Introduction à la plateforme .NET

Mnacho Echenim
Clément Huez

Pourquoi ce projet?

- Très utilisé dans les institutions financières
  - Demande des recruteurs
  - Confirmé par les anciens élèves
- Utilisé dans d'autres enseignements
  - TP Monte Carlo
  - Projet Evaluation de Produits Structurés
- **Vous ne serez pas des experts à la fin du projet**
  - Extrêmement vaste, on n'a pas le temps de tout couvrir
  - La formation à la plateforme se poursuivra au cours des autres TP et projets

LA PLATEFORME

La plateforme .NET

.NET, ce n'est pas

- Un système d'exploitation
- Un langage de programmation

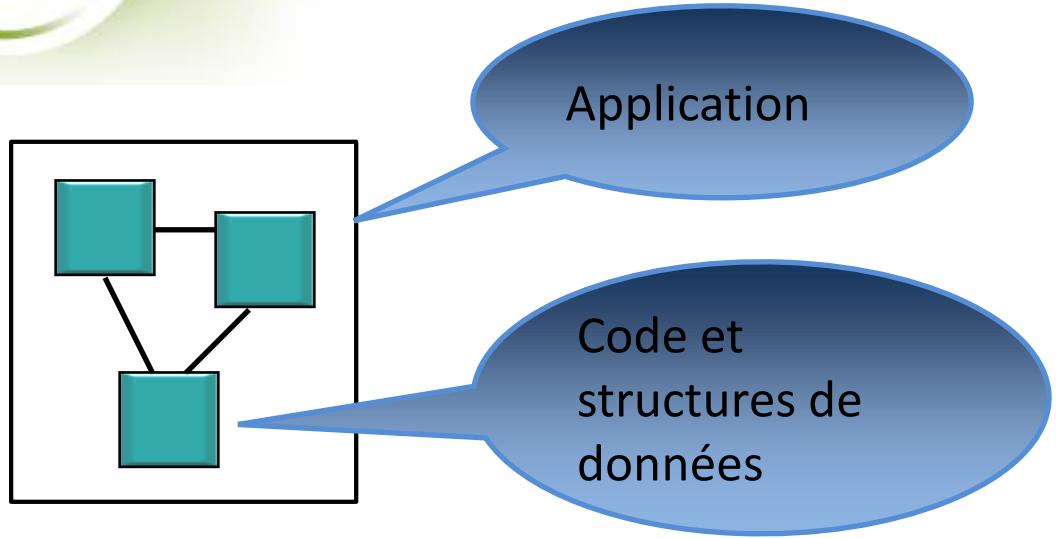
.NET, c'est

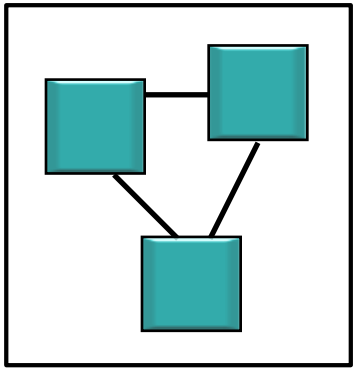
- Une plateforme de développement logiciel
- Développement simple d'applications distribuées
  - Compatibilité inter-langages
  - Facilité de communication

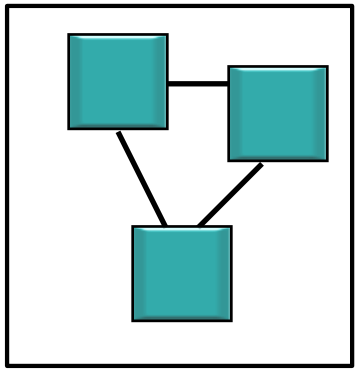
Pourquoi une nouvelle plateforme?

- Résoudre ‘l’Enfer des dlls’
- Déploiement simplifié
- Réutilisation simplifiée de code
- Intégration de nombreux langages

L'évolution vers .NET

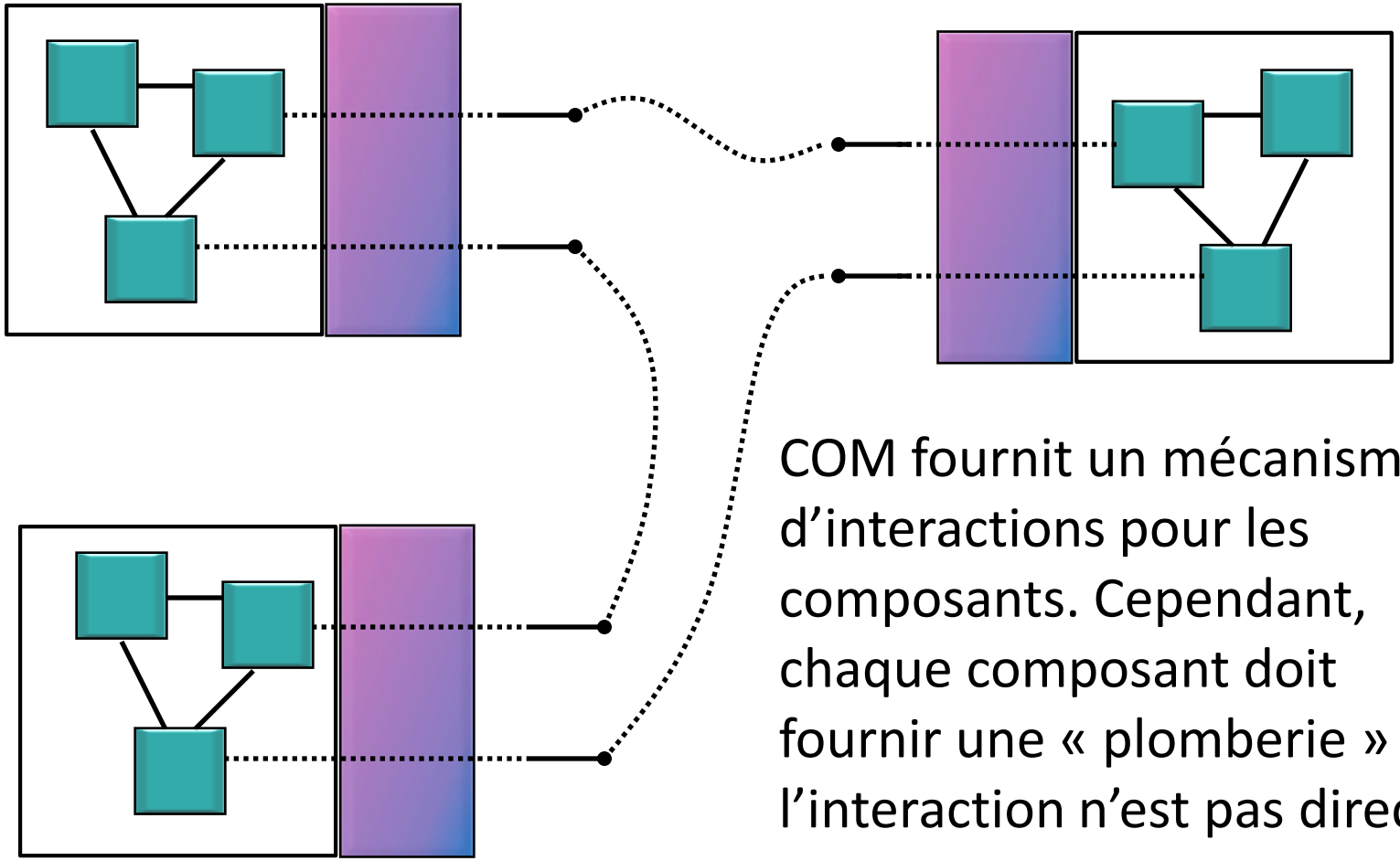






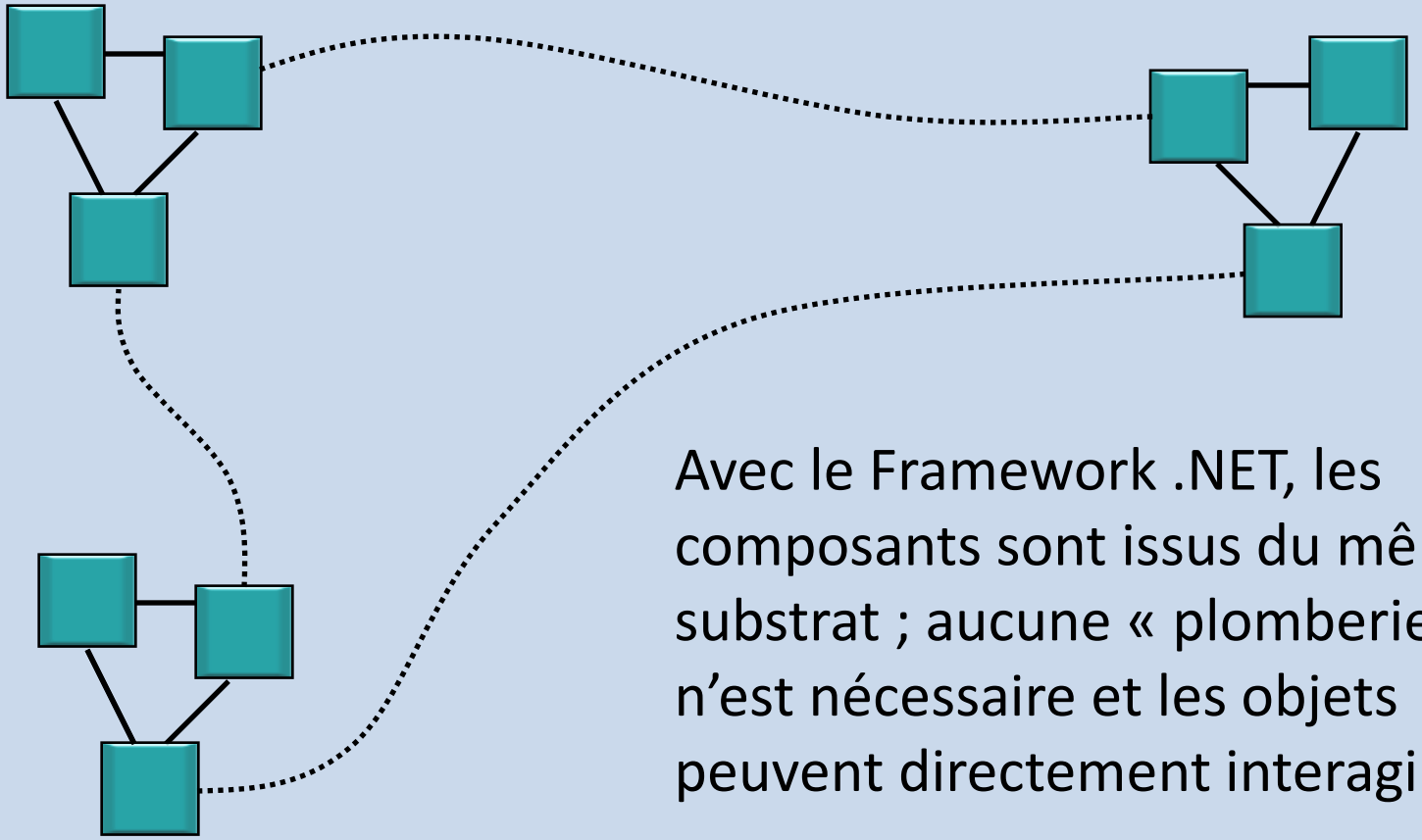
Avant COM, les applications étaient des entités complètement séparées avec peu ou pas d'interactions.

L'évolution vers .NET



COM fournit un mécanisme d'interactions pour les composants. Cependant, chaque composant doit fournir une « plomberie » et l'interaction n'est pas directe

L'évolution vers .NET



Avec le Framework .NET, les composants sont issus du même substrat ; aucune « plomberie » n'est nécessaire et les objets peuvent directement interagir.

EXEMPLE: MULTILANGAGES

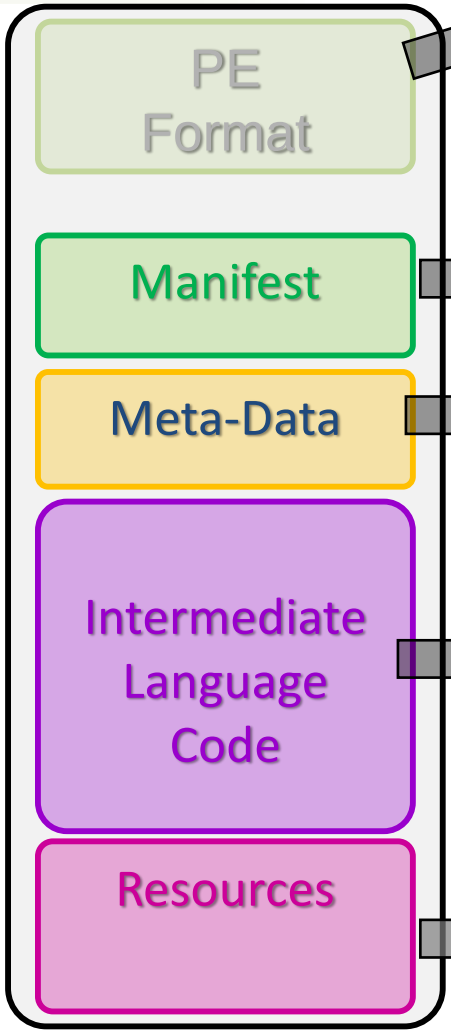
Pourquoi ça marche?

- Instrument.vb => Instrument.dll
- Option.cpp => Option.dll
- PortefeuilleOptions => PortefeuilleOptions.dll

Ces trois fichiers contiennent *exactement* le même type d'information.

Ce sont des **assemblies .NET**

Assembly (dll/exe)



File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
14C machine (x86)
3 number of sections
3E78E307 time date stamp Wed Mar 19 22:37:11 2003
0 file pointer to symbol table
0 number of symbols
E0 size of optional header
10E characteristics
Executable
Line numbers stripped
Symbols stripped

- Nom de l'assembly
- Version
- Culture
- Strong Name
- Assemblies références
- Fichiers de l'assembly

Description des types exposés (classes, méthodes, propriétés et prototypages)

```
.method public hidebysig specialname rtspecialname
      instance void  .ctor() cil managed
{
                .maxstack  2
                ldarg.0
                call       class [mscorlib]System.IO.TextWriter
[mscorlib]System.Console::get_Out()
                call       instance void Dumper.Dumper::.ctor(class
[mscorlib]System.IO.TextWriter)
                ret
} // end of method Dumper::.ctor
```

Fichiers texte, vidéos, images…

Compilation et Exécution

```
Sub Main()
    Console.WriteLine("Hello World!")
End Sub
```

Microsoft Visual Basic.NET

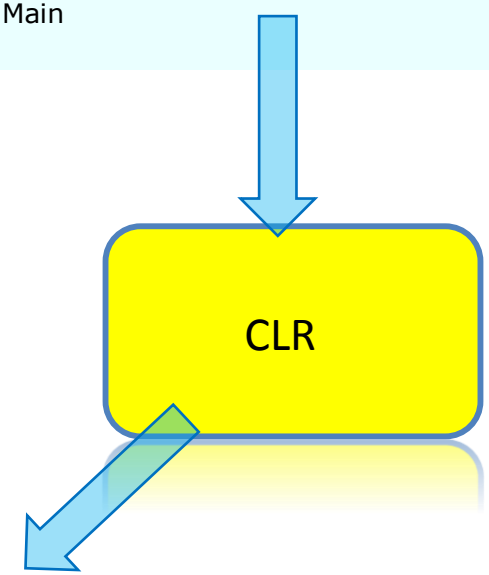
```
public static void Main()
{
System.Console.WriteLine("Hello World!");
}
```

Microsoft Visual C#.NET

```
int main(void)
{
    Console::WriteLine(S"Hello World");
    return 0;
}
```

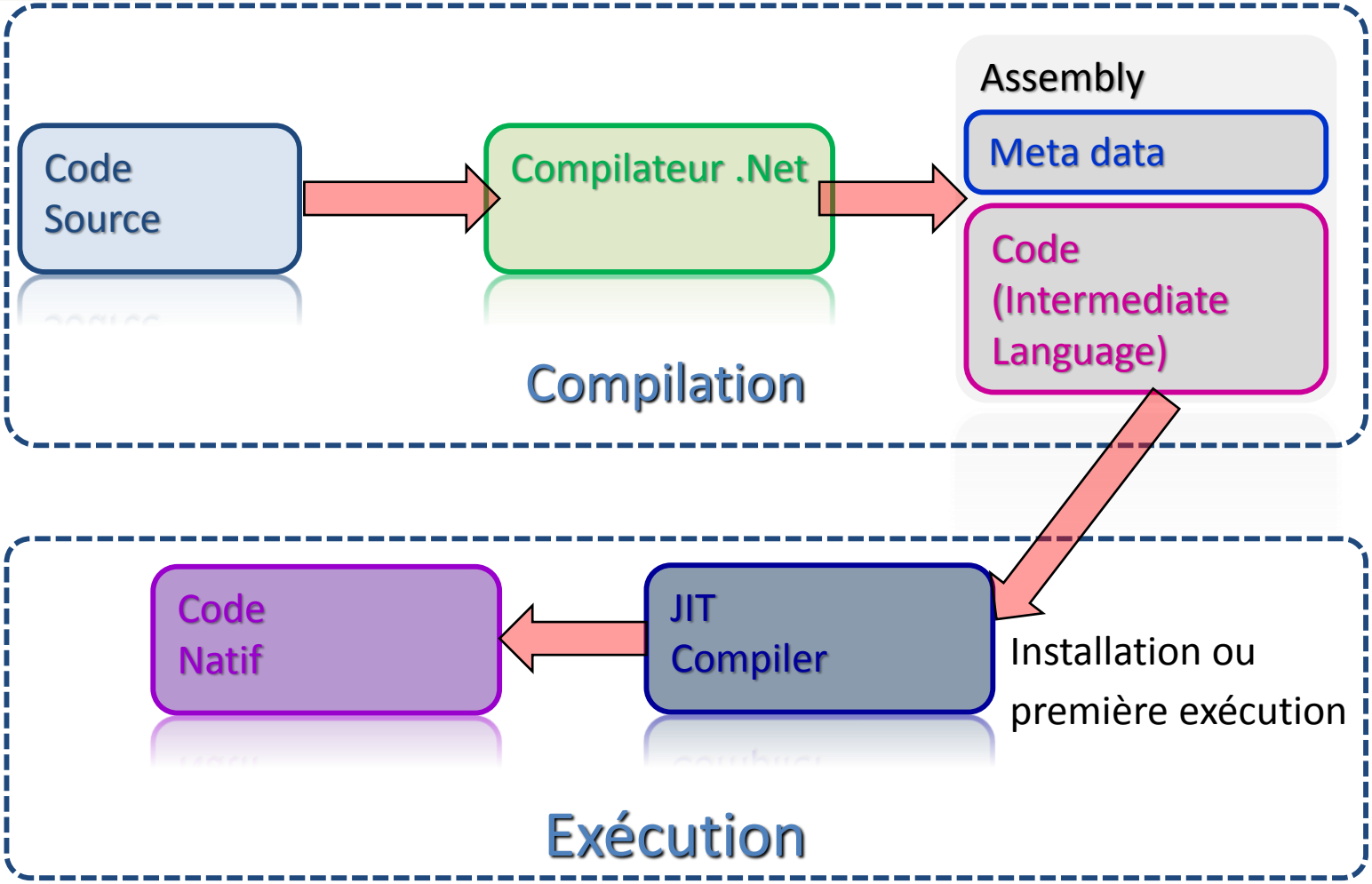
Microsoft Visual C++.NET

```
.method public hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       11 (0xb)
  .maxstack  8
  IL_0000:  ldstr      "Hello World!"
  IL_0005:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000a:  ret
} // end of method Hello::Main
```

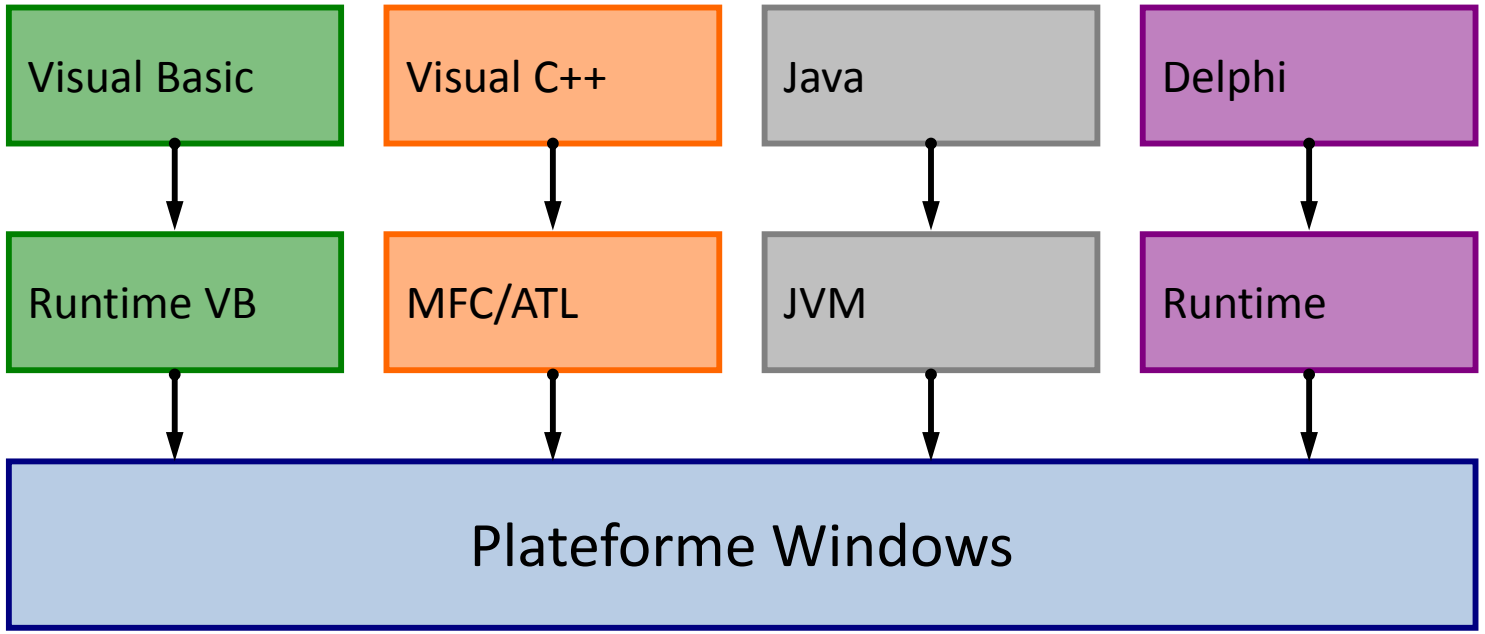



```
00000000  push        ebp
00000001  mov         ebp,esp
00000004  mov         ecx,dword ptr ds:[01BF0064h]
0000000a  call        dword ptr ds:[02E521A0h]
00000012  pop         ebp
```

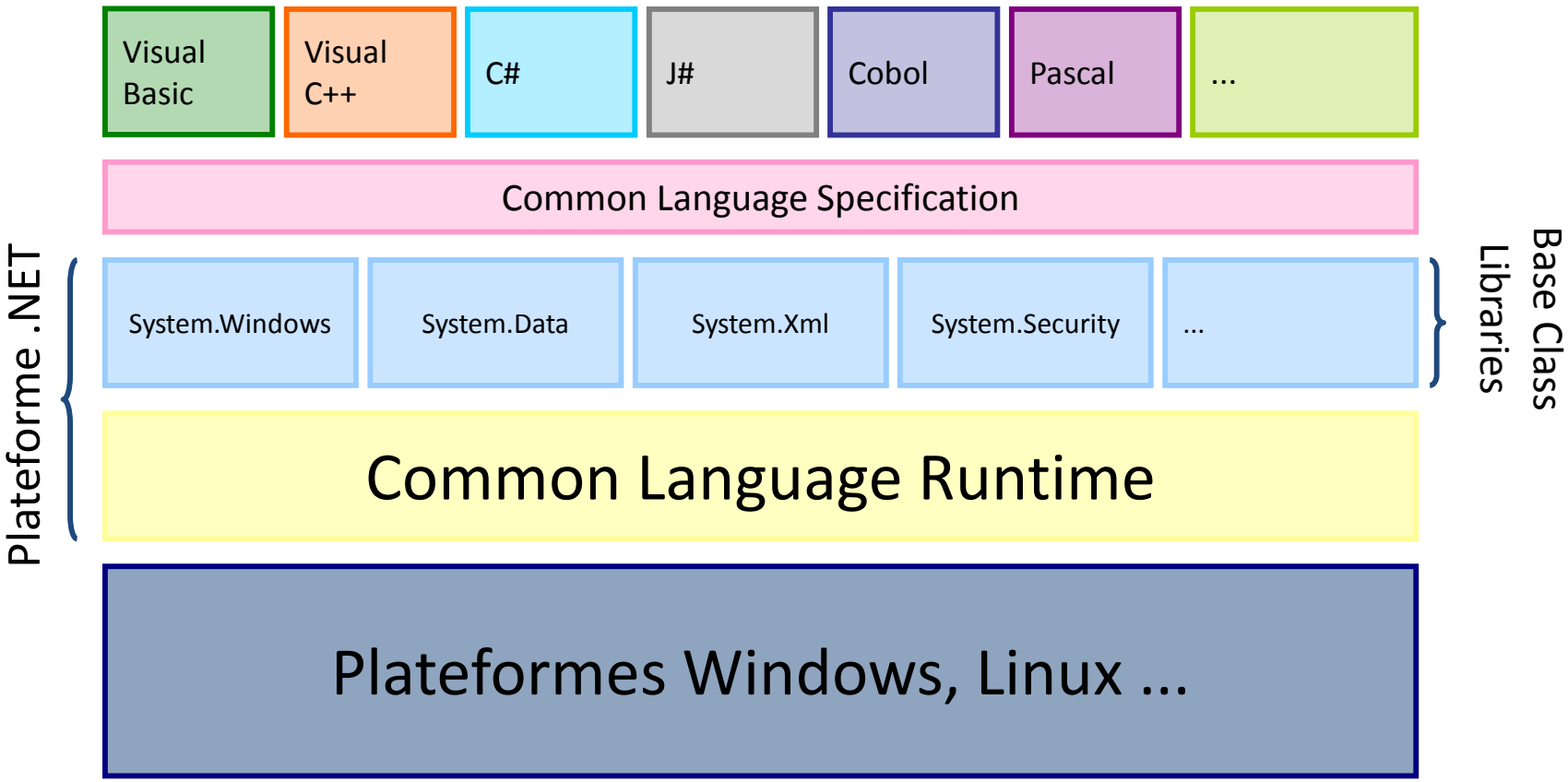
Compilation et Exécution (suite)



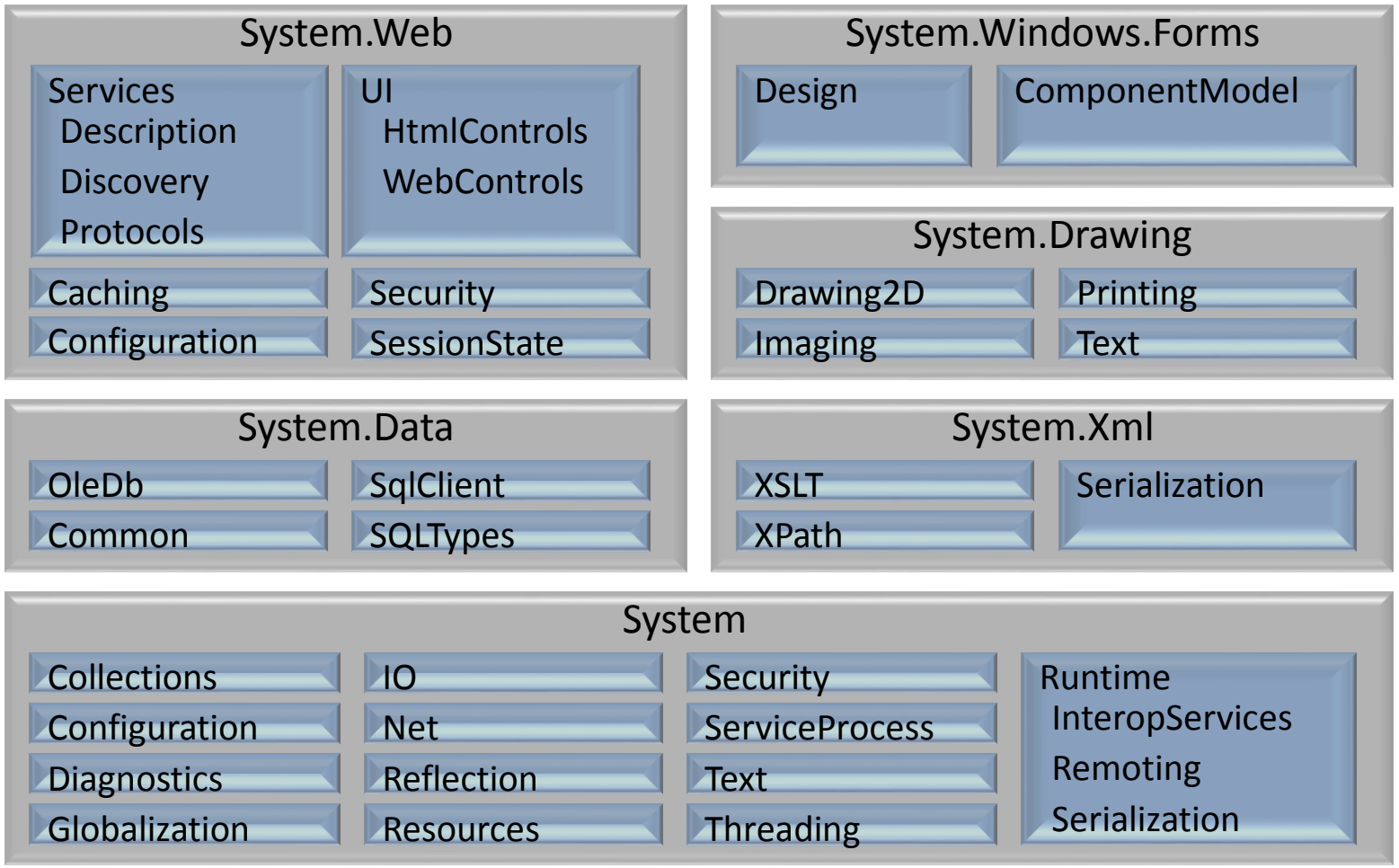
Modèle avant .Net



Modèle .NET


Visual Basic
Visual C++
C#
J#
Cobol
Pascal
...
Common Language Specification
System.Windows
System.Data
System.Xml
System.Security
...
Base Class Libraries
Plateforme .NET
Common Language Runtime
Plateformes Windows, Linux ...

Base Class Libraries



Compléments sur la CLR

- Prend du MSIL en entrée
- Compilation JIT: produit des instructions binaires qui dépendent de la plateforme, du processeur
  - Code compilé à l'exécution
  - Le code n'est pas interprété
- Quelques fonctionnalités:
  - Types unifiés
  - Debug
  - **Ramasse-miette**…

Garbage collector

- Objets alloués sur un tas *managé*
- Collection effectuée quand pointeur interne passe la fin de l'espace d'adressage
  - Le GC vérifie si certains objets du tas ne sont plus utilisés
  - Destruction non-déterministe des objets pour améliorer les performances
  - Heuristiques puissantes pour lancement du ramasse-miette
  - On peut l'appeler explicitement mais c'est fortement déconseillé.

ÉLÉMENTS DU LANGAGE C#

Structure d'un programme C#

- L'exécution d'un programme commence avec Main()
- Le mot clé using fait référence à une ressource du framework .NET
- Premier exemple

```
using System;
class HelloWorld {
        static void Main() {
                Console.WriteLine ("Hello, World");
        }
}
```

Types valués, types référence

Type valué

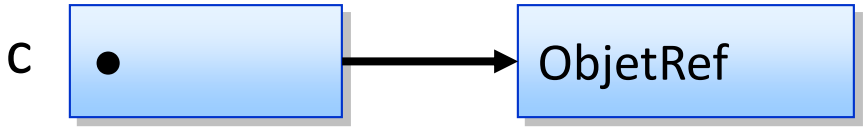
- Contient directement les données
- Stocké sur la pile
- Doit être initialisé
- Ne peut pas être nul

```
int i;
i = 42;
```

Type référence

- Contient une référence aux données
- Stocké sur le tas (managé)
- Déclaré avec le mot clé new
- Destruction gérée par le ramasse-miette

```
ObjetRef c;
```



Les espaces de nommage

- Espaces de nommage (imbriqués)

```
namespace NomDeLaCompagnie {
    namespace Ventes {
            public class Client () { }
    }
}
// Ou bien
namespace NomDeLaCompagnie.Ventes { ... }
```

- L'instruction using

```
using System;
using NomDeLaCompagnie.Ventes;
```

Accessibilité

- Mots clés utilisés pour définir le niveau d'accessibilité des membres d'une classe

Déclaration	Définition
public	L'accès n'est pas limité
private	Accès limité à la classe même
internal	Accès limité à l'Assembly
protected	Accès limité à la classe même ainsi qu'aux classes dérivées
protected internal	Accès limité à la classe, aux classes dérivées ou aux membres de l'Assembly

Classes scellées

- On ne peut dériver d'une classe scellée
- Évite que la classe ne soit redéfinie ou étendue par des tiers

```
public sealed class MaClasse {
    // membres de la classe
}
```

Type nullable

- A partir du framework 2.0
- Pour avoir des types valués de valeur nulle
- Utilisé en général avec les bases de données

```
int? nomVariable = null;

(…)

if (nomVariable.HasValue) {
        // faire quelque chose
}
```

(Classes partielles)

- Nouveauté depuis le .NET Framework 2.0
- Permet un découpage d'une classe sur plusieurs fichiers
  - Ex: Code généré par VS pour une Winform

```
partial class NomDeLaClasse {

        (…)

}
```

Les propriétés

Les propriétés sont des méthodes qui protègent l'accès aux membres d'une classe

```
private int poidsAnimal;
public int Poids {
        get {
                return poidsAnimal;
        }
        set {
                poidsAnimal = value;
        }
}
```

L'instruction *foreach*

L'instruction *foreach* permet de récupérer chaque élément d'une collection quelconque dans une variable du même type

```
int[] nombres = {4, 5, 6, 1, 2, 3, -2, -1, 0};
foreach (int nombre in nombres) {
        Console.WriteLine(nombre);
}

List<maClasse> maListe = RecupererUneListe();
foreach (maClasse element in maListe) {
      Console.WriteLine(element.texte);
}
```

Initialisateurs d'objets, de collections

Pour déclarer et instancier une variable dans la même instruction

```
List<int>  nombres  = new  List<int>  { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
maClasse  monObj = new maClasse() {nom = « Nom », age = 18};
List<maClasse>  maListe  =  new  List<maClasse>() {
        new maClasse() {nom = « Alice », age = 5},
        new maClasse() {nom = « Bob », age = 8}
};
```

Le mot-clé *var*

- Inférence de type: le compilateur détecte automatiquement le type de la variable
- **Nb:** la variable doit être déclarée et instanciée dans la même instruction

```
var  i = 5; // le compilateur détecte un entier
var  s = « Hello world »; // chaîne de caractères
var  a = new[] {1, 2, 3}; // tableau d'entiers
```

Les types anonymes

- Pour définir une classe « temporaire »
  - Encapsulation d'un ensemble de données
  - Aucune méthode, gestion d'évènement n'est nécessaire
  - Utilisation commune: traitement de données (BD, XML…)
- Définition d'un type anonyme:

```
var premierTypeAnonyme  =  new {Nom = « premierObjet »,  numero = 1};
var deuxiemeTypeAnonyme  =  new {numero = 1, Nom = « premierObjet »};
// les objets ne sont pas identiques

var  autreTypeAnonyme  =  new {Date = new DateTime(2010, 2, 3), Valeur = 23.5};
```

Délégués

- Pour invoquer une méthode qu'on ne connaît pas au moment de la compilation
  - Gestion d'évènements
  - Critères de recherche
- Très proches des pointeurs de fonctions

```
int[] tabEntiers = new int[] {1, 5, 3, 6, 10, 12, 13, 15 };
static  bool  estInferieurADix(int i) {
        return i <= 10;
}

static  void  Main(string[] args){
        int[] infDix = Array.FindAll(tabEntiers,  estInferieurADix);
        // renvoie [1, 5, 3, 6, 10]
}
```

Méthodes anonymes, lambda expressions

- Pour rendre le code plus lisible
- Compilateur s'occupe de (presque) tout

```
int[] tabEntiers = new int[] {1, 5, 3, 6, 10, 12, 13, 15 };


static  void  Main(string[] args){
        int[] multTrois = Array.FindAll(tabEntiers, delegate (int i) {return i%3 == 0;});
        // renvoie [3, 6, 12, 15]

        int[] multCinq = Array.FindAll(tabEntiers, i => (i % 5 == 0));
        // renvoie [5, 10, 15]
}
```

L'essentiel

- Facilité d'installation
- Compatibilité inter-langages
- La plateforme s'occupe de tout
  - Base class library
  - Ramasse-miette
  - Ne pas réinventer la roue
  - « Privilégier la propreté sur l'efficacité »