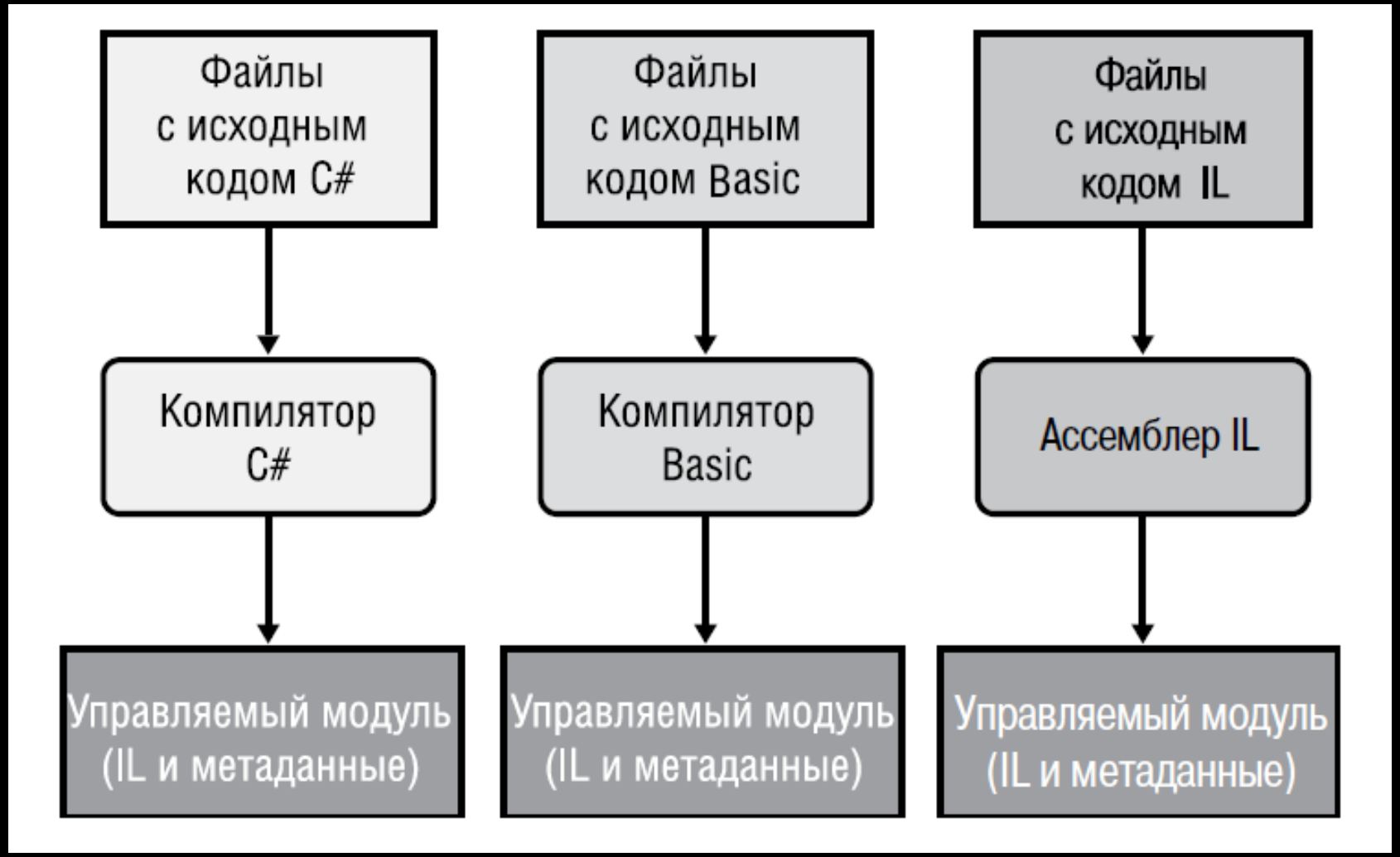
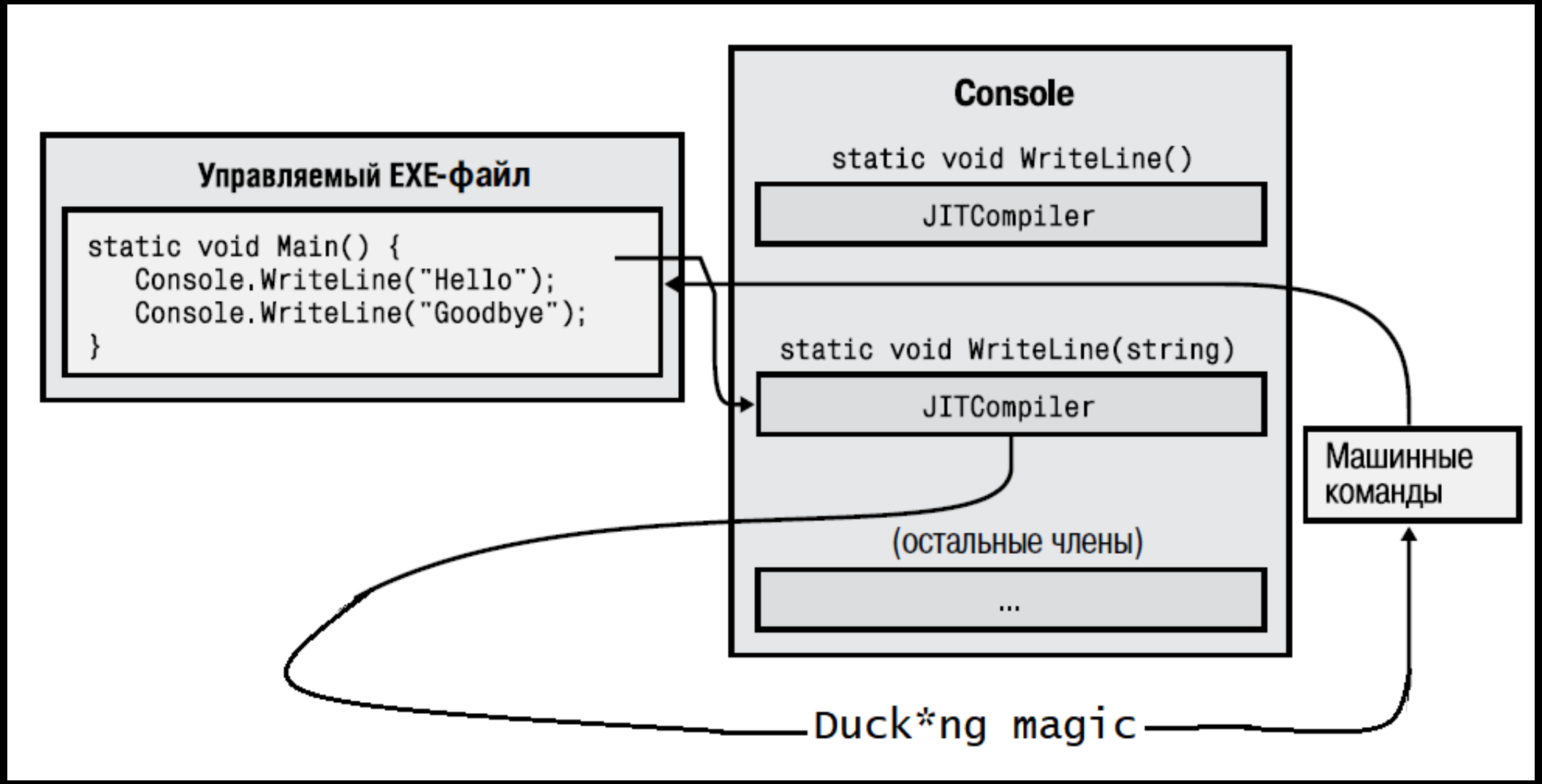
# C#

Brief

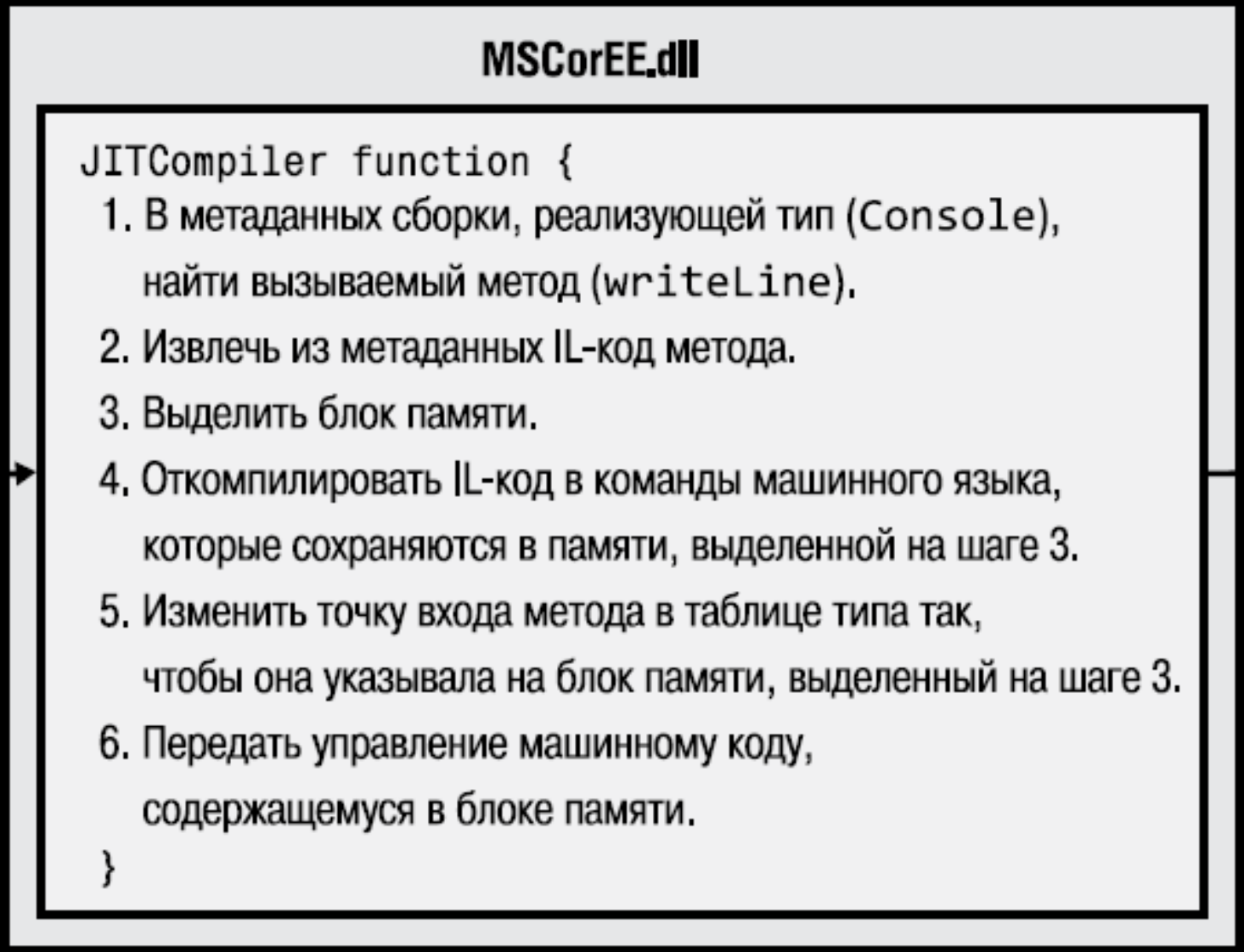
# Common Language Runtime



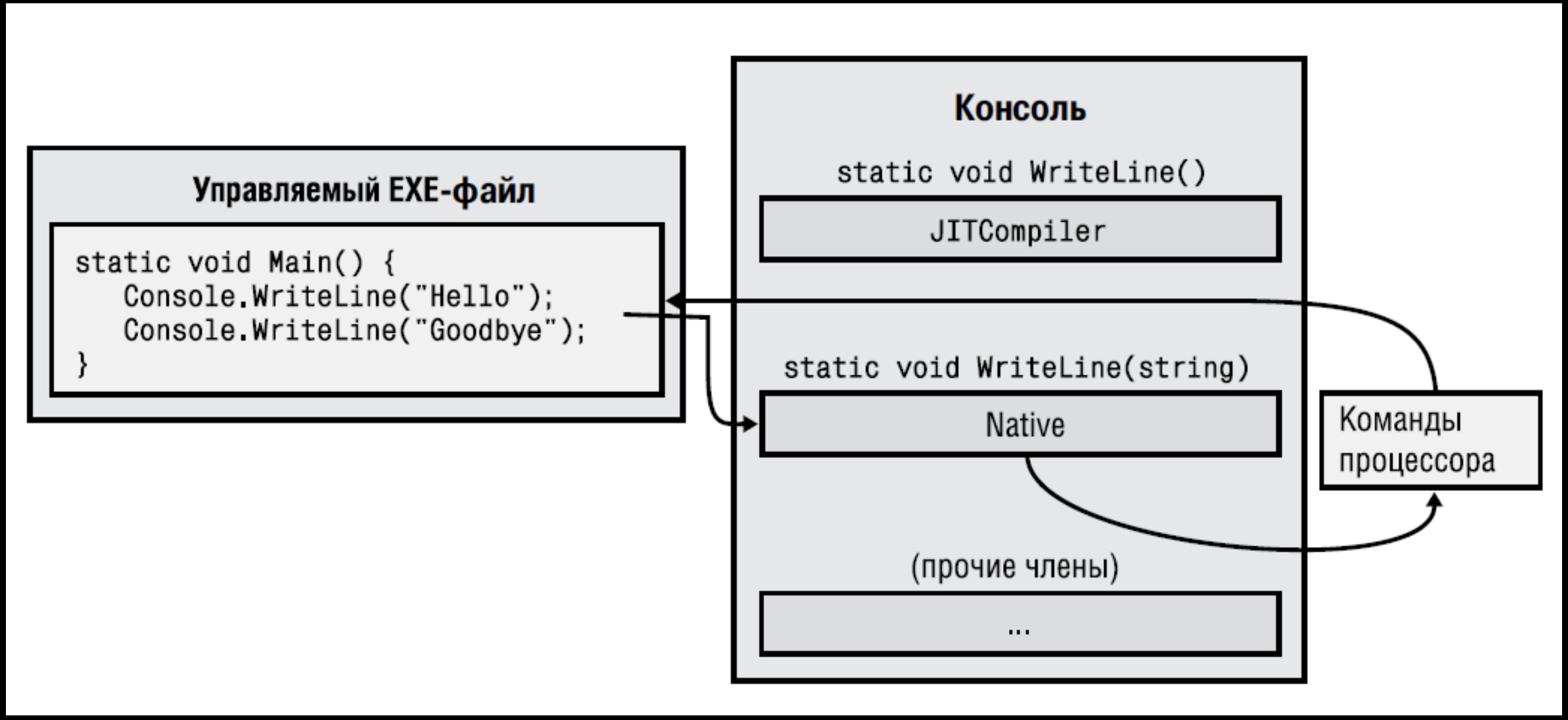
# Just In Time (JIT) Компилятор



# JIT Duck*ng magic



# JIT Повторный вызов метода



# JIT + -

# System.Object

- Equals()
- GetHashCode()
- ToString()
- GetType()
- - MemberwiseClone()
- - Finalize()

# Оператор new

Employee e = new Employee("ConstructorParam1");

- Вычисление кол-ва байтов
- Выделение в куче (заполняется 0)
- Инициализация указателя на объект-тип и индекса блока синхронизации
- Вызов конструктора экземпляра типа

# Приведение типов

```
// Этот тип неявно наследует от типа System.Object
internal class Employee
{
}

public sealed class Program
{
    public static void Main()
    {
        // Приведение типа не требуется, т. к. new возвращает объект Employee,
        // а Object — это базовый тип для Employee.
        Object o = new Employee();
        // Приведение типа обязательно, т. к. Employee — производный от Object
        // В других языках (таких как Visual Basic) компилятор не потребует
        // явного приведения
        Employee e = (Employee)o;
    }
}
```

# Приведение типов

```
object o = new object();
bool b1 = (o is object); // b1 равно true
bool b2 = (o is Employee); // b2 равно false

if (o is Employee)
{
    Employee e = (Employee)o; // Используем e внутри инструкции if
}

Employee e = o as Employee;
if (e != null)
{
}
```

# ValueType / ReferenceType

- память для ссылочных типов всегда выделяется из управляемой кучи
- каждый объект, размещаемый в куче, содержит дополнительные члены, подлежащие инициализации
- незанятые полезной информацией байты объекта обнуляются (это касается полей)
- размещение объекта в управляемой куче со временем инициирует сборку мусора.

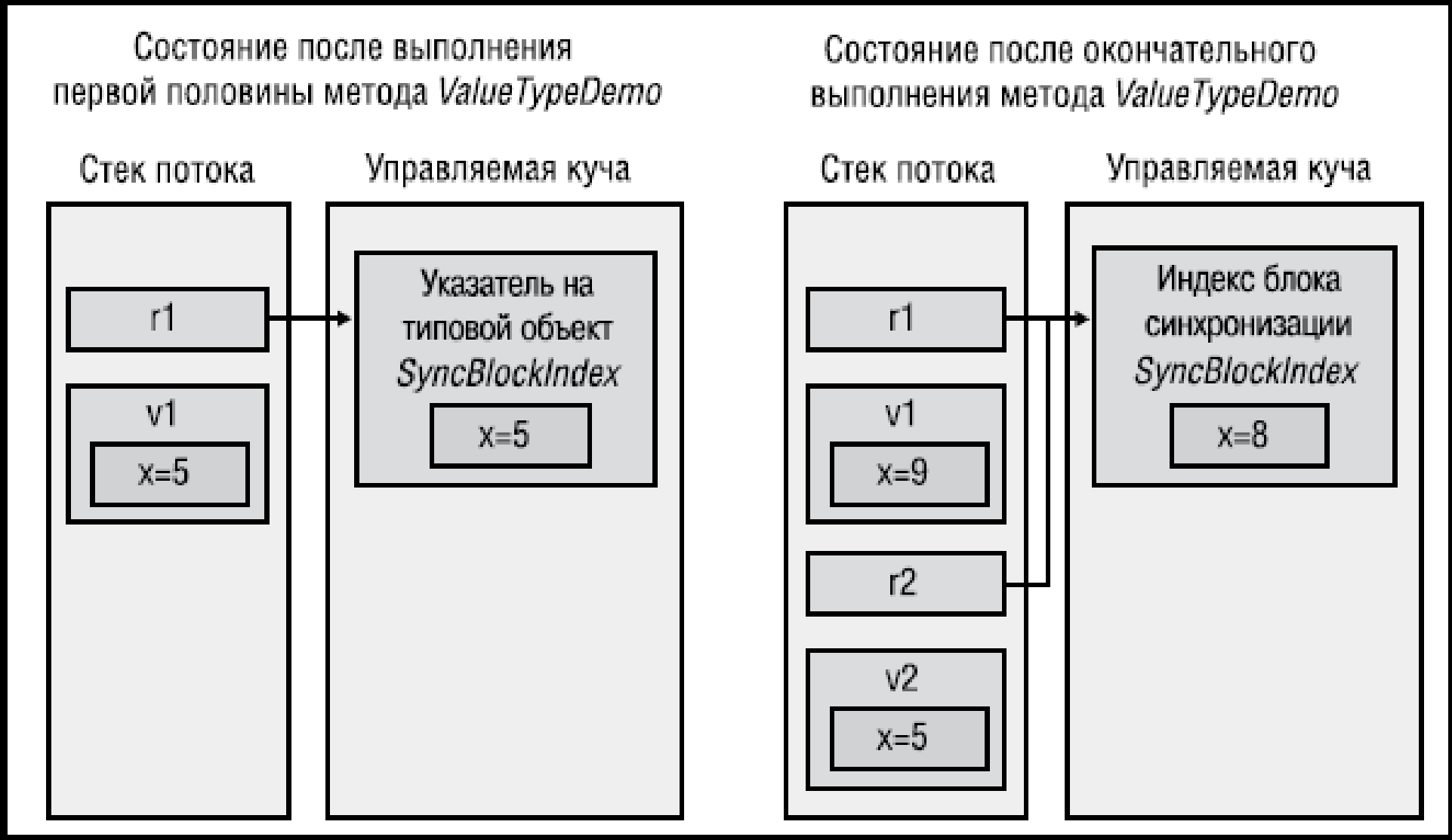
# Demo

```
// Ссылочный тип (поскольку 'class')
class SomeRef { public Int32 x; }
// Значимый тип (поскольку 'struct')
struct SomeVal { public Int32 x; }
```

# Demo

```
static void ValueTypeDemo()
{
    SomeRef r1 = new SomeRef(); // Размещается в куче
    SomeVal v1 = new SomeVal(); // Размещается в стеке
    r1.x = 5;
    v1.x = 5;
    Console.WriteLine(r1.x);
    Console.WriteLine(v1.x);
    //-----------------------------------------
    SomeRef r2 = r1;
    SomeVal v2 = v1;
    r1.x = 8;
    v1.x = 9;
    Console.WriteLine(r1.x);
    Console.WriteLine(r2.x);
    Console.WriteLine(v1.x);
    Console.WriteLine(v2.x);
}
```

# Demo


Состояние после выполнения первой половины метода *ValueTypeDemo*
Стек потока
Управляемая куча
r1
v1
x=5
Указатель на типовой объект *SyncBlockIndex*
x=5
Состояние после окончательного выполнения метода *ValueTypeDemo*
Стек потока
Управляемая куча
r1
v1
x=9
r2
v2
x=5
Индекс блока синхронизации *SyncBlockIndex*
x=8

# Выбор ValueType

- Тип похож на притивный
- Тип не обязан иметь базовый тип
- Тип не имеет производных типов
- Размер экземпляров типа мал (16 байт)
- Размер экземпляров велик, но они не передаются в качестве параметров и не являются результатами методов

# Отличия ValueTypes

- boxed/unboxed
- ValueType.Equals, ValueType.GetHashCode
- no abstract/virtual methods
- no null-value
- Копирование при присваивании
- Мгновенное высвобождение памяти

# Box/unbox

```
struct Point
{
    public Int32 x, y;
}

void Main()
{
    ArrayList a = new ArrayList();
    Point p; // Выделяется память для Point (не в куче)
    for (Int32 i = 0; i < 10; i++)
    {
        p.x = p.y = i; // Инициализация членов в нашем значимом типе
        a.Add(p); // Упаковка значимого типа и добавление
        // ссылки в ArrayList
    }
}
```

# Box/unbox

```
struct Point
{
    private Int32 m_x, m_y;
    public Point(Int32 x, Int32 y)
    {
        m_x = x;
        m_y = y;
    }
    public void Change(Int32 x, Int32 y)
    {
        m_x = x; m_y = y;
    }
}
```

# Box/unbox

```
void Main()
{
    Point p = new Point(1, 1);
    Console.WriteLine(p);

    p.Change(2, 2);
    Console.WriteLine(p);

    Object o = p;
    Console.WriteLine(o);

    ((Point)o).Change(3, 3);
    Console.WriteLine(o);
}
```

# Box/unbox

```
interface IChangeBoxedPoint
{
    void Change(Int32 x, Int32 y);
}
struct Point : IChangeBoxedPoint
{
    private Int32 m_x, m_y;
    public Point(Int32 x, Int32 y)
    {
        m_x = x;
        m_y = y;
    }
    public void Change(Int32 x, Int32 y)
    {
        m_x = x;
        m_y = y;
    }
}
```

# Box/unbox

```
void Main()
{
    Point p = new Point(1, 1);
    Console.WriteLine(p);

    p.Change(2, 2);
    Console.WriteLine(p);

    Object o = p;
    Console.WriteLine(o);

    ((Point)o).Change(3, 3);
    Console.WriteLine(o);
    // p упаковывается, упакованный объект изменяется и освобождается
    ((IChangeBoxedPoint)p).Change(4, 4);
    Console.WriteLine(p);

    // Упакованный объект изменяется и выводится
    ((IChangeBoxedPoint)o).Change(5, 5);
    Console.WriteLine(o);
}
```

# Свойства

```
public sealed class Employee
{
    public String Name; // Имя сотрудника
    public Int32 Age; // Возраст сотрудника
}
```

# Свойства

```
void Main()
{
    Employee e = new Employee();
    e.Name = "Jeffrey Richter"; // Задаем имя сотрудника
    e.Age = 48; // Задаем возраст сотрудника

    e.Age = -5; // Можете вообразить человека, которому минус 5 лет?
}
```

# Свойства

```
public sealed class Employee
{
    private String m_Name; // Поле стало закрытым
    private Int32 m_Age; // Поле стало закрытым
    public String GetName()
    {
        return (m_Name);
    }
    public void SetName(String value)
    {
        m_Name = value;
    }
    public Int32 GetAge()
    {
        return (m_Age);
    }
    public void SetAge(Int32 value)
    {
        if (value < 0)
            throw new ArgumentOutOfRangeException("value", value.ToString(),
                "The value must be greater than or equal to 0");
        m_Age = value;
    }
}
```

# Свойства

```
class Employee
{
    private Int32 m_Age;
    public String Name { get; set; }
    public Int32 Age
    {
        get => (m_Age);
        set
        {
            if (value < 0) // Ключевое слово value всегда
                // идентифицирует новое значение
                throw new ArgumentOutOfRangeException("value", value.ToString(),
                    "The value must be greater than or equal to 0");
            m_Age = value;
        }
    }
}
```

# Свойства

```
void Main()
{
    var e = new Employee();
    e.Name = "Jeffrey Richter"; // "Задать" имя сотрудника
    var EmployeeName = e.Name; // "Получить" имя сотрудника
    e.Age = 41; // "Задать" возраст сотрудника
    e.Age = -5; // Вброс исключения
    // ArgumentOutOfRangeException
    var EmployeeAge = e.Age; // "Получить" возраст сотрудника
}
```

# Обобщения

```
void SomeMethod()
{
    // Создание списка (List), работающего с объектами DateTime
    var dtList = new List<DateTime>();
    // Добавление объекта DateTime в список
    dtList.Add(DateTime.Now); // Без упаковки
    // Добавление еще одного объекта DateTime в список
    dtList.Add(DateTime.MinValue); // Без упаковки
    // Попытка добавить объект типа String в список
    dtList.Add("1/1/2004"); // Ошибка компиляции
    // Извлечение объекта DateTime из списка
    var dt = dtList[0]; // Приведение типов не требуется

    var inttList = new List<int>();
    //...
}
```

# Обобщения Ограничения

```
void SomeMethod1<T>(T t) where T : class {}
void SomeMethod2<T>(T t) where T : struct {}
void SomeMethod3<T>(T t) where T : Employee {}
//-------------------------------------
void SomeMethod4<T, T2>(T t, T2 t2) where T : T2{}
void SomeMethod5<T>(T t) where T : new() { }
```