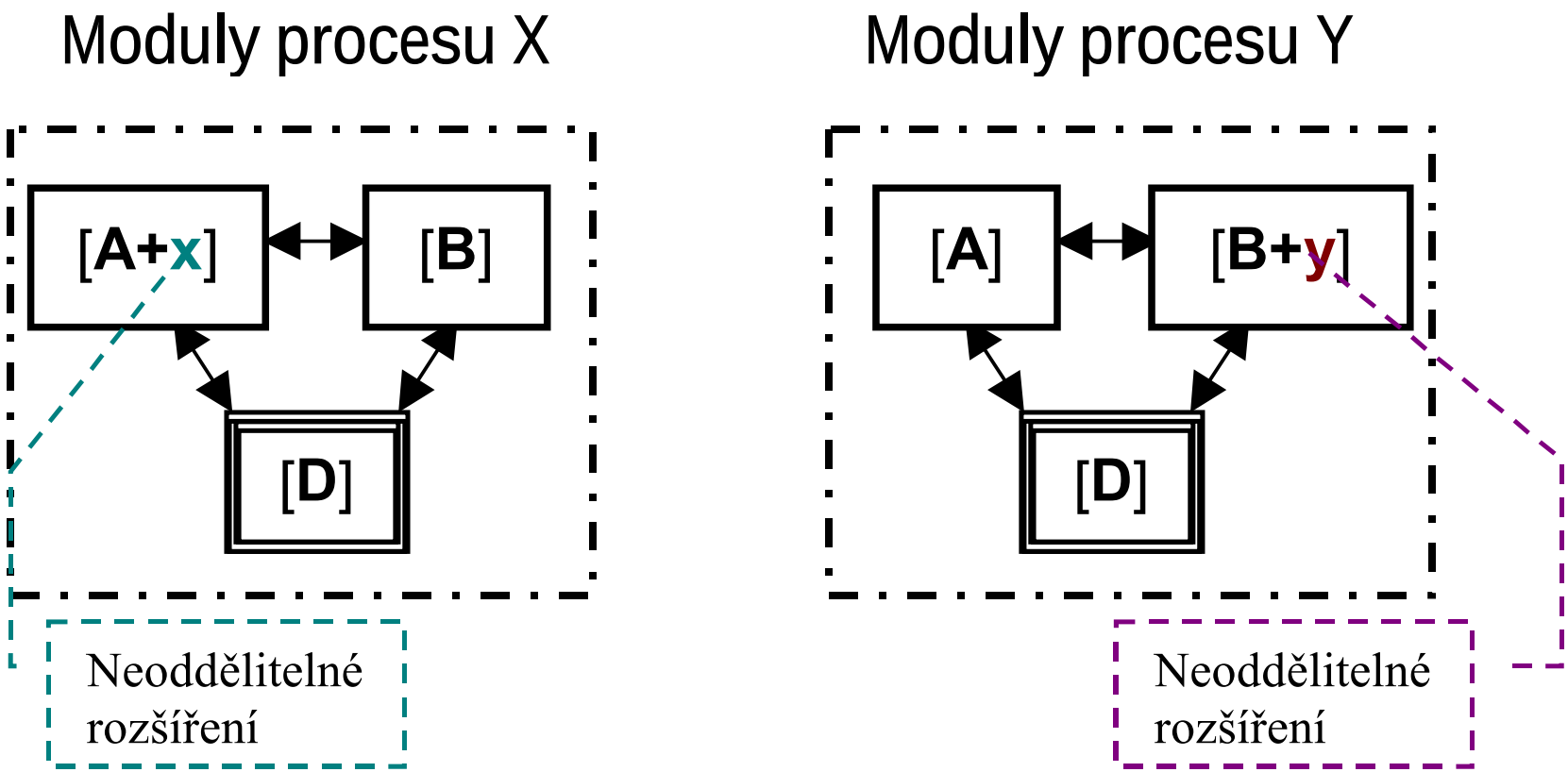
Vývoj aplikací v prostředí .NET

© Katedra řídicí techniky,
ČVUT-FEL Praha

Přednáška – 8. týden

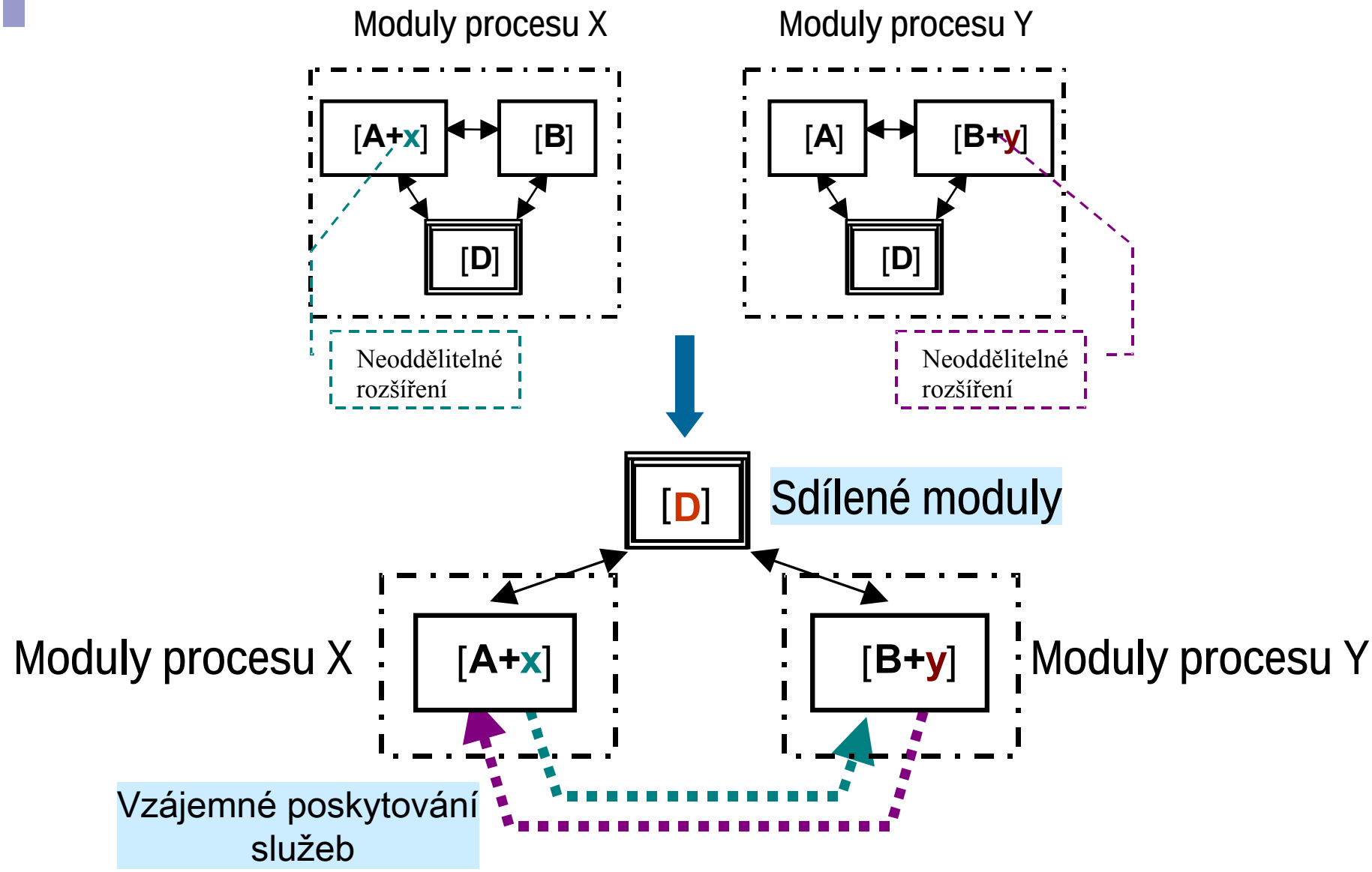
Software Shortage

Problém dnešní doby



Izolované aplikace → opakované řešení stejných problémů

Vývoj aplikací se urychluje sdílením částí kódu a služeb



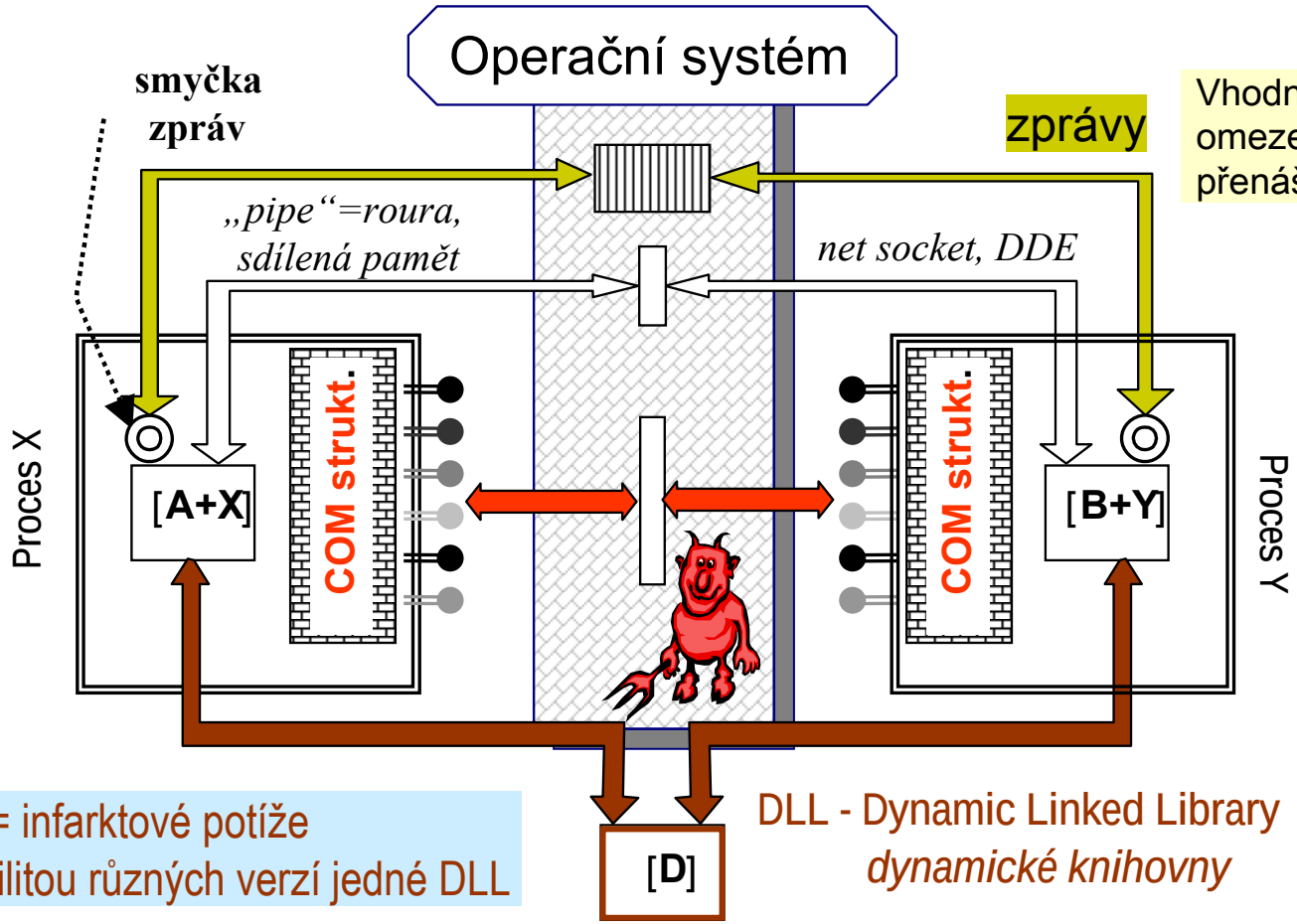
Procesy, základ OS, ale překážka sdílení...

- procesy

- operační systém

Operační systém sice bezpečně izoluje jednotlivé procesy, ale ztěžuje tím vzájemné poskytování služeb

Některé metody komunikace mezi procesy



Vhodná metoda pro omezené množství přenášené informace

- „DLL hell“ = infarktové potíže s kompatibilitou různých verzí jedné DLL

DLL - Dynamic Linked Library
dynamické knihovny

- OLE (Object Linked Exchange) vycházející z COM (Component Objekt Model) žádá složitou infrastrukturu a instalaci komponent se zápisy do registrů
  → obtížná manipulace i ztíženo korektní odinstalování

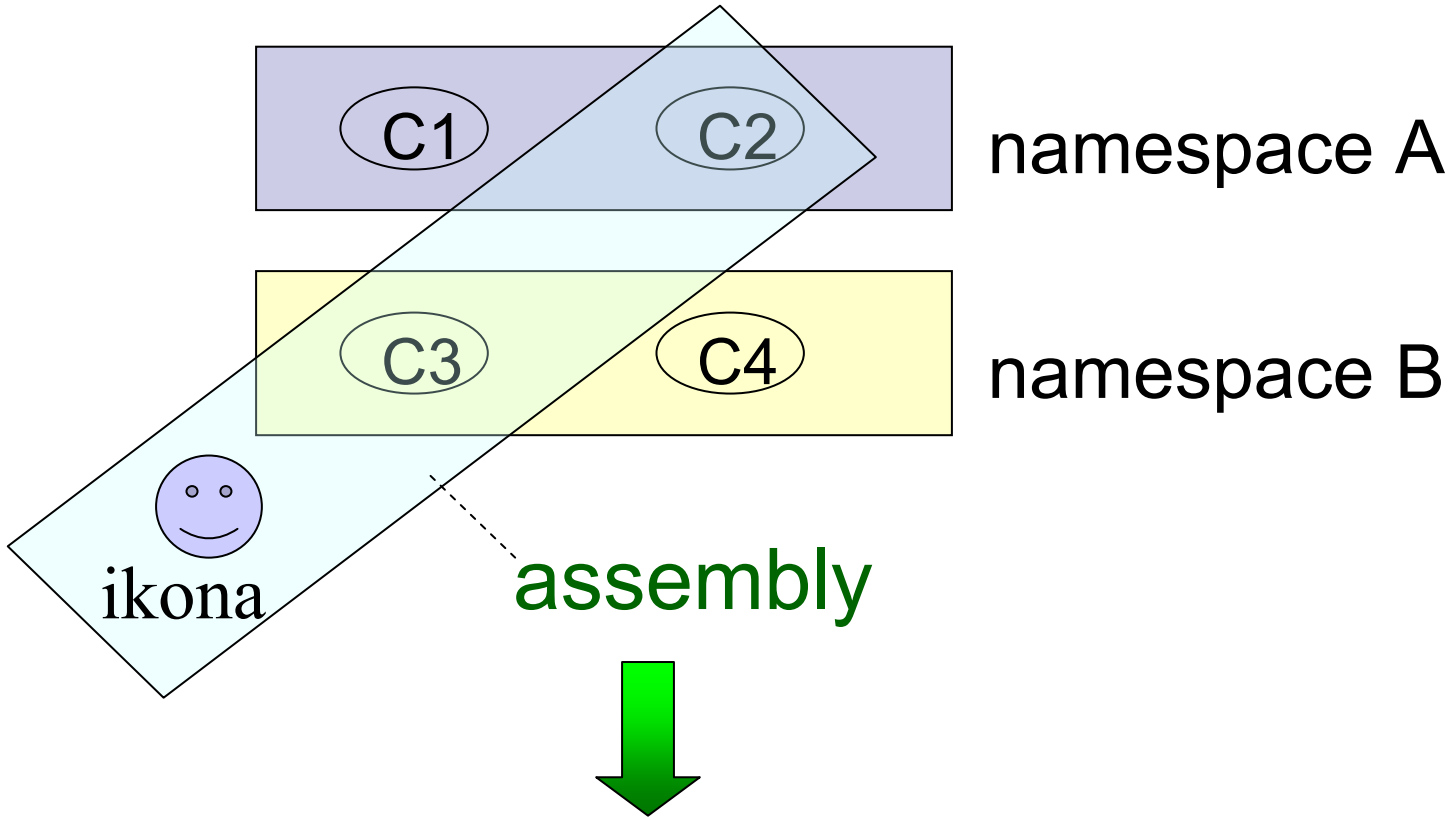
NET.Framework

- Přináší „runtime type info“ techniky reflexe, což umožňuje sdílení kódu jednodušeji než pomocí OLE a přitom efektivněji a se stejnou bezpečností v aplikačních doménách .
- Dovoluje používat několik verzí téže DLL knihovny, jejichž originálnost si lze pojistit otiskem privátního šifrovacího klíče
  → tím se řeší „DLL hell“

Assembly C# EXE i DLL

[Rob Gonsalves]

Assembly

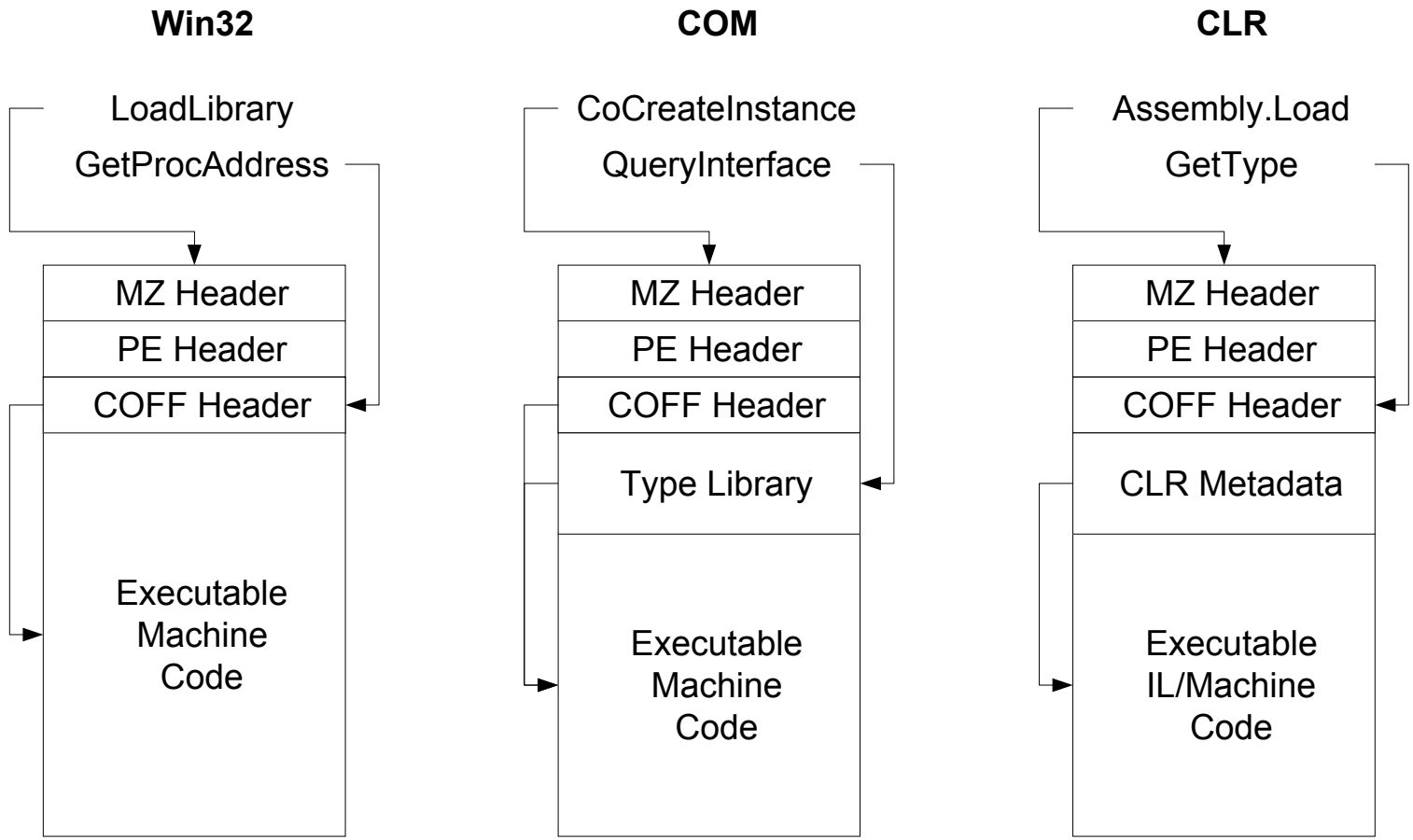


Nejmenší jednotka pro sestavení, pro ověřování verze i pro spuštění

Užití assembly

- **Nejčastěji:**
  1 assembly = 1 namespace = 1 program
- avšak:
  1 assembly smí obsahovat i **několik namespace**
- **1 namespace** lze definovat **v různých assembly**
- **1 assembly** lze rozprostřít do **několika souborů** spojených společným manifestem
- **Assembly**
  ≈ JAR soubor v Java
  ≈ komponenta v .NET (component)

Tabulky v programových modulech

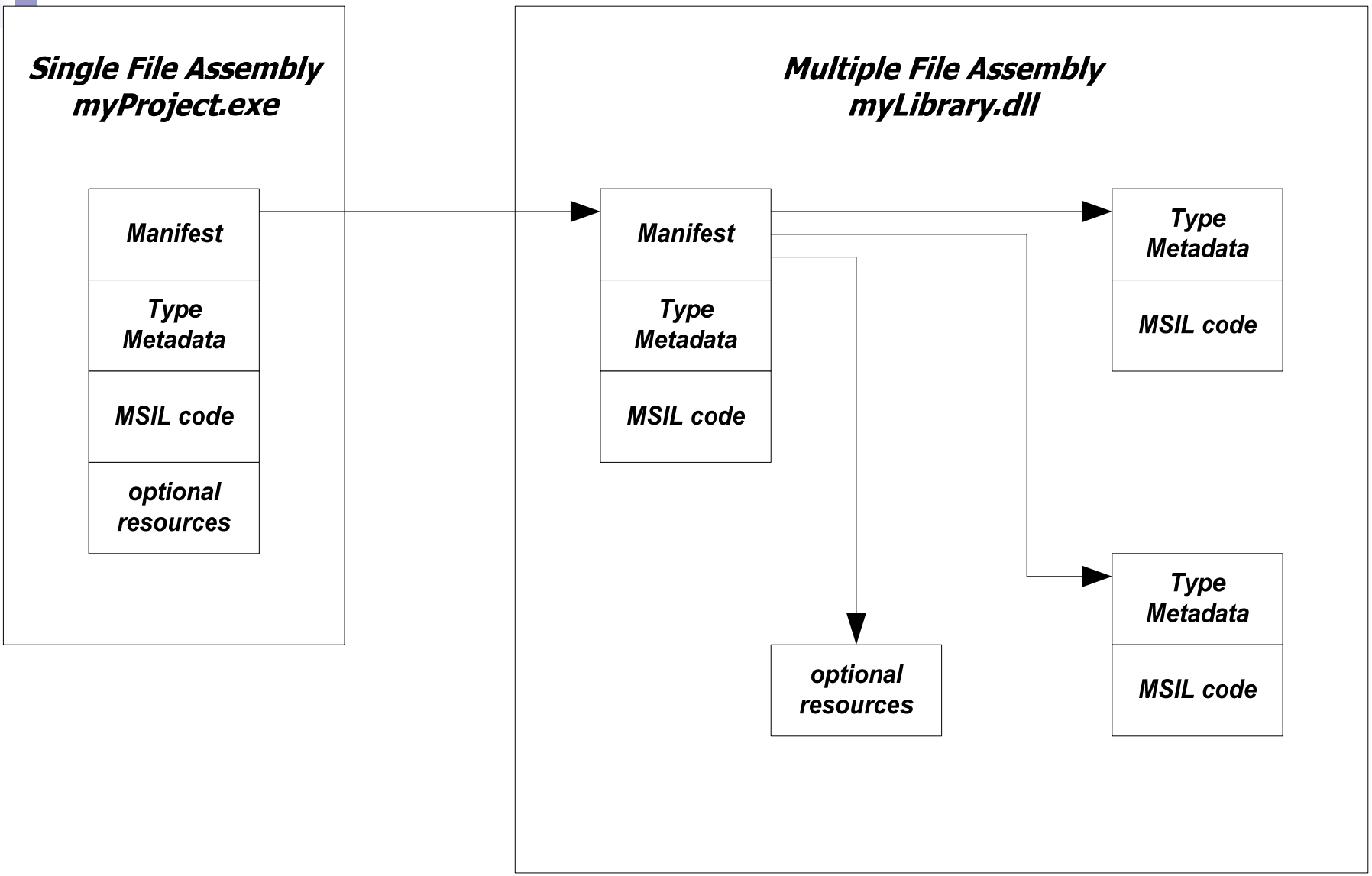


MZ – první 2 byty v MS-DOS programech, iniciály Mark Zbikowski – jeden z vývojářů MS-DOSu

PE – Portable Executable

COFF – Common Object File Format

Assembly v paměti



Struktura asembly

Assembly – *soubor *.exe nebo *.dll*

Portable Executable (PE) Wrapper

EXE header	**Hlavička kompatibilní s MS-DOS programy**
STUB	**Kód MS-DOS programu** vypisující obvykle hlášení "*Program nelze spustit v MS-DOS.*"
NEW header	. + další informace nutné pro OS

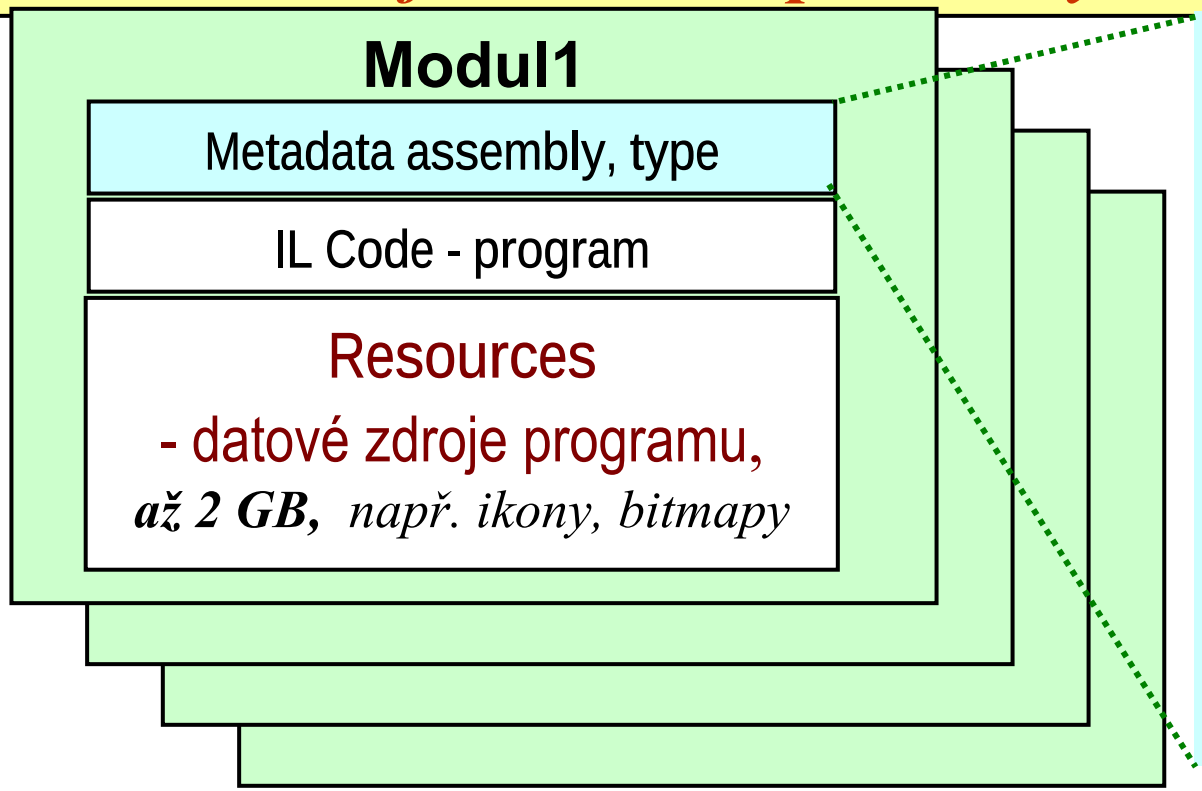
Manifest – *údaje o modulu*

Modul

- Metadata
- IL Types
- Resources - datové zdroje programu, ***až 2 GB*** *např. ikony, bitové mapy*

Detail manifestu a modulů

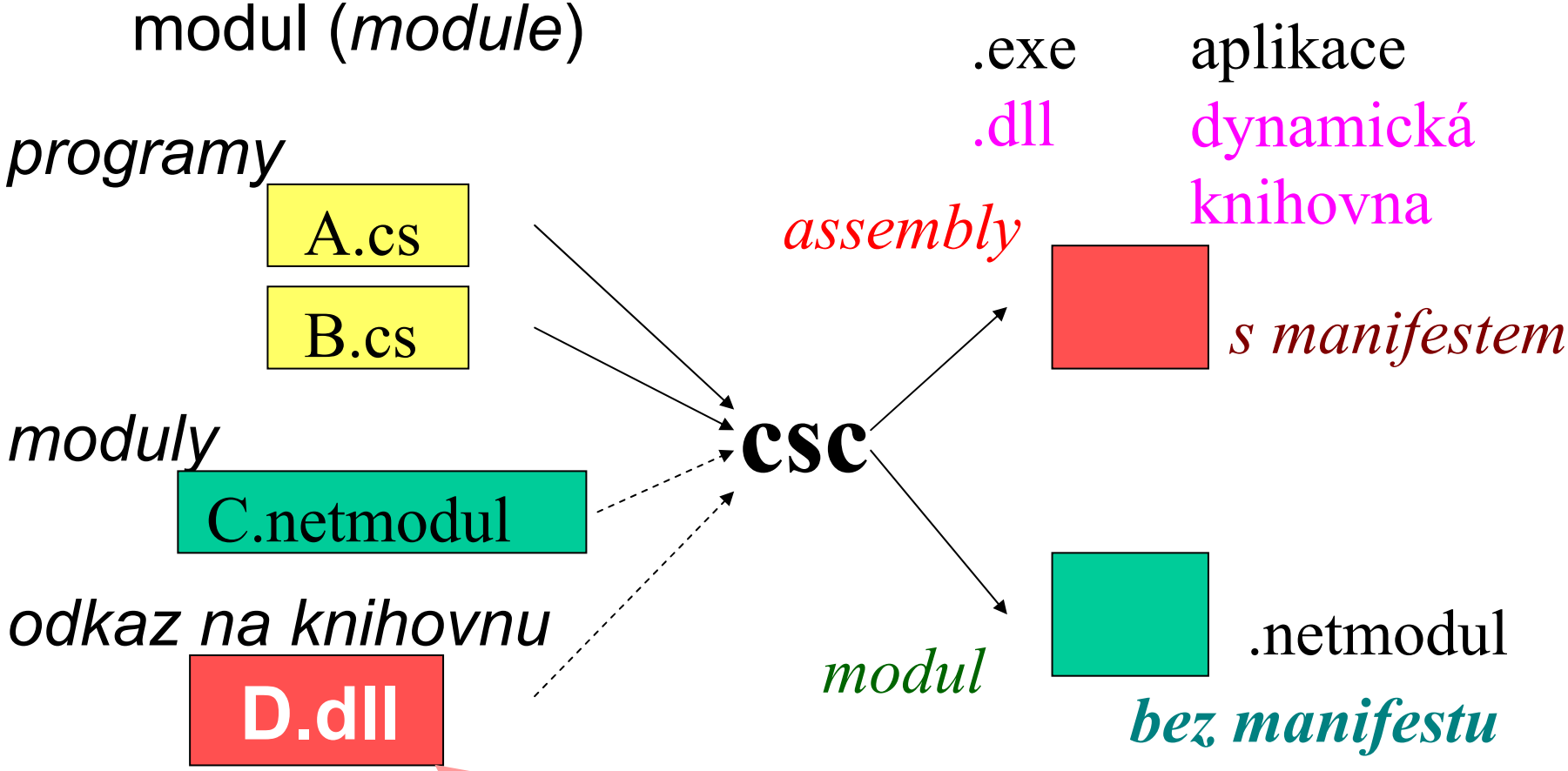
Manifest →*údaje o modulu*
→*odkazy na použité DLL*
→ *číslo verze*
→ *šifrovací klíč 'public key'*



- popis rozhraní
  - třídy
  - metody
  - proměnné
  - parametry
  - typy
- ...

Jak se assembly vytvářejí?

- Každá kompilace vytvoří buď assembly nebo modul (*module*)



Pro samostudium

Takto označené snímky

- slouží jako další rozšíření přednášky;
- nebudeme se u nich zastavovat
- ale nebudou se ani zkoušet.

Překlad Assembly

Podmínky překladu csc

/t[arget]: **exe** = console application (default)
| **winexe** = Windows GUI application
| **library** = knihovna (DLL)
| **module** = modul (.netmodule)

/out:*name* *jiné jméno assembly nebo modulu než default*

Např. csc /t:library /out:MyLib.dll A.cs B.cs C.cs

/doc:*name* *vytvoří XML soubor z /// komentářů*

/r[eference]:*name* připojí referenci na metadata v *name* (např. *xxx.dll*)

/lib:dirpath{,dirpath} adresáře, kde se hledají knihovny uvedené v /r.

/addmodule:name {,name} připojí moduly, ty musí být ve stejném adresáři jako assembly, k níž patří

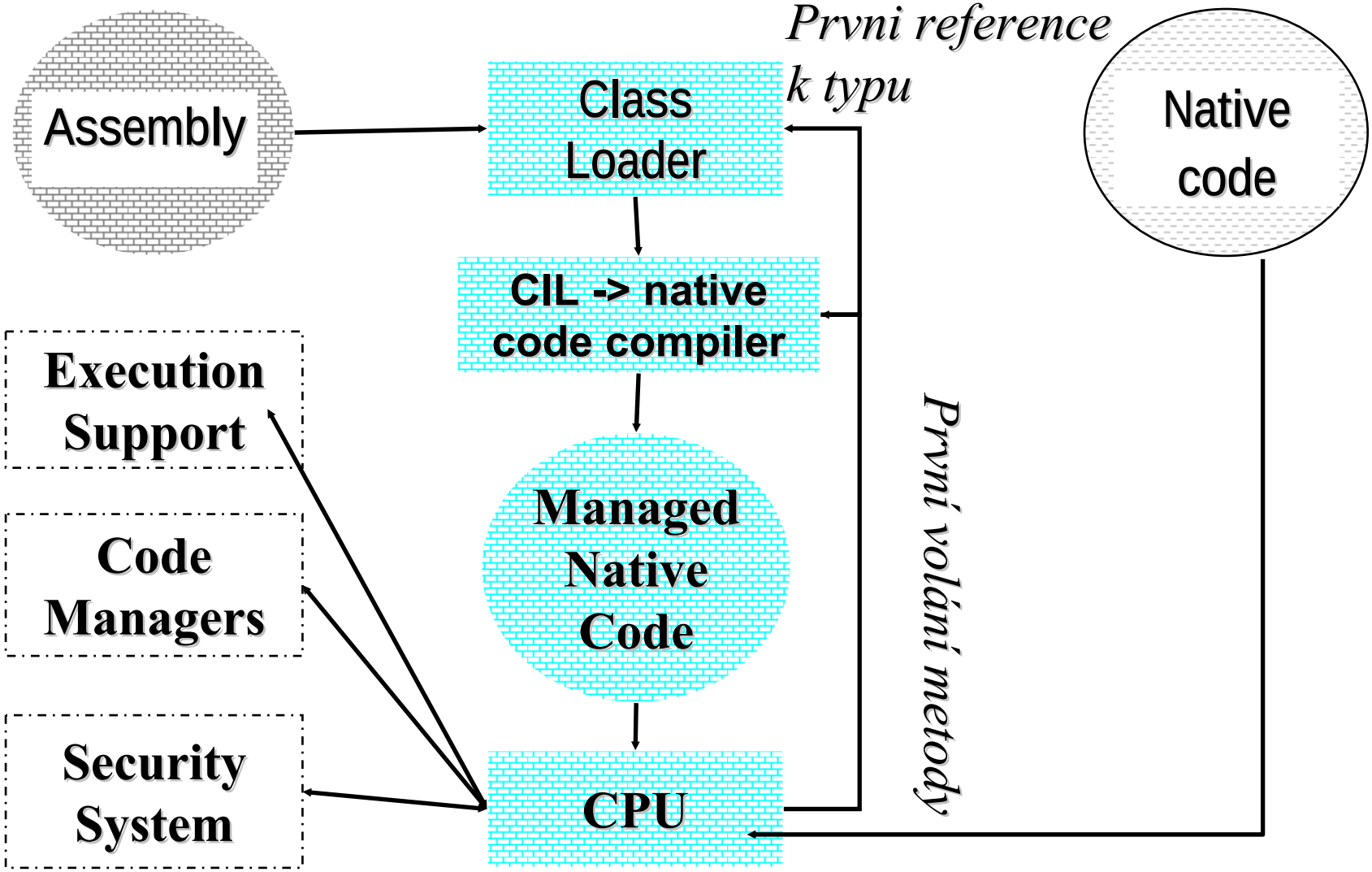
Příklady překladu

- csc A.cs => A.exe
- csc A.cs B.cs C.cs => B.exe
  (pokud kód B.cs obsahuje Main metodu)
- csc /out:X.exe A.cs B.cs => X.exe
- csc /t:library A.cs => A.dll
- csc /t:library A.cs B.cs => A.dll
- csc /t:library /out:X.dll A.cs B.cs => X.dll
- csc /r:X.dll A.cs B.cs => A.exe
  (kde A či B odkazuje na typy v X.dll)
- csc /addmodule:Y.netmodule A.cs => A.exe
  (Y je kopírován do assembly)

CIL = Common Intermediate Language
MSIL = Microsoft –" "–

mezikód programů

CIL-Common Intermediate Language



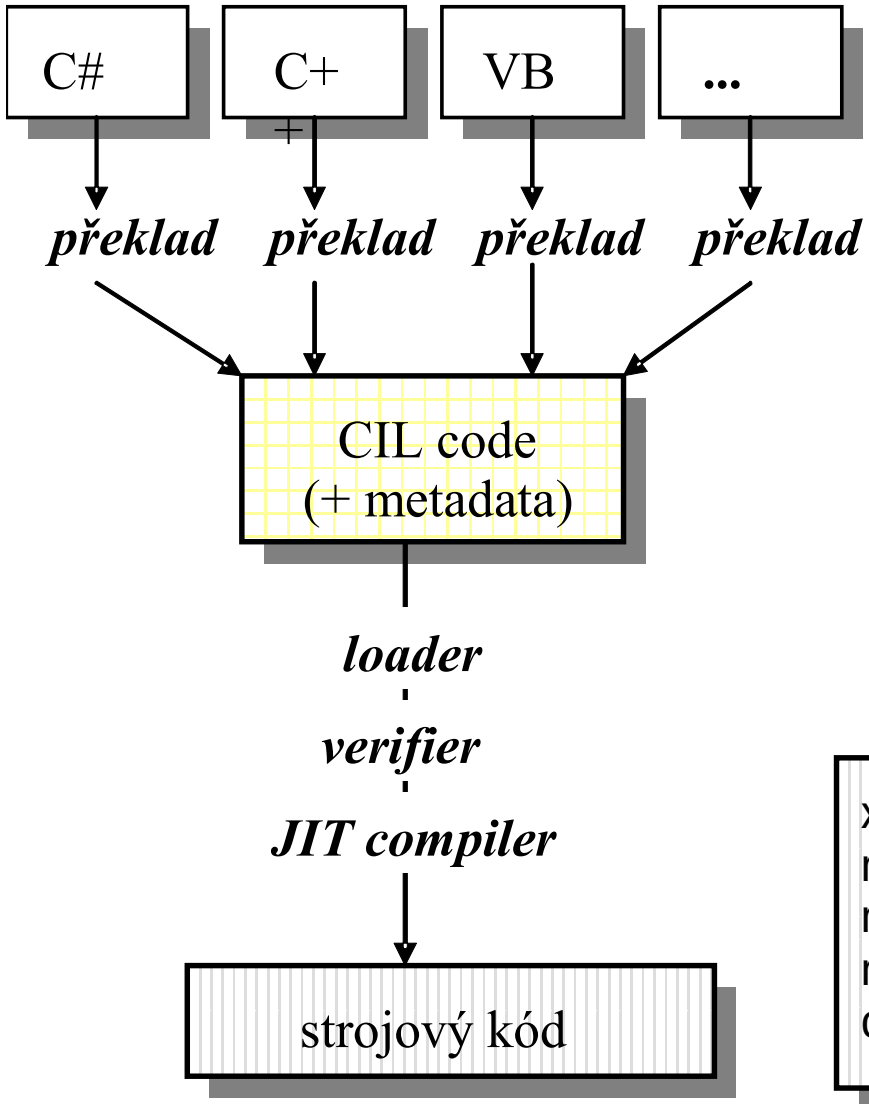
JIT (*Just-in-time*) překladač

- **JIT překladač může pracovat v různých okamžicích**
  - při instalaci Assembly - "install"
  - při nahrání (spuštění) Assembly - "load"
  - při volání metody - "method call"
- Výchozí způsob = “method call”.
  Pro jiné případy lze využít ILASM.EXE utility.

MSIL (Microsoft intermediate Language) respektivě CIL (Common Intermediate Language)

- Pseudo-asembler s objekty
- Některé instrukce
  - Add, mul, call, ret
  - Stloc – ulož hodnotu na zásobník
  - Ldstr – nahraj řetězec na
  - Newobj, throw, catch
- *Běh CIL závisí pouze na zásobníku, tj. operandy nejsou v registrech, což dovoluje matematicky ověřit bezpečnou práci s pamětí. Registry využívá až kód procesoru vygenerovaný ze CIL.*

CIL zůstává čitelný




```
String msg = "Hello World!";
System.Console.WriteLine(msg);
```

```
.locals init ([0] string msg)
ldstr "Hello World"
stloc.0
ldloc.0
call void [mscorlib]System.
      Console::WriteLine(string)
```

CIL

```
xor   esi,esi
mov   eax,dword ptr ds:[01C41058h]
mov   esi,eax
mov   ecx,esi
call  dword ptr ds:[79C56678h]
```

Intel code

Proměnné a překlad do CIL

```
static void m()  { byte b; int i; long l; b=1; i=2; l=b+i; Console.WriteLine(l);   }
```

CIL vypíše: ILDASM.EXE (<…*MSVS.NET*…>\SDK\v1.1\Bin\)

```
.method private hidebysig static void  m() cil managed
{  // Code size       16 (0x10)
  .maxstack  2
  .locals init ([0] unsigned int8 b,  [1] int32 i, [2] int64 l)
  IL_0000:  ldc.i4.1
  IL_0001:  stloc.0
  IL_0002:  ldc.i4.2
  IL_0003:  stloc.1
  IL_0004:  ldloc.0
  IL_0005:  ldloc.1
  IL_0006:  add
  IL_0007:  conv.i8
  IL_0008:  stloc.2
  IL_0009:  ldloc.2
  IL_000a:  call  void [mscorlib] System.Console::WriteLine(int64)
  IL_000f:  ret
} // end of method Class1::m
```

Debug (neoptimalizovaný) kód procesoru

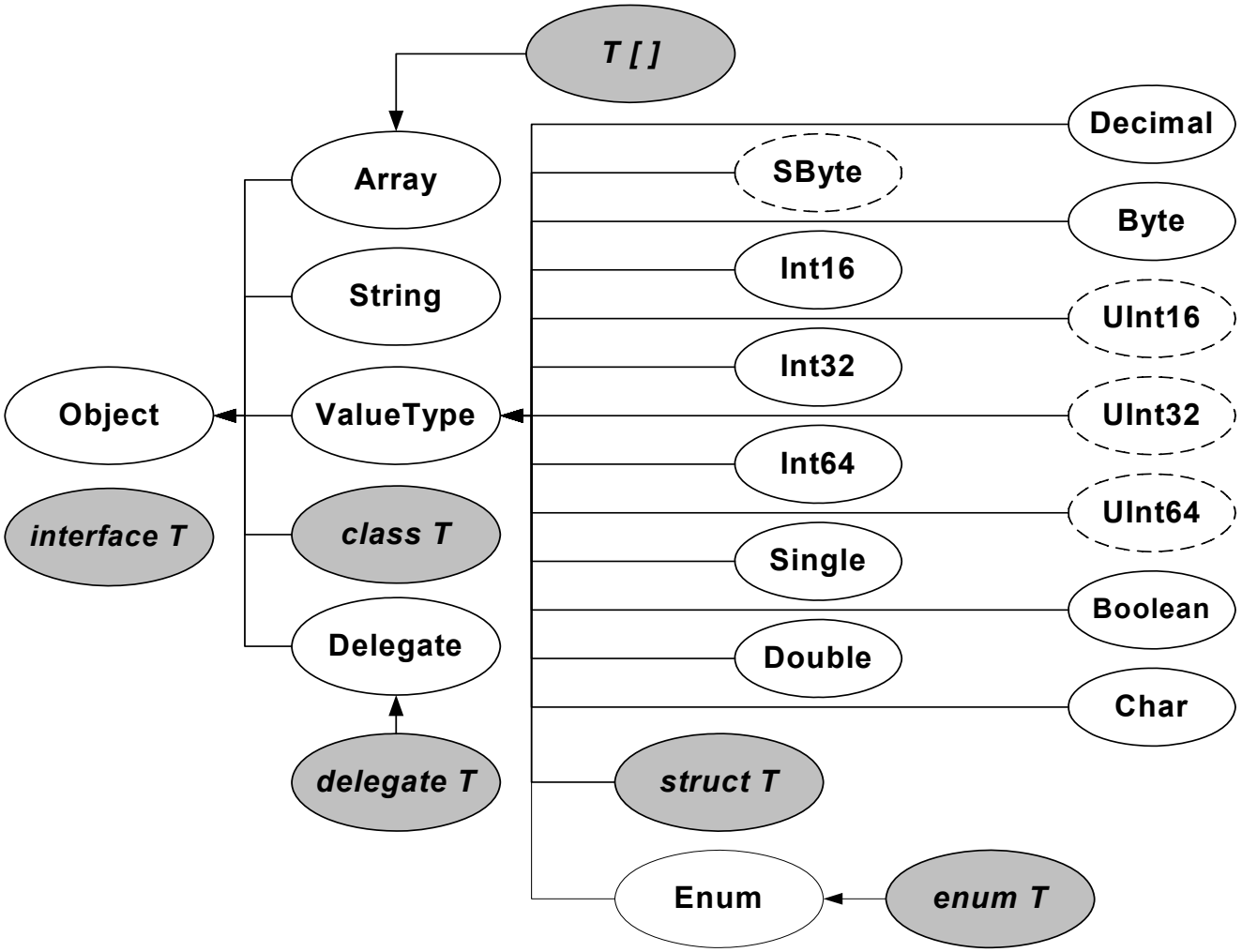
- Visual Studio: Debug->Windows->**Disassembly** *(jen při ladění)*

```
static void m()
 {
byte b; int i; long l;
  push   ebp;
  mov    ebp,esp  // původní hodnota SP
  sub    esp,10h  //paměť pro lokální proměnné
  push   edi
  push   esi
  push   ebx
  xor    ebx,ebx  // ebx := 0 → b
  mov    dword ptr [ebp-8],0          // i:=0
  xor    esi,esi    // ebx := 0 → l
  xor    edi,edi    // ebx := 0
b=1;
  mov    ebx,1
i=2;
  mov    dword ptr [ebp-8], 2
l=b+i;
  mov    eax,dword ptr [ebp-8]
  add    eax,ebx
  mov    edx,eax
  sar    edx,1Fh
// aritmetickým posunem doprava dělíme
// 2^32 -> horní 4 byty čísla long
  mov    esi,eax  // číslo long do esi,edi
  mov    edi,edx
Console.WriteLine(l);
  push   edi
  push   esi
  call   dword ptr ds:[79C5666Ch]
}
  nop
  pop ebx; pop esi; pop edi;
  mov esp,ebp;  pop ebp;
  ret 0
```

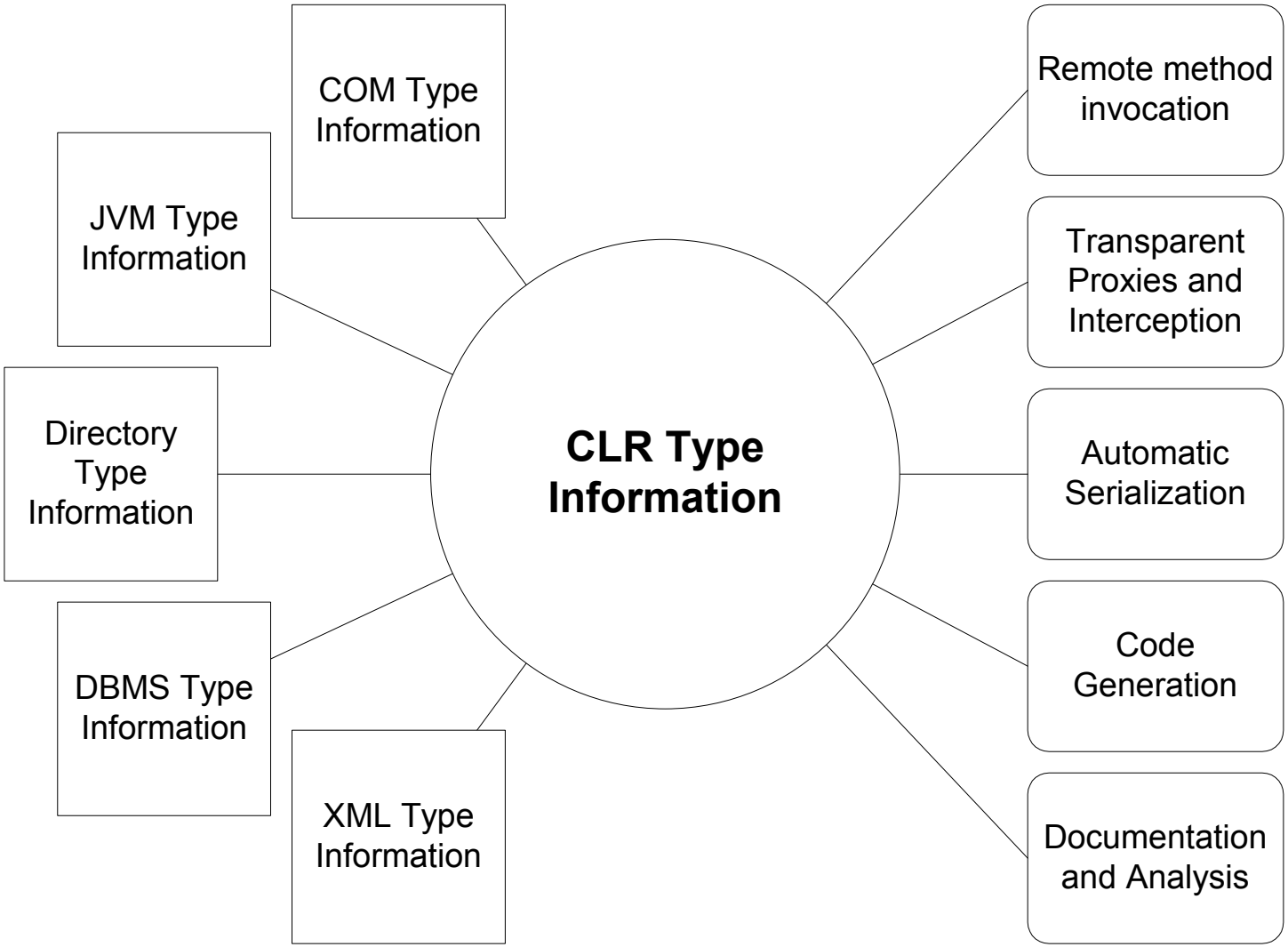
Reflection – základ pro sdílení kódu

[Rob Gonsalves]

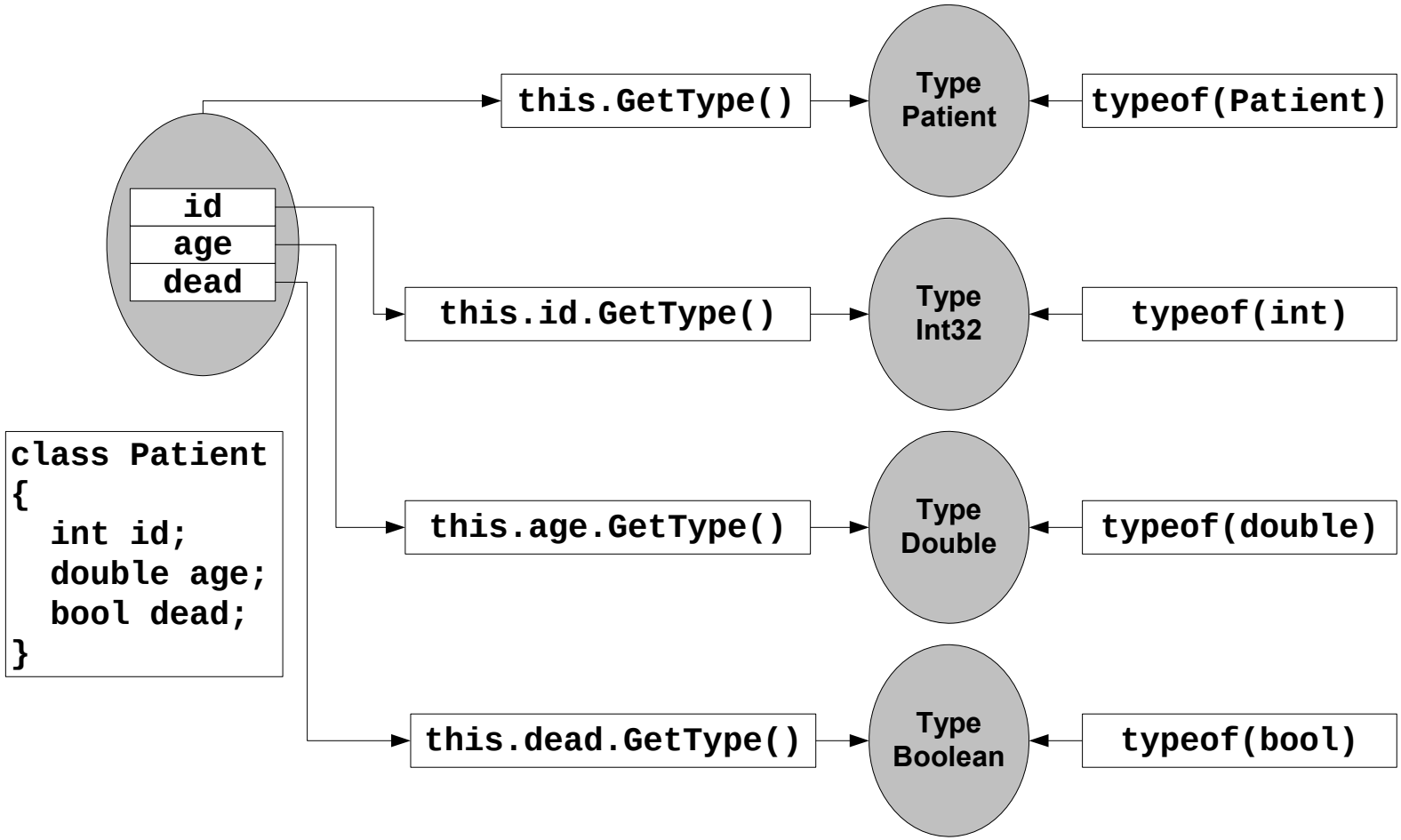
The CLR Type System



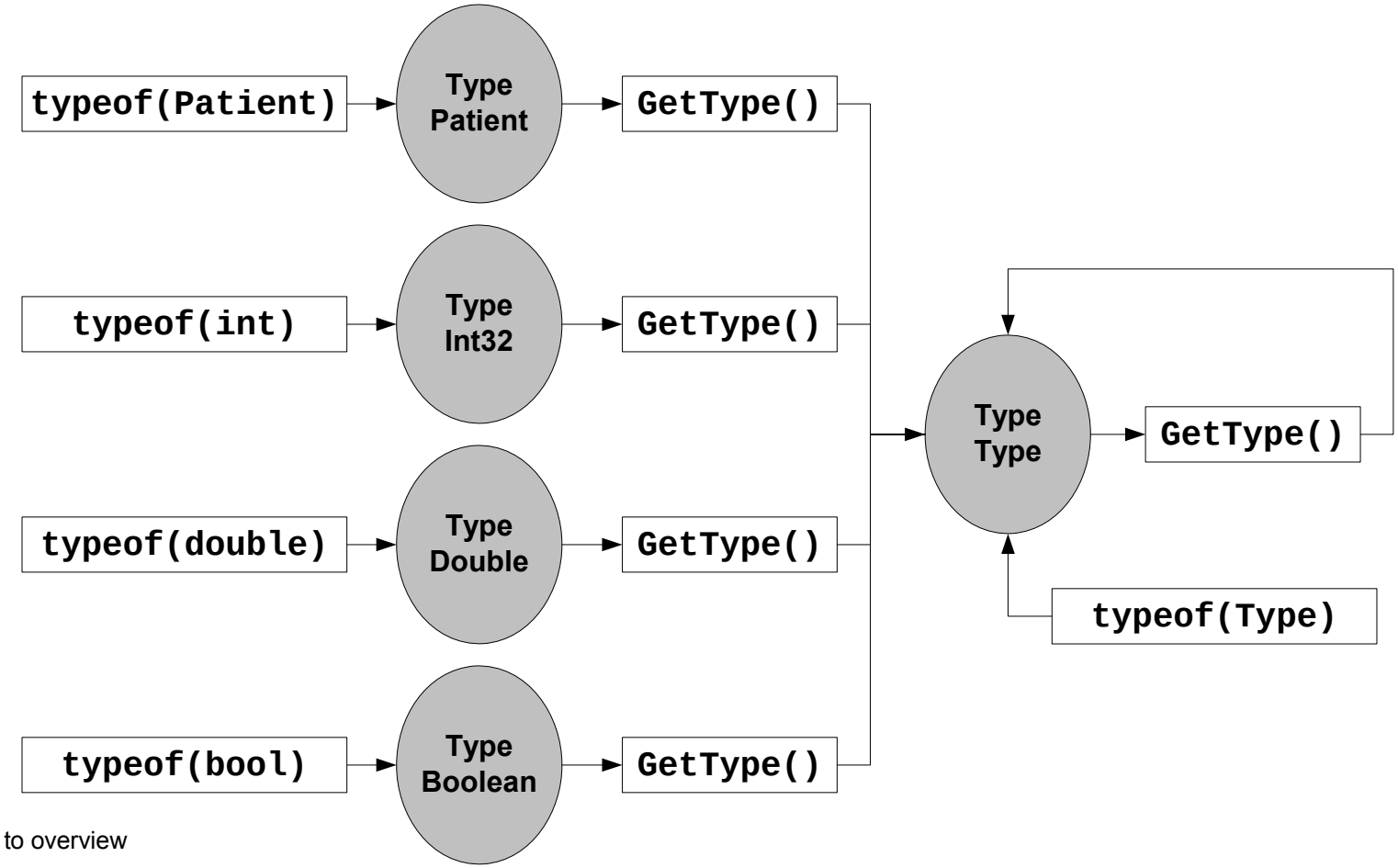
Metaprogramming and Reflection in .NET



Pervasive type and GetType



Pro samostudium Pervasive type and System.Type



Back to overview

Aplikační domény

Jana Jůzová, Indie

Smysl aplikačních domén (AppDomains)

- Izolace paměti
  - Dvě aplikace jsou plně paměťově izolované.
    - Win32 procesy
    - CLR Appdomains
- Smyslem je vytvořit stabilní systém s vysokou bezpečností, a přitom minimálně zatěžující OS.
    - Webové služby a ASP aplikace běží v AppDomains, které je izolují od IIS (Internet Information Service)

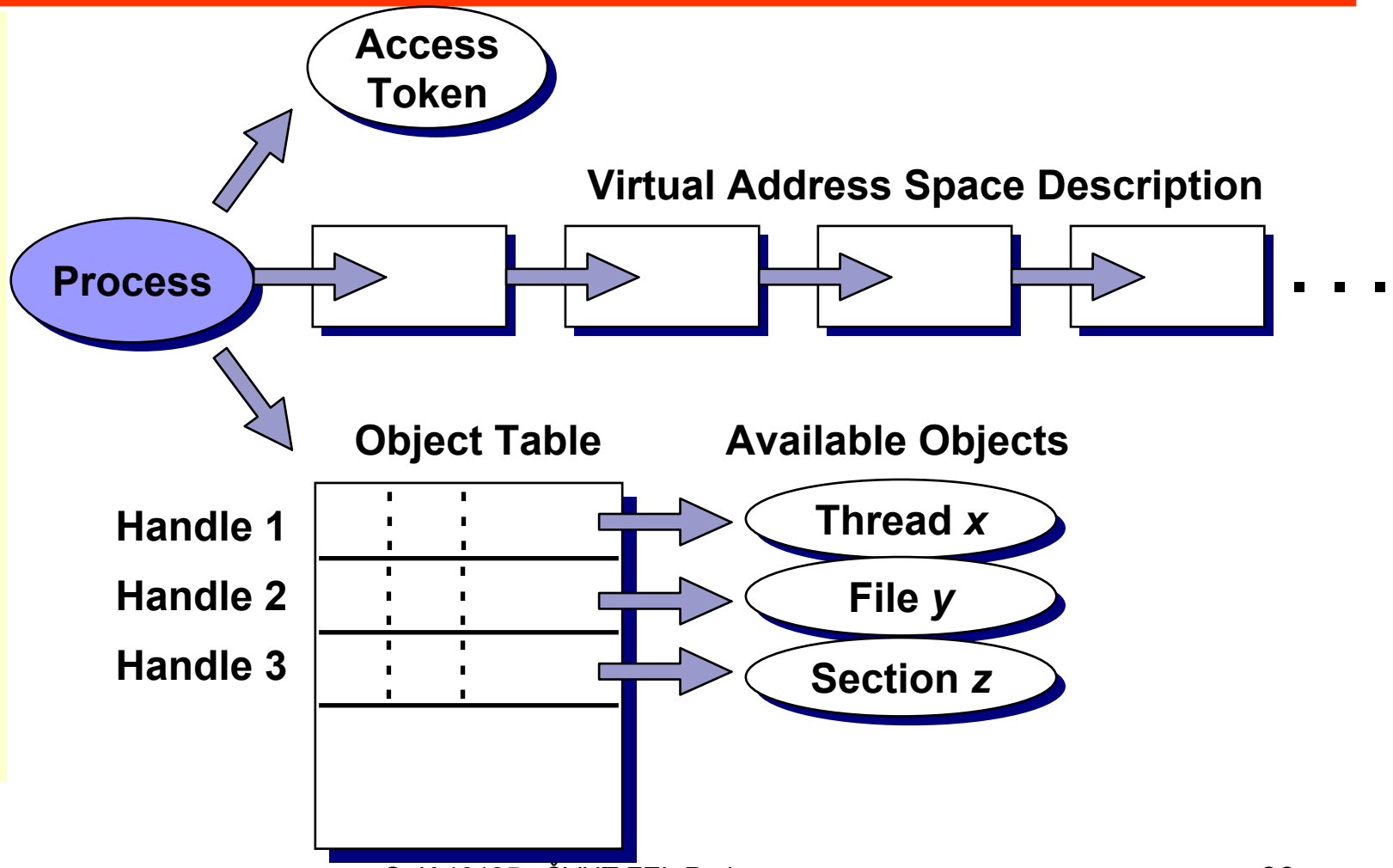
Co je to "Application Domain"?

- Prostředí pro běh „managed“ aplikace, tj. s "garbage collector"
- Kód přeložený jako „safe managed“ v aplikační doméně je izolovaný od „safe managed“ kódu v jiné aplikační doméně.
  - Kód přeložený jako „safe managed“ lze verifikovat JIT překladačem.
  - C# kód bez unsafe bloků je „safe managed“ kód.
- Aplikační domény jsou méně náročné při spuštění, ukončení a běhu než Windows procesy, a to ve smyslu počtu operací a nároků na paměť.
- Funkce v aplikačních doménách se volají mnohem snadněji a efektivněji než funkce druhého procesu Windows.

Náročné vytvoření či zrušení procesu

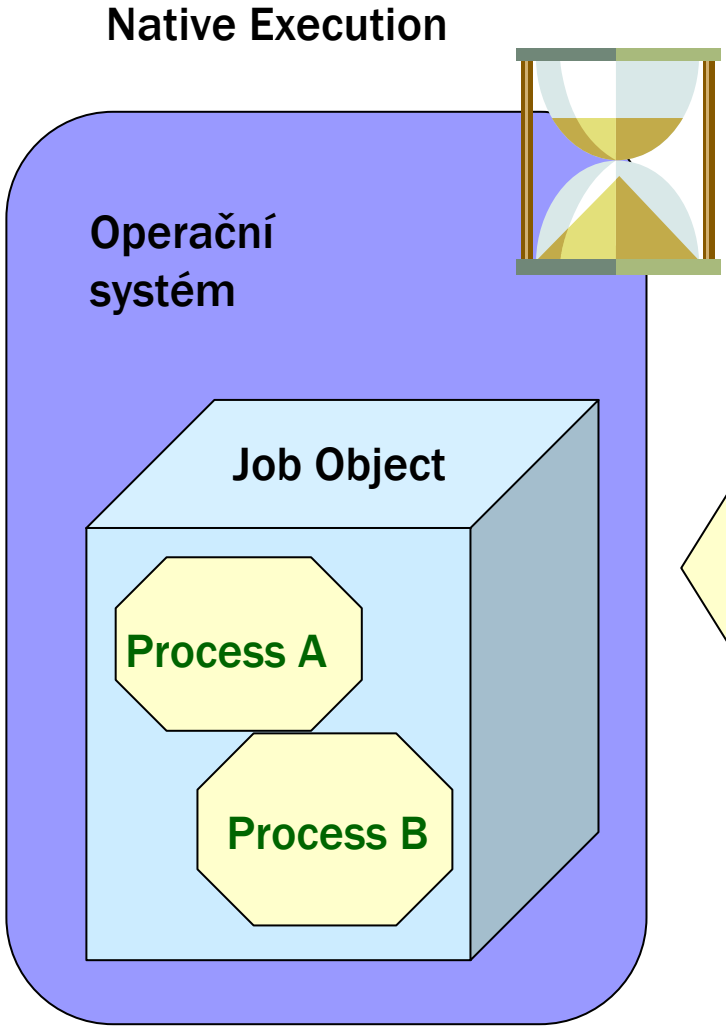
Proces ve Win32 představuje tabulku objektů, paměťový prostor a přidělená oprávnění

OS musí vytvořit / zrušit
* řadu tabulek,
* mapování paměti,
* přidělit oprávnění,
* zařadit proces do plánovače
* a další operace...

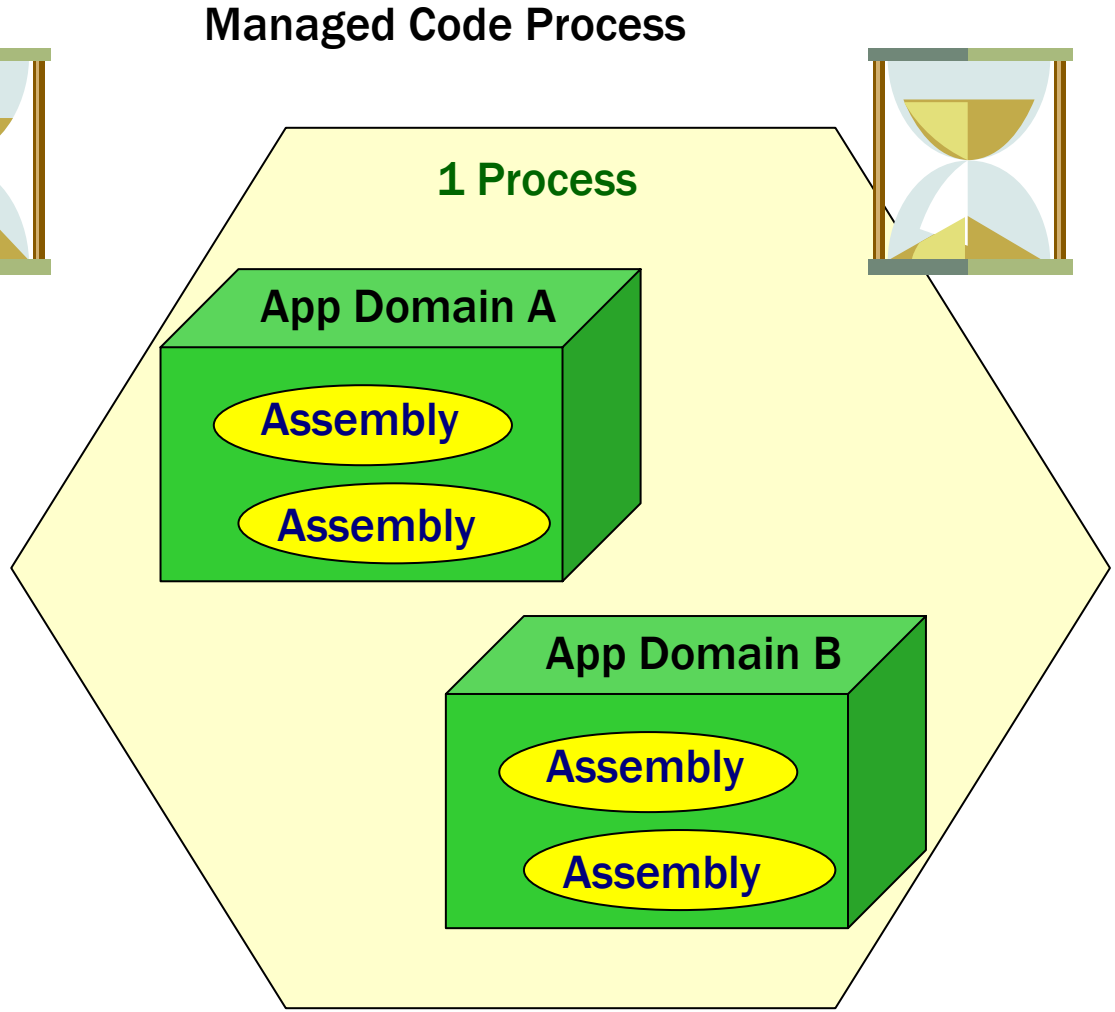


Aplikační domény

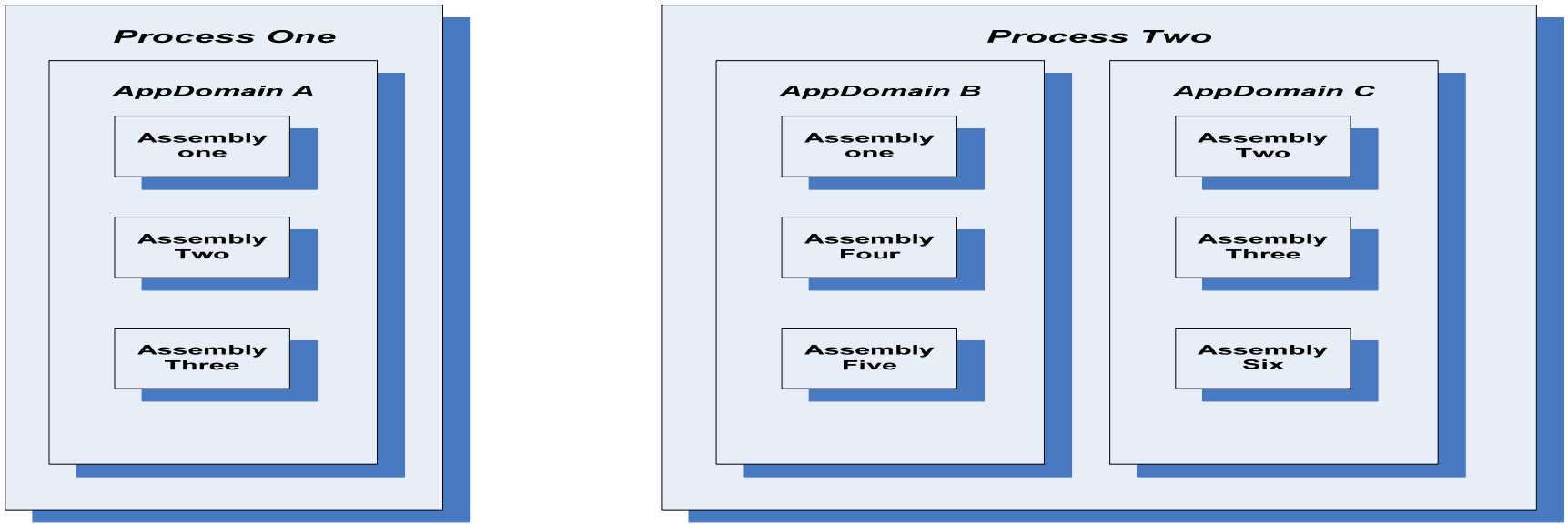
Klasické spuštění dvou operací



Spuštění dvou operací v doméně



Aplikační domény a procesy



- Každá aplikační doména běží v kontextu právě jednoho Windows procesu.
- Proces Windows může mít žádnou aplikační doménu, nebo jednu, případně i více.
- Aplikační domény sdílejí jedno vlákno. Pokud aplikační doménu má běžet v samostatném vlákně, musíme ho pro ni vytvořit.
- U aplikací Console sdílejí aplikační domény i jedno emulované okno.

Příklad: Plugin 1/2 – kódy plugin-ů

```
// PluginA.dll
namespace Plugins
{  [Serializable]
   public class ClassA
   {  public int Run(int data)  { return data * 2; }   }
}
```

```
//pluginB.dll
namespace Plugins
{ [Serializable]
  public class ClassB
  {  public int Run(int data)  { return data * 4; }   }
}
```

Příklad: Plugin 2/5 - příprava

```
static void LoadAndUnload(string pluginName)
{
  AppDomainSetup setup = new AppDomainSetup();
  setup.ApplicationBase = System.IO.Path.GetDirectoryName(
        (Assembly.GetExecutingAssembly().Location));
   //+setup.ConfigurationFile = "(some file)";

 Evidence baseEvidence = AppDomain.CurrentDomain.Evidence;
 Evidence evidence = new Evidence(baseEvidence);
 // +Evidence evidence.AddAssembly("(some assembly)");
 // +evidence.AddHost("(some host)");
```

Příklad: Plugin 3/5 – užití reference

```
int result;
 AppDomain appDomain
      = AppDomain.CreateDomain("newDomain", evidence, setup);
 ObjectHandle pluginHandle;
 switch (pluginName)
 {    case "A": // Run with a priori knowledge from assembly reference
         pluginHandle = appDomain.CreateInstance(
                  // jméno assembly není obecně shodné se jménem souboru
                  AssemblyName.GetAssemblyName("PluginA.dll").ToString(),
                  "Plugins.ClassA");
        Plugins.ClassA pluginA = ((Plugins.ClassA)(pluginHandle.Unwrap()));
         result = pluginA.Run(1);
     break;
```

Příklad: Plugin 4/5 – spuštění přes reflexi

```
case "B": // Run without any assembly reference
   string path = Path.GetDirectoryName(
                        Assembly.GetExecutingAssembly().Location);
   Assembly assembly = Assembly.LoadFile(
                     path+@"\PluginB.dll"); /* nutná absolutní cesta */
   Type type = assembly.GetType("Plugins.ClassB");
   object runnable = Activator.CreateInstance(type);
   object resultObj=type.InvokeMember("Run",
       BindingFlags.Default | BindingFlags.InvokeMethod,
       null,  /*Binder - výběr overload metody, převod parametrů, apod. * */
       runnable, /* instance */
       new object[ ] {2} /* parametry */  );
   result = (int)resultObj;
break;
} // switch
```

Příklad: Plugin 5/5 –uvolnění assembly

```
try {  if (appDomain != null)
                    AppDomain.Unload(appDomain);
       appDomain = null;
      Console.WriteLine("Unloaded");
   }
catch (AppDomainUnloadedException ex)
   {  Console.WriteLine("Unload has failed for "
                        + ex.Message);
   }
```

Disassembled

assembly

Demo

- Podíváme se na naši ApplicationDomain

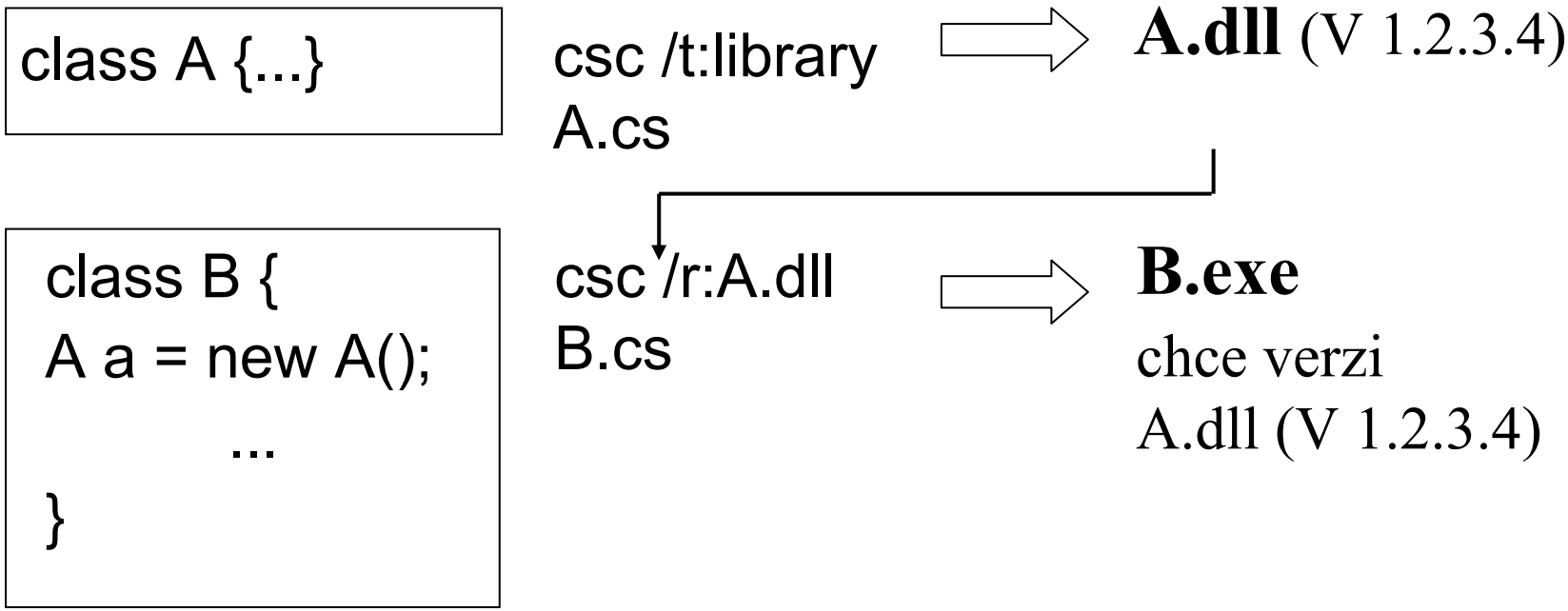
Assembly: verze a digitální podpis

Zdroj [http://farm1.static.flickr.com/]

[Rob Gonsalves]

Verze DLL

- *V C# se hlídá verze knihovny, ta* je uložena překladačem v assembly



- Verze se kontroluje při nahrávání assembly

Druhy assembly

Soukromé (private) assembly

- používá ho jedna aplikace
- leží v adresáři aplikace
- nemá silné (strong) jméno
- nelze ho podepsat

Veřejné assembly (public *nebo* shared assembly)

- může ho používat víc aplikací
- má silné jméno
- lze ho podepsat
- umístěno v GAC (Global Assembly Cashe)
- v GAC lze umístit několik verzí téže assembly

Strong Names

Consist of 4 parts

- the *name* of the assembly (e.g. A.dll)
- the *version number* of the assembly (e.g. 1.0.1033.17)
- the *culture* of the assembly (System.Globalization.CultureInfo)
- the *public key* of the assembly

can be set with attributes

```
using System.Reflection;
[assembly:AssemblyVersion("1.0.1033.17")]
[assembly:AssemblyCulture("en-US")]
[assembly:AssemblyKeyFile("myKeyFile.snk")]
class A {
    ...
}
```

default: 0.0.0.0
default: neutral

Version number

Major.Minor.Build.Revision

Build and *Revision* can be specified as * in the *AssemblyVersion* Attribute:

```
[assembly:AssemblyVersion("1.0.*")]
```

The compiler chooses suitable values then.

Signing Assemblies

Using Public Key Cryptography

1. Generate a key file with *sn.exe* (Strong Name Tool)

```
sn  /k  myKeyFile.snk
```

myKeyFile.snk is generated and contains:
- public key (128 bytes)
- private key (436 bytes)

2. Sign the assembly with the AssemblyKeyFile attribute

```
[assembly:AssemblyKeyFile("myKeyFile.snk")]
public class A {…}
```

During compilation
- the assembly is signed with the private key
- the public key is stored in the assembly

Podpis asssembly

1. vygeneruji klíč

sn /k myKeyFile.snk -> myKeyFile.snk

- public key 128 bytů
- private key 436 bytů

2. podepíši assembly přidáním klíče do attributu

[assembly:AssemblyKeyFile("myKeyFile.snk")]

class A {...} *// nesmí být internal*

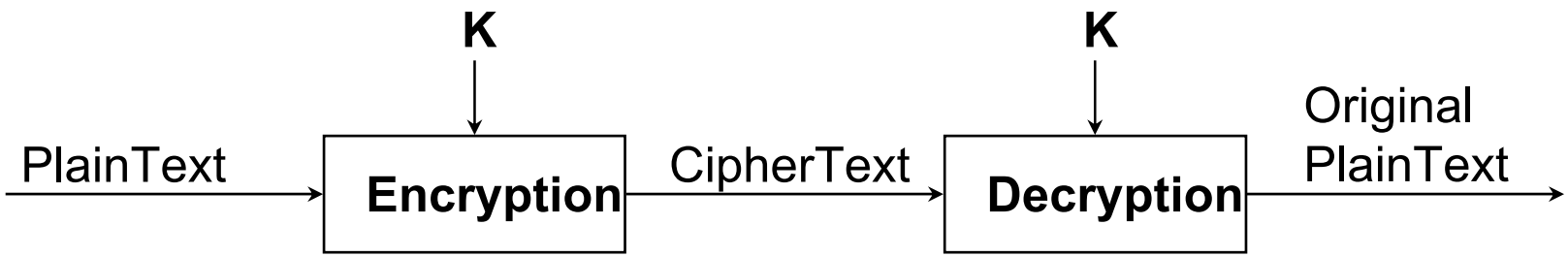
3. přeložíme

Při překladu bude assembly podepsána privátním klíčem a veřejný klíč uložený do manifestu

Cryptography Algorithms

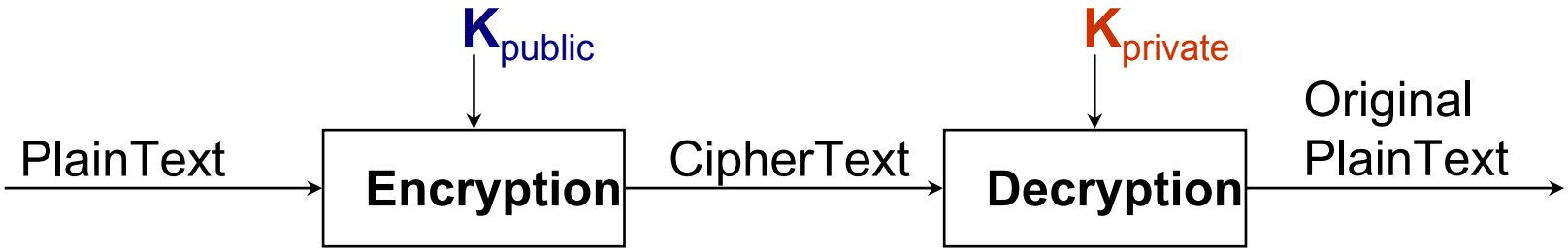
- **Symmetric Algorithm (Secret Key Algorithm)**
- **Public Key Algorithm**
- **Message Digest**

Symmetric Algorithm



- Sender and Receiver Use Same Secret Key
- Fast Encryption and Decryption : Used in Data Encryption
- Problems in sharing the Keys, Short in authentication
- Algorithms : RC4, DES, IDEA, etc

Public Key Algorithm



- Encryption Key(Public Key) and Decryption Key(Private Key) are different.
- No defect in security when send the key to receiver : Used in Key distribution or electronic signature
- Low Speed in Encryption/Decryption
- Algorithm :, DSA, RSA based of **Euler totient**

Leonhard Euler 1707-1783

RSA Cryptography: An Illustrative Example

Example from: Joseph Bonneau, RSA Cryptography

Alice wants to send the message 5678.
She encrypts by her public key [11, 10961]:

$$C \equiv (5768)^{11} \pmod{10961}$$
$$C = 9654$$

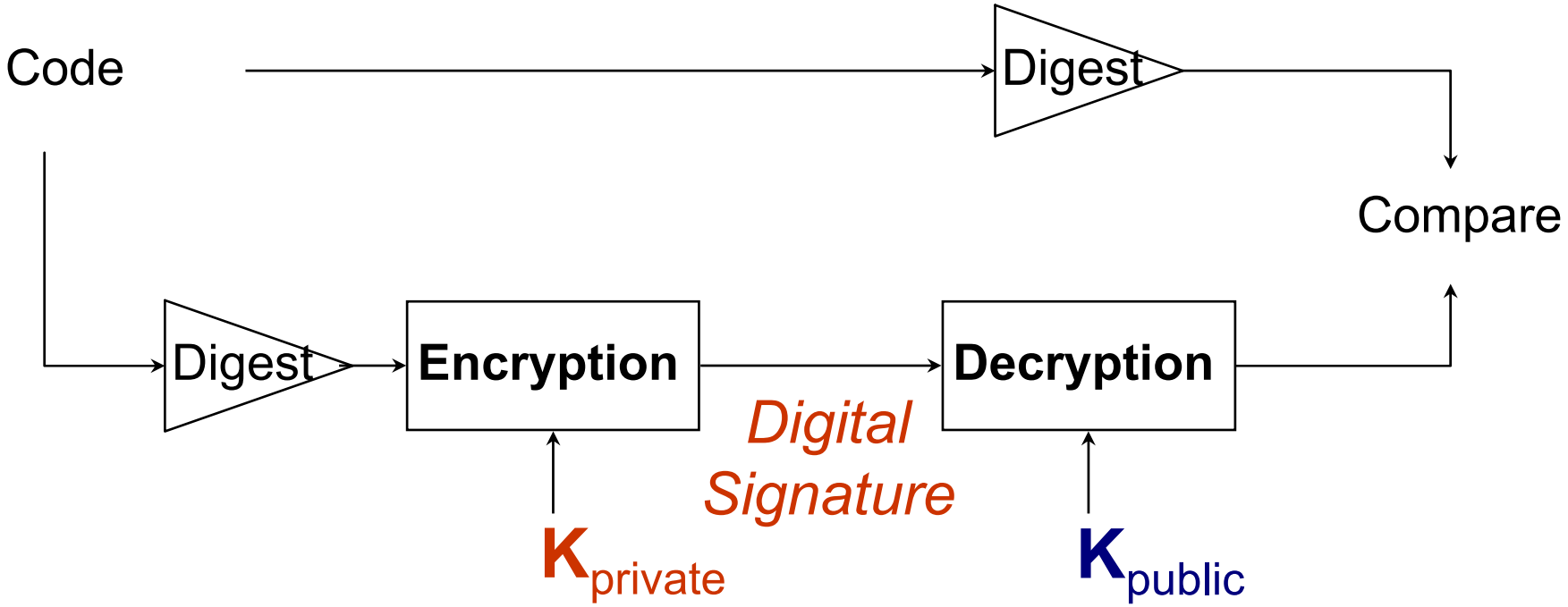
Bob receives the message 9654
He decrypts by his private key [1955,10961]:

$$M \equiv (9654)^{1955} \pmod{10961}$$
$$M = 5678$$

Proper number and exponents are selected with the aid of Euler's Theorem, which explanation is too complex for our short lesson…

Signature

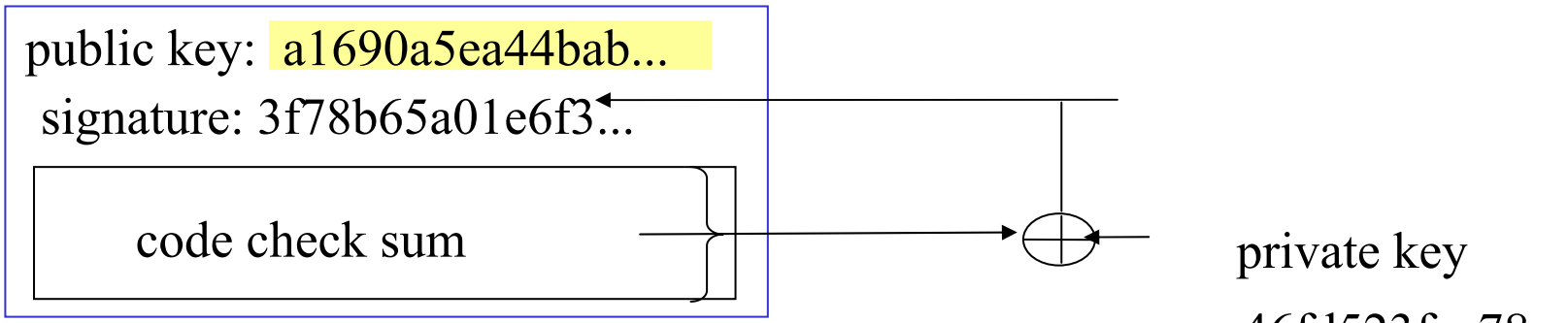
- **Authentication/Integrity/Non-Repudiation**



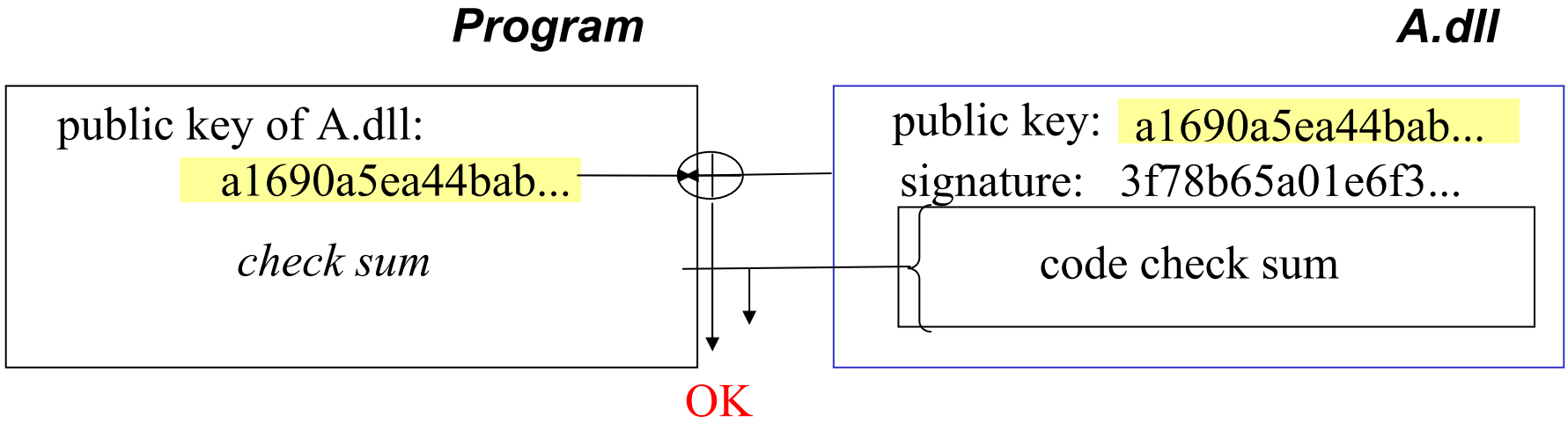
- Convert Given Information into Large Number(Hash Value) within Fixed Length Using One-way Function (Hash)
- Check the Modification of Original Text : Getting the Hash Value from Received Information, then Compare Hash Value with Information
- Hash Function : MD4, MD5, SHA(Secure Hash Algorithm)

Checking signature

Signed on the compilation of *A.dll*

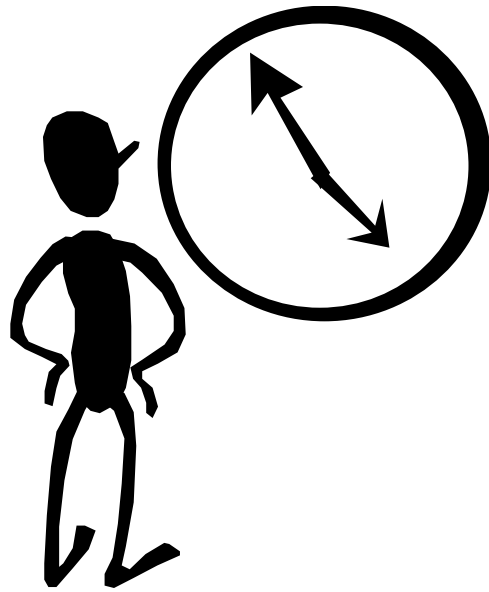


Check on the load of *A.dll*



Delay-signing

- Keeps the private key private
- SN.exe extracts public key from key pair to a new file
  - sn -p mykeyfile.snk publickey.snk
- Create assembly as delay-signed
  - Key file stated contains only public key
  - [assembly: AssemblyDelaySign(true)]
  - AL.EXE with /delay+ parameter
- Fully sign code later
  - SN.exe with -R parameter and full key file



Opožděný podpis

```
[assembly:AssemblyKeyFileAttribute("myKey.snk")]
[assembly:AssemblyDelaySignAttribute(true)]
class A { ... }
```

- Překladač vloží "public key" do assembly a nechá místo pro podpis privátním klíčem
- Dílčí programátoři nemusí znát privátní klíč
- Assembly není ale podepsaná, takže se musí pro ladění vypnout kontrola
  sn –Vr myAssembly.dll

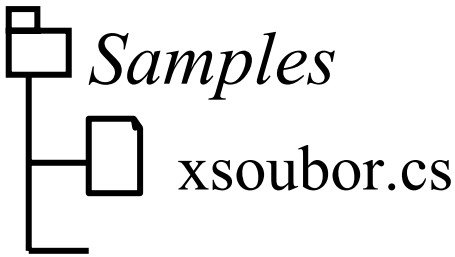
Shared Assemblies

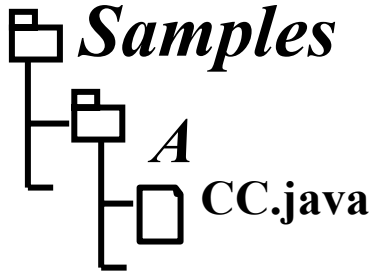
- Strong-named assemblies only
- Global Assembly Cache (GAC)
  - C:\Windows\Assembly
- GACUTIL.exe
  - List assemblies
  - Install assembly
  - Uninstall assembly
    - Use full identity, else all versions removed!
- Use Windows Installer for production deployment
  - Enables application repair and "install on demand"
  - Maintains list of clients referencing each assembly it installs

C# versus Java - 1/3

C#	Java
1 soubor může obsahovat více assembly	1 soubor = 1 package (balíček)
`xsoubor.cs`	`xsoubor.java`
namespace A {...} namespace B {...} namespace C {...}	package A; ... ...
Soubory zůstávají soubory.	Balíčky (packages) se mapují na adresáře a třídy (classes) na soubory

`xsoubor.cs`
namespace A {
class CC {...} }
	`cc.java`
	package A;
	class CC {...}





C# versus Java – 2/3

C#	Java
Importuje namespace	Importuje třídy.
using System	import java.util.LinkedList; import java.awt.*;
Namespace lze importovat také do jiných namespace	Třídy se importují v souborech

namespace A {
  using C; // imporuje C do A
} // jen pro tento soubor
 	
namespace B {
  using D;
}
	import java.util.LinkedList;

C# versus Java – 2/3

C#	Java
C# má viditelnost internal omezující přístup na tutéž assembly	Java má viditelnost "package"

namespace A {
  class CC {...}
  internal class CIn {...}
}
	package A;
	class C {
	  void f() {...} // *package*
	}

Lze používat "alias" pro zkrácení jmen explicitní kvalifikace	
using F = System.Windows.Forms; ... F.Button b;	

Příští přednáška

NETwork of ASP.NET

http://www.biojobblog.com/world-wide-web.jpg