Runtimes

Software techniques

BME AUT
Department of
Automation and
Applied Informatics

Content

- A quick look at runtimes
- JAVA
  - JVM
- .NET
  - Overview
  - Common Language Runtime
  - Building
  - Assemblies

Runtime

- The virtual machine is an abstract computer architecture
- Software above real hardware
- The same application can be executed on different operating systems
- Can manage the code and the data
- Wrapper above the OS API
- We will deal with two concrete cases
  - JVM (Java)
  - CLR (.NET)

Why do we need a runtime?

- VM, IL (intermediate language), LIEP (Language Independent Execution Platforms) These topics have became quite fashionable nowadays for the following reasons:
- Portability: with the help of IL instead of using n*m translators we just need n+m, where n is the number of supported languages and m is the number of platforms
- Compact: The IL code can be more compact than native code
- Efficiency:
  - The runtime can gather statistics about running code, tailoring the native code generated from the IL to the specific platforms.
  - Developers work in a single environment and do not need to learn the specifics of each platform.

Why do we need a runtime?

- Security:
  - managed data: no memory leaks, garbage collector, etc.
  - managed code: catching runtime errors as they happen.

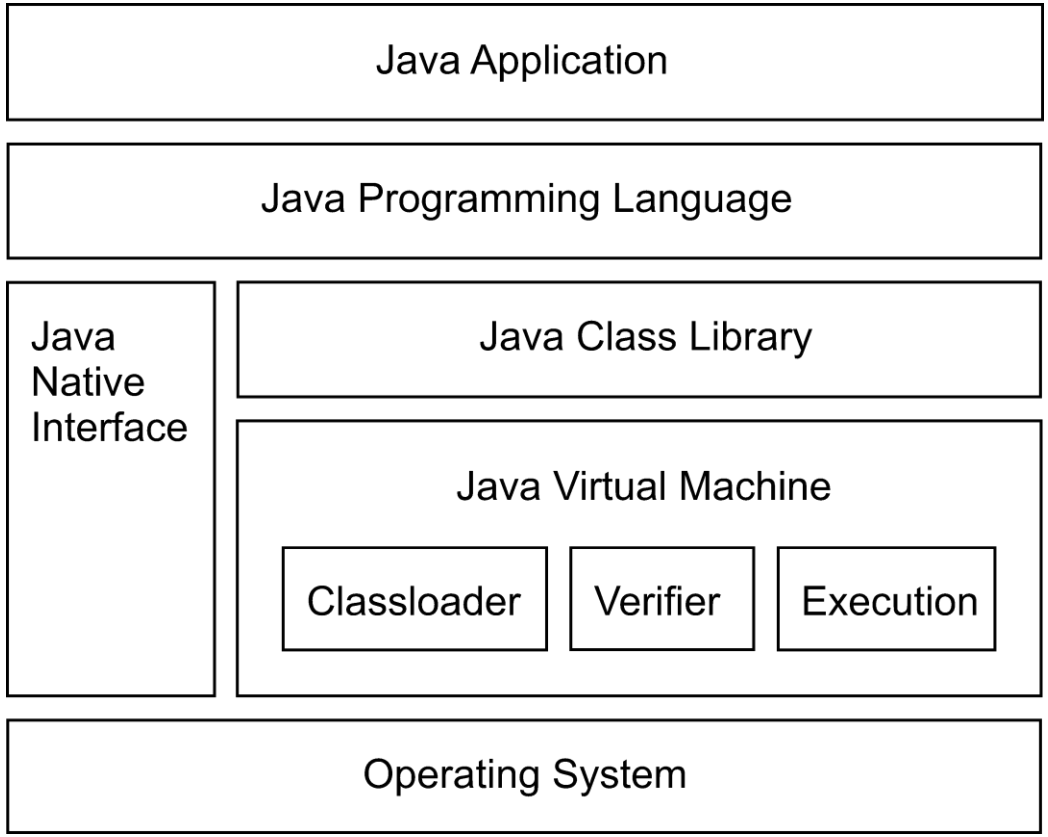
- Interoperability: Supports many languages, these can call each other's assemblies

- Flexibility: „metaprogramming”, reflection, dynamic code generation, serialization, type browsing, etc.

JAVA

Quick recap (skipped during lecture)

Java


Java Application
Java Programming Language
Java Native Interface
Java Class Library
Java Virtual Machine
Classloader
Verifier
Execution
Operating System

Particularities of Java 2

- Availability
  - > Windows, Linux, Solaris,…
  - > Embedded systems
  - > Compiler and runtime for free
  - > Free IDE: Eclipse, Netbeans
- Library
  - > Rich class library
  - > Mobil, Standard, Enterprise
  - > Standardized GUI
  - > Etc..

Particularities of Java 3

- „Built-in model” for handling concurrency
  - > The thread handling is the part of the language
  - > Supports synchronization
- Security
  - > No pointers!
  - > Compile-time checking
  - > Runtime checking
  - > Automatic memory management – GC

JVM

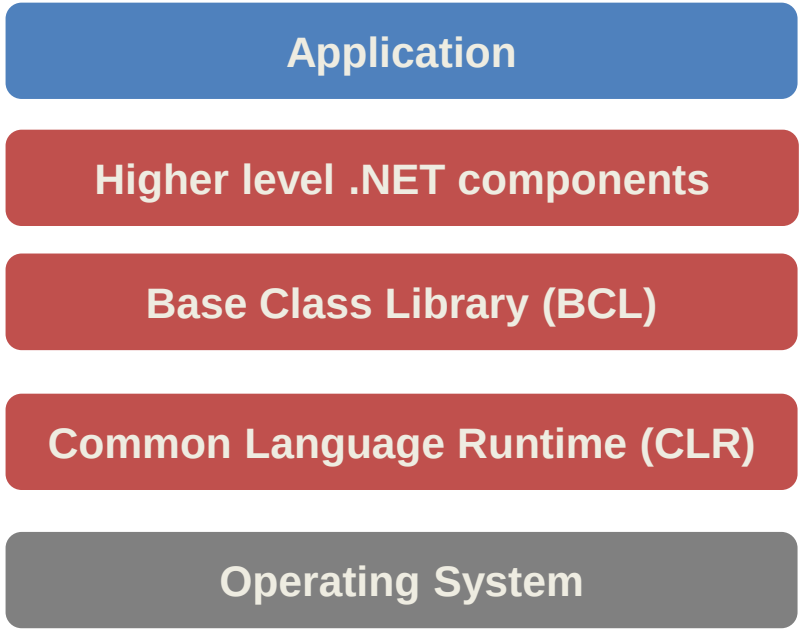
- Runtime environment for executing Java code
- The implementation is not defined!!!!
- Executes the Java .class files
- Must meet the requirements defined in the Sun specification

JVM implementations

- Interpreter
  - > Simple, compact
  - > Slow
- Just-in-time compilation
  - > State-of-the-art desktop/in server cases
  - > „Too resource consuming in embedded systems”
- Hardware implementation

.NET

.NET concepts

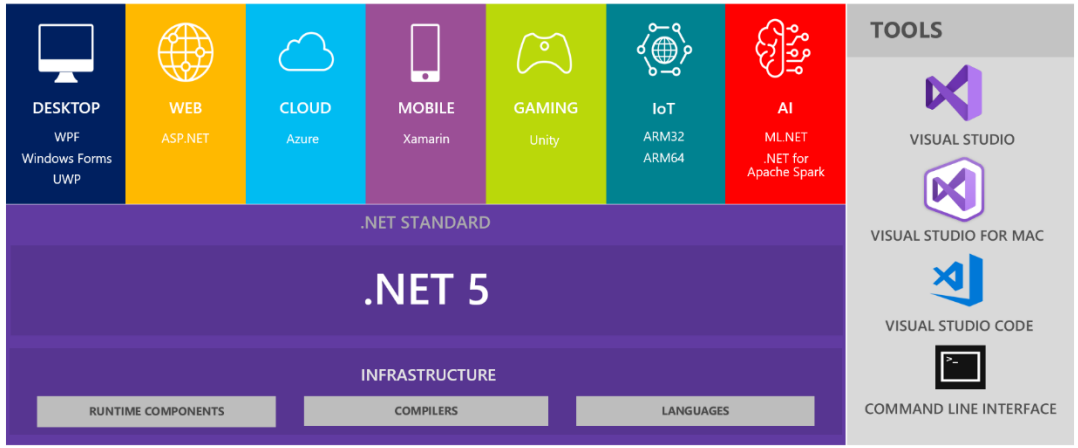


- The .NET Framework
  - > Common Language Runtime – CLR
    - Shared by all .NET languages
    - GC, JIT compiler, etc
  - > Base Class Library – BCL
    - Most important classes (string, file, collections, threads, etc.)
  - > Several built-in features (see later)
  - > NuGet packages – a simple way to download and use a large number of libraries made available by the community

.NET Technologies

- Simple console applications (will look at during this course)
- Web frontend
  - ASP.NET Forms/ ASP.NET MVC/ ASP.NET Core
- Backend services
  - Web API (REST)
  - Windows Communication Foundation (WCF)
- Desktop applications
  - Windows Forms (will look at during this course)
  - Windows Presentation Foundation (WPF)
  - Universal Windows Platform (UWP)
  - WinUI 3
- Thick and mobile client applications
  - Xamarin (uses Mono)
  - MAUI, .NET Multi-platform App UI
- Games
  - Unity
- Data management
  - ADO.NET (will look at during this course)
  - Entity Framework
- Etc.

.NET – A unified platform



.NET (runtime) versions

- **.NET Framework** – focus of this course
  - Originally, this was the only type of .NET
  - Windows only
  - Latest version is 4.8.1, no longer updated in earnest
- **.NET Core**
  - Cross-platform (Windows, Linux, Mac)
  - Less functionality – but has been growing constantly
  - Version 3.1 is the latest, stopped at this point, .NET took over its role
- **.NET**
  - Continuation of .NET Core
  - .NET 7 is the latest (however, .NET 6 is the LTS – Long Term Support)
- **Mono**
  - Cross-platform (Windows, Linux, Mac, Android, iOS (Xamarin), Unity3D)
  - Feature set somewhere between Full and Core
- **.NET Standard**
  - This is not a real implementation, but like the name suggests, a **standard** that has the role of ensuring cross-compatibility between the framework's different versions.
    - Full .NET, .NET Core, Mono implement this
  - Multiple versions

.NET

CLR

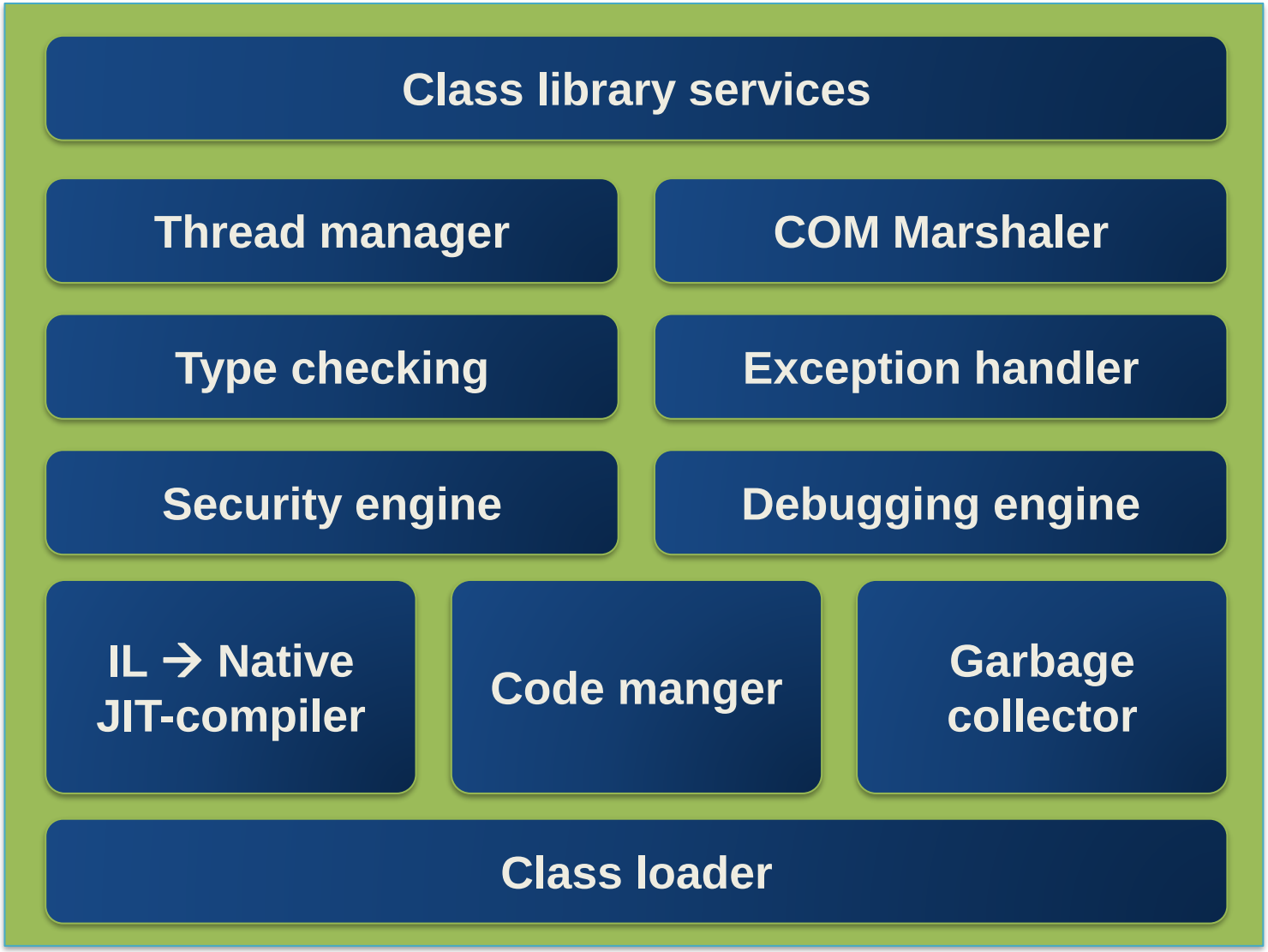
CLR – Common Language Runtime

- Designing considerations
  - > Simple application development
  - > Support for/integration of multiple programming languages
    - > Common Language Specification - CLS
  - > Type compatibility
    - – Common Type System - CTS
  - > Robust and secure managed runtime
    - – IL, GC, CAS, …
  - > Simple deployment and administration

Easier development

- Object-oriented
  - The typesystem of the runtime is object-oriented
  - A class can derive from one base class
  - A class can implement any number of interfaces
- Built-in classes of the framework
  - Hierarchical, the code is grouped into classes and namespaces
- Uniform and rich type system
  - Every type is a class
  - No VARIANT
    - Every character is a Unicode character (even in strings)
- Supports component oriented development
  - New language elements: Properties, Delegates, Events, Attributes, …
  - Rich design-time support for developers
- Seamless backward compatibility with earlier technologies
  - COM, COM+ Enterprise Services, C++ native code, etc…
  - COM (and ActiveX)
    - COM components can be use in .NET code
    - .NET objects can be used as COM components
  - Win32 API can be used: PInvoke (Platform Invoke)

CLR components



Languages

- .NET is language independent
  - Every feature of the framework can be used in all the languages
  - Every language is compiled to the same type of code (intermediate language), none of them is interpreted
  - Classes that are created using different programming languages can even derive from each other, no distinguished language
- Common Language Specification
  - The Framework can be extended with new languages!

Some more languages

- Compiler platform (Roslyn)
  - > C#, VB, Managed C++, F#
  - > extensible compiler platform
- A few more

Common CLI Languages

- **A#**: CLI implementation of Ada.
- **Boo**: A statically typed CLI language, inspired by Python.
- **C#**: Most widely used CLI language, bearing similarities to Java, Delphi and C++. Implementations provided by .NET Framework, Portable.NET and Mono.
- **C++/CLI**: A version of C++ including extensions for using CLR objects. Implementation provided only by .NET Framework. Can produce either CIL-based managed code or mixed-mode code that mixes both managed code as well as native code. The compiler is provided by Microsoft.
- **Cobra**: A CLI language with both static as well as dynamic typing, design-by-contract and built-in unit testing.
- **Component Pascal**: A CLI-compliant Oberon dialect. It is a strongly typed language in the heritage of Pascal and Modula-2 but with powerful object-oriented extensions.
- **F#**: A multi-paradigm CLI language supporting functional programming as well as imperative object-oriented programming disciplines. Variant of ML and is largely compatible with OCaml. The compiler is provided by Microsoft. The implementation provided by Microsoft officially targets both .NET and Mono.
- **IronPython**: An open-source CLI implementation of Python, built on top of the DLR.
- **IronRuby**: An open-source CLI implementation of Ruby, built on top of the DLR.
- **IronLisp**: A CLI implementation of Lisp. Deprecated in favor of IronScheme.
- **J#**: A CLS-compliant implementation of Java. The compiler is provided by Microsoft. Microsoft has announced that J# will be discontinued.
- **JScript .NET**: A CLI implementation of ECMAScript version 3, compatible with JScript. Contains extensions for static typing. Deprecated in favor of Managed JScript.
- **L#**: A CLI implementation of Lisp.
- **Managed Extensions for C++**: A version of C targeting the CLR. Deprecated in favor of C++/CLI.
- **Managed JScript**: A CLI implementation of JScript built on top of the DLR. Conforms to ECMAScript version 3.
- **Nemerle**: A multi-paradigm language similar to C#, OCaml and Lisp.
- **Oxygene**: An Object Pascal-based CLI language.
- **P#**: A CLI implementation of Prolog
- **Phalanger**: An implementation of PHP with extensions for ASP.NET
- **Phrogram**: A custom CLI language for beginners and intermediate users produced by The Phrogram Company
- **PowerBuilder**: Can target CLI since version 11.1.
- **Team Developer**: SQLWindows Application Language (SAL) since Team Developer 6.0.
- **VBx**: A dynamic version of VB.NET built on top of the DLR. See VBScript and VBA as this could be thought of being used like a Managed VBScript (though so far this name has not been applied to this) and could be used to replace VBA as well.
- **VB.NET**: A redesigned, object-oriented dialect of Visual Basic. Implementations provided by .NET Framework and Mono.
- **Windows PowerShell**: An object-oriented command-line shell. PowerShell can dynamically load .NET assemblies that were written in any CLI language. PowerShell itself uses a unique scripting syntax, uses curly-braces, similar to other C-based languages.

Other CLI languages

- **Active Oberon** - a CLI implementation of Oberon
- **APLNext** - a CLI implementation of APL
- **AVR.NET** - a CLI implementation of RPG
- **clojure-clr** - a CLI implementation of Clojure
- **Delphi.NET** - a CLI language implementation of the Delphi language.
- **DotLisp** - a CLI language inspired by Lisp
- **Delta Forth .NET** - a CLI implementation of Forth from Dataman
- **dylan.NET**
- **EiffelEnvision** - a CLI implementation of Eiffel
- **Fantom** - a language compiling to .NET and to the JVM
- **Fortran .NET**: Fortran compiling to .NET
- **Gardens Point Modula-2/CLR** - an implementation of Modula-2 that can target CIL
- **GrGen.NET** - a CLI language for graph rewriting
- **IoNET** - a CLI implementation of Io
- **IronScheme** - a R6RS-compliant Scheme implementation built on top of the DLR
- **IronSmalltalk** - a CLI implementation of Smalltalk built on top of the DLR
- **Ja.NET** - an open source implementation of a Java 5 JDK (Java development tools and runtime) for .NET
- **Common Larceny** - a CLI implementation of Scheme
- **LOLCode.NET** - a CLI implementation of LOLCODE
- **Mercury on .NET** - an implementation of Mercury that can target CIL
- **Net Express** - a CLI implementation of COBOL
- **NetCOBOL** - a CLI implementation of COBOL
- **COBOL2002 for .NET Framework** - a CLI implementation of COBOL
- **COBOL2002 for .NET Framework** - a CLI implementation of COBOL
- **OxygenScheme** - a CLI implementation of Scheme
- **PL/IL** - a CLI implementation of PL/I
- **#S** - A CLI language that implements Scheme (a port of Peter Norvig's Jscheme).
- **#Smalltalk** - a CLI implementation of Smalltalk
- **sml.net** - a CLI implementation of Standard ML
- **Synergy.NET** - a CLI implementation of DIBOL
- **Visual COBOL** - a CLI implementation of COBOL
- **Vulcan.NET** - a CLI implementation of xBase Visual Objects
- **X#** - a CLI implementation of ASM developed for Cosmos. X# was also the codename for the XML-capabilities Cω.
- **Zonnon**, Yet another CLI-compliant Oberon dialect.

Managed C++ and C++/CLI

- Existing C++ code can be reused in a .NET application
- Full access to all the features of the .NET Framework
- It is still C++
  - Minimal extensions to the ANSI approach
  - Doesn’t reduce the power or C++
- Still under „Total Control”
  - This is the only language where native and managed code can be combined
- Managed C++ has been superseded by C++/CLI

C#

- The first really object-oriented member of the C/C++ language family
  - No pointers, just references
  - Knows everything that a .NET language may know
    - Properties, delegates, references, events, attributes, XML documentation
  - Declaration and code in the same place
  - Every variable is an object
- A major part of the .NET Framework and Visual Studio was written in C#
- Comparing with other languages
  - Cleaner than C++
  - Younger than Java, there are many things in common
  - Simpler than VB (or Delphi)
  - Constantly evolving, reached version 11.0 in 2023 as part of .NET 7

Managed environment

Managed data

+

Managed code

Let’s take a look in detail ➔

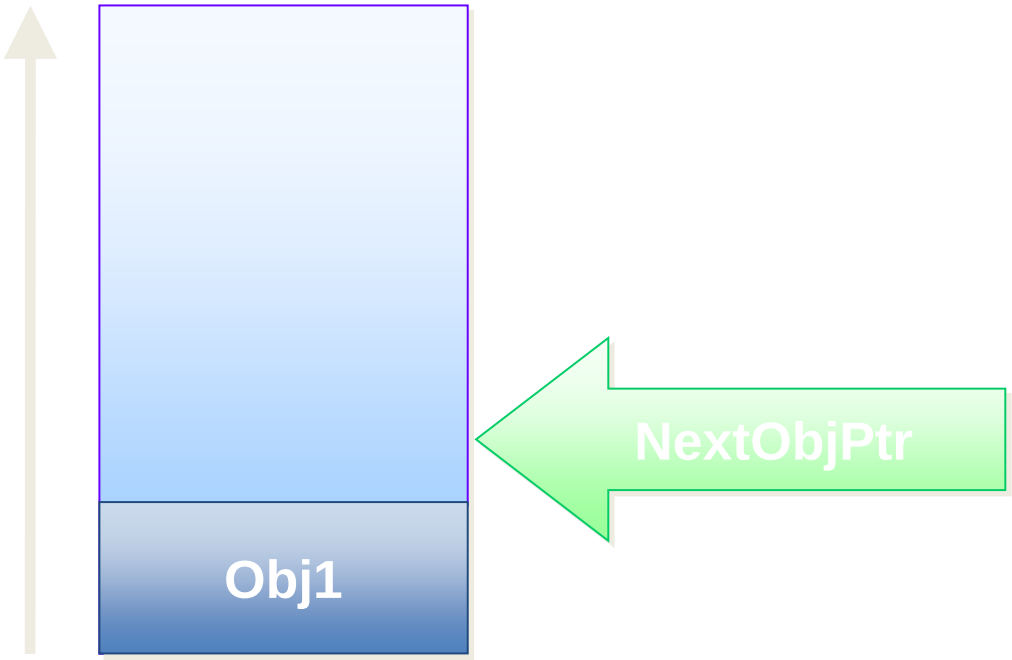
Managed data

- **Automatic life-cycle management**
  - Every .NET object is deleted by the GC
  - No forgotten *delete* calls (on pointers to data on the heap)
  - Efficient
    - Self tuning
    - No reference counting, traverses the whole object graph in memory
  - Server and workstation versions
    - High server side throughput
    - Responsive user interface
    - Fast (>50 million objects per second)
  - 2-phase collection: *mark & compact*

Managed data

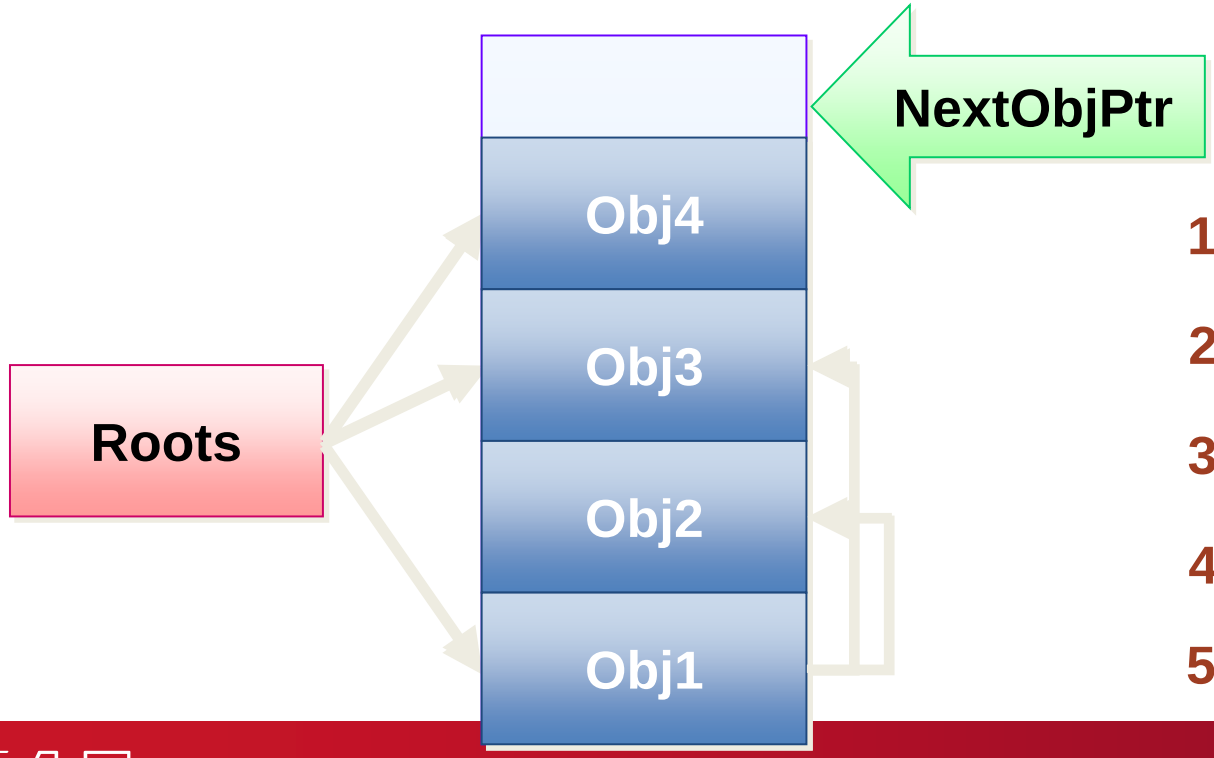
- Memory allocation

  new: increasing the pointer → fast!



Managed data

- Freeing up memory
  - When it is needed: *mark & compact*
  - Uses generations: Young objects die earlier



1. Roots
2. Availability graph
3. Clean up
4. Compact memory
5. Refresh pointers

Managed data

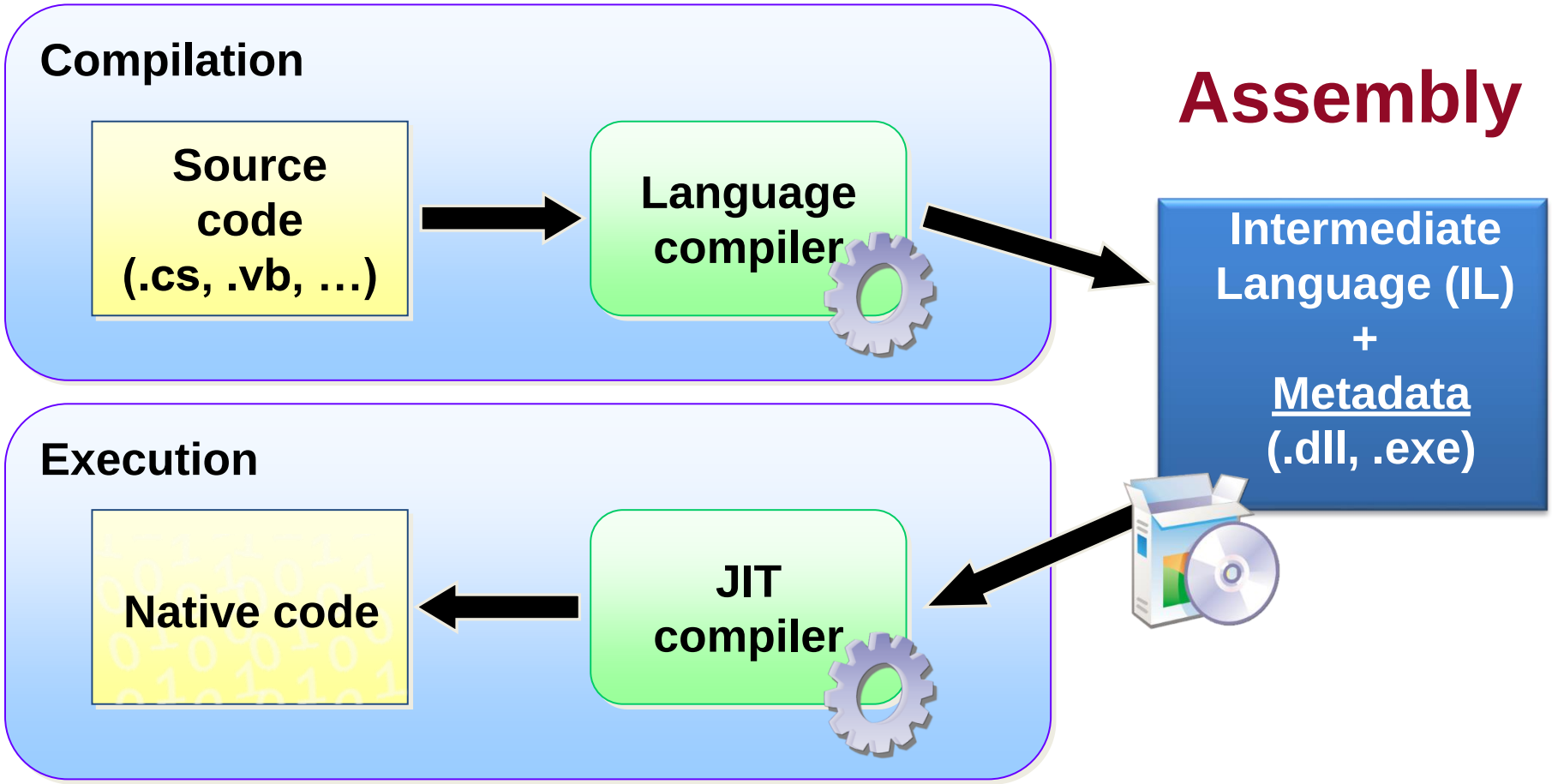
- Disadvantages of GC
  - Overhead
  - Moving large object would be slow → Large Object Heap (LOH)
  - Non-deterministic
    - When will the destructor be called?
    - What order will the destructors be called in?
    - Unnecessary objects shouldn't be kept alive → Dispose patter in a future lecture
- If we are sure that is helps:

```
System.GC.Collect();
```

Managed code

- Managed code: *can be controlled*
  - Compulsory metainformation
  - The code must undergo a strict type check
  - **The most common types of errors are eliminated**
    - Dangerous type casting
    - Uninitialized variables
    - Index out of range
    - Misuse of pointer tables
- Uniform exception handling
  - Part of the CLR
  - Integration with the Windows structured exceptions (SEH)

Compilation and execution



During deployment or when the method is called for the first time

Managed code – language compilers

- Common Language Specification (CLS)
- Different compilers for each languages
- The output is the same: **IL (Intermediate Language)**
- Common services:
  - Inheritance between languages
  - Uniform exception handling
  - Type check, type compatibility
  - Debug symbols
  - Performance counters

Managed code – IL

- IL: Intermediate Language
  - Now officially Common Intermediate Language, but more commonly simply called IL, previously also called MSIL
- The equivalent of Java bytecode
- Processor and architecture independent
- Verifiable
  - references, types, call stack, …
- Designed for further compilation
  - Not interpreted!
- Language independent
- Object-oriented
- Metadata: description for types, member variables and methods
- „Easy” to disassemble, e.g. reflector

.NET Assembly

Deployment – Assembly

- Physical unit of deployment
  - Usually one .dll or one .exe file (but multiple files can also make up an assembly)
  - It contains:
    - IL code
    - Metadata about itself (manifest)
    - Metadata about the .NET classes
    - Resources (.jpg, .txt, …)
  - Not like in Java where we have a separate .class file for each .java source files
- Every application consists of assemblies
  - One namespace can be put into multiple assemblies
  - One assembly can contain multiple namespaces
  - An assembly can refer to other assemblies
- In Visual studio, the project's output is usually one .exe or a .dll

Assembly metainformation (manifest)

- Name (generally the name of the file, without the extension)
- Version: major, minor, build number, revision
- Language and culture information
- Processor and OS
- Assembly references. Information about the referenced assembly:
  - Name
  - Version number
  - Public key, if specified
- Other attributes: publisher, description
- Public key, if it is a shared assembly
- Hash algorithm identifier, if it is a shared assembly
- List of modules in the assembly + footprint (hash)

Assembly as a unit…

- Unit for types
  - Types are bound to assemblies not to namespaces!
- Unit of side by side execution
  - Two different versions of the same assembly can live side by side.
- Security unit
  - Minimal security permissions can be requested for an assembly – ex: `File IO Permission`

Version control

- major . minor . buildnumber . revisionnumber
  - Major, Minor: incompatible versions
    - A new version of the assembly (new features might have also been added)
  - Buildnumber: they may be compatible
    - Small changes – service pack
  - Revisionnumber: compatible
    - Only bugfixes
    - QFE: Quick Fix Engineering

- The version is determined by the *AssemblyVersion* attribute (in the assemblyinfo.cs when using VS.NET)

Dependencies between assemblies

- Assemblies can depend on eachother
  - > For example A.exe uses classes from B.dll
  - > Dependencies must be one-way
    - – If A.dll uses B.dll, B.dll can't use A.dll
    - – This avoids dependency circles
- Based on how we refer to assemblies, there can be
  - > Private
  - > Shared

Let's look at both more closely →

Private assemblies

- The assembly is used by only one application
- The name identifies the assembly
- The assembly is searched for in the directory of the application
  - In the config file a different path can also be defined
- Easy deployment: xcopy
- Wastes disk space as duplicate files are needed when multiple applications use the same assembly
- In the new .NET world, only private assemblies are supported (in .NET Core and from .NET 5)

Shared assemblies

- Synonyms: also refered to as **identified** or **strong-named** assemblies
- More than one application can use them
  - > DLL Hell problem
- Their strong names make them unique:
  - > Name
  - > Public key of the publisher
  - > Version number
  - > Language and culture info (optional)

Avoids DLL hell –assembly name collision

Versions

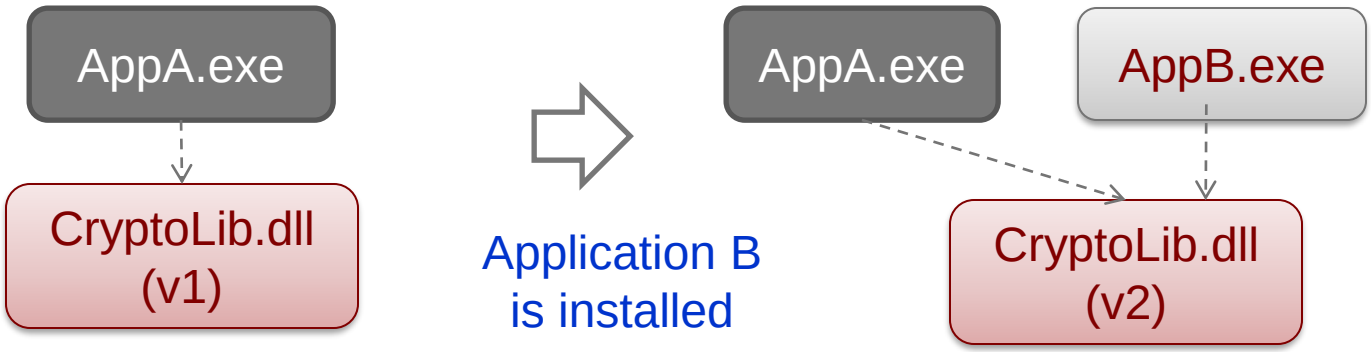
- Digital signature made by the private key of the publisher→ preserves integrity
- A shared assembly can only refer to shared assemblies

Shared assemblies

- Assembly store, Global Assembly Cache (GAC)
  - A store for the assemblies that are used by multiple applications
    - „~ Windows\Microsoft.NET\assembly”
  - Only for shared assemblies- uniqueness, security
  - %WINDIR%\Assembly directory
    - Shell in the browser: drag&drop deployment
    - ... or gacutil command-line tool
    - Only an administrator can remove them

Shared assemblies - versions

- The classic DLL Hell problem



- By installing „AppB” we broke „AppA” because „CryptoLib.dll” v2 is not fully backwards compatible with v1-el.
- DLL Hell can be avoided by using shared assemblies
  - If we install a new version of a DLL, the old applications will continue to use the old version
  - The company that created the assembly can set up a redirect (for example to fix a bug)
  - Administrators can override this (in case the admin thinks the publisher is wrong and the new versions breaks more things than it fixes ☺)

NuGet

- Package manager to quickly install dependencies
- A nice move towards community-driven development
  - Packages developed by both Microsoft and the community
  - Stats like the number of downloads and when it was last updated
- https://www.nuget.org/packages

.NET class library- System namespace*

- System – basic typles like Int32, Array, …
- System.Collections
- System.Data
- System.Diagnostics
- System.DirectoryServices
- System.Drawing
- System.IO
- System.Net
- System.Reflection
- System.Security
- System.Text
- System.Threading
- System.Timers
- System.Web
- System.Web.Services
- System.Web.UI, System.Web.UI.HtmlControls, System.Web.UI.WebControls
- System.Windows.Forms
- System.Xml
- And a lot more…

References

- JVM
  - *The Java Virtual Machine Specification*. http://java.sun.com/docs/books/jvms/first_edition/html/VMSpecTOC.doc.html
- CLR
  - www.microsoft.com/net
  - https://docs.microsoft.com/en-us/dotnet/
- C# Programming guide
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index