Microsoft .net

C#

Plateforme .Net

Technical Overview

Agenda

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**
- **CLR et le monde « Non Managé »**

Agenda

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**

Introduction

Unifier le developpement

- En finir avec le langage C++
- Unifier le développement Web et les applications sur poste
- Unifier les developpeurs C++ et VB (et plus)

Stabiliser les applications

- Résoudre l'enfer des Dll
- Améliorer la sécurité à tout les niveaux

Simplifier les développement

- Offrir une architecture à base de composants plus simple que COM
- Simplifier les frameworks utilitaires
- Faciliter le devellopement

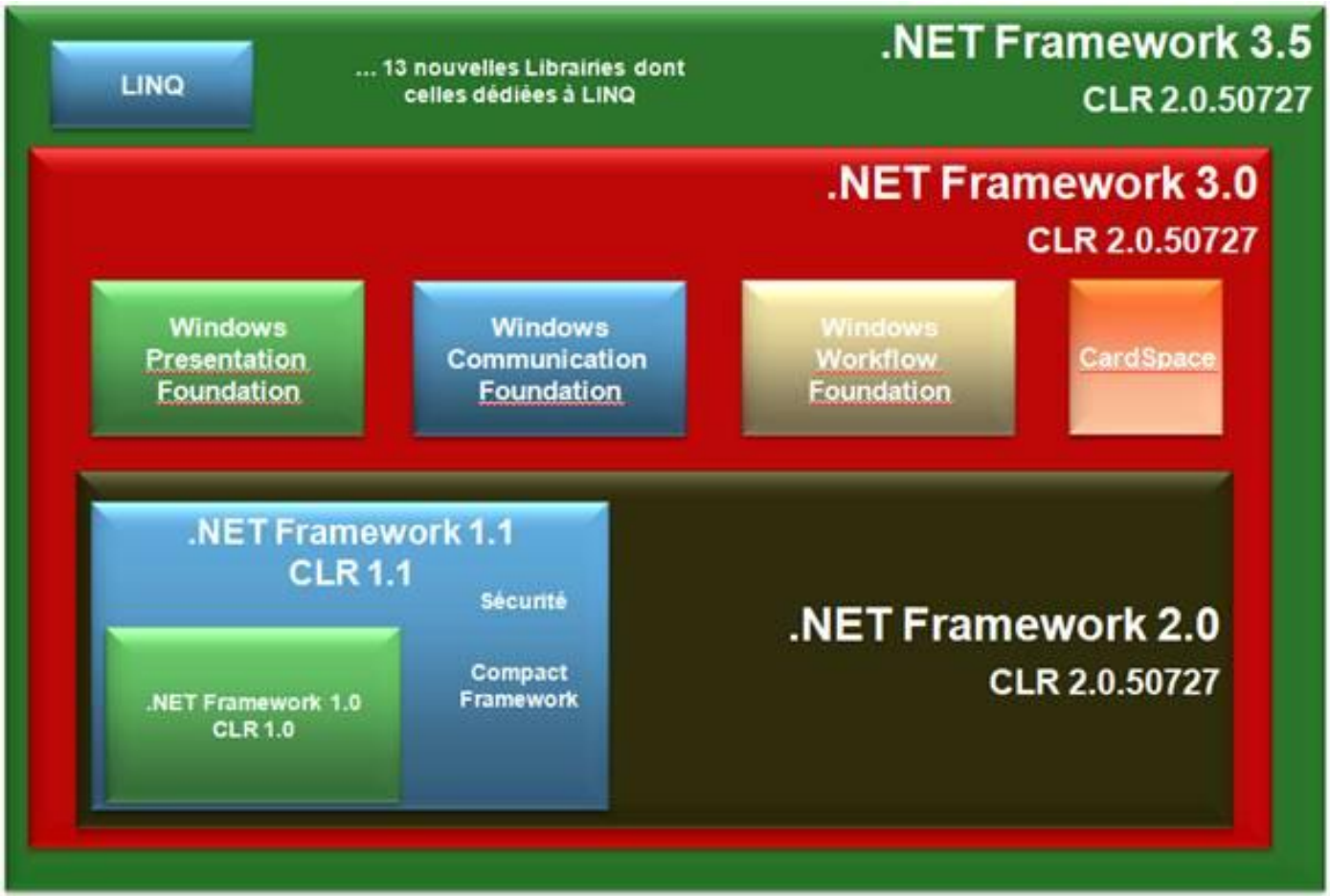
Introduction

Février 2002	.Net 1.0 / Visual Studio 7
Mars 2003	.Net 1.1 / Visual Studio 2003
Novembre 2005	.Net 2.0 / Visual Studio 2005
Février 2007	.Net 3.0 / AddIn Visual Studio 2005
Novembre 2007	.Net 3.5 / Visual Studio 2008
Avril 2010	.Net 4.0 / Visual Studio 2010
Aout 2012	.Net 4.5 / Visual Studio 2012

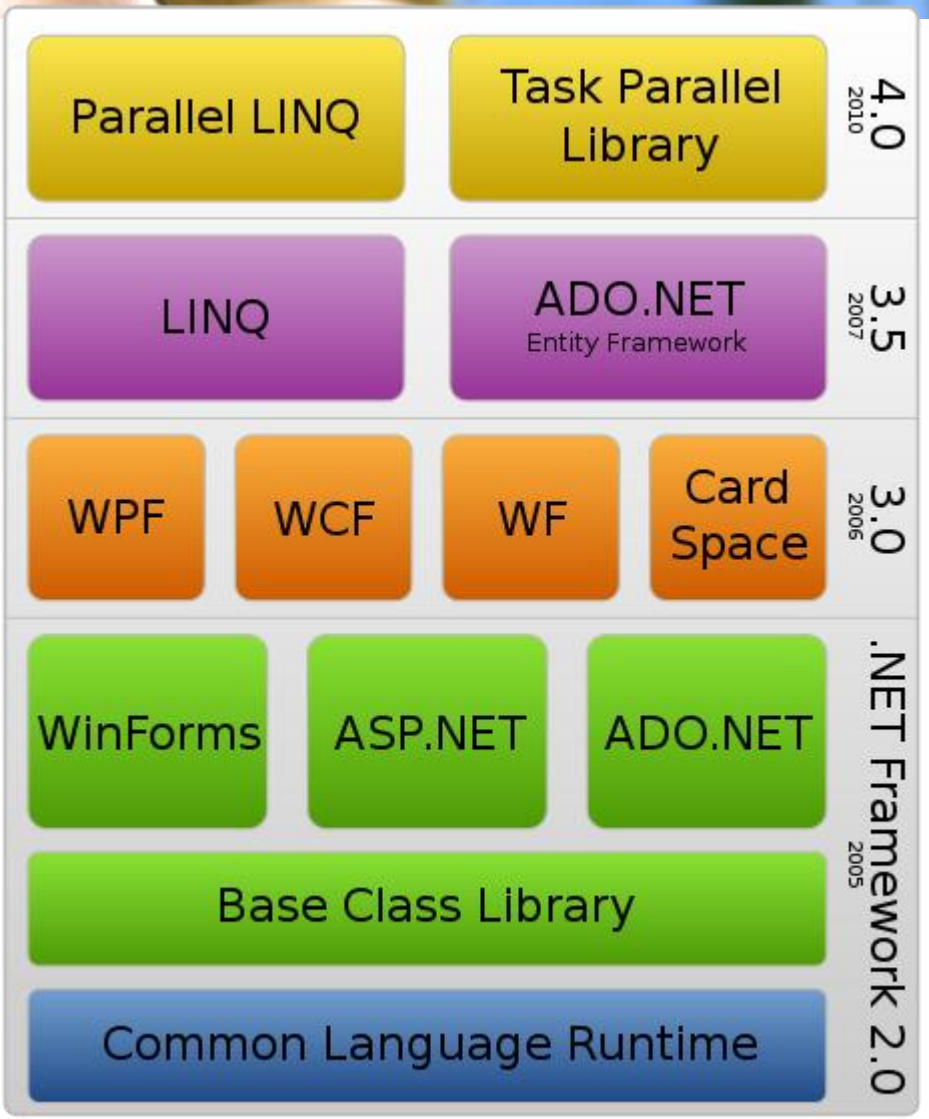
2007/2008	Framework 3.5 (WPF / WCF / WWF …)
	C# 3.0 / LINQ
	Visual Studio 2008

2010	Type dynamique
	Paramètres optionnels
	Task Parallel Library
	Parallel LINQ

Introduction

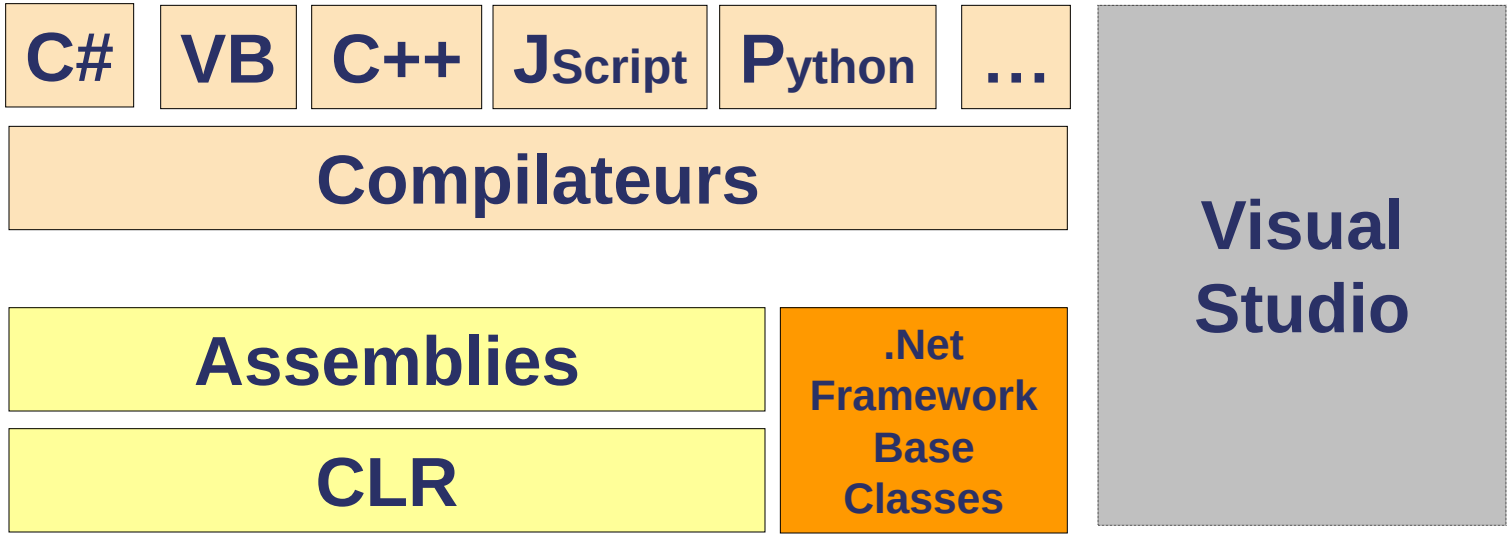


Introduction



The .NET Framework Stack

Architecture



Introduction

Du développeur à l'utilisateur

Code source

Compilateur

Assemblies

IL

Type Def

Host du CLR

CLR

JIT

Résolution de Types
Chargement des Assemblies

Code Machine

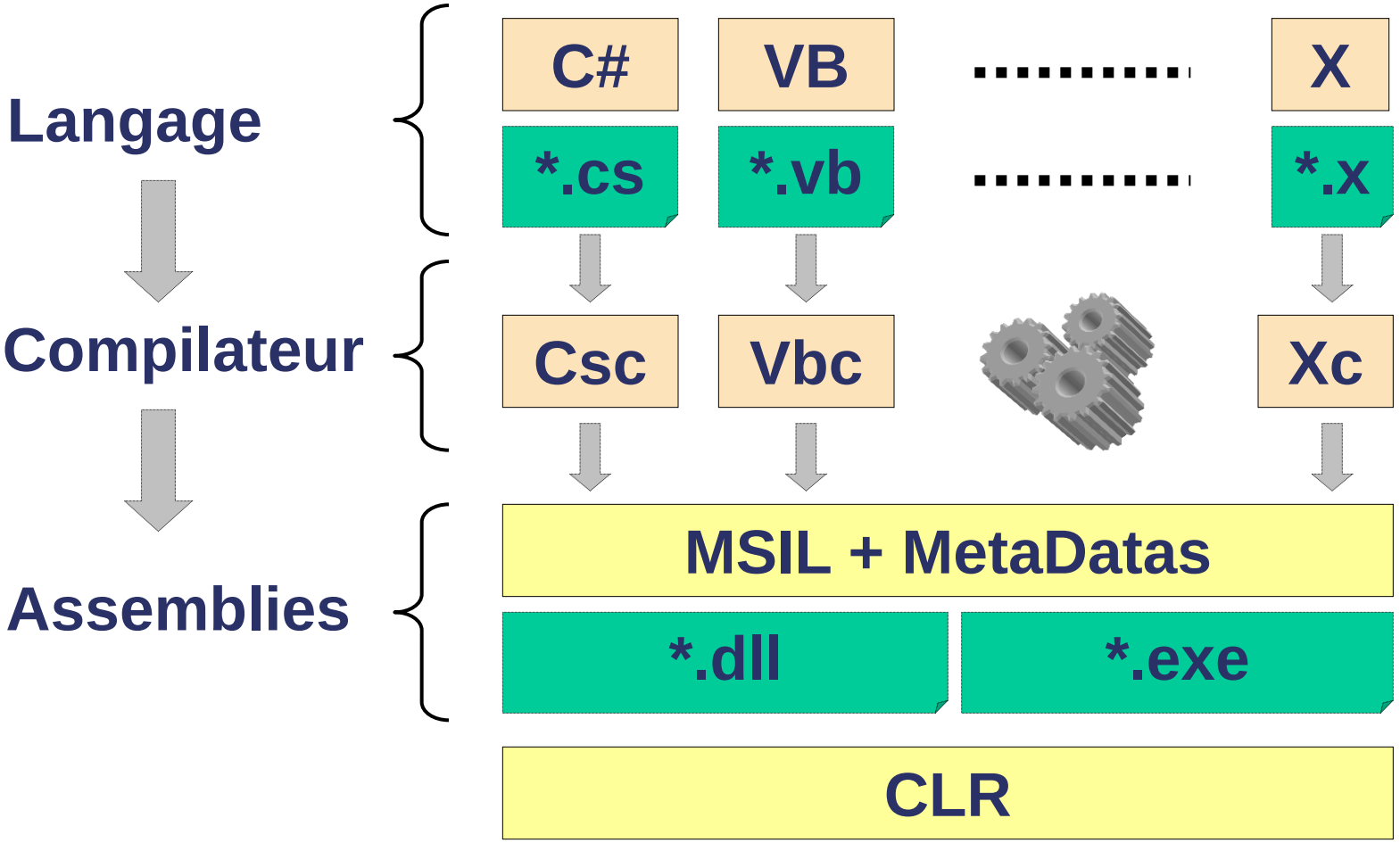
Agenda

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**

La compilation

Du langage vers IL

Le Rôle des compilateurs : Créer des Assemblies



IL : Définition

- **Plus petit dénominateur commun des langages .Net**
- **C'est LE seul langage compris par le CLR**
- **Spécifié par l'ECMA**

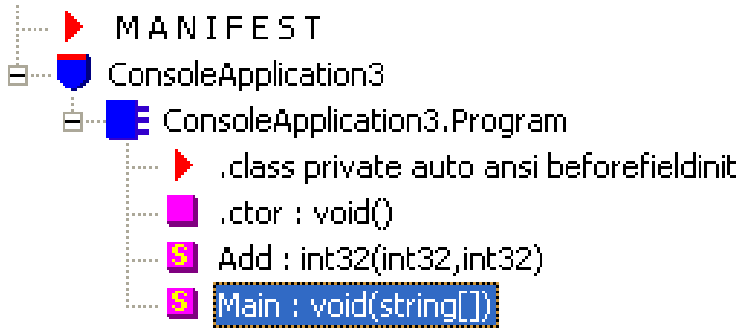
- **Langage à pile qui est gérée par le CLR**
  - `stloc` dépile la valeur du sommet
  - `ldloc` ajoute la valeur au sommet
- **Langage objet**
  - `newobj` créer un objet, appel le constructeur et le place en haut de la pile
  - `callvirt` supporte le polymorphisme

IL : Exemple

ILDAsm.exe

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint // Code size       14 (0xe)
  .maxstack  2
  .locals init ([0] int32 a,[1] int32 b,[2] int32 result)
  IL_0000:  nop
  IL_0001:  ldc.i4.1
  IL_0002:  stloc.0
  IL_0003:  ldc.i4.2
  IL_0004:  stloc.1
  IL_0005:  ldloc.0
  IL_0006:  ldloc.1

  IL_0007:  call int32 ConsoleApplication3.Program::Add(int32,
                                                        int32)
  IL_000c:  stloc.2
  IL_000d:  ret
} // end of method Program::Main
```



Agenda

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**

Les Rôles du CLR (Common Langage Runtime)

- Chargement des assemblages et résolution des types
- Compilation du code IL en code machine
- Hébergement de plusieurs applications dans un même processus Windows (AppDomain)
- Résolution des types
- *Gestion des exceptions*
- *Destruction des objets devenus inutiles (GC)*

CLR
Chargement

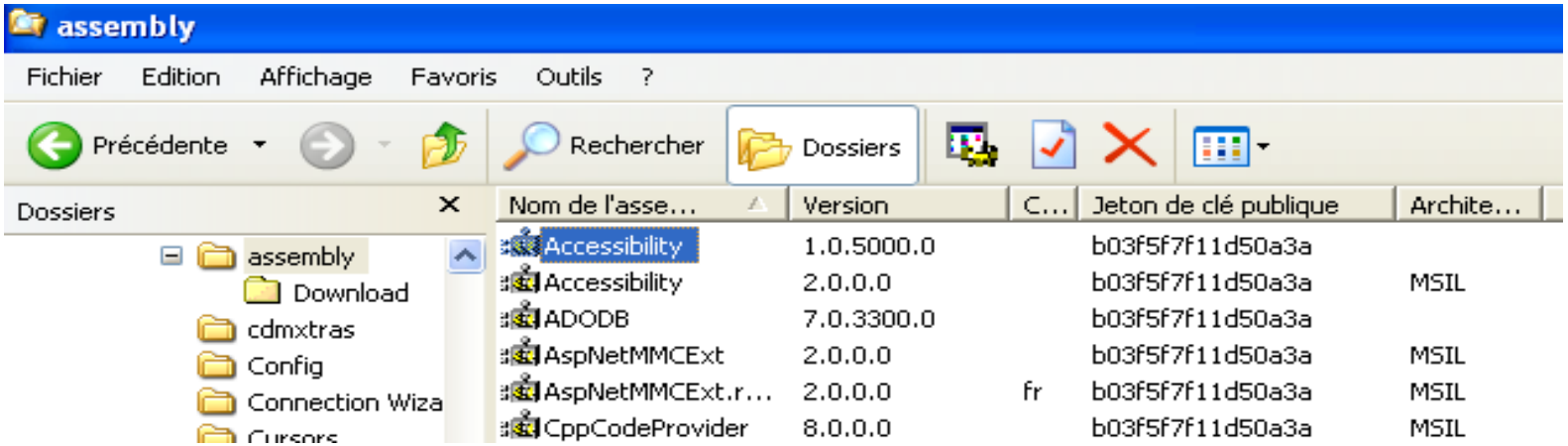
- Le CLR est un jeu de DLL « in-process »
- Les exécutables .Net sont « linkés » à la librairie mscoree.dll (cale)
- Le « cale » charge soit mscorwks.dll (mono-pro) ou mscorsvr.dll (bi-pro)

CLR
Le GAC

Global Assembly Cache

- C'est le premier endroit où le CLR va chercher une Assembly qui est « ***signée*** ».
- Le GAC permet le stockage « ***side by side*** » de plusieurs versions d'une même Assembly

(Résout le problème de l'enfer des Dll)



CLR
Localisation Assembly

- **CodeBase**
  - Chemin complet vers une Assembly
- **Probing**
  - Algorithme de recherche d'une Assembly dans le répertoire courant
- **Stratégie Applicative, éditeur , système…**

- **AssemblyResolve**
  - Event sur l'AppDomain où l'on peut définir sa propre stratégie de localisation

CLR
Algorithme Fusion

Localisation d'un assembly

Assembly.Load

Nom fort

o

Stratégie Editeur

o

Application de la stratégie

o

Dans le GAC ?

n

n

o

CodeBase

o

Assembly Localisée à l'URL

o

Assembly trouvée

n

n

Le Probing Trouve l'Assembly

n

AssemblyResolve Retourne l'Assembly

o

o

Echec

n

Exemple de fichier de configuration

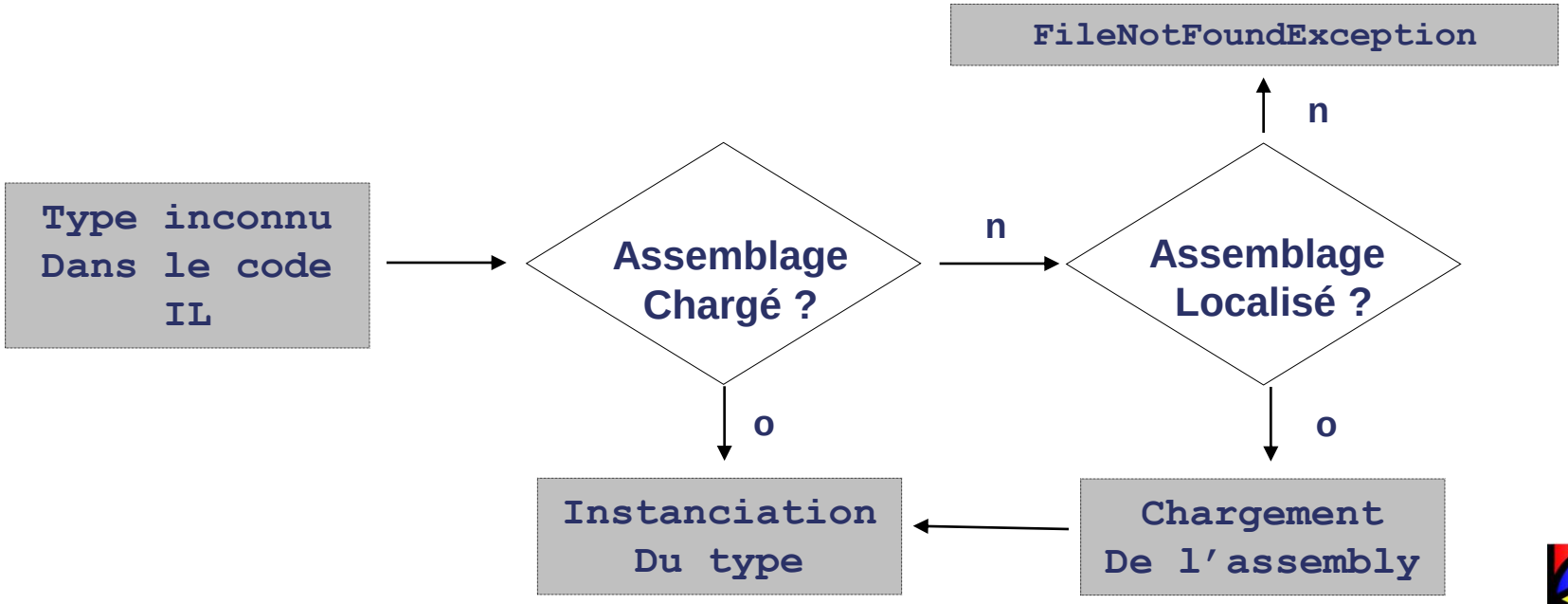
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <runtime>
  <assemblyBinding xmlns='urn:schemas-microsoft-com:asm.v1'>
   <probing privatePath='?;?;?'/>
    <dependentAssembly>
     <assemblyIdentity name ='?' culture ='neutral' publicKeyToken='?'>
      <publisherPolicy apply ='no'/>
      <bindingRedirect oldVersion='1.0.0.0' newVersion ='2.0.0.0'/>
     </assemblyIdentity>
    </dependentAssembly>
   </assemblyBinding>
   <assemblyBinding>
    <assemblyIdentity name ='?' culture ='neutral' publicKeyToken='?'>
     <codebase version ='2.0.0.0' href ='file:///c:/....'/>
    </assemblyIdentity>
   </assemblyBinding>
 </runtime>
</configuration>
```

CLR
Résolution des types

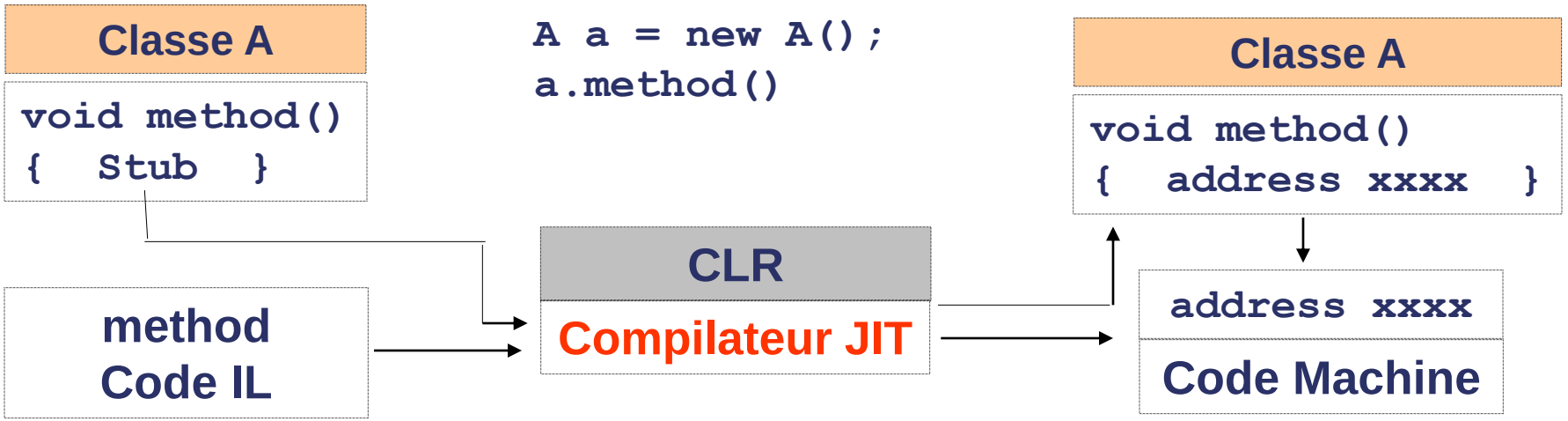
2 cas possibles :

- Chargement implicite
- Chargement explicite

« loader » de la CLR
qui charge l'assembly



CLR
Compilation



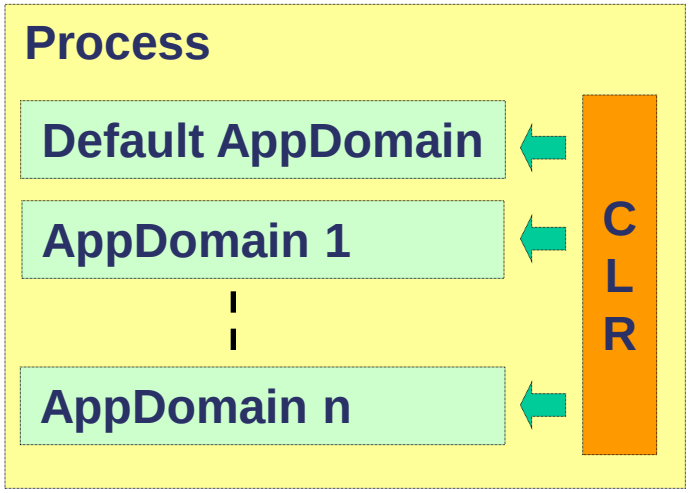
JIT : Just In Time

Passage du code IL en code Machine

Ngen : outil qui permet de tout JIT Compiler

Pitching : Opération inverse du CLR

CLR AppDomain



AppDomain est un *« Process Léger »*

- Partage la CLR et les types de base
- Chaque AppDomain est isolé
- Chaque AppDomain a sa propre configuration (sécurité et exceptions)
- Seul moyen de décharger une assembly

CLR AppDomain

EXEMPLE

```
static void Main(string[] args)
 {
  AppDomain currentDomain = AppDomain.CurrentDomain;
  System.Console.WriteLine(currentDomain.FriendlyName);
  AppDomain newDomain = AppDomain.CreateDomain("Nouveau Domaine");
  newDomain.DoCallBack(new CrossAppDomainDelegate(Fct));
  AppDomain.Unload(newDomain);
 }

 public static void Fct()
 {
   System.Console.WriteLine(AppDomain.CurrentDomain.FriendlyName);
 }
```

CLR
Un mot sur le GC

Le CLR possède un « tas géré»

Garbage Collector (Ramasse Miettes)

- Prend en charge la désallocation des objets du « tas géré »
- Un seul par processus

Algorithme en trois étapes

- Collecte
- Désallocation (thread dédié)
- Défragmentation du tas

Algorithme à base de générations (3)

- Les objets passent d’une génération à l’autre

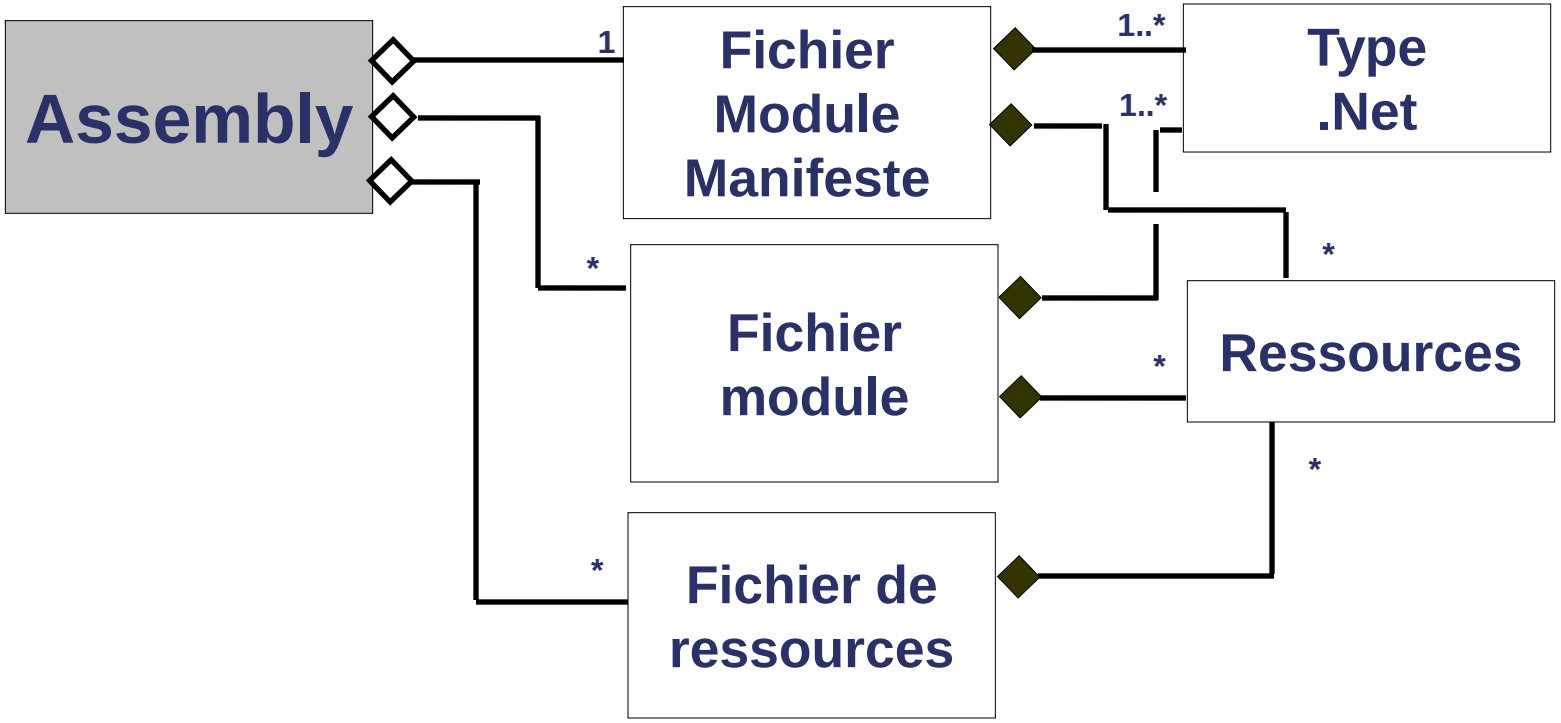
Agenda

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**

Assembly
Structure et définition

Structure logique

« Une « Assembly » est une structure logique »



Assembly
Structure et définition

Structure physique d'un Module

- *Entête PE* : C'est un fichier Windows
- *Entête CLR* : Version du CLR, point d 'entrée géré
- *Manifeste* : Section qui décrit l'assembly et des attributs « métadonnées de l'assemblage »
- *Métadonnées* : décrit complètement les types contenus dans le module

Entête PE	Entête CLR	Manifeste	Metadonnées	Code IL	Ressources

Assembly
Les « MetaDonnées »

Les « MétaDonnées » :

Mini base de données composée d'un ensemble de tables

- Structures du manifeste
  - AssemblyDef, FileDef, ManifestRessourceDef…
- Définition des types
  - TypeDef, MethodDef, FieldDef, PropertyDef…
- Les types référencés
  - ModuleRef, TypeRef, MemberRef …

Assembly
Structure et définition

Une table d'une ligne :

Les attributs de l'Assembly

Attribut	Description
AssemblyCompagny	•*Nom de l'éditeur*
AssemblyProduct	•*Nom du produit*
AssemblyTitle	•*Titre de l'assemblage*
AssemblyVersion	•*Nom utilisé pour constitué le nom fort*
AssemblyCulture	•*Culture de l'assemblage utilisé pour le nom fort*
AssemblyKeyFile	•*Spécifie le fichier clé / valeure*
….	

Assembly
Structure et définition

Structure du manifeste

Tables	Colonnes	Descriptions
AssemblyDef	•*Nom* •*Version* •*Culture* •*Clé Publique*	Une seule ligne par assemblage
FileDef	•*Nom du fichier* •*Valeur de hash*	Une ligne pour tout les autres modules excepté celui du Manifest
ManifestRessourceDef	•*Un lien vers FileDef* •*Fichier et type binaire*	Une ligne pour chaque ressource de l'assemblage
ExportedTypeDef	•*Nom du type* •*Lien vers FileDef* •*Lien TypeDef (metadonnées)*	Une ligne pour chaque type exporté

Assembly
Structure et définition

Les tables des définitions

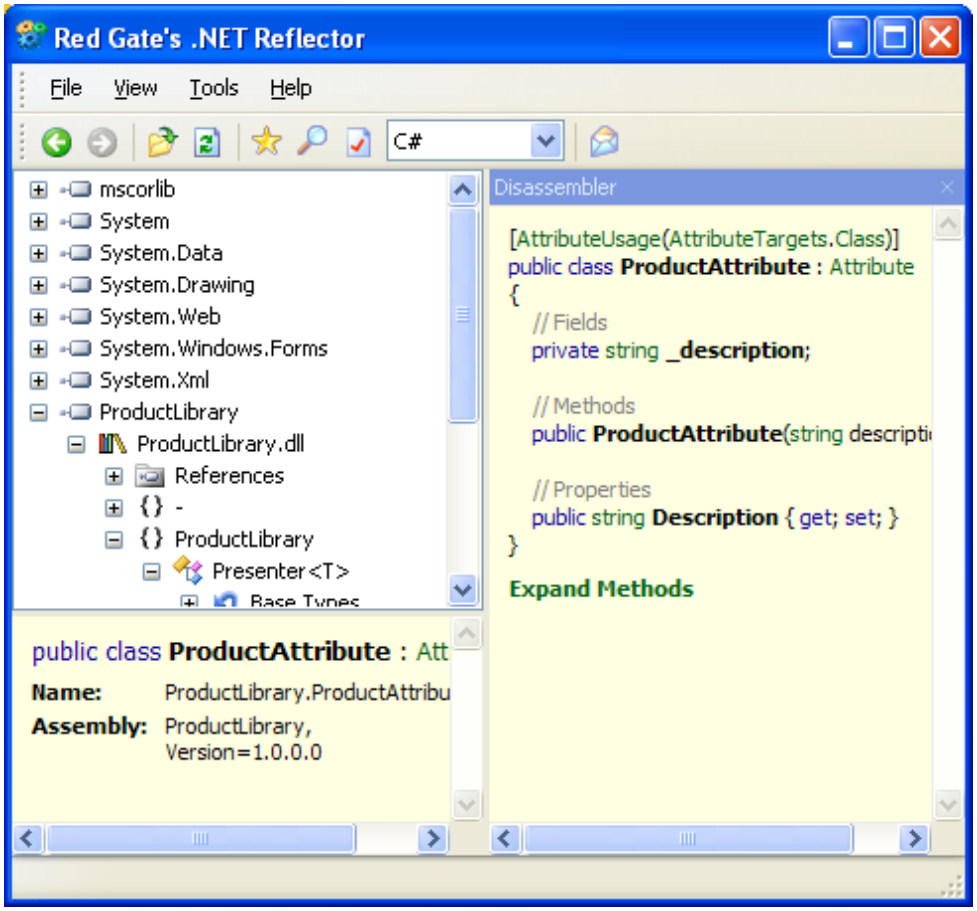
Tables	Colonnes
ModuleDef	*•Nom du ficher*
TypeDef	*•Nom et portée* *•Pointeurs vers ses « méthodes / MethodDef »,* *« champs/FieldDef »,* *« propriétés/PropertyDef », «event/EventDef »*
MethodDef	*•Nom et portée* *•Signature* *•Offset vers IL* *•Pointeurs vers les « paramètres/ ParamDef »*
FieldDef	*•Nom et portée* *•Type*
PropertyDef	*•Nom et portée* *•Type …*
EventDef	*•Nom et portée …*
ParamDef	*•Nom* *•In, out retval …*
….	

Assembly
Structure et définition

Les tables des références

Tables	Colonnes	Descriptions
AssemblyRef	•*Nom de l'assemblage* •*Version* •*Culture* •*Clé publique*	Une ligne pour chaque Assembly référencé
ModuleRef	•*Nom du Module*	Une ligne pour chaque module contenu dans cet assemblage
TypeRef	•*Nom* •*Reference vers ModuleRef, AssemblyRef, ou TypeRef*	Si le type est dans un module : *ModuleRef*, dans une autre Assembly : *AssemblyRef*, et *TypeRef* si c'est un type encapsulé
MemberRef	•*Nom* •*Pointeurs vers TypeRef*	Un membre peut être une méthode, un champs , un propriété …

Assembly Reflector



Reflector .Net affiche :

- La définition d'une asembly
- Les « Meta-Datas »
- Le code IL dans un langage spécifié
- Analyse les dépendances
- ….

Assembly
Nom fort : définition

Nom Fort (Strong Name) :

- Authentifier une Assembly
- Nommer de façon unique

Composition du Nom Fort

- Nom du fichier
- Clé publique
- Version de l'Assembly
- Culture de l'Assemblage

Ne référence que des Assembly à Nom Fort

Assembly
Signature : définition

Entête PE
Entête CLR
Manifeste
Métadonnées
Code IL
Ressources

Signature numérique
Clé publique
Clé privée
Valeur de Hachage

Entête PE
Entête CLR
Signature numérique
Manifeste
Clé publique
Métadonnées
Code IL
Ressources

Signature numérique
Clé publique
Valeur de Hachage Décrypté

?

Valeur de Hachage calculé

Entête PE
Entête CLR
Manifeste
Clé publique
Métadonnées
Code IL
Ressources

Assembly
Signature : construction

- **Création du couple Clé Privé / Clé Publique**
  - Outil **`sn.exe`**
- **Signer l'Assembly**
  - Option **`/keyfile`** du compilateur
  - Propriétés du projet
  - Attribut AssemblyKeyFile
- **Possibilité de signature retardée**

Les types

- **Introduction et architecture**
- **Compilation et IL**
- **Le CLR**
  - Load Assemblies
  - JIT
  - AppDomain
- **Assembly**
  - Structure et définition
  - Signature
- **Les types**

CLI / CLS / CTS

CLI : Common Langage Infrastructure

- Contrainte du CLR et Assemblage (ECMA-335)

CLS : Common Langage Specification

- Contraintes imposées au langage et compilateur

CTS : Common Type System

- Caractéristiques des types connus par le CLR

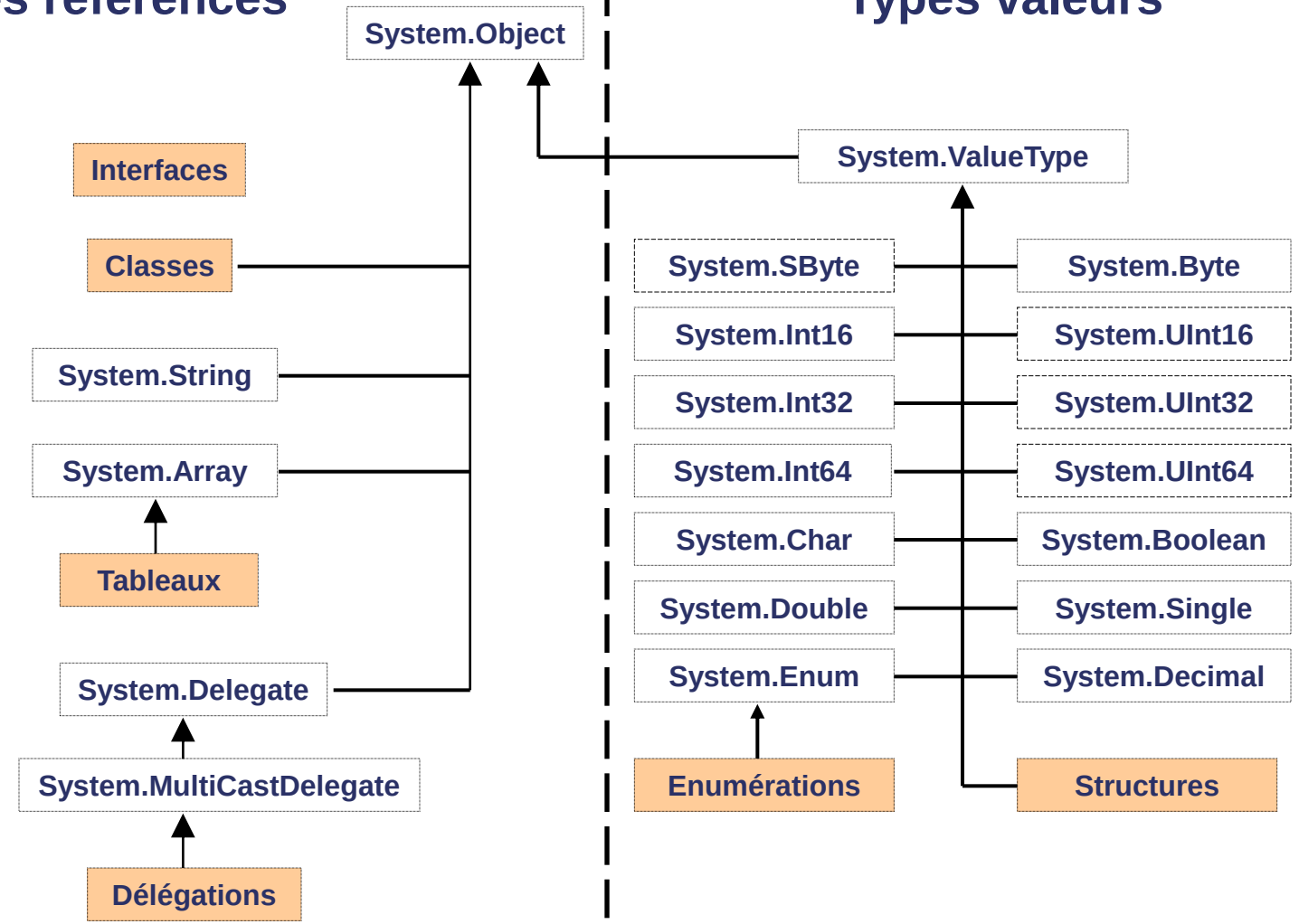
Compatibilité consommateur/extenseur

```
CLSCompliantAttribute
```

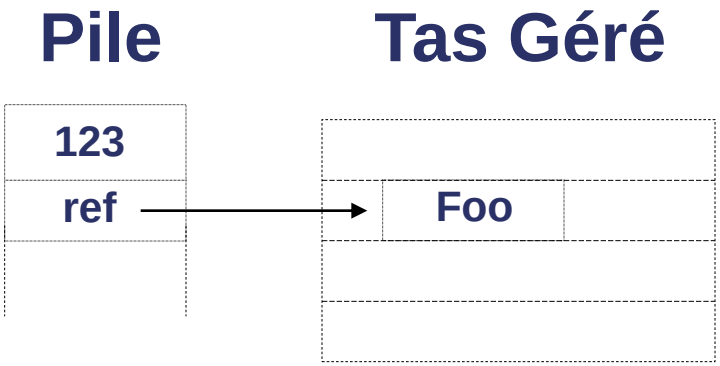
Les différents types

Types références

Types valeurs



Boxing / UnBoxing




```
{
  int i = 123;
  Foo foo = new Foo();
  ….
}
```

Type valeur

- Instanciers directement sur la pile

Type référence

- Instancié sur le tas

Boxing / Unboxing

- Passage d'un monde à l'autre

```
{
  int i = 123;
  object o = i;   // boxing
  i = (int) o;    // unboxing
}
```

System.object

object

```
Type GetType()
virtual string ToString()
virtual Finalize()

static bool ReferenceEquals(object,object)
virtual bool Equals(object)
virtual int GetHashCode()

protected object MemberwiseClone()
```

ICloneable

```
virtual public object Clone()
```

Les types anonymes

- Une ou plusieurs propriétés en lecture seule
- Obligation de passer par le mot clé var

```
var query = from proc in System.Diagnostics.Process.GetProcesses()
            orderby proc.ProcessName ascending
            select proc; // Type: System.Diagnostics.Process

foreach (var item in query)
{
    Console.WriteLine("{0}\t {1}", item.Id, item.ProcessName);
}
```

```
var query = from proc in System.Diagnostics.Process.GetProcesses()
            orderby proc.ProcessName ascending
            select new { proc.Id, proc.ProcessName };

foreach (var item in query)
{
    Console.WriteLine("{0}\t {1}", item.Id, item.ProcessName);
}
```

CLR et le monde « Non Managé »

PInvoke

- Permet à du code géré d'appeler des fonctions compilées en code natif
  - `[DllImport(« Kernel32.dll »]`
  - `static extern`

- Exemple

```
[DllImport("Kernel32.dll")]
 public static extern bool Beep(uint ifreq, uint iDuration);
 static void Main(string[] args)
 {
  bool b = Beep(100, 100);
 }
```

CLR et le monde « Non Managé »

C++ CLI

- **IJW (It Just Work) : Simplification de PInvoke**

```
#include « stdafx.h »
#include « windows.h »
int main() {
        bool b = Beep(100,100);
}
```

- **Langage qui permet de créer des assemblage contenant à la fois du code natif et du code géré.**
  - #pragma managed
  - #pragma unmanaged
  - gcroot<>

CLR et le monde « Non Managé »

COM

- **De . NET à COM**
  - tlbimp.exe : Construit une « Assembly » à partir d'un composant COM
    - Runtime Callable Wrapper
- **De COM à .NET**
  - tlbexp.exe : Encapsule une « Assembly » dans un composant COM
    - COM Callable Wrapper