

CLI e modelo de componentes baseado em IL e Metadata

Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

F. Miguel Carvalho (mcarvalho@cc.isel.ipl.pt)

Agenda

- O que é a Plataforma .Net?
- Critérios de desenho: Portabilidade
- Critérios de desenho: Interoperabilidade
- Critérios de desenho: Serviços
- Critérios de desenho: Funcionalidades
- O que é a Plataforma .Net? (revisitado)

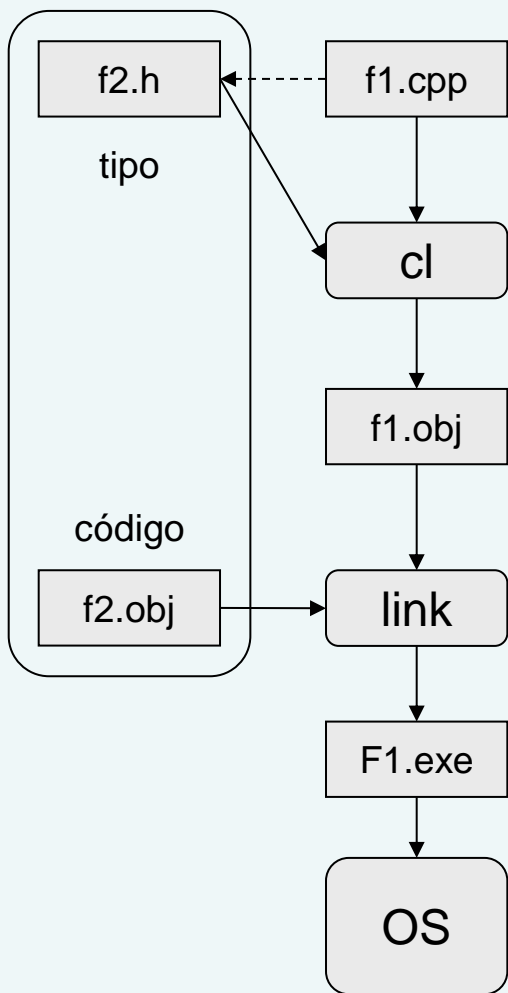
O que é a Plataforma .NET?

- Plataforma “moderna” de desenvolvimento e execução.
 - Também identificado como modelo managed, por ser um **ambiente de execução controlado** (*managed runtime environment*).
 - Em contraste com o modelo unmanaged do C ou C++.
- Suporte à construção de software por componentes:
 - Componente: pode ser visto como uma unidade (ou peça) de software reutilizável (como “legos”).
- Visa Potenciar o aumento:
 - da **produtividade** do programador
 - Maior parte do tempo de desenvolvimento de aplicações em C++ é dedicado à correcção de bugs de gestão de memória.
 - da **robustez e reutilização** do software produzido.

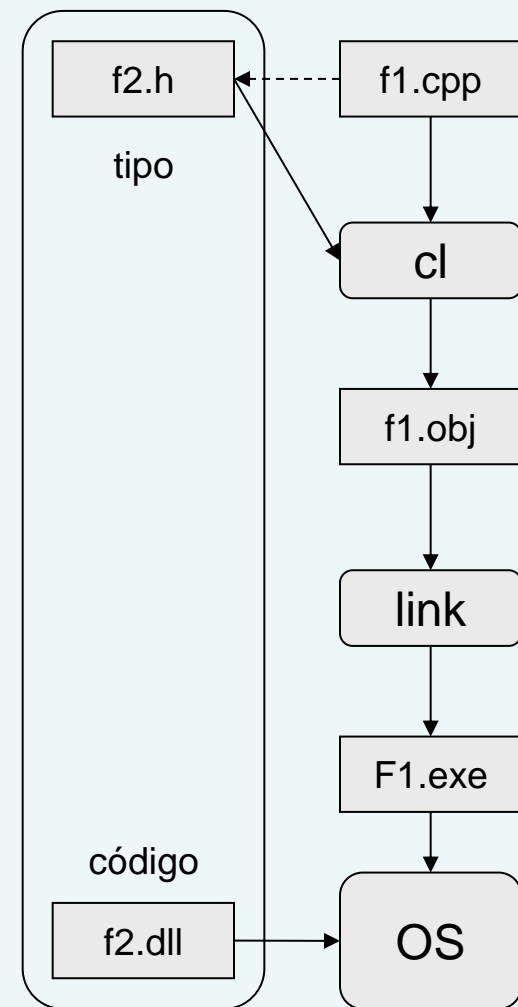
Modelos “unmanaged”

<versus> “managed”

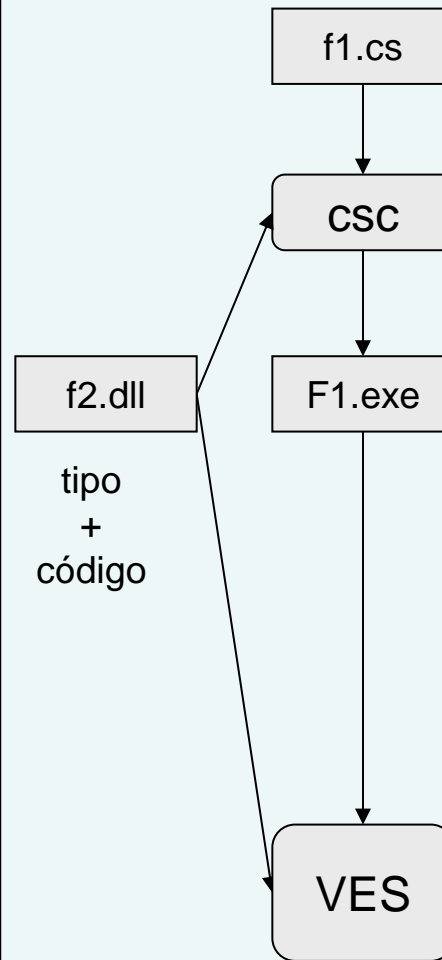
Ligação estática:



Ligação dinâmica:



Ligação dinâmica:



Algumas tecnologias baseadas em componentes

- 1986: Objective-C by Brad Cox
- 1990: System Object Model (SOM) da IBM
- 1990: Object Linking and Embedding (OLE) da Microsoft
- 1993: Component Object Model (COM) da Microsoft
- 1996: Java Development Kit (JDK) 1.0.

Introduz componentes **auto descritos**.

O programador é obrigado a descrever o componente em IDL, ou numa TLB, ou .h.

Dificuldades:

- **Não existe um vínculo** entre o **componente** e o seu contracto (IDL ou TLB).
- **Componentes em código nativo** compatíveis com uma arquitectura.
- Diferentes compiladores podem usar diferentes convenções de chamada por omissão.

Critérios de desenho (requisitos)

- Portabilidade;
- Interoperabilidade:
 - entre linguagens;
 - entre módulos de *software* (serviços e aplicações).
- Serviços:
 - Gestão automática de memória (***garbage collection***);
 - Segurança – ***type safety***, controle de acessos e isolamento;
 - Ligação dinâmica;
 - Exceções;
 - AppDomains.
 - ...
- Funcionalidades:
 - IO, contentores, *networking*, entre outras

Componente

- Uma das ideias chave para a concretização dos requisitos anteriores é a existência de **componentes**:
 - **auto descritos – metadata**
 - Independente da linguagem de programação usada no desenvolvimento do componente (C#, C++, etc).
 - Substitui os ficheiros *header* – .h – do modelo *unmanaged* ou IDLs e TLB do COM e DCOM.
 - **instruções em linguagem intermédia**
 - **CIL** – *common intermediate language*;
 - Independente da arquitectura do processador;
 - Traduzida em tempo de execução.

➔ A plataforma Java foi a primeira a concretizar a ideia de:
Componente = IL + metadata (indissociáveis)

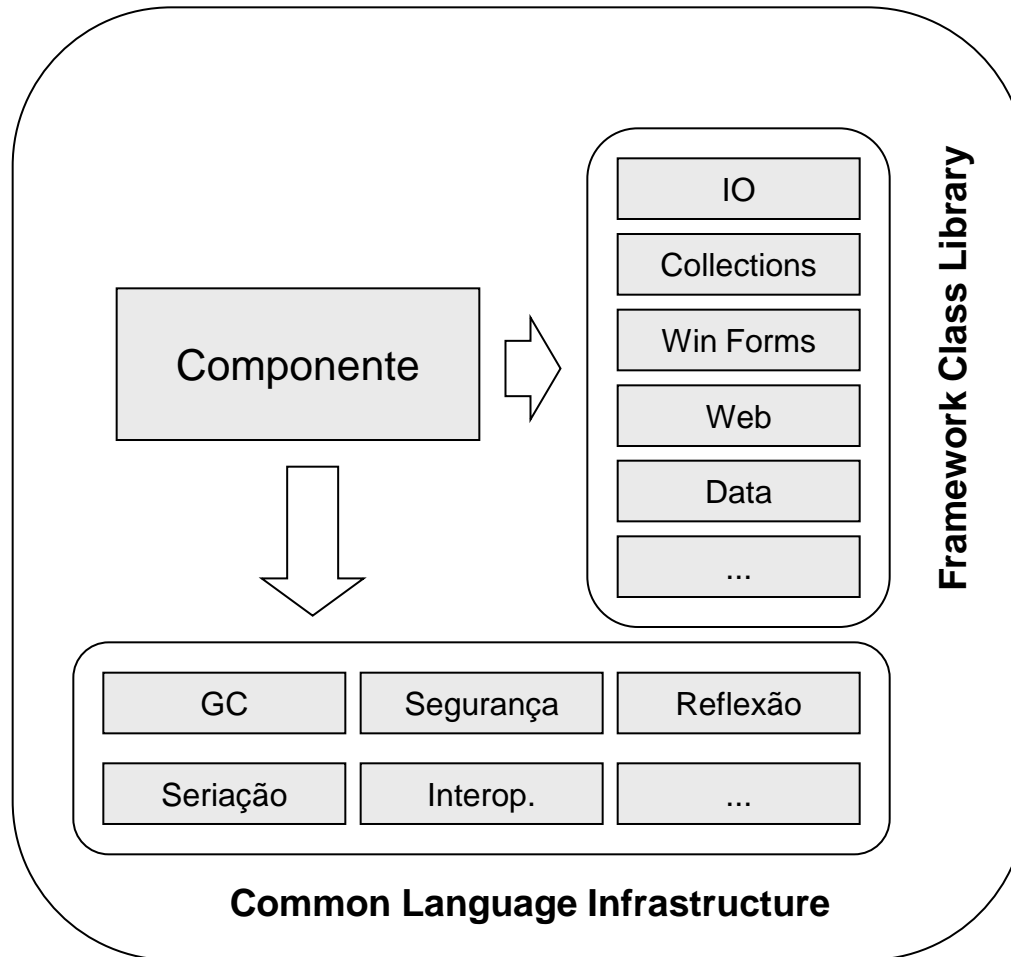
IL = *intermediate language*. Em Java designado por **bytecodes**.

➔ A plataforma .Net introduziu a possibilidade de estender a metadata através de anotações (***custom attributes***).

Componente...

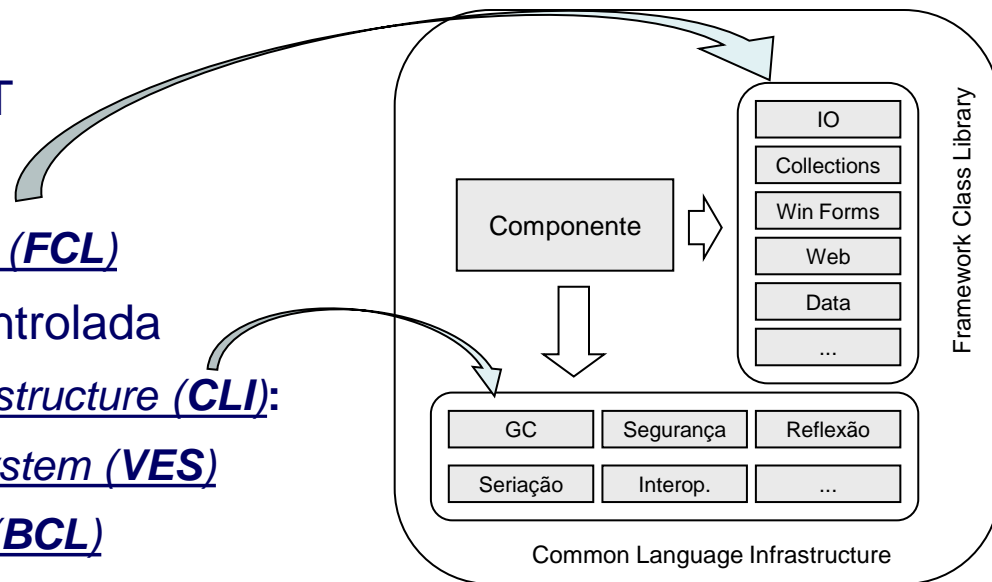
- Fisicamente um componente .Net pode ser um ficheiro .exe ou uma .dll, designados por **módulos**.
- Um **módulo** pode não ser um **componente – assembly .net**.
 - ➔ E.g .netmodule
 - ➔ Por simplificação estes não serão tratados.

O que é a Plataforma .NET? ...



O que é a Plataforma .NET? ...

- Elementos da Plataforma .NET
 - Biblioteca de classes
 - Framework Class Library (FCL)
 - Ambiente de execução controlada
 - Common Language Infrastructure (CLI):
 - Virtual Execution System (VES)
 - Base Class Library (BCL)
 - Novas linguagens de programação
 - Por exemplo: C# e VB.NET
 - Ferramentas de apoio à criação (desenvolvimento, verificação, instalação e manutenção) de *software*
 - Por exemplo: Visual Studio



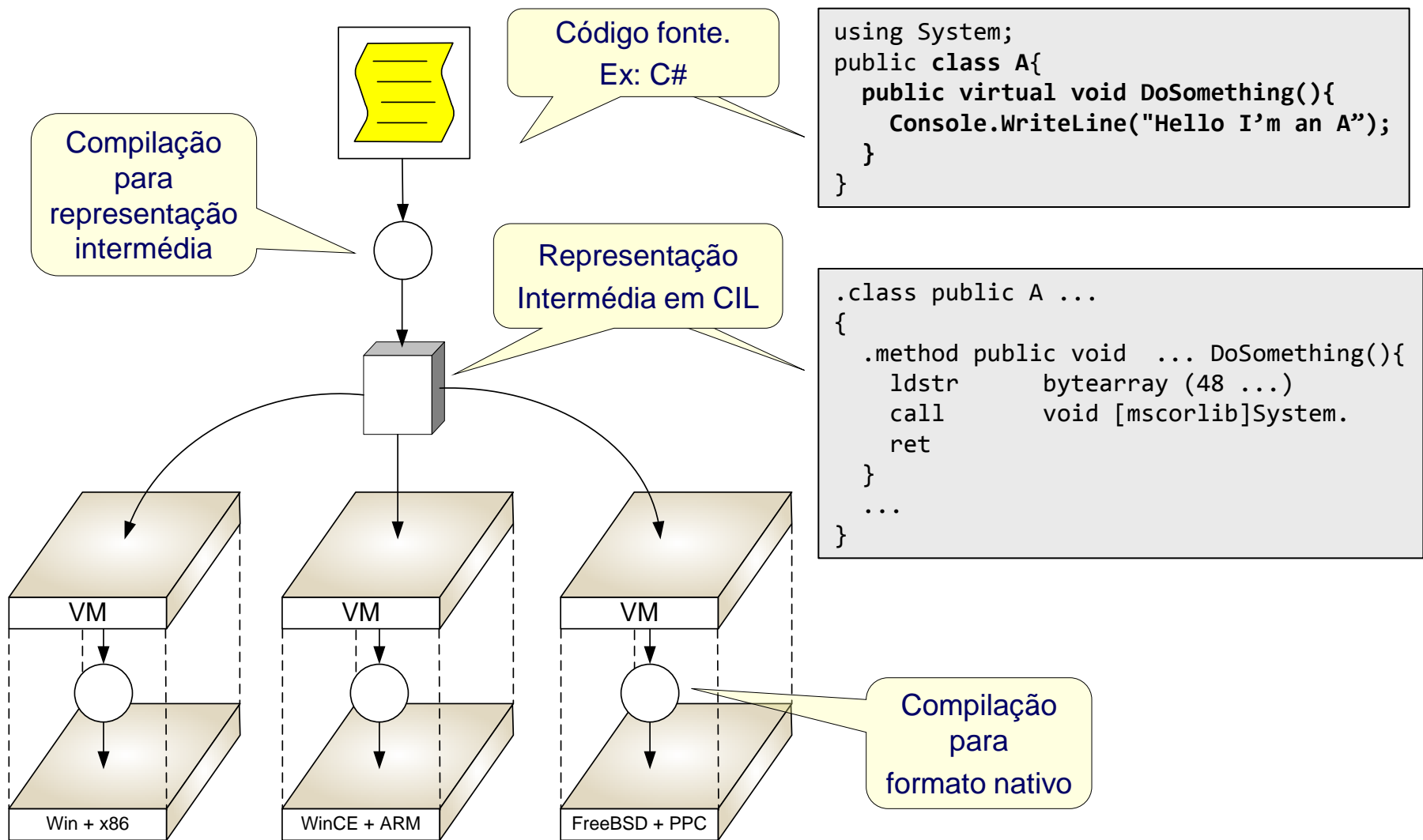
Critérios de desenho (requisitos)... Portabilidade

- Portabilidade;
- Interoperabilidade:
 - entre linguagens;
 - entre módulos de *software* (serviços e aplicações).
- Serviços:
 - Gestão automática de memória (***garbage collection***);
 - Segurança – ***type safety***, controle de acessos e isolamento;
 - Ligação dinâmica;
 - Exceções;
 - AppDomains;
 - ...
- Funcionalidades:
 - IO, contentores, *networking*, entre outras

Critérios de desenho (requisitos)... Portabilidade

- A que se refere?
 - Capacidade de utilização da mesma peça de software em arquitecturas diversas:
 - Arquitectura refere-se ao par SO / Hardware;
 - Exemplos: Windows + x86, Windows CE + ARM, FreeBSD + PowerPC, ...
- Solução - compilação a 2 níveis:
 - Linguagem de alto nível é compilada para uma linguagem intermédia
 - representação **eficiente** do programa (CIL + metadata);
 - representação **independente** da arquitectura.
 - Na máquina de execução final:
 - **interpretador** executa instruções expressas em representação intermédia,ou
 - **compilador** just-in-time (“Jitter”) traduz a representação intermédia para formato nativo.

Compilação a dois níveis

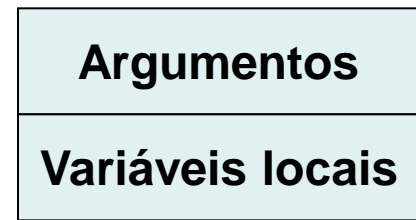


Linguagem intermédia - Estado de execução

Estado de execução:

- **Registo de activação (activation record)**
 - Criado antes da chamada à função (activação) e destruído uma vez terminada a sua execução
 - Composto pelas colecções numeradas de argumentos e de variáveis locais
- **Stack de avaliação (evaluation stack)**
 - Estrutura de dados sobre a qual é realizada a sequência de operações do corpo da função
 - As operações consomem os operandos do topo do stack e produzem o resultado para o topo do stack

Activation Record



Evaluation Stack

top

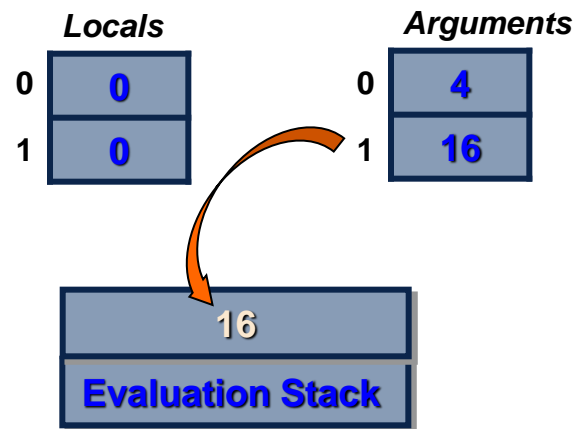


Linguagem intermédia ...

- Instruções com prefixo **ld** fazem **push** no *stack* da variável, argumento ou campo passado por parâmetro.

- ldloc – variável local
- ldarg – argumento da função
- ldfld - campo

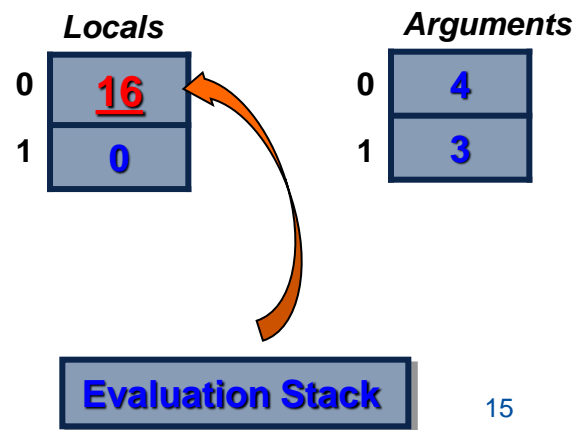
Ex: ldarg.1



- Instruções com prefixo **st** fazem **pop** de um valor do *stack* e armazenam-no na variável, argumento ou campo passado por parâmetro.

- stloc – variável local
- starg – argumento da função
- stfld - campo

Ex: stloc.0



Exemplo

C#

```
static double Modulo(int x, int y){  
    int x2 = x * x;  
    int y2 = y * y;  
    return Math.Sqrt(x2 + y2);  
}
```

CIL

```
.method private hidebysig static float64  
    Modulo(int32 x, int32 y) cil managed  
{  
    .locals init (int32 V_0, int32 V_1)  
    IL_0000: ldarg.0  
    IL_0001: ldarg.0  
    IL_0002: mul  
    IL_0003: stloc.0  
    IL_0004: ldarg.1  
    IL_0005: ldarg.1  
    IL_0006: mul  
    IL_0007: stloc.1  
    IL_0008: ldloc.0  
    IL_0009: ldloc.1  
    IL_000a: add  
    IL_000b: conv.r8  
    IL_000c: call        float64 [mscorlib]System.Math::Sqrt(float64)  
    IL_0011: ret  
}
```


Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V 0, int32 V 1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0	0
1	0

Arguments

0	4
1	3

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

0

1

0

Arguments

0

4

1

3

4

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){
    Point p = new Point(4, 3);
    double res = Modulo(p.x, p.y);
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;
int y2 = y * y;
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

0

1

0

Arguments

0

4

1

3

4

4

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

0

1

0

Arguments

0

4

1

3

16

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

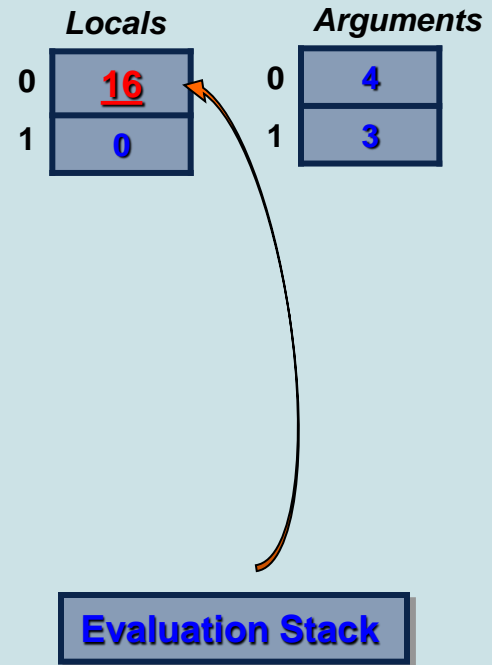
IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret



Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0	16
1	0

Arguments

0	4
1	3

3

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0	16
1	0

Arguments

0	4
1	3

3
3
Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0	16
1	0

Arguments

0	4
1	3

9

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

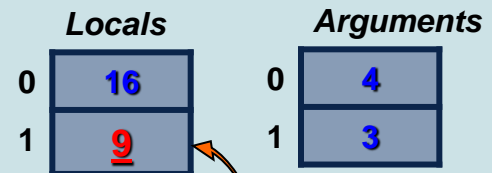
IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret



Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

16

1

9

Arguments

0

4

1

3

16

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0	16
1	9

Arguments

0	4
1	3

9
16
Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

16

1

9

Arguments

0

4

1

3

25

Evaluation Stack

Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

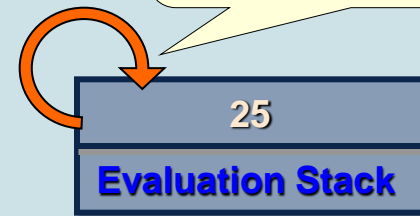
IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals		Arguments	
0	16	0	4
1	9	1	3

Convertido de inteiro com
sinal de 4 bytes, a real
(vírgula flutuante) de 8 bytes



Exemplo... Estado de Execução

```
static void Main(){  
    Point p = new Point(4, 3);  
    double res = Modulo(p.x, p.y);  
}
```

```
static double Modulo(int x, int y)
```

```
int x2 = x * x;  
int y2 = y * y;  
return Math.Sqrt(x2 + y2);
```

.locals init (int32 V_0, int32 V_1)

IL_0000: ldarg.0

IL_0001: ldarg.0

IL_0002: mul

IL_0003: stloc.0

IL_0004: ldarg.1

IL_0005: ldarg.1

IL_0006: mul

IL_0007: stloc.1

IL_0008: ldloc.0

IL_0009: ldloc.1

IL_000a: add

IL_000b: conv.r8

IL_000c: call float64 [mscorlib]System.Math::Sqrt(float64)

IL_0011: ret

Locals

0

16

1

9

Arguments

0

4

1

3

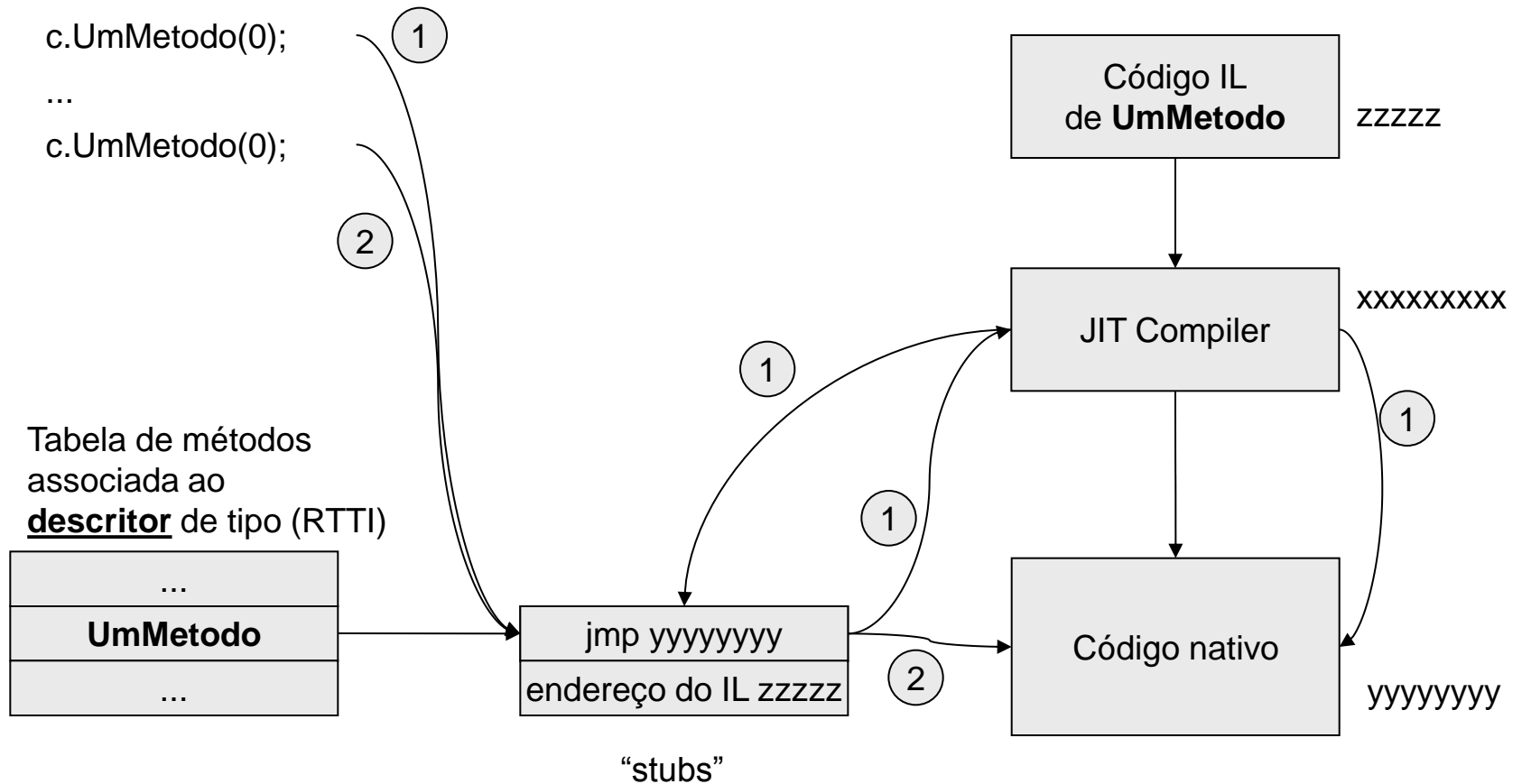
5

Evaluation Stack

Código intermédio...

- Inclusão de instruções para o suporte ao paradigma da orientação aos objectos
 - Noção de campo de objecto
 - **ldfld** e **stfld**
 - Chamada a métodos
 - **call** e **callvirt**
 - Criação e inicialização de instâncias
 - **newobj** e **initobj**
 - *Casting*
 - **castclass**, **isinst**
 - Excepções
 - **throw**, **rethrow**

Geração de código *just-in-time*



Compilação *just-in-time* (JIT)

- Desvantagens:
 - Peso computacional adicional para a geração do código nativo;
 - Memória necessária para a descrição intermédia e código nativo.
- Vantagens:
 - Optimização para o processador nativo;
 - Melhoria da localidade do código;
 - Oportunidade para realização de optimizações baseadas em estatísticas de execução do código.

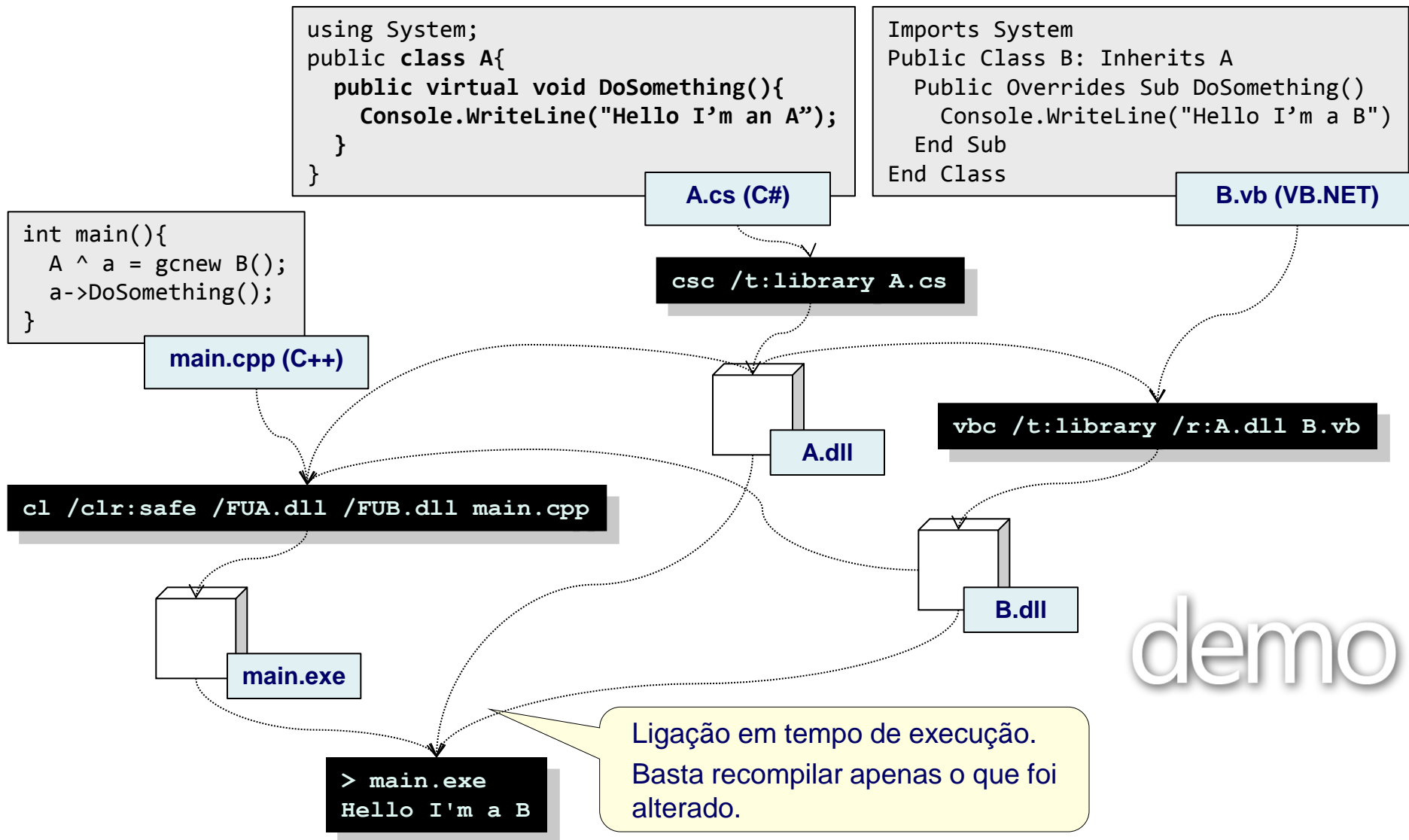
Critérios de desenho (requisitos)... Interoperabilidade

- Critérios de desenho (requisitos)
 - Portabilidade;
 - Interoperabilidade:
 - entre linguagens;
 - entre módulos de *software* (serviços e aplicações).
 - Serviços:
 - Gestão automática de memória (***garbage collection***);
 - Segurança – ***type safety***, controle de acessos e isolamento;
 - Ligação dinâmica;
 - Exceções;
 - AppDomains;
 - ...
 - Funcionalidades:
 - IO, contentores, *networking*, entre outras

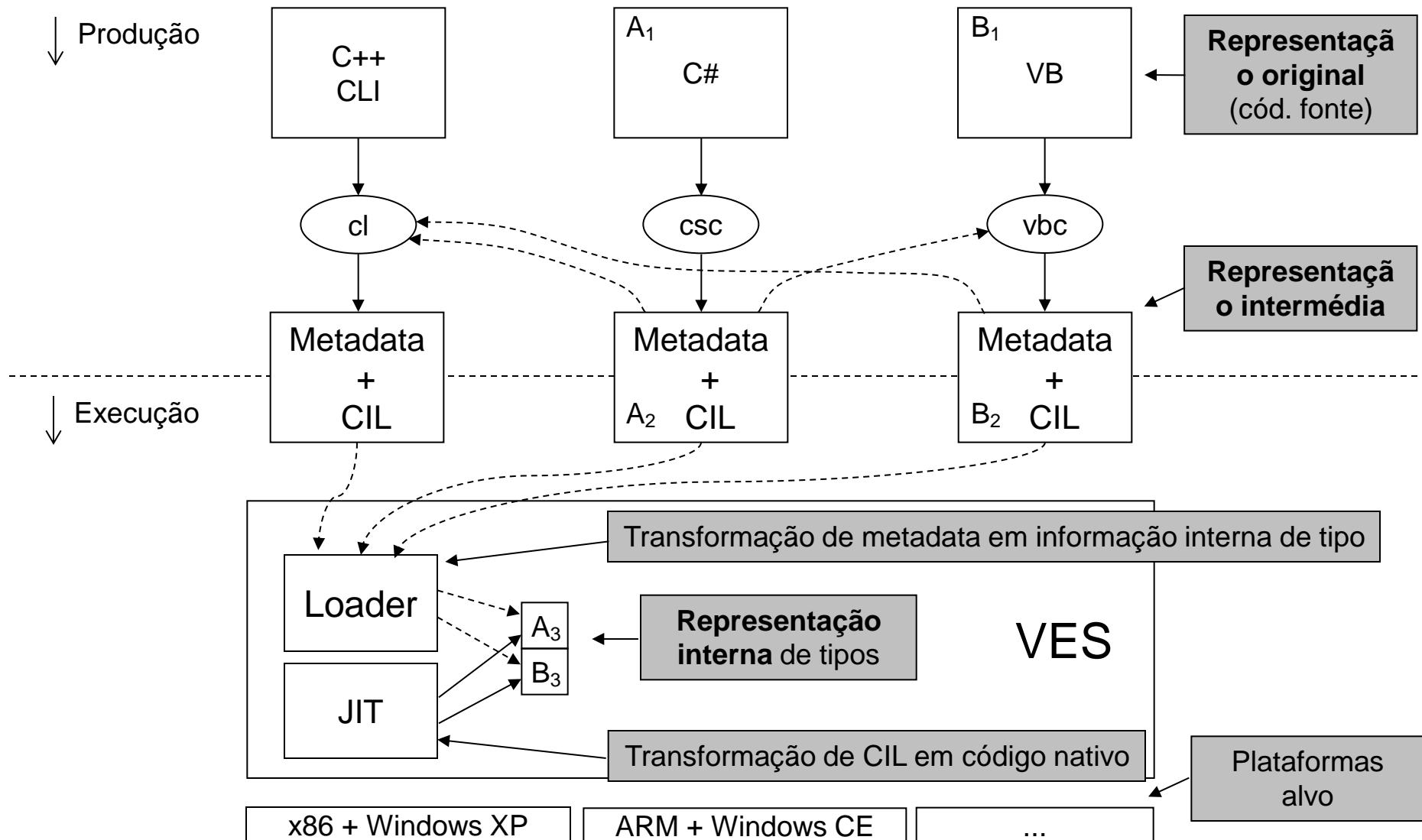
Critérios de desenho (requisitos)... Interoperabilidade

- Utilização de componentes *software* realizados em linguagens diferentes.
- A interoperabilidade não se resume à chamada de “funções” escritas em linguagens diferentes.
- Em modelos orientados aos objectos, são desejáveis outras formas de interoperabilidade
 - Exemplo:
 - Realizar a classe A na linguagem C#;
 - Realizar a classe B que deriva de A, na linguagem VB;
 - Utilizar a classe B na linguagem C++.
- Representação intermédia com *sistema de tipos* independente da linguagem.

Interoperabilidade: exemplo



Produção e execução: 3 representações



Requisito: Interoperabilidade entre linguagens

- A especificação da representação intermédia é a oportunidade para resolver o problema!
 - Esta oportunidade não foi aproveitada originalmente na Plataforma Java
 - Actualmente já várias linguagens compilam para Java, e.g.: Scala, Clojure, Ruby, etc.
- A representação intermédia deve ter capacidade de representar TODAS as construções possíveis em TODAS as linguagens!
- Na prática a representação intermédia suporta a MAIORIA das construções próprias da MAIORIA das linguagens
 - Procedimentais
 - Funcionais
 - Orientadas por objectos
- O CLI inclui os elementos necessários:
 - Sistema de tipos comum (CTS) e *metadata* para descrição dos dados
 - *Common Intermediate Language* (CIL) para descrição de comportamento

Critérios de desenho (requisitos)... Serviços

- Portabilidade;
- Interoperabilidade:
 - entre linguagens;
 - entre módulos de *software* (serviços e aplicações).
- Serviços:
 - Gestão automática de memória (***garbage collection***);
 - Segurança – ***type safety***, controle de acessos e isolamento;
 - Ligação dinâmica;
 - Exceções;
 - AppDomains;
 - ...
- Funcionalidades:
 - IO, contentores, *networking*, entre outras

Requisitos: serviços

- Conjunto de serviços disponíveis em tempo de execução
- **Exemplos:**
 - Ligação dinâmica de componentes;
 - Compilação *just-in-time*;
 - *Garbage collection*: recolha automática de objectos não utilizados;
 - Propagação de excepções;
 - Segurança
 - Verificação da *type safety* em tempo de execução,
 - garante ligação segura entre componentes;
 - Controle de acesso baseado na identidade do utilizador ou do código;
 - Isolamento entre componentes.
- Informação de tipo em tempo de execução
 - Necessária para a implementação destes serviços;
 - Presente na representação intermédia.

demo

Requisitos: serviços ... *Application domains*

- A execução é realizada no contexto de *application domains* – AppDomain:
 - Uma só instância do VES por processo;
 - Vários AppDomain por processo;
 - Isolamento entre diferentes aplicações em execução no mesmo processo – possível devido ao modelo *managed*.
- Utilização – isolamento
 - *Internet Explorer* - isolamento entre *assemblies* provenientes de diferentes origens;
 - ASP.NET – isolamento entre diferentes aplicações;
 - *Plugins* – isolamento entre aplicações e *plugins*.
- Um objecto existe num só AppDomain
- Comunicação
 - Por valor – cópia do conteúdo dos objectos (seriação);
 - Por referência – cópia da *referência* e criação de objectos procuradores (*proxies*).

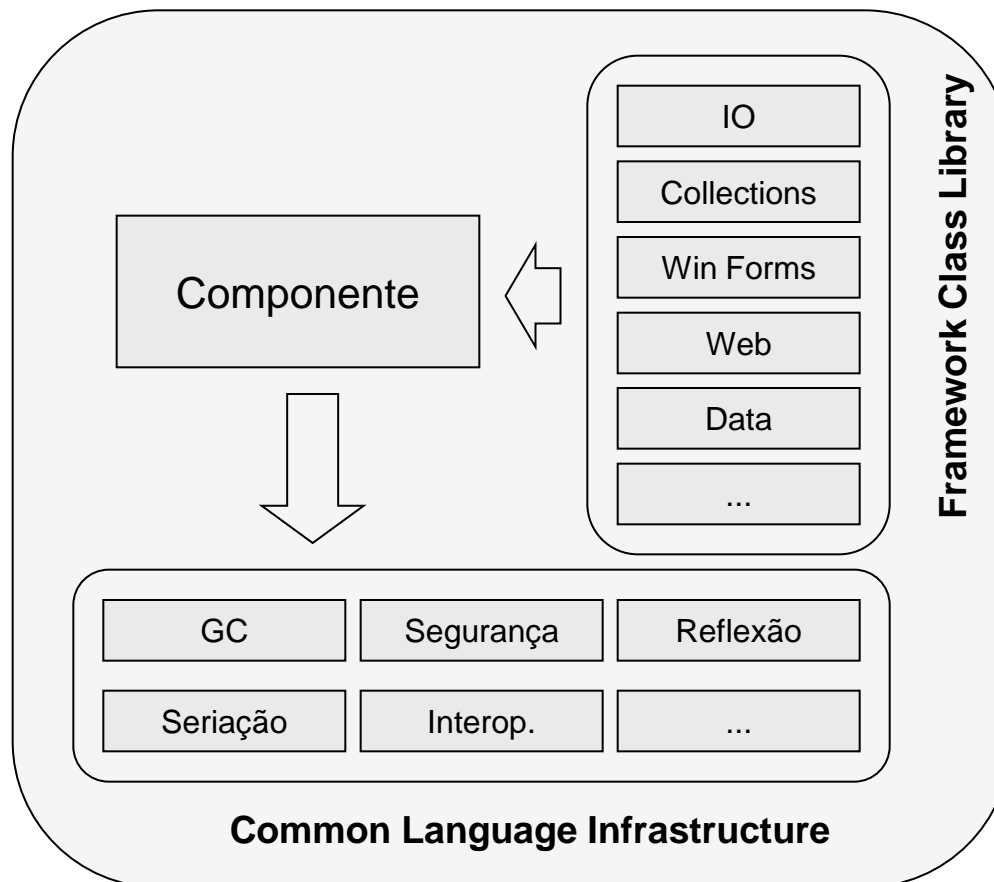
Critérios de desenho (requisitos)... Funcionalidades

- Portabilidade;
- Interoperabilidade:
 - entre linguagens;
 - entre módulos de *software* (serviços e aplicações).
- Serviços:
 - Gestão automática de memória (***garbage collection***);
 - Segurança – ***type safety***, controle de acessos e isolamento;
 - Ligação dinâmica;
 - Exceções;
 - AppDomains;
 - ...
- Funcionalidades:
 - IO, contentores, *networking*, entre outras

Arquitetura da Plataforma .NET

- Funcionalidades na BCL e FCL;
- CLI normaliza parte da FCL

Ex.: bibliotecas não normalizadas: Windows Forms, ASP.NET, WCF, WF, WPF, ...

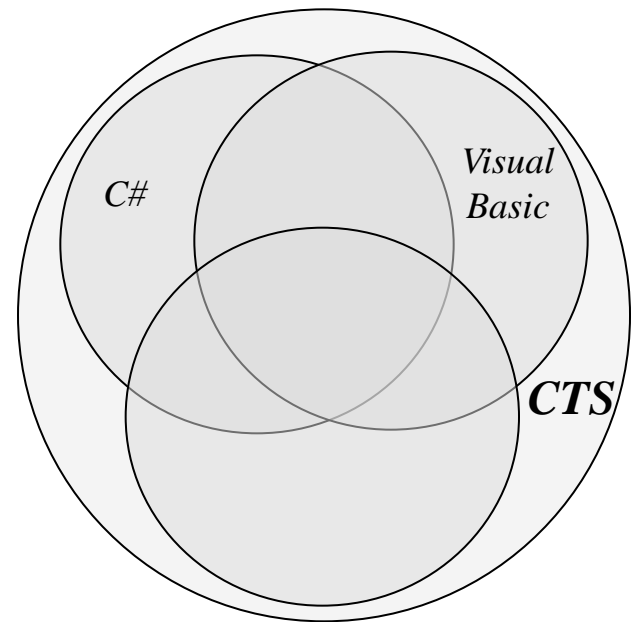


Agenda

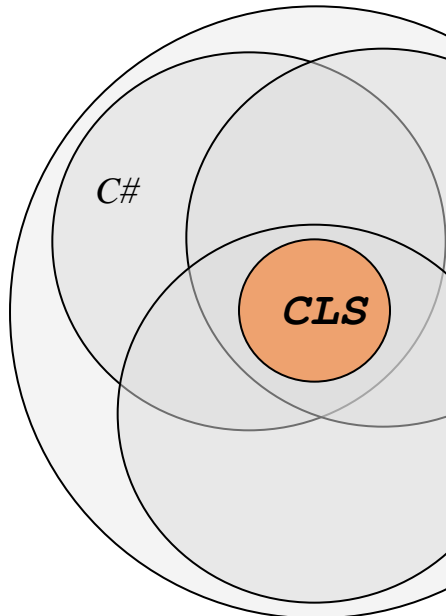
- O que é a Plataforma .Net?
- Critérios de desenho: Portabilidade
- Critérios de desenho: Interoperabilidade
- Critérios de desenho: Serviços
- Critérios de desenho: Funcionalidades
- O que é a Plataforma .Net? (revisitado)

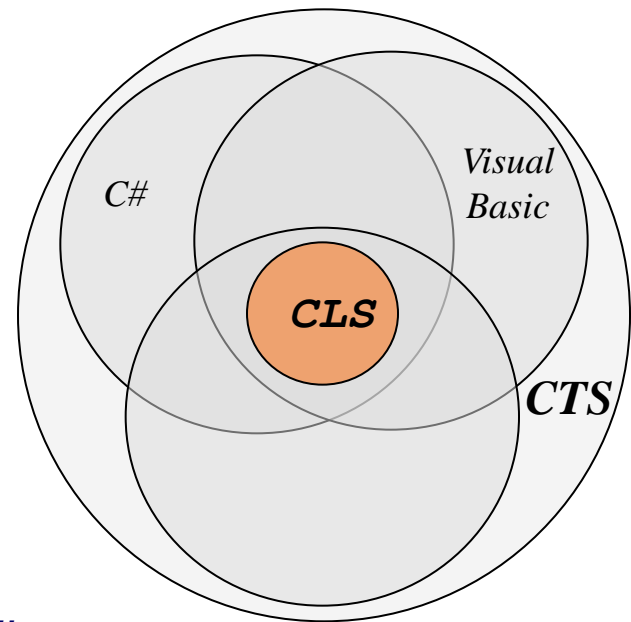
Common Type System (CTS)

- Permite representar tipos de várias linguagens
 - “União” dos sistemas de tipos das linguagens a suportar
- Cada componente contém a descrição completa dos seus tipos
 - formato de descrição: **metadata**
- *Common Type System (CTS)* define:
 - hierarquia de categorias de tipos;
 - conjunto de tipos intrínsecos;
 - construção e definição de novos tipos e respectivos membros;
 - utilização de tipos.

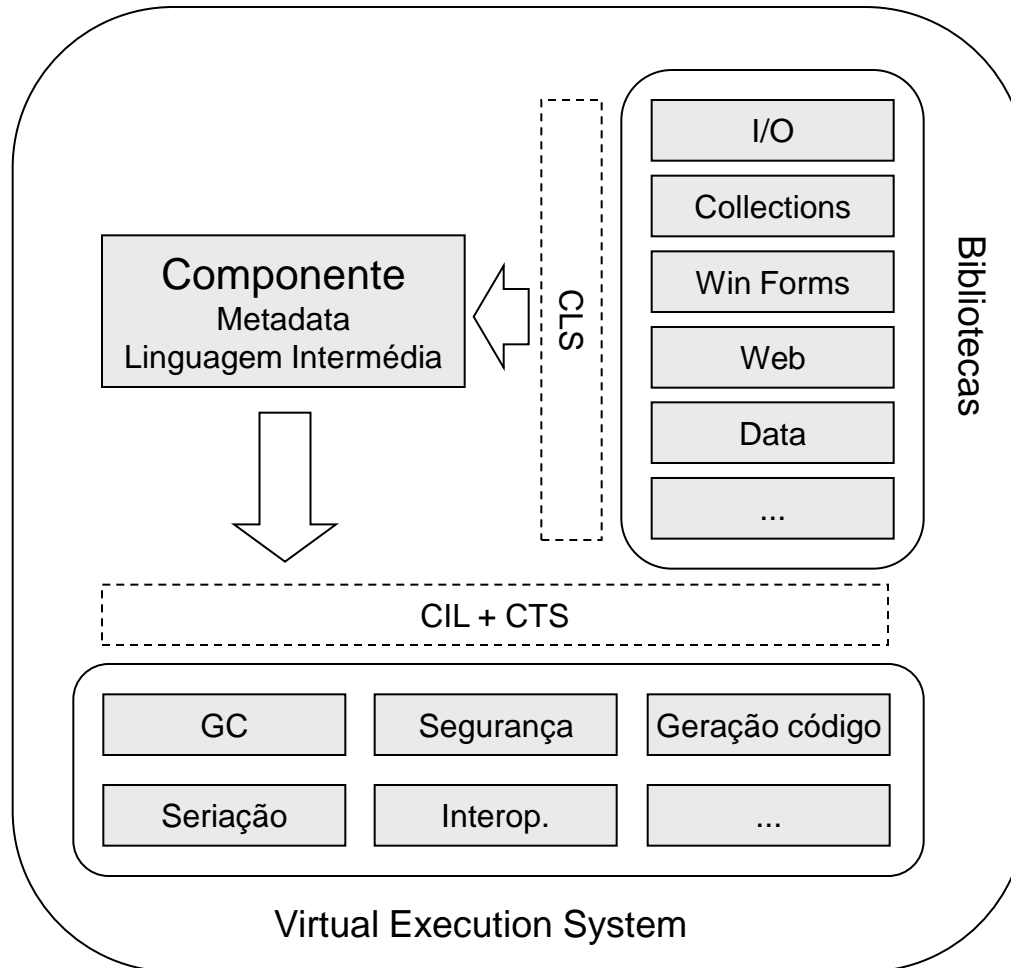


Common Language Specification (CLS)

- Definições de tipos suportados pelas várias linguagens
 - “Intersecção” dos sistemas de tipos das linguagens a suportar
 - *Common Language Specification*
 - subconjunto do CTS;
 - conjunto mínimo de características do CTS a suportar pelos clientes dos componentes;
 - restrições no desenho de interfaces de componentes.
 - Componentes *CLS compliant*
 - aspectos visíveis de um componente (a sua interface) devem respeitar a CLS quando se pretende que este possa ser usado a partir de linguagens diferentes;
 - Bibliotecas normalizadas são *CLS compliant*.
- 



Constituintes da arquitectura



Constituintes da arquitectura

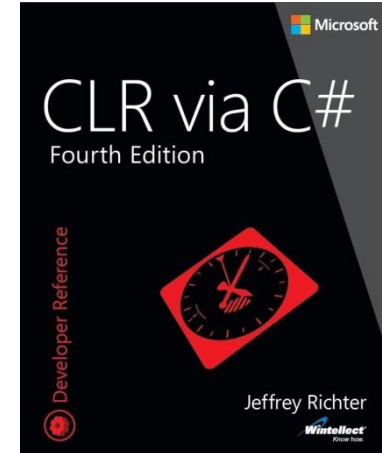
- *Common Language Infrastructure* (CLI) – normas ECMA-335 e ISO/IEC 23271:2006
- Representação intermédia, independente da linguagem fonte
 - *Common Intermediate Language* (CIL) - Representação intermédia para o código
 - *Common Type System* (CTS) – Especificação de como os tipos são definidos e se comportam.
 - *Common Language Specification* (CLS) – Subconjunto do CTS
- *Virtual Execution System* (VES) - Ambiente de execução virtual
 - Implementações
 - *Common Language Runtime* (CLR) – Microsoft
 - *Shared Source CLI* (“Rotor”) – Microsoft
 - Mono – Mono Project
- Bibliotecas de classes

Comparação com a plataforma Java

- Sistema de tipos
 - Java – desenhado para a linguagem Java. Actualmente suporta várias linguagens de alto nível.
 - .NET (CTS) – suporte para diferentes linguagens e paradigmas.
- Diferenças do sistema de tipos .Net para o Java:
 - Tipos valor compostos;
 - Passagem de parâmetros por valor e por referência;
 - *Delegates*;
 - Genéricos – suporte em tempo de execução <vs> compilação;
 - EIMI – *Explicit interface methods implementation*;
 - Anotações extensíveis (Java inclui a partir da versão 1.5).
 - Expressões Lambda – no .Net 3.0 e no Java 1.8 (Março de 2014).
- Código intermédio
 - Suporte para o CTS (ex. tipos valor, genéricos)
 - Não foi desenhado para interpretação (ex. instruções polimorfias)
 - *Stack* abstracto

Bibliografia recomendada

Jeffrey Richter, “CLR via C#, Second Edition”,
Microsoft Press; 4nd edition, 2012



Don Box, “Essential .NET, Volume I:
The Common Language Runtime ”,
Addison-Wesley Professional; 1st edition, 2002

