# *The .NET Framework*

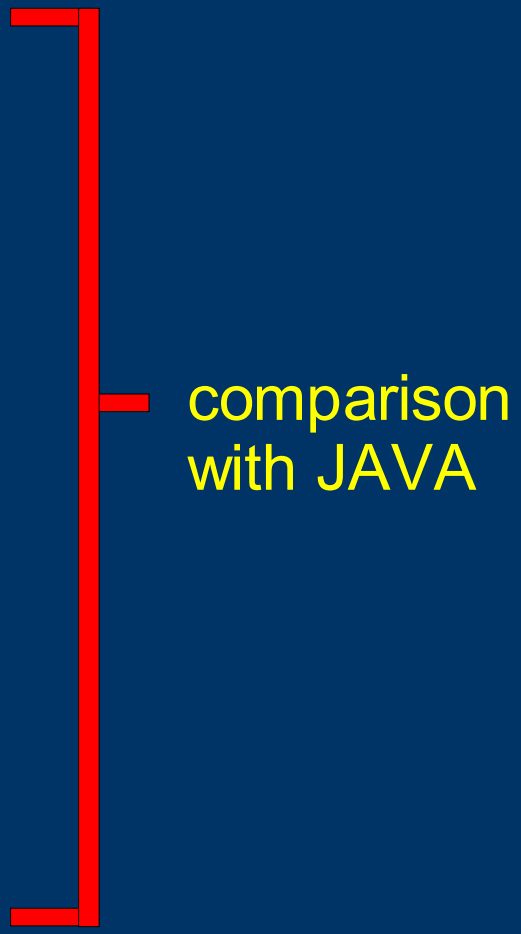
## Short Introduction of the Microsoft .NET Framework

**Dott. Mario Di Raimondo**
Dipartimento di Matematica e Informatica
Università degli Studi di Catania

contents from
Microsoft Research Academic Days 2003
“.NET Mobile and Distributed Technologies”
(Turin, 29-30 September, 1st October)

# Overview

- Motivations and Targets
- A look at the .NET Framework
- The supported languages
- Mention about C#
- Microsoft and the standards
- Conclusions



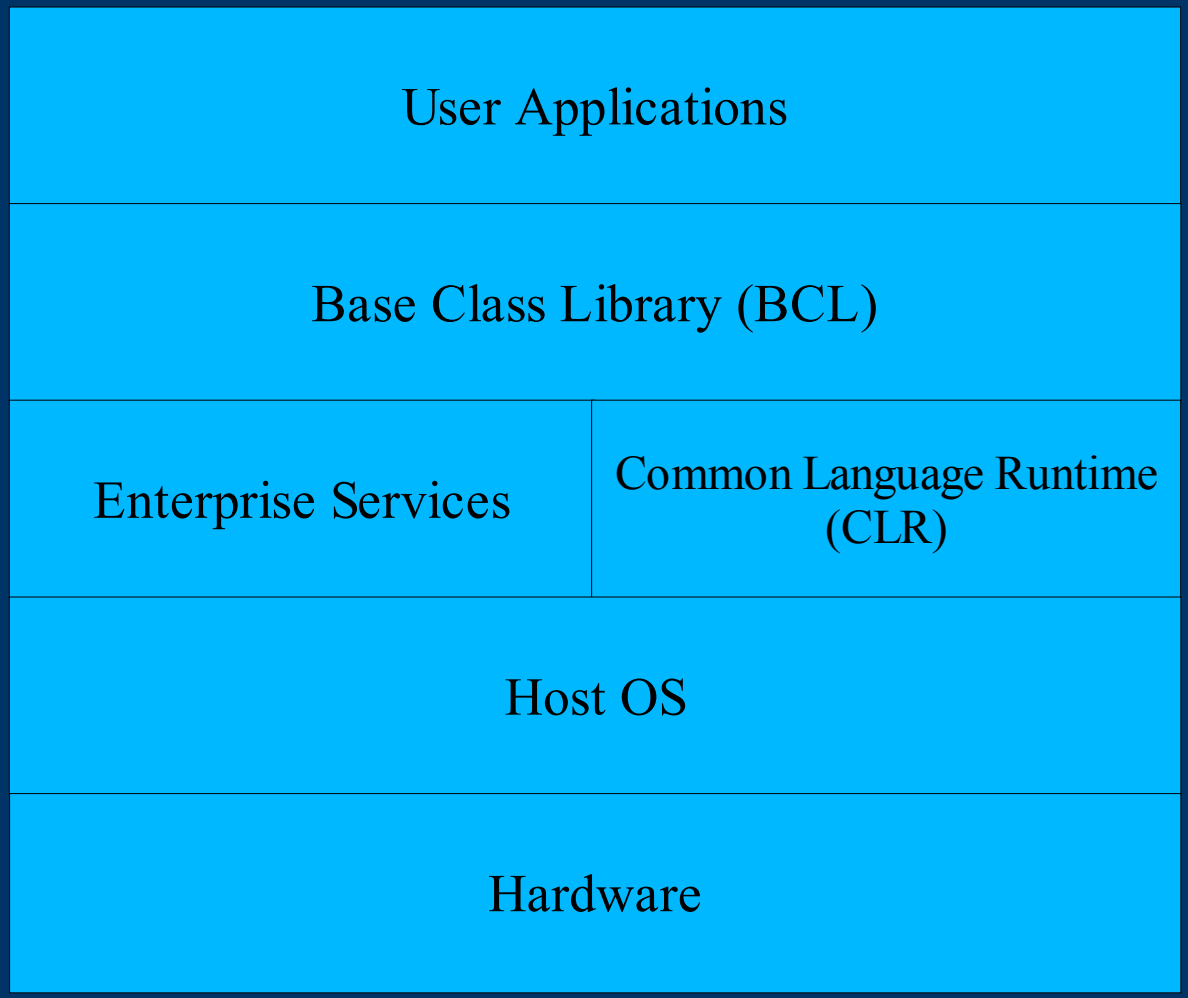
# .NET

- it's a “software platform” on which develop applications
- characteristics:
  - projected for the applications distributed through the net
  - multi-language
  - multi-platform (?!?)
  - semi-standardized

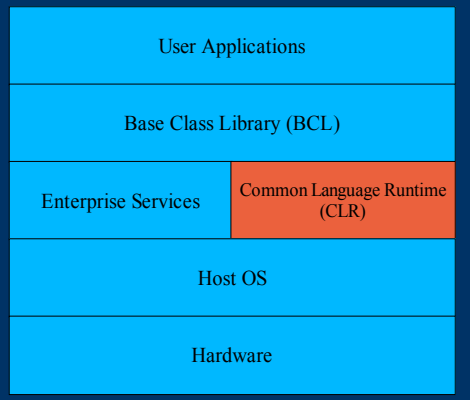
# Motivations

- we can't define it as a clone of Java...
- ... but we can certainly consider it as the answer of Microsoft to Java! (or more precisely, to the J2EE infrastructure of Sun)
  - insinuations...
- simplify the life of people that develop complex applications
  - Visual-Basic: little expressive
  - Visual-C: too complex
- make the applications portable (as in Java)
- rearrange the immense (and untidy) MS libraries

# Architecture



# Common Language Runtime (CLR)

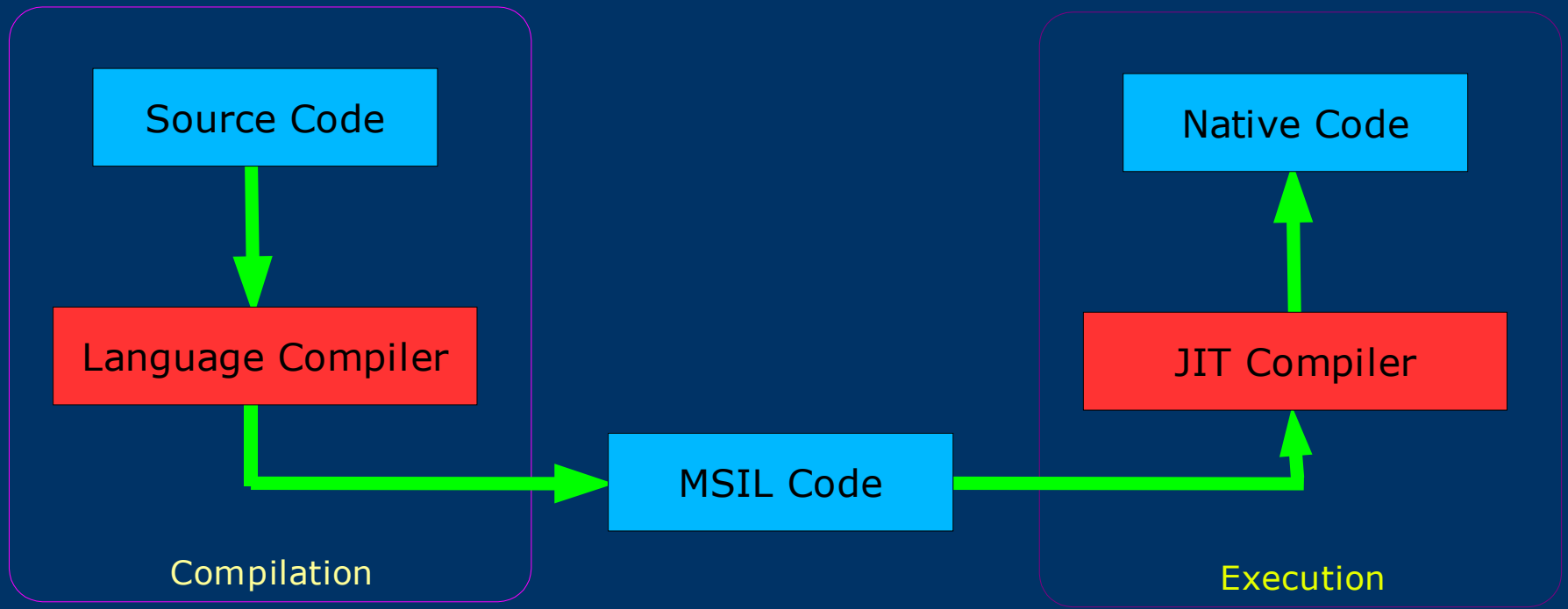



- the core of the .NET architecture
- environment independent from the platform for the execution of the applications
- the equivalent of the Java Virtual Machine (JVM) in the Java architecture
- offers a series of basic functionalities:
  - memory management (Garbage collection)
  - concurrency management
  - verification on the types of the data
  - security controls during the loading and the execution of the modules (can be disabled)
  - controls of reading/writing rights on the memory areas (can be disabled or managed with a careful right management)
  - multi-versioning management... end of the DLL hell?!
- unlike Java, it doesn't depend on a specific language

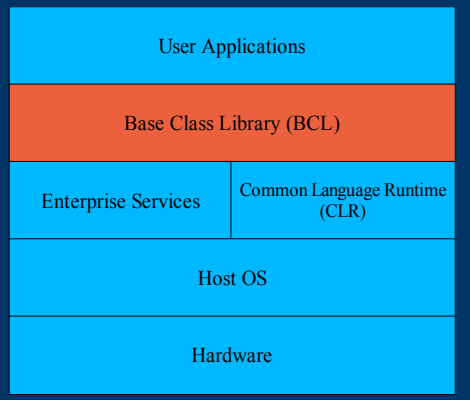
# *Microsoft Intermediate Language (MSIL) and JIT*

- the intermediate language in which the .NET applications are compiled
- the equivalent of the byte-code of the JVM
- it has an higher level than the byte-code (for example: operations without types), then:
  - the interpretation isn't provided by the system
  - it's compiled in native code at execution time (JIT)
    - **“normal” JIT**: translates the MSIL code in native code applying several optimizations; requires many resources (computational and memory)
    - **EconoJIT**: translates the MSIL code in native code using basic and fast techniques, without any optimizations; useful for embedded systems or with few resources
    - **PreJIT** (or **NGen** = “Native [code] GENeration”): stores, next to the copy in MSIL code, a copy in native code already optimized; at the moment, it's a technique only partially used in the BCL
    - **OptJIT** (just an idea for the future): optimizes the on-the-fly compilation phase using some additional information embedded in the MSIL; speeds-up this process using some precomputed information
  - at the moment the technologies used in the JIT of Java are more advanced
    - profiling JVM and incremental optimizations

# Execution of the code in .NET

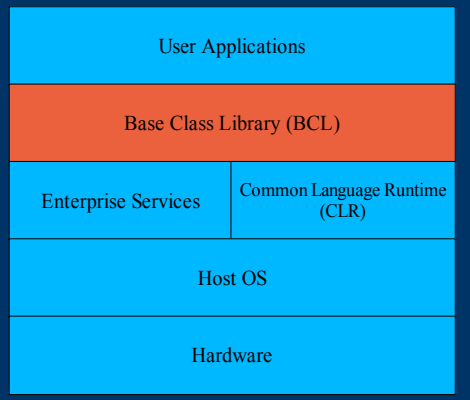


# Base Class Library (BCL)



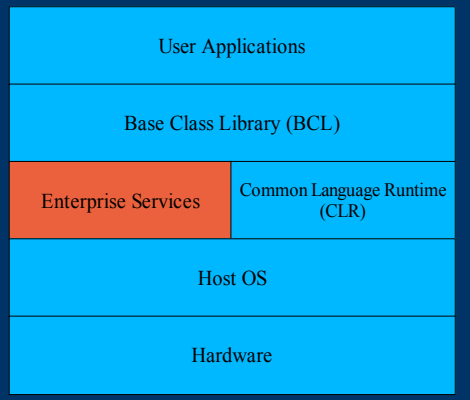

- the set of the base classes of the platform
  - analogue to the Standard Library of C/C++ and to the packages java.* of Java
- encapsulates the main functions of a system:
  - complex data type (collection)
  - networking
  - access to File System
  - user interface
  - security
  - concurrent programming
  - XML
  - etc...

# Base Class Library (BCL)




- the BCL is organized in namespace (like Java) starting from the namespace System
- moreover, there are some typical functions of Windows:
  - access to the system register
  - interoperability with COM (reuse of old components)
  - functionalities of low level
- **Positive aspect:** in this way we have a rearrangement of the whole library set of Windows, providing an homogeneous and coherent way to access to them

# Enterprise Services




- they consist of base services for the realization of .NET applications
- they don't belong to the standard, but they are essential for the development of enterprise applications
- they correspond to MS server applications(SQL server, IIS, BizTalk, etc...) and they aren't .NET applications (so not portable)
  - GUI (WinForms)
  - Web Applications (WebForms and ASP.NET)
  - Access to DB (ADO.NET)
  - Transactions (MTS)
  - Scripting (VBScript and Jscript)
  - Web Services

# .NET and the languages

- the .NET architecture, dislike J2EE, isn't related to a specific language: it's **multi-language**
- we can write an application in any supported language, compile it in MSIL and then execute it on any .NET platform
- objects written in one language can be compiled and reused as modules in programs written in a different (but supported) language
  - there are many limitations:
    - unmanaged code
    - dependency on specific Windows libraries
    - ....

# Common Type System (CTS)

- to obtain a multi-language environment we need to establish a maximum common divisor among all the supported languages:
  - **Common Type System (CTS)**
- specifies details as:
  - the primitive types
  - what a class is (or “type” in the .NET slang) and which are their characteristics:
    - methods
    - data fields
    - properties
  - visibility policy
  - inheritance
    - the multi-inheritance is not provided....
    - the interfaces can be used

# Supported languages

Microsoft directly supports:

- C# (a new language...)
- C
- C++
- Visual-Basic
- JScript

Third-parts provide .NET compilers for:

- APL, CAML, Cobol
- Haskell, Mercury
- ML, Oberon, Oz
- Pascal, Perl, Python
- Scheme, Smaltak
- ... and may be Java!

# But do we need all these languages?

- POSITIVE:
  - in this way many matured competence may be “recycled”
- NEGATIVE:
  - too many compromises and limitations (for example: all becomes case-insensitive)
  - they radically changed some languages:
    - C++.... WAS a standard (goodbye multi-inheritance)!
    - Visual-Basic... has practically become another language... it's better to use C#!
- also in Java we can use other languages to produce bytecode, but this isn't a basic functionality as introduced in .NET
- Java is a lone language that you can use in many spheres
  - desktop applications, server applications, web (applet)
  - desktop, server,  embedded devices

# C# languages

**Java:JVM = C#:CLR**

- it's the main language of the .NET architecture
- object-oriented language
- derived from C++ (simplified)
- there are much “syntactic sugar”
- the specification regards only the grammar of the language
- it doesn't got any library but it lean against the BCL of .NET
- .NET is independent on C#, but C# doesn't have any utility without .NET

# Standardization of .NET

- .NET is a technology entirely developed by Microsoft
- the ECMA has standardized the **Common Language Infrastructure** (**CLI**) which:
  - can be considered a “subset” of the CLR specifications
  - contains only a minimum set (a core) of the system libraries
- it seams that other parts of the architecture are in a standardization phase in a near future
- is .NET an open standard?!
  - No, it isn't!
  - .NET is an implementation of some open standards
  - many parts of the framework (as the major part of BCL) are proprietary and not standardized

# Conclusions

- .NET is a project with great ambitions
- Microsoft will use its monopolistic position to impose this technology on the market
- the similarity between Java and .NET can't be unobserved
  - there are some differences
  - but they aren't substantial and prevalent
- there are many doubts about the portability of .NET applications on platforms different from Windows (although there are some brave projects as **Mono** on Linux/Unix systems...)
- let's see if the field experts will reject or accept these deep changes of their development tools/environment