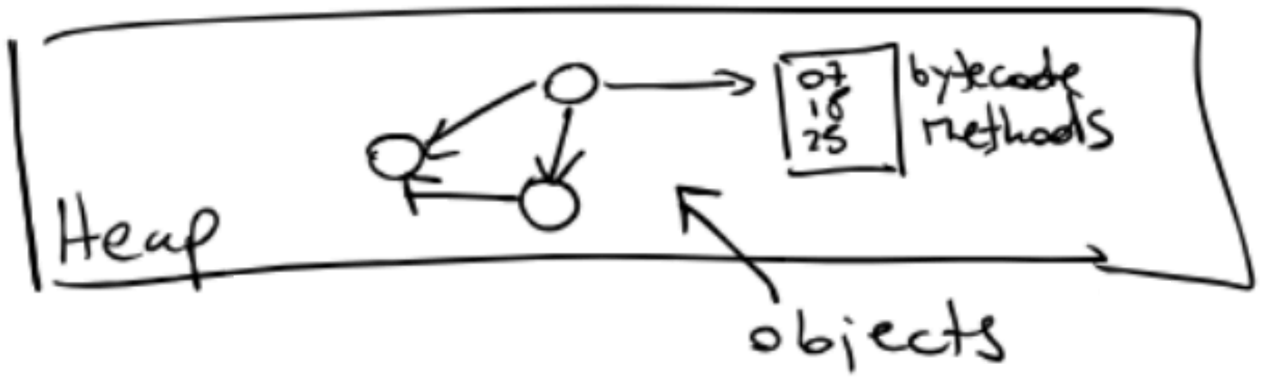
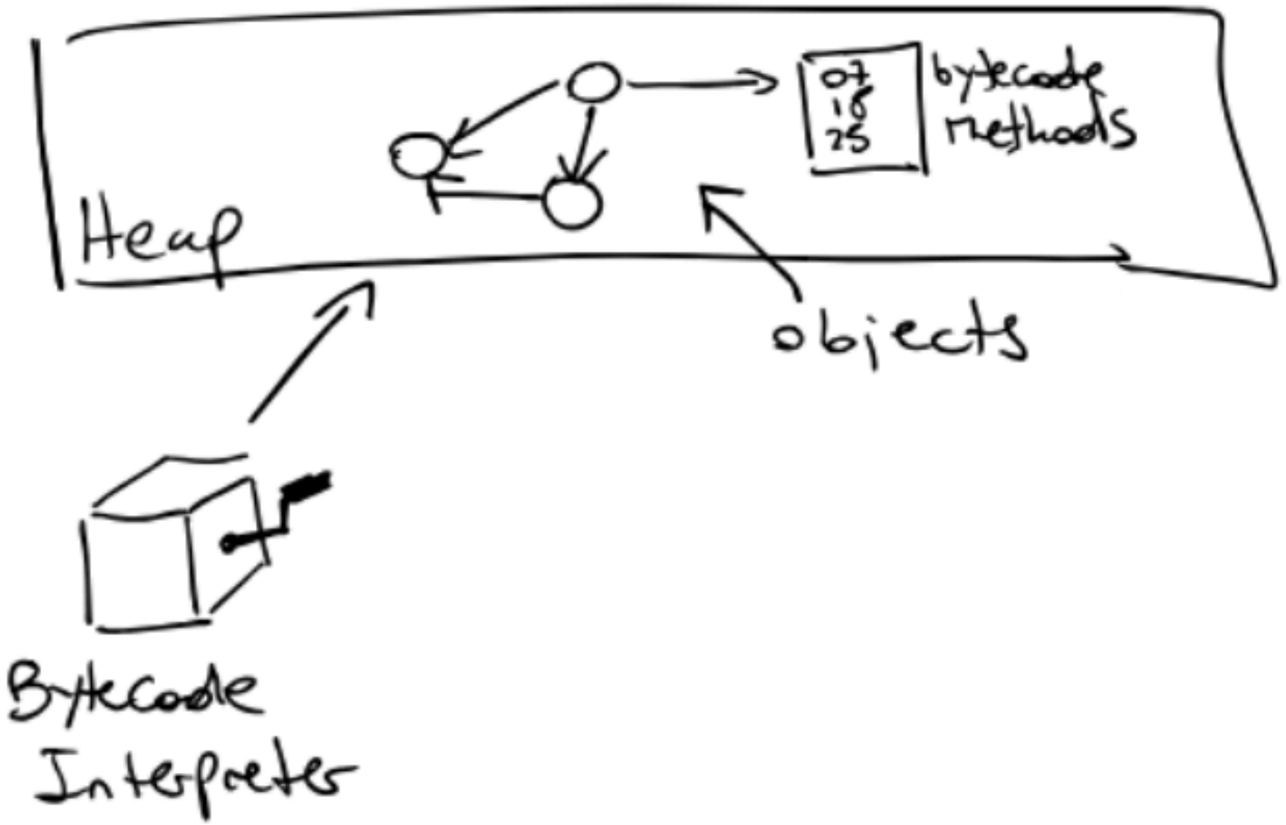
Pharo VM Runtime

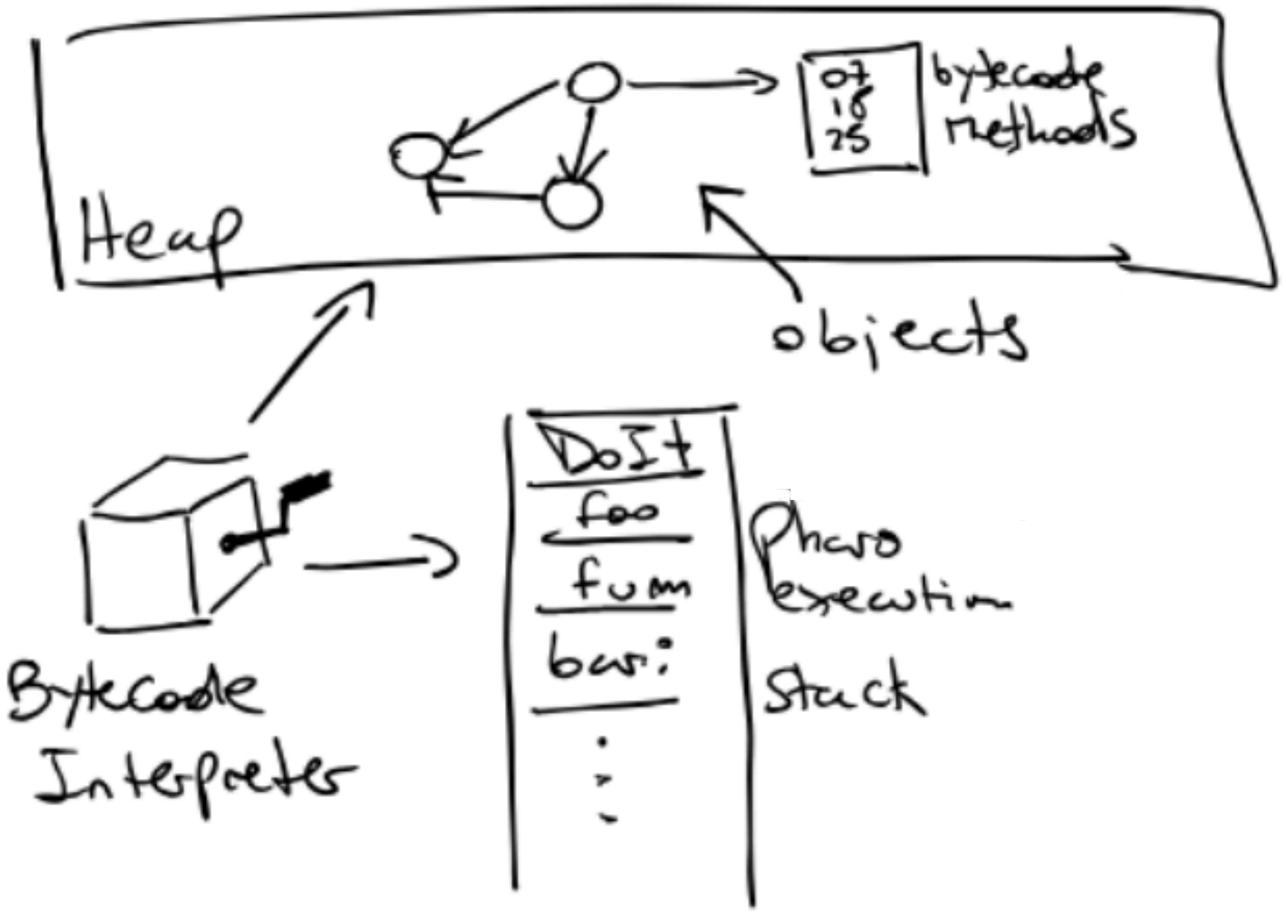
What we know so far…



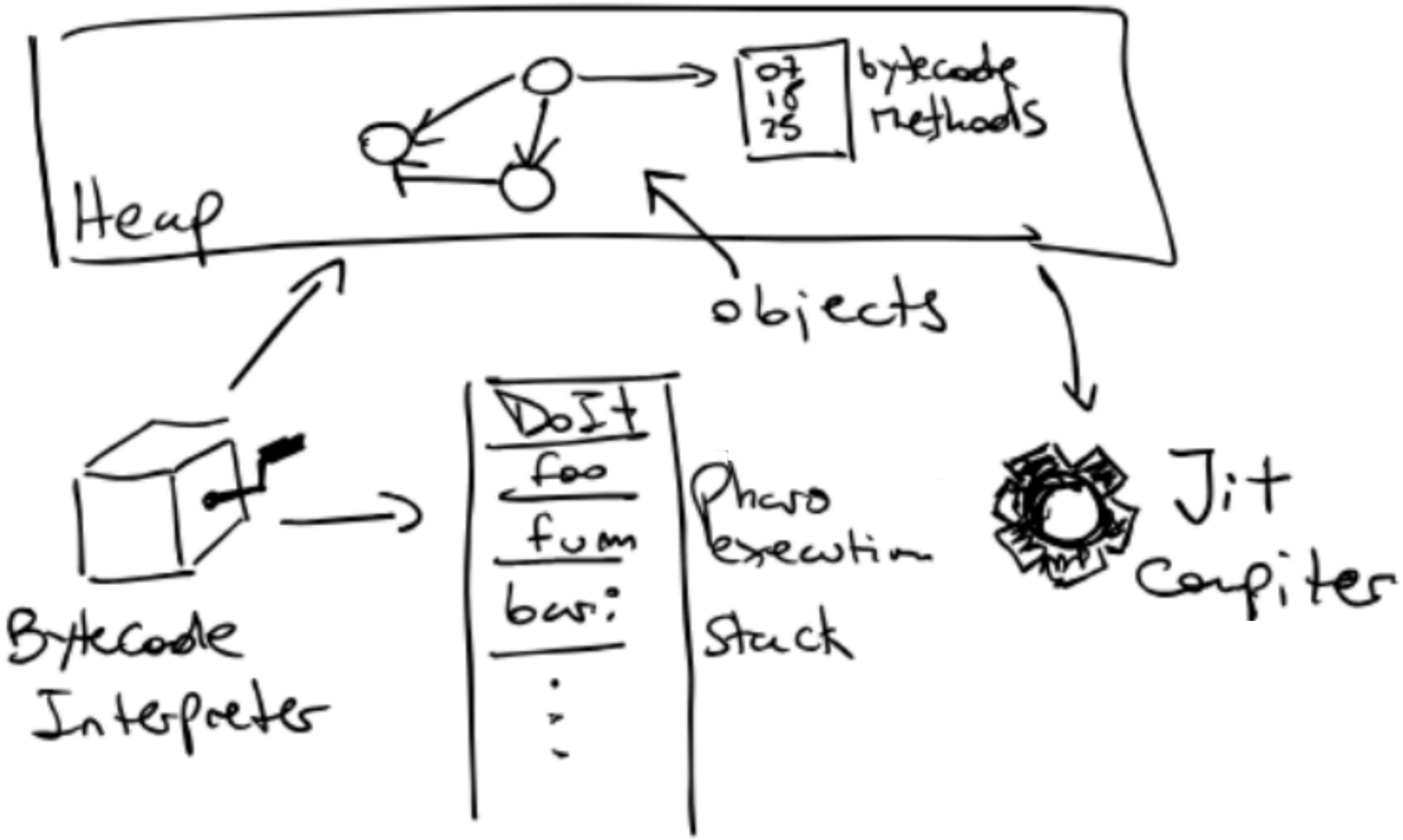
What we know so far…



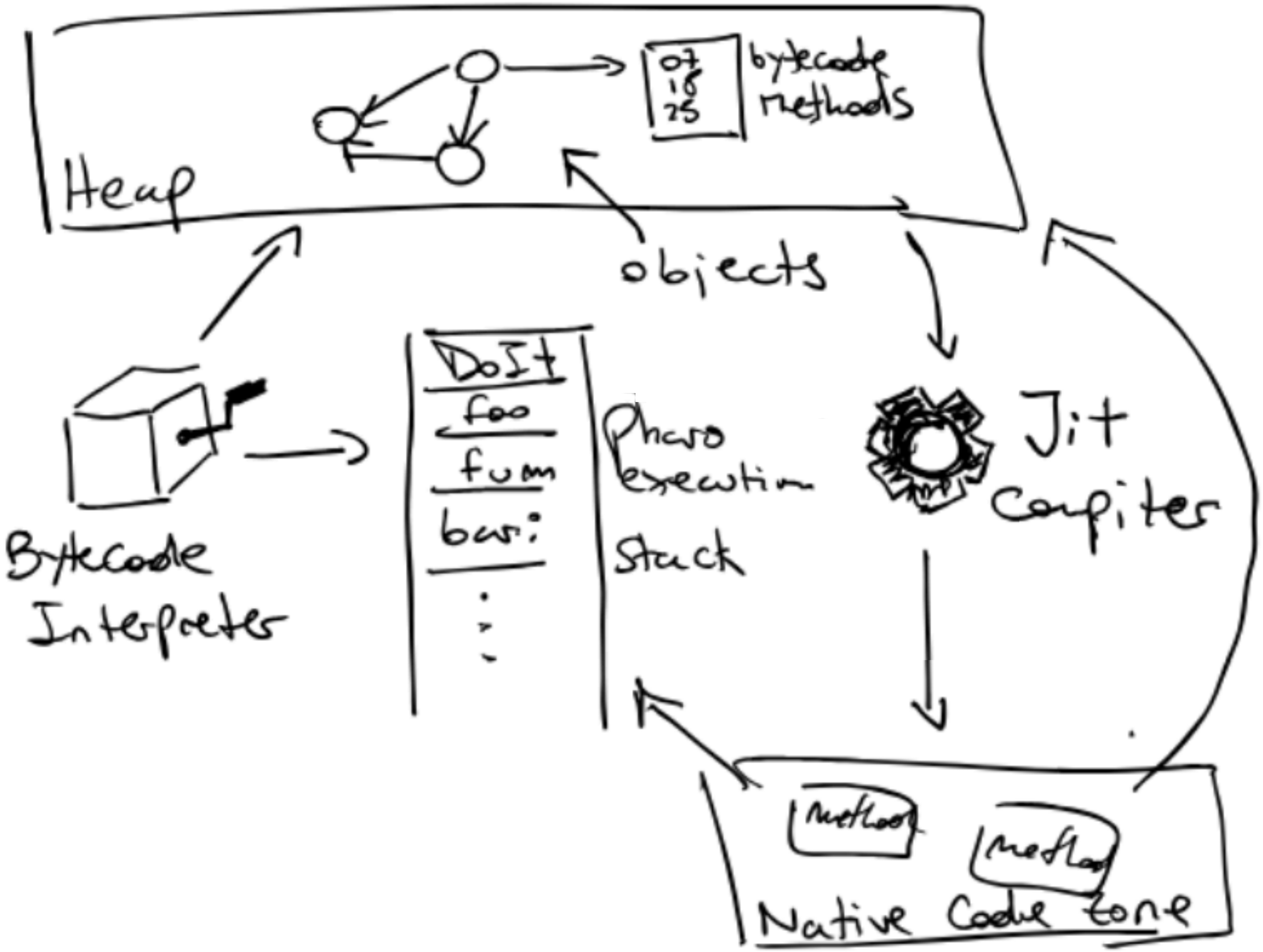
What we know so far…



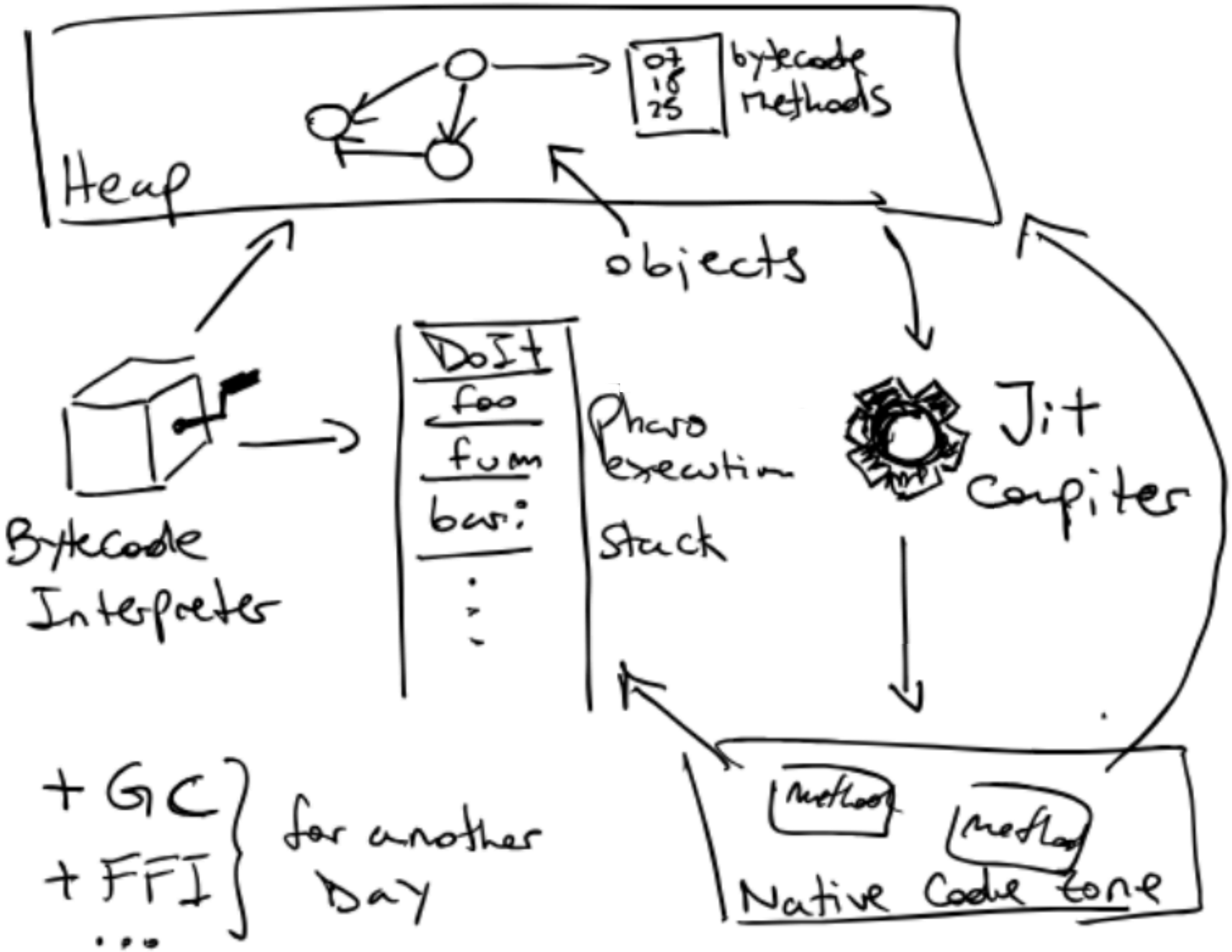
What we know so far…



What we know so far…



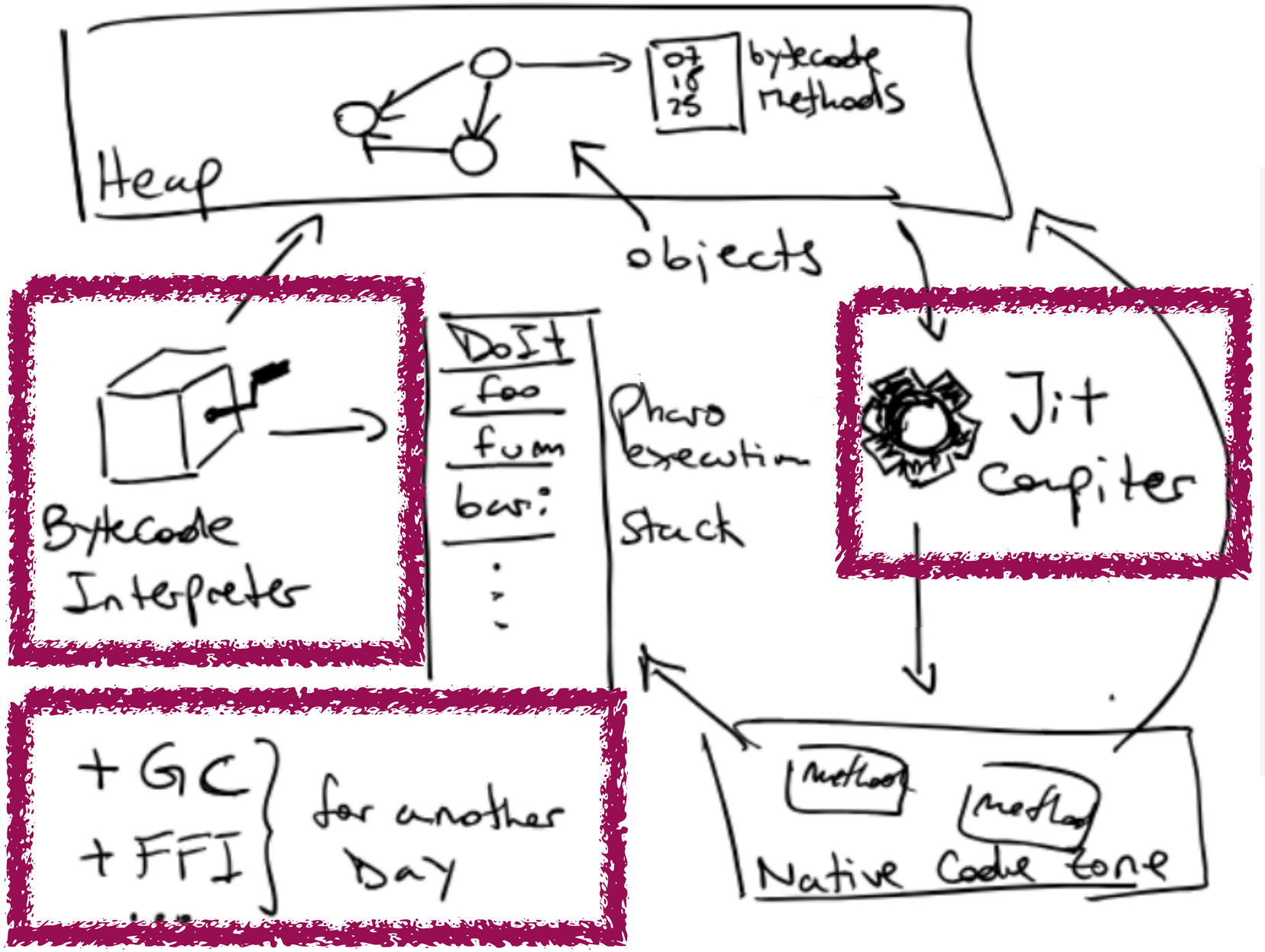
What we know so far…



What we know so far…

- **VM C Runtime**

  Compiled Ahead of Time
  Slang -> C -> Native code



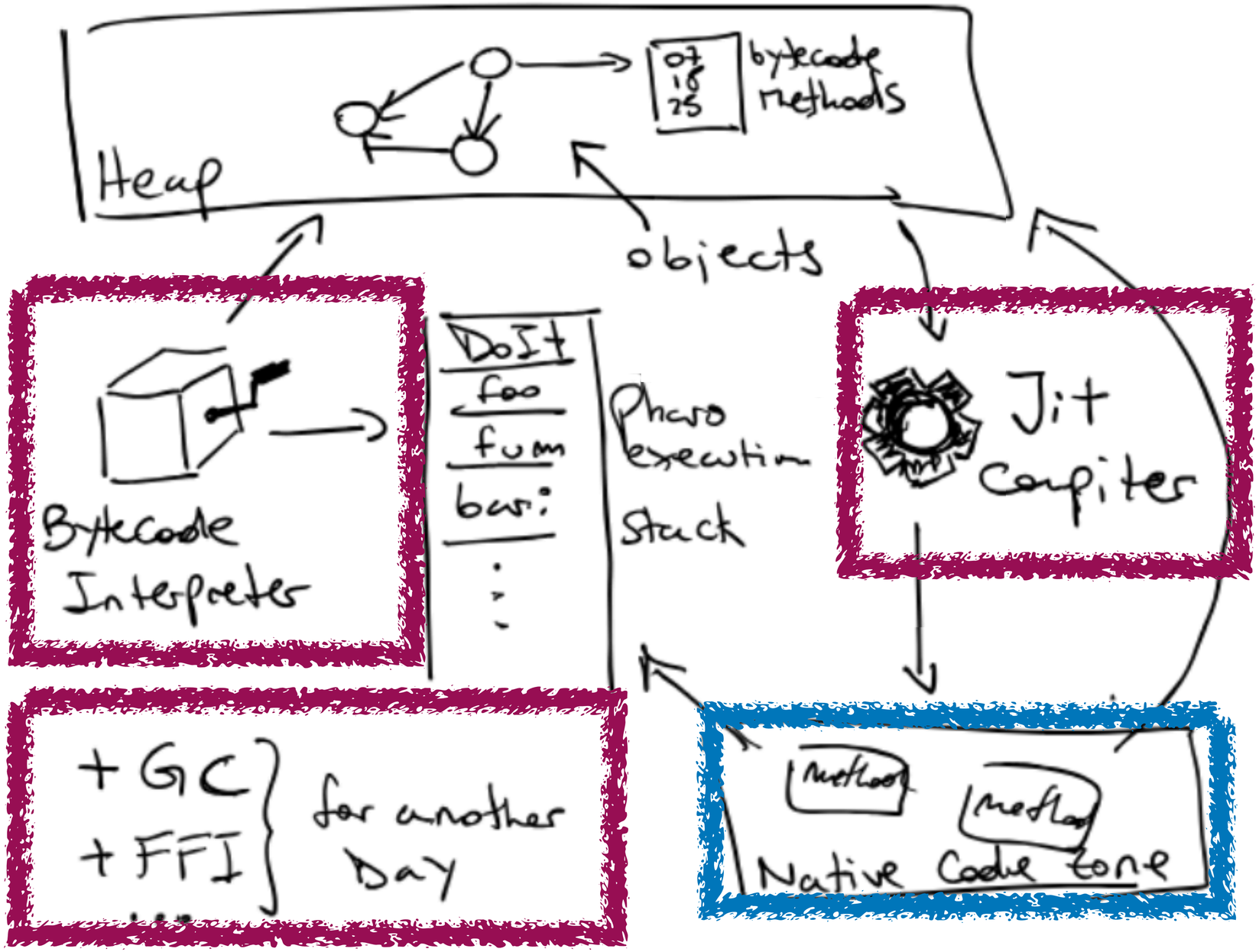
What we know so far…

VM C Runtime

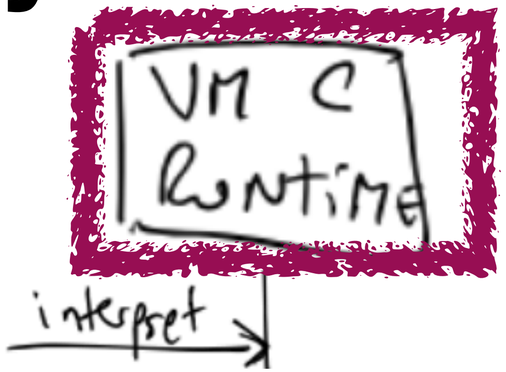
Compiled Ahead of Time
Slang -> C -> Native code

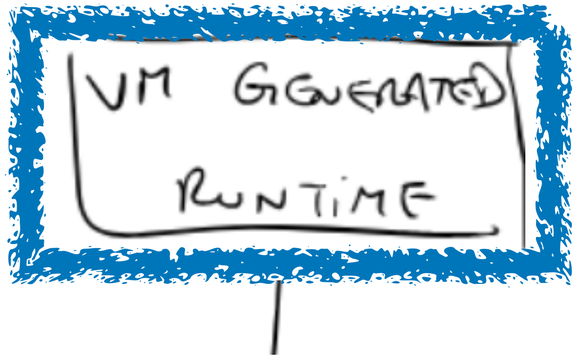
VM Generated Runtime

Compiled at Run Time
Bytecode -> Native code

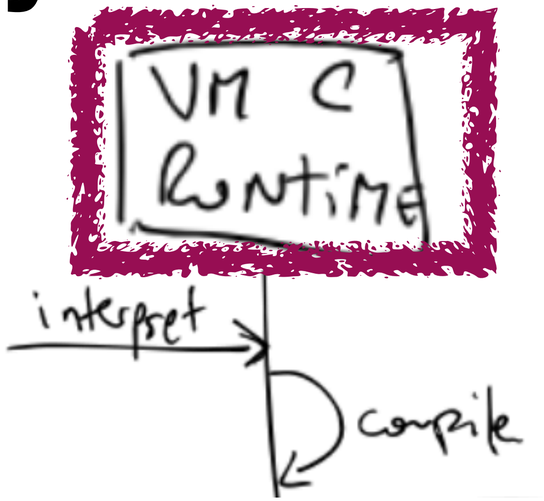


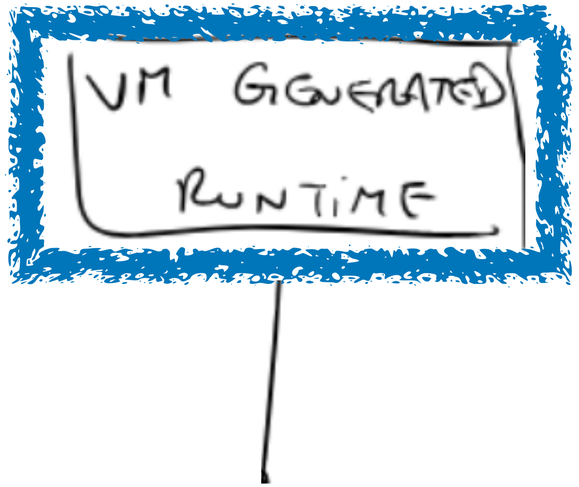
How they interact at run time



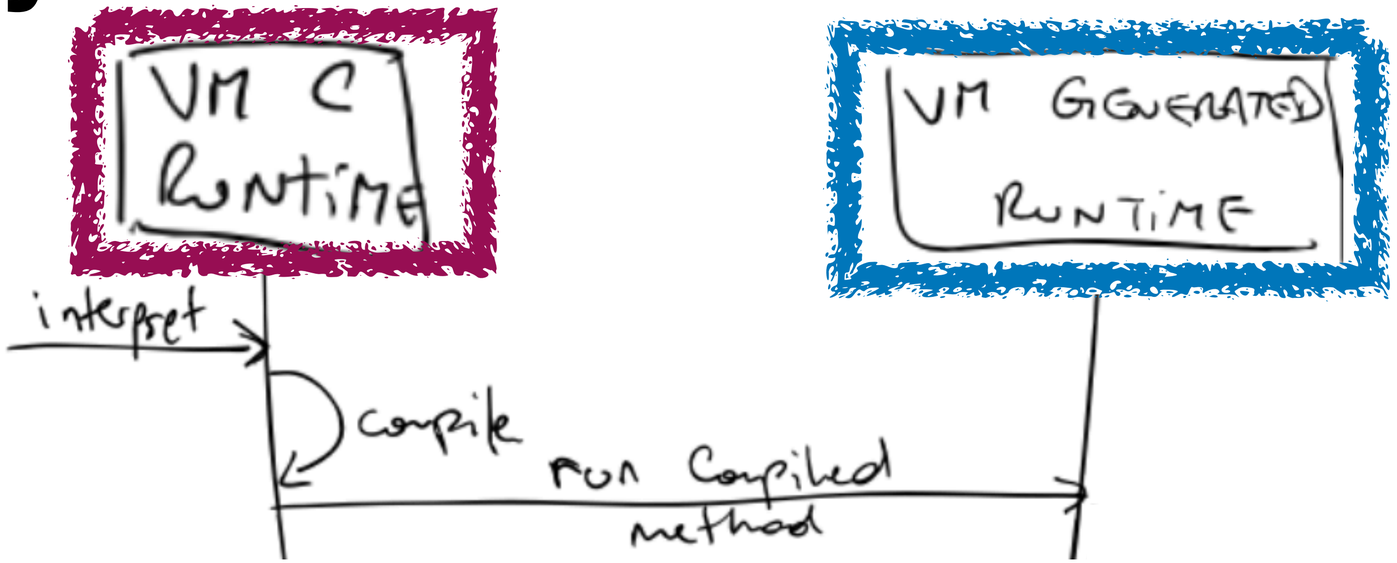


How they interact at run time

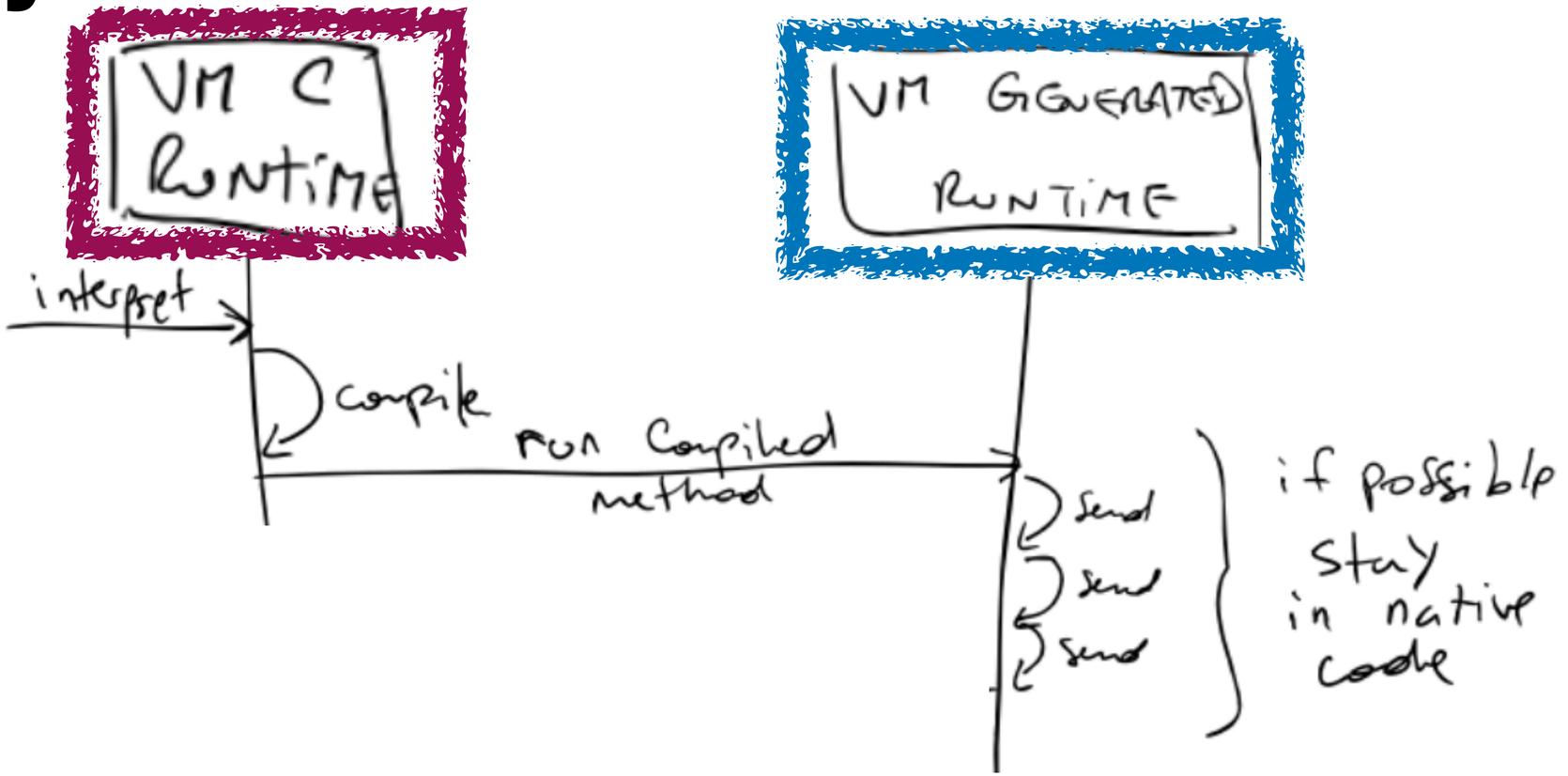




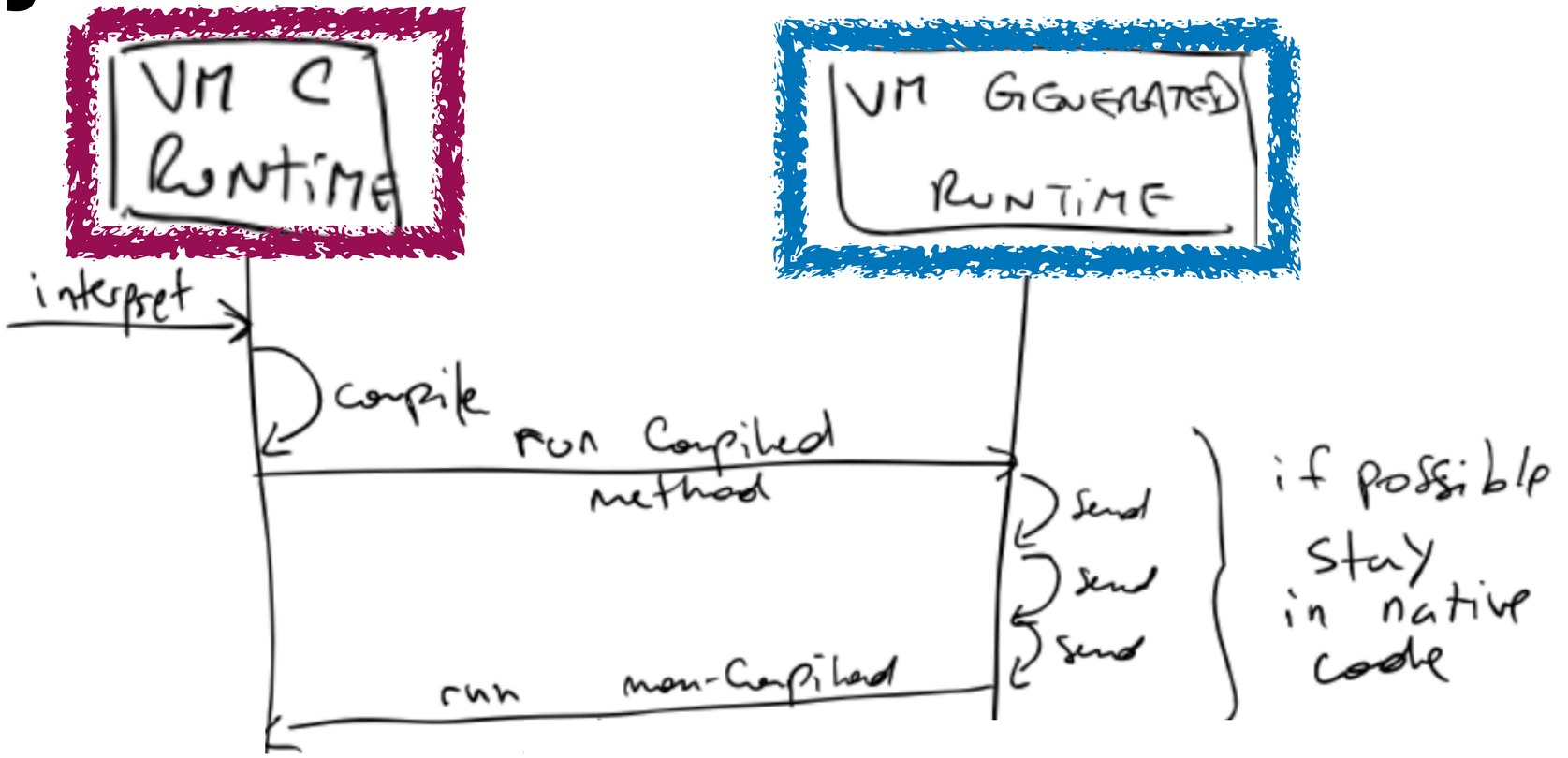
How they interact at run time



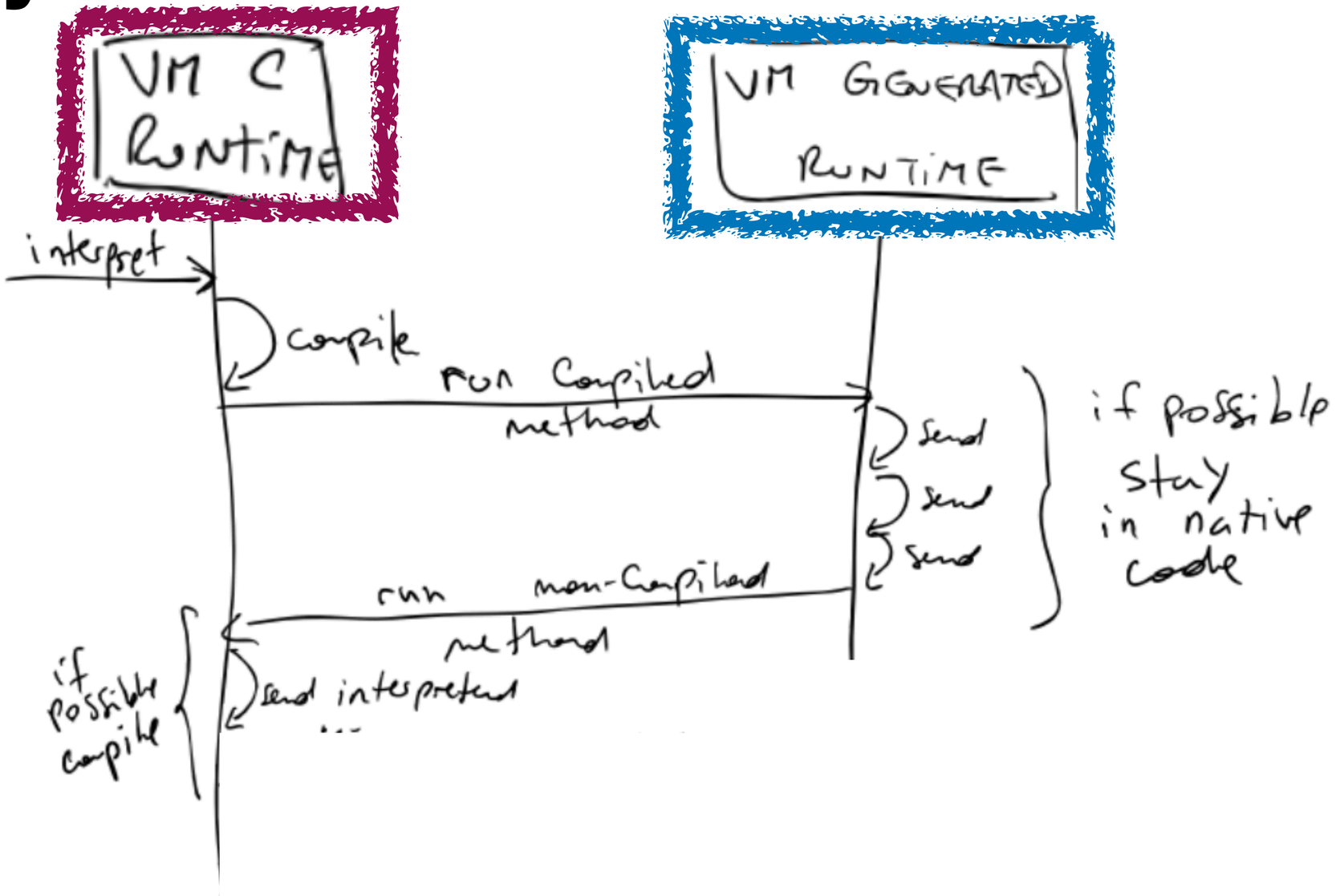
How they interact at run time



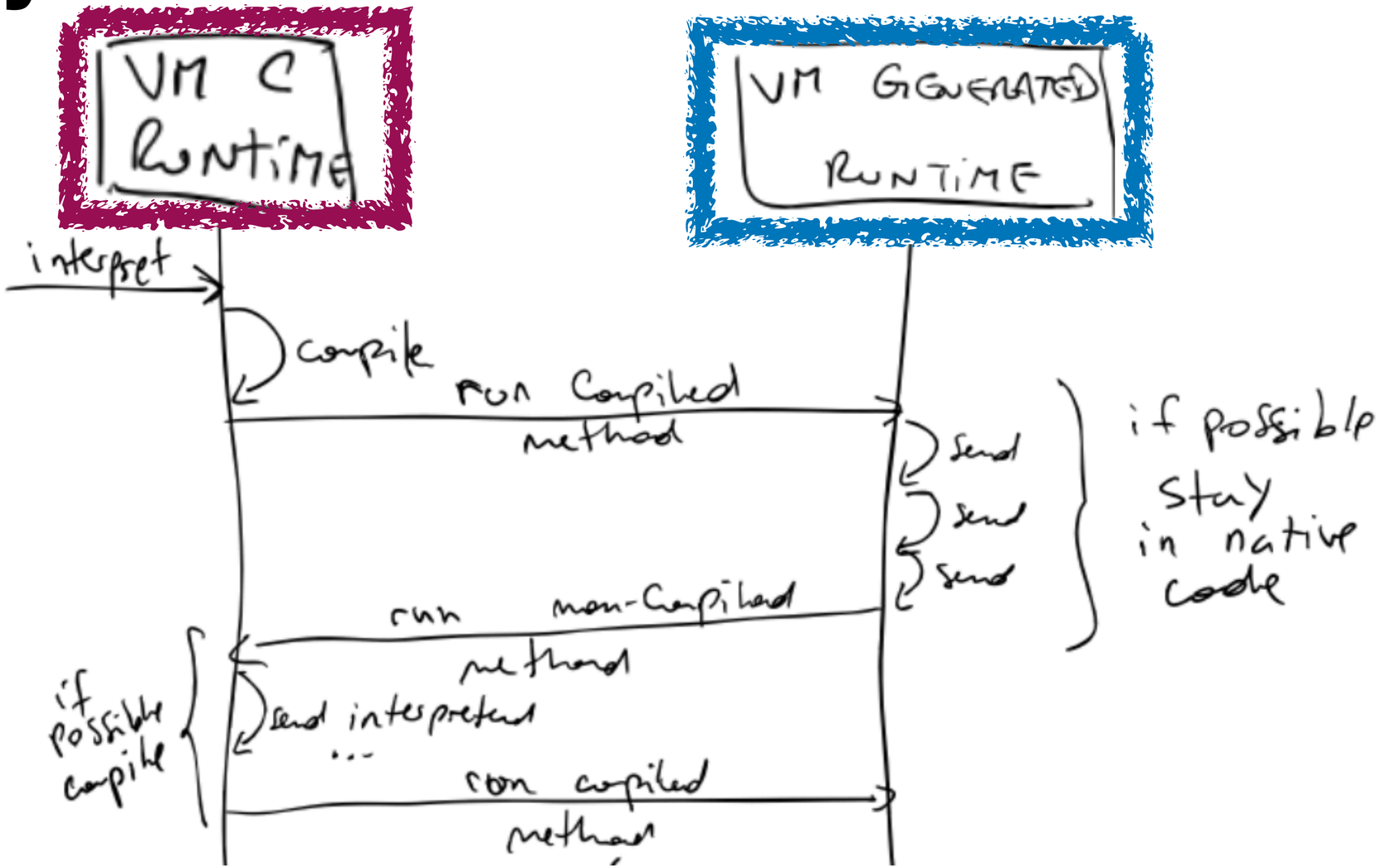
How they interact at run time



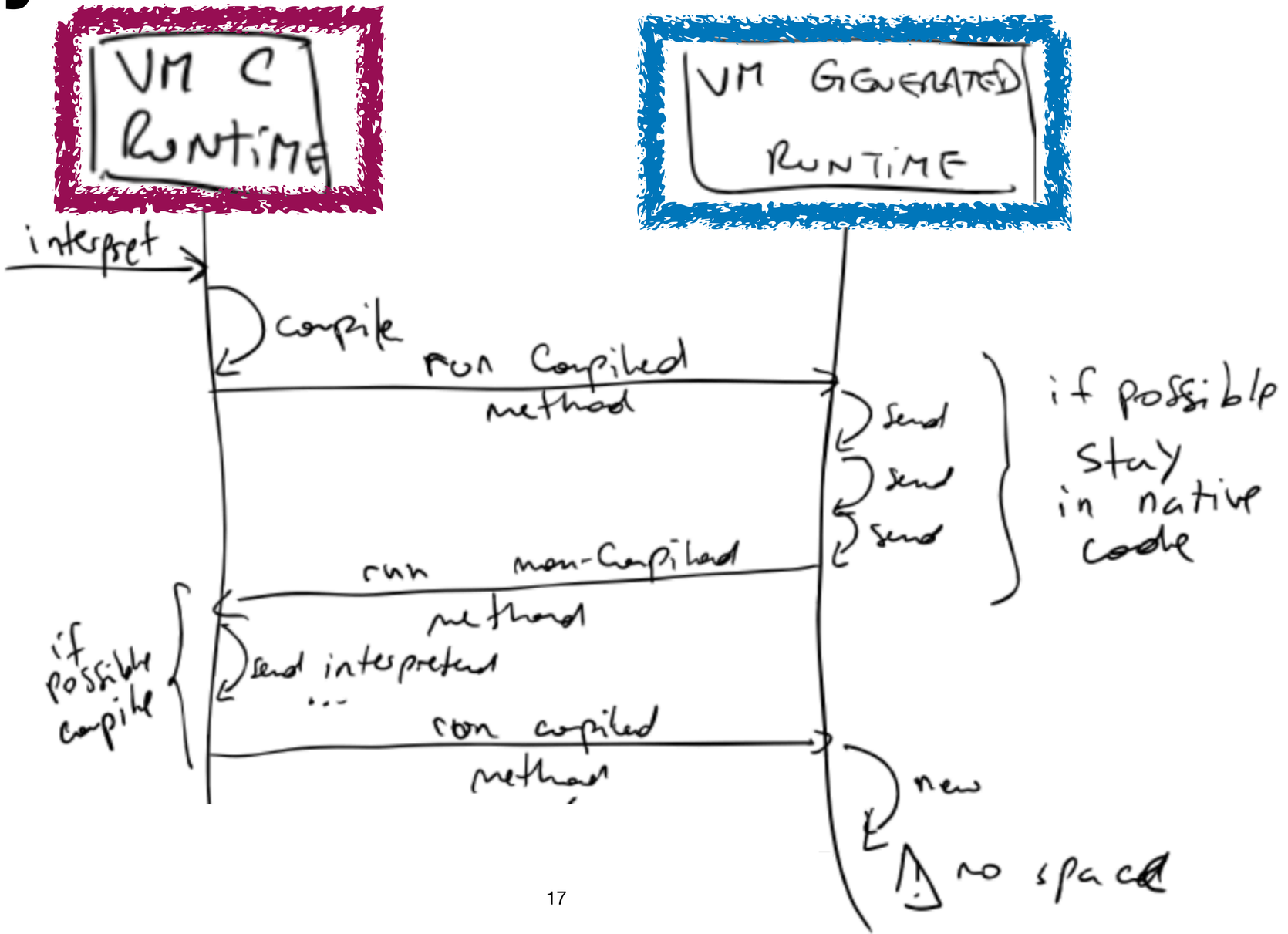
How they interact at run time



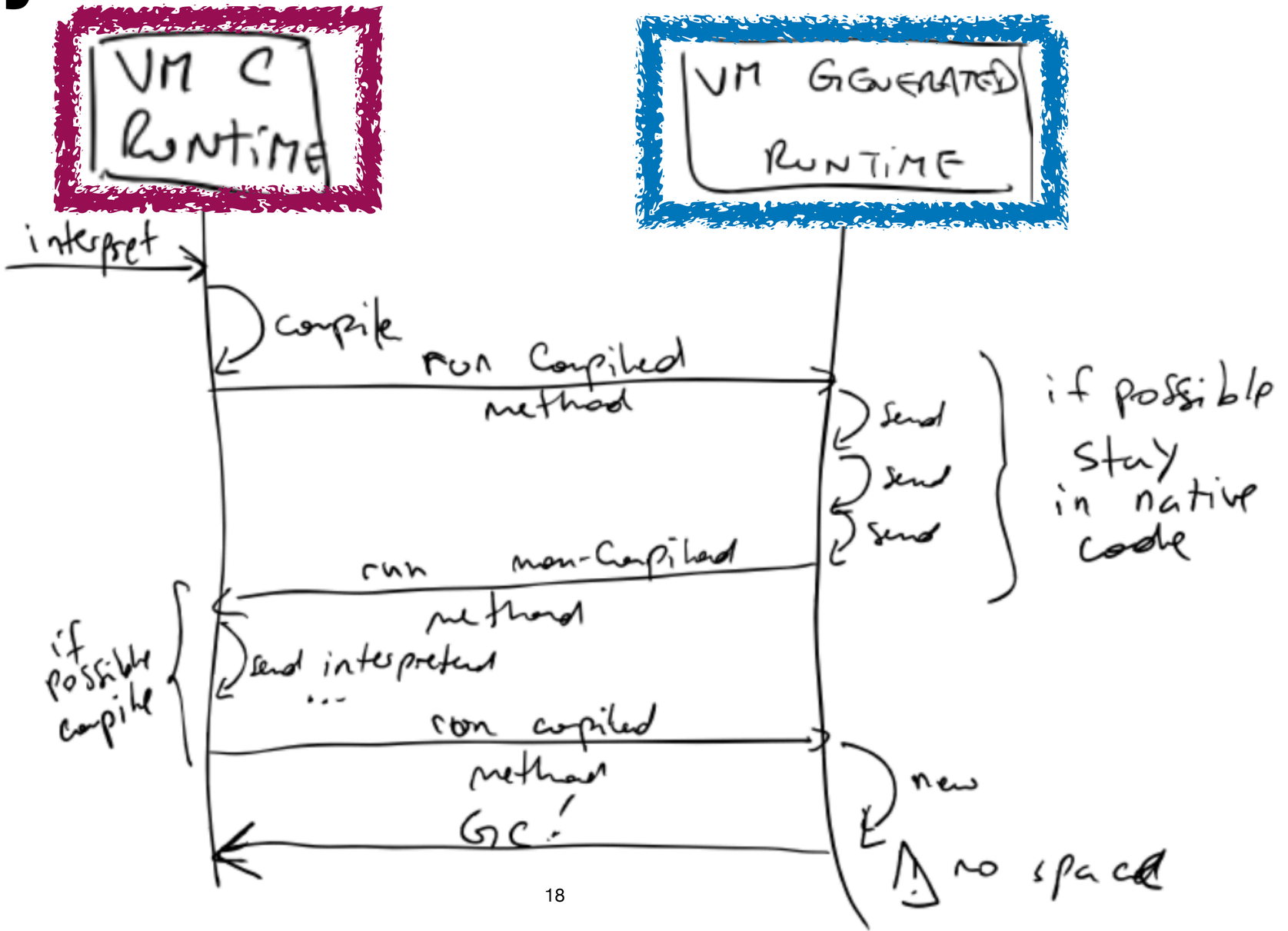
How they interact at run time



How they interact at run time



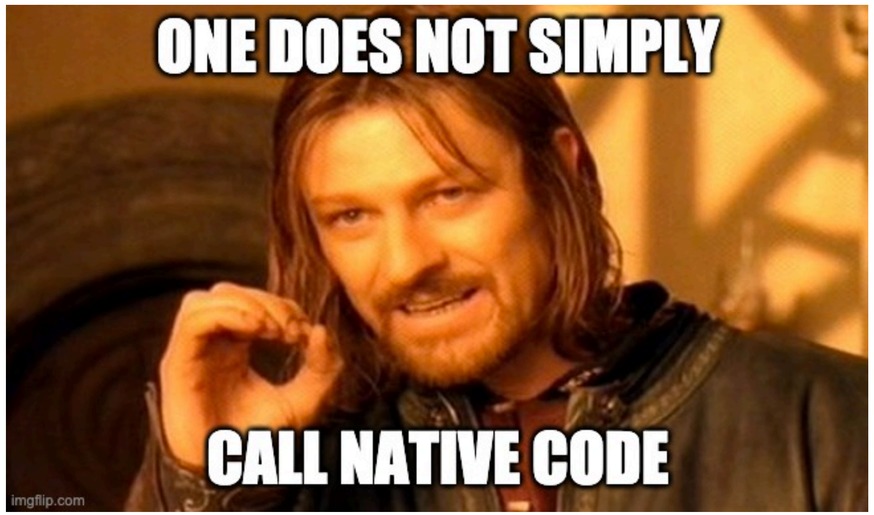
How they interact at run time

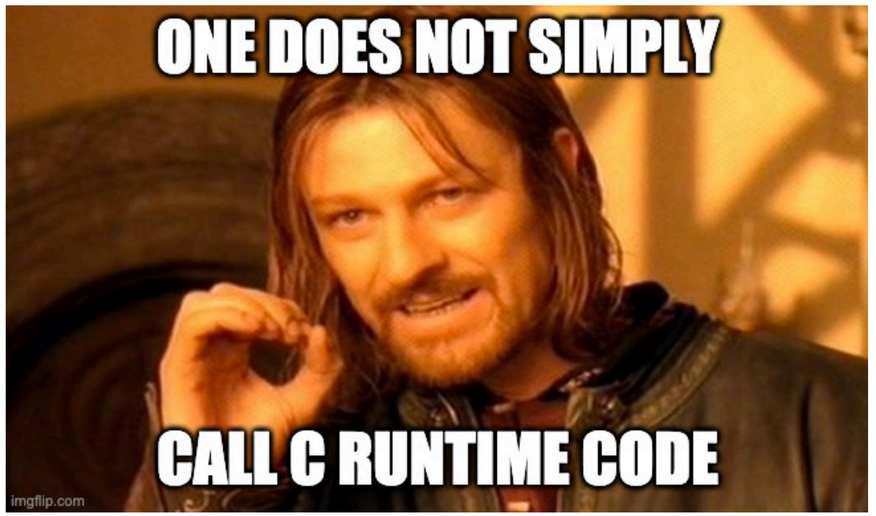


VM C Runtime vs Generated Runtime

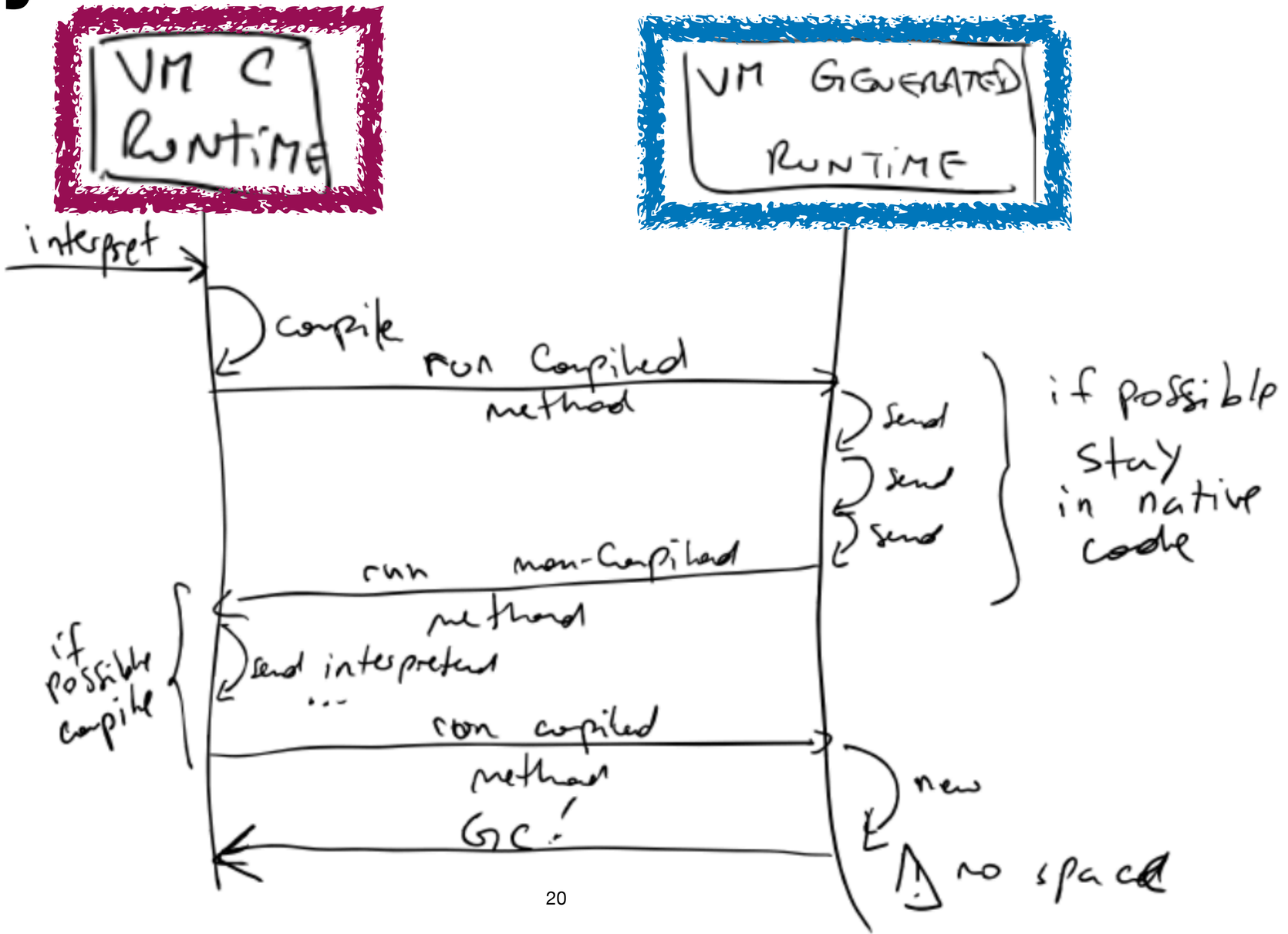
- The JITted methods do not include code for all cases
- Slow paths, complex cases (such as the GC) are NOT compiled

- Instead => delegate to the VM C code … But…

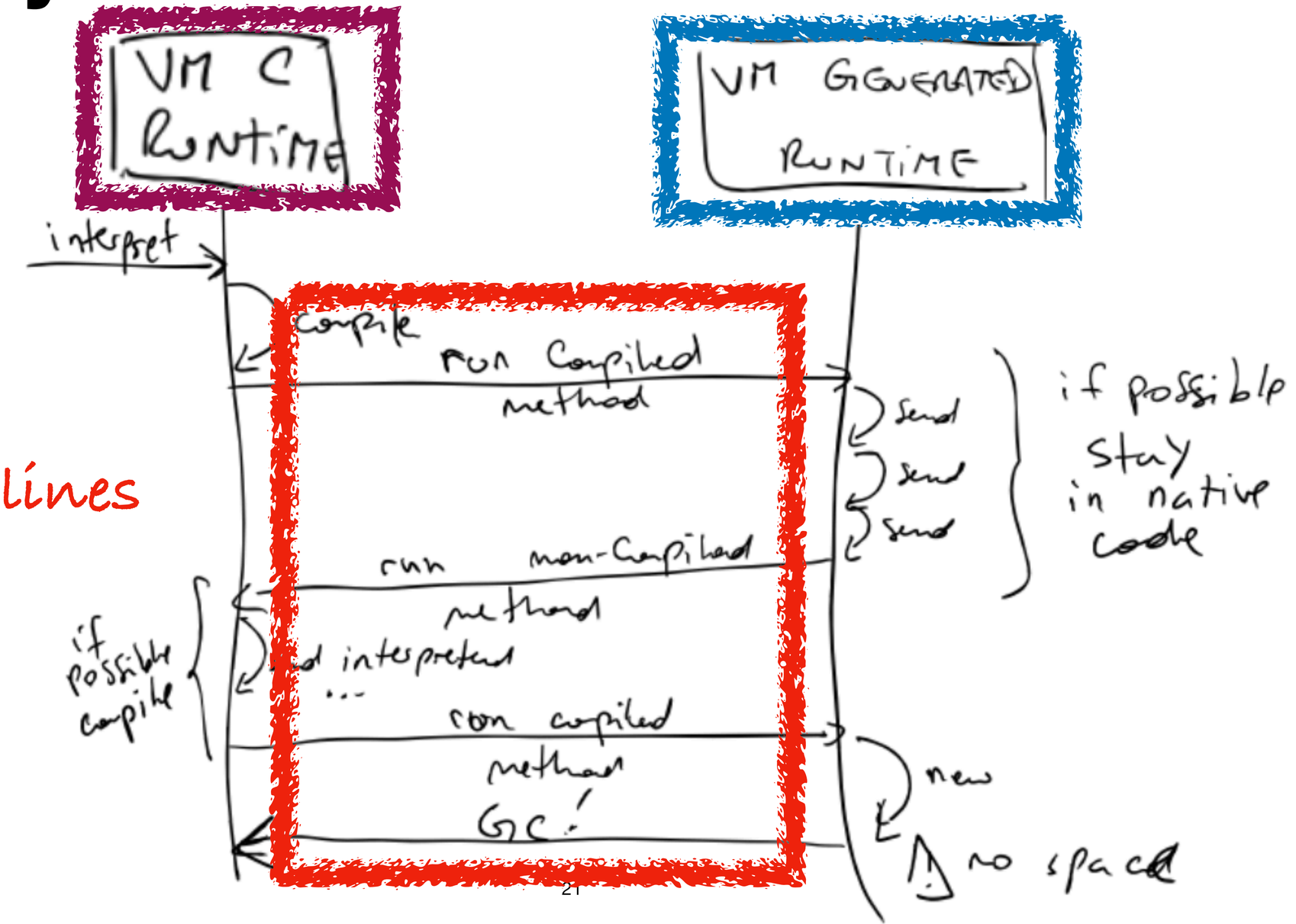




How they interact at run time



How they interact at run time



Trampolines

Trampolines vs Reverse Trampolines

Two kinds of trampolines

- Trampolines:
  - Generated native code —> C code

- Reverse trampolines:
  - C code —> Generated native code

What are trampolines used for?

Mostly, slow execution paths

- Send a message to a non-JITted method
- PICs
- Call the GC
- Slow allocations
- Immutability checks
- Managing GC invariants (e.g., remembered set)
- Must be boolean
- …

What are reverse trampolines used for?

- Call JITted methods
- Low level Routines (e.g., get the state of the machine)

- That’s most of it

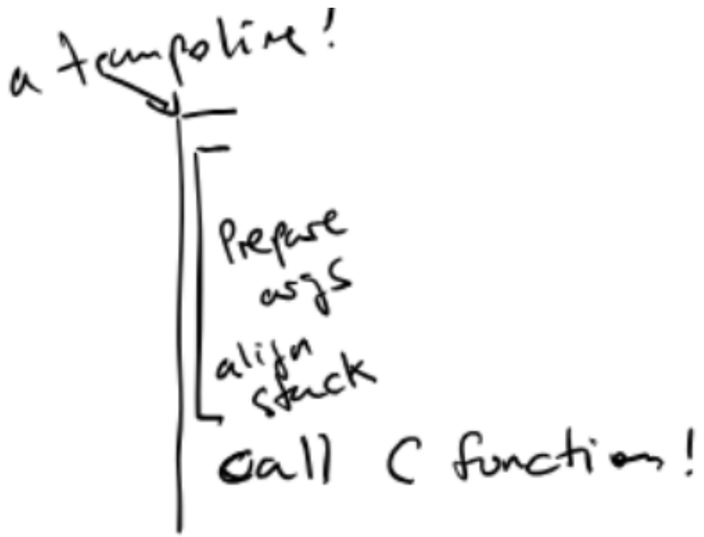
Why do we need trampolines?
Communication Problems

- The C functions where compiled by ****some**** C compiler
  - **We don’t know:** Are they optimised? What registers do they use?
  - **We know:** they have a standard calling convention
  - We have to be conservative and consider C functions could do anything!
- Our generated native code
  - Does not follow the C calling convention
  - Runs in a separate stack!

Trampolines

Calling the VM C runtime from the Generated Runtime

- Native code routines that make the transition
- Jitted methods call a trampoline
- The trampoline sets-up the conditions to call C code
- Then calls the C code

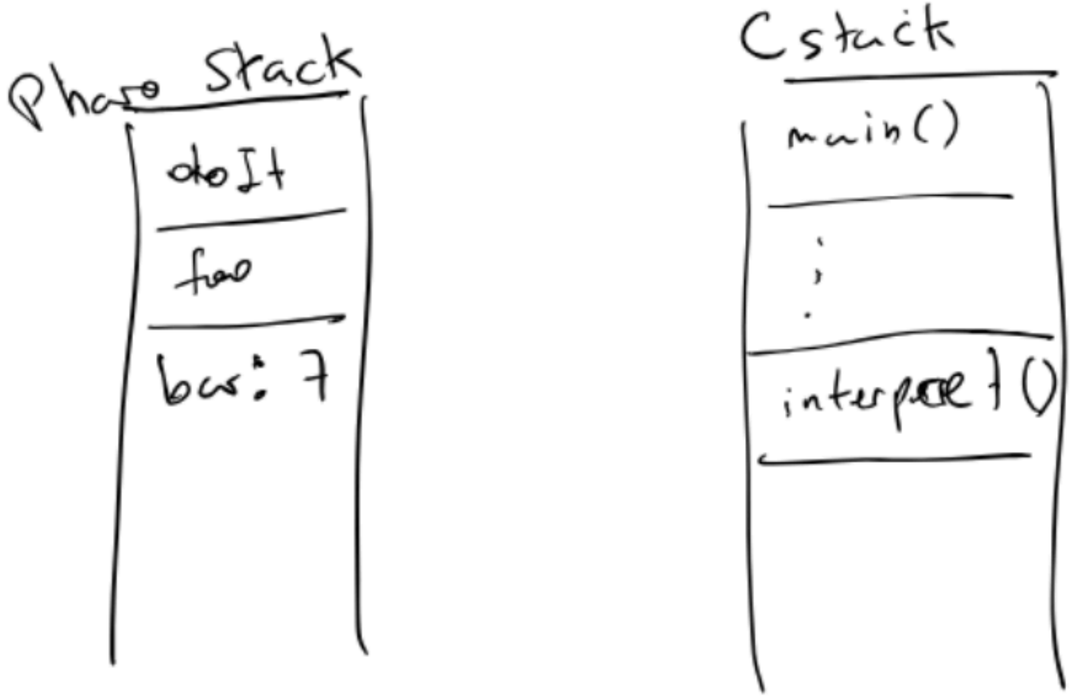


Two Execution Stacks

One for the Pharo execution, one for the VM C runtime execution

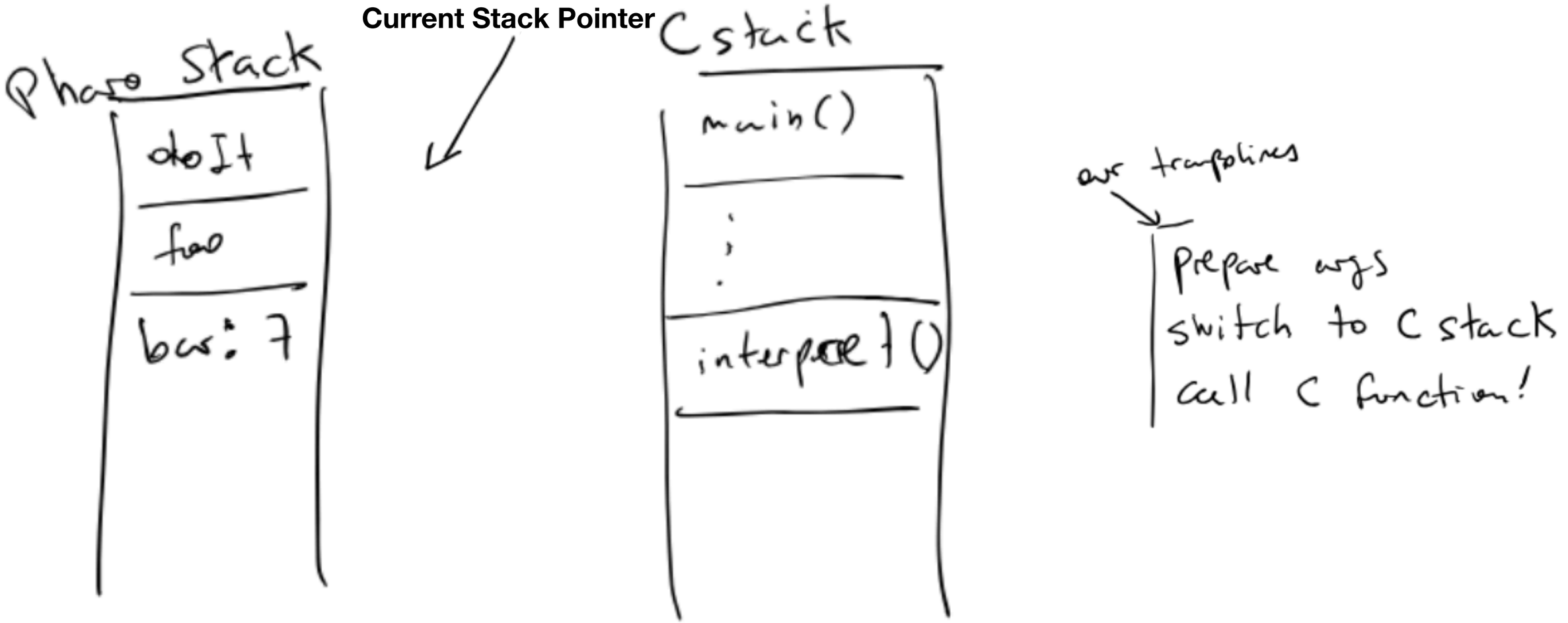
- The Pharo execution is persistent (e.g., when you save the image)
- And has green threads!
- The C execution is single-threaded non-persistent

=> Two stacks to **separate** concerns



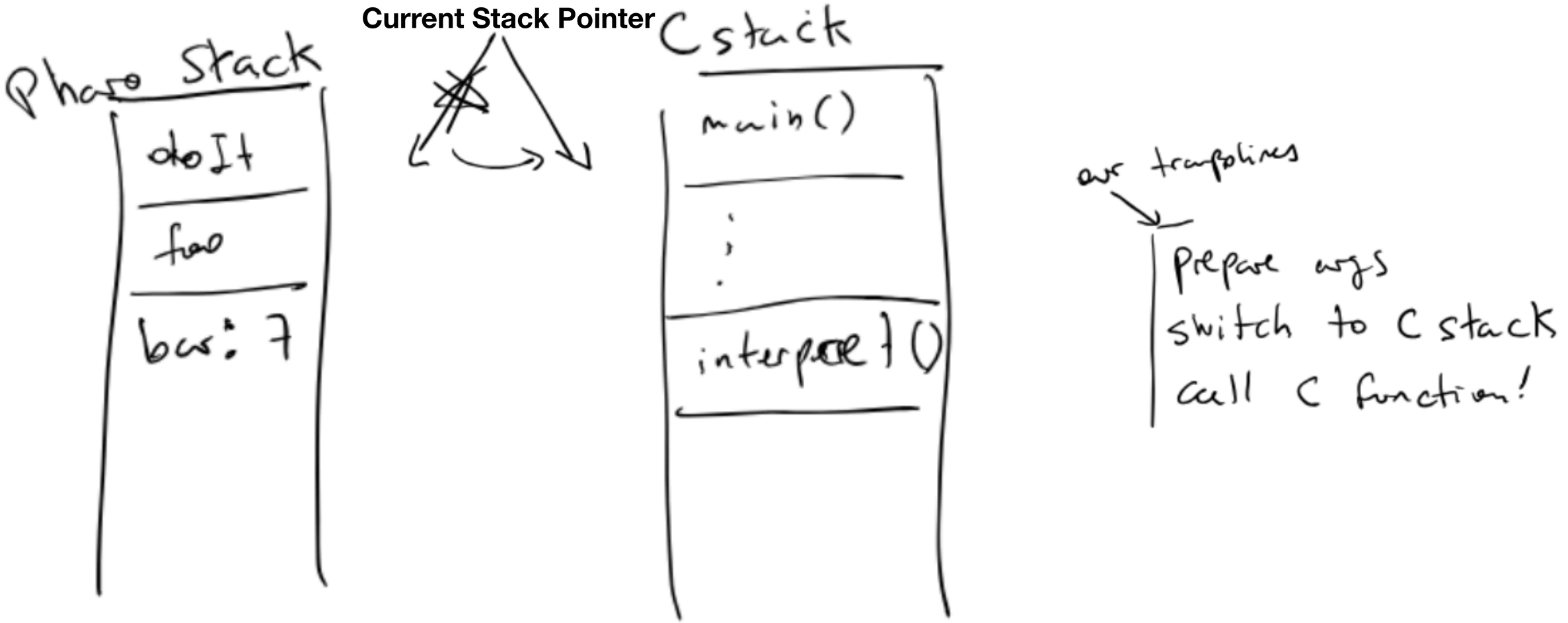
Trampolines in the Pharo VM

Switching to the C stack



Trampolines in the Pharo VM

Switching to the C stack



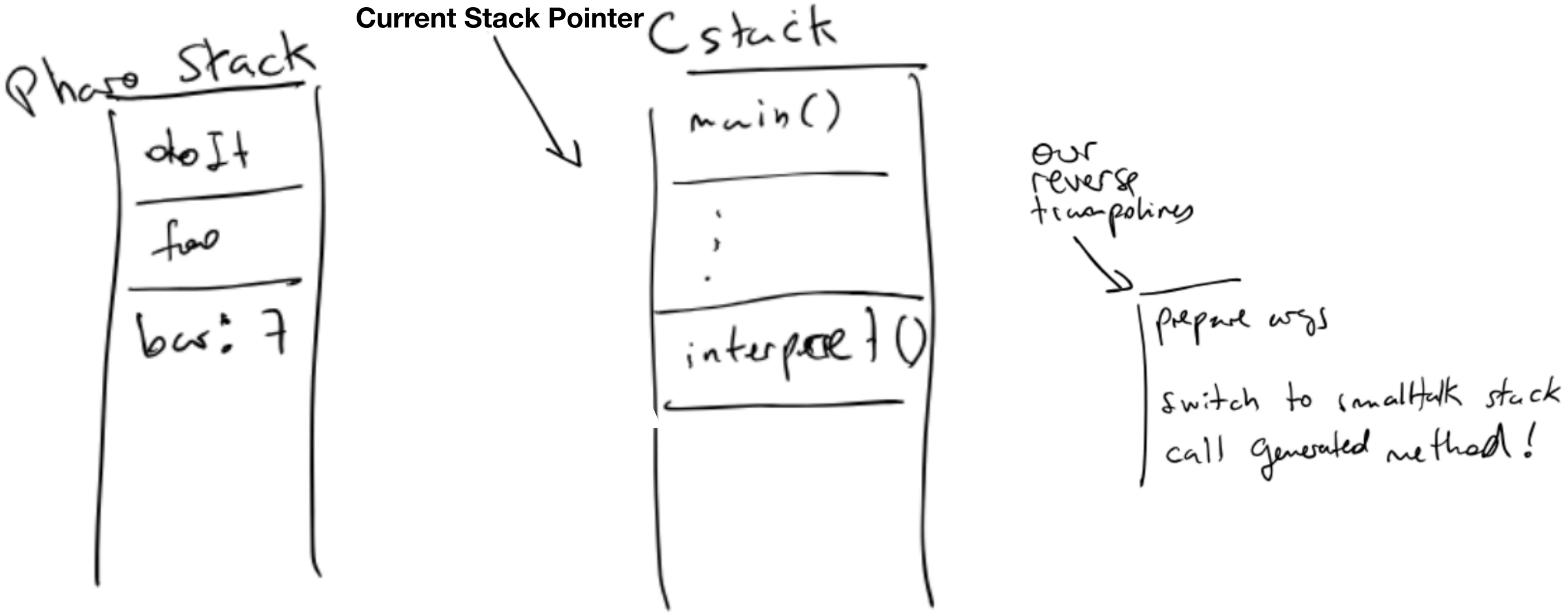
Reverse Trampolines (enilopmarts)

Calling the generated runtime from C runtime

- Native code routines that make the transition
- The VM C runtime calls a reverse trampoline
- The reverse trampoline sets-up the conditions to call the generated runtime
- Then calls the generated code

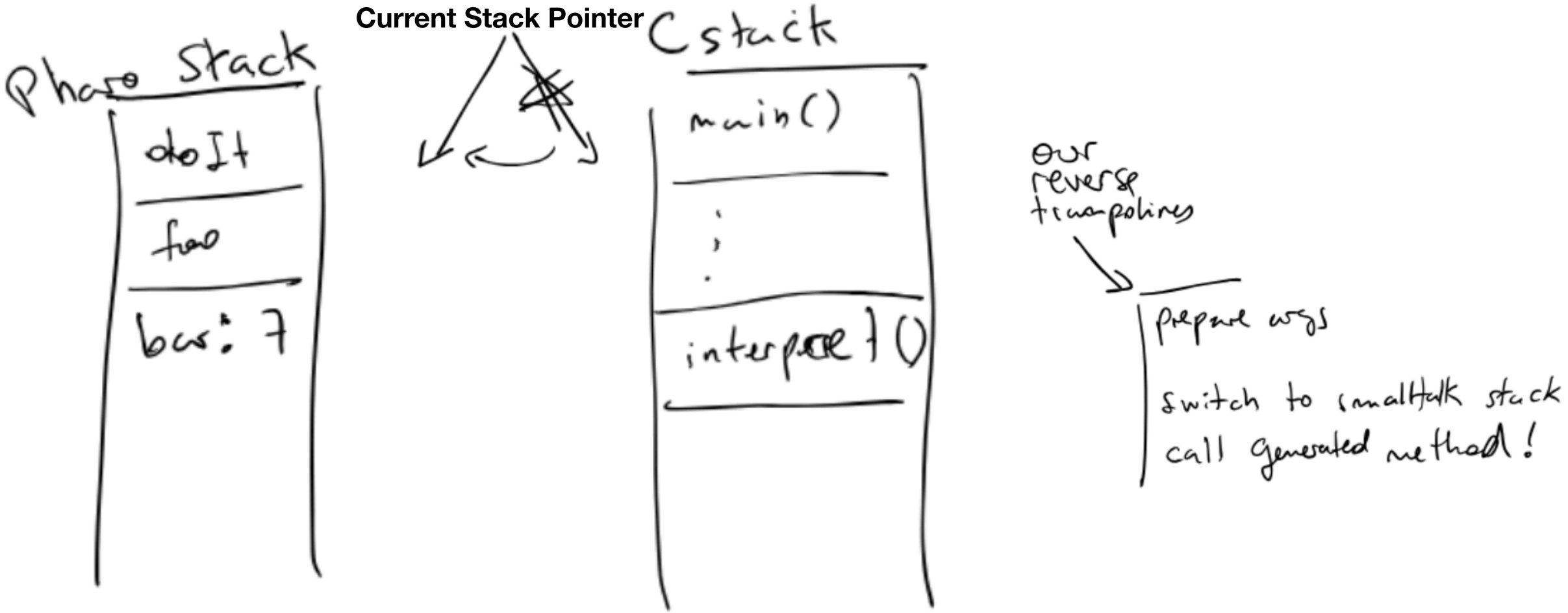
Reverse Trampolines in the Pharo VM

Switching to the Pharo Stack



Reverse Trampolines in the Pharo VM

Switching to the Pharo Stack



Where are the trampolines?

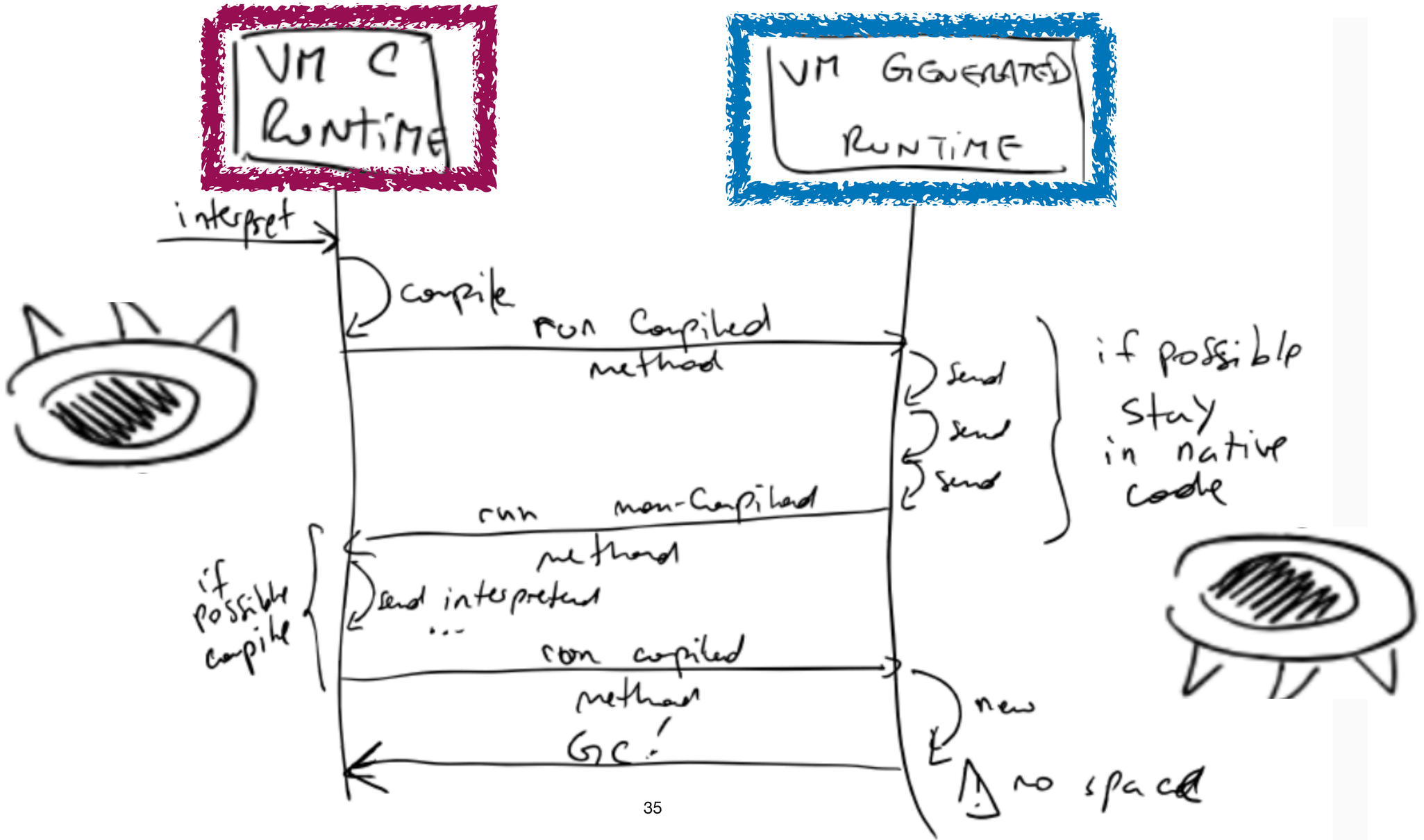
The Native Code Zone

- It has Intrinsics + Jitted Methods
- Intrinsics: routines that are pre-defined by the compiler
  - Intrinsics are generated at VM startup and stay there
  - Trampolines are not the only intrinsics (e.g., marrying a machine code frame to a context object)

Other details about trampolines

- Reverse trampolines do not actually **call**, they **return** to generated code, to avoid having the reverse trampoline in the stack
- Reverse trampolines do not receive arguments, arguments are passed through the smalltalk stack
  - They then pop the arguments and puts them in the right place before returning to the generated method
- Pharo Trampolines do not ensure that registers are properly saved-restored. It’s the caller method that ensures that. This is specially problematic since the called C function could do virtually **anything**, even smashing our registers!

Conclusion


VM C RUNTIME
VM GENERATED RUNTIME
interpret
compile
run Compiled method
send
send
send
if possible stay in native code
run non-Compiled method
if possible compile
send interpreted
...
run compiled method
GC!
new
no space