Meta-compilation of Baseline JIT Compilers with Druid

Programming ’25

PALUMBO Nahuel, POLITO Guillermo, TESONE Pablo, DUCASSE Stephane

Meta-compilation of Baseline JIT Compilers with Druid

Programming ’25

PALUMBO Nahuel, POLITO Guillermo, TESONE Pablo, DUCASSE Stephane

Inria

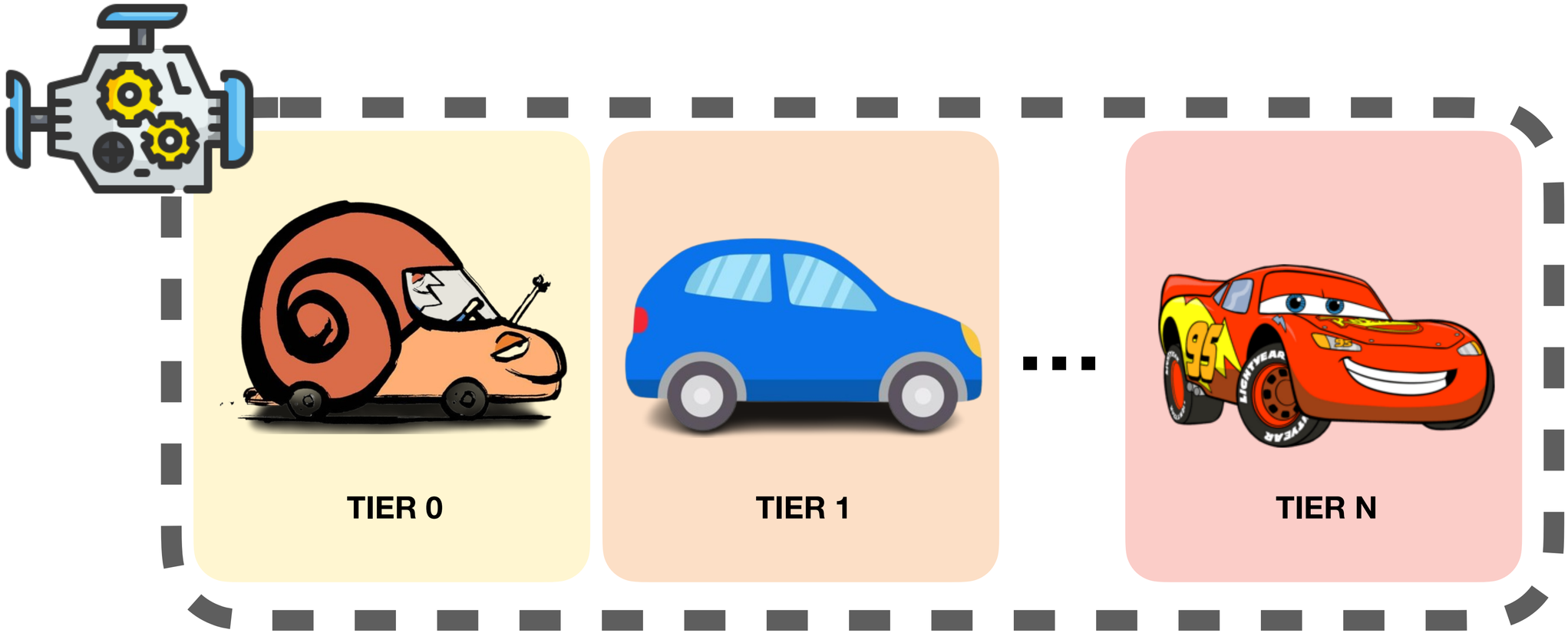
Evref

cnrs

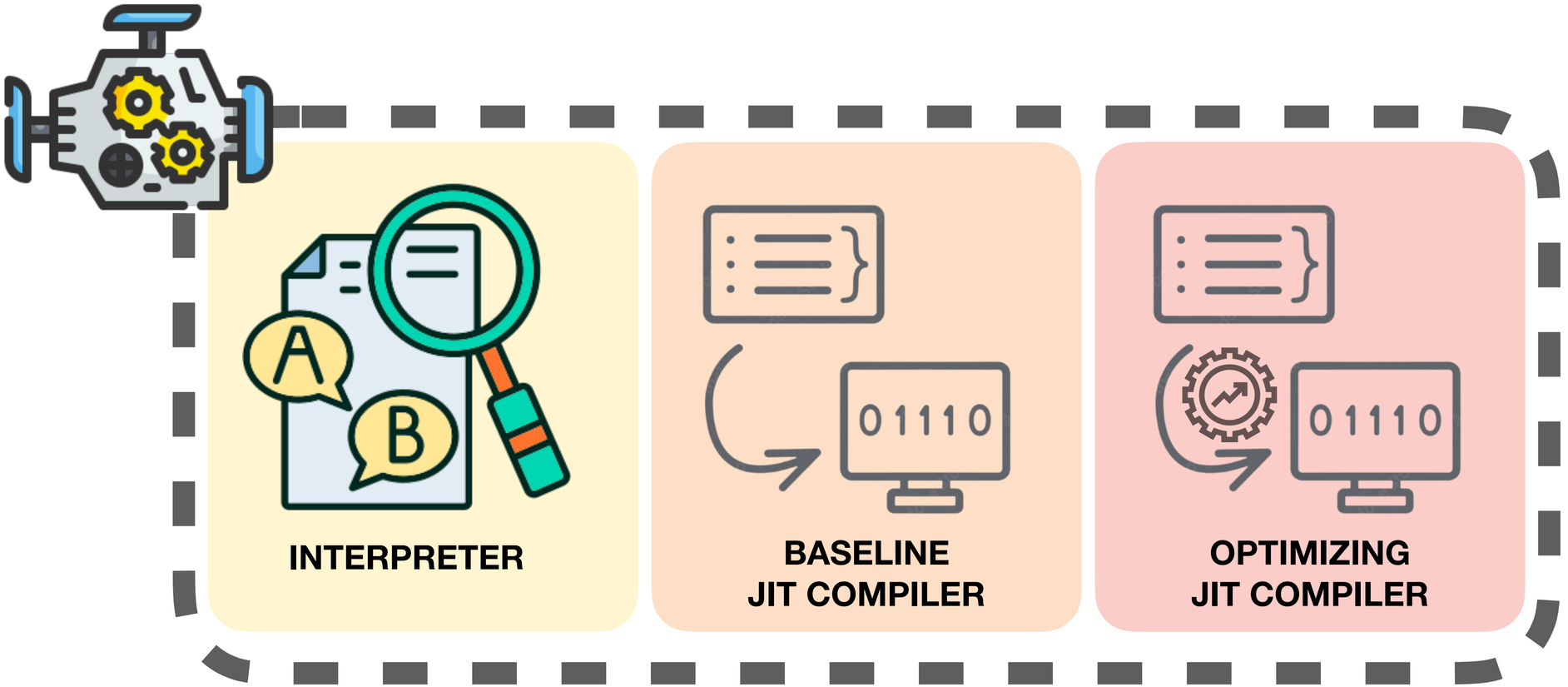
CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

Université de Lille 1 SCIENCES ET TECHNOLOGIES

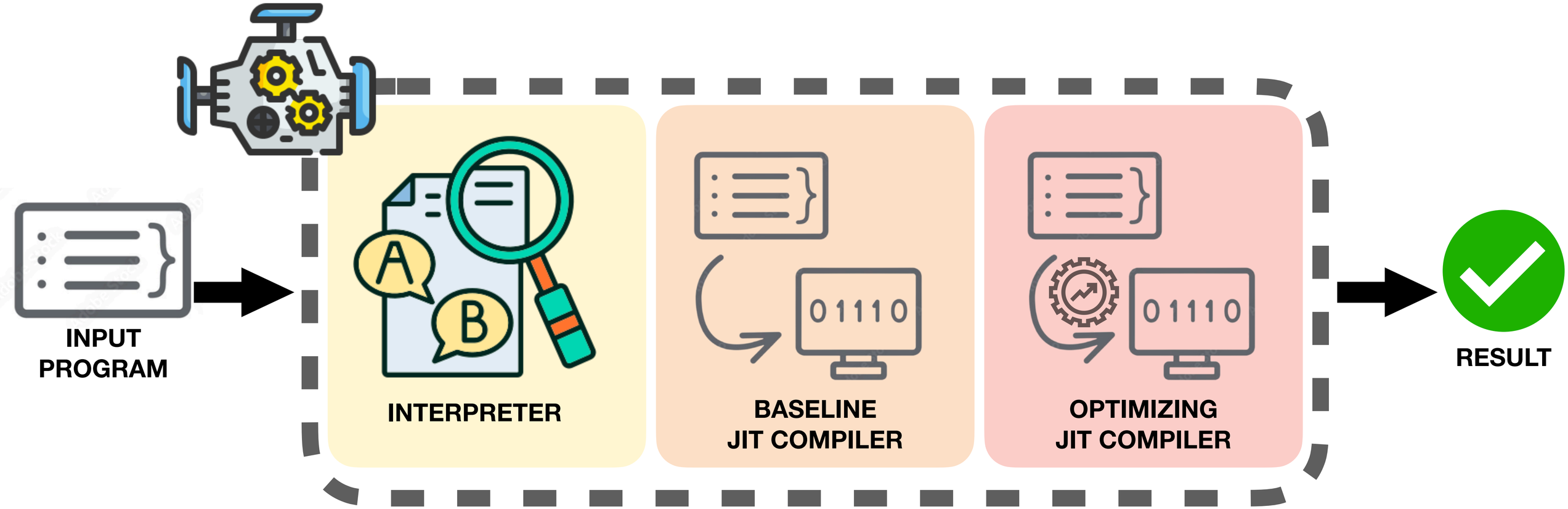
Multi-tier Virtual Machine



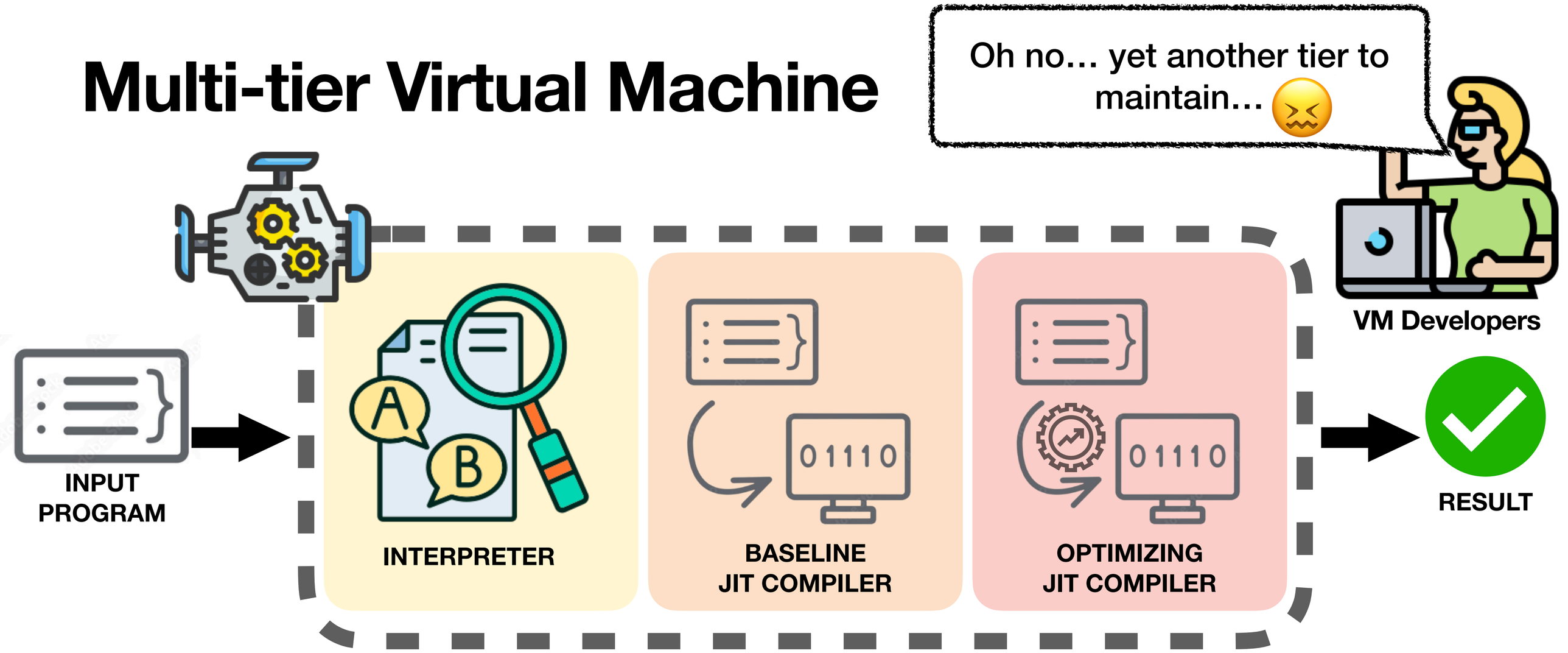
Multi-tier Virtual Machine



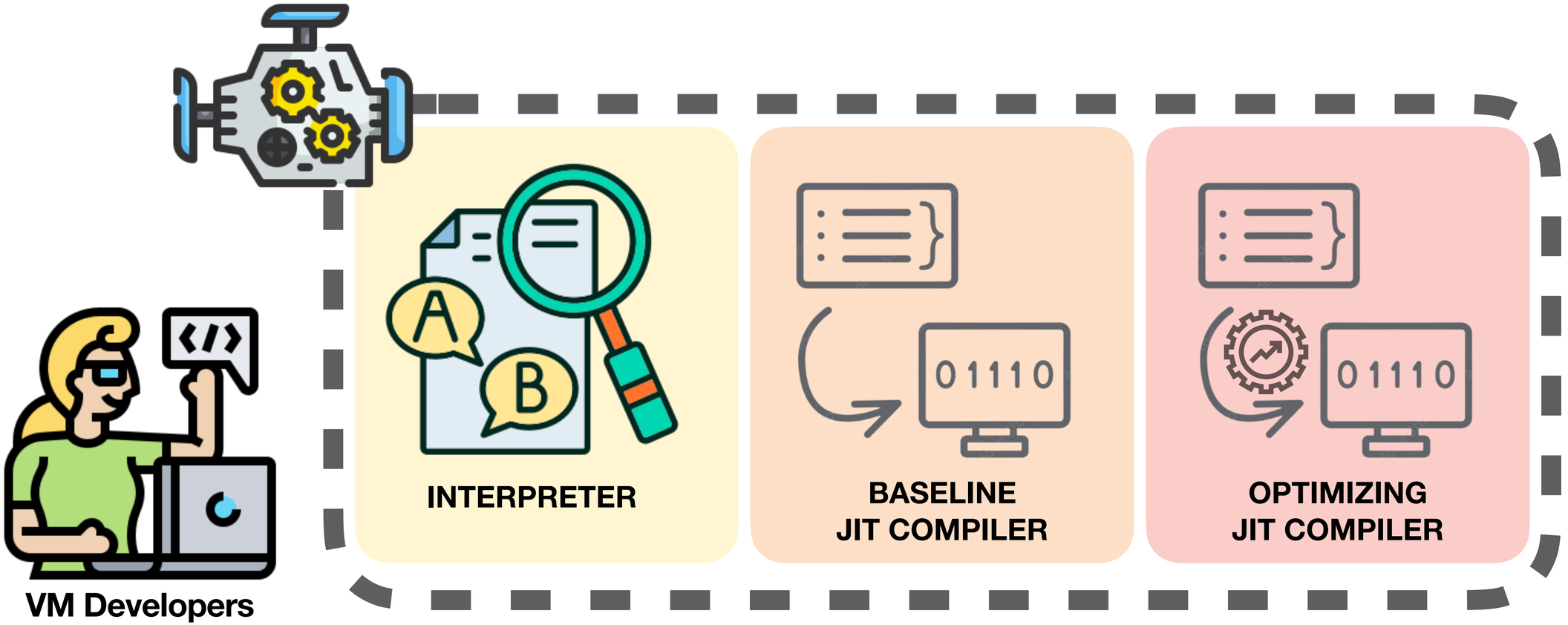
Multi-tier Virtual Machine



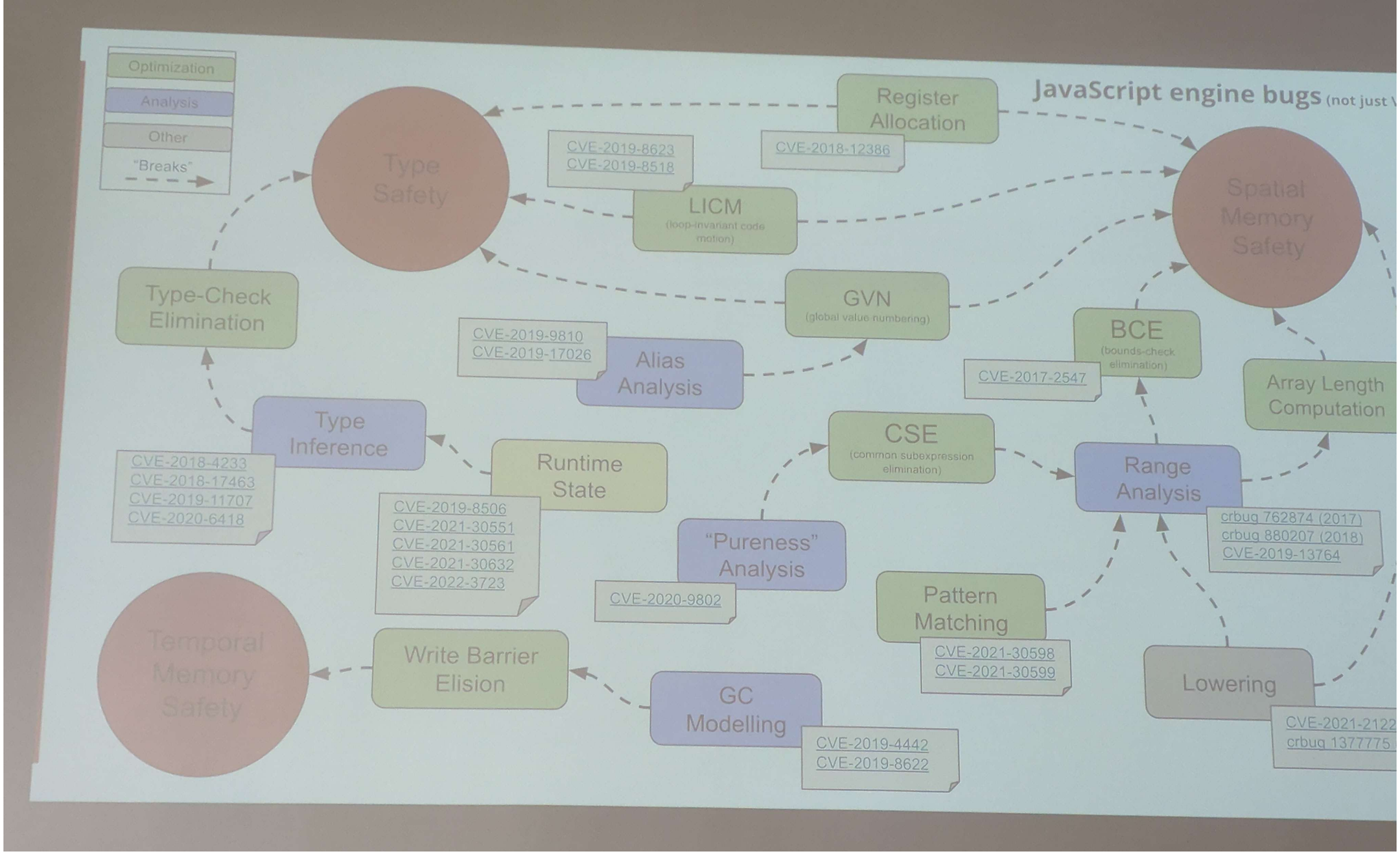
Multi-tier Virtual Machine



Multi-tier Virtual Machine

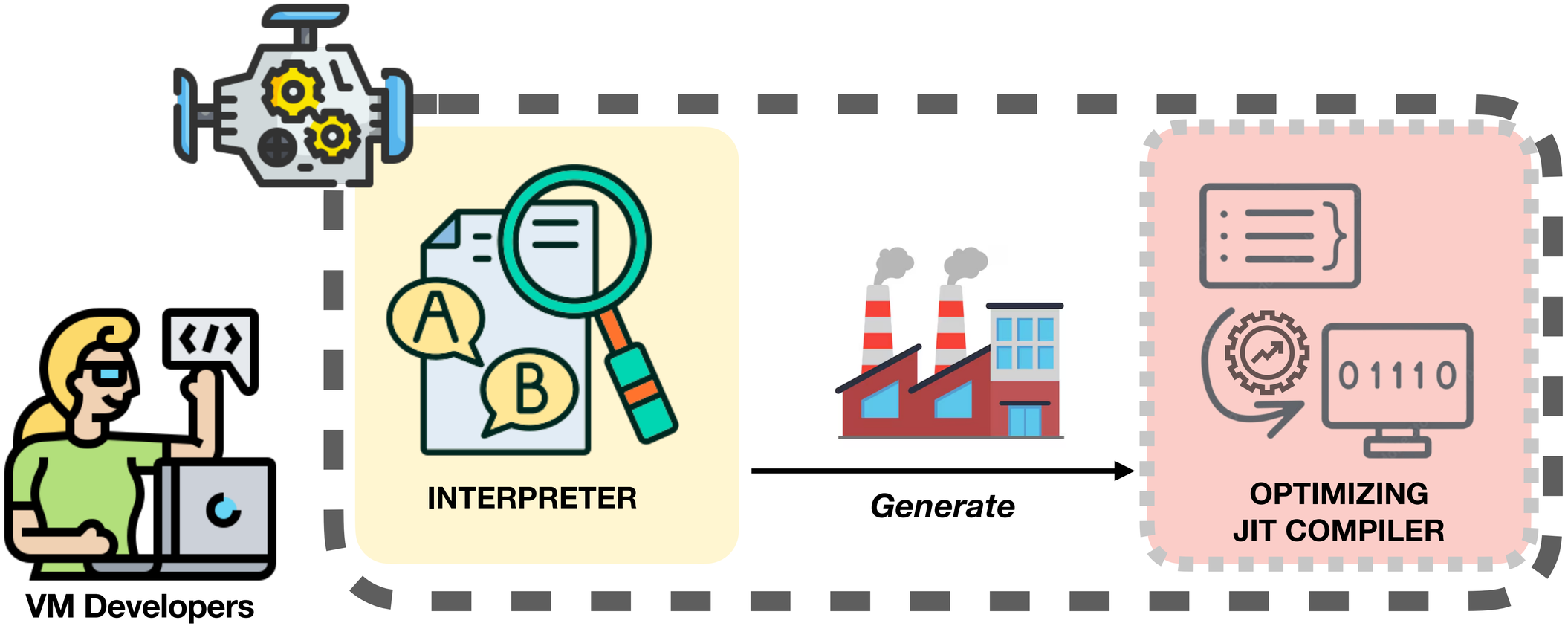


Easier ⟷ Harder

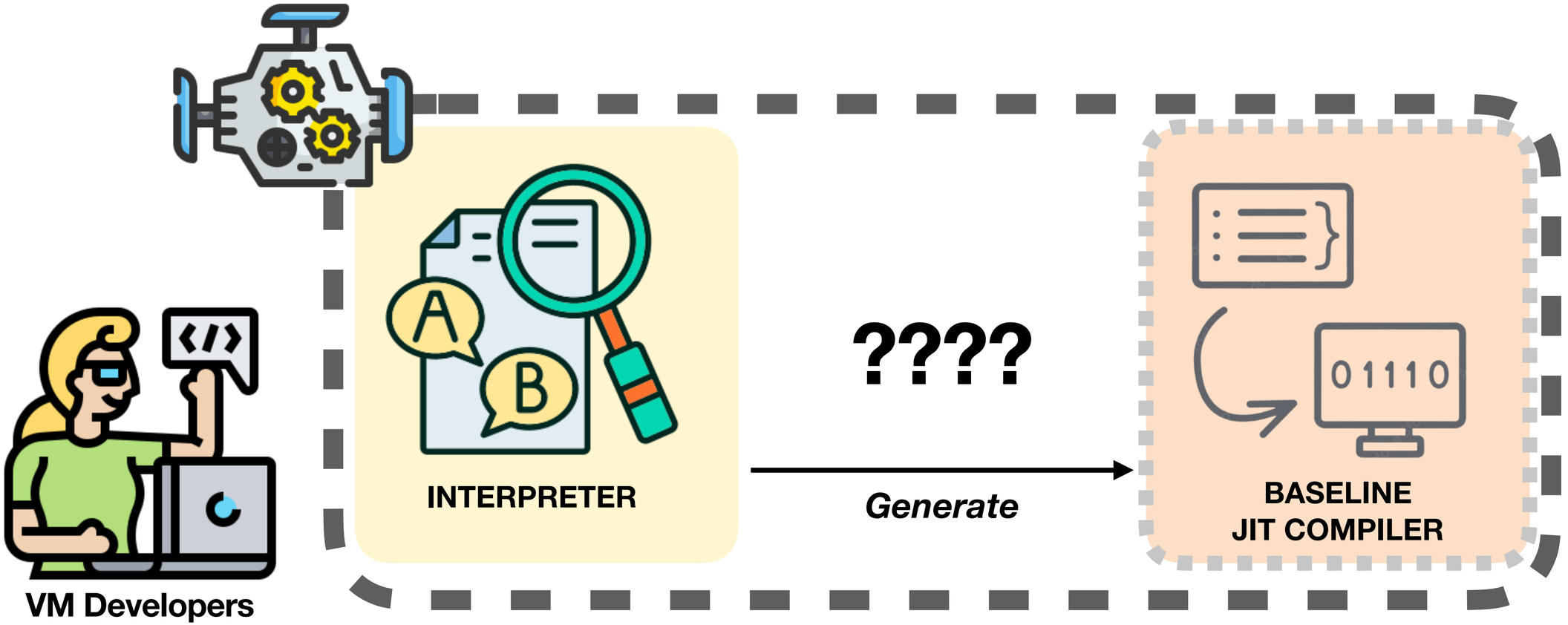


[*] Michael Lippautz. MoreVMs '25. The V8 Sandbox

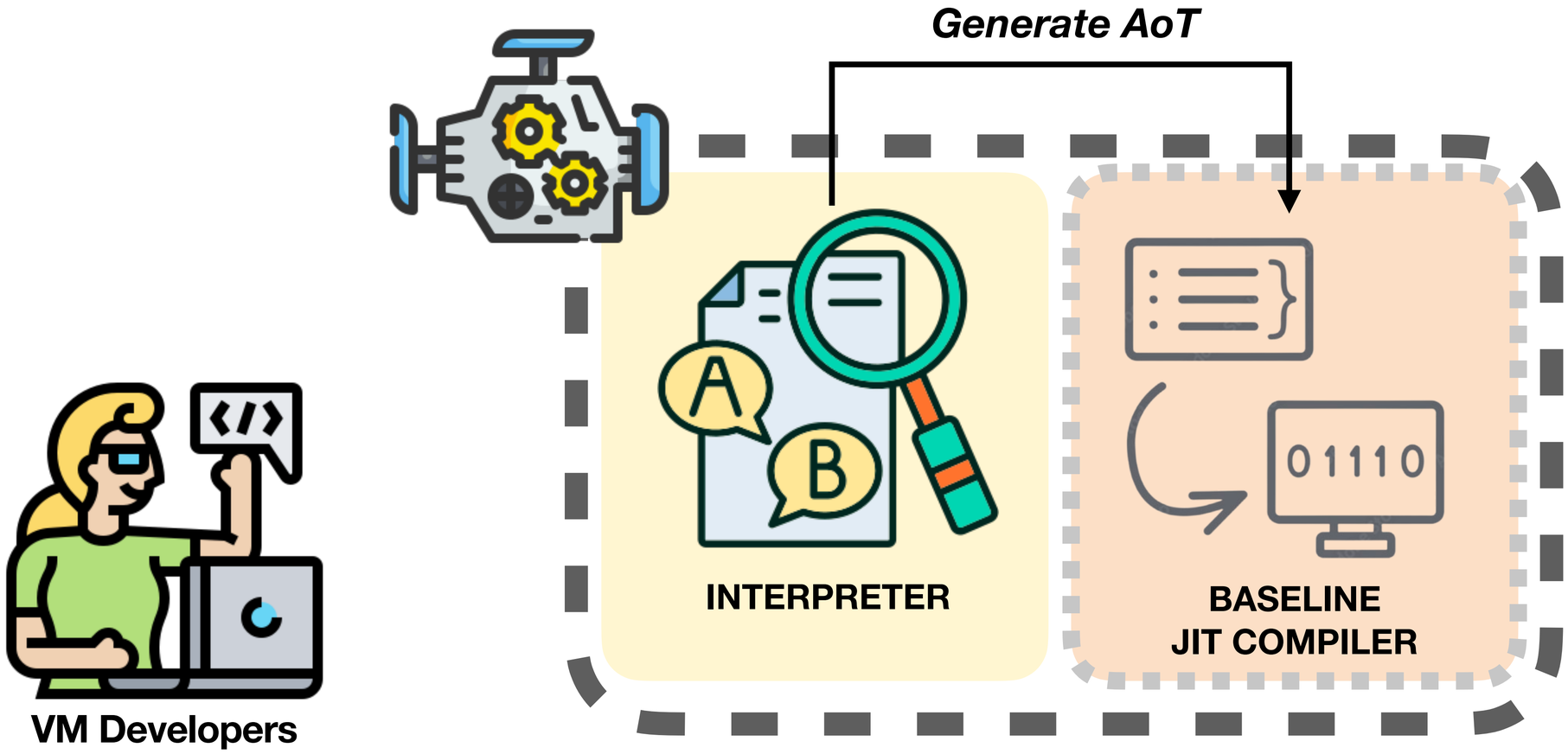
JIT Compiler Generators



JIT Compiler Generators

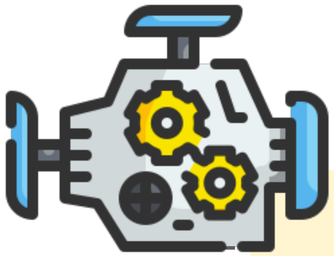


Baseline JIT Compiler Generators



Baseline JIT Compiler Generators

Source-to-source
meta-compiler

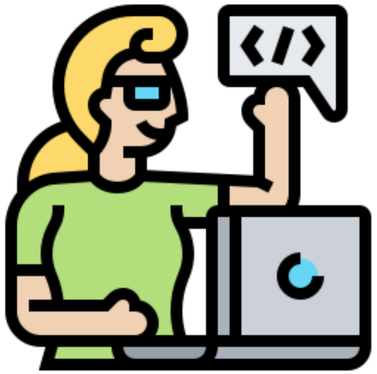


Generate AoT

DRUID

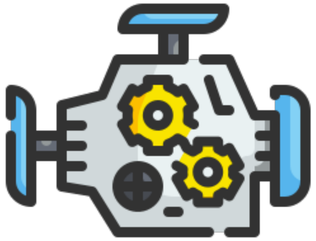
INTERPRETER

BASELINE
JIT COMPILER

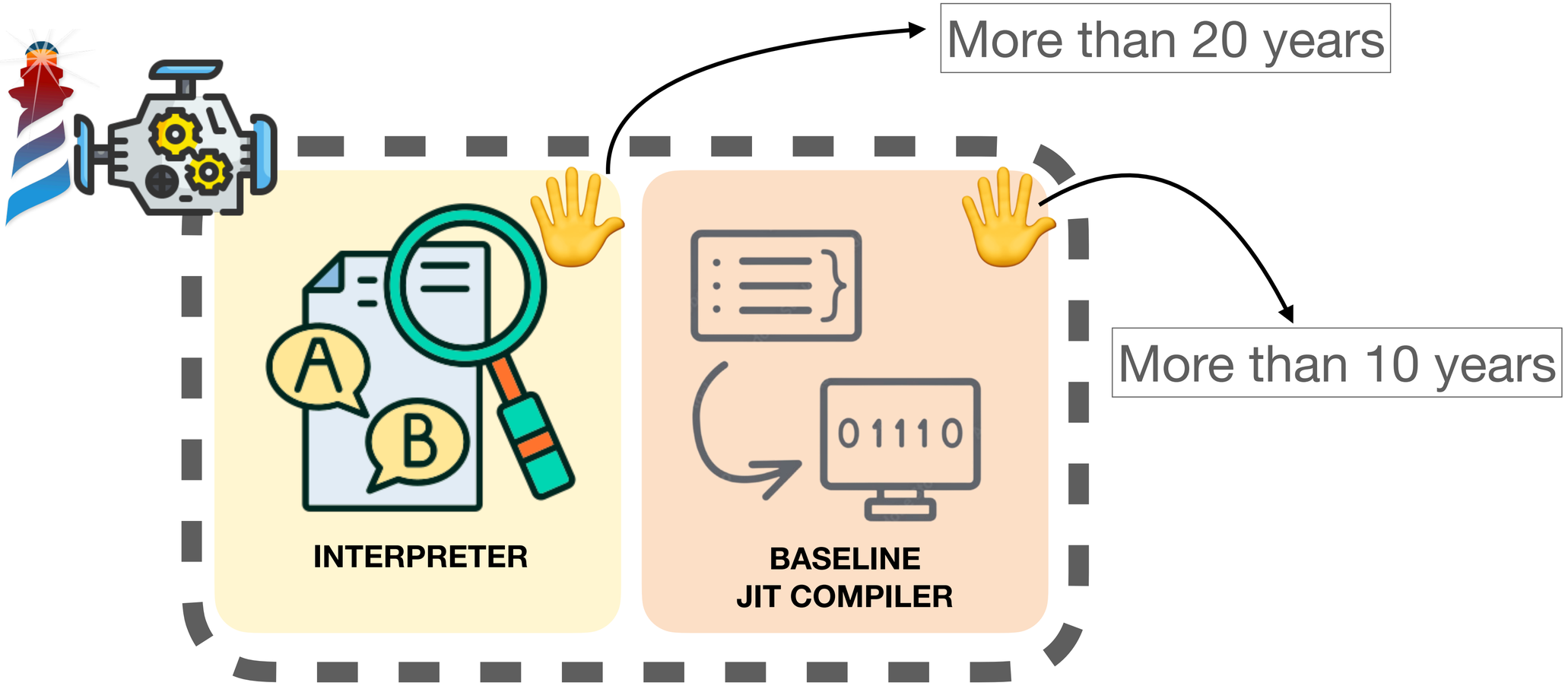


VM Developers

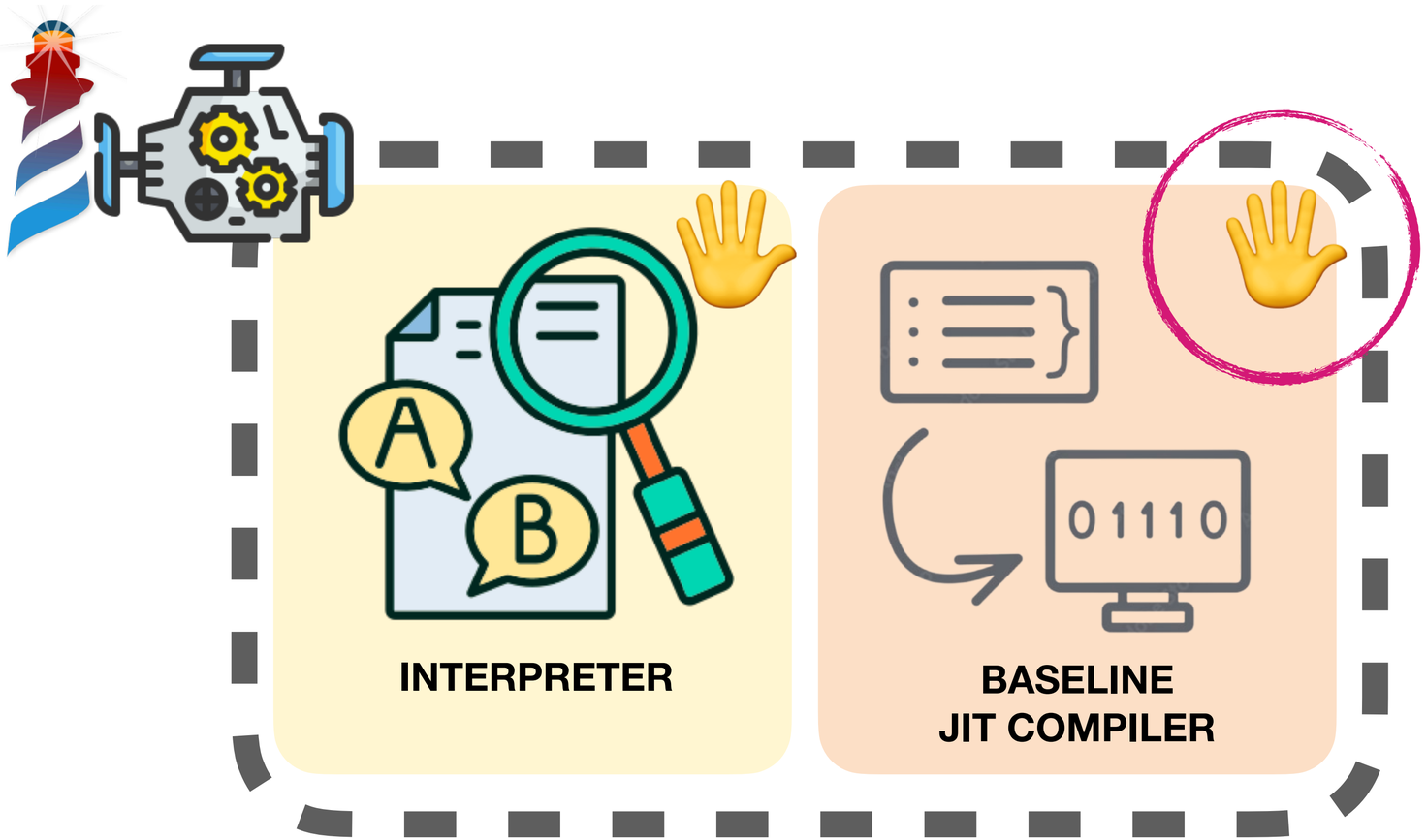
PHARO



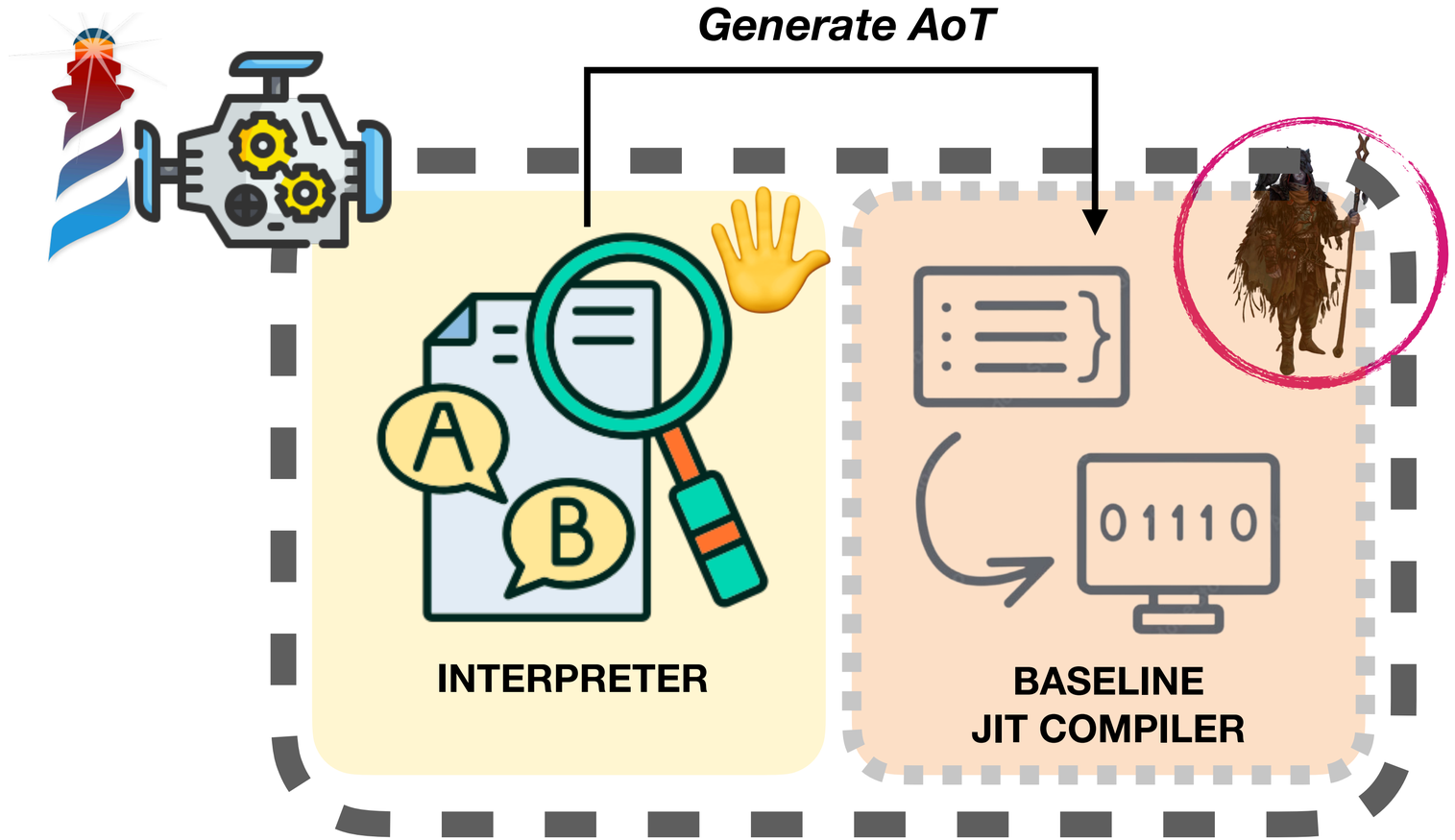
Pharo VM


More than 20 years
More than 10 years
A
B
01110
INTERPRETER
BASELINE
JIT COMPILER

Pharo VM

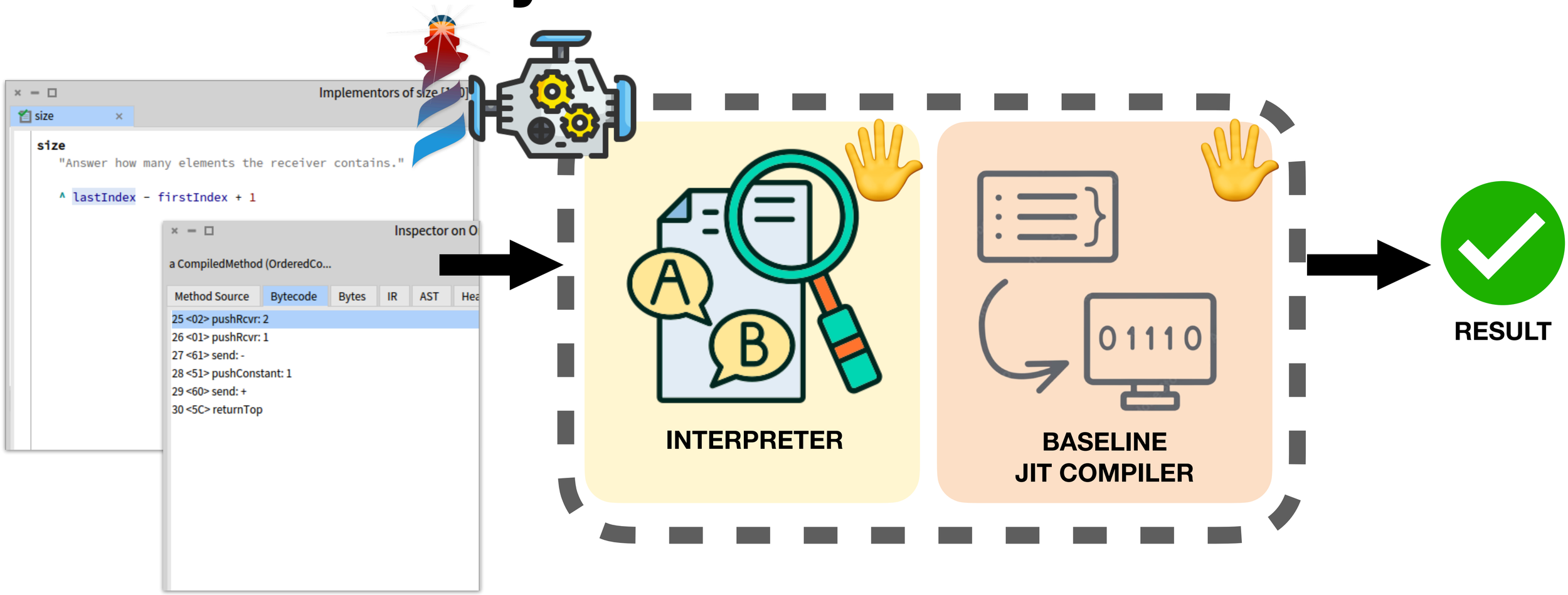


Pharo VM

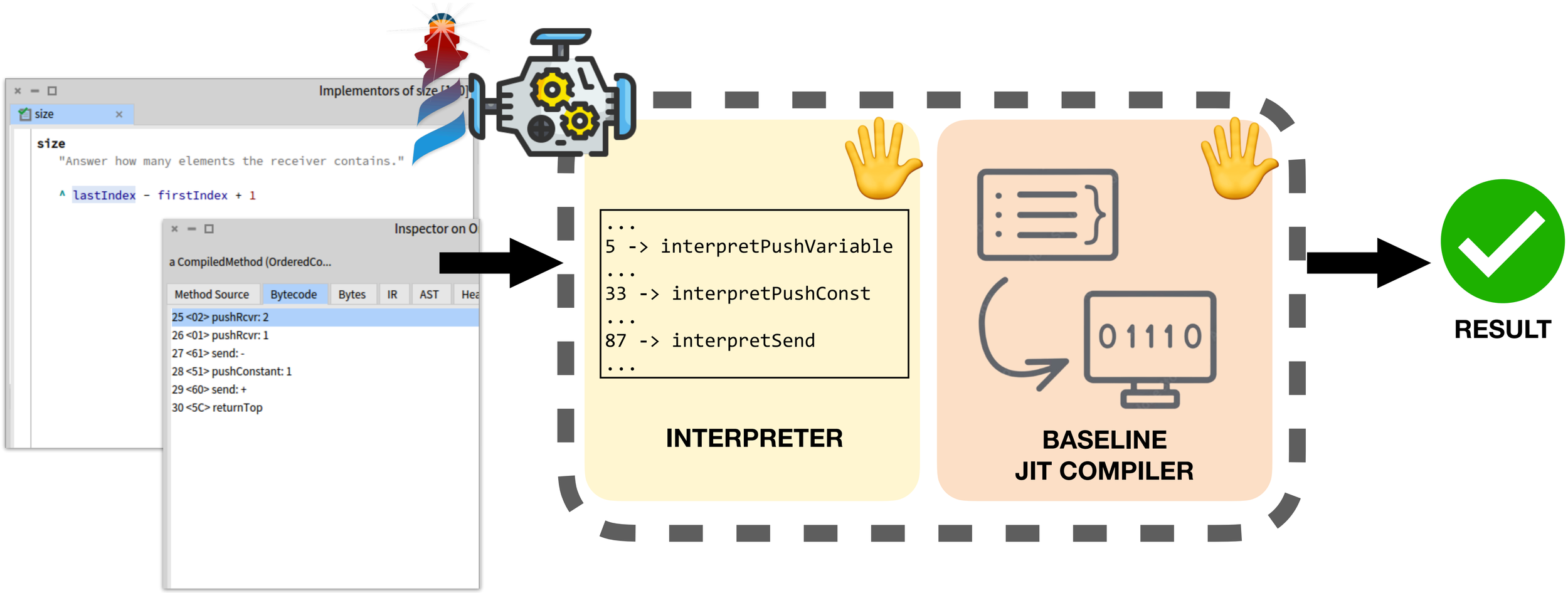


How do we generate Baseline JIT Compilers AoT?

Pharo VM - Bytecodes and Primitives



Pharo VM - Instruction handlers



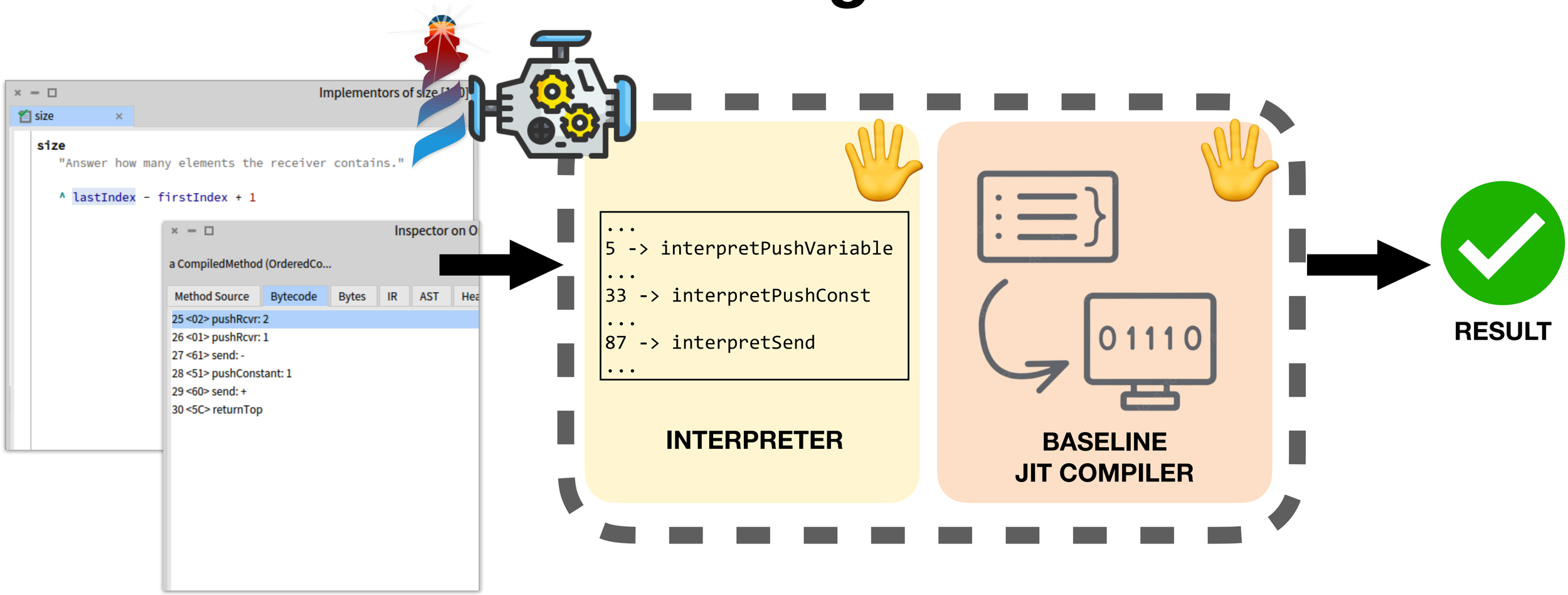
The addition primitive

Interpreter

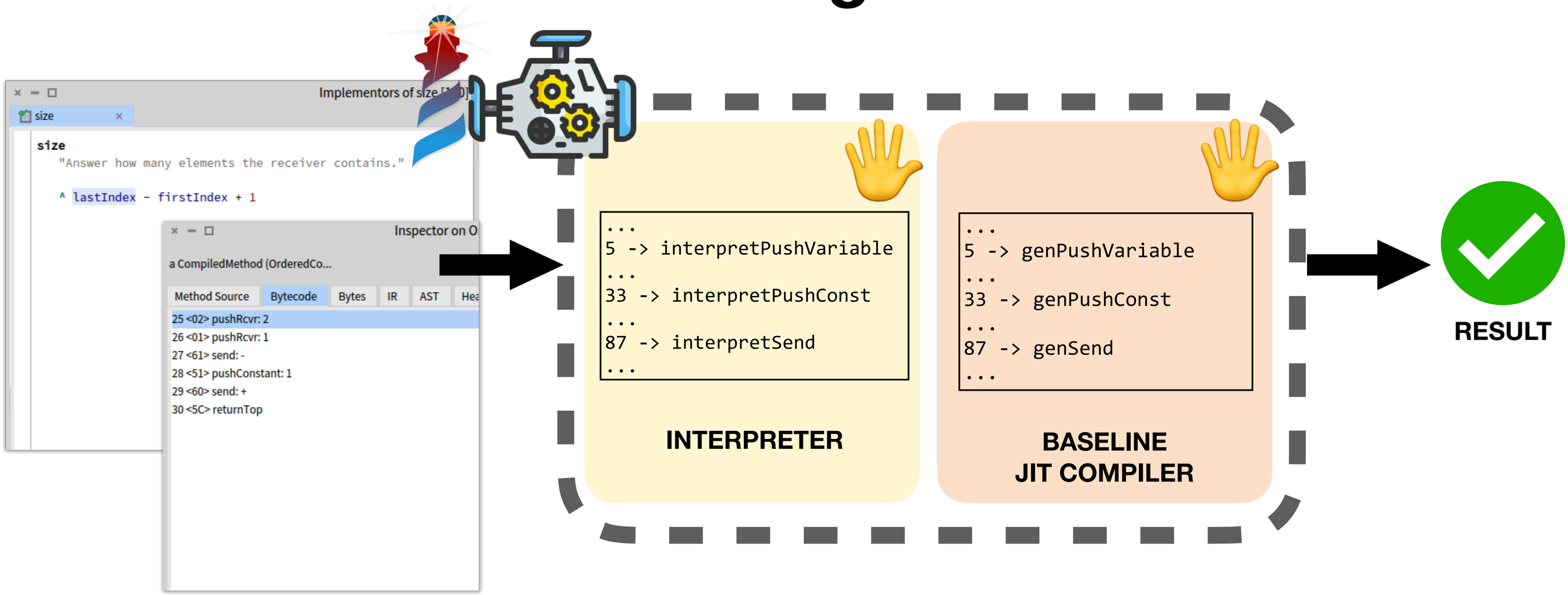
```
1  primitiveAdd
2      <numberOfArguments: 1>
3      <customisedReceiverFor: #smallInteger>
4
5      | maybeSmallInteger maybeSmallInteger2 result |
6
7      maybeSmallInteger := self stackValue: 0.
8      maybeSmallInteger2 := self stackValue: 1.
9
10     (objectMemory isIntegerObject: maybeSmallInteger)
11         ifFalse: [ ^ self primitiveFail ].
12     (objectMemory isIntegerObject: maybeSmallInteger2)
13         ifFalse: [ ^ self primitiveFail ].
14
15     "Check for overflow"
16     result := self
17         sumSmallInteger: maybeSmallInteger
18         withSmallInteger: maybeSmallInteger2
19         ifOverflow: [ ^ self primitiveFail ].
20
21     self pop: 2 thenPush: result
```

- Pops the two integers from the stack
- Push the addition
- On checks failure, a slow path is taken

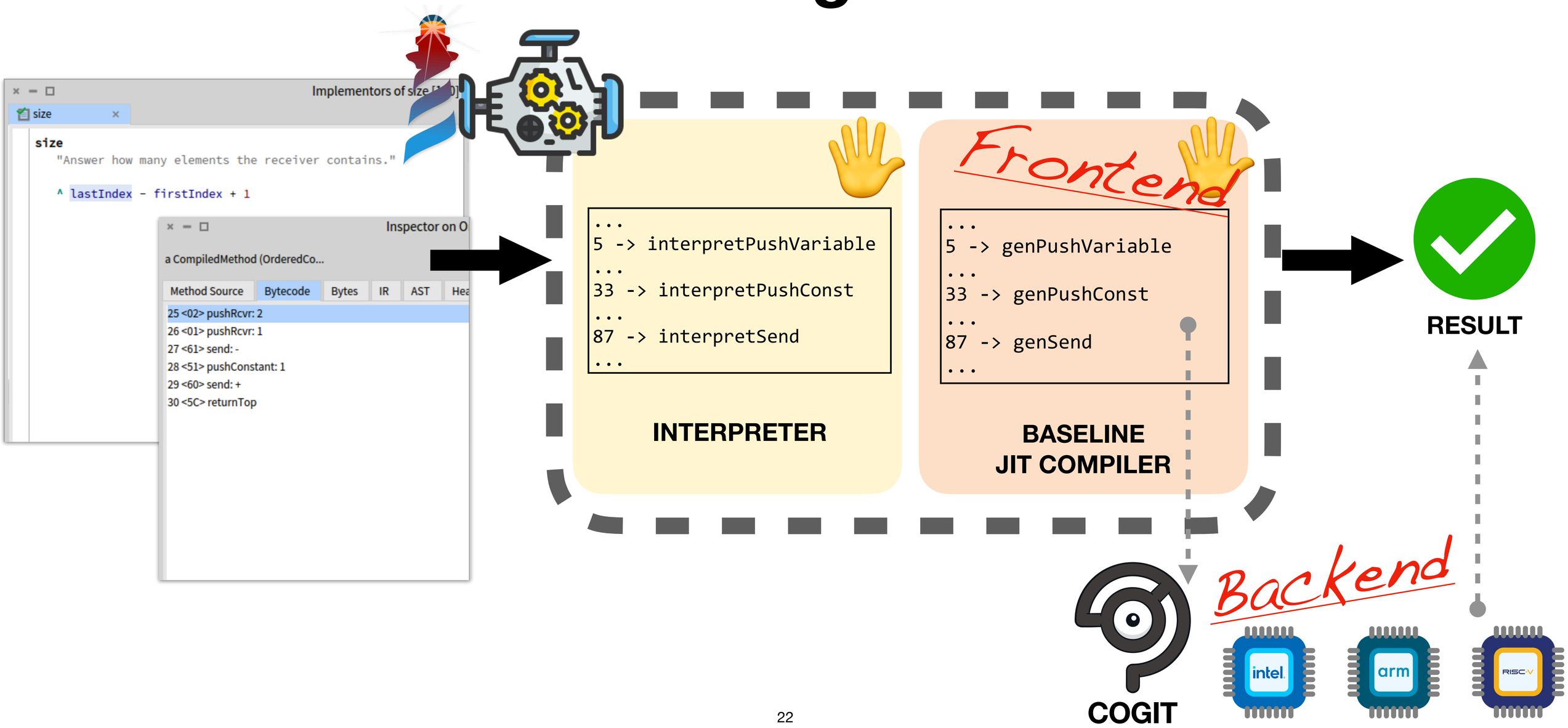
Pharo VM - Instruction *generator* handlers



Pharo VM - Instruction *generator* handlers



Pharo VM - Instruction *generator* handlers



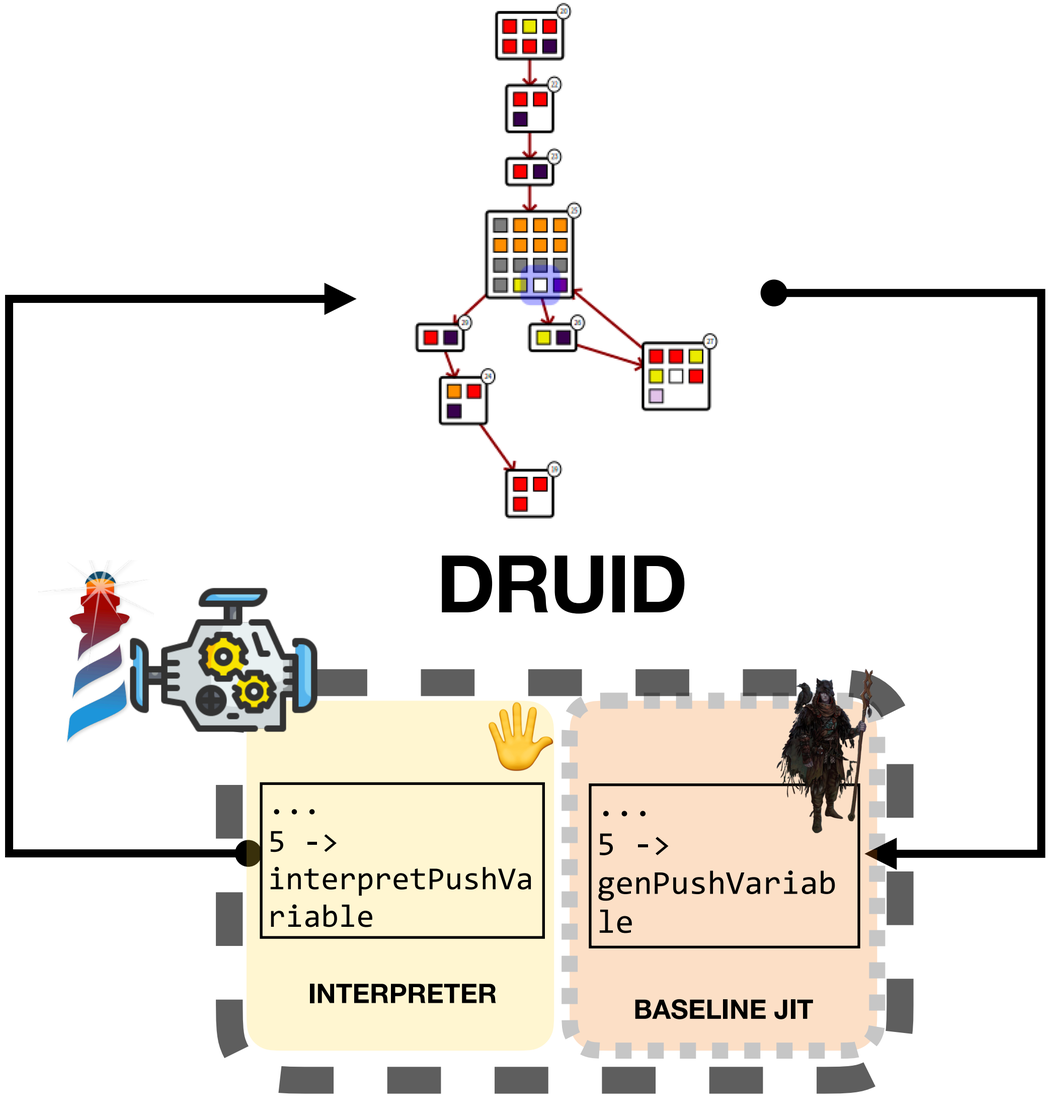
The addition *generator* primitive - Cogit RTL

JIT Compiler

```
1  genPrimitiveAdd
2      | jumpNotSI jumpOvfl |
3      <var: #jumpNotSI type: #'AbstractInstruction *'>
4      <var: #jumpOvfl type: #'AbstractInstruction *'>
5      cogit mclassIsSmallInteger ifFalse:
6          [^UnimplementedPrimitive].
7
8      cogit genLoadArgAtDepth: 0 into: Arg0Reg.
9      cogit MoveR: Arg0Reg R: ClassReg.
10     jumpNotSI := self
11         genJumpNotSmallInteger: Arg0Reg scratchReg: TempReg.
12
13     self genRemoveSmallIntegerTagsInScratchReg: ClassReg.
14     cogit AddR: ReceiverResultReg R: ClassReg.
15     jumpOvfl := cogit JumpOverflow: 0.
16
17     cogit MoveR: ClassReg R: ReceiverResultReg.
18     cogit genPrimReturn.
19
20     jumpOvfl jmpTarget: (jumpNotSI jmpTarget: cogit Label).
21     ^CompletePrimitive
```

- Generate code
- Virtual register-based (2AC)
- Low level machine code

Meta-compile a baseline JIT compiler for Pharo



Code generated by Druid

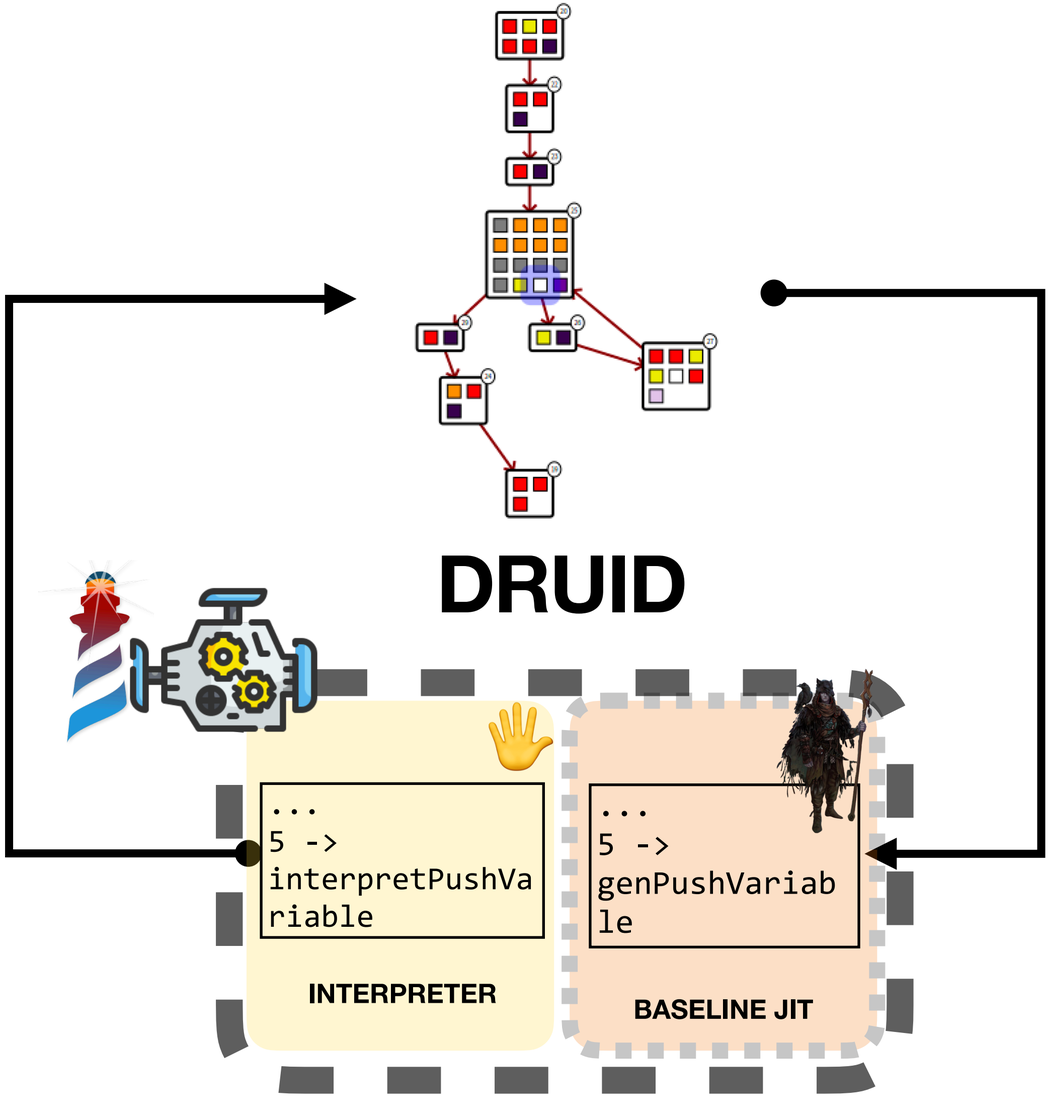
Interpreter

```
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
```

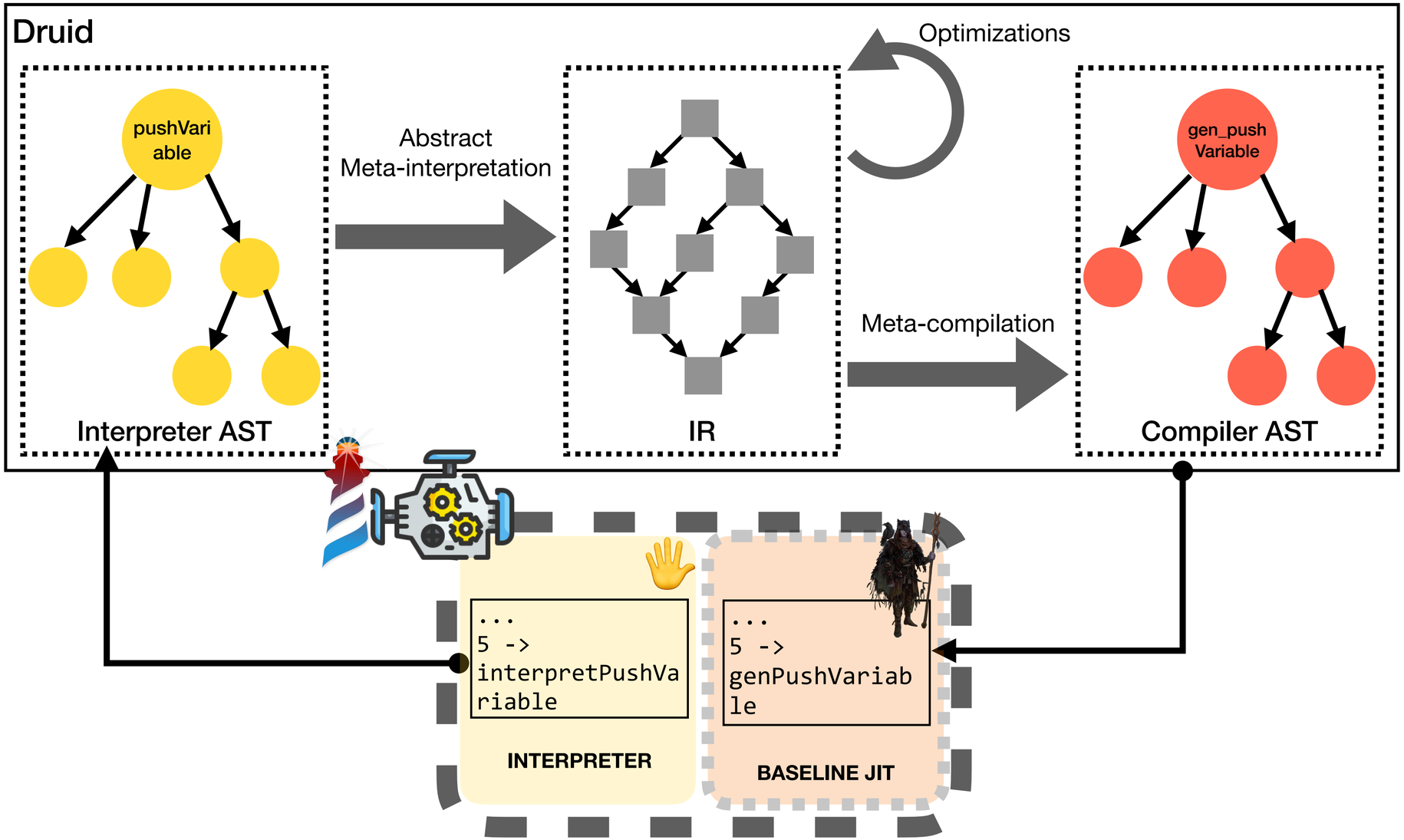
Baseline JIT Compiler

```
gen_PrimitiveAdd
    "AutoGenerated by Druid"

    | jump1 jump2 currentBlock |
    self mclassIsSmallInteger ifFalse: [ ^ UnimplementedPrimitive ].
    self MoveR: ReceiverResultReg R: ClassReg.
    self MoveR: Arg0Reg R: SendNumArgsReg.
    self TstCq: 1 R: SendNumArgsReg.
    jump1 := self JumpZero: 0.
    self AddCq:  -1 R: SendNumArgsReg.
    self AddR: ClassReg R: SendNumArgsReg.
    jump2 := self JumpOverflow: 0.
    self MoveR: SendNumArgsReg R: ReceiverResultReg.
    self genPrimReturn.
    currentBlock := self Label.
    jump1 jmpTarget: currentBlock.
    jump2 jmpTarget: currentBlock.
    ^ CompletePrimitive
```

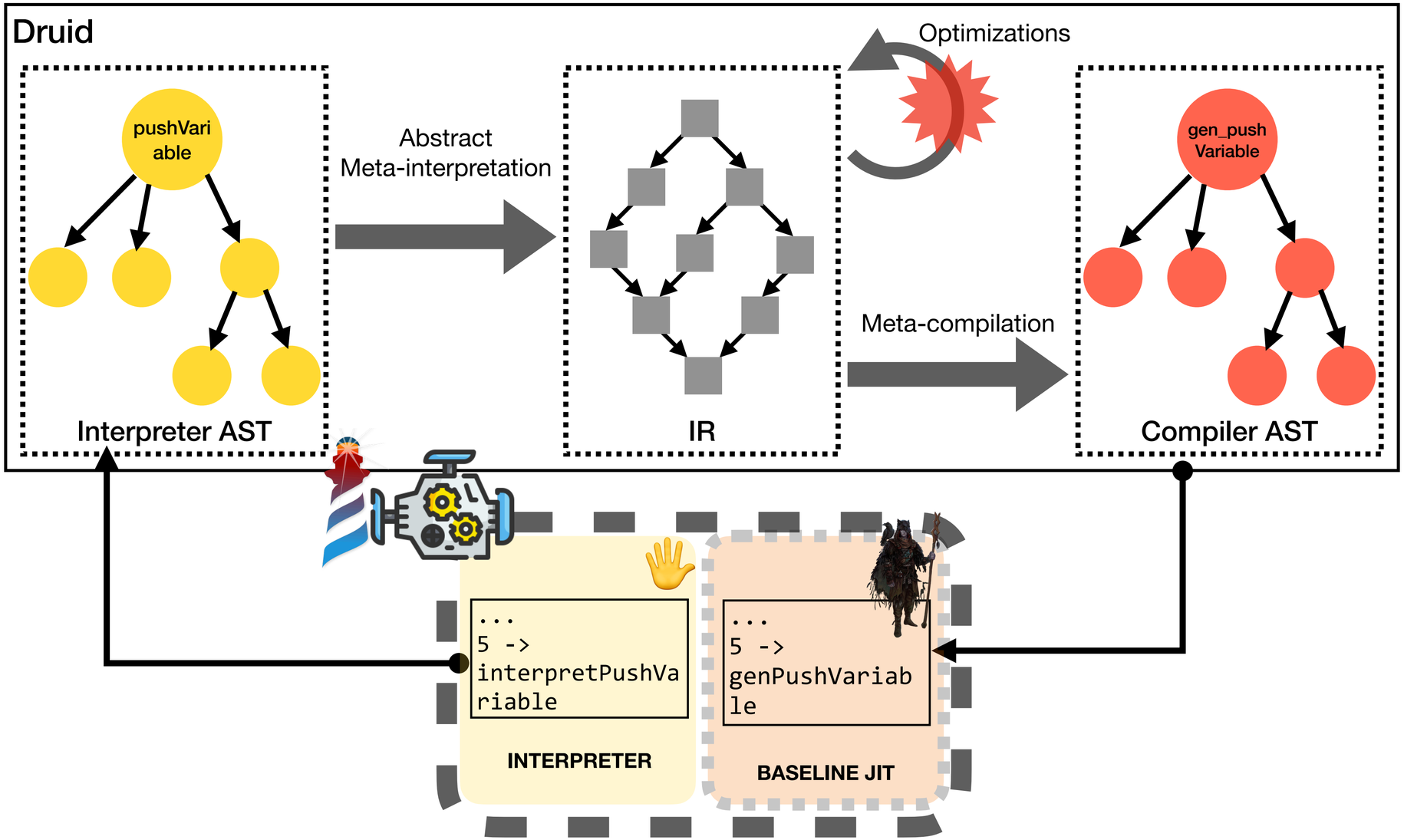
Meta-compile a baseline JIT compiler for Pharo



Meta-compile a baseline JIT compiler for Pharo



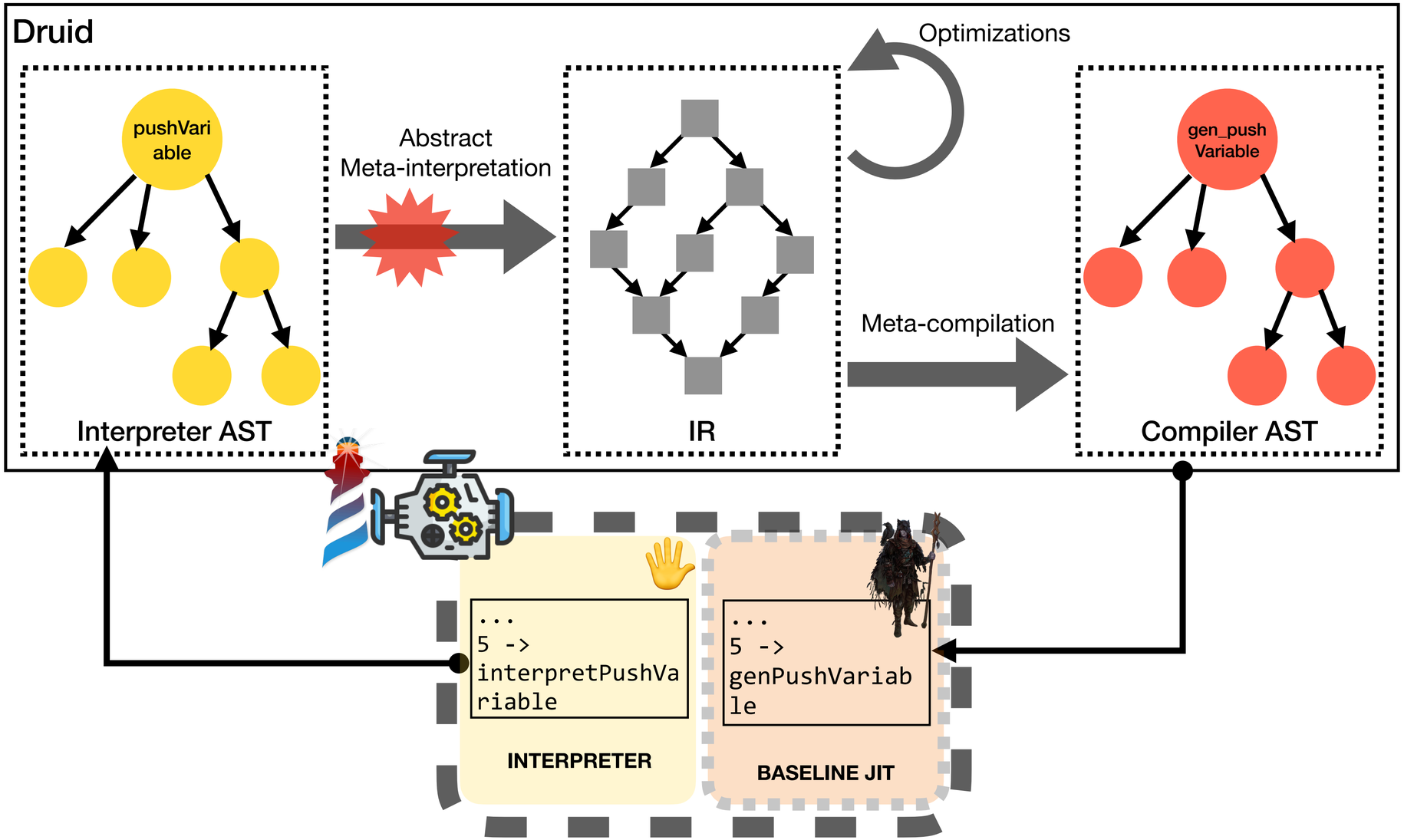
Meta-compile a baseline JIT compiler for Pharo



Druid Optimizations

Name	Description	Open opportunities
BRANCH COLLAPSE	Inline conditions in conditional jumps	Dead path analysis
CLEAN CONTROL FLOW	Merge instructions in consecutive blocks to avoid unnecessary jumps	Better local analysis
COPY PROPAGATION	Replace copy instructions in operands by real value	Better dependency analysis and possible unused code
DEAD BLOCK ELIMINATION	Remove inaccessible blocks	-
DEAD BRANCH ELIMINATION	Remove branches with only dead paths	Better propagations
DEAD CODE ELIMINATION	Remove unused instructions	-
DEAD EDGE SPLITTING	Duplicate blocks with dead and not-dead paths	Create branches with only dead paths
FAILURE CODE TAIL DUPLICATION	Tail duplicate block with resulted code	Divide fail and success paths
PHI SIMPLICATION	Replace one operand phis with copy instruction	Better propagations
REDUNDANT COPY ELIMINATION	Remove copies of form x := x	-
SCCP	Performs constants folding and propagation	Better dependency analysis and possible unused code

Meta-interpretation guided by intrinsics



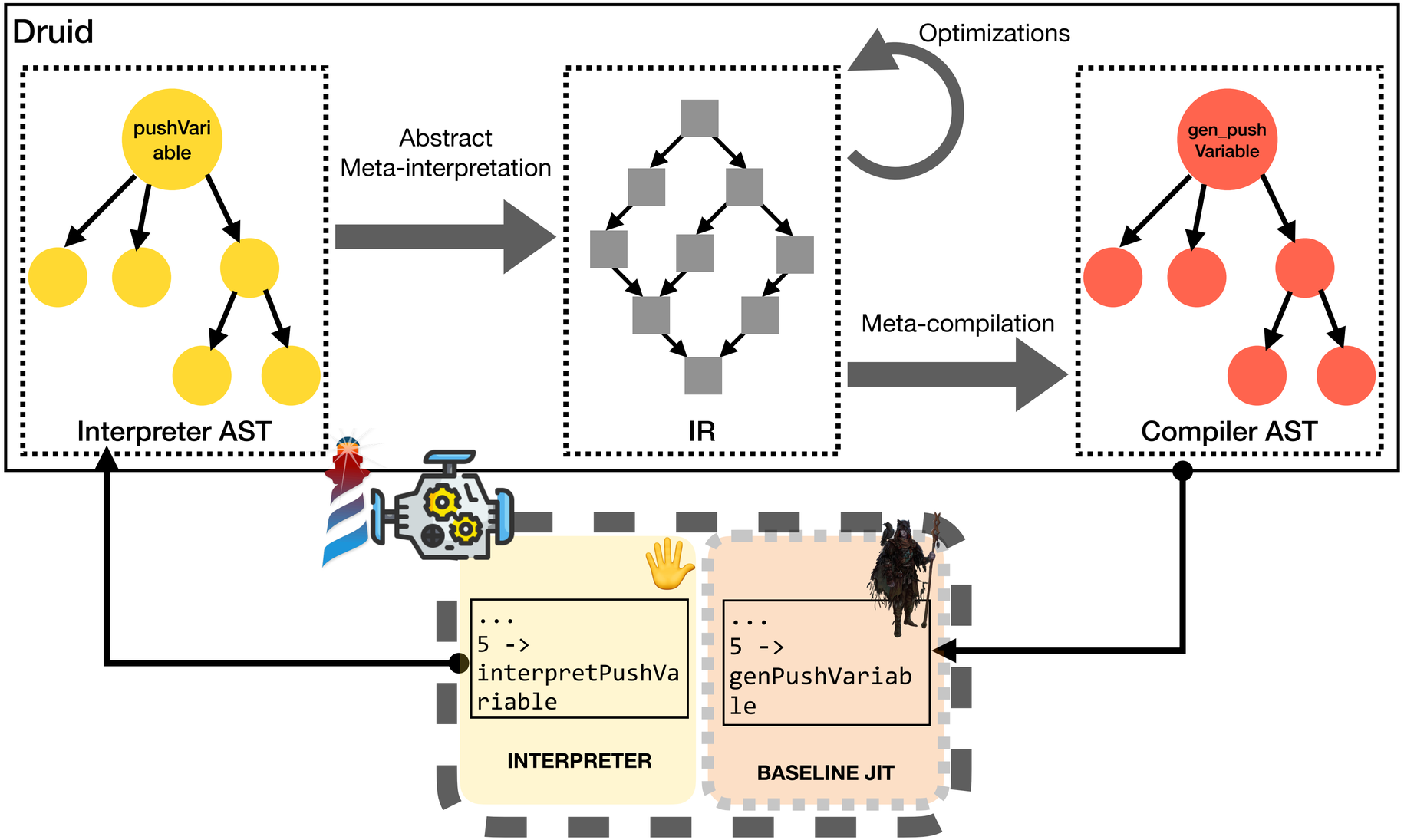
Meta-compilation guided by intrinsics

Interpreter

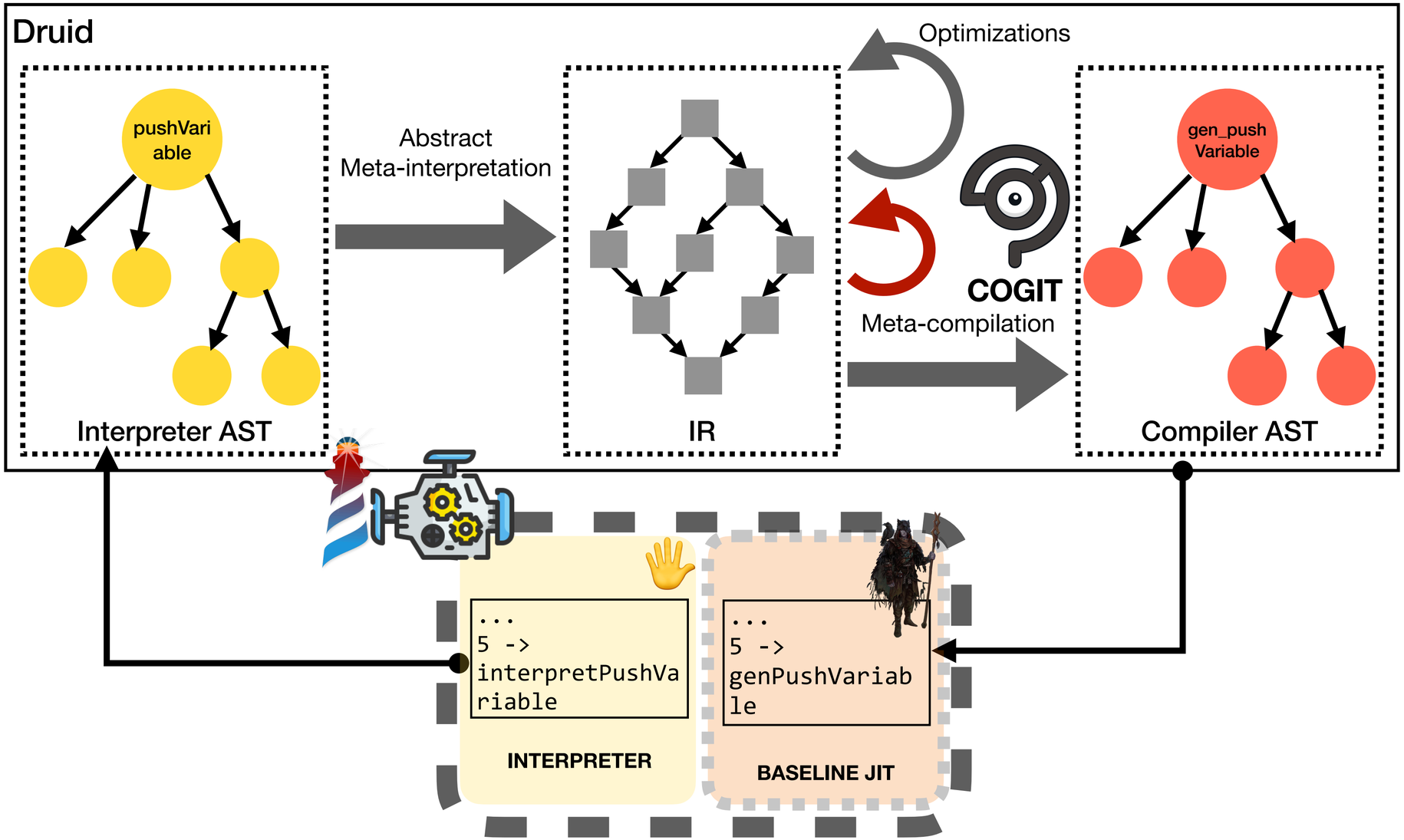
pushMaybeCon ×

```
1  pushMaybeContext: obj receiverVariable: fieldIndex
2      "Must trap accesses to married and widowed contexts.
3       But don't want to check on all inst var accesses.  This
4       method is only used by the long-form bytecodes, evading
5       the cost. Note that the method, closure and receiver fields
6       of married contexts are correctly initialized so they don't
7       need special treatment on read.  Only sender, instruction
8       pointer and stack pointer need to be intercepted on reads."
9
10     <inline: true>
11     ((self isReadMediatedContextInstVarIndex: fieldIndex) and: [
12         objectMemory isContextNonImm: obj ])
13       ifTrue: [
14           self druidForceIntepretation.
15           self druidIgnore: [
16              self push: (self instVar: fieldIndex ofContext: obj) ] ]
17       ifFalse: [
18         self push: (objectMemory fetchPointer: fieldIndex ofObject: obj) ]
```

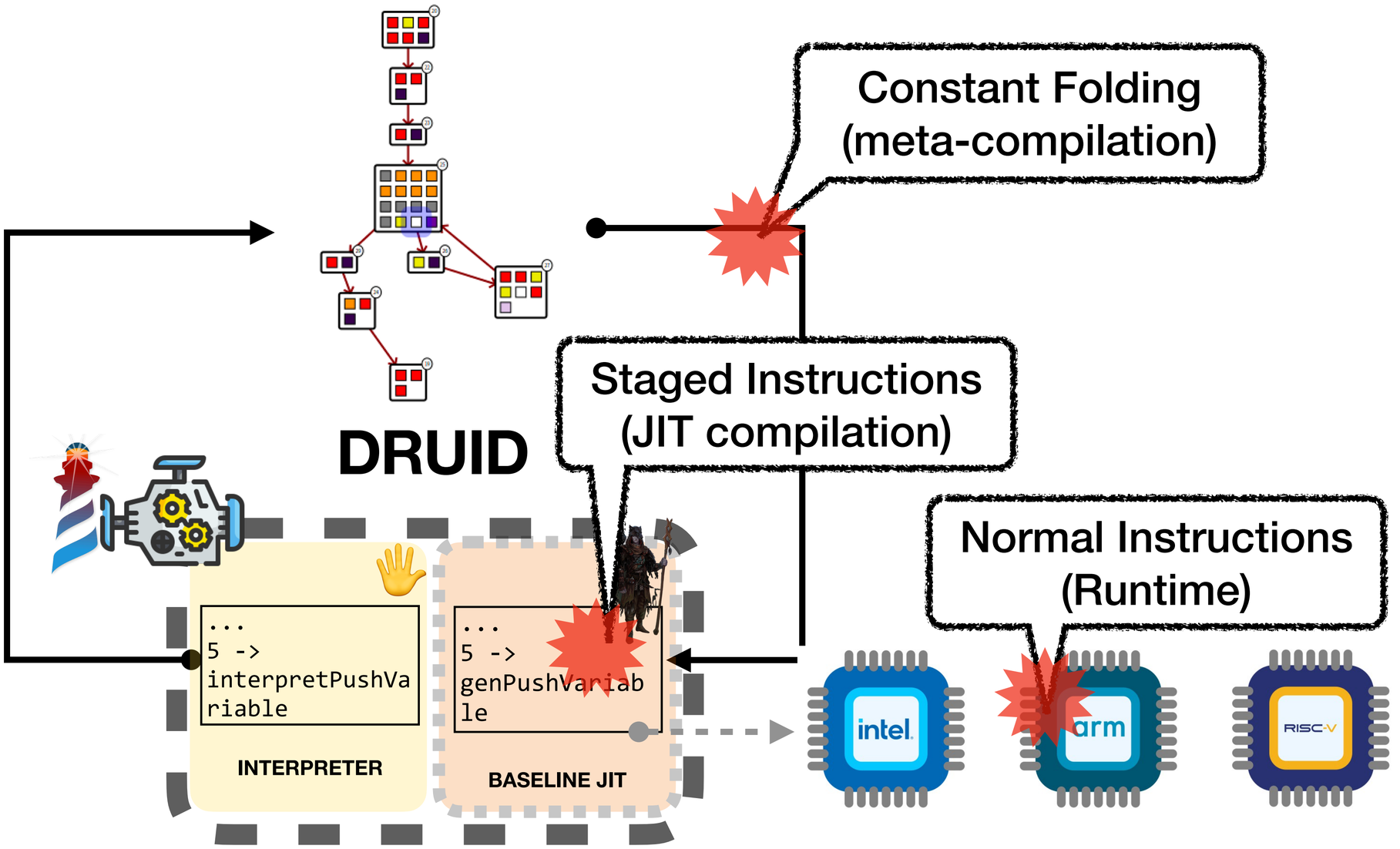
Target Cogit framework



Target Cogit framework



Staging


Constant Folding
(meta-compilation)
Staged Instructions
(JIT compilation)
Normal Instructions
(Runtime)
DRUID
...
5 ->
interpretPushVa
riable
INTERPRETER
...
5 ->
genPushVariab
le
BASELINE JIT
intel
arm
RISC-V

Evaluation

Druid JIT vs Interpreter

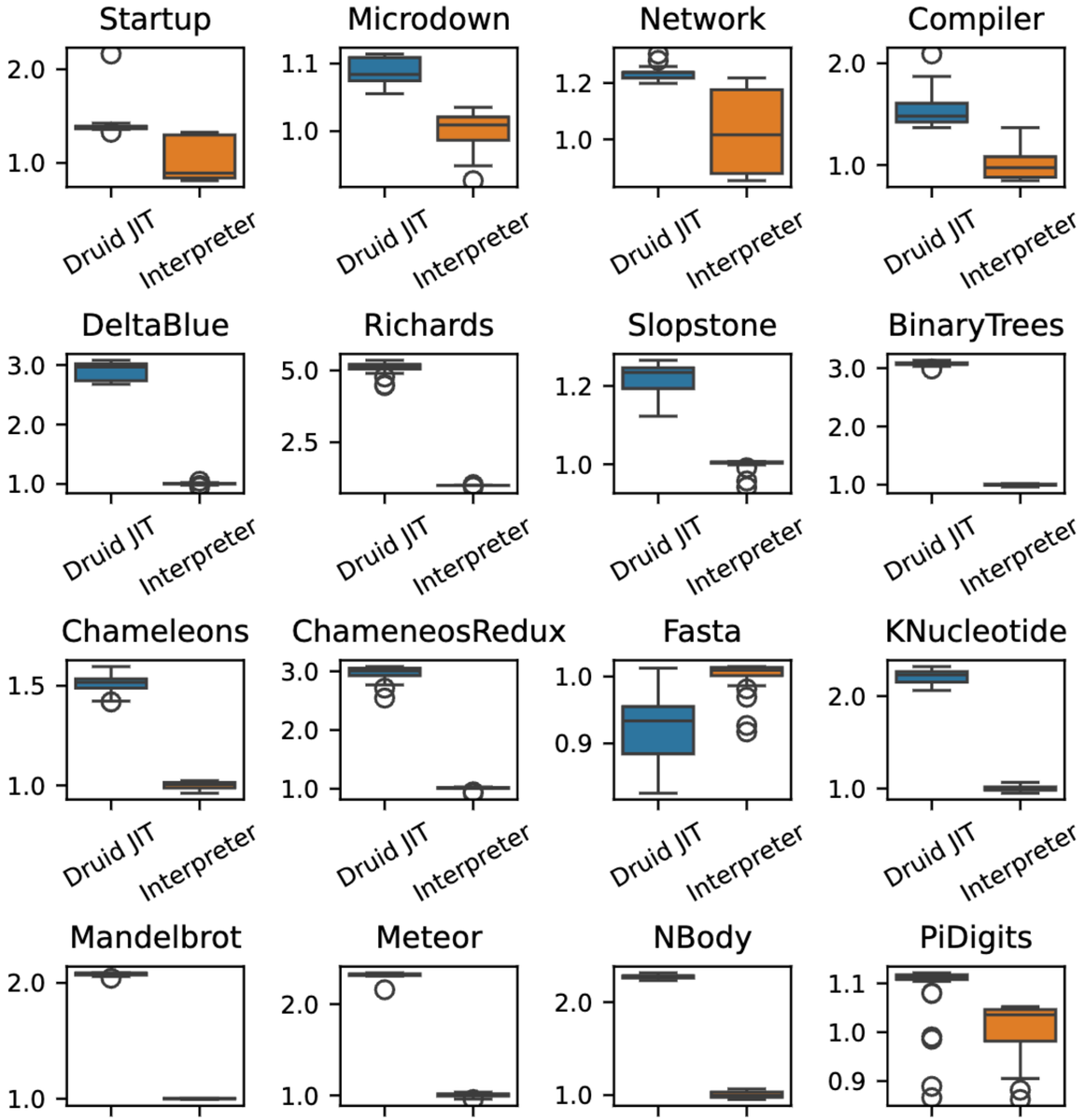


Figure 9 Speed up of the Pharo VM using the JIT compiler generated automatically by our approach (*DruidJITVM*) compared to *InterpreterOnly* (baseline). Higher is better.

Druid JIT vs Interpreter

Faster than the Interpreter

- **2x** on average
- Up to **5x**

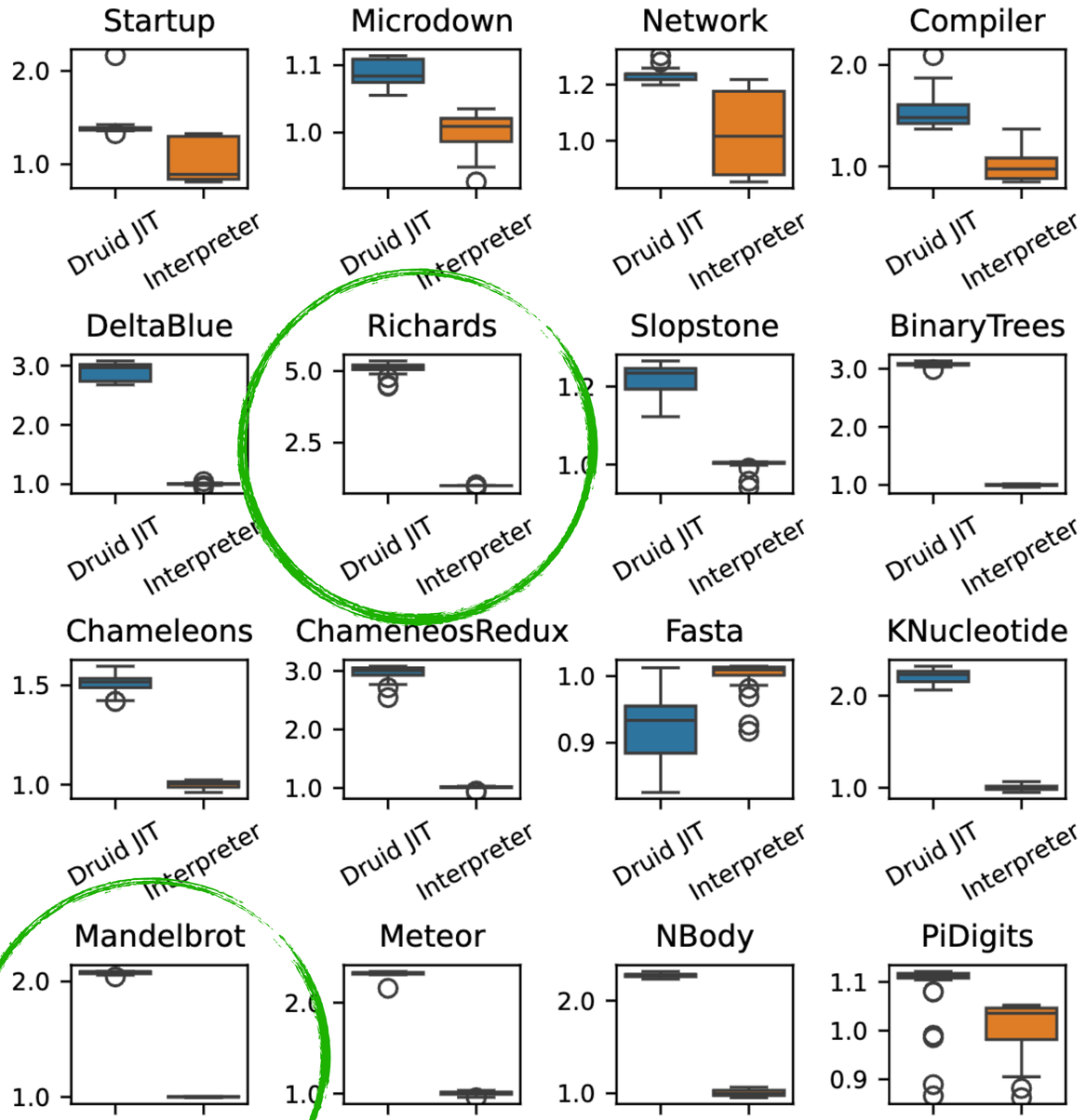


Figure 9 Speed up of the Pharo VM using the JIT compiler generated automatically by our approach (*DruidJITVM*) compared to *InterpreterOnly* (baseline). Higher is better.

Druid JIT vs Interpreter

Faster than the Interpreter

- **2x** on average
- Up to **5x**

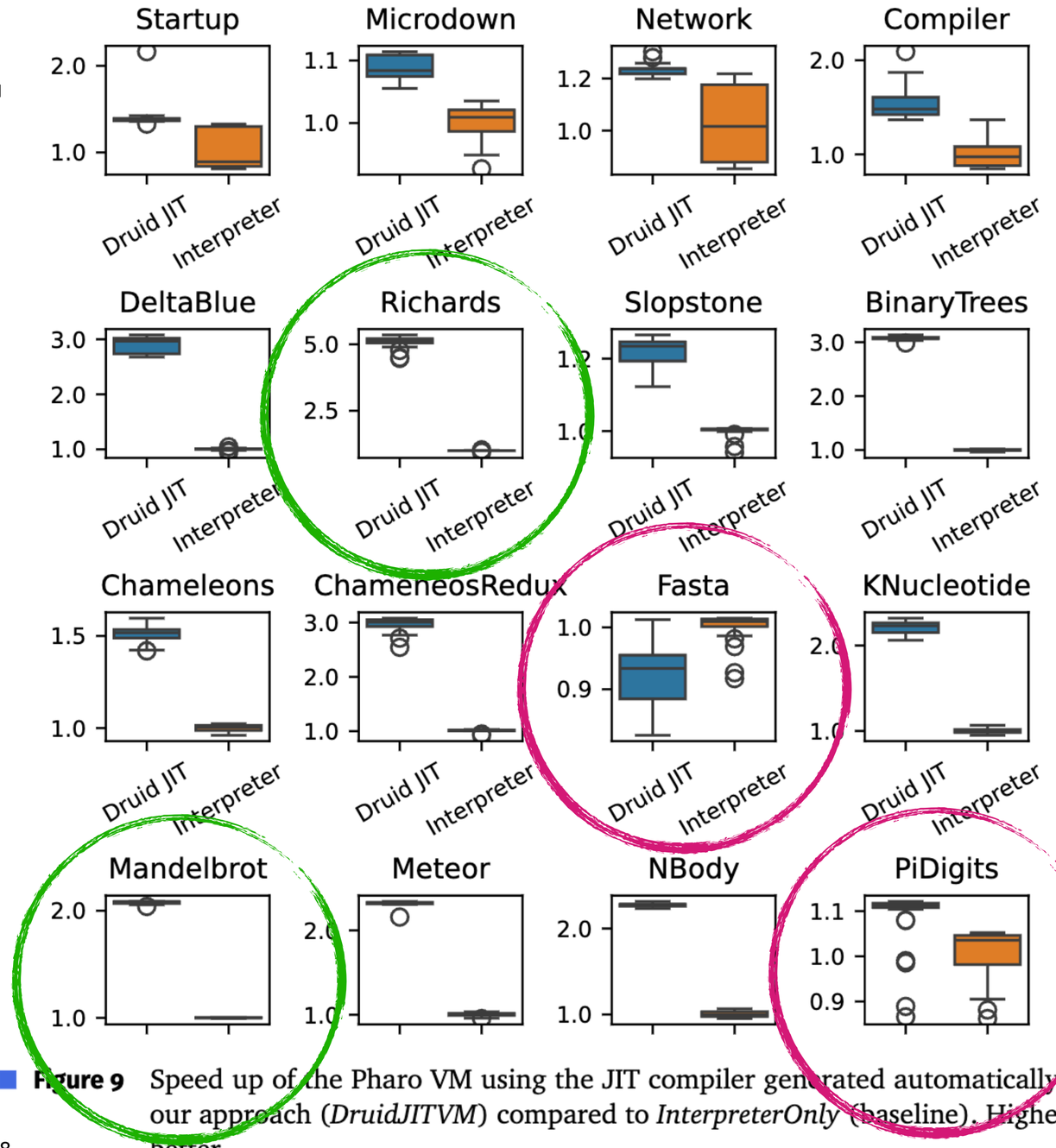


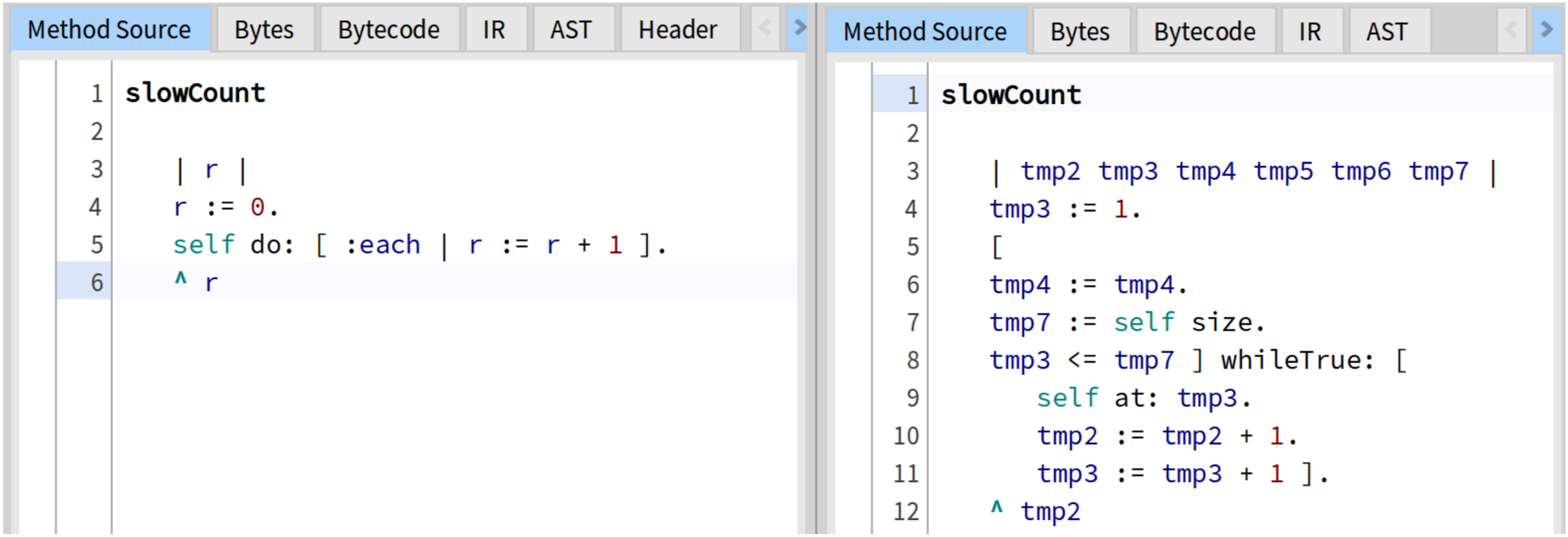
Figure 9 Speed up of the Pharo VM using the JIT compiler generated automatically by our approach (*DruidJITVM*) compared to *InterpreterOnly* (baseline). Higher is better.

What else?

Extra!

We are also using Druid as an

Optimizing Bytecode Compiler (WIP)



Optimizing Bytecode Compiler

With Inlinings

Method Source | Bytes | Bytecode | IR | AST | Header

```
1  slowCount
2
3      | r |
4      r := 0.
5      self do: [ :each | r := r + 1 ].
6      ^ r
```

Method Source | Bytes | Bytecode | IR | AST

```
1  slowCount
2
3      | tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 |
4      tmp3 := 1.
5      [
6      tmp4 := tmp4.
7      tmp7 := self size.
8      tmp3 <= tmp7 ] whileTrue: [
9          self at: tmp3.
10         tmp2 := tmp2 + 1.
11         tmp3 := tmp3 + 1 ].
12     ^ tmp2
```

Optimizing Bytecode Compiler

With Inlinings

Why is this important?

https://github.com/ESUG/esug.github.io/blob/source/2024-Conference/slides/show-us-your-project/show-us-your-project-esug-2024-dropal.pdf

```
4    r := 0.
5    self do: [ :each | r := r + 1 ].
6    ^ r
```

Method Source | Bytes | Bytecode | IR | AST

```
1  slowCount
2
3      | tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 |
4      tmp3 := 1.
5      [
6      tmp4 := tmp4.
7      tmp7 := self size.
8      tmp3 <= tmp7 ] whileTrue: [
9          self at: tmp3.
10         tmp2 := tmp2 + 1.
11         tmp3 := tmp3 + 1 ].
12     ^ tmp2
```

Conclusions

- AoT meta-compilation of Baseline JITs is feasible
- It achieves decent performance
  - Traditional compiler optimizations
  - Intrinsics + Staging

- Extra!
  - Optimizing Bytecode Compiler

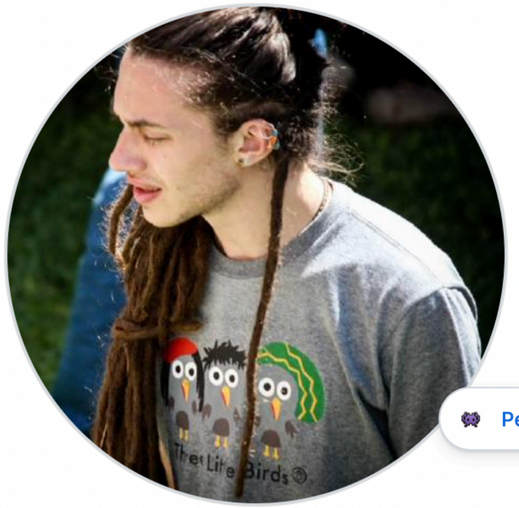
nahuel.palumbo@inria.fr

/Alamvic/druid

/pharo-project/pharo-vm

Let's keep in contact!

https://discord.gg/QewZMZa



Nahuel Palumbo
PalumboN

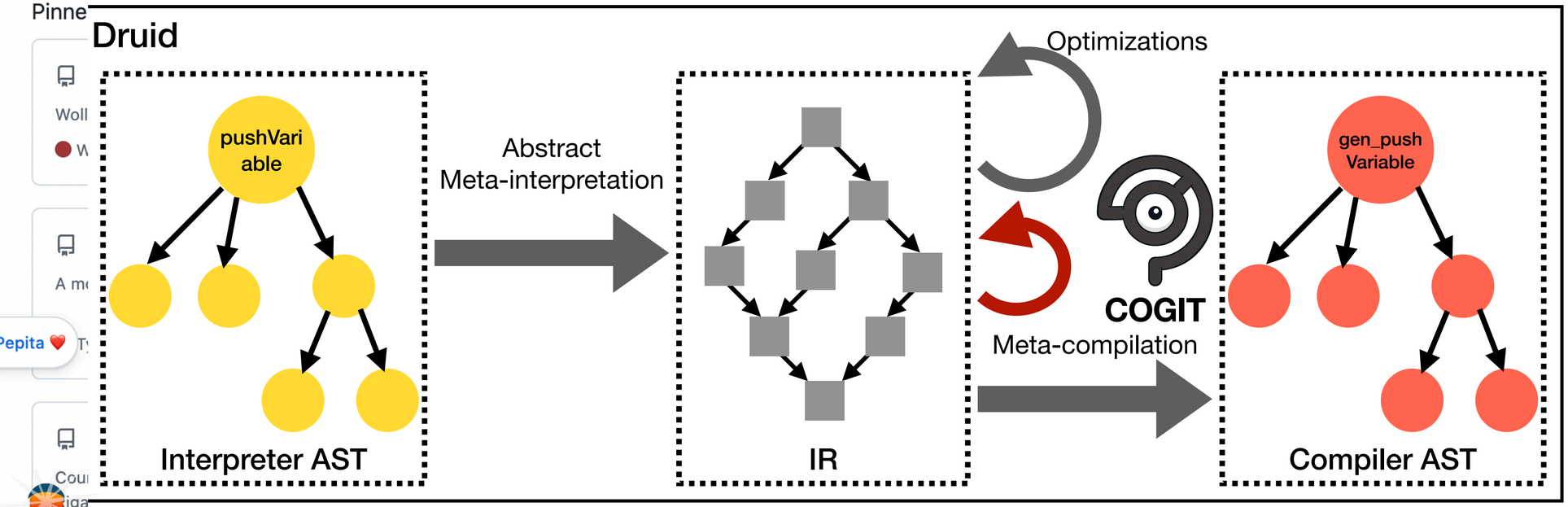
nahuel.palumbo@inria.fr

/Alamvic/druid

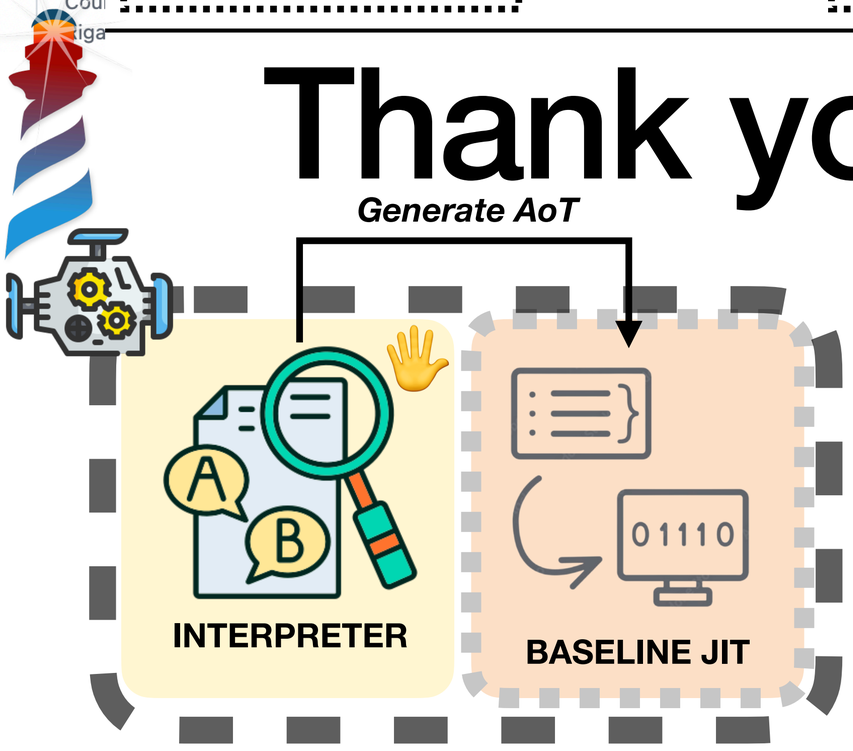
/pharo-project/pharo-vm

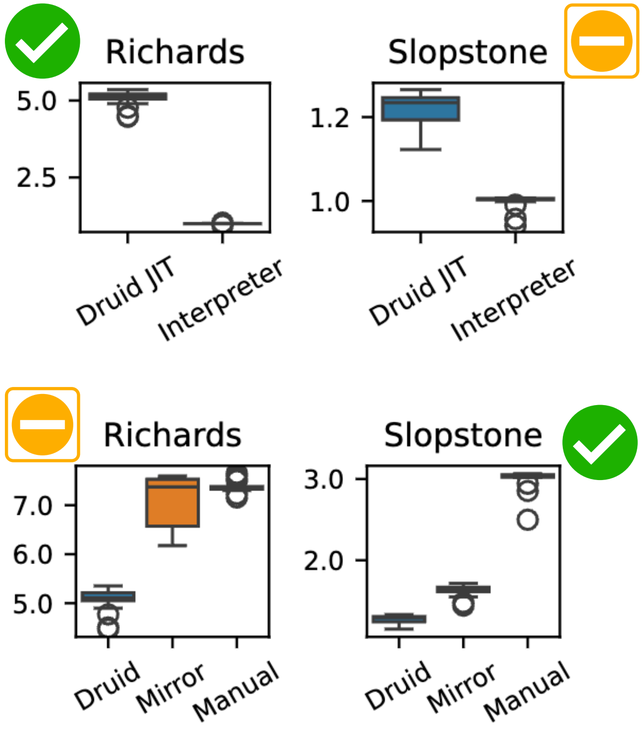
Let's keep in contact!

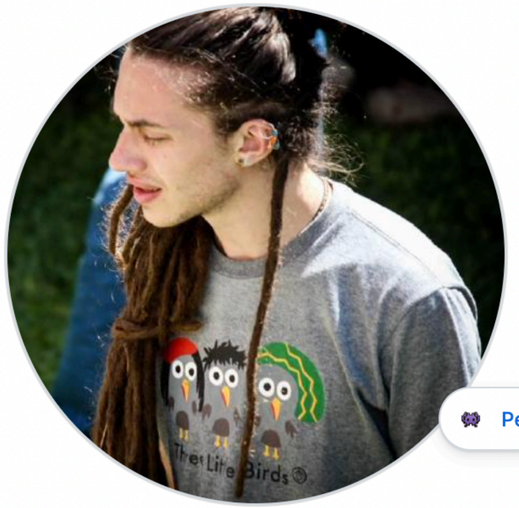
https://discord.gg/QewZMZa



Thank you!







Nahuel Palumbo
PalumboN

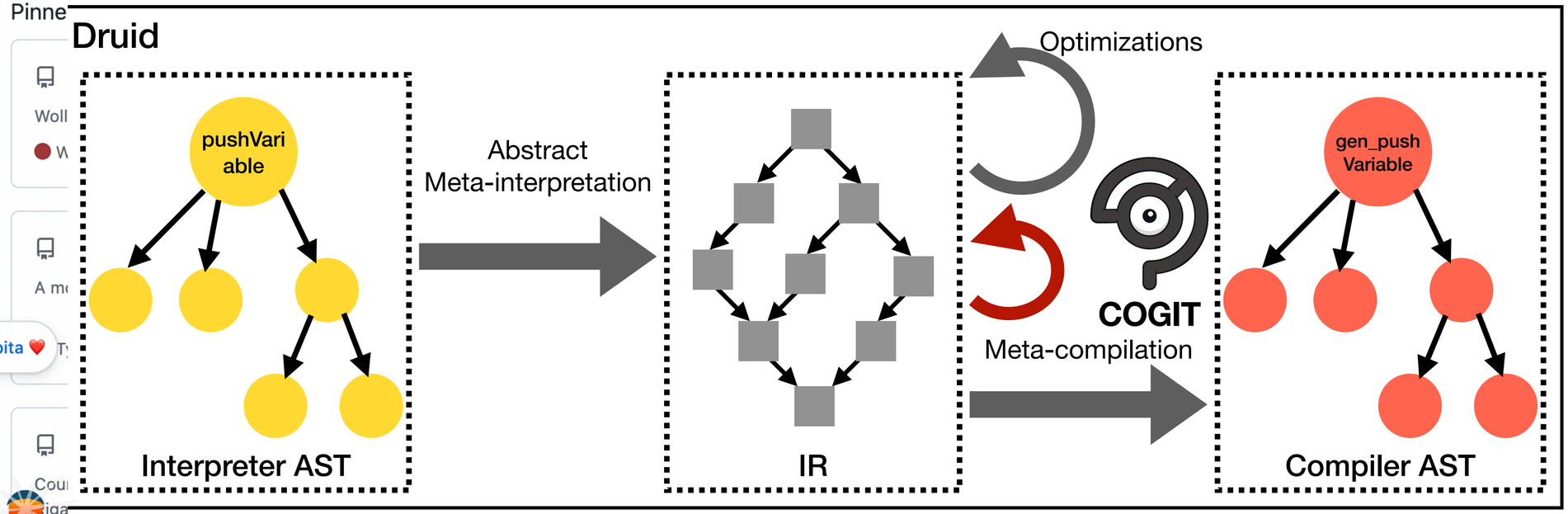
nahuel.palumbo@inria.fr

/Alamvic/druid

/pharo-project/pharo-vm

Let's keep in contact!

https://discord.gg/QewZMZa



Questions?