Bee Dynamic Metacircular Runtime

… can it be faster than
classic vms?

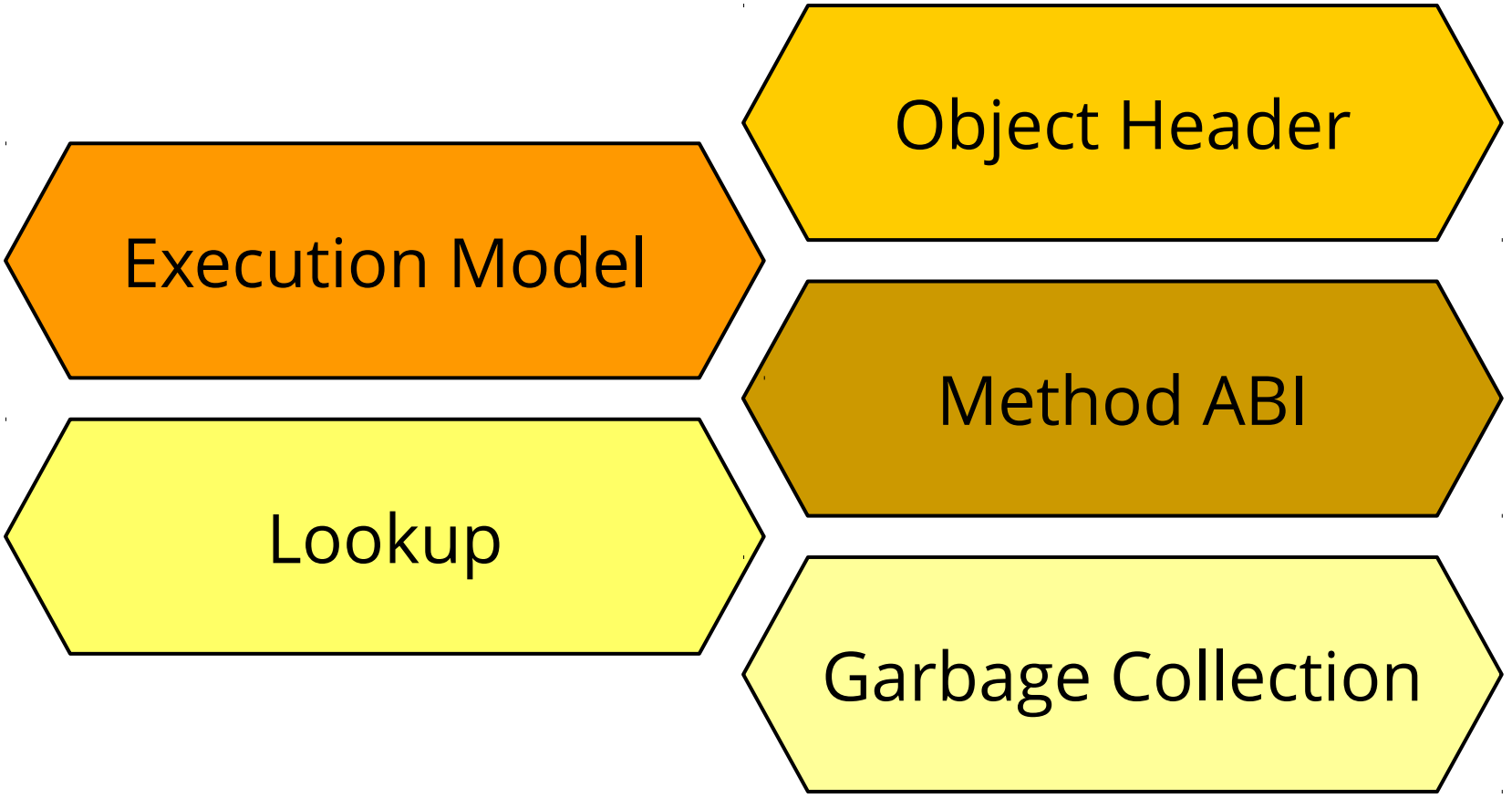
Javier Pimás

CæsarSystems

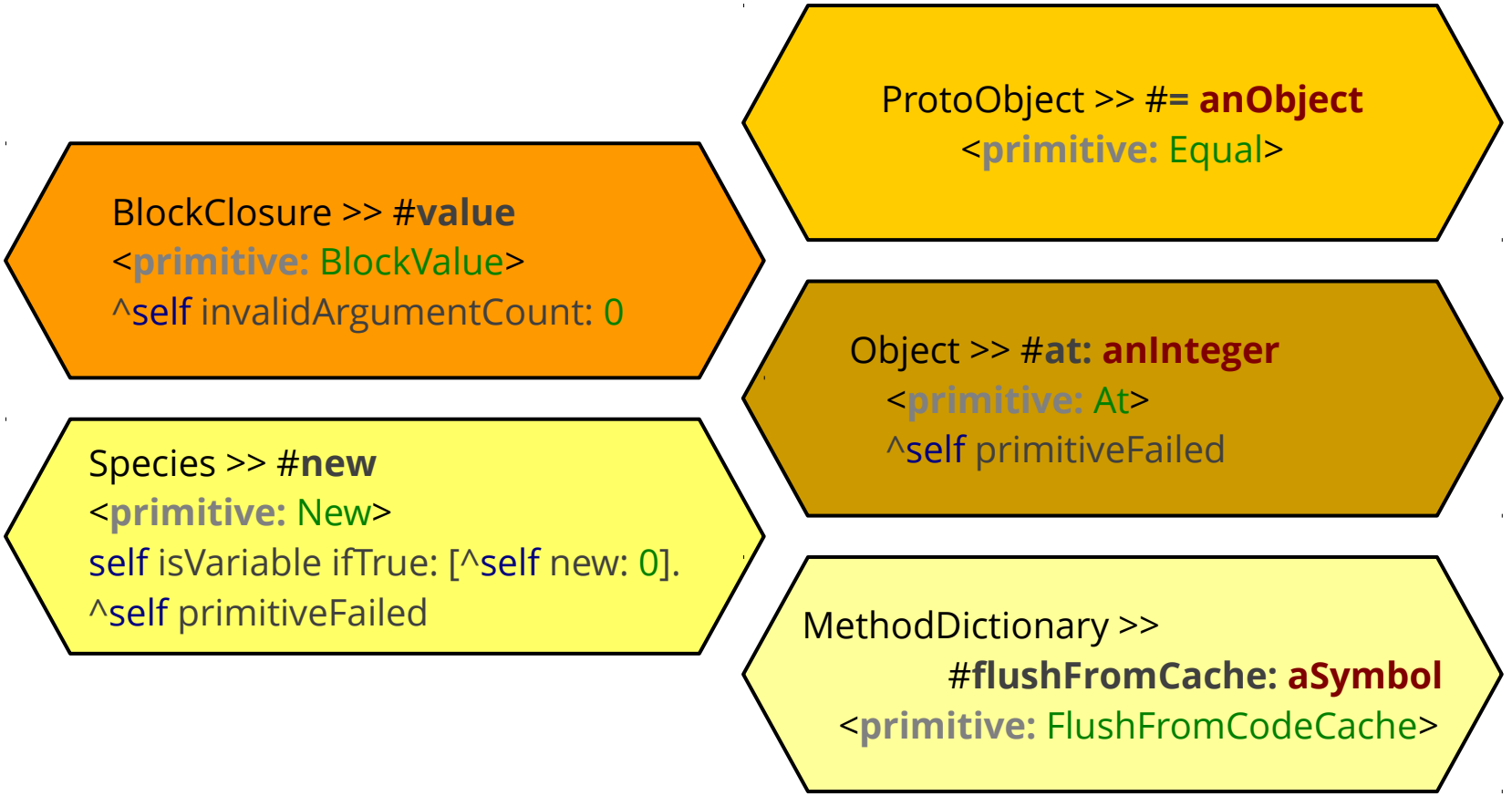
Everything in Smalltalk is an object

not so true!

There are things we don't see nor know that exist



And things we see but cannot look deeper…



How can we break through this barrier?

With Objects :)

Self-hosted VM in 3 little steps

Javier Burroni

Gera Richarte

How can we uncover more objects?

1) Take an *already working* Smalltalk
2) Code a Smalltalk JIT in Smalltalk
3) Conquer the world!

What can we possibly do with just this?

Code Objects

Harness the JIT

```
Array variableSubclass: #CompiledMethod
  instanceVariableNames:
    ' bytecodes nativeCode class
     selector source flags '
```

```
Object subclass: #NativeCode
  instanceVariableNames:
    ' code references '
```

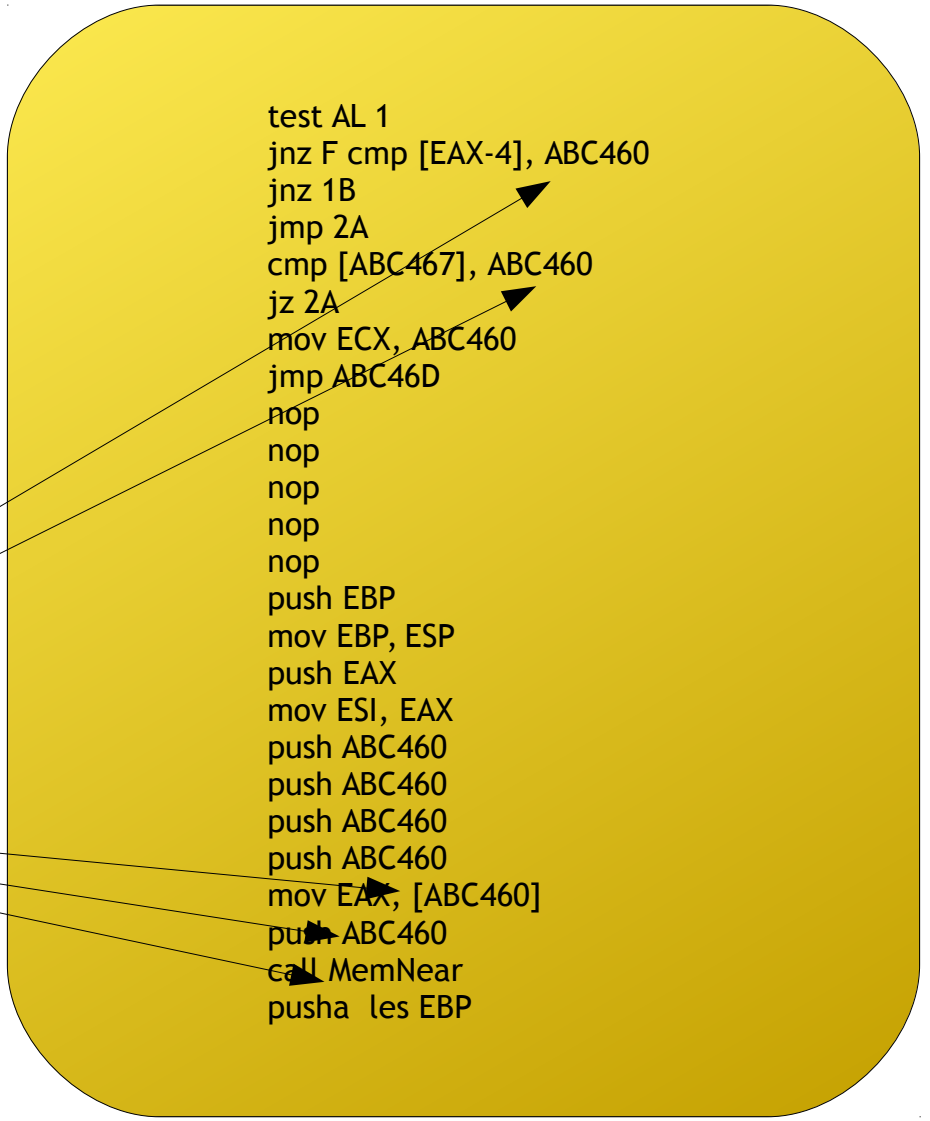
```
<16r0000>  A8 01 75 0B  81 78 FC 60  C4 AB 00 75  0E EB 1B 81
<16r0010>  3D 67 C4 AB  00 60 C4 AB  00 74 0F B9  60 C4 AB 00
<16r0020>  E9 48 C4 AB  00 90 90 90  90 90 55 8B  EC 50 8B
<16r0030>  68 60 C4 AB  00 68 60 C4  AB 00 68 60  C4 AB 00 68
<16r0040>  60 C4 AB 00  A1 60 C4 AB  00 68 60 C4  AB 00 FF 15
<16r0050>  60 C4 AB 00  89 45 F4 A1  60 C4 AB 00  68 60 C4 AB
<16r0060>  00 FF 15 60  C4 AB 00 89  45 F0 A1 60  C4 AB 00 68
<16r0070>  60 C4 AB 00  FF 15 60 C4  AB 00 50 8B  45 F0 68 60
<16r0080>  C4 AB 00 FF  15 60 C4 AB  00 FF 75 08  FF 75 F4 FF
<16r0090>  75 F0 8B 45  E4 68 60 C4  AB 00 FF 15  60 C4 AB 00
<16r00A0>  58 68 60 C4  AB 00 FF 15  60 C4 AB 00  8B 45 F4 68
<16r00B0>  60 C4 AB 00  FF 15 60 C4  AB 00 89 45  EC FF 35 60
<16r00C0>  C4 AB 00 8B  45 EC 68 60  C4 AB 00 FF  15 60 C4 AB
<16r00D0>  00 50 8B 45  E8 68 60 C4  AB 00 FF 15  60 C4 AB 00
<16r00E0>  83 C4 04 89  06 50 8B 45  EC 68 60 C4  AB 00 FF 15
<16r00F0>  60 C4 AB 00  50 FF 75 EC  8B 45 E8 68  60 C4 AB 00
<16r0100>  FF 15 60 C4  AB 00 83 C4  04 8B 45 F4  68 60 C4 AB
<16r0110>  00 FF 15 60  C4 AB 00 89  46 04 8B C6  68 60 C4 AB
<16r0120>  00 FF 15 60  C4 AB 00 8B  C6 8B E5 5D  8B 75 FC C2
<16r0130>  04 00
```

Code Objects

Harness the JIT

```
Array variableSubclass: #CompiledMethod
  instanceVariableNames:
    ' bytecodes nativeCode class
     selector source flags '
```

```
Object subclass: #NativeCode
  instanceVariableNames:
    ' code references '
```



Play with messages

Harness the JIT: underprimitives

```
behavior
    ^self _isSmallInteger
        ifTrue: [SmallInteger methodDictionaries]
        ifFalse: [self _basicAt: 0]
```

```
assembleTestSmallInteger
    | integer |
    integer := assembler testAndJumpIfInteger.
    self loadObject: false.
    assembler uncontionalSkip: [
        assembler jumpDestinationFor: integer.
        self loadObject: true]
```

```
assembleBasicAt
    | nonInteger |
    nonInteger := assembler convertArgToIntOrJump
    assembler
        framelessSlotAtArg;
        jumpDestinationFor: nonInteger
```

Play with lookup

Harness the JIT: lookup

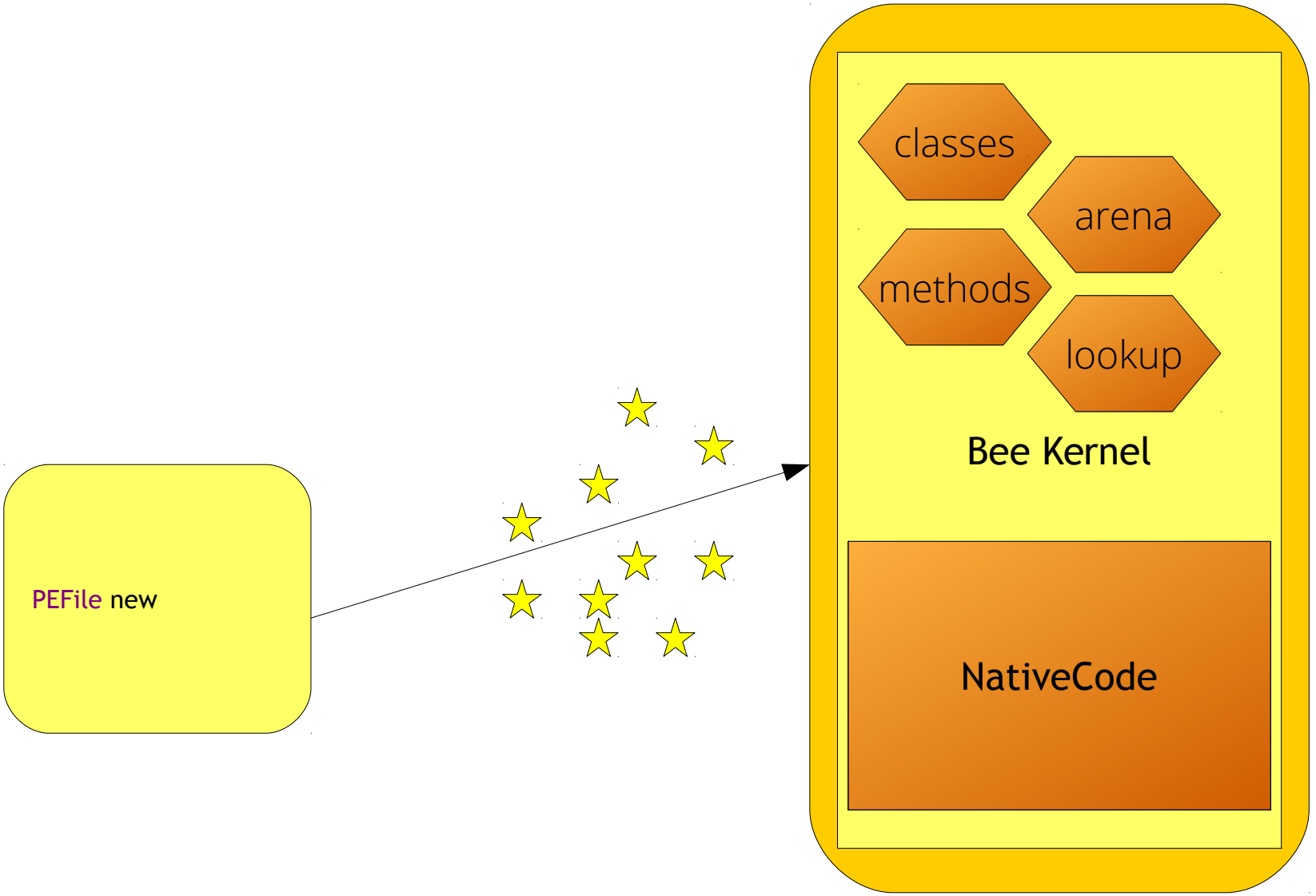
anObject printString

```
mov     eax, anObject ; receiver
push    offset #printString ; selector
call    Object__lookupAndInvokeNativeCode
```

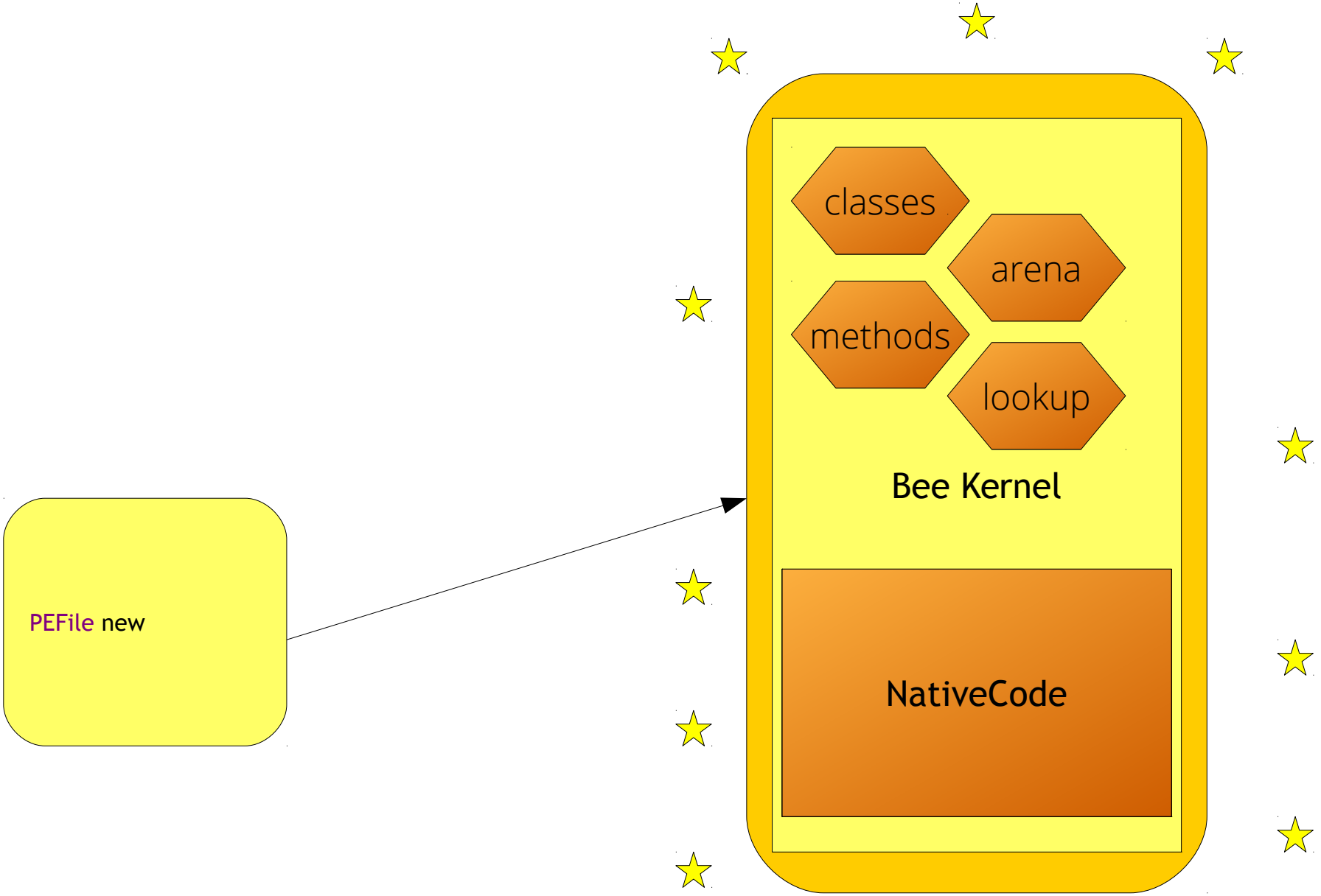
Lookup Nativizer

```
Object>>#_lookupAndInvoke: aSymbol
    | cm nativeCode |
    cm := self _lookup: aSymbol.
    cm == nil ifTrue: [ self doesNotUnderstandSelector: aSymbol ].
    cm prepareForExecution; patchClassCheckTo: self behavior.
    nativeCode := cm nativeCode.
    self
        _patchIndirectCallSiteTo: nativeCode;
        _transferControlSkippingPrologue: nativeCode;
```

Some magic

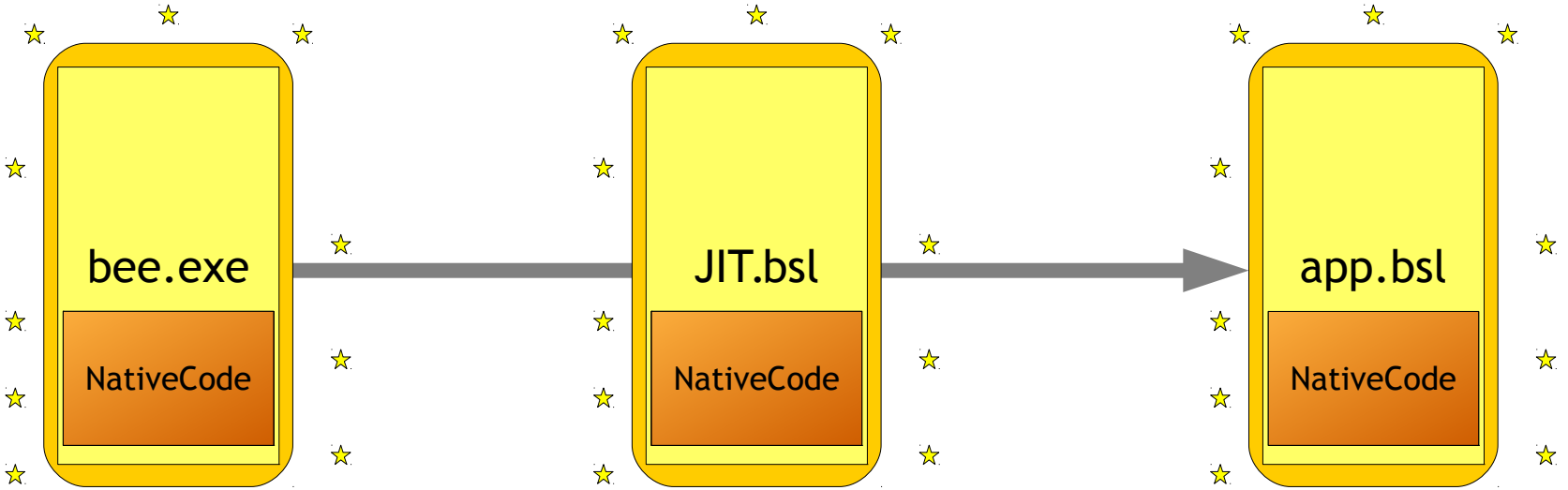

classes
arena
methods
lookup
Bee Kernel
NativeCode
PEFile new

Some magic

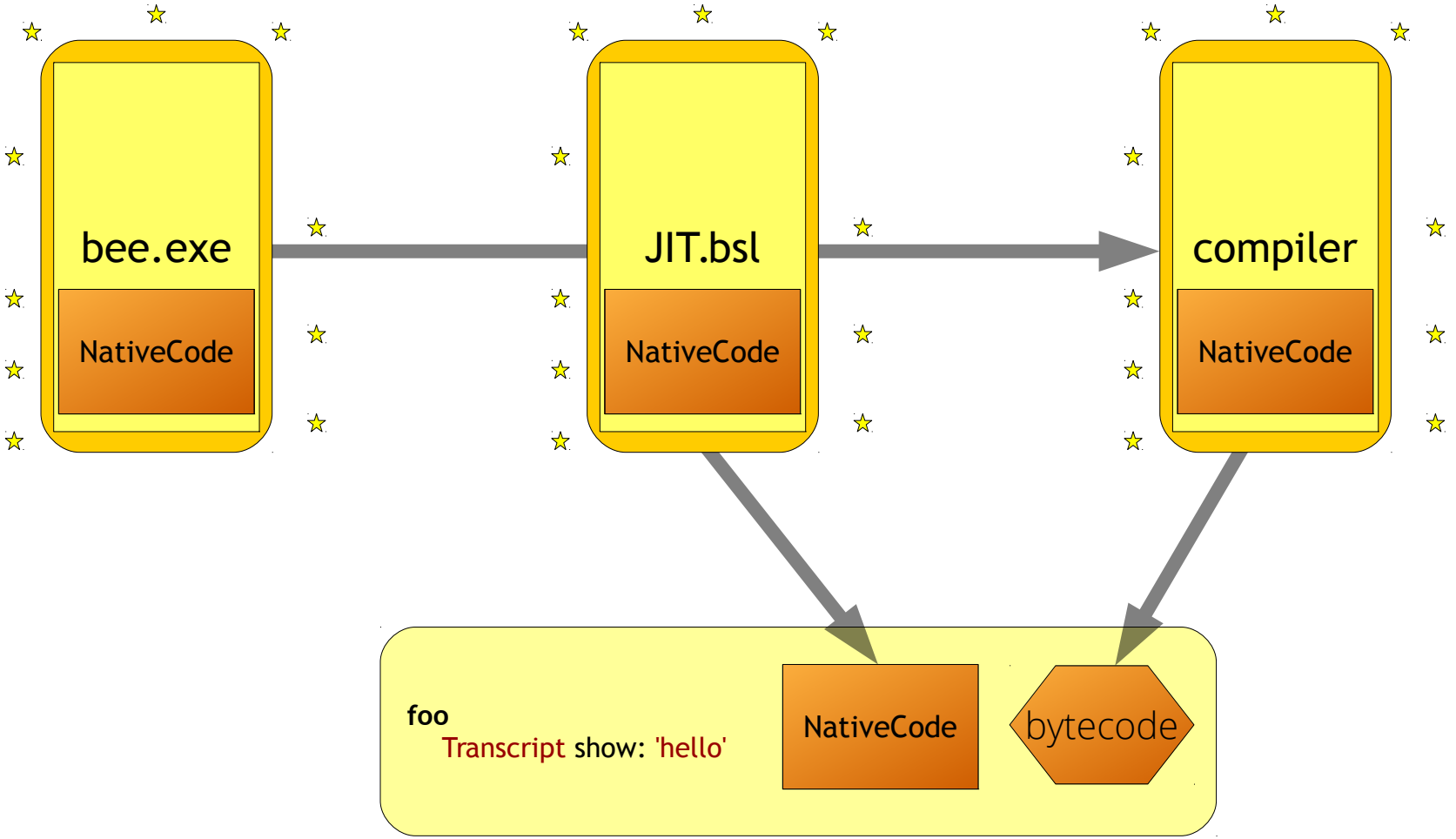

classes
arena
methods
lookup
Bee Kernel
NativeCode
PEFile new

Many small libraries

Adding nativizer support

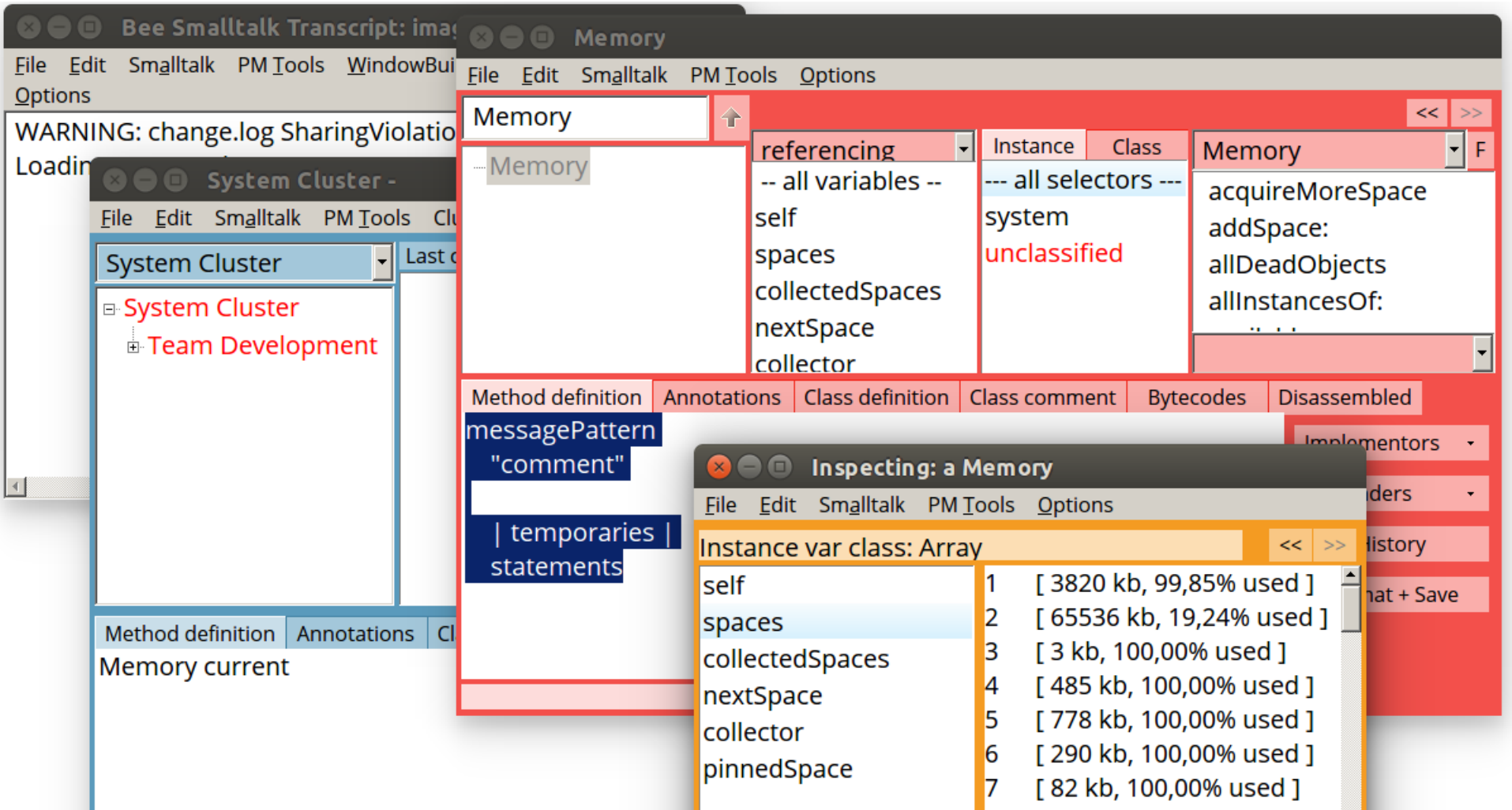


Adding compiler support

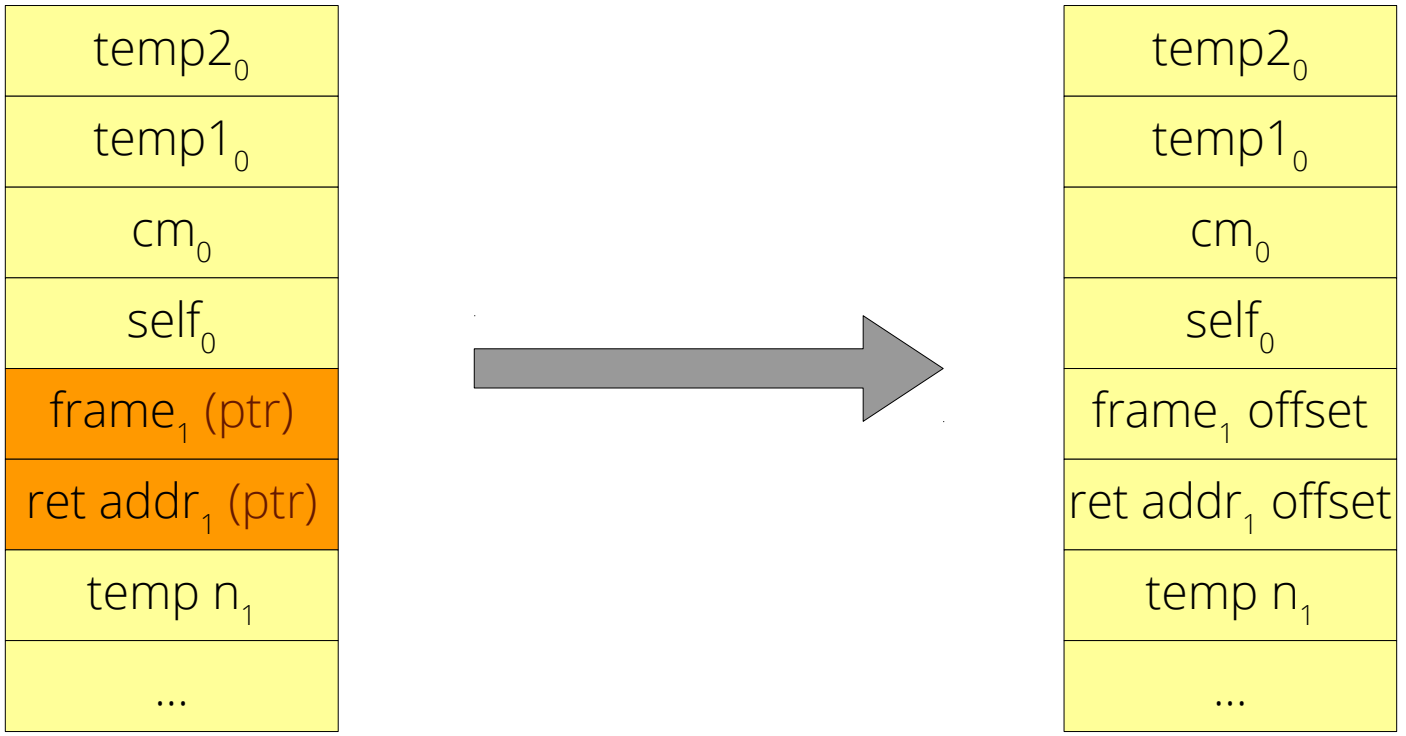


User Interface

Already in image, wraps native UI by FFI



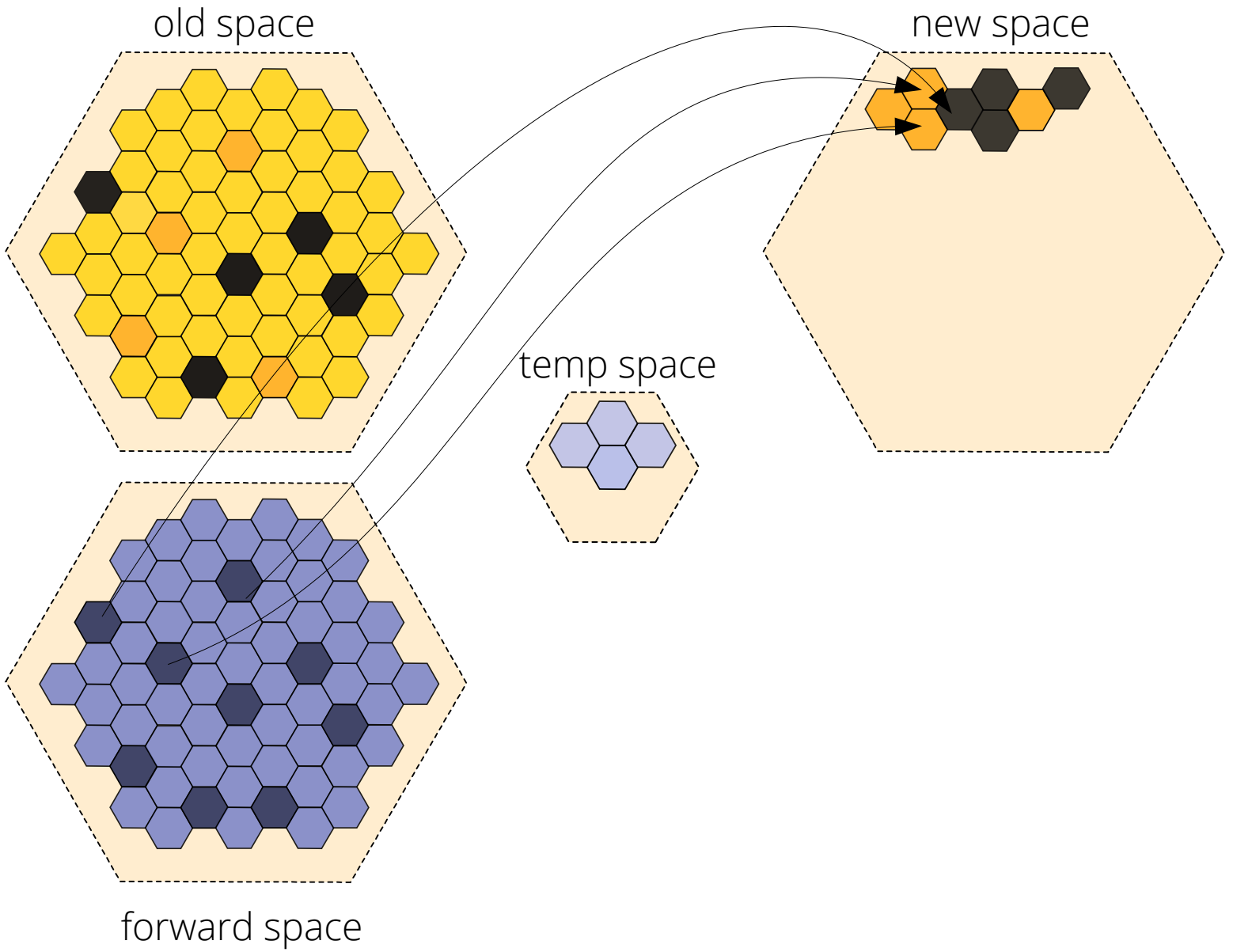
Reifying the native stack



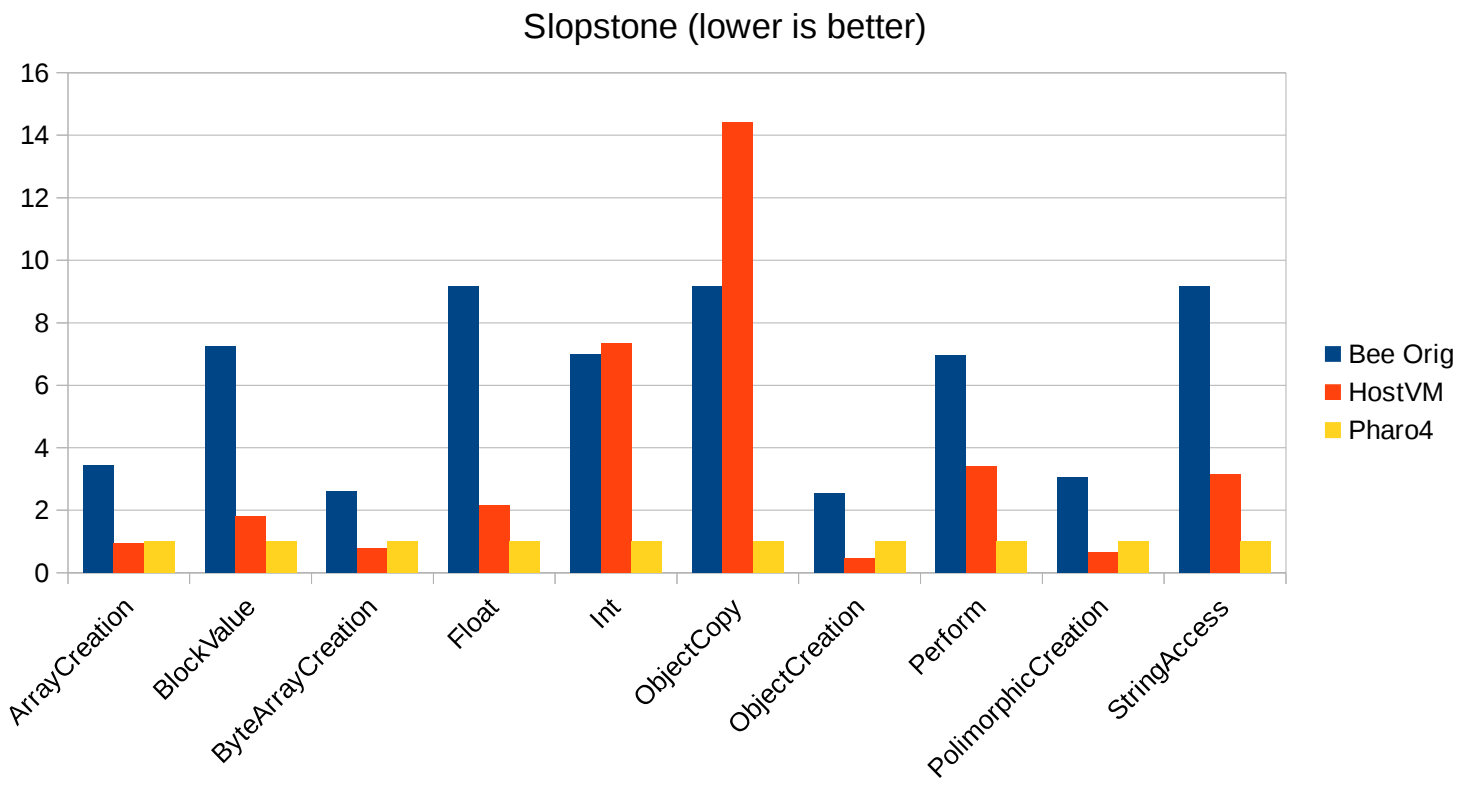
A native stack

A reified stack

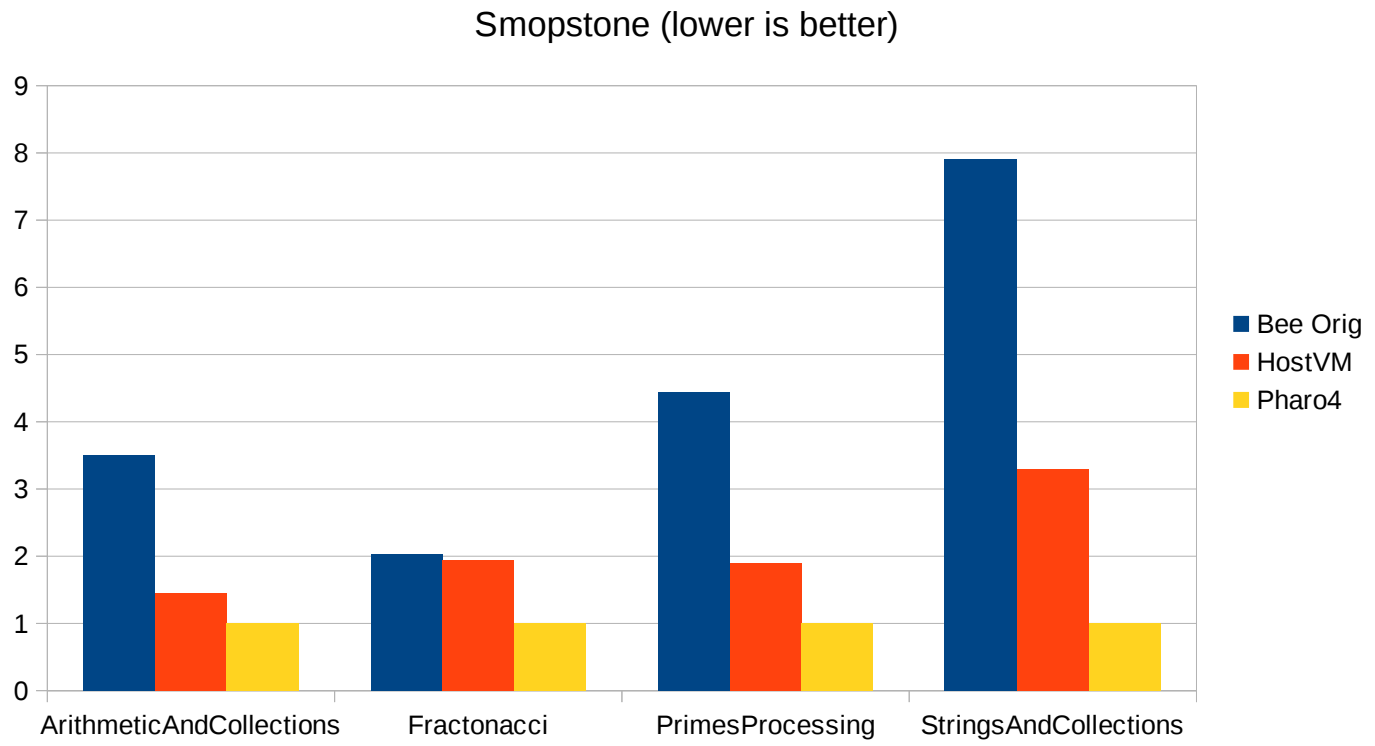
A copying collector



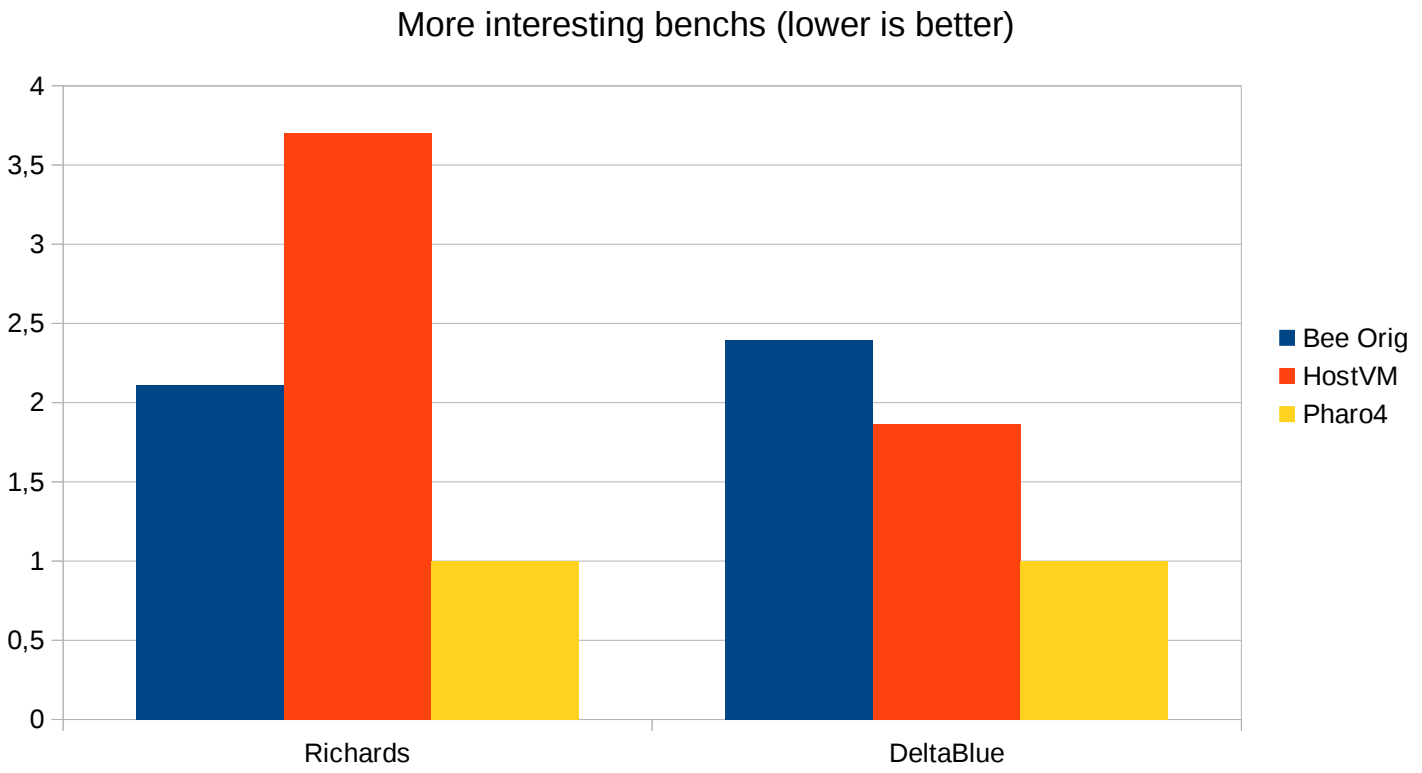
Performance, one year ago



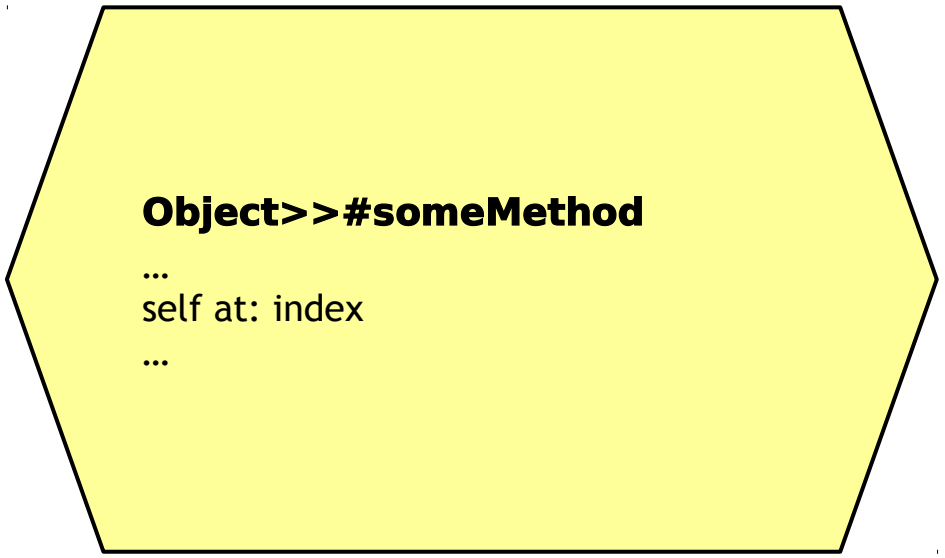
Performance, one year ago



Performance, one year ago



Experiment: play with lookup



Experiment: play with lookup

Uninitialized send:

mov edx, #at:
call lookupAndInvoke:

Monomorphic cached send:

mov edx, #at:
call Object>>#at:

Polymorphic cached send:

mov edx, #at:
call aPicStub_at:

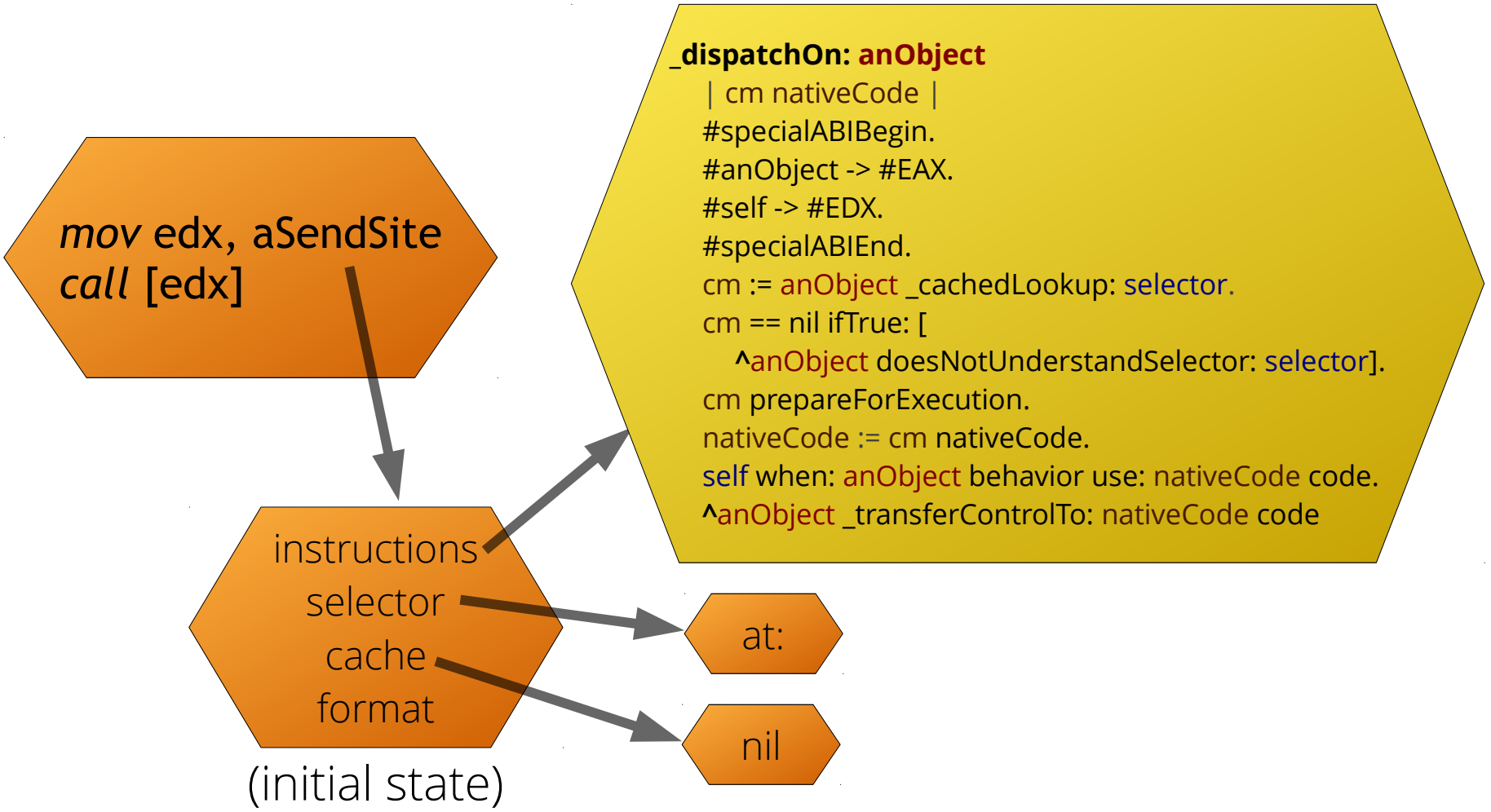
Monomorphic IC prologue

```
  test   al, 1
  jnz    smi_receiver
hdr_cls_chk:
  cmp    [eax-4], DW Dictionary_b
  jnz    lookupAndInvokeCreatingStub
  jmp    at:put:_start
smi_receiver:
  cmp    hdr_cls_chk+3, DW SmallInteger_b
  jz     lookupAndInvokeCreatingStub
```

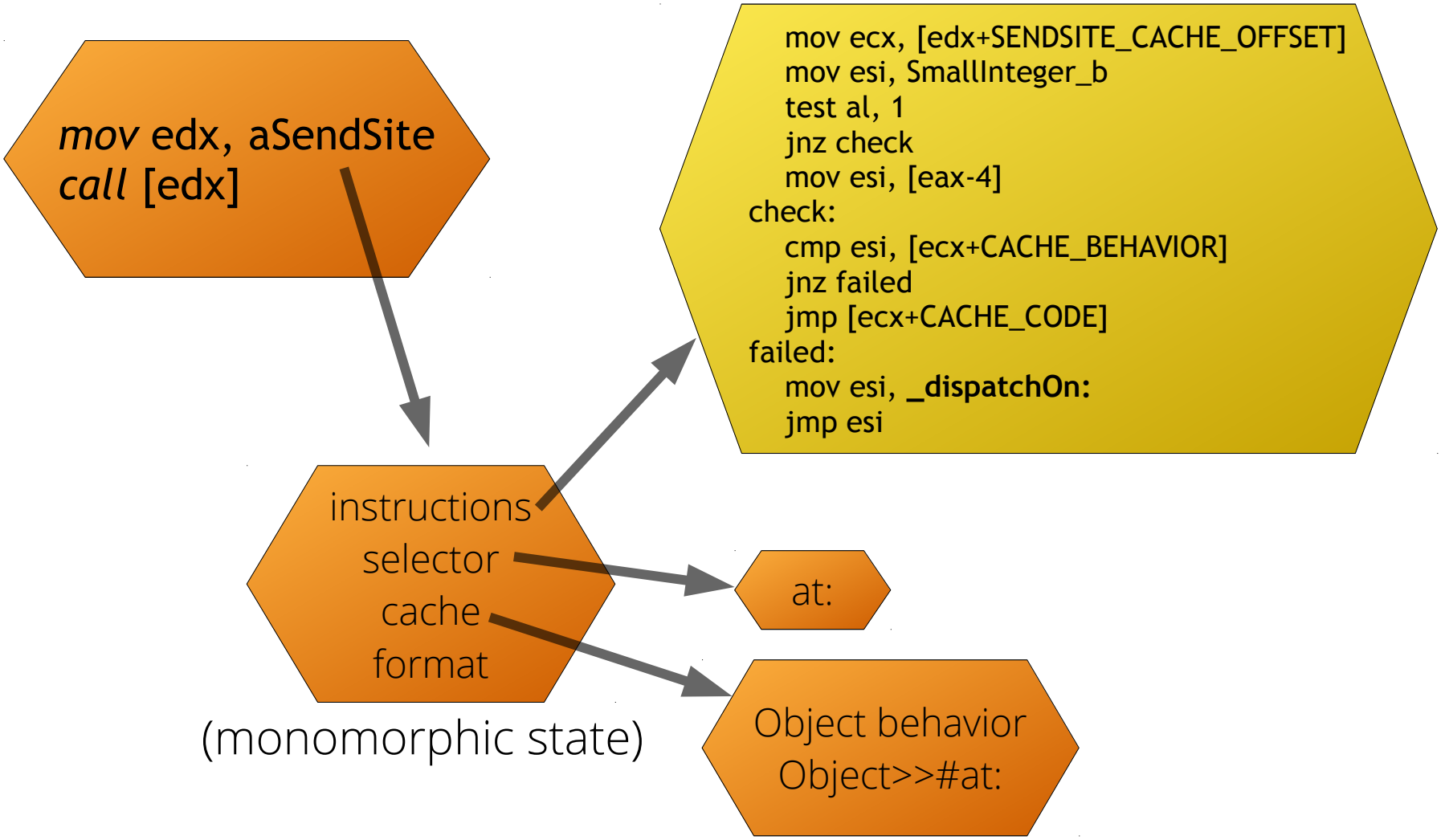
Polymorphic IC prologue

```
aPicStub_at:
   test   al, 1
   jz     entry_1
   jmp    do_lookup
entry_1:
   mov    ecx, [eax-4]
   cmp    ecx, Object_b
   jnz    entry_2
   jmp    do_lookup
entry_2:
   cmp    ecx, 0
   jnz    entry_3
   jmp    do_lookup
...
do_lookup:
   mov    ecx, thisPICStub
   jmp    _lookupAndInvoke:patching:
```

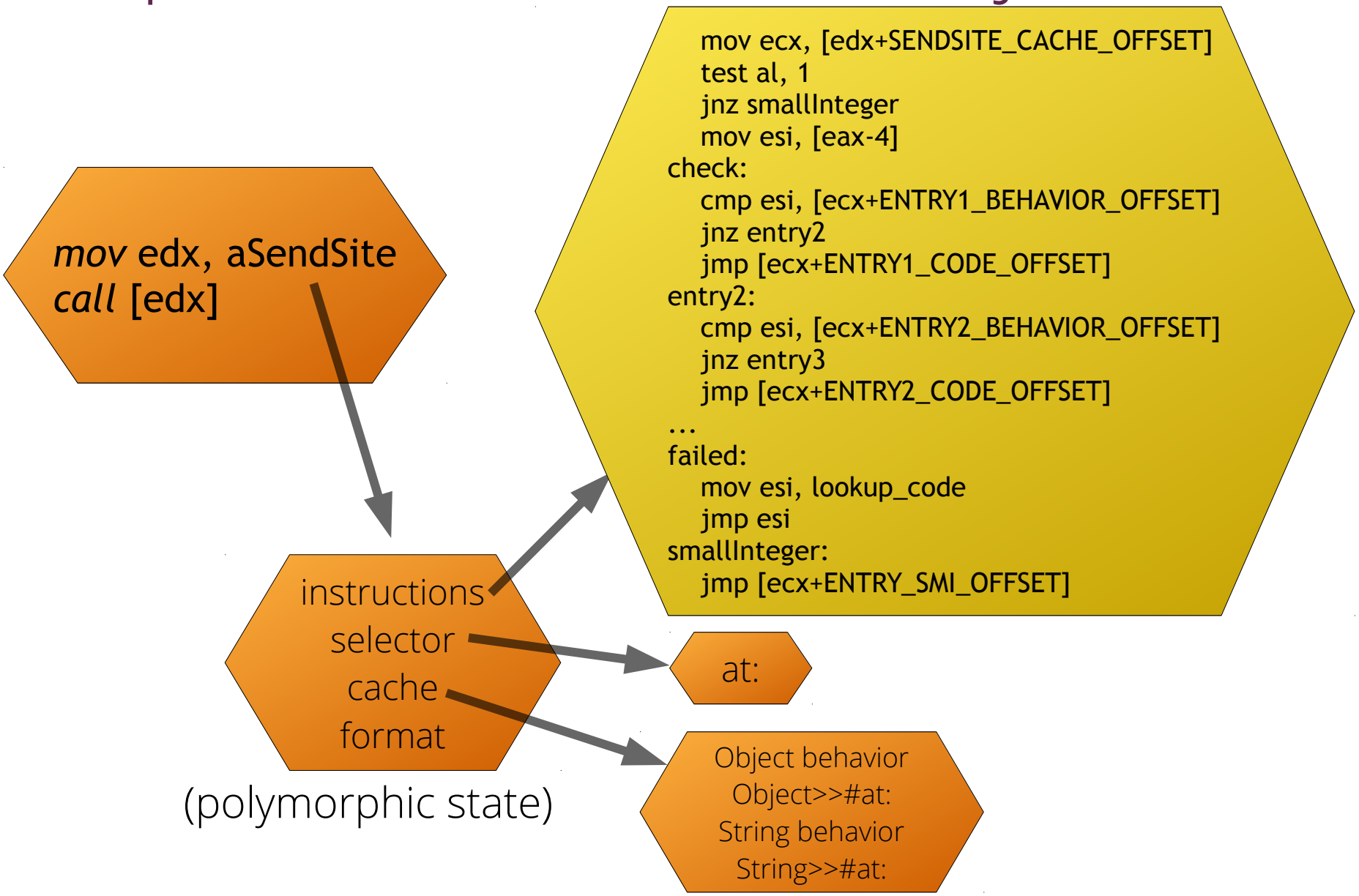
Experiment: SendSite objects



Experiment: SendSite objects



Experiment: SendSite objects



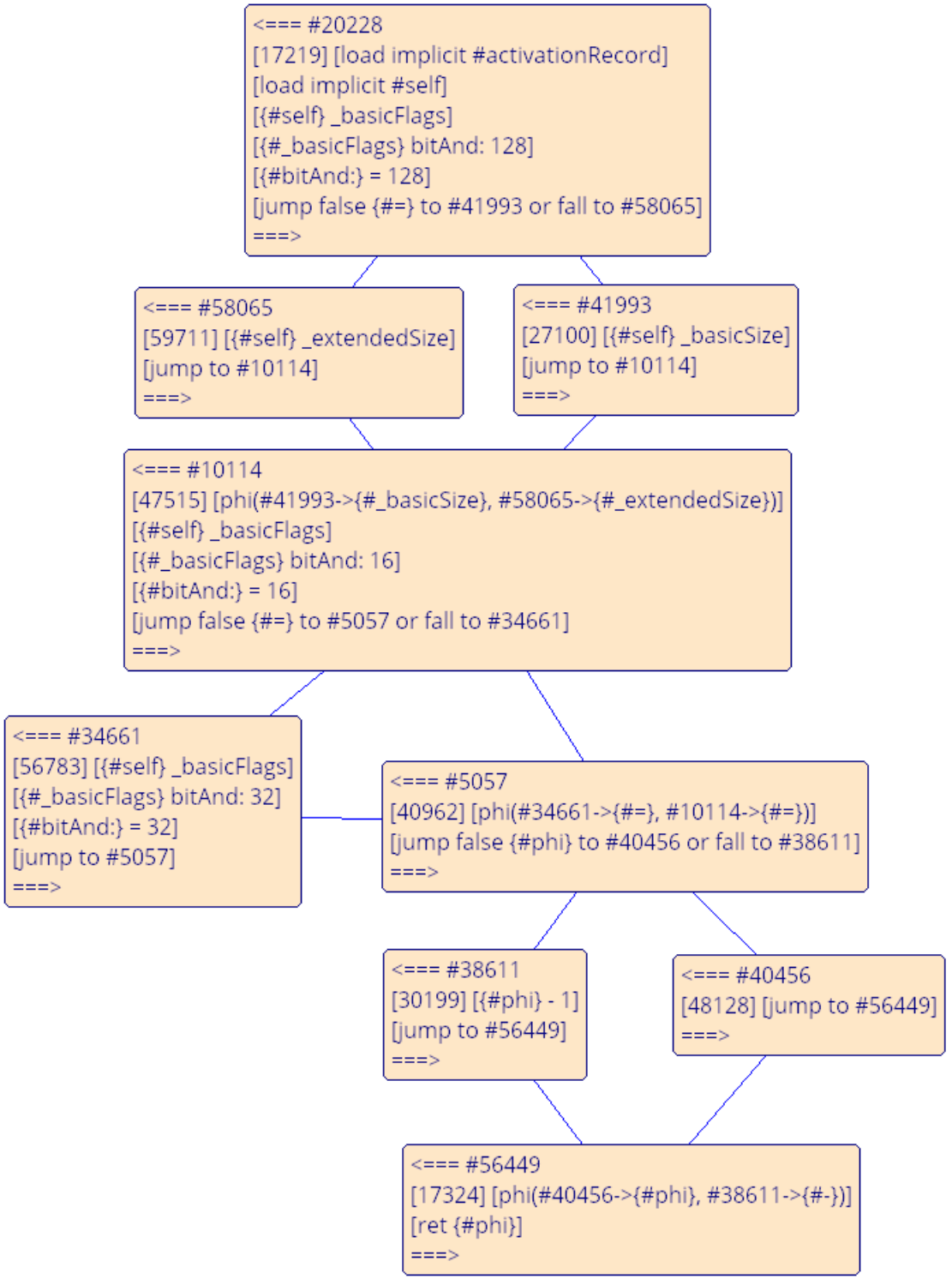
Experiment: SendSite objects

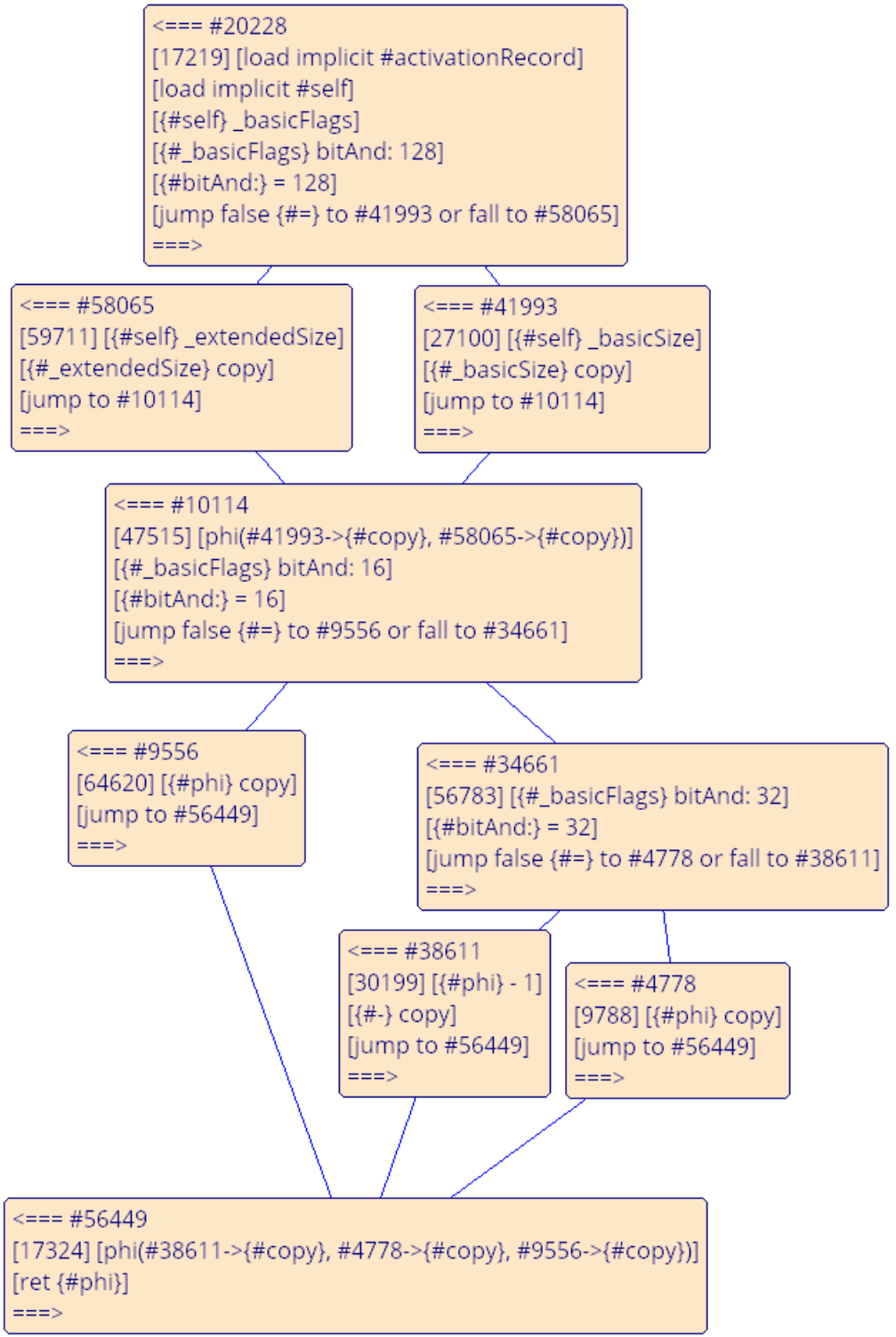
```
when: aBehavior use: code
    cache == nil
        ifTrue: [
            self isStaticSend
                ifTrue: [instructions := code]
                ifFalse: [
                    self monomorphicMap: aBehavior to: code]]
        ifFalse: [self polymorphicMap: aBehavior to: code]
```

```
monomorphicMap: aBehavior to: code
    instructions := self monomorphicStub.
    cache := self takeNextFreeMIC.
    cache objectAtValid: 1 put: aBehavior.
    cache objectAtValid: 2 put: code
```

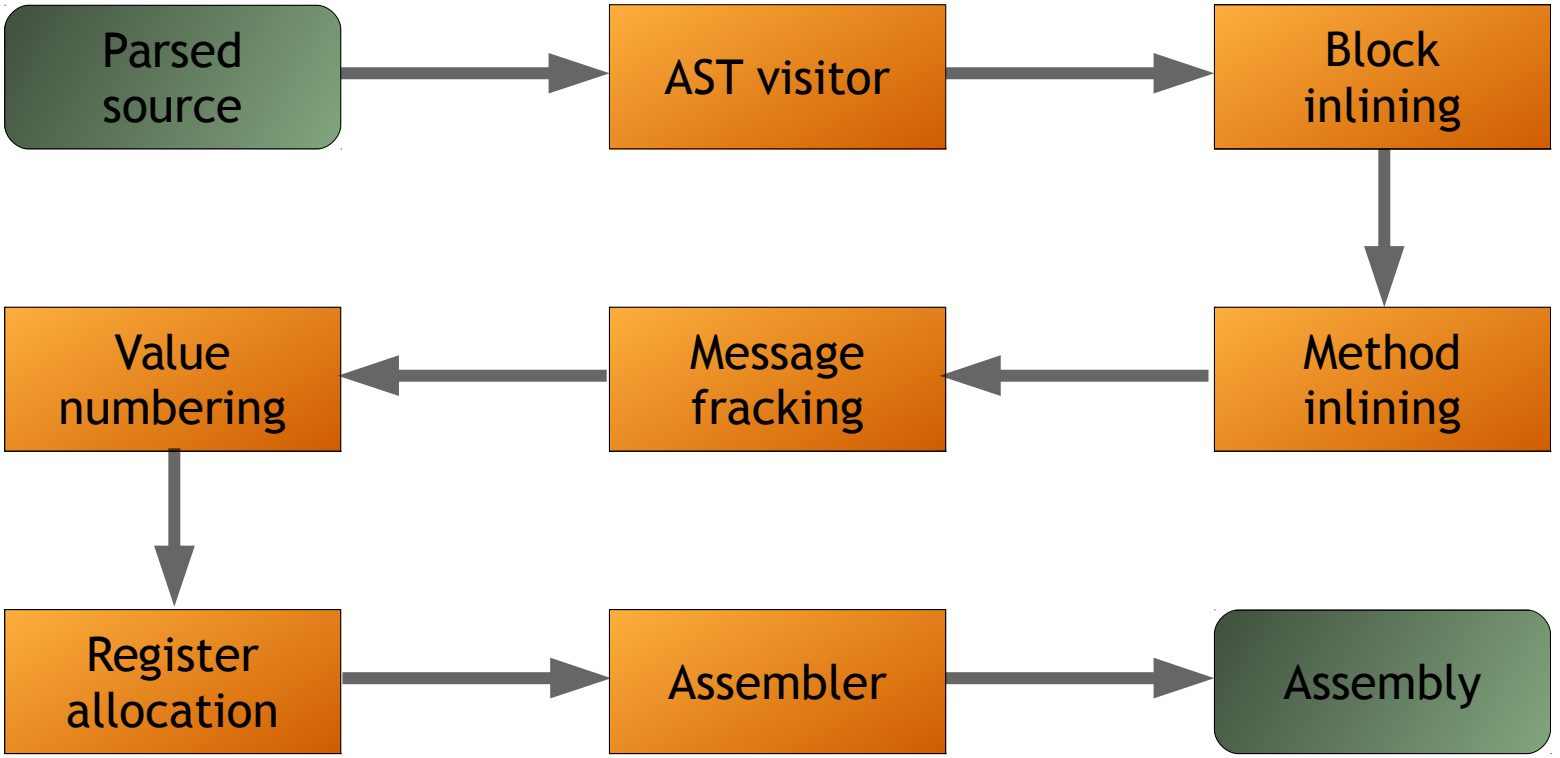
Optimizing Method Nativizer

1. CFG of basic blocks
2. Double-linked list of SSA instructions
3. Statements get lowered in different stages
4. Instructions are untyped (everything is “an object”), for now

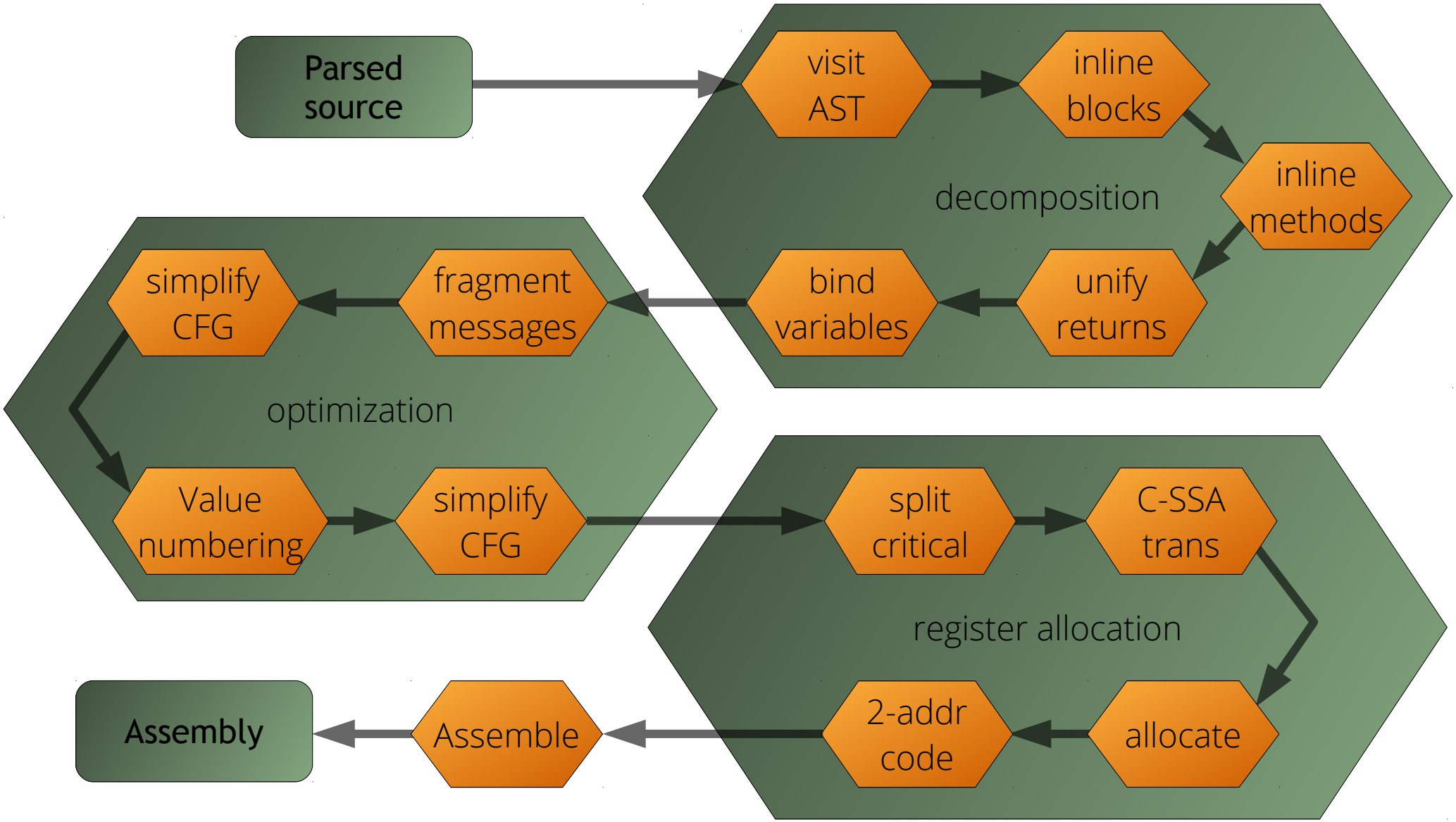

<=== #20228
[17219] [load implicit #activationRecord]
[load implicit #self]
[{#self} _basicFlags]
[{#_basicFlags} bitAnd: 128]
[{#bitAnd:} = 128]
[jump false {#=} to #41993 or fall to #58065]
===>
<=== #58065
[59711] [{#self} _extendedSize]
[jump to #10114]
===>
<=== #41993
[27100] [{#self} _basicSize]
[jump to #10114]
===>
<=== #10114
[47515] [phi(#41993->{#_basicSize}, #58065->{#_extendedSize})]
[{#self} _basicFlags]
[{#_basicFlags} bitAnd: 16]
[{#bitAnd:} = 16]
[jump false {#=} to #5057 or fall to #34661]
===>
<=== #34661
[56783] [{#self} _basicFlags]
[{#_basicFlags} bitAnd: 32]
[{#bitAnd:} = 32]
[jump to #5057]
===>
<=== #5057
[40962] [phi(#34661->{#=}, #10114->{#=})]
[jump false {#phi} to #40456 or fall to #38611]
===>
<=== #38611
[30199] [{#phi} - 1]
[jump to #56449]
===>
<=== #40456
[48128] [jump to #56449]
===>
<=== #56449
[17324] [phi(#40456->{#phi}, #38611->{#-})]
[ret {#phi}]
===>


<=== #20228
[17219] [load implicit #activationRecord]
[load implicit #self]
[{#self} _basicFlags]
[{#_basicFlags} bitAnd: 128]
[{#bitAnd:} = 128]
[jump false {#=} to #41993 or fall to #58065]
===>
<=== #58065
[59711] [{#self} _extendedSize]
[{#_extendedSize} copy]
[jump to #10114]
===>
<=== #41993
[27100] [{#self} _basicSize]
[{#_basicSize} copy]
[jump to #10114]
===>
<=== #10114
[47515] [phi(#41993->{#copy}, #58065->{#copy})]
[{#_basicFlags} bitAnd: 16]
[{#bitAnd:} = 16]
[jump false {#=} to #9556 or fall to #34661]
===>
<=== #9556
[64620] [{#phi} copy]
[jump to #56449]
===>
<=== #34661
[56783] [{#_basicFlags} bitAnd: 32]
[{#bitAnd:} = 32]
[jump false {#=} to #4778 or fall to #38611]
===>
<=== #38611
[30199] [{#phi} - 1]
[{#-} copy]
[jump to #56449]
===>
<=== #4778
[9788] [{#phi} copy]
[jump to #56449]
===>
<=== #56449
[17324] [phi(#38611->{#copy}, #4778->{#copy}, #9556->{#copy})]
[ret {#phi}]
===>

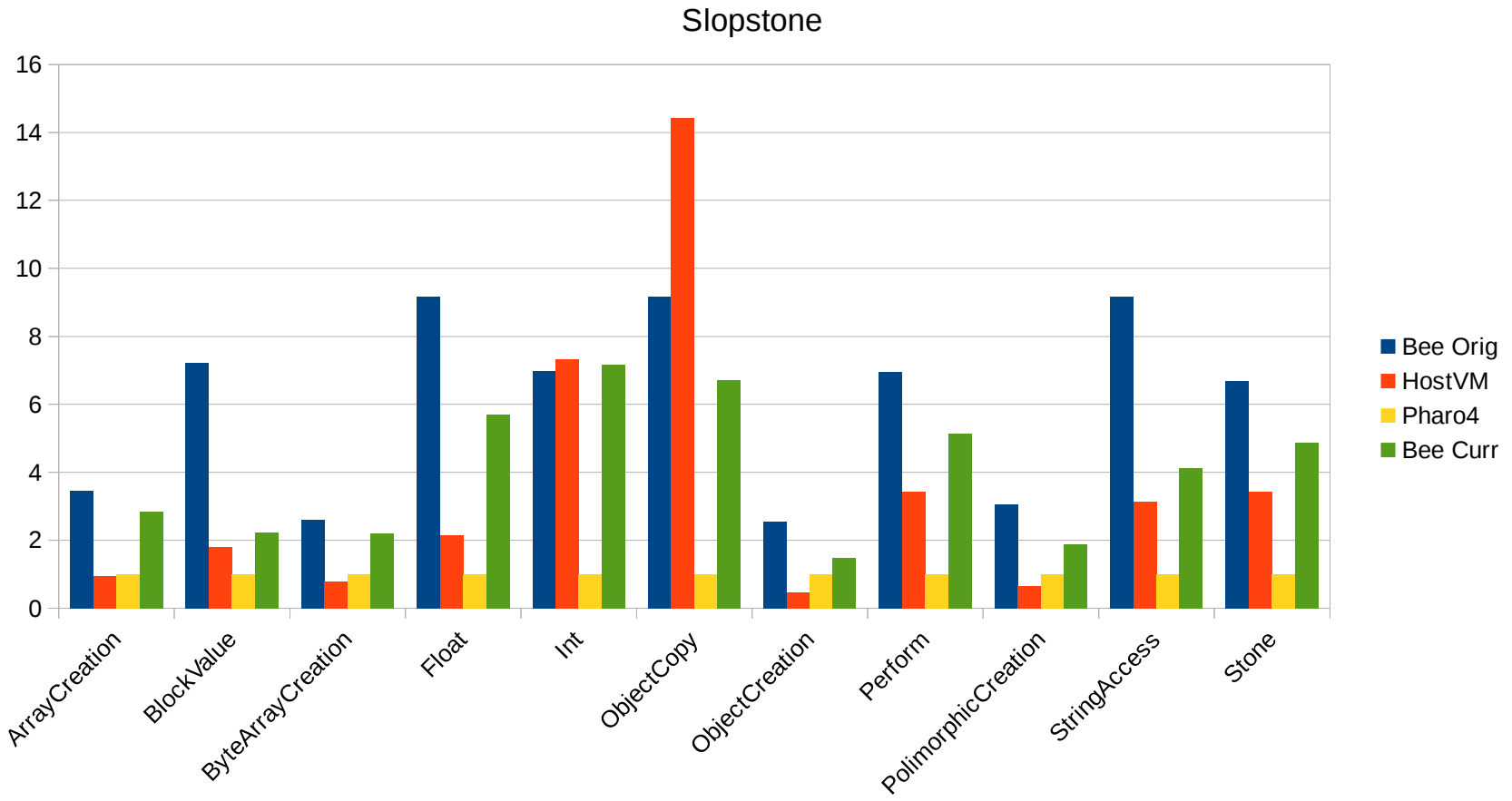
Optimizing Method Nativizer (old)



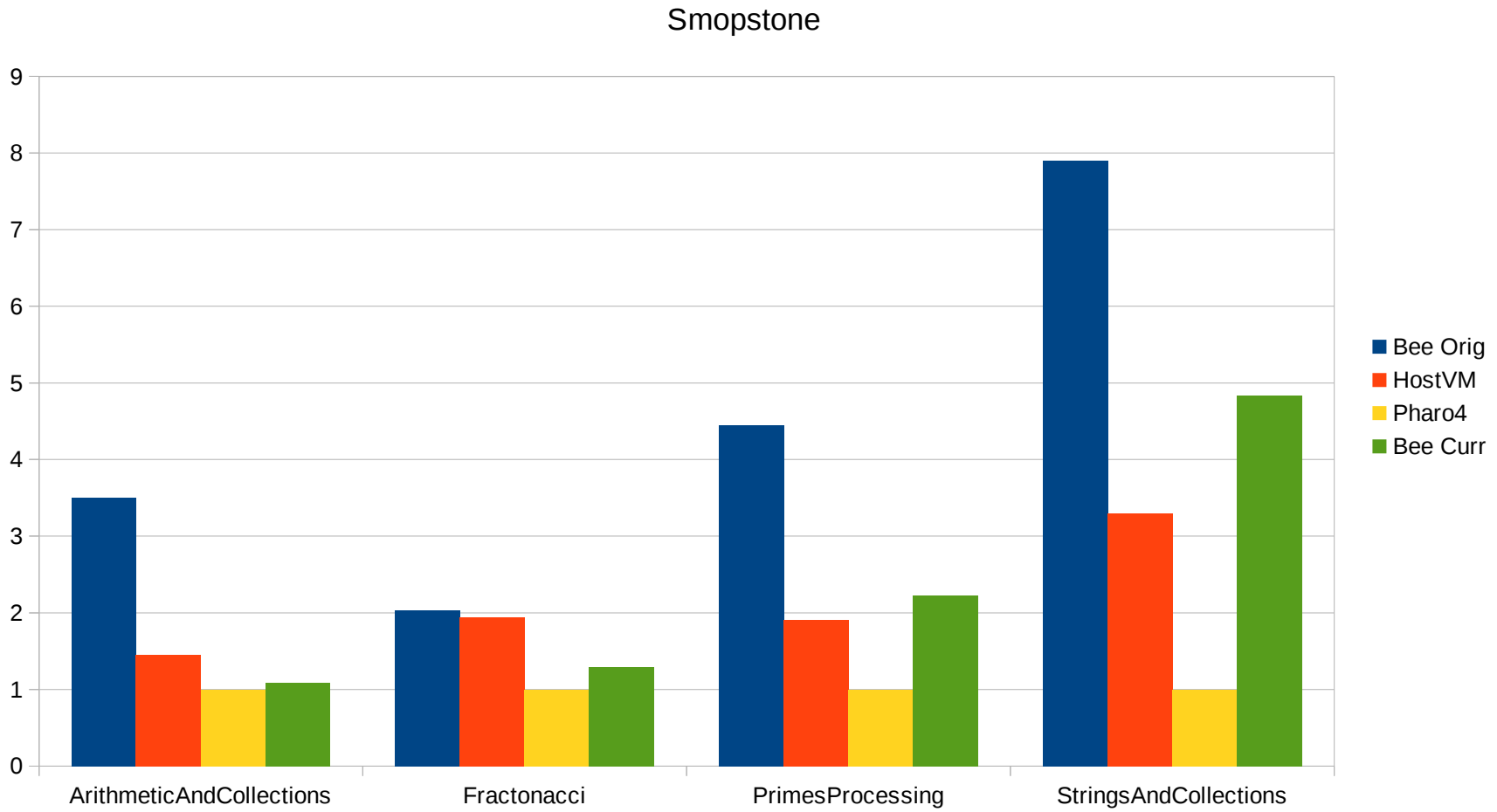
Optimizing Method Nativizer



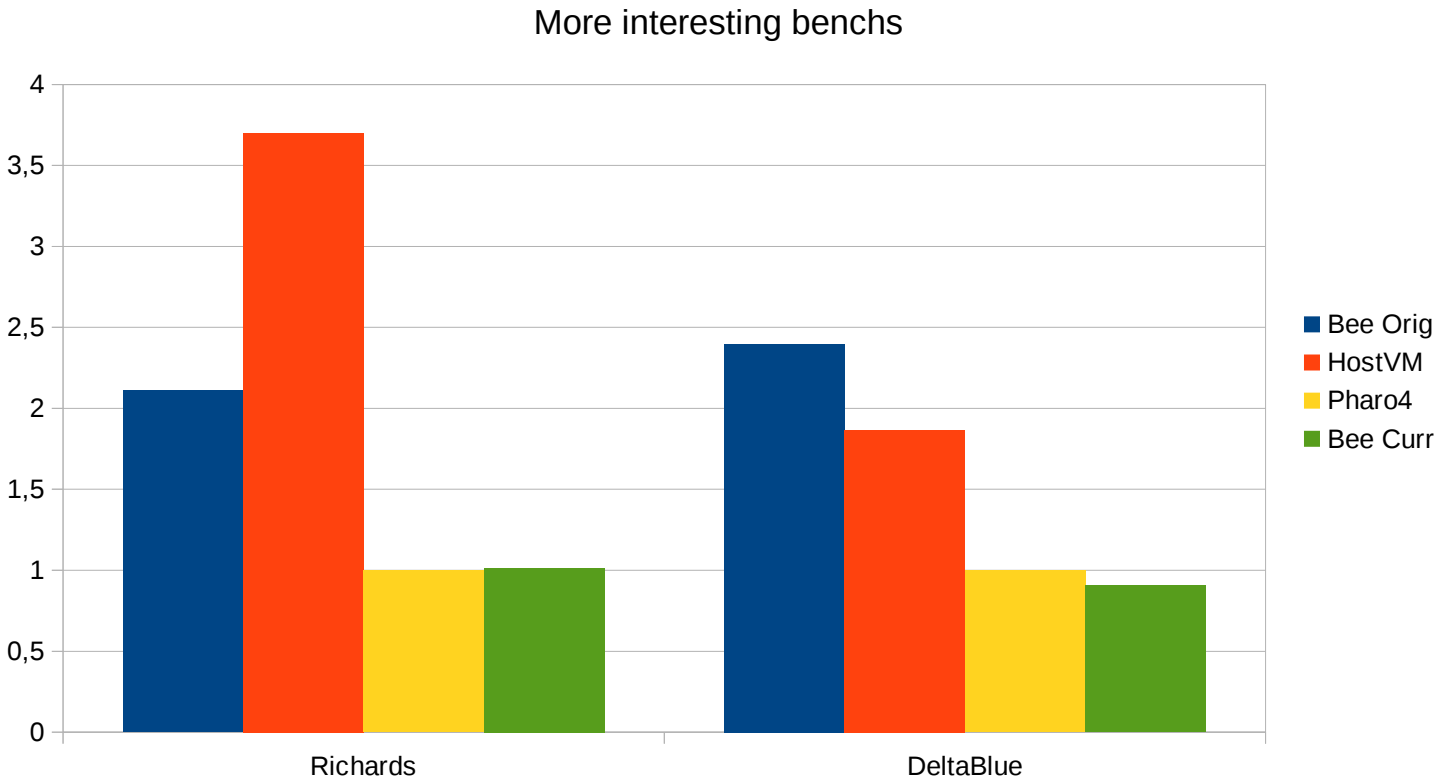
Performance, today



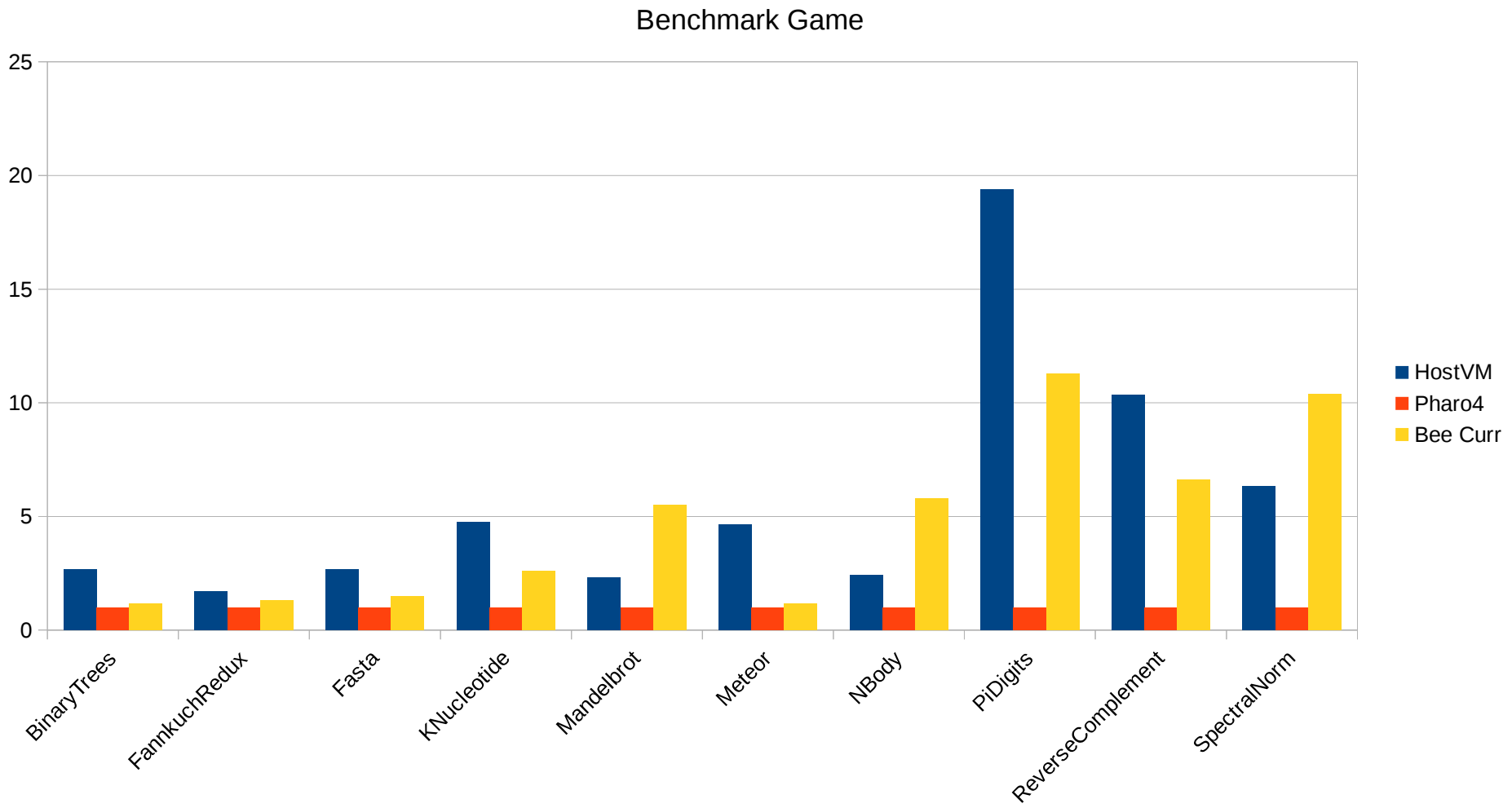
Performance, today



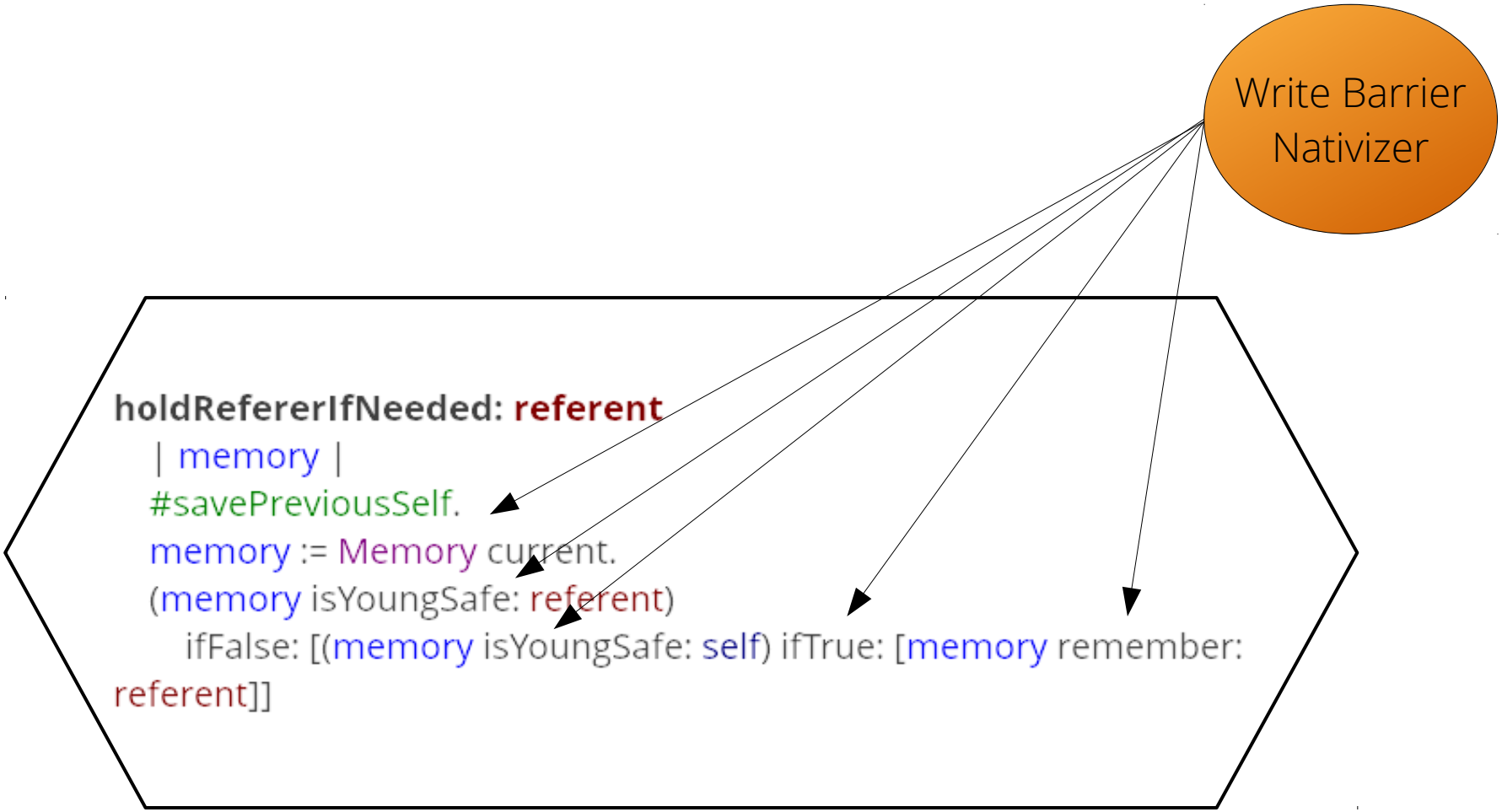
Performance, today



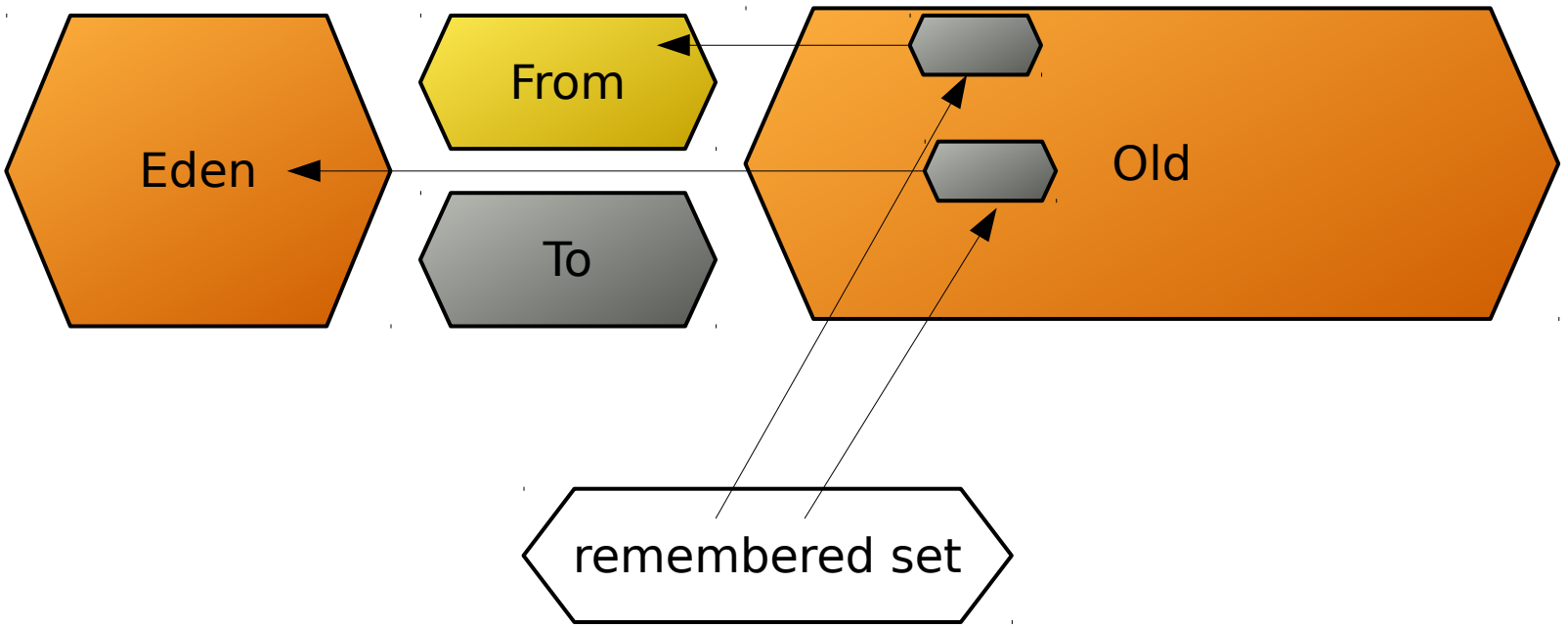
Performance, today



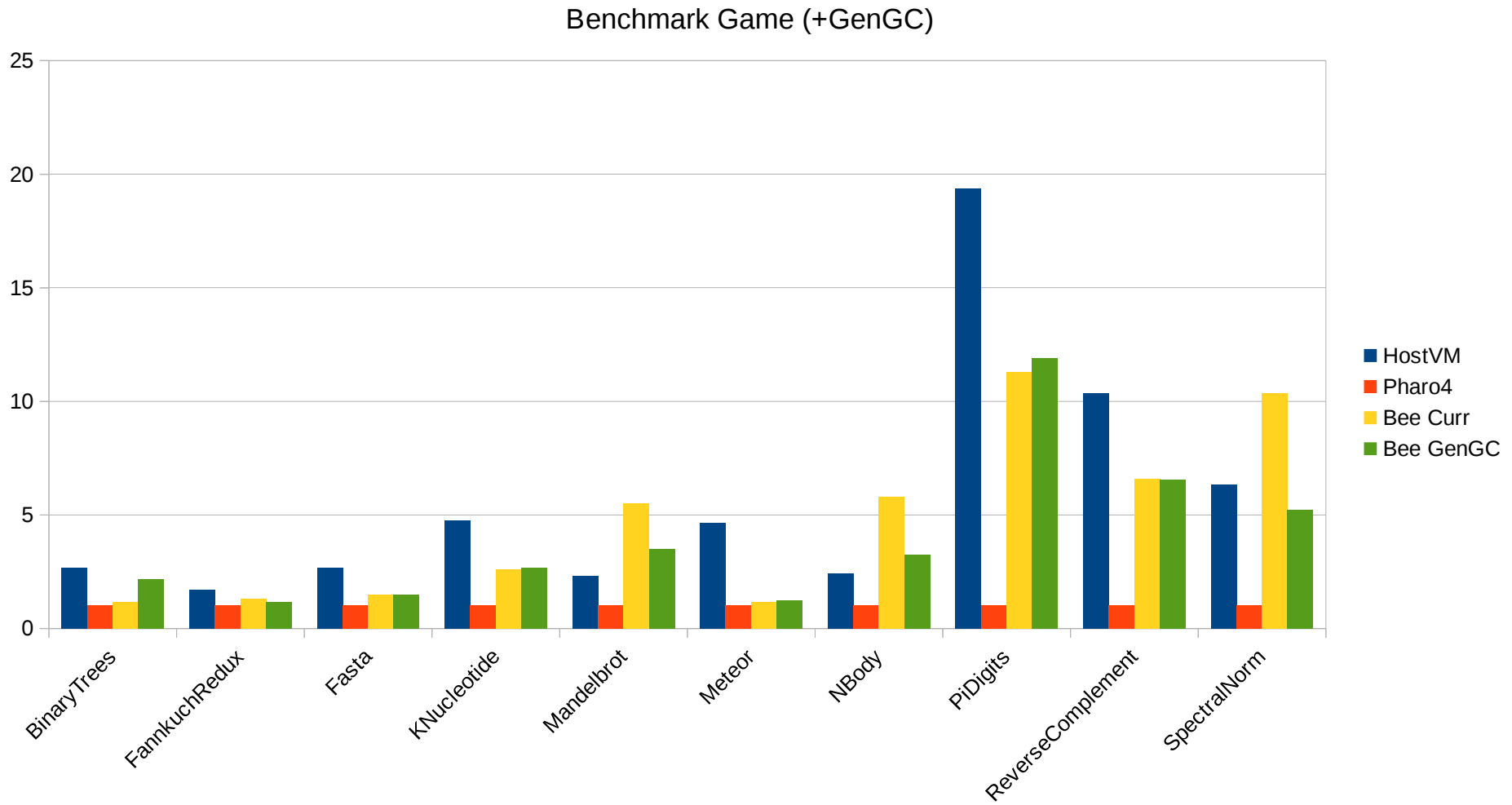
Experiment: generational GC



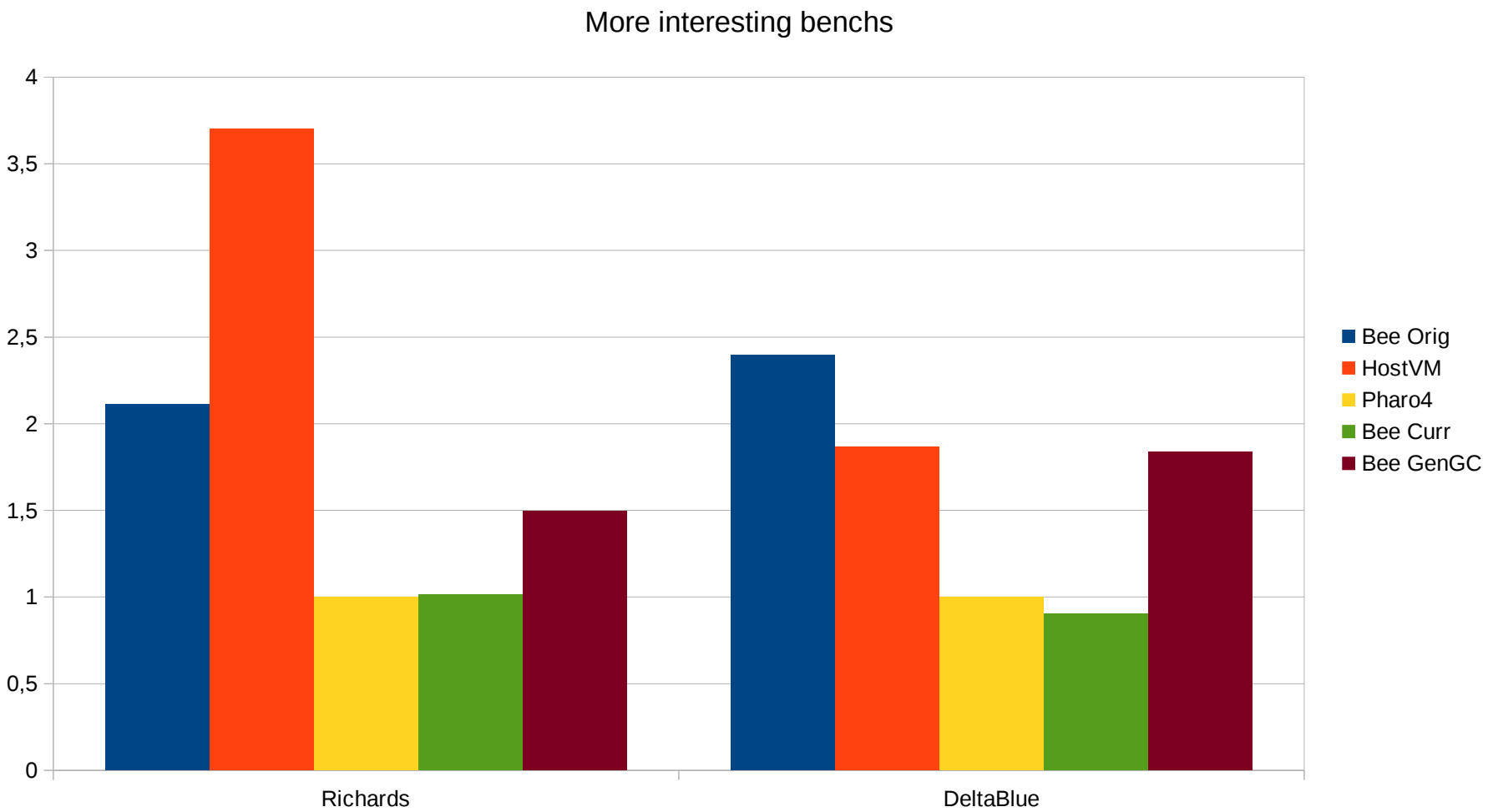
Experiment: generational GC



Performance, with GenGC



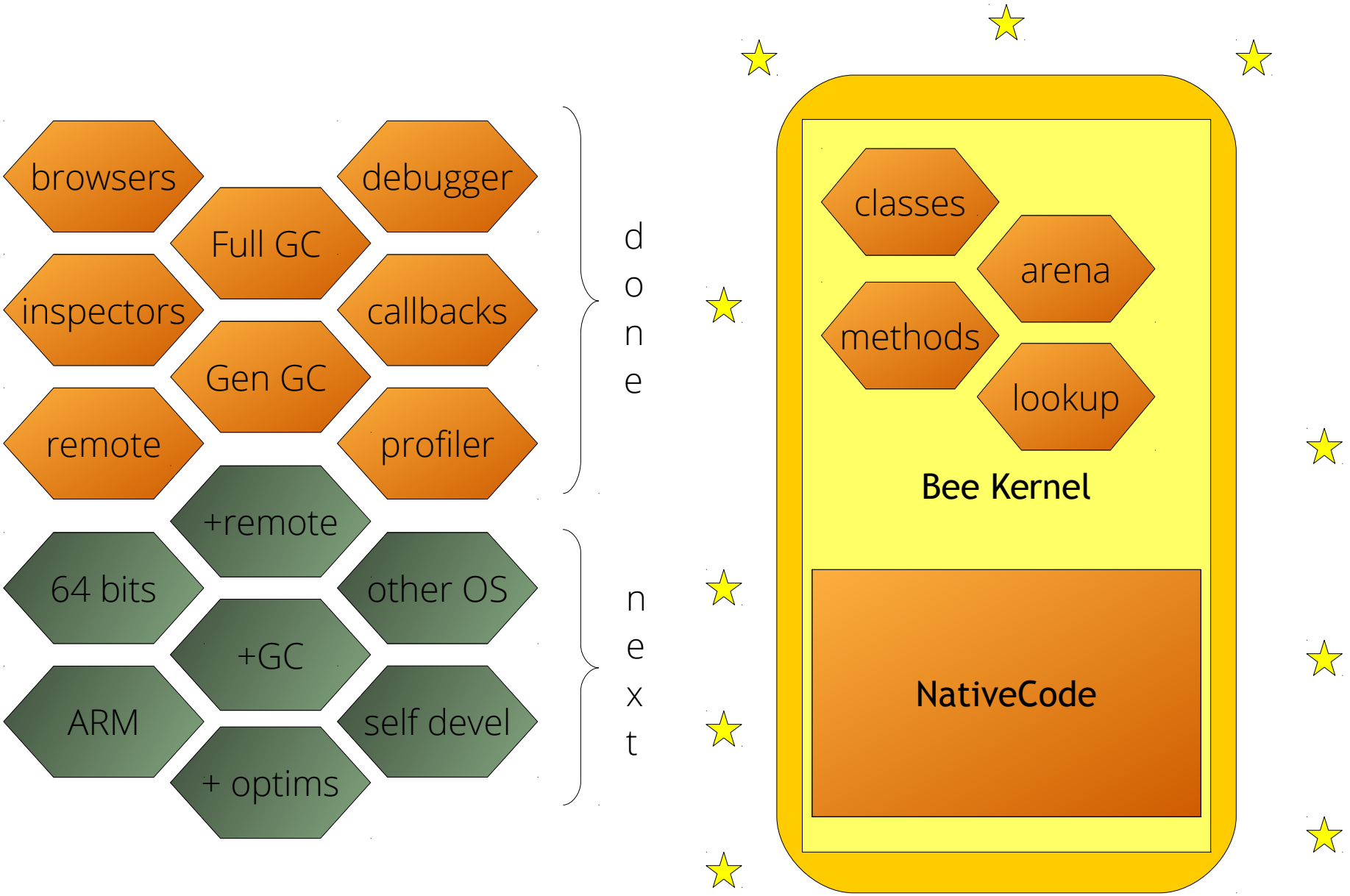
Performance, with GenGC



Conclusions

1. A complete runtime
2. Can compete with classic VMs
3. Experiments are easier to implement
4. Less/different performance constraints in experiments

Past, present, future



```
[Audience hasQuestions] whileTrue: [
    self answer: Audience nextQuestion].

Audience do: [:you | self thank: you].

self returnTo: Audience
```

beesmalltalk.blogspot.com