JITHook

.NET JIT Compilation Hooking Technique and Its Packer / Unpacker

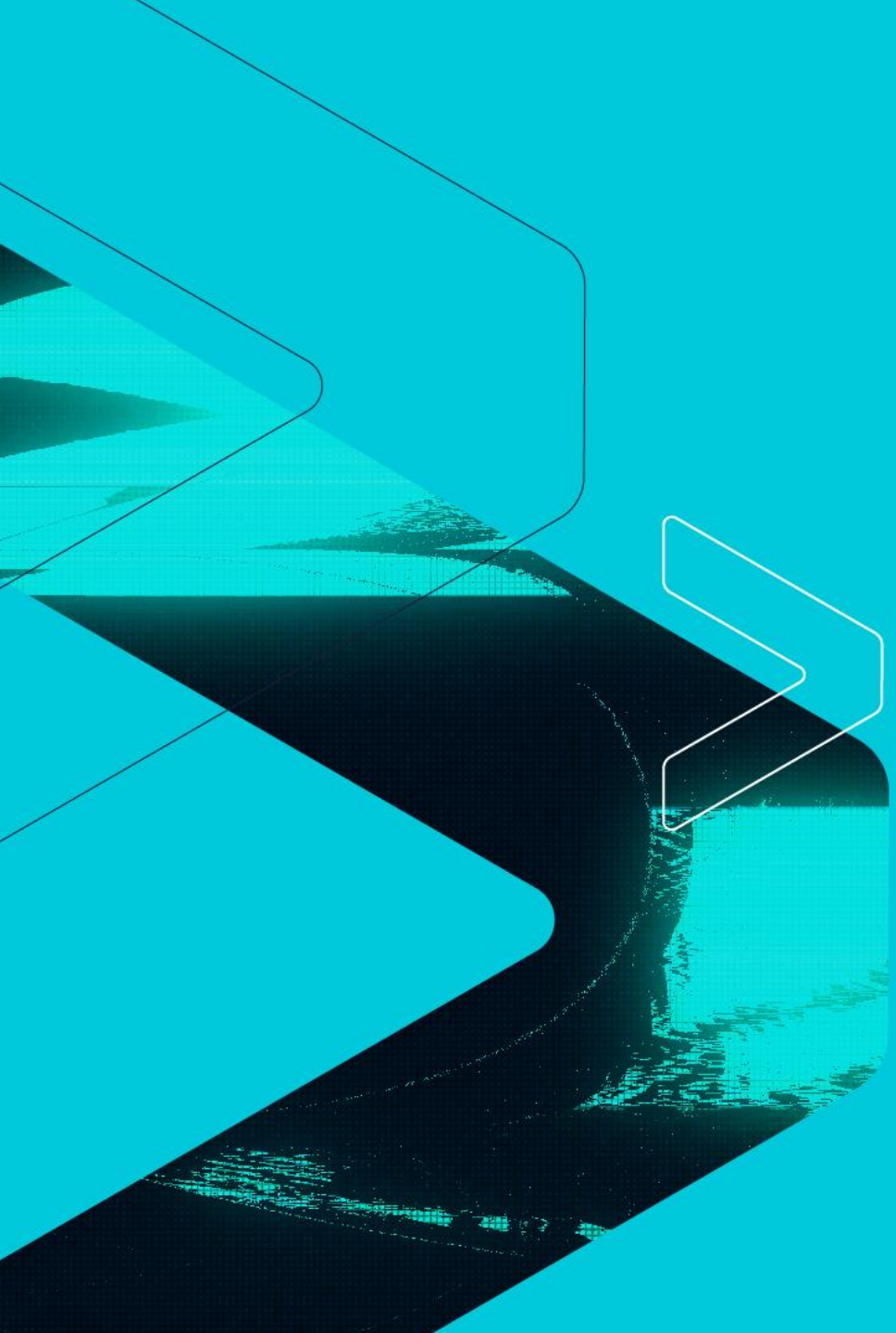
CYCRAFT

Whoami

- Shu-Ming Chang (@LJP-TW)
- Intern at CyCraft
- SQLab@NYCU
- Pwn / Reverse / Cat <3
- CTF Team
  - 10sec
  - XxTSJxX

Outline

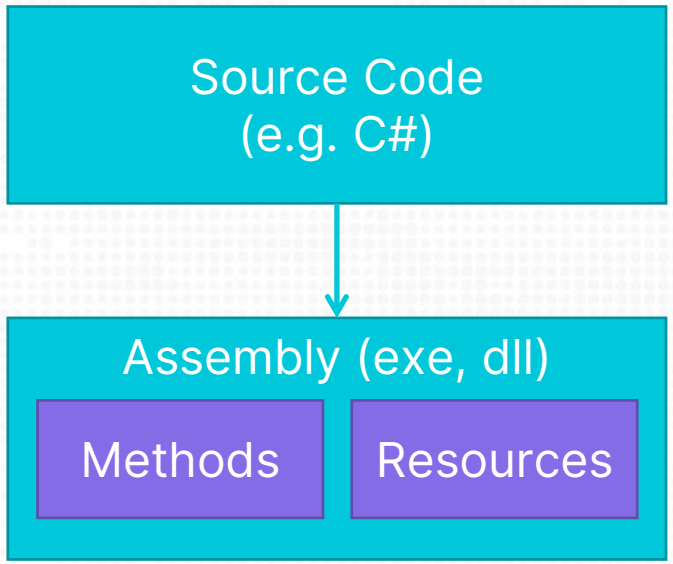
- .NET Concepts
- .NET Packers
- JITHook
- JITPacker
- JITUnpacker
- Evaluation

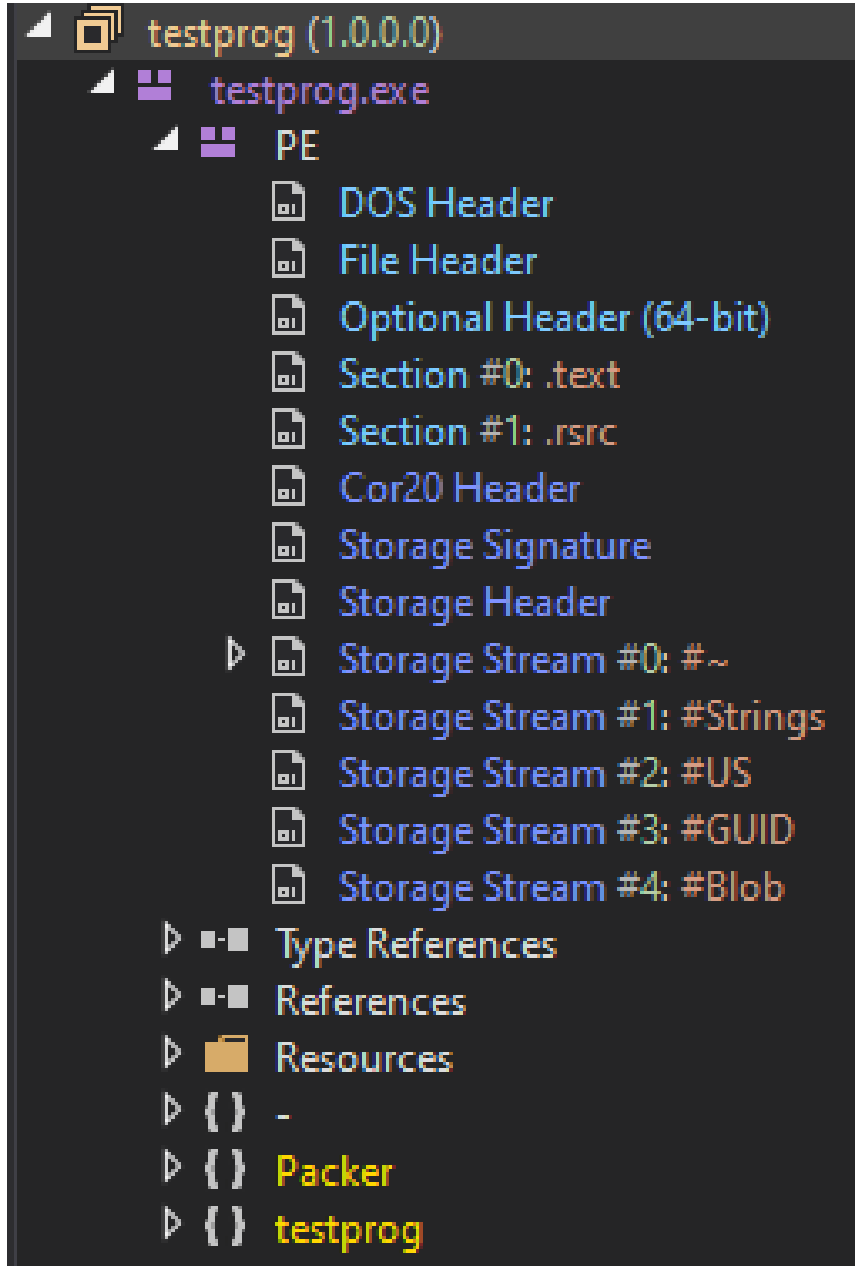


.NET Concepts

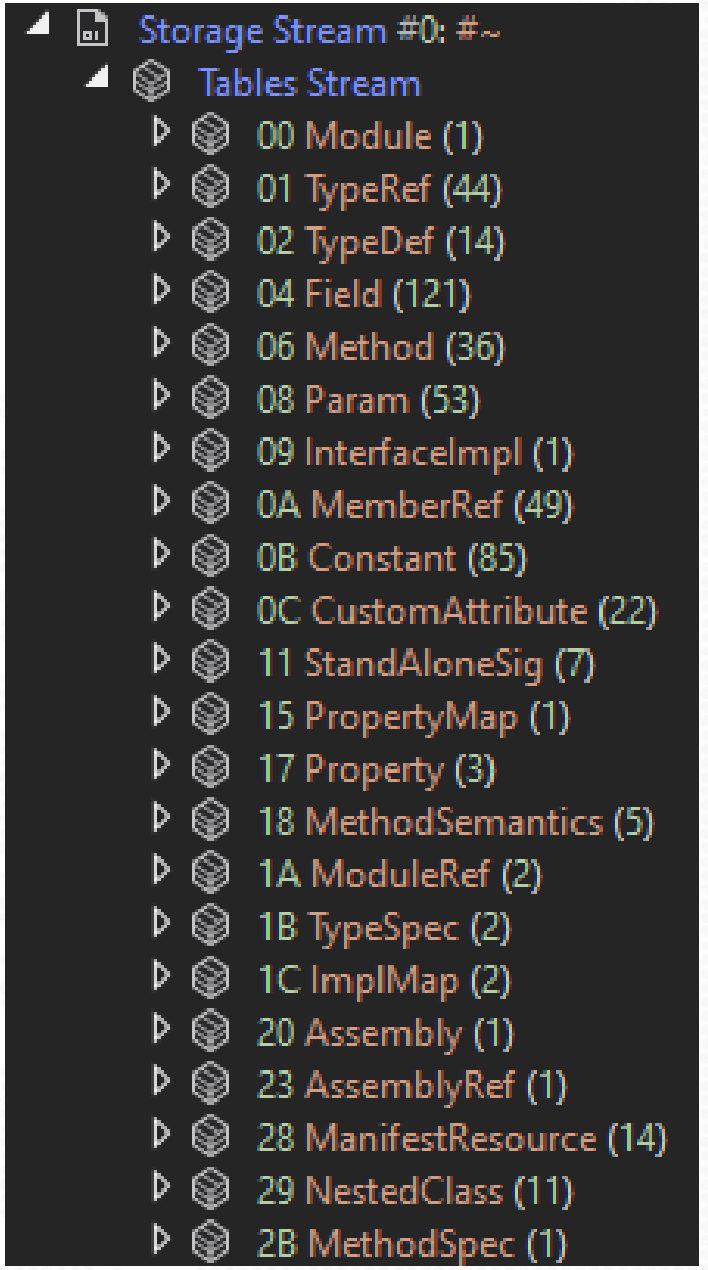
How .NET works?

- Compile source code to CIL (Common Intermediate Language), stored in assembly
- The assembly stores a lot of stuff
  - MethodDef
  - Param
  - ManifestResource
  - ModuleRef
  - ImplMap





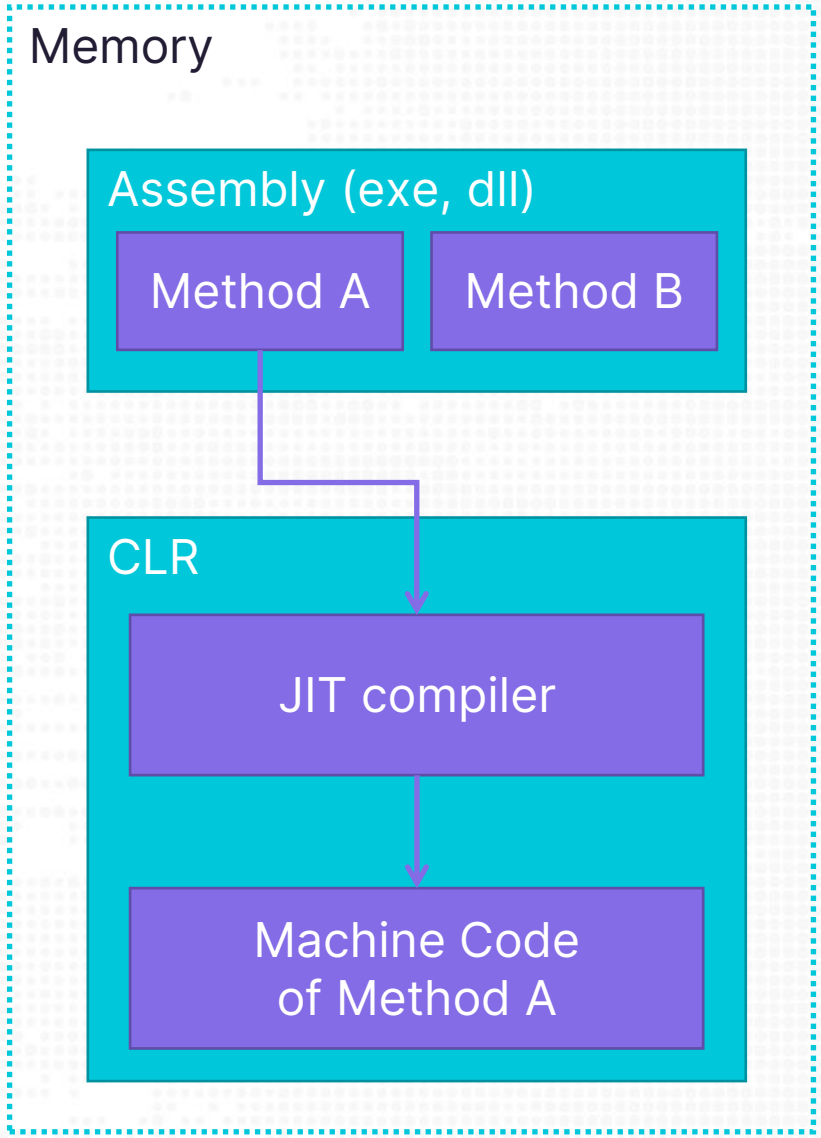
Inspect assembly with dnSpy

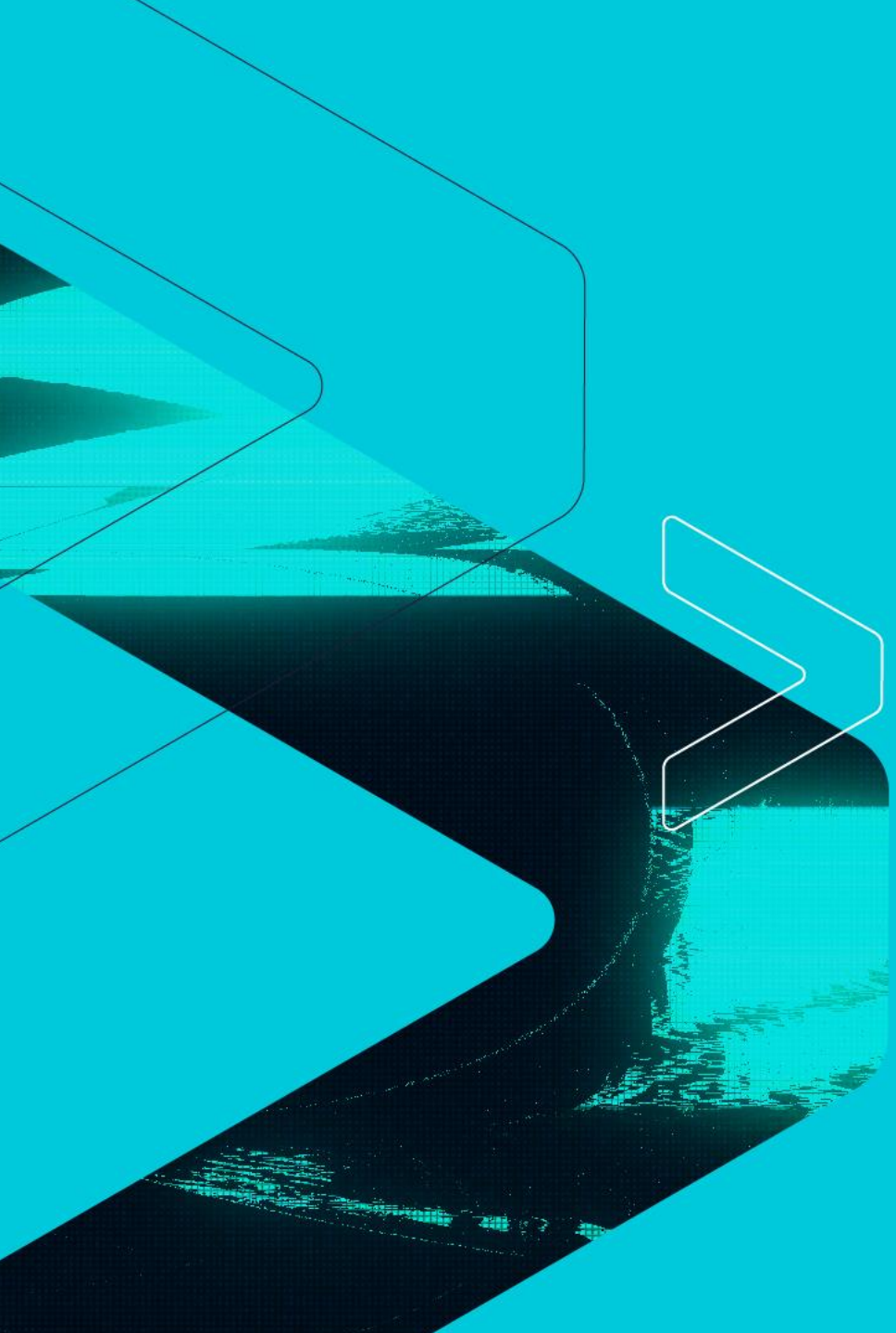


Stream #~

How .NET works?

- CLR (Common Language Runtime)
  - JIT compiling CIL to machine code
  - Managing codes





.NET Packers

.NET Packers

- Obfuscate/Encrypt the original CIL in assembly
- Restore the original CIL at runtime

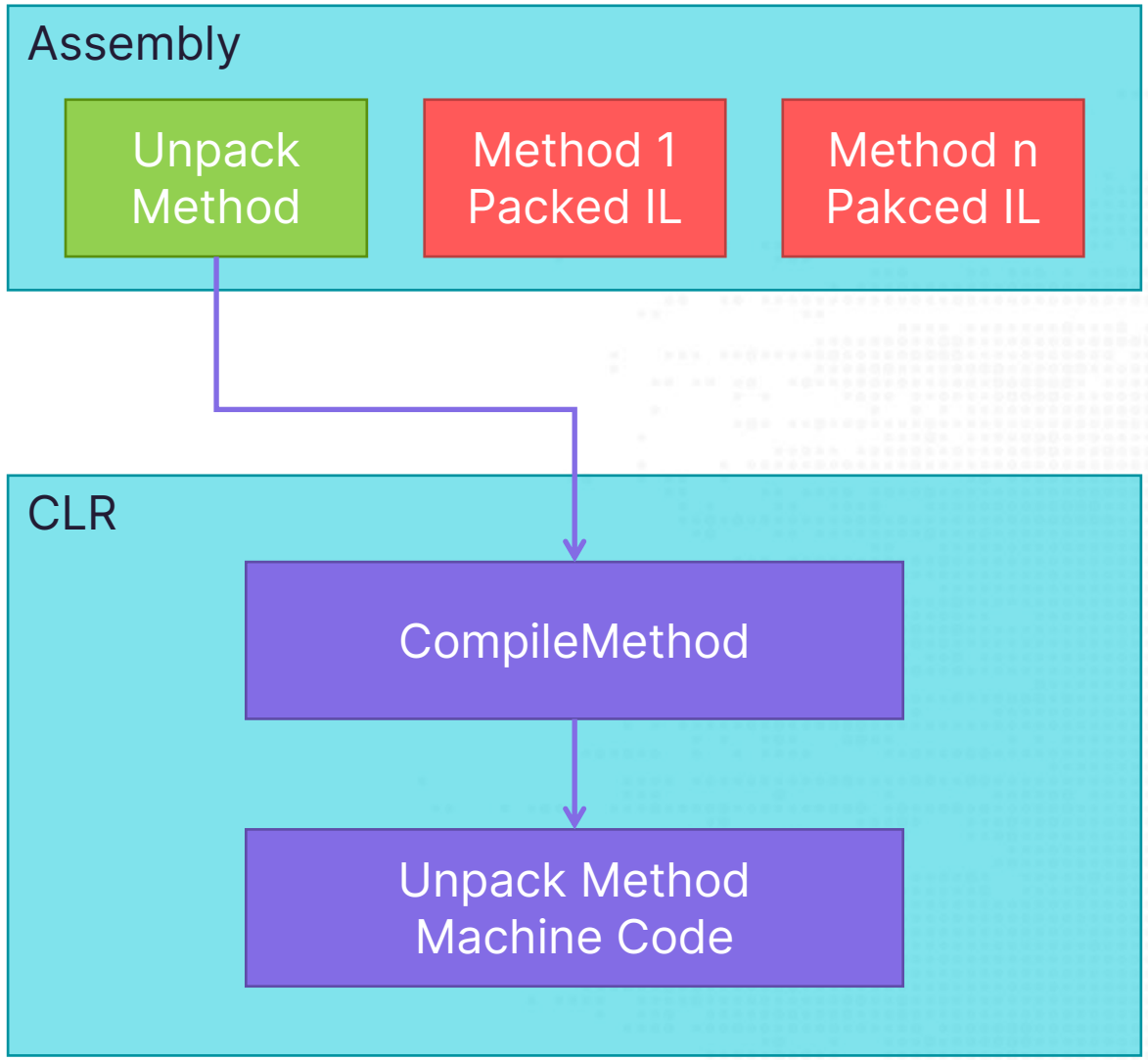
.NET Packers

- Categorize packers based on how they restore the CIL
- There are 2 types of packers
  - Type 1: Restore CIL before it's been JIT compiled
    - e.g. ConfuserEx
  - Type 2: Hook JIT compiler's method, specifically compileMethod.
    - e.g. .NET Reactor
    - JITHook

.NET Packers

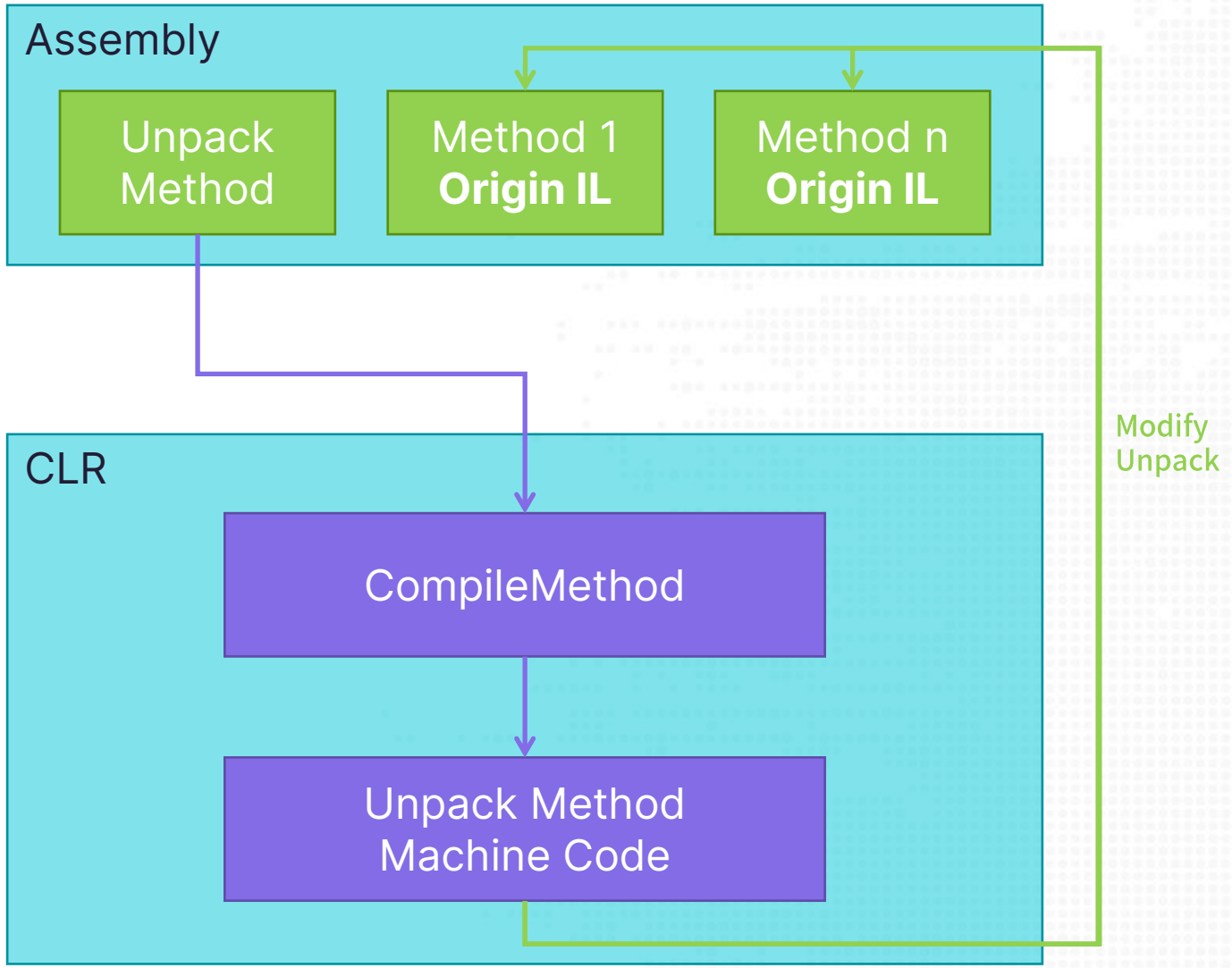
- Categorize packers based on how they restore the CIL
- There are 2 types of packers
  - Type 1: Restore CIL before it's been JIT compiled
    - e.g. ConfuserEx
  - Type 2: Hook JIT compiler's method, specifically compileMethod.
    - e.g. .NET Reactor
    - JITHook

Type 1 Packer: Restore CIL before it's been JIT compiled



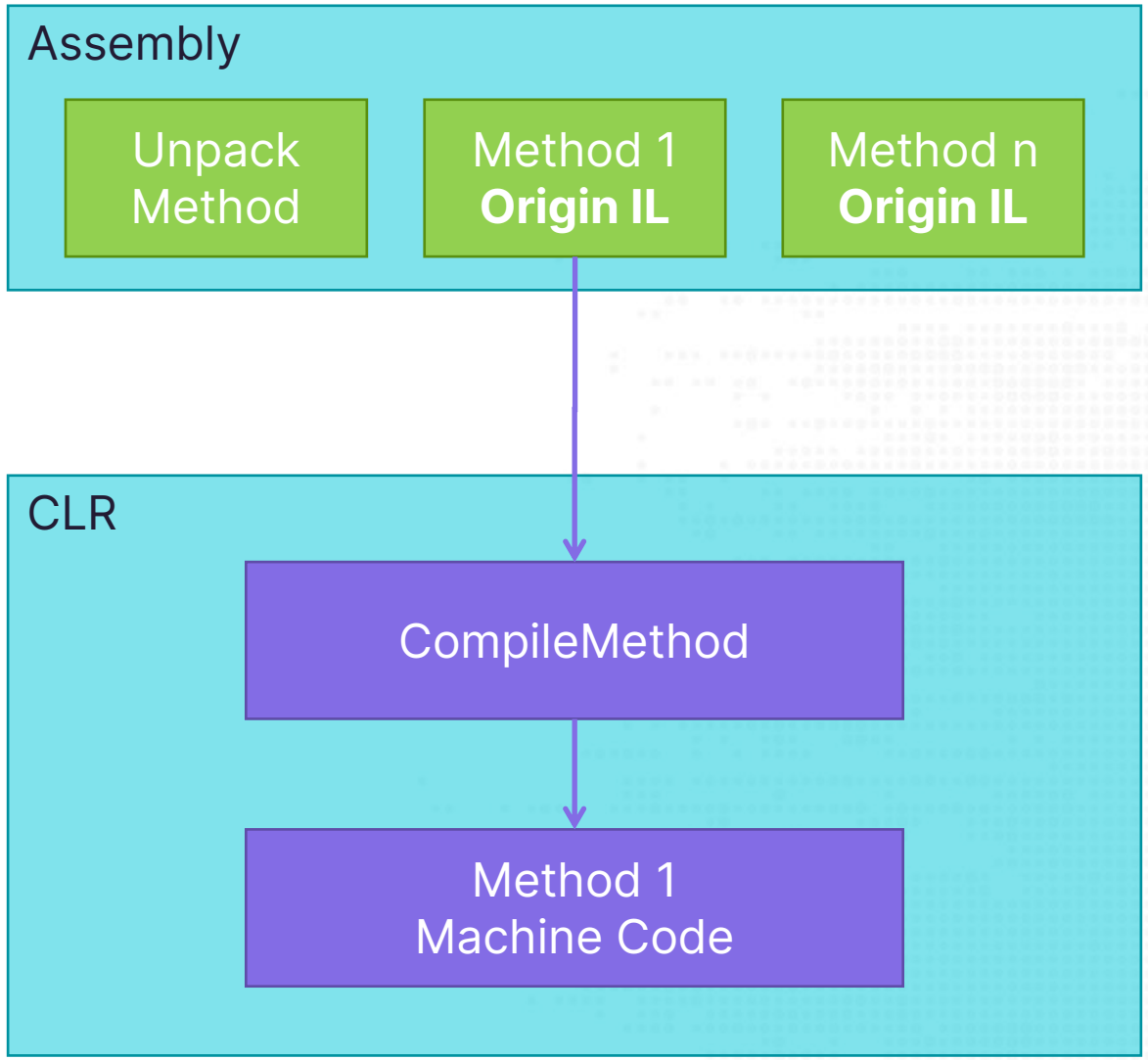
Type 1 Packer: Restore CIL before it's been JIT compiled

Unpack Method restores all others methods IL



Type 1 Packer: Restore CIL before it's been JIT compiled

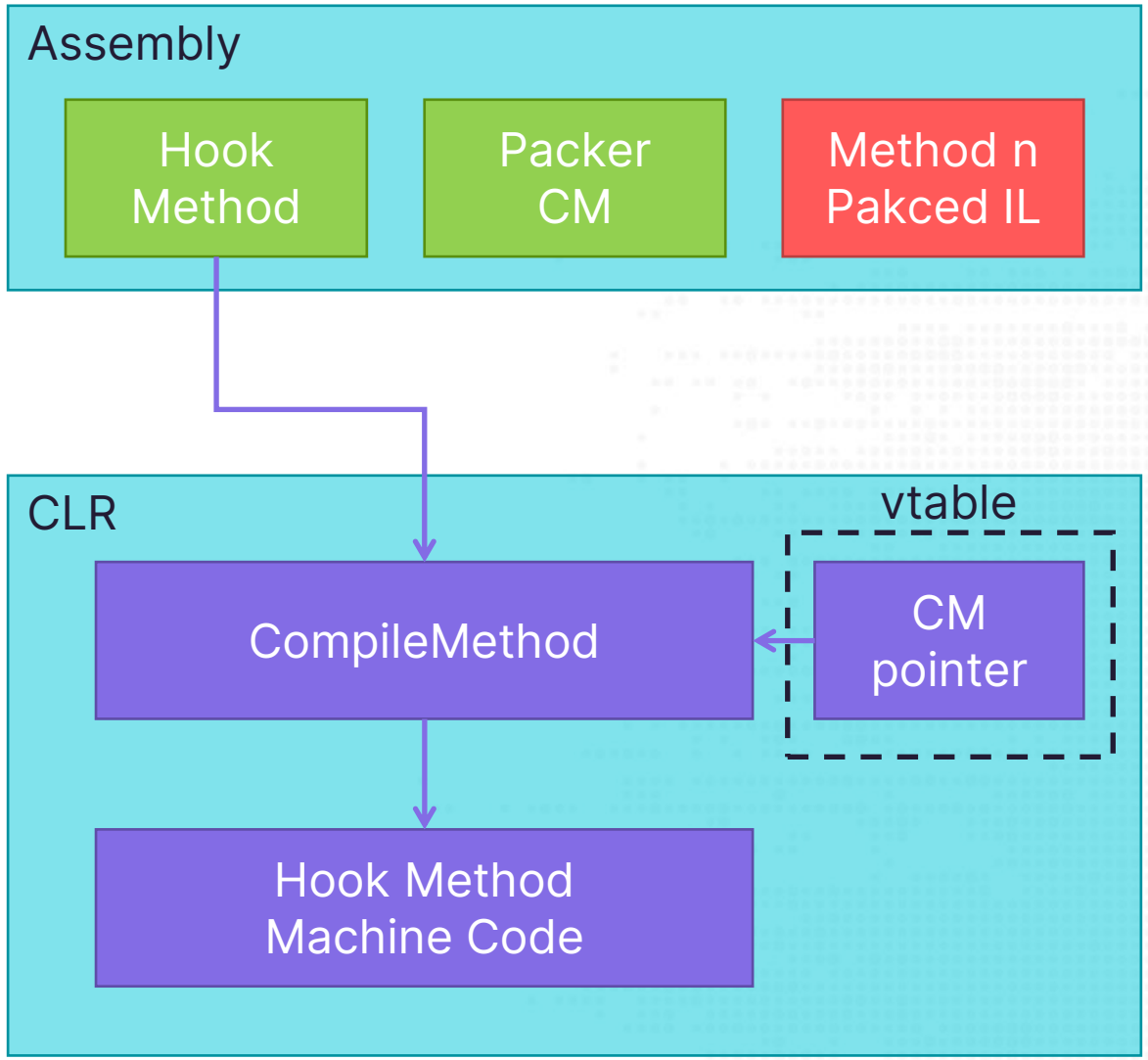
Then the program will run normally



.NET Packers

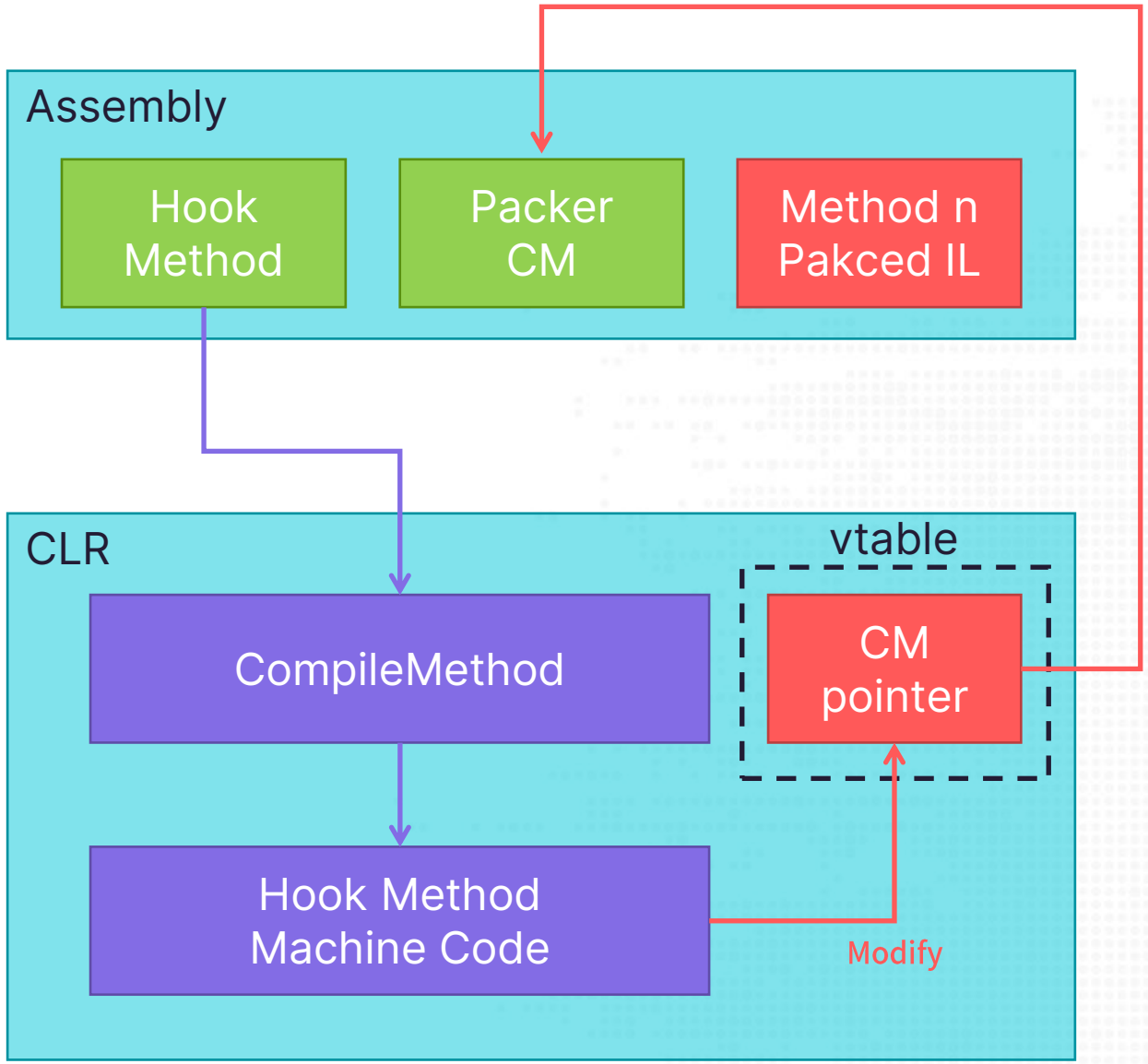
- Categorize packers based on how they restore the CIL
- There are 2 types of packers
  - Type 1: Restore CIL before it's been JIT compiled
    - e.g. ConfuserEx
  - Type 2: Hook JIT compiler's method, specifically compileMethod.
    - e.g. .NET Reactor
    - JITHook

Type 2 Packer: JITHook



Type 2 Packer: JITHook

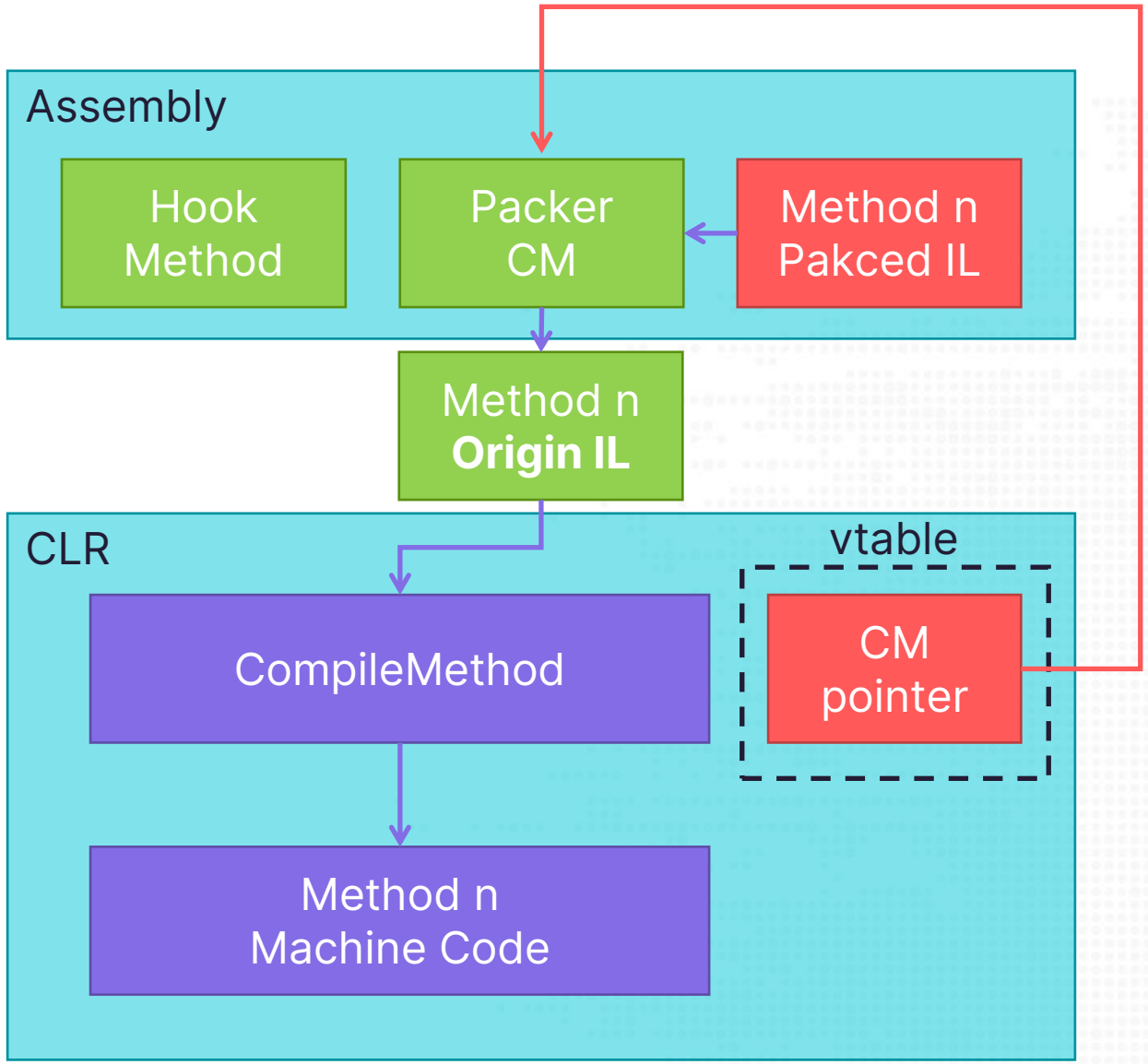
Hook CompileMethod to Packer CM



Type 2 Packer: JITHook

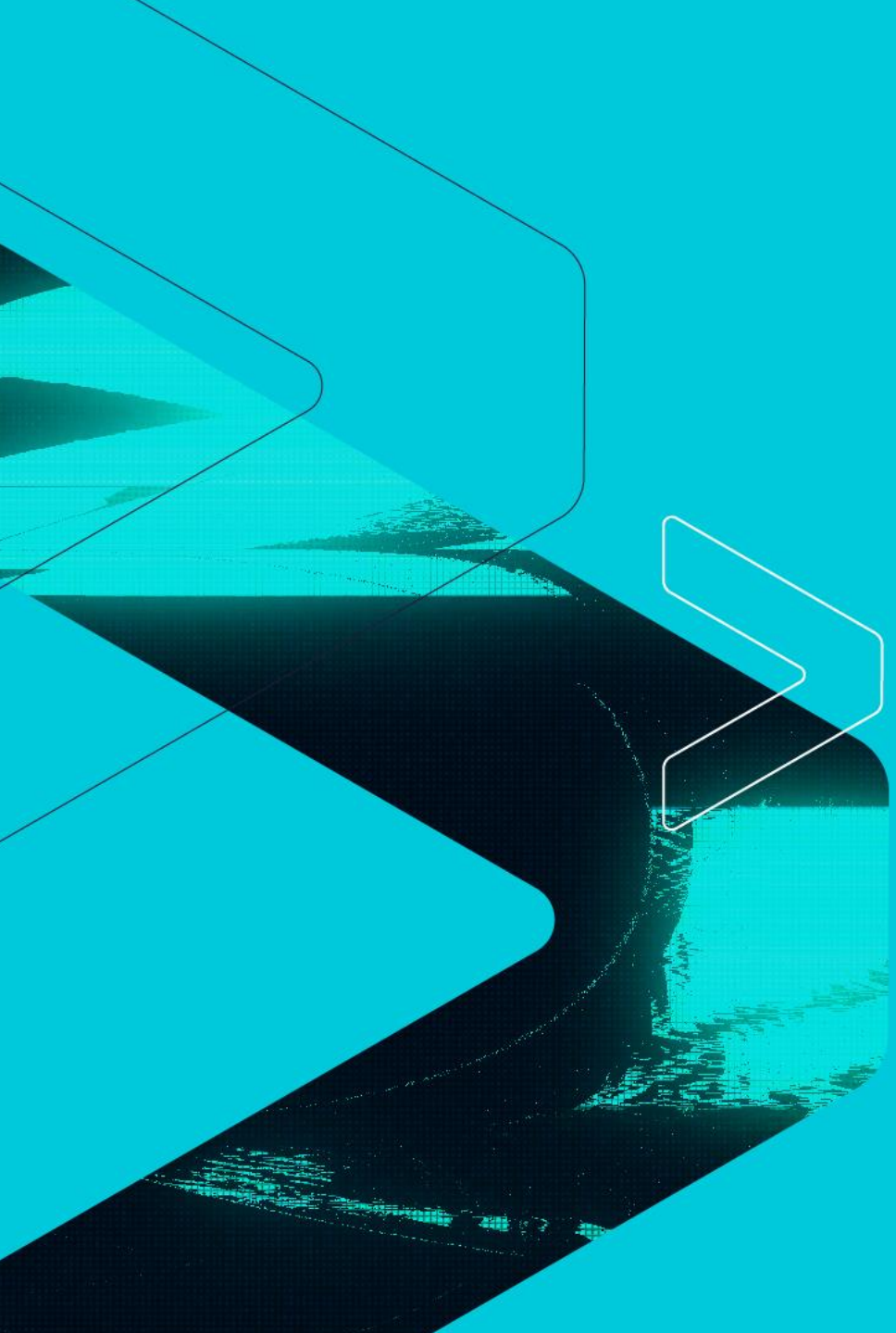
Packer CM can allocate a space to put original IL

Send it to original CompileMethod



Unpacker

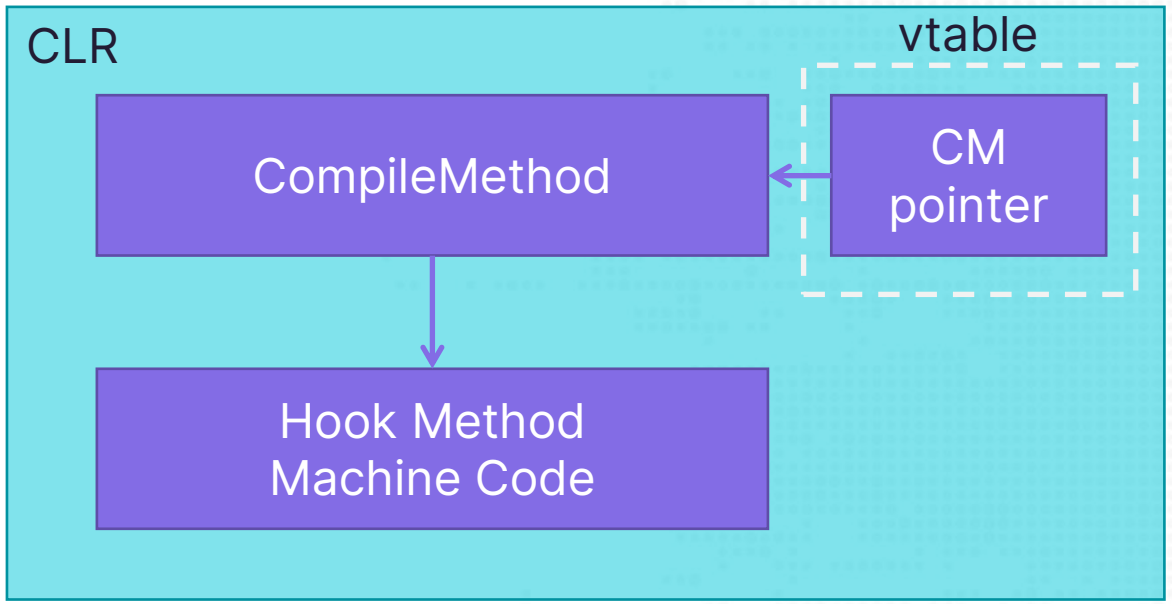
- Type 1: Restore CIL before it’s been JIT compiled
  - Dump unpacked assembly from memory at runtime
- Type 2: JITHook
  - That's what we're going to deal with



JITHook

JITHook

- Let’s see the definition of the class that has this vtable
- .NET source code → Github: dotnet/runtime
- ICorJitCompiler



```
class ICorJitCompiler
{
public:
    virtual CorJitResult compileMethod (
            ICorJitInfo                 *comp,              /* IN */
            struct CORINFO_METHOD_INFO *info,               /* IN */
            unsigned /* code:CorJitFlag */   flags,         /* IN */
            uint8_t                      **nativeEntry,       /* OUT */
            uint32_t                     *nativeSizeOfCode    /* OUT */
            ) = 0;

    virtual void ProcessShutdownWork(ICorStaticInfo *info) {};

    virtual void getVersionIdentifier(
            GUID *versionIdentifier   /* OUT */
            ) = 0;

    virtual unsigned getMaxIntrinsicSIMDVectorLength(CORJIT_FLAGS cpuCompileFlags) {
            return 0; }

    virtual void setTargetOS(CORINFO_OS os) = 0;
};
```

```
class ICorJitCompiler
{
public:
    virtual CorJitResult compileMethod (
            ICorJitInfo                 *comp,              /* IN */
            struct CORINFO_METHOD_INFO  *info,              /* IN */
            unsigned /* code:CorJitFlag */   flags,         /* IN */
            uint8_t                     **nativeEntry,        /* OUT */
            uint32_t                    *nativeSizeOfCode     /* OUT */
            ) = 0;

            GUID *versionIdentifier     /* OUT */
            ) = 0;

    virtual unsigned getMaxIntrinsicSIMDVectorLength(CORJIT_FLAGS cpuCompileFlags) {
            return 0; }

    virtual void setTargetOS(CORINFO_OS os) = 0;
};
```

compileMethod is the main routine to
ask the JIT Compiler to create native code for a method

CORINFO_METHOD_INFO

```
// * In the 32 bit jit this is implemented by code:CILJi
// * For the 64 bit jit this is implemented by code:PreJ
// Note: setTargetOS must be called before this api is u
virtual CorJitResult compileMethod (
        ICorJitInfo                 *comp,
        struct CORINFO_METHOD_INFO  *info,
        unsigned /* code:CorJitFlag */   flags,
        uint8_t                        **nativeEntry,
        uint32_t                       *nativeSizeOfCode
        ) = 0;
```

```
struct CORINFO_METHOD_INFO
{
    CORINFO_METHOD_HANDLE       ftn;
    CORINFO_MODULE_HANDLE       scope;
    uint8_t *                   ILCode;
    unsigned                    ILCodeSize;
    unsigned                    maxStack;
    unsigned                    EHcount;
    CorInfoOptions              options;
    CorInfoRegionKind           regionKind;
    CORINFO_SIG_INFO            args;
    CORINFO_SIG_INFO            locals;
};
```

getJit()

- How do we get the ICorJitCompiler object?
- Use getJit()! And it is exported by clrjit.dll
- Then we can know the address of the vtable
- Then we can get/set the compileMethod function pointer!

```
extern "C" ICorJitCompiler * getJit();
```

JITHook

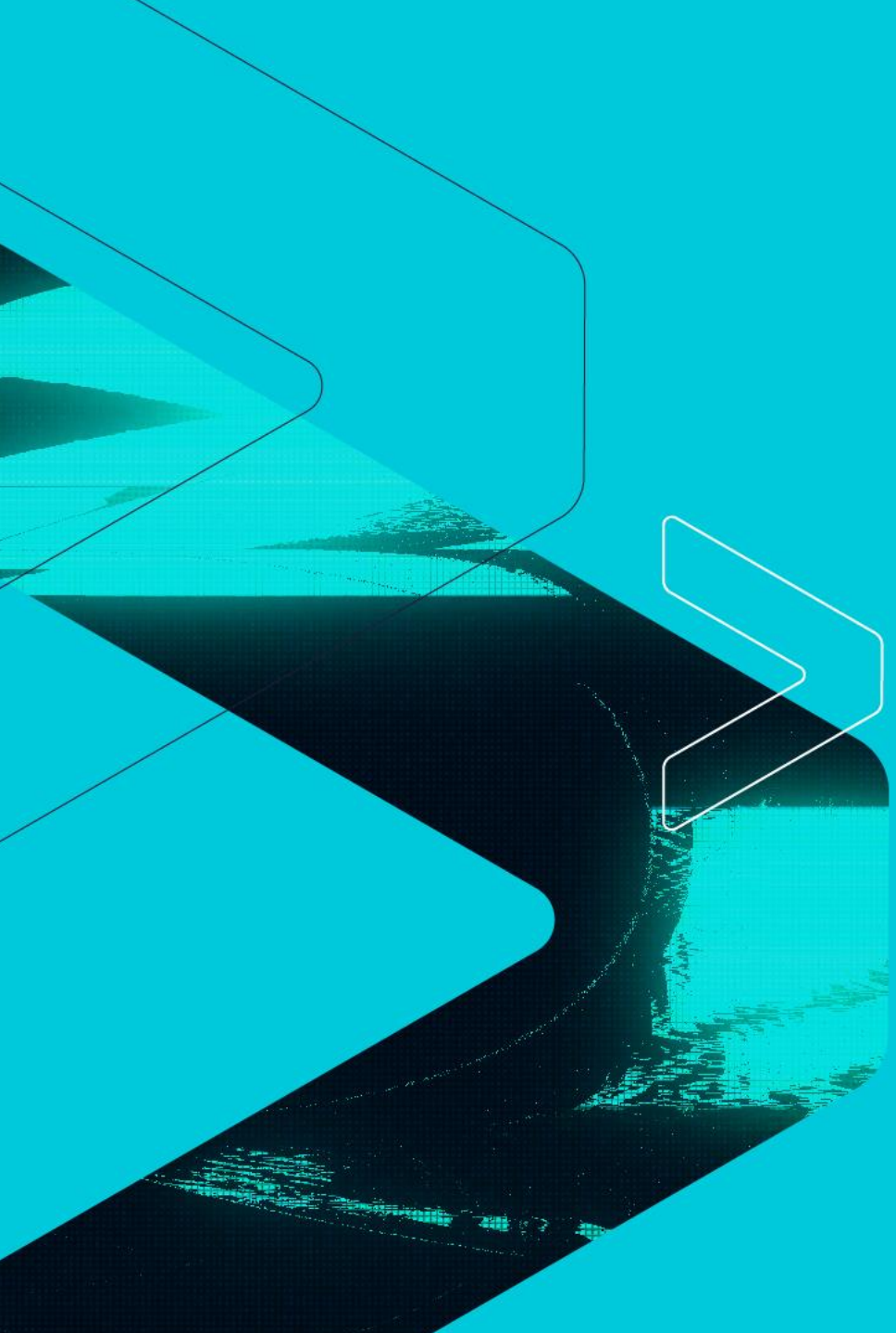
- Save the original compileMethod
- Overwrite it with compileMethodHook

```
AddDllDirectory(L"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\");
clrjit = LoadLibraryExA("clrjit.dll", NULL, LOAD_LIBRARY_SEARCH_USER_DIRS);

getjit = (func *)GetProcAddress(clrjit, "getJit");
ICorJitCompilerPtr = getjit();
ICorJitCompilerVtable = *(void ***)CorJitCompilerPtr;
originCompileMethod = (compileMethodFunc *)ICorJitCompilerVtable[0];

VirtualProtect(&ICorJitCompilerVtable[0], 0x8, PAGE_EXECUTE_READWRITE, &old);
ICorJitCompilerVtable[0] = compileMethodHook;
VirtualProtect(&ICorJitCompilerVtable[0], 0x8, old, &old);
```

compileMethod

- Do anything before calling the original compileMethod

```
CorJitResult compileMethodHook(
    void                                *thisptr,
    ICorJitInfo                         *comp,            /* IN */
    struct CORINFO_METHOD_INFO          *info,            /* IN */
    unsigned /* code:CorJitFlag */       flags,           /* IN */
    uint8_t                            **nativeEntry,     /* OUT */
    uint32_t                            *nativeSizeOfCode /* OUT */
) {
    // Do something

    return originCompileMethod(thisptr, comp, info, flags, nativeEntry, nativeSizeOfCode);
}
```



JITPacker

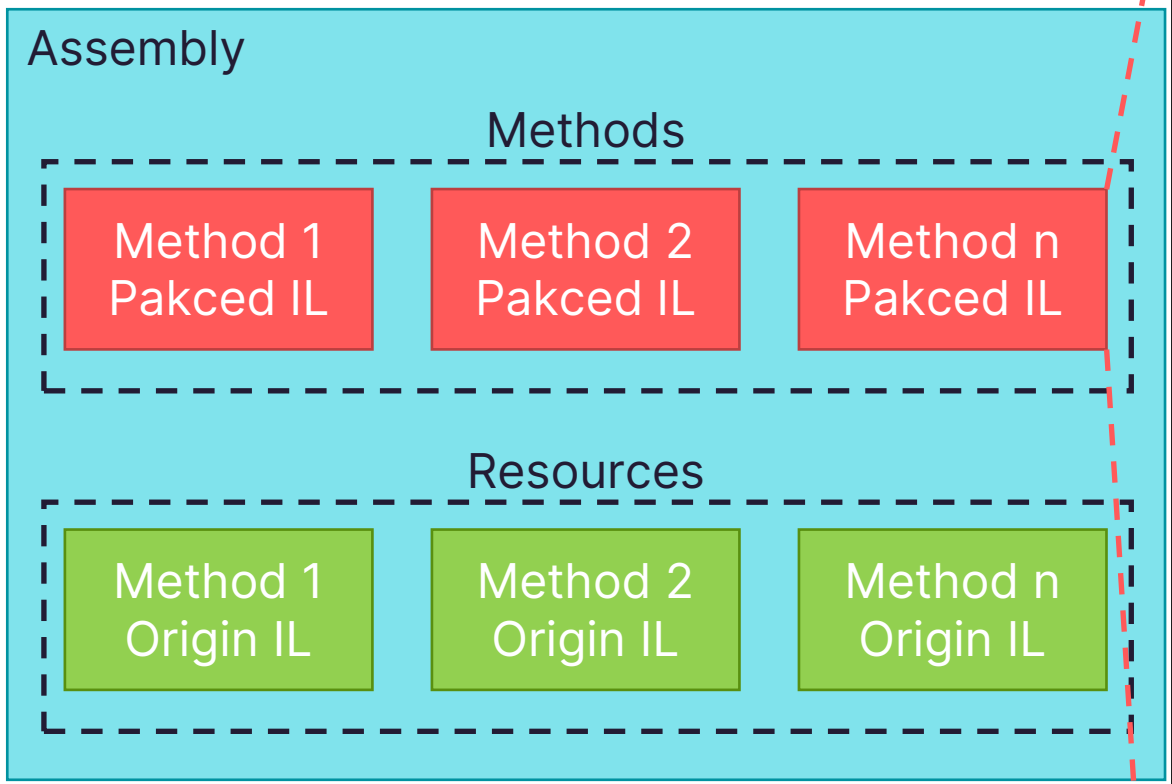
JITPacker

- We implemented JITPacker using JITHook
- Save the original CIL to the resource
- Modify most of the original CIL code to byte 0x87
- Add module initializer to the assembly to hook CM ptr to packer CM
- Restore the CIL in packer CM

JITPacker

- We implemented JITPacker using JITHook
- Save the original CIL to the resource
- Modify most of the original CIL code to byte 0x87
- Add module initializer to the assembly to hook CM ptr to packer CM
- Restore the CIL in packer CM

```
static void packMethod(TypeDef type, MethodDef method)
{
    // Get codesize

    // Read origin IL
    byte[] originILbytes = reader.ReadBytes(codesize);

    // Save origin IL to resources
    module.Resources.Add(new EmbeddedResource(method.MDToken.ToString().ToLower(),
        originILbytes,
        ManifestResourceAttributes.Private));

    for (int i = 0; i < method.Body.Instructions.Count - 1; i++) {
        // If this instruction cannot be patched
        if (...) { continue; }

        // Patch instructions to byte 0x87
        method.Body.Instructions[i] = OpCodes.Conv_Ovf_U2_Un.ToInstruction();
        for (int _ = 0; _ < inssize - 1; ++_) {
            method.Body.Instructions.Insert(i, OpCodes.Conv_Ovf_U2_Un.ToInstruction());
        }
    }
}
```



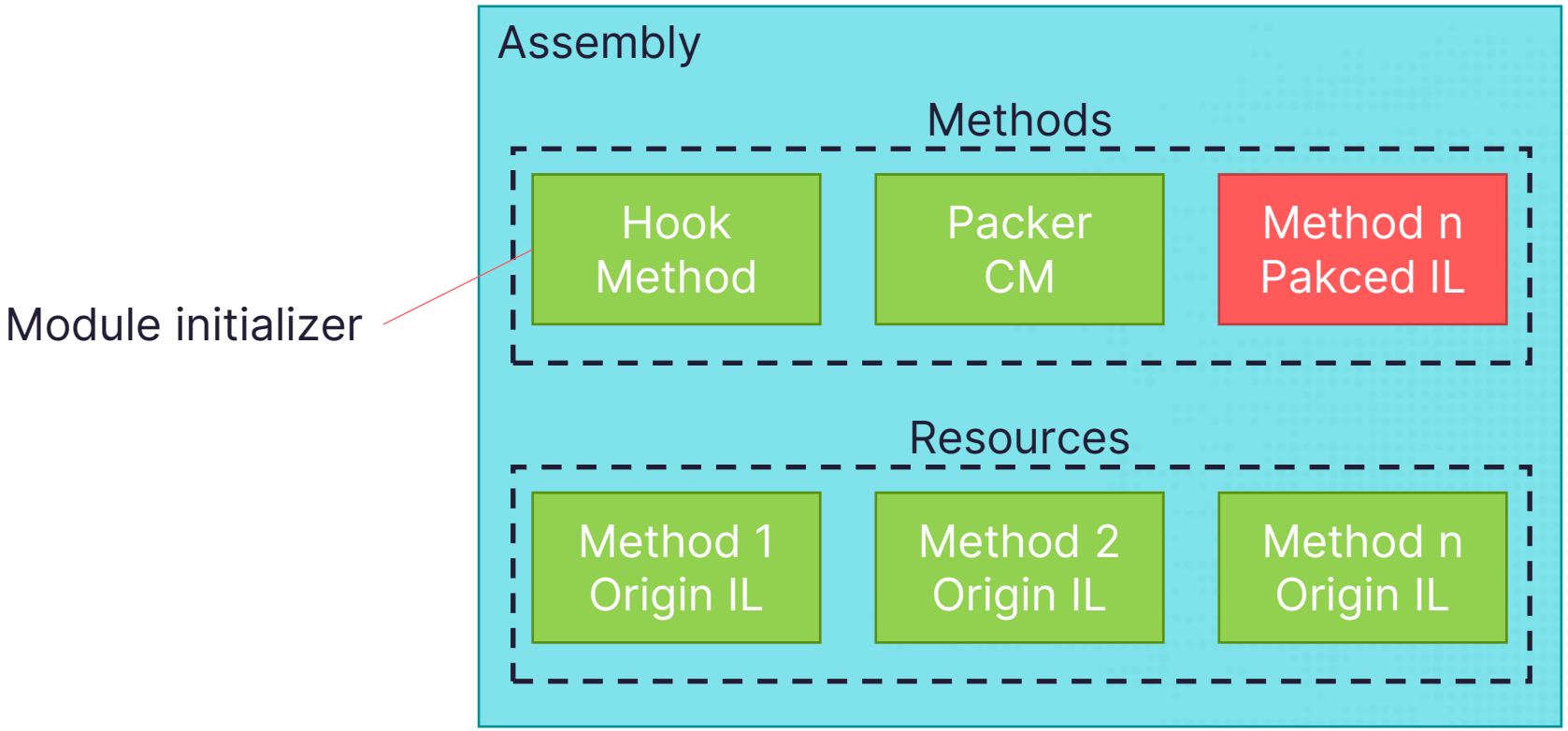

```
.method private hidebysig static
    int32 fatFunc7 (
        int32 a,
        int32 b,
        int32 c
    ) cil managed
{
    // Header Size: 12 bytes
    // Code Size: 59 (0x3B) bytes
    // LocalVarSig Token: 0x11000001 RID: 1
    .maxstack 2
    .locals init (
        [0] int32
    )

    /* 0x0000042C 87                */ IL_0000: conv.ovf.u2.un
    .try
    {
        /* 0x0000042D 00            */ IL_0001: nop
        .try
        {
            /* 0x0000042E 00        */ IL_0002: nop
            /* 0x0000042F 87        */ IL_0003: conv.ovf.u2.un
            /* 0x00000430 87        */ IL_0004: conv.ovf.u2.un
            /* 0x00000431 87        */ IL_0005: conv.ovf.u2.un
            /* 0x00000432 87        */ IL_0006: conv.ovf.u2.un
            /* 0x00000433 87        */ IL_0007: conv.ovf.u2.un
            /* 0x00000434 87        */ IL_0008: conv.ovf.u2.un
            /* 0x00000435 87        */ IL_0009: conv.ovf.u2.un
            /* 0x00000436 87        */ IL_000A: conv.ovf.u2.un
        } // end .try
```

JITPacker

- We implemented JITPacker using JITHook
- Save the original CIL to the resource
- Modify most of the original CIL code to byte 0x87
- Add module initializer to the assembly to hook CM ptr to packer CM
- Restore the CIL in packer CM

```
// Create module initializer
MethodDef cctor = new MethodDefUser(".cctor",
    MethodSig.CreateStatic(module.CorLibTypes.Void));
cctor.Attributes = MethodAttributes.Public | MethodAttributes.SpecialName |
    MethodAttributes.RTSpecialName | MethodAttributes.Static;
cctor.ImplAttributes = MethodImplAttributes.IL | MethodImplAttributes.Managed;
moduleType.Methods.Add(cctor);

// Find “entry” method of packer module
MethodDef packerEntry = packerType.FindMethod("entry");

// Call “entry” method of packer module in module initializer
var cctorILbody = new CilBody();
cctor.Body = cctorILbody;
cctorILbody.Instructions.Add(OpCodes.Call.ToInstruction(packerEntry));
cctorILbody.Instructions.Add(OpCodes.Ret.ToInstruction());
```

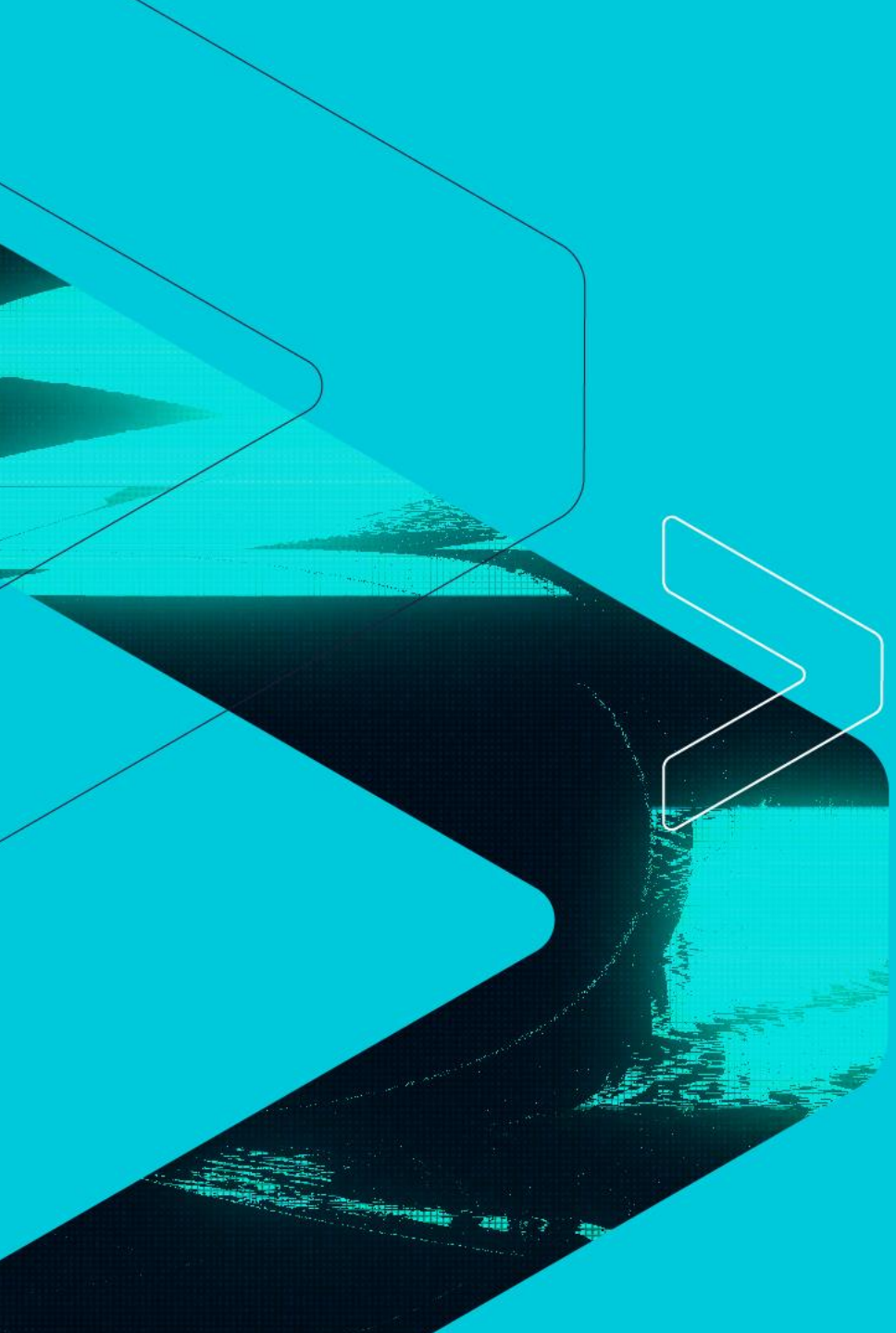
```
public static unsafe void entry() {
    // Use getJit() to get ICorJITCompiler, the get original compileMethod
    pCompileMethod = Marshal.ReadIntPtr(VTableAddr);
    OriginalCompileMethod =
        (CompileMethodDel64)Marshal.GetDelegateForFunctionPointer(
            Marshal.ReadIntPtr(pCompileMethod),
            typeof(CompileMethodDel64));

    // Pre-compile some functions
    RuntimeHelpers.PrepareMethod(
        typeof(Console).GetMethod("WriteLine", new[] { typeof(string) }).MethodHandle);

    // Overwrite compileMethod pointer in vtable of ICorJITCompiler
    if (!VirtualProtect(pCompileMethod, (uint)IntPtr.Size,
            Protection.PAGE_EXECUTE_READWRITE, out old))
        throw new Exception("[!] Cannot change memory protection flags.");

    Marshal.WriteIntPtr(pCompileMethod,
        Marshal.GetFunctionPointerForDelegate(packerCM));

    VirtualProtect(pCompileMethod, (uint)IntPtr.Size, (Protection)old, out old);
}
```



JITPacker

- We implemented JITPacker using JITHook
- Save the original CIL to the resource
- Modify most of the original CIL code to byte 0x87
- Add module initializer to the assembly to hook CM ptr to packer CM
- Restore the CIL in packer CM

```
private static unsafe int packerCM(IntPtr thisPtr, [In] IntPtr corJitInfo,
                [In] CorMethodInfo64* methodInfo, CorJitFlag flags,
                [Out] IntPtr nativeEntry, [Out] IntPtr nativeSizeOfCode) {
    // Calculate methodToken
    string methodToken = (0x06000000 + *(ushort*)methodInfo->ftn).ToString("x8");

    // Try to get original CIL from resource
    Assembly assembly = System.Reflection.Assembly.GetExecutingAssembly();
    System.IO.Stream stream = assembly.GetManifestResourceStream(methodToken);
    if (stream == null) {
        return OriginalCompileMethod(...);
    }

    // Patch
    byte[] newil = new byte[stream.Length];
    stream.Read(newil, 0, newil.Length);

    IntPtr ilCodeHandle = Marshal.AllocHGlobal(newil.Length);
    Marshal.Copy(newil, 0, ilCodeHandle, newil.Length);

    methodInfo->ilCode = (byte*)ilCodeHandle.ToPointer();
    methodInfo->ilCodeSize = (uint)newil.Length;
    return OriginalCompileMethod(thisPtr, corJitInfo, methodInfo,
                                 flags, nativeEntry, nativeSizeOfCode); }
```



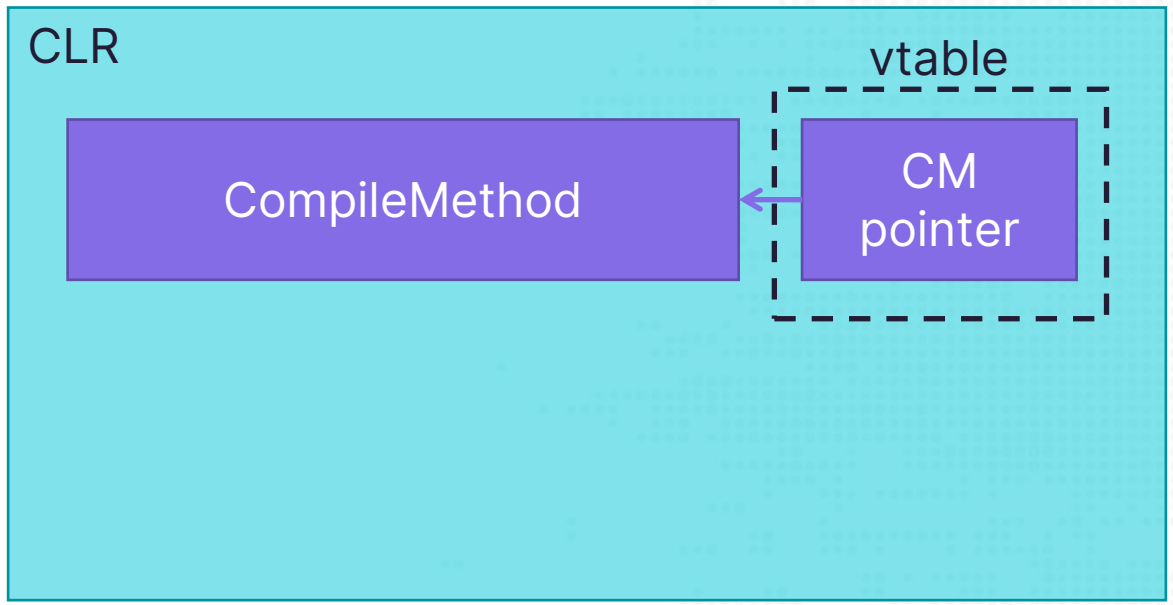
JITUnpacker

Unpack JITHook .NET assembly

- Hook CompileMethod before it’s been hooked
- Get the unpacked CIL from Packer CM
- Rebuild the unpacked assembly

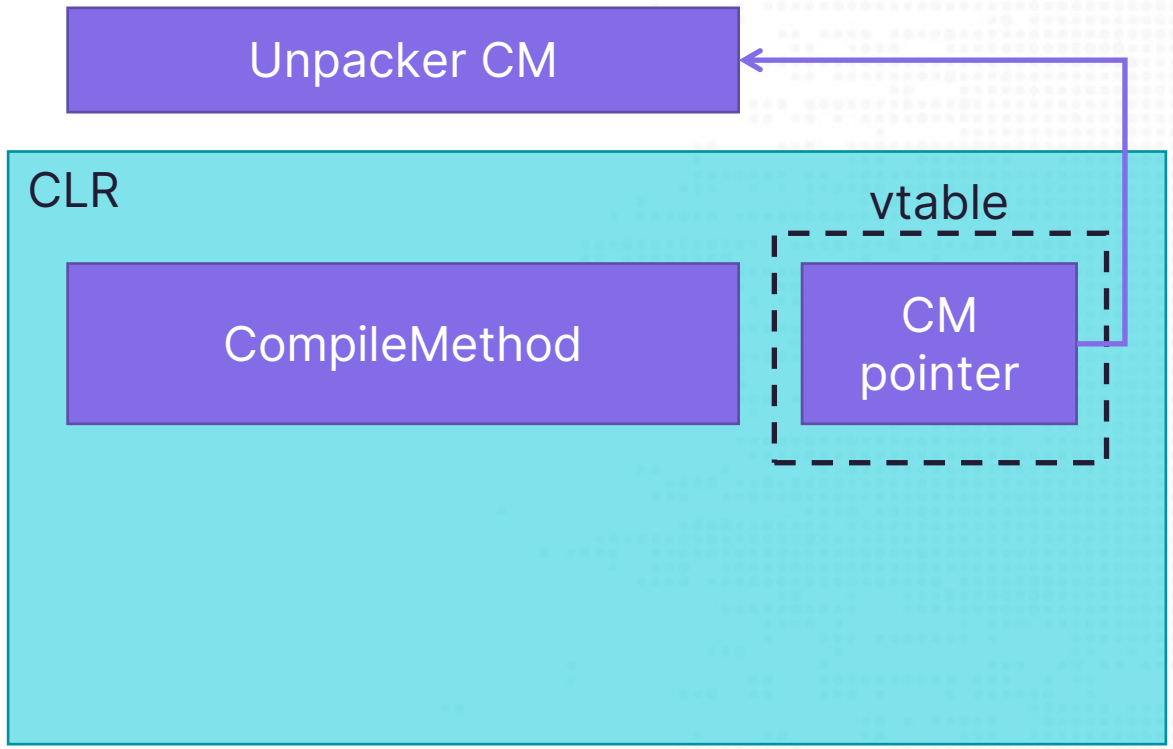
JITUnpacker

Prepare CLR with
CLR Hosting

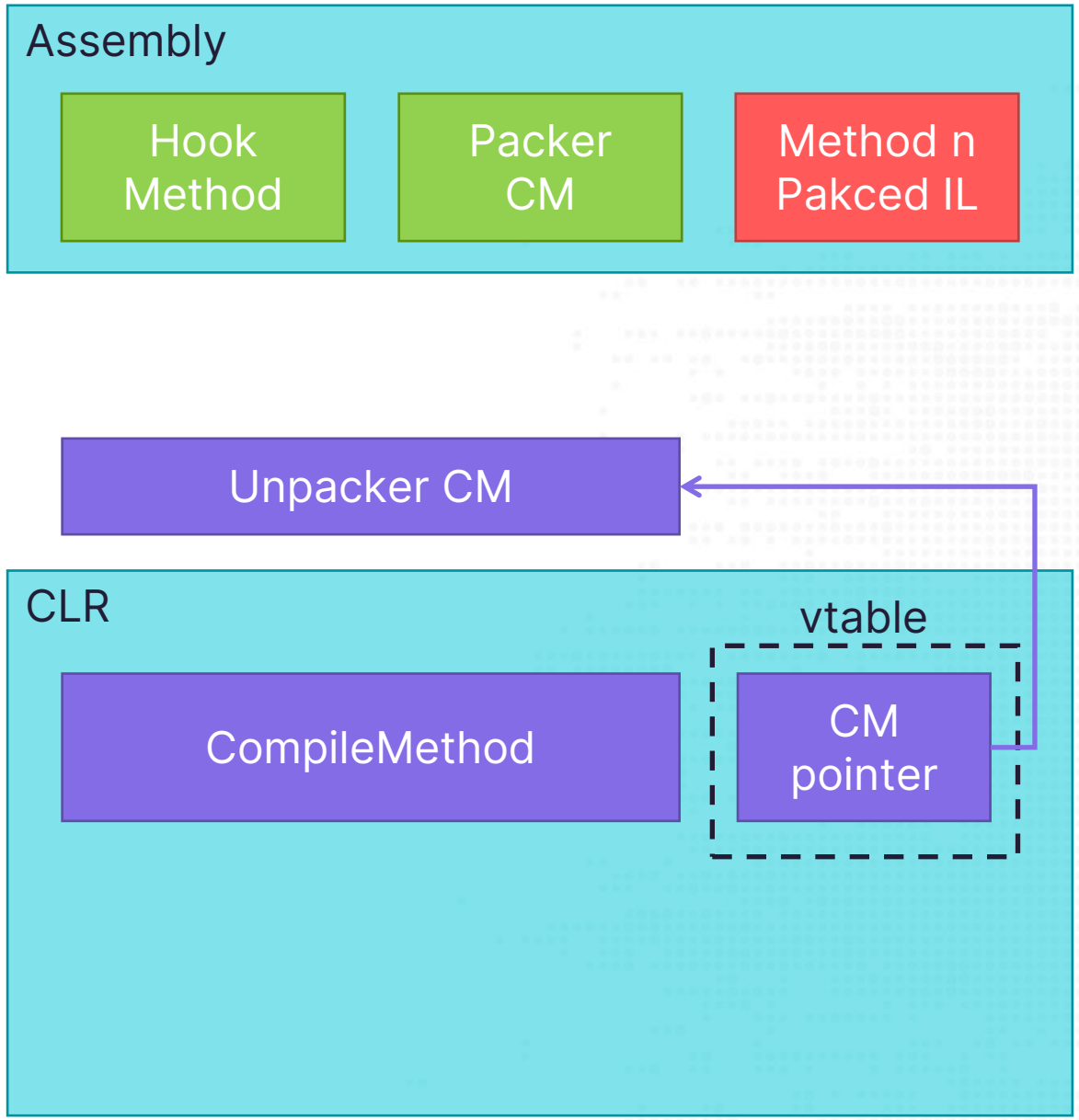


JITUnpacker

Hook CompileMethod to **Unpacker CM**

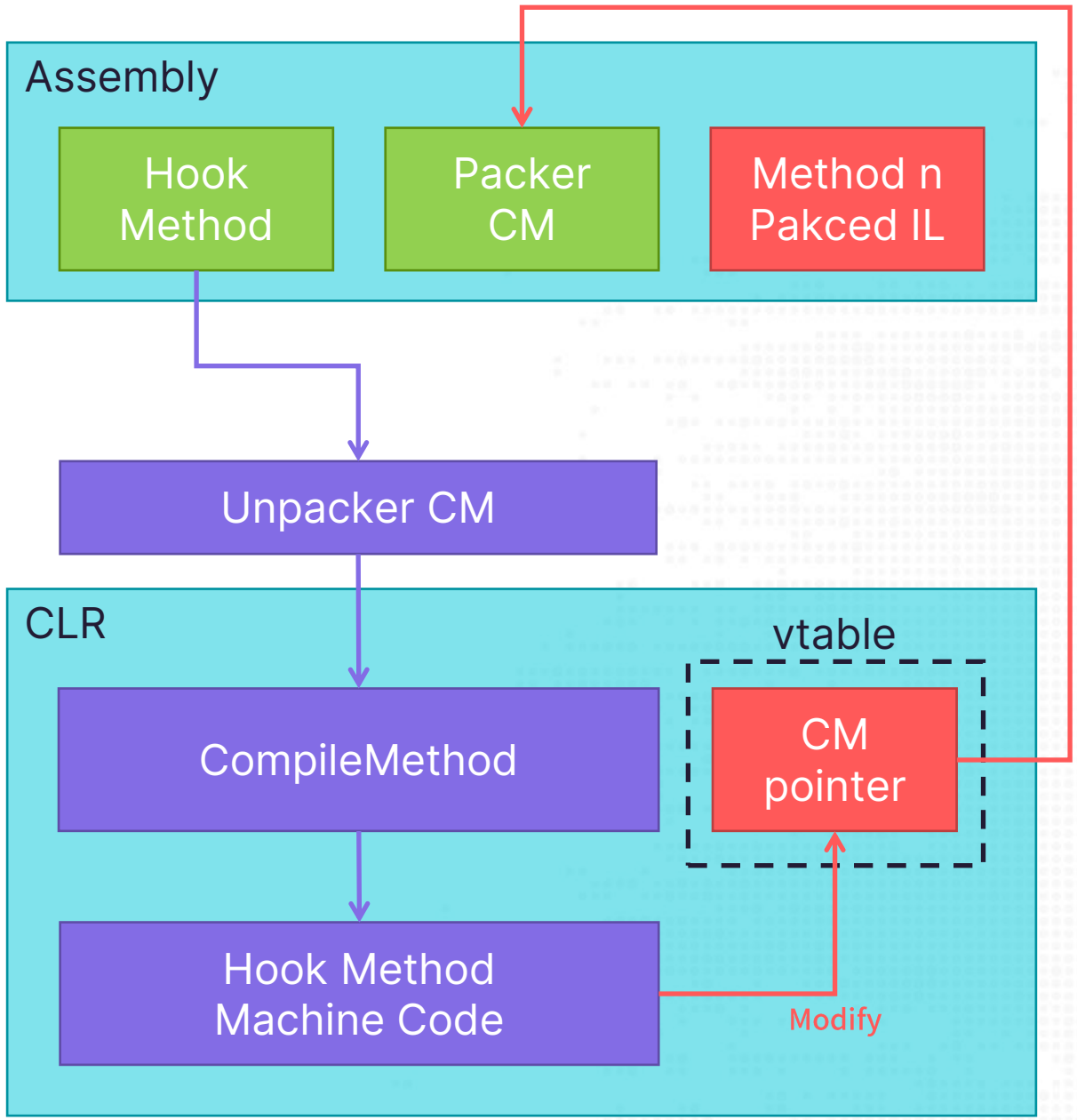


JITUnpacker



JITUnpacker

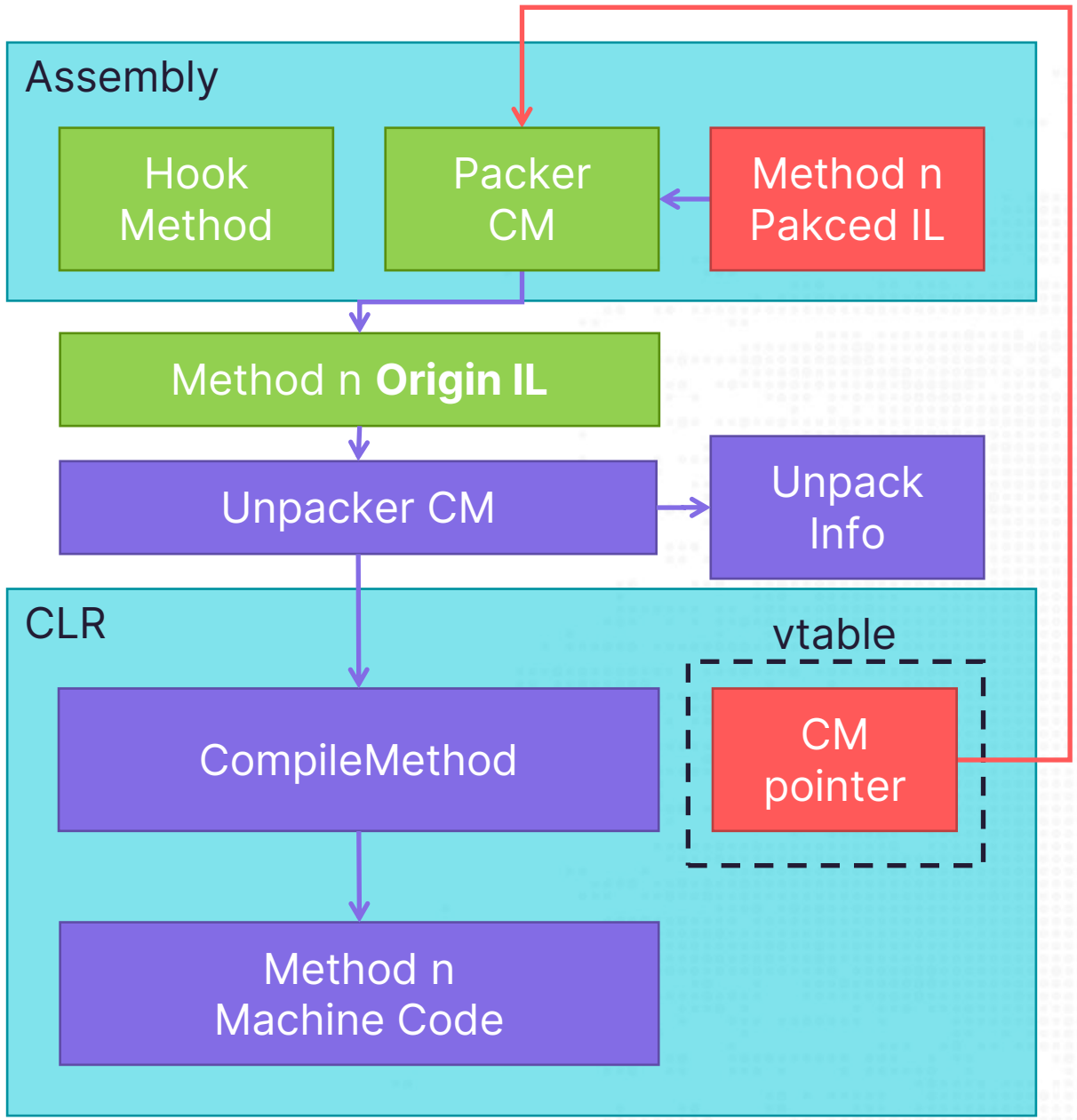
Assembly hooks CompileMethod to Packer CM



JITUnpacker

Intercept unpacked IL
Save it
Pass it to original CM

Use Unpack Info to rebuild unpacked assembly after the process is terminated



Unpack JITHook .NET assembly

- There are 2 type of method headers
- If
  - IL code size >= (1<<6)
  - Or the method has local variables
  - Or the method has EH (Exception Handler)
- Then the method header is fat format
- Otherwise, the header is tiny format

Unpack JITHook .NET assembly

- CorILMethod_TinyFormat
  - Only need the unpacked CIL code and size to rebuild
- CorILMethod_FatFormat
  - Not only need the unpacked CIL code and size to rebuild, but also
    - EH (Exception Handler) table
    - LocalVarSigTok

EH Table

- How to get EH table?
- There is a member function getEHinfo() in class ICorJitInfo

EH Table

```
// src/coreclr/inc/corjit.h
class ICorJitCompiler
{
    virtual CorJitResult compileMethod (
        ICorJitInfo                     *comp,
        struct CORINFO_METHOD_INFO  *info,
        unsigned /* code:CorJitFlag */   flags,
        uint8_t                        **nativeEntry,
        uint32_t                       *nativeSizeOfCode
        ) = 0;
    ...
}

class ICorJitInfo : public ICorDynamicInfo
{
    ...
}
```

```
// src/coreclr/inc/corinfo.h
class ICorDynamicInfo : public ICorStaticInfo
{
    ...
    // get individual exception handler
    virtual void getEHinfo(
                CORINFO_METHOD_HANDLE ftn,
                unsigned              EHnumber,
                CORINFO_EH_CLAUSE* clause
                ) = 0;
    ...
}
```

EH Table

- What is the index of getEHinfo() in the vtable?
- Let’s find a callsite of getEHinfo() and reverse it

```
void Compiler::fgFindBasicBlocks() {
    // Allocate the 'jump target' bit vector
    FixedBitVect *jumpTarget = FixedBitVect::bitVectInit(info.compILCodeSize + 1, this);

    // Walk the instrs to find all jump targets
    fgFindJumpTargets(info.compCode, info.compILCodeSize, jumpTarget);
    if (compDonotInline()) {
        return;
    }

    unsigned XTnum;

    /* Are there any exception handlers? */

    if (info.compXcptnsCount > 0) {
        noway_assert(!compIsForInlining());

        /* Check and mark all the exception handlers */

        for (XTnum = 0; XTnum < info.compXcptnsCount; XTnum++) {
            CORINFO_EH_CLAUSE clause;
            info.compCompHnd->getEHinfo(info.compMethodHnd, XTnum, &clause);
    ...
```

```
void Compiler::fgFindBasicBlocks() {
    // Allocate the 'jump target' bit vector
    FixedBitVect *jumpTarget = FixedBitVect::bitVectInit(info.compILCodeSize + 1, this);

    // Walk the instrs to find all jump targets
    fgFindJumpTargets(info.compCode, info.compILCodeSize, jumpTarget);
    if (compDonotInline()) {
        return;
    }

    unsigned XTnum;

    /* Are there any exception handlers? */

    if (
```

Info.compCompHnd is a ICorJitInfo pointer

```
        /* Check and mark all the exception handlers */

        for (XTnum = 0; XTnum < info.compXcptnsCount; XTnum++) {
            CORINFO_EH_CLAUSE clause;
            info.compCompHnd->getEHinfo(info.compMethodHnd, XTnum, &clause);
    ...
```

```
*       mov     rdi, [rsi+1AB8h]; this->info.compCompHnd (ICorJitInfo *)
*       mov     rax, [rdi]      ; vtable (clr!CEEJitInfo::'vftable')
*       mov     rbx, [rax+40h]  ; vtable[8] (clr!CEEJitInfo::getEHinfo)
*       mov     rcx, rbx
*       call    cs:__guard_check_icall_fptr ;
*       mov     rdx, [rsi+1AD0h]
*       lea     r9, [rbp+clause]
*       mov     r8d, r12d
*       mov     rcx, rdi
*       call    rbx             ; (clr!CEEJitInfo::getEHinfo)
```

Info.compCompHnd is a ICorJitInfo pointer

```
/* Check and mark all the exception handlers */

for (XTnum = 0; XTnum < info.compXcptnsCount; XTnum++) {
    CORINFO_EH_CLAUSE clause;
    info.compCompHnd->getEHinfo(info.compMethodHnd, XTnum, &clause);
...
```

EH Table

- We can get getEHinfo() address from ICorJitInfo pointer passed to compileMethod()
- Then use it to get EH table

```
CorJitResult compileMethodHook(
    void                                *thisptr,
    ICorJitInfo                         *comp,              /* IN */
    struct CORINFO_METHOD_INFO          *info,              /* IN */
    unsigned /* code:CorJitFlag */      flags,              /* IN */
    uint8_t                            **nativeEntry,       /* OUT */
    uint32_t                            *nativeSizeOfCode   /* OUT */
)
{
    ...
    vtable = *((void ***)comp);
    getEHinfo = (getEHinfoFunc *)vtable[8];
    ...
}
```

LocalVarSigTok

> info->locals.pSig points to the LocalVarSig in #Blob stream

```
CorJitResult compileMethodHook(
    void                            *thisptr,
    ICorJitInfo                     *comp,
    struct CORINFO_METHOD_INFO      *info,
    unsigned /* code:CorJitFlag */   flags,
    uint8_t                        **nativeEntry,
    uint32_t                        *nativeSizeOfCode
)
```

```
struct CORINFO_METHOD_INFO
{
    CORINFO_METHOD_HANDLE        ftn;
    CORINFO_MODULE_HANDLE        scope;
    uint8_t                     *ILCode;
    unsigned                     ILCodeSize;
    unsigned                     maxStack;
    unsigned                     EHcount;
    CorInfoOptions               options;
    CorInfoRegionKind            regionKind;
    CORINFO_SIG_INFO             args;
    CORINFO_SIG_INFO             locals;
};
```

LocalVarSigTok

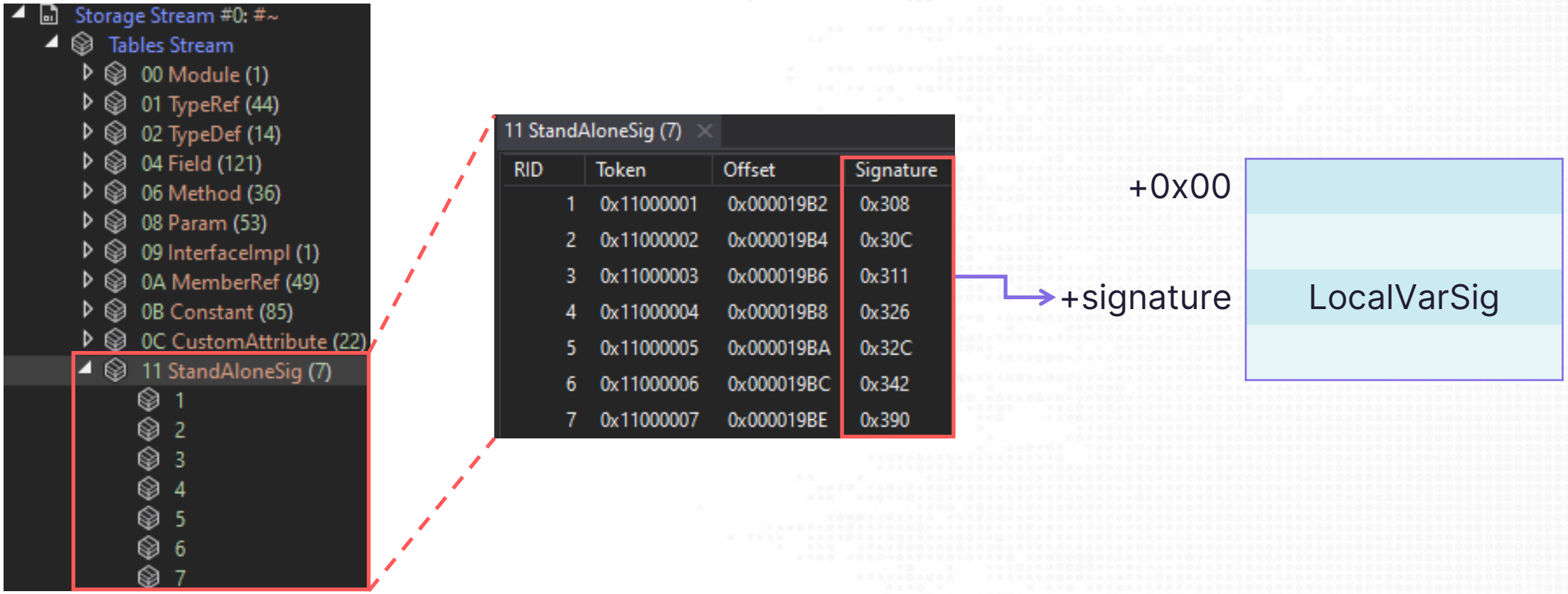
> info->locals.pSig points to the LocalVarSig in #Blob stream

```
struct CORINFO_METHOD_INFO
{
    CORINFO_METHOD_HANDLE        ftn;
    CORINFO_MODULE_HANDLE        scope;
    uint8_t                     *ILCode;
    unsigned                     ILCodeSize;
    unsigned                     maxStack;
    unsigned                     EHcount;
    CorInfoOptions               options;
    CorInfoRegionKind            regionKind;
    CORINFO_SIG_INFO             args;
    CORINFO_SIG_INFO             locals;
};
```

```
+ 0:007 > dt clrjit!CORINFO_SIG_INFO
+ 0x000 callConv         : CorInfoCallConv
+ 0x008 retTypeClass     : Ptr64 CORINFO_CLASS_STRUCT_
+ 0x010 retTypeSigClass  : Ptr64 CORINFO_CLASS_STRUCT_
+ 0x018 retType          : Pos 0, 8 Bits
+ 0x018 flags            : Pos 8, 8 Bits
+ 0x018 numArgs          : Pos 16, 16 Bits
+ 0x020 sigInst          : CORINFO_SIG_INST
+ 0x040 args             : Ptr64 CORINFO_ARG_LIST_STRUCT_
+ 0x048 pSig             : Ptr64 UChar
+ 0x050 cbSig            : Uint4B
+ 0x058 scope            : Ptr64 CORINFO_MODULE_STRUCT_
+ 0x060 token            : Uint4B
```

LocalVarSigTok

> LocalVarSigTok of StandAlongSig.row[n] is 0x11000000 | (n + 1)

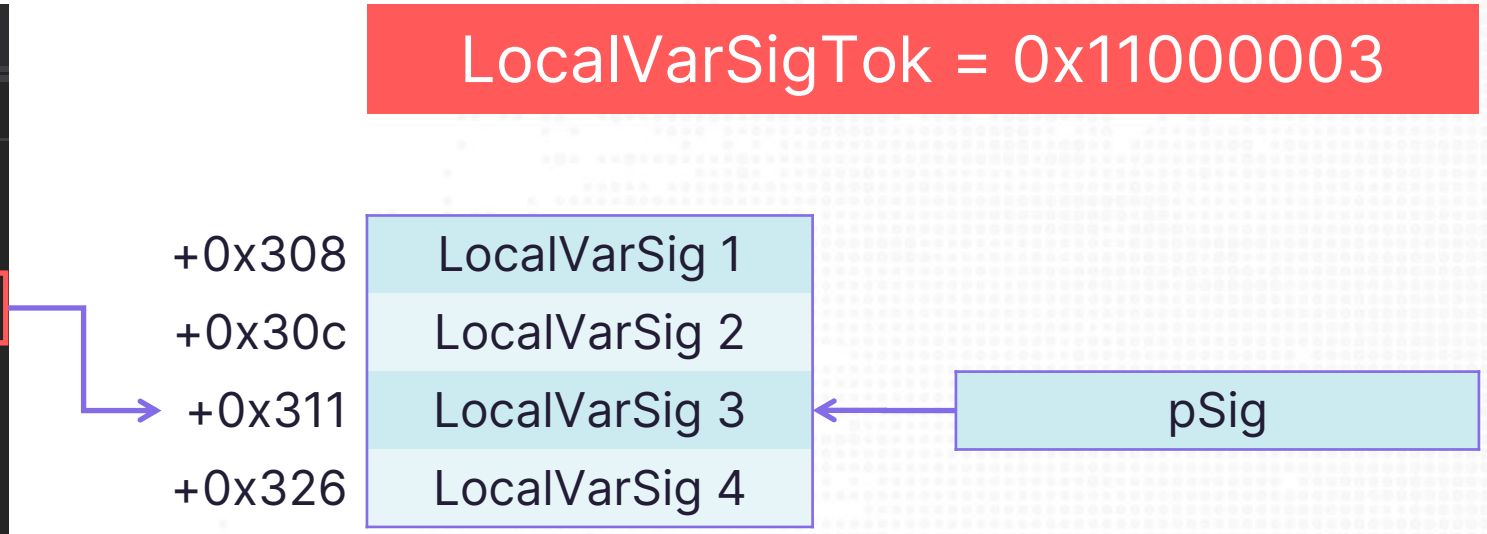


LocalVarSigTok

> Compare pSig and StandAlongSig.row[n] to find LocalVarSigTok

11 StandAloneSig (7)

RID	Token	Offset	Signature
1	0x11000001	0x000019B2	0x308
2	0x11000002	0x000019B4	0x30C
3	0x11000003	0x000019B6	0x311
4	0x11000004	0x000019B8	0x326
5	0x11000005	0x000019BA	0x32C
6	0x11000006	0x000019BC	0x342
7	0x11000007	0x000019BE	0x390



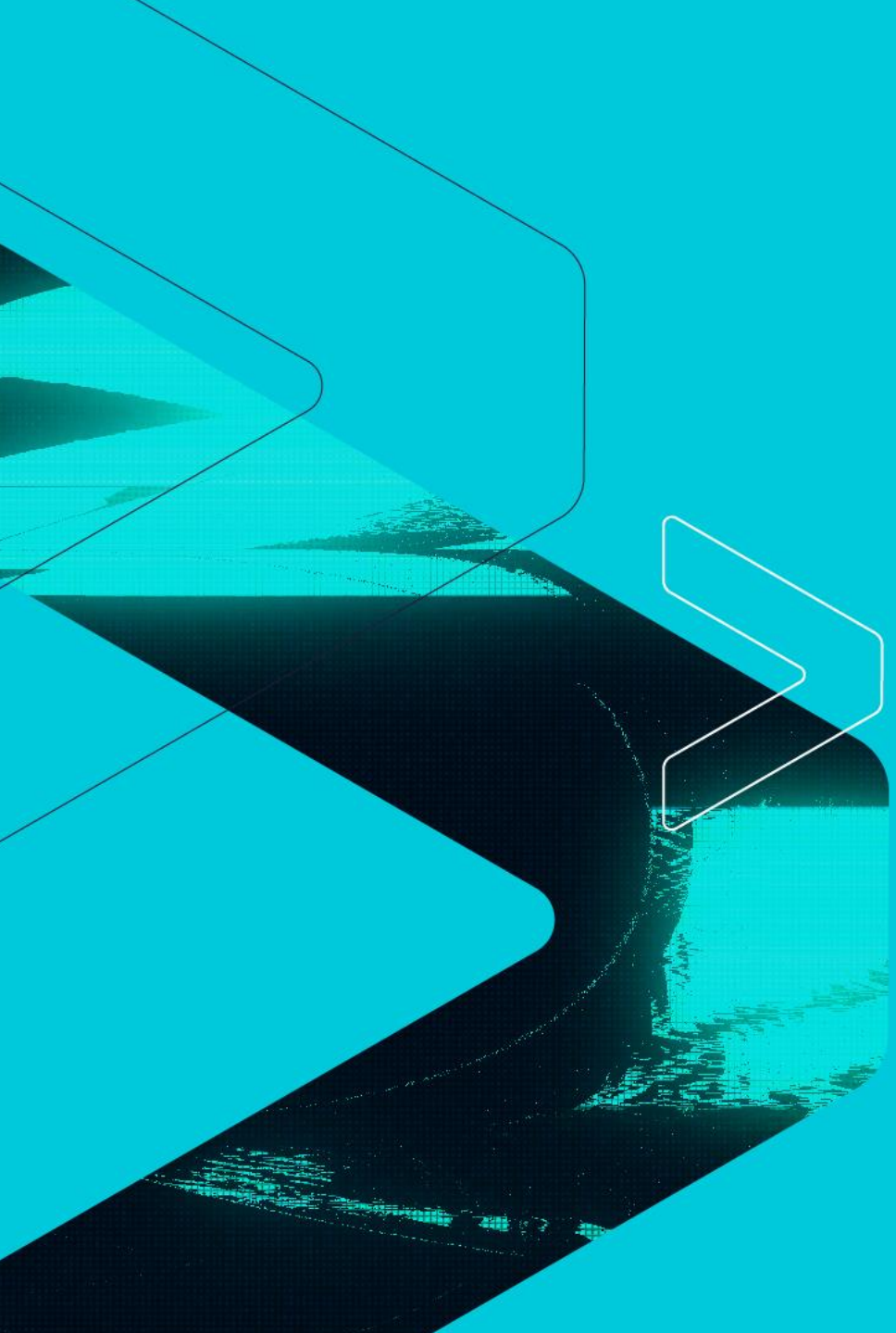
Rows of StandAlongSig

#Blob Stream

info->locals.pSig

Unpack JITHook .NET assembly

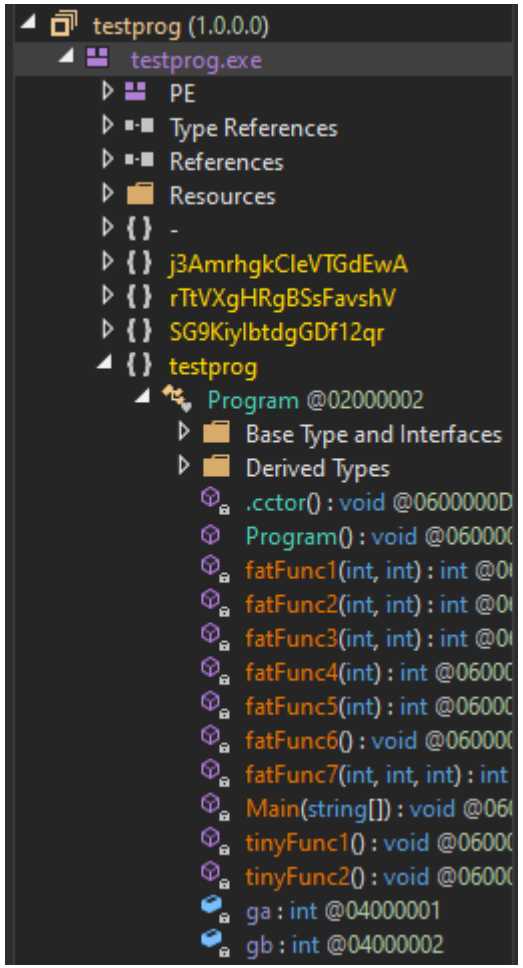
- CorILMethod_TinyFormat
  - Only need the unpacked CIL code and size to rebuild
- CorILMethod_FatFormat
  - Not only need the unpacked CIL code and size to rebuild, but also
    - EH (Exception Handler) table
    - LocalVarSigTok
- Now we have gathered all the information we need
- So we can rebuild the unpacked assembly!



Evaluation

Sample

- Sample 1:
  - Use .NET Reactor packer to pack test assembly
  - Only enable NecroBit option, focus on testing type 2 packer




```
namespace testprog
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000002 RID: 2 RVA: 0x00002050
        [MethodImpl(MethodImplOptions.NoInlining)]
        private static void tinyFunc1()
        {
        }

        // Token: 0x06000003 RID: 3 RVA: 0x00002058
        [MethodImpl(MethodImplOptions.NoInlining)]
        private static void tinyFunc2()
        {
        }

        // Token: 0x06000004 RID: 4 RVA: 0x00002060
        [MethodImpl(MethodImplOptions.NoInlining)]
        private static int fatFunc1(int a, int b)
        {
            return 0;
        }
```

```
.method private hidebysig static
    int32 fatFunc7 (
        int32 a,
        int32 b,
        int32 c
    ) cil managed noinlining
{
    // Header Size: 12 bytes
    // Code Size: 4 (0x4) bytes
    .maxstack 8

    /* 0x000002FC 00           */ IL_0000: nop
    /* 0x000002FD 00           */ IL_0001: nop
    /* 0x000002FE 16           */ IL_0002: ldc.i4.0
    /* 0x000002FF 2A           */ IL_0003: ret
} // end of method Program::fatFunc7
```

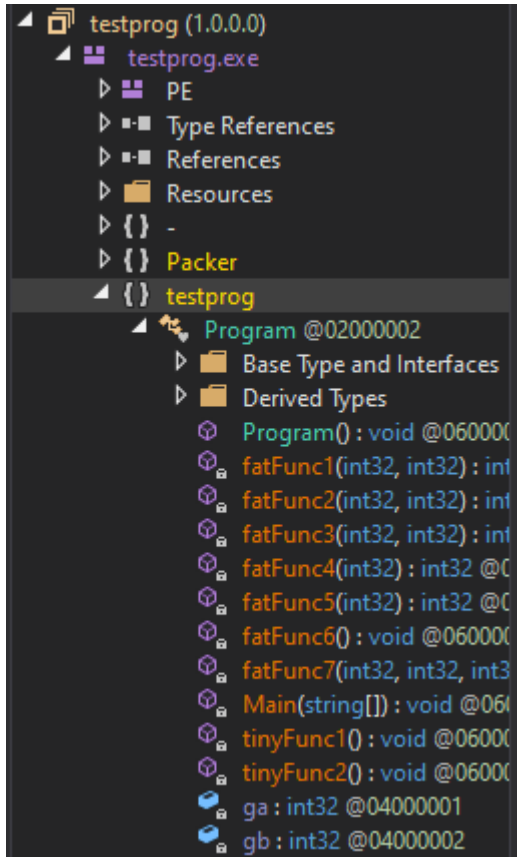
Assembly explorer

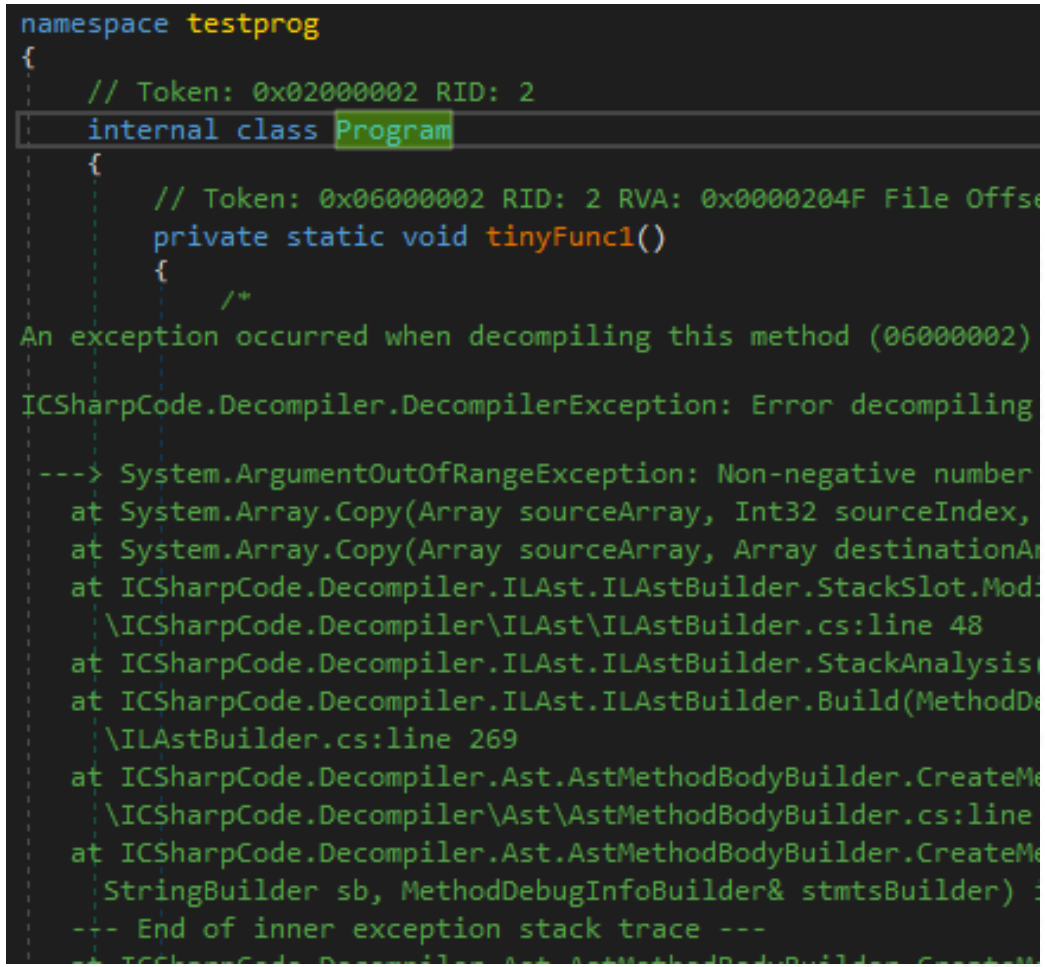
Class

CIL disassembly

Sample

- Sample 2:
  - Use JITPacker to pack test assembly





```
namespace testprog
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000002 RID: 2 RVA: 0x0000204F File Offse
        private static void tinyFunc1()
        {
            /*
An exception occurred when decompiling this method (06000002)

ICSharpCode.Decompiler.DecompilerException: Error decompiling

---> System.ArgumentOutOfRangeException: Non-negative number
  at System.Array.Copy(Array sourceArray, Int32 sourceIndex,
  at System.Array.Copy(Array sourceArray, Array destinationAr
  at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackSlot.Modi
    \ICSharpCode.Decompiler\ILAst\ILAstBuilder.cs:line 48
  at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackAnalysis(
  at ICSharpCode.Decompiler.ILAst.ILAstBuilder.Build(MethodDe
    \ILAstBuilder.cs:line 269
  at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMe
    \ICSharpCode.Decompiler\Ast\AstMethodBodyBuilder.cs:line
  at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMe
    StringBuilder sb, MethodDebugInfoBuilder& stmtsBuilder)
  --- End of inner exception stack trace ---
```

```
.method private hidebysig static
    int32 fatFunc7 (
        int32 a,
        int32 b,
        int32 c
    ) cil managed
{
    // Header Size: 12 bytes
    // Code Size: 59 (0x3B) bytes
    // LocalVarSig Token: 0x11000001 RID: 1
    .maxstack 2
    .locals init (
        [0] int32
    )

    /* 0x0000042C 87          */ IL_0000: conv.ovf.u2.un
    .try
    {
        /* 0x0000042D 00          */ IL_0001: nop
        .try
        {
            /* 0x0000042E 00          */ IL_0002: nop
            /* 0x0000042F 87          */ IL_0003: conv.ovf.u2.un
            /* 0x00000430 87          */ IL_0004: conv.ovf.u2.un
            /* 0x00000431 87          */ IL_0005: conv.ovf.u2.un
            /* 0x00000432 87          */ IL_0006: conv.ovf.u2.un
            /* 0x00000433 87          */ IL_0007: conv.ovf.u2.un
            /* 0x00000434 87          */ IL_0008: conv.ovf.u2.un
            /* 0x00000435 87          */ IL_0009: conv.ovf.u2.un
            /* 0x00000436 87          */ IL_000A: conv.ovf.u2.un
        } // end .try
```

Assembly explorer

Class

CIL disassembly

CYCRAFT

Unpacker

- Try unpacking the samples using the following unpackers
  - De4dot
  - .NETReactorSlayer
  - mandiant/jitm
  - JITUnpacker

Result

	Sample 1 (.NET Reactor)	Sample 2 (JITPacker)
De4dot	✕	✕
.NETReactorSlayer	○	✕
mandiant/JITM	△	△
JITUnpacker	○	○

mandiant/jitm

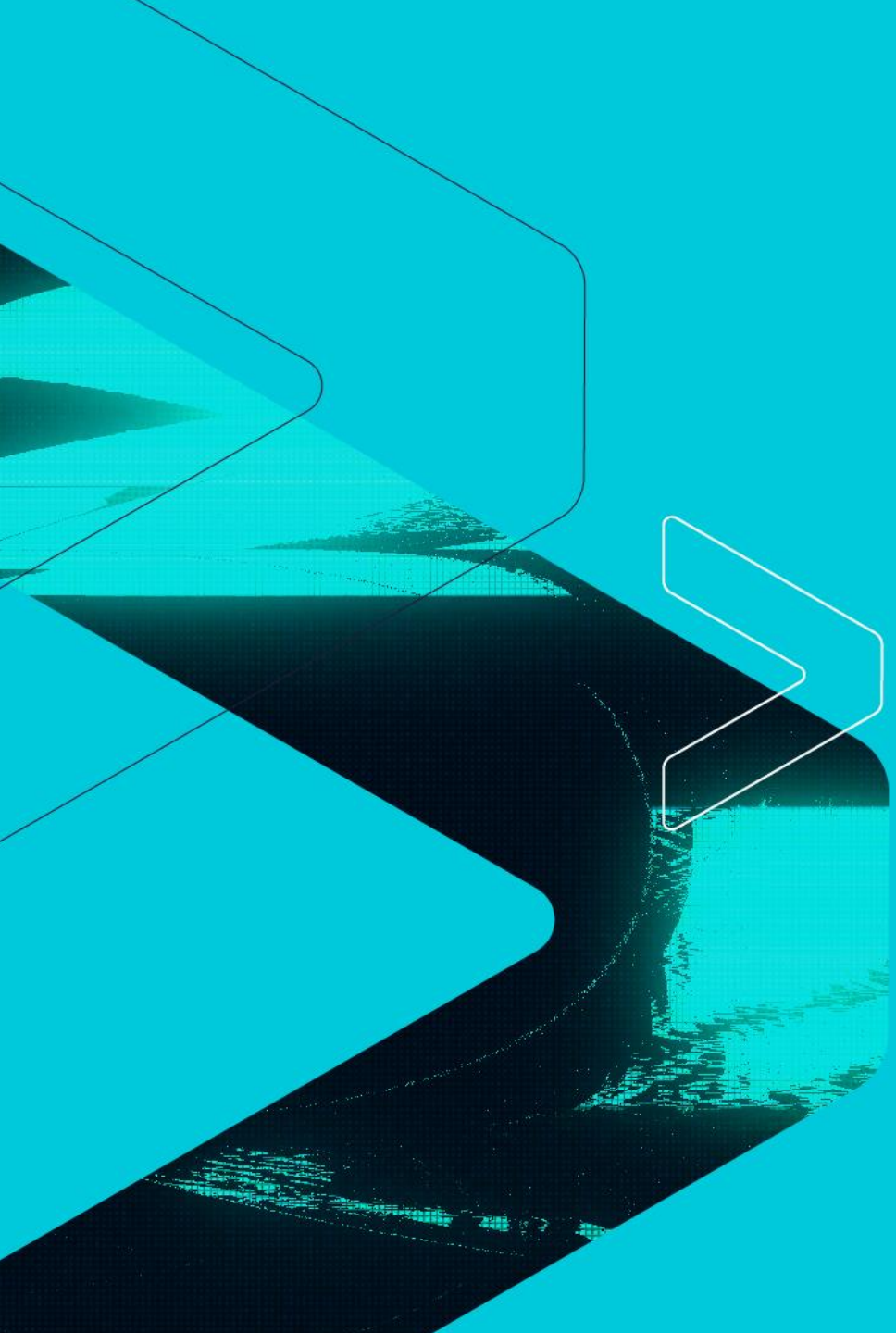
- A github repository, last updated 2 years ago (Dec, 2020)
- The mechanism of JITM is very similar with JITUnpacker
- Problems of JITM:
  - JITM doesn't handle EH table
  - JITM can't process assemblies that require user interaction
  - JITM can't process assemblies which's machine field of PE file header isn't 0x14c (Intel 386)
  - JITM only collects unpacked CIL which the name of the assembly which the method belongs to is equal to file name

Conclusion

- JITHook: Overwrite the vtable of JIT compiler.
- The current unpacker only supports unpacking assemblies packed with specific packers
- JITUnpacker does not target to certain packers
  but JITHook
- In theory, JITUnpacker can handle any assembly that packed by a packer that uses JITHook technique

Source code

- https://github.com/LJP-TW/JITHook



References

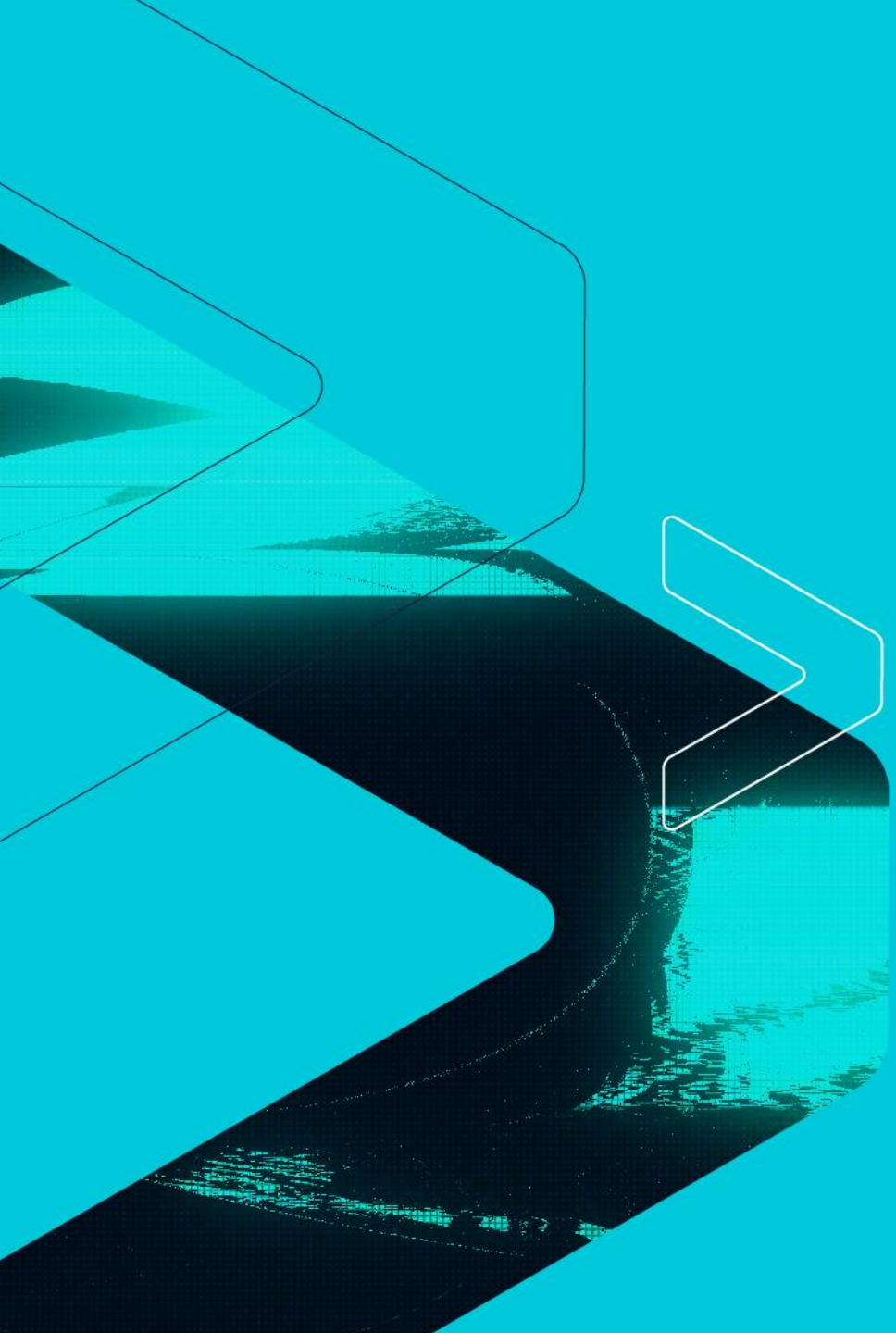
References

- Hijacking .NET to Defend PowerShell
- Unpack Your Troubles: .NET Packer Tricks And Countermeasures
- Jerry Wang - .NET CLR Injection: Modify IL Code during Run-time
- 浅谈.Net脱壳中方法体的局部变量签名还原
- .NET JIT脱壳指南与工具源码

Thank You!

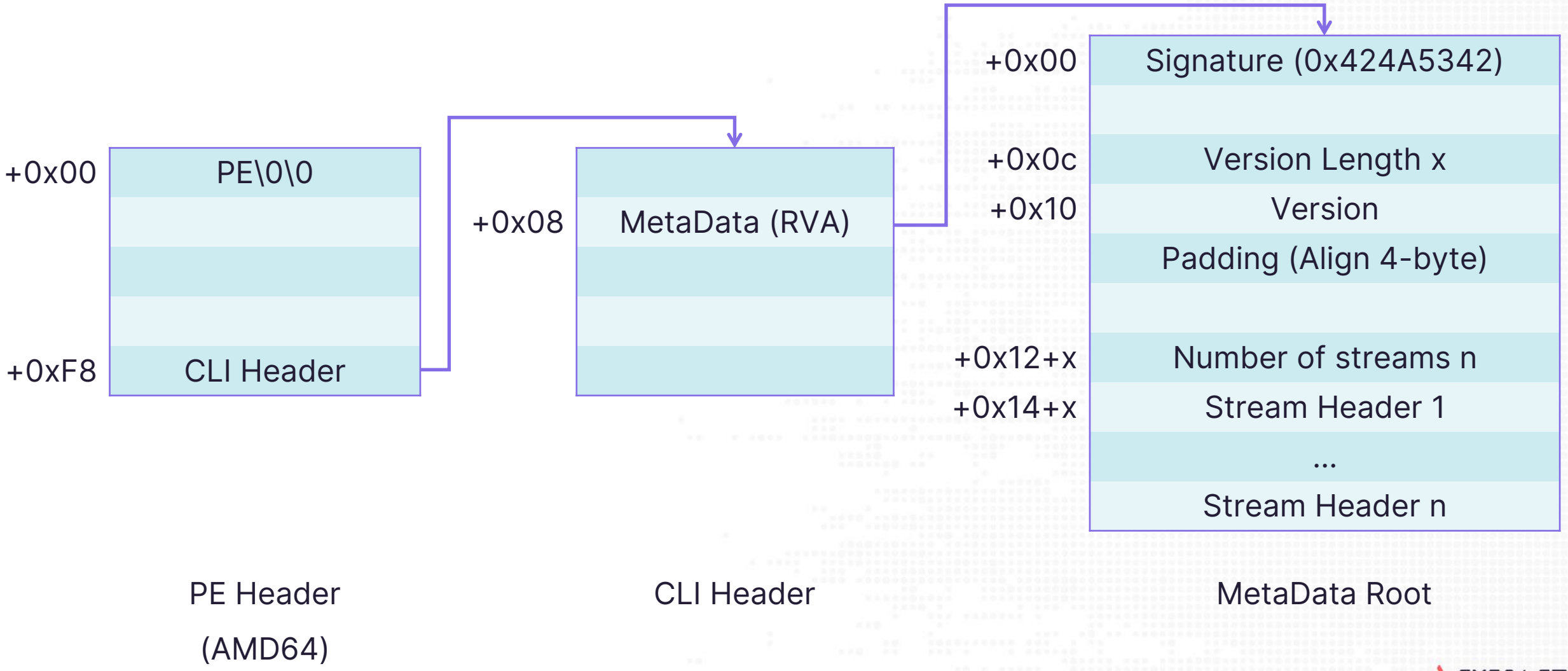
Q & A

EVERYTHING
STARTS
FROM
SECURITY

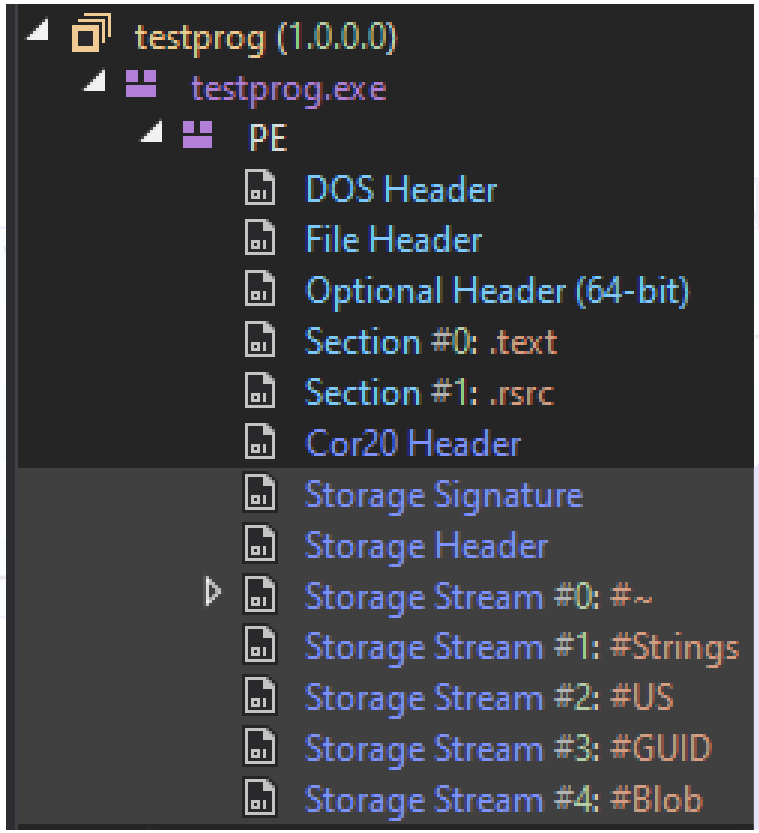


Method in PE

MetaData



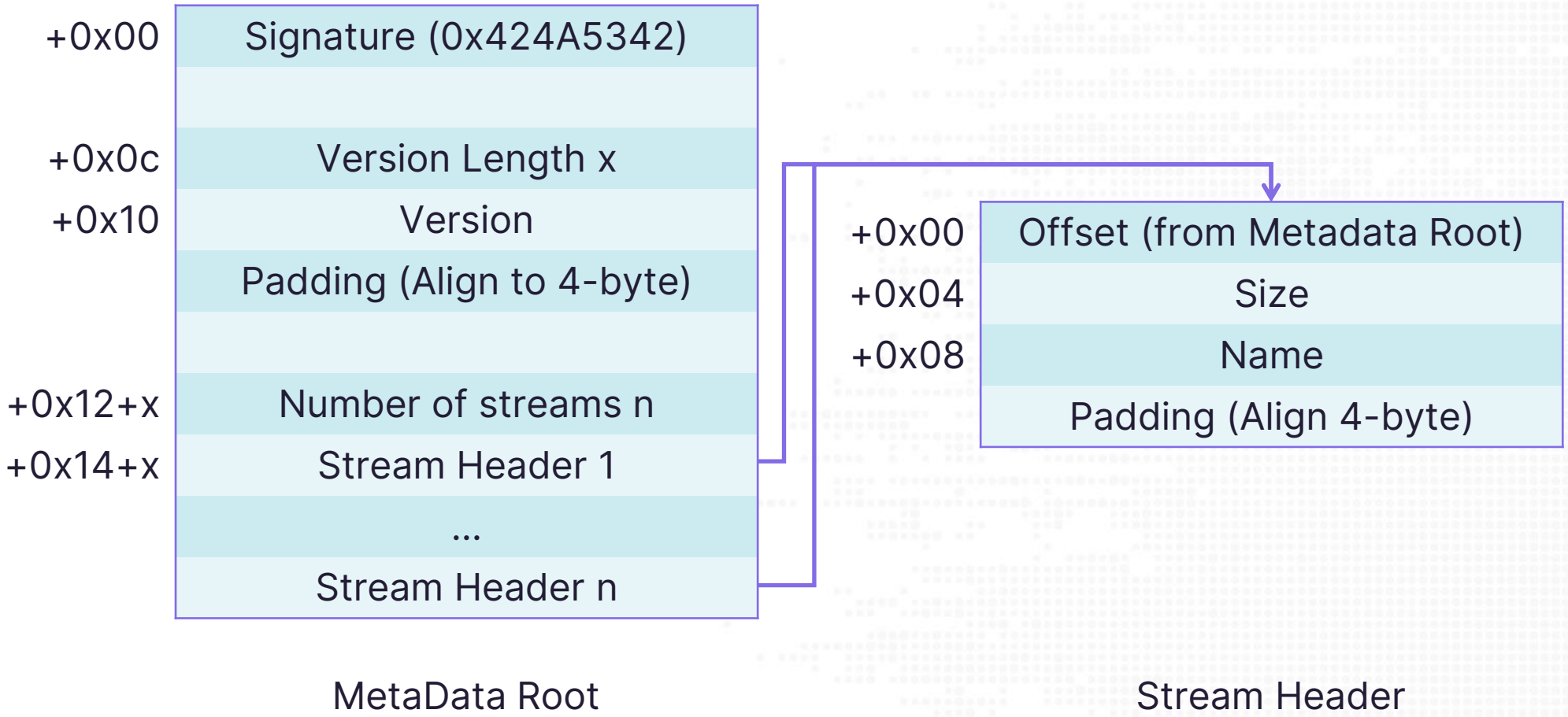
MetaData



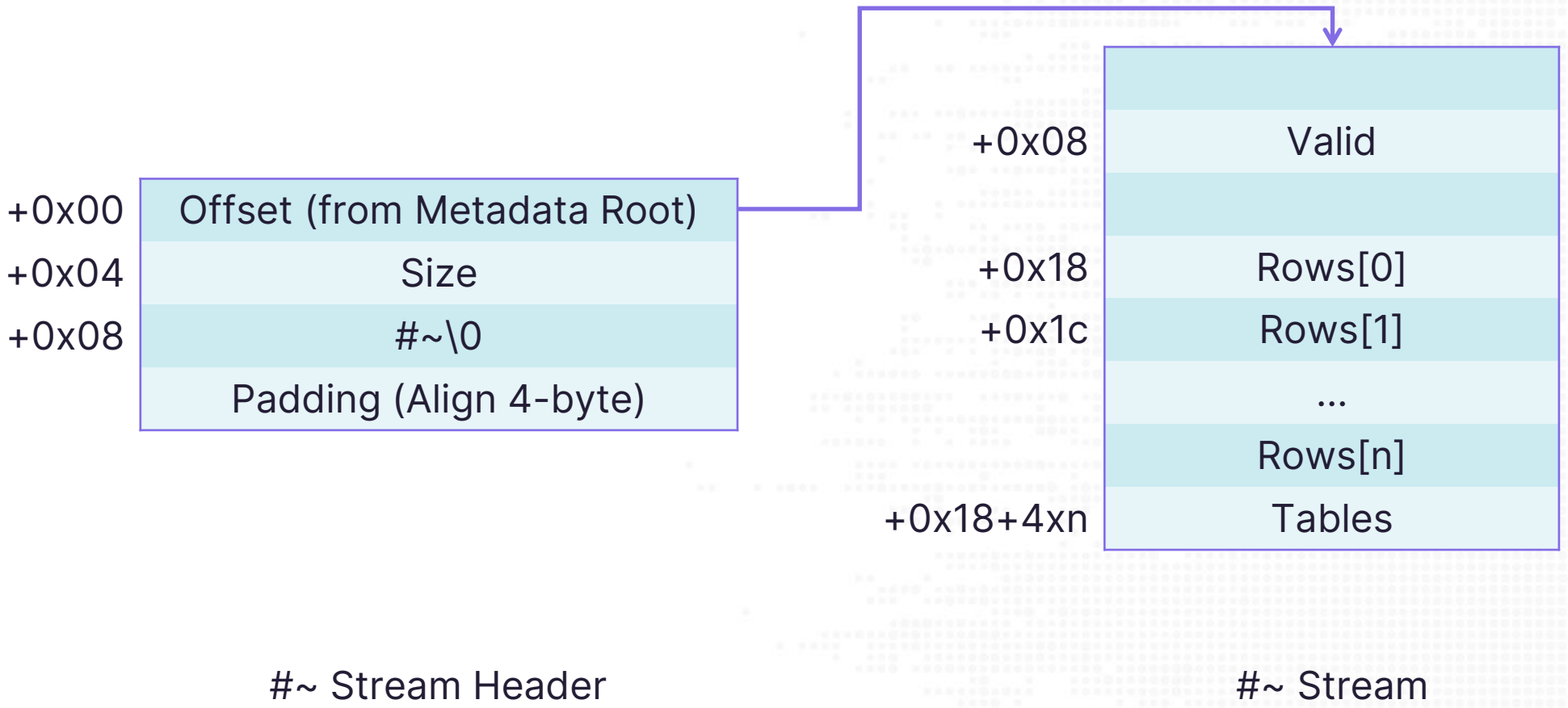
+0x00	Signature (0x424A5342)
+0x0c	Version Length x
+0x10	Version
	Padding (Align 4-byte)
+0x12+x	Number of streams n
+0x14+x	Stream Header 1
	...
	Stream Header n

PE Header dnSpy CLI Header MetaData Root

Stream Header

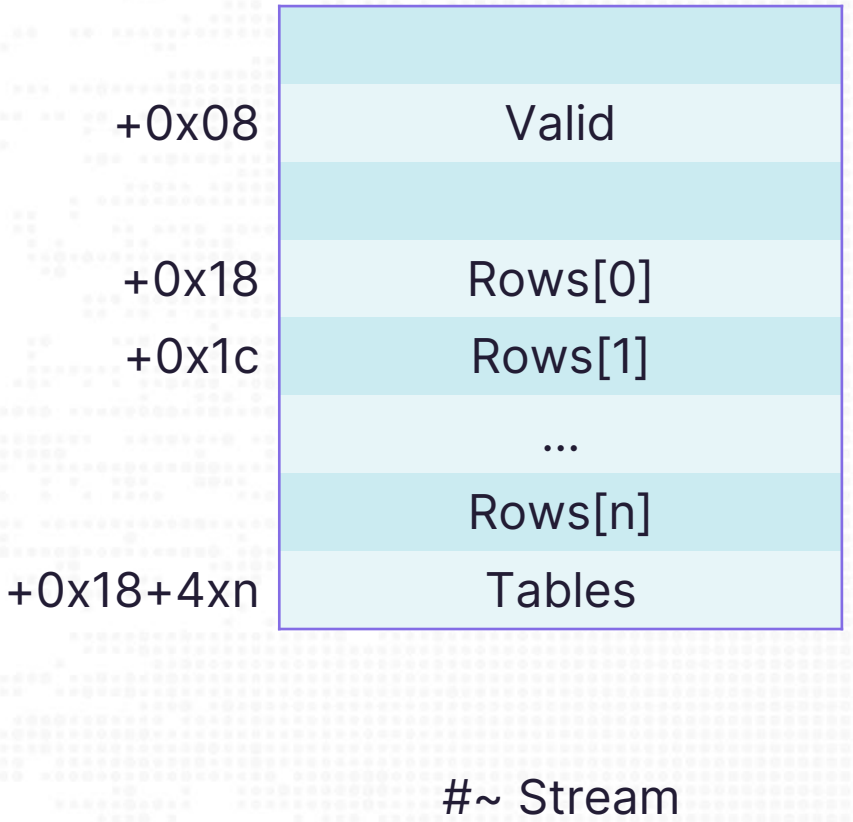


#~ Stream



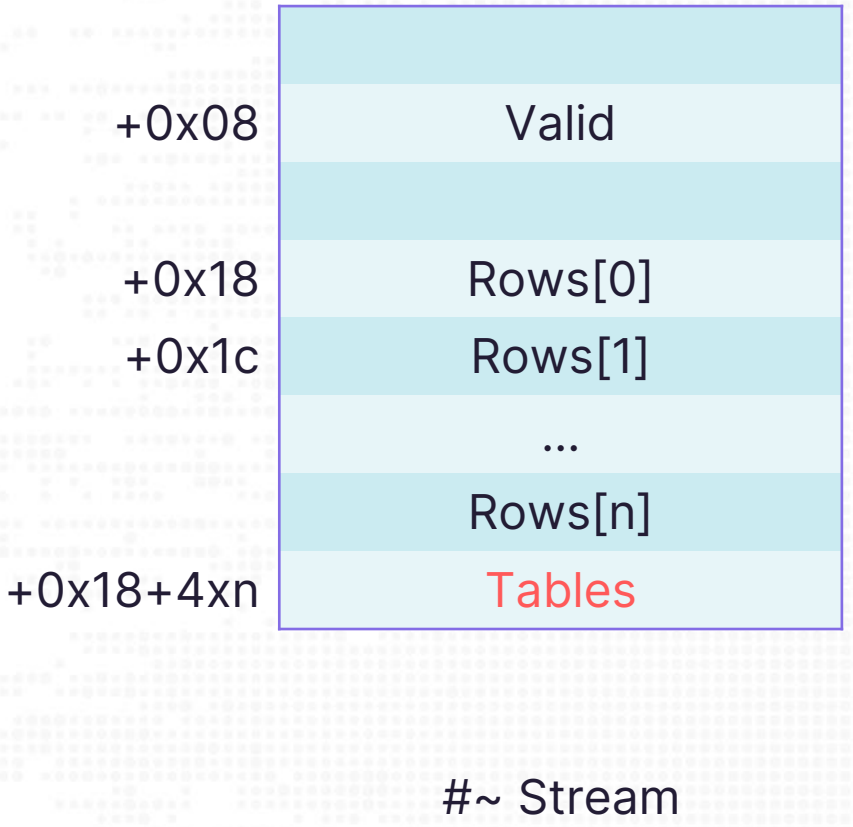
Metadata Tables

- There are various metadata tables
  - MethodDef
  - Param
  - ManifestResource
  - ModuleRef
  - ImplMap
  - ...



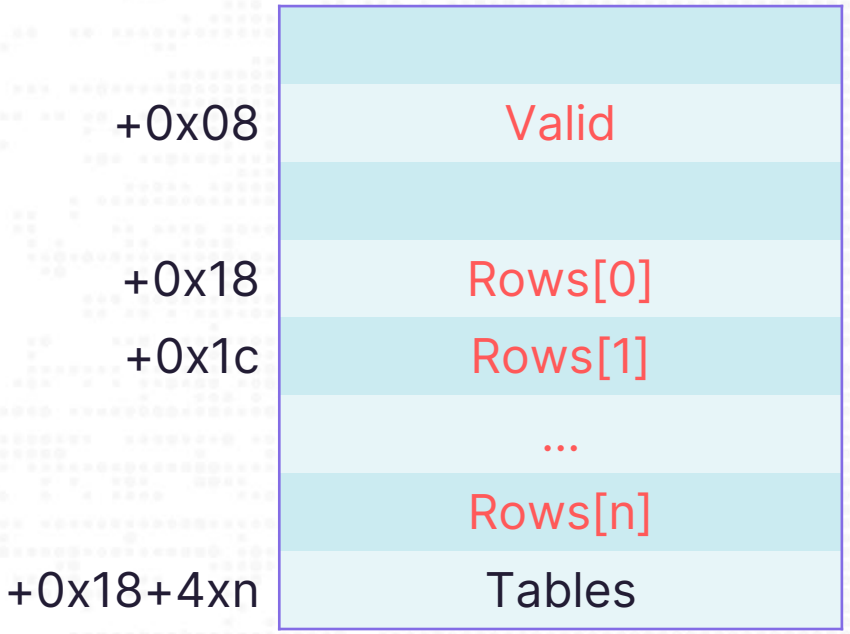
Metadata Tables

- Metadata tables are stored in the column Tables in #~ stream
- Tables are in a certain order
  - 0x00 Module
  - 0x01 TypeRef
  - ...
  - 0x06 MethodDef



Metadata Tables

- If n-th bit of Valid is 0
  - The n-th table has no rows.
- Otherwise
  - The n-th table has Rows[i] rows
  - i++



#~ Stream

Example

0^{th}-bit of Valid: 1

Module table has Rows[0] rows

+0x08	Valid	0x....57
+0x18	Rows[0]	0x01
+0x1c	Rows[1]	0x2c
+0x20	Rows[2]	0x0e
+0x24	Rows[3]	0x79
	...	
Module	Module.row[0]	

Example

1st-bit of Valid: 1

TypeRef table has Rows[1] rows

+0x08	Valid	0x....57
+0x18	Rows[0]	0x01
+0x1c	Rows[1]	0x2c
+0x20	Rows[2]	0x0e
+0x24	Rows[3]	0x79
	...	
Module	Module.row[0]	
TypeRef	TypeRef.row[0]	
	...	
	TypeRef.row[0x2b]	

Example

2nd-bit of Valid: 1

TypeDef table has Rows[2] rows

+0x08	Valid	0x....57
+0x18	Rows[0]	0x01
+0x1c	Rows[1]	0x2c
+0x20	Rows[2]	0x0e
+0x24	Rows[3]	0x79
	...	
TypeRef	TypeRef.row[0x2b]	
TypeDef	TypeDef.row[0]	
	...	
	TypeDef.row[0xd]	

Example

3rd-bit of Valid: 0

FieldPtr table has no rows

+0x08	Valid	0x....57
+0x18	Rows[0]	0x01
+0x1c	Rows[1]	0x2c
+0x20	Rows[2]	0x0e
+0x24	Rows[3]	0x79
	...	
TypeDef	TypeDef.row[0xd]	

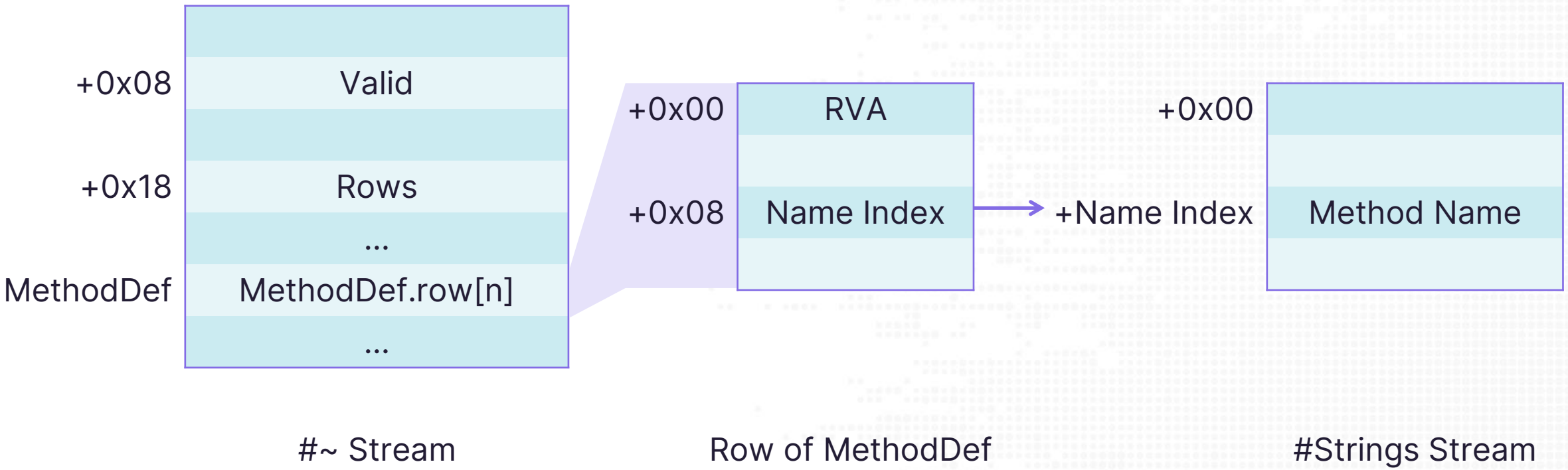
Example

4th-bit of Valid: 1

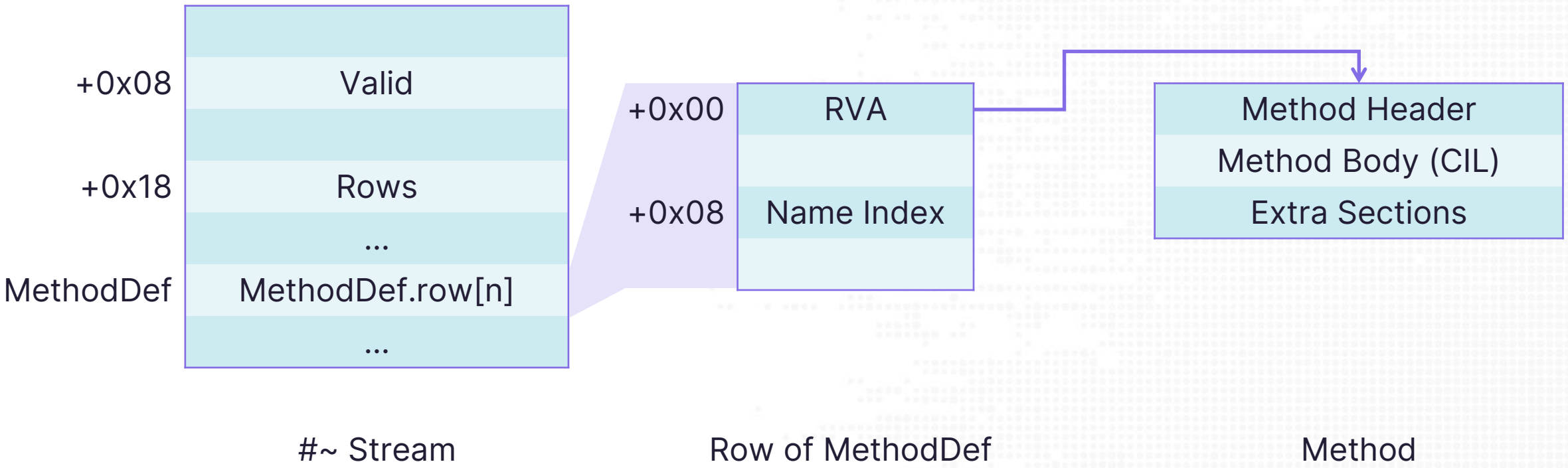
Field table
has Rows[3] rows

+0x08	Valid	0x....57
+0x18	Rows[0]	0x01
+0x1c	Rows[1]	0x2c
+0x20	Rows[2]	0x0e
+0x24	Rows[3]	0x79
	...	
TypeDef	TypeDef.row[0xd]	
Field	Field.row[0]	
	...	
	Field.row[0x78]	

MethodDef

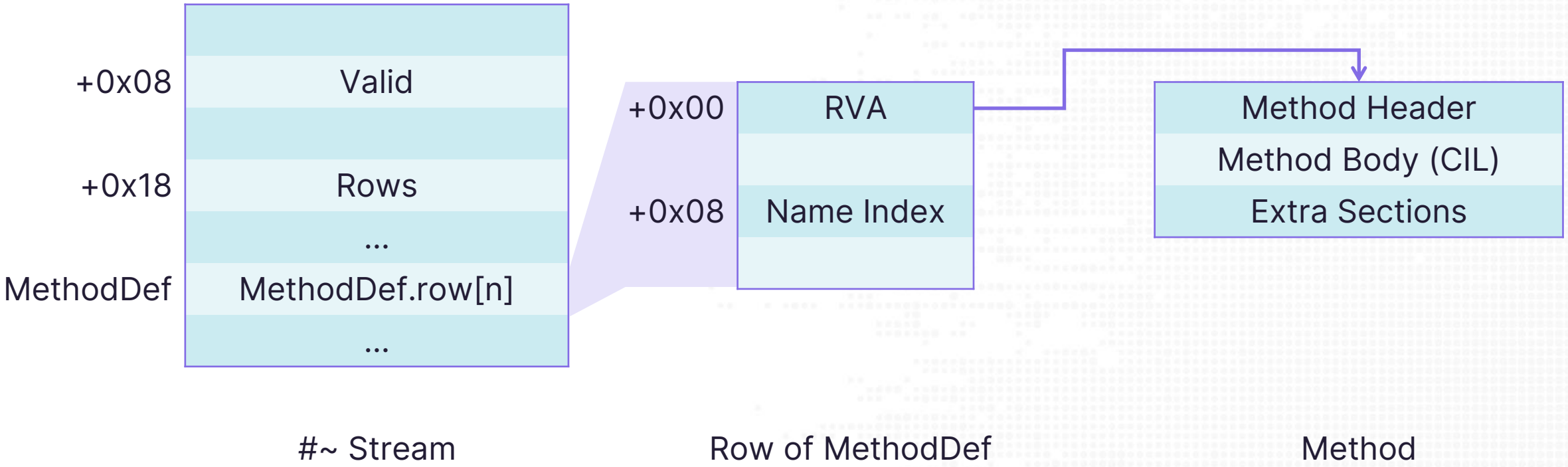


Method



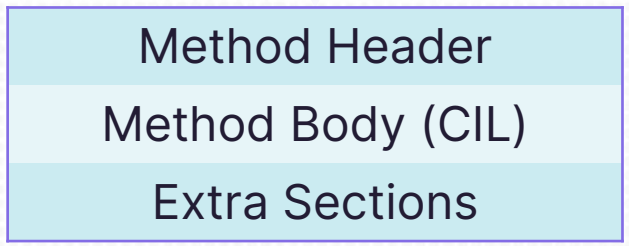
Method

- MethodToken of MethodDef.row[n] is
  - 0x06000000 | (n + 1)



Method

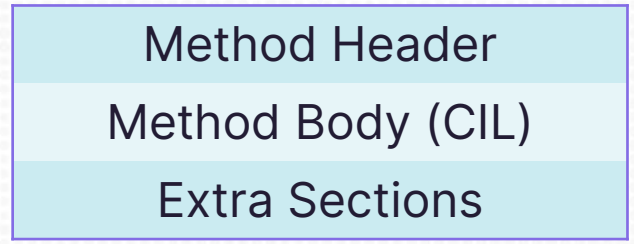
- There are 2 type of method headers
  - CorILMethod_TinyFormat
  - CorILMethod_FatFormat



Method

Method

- If
  - IL code size >= (1<<6)
  - Method has local variables
  - Method has EH (Exception Handler)
- Then the method header is fat format
- Otherwise, the header is tiny format



Method

CorILMethod_TinyFormat

```
4A 00 7E 01 00 00 04 7E 02 00 00 04 58 80 01 00
00 04 2A
```

CorILMethod_TinyFormat

```
4A 00 7E 01 00 00 04 7E 02 00 00 04 58 80 01 00
00 04 2A
```

- Bit[0:2] (0x02)
  - Flags (Must be 0x2)
- Bit[2:8] (0x48 >> 2 = 0x12)
  - Size of method body (CIL)

CorILMethod_TinyFormat

```
4A 00 7E 01 00 00 04 7E 02 00 00 04 58 80 01 00
00 04 2A
```

- Method body (CIL)

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method Header

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method Header
  - Flags

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method Header
  - Flags
  - MaxStack

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method Header
  - Flags
  - MaxStack
  - CodeSize

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method Header
  - Flags
  - MaxStack
  - CodeSize
  - LocalVarSigTok

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method body (CIL)

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Method body (CIL)
  - Pad with 0x00 to align to 4-byte boundary

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Extra method data section (Optional)
- Currently, this section is only used for EH (Exception Handler) table

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Extra method data section (Optional)
  - Flag (0x01 means this section is EH Table)

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Extra method data section (Optional)
  - Flag
  - DataSize (size of this section)

CorILMethod_FatFormat

```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Extra method data section (Optional)
  - Flag
  - DataSize
  - Reserved

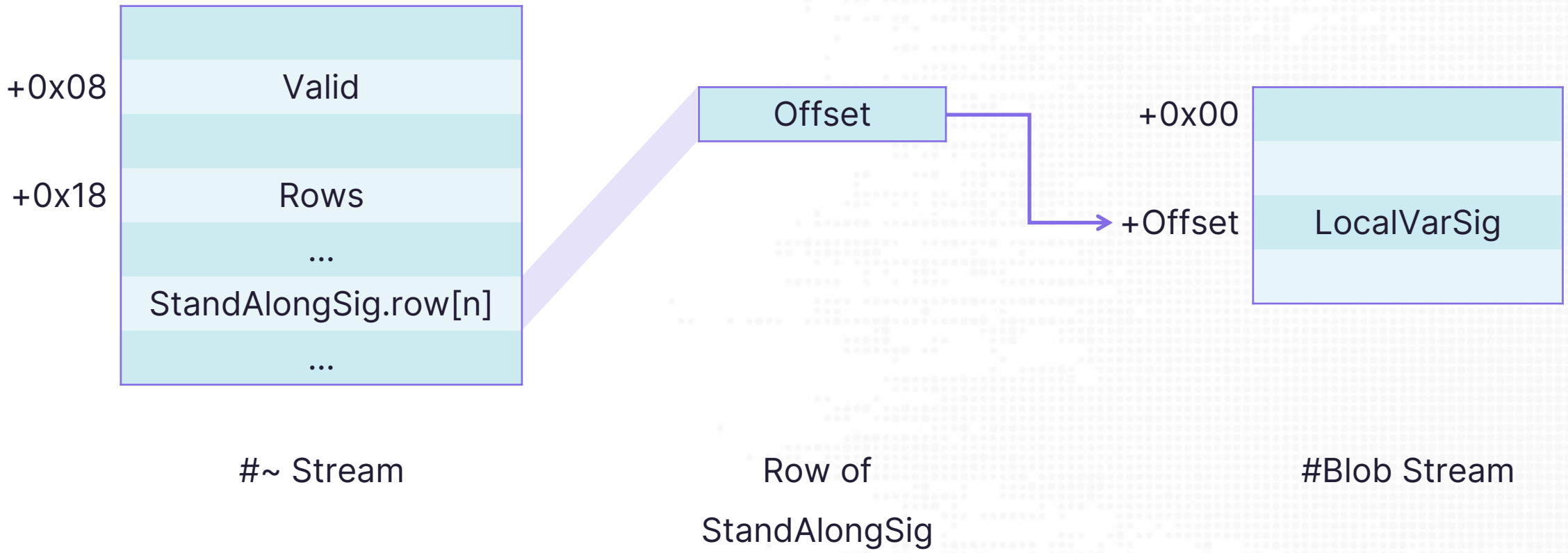
CorILMethod_FatFormat

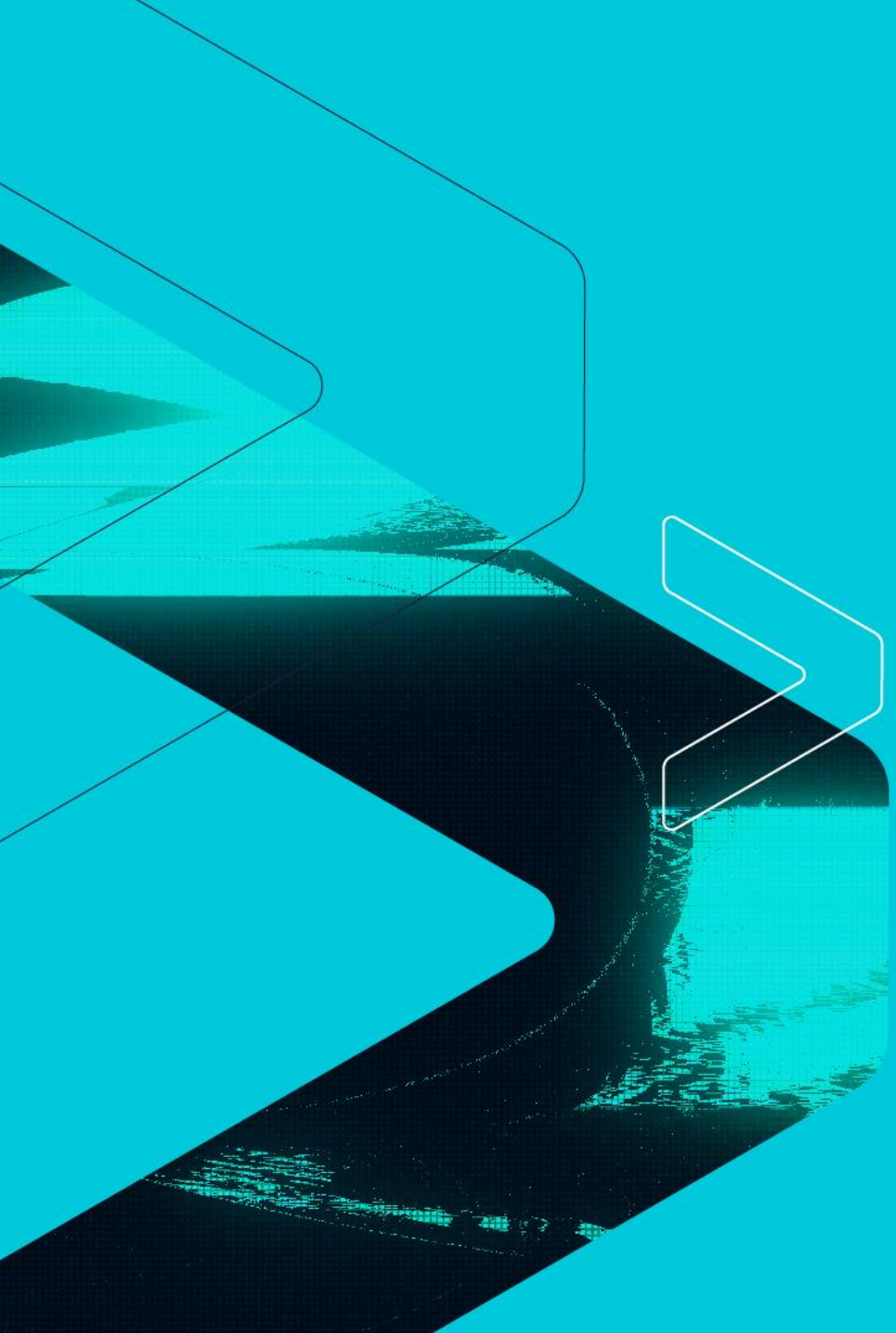
```
1B 30 01 00 12 00 00 00 01 00 00 11 00 00 28 03
00 00 06 00 02 0A DE 04 26 00 FE 1A 06 2A 00 00
01 10 00 00 00 00 01 00 0B 0C 00 04 01 00 00 01
```

- Extra method data section (Optional)
  - Flag
  - DataSize
  - Reserved
  - EH Clauses (Records the offset and length of try blocks and handlers)

StandAlongSig

> LocalVarSigToken of StandAlongSig.row[n] is 0x11000000 | (n + 1)

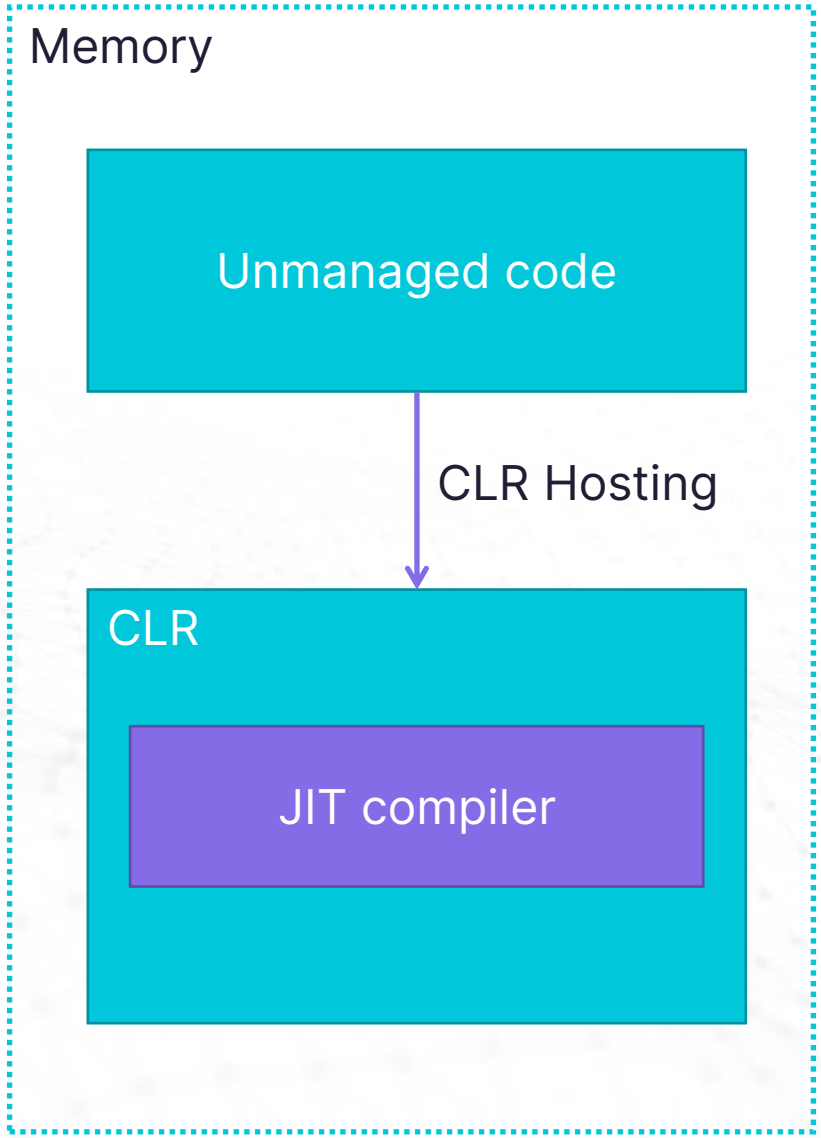




CLR Hosting

CLR Hosting

- Use unmanaged code to host CLR
- Let native process has the ability to load assemblies and run managed code



```
int clrHost(ICorRuntimeHost **pRuntimeHost)
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    BOOL bLoadable;

    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost,
                           (LPVOID *)&pMetaHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] CLRCreateInstance(...) succeeded\n");

    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (VOID **)&pRuntimeInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMetaHost->GetRuntime(...) succeeded\n");

    hr = pRuntimeInfo->IsLoadable(&bLoadable);

    if (FAILED(hr) || !bLoadable) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->IsLoadable(...) succeeded\n");

    hr = pRuntimeInfo->GetInterface(CLSID_CorRuntimeHost, IID_ICorRuntimeHost, (VOID **)pRuntimeHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->GetInterface(...) succeeded\n");

    hr = (*pRuntimeHost)->Start();

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->Start() succeeded\n");
}
```

Get **ICLRMetaHost** interface

```
int clrHost(ICorRuntimeHost **pRuntimeHost)
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    BOOL bLoadable;

    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost,
                           (LPVOID *)&pMetaHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] CLRCreateInstance(...) succeeded\n");

    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (VOID **)&pRuntimeInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMetaHost->GetRuntime(...) succeeded\n");

    hr = pRuntimeInfo->IsLoadable(&bLoadable);

    if (FAILED(hr) || !bLoadable) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->IsLoadable(...) succeeded\n");

    hr = pRuntimeInfo->GetInterface(CLSID_CorRuntimeHost, IID_ICorRuntimeHost, (VOID **)pRuntimeHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->GetInterface(...) succeeded\n");

    hr = (*pRuntimeHost)->Start();

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->Start() succeeded\n");
}
```

Get **ICLRRuntimeInfo** interface

```
int clrHost(ICorRuntimeHost **pRuntimeHost)
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    BOOL bLoadable;

    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost,
                           (LPVOID *)&pMetaHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] CLRCreateInstance(...) succeeded\n");

    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (VOID **)&pRuntimeInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMetaHost->GetRuntime(...) succeeded\n");

    hr = pRuntimeInfo->IsLoadable(&bLoadable);

    if (FAILED(hr) || !bLoadable) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->IsLoadable(...) succeeded\n");

    hr = pRuntimeInfo->GetInterface(CLSID_CorRuntimeHost, IID_ICorRuntimeHost, (VOID **)pRuntimeHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->GetInterface(...) succeeded\n");

    hr = (*pRuntimeHost)->Start();

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->Start() succeeded\n");
}
```

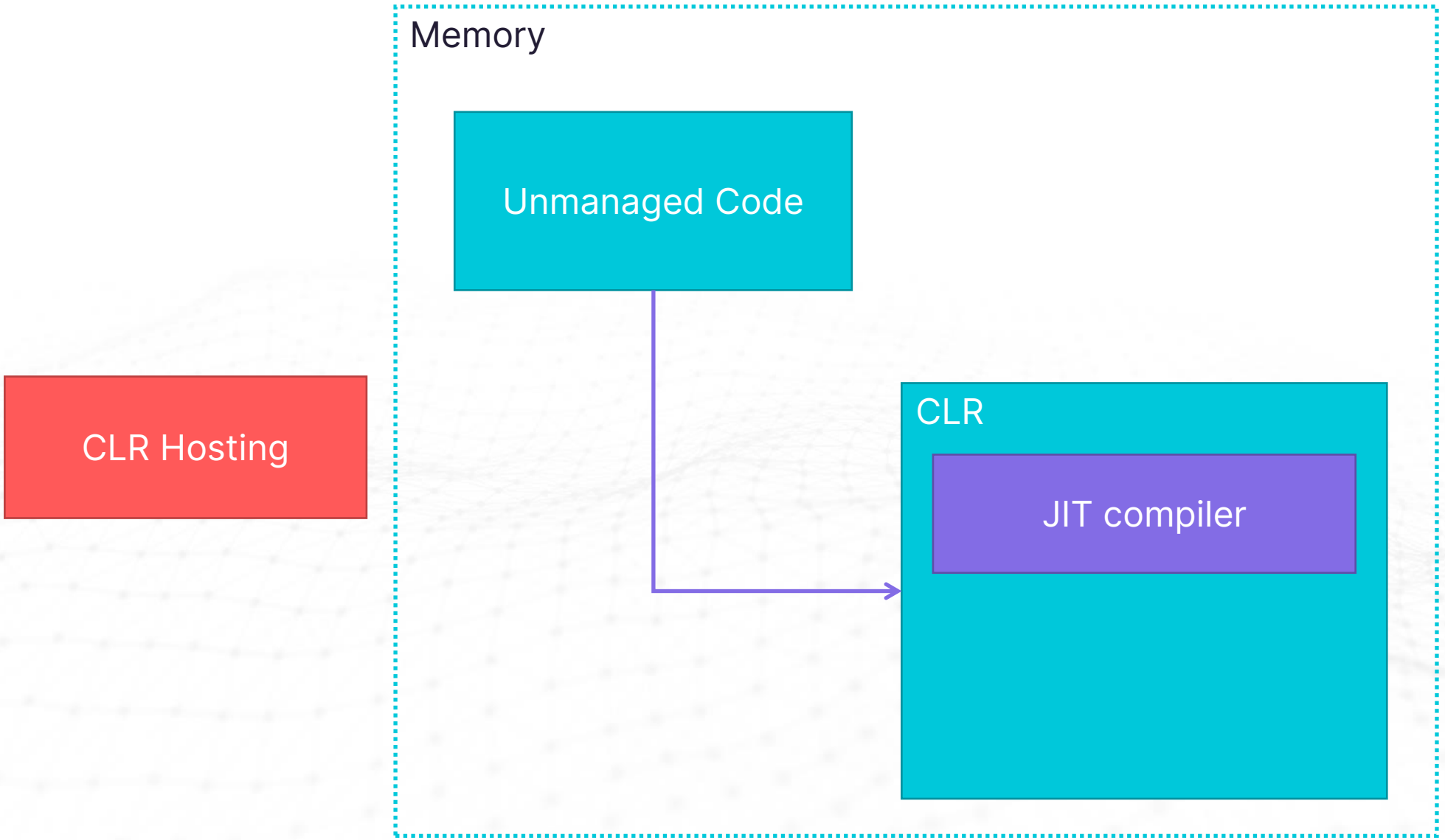
Check whether the CLR is loadable

```
int clrHost(ICorRuntimeHost **pRuntimeHost)
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    BOOL bLoadable;

    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost,
                           (LPVOID *)&pMetaHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] CLRCreateInstance(...) succeeded\n");

    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (VOID **)&pRuntimeInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMetaHost->GetRuntime(...) succeeded\n");

    hr = pRuntimeInfo->IsLoadable(&bLoadable);

    if (FAILED(hr) || !bLoadable) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->IsLoadable(...) succeeded\n");

    hr = pRuntimeInfo->GetInterface(CLSID_CorRuntimeHost, IID_ICorRuntimeHost, (VOID **)pRuntimeHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->GetInterface(...) succeeded\n");

    hr = (*pRuntimeHost)->Start();

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->Start() succeeded\n");
}
```

Load the CLR and Return **ICorRuntimeHost** interface pointer

```
int clrHost(ICorRuntimeHost **pRuntimeHost)
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    BOOL bLoadable;

    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost,
                           (LPVOID *)&pMetaHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] CLRCreateInstance(...) succeeded\n");

    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (VOID **)&pRuntimeInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMetaHost->GetRuntime(...) succeeded\n");

    hr = pRuntimeInfo->IsLoadable(&bLoadable);

    if (FAILED(hr) || !bLoadable) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->IsLoadable(...) succeeded\n");

    hr = pRuntimeInfo->GetInterface(CLSID_CorRuntimeHost, IID_ICorRuntimeHost, (VOID **)pRuntimeHost);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeInfo->GetInterface(...) succeeded\n");

    hr = (*pRuntimeHost)->Start();

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->Start() succeeded\n");
}
```

Start the CLR



```
int assemblyLoad(ICorRuntimeHost *pRuntimeHost,
                 mscorlib::_AssemblyPtr *pAssembly,
                 char *fileData,
                 int fileLength)
{
    HRESULT hr;
    IUnknownPtr pAppDomainThunk = NULL;
    mscorlib::_AppDomainPtr pDefaultAppDomain = NULL;
    SAFEARRAYBOUND rgsabound[1];
    SAFEARRAY *pSafeArray = NULL;
    void *pvData = NULL;

    hr = pRuntimeHost->GetDefaultDomain(&pAppDomainThunk);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pRuntimeHost->GetDefaultDomain(...) succeeded\n");

    hr = pAppDomainThunk->QueryInterface(__uuidof(mscorlib::_AppDomain), (VOID **)&pDefaultAppDomain);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pAppDomainThunk->QueryInterface(...) succeeded\n");
```

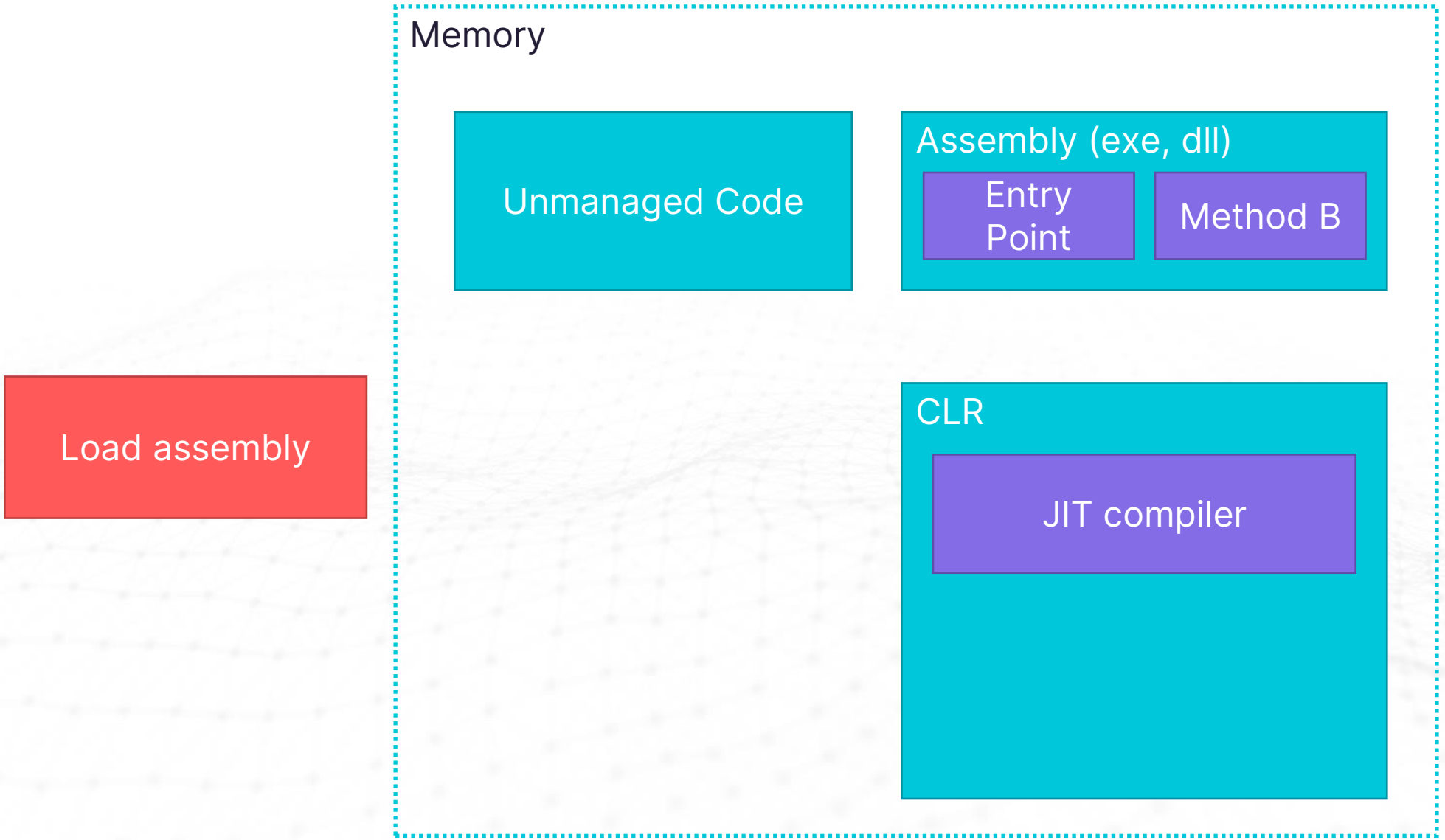
Gets an interface pointer of type **System._AppDomain**

```
int assemblyLoad(ICorRuntimeHost *pRuntimeHost,
                 mscorlib::_AssemblyPtr *pAssembly,
                 char *fileData,
                 int fileLength)
{
    HRESULT hr;
    IUnknownPtr pAppDomainThunk = NULL;
    mscorlib::_AppDomainPtr pDefaultAppDomain = NULL;
    SAFEARRAYBOUND rgsabound[1];
    SAFEARRAY *pSafeArray = NULL;
    void *pvData = NULL;

    rgsabound[0].cElements = fileLength;
    rgsabound[0].lLbound = 0;

    pSafeArray = SafeArrayCreate(VT_UI1, 1, rgsabound);

    hr = SafeArrayAccessData(pSafeArray, &pvData);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] SafeArrayAccessData(...) succeeded\n");

    memcpy(pvData, fileData, fileLength);

    hr = SafeArrayUnaccessData(pSafeArray);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] SafeArrayUnaccessData(...) succeeded\n");

    hr = pDefaultAppDomain->raw_Load_3(pSafeArray, &(*pAssembly));

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pDefaultAppDomain->Load_3(...) succeeded\n");
}
```

Prepare **SAFEARRAY**
Copy entire assembly to array

```
int assemblyLoad(ICorRuntimeHost *pRuntimeHost,
                 mscorlib::_AssemblyPtr *pAssembly,
                 char *fileData,
                 int fileLength)
{
    HRESULT hr;
    IUnknownPtr pAppDomainThunk = NULL;
    mscorlib::_AppDomainPtr pDefaultAppDomain = NULL;
    SAFEARRAYBOUND rgsabound[1];
    SAFEARRAY *pSafeArray = NULL;
    void *pvData = NULL;

    rgsabound[0].cElements = fileLength;
    rgsabound[0].lLbound = 0;

    pSafeArray = SafeArrayCreate(VT_UI1, 1, rgsabound);

    hr = SafeArrayAccessData(pSafeArray, &pvData);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] SafeArrayAccessData(...) succeeded\n");

    memcpy(pvData, fileData, fileLength);

    hr = SafeArrayUnaccessData(pSafeArray);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] SafeArrayUnaccessData(...) succeeded\n");

    hr = pDefaultAppDomain->raw_Load_3(pSafeArray, &(*pAssembly));

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pDefaultAppDomain->Load_3(...) succeeded\n");
}
```

Load the assembly to CLR



```
int assemblyRun(mscorlib::_AssemblyPtr pAssembly, int argc, char *argv[])
{
    HRESULT hr;
    mscorlib::_MethodInfoPtr pMethodInfo = NULL;
    VARIANT retVal;
    VARIANT obj;
    VARIANT args;
    SAFEARRAYBOUND argsBound[1];
    long idx[1];
    SAFEARRAY *params = NULL;
    SAFEARRAYBOUND paramsBound[1];

    hr = pAssembly->get_EntryPoint(&pMethodInfo);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pAssembly->get_EntryPoint(...) succeeded\n");
```

Get **MethodInfoPtr**
of entry point

```
int assemblyRun(mscorlib::_AssemblyPtr pAssembly, int argc, char *argv[])
{
    HRESULT hr;
    mscorlib::_MethodInfoPtr pMethodInfo = NULL;
    VARIANT retVal;
    VARIANT obj;
    VARIANT args;
    SAFEARRAYBOUND argsBound[1];
    long idx[1];
    SAFEARRAY *params = NULL;
    SAFEARRAYBOUND paramsBound[1];

    ZeroMemory(&retVal, sizeof(VARIANT));
    ZeroMemory(&obj, sizeof(VARIANT));
    obj.vt = VT_NULL;

    args.vt = VT_ARRAY | VT_BSTR;
    argsBound[0].lLbound = 0;
    argsBound[0].cElements = argc;
    args.parray = SafeArrayCreate(VT_BSTR, 1, argsBound);
    for (int i = 0; i < argc; i++) {
        std::wstring wc(strlen(argv[i]), L'#');
        mbstowcs(&wc[0], argv[i], strlen(argv[i]));
        idx[0] = i;
        SafeArrayPutElement(args.parray, idx, SysAllocString(wc.c_str()));
    }
    paramsBound[0].lLbound = 0;
    paramsBound[0].cElements = 1;
    params = SafeArrayCreate(VT_VARIANT, 1, paramsBound);
    idx[0] = 0;
    SafeArrayPutElement(params, idx, &args);
```

Prepare parameter for the assembly

```
int assemblyRun(mscorlib::_AssemblyPtr pAssembly, int argc, char *argv[])
{
    HRESULT hr;
    mscorlib::_MethodInfoPtr pMethodInfo = NULL;
    VARIANT retVal;
    VARIANT obj;
    VARIANT args;
    SAFEARRAYBOUND argsBound[1];
    long idx[1];
    SAFEARRAY *params = NULL;
    SAFEARRAYBOUND paramsBound[1];

    // hr = 8002000E: https://github.com/etormadiv/HostingCLR/issues/4
    hr = pMethodInfo->raw_Invoke_3(obj, params, &retVal);

    if (FAILED(hr)) { ... }
    logPrintf(LOG_LEVEL_DEBUG, "[*] pMethodInfo->Invoke_3(...) succeeded\n");
}
```

Invoke entry point!