

Haladó fejlesztési technikák

DLL

Reflexió

Haladó fejlesztési technikák

DLL

Reflexió

Tipikus futtatható állományok

- **Binary executables**

- https://en.wikipedia.org/wiki/Comparison_of_executable_file_formats
- Ugyanazok a részek: Header, Imports, Data (RW/RO), Code Segment/.Text
- Container formátum: futtatható kód, és minden, a futtatáshoz szükséges adat

- **Linux: ELF = Executable and Linkable Format**

- Tipikusan kiterjesztés nélküli bináris futtatható állomány (nem script futtatható állomány!) vagy SO file
- Kiemelt extra feature: fatELF = több platformfüggő futtatható állomány egyetlen platformfüggetlen file-ban

- **Windows: PE = Portable Executable**

- Minden EXE/DLL file
- Kiemelt extra feature: ikon elhelyezése a file-ban (ELF-ben is: elfres), „egyszerűbb” (non-global) import namespaces
- <https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/corkami/PE101-v20L.png>

A legegyszerűbb „Hello world”

Hello1.asm

```
.MODEL small
```

```
.STACK 100h
```

```
.DATA
```

```
Message db 'Hello, World!',0Dh,0Ah,'$'
```

```
.CODE
```

```
main proc
```

```
    mov ax,@data
```

```
    mov ds,ax
```

```
    mov dx,OFFSET Message
```

```
    mov ah,9
```

```
    int 21h
```

```
    mov ah,4ch
```

```
    int 21h
```

```
main endp
```

```
END main
```

```
D:\TASM5\BIN>tasm hello1.asm
```

```
Turbo Assembler Version 4.1 Copyright (c) 1989 Intel Corp.
```

```
Assembling file:      hello1.asm
```

```
Error messages:      None
```

```
Warning messages:    None
```

```
Passes:               1
```

```
Remaining memory:    468k
```

```
D:\TASM5\BIN>tlink hello1.obj
```

```
Turbo Link Version 7.1.30.1. Copyright (c) 1989 Intel Corp.
```

```
D:\TASM5\BIN>hello1.exe
```

```
Hello, World!
```

Ugyanez, függvényhívással

```
.MODEL small
```

```
.STACK 100h
```

```
.DATA
```

```
Message db 'Hello, World!', 0Dh, 0Ah, '$'
```

```
.CODE
```

```
WriteMsg proc
```

```
    mov ax, @data
```

```
    mov ds, ax
```

```
    mov dx, OFFSET Message
```

```
    mov ah, 9
```

```
    int 21h
```

```
    ret
```

```
WriteMsg endp
```

```
main proc
```

```
    CALL WriteMsg
```

```
    mov ah, 4ch
```

```
    int 21h
```

```
main endp
```

```
END main
```

Hello2.asm

```
D:\TASM5\BIN>tasm hello2.asm
Turbo Assembler Version 4.1 C
```

```
Assembling file:    hello2.asm
```

```
Error messages:    None
```

```
Warning messages:  None
```

```
Passes:            1
```

```
Remaining memory:  468k
```

```
D:\TASM5\BIN>tlink hello2.obj
Turbo Link Version 7.1.30.1. C
```

```
D:\TASM5\BIN>hello2.exe
```

```
Hello, World!
```

Ha nincs minden függvény ugyanott

```
.MODEL small
.STACK 100h
PUBLIC WriteMsg
.DATA
Message db 'Hello, World!',0Dh,0Ah,'$'
.CODE
WriteMsg proc
```

Hello3a.asm

```
.MODEL small
.STACK 100h
EXTRN WriteMsg:PROC
.CODE
```

Hello3b.asm

```
main proc
    CALL WriteMsg           ;Call function
    mov ah,4ch              ;4Ch = terminate program
    int 21h                 ;API CALL
main endp
END main
```

Code, Compile, LINK – static linking!

```
D:\TASM5\BIN>tasm hello3a.asm
```

```
Turbo Assembler Version 4.1 Copyright (c) 1987, 1996
```

```
Assembling file:    hello3a.asm
```

```
Error messages:     None
```

```
D:\TASM5\BIN>tlink hello3a.obj
```

```
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996
```

```
Fatal: No program entry point
```

```
D:\TASM5\BIN>tlink hello3b.obj
```

```
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996
```

```
Error: Undefined symbol WRITEMSG in module HELLO3B.ASM
```

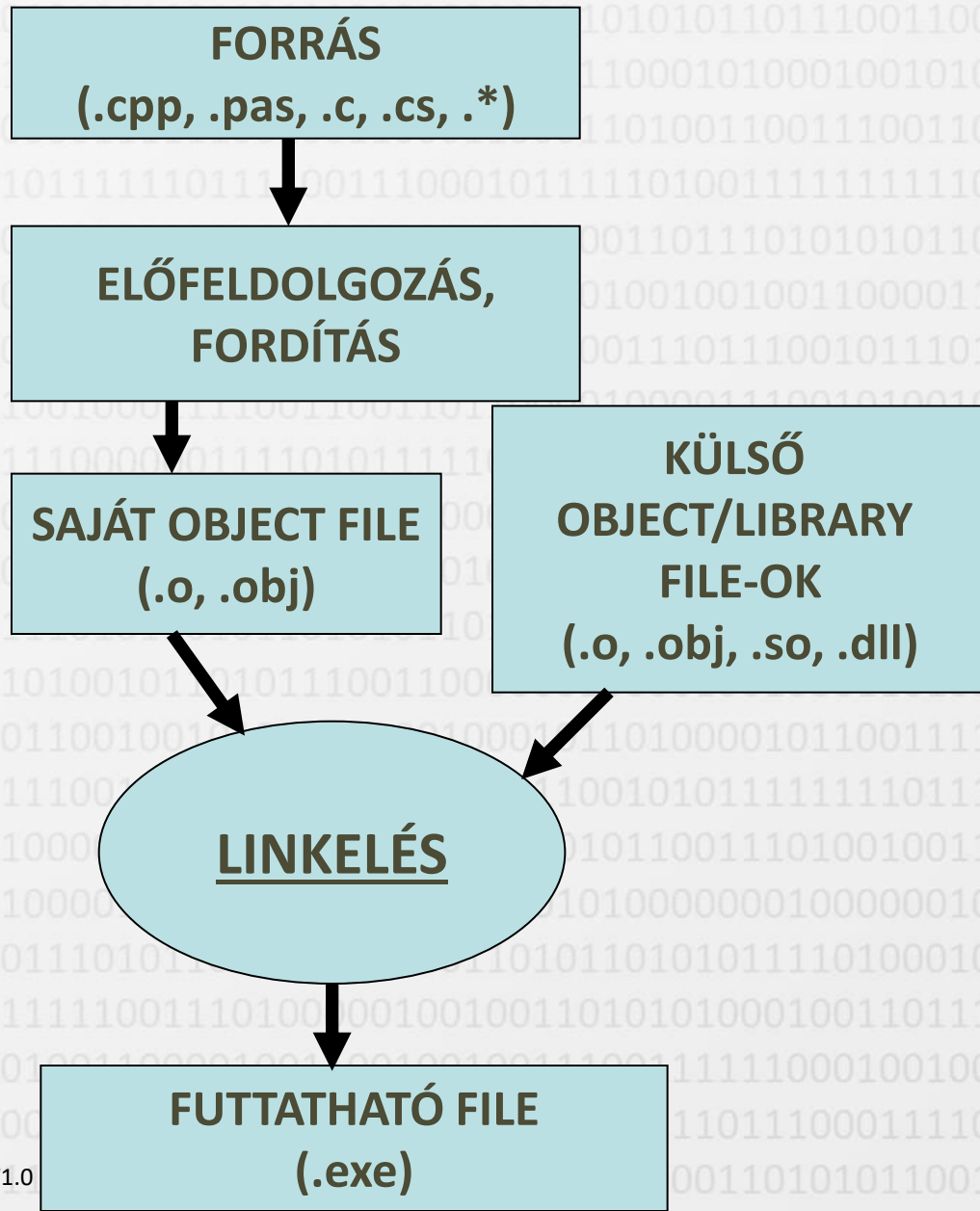
```
D:\TASM5\BIN>tlink hello3a.obj hello3b.obj, hello3.exe
```

```
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996
```

```
D:\TASM5\BIN>hello3.exe
```

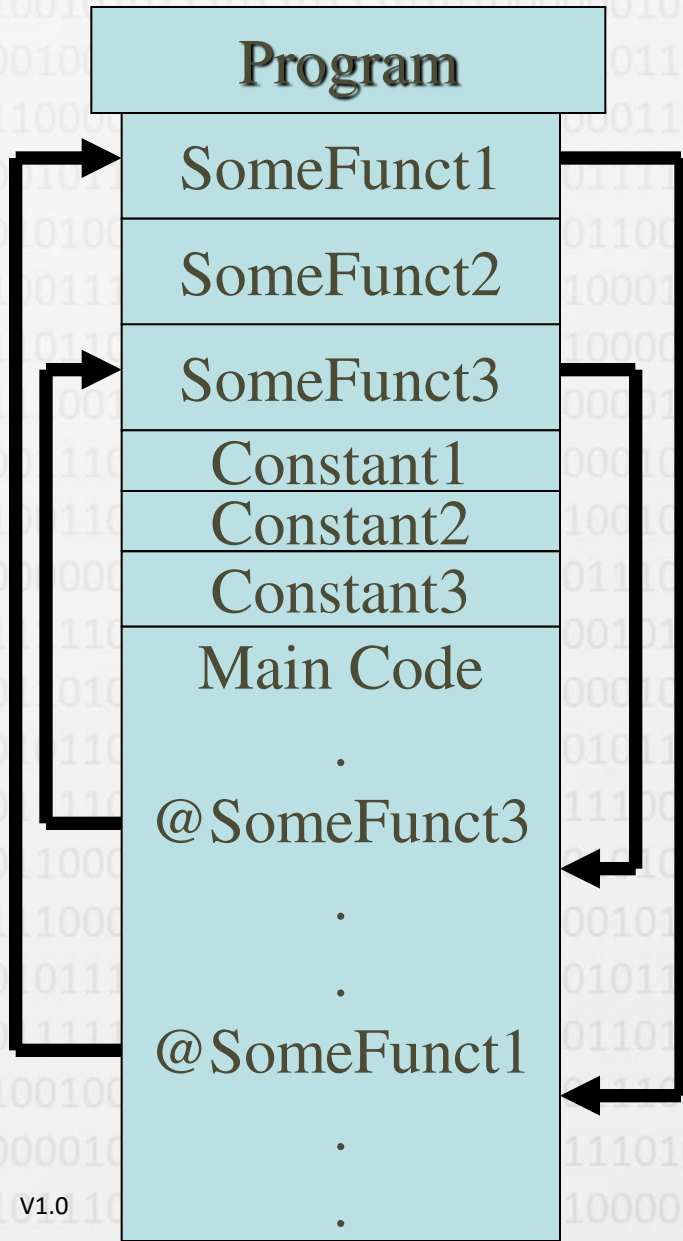
```
Hello, World!
```

Klasszikus fordítás



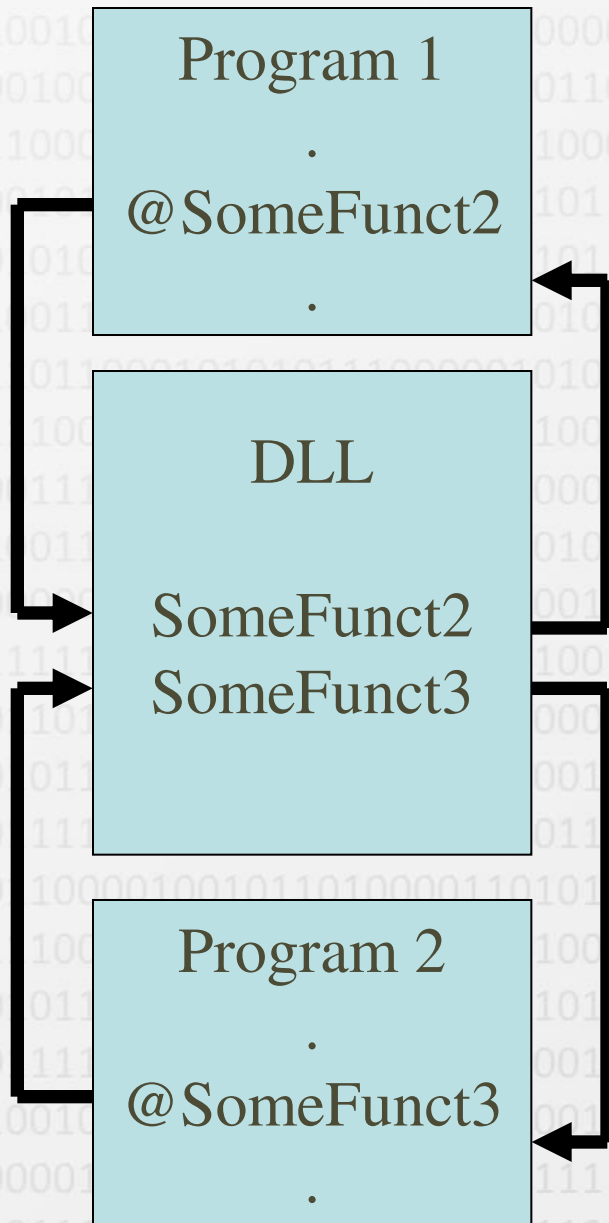
- **Object File:** Egy köztes kód-reprezentáció, a fordító generálja a forráskód lefordítása után
- **Tartalmazza:** az értelmezett, fordított kódot és relokációs adatokat, utóbbit a linker használja a futtatható állomány generálásához
- A külső library kód static linking esetén bekerül az EXE állományba; dynamic linking esetén nem

Static linking



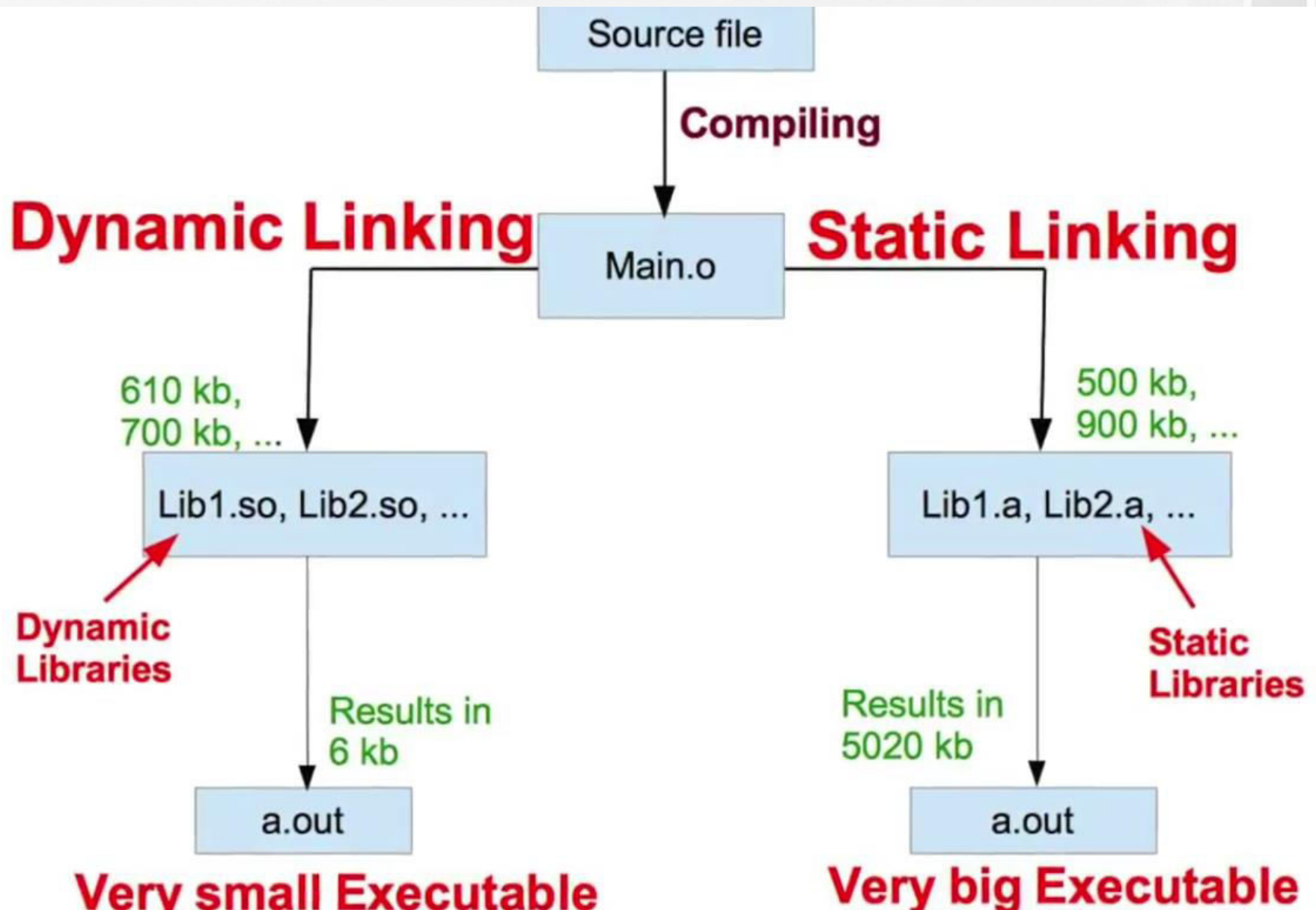
- Az összes használt függvény/erőforrás a programban van
- Ezek helye a fordítóprogram számára ismert
- Így fordításkor az ezekre történő hivatkozás előre megadható, mert az adott kód a program saját címtérületén található
- Ugyanazt a függvényt több program is külön-külön betölti → pazarlás
- A Klasszikus (nem overlay) DOS programok (pl: Turbo Pascal)

Dynamic linking



- Egyes függvények/erőforrások a program saját címtérületén kívül vannak
- Ezek helye a fordítóprogram számára nem ismert
- Így fordításkor az ezekre történő hivatkozás dinamikus, futási időben derül ki a pontos hely (a betöltést az OS végzi, Dynamically Linked)
- Ugyanazt a függvényt több program is együtt használja → megosztott erőforrás (Shared Object)
- A Legtöbb modern program így működik

Static vs Dynamic Linking



RDATA, DATA, CODE/TEXT

AFTER LOADING,

0X402068 WILL POINT TO KERNEL32.DLL'S **EXITPROCESS**

0X402070 WILL POINT TO USER32.DLL'S **MESSAGEBOXA**

STRINGS

a simple PE executable\0	0x403000
Hello world!\0	0x403017

X86 ASSEMBLY

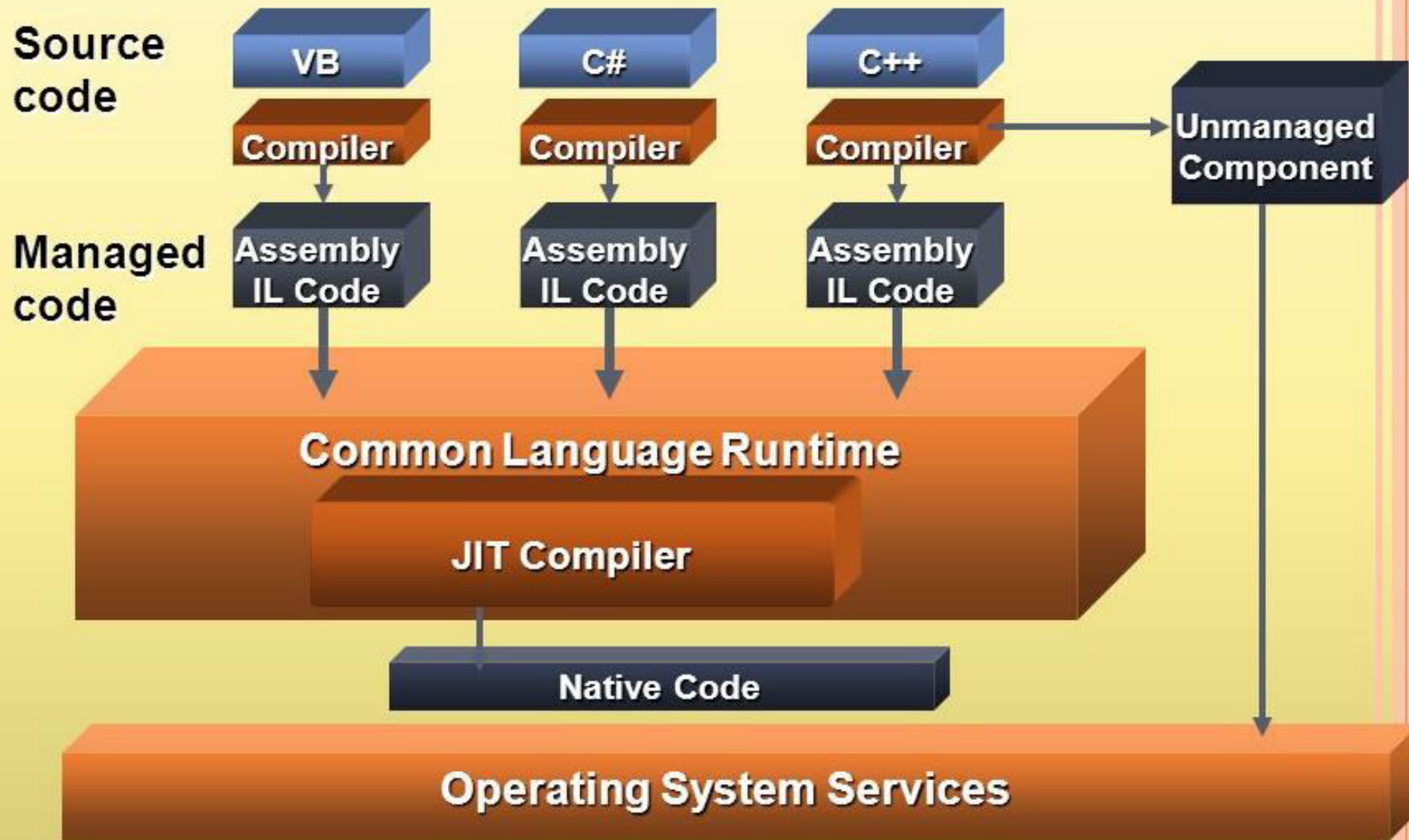
EQUIVALENT C CODE

```
push 0
push 0x403000
push 0x403017
push 0
call [0x402070] → MessageBox(0, "Hello world!", "a simple PE executable", 0);
push 0
call [0x402068] → ExitProcess(0);
```

Dynamic Link Library / Shared Object

- A mai operációs rendszerek felhasználó számára elérhető programjai/moduljai kizárólagosan így működnek (bár több nyelvben is lehetséges statikus fordítást kérni, nem illik)
- Azonos feladatkör: Windowsban DLL file, Linuxban: SO file
 - Külön fordított, a programhoz ténylegesen csak a futási időben kapcsolódik (= dynamic, mindkettőre igaz)
 - A memóriába csak egyszer töltődik be, több program is használhatja (= shared, mindkettőre igaz, de .NET alatt csak korlátozottan (AppDomain))
- Windows OS alatt meg kell különböztetni a natív és a felügyelt futtatható állományt:
 - Natív (*Native / Unmanaged*): OS/CPU számára értelmezhető bytekód: procedurális kód, egyszerű típusok, közvetlen HW elérés is akár. Nyelvfüggetlen (bármilyen fordítható ilyen EXE/DLL kóddá), de platformfüggő
 - Felügyelt (*Managed*): Egy (vagy több) osztály van benne, csak a .NET/JVM értelmező tud vele dolgozni. Nyelvfüggetlen és platformfüggetlen (de a .NET/JVM értelmező kell hozzá). .NET esetén DLL/EXE file, JVM esetén CLASS/JAR file

Managed/Unmanaged



Natív DLL-ek

- Az aktuális könyvtárból vagy a %PATH%-ból töltődik be
 - %PATH% = a WINDOWS, SYSTEM, SYSTEM32 könyvtárak
- Lassú betöltődés
- DLL HELL
 - Verziózás nem megoldott
 - Ugyanazon DLL-ből különféle alkalmazások különféle verziót igényelnek
 - A több különféle verziójú DLL együtt problémát okoz, törléskor különösen
 - „Megoldás”: *DLL stomping* VAGY minden DLL az EXE mellett...
 - Linux megoldás: file szintű csomagkezelő + verziózott symlinkek
 - .NET megoldás: GAC = Global Assembly Cache, ld. később
- Windows API
 - Unmanaged DLL-ek gyűjteménye, rendszer-hívásokkal
 - Alacsony szintű műveletvégzés, hardver-elérés
 - Az operációs rendszer minden publikus funkcionálitása elérhető
 - A fontosabbakhoz létezik .NET osztály, de a legalsó szinten akkor is a WINAPI hívódik (pl. System.Diagnostics.Stopwatch = QueryPerformanceCounter)

Natív DLL hívása

- **Platformfüggetlen natív kód**

- 32 / 64 bites (CPU szóhossz függő)
- OS függő

- **Nyelvfüggetlen?**

- Elvileg igen, de....
- Paraméterek átadása
- Eredmény visszaadása
- Ki mit szabadít fel?
- Lehetőségek: cdecl, stdcall, fastcall, ...
- https://en.wikipedia.org/wiki/X86_calling_conventions

- **Mi érhető el?**

- Saját programból nem dönthető el, fejlesztés közben ellenőrizendő
- dumpbin /exports

- **A felhasznált DLL függőségei**

- Saját programból nem dönthető el, fejlesztés közben ellenőrizendő
- Dependency Walker

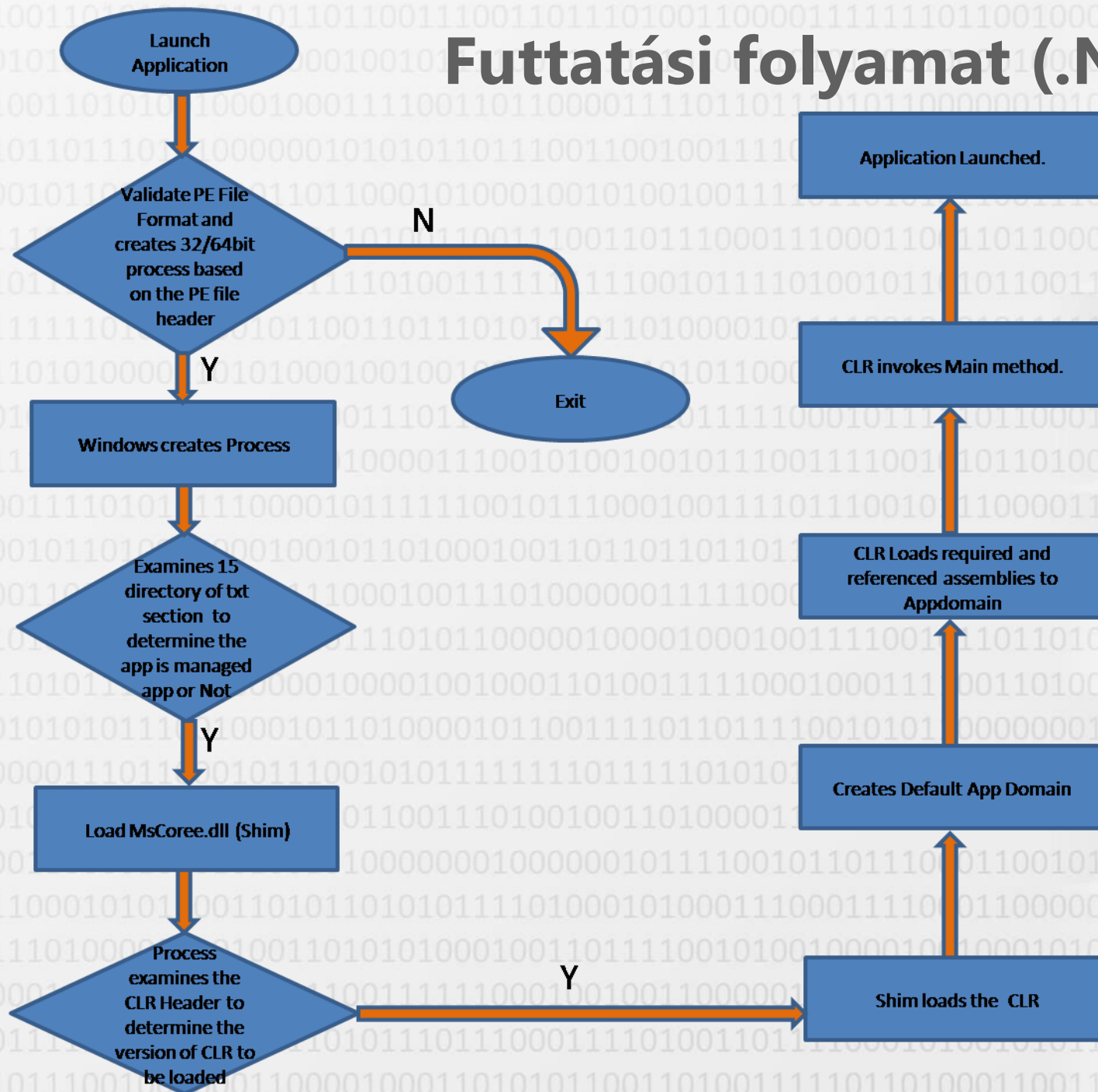
Natív DLL hívása C# nyelven

- A programhoz szigorúan csak futási időben csatolt
- Tényleges ellenőrzést (létezik –e a DLL, létezik –e a benne lévő entry point) a fordító nem végez
- Platform Invoke (P/Invoke: natív bytekód hívása felügyelt környezetből) → DllImport attribútum
 - `using System.Runtime.InteropServices;`
 - `[DllImport("winmm.dll", SetLastError = true)]`
`static extern bool PlaySound(string pszSound,`
`IntPtr hmod, uint fdwSound);`
 - `string fname = @"c:\Windows\Media\tada.wav";`
`PlaySound(fname, IntPtr.Zero, 1);`
- WINAPI Szignatúrák, importok: www.pinvoke.net

Felügyelt DLL-ek

- **Minden metódus egy DLL hívás volt, amit eddig használtunk**
 - Egy projekt „References” része tárolja azt, hogy mely felügyelt DLL-ek érhetőek el az adott projektből, ezt kell szerkeszteni
 - A fordító is ellenőrzi a DLL és az osztály/metódus meglétét
 - Gyors betöltődés, ugyanolyan sebességű, mintha a saját kód lenne
- **Felügyelt DLL hívása**
 - Referencia hozzáadása: Project/Add reference
 - Ezután a DLL-ben tárolt névtér és az abban tárolt osztályok/metódusok a szokványos módon elérhetőek
- **EXE vagy DLL?**
 - .NET-en belül nincs nagy különbség, mindkettő felügyelt osztályokat tartalmaz, ugyanolyan köztes kód formában
 - Az EXE klasszikus (PE) része csak a CLR értelmezőt tölti be
 - Az EXE file-on belül van egyetlen **static void/int Main()**
 - Projekttípus: Console App / WPF vagy Windows Forms / Class Library

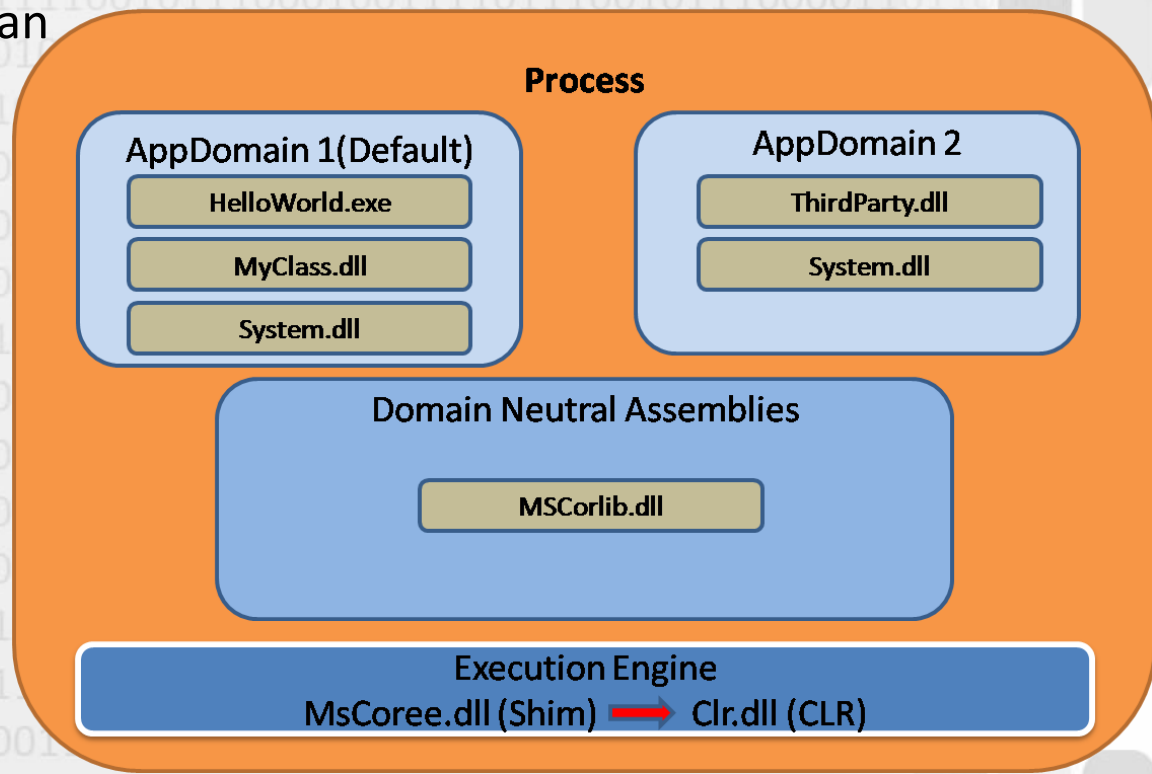
Futtatási folyamat (.NET)



Felügyelt Assembly - Sandboxing

- **AppDomain**

- A .NET assembly és az OS folyamat közötti réteg
- A kód végrehajtásának és az erőforrások elérésének szabályozása
- Pl. web esetén az azonos weboldalhoz (Virtual Directory-hoz) tartozó .NET alkalmazások azonos erőforrásokat érhetnek el, még akkor is, ha esetleg különböző .NET assembly-ről beszélünk... Különböző weboldalakhoz tartozó folyamatok különböző erőforrásokat érhetnek el, még akkor is, ha ugyanaz az EXE file fut több példányban
- Több AppDomain is létezhet egyetlen Win32 folyamaton belül
- Folyamaton belüli erőforrás-leválasztás válik lehetségessé
➔ sandboxing



Felügyelt DLL betöltése

- **Fusion**

- A .NET-en belüli modul, amely a felügyelt DLL-ek betöltését végzi
- „Assembly binding”: felügyelt Assembly futtatásakor a references részben hivatkozott külső Assembly-k megkeresését és betöltését végző művelet
- Naplózás engedélyezése: fuslogvw.exe / Registry bejegyzések

```
*** Assembly Binder Log Entry (9/23/2013 @ 5:25:37 PM) ***
```

```
The operation was successful.
```

```
Bind result: hr = 0x0. The operation completed successfully.
```

```
Assembly manager loaded from: C:\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll
```

```
Running under executable C:\Users\ \AppData\Local\Apps\2.0\CCW29YW.39L\HAKH6L9.ETR\getc..tion_25403a3e
```

```
--- A detailed error log follows.
```

```
=== Pre-bind state information ===
```

```
LOG: DisplayName = System.Xml, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

```
(Fully-specified)
```

```
LOG: Appbase = file:///C:/Users/ /AppData/Local/Apps/2.0/CCW29YW.39L/HAKH6L9.ETR/getc..tion_25403a3e
```

```
LOG: Initial PrivatePath = NULL
```

```
LOG: Dynamic Base = NULL
```

```
LOG: Cache Base = NULL
```

```
LOG: AppName = GetTime.exe
```

```
Calling assembly : (Unknown).
```

```
====
```

```
LOG: This bind starts in default load context.
```

```
LOG: No application configuration file found.
```

```
LOG: Using host configuration file:
```

```
LOG: Using machine configuration file from C:\Windows\Microsoft.NET\Framework\v4.0.30319\config\machine.config
```

```
LOG: Found assembly by looking in the GAC.
```

```
LOG: Binding succeeds. Returns assembly from C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xml\v4.0_4.0.
```

```
LOG: Assembly is loaded in default load context.
```

Eszközök

- **gacutil.exe**

- DLL regisztrálása / törlése a GAC-ból, itt vannak a hivatalos .NET DLL-ek is
- Kezelhetőek a különféle verziók és függőségek
- MSDN-en megtalálható, hogy melyik osztály/névtér melyik DLL-ben található

- **NuGet**

- Központi .NET csomagkezelő, tipikusan felügyelt DLL állományokhoz
- Tools/NuGet Package Manager/Manage NuGet Packages for Solution
- Powershell parancssori felületből is kezelhető (Install-Package)
- Szinte az összes C# library/tool letölthető
- Függőségek/frissítések kezelése
- Consolidate: ellentmondó verziók kezelése

- **Dotpeek (ILDasm, Reflector ...)**

- .NET DLL/EXE állományok kódjának tanulmányozását engedik meg
- Reflexió segítségével is kinyerhető információk megjelenítése (*később*)
- C# kód visszafejtése (többnyire használható formában, kivéve ha Code Obfuscator-t használunk)

Managed Assembly tartalma

- **Assembly ~ Szerelvény ~ Futtatási egység ~ managed .NET EXE/DLL állomány (Semmi köze az assembly nyelvhez!)**
- **Assembly Manifest/Metadata**
 - Név
 - Verzió
 - Culture/Localization információk
 - Belső file/erőforrás lista
 - Type metadata
 - Referencia-lista
- **Type metadata**
 - Minden információ a típusokról
 - Reflexióval kinyerhető (*később*)
- **IL/CLR kód**
- **Erőforrások**

Haladó fejlesztési technikák

DLL

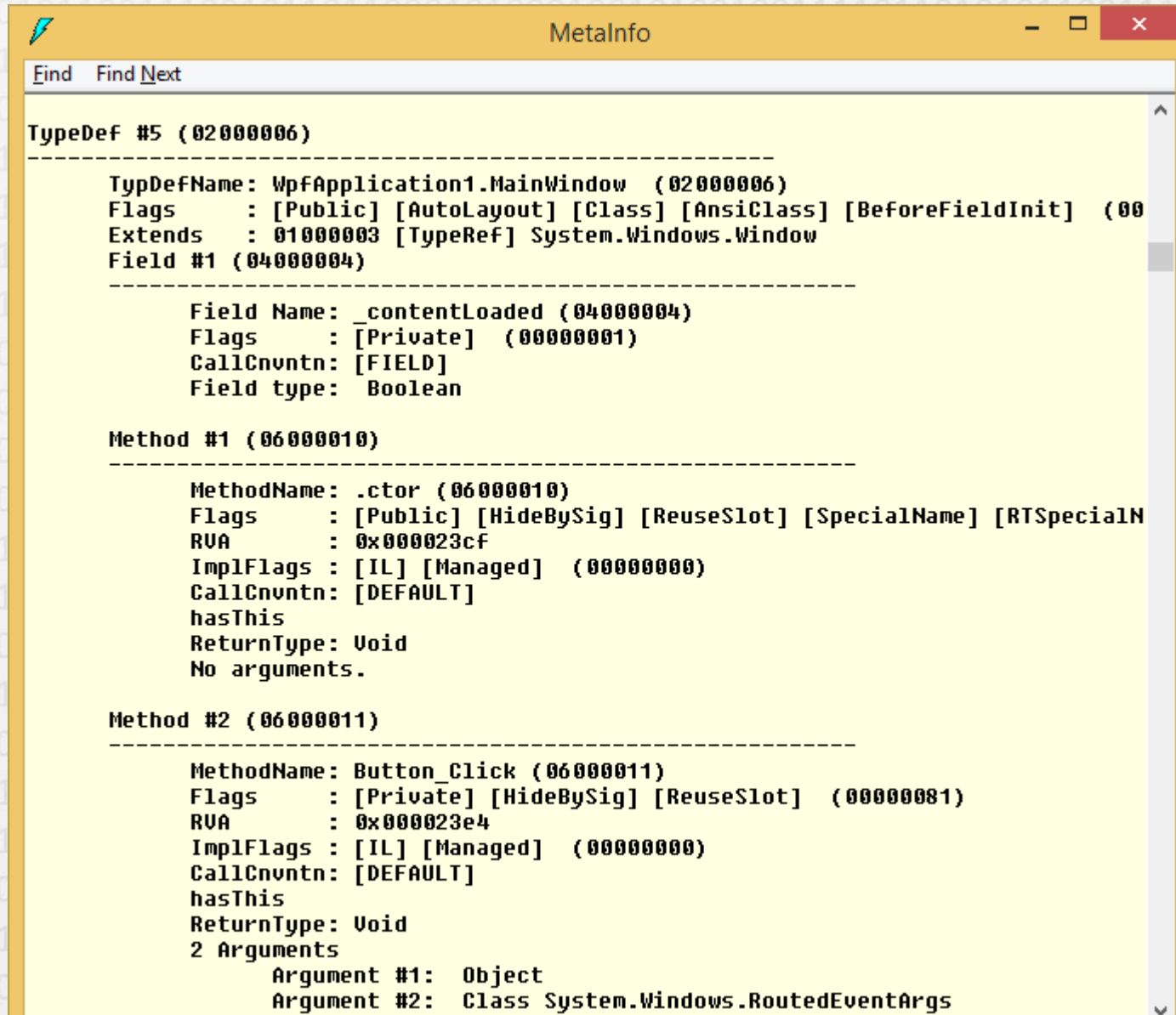
Reflexió

Reflexió

- **A reflexió az a képesség, amellyel a program önmaga struktúráját és viselkedését futásidőben analizálni és alakítani tudja**
 - Magas szintű nyelv kell hozzá (Java, PHP, ... C#)
 - Különböző mértékű támogatás a nyelvekben
- **C#-ban a leggyakoribb használati módja a futásidőben végrehajtott típusanalízis**
 - De lehetséges típusok futásidejű létrehozása is: `System.Reflection.Emit`
- **Több technológia használja**
 - Intellisense, Properties és más IDE-szolgáltatások
 - Több .NET technológia (szerializáció, .NET Remoting, WCF)
 - Tesztek

Metaadat

- Visual Studio Command Prompt / Ildasm.exe, Ctrl+M



The screenshot shows a window titled "MetalInfo" with a yellow header bar. Below the header is a search bar with "Find" and "Find Next" buttons. The main content area displays the following metadata:

```
TypeDef #5 (02000006)
-----
TypeDefName: WpfApplication1.MainWindow (02000006)
Flags       : [Public] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit] (00
Extends     : 01000003 [TypeRef] System.Windows.Window
Field #1 (04000004)
-----
Field Name: _contentLoaded (04000004)
Flags      : [Private] (00000001)
CallConvtn: [FIELD]
Field type: Boolean

Method #1 (06000010)
-----
MethodName: .ctor (06000010)
Flags      : [Public] [HideBySig] [ReuseSlot] [SpecialName] [RTSpecialN
RVA        : 0x0000023cf
ImplFlags  : [IL] [Managed] (00000000)
CallConvtn: [DEFAULT]
hasThis
ReturnType: Void
No arguments.

Method #2 (06000011)
-----
MethodName: Button_Click (06000011)
Flags      : [Private] [HideBySig] [ReuseSlot] (00000001)
RVA        : 0x0000023e4
ImplFlags  : [IL] [Managed] (00000000)
CallConvtn: [DEFAULT]
hasThis
ReturnType: Void
2 Arguments
Argument #1: Object
Argument #2: Class System.Windows.RoutedEventArgs
```

Futásidejű típusanalízis - Assembly

- **Assembly a = Assembly.GetExecutingAssembly();**
- **Assembly a = Assembly.LoadFrom(„Path.To.Assembly”);**
- **Assembly a = Assembly.Load(bytes);**
- **Assembly a = type.Assembly;**
- **a.GetTypes()** – típusok kinyerése → eredmény: Type[]
- **a.EntryPoint** – belépési pont (metódust ad vissza, csak exe file-ok esetén)

Futásidejű típusanalízis - Type

- **Type t = assembly.GetType(„Type.Name.In.Assembly”);**
- **Type t = typeof(int);**
- **Type t = typeof(T);**
- **Type t = obj.GetType();**
- **Type t = Type.GetType(„Type.Name.In.Any.Assembly”);**
 - Ha nem az aktuálisan végrehajtódó szerelvényben vagy az mscorlib.dll-ben van, akkor ún. „assembly-qualified name” megadása szükséges
- **t.FullName, t.AssemblyQualifiedName** – nevek különféle formában
- **t.BaseType, t.IsSubclassOf(anotherType), t.IsAssignableFrom(anotherType)** – ős, utód vizsgálat

Futásidejű típusanalízis – xxxInfo

- `PropertyInfo pi = t.GetProperty("PropName");`
- `PropertyInfo[] pis = t.GetProperties();`
- `FieldInfo fi = t.GetField("FieldName");`
- `FieldInfo[] fis = t.GetFields();`
- `MethodInfo mi = t.GetMethod("MethodName");`
- `MethodInfo mis = t.GetMethods();`
- Általában átadható `BindingFlags` paraméter, amivel a keresés szűkíthető/konfigurálható
- `PropertyInfo pi = t.GetProperty("PropName", BindingFlags.Static | BindingFlags.NonPublic)`
 - Nem publikus (privát) tagok is megkaphatók
 - Ez (elsősorban) nem arra való, hogy kijátsszuk a láthatóságokat!

Reflektált kódelemek használata futásidőben

- A reflexióval elért típusok/tagok futásidőben felhasználhatóak
- `List<int> something = new List<int>();`
`something.Add(8);`
`int cnt = something.Count;`

```
Type listType = typeof(List<int>);  
MethodInfo addMethod = listType.GetMethod("Add");  
PropertyInfo countProperty = listType.GetProperty("Count");  
  
object listInstance = Activator.CreateInstance(listType);  
  
object methodResult = addMethod.Invoke(listInstance,  
                                         new object[] { 8 });           // null  
object propertyResult = countProperty.GetValue(listInstance); // 1
```

- Lassabb, mint a nem reflexív kód → csak akkor, ha máshogy nem megoldható (pl. egy metódus paramétere „teljesen ismeretlen”, aminek meghívjuk az Add metódusát / Count tulajdonságát)
- Hasonlóan flexibilis kód, sokkal gyorsabb, de compiler/intellisense segítség nélkül → dynamic (DLR)

Attribútumok

- **A fordító által generált metaadat mellé saját metaadat is felvehető**
 - Szerelvény, típus vagy tagok esetében is
- **System.Attribute osztály utódai**
 - Léteznek beépített attribútumtípusok különféle célokra, vagy saját Attribute utód is létrehozható
- **Használata speciális formában történik**
 - Névtér, osztály, metódus, tulajdonság, mező stb. fölött – attribútumtól függően ahol engedélyezett
 - Formátuma: [XXX], ha az Attribute utódosztály neve XXXAttribute

```
[Obsolete("Do not use this method, use the NewMethod() instead.")]
```

```
static void OldMethod()
```

```
{ }
```

```
static void NewMethod() { }
```

```
static void Main(string[] args)
```

```
{
```

```
    OldMethod();    //Warningot eredményez
```

```
}
```

Tipikus használati esetek

- **Az attribútum a jelölt tag szokványos használatát nem befolyásolja**
 - Minden metódus, tulajdonság, stb. ugyanúgy meghívható/elérhető
- **Kell egy „másik fél”, ami az attribútum meglétét majd reflexióval figyeli, és attól függően hajt végre lépéseket**
- **Tipikus felhasználása: automatizmusok/ellenőrzések**
 - Másik programozó segítése: Obsolete, DisplayName, Description
 - Visual Studio, debugger viselkedése: DebuggerDisplay, DebuggerStepThrough
 - Visual Studio, automata generálás: WebMethod, ServiceContract, OperationContract, FaultContract, DataContract, DataMember
 - Kód használhatósága: Serializable, Flags, ThreadStatic, DllImport
 - Automatizmusok támogatása: TestFixture, TestCase, TestCaseSource, Key, ForeignKey, Column
 - Egyéb (saját) metaadatok: saját attribútumokkal

Attribútumok

- **Példa: CallerMemberName**

- Ha a paraméter nem kap értéket, akkor a hívó neve kerül bele

```
protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

```
class Settings
{
    private string someSetting;
    public string SomeSetting
    {
        get { return someSetting; }
        set { someSetting = value; OnPropertyChanged(); }
    }
}
```

Attribútumok

- **Példa: Szerializáció** (=adat tárolható formába alakítása, a példában bináris)

```
[Serializable]
class Settings
{
    public string Setting1 { get; set; }
    public int Setting2 { get; set; }
    [NonSerialized]
    private int temp;
}

class Program
{
    static void Main(string[] args)
    {
        Settings settings = new Settings();
        //...
        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream stream =
            new FileStream("settings.dat", FileMode.Create))
        {
            formatter.Serialize(stream, settings);
        }
    }
} //mentett információ visszaolvasása: FileMode.Open, Deserialize()
```

Saját attribútum

- Saját attribútum létrehozása

```
[AttributeUsage(AttributeTargets.Property)]  
class HelpAttribute : Attribute  
{  
    public string HelpURL { get; private set; }  
  
    public HelpAttribute( string helpURL)  
    {  
        this.HelpURL = helpURL;  
    }  
}
```

```
[Help("http://path.to.my.help.for.setting1.html")]  
public string Setting1 { get; set; }
```

- Attribútum elérése reflexióval

- Az eddig említett módszerek is ezt használják

```
//PropertyInfo propertyInfo = typeof(Settings).GetProperty("Setting1");  
HelpAttribute helpAttribute =  
    propertyInfo.GetCustomAttribute<HelpAttribute>();  
  
Console.WriteLine(helpAttribute.HelpURL);
```

Annotációk

- Más nyelvekben (Java/PHP) hasonló célú nyelvi elem
- PHP
 - Teljesen a kikommentezett részben van
 - Tipikusan az IDE/külső eszköz számára értelmezhető
 - Futás közben nem használható
- Java
 - Fordító számára értelmezett
 - A fordított osztályokban is megmarad
 - Futás közben is kiolvasható: https://en.wikipedia.org/wiki/Java_annotation

```
class Foo
{
    /**
     * @var integer
     * @range(0, 100)
     * @label('Number of Bars')
     */
    public $bar;
}
```

```
@Override
public String toString(){
    return "Accounts: " + acc
```

Példa - XmlSerializer

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]  
class ExcludeFromXmlAttribute : Attribute  
{  
    public string Reason { get; set; }  
}
```

```
class Person  
{  
    [DisplayName("Személynév")]  
    public string Name { get; set; }  
  
    [DisplayName("E-Mail cím")]  
    public string Email { get; set; }  
  
    [DisplayName("Életkor")]  
    public int Age { get; set; }  
  
    [DisplayName("Lakcím")]  
    [ExcludeFromXml(Reason = "Top Secret")]  
    public string Address { get; set; }  
  
    [DisplayName("Születési dátum")]  
    public DateTime BirthDate { get; set; }  
}
```

Példa - XmlSerializer

```
class XmlBuilder
{
    string GetPrettyName(PropertyInfo property)
    {
        var attr = property.GetCustomAttribute<DisplayNameAttribute>();
        return attr == null ? property.Name : attr.DisplayName;
    }
    bool IsAllowed(PropertyInfo property)
    {
        return property.GetCustomAttribute<ExcludeFromXmlAttribute>() == null;
    }
}
```

Példa - XmlSerializer

```
<instance typeName="Lecture_XmlSerializer.Person">
  <data name="Name" prettyName="Személynév">Béla</data>
  <data name="Email" prettyName="E-Mail cím">bel@bela.hu</data>
  <data name="Age" prettyName="Életkor">42</data>
  <data name="BirthDate" prettyName="Születési dátum">1986. 11. 27. 14:13:24</data>
</instance>
```

```
}
Type type = instance.GetType();
XElement node = new XElement("instance");
node.Add(new XAttribute("typeName", type.FullName));
foreach (PropertyInfo property in type.GetProperties())
{
    if (IsAllowed(property))
    {
        XElement dataNode = new XElement("data");
        dataNode.Add(new XAttribute("name", property.Name));
        dataNode.Add(new XAttribute("prettyName", GetPrettyName(property)));
        dataNode.Value = property.GetValue(instance).ToString();
        node.Add(dataNode);
    }
}
return node;
}
```

Példa - XmlSerializer

```
class Program
{
    static void Main(string[] args)
    {
        Person person = new Person() { Name = "Béla",
            Age = 42,
            Address = "Bélavár 42",
            BirthDate = DateTime.Now.AddDays(-12345),
            Email = "bela@bela.hu" };
        var product = new { Name = "Something",
            Price = 12345, Quantity = 42 };
        XmlBuilder builder = new XmlBuilder();
        XElement personXml = builder.ToXml(person);
        XElement productXml = builder.ToXml(product);
        Console.WriteLine(personXml);
    }
}
```

```
<instance typeName="Lecture_XmlSerializer.Person">
  <data name="Name" prettyName="Személynév">Béla</data>
  <data name="Email" prettyName="E-Mail cím">bela@bela.hu</data>
  <data name="Age" prettyName="Életkor">42</data>
  <data name="BirthDate" prettyName="Születési dátum">1986. 11. 27. 14:13:24</data>
</instance>
<instance typeName="&lt;&gt;f__AnonymousType0`3[[System.String, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e],[System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e],[System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e]]">
  <data name="Name" prettyName="Name">Something</data>
  <data name="Price" prettyName="Price">12345</data>
  <data name="Quantity" prettyName="Quantity">42</data>
</instance>
```


Példa - Sorbarendezés név szerint

```
List<object> objects = new List<object>() { product, person };
objects.Sort(new NameComparer());
foreach (object item in objects)
{
    Console.WriteLine(item.GetType().
        GetProperty("Name").GetValue(item)?.ToString());
}
```

```
class NameComparer : IComparer<object>
{
    public int Compare(object x, object y)
    {
        string name1 = x.GetType().
            GetProperty("Name").GetValue(x)?.ToString();
        string name2 = y.GetType().
            GetProperty("Name").GetValue(y)?.ToString();
        return name1.CompareTo(name2);
    }
}
```

Példa - Sorbarendezés név szerint

```
class NameComparer : IComparer<object>
{
    public int Compare(dynamic x, dynamic y)
    {
        return x.Name.CompareTo(y.Name);
    }
}
```

```
List<object> objects = new List<object>() { product, person };
objects.Sort(new NameComparer());
foreach (dynamic item in objects)
{
    Console.WriteLine(item.Name);
}
```

Feladat / Reflection

- Hozzon létre egy osztályt, amely képes egy tetszőleges példány tetszőleges szabályok szerinti validálására
- A megoldás során használjon reflexiót
 - A RangeAttribute segítségével egy tulajdonság minimum és maximum értékét lehessen beállítani
 - A MaxLengthAttribute segítségével egy tulajdonság maximum hosszát lehessen beállítani
 - Az ezekhez illő MaxLengthValidation és RangeValidation osztályok végzik el a tényleges ellenőrzést. Mindkét osztály implementálja az IValidation interfészt, és a validációt egy **Validate(xxx)** metóduson keresztül végezzék el
 - A ValidationFactory osztály felelős egy megadott attribútumhoz a megfelelő validátor osztály létrehozásáért
 - A Validator osztály **public bool Validate(object instance)** metódusa végzi a tényleges ellenőrzést. A paraméterül kapott példány tulajdonságait megjelölő attribútumokra a Factory segítségével kérje le az ellenőrzést elvégző példányt, és annak a **Validate(xxx)** metódusa segítségével futtassa le a példányra az összes ellenőrzést

SOLID elvek

- **S = Single Responsibility**

