

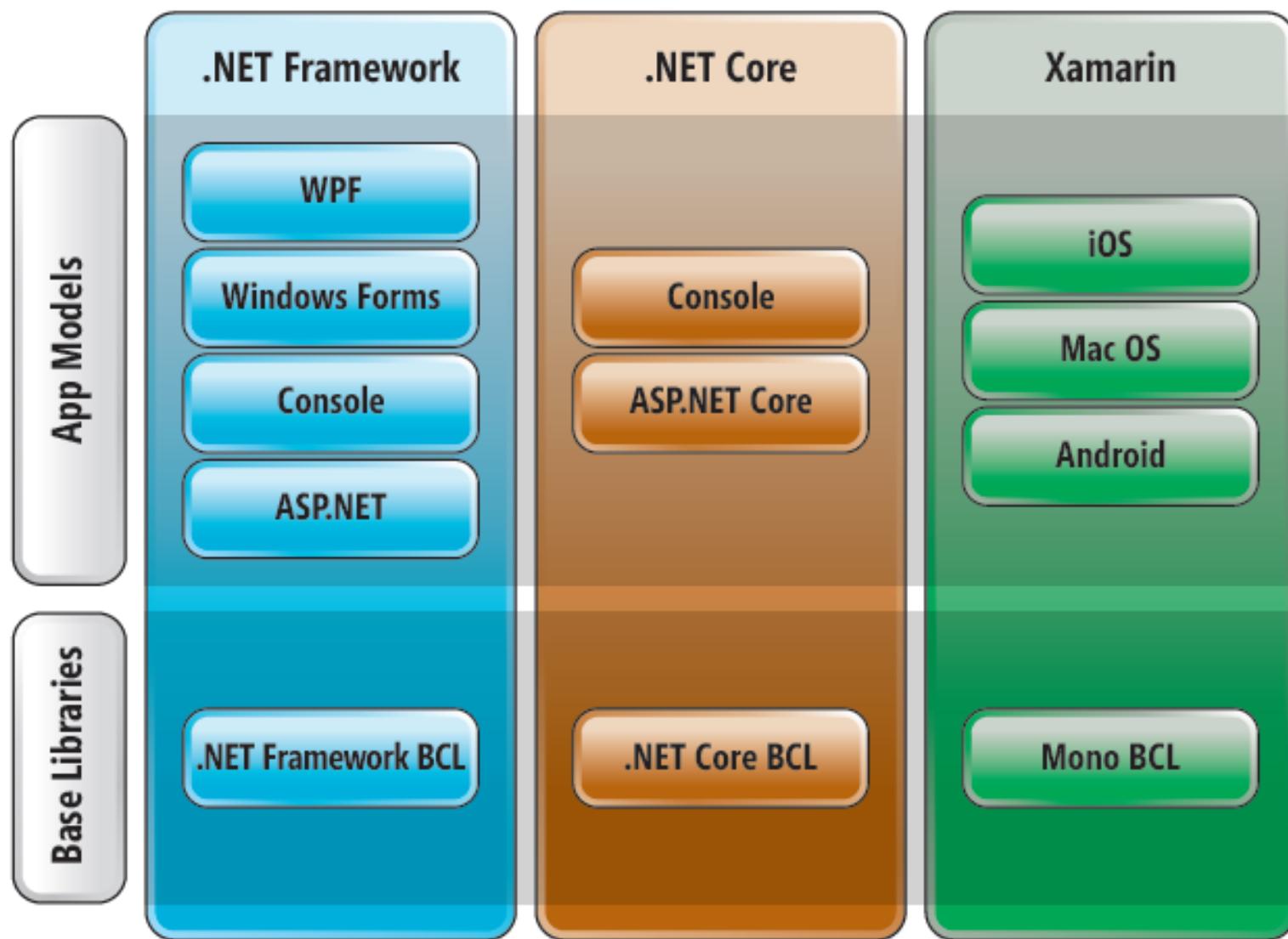


INTRODUCTION TO THE C# LANGUAGE AND THE .NET FRAMEWORK (DRAFT)

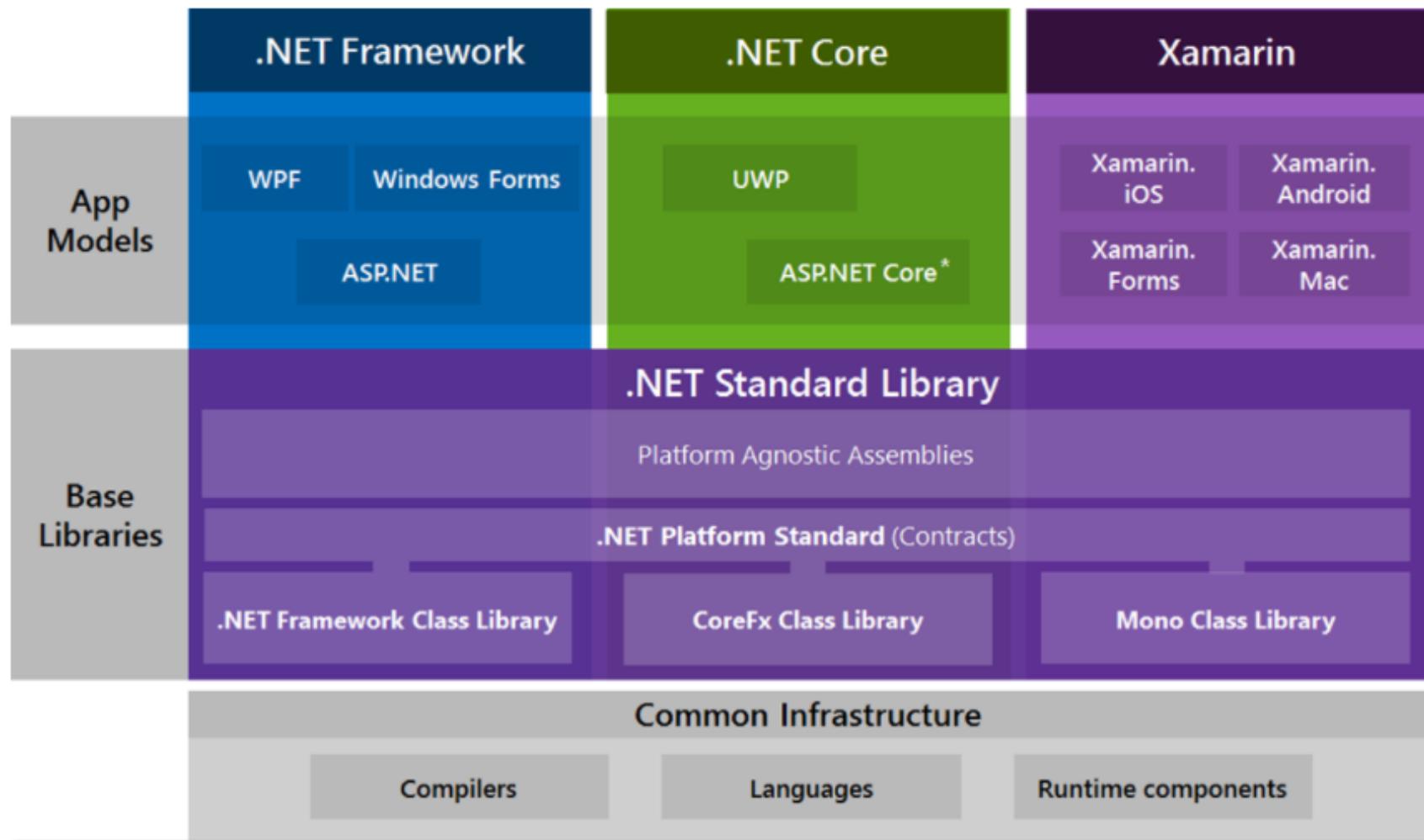
.NET LAB, MINSK

ANZHELIKA KRAVCHUK

.NET – Back to the Future



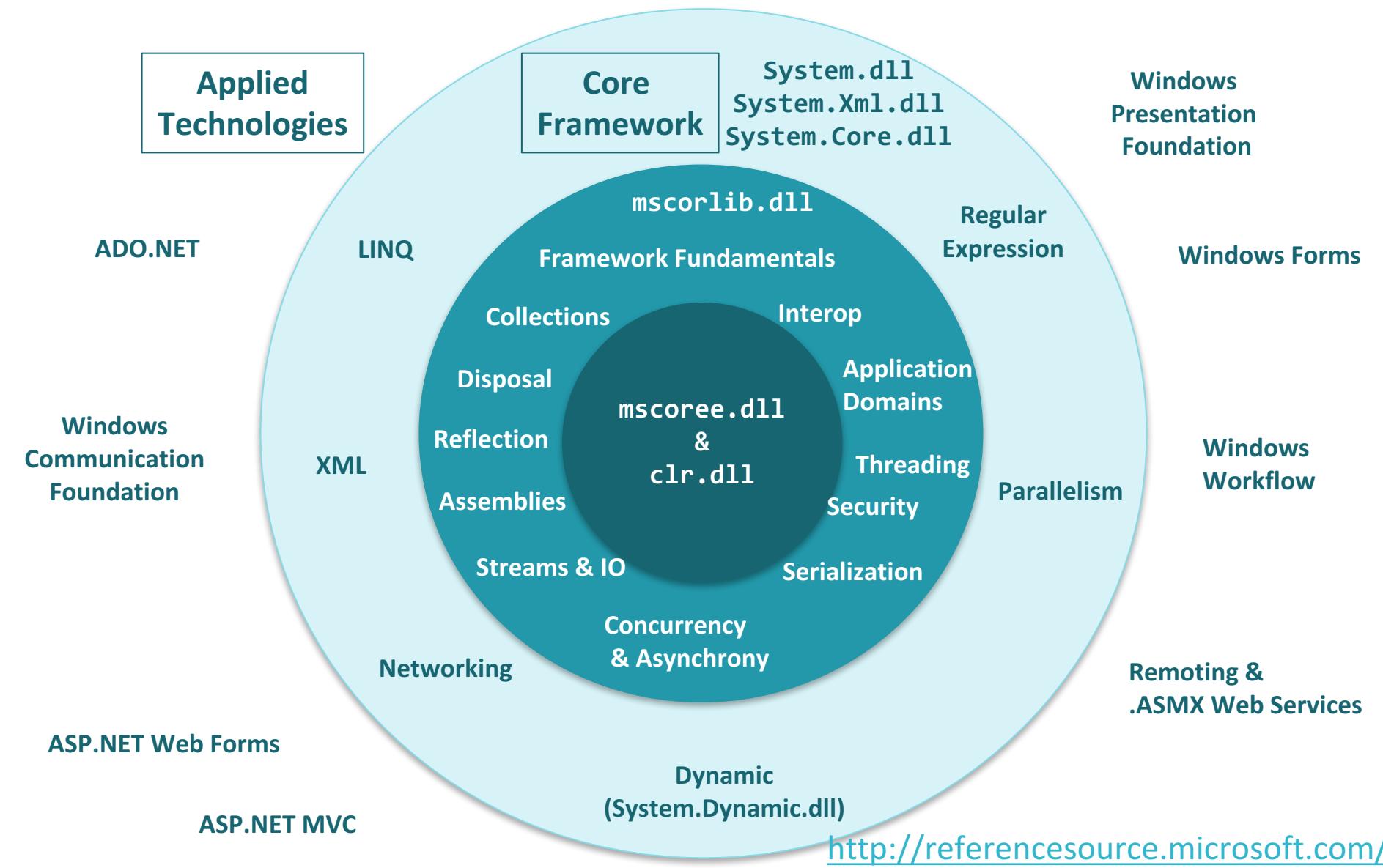
.NET – Back to the Future



.NET – Back to the Future

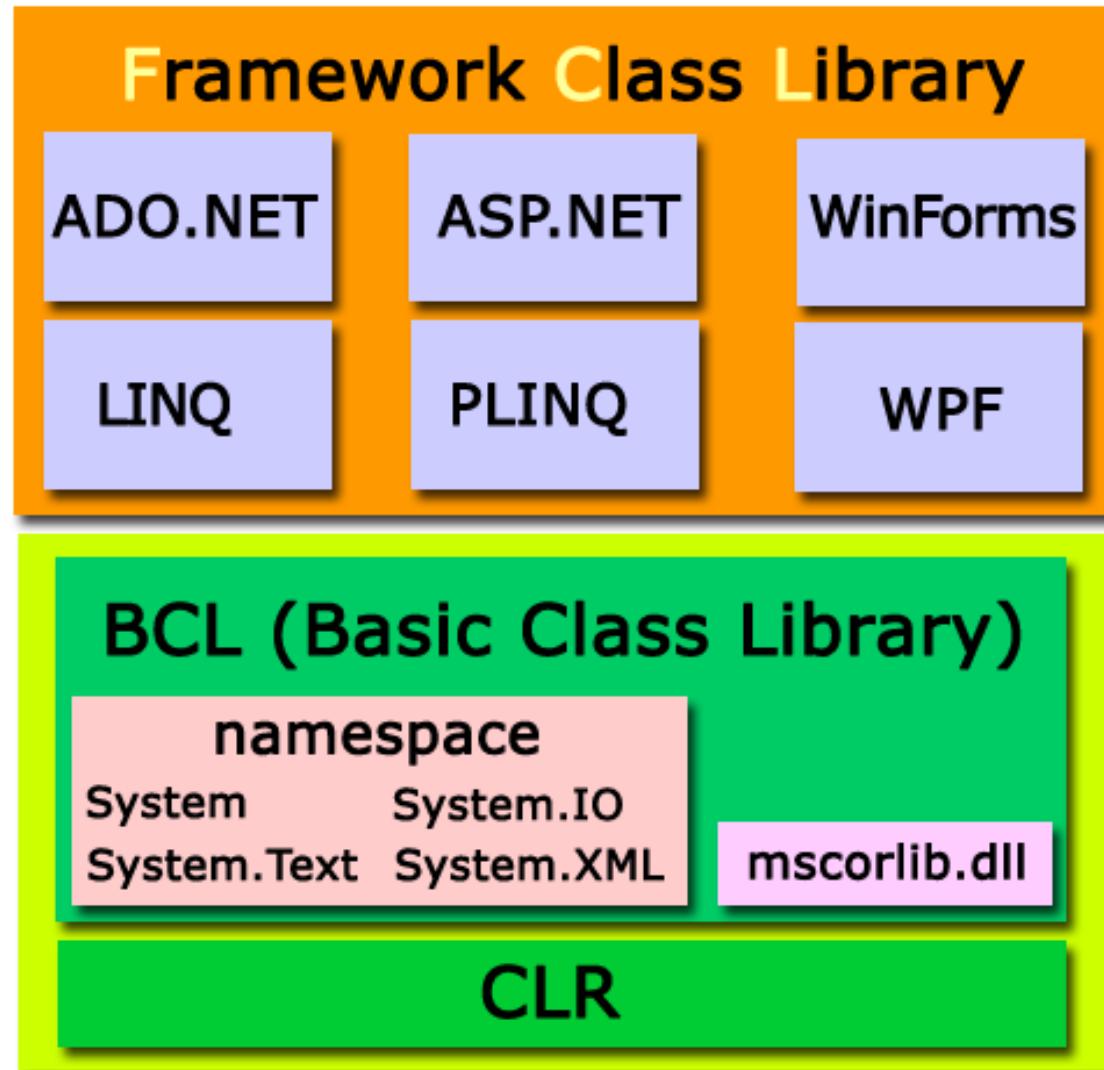
	ОС	Open Source	Назначение
.NET Framework	Windows	Нет	Создание классических Windows-приложений и веб-приложений ASP.NET для IIS.
.NET Core	Windows, Linux, macOS	Да	Создание кроссплатформенных консольных приложений, а также веб-приложений и облачных служб ASP.NET Core.
Xamarin	iOS, Android, macOS	Да	Создание мобильных приложений для iOS и Android, классических приложений для macOS.
.NET Standard	—	Да	Создание библиотек, которые можно использовать в любых реализациях .NET, в том числе .NET Framework, .NET Core и Xamarin.

Платформа .NET Framework = CLR + Libraries (BCL, FCL)



<http://referencesource.microsoft.com/>

Платформа .NET Framework = CLR + Libraries (BCL, FCL)



Основные компоненты .NET Framework - CLR & FCL. CLR

Some .NET Application

Common Language Runtime (CLR)

Operation system

Общеязыковая среда выполнения (Common Language Runtime, CLR) – среда выполнения, которая подходит для разных языков программирования. Основные возможности CLR (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация) доступны в любых языках программирования, использующих эту среду.

Основные компоненты .NET Framework - CLR & FCL. 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 (including 4.5.1 and 4.5.2)	4
4.6 (including 4.6.1 and 4.6.2)	4
4.7 (including 4.7.1)	4

Платформа .NET Framework. CLR vs CLI

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 состоит из 4 аспектов:

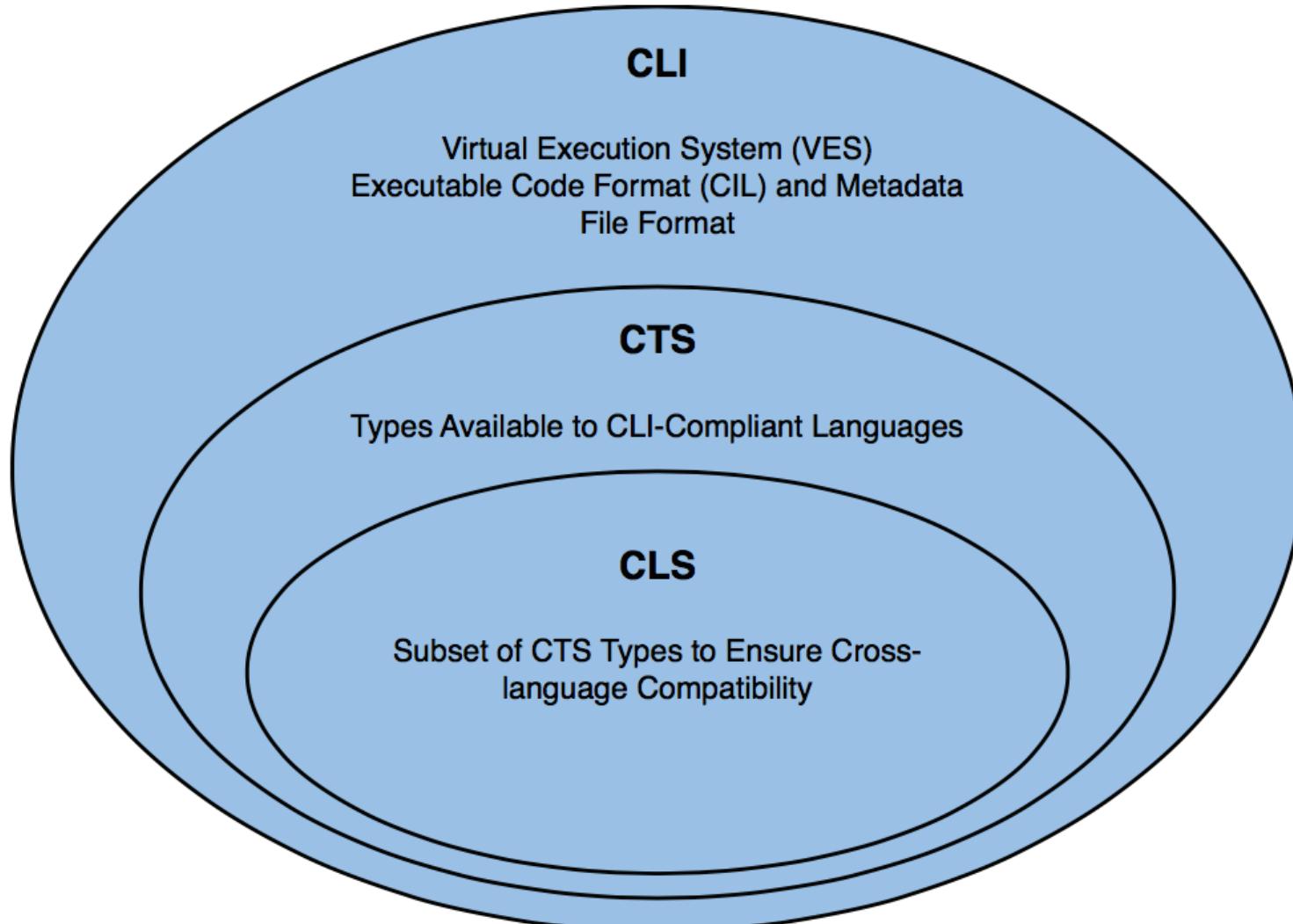
Common Type System (CTS) - Набор типов и операций которые используются во многих языках программирования.

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.

Платформа .NET Framework. CLR vs CLI



<https://www.gnu.org/software/dotgnu/pnet-install.html>

<http://www.mono-project.com/>

Common Type System, Common Language Specification

Спецификация CTS описывает способ определения и поведение типов

Стандарт CTS вместе с другими частями .NET Framework (форматы файлов, метаданные, IL, механизм вызова P/Invoke и т. д.) называется CLI (Common Language Infrastructure) и определяется спецификацией ECMA-335

Спецификация CLS (Common Language Specification) перечисляет минимальный набор возможностей, которые должны поддерживаться компилятором для генерирования типов, совместимых с другими компонентами, написанными на других CLS-совместимых языках на базе CLR

Common Type System

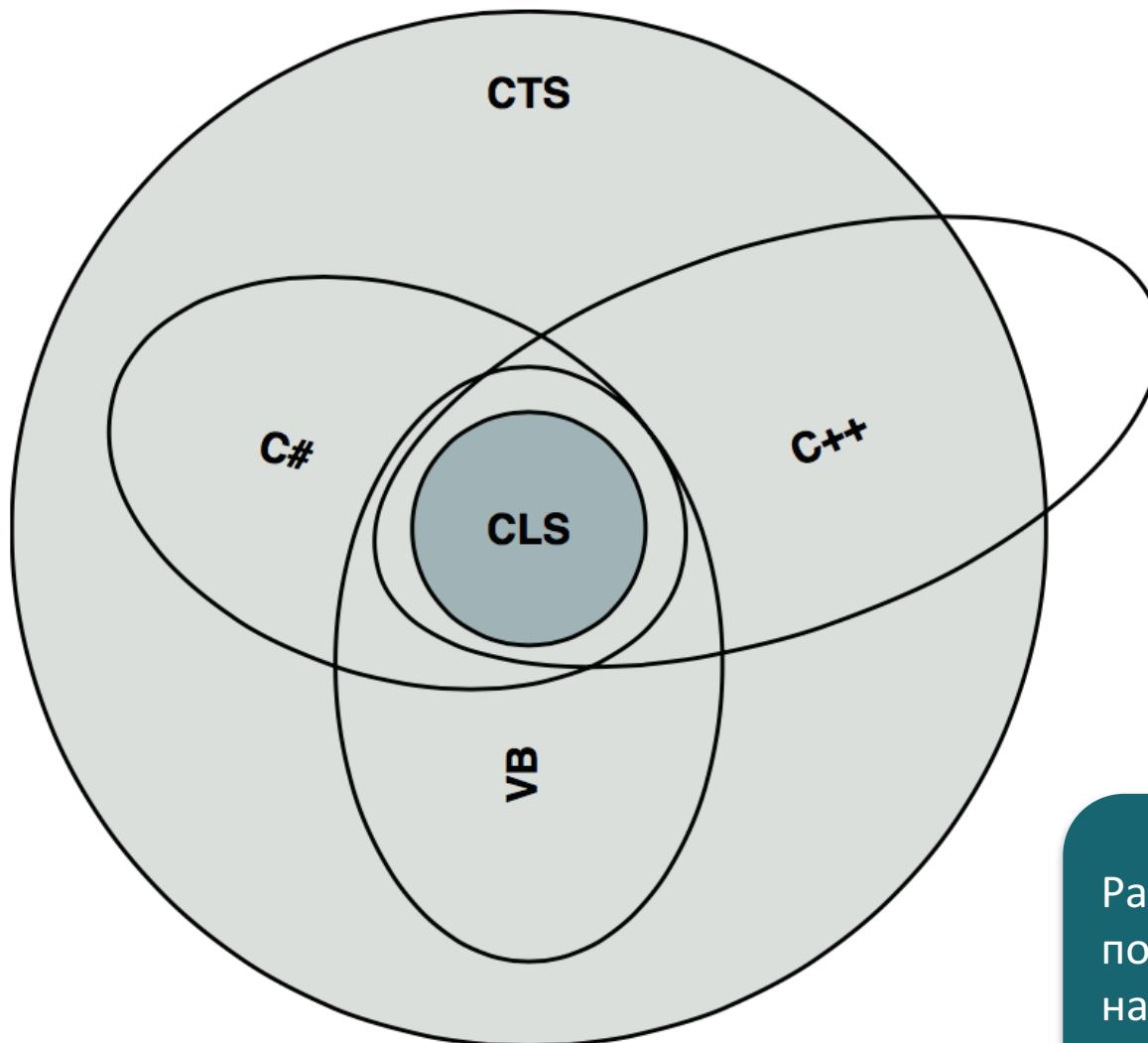
ECMA-335

ECMA-334

Common Language Specification

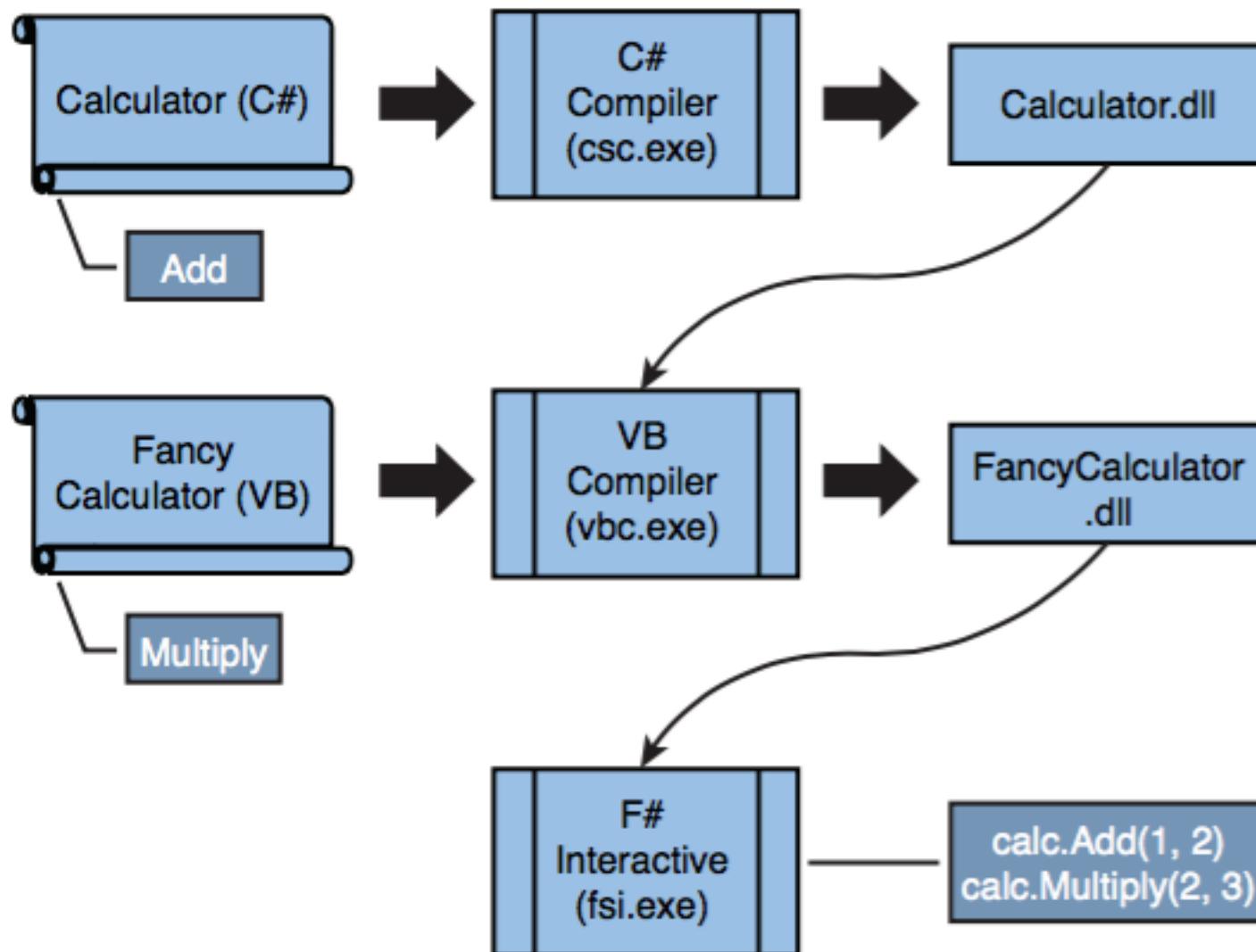
<http://www.ecma-international.org/>

Common Type System, Common Language Specification



Разные языки поддерживают подмножество CLR/CTS и надмножество CLS (возможно, разные подмножества)

Common Type System, Common Language Specification



CLR, управляемые модули

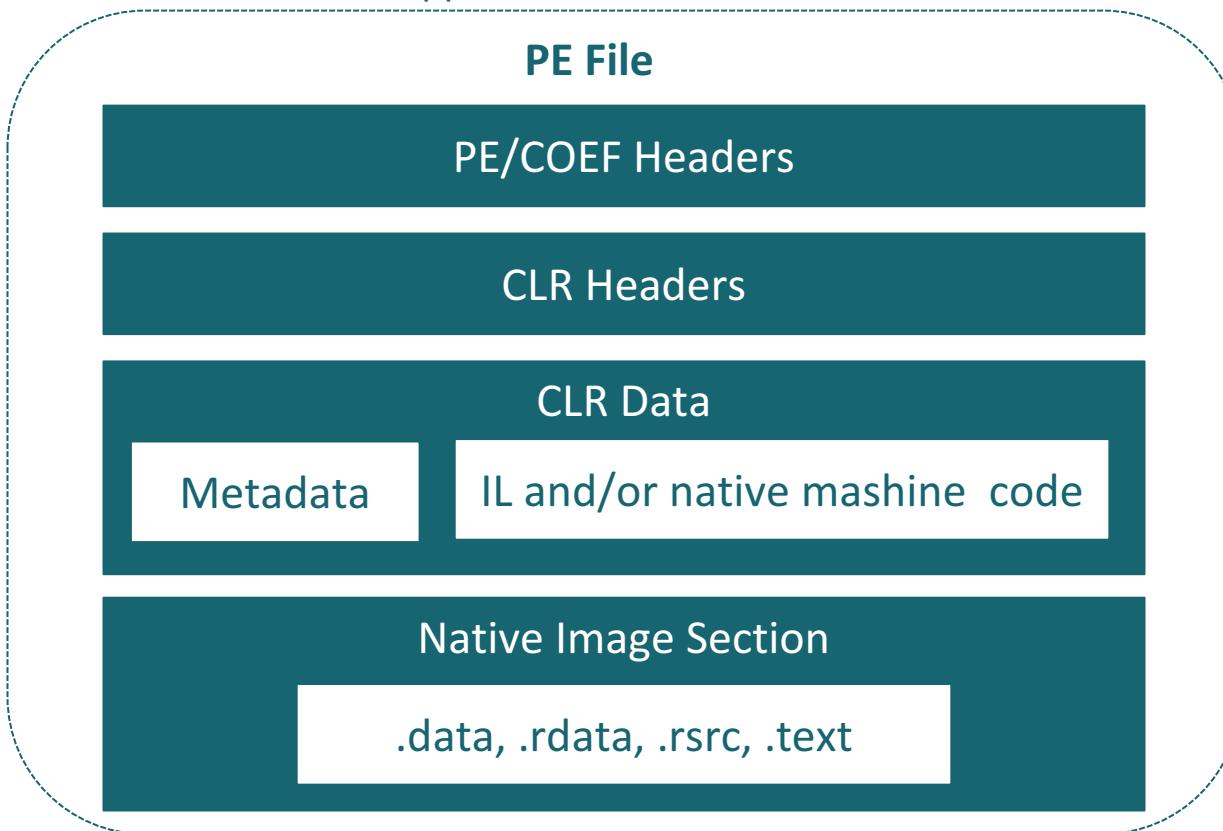
Для создания, развертывания и поиска компонентов CLR имеет собственный набор концепций и технологий, принципиально отличающихся от тех, которые используются в COM, Java или Win32.

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

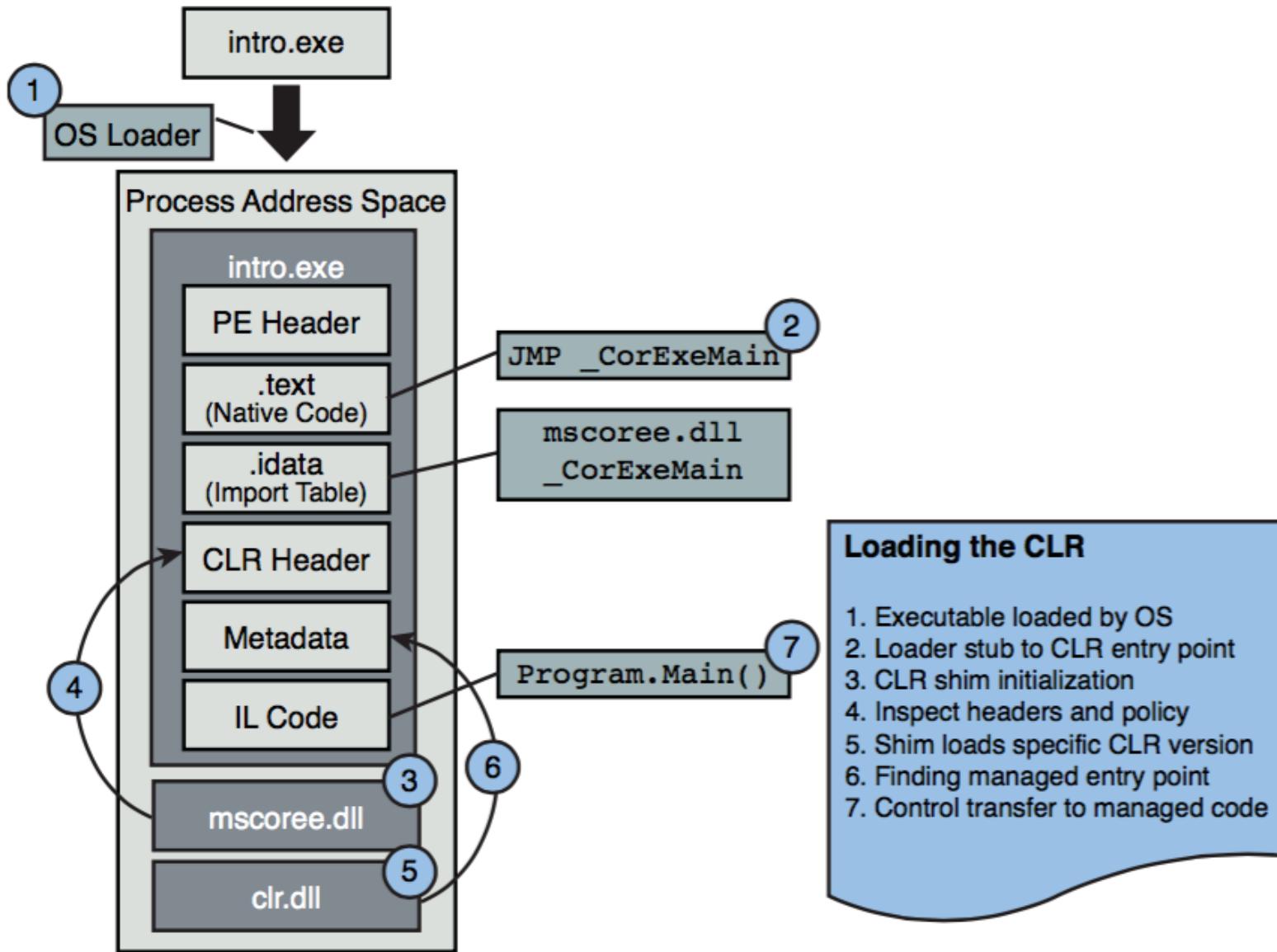
Чтобы загрузчик операционной системы распознал исполняемый файл как действительный, он должен иметь структуру, определенную в формате файла PE/COFF (Portable Executable, Common Object File Format).

CLR, управляемые модули, формат модуля CLR

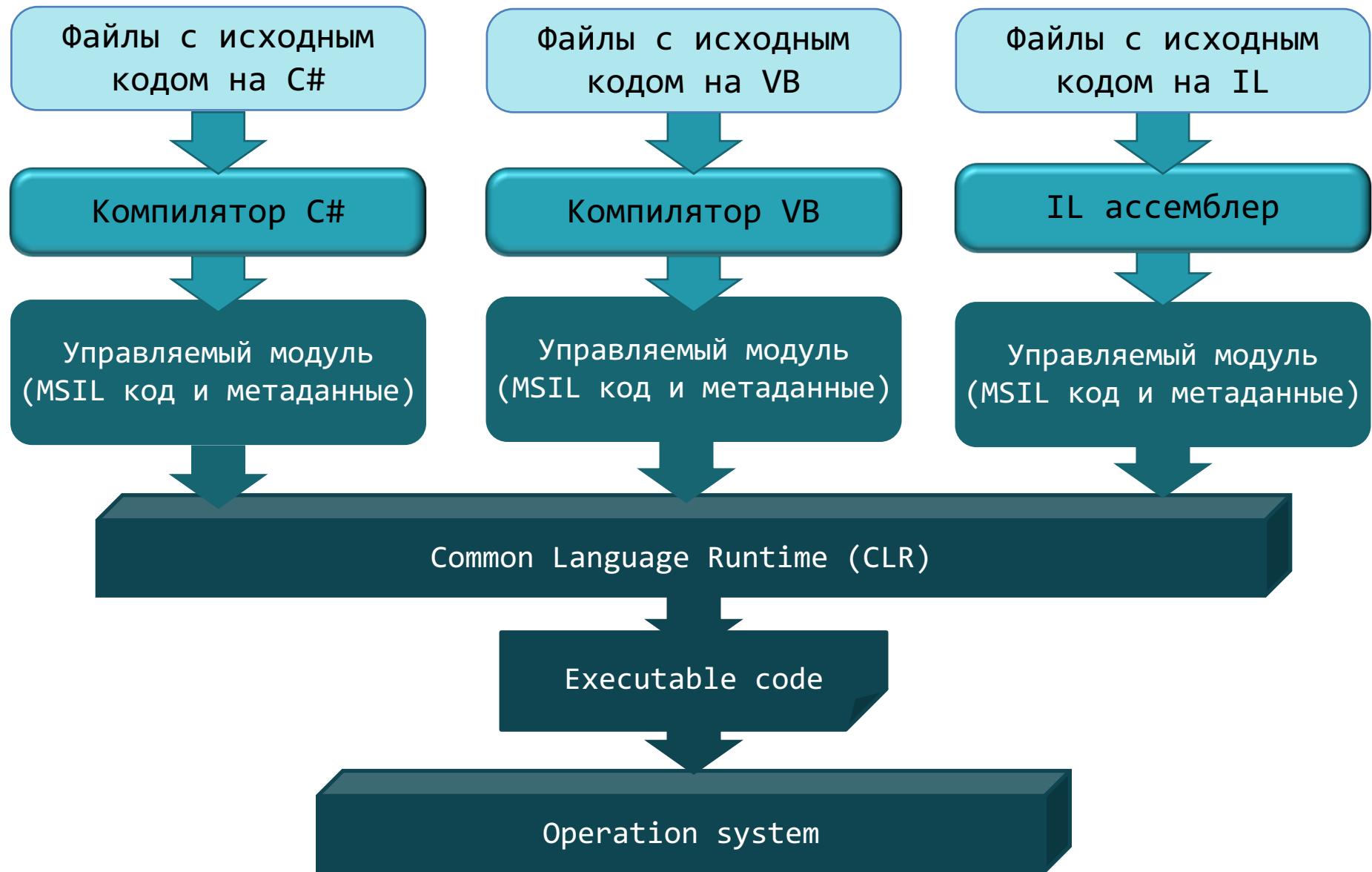
Модуль CLR содержит код, метаданные и ресурсы. Код обычно хранится в формате CIL, хотя он также может быть сохранен в виде машинных инструкций, специфичных для процессора. Метаданные модуля описывают типы, определенные в модуле, включая имена, отношения наследования, сигнатуры методов и информацию о 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.



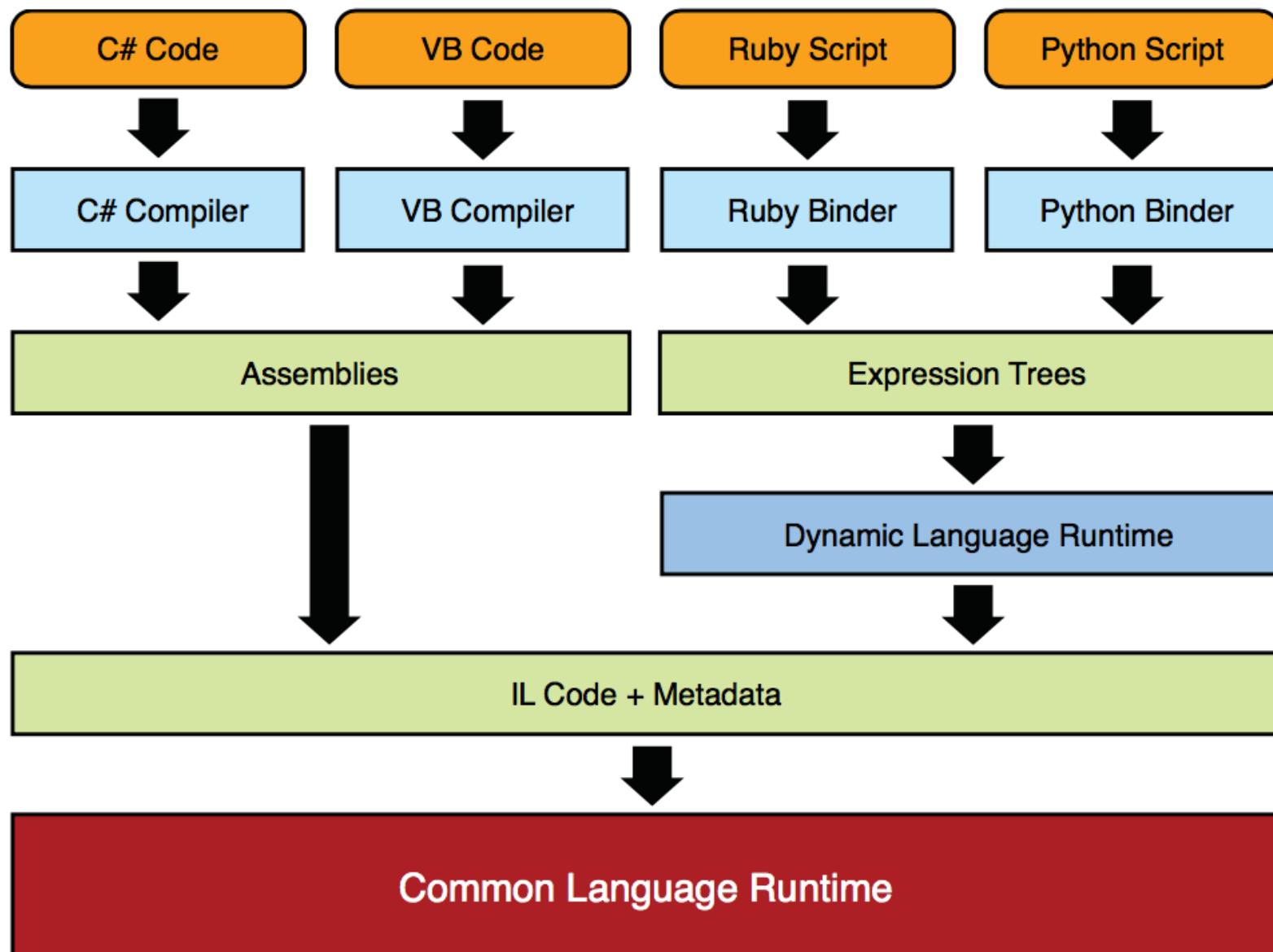
CLR, управляемые модули



Управляемые модули, MSIL код и метаданные



Управляемые модули, MSIL код и метаданные



Управляемые модули, MSIL код и метаданные

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

IL и верификация

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.

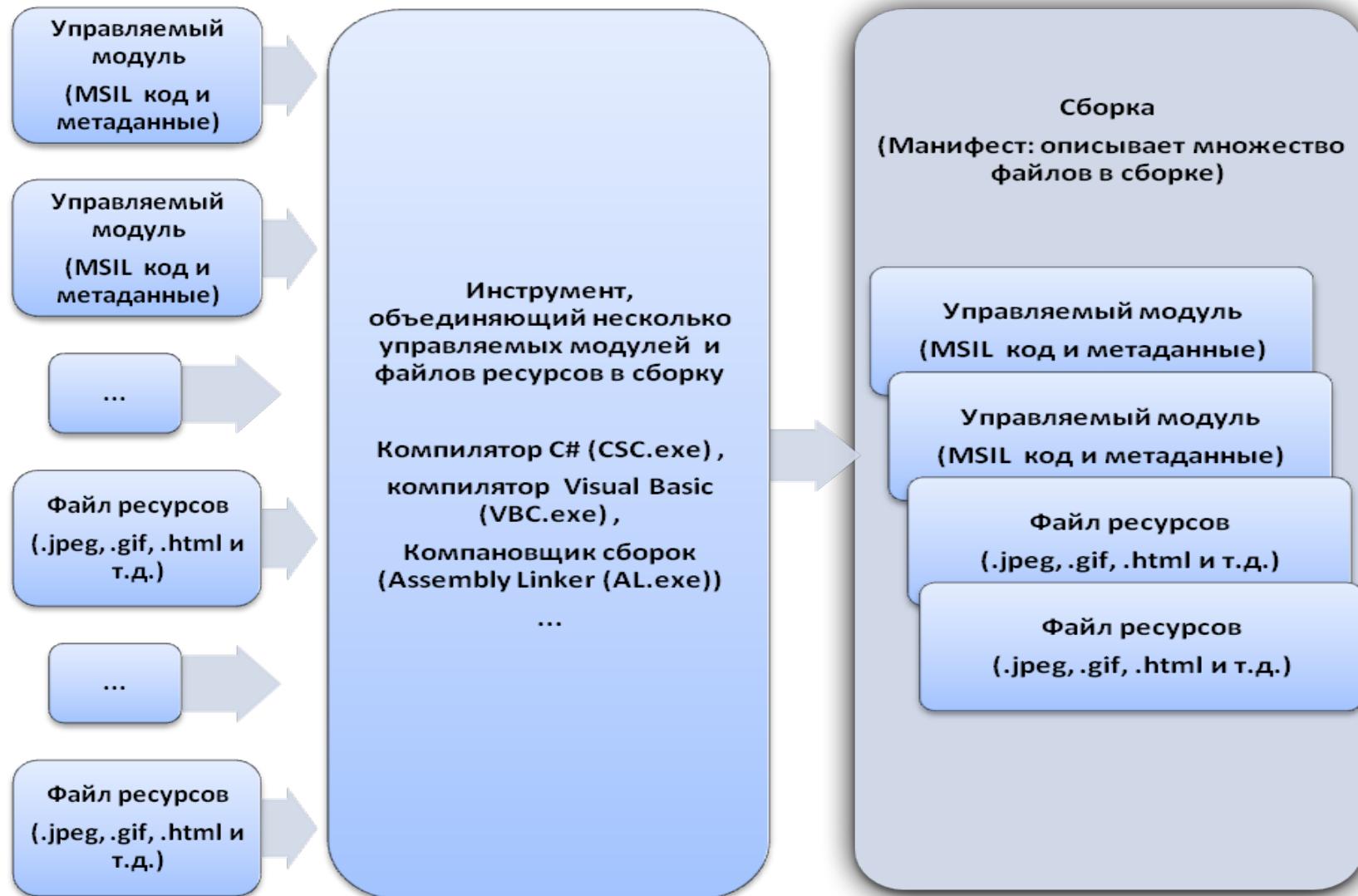
IL-код обеспечивает безопасность приложения и его устойчивость перед ошибками – в процессе компиляции IL в машинные инструкции CLR выполняется процедура, называемая *верификацией* — анализ высокоуровневого кода IL и проверка безопасности всех операций.

Сборки в .NET

Логическая группировка одного или нескольких управляемых модулей и файлов ресурсов

Самая маленькая единица с точки зрения повторного использования, безопасности и управления версиями

Сборки в .NET



Некоторые характеристики сборки:

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

Сборки в .NET

Номер версии сборки состоит из следующих компонент:

- Главный номер версии (Major version number)
- Второстепенный номер версии (Minor version number)
- Номер сборки (Build number)
- Номер редакции (Revision number)

```
NameAssembly, Version=1.1.0.0, Culture=en, PublicKeyToken=null
```

Сборки в .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

Сборки в .NET

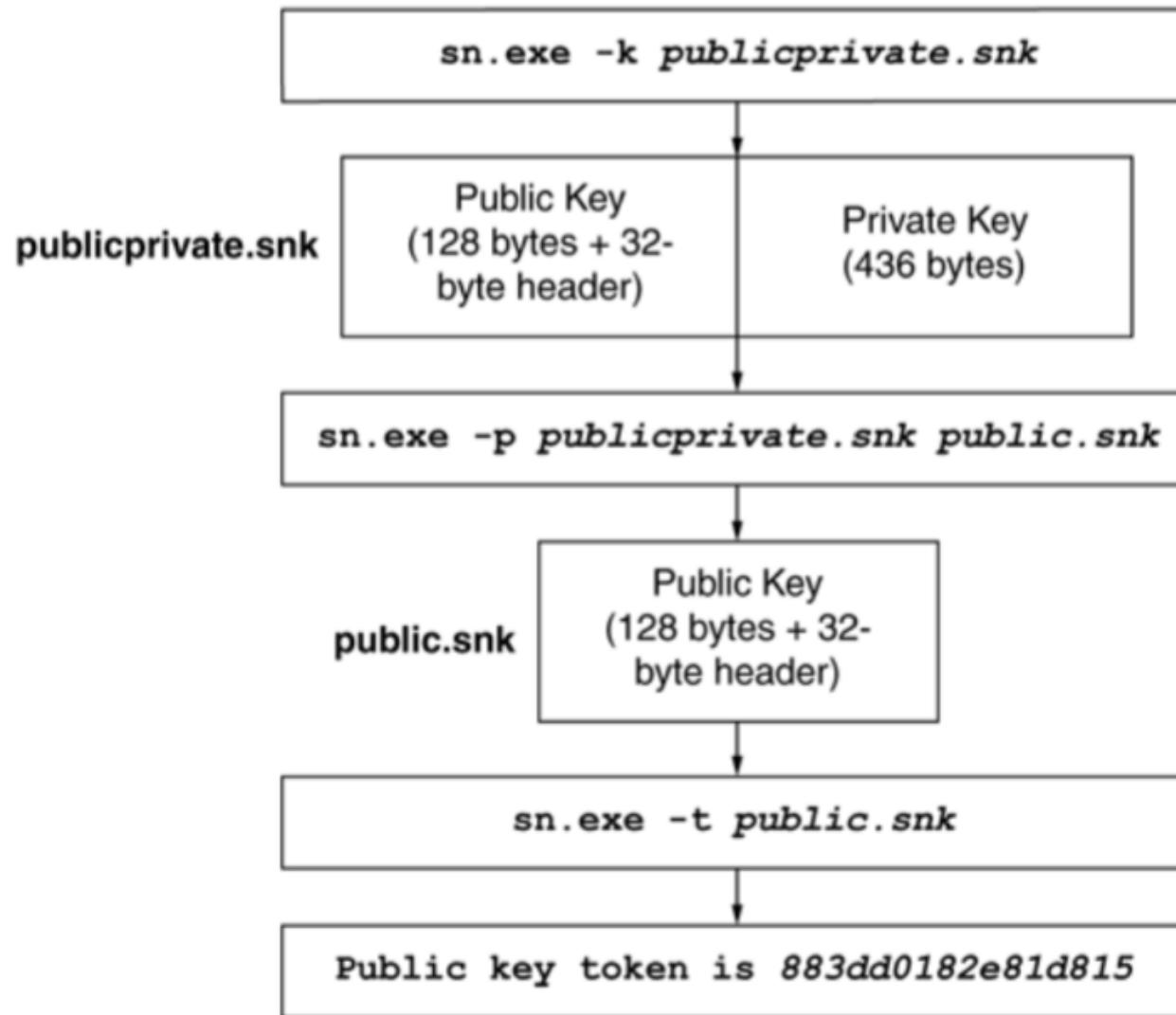
```
sn -k keyPair.snk
```

```
sn -p keyPair.snk publicKey.snk
```

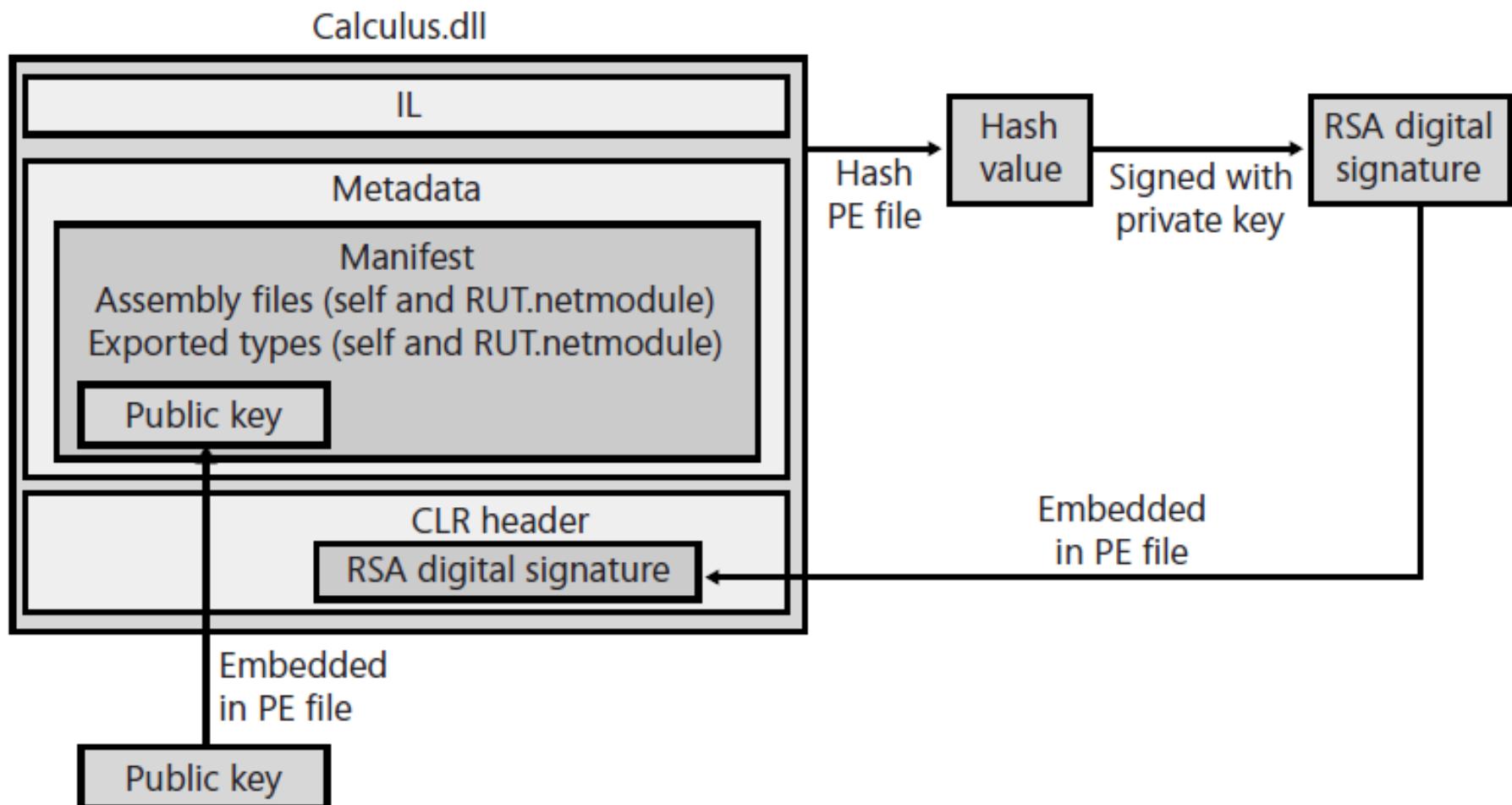
```
sn -tp keyPair.snk
```

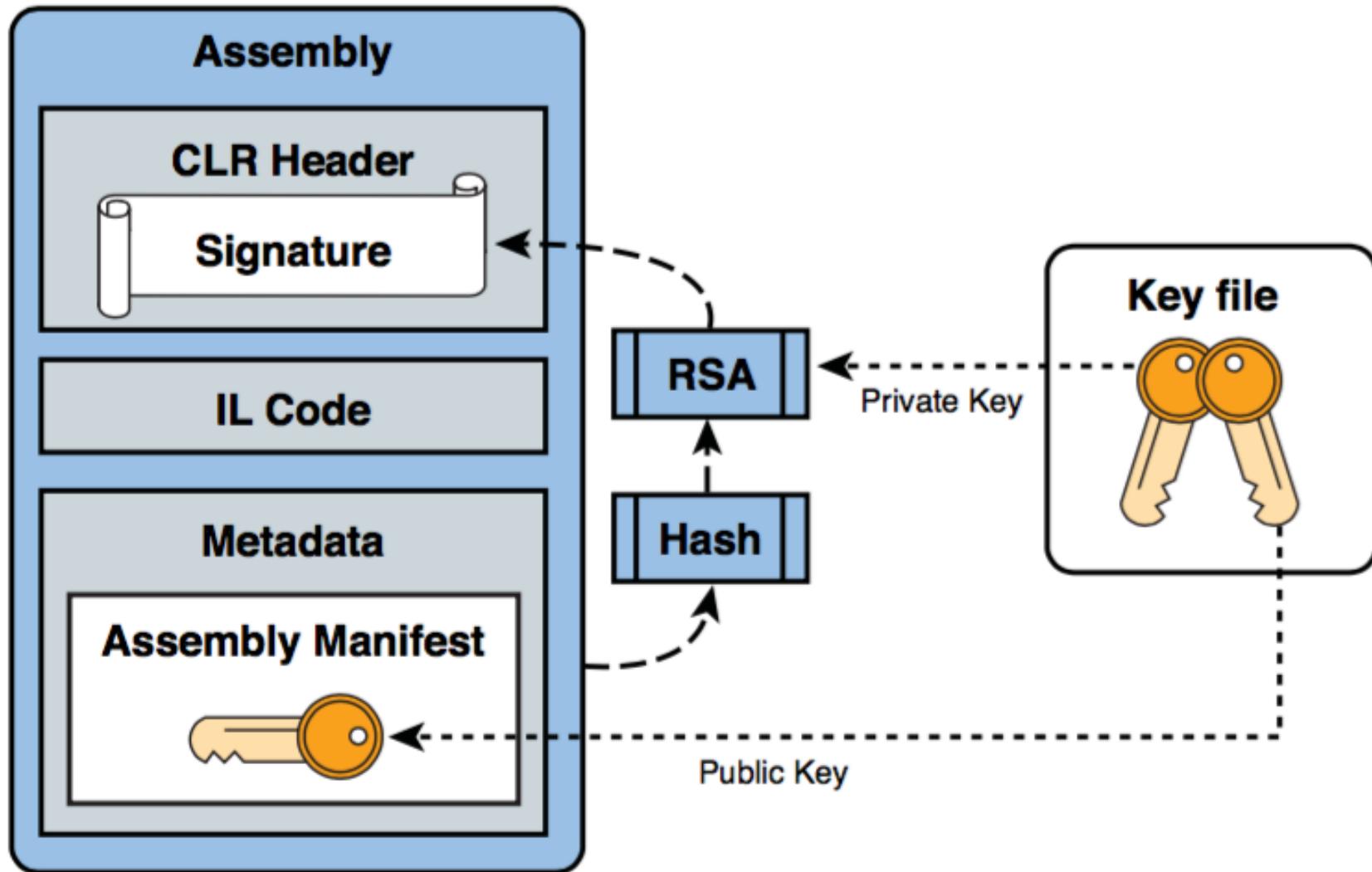
```
gacutil.exe -i NameAssambly.dll
```

Сборки в .NET



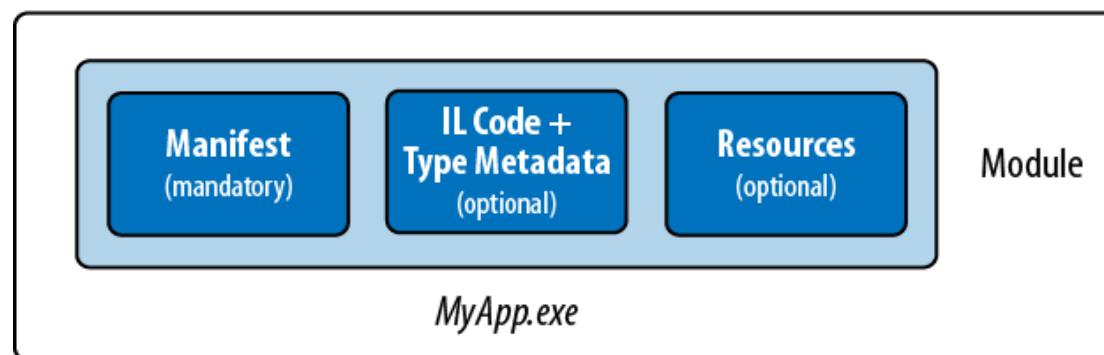
Сборки в .NET



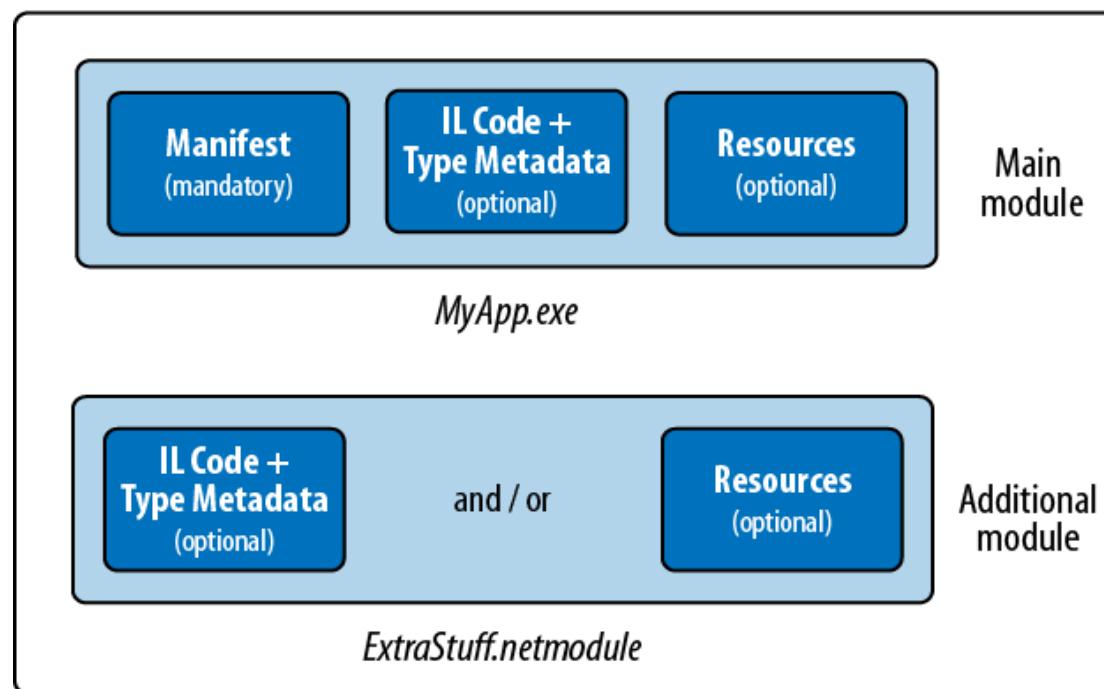


Сборки в .NET

Assembly



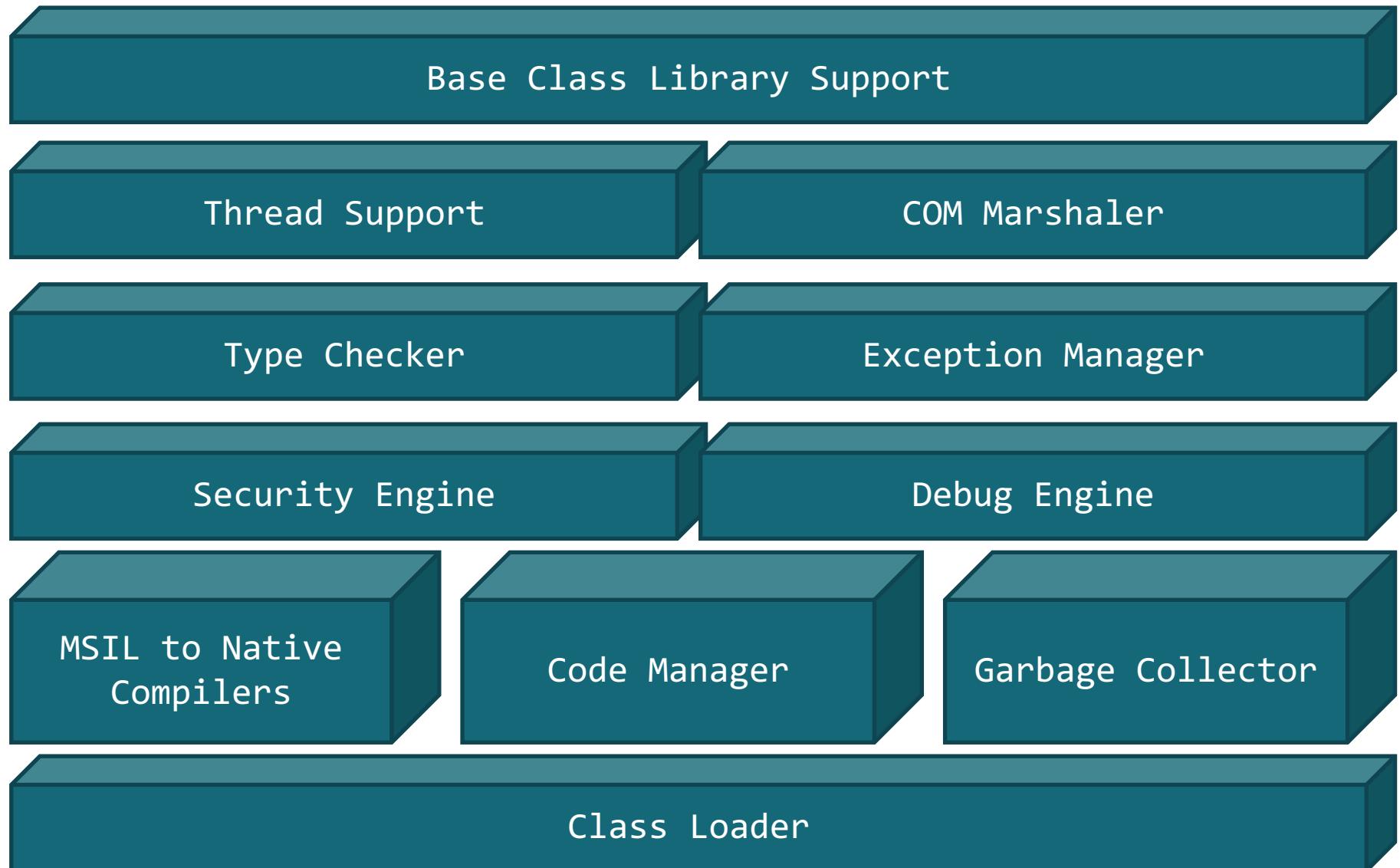
Assembly



Изучение сборок с помощью утилит ildasm.exe и Reflector

- Утилита ildasm.exe, поставляемая в составе пакета .NET Framework 4 SDK, позволяет загружать любую сборку .NET и изучать ее содержимое, в том числе ассоциируемый с ней манифест, CIL-код и метаданные типов
- NET Reflector – платная утилита для Microsoft .NET, комбинирующая браузер классов, статический анализатор и декомпилятор <http://www.red-gate.com/products/dotnet-development/reflector/>
- Free .NET Decompiler and Assembly Browser <https://www.jetbrains.com/decompiler/>
ILSpy is the open-source .NET assembly browser and decompiler <http://ilspy.net/>

Основные компоненты CLR



Основные компоненты CLR

Class Loader (Загрузчик классов)

- находит, 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 соответствующие метаданные для инициализации статических переменных и создания экземпляра загруженного класса

Основные компоненты CLR

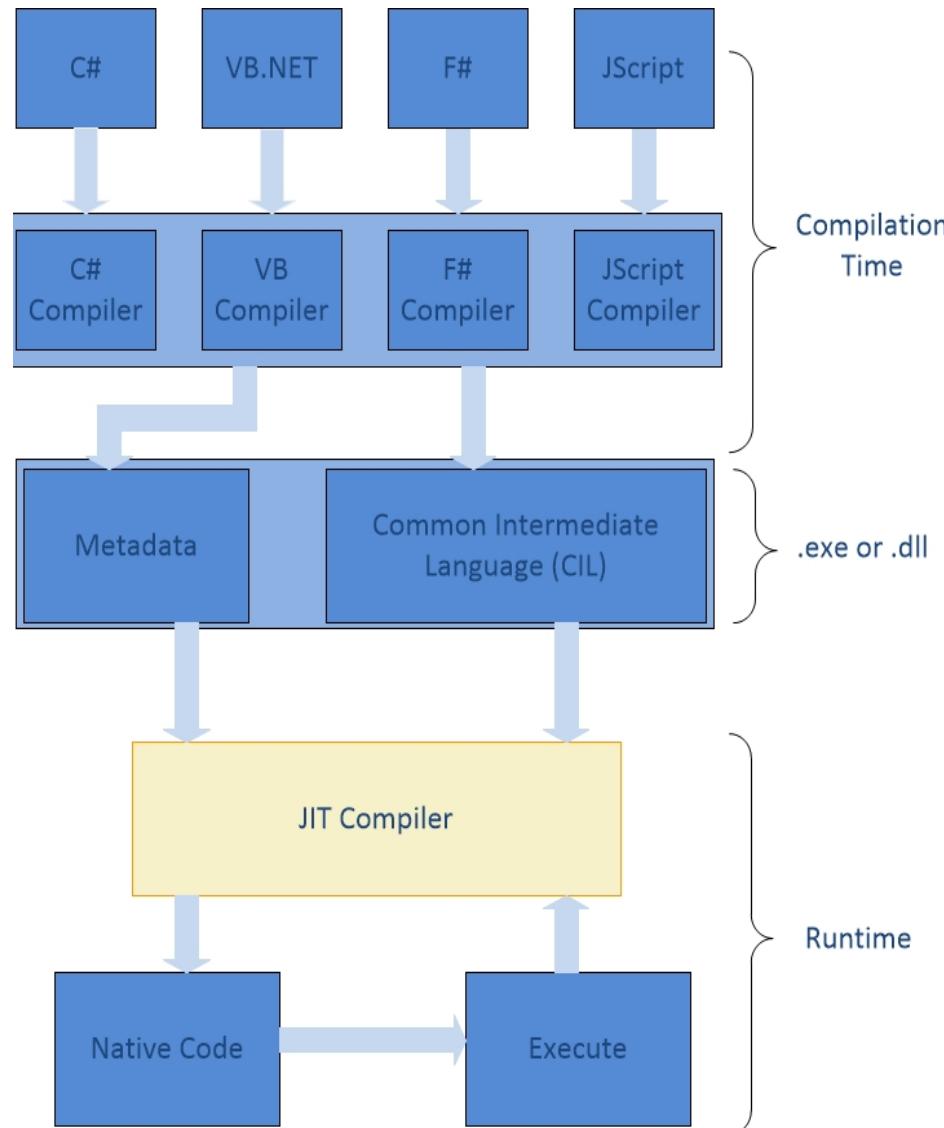
Type Checker (Верификатор)

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

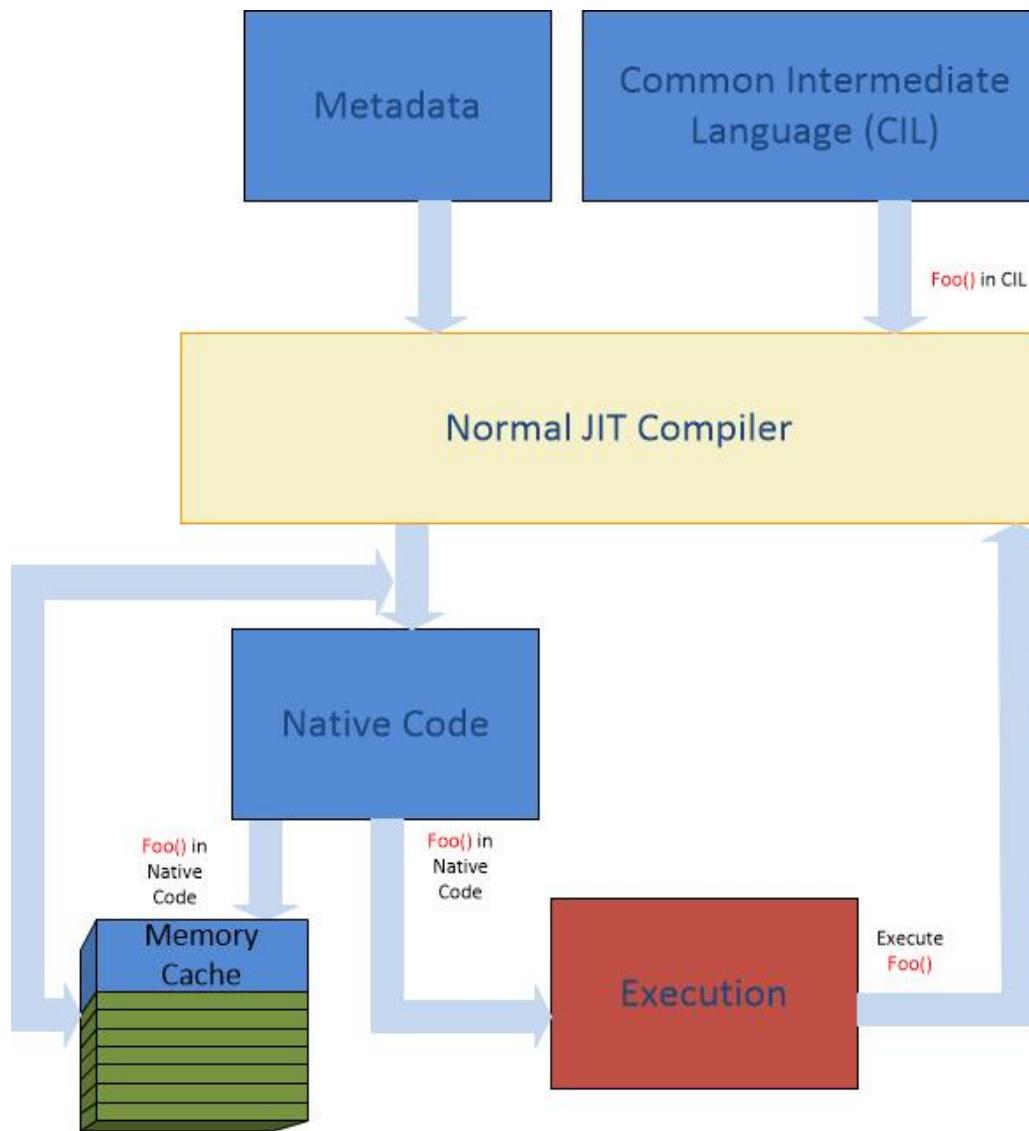
MSIL to Native Compilers (JIT -компиляторы)

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

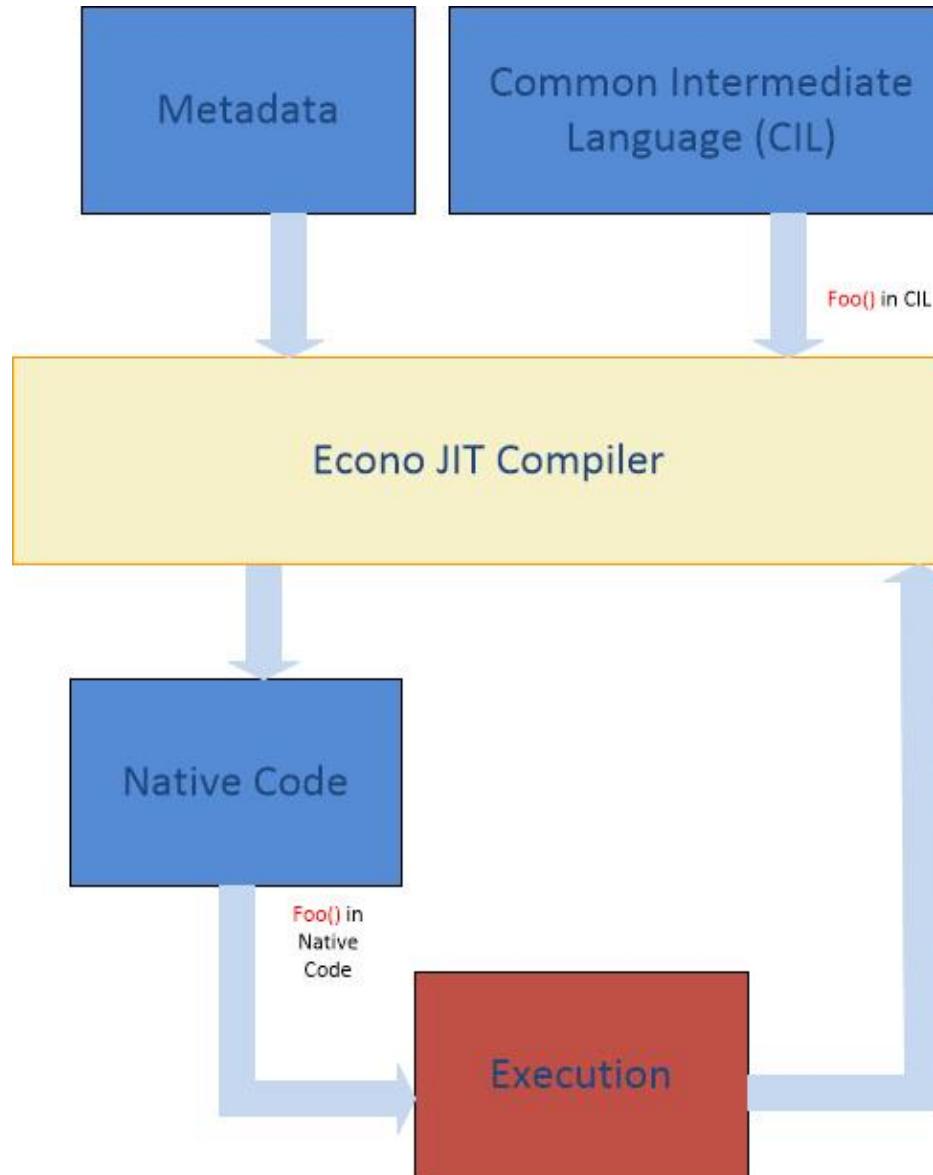
Виды JIT-компиляции



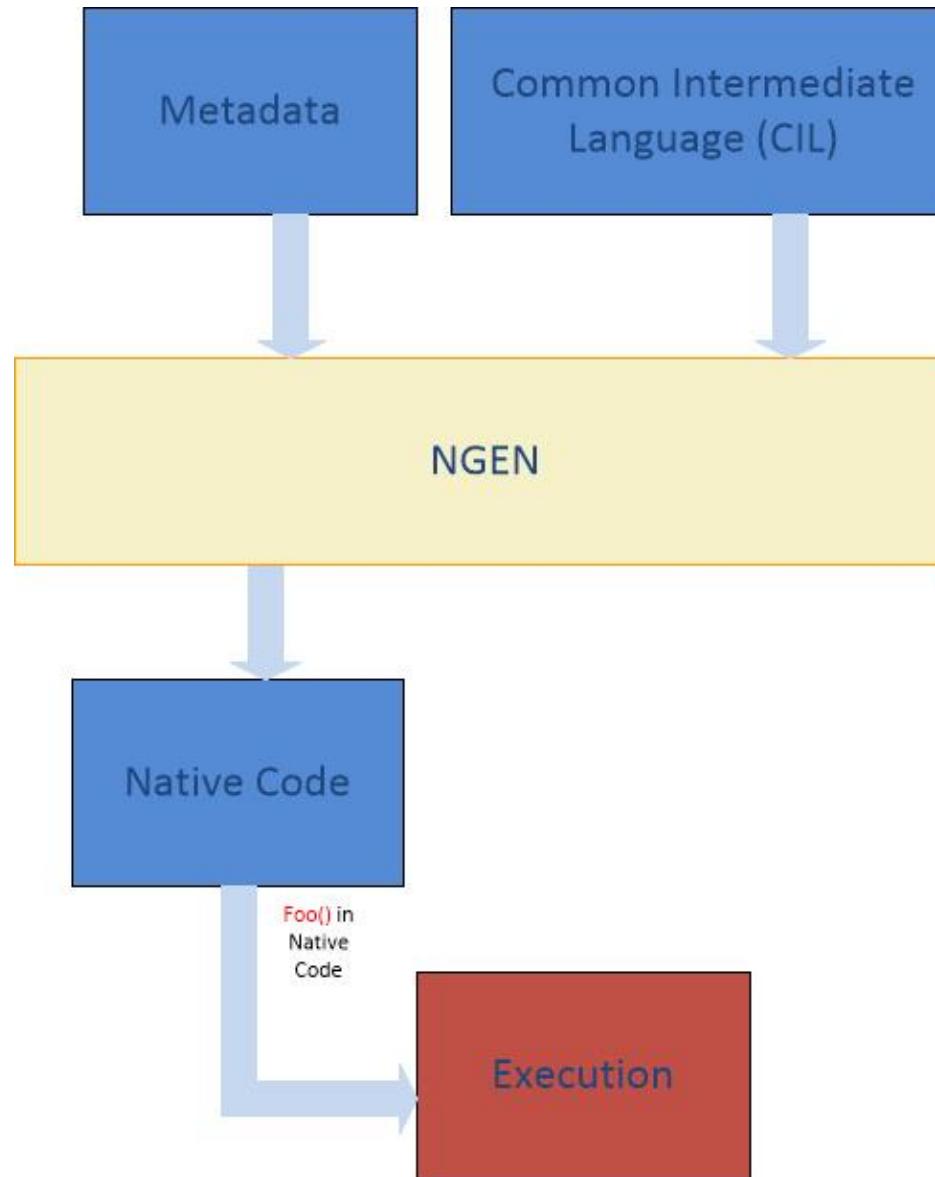
Виды компиляции. Normal JIT-компиляция



Виды компиляции. Econo-JIT-компиляция



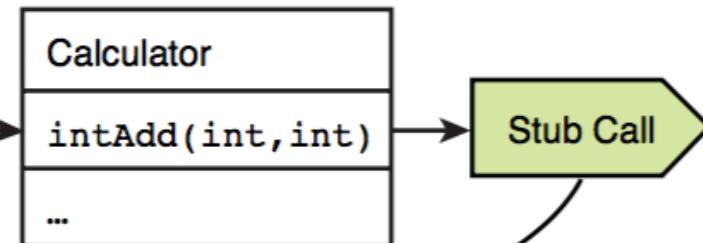
Виды компиляции. Pre-JIT-компиляция



Как CLR загружает, компилирует и запускает сборки

First Call to the Calculator's Add Method

```
static void Main()
{
    var calc = new Calculator();
    ➔ int four = calc.Add(1, 3);
    int five = calc.Add(2, 3);
    Console.WriteLine(sum);
}
```



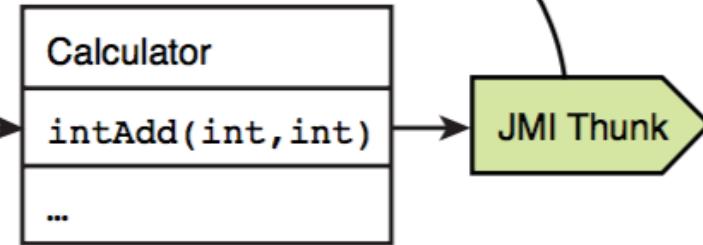
```
ldarg.1  
ldarg.2  
add  
ret
```

➔ JIT Compiler

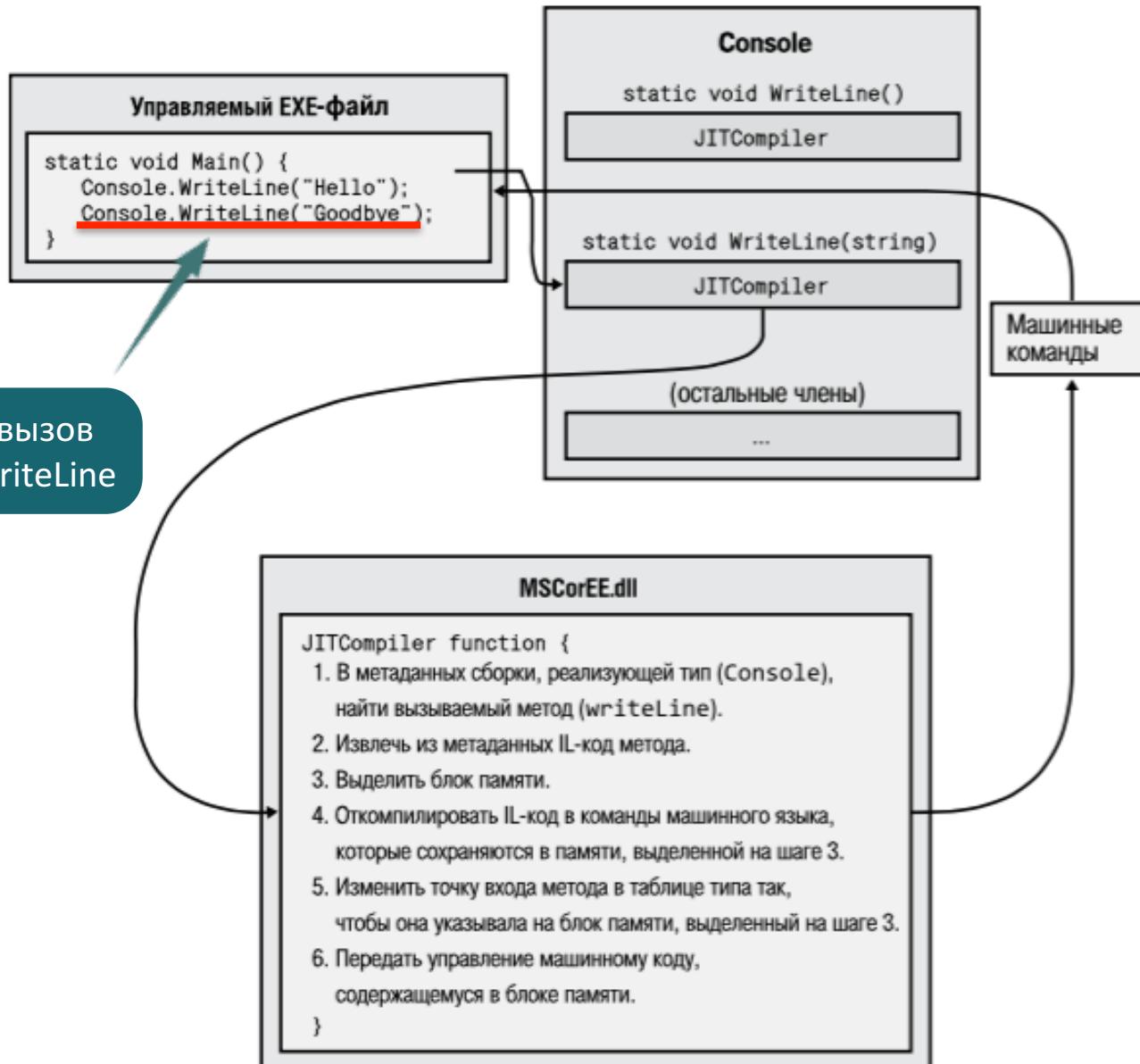
```
// x86 assembler fragment
mov eax, dword ptr [ebp-40h]
add eax, dword ptr [ebp+8]
mov dword ptr [ebp-44h], eax
jmp 00000044
```

Subsequent Calls to the Calculator's Add Method

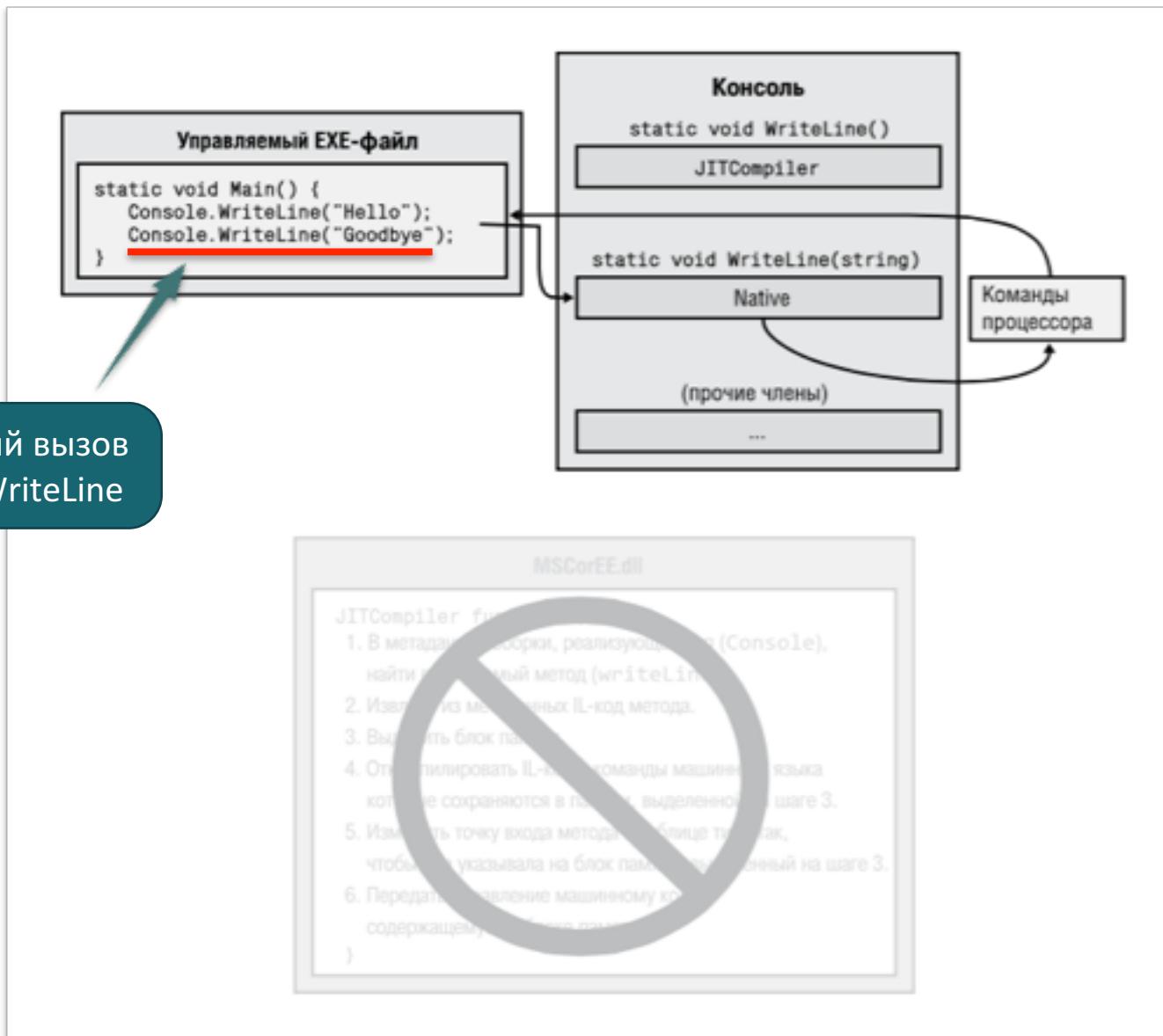
```
static void Main()
{
    var calc = new Calculator();
    int four = calc.Add(1, 3);
    ➔ int five = calc.Add(2, 3);
    Console.WriteLine(sum);
}
```



Как CLR загружает, компилирует и запускает сборки



Как CLR загружает, компилирует и запускает сборки



Сервисы CLR для поддержки и управления исполнением

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

Garbage Collector (сборка мусора) поддерживает 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

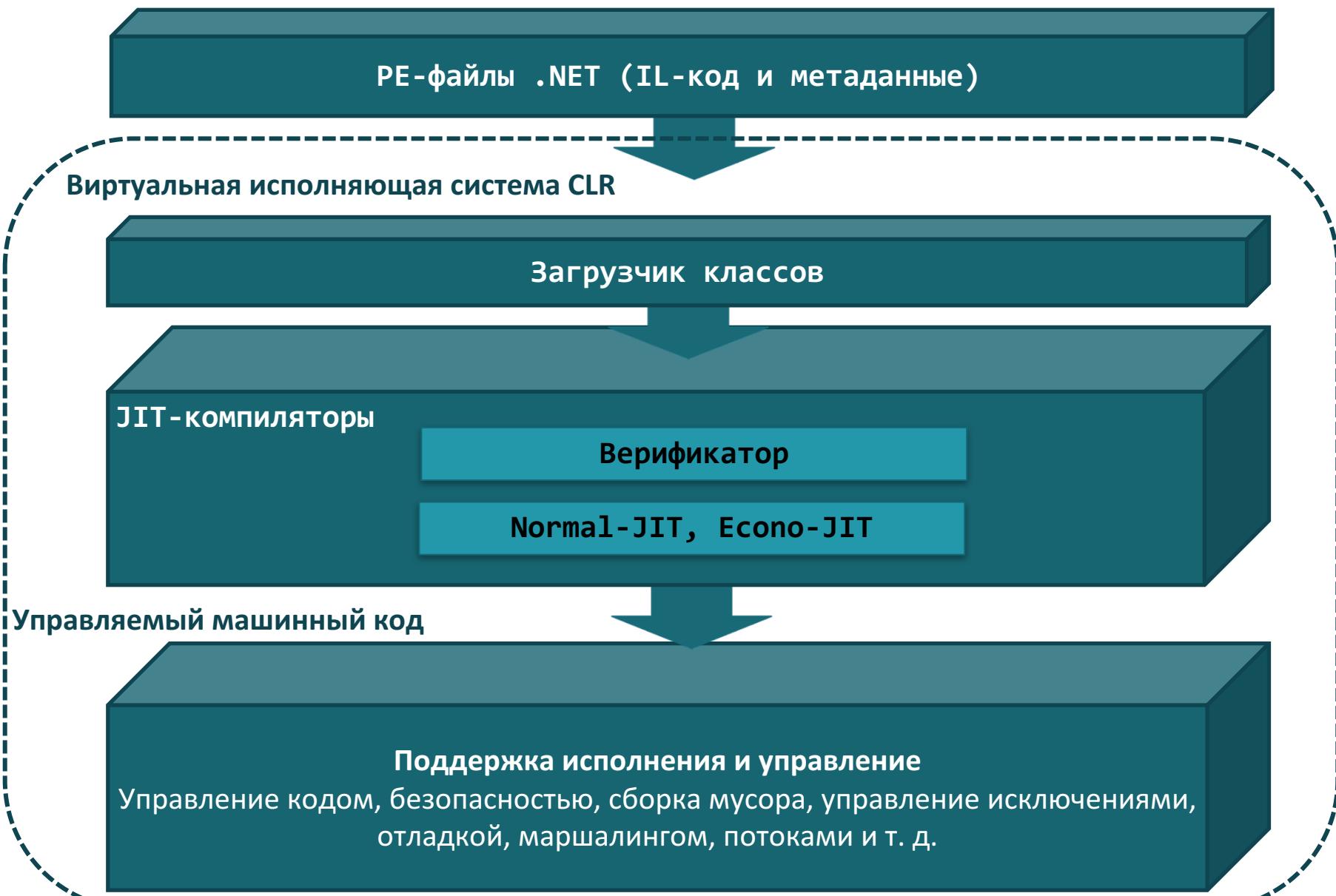
Сервисы CLR для поддержки и управления исполнением

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

Основные компоненты CLR



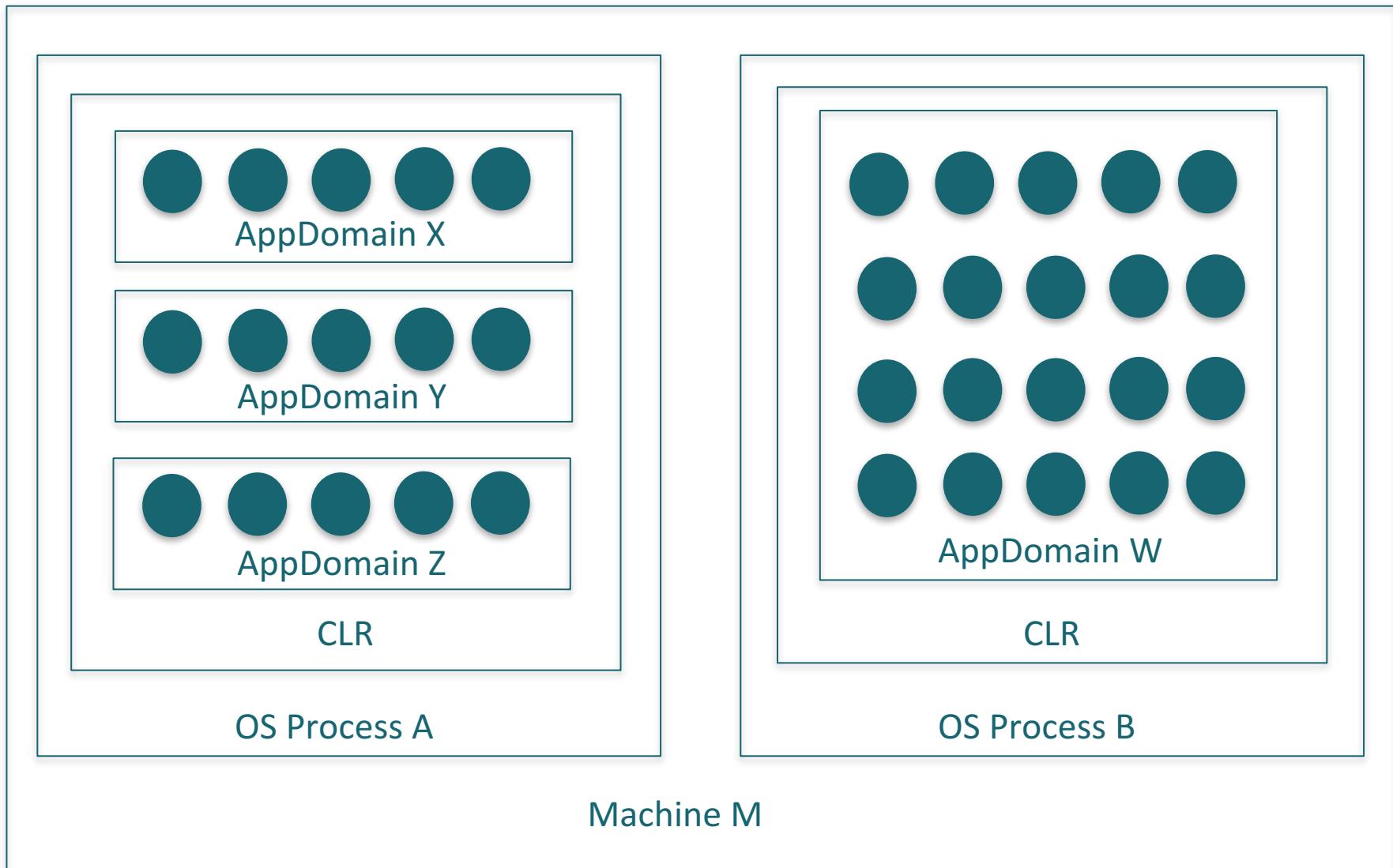
Инфраструктура времени выполнения. Домены приложений

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 сервера IIS - на виртуальном каталоге. Для среды CLR основными областями разграничениями служат домены приложения.

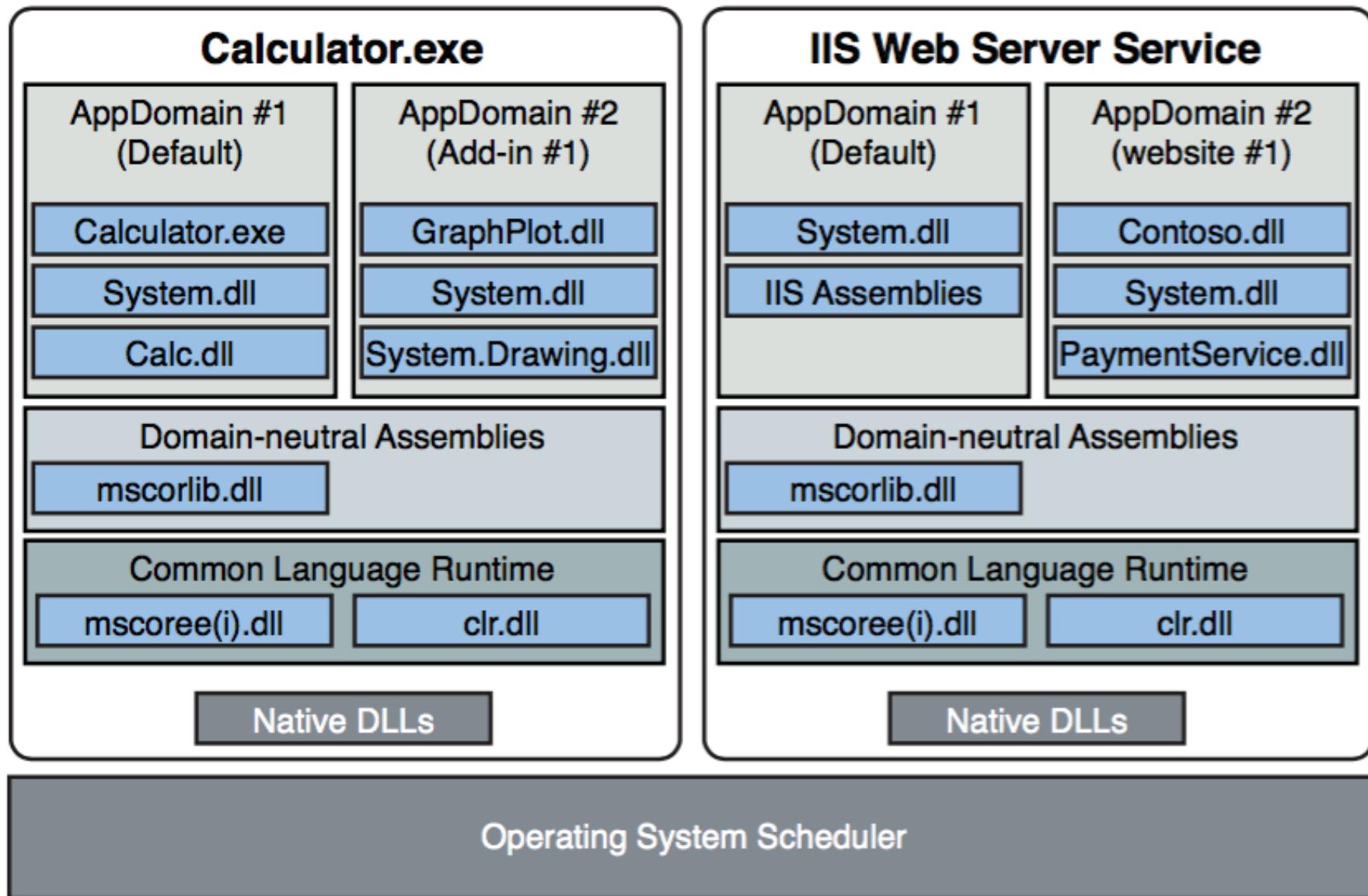
Домены приложений (AppDomains) играют ту же роль, что и процессы в операционной системе. Как и процесс, домен приложения, выделяет (ограничивает) некоторую область выполнения кода, предоставляет уровни изоляции ошибок, обеспечивает изоляцию области и уровень безопасности, владеет ресурсами «от лица»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

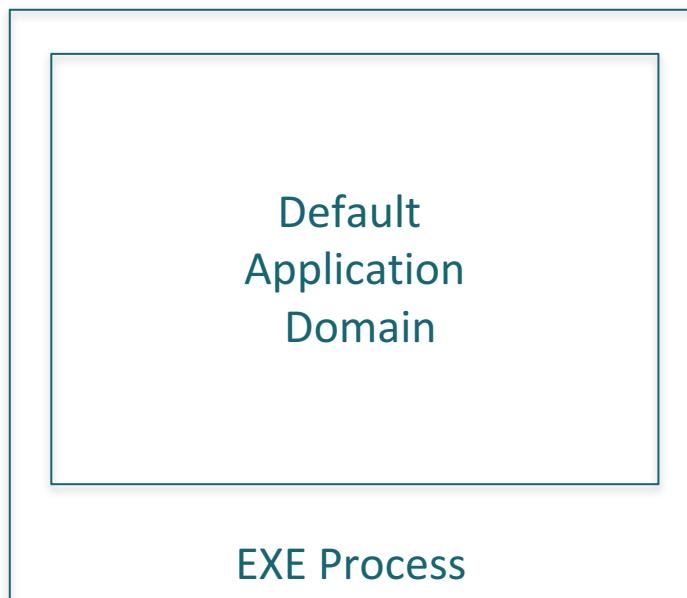
Объекты, домены и процессы



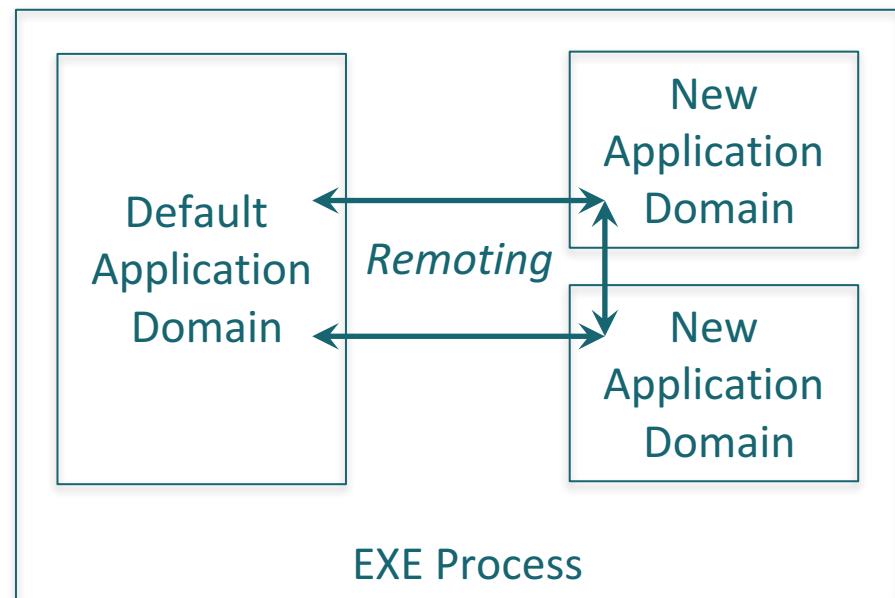
Инфраструктура времени выполнения. Сборки и домены



Инфраструктура времени выполнения. Сборки и домены

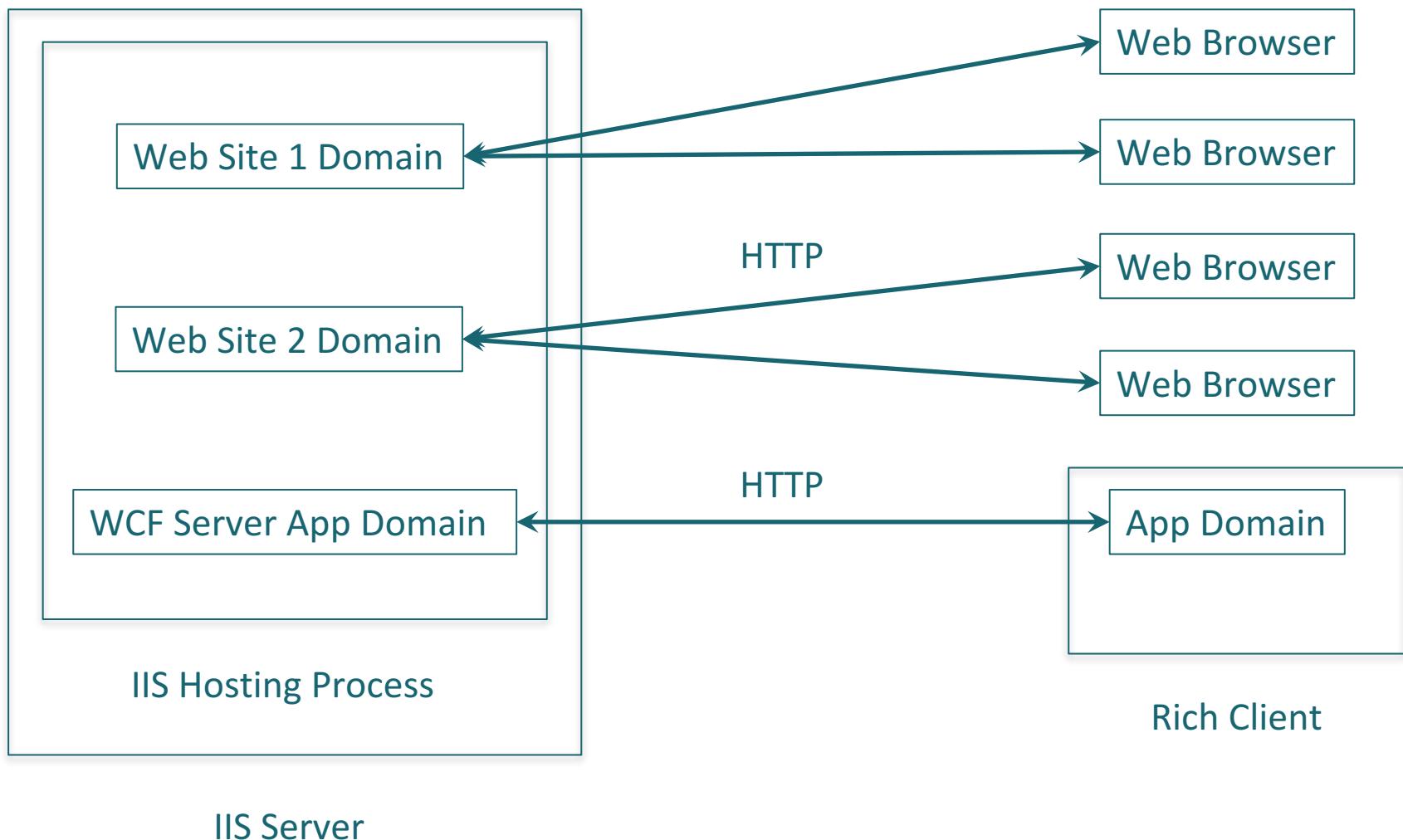


Single-Application-Domain-Program



Multy-Application-Domain-Program

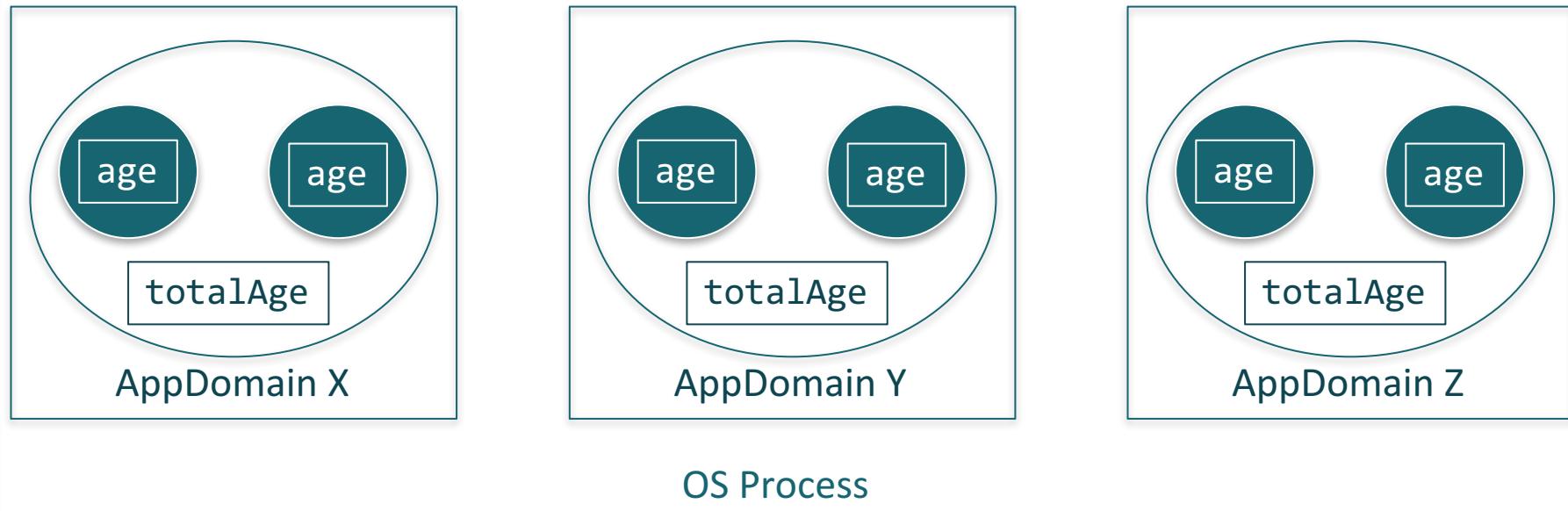
Инфраструктура времени выполнения. Сборки и домены



Инфраструктура времени выполнения. Домены приложений

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;
}
```



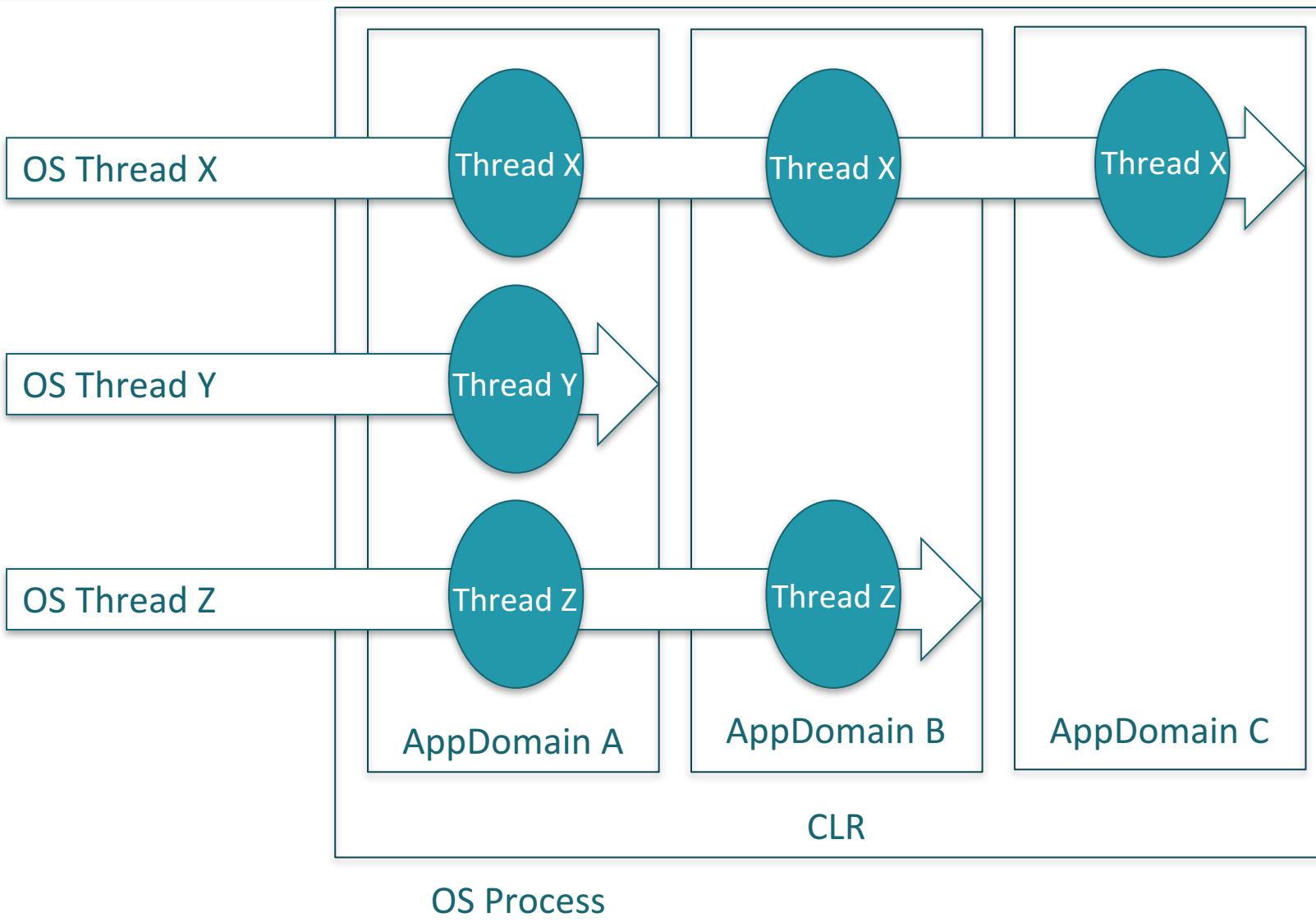
Инфраструктура времени выполнения. Потоки

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.

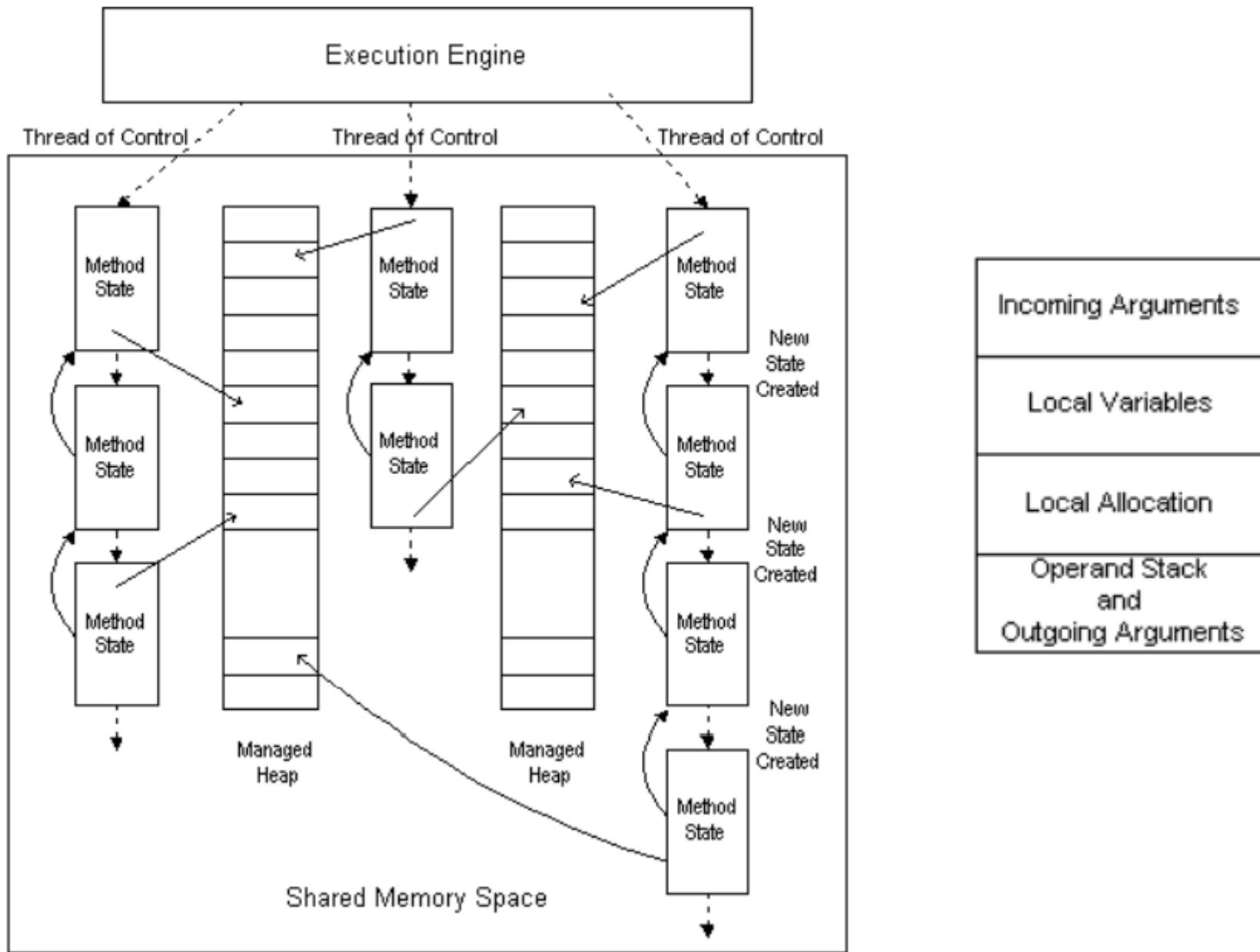
- Объект мягкого потока CLR находится в одном домене приложения
- Один домен приложения может содержать много объектов мягких потоков – в текущей реализации это происходит, когда два или более жестких потока выполняют код в одном домене приложения

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.

Инфраструктура времени выполнения. Потоки



Mashine Model State (ECMA 335)



Инструменты, предоставляемые .NET Framework

Caspol.exe

Makecert.exe

Ildasm.exe

Gacutil.exe

Ngen.exe

Sn.exe

[https://msdn.microsoft.com/ru-ru/library/d9kh6s92\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/d9kh6s92(v=vs.110).aspx)

Инструменты, предоставляемые .NET Framework

Инструмент	Описание
Global Assembly Cache Tool (Gacutil.exe)	Позволяет пользователям просматривать содержимое глобального кэша сборок и кэша загрузки, а также управлять ими. С помощью этого инструмента можно добавлять и удалять сборки в GAC, для того, чтобы приложения могли получать к ним доступ.
Native Image Generator (Ngen.exe)	Генератор образов в машинном коде (Native Image Generator) — это средство повышения быстродействия управляемых приложений. Ngen.exe создает образы в машинном коде, представляющие собой файлы, содержащие компилированный специфический для процессора машинный код, и устанавливает их в кэш образов в машинном коде на локальном компьютере. Среда выполнения может использовать образы в машинном коде, находящиеся в кэше, вместо использования JIT-компилятора для компиляции исходной сборки.

Инструменты, предоставляемые .NET Framework

Инструмент	Описание
MSIL Disassembler (Ilasm.exe)	MSIL Disassembler является парным инструментом к ассемблеру MSIL (Ilasm.exe). Ilasm.exe принимает входной исполняемый файл (PE-файл). Содержащий код на языке MSIL, и создает на его основе текстовый файл, который может служить входным для программы Ilasm.exe. Можно использовать Ilasm.exe для просмотра промежуточного языка MSIL в файле. Если анализируемый файл является сборкой, то эти данные могут включать в себя атрибуты сборки, а также ссылки на другие модули и сборки. Эти данные полезны для определения того, является ли файл сборкой или частью сборки и имеет ли он ссылки на другие модули и сборки.
Strong Name Tool (Sn.exe)	Позволяет пользователям подписывать сборки строгими именами. Strong Name Tool включает в себя команды для создания новой пары ключей, извлечения открытого ключа из пары ключей и верификации сборки.

Common Type System, Common Language Specification

- Microsoft VB .NET
- Microsoft VC++ .NET
- Microsoft C#
- Microsoft J#
- Microsoft Jscript
- APL
- ASNA Visual RPGRPG.NET
- Fujitsu COBOL
- Micro Focus Cobol NetExpress
- F# (a mixed functional/imperative language based on Caml from Microsoft Research)
- Eiffel
- Delta Forth
- Lahey/Fujitsu Fortran for .NET
- Glasgow Haskell ...

C# представляет собой унифицированный, безопасный к типам, объектно-ориентированный язык, не зависящий от платформы, но написанный для эффективной работы с платформой .NET Framework

Основные особенности языка

1. Объектная ориентация, унифицированная и расширенная (формальные синтаксические конструкции для классов, интерфейсов, структур, перечислений и делегатов) система типов; свойства, события
2. Безопасность в отношении типов, поддержка статической типизации
3. Управление памятью
4. Поддержка платформ

Эволюция C#, CLR и .NET Framework

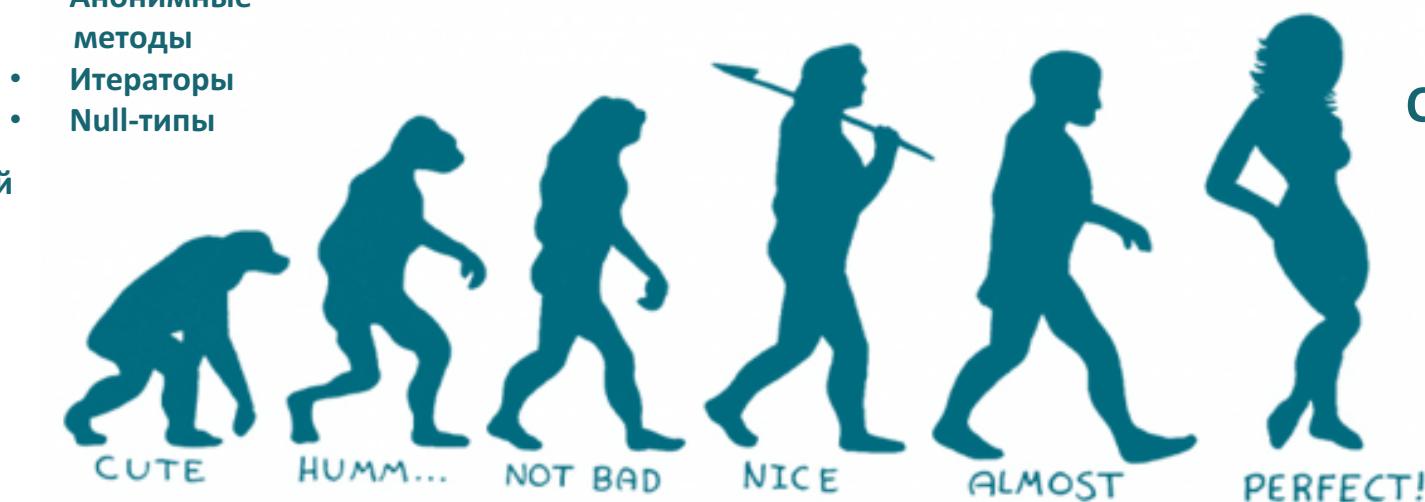
Версия C#	Версия CLR	Версия Framework
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP1)	3.5
4.0	4.0	4.0
5.0	4.5 (Patched 4.0)	4.5 (including 4.5.1 и 4.5.2)
6.0	4.5 (Patched 4.0)	4.6
7.*	4.5 (Patched 4.0)	4.7

<https://github.com/dotnet>

<https://habrahabr.ru/post/280978/>

Эволюция языка C#

C# 1.0
Управляемый
код



- C# 2.0
- Обобщения
 - Смешанные типы
 - Анонимные методы
 - Итераторы
 - Null-типы

C# 3.0

- Неявно типизируемые локальные переменные
- Инициализаторы объектов и коллекций
- Автоматическая реализация свойств
- Анонимные типы
- Методы расширения
- Запросы
- Лямбда-выражения
- Деревья выражений

C# 5.0 Асинхронные функции

C# 6.0

- Getter-only auto-properties
- Auto-property initializers
- Expression-bodied members
- Null-conditional operators
- Using static members
- Index initializers
- String interpolation
- nameof operator
- Await in catch/finally
- Exception filters
- Extension Add in collection initializers
- Improved overload resolution

C# 4.0

- Динамическое связывание
- Именованные и дополнительные аргументы
- Обобщенная ковариантность и контравариантность

C# 7.*!

Документирование приложений. XML комментарии

В Visual Studio можно добавить комментарии к исходному коду, который будет обработан в XML файл

XML файл может быть включен в процесс создания справочной документации по классу или использован для поддержки IntelliSense

```
/// <summary> The Hello class prints a greeting on the screen
/// </summary>
public class Hello
{
    /// <summary> We use console-based I/O. For more information about
    /// WriteLine, see <seealso cref="System.Console.WriteLine()" />
    /// </summary>
    public static void Main()
    {
        Console.WriteLine("Hello World");
    }
}
```

Документирование приложений. XML комментарии

Тег	Назначение
<summary>	Предоставляет краткое описание. Для более подробного описания используются теги <remarks>.
<remarks>	Содержит подробное описание. Этот тег может содержать вложенные разделы (пункты), списки и другие типы тегов.
<example>	Предоставляет пример того, как метод, свойство или другой член библиотеки должен быть использован. Этот тег часто связано с использованием вложенных тегов <code>.
<code>	Указывает, что прилагаемый текст является кодом приложения.
<returns>	Документирует возвращаемое значение и тип метода.
<exception>	Документирует класс исключения (синтаксис проверяется компилятором)
<param>	Помечает параметр метода (синтаксис проверяется компилятором)
<value>	Описывает свойство

<https://msdn.microsoft.com/ru-ru/library/b2s063f7.aspx>

Создание документации из XML комментариев



Sandcastle - Documentation Compiler for Managed Class Libraries - генератор документации (открытая часть некоторого внутреннего инструмента) компании Microsoft, который позволяет автоматически получить техническую документацию в стиле MSDN по заданной .NET-сборке с управляемым кодом.

Sandcastle состоит из набора утилит, которые получают метаинформацию из сборки, и затем рядом операций преобразуют ее в конечный вид. В процессе преобразования информация представлена в формате XML, часть преобразований выполняется с помощью шаблонов XSLT.

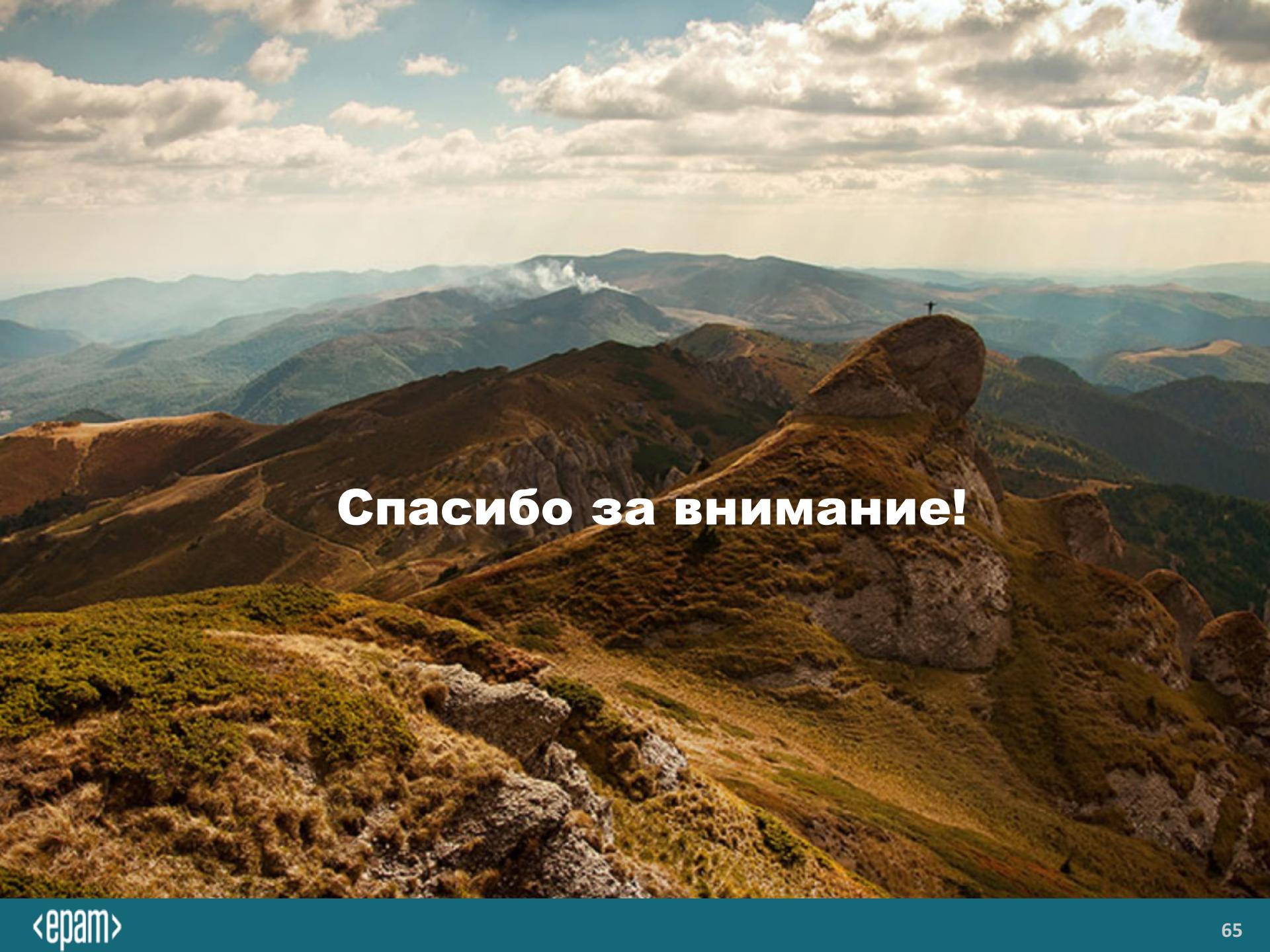
Sandcastle используется внутри Microsoft для получения документации на Visual Studio и .NET Framework.

<http://sandcastle.codeplex.com/>

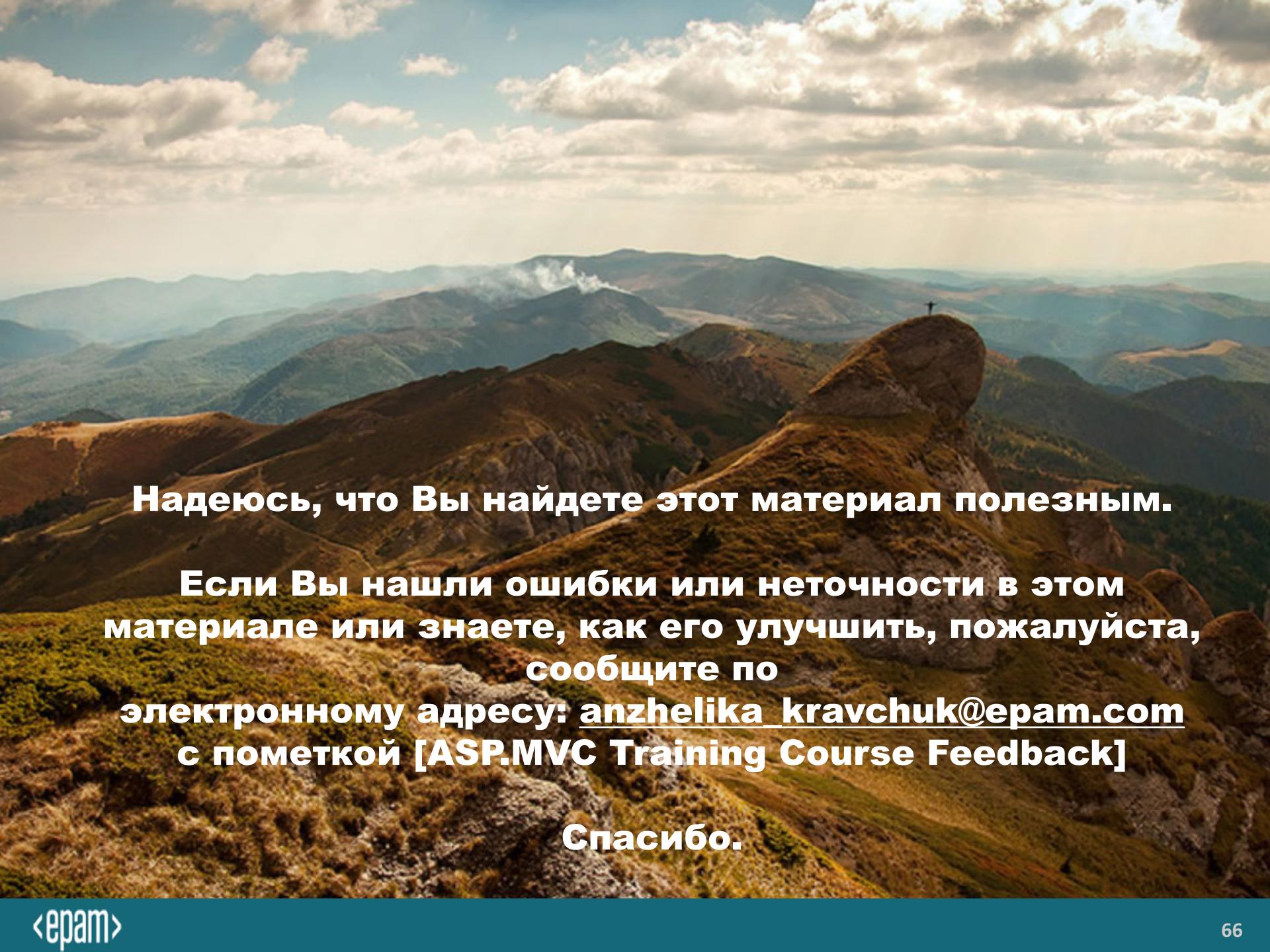
Создание документации из XML комментариев

На основании существующего XML файла, содержащего комментарии, которые были извлечены из проекта, можно создать .chm файл с помощью такого инструмента как Sandcastle Help File Builder. Для этого нужно выполнить следующие действия

- Нажать кнопку Start, пункт All Programs, выбрать Sandcastle Help File Builder, а затем нажать Sandcastle Help File Builder GUI.
- В Sandcastle Help File Builder, в меню File выбрать команду New Project.
- В диалоговом окне Save New Help Project As выполнить следующие действия, а затем нажать Save:
 - Выбрать путь, по которому следует сохранить проект.
 - Указать имя для Sandcastle проекта.
- В окне Project Explorer щелкнуть правой кнопкой мыши Documentation Sources, а затем нажать кнопку Add Documentation Source.
- В диалоговом окне Select the documentation source(s) перейти к папке, содержащий XML файл, а затем нажать кнопку Open.
- В меню Documentation, выбрать Build Project. Подождать, пока проект успешно построится. Это займет некоторое время.

A wide-angle photograph of a mountain range under a dramatic sky. In the foreground, rocky terrain and green slopes are visible. In the middle ground, a person stands on a prominent, rounded rock formation on a ridge. The background features multiple layers of mountains, with one peak showing a small plume of smoke or steam. The sky is filled with large, white, billowing clouds.

Спасибо за внимание!



Надеюсь, что Вы найдете этот материал полезным.

**Если Вы нашли ошибки или неточности в этом
материале или знаете, как его улучшить, пожалуйста,
сообщите по**

**электронному адресу: anzhelika_kravchuk@epam.com
с пометкой [ASP.MVC Training Course Feedback]**

Спасибо.