Playing with CLR

Станислав Сидристый,
Twitter: **@sidristij**
Н: habrahabr.ru/users/sidristij

Цели доклада

- **Теоретическая часть**
  - Вспомнить базовые знания по памяти в .Net (совсем быстро)
  - Рассмотрим модель типов .Net и поддержку наследования
  - Рассмотрим как работает код в принципе

- **Практическая часть**
  - Достанем скрытые структуры CLI без использования рефлексии
  - Пробросим ***object*** в другой ***AppDomain***
  - Построим свой ***ObjectPool*** вне .Net памяти
  - Соберем в памяти все .Net объекты
  - Склонируем поток

- **Сделать выводы**

От куч до процессора

БАЗОВЫЕ ПРЕДСТАВЛЕНИЯ О ПАМЯТИ В .NET

Базовые знания

Области памяти:

- Есть **стек** потока – он для работы методов
- Есть **SOH** – он для «маленьких» объектов (<85000B)
- Есть **LOH** – он для «больших» объектов
- Есть **Code Heap** – там размещаются результаты JIT компиляции
- Есть также **High Frequency Heap** – содержит структуры поддержки системы типов, к которым идет частое обращение (MethodTable, VTable)
- Есть еще **Low Frequency Heap** – содержит редкоиспользуемые структуры поддержки системы типов
- **Stub Heap** – содержит заглушки для COM Interop, p/invoke

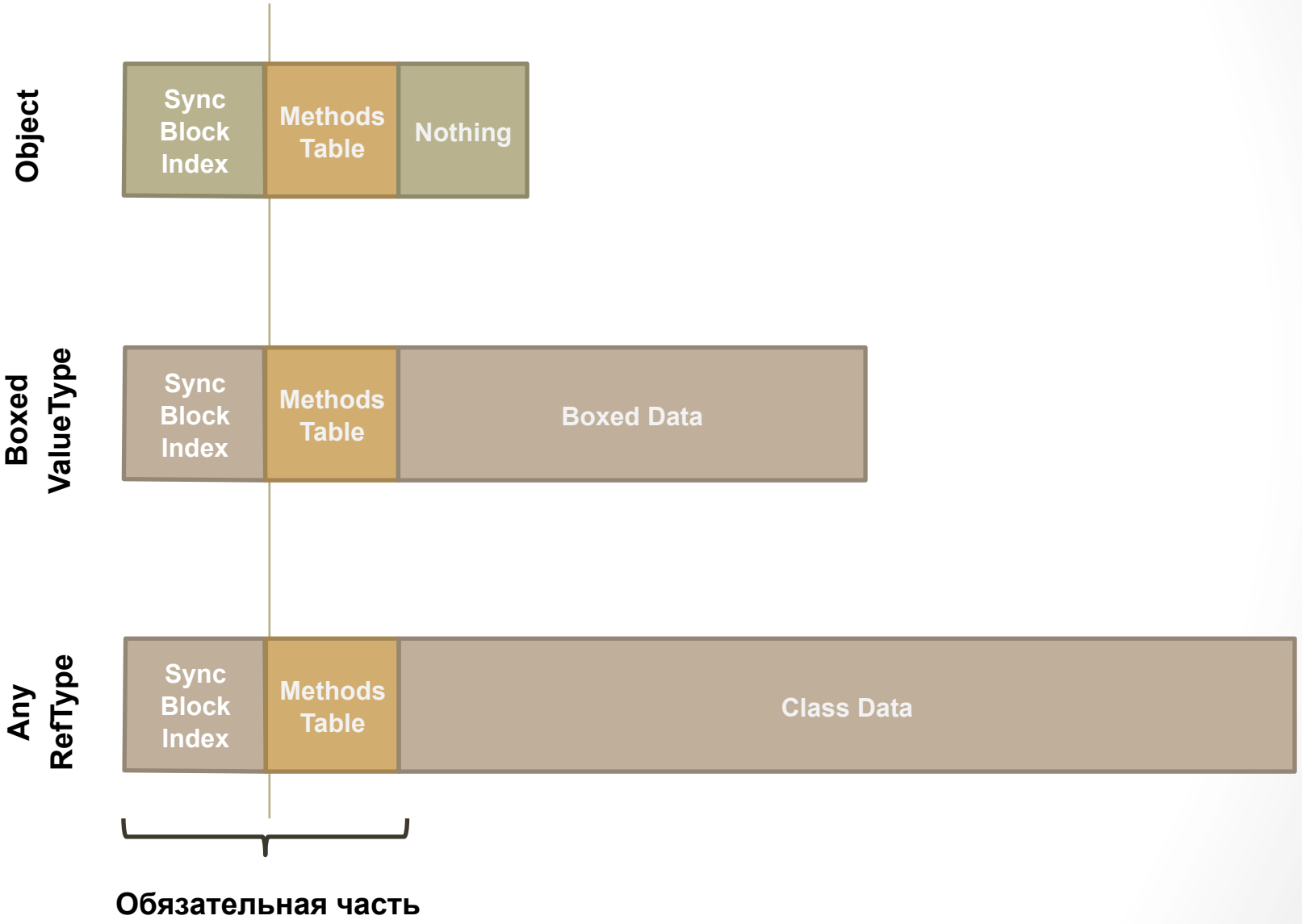
Память – виртуальная

- Это значит что доступное адресное пространство – не RAM
- Часть – на диске, часть – в RAM
- Доступность не линейная, а «островками» с настройками прав доступа
- Нет понятия AppDomain – значит нет ограничения доступа между ними
- А при наличии прав, можно снять память чужого процесса

От объекта до массива

СТРУКТУРА ОБЪЕКТОВ .NET

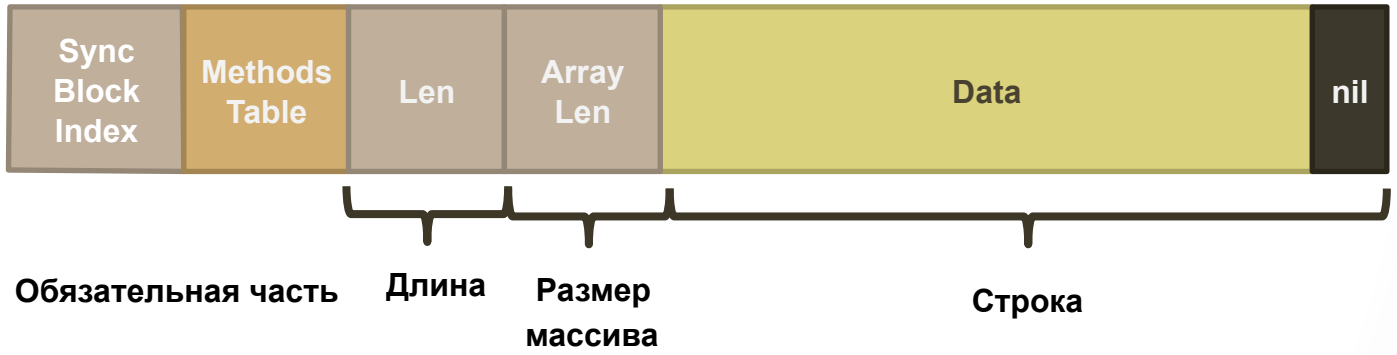
Структура типов данных



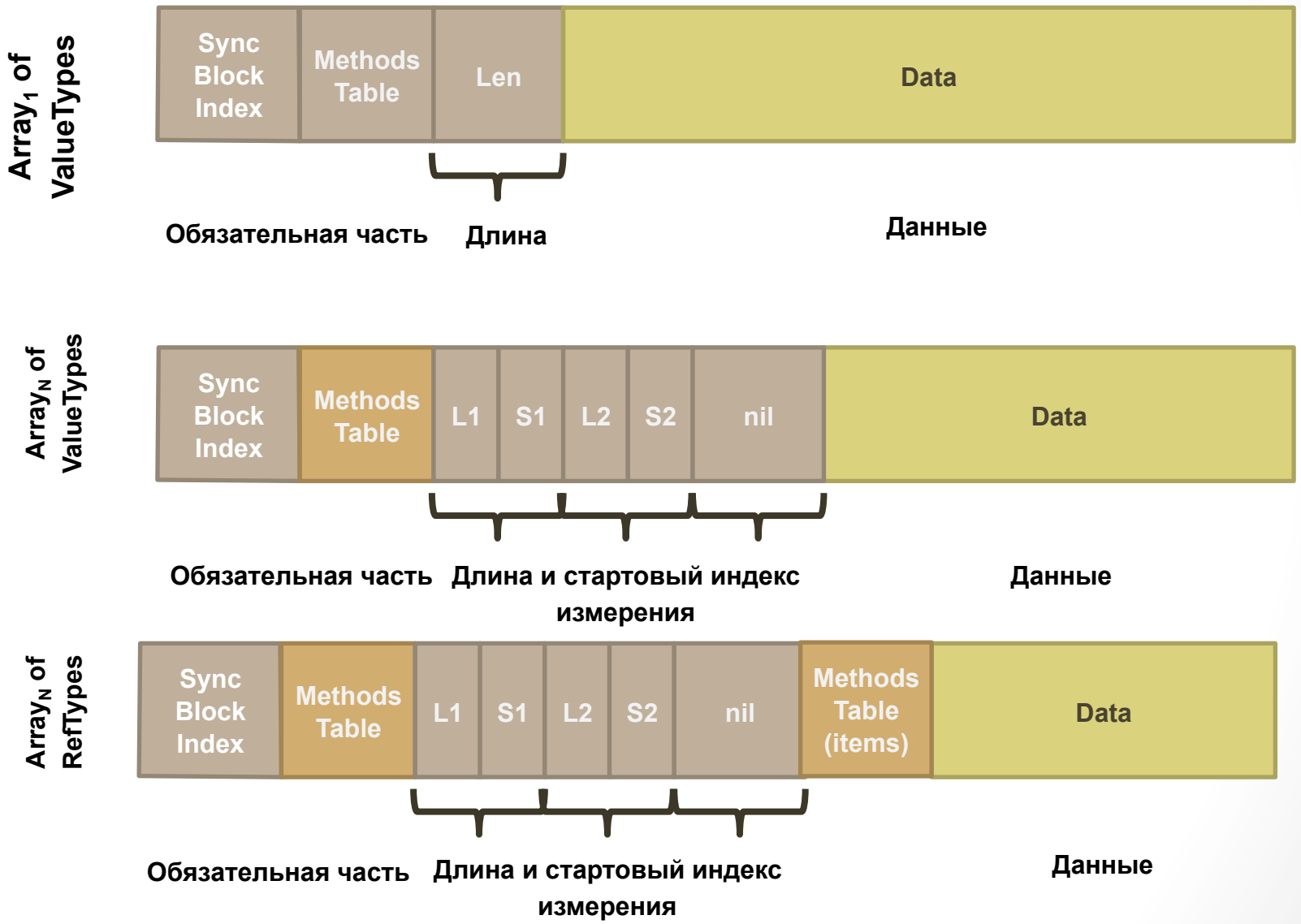
Структура типов данных

String 4.0+

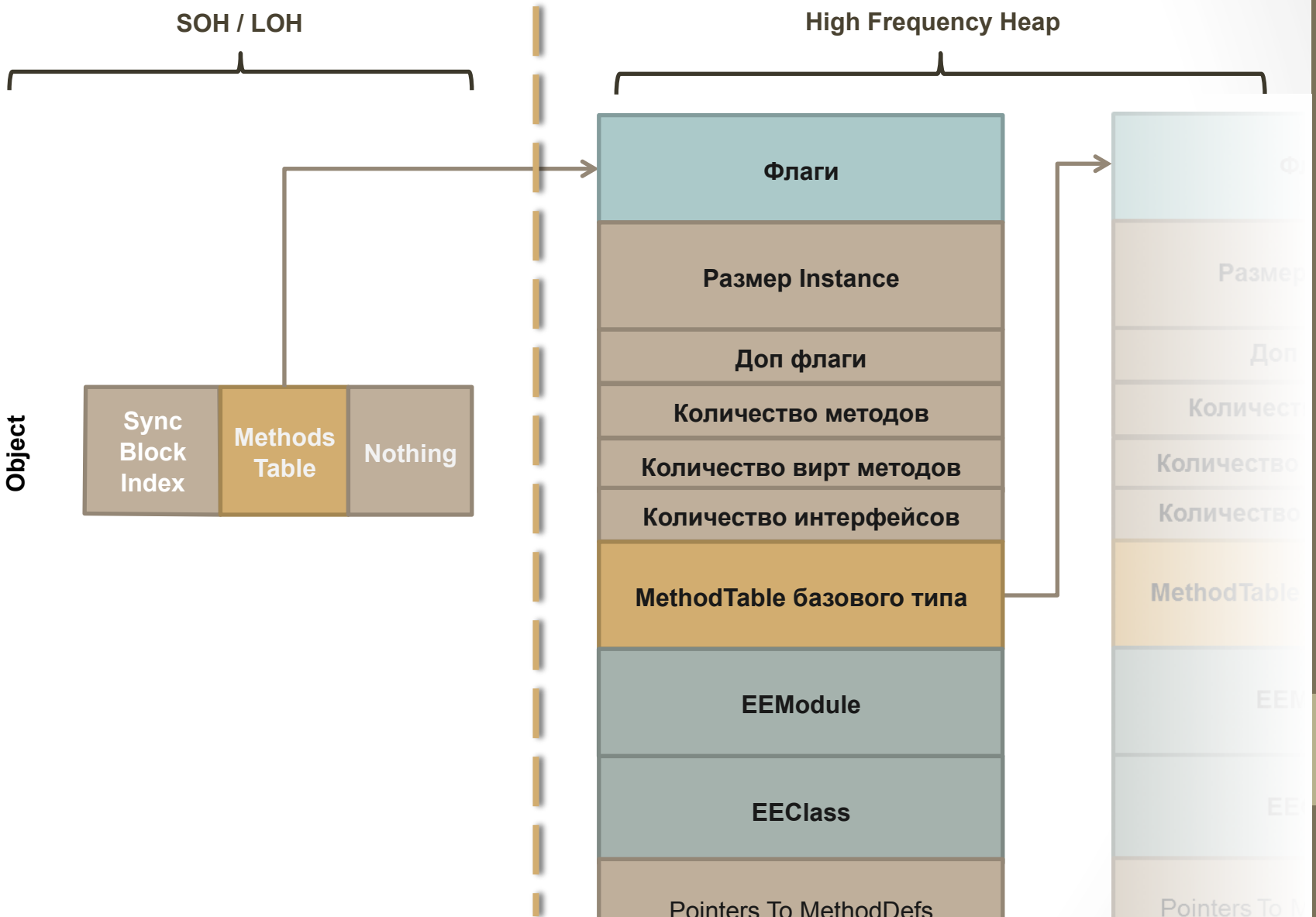
String 3.5-



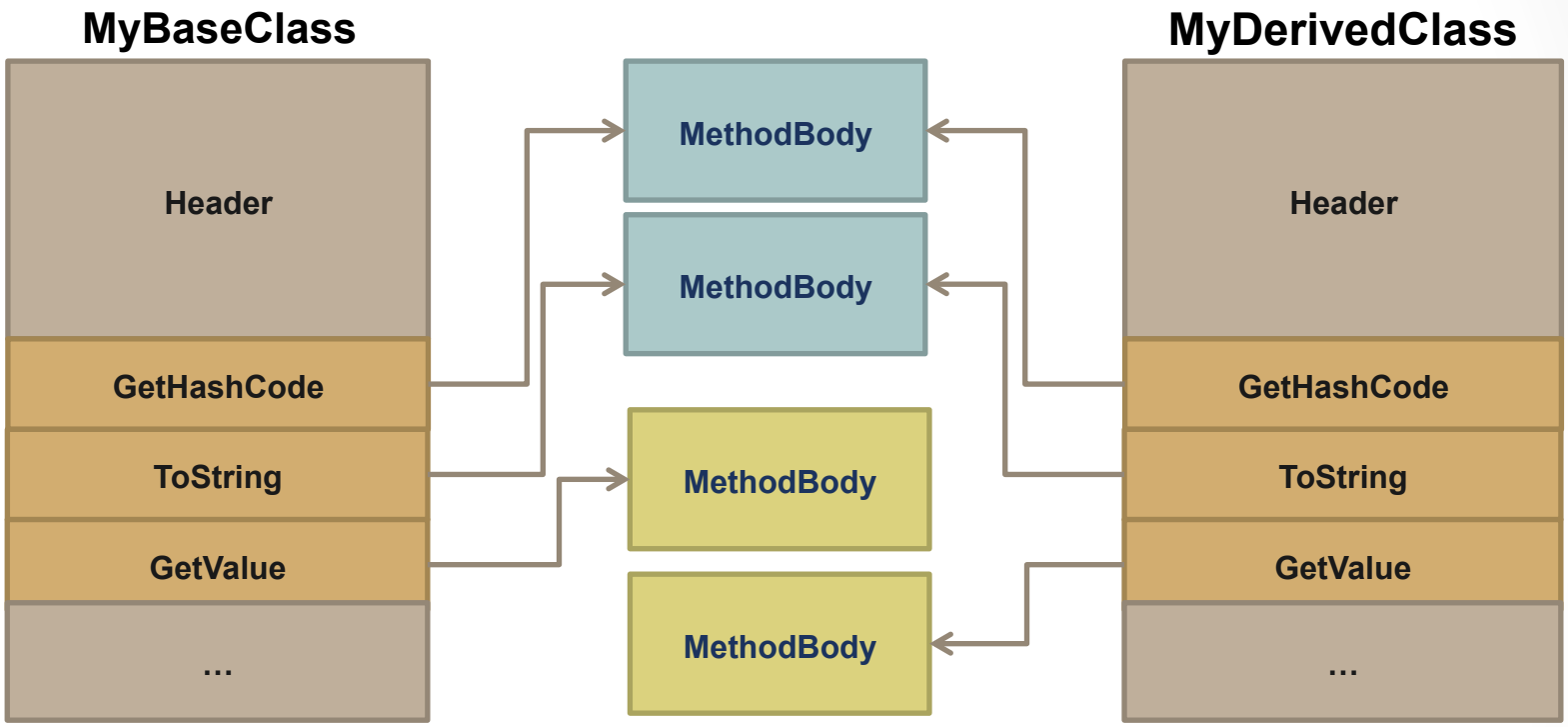
Структура типов данных



Структура типов данных



Работа наследования

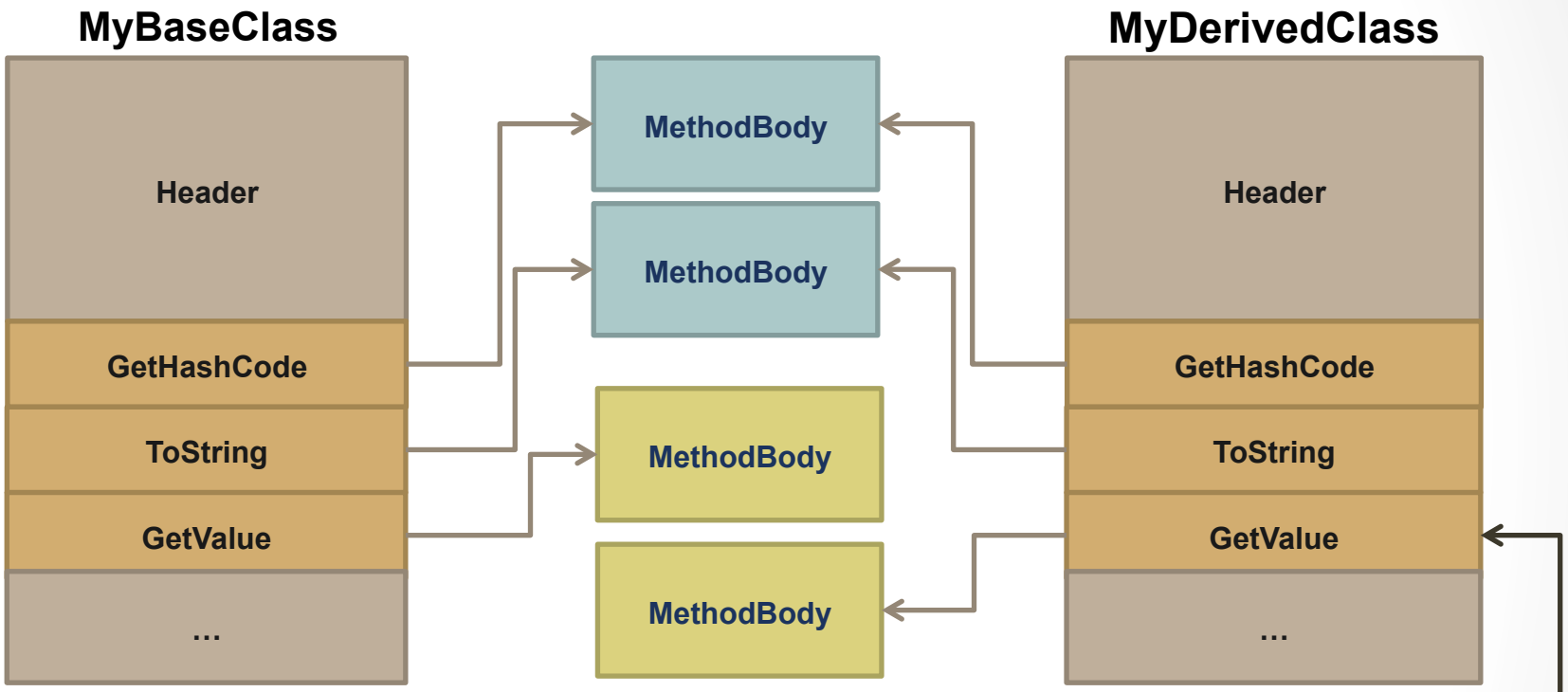



```
public class MyClass
{
    public virtual string GetValue()
    {
        return "base";
    }
}
```

```
public class MyOverridedClass : MyClass
{
    public override string GetValue()
    {
        return "overrided";
    }
}
```

```
MyClass var = new MyOverridedClass();
Console.WriteLine(var.GetValue());
```

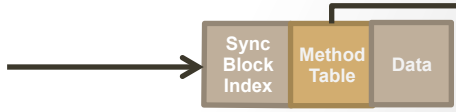
Работа наследования




```
public class MyClass
{
    public virtual string GetValue()
    {
        return "base";
    }
}
```

```
public class MyOverridedClass : MyClass
{
    public override string GetValue()
    {
        return "overrided";
    }
}
```

```
MyClass var = new MyOverridedClass();
Console.WriteLine(var.GetValue());
```



Промежуточные выводы

- Зная структуру внутренних объектов CLR можно манипулировать ими
- Например, попробовать создать объект без использования оператора new
- Также можно попробовать работать с не-MarshalByRefObjects из другого AppDomain, уменьшая расходы памяти и времени на сериализацию / десериализацию, но оставляя возможность выгрузки сборок

От получения указателя на объект до снятия дампа с виртуальной памяти с разбором объектов в ней

ПРАКТИЧЕСКИЕ ПРИМЕРЫ

Как работает код?

КЛОНИРОВАНИЕ ПОТОКА

Как работает код?

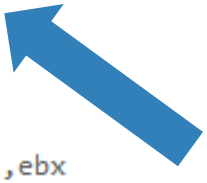
Для того чтобы в различных потоках имелась поддержка вложенности вызова методов с разделением истории вызовов, используются стеки

```
public void Method1()
{
    Method2();
}

public void Method2()
{
    // ...
}
```

```
        public Sample()
00000000  push        ebp
00000001  mov         ebp,esp
00000003  push        edi
00000004  push        esi
00000005  push        ebx
00000006  sub         esp,30h
00000009  xor         ebx,ebx
0000000b  mov         dword ptr [ebp-10h],ebx
0000000e  mov         dword ptr [ebp-1Ch],ebx
00000011  mov         dword ptr [ebp-3Ch],ecx
00000014  cmp         dword ptr ds:[004D0B14h],0
0000001b  je          00000022
0000001d  call        56413AFA
00000022  mov         ecx,dword ptr [ebp-3Ch]
00000025  call        554B0A58
0000002a  nop
        {
0000002b  nop
            Method1();
0000002c  mov         ecx,dword ptr [ebp-3Ch]
0000002f  call        FFF78810
00000034  nop
        }
00000035  nop
00000036  nop
00000037  lea         esp,[ebp-0Ch]
0000003a  pop         ebx
0000003b  pop         esi
0000003c  pop         edi
0000003d  pop         ebp
0000003e  ret
```

```
        public Sample()
00000000  push        ebp
00000001  mov         ebp,esp
00000003  push        edi
00000004  push        esi
00000005  push        ebx
00000006  sub         esp,30h
00000009  xor         ebx,ebx
0000000b  mov         dword ptr [ebp-10h],ebx
0000000e  mov         dword ptr [ebp-1Ch],ebx
00000011  mov         dword ptr [ebp-3Ch],ecx
00000014  cmp         dword ptr ds:[004D0B14h],0
0000001b  je          00000022
0000001d  call        56413AFA
00000022  mov         ecx,dword ptr [ebp-3Ch]
00000025  call        554B0A58
0000002a  nop
        {
0000002b  nop
            Method1();
0000002c  mov         ecx,dword ptr [ebp-3Ch]
0000002f  call        FFF78810
00000034  nop
        }
00000035  nop
00000036  nop
00000037  lea         esp,[ebp-0Ch]
0000003a  pop         ebx
0000003b  pop         esi
0000003c  pop         edi
0000003d  pop         ebp
0000003e  ret
```



Когда любой метод вызывается, тот сохраняет EBP – указатель на стек родительского метода. После чего сохраняет в EBP свой.

Перед выходом указатель восстанавливается.

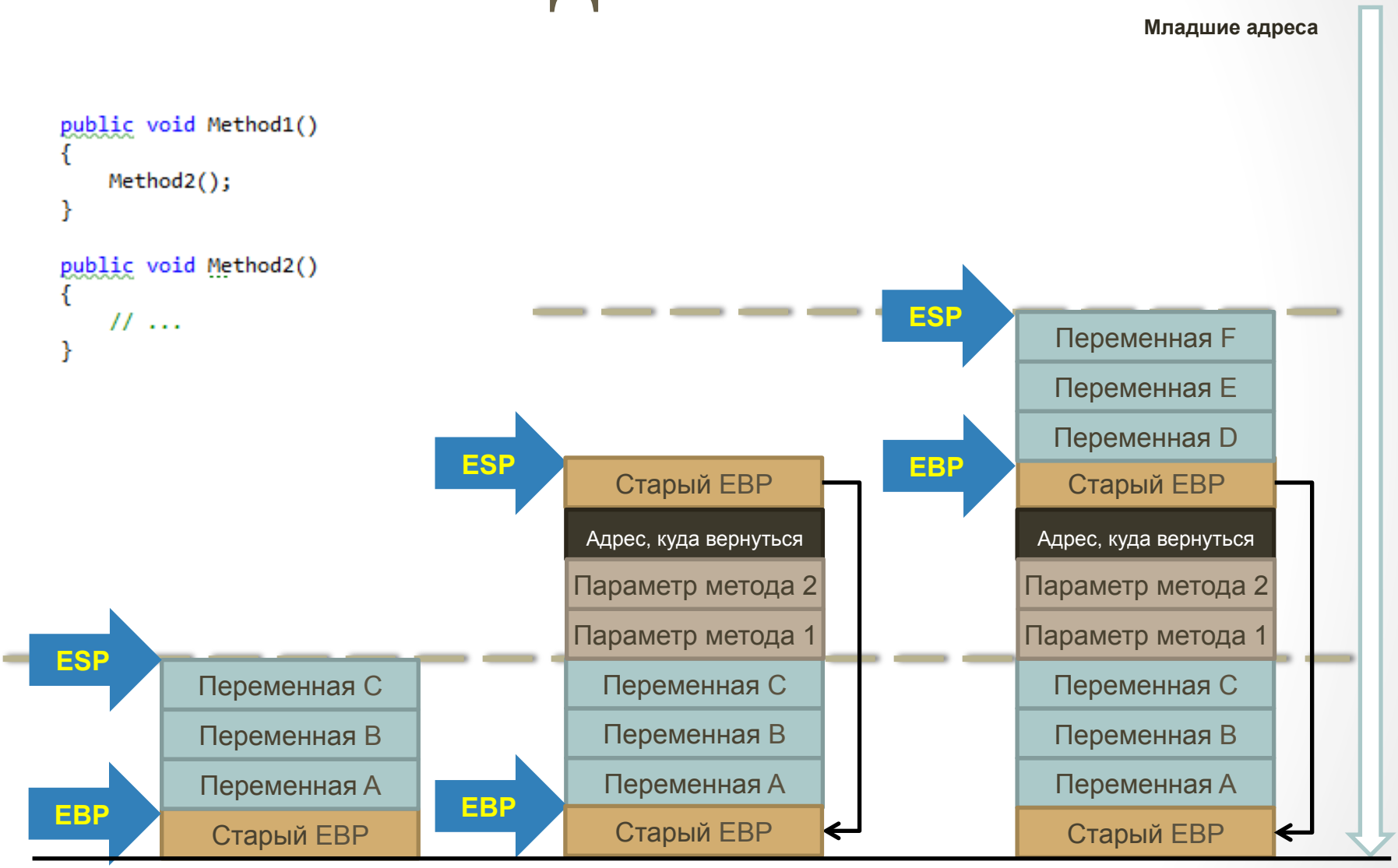
```
        {
00000000  push        ebp
00000001  mov         ebp,esp
00000003  push        edi
00000004  push        esi
00000005  push        ebx
00000006  sub         esp,30h
00000009  xor         edx,edx
0000000b  mov         dword ptr [ebp-10h],edx
0000000e  mov         dword ptr [ebp-1Ch],edx
00000011  mov         dword ptr [ebp-3Ch],ecx
00000014  cmp         dword ptr ds:[00930B14h],0
0000001b  je          00000022
0000001d  call        56263AAA
00000022  nop
            Method2(123, 456);
00000023  push        1C8h
00000028  mov         ecx,dword ptr [ebp-3Ch]
0000002b  mov         edx,7Bh
00000030  call        FFF487C8
00000035  nop
        }
00000036  nop
00000037  lea         esp,[ebp-0Ch]
0000003a  pop         ebx
0000003b  pop         esi
0000003c  pop         edi
0000003d  pop         ebp
0000003e  ret
```

- Для того чтобы передать в метод параметры, также используется стек. Однако, некоторые параметры могут идти через регистры (fastcall)
- Инструкция **Call** помещает в стек адрес следующей инструкии чтобы знать куда сделать **return**

Вызов метода

```
public void Method1()
{
    Method2();
}

public void Method2()
{
    // ...
}
```



Вызов метода

Промежуточные выводы

- У каждого потока есть своя область памяти для поддержки вызовов методов
- В этой области хранятся:
  - Параметры, передаваемые в метод
  - Адрес, с которого CPU продолжит выполнения после выхода из метода
  - Адрес предыдущего фрейма вызова
  - Все локальные переменные
- Именно поэтому локальные переменные потоков не пересекаются
- Именно поэтому при рекурсивном вызове переменные не пересекаются
- Необходимо дополнительно востановить все регистры родительского потока

Описание процесса клонирования

КЛОНИРОВАНИЕ ПОТОКА

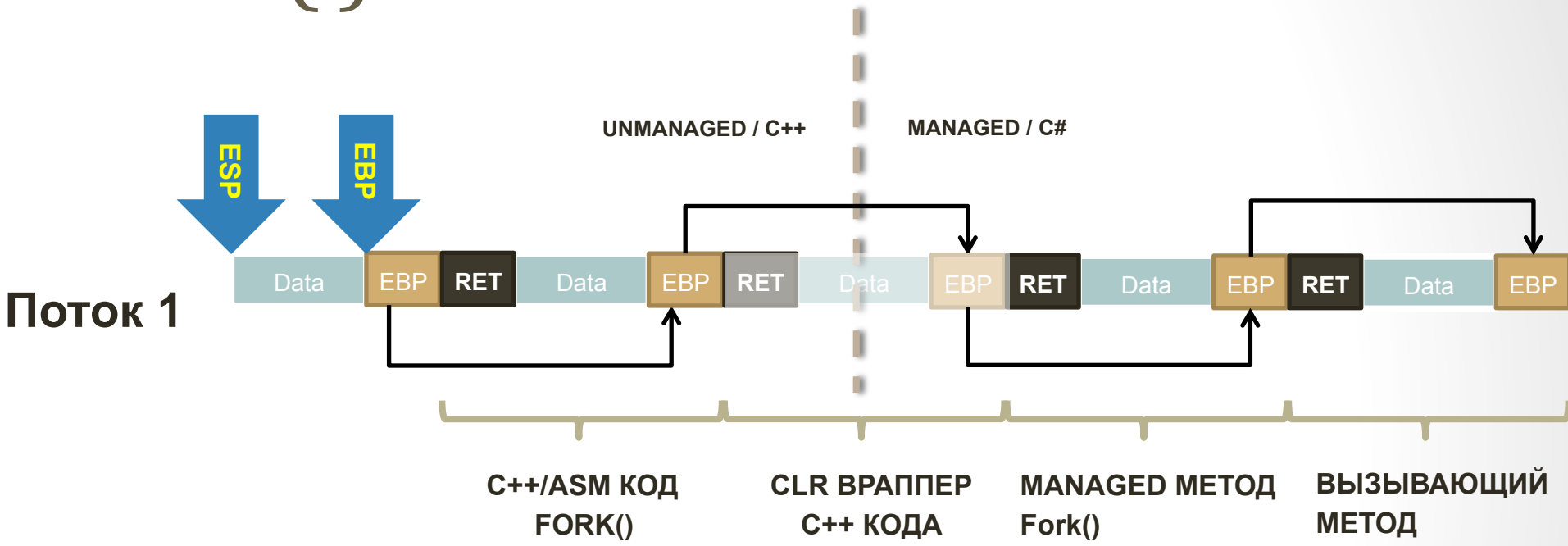
Клонируем поток

- Для того чтобы склонировать поток, необходимо
  - Создать новый поток
  - Сымитировать вызов метода, в котором происходит клонирование
  - Восстановить регистры родительского потока
  - Передать управление клонированного потока в точку ветвления родительского
- Что получим
  - Родительский поток будет исполняться как ни в чем ни бывало
  - С момента вызова Fork() дочерний поток начнет исполнение с точки вызова Fork()
  - Дочерний поток завершится с выходом из метода, в котором был вызван Fork(), т.е. Контекст работы Fork() ограничивается телом метода, в котором он был вызван

Клонируем поток

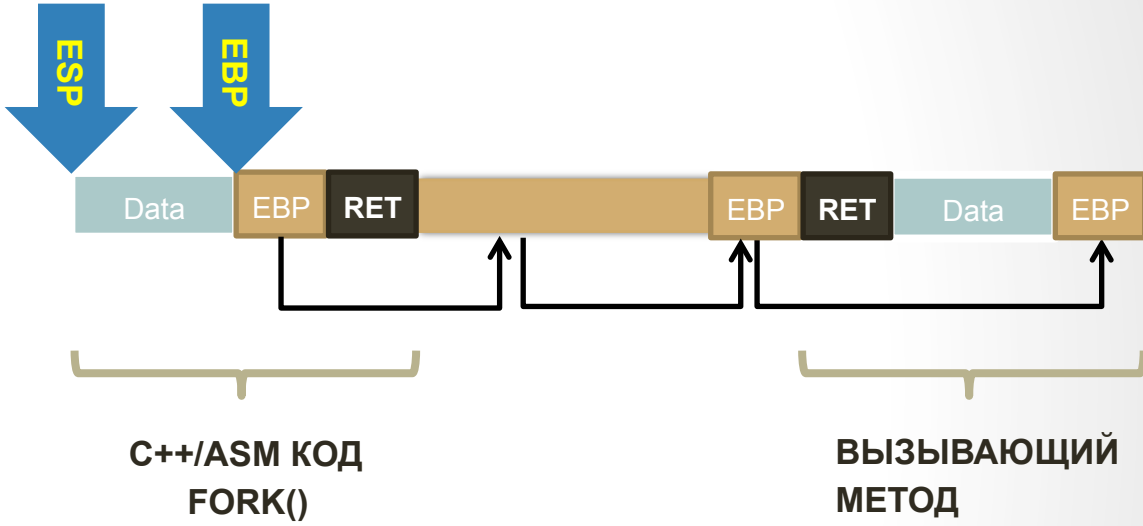
- Для того чтобы склонировать поток, необходимо
  - Создать новый поток
  - Сымитировать вызов метода, в котором происходит клонирование
  - Восстановить регистры родительского потока
  - Передать управление клонированного потока в точку ветвления родительского
- Что получим
  - Родительский поток будет исполняться как ни в чем ни бывало
  - С момента вызова Fork() дочерний поток начнет исполнение с точки вызова Fork()
  - Дочерний поток завершится с выходом из метода, в котором был вызван Fork(), т.е. Контекст работы Fork() ограничивается телом метода, в котором он был вызван

Fork( )


ESP
EBP
UNMANAGED / C++
MANAGED / C#
Data
EBP
RET
Data
EBP
RET
Data
EBP
RET
Data
EBP
RET
Data
EBP
Поток 1
C++/ASM КОД
FORK()
CLR ВРАППЕР
C++ КОДА
MANAGED МЕТОД
Fork()
ВЫЗЫВАЮЩИЙ
МЕТОД

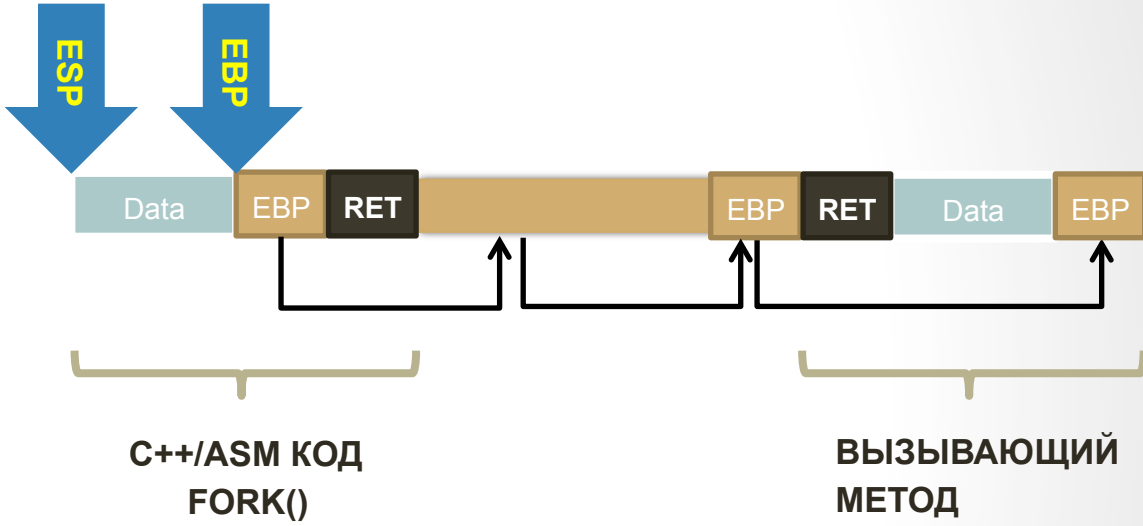
Fork( )

Поток 1

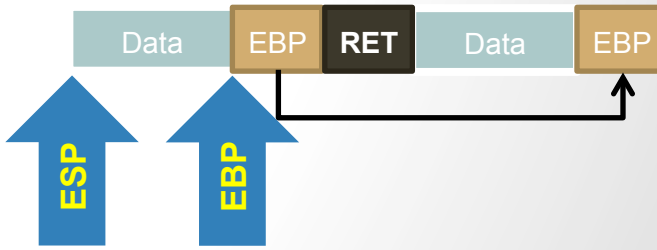

ESP
EBP
Data
EBP
RET
EBP
RET
Data
EBP
C++/ASM КОД
FORK()
ВЫЗЫВАЮЩИЙ
МЕТОД

Fork( )

Поток 1

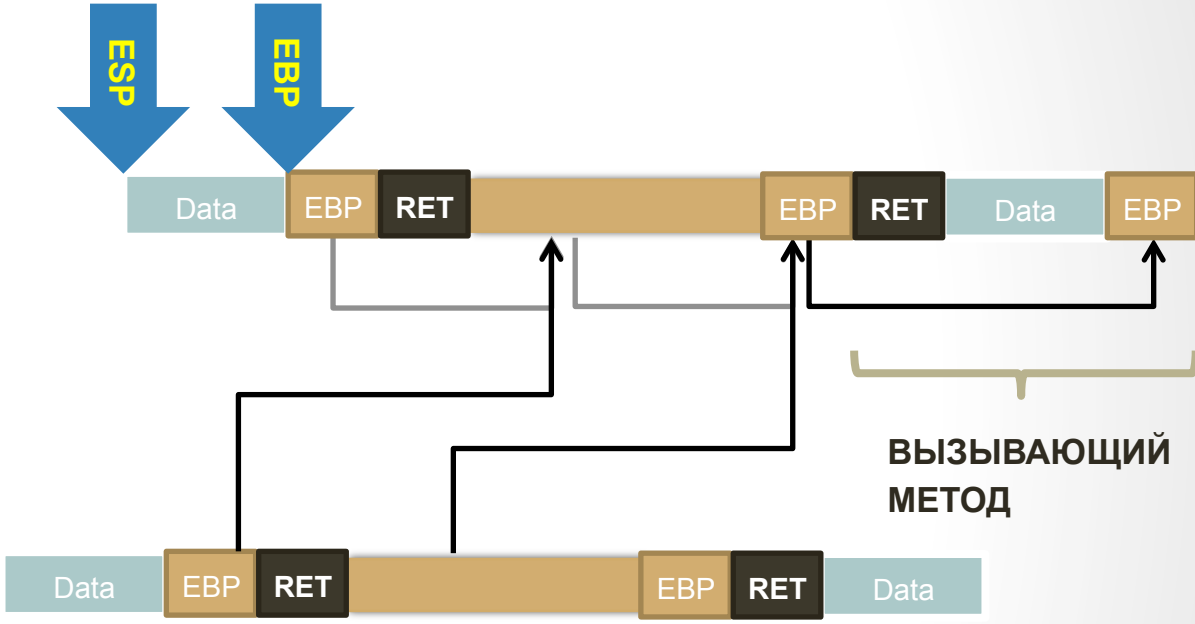


Поток 2

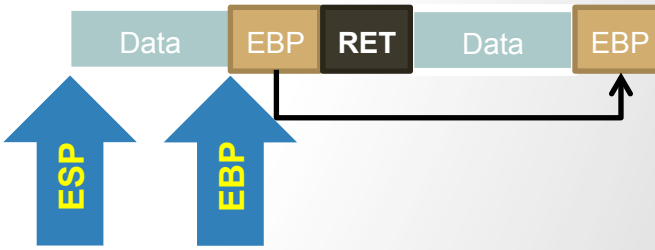


Fork( )

Поток 1

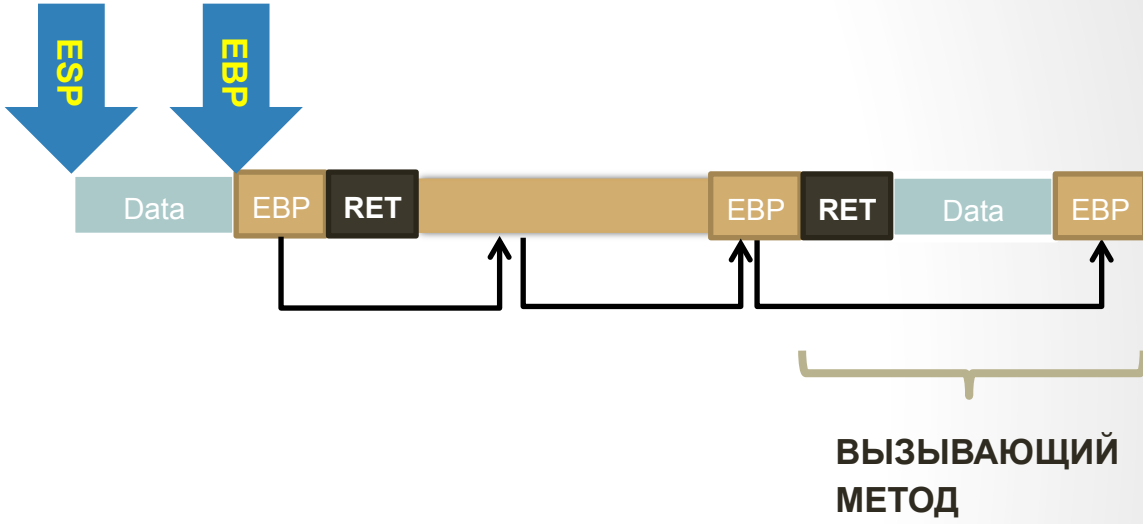


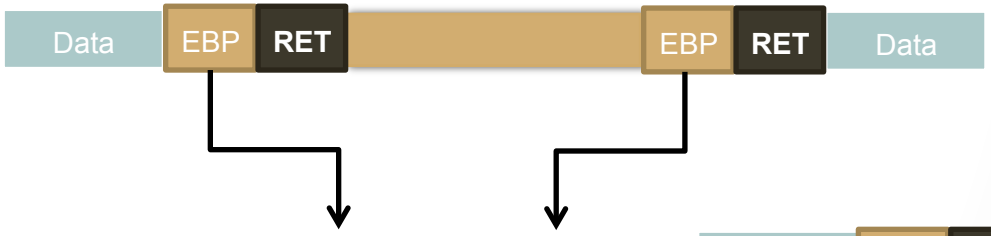
Поток 2



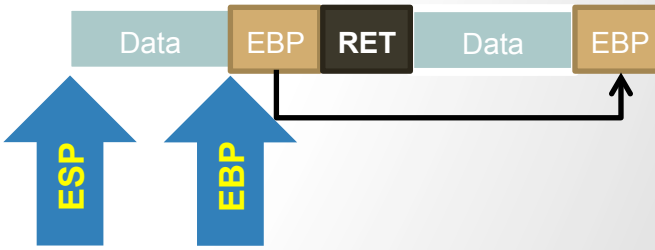
Fork( )

Поток 1



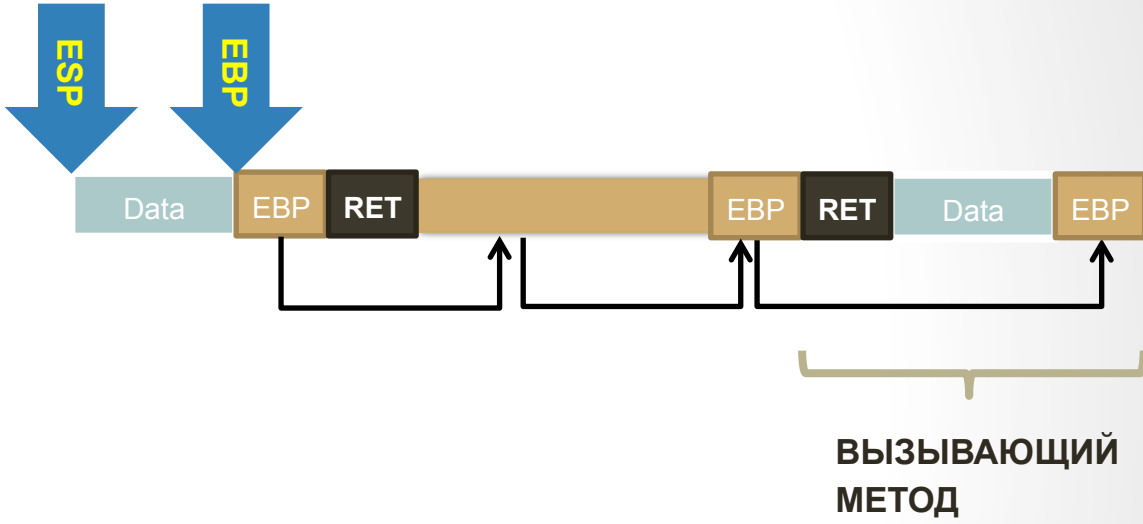


Поток 2

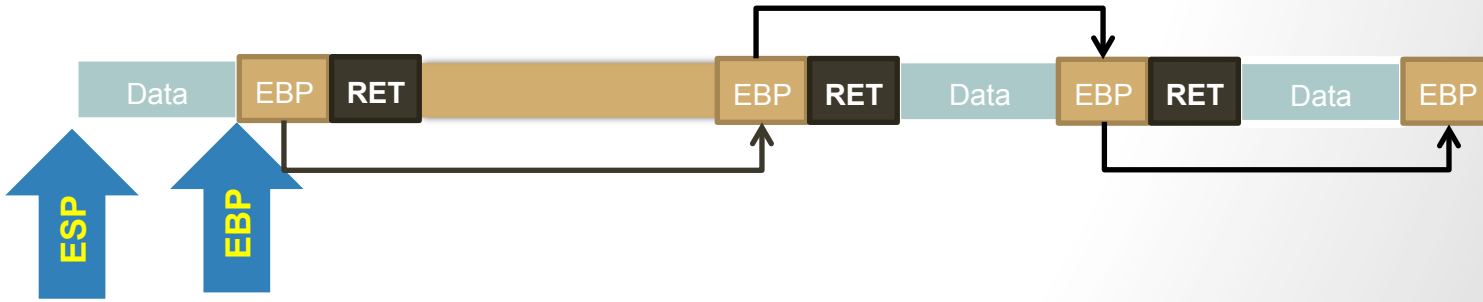


Fork( )

Поток 1

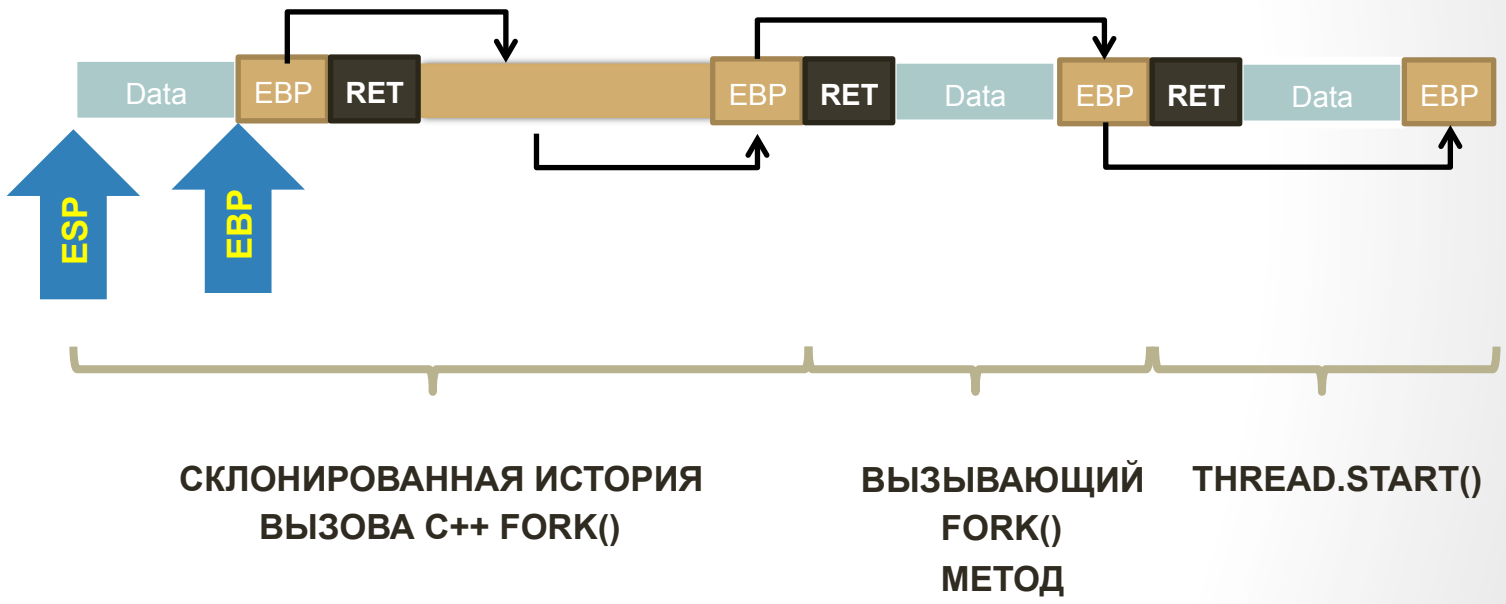


Поток 2



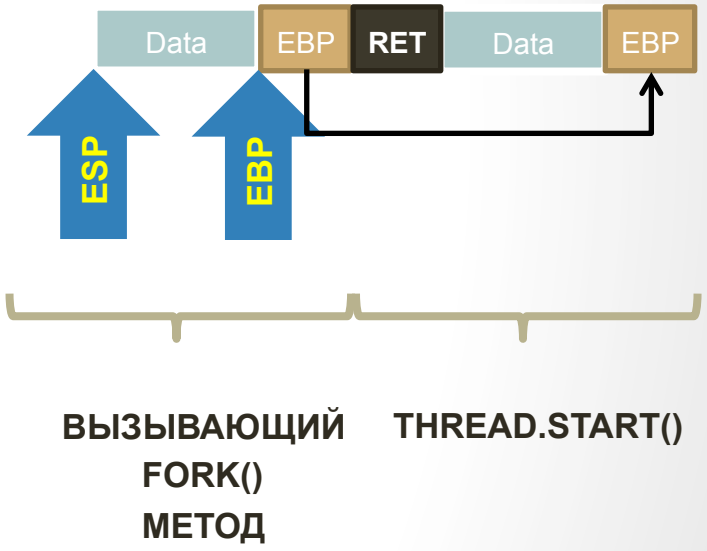
Fork( )

Поток 2


Data
EBP
RET
EBP
RET
Data
EBP
RET
Data
EBP
ESP
EBP
СКЛОНИРОВАННАЯ ИСТОРИЯ ВЫЗОВА C++ FORK()
ВЫЗЫВАЮЩИЙ FORK() МЕТОД
THREAD.START()

Fork( )

Поток 2


Data
EBP
RET
Data
EBP
ESP
EBP
ВЫЗЫВАЮЩИЙ FORK() МЕТОД
THREAD.START()

It’s **DEMO** time -)

КЛОНИРОВАНИЕ ПОТОКА

It’s **QUESTIONS** time -)

WELCOME!