MICROSOFT .NET

INTRODUCTION TO MICROSOFT'S .NET TECHNOLOGY

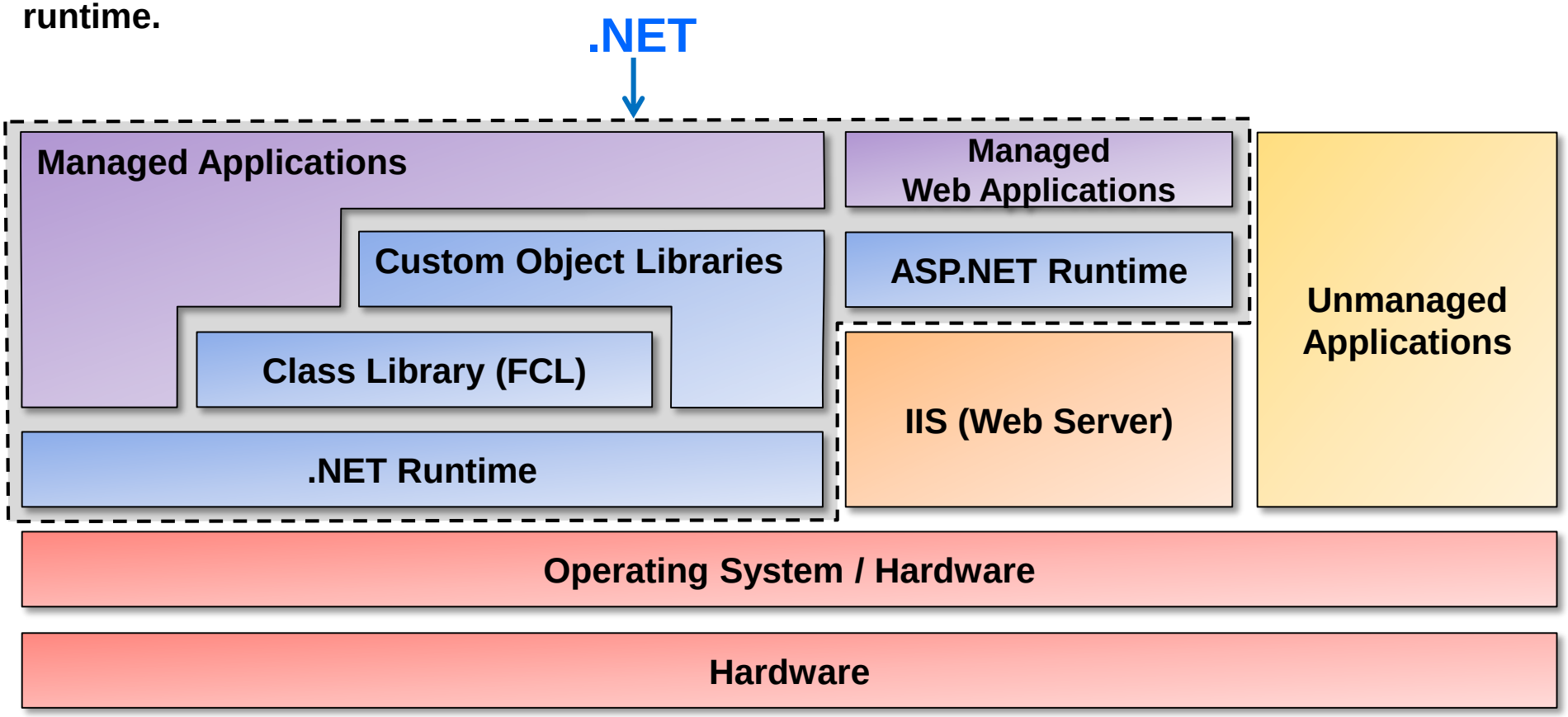
Peter R. Egli
INDIGOO.COM

Contents

1. What is .NET?

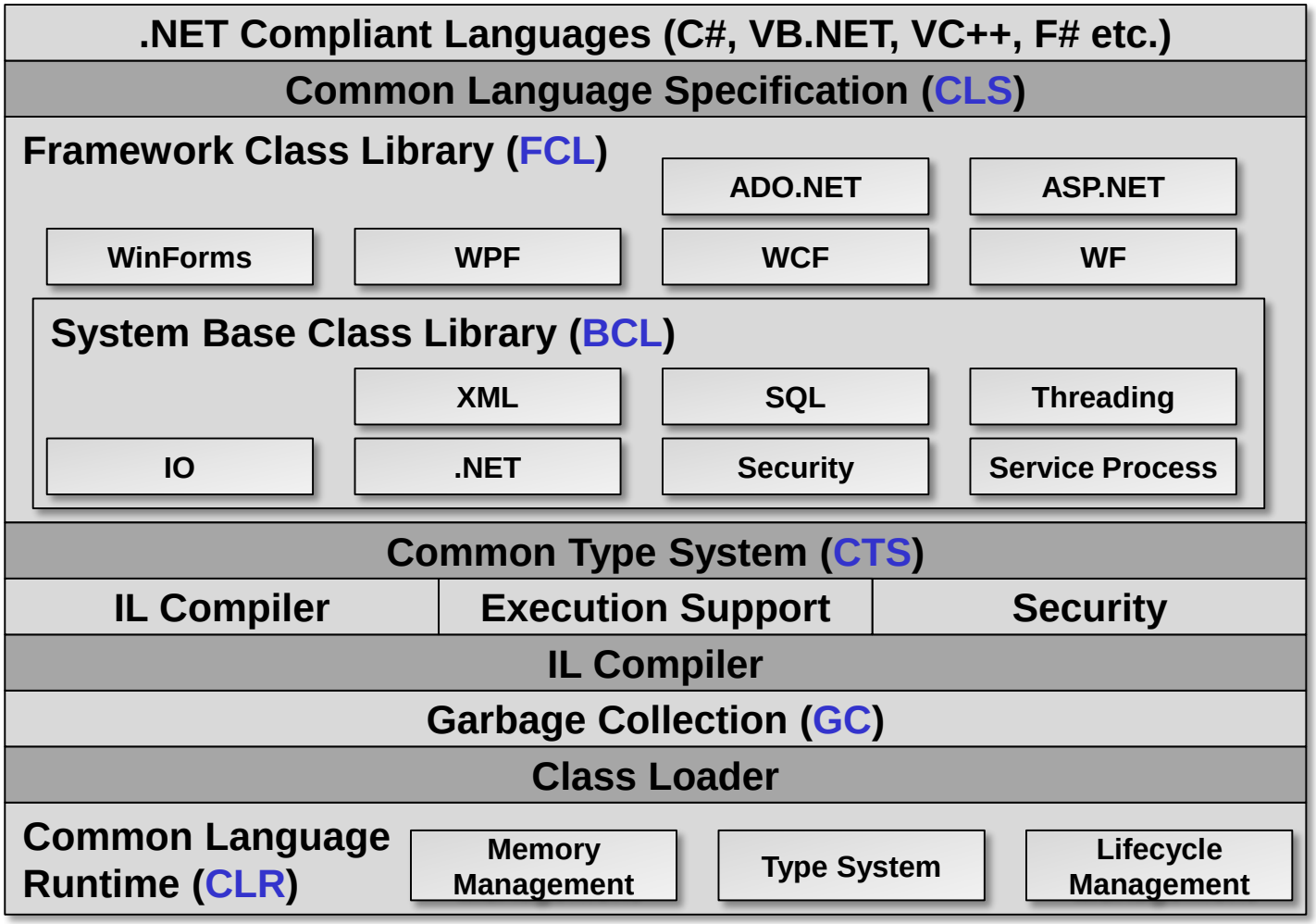
.NET introduces the concept of managed applications. Win32 applications are unmanaged, i.e. run natively on the Windows OS (WinXP, Win7, Win8 etc.).

.NET is an addition to the Windows OS. Managed applications run in the .NET runtime, i.e. the byte code is executed by a virtual machine which executes functions like type checks at runtime.

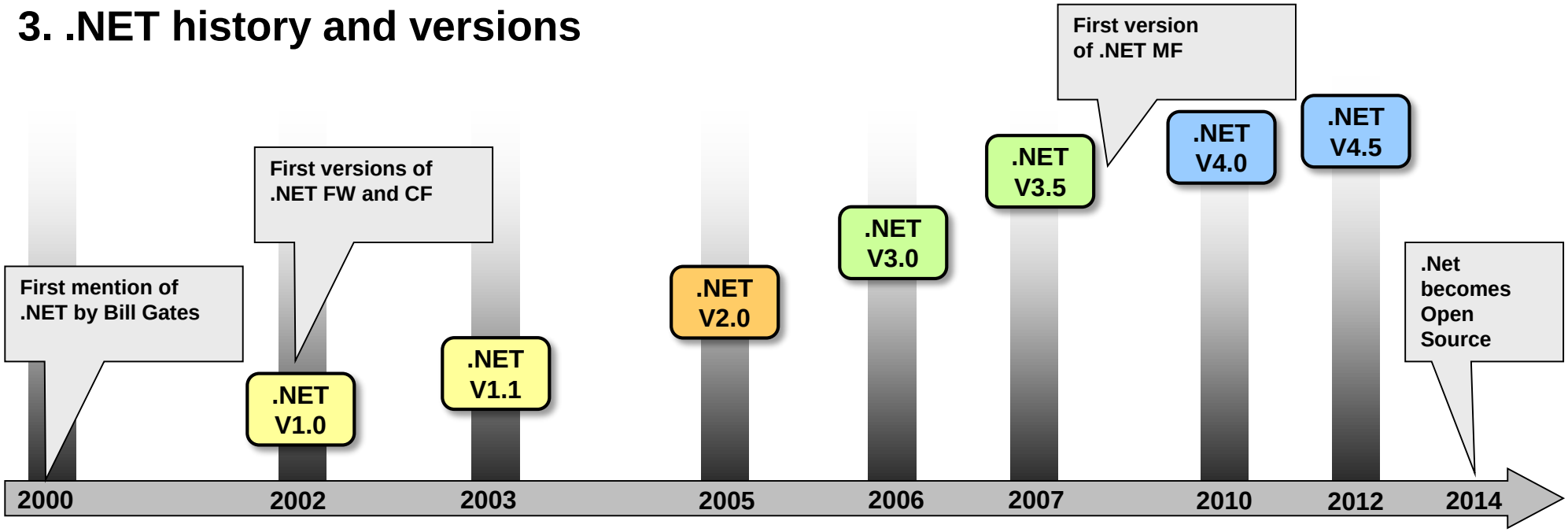


2. .NET platform overview

The .NET platform defines core services such as a type system as well as a wealth of libraries.



3. .NET history and versions



***.NET Framework*:**
Standard framework for desktop and server hosts.

***.NET Compact Framework*:**
Strippted-down .NET FW for embedded devices, based on Windows CE / Windows Embedded Compact.

***.NET Micro Framework*:**
Variant for „deeply“ embedded devices (no OS required, .NET is the OS). Open Source.

4. CLR - Common Language Runtime (1/3)

The common language runtime is the centerpiece of .NET.
It provides an abstracted and generalized interface for application code.

Multiple language support:
.NET supported multiple languages from the outset.
The languages supported by Microsoft are C#, F#, VB.NET and managed C++.
Third party vendors, however, may provide additional languages such as Fortran or Haskell.

Common type system:
A common type system allows writing parts of applications in different managed .NET languages (usually C#, VB, managed C++).
This is an improvement over ATL (Active Template Library) or COM (Component Object Model) technologies that required to write complicated wrappers to connect code written in different languages.
CTS also supports cross-language inheritance.

Garbage collection:
A garbage collector in the background transparently deallocates unused objects thus freeing the developer from allocation and deallocation duties.

4. CLR - Common Language Runtime (2/3)

Strong type checking, inspection / reflection:

Type checks at compile and runtime assure that types are compatible.
Reflection allows inspecting types at runtime.

Version checks:

Libraries are loaded by the CLR. On loading it performs version compatibility checks (find a compatible version of a required library).
The CLR also supports side-by-side execution (simultaneous execution of different versions of the same library).
In this respect, the CLR goes farther than the Java class loader which loads Jar-files based on the class path (location only).

Unified exception handling:

Exception handling is unified across the different languages.

Just In Time (JIT) compilation:

CLR is performance optimized due to JITing of CIL to native code. Performance can even exceed that of native C/C++ in certain cases.

Interoperability with COM:

The CLR supports interoperability with legacy COM-objects as well as calls to Win32 DLLs (native calls).

4. CLR - Common Language Runtime (3/3)

Architecture of .NET CLR:

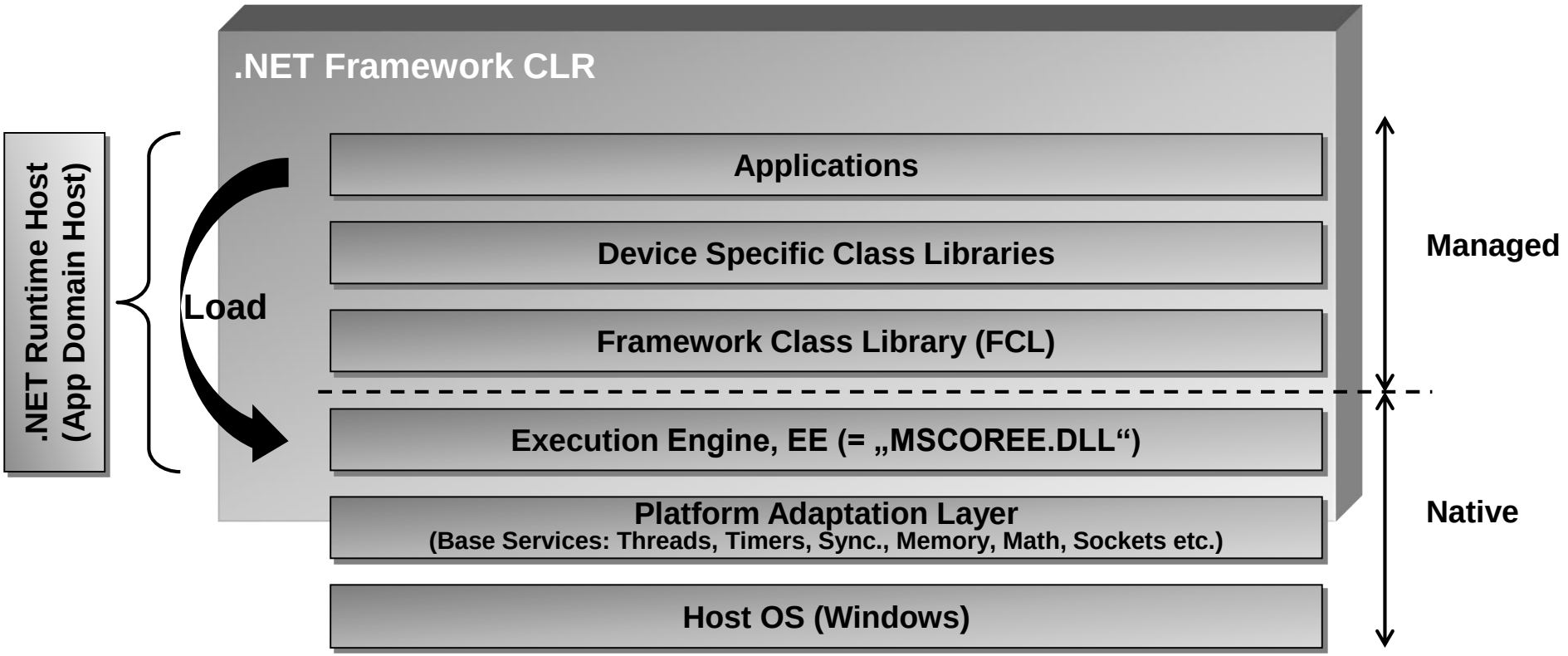
Applications are loaded and started by the .NET Runtime Host (aka App Domain Host).

1. Runtime host loads execution engine (EE) into memory

2. Runtime host loads application into memory

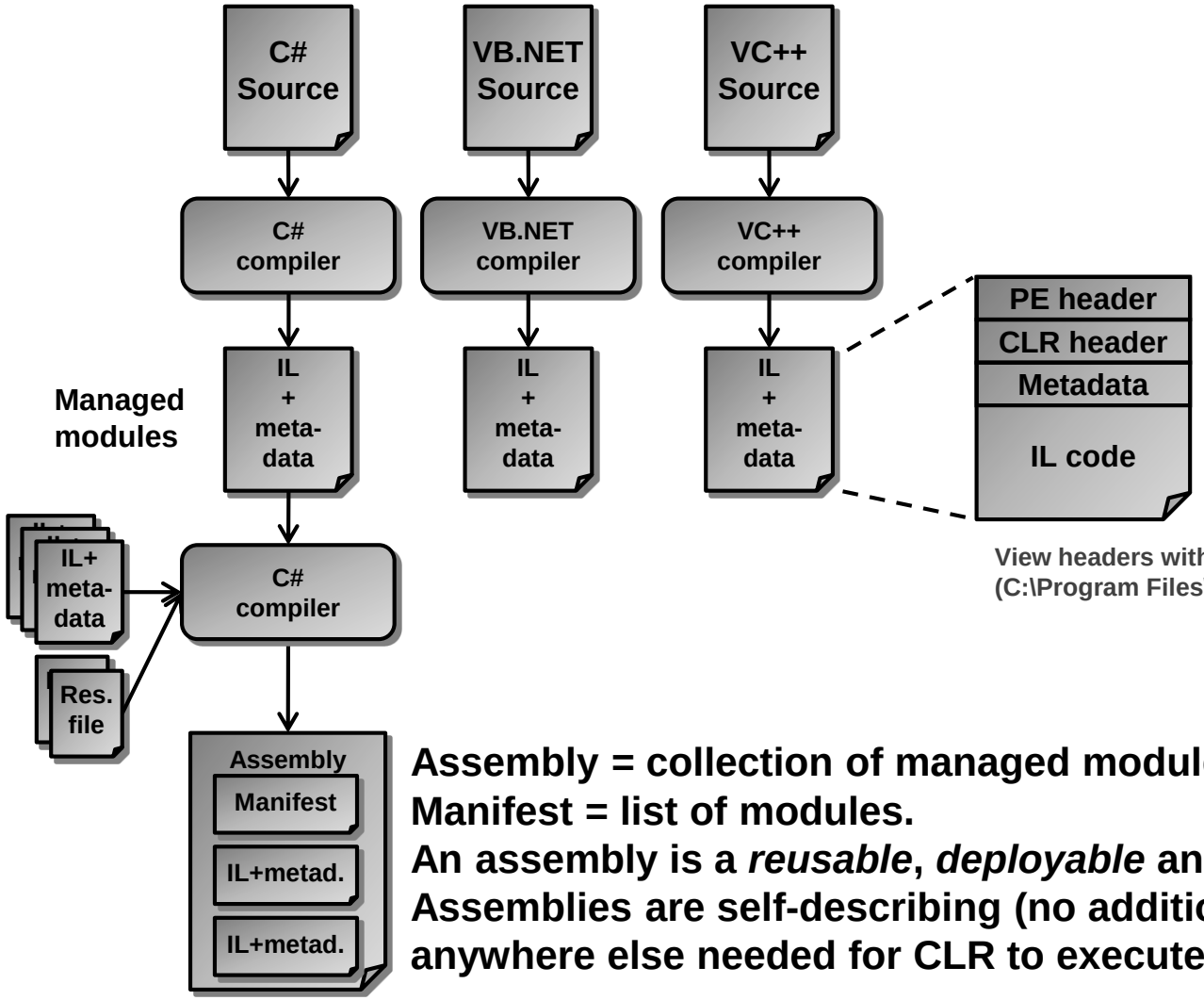
3. Runtime host passes the application to the CLR to be started

Application domain = Win32 process (execution environment of application, basically a container of application threads).



5. .NET framework code generation (1/2)

Code production in .NET is a 2-step process.



CUI: Command User Interface
IL: Intermediate Language
Res.: Resource
PE: Portable Executable
DLL: Dynamic Link Library

Type of file (DLL, GUI, CUI)
Version of CLR, resources, entry point etc.
Exported (provided) and imported (used) types

The code in IL format

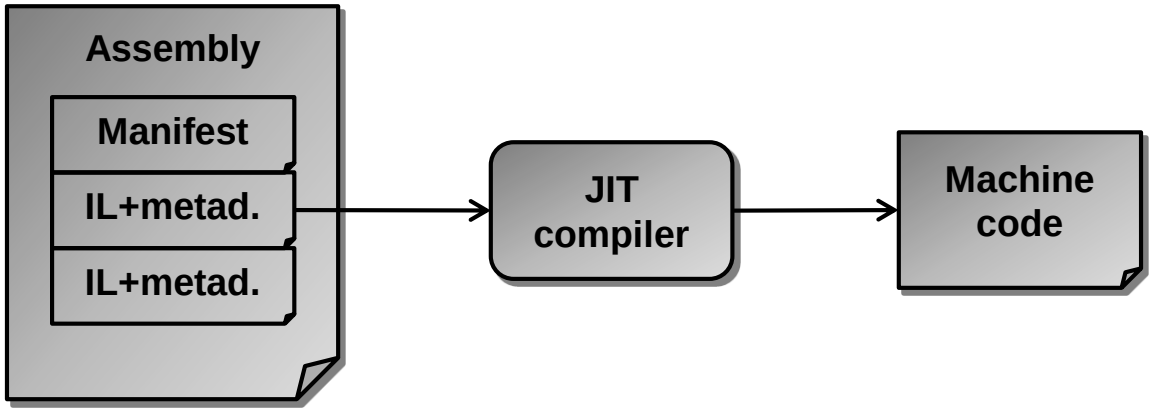
View headers with CIL Disassembler
(C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\ildasm.exe)

Assembly = collection of managed modules and resource files.
Manifest = list of modules.
An assembly is a *reusable*, *deployable* and *versionable* component.
Assemblies are self-describing (no additional info in registry or anywhere else needed for CLR to execute the assembly).

5. .NET framework code generation (2/2)

Just In Time compilation (JIT):

The Just In Time compiler of the CLR translates IL code into machine code.



→ The performance penalty of the translation at runtime applies only for first the call of a method. For subsequent calls, the CLR re-uses the JIT-compiled native method code.

→ During compilation the JIT compiler of the CLR performs a verification to ensure that the code is safe. Security checks comprise type safety checks, checks of the correct number of parameters etc.

→ JIT compiled code is potentially even faster than native code because the JIT compiler uses runtime information for the compilation / optimization.

6. CTS – Common Type System (1/3)

CTS is the type foundation for the .NET framework.
CTS defines that:

→ A type may contain zero or more members:
Member = field | method | property | event

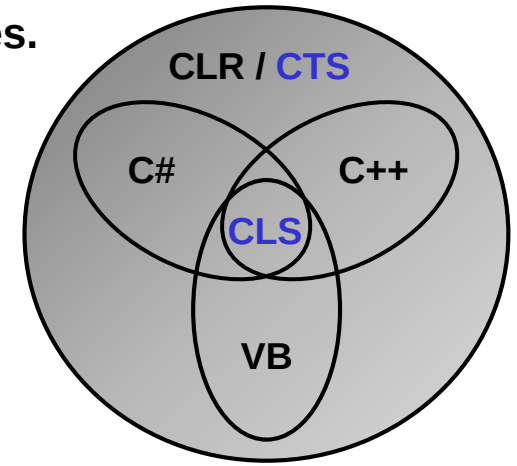
→ Rules for type visibility:
public, protected, private, internal (=visible within assembly only)

→ Type inheritance:
E.g. only single inheritance supported, thus C++ compiler will complain if a managed C++ class derives from multiple base classes.

→ Common base type:
All types inherit from the „mother of all types“ = System.Object (everything is an object).

CTS defines a standardized type system common to all .NET languages.

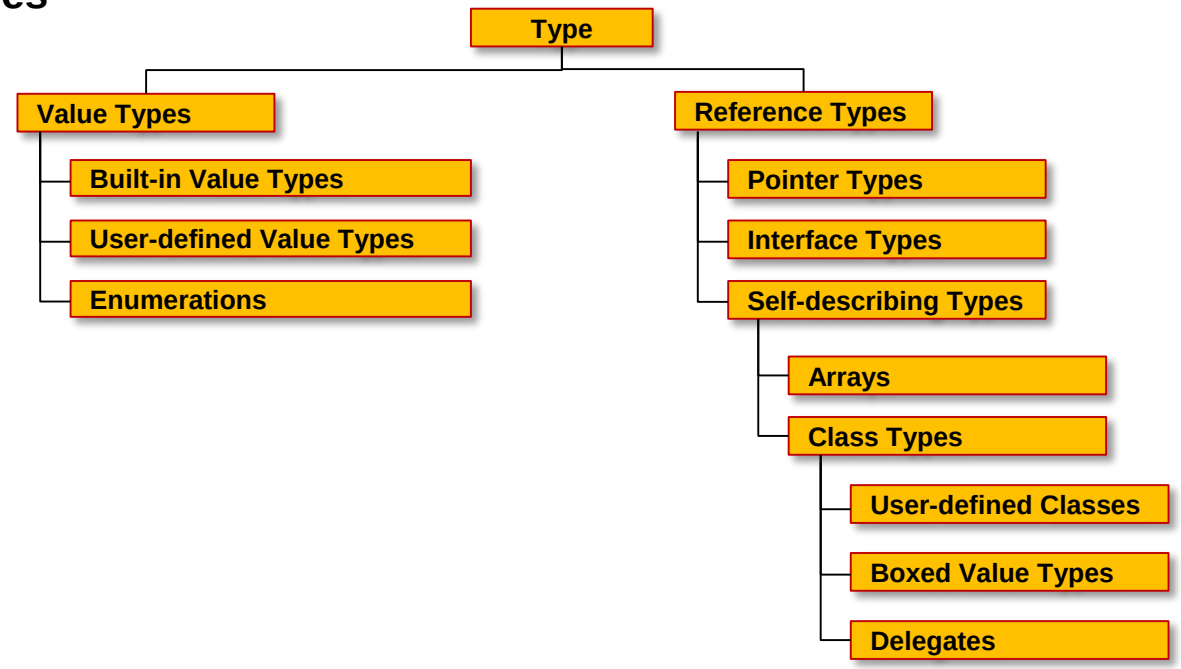
CLS (Common Language Specification) is the basis for interworking of code written in different (managed) .NET languages.
CLS is the base set of language features that should be supported by a .NET language thus allowing to use a type written in one language in another language.
CLS applies to public members („public“ and „protected“) only.
Private members are not affected by CLS.



6. CTS – Common Type System (2/3)

CTS type hierarchy:

The type system comprises value types and reference types.

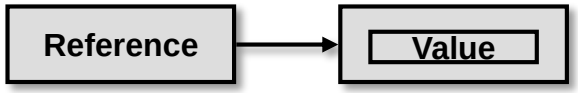


Value types:

Value types directly contain their data.

Reference types:

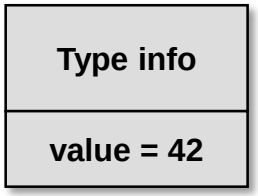
Reference types store a reference to the value‘s memory address.



6. CTS – Common Type System (3/3)

Boxing:

Every value type has a corresponding reference type called «boxed type».
Some reference types have a corresponding unboxed value type.



Boxed (value is wrapped in an object)

Unboxed

CTS built-in types:

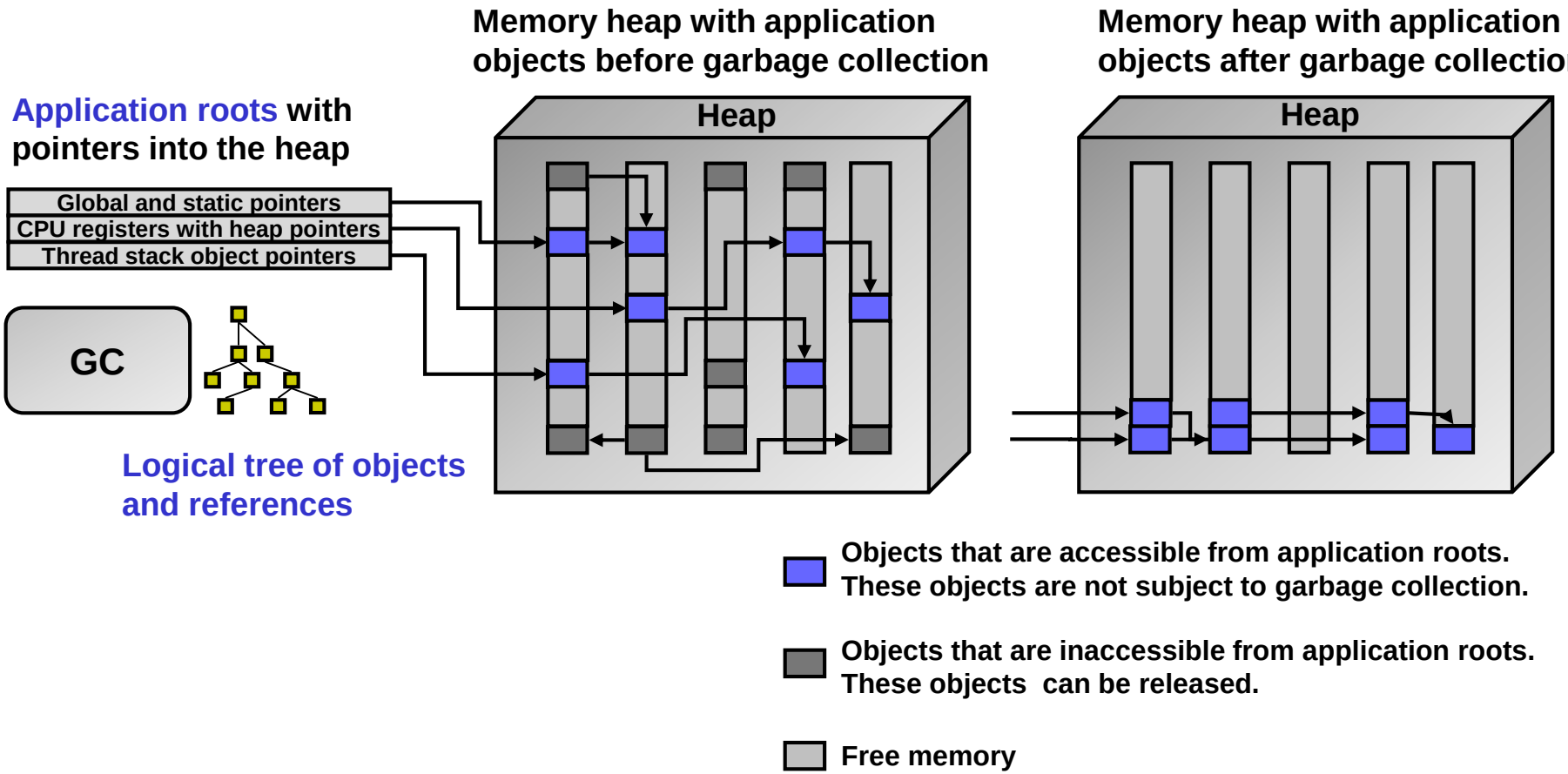
- `bool`
- `char`
- `int8, int16, int32, int64`
- `unsigned int8, unsigned int16, unsigned int32,`
- `unsigned int64`
- `native int, native unsigned int`
- `float32, float64`
- `System.object, System.string`

7. .NET garbage collection (1/4)

Garbage collection procedure:

The garbage collector periodically sweeps heap memory and reclaims unused objects thus offloading the application programmer from memory management.

N.B.: Garbage collector avoids memory leaks, but „Out of memory“ exceptions may still occur!



7. .NET garbage collection (2/4)

GC phases:

Phase 1: Mark

→ Find and mark garbage-collectable objects.
1. GC assumes all objects on heap to be garbage.
2. GC identifies application roots (static object pointers, registers etc.).
3. GC walks the roots and builds a tree of objects reachable from roots (=live objects).

Phase 2: Relocate

→ Relocate non-collectable objects.
GC adjusts pointers in (non-collectable) objects to other (non-collectabe) objects.

Phase 3: Compact

→ Move non-collectable objects, free collectable objects.
1. GC walks through the heap linearly marking all garbage objects.
2. GC shifts non-garbage objects down in memory thus removing gaps in the heap.
3. Frees garbage objects (adds them to the finalizer queue first).

N.B.: GC needs to freeze all threads to protect the heap (no accesses to heap during GC, «Stop the world»).
The GC execution duration depends on the number of objects.
Thus the GC execution is *undeterministic*.

7. .NET garbage collection (3/4)

Triggers of .NET FW GC:

1. Low physical memory
2. Objects allocated on the managed GC heap exceed a threshold
3. `GC.Collect()` called

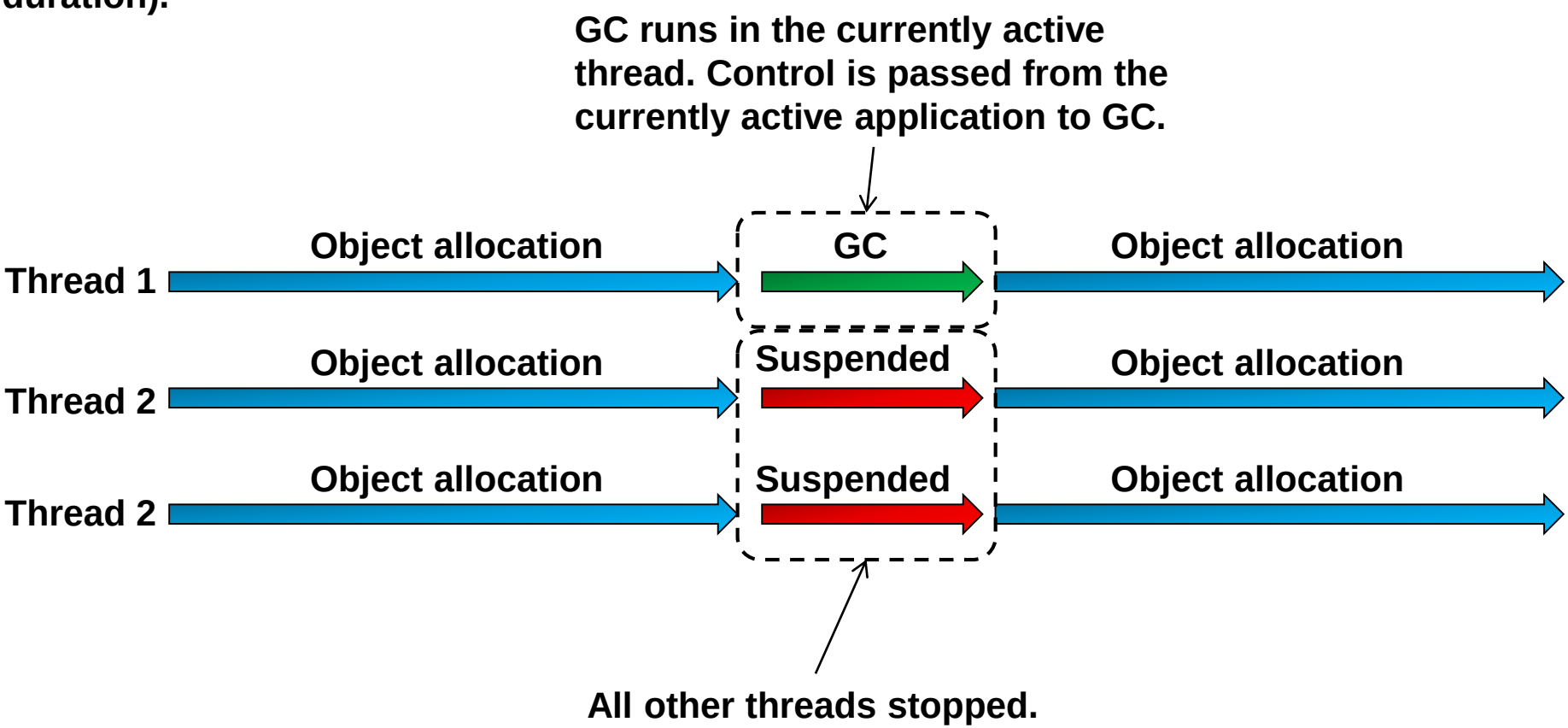
Generational GC:

.NET FW implements a generational GC.
Objects that are not collected in the current GC run in the current generation are elevated to a higher generation.
Higher generation garbage collection is run less frequently.
Purpose: Identify long-lived objects and avoid frequent GC on these objects thus increasing the performance.
Generation 0: Youngest objects (short-lived objects such as temporary variables)
Generation 1: Short-lived objects, buffer between generation 0 and generation 2 objects.
Generation 2: Long-lived objects such as objects with static data that live as long as the application process exists.

7. .NET garbage collection (4/4)

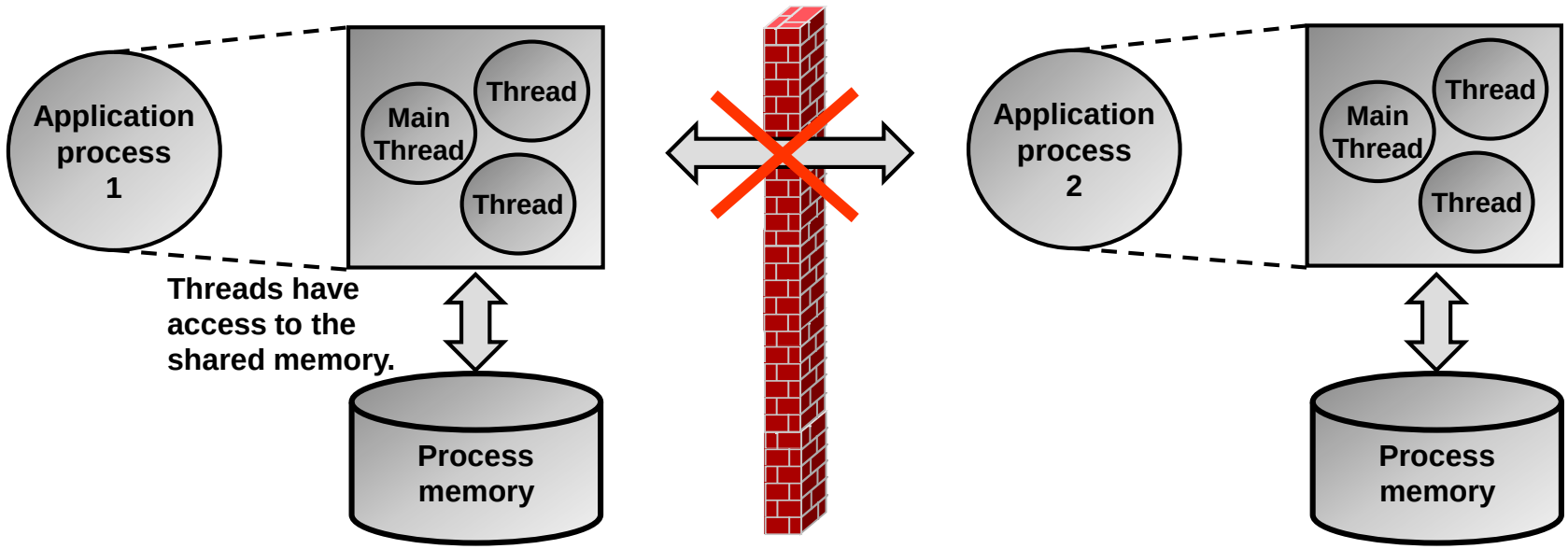
Garbage collection thread:

GC is run in the (application or system) thread that is active when the GC kicks in. This means that thread execution time is undeterministic (GC runs for an undeterministic duration).



8. .NET application domains (1/2)

Applications usually run in a process and consist of 1 or multiple threads.



The operating system provides isolation between the processes.
The memory is mutually protected through the MMU (threads in application processes can not access the memory in the other process).
Communication between the processes is restricted to IPC (sockets, pipes, mapped files etc.).

☺ **Good protection**

But:

☹ **Processes are heavy-weight (costly to create)**

8. .NET application domains (2/2)

.NET 2.0 introduced the concept of application domains.

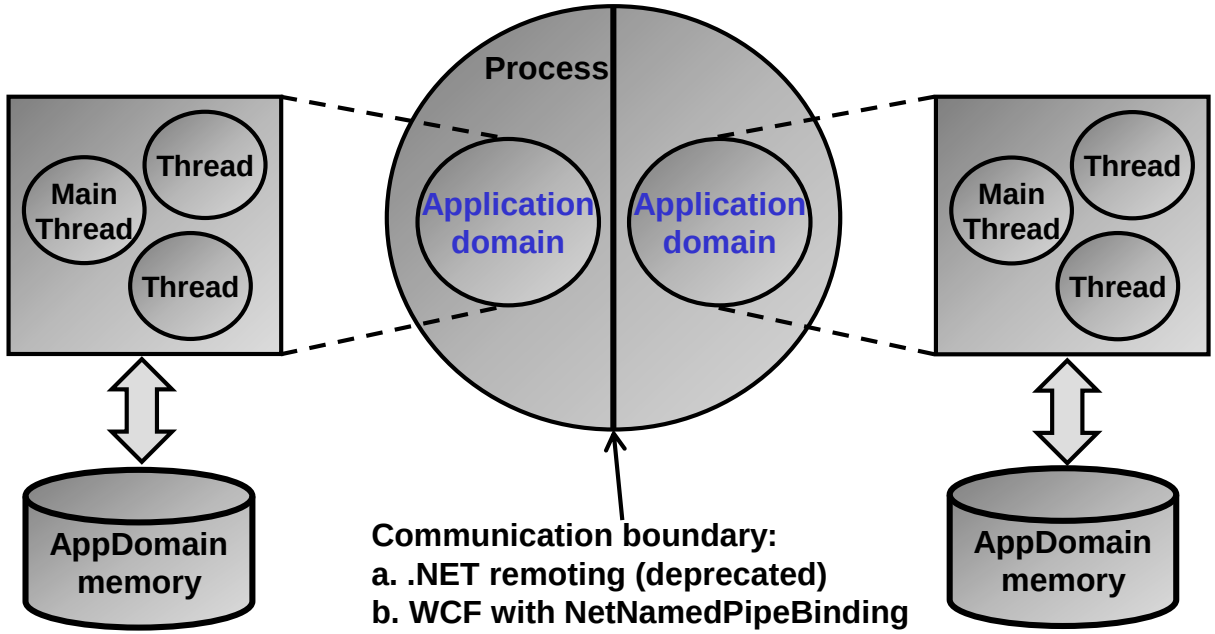
→ Applications are isolated through application domains (no direct access of code or memory from domain to domain).

→ CLR runs as a process containing 1...n application domains.

→ Multiple threads may run in a single application domain.

→ An application domain is similar to a mini-OS (helps avoid use of costly processes for isolation).

- **Protection**
- **Lightweight (only 1 process to be created)**



9. .NET assemblies (1/5)

An assembly is a file that contains executable code and resources.
An assembly is either a DLL (.dll extension) or an executable (.exe extension).
Java equivalent: Jar-file.

Problems with Win32-DLLs ("DLL hell"):

- **Win32 DLL must maintain backward compatibility to ensure that existing depending applications do not break when updating the DLL. This is difficult to achieve.**
- **No way to enforce compatibility between DLL version and referencing application.**

Improvements brought with Win2K:

- **A DLL can be placed into an application's folder thus ensuring other applications using different versions of the DLL do not break.**
- **Locking of DLLs placed in System32 folder so that they cannot be overwritten.**

Solution introduced with .NET:

- **Versioning scheme: Every assembly has a version and a manifest defines that what version of other assemblies an assembly requires.**
- **Version checks performed on loading an assembly.**

9. .NET assemblies (2/5)

Assembly functions:

Assemblies fulfill the following functions:

1. Versioning (name and version uniquely identifies an assembly)

2. Security (types are fully contained in an assembly, assembly can be cryptographically signed)

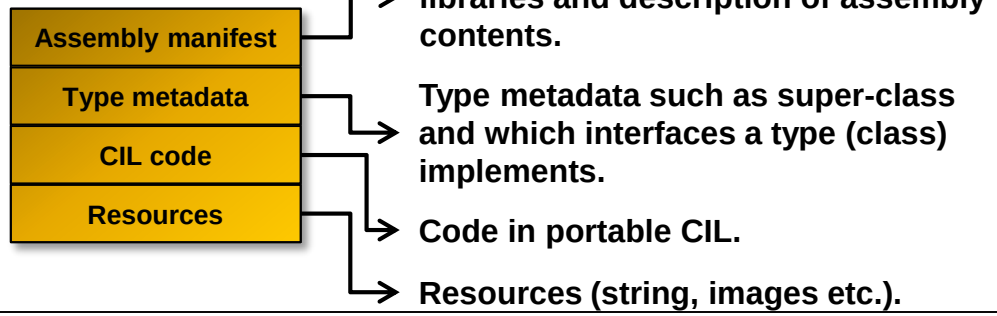
3. Deployment (assemblies are machine- and OS-independent deployment units for executable code)

Assembly contents and structure:

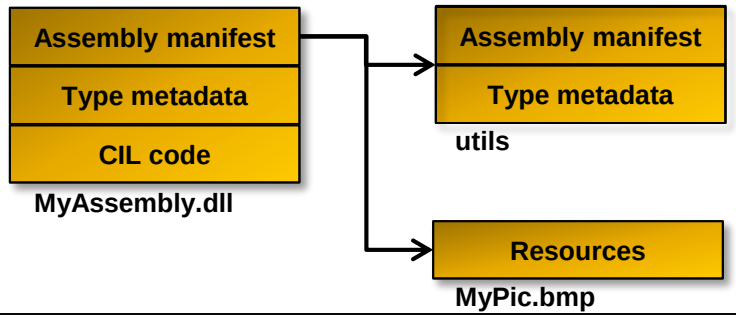
Assemblies are self-describing in that they contain a manifest and meta-data that describes their contents.

Assemblies may be either single-file (common case) or spread over multiple files (applicable for scenarios where multiple teams work on a large program or library at the same time).

Single-file assembly



Multiple-file assembly



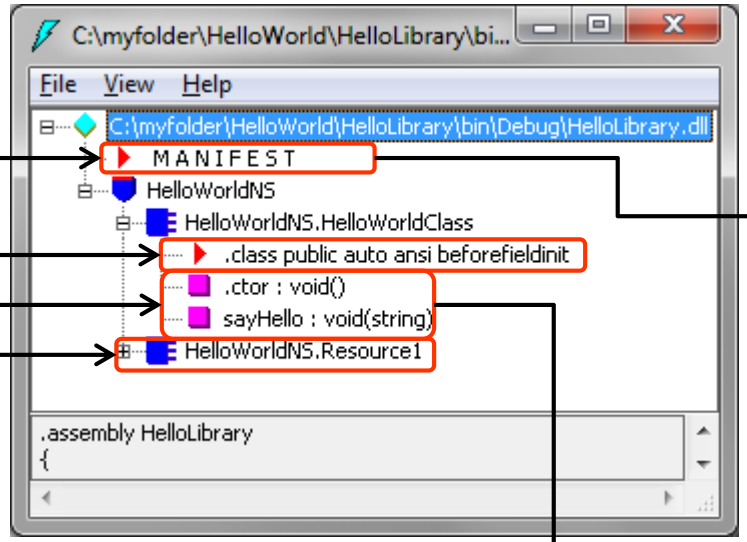
9. .NET assemblies (3/5)

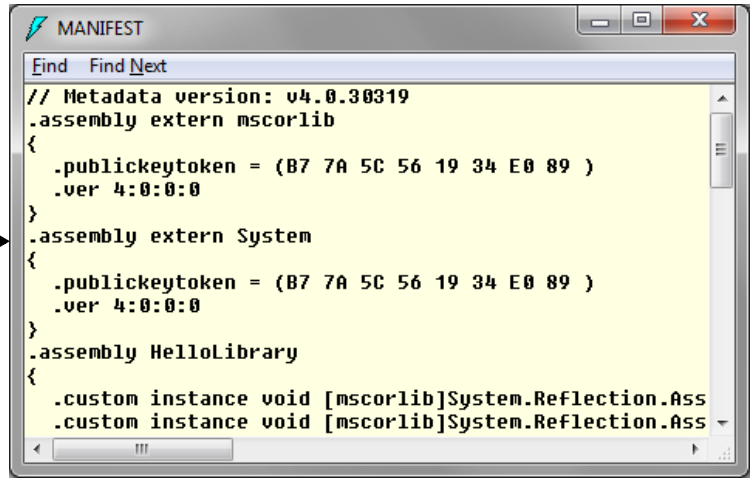
Assembly contents and structure:

The contents of an assembly can be disassembled with Microsoft SDKs/Windows/v7.0A/Bin/ildasm.exe.

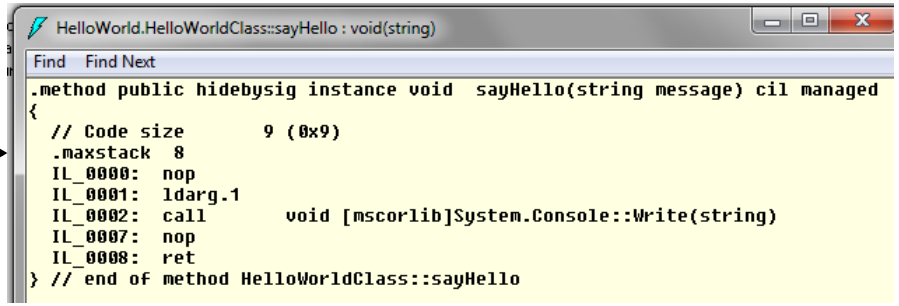
HelloWorld.exe

- Assembly manifest
- Type metadata
- CIL code
- Resources





Required external assembly and version.
Assembly meta-information.



CIL code.

9. .NET assemblies (4/5)

Private assembly and shared assembly:

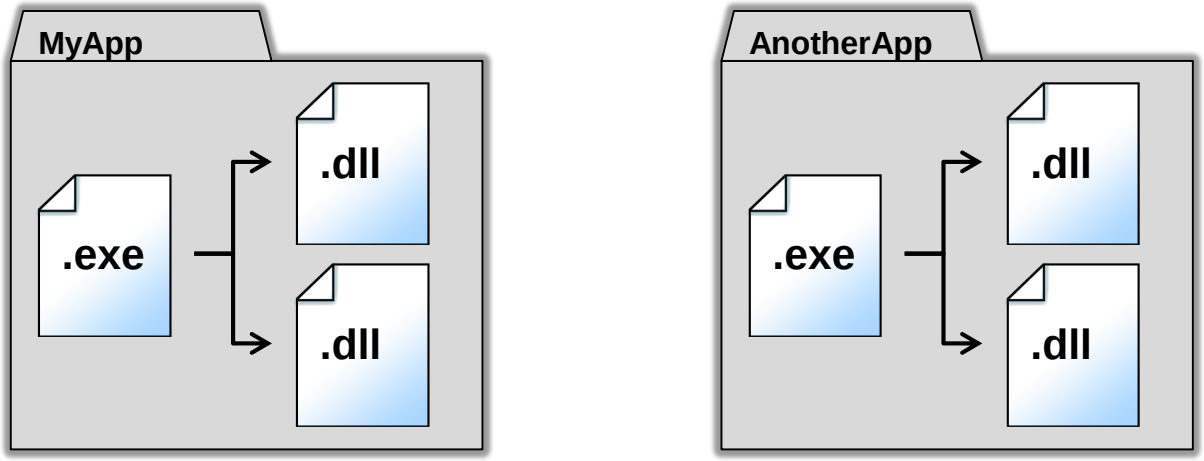
Usually assemblies are stored in the application's folder (private assembly). In order to conserve space, it is possible to make assemblies available for other applications by storing them in the GAC (shared or global assembly).

Private assembly (default):

→ Stored in application's folder.

→ CLR assumes that assemblies are compatible and does not perform version checks on loading assemblies.

→ The compatibility is meant to be achieved by always deploying the entire application including all dependencies (required assemblies) by XCOPY-deployment (just copy the assemblies, do not register them in the system).



9. .NET assemblies (5/5)

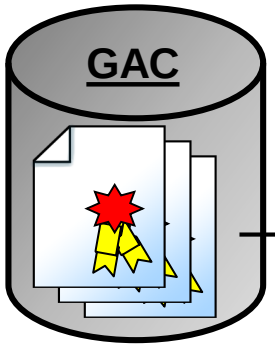
Private assembly and shared assembly (2/2):

Shared assembly:

→ Stored in Global Assembly Cache (GAC) (C:\Windows\assembly)

→ Identification of assembly through strong name = assembly name + version number + culture information (if available) + public key and digital signature. This combination of information guarantees uniqueness of an assembly name.

→ The CLR checks and verifies the strong name on loading the assembly.



Version = <major version>.<minor version>.<build number>.<revision>

Assembly name = user defined name

Assembly Name	Version	Culture	Public Key Token	Processor Architecture
Microsoft.ExceptionMessageBox	9.0.242.0		89845dcd8080cc91	MSIL
Microsoft.ExceptionMessageBox	10.0.0.0		89845dcd8080cc91	MSIL
Microsoft.GroupPolicy.AdmTmplEditor	6.1.0.0		31bf3856ad364e35	x86
Microsoft.GroupPolicy.AdmTmplEditor	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.GroupPolicy.AdmTmplEditor.Resources	6.1.0.0	en	31bf3856ad364e35	x86
Microsoft.GroupPolicy.AdmTmplEditor.Resources	6.1.0.0	en	31bf3856ad364e35	AMD64

Culture = RFC1766 language identifier (optional)

Key token (signature)

10. Overview of .NET components / libraries (1/6)

ASP.NET:

Web application framework for building dynamic web pages.
ASP.NET is the successor to ASP technology (Active Server Pages).
ASP.NET allows inlining code and declaring code behind that and fills a part of a web page.

Inline model:

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "---//W3C//DTD XHTML 1.0  //EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    label1.Text = DateTime.Now.ToLongTimeString();
  }
</script>

  <div>
    The current time is: <asp:Label runat="server" id="label1" />
  </div>
```

Code and HTML markup are contained in the same file (Default.aspx).

Code behind model:

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="WebApplication2._Default" %>
```

Default.aspx

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    //Do stuff here...
  }
}
```

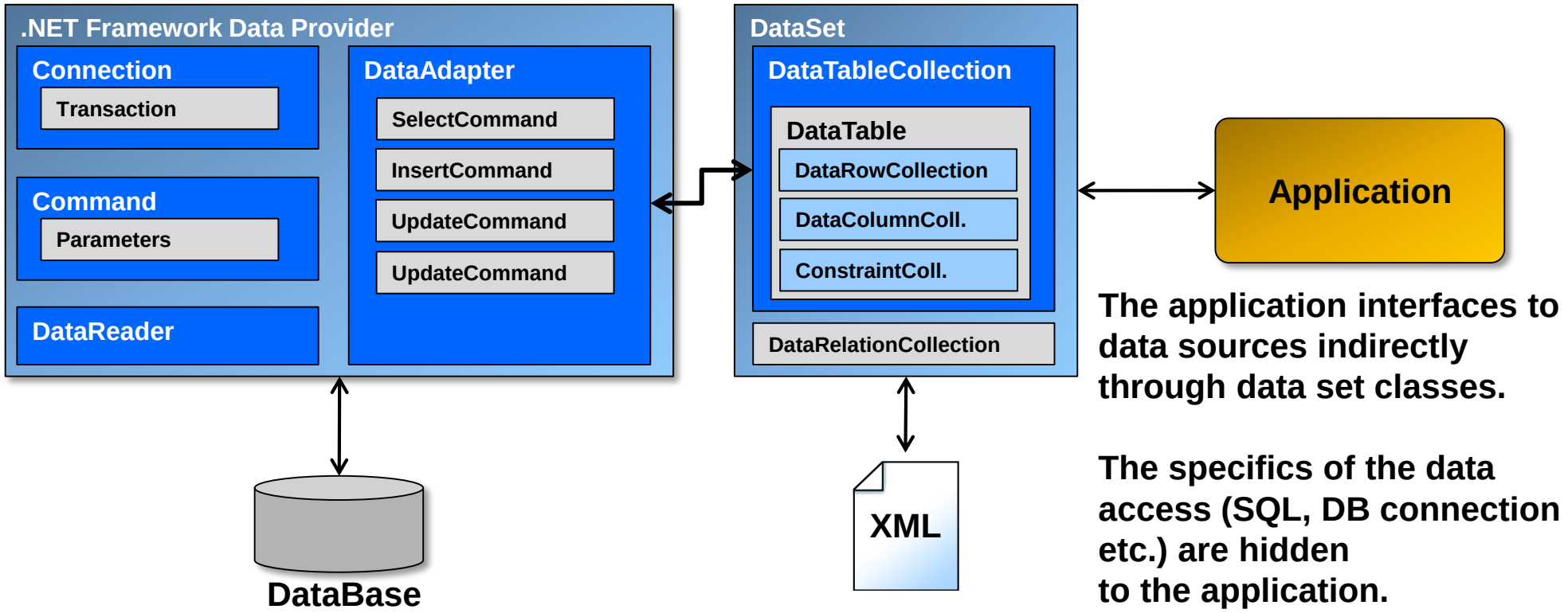
Default.aspx.cs

10. Overview of .NET components / libraries (2/6)

ADO.NET:

Classes for consistently access data sources such as relational DBs (SQL) and XML-based data.

The main components of ADO.NET are data providers that provide access to data sources and data sets that provide a generic application interface (API) for accessing the data independently of the actual data source.



The application interfaces to data sources indirectly through data set classes.

The specifics of the data access (SQL, DB connection etc.) are hidden to the application.

10. Overview of .NET components / libraries (3/6)

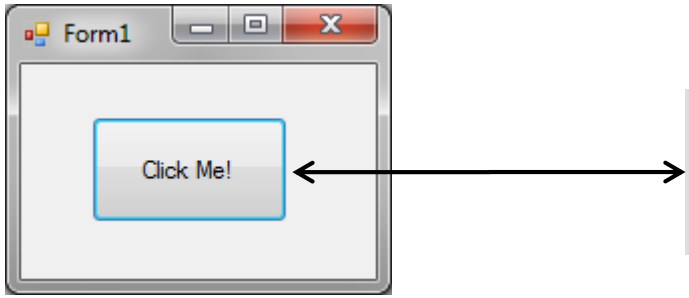
WinForms:

WinForms is the most widely used GUI technology.

WinForms are somewhat limited in their graphical capabilities (color gradients, blending effects etc.) so a newer technology named WPF was introduced (see below).

WinForms pros and cons:

- **Low entry level, simple to use.**
- **No strict MVC model (model – view – controller).**
- **No strict separation of business logic and UI code.**



Event method containing event handling code

```
private void button1_Click(object sender, EventArgs e)
{
  HelloWorldNS.HelloWorldClass hw = new HelloWorldClass();
  hw.sayHello("Hello world");
}
```

10. Overview of .NET components / libraries (4/6)

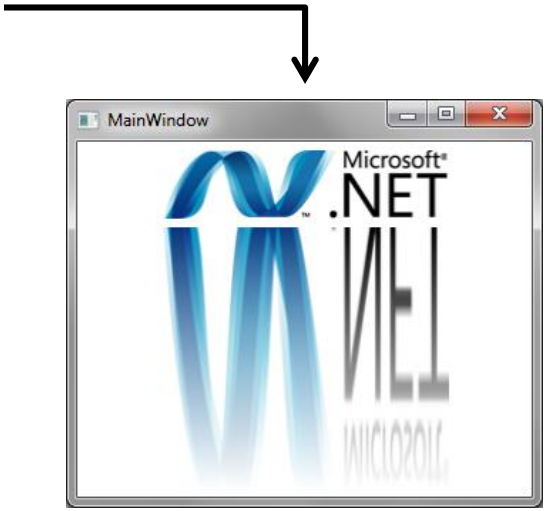
WPF (1/2):

WPF was introduced with .NET FW 3.0 as a successor to WinForms providing enhanced graphical capabilities and a declarative GUI definition model with XAML.

- **Enforce separation of business logic and GUI logic**
- **WPF uses declarative XAML (eXtensible Application Markup Language) language to describe the GUI**
- **WPF supports 3 different data bindings of UI controls to data sources (one time, 1-way, 2-way)**

XAML

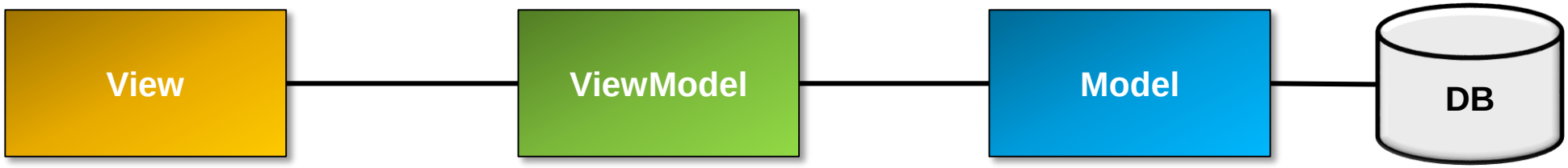
```
<Window x:Class="HelloWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <StackPanel>
        <Border BorderBrush="White" BorderThickness="5" HorizontalAlignment="Center" VerticalAlignment="Center">
            <Image Source="file:///C:/myfolder/HelloWorld/HelloWPF/net_logo.jpg" Width="200" Height="50" Stretch="Fill"
        </Border>
        <Border Width="203" Height="231">
            <Border.Background>
                <VisualBrush Visual="{Binding ElementName=myImage}">
                    <VisualBrush.Transform>
                        <ScaleTransform ScaleX="1" ScaleY="-1" CenterX="0" CenterY="100"></ScaleTransform>
                    </VisualBrush.Transform>
                </VisualBrush>
            </Border.Background>
            <Border.OpacityMask>
                <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                    <GradientStop Offset="0" Color="Black"></GradientStop>
                    <GradientStop Offset="0.8" Color="Transparent"></GradientStop>
                </LinearGradientBrush>
            </Border.OpacityMask>
        </Border>
    </StackPanel>
</Window>
```



10. Overview of .NET components / libraries (5/6)

WPF (2/2):

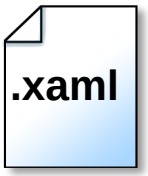
Microsoft promotes the MVVM (Model View ViewModel) model to connect UI with business logic or a data access layer.
MVVM is very similar to the MVC (Model View Controller) pattern.



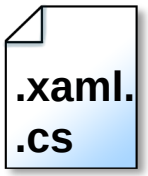
The view defines the graphical properties and how the data is presented to the user.

The ViewModel connects the view with model objects.

Class objects holding or providing access to data.
May also be pure business logic objects.



XAML code to describe the view.



Optionally there may be some code behind that does pure UI logic (rendering).

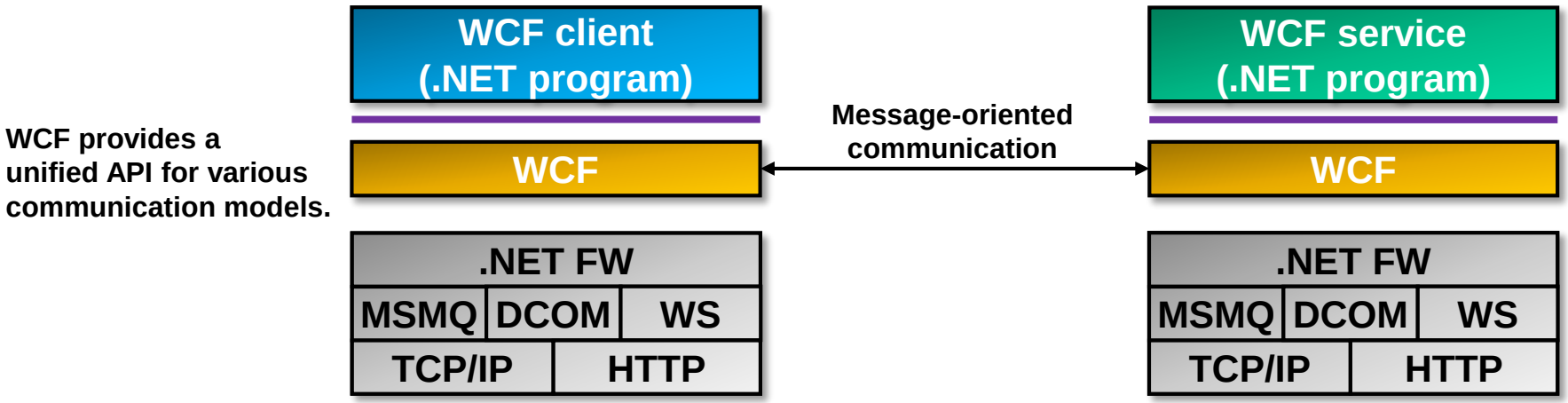
Core code behind is in ViewModel.

Simple data access objects without business logic.

10. Overview of .NET components / libraries (6/6)

WCF:

WCF is the successor to various communication technologies and introduces a common message-based communication model.
WCF is service-oriented and clearly separates interface (contract) from address and binding (transport protocol to be used).



LINQ:

LINQ (Language Integrated Queries) is a new language-construct that allows placing SQL-like queries directly into source code (internal DSL – Domain Specific Language).

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var lowNums =
    from n in numbers
    where n < 5
    select n;
```

11. .NET versus Java

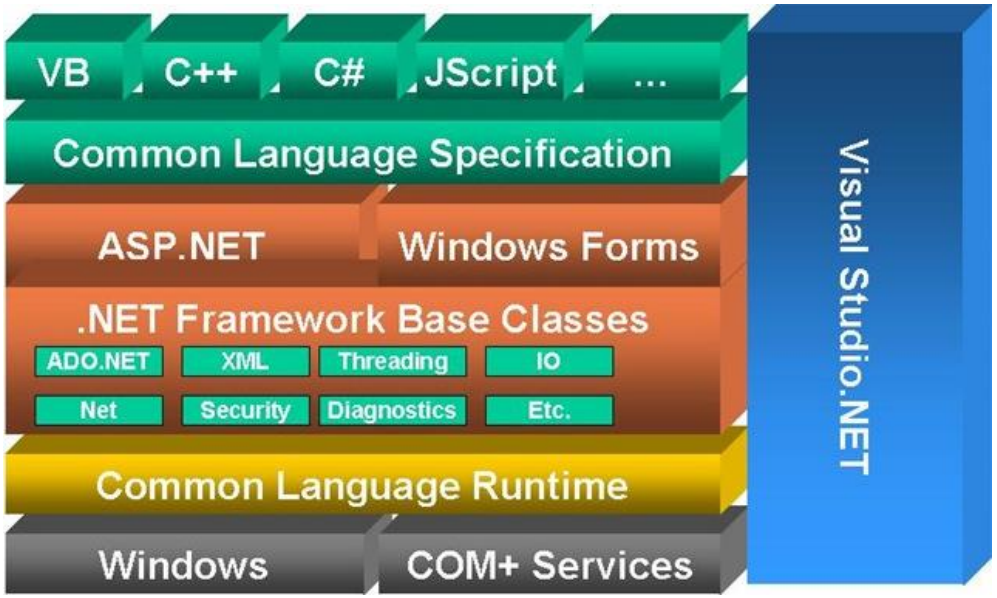
Both .NET and Java share similar concepts.

Differentiating features of .NET:

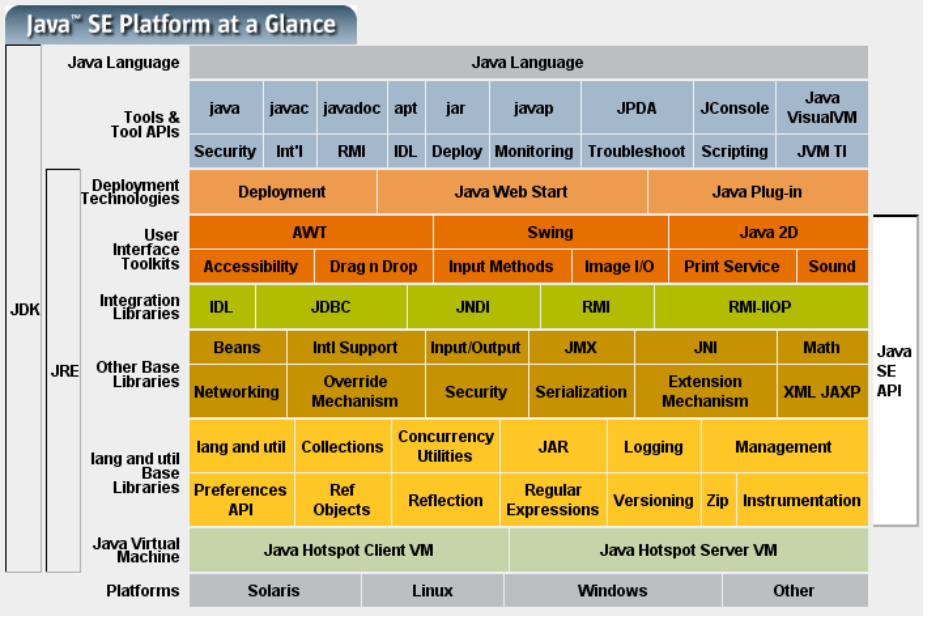
- **Multi-language**
- **Better integration of native code into managed code (use of „unsafe sections“)**
- **Faster specification process; Java is partly developed by the community (JCP – Java Community Process) which tends to be slower than a process driven by a company with commercial interests.**

But:

- **.NET is mainly limited to Microsoft OS (there are .NET clones such as Mono, but the legal situation is unclear).**



Source: Microsoft



Source: Sun Microsystems

12. .NET programming guidelines (1/2)

Programming managed code (C#) is not fully transparent to the programmer.
Some guidelines need to be observed to achieve sufficient performance.

Recycle expensive resources:
Expensive resources such as threads should be recycled (use a thread pool that is initialized at process start).

Lazy initialization:
Initialize lazily (load and initialize things just before usage, do not load everything at application start).

Use background workers:
Do work in the background to improve the perceived performance.

Use object allocation with care:
Use object allocation only when needed, recycle objects (frequent object allocation / deallocation results in longer GC periods).

Unmanaged resources:
The lifecycle of unmanaged resources is still under the responsibility of the developer.
Use patterns like the Dispose-pattern to safely de-allocate unmanaged resources (see below).

12. .NET programming guidelines (2/2)

Finalization pattern:

Problem with GC:

`Finalize()` is called as part of the GC process by a background thread. This means that the `Finalize()` on an object is undeterministic (it is unkown when `Finalize()` is called).
If precious unmanaged resources are only de-allocated in `Finalize()` they may be freed late or never.
The `Dispose()` pattern allows the user of an object to free these resources. If forgotten the CRL will eventually free the unmanaged resources by calling `Finalize()`.

N.B.: A class can have either a destructor (`~MyClass()`) or a `Finalize()` method. The compiler automatically creates a `Finalize()` method from a destructor if one is available.

Example:

```
// Design pattern for a base class.
public class Base: IDisposable {
  //Implement IDisposable.
  public void Dispose() {
    Dispose(true);
    GC.SuppressFinalize(this);
  }
  protected virtual void Dispose(bool disposing) {
    if (disposing) {
      // Free other state (managed objects).
    }
    // Free your own state (unmanaged objects).
    // Set large fields to null.
  }
  // Use C# destructor syntax for finalization code.
  ~Base() {
    // Simply call Dispose(false).
    Dispose (false);
  }
}
```