Chapter 1

Introduction to .NET

Objective

Understanding

- What was before .NET
- Vision behind .NET
- .NET strengths
- .NET Framework
- Working of CLR
- Common Language Integration
- Common Type System
- Common Language Specification

Syllabus

DOTNET Framework

- Introduction to DOTNET
- DOT NET class framework
- Common Language Runtime
  - Overview, Elements of .NET application, Memory Management, Garbage Collection
- Common Language Integration
- Common type system
- Assemblies: Private assemblies, Shared assemblies

Before .NET

- Windows GUI development: Win32 API, MFC, Visual Basic
- Web development: ASP
- Java – “Write once, run anywhere.”

.NET initiative

- **Vision** for embracing the **Internet** in software development

- **Vision** from Developer perspective
  - to provide a **feature-rich** application development platform and a **managed**, **protected execution environment**.

- **Framework** intends to **provide an environment** that simplifies the development, deployment and execution of distributed applications.

- **Independence** from specific language or platform
  - Supports portability and interoperability
  - Applications developed in any .NET-compatible language
    - Visual Basic.NET, Visual C++.NET, C# and more

- **Architecture** capable of existing on multiple platforms
    - Supports portability
    - Architecture can be deployed on many platforms

Design Goals of .NET

- **Simplify Software Development**
  - Provide **rich functional base classes** that are extensible and **easier to access**
  - Support **XML Web Services Development**, e.g. ASP.NET provides native support for the Simple Object Access Protocol (SOAP)

- **code-execution environment**
  - Promotes safe execution of code
  - Eliminates the performance problems of scripted or interpreted environments.

Design Goals of .NET

- **Language Integration**
  - Classes implemented in one language can inherit from classes, catch exceptions, and take advantage of polymorphism across different languages
  - Achieved by **Common Type System**

- **Unify Programming Models**
  - All .NET languages must provide **same set of Framework classes** and services outlined by **Common Language Specification**
  - i.e. .NET languages must be **interoperable**.

Design Goals of .NET

- Leverage standard Web protocols / **Internet Interoperation**
  - Deep XML support
  - .NET Framework uses the industry-supported **SOAP protocol**, which is based on the widely accepted **XML and HTTP standards.**

- Simplify Code Deployment and Maintenance
  - Simplify Code Deployment, Minimizes software deployment and versioning conflicts and Maintenance
  - Provide richer compilers and runtime support

Design Goals of .NET

- Reliability
  - .NET Framework provide **reliable** and a **robust runtime** or infrastructure
  - Microsoft .NET **requires type safety**.
  - CLR must recognize and verify types before they can be loaded and executed. This **reduces programming errors** and prevents buffer overruns
  - Robustness
    - Garbage collection takes care of freeing memory, which ensures accumulation of unused resources will not bring app down.
- Security
  - Framework protect access to specific parts of the executable code
    - declarative security checks
    - imperative security checks

What is dot net core?

- https://youtu.be/eIHKZfgddLM?list=PLdo4fOcmZ0oWoazjhXQzBKMrFuArxpW80

- NET Core is a free, cross-platform, open source **developer platform** for building many different types of applications.

- With .NET, you can use **multiple languages**, editors, and **libraries** to build for web, mobile, desktop, games, and IoT

Languages

- You can write .NET apps in C#, F#, or Visual Basic.
- **C#** is a simple, modern, object-oriented, and type-safe programming language.
- **F#** is a cross-platform, open-source, **functional programming language** for .NET. It also includes object-oriented and imperative programming.
- **Visual Basic** is an approachable language with a simple syntax for building type-safe, object-oriented apps.

Cross Platforms

- **.NET Core** (runs anywhere) is a cross-platform .NET implementation for websites, servers, and console apps on Windows, Linux, and macOS.

- **.NET Framework** supports websites, services, desktop apps, and more on Windows.

- **Xamarin/Mono** is a .NET implementation for running apps on all the **major mobile operating systems.**

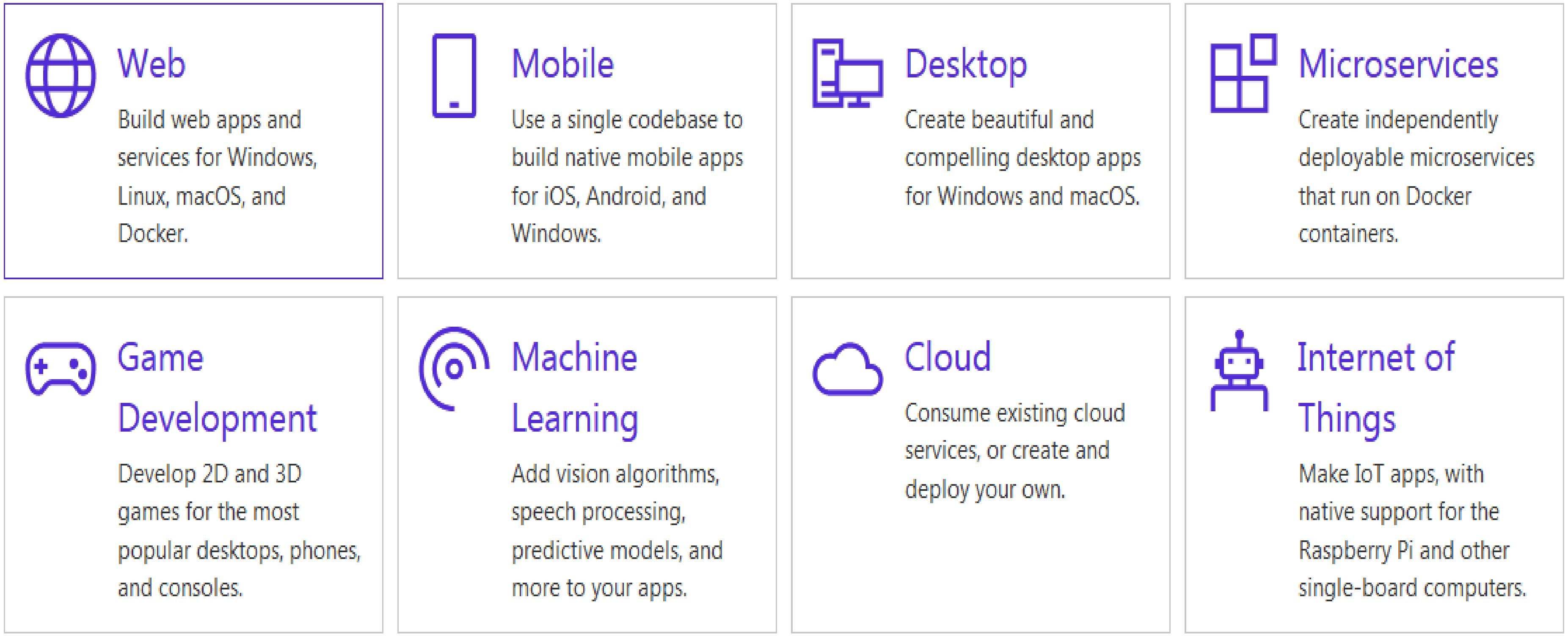
One consistent API

- **NET Standard** is a base set of APIs that are common to all .NET implementations.

- Each implementation can also **expose additional APIs that are specific to the operating systems** it runs on.

- For example, .NET Framework is a Windows-only .NET implementation that includes **APIs for accessing the Windows Registry.**

Applications build using .NET



Web
Build web apps and services for Windows, Linux, macOS, and Docker.

Mobile
Use a single codebase to build native mobile apps for iOS, Android, and Windows.

Desktop
Create beautiful and compelling desktop apps for Windows and macOS.

Microservices
Create independently deployable microservices that run on Docker containers.

Game Development
Develop 2D and 3D games for the most popular desktops, phones, and consoles.

Machine Learning
Add vision algorithms, speech processing, predictive models, and more to your apps.

Cloud
Consume existing cloud services, or create and deploy your own.

Internet of Things
Make IoT apps, with native support for the Raspberry Pi and other single-board computers.

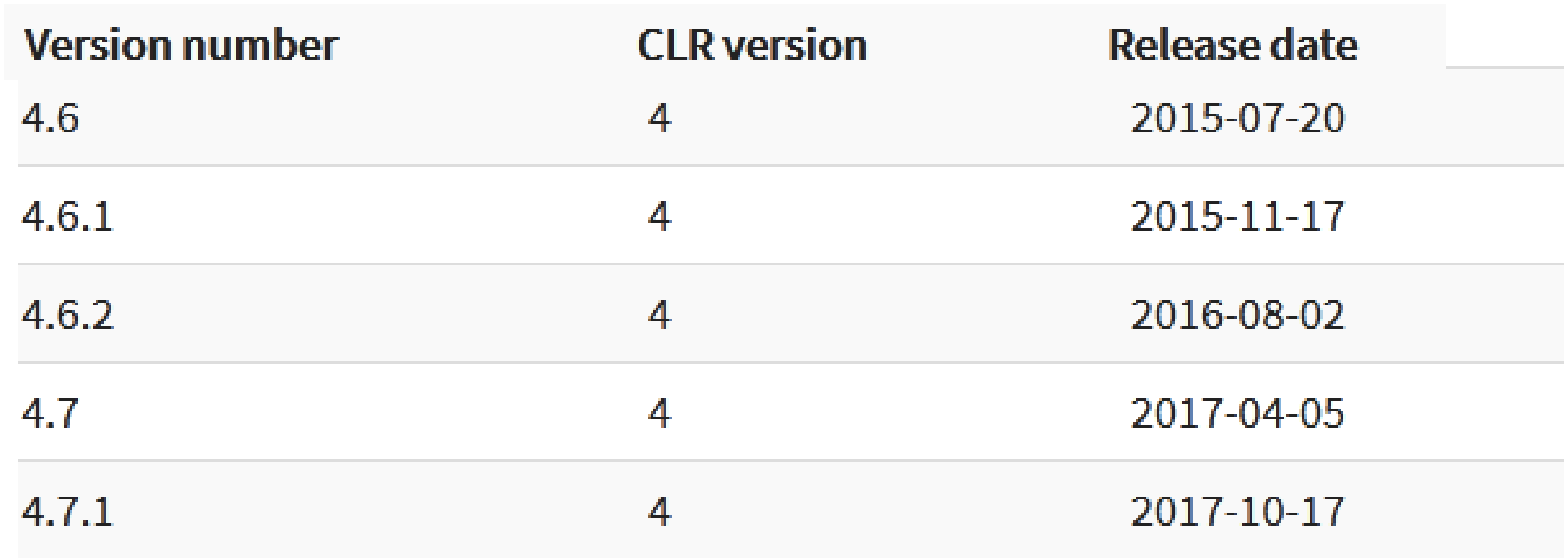
Some advantages .NET has over Java

- Java is not just a platform, but a language. .NET is language independent. Adopting .NET **does not force you to adopt a single, or even specific language**. .NET can even support the Java language, and J# is very close to Java.
- **Language interoperability**.
- .NET can coexist with and even integrate with existing Win32 code.
- Moving to Java from Win32 is a complete paradigm shift. Moving to .NET is a more natural path for Win32 developers.

Version History

Date	Version	Remarks
16-Jan-02	1.0	First version released together with Visual Studio .NET
24-Apr-03	1.1	released together with Windows Server 2003 released together with Visual Studio .NET 2003
7-Nov-05	2.0	codename Whidbey released together with Visual Studio 2005 and Visual Web Developer Express and SQL Server 2005
21-Nov-06	3.0	
19-Nov-07	3.5	Released with Visual Studio 2008 and Windows Server 2008
11-Aug-08	3.5 SP1	Released with Visual Studio 2008 SP1
12-Apr-10	4.0	Parallel extensions and other .NET Framework 4 features
15-Aug-12	4.5	
17-Oct-13	4.5.1	
8-May-14	4.5.2	

Version History



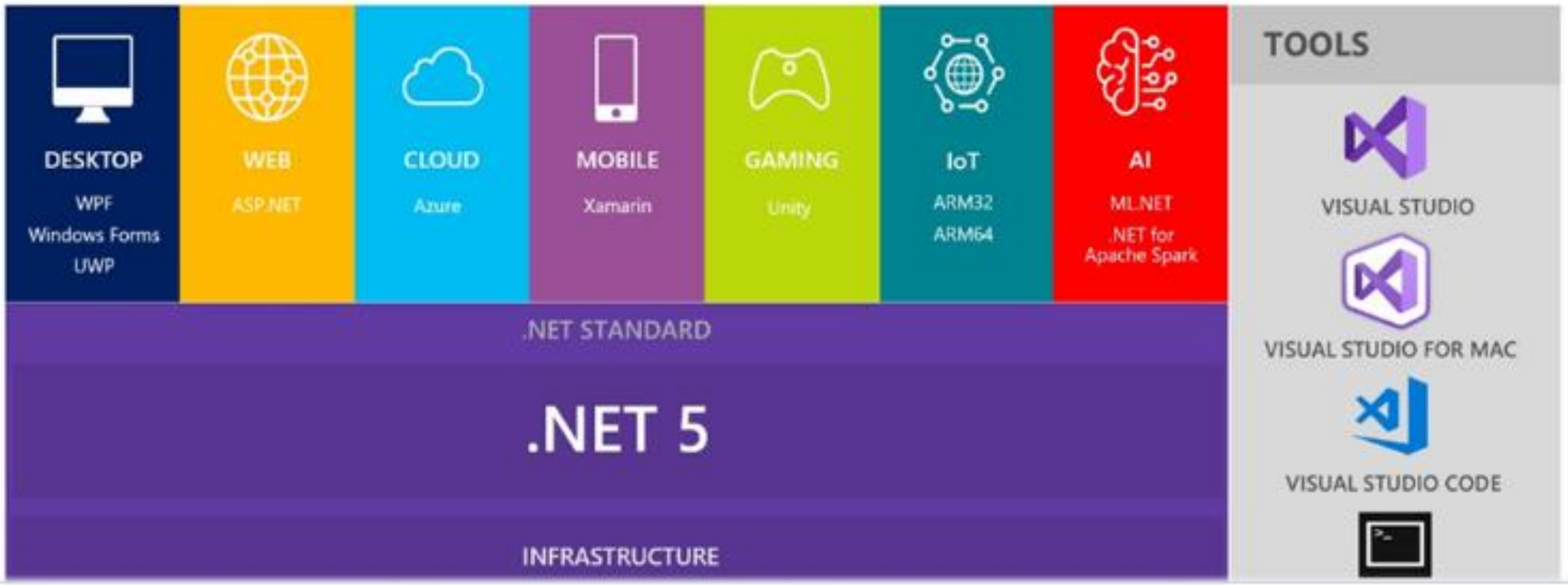
Version number	CLR version	Release date
4.6	4	2015-07-20
4.6.1	4	2015-11-17
4.6.2	4	2016-08-02
4.7	4	2017-04-05
4.7.1	4	2017-10-17

Latest version is 4.8 released on 18th April 2019

Future….. .NET 5

.Net 5 that is **Opensource** and **Cross-platform**, which will **replace** .Net Framework, .Net Core and Xamarin with a single unified platform called .Net 5 Framework.

.Net 5 - A Unified Platform

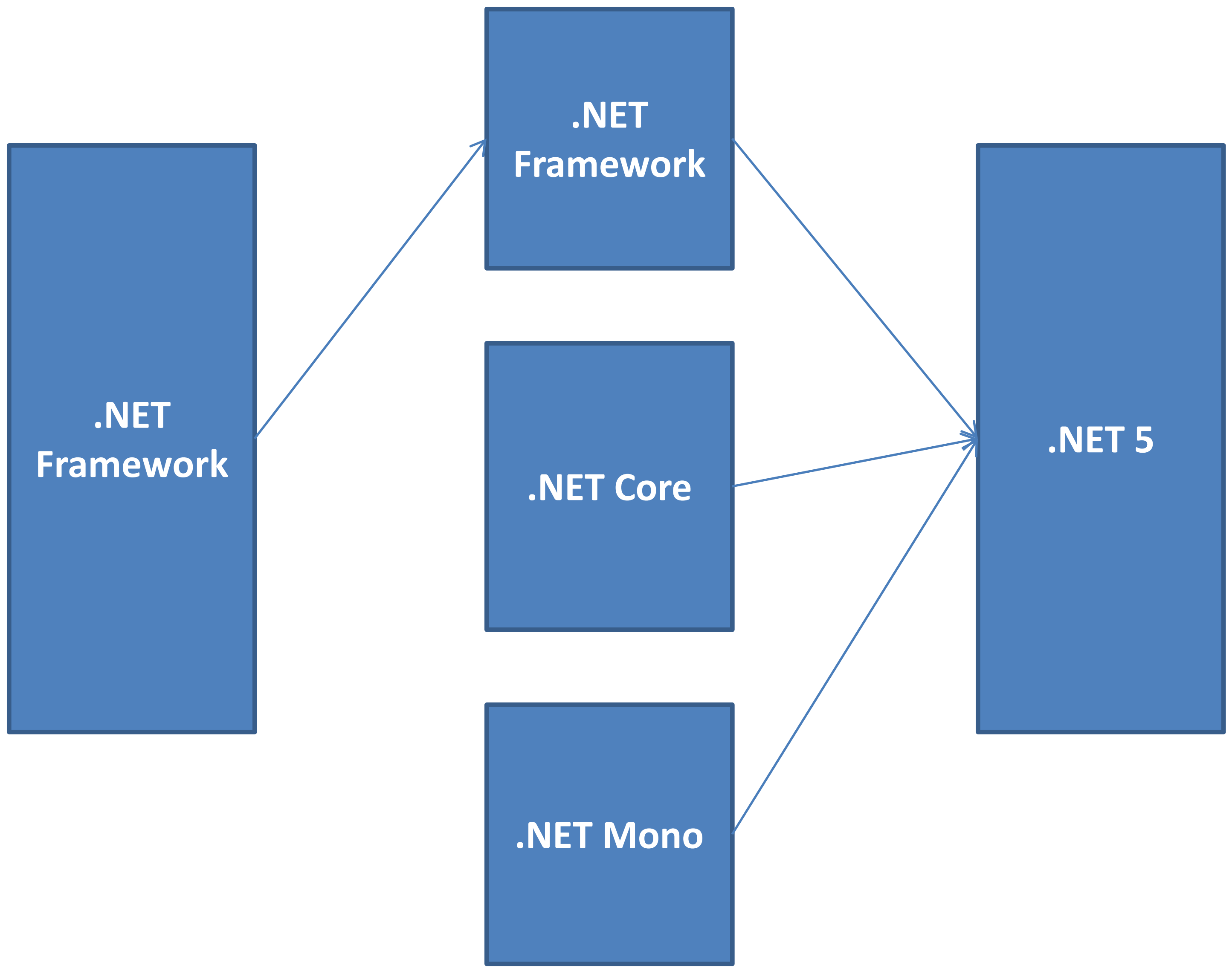


.NET 5

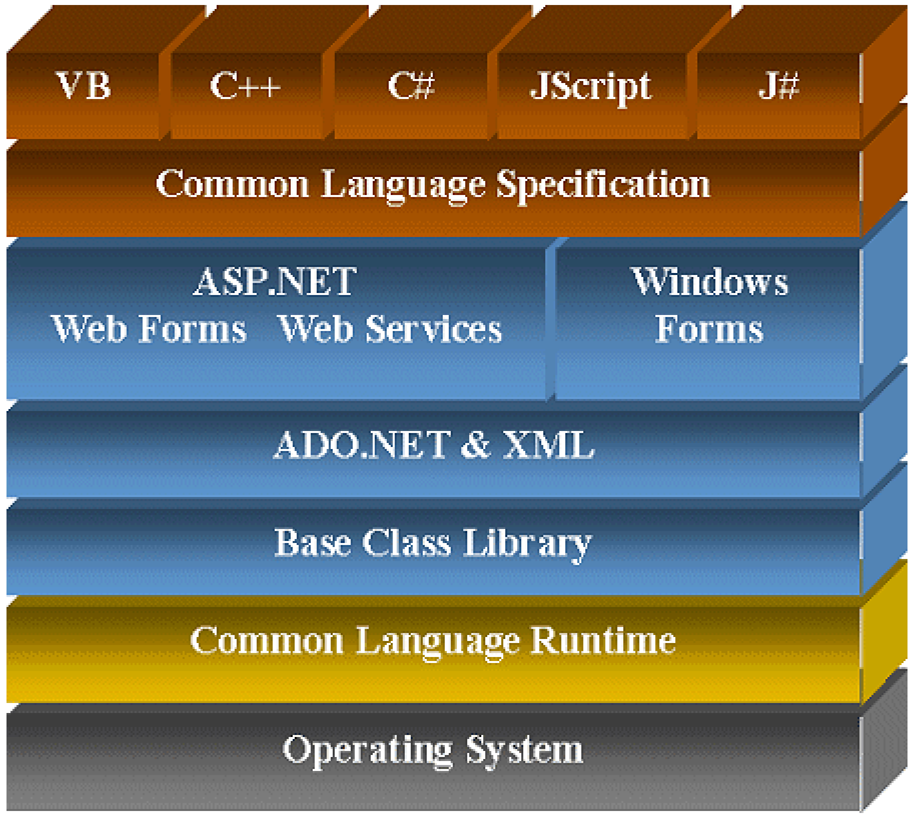
- .Net 5 will be powered to provide
  - APIs, libraries, and
  - run-time to create apps for Web, Windows, Mobile & IoT devices.

- The main goal of .Net 5 is to **empower unified .Net Client Application projects** to create deliverables for various platforms including Windows, UNIX, Linux, Legacy Windows, iOS, Andriod, HTML5, and Macintosh.

.NET framework roadmap

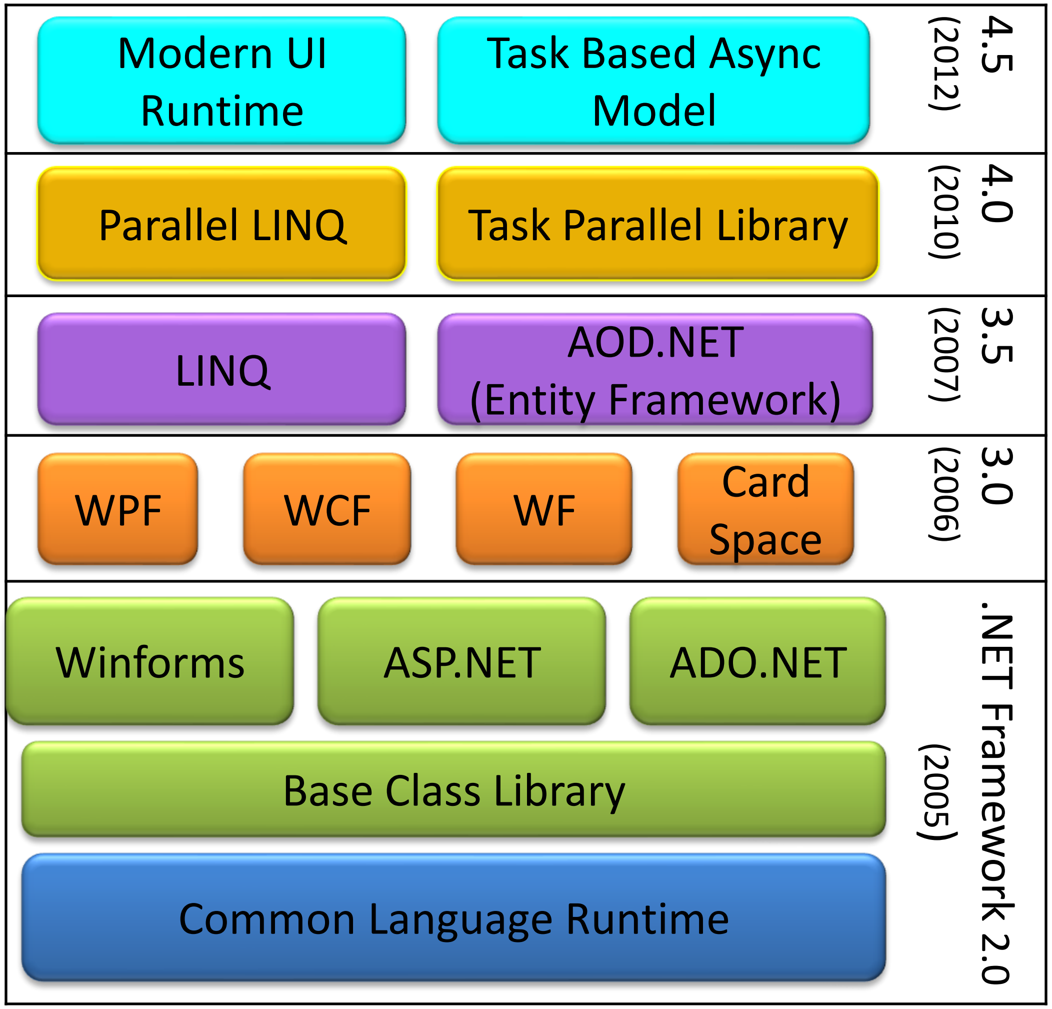


The .NET Framework Architecture

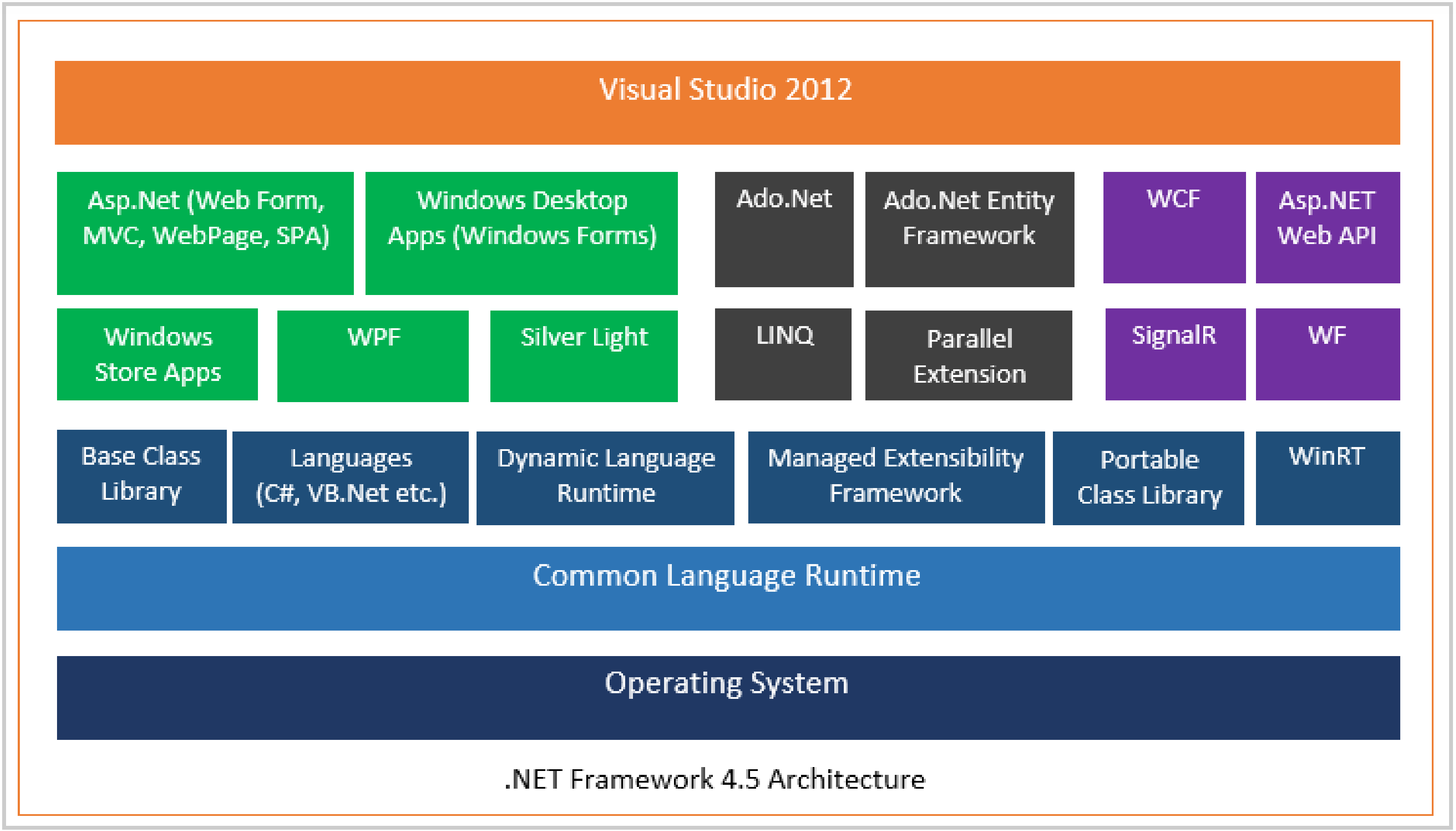


- At the heart of Microsoft .NET is the .NET Framework
  - CLR
  - Class libraries
- CLR is the **execution engine**
- Class Libraries provide **programming APIs** for building .NET applications
- ADO. Net is within Base Class libraries, which provides access to and management of data.
- ASP.NET is new shift in Web Application Development

The .NET Framework stack



DOT NET Framework 4.5



.NET Framework 4.5 Architecture

DOT NET Framework 4.5

Common Language Runtime (CLR)

- This provides the **environment to execute the** .NET Framework **programs**. Moreover, it also takes care of **memory management** and **thread management** of all the .Net programs.

Base Class Library (BCL)

- The BCL functions as a **library of various functionalities which are common for all the languages** using the .Net Framework, comprising of classes and interfaces of reusable types.

DOT NET Framework 4.5

Portable Class Library (PCL)

- The **PCL** project in Visual Studio 2012 **enables** you **to write and develop managed assemblies that function on multiple .Net Framework platforms**. One can choose their platform such as .NET Framework, Silverlight, Windows Phone 7, or Xbox 360 platforms to target, using the Portable Class Library project.

Managed Extensibility Framework (MEF)

- The MEF is a library that **enables the generation of lightweight and extensible applications**. It allows the app developers to discover and employ extensions without the use of any configurations.

DOT NET Framework 4.5

Dynamic Language Runtime (DLR)

- This **gives the runtime environment** for **languages like python** and so forth to perform under the full control of the CLR.

WinRT

- The WinRT or **Windows Runtime APIs** provide the **user interface functions for developing Windows Store apps** and allow access to various features of Windows 8 or Windows RT OS. It supports native C++, HTML, C#, VB.NET, as well as JavaScript and TypeScript.

DOT NET Framework 4.5

Asp.Net

- This helps in creating **rich internet-based, website** applications.

Windows Store Apps (Metro Style Apps)

- This is a new app specially designed to run on **Windows 8 devices** and can take the advantage of the features of the new WinRT APIs.
- The .Net Framework supports the **Windows Store Apps** which can only be distributed in the Windows 8 store.

DOT NET Framework 4.5

Desktop Apps (Windows Forms)

- The Windows Desktop app is nothing but the traditional Windows application developed for the previous Windows versions like Windows XP, Windows Vista and Windows 7 but designated with a new name for Windows 8. The classic features of the Windows Desktop app include the **Microsoft Office family products** and notepad.

Windows Presentation Foundation (WPF)

- The WPF is used for rendering user interfaces to create applications with a **rich user-experience**.
- It comes with UI applications, **2D graphics, 3D graphics and multimedia**.
- **It is a resolution-independent engine** that is built to take advantage of the hardware acceleration of modern graphics cards.
- Additionally, the WPF **makes the UI perform faster**.

DOT NET Framework 4.5

Silverlight

- A web-based, cross-browser technology, Silverlight, enables designers and developers to create **Rich Internet Applications (RIA) embedded in web pages.**

Ado.Net

- It is a set of software components that allows the developers to create a Data Access Layer to access and manipulate data from underlying data sources like SQL Server, DB2, Oracle and so forth.

Language-Integrated Query or LINQ

- **LINQ makes a query from various data sources** such as SQL databases, XML documents, Ado.Net Datasets, Various Web services, Collections and Generics using Visual Basic or C#.

DOT NET Framework 4.5

Net Entity Framework

- Ado.Net allows **access into databases** like SQL Server, Oracle, DB2 and so forth, and to **accordingly process and update the data contained in them** in an Object relational mapping (ORM) fashion.

Parallel Extension

- With the use of Parallel Extension, developers can **distribute their work code across various multiple processors in order to gain hardware advantage**.

Windows Communication Foundation (WCF)

- WCF can be used to **send across data as messages from a service endpoint to another** using WS-* standards.

DOT NET Framework 4.5

ASP.NET Web API

- ASP.NET Web API acts as a **framework for creating HTTP services that are of use to a wide range of clients such as** mobiles, iPhone and tablets and browsers.

SignalR

- ASP.NET SignalR is a **library that facilitates the process of adding real-time web functionality to applications** by making it less complicated.

- Real-time web functionality **allows the server code push content to connected clients instantaneously**, rather than having to wait for a client to request for new data.

- E.g. Dashboards and monitoring applications, collaborative applications (such as simultaneous editing of documents), job progress updates, and real-time forms.

Overview of .NET Framework Applications

Types of Applications

- Windows Forms Applications
- Windows Form Controls
- Windows Service Application
  - back ground services like IIS, SMTP
- ASP.NET Web Application
- Web Services

Microsoft Intermediate Language (MSIL)

- While compiling managed code, the compiler translates your source code into **Microsoft intermediate language (MSIL).**

- MSIL is **a CPU-independent set of instructions** that can be efficiently **converted to native code**.

- MSIL includes
  - instructions for **loading**, **storing**, **initializing**, and **calling methods** on objects, as well as
  - instructions for **arithmetic and logical operations**, **control flow**, **direct memory access**, exception handling, and other operations.

MSIL and JIT

Compile Time

$$\begin{tikzcd}\boxed{\text{C\# code}} \arrow[d, "\text{Compile}"'] \\ \boxed{\text{MSIL code}}\end{tikzcd}$$

- Language neutral code
- Assembly
  a) Exe
  b) dll

Run Time

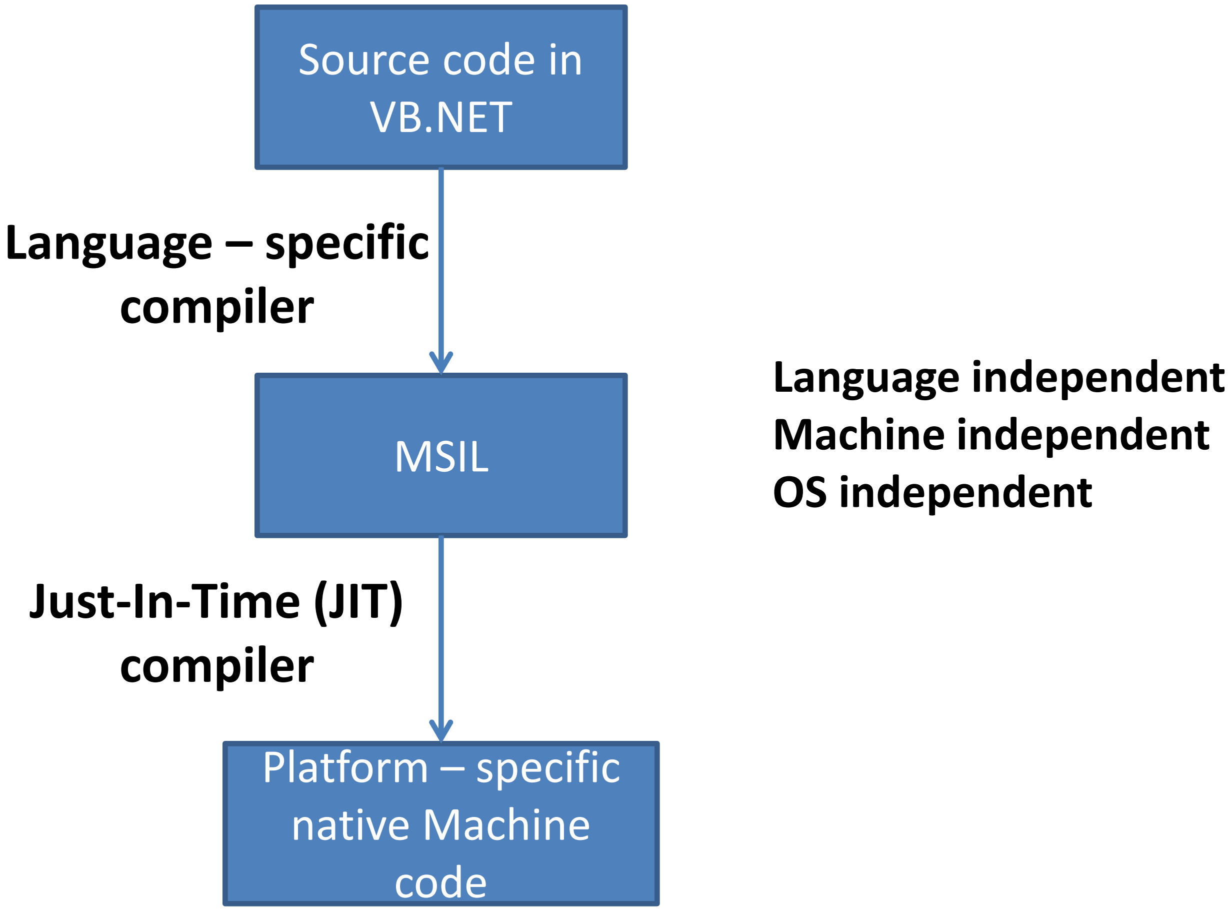
$$\begin{tikzcd}\boxed{\text{MSIL code}} \arrow[d, "\text{JIT Compilation}"'] \\ \boxed{\text{Native code}}\end{tikzcd}$$

- Operating System specific
- Machine Architecture Specific

This is time OS executes application

Microsoft Intermediate Language (MSIL)

- MSIL makes cross-language integration possible in support with **meta data** and **Common Type System**.


Source code in VB.NET
Language – specific compiler
MSIL
Language independent
Machine independent
OS independent
Just-In-Time (JIT) compiler
Platform – specific native Machine code

Microsoft Intermediate Language (MSIL)

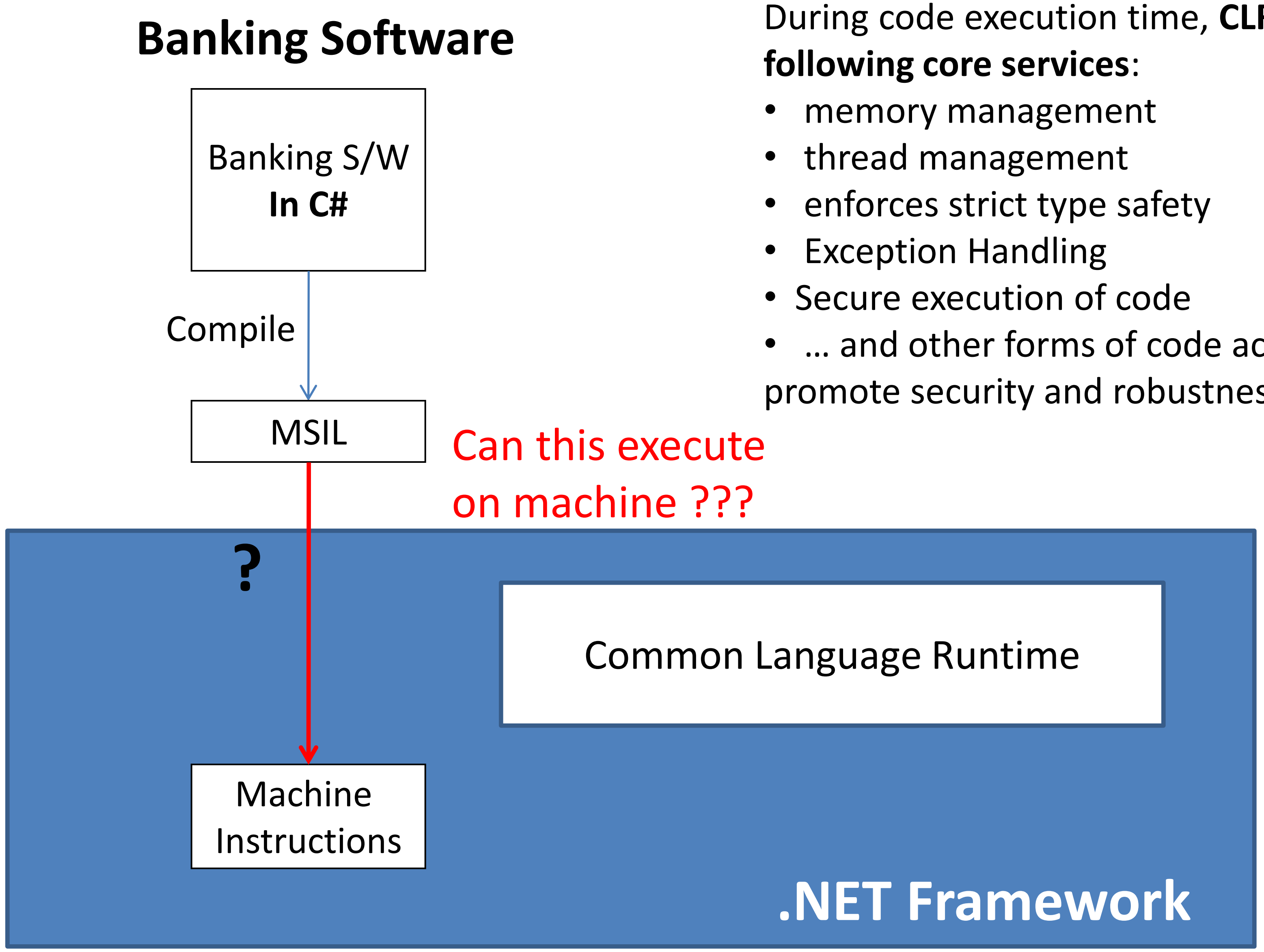
- When a compiler produces MSIL, **it also produces metadata.**

- The MSIL and metadata are contained in **a portable executable** (PE file).

.NET Framework - Design features

- Interoperability
- Common Language Runtime Engine
- Base Class Library
- Simplified Deployment
- Security
- Portability

CLR - Framework Component



During code execution time, **CLR provides following core services**:

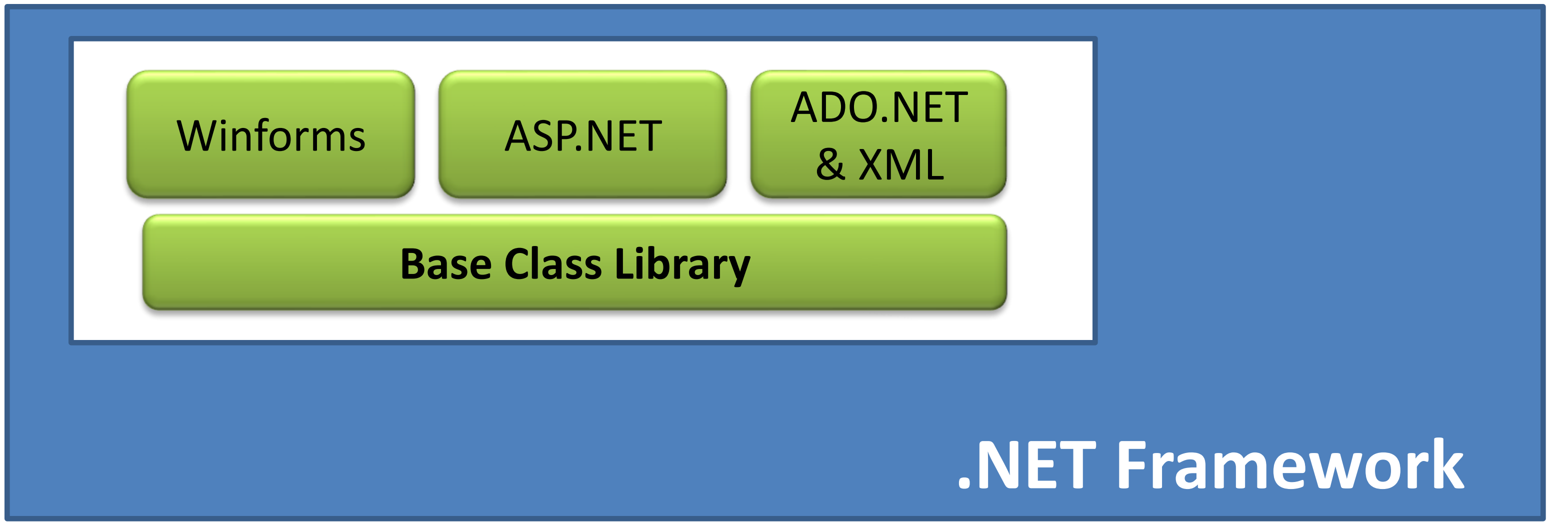
- memory management
- thread management
- enforces strict type safety
- Exception Handling
- Secure execution of code
- … and other forms of code accuracy that promote security and robustness

Base Class Library - Framework Component

Banking S/W
In C#

How the UI and Coding is done for application??

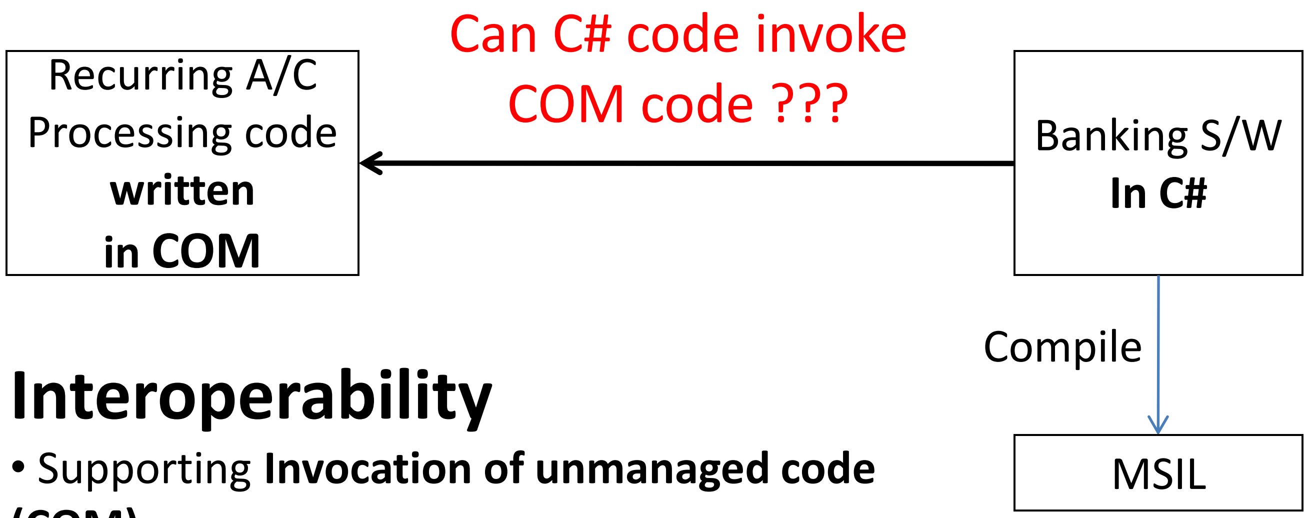
- By using different UI Controls i.e. **Web Forms / Windows Forms**
- Accessing Data storage like **DB / XMLs**



Base Class Library - Framework Component

- Part of the Framework Class Library (FCL)
- Provides **classes, interfaces of reusable types** that encapsulate a number of common functions,
  - Like **file reading** and writing, graphic rendering, database interaction,
  - **XML document manipulation**

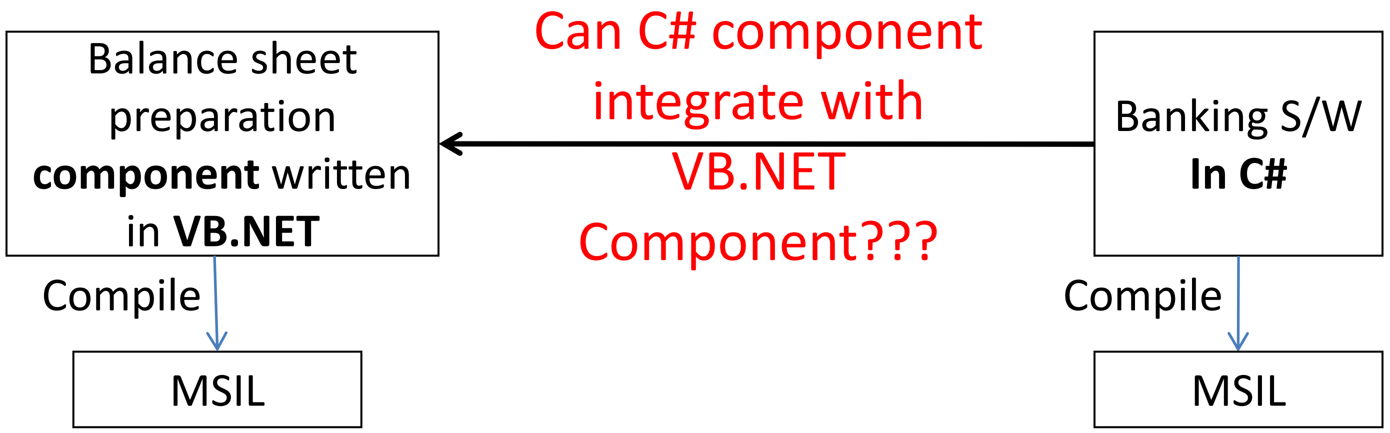
Interoperability - Feature


Can C# code invoke COM code ???
Recurring A/C Processing code **written in COM**
Banking S/W **In C#**
Compile
MSIL

Interoperability

- Supporting **Invocation of unmanaged code (COM)**
- Framework provides a **way to access COM functionality through wrapper i.e.** InteropService
- **Execution of unmanaged** code will be done **outside the .NET Environment**

Language Independence - Feature


Balance sheet preparation **component** written in **VB.NET**
Can C# component integrate with VB.NET Component???
Banking S/W **In C#**
Compile
MSIL
Compile
MSIL

Language Independence

- Compile code of VB.NET generate MSIL
- Similarly compiled code of C# generates MSIL
- technically MSIL is **Language Neutral** code
- **Execution / invocation** of one other is **no different**
- This is possible with only **DOTNET compliant languages**

What Components share when they integrate?

- **Objects , Data**

How does this work internally ???

Language Independence - internals

Common Type System (CTS)

- It **describes set of data types** that can be used in different .Net languages in common.
- It **ensures** that **objects** written in different .Net languages **can interact with each other**.
- **CTS is the first pre-requisite to allow languages to interact with each other**

- **For Communicating between programs** written in any .NET complaint language, the **types have to be compatible on the basic level.**
  i.e. short in VB.net == Int16 in C#

  i.e. C# has **int** == **Integer** in VB.Net == **Int32 from CTS.**

Common Type System

Common Type System

The common type system defines **how types are declared, used, and managed** in the common language runtime

Functions of CTS

- It enables cross-language integration, type safety, and high-performance code execution.

- It **provides an object-oriented model** for implementation of many programming languages.

- It **defines rules that every language must follow** which runs under .NET framework. It **ensures that objects** written in different .NET Languages like C#, VB.NET, F# etc. **can interact with each other.**

Common Type System

- Types in .NET
- Type Definitions
- Type Members
- Characteristics of Type Members

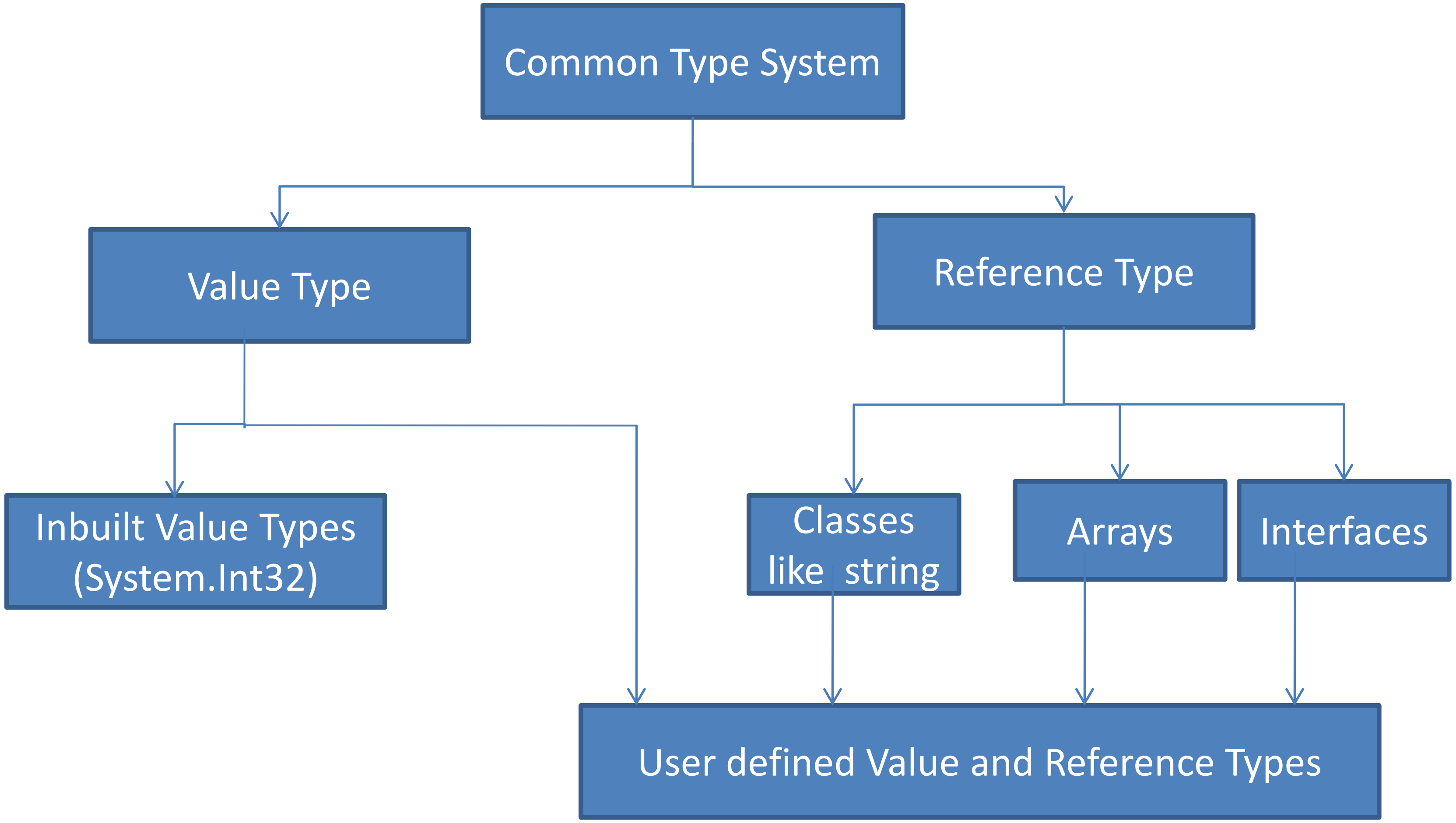
Types in .NET

- Any Type can be divided into two general categories:
  - Reference type and Value type

- **Value types** are data types whose objects are represented by the object's actual value.

- **Reference types** are data types whose objects are represented by a **reference to the object's actual value.**

Types in .NET


Common Type System
Value Type
Reference Type
Inbuilt Value Types (System.Int32)
Classes like string
Arrays
Interfaces
User defined Value and Reference Types

Types in .NET

Common type system in .NET supports the following five categories of types

- Classes
- Structures
- Enumerations
- Interfaces
- Delegates

Types in .NET

- **Classes** - reference type, implicitly derived from System.Object.

- Rules
  - A class can inherit from **only one base class** in addition to System.Object.
  - All classes must have **at least one constructor**.
  - Each language that supports the runtime provides a way to indicate that **a class or class member has one or more of these characteristics.**
    - **sealed / implements / abstract / inherits / exported or not exported**

Types in .NET

Structures

- **value type** that derives implicitly from System.ValueType
- **very useful** for representing values whose **memory requirements are small**, and
- for **passing values as by-value** parameters to methods that have **strongly typed parameters**.
- all primitive types i.e. (Boolean, Byte, Char, DateTime, Decimal, Double, Int16, Int32, Int64, SByte, Single, UInt16, UInt32, and UInt64)

Structures – CTS Rules

- For each **value type**, the common language runtime supplies a **corresponding boxed type**, which **is a class that has the same state and behaviour as the value type**. e.g. int ➔ Integer

- When you **define a value type**, you are defining both the **boxed** and **the unboxed type**.

Types in .NET

- **Enumerations**
  - value type that inherits directly from System.Enum
  - has **name and value** (must be signed or unsigned integer such as Byte, Int32, or UInt64))

E.g enum shapes
{ circle, square}

Enumerations– Restrictions

- They cannot define their own methods.
- They cannot implement interfaces.
- They cannot define properties or events.
- They cannot be generic, unless they are generic only because they are nested within a generic type.

Types in .NET

Interface

- An interface defines a contract that specifies a "**can do**" relationship or a "**has a**" relationship.
- Interfaces are often used to implement functionality, such as **comparing** and **sorting** (the IComparable and IComparable<T> interfaces), **testing for equality** (the IEquatable<T> interface),

Restrictions apply to interfaces:

- An interface can be declared with any accessibility, but **interface members must all have public accessibility**.
- Interfaces **cannot define constructors**.
- Interfaces **cannot define fields**.
- Interfaces can define **only instance members. They cannot define static members**.

Types in .NET

Interface Rules

- Each language must provide rules for mapping an implementation to the interface that requires the member, because more than one interface can declare a member with the same signature, and these members can have separate implementations.

Types in .NET

Delegates

- reference types similar to that of function pointers in C++
- Single cast / multicast

- In many cases, such as with **callback methods**, a delegate **represents only one method**, and the only actions you have to take are creating the delegate and invoking it.

CTS - Type Definitions

A type definition includes the following:

- Any attributes (metadata) defined on the type.
- The type's accessibility (visibility) - public / assembly
- The type's name
  - naming convention, case sensitivity etc
- The type's base type and Interfaces
  - The common type system does not allow types to inherit from more than one base type.
- Any interfaces implemented by the type.
- Definitions for each of the type's members.

CTS - Type Members

Type members specifies the behaviour and state of a type. Type members include the following

- Fields
- Properties
- Methods
- Constructors
- Events
- Nested types

CTS - Characteristics of Type Members

- The common type system allows **type members** to have a **variety of characteristics**.
- Languages can support whichever appropriate from these characteristics

Characteristic	Can apply to	Description
abstract	Methods, properties, and events	The type **does not supply the method's implementation**
Private / public / family / assembly	All	Defines the **accessibility** of the member
final	Methods, properties, and events	The virtual method **cannot be overridden in a derived type.**
initialize-only	Fields	The value can only be initialized, and **cannot be written after initialization.**

CTS - Characteristics of Type Members

Characteristic	Can apply to	Description
literal	Fields	The value assigned to the field is a fixed value, known at compile time . e.g. **constants**
static	Fields, methods, properties, and events	
virtual	Methods, properties, and events	
Newslot / override	All	newslot Hides inherited members that have the same signature. override Replaces the definition of an inherited virtual method.

Language Independence - internals

Common Language Specification (CLS)

- It is a **sub set of CTS**
- it **specifies a set of rules** that needs to be **followed or satisfied** by all language compilers targeting CLR.
- It helps in **cross language inheritance** and **cross language debugging.**

Common language specification Rules:

- It **describes** the **minimal and complete set of features** to produce code that can be hosted by CLR.
- It **ensures that products of compilers will work properly** in .NET environment.

Sample Rules:

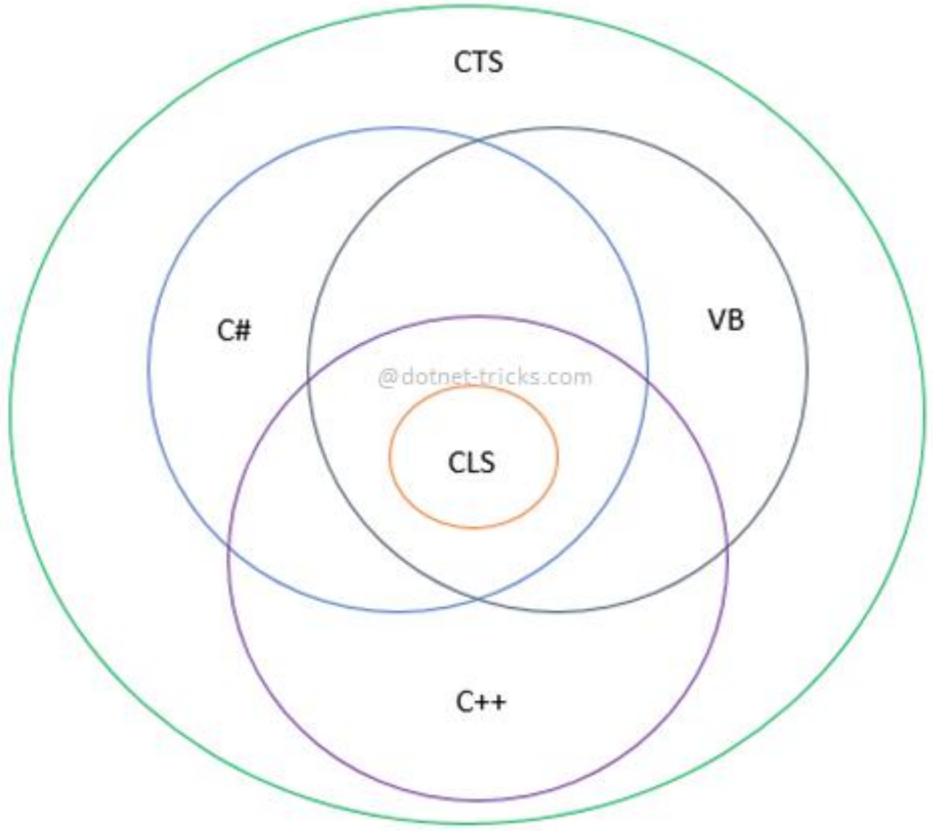
1. Representation of text strings
2. Internal representation of enumerations
3. Definition of static members

Common Language Specification

CLS Rules

- **Arrays** shall have elements with a CLS-compliant type, and all dimensions of the array shall have **lower bounds of zero**.
- An object **constructor** shall call some instance constructor of its base class before any access occurs to inherited instance data
- An object constructor shall not be called except as part of the creation of an object, and an **object shall not be initialized twice.**

Relationship Between CTS and CLS



CTS and CLS

Language Independence - internals

- To summarize,
  - So while working with multiple language components, because of these CLS and CTS, .NET Framework **supports** the **exchange of types and object instances** between libraries and applications written using any conforming .NET language.

.NET Compatible Languages

C#, **C++/CLI**:, Cobra, Eiffel, F#, F*, Fantom, IronPython, Jscript.NET., Xsharp… many more

- Third-parties are building
  - APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk…

COMMON TERMINOLOGY

Managed Code

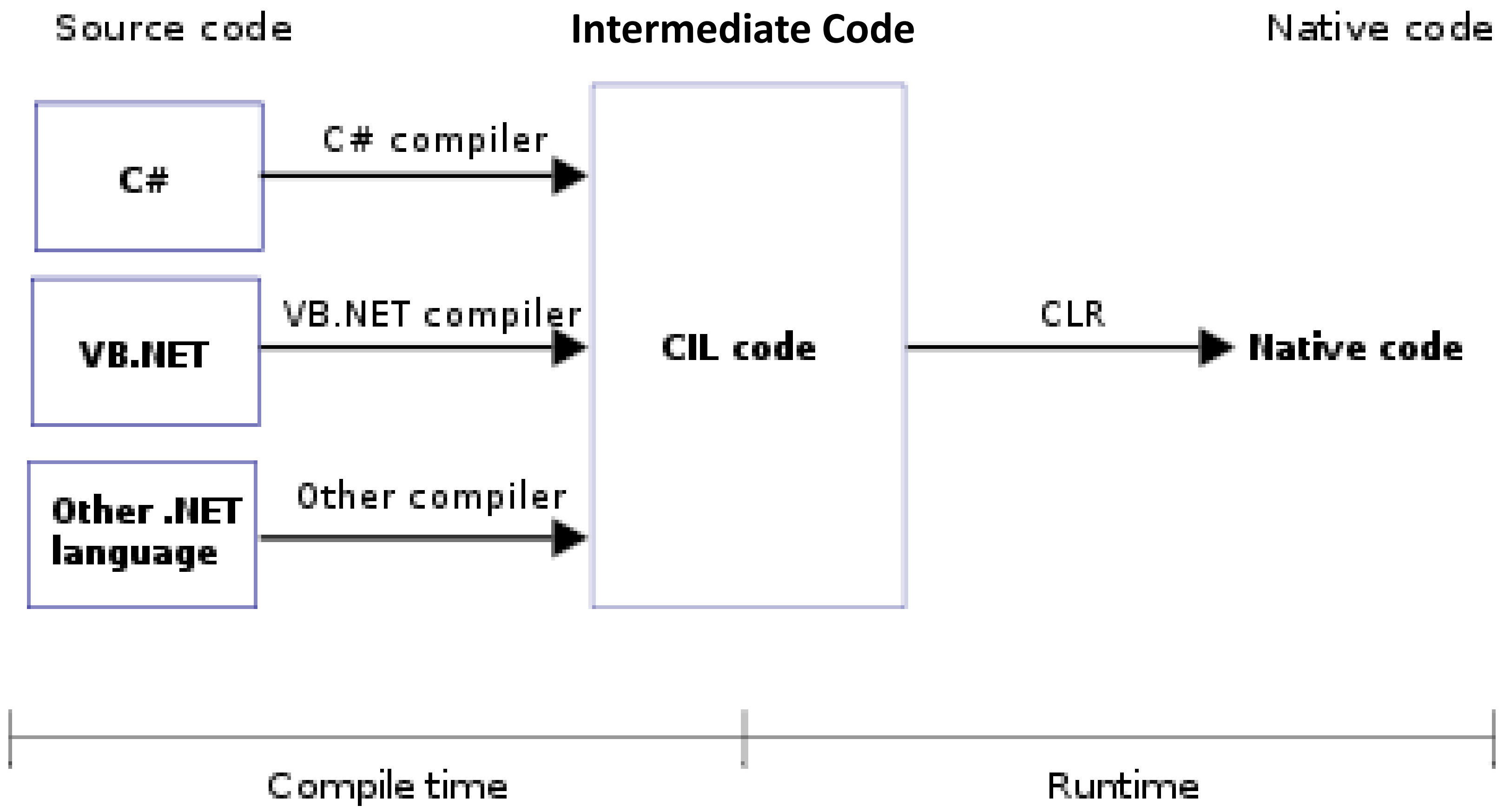
- NET supports two kind of coding :
  - **Managed Code** and **Unmanaged Code**.
- **The code, which is developed in .NET compliant languages, is known as managed code.**
- This code is **directly executed by CLR** with help of managed code execution.
- Any language that is written targeting .NET Framework is managed code.
- Managed code uses CLR which in turns looks after your applications by
  - managing memory,
  - handling security,
  - allowing cross - language debugging, and so on.

Unmanaged code

- **The code,** which is **developed outside** .NET Framework is known as **unmanaged code**.
- Applications that **do not run under the control** of the **CLR** are said to be unmanaged.
- **Unmanaged Code**
  - Certain languages such as **C++** can be used to write such applications, which, for example, **to access low - level functions** of the operating system.
  - **Backward compatibility** with code of VB, COM are examples of unmanaged code.

COMMON LANGUAGE RUNTIME

CLR Execution

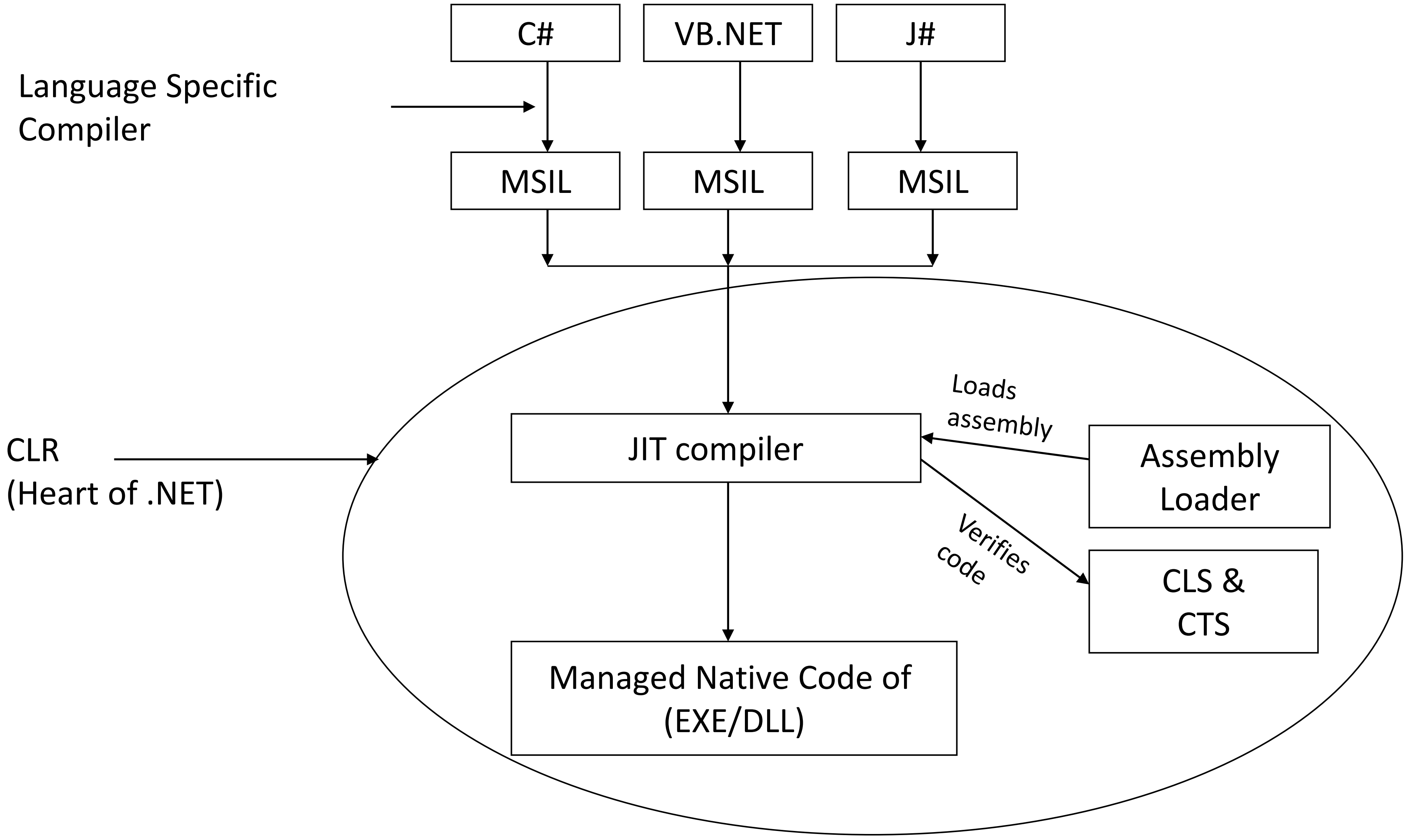


During just-in-time (JIT) compilation,

a) the CLR compiles the intermediate language code known as CIL (Common Intermediate Language) into the machine instructions

b) These machine instructions are executed by the CPU.

Working of CLR



Common Language Runtime (CLR)

- CLR works like a **virtual machine** in executing.NET applications (from various languages).
- All .NET languages **must obey the rules** and **standards** that are imposed by CLR. For example:
  - Object declaration, creation and use
  - Data types, language libraries
  - Error and exception handling
- CLR implements the **Common Language Specification** (CLS), which is the set of specifications that **language and library designers need to follow**.
  - This essentially **ensures the inter working** between the different .NET languages.

Other CLR Features

- Code Verification
  - While converting IL code to Native, checks are made to ensure
    - **Operation** being performed is **safe** or not
    - Whether the **instructions are referring** to **Valid Address**
- Code Access Security
  - CLR keeps track on Assemblies like
    - From where they are getting loaded
    - What security constraints should be placed on them
- Garbage Collection
  - Once an objects lifetime is over, its memory is reclaimed through garbage collector
- Profiling and Debugging
- Thread Management
- Complete Code Execution providing Type Safety.
- Execution of MSIL Code

Framework Class Library (FCL)

- The FCL is a **collection of thousands of reusable** **classes** (within hundreds of namespaces), **interfaces** and **value types**.

- Enables us to accomplish a range of **common programming tasks:**

  - string management
  - data collection
  - database connectivity
  - file access

- Also supports a variety of **specialized development scenarios**:

  - Console applications
  - ASP.NET applications
  - Web services

Framework Class Library (FCL)

- FCL provides support for user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.

- **Classes are divided into namespaces** grouping similar classes.

- For organization, **each class belongs to only one namespace**.

- Most classes are lumped into a namespace called **System**
  - System.Data: DB access
  - System.XML: reading/writing XML
  - System.Windows.Forms: Forms manipulation
  - System.Net: network communication.
- All framework **classes are inherited from root class called object**.

Windows Forms

- Framework for Building Rich Clients
  - RAD (Rapid Application Development)
  - Rich set of controls
  - Data ware
  - ActiveX® Support
  - Licensing
  - Accessibility
  - Printing support
  - Unicode support
  - UI inheritance

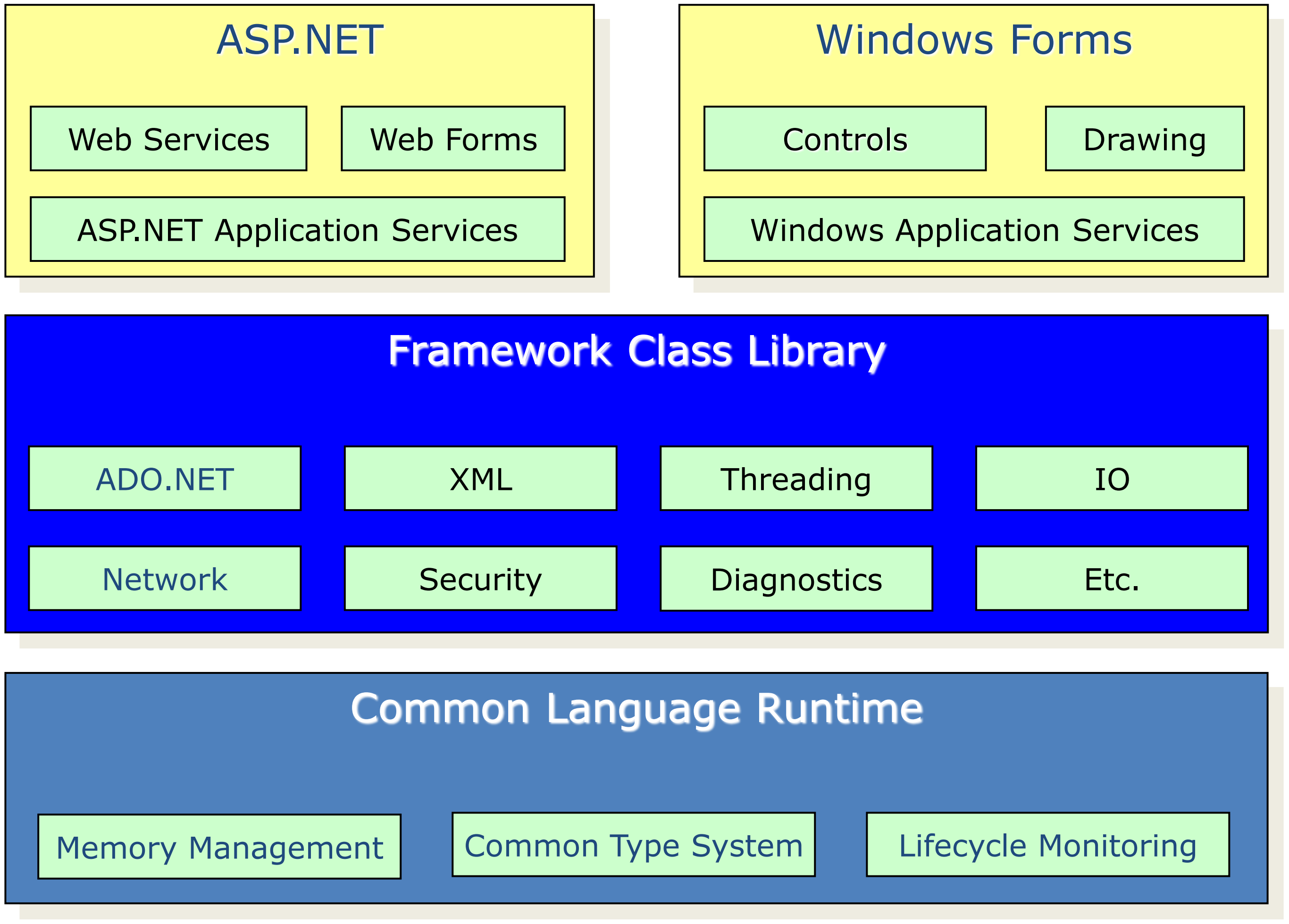
ASP.NET

- A **web application framework** developed and marketed by Microsoft to allow programmers
  - to build dynamic web sites, web applications and
  - web services.

- ASP.NET uses .NET languages **to generate HTML pages.**

- **HTML page is targeted** to the capabilities of the **requesting browser.**

Summary


ASP.NET
Web Services
Web Forms
ASP.NET Application Services
Windows Forms
Controls
Drawing
Windows Application Services
Framework Class Library
ADO.NET
XML
Threading
IO
Network
Security
Diagnostics
Etc.
Common Language Runtime
Memory Management
Common Type System
Lifecycle Monitoring

Memory Management

Memory Management

- In C, C++:
  - developer was responsible for memory allocation and release
  - memory is managed through the Operating System directly.
- CLR basic premise (principle) :
  - **System (framework) should be responsible for memory management**
  - Programmers **should not worry**.
- Approach can be similar to VBs memory management
  - Usage of **reference count**
  - **Disadvantage** of incrementing and decrementing counts
- CLR, **focus on objects lifetime.**

Memory Management

- .NET CLR requires that all resources be **allocated** from the **managed heap**.
- With the help of Garbage Collector, **objects are automatically freed** when they are **no longer needed** by the application.

Memory Management

- In .NET, **CLR manages memory** through the **use of Managed Heaps**.
- During application execution, CLR
  - **creates a process** in the name of application executable.
  - System **allocates** a segment of **memory to store and manage objects**. This memory is called the **managed heap**.
- There is a **managed heap** for **each managed process.**
- **All threads** in the process **allocate memory** for objects **on the same heap**.

Memory Management

Four sections of memory (heaps) are created to be used for storage

- The **Code Heap** stores the **actual native code** instructions
  - after they have been Just in Time Compiled (**JITed**).

- The **Small Object Heap (SOH)** stores allocated **objects** that are **less than 85K** in size .

- The **Large Object Heap (LOH)** stores allocated objects ***greater* than 85K**.

- Finally, there’s the **Process Heap.**

Memory Management

Code Heap	Small Object Heap (< 85K)
Large Object Heap (> 85K)	Process Heap

Memory Management

- For each process .NET maintains a **stack data structure.**
- If process creates multiple threads, **separate stacks** will be created for **each thread**.

Storage Allocation Mechanisms

Objects lifetime corresponds to storage allocation mechanism.

- Storage Allocation mechanisms
  - Static
  - Stack
  - Heap

Static objects are given absolute address that is retained throughout program's execution.

Stack objects are allocated and deallocated in last in, first out order (e.g. subroutine calls and returns are allocated in last in first-out order).

Heap objects may be allocated and deallocated at arbitrary times. They require more **general (and expensive) storage mgmt** algorithm.

Stack – based allocation

- Why a stack?
  - allocate space for recursive routines.
  - reuse space

- Central stack for
  - Function execution
  - Storing static and dynamic links
- Contents of a stack frame (refer next Figure)

  arguments and returns

  local variables

  temporaries

  bookkeeping (saved registers, line number static link, etc.)

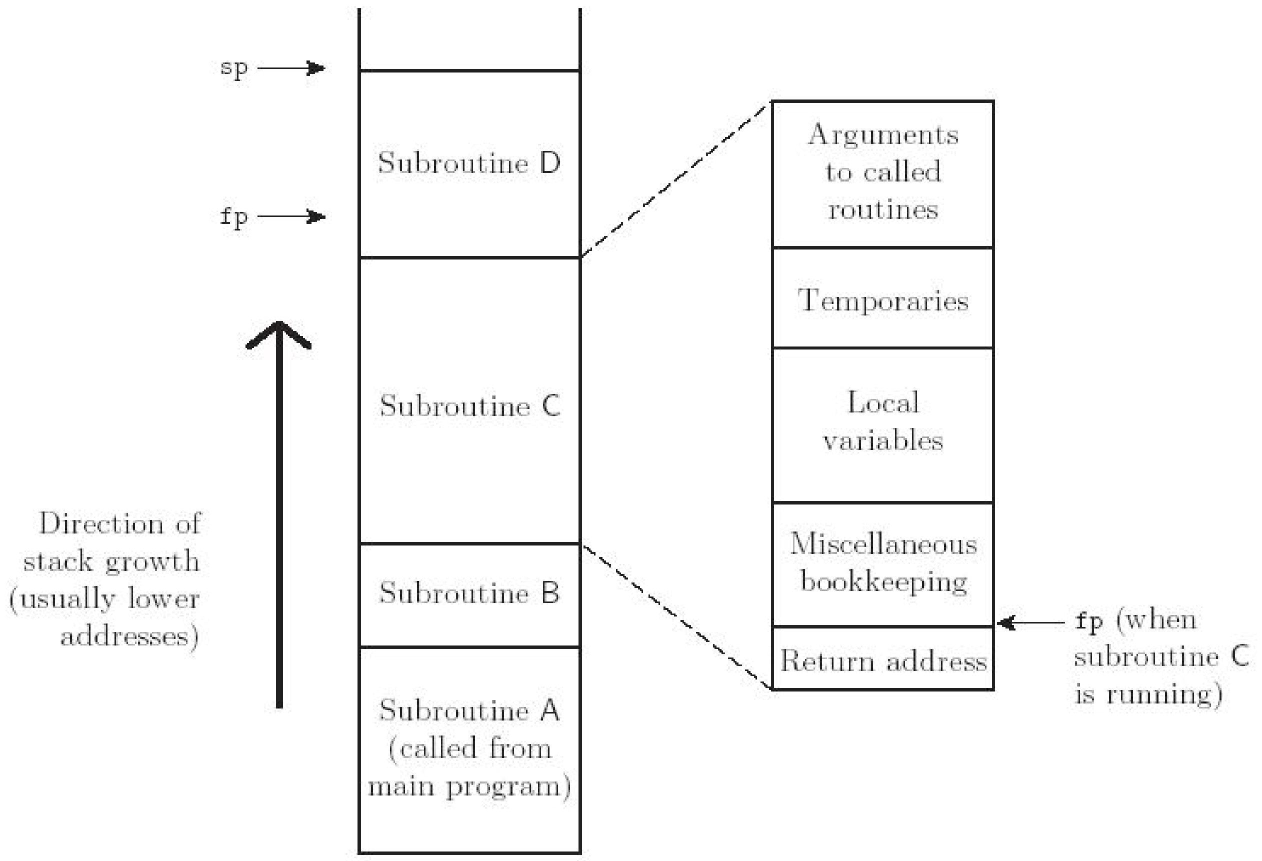

sp
fp
Subroutine D
Subroutine C
Subroutine B
Subroutine A (called from main program)
Direction of stack growth (usually lower addresses)
Arguments to called routines
Temporaries
Local variables
Miscellaneous bookkeeping
Return address
fp (when subroutine C is running)

Figure 3.2: **Stack-based allocation of space for subroutines.** We assume here that subroutine A has been called by the main program, and that it then calls subroutine B. Subroutine B subsequently calls C, which in turn calls D. At any given time, the stack pointer (`sp`) register points to the first unused location on the stack (or the last used location on some machines), and the frame pointer (`fp`) register points to a known location within the frame (activation record) of the current subroutine. The relative order of fields within a frame may vary from machine to machine and compiler to compiler.

Memory Management – Stack data structure

- Stack helps **to track the state of an execution thread** and **all the method calls made**.

- When a method is called, .NET creates a container (**a stack frame**) that contains all of the data necessary to complete the call,
  - including **parameters**,
  - **locally** declared **variables** and
  - **return address**

Memory Management – Stack data structure

- **For each** method **call**, **new stack frame will be created** and placed on top of stack.

- When a method completes,
  - its' **stack frame** i.e. container **is removed** from the top of the stack and
  - the **execution returns to the next line** of code within the calling method (with its own stack frame).

- The **frame at the top** of the stack is always the one used by the **current executing method**.

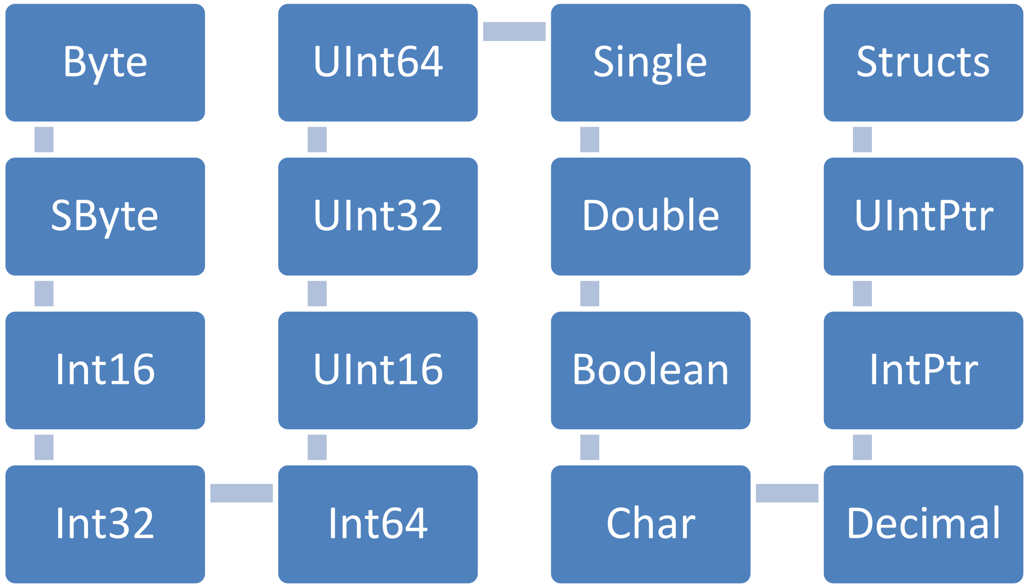
Memory Management – Stack data structure

- Using this simple mechanism, **the state of every method** is **preserved** in between calls to other methods, and they are **all kept separate from each other.**

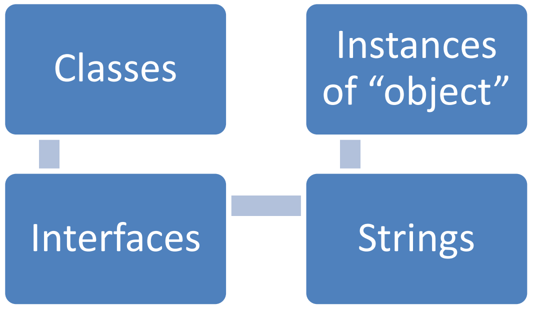
- The **stack** can **store variables** that are the **primitive data types** defined by .NET. These include the following types:-

Memory Management – Stack data structure

Primitive data types



Reference types :



Memory Management - Stack data structure

Handling of reference types

- Instance of a **reference type** is created (usually involving the **new** keyword),
  - Are stored **on the heap** (the **SOH** or **LOH**, depending on their size).
  - Its address reference is **stored on stack**.

- Whenever **process makes a memory request**
  - Request is transferred to CLR
  - Based on the requested memory size, memory is allocated from one of the heap.

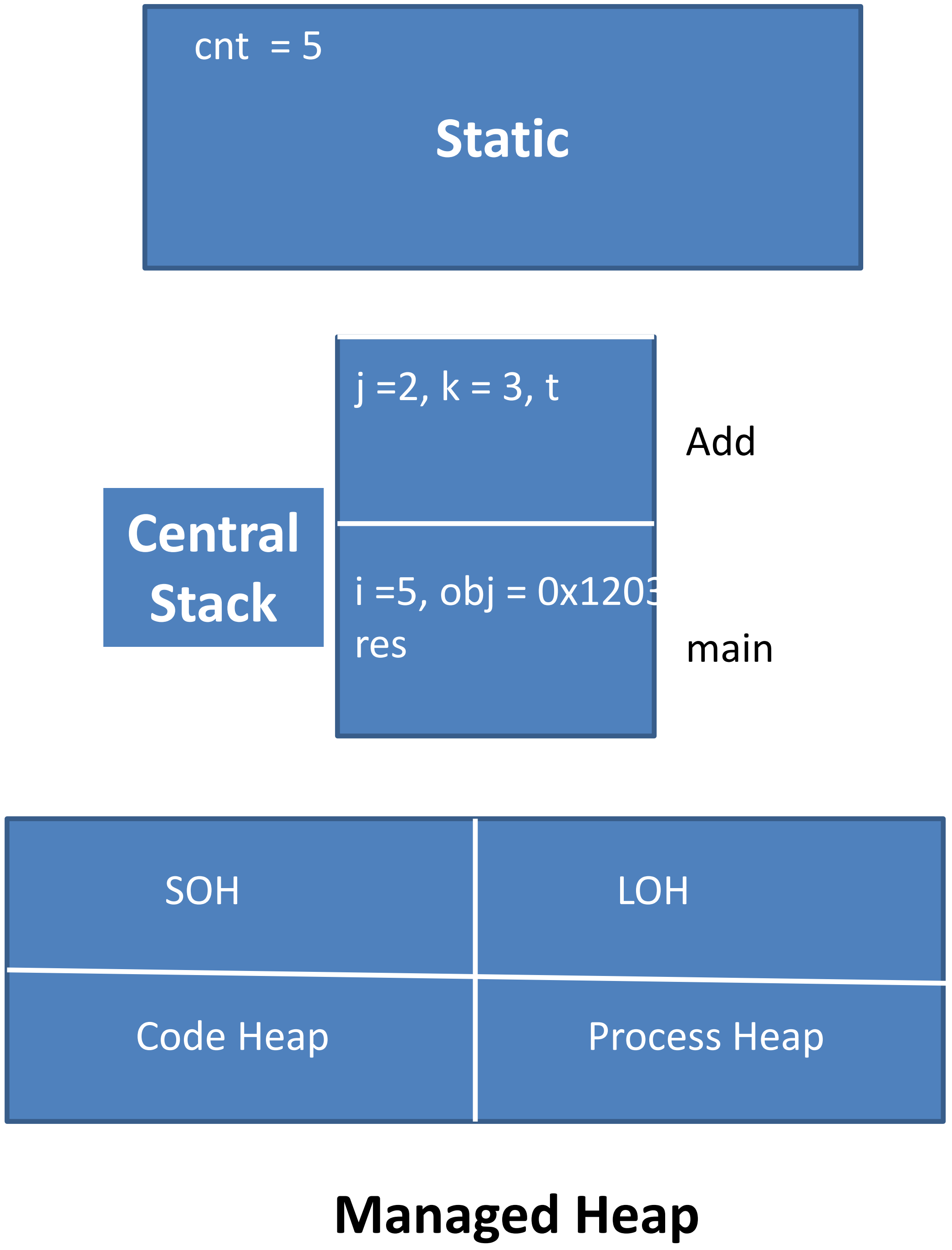
```
main()
{
int i = 5;
static int cnt = 5;

student obj = new student();

int res = Add(2,3)
}


int Add(int j, k)
{
 int t = j +k;
 return t;

}
```



Heap – Based allocation

- Heap is a **region of memory** in which sub blocks can be allocated and de-allocated at arbitrary times.

- Who gets allocated on Heap?
  - Dynamically allocated data e.g. **new, malloc**
  - Objects like fully general character strings, lists, sets (whose size may change upon assignment)

- Issues with Heap based allocation ??

  Due to random allocation and de-allocation
  - Memory may become **fragmented**
  - **Need for garbage collection**

Heap Management

- Internal fragmentation:

  Occurs when storage mgmt allocates block larger than required.
- External fragmentation:

  Occurs when allocate blocks are scattered through heap in such way that free space is composed of multiple blocks.
- First fit

  Algorithm for searching first block which is large enough to cater memory request.
- Best Fit

  Algorithm which scan whole list to find smallest block which will be large enough to cater memory request.

Heap for dynamic allocation

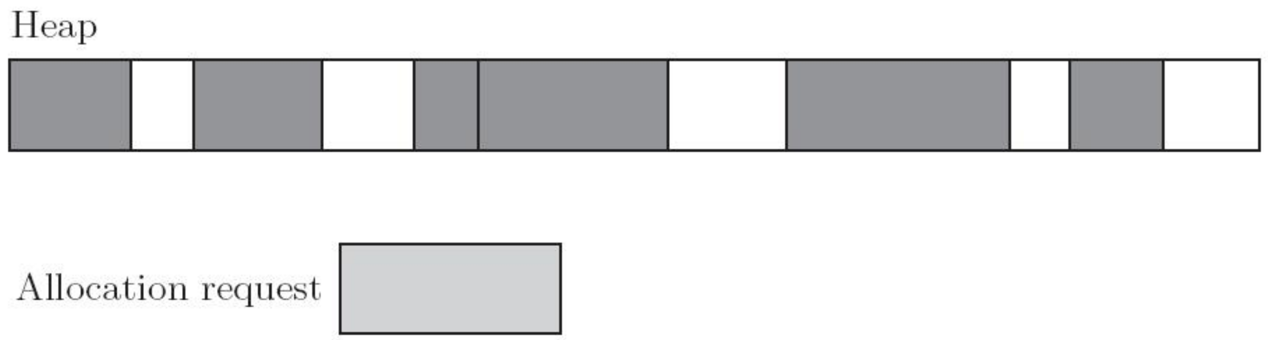


Figure 3.3: **External fragmentation.** The shaded blocks are in use; the clear blocks are free. While there is more than enough total free space remaining to satisfy an allocation request of the illustrated size, no single remaining block is large enough.

Speed and Space are the main concerns

.NET Heap - Memory Management

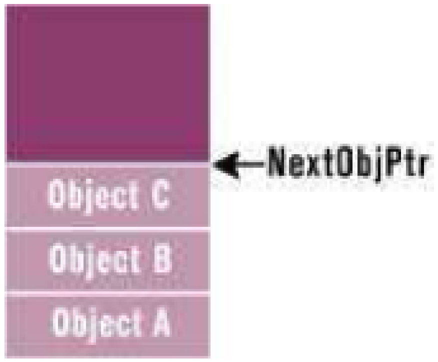
- Heap maintains a pointer, say **NextObjPtr**.
- This **pointer indicates where the next object** is to be allocated within the heap.
- **Initially**, the NextObjPtr is **set to the base address** of the managed heap.
- An **application creates an object** using the **new** operator.
- This **operator** first **makes sure** that **space is available**
  - i.e. the bytes required by the new object fit in the reserved region.

Memory Management

- **If** the **object fits**, this **object's constructor is called**, and the new operator returns the address of the object.

- At this point, **NextObjPtr is incremented past the object** so that it **points to where the next object** will be placed in the heap.

- **Figure shows a managed heap consisting of three objects: A, B, and C.**
- The next object to be allocated will be placed where NextObjPtr points (immediately after object C).



Garbage Collection

- The garbage collector checks to see if there are any objects in the heap that are **no longer being used** by the application.

- **If such objects exist**, then the **memory** used by these objects **can be reclaimed**.

- (If **no more memory is available** for the heap, then the **new** operator throws an OutOfMemoryException.)

Garbage Collection

- Every application has a set of roots.
- **Roots identify storage locations**, which refer to objects on the managed heap or to objects that are set to null.
- An application's roots include global and static object pointers, local variables and reference object parameters on a thread's stack, and CPU registers.

Garbage Collection

- The garbage collector has access to the list of active roots that the just-in-time (JIT) compiler and the runtime maintain.

- Using this list,
  - it **examines an application's roots**,
  - and in the process **creates a graph** that contains **all the objects** that are **reachable from the roots**.

Garbage Collection

- Objects **that are not in the graph** are **unreachable** from the application's roots.

- The garbage collector considers **unreachable objects as garbage** and will **release the memory** allocated for them.

- During a collection, the garbage collector **examines the managed heap**, looking for the blocks of address space occupied by unreachable objects.

Garbage Collection

SOH Cleanup - Heap Compaction

- Garbage collection of the Small Object Heap (SOH) involves **compaction**.

- This is because the small object heap **is a contiguous heap** where objects are allocated consecutively on top of each other.

- When compaction occurs,
  - marked objects are copied over the space taken up by unmarked objects,
  - overwriting those objects, removing any gaps, and keeping the heap contiguous; this process is known as **Copy Collection**.

Garbage Collection

- The advantage of this is that heap **fragmentation** (i.e. unusable memory gaps) **is kept to a minimum**.

- **Disadvantage**
  - Compaction **involves copying chunks** of memory around, which **requires CPU cycles** and so, depending on frequency, can **cause performance problems**.

- What you **gain in efficient allocation** you could **lose in compactions costs.**

Garbage Collection

LOH Sweeping - Free Space Tracking

- The Large Object Heap (LOH) **isn't compacted**, and this is simply because of **the time it would take to copy large** objects over the top of unused ones.

- Instead, the **LOH keeps**
  - **track of free and used space**, and
  - attempts to **allocate new objects** into the most **appropriately-sized free slots** left behind by collected objects.

Garbage Collection

- As a result of this, the LOH is **prone to fragmentation**, wherein memory gaps are left behind that can only be used if large objects (i.e. >85K) of a similar or smaller size to those gaps are subsequently allocated.

Meta Data

Meta Data

- Compilation of code produces **MSIL** and **Meta data**
- It is the **structured description** of the code in an assembly.
- It contains
  - **Description of the assembly** (deployment unit)
    - Identity: Name, Version and culture
    - Dependencies (on other assemblies)
    - Security permission that the assembly requires to run
  - **Description of the Types**
    - classes and interfaces
  - **Custom attributes**
    - Defined by User
    - Defined by Compiler
    - Defined by Framework

Meta Data

- Meta Data is **language independent**.
- Why do we need Meta Data??
  - CLR needs this meta data **to provide compile time and runtime services**. E.g.
    - Loading of Class Files
    - Memory management
    - Debugging
    - Object Browsing
    - MSIL Translation to Native Code

Assembly

What is Assembly?

- Assembly is the logical unit of deployment.
- Can consist of one or more compiled files.
- Assembly can be EXE or DLL

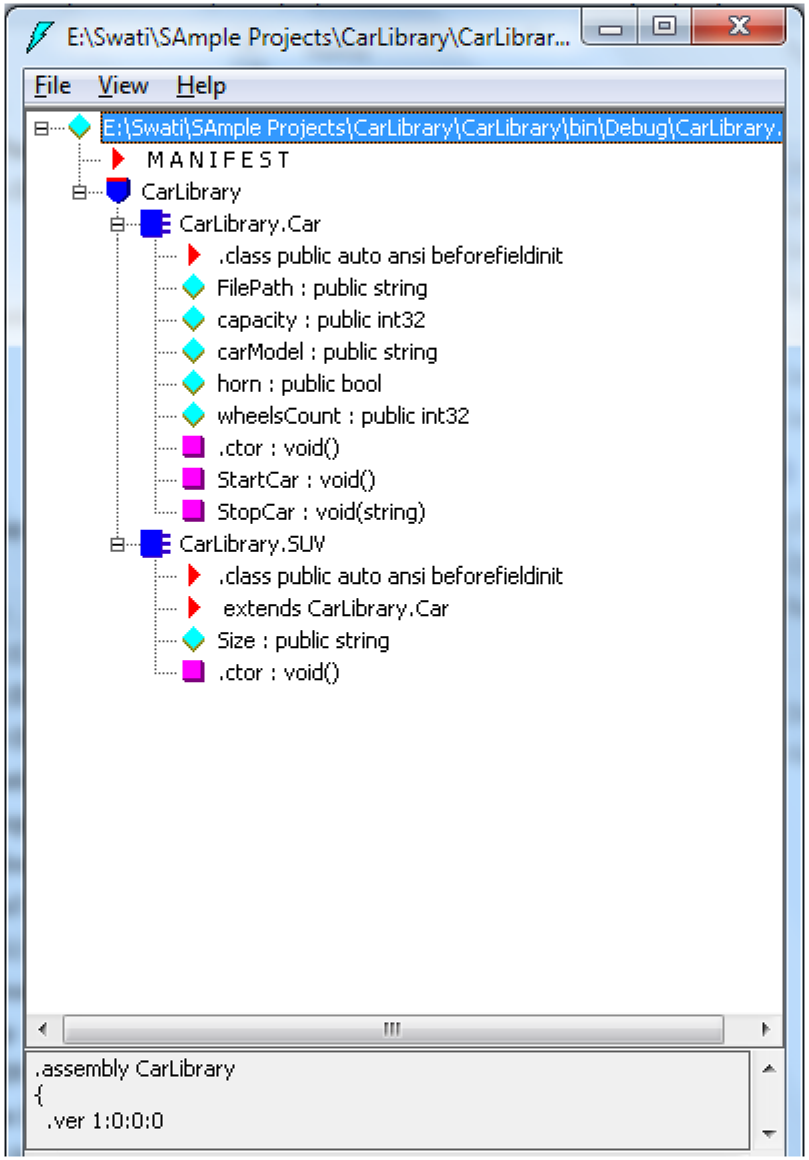
Role of .NET Assemblies

- Assemblies Promote Code Reuse
  - Are the smallest executable unit
- Assemblies Establish a Type Boundary
  - Types in a .Net assembly are considered to be unique
- Assemblies Are Versionable Units
  - Contain exact version info
- Assemblies Are Self-Describing
  - Contain all type info
- Assemblies Are Configurable
  - Can be stored anywhere

Viewing Assembly Contents

- **Ildasm** tool is used for viewing the contents of the Assmebly.
- Ildasm stands for **Intermediate Language disassembler**.

Ildasm demo

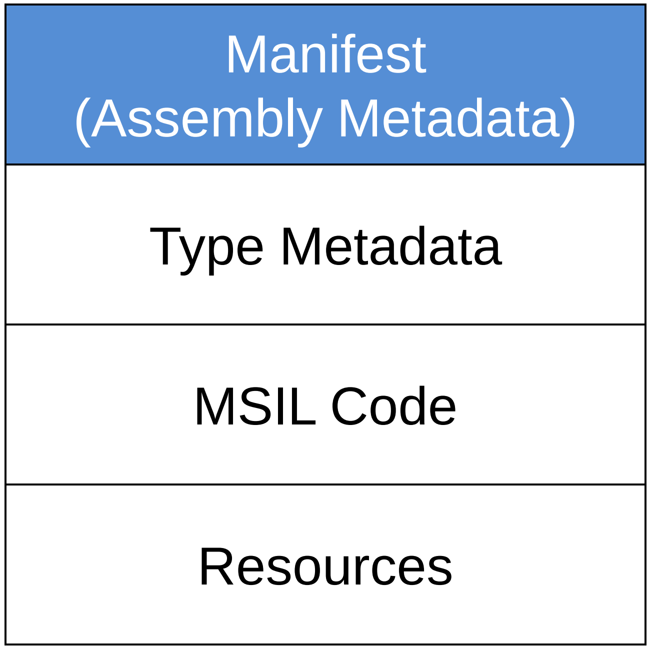


```
E:\Swati\SAmple Projects\CarLibrary\CarLibrar...
File  View  Help
E:\Swati\SAmple Projects\CarLibrary\CarLibrary\bin\Debug\CarLibrary.
  MANIFEST
  CarLibrary
    CarLibrary.Car
      .class public auto ansi beforefieldinit
      FilePath : public string
      capacity : public int32
      carModel : public string
      horn : public bool
      wheelsCount : public int32
      .ctor : void()
      StartCar : void()
      StopCar : void(string)
    CarLibrary.SUV
      .class public auto ansi beforefieldinit
       extends CarLibrary.Car
      Size : public string
      .ctor : void()

.assembly CarLibrary
{
  .ver 1:0:0:0
```

- Launch Visual studio Command prompt.
- Execute Ildasm command.
- Open any assembly in ildasm Viewer.

Assembly Format - Manifest



- Describes the contents of Assembly, hence called Assembly Metadata.

- Describes
  - assembly's version requirements
  - security identity,
  - Module contents
  - References to other assemblies.

- .NET runtime uses manifest in programs assembly to **resolve references to other assemblies.**

Manifests

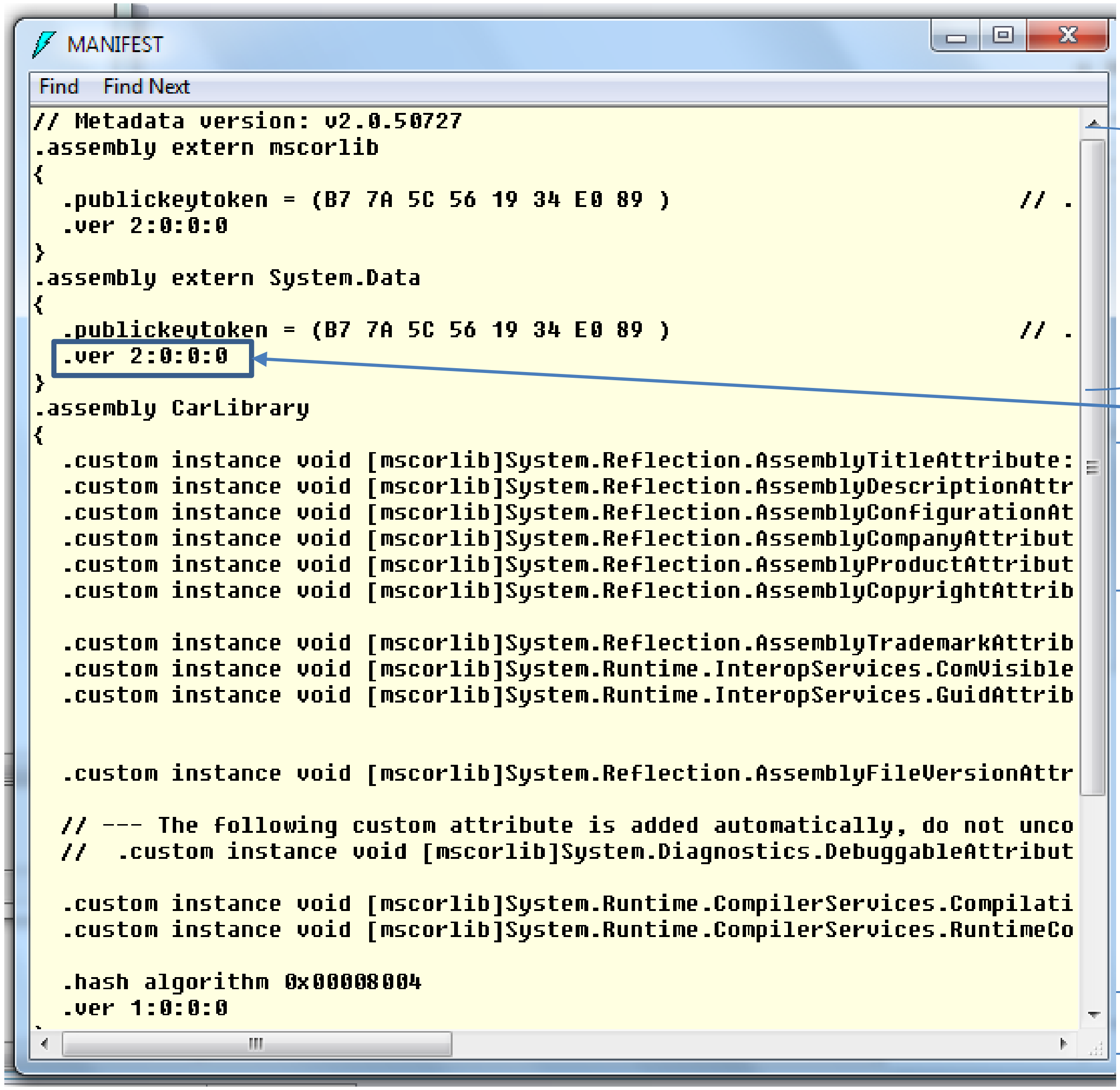
- Manifest describes each file or module within Assembly.

- It describes **External Assemblies** that has been referenced.
  - E.g. Manifest may contain details of System.Data.dll, in case referenced.

- Manifest contains details of
  - External Assemblies
  - Assembly attributes

    AssemblyInfo.cs file is used to set the attributes tin manifest file
  - Assembly Culture
  - Versioning details

Manifests



```
// Metadata version: v2.0.50727
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .
  .ver 2:0:0:0
}
.assembly extern System.Data
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .
  .ver 2:0:0:0
}
.assembly CarLibrary
{
  .custom instance void [mscorlib]System.Reflection.AssemblyTitleAttribute:
  .custom instance void [mscorlib]System.Reflection.AssemblyDescriptionAttr
  .custom instance void [mscorlib]System.Reflection.AssemblyConfigurationAt
  .custom instance void [mscorlib]System.Reflection.AssemblyCompanyAttribut
  .custom instance void [mscorlib]System.Reflection.AssemblyProductAttribut
  .custom instance void [mscorlib]System.Reflection.AssemblyCopyrightAttrib

  .custom instance void [mscorlib]System.Reflection.AssemblyTrademarkAttrib
  .custom instance void [mscorlib]System.Runtime.InteropServices.ComVisible
  .custom instance void [mscorlib]System.Runtime.InteropServices.GuidAttrib


  .custom instance void [mscorlib]System.Reflection.AssemblyFileVersionAttr

  // --- The following custom attribute is added automatically, do not unco
  //  .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribut

  .custom instance void [mscorlib]System.Runtime.CompilerServices.Compilati
  .custom instance void [mscorlib]System.Runtime.CompilerServices.RuntimeCo

  .hash algorithm 0x00008004
  .ver 1:0:0:0
```

External Assembly References

.ver Attribute

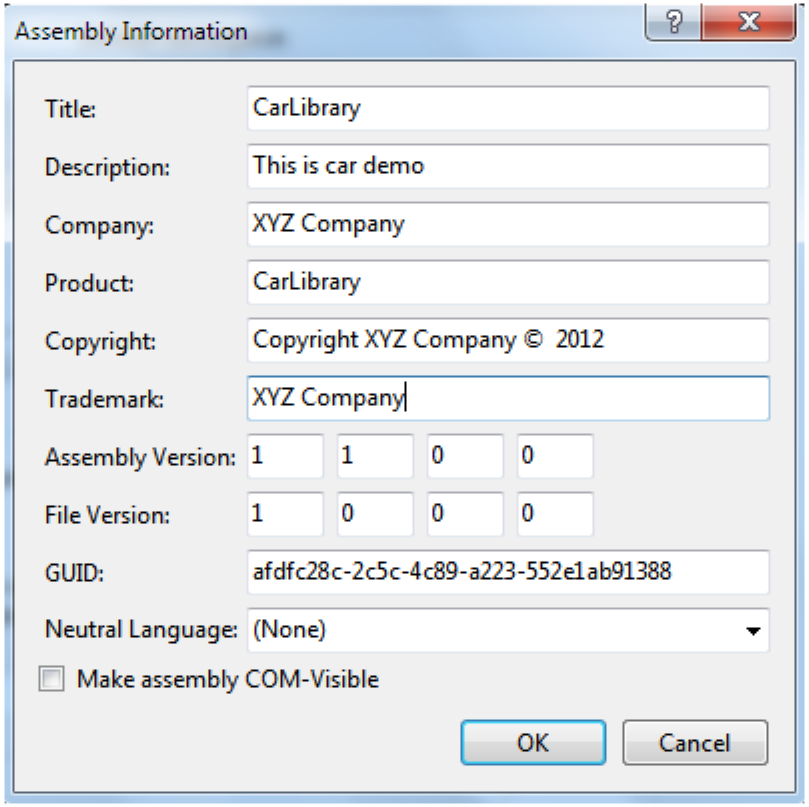
Assembly Attributes

Version Details

Manifests

Assembly Attributes

- Assembly information gets saved in AssemblyInfo.cs
- This information is set as attributes in the Assembly manifest.



Assembly Information

Title:	CarLibrary
Description:	This is car demo
Company:	XYZ Company
Product:	CarLibrary
Copyright:	Copyright XYZ Company © 2012
Trademark:	XYZ Company
Assembly Version:	1 1 0 0
File Version:	1 0 0 0
GUID:	afdfc28c-2c5c-4c89-a223-552e1ab91388
Neutral Language:	(None)

☐ Make assembly COM-Visible

Manifests

- Assembly Culture
  - One can set national language to be used for this assembly.
  - If culture needs to be set then distribute different language versions of component.

- Assembly Version
  - Version for .NET assembly has four parts

    Major Version . Minor Version . Build Number. Revision

    Build Number: is the number incremented every time the Assembly is built.

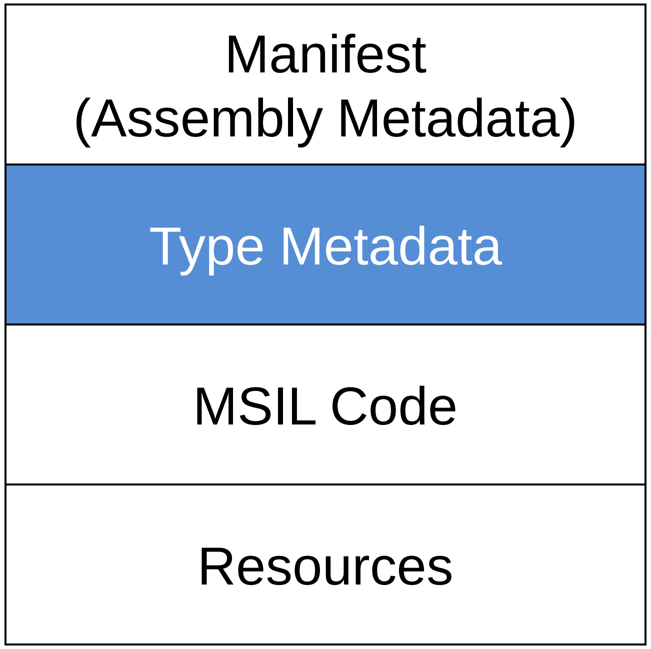
    Revision Number: is the number which is changed while taking patches or hot fixes.

Manifests

- Version Compatibility
  - Before loading the referenced assembly, .NET runtime **cross checks** the version of the referenced assembly against the version mentioned in manifest of the consumer Assembly.
  - In case of incompatibilities, load will fail.

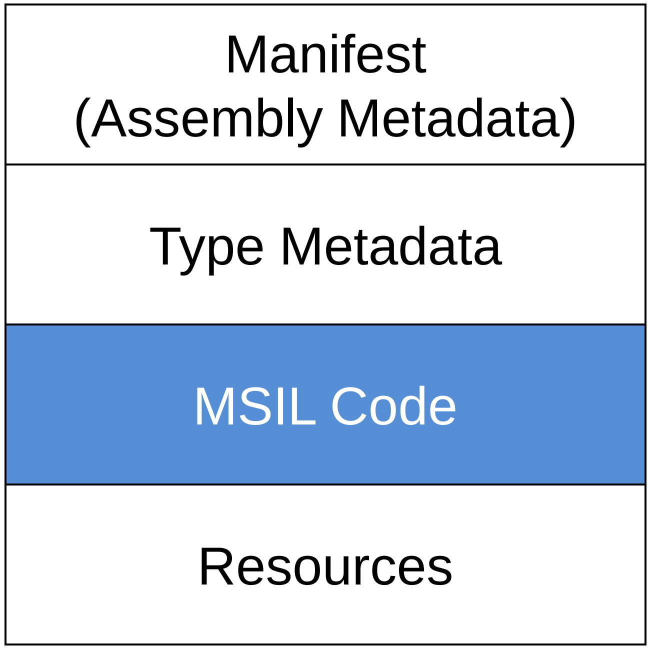
- Multiple versions of same assembly can be installed on the **same machine** and executed, commonly known as **side–by–side execution**.

Assembly Format - Type Metadata



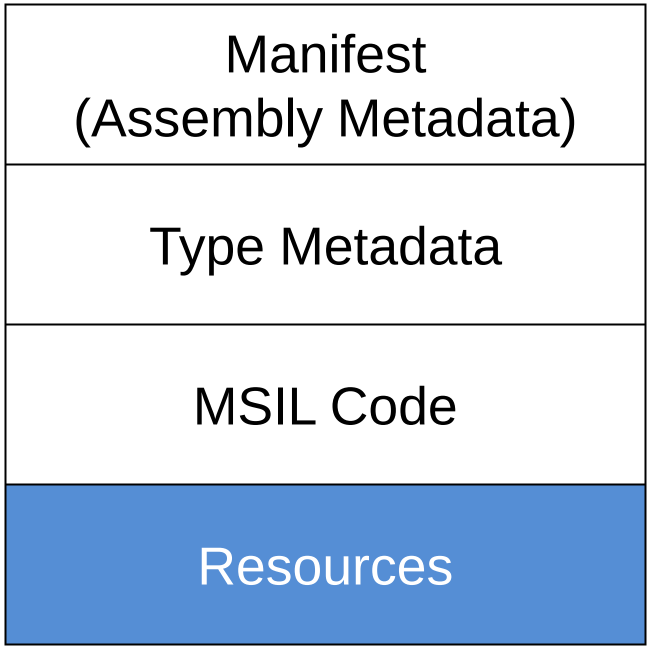
- Contains description of
  - Classes
  - Properties
  - Methods
  - Data types of parameters
  - Return values
  - etc

Assembly Format – MSIL Code



- **Actual binary code** stored for each of the type.

Assembly Format - Resources



- Resources are non executable parts, such as
  - Images
  - Icons
  - Message files
- Specified in .Resources files.

Types of Assembly

Two categories of Assembly

1. Private Assembly
   - Private Assemblies are created when the functionality of component is specifically associated with an application.

2. Shared Assembly
   - Class libraries which are shared between applications.

Private Assembly

- Private Assembly are created when the functionality of component or .dll is specifically associated with an application.

- Private assemblies are created in Application folder, which is bin folder of .NET application

Shared Assembly

- Class libraries which are shared between applications, are called Shared Assemblies.

- Shared assemblies should have **strong names**.

- Shared assemblies should be **installed in GAC.**

Strong Names

- Strong names requires assembly to be digitally signed.
- Digital Signature contains **public and private key.**
- Signed assemblies **ensure security** as well as do not conflict against simple name or versioning.
- In case of same named assemblies, if keys are different then assemblies are treated different.
- Unique **combination of assembly name, version, key** is called as strong name.
- .NET tool for generating strong name is **sn.exe.**

  sn –k <assembly name>.snk

Global Assembly Cache

- Assemblies loaded in GAC are shared assemblies and can be accessed from any application.
- Framework Class Library (FCL) also installed in GAC.
- GAC directory is located in Windows\assembly

Creating Shared Assembly

Perform the following Steps

- Generate strong name for assembly by executing following from visual studio command prompt.

  sn –k <car>.snk

- Configure generated snk file as Key file under Project Settings-> Signing

- Install assembly in GAC by executing

  gacutil –i [full_path_of_ your_dll]

References

- Book referred “Professional .NET Framework” by Wrox publication.
- Book referred “Beginning Visual C# 2005” by Wrox publication.
- http://en.wikipedia.org/wiki/.NET_Framework
- http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx
- http://www.dotnet-tricks.com/Tutorial/netframework/J7N9181013-Understanding-Relationship-Between-CTS-and-CLS.html