



Introduction to .NET Framework



.NET – What Is It?

- Software platform
- Language neutral
- In other words:

.NET is not a language (Runtime and a library for writing and executing written programs in any compliant language)

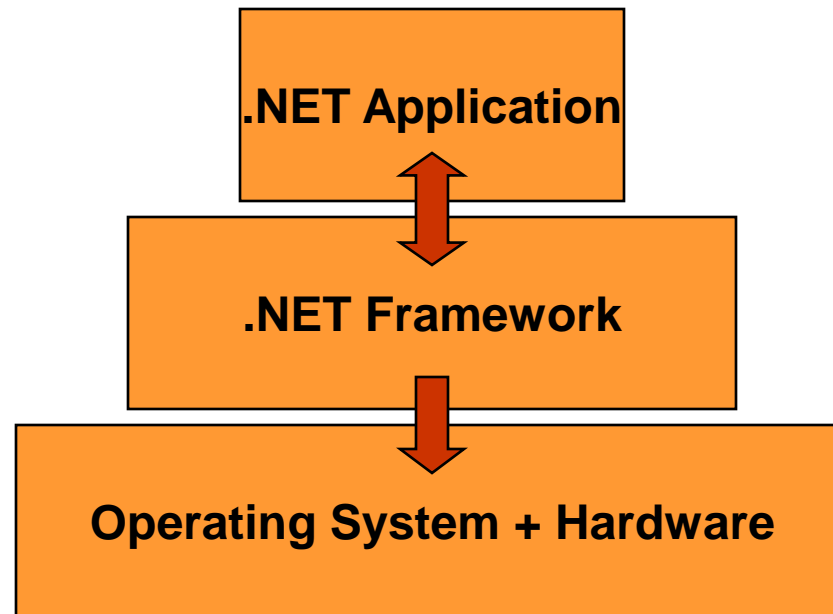


What Is .NET

- .Net is a new framework for developing web-based and windows-based applications within the Microsoft environment.
- The framework offers a fundamental shift in Microsoft strategy: it moves application development from client-centric to server-centric.



.NET – What Is It?





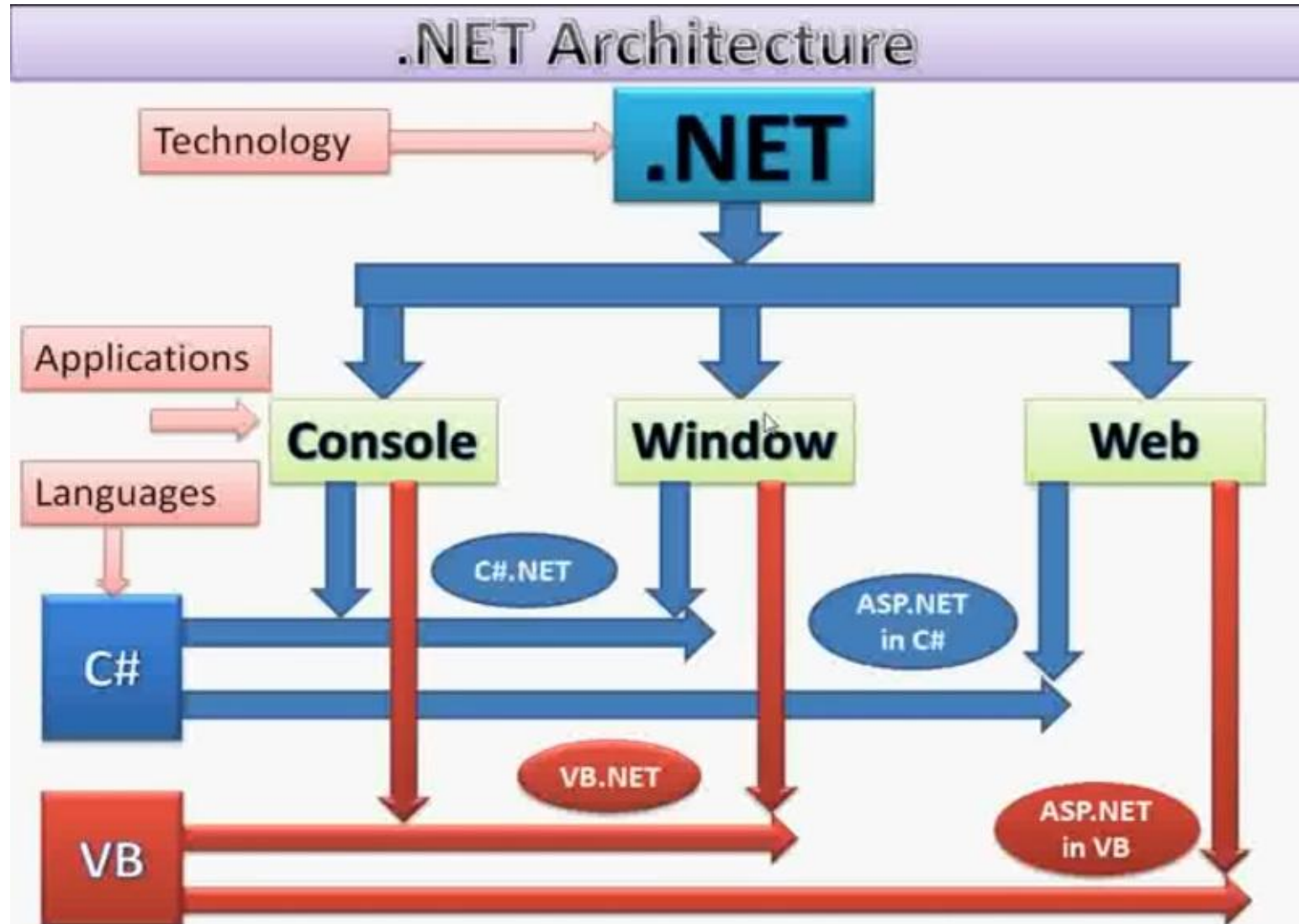
.NET Framework

- Common Intermediate Language (CIL)
- Common Language Runtime (CLR)
- Just-In-Time (JIT) Compiler
- Common Language Specification
- Framework Class Library (FCL)



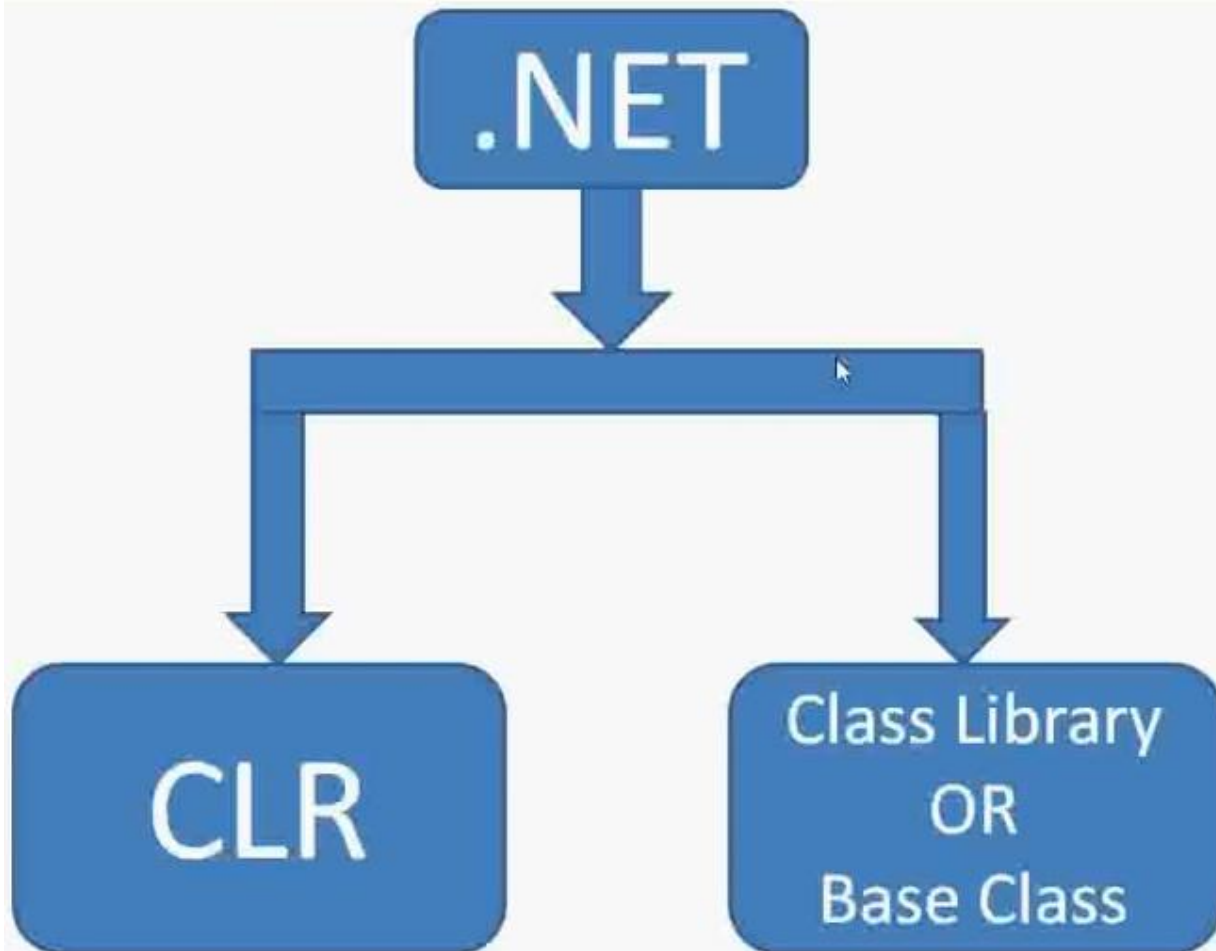
.NET Syllabus

1	.NET Architecture	+	C# Architecture	6	LINQ [Language Integrated Query]
2	OOPS	+	Project	7	WPF [Windows Presentation Foundation]
3	Window + ADO.NET	+	2 Tier 3 Tier Architecture	8	WCF [Windows Communication Flow]
4	ASP.NET AJAX	+	Project	9	SilverLight
5	WebService	+	Class Library	10	MVC[Model View Controller]



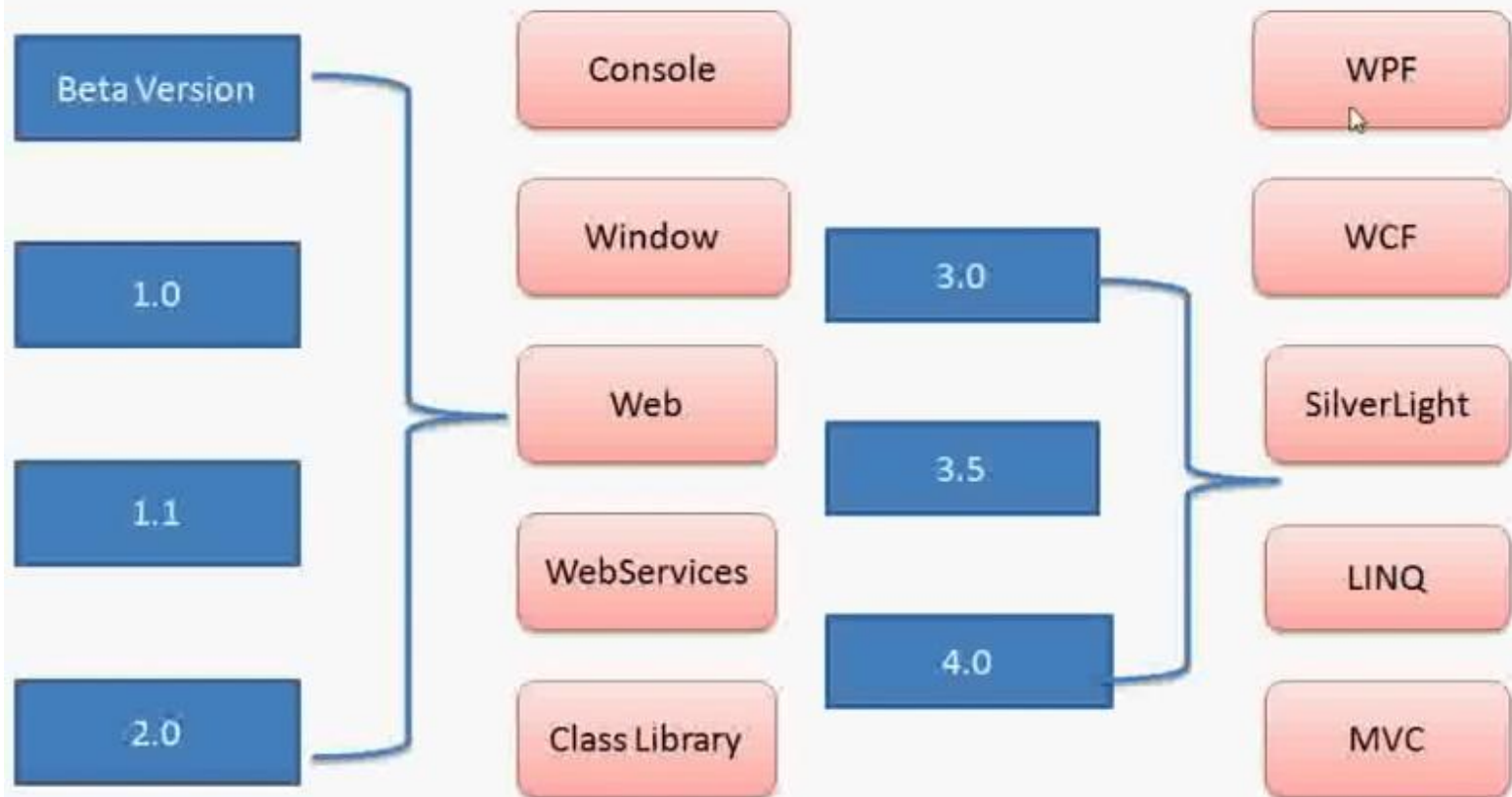


Definition Of .NET



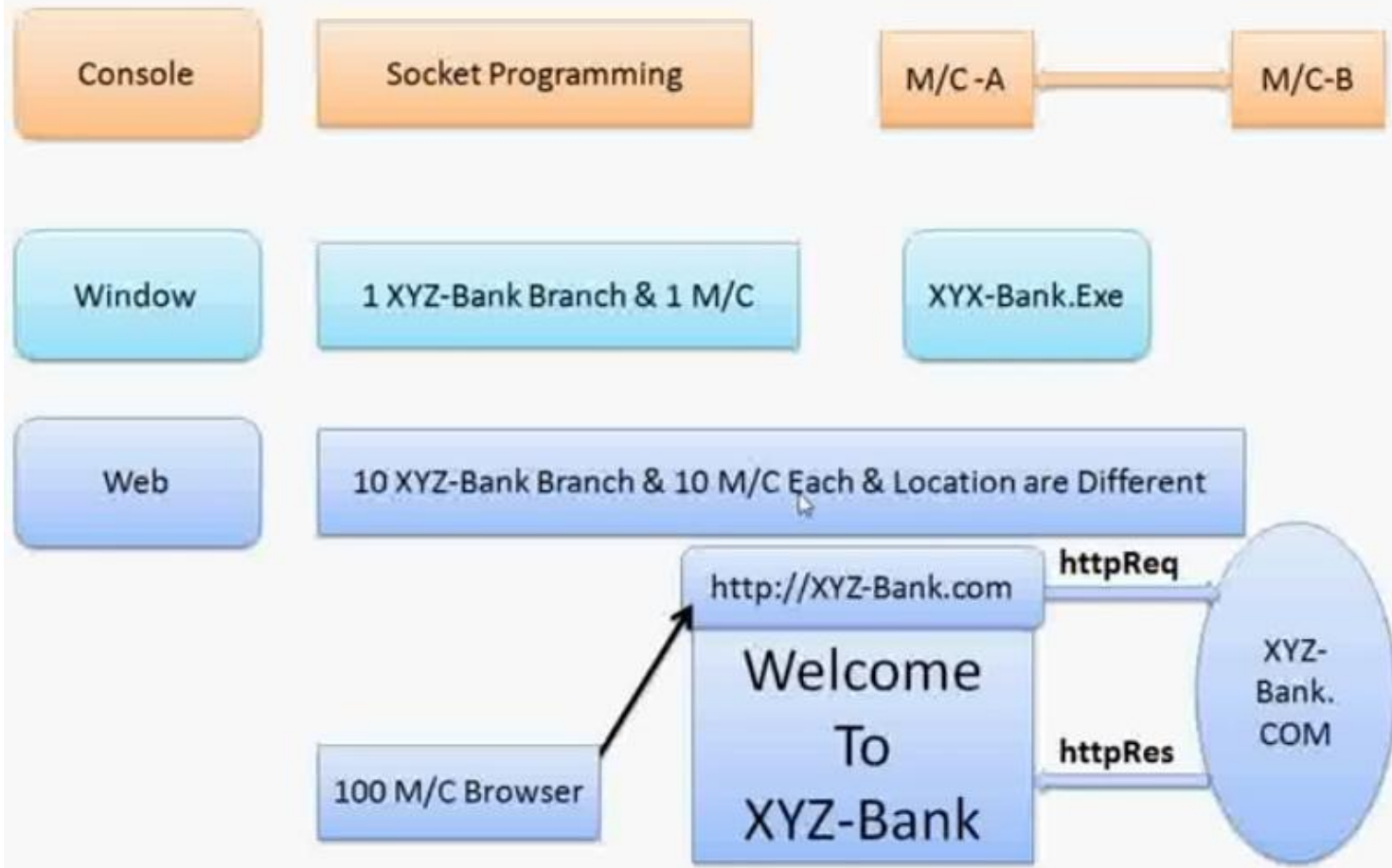


NET Frameworks OR .NET Version



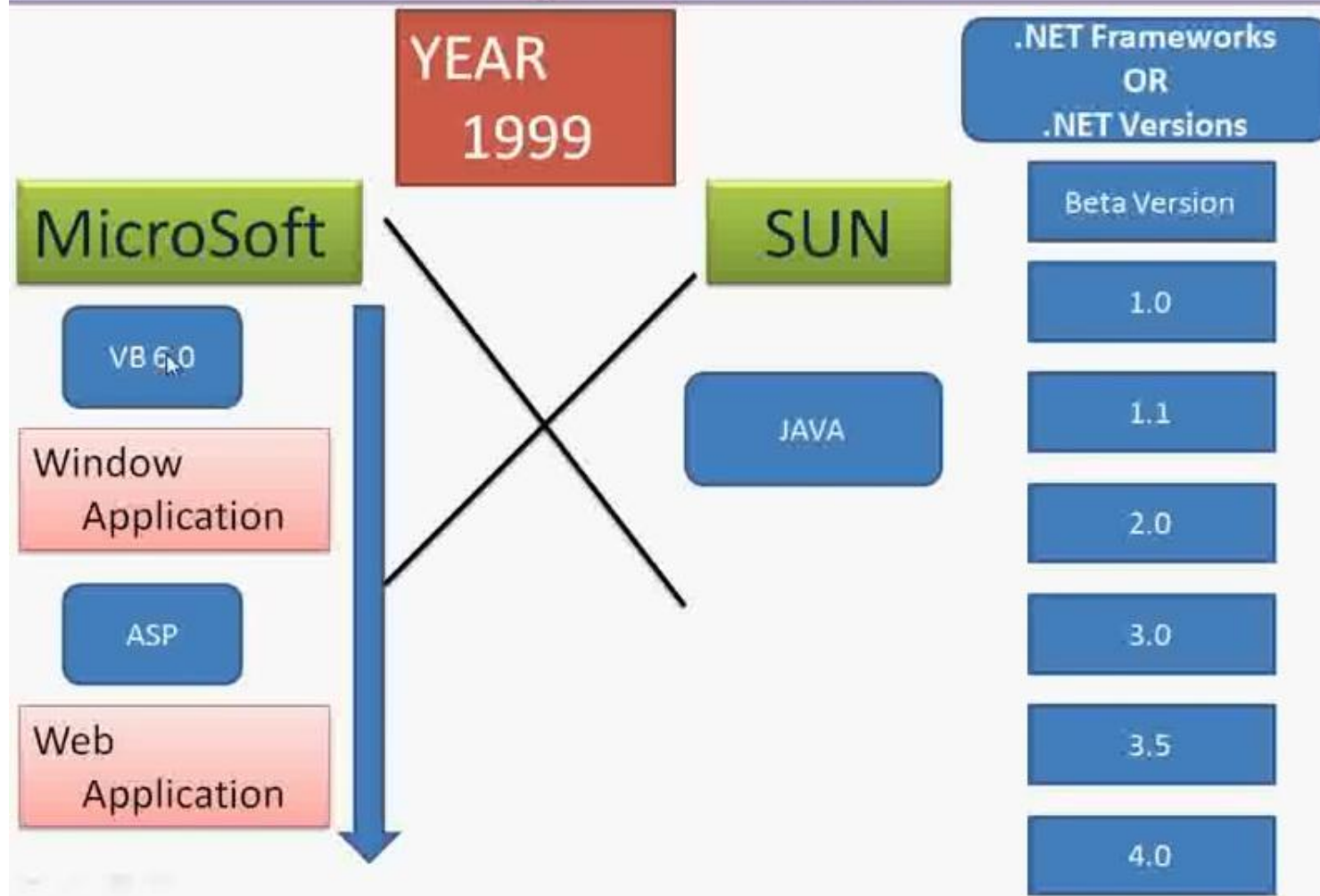


When to Use Where



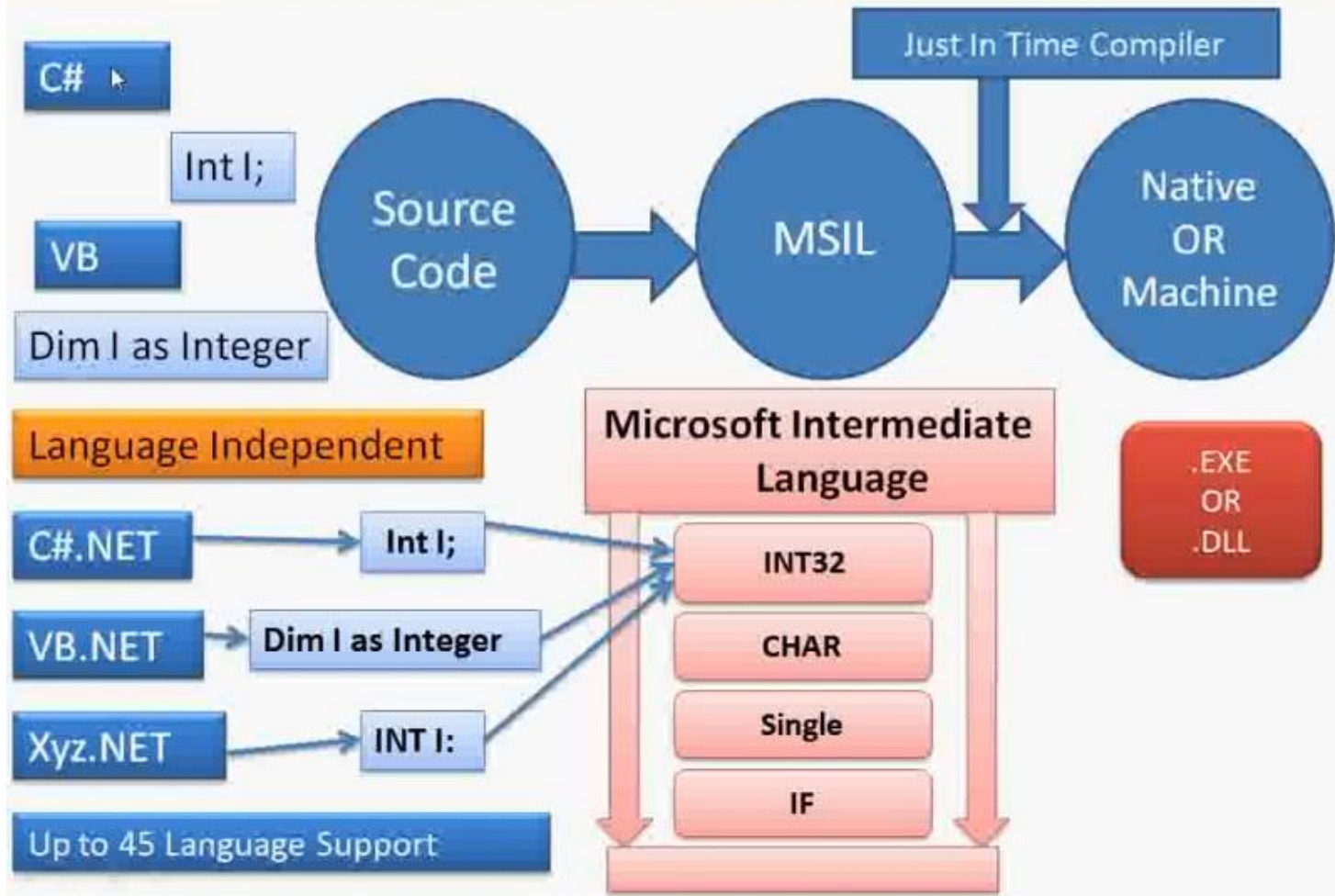


Why .NET Come



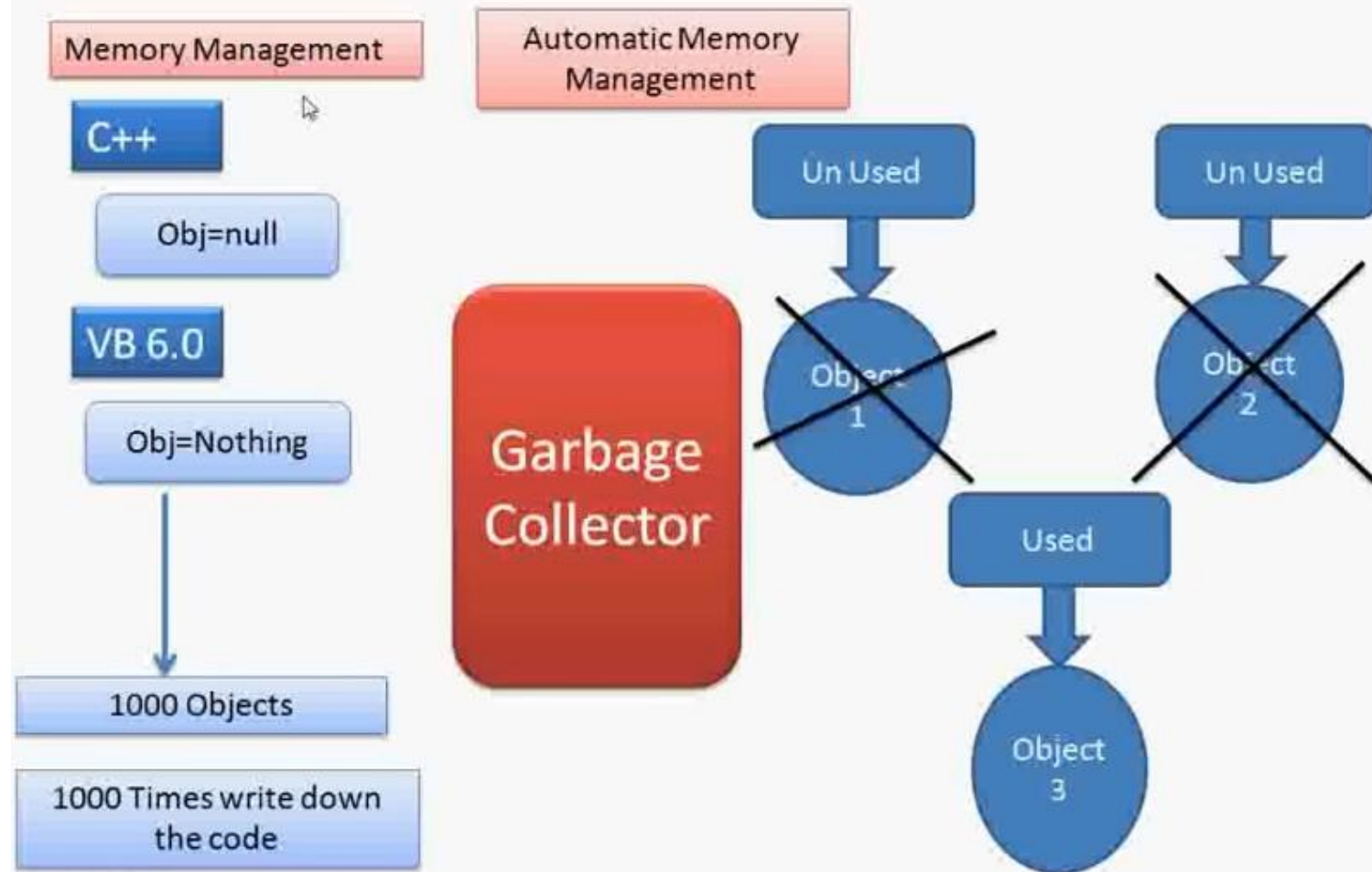


Common Language Runtime





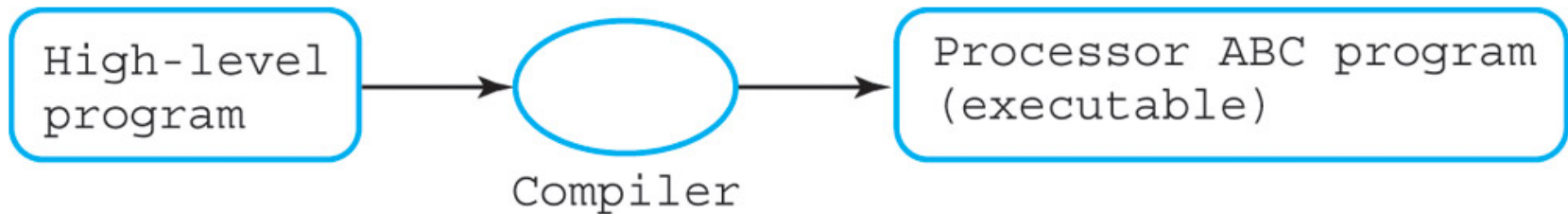
Automatic Memory Management





Compiler

- Software that translate high-level language (C++, Java, C#, etc) to machine language.



- Compiler X can covert high-level Y to machine language Z.



Compiler Example

C++ Code

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

C++
Compiler for
Intel

```
.MODEL Small
.STACK 100h
.DATA
    db msg 'Hello, world!$'
.CODE
start:
    mov ah, 09h
    lea dx, msg ; or mov dx, offset msg
    int 21h
    mov ax, 4C00h
    int 21h
end start
```

Intel x86 machine
code

C++
Compiler for
Motorola

```
;print
    move.l    #Hello, -(A7)
    move.w    #9, -(A7)
    trap      #1
    addq.l    #6, A7

;wait for key
    move.w    #1, -(A7)
    trap      #1
    addq.l    #2, A7

;exit
    clr.w     -(A7)
    trap      #1

Hello
    dc.b      'Hello, world!', 0
```

Motorola 68000
machine code

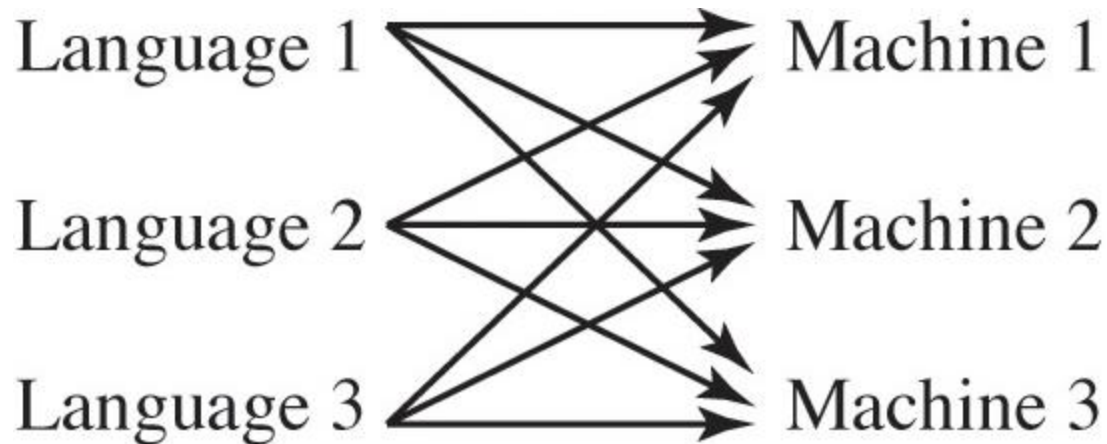


Problem

- A big problem facing developers is the many different types of processors that run code.
- Windows, Macintosh, and Unix machines use a wide variety of hardware, as do personal digital assistants, cell phones, large computers, and other platforms.
- One way to make a program work on each of these devices is to translate the program to the native instruction



- So if we have 3 programming languages and 3 devices, how many compilers do we need?

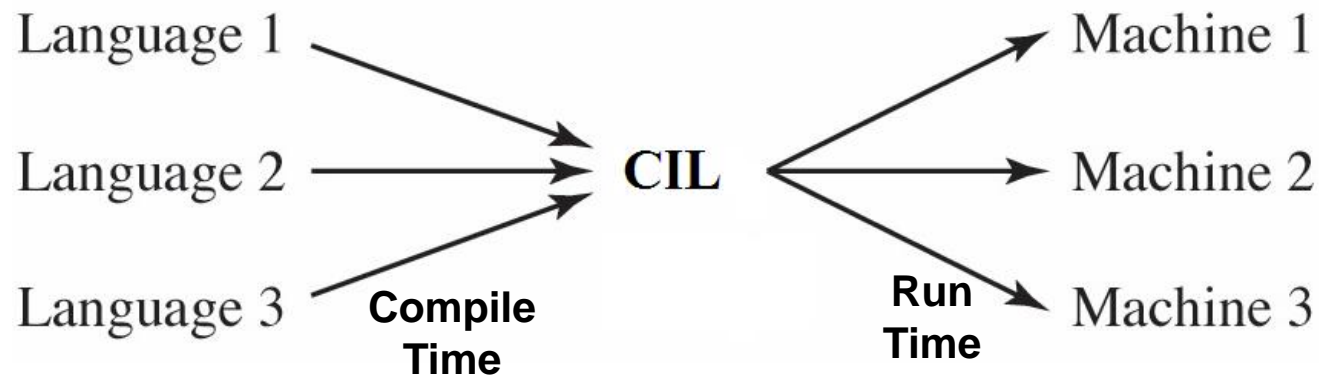


- So, how they solved this?!



Two Steps Compilation Process

- Compilation is done in two steps:
 - At compile time: compile each language (C#, C++, etc) to Common Intermediate Language (CIL)
 - At runtime: Common Language Runtime (CLR) uses a Just In Time (JIT) compiler to compile the CIL code to the native code for the device used





Common Intermediate Language (CIL)

- Much like the native languages of devices.
- CIL was originally known as Microsoft Intermediate Language (MSIL).
- CIL is a CPU- and platform-independent instruction set.
- It can be executed in any environment supporting the .NET framework
- Hello World Example in CIL

```
.assembly Hello {}  
.method public static void Main() cil managed  
{  
    .entrypoint  
    .maxstack 1  
    ldstr "Hello, world!"  
    call void [mscorlib]System.Console::WriteLine(string)  
    ret  
}
```



Common Language Runtime (CLR)

- The Common Language Runtime (CLR) manages the execution of code.
- CLR uses Just-In-Time (JIT) compiler to compile the CIL code to the native code for device used.
- Through the runtime compilation process CIL code is verified for safety during runtime, providing better security and reliability than natively compiled binaries.
- Native image generator compilation (NGEN) can be used to produces a native binary image for the a specific environment. What is the point?



The .NET Framework

.NET Framework Services

- Common Language Runtime
- Windows® Forms
- ASP.NET
 - Web Forms
 - Web Services
- ADO.NET, evolution of ADO
- Visual Studio.NET



Common Language Runtime (CLR)

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
 - Object declaration, creation and use
 - Data types,language libraries
 - Error and exception handling
 - Interactive Development Environment (IDE)



Common Language Runtime

- Development
 - Mixed language applications
 - Common Language Specification (CLS)
 - Common Type System (CTS)
 - Standard class framework
 - Automatic memory management
 - Consistent error handling and safer execution
 - Potentially multi-platform
- Deployment
 - Removal of registration dependency
 - Safety – fewer versioning problems



Common Language Runtime

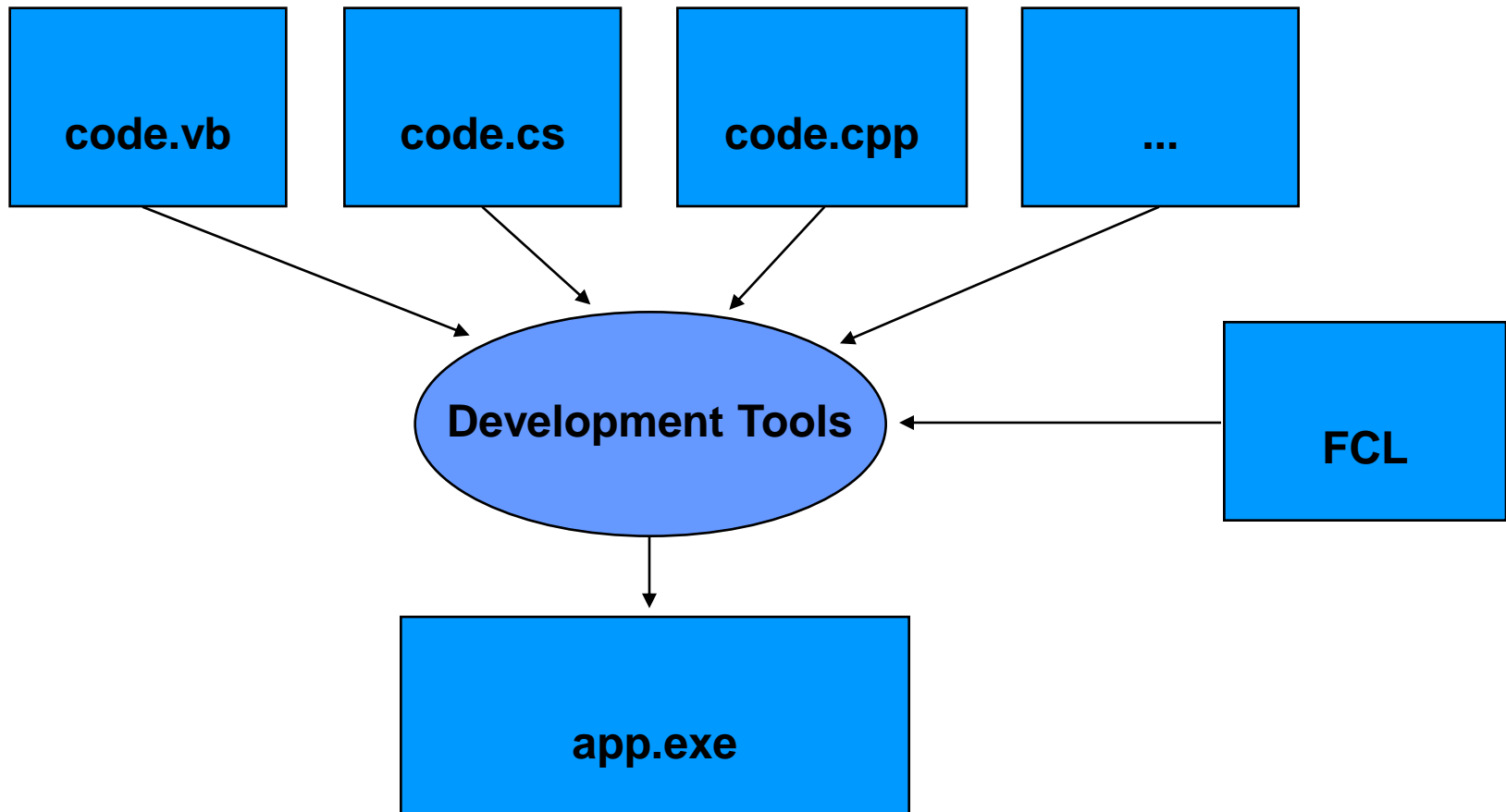
Multiple Language Support

- **CTS is a rich type system built into the CLR**
 - Implements various types (int, double, etc)
 - And operations on those types
- **CLS is a set of specifications that language and library designers need to follow**
 - This will ensure interoperability between languages



.NET is multi-language

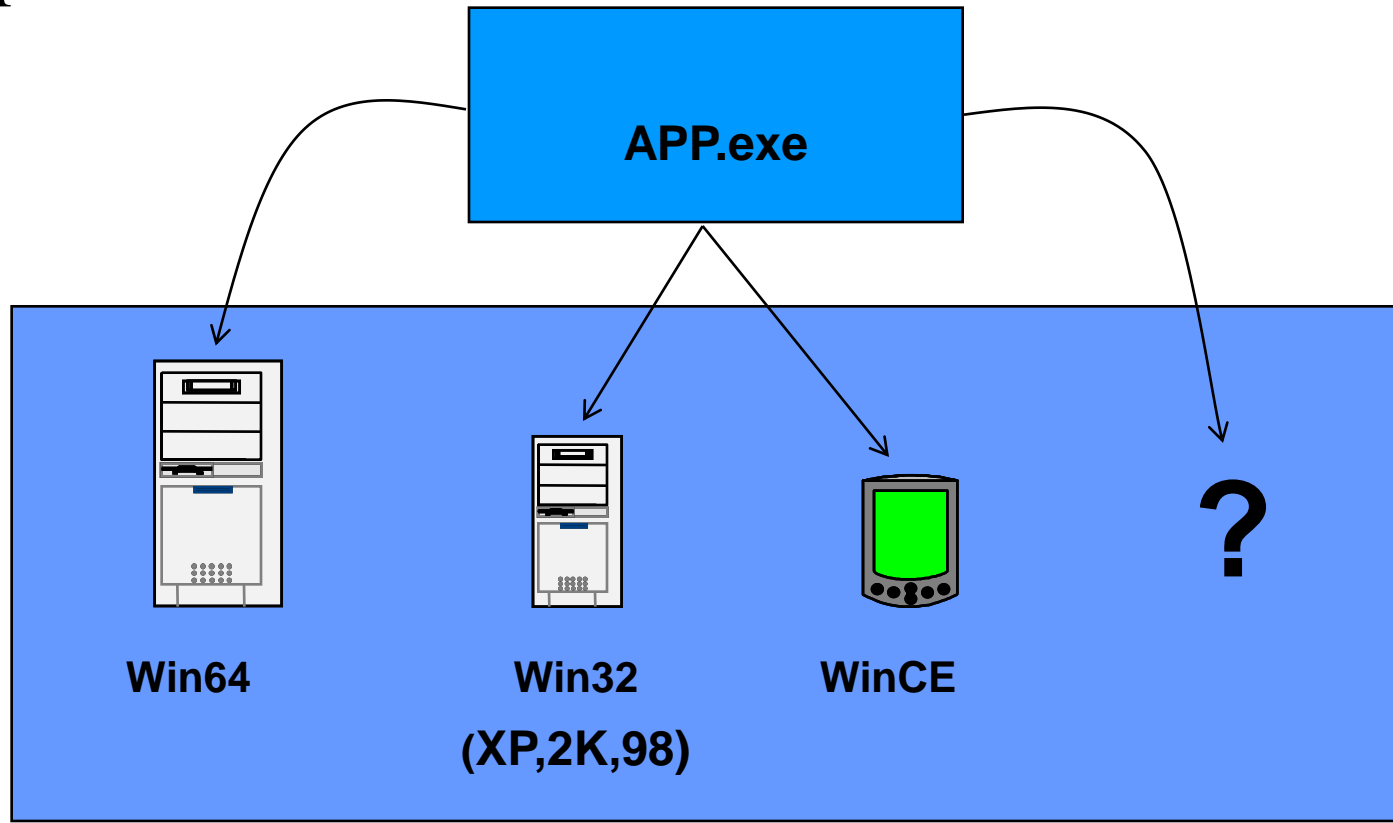
- .NET supports VB, C# (C-sharp), C++, J# (Java 1.2), Eiffel, etc.





.NET is cross-platform

- Compiled .NET apps run on any supported platform:





How is cross-platform achieved?

- Cross-platform execution realized in two ways:
 1. apps are written against *Framework Class Library* (FCL), not underlying OS
 2. compilers generate generic assembly language which must be executed by the *Common Language Runtime* (CLR)



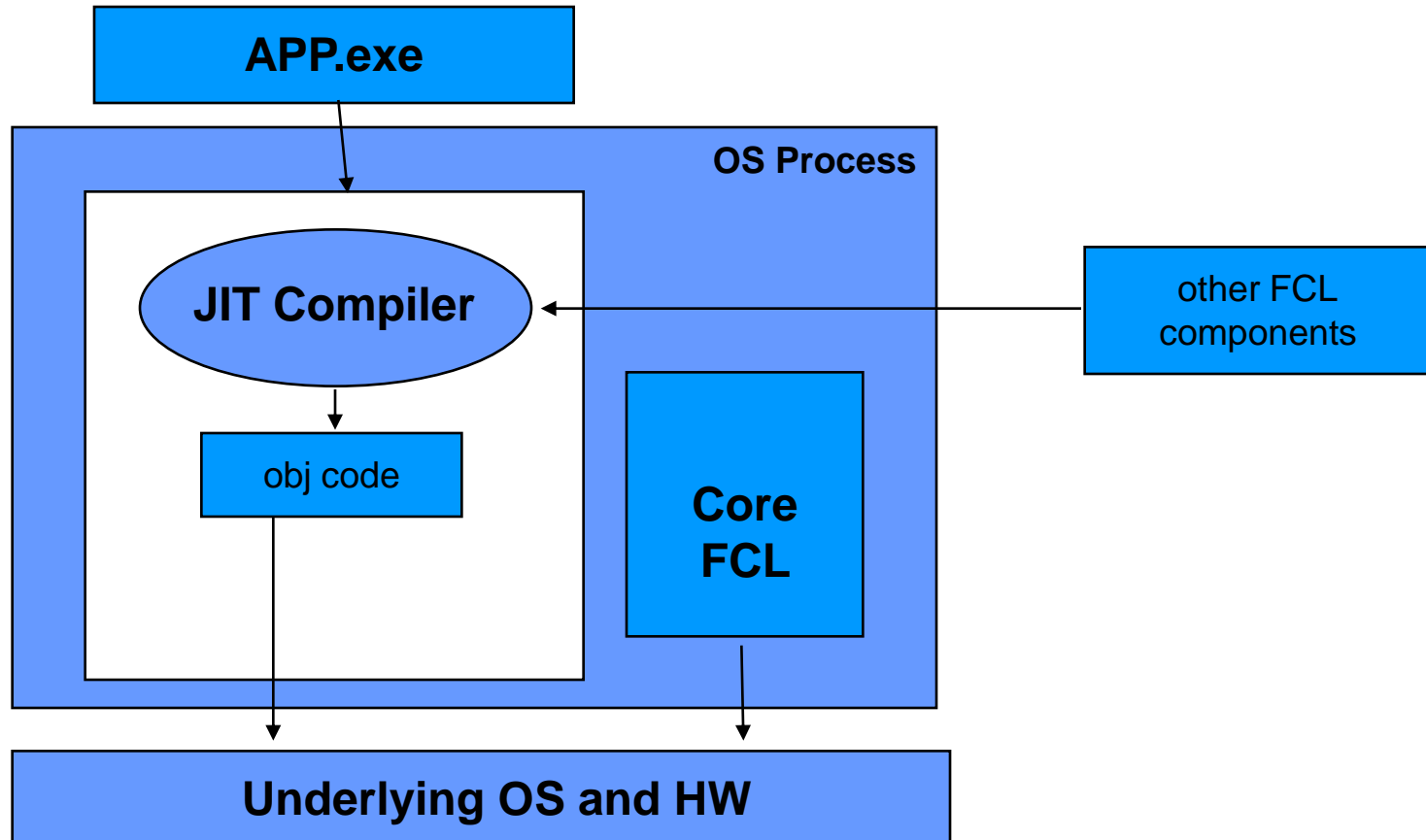
(1) FCL

- Framework Class Library
 - 1000's of predefined classes
 - common subset across all platforms & languages
 - networking, database access, XML processing, GUI, Web, etc.
- Goal?
 - FCL is a portable operating system



(2) CLR-based execution

- Common Language Runtime must be present to execute code:





Implications of .NET's execution model

1. Clients need CLR & FCL to run .NET apps

- available via *Redistributable .NET Framework*
- 20MB download
- runs on 98 and above, NT (sp6a) and above

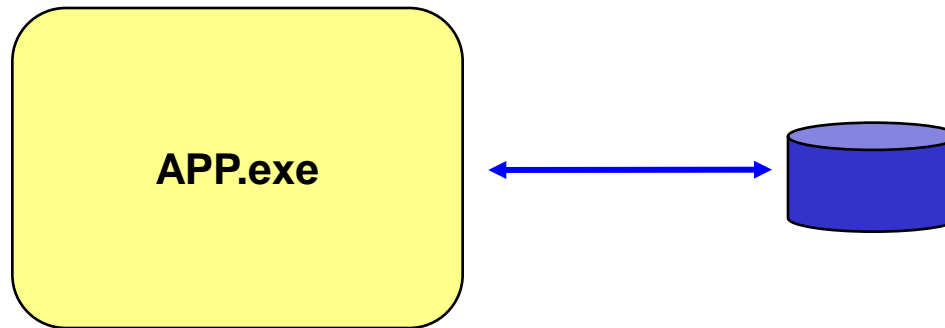
2. Design trade-off...

- + managed execution (memory protection, verifiable code, etc.)
- + portability:
- slower execution?



Monolithic

- Monolithic app: all source code compiled into one .EXE

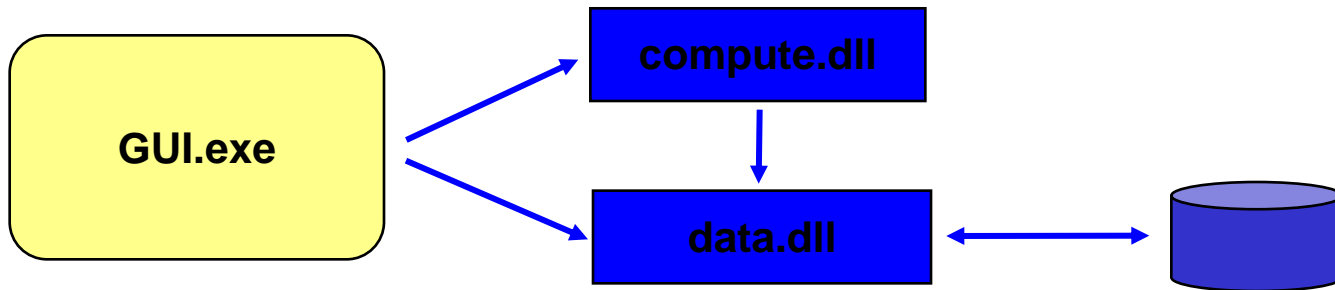


– ***not*** the norm on Windows...



Component-based

- Component-based app: .EXE + 1 or more .DLLs



– standard practice on Windows...



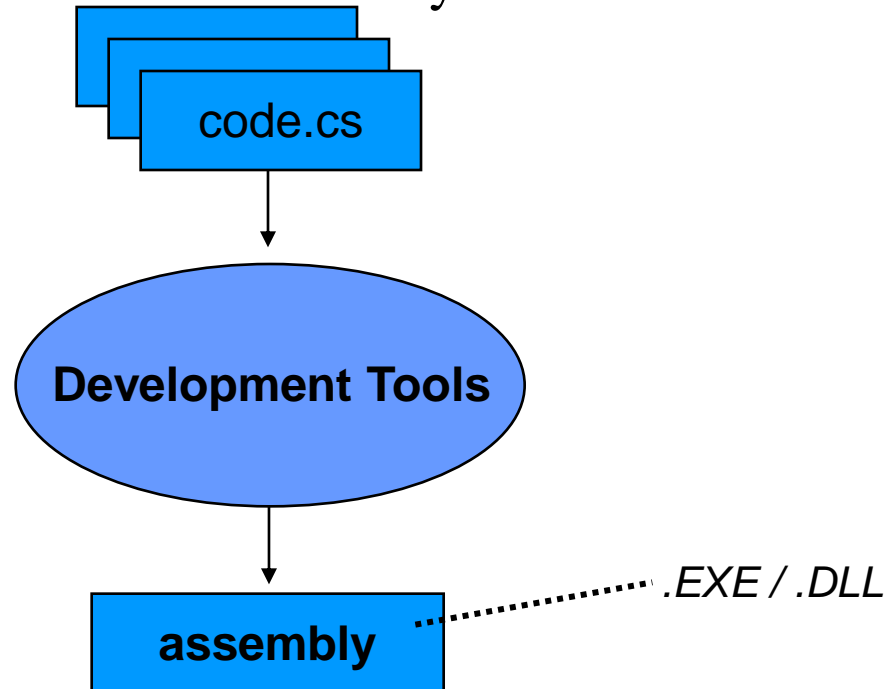
Why component-based?

- Many motivations:
 - team programming
 - multi-language development (I like VB, you like C#)
 - code reuse (e.g. across different .EXEs)
 - independent updating (update just component X)
- FCL ships as a set of components!



Assemblies

- .NET packages components into *assemblies*
- 1 assembly = 1 or more compiled classes
 - .EXE represents an assembly with classes + Main program
 - .DLL represents an assembly with classes





Meta Data

- Metadata (meta data, or sometimes metainformation) is "data about other data", of any sort in any media. An item of metadata may describe an individual datum, or content item, or a collection of data including multiple content items and hierarchical levels, for example a database schema. In data processing, metadata is definitional data that provides information about or documentation of other data managed within an application or environment.



Difference between Metadata and Manifest

- "Metadata is data about data and Manifest is data about assembly!!"
- Manifest Maintains the information about the assemblies like version, name local and an optional strong name that uniquely identifying the assembly.
- This manifest information is used by the CLR. The manifest also contains the security demands to verify this assembly. It also contains the names and hashes of all the files that make up the assembly.
- The .NET assembly manifest contains a cryptographic hash of different modules in the assembly. And when the assembly is loaded, the CLR recalculates the hash of the modules at hand, and compares it with the embedded hash. If the hash generated at runtime is different from that found in the manifest, .NET refuses to load the assembly and throws an exception.
- Metadata means Data about the data. metadata yields the types available in that assembly, viz. classes, interfaces, enums, structs, etc., and their containing namespaces, the name of each type, its visibility/scope, its base class, the interfaces it implemented, its methods and their scope, and each method's parameters, type's properties, and so on.
- The assembly metadata is generated by the high-level compilers automatically from the source files. The compiler embeds the metadata in the target output file, a dll, an .exe or a .net module in the case of multi-module assembly.



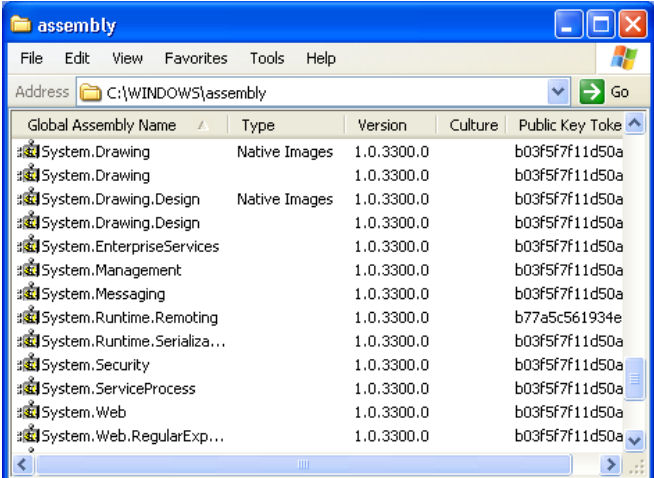
Assembly resolution

- How does CLR find assemblies?
- For now, simple answer is sufficient:
 - our DLLs must reside in same directory as our EXE
 - FCL assemblies reside in GAC
 - CLR looks in GAC first, then EXE's directory...



GAC?

- GAC = Global Assembly Cache
 - C:\Windows or C:\WinNT directory



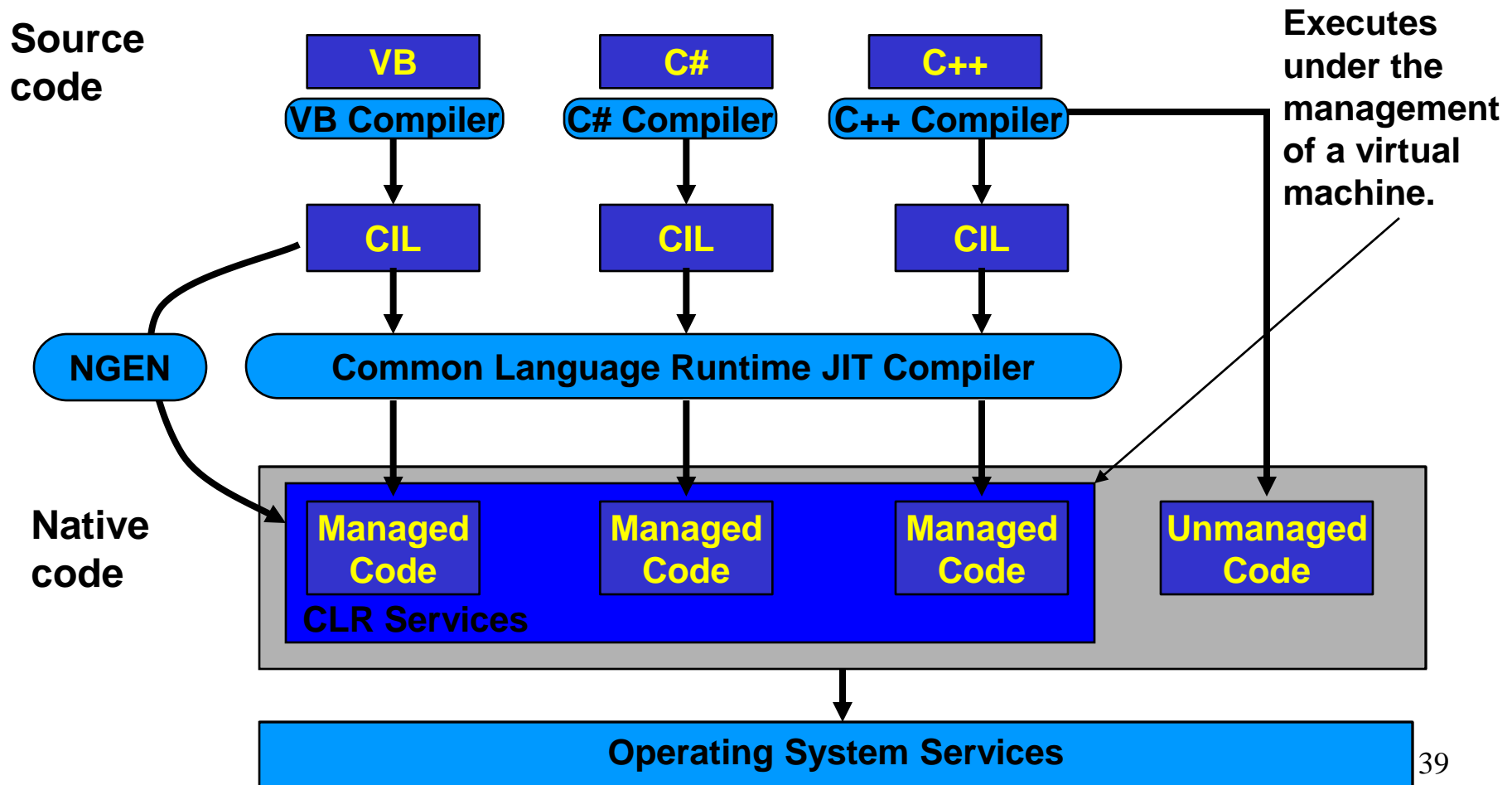
Global Assembly Name	Type	Version	Culture	Public Key Token
System.Drawing	Native Images	1.0.3300.0		b03f5f7f11d50a
System.Drawing		1.0.3300.0		b03f5f7f11d50a
System.Drawing.Design	Native Images	1.0.3300.0		b03f5f7f11d50a
System.Drawing.Design		1.0.3300.0		b03f5f7f11d50a
System.EnterpriseServices		1.0.3300.0		b03f5f7f11d50a
System.Management		1.0.3300.0		b03f5f7f11d50a
System.Messaging		1.0.3300.0		b03f5f7f11d50a
System.Runtime.Remoting		1.0.3300.0		b77a5c561934e
System.Runtime.Serializa...		1.0.3300.0		b03f5f7f11d50a
System.Security		1.0.3300.0		b03f5f7f11d50a
System.ServiceProcess		1.0.3300.0		b03f5f7f11d50a
System.Web		1.0.3300.0		b03f5f7f11d50a
System.Web.RegularExp...		1.0.3300.0		b03f5f7f11d50a

- Observations:
 - explorer yields a flat view of GAC
 - command-shell yields actual representation
 - GAC can hold different versions of the same assembly
 - some assemblies have been pre-JIT ("native image")
 - tamper proof via digital signatures...



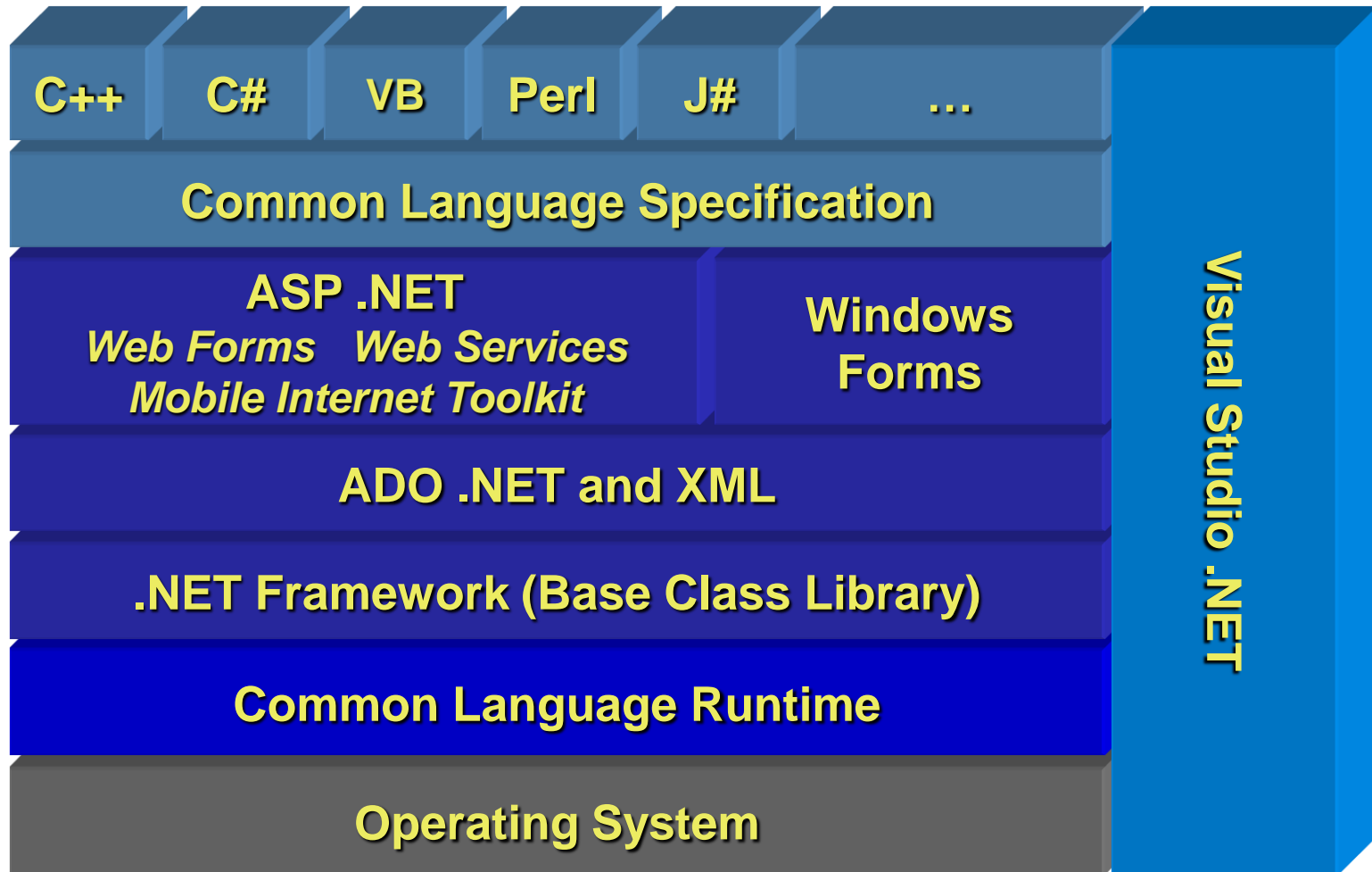
Compilation Process

- So if we have 3 programming languages and 3 devices, how many compilers do we need?





.NET Framework Visual Studio .NET





The .NET Framework Stack



CIL

- C# compiler translates C# source code into CIL

C# source

```
Calc c = new Calc();  
int sum = c.Add(2, 4);
```



C# compiler



CIL

```
.locals init ([0] class Calc c, [1] int32 sum)  
newobj instance void Calc::.ctor()  
stloc.0 // c = ptr to new object  
ldloc.0  
ldc.i4.2 // pass second arg  
ldc.i4.4 // pass first arg  
callvirt instance int32 Calc::Add(int32,int32)  
stloc.1 // sum = retval
```



Platform and Language Independent

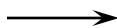
- What we have described so far will lead us to Platform independent environment. How?
- Can we use compiled classes written in X language in a program written in Y language?
- VB.NET + C#.NET code



Language interoperability

- All .NET languages can interoperate

C# calling
VB.NET



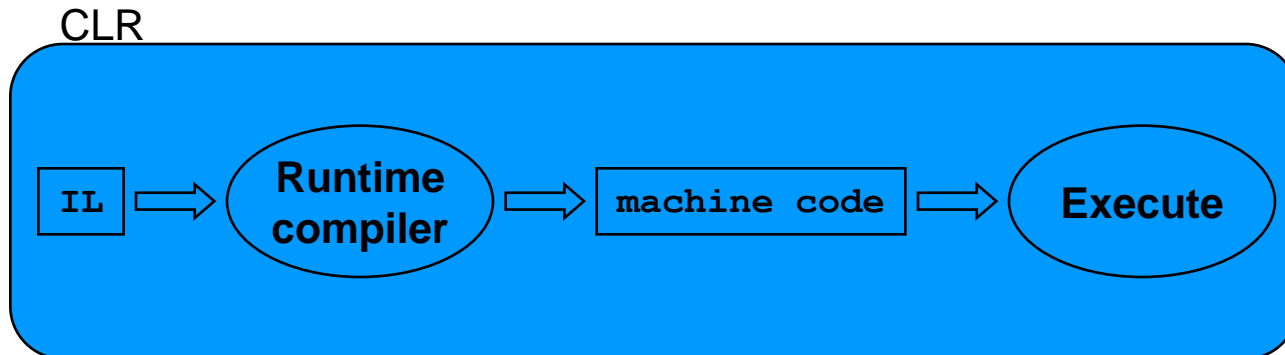
```
class Hello
{
    static void Main()
    {
        System.Console.WriteLine(Greeting.Message());
    }
}
```

```
Class Greeting
    Shared Function Message() As String
        Return "hello"
    End Function
End Class
```



Execution engine

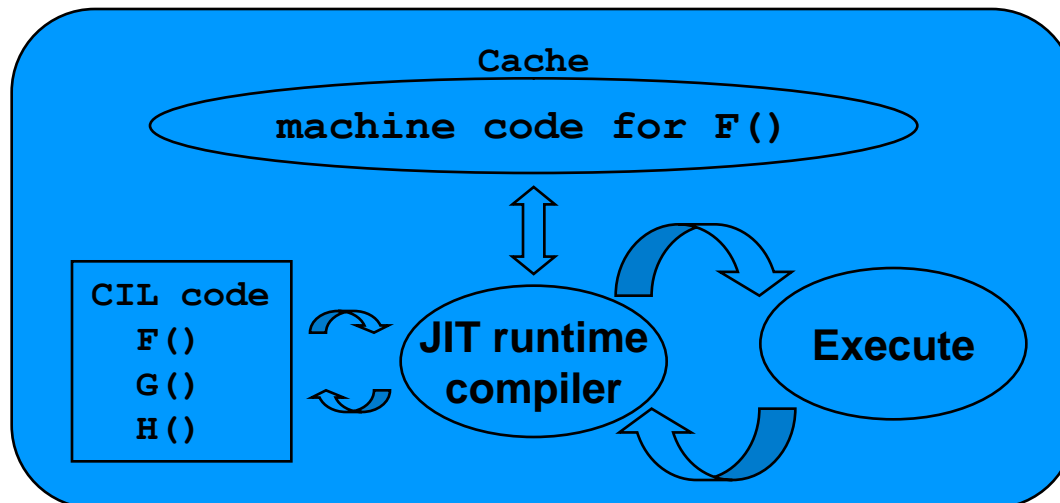
- *Common Language Runtime* (CLR) is the execution engine
 - loads IL
 - compiles IL
 - executes resulting machine code





JIT runtime compile

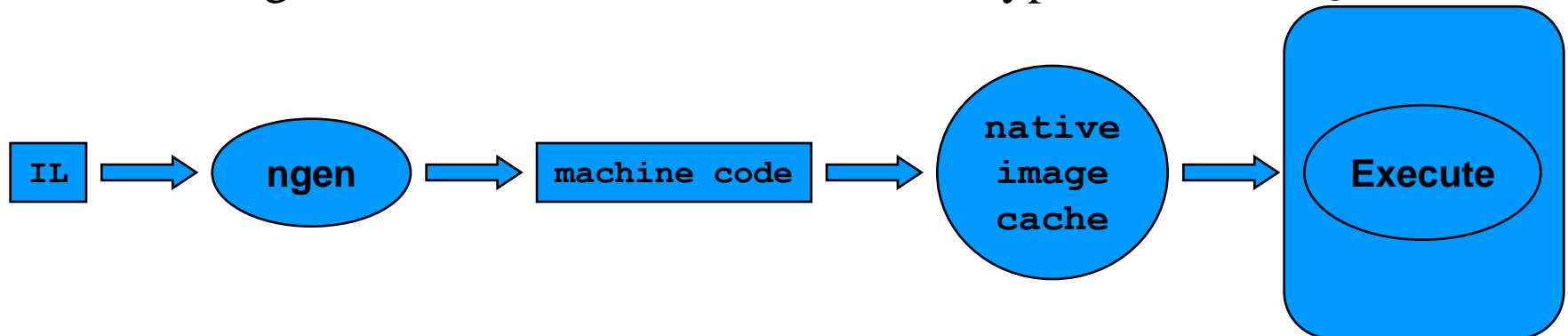
- CIL is compiled into machine code at runtime by the CLR
 - compiles methods as needed
 - called *just in time* (JIT) compile
- JIT compilation model:
 - first time method is called the IL is compiled and optimized
 - compiled machine code is cached in transient memory
 - cached copy used for subsequent calls





NGEN install time compile

- Can compile CIL into machine code when app installed
 - use native image generator **ngen.exe**
 - can speed startup time since code pre-compiled
 - but cannot do as many optimizations
 - original IL must still be available for type information CLR





Language variability

- Not all .NET languages have exactly the same capabilities
 - differ in small but important ways

C#

```
class Hello
{
    static void Main()
    {
        int i;
        uint u;
    }
}
```

signed integer →

unsigned integer →

VB.NET

```
Class Greeting
    Shared Sub Main()
        Dim i as Integer
    End Sub
End Class
```

signed integer only →



Common Language Specification

- Common Language Specification (CLS) defines type subset
 - required to be supported by all .NET languages
 - limiting code to CLS maximizes language interoperability
 - code limited to CLS called *CLS compliant*

not CLS compliant
to use `uint` in public
interface of public class



```
public class Calculator
{
    public uint Add(uint a, uint b)
    {
        return a + b;
    }
}
```



Framework Class Library (FCL)

- Namespace: A collection of classes and their methods. Example: System.Windows.Forms
- The .NET Framework class library is a library of classes, interfaces, and value types that are included in the Windows Software Development Kit (SDK).
- Namespaces are stored in DLL files called assemblies
- .NET applications must have “references” to these DLLs so that their code can be linked in
- Included in a C# program with the *using* keyword
 - If not included, you must give the fully qualified name of any class method or property you use
- System.Windows.Forms.MessageBox.Show(...)



Some Important .Net Namespaces in FCL

- System Core data/auxiliary classes
- System.Collections Resizable arrays + other containers
- System.Data ADO.NET database access classes
- System.Drawing Graphical Output classes (GDI+)
- System.IO Classes for file/stream I/O
- System.Net Classes to wrap network protocols
- System.Threading Classes to create/manage threads
- System.Web HTTP support classes
- System.Web.Services Classes for writing web services
- System.Web.UI Core classes used by ASP.NET
- **System.Windows.Forms** **Classes for Windows GUI apps**
- See online help on 'Class Library' or use MSDN (3CDs)



Required CLR

- CLR and .NET Framework required to run .NET app
 - will be incorporated into Windows and service packs
 - developers install as part of .NET Framework SDK
 - users can run **dotnetredist.exe**