



# INTRODUCTION TO THE .NET

.NET RD LAB, MINSK, 2019

ANZHELIKA KRAVCHUK

.NET — это платформа разработки общего назначения. Она включает несколько основных функций

- поддержку нескольких языков программирования
- модель асинхронного программирования
- модель параллельного программирования
- взаимодействие на уровне машинного кода

благодаря которым на различных платформах доступно множество разнообразных сценариев.

## Programming languages

Архитектура .NET поддерживает различные языки программирования. Реализации .NET реализуют [инфраструктуру CLI \(Common Language Infrastructure\)](#), которая, среди прочего указывает среду выполнения, не зависящую от языка, а также взаимодействие языков. Это означает, что для создания приложений и служб на платформе .NET можно выбрать любой язык .NET.

Корпорация Майкрософт активно занимается разработкой и поддержкой трех языков .NET: C#, F# и Visual Basic (VB).

- C# - это простой, эффективный, типобезопасный и объектно-ориентированный язык, сохраняющий выразительность и элегантность, присущие языкам С.
- F# - это кроссплатформенный и функционально-императивный язык программирования, который также поддерживает объектно-ориентированное и императивное программирование.
- Visual Basic - это простой язык, позволяющий научиться разрабатывать разнообразные приложения на .NET.

# Automatic memory management

## Working with unmanaged resources

# Type safety

## Delegates and lambdas

# Generics

# Async programming

# Language Integrated Query (LINQ)

Приложение .NET разрабатывается и выполняется в одной или нескольких *реализациях .NET*. К реализации .NET относятся .NET Framework, .NET Core, Mono. Существует спецификация API, общая для всех реализаций .NET, которая называется .NET Standard.

## .NET Standard

.NET Standard — это набор API-интерфейсов, которые реализуются библиотекой базовых классов реализации .NET. Фактически это спецификация API .NET, представляющая единый набор контрактов, на основе которых компилируется код. Эти контракты реализуются в каждой реализации .NET. Благодаря этому возможен перенос между различными реализациями .NET, что обеспечивает эффективную работу кода везде.

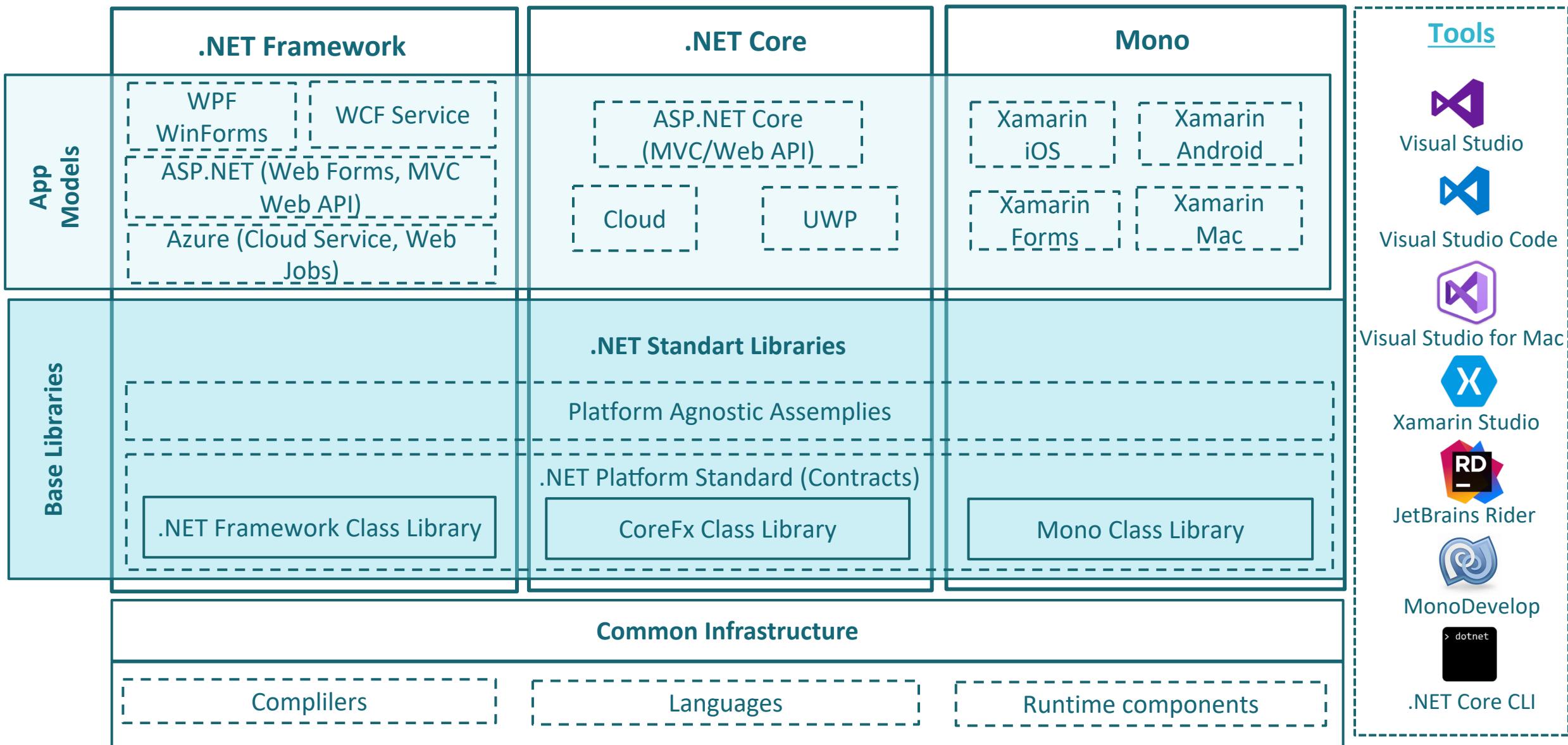
.NET Standard также является целевой платформой. Если код предназначен для версии .NET Standard, он будет работать в любой реализации .NET, которая поддерживает эту версию .NET Standard.

## .NET architectural components

Каждая реализация .NET включает в себя следующие компоненты.

- Одна среда выполнения или несколько.
- Библиотека классов, которая реализует .NET Standard, а также может реализовывать дополнительные API-интерфейсы.
- (Необязательно) Одна или несколько моделей приложений.
- (Необязательно) Средства разработки.

# .NET implementations



## .NET implementations. .NET Core

.NET Core — это кроссплатформенная реализация .NET, предназначенная для обработки обширного ряда серверных и облачных рабочих нагрузок. Работает в Windows, macOS и Linux, реализует .NET Standard, следовательно, любой код, предназначенный для .NET Standard, может работать на .NET Core. ASP.NET Core выполняется в .NET Core.

## .NET implementations. .NET Framework

.NET Framework является исходной реализацией .NET (2002 г.). Версии 4.5 и более поздние реализуют .NET Standard, следовательно, любой код, предназначенный для .NET Standard, может работать в этих версиях .NET Framework. Содержит дополнительные API для Windows, например API для разработки настольных приложений с помощью Windows Forms и WPF. .NET Framework оптимизирована для создания классических приложений Windows.

## .NET implementations. Mono

Mono является реализацией .NET, которая в основном используется, если требуется небольшая среда выполнения. Это среда выполнения, которая может работать в приложениях Xamarin на Android, Mac, iOS, tvOS и watchOS, и предназначена для небольших разработок. Mono также подходит для работы игр, созданных на базе подсистемы Unity.

Поддерживает все текущие опубликованные версии .NET Standard.

Исторически Mono реализовывала крупный API .NET Framework и эмулировала некоторые из наиболее популярных возможностей в Unix. Иногда она использовалась для запуска приложений .NET, которые применяют эти возможности в Unix.

Mono обычно используется с JIT-компилятором, но также располагает полным статическим компилятором (заблаговременная компиляция), который используется на таких платформах, как iOS.

## .NET implementations. Universal Windows Platform (UWP)

UWP представляет собой реализацию .NET, которая используется для создания современных приложений Windows с поддержкой сенсорного ввода и программного обеспечения для Интернета вещей (IoT). Она предназначена для объединения различных типов устройств, которые могут потребоваться, включая ПК, планшеты, планшетофоны, телефоны и даже Xbox. UWP предоставляет много служб, таких как централизованный магазин приложений, среда выполнения (AppContainer) и набор API-интерфейсов Windows для использования вместо Win32 (WinRT). Приложения могут быть написаны на C++, C#, VB.NET и JavaScript. При использовании C# и VB.NET API-интерфейсы .NET предоставляются .NET Core.

## .NET runtimes

Среда выполнения — это среда выполнения для управляемой программы. Операционная система является частью среды выполнения, но не входит в среду выполнения .NET. Среды выполнения .NET.

- Среда CLR для .NET Framework
- Среда CoreCLR для .NET Core
- .NET Native для универсальной платформы Windows
- Среда выполнения Mono для Xamarin.iOS, Xamarin.Android, Xamarin.Mac и платформы Mono для рабочего стола

## .NET tooling and common infrastructure

Существует широкий набор средств и компонентов инфраструктуры, которые работают в каждой реализации платформы .NET. Они включают в себя, помимо прочих средств, следующее:

- языки .NET и соответствующие компиляторы;
- систему проектов .NET (на основе файлов *CSPROJ*, *VBPROJ* и *FSPROJ*);
- [MSBuild](#), обработчик для сборки проектов;
- [NuGet](#), диспетчер пакетов корпорации Майкрософт для .NET;
- инструменты для управления сборкой с открытым исходным кодом, например [CAKE](#) и [FAKE](#).

*Common Language Infrastructure (ECMA-335)* это открытая спецификация, разработанная фирмой Microsoft, которая описывает код исполнительной программы и среду выполнения. Спецификация подразумевает среду разрешающую нескольким языкам высокого уровня быть использованными на разных компьютерных платформах без переписи под специфику архитектур.

CLI это *спецификация*, а не *реализация*, которая содержит аспекты вне спецификации.

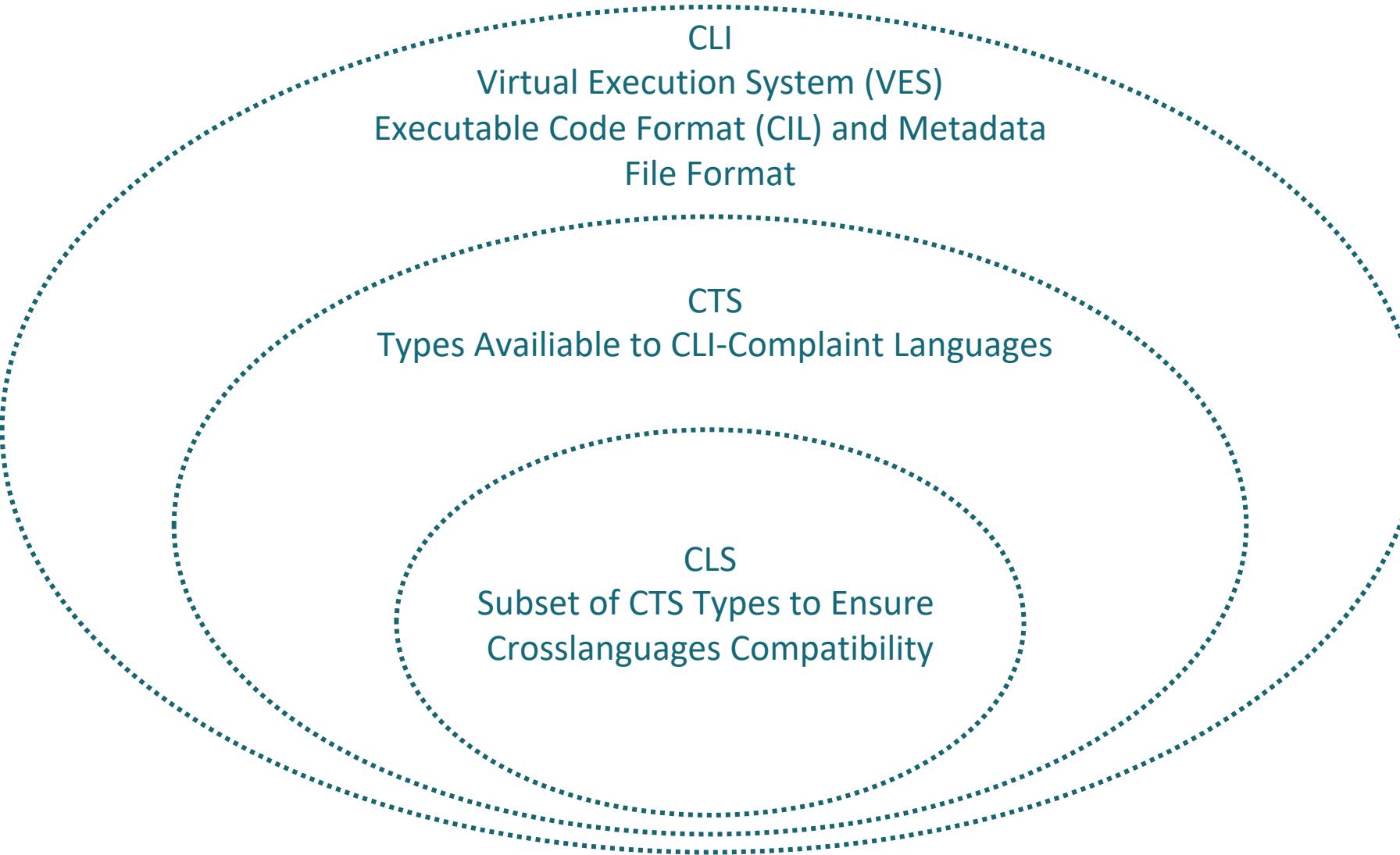
Спецификация CLI содержит:

*Common Type System (CTS)* - Набор типов и операций которые используются во многих языках программирования.

*Metadata* - Информация о структуре программы является независимой от языка, потому вы можете использовать программу в разных языках.

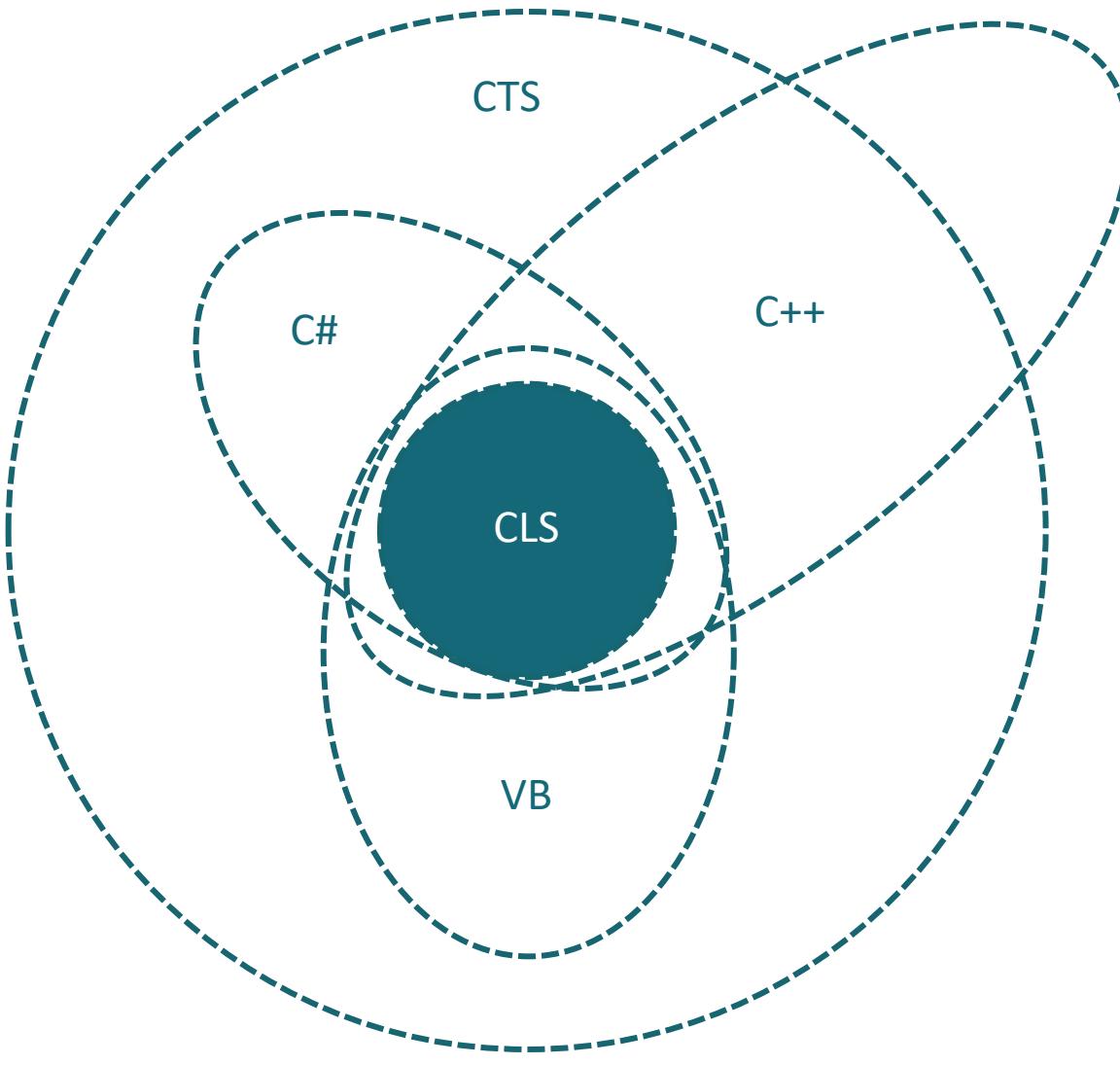
*Common Language Specification (CLS)* - Набор базовых правил, которые язык программирования CLI должен соблюдать, чтобы общаться с другими CLS языками.

*Virtual Execution System (VES)* - Загружает и выполняет CLI-совместимые программы, используя метаданные чтобы совместить сгенерированные куски кода во время исполнения.



## CLI/CTS Languages

- VB .NET
- C++/CLI
- C#
- F#
- Oxygen
- Nemerle
- IronPython
- ...



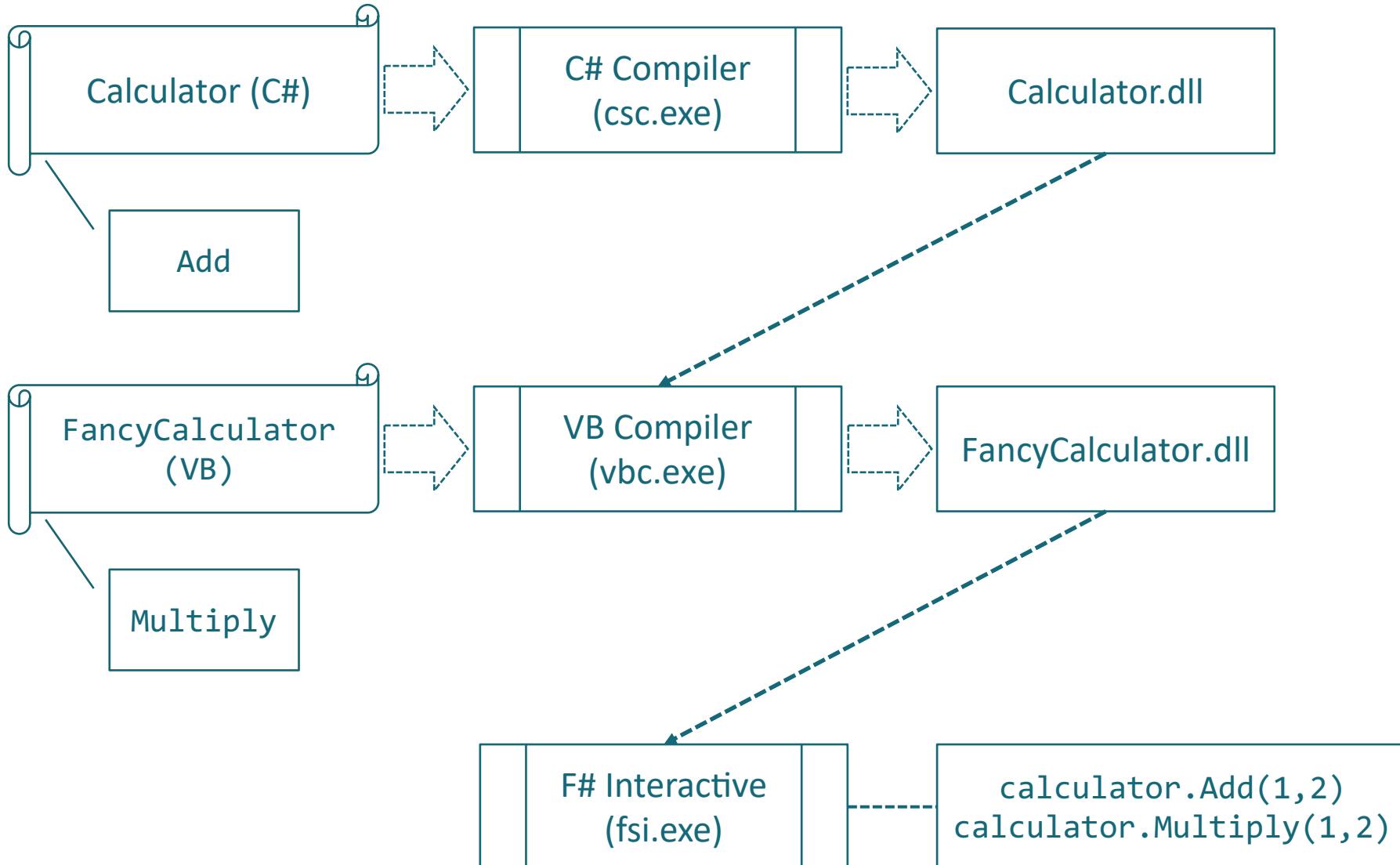
Not CLS Compliant: UInt – not CLS type

```
[assembly: CLSCompliant(true)]  
  
using System;  
  
public class Person  
{  
    private UInt16 personAge = 0;  
  
    public UInt16 Age  
    {  
        get => personAge;  
    }  
}
```

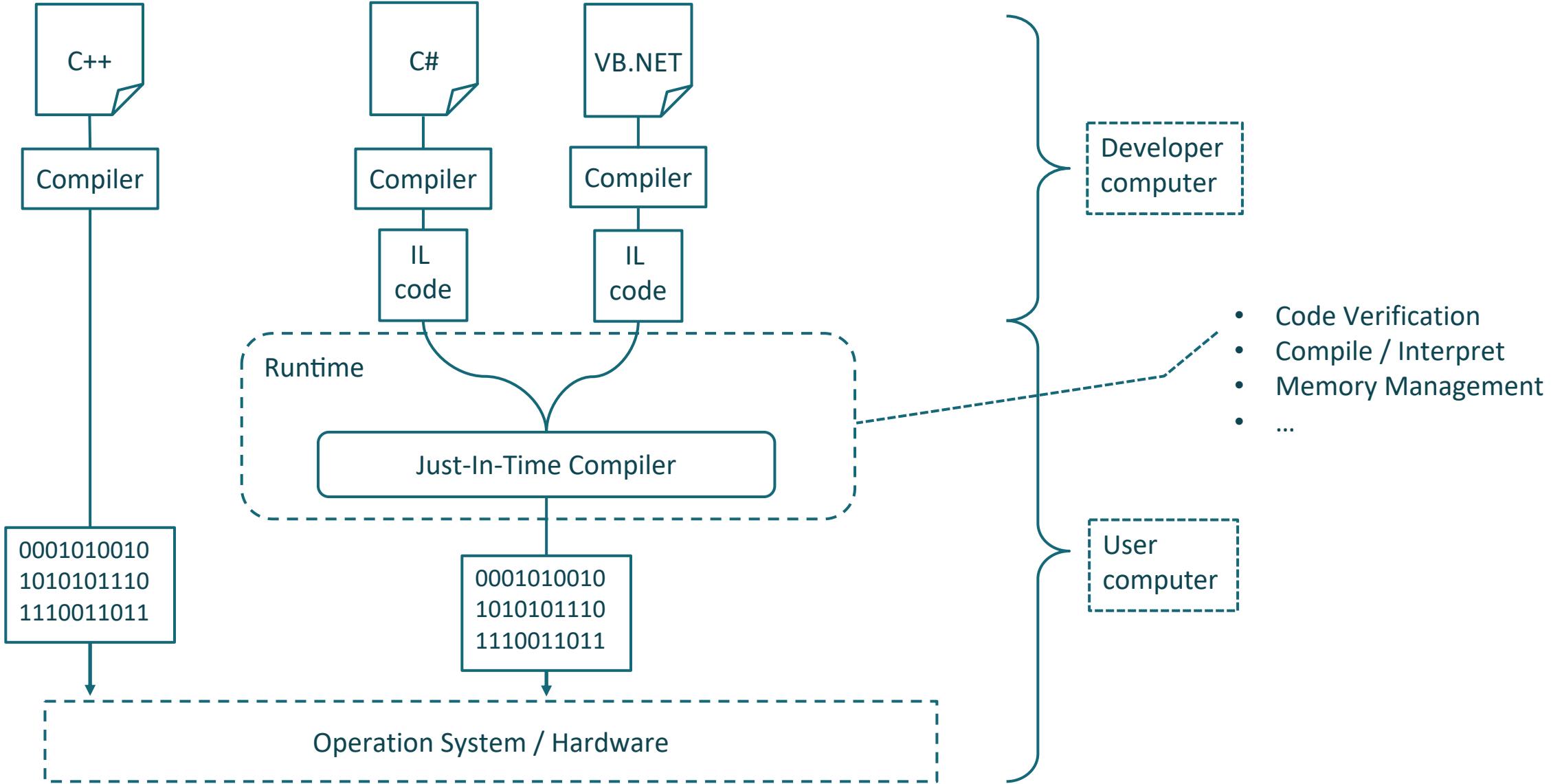
CLS Compliant

```
[assembly: CLSCompliant(true)]  
  
using System;  
  
public class Person  
{  
    private Int16 personAge = 0;  
  
    public Int16 Age  
    {  
        get => personAge;  
    }  
}
```

# Common Type System, Common Language Specification



# Programming languages / IL (CIL) / Managed vs unmanaged environment



## CIL Language. Some key points

- Язык CIL – набор инструкций, независимых от типа процессора.
- CIL позволяет абстрагироваться от проблем физического представления данных, присущих машинному коду.

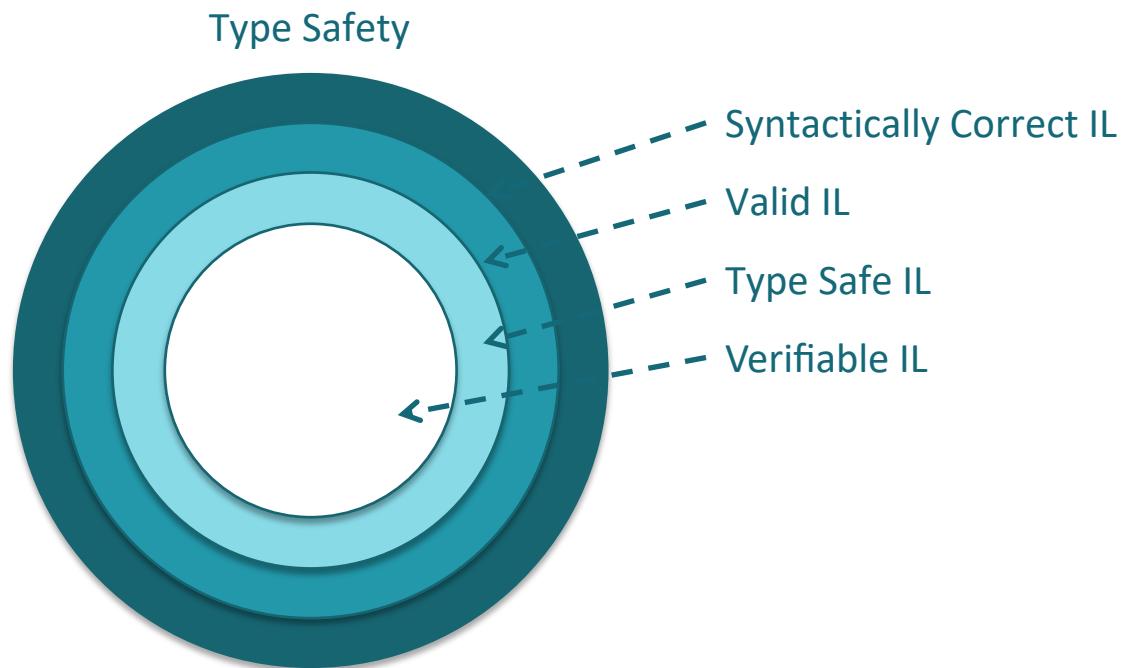
В CIL доступ к полям и вызовы методов выполняются с помощью кодов операций (opcode), абсолютные смещения и адреса не используются – инструкции CIL содержат ссылки на метаданные, описывающие методы или поля, над которыми выполняется операция. Ссылки основаны исключительно на имени и сигнатуре поля или метода, а не нахождении или смещении – CLR не нужно смещение, она сама его вычисляет.

```
IL_0001: ldstr      "Hello, world!"  
IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
```

## CIL Language. Some key points

IL является стековым языком – все его инструкции заносят операнды в стек вычислений (evaluation stack) и извлекают результаты из стека. IL не содержит инструкций для работы с регистрами (абстрагирует разработчика от конкретного процессора), что упрощает создание новых языков и компиляторов, генерирующих код для CLR.

IL-код обеспечивает безопасность приложения и его устойчивость перед ошибками – в процессе компиляции IL в машинные инструкции CLR выполняется процедура, называемая *верификацией* — анализ высокоуровневого кода IL и проверка безопасности всех операций.



## CIL Language. Some key points

- CLR никогда не выполняет код CIL непосредственно – перед выполнением он всегда транслируется в машинный код.

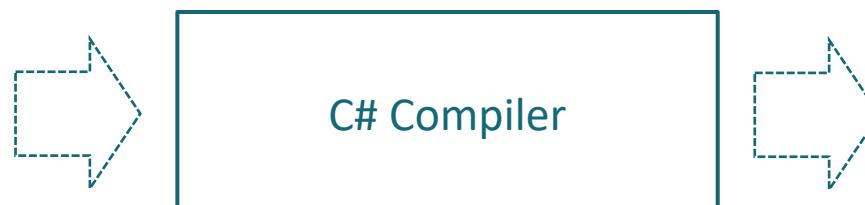
Трансляция выполняется или при загрузке компонента в память, или при его установке на развертывающем компьютере. В обоих случаях после окончания трансляции CIL в машинный код фактическое представление в памяти для любых типов данных и таблиц методов используется для генерации машинного кода, что позволяет сгенерировать эффективный машинный код с достаточно небольшим количеством служебных переходов.

## CIL Language. Some key points

- Сгенерированный средой CLR машинный код обладает такой же эффективной физической компоновкой, как и в случае использования C++ или COM, однако в отличие от последних, для которых физическая компоновка происходит во время разработки, среда CLR не вычисляет детали физического связывания до тех пор, пока не будет выполнена трансляция CIL в машинный код. В силу трансляции в машинный код на развертывающем компьютере, а не на компьютере разработчика, определения типов, требуемые для внешних компонентов, совпадут с определениями типов, найденными на развертывающем компьютере, что снижает уязвимость межкомпонентных контрактов, не снижая производительность.
- Поскольку трансляция в машинный код происходит на развертывающем компьютере, любые раскладки или правила выравнивания, зависящие от типа процессора, полностью удовлетворяют архитектуру процессора, который будет выполнять код.

## C# Code

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello, world!");
        }
    }
}
```



# CIL (Common Intermediate Language) Code

```
.class private auto ansi beforefieldinit HelloWorld.Program extends [mscorlib]System.Object
{
    .method private hidebysig static void
        Main(string[] args) cil managed
    {
        .entrypoint
        .maxstack 8
        // [12 9 - 12 10]
        IL_0000: nop
        // [13 13 - 13 55]
        IL_0001: ldstr      "Hello, world!"
        IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
        IL_000b: nop
        // [14 9 - 14 10]
        IL_000c: ret
    } // end of method Program::Main

    .method public hidebysig specialname rtspecialname instance void .ctor() cil managed
    {
        .maxstack 8
        IL_0000: ldarg.0      // this
        IL_0001: call       instance void [mscorlib]System.Object:::.ctor()
        IL_0006: nop
        IL_0007: ret
    } // end of method Program::ctor
} // end of class HelloWorld.Program
```



## Machine Code



```
016C0448 push    ebp
016C0449 mov     ebp,esp
016C044B push    edi
016C044C push    esi
016C044D push    ebx
016C044E sub     esp,30h
016C0451 xor     ebx,ebx
016C0453 mov     dword ptr [ebp-10h],ebx
016C0456 mov     dword ptr [ebp-1Ch],ebx
016C0459 mov     dword ptr [ebp-3Ch],ecx
016C045C cmp     dword ptr ds:[1634288h],0
016C0463 je      016C046A
016C0465 call    69E96340
016C046A nop
                    System.Console.WriteLine("Hello, world!");
016C046B mov     ecx,dword ptr ds:[41622C4h]
016C0471 call    68B46600
016C0476 nop
```

# .NET Framework

## Релизы

- Первая в 2002 (1.0)
- Current in апрель, 2018(4.8)

## Поддерживаемые платформы

- Только Windows – Desktop and Server
- x86 and x64

## Частично open source

- <https://referencesource.microsoft.com/>
- <https://github.com/Microsoft/referencesource>

## Redistributable Packages

- CLR + BCL + Frameworks (WCF, WPF, ...) + Old developer tools (MSBuild 4.0, C# 4.0/VB.Net 9.0, ...)
- <https://www.microsoft.com/net/download/all> or OS part

## Developer Pack

- Redistributable + SDK (some utilities) + Reference Assemblies
- <https://www.microsoft.com/net/download/all>

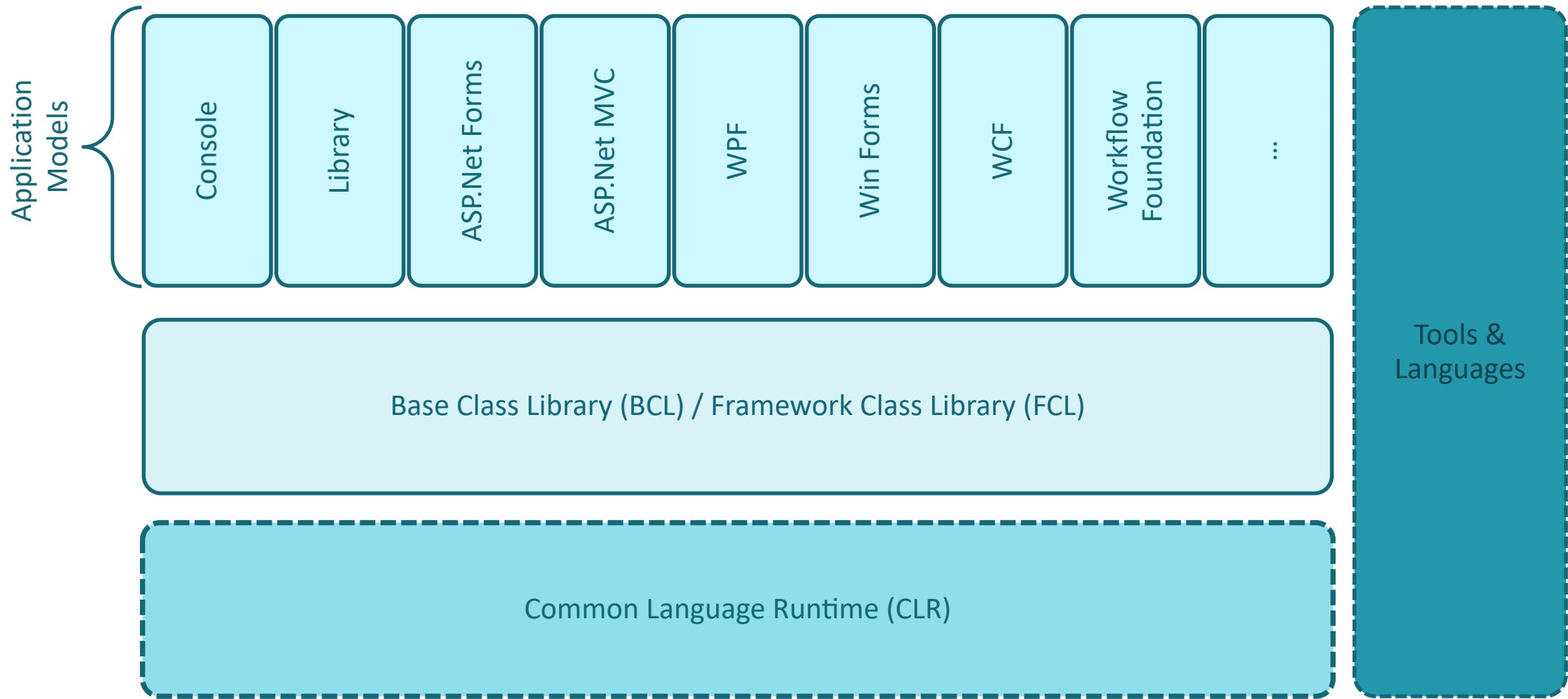
## Build tools

- MSBuild + Compilers (C#/VB.Net/C++) + some libraries (MFC/ATL)
- <https://www.visualstudio.com/downloads/> (Build Tools for Visual Studio 2017)

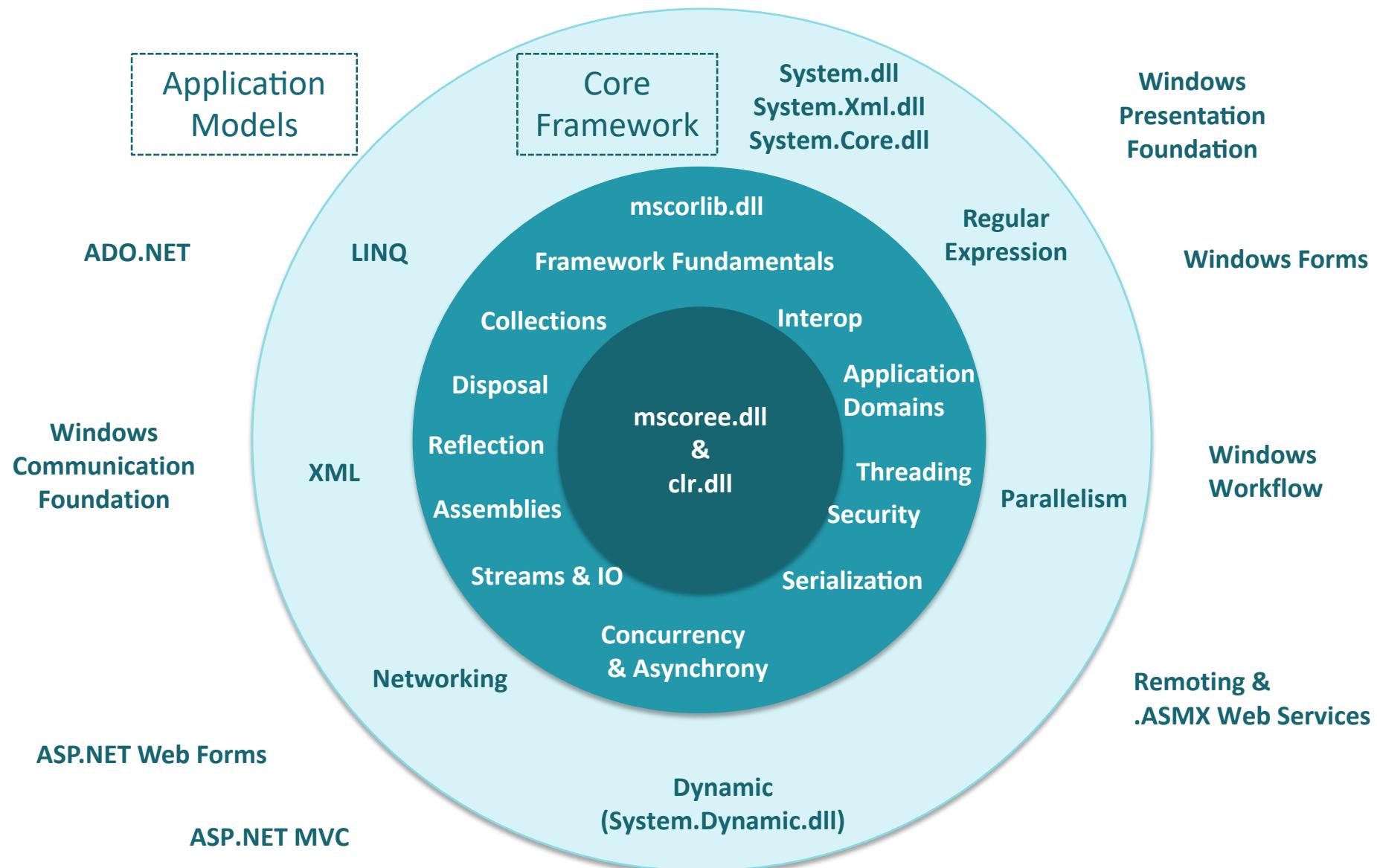
## Visual Studio (Community)

- All together
- <https://www.visualstudio.com/downloads/>

# .NET Framework Components



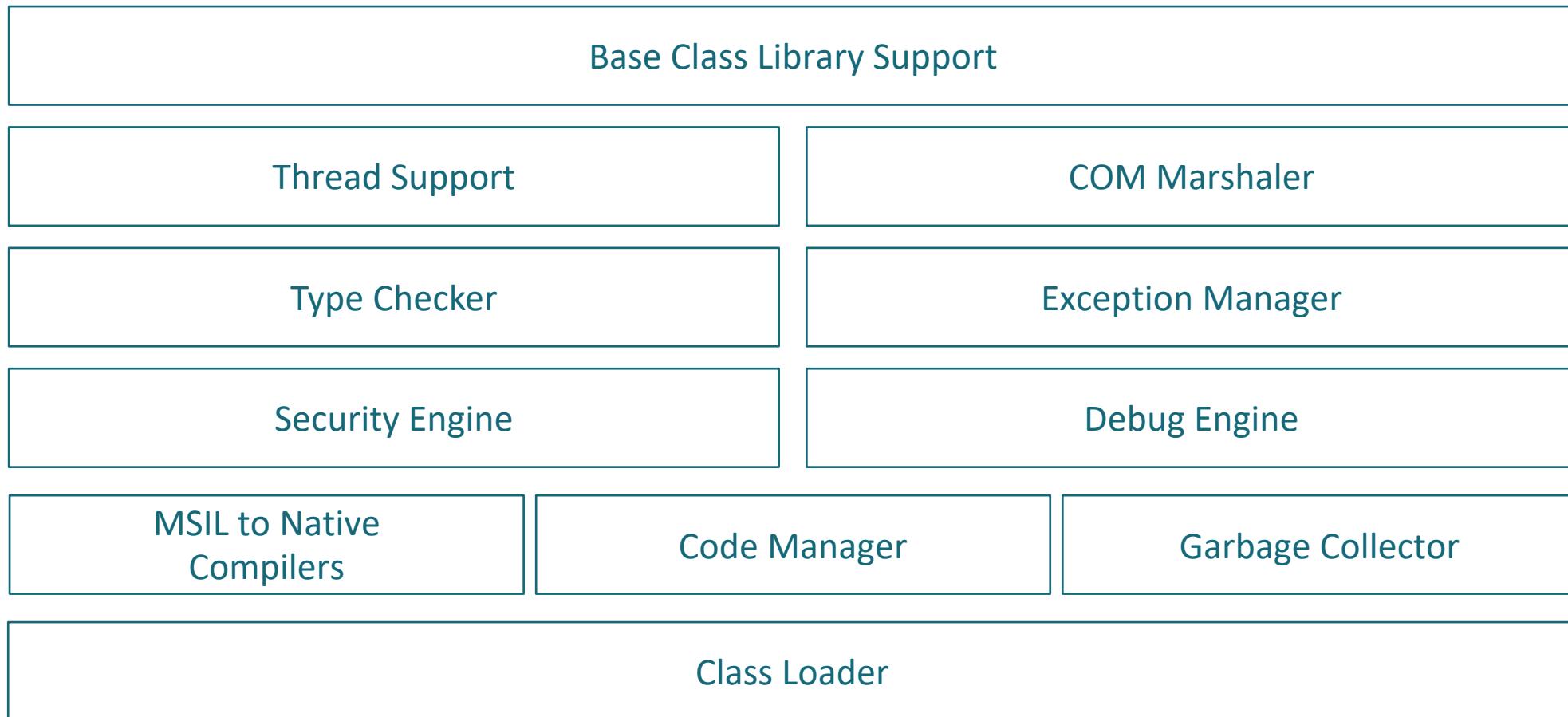
# .NET Framework Components



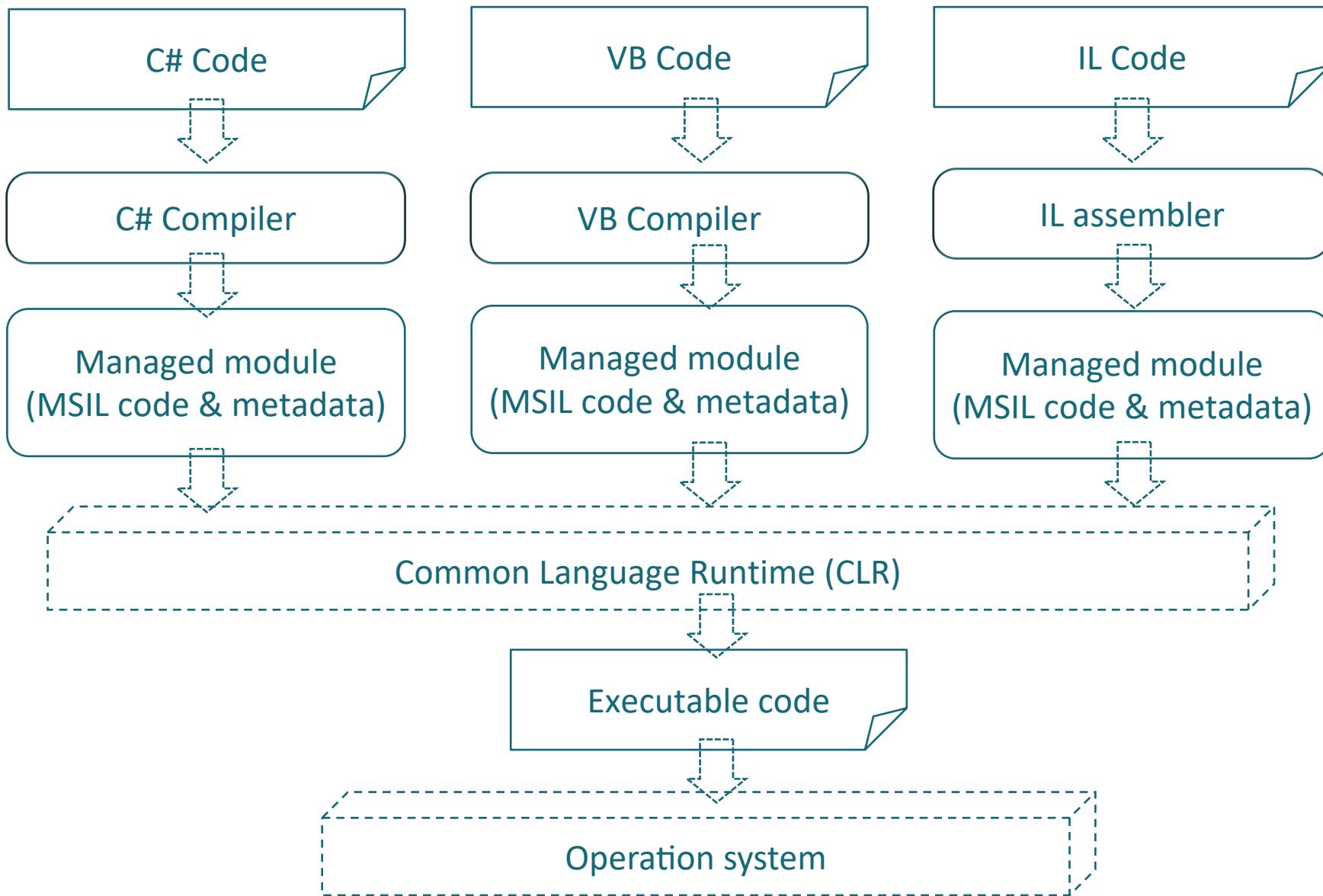
## CLR versions

| .NET Framework version | Includes CLR version |
|------------------------|----------------------|
| 1.0                    | 1.0                  |
| 1.1                    | 1.1                  |
| 2.0                    | 2.0                  |
| 3.0                    | 2.0                  |
| 3.5                    | 2.0                  |
| 4                      | 4                    |
| 4.5.*                  | 4                    |
| 4.6.*                  | 4                    |
| 4.7.*                  | 4                    |
| 4.8                    | 4                    |

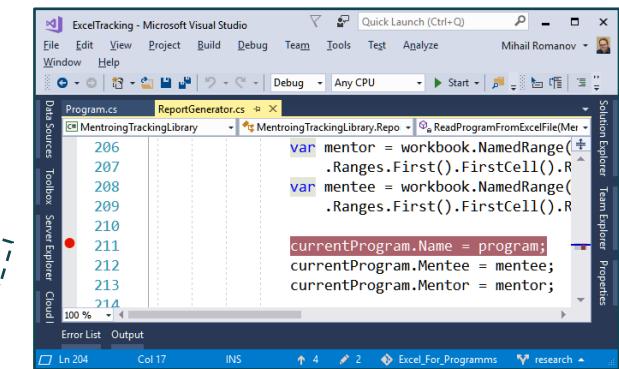
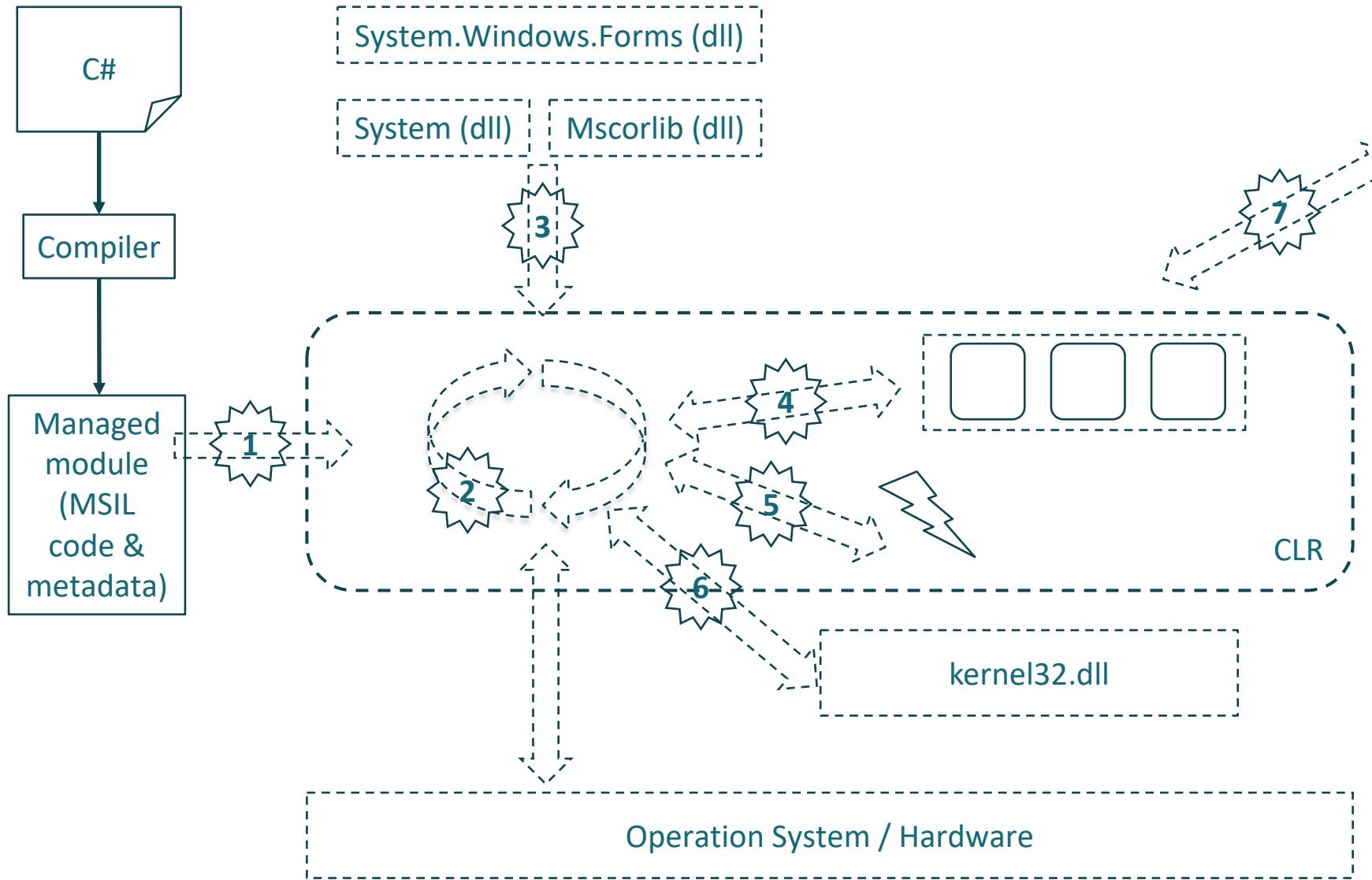
# CLR runtime services



## Управляемые модули, MSIL код и метаданные



# CLR runtime services



## Managed Modules

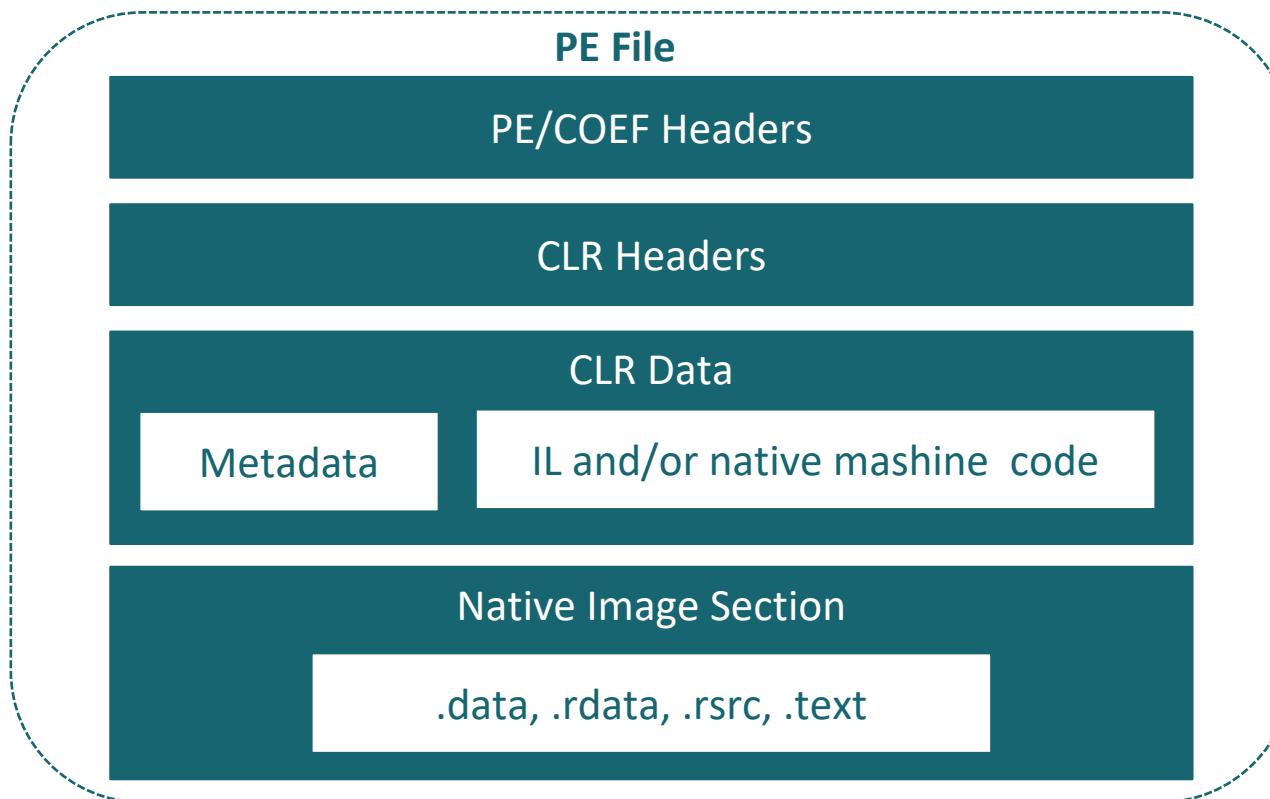
Для создания, развертывания и поиска компонентов CLR имеет собственный набор концепций и технологий, принципиально отличающихся от тех, которые используются в COM, Java или Win32.

Программы, написанные для выполнения CLR, находятся в управляемых модулях - байтовых потоках, хранящихся в виде файлов в локальной файловой системе или на web-сервере.

Чтобы загрузчик операционной системы распознал исполняемый файл как действительный, он должен иметь структуру, определенную в формате файла PE/COFF (Portable Executable, Common Object File Format).

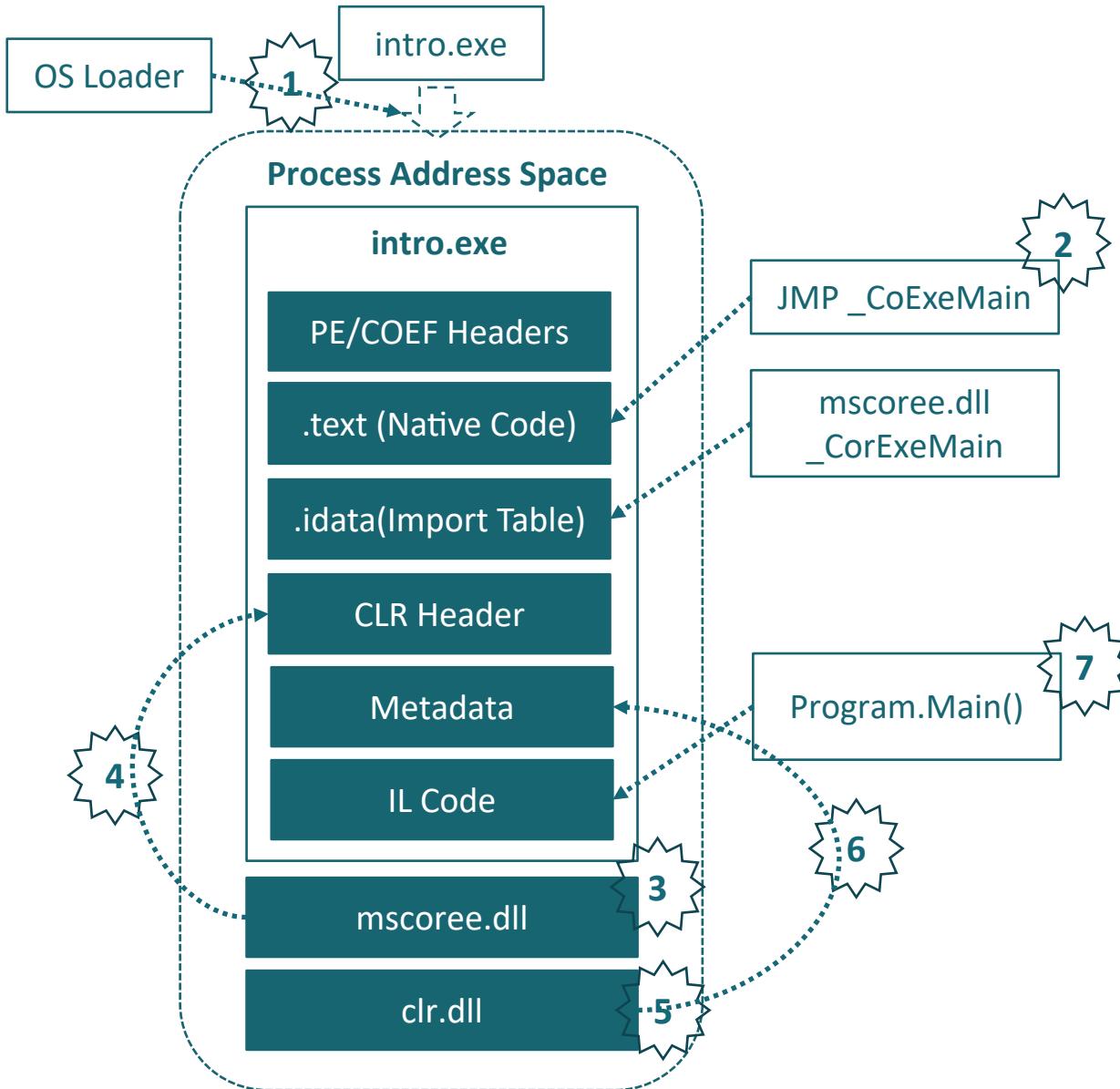
## Managed Modules. PE-file

Модуль CLR содержит код, метаданные и ресурсы. Код обычно хранится в формате CIL, хотя он также может быть сохранен в виде машинных инструкций, специфичных для процессора. Метаданные модуля описывают типы, определенные в модуле, включая имена, отношения наследования, сигнатуры методов и информацию о зависимостях. Ресурсы модуля состоят из статических данных только для чтения, таких как строки, растровые изображения и другие аспекты программы, которые не сохраняются как исполняемый код.



# Demo

# Loading the CLR



## Loading the CLR

1. Исполняемый файл загружается ОС
2. Loader stub to CLR entry point
3. CLR shim initialization
4. Inspect headers and policy
5. Shim loads specific CLR version
6. Finding managed entry point
7. Control transfer to managed code

## Class Loader (Загрузчик классов)

- находит и загружает .NET-классы в память
- готовит их для исполнения
- кэширует информацию о классе, чтобы класс не пришлось загружать снова в процессе работы
- определяет, сколько требуется выделить памяти для экземпляра класса
- вставляет заглушку, вроде пролога функции, в каждый метод загруженного класса, предназначенную для того, чтобы отмечать состояние JIT-компиляции и для перехода между управляемым и неуправляемым кодом
- если загруженный класс ссылается на другие классы, загрузчик попытается загрузить эти классы, если классы, указанные в ссылках, уже были загружены, загрузчику ничего делать не надо
- использует метаданные для инициализации статических переменных и создания экземпляра загруженного класса

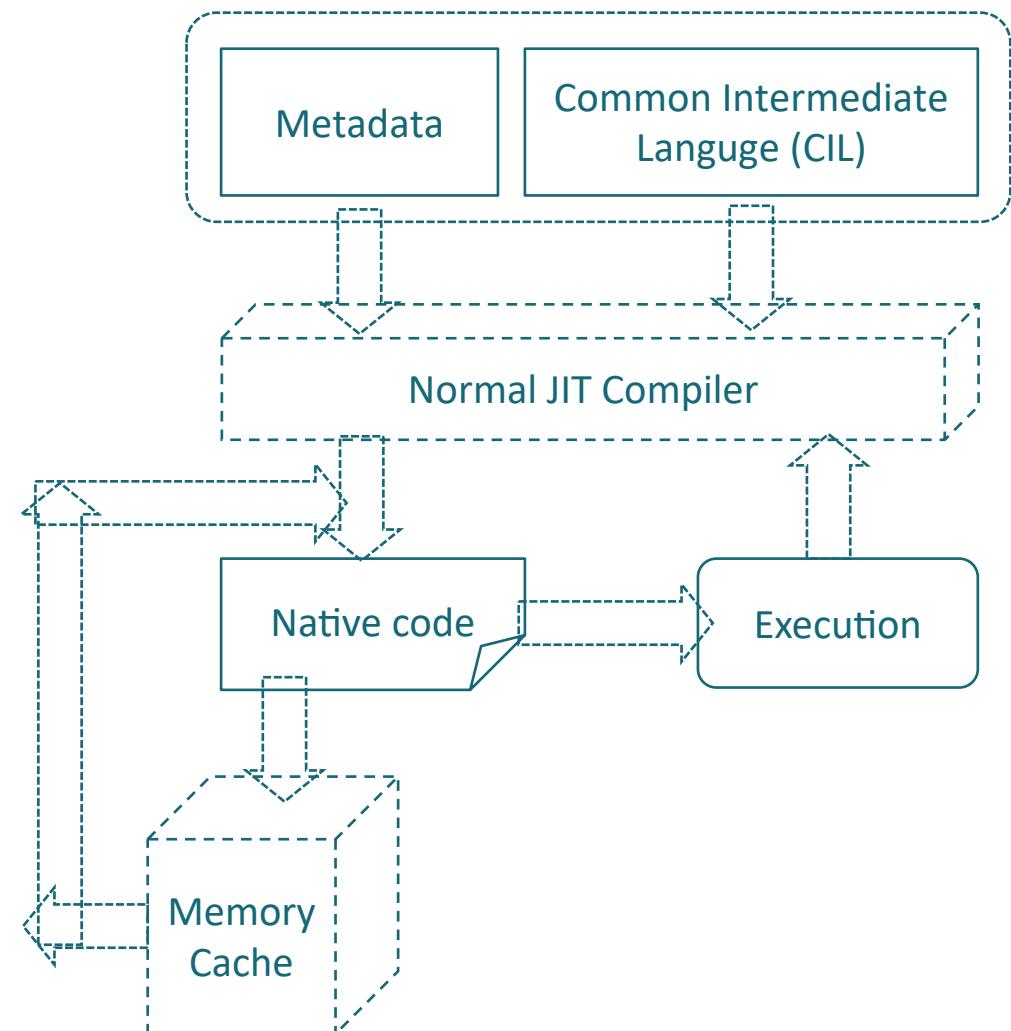
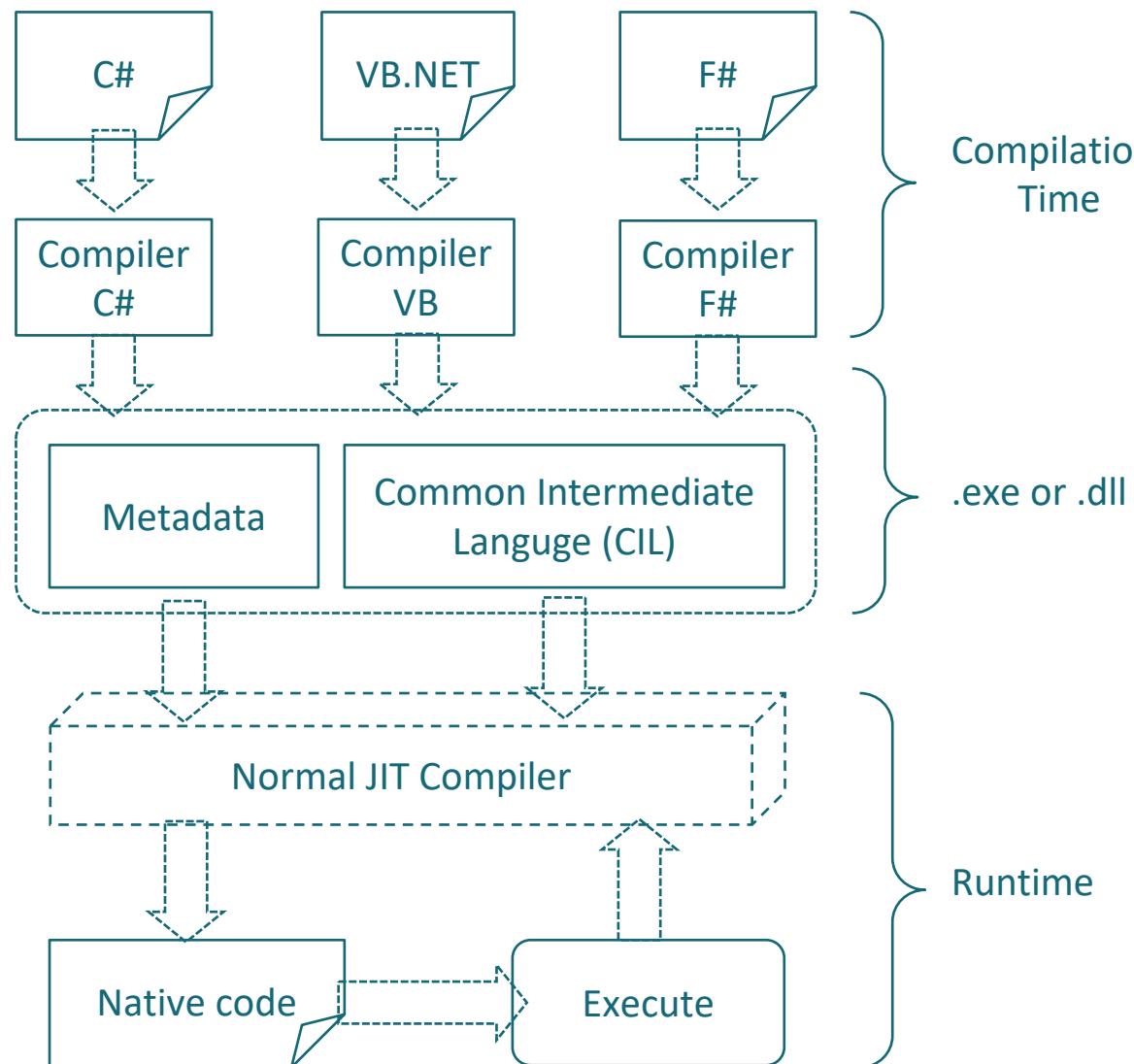
## Type Checker (Верификатор)

- проверяет являются ли метаданные корректными и действительными
- проверяет безопасен ли IL-код в отношении типов, т. е. корректно ли используются сигнатуры типов

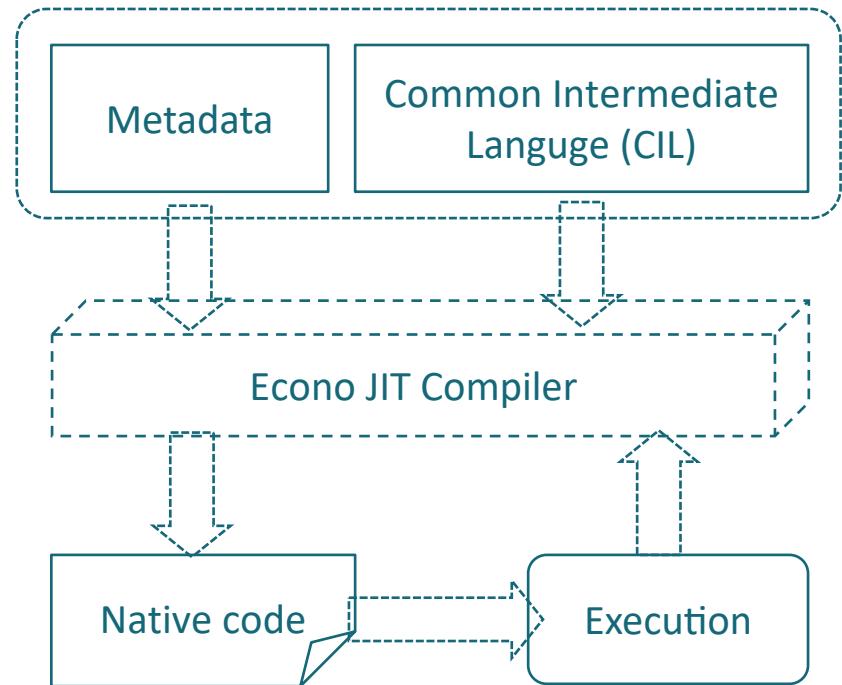
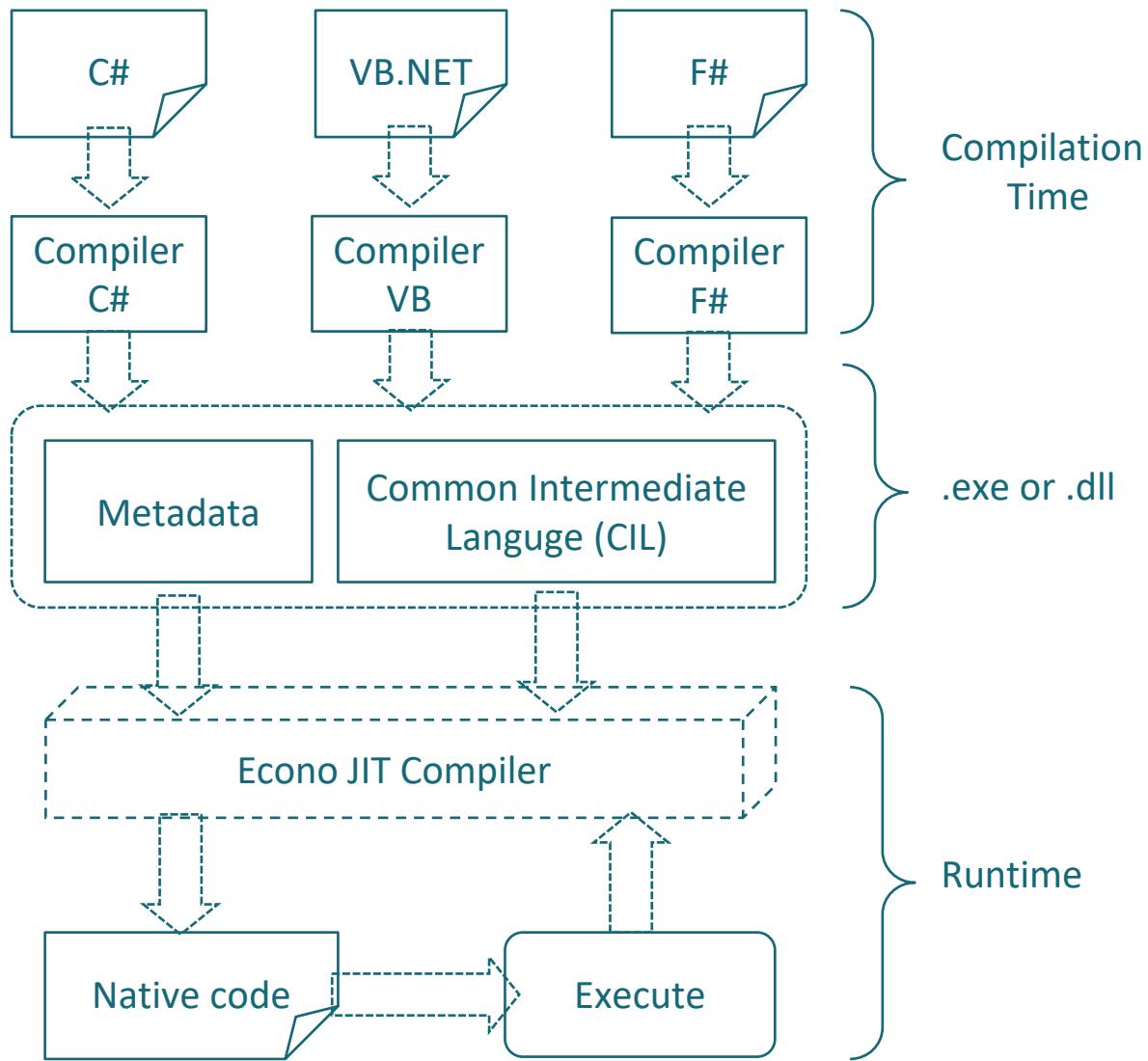
## MSIL to Native Compilers (JIT -компиляторы)

- компилирует метод и преобразует его в управляемый машинный код
- генерирует управляемые данные, необходимые диспетчеру кода для поиска и разворачивания стековых фреймов

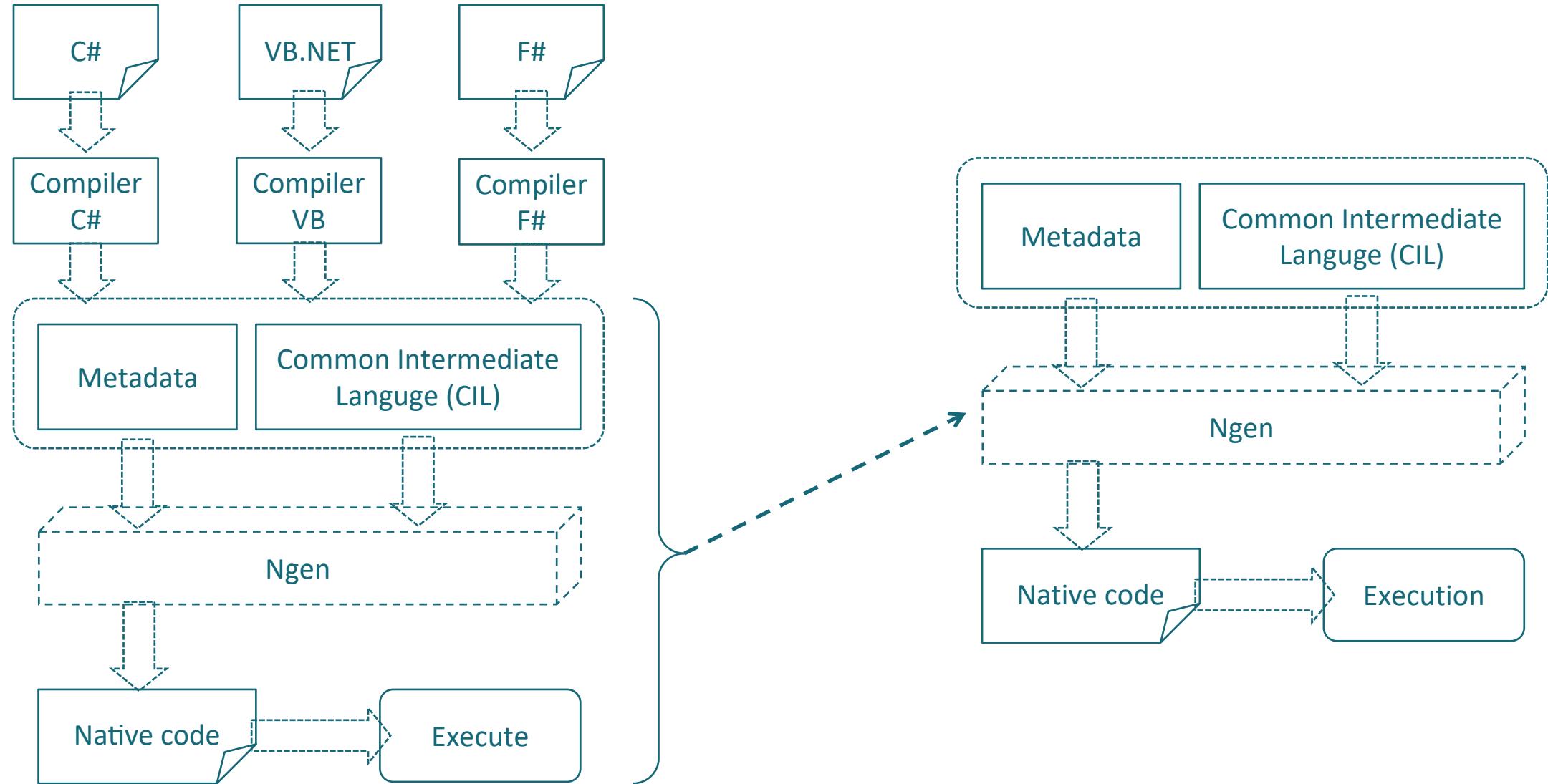
# Виды JIT-компиляции Normal JIT-компиляция



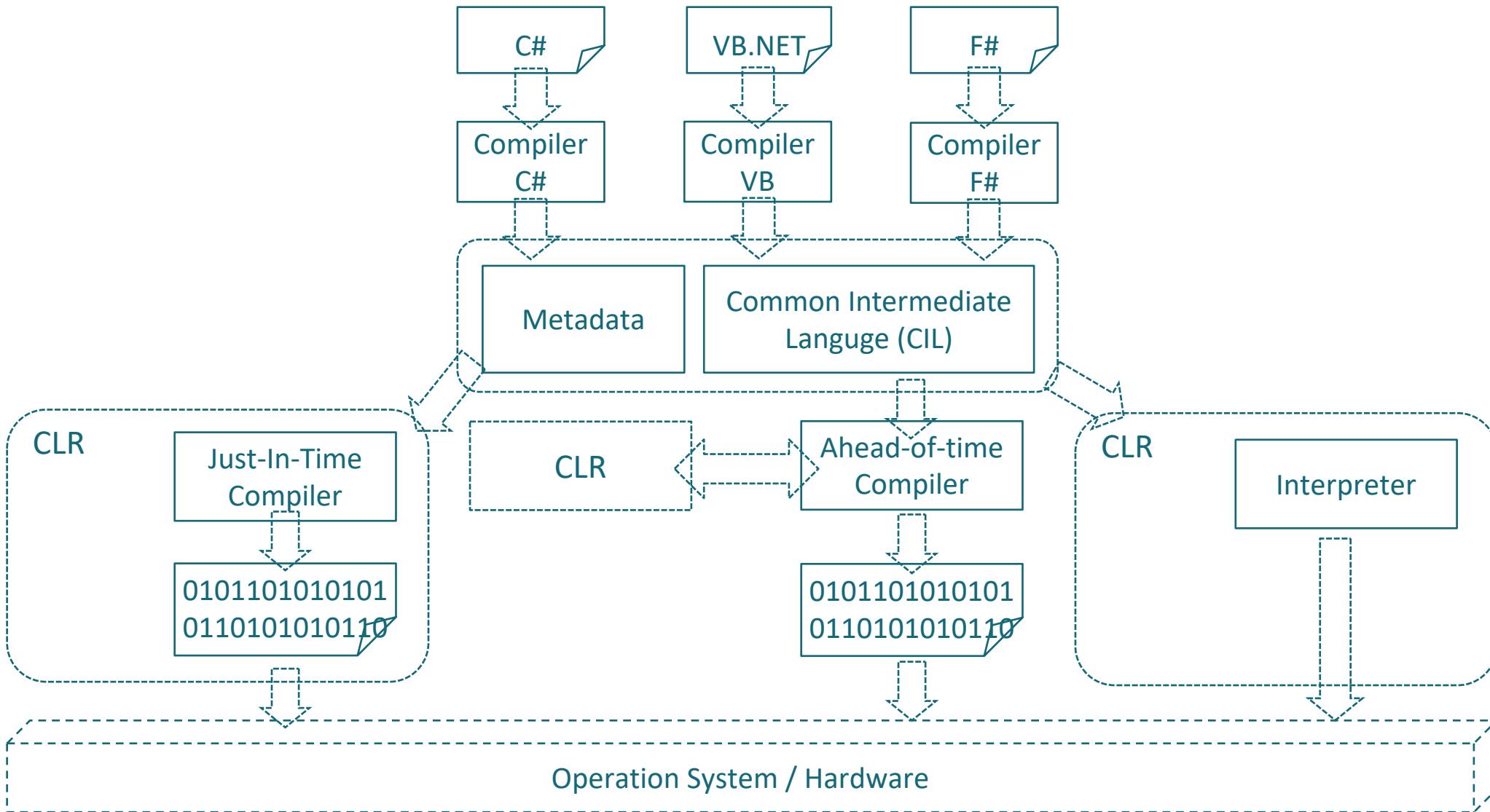
# Виды JIT-компиляции Econo-JIT-компиляция

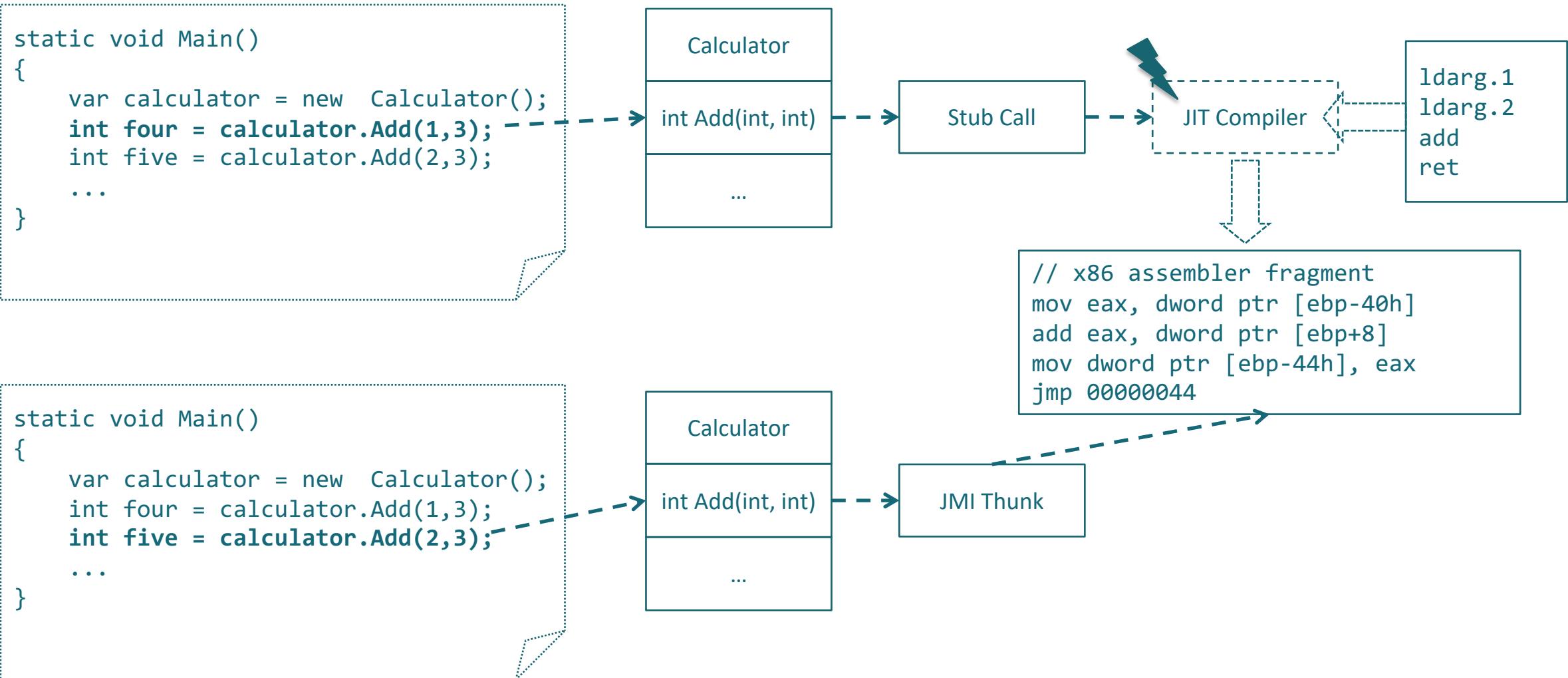


# Виды JIT-компиляции Pre-JIT-компиляция



# Виды JIT-компиляции





## CLR runtime services

**Code Manager (диспетчер кода)** использует управляемые данные для управления исполнением кода, в том числе перемещения по стеку

**Garbage Collector (сборка мусора)** поддержка автоматического управления временем жизни всех объектов среды .NET

**Exception Manager (Обработка исключений)** поддерживает стандартный механизм обработки исключений, работающий во всех языках, позволяя всем программам использовать общий механизм обработки ошибок. Механизм обработки исключений в CLR интегрирован со структурной обработкой исключений (Windows Structured Exception Handling, SEH)

**Security Engine (поддержка безопасности)** осуществляет различные проверки безопасности, чтобы гарантировать, что код безопасен для исполнения и не нарушает никаких требований безопасности

## CLR runtime services

Debug Engine (поддержка отладки) предоставляет богатую поддержку отладки и профилирования, содержит поддержку управления исполнением программы, точек останова, исключений, управляющей логики и т. п.

COM Marshaler (поддержка взаимодействия кода) поддерживает взаимодействие между управляемым (CLR) и неуправляемым (без CLR) «мирами». Средство COM Interop играет роль моста, соединяющего COM и CLR, обеспечивая COM-объекту возможность использовать .NET-объект, и наоборот. Средство Platform Invoke (P/Invoke) позволяет вызывать функции Windows API.

Thread Support (поддержка потоков) – CLR обладает собственной абстракцией, концептуально аналогичной потоку операционной системы

## Managed modules, MSIL code and metadata

Метаданные устраняют необходимость в заголовочных и библиотечных файлах при компиляции

Visual Studio .NET использует метаданные для облегчения написания кода

В процессе верификации кода CLR использует метаданные, чтобы убедиться, что код совершает только «безопасные» операции

Метаданные позволяют сериализовать поля объекта в блок памяти на удаленной машине и затем десериализовать, восстановив объект и его состояние на этой машине

Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов

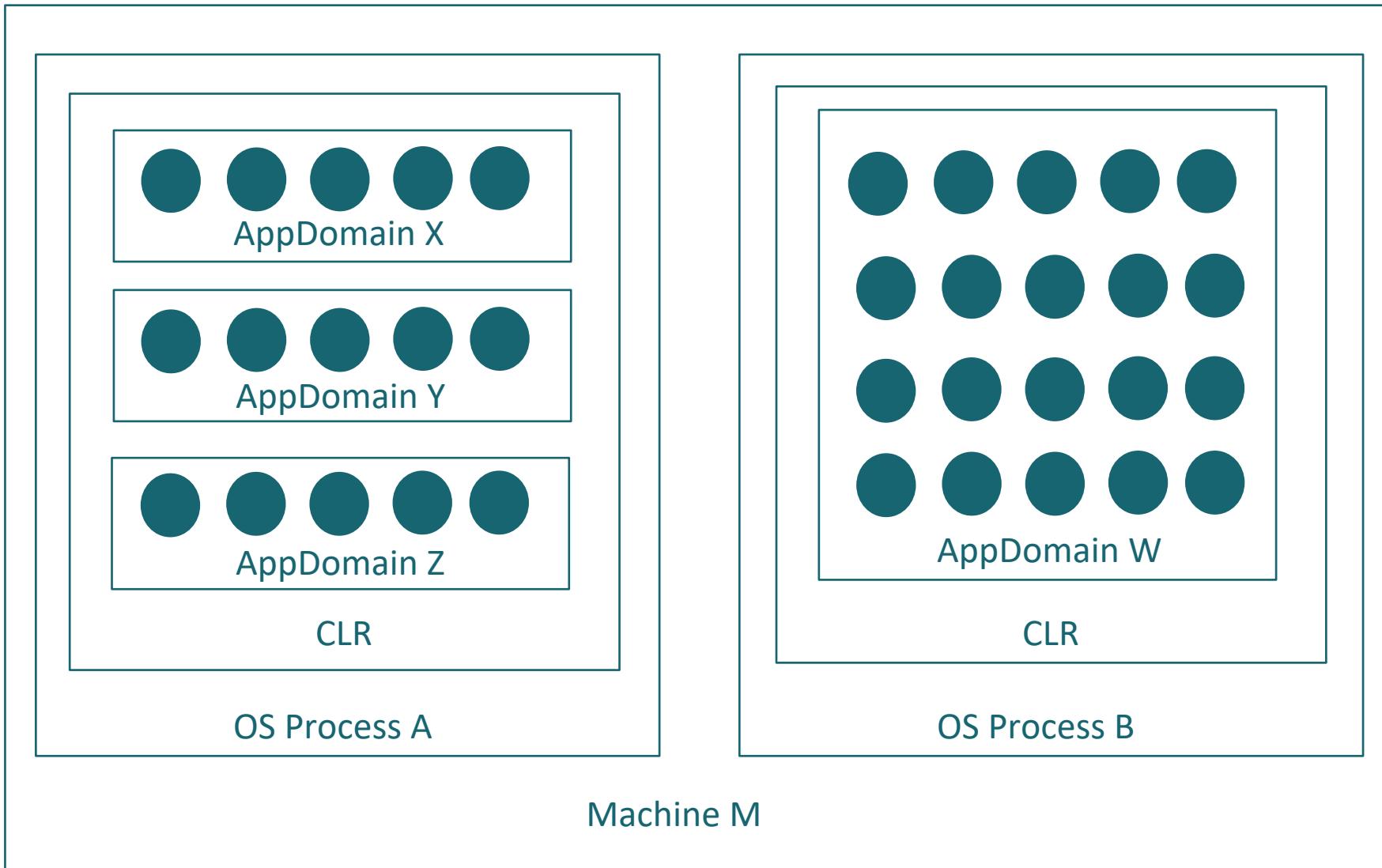
## Инфраструктура времени выполнения. Домены приложений

Многие технологии и среды программирования определяют свои собственные уникальные модели разграничения областей выполняющегося кода и владения ресурсами. На уровне операционной системы модель разграничения базируется на концепции процессов. Для виртуальной машины Java она базируется на загрузчиках классов, для web-сервера IIS - на виртуальном каталоге. Для среды CLR основными областями разграничениями служат домены приложения.

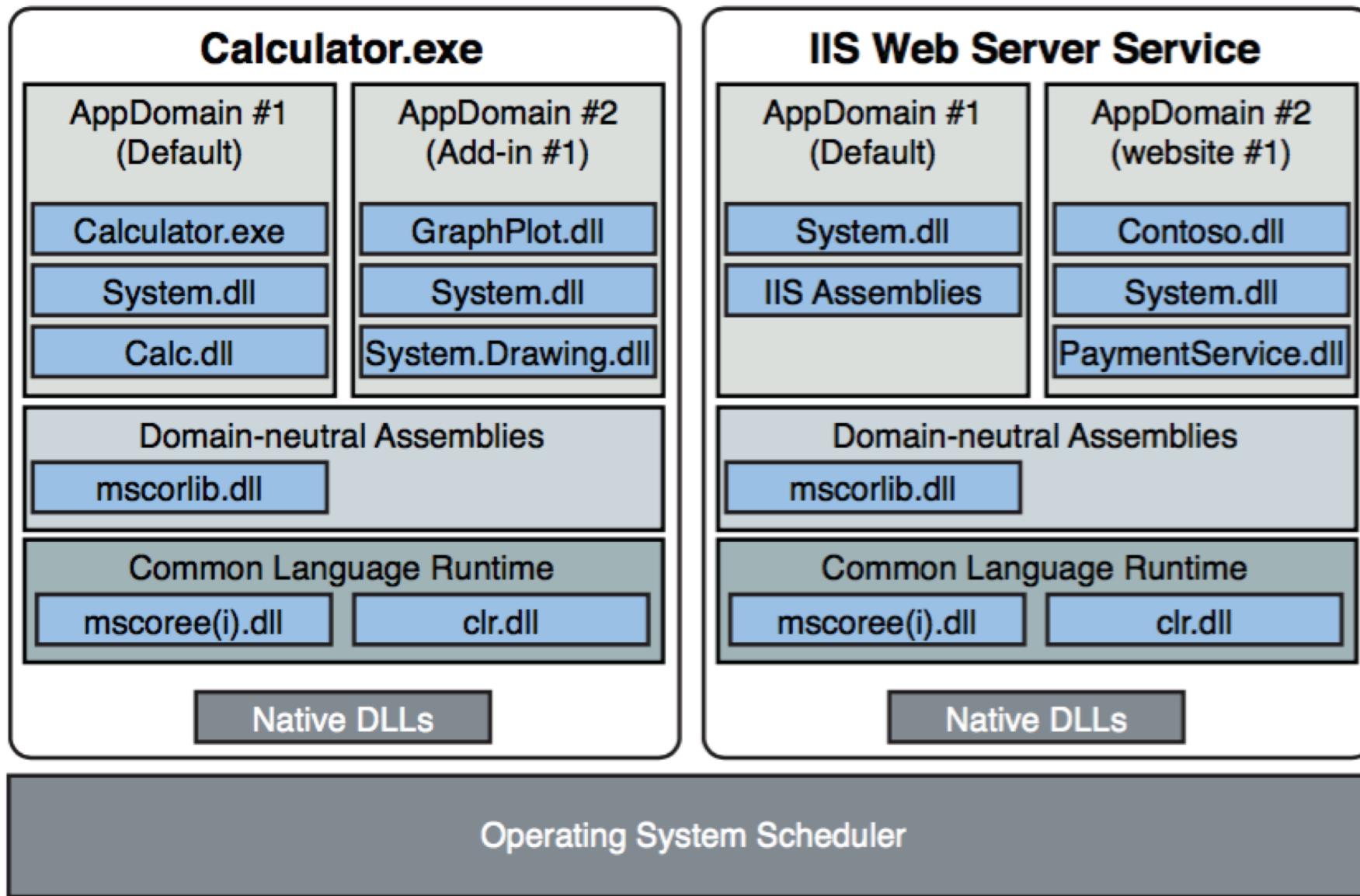
Домены приложений (AppDomains) играют ту же роль, что и процессы в операционной системе. Как и процесс, домен приложения, выделяет (ограничивает) некоторую область выполнения кода, предоставляет уровни изоляции ошибок, обеспечивает изоляцию области и уровень безопасности, владеет ресурсами «от лица» программы, которая в нем выполняется.

Процесс – это абстракция, созданная операционной системой, а домен – абстракция, созданная средой CLR. Домен приложения всегда существует точно в одном процессе операционной системы, в то же время один процесс может поддерживать произвольное количество доменов.

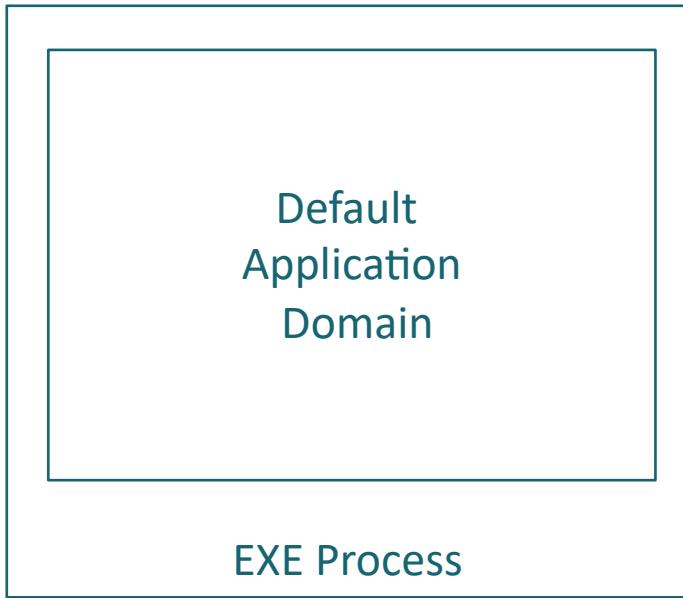
# Объекты, домены и процессы



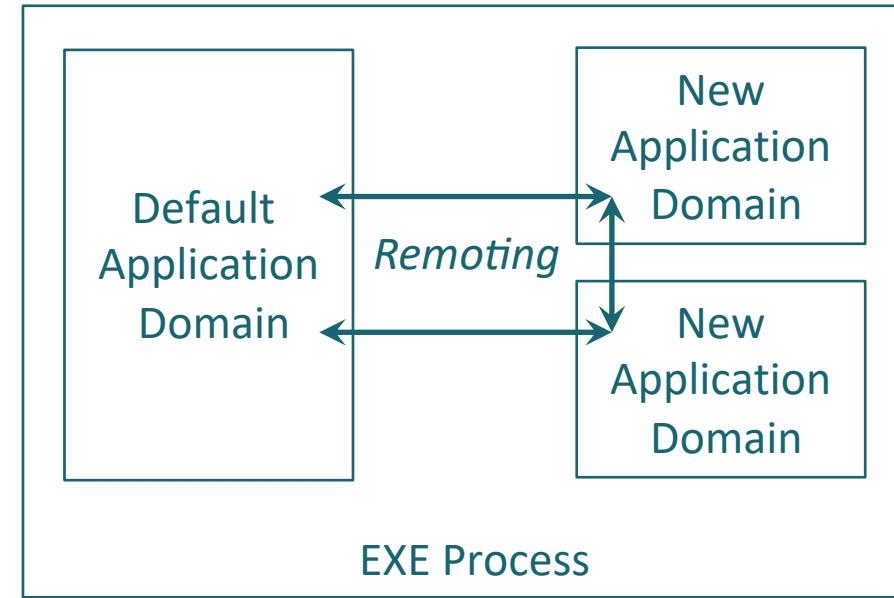
# Инфраструктура времени выполнения. Сборки и домены



# Инфраструктура времени выполнения. Сборки и домены

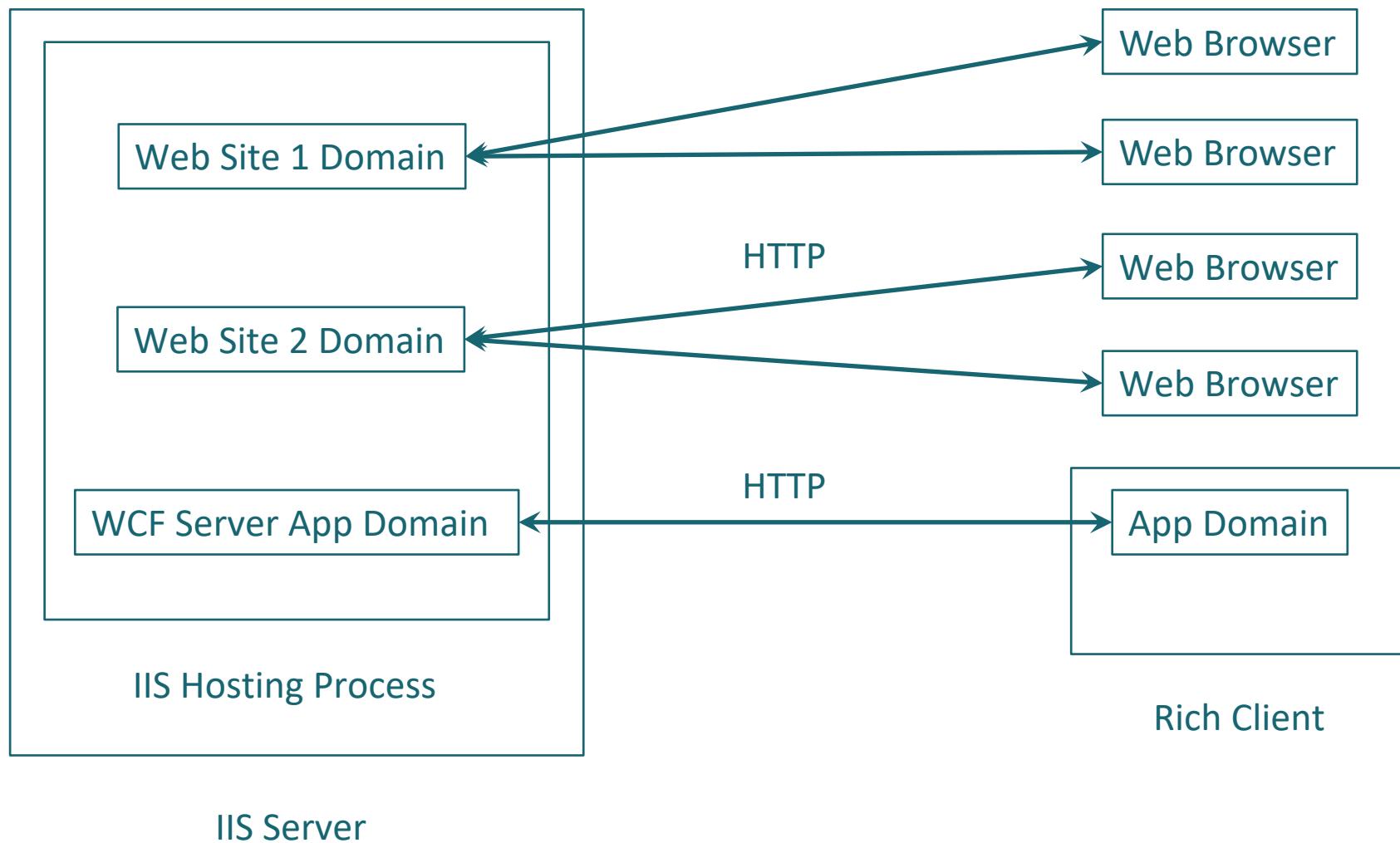


Single-Application-Domain-Program



Multy-Application-Domain-Program

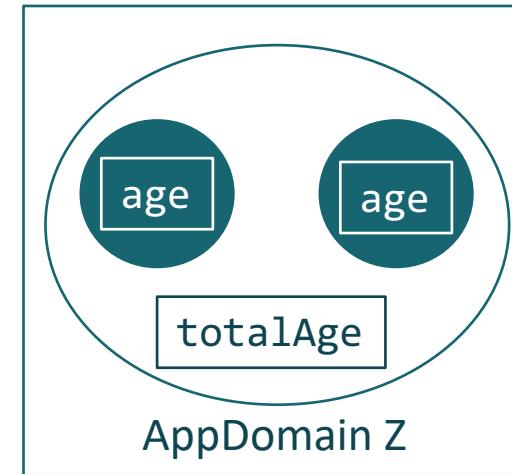
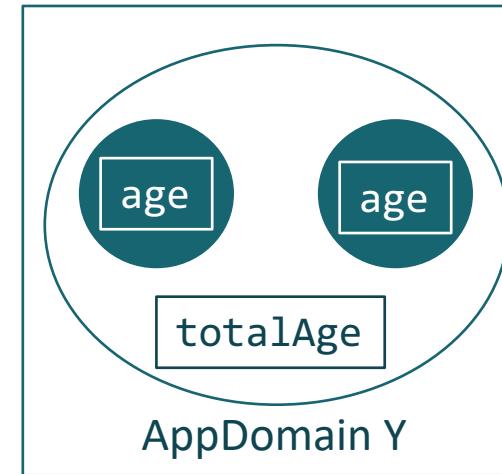
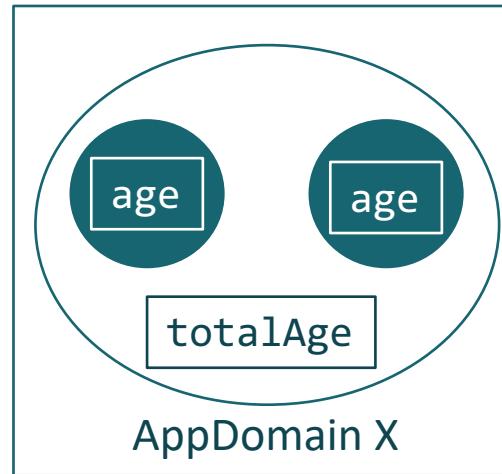
## Runtime infrastructure. Threads



## Runtime infrastructure. Application Domains

Домен приложения – это единица изоляции времени выполнения, внутри которой запускается программа .NET. Он представляет границы управляемой памяти, контейнер для загруженных сборок и параметров конфигурации приложения.

```
public class Person
{
    private static int totalAge;
    private int age;
}
```



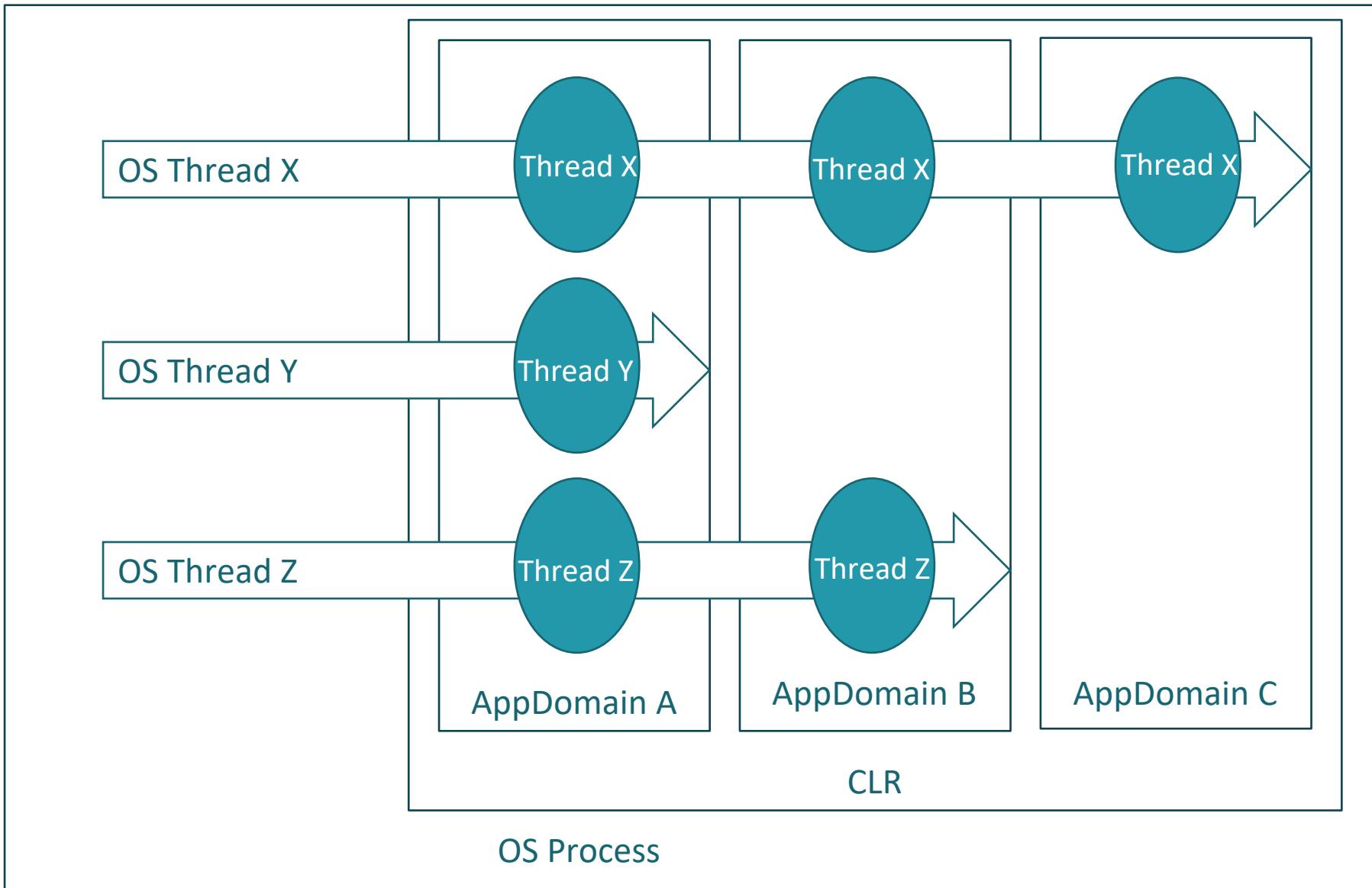
OS Process

## Runtime infrastructure. Threads

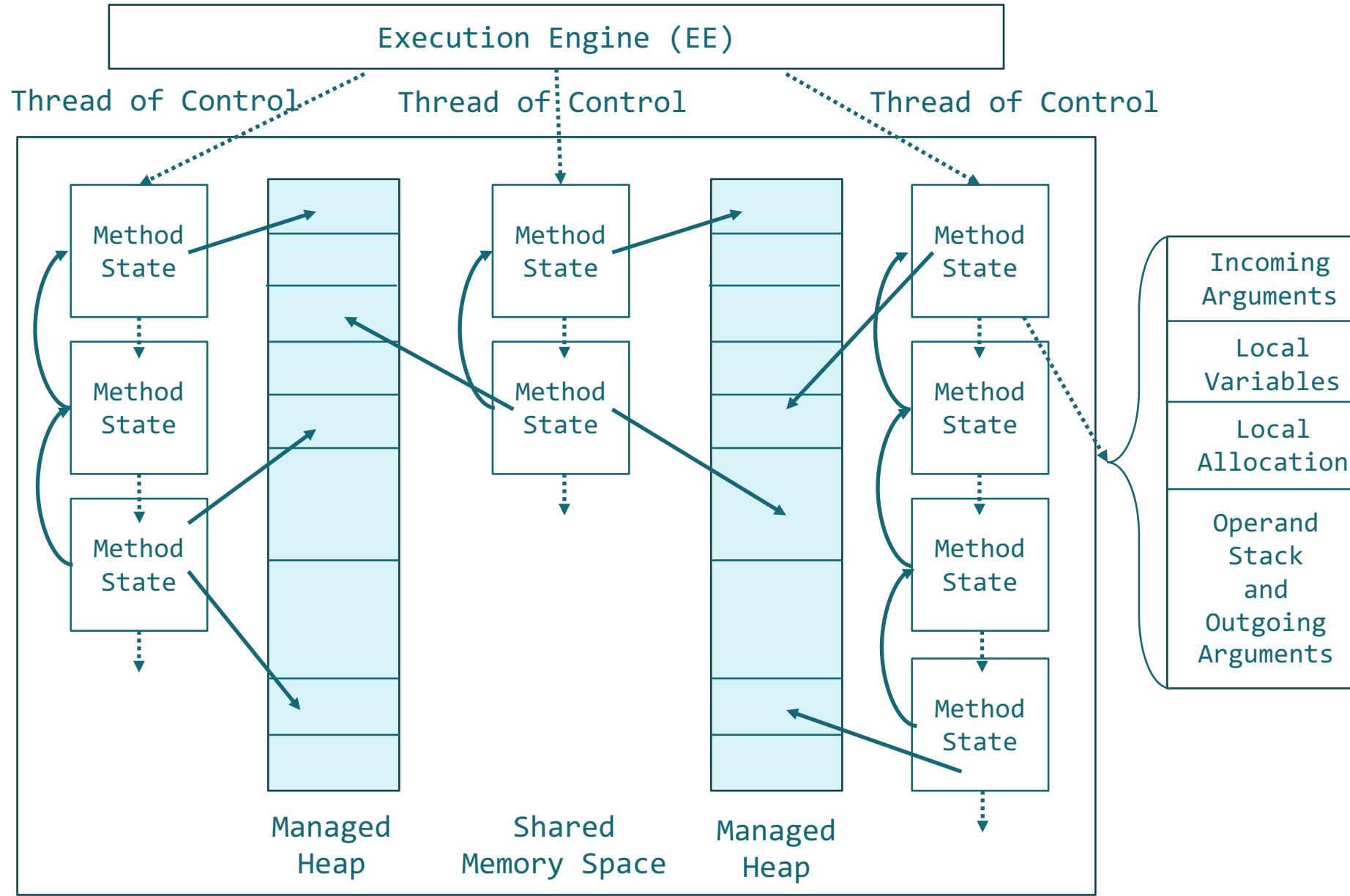
В модели выполнения кода среда CLR обладает собственной абстракцией, концептуально аналогичной потоку операционной системы. CLR определяет тип, `System.Threading.Thread`, который представляет «распределяемую» сущность в домене приложения. Объект `System.Threading.Thread` иногда называют «мягким потоком» (*soft thread*), поскольку он является конструкцией, которая не распознается операционной системой. Напротив, потоки ОС называются «жесткими потоками» (*hard thread*), потому с ними работает ОС.

- Объект мягкого потока CLR находится в одном домене приложения
- Один домен приложения может содержать много объектов мягких потоков – в текущей реализации это происходит, когда два или более жестких потока выполняют код в одном домене приложения
- В текущей реализации CLR с каждым жестким потоком ассоциирован один объект мягкого потока в соответствующем домене приложения. В том случае, когда жесткий поток выполняет код в нескольких доменах приложений, каждый домен приложения будет иметь отдельный объект мягкого потока, связанный с этим потоком. Однако, если жесткий поток не входит в домен приложения, тогда у домена приложения не будет объекта мягкого потока, представляющего собой жесткий поток.

## Runtime infrastructure. Threads



# Machine Model State (ECMA 335)



# Assemblies .NET

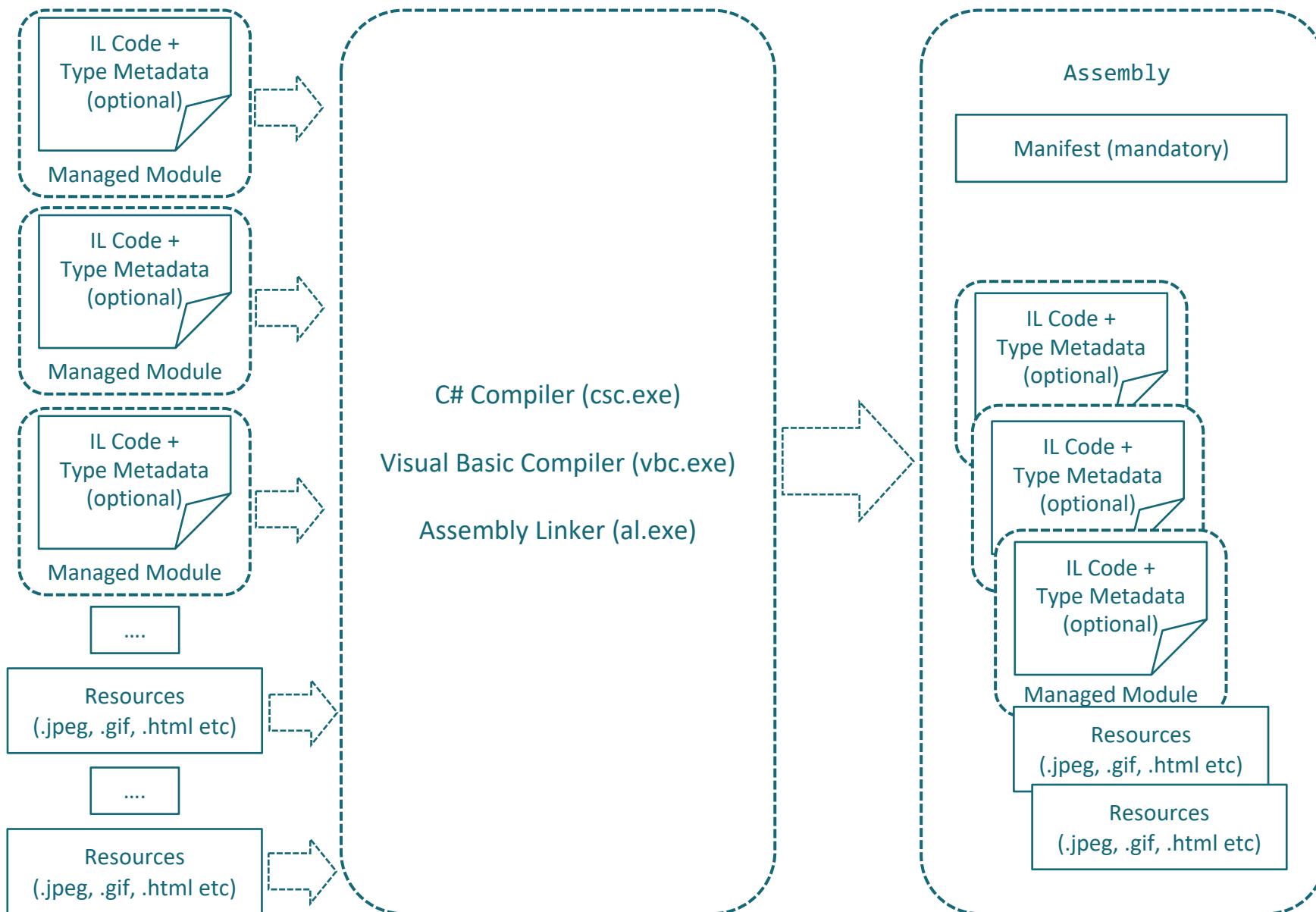
Сборка - логическая группировка одного или нескольких управляемых модулей и файлов ресурсов

Сборка - самая маленькая единица с точки зрения повторного использования, безопасности и управления версиями

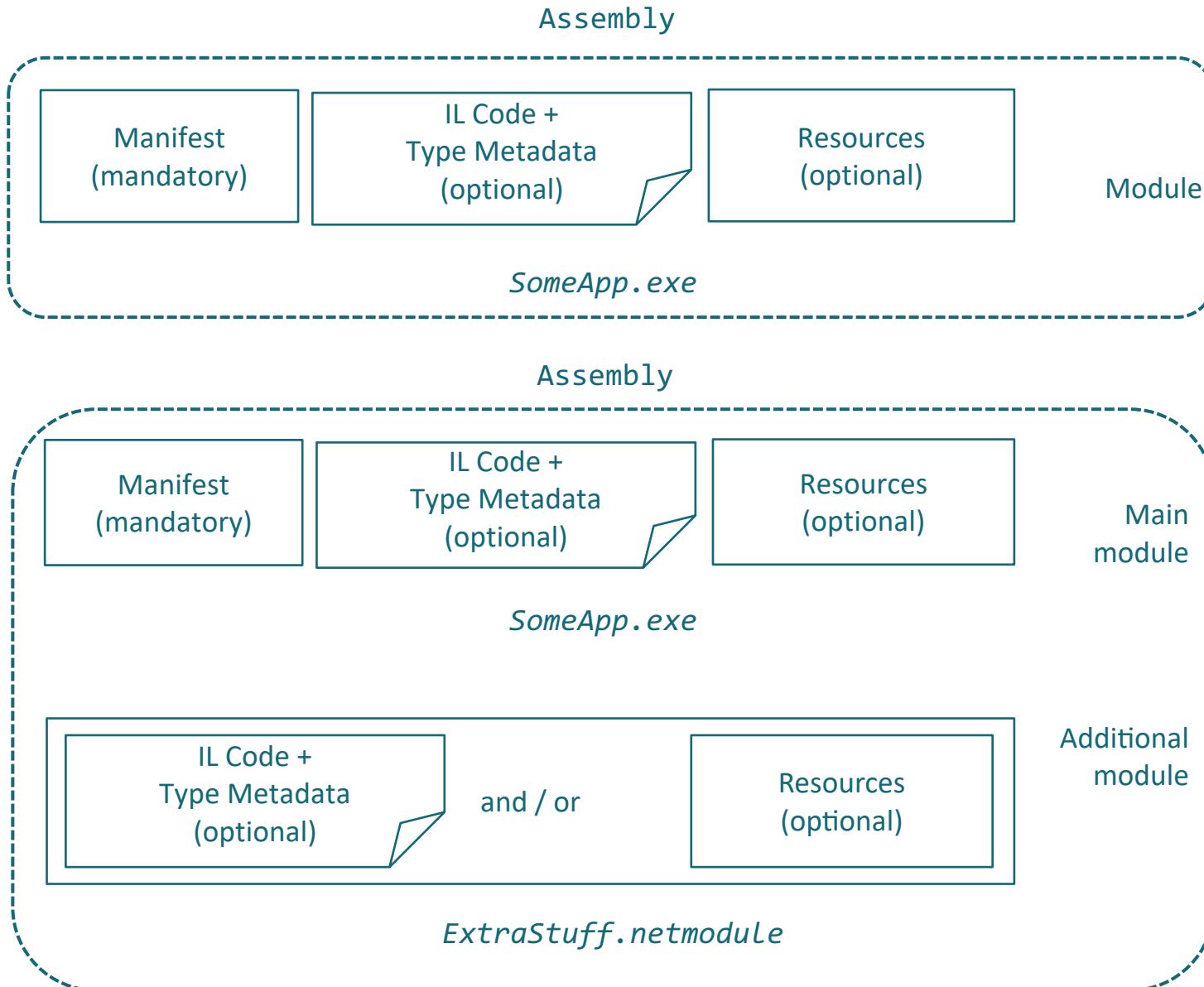
Некоторые характеристики сборки:

- Сборка определяет границы типов – в ней определены повторно используемые типы
- Сборка помечена номером версии
- Со сборкой может быть связана информация безопасности
- Сборки являются самоописываемыми
- Сборки поддаются конфигурированию

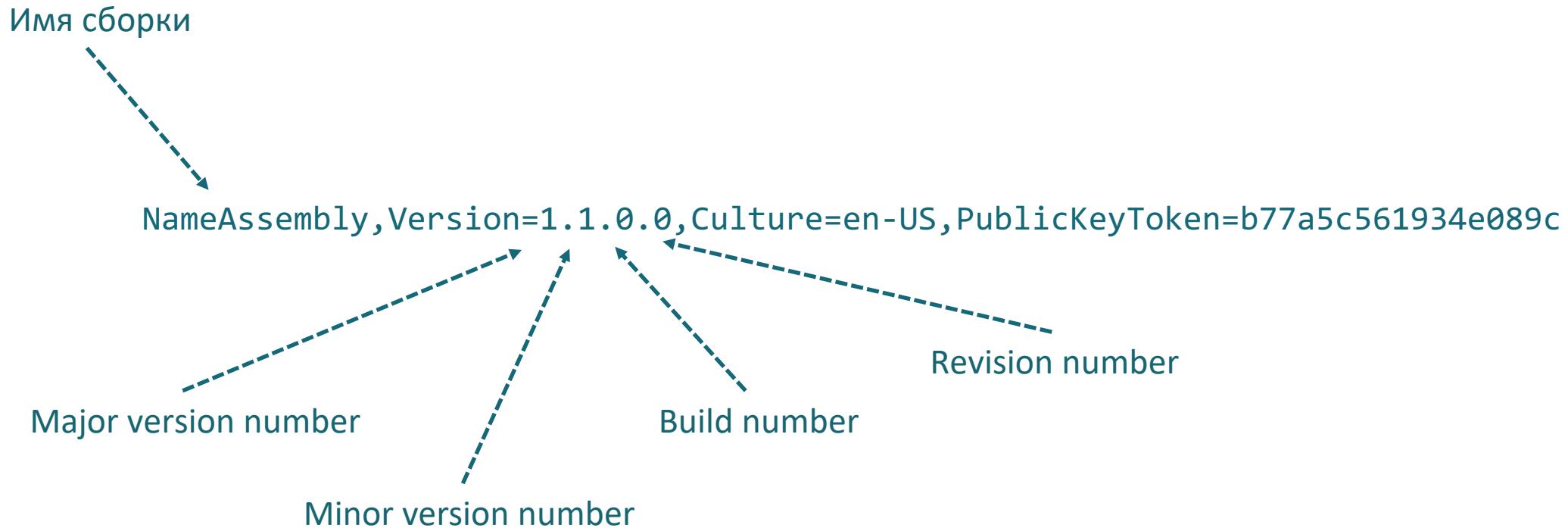
# Assemblies .NET



# Assemblies .NET



## Assemblies .NET

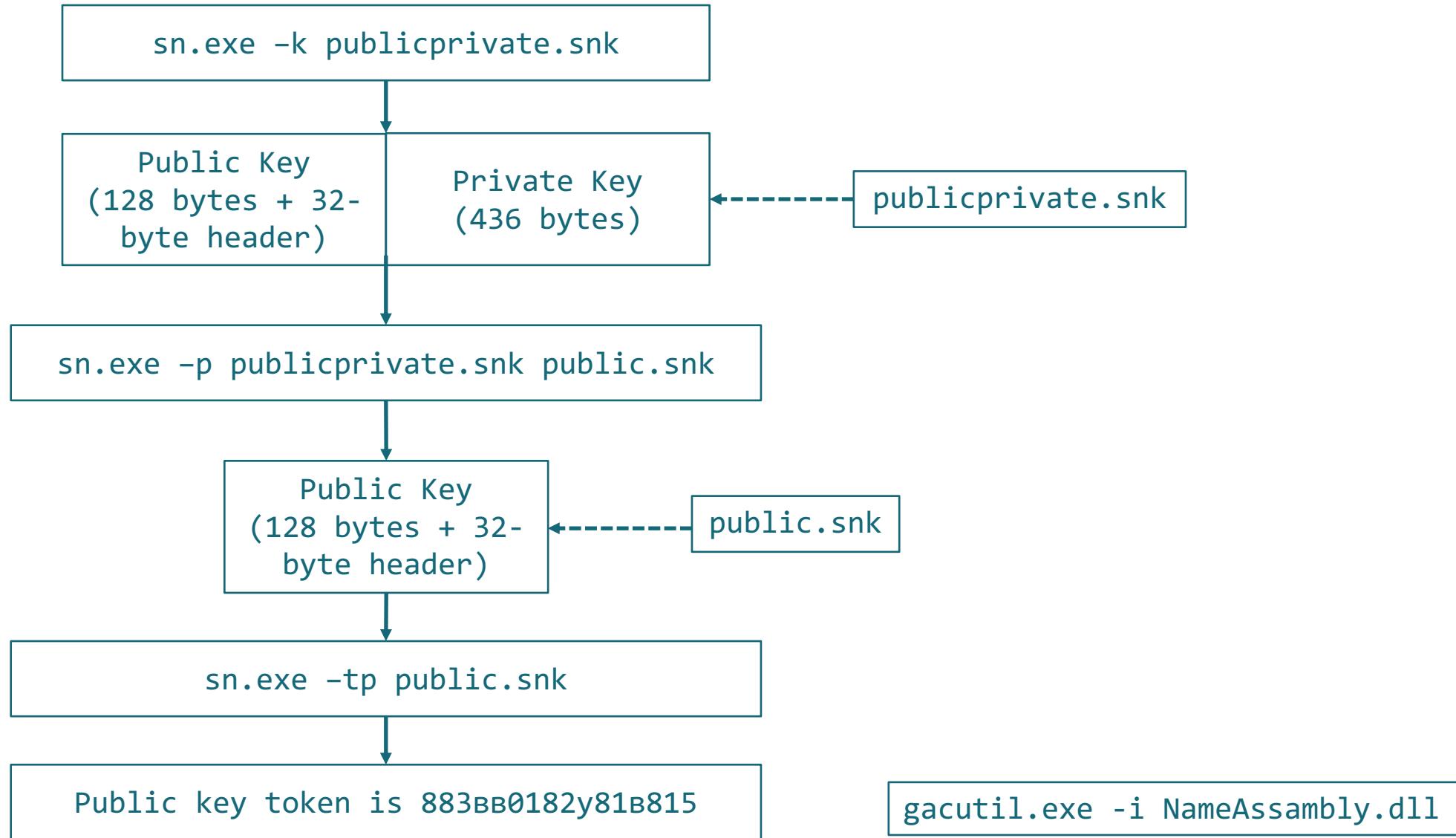


## Подписание сборки:

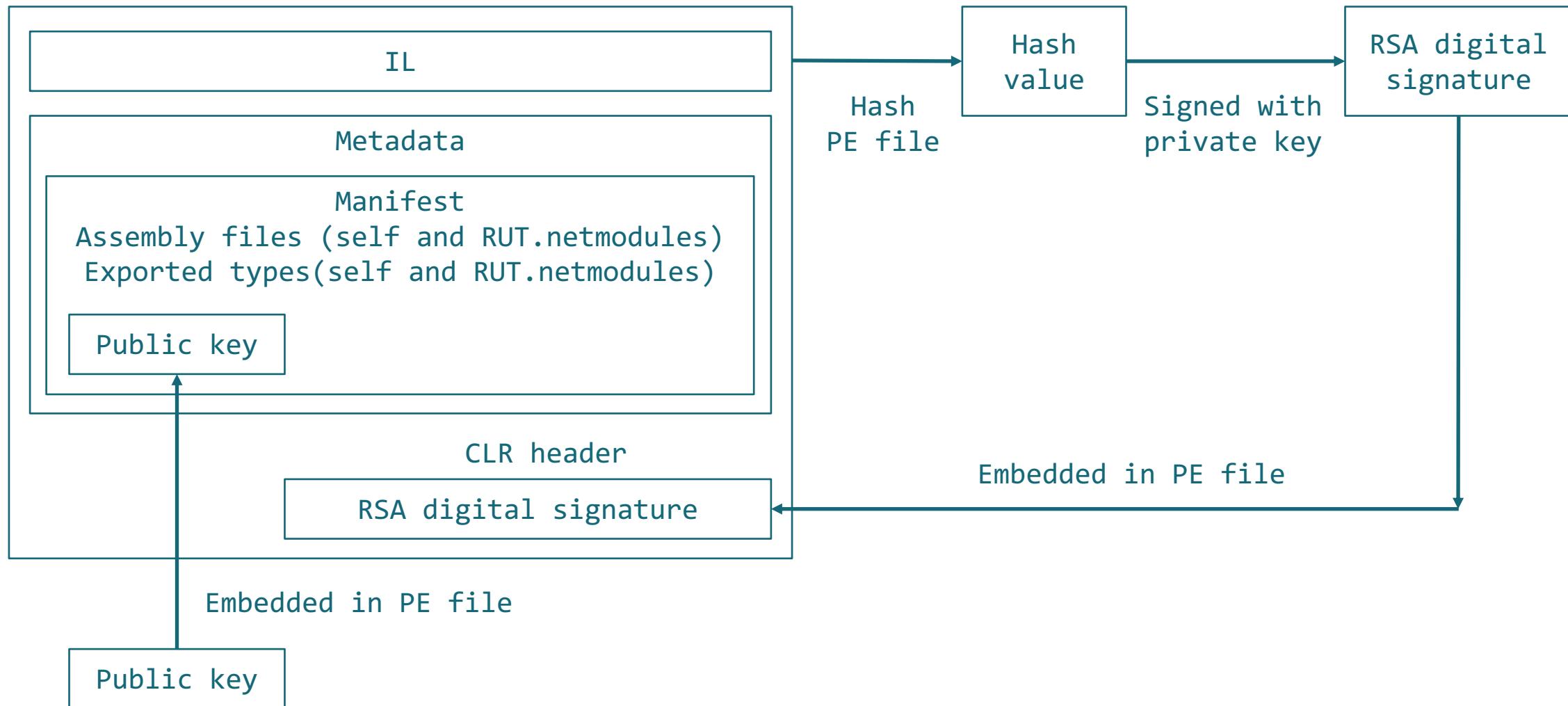
- Защищает сборки от модификаций
- Позволяет включать подписанную сборку в глобальный кэш сборок (GAC), позволяя ее использование другими приложениями
- Гарантирует, что имя сборки является уникальным

| Тип сборки                | Закрытое развертывание | Глобальное развертывание |
|---------------------------|------------------------|--------------------------|
| Сборка с нестрогим именем | Да                     | Нет                      |
| Сборка со строгим именем  | Да                     | Да                       |

## Assemblies .NET



# Assemblies .NET



## Disassemblers. Ildasm.exe. Reflector

- ildasm.exe – утилита, поставляемая в составе пакета .NET Framework SDK, позволяет загружать любую сборку .NET и изучать ее содержимое, в том числе ассоциируемый с ней манифест, CIL-код и метаданные
- .NET Reflector – платная утилита для Microsoft .NET, комбинирующая браузер классов, статический анализатор и декомпилятор
- dotPeek – бесплатный .NET декомпилятор и браузер сборок
- ILSpy – open-source .NET декомпилятор и браузер сборок

## .NET Framework Tools

- Caspol.exe
- Makecert.exe
- Ildasm.exe
- Gacutil.exe
- Ngen.exe
- Sn.exe

# Demo

## .NET Core Versions

| Version                                     | Status        | Latest release | Latest release date |
|---|---------------|----------------|---------------------|
| <a href="#">.NET Core 3.0</a>               | Preview ⓘ     | 3.0.0-preview6 | 2019-06-12          |
| <a href="#">.NET Core 2.2</a> (recommended) | Current ⓘ     | 2.2.5          | 2019-05-21          |
| <a href="#">.NET Core 2.1</a>               | LTS ⓘ         | 2.1.11         | 2019-05-21          |
| <a href="#">.NET Core 2.0</a>               | End of life ⓘ | 2.0.9          | 2018-07-10          |
| <a href="#">.NET Core 1.1</a>               | LTS ⓘ         | 1.1.13         | 2019-05-14          |
| <a href="#">.NET Core 1.0</a>               | LTS ⓘ         | 1.0.16         | 2019-05-14          |

<https://dotnet.microsoft.com/download/dotnet-core>

# .NET Core Tools

v2.2.5

Current ⓘ

Released 2019-05-21

[Release notes](#)

Supports C# 7.3

Supports F# 4.6

Supports Visual Basic 15.9

Supports Visual Studio 2019 (v16.1)

Supports Visual Studio 2019 for Mac (v8.1 & v8.2)

Included in 16.1.0

ASP.NET Core IIS Module 12.2.19109.5

SDK 2.2.300

Windows

- .NET Core Installer: [x64](#) | [x86](#)
- .NET Core Binaries: [x64](#) | [x86](#) | [ARM32](#)

macOS

- .NET Core Installer: [x64](#)
- .NET Core Binaries: [x64](#)

Linux

- Package Manager Instructions: [x64](#)
- .NET Core Binaries: [x64](#) | [ARM32](#) | [ARM64](#) | [x64 Alpine](#) | [RHEL 6 x64](#)

Other

- Checksums: [Checksums](#)

Runtime 2.2.5

Windows

- ASP.NET Core/.NET Core: [Runtime & Hosting Bundle](#)
- ASP.NET Core Installer: [x64](#) | [x86](#)
- ASP.NET Core Binaries: [x64](#) | [x86](#) | [ARM32](#)
- .NET Core Installer: [x64](#) | [x86](#)
- .NET Core Binaries: [x64](#) | [x86](#) | [ARM32](#)

macOS

- ASP.NET Core Binaries: [x64](#)
- .NET Core Installer: [x64](#)
- .NET Core Binaries: [x64](#)

Linux

- Package Manager Instructions: [x64](#)
- ASP.NET Core Binaries: [x64](#) | [ARM32](#) | [x64 Alpine](#)
- .NET Core Binaries: [x64](#) | [ARM32](#) | [ARM64](#) | [x64 Alpine](#) | [RHEL 6 x64](#)

Other

- Checksums: [Checksums](#)

<https://dotnet.microsoft.com/download/dotnet-core/2.2>

## .NET Framework vs .NET Core

|            | <b>.NET Framework</b>   | <b>.NET Core</b>  |
|------------|---|---|
| App models | <ul style="list-style-type: none"><li>- Windows Console</li><li>- Desktop (WinForms, WPF)</li><li>- WCF Service</li><li>- ASP.Net (Web Forms, MVC, Web API)</li><li>- Azure (Cloud Service, Web Jobs)</li><li>- ...</li></ul> | <ul style="list-style-type: none"><li>- Console</li><li>- UWP</li><li>- ASP.Net (MVC / Web API)</li><li>- Cloud</li></ul> |
| Libraries  | <ul style="list-style-type: none"><li>- BCL</li><li>- Windows specific</li></ul>  | <ul style="list-style-type: none"><li>- .Net Standard /BCL subset</li><li>- Own specific libs (e.g. Extensions)</li></ul> |
| Platforms  | <ul style="list-style-type: none"><li>- Windows (Server and Desktop)</li></ul>  | <ul style="list-style-type: none"><li>- Windows (from Server to IoT)</li><li>- Linux</li><li>- macOS</li></ul>            |

# Demo