



IT не волшебство: как он работает и как не мешать

Егоров Дмитрий
Head of backend Artsoft



План доклада

Систематизируем “ГДЕ, ЗАЧЕМ,
ЧТО и КАК” делает JIT

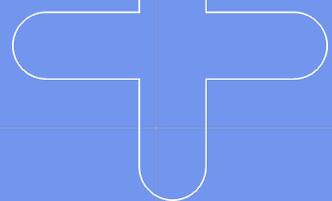


Развитие JIT в последних версиях .NET

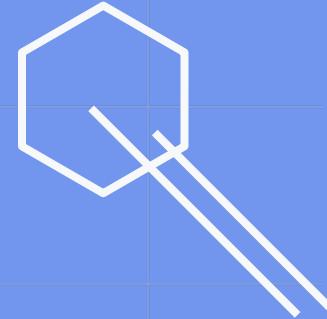


Оптимизация работы с памятью
благодаря JIT



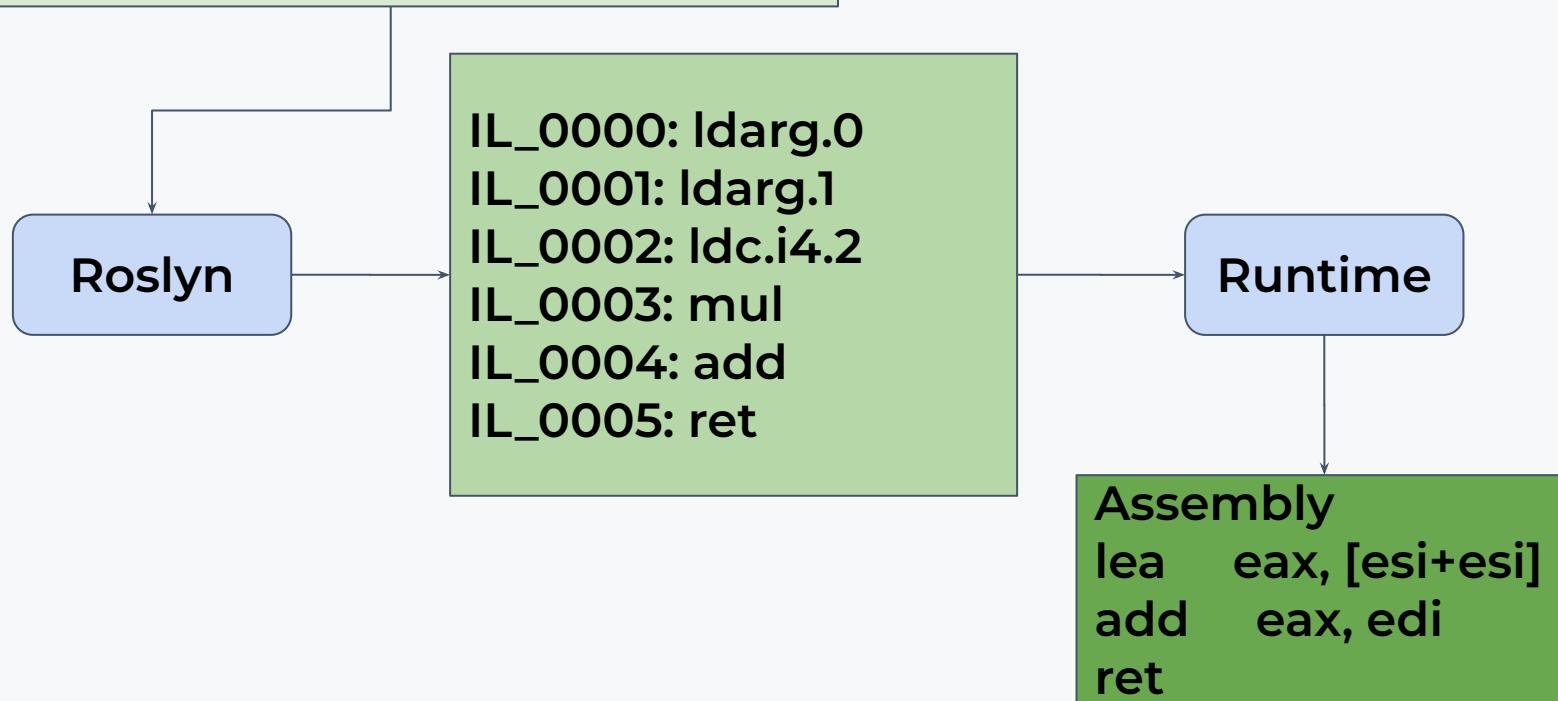


Систематизируем “ГДЕ, ЗАЧЕМ,
ЧТО и КАК” делает JIT

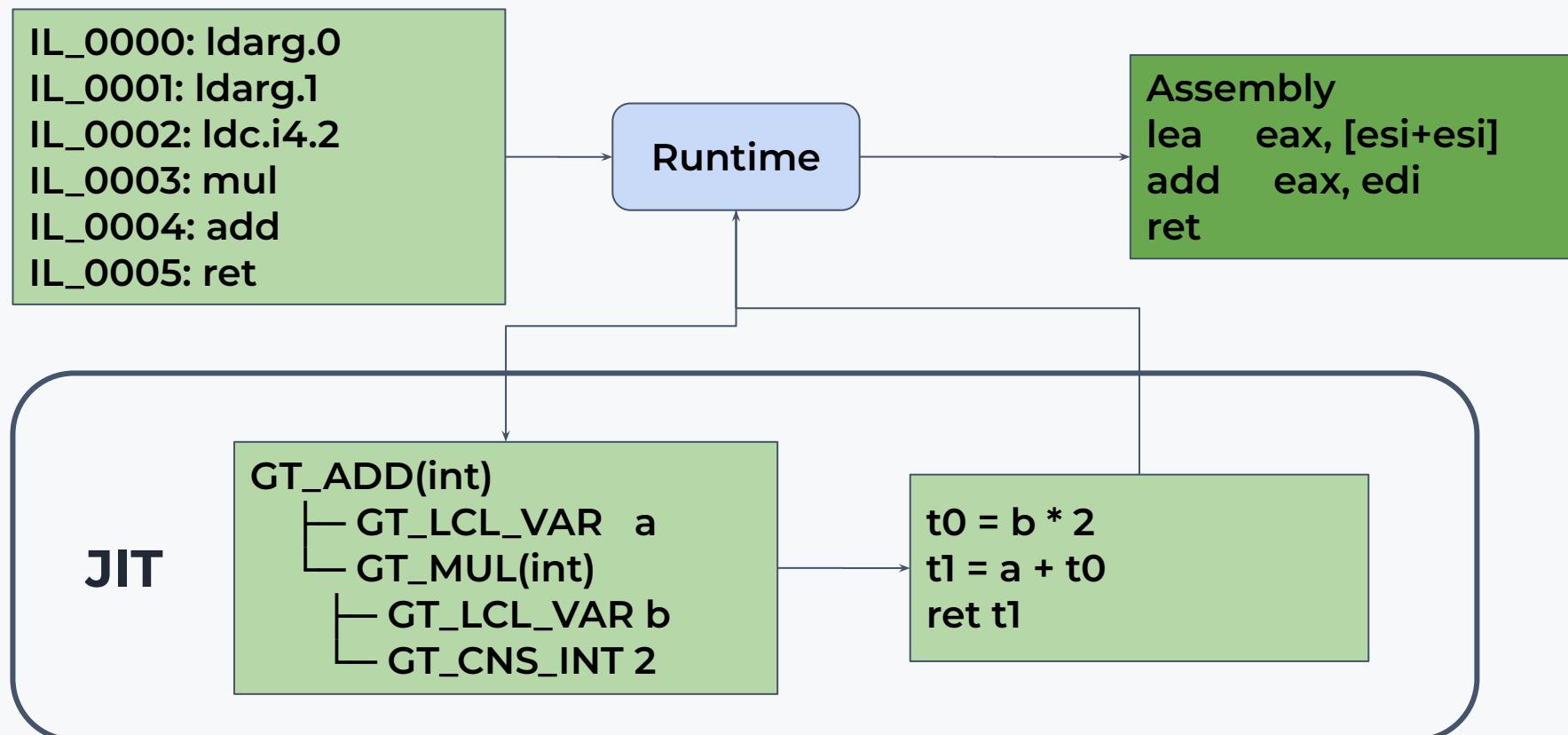


(ГДЕ) Схема компиляции C#

```
int AddMul(int a, int b) => a + 2 * b;
```



(ГДЕ) Расширенная схема компиляции c#



(ГДЕ) Формы и этапы

Morph / Оптимизации

- Встраивание методов
- Свертка констант
- Удаление мертвого кода

SSA + Классические оптимизации

- устранение повторных вычислений
- вынос инвариантов из циклов

Lowering (адаптация под
целевую архитектуру)

(ГДЕ) Стадии работы JIT

[runtime](#) / [src](#) / [coreclr](#) / [jit](#) / [compiler.cpp](#)

Code

Blame

10908 lines (9357 loc) · 365 KB

```
4270     void Compiler::compCompile(void** methodCodePtr, uint32_t* methodCodeSize, JitFlags* c
4271     DoPhase(this, PHASE_CREATE_FUNCLETS, &Compiler::fgCreateFunclets);
4272 }
4273
4274 // Expand casts
4275 DoPhase(this, PHASE_EXPAND_CASTS, &Compiler::fgLateCastExpansion);
4276
4277 // Expand runtime lookups (an optimization but we'd better run it in tier0 too)
4278 DoPhase(this, PHASE_EXPAND_RTLLOOKUPS, &Compiler::fgExpandRuntimeLookups);
4279
4280 // Partially inline static initializations
4281 DoPhase(this, PHASE_EXPAND_STATIC_INIT, &Compiler::fgExpandStaticInit);
4282
4283 // Expand thread local access
4284 DoPhase(this, PHASE_EXPAND_TLS, &Compiler::fgExpandThreadLocalAccess);
4285
4286 // Expand stack allocated arrays
4287 DoPhase(this, PHASE_EXPAND_STACK_ARR, &Compiler::fgExpandStackArrayAllocations);
4288
4289 // Insert GC Polls
4290 DoPhase(this, PHASE_INSERT_GC_POLLS, &Compiler::fgInsertGCPolls);
4291
4292
4293
4294
4295
4296
4297
4298
4299
```

(ГДЕ) Схема в/д с runtime



```
G_M000_IG02:          ; offset=0x000A
    mov    rdi, 0x7FD9A27FBEA8
    call   CORINFO_HELP_NEWSFAST
    mov    dword ptr [rax+0x08], ebx
    mov    esi, dword ptr [rax+0x08]
    mov    rdi, rax
    mov    rax, 0x7FD9A2638FF0
    call   rax
    add    eax, eax
```



```
call   CORINFO_HELP_ASSIGN_REF
mov    rsi, 0x7F5B4EE78A08
mov    qword ptr [rbx+0x18], rsi
mov    esi, dword ptr [r15+0x08]
mov    rdi, qword ptr [rbx+0x08]
call   [rbx+0x18]System.Func`2[int,int]:Invoke(int):int:this
```



```
G_M000_IG02:          ; offset=0x0001
    mov    rdi, 0x7F8521C71D90
    mov    esi, 1
    call   CORINFO_HELP_NEWARR_1_OBJ
    mov    rdi, 0x7F859BFE3718
    mov    qword ptr [rax+0x10], rdi
    mov    rdi, rax
```

```
G_M000_IG03:          ; offset=0x0042
    add    rsp, 8
    tail.jmp [Tests:<Test>g__Process|0_0(System.String[])]
```

отдельный вопрос про
рефлексию в следующих
сериях!

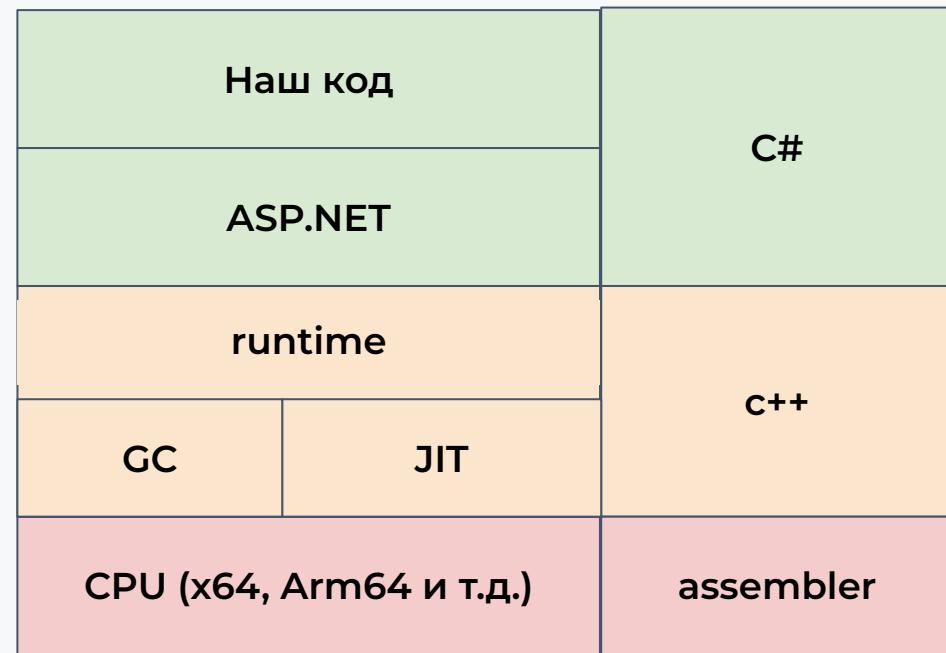
(ЗАЧЕМ) memory model .net

Связь с JIT

- 1) Ограничивает JIT в оптимизациях
- 2) JIT борется с ее избыточностью
- 3) JIT ее выполняет

- 1) Выравнивание памяти
- 2) Атомарность операций
- 3) Unmanaged memory access (проверка границ массивов)
- 4) Доступ к памяти не имеет сайд эффектов
- 5) Гарантирует Cross-thread access to local variables
- 6) Ограничение переупорядочивания READ & Write
- 7) & e t c

Что и на каком слое делает JIT ?

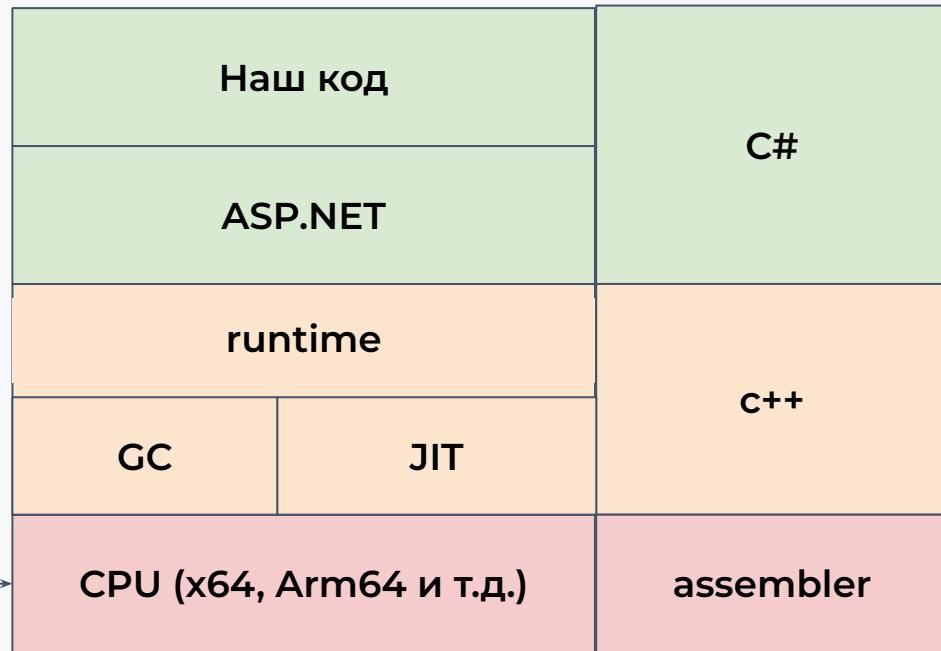


Что и на каком слое делает JIT ?

Не вошло, но тоже очень важно:

Разложение переменных на
регистры (что бы не было спилов)

Подбирает наилучшие
процессорные инструкции
`internal`



Уменьшение числа операторов



```
static int Test1(int a, int b, int c) => a * b + c;
```



; Старый код:

```
mul      w0, w0, w1  
add      w0, w0, w2
```

; Новый код:

```
madd    w0, w0, w1, w2
```

Зануление стека .net9

это места для
применения
векторных
оптимизаций

например:
vmovdqa ->
vmovdqu32

```
class Demo
{
    public static unsafe void WritePattern()
    {
        Span<byte> s = stackalloc byte[32];
        s.Fill(0xAA);
    }

    [SkipLocalsInit]
    public static unsafe void ReadUninitialized()
    {
        Large x;
        Unsafe.SkipInit(out x);
        var bytes = new ReadOnlySpan<byte>(x.Buf, length: 32);
        Console.WriteLine(Convert.ToString(bytes));
    }
}

public static void Main()          2000000000000000100000
{                                0000000000D8298C6D010
    WritePattern();               000000300000000000000000
    ReadUninitialized();          }
}
```

Обнуление полей структуры

```
struct MyStruct
{
    string gcl;
    long a;
    long b;
    long c;
    long d;
    long e;
    long f;
    long g;
    long h;
}
```

```
MyStruct Test() => new();
```

```
xor    eax, eax
mov    qword ptr [rdx], rax      ; GC slot
mov    qword ptr [rdx+0x08], rax
mov    qword ptr [rdx+0x10], rax
mov    qword ptr [rdx+0x18], rax
mov    qword ptr [rdx+0x20], rax
mov    qword ptr [rdx+0x28], rax
mov    qword ptr [rdx+0x30], rax
mov    qword ptr [rdx+0x38], rax
mov    qword ptr [rdx+0x40], rax
mov    rax, rdx
ret
; Total bytes of code: 41
```

Обнуление полей структуры



```
struct MyStruct
{
    string gc1;
    long a;
    long b;
    long c;
    long d;
    long e;
    long f;
    long g;
    long h;
}
```

```
MyStruct Test() => new();
```



```
xorps    xmm0, xmm0
vmovdqu32 zmmword ptr [rdx+0x08], zmm0
mov      rax, rdx
ret
; Total bytes of code: 23
```

Дефолтное поведение

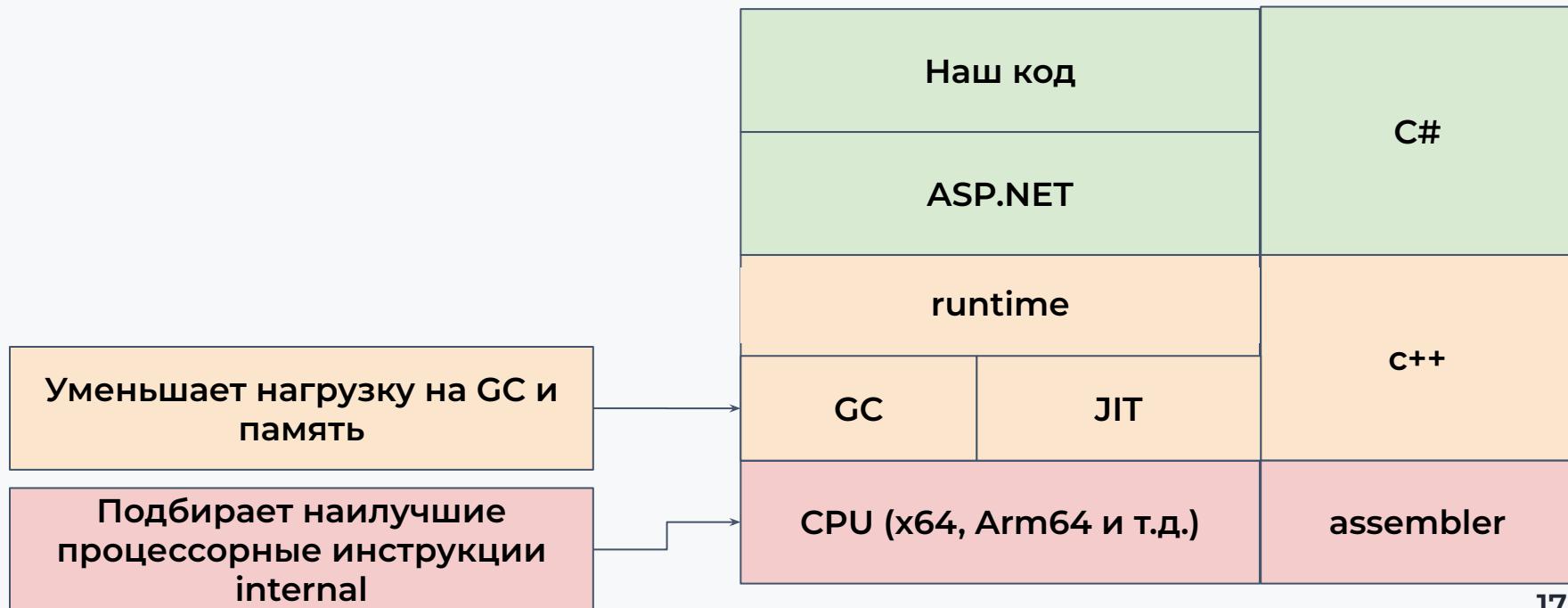


```
[StructLayout(LayoutKind.Sequential)]  
struct MyStruct { ... }
```



```
[StructLayout(LayoutKind.Auto)]  
class MyClass { ... }
```

Что и на каком слое делает JIT ?



Unbox



```
if (o is IDisposable d) d.Dispose(); // Boxing!
```



```
if (typeof(T).IsAssignableFrom(typeof(IDisposable))) {  
    ((IDisposable)(object)o).Dispose(); // вручную  
}
```

impBoxPatternMatchn



```
struct S { public int Value; }

static S SideEffectingMethod()
{
    Console.WriteLine("called");
    return new S { Value = 42 };
}

static void Test()
{
    object o = SideEffectingMethod(); // boxing
    if (o == null)
        Console.WriteLine("null");
}
```

Struct Promotion (не создавать всю целиком)



```
public ulong GetTime()
{
    var stat = new ParsedStat();
    return stat.utime + stat.stime;
}

internal struct ParsedStat
{
    internal int pid;
    internal string comm;
    internal char state;
    internal int ppid;
    internal int session;
    internal ulong utime;
    internal ulong stime;
    internal long nice;
    internal ulong starttime;
    internal ulong vsize;
    internal long rss;
    internal ulong rsslim;
}
```

В будущем мы увидим как лаконично эта оптимизация встраивается в общий pipeline

Т е она цена не просто сама по себе

Stack Allocation

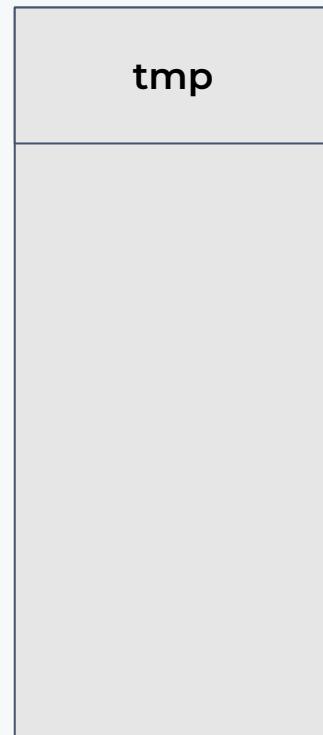


```
class Acc { public int A, B; }
int Use(Acc s) => s.A + s.B;

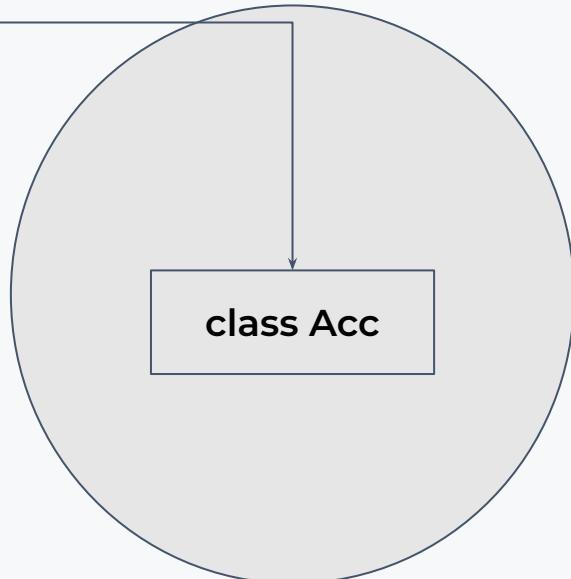
int Demo()
{
    var tmp = new Acc
    {
        A = 10,
        B = 20
    };

    return Use(tmp);
}
```

Stack



heap



Stack Allocation

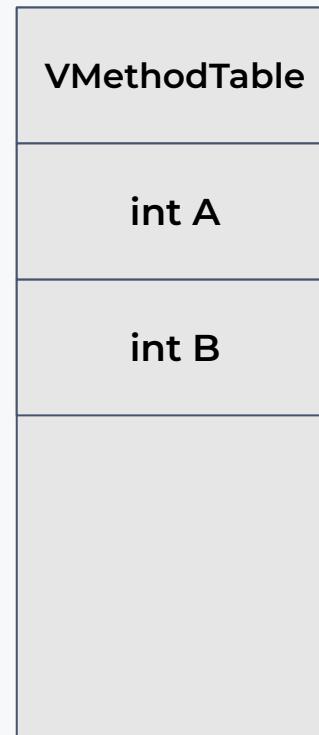


```
class Acc { public int A, B; }
int Use(Acc s) => s.A + s.B;

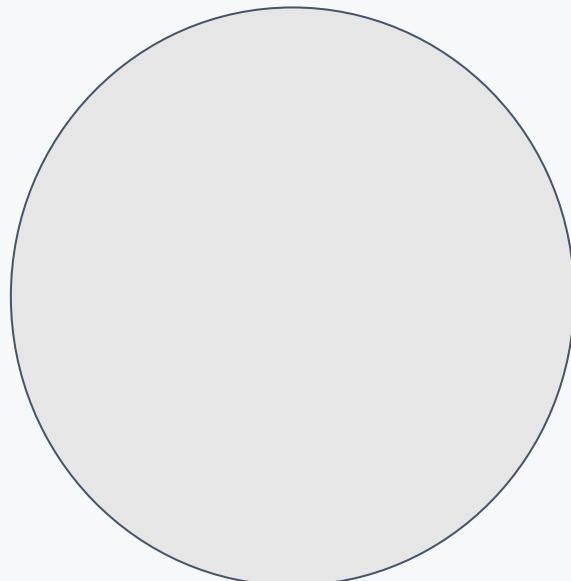
int Demo()
{
    var tmp = new Acc
    {
        A = 10,
        B = 20
    };

    return Use(tmp);
}
```

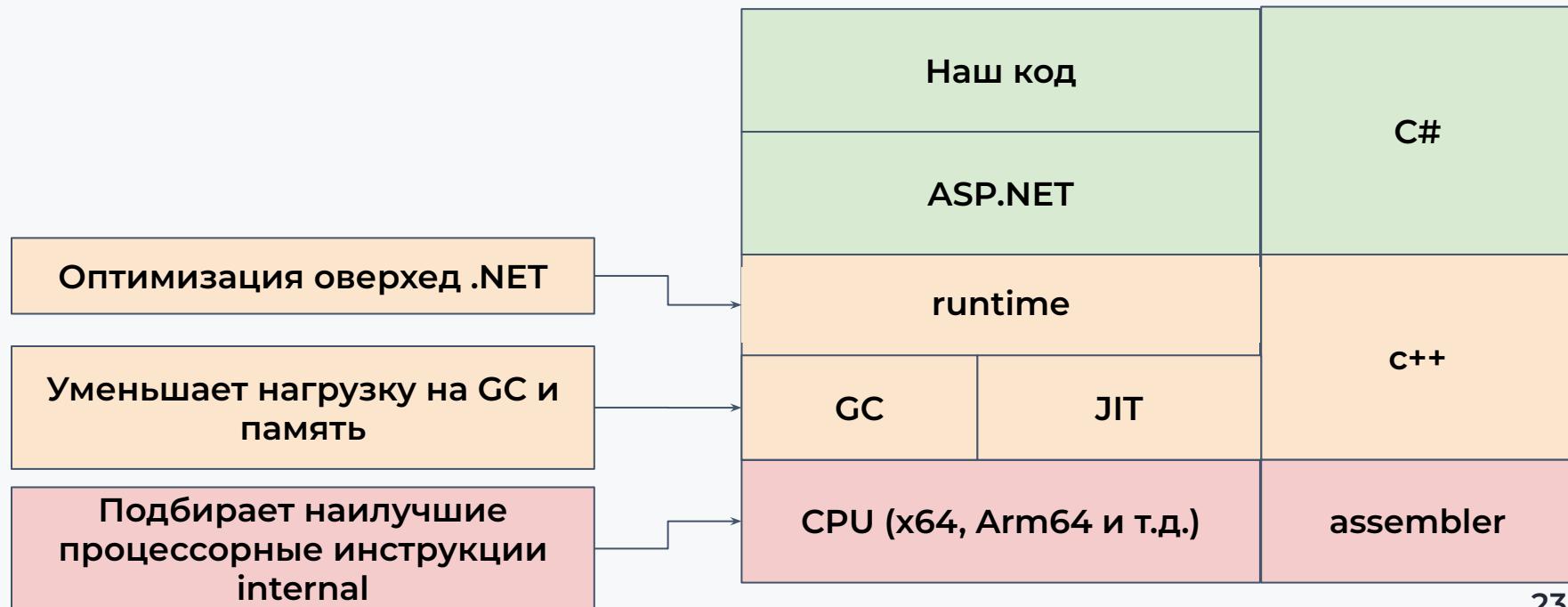
Stack (растет вниз)



heap



Что и на каком слое делает JIT ?



Представление class



Что тут оптимизировать ?

- 1) Method Table - девиртуализация или Guard devirtualization
- 2) Место аллокации (stack/heap) - это будет общее для всех дальше
- 3) Использование полей (не выделять всю память) - это будет общее для всех дальше

Generic (свести стоимость к обычному методу)



```
SyncBlk | MethodTable* (Stack<int>) | _array | _size | ...
```

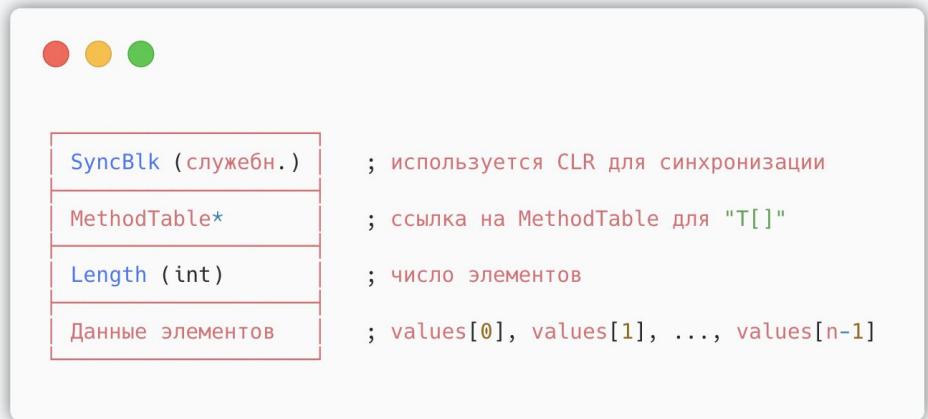
Что тут оптимизировать ?

- 1) Code sharing: Stack<string>, Stack<object>, Stack<MyClass> ходят в один и тот же кусок машинного кода, а различия по типу берутся из контекста (generic dictionary).
- 2) Инлайн
- 3) Стоимость обращения

Массивы

Что тут оптимизировать? ?

- 1) Проверки границ
- 2) Память/GC
- 3) loop
- 4) Девиртуализация



Проверка границ массива



```
[MethodImpl(MethodImplOptions.NoInlining)]
private static ushort[]? Convert(ReadOnlySpan<byte> bytes)
{
    if (bytes.Length != 16)
    {
        return null;
    }

    var result = new ushort[8];
    for (int i = 0; i < result.Length; i++)
    {
        result[i] = (ushort)(bytes[i * 2] * 256 + bytes[i * 2 + 1]);
    }

    return result;
}
```

Развертка циклов



```
public static void Main1()
{
    var array = 1;
    for (int i = 0; i < 5; i++)
    {
        array = array + 1;
    }
}
```



```
G_M000_IG02:                ; offset=0x0004
    mov    eax, 5
    align [0 bytes for IG03]
```

Развертка циклов



```
public static void Main1()
{
    var array = new int[5];
    for (int i = 0; i < array.Length; i++)
    {
        array[i] = 0;
    }
}
```



```
mov     ecx, 5
align  [0 bytes for IG03]

G_M000_IG03:           ; offset=0x0044
    xor     edx, edx
    mov     dword ptr [rax], edx
    add     rax, 4
    dec     ecx
    jne     SHORT G_M000_IG03
```

Array.Fill(new object[5], null);



```
public static void Fill<T>(T[] array, T value)
{
    if (array == null)
    {
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.array);
    }

    if (!typeof(T).IsValueType && array.GetType() != typeof(T[]))
    {
        for (int i = 0; i < array.Length; i++)
        {
            array[i] = value;
        }
    }
    else
    {
        new Span<T>(array).Fill(value);
    }
}
```

Array.Fill(new object[5], null);

```
[Intrinsic]
◆ IL code
public static void Fill<T>(ref T refData, UIntPtr numElements, T value)
{
    if (!RuntimeHelpers.IsReferenceOrContainsReferences<T>() && Vector<byte>.IsTypeOf(T))
    {
        Vector<byte> vector;
        if (sizeof (T) == 1)
            vector = new Vector<byte>(Unsafe.BitCast<T, byte>(value));
        else if (sizeof (T) == 2)
            vector = (Vector<byte>) new Vector<ushort>(Unsafe.BitCast<T, ushort>(value));
        else if (sizeof (T) == 4)
            vector = typeof (T) == typeof (float) ? (Vector<byte>) new Vector<float>(value);
        else if (sizeof (T) == 8)
            vector = typeof (T) == typeof (double) ? (Vector<byte>) new Vector<double>(value);
        else if (sizeof (T) == Vector<byte>.Count)
            vector = Unsafe.BitCast<T, Vector<byte>>(value);
        else if (sizeof (T) == 16)
        {
            switch (Vector<byte>.Count)
            {
                case 2:
                    vector = Unsafe.BitCast<T, Vector<byte>>(value);
                    break;
                case 4:
                    vector = Unsafe.BitCast<T, Vector<float>>(value);
                    break;
                case 8:
                    vector = Unsafe.BitCast<T, Vector<double>>(value);
                    break;
                case 16:
                    vector = Unsafe.BitCast<T, Vector<Vector<byte>>>(value);
                    break;
            }
        }
    }
}
```

Array.Fill(new object[5], null);



```
[MethodImpl(MethodImplOptions.NoInlining)]
public static object[] Main1()
{
    var arr = new string[5];
    Array.Fill(arr, new object());
    return arr;
}
```



G_M000_IG05:

movsxsd	rsi, r14d	; i
mov	rdi, rbx	; array
mov	rdx, r15	; value
call	CORINFO_HELP_ARRADDR_ST	
inc	r14d	
cmp	r12d, r14d	
jg	SHORT G_M000_IG05	

Delegate

Что тут
оптимизировать ?

- 1) Замыкание
- 2) Аллокацию



SyncBlock
MethodTable*
Target object
MethodPtr
InvokeList

- тип делегата (например, `Func<int,int>`)
- `this` или `null` (для статических методов)
- указатель на JIT'ованный метод
- `null` или ссылка на `MulticastDelegate`

Замыкание



```
public partial class Tests
{
    public int Sum(int y)
    {
        Func<int, int> addY = x => x + y;
        return DoubleResult(addY, y);
    }

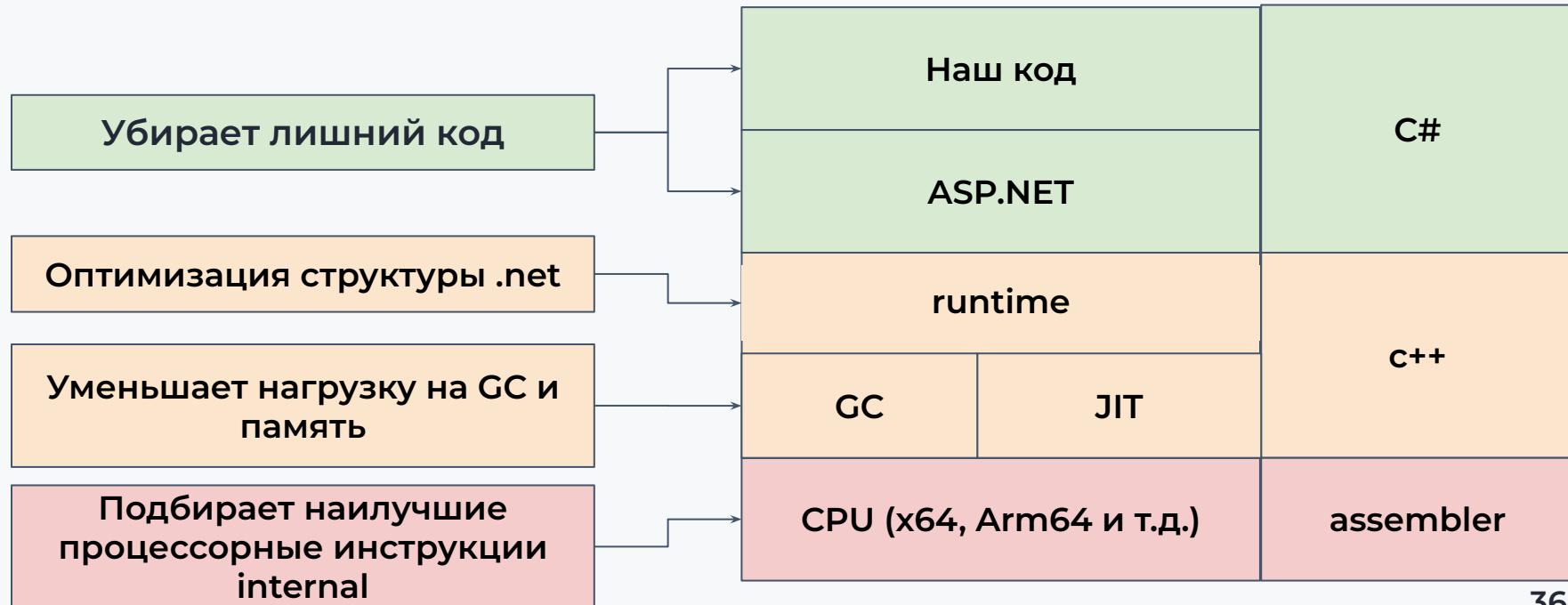
    private int DoubleResult(Func<int, int> func, int arg)
    {
        int result = func(arg);
        return result + result;
    }
}
```

Замыкание

```
.method public hidebysig instance int32 Sum (int32 y) cil managed
{
    .maxstack 3
    .locals init (
        [0] class Tests/'<>c__DisplayClass0_0' 'CS$<>8__locals0',
        [1] class [System.Runtime]System.Func`2<int32, int32> addY
    )

    IL_000 > newobj      instance void Tests/'<>c__DisplayClass0_0'::ctor()
    // тут был кода
    IL_0008: stfld       int32 Tests/'<>c__DisplayClass0_0'::y
    // тут был кода
    IL_00 > newobj      instance void class
    [System.Runtime]System.Func`2<int32,
    native int)           int32>::ctor(object,
    // тут был кода
    IL_0022: call         instance int32 Tests::DoubleResult(class
    [System.Runtime]System.Func`2<int32, int32>, int32)
    IL_0027: ret
}
```

Что и на каком слое делает JIT ?



Цикловой инвариант



```
for (int i = 0; i < n; i++)
{
    int val = a * b; // Не зависит от i, можно вынести
    arr[i] = val + i;
}
```



```
int val = a * b; // Вынесено за цикл
for (int i = 0; i < n; i++)
{
    arr[i] = val + i;
}
```

Дублирующие вычисления



```
[MethodImpl(MethodImplOptions.NoInlining)]
private static ushort[]? Convert(ReadOnlySpan<byte> bytes)
{
    if (bytes.Length != 16)
    {
        return null;
    }

    var result = new ushort[8];
    for (int i = 0; i < result.Length; i++)
    {
        result[i] = (ushort)(bytes[i * 2] * 256 + bytes[i * 2 + 1]);
    }

    return result;
}
```

Branch if



```
void Test(int x)
{
    if (x > 100)
        if (x > 10) // always true
            Console.WriteLine();
}
```

Убирает лишний код

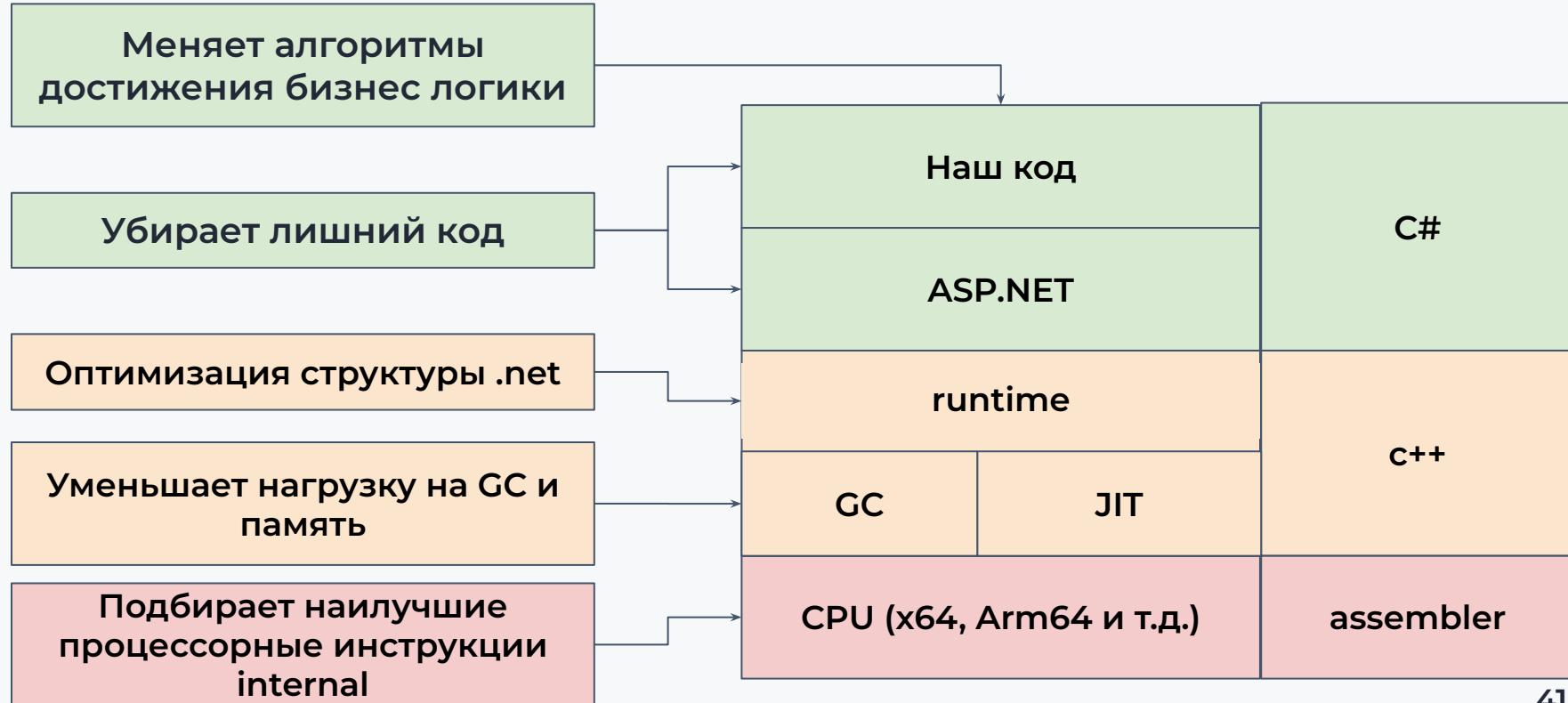


```
public void Test(int x)
{
    if (x > 100) { Helper(x); }
}

private void Helper(int x)
{
    if (x > 10) { Console.WriteLine("Hello!"); }
}
```

Это хороший пример про то, что мы можем не изгибать бизнес логику под оптимизации !

Что и на каком слое делает JIT ?



Циклы с восходящим счётом в циклы с нисходящим счётом



```
public int UpwardCounting()
{
    int count = 0;
    for (int i = 0; i < 100; i++)
    {
        count++;
    }
    return count;
}
```

```
M00_L00:
    inc    eax          ; count++
    inc    ecx          ; i++
    cmp    ecx, 100     ; i < 100 ?
    jl     M00_L00
```



```
public int DownwardCounting()
{
    int count = 0;
    for (int i = 99; i >= 0; i--)
    {
        count++;
    }
    return count;
}
```

```
M00_L00:
    inc    eax          ; count++
    dec    ecx          ; i--
    jns    M00_L00      ; пока i не отрицателен (i >= 0)
```

Инициализация списка



```
static void Test(char[] arr)
{
    if (arr.Length >= 4)
    {
        arr[0] = 'T';
        arr[1] = 'R';
        arr[2] = 'U';
        arr[3] = 'E';
    }
}
```

```
mov rax, 0x45005500520054      ; "TRUE" упакован в 64-битное значение
mov qword ptr [rcx+0x10], rax ; одним присвоением
```

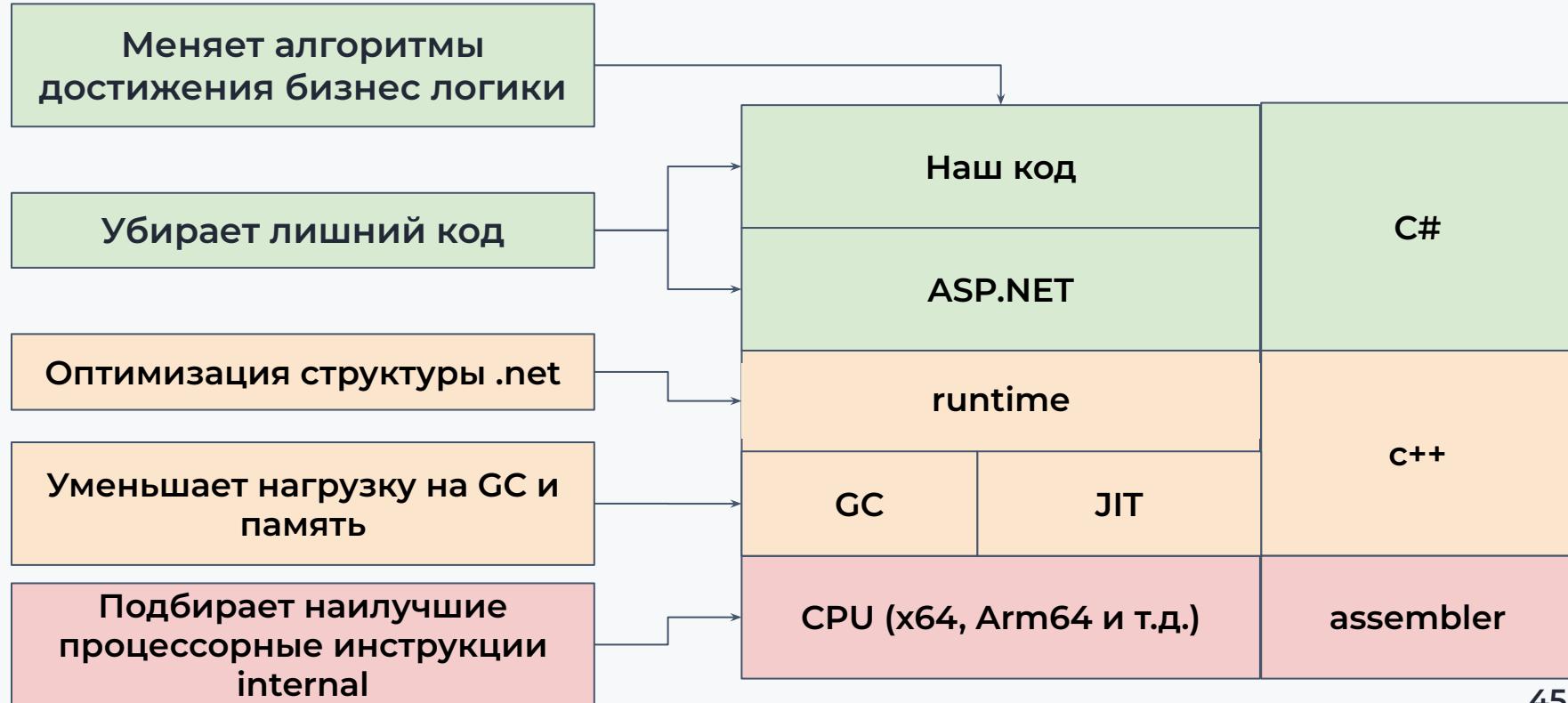
Hot/Cold Splitting



```
; С PG0: горячий путь и холодный (виртуальный) путь разделены
    cmp      [rcx], r11          ; проверка типа объекта _source
    jne      cold_path          ; если тип не совпал – перейти в "холодный" код
    ... (inline-код оптимизированного MoveNext) ...
    jmp      end
cold_path:
    mov      r11, [virtual stub]
    call     qword ptr [r11]       ; холодный виртуальный вызов
end:
    ret
```

- 1) Кеш инструкций
- 2) Оптимизация отдельных кусков

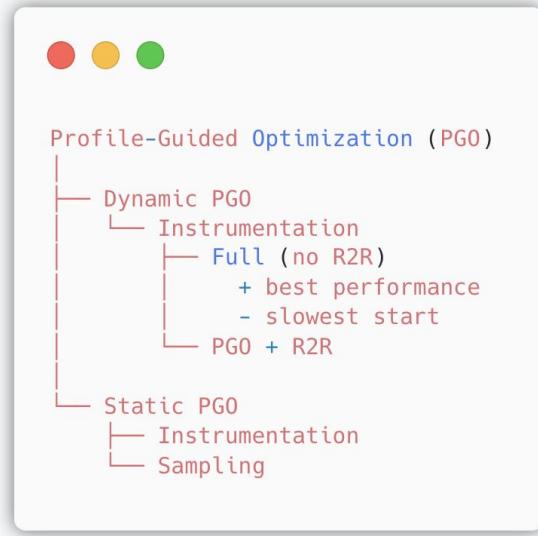
Что и на каком слое делает JIT ?



Как это достигается ?

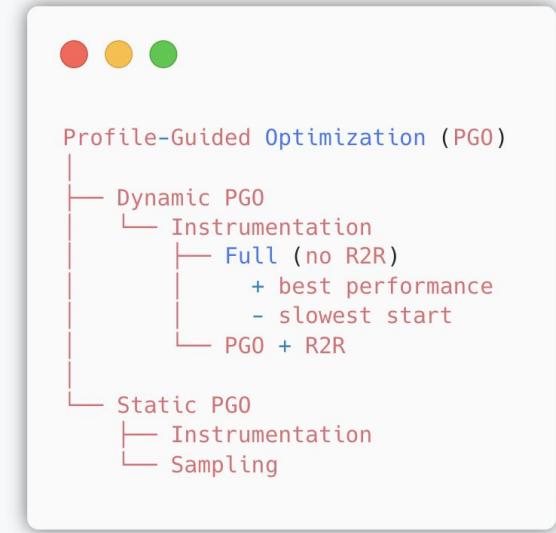
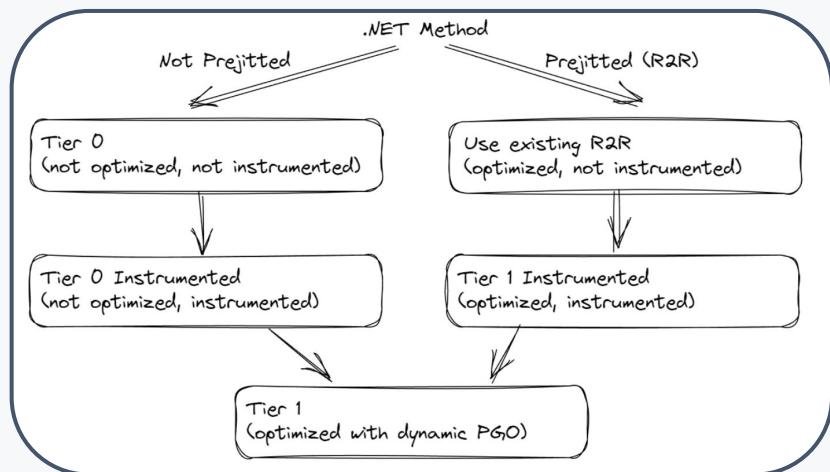
Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO



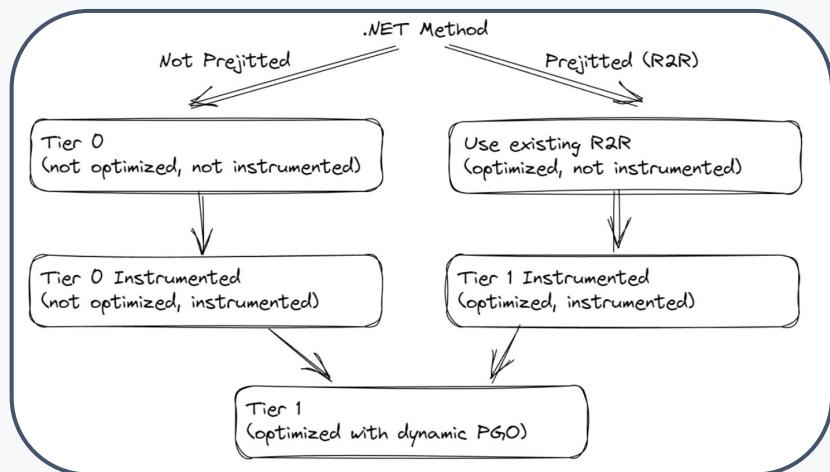
Состав оптимизаций

1) Оптимизации для оптимизаций a) PGO + Tier



Состав оптимизаций

1) Оптимизации для оптимизаций a) PGO + Tier + OSR



Profile-Guided Optimization (PGO)

- Dynamic PGO
 - Instrumentation
 - Full (no R2R)
 - + best performance
 - slowest start
 - PGO + R2R

Static PGO

- Instrumentation
- Sampling

```
class Program
{
    static void Main()
    {
        var sw = new System.Diagnostics.Stopwatch();
        while (true)
        {
            sw.Restart();
            for (int trial = 0; trial < 10_000; trial++)
            {
                int count = 0;
                for (int i = 0; i < char.MaxValue; i++)
                    if (IsAsciiDigit((char)i))
                        count++;
            }
            sw.Stop();
            Console.WriteLine(sw.Elapsed);
        }
    }

    static bool IsAsciiDigit(char c) => (uint)(c - '0') <= 9;
}
```

Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO + Tier + OSR
 - b) Inline



Состав оптимизаций

1) Оптимизации для оптимизаций

- a) PGO + Tier + OSR
- b) Inline
- c) escape analize (опа что-то новенькое)

Метод

```
Obj a = new Obj();
Use(a);
return a;    <— утечка
```

Obj b = new Obj();
int x = b.Value;
return x; <— локален

heap/вызывавший код

(b исчезает, поля заменены скалярами)

The screenshot shows a window titled 'Метод' (Method) containing Java code. The code creates an object 'a' and returns it. A red arrow points from the line 'return a;' to the text 'утечка' (leak). Another red arrow points from the line 'return x;' to the text 'локален' (local). To the right of the code, the text 'heap/вызывавший код' (heap/calling code) is shown with a red arrow pointing to it. Below this, the text '(b исчезает, поля заменены скалярами)' (b disappears, fields are replaced by scalars) is written in parentheses.

Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO + Tier + OSR
 - b) Inline
 - c) escape analize (опа что-то новенькое)
- 2) Самодостаточные оптимизации
 - a) Проверка границ

```
int GetItemAtIndex(int[] a, int index) {  
    return a[index];  
}
```

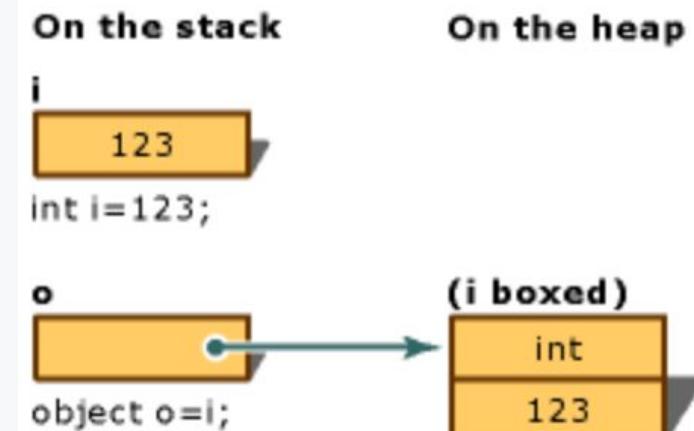
Array Bounds Check



```
L0000: mov eax, [esp+4]  
L0004: cmp eax, [edx+4]  
L0007: jae short L0010  
L0009: mov eax, [edx+eax*4+8]  
L000d: ret 4  
L0010: call 0x727c1070  
L0015: int3
```

Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO + Tier + OSR
 - b) Inline
 - c) escape analize (опа что-то новенькое)
- 2) Самодостаточные оптимизации
 - a) Проверка границ
 - b) Unbox



Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO + Tier + OSR
 - b) Inline
 - c) escape analize (опа что-то новенькое)
- 2) Самодостаточные оптимизации
 - a) Проверка границ
 - b) Unbox
 - c) & т.с (много докладов по отдельным оптимизациям)
- 3) связность оптимизаций
 - a) Inline -> bound check -> Loop unrolling -> SIMD
 - b) Escape Analize -> Stackalloc -> Struct promotion -> const Folding

Состав оптимизаций

- 1) Оптимизации для оптимизаций
 - a) PGO + Tier + OSR
 - b) Inline
 - c) escape analize (опа что-то новенькое)
- 2) Самодостаточные оптимизации
 - a) Проверка границ
 - b) Unbox
 - c) & т.с (много докладов по мелким оптимизациям)
- 3) связность оптимизаций
 - a) Inline -> bound check -> Loop unrolling -> SIMD
 - b) Escape Analize -> Stackalloc -> Struct promotion -> const Folding
- 4) Intrinsic - методы, обрабатываемые специальным образом...
 - a) ArgumentNullException.ThrowIfNull
 - b) Activator.CreateInstance

Avoid boxing ArgumentNullException.ThrowIfNull



```
static void Test()
{
    ThrowIfNull(i);
}
```

```
// тут произошел box в ArgumentNullException.ThrowIfNull(value)
static void ThrowIfNull<T>(T value) => ArgumentException.ThrowIfNull(value);
```

Не generic, потому что АОТ и потому что кому это надо

JIT: Avoid boxing ArgumentNullException.ThrowIfNull



```
src/coreclr/jit/importercalls.cpp
.
.
else if (strcmp(className, "ArgumentNullException") == 0)
{
    if (strcmp(methodName, "ThrowIfNull") == 0)
    {
        result = NI_System_ArgumentNullException_ThrowIfNull;
    }
}
.
```

JIT: Avoid boxing ArgumentNullException.ThrowIfNull



src/coreclr/jit/importercalls.cpp

```
if (value->OperIs(GT_BOX))
{
    // Now we need to spill the addr and argName arguments in the correct order
    // to preserve possible side effects.
    unsigned boxedValTmp      = lvaGrabTemp(true DEBUGARG("boxedVal spilled"));
    unsigned boxedArgNameTmp = lvaGrabTemp(true DEBUGARG("boxedArg spilled"));
    impStoreToTemp(boxedValTmp, value, CHECK_SPILL_ALL);
    impStoreToTemp(boxedArgNameTmp, valueName, CHECK_SPILL_ALL);
    gtTryRemoveBoxUpstreamEffects(value, BR_REMOVE_AND_NARROW);
    return gtNewNothingNode();
}
```

А много еще таких intrinsic ?

```
case 'A':  
{  
    if (strcmp(className, "Activator") == 0)  
    {  
        if (strcmp(methodName, "AllocatorOf") == 0)  
        {  
            result = NI_System_Activator_AllocatorOf;  
        }  
        else if (strcmp(methodName, "DefaultConstructorOf") == 0)  
        {  
            result = NI_System_Activator_DefaultConstructorOf;  
        }  
    }  
}
```

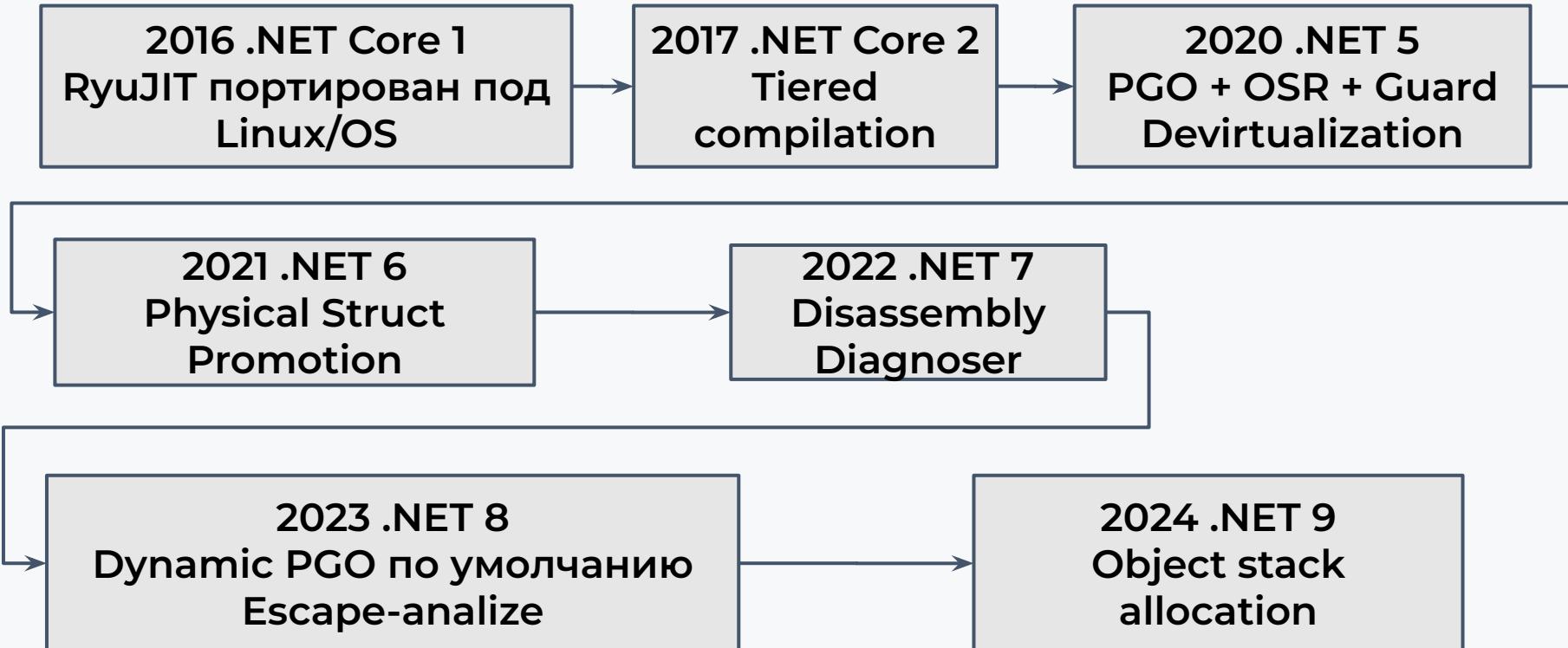
```
else if (strcmp(className, "Array") == 0)  
{  
    if (strcmp(methodName, "Clone") == 0)  
    {  
        result = NI_System_Array_Clone;  
    }  
    else if (strcmp(methodName, "GetLength") == 0)  
    {  
        result = NI_System_Array_GetLength;  
    }  
    else if (strcmp(methodName, "GetLowerBound") == 0)  
    {  
        result = NI_System_Array_GetLowerBound;  
    }  
    else if (strcmp(methodName, "GetUpperBound") == 0)  
    {  
    }
```



Развитие JIT в последних версиях

.NET

Road map JIT



Улучшения алгоритмов JIT

Считаем кол-во вызовов чего то...

Что считаем	Инкрементов в секунду
Вызовы строк кода для test cover	1-10
Http request by service	100-300
Вызовы методов	...

РГО и подсчет вызовов

- Быстро
- Не потокобезопасно
(race condition) -> Потеря
обновлений при
многопоточном доступе



counter++;

PGO и подсчет вызовов



```
Interlocked.Increment(ref counter);
```

- Потокобезопасно
- Точная семантика "прибавь 1"
- Медленно при высокой конкуренции, узкое место

РГО и подсчет вызовов

- Потокобезопасно
- Почти такая же производительность, как у `++`
- Приблизенно верно, отклонение < 1%

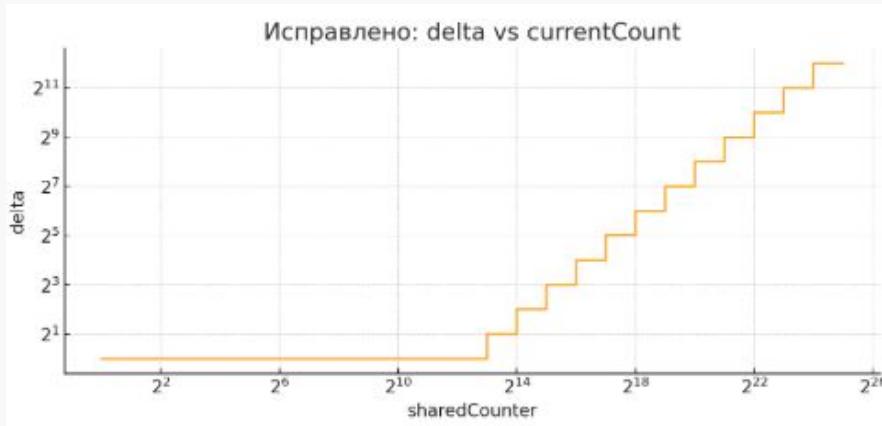


```
static void Count(ref uint sharedCounter)
{
    uint currentCount = sharedCounter, delta = 1;
    if (currentCount > 0)
    {
        int logCount = 31 - (int)uint.LeadingZeroCount(currentCount);
        if (logCount >= 13)
        {
            delta = lu << (logCount - 12);
            uint random = (uint)Random.Shared.NextInt64(0, uint.MaxValue + 1L);
            if ((random & (delta - 1)) != 0) return;
        }
    }
    Interlocked.Add(ref sharedCounter, delta);
}
```

delta vs sharedCounter

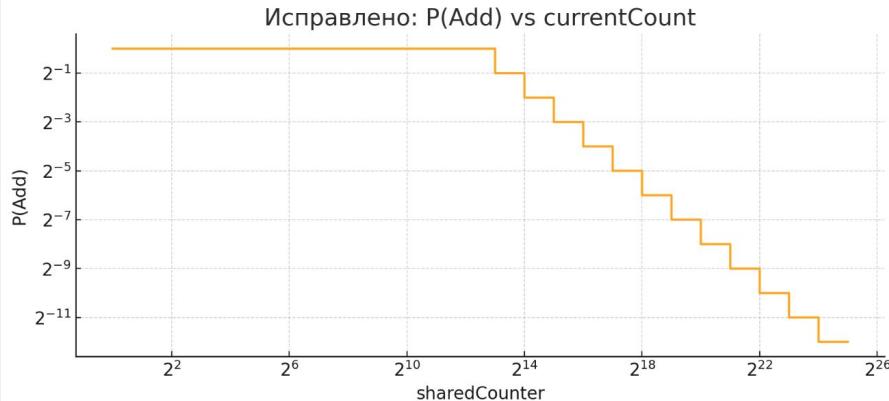
до $8192 (2^{13})$ — шаг
ровно 1, дальше
удваивается каждые
 $2 \times$

```
static void Count(ref uint sharedCounter)
{
    uint currentCount = sharedCounter, delta = 1;
    if (currentCount > 0)
    {
        int logCount = 31 - (int)uint.LeadingZeroCount(currentCount);
        if (logCount >= 13)
        {
            delta = 1u << (logCount - 12);
            uint random = (uint)Random.Shared.NextInt64(0, uint.MaxValue + 1L);
            if ((random & (delta - 1)) != 0) return;
        }
    }
    Interlocked.Add(ref sharedCounter, delta);
}
```



P(Add)
vs sharedCounter
вероятность вызова
Interlocked.Add
падает как $1/\delta$

```
static void Count(ref uint sharedCounter)
{
    uint currentCount = sharedCounter, delta = 1;
    if (currentCount > 0)
    {
        int logCount = 31 - (int)uint.LeadingZeroCount(currentCount);
        if (logCount >= 13)
        {
            delta = 1u << (logCount - 12);
            uint random = (uint)Random.Shared.NextInt64(0, uint.MaxValue + 1L);
            if ((random & (delta - 1)) != 0) return;
        }
    }
    Interlocked.Add(ref sharedCounter, delta);
}
```



PGO и подсчет вызовов



```
while (true)
{
    Run("Interlock", _ => {
        for (int i = 0; i < ItersPerThread; i++) Interlocked.Increment(ref counter);
    });
    Run("Racy      ", _ => {
        for (int i = 0; i < ItersPerThread; i++) counter++;
    });
    Run("Scalable ", _ => {
        for (int i = 0; i < ItersPerThread; i++) Count(ref counter);
    });
}
```

PGO и подсчет вызовов



Попытка 1

```
Interlock => Expected: 1 200 000 000, Actual: 1 200 000 000, Elapsed: 108030,9295ms
Racy      => Expected: 1 200 000 000, Actual:    148 620 908, Elapsed: 283,6168ms
Scalable  => Expected: 1 200 000 000, Actual: 1 200 886 164, Elapsed: 557,9463ms
```

Попытка 2

```
Interlock => Expected: 1 200 000 000, Actual: 1 200 000 000, Elapsed: 88186,767ms
Racy      => Expected: 1 200 000 000, Actual:    179 682 457, Elapsed: 321,0945ms
Scalable  => Expected: 1 200 000 000, Actual: 1 193 279 524, Elapsed: 520,8084ms
```

Попытка 3

```
Interlock => Expected: 1 200 000 000, Actual: 1 200 000 000, Elapsed: 94467,3402ms
Racy      => Expected: 1 200 000 000, Actual:    147 883 941, Elapsed: 284,1835ms
Scalable  => Expected: 1 200 000 000, Actual: 1 185 415 649, Elapsed: 427,7372ms
```

Упрощение написания кода

GDV делегатов .net 8



```
public class Tests
{
    public int Sum() => Sum(i => i + 1);

    private static int Sum(Func<int, int> func)
    {
        int sum = 0;
        for (int i = 0; i < 10_000; i++)
        {
            sum += func(i);
        }

        return sum;
    }
}
```



```
private static int Sum(Func<int, int> func)
{
    int sum = 0;
    if (func.Method == KnownLambda)
    {
        for (int i = 0; i < 10_000; i++)
        {
            sum += i + 1;
        }
    }
    else
    {
        for (int i = 0; i < 10_000; i++)
        {
            sum += func(i);
        }
    }
    return sum;
}
```

.net 9 support inline Shared-generic in Shared-generic



```
[MethodImpl(MethodImplOptions.NoInlining)]
bool Test<T>() => Callee<T>();

bool Callee<T>() => typeof(T).IsValueType;
```

Улучшение проверки границ массива .net 9

Зачем ???



```
✓ ✨ 38 src/libraries/System.Private.CoreLib/src/System/Collections/Generic/Dictionary.cs □
574 565         else
575 566     {
576 567         Debug.Assert(comparer != null);
577 568     -     while (true)
578 569     {
579 570         // Should be a while loop https://github.com/dotnet/runtime/issues/9422
580 571         // Test uint in if rather than loop condition to drop range check for following array access
581 572         if ((uint)i >= (uint)entries.Length)
582 573         {
583 574             break;
584 575         }
585 576
586 577         if (entries[i].hashCode == hashCode && comparer.Equals(entries[i].key, key))
587 578         {
588 579             if (behavior == InsertionBehavior.OverwriteExisting)
589 580                 @@ -690,15 +674,8 @@ internal static class CollectionsMarshalHelper
590 584                     comparer == null)
591 585                 {
592 586                     // ValueType: Devirtualize with EqualityComparer< TKey >.Default intrinsic
593 587                 -             while (true)
594 588                 +             while ((uint)i < (uint)entries.Length)
595 589                 {
596 590                     // Should be a while loop https://github.com/dotnet/runtime/issues/9422
597 591                     // Test uint in if rather than loop condition to drop range check for following array access
598 592                     if ((uint)i >= (uint)entries.Length)
599 593                     {
600 594                         break;
601 595                     }
602 596
603 597                     if (entries[i].hashCode == hashCode && EqualityComparer< TKey >.Default.Equals(entries[i].key, key))
604 598                     {
605 599                         exists = true;
606 600
607 601
608 602
609 603
610 604
611 605
612 606
613 607
614 608
615 609
616 610
617 611
618 612
619 613
620 614
621 615
622 616
623 617
624 618
625 619
626 620
627 621
628 622
629 623
630 624
631 625
632 626
633 627
634 628
635 629
636 630
637 631
638 632
639 633
640 634
641 635
642 636
643 637
644 638
645 639
646 640
647 641
648 642
649 643
650 644
651 645
652 646
653 647
654 648
655 649
656 650
657 651
658 652
659 653
660 654
661 655
662 656
663 657
664 658
665 659
666 660
667 661
668 662
669 663
670 664
671 665
672 666
673 667
674 668
675 669
676 670
677 671
678 672
679 673
680 674
681 675
```

Сравним подходы

- 1) Какой код производительнее ?
- 2) Какой код понятнее?



```
while (true)
{
    if ((uint)i >= src.Length) break; // охранная проверка
    sum += src[i++]; // доступ + пост-инкремент
}
```



```
for ( ; (uint)i < src.Length; i++)
{
    sum += src[i];
}
```

Почему 1 чуть быстрее ?



while

```
cmp i, len      ; guard
jae exit
mov t, [base+i*2]
add sum, t
inc i
jmp top
```



for

```
cmp i, len      ; guard на входе
jae exit
mov t, [base+i*2]
add sum, t
inc i
cmp i, len      ; условие for (второе сравнение)
jb top
```

.net 10 devirtualization array

```
private ReadOnlyCollection<int> _list = new ReadOnlyCollection<int>(
    Enumerable.Range(1, 1000).ToArray()
);
```

```
public int SumEnumerable()
{
    int sum = 0;
    foreach (var item in _list)
    {
        sum += item;
    }
    return sum;
}
```

```
public int SumForLoop()
{
    ReadOnlyCollection<int> list = _list;
    int sum = 0;
    int count = list.Count;
    for (int i = 0; i < count; i++)
    {
        sum += _list[i];
    }
    return sum;
}
```

.net 10 devirtualization array

```
private ReadOnlyCollection<int> _list = new ReadOnlyCollection<int>(
    Enumerable.Range(1, 1000).ToArray()
);
```

```
public int SumEnumerable()
{
    int sum = 0;
    foreach (var item in _list)
    {
        sum += item;
    }
    return sum;
}
```

```
public int SumForLoop()
{
    ReadOnlyCollection<int> list = _list;
    int sum = 0;
    int count = list.Count;
    for (int i = 0; i < count; i++)
    {
        sum += _list[i];
    }
    return sum;
}
```

949nc

1932nc

.net 10 devirtualization array

`foreach` → ОДИН виртуальный вызов + дальше прямые вызовы

- виртуальным был только `_list.GetEnumerator()`
- сам enumerator типа `SZArrayHelper` → JIT легко inline
- `MoveNext` и `Current` становились оптимизированными вызовами

`for` → 1000 виртуальных вызовов

Т.к. `_list[i]` = виртуальный вызов `IList<T>.Item.get`

.net 10 Удаление границ для span

```
public partial class Tests
{
    private int[] _arr = new int[16];
    private int _count = 8;

    [Benchmark]
    public void WithSpan()
    {
        Span<int> span = _arr;
        int count = _count;

        for (int i = 0; i < count; i++)
        {
            span[i] = i;
        }
    }
}
```

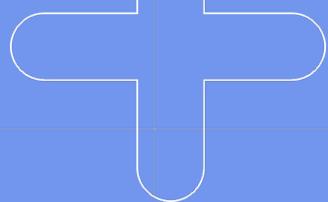
```
[Benchmark]
public void WithArray()
{
    int[] arr = _arr;
    int count = _count;

    for (int i = 0; i < count; i++)
    {
        arr[i] = i;
    }
}
```

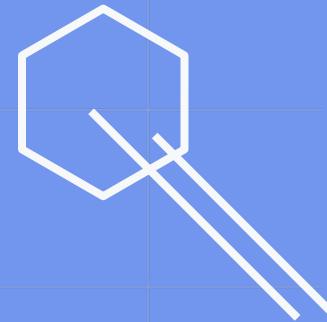
.net 10 Удаление границ для span

```
// FAST PATH: гарантируем, что дальше i < count => i всегда < arr.Length
if ((uint)count <= (uint)arr.Length)
{
    // здесь JIT может вообще выкинуть bounds-checks
    for (int i = 0; i < count; i++)
    {
        arr[i] = i;
    }
}
else
{
    // SLOW PATH: нам нужно "страховаться" на каждой итерации
    for (int i = 0; i < count; i++)
    {
        // вручную эмулируем bounds check
        if ((uint)i >= (uint)arr.Length)
        {
            ThrowRangeCheckFail(); // аналог CORINFO_HELP_RNGCHKFAIL
        }

        arr[i] = i;
    }
}
```



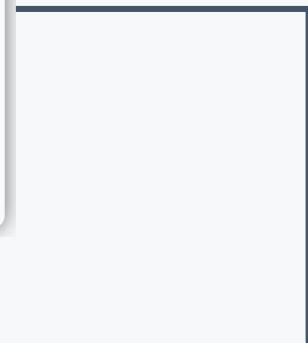
Упрощение работы с памятью
благодаря JIT



Пример



```
class Acc { public int A, B, C, D; }
int Use(Acc s) => s.A + s.B;
int Demo()
{
    var tmp = new Acc { A = 10, B = 20 };
    return Use(tmp);
}
```



```
int Demo()
{
    return 30;
}
```

Пример



```
class Acc { public int A, B, C, D; }
int Use(Acc s) => s.A + s.B;
int Demo()
{
    var tmp = new Acc { A = 10, B = 20 };
    return Use(tmp);
}
```

stack allocation



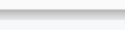
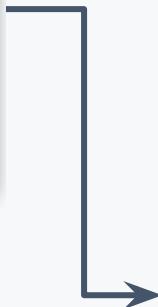
```
struct Acc { public int A, B, C, D; }
int Use(Acc s) => s.A + s.B;
int Demo()
{
    var tmp = new Acc { A = 10, B = 20, C = 3, D = 4 };
    return Use(tmp);
}
```

Пример



```
struct Acc { public int A, B, C, D; }
int Use(Acc s) => s.A + s.B;
int Demo()
{
    var tmp = new Acc { A = 10, B = 20, C = 3, D = 4 };
    return Use(tmp);
}
```

Нужен inline



```
struct Acc { public int A, B, C, D; }
int Demo()
{
    var tmp = new Acc { A = 10, B = 20, C = 3, D = 4 };
    return s.A + s.B;
}
```

Пример



```
struct Acc { public int A, B, C, D; }
int Demo()
{
    var tmp = new Acc { A = 10, B = 20, C = 3, D = 4 };
    return s.A + s.B;
}
```



Иначе структура остается единой областью памяти с доступом по смещению `mov eax, [rsp+offset]` и я не могу ее разложить в SSA и соответственно удалить ненужное и свернуть нужное

Struct Promotion



```
int Demo()
{
    int A = 10, B = 20, C = 3, D = 4;
    return s.A + s.B;
}
```

Пример



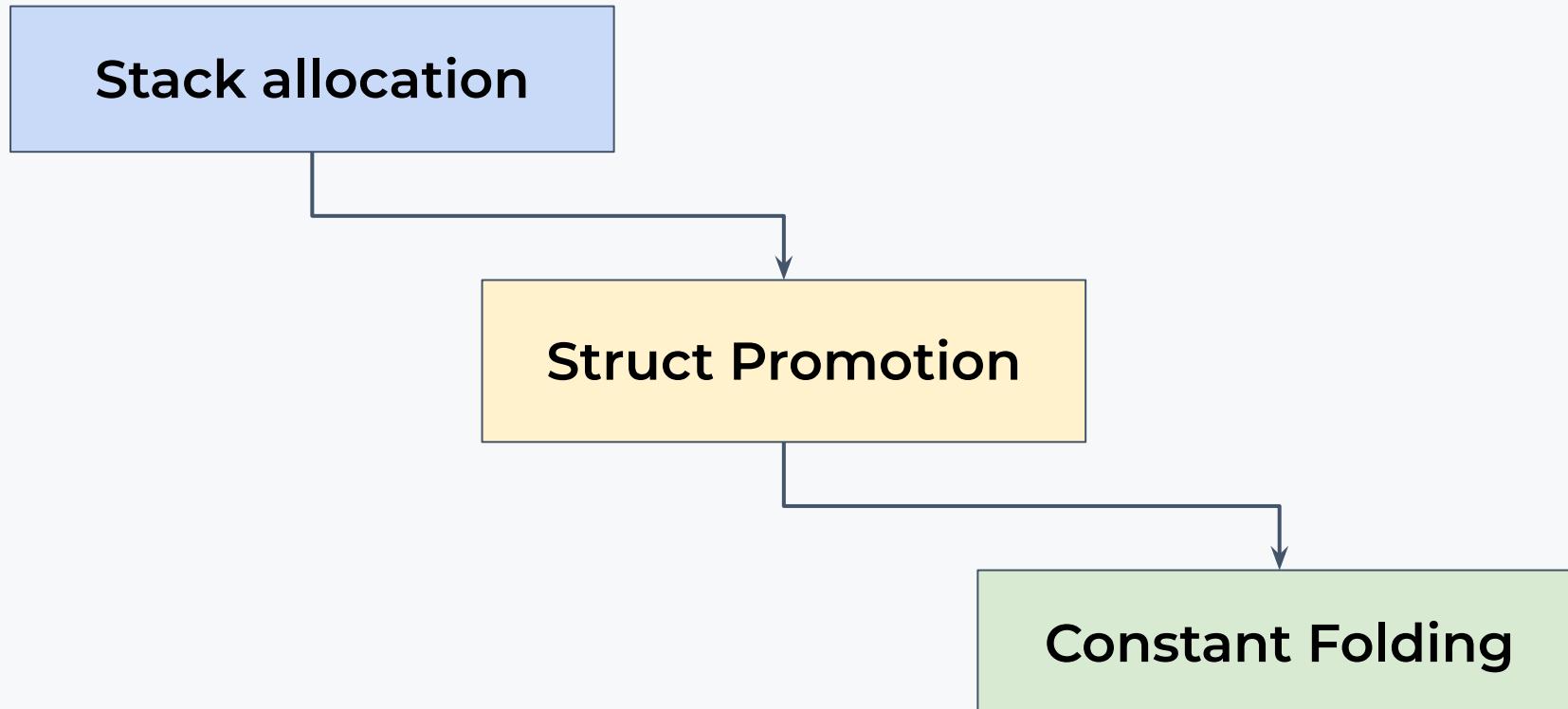
```
int Demo()
{
    int A = 10, B = 20, C = 3, D = 4;
    return s.A + s.B;
}
```

Нужен Constant
Folding



```
int Demo()
{
    return 30;
}
```

Спойлер



Stack allocation

Включена ли фича?
JitObjectStackAllocation.

Проверка
ограничения

Escape
Analysis

Проверка конструктора
и инициализации

Struct Promotion

Ограничения



```
// 3. Класс с финализатором запрещён
class WithFin
{
    ~WithFin() { /*...*/ }
}

void FinDemo()
{
    // всегда heap (finalizer)
    _ = new WithFin();
}
```



```
// 4. Аллокация в цикле – анализ отключён
void LoopAlloc()
{
    for (int i = 0; i < 3; i++)
        _ = new C();           // heap .NET 9/10
}
```

Ограничения



```
// 10 async
static async Task<int> AsyncEscape()
{
    var data = new SmallClass(); // ✗ уйдёт в кучу
    await Task.Delay(10);      // state-машина переживёт стек
    return data.Value;
}

class SmallClass
{
    public int Value = 7;
}
```



```
static void TryCatchEscape()
{
    var res = new SmallClass(); // ✗ heap
    try
    {
        MightThrow(res);
    }
    catch
    {
        // re-throw сохранит res до обработки
        throw;
    }
}
```

Проверка размера



```
[StructLayout(LayoutKind.Explicit)]
class MyExplicit
{
    [FieldOffset(0)]
    public int A;

    [FieldOffset(0)]
    public int B;
}
```



```
bool CanTypeBeStackAllocated(TypeDesc t)
{
    if (getHeapClassSize(t) > s_StackAllocMaxSize) // ~512 B
        return false;

    if (TARGET_X86 && t.RequiresAlign8)
        return false;

    if (t.HasExplicitLayout || t.HasBlockLayout)
        return false;

    if (t.HasFinalizer)
        return false;

    if (t.IsValueType && t.ContainsGCPointers)
        return false;

    if (t.IsWeirdSpecialRuntimeType) // COM, collectible и пр.
        return false;

    // тип в принципе годится для object stack allocation
    return true;
}
```

.NET 9 Когда JIT видит, что объект НЕ кандидат на стек ?



```
// 5. Маленький массив значимых типов (только .NET 10)
void SmallValArray()
{
    // stack в 10-ке, heap в 9-ке
    int[] arr = { 1, 2, 3 };
}
```

и т. д. например **out** или **ref**

Проверка конструктора и инициализации

newobj всегда вызывает .ctor.

Если .ctor слишком сложный → JIT не сможет развернуть его в IR → объект остается в куче.



// Хорошо

```
class MyObj { public int A; public MyObj(int a) { A = a; } }
```

// Плохо

```
class Bad {
    public static List<Bad> Cache = new();
    public Bad() { Cache.Add(this); }
}
```

Проверка конструктора и инициализации



```
void Execute()
{
    var classDa = new Example();
    var classDa2 = new Example2();
    classDa.ExampleInst = classDa2;
}
```



```
mov    rcx, 0x7FFCD6F219B0
call   CORINFO_HELP_NEWSFAST
mov    qword ptr [rbp-0x10], rax
mov    rcx, qword ptr [rbp-0x10]
call   [Example::ctor():this]
```

Tier0

Escape analize

Анализ, который проверяет — «убегает ли объект/структура за пределы текущего метода?».

Если нет → JIT может безопасно разместить объект на стеке, а не в куче



- new Obj
 - |
 - ↳ ◇ local

- |
- └ only local uses → NoEscape
- └ return / pass upward → ArgEscape
- └ store field/ref/out → GlobalEscape

.NET 9 JIT видит, что объект убегает со стека



```
// 1. Объект «убегает» через return – всегда куча
class C { public int X; }
[MethodImpl(MethodImplOptions.NoInlining)]
C MakeEscaping()
{
    return new C { X = 1 }; // heap в .NET 9/10
}
```



```
// 2. Сохранение в статическое поле – куча
static C _root;
void StoreToStatic()
{
    _root = new C();
}
```

.NET 9 JIT видит, что объект убегает со стека



```
// 9. Делегат-замыкание: стек только в .NET 10
int DelegateDemo()
{
    int captured = 5;
    // stack в 10, heap в 9
    Func<int, int> f = x => x + captured;
    return f(1);
}
```



```
ref int GetRef() {
    var p = new Point();
    return ref p.X;
}
```

.NET 9 JIT видит, что объект убегает со стека



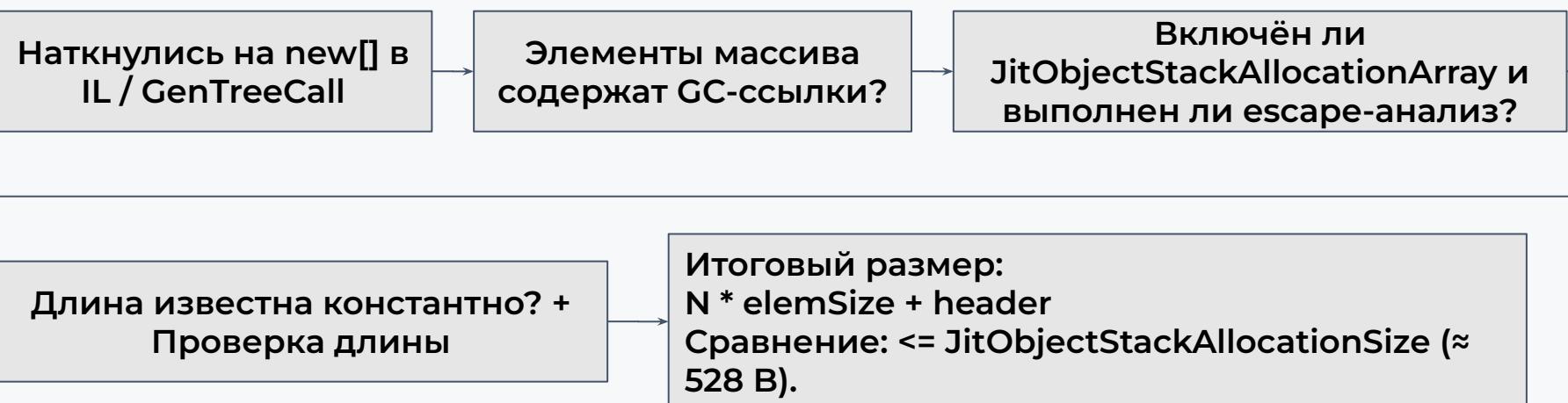
```
class Holder { public MyClass Ref; }
void F(Holder h) {
    var o = new MyClass();
    h.Ref = o;
}
```

inline опять помог ->



```
/* может сохранить */
void Unknown(MyClass x) { }
void F() {
    var o = new MyClass();
    // ESCAPE (консервативно)
    Unknown(o);
}
```

.NET 10 Extend escape analysis to account for arrays with non-gc ref elements #104906



.NET 10 initial support for stack allocating arrays of GC type #112250

Наткнулись на new[] в
IL / GenTreeCall

Включён ли
JitObjectStackAllocationArray и
выполнен ли escape-анализ?

Длина известна константно? +
Проверка длины

Итоговый размер:
 $N * \text{elemSize} + \text{header}$
Сравнение: $\leq \text{JitObjectStackAllocationSize} (\approx 528 \text{ B})$.

Пример разложения массива .net 9



```
public partial class Tests
{
    public void Test()
    {
        Process(new int[] { 1, 2, 3 });

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        static void Process(string[] inputs)
        {
            foreach (int input in inputs)
            {
                Use(input);
            }

            [MethodImpl(MethodImplOptions.NoInlining)]
            static void Use(int input) { }
        }
    }
}
```



```
G_M000_IG02: ; offset=0x0001
    mov     rdi, 0x7F8521C71D90
    mov     esi, 3
    call    CORINFO_HELP_NEWARR_1_OBJ
    mov     rdi, 0x7F859BFE3718
    mov     qword ptr [rax+0x10], rdi
    mov     rdi, 0x7F859BFE3420
    mov     qword ptr [rax+0x18], rdi
    mov     rdi, 0x7F859BFE3730
    mov     qword ptr [rax+0x20], rdi
    mov     rdi, rax
```

Пример разложения массива .net 10

```
public partial class Tests
{
    public void Test()
    {
        Process(new int[] { 1, 2, 3 });

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        static void Process(string[] inputs)
        {
            foreach (int input in inputs)
            {
                Use(input);
            }

            [MethodImpl(MethodImplOptions.NoInlining)]
            static void Use(int input) { }
        }
    }
}
```

```
G_M000_IG02: ; offset=0x0021
    mov    rdi, 0x7FC1B5EBD088
    mov    qword ptr [rbp-0x38], rdi

    lea    rdi, [rbp-0x38]
    mov    dword ptr [rdi+0x08], 3

    lea    rbx, [rbp-0x38]
    mov    rdi, 0x7FC230107758
    mov    qword ptr [rbx+0x10], rdi
    mov    rdi, 0x7FC230107460
    mov    qword ptr [rbx+0x18], rdi
    mov    rdi, 0x7FC230107770
    mov    qword ptr [rbx+0x20], rdi
```

Делегаты



```
.method public hidebysig instance int32 Sum (int32 y) cil managed
{
    .maxstack 3
    .locals init (
        [0] class Tests/<>c__DisplayClass0_0' 'CS<>8__locals0',
        [1] class [System.Runtime]System.Func`2<int32, int32> addY
    )

    IL_0000: newobj     instance void Tests/<>c__DisplayClass0_0'::ctor()
    // тут был кода
    IL_0008: stfld      int32 Tests/<>c__DisplayClass0_0'::y
    // тут был кода
    IL_0014: newobj     instance void class
    [System.Runtime]System.Func`2<int32,           int32>::ctor(object,
    native int)
    // тут был кода
    IL_0022: call        instance int32 Tests::DoubleResult(class
    [System.Runtime]System.Func`2<int32, int32>, int32)
    IL_0027: ret
}
```

.NET9 и .NET 10 (Tier0)



```
G_M000_IG02:                      ; offset=0x000B
    mov    rdi, 0x7F5B4F010250
    call   CORINFO_HELP_NEWSFAST // class
    mov    r15, rax
    mov    dword ptr [r15+0x08], ebx // y

    mov    rdi, 0x7F5B4F010518
    call   CORINFO_HELP_NEWSFAST // func
    mov    rbx, rax

    lea    rdi, bword ptr [rbx+0x08]
    mov    rsi, r15
    call   CORINFO_HELP_ASSIGN_REF // class in func

    mov    rsi, 0x7F5B4EE78A08
    mov    qword ptr [rbx+0x18], rsi

    mov    esi, dword ptr [r15+0x08]
    mov    rdi, qword ptr [rbx+0x08]
    call   [rbx+0x18]System.Func`2[int,int]:Invoke(int):int:this

    add    eax, eax
```

Делегаты .NET 10 (Tier1)



```
G_M000_IG02:          ;; offset=0x000A
    mov    rdi, 0x7FD9A27FBEA8
    call   CORINFO_HELP_NEWSFAST
    mov    dword ptr [rax+0x08], ebx
    mov    esi, dword ptr [rax+0x08]
    mov    rdi, rax
    mov    rax, 0x7FD9A2638FF0
    call   rax
    add    eax, eax
```

Почему .NET внедряет это только сейчас?

Меньшая необходимость из-за наличия `value types` в .NET.



Компромисс между временем JIT-компиляции и выигрышем



Ранние попытки и приоритеты развития.



Управление кодом

Зачем

1) Куча неочевидных вещей, которые реально сложно запомнить



```
if (typeof(T).IsAssignableTo(typeof(IDisposable))) {  
    ((IDisposable)(object)o).Dispose(); // вручную  
}
```

Зачем ???

|
(2) length – константа,
sumSize ≤ JitObjectStackAllocationSize?
|
нет → heap
|
v
(3) Элементный тип:
• GC? yes
• sealed? yes
|

```
✓ ✎ 38 src/libraries/System.Private.CoreLib/src/System/Collections/Generic/Dictionary.cs  
574 565 else  
575 566 {  
576 567     Debug.Assert(comparer is not null);  
577 568     while (true)  
578 569         while ((uint)i < (uint)entries.Length)  
579 580             {  
580 581                 // Should be a while loop https://github.com/dotnet/runtime/issues/568  
581 582                 // Test uint in if rather than loop condition to drop range check  
582 583                 if ((uint)i >= (uint)entries.Length)  
583 584                     {  
584 585                         break;  
585 586                 }  
586 587             if (entries[i].hashCode == hashCode && comparer.Equals(entries[i], entry))  
587 588                 return entry;  
588 589             i++;  
589 590         }  
590 591     }  
591 592 }
```

Зачем

2) Правила меняются от версии к версии

.net 7 JIT научился **inline** методы, содержащие **string literals** между сборками



```
private bool _value = true;  
  
[Benchmark]  
public int BoolStringLength() => _va
```

Это ограничение существовало из-за механизма ин

.net 9 add support Shared-generic → shared-generic



```
[MethodImpl(MethodImplOptions.NoInlining)]  
bool Test<T>() => Callee<T>();  
  
bool Callee<T>() => typeof(T).IsValueType;
```

JIT умел искать нужный TypeHandle только в собственном root-методе.



```
; Assembly listing for method Program:Test[System.__Canon]()<bool:this  
push rbx  
sub rsp, 48  
mov qword ptr [rsp+0x28], rdx  
mov rbx, rcx  
mov rcx, qword ptr [rdx+0x10]  
mov rax, qword ptr [rcx+0x10]  
test rax, rax  
je SHORT G_M000_IG04  
mov rdx, rax  
jmp SHORT G_M000_IG05  
G_M000_IG04:  
mov rdx, rdx  
mov rdx, 0x7FFCC5FE7088  
call CORINFO_HELP_RUNTIMEHANDLE_METHOD  
mov rdx, rax  
G_M000_IG05:  
mov rcx, rbx  
call [Program:Callee[System.__Canon]()<bool:this>] ;;; <---- not inlined  
nop  
add rsp, 48  
pop rbx  
ret
```

CA1871: не передавайте структуру, допускающую значение NULL, в "ArgumentNullException.ThrowIfNull"

```
[MethodImpl(MethodImplOptions.NoInlining)]
static void Test()
{
    long gc = GC.GetAllocatedBytesForCurrentThread();
    for (int i = 0; i < 100; i++)
    {
        ThrowIfNull(i);
    }
    gc = GC.GetAllocatedBytesForCurrentThread() - gc;

    Console.WriteLine(gc);
    Thread.Sleep(1000);
}

static void ThrowIfNull<T>(T value) => ArgumentNullException.ThrowIfNull(value);
```

При запуске этого кода на .NET 8 я получаю такие результаты:

```
2400
2400
2400
0
0
0
```

CA1852: Seal internal types

Да у нас есть
PGO, Devirtualization, GDV но
все равно...

как мы видели например для
переноса на stack sealed нужен

Example

The following code snippet shows a violation of CA1852:

C#

```
internal class C  
{ }
```

The following code snippet fixes the violation:

C#

```
internal sealed class C  
{ }
```

Вывод

Системно разложили ГДЕ ЧТО и КАК делает JIT



Рассмотрели некоторые из огромного множества
улучшения за последние версии .net у JIT



Разобрали связку оптимизаций escape analize ->
Stack replacement -> Struct Promotion -> Constant
Folding



Еще больше про оптимизации в новых версиях

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-10>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-9>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-8>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-7>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-6>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-4>

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-3>