Bee Smalltalk RunTime: anchor's aweigh

Javier Pimás

DISARMISTA S.R.L. CæsarSystems

Bits of Bee history

Gerardo Richarte

Javier Burroni

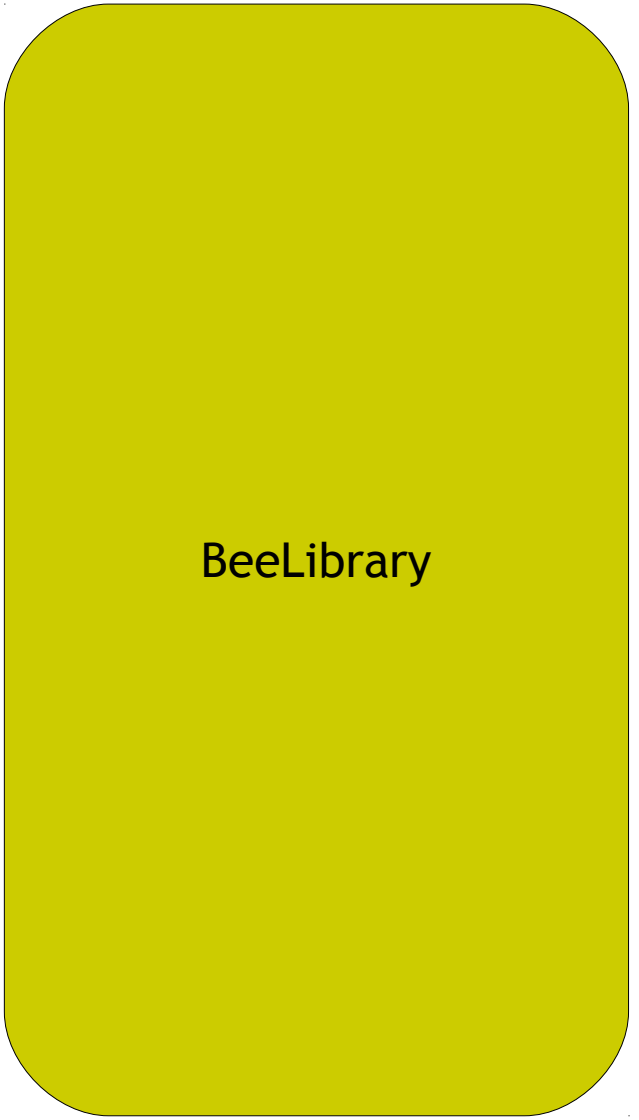
Bits of Bee history

1. a JIT in smalltalk
2. added underprimitives and a GC
3. can we get rid of the VM?

How to start?

Start afresh

With Objects :)



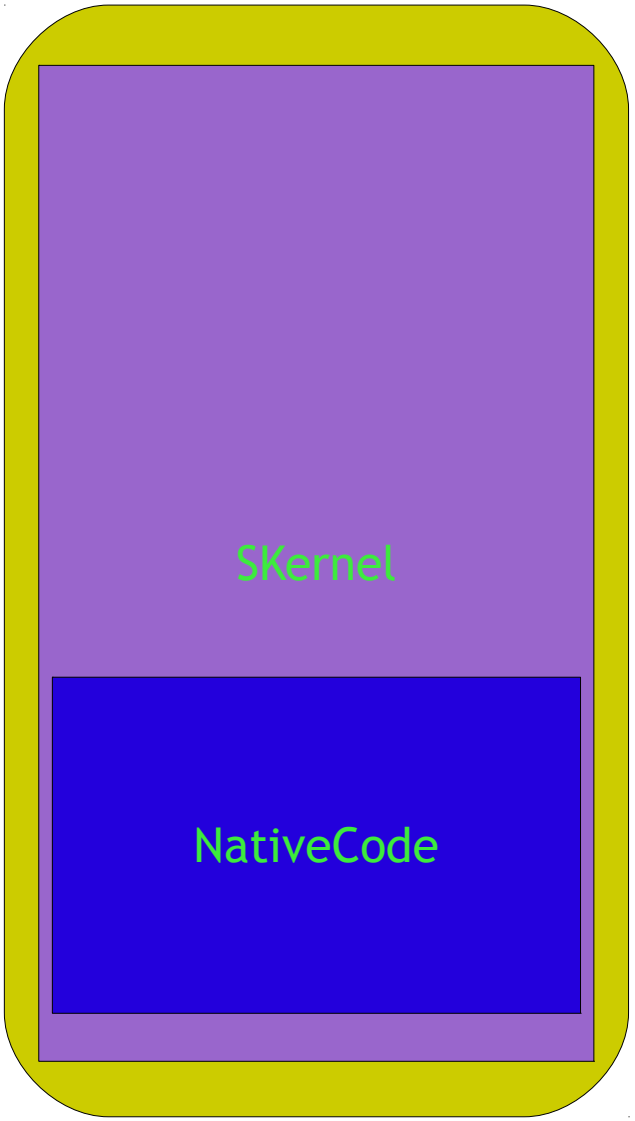
Start afresh

With Objects :)

Start afresh

With Objects :)

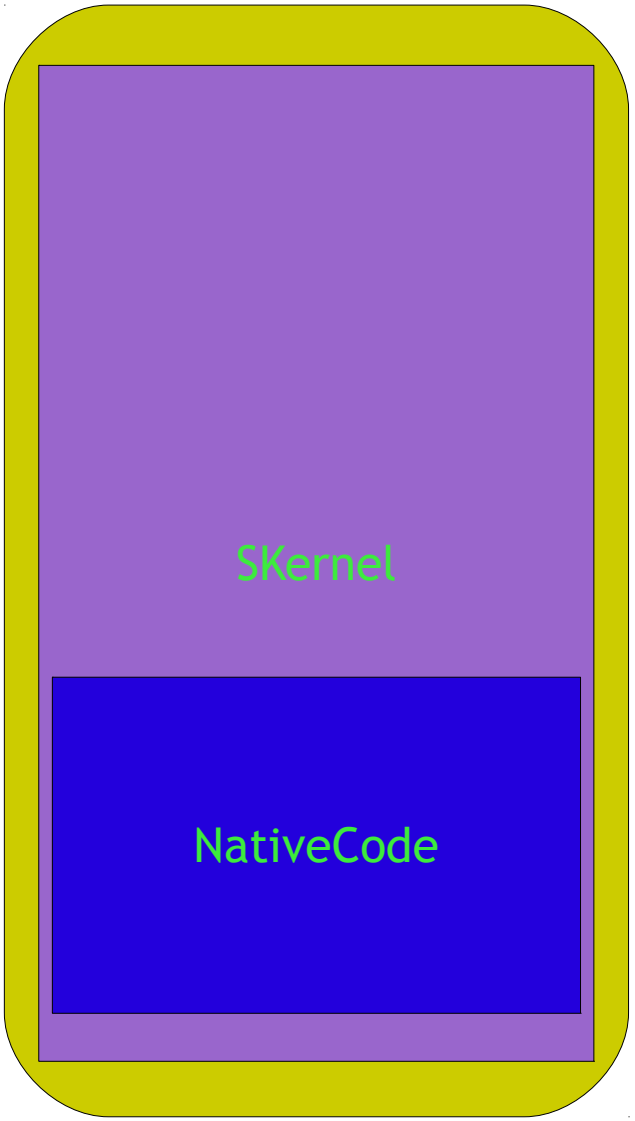
```
Array variableSubclass: #CompiledMethod
  instanceVariableNames:
    ' bytecodes nativeCode class
     selector source flags '
```



Start afresh

With Objects :)

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



Start afresh

How to start?

With Objects :)

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

```
<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
```

Start afresh

How to start?

With Objects :)

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

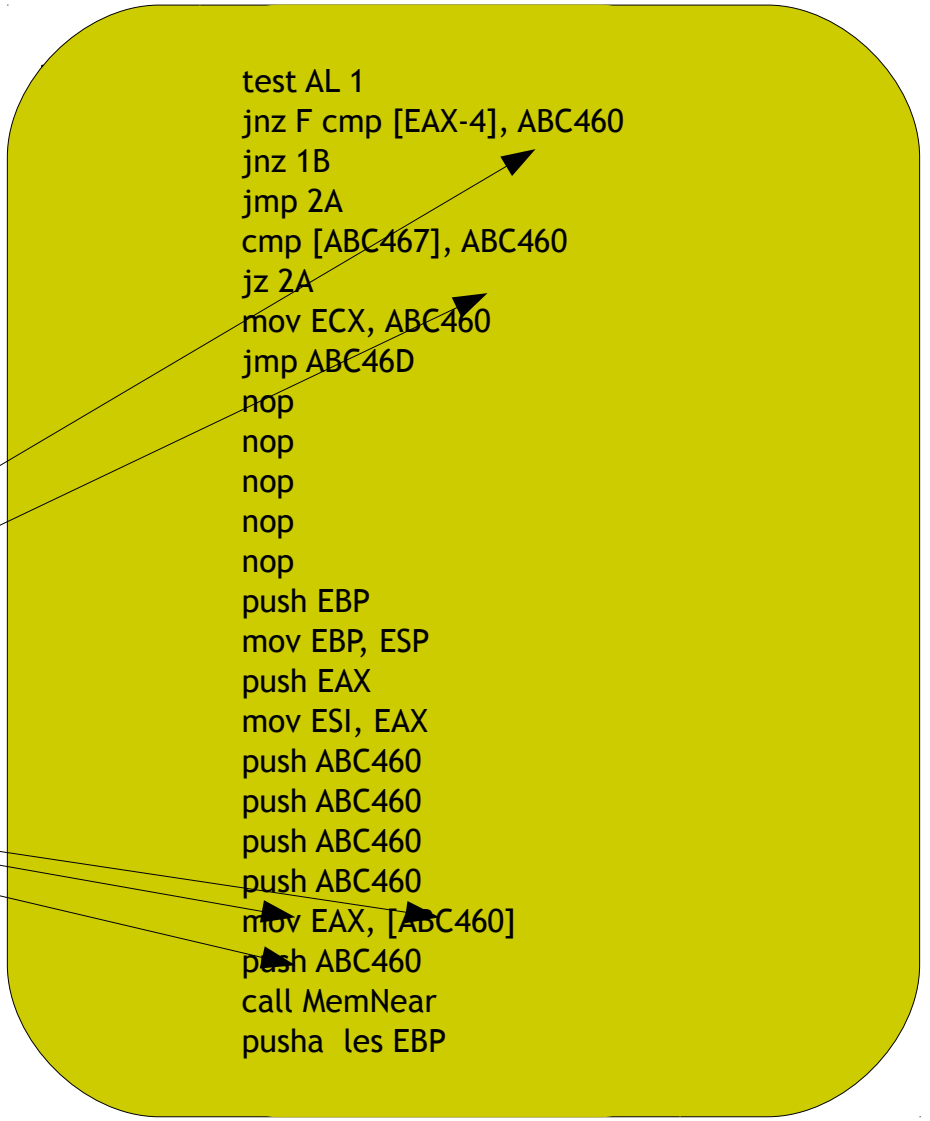
```
test AL 1
jnz F cmp [EAX-4], ABC460
jnz 1B
jmp 2A
cmp [ABC467], ABC460
jz 2A
mov ECX, ABC460
jmp ABC46D
nop
nop
nop
nop
nop
push EBP
mov EBP, ESP
push EAX
mov ESI, EAX
push ABC460
push ABC460
push ABC460
push ABC460
mov EAX, [ABC460]
push ABC460
call MemNear
pusha  les EBP
```

Start afresh

How to start?

With Objects :)

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



So we got rid of the VM, right?

NO!

Working hypothesis: HostVM leverage

- Basic objects
- Method lookup
- Primitives
- Native Helpers

HostVM leverage – Basic Objects

for instance:

- true
- false
- nil
- characters
- Array

```
LoadTrueBytecode>>#assemble
  self loadTrue
```

HostVM leverage – Method lookup

Direct access to VM functions

```
lookupAndCall
    | selector |
    selector := self compiledMethod selector.
    assembler loadSelector: selector oop.
    self holdReferenceTo: selector.
    assembler jumpTo: #VM_lookupAndCall from: #hostVM.dll'
```

HostVM leverage - Primitives

We used HostVM' primitives as a way to have bit-a-bit compatibility

```
SmalltalkBytecode>>#prepareFrame
   primitive := self nextByte.
   cm := self compiledMethod.
   assembler loadTempPointer: cm oop.
   self holdReferenceTo: cm; callPrimitive: primitive.
   super prepareFrame
```

HostVM leverage - Native Helpers

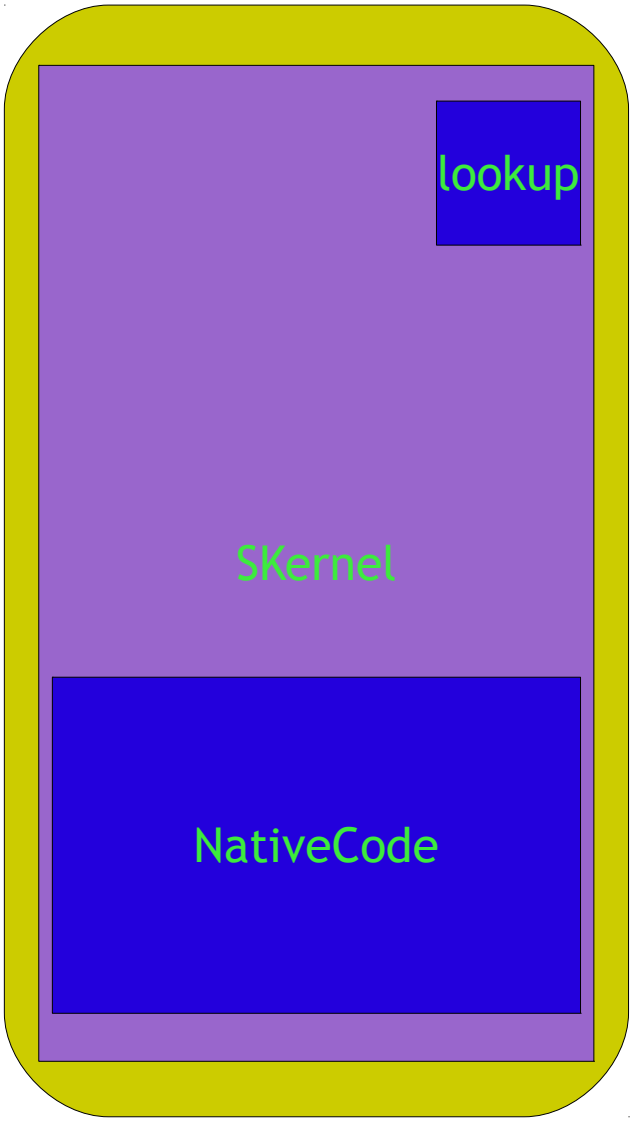
Small pieces of code in the HostVM acting as primitives but without their protocol.

as a bit-a-bit compatible JIT, we were using those helpers

```
callNewArray
  self emitCallTo: #VM_newArray from: #'hostVM.dll'
```

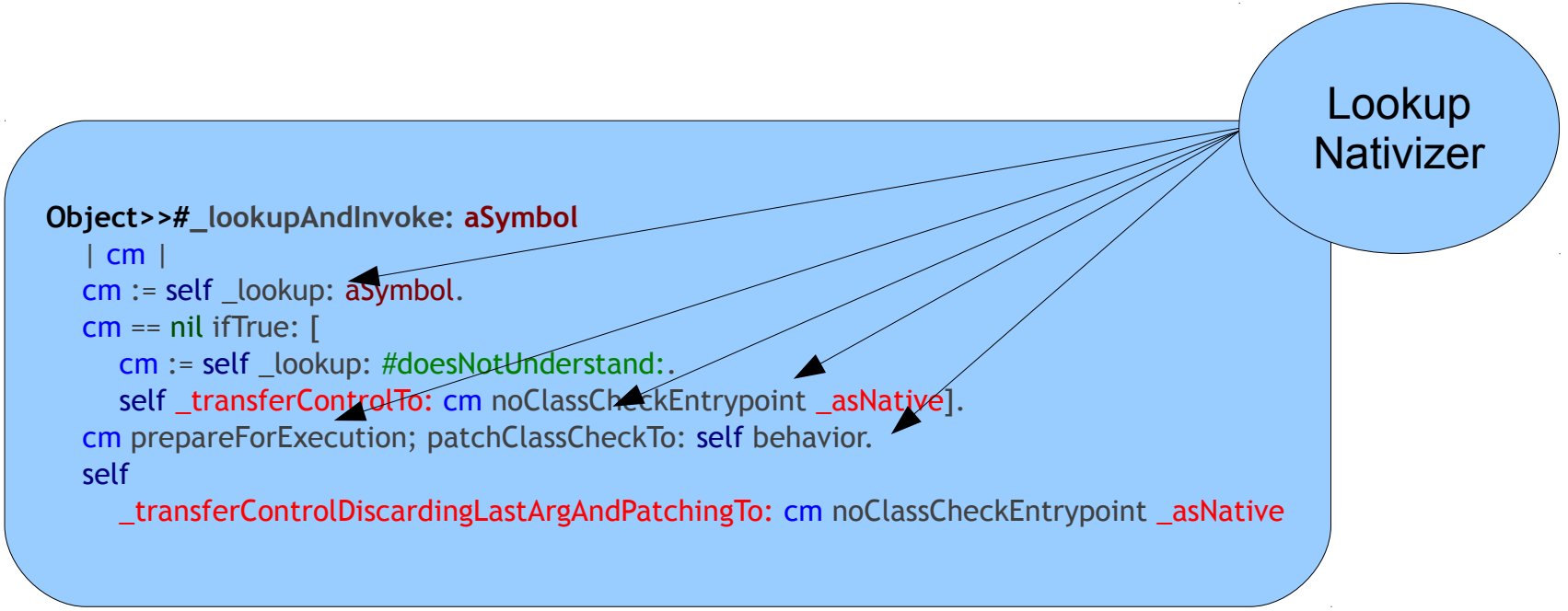
Connecting methods: lookup

Object>>#_lookupAndInvoke: aSymbol



Connecting methods: lookup

```
mov     eax, off_427090
push    offset aGetcommandline ; "getCommandLine"
call    Object__lookupAndInvokeNativeCode
```



Smalltalk semantics: 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 convertArgToIntegerOrJump
    assembler
        framelessSlotAtArg;
        jumpDestinationFor: nonInteger
```

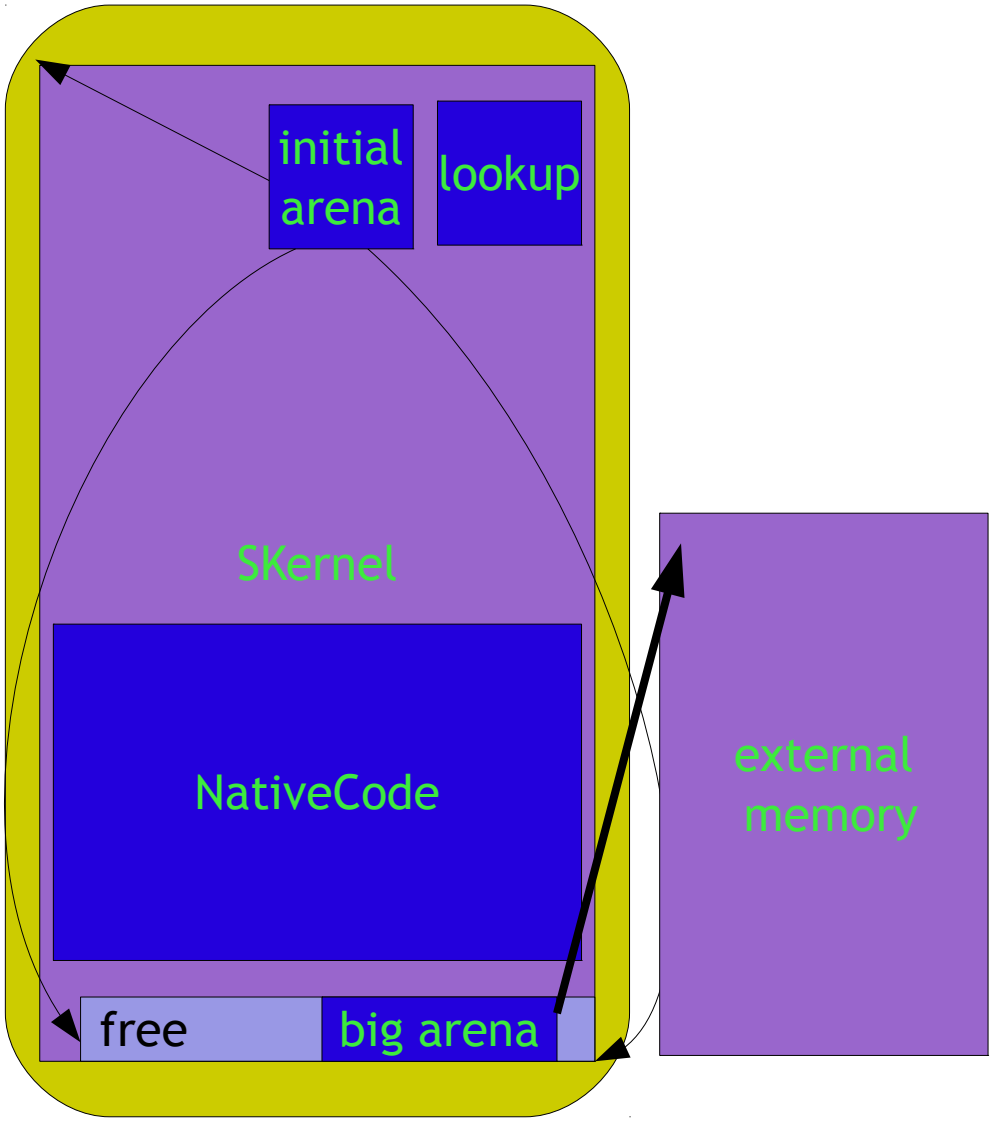
Connecting methods: BlockClosure

```
anySatisfy: aBlock
    self detect: aBlock ifNone: [^false].
    ^true
```

```
BlockClosure>>#primitiveBeeValue
    self argumentCount = 0 ifFalse: [^self primitiveFailed].
    self _transferControlTo: self code
```

Creating objects: new

Object new



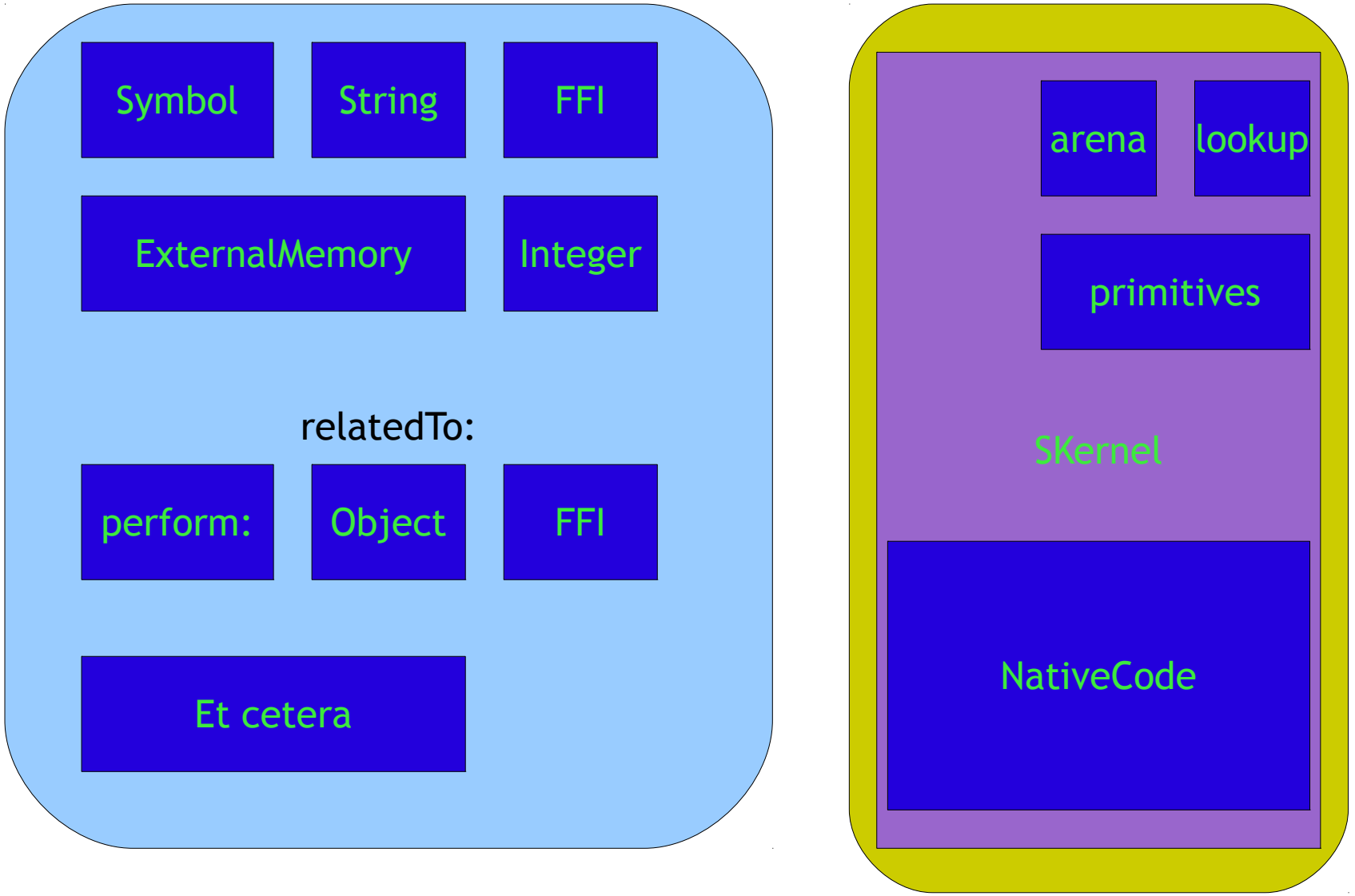
Creating objects: new

```
Object new
```

```
Behavior>>#primitiveNew
  ^self primitiveNewPointers: self instSize
```

```
Behavior>>#primitiveNewPointers: size
  | object |
  object := self allocate: size * 4 size: size.
  self isFixed ifTrue: [object _beFixed; _beNamed].
  1 to: size do: [:i | object _basicAt: i put: nil].
  ^object
```

More primitives

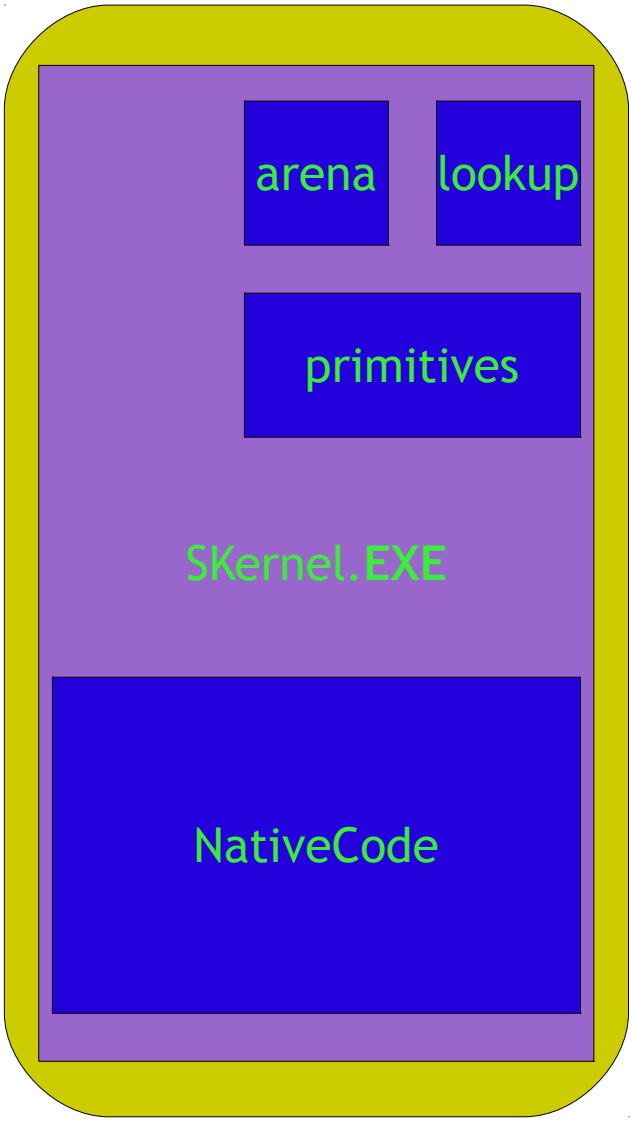


Putting things together

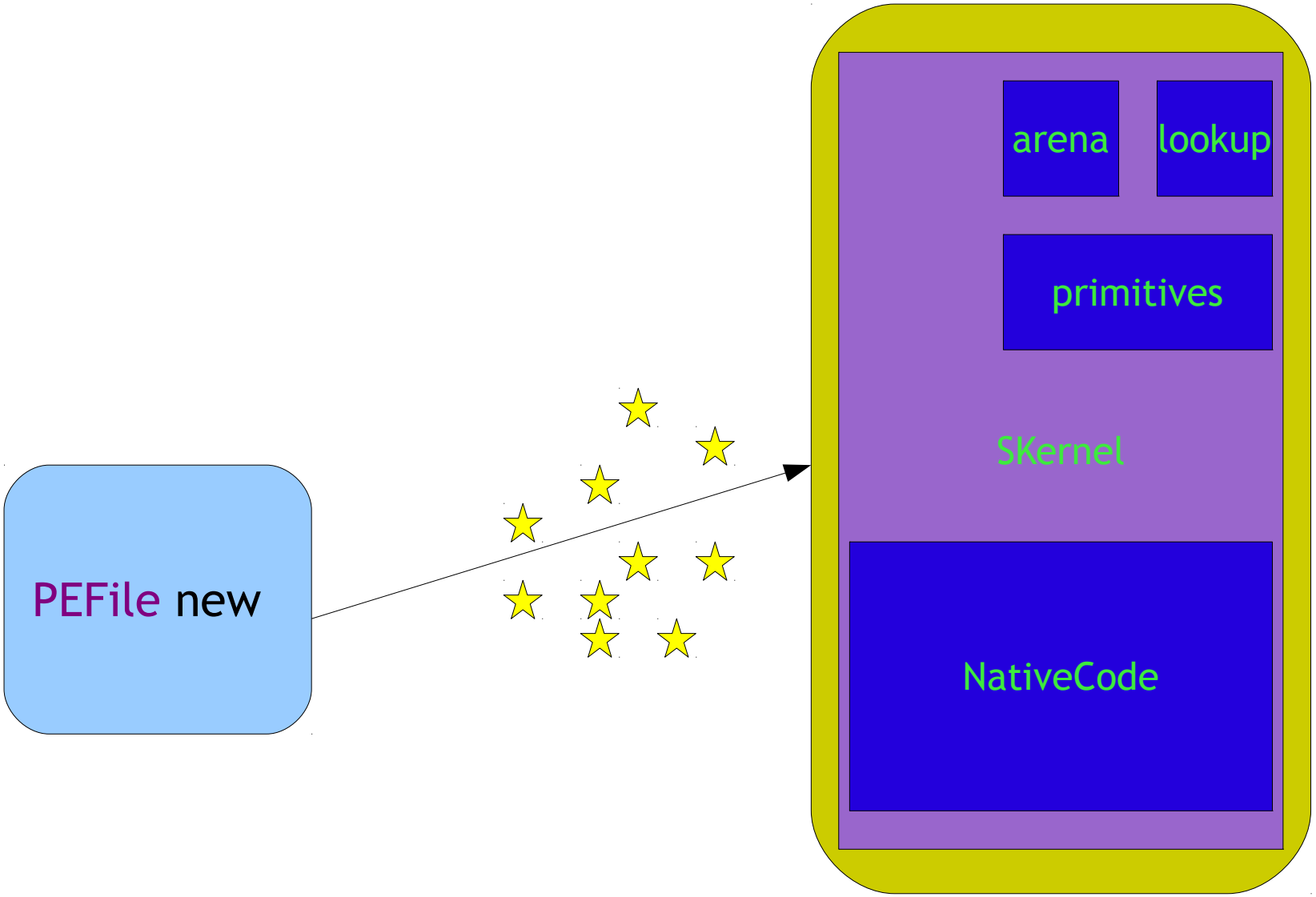
```
generateFullKernel
| closure filename |
    builder := KernelLibraryBuilder newNamed: 'bee'.
    project := SmalltalkProject getProject: 'Skernel'.
    project buildLibraryOn: builder.

    self
        addFutureSmalltalk;
        addTrueFalseAndNil;
        addBeeKernelClasses;
        addMinimalKernelMethods;
        addLookup;
        addPrimitives;
        addArena;
        remapCharactersArray.
    aBlock value.
    ^builder
        storeNativeCode;
        useDllMode;
        writeBSL
```

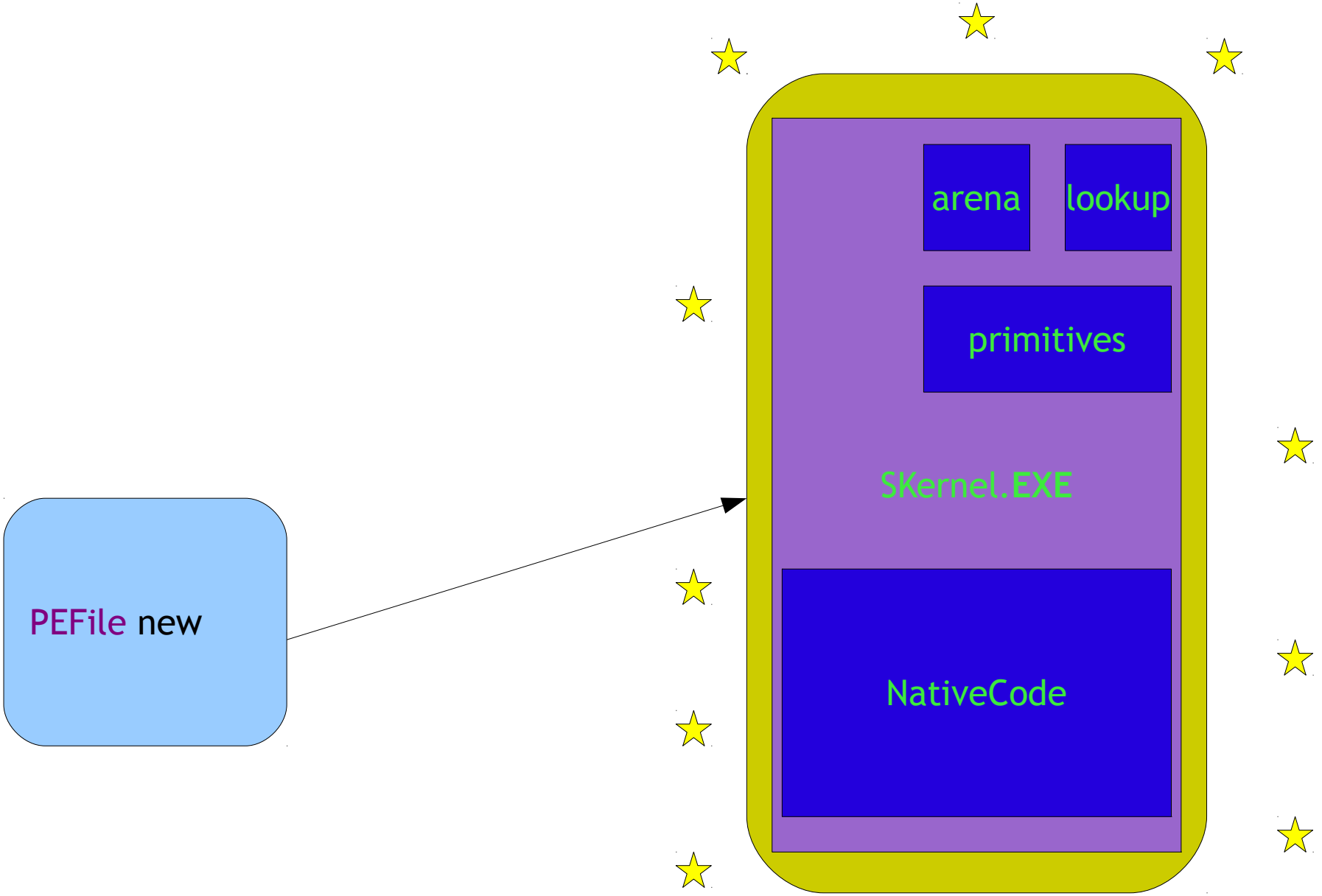
```
start
    | result |
    result := self initializeBee; run.
    self exit: result
```



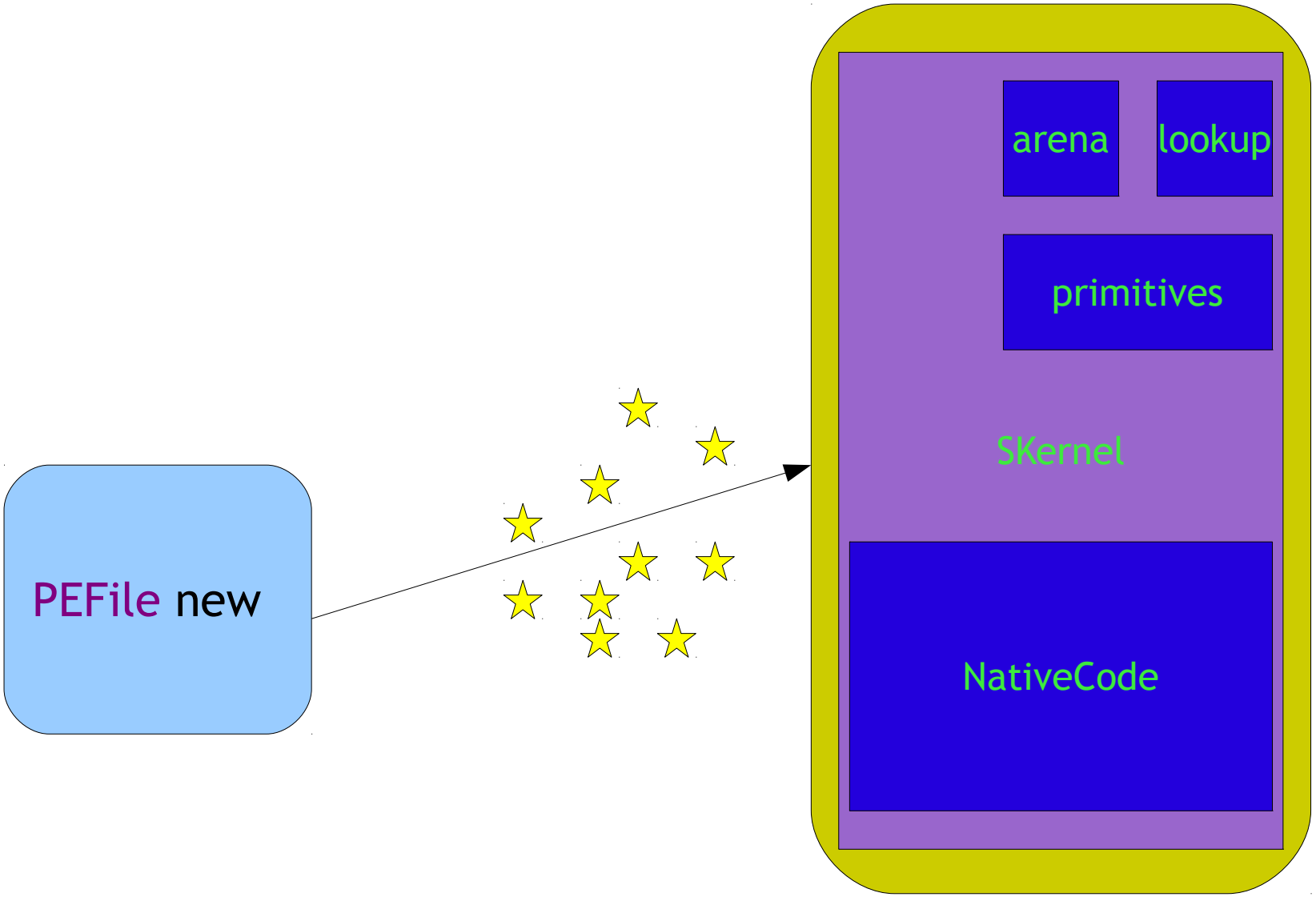
Some magic



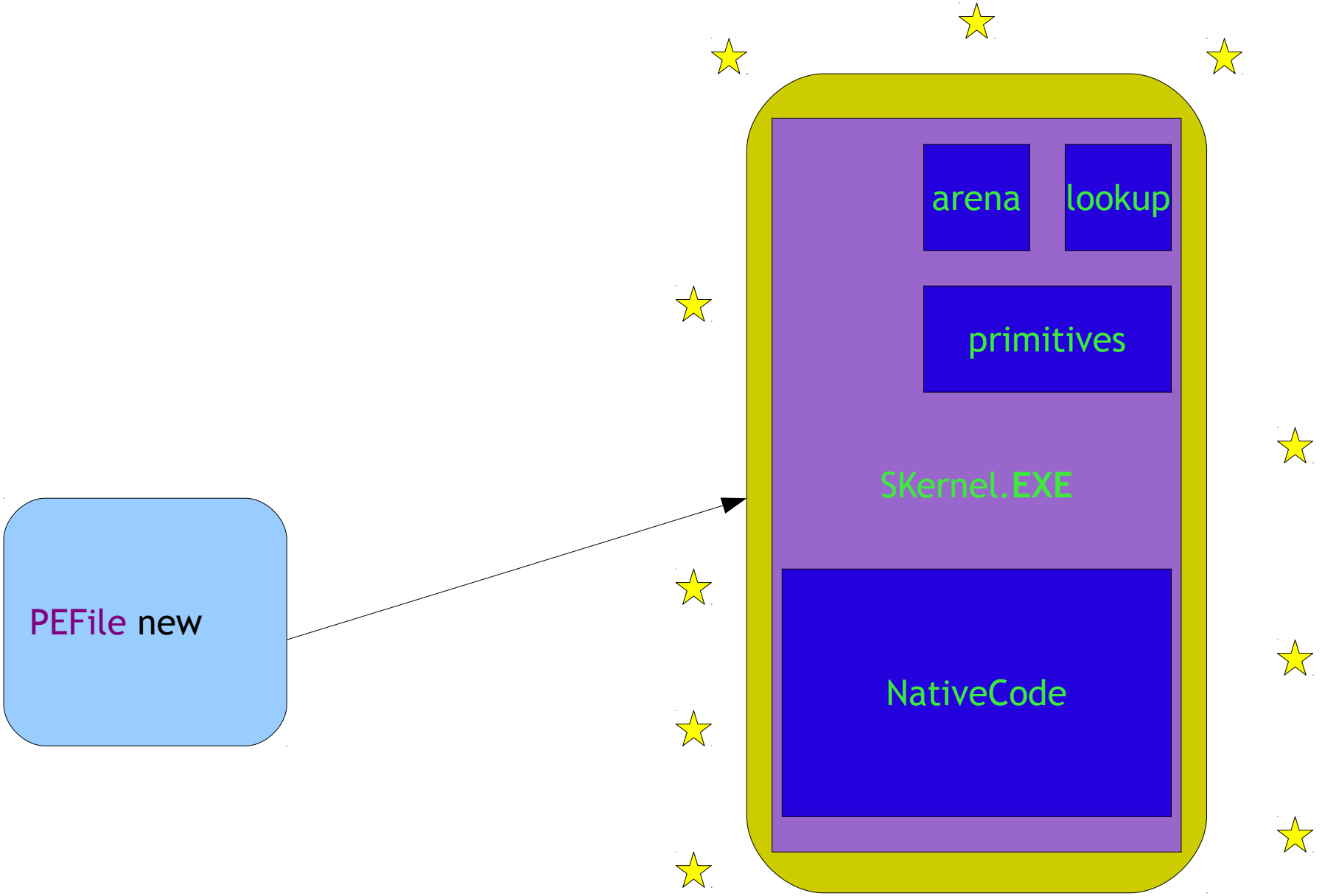
Some magic


arena
lookup
primitives
SKernel.EXE
NativeCode
PEFile new

Some magic



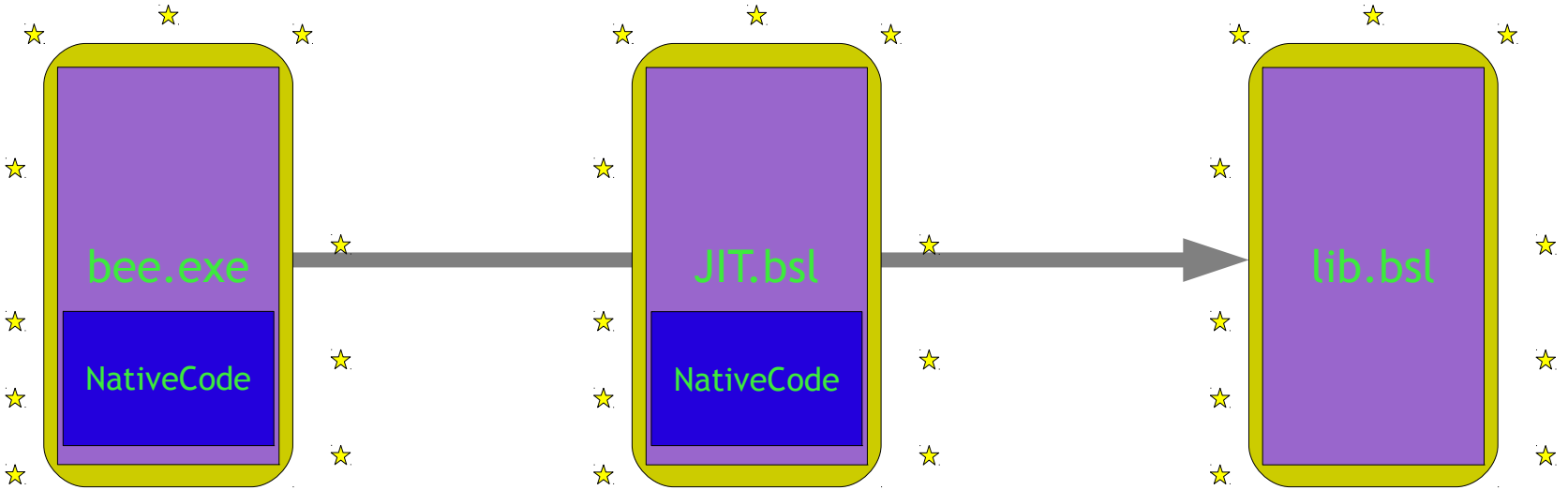
Some magic


arena
lookup
primitives
SKernel.EXE
NativeCode
PEFile new

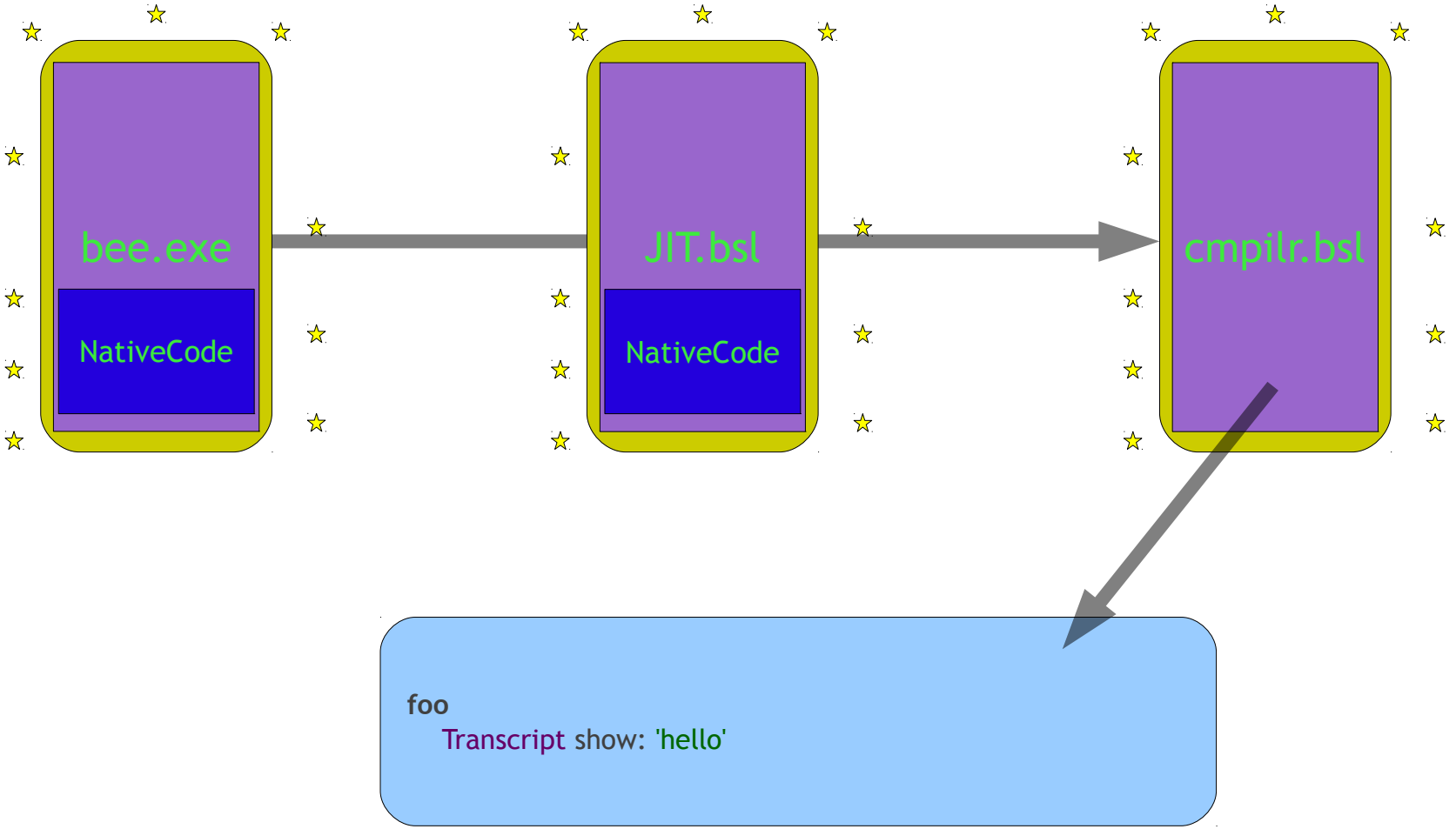
Bee-ing minimal

Many small libraries

Adding nativizer support

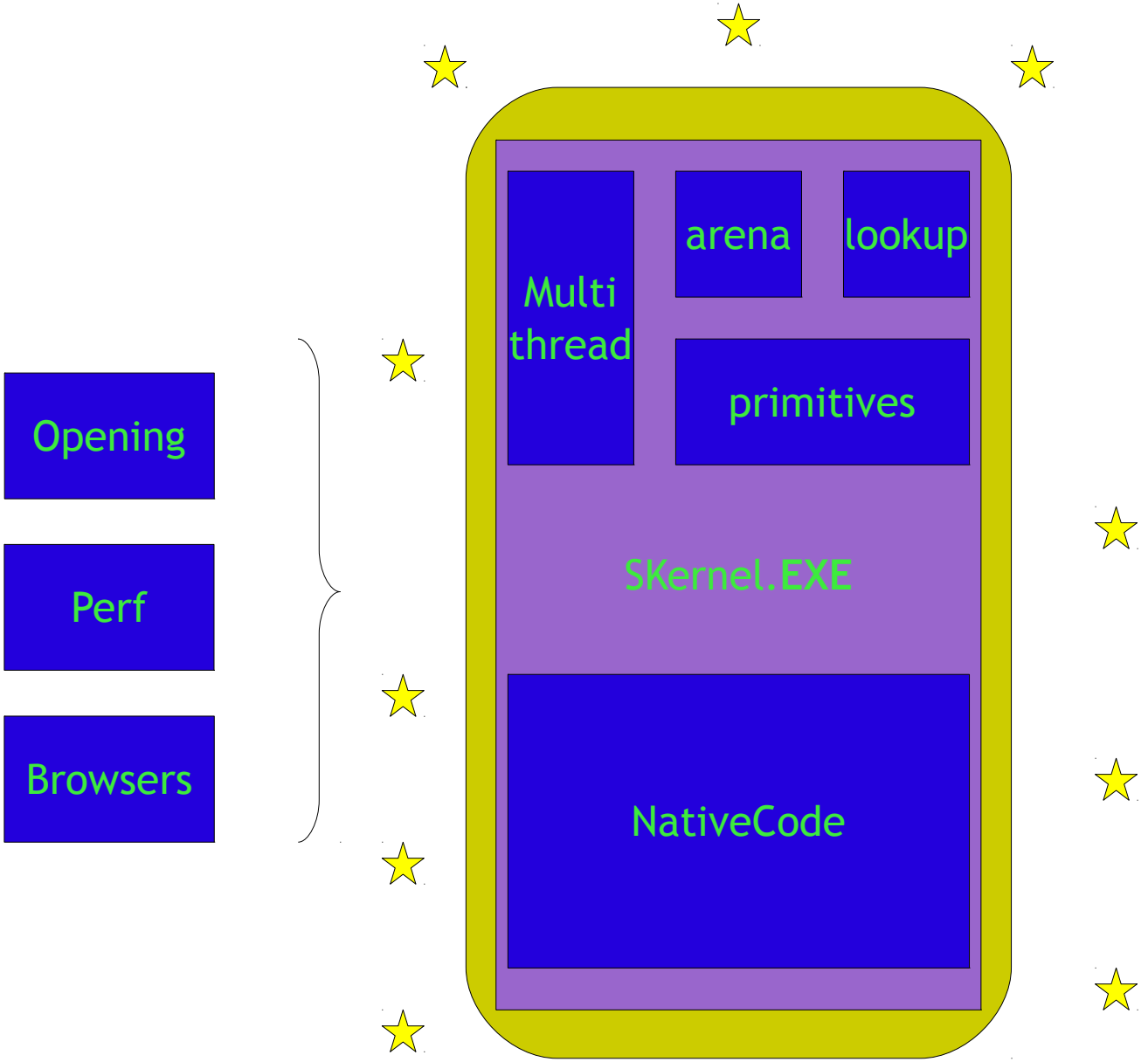


Adding compiler support

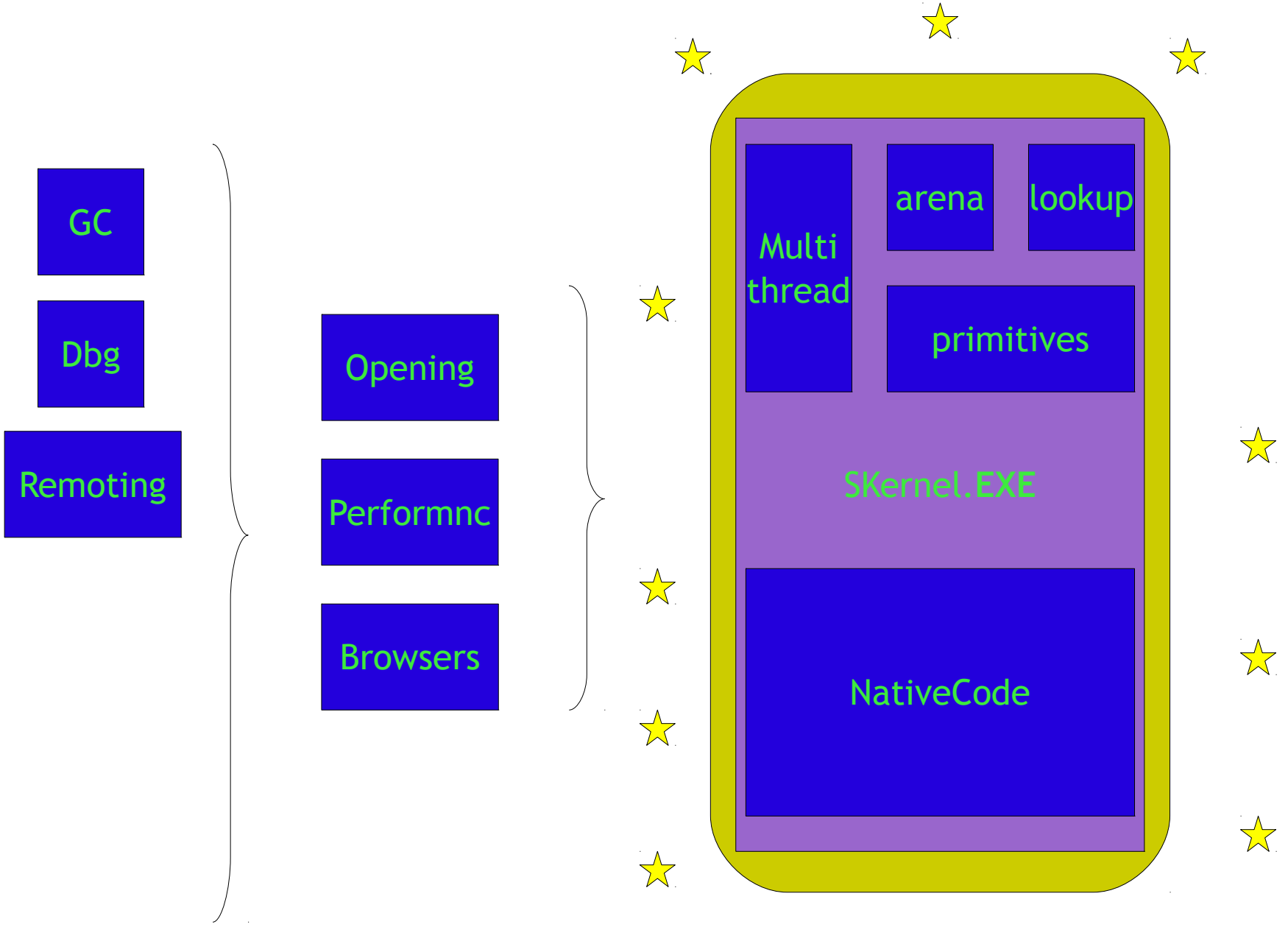


Demo

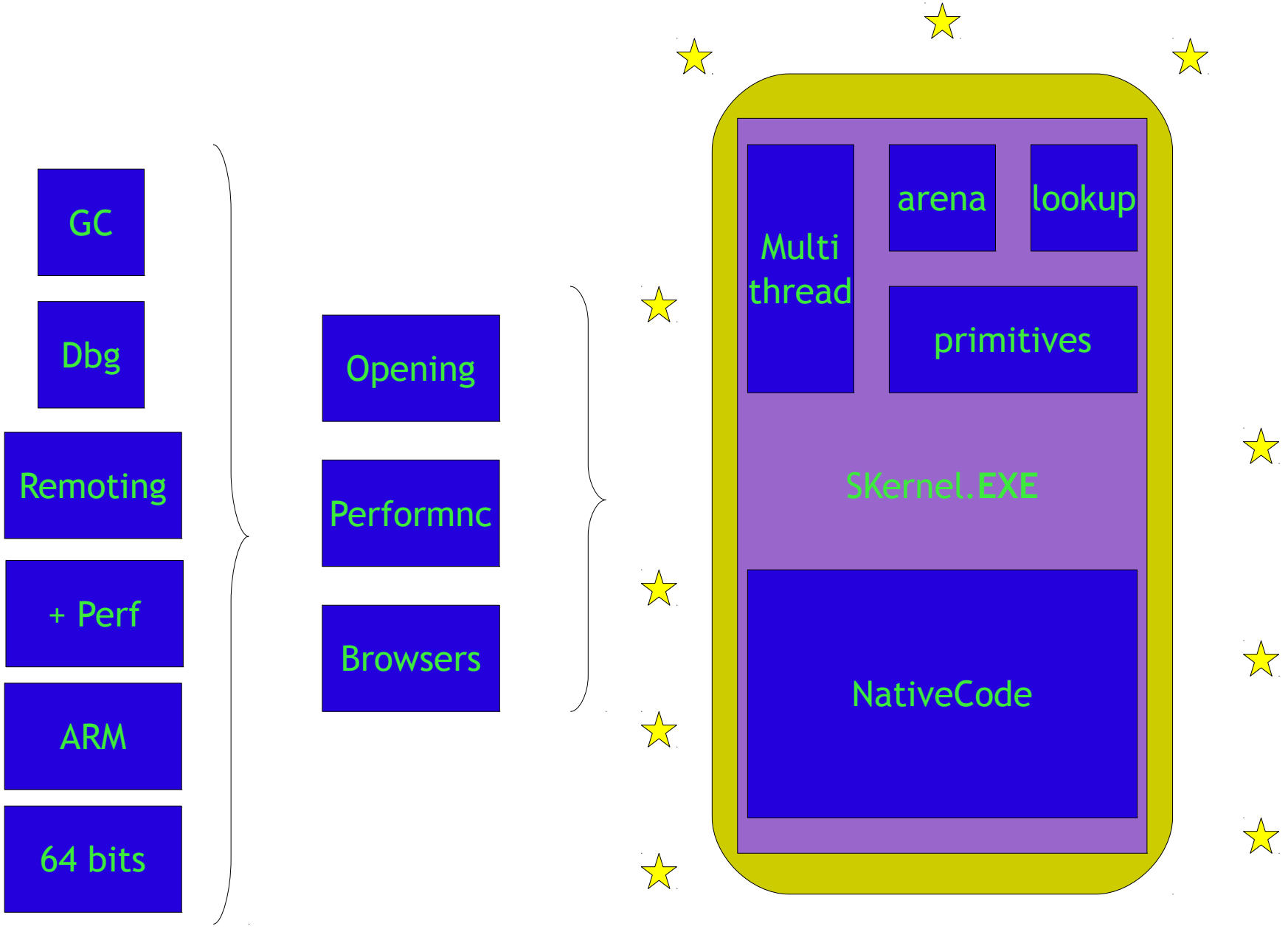
Working on:


Opening
Perf
Browsers
Multi thread
arena
lookup
primitives
SKernel.EXE
NativeCode

Future:


GC
Dbg
Remoting
Opening
Performnc
Browsers
Multi thread
arena
lookup
primitives
SKernel.EXE
NativeCode

(hoping not so) far future:



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

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

self returnTo: Audience
```