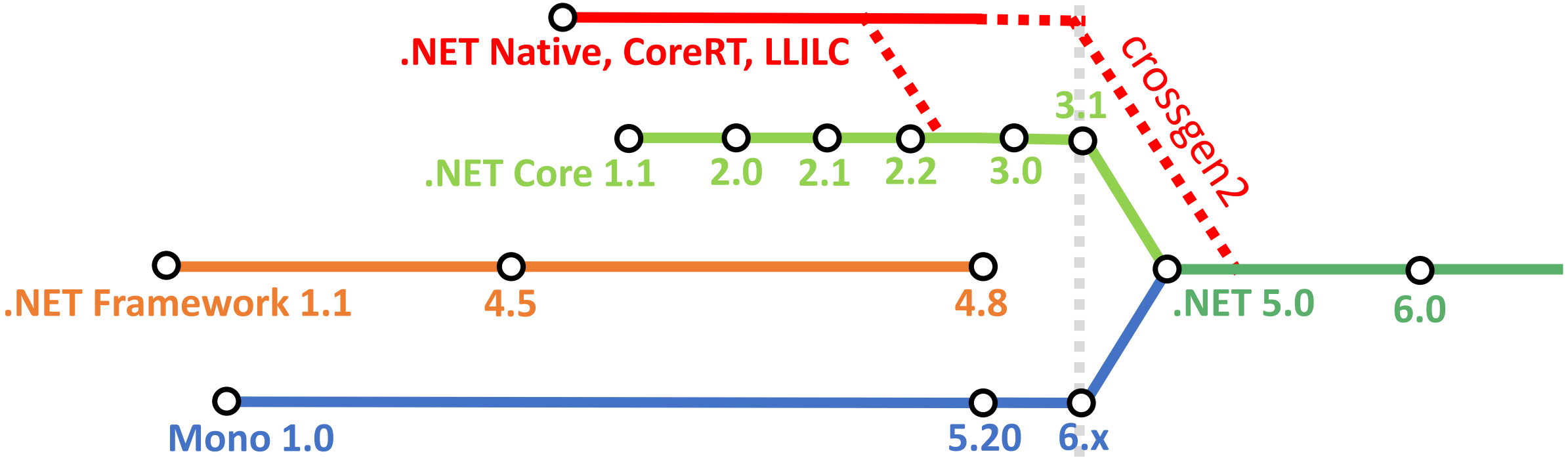
.NET 5.0 Runtimes

@EgorBo
Engineer at Microsoft

Timeline

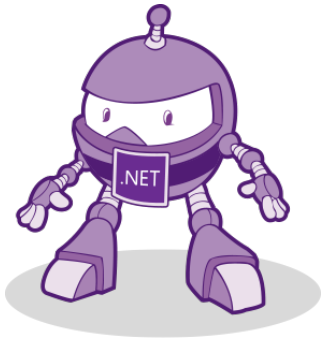

.NET Native, CoreRT, LLILC
crossgen2
3.1
.NET Core 1.1
2.0
2.1
2.2
3.0
.NET Framework 1.1
4.5
4.8
.NET 5.0
6.0
Mono 1.0
5.20
6.x

.NET 5.0

- Swappable runtimes
- Interop with Java and ObjC/Swift
- Smaller binaries
- Faster startup
- More targets (e.g. WebAssembly)
- Performance!

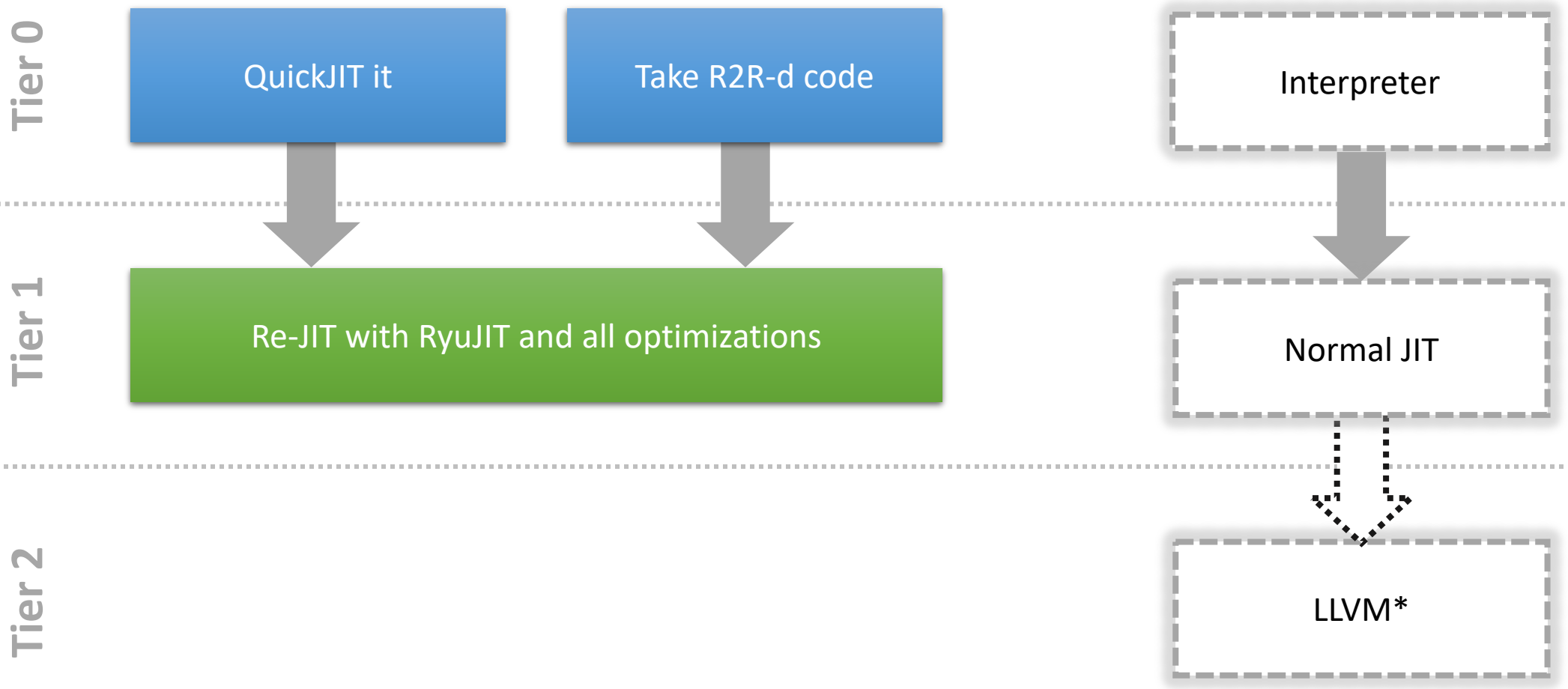
Runtimes for C# (IL):

- **.NET Framework** 4.x
  - JIT
  - AOT (Fragile NGEN)
- **CoreCLR**
  - JIT (Tiered)
  - AOT (ReadyToRun – R2R)
- **Mono**
  - JIT
  - (Full)AOT
  - LLVM
  - Interpreter
- **CoreRT**
- Unity **IL2CPP**
- Unity **Burst**



Tiering Compilation

Mono (working prototype):
@migueldeicaza @lewurm



Tiered vs non-Tiered


```
private static int MyReadOnlyProp { get; } = SetupField();

private static int SetupField() => 42;



static string Test(int x)
{
    return MyReadOnlyProp / 10;
}
```

Tiered vs non-Tiered

```
static int Test(int x)
{
    return MyReadOnlyProp / 10;
}
```


1) Tiered Compilation is **ON**

```
G_M7895_IG01:
       push     rbp
       sub      rsp, 32
       lea      rbp, [rsp+20H]
       mov      gword ptr [rbp+10H…
       mov      dword ptr [rbp+18H]…

G_M7895_IG02:
       call     Program:get_MyReadOnly…
       mov      ecx, 10
       cdq
       idiv     edx:eax, ecx

G_M7895_IG03:
       lea      rsp, [rbp]
       pop      rbp
       ret
; Total bytes of code: 36
```

2) Tiered Compilation is **OFF**

```
G_M7896_IG01:
       sub      rsp, 40

G_M7896_IG02:
       mov      rcx, 0xD1FFAB1E
       mov      edx, 1
       call     CORINFO_HELP_GETSHAR…
       mov      edx, 0xD1FFAB1E
       mov      eax, edx
       imul     edx:eax, dword …
       mov      eax, edx
       shr      eax, 31
       sar      edx, 2
       add      eax, edx

G_M7896_IG03:
       add      rsp, 40
       ret
; Total bytes of code: 52
```

3) Tiered Compilation is **ON** + [AggressiveOptimization]

```
G_M7896_IG01:
       sub      rsp, 40

G_M7896_IG02:
       mov      rcx, 0xD1FFAB1E
       mov      edx, 1
       call     CORINFO_HELP_GETSHAR…
       mov      edx, 0xD1FFAB1E
       mov      eax, edx
       imul     edx:eax, dword …
       mov      eax, edx
       shr      eax, 31
       sar      edx, 2
       add      eax, edx

G_M7896_IG03:
       add      rsp, 40
       ret
; Total bytes of code: 52
```

4) Tiered Compilation is **ON**

After a while in a hot path:

```
G_M7892_IG02:
       mov      eax, 4

G_M7892_IG03:
       ret
```

Don't turn off Tiered Compilation ☺

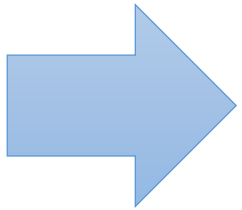
Start up time

	R2R for BCL (default)	Full R2R	IL only
QuickJIT=0			
QuickJIT=1 (default)			
QuickJIT=1, ForLoops=1			

```
COMPlus_TieredCompilation=1
COMPlus_TC_QuickJit=1
COMPlus_TC_QuickJitForLoops=0
```

Profile guided optimizations

```
void Foo1(int a, int b)
{
    if (a == 42 || b == 43)
        mostly false  mostly true
    {
        Foo2();
    }
}
```



```
void Foo1(int a, int b)
{
    if (b == 43 || a == 42)
    {
        Foo2();
    }
}
```

Provide block counts in tiered compilation from R2R images #27511

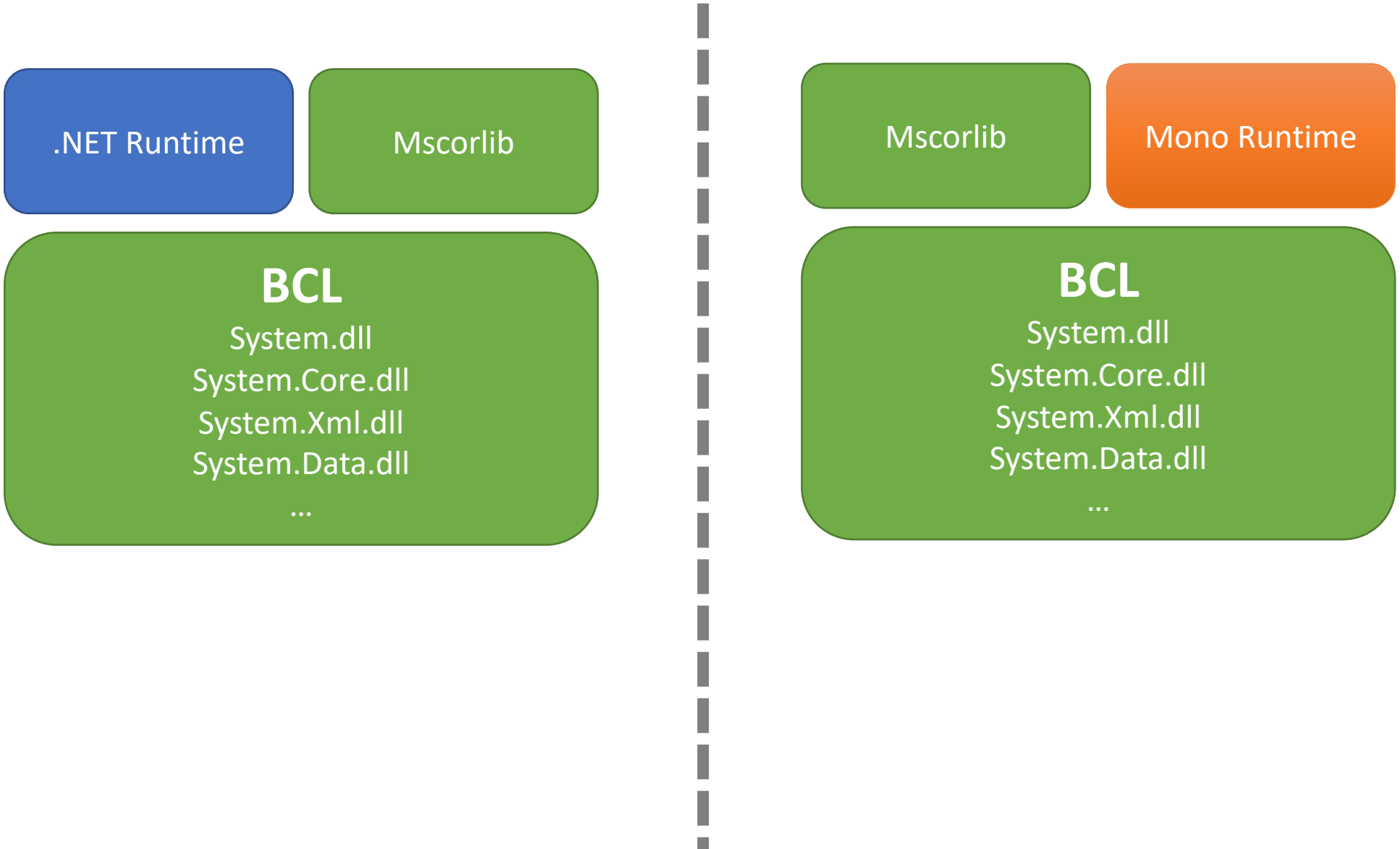
Open davidwrighton opened this issue 19 hours ago · 4 comments

.NET 5: Swappable runtimes

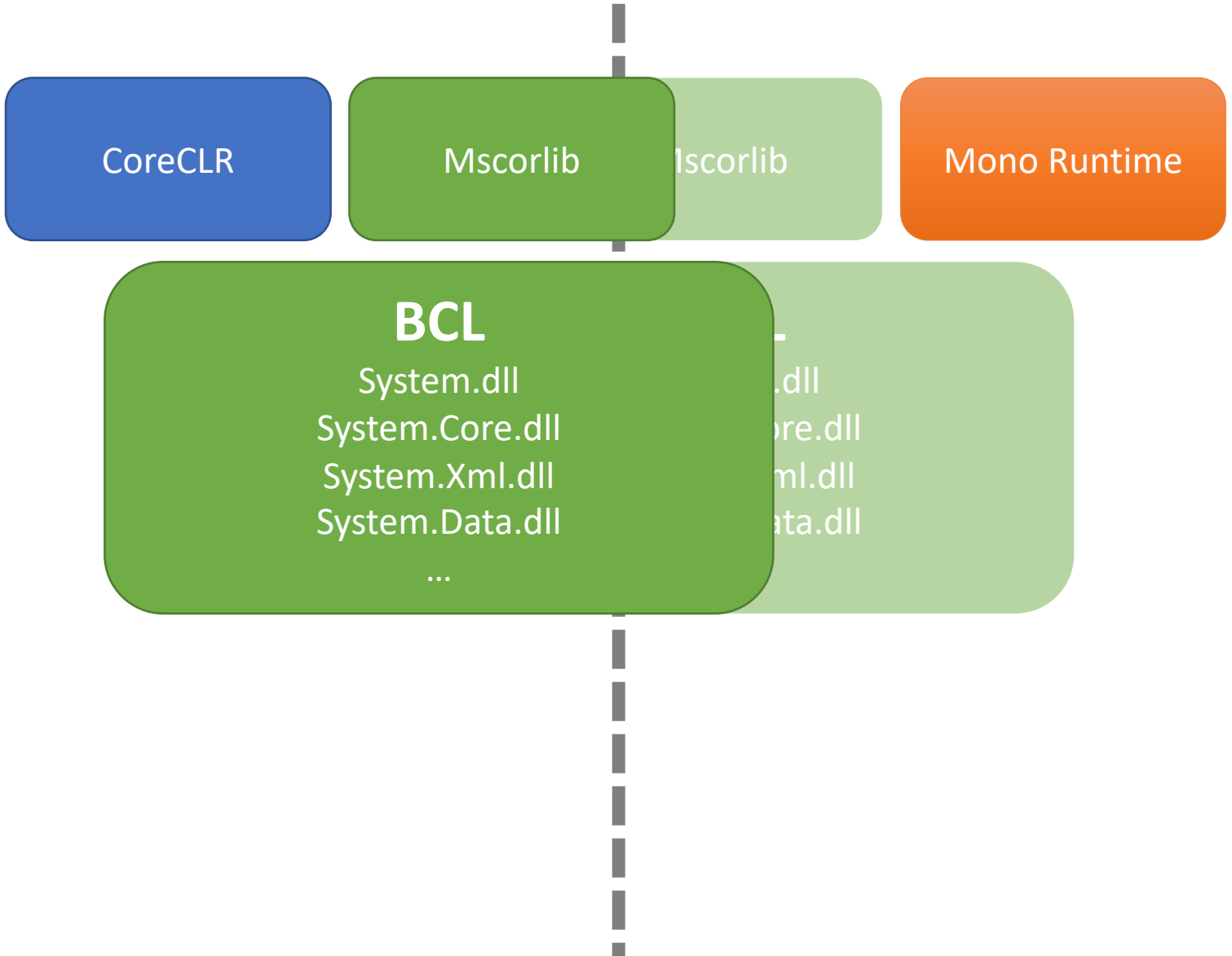
Possible syntax:

```
dotnet publish      \
    -r osx-x64      \
    -c Release      \
    -vm Mono        \
    /vm:aot=llvm    \
    /vm:ffast-math  \
```

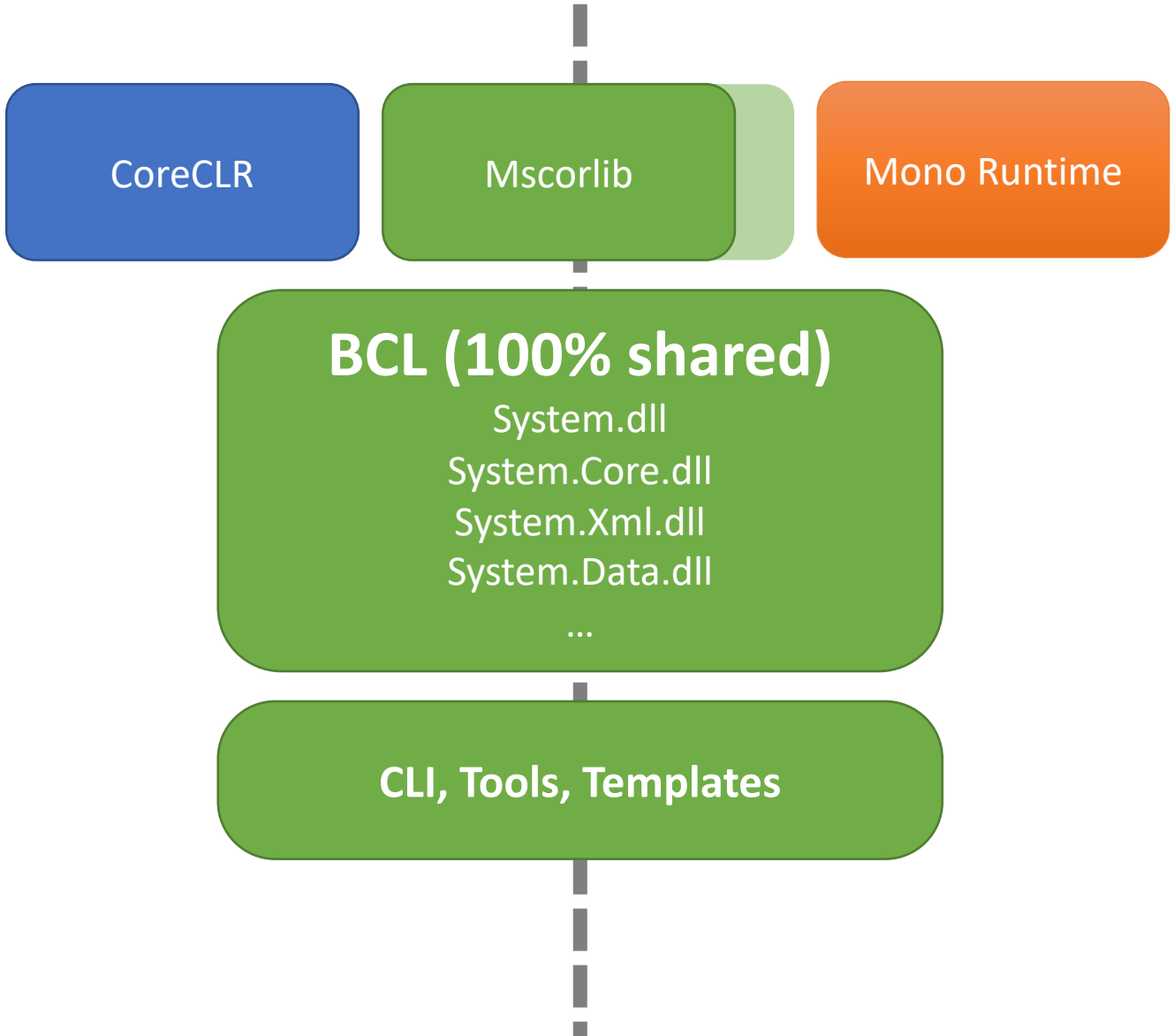
Ancient times



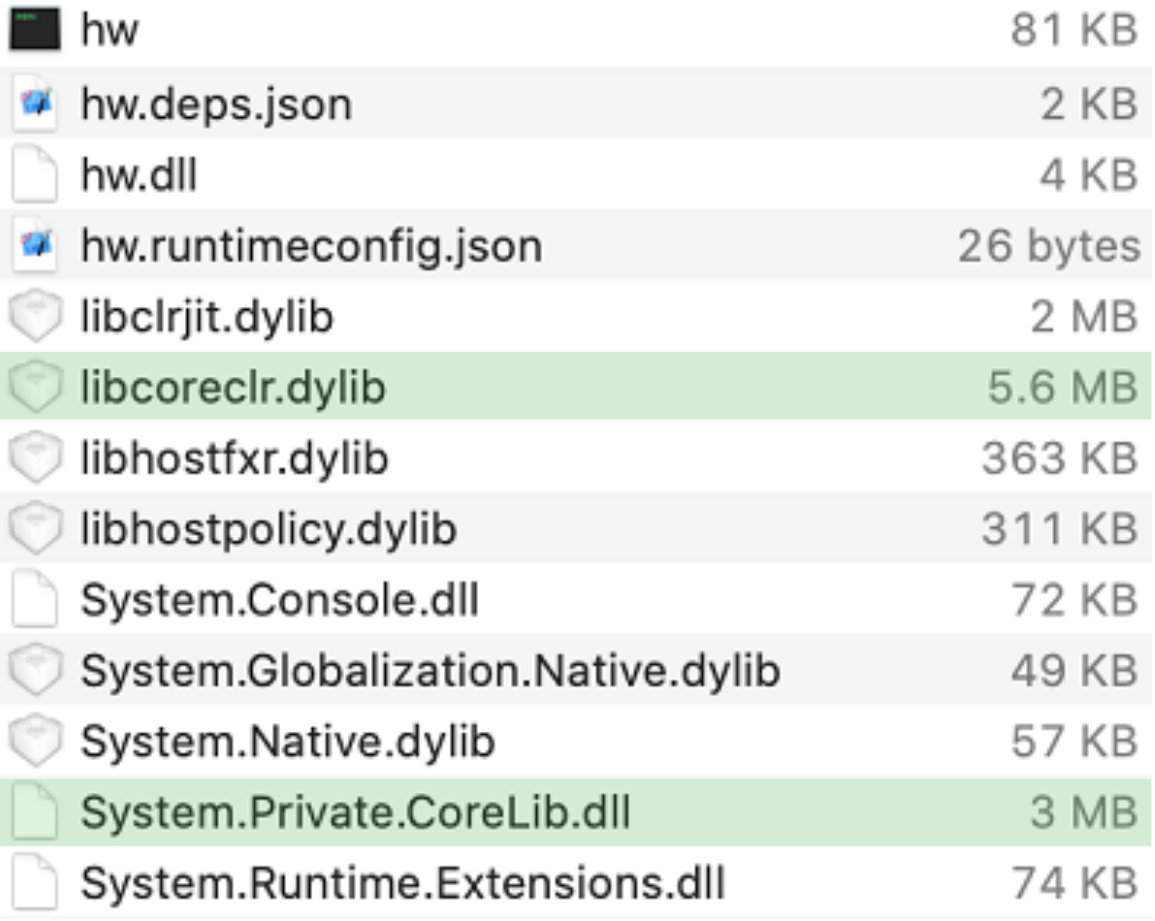
Nowadays


CoreCLR
Mscorlib
Mono Runtime
BCL
System.dll
System.Core.dll
System.Xml.dll
System.Data.dll
…

.NET 5


CoreCLR
Mscorlib
Mono Runtime
BCL (100% shared)
System.dll
System.Core.dll
System.Xml.dll
System.Data.dll
…
CLI, Tools, Templates

Custom runtime for .NET

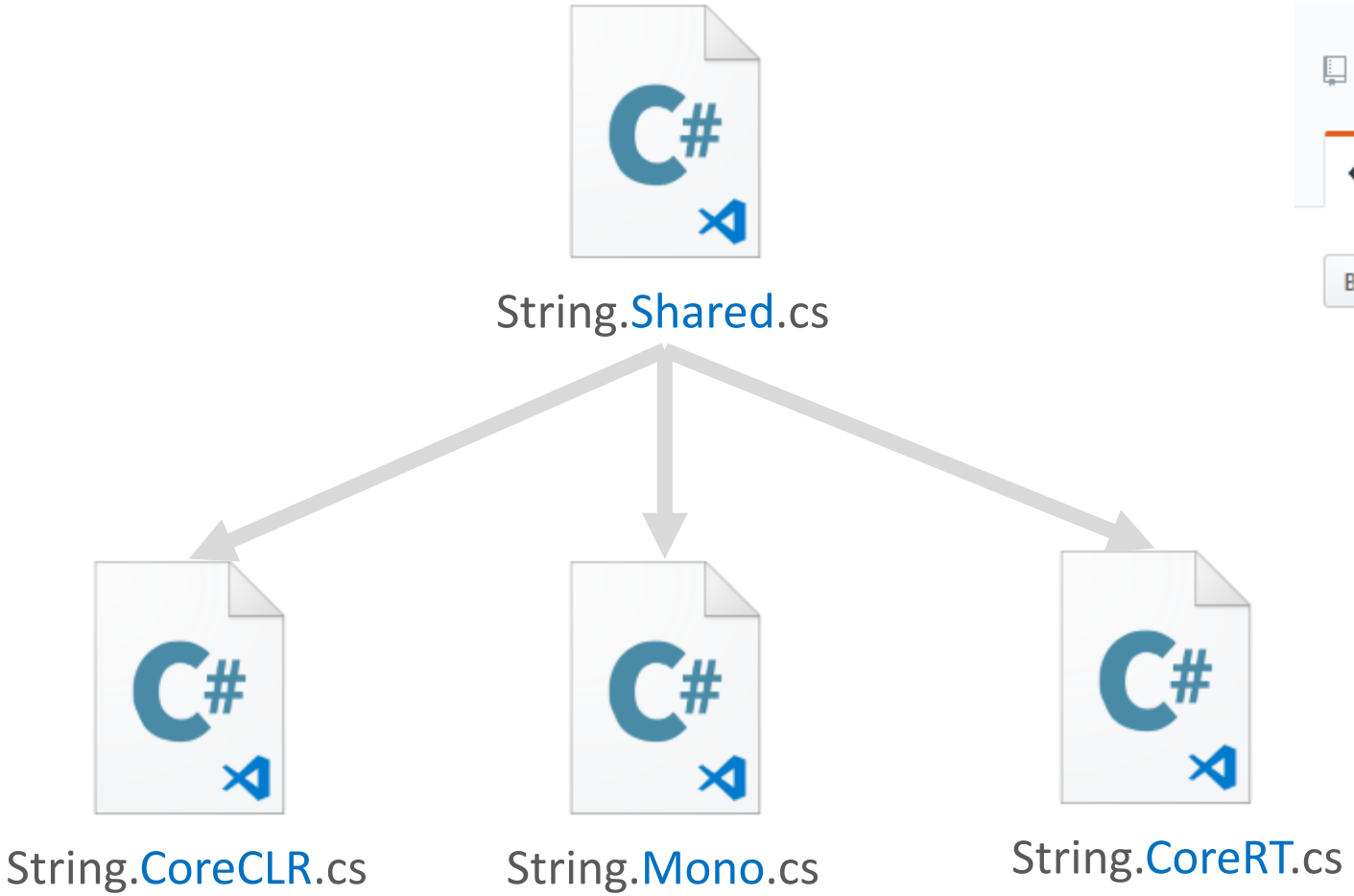


Custom runtime for .NET: native interface


```
extern "C" int coreclr_initialize(
                const char* exePath,
                const char*
                appDomainFriendlyName,
                int propertyCount,
                const char** propertyKeys,
                const char** propertyValues,
                void** hostHandle,
                unsigned int* domainId)
{
    printf("coreclr_initialize!\n");
    return 0;
}
```

```
extern "C" int coreclr_shutdown(…)
extern "C" int coreclr_shutdown_2(…)
extern "C" int coreclr_create_delegate(…)
extern "C" int coreclr_execute_assembly(…)
```

Shared System.Private.CoreLib (mscorlib)



SOON: Monorepo!

Mono modes:

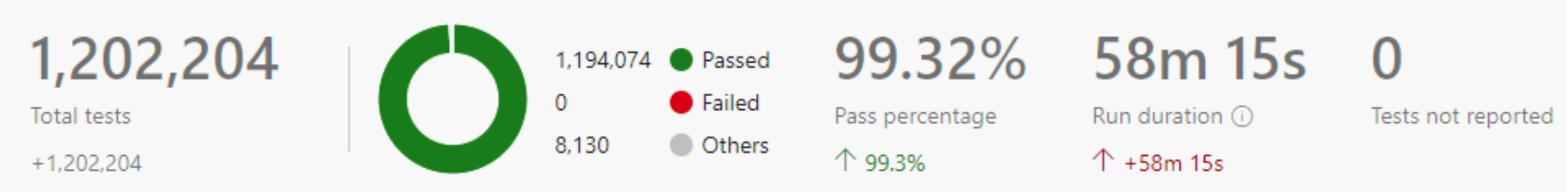
- JIT
  - Normal JIT
  - LLVM JIT
  - WIP: Tiered
- AOT
  - Mini
  - LLVM
- fullAOT (for platforms where there is no JIT)
- llvmonly (pure llvm IR for watchOS and wasm)
- Interprpeter

Mono targets:

- WebAssembly
- Android
- iOS, tvOS, watchOS
- Big-Endian (community support)
- Other/new platforms where Mono can easily jump into via interpreter or LLVM

Current state of MonoVM for .NET 5

Linux x64 + macOS x64 + Linux AArch64 CI agents:



	A	B	C
1	**Benchmark**	**coreclr**	**llvm-jit**
2	BenchmarksGame.BinaryTrees_2.RunBench	1.00	1.13
3	BenchmarksGame.BinaryTrees_5.RunBench	1.00	1.55
4	BenchmarksGame.FannkuchRedux_2.RunBench(n: 10; expectedS	1.00	1.09
5	BenchmarksGame.FannkuchRedux_5.RunBench(n: 10; expectedS	1.00	0.89
6	BenchmarksGame.Fasta_1.RunBench	1.00	0.93
7	BenchmarksGame.Fasta_2.RunBench	1.00	0.75
8	BenchmarksGame.KNucleotide_1.RunBench	1.00	0.93
9	BenchmarksGame.KNucleotide_9.RunBench	1.00	0.97
10	BenchmarksGame.Mandelbrot_2.Bench(width: 4000; checksum: "	1.00	0.87
11	BenchmarksGame.MandelBrot_7.Bench(size: 4000; lineLength: 50	1.00	0.95
12	BenchmarksGame.NBody_3.RunBench	1.00	0.98
13	BenchmarksGame.PiDigits_3.RunBench(n: 3000; expected: "86494	1.00	1.98
14	BenchmarksGame.RegexRedux_1.RunBench	1.00	0.99
15	BenchmarksGame.RegexRedux_5.RunBench	1.00	1.05
16	BenchmarksGame.ReverseComplement_1.RunBench	1.00	0.93
17	BenchmarksGame.ReverseComplement_6.RunBench	1.00	0.73
18	BenchmarksGame.SpectralNorm_1.RunBench	1.00	1.00
19	BenchmarksGame.SpectralNorm_3.RunBench	1.00	0.40
20	Benchstone.BenchF.Adams.Test	1.00	0.96
21	Benchstone.BenchF.BenchMk2.Test	1.00	0.40
22	Benchstone.BenchF.BenchMrk.Test	1.00	0.38
23	Benchstone.BenchF.Bisect.Test	1.00	0.72
24	Benchstone.BenchF.DMath.Test	1.00	0.64
25	Benchstone.BenchF.FFT.Test	1.00	0.95
26	Benchstone.BenchF.InProd.Test	1.00	1.01
27	Benchstone.BenchF.InvMt.Test	1.00	1.14
28	Benchstone.BenchF.LLoops.Test	1.00	1.00
29	Benchstone.BenchF.Lorenz.Test	1.00	0.54
30	Benchstone.BenchF.MatInv4.Test	1.00	1.01
31	Benchstone.BenchF.NewtR.Test	1.00	0.87
32	Benchstone.BenchF.Regula.Test	1.00	0.96
33	Benchstone.BenchF.Romber.Test	1.00	0.86
34	Benchstone.BenchF.Secant.Test	1.00	0.86
35	Benchstone.BenchF.SqMtx.Test	1.00	1.10
36	Benchstone.BenchF.Whetsto.Test	1.00	0.78
37	Benchstone.BenchI.Ackermann.Test	1.00	1.64
38	Benchstone.BenchI.AddArray.Test	1.00	2.08
39	Benchstone.BenchI.AddArray2.Test	1.00	1.27
40	Benchstone.BenchI.Array1.Test	1.00	0.86

LLVM back-end:

Pros:

- Thousands of peephole optimizations
- Built-in Fast-Math mode
- Better register management
- Loop-related optimizations including vectorization
- Lots of targets, tools and libs (e.g. superoptimizers, GPU-related, etc)

Cons:

- Compilation speed (doesn't matter for AOT or tiered compilation)
- Binary size for LLVM-JIT mode (+20-30 Mb)
- Knows nothing about C# specific optimizations such as devirtualization, de-boxing, etc.
- Expects high quality IR from front-end

LLVM: From C# to machine code

C#

```
public static float Add(float a, float b)
{
    float c = a + b;
    return c;
}
```

IL

```
IL_0000:  ldarg.0
IL_0001:  ldarg.1
IL_0002:  add
IL_0003:  ret
```

LLVM IR

```
%3 = fadd float %0, %1
ret float %3
```

ASM

```
vaddss xmm0, xmm0, xmm1
ret
```

LLVM: tons of optimizations

- `-adce`: Aggressive Dead Code Elimination
- `-always-inline`: Inliner for `always_inline` functions
- `-argpromotion`: Promote 'by reference' arguments to scalars
- `-bb-vectorize`: Basic-Block Vectorization
- `-block-placement`: Profile Guided Basic Block Placement
- `-break-crit-edges`: Break critical edges in CFG
- `-codegenprepare`: Optimize for code generation
- `-constmerge`: Merge Duplicate Global Constants
- `-constprop`: Simple constant propagation
- `-dce`: Dead Code Elimination
- `-deadargelim`: Dead Argument Elimination
- `-deadtypeelim`: Dead Type Elimination
- `-die`: Dead Instruction Elimination
- `-dse`: Dead Store Elimination
- `-functionattrs`: Deduce function attributes
- `-globaldce`: Dead Global Elimination
- `-globalopt`: Global Variable Optimizer
- `-gvn`: Global Value Numbering
- `-indvars`: Canonicalize Induction Variables
- `-inline`: Function Integration/Inlining
- `-instcombine`: Combine redundant instructions
- `-aggressive-instcombine`: Combine expression patterns
- `-internalize`: Internalize Global Symbols
- `-ipconstprop`: Interprocedural constant propagation
- `-ipsccp`: Interprocedural Sparse Conditional Constant Propagation
- `-jump-threading`: Jump Threading
- `-lcssa`: Loop-Closed SSA Form Pass
- `-licm`: Loop Invariant Code Motion
- `-loop-deletion`: Delete dead loops
- `-loop-extract`: Extract loops into new functions
- `-loop-extract-single`: Extract at most one loop into a new function
- `-loop-reduce`: Loop Strength Reduction
- `-loop-rotate`: Rotate Loops
- `-loop-simplify`: Canonicalize natural loops
- `-loop-unroll`: Unroll loops
- `-loop-unroll-and-jam`: Unroll and Jam loops
- `-loop-unswitch`: Unswitch loops
- `-loweratomic`: Lower atomic intrinsics to non-atomic form
- `-lowerinvoke`: Lower invokes to calls, for unwindless code generators
- `-lowerswitch`: Lower `SwitchInst`s to branches
- `-mem2reg`: Promote Memory to Register
- `-memcpyopt`: MemCpy Optimization
- `-mergefunc`: Merge Functions
- `-mergereturn`: Unify function exit nodes
- `-partial-inliner`: Partial Inliner
- `-prune-eh`: Remove unused exception handling info
- `-reassociate`: Reassociate expressions
- `-reg2mem`: Demote all values to stack slots
- `-sroa`: Scalar Replacement of Aggregates
- `-sccp`: Sparse Conditional Constant Propagation
- `-simplifycfg`: Simplify the CFG
- `-sink`: Code sinking
- `-strip`: Strip all symbols from a module
- `-strip-dead-debug-info`: Strip debug info for unused symbols
- `-strip-dead-prototypes`: Strip Unused Function Prototypes
- `-strip-debug-declare`: Strip all `llvm.dbg.declare` intrinsics
- `-strip-nondebug`: Strip all symbols, except dbg symbols, from a module
- `-tailcallelim`: Tail Call Elimination

LLVM opt example: loop related

```
static long Test(long x, bool condition)
{
    for (int i = 100; i < 1000; i += 5) {
        if (condition)
            x++;
    }
    return x;
}
```

RuyJIT

```
       mov      eax, 10
       movzx    rdx, dl
G_M14331_IG03:
       test     edx, edx
       je       SHORT G_M14331_IG05
G_M14331_IG04:
       inc      rcx
G_M14331_IG05:
       add      eax, 2
       cmp      eax, 0x186A0
       jl       SHORT G_M14331_IG03
G_M14331_IG06:
       mov      rax, rcx
G_M14331_IG07:
       ret
```

LLVM

```
mov      rax, rdi
lea      rdx, [rdi+450]
test     sil, sil
cmovne   rax, rdx
ret
```

LLVM opt example: ffast-math


```
float Max(float a, float b) => a > b ? a : b;
```

CoreCLR

```
G_M60875_IG01:
       vzeroupper

G_M60875_IG02:
       vucomiss xmm1, xmm2
       ja       SHORT G_M60875_IG04
       vmovaps  xmm0, xmm2

G_M60875_IG03:
       ret

G_M60875_IG04:
       vmovaps  xmm0, xmm1

G_M60875_IG05:
       ret
```

Mono-LLVM (--ffast-math)

```
vmaxss xmm0, xmm0, xmm1
ret
```

LLVM opt example: ffast-math

1
```
float result = (x / 10) + 1 + 1;
```

2
```
float result = (x / 10) + 2;
```

3
```
float result = (x * 0.1) + 2;
```

4
```
float result = Math.FusedMultiplyAdd(x, 0.1, 2);
```

CoreCLR

```
vdivsd xmm0, xmm0, QWORD PTR .LC0[rip]
vaddsd xmm0, xmm0, xmm1
vaddsd xmm0, xmm0, xmm1
```

Mono-LLVM (--ffast-math)

```
; x * 0.1 + 2
vfmadd132sd xmm0, xmm1, QWORD PTR .LC0[rip]
```

Math/MathF are LLVM intrinsics!


```
Math.Abs(X)  * Math.Abs(X)   => X * X
Math.Sin(X)  / Math.Cos(X)   => Math.Tan(X)
Math.Sqrt(X) * Math.Sqrt(X)  => X
Math.Sin(-X)                 => -Math.Sin(X)

MathF.Pow(X, 0.5f)           => MathF.Sqrt(X)
MathF.Pow(X, 2)              => X * X
MathF.Pow(X, 4)              => (X * X) * (X * X)
```

LLVM opt example: ffast-math

Benchmark	Fast-Math	No Fast-Math
Benchstone.BenchF.BenchMk2.Test	1.00	2.25
Benchstone.BenchF.BenchMrk.Test	1.00	2.37
Benchstone.BenchF.Lorenz.Test	1.00	1.85
Burgers.Test0	1.00	1.39
Burgers.Test1	1.00	2.30
Burgers.Test2	1.00	2.32
System.MathBenchmarks.Double.Cosh	1.00	1.50

Performance bottlenecks:

- Slow heap allocations (especially for arrays)
- Missing C# specific optimizations, e.g. devirtualization
- System.Runtime.Intrinsics.* (some of them are already implemented, including Arm64's ArmBase)
- Some reflection stuff is slow (e.g. Activator.CreateInstance, etc)
- Vector<T> is 128 bit only

How can I try mono-vm today


```
git clone git@github.com:mono/mono.git
cd mono/netcore
./build.sh -c Release -llvm
make run-sample
```

Q&A

Twitter: EgorBo