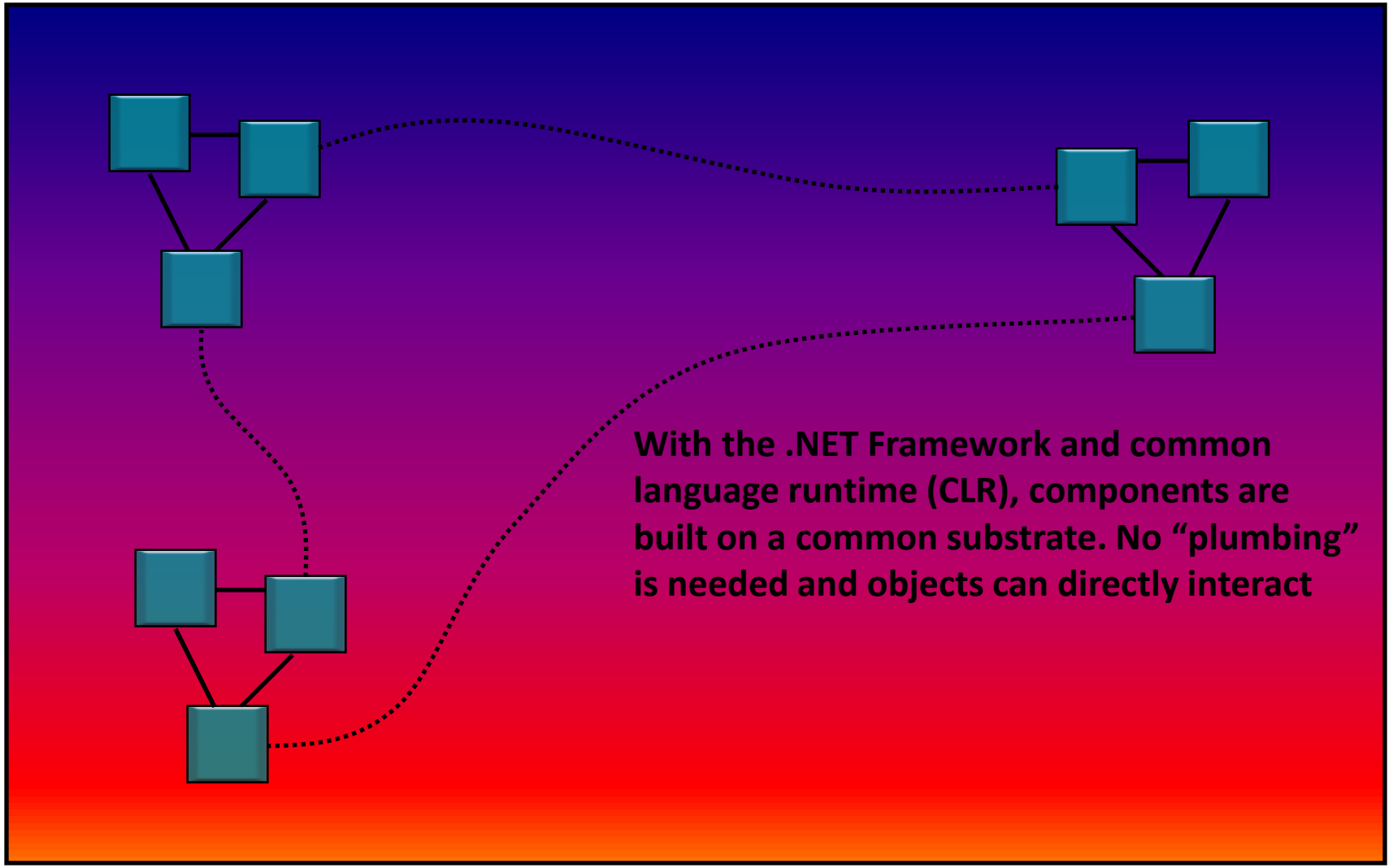
Before COM, applications were completely separate entities with little or no integration

The .NET Evolution

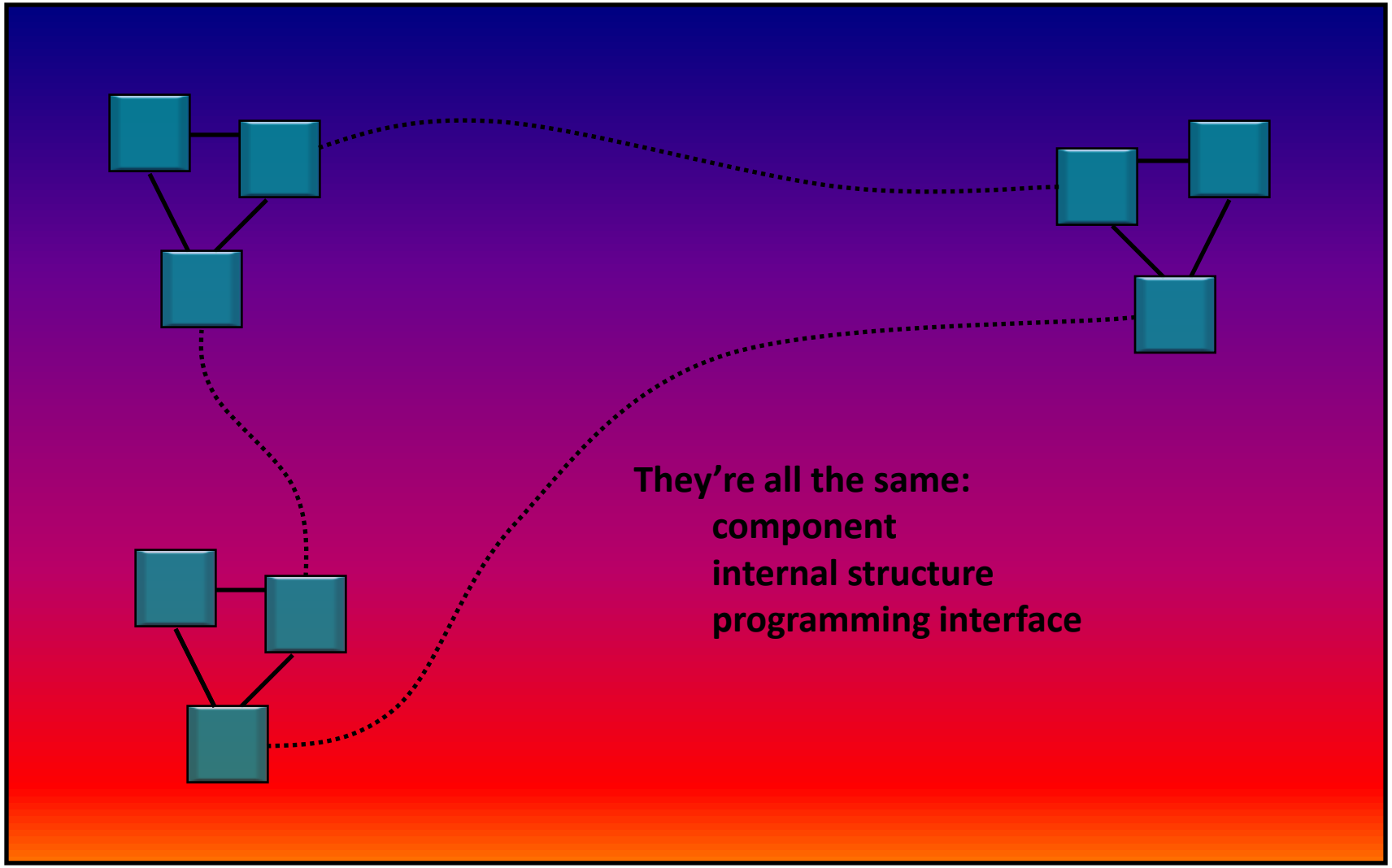


COM provides a way for components to integrate. However, each component must provide the "plumbing" and objects cannot directly interact.

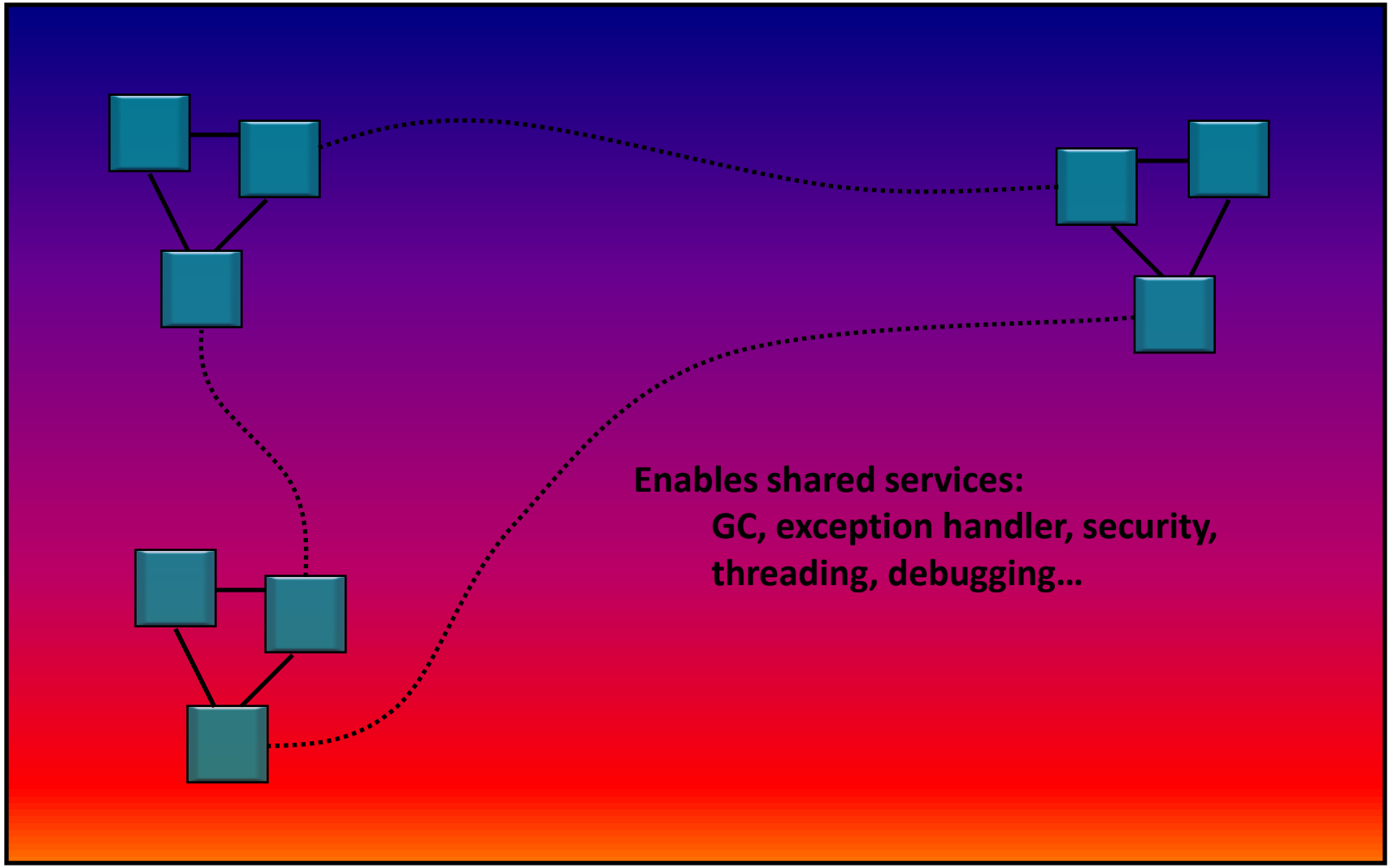
The .NET Evolution



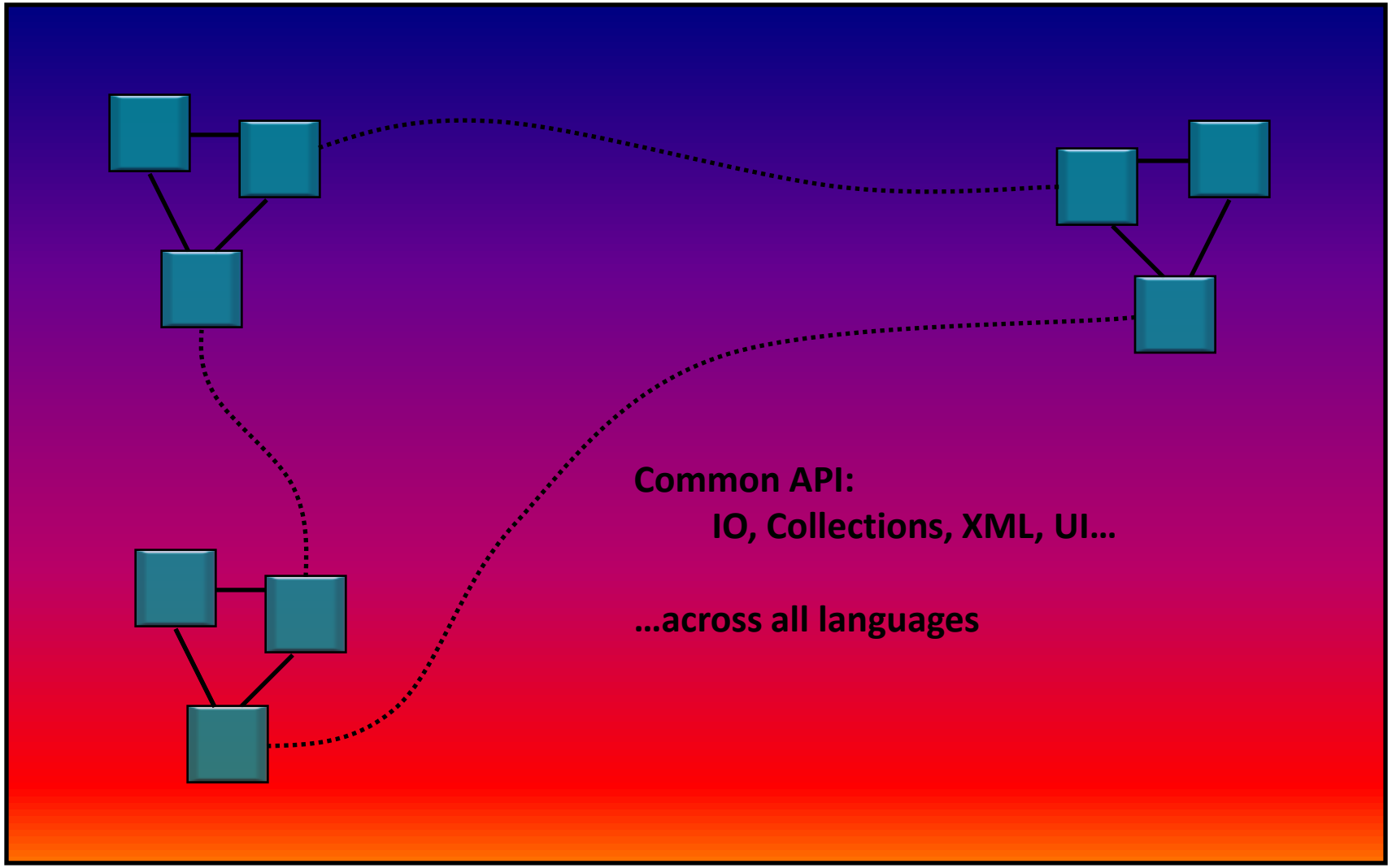
The .NET Evolution



The .NET Evolution



The .NET Evolution



.Net's component architecture

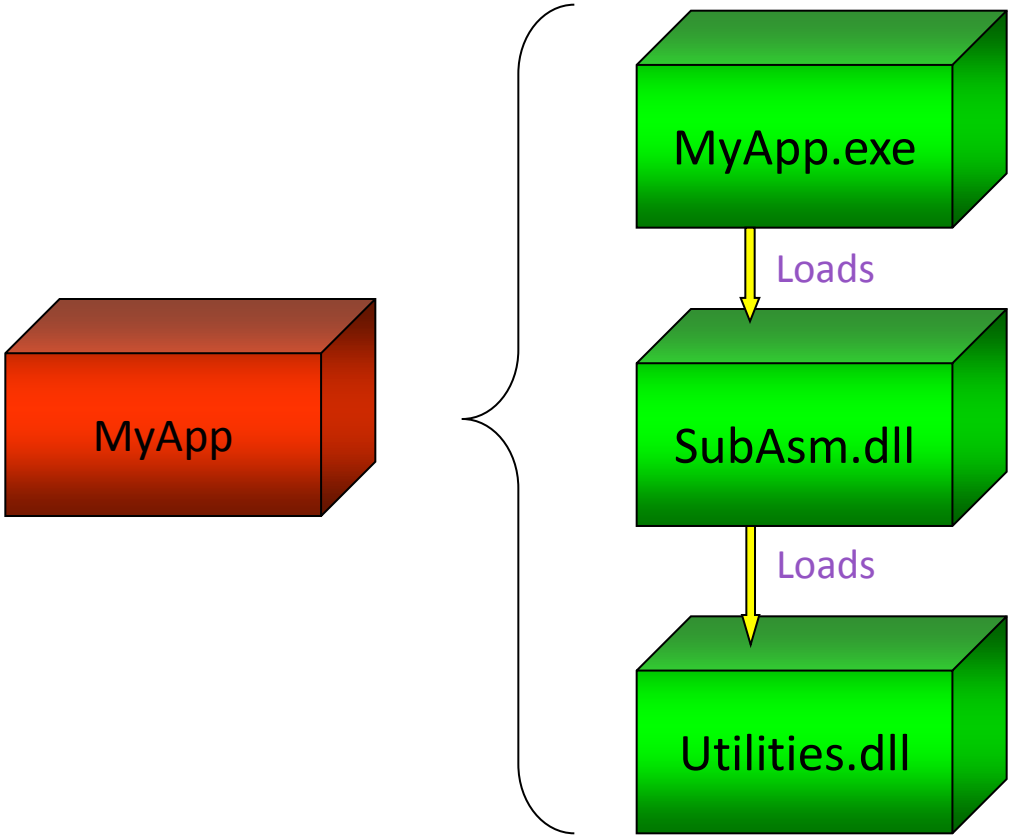
life after iunknown

.Net Simplify Development

- Completely eliminates com plumbing
- No more…
  - Registration **=>self described apps**
  - GUIDs **=>hierarchical namespaces**
  - .IDL files **=>unified object model**
  - HRESULTs **=>exceptions**
  - Iunknown **=>root object**
  - AddRef/release **=>garbage collector**
  - CoCreateInstance **=>”new” operator**

The building blocks of .Net

Windows Applications consist of one or more **assemblies**



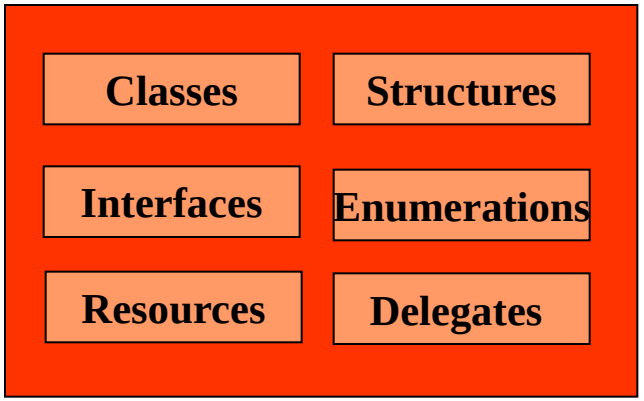
Only assemblies with .exe extension can be started from windows.

.dll assemblies cannot be executed from shell.

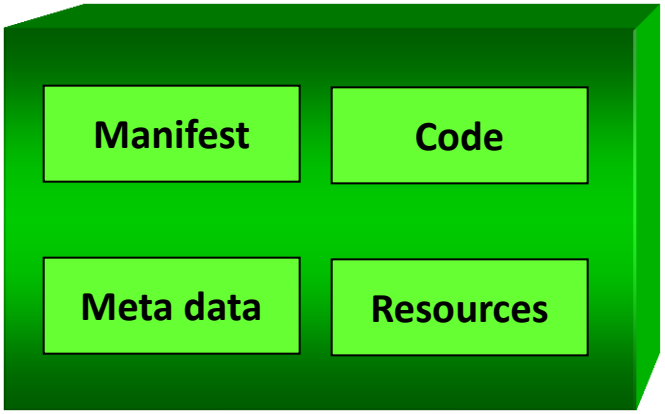
References are resolved at loadtime – no static linking!

Assemblies

Logical view



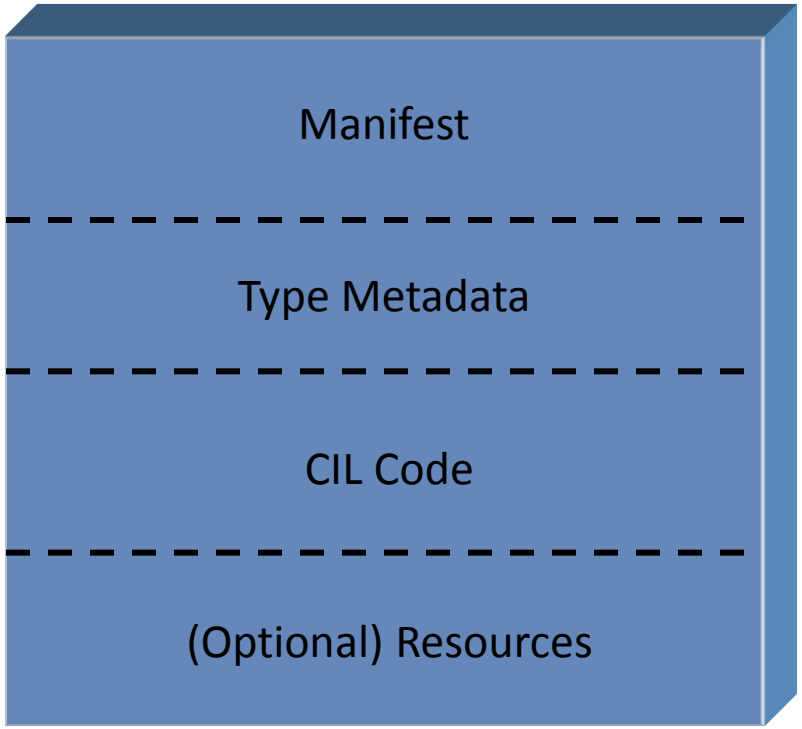
Physical view



- Unit of deployment
  - Consist of one or more modules
  - Self-describing via manifest and metadata
- Level of versioning
  - Version number is build in by compiler
  - E.g. `2.5.719.2`
    major.minor.build.revision
- Security boundary
  - Assemblies are granted permissions based on evidence of origin and author
- Scope boundary
  - Types named are relative to assembly

A Single Module Assembly

Foo.exe (or .dll)



Manifest contains info about:

Strong name
- Assembly name
- Version information
- PublicKeyToken
- Culture information

- Files that make up this assembly
- Referenced assemblies
- Etc. – *You may add custom info*

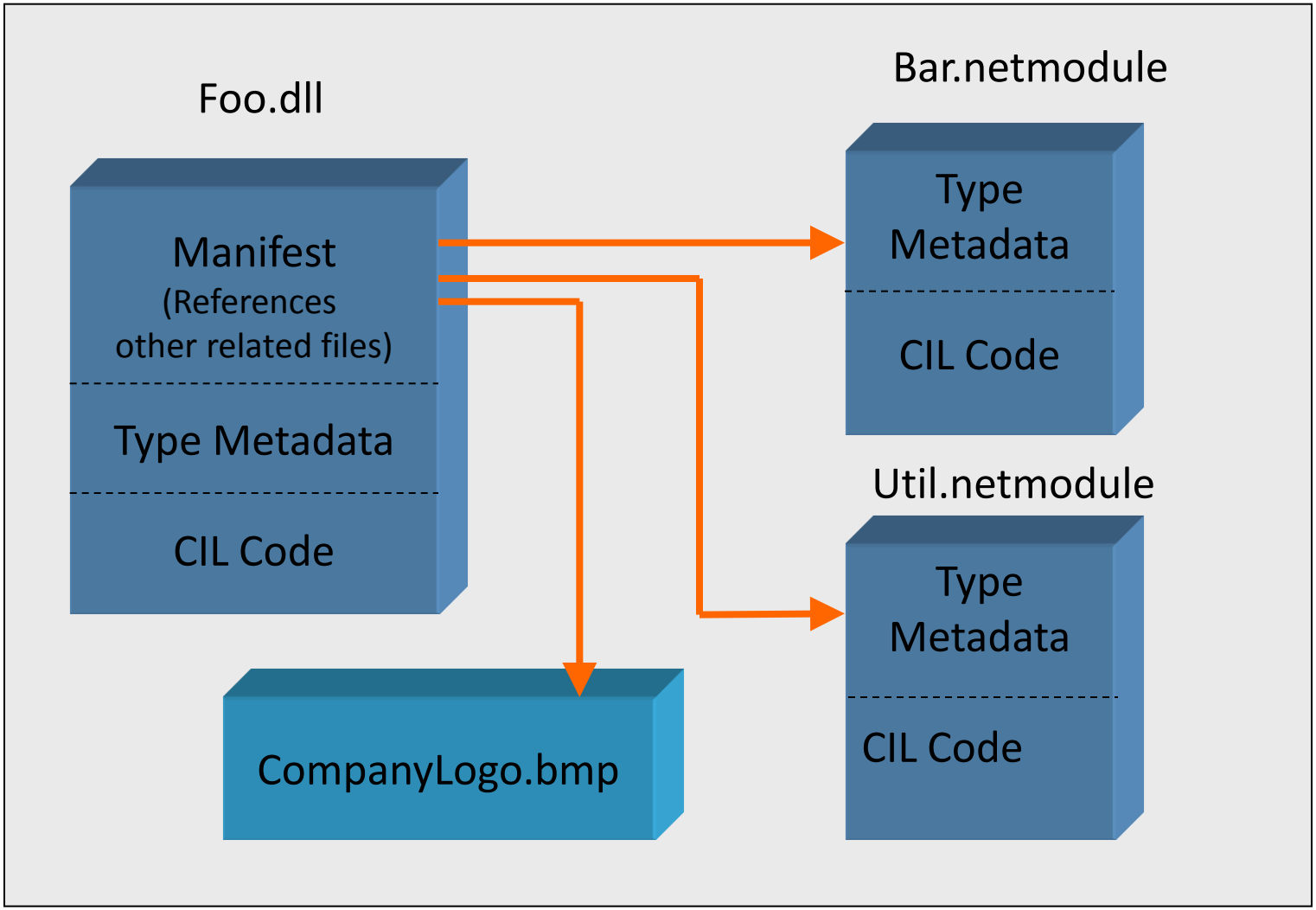
Metadata contain a complete description of all data types within this module

The **Code** block contains the **C**ommon **I**ntermediate **L**anguage code (*also known as MSIL*) – (or native code)

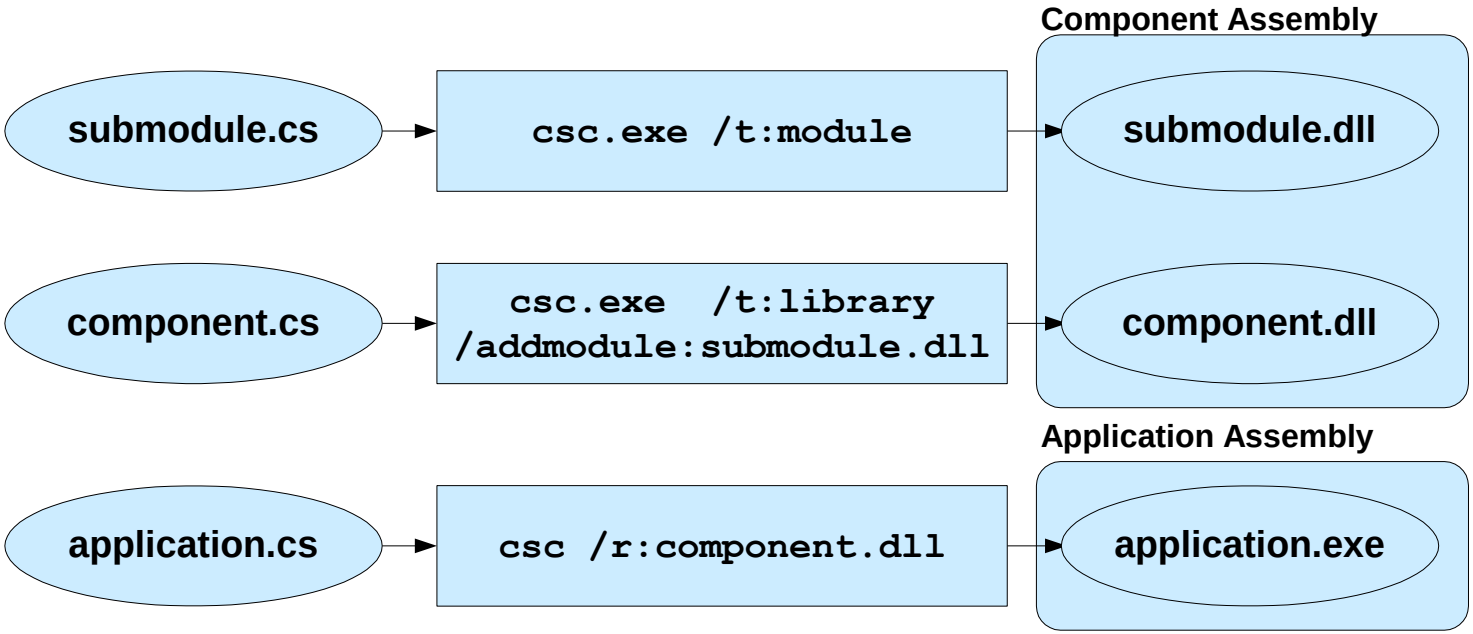
Resources may be images, script files and similar items

A Multi File Assembly (Rarely Used)

Satalite modules can only be deployed as part of an assembly, but can be (down-)loaded by containing assembly only if needed.



Types and multi file assemblies



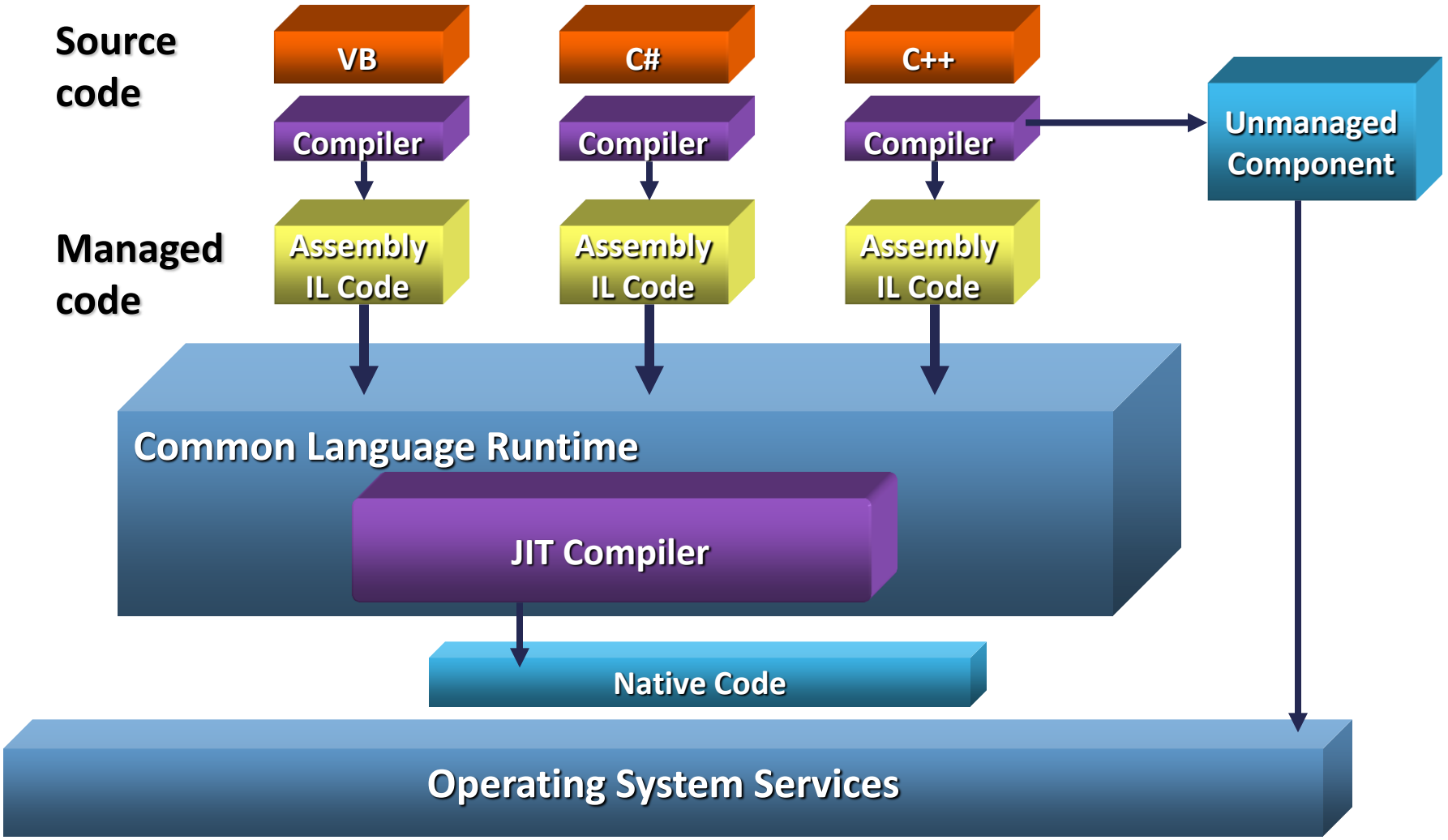
Logical view:

The application consist of 2 assemblies

Physical view:

The application consist of 3 files

.Net Framework Execution Model



Assembly Names

- Each assembly has a four-part name that uniquely identifies it
  friendly name=MyAssembly, **Version=**1.2.3.4, **Culture=**en-US, **PublicKeyToken=**a169ca1231b23456
- The four parts are:
  - A friendly name
    - This is the name of the file with the assembly manifest
  - Version
    - All assemblies has a four-part version number. If you don't set it, the compiler will set in to 0.0.0.0
  - Culture
    - Two-part string indicating spoken language and region as specified in RFC 1766. For non localized assemblies the culture is set to neutral
  - Public Key
    - **Is used to sign the assembly with a strong name.**
    - Is set to null if not used.
    - This key indicate the producer of this assembly.

Installation

- Private vs. shared assemblies
  - Private assemblies are deployed in local directory
  - Shared assemblies are stored in Global Assembly Cache (GAC) or in local directory
- To install a strong named assembly in the GAC use:
  - **gacutil –i myAssembly.DLL**
  - **Or drag-and-drop the assembly to the C:\WINDOWS\assembly folder**
- To view the content of the GAC use
  - Gacutil -l
  - Or open the „folder“ **C:\WINDOWS\assembly in Windows Explorer** (only possible if the Cache viewer shell extension (`shfusion.dll`) is installed)
  - Or by use the snap-In for Management Console (**`mscorcfg.msc`**)

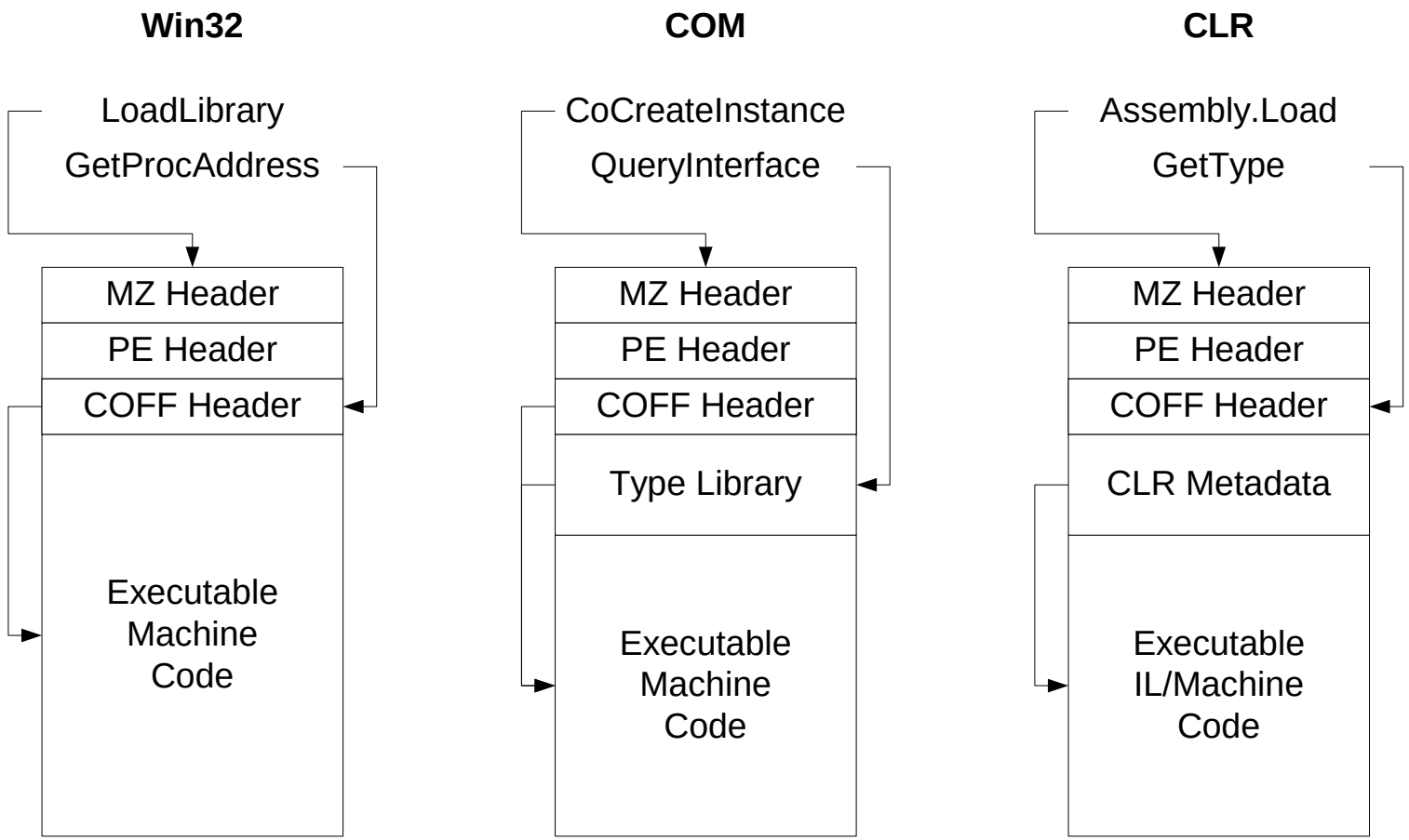
Global Assembly Cache Advantages

- Using the GAC has advantages:
  - Performance improvements
  - Integrity checking
  - File security
  - Versioning
  - Automatic pickup of Quick Fixes
  - Additional search location

Loading an Assembly

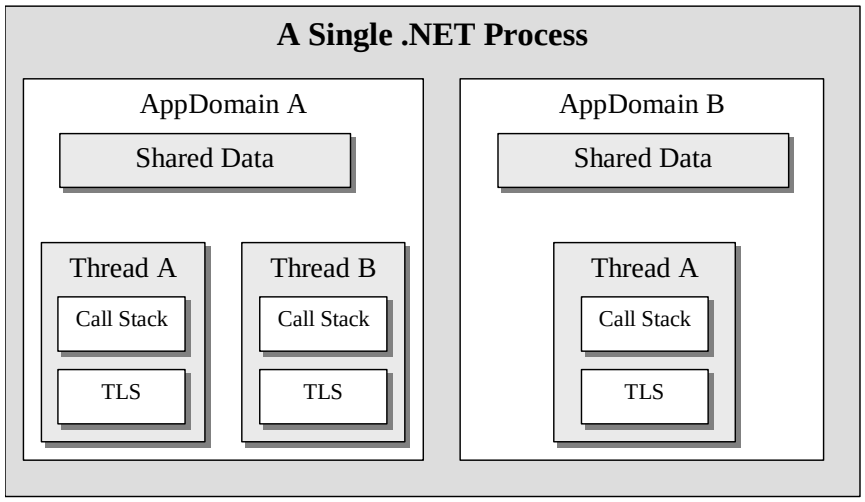
- Assembly is Portable Executable (PE) file ...
  - ... with CLR related information added
- Runtime aware environment loads assembly directly
- Unaware operating system loads assembly as PE
  - Entry point: stub that loads and calls CLR
  - CLR examines addtional header information

The evolution of the loader



Application Domain

- Concept for application isolation
- Provide isolation at lower cost than processes
- AppDomains are created by the runtime host
- AppDomain is created for each application
- Direct references between AppDomains disallowed
  - Requires proxies or copied objects



Loader Optimization

- Assembly is SingleDomain by default
  - Each AppDomain loads and compiles assembly
- Assembly can be marked for MultiDomain use
  - Assembly is compiled once
  - Mapped into all referencing AppDomains
  - A copy is available for each process
  - References to static data is indirected
  - Assembly is unloaded when process ends
- MultiDomainHost
  - Copy of code is hosted in each AppDomain

Just-In-Time Compilation

- MSIL is made for compilation
  - Needs some per-method analysis
- Code is compiled when needed
  - Compilation on a per-method base
  - Code that is not called is not compiled
  - Loader creates stub for each method
- First step: Verification of type safety
- JITted code is not persisted
  - But code can be compilet at installation by use of ngen

PreJITting with ngen

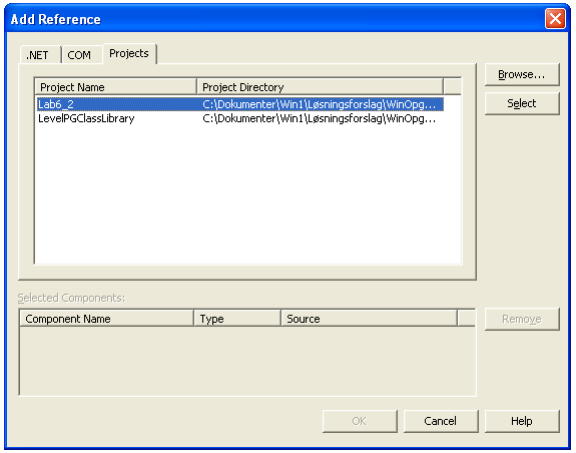
- Complete compile at installation time
  - PreJITted assembly is installed in GAC
- Speeds up loading time significantly
  - Both IL and native image are loaded
  - No verification needed
- Native image is not used...
  - ...When module version ID of IL is different
  - ...If the same applies to any dependencies
  - ...Assembly binding policy has changed
  - Fallback to normal JIT process

How to use software components

Assembly vs. Namespace

- Namespaces are used to group names (types) logically
- Assemblies are used to group names (types) physically
- Assemblies can contain several namespaces
- Namespaces can be partitioned across assemblies
- Names (types) are scoped to the assembly they are defined in.

- Both must be included into project independently:
  - Namespaces are imported in the source code
    - using System.Runtime.Remoting.Services;
  - Assemblies are referenced by compiler switch
    - csc /r:System.Runtime.Remoting.DLL ...
  - Or in Visual Studio by Adding a reference to the wanted assembly (DLL) by use of the „Add Reference“ menu item in the Project menu.



How to design software components

Non-UI Custom Controls

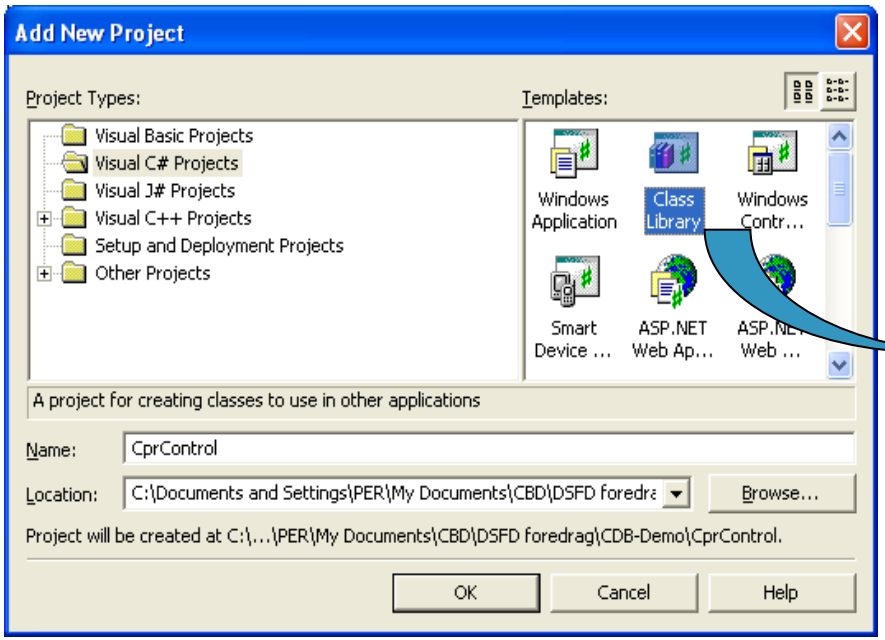
- Non-UI Controls are created with the Class Library project wizard

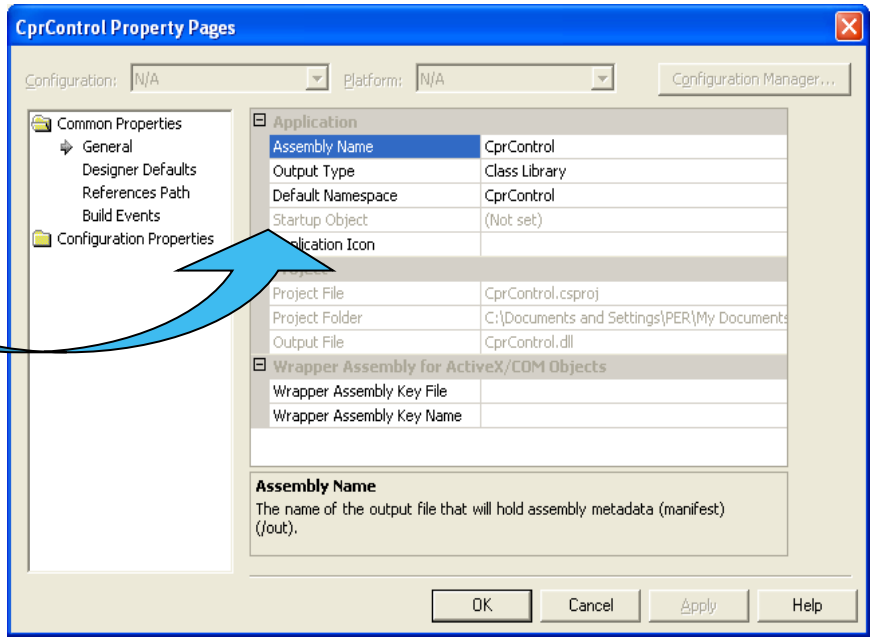
- To add design time capability to a class, the class must inherit from **`System.ComponentModel.Component.`**
  - All public properties will then show up in the designer
  - Give the class the custom attribute [ToolboxItem(true)] if you want it to show up in the toolbox.
  - Add it to the toolbox with the toolbox's add new item context menu item

- Every class that is to be used as a control must have a parameter-less public constructor if it is to be used with the VS.Net designer, or be visible to COM clients.

Components how to build 1

- Add a new Class Library project to the solution
  - **The wizard will then set output type to Class Library (a DLL-assembly)**





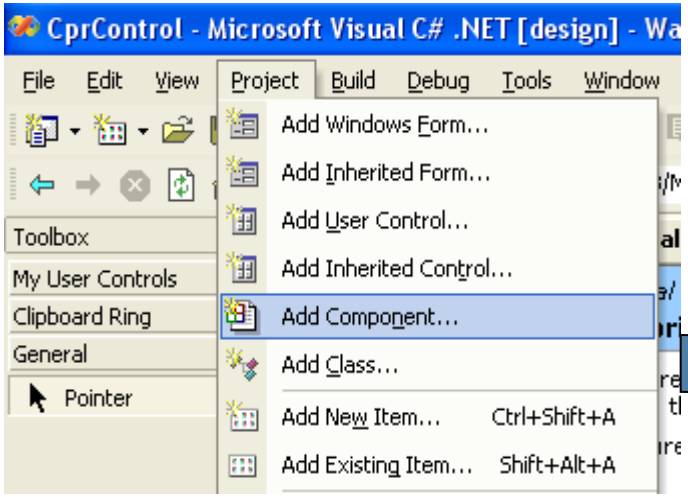
- Is all DLL-assemblies a component?
  - Yes, as they obey Szyperski's rules:
    1. Is a unit of independent deployment
    2. Is a unit of third-party composition
    3. Has no (externally) observable state

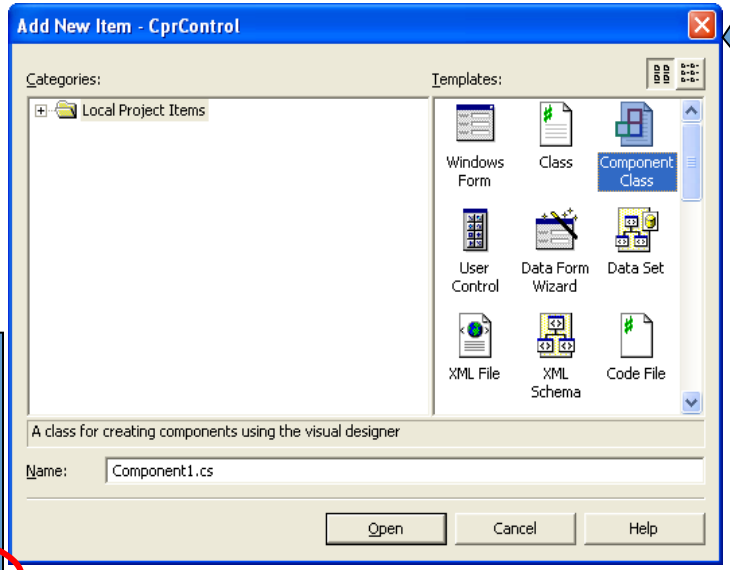
But to make them really fell like a component, the top level classes should inherit from System.ComponentModel.Component

Components how to build 2

- In Visual Studio you can chose Add Component in the Project menu.
  This give you a component (class) that can host other components.

- **Component** is the base class for all components in the common language runtime which marshal by reference.
  If you want to give a class design time capabilities (the component "feel"), just let the class inherit from
  `System.ComponentModel.Component`

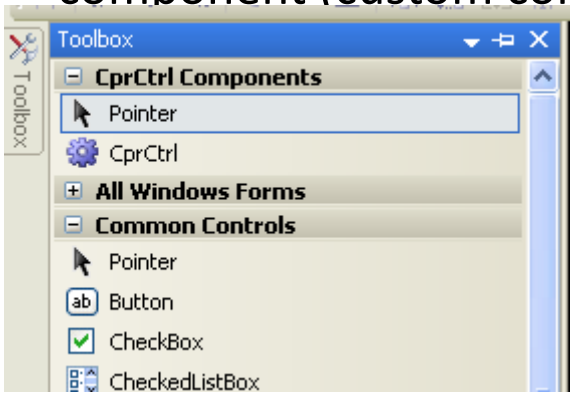




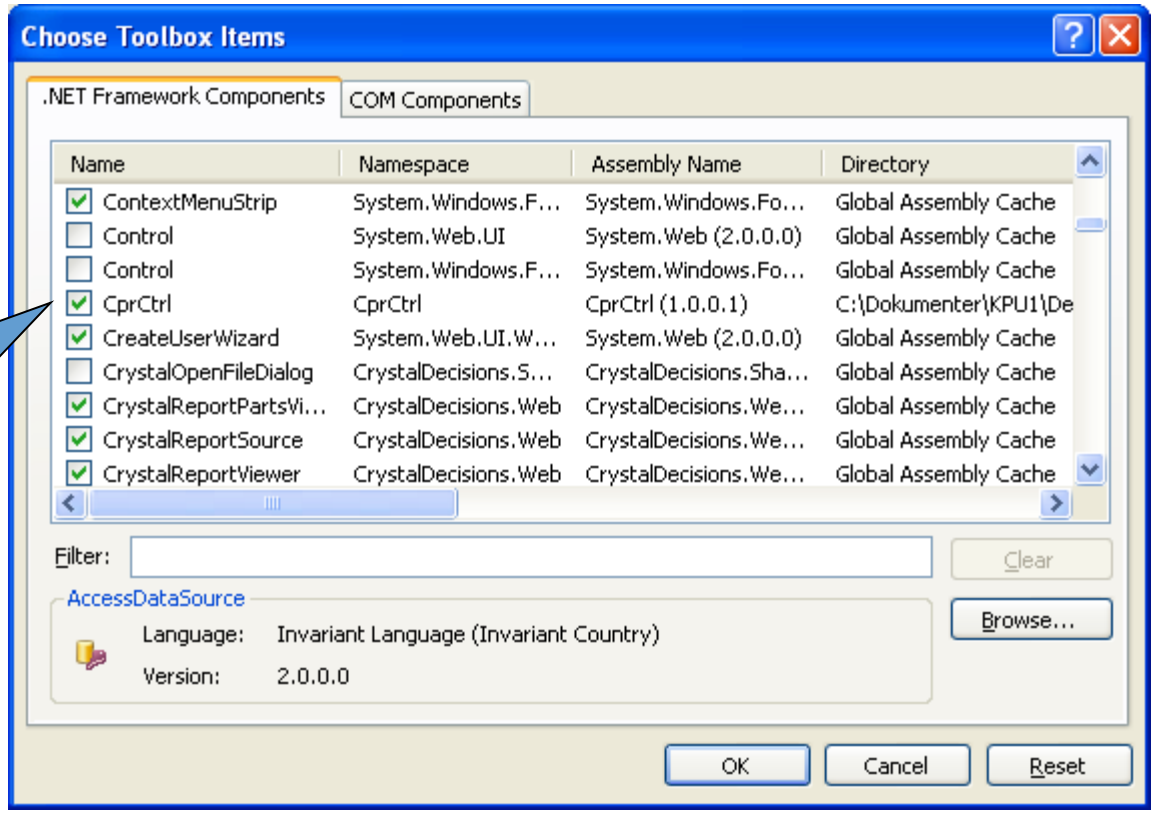

```
using System.ComponentModel;

namespace CprControl
{
  public class CprTest : Component
  {
     public CprTest()
```

How to add custom controls to the Toolbox?

- Right click on the Toolbox and select Choose Items…
- Press Browse in the dialog box and locate the assembly that contains the component (custom control) to add to the toolbox.



You can get your custom controls to show up under the .Net Framework Components tab by adding a key to the registry



Key: **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AssemblyFolders\Kpu1**
Value: *<a directory on your PC>*
(you must place a copy of your component in the referenced directory)

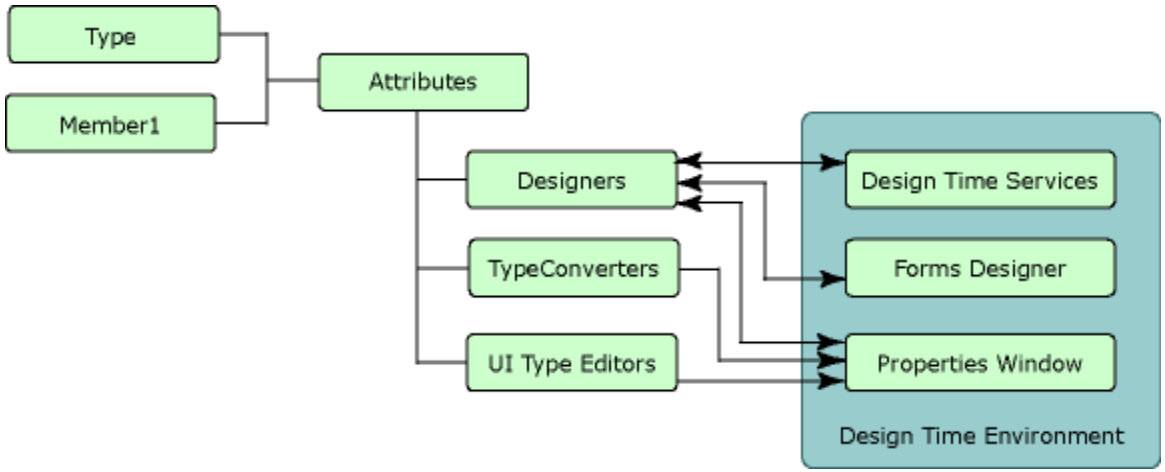
Component Naming Recommendations

The names you select for your classes — and for their properties, methods, and events — are one of the most important factors in how easy it is to use your component.

You can help the user of your component by following a few simple rules:

- Use complete words whenever possible — for example, SpellCheck. Abbreviations can take many forms, and hence can be confusing. If whole words are too long, use complete first syllables or carefully chosen abbreviations.
- Use mixed case for your class, method, and property names, capitalizing each word or syllable — for example, ShortcutMenus, or AsyncReadComplete.
- Use the same word or phrase your users would use to describe a concept. For example, you might have a SavingsAccount component that represented a savings account.
- Append "Collection" onto the correct name of the object contained by your collection — for example WorksheetCollection, FormCollection, or WidgetCollection.
- Use either the verb/object or object/verb order consistently for your method names. For example, placing the verb first results in names like InsertWidget and InsertSprocket, whereas placing the object first yields names like WidgetInsert and SprocketInsert.
- Do not repeat the name of the class in the method. For example, if you have a class named Book, do not have a method called Book.CloseBook. Instead, name your method Book.Close.

Design-Time Support



The .NET Framework provides interfaces and classes for customizing component behavior and user interfaces in a design-time environment. A design-time environment typically includes a forms designer for arranging components and a property browser for configuring the values of a component's properties. A design-time environment typically also provides design-time services that can be accessed and used by design-time mechanisms.

The .NET Framework defines interfaces that developers can use to implement customized design-time support. The primary mechanisms of extending design-time support fall within the following categories: designers, type converters, and UI type editors. Attributes are applied to types and type members to associate them with these design-time support providers.

References & Links

- Programming with Components How-to and Walkthrough Topics
  http://msdn.microsoft.com/en-us/library/ms171768.aspx
- http://msdn.microsoft.com/en-us/library/0ffkdtkf.aspx

- Design Guidelines for Developing Class Libraries
  http://msdn.microsoft.com/en-us/library/ms229042.aspx
  (Se også IDesign's kodestandard – findes i fildelingen under .Net extra)