Open Source .NET Interop Debugger

Gleb Balykov

Plan

- What is a Debugger

Plan

- What is a Debugger
- What is NetCoreDbg

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg
- How to use NetCoreDbg

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg
- How to use NetCoreDbg
- What is interop debugging

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg
- How to use NetCoreDbg
- What is interop debugging
- Features of interop NetCoreDbg

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg
- How to use NetCoreDbg
- What is interop debugging
- Features of interop NetCoreDbg
- How to use interop NetCoreDbg

Plan

- What is a Debugger
- What is NetCoreDbg
- Features and architecture of NetCoreDbg
- How to use NetCoreDbg
- What is interop debugging
- Features of interop NetCoreDbg
- How to use interop NetCoreDbg
- Future interop debugging directions

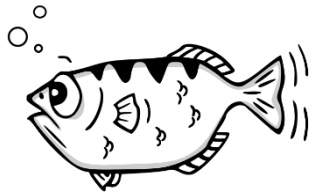
What is a Debugger?

Features:

- backtraces
- breakpoints
- evaluation of expressions
- stepping
- etc.

Popular Debuggers

- GDB

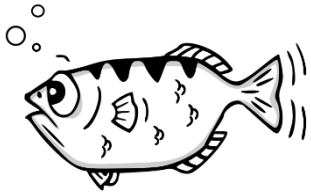


Popular Debuggers

- GDB
- LLDB

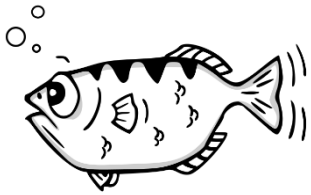
Popular Debuggers

- GDB
- LLDB
- Visual Studio Debugger



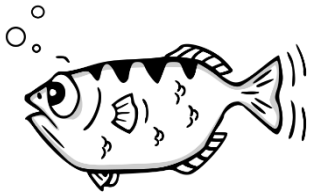
Popular Debuggers

- GDB
- LLDB
- Visual Studio Debugger
- WinDbg



Popular Debuggers

- GDB
- LLDB
- Visual Studio Debugger
- WinDbg
- Android Studio Debugger



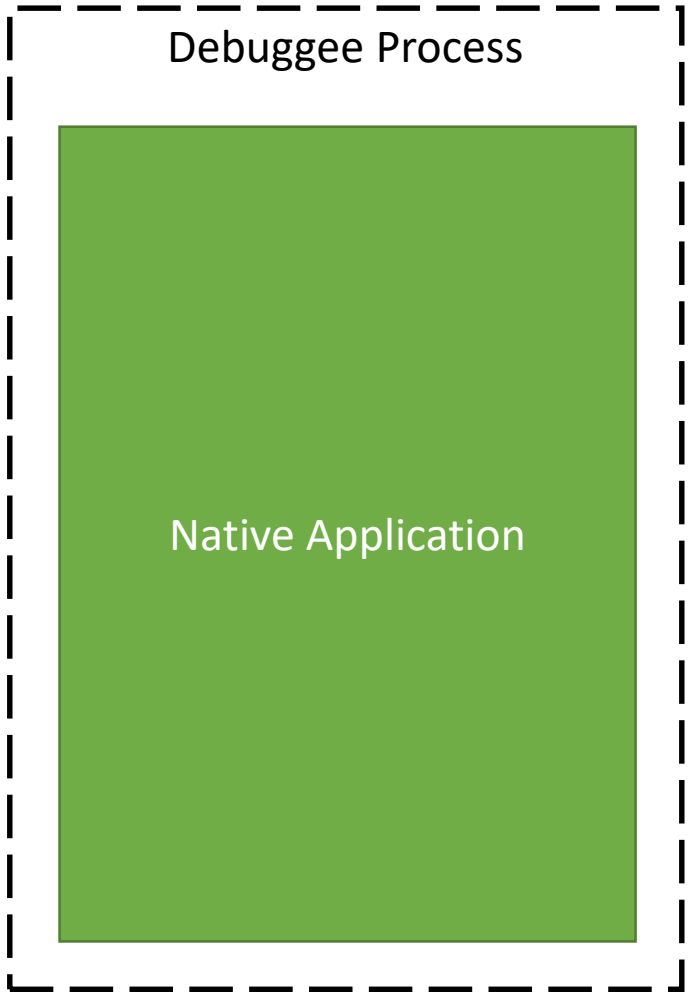

Popular Debuggers

- GDB
- LLDB
- Visual Studio Debugger
- WinDbg
- Android Studio Debugger
- Firefox JavaScript Debugger

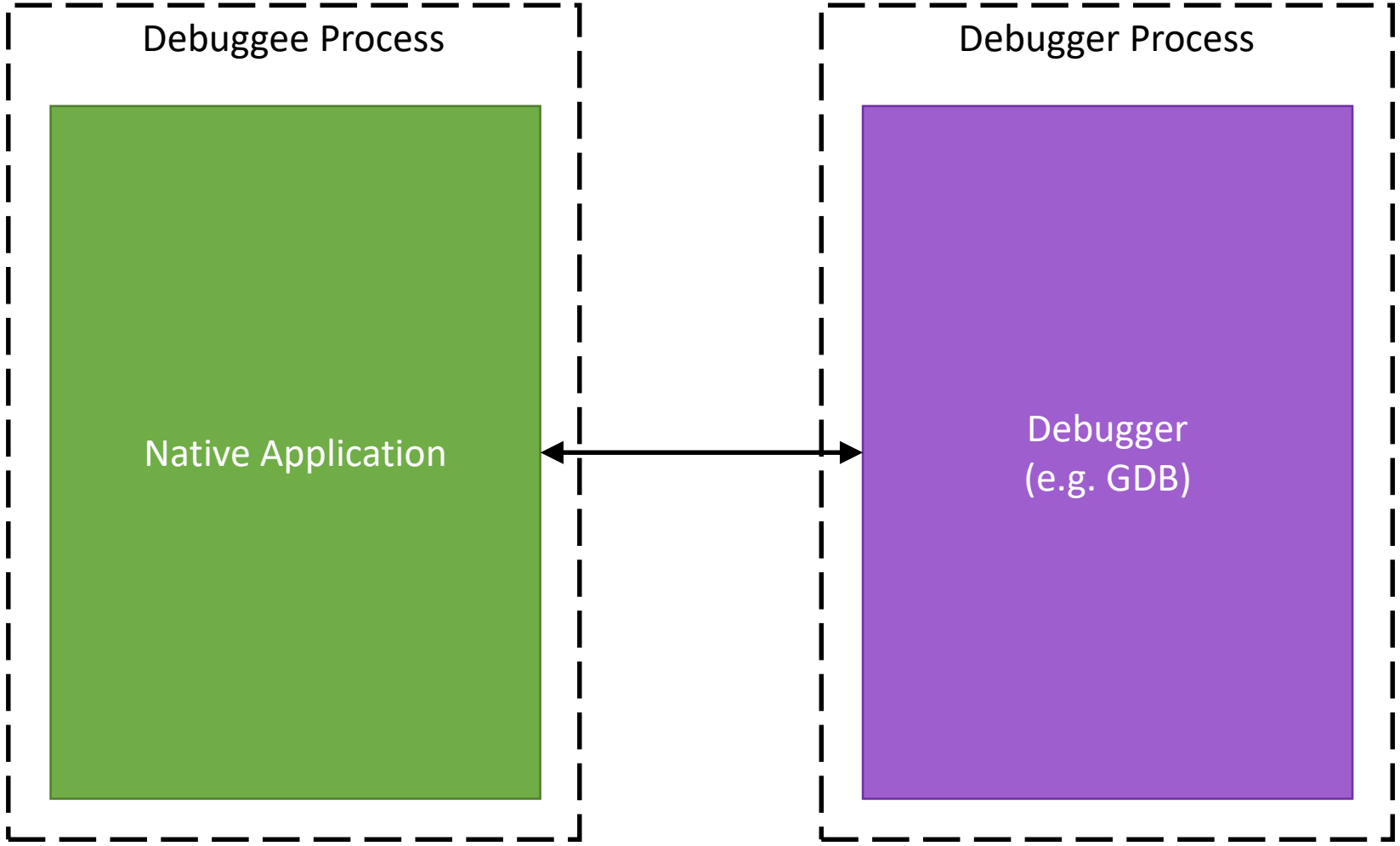
Popular Debuggers

- GDB
- LLDB
- Visual Studio Debugger
- WinDbg
- Android Studio Debugger
- Firefox JavaScript Debugger
- and many more…

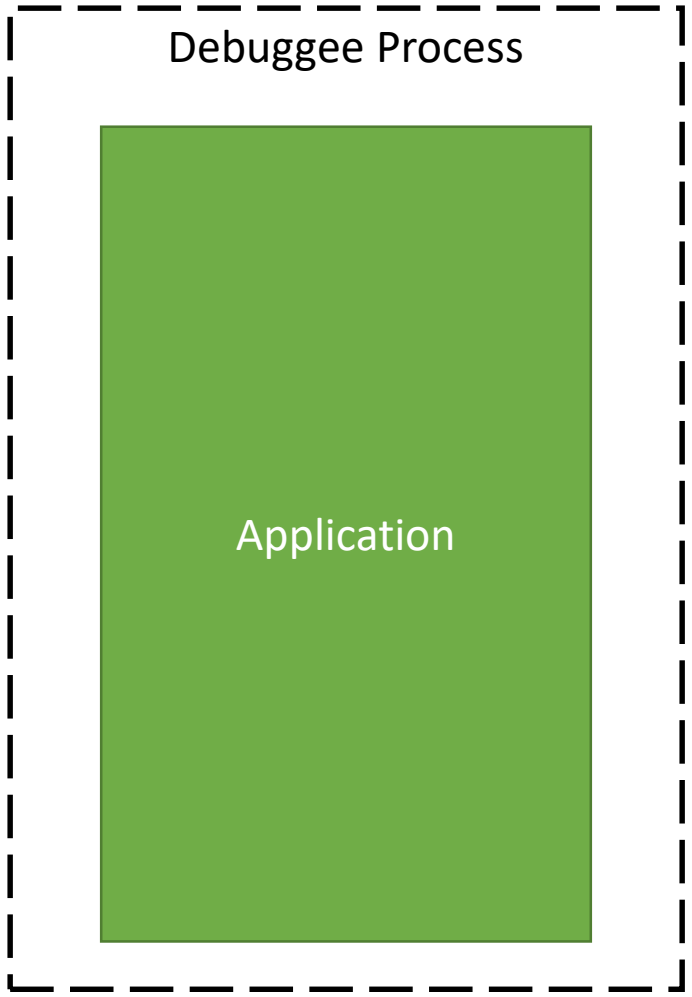
Debugger Architecture (GDB)



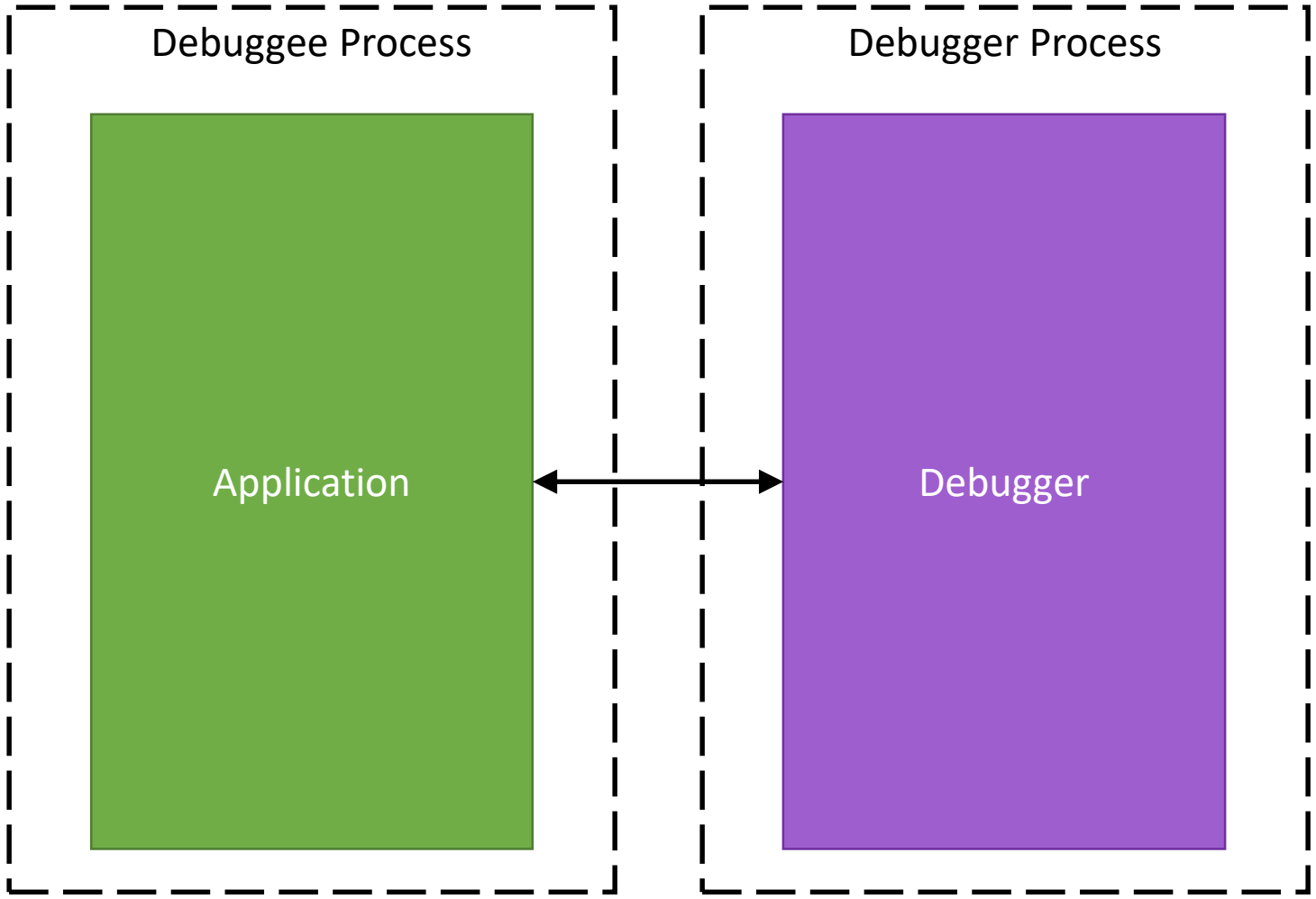
Debugger Architecture (GDB)



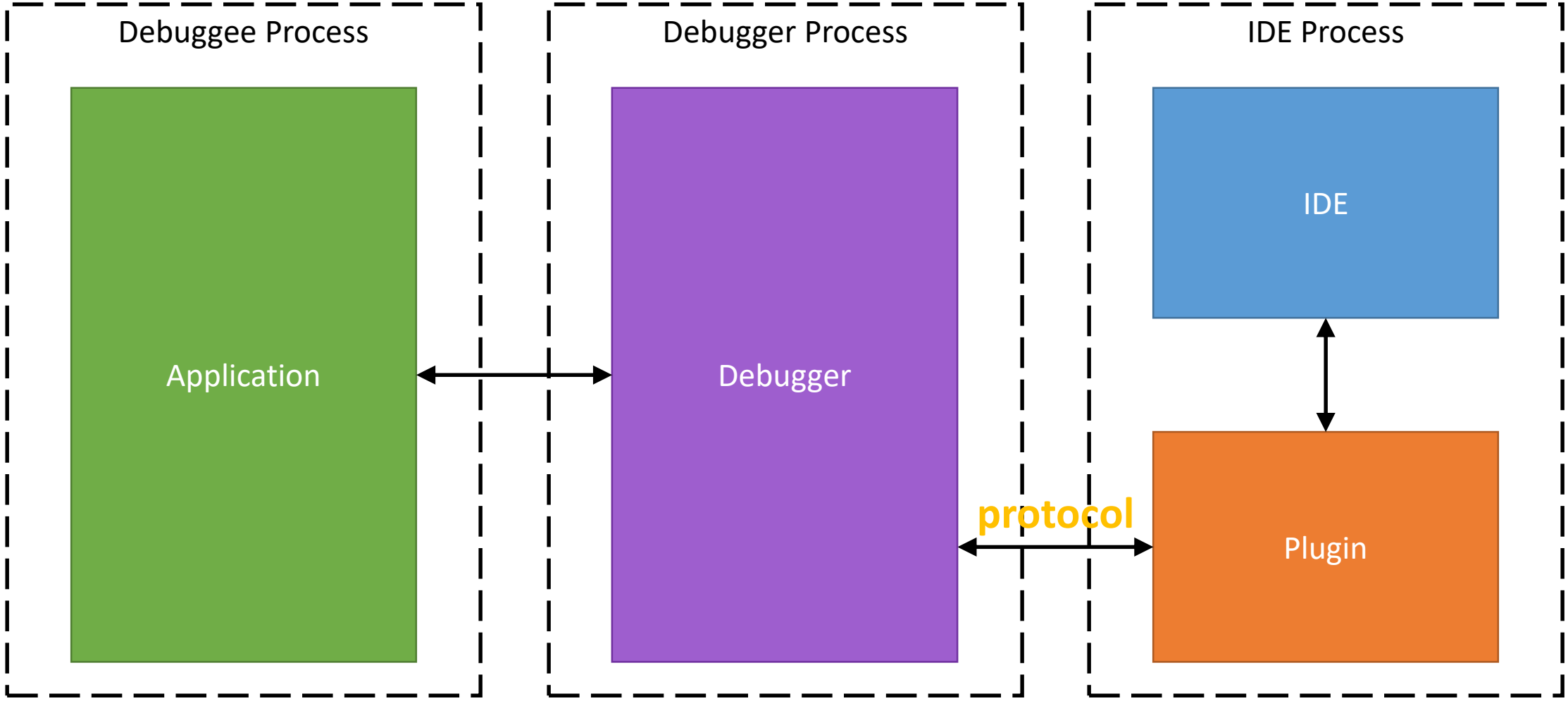
IDE Debugger Architecture

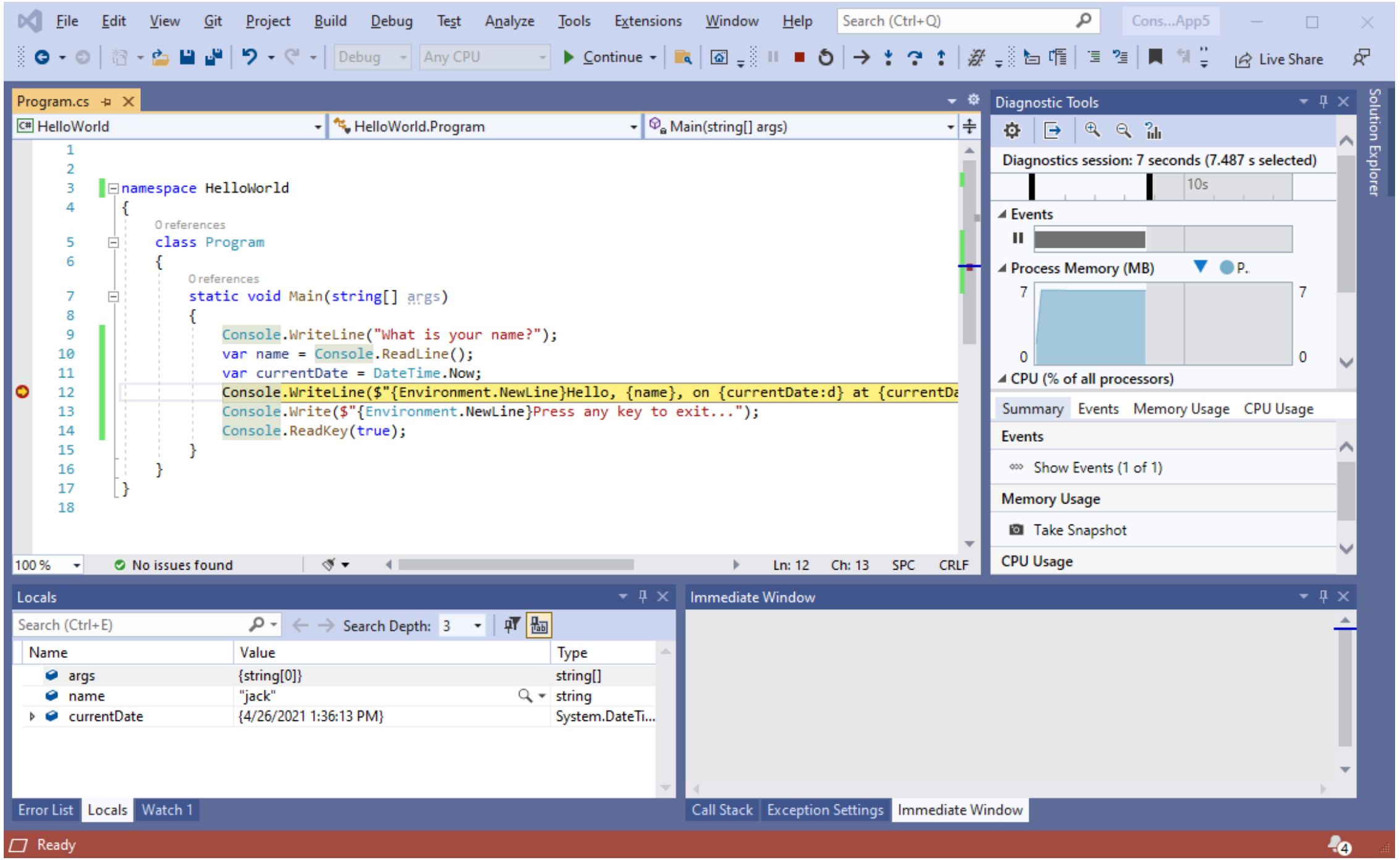


IDE Debugger Architecture



IDE Debugger Architecture




File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Any CPU
Continue
Live Share
Program.cs
HelloWorld
HelloWorld.Program
Main(string[] args)
namespace HelloWorld
{
0 references
class Program
{
0 references
static void Main(string[] args)
{
Console.WriteLine("What is your name?");
var name = Console.ReadLine();
var currentDate = DateTime.Now;
Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDa
Console.Write($"{Environment.NewLine}Press any key to exit...");
Console.ReadKey(true);
}
}
}
100 %
No issues found
Ln: 12
Ch: 13
SPC
CRLF
Diagnostic Tools
Diagnostics session: 7 seconds (7.487 s selected)
10s
Events
Process Memory (MB)
P.
7
0
CPU (% of all processors)
Summary
Events
Memory Usage
CPU Usage
Show Events (1 of 1)
Take Snapshot
Solution Explorer
Locals
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
args
{string[0]}
string[]
name
"jack"
string
currentDate
{4/26/2021 1:36:13 PM}
System.DateTi...
Error List
Locals
Watch 1
Immediate Window
Call Stack
Exception Settings
Immediate Window
Ready

Why one more .NET debugger was needed?

- GDB JIT
- LLDB + SOS plugin
- Mono Soft Debugger
- Visual Studio Debugger
- VS Code Debugger (vsdbg)

Why one more .NET debugger was needed?

- GDB JIT (no evaluation)
- LLDB + SOS plugin (no evaluation)
- Mono Soft Debugger (different debug API)
- Visual Studio Debugger (proprietary)
- VS Code Debugger (vsdbg) (proprietary)

Why one more .NET debugger was needed?

ICorDebug API

- Since .NET Core 2.1
- Rich debugging functionality

NetCoreDbg

NetCoreDbg

- Open Source .NET debugger (MIT license)

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg

- Open Source .NET debugger (MIT license)
- Developed by Samsung

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg

- Open Source .NET debugger (MIT license)
- Developed by Samsung
- Tizen/Linux/MacOS/Windows support

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg

- Open Source .NET debugger (MIT license)
- Developed by Samsung
- Tizen/Linux/MacOS/Windows support
- Arm/Arm64/x86/x64 support

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg

- Open Source .NET debugger (MIT license)
- Developed by Samsung
- Tizen/Linux/MacOS/Windows support
- Arm/Arm64/x86/x64 support
- Based on ICorDebug API

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg

- Open Source .NET debugger (MIT license)
- Developed by Samsung
- Tizen/Linux/MacOS/Windows support
- Arm/Arm64/x86/x64 support
- Based on ICorDebug API
- Supports .NET Core 2.1+, .NET 5+

GitHub repo: https://github.com/samsung/netcoredbg

NetCoreDbg: Features

- GDB MI (Visual Studio) and VS Code protocols support

NetCoreDbg: Features

- GDB MI (Visual Studio) and VS Code protocols support
- Integration with Visual Studio

NetCoreDbg: Features

- GDB MI (Visual Studio) and VS Code protocols support
- Integration with Visual Studio
- Integration with VS Code

NetCoreDbg: Features

- GDB MI (Visual Studio) and VS Code protocols support
- Integration with Visual Studio
- Integration with VS Code
- GDB-like command line interface (CLI)

NetCoreDbg: Features

- GDB MI (Visual Studio) and VS Code protocols support
- Integration with Visual Studio
- Integration with VS Code
- GDB-like command line interface (CLI)
- Stack unwinding, stepping, breakpoints, expression evaluation

NetCoreDbg: Open Source Community

- Addition to various distros (Arch, NixOS)

NetCoreDbg: Open Source Community

- Addition to various distros (Arch, NixOS)
- Integration to Vim via Vimspector

NetCoreDbg: Open Source Community

- Addition to various distros (Arch, NixOS)
- Integration to Vim via Vimspector
- Integration to Eclipse

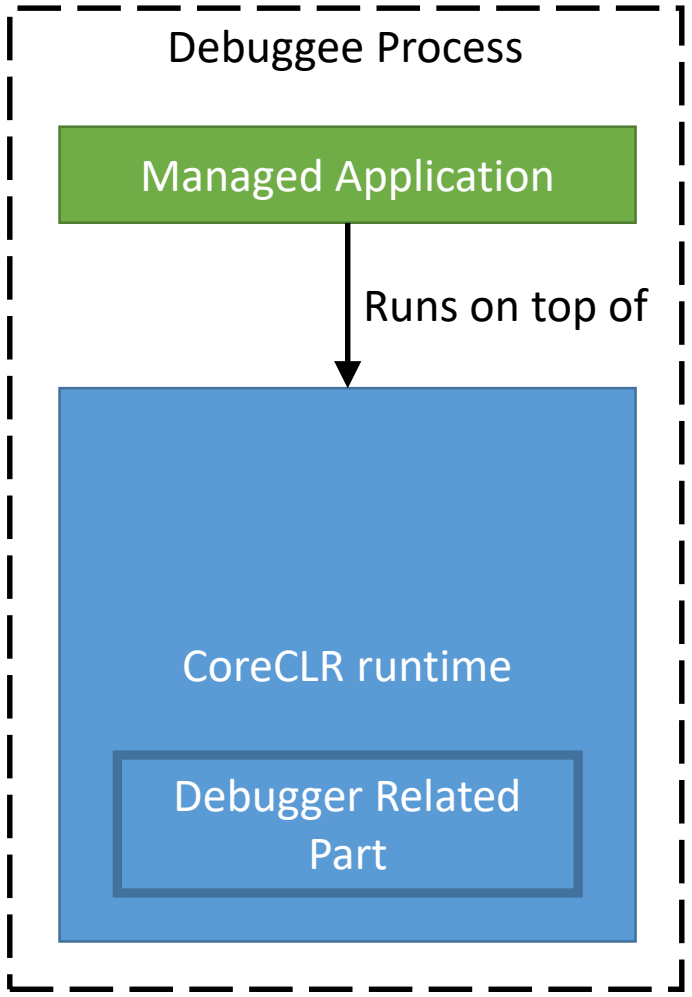
NetCoreDbg: Open Source Community

- Addition to various distros (Arch, NixOS)
- Integration to Vim via Vimspector
- Integration to Eclipse
- Integration to Red Hat Code Ready

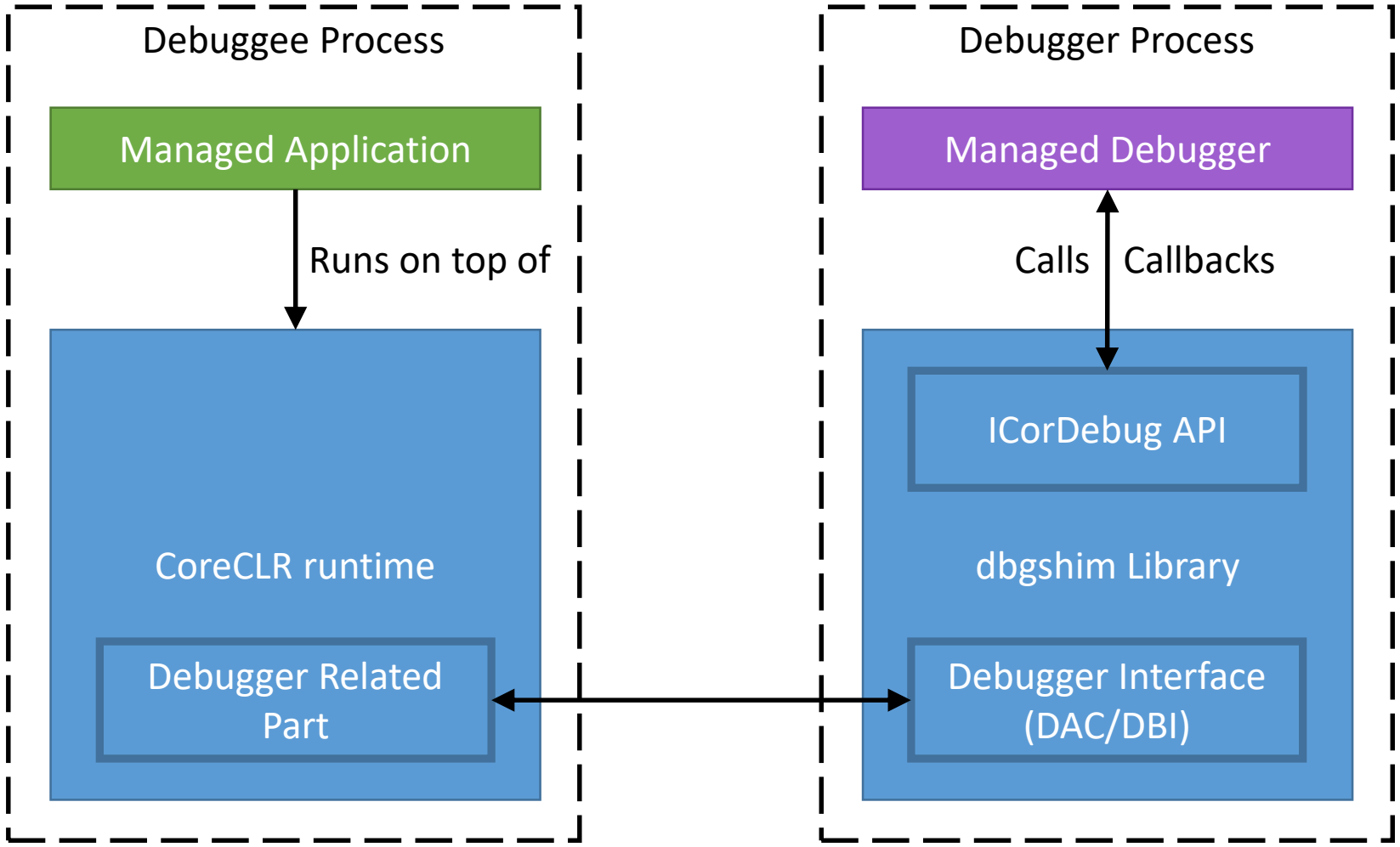
NetCoreDbg: Open Source Community

- Addition to various distros (Arch, NixOS)
- Integration to Vim via Vimspector
- Integration to Eclipse
- Integration to Red Hat Code Ready
- Apple M1 support

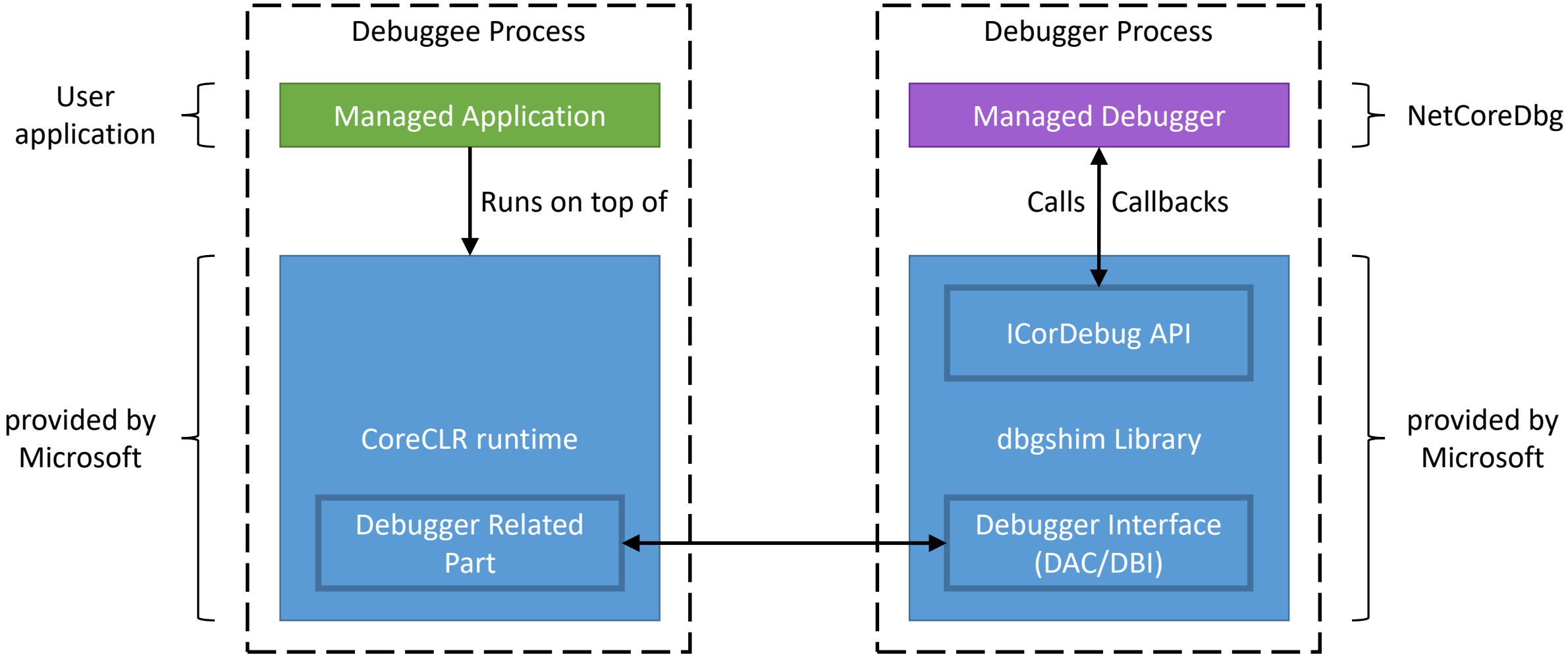
NetCoreDbg Architecture



NetCoreDbg Architecture



NetCoreDbg Architecture

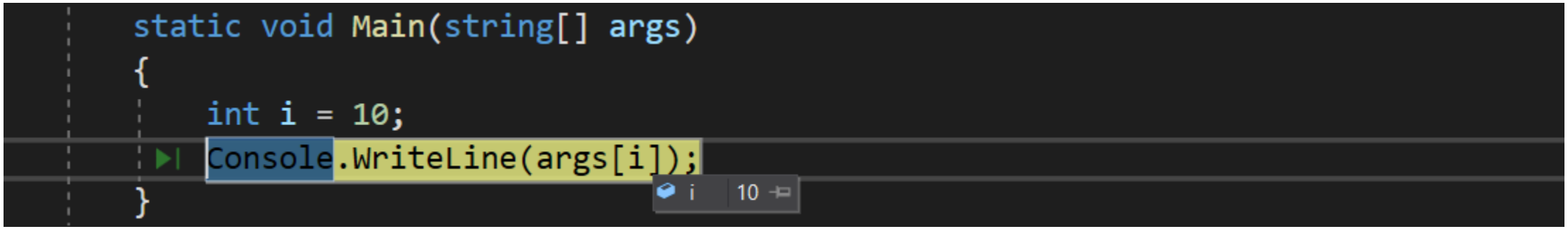


Expression Evaluation

One of the main features of the debugger

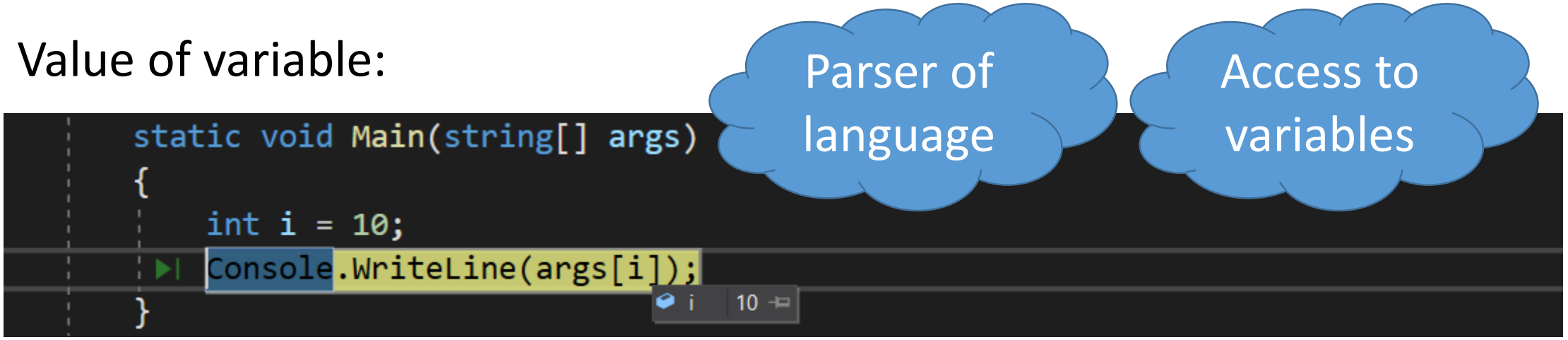
Expression Evaluation

Value of variable:



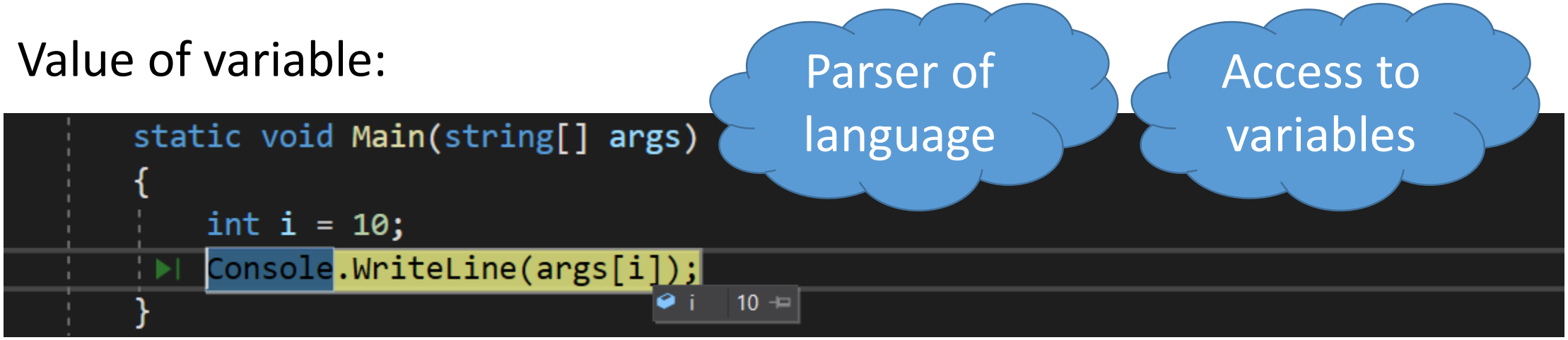
Expression Evaluation

Value of variable:



Expression Evaluation

Value of variable:



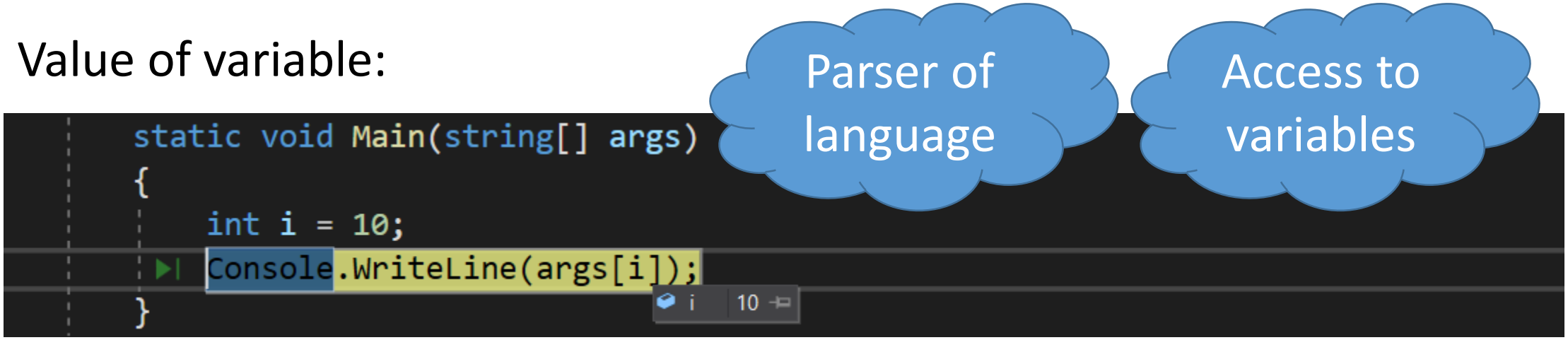
Field of object:

```
static void Main(string[] args)
{
    Coordinate coord = new Coordinate(100, 200);
    Console.WriteLine(coord.x);
    Console.WriteLine(coord.y);
}
```

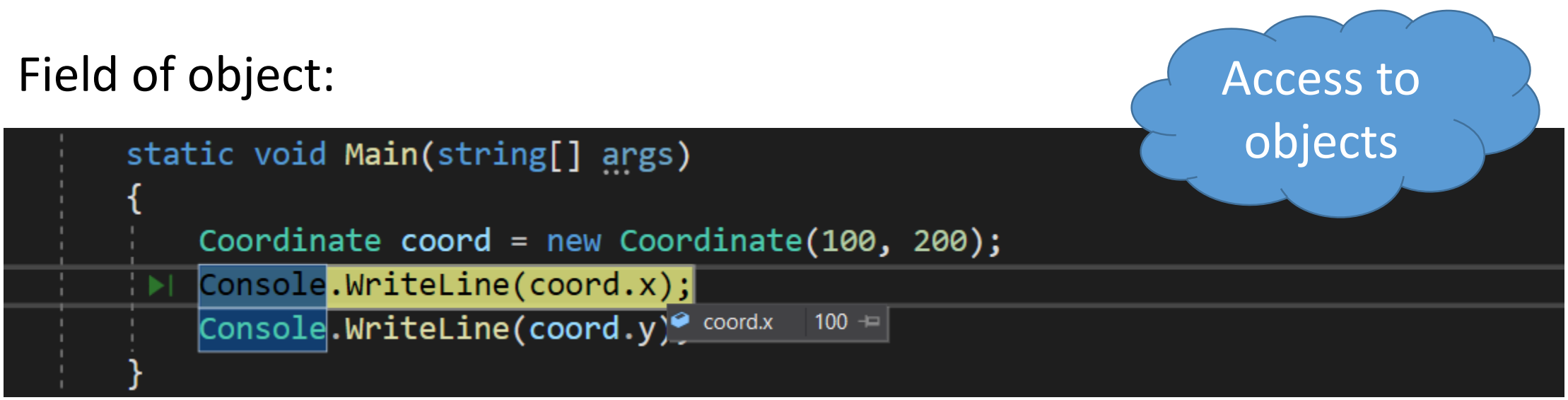
coord.x 100

Expression Evaluation

Value of variable:



Field of object:



Expression Evaluation

Function call:

```
public int getSquare()
{
    return x * y;
}
```

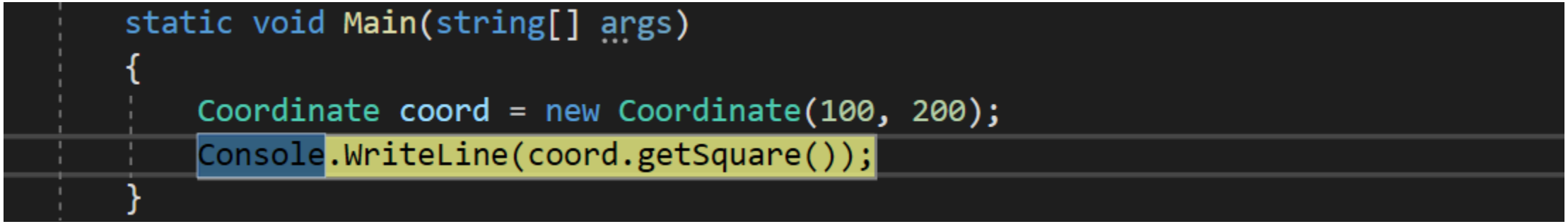
```
static void Main(string[] args)
{
    Coordinate coord = new Coordinate(100, 200);
    Console.WriteLine(coord.getSquare());
}
```

Name	Value	Type
coord.getSquare()	20000	int
Add item to watch		

Expression Evaluation

Function call:

Code execution



Name	Value	Type
coord.getSquare()	20000	int
Add item to watch		

Expression Evaluation

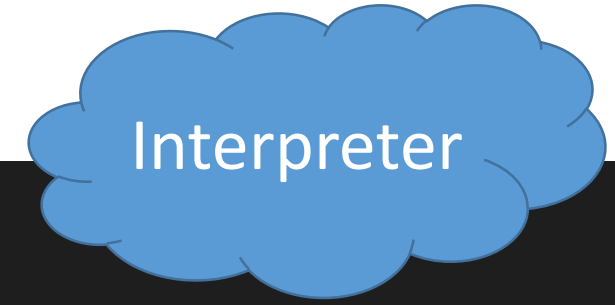
Computations:

```
static void Main(string[] args)
{
    double[] array = { 1, 2, 3, 4, 5 };
    Console.WriteLine(array[array.Length - 1]);
}
```

Name	Value	Type
array[array.Length - 1]	5	double
Add item to watch		

Expression Evaluation

Computations:




```
static void Main(string[] args)
{
    double[] array = { 1, 2, 3, 4, 5 };
    Console.WriteLine(array[array.Length - 1]);
}
```

Name	Value	Type
array[array.Length - 1]	5	double
Add item to watch		

Expression Evaluation

- Parser
- Interpreter

Expression Evaluation: how to execute code?

- in debugger process

- in debuggee process

Expression Evaluation: how to execute code?

- in debugger process
  need to copy all required objects and load all libs
  affects performance and memory consumption
  faster for arithmetic operations
- in debuggee process

Expression Evaluation: how to execute code?

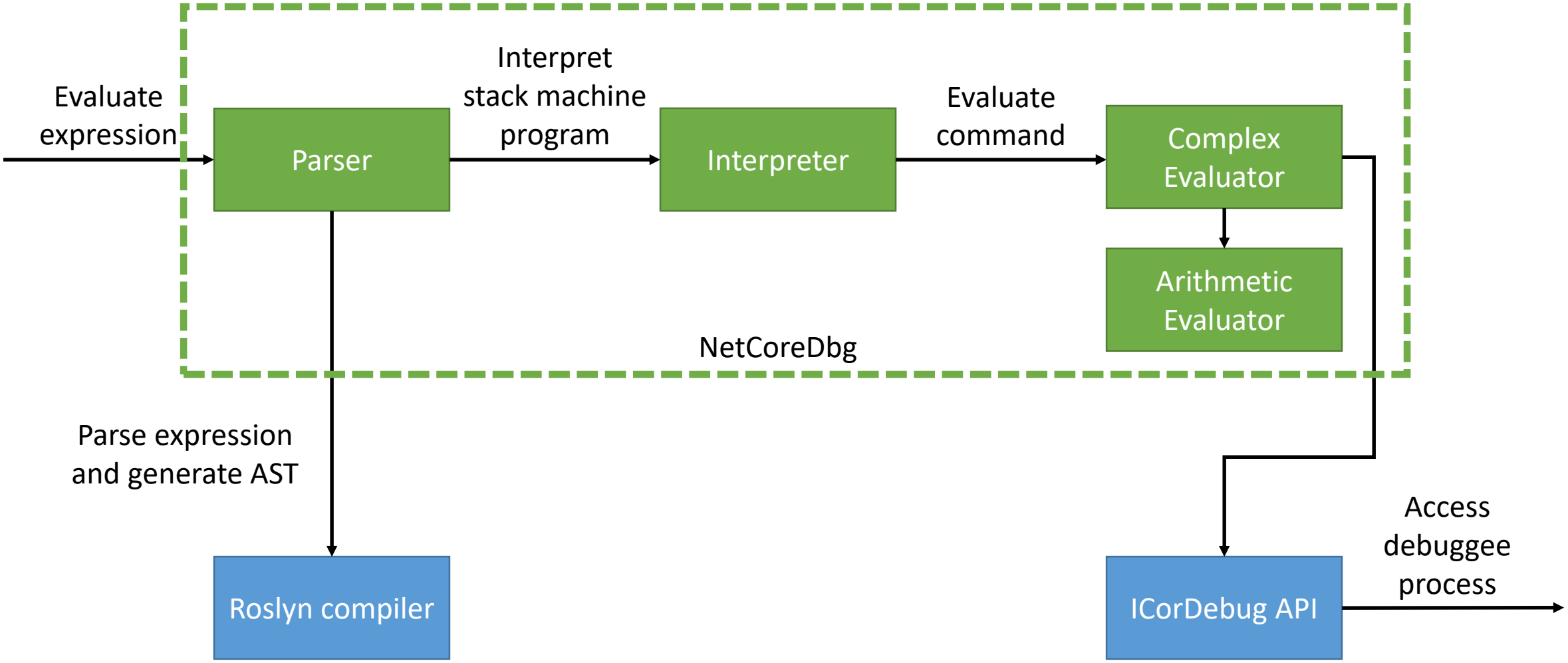
- in debugger process
  need to copy all required objects and load all libs
  affects performance and memory consumption
  faster for arithmetic operations
- in debuggee process
  state may change due to side effects
  possible dead locks
  faster for complex expressions

Expression Evaluation: how to execute code?

- Arithmetic operations on side of debugger
- Other operations on side of debuggee

Both use ICorDebug API

Expression Evaluation: Architecture



Expression Evaluation: Parsing

Roslyn AST for expression `foo(boo(), a.b.c)`

```
InvocationExpression --- foo(boo(), a.b.c)
    IdentifierName --- foo
    ArgumentList --- (boo(), a.b.c)
        Argument --- boo()
            InvocationExpression --- boo()
                IdentifierName --- boo
                ArgumentList --- ()
        Argument --- a.b.c
            SimpleMemberAccessExpression --- a.b.c
                SimpleMemberAccessExpression --- a.b
                    IdentifierName --- a
                    IdentifierName --- b
                IdentifierName --- c
```

Expression Evaluation: Parsing

Roslyn AST for expression `a+b-c*d/e%f`

```
SubtractExpression --- a+b-c*d/e%f
    AddExpression --- a+b
        IdentifierName --- a
        IdentifierName --- b
    ModuleExpression --- c*d/e%f
        DivideExpression --- c*d/e
            MultiplyExpression --- c*d
                IdentifierName --- c
                IdentifierName --- d
            IdentifierName --- e
        IdentifierName --- f
```

Expression Evaluation: Parsing

Roslyn AST for expression `a+b-c*d/e%f`

```
SubtractExpression --- a+b-c*d/e%f
    AddExpression --- a+b
        IdentifierName --- a
        IdentifierName --- b
    ModuleExpression --- c*d/e%f
        DivideExpression --- c*d/e
            MultiplyExpression --- c*d
                IdentifierName --- c
                IdentifierName --- d
            IdentifierName --- e
        IdentifierName --- f
```

Depth-First Search

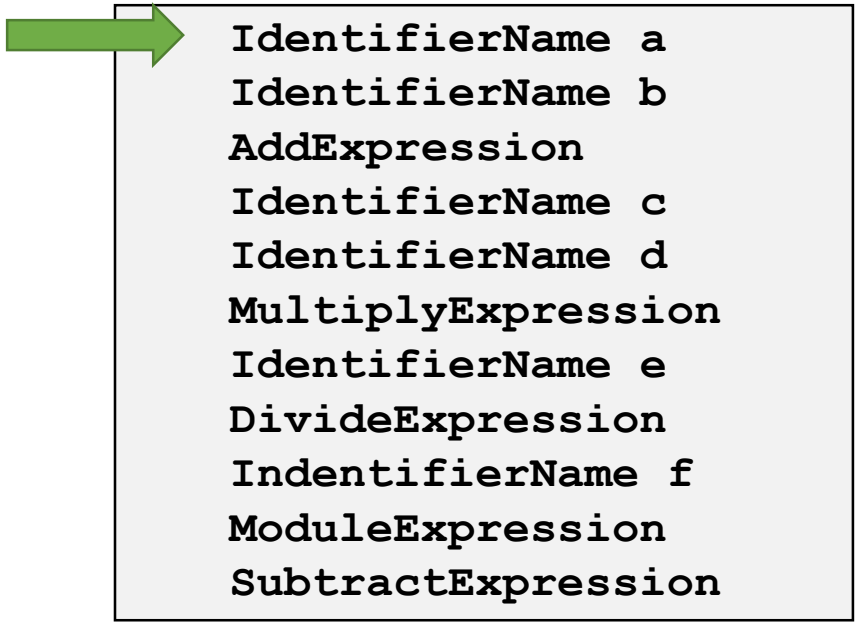
Expression Evaluation: Parsing

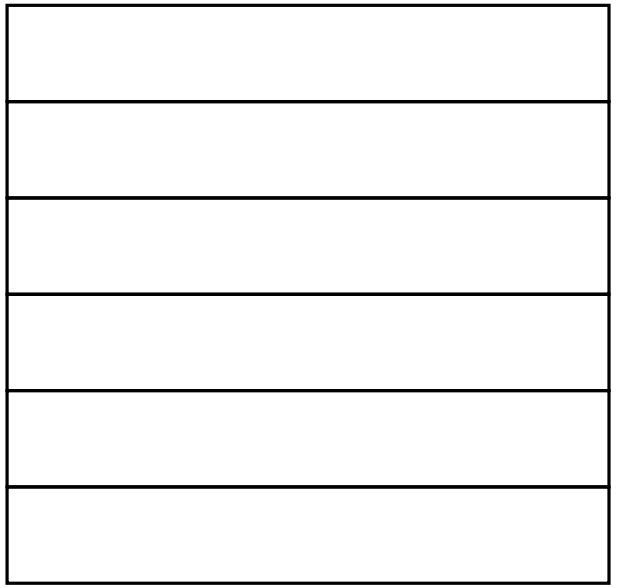
Stack machine program for expression `a+b-c*d/e%f`

```
IdentifierName a
IdentifierName b
AddExpression
IdentifierName c
IdentifierName d
MultiplyExpression
IdentifierName e
DivideExpression
IndentifierName f
ModuleExpression
SubtractExpression
```

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

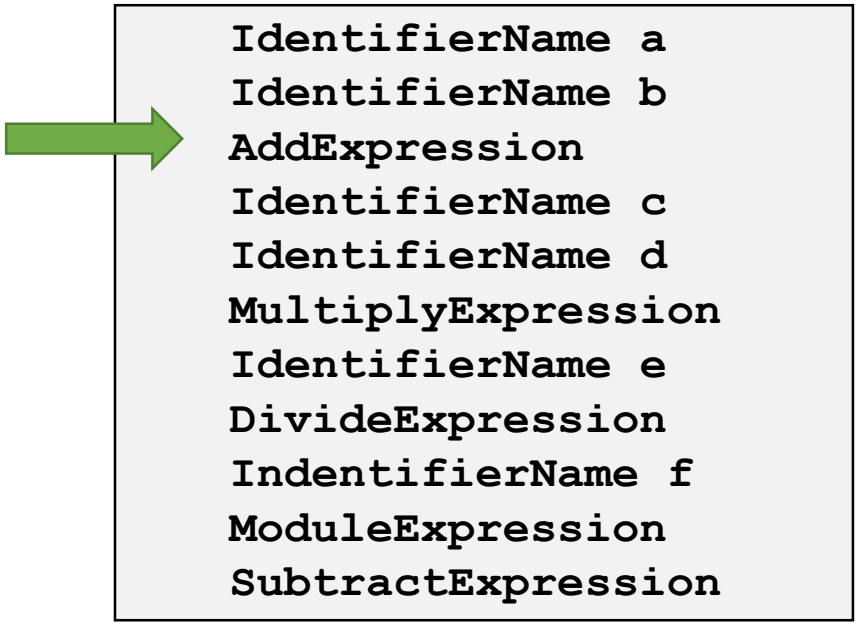


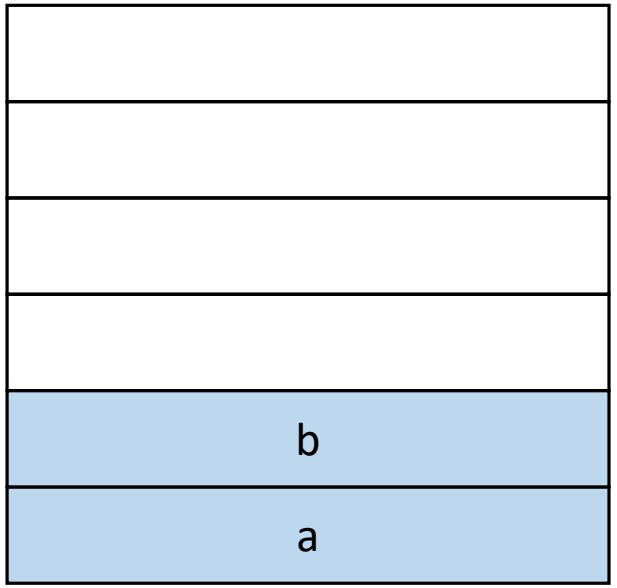


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

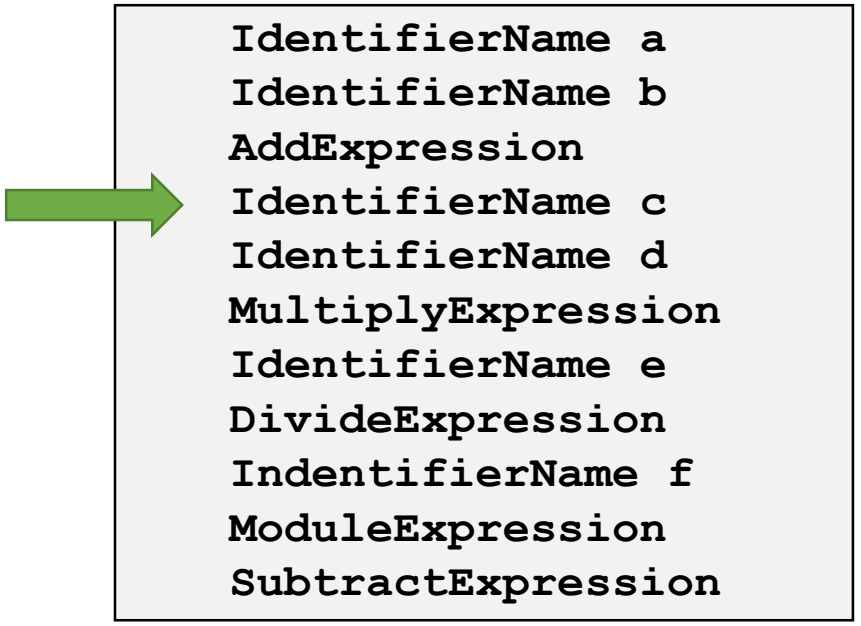


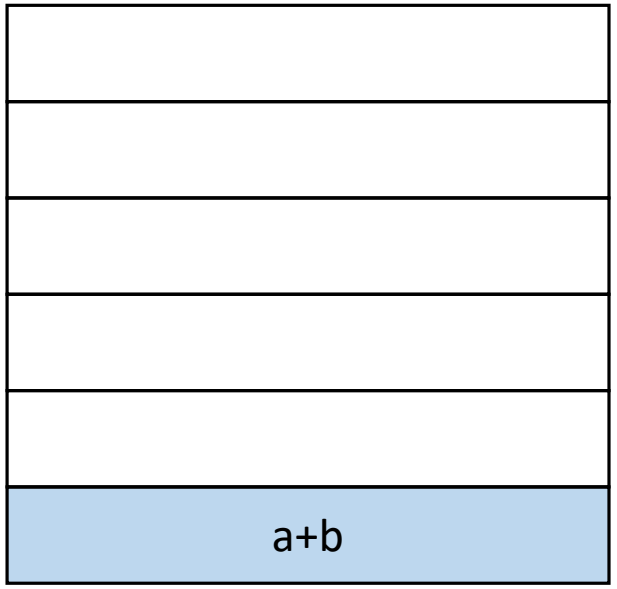


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

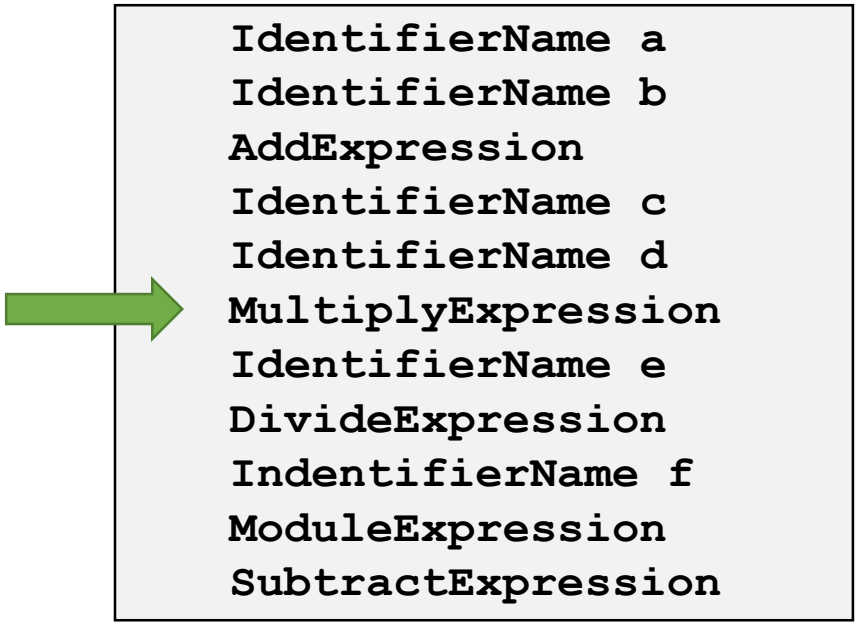


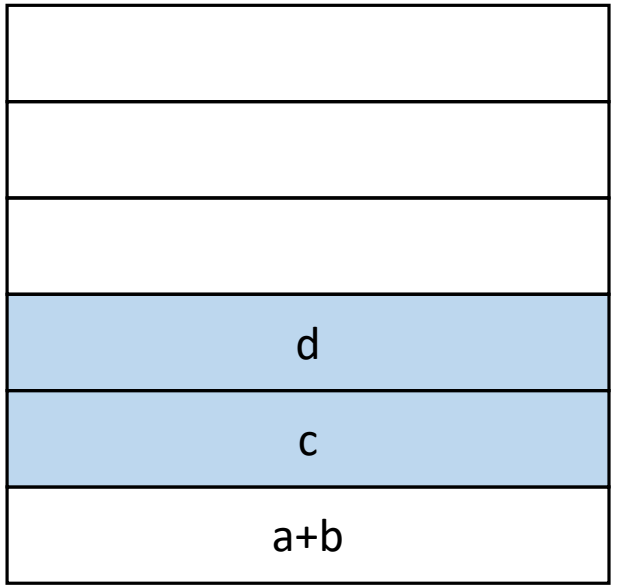


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

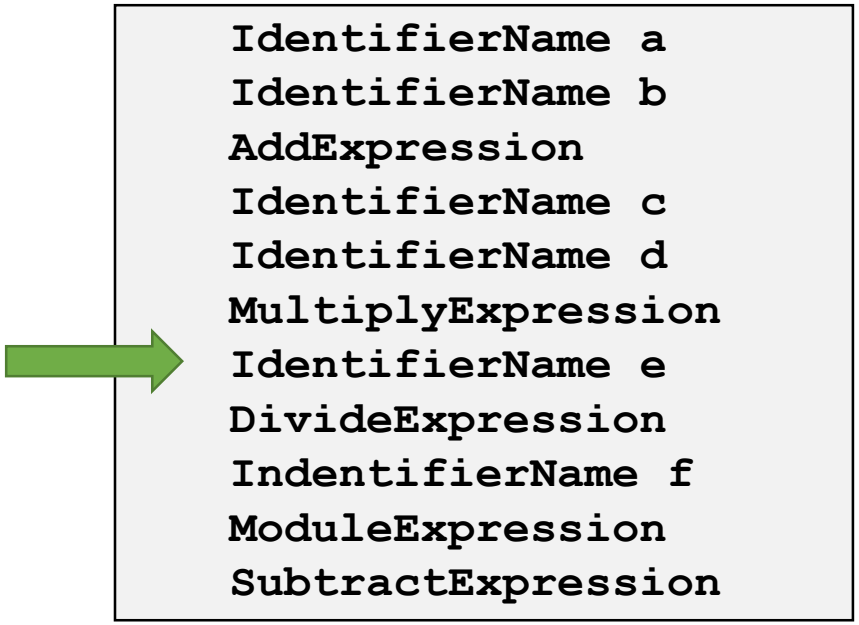


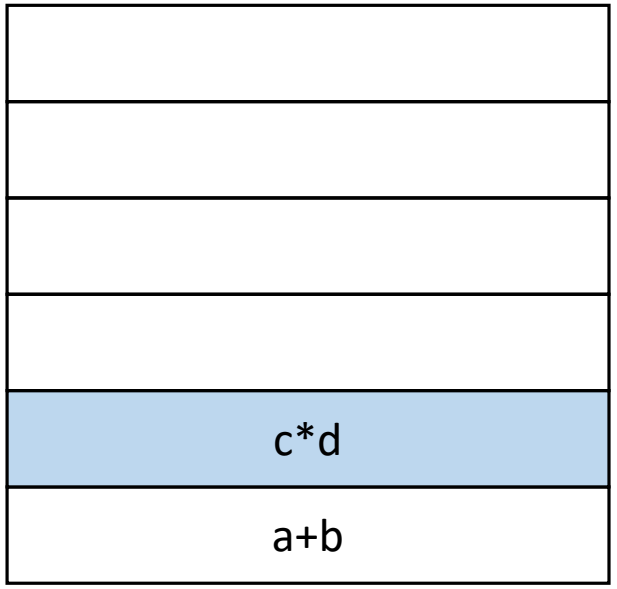


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

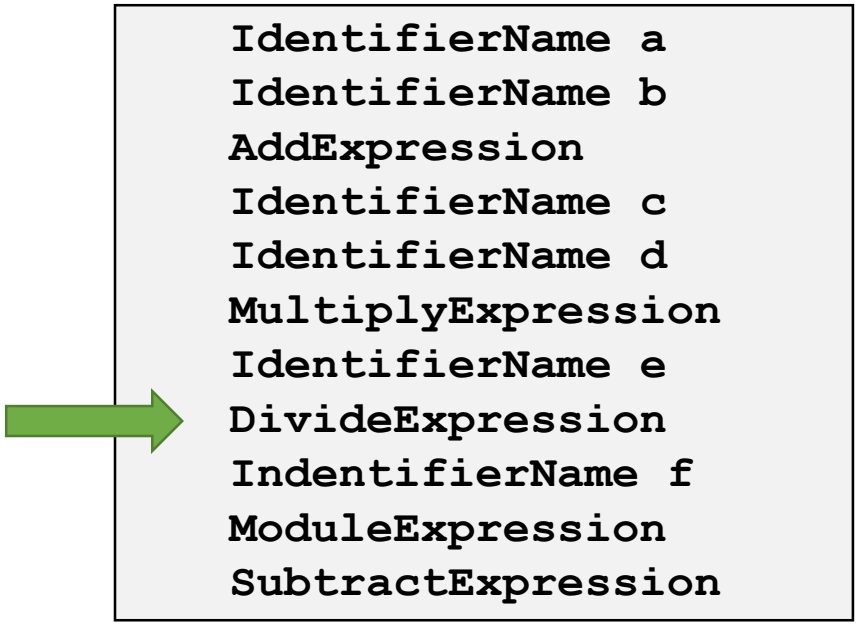


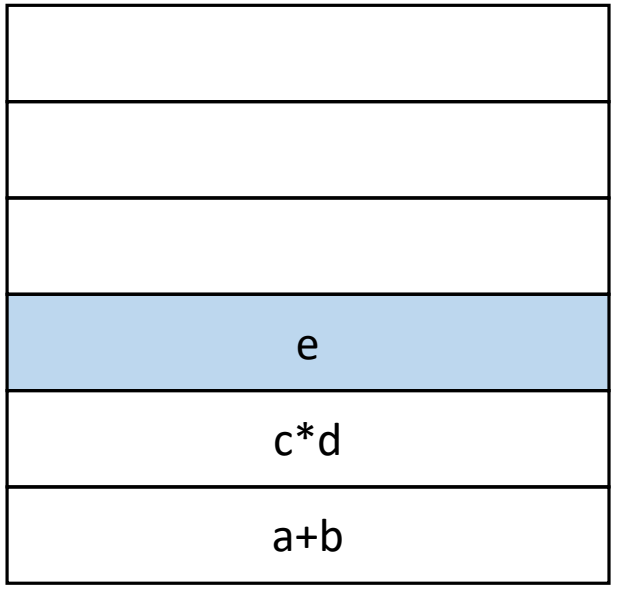


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

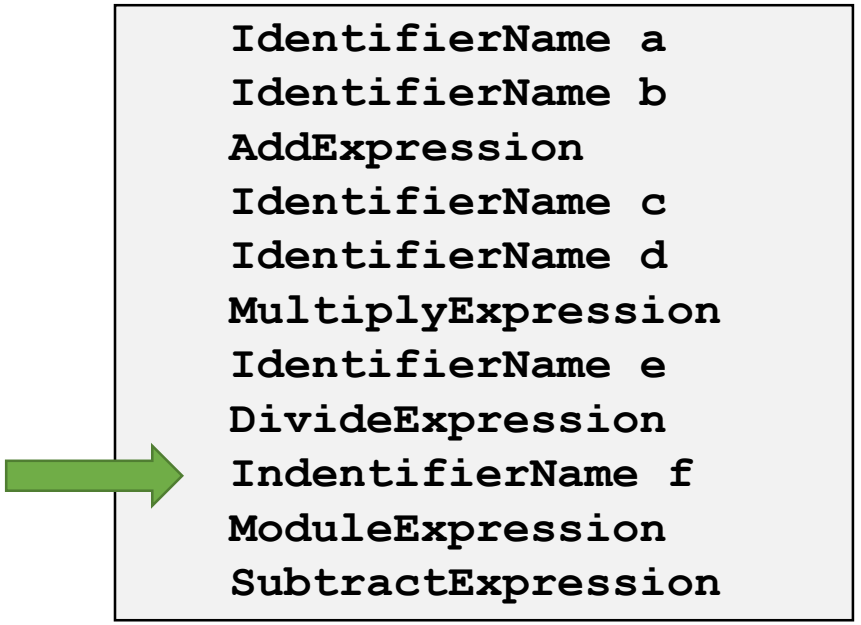


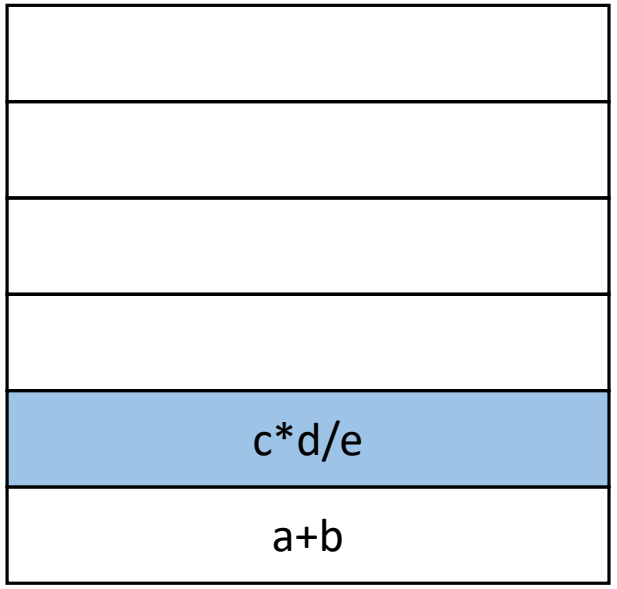


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`



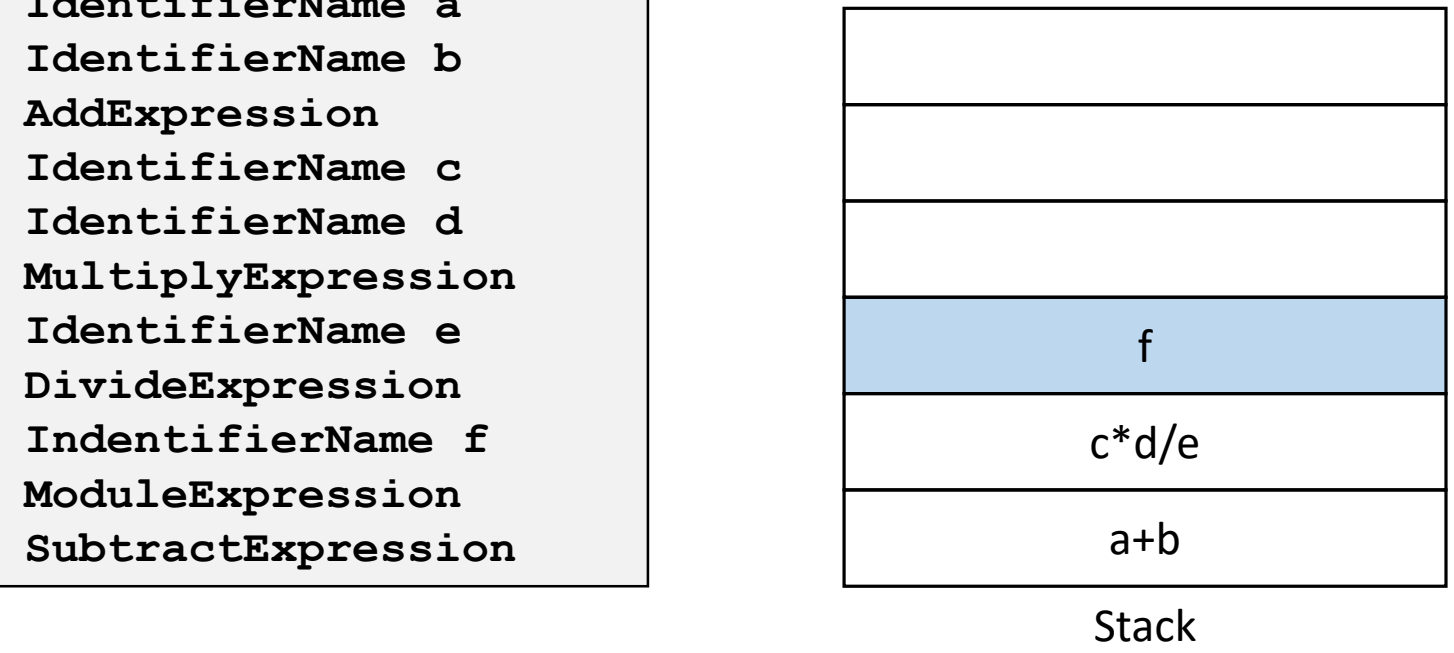


Stack

Expression Evaluation: Interpretation

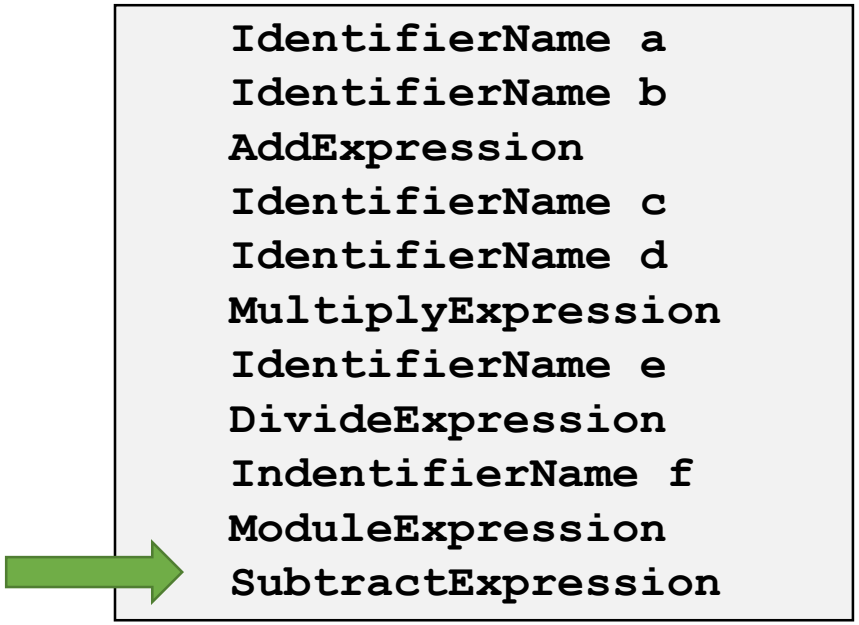
Stack machine program for expression `a+b-c*d/e%f`

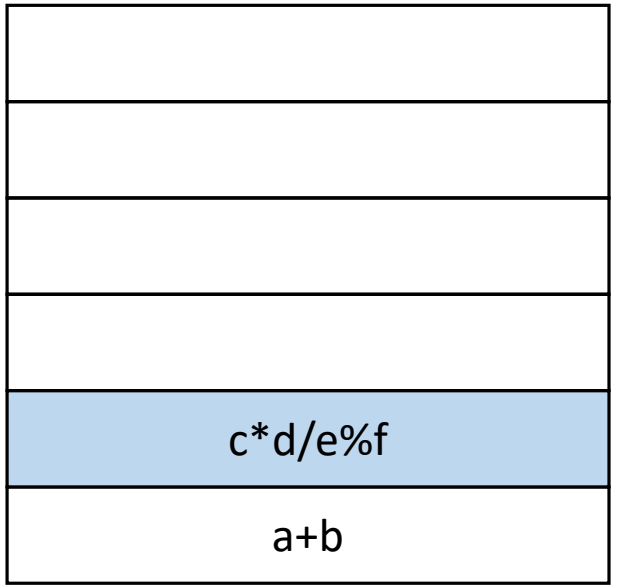
```
IdentifierName a
IdentifierName b
AddExpression
IdentifierName c
IdentifierName d
MultiplyExpression
IdentifierName e
DivideExpression
IndentifierName f
ModuleExpression
SubtractExpression
```



Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`



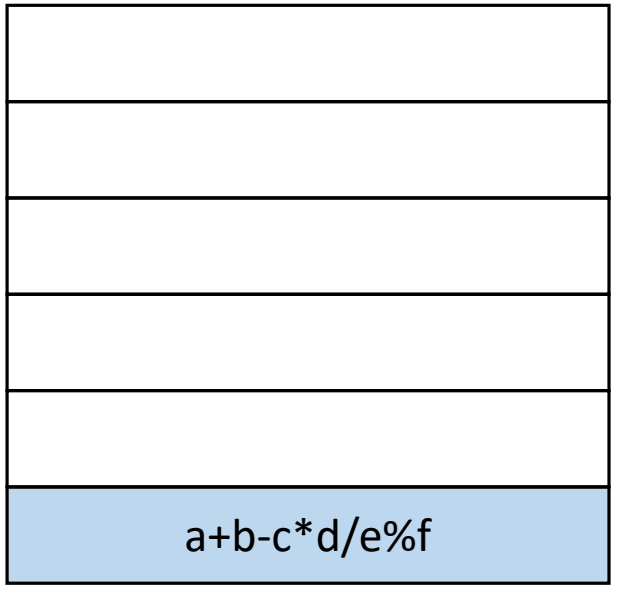


Stack

Expression Evaluation: Interpretation

Stack machine program for expression `a+b-c*d/e%f`

```
IdentifierName a
IdentifierName b
AddExpression
IdentifierName c
IdentifierName d
MultiplyExpression
IdentifierName e
DivideExpression
IndentifierName f
ModuleExpression
SubtractExpression
```



Stack

Expression Evaluation: Interpretation

- Roslyn doesn’t know what exactly is certain identifier

Expression Evaluation: Interpretation

- Roslyn doesn’t know what exactly is certain identifier

```
InvocationExpression --- foo(boo(), a.b.c)
    IdentifierName --- foo
    …
```

Expression Evaluation: Interpretation

- Roslyn doesn’t know what exactly is certain identifier
- Even simple operations may be function calls

```
InvocationExpression --- foo(boo(), a.b.c)
    IdentifierName --- foo
    …
```

Expression Evaluation: Interpretation

- Roslyn doesn’t know what exactly is certain identifier
- Even simple operations may be function calls
- ICorDebug API is responsible for interaction with debuggee

```
InvocationExpression --- foo(boo(), a.b.c)
    IdentifierName --- foo
    …
```

Expression Evaluation: Interpretation

- Roslyn doesn’t know what exactly is certain identifier
- Even simple operations may be function calls
- ICorDebug API is responsible for interaction with debuggee
- Not all language features are supported yet (e.g. lambdas)

```
InvocationExpression --- foo(boo(), a.b.c)
    IdentifierName --- foo
    …
```

Expression Evaluation: Arithmetic Eval

- Evaluation in debugger process for all operations over primitive types
- Manual complex type casting not needed

Expression Evaluation: Arithmetic Eval

- Evaluation in debugger process for all operations over primitive types
- Manual complex type casting not needed

Example of arithmetic addition operation:

```
private static object AddExpression(dynamic first, dynamic second)
{
    return first + second;
}
```

How to use NetCoreDbg (CLI)

How to use NetCoreDbg (CLI)

- crossgen2 is new AOT compiler (since .NET 5)
- System.Private.CoreLib.dll is the main library

```
user@linux$ ./netcoredbg \
--interpreter=cli \
-- \
/home/runtime/overlay/corerun \
/home/runtime/overlay/crossgen2/crossgen2.dll -r:/home/runtime/overlay/*.dll \
-O \
-o:/home/runtime/overlay/System.Private.CoreLib.ni.dll \
/home/runtime/overlay/System.Private.CoreLib.dll
```

How to use NetCoreDbg (CLI)

- crossgen2 is new AOT compiler (since .NET 5)
- System.Private.CoreLib.dll is the main library

```
user@linux$ ./netcoredbg \
--interpreter=cli \
-- \
/home/runtime/overlay/corerun \
/home/runtime/overlay/crossgen2/crossgen2.dll -r:/home/runtime/overlay/*.dll \
-O \
-o:/home/runtime/overlay/System.Private.CoreLib.ni.dll \
/home/runtime/overlay/System.Private.CoreLib.dll
```

How to use NetCoreDbg (CLI)

- crossgen2 is new AOT compiler (since .NET 5)
- System.Private.CoreLib.dll is the main library

```
user@linux$ ./netcoredbg \
--interpreter=cli \
-- \
/home/runtime/overlay/corerun \
/home/runtime/overlay/crossgen2/crossgen2.dll -r:/home/runtime/overlay/*.dll \
-O \
-o:/home/runtime/overlay/System.Private.CoreLib.ni.dll \
/home/runtime/overlay/System.Private.CoreLib.dll
```

How to use NetCoreDbg (CLI)

- crossgen2 is new AOT compiler (since .NET 5)
- System.Private.CoreLib.dll is the main library

```
user@linux$ ./netcoredbg \
--interpreter=cli \
-- \
/home/runtime/overlay/corerun \
/home/runtime/overlay/crossgen2/crossgen2.dll -r:/home/runtime/overlay/*.dll \
-O \
-o:/home/runtime/overlay/System.Private.CoreLib.ni.dll \
/home/runtime/overlay/System.Private.CoreLib.dll
```

How to use NetCoreDbg (CLI)

Set breakpoint at Main and show backtrace

```
ncdb> b Main
 Breakpoint 1 at Main() --pending, warning: No executable code of the debugger's
target code type is associated with this line.
ncdb> r
^running

…

stopped, reason: breakpoint 1 hit, thread id: 27750, stopped threads: all,
times= 0, frame={ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
}
ncdb> bt
#0 ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
```

How to use NetCoreDbg (CLI)

Set breakpoint at Main and show backtrace

```
ncdb> b Main
 Breakpoint 1 at Main() --pending, warning: No executable code of the debugger's
target code type is associated with this line.
ncdb> r
^running

…

stopped, reason: breakpoint 1 hit, thread id: 27750, stopped threads: all,
times= 0, frame={ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
}
ncdb> bt
#0 ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
```

How to use NetCoreDbg (CLI)

Set breakpoint at Main and show backtrace

```
ncdb> b Main
 Breakpoint 1 at Main() --pending, warning: No executable code of the debugger's
target code type is associated with this line.
ncdb> r
^running

…

stopped, reason: breakpoint 1 hit, thread id: 27750, stopped threads: all,
times= 0, frame={ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
}
ncdb> bt
#0 ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
```

How to use NetCoreDbg (CLI)

Stop on Ctrl+C

```
ncdb> ^C
stopped, reason: interrupted, thread id: 27750, stopped threads: all,
frame={ILCompiler.Compilation..ctor() at
/home/runtime/src/coreclr/tools/aot/ILCompiler.ReadyToRun/Compiler/ReadyToRunCod
egenCompilation.cs:64
}
```

How to use NetCoreDbg (CLI)

Stop on Ctrl+C

```
ncdb> ^C
stopped, reason: interrupted, thread id: 27750, stopped threads: all,
frame={ILCompiler.Compilation..ctor() at
/home/runtime/src/coreclr/tools/aot/ILCompiler.ReadyToRun/Compiler/ReadyToRunCod
egenCompilation.cs:64
}
```

How to use NetCoreDbg (CLI)

Show backtrace

```
ncdb> bt
#1 ILCompiler.Compilation..ctor() at
/home/runtime/src/coreclr/tools/aot/ILCompiler.ReadyToRun/Compiler/ReadyToRunCod
egenCompilation.cs:64
#2 ILCompiler.ReadyToRunCodegenCompilation..ctor() at
/home/runtime/src/coreclr/tools/aot/ILCompiler.ReadyToRun/Compiler/ReadyToRunCod
egenCompilation.cs:304
#3 ILCompiler.ReadyToRunCodegenCompilationBuilder.ToCompilation() at
/home/runtime/src/coreclr/tools/aot/ILCompiler.ReadyToRun/Compiler/ReadyToRunCod
egenCompilationBuilder.cs:297
#4 ILCompiler.Program.RunSingleCompilation() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:713
#5 ILCompiler.Program.Run() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:509
#6 ILCompiler.Program.Main() at
/home/runtime/src/coreclr/tools/aot/crossgen2/Program.cs:1002
```

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines
- **`step`** to step-over

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines
- **`step`** to step-over
- **`next`** to step-into

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines
- **`step`** to step-over
- **`next`** to step-into
- **`print`** to perform expression evaluation

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines
- **`step`** to step-over
- **`next`** to step-into
- **`print`** to perform expression evaluation
- **`attach`** to attach to process

How to use NetCoreDbg (CLI)

- **`catch`** to set exception breakpoints
- **`list`** to list source code lines
- **`step`** to step-over
- **`next`** to step-into
- **`print`** to perform expression evaluation
- **`attach`** to attach to process
- etc.

Interop Debugging

.NET Interop Debugging (Microsoft)

- Mixed mode, interop, hybrid debugger

.NET Interop Debugging (Microsoft)

- Mixed mode, interop, hybrid debugger
- Debug both managed and native DLLs simultaneously in the same debugging session

.NET Interop Debugging (Microsoft)

- Mixed mode, interop, hybrid debugger
- Debug both managed and native DLLs simultaneously in the same debugging session
- Currently available only on Windows x86/x64/arm64 (Visual Studio)

.NET Interop Debugging (Microsoft)

- Mixed mode, interop, hybrid debugger
- Debug both managed and native DLLs simultaneously in the same debugging session
- Currently available only on Windows x86/x64/arm64 (Visual Studio)
- Increase developers productivity

Example: Android

```
coral:/ # debuggerd -b $(pidof com.example.logevents)


----- pid 4483 at 2022-11-07 12:57:41.523083660+0000 -----
Cmd line: com.example.logevents
ABI: 'arm64'

"ample.logevents" sysTid=4483
    #00 pc 0000000000009e598  /apex/com.android.runtime/lib64/bionic/libc.so (__epoll_pwait+8) (BuildId: ba489d4985c0cf173209da67405662f9)
    #01 pc 0000000000016640  /system/lib64/libutils.so (android::Looper::pollInner(int)+180) (BuildId: a3acb0eba7fd91ea48db6f0befa41c65)
    #02 pc 0000000000016524  /system/lib64/libutils.so (android::Looper::pollOnce(int, int*, int*, void**)+112) (BuildId:
a3acb0eba7fd91ea48db6f0befa41c65)
    #03 pc 000000000014acf8  /system/lib64/libandroid_runtime.so (android::android_os_MessageQueue_nativePollOnce(_JNIEnv*, _jobject*, long, int)+44)
(BuildId: 072ee58703f191b8e57c0025d9adae9b)
    #04 pc 000000000019d22c  /system/framework/arm64/boot-framework.oat (art_jni_trampoline+108) (BuildId: f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #05 pc 00000000004f6db8  /system/framework/arm64/boot-framework.oat (android.os.MessageQueue.next+232) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #06 pc 00000000004f4404  /system/framework/arm64/boot-framework.oat (android.os.Looper.loopOnce+100) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #07 pc 00000000004f4304  /system/framework/arm64/boot-framework.oat (android.os.Looper.loop+516) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #08 pc 00000000002d038c  /system/framework/arm64/boot-framework.oat (android.app.ActivityThread.main+732) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #09 pc 0000000000218be8  /apex/com.android.art/lib64/libart.so (art_quick_invoke_static_stub+568) (BuildId: 556e634687c8250966f28a53a5a17b2b)
    #10 pc 0000000000284224  /apex/com.android.art/lib64/libart.so (art::ArtMethod::Invoke(art::Thread*, unsigned int*, unsigned int, art::JValue*,
char const*)+216) (BuildId: 556e634687c8250966f28a53a5a17b2b)
    #11 pc 000000000061e234  /apex/com.android.art/lib64/libart.so (_jobject*
art::InvokeMethod<(art::PointerSize)8>(art::ScopedObjectAccessAlreadyRunnable const&, _jobject*, _jobject*, _jobject*, unsigned long)+1384) (BuildId:
556e634687c8250966f28a53a5a17b2b)
    #12 pc 000000000058f780  /apex/com.android.art/lib64/libart.so (art::Method_invoke(_JNIEnv*, _jobject*, _jobject*, _jobjectArray*)+52) (BuildId:
556e634687c8250966f28a53a5a17b2b)
    #13 pc 00000000000b2f74  /apex/com.android.art/javalib/arm64/boot.oat (art_jni_trampoline+132) (BuildId: ab2bf4ec264efdb6c452a238be38fe624de826b8)
    #14 pc 000000000081cb2c  /system/framework/arm64/boot-framework.oat (com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run+140) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    #15 pc 0000000000824d98  /system/framework/arm64/boot-framework.oat (com.android.internal.os.ZygoteInit.main+2232) (BuildId:
f578ed4f6592643e1c6d84081ab852ea2db681bb)
    ...
```

.NET Interop Debugging Wishlist

- Linux support

.NET Interop Debugging Wishlist

- Linux support
- See real (mixed) stack trace

.NET Interop Debugging Wishlist

- Linux support
- See real (mixed) stack trace
- Set breakpoints in native and managed code

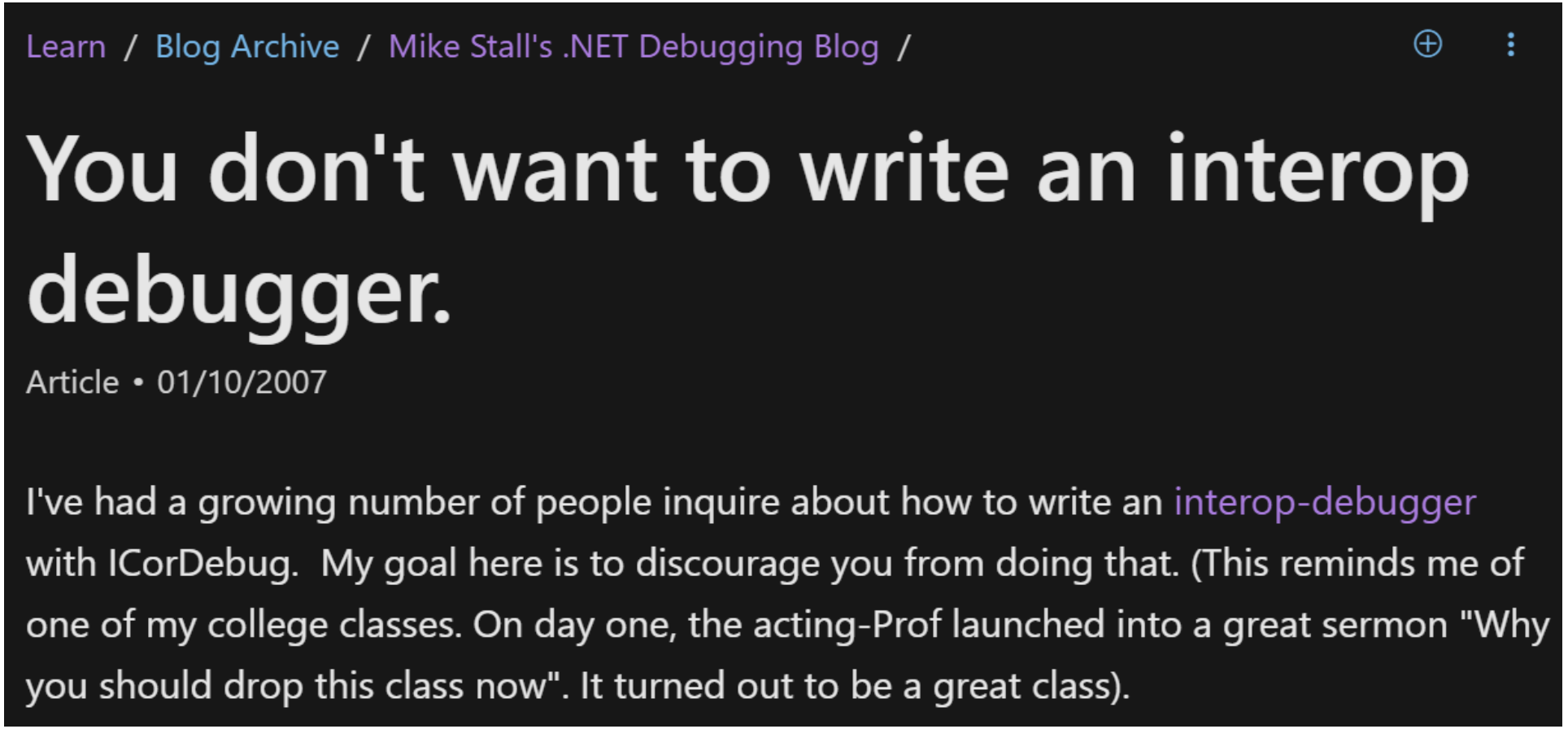
.NET Interop Debugging Wishlist

- Linux support
- See real (mixed) stack trace
- Set breakpoints in native and managed code
- Perform native and managed evaluation

.NET Interop Debugging Wishlist

- Linux support
- See real (mixed) stack trace
- Set breakpoints in native and managed code
- Perform native and managed evaluation
- Perform native and managed stepping

Interop Debugging: Challenges



Interop Debugging: Challenges

- Have to deal with user and system threads

Interop Debugging: Challenges

- Have to deal with user and system threads
- Possible dead locks

Interop Debugging: Challenges

- Have to deal with user and system threads
- Possible dead locks
- Various versions of DWARF format

Interop Debugging: Challenges

- Have to deal with user and system threads
- Possible dead locks
- Various versions of DWARF format
- CoreCLR uses native breakpoints and signals internally and should handle them itself

Interop Debugging: Challenges

- Have to deal with user and system threads
- Possible dead locks
- Various versions of DWARF format
- CoreCLR uses native breakpoints and signals internally and should handle them itself
- Need to make full implementation of stepping for every supported CPU arch

Interop Debugging: Challenges

- Have to deal with user and system threads
- Possible dead locks
- Various versions of DWARF format
- CoreCLR uses native breakpoints and signals internally and should handle them itself
- Need to make full implementation of stepping for every supported CPU arch
- Unclear changes in runtime and debugging API

Interop Debugging: Approaches

- Two debuggers: native and managed

Interop Debugging: Approaches

- Two debuggers: native and managed
- Self debugging

Interop Debugging: Approaches

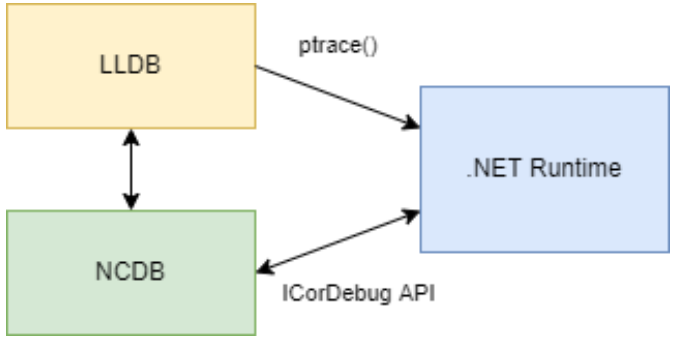
- Two debuggers: native and managed
- Self debugging
- Native debugging in managed debugger

Interop Debugging: Approaches

- Two debuggers: native and managed
- Self debugging
- Native debugging in managed debugger (or vice versa)

Interop Debugging: Approach #1

Use two debuggers: native and managed

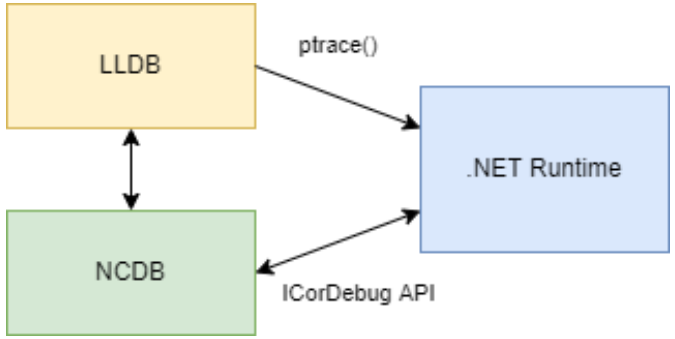


Brief design

- Use GDB/LLDB for debugging native code

Interop Debugging: Approach #1

Use two debuggers: native and managed

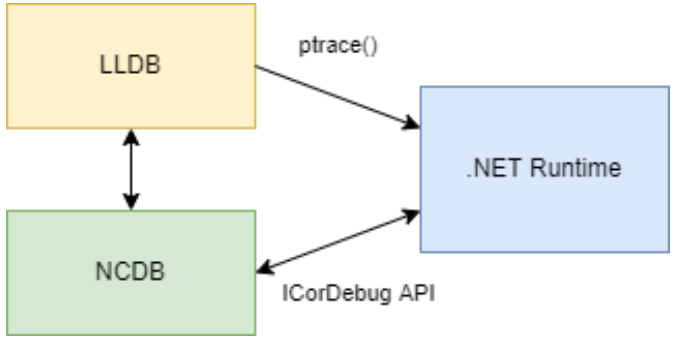


Brief design

- Use GDB/LLDB for debugging native code
- Use NetCoreDbg for debugging managed code

Interop Debugging: Approach #1

Use two debuggers: native and managed



Brief design

- Use GDB/LLDB for debugging native code
- Use NetCoreDbg for debugging managed code
- Transfer control over debuggee process between debuggers

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory
- Synchronization of 2 debuggers

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory
- Synchronization of 2 debuggers
- It’s required to not suspend system threads

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory
- Synchronization of 2 debuggers
- It’s required to not suspend system threads
- Merging native and managed stack traces

Interop Debugging: Approach #1

Pros:

- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory
- Synchronization of 2 debuggers
- It’s required to not suspend system threads
- Merging native and managed stack traces
- Patching of native debugger

Interop Debugging: Approach #1

Pros:

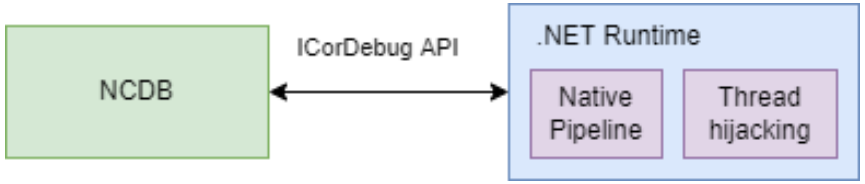
- Use of standard debuggers may simplify work with native code

Cons:

- Consumes a lot of memory
- Synchronization of 2 debuggers
- It’s required to not suspend system threads
- Merging native and managed stack traces
- Patching of native debugger

Interop Debugging: Approach #2

Use self debugging

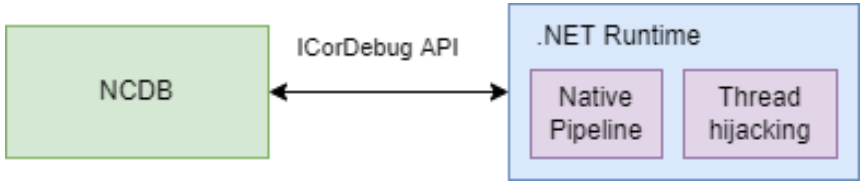


Brief design

- Use runtime self debugging for native code inside debuggee process similar to managed code

Interop Debugging: Approach #2

Use self debugging

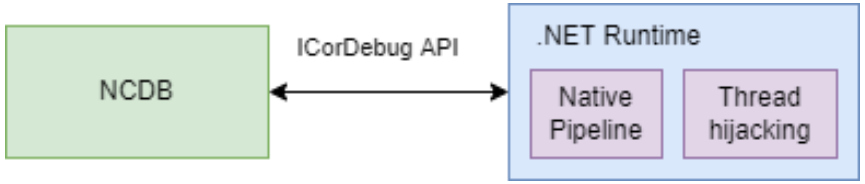


Brief design

- Use runtime self debugging for native code inside debuggee process similar to managed code
- Use ICorDebug API in NetCoreDbg for communication with debuggee process

Interop Debugging: Approach #2

Use self debugging



Brief design

- Use runtime self debugging for native code inside debuggee process similar to managed code
- Use ICorDebug API in NetCoreDbg for communication with debuggee process
- Implement Windows like debugging API in .NET Runtime for Linux

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows
- No issues with synchronization between two debugger processes

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows
- No issues with synchronization between two debugger processes
- All needed information from the debuggee process could be easily provided to debugger

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows
- No issues with synchronization between two debugger processes
- All needed information from the debuggee process could be easily provided to debugger

Cons:

- Windows implementation also has limitations

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows
- No issues with synchronization between two debugger processes
- All needed information from the debuggee process could be easily provided to debugger

Cons:

- Windows implementation also has limitations
- Impossible to self debug runtime and low level libraries

Interop Debugging: Approach #2

Pros:

- Similar to what is done for Windows
- No issues with synchronization between two debugger processes
- All needed information from the debuggee process could be easily provided to debugger

Cons:

- Windows implementation also has limitations
- Impossible to self debug runtime and low level libraries
- Won’t work with older .NET

Interop Debugging: Approach #2

Pros:

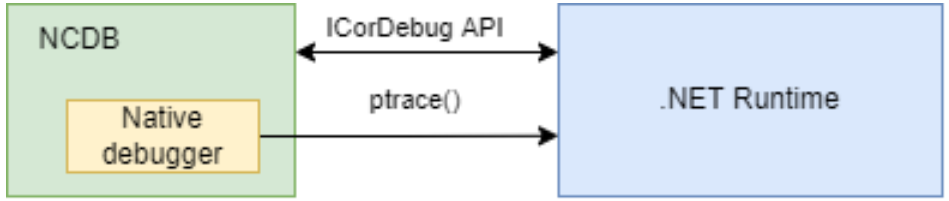
- Similar to what is done for Windows
- No issues with synchronization between two debugger processes
- All needed information from the debuggee process could be easily provided to debugger

Cons:

- Windows implementation also has limitations
- Impossible to self debug runtime and low level libraries
- Won’t work with older .NET

Interop Debugging: Approach #3

Native debugging in managed debugger

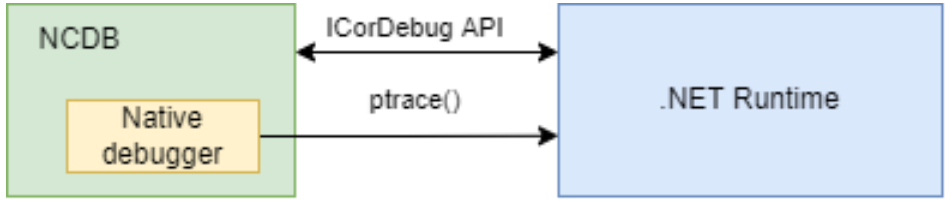


Brief design

- Use `ptrace()` syscall for native debugging in NetCoreDbg

Interop Debugging: Approach #3

Native debugging in managed debugger

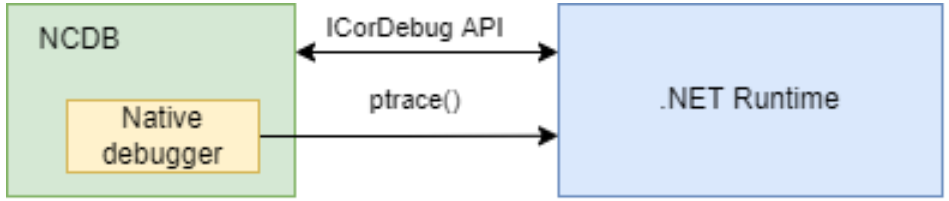


Brief design

- Use `ptrace()` syscall for native debugging in NetCoreDbg
- Use libs to work with DWARF and stack unwinding

Interop Debugging: Approach #3

Native debugging in managed debugger



Brief design

- Use `ptrace()` syscall for native debugging in NetCoreDbg
- Use libs to work with DWARF and stack unwinding
- Extend debugging API if needed

Interop Debugging: Approach #3

Pros:

- Full control over code

Interop Debugging: Approach #3

Pros:

- Full control over code
- No dramatic changes in .NET Runtime

Interop Debugging: Approach #3

Pros:

- Full control over code
- No dramatic changes in .NET Runtime

Cons:

- Still need to synchronize native and managed parts of debugger

Interop Debugging: Approach #3

Pros:

- Full control over code
- No dramatic changes in .NET Runtime

Cons:

- Still need to synchronize native and managed parts of debugger
- Impossible to self debug runtime and low level libraries

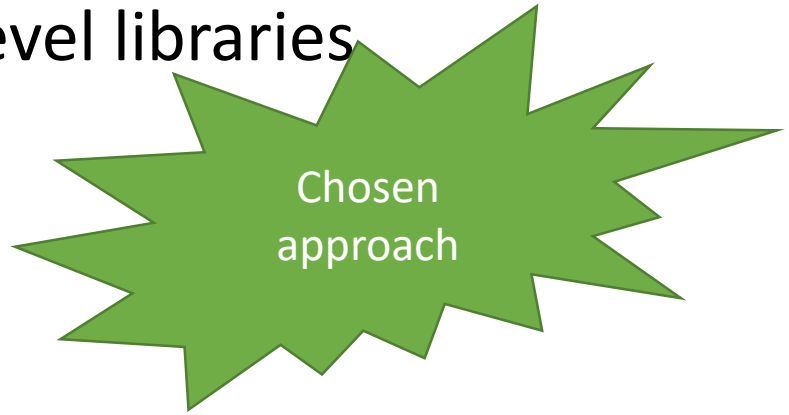
Interop Debugging: Approach #3

Pros:

- Full control over code
- No dramatic changes in .NET Runtime

Cons:

- Still need to synchronize native and managed parts of debugger
- Impossible to self debug runtime and low level libraries



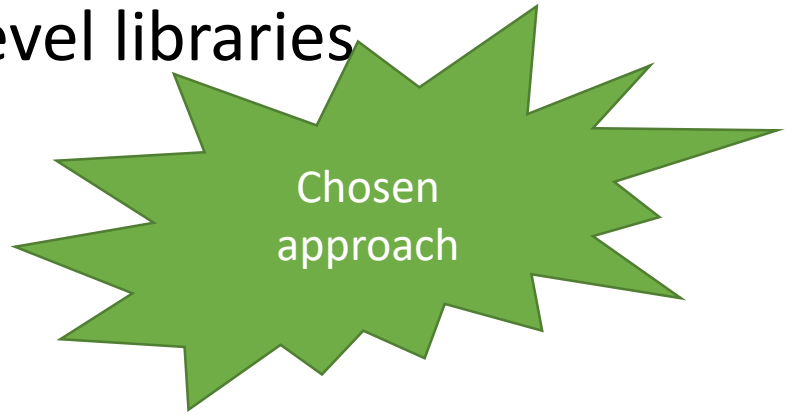
Interop Debugging: Approach #3

Pros:

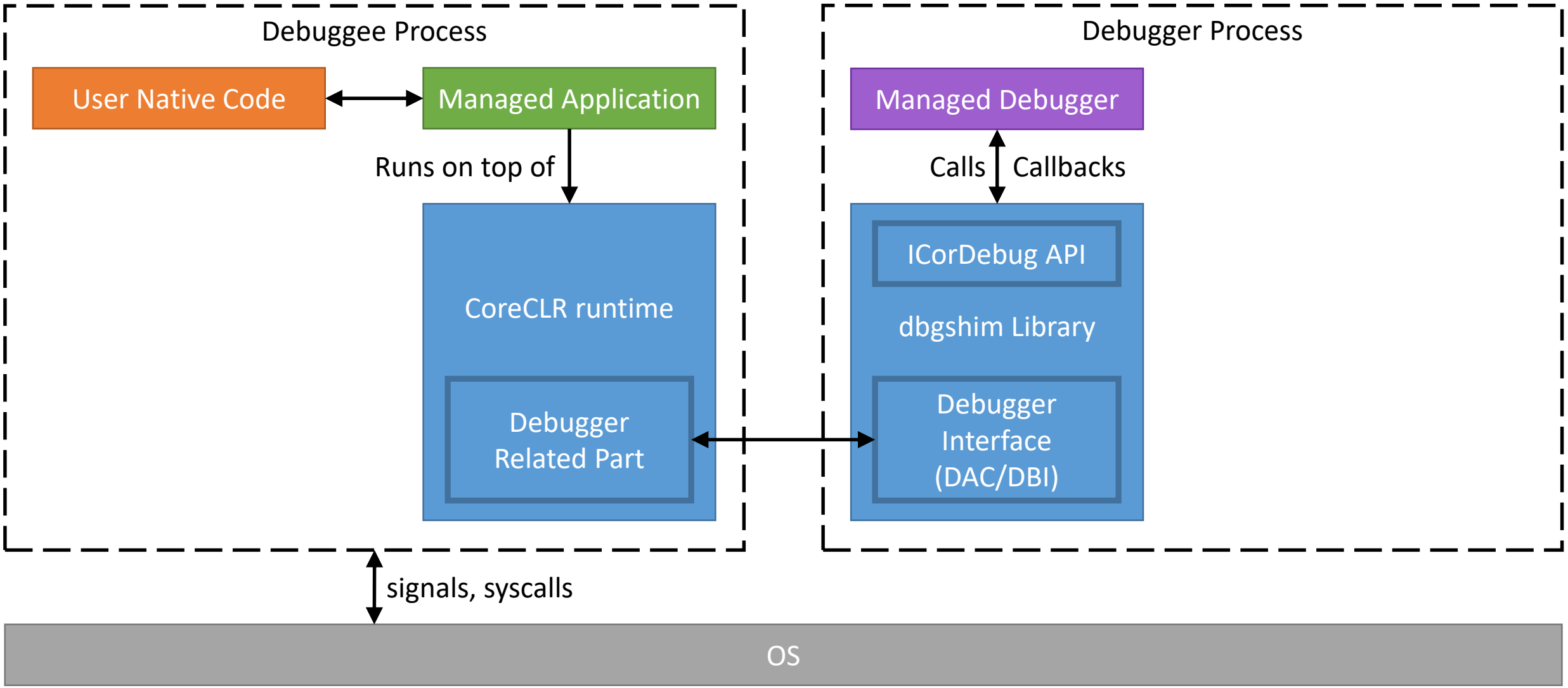
- Full control over code
- No dramatic changes in .NET Runtime (**currently no changes in runtime needed => works even on .NET Core 2.1**)

Cons:

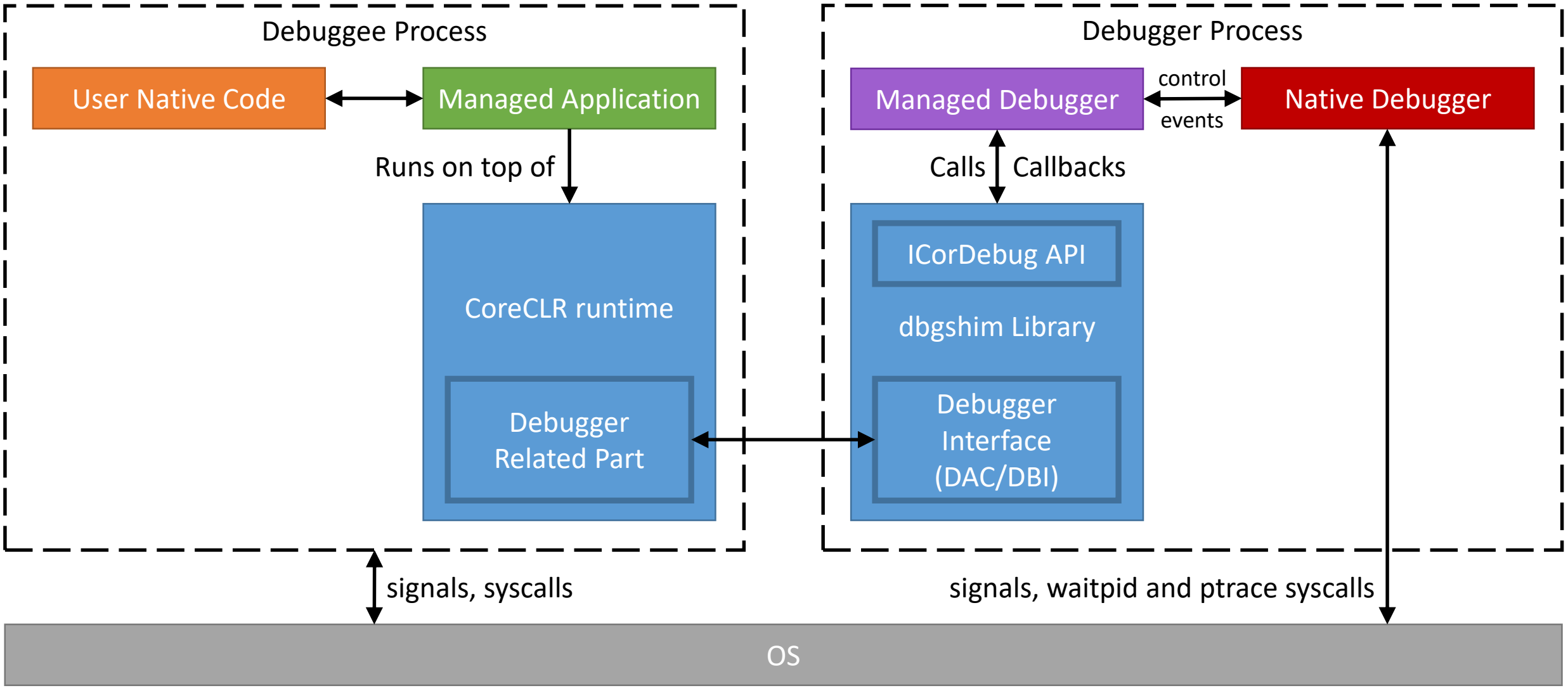
- Still need to synchronize native and managed parts of debugger
- Impossible to self debug runtime and low level libraries



Interop Debugging: Architecture



Interop Debugging: Architecture



Interop Debugging: How to set breakpoint

- x86/x64: `INT3` instruction (0xcc, 1 byte)
- arm64: `brk #0` instruction (0xd4200000, 4 bytes)
- arm: illegal instruction (0x07f001f0, 4 bytes)
- arm (thumb2): illegal instruction (0xde01, 2 bytes)

Interop Debugging: How to set breakpoint

- x86/x64: `INT3` instruction (0xcc, 1 byte)
  (program counter register is changed)
- arm64: `brk #0` instruction (0xd4200000, 4 bytes)
  (program counter register is not changed)
- arm: illegal instruction (0x07f001f0, 4 bytes)
  (program counter register is not changed)
- arm (thumb2): illegal instruction (0xde01, 2 bytes)
  (program counter register is not changed)

Interop Debugging: How to set breakpoint

1. stop threads

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory
3. encode breakpoint

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory
3. encode breakpoint
4. `ptrace(PTRACE_POKEDATA,…)` to write memory

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory
3. encode breakpoint
4. `ptrace(PTRACE_POKEDATA,…)` to write memory
5. save original bytes

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory
3. encode breakpoint
4. `ptrace(PTRACE_POKEDATA,…)` to write memory
5. save original bytes
6. start threads

Interop Debugging: How to set breakpoint

1. stop threads
2. `ptrace(PTRACE_PEEKDATA,…)` to read other process memory
3. encode breakpoint
4. `ptrace(PTRACE_POKEDATA,…)` to write memory
5. save original bytes
6. start threads

PEEKDATA and POKEDATA read/write machine word, while instruction might be smaller.

Interop Debugging: How to step over

1. get to the start of instruction

Interop Debugging: How to step over

1. get to the start of instruction
2. restore original instruction

Interop Debugging: How to step over

1. get to the start of instruction
2. restore original instruction
3. execute one instruction (single step)

Interop Debugging: How to step over

1. get to the start of instruction
2. restore original instruction
3. execute one instruction (single step)
4. restore breakpoint

Interop Debugging: How to track native libs

- by setting rendezvous breakpoint

Interop Debugging: How to track native libs

- by setting rendezvous breakpoint
- rendezvous structure (`r_debug`) contains:
  `r_brk`, `r_state`, `r_map`

Interop Debugging: How to track native libs

- by setting rendezvous breakpoint
- rendezvous structure (`r_debug`) contains:
  `r_brk`, `r_state`, `r_map`
- rendezvous breakpoint is hit two consecutive times

Interop Debugging: How to track native libs

- by setting rendezvous breakpoint
- rendezvous structure (`r_debug`) contains:
  `r_brk`, `r_state`, `r_map`
- rendezvous breakpoint is hit two consecutive times
- unresolved breakpoints are checked on new library load

Interop Debugging: What works?

- All managed functions of NetCoreDbg

Interop Debugging: What works?

- All managed functions of NetCoreDbg
- Attach to process and launch of application from scratch

Interop Debugging: What works?

- All managed functions of NetCoreDbg
- Attach to process and launch of application from scratch
- Track of load/unload of native libraries

Interop Debugging: What works?

- All managed functions of NetCoreDbg
- Attach to process and launch of application from scratch
- Track of load/unload of native libraries
- Handling of debug information for native libraries

Interop Debugging: What works?

- All managed functions of NetCoreDbg
- Attach to process and launch of application from scratch
- Track of load/unload of native libraries
- Handling of debug information for native libraries
- Native breakpoints for x86, x64, arm, arm64

Interop Debugging: What works?

- All managed functions of NetCoreDbg
- Attach to process and launch of application from scratch
- Track of load/unload of native libraries
- Handling of debug information for native libraries
- Native breakpoints for x86, x64, arm, arm64
- Stack unwind for x86, x64, arm, arm64 (managed, native, mixed)

How to use Interop NetCoreDbg (CLI)

```
 8        public delegate void FPtr(IntPtr ptr, int size);
 9
10        [DllImport("liballocator.so", CallingConvention = CallingConvention.Cdecl)]
          1 reference
11        internal static extern IntPtr alloc(int size, FPtr callback);
          1 reference
12        private static void reportCB(IntPtr ptr, int size)
13        {
14            Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15        }
          0 references
16        static void Main(string[] args)
17        {
18            Console.WriteLine("Starting!");
19            FPtr cb = new FPtr(reportCB);
20            for (int i = 0; i < 10; ++i)
21            {
22                IntPtr value = alloc(i * 17 + 123, cb);
23            }
24        }
```

How to use Interop NetCoreDbg (CLI)

```
1   #include <malloc.h>
2
3   typedef void* FPtr(void*, int);
4
5   extern "C"
6   {
7       void* alloc(int size, FPtr callback)
8       {
9           void* mem = malloc(size);
10          (*callback)(mem, size);
11          return mem;
12      }
13  }
```

How to use Interop NetCoreDbg (CLI)

- Run on x64

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6
- `<EmbedAllSources>true</EmbedAllSources>`

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6
- `<EmbedAllSources>true</EmbedAllSources>`
- Build of managed part on Linux with:
  `dotnet build -c Debug`

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6
- `<EmbedAllSources>true</EmbedAllSources>`
- Build of managed part on Linux with:
  `dotnet build -c Debug`
- Build of native part on Linux with:
  `g++ -fPIC -shared -g allocator.cpp -o liballocator.so`

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6
- `<EmbedAllSources>true</EmbedAllSources>`
- Build of managed part on Linux with:
  `dotnet build -c Debug`
- Build of native part on Linux with:
  `g++ -fPIC -shared -g allocator.cpp -o liballocator.so`
- `liballocator.so` contains debug info

How to use Interop NetCoreDbg (CLI)

- Run on x64
- Use .NET 6
- `<EmbedAllSources>true</EmbedAllSources>`
- Build of managed part on Linux with:
  `dotnet build -c Debug`
- Build of native part on Linux with:
  `g++ -fPIC -shared -g allocator.cpp -o liballocator.so`
- `liballocator.so` contains debug info
- `-DINTEROP_DEBUGGING=1` build option is needed for interop

How to use Interop NetCoreDbg (CLI)

Launch without debugger

```
user@linux:~/demo/managed/demo$ dotnet run
Starting!
Reporting 94713499723280, size 123
Reporting 94713499855696, size 140
Reporting 94713499855856, size 157
Reporting 94713499684176, size 174
Reporting 94713499611840, size 191
Reporting 94713498804992, size 208
Reporting 94713499669520, size 225
Reporting 94713498763248, size 242
Reporting 94713499687792, size 259
Reporting 94713499389456, size 276
user@linux:~/demo/managed/demo$
```

How to use Interop NetCoreDbg (CLI)

Launch with managed NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f5689d10000, size: 10601472(0xa1c400)

thread created, id: 9977
```

How to use Interop NetCoreDbg (CLI)

Launch with managed NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f5689d10000, size: 10601472(0xa1c400)

thread created, id: 9977
```

How to use Interop NetCoreDbg (CLI)

Launch with managed NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f5689d10000, size: 10601472(0xa1c400)

thread created, id: 9977
```

How to use Interop NetCoreDbg (CLI)

Launch with managed NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f5689d10000, size: 10601472(0xa1c400)

thread created, id: 9977
```

How to use Interop NetCoreDbg (CLI)

Launch with managed NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f5689d10000, size: 10601472(0xa1c400)

thread created, id: 9977
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
symbols loaded, base address: 0x7f57052b8000, size: 5632(0x1600)
breakpoint modified,  Breakpoint 1 at reportCB()

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Runtime.dll
no symbols loaded, base address: 0x7f5700089000, size: 32256(0x7e00)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Console.dll
no symbols loaded, base address: 0x7f568a890000, size: 376832(0x5c000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Threading.dll
no symbols loaded, base address: 0x7f568a900000, size: 266752(0x41200)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/Microsoft.Win32.Primitives.dll
no symbols loaded, base address: 0x7f568a970000, size: 210944(0x33800)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
symbols loaded, base address: 0x7f57052b8000, size: 5632(0x1600)
breakpoint modified,  Breakpoint 1 at reportCB()

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Runtime.dll
no symbols loaded, base address: 0x7f5700089000, size: 32256(0x7e00)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Console.dll
no symbols loaded, base address: 0x7f568a890000, size: 376832(0x5c000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Threading.dll
no symbols loaded, base address: 0x7f568a900000, size: 266752(0x41200)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/Microsoft.Win32.Primitives.dll
no symbols loaded, base address: 0x7f568a970000, size: 210944(0x33800)
```

How to use Interop NetCoreDbg (CLI)

```
Starting!

stopped, reason: breakpoint 1 hit, thread id: 9977, stopped threads: all, times=
0, frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:13}
ncdb> bt
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f568a7baa9c [Native Frames]
#2: 0x00007f568a7ba9ad [Native Frames]
#3: 0x00007f568a7b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22
```

```
12      private static void reportCB(IntPtr ptr, int size)
13      {
14          Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15      }
```

```
20          for (int i = 0; i < 10; ++i)
21          {
22              IntPtr value = alloc(i * 17 + 123, cb);
23          }
```

How to use Interop NetCoreDbg (CLI)

```
Starting!

stopped, reason: breakpoint 1 hit, thread id: 9977, stopped threads: all, times=
0, frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:13}
ncdb> bt
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f568a7baa9c [Native Frames]
#2: 0x00007f568a7ba9ad [Native Frames]
#3: 0x00007f568a7b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22
```

```
12          private static void reportCB(IntPtr ptr, int size)
13          {
14              Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15          }
```

```
20          for (int i = 0; i < 10; ++i)
21          {
22              IntPtr value = alloc(i * 17 + 123, cb);
23          }
```

How to use Interop NetCoreDbg (CLI)

```
Starting!

stopped, reason: breakpoint 1 hit, thread id: 9977, stopped threads: all, times=
0, frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:13}
ncdb> bt
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f568a7baa9c [Native Frames]
#2: 0x00007f568a7ba9ad [Native Frames]
#3: 0x00007f568a7b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22
```

```
12      private static void reportCB(IntPtr ptr, int size)
13      {
14          Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15      }
```

```
20          for (int i = 0; i < 10; ++i)
21          {
22              IntPtr value = alloc(i * 17 + 123, cb);
23          }
```

How to use Interop NetCoreDbg (CLI)

```
Starting!

stopped, reason: breakpoint 1 hit, thread id: 9977, stopped threads: all, times=
0, frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:13}
ncdb> bt
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f568a7baa9c [Native Frames]
#2: 0x00007f568a7ba9ad [Native Frames]
#3: 0x00007f568a7b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22
```

```
12          private static void reportCB(IntPtr ptr, int size)
13          {
14              Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15          }
```

```
20          for (int i = 0; i < 10; ++i)
21          {
22              IntPtr value = alloc(i * 17 + 123, cb);
23          }
```

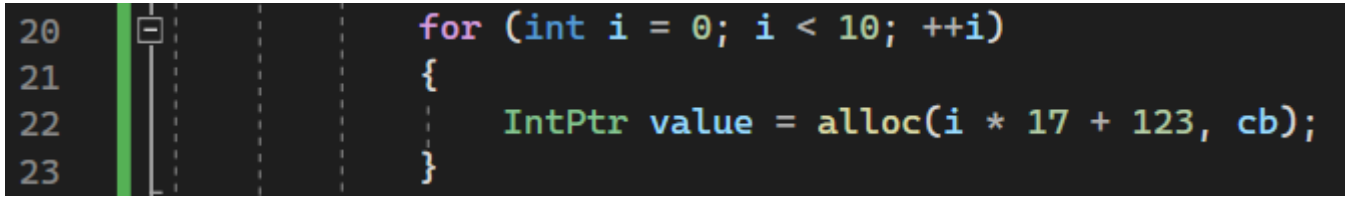
How to use Interop NetCoreDbg (CLI)

```
ncdb> p ptr
ptr = -1986462080
ncdb> p size
size = 123
```

```
12      private static void reportCB(IntPtr ptr, int size)
13      {
14          Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15      }
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> frame 3
#3: 0x00007f568a7b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22
ncdb> p i
i = 0
ncdb> p i*17+123
i*17+123 = 123
ncdb> p cb
cb = {demo.Program.FPtr}: {_invocationList = null, _invocationCount = 0, _target
= {demo.Program.FPtr}, _methodBase = null, _methodPtr = -1983459264,
_methodPtrAux = -1971653744, Target = null, Method =
{System.Reflection.RuntimeMethodInfo}}
```



How to use Interop NetCoreDbg (CLI)

```
ncdb> frame 0
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13

ncdb> step
^running

stopped, reason: end stepping range, thread id: 9977, stopped threads: all,
frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:14}
ncdb> step
^running
Reporting 94392804769408, size 123

stopped, reason: end stepping range, thread id: 9977, stopped threads: all,
frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:15}
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> frame 0
#0: 0x00007f568a7bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13

ncdb> step
^running

stopped, reason: end stepping range, thread id: 9977, stopped threads: all,
frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:14}
ncdb> step
^running
Reporting 94392804769408, size 123

stopped, reason: end stepping range, thread id: 9977, stopped threads: all,
frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:15}
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> c
^running

stopped, reason: breakpoint 1 hit, thread id: 9977, stopped threads: all, times=
1, frame={demo.Program.reportCB() at /home/user/demo/managed/demo/Program.cs:13}
ncdb> p size
size = 140
```

```
12      private static void reportCB(IntPtr ptr, int size)
13      {
14          Console.WriteLine("Reporting " + ptr.ToString() + ", size " + size.ToString());
15      }
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> list
   8            public delegate void FPtr(IntPtr ptr, int size);
   9
   10           [DllImport("liballocator.so", CallingConvention =
CallingConvention.Cdecl)]
   11           internal static extern IntPtr alloc(int size, FPtr callback);
   12           private static void reportCB(IntPtr ptr, int size)
 > 13           {
   14               Console.WriteLine("Reporting " + ptr.ToString() + ", size " +
size.ToString());
   15           }
   16           static void Main(string[] args)
   17           {
```

How to use Interop NetCoreDbg (CLI)

Launch with interop NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--interop-debugging
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

native thread created, id: 12894

native thread created, id: 12896
```

How to use Interop NetCoreDbg (CLI)

Launch with interop NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--interop-debugging
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

native thread created, id: 12894

native thread created, id: 12896
```

How to use Interop NetCoreDbg (CLI)

Launch with interop NetCoreDbg

```
user@linux$ ./netcoredbg
--interpreter=cli
--interop-debugging
--
/home/user/demo/dotnet-sdk-6.0/dotnet
/home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
ncdb> b reportCB
 Breakpoint 1 at reportCB() --pending, warning: No executable code of the
debugger's target code type is associated with this line.
ncdb> r
^running

native thread created, id: 12894

native thread created, id: 12896
```

How to use Interop NetCoreDbg (CLI)

```
native thread created, id: 12897

native thread created, id: 12898

native thread created, id: 12899

native thread created, id: 12900

native thread created, id: 12901

native thread created, id: 12906

library loaded: /lib/x86_64-linux-gnu/libpthread-2.27.so
no symbols loaded, base address: 0x7f708f086000, size: 2207744(0x21b000)

library loaded: /lib/x86_64-linux-gnu/libdl-2.27.so
no symbols loaded, base address: 0x7f708ee82000, size: 2113536(0x204000)
```

How to use Interop NetCoreDbg (CLI)

```
native thread created, id: 12897

native thread created, id: 12898

native thread created, id: 12899

native thread created, id: 12900

native thread created, id: 12901

native thread created, id: 12906

library loaded: /lib/x86_64-linux-gnu/libpthread-2.27.so
no symbols loaded, base address: 0x7f708f086000, size: 2207744(0x21b000)

library loaded: /lib/x86_64-linux-gnu/libdl-2.27.so
no symbols loaded, base address: 0x7f708ee82000, size: 2113536(0x204000)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25
no symbols loaded, base address: 0x7f708eaf9000, size: 3690496(0x385000)

library loaded: /lib/x86_64-linux-gnu/libm-2.27.so
no symbols loaded, base address: 0x7f708e75b000, size: 3792896(0x39e000)

library loaded: /lib/x86_64-linux-gnu/libgcc_s.so.1
no symbols loaded, base address: 0x7f708e543000, size: 2195456(0x218000)

library loaded: /lib/x86_64-linux-gnu/libc-2.27.so
no symbols loaded, base address: 0x7f708e152000, size: 4116480(0x3ed000)

library loaded: /lib/x86_64-linux-gnu/ld-2.27.so
no symbols loaded, base address: 0x7f708f2a5000, size: 167936(0x29000)

library loaded: /home/user/demo/dotnet-sdk-6.0/host/fxr/6.0.3/libhostfxr.so
no symbols loaded, base address: 0x7f708f436000, size: 425984(0x68000)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libhostpolicy.so
no symbols loaded, base address: 0x7f708f3d7000, size: 389120(0x5f000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libcoreclr.so
no symbols loaded, base address: 0x7f708da1b000, size: 7303168(0x6f7000)

library loaded: /lib/x86_64-linux-gnu/librt-2.27.so
no symbols loaded, base address: 0x7f708d813000, size: 2129920(0x208000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libcoreclrtraceptprovider.so
no symbols loaded, base address: 0x7f708f31f000, size: 753664(0xb8000)

library loaded: /usr/lib/x86_64-linux-gnu/liblttng-ust.so.0.0.0
no symbols loaded, base address: 0x7f708d598000, size: 2564096(0x272000)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libhostpolicy.so
no symbols loaded, base address: 0x7f708f3d7000, size: 389120(0x5f000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libcoreclr.so
no symbols loaded, base address: 0x7f708da1b000, size: 7303168(0x6f7000)

library loaded: /lib/x86_64-linux-gnu/librt-2.27.so
no symbols loaded, base address: 0x7f708d813000, size: 2129920(0x208000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libcoreclrtraceptprovider.so
no symbols loaded, base address: 0x7f708f31f000, size: 753664(0xb8000)

library loaded: /usr/lib/x86_64-linux-gnu/liblttng-ust.so.0.0.0
no symbols loaded, base address: 0x7f708d598000, size: 2564096(0x272000)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /usr/lib/x86_64-linux-gnu/liblttng-ust-tracepoint.so.0.0.0
no symbols loaded, base address: 0x7f708d37c000, size: 2146304(0x20c000)

library loaded: /usr/lib/x86_64-linux-gnu/liburcu-bp.so.6.0.0
no symbols loaded, base address: 0x7f708d174000, size: 2129920(0x208000)

library loaded: /usr/lib/x86_64-linux-gnu/liburcu-cds.so.6.0.0
no symbols loaded, base address: 0x7f708cf6a000, size: 2138112(0x20a000)
^running

native thread created, id: 12918

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libclrjit.so
no symbols loaded, base address: 0x7f7088620000, size: 3358720(0x334000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f7013f10000, size: 10601472(0xa1c400)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /usr/lib/x86_64-linux-gnu/liblttng-ust-tracepoint.so.0.0.0
no symbols loaded, base address: 0x7f708d37c000, size: 2146304(0x20c000)

library loaded: /usr/lib/x86_64-linux-gnu/liburcu-bp.so.6.0.0
no symbols loaded, base address: 0x7f708d174000, size: 2129920(0x208000)

library loaded: /usr/lib/x86_64-linux-gnu/liburcu-cds.so.6.0.0
no symbols loaded, base address: 0x7f708cf6a000, size: 2138112(0x20a000)
^running

native thread created, id: 12918

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libclrjit.so
no symbols loaded, base address: 0x7f7088620000, size: 3358720(0x334000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Private.CoreLib.dll
no symbols loaded, base address: 0x7f7013f10000, size: 10601472(0xa1c400)
```

How to use Interop NetCoreDbg (CLI)

```
native thread created, id: 12919

managed thread created, id: 12894

library loaded: /home/user/demo/managed/demo/bin/Debug/net6.0/demo.dll
symbols loaded, base address: 0x7f708f4a5000, size: 5632(0x1600)
breakpoint modified,  Breakpoint 1 at reportCB()

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Runtime.dll
no symbols loaded, base address: 0x7f7088618000, size: 32256(0x7e00)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Console.dll
no symbols loaded, base address: 0x7f7014a90000, size: 376832(0x5c000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/System.Threading.dll
no symbols loaded, base address: 0x7f7014b00000, size: 266752(0x41200)
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/libSystem.Native.so
no symbols loaded, base address: 0x7f7088600000, size: 98304(0x18000)

library loaded: /usr/lib/x86_64-linux-gnu/libicuuc.so.60.2
no symbols loaded, base address: 0x7f708821f000, size: 3895296(0x3b7000)

library loaded: /usr/lib/x86_64-linux-gnu/libicudata.so.60.2
no symbols loaded, base address: 0x7f7079c56000, size: 29003776(0x1ba9000)

library loaded: /usr/lib/x86_64-linux-gnu/libicui18n.so.60.2
no symbols loaded, base address: 0x7f70797b5000, size: 4853760(0x4a1000)

library loaded: /home/user/demo/dotnet-sdk-
6.0/shared/Microsoft.NETCore.App/6.0.3/Microsoft.Win32.Primitives.dll
no symbols loaded, base address: 0x7f7014b70000, size: 210944(0x33800)

native thread created, id: 12920
Starting!
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/managed/demo/bin/Debug/net6.0/liballocator.so
symbols loaded, base address: 0x7f708801d000, size: 2105344(0x202000)

stopped, reason: breakpoint 1 hit, thread id: 12894, stopped threads: all,
times= 0, frame={demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13}
```

How to use Interop NetCoreDbg (CLI)

```
library loaded: /home/user/demo/managed/demo/bin/Debug/net6.0/liballocator.so
symbols loaded, base address: 0x7f708801d000, size: 2105344(0x202000)

stopped, reason: breakpoint 1 hit, thread id: 12894, stopped threads: all,
times= 0, frame={demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13}
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> bt
#0: 0x00007f70149bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f70149baa9c [CoreCLR Native Frame]
#2: 0x00007f708801d63c liballocator.so` alloc() at /home/user/demo/native/allocator.cpp:10
#3: 0x00007f70149ba9ad [CoreCLR Native Frame]
#4: 0x00007f70149b2f3e demo.dll` demo.Program.Main() at /home/user/demo/managed/demo/Program.cs:22
#5: 0x00007f708dd9d9c7 libcoreclr.so` unnamed_symbol, libcoreclr.so + 3680711
#6: 0x00007f708dbd400b libcoreclr.so` unnamed_symbol, libcoreclr.so + 1806347
#7: 0x00007f708daa9e9a libcoreclr.so` unnamed_symbol, libcoreclr.so + 585370
#8: 0x00007f708daaa1f1 libcoreclr.so` unnamed_symbol, libcoreclr.so + 586225
#9: 0x00007f708dadec13 libcoreclr.so` unnamed_symbol, libcoreclr.so + 801811
#10: 0x00007f708da9212f libcoreclr.so` coreclr_execute_assembly() + 127
#11: 0x00007f708f3ef5b1 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 99761
#12: 0x00007f708f3efa41 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 100929
#13: 0x00007f708f3f046c libhostpolicy.so` corehost_main() + 172
#14: 0x00007f708f449d04 libhostfxr.so` unnamed_symbol, libhostfxr.so + 81156
#15: 0x00007f708f448409 libhostfxr.so` unnamed_symbol, libhostfxr.so + 74761
#16: 0x00007f708f4438fb libhostfxr.so` hostfxr_main_startupinfo() + 171
#17: 0x00005591941a1faa unnamed_symbol
#18: 0x00005591941a2420 unnamed_symbol
#19: 0x00007f708e173c87 libc-2.27.so` __libc_start_main() + 231
#20: 0x0000559194196d2a unnamed_symbol
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> bt
#0: 0x00007f70149bab0a demo.dll` demo.Program.reportCB() at
/home/user/demo/managed/demo/Program.cs:13
#1: 0x00007f70149baa9c [CoreCLR Native Frame]
#2: 0x00007f708801d63c liballocator.so` alloc() at /home/user/demo/native/allocator.cpp:10
#3: 0x00007f70149ba9ad [CoreCLR Native Frame]
#4: 0x00007f70149b2f3e demo.dll` demo.Program.Main() at /home/user/demo/managed/demo/Program.cs:22
#5: 0x00007f708dd9d9c7 libcoreclr.so` unnamed_symbol, libcoreclr.so + 3680711
#6: 0x00007f708dbd400b libcoreclr.so` unnamed_symbol, libcoreclr.so + 1806347
#7: 0x00007f708daa9e9a libcoreclr.so` unnamed_symbol, libcoreclr.so + 585370
#8: 0x00007f708daaa1f1 libcoreclr.so` unnamed_symbol, libcoreclr.so + 586225
#9: 0x00007f708dadec13 libcoreclr.so` unnamed_symbol, libcoreclr.so + 801811
#10: 0x00007f708da9212f libcoreclr.so` coreclr_execute_assembly() + 127
#11: 0x00007f708f3ef5b1 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 99761
#12: 0x00007f708f3efa41 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 100929
#13: 0x00007f708f3f046c libhostpolicy.so` corehost_main() + 172
#14: 0x00007f708f449d04 libhostfxr.so` unnamed_symbol, libhostfxr.so + 81156
#15: 0x00007f708f448409 libhostfxr.so` unnamed_symbol, libhostfxr.so + 74761
#16: 0x00007f708f4438fb libhostfxr.so` hostfxr_main_startupinfo() + 171
#17: 0x00005591941a1faa unnamed_symbol
#18: 0x00005591941a2420 unnamed_symbol
#19: 0x00007f708e173c87 libc-2.27.so` __libc_start_main() + 231
#20: 0x0000559194196d2a unnamed_symbol
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> b allocator.cpp:9
 Breakpoint 2 at /home/user/demo/native/allocator.cpp:9
ncdb> c
^running
Reporting 94083752734336, size 123

stopped, reason: breakpoint 2 hit, thread id: 12894, stopped threads: all,
times= 1, frame={liballocator.so` alloc() at
/home/user/demo/native/allocator.cpp:9}
```

```
1   #include <malloc.h>
2
3   typedef void* FPtr(void*, int);
4
5   extern "C"
6   {
7       void* alloc(int size, FPtr callback)
8       {
9           void* mem = malloc(size);
10          (*callback)(mem, size);
11          return mem;
12      }
13  }
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> bt all

Thread 11, id="12920", name="<No name>", state="running", type="native"
#0: 0x00007f708f097474 libpthread-2.27.so` read() + 68
#1: 0x00007f708860f03f libSystem.Native.so` unnamed_symbol, libSystem.Native.so
+ 61503
#2: 0x00007f708f08d6db libpthread-2.27.so` start_thread() + 219
#3: 0x00007f708e27361f libc-2.27.so` clone() + 63
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> bt all

Thread 11, id="12920", name="<No name>", state="running", type="native"
#0: 0x00007f708f097474 libpthread-2.27.so` read() + 68
#1: 0x00007f708860f03f libSystem.Native.so` unnamed_symbol, libSystem.Native.so
+ 61503
#2: 0x00007f708f08d6db libpthread-2.27.so` start_thread() + 219
#3: 0x00007f708e27361f libc-2.27.so` clone() + 63
```

How to use Interop NetCoreDbg (CLI)

```
Thread 1, id="12894", name="Main Thread", state="stopped", type="managed"
#0: 0x00007f708801d61a liballocator.so` alloc() at /home/user/demo/native/allocator.cpp:9
#1: 0x00007f70149ba9ad [CoreCLR Native Frame]
#2: 0x00007f70149b2f3e demo.dll` demo.Program.Main() at /home/user/demo/managed/demo/Program.cs:22
#3: 0x00007f708dd9d9c7 libcoreclr.so` unnamed_symbol, libcoreclr.so + 3680711
#4: 0x00007f708dbd400b libcoreclr.so` unnamed_symbol, libcoreclr.so + 1806347
#5: 0x00007f708daa9e9a libcoreclr.so` unnamed_symbol, libcoreclr.so + 585370
#6: 0x00007f708daaa1f1 libcoreclr.so` unnamed_symbol, libcoreclr.so + 586225
#7: 0x00007f708dadec13 libcoreclr.so` unnamed_symbol, libcoreclr.so + 801811
#8: 0x00007f708da9212f libcoreclr.so` coreclr_execute_assembly() + 127
#9: 0x00007f708f3ef5b1 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 99761
#10: 0x00007f708f3efa41 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 100929
#11: 0x00007f708f3f046c libhostpolicy.so` corehost_main() + 172
#12: 0x00007f708f449d04 libhostfxr.so` unnamed_symbol, libhostfxr.so + 81156
#13: 0x00007f708f448409 libhostfxr.so` unnamed_symbol, libhostfxr.so + 74761
#14: 0x00007f708f4438fb libhostfxr.so` hostfxr_main_startupinfo() + 171
#15: 0x00005591941a1faa unnamed_symbol
#16: 0x00005591941a2420 unnamed_symbol
#17: 0x00007f708e173c87 libc-2.27.so` __libc_start_main() + 231
#18: 0x0000559194196d2a unnamed_symbol
```

How to use Interop NetCoreDbg (CLI)

```
Thread 1, id="12894", name="Main Thread", state="stopped", type="managed"
#0: 0x00007f708801d61a liballocator.so` alloc() at /home/user/demo/native/allocator.cpp:9
#1: 0x00007f70149ba9ad [CoreCLR Native Frame]
#2: 0x00007f70149b2f3e demo.dll` demo.Program.Main() at /home/user/demo/managed/demo/Program.cs:22
#3: 0x00007f708dd9d9c7 libcoreclr.so` unnamed_symbol, libcoreclr.so + 3680711
#4: 0x00007f708dbd400b libcoreclr.so` unnamed_symbol, libcoreclr.so + 1806347
#5: 0x00007f708daa9e9a libcoreclr.so` unnamed_symbol, libcoreclr.so + 585370
#6: 0x00007f708daaa1f1 libcoreclr.so` unnamed_symbol, libcoreclr.so + 586225
#7: 0x00007f708dadec13 libcoreclr.so` unnamed_symbol, libcoreclr.so + 801811
#8: 0x00007f708da9212f libcoreclr.so` coreclr_execute_assembly() + 127
#9: 0x00007f708f3ef5b1 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 99761
#10: 0x00007f708f3efa41 libhostpolicy.so` unnamed_symbol, libhostpolicy.so + 100929
#11: 0x00007f708f3f046c libhostpolicy.so` corehost_main() + 172
#12: 0x00007f708f449d04 libhostfxr.so` unnamed_symbol, libhostfxr.so + 81156
#13: 0x00007f708f448409 libhostfxr.so` unnamed_symbol, libhostfxr.so + 74761
#14: 0x00007f708f4438fb libhostfxr.so` hostfxr_main_startupinfo() + 171
#15: 0x00005591941a1faa unnamed_symbol
#16: 0x00005591941a2420 unnamed_symbol
#17: 0x00007f708e173c87 libc-2.27.so` __libc_start_main() + 231
#18: 0x0000559194196d2a unnamed_symbol
```

How to use Interop NetCoreDbg (CLI)

```
ncdb> frame 2
#2: 0x00007f70149b2f3e demo.dll` demo.Program.Main() at
/home/user/demo/managed/demo/Program.cs:22

ncdb> p i
i = 1
ncdb> p i*17+123
i*17+123 = 140
```

```
20          for (int i = 0; i < 10; ++i)
21          {
22              IntPtr value = alloc(i * 17 + 123, cb);
23          }
```

Interop Debugging: Competitors

debugger	platform	target OS for apps	architecture support	open source
Visual Studio Debugger	.NET, Python	Windows	x64, x86, arm64	no
Android Studio Debugger	Java	Android	x64, arm64	yes
LLDB with SOS plugin	.NET	Linux	x64, x86, arm32, arm64	yes
NetCoreDbg Interop Debugger	.NET	Linux	x64, x86, arm32, arm64	yes

Interop Debugging: Competitors

debugger	native, managed, mixed backtrace	native breakpoints	managed breakpoints	native eval	managed eval	native stepping	managed stepping
Visual Studio Debugger	yes	yes	yes	yes (3)	yes	yes	yes
Android Studio Debugger	yes (4)	yes	yes	yes (3)	yes	yes	yes
LLDB with SOS plugin	yes	yes	yes (5)	yes	yes (6)	yes	no
NetCoreDbg Interop Debugger	yes	yes (1)	yes	no (2)	yes	no (2)	yes

(1) only line breakpoints are supported currently
(2) not supported yet
(3) limited, can't evaluate all native code
(4) native or managed only, combined backtrace can be obtained using additional cli tool (debuggerd) separate from GUI debugger
(5) only function and line breakpoints are supported, no exception breakpoints
(6) very limited by design, can only show variables and fields (no managed code execution)

Interop Debugging: Competitors

debugger	memory consumption (PSS) of debugger process on arm64, Mb
LLDB with SOS plugin	230
NetCoreDbg Interop Debugger	47

Interop Debugging: Future Directions

- Native function breakpoints

Interop Debugging: Future Directions

- Native function breakpoints
- Stop on native signals and C++ exceptions from user native code

Interop Debugging: Future Directions

- Native function breakpoints
- Stop on native signals and C++ exceptions from user native code
- Native evaluation

Interop Debugging: Future Directions

- Native function breakpoints
- Stop on native signals and C++ exceptions from user native code
- Native evaluation
- Native stepping

Summary

What we learned today?

- how .NET debugger is implemented

Summary

What we learned today?

- how .NET debugger is implemented
- how interop .NET debugger is implemented

Summary

What we learned today?

- how .NET debugger is implemented
- how interop .NET debugger is implemented
- how to use .NET interop debugger

Want to learn more?

- ICorDebug API:
  - https://learn.microsoft.com/en-us/dotnet/framework/unmanaged-api/debugging/icordebug-interface
- Interop Debugging:
  - https://learn.microsoft.com/en-us/archive/blogs/jmstall/what-is-interop-debugging
  - https://learn.microsoft.com/en-gb/archive/blogs/jmstall/tips-for-writing-an-interop-debugger
  - https://learn.microsoft.com/en-us/archive/blogs/jmstall/you-dont-want-to-write-an-interop-debugger
- How to use NetCoreDbg:
  - https://github.com/Samsung/netcoredbg/blob/master/docs/cli.md

How to participate?

Contributions are welcomed!

https://github.com/samsung/netcoredbg

Questions

Thank you!