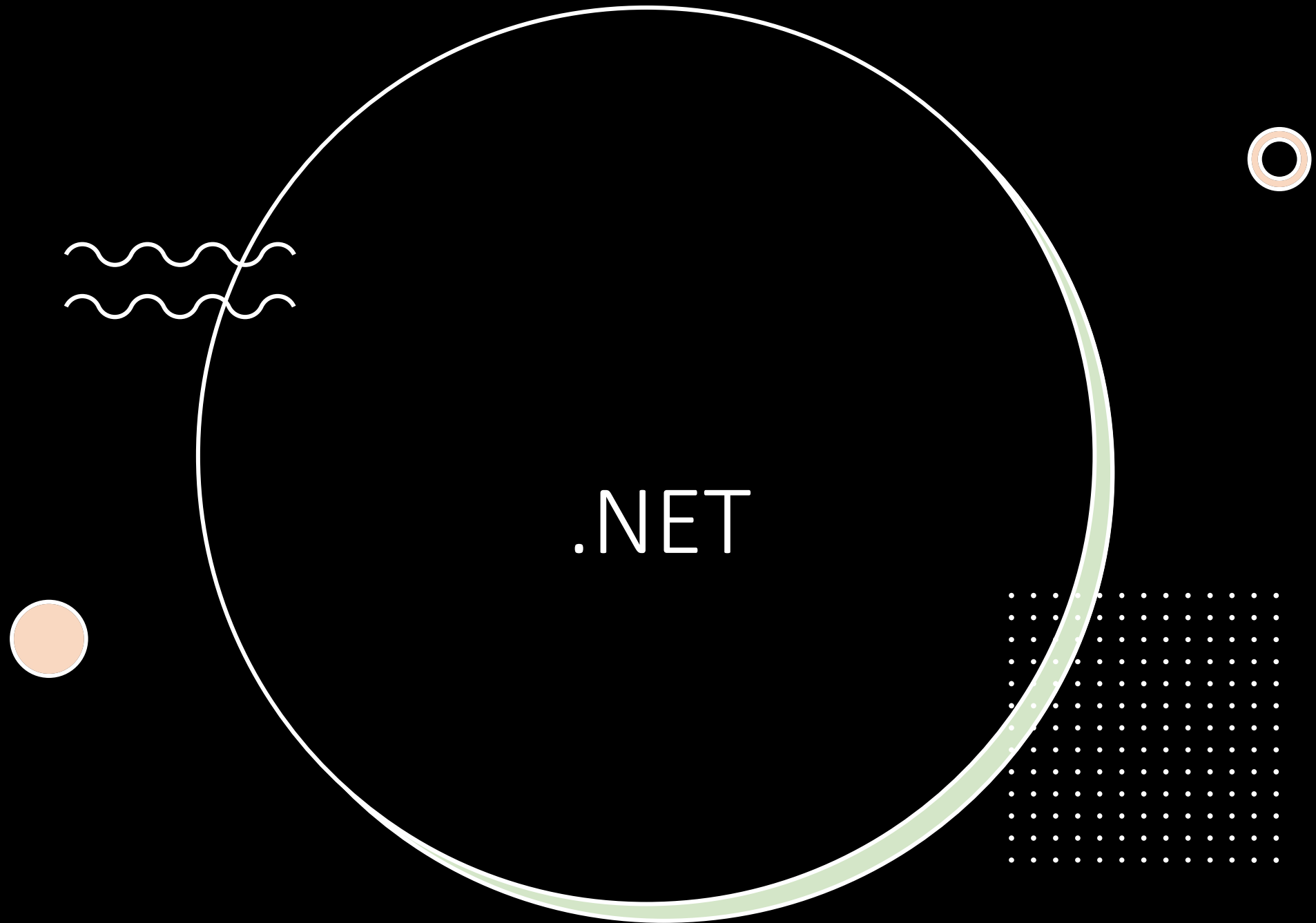
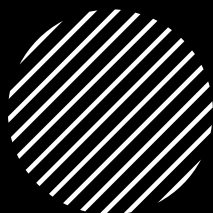
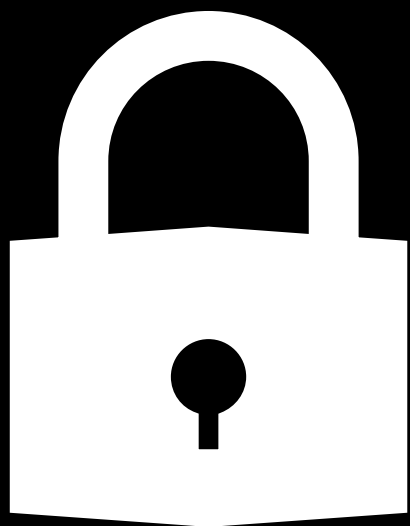


ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

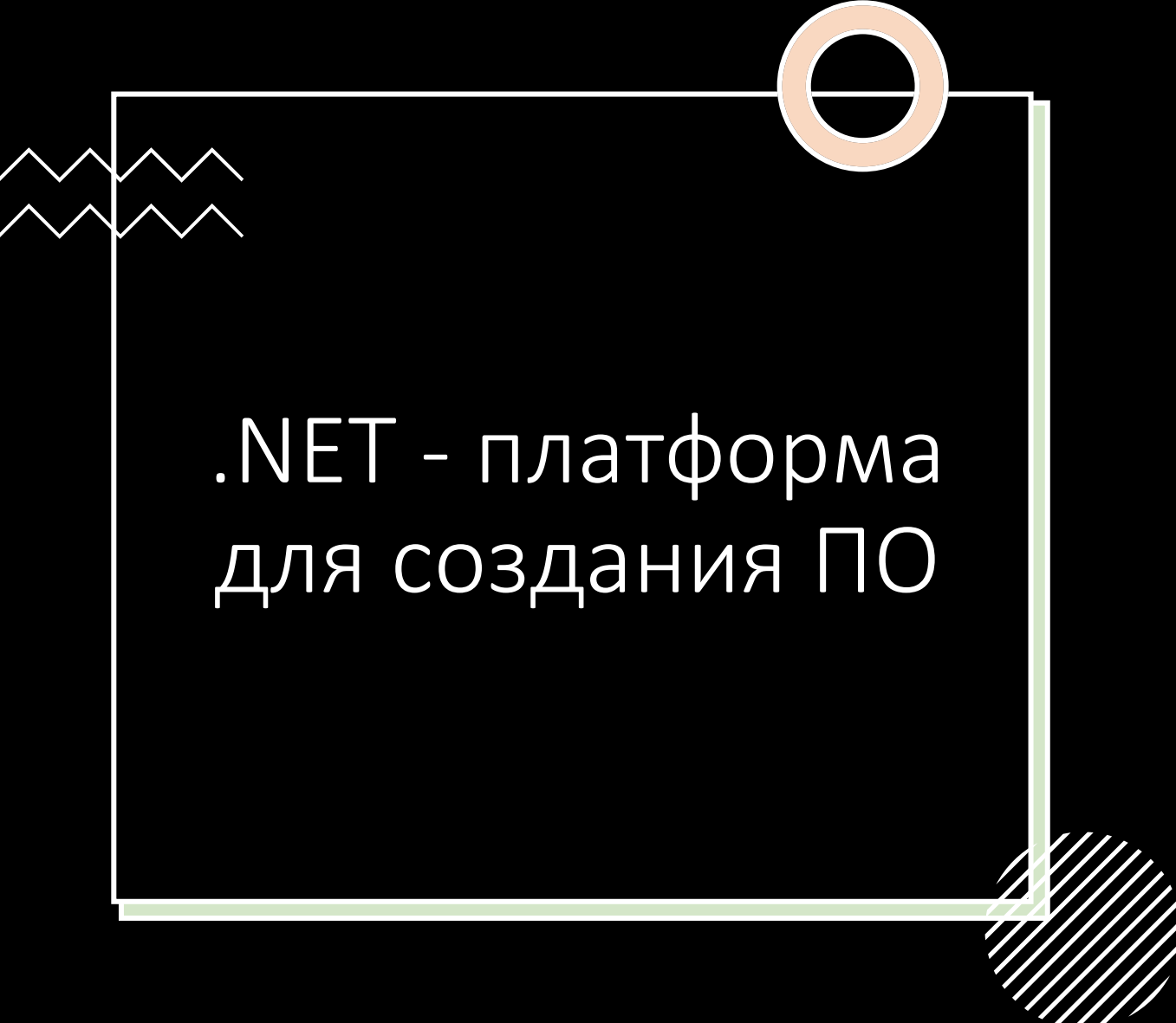
Лекция 5





С++ и его проблемы

- Утечки памяти и нарушение прав доступа
- Оптимизация под различные платформы только на этапе компиляции
- Большой размер занимаемой оперативной памяти
- Невозможность использования в программе вставок на других языках*



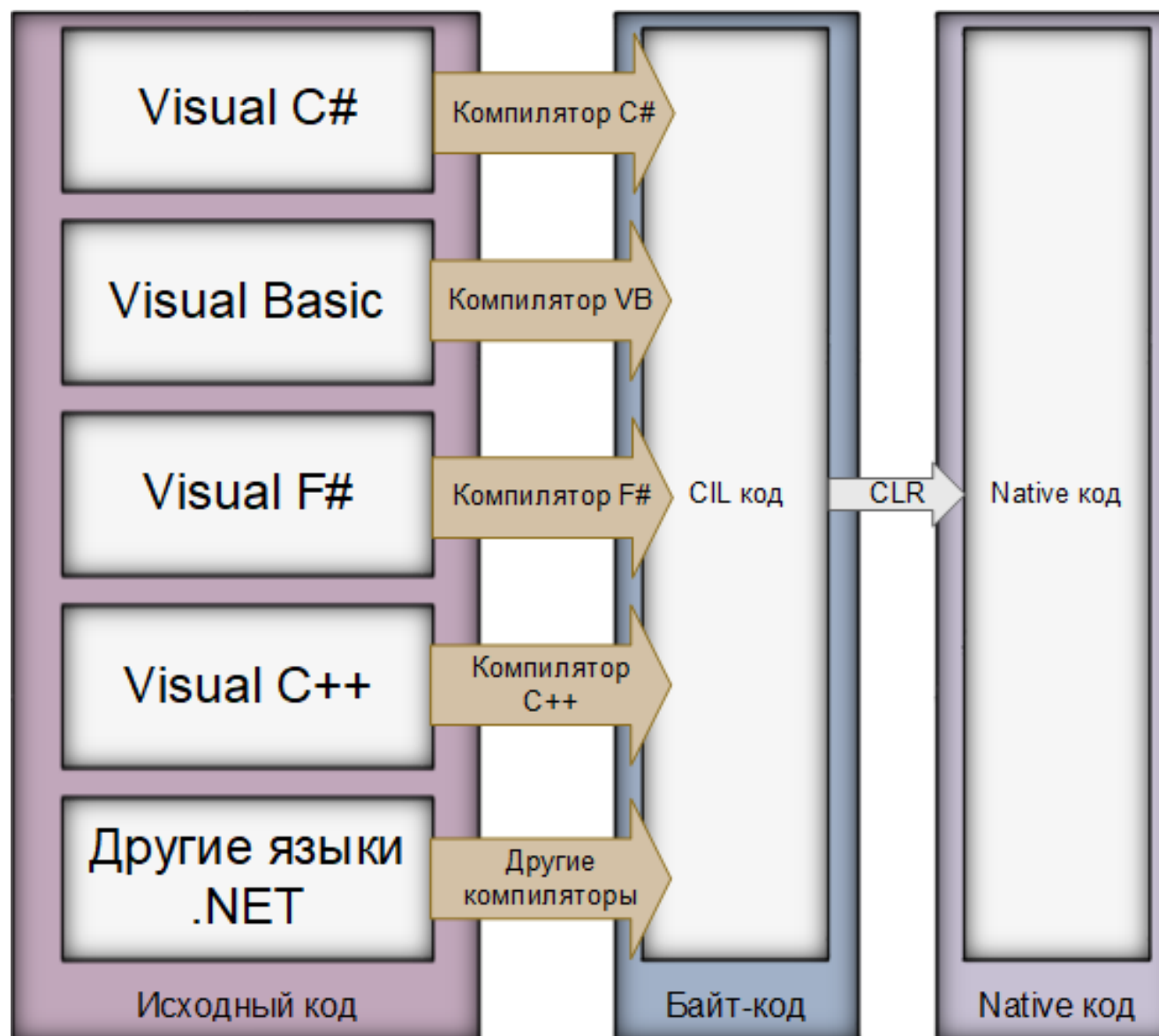
.NET - платформа для создания ПО

- С открытым исходным кодом
- Кроссплатформенная*
- Позволяет использовать одни и те же пространства имён, библиотеки и API для разных языков
- Первый релиз в 2002

* - в новых версиях, не .NET Framework

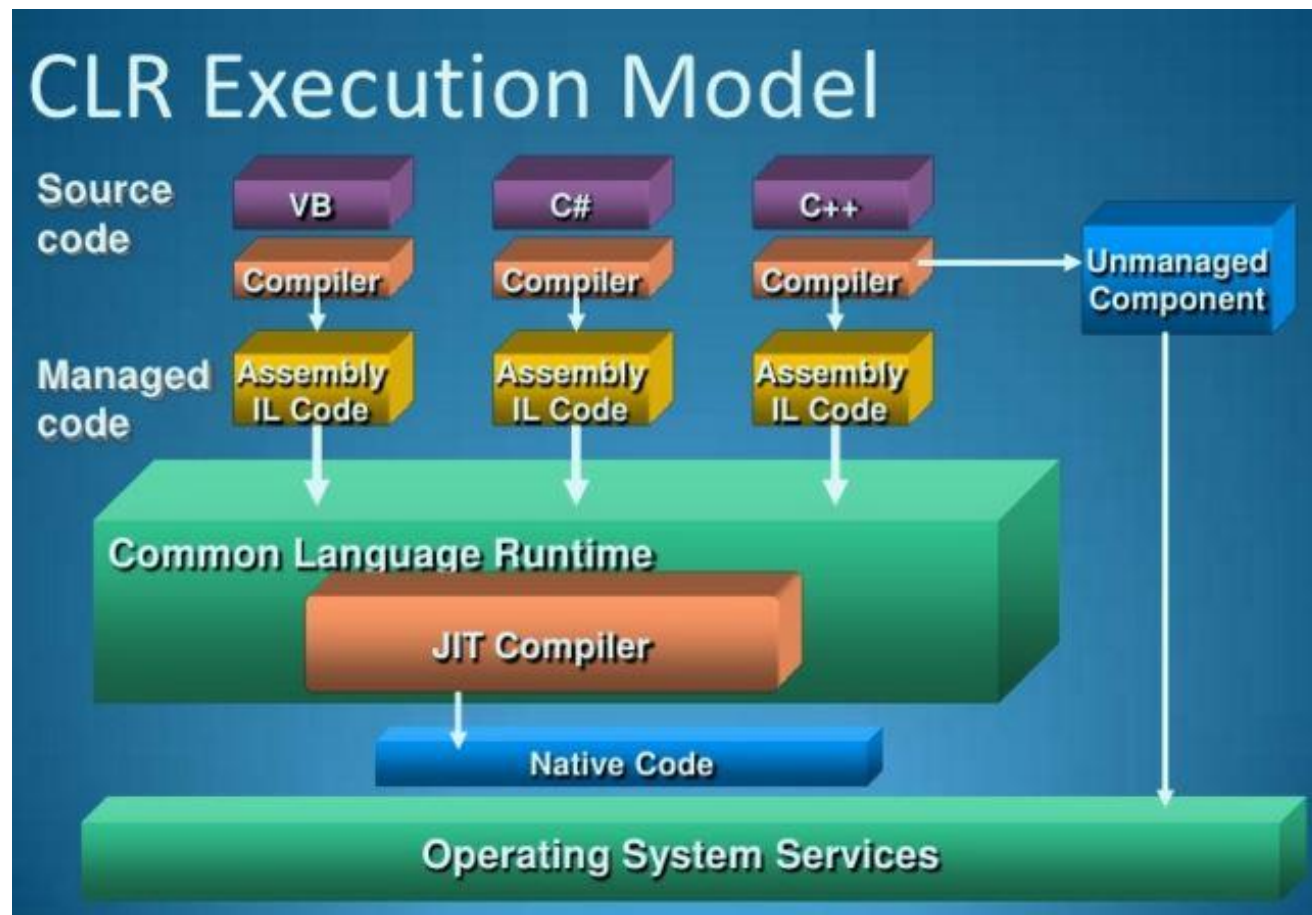
CLR И CLI


- CIL – Common Intermediate Language, платформонезависимый язык ассемблера для .NET
- CLR – Common Language Runtime, платформозависимая среда исполнения для CIL
- CLI – Common Language Infrastructure, общая инфраструктура, описывающая спецификации для сред, систем и метаданных в .NET



JIT-компилятор (Just In Time)

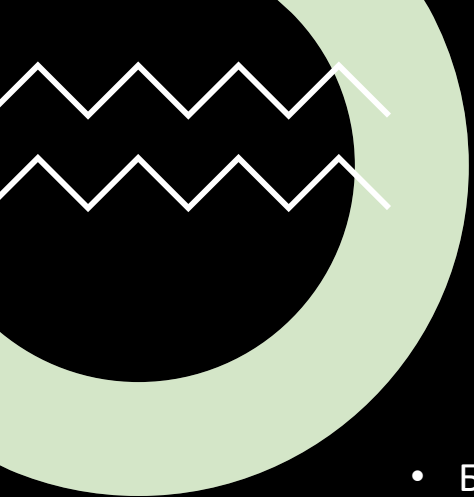
- Загрузка сборок по мере необходимости
- Минимизация затрачиваемой RAM
- Минимизация затрачиваемой памяти на диске
- Оптимизации кэш-промахов и размещения страниц
- Оптимизация под конкретную машину





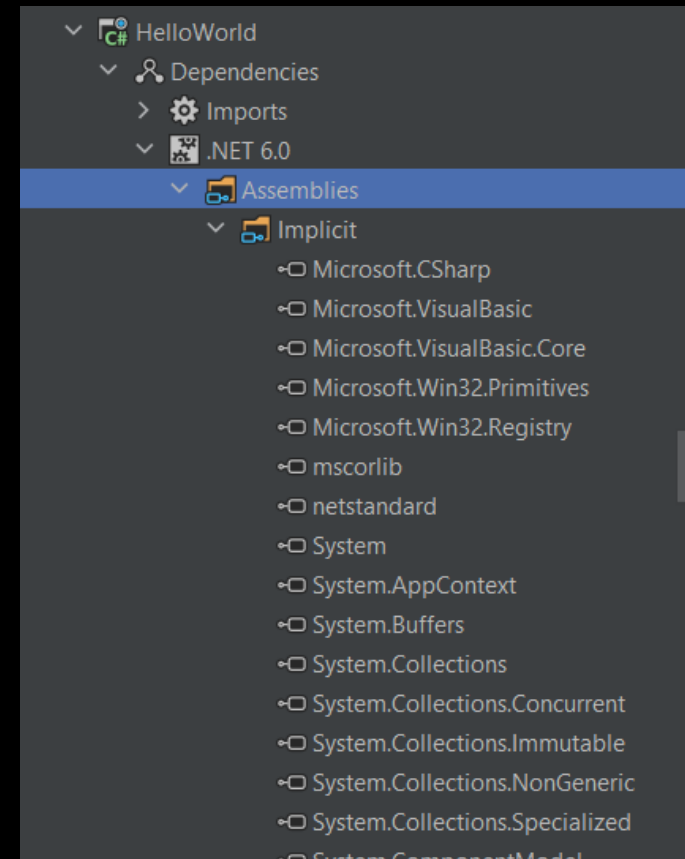
Управление памятью в CLR


- CLR осуществляет полное управление памятью в куче
 - Подсчёт ссылок
 - Дефрагментация
 - Изменение ссылок
 - Разрешение циклических зависимостей
 - Сборка мусора (Garbage Collector)



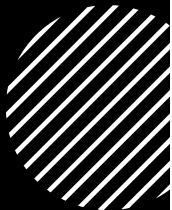

Сборка (проект)

- Базовая структурная единица в .NET, на уровне которой проходит версионирование, развертывание и конфигурация приложения.
- Сборка состоит из:
 - CIL кода
 - Манифеста (метаданные сборки)
 - Метаданных типов
 - Ресурсов (доп файлы)
- Сборка это один или несколько файлов, зачастую - .dll






Решение проблем C++ в .NET



Проблемы классических C++ приложений	Решение с помощью .Net
Утечки памяти и нарушение прав доступа	Сборщик мусора GC (Garbage Collector)
Оптимизация под различные платформы	Оптимизация под различные платформы – промежуточный код CIL (Common Intermediate Language) и общезыковая исполняющая среда CLR (Common Language Runtime)
Большой размер занимаемой оперативной памяти	Компиляция «на лету» с помощью JIT (Just In Time) компилятора
Использование в программе вставок на других языках	Объединение кода на разных языках в одну программу благодаря промежуточному коду



Какой
платформой
пользоваться?

.NET
Framework,
.NET Core, .NET
Standard ...

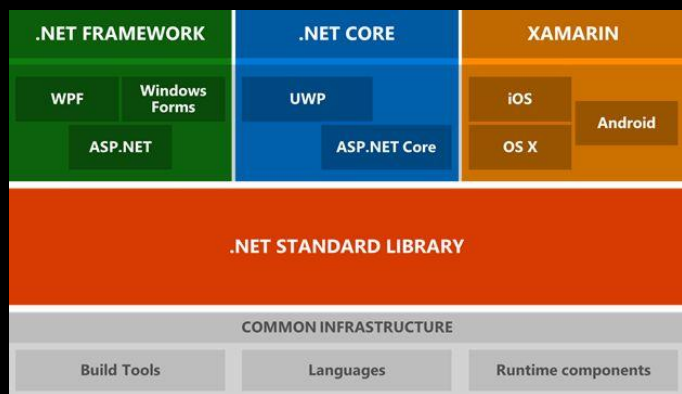




.NET Framework

- Первая реализация платформы
- Последняя версия 4.8, поддерживается, но новый функционал не добавляется
- Windows only
- *Забудьте...*

.NET Core и .NET Standard

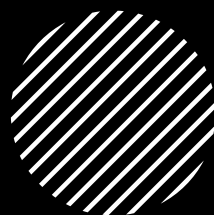


- 2016 год...
- форк .NET Framework, чья реализация была оптимизирована с учетом задач декомпозиции.
- Модульность
- Оптимизация алгоритмов для повышения производительности
- Ограниченная кроссплатформенность





.NET 5, 6, 7...



- Наследник .NET Core
- Open source
- Кроссплатформенность
- Дальнейшие оптимизации



The image features a large, thin white circle centered on a black background. Inside this circle, the text "А на чем писать UI?" is written in a white, sans-serif font. Surrounding the circle are several decorative elements: two white wavy lines to the upper left, a small orange circle with a white outline to the lower left, a small orange circle with a white outline to the upper right, and a grid of small white dots to the lower right. The circle itself has a thick green border on its right side.

А на чем писать
UI?

Win-forms,
WPF, MAUI
etc...



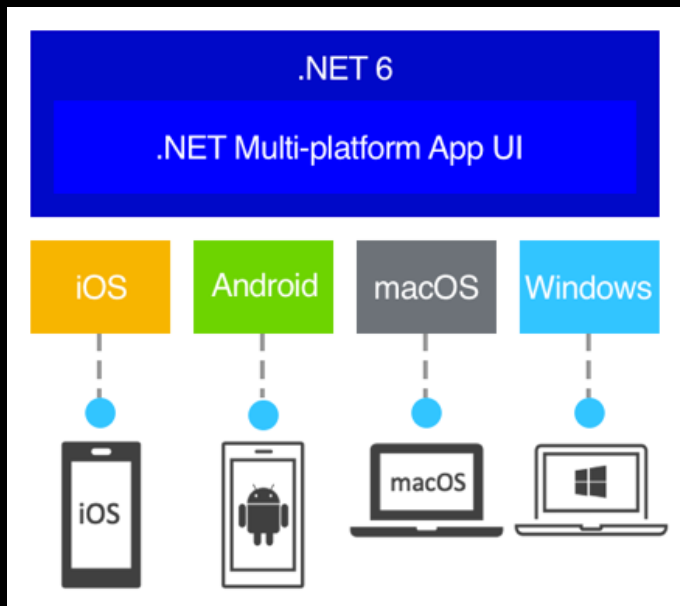


Win-Forms & WPF

- Оба - Windows only =(
- Win-Forms – UI при помощи паттерна MVP
- WPF – чуть новее...
использует паттерн MVVM




.NET MAUI



- Кросс-платформенная платформа для создания мобильных и классических приложений с помощью C# и XAML.
- Наследник Xamarin forms.

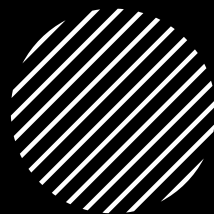




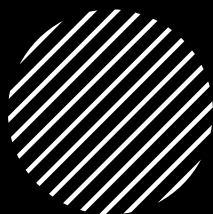
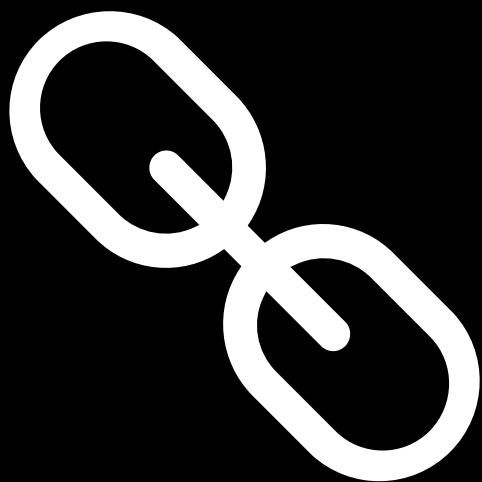
Устройство памяти в .NET



Стек и куча

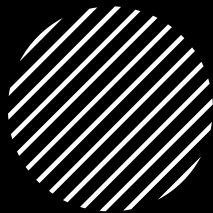
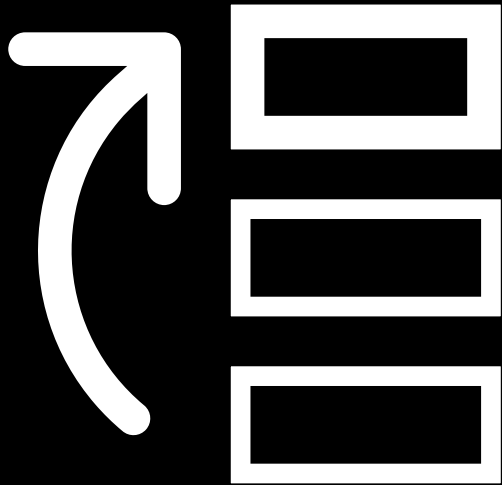


- Стек
 - Имеет фиксированный размер
 - Не требует очистки (очищается автоматически при выходе за область видимости).
- Куча
 - Имеет динамический размер, ограниченный только объемом RAM*
 - Операции выполняются дольше чем на стеке



Reference type

- Ссылки хранятся на стеке, объект хранится в куче
- Передача – по ссылке
- Требуется очистка памяти => нагрузка на GC
- Все классы – reference type



Value type

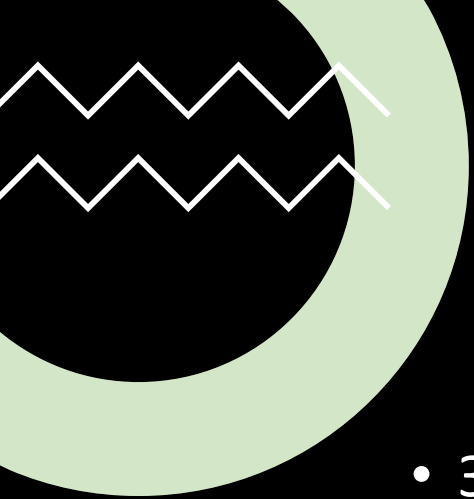
- Хранится на стеке
- Стек ограничен => слишком большие value type «объекты» будут его быстро забивать
- Нет работы с кучей => нет нагрузки на GC => выше производительность
- Передаются по значению



Boxing

- Если работать с value type как с Object (классом) произойдет boxing - упаковка в «объект» и размещение в куче
- Это создает накладные расходы
- Если привести «запакованный» value type обратно – произойдет распаковка – т.е. перемещение из кучи на стек






Garbage Collector (GC)

- Задача GC – очистка кучи от уже не используемых объектов
- Куча (ссылки) делится на поколения (0, 1, 2)
- GC
 - проходит по объектам поколения, если на них нет корневых ссылок – удаляет
 - Если объект не был очищен несколько раз – он перемещается в следующее поколение.
 - Время от времени запускается дефрагментация.
 - Кольцевые зависимости
 - Large object heap

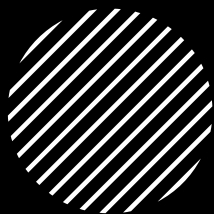




С# первое погружение

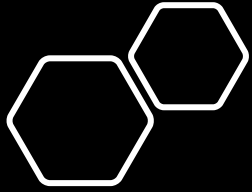


Что такое C#



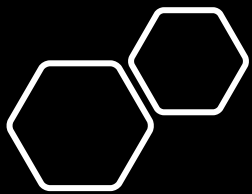
- Объектно-ориентированный ЯП, с элементами:
 - Функционального
 - Событийного
 - ...
- Со статической типизацией
- Под платформу .NET
- Ориентирован на компонентную разработку





Hello world

```
namespace HelloWorld
{
    internal static class Program
    {
        private static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

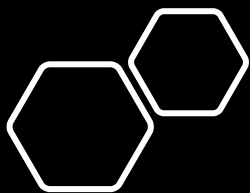


Пространства имен

- Используются для организации элементов кода и для создания глобально уникальных типов
- Подробнее [тут](#)

```
namespace BaseConstructions
{
    public class NamespacesExample
    {
        public void Bar()
        {
            Nested.NestedClass.Foo();
        }
    }

    namespace Nested
    {
        public static class NestedClass
        {
            public static void Foo()
            {
                Console.WriteLine("Foo");
            }
        }
    }
}
```

Типы данных

```
// Variables example.
```

```
bool boolVar = true;
```

```
int intVar = 42;
```

```
double doubleVar = 42.0;
```

```
int[] array = new[] {1, 2, 3, 4, 5 };
```

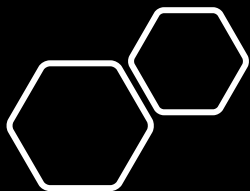
```
string stringVar = "Good string";
```

```
char charVal = stringVar[0];
```

```
ClassExample classVariable = new ClassExample();
```

```
EnumExample enumVar = EnumExample.First;
```

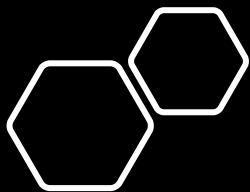
```
var autoVariable = doubleVar - 10;
```



Условные конструкции

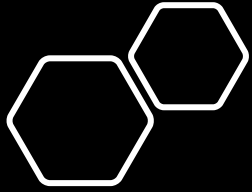
- Тернарный оператор тоже есть

```
// Conditions example.  
if (boolVar)  
    Console.WriteLine(stringVar);  
else  
    Console.WriteLine(intVar);  
  
switch (enumVar)  
{  
    case EnumExample.First:  
        break;  
    case EnumExample.Second:  
        break;  
    default:  
        throw new ArgumentOutOfRangeException();  
}
```



ЦИКЛЫ

```
// Cycles example.  
for (int i = 0; i < array.Length; i++)  
|   Console.Write(array[i].ToString() + " ");  
Console.WriteLine();  
  
var flag = true;  
var counter = 0;  
  
while (flag)  
|   flag = (++counter) == intVar;  
  
// or  
do  
{  
|   flag = (++counter) == intVar;  
} while (flag);
```



Enums

```
namespace BaseConstructions;
```

```
internal enum EnumExample
```

```
{
```

```
    First = 0,
```

```
    Second
```

```
}
```

Классы в C#

- Нет разделения на объявления и определения
- Нет множественного наследования
- Могут реализовывать интерфейсы – аналог объявлений
- Все наследуются от Object (автоматически)

```
namespace BaseConstructions;

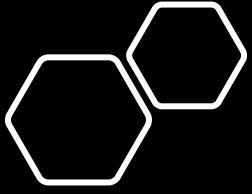
internal class ClassExample
{
    private int _privateField;

    private const int ConstExample = 42;

    public int PublicMethod(int argument)
    {
        _privateField = ConstExample + argument;
        PrivateLogExample();
        return _privateField;
    }

    private void PrivateLogExample()
    {
        Console.WriteLine("Log action");
    }

    public int PublicProperty { get; set; }
```




Класс Object

```
using System.Runtime;
using System.Runtime.ConstrainedExecution;
using System.Runtime.InteropServices;
using System.Security;

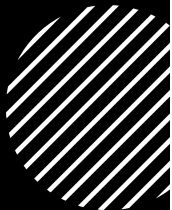

namespace System
{
    ...public class Object
    {
        ...public Object();

        ...~Object();


        ...public static bool Equals(Object objA, Object objB);
        ...public static bool ReferenceEquals(Object objA, Object objB);
        ...public virtual bool Equals(Object obj);
        ...public virtual int GetHashCode();
        ...public Type GetType();
        ...public virtual string ToString();
        ...protected Object MemberwiseClone();
    }
}
```



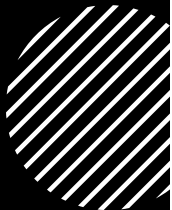

Приоритет операций



Приоритет	Категория	Операции	Порядок
0	Первичные	(expr); x.y; f(x); a[x]; x++; x new; sizeof(t);	Слева направо
1	Унарные	+ - ! ~ ++x --x (T)x	Слева направо
2	Мультипликативные (Умножение)	- * / %	Слева направо
3	Аддитивные (Сложение)	+ -	Слева направо
4	Сдвиг	<< >>	Слева направо
5	Отношения, проверка типов	< > <= >= is as	Слева направо
6	Эквивалентность	== !=	Слева направо
7	Логическое И	&	Слева направо
8	Логическое исключающее ИЛИ (XOR)	^	Слева направо
9	Логическое ИЛИ (OR)		Слева направо
10	Условное И	&&	Слева направо
11	Условное ИЛИ		Слева направо
12	Условное выражение	? :	Справа налево
13	Присваивание	= *= /= %= += -= <<= >>= &= ^= =	Справа налево



Полезные ССЫЛКИ



- Среды разработки:
 - Visual Studio
 - Rider
- Материалы к лекциям:
 - <https://github.com/Sych474/BMSTU-app-programming-languages>
- Полезные сайты
 - <https://docs.microsoft.com/ru-ru/dotnet/csharp/>
 - <https://metanit.com/sharp/tutorial/>