

# Object Internals

**Tomáš Matoušek**  
&  
**Ladislav Prošek**

[tmd.havit.cz/teaching/csharp.htm](http://tmd.havit.cz/teaching/csharp.htm)

# Introduction to SSCLI – Rotor

- one of the CLI implementations
- source code available
- built using non-trivial scripts (multi-phase build)
- interesting source code directories:
  - clr/src
    - /vm – Execution Engine (C++)
    - /fjit – JITter (C++)
    - /csharp – C# compiler (C++)
    - /ilasm – IL Assembler tool (C++)
    - /ildasm – IL Disassembler tool (C++)
    - /bcl – Base Class Library (C#)
  - fx/src – some .NET Framework managed libraries (C#)
  - managedlibraries – additional managed libraries (C#)
  - jscript – JScript compiler (C#)

# External Methods

- methods implemented outside a declaring assembly
- declared
  - without a body
  - using modifier extern (a C# keyword)

```
public extern void ExternalMethod();
```

- a targets implementation defined by custom attributes
  - DllImportAttribute
    - binding to an unmanaged static entry point of a specified DLL
    - converted to P/Invoke call by C# compiler

```
[DllImport("kernel32.dll", EntryPoint="GetDiskFreeSpaceEx")]  
public static extern bool GetDiskFreeSpace(...);
```

- MethodImplAttribute

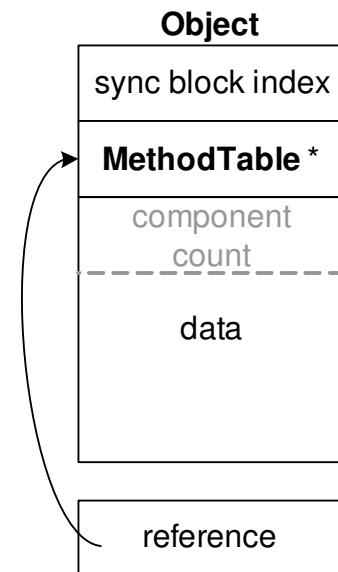
- binding to an EE internal method
- used in BCL (e.g. System.Object.GetHashCode method)

```
[MethodImplAttribute(MethodImplOptions.InternalCall)]  
public extern override int GetHashCode();
```

Rotor:  
vm/ecall.cpp

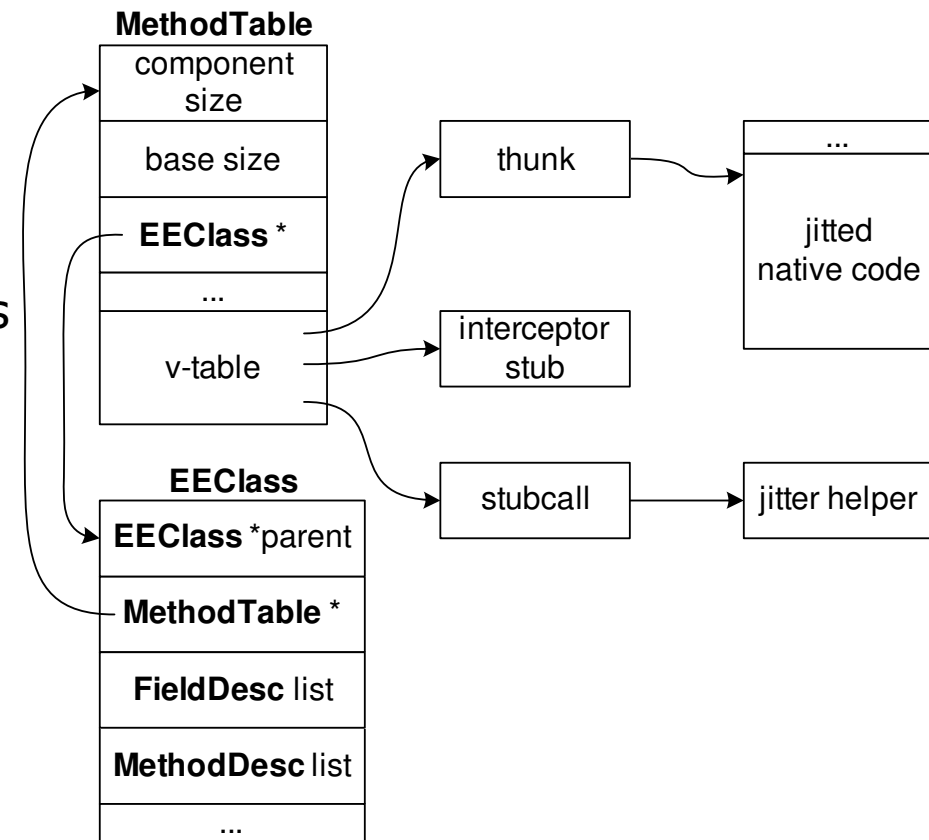
# Managed Object Representation

- two points of view on a managed object
  - a managed world view
    - through class `System.Object` in BCL
  - EE view
    - through class **Object**
- object layout
  - a sync block index
    - an index to EE internal table of sync blocks
    - associates additional data with the object
      - such as a synchronization lock
  - a pointer to a method table
    - an instance of class **MethodTable**
    - one per loaded type
  - component count
    - valid only for arrays and strings
  - some bits of sync block and method table pointer exploited for other purposes (e.g. GC)



# Method Table Content

- an object size (for GC)
  - $\text{BaseSize} + \text{ComponentSize} * \text{ComponentCount}$
- a pointer to an **EEClass** instance
  - run-time metadata
  - allocated separately to make working set smaller
- v-table slots
  - including non-virtuals and statics
  - initialized with *stubcalls*
    - invoke the JITter
  - thunk
    - an indirection allowing native code reclamation
  - interceptor stubs
    - remoting, profiler, security, ...



# Synchronization via Monitors

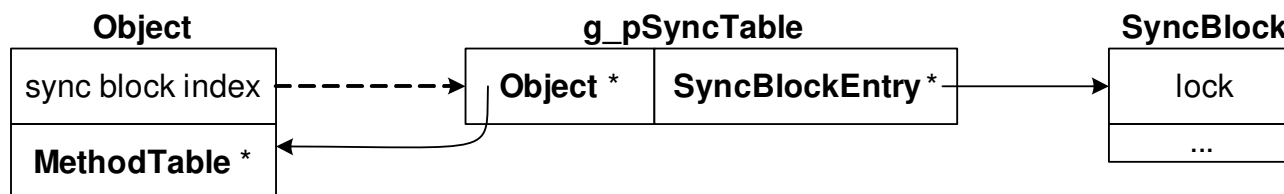
- class System.Threading.Monitor
  - statics: Enter, TryEnter, Exit, Wait, Pulse, PulseAll
  - all take an object treated as a monitor
- C# keyword lock

```
lock(obj) { /* critical section */ }
```

- is equivalent to

```
Monitor.Enter(obj);  
try  
{  
    /* critical section */  
}  
finally  
{  
    Monitor.Exit(obj);  
}
```

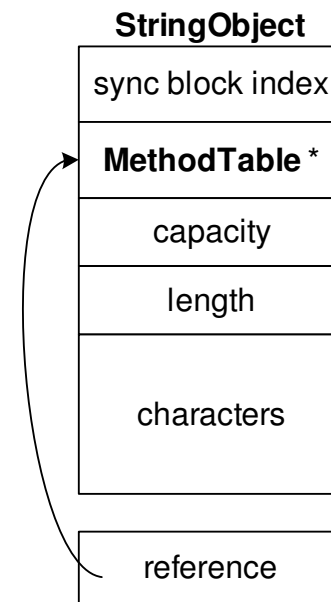
- lock is a part of a lazily allocated SyncBlock structure



# Strings

- two points of view on a string object
  - managed: `System.String` (subclass of `System.Object`)
  - EE: **StringObject** (subclass of **Object**)
- data internally ends with a `\0` character
  - addresses interoperability with native OS API
  - hidden from outside
  - a string can contain `\0` in the middle
- $\text{length} < \text{capacity} \leq 2 * \text{length}$ 
  - depends on the way how it is built up
- strings are immutable
  - more references can point to a single string object
  - conserves a space
  - time for space trade-off

Rotor:  
bcl/string.cs  
vm/object.h  
vm/comstring  
vm/stringliteralmap



# String Interning

- internal EE hash table
  - holds references to some strings
  - these strings are said to be *interned*
- implicit internining
  - all string literals interned during JIT compilation
- explicit internining
  - string `String.Intern(string)` method
    - adds a string to the internal hash table if not there yet
  - string `String.IsInterned(string)` method
    - returns interned string having the same value as the passed one
    - null if there is no such one
    - used for fast switching over strings in C# 1.0

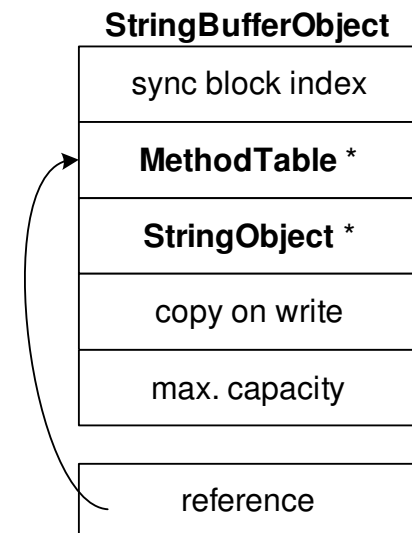
# String Literals

- stored in metadata stream one by one
- loaded by ldstr IL instruction
  - offset in the metadata stream is a part of the instruction
  - loads a reference to a string on evaluation stack
- when a method is being compiled
  - EE internal hash table is looked up for each string literal
  - not found
    - ⇒ a new one is allocated on managed heap
    - ⇒ its value is copied from metadata stream with possible conversion (endianness)
    - ⇒ the literal is interned

# String Builder

Rotor:  
bcl/stringbuilder.cs  
vm/object.h  
vm/comstringbuffer


- used for generating strings efficiently
- two points of view on a mutable string object
  - System.Text.StringBuilder (subclass of System.Object)
  - EE: **StringBufferObject** (subclass of **Object**)
- modifies a wrapped string
  - ensures it is not interned nor referenced by other objects
  - ⇒ constructor taking a string makes a copy
- conversion to immutable System.String
  - via ToString() method
  - doesn't do a copy unless
    - capacity > 2\*length (shortens the resulting string)
  - sets copy-on-write flag



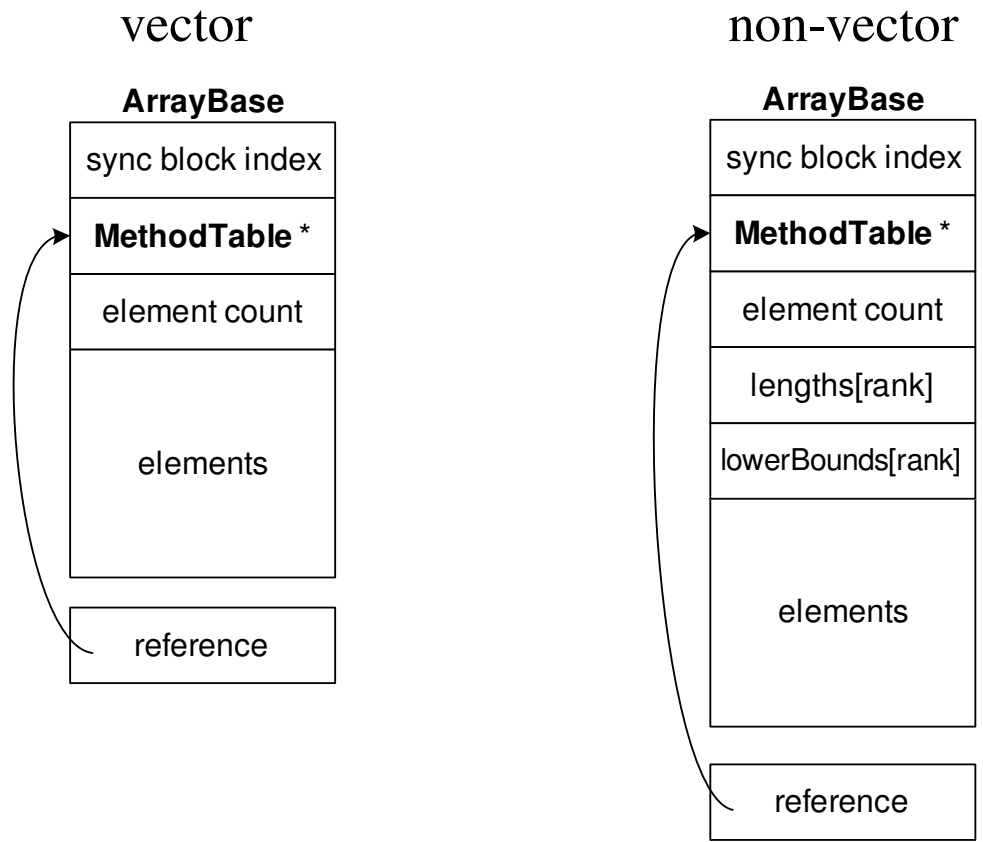
# Arrays

- base classes for arrays
  - managed: `System.Array` (subclass of `System.Object`)
  - EE: **ArrayBase** (subclass of **Object**)
- subclasses of `System.Array` class
  - dynamically generated types (**EEClass** structures)
  - differ in element types and/or ranks
  - generated methods: `Get(...)`, `Set(...)`, `Address(...)`
    - argument count = rank
    - each argument type = element type
- zero-based single-dimensional arrays (vectors)
  - instructions in IL (`newarr`, `ldelem`, `ldlen`, ...)
  - element access range checks
  - common patterns optimized by JITter

```
int[] a;  
for (int i = 0; i < a.Length; i++) a[i] = ...;
```
- other arrays (non-vectors)
  - accessed via generated methods

Rotor:  Arrays  
bcl/array.cs  
vm/object.h  
vm/class.h  
vm/array  
vm/comarrayinfo  
vm/comarrayhelpers

# Array Base Layouts



# Delegates: C# Level

- type-safe pointer to a method
- declaration defines target method signature

```
delegate void MyDelegate(string x);
```

- can “point” to both static and instance methods

```
class A
{
    static void f(string x) { }
    public virtual void g(string x) { }

    static void Foo(MyDelegate d)
    {
        if (d != null) d("bar"); // invokes the target method
    }

    static void Main(string[] args)
    {
        Foo(new MyDelegate(f));
        Foo(new MyDelegate(new A().g));
    }
}
```

# Delegates: MSIL Level

- delegate declaration turned into class declaration by the C# compiler

```
.class sealed MyDelegate extends [mscorlib]System.MulticastDelegate
{
    .method public instance
        void .ctor(object 'object', native int 'method') runtime managed
    {
    }

    .method public virtual instance
        void Invoke(string x) runtime managed
    {
    }

    .method public virtual instance class [mscorlib]System.IAsyncResult
        BeginInvoke(string x,
                    class [mscorlib]System.AsyncCallback callback,
                    object 'object') runtime managed
    {
    }

    .method public virtual instance
        void EndInvoke(class [mscorlib]System.IAsyncResult result) runtime managed
    {
    }
}
```

# Delegates: MSIL Level (cont.)

- delegate construction (static method)

```
ldnull
ldftn      void A::f(string)
newobj    instance void MyDelegate::.ctor(object, native int)
```

- delegate construction (virtual method)

```
// obtain a reference to target object
dup
ldvirtftn instance void A::g(string)
newobj    instance void MyDelegate::.ctor(object, native int)
```

- delegate invocation

```
// obtain a reference to the delegate
ldstr     "bar"
callvirt  instance void MyDelegate::Invoke(string)
```

# Delegates: EE Level

- `.ctor(object, native int)` implemented by `COMDelegate::DelegateConstruct`
  - the same method for all delegates
- special `Invoke` emitted for each delegate class
  - hardcore optimizations (`StubLinkerCPU::EmitMulticastInvoke`)

...

```
// Replace the pointer to a Delegate with the THIS pointer
```

```
static const BYTE replaceArg1[] = { 0x89, 0x4C, 0x24, 0x04 }; // mov [esp+4], ecx
```

```
EmitBytes(replaceArg1, sizeof(replaceArg1));
```

...

- if the delegate points to static method, arguments must be shuffled
  - `Invoke` called with 'this' – reference to the delegate instance
  - the stub must get rid of this extra parameter
  - `StubLinkerCPU::EmitShuffleThunk`
- see `DelegateEEClass`, `COMDelegate` classes in Rotor for details

# Delegates: Performance

