The **.NET Framework** (pronounced *dot net*) is a proprietary, partially open source freeware software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (as contrasted tohardware environment), known as Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling. FCL and CLR together constitute .NET Framework.

- It is a platform for application developers.
- It is a Framework that supports Multiple Language and Cross language integration.
- It has IDE (Integrated Development Environment).
- Framework is a set of utilities or can say building blocks of your application system.
- .NET Framework provides GUI in a GUI manner.
- .NET Framework provides interoperability between languages i.e. Common Type System (CTS) .
- .NET Framework also includes the .NET Common Language Runtime (CLR), which is responsible for maintaining the execution of all applications developed using the .NET library.
- The .NET Framework consists primarily of a gigantic library of code.

Definition: A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

Cross Language integration

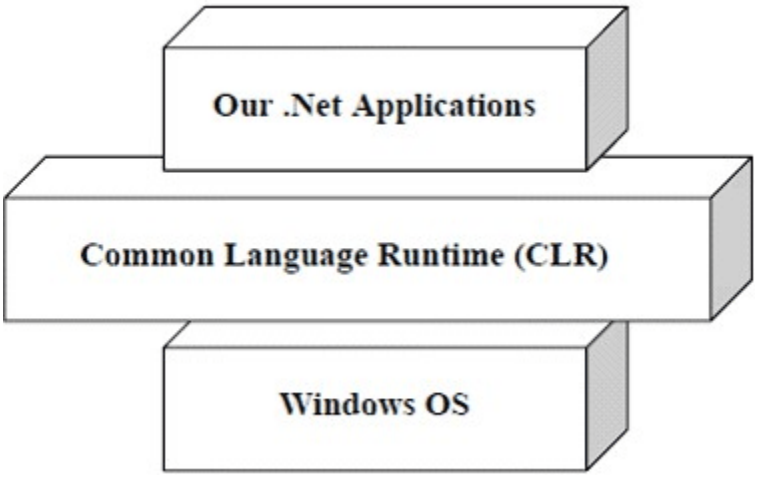
You can use a utility of a language in another language (It uses Class Language Integration).

.NET Framework includes no restriction on the type of applications that are possible. The .NET Framework allows the creation of Windows applications, Web applications, Web services, and lot more.

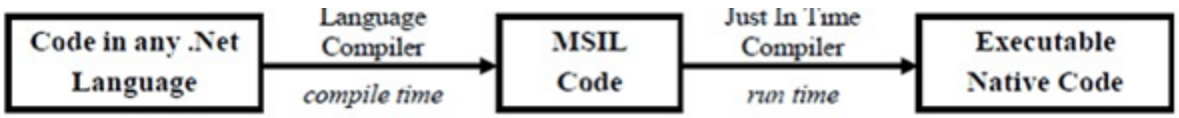
The .NET Framework has been designed so that it can be used from any language, including C#, C++, Visual Basic, JScript, and even older languages such as COBOL.

.Net Architecture and .Net Framework basics:

1. **Common Language Runtime (CLR):** The heart of the .Net Framework. It is also called the .Net runtime. It resides above the operating system and handles all .Net applications. It handles garbage collection, Code Access Security (CAS) etc.



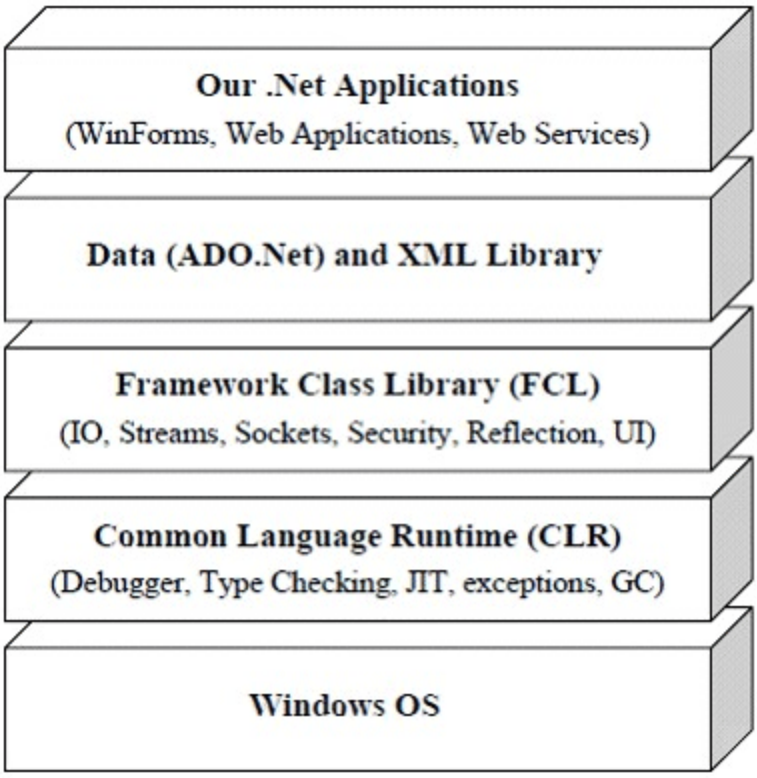
2. **Microsoft Intermediate Language (MSIL) Code:** When we compile our .Net code then it is not directly converted to native/binary code; it is first converted into intermediate code known as MSIL code which is then interpreted by the CLR. MSIL is independent of hardware and the operating system. Cross language relationships are possible since MSIL is the same for all .Net languages. MSIL is further converted into native code.



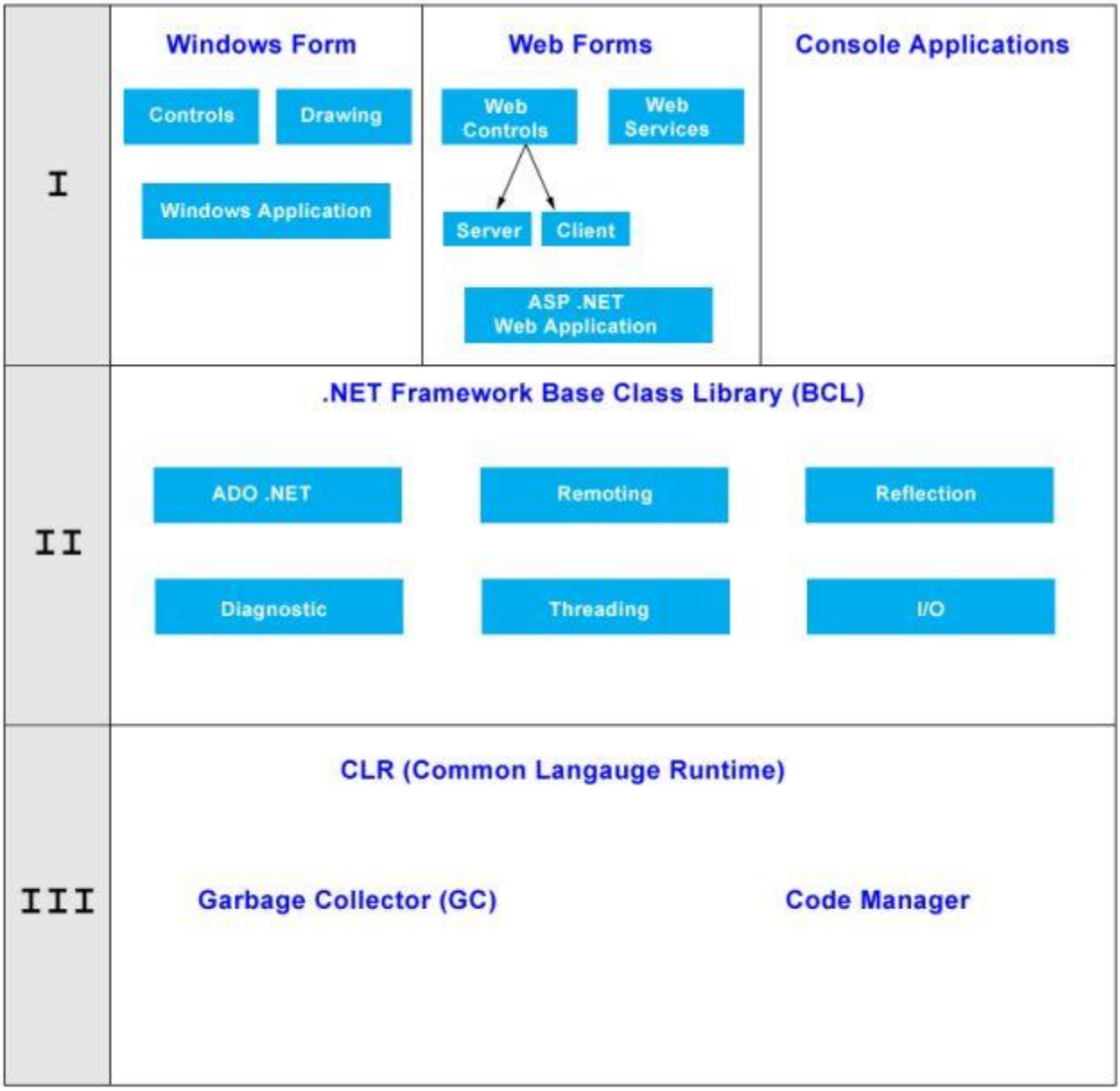
3. **Just in Time Compilers (JIT):** It compiles IL code into native executable code (exe or dlls). Once code is converted to IL then it can be called again by JIT instead of recompiling that code.
4. **Framework class library:** The .Net Framework provides a huge class library called FCL for common tasks. It contains thousands of classes to access Windows APIs and common functions like string manipulations, Data structures, stream, IO, thread, security etc.
5. **Common Language Specification (CLS):** What makes a language to be .Net compliant? Answer is CLS. Microsoft has defined some specifications that each .Net language has to follow. For e.g.: no pointer, no multiple inheritances etc.

6. **Common Type System (CTS):** CTS defines some basic data types that IL can understand. Each .Net compliant language should map its data types to these standard data types. This makes it possible for two .Net compliant languages to communicate by passing/receiving parameters to and from each other. For example CTS defines Int32 for C# int and VB integer data types.

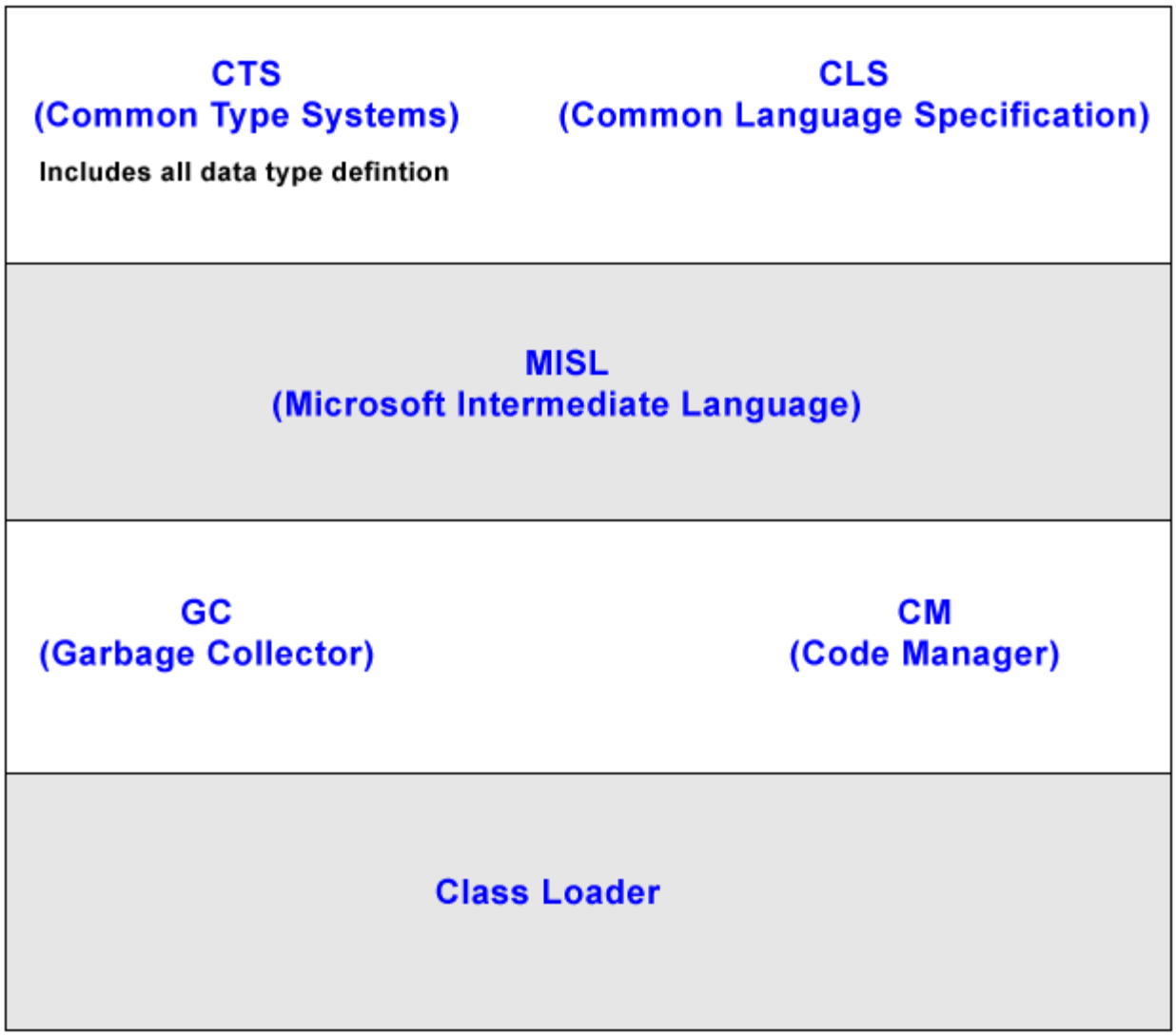
7. **The .Net Framework:** Is a combination of CLR, FCL, ADO.Net and XML classes, Web/Window applications and Web services.



Architecture of .NET Framework



Architecture of CLR



CLS (Common Language Specification)

It is a subset of CTS. All instruction is in CLS i.e. instruction of CTS is written in CLS.

Code Manager

Code manager invokes class loader for execution.

.NET supports two kind of coding

1) Managed Code

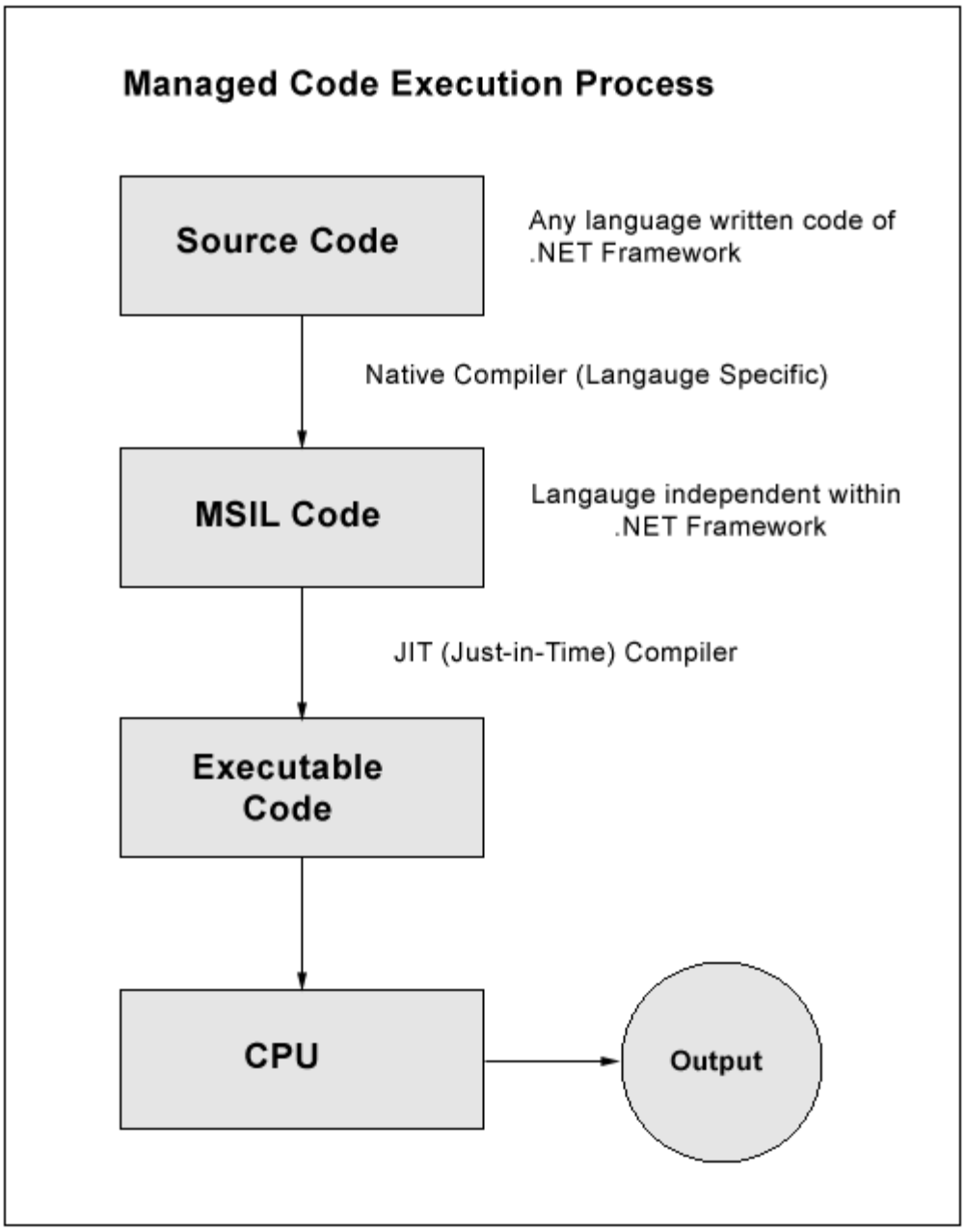
2) Unmanaged Code

Managed Code

The resource, which is with in your application domain is, managed code. The resources that are within domain are faster.

The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.

Managed code uses CLR which in turns looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on.



Unmanaged Code

The code, which is developed outside .NET, Framework is known as unmanaged code.

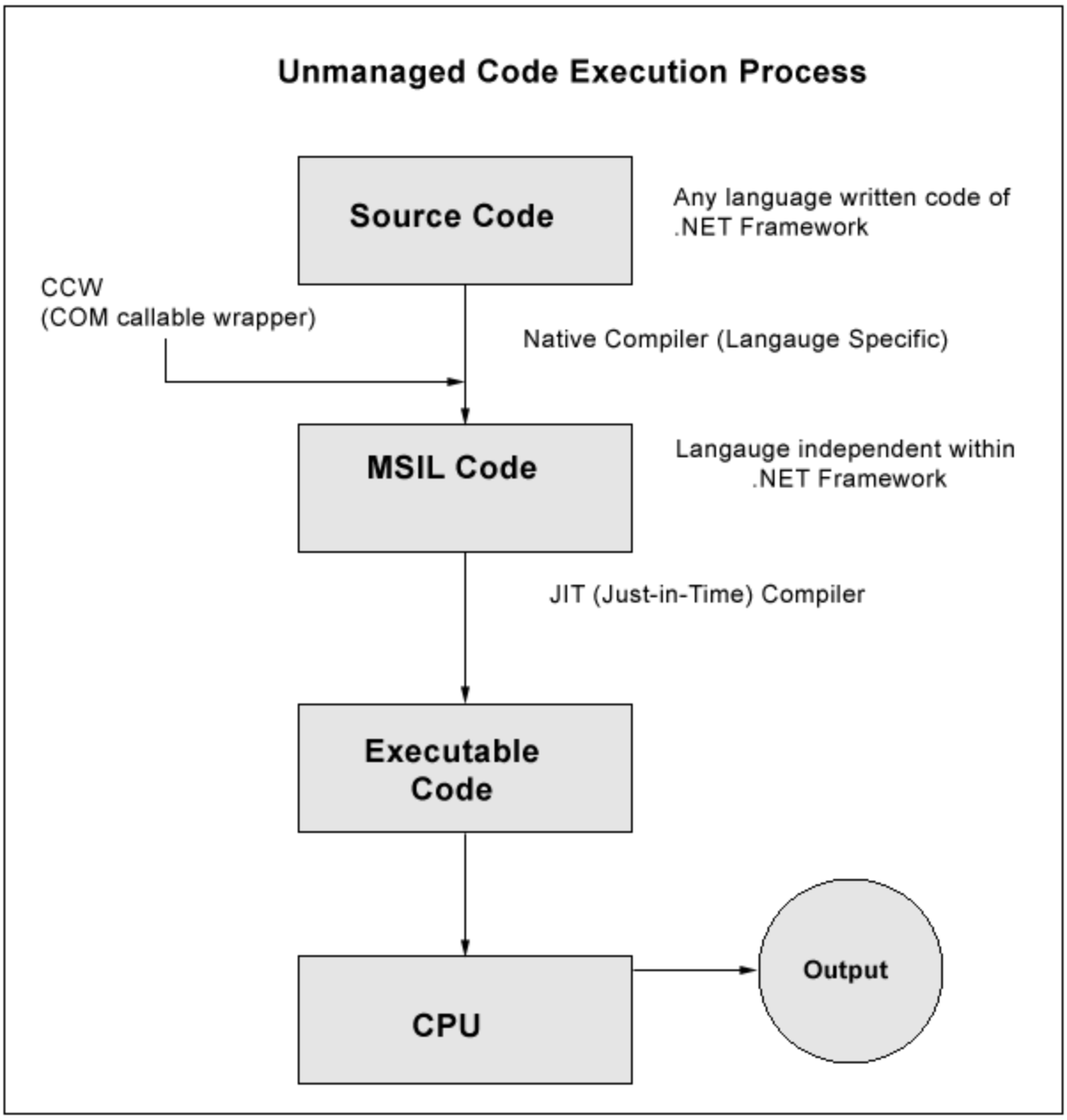
Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low - level functions of the operating system. Background compatibility with code of VB, ASP and COM are examples of unmanaged code.

Unmanaged code can be unmanaged source code and unmanaged compile code.

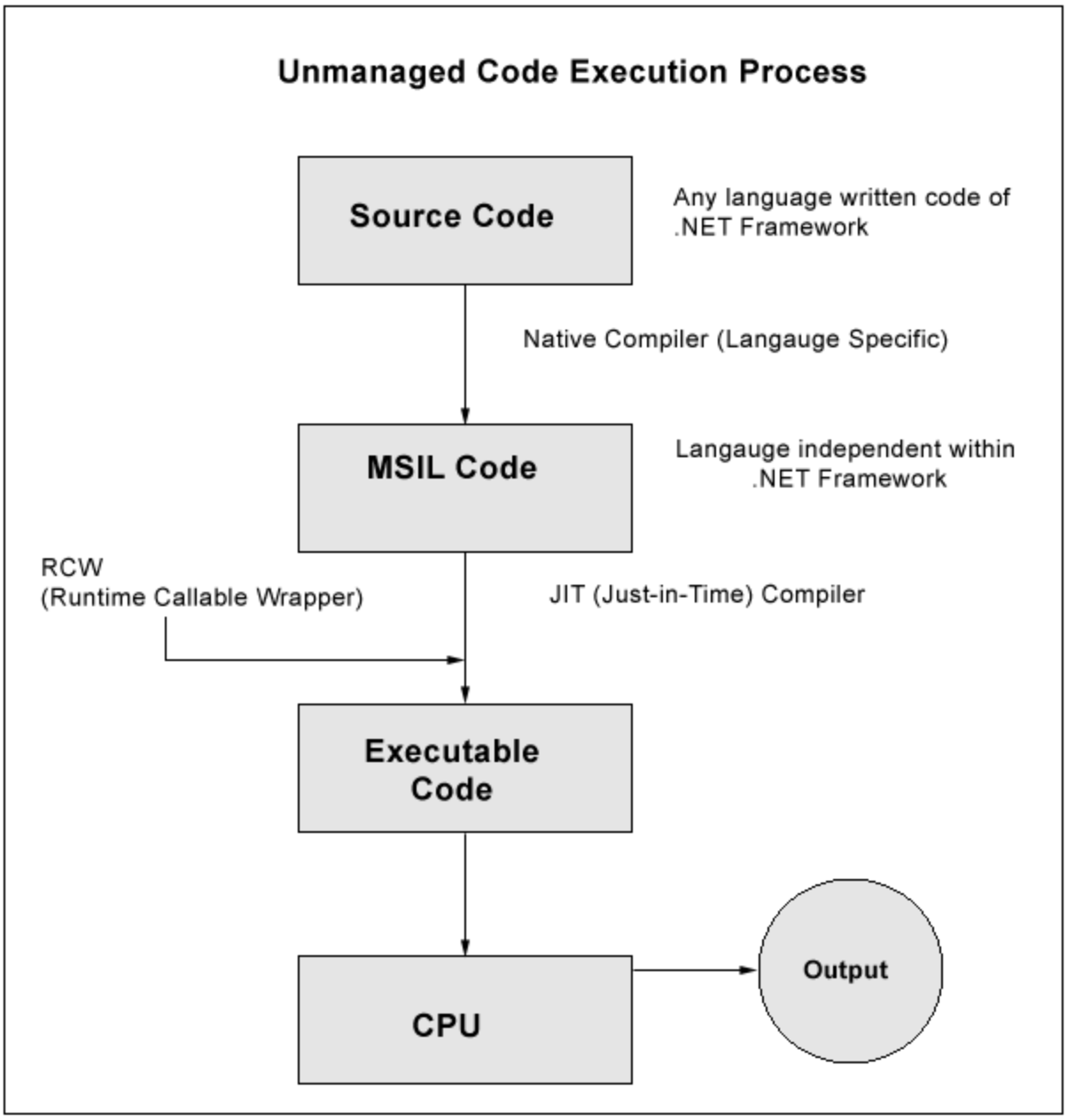
Unmanaged code is executed with help of wrapper classes.

Wrapper classes are of two types: CCW (COM callable wrapper) and RCW (Runtime Callable Wrapper).

Wrapper is used to cover difference with the help of CCW and RCW.
COM callable wrapper unmanaged code



Runtime Callable Wrapper unmanaged code



Native Code

The code to be executed must be converted into a language that the target operating system understands, known as native code. This conversion is called **compiling** code, an act that is performed by a compiler.

Under the .NET Framework, however, this is a two - stage process. With help of MSIL and JIT.

MSIL (Microsoft Intermediate Language)

It is language independent code. When you compile code that uses the .NET Framework library, you don't immediately create operating system - specific native code.

Instead, you compile your code into Microsoft Intermediate Language (MSIL) code. The MSIL code is not specific to any operating system or to any language.

JIT (Just-in-Time)

Just - in - Time (JIT) compiler, which compiles MSIL into native code that is specific to the OS and machine architecture being targeted. Only at this point can the OS execute the application. The just - in - time part of the name reflects the fact that MSIL code is only compiled as, and when, it is needed.

In the past, it was often necessary to compile your code into several applications, each of which targeted a specific operating system and CPU architecture. Often, this was a form of optimization.

This is now unnecessary, because JIT compilers (as their name suggests) use MSIL code, which is independent of the machine, operating system, and CPU. Several JIT compilers exist, each targeting a different architecture, and the appropriate one will be used to create the native code required.

The beauty of all this is that it requires a lot less work on your part - in fact, you can forget about system - dependent details and concentrate on the more interesting functionality of your code.

JIT are of three types:

1. Pre JIT
2. Econo JIT
3. Normal JIT

Pre JIT

It converts all the code in executable code and it is slow

Econo JIT

It will convert the called executable code only. But it will convert code every time when a code is called again.

Normal JIT

It will only convert the called code and will store in cache so that it will not require converting code again. Normal JIT is fast.

Assemblies

When you compile an application, the MSIL code created is stored in an assembly. Assemblies include both executable application files that you can run directly from Windows without the need for any other programs (these have a .exe file extension), and libraries (which have a .dll extension) for use by other applications.

In addition to containing MSIL, assemblies also include meta information (that is, information about the information contained in the assembly, also known as metadata) and optional resources (additional data used by the MSIL, such as sound files and pictures).

The meta information enables assemblies to be fully self - descriptive. You need no other information to use an assembly, meaning you avoid situations such as failing to add required data to the system registry and so on, which was often a problem when developing with other platforms.

This means that deploying applications is often as simple as copying the files into a directory on a remote computer. Because no additional information is required on the target systems, you can just run an executable file from this directory and (assuming the .NET CLR is installed) you're good to go.

Of course, you won't necessarily want to include everything required to run an application in one place. You might write some code that performs tasks required by multiple applications. In situations like that, it is often useful to place the reusable code in a place accessible to all applications. In the .NET Framework, this is the Global Assembly Cache (GAC). Placing code in the GAC is simple - you just place the assembly containing the code in the directory containing this cache.

Garbage Collection (GC)

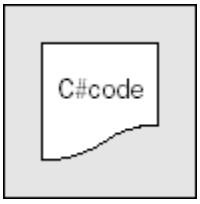
One of the most important features of managed code is the concept of garbage collection. This is the .NET method of making sure that the memory used by an application is freed up completely when the application is no longer in use.

Prior to .NET this was mostly the responsibility of programmers, and a few simple errors in code could result in large blocks of memory mysteriously disappearing as a result of being allocated to the wrong place in memory. That usually meant a progressive slowdown of your computer followed by a system crash.

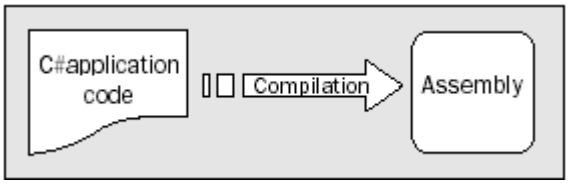
.NET garbage collection works by inspecting the memory of your computer every so often and removing anything from it that is no longer needed. There is no set time frame for this; it might happen thousands of times a second, once every few seconds, or whenever, but you can rest assured that it will happen.

Will try to explain the processing in terms of C# code which is written using .NET Framework.

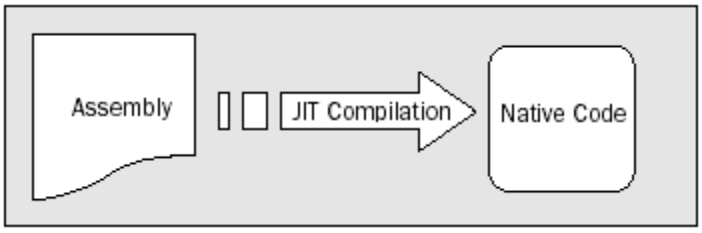
Step 1- Application code is written using a .NET - compatible language C#.



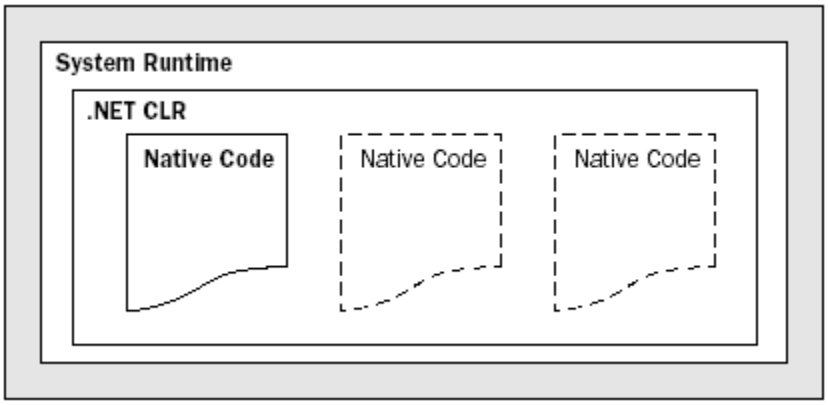
Step 2 - Code is compiled into MSIL, which is stored in an assembly (see Figure 1 - 2).



Step 3 - When this code is executed (either in its own right if it is an executable or when it is used from other code), it must first be compiled into native code using a JIT compiler.



Step 4 - The native code is executed in the context of the managed CLR, along with any other running applications or processes.



Note: One additional point concerning this process. The C# code that compiles into MSIL in step 2 needn't be contained in a single file. It's possible to split application code across multiple source code files, which are then compiled together into a single assembly. This extremely useful process is known as linking.

This is because it is far easier to work with several smaller files than one enormous one. You can separate out logically related code into an individual file so that it can be worked on independently and then practically forgotten about when completed.

This also makes it easy to locate specific pieces of code when you need them and enables teams of developers to divide up the programming burden into manageable chunks, whereby individuals can check out pieces of code to work on without risking damage to otherwise satisfactory sections or sections other people are working on.