

АРХИТЕКТУРА НА ПЛАТФОРМАТА MICROSOFT .NET

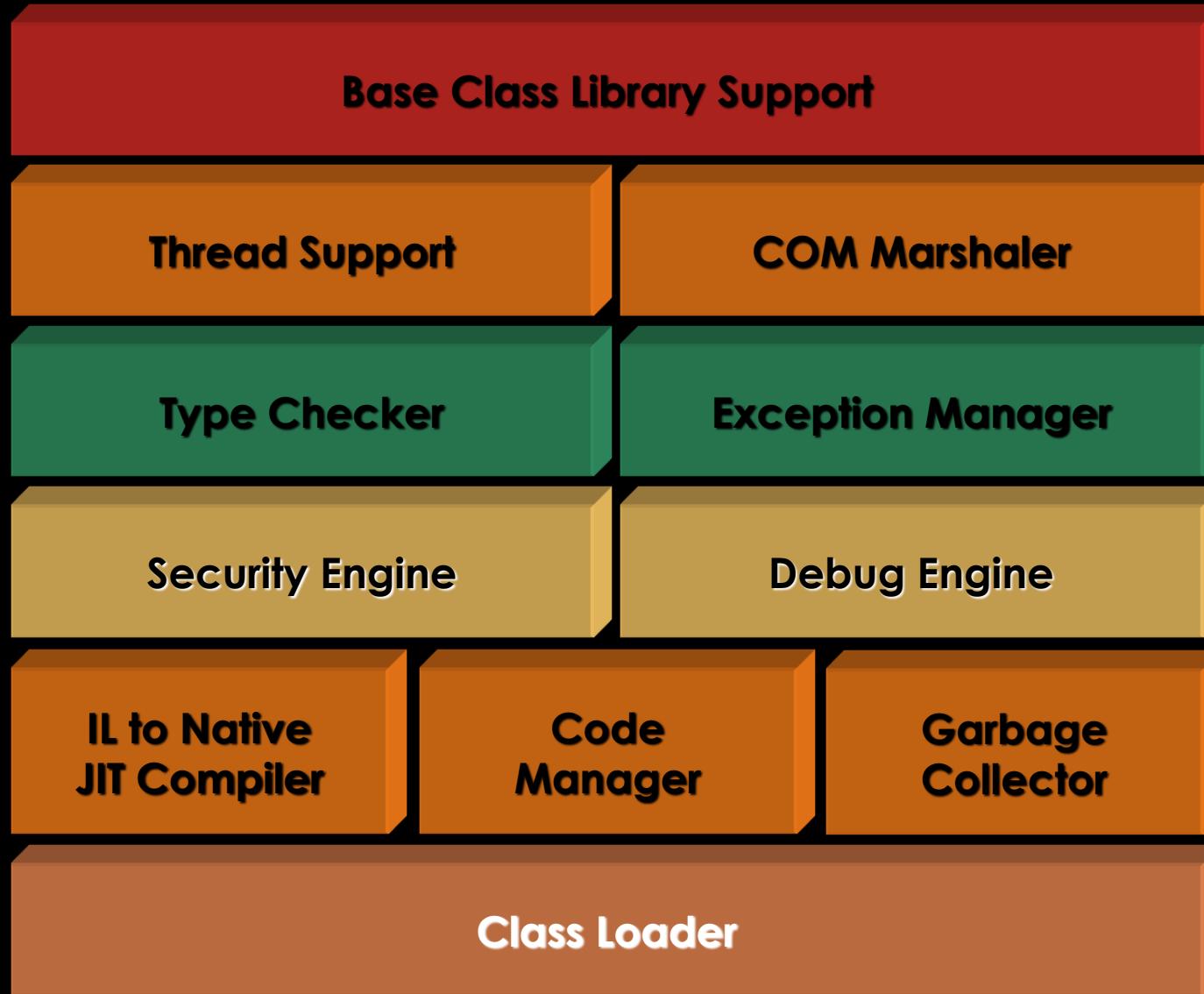
COMMON LANGUAGE RUNTIME

- **Common Language Runtime (CLR)** е най-важният компонент на .NET Framework. Той представлява **среда за контролирано изпълнение на управляем код**.
- На практика CLR е тази част от .NET Framework, която изпълнява компилираните .NET програми в специална изолирана среда.
- В своята същност CLR представлява виртуална машина, която изпълнява инструкции на езика **Intermediate Language (IL)**. Това е езикът, до който се компилират всички .NET езици.

COMMON LANGUAGE RUNTIME

- **CLR е сходен на виртуален компютър**, който обаче не изпълнява асемблерен код за процесор Pentium, AMD или някакъв друг, **а междинен (IL) код**.
- Има голямо сходство между .NET CLR и Java Virtual Machine, но между двете технологии има и много разлики.
- По предназначение те служат за едно също нещо – да изпълняват код за определен **виртуален процесор**.
- В .NET това е IL кода, а при Java платформата – т. нар. Java bytecode.
- Основната разлика между IL и bytecode е, че **IL е език от по-високо ниво**, а това позволява да бъде компилиран много по-ефективно от Java bytecode.

Архитектура на CLR



ЗАДАЧИ И ОТГОВОРНОСТИ НА CLR



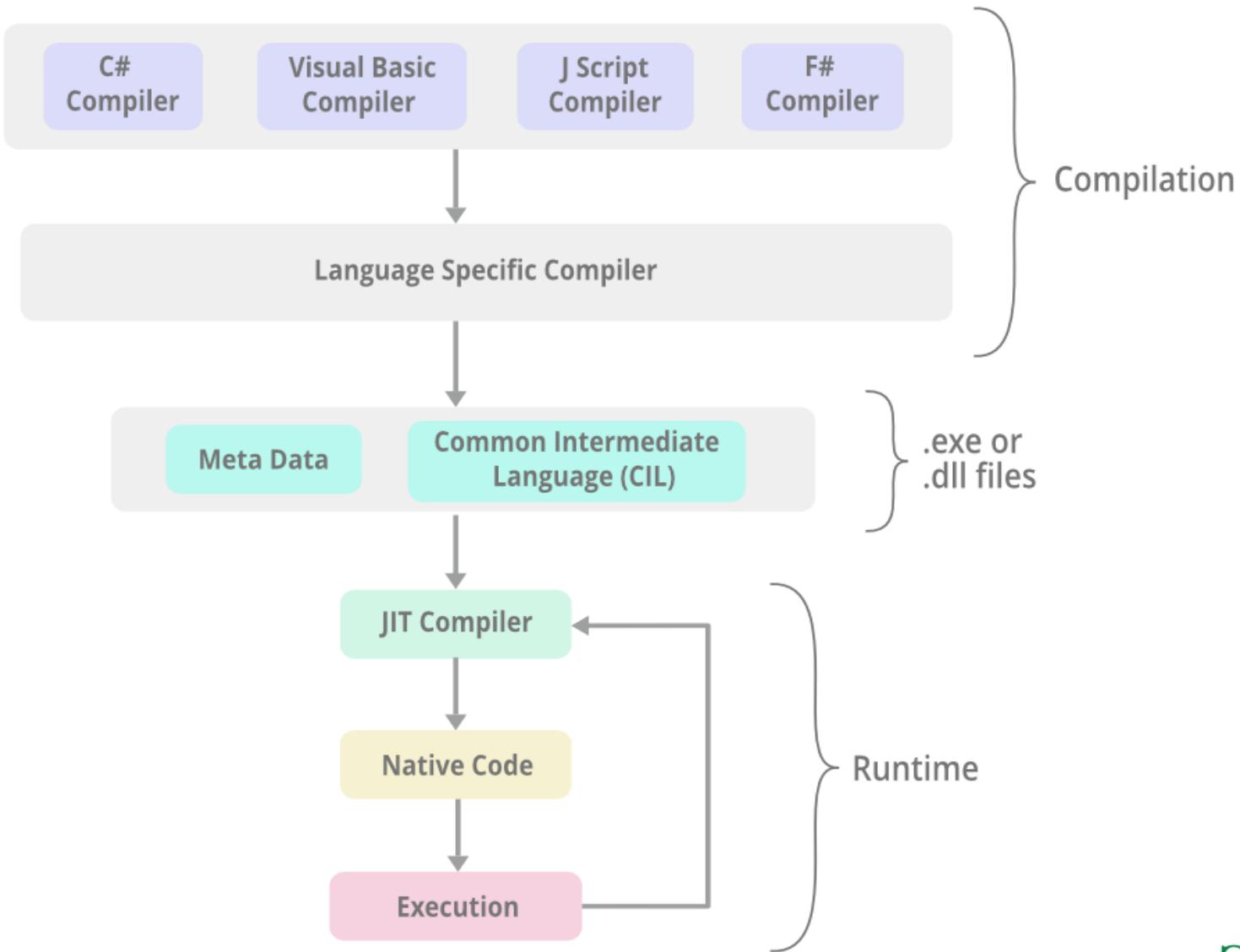
ИЗПЪЛНЕНИЕ НА IL КОДА

- Реално **IL инструкциите**, преди да бъдат изпълнени за първи път, се компилират до инструкции за текущия процесор и след това се изпълняват от системния процесор.
- Този процес на междинно компилиране до **машиннозависим** (native) код се нарича **JIT компилиация** (Just-In-Time compilation).

ИЗПЪЛНЕНИЕ НА IL КОДА

- **Native code** се отнася до инструкциите на машинно ниво, които **се изпълняват** директно от хардуера на компютъра, по-конкретно от неговия процесор, **без да се нуждаят от интерпретация или превод по време на изпълнение.**

Working of JIT Compiler



УПРАВЛЕНИЕ НА ПАМЕТТА И РЕСУРСИТЕ

- CLR включва в себе си система за **заделяне на памет** и система за почистване на неизползваната памет и ресурси (т. нар. **garbage collector**).
- Управлението на паметта при .NET приложенията се извършва в голяма степен автоматизирано и в повечето случаи програмистът не трябва да се грижи за освобождаване на заделената памет.



ОСИГУРЯВАНЕ БЕЗОПАСНОСТТА НА ТИПОВЕТЕ

- .NET Framework е среда за **контролирано изпълнение на програмен код** (managed execution environment).
- Това не позволява:
 - директен достъп до паметта;
 - директна работа с указатели;
 - преобразуване от един тип към друг, който не е съвместим с него;
 - излизане от границите на масив, както и всякакви други опасни операции.
- По тази причина .NET се нарича **управлявана среда**: тя управлява изпълнението на кода и по този начин предпазва програмите от много досадни проблеми, които възникват при неуправляваните среди.

УПРАВЛЕНИЕ НА СИГУРНОСТТА

.NET Framework има добре изградена концепция за сигурност на различни нива.

- От една страна .NET приложенията могат да се изпълняват с различни права. Правата могат да се задават от администраторите чрез т. нар. **политики за сигурност**.

 CLR следи дали кодът, който се изпълнява, спазва зададената политика за сигурност и не позволява тя да бъде нарушена. Тази техника се нарича "**code access security**".

- От друга страна .NET Framework поддържа и средства за управление на сигурността, базирана на роли (**role-based security**).

УПРАВЛЕНИЕ НА ИЗКЛЮЧЕНИЯТА

- .NET Framework е изцяло **обектно-ориентирана среда** за разработка и изпълнение на програмен код. В нея механизъмът на изключенията е залегнал като **основно средство за управление на грешки и непредвидени ситуации.**
- Една от задачите на CLR е да се грижи за **изключенията, които възникват по време на изпълнение на кода.** При настъпване на изключение CLR има грижата да намери съответния обработчик и да му предостави управлението.

УПРАВЛЕНИЕ НА КОНКУРЕНТОСТТА

- CLR контролира паралелното изпълнението на нишки (threads), като за целта си взаимодейства с операционната система.

ВЗАИМОДЕЙСТВИЕ С НЕУПРАВЛЯВАН КОД

- CLR осигурява връзка между управляем (NET) код и неуправляем (Win32) код.
- За целта той изпълнява доста сложни задачи, свързани с **конвертиране на данни, синхронизация, прехвърляне на извиквания, взаимодействие с компонентния модел на Windows (COM)** и много други.
- Подпомага процесите на дебъгване (debugging) и оптимизира (profiling) управляемия код.
- CLR осигурява инфраструктура и средства за реализацията на дебъгване и оптимизиране на кода от външни специализирани програми.

УПРАВЛЯВАН КОД

- Управляваният код (**managed code**) е кодът, който се изпълнява от CLR. Той представлява поредица от IL инструкции, които се получават при компилиацията на .NET езиците.
- По време на изпълнение управляваният код се компилира допълнително до **машиннозависим код** за текущата платформа и след това се изпълнява директно от процесора.

УПРАВЛЯВАН КОД И НЕУПРАВЛЯВАН КОД

- Управляваният код (.NET код) се различава значително от неуправлявания код (например Win32 кода).
- Управляваният код е **машиннонезависим**, т. е. може да работи на различни хардуерни архитектури, процесори и операционни системи, стига за тях да има имплементация на CLR.
- Неуправляваният код е **машинозависим**, компилиран за определена хардуерна архитектура и определен процесор. Например програмите, написани на езика C, се компилират до неуправляван код за определена архитектура.

УПРАВЛЯВАН КОД И НЕУПРАВЛЯВАН КОД

- Ако компилираме една С програма за Windows върху архитектура x86, ще получим неуправляван код за процесор x86 (примерно Pentium, Athlon и т.н.), който използва системни извиквания към Windows.
- Този код се **нарича Win32 код** и може да работи само върху 32-битова Windows операционна система. За да се стартира върху друга платформа, трябва да се пре-компилира.

УПРАВЛЯВАН КОД И НЕУПРАВЛЯВАН КОД

При управляемия код нещата стоят по различен начин.

- Ако компилираме една C# програма за платформа .NET Framework, ще получим управляем, машиннонезависим IL код, който може да работи върху различен хардуер.
- Кодът реално ще е компилиран за платформа CLR и ще се състои от IL инструкции за виртуалния процесор на CLR и ще използва системни извиквания към **.NET Base Class Library**.
- Управляваният код лесно може да бъде пренесен върху различни платформи без да се променя или прекомпилира.
- Така например програма на C#, която е компилирана под Windows до управляем IL код, може да се изпълнява без промени както върху Windows под .NET Framework, така и върху Linux под Mono, а също и върху мобилни устройства.

МЕТАДАННИТЕ В УПРАВЛЯВАНИЯ КОД

- Управляваният код се самоописва чрез **метаданни** и носи в себе си описание на типове данни, класове, интерфейси, свойства, полета, методи, параметри на методите и други, както и описание на библиотеки с типове, описание на изисквания към сигурността при изпълнение и т. н.
- **Това дава голяма гъвкавост на разработчика и възможност за динамично зареждане, изследване и изпълнение на функционалност, компилирана като управляем (IL) код.**
- Неуправляваният код стандартно не съдържа метаданни и това силно затруднява динамичното зареждане и изпълнение на неуправлявана функционалност.

УПРАВЛЯВАНИЯТ КОД Е ОБЕКТНО-ОРИЕНТИРАН

- Управляваният код задължително е обектно-ориентиран, докато за неуправлявания няма такова изискване.

Всички .NET езици са обектно-ориентирани. Всички .NET програми се компилират до класове и други типове от общата система от типове на .NET Framework.

- Всички данни, използвани от управлявания код, са наследници (в смисъла на обектно-ориентираното програмиране) на базовия тип `System.Object`.

ИНТЕГРИРА РАЗЛИЧНИ ЕЗИЦИ

- **До управляем код се компилират всички .NET езици.**
- Това дава възможност за широко взаимодействие между код, писан на различни езици за програмиране. Възможно е дори клас, написан на един .NET език, да бъде наследен и разширен от клас, написан на друг .NET език.
- **За .NET Framework няма значение на какъв език е бил написан кода преди да бъде компилиран. Всичкият код се компилира до IL и се изпълнява от CLR по еднакъв начин.**

ОБОБЩЕНИЕ

Управляван код

- Обектно-ориентиран
- Защитен от неправилна работа с типове (type-safety)
- Сигурен и надежден
- Позволява интеграция между различни езици за програмиране
- Позволява преносимост между различни платформи

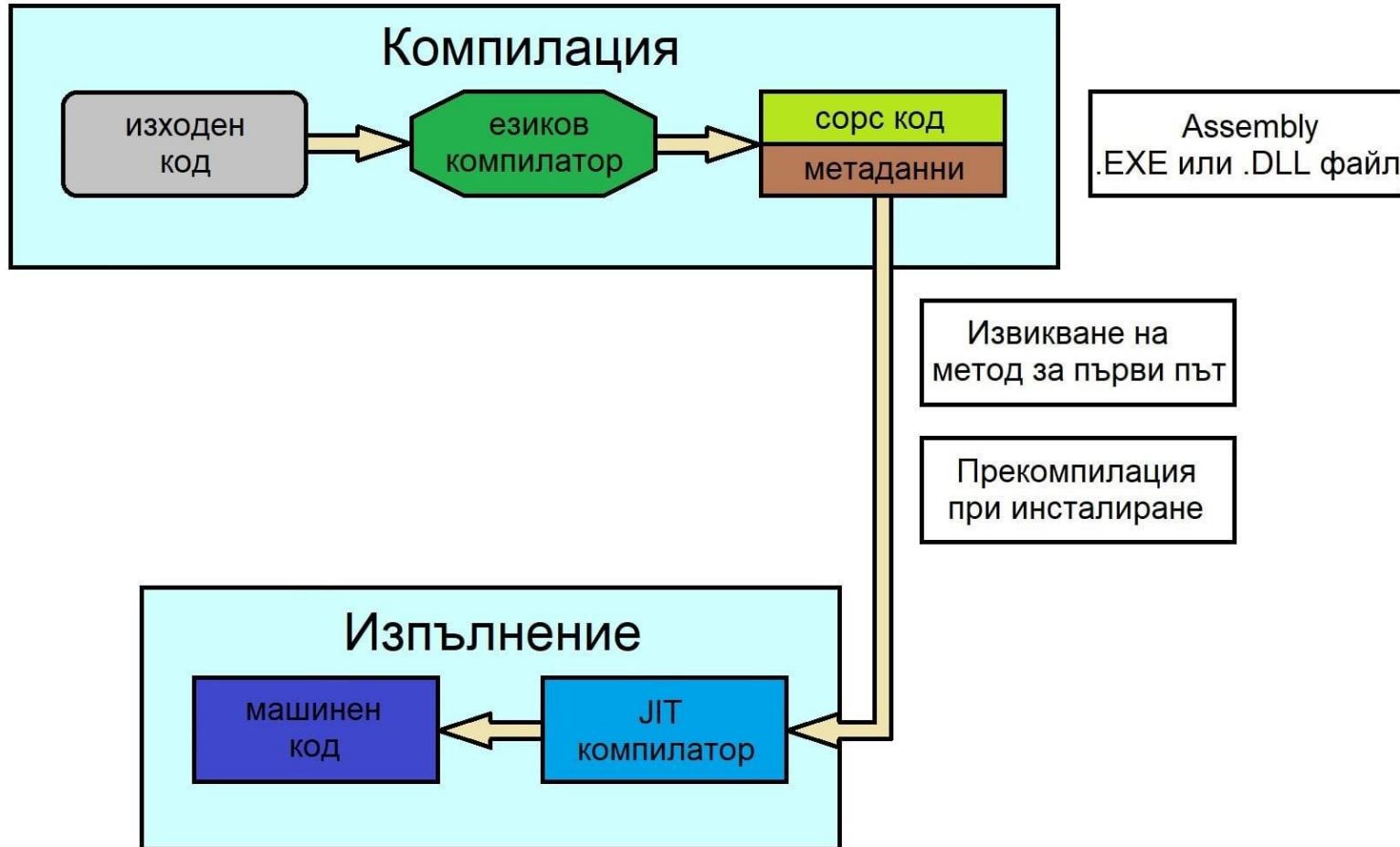
Неуправляван код (Win32)

- Няма защита на паметта и типовете
- Създава проблеми с надеждността
- Не съдържа метаданни, които го описват

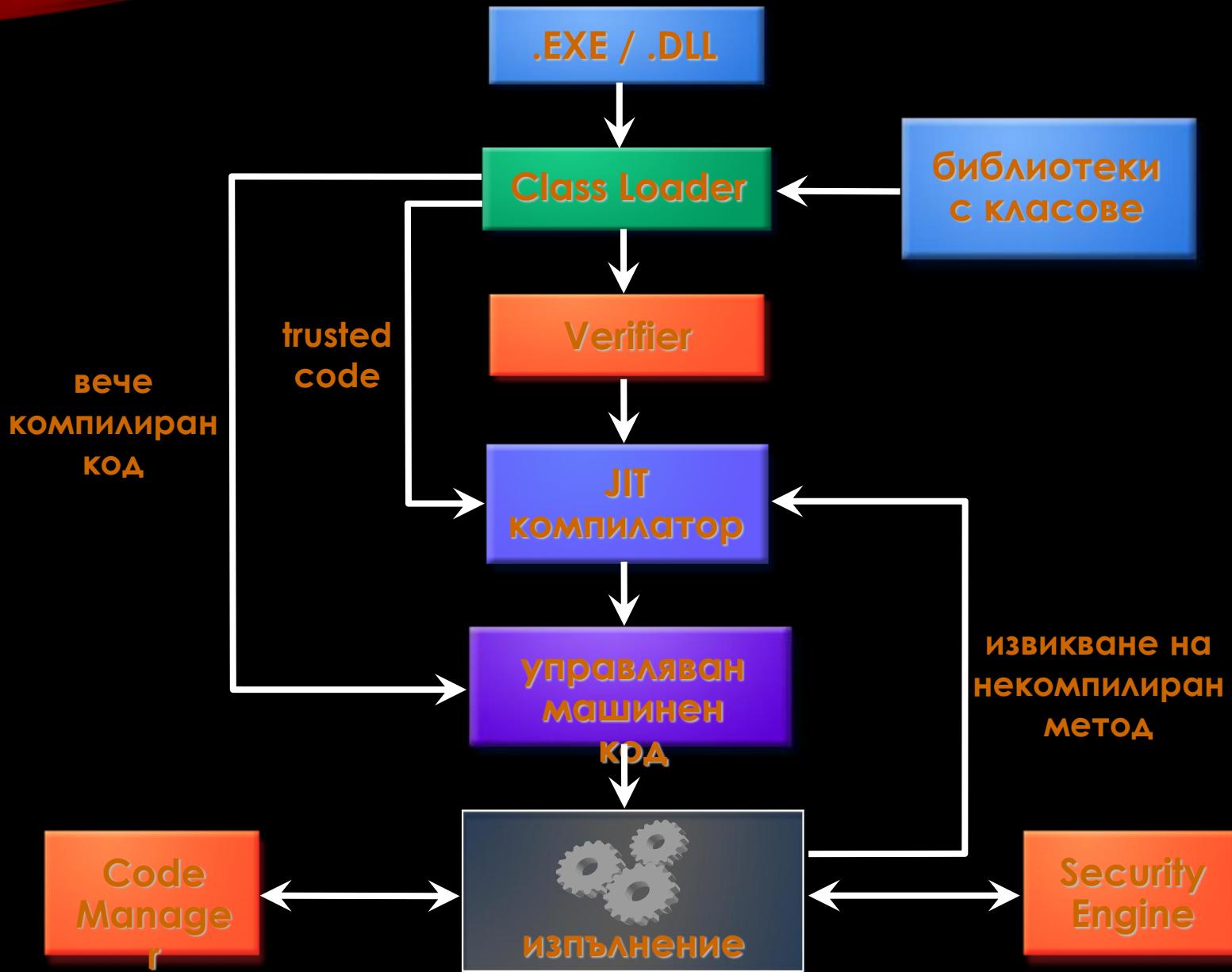
INTERMEDIATE LANGUAGE- ОБОБЩЕНИЕ

- Intermediate Language (IL, MSIL, CIL)
- Език от ниско ниво (машинен език)
- Стекова архитектура
- Като асемблер, но с обектно-ориентирани възможности:
 - инструкции за заделяне на памет
 - инструкции за предизвикване и обработка на изключения
 - инструкция за извикване на виртуален метод
- Позволява ефективно компилиране до машинен код за различни платформи
- Винаги се компилира преди изпълнение

Компиляция и изпълнение



Как CLR изпълнява IL кода



How to write a CV



**Физиономията ми,
когато госпожата по Интегрирани среди
каже някоя тъпа шега,
но ми трябва б за края на семестъра**



БИБЛИОТЕКИ ЗА ДИНАМИЧНО СВЪРЗВАНЕ

- DLL е съкращение от Dynamic Link Library (библиотека за динамично свързване). **Те са библиотечни файлове с компилиран код, който може да се използва в други приложни програми.**

Тези модули приличат на библиотечните (LIB) по това че също съдържат определени функционални възможности, пакетирани за употреба в приложните програми.

БИБЛИОТЕКИ ЗА ДИНАМИЧНО СВЪРЗВАНЕ

Различията се изразяват при свързване на приложната програма към библиотеката.

- При обикновените библиотечни модули (**LIB**) програмата се свързва с функционалните възможности по време на компилиране и изграждане.

Всички функционални възможности от библиотечния файл се превръщат в част от изпълнимия файл на програмата.

- При DLL файла приложната програма се свързва към функционалните възможности на библиотечния файл по време на своето изпълнение. Библиотеката остава самостоятелен файл, към който програмата извършва обръщения.

ПРЕДИМСТВА НА DLL ФАЙЛА ПРЕД LIB

- Употребата на DLL **влияе пряко върху големината на изпълнимия файл**, защото в него не се включват функционалните възможности пакетирани в библиотеката, а те от своя страна могат да се използват от различни приложни програми.
- Функционалните възможности на DLL **могат да се актуализират и модифицират, без да се налагат каквите и да е промени или допълнителни обработки в изпълнимия файл на програмата** (разбира се, ако не се коригира експортния интерфейс на DLL).
- **Възможност DLL да се използва с всеки от езиците за програмиране под WINDOWS.**

СЪЗДАВАНЕ И УПОТРЕБА НА DLL

- Библиотеките DLL предлагат определени функции и класове за приложните програми посредством експортиране на функции.
- Когато дадена функция се експортира, тя се добавя към таблица, включена в DLL. **В тази таблица се съдържа списък на всички експортирани функции, съдържащи се в DLL и тя се използва за намиране и извикване на всяка от тях.**
- Всяка функция която не е експортирана, не се добавя в таблицата и поради тази причина тя не може да бъде видяна или извикана по какъвто и да е начин от външна програма или DLL.

СЪЗДАВАНЕ И УПОТРЕБА НА DLL

Програмите могат да извикват функции от DLL по два способа:

1. Намира се мястото на нужната функция в DLL и се получава указател към нея, който се използва за обръщение към функцията.

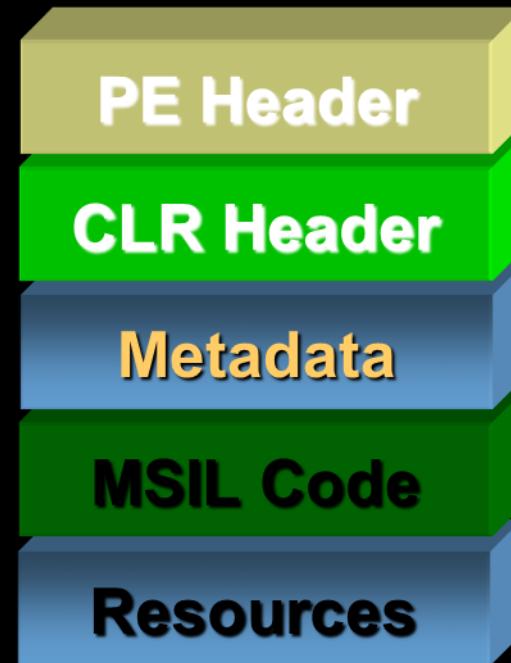
СЪЗДАВАНЕ И УПОТРЕБА НА DLL

2. При другия начин се осъществява свързване на приложната програма към файл LIB, създан с DLL.

- Файлът LIB се третира от свързыващия редактор като стандартен библиотечен файл.
- Файлът LIB съдържа **тапи (stubs)** за всяка от експортирани функции в DLL.
- **Тапата** представлява псевдо-функция, която притежава наименование и списък с аргументи, еднакви с тези на действителната функция.
- В тялото на функцията заглушалката е ограничено количество код, който извиква действителната функция от DLL, сякаш тя е част от кода на приложната програма и не е обособена в отделен файл.

ПРЕНОСИМИ ИЗПЪЛНИМИ ФАЙЛОВЕ

- Преносимите изпълними файлове (portable executables, PE)
- Представляват .EXE или .DLL файлове
- Състоят се от:



COMMON LANGUAGE INFRASTRUCTURE

Стандартизирана част от CLR: ISO 23271:2003

- Описва как приложения, написани на различни езици, могат да се изпълняват в различни среди без да се променят и прекомпилират
- .NET Framework – имплементация на CLI за Windows
- Mono – имплементация на CLI за UNIX и Linux –
<http://www.mono-project.com/>

COMMON LANGUAGE INFRASTRUCTURE

CLI стандартизира:

- Common Language Specification (CLS)
- Common Type System (CTS)
- Common Intermediate Language (CIL)
- Управление на изключения
- Асемблита и метаданни
- Части от .NET Framework Class Library

ПОДДРЪЖКА НА МНОГО ЕЗИЦИ

Common Language Specification (CLS)

- Съвкупност от правила, които всички .NET езици са задължени да спазват
- Специфицира минималните изисквания на всички .NET езици
- Осигурява съвместимост между езиците

Intermediate Language (IL, MSIL)

- Всички .NET езици се компилират до MSIL
- Осигурява съвместимост на компилирания .NET код

Common Type System (CTS)

- Обща система от типове за всички .NET езици
- Осигурява съвместимост на данните

COMMON LANGUAGE SPECIFICATION

- Система от правила и изисквания, на които отговарят всички .NET езици
- **Осигурява съвместимост и лесно взаимодействие между .NET езиците**
- Например: CLS налага всички .NET езици да са обектно-ориентирани
- **Някои .NET езици поддържат много повече от изискванията на CLS**
- При използване на нестандартни за CLS програмни средства и техники се губи съвместимост с останалите .NET езици

COMMON TYPE SYSTEM

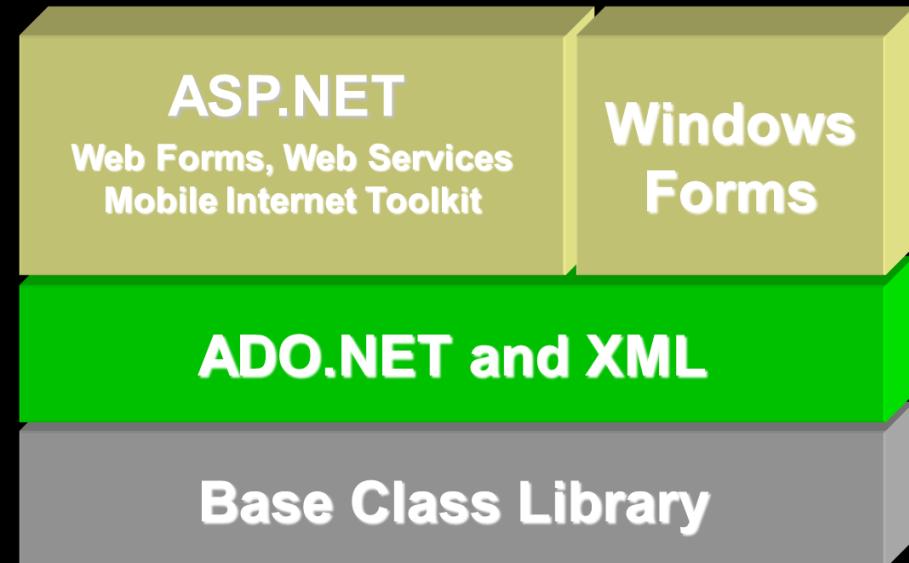
- Дефинира поддържаните от CLR типове данни и операции над тях
- **Осигурява съвместимост на данните между различните .NET езици – String в C# е същият като String във VB.NET**
- Два типа обекти – по стойност и по референция
- **Всички типове наследяват System.Object**
- String е примитивен тип

COMMON TYPE SYSTEM

- „Типът“ е общ термин, който може да се използва за обозначение на какъвто и да е елемент от множество (клас, интерфейс, структура, изброяване, делегат).

FRAMEWORK CLASS LIBRARY (FCL)

- Framework Class Library е стандартната библиотека на .NET Framework



ПАКЕТИТЕ ОТ БИБЛИОТЕКАТА FCL

SYSTEM.WEB

SYSTEM.WINDOWS.FORMS

SYSTEM.DRAWING

SYSTEM.DATA

SYSTEM.XML

SYSTEM

ПАКЕТИТЕ ОТ БИБЛИОТЕКАТА FCL

System.Web

Services
Description
Discovery
Protocols
Caching
Configuration

UI
HtmlControls
WebControls
Security
SessionState

System.Windows.Forms

Design
ComponentModel

System.Drawing

Drawing2D
Imaging
Printing
Text

System.Data

ADO
Design
SQL
SQLTypes

System.Xml

XSLT
XPath
Serialization

System

Collections
Configuration
Diagnostics
Globalization

IO
Net
Reflection
Resources

Security
ServiceProcess
Text
Threading

Runtime
InteropServices
Remoting
Serialization



Шерлок Холмс и доктор Уотсън летели с балон. Последвала изведнъж авария и балонът се приземил на сред блато. Седели двамата пътници вътре, а по пътя до блатото минал някакъв човек. Холмс му подвикнал:

- Хей, добри човече! Знаеш ли къде се намираме?

Мислил човекът, мислил, пък накрая изтърсиł:

- Вие се намирате в кошницата на балона на сред блатото. Заминал човекът и ги оставил втрещени. Холмс се обърнал към Уотсън и му казал:

- Ето този, скъпи Уотсън, беше програмист.

- Как познахте, Холмс?

- Елементарно, скъпи Уотсън. Първо: този човек мисли дълго време над прост въпрос. Второ: дава съвсем точен отговор. Трето: от този негов отговор ние нямаме никаква полза.

LET'S PLAY!

- Join at www.kahoot.it or with the Kahoot! app
- **Game PIN:**