

# Ingegneria del Software T

Introduzione al  
*Framework .NET*

# Tecnologia COM

## Component Object Model

- 1993
- COM is a platform-independent, distributed, object-oriented system for creating binary software components
- COM is not an object-oriented language but a **standard**
- COM specifies an **object model** and **programming requirements** that enable COM objects to interact with other objects

# Tecnologia COM

## Component Object Model

```
interface IUnknown
{
    virtual HRESULT QueryInterface(IID iid, void **ppvObject) = 0;
    virtual ULONG AddRef(void) = 0;
    virtual ULONG Release(void) = 0;
};
```

- **QueryInterface** is used to obtain a pointer to another interface, given a **GUID** that uniquely identifies that interface (commonly known as an interface ID, or IID) – if the COM object does not implement that interface, an **E\_NOINTERFACE** error is returned instead
- **AddRef** is used by clients to indicate that a COM object is being referenced
- **Release** is used by clients to indicate that they have finished using the COM object

# Tecnologia COM

## Component Object Model

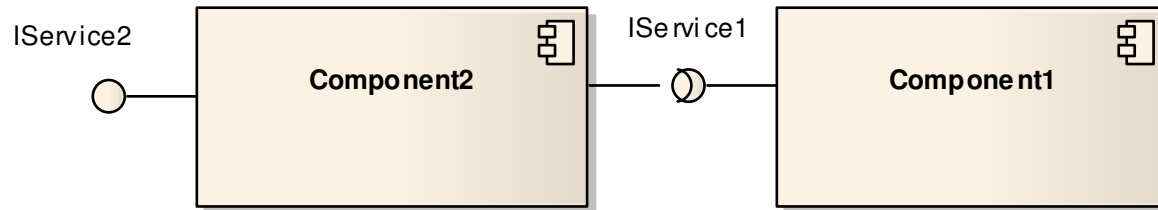
- The COM specifications require a technique called **reference counting** to ensure that individual objects remain alive as long as there are clients which have acquired access to one or more of its interfaces and, conversely, that the same object is properly disposed of when all code that used the object have finished with it and no longer require it
  - **A COM object is responsible for freeing its own memory** once its reference count drops to zero
  - Reference counting may cause **problems if two or more objects are circularly referenced**

# Tecnologia COM

## Component Object Model

- Ereditarietà solo con composizione e delega

```
interface IService2 : IService1
```



# Tecnologia COM

## Component Object Model

- The **location** of each component is stored in the **Windows registry**
  - There can be only one version of a certain component installed
  - This limitation can seriously complicate the **deployment** of COM-based applications, due to the possibility that different programs, or even different versions of the same program, may be designed to work with different versions of the same COM component
  - This condition is known as **DLL hell**

# Framework .NET

- Ambiente di esecuzione (*runtime environment*) + Libreria di classi (standard + estensioni MS)
- Versione 1.0 del 2002 ► v. 4.5
- Semplifica lo sviluppo e il *deployment*
- Aumenta l'affidabilità del codice
- Unifica il modello di programmazione
- È completamente indipendente da COM
- È fortemente integrato con COM

# Framework .NET

- Ambiente *object-oriented*
  - Qualsiasi entità è un oggetto
  - Classi ed ereditarietà pienamente supportati
- Riduzione errori comuni di programmazione
  - ***Garbage Collector***
  - Linguaggi fortemente tipizzati – ***Type Checker***
  - Errori non gestiti ► generazione di eccezioni



# Framework .NET

- Libertà di scelta del linguaggio
  - Funzionalità del *framework* disponibili in tutti i linguaggi .NET
  - I componenti della stessa applicazione possono essere scritti in linguaggi diversi
  - Ereditarietà supportata anche tra linguaggi diversi
- Possibilità di estendere una qualsiasi classe .NET (non *sealed*) mediante ereditarietà  
Diversamente da COM,
  - si usa e si estende la **classe stessa**
  - non si deve utilizzare **composizione e delega**

# Framework .NET

- .NET è un'implementazione di CLI
  - *Common Language Infrastructure*
- CLI e il linguaggio C# sono standard ECMA
  - ECMA-334 (C#), ECMA-335 (CLI)
- Esistono altre implementazioni di CLI:
  - **SSCLI** (Shared Source CLI by Microsoft, per Windows, FreeBSD e Macintosh) - **Rotor**
  - **Mono** (per Linux)
  - **DotGNU**
  - **Intel OCL** (Open CLI Library)
  - ...

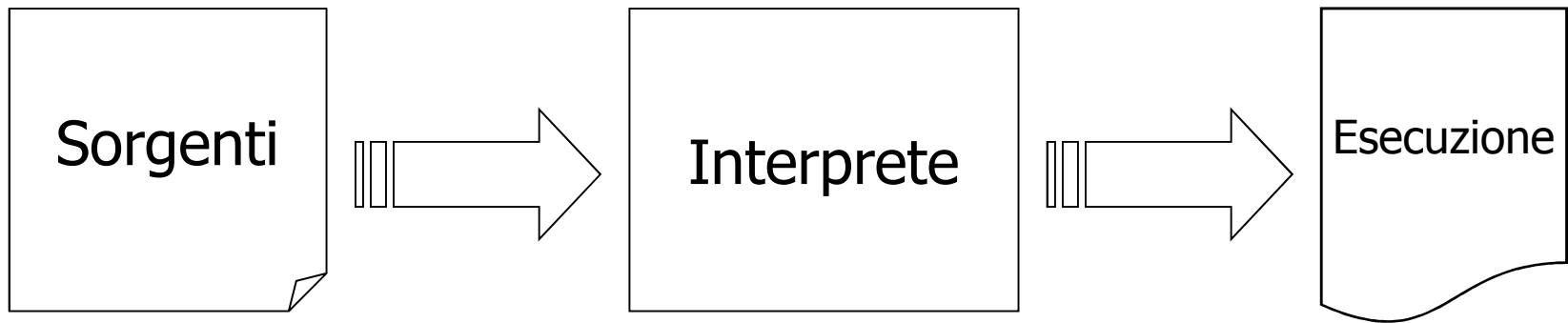
## Standard ECMA-335

- Defines the Common Language Infrastructure (CLI) in which applications written in **multiple high level languages** may be executed in **different system environments** without the need to rewrite the application to take into consideration the unique characteristics of those environments
- CLI is a **runtime environment**, with:
  - a file format
  - a common type system
  - an extensible metadata system
  - an intermediate language
  - access to the underlying platform
  - a factored base class library

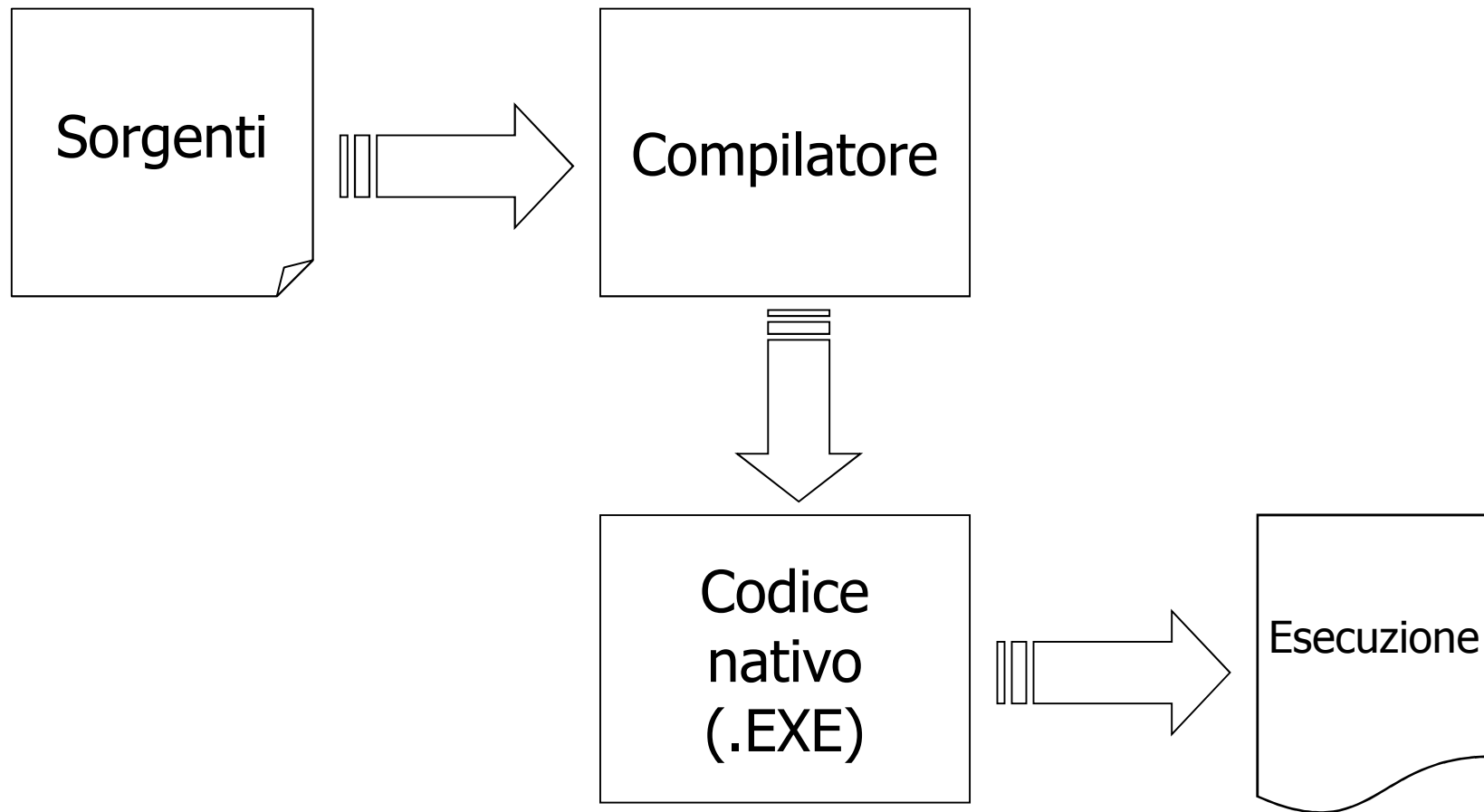
# Framework .NET

- Concetti chiave:
  - (Microsoft) **Intermediate Language** - (MS)IL
  - **Common Language Runtime** - CLR  
ambiente di esecuzione *runtime* per le applicazioni .NET  
il codice che viene eseguito sotto il suo controllo si dice **codice gestito** (*managed*)
  - **Common Type System** - CTS  
tipi di dato supportati dal *framework* .NET  
consente di fornire un modello di programmazione unificato
  - **Common Language Specification** - CLS  
regole che i linguaggi di programmazione devono seguire per essere interoperabili all'interno del *framework* .NET  
sottoinsieme di CTS

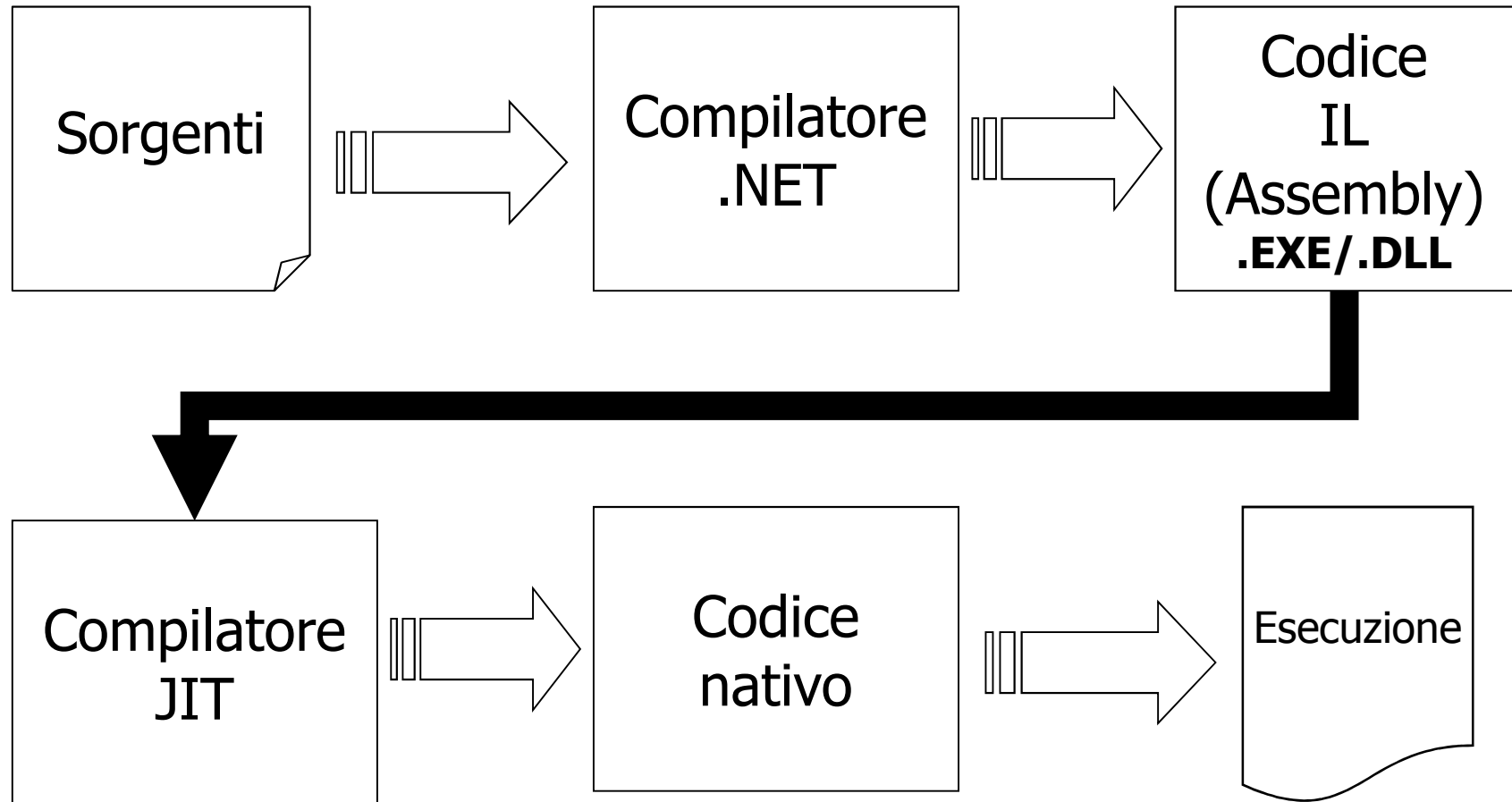
# Codice interpretato



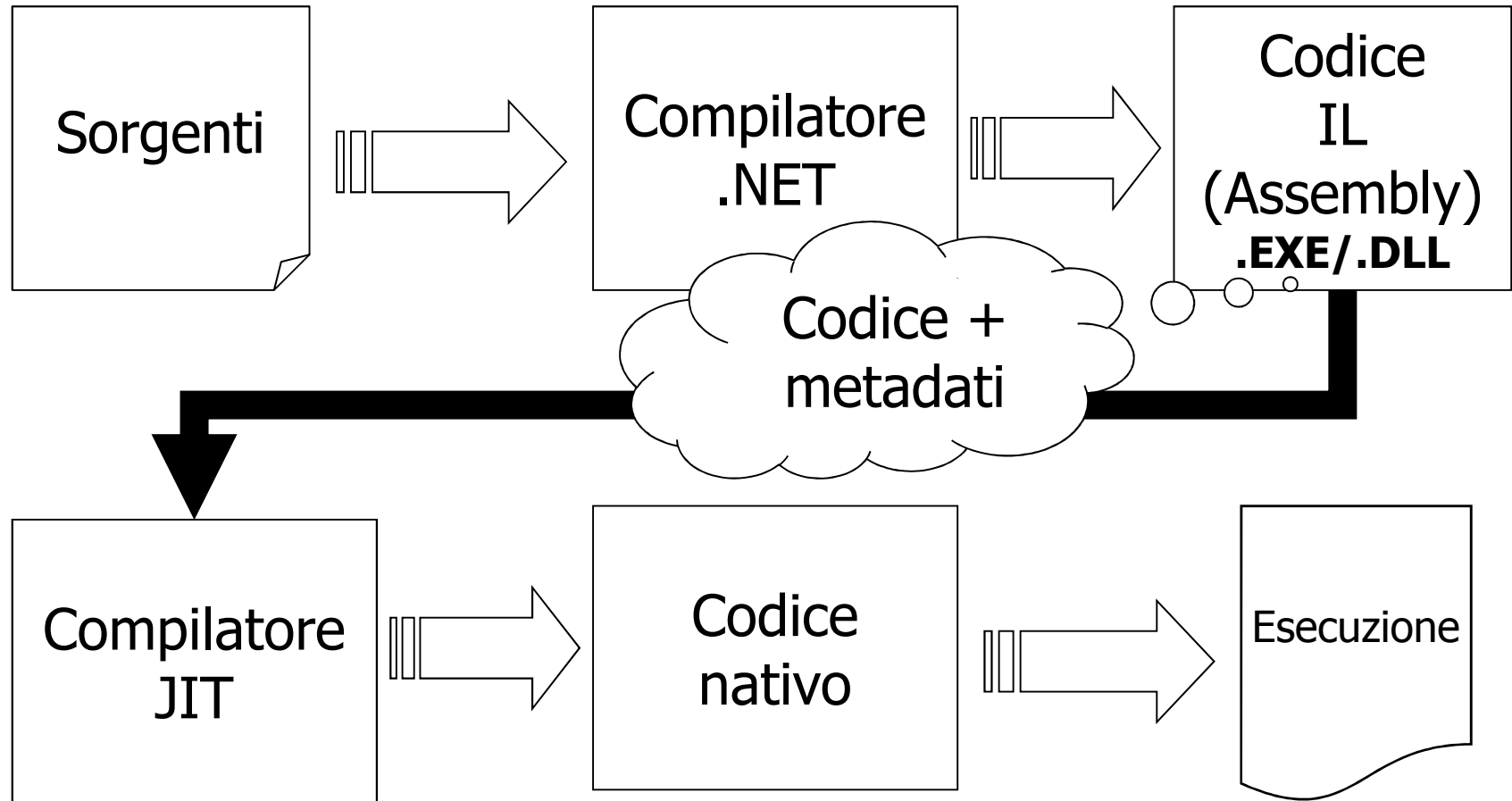
# Codice nativo



# Codice IL

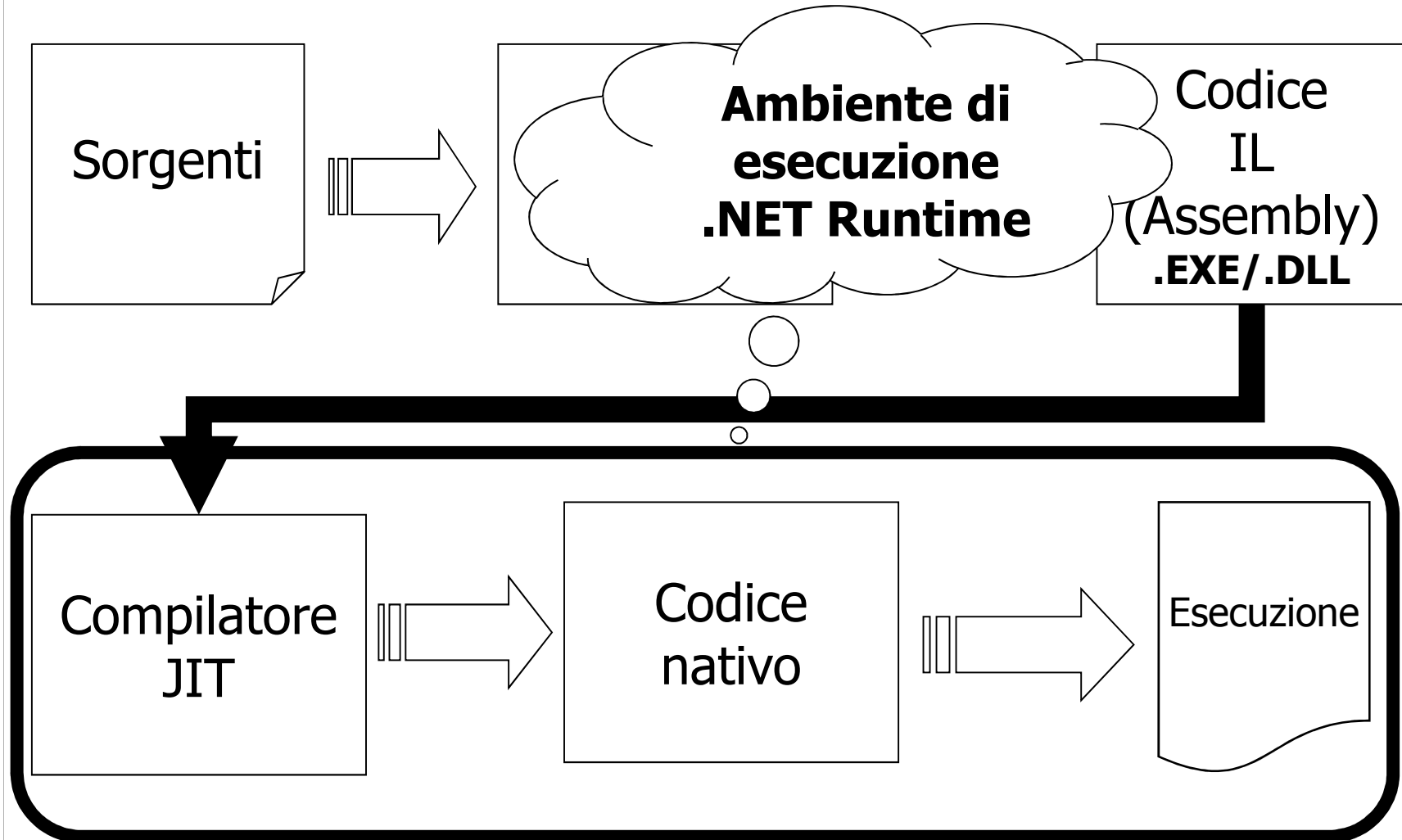


# Codice IL



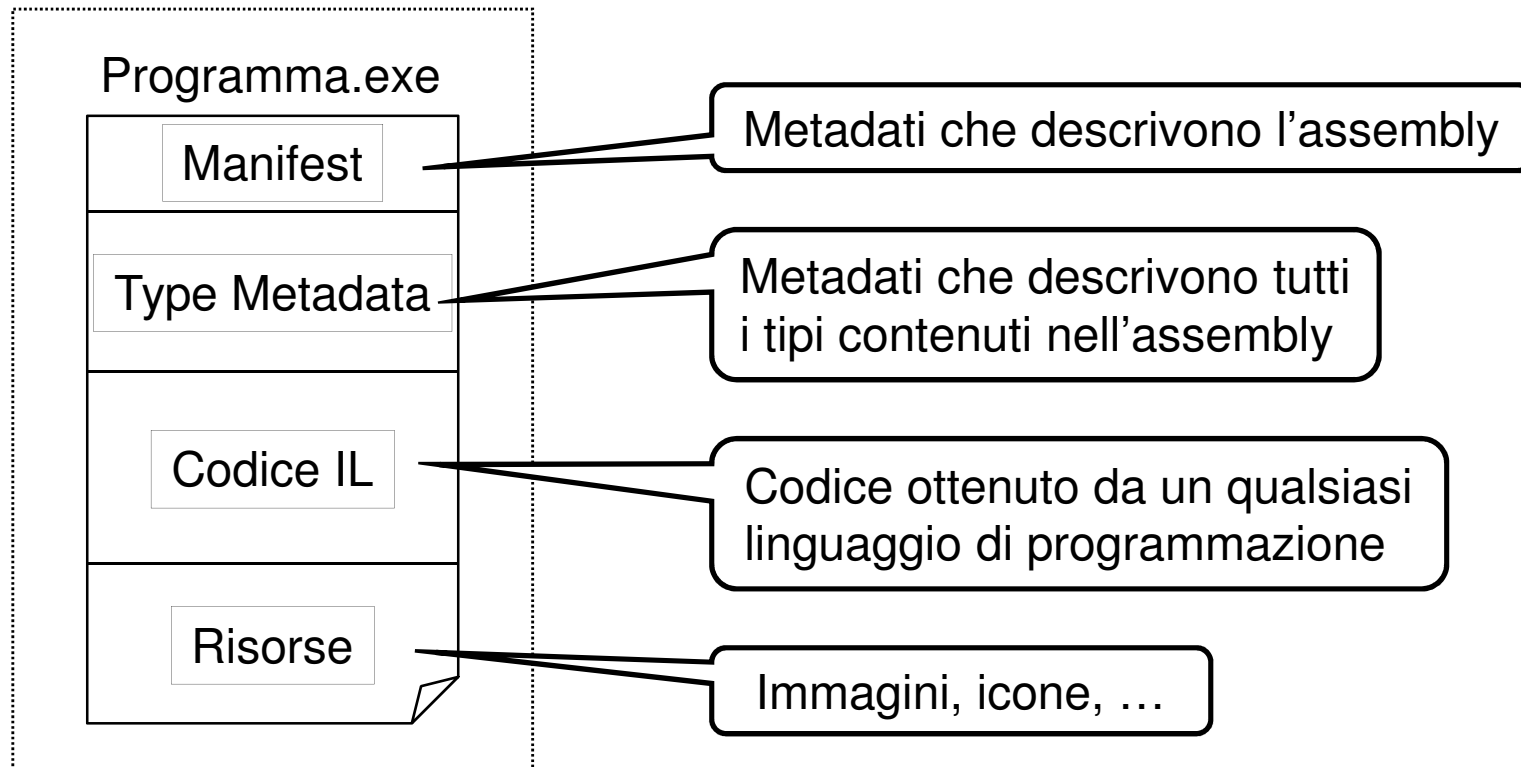


# Codice IL



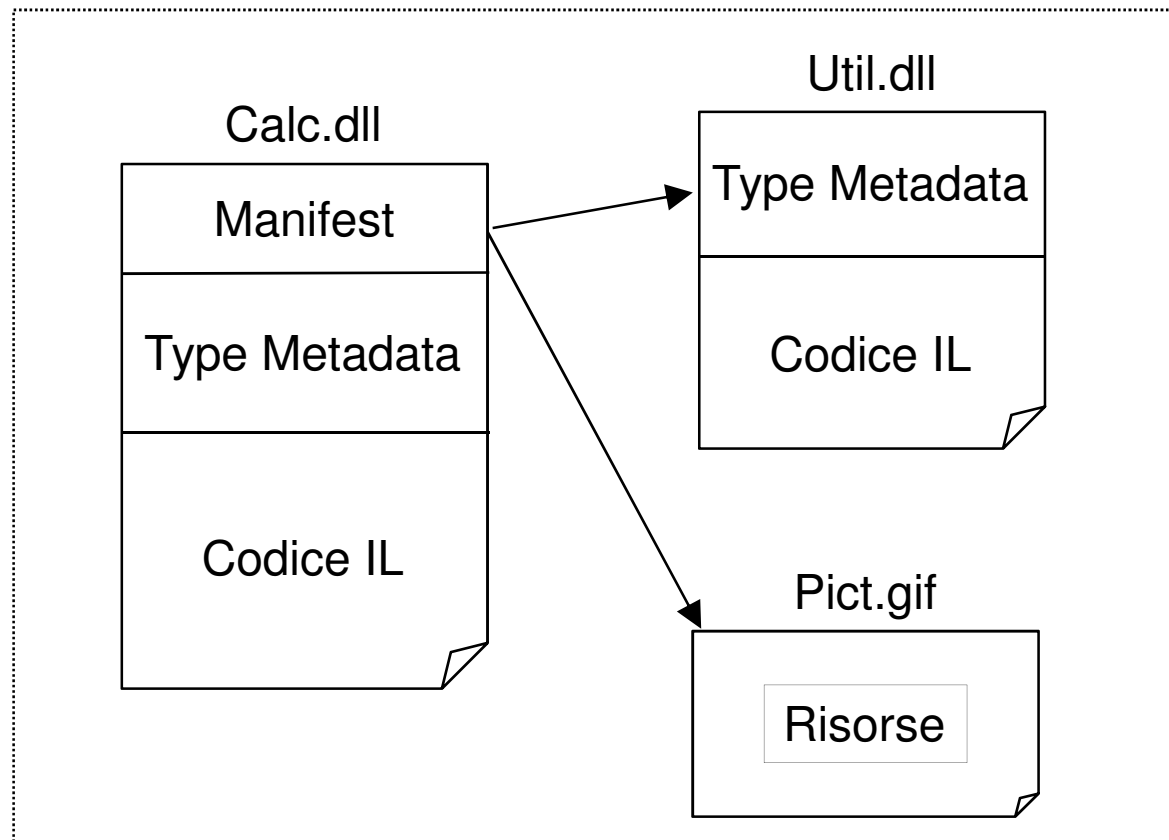
# Assembly

- Unità minima per la distribuzione e il versioning
- Normalmente è composto da un solo file



# Assembly

- Ma può essere composto anche da più file (*module*)



# Metadati

- **Descrizione dell'assembly - Manifest**
  - Identità: nome, versione, cultura [, public key]
  - Lista dei file che compongono l'assembly
  - Riferimenti ad altri assembly da cui si dipende
  - Permessi necessari per l'esecuzione
  - ...
- **Descrizione dei tipi contenuti nell'assembly**
  - Nome, visibilità, classe base, interfacce
  - Campi (attributi membro), metodi, proprietà, eventi, ...
  - Attributi (caratteristiche aggiuntive)
    - definiti dal compilatore
    - definiti dal framework
    - definiti dall'utente

# Chi usa i metadati?

- Compilatori
  - Compilazione condizionale
- Ambienti RAD
  - Informazioni sulle proprietà dei componenti
    - Categoria
    - Descrizione
    - Editor specializzati per tipo di proprietà
- Tool di analisi dei tipi e del codice
  - Intellisense, ILDASM, Reflector, ...
- Sviluppatori – **Reflection** (introspezione)
  - Analisi del contenuto di un assembly

# Assembly

```
.assembly Hello { }  
.assembly extern mscorlib { }  
.method public static void main()  
{  
    .entrypoint  
    ldstr "Hello IL World!"  
    call void [mscorlib]System.Console::WriteLine  
        (class System.String)  
    ret  
}
```

```
ilasm helloil.il
```

# Assembly

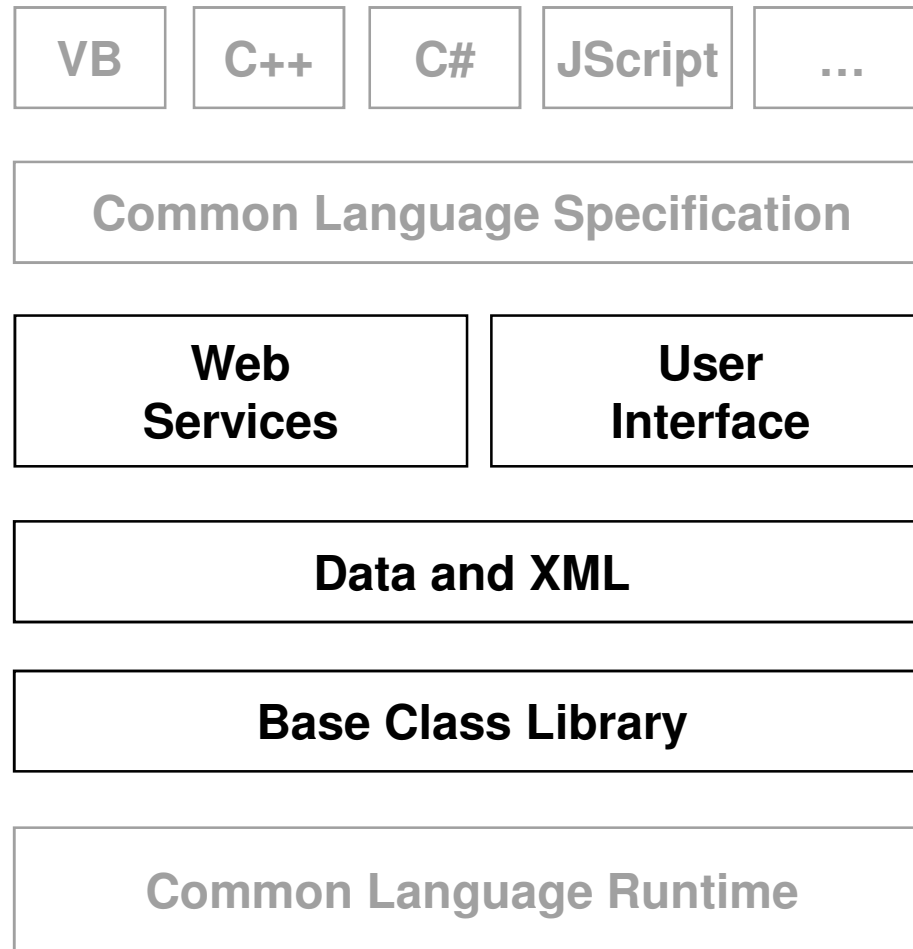
- **Assembly privati**
  - Utilizzati da un'applicazione specifica
  - *Directory* applicazione (e *sub-directory*)
- **Assembly condivisi**
  - Utilizzati da più applicazioni
  - ***Global Assembly Cache (GAC)***
  - c:\windows\assembly
- **Assembly scaricati da URL**
  - *Download cache*
  - c:\windows\assembly\download

# Deployment semplificato

- Installazione senza effetti collaterali
  - Applicazioni e componenti possono essere
    - condivisi o
    - privati
- Esecuzione *side-by-side*
  - Versioni diverse dello stesso componente possono coesistere, anche nello stesso processo

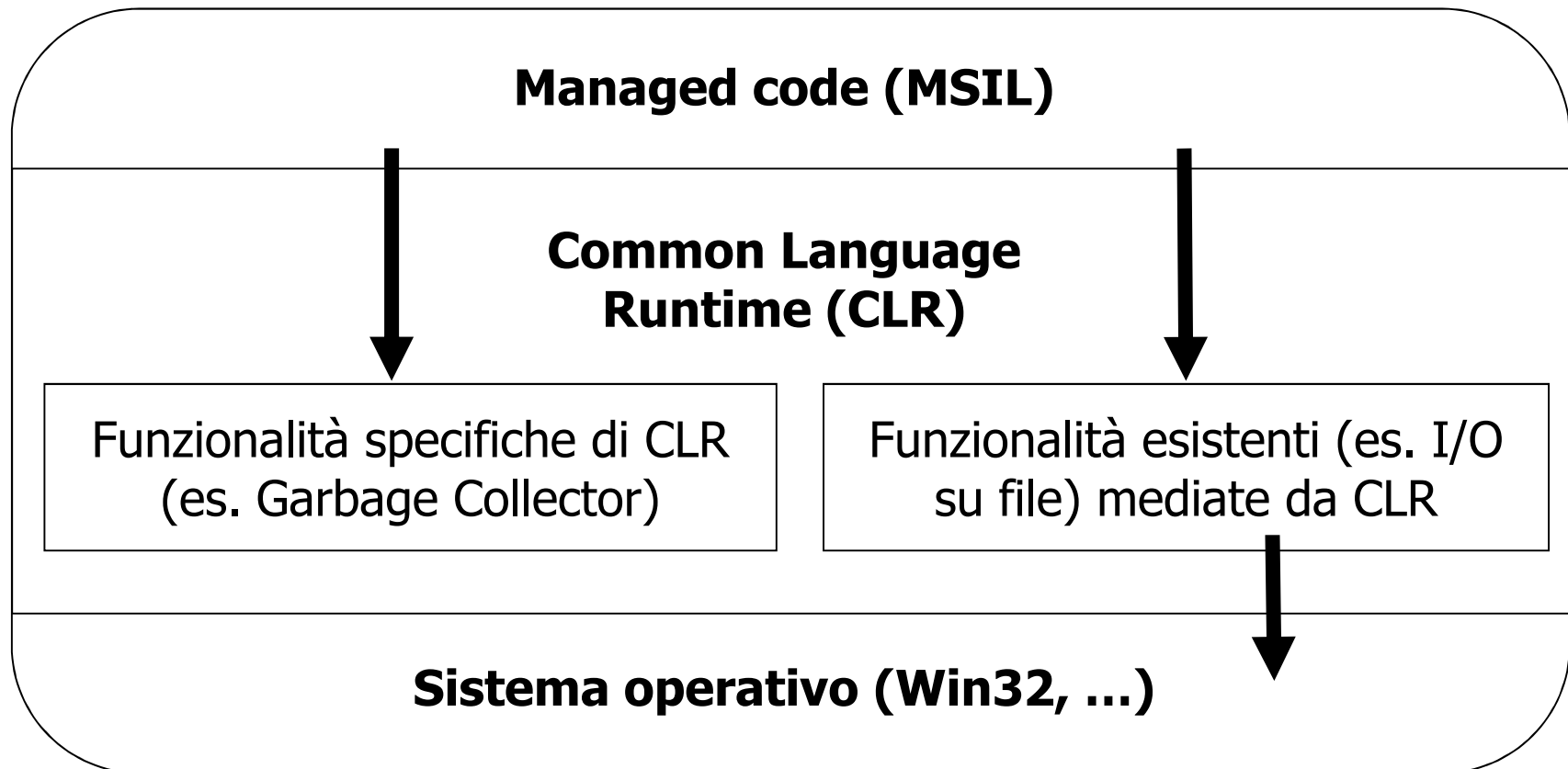


# Common Language Runtime

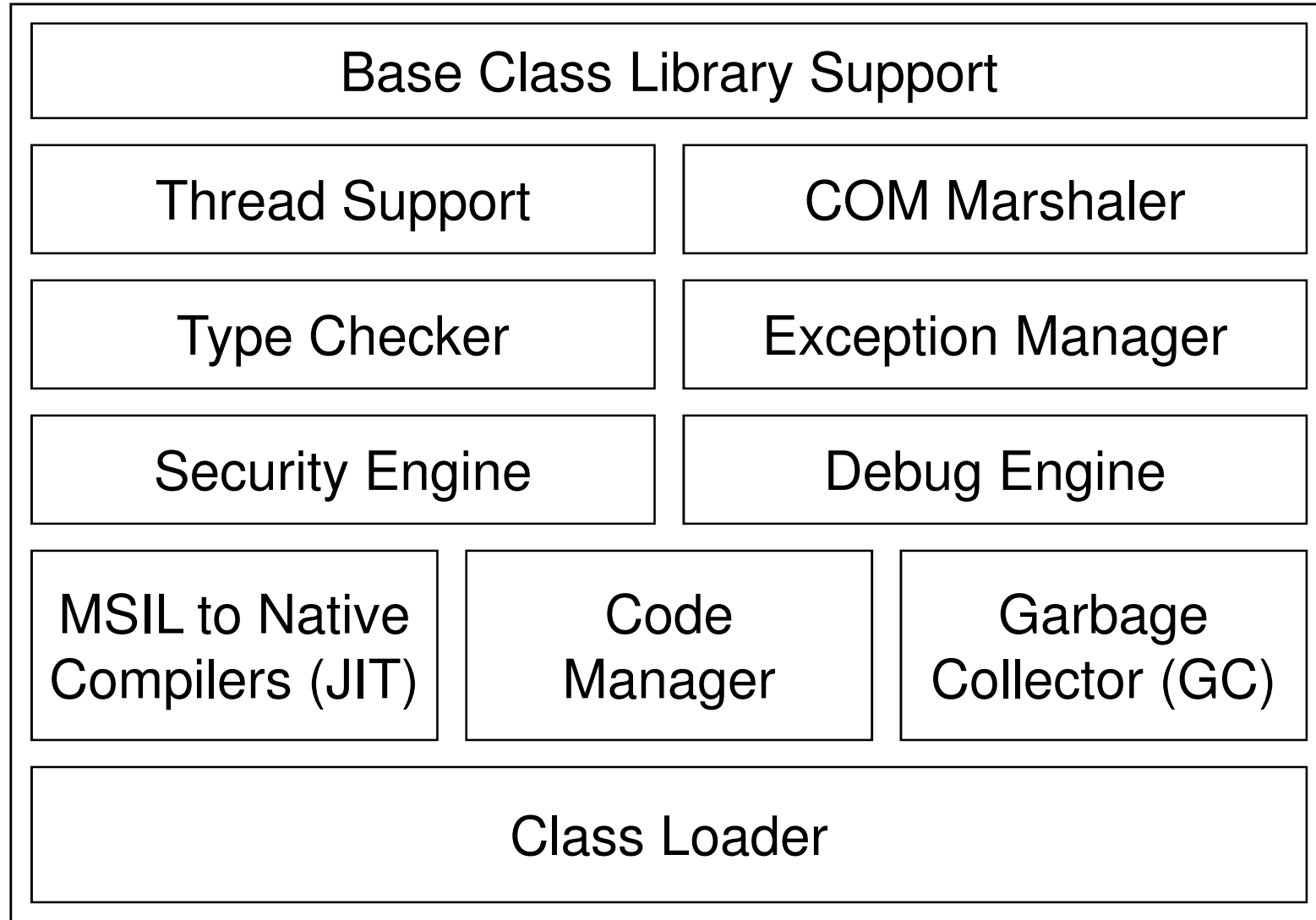


# Common Language Runtime

- IL CLR offre vari servizi alle applicazioni



# Common Language Runtime



# Garbage Collector

- Gestisce il ciclo di vita di tutti gli oggetti .NET
- Gli oggetti vengono distrutti automaticamente quando non sono più referenziati
- A differenza di COM,  
non si basa sul *Reference Counting*
  - Maggiore velocità di allocazione 😊
  - Consentiti i riferimenti circolari 😊
  - Perdita della distruzione deterministica 😊

# Gestione delle eccezioni

- Un'eccezione è
  - una condizione di errore
  - un comportamento inaspettatoincontrato durante l'esecuzione del programma
- Un'eccezione può essere generata dal
  - codice del programma in esecuzione
  - ambiente di *runtime*
- In CLR, un'eccezione è un oggetto che eredita dalla classe **System.Exception**
- Gestione uniforme - elimina
  - codici **HRESULT** di COM
  - codici di errore Win32
  - ...

# Gestione delle eccezioni

- Concetti universali
  - Lanciare un'eccezione (**throw**)
  - Catturare un'eccezione (**catch**)
  - Eseguire codice di uscita da un blocco controllato (**finally**)
- Disponibile in tutti i linguaggi .NET con sintassi diverse

# Common Type System

- Tipi di dato supportati dal *framework* .NET
  - Alla base di tutti i linguaggi .NET
- Fornisce un modello di programmazione unificato
- Progettato per linguaggi object-oriented, procedurali e funzionali
  - Esamine le caratteristiche di 20 linguaggi
  - Tutte le funzionalità disponibili con IL
  - Ogni linguaggio utilizza alcune caratteristiche

# Common Type System

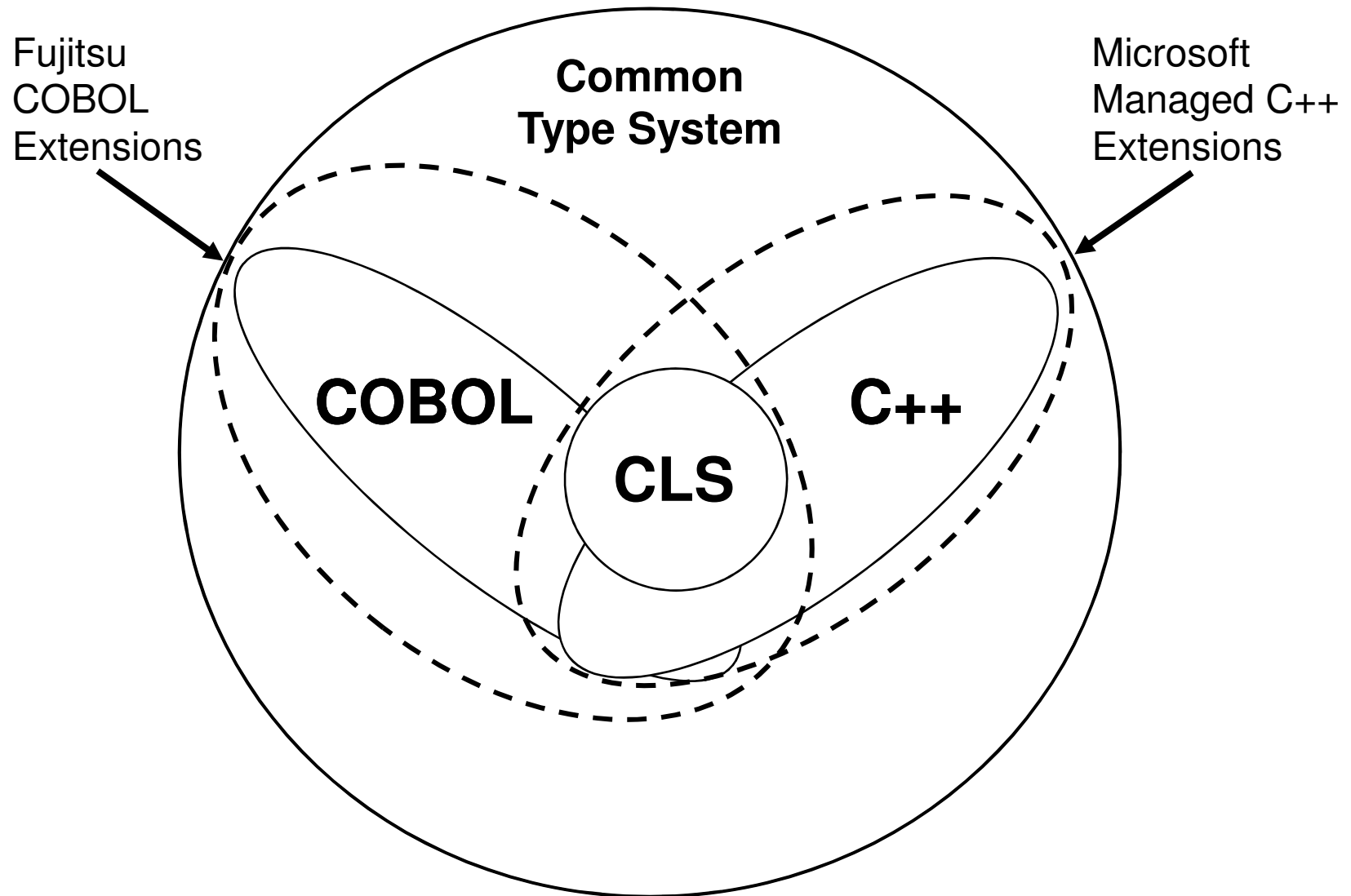
- Alla base di tutto ci sono i tipi:  
**classi, strutture, interfacce, enumerativi, delegati**
- Fortemente tipizzato (*compile-time*)
- *Object-oriented*
  - Campi, metodi, tipi nidificati, proprietà, ...
- *Overload* di funzioni (*compile-time*)
- Invocazione metodi virtuali risolta a *run-time*
- Ereditarietà singola di estensione
- Ereditarietà multipla di interfaccia
- Gestione strutturata delle eccezioni



# Common Language Specification

- Definisce le regole di compatibilità tra linguaggi (sottoinsieme di CTS)
  - Regole per gli identificatori
    - Unicode, *case-sensitivity*
    - *Keyword*
  - Regole per denominazione proprietà ed eventi
  - Regole per costruttori degli oggetti
  - Regole di *overload* più restrittive
  - Ammesse interfacce multiple con metodi con lo stesso nome
  - Non ammessi puntatori *unmanaged*
  - ...

# Common Language Specification



# Common Type System

## Tipi nativi

CTS	C#
System.Object	object
System.String	string
System.Boolean	bool
System.Char	char
System.Single	float
System.Double	double
System.Decimal	decimal
System.SByte	sbyte
System.Byte	byte
System.Int16	short
System.UInt16	ushort
System.Int32	int
System.UInt32	uint
System.Int64	long
System.UInt64	ulong

# Common Type System

- Tutto è un oggetto
  - `System.Object` è la classe radice
- Due categorie di tipi
  - **Tipi riferimento**
    - Riferimenti a oggetti allocati sull'*heap* gestito
    - Indirizzi di memoria
  - **Tipi valore**
    - Allocati sullo *stack* o parte di altri oggetti
    - Sequenza di byte

# Common Type System

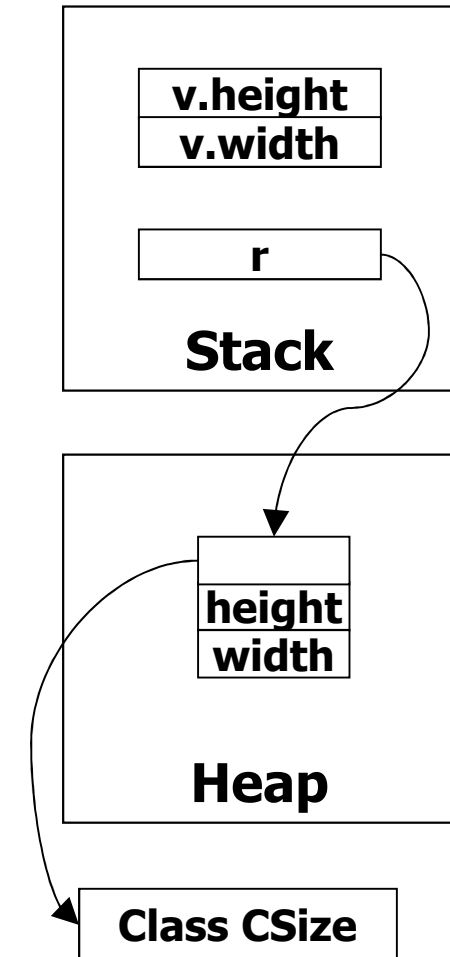
## Tipi valore

- I tipi valore comprendono:
  - Tipi primitivi (*built-in*)
    - Int32, ...
    - Single, Double
    - Decimal
    - Boolean
    - Char
  - Tipi definiti dall'utente
    - Strutture (**struct**)
    - Enumerativi (**enum**)

# Common Type System

## Tipi valore vs tipi riferimento

```
public struct Size
{
    public int height;
    public int width;
}
public class CSize
{
    public int height;
    public int width;
}
public static void Main()
{
    Size v;           // v istanza di Size
    v.height = 100;  // ok
    CSize r;         // r è un reference
    r.height = 100;  // NO, r non assegnato
    r = new CSize(); // r fa riferimento a un CSize
    r.height = 100;  // ok, r inizializzata
}
```



# Common Type System

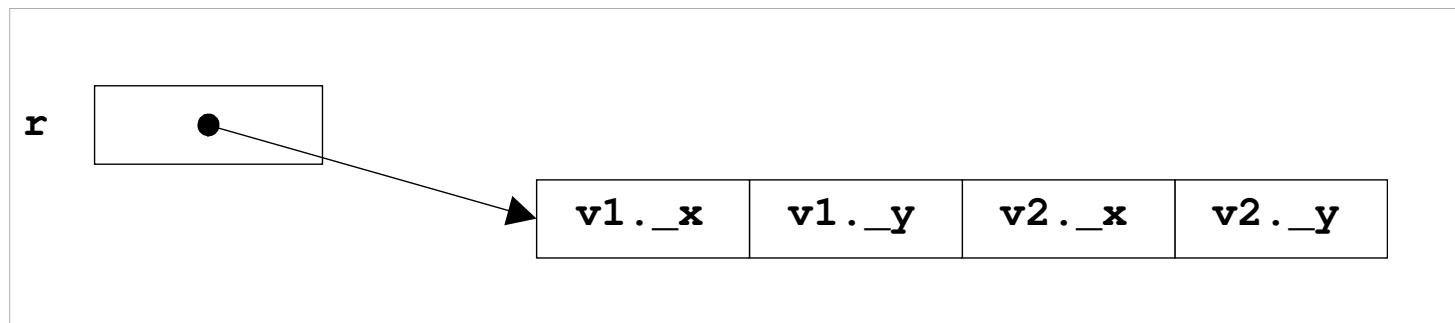
## Tipi valore vs tipi riferimento

```
public struct Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
}
```

# Common Type System

## Tipi valore vs tipi riferimento

```
public class Rectangle
{
    Point v1;
    Point v2;
    ..
}
..
Rectangle r = new Rectangle();
```





# Common Type System

## Tipi valore vs tipi riferimento

```
...  
Point[] points = new Point[100];  
for (int i = 0; i < 100; i++)  
    points[i] = new Point(i, i);  
...
```

- Alla fine, rimane 1 solo oggetto nell'*heap* (l'array di **Point**)

```
...  
Point[] points = new Point[100];  
for (int i = 0; i < 100; i++)  
{  
    points[i].X = i;  
    points[i].Y = i;  
}  
...
```

# Common Type System

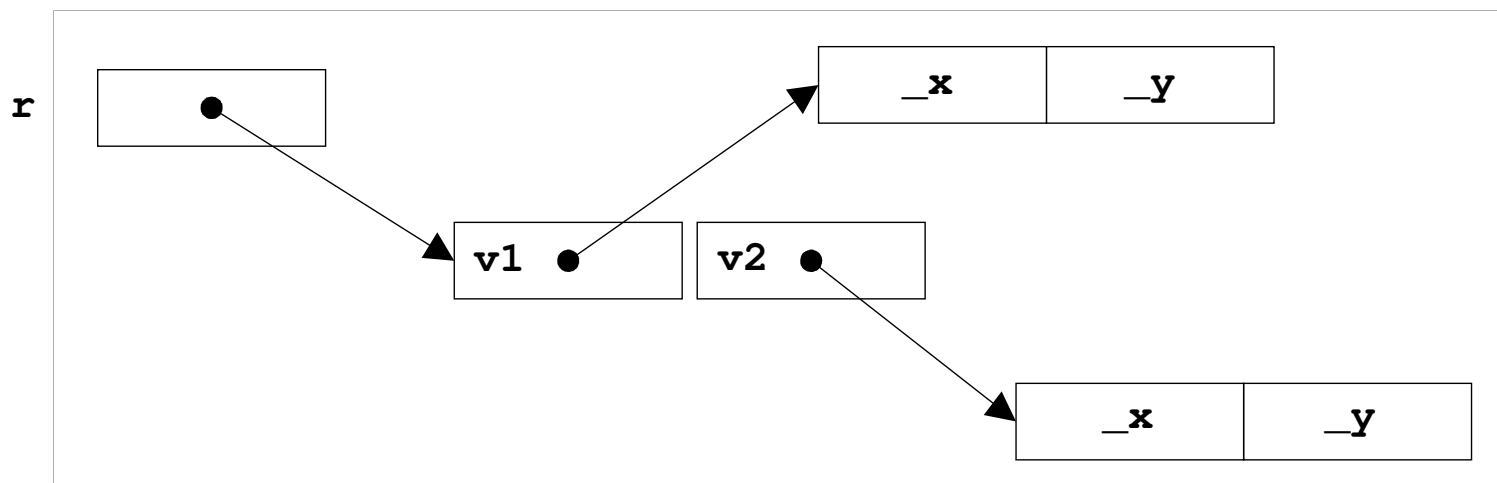
## Tipi valore vs tipi riferimento

```
public class Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
}
```

# Common Type System

## Tipi valore vs tipi riferimento

```
public class Rectangle
{
    Point v1;
    Point v2;
    ..
}
..
Rectangle r = new Rectangle();
```



# Common Type System

## Tipi valore vs tipi riferimento

```
...  
Point[] points = new Point[100];  
for (int i = 0; i < 100; i++)  
    points[i] = new Point(i, i);  
...
```

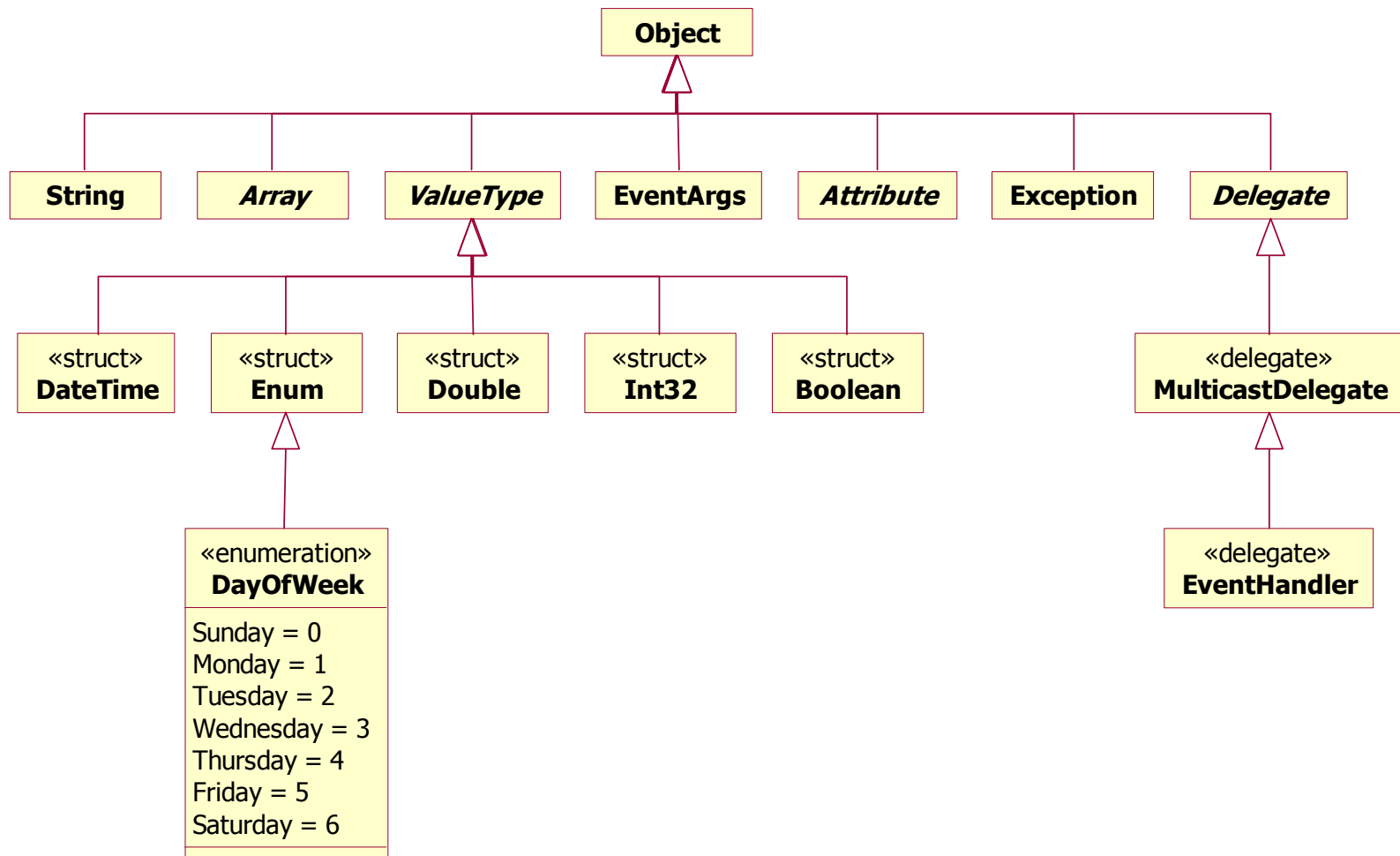
- Alla fine, rimangono 101 oggetti nell'*heap* (1 array di `Point` + 100 `Point`)

```
...  
Point[] points = new Point[100];  
for (int i = 0; i < 100; i++)  
{  
    points[i].X = i;  
    points[i].Y = i;  
}  
...
```

**NO!**

# Common Type System

## Tipi valore e tipi riferimento



# Boxing / Unboxing

- Un qualsiasi tipo valore può essere automaticamente convertito in un tipo riferimento (***boxing***) mediante un *up cast* implicito a **System.Object**

```
int i = 123;  
object o = i;
```

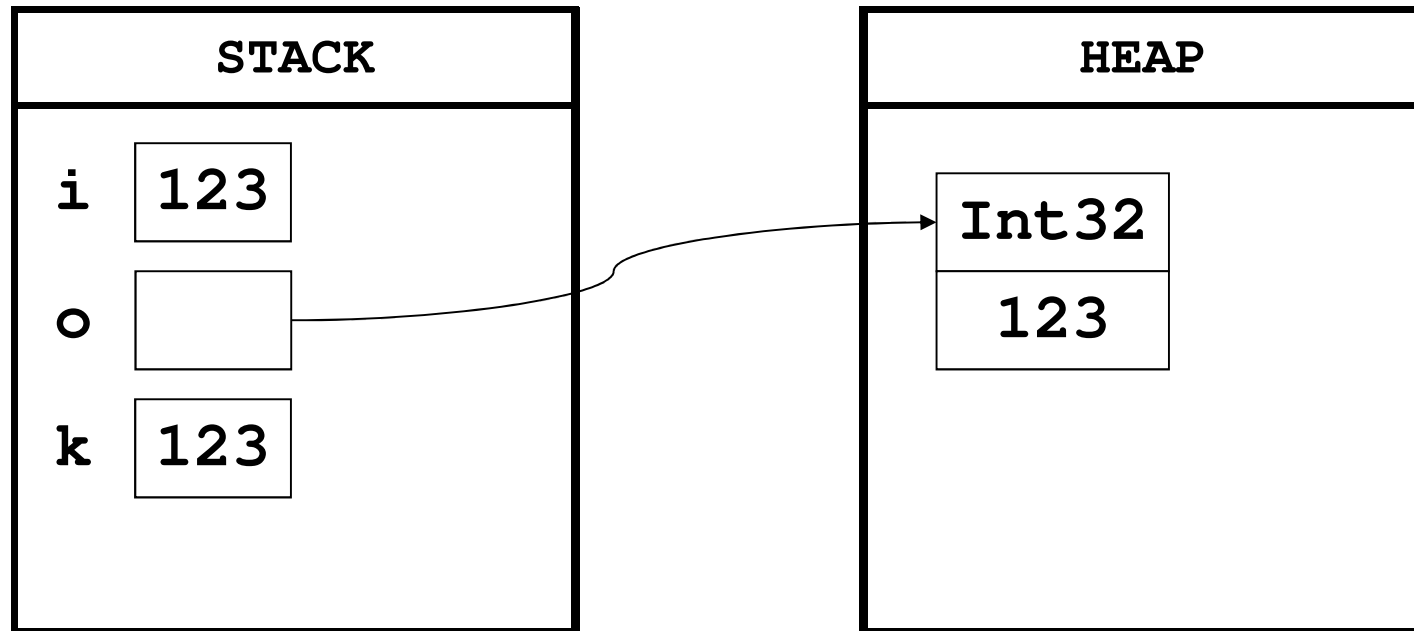
- Un tipo valore “*boxed*” può tornare ad essere un tipo valore standard (***unboxing***) mediante un *down cast* esplicito

```
int k = (int) o;
```

- Un tipo valore “*boxed*” è un **clone indipendente**

# Boxing / Unboxing

```
int i = 123;  
object o = i;  
int k = (int) o;
```



# Bibliografia

Libri di base:

- *D. S. Platt*, **Introducing Microsoft® .NET**, Second Edition (\*)
- *J. Sharp, J. Jagger*, **Microsoft® Visual C#™ .NET Step by Step** (\*)
- *T. Archer, A. Whitechapel*, **Inside C#**, Second Edition (\*)
- *M. J. Young*, **XML Step by Step**, Second Edition (\*)
- *R. M. Riordan*, **Microsoft® ADO.NET Step by Step** (\*)

Libri avanzati:

- *J. Richter*, **Applied Microsoft® .NET Framework Programming**
- *C. Petzold*, **Programming Microsoft® Windows® with C#** (\*)
- *S. Lidin*, **Inside Microsoft® .NET IL Assembler**

(\*) Disponibile nella biblioteca del DEIS