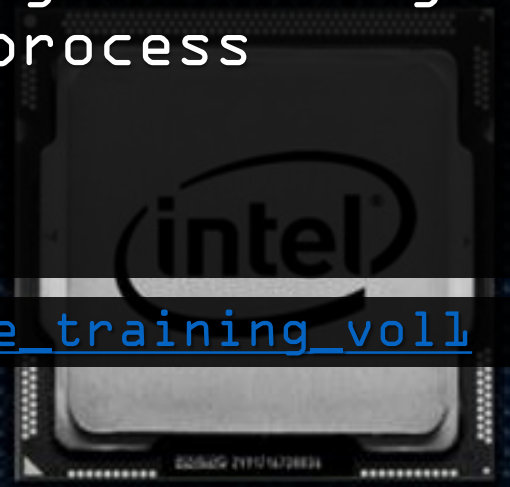
Module 1

A journey from high level languages, through assembly, to the running process



https://github.com/hasherezade/malware_training_vol1

CC BY

Creating Executables

Compiling, linking, etc

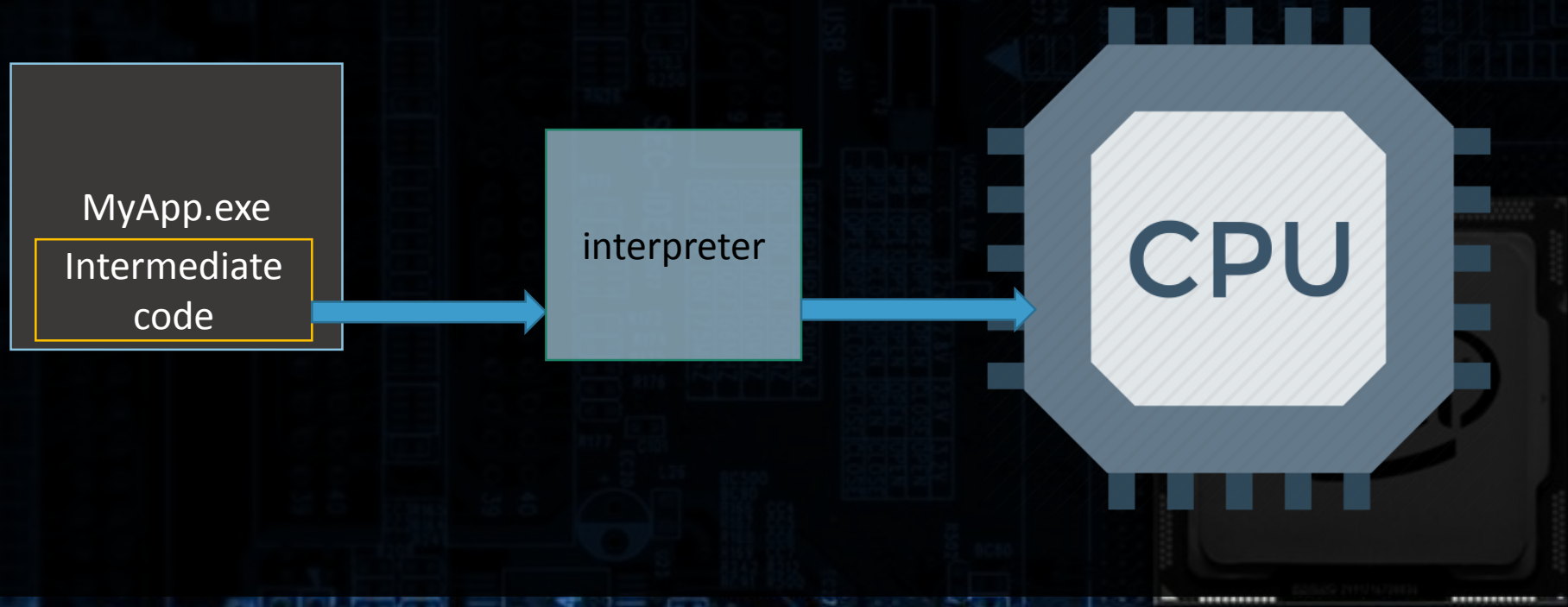
- The code of the application must be executed by a processor
- Depending on the programming language that we choose, the application may contain a native code, or an intermediate code

Compiling, linking, etc

- Native languages – compiled to the code that is native to the CPU

Compiling, linking, etc

- Interpreted languages – require to be translated to the native code by an interpreter

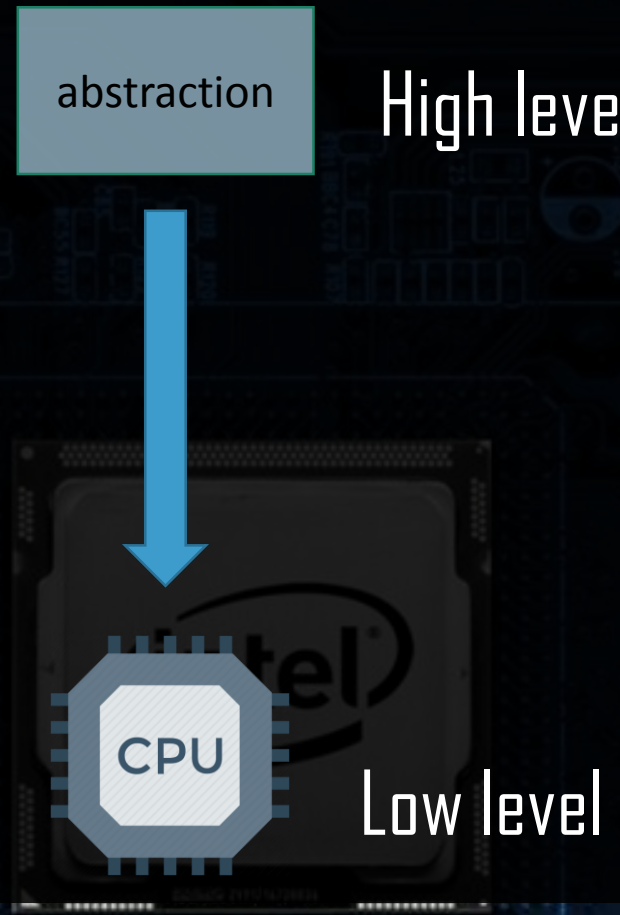


Compiling, linking, etc

- Programming languages:
  - compiled to native code (processor-specific), i.e. C/C++, assembly
  - with intermediate code (bytecode, p-code): i.e. C# (compiled to Common Intermediate Language: CIL aka MSIL), Java
  - interpreted i.e. Python, Ruby

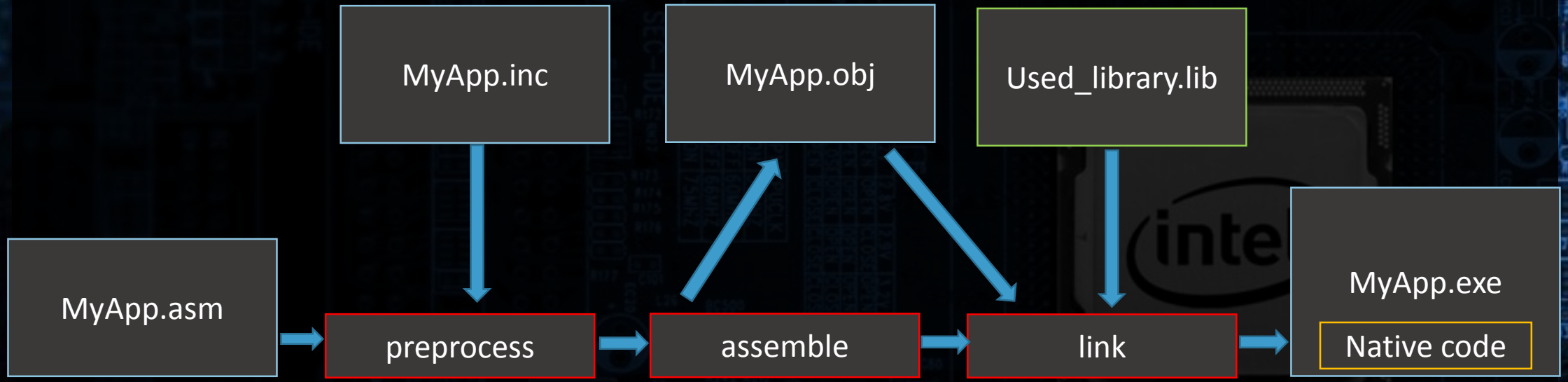
Compiling, linking, etc

- PowerShell scripts
- Python, Ruby
- Java
- C#, Visual Basic
- C/C++, Rust
- assembly



Compiling, linking, etc

- From an assembly code to a native application:
  - Preprocessing
  - Assembling
  - Linking



Compiling, linking, etc

- From an assembly code to a native application: demo in assembly
- MASM – Microsoft Macro Asembler
  - Windows-only
- YASM – independent Assembler built upon NASM (after development of NASM was suspended)
  - Multiplatform
- YASM has one advantage over MASM: allows to generate binary files (good for writing shellcodes in pure assembly)

Compiling, linking, etc

- Using YASM to create PE files
  - YASM will be used to create object file
  - LINK (from MSVC) will be used for linking

```
yasm -f win64 demo.asm

link demo.obj /entry:main /subsystem:console /defaultlib:kernel32.lib
/defaultlib:user32.lib
```

Compiling, linking, etc

- Using MASM to create PE files
  - MASM will be used to create object file
  - LINK (from MSVC) will be used for linking

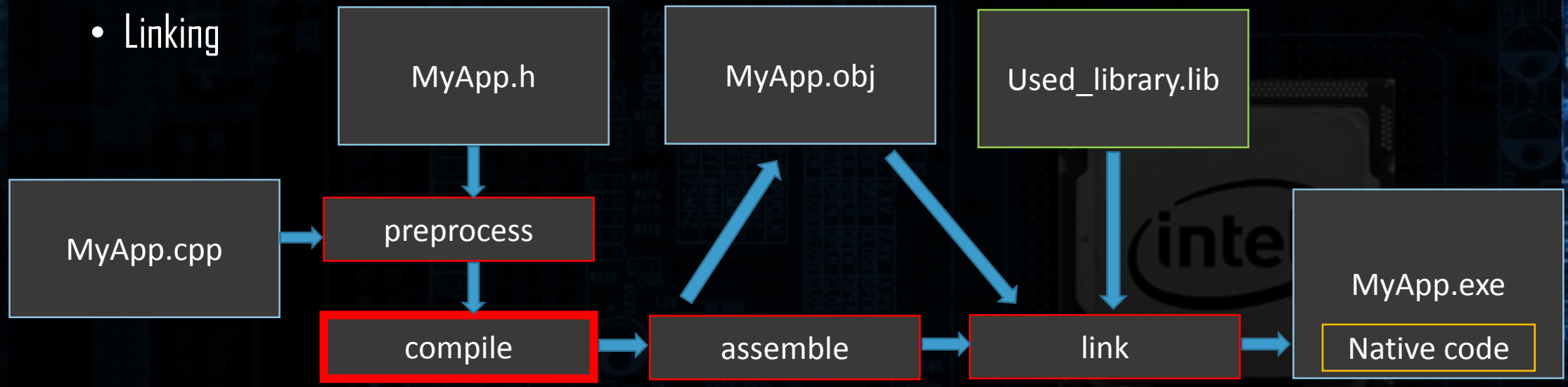
```
ml /c demo.asm

link demo.obj /entry:main /subsystem:console /defaultlib:kernel32.lib
/defaultlib:user32.lib
```

Compiling, linking, etc

- What you write is what you get: the compiled/decompiled code is identical to the assembly code that you wrote
- Assembly language is very powerful for writing shellcodes, or binary patches
- Generated binaries are much smaller than binaries generated by other languages

Compiling, linking, etc

- From a C/C++ code to a native application:
  - Preprocessing
  - Compilation
  - Assembly
  - Linking



Compiling, linking, etc

- Preprocess C++ file:

```
CL /P /C demo.cpp
```

- Using MSVC to create PE files
  - MSVC compiler: preprocess + compile: create object file
  - LINK (from MSVC) used for linking: create exe file

```
CL /c demo.cpp
LINK demo.obj /defaultlib:user32.lib
```

Compiling, linking, etc

- It is possible to supply custom linker, applying executable compression or obfuscation
- Example: Crinkler (crinkler.net)

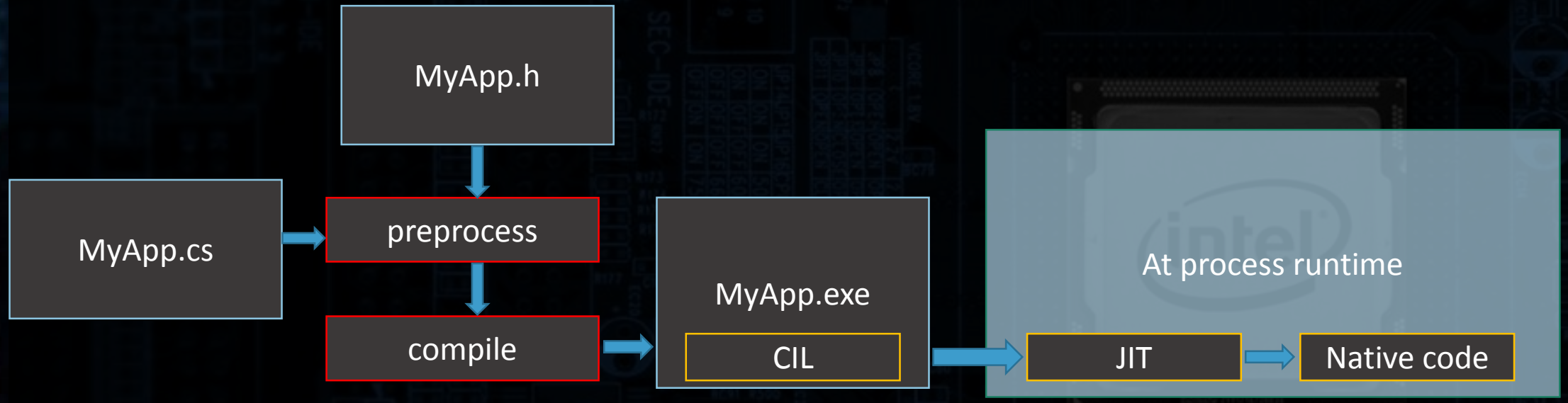
```
crinkler.exe demo.obj kernel32.lib user32.lib msvcrt.lib /ENTRY:main
```

Compiling, linking, etc

- In higher level languages the generated code depends on the compiler and its settings
- The same C/C++ code can be compiled to a differently-looking binary by different compilers
- Decompiler generated code is a reconstruction of the C/C++ code, but it can never be identical to the original one (the original code is irreversibly lost in the process of compilation)

Compiling, linking, etc

- Intermediate languages (.NET)
  - Preprocessing
  - Compilation to the intermediate code (CIL)

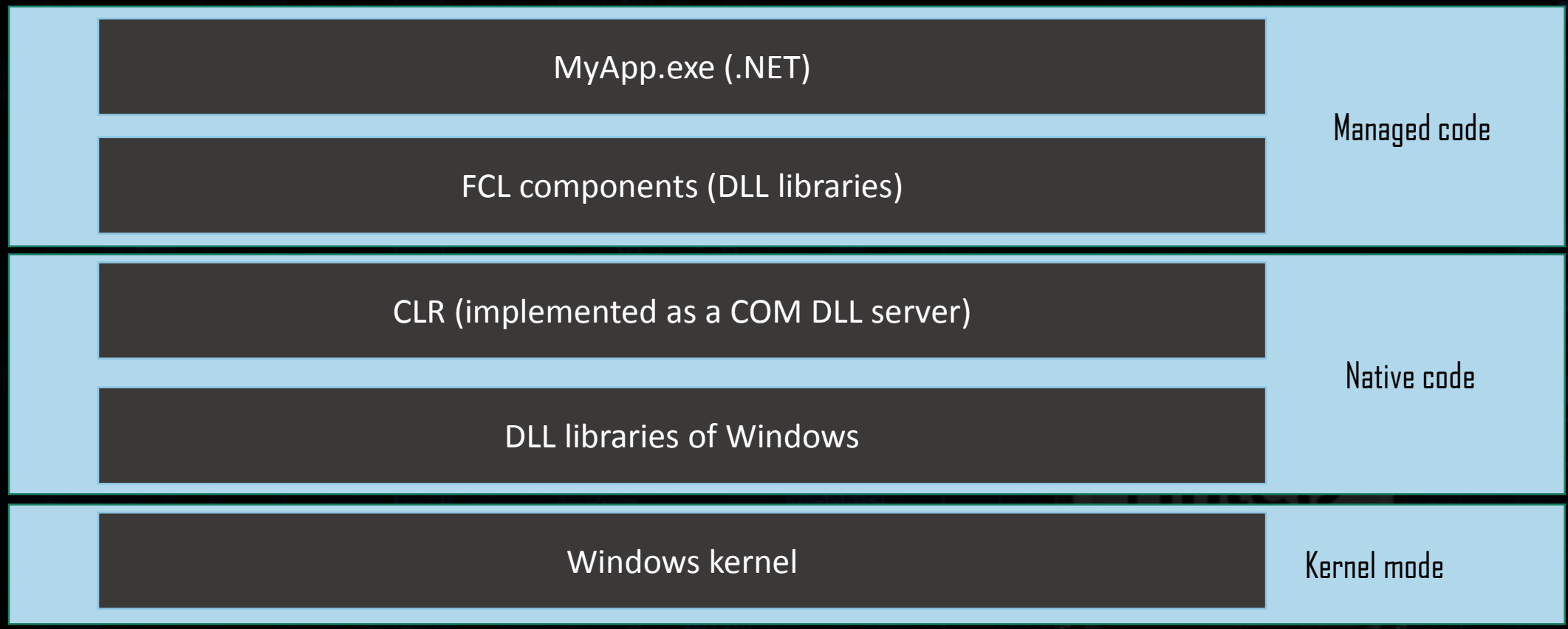


.NET framework

- In case of .NET part of the compilation is done once the executable is run (JIT – Just-In-Time)
- CLR (Common Language Runtime)
  - contains: JIT compiler (translating CIL instructions to machine code), garbage collector, etc
- FCL (Framework Class Library)
  - a collection of types implementing functionallity

https://www.geeksforgeeks.org/net-framework-class-library-fcl/

.NET framework



Based on: „Windows Internals Part 1 (7th Edition)"

Exercise

- Compile supplied examples from a commandline, with steps divided (separate compiling and linking).
  - In case of C files, see the generated assembly
  - In case of assembly and C, see the OBJ files
- See the final executables under dedicated tools:
  - PE-bear
  - dnSpy
- Notice, that files written in assembly are much smaller, and contain exactly the code that we wrote