How difficult is to get a JIT right?

guillermo.polito@inria.fr

Guillermo Polito - ESUG'24

Quick About Me: Guille

guillermo.polito@inria.fr
@guillep

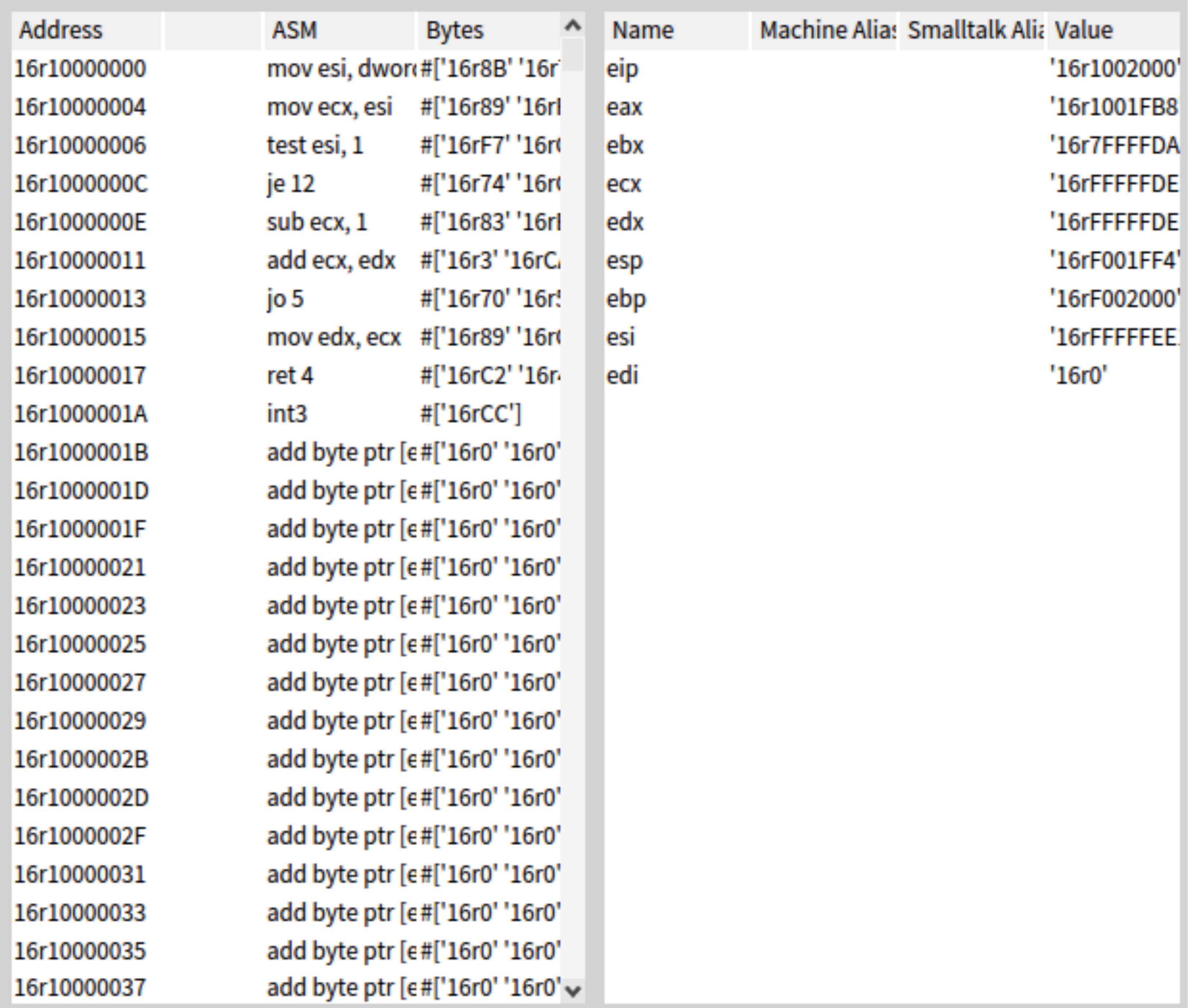
- Pronounced *giʃe* (guichet in FR, ~ghisheh in EN?)
- **Now:** Researcher at Inria - Lille
- Pharo Contributor since ~2010

- **Keywords:** compilers, testing, test generation
- **Interests:** tooling, benchmarking, 日本語, board games, batman, concurrency

If any of that interests you, come talk to me!

Debugging Assembly Code

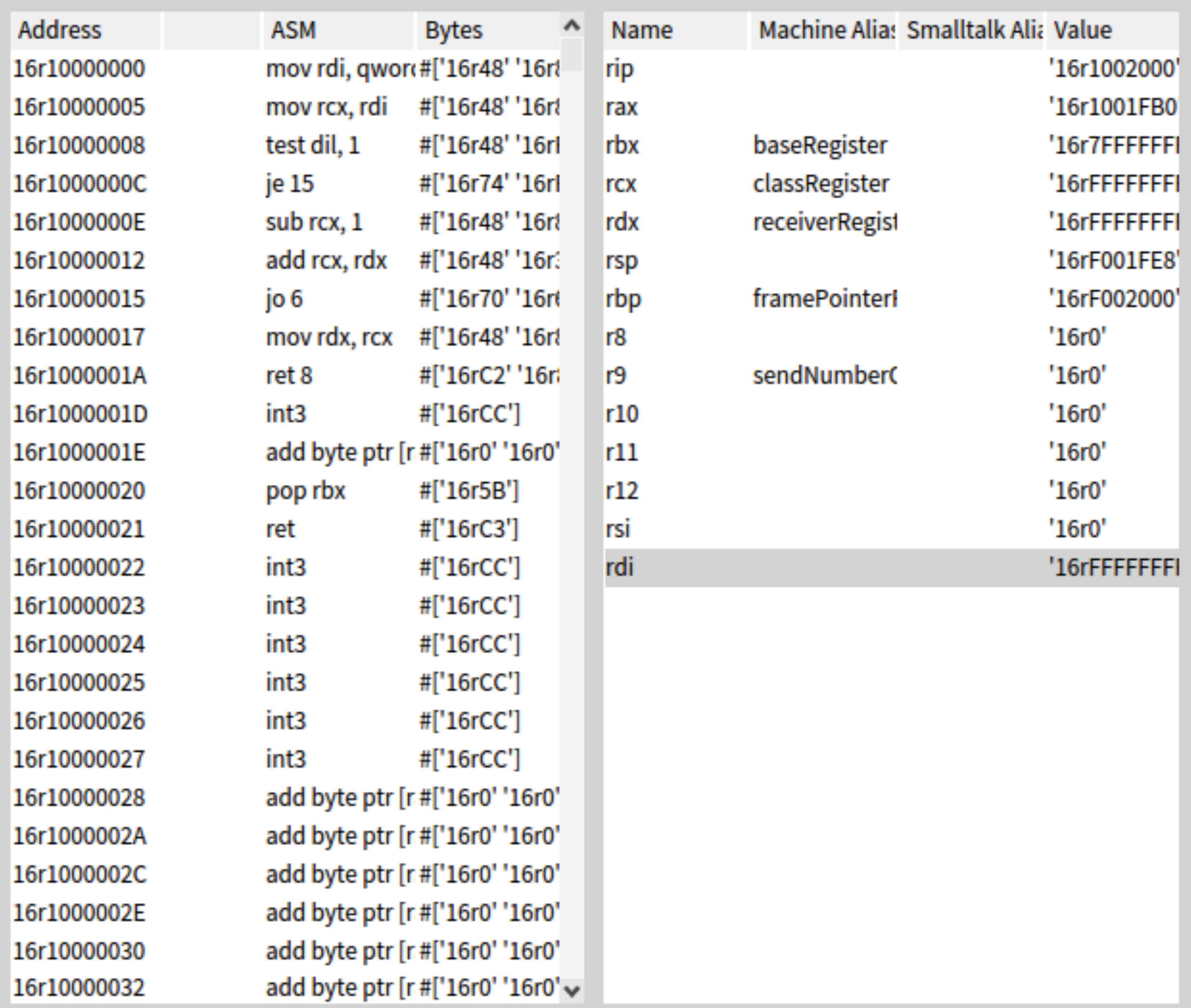
IA32



Address	ASM	Bytes
16r10000000	mov esi, dwor	#['16r8B' '16r
16r10000004	mov ecx, esi	#['16r89' '16r
16r10000006	test esi, 1	#['16rF7' '16r
16r1000000C	je 12	#['16r74' '16r
16r1000000E	sub ecx, 1	#['16r83' '16r
16r10000011	add ecx, edx	#['16r3' '16rC.
16r10000013	jo 5	#['16r70' '16r
16r10000015	mov edx, ecx	#['16r89' '16r
16r10000017	ret 4	#['16rC2' '16r
16r1000001A	int3	#['16rCC']
16r1000001B	add byte ptr [e	#['16r0' '16r0'
16r1000001D	add byte ptr [e	#['16r0' '16r0'
16r1000001F	add byte ptr [e	#['16r0' '16r0'
16r10000021	add byte ptr [e	#['16r0' '16r0'
16r10000023	add byte ptr [e	#['16r0' '16r0'
16r10000025	add byte ptr [e	#['16r0' '16r0'
16r10000027	add byte ptr [e	#['16r0' '16r0'
16r10000029	add byte ptr [e	#['16r0' '16r0'
16r1000002B	add byte ptr [e	#['16r0' '16r0'
16r1000002D	add byte ptr [e	#['16r0' '16r0'
16r1000002F	add byte ptr [e	#['16r0' '16r0'
16r10000031	add byte ptr [e	#['16r0' '16r0'
16r10000033	add byte ptr [e	#['16r0' '16r0'
16r10000035	add byte ptr [e	#['16r0' '16r0'
16r10000037	add byte ptr [e	#['16r0' '16r0'

Name	Machine Alia:	Smalltalk Ali:	Value
eip			'16r1002000'
eax			'16r1001FB8
ebx			'16r7FFFFDA
ecx			'16rFFFFFDE
edx			'16rFFFFFDE
esp			'16rF001FF4'
ebp			'16rF002000'
esi			'16rFFFFFEE.
edi			'16r0'

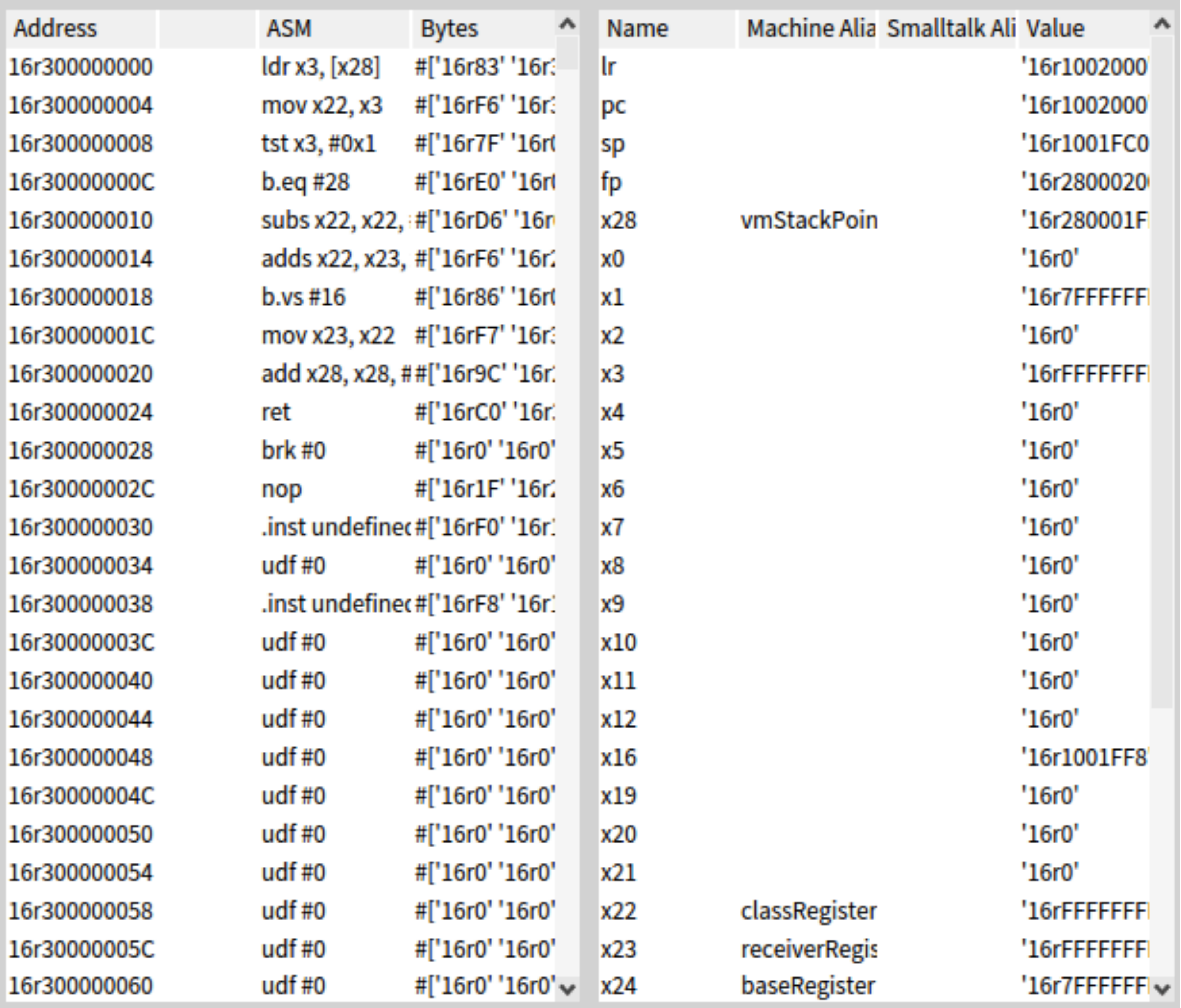
X64



Address	ASM	Bytes
16r10000000	mov rdi, qwor	#['16r48' '16r
16r10000005	mov rcx, rdi	#['16r48' '16r
16r10000008	test dil, 1	#['16r48' '16r
16r1000000C	je 15	#['16r74' '16r
16r1000000E	sub rcx, 1	#['16r48' '16r
16r10000012	add rcx, rdx	#['16r48' '16r
16r10000015	jo 6	#['16r70' '16r
16r10000017	mov rdx, rcx	#['16r48' '16r
16r1000001A	ret 8	#['16rC2' '16r
16r1000001D	int3	#['16rCC']
16r1000001E	add byte ptr [r	#['16r0' '16r0'
16r10000020	pop rbx	#['16r5B']
16r10000021	ret	#['16rC3']
16r10000022	int3	#['16rCC']
16r10000023	int3	#['16rCC']
16r10000024	int3	#['16rCC']
16r10000025	int3	#['16rCC']
16r10000026	int3	#['16rCC']
16r10000027	int3	#['16rCC']
16r10000028	add byte ptr [r	#['16r0' '16r0'
16r1000002A	add byte ptr [r	#['16r0' '16r0'
16r1000002C	add byte ptr [r	#['16r0' '16r0'
16r1000002E	add byte ptr [r	#['16r0' '16r0'
16r10000030	add byte ptr [r	#['16r0' '16r0'
16r10000032	add byte ptr [r	#['16r0' '16r0'

Name	Machine Alia:	Smalltalk Ali:	Value
rip			'16r1002000'
rax			'16r1001FB0
rbx	baseRegister		'16r7FFFFFF
rcx	classRegister		'16rFFFFFFF
rdx	receiverRegist		'16rFFFFFFF
rsp			'16rF001FE8'
rbp	framePointer		'16rF002000'
r8			'16r0'
r9	sendNumber(		'16r0'
r10			'16r0'
r11			'16r0'
r12			'16r0'
rsi			'16r0'
rdi			'16rFFFFFFF

AARCH64



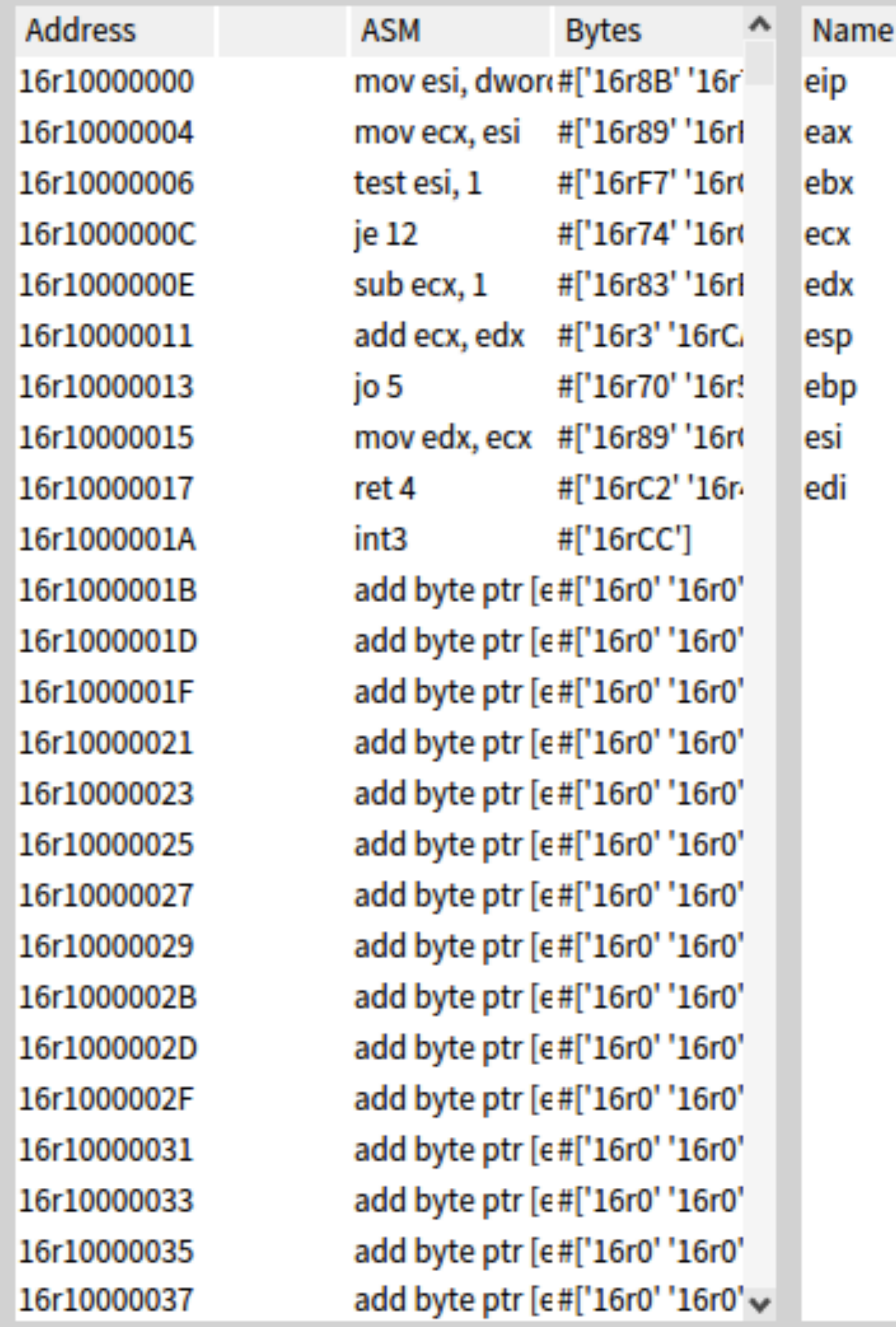
Address	ASM	Bytes
16r300000000	ldr x3, [x28]	#['16r83' '16r
16r300000004	mov x22, x3	#['16rF6' '16r
16r300000008	tst x3, #0x1	#['16r7F' '16r
16r30000000C	b.eq #28	#['16rE0' '16r
16r300000010	subs x22, x22,	#['16rD6' '16r
16r300000014	adds x22, x23,	#['16rF6' '16r
16r300000018	b.vs #16	#['16r86' '16r
16r30000001C	mov x23, x22	#['16rF7' '16r
16r300000020	add x28, x28, #	#['16r9C' '16r
16r300000024	ret	#['16rC0' '16r
16r300000028	brk #0	#['16r0' '16r0'
16r30000002C	nop	#['16r1F' '16r
16r300000030	.inst undefinec	#['16rF0' '16r
16r300000034	udf #0	#['16r0' '16r0'
16r300000038	.inst undefinec	#['16rF8' '16r
16r30000003C	udf #0	#['16r0' '16r0'
16r300000040	udf #0	#['16r0' '16r0'
16r300000044	udf #0	#['16r0' '16r0'
16r300000048	udf #0	#['16r0' '16r0'
16r30000004C	udf #0	#['16r0' '16r0'
16r300000050	udf #0	#['16r0' '16r0'
16r300000054	udf #0	#['16r0' '16r0'
16r300000058	udf #0	#['16r0' '16r0'
16r30000005C	udf #0	#['16r0' '16r0'
16r300000060	udf #0	#['16r0' '16r0'

Name	Machine Alia	Smalltalk Ali	Value
lr			'16r1002000
pc			'16r1002000
sp			'16r1001FC0
fp			'16r2800020
x28	vmStackPoin		'16r280001F
x0			'16r0'
x1			'16r7FFFFFF
x2			'16r0'
x3			'16rFFFFFFF
x4			'16r0'
x5			'16r0'
x6			'16r0'
x7			'16r0'
x8			'16r0'
x9			'16r0'
x10			'16r0'
x11			'16r0'
x12			'16r0'
x16			'16r1001FF8
x19			'16r0'
x20			'16r0'
x21			'16r0'
x22	classRegister		'16rFFFFFFF
x23	receiverRegis		'16rFFFFFFF
x24	baseRegister		'16r7FFFFFF

Debugging Assembly Code

Without looking at it

IA32



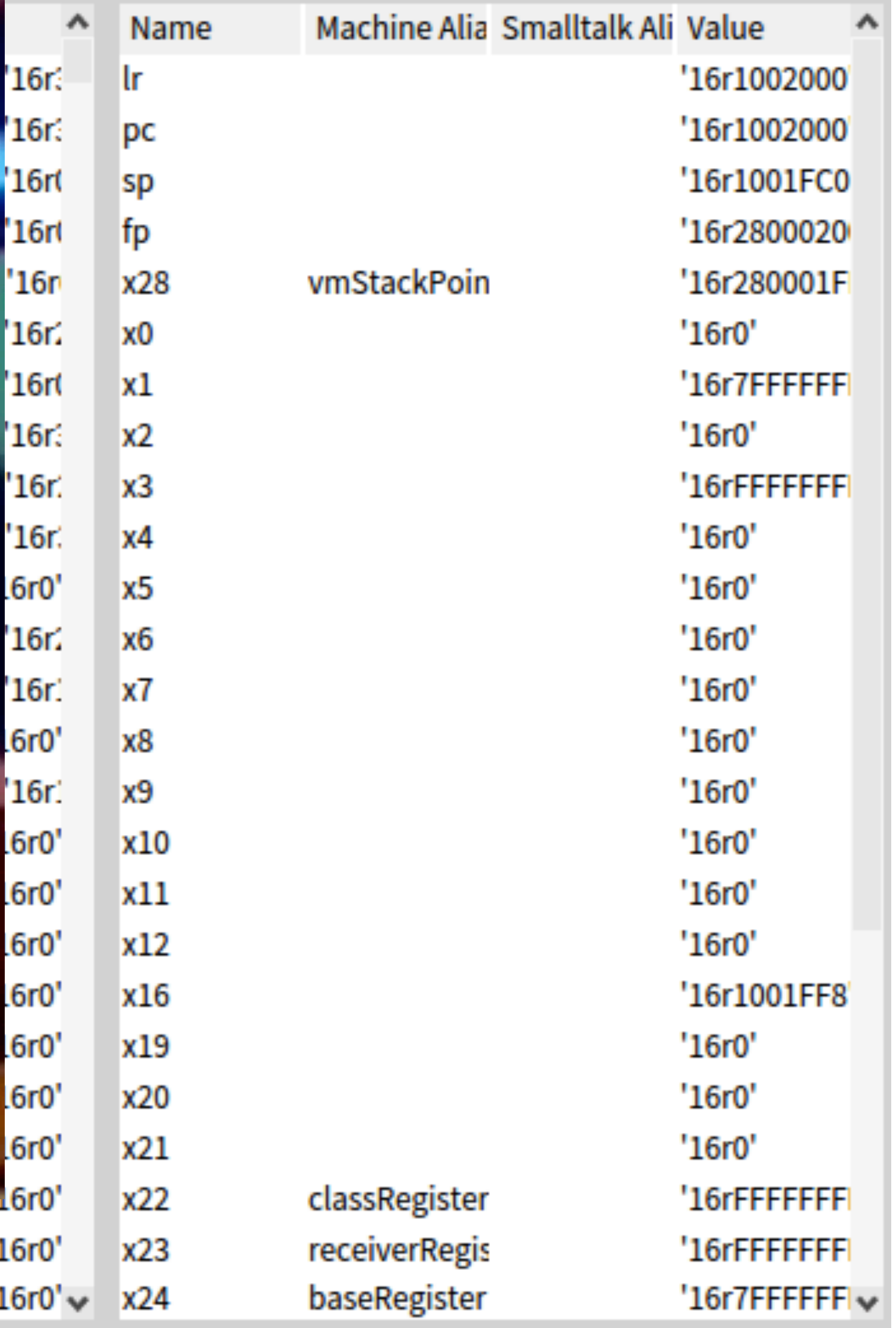
Address	ASM	Bytes
16r10000000	mov esi, dwor	#['16r8B' '16r
16r10000004	mov ecx, esi	#['16r89' '16r
16r10000006	test esi, 1	#['16rF7' '16r
16r1000000C	je 12	#['16r74' '16r
16r1000000E	sub ecx, 1	#['16r83' '16r
16r10000011	add ecx, edx	#['16r3' '16rC
16r10000013	jo 5	#['16r70' '16r
16r10000015	mov edx, ecx	#['16r89' '16r
16r10000017	ret 4	#['16rC2' '16r
16r1000001A	int3	#['16rCC']
16r1000001B	add byte ptr [e	#['16r0' '16r0'
16r1000001D	add byte ptr [e	#['16r0' '16r0'
16r1000001F	add byte ptr [e	#['16r0' '16r0'
16r10000021	add byte ptr [e	#['16r0' '16r0'
16r10000023	add byte ptr [e	#['16r0' '16r0'
16r10000025	add byte ptr [e	#['16r0' '16r0'
16r10000027	add byte ptr [e	#['16r0' '16r0'
16r10000029	add byte ptr [e	#['16r0' '16r0'
16r1000002B	add byte ptr [e	#['16r0' '16r0'
16r1000002D	add byte ptr [e	#['16r0' '16r0'
16r1000002F	add byte ptr [e	#['16r0' '16r0'
16r10000031	add byte ptr [e	#['16r0' '16r0'
16r10000033	add byte ptr [e	#['16r0' '16r0'
16r10000035	add byte ptr [e	#['16r0' '16r0'
16r10000037	add byte ptr [e	#['16r0' '16r0'

Name
eip
eax
ebx
ecx
edx
esp
ebp
esi
edi

16r10000030	add byte ptr [r	#['16r0' '16r0'
16r10000032	add byte ptr [r	#['16r0' '16r0'

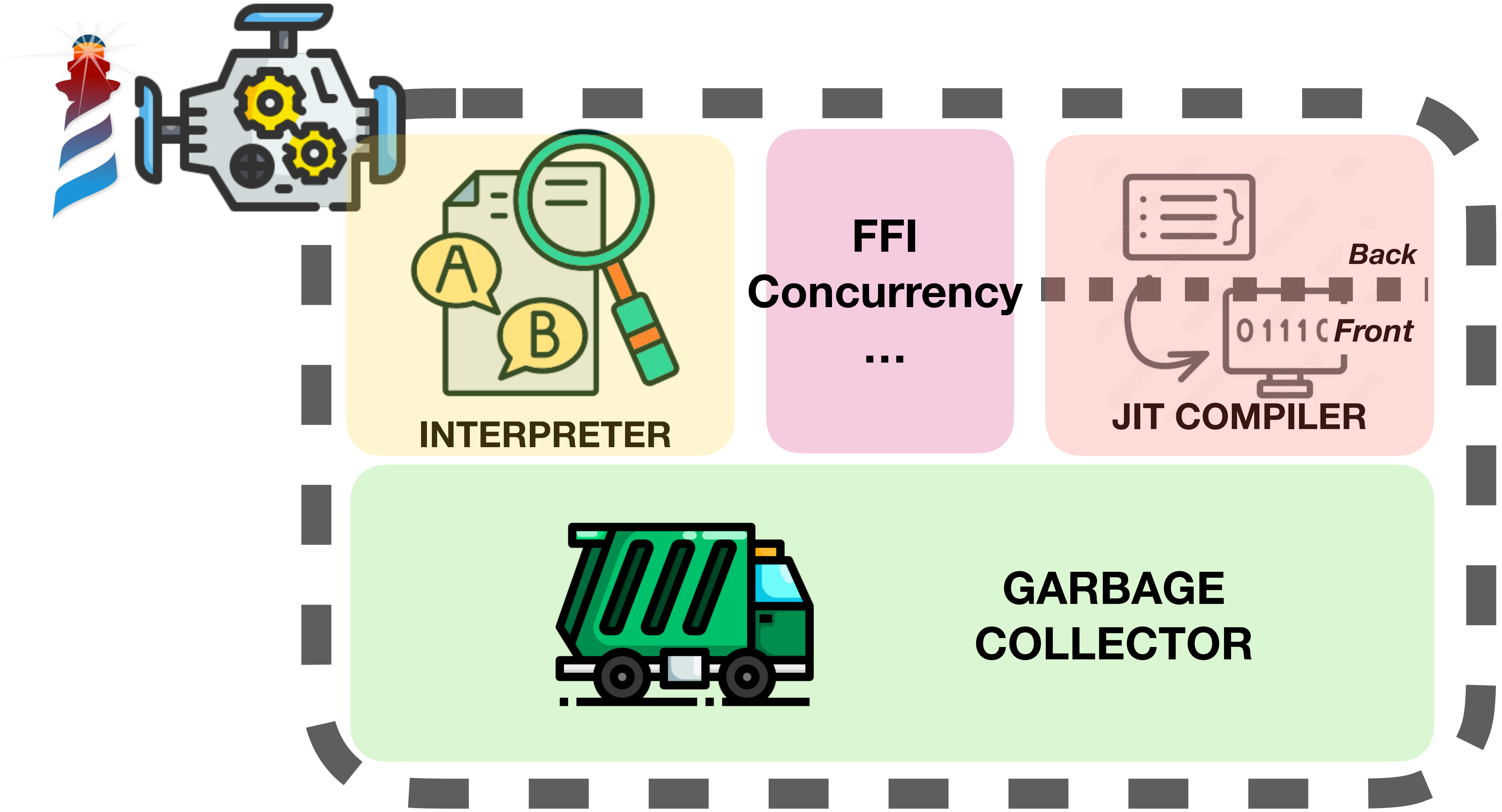
16r3000005C	udf #0	#['16r0' '16r0'
16r300000060	udf #0	#['16r0' '16r0'

ARCH64

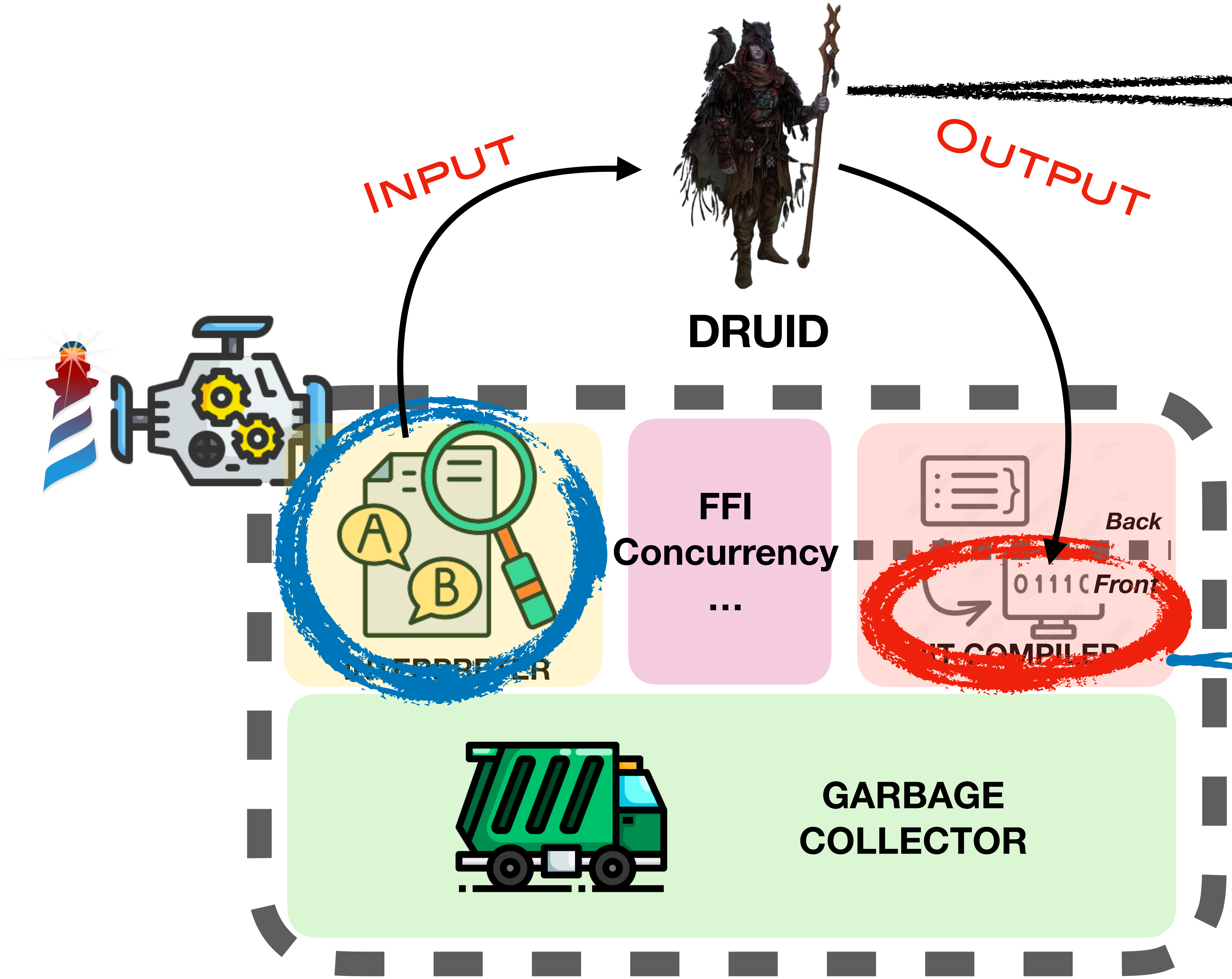


Name	Machine Alia	Smalltalk Ali	Value
lr			'16r1002000
pc			'16r1002000
sp			'16r1001FC0
fp			'16r2800020
x28	vmStackPoin		'16r280001F
x0			'16r0'
x1			'16r7FFFFFF
x2			'16r0'
x3			'16rFFFFFFF
x4			'16r0'
x5			'16r0'
x6			'16r0'
x7			'16r0'
x8			'16r0'
x9			'16r0'
x10			'16r0'
x11			'16r0'
x12			'16r0'
x16			'16r1001FF8
x19			'16r0'
x20			'16r0'
x21			'16r0'
x22	classRegister		'16rFFFFFFF
x23	receiverRegis		'16rFFFFFFF
x24	baseRegister		'16r7FFFFFF

The Pharo VM


INTERPRETER
FFI
Concurrency
...
Back
0111(
Front
JIT COMPILER
GARBAGE
COLLECTOR

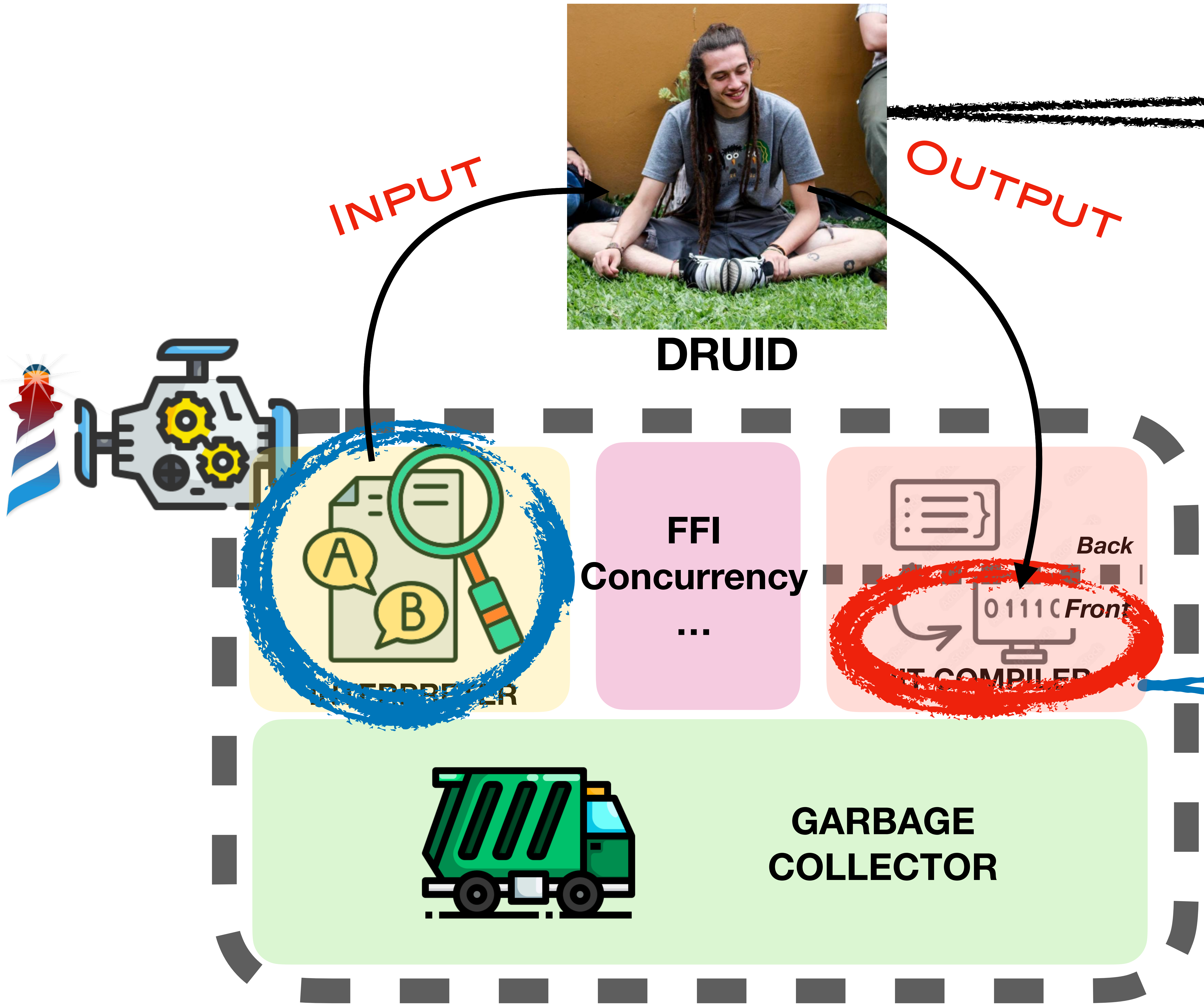
Context: Druid JIT compiler generation



I am performed at VM building time (AoT)

VM developers avoid to write and maintain the *frontend* of the JIT compiler code (language dependencies)

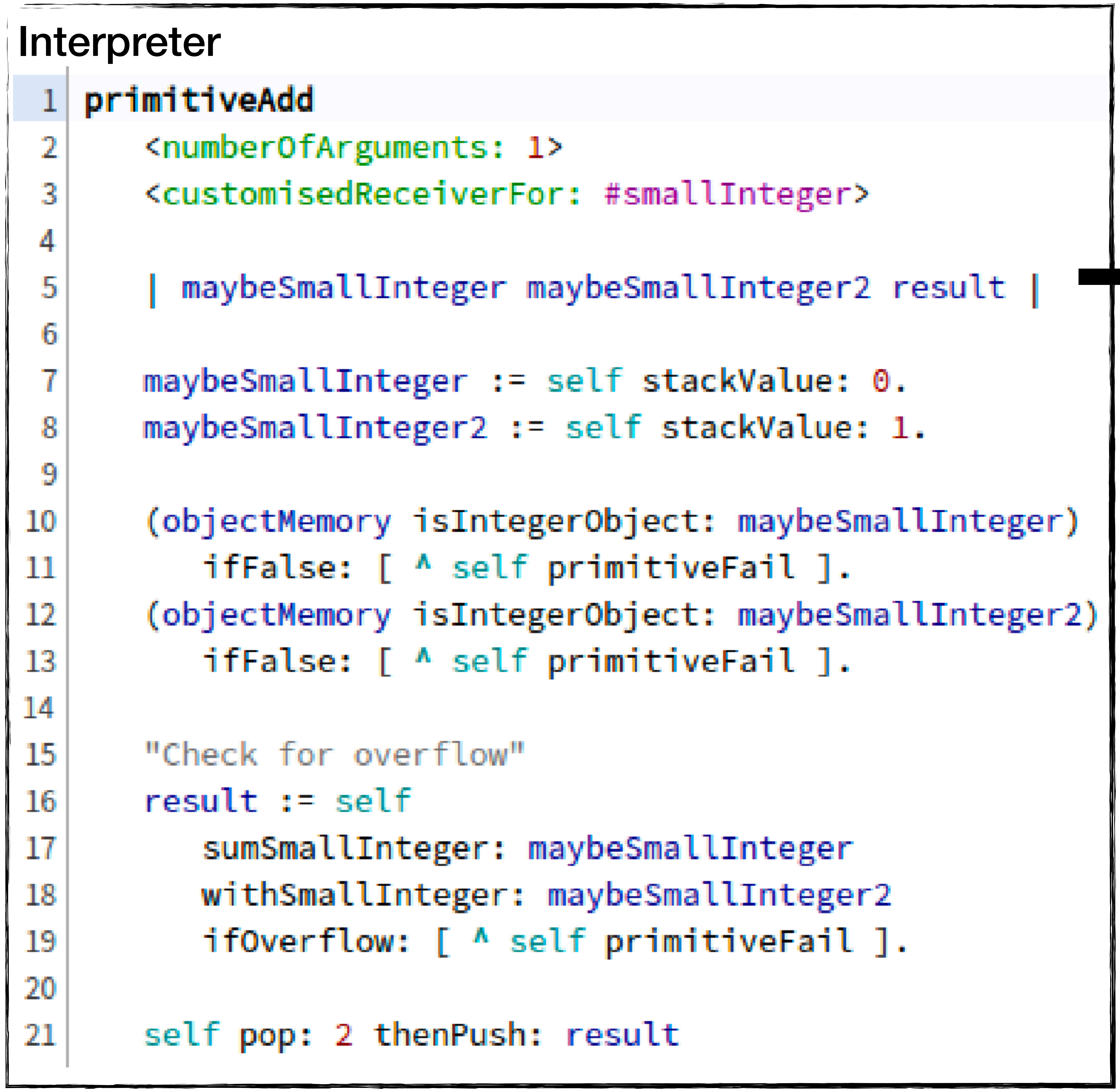
Context: Druid JIT compiler generation



I am performed at VM building time (AoT)

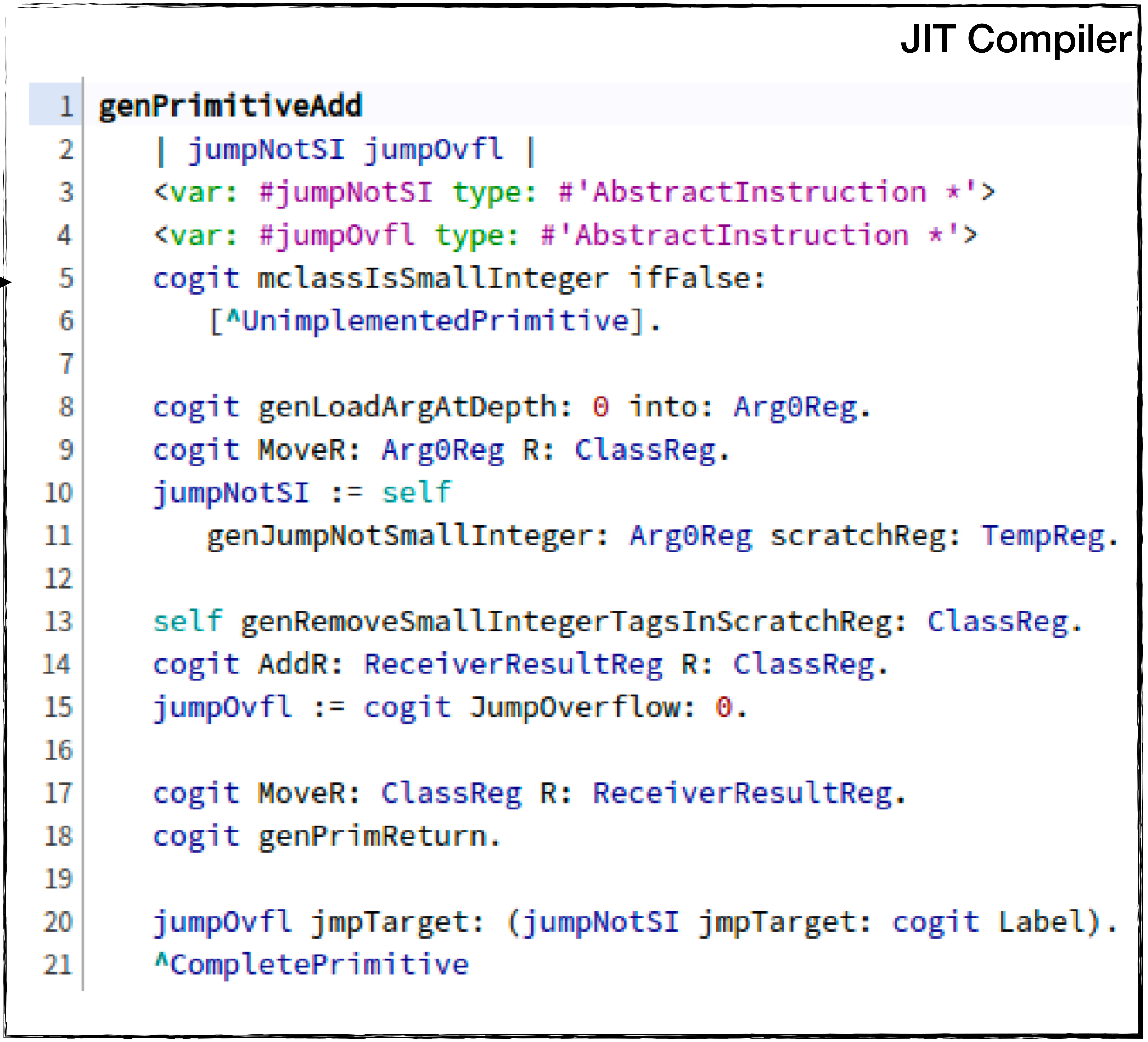
VM developers avoid to write and maintain the *frontend* of the JIT compiler code (language dependencies)

Druid by example: the addition primitive



Interpreter

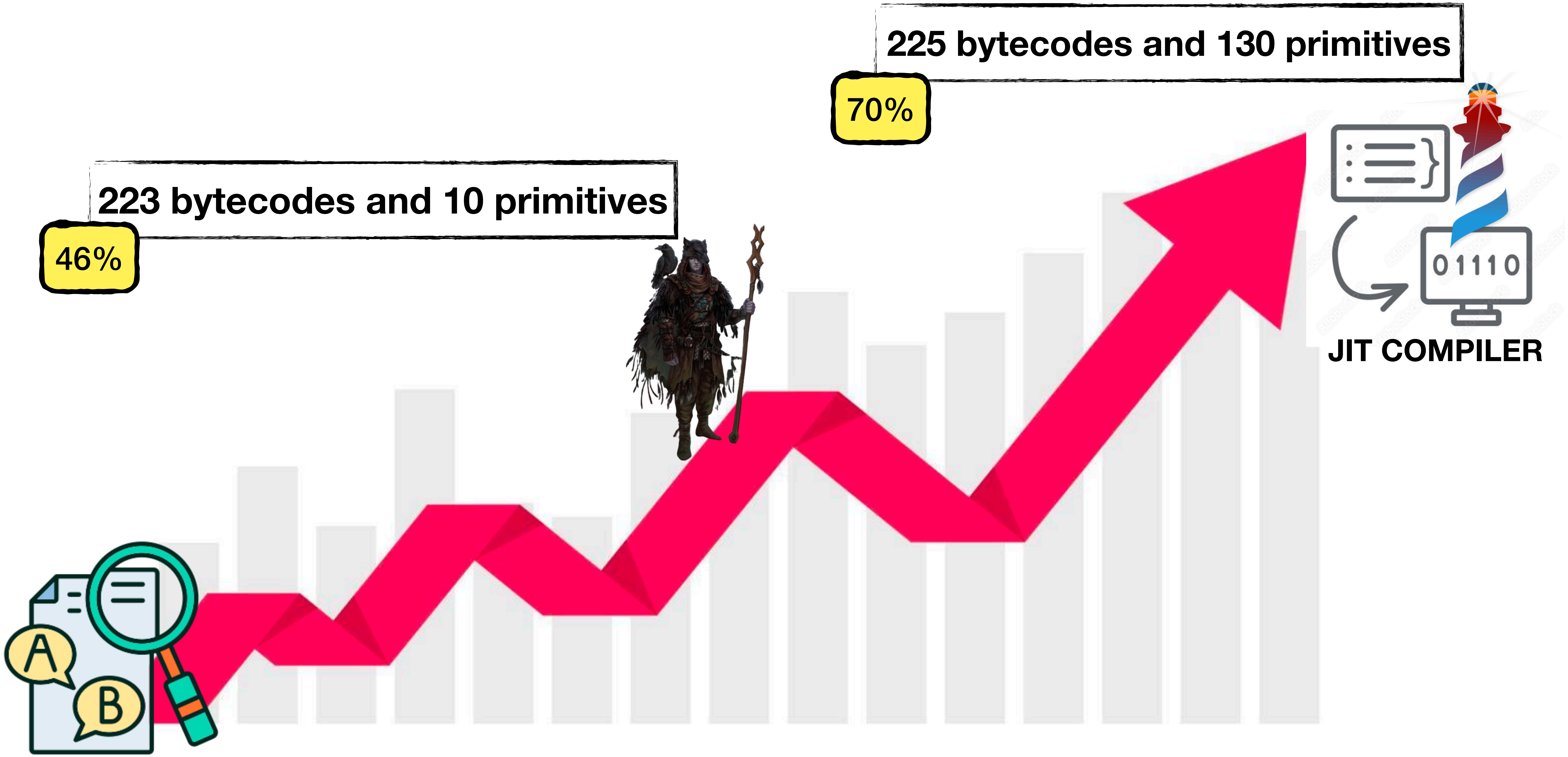
```
primitiveAdd
    <numberOfArguments: 1>
    <customisedReceiverFor: #smallInteger>

    | maybeSmallInteger maybeSmallInteger2 result |

    maybeSmallInteger := self stackValue: 0.
    maybeSmallInteger2 := self stackValue: 1.

    (objectMemory isIntegerObject: maybeSmallInteger)
        ifFalse: [ ^ self primitiveFail ].
    (objectMemory isIntegerObject: maybeSmallInteger2)
        ifFalse: [ ^ self primitiveFail ].

    "Check for overflow"
    result := self
        sumSmallInteger: maybeSmallInteger
        withSmallInteger: maybeSmallInteger2
        ifOverflow: [ ^ self primitiveFail ].

    self pop: 2 thenPush: result
```



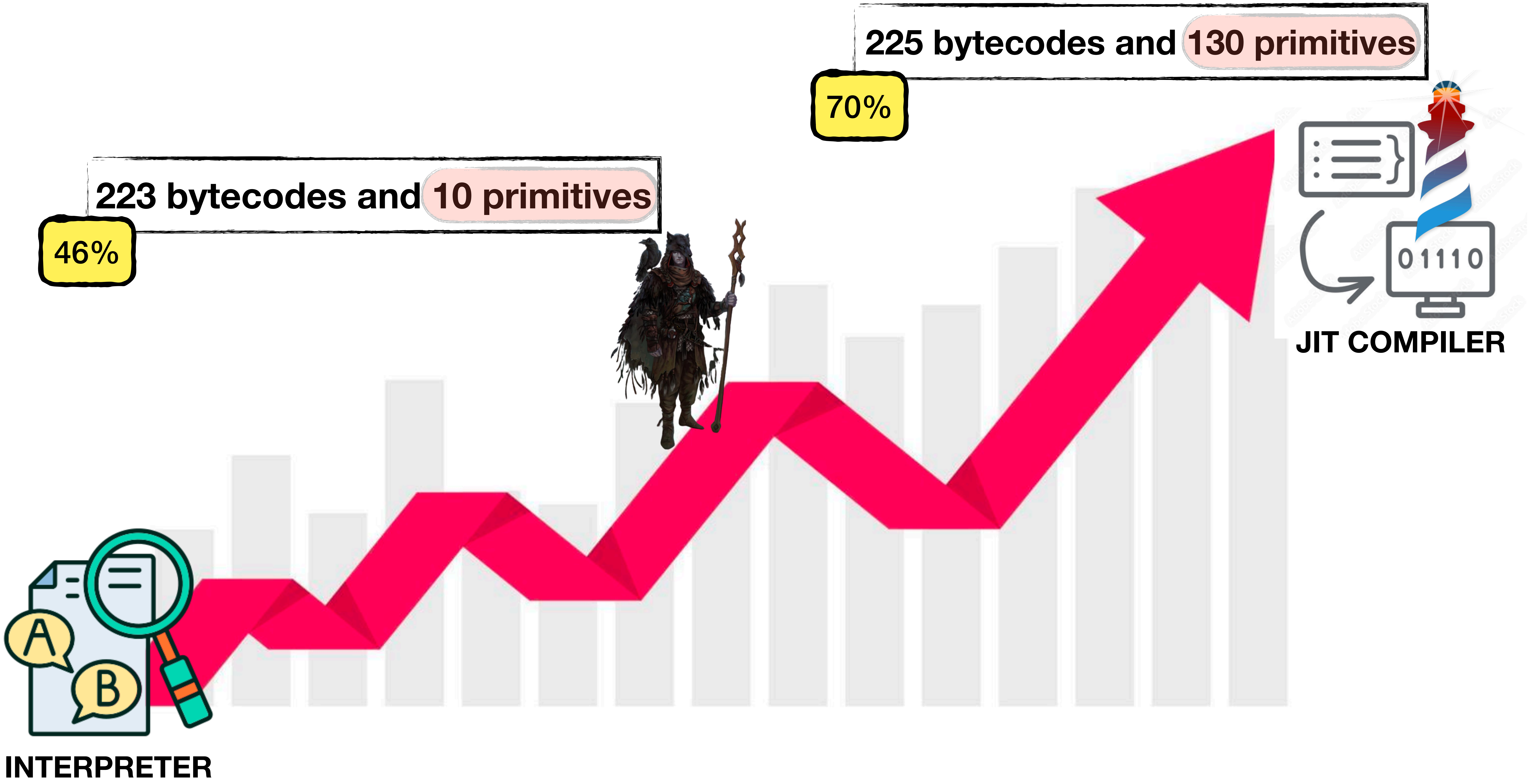
JIT Compiler

```
genPrimitiveAdd
    | jumpNotSI jumpOvfl |
    <var: #jumpNotSI type: #'AbstractInstruction *'>
    <var: #jumpOvfl type: #'AbstractInstruction *'>
    cogit mclassIsSmallInteger ifFalse:
        [^UnimplementedPrimitive].

    cogit genLoadArgAtDepth: 0 into: Arg0Reg.
    cogit MoveR: Arg0Reg R: ClassReg.
    jumpNotSI := self
        genJumpNotSmallInteger: Arg0Reg scratchReg: TempReg.

    self genRemoveSmallIntegerTagsInScratchReg: ClassReg.
    cogit AddR: ReceiverResultReg R: ClassReg.
    jumpOvfl := cogit JumpOverflow: 0.

    cogit MoveR: ClassReg R: ReceiverResultReg.
    cogit genPrimReturn.

    jumpOvfl jmpTarget: (jumpNotSI jmpTarget: cogit Label).
    ^CompletePrimitive
```

A Couple of Months Ago


225 bytecodes and 130 primitives
70%
223 bytecodes and 10 primitives
46%
01110
JIT COMPILER
INTERPRETER

A Couple of Months Ago



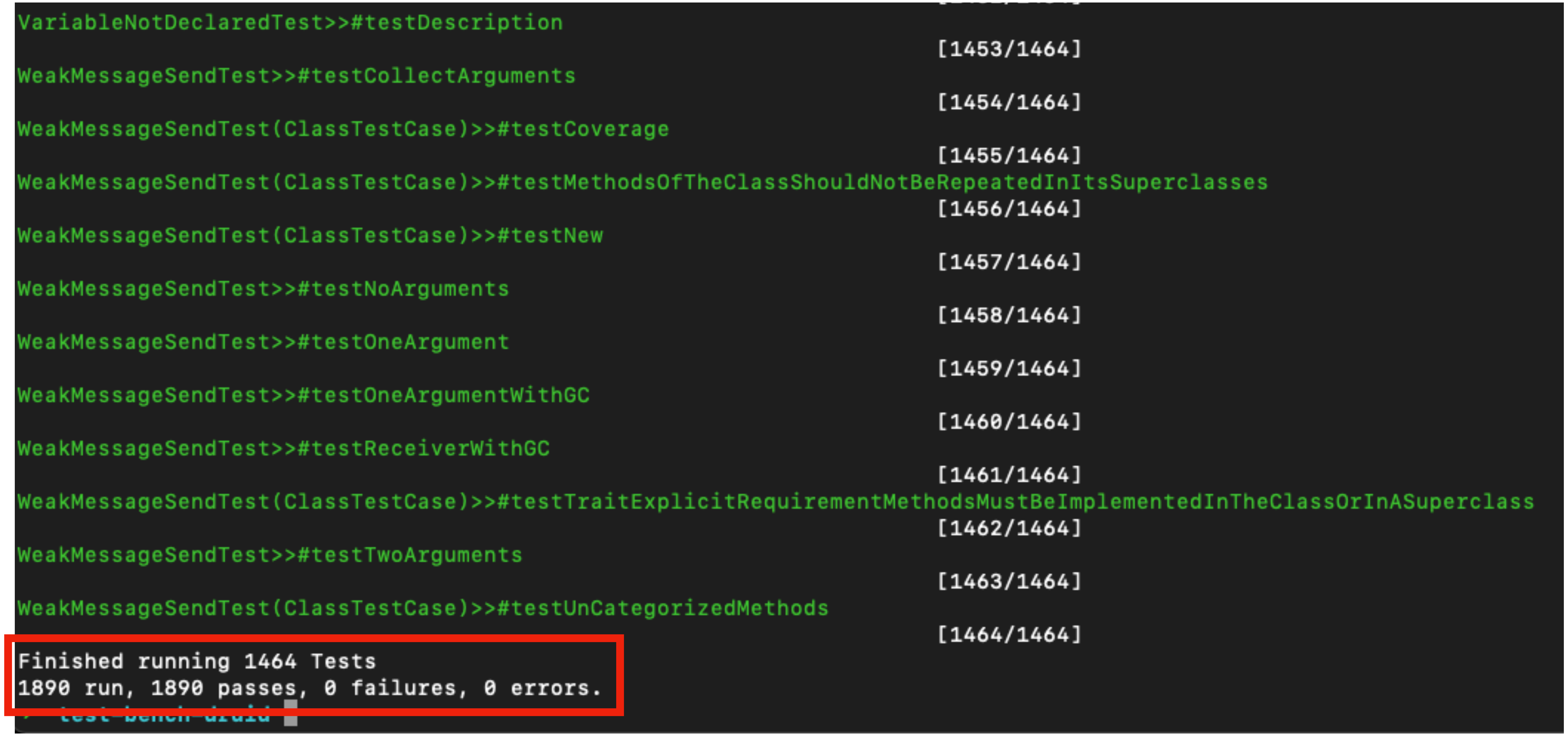
223 bytecodes and 10 primitives

46%

225 bytecodes and 130 primitives

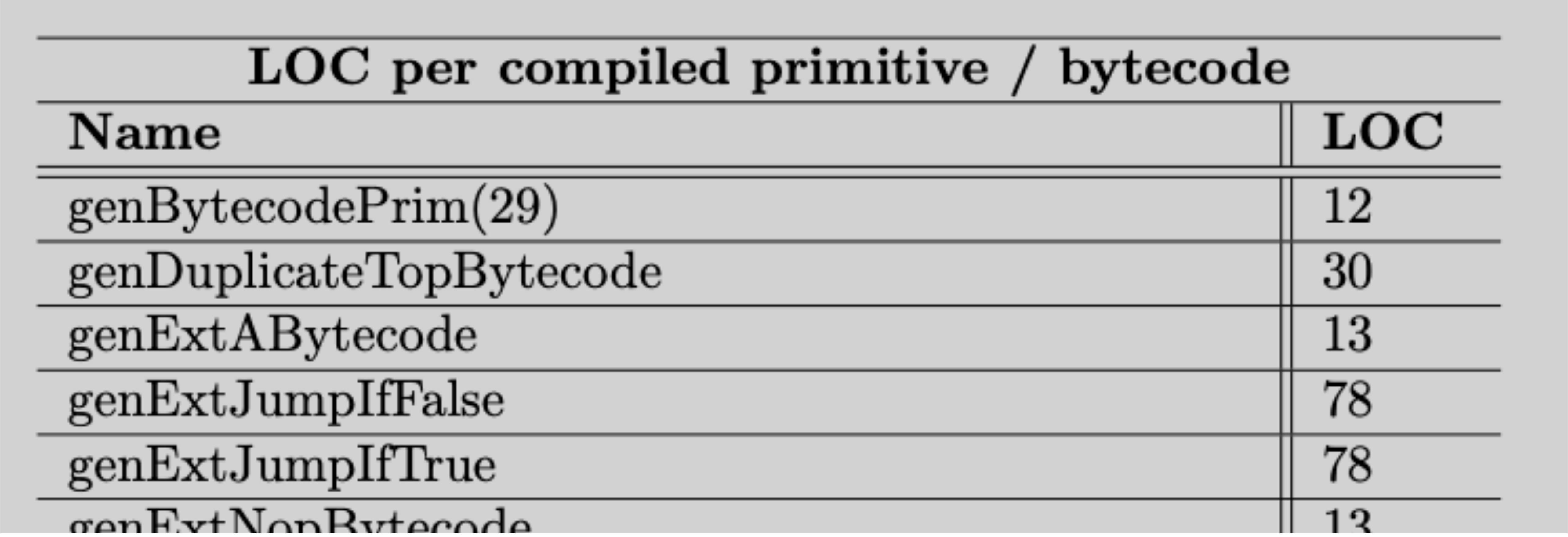
70%

Generated JIT-Compiler



```
                                                                     [1452/1464]
VariableNotDeclaredTest>>#testDescription
                                                                     [1453/1464]
WeakMessageSendTest>>#testCollectArguments
                                                                     [1454/1464]
WeakMessageSendTest(ClassTestCase)>>#testCoverage
                                                                     [1455/1464]
WeakMessageSendTest(ClassTestCase)>>#testMethodsOfTheClassShouldNotBeRepeatedInItsSuperclasses
                                                                     [1456/1464]
WeakMessageSendTest(ClassTestCase)>>#testNew
                                                                     [1457/1464]
WeakMessageSendTest>>#testNoArguments
                                                                     [1458/1464]
WeakMessageSendTest>>#testOneArgument
                                                                     [1459/1464]
WeakMessageSendTest>>#testOneArgumentWithGC
                                                                     [1460/1464]
WeakMessageSendTest>>#testReceiverWithGC
                                                                     [1461/1464]
WeakMessageSendTest(ClassTestCase)>>#testTraitExplicitRequirementMethodsMustBeImplementedInTheClassOrInASuperclass
                                                                     [1462/1464]
WeakMessageSendTest>>#testTwoArguments
                                                                     [1463/1464]
WeakMessageSendTest(ClassTestCase)>>#testUnCategorizedMethods
                                                                     [1464/1464]
Finished running 1464 Tests
1890 run, 1890 passes, 0 failures, 0 errors.
> test-bench-druid
```

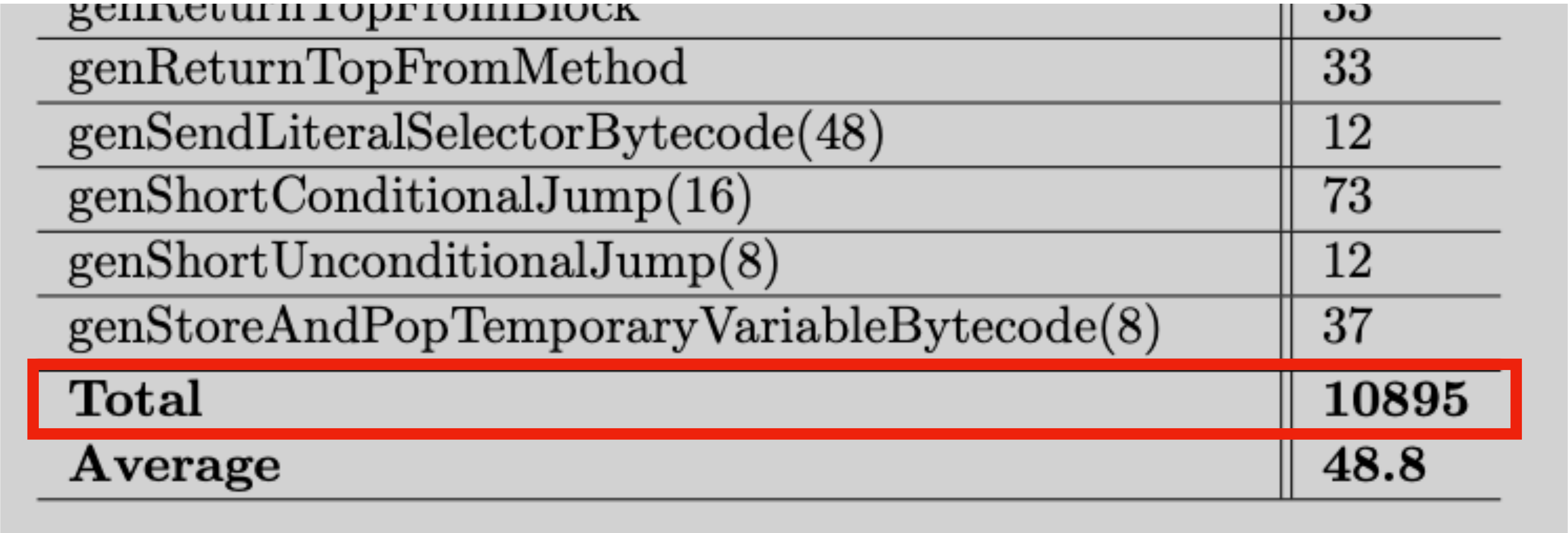
Generated JIT-Compiler



LOC per compiled primitive / bytecode

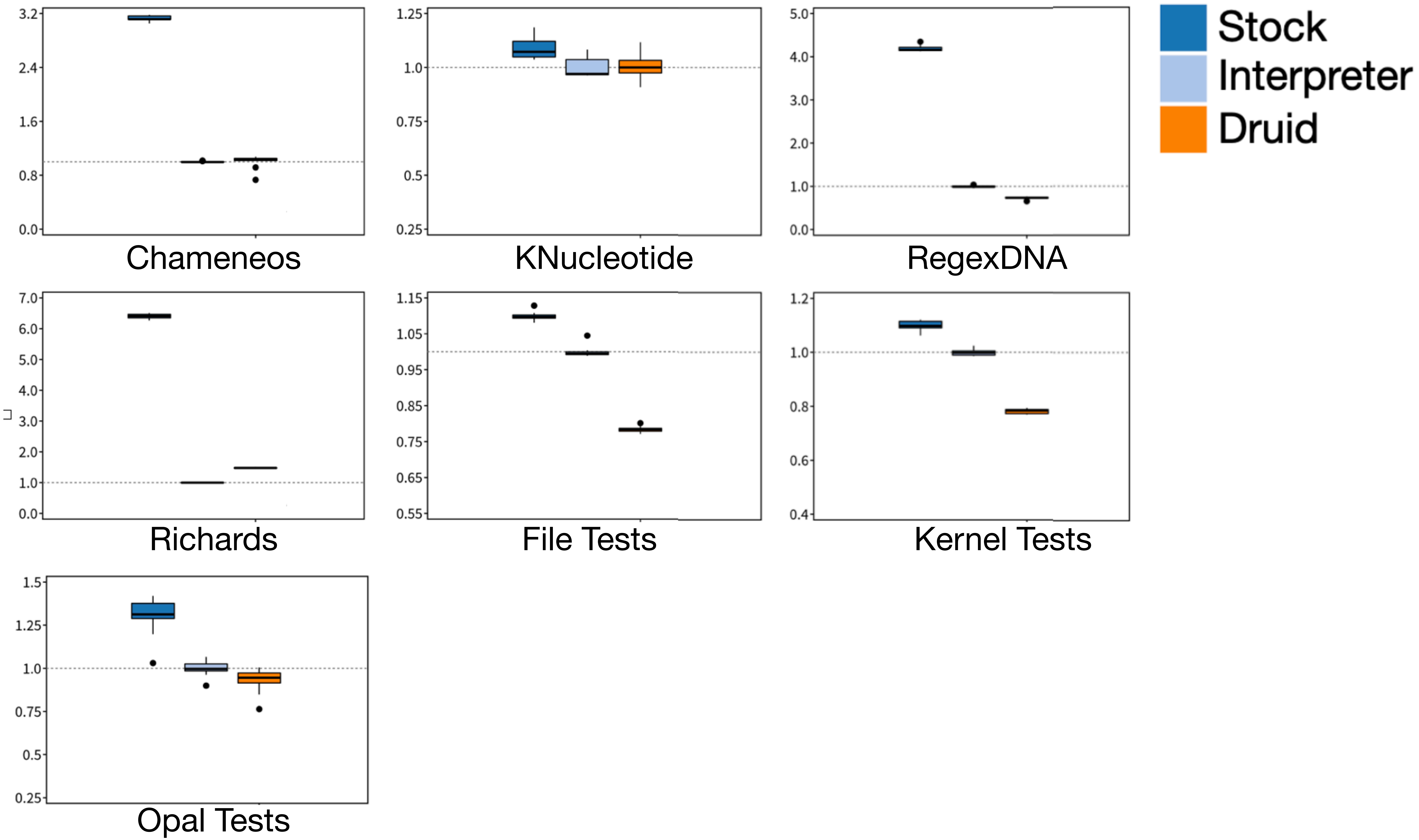
Name	**LOC**
genBytecodePrim(29)	12
genDuplicateTopBytecode	30
genExtABytecode	13
genExtJumpIfFalse	78
genExtJumpIfTrue	78
genExtNopBytecode	13

...



genReturnTopFromBlock	33
genReturnTopFromMethod	33
genSendLiteralSelectorBytecode(48)	12
genShortConditionalJump(16)	73
genShortUnconditionalJump(8)	12
genStoreAndPopTemporaryVariableBytecode(8)	37
Total	**10895**
Average	**48.8**

Some Initial benchmarks



Chameneos

KNucleotide

RegexDNA

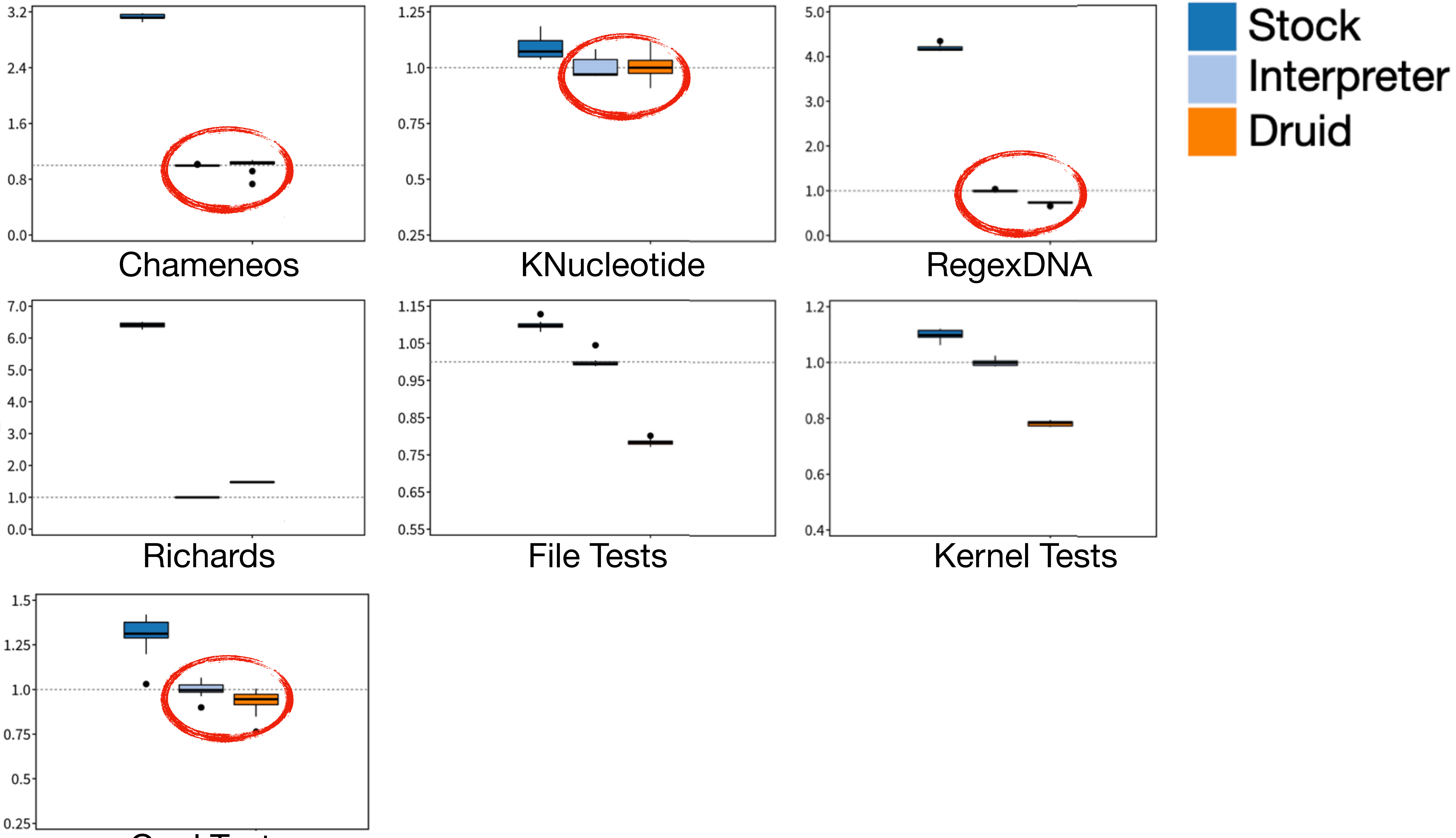
Richards

File Tests

Kernel Tests

Opal Tests

À la par with the interpreter



Chameneos

KNucleotide

RegexDNA

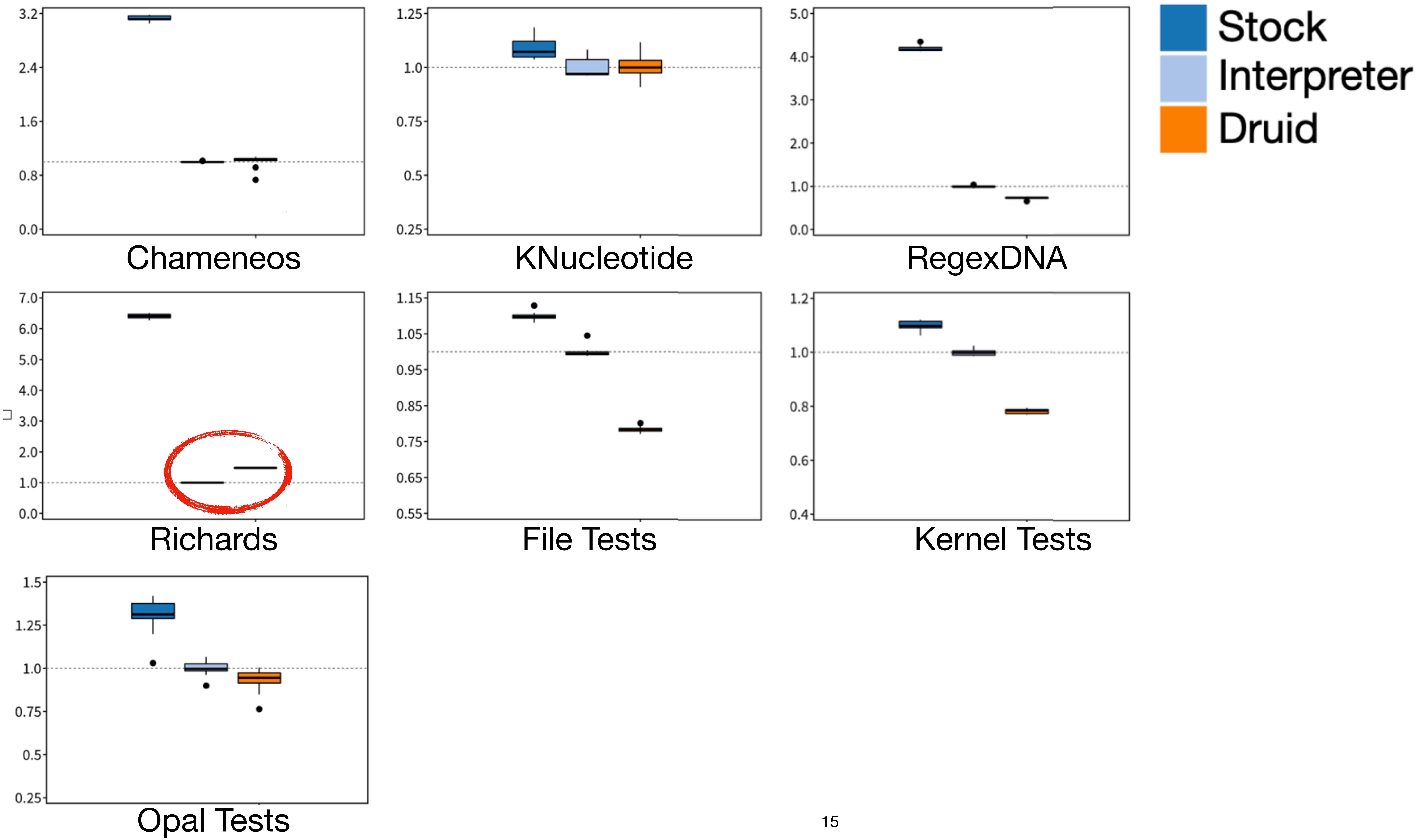
Richards

File Tests

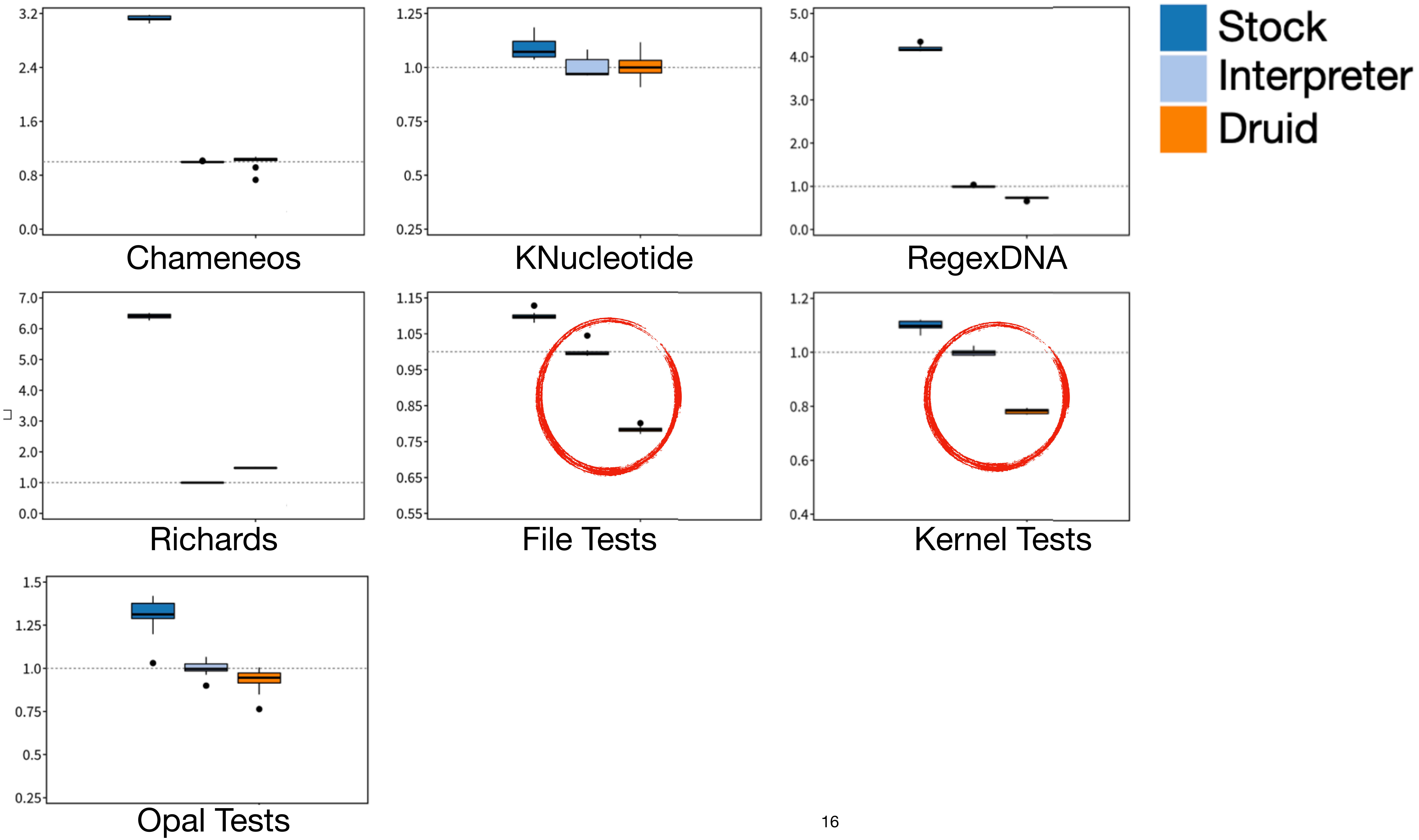
Kernel Tests

Opal Tests

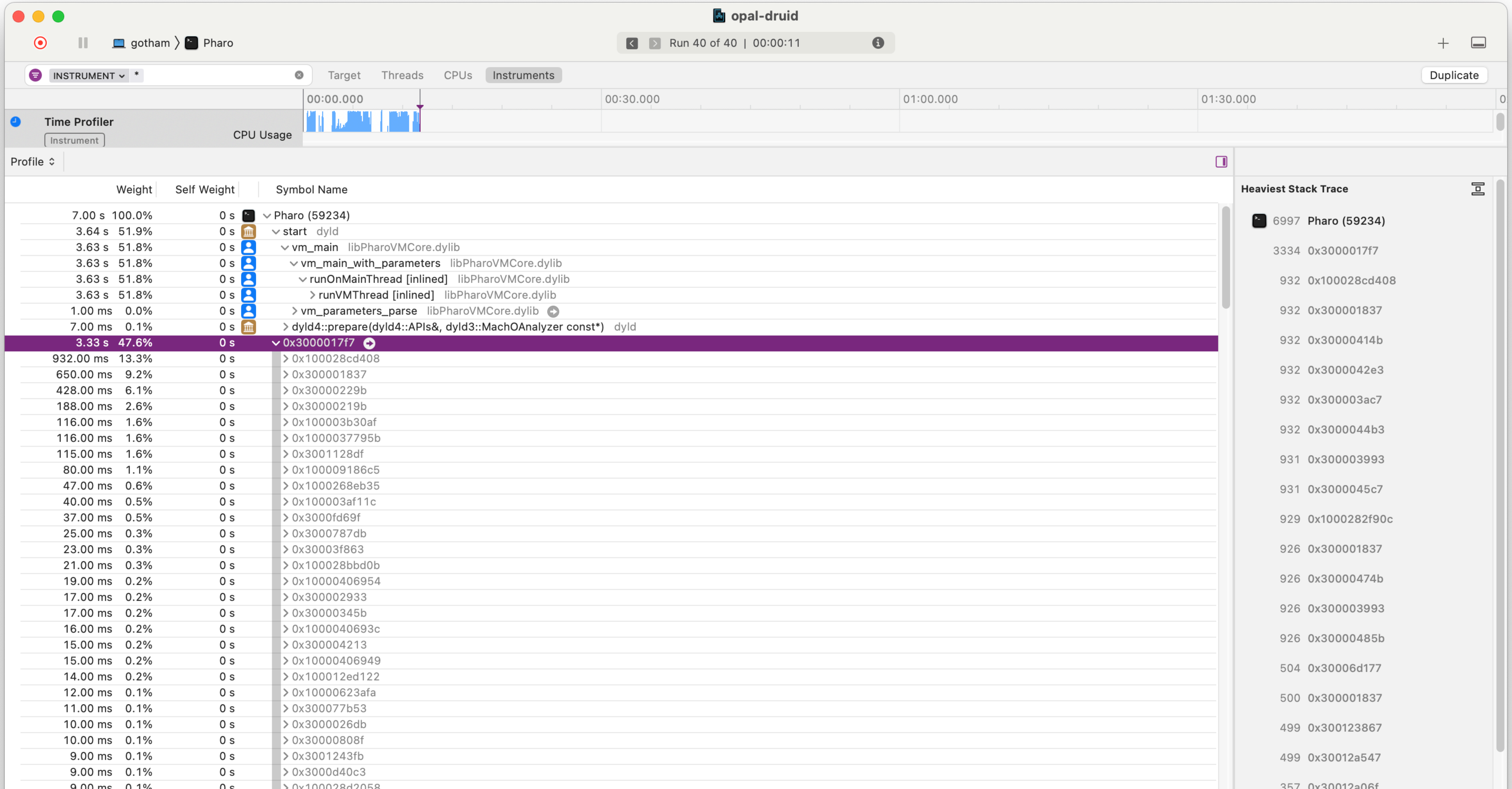
Slightly faster?



And much slower too!



Where does the time go?

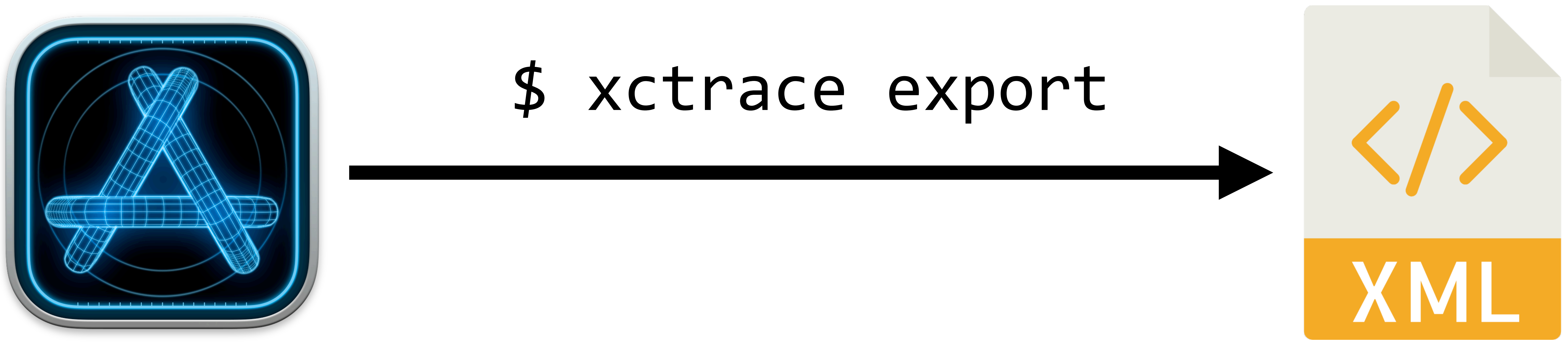


Weight		Self Weight	Symbol Name
7.00 s	100.0%	0 s	⌄ Pharo (59234)
3.64 s	51.9%	0 s	⌄ start dyld
3.63 s	51.8%	0 s	⌄ vm_main libPharoVMCore.dylib
3.63 s	51.8%	0 s	⌄ vm_main_with_parameters libPharoVMCore.dylib
3.63 s	51.8%	0 s	⌄ runOnMainThread [inlined] libPharoVMCore.dylib
3.63 s	51.8%	0 s	› runVMThread [inlined] libPharoVMCore.dylib
1.00 ms	0.0%	0 s	› vm_parameters_parse libPharoVMCore.dylib
7.00 ms	0.1%	0 s	› dyld4::prepare(dyld4::APIs&, dyld3::MachOAnalyzer const*) dyld
3.33 s	47.6%	0 s	⌄ 0x3000017f7
932.00 ms	13.3%	0 s	› 0x100028cd408
650.00 ms	9.2%	0 s	› 0x300001837
428.00 ms	6.1%	0 s	› 0x30000229b
188.00 ms	2.6%	0 s	› 0x30000219b
116.00 ms	1.6%	0 s	› 0x100003b30af
116.00 ms	1.6%	0 s	› 0x1000037795b
115.00 ms	1.6%	0 s	› 0x3001128df
80.00 ms	1.1%	0 s	› 0x100009186c5
47.00 ms	0.6%	0 s	› 0x1000268eb35
40.00 ms	0.5%	0 s	› 0x100003af11c
37.00 ms	0.5%	0 s	› 0x3000fd69f
25.00 ms	0.3%	0 s	› 0x3000787db
23.00 ms	0.3%	0 s	› 0x30003f863
21.00 ms	0.3%	0 s	› 0x100028bbd0b
19.00 ms	0.2%	0 s	› 0x10000406954
17.00 ms	0.2%	0 s	› 0x300002933
17.00 ms	0.2%	0 s	› 0x30000345b
16.00 ms	0.2%	0 s	› 0x1000040693c
15.00 ms	0.2%	0 s	› 0x300004213
15.00 ms	0.2%	0 s	› 0x10000406949
14.00 ms	0.2%	0 s	› 0x100012ed122
12.00 ms	0.1%	0 s	› 0x10000623afa
11.00 ms	0.1%	0 s	› 0x300077b53
10.00 ms	0.1%	0 s	› 0x3000026db
10.00 ms	0.1%	0 s	› 0x30000808f
9.00 ms	0.1%	0 s	› 0x3001243fb
9.00 ms	0.1%	0 s	› 0x3000d40c3
9.00 ms	0.1%	0 s	› 0x100028d2058

Heaviest Stack Trace

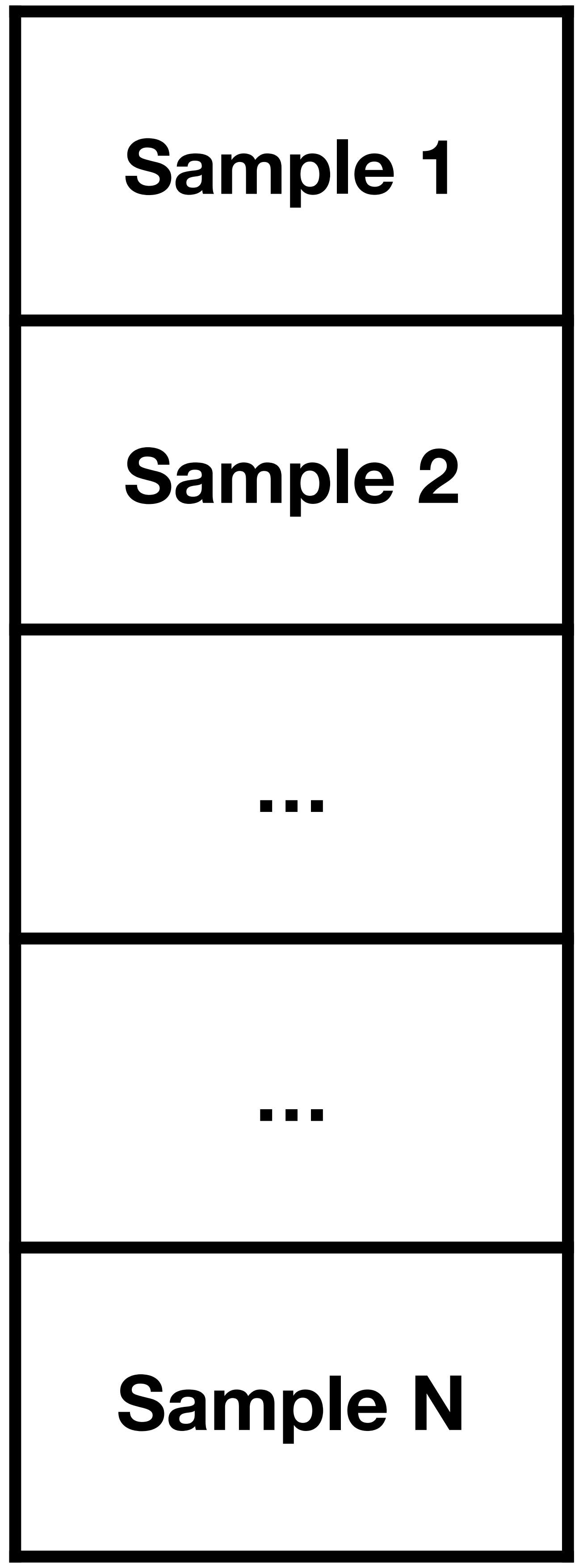
6997	Pharo (59234)
3334	0x3000017f7
932	0x100028cd408
932	0x300001837
932	0x30000414b
932	0x3000042e3
932	0x300003ac7
932	0x3000044b3
931	0x300003993
931	0x3000045c7
929	0x1000282f90c
926	0x300001837
926	0x30000474b
926	0x300003993
926	0x30000485b
504	0x30006d177
500	0x300001837
499	0x300123867
499	0x30012a547
357	0x30012a06f

Analysing Instruments Profiles



```
$ xctrace export
```

Analyzing Samples



Sample 1

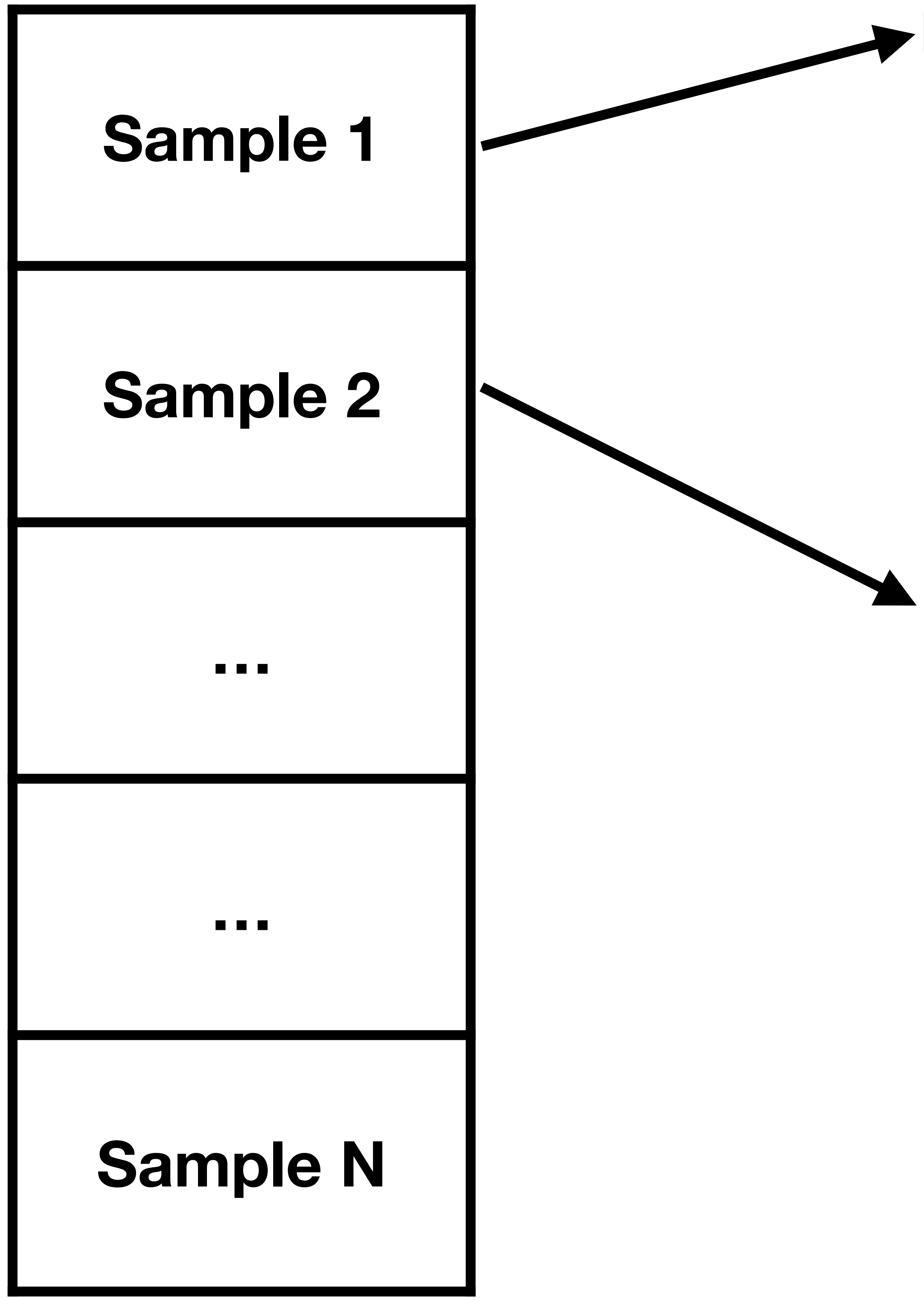
Sample 2

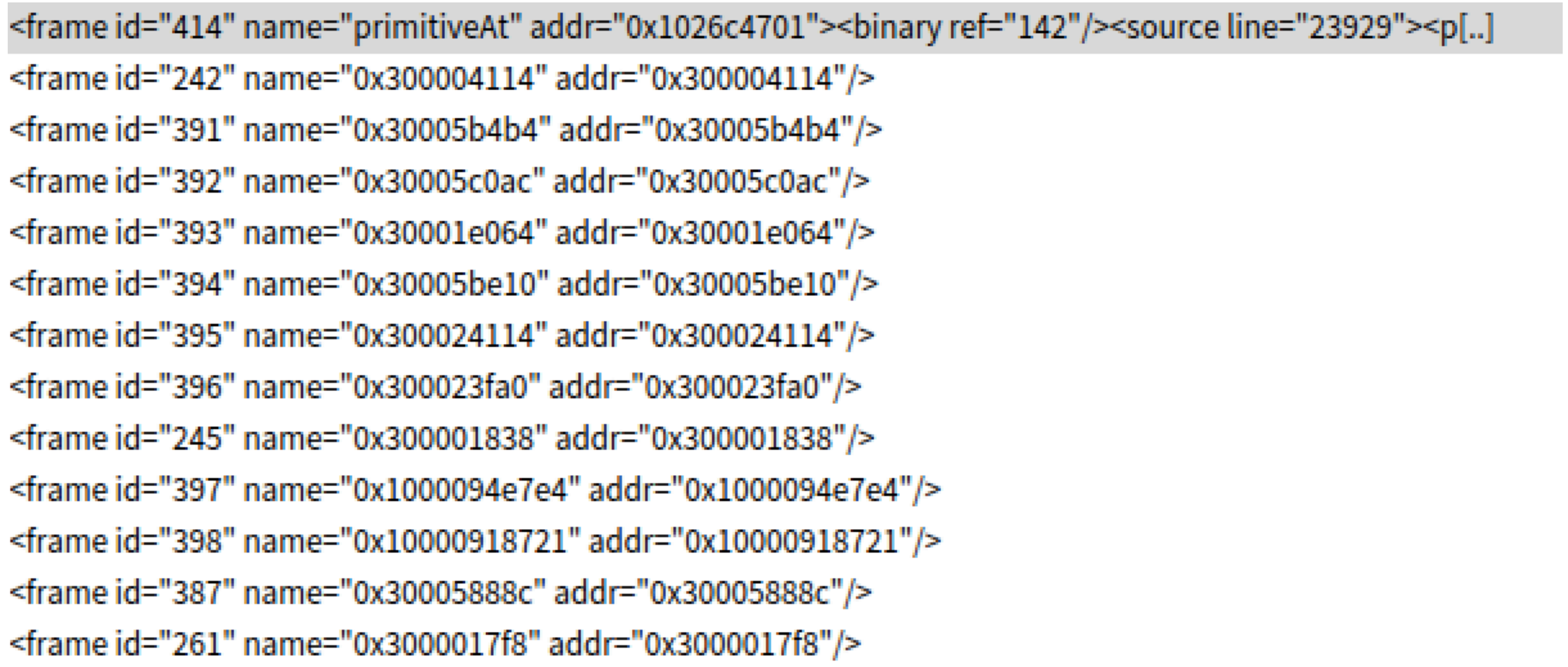
...

...

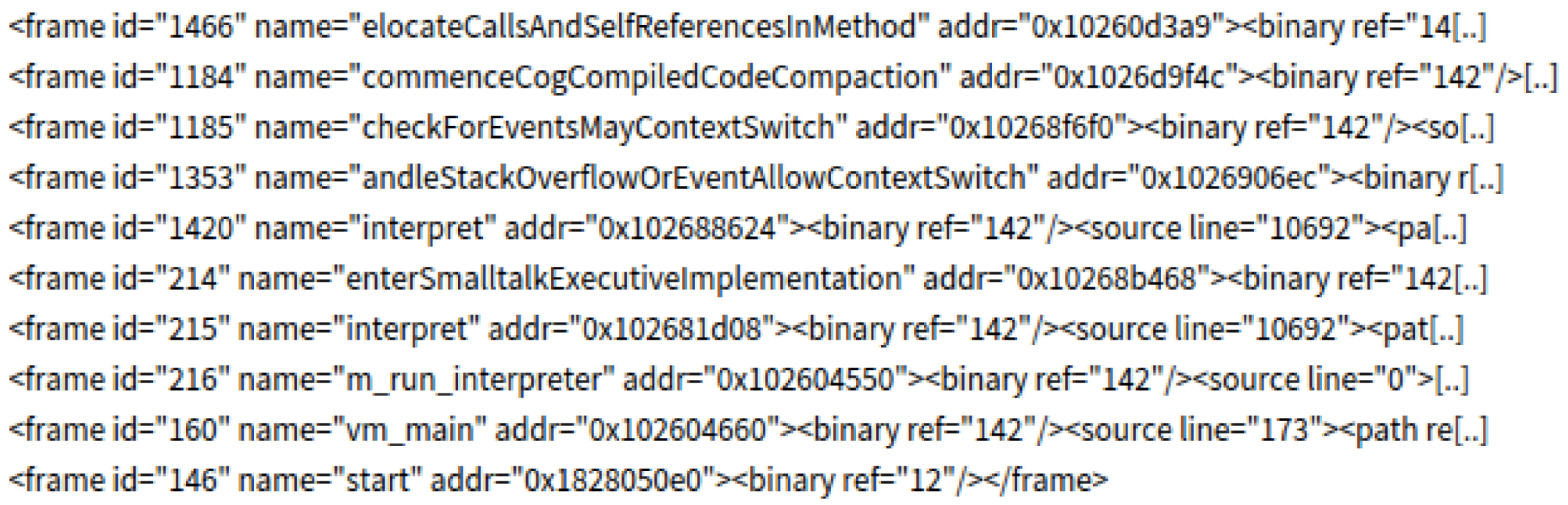
Sample N

Analyzing Samples





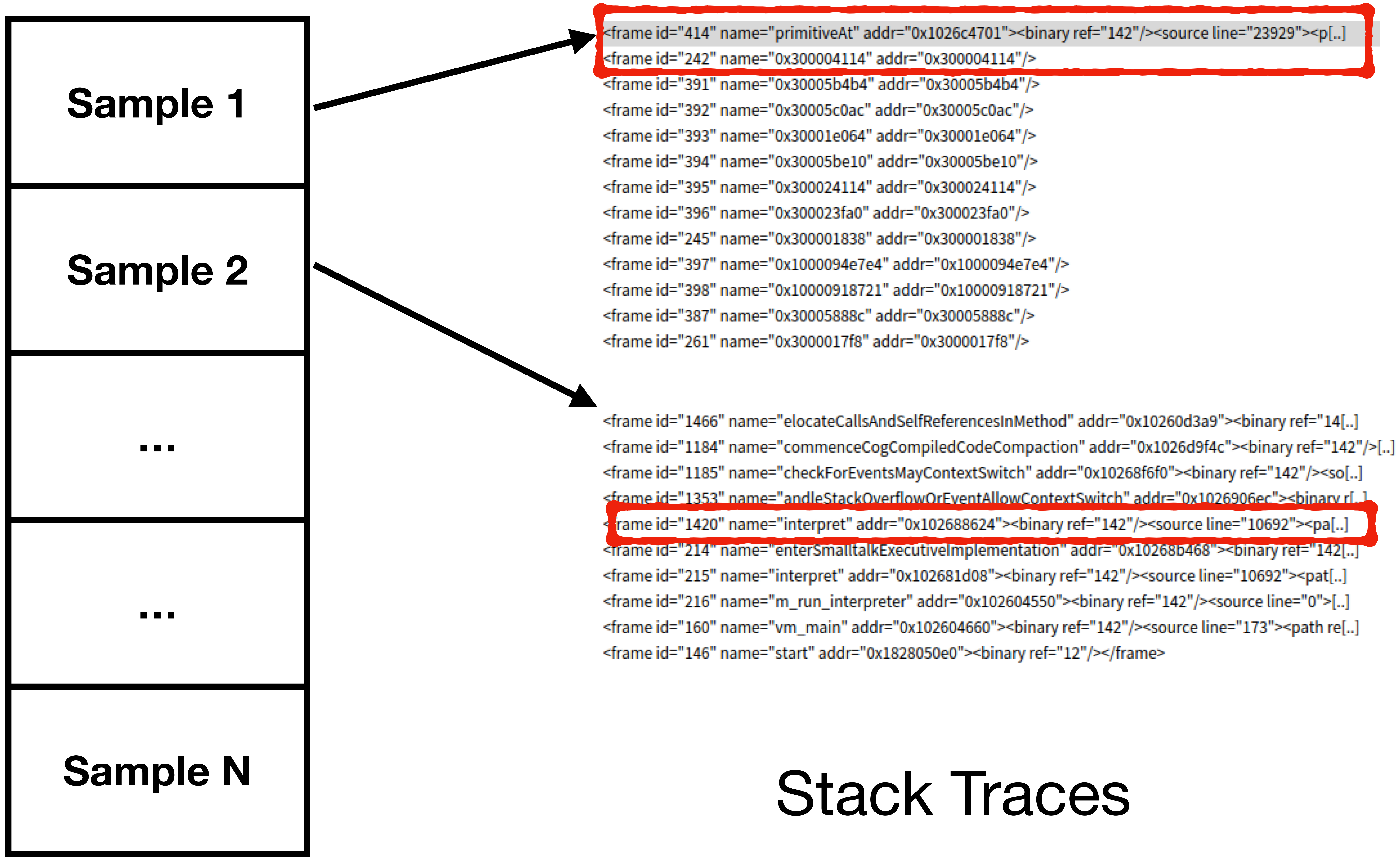
```
<frame id="414" name="primitiveAt" addr="0x1026c4701"><binary ref="142"/><source line="23929"><p[..]
<frame id="242" name="0x300004114" addr="0x300004114"/>
<frame id="391" name="0x30005b4b4" addr="0x30005b4b4"/>
<frame id="392" name="0x30005c0ac" addr="0x30005c0ac"/>
<frame id="393" name="0x30001e064" addr="0x30001e064"/>
<frame id="394" name="0x30005be10" addr="0x30005be10"/>
<frame id="395" name="0x300024114" addr="0x300024114"/>
<frame id="396" name="0x300023fa0" addr="0x300023fa0"/>
<frame id="245" name="0x300001838" addr="0x300001838"/>
<frame id="397" name="0x1000094e7e4" addr="0x1000094e7e4"/>
<frame id="398" name="0x10000918721" addr="0x10000918721"/>
<frame id="387" name="0x30005888c" addr="0x30005888c"/>
<frame id="261" name="0x3000017f8" addr="0x3000017f8"/>
```



```
<frame id="1466" name="elocateCallsAndSelfReferencesInMethod" addr="0x10260d3a9"><binary ref="14[..]
<frame id="1184" name="commenceCogCompiledCodeCompaction" addr="0x1026d9f4c"><binary ref="142"/>[..]
<frame id="1185" name="checkForEventsMayContextSwitch" addr="0x10268f6f0"><binary ref="142"/><so[..]
<frame id="1353" name="andleStackOverflowOrEventAllowContextSwitch" addr="0x1026906ec"><binary r[..]
<frame id="1420" name="interpret" addr="0x102688624"><binary ref="142"/><source line="10692"><pa[..]
<frame id="214" name="enterSmalltalkExecutiveImplementation" addr="0x10268b468"><binary ref="142[..]
<frame id="215" name="interpret" addr="0x102681d08"><binary ref="142"/><source line="10692"><pat[..]
<frame id="216" name="m_run_interpreter" addr="0x102604550"><binary ref="142"/><source line="0">[..]
<frame id="160" name="vm_main" addr="0x102604660"><binary ref="142"/><source line="173"><path re[..]
<frame id="146" name="start" addr="0x1828050e0"><binary ref="12"/></frame>
```

Stack Traces

Analyzing Samples

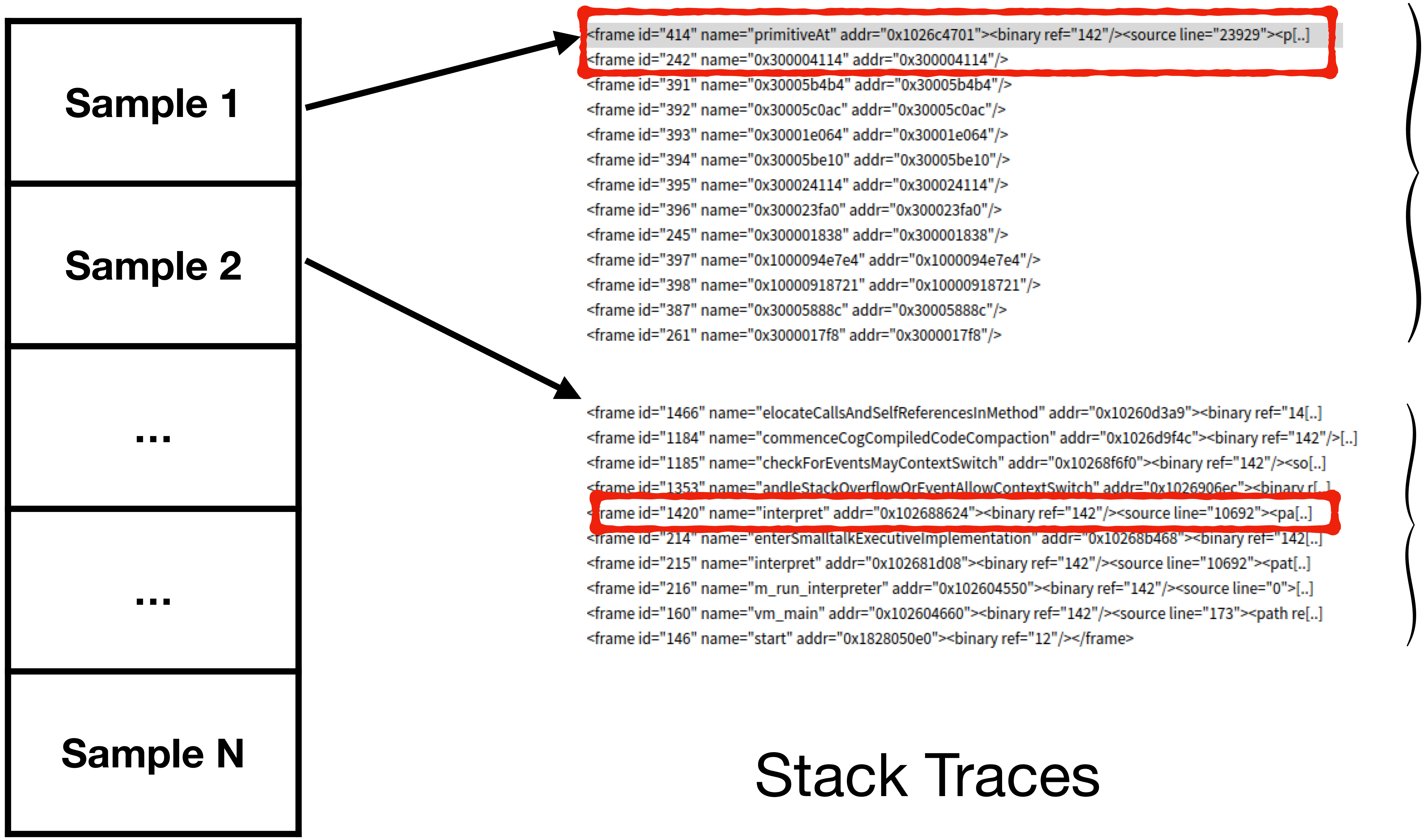


```
<frame id="414" name="primitiveAt" addr="0x1026c4701"><binary ref="142"/><source line="23929"><p[..]
<frame id="242" name="0x300004114" addr="0x300004114"/>
<frame id="391" name="0x30005b4b4" addr="0x30005b4b4"/>
<frame id="392" name="0x30005c0ac" addr="0x30005c0ac"/>
<frame id="393" name="0x30001e064" addr="0x30001e064"/>
<frame id="394" name="0x30005be10" addr="0x30005be10"/>
<frame id="395" name="0x300024114" addr="0x300024114"/>
<frame id="396" name="0x300023fa0" addr="0x300023fa0"/>
<frame id="245" name="0x300001838" addr="0x300001838"/>
<frame id="397" name="0x1000094e7e4" addr="0x1000094e7e4"/>
<frame id="398" name="0x10000918721" addr="0x10000918721"/>
<frame id="387" name="0x30005888c" addr="0x30005888c"/>
<frame id="261" name="0x3000017f8" addr="0x3000017f8"/>
```

```
<frame id="1466" name="elocateCallsAndSelfReferencesInMethod" addr="0x10260d3a9"><binary ref="14[..]
<frame id="1184" name="commenceCogCompiledCodeCompaction" addr="0x1026d9f4c"><binary ref="142"/>[..]
<frame id="1185" name="checkForEventsMayContextSwitch" addr="0x10268f6f0"><binary ref="142"/><so[..]
<frame id="1353" name="andleStackOverflowOrEventAllowContextSwitch" addr="0x1026906ec"><binary r[..]
<frame id="1420" name="interpret" addr="0x102688624"><binary ref="142"/><source line="10692"><pa[..]
<frame id="214" name="enterSmalltalkExecutiveImplementation" addr="0x10268b468"><binary ref="142[..]
<frame id="215" name="interpret" addr="0x102681d08"><binary ref="142"/><source line="10692"><pat[..]
<frame id="216" name="m_run_interpreter" addr="0x102604550"><binary ref="142"/><source line="0">[..]
<frame id="160" name="vm_main" addr="0x102604660"><binary ref="142"/><source line="173"><path re[..]
<frame id="146" name="start" addr="0x1828050e0"><binary ref="12"/></frame>
```

Stack Traces

Analyzing Samples



```
<frame id="414" name="primitiveAt" addr="0x1026c4701"><binary ref="142"/><source line="23929"><p[..]
<frame id="242" name="0x300004114" addr="0x300004114"/>
<frame id="391" name="0x30005b4b4" addr="0x30005b4b4"/>
<frame id="392" name="0x30005c0ac" addr="0x30005c0ac"/>
<frame id="393" name="0x30001e064" addr="0x30001e064"/>
<frame id="394" name="0x30005be10" addr="0x30005be10"/>
<frame id="395" name="0x300024114" addr="0x300024114"/>
<frame id="396" name="0x300023fa0" addr="0x300023fa0"/>
<frame id="245" name="0x300001838" addr="0x300001838"/>
<frame id="397" name="0x1000094e7e4" addr="0x1000094e7e4"/>
<frame id="398" name="0x10000918721" addr="0x10000918721"/>
<frame id="387" name="0x30005888c" addr="0x30005888c"/>
<frame id="261" name="0x3000017f8" addr="0x3000017f8"/>
```

Primitive from

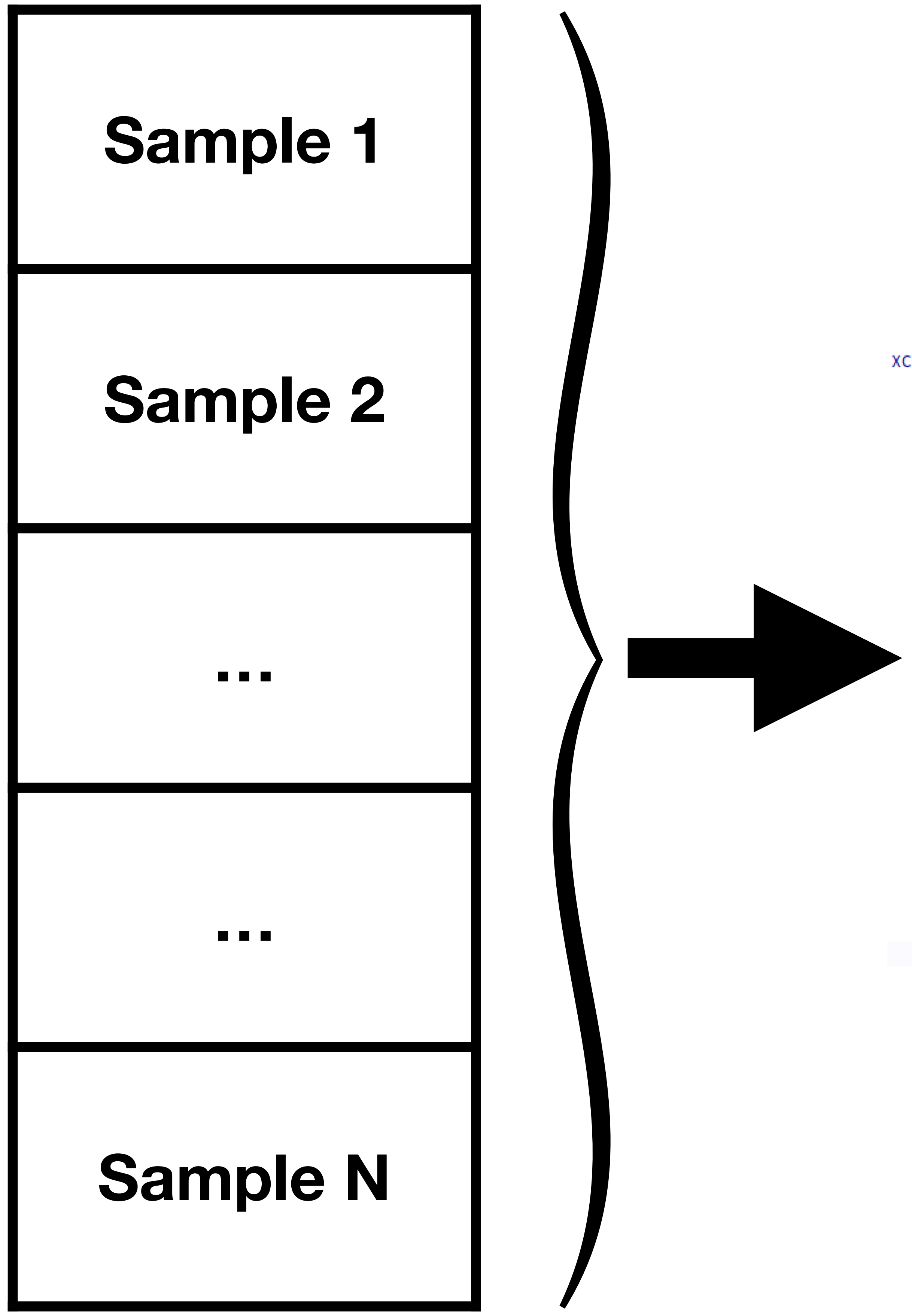
Machine Code

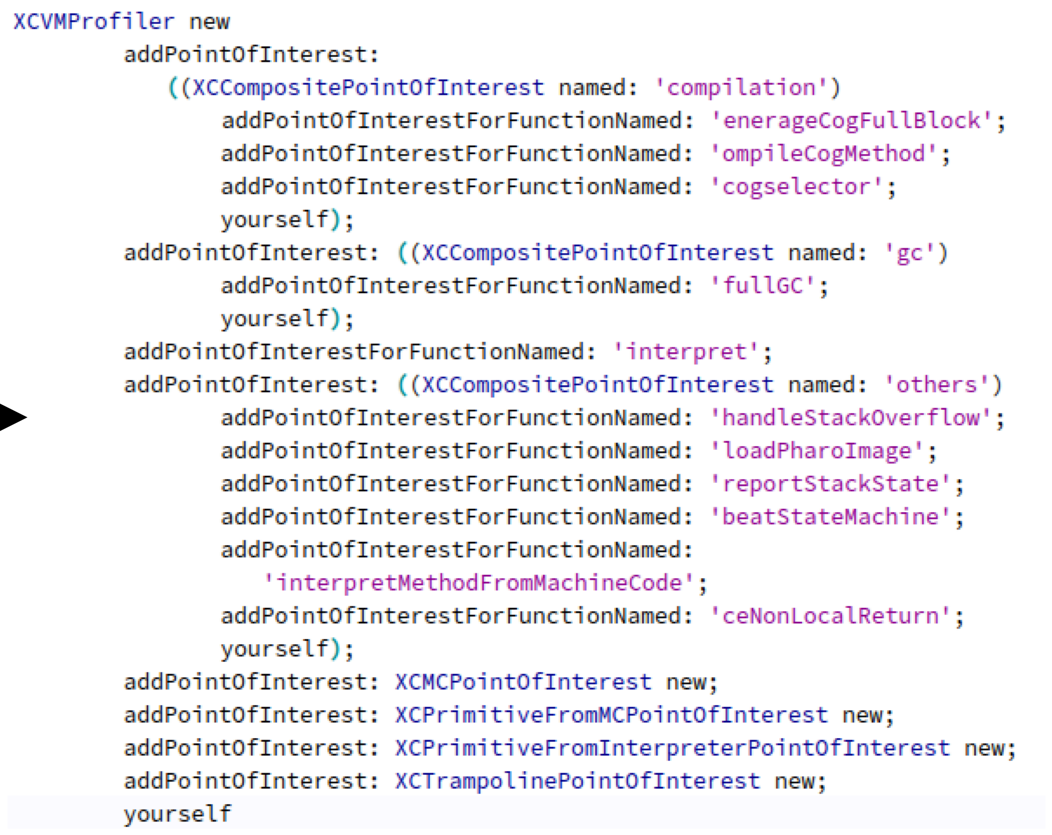
```
<frame id="1466" name="elocateCallsAndSelfReferencesInMethod" addr="0x10260d3a9"><binary ref="14[..]
<frame id="1184" name="commenceCogCompiledCodeCompaction" addr="0x1026d9f4c"><binary ref="142"/>[..]
<frame id="1185" name="checkForEventsMayContextSwitch" addr="0x10268f6f0"><binary ref="142"/><so[..]
<frame id="1353" name="andleStackOverflowOrEventAllowContextSwitch" addr="0x1026906ec"><binary r[..]
<frame id="1420" name="interpret" addr="0x102688624"><binary ref="142"/><source line="10692"><pa[..]
<frame id="214" name="enterSmalltalkExecutiveImplementation" addr="0x10268b468"><binary ref="142[..]
<frame id="215" name="interpret" addr="0x102681d08"><binary ref="142"/><source line="10692"><pat[..]
<frame id="216" name="m_run_interpreter" addr="0x102604550"><binary ref="142"/><source line="0">[..]
<frame id="160" name="vm_main" addr="0x102604660"><binary ref="142"/><source line="173"><path re[..]
<frame id="146" name="start" addr="0x1828050e0"><binary ref="12"/></frame>
```

Interpreter

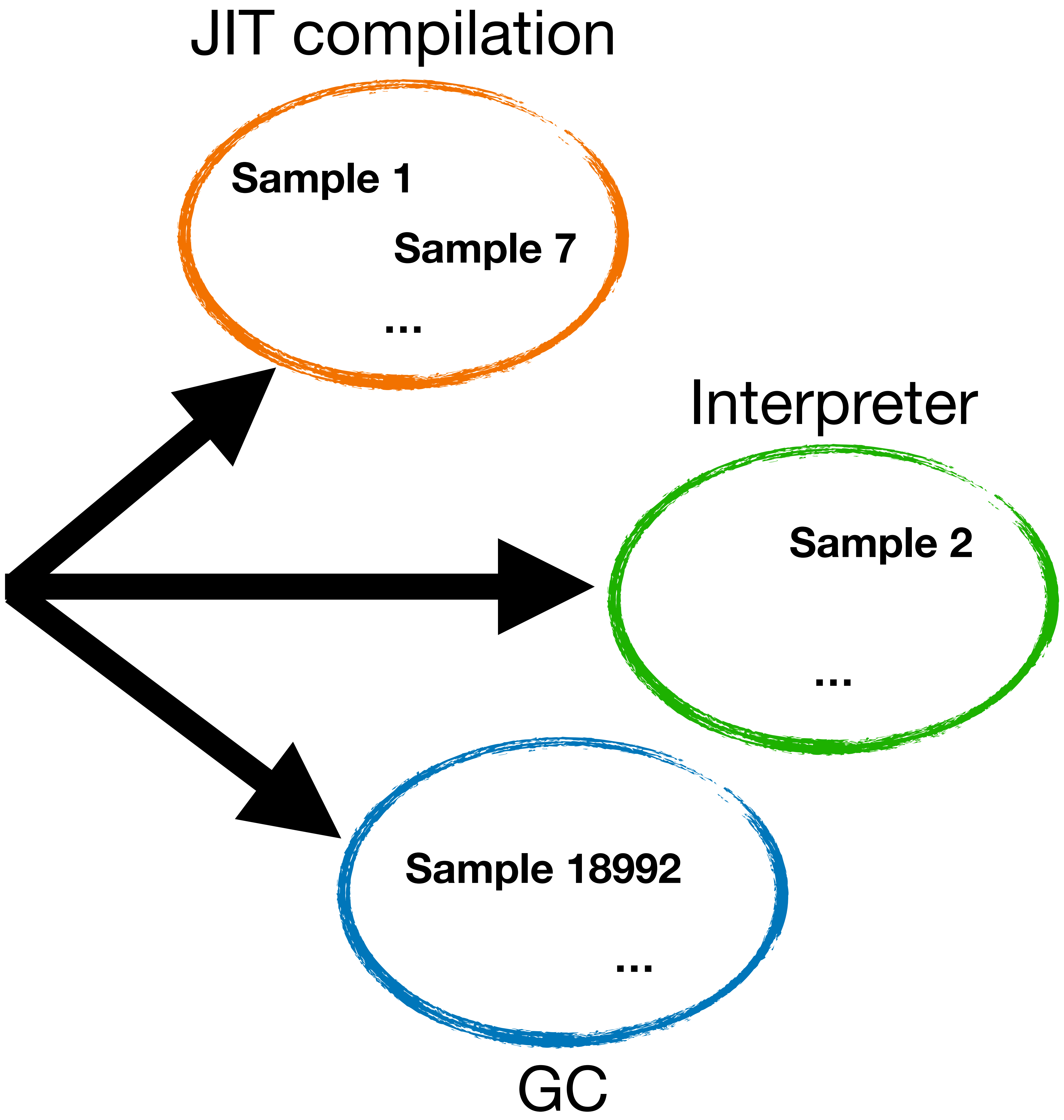
Stack Traces

Group Traces Using Heuristics



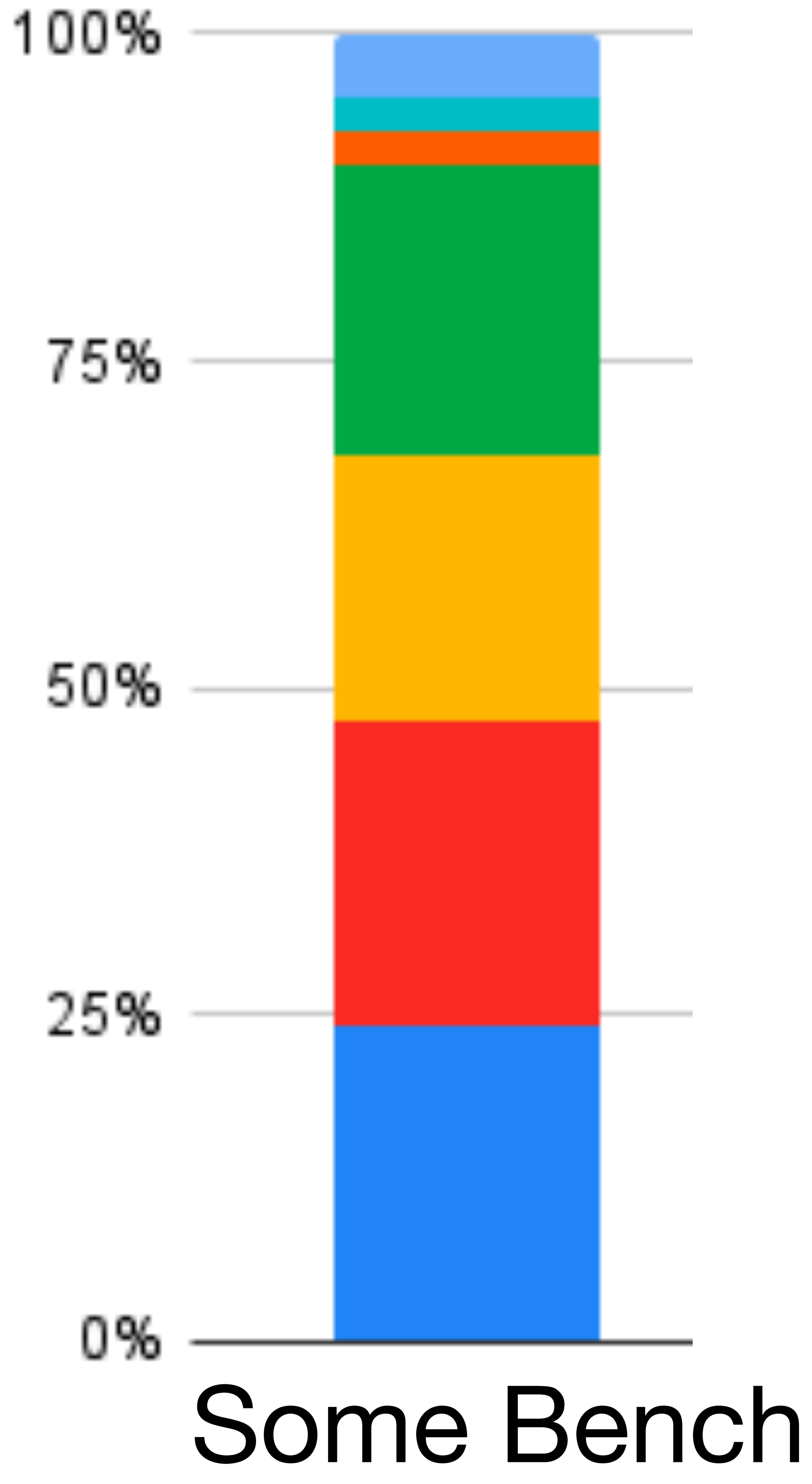


```
XCVMProfiler new
        addPointOfInterest:
           ((XCCompositePointOfInterest named: 'compilation')
              addPointOfInterestForFunctionNamed: 'eneragecogFullBlock';
              addPointOfInterestForFunctionNamed: 'ompileCogMethod';
              addPointOfInterestForFunctionNamed: 'cogselector';
              yourself);
        addPointOfInterest: ((XCCompositePointOfInterest named: 'gc')
              addPointOfInterestForFunctionNamed: 'fullGC';
              yourself);
        addPointOfInterestForFunctionNamed: 'interpret';
        addPointOfInterest: ((XCCompositePointOfInterest named: 'others')
              addPointOfInterestForFunctionNamed: 'handleStackOverflow';
              addPointOfInterestForFunctionNamed: 'loadPharoImage';
              addPointOfInterestForFunctionNamed: 'reportStackState';
              addPointOfInterestForFunctionNamed: 'beatStateMachine';
              addPointOfInterestForFunctionNamed:
                 'interpretMethodFromMachineCode';
              addPointOfInterestForFunctionNamed: 'ceNonLocalReturn';
              yourself);
        addPointOfInterest: XCMCPointOfInterest new;
        addPointOfInterest: XCPrimitiveFromMCPointOfInterest new;
        addPointOfInterest: XCPrimitiveFromInterpreterPointOfInterest new;
        addPointOfInterest: XCTrampolinePointOfInterest new;
        yourself
```

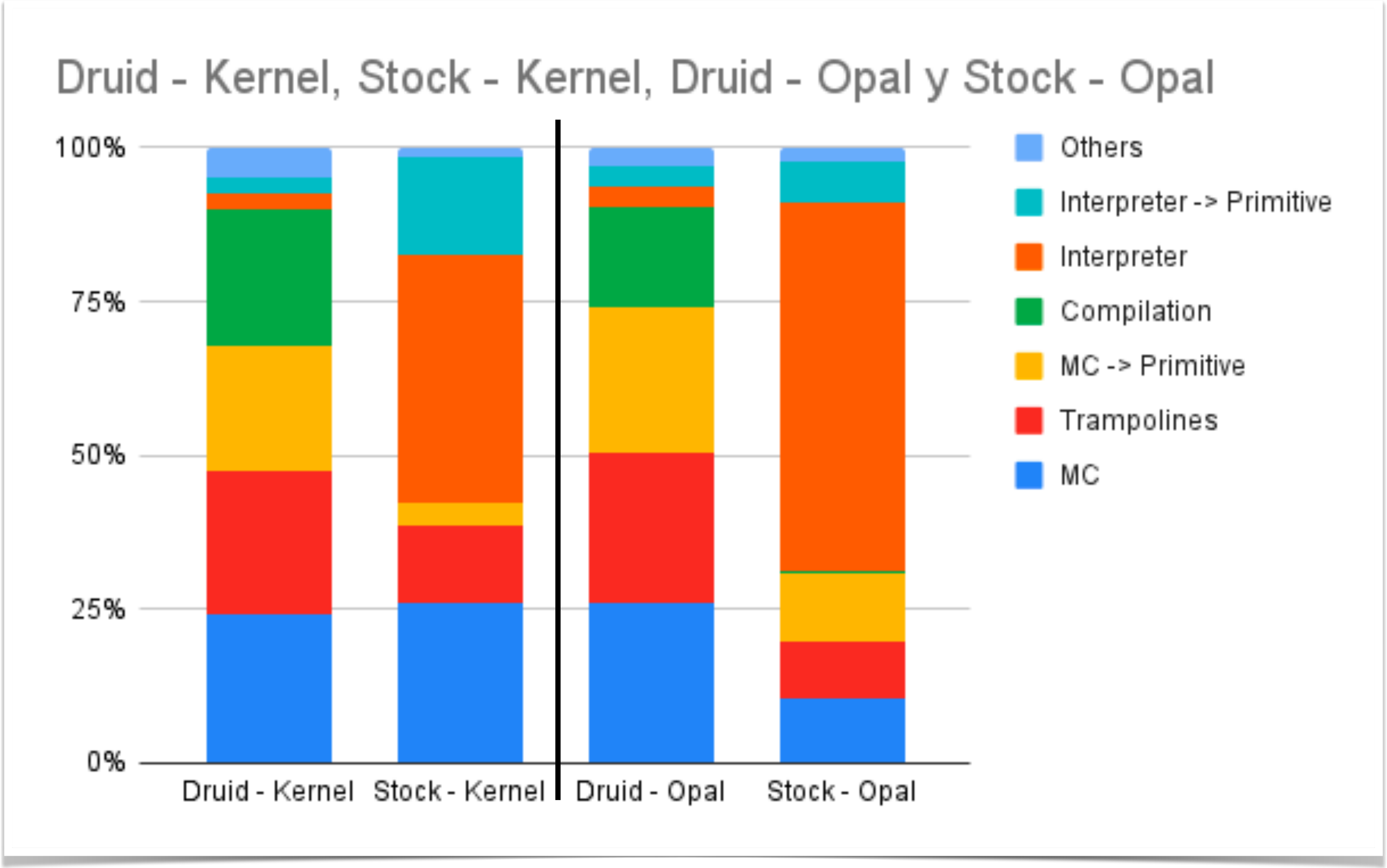


High-level VM Profile

- Time spent in
  - Interpreter
  - JIT compilation
  - JIT compiled code
  - GC
  - Primitives
  - …

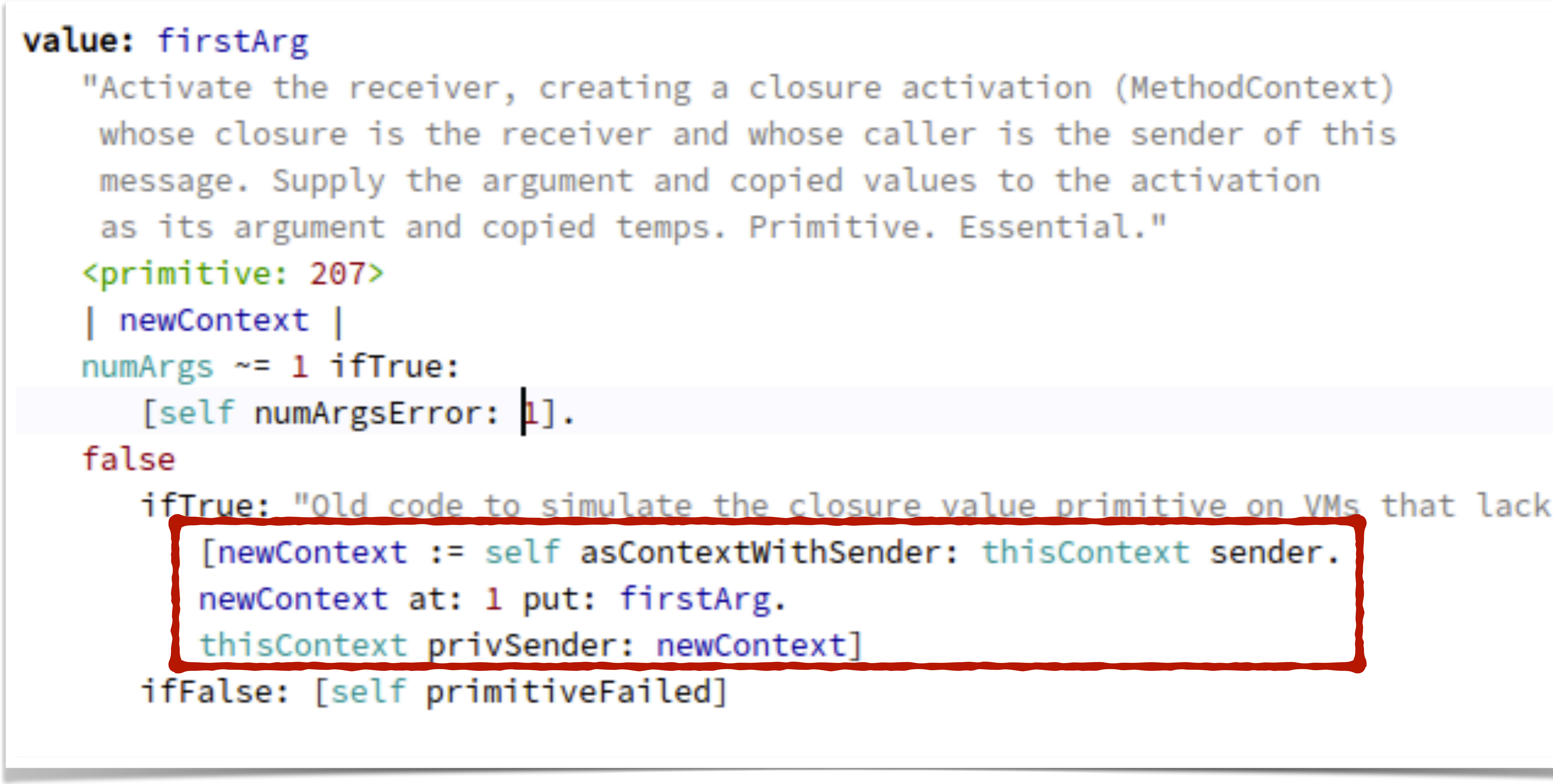


Scenario 1: Cross-JIT Profiling



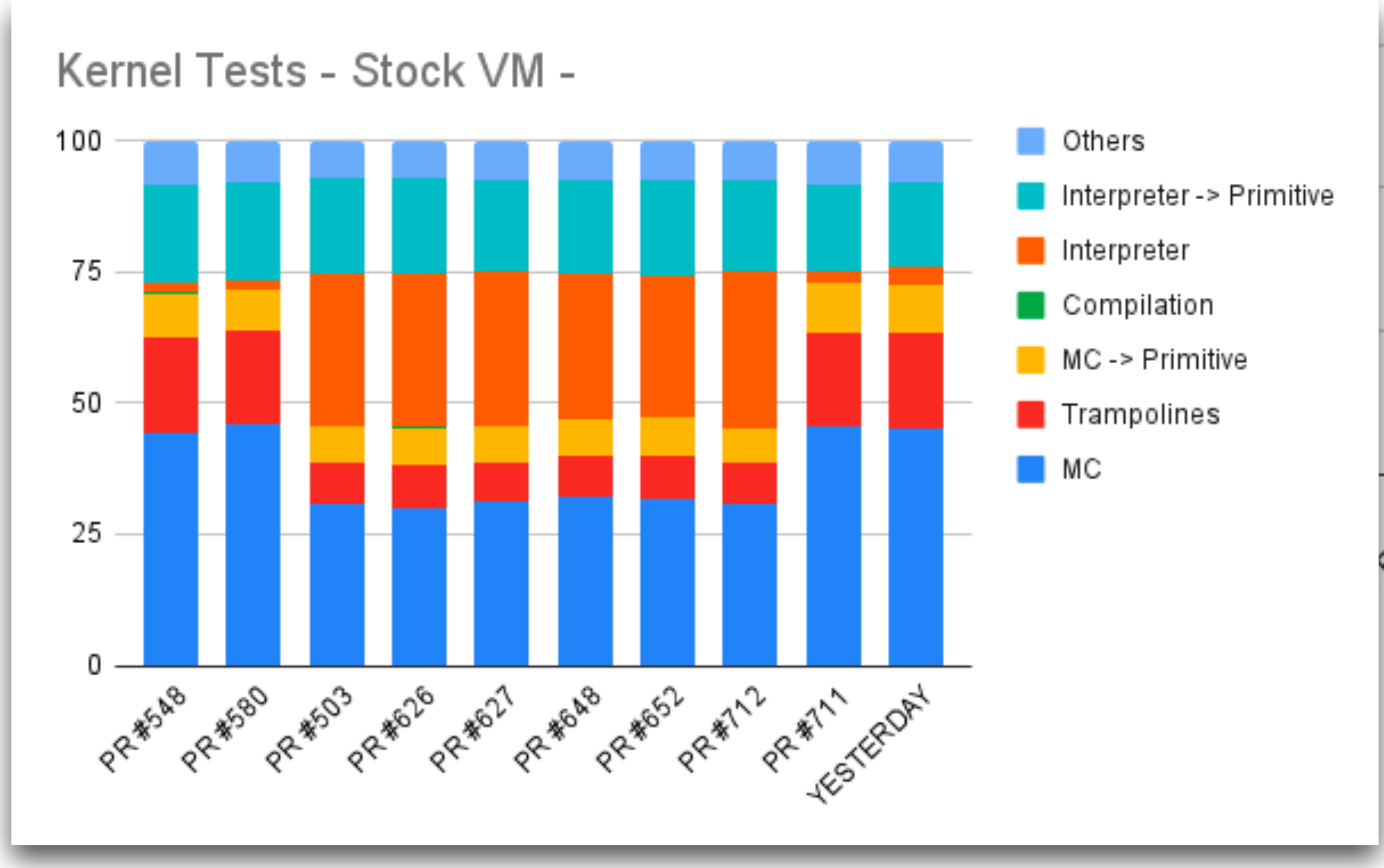
Hot Paths and Our Partial JIT Implementation

- Cogit is *all or nothing compiler*
- Hot path is not compiled!

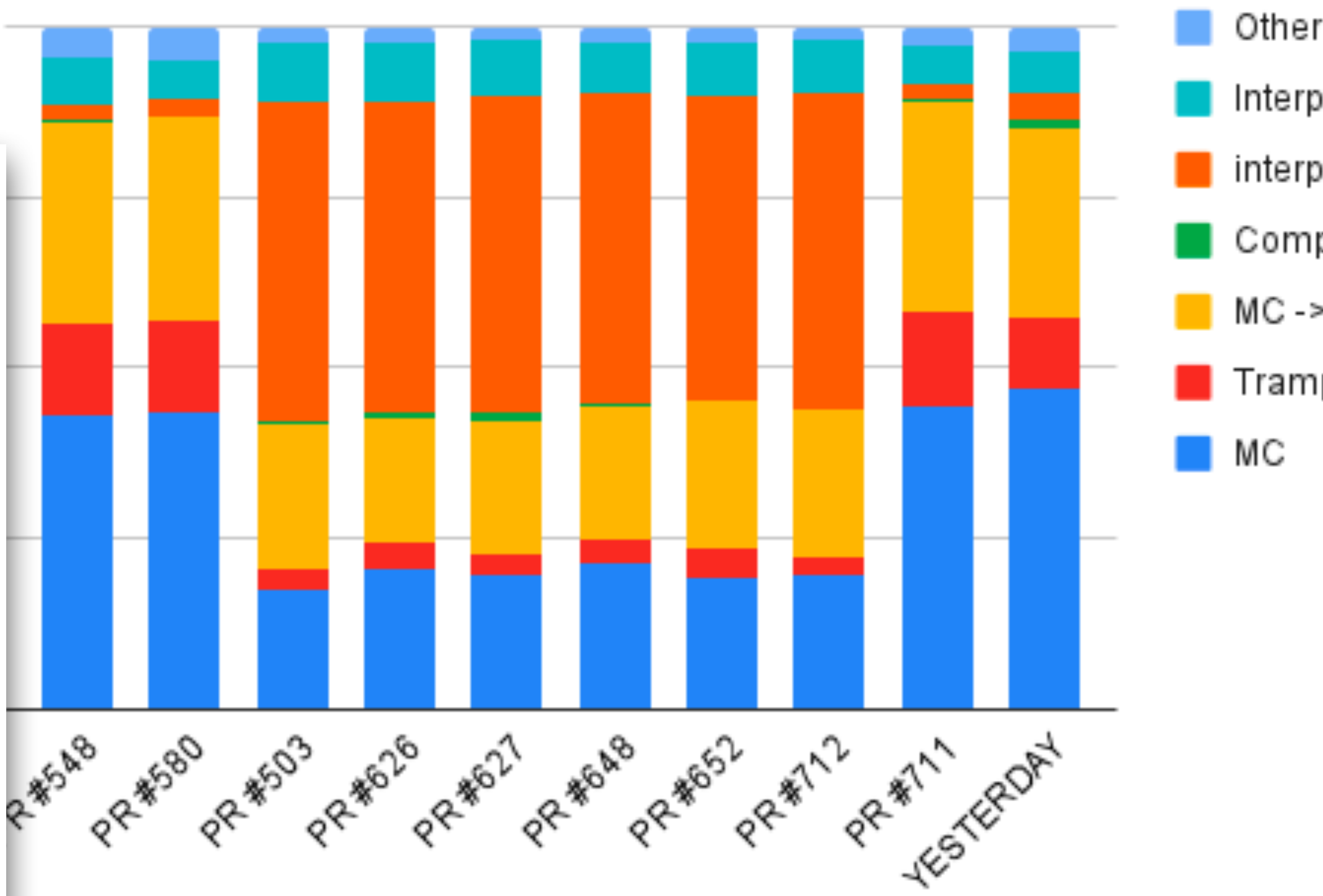


```
value: firstArg
    "Activate the receiver, creating a closure activation (MethodContext)
     whose closure is the receiver and whose caller is the sender of this
     message. Supply the argument and copied values to the activation
     as its argument and copied temps. Primitive. Essential."
    <primitive: 207>
    | newContext |
    numArgs ~= 1 ifTrue:
        [self numArgsError: 1].
    false
        ifTrue: "Old code to simulate the closure value primitive on VMs that lack
            [newContext := self asContextWithSender: thisContext sender.
            newContext at: 1 put: firstArg.
            thisContext privSender: newContext]
        ifFalse: [self primitiveFailed]
```

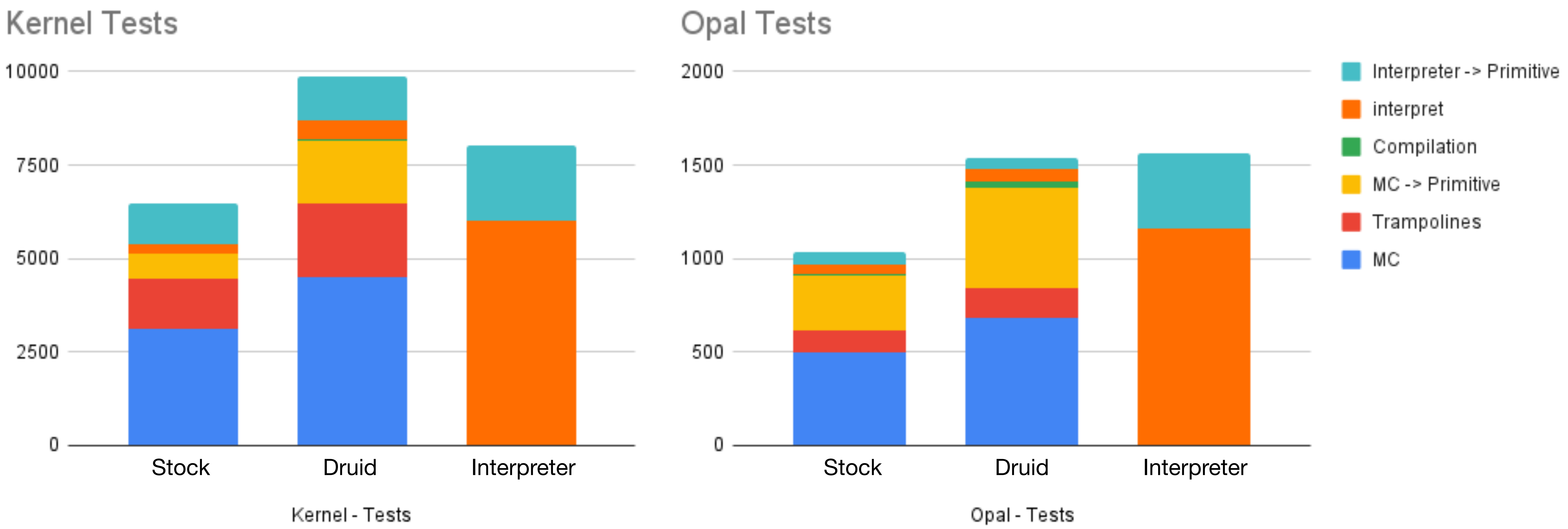
Scenario 2: Cross-Version Profiling



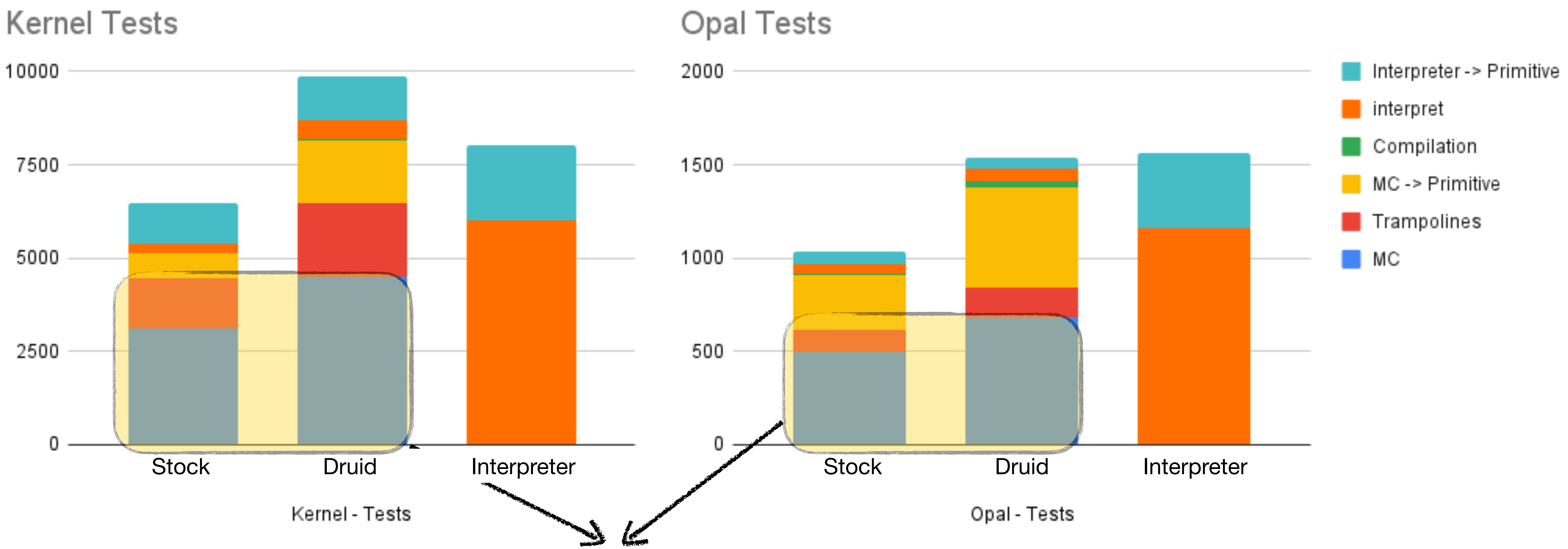
Opal Tests - Stock VM - Druid branch



Differential Profiling + Absolute Values

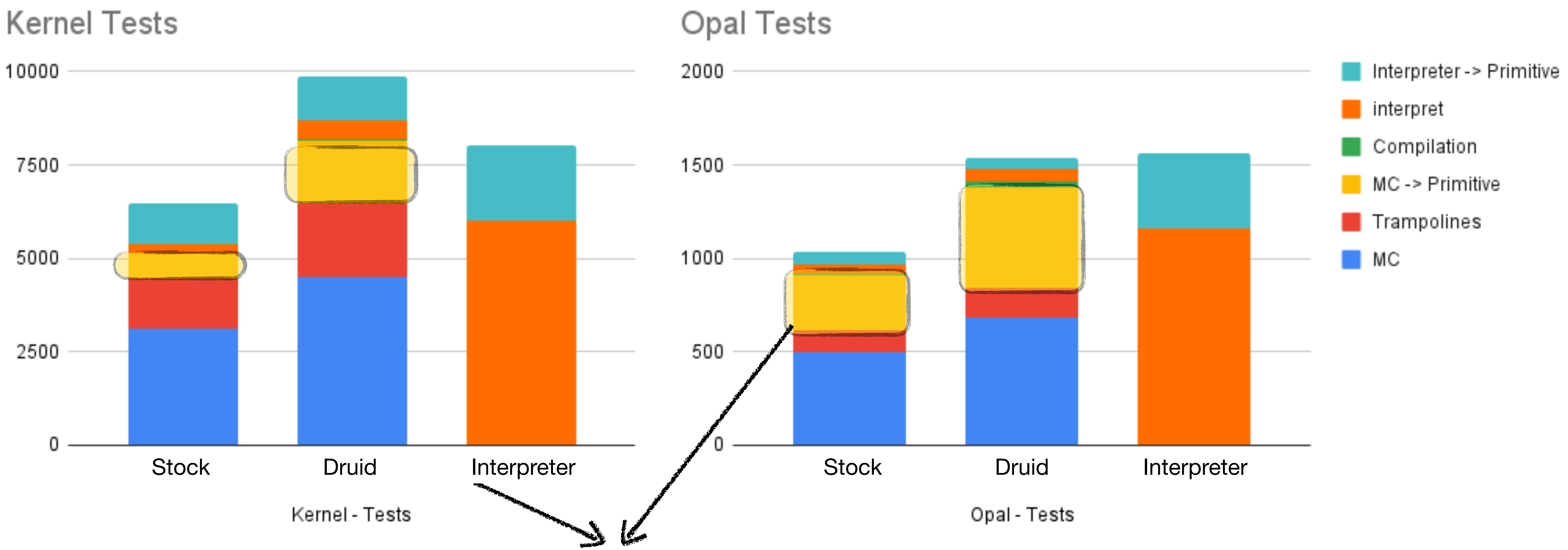


Differential Profiling + Absolute Values



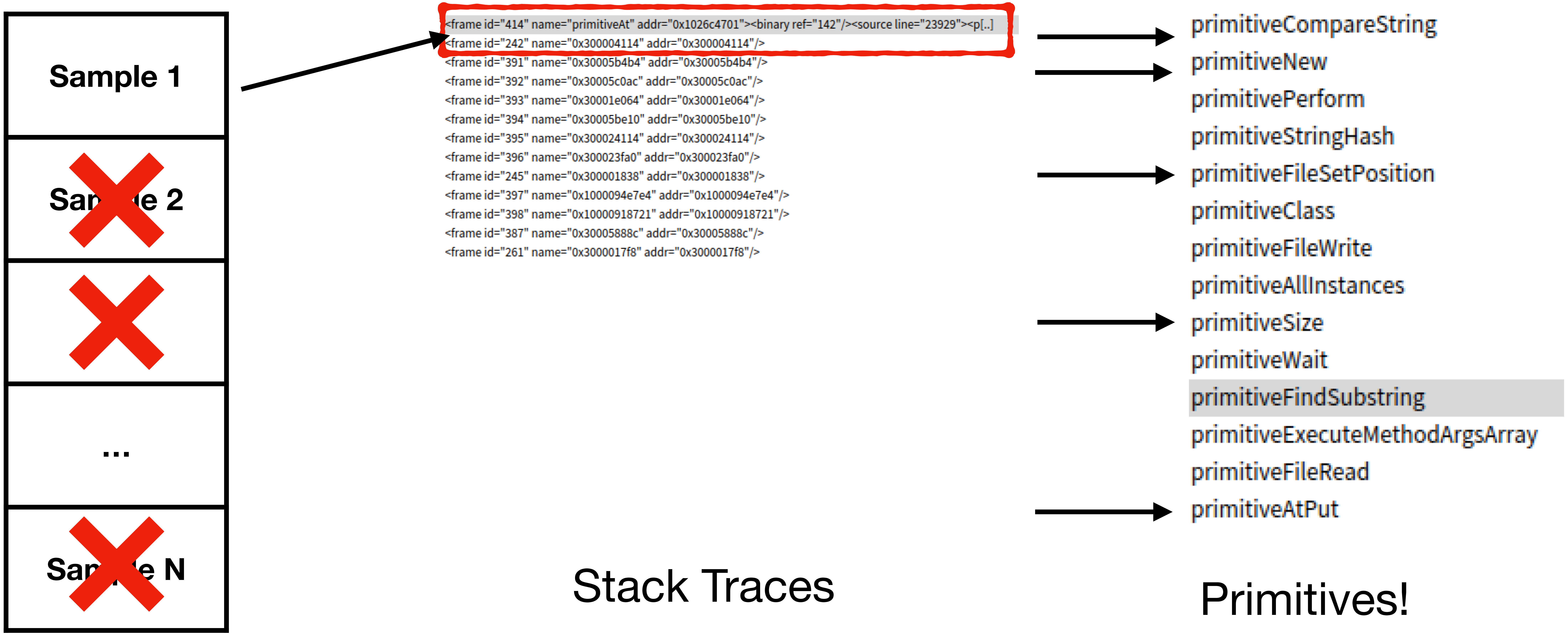
Worse Quality MC

Differential Profiling



More time in Primitives

Drill-down in MC -> Primitives



```
<frame id="414" name="primitiveAt" addr="0x1026c4701"><binary ref="142"/><source line="23929"><p[..]
<frame id="242" name="0x300004114" addr="0x300004114"/>
<frame id="391" name="0x30005b4b4" addr="0x30005b4b4"/>
<frame id="392" name="0x30005c0ac" addr="0x30005c0ac"/>
<frame id="393" name="0x30001e064" addr="0x30001e064"/>
<frame id="394" name="0x30005be10" addr="0x30005be10"/>
<frame id="395" name="0x300024114" addr="0x300024114"/>
<frame id="396" name="0x300023fa0" addr="0x300023fa0"/>
<frame id="245" name="0x300001838" addr="0x300001838"/>
<frame id="397" name="0x1000094e7e4" addr="0x1000094e7e4"/>
<frame id="398" name="0x10000918721" addr="0x10000918721"/>
<frame id="387" name="0x30005888c" addr="0x30005888c"/>
<frame id="261" name="0x3000017f8" addr="0x3000017f8"/>
```

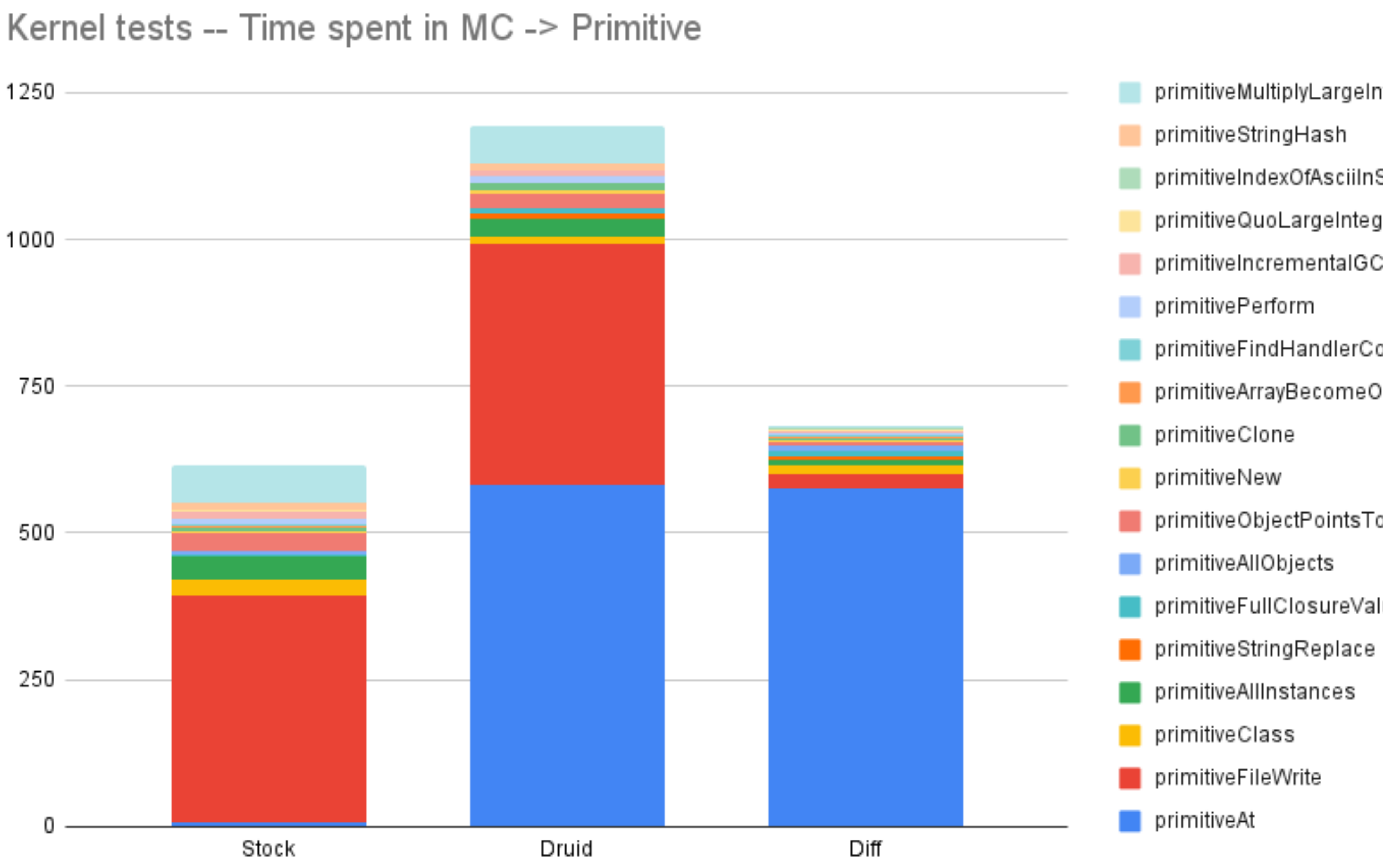
primitiveCompareString
primitiveNew
primitivePerform
primitiveStringHash
primitiveFileSetPosition
primitiveClass
primitiveFileWrite
primitiveAllInstances
primitiveSize
primitiveWait
primitiveFindSubstring
primitiveExecuteMethodArgsArray
primitiveFileRead
primitiveAtPut

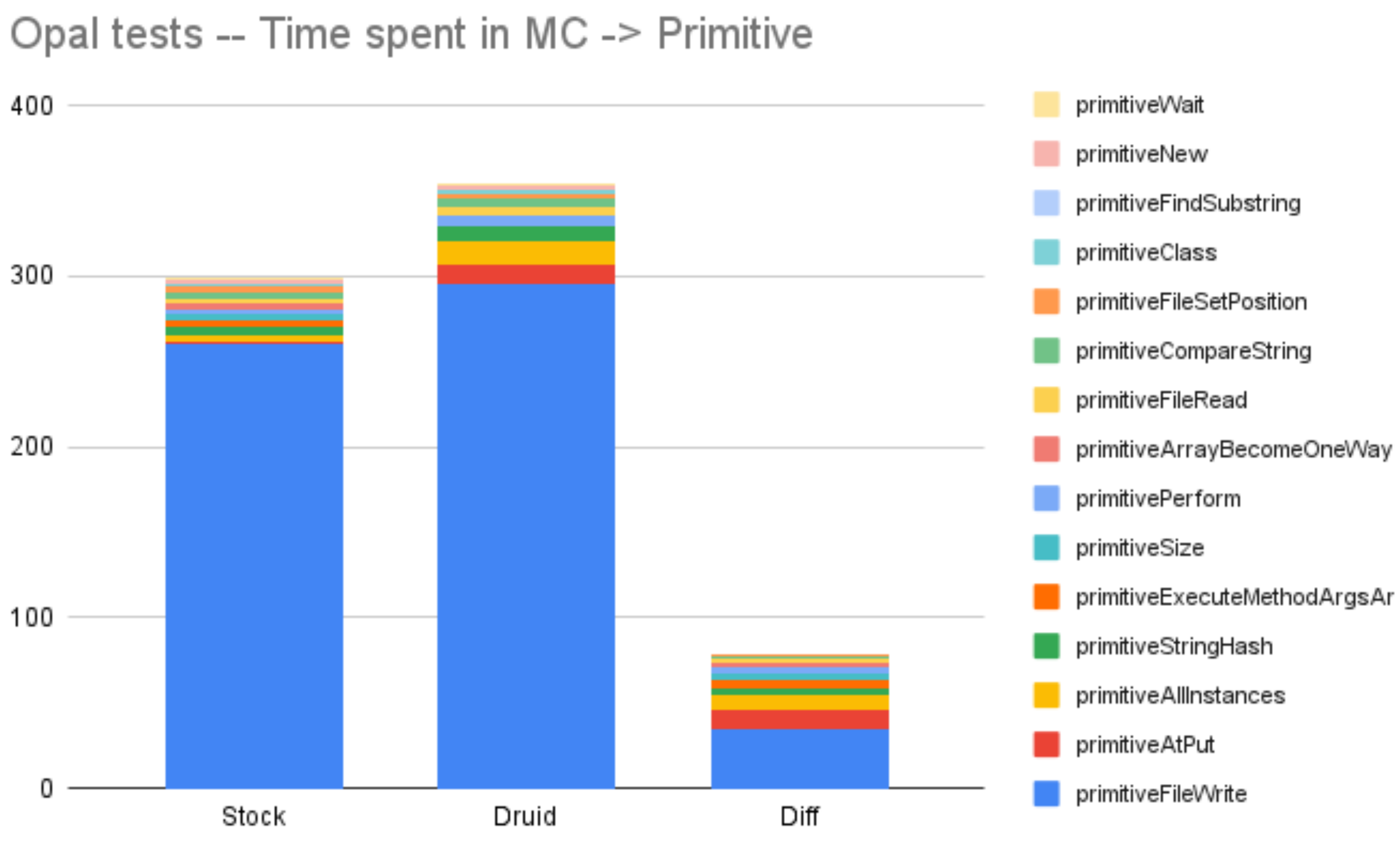
Primitive Samples

Stack Traces

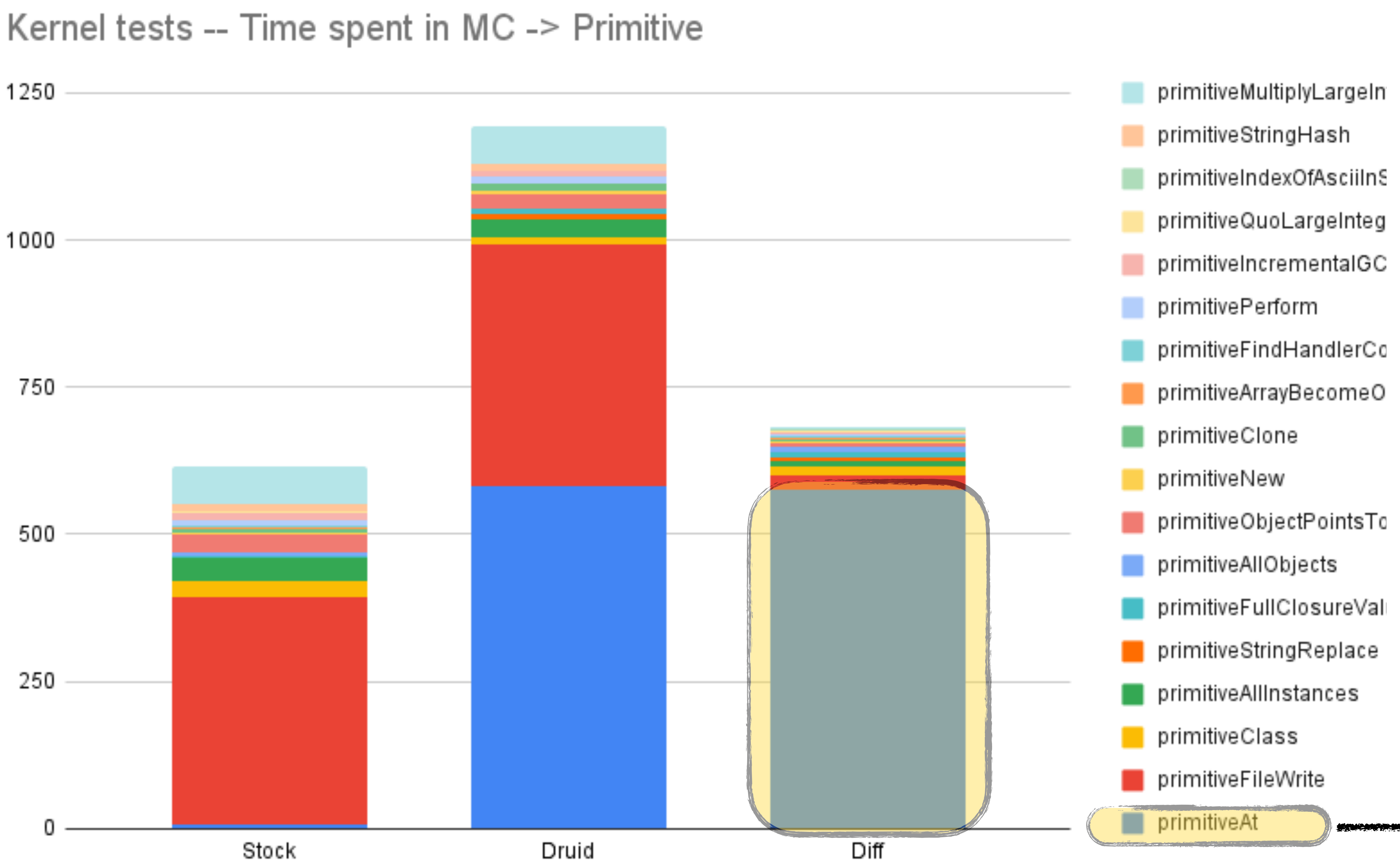
Primitives!

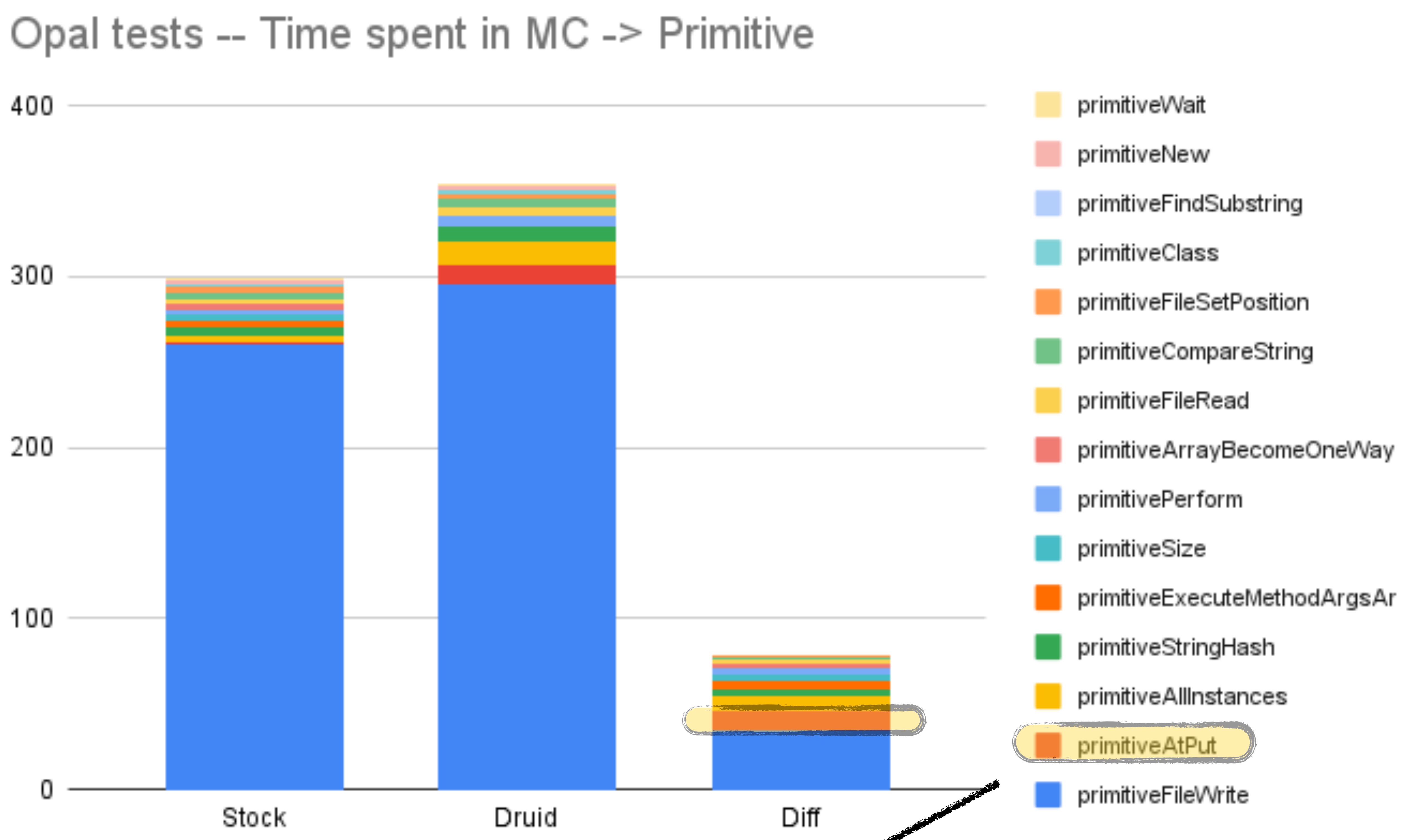
Differential MC->Primitive Profiling


Kernel tests -- Time spent in MC -> Primitive


Opal tests -- Time spent in MC -> Primitive

Differential MC->Primitive Profiling


Kernel tests -- Time spent in MC -> Primitive

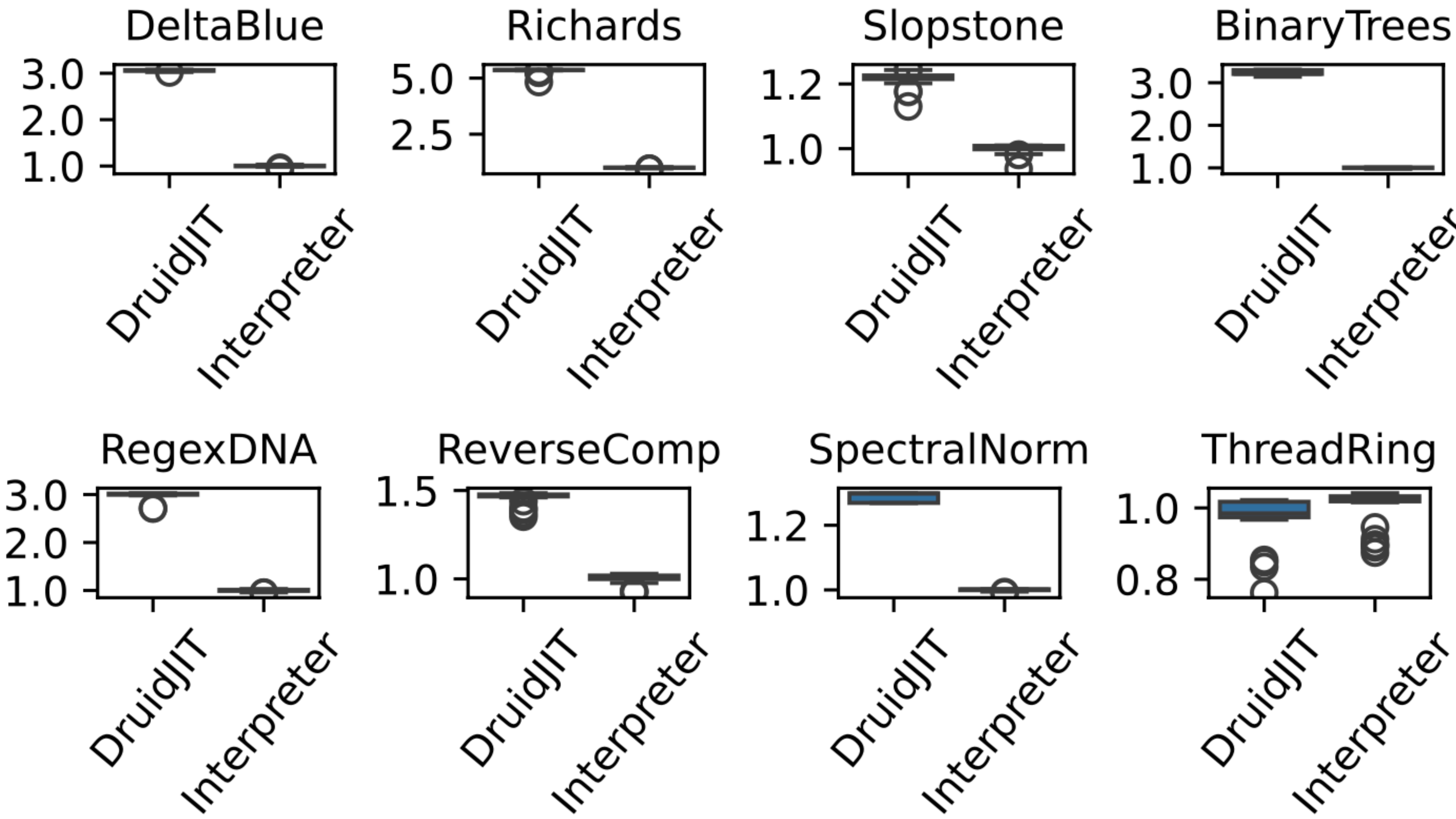

Opal tests -- Time spent in MC -> Primitive

Low-hanging fruits

After Some Bit of “well-placed” Work :)

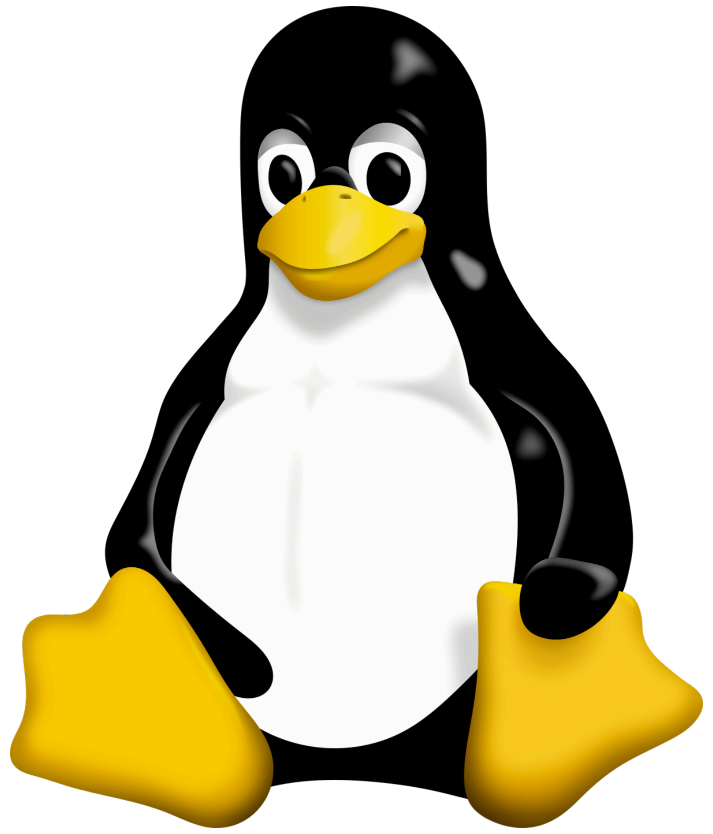
- **2x faster!**
  than interpreter on avg

- Almost there:
  - ~0.7x manual JIT
  - Missing
    - static type predictions
    - peephole optimizations on conditionals



What’s next?

- **Linux integration:**
  - Perf support
  - *Matéo Boury*



- **Tracking** Pharo’s performance:
  - Performance dashboards
  - Benchmark Generation
  - daily, monthly, yearly

Takeaways

- Integrate with tools that do their job well (Instruments, Perf)
- Simple custom tools help debugging *complex VM scenarios*

- Tests first for good behavior
- Bench first for good performance!