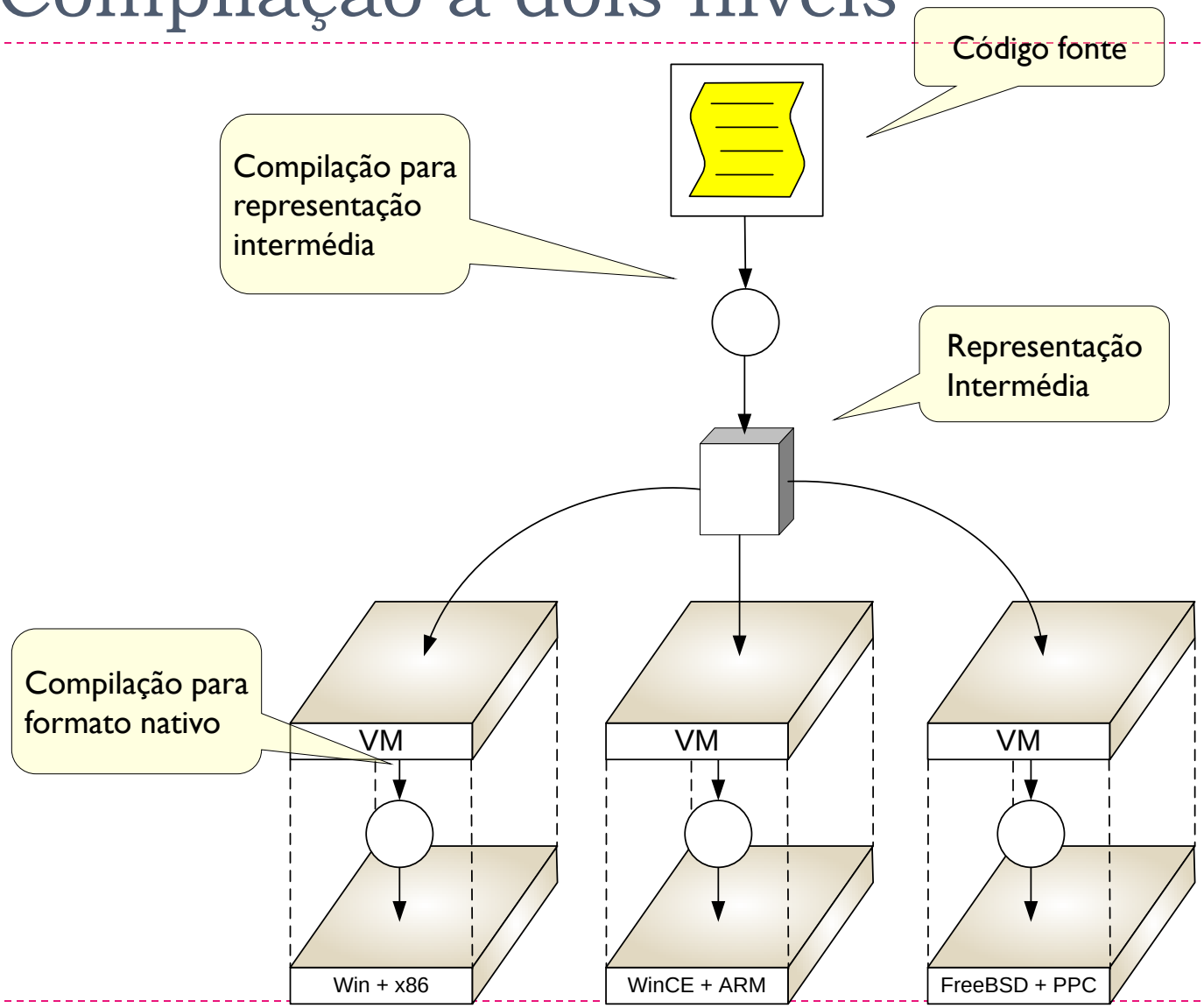
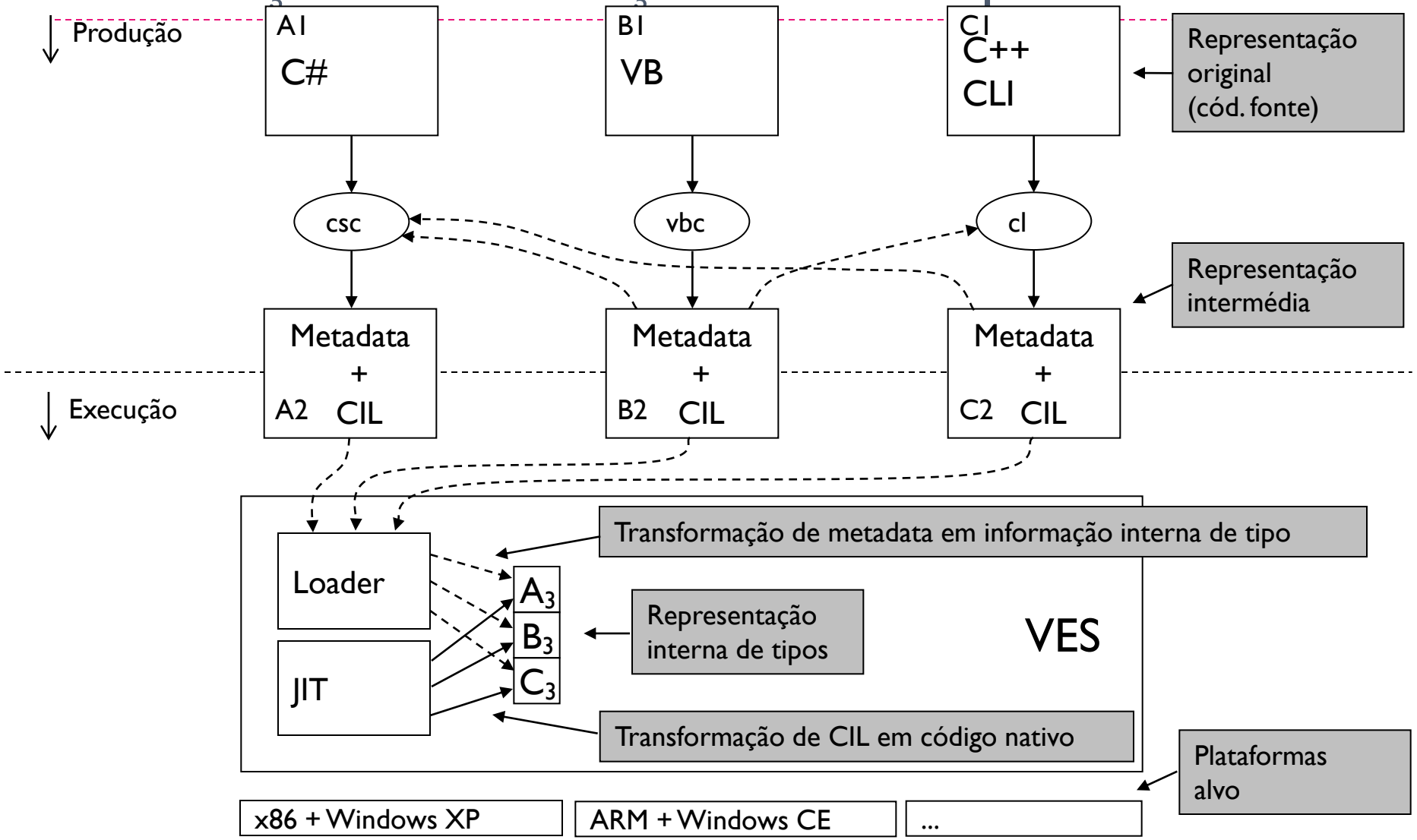
Ambientes Virtuais de Execução

O modelo de execução do CLR (**continuação**)

Compilação a dois níveis



Produção e execução de componentes



Virtual Execution System (VES)

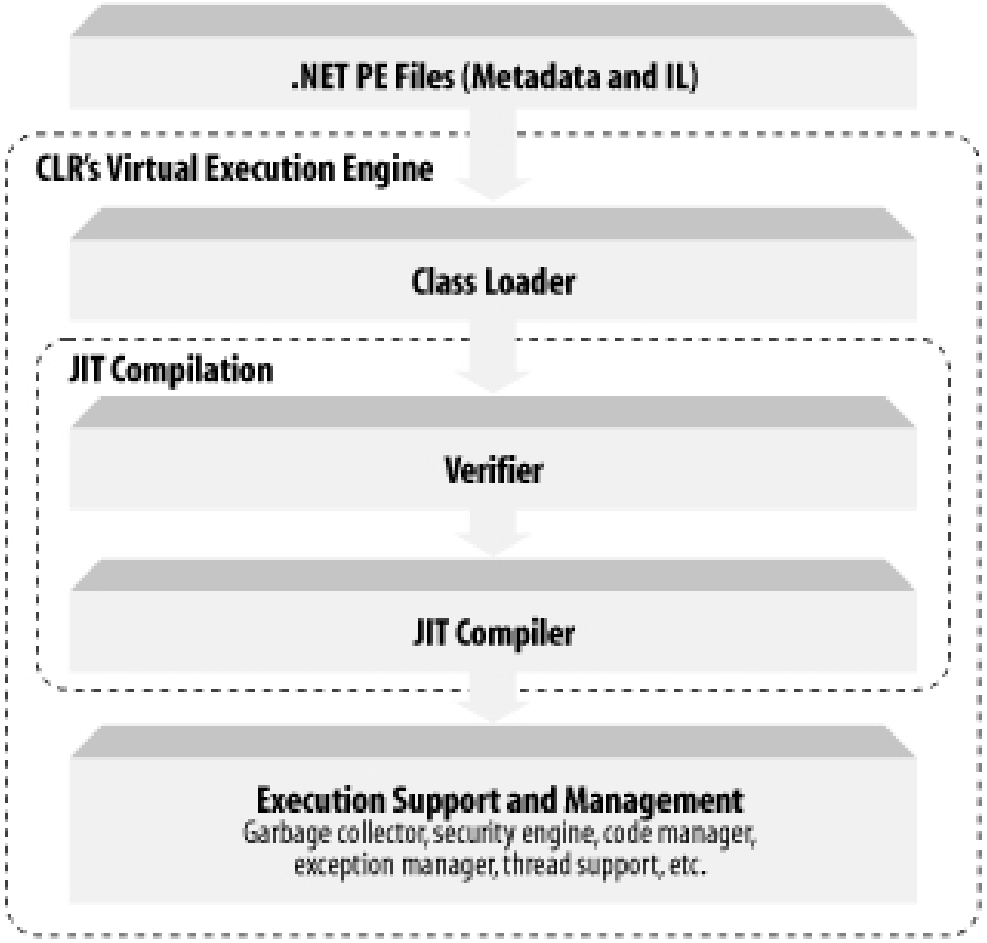


Fig: As componentes mais importantes do CLR: o sistema virtual de execução. Imagem adaptada do livro *.Net Framework Essencials*

Loader

- O *Loader* é responsável por armazenar e inicializar *assemblies*, recursos e tipos;
  - Armazenar os tipos implica armazenar o *assembly* e o módulo que contém a definição dos tipos.
- Utiliza um política de armazenar tipos (e *assemblies* e módulos) *on demand*
  - Apenas são armazenados quando são necessários na execução.

Loader

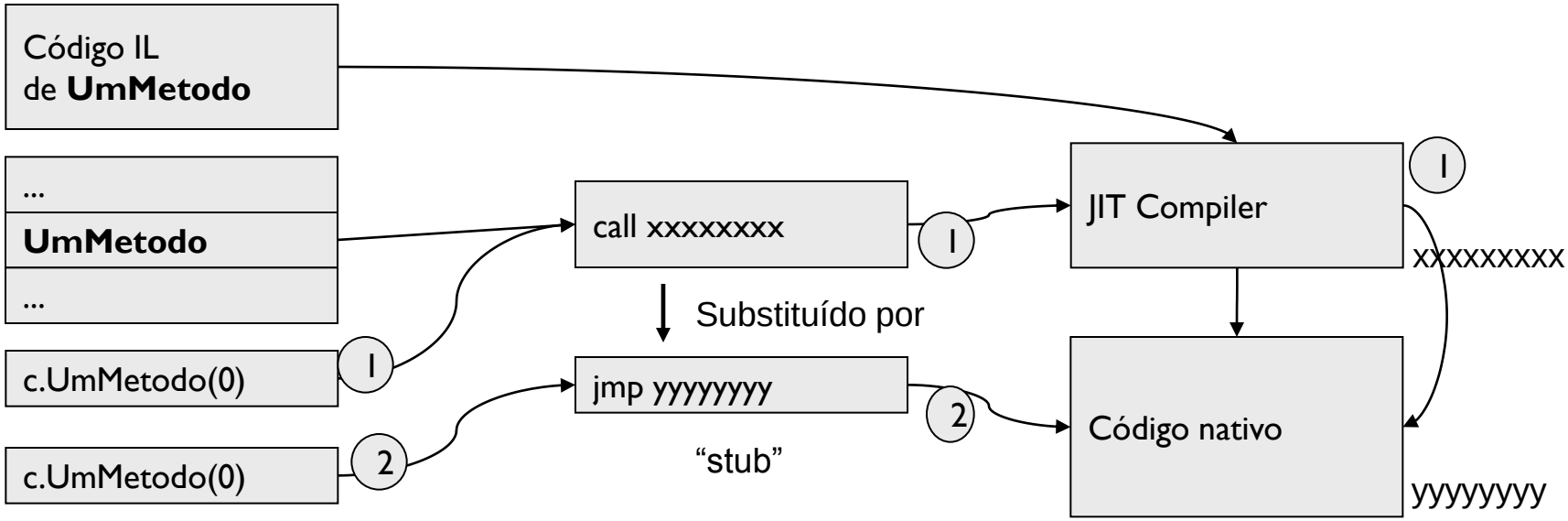
- O *Loader* faz *cache* da informação dos tipos definidos e referênciados no *assembly*, injectando um pequeno *stub* em cada método armazenado.
- O *stub* é utilizado para
  - Denotar o estado da compilação JIT.
  - Transitar entre código *managed* e *unmanaged*.
- O *loader* irá armazenar os tipos referênciados, caso os mesmos ainda não tenham sido armazenados.
- O *loader* utiliza a metadata para inicializar as variáveis estáticas e instânciar os objectos.

Verifier

- Uma das grandes vantagens da máquina virtual é a robustez do código executado
  - Aquando da compilação JIT o CLR executa um processo designado por verificação (pode ser executada manualmente através da ferramenta `PEVerify.exe`)
- O verificador é responsável por verificar que:
  - A metadada está bem definida, isto é válida.
  - O código IL é *type safe*, isto é as assinaturas dos tipos estão a ser utilizadas correctamente verificando deste modo a segurança das instruções. Por exemplo:
    - Verifica se todos os métodos são chamados com o número correcto de parâmetros e que cada parâmetro é do tipo correcto;
    - No acesso a um campo verifica se o objecto acedido é do tipo correcto;
    - Que o tipo de retorno de um método é usado correctamente;
    - Numa operação aritmética, verifica: compatibilidade dos operandos.

Geração de código "just in time"

- As invocações de métodos são realizadas indirectamente através de "stubs"
  - O "stub" de cada método é apontado pela tabela de métodos
- Inicialmente o "stub" aponta para o JIT
- Na primeira chamada do método é invocado o "JIT compiler".
  - Usa o código IL do método e a informação de tipo para gerar o código nativo
  - Altera o "stub" para apontar para o código nativo gerado
  - Salta para o código nativo
- As restantes chamadas usam o código nativo



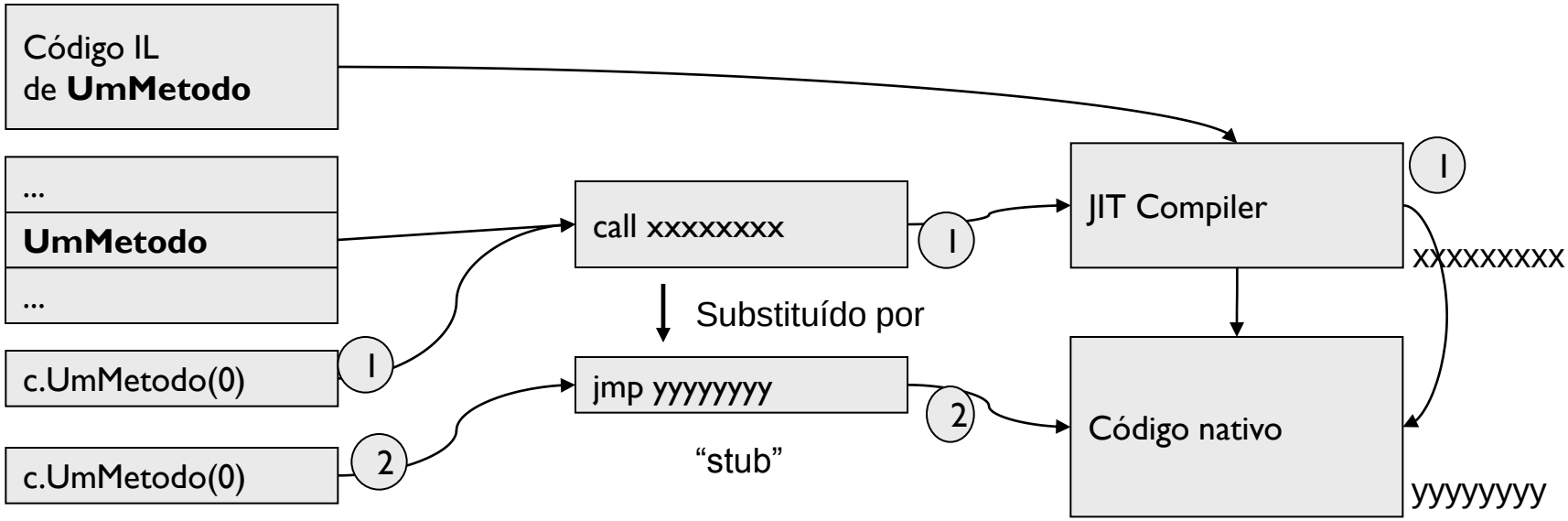
Consequências do modelo "JIT"

- Desvantagens
  - Peso computacional adicional para a geração do código nativo
  - Memória necessária para a descrição intermédia e código nativo

- Vantagens
  - A geração de código tem informação sobre a plataforma nativa de execução
    - Optimização para o processador nativo
    - Utilização de informação de "profiling" (características de execução do código)

Geração de código "just in time"

- As invocações de métodos são realizadas indirectamente através de "stubs"
  - O "stub" de cada método é apontado pela tabela de métodos
- Inicialmente o "stub" aponta para o JIT
- Na primeira chamada do método é invocado o "JIT compiler".
  - Usa o código IL do método e a informação de tipo para gerar o código nativo
  - Altera o "stub" para apontar para o código nativo gerado
  - Salta para o código nativo
- As restantes chamadas usam o código nativo



Consequências do modelo "JIT"

- Desvantagens
  - Peso computacional adicional para a geração do código nativo
  - Memória necessária para a descrição intermédia e código nativo

- Vantagens
  - A geração de código tem informação sobre a plataforma nativa de execução
    - Optimização para o processador nativo
    - Utilização de informação de "profiling" (características de execução do código)

Sistema de tipos

- O *Common Type System* especifica a definição, comportamento, declaração, uso e gestão de tipos.
- Suporta o paradigma da Programação Orientada por Objectos.
- Desenhado por forma a acomodar a semântica expressável na maioria das linguagens modernas.

- Define:
  - Hierarquia de tipos
  - Conjunto de tipos "built-in"
  - Construção de tipos e definição dos seus membros
  - Utilização e comportamento dos tipos

Common Language Specification

- Conjunto de restrições ao CTS para garantir a interoperabilidade entre linguagens
  - Define um sub-conjunto do CTS
  - Contém as regras que os tipos devem respeitar por forma a serem utilizados por qualquer linguagem "CLS-compliant"