Tecniche di Programmazione avanzata

Corso di Laurea Specialistica in Ingegneria Telematica
Università Kore – Enna – A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Il framework .NET

Piattaforma .NET

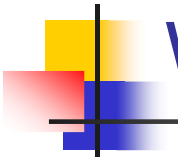
Cosa è la piattaforma .NET? Un modello a componenti per Internet per:

- costruire sistemi distribuiti su larga scala
- integrare dispositivi multipli
- operare con tool e protocolli standard (XML, WSDL, SOAP, HTTP)
- Permettere Interoperabilità fra linguaggi e ambienti
- Usare XML per lo scambio dei Dati
- Estendere o utilizzare codice esistente
- Ridurre della complessità di programmazione di ambienti
- Semplificare lo sviluppo e il deployment
  - Ambiente object-oriented
    - Qualsiasi entità è un oggetto
    - Classi ed ereditarietà pienamente supportati
    - Anche tra linguaggi diversi
  - Riduzione errori comuni di programmazione
    - Linguaggi fortemente tipizzati
    - Errori non gestiti – generazione di eccezioni
    - Meno memory leak – Garbage Collector
- Aumentare l'affidabilità del codice
- Unificare il modello di programmazione
- Modello basato su Web Services XML

Piattaforma .NET e Internet

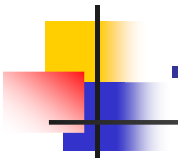
Web Service

- Un **Web Service** è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete
- caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad esempio, il Web Services Description Language, **WSDL**) utilizzando la quale altri sistemi possono interagire con il Web Service stesso attivando le operazioni descritte nell'interfaccia tramite appositi "messaggi" inclusi in una "busta" (la più famosa è **SOAP**): tali messaggi sono, solitamente, trasportati tramite il protocollo **HTTP** e formattati secondo lo standard **XML**.
- Proprio grazie all'utilizzo di standard basati su XML, tramite un'architettura basata sui Web Service (chiamata Service oriented Architecture - **SOA**), applicazioni software scritte in diversi linguaggi di programmazione e implementate su diverse piattaforme hardware possono quindi essere utilizzate



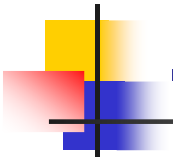
Web Service

- **Svantaggi** dei Web Services:
  - attualmente non esistono standard consolidati
  - le performance sono minori di quelle riscontrabili utilizzando altri approcci di distributed computing quali Java RMI, CORBA, o DCOM
  - L'uso dell'HTTP permette ai Web Service di evitare le misure di sicurezza dei firewall (le cui regole sono stabilite spesso proprio per evitare le comunicazioni fra programmi "esterni" ed "interni" al firewall).



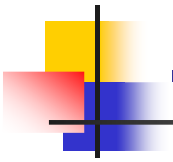
.NET Framework

- Le fondamenta dell'architettura .NET
  - Sistema di sviluppo unificato e standard
    - C# e CLI certificati ECMA (La European Computer Manufacturers Association è un'associazione fondata nel 1961 e dedicata alla standardizzazione nel settore informatico e dei sistemi di comunicazione)
    - Sviluppo in collaborazione con HP e Intel
  - Supporto per qualunque linguaggio di programmazione
    - Conservazione del know-how esistente.
    - Utilizzo di linguaggi adatti all'applicazione richiesta
    - Interoperabilità tra linguaggi conformi: This means that you can inherit from classes, catch exceptions, and take advantage of polymorphism across different languages.



.NET Framework

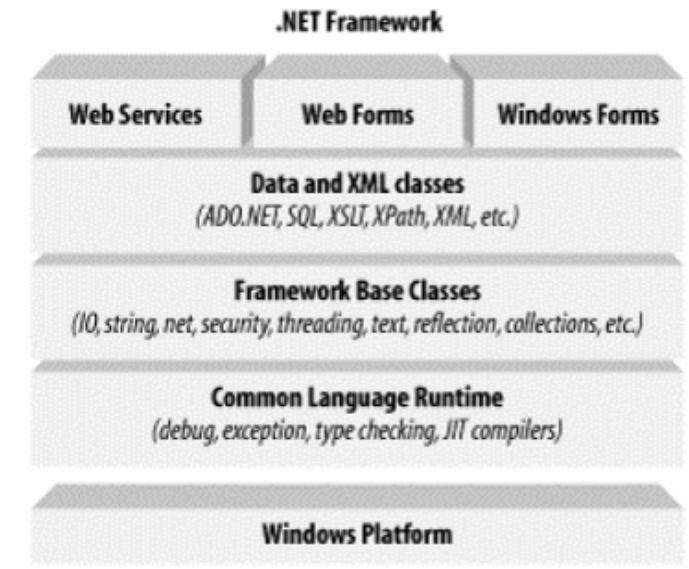
- Le fondamenta dell'architettura .NET
  - Architettura "Multipiattaforma"
    - Accessibile a qualunque piattaforma via Web Service.
    - Disponibile per qualunque piattaforma Windows presente e futura.
    - Scalabile
    - Pronto per dispositivi non strettamente informatici (cellulari,TV).
    - Elevato supporto per le interfacce grafiche lato client.



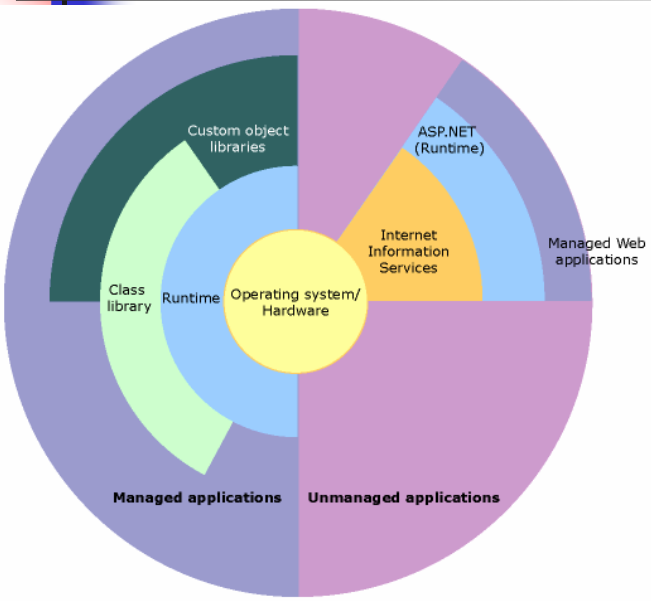
.NET Framework

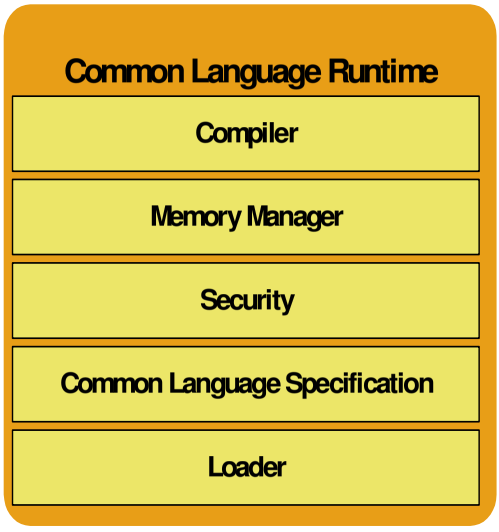
- Le fondamenta dell'architettura .NET
  - Sistema di comunicazione basato su XML e SOAP
    - Facilità di trasporto.
    - Modello dati unificato.
    - Interoperabilità con altri sistemi.
    - Possibilità di rendere persistenti dati e strutture.

Componenti principali del Framework

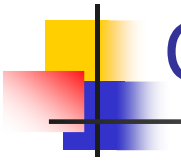


Common Language Runtime



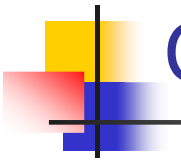


- **CLR, anche noto come CLI (Common Language Infrastructure), è:**
  - **un motore di esecuzione ad alte prestazioni**
  - **uno strumento di sviluppo**



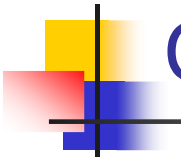
Common Language Runtime

- Motore di esecuzione ad elevate prestazioni
  - Gestione della memoria e Garbage Collection
  - Gestione dei thread e dei servizi del sistema operativo
  - Gestione della sicurezza
  - Gestione automatica delle dipendenze da altre componenti
  - Compilazione JIT di tutto il codice (nessuna interpretazione, vs java)



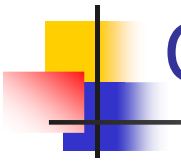
Common Language Runtime

- Strumento di sviluppo
  - Controllo sui tipi
  - Gestione delle eccezioni interlinguaggio
  - Accesso facilitato a servizi avanzati
  - Ambiente di debug unificato per tutti i linguaggi conformi
  - Linguaggi supportati : C#, VB.NET, C++, J#, …



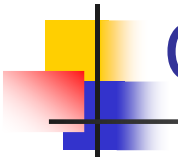
Common Language Runtime

- Linguaggi supportati:
  - **C#**, un linguaggio OO sviluppato da Microsoft all'interno dell'iniziativa .NET. Il C# prende spunto da Delphi (stesso autore), dal C++ (con meno simbolismo), da Java (con meno elementi "decorativi") e da Visual Basic (programmazione visuale e semplicità)
  - Notare che C# è un linguaggio, **Visual C#** è un tool di Visual Studio Microsoft per lo sviluppo di applicazioni C#; generalmente, se viene sfruttato il Visual Studio, probabilmente le applicazioni sono dotate di GUI, anche se questo non è obbligatorio (posso lavorare ad una GUI anche con un editor di testo, soltanto non sarà disponibile nessuna anteprima).
  - **Visual Basic .NET**, come C#, è un linguaggio OO sviluppato da Microsoft all'interno dell'iniziativa .NET. Con VB.NET è possibile realizzare applicazioni windows forms, web, servizi, componenti COM, Web service ed anche destinate a dispositivi mobile tramite l'uso delle librerie del Compact framework.
  - **C++** [...]



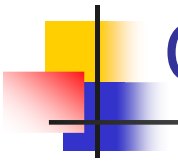
Common Language Runtime

- Linguaggi supportati:
  - **J#** è un linguaggio nato per consentire ai programmatori in Java (SUN) e in J++ (Microsoft) di migrare verso i linguaggi della piattaforma .NET. Come il J++, anche il J# supporta solo una parte delle funzionalità di Java. Se si vuole eseguire codice Java su .NET si può ricorrere a IKVM. Microsoft continuerà a supportare J# fino al 2015, ma J# già non è supportato da Visual Studio 9.0 2008
  - **J++** was Microsoft's specific implementation of Java, shipped with Microsoft Visual Studio 6.0 (before .NET deployment). Optimized for the Windows Platform, J++ programs could only run on the MSJVM (Microsoft Java Virtual Machine), which was Microsoft's attempt at a faster interpreter. Syntax is the same as Java's but some details (JNI, RMI, applets) are not supported.



Common Language Runtime

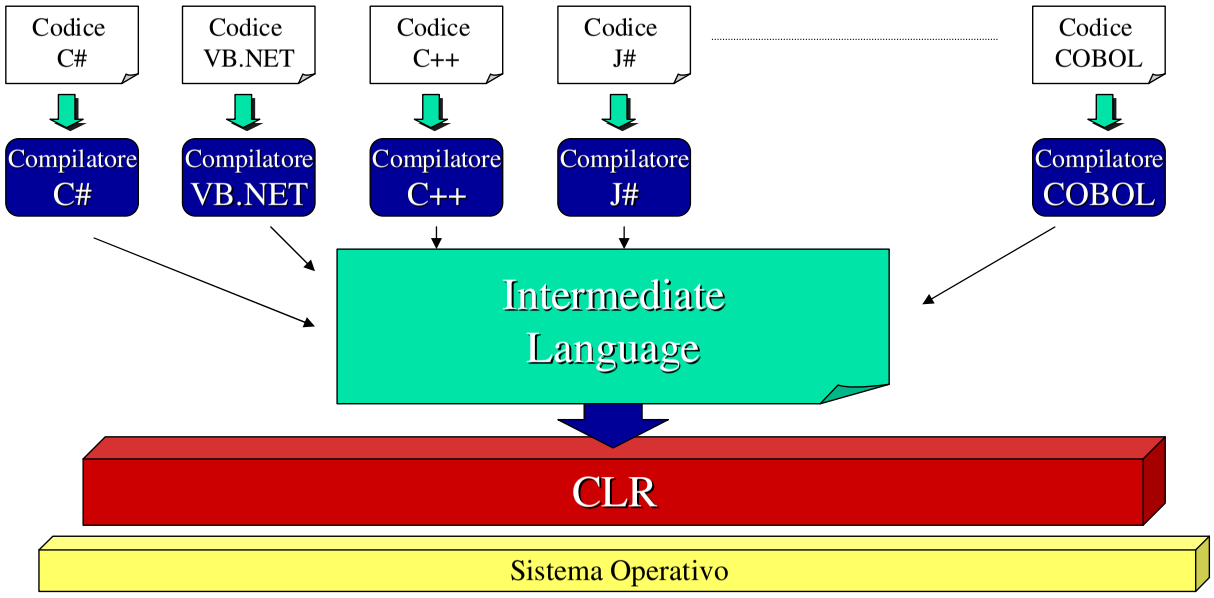
- Managed Code (Codice Gestito)
  - Tutto il codice aderente alle specifiche del CLR del quale può sfruttare i servizi
  - Codice "Sicuro"
- Unmanaged Code
  - Tutto il resto…
  - Codice "Insicuro" perché scavalca il CLR



Common Language Runtime

- Il Common Language Runtime è composto da cinque componenti:
  - CTS – **Common Type System**
  - CLS – **Common Language Specification**
  - CIL – **Common Intermediate Language**
  - JIT – **Just In Time Compiler**
  - VES – **Virtual Execution System**
- Di fatto il CLR gestisce la compilazione del CIL in linguaggio macchina

Common Language Runtime



Common Language Runtime

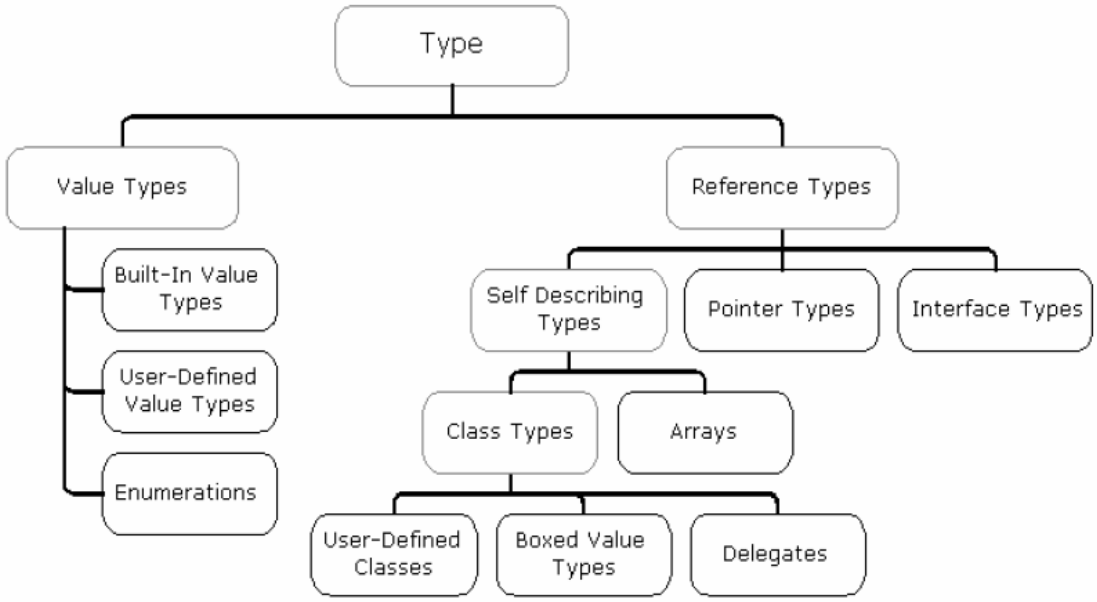
Altri servizi del CLR:

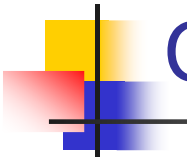
- Reflection
  - È possibile interrogare un assembly caricato in memoria
  - Tipi (classi, interfacce, enumeratori, etc.)
  - Membri (attributi, proprietà, metodi, etc.)
  - Parametri
  - È possibile forzare il caricamento in memoria di un assembly con i metodi Load/LoadFrom
- Remoting
  - Chiamata di componenti remoti .NET
  - Interoperabilità (COM, Platform Invoke)

Common Type System (CTS)

- Sistema di Tipi unificato ed interlinguaggio
- Un insieme standard di tipi di dato e di regole necessarie per la realizzazione di nuovi tipi
- Due Categorie di Tipi disponibili:
  - Value Type
  - Reference Type

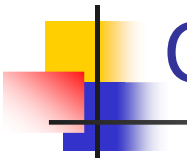
Common Type System (CTS)





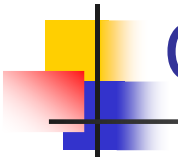
Common Type System (CTS)

- **Value Type**
  - Tipi atomici come integer e char
  - Divisi in *built-in* ed *user defined*
  - Descrivono valori che sono rappresentati come sequenze di bit
  - Allocati nello Stack del Thread
  - Non soggetti al Garbage Collector



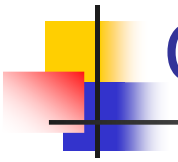
Common Type System (CTS)

- **Reference Type**
  - Entità autodefinite contententi sia metodi che variabili
  - Divisi in quattro sottocategorie:
    - Self Describing
    - Interface
    - Pointer
    - Built-in
  - Descrivono valori che sono rappresentati come la locazione di una sequenza di bit
  - Allocati nell' Heap Gestito (Managed Heap)
  - Soggetti al Garbage Collector



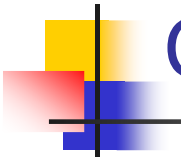
Common Type System (CTS)

- The primary difference between reference and value types is how instances of the two types are treated by the CLR. One difference is that the GC collects instances of reference types that are no longer referenced by the application. Instances of value types are automatically cleaned up when the variable goes out of scope.
- Another difference is when one variable is set equal to another or passed as a parameter to a method call. When a variable of a *reference type (A)* is set equal to another variable of the same type *(B)*, variable *A* is assigned a reference to *B.* Both variables reference the same object. When a variable of *value type (A)* is set equal to another variable of the same type *(B)*, variable *A* receives a copy of the contents of *B.* Each variable will have its own individual copy of the data.



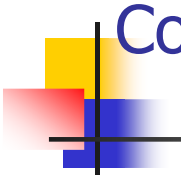
Common Type System (CTS)

- Yet another difference between the behaviors of value types versus reference types is how equality is determined. Two variables of a given reference type are determined to be equal if both the variables refer to the same object. Two variables of a given value type are determined to be equal if the state of the two variables are equal.
- The final difference between the two is the way the instances of a type are initialized. In a reference type, the variable is initialized with a default value of *Null.* The variable will not reference an object until explicitly done by the object. Whereas a variable declared as a value type will always reference a valid object.



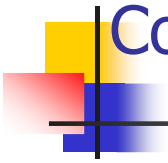
Common Type System (CTS)

- Conversione tra Value e Reference (Boxing e Unboxing) gestito dal CTS
- Regole di Casting gestite dal CTS



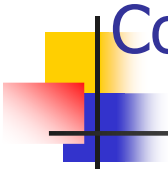
Common Language Specification (CLS)

- Il CLS definisce un sottoinsieme del CTS (Common Type System) al quale tutti i fornitori di librerie di classi e progettisti di linguaggi che puntano al CLR, devono aderire.
- Se un componente scritto in un linguaggio (ad esempio C#) dovrà essere utilizzato da un altro linguaggio (ad esempio VB.NET), allora chi scrive il componente dovrà aderire ai tipi e alle strutture definite dal CLS.
  - Ad esempio, il tipo Int32 è compatibile con il CLS ed i linguaggi e gli strumenti possono aspettarsi che altri linguaggi e strumenti conformi al CLS sappiano come utilizzarlo correttamente



Common Language Specification (CLS)

- CLS Framework
  - Una libreria costituita da codice aderente al CLS
- CLS Consumer
  - Un linguaggio o tool di sviluppo progettato per accedere a tutte le caratteristiche fornite dai CLS Framework, ma non necessariamente in grado di produrne di nuove.
- CLS Extender
  - Superset del CLS Consumer



Common Intermediate Language (CIL)

- CIL per ECMA, MSIL o IL per Microsoft
  - Tutti i compilatori che aderiscono alla struttura del CLR devono generare un rappresentazione intermedia del codice, indipendente dalla CPU, chiamata Common Intermediate Language (CIL). Il runtime utilizza questo linguaggio intermedio per generare codice nativo oppure viene eseguito al volo mediante la compilazione Just In Time

Common Intermediate Language (CIL)

- Presenta similitudini con linguaggi ad alto livello, ma anche con il linguaggio assembly:
  - Istruzioni per
    - il caricamento, la memorizzazione e l'inizializzazione dei dati
    - richiamare metodi da oggetti
    - aritmetiche e logiche
    - gestione eccezioni di tipo "Try/Catch"
    - Operazioni sui registri, ma indipendente dalla piattaforma
    - Operazioni "atomiche"

Common Intermediate Language (CIL)

- Permette al CLR controlli durante la compilazione:
  - Codice Type Safe
  - Puntatori corretti
  - Conversioni corrette
  - ecc.
- Di fatto rappresenta il linguaggio a livello più basso e l'unico "eseguibile" dal CLR

Common Intermediate Language (CIL)

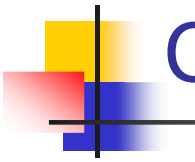
- Un compilatore conforme al CLS produce
  - Codice IL
    - Rappresenta il programma vero e proprio
  - Metadati
    - Descrivono i tipi specifici appartenenti al Common Language Types (CLT) utilizzati nel codice, comprendente la definizione di ogni tipo, le signature per ogni membro del tipo, i membri ai quali il codice fa riferimento e gli altri dati che il runtime usa durante l'esecuzione.
    - Permettono componenti autodescrittivi

Common Intermediate Language (CIL)

- IL e Metadati sono alla fine contenuti in uno o più file PE (Portable Executable) nella forma tradizionale:
  - .exe
    - Se è codice di programma eseguibile
  - .dll
    - Se è un insieme di librerie
- Il file in formato PE è detto anche Assembly, unità atomica di deployment

CIL e formato PE

- Il formato **Portable Executable** (PE) è un formato di file per file eseguibili, file oggetto, librerie condivise e device drivers, usato nelle versioni a 32-bit e 64-bit di Windows a partire da Windows NT 3.51 (quindi PE non è nato con .NET)
- Il formato PE è praticamente una struttura dati che incapsula le informazioni necessarie per il loader di Windows per gestire il codice eseguibile. Ciò include la risoluzione delle dipendenze dalle librerie condivise, tabelle di import ed export dell'API, dati per la gestione delle risorse e dati thread-local storage (TLS). Sui sistemi operativi della famiglia NT, il formato PE è usato per EXE, DLL, OBJ, SYS (device driver), OCX (controlli ActiveX).
- Del formato PE sono state create alcune estensioni:
  - il formato .NET PE per le applicazioni .NET, in cui sono aggiunti un header ed una sezione data che contiene l'IL ed il manifesto
  - una versione a 64-bit chiamata PE32+ (anche chiamato PE+)
  - una versione sviluppata specificamente per Windows CE.



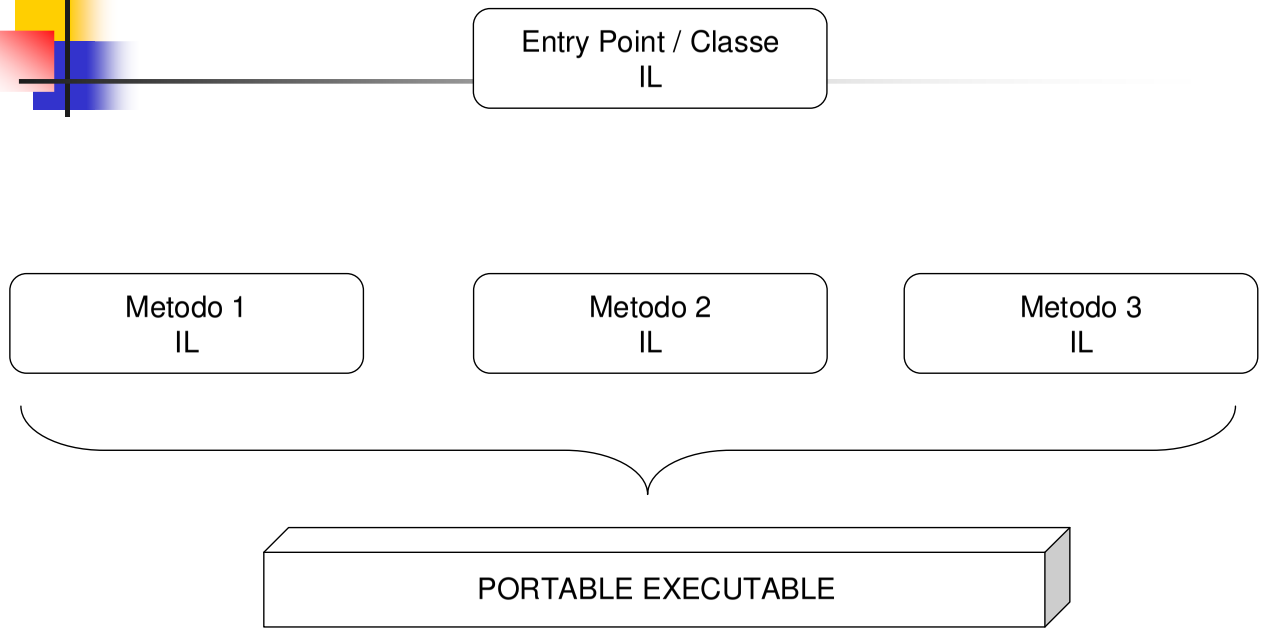
CIL e formato PE

- Il formato PE del mondo Windows si contrappone al **formato ELF** (Executable and Link Format) diffuso in ambiente Unix
- Esistono emulatori di PE su Unix
- Nel caso che non ci siano rilocazioni il formato PE ha il vantaggio di un codice veramente efficiente rispetto ad ELF, ma in presenza di queste l'utilizzo della memoria diventa davvero eccessivo.
- Il formato ELF invece supporta codice PIC (Position-Independent Code), cosa che diminuisce di poco i tempi di esecuzione ma elimina i problemi della rilocazione, consentendo di non sprecare mai memoria.

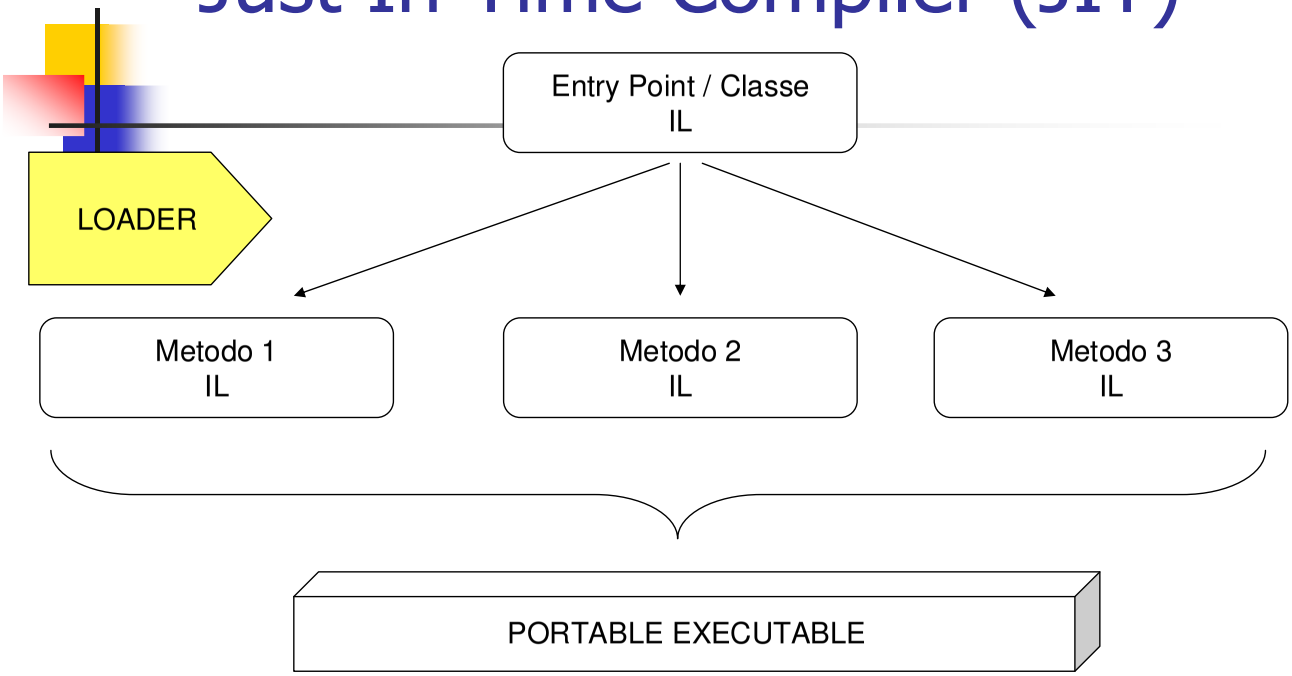
Just In Time Compiler (JIT)

- Compilatore al volo basato sul concetto JIT:
  - Non tutto l'IL di un PE viene eseguito durante un programma, solo la parte necessaria viene compilata un istante prima della sua esecuzione.
  - Il codice compilato viene memorizzato per successive esecuzioni
  - Tutto il codice .NET è compilato JIT, anche linguaggi di scripting come VB Script, J Script, JavaScript ecc.

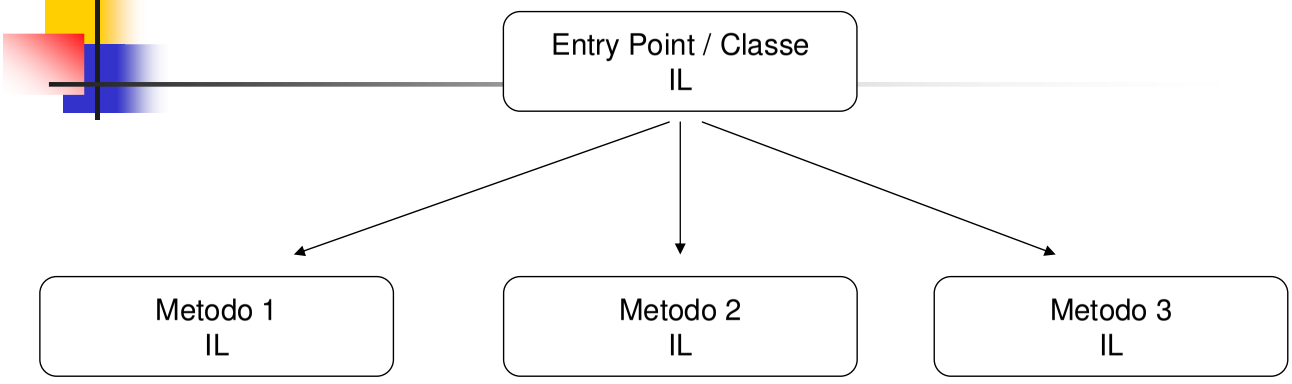
Just In Time Compiler (JIT)



Just In Time Compiler (JIT)

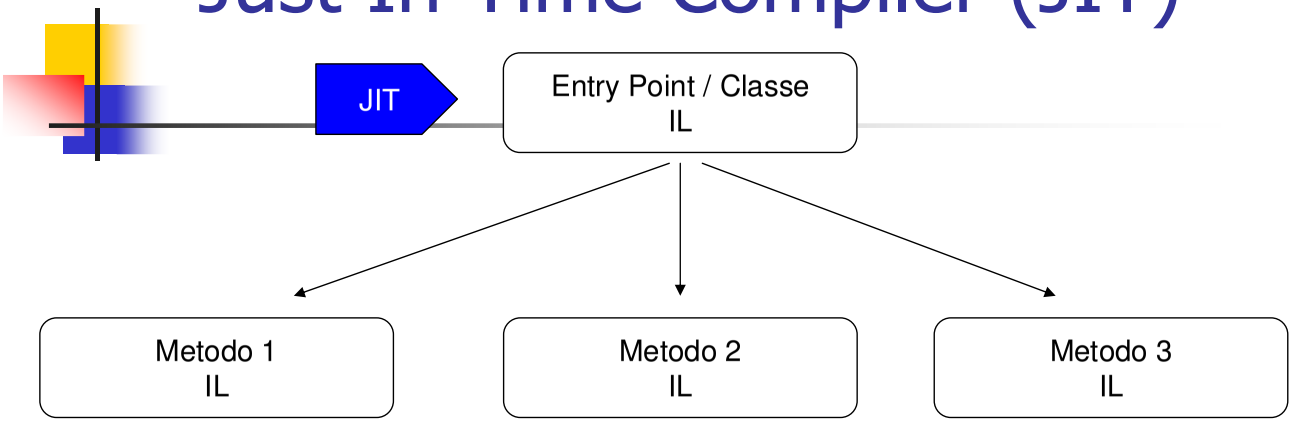


Just In Time Compiler (JIT)



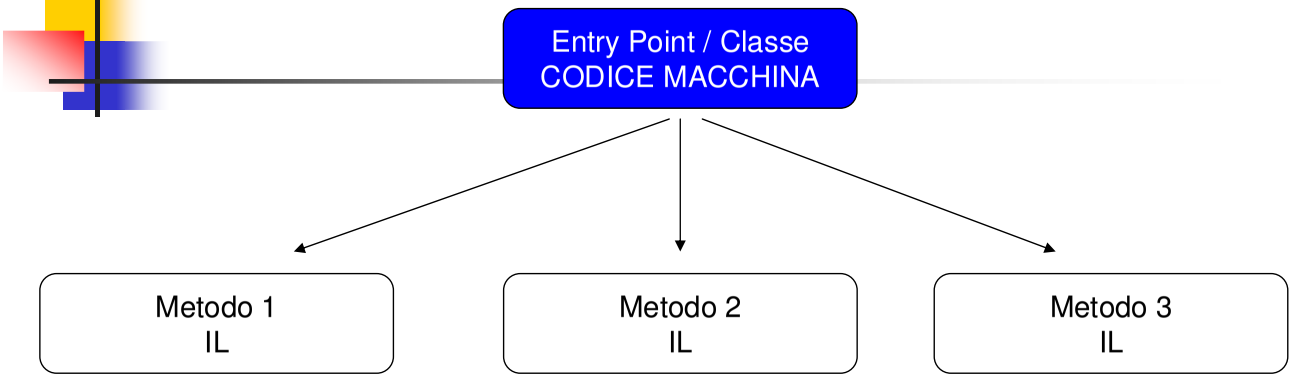
Esecuzione

Just In Time Compiler (JIT)



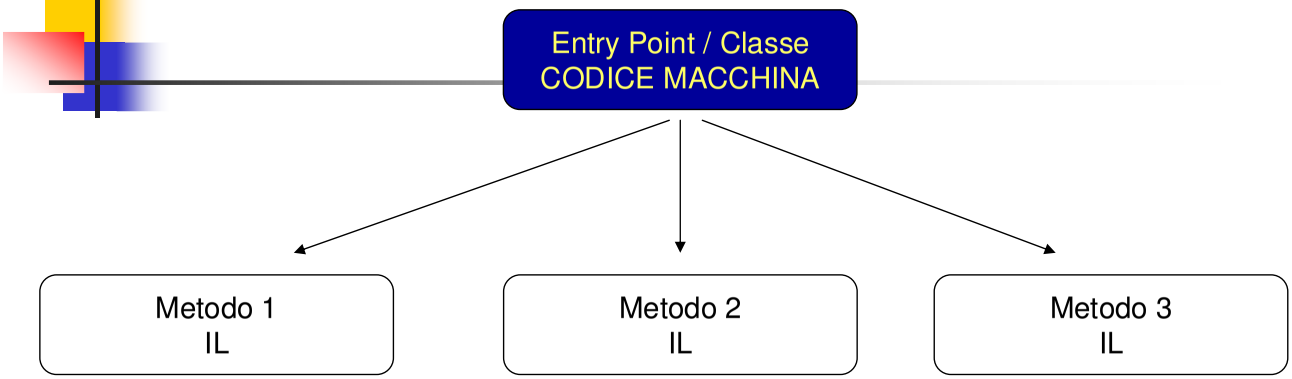
Esecuzione

Just In Time Compiler (JIT)



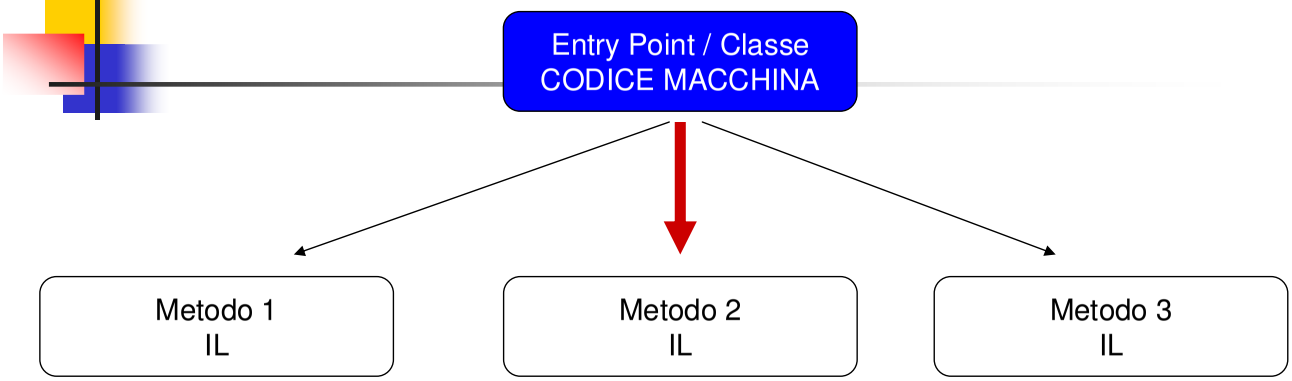
Esecuzione

Just In Time Compiler (JIT)

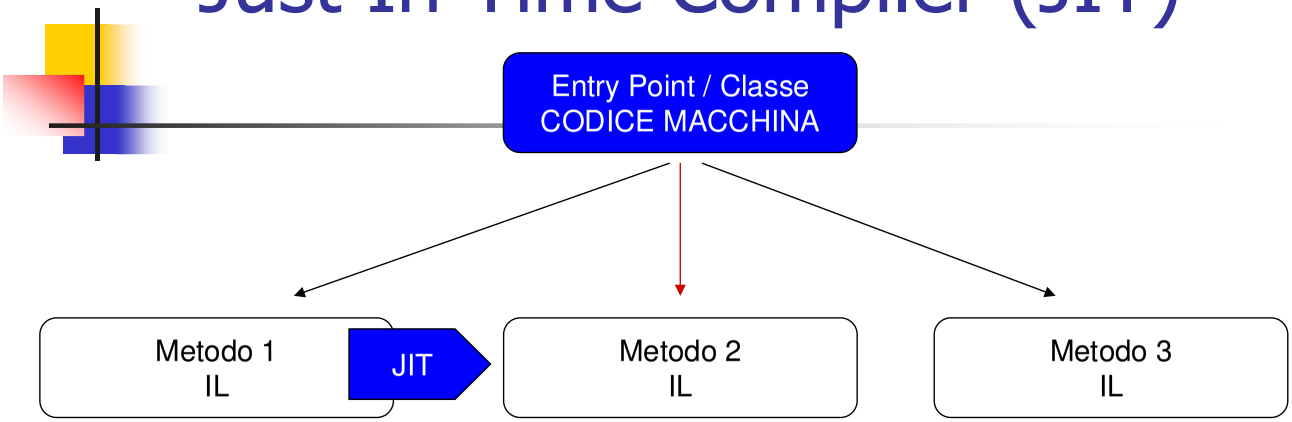


Esecuzione

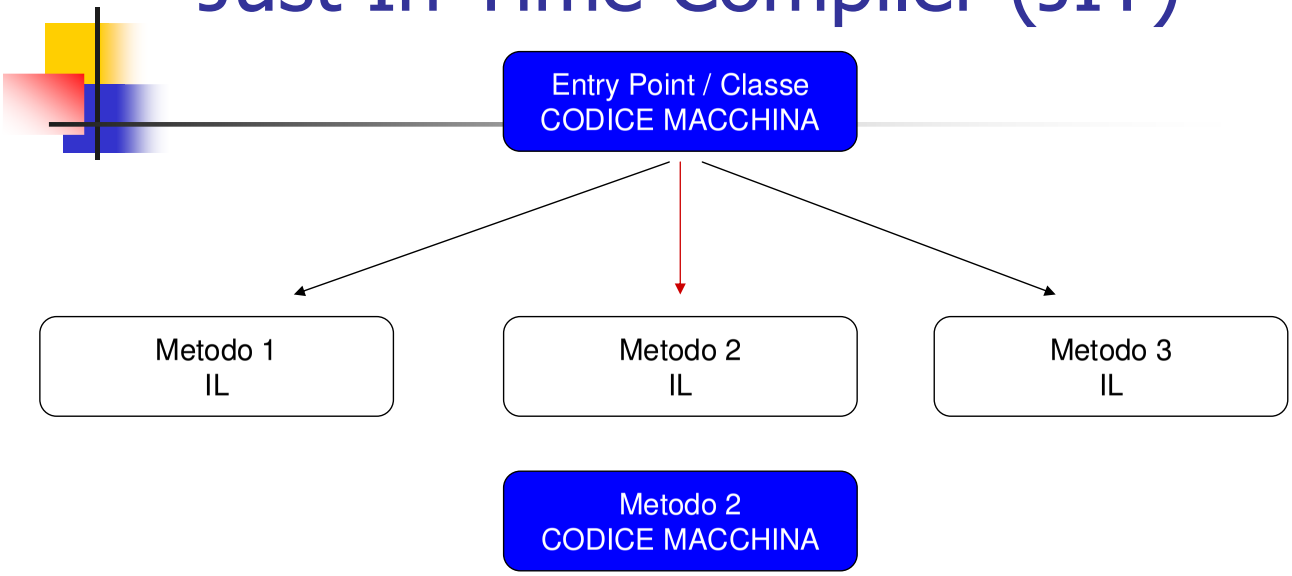
Just In Time Compiler (JIT)



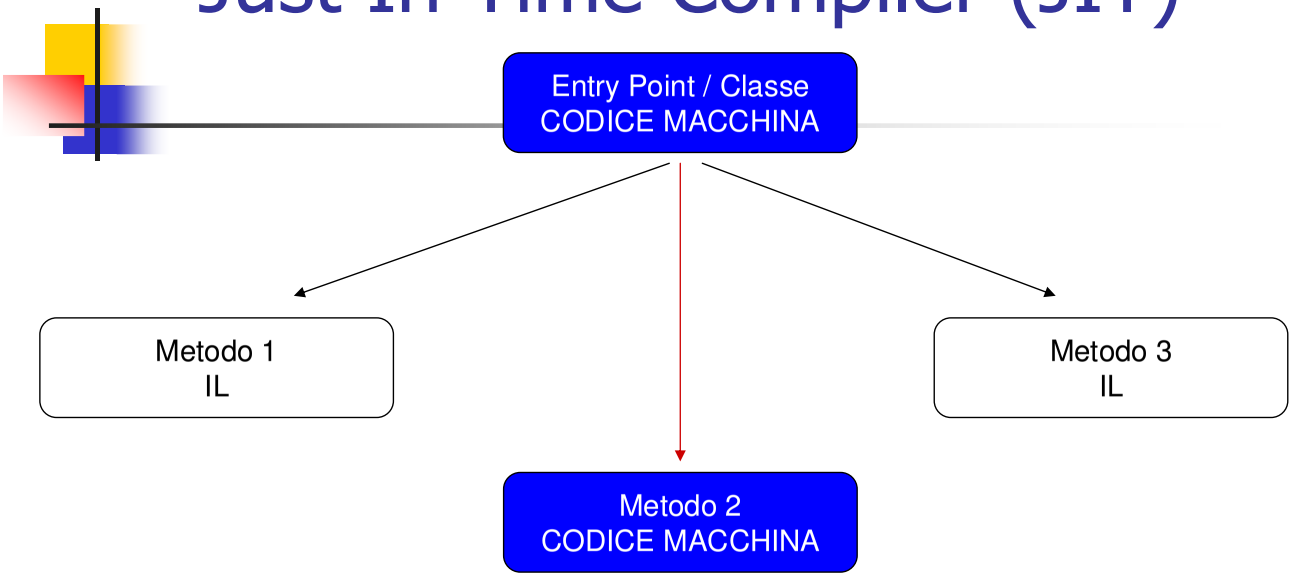
Just In Time Compiler (JIT)



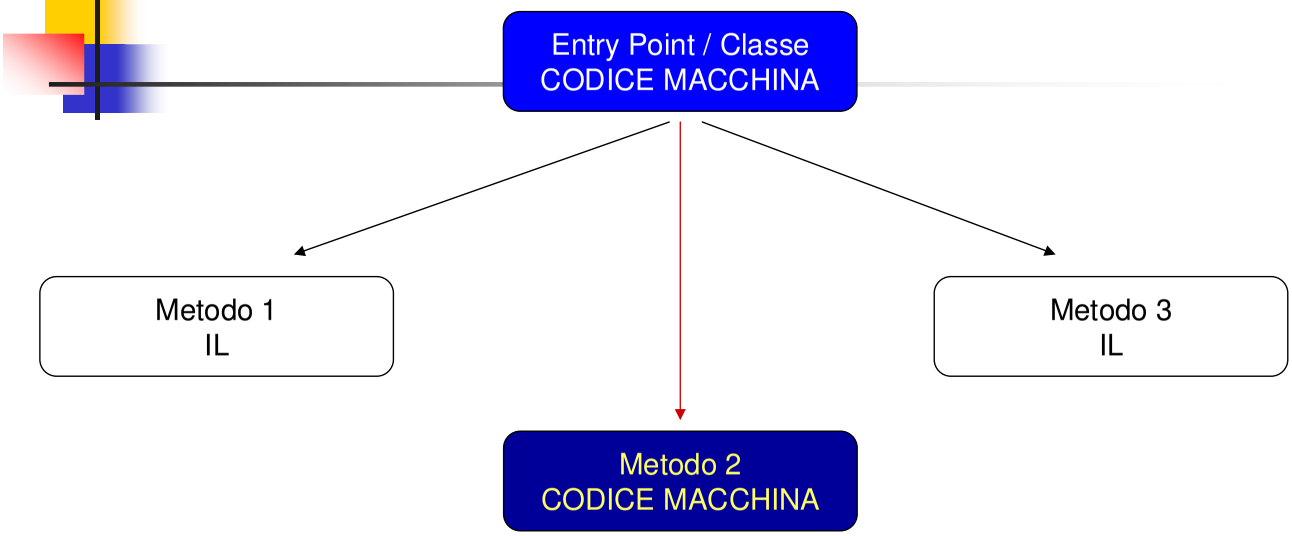
Just In Time Compiler (JIT)



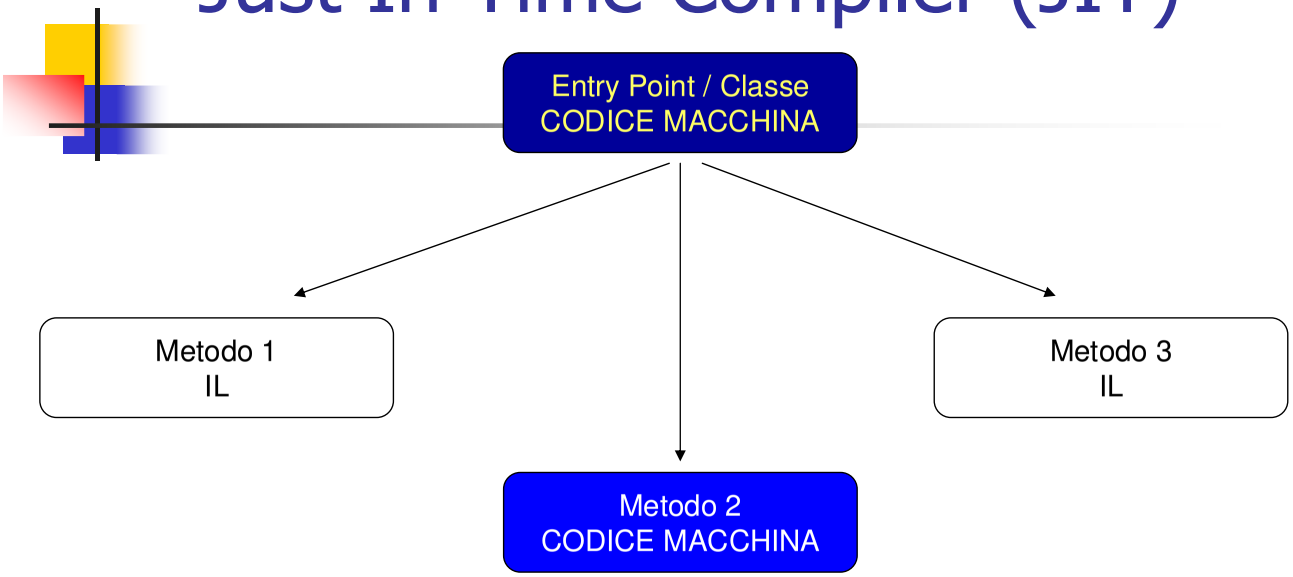
Just In Time Compiler (JIT)



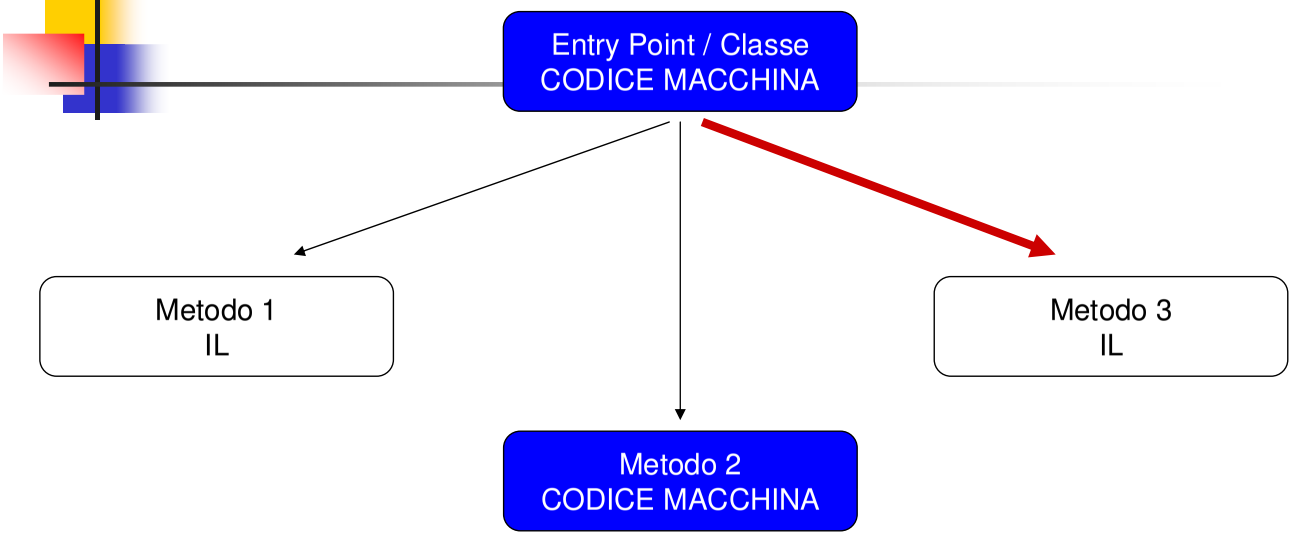
Just In Time Compiler (JIT)



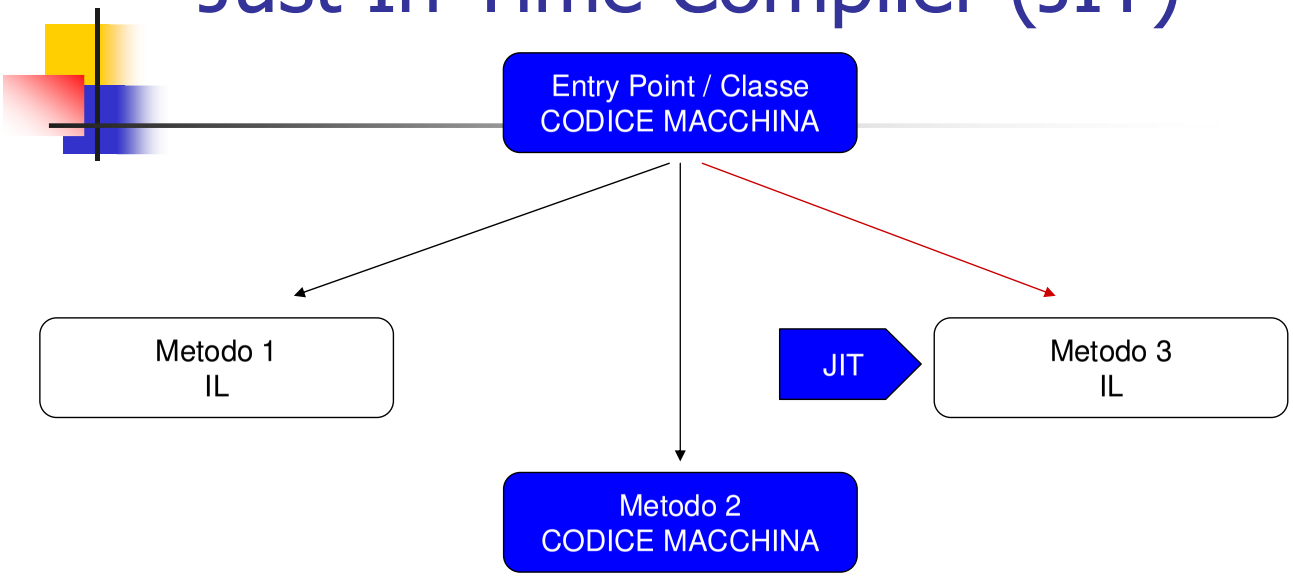
Just In Time Compiler (JIT)



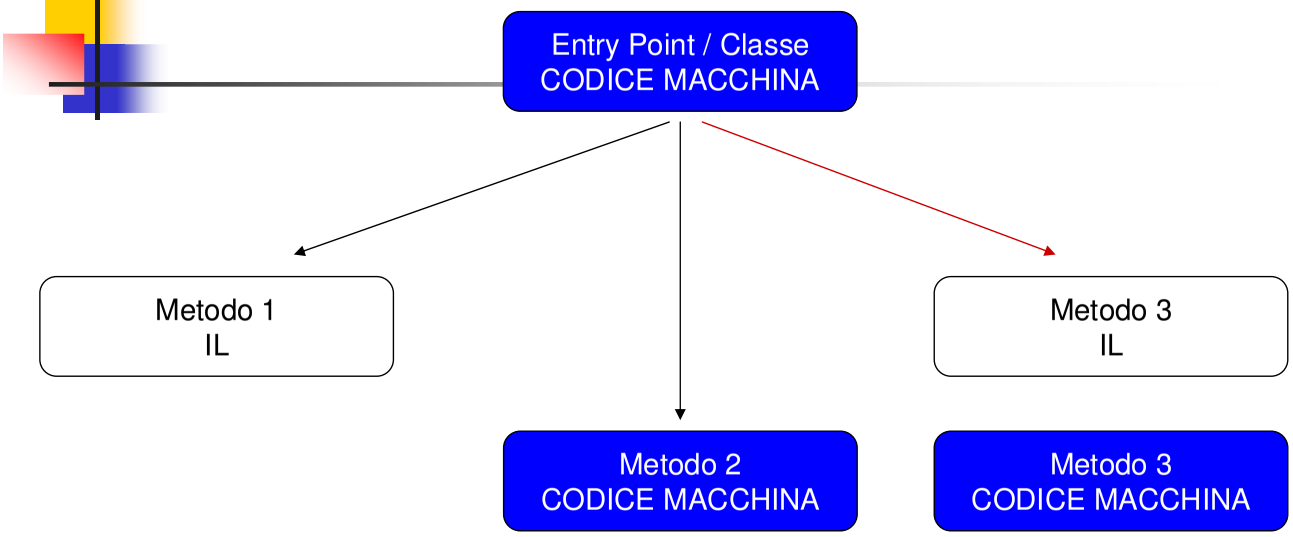
Just In Time Compiler (JIT)



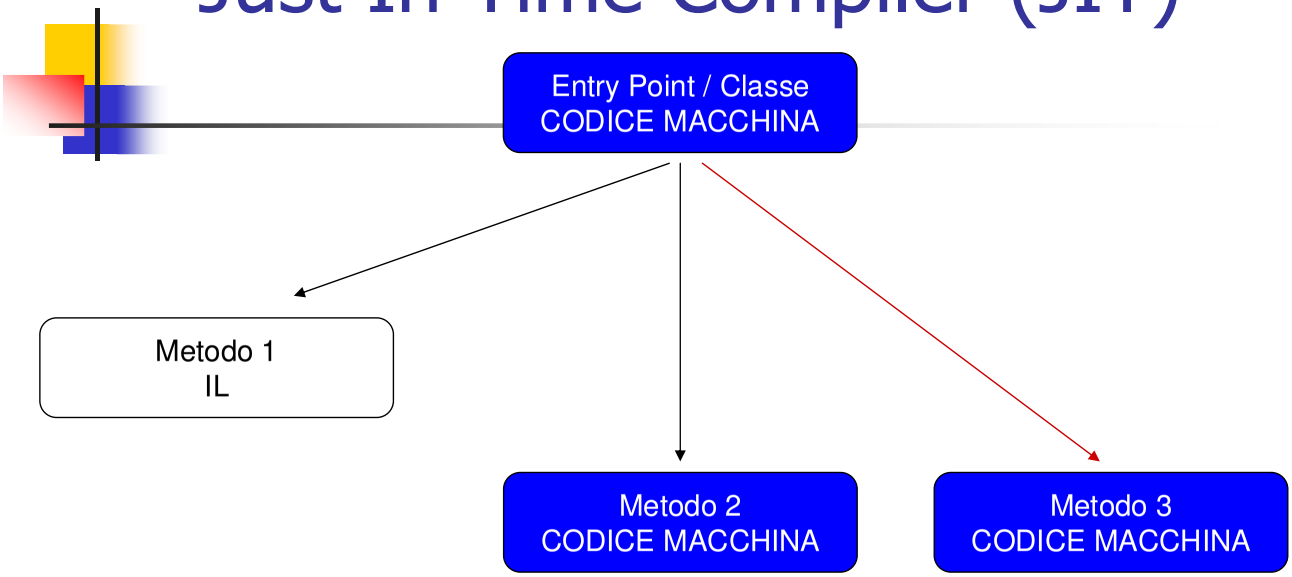
Just In Time Compiler (JIT)



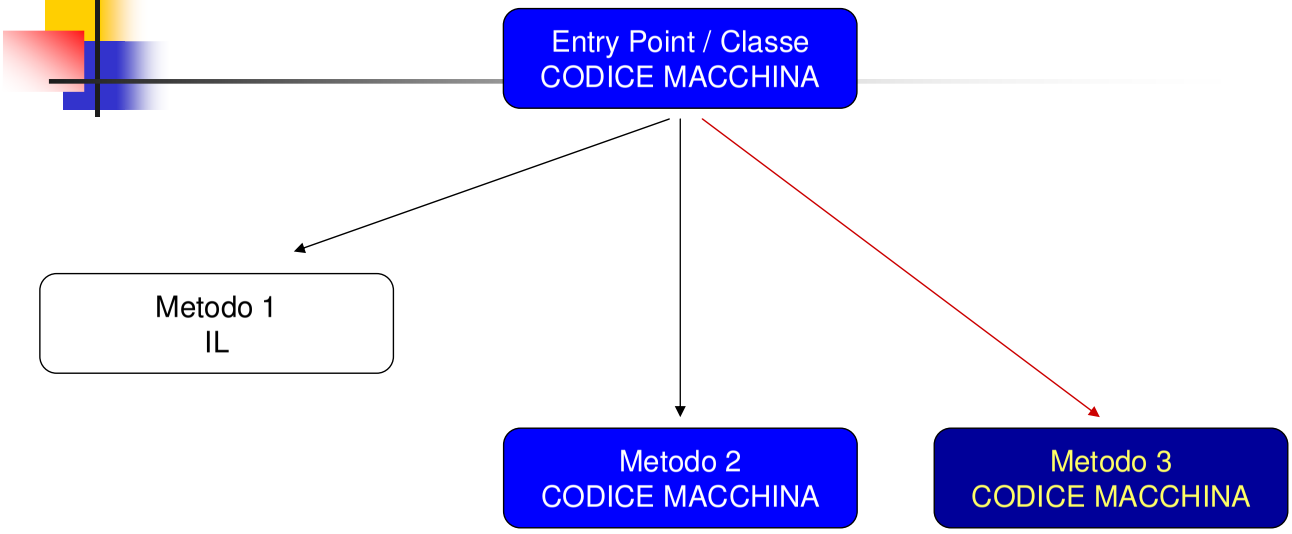
Just In Time Compiler (JIT)



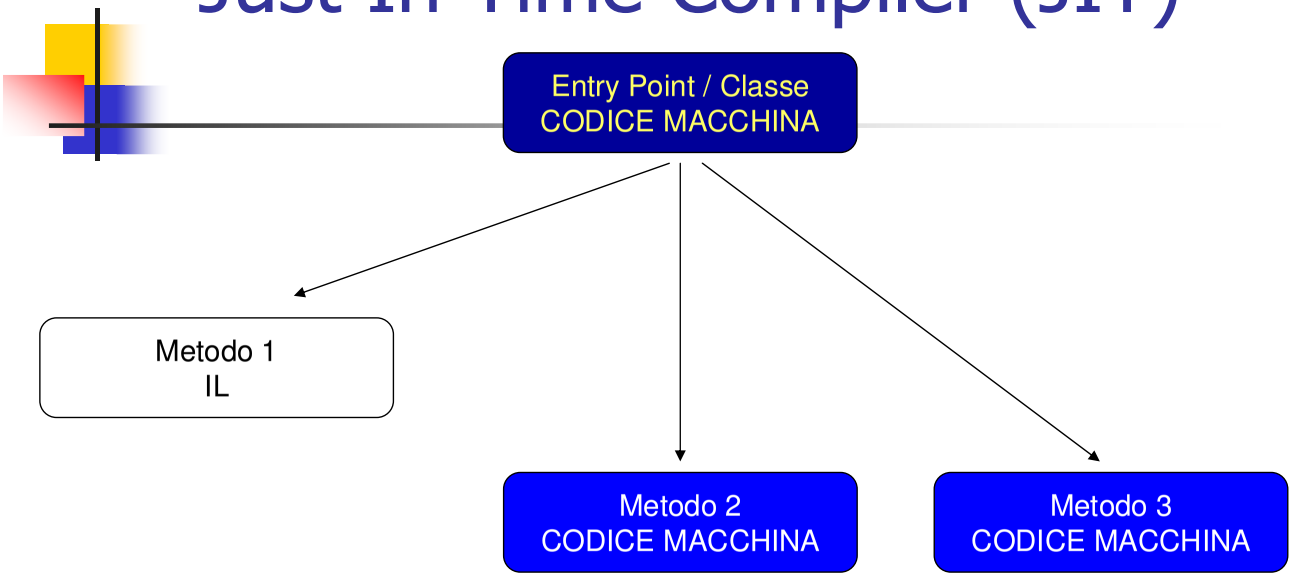
Just In Time Compiler (JIT)



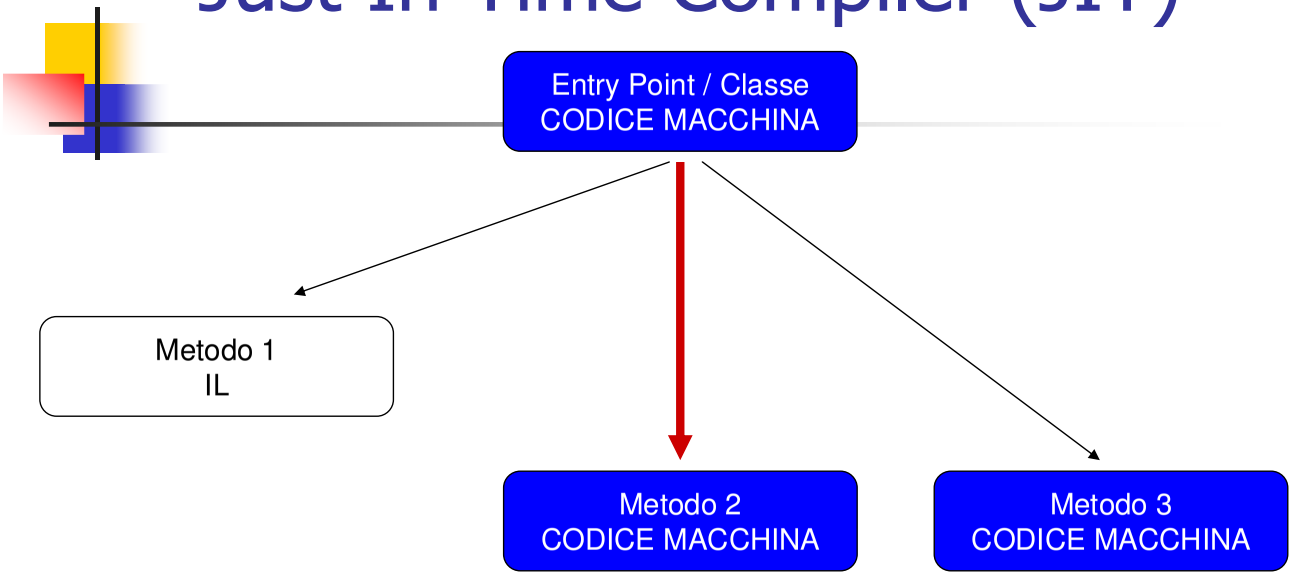
Just In Time Compiler (JIT)



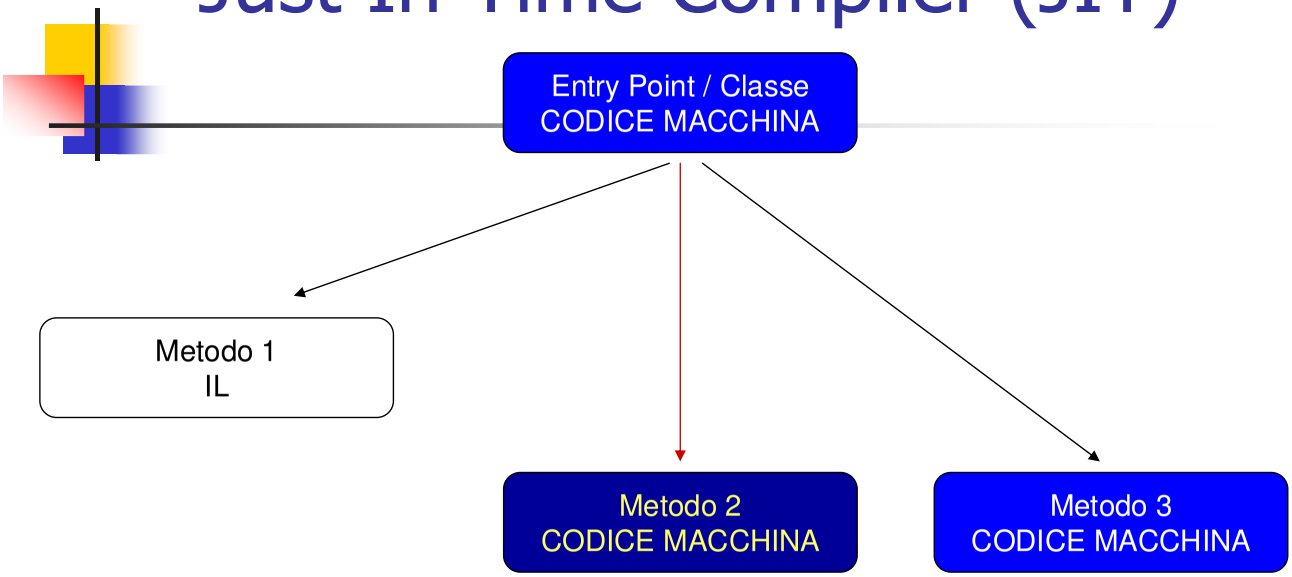
Just In Time Compiler (JIT)



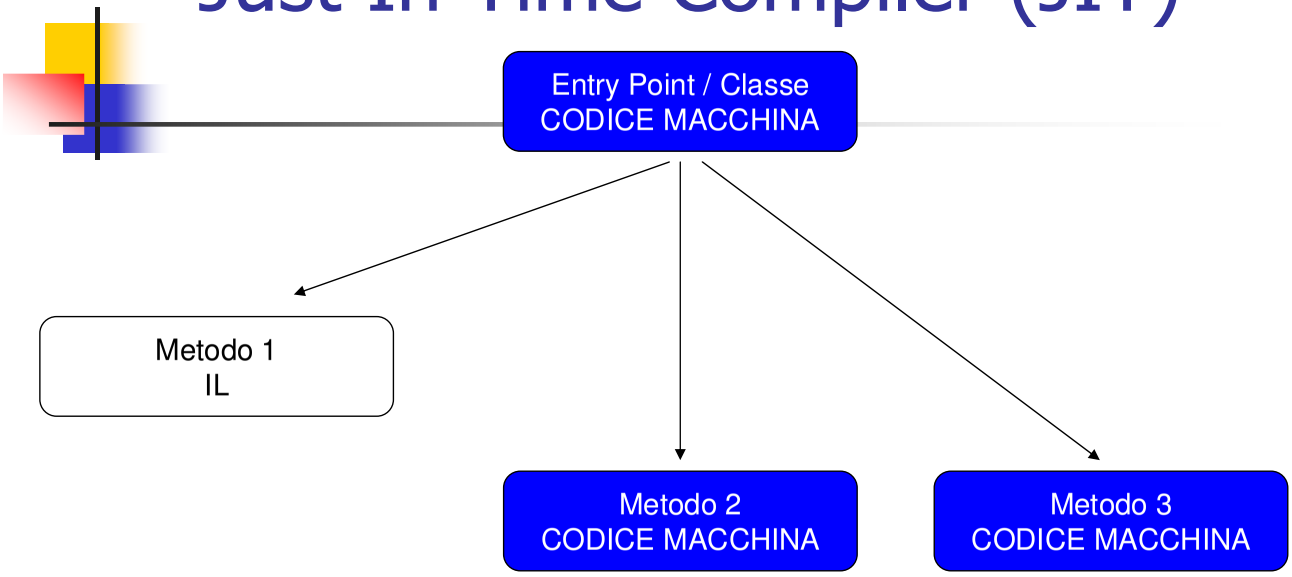
Just In Time Compiler (JIT)

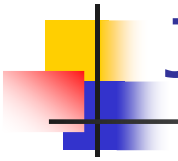


Just In Time Compiler (JIT)



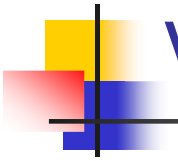
Just In Time Compiler (JIT)





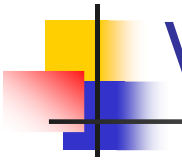
Just In Time Compiler

- The standard JIT compiler runs *on demand*. When a method is called, the JIT compiler analyzes the IL and produces highly efficient machine code, which runs very fast. The JIT compiler is smart enough to recognize when the code has already been compiled, so as the application runs, compilation happens only as needed. As .NET applications run, they tend to become faster and faster, as the already compiled code is reused.
- Dati per la compilazione contenuti nello stesso file del codice (metadati)
- Compilazione ottimizzante perché conosce lo stato preciso dell'ambiente di esecuzione



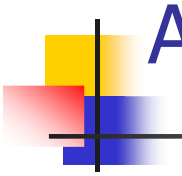
Virtual Execution System

- E' l'ambiente di esecuzione (macchina virtuale) del CLR.
  - Il VES carica, realizza i link ed esegue i programmi scritti per il Common Language Runtime contenuti nei file Portable Executable.
  - Il VES adempie alle sue funzioni di Loader utilizzando le informazioni contenute nei metadati ed utilizza late binding per integrare moduli compilati separatamente, che possono essere anche scritti in linguaggi differenti.
  - Il VES inoltre fornisce servizi durante l'esecuzione dei codici, che includono la gestione automatica della memoria, supporto per debugging, sandbox per la sicurezza analoghe a quelle Java e l'interoperabilità con il codice non gestito come ad esempio componenti COM.



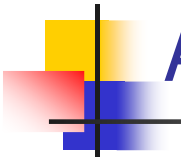
Virtual Execution System

- CLR VES supports code managing, debugging, exceptions:
  - Code execution is managed inside the CLR, therefore it is under its complete control
  - Debugging an application and tracing through its source code is simpler and can be reused by any .NET language or tool
  - Exceptions thrown during code execution can be caught at whatever level they occurred because the Exception Manager is inside the CLR and can be displayed in any debugger since the Debug Engine is also located inside the CLR. Exceptions are available for **all languages**



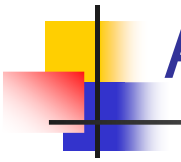
Applicazione .NET

- Il VES opera sui singoli file in formato PE, denominati Assembly. Tali file sono organizzati in Application Domain, che sono processi leggeri i cui confini permettono di implementare la policy di sicurezza di un gruppo di Assembly. Più AD possono girare come unico processo Win32:
- Applicazione 1 (Processo Win32 Separato)
  - Application Domain 1
  - Application Domain 2
    - Assembly 1
    - Assembly 2
    - Assembly 3
- Applicazione 2 (Processo Win32 Separato)
- Applicazione 3 (Processo Win32 Separato)



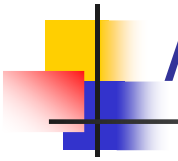
Application Domain

- Sono i processi leggeri del CLR
  - Possono essere immaginati come una fusione della Sandbox di Java e del modello a Thread
  - "Leggeri" perché **più AD sono eseguiti in un unico processo Win32**, ma con meccanismi di sicurezza ed isolamento



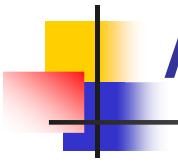
Application Domain

- Modello di sicurezza
  - Controllo di sicurezza in fase di compilazione
  - Ogni applicazione può avere application domain multipli associata con essa, ed ognuno di questi ha un file di configurazione contenente i permessi di sicurezza. Queste informazioni di configurazione sono utilizzate dal CLR per fornire un sistema di sicurezza tipo sandbox analogo a quello presente in Java.



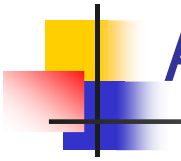
Application Domain

- Modello di sicurezza
  - Nonostante più application domain possano essere eseguiti in un unico processo, **nessuna chiamata diretta è permessa tra metodi di oggetti presenti in differenti application domain.** In alternativa un meccanismo di tipo proxy è utilizzato per isolare lo spazio dei codici.



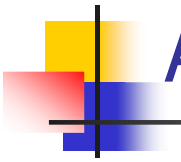
Assembly

- È una collezione di funzionalità sviluppate e distribuite come una singola unità applicativa (uno o più file).
- In pratica è una raccolta di codice compilato.
- Completamente autodescrittivo grazie al suo manifesto.
- Installazione di tipo XCOPY (non è necessaria la registrazione come accade nelle DLL di Windows).



Assembly

- Il manifesto è un metadato che:
  - Stabilisce l'identità dell'assembly in termini di nome, versione, livello di condivisione tra applicazioni diverse, firma digitale.
  - Definisce quali file costituiscono l'implementazione dell'assembly.
  - Specifica le dipendenze in fase di compilazione da altri assembly.
  - Specifica i tipi e le risorse che costituiscono l'assembly, inclusi quelli che vengono esportati dall'assembly.
  - Specifica l'insieme dei permessi necessari al corretto funzionamento dell'assembly.



Assembly

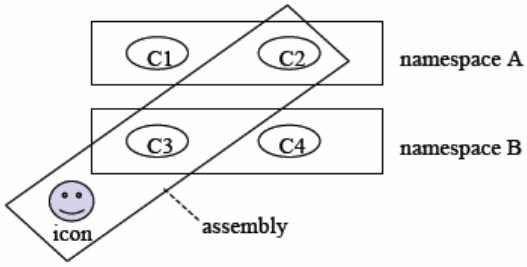
- Il manifesto è parte indissolubile dell'assembly ed è compreso nello stesso file.
- E' il CLR che si preoccupa che le dipendenze espresse nel manifesto siano verificate ed eventualmente si occupa di "ripararle"

Assembly

- Il runtime è in grado di eseguire due versioni diverse della stessa componente side-by-side.
- Il runtime è in grado di rendere disponibile due versioni diverse della stessa libreria
- Nessuna registrazione necessaria

Assembly

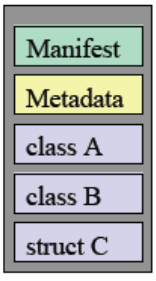
Run time unit consisting of types and other resources (e.g. icons)



An assembly is a
- Unit of deployment: assembly is smallest unit that can be deployed individually
- Unit of versioning: all types in an assembly have the same version number

Often: 1 assembly = 1 namespace = 1 program
But:
- one assembly may consist of multiple namespaces
- one namespace may be spread over several assemblies
- an assembly may consist of multiple files, held together by a *manifest* ("table of contents")

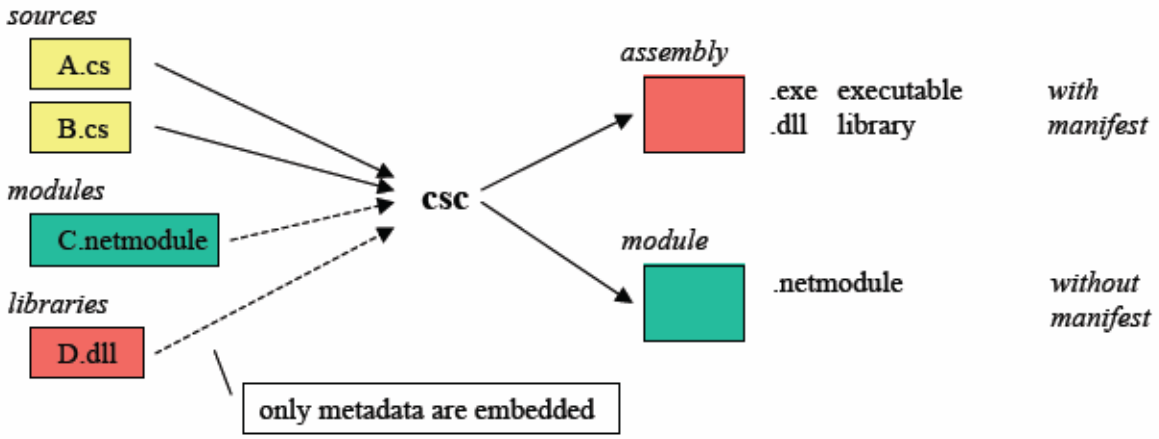
Assembly ≈ JAR file in Java
Assembly ≈ Component in .NET



Assembly

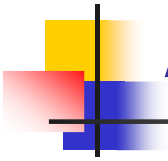
- Come viene creato un assembly?

Every compilation creates either an *assembly* or a *module*



Assembly

- Modules are created as DLLs and are the constituent pieces of assemblies. **Standing alone, modules cannot be executed**; they must be combined into assemblies to be useful.
- **Other modules can be added** with the assembly linker (al)
- Difference to Java: Java creates a *.class file for every class
- an assembly must have **exactly one entry point** -- DLLMain, WinMain, or Main. DLLMain is the entry point for DLLs, WinMain is the entry point for Windows applications, and Main is the entry point for DOS and Console applications.
- **Assemblies form security boundaries as well as type boundaries**. That is, an assembly is the scope boundary for the types it contains, and types cannot cross assemblies. You can, of course, refer to types across assembly boundaries by adding a reference to the required assembly, either in the Integrated Development Environment (IDE) or on the command line, at compile time. What you cannot do is have the definition of a type span two assemblies.



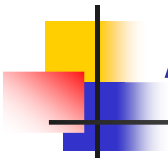
Assembly

- Diverse sono le opzioni del compilatore per la generazione di assembly

/t[arget]: exe	output file = console application (default)
| winexe	output file = Windows GUI application
| library	output file = library (DLL)
| module	output file = module (.netmodule)

/out:*name*	specifies the name of the assembly or module
	default for /t:exe *name*.exe, where *name* is the name of the source file containing the *Main* method
	default for /t:library *name*.dll, where *name* is the name of the first source file
	Example: csc /t:library /out:MyLib.dll A.cs B.cs C.cs

/doc:*name*	generates an XML file with the specified name from /// comments



Assembly

- Diverse sono le opzioni del compilatore per la generazione di assembly

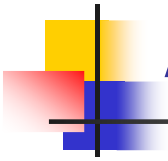
/r[eference]:*name*	makes metadata in *name* (e.g. *xxx.dll*) available in the compilation. *name* must contain metadata.

/lib:dirpath{,dirpath}	specifies the directories in which libraries are searched that are referenced by /r.

/addmodule:name {,name}	adds the specified modules (e.g. *xxx.netmodule*) to the generated assembly. At run time these modules must be in the same directory as the assembly to which they belong.

Example

```
csc /r:MyLib.dll  /lib:C:\project   A.cs B.cs
```



Assembly

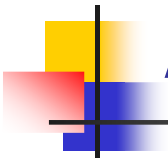
- Diverse sono le opzioni del compilatore per la generazione di assembly

```
csc A.cs                          => A.exe
csc A.cs B.cs C.cs                => B.exe  (if B.cs contains a Main method)
csc /out:X.exe A.cs B.cs          => X.exe

csc /t:library A.cs               => A.dll
csc /t:library A.cs B.cs          => A.dll
csc /t:library /out:X.dll A.cs B.cs => X.dll

csc /r:X.dll A.cs B.cs            => A.exe (where A or B reference types in X.dll)

csc /addmodule:Y.netmodule A.cs   => A.exe (Y is added to this assembly;
                                     but Y.netmodule remains as a separate file)
```

Assembly

- Caricamento di assembly:

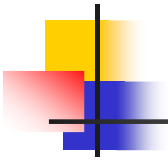
Executables are loaded when a program is invoked from the shell
(e.g., invocation of *MyApp* loads *MyApp.exe* and executes it)

Libraries (DLLs) are searched in the following directories:

- in the application directory
- in all directories that are specified in a configuration file (e.g. *MyApp.exe.config*) under the <probing> tag

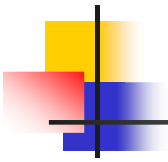
```
<configuration>
   ...
   <runtime>
      ...
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <probing privatePath="bin;bin2\subbin;bin3"/>
      </assemblyBinding>
   </runtime>
</configuration>
```

- in the Global Assembly Cache (for "shared assemblies")



Private vs Shared Assemblies

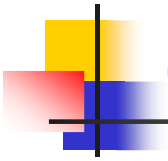
- Assemblies come in **two flavors: private and shared**. Private assemblies are intended to be used by only one application; shared assemblies are intended to be shared among many applications.
- A private assembly can have any name you choose. It does not matter if that name clashes with assemblies in another application; the names are local only to a single application. In the past, DLLs were installed on a machine and an entry was made in the Windows Registry. It was difficult to avoid corrupting the Registry.
- If you want to share your assembly, it must meet three requirements.
  - First, your assembly must have a **strong name**. Strong names are globally unique.
  - Second, your shared assembly must be protected against newer versions trampling over it, and so it must have **version control**.
  - Finally, to share your assembly, place it in the **Global Assembly Cache** (GAC). This is an area of the filesystem set aside by the Common Language Runtime (CLR) to hold shared assemblies.



Private vs Shared Assemblies

Private Assembly
- is used by only one application
- resides in the application directory
- does not have a "strong name"
- cannot be signed

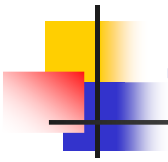
Public Assembly (or shared assembly)
- can be used by all applications
- resides in the Global Assembly Cache (GAC)
- has a "strong name"
- must be signed in order to get a strong name
- GAC can hold assemblies with the same name and different version



Shared Assemblies

Strong Names:

- Strong names must be globally unique and use public key encryption to ensure that the assembly hasn't been tampered with and was written by the creator. A strong name is a string of hexadecimal digits and is not meant to be human-readable.
- To create a strong name, a public-private key pair is generated for the assembly. A hash is taken of the names and contents of the files in the assembly. The hash is then encrypted with the private key for the assembly and placed in the manifest. This is known as *signing the assembly*. The public key is incorporated into the strong name of the assembly.



Shared Assemblies

- You can **create a strong name** with the sn utility:

  sn -k c:\myStrongName.snk
- The -k flag indicates that you want a new key pair written to the specified file. You can call the file anything you like. Remember, a strong name is a string of hexadecimal digits and is not meant to be human-readable.
- You can **associate strong name with your assembly** by using an attribute (placed at the beginning of your file):

  using System.Runtime.CompilerServices;

  [assembly: AssemblyKeyFile("c:\myStrongName.key")]

Shared Assemblies

- Strong Names:

Consist of 4 parts

- the *name* of the assembly (e.g. A.dll)
- the *version number* of the assembly (e.g. 1.0.1033.17)
- the *culture* of the assembly (System.Globalization.CultureInfo)
- the *public key* of the assembly

can be set with attributes

```
using System.Reflection;
[assembly:AssemblyVersion("1.0.1033.17")]
[assembly:AssemblyCulture("en-US")]
[assembly:AssemblyKeyFile("myKeyFile.snk")]
class A {
   ...
}
```

default: 0.0.0.0
default: neutral

Shared Assemblies

- Signing Assemblies

1. Generate a key file with *sn.exe* (Strong Name Tool)

```
sn  /k  myKeyFile.snk
```

myKeyFile.snk is generated and contains:
- public key (128 bytes)
- private key (436 bytes)

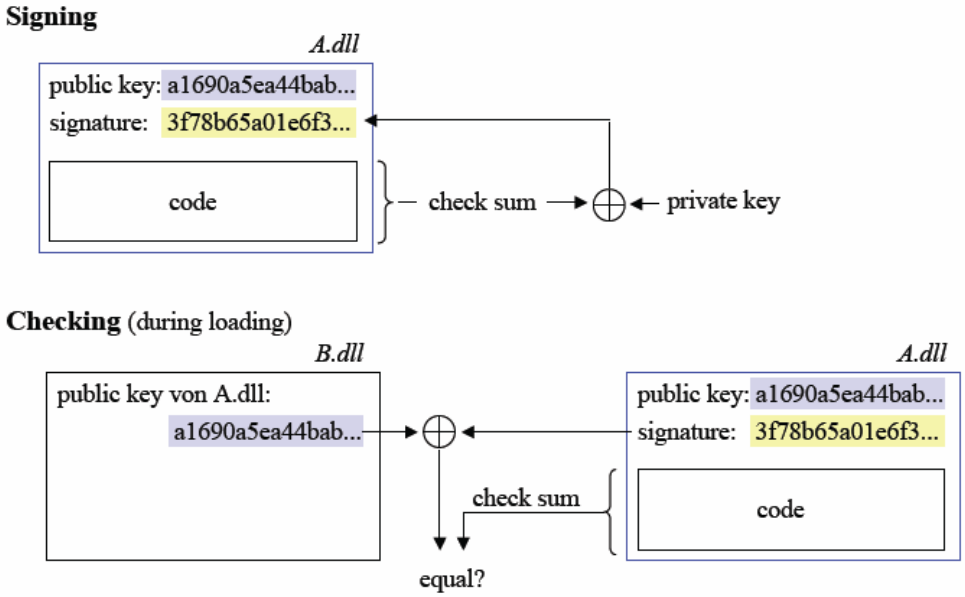
2. Sign the assembly with the AssemblyKeyFile attribute

```
[assembly:AssemblyKeyFile("myKeyFile.snk")]
public class A {...}
```

During compilation
- the assembly is signed with the private key
- the public key is stored in the assembly

Shared Assemblies

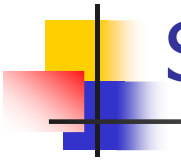
- Signing Assemblies



Shared Assemblies

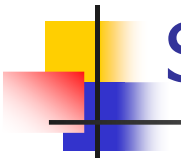
Version control:

- A version number for an assembly might look like this: 1:0:2204:21 (four numbers, separated by colons). The first two numbers (1:0) are the major and minor version. The third number (2204) is the build, and the fourth (21) is the revision.
  - When two assemblies have different major or minor numbers, they are considered to be incompatible.
  - When they have different build numbers, they might or might not be compatible
  - when they have different revision numbers, they are considered *definitely* compatible with each other.
- Revision numbers are intended for bug fixes. If you fix a bug and are prepared to certify that your DLL is fully backward-compatible with the existing version, you should increment the revision.
- Mantenendo traccia delle versioni, si risolve il problema delle DLL, in realta' uno dei motivi principali dell'introduzione del .NET



Shared Assemblies

- Global Assembly Cache (GAC)
  - Memoria per gli assembly "sicuri".
    - Gestione riservata agli amministratori
    - Eseguiti fuori dalla "Sandbox", maggiori privilegi di accesso alle risorse
- Downloaded Assembly Cache (DAC)
  - Memoria per gli assembly transitori e/o "insicuri".
    - Assembly esterni, ad esempio scaricati dalla rete.
    - Eseguiti nella "Sandbox" più lenti e con minor accesso alle risorse



Shared Assemblies

- Dopo aver creato un assembly da condividere ed averlo firmato ed associato con uno strong name, si deve mettere quindi nella GAC
- You can do that with the gacutil utility:

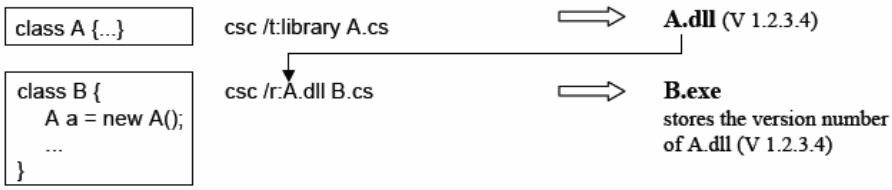
  gacutil /i MySharedAssembly.dll
- Or you can open your File Explorer and drag your assembly into the GAC. To see the GAC, open the File Explorer and navigate to *%SystemRoot%\assembly*; Explorer turns into a GAC utility

Shared Assemblies

- Gestione delle versioni nel caricamento di assembly:

Causes those libraries to be loaded, which have the expected version number

Version number is stored in the assembly by the compiler



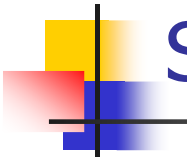
Version number is checked when the assembly is loaded

Call: B
- loads B.exe
- finds a reference to A.dll (V 1.2.3.4)
- loads A.dll in version V 1.2.3.4
(even if there are other versions of A.dll)

avoids "DLL hell"

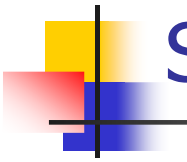
Shared Assemblies

- Cosa significa "evitare il DLL Hell?"
- una **dynamic-link library** è una libreria software che viene caricata dinamicamente a run-time. Queste librerie sono note con l'acronimo **DLL** nei sistemi Windows, o anche con il termine **shared library** nei sistemi Unix (".so").
- Vantaggi di uso di DLL:
- permettono di **spezzare i programmi** in parti concettualmente separate, che verranno caricate solo se necessarie.
- una singola libreria può essere **caricata in memoria una sola volta** e **condivisa** da più programmi.
- Sono possibili **installazioni parziali** di un sistema software, in cui sono effettivamente presenti sulla memoria di massa solo le librerie associate alle funzioni che l'utente desidera.
- È possibile **aggiornare un programma modificando solo le DLL**: inserendo una versione diversa della DLL, che contiene ad esempio dei *bug fix*, tutti i programmi che la usano saranno automaticamente "aggiornati" senza bisogno di essere ricompilati



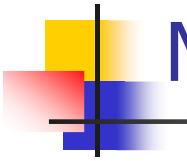
Shared Assemblies

- Il principale svantaggio è legato al fatto che una nuova versione di una DLL potrebbe effettuare dei cosidetti *breaking changes* in modo volontario o a causa di bug. Un breaking change è una modifica del comportamento di una funzione che la rende non più compatibile con le convenzioni in uso (ad esempio, una funzione che prima ritornava NULL in caso di errore nei parametri e ora setta errno e ritorna un valore non nullo). Ancora più critico il caso in cui un programma di installazione sovrascriva una DLL con una versione più vecchia. Questi problemi sono chiamati in gergo "DLL Hell" (**inferno delle DLL**).
- Example: you install Application A, and it loads some DLLs into your Windows directory. It works great for months. You then install Application B, and unexpectedly Application A breaks. Application B is no related to A. So what happened? It turns out, you later learn, that Application B replaced a DLL that Application A needed, hence A begins to stagger about, blind and senseless.
- This phenomenon led customers to be justifiably leery of installing new software, or even of upgrading existing programs, and it is one of the reasons **Windows machines are perceived to be unstable**.



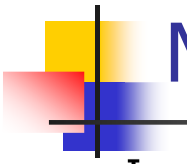
Shared Assemblies

- Possibili soluzioni al DLL Hell:
- **statically link** against all the libraries. This is common in C/C++ applications, though the main purpose of DLLs (runtime library sharing between programs) is sacrificed.
- **prevents unauthorized applications** from overwriting DLLs, but the problem still remains when invoking Windows update
- Nei sistemi Unix-like, è possibile far **convivere versioni diverse** fra loro incompatibili di una stessa libreria, purché entrambe siano presenti sul Filesystem in differenti percorsi e sia possibile in fase di collegamento del programma l'identificazione della versione da utilizzare.
- In .NET platform, **shared assemblies in .NET are uniquely identified** by their names and their versions. The GAC allows for "side-by-side" versions in which an older version of an assembly is available alongside a newer version. This allows particular applications to say "give me the newest" or "give me the latest build of Version 2," or even "give me only the version I was built with."



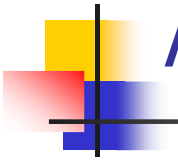
Multi-module Assemblies

- A **single-module assembly** has a single file that can be an EXE or DLL file. This single module contains all the types and implementations for the application. The assembly manifest is embedded within this module.
- A **multi-module assembly** consists of multiple files (zero or one EXE and zero or more DLL files, though you must have at least one EXE or DLL). The assembly manifest in this case can reside in a standalone file, or it can be embedded in one of the modules. When the assembly is referenced, the runtime loads the file containing the manifest and then loads the required modules as needed.
- Multi-module assemblies have advantages for real-world programs, especially if they are developed by multiple developers or are very large



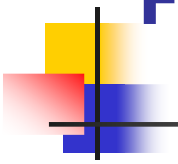
Multi-module Assemblies

- Imagine that 25 developers are working on a single project. Each of them develops a separate module, each in its own DLL. The person responsible for building the application would then create a 26th module with the manifest for the entire assembly. These 26 files can be deployed to the end user. The end user then need only load the one module with the manifest, and he can ignore the other 25. The manifest will identify which of the 25 modules has each method, and the appropriate modules will be loaded as methods are invoked. This will be transparent to the user.
- As modules are updated, the programmers need only to send the updated modules (and a module with an updated manifest). Additional modules can be added and existing modules can be deleted; the end user continues to load only the one module with the manifest.
- Multi modules cannot be created visually with Visual Studio; they need a makefile to be run



Assembly

- Il CLR fornisce anche API utilizzate dai motori di scripting che creano assembly dinamici durante l'esecuzione degli script; questi assembly sono eseguiti direttamente senza essere salvati su disco.



Framework Classes Library (FCL)

- **FCL** is one of the largest class libraries in history and one that provides an object-oriented API to all the functionality that the .NET platform encapsulates.
- FCL comprende le classi base, quelle per il supporto alla gestione dei dati (DB e XML) e le classi per web services, forms e windows forms
- With more than 4,000 classes, the FCL facilitates rapid development of desktop, client/server, and other web services and applications.
- The set of **framework base classes**, the lowest level of the FCL, is similar to the set of classes in Java. These classes support rudimentary input and output, string manipulation, security management, network communication, thread management, text manipulation, reflection and collections functionality.

Framework Classes Library (FCL)

- Le classi per XML e DB support persistent management of data that is maintained on backend databases.
- These classes include the Structured Query Language (SQL) classes to let you manipulate persistent data stored through a standard SQL interface.
- Additionally, a set of classes called ADO.NET allows you to manipulate persistent data.
- The .NET Framework also supports a number of classes to let you manipulate XML data and perform XML searching and translations.

Framework Classes Library (FCL)

- Extending the framework base classes and the data and XML classes is a tier of classes geared toward building applications using **three different technologies**: Web Services, Web Forms, and Windows Forms.
- **Web services** include a number of classes that support the development of lightweight distributed components, which will work even in the face of firewalls and NAT software. Because web services employ standard HTTP and SOAP as underlying communications protocols, these components support plug-and-play across cyberspace.
- **Web Forms** and **Windows Forms** allow you to apply Rapid Application Development (**RAD**) techniques to building web and Windows applications. Simply drag and drop controls onto your form, double-click a control, and write the code to respond to the associated event.