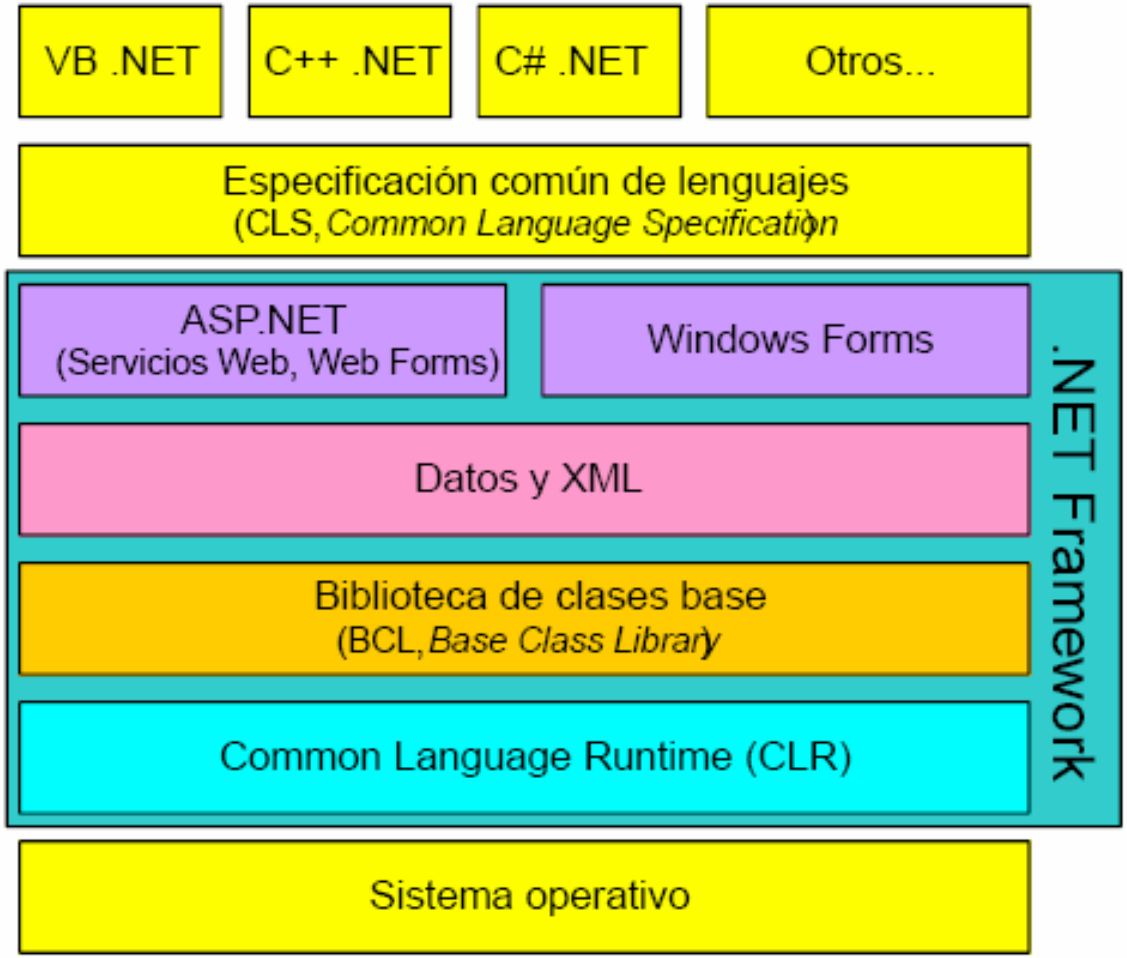
# Tema 1: Conceptos básicos de la arquitectura .NET

- El entorno .NET Framework
- Arquitectura de .NET Framework
- Common Language Runtime (CLR)
- Especificación de lenguajes común (CLS)
- Lenguaje intermedio de Microsoft (MSIL)
- Estructura de una aplicación .NET
- Ensamblados, manifiestos y metadatos
- Modelo de ejecución
- El compilador Just-in-Time
- La biblioteca de clases
- Espacios de nombre

# El entorno .NET Framework

- Entorno de ejecución de aplicaciones informáticas sobre el que se ejecuta cualquier programa desarrollado en .NET en cualquiera de sus lenguajes (VB.NET, Visual C++ .NET, Visual C# .NET, Visual J#, NetCOBOL, etc.).
  - Forma parte de Microsoft .NET que incluye además herramientas de programación (Visual Studio .NET), servidores (Windows 2008 Server o SQL Server), software cliente (Windows XP, Windows CE, Office XP), etc.)
- Ofrecen un entorno de ejecución común:
  - Instalación transparente.
  - Fin de las incompatibilidades de DLL y otros componentes.
  - Mismas capacidades para todos los lenguajes.

# Arquitectura de .NET Framework



# Arquitectura de .NET Framework (II)

❑ Unas definiciones del .NET Framework podrían ser, la siguientes:

- es un entorno multi-lenguaje para la construcción, distribución y ejecución de Servicios Webs y aplicaciones.
- es una nueva plataforma diseñada para simplificar el desarrollo de aplicaciones en el entorno distribuido de Internet .
- consta de dos componentes principales:
  - el Common Language Runtime.
  - la librería de clases .NET Framework.

- **Resumiendo:** *es el corazón de .NET, cualquier cosa que queramos hacer en cualquier lenguaje .NET debe pasar por el filtro cualquiera de las partes integrantes del .NET Framework.*

# Arquitectura de .NET Framework (III)

❑ Entorno común de ejecución (*Common Language Runtime*, CLR).

- Administra el código en tiempo de ejecución y proporciona los servicios básicos (administración de memoria, control de excepciones, control de hilos de ejecución).
- El Common Lenguage Runtime (CLR) es una serie de librerías dinámicas (DLLs), también llamadas assemblies, que hacen las veces de las DLLs del API de Windows así como las librerías runtime de Visual Basic o C++
- **Resumiendo:** *al final consiste en una serie de librerías usadas en tiempo de ejecución para que nuestros ejecutables o cualquiera basado en .NET puedan funcionar.*

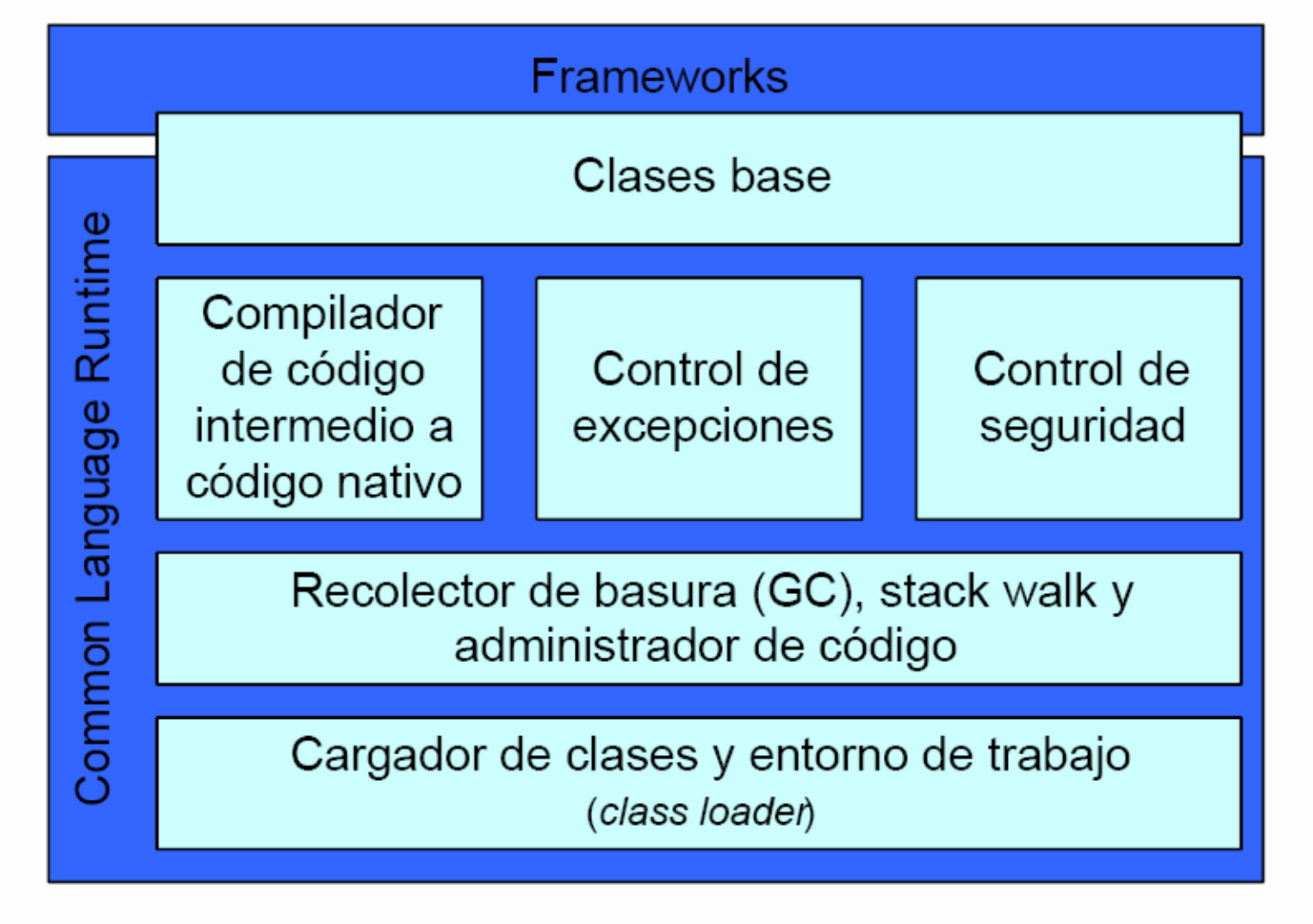
# Arquitectura de .NET Framework (IV)

- Biblioteca de clases base (Base Class Library, BCL).
  - Colección de código OO que puede ser empleado desde cualquier lenguaje .NET.
  - Contiene los tipos básicos, clases para la entrada/salida, seguridad, etc.
  - Al tener definidos los tipos de datos para todos los lenguajes, facilita el intercambio de datos entre aplicaciones desarrolladas en distintos lenguajes.
  - **Resumiendo:** *la librería de clases de .NET Framework proporcionan una jerarquía de clases orientadas a objeto disponibles para cualquiera de los lenguajes basados en .NET, incluido el Visual Basic.*

# Arquitectura de .NET Framework (V)

- Capa de datos y XML.
  - Gestiona el acceso a los datos y el tratamiento de datos XML.
    - ✓ Los datos los gestiona mediante ADO.NET.
    - ✓ Gran parte de la información de .NET (configuración, estructura de archivos y de aplicaciones) se gestiona mediante XML.
      - Facilidad para importar, exportar y tratar datos de/hacia XML.
- Definición de la interfaz: ASP.NET y Windows Forms.
  - ASP.NET utiliza Web Forms (para aplicaciones Web basadas en ASP) y los servicios Web.
  - Windows Forms proporciona un conjunto de componentes de interfaz para desarrollar aplicaciones cliente basadas en Windows.

# Common Language Runtime



# Common Language Runtime (II)

- Cargador de clases (*Class Loader*).
  - Llama al puntero de inicio del procedimiento, establece su entorno de memoria y mantiene la ejecución bajo control.
- Control de recursos: recolector de basura, control de la pila, administrador de códigos.
- Interacción con el sistema operativo: control de excepciones, control de seguridad, compilador de código nativo.
  - El código intermedio debe compilarse a código nativo.
- Clases base.
  - Definen el entorno de trabajo sobre el que se apoya el código.

# Especificación de lenguajes común (CLS)

- Conjunto de directivas proporcionadas por Microsoft para el desarrollo de lenguajes compatibles con .NET.
  - Fijan las características comunes de los lenguajes.
  - Los fabricantes pueden ampliarlas o modificarlas.
    - ✓ Una aplicación no debería hacer uso de las características ampliadas cuando se comunique con otros componentes .NET.
- Actualmente están desarrollados o en fase de desarrollo más de 60 lenguajes .NET.
- Código administrado y código no administrado.
  - Todos los lenguajes .NET producen código administrado.
    - ✓ Código que ejecuta el CLR directamente.
  - .NET puede ejecutar también código generado por lenguajes que no sigan el CLS.
    - ✓ Código no administrado ( código nativo) que permite, por ejemplo, utilizar API antiguas y controles ActiveX.

# Especificación de lenguajes común (CLS) (II)

❑ Los lenguajes .NET son más parecidos entre si que otros lenguajes clásicos.

- Por lo tanto:
  - ✓ Su velocidad de ejecución es bastante similar. La elección de un lenguaje u otro no dependerá de la velocidad de ejecución.
  - ✓ Garantiza la interoperabilidad entre lenguajes.
    - Todos utilizan los mismos tipos basados en el Sistema de tipo común (*Common Type System*, CTS).
    - Posibilidad de crear una aplicación con componentes desarrollados en distintos lenguajes.
  - ✓ La curva de aprendizaje para un nuevo lenguaje .NET es menor.

# Especificación de lenguajes común (CLS) (III)

```
'HolaMundo en VB.NET
Public Class HolaMundo
        Shared Sub main()
           System.Console.WriteLine("¡Hola, mundo!")
           System.Console.ReadLine()
        End Sub
End Class
```

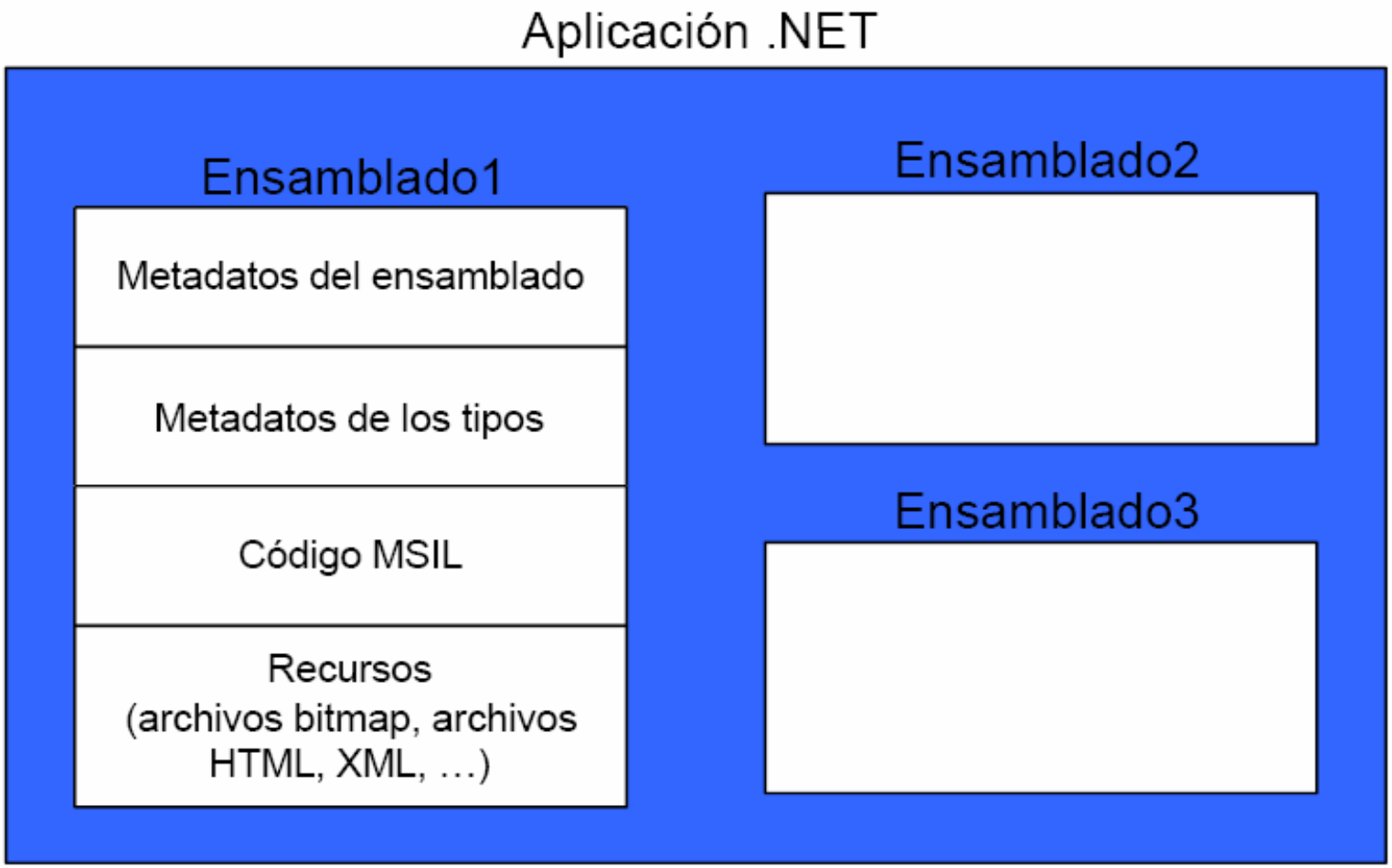
```
//HolaMundo en C#
public class HolaMundo{
static void Main(){
        System.Console.WriteLine("¡Hola, mundo!");
        System.Console.ReadLine();
        }
}
```

```
//HolaMundo en VJ#
public class HolaMundo{
        public static void main(String args[]){
        System.Console.WriteLine("¡Hola, mundo!");
        System.Console.ReadLine();
        }
}
```

# Lenguaje intermedio de Microsoft (MSIL)

- Los compiladores .NET no producen código nativo (código binario ejecutable directamente por el procesador).
  - Producen código intermedio (MSIL o simplemente IL).
- Se trata de una especie de lenguaje máquina asociado a un procesador virtual que no corresponde a ninguna CPU disponible en la actualidad.
- El compilador a código nativo (compliador JIT, Just-In-Time) de la CLR es el que se encargará de compilar “al vuelo” el código IL a lenguaje nativo de la plataforma donde se ejecute la aplicación.
- Esta característica permitirá que el código de una aplicación .NET pueda ejecutarse en otra plataforma distinta sin ningún cambio (siempre y cuando exista un CLR para esa plataforma).

# Estructura de una aplicación .NET



# Ensamblados, manifiestos y metadatos

- Ensamblado: unidad de instalación de una aplicación .NET.
- Compuesto de uno o más archivos de recursos y de archivos ejecutables MSIL (módulos administrados).
  - Los módulos administrados contienen código IL y metadatos que describen los tipos del módulo.
- Un ensamblado tendrá un *archivo ejecutable portable* (PE) que se generará al compilar la aplicación a MSIL.
  - Archivo .exe o .dll.
  - Un archivo PE no podrá ejecutarse si no está contenido en un manifiesto de ensamblado.

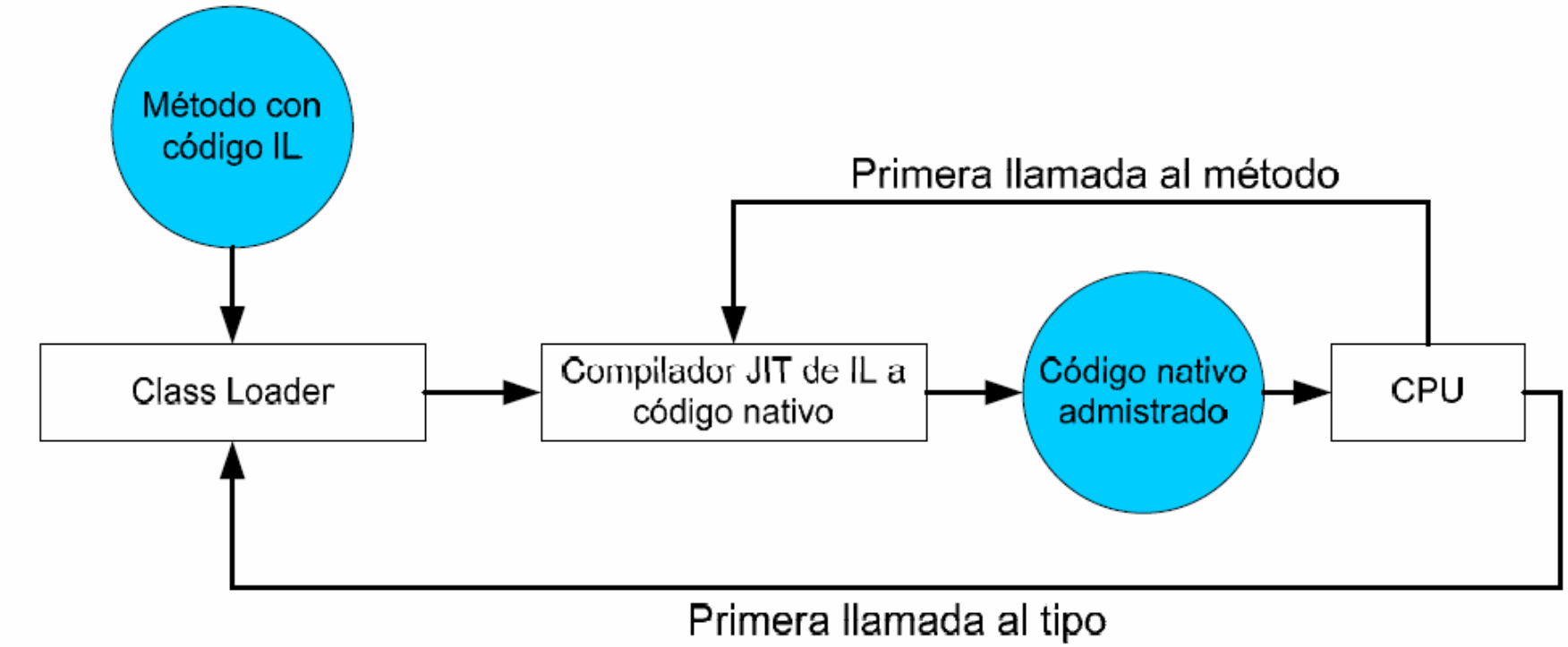
# Ensamblados, manifiestos y metadatos (II)

- Los ensamblados:
  - Crean un límite de seguridad.
    - ✓ Es la unidad donde se solicitan y conceden los permisos.
  - Crean un límite de tipos.
    - ✓ La referencia a un tipo incluye el nombre del ensamblado en que reside.
  - Crean un límite de ámbito de referencia.
    - ✓ El manifiesto del ensamblado contiene metadatos para resolver los tipos: especifica que tipos se exponen fuera del ensamblado y enumera los ensamblados de los que depende.
  - Forma un límite de versión.
    - ✓ Todos los recursos de un ensamblado pertenecen a la misma versión.
  - Crean la unidad de implementación.
    - ✓ Al arrancar una aplicación sólo deben estar presentes los ensamblados a los que llama la aplicación inicialmente. El resto se recuperan a petición

# Ensamblados, manifiestos y metadatos (III)

- Solución al problema de versiones.
  - Las aplicaciones Win32 requieren que los componentes que utiliza estén especificados como entradas del registro.
    - ✓ Dos aplicaciones distintas pueden usar el mismo componente, pero con versiones distintas.
  - Los ensamblados son autodescriptivos: no dependen de las entradas del registro.
    - ✓ La instalación de aplicaciones se simplifica: sólo es necesario copiar los directorios donde están contenidos los recursos a los que llama el ensamblado.
    - ✓ Con el ensamblado se instala siempre la versión adecuada de los componentes.

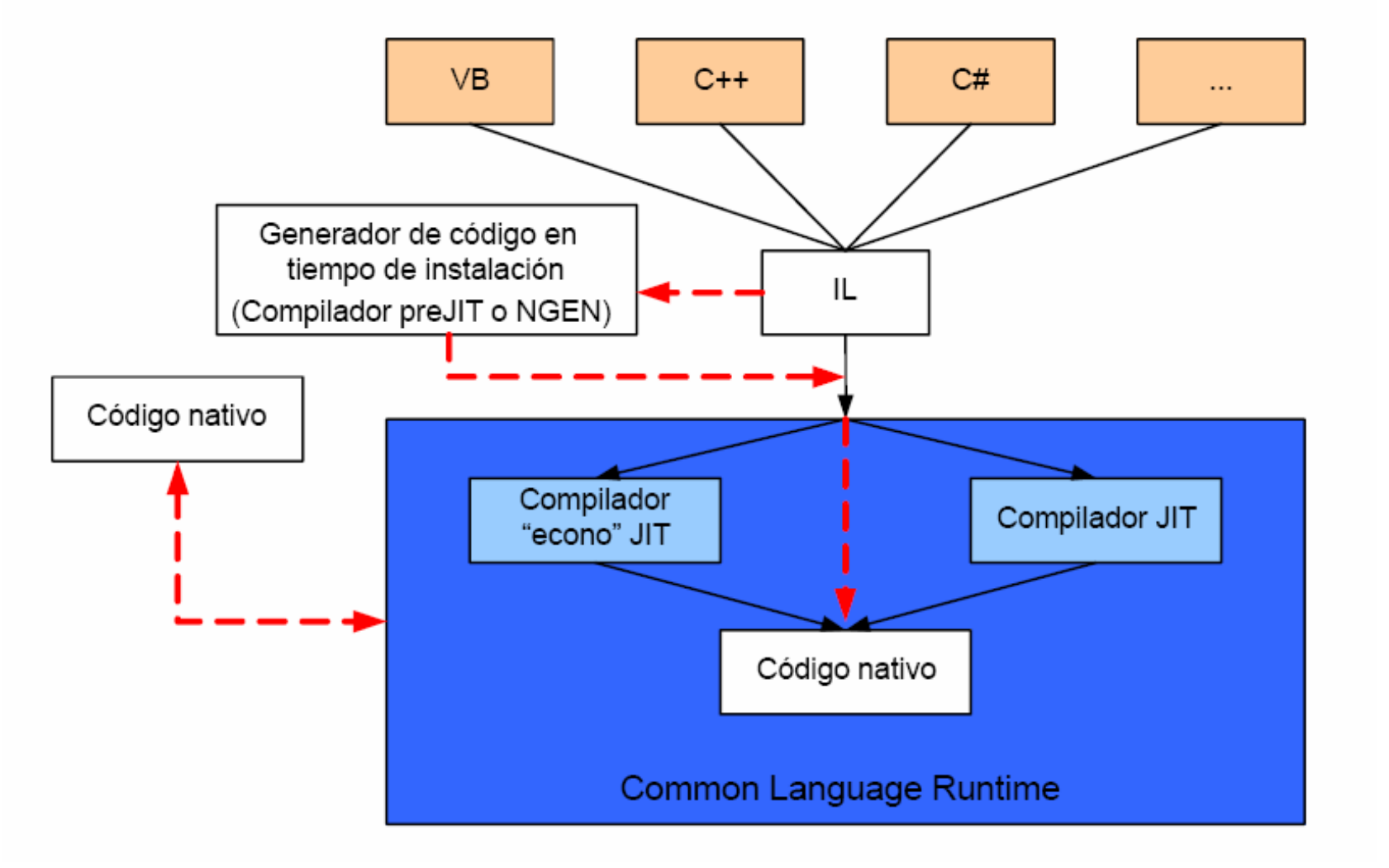
# Modelo de ejecución



# Modelo de ejecución (II)

- .NET Framework no es un interprete de *bytecode* como la máquina virtual Java.
  - No traduce las instrucciones una a una: compila todo el método a la vez.
- No carga toda la aplicación a la vez.
  - El tiempo de carga inicial es despreciable.
- No necesita utilizar técnicas de optimización de bajo nivel.

# El compilador Just-in-Time



# El compilador Just-in-Time (II)

- Compilador JIT estándar.
  - Compilación completa, optimizando el código y verificando su integridad.
- EconoJIT.
  - Diseñado para aplicaciones en el que la velocidad de descarga es importante (aplicaciones ASP.NET) o cuando se ejecuta en dispositivos con recursos limitados (Windows CE).
  - No realiza una optimización del código.
  - Permite el *rechazo de código*.
- Generador de imágenes nativas (NGEN).
  - Genera código en tiempo de instalación.
  - Mejora de rendimiento en casos especiales.
- Utilización de código nativo de forma directa.

# La biblioteca de clases

- Biblioteca de clases base (BCL).
  - Conjunto de clases que permiten a todos los lenguajes .NET realizar tareas comunes (entrada/salida, acceso a BBDD, interfaz de usuario, etc.).
- Su forma de acceso es común a todos los lenguajes.
  - Mantiene la interoperabilidad entre lenguajes que se ajusten al CLS.
- Permiten su modificación mediante herencia.

# Espacios de nombre

- Los espacios de nombre o Namespaces organizan la infinidad de clases de la BCL.
- Para utilizar los tipos de la BCL de un espacio de nombres concreto se utiliza la notación de punto.
  - Permite agrupar espacios de nombres y clases relacionadas bajo una raíz común.
    - ✓ Por ejemplo, System.Data guarda las clases relacionadas con el acceso a datos, System.Windows.Forms guarda las clases de interfaz para las aplicaciones basadas en formularios Windows, etc.
- Cada espacio de nombres se puede guardar en una o varias DLL.
  - La ubicación física no tiene porqué guardar correspondencia con la organización lógica de los espacios de nombres.
- El concepto de espacios de nombres permite ampliar la biblioteca con clases propias de una forma coherente.

# Espacios de nombre (II)

❑ Para crear nuevos espacios de nombres en VB.NET se utiliza la declaración Namespace…End Namespace.

```
Namespace MiCompañia
        Namespace MiNuevaTecnologia
                Class MiClase
                        …
                End Class
        End Namespace
End Namespace
```

❑ Para utilizar un espacio de nombres ya creado y no usar el nombre completamente cualificado, en VB.NET se utiliza la declaracion Imports.

- **Imports no importa nada, simplemente añade el espacio de nombres cuando no se utiliza un nombre completamente cualificado.**

```
Imports MiCompañia.MiNuevaTecnología
...
Dim x as New MiClase()
'utilizar MiCompañia MiNuevaTecnología Es lo mismo que New MiCompañia.MiNuevaTecnología.MiClase()
```

# Espacios de nombre (III)

```
'En el ensamblado MiCompañía
Namespace MiNuevaTecnología
        Public Class Class1
                …
        End Class

        Public Class Class2
                …
        End Class
End Namespace

Namespace MiOtraNuevaTecnología
        Namespace AlgunasClases
                Public Class Class3
                …
                End Class

                Public Class Class4
                                                ...
                End Class

        End Namespace
End Namespace
```

```
Imports MiCompañía.MiNuevaTecnología
Imports MiCompañía.MiOtraNuevaTecnología

Module Module1
        Sub Main()
                Dim a As Class1
                Dim b As Class2
                Dim c As AlgunasClases.Class3
                Dim d As AlgunasClases.Class4
        End Sub
End Module
```

El ensamblado MiCompañía (una biblioteca de clases) incluye 3 espacios de nombre.

La aplicación importa 2 espacios de nombre, por lo que no tiene que utilizar el nombre cualificado para las clases Class3 y Class4.