
Microsoft®
.net™
BSS

The .NET strategy

- Microsoft wanted to make the WWW more vibrant by enabling individual devices, computers, and web services to work altogether intelligently to provide rich solutions to the user.
- With the advent of websites on the internet, user can create wide variety of web application as well as web services, that run on a continuous server.
- This web service can be available to any user on the internet regardless of who developed it.
- The software strategy for implementing and delivering these services is known as “.NET”

BSS

So what is .NET?

- .NET is a platform that provides everything as a single package for developing software for web services.

- It integrates presentation technologies, component technologies and data technologies on a single platform, hence,
  - It's just like Windows, except distributed over the Internet.
  - It exports a common interface so that it's programs can be run on any system that supports .NET.

  - If you think of developing most efficient windows application, web application, database application, services, and that to on a single platform, there is only one solution called "MICROSOFT .NET.

The Origin of .NET Technology

- Initially in 1990's monolithic approach in designing software were used.
- It worked until program size was upto one limit.
- Next generation, welcomed COM technology.
- As demand increased, software became too large and complex, hence its maintenance and testing involved lots of problem.
- Problem overcome by COM – using Modularization.
- Each module form one component.
- Individually developed , tested and then integrated.
- Hence overall complexity reduced.
- Ease of maintenance.
- Component can be distributed.

BSS

The Origin of .NET Technology

- Third generation --- .NET
- .NET is again a component model, which provides a new level of inter-operability among different modules.
- COM component are platform dependent, where as .NET component (assemblies) are platform independent.
- Because .NET component are pre-compiled. That is they are compiled into Microsoft Intermediate Language.
- IL is compatible with other IL module. (assembly).
- COM components needs to be registered hence works on only windows.
- .NET components are managed code (memory disposal is done for you)
- Whereas COM component are unmanaged code.

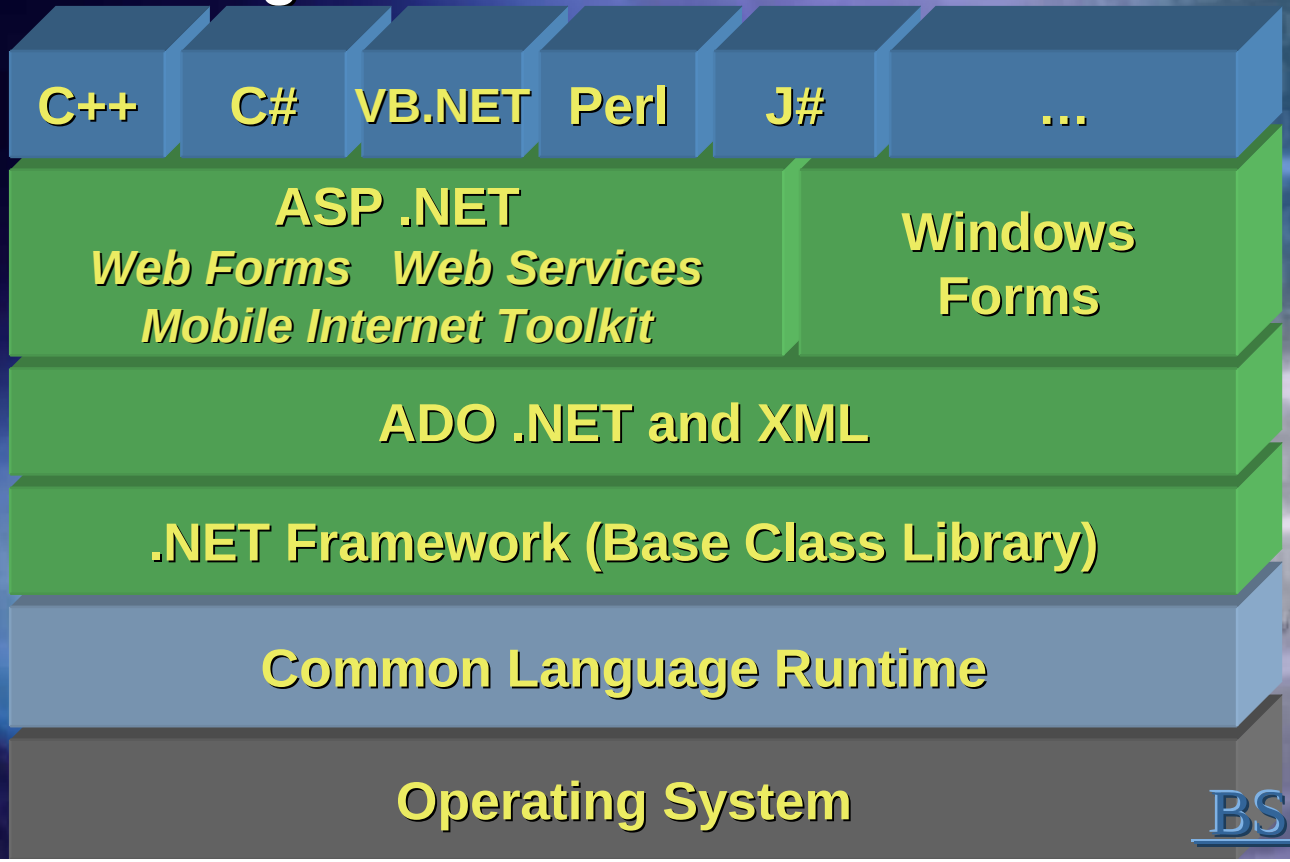
The Origin of .NET Technology

As all modules are converted to IL code. Modules developed under different language can be integrated.

After final integration, one more time code is compiled to generate native code.

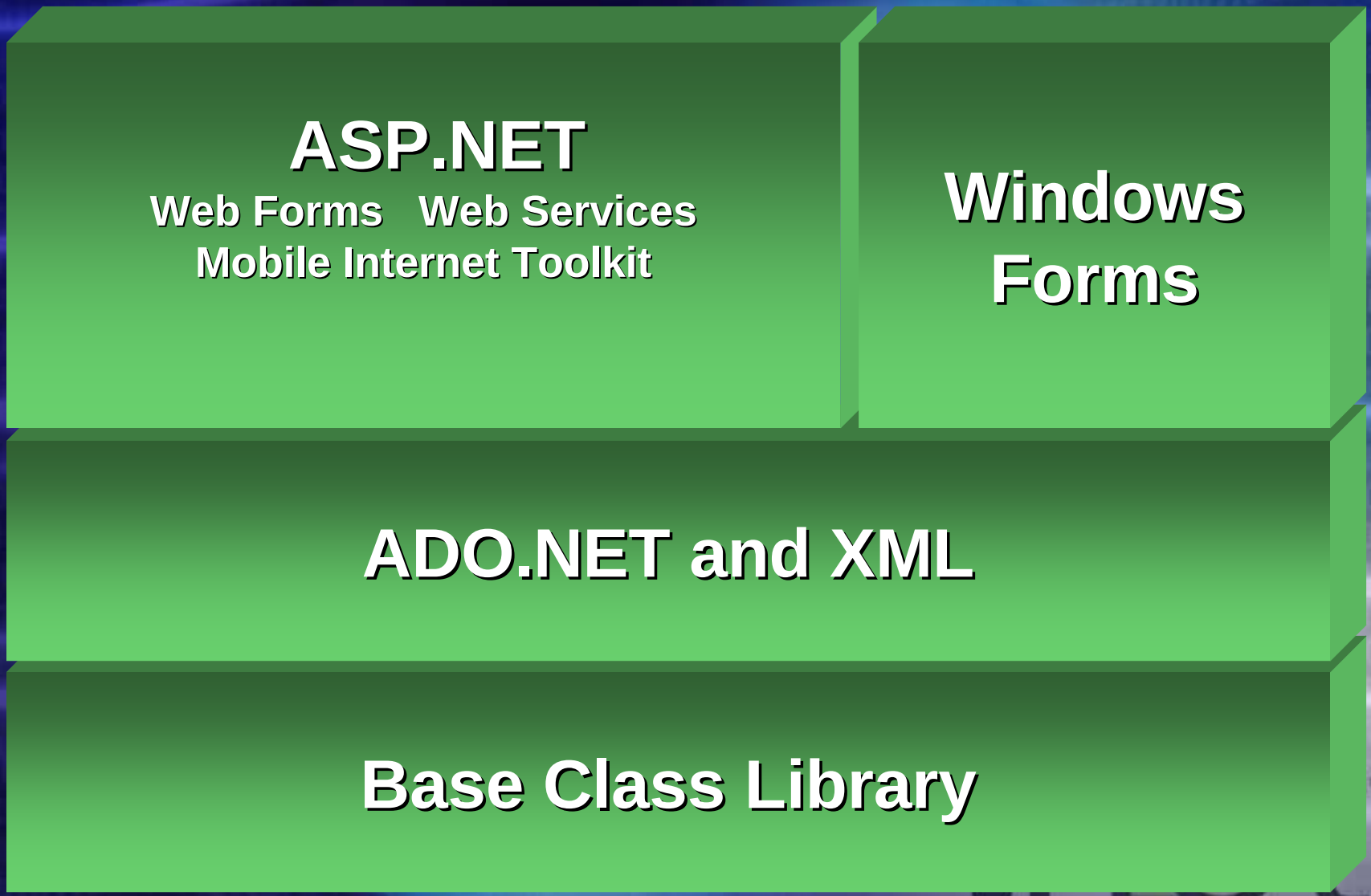
THE .NET FRAMEWORK

- As already said, the .NET platform provides a new environment for creating and running robust (strong), scalable and distributed applications over the WEB.
- Framework provides environment for building, deploying and running services of the web and other application.



BSS

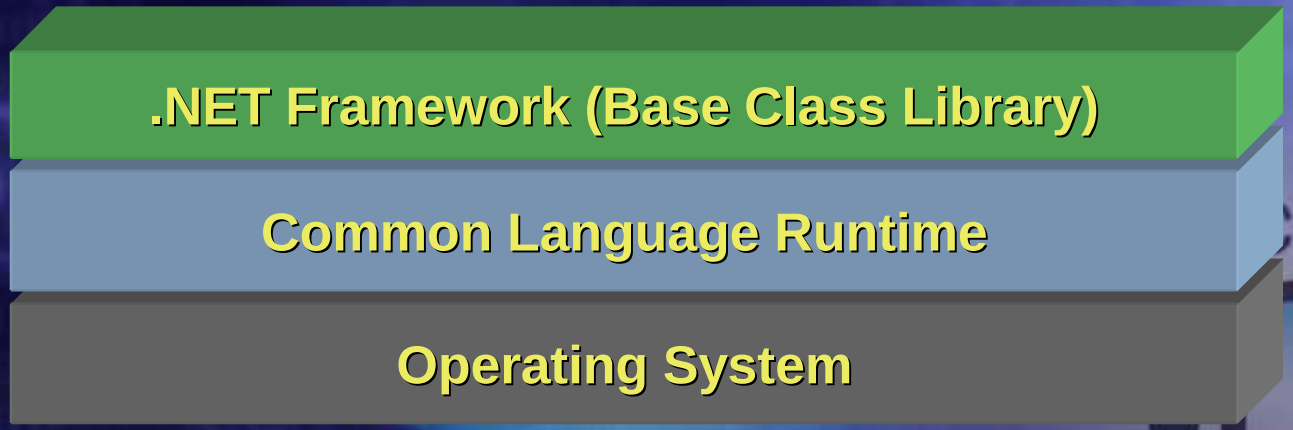
The .NET Framework Library



.NET Framework

Base Class Library

- .NET supplies a library of base classes that we can use to implement applications quickly.
- String handling, IO operations, graphics, text handling, file operations and many more … etc.

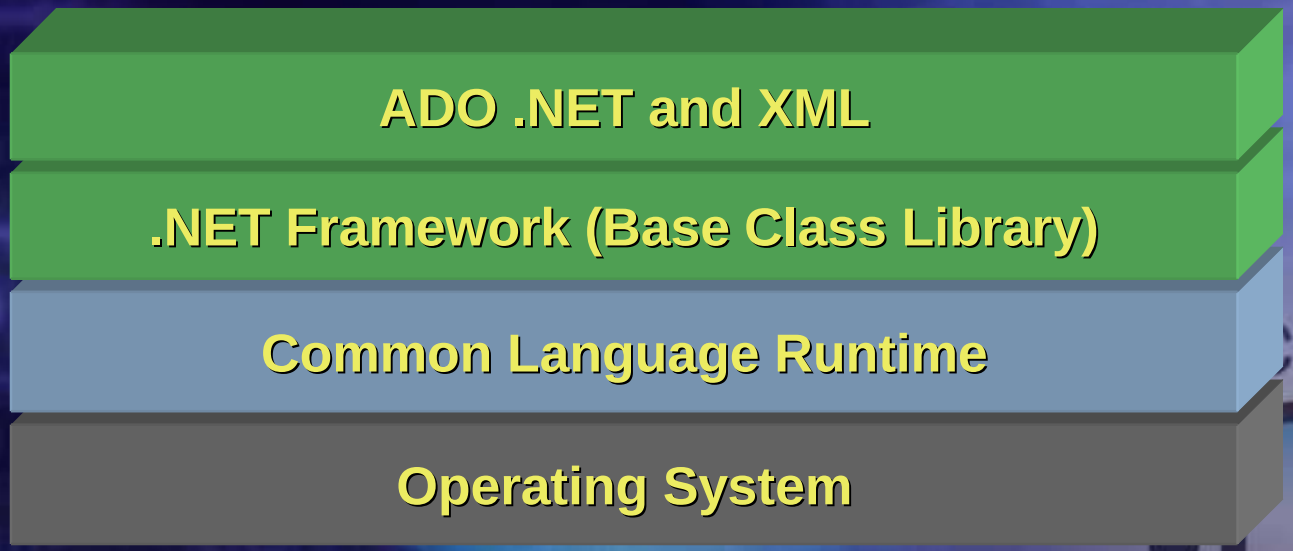


BSS

.NET Framework

Data Access Layer

- XML – Extended mark-up language, is a universally adopted language for internet message passing.
- ADO.NET provides class for handling database connection and maintenance with front-end .NET technology.

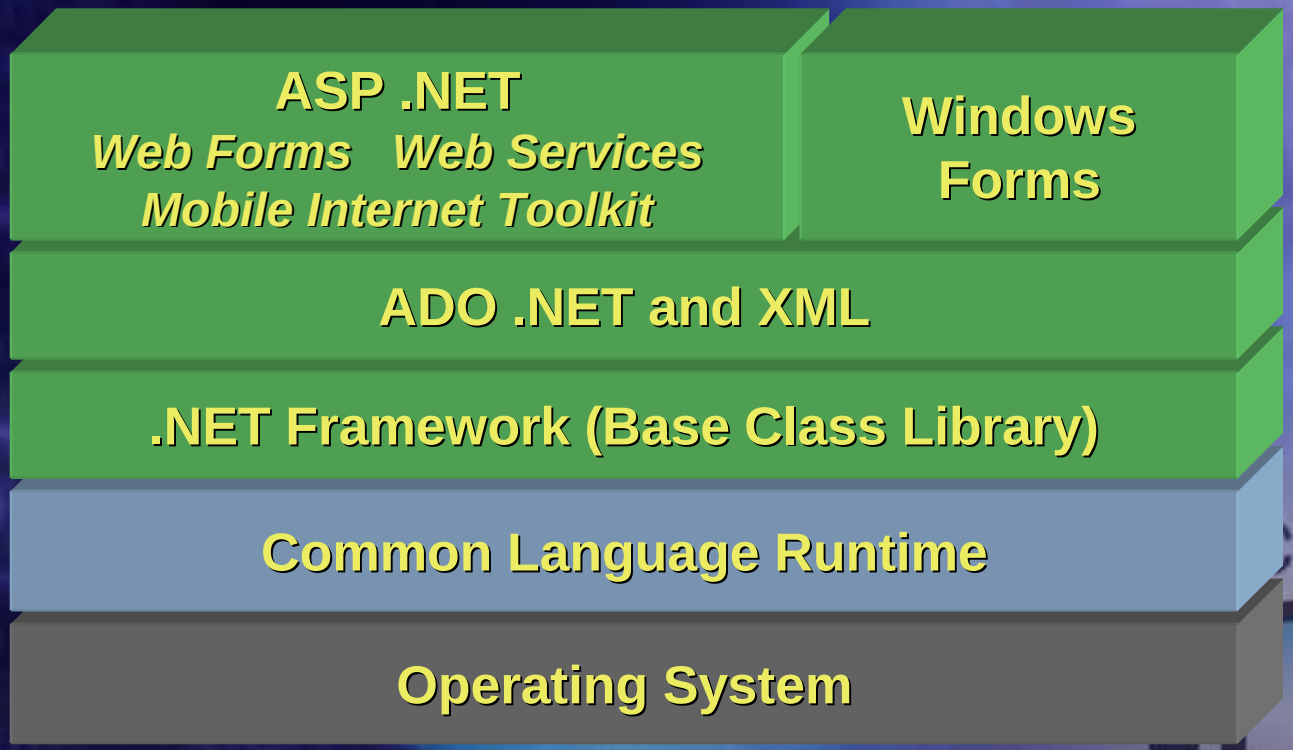


BSS

.NET Framework

ASP.NET & Windows Forms

- Create application's front-end – Web-based user interface, Windows GUI, Web services, …

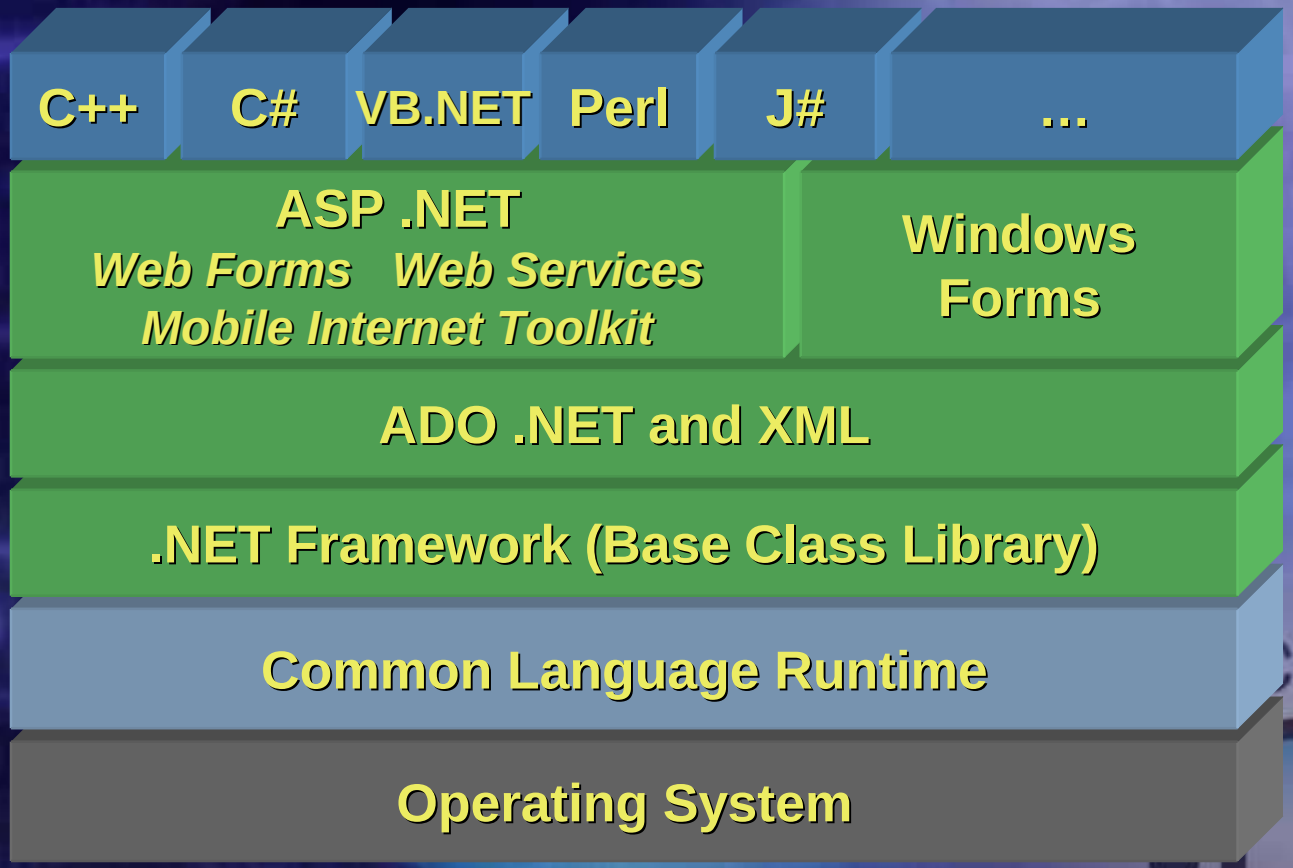


BSS

.NET Framework

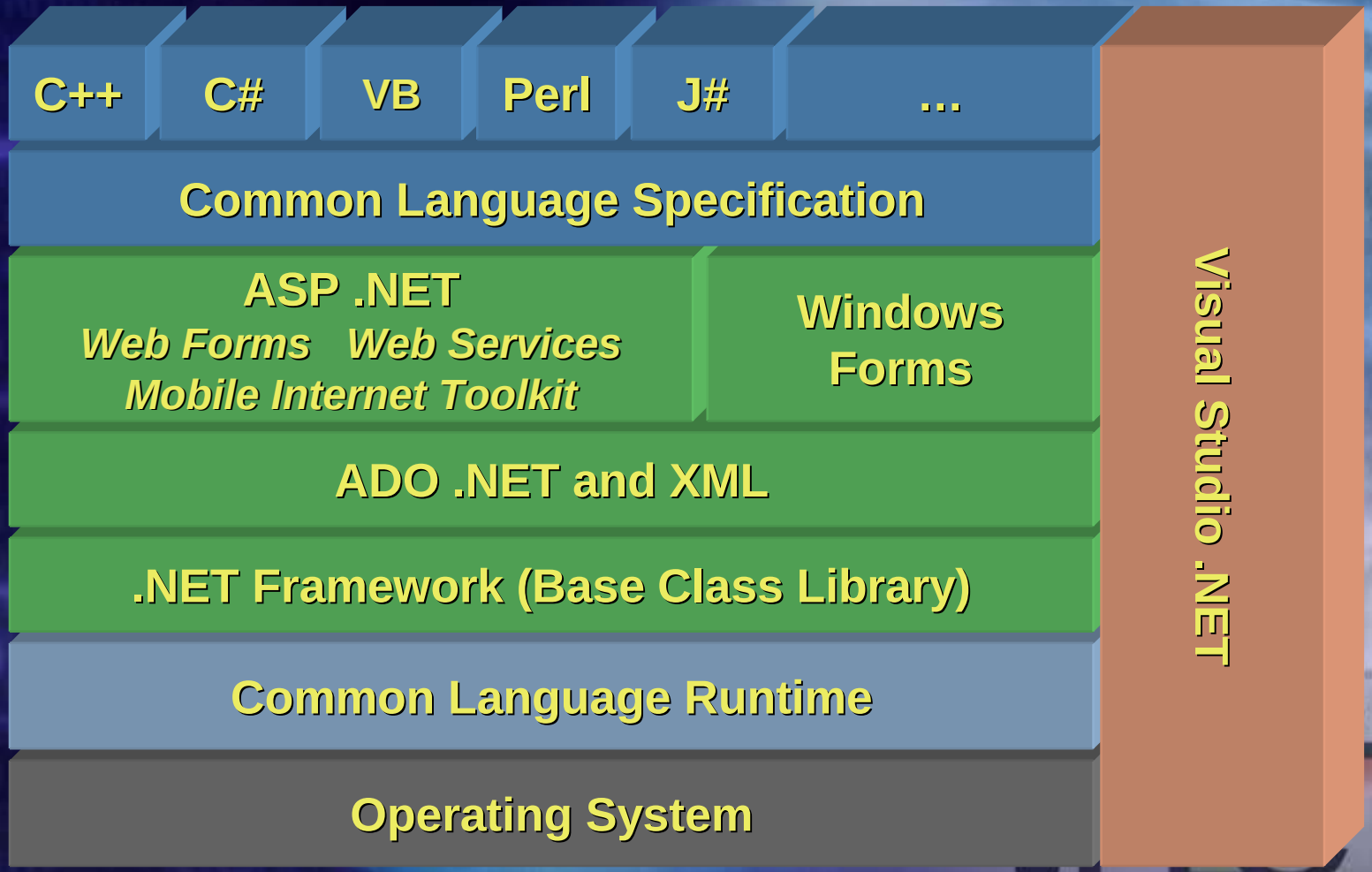
Programming Languages

- Use your favorite language



.NET Framework

Visual Studio .NET

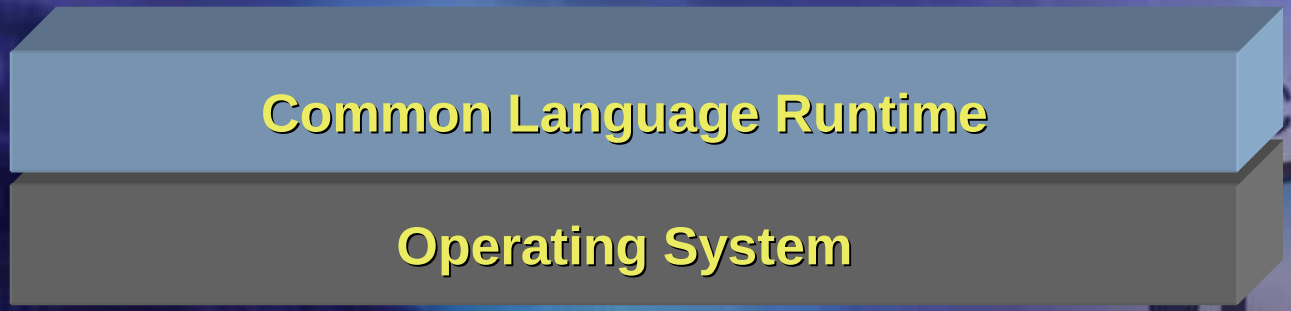


BSS

.NET Framework

Common Language Runtime

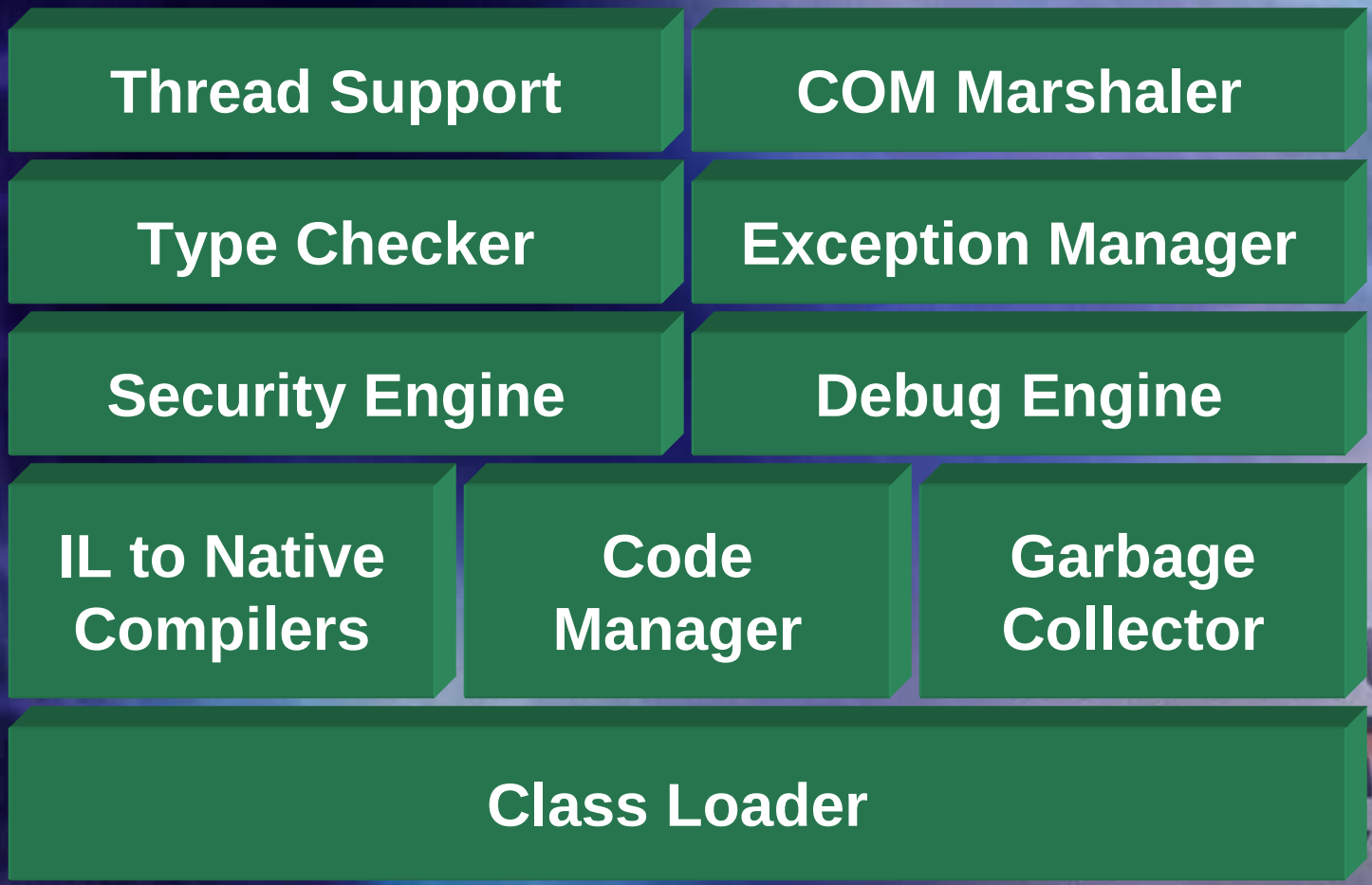
- CLR is the heart and soul of the .net framework.
- CLR is a runtime environment in which programs written in C# and other .NET language are executed.
- It also support s cross-language interoperability.



BSS

.NET Framework

Common Language Runtime



Common Language Runtime

- **Manages running code – like a virtual machine**
- COM marshaler, which allows .NET applications to exchange data with COM applications.
  - **Threading**
  - **Memory management**
  - **No interpreter: JIT-compiler produces native code – during the program installation or at run time**

  Type checker
  - **Code can be verified to guarantee type safety**
  - **No unsafe casts, no un-initialized variables and no out-of-bounds array indexing.**

Automatic Memory Management

- **The CLR manages memory for managed code**
  - **Unused objects and buffers are cleaned up automatically through *Garbage Collection.***
  - In computer science, **garbage collection** (**GC**) is a form of automatic memory management. It is a special case of *resource management*, in which the limited resource being managed is memory. The *garbage collector*, or just *collector*, attempts to reclaim *garbage*, or memory occupied by objects that are no longer in use by the program.
  - Opposite of manual memory management, which requires the programmer to specify which objects to deallocate and return to the memory system

Common Language Runtime

- **Code Access Security** (CAS), in the Microsoft .NET framework, is Microsoft's solution to prevent untrusted code from performing privileged actions .

- When the CLR loads an **assembly** it will obtain evidence for the assembly and use this to identify the code group that the assembly belongs to.

- A code group contains a permission set (one or more permission). Code that performs a privileged action will perform a code access **demand** which will cause the CLR to walk up the call stack and examine the permission set granted to the assembly of each **method** in the call stack.

Mscorcfg.msc – tool to configure code group policy

Common Language Runtime

- **Role Based Security**
- Enforcing security consists of two parts, Authentication and Authorization
- **Authentication : username & password/ anonymous access**
- **Authorization : is after authentication, checking for grant or deny permission on particular set or portion of program.**
- Find's the security information associated with that user **and based on this whether to grant or deny permission that is decided by application.**

Managed Code

- Code that targets the CLR is referred to as managed code
- All managed code has the features of the CLR
  - Object-oriented
  - Type-safe
  - Cross-language integration
  - Cross language exception handling
  - Multiple version support
- Managed code is represented in special Intermediate Language (IL)

Example of MSIL Code

```
.method private hidebysig static void  Main()
  cil managed
{
  .entrypoint
  // Code size       11 (0xb)
  .maxstack  8
  IL_0000:  ldstr      "Hello, world!"
  IL_0005:  call       void
  [mscorlib]System.Console::WriteLine(string)
  IL_000a:  ret
} // end of method HelloWorld::Main
```

IL Disassembler
Ildasm.exe

IL Assembler
Ilasm.exe Tool

Common Type System (CTS)

- **All .NET languages have the same primitive data types. An *int* in C# is the same as an *int* in VB.NET. This is how .NET provide multiple language support.**

- **When communicating between modules written in any .NET language, the types are guaranteed to be compatible on the Intermediate level.**
- **Hence calling one language from another does not require type conversions.**
- The CTS also defines the rules that ensures that the data types of objects written in various languages are able to interact with each other.

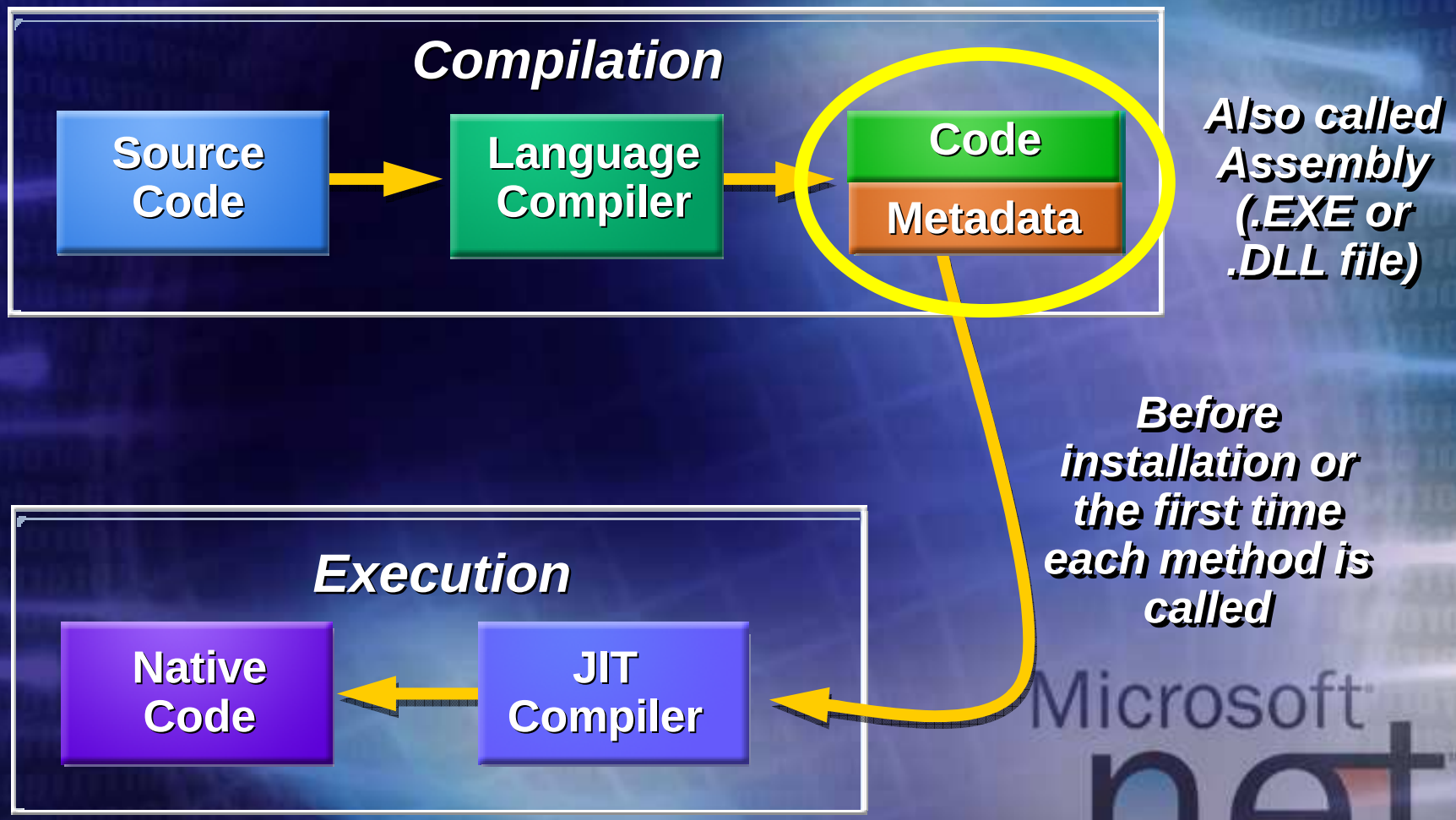
- **Strings are a primitive data type now.**

BSS

Common Language Specification (CLS)

- **Any language that conforms to the CLS is a .NET language**
- **Specification implies some set of rules.**
- **CLS is a subset of CTS, hence language supporting CLS can use each other's library as if they are their own.**
- **API designed with rules of CLS, can easily be used by all the .NET languages.**
- The specification defines an environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures.
- **Hence due to CLS, .NET supports cross-language interoperability.**

Code Compilation and Execution



Microsoft .net

BSS

CLR activities during app. Execution.

- **Your program source code is converted to MSIL or CIL with the use of language compiler, rather platform or processor-specific object code.**
- **MSIL is assembled into byte code.**
- **Bytecode** is a term which has been used to denote various forms of instruction sets designed for efficient execution by a compiler as well as being suitable for further compilation into machine code
- **This MSIL is platform independent insruction set executed by virtual machine (i.e CLR)**
- **CIL is then verified for safety during runtime, providing better security and reliability.**

CLR activities during app. Execution.

- **JIT involves turning bytecode into code immediately executable by CPU(Native code).**
- **JIT compiler uses metadata, which is data of data to verify any illegal access and violations appropriately.**
- **JIT compiles MSIL as and when needed, this saves time and space in memory.**
- **But results into performance hit.**
- NGEN compilation eliminates this step at run time. **It compiles entire MSIL generated.**

CLR activities during app. Execution.

- Step 1:- When you develop a program in a language that targets the CLR, at the time of compiling. Compiler, instead of compiling it to machine-level code, the compiler translates it into MSIL (Microsoft Intermediate Language). This ensures the language interoperability.

  Step 2:- In addition of translating the source code to IL, the compiler also produces the Metadata about the program. Metadata contains the description of the program, such as the classes and interfaces, the dependencies, and the versions of the components used in the program.

  Step 3:- The MSIL and the metadata are accommodated into the Assembly.

  Step 4:- The compiler crates the .exe or .dll file. This file is also called portable executable (PE).

CLR activities during app. Execution.

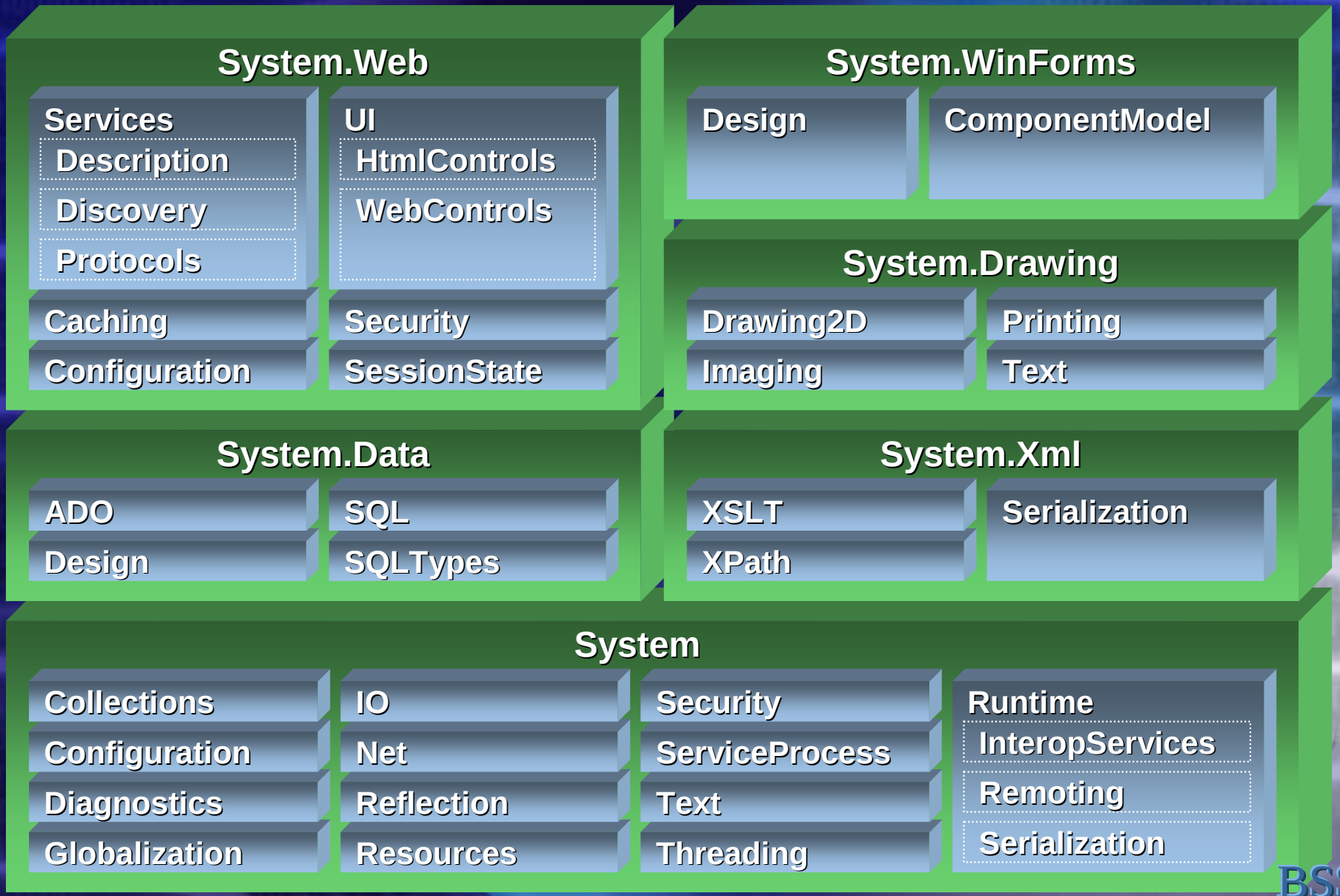
- **Step 5:-** When you executed the .EXE or .DLL file, the code and all the relevant information is loaded into the Class loader. Class loader is the integrated component of CLR and it loads the code into system memory.

  Step 6:- Before the code can be executed, the .NET framework needs to convert the IL into the native or CPU-specific code. The JIT compiler translates the code from IL into managed native code.

  Step 7:- During JIT compilation, the code is also checked for type safety. Type Safety ensures that objects are always accessed in a compatible way.

  Step 8:- After translation the IL into native code, the translated code is sent to the .NET runtime manager. The .NET runtime manager executes the code. While executing the code, a security check is performed to ensure that the code has the appropriate permissions for accessing the available resources.

.NET Framework Namespaces



BSS

Namespaces

- Namespaces are the basic building block for the .NET framework.
- Namespaces are a way to define the classes and other types of information into one hierarchical structure
- **System** is the basic namespace used by every .NET code. If we can explore the System namespace little bit, we can see it has lot of namespace user the system namespace.
- A namespace can be created via the Namespace keyword

Namespaces

- Sample namespace creation code
- **C# Code:**
- namespace Books
{
    namespace Inventory
    {
        using System;
        class AddInventory
        {
            public void MyMethod()
            {
Console.WriteLine("Adding Inventory Method!");
            }
        }
    }
}

BSS

Namespaces

- How to use that namespace?
- **C# Code:**
- using Books;
  class HelloWorld
  {
  public static void Main()
  {
  Inventory.AddInventory AddInv
  = new AddInventory();
  AddInv.MyMethod();
  }
  }

Microsoft .net

BSS

Namespaces

- Alternatively :
- **OR**
- using Books.Inventory;
  class HelloWorld
  {
  public static void Main()
  {
  AddInventory AddInv = new AddInventory();
  AddInv.MyMethod();
  }
  }
- **Note:** When using Imports statement or Using statement we can use only the namespace names, we can't use the class names.

Base Class Library Namespaces

Data And XML Namespaces

System.Data

- OleDb
- SQLClient
- Common
- SQLTypes

System.Xml

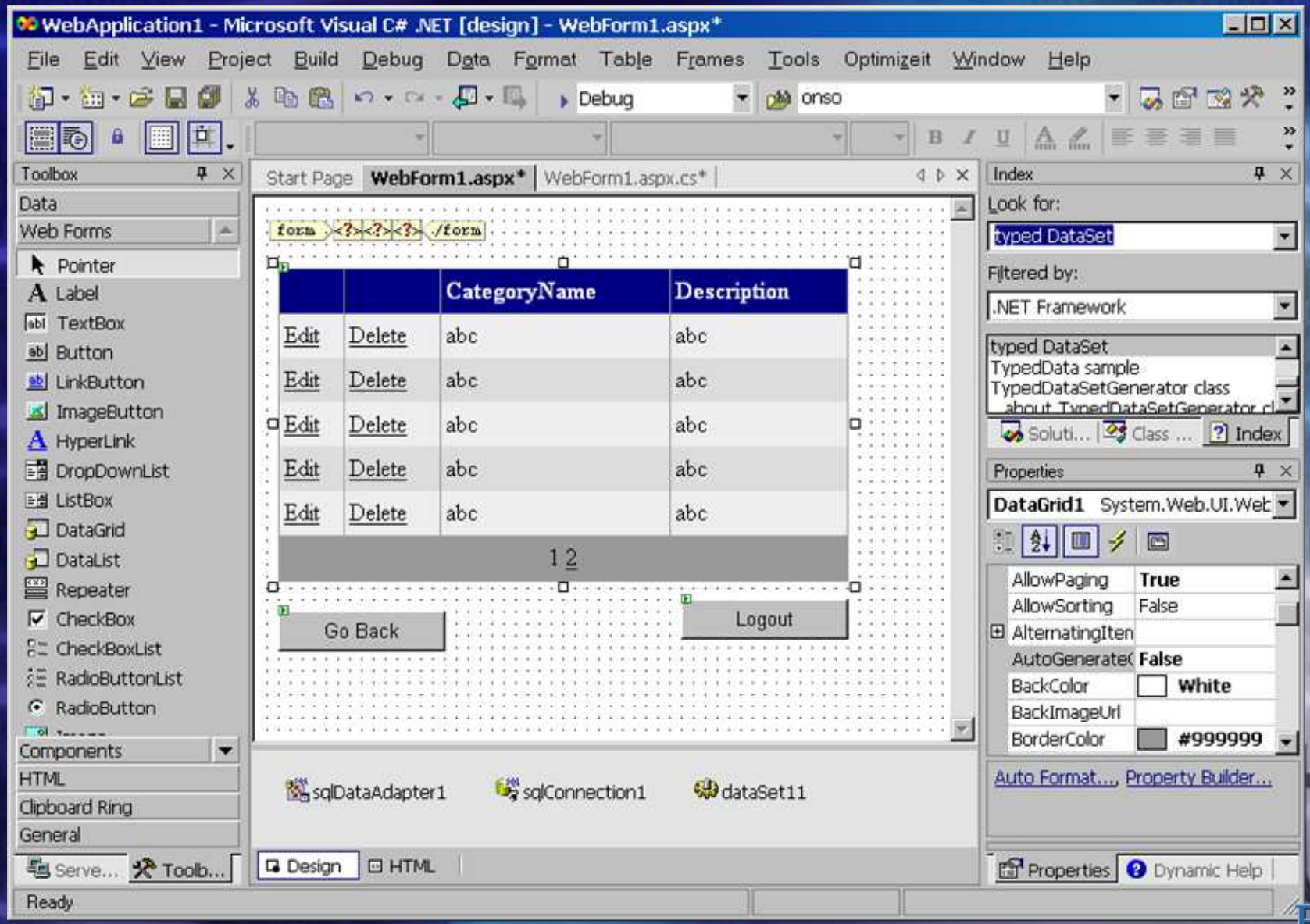
- XSLT
- Serialization
- XPath

.NET Languages Available (or soon to be)

- Visual Basic
- C#
- Jscript
- C++
- **Perl**
- **Python**
- **COBOL**
- **Haskell**
- **ML**
- **Ada**
- **Pascal**
- **C**
- **SmallTalk**
- **Oberon**
- **Scheme**
- **Mercury**
- **APL**
- **Eiffel**
- **Oz**
- **Objective Caml**

BSS

VS.NET – Web Forms Designer



.NET Framework on Linux

- Mono Project
  - Open Source C# compiler, CLR and Framework Class Library
  - Runs on various platforms and hardware:
    - Linux, Unix, FreeBSD, Windows – JIT-compiler for x86
    - s390, SPARC, PowerPC – interpreter for these hardware architectures
  - Supports also:
    - ADO.NET and XML
    - Windows Forms (not fully)
    - ASP.NET
    - Web Services

BSS

.NET Framework on Linux (2)

- Mono Project
  - Runs .NET portable executables on Linux, e.g.

    ```
    mono myapp.exe
    ```

  - Compiles .NET applications to portable executables, e.g.

    ```
    mcs myapp.cs
    ```

  - The obtained .exe file can taken and run on Windows
- DotGNU Portable.NET
  - Build and execute .NET applications on GNU/Linux, Windows, Solaris, NetBSD, FreeBSD, and MacOS X

.NET Framework – Resources

- Visit following web sites:
  - .NET Framework Home Site – http://msdn.microsoft.com/netframework/
  - The Microsoft .NET Framework Community – http://www.gotdotnet.com/
  - ASP.NET – http://www.asp.net/
  - .NET Windows Forms – http://www.windowsforms.net/
  - Code Project – http://www.codeproject.net/
  - Mono – Open Source .NET Framework – http://www.go-mono.org/
  - Rotor – Shared Source .NET CLI – http://msdn.microsoft.com/net/sscli/
- Read the news groups:
  - news://msnews.microsoft.com/microsoft.public.dotnet.framework