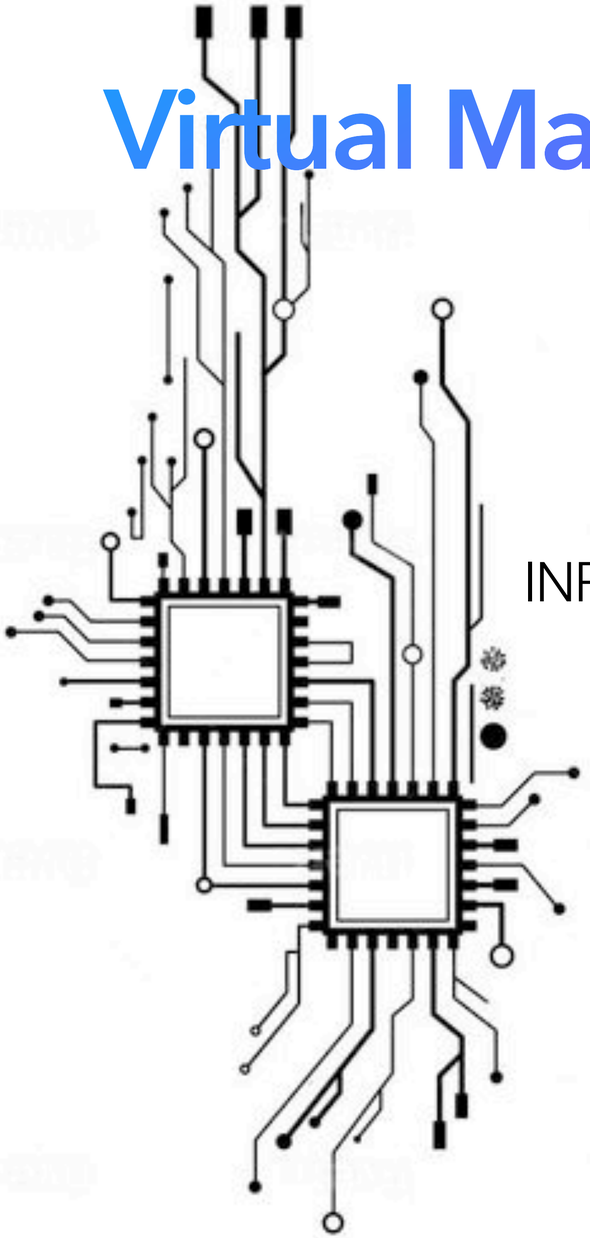


Virtual Machines and Programming Languages

Sebastian JORDAN MONTAÑO

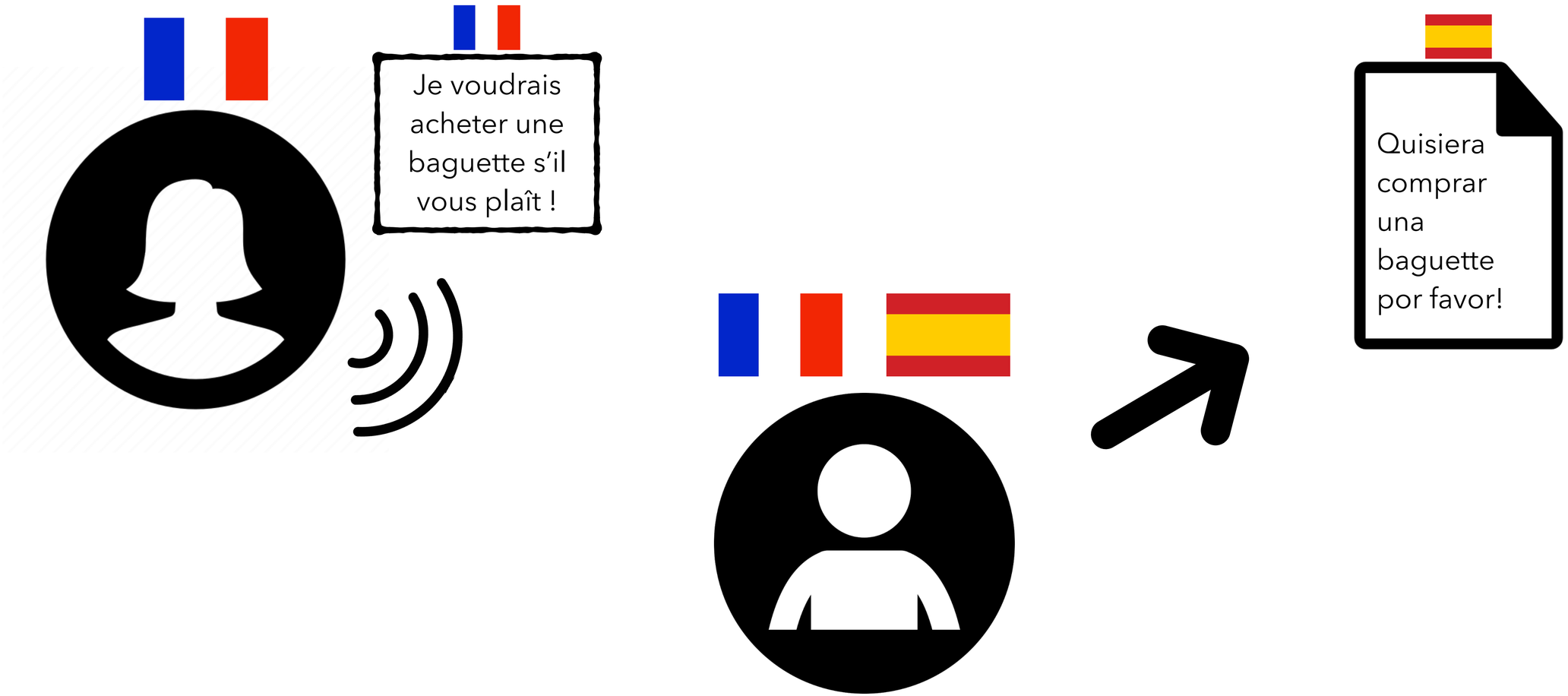
sebastian.jordan@inria.fr

INRIA, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL

December 2022

Compiled vs Interpreted Languages

A Translator

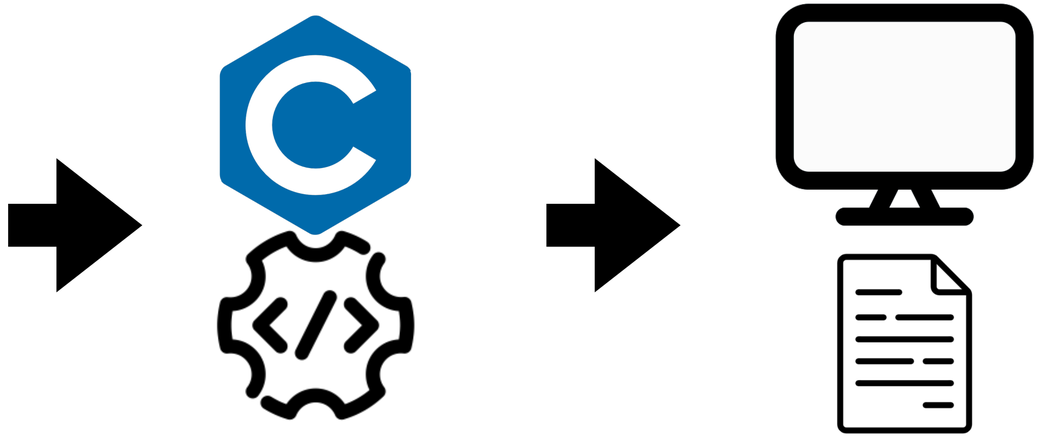


Compiled Language: C

```
Bureau — -bash — 90×35
[LeRoiCrimson:Desktop sebastian$ gcc -o hello_world hello_world.c
[LeRoiCrimson:Desktop sebastian$ ./hello_world
  Hello World!
 LeRoiCrimson:Desktop sebastian$
```

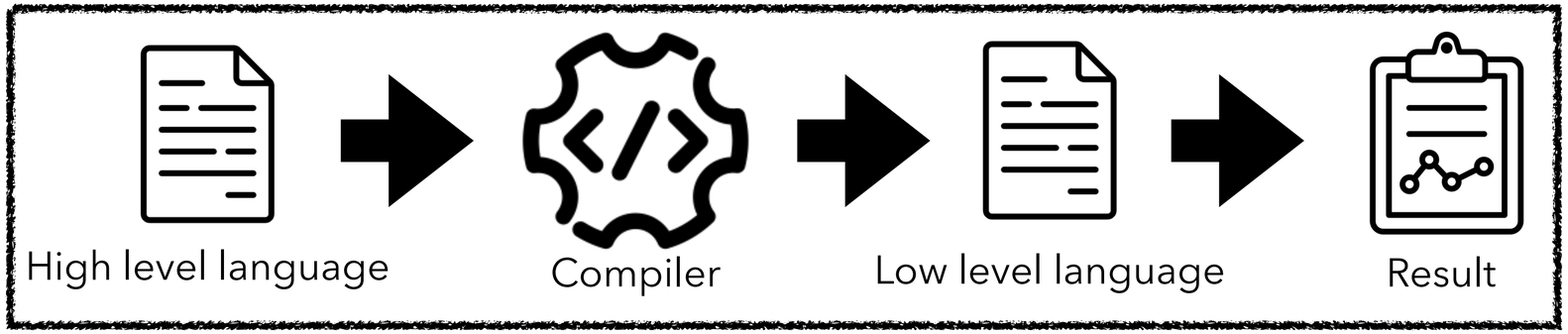
```c
#include <stdio.h>

int main() {
    printf(" Hello World!\n");
    return 0;
}
```

hello_world.c



hello_world.o

The Compiler



An Interpreter

Interpreted Languages: Pharo

Playground

Do it | Publish | Bindings | Versions | Pages

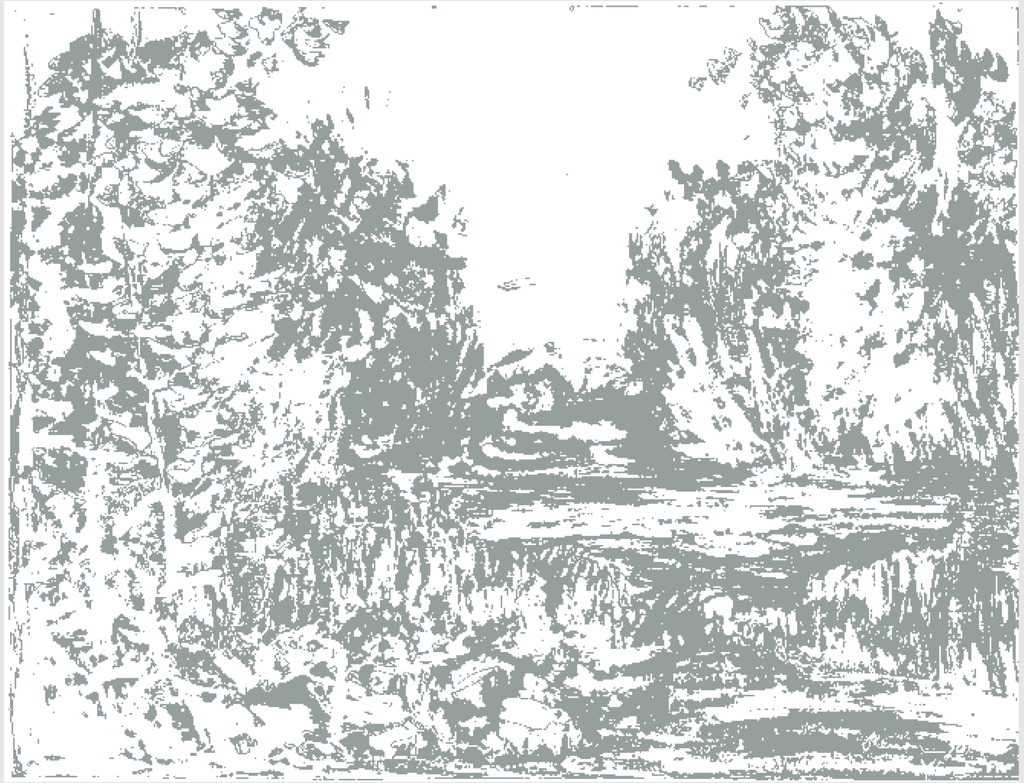
```
1  file :='pharo-local/iceberg/pharo-ai/image-segmentation/' ,
   'img/renoir_river.jpg' asFileReference.
2
3  segmentator := AIImageSegmentator new
4    loadImage: file;
5    numberOfSegments: 3;
6    yourself.
7  segmentator clusterImagePixels.
8  segmentator segmentate.
9  segmentator segments first
```

Line: 9:27 +L

Inspector on Form(669x512x32)

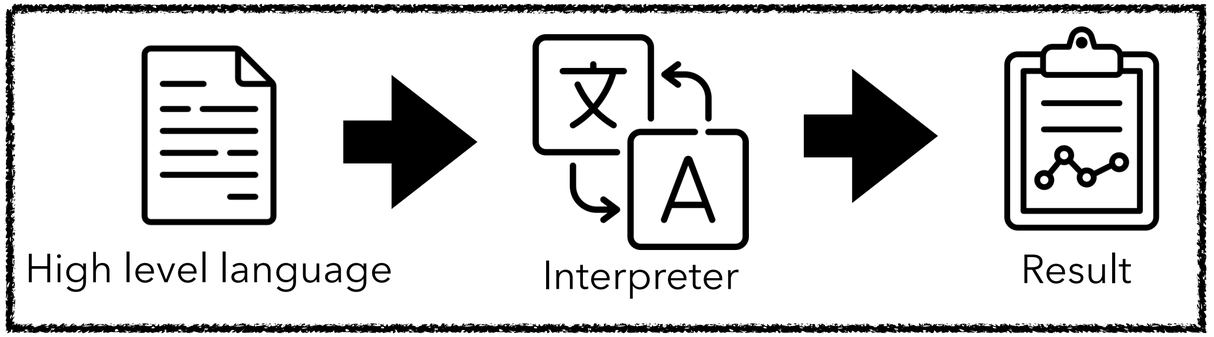
a Form (Form(669x512x32))

Form | Raw | Breakpoints | Meta



```
1  self class "Form"
```

The Interpreter



and how does that work?

A Piece of Code

to compile or to interpret

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

Returning operator

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

A Literal

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

A message (identifier of a method)

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

Opening parenthesis

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

A Variable

```
^ 4 + (a * 5)
```

Identifying the Tokens

Lexical Analysis

A message

^ 4 + (a * 5)

Identifying the Tokens

Lexical Analysis

A Literal

^ 4 + (a * 5)

Identifying the Tokens

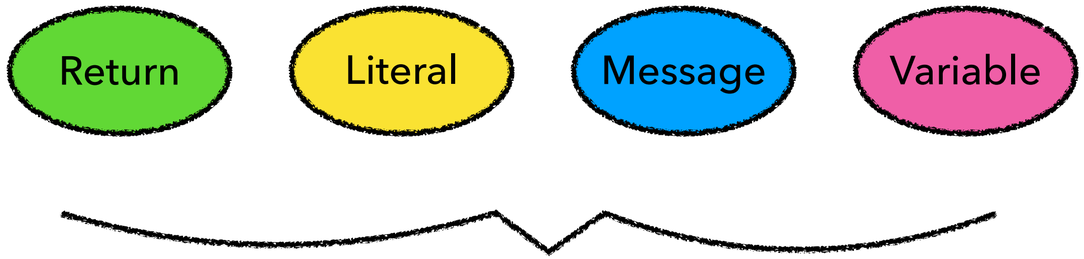
Lexical Analysis

Closing parenthesis

^ 4 + (a * 5)

Identifying the Tokens

(Lexical Analysis)

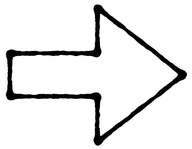


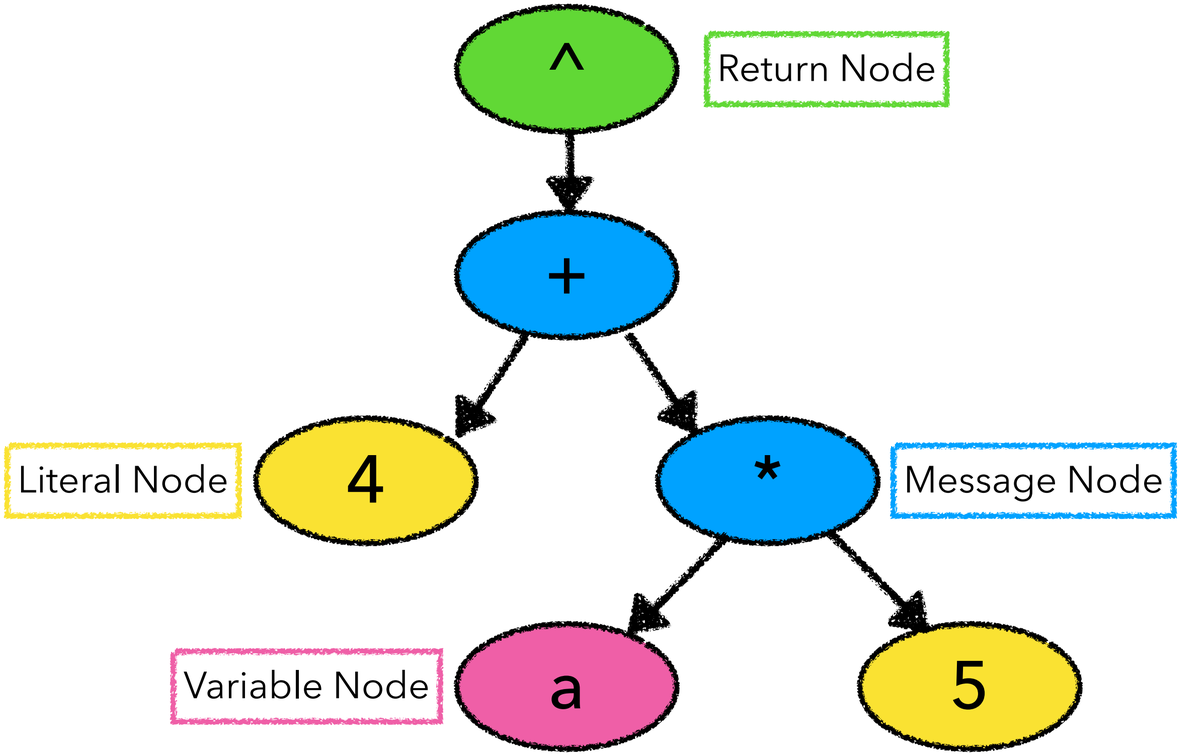
List of Tokens

Parsing

Syntax Analysis

```
^ 4 + (a * 5)
```





The Parser

Lexical Analysis (Tokenisation)

+

Syntax Analysis (Parsing)

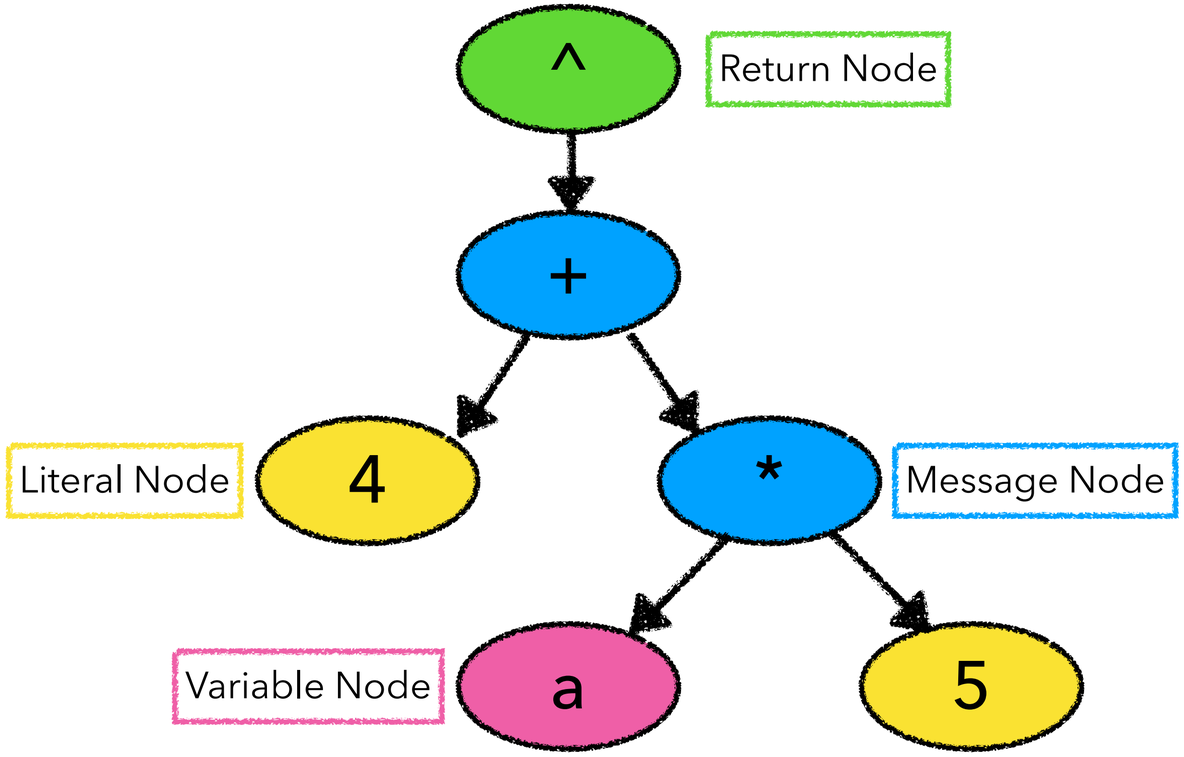
=

The Parser

The Parser

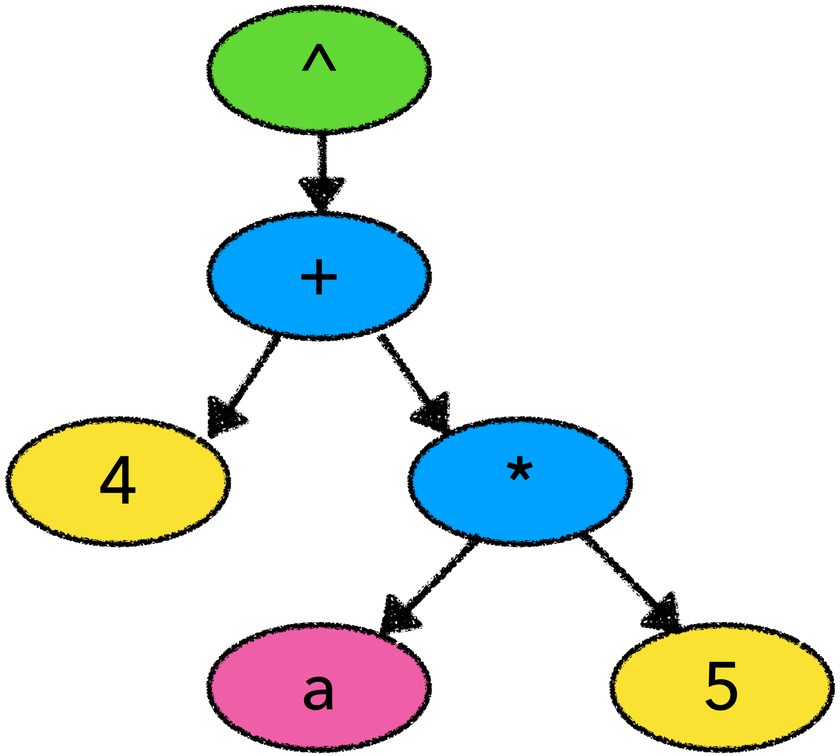
The Parser

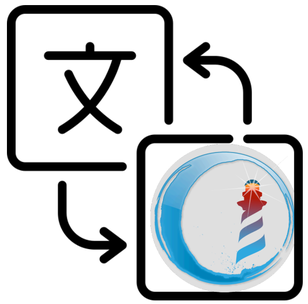
Returns



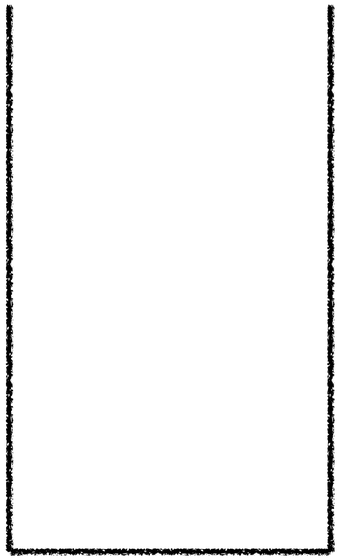
An AST

Interpreting the AST



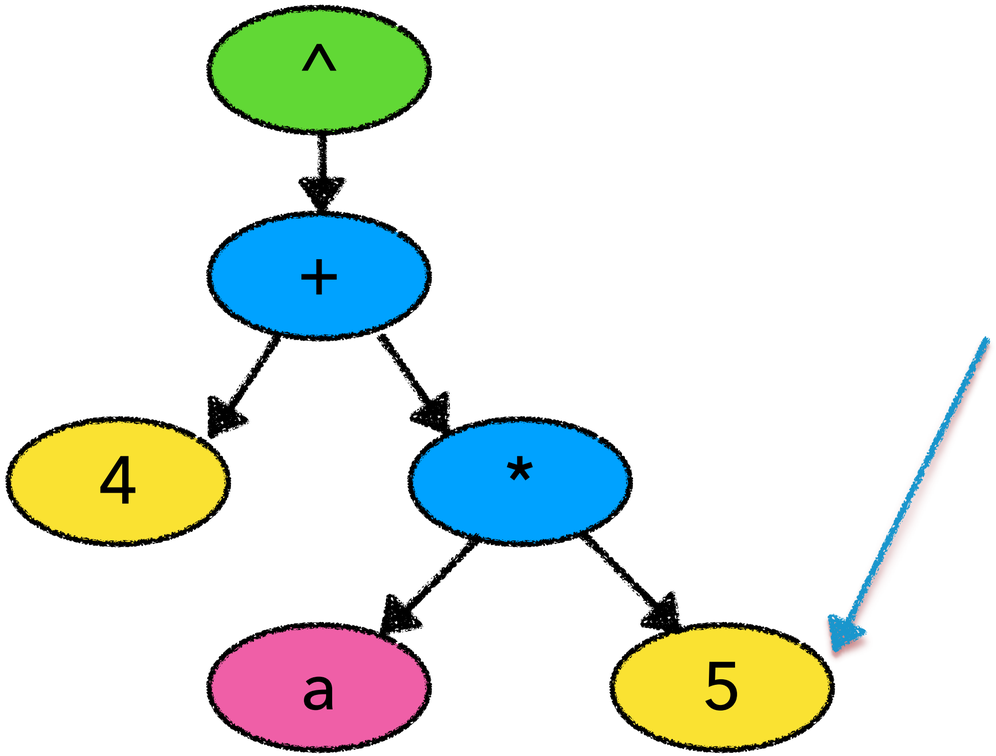


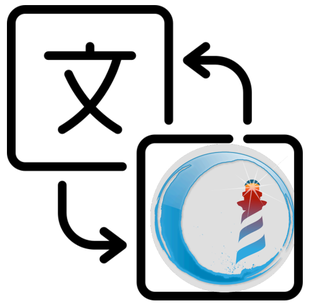
Pharo Interpreter



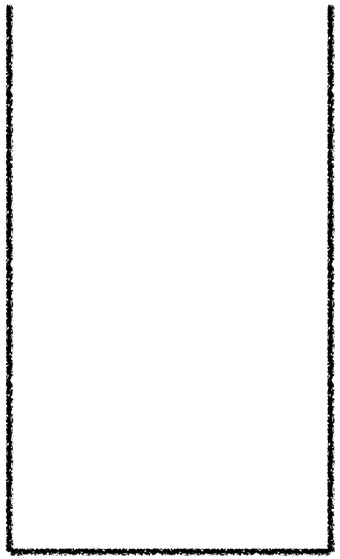
Execution Stack

Interpreting the AST



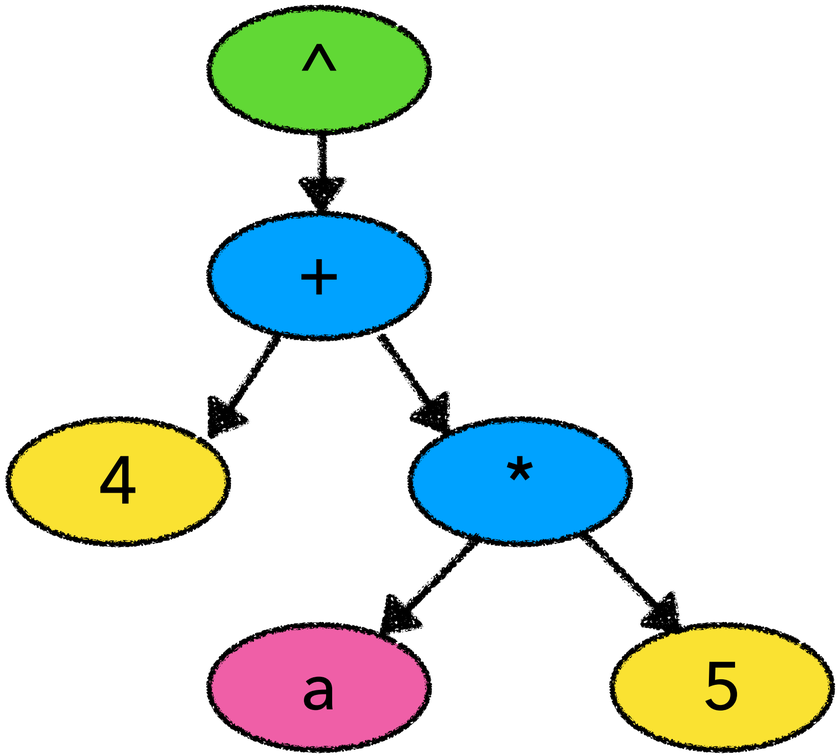


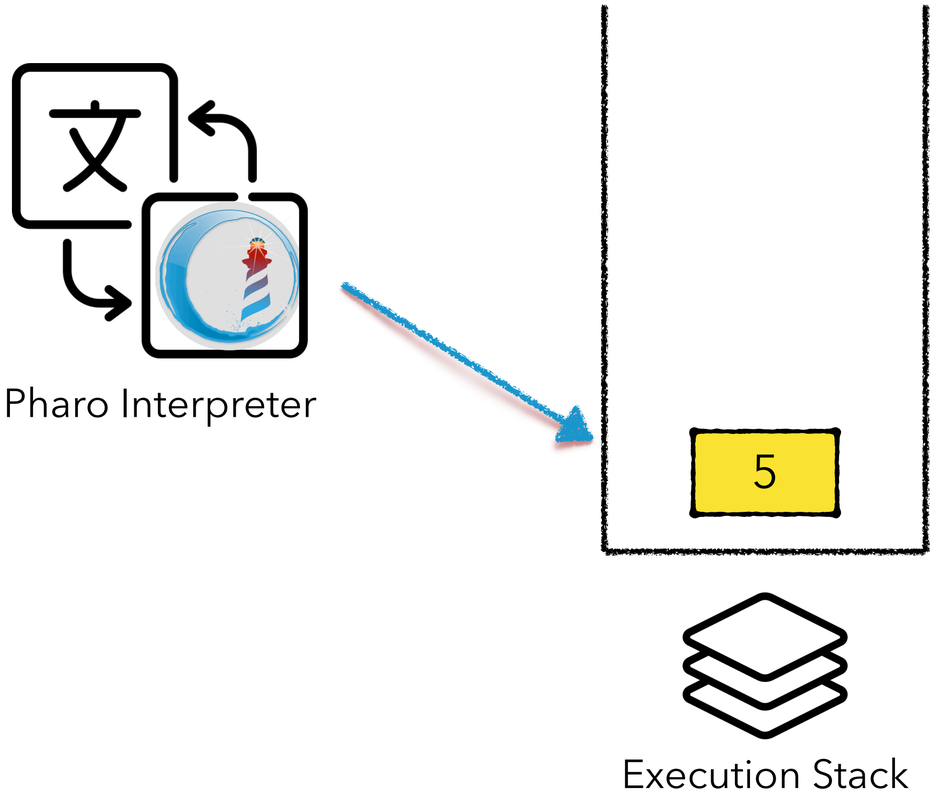
Pharo Interpreter



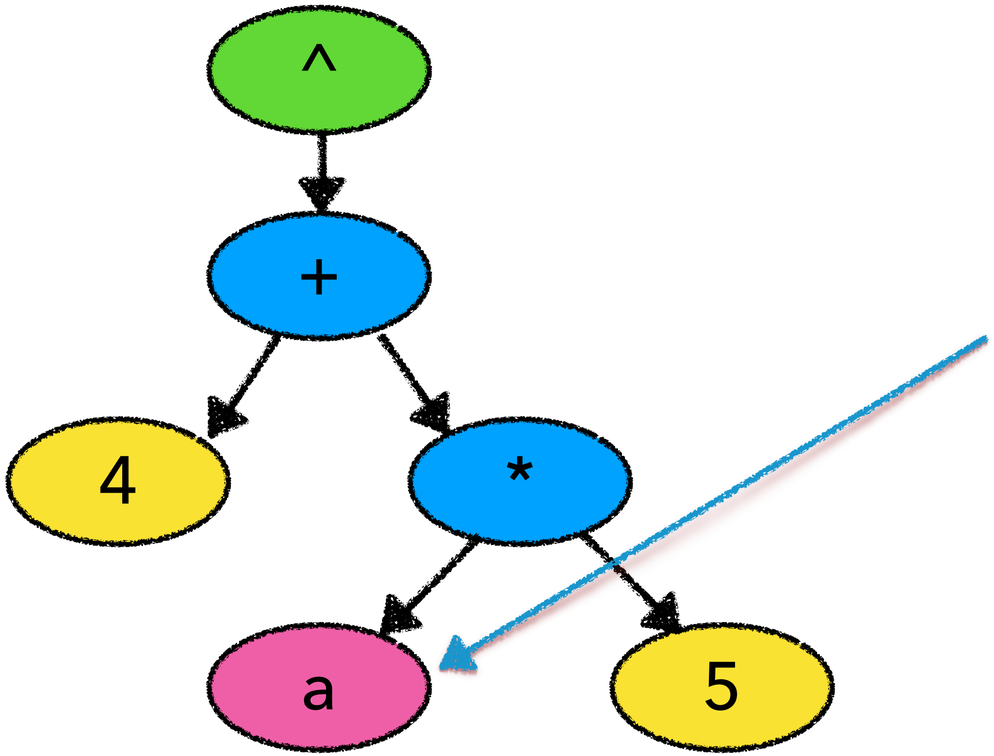
Execution Stack

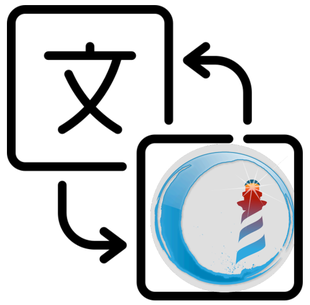
Interpreting the AST



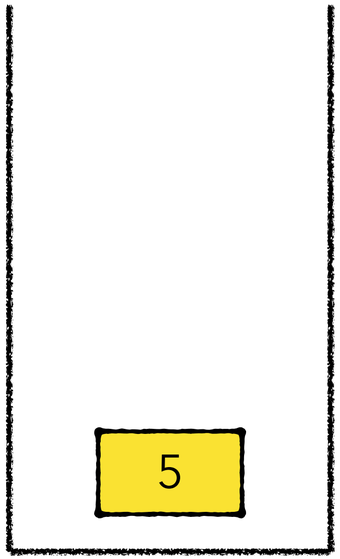


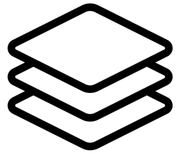
Interpreting the AST





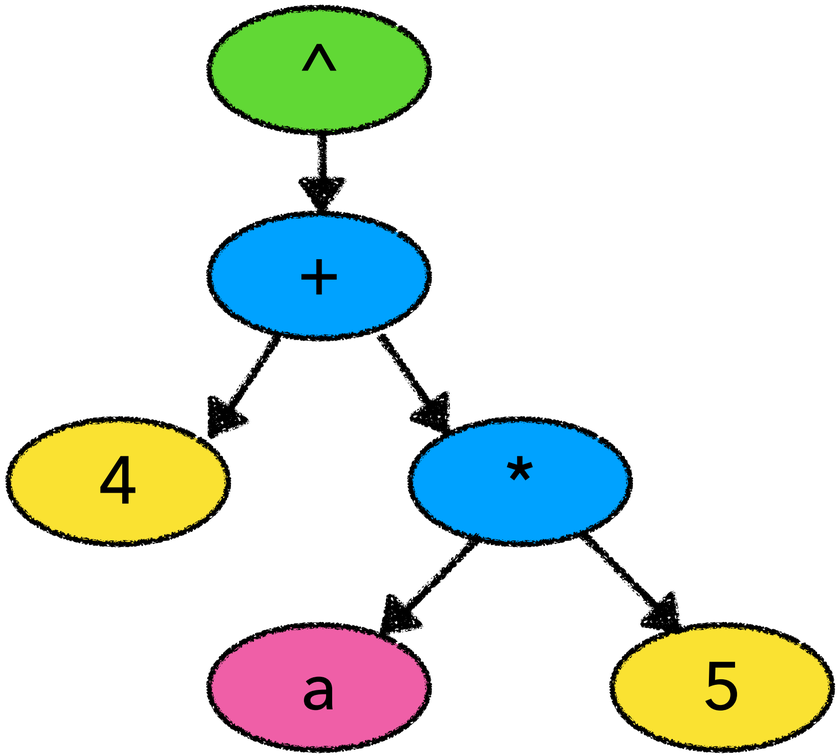
Pharo Interpreter

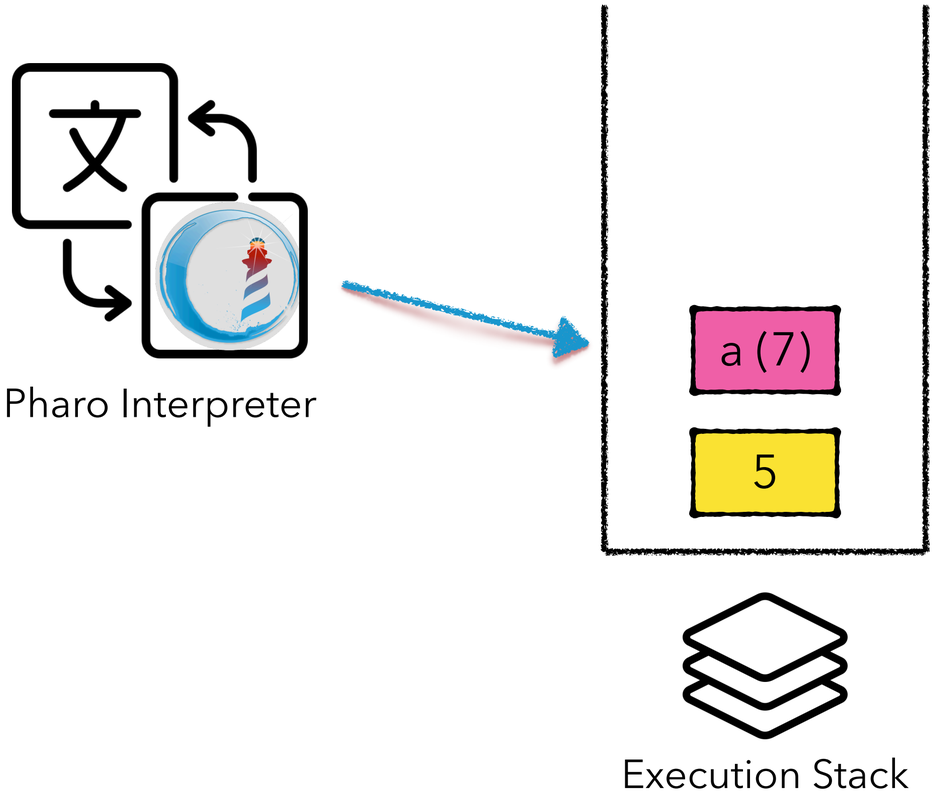




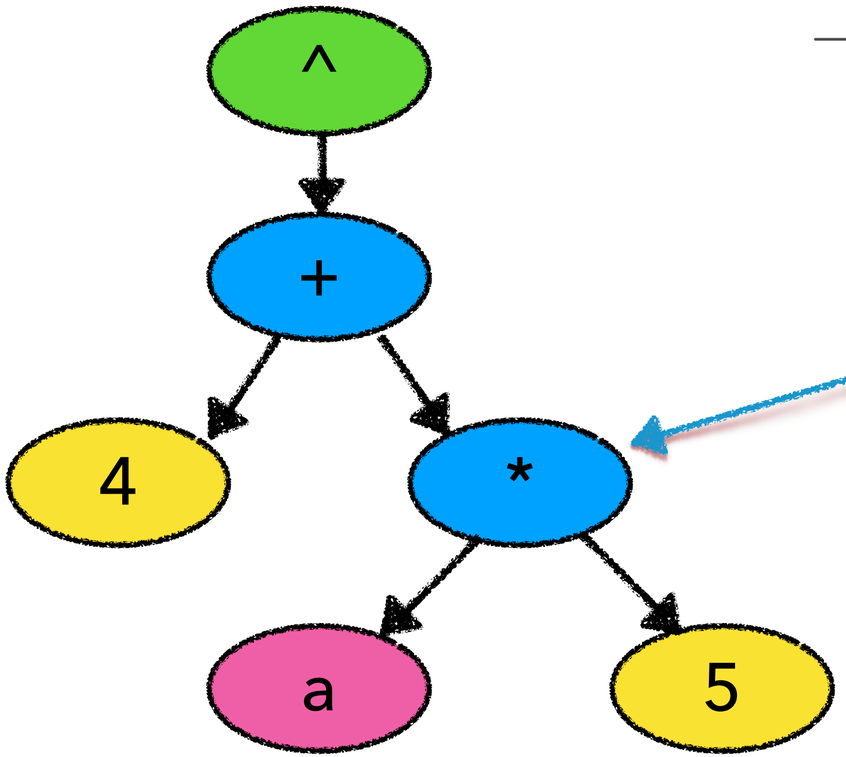
Execution Stack

Interpreting the AST

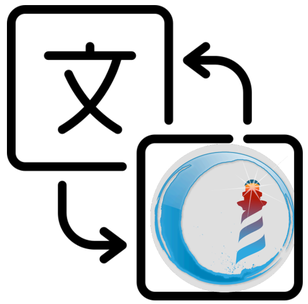




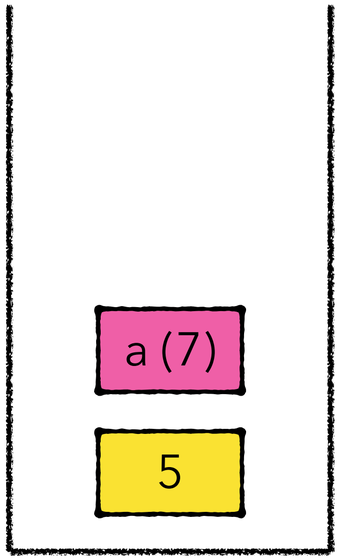
Interpreting the AST



A Message
(An invocation of a method)

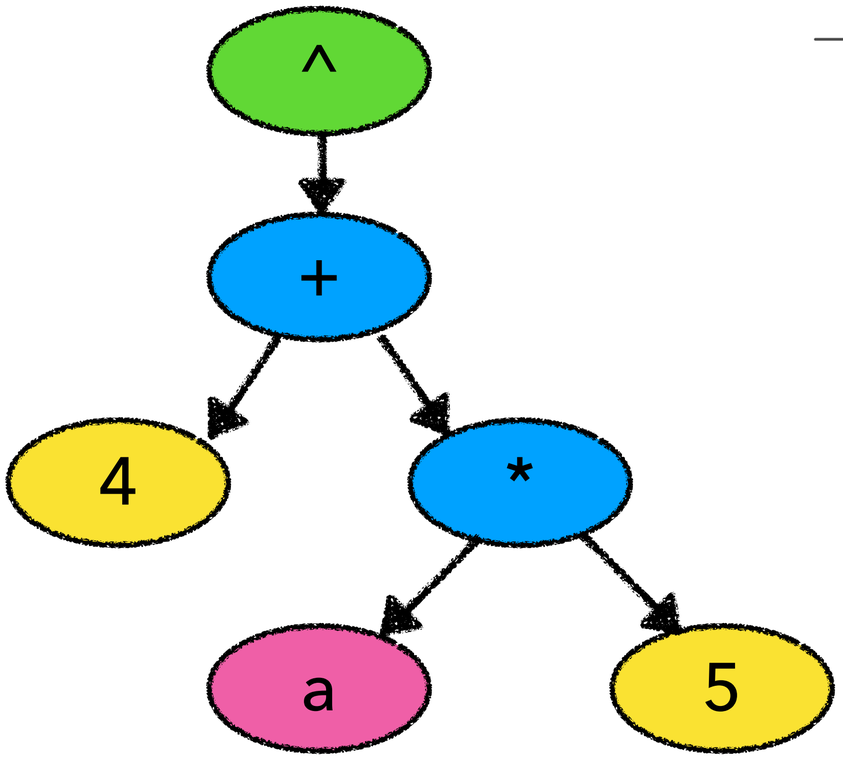


Pharo Interpreter

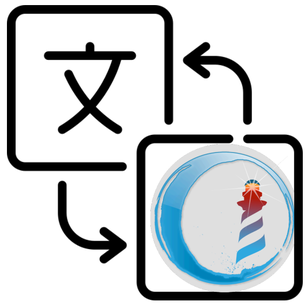


Execution Stack

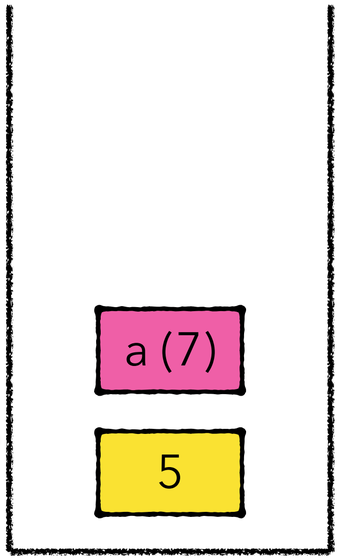
Interpreting the AST

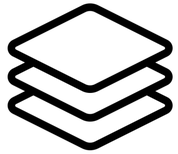


A Message
(An invocation of a method)



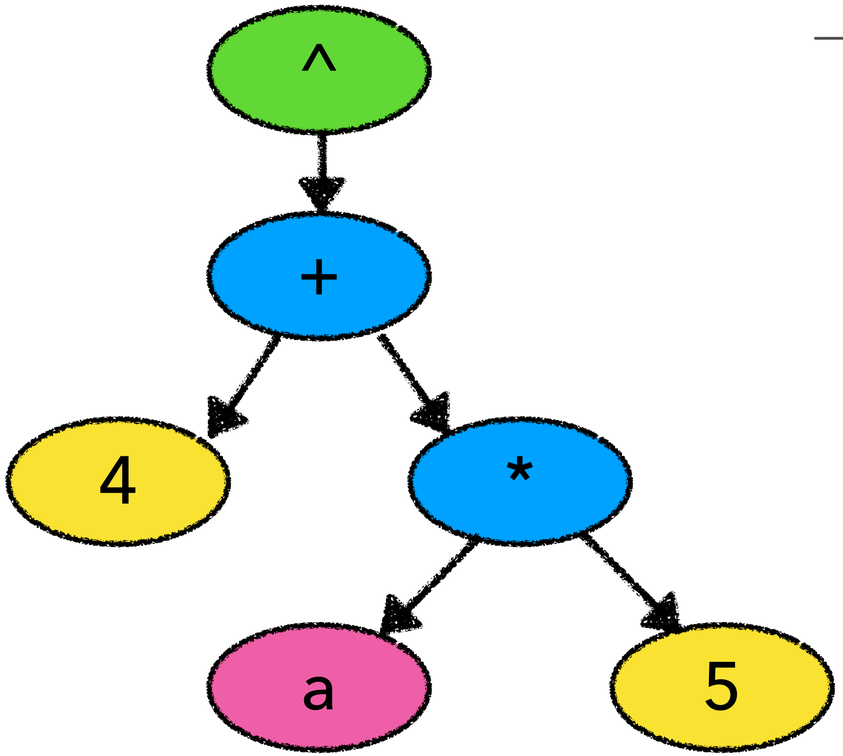
Pharo Interpreter



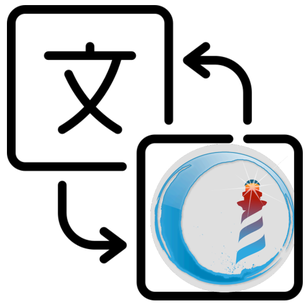


Execution Stack

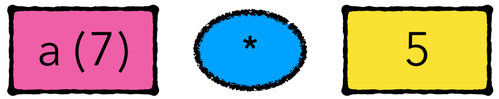
Interpreting the AST

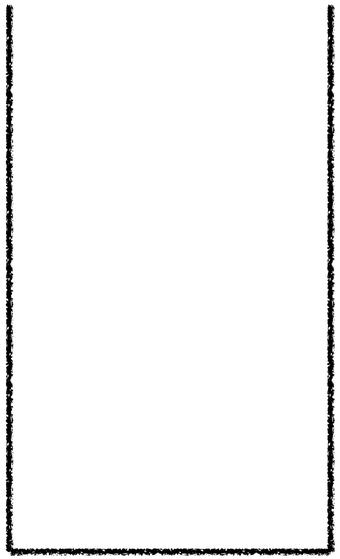


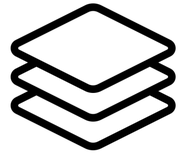
A Message
(An invocation of a method)



Pharo Interpreter

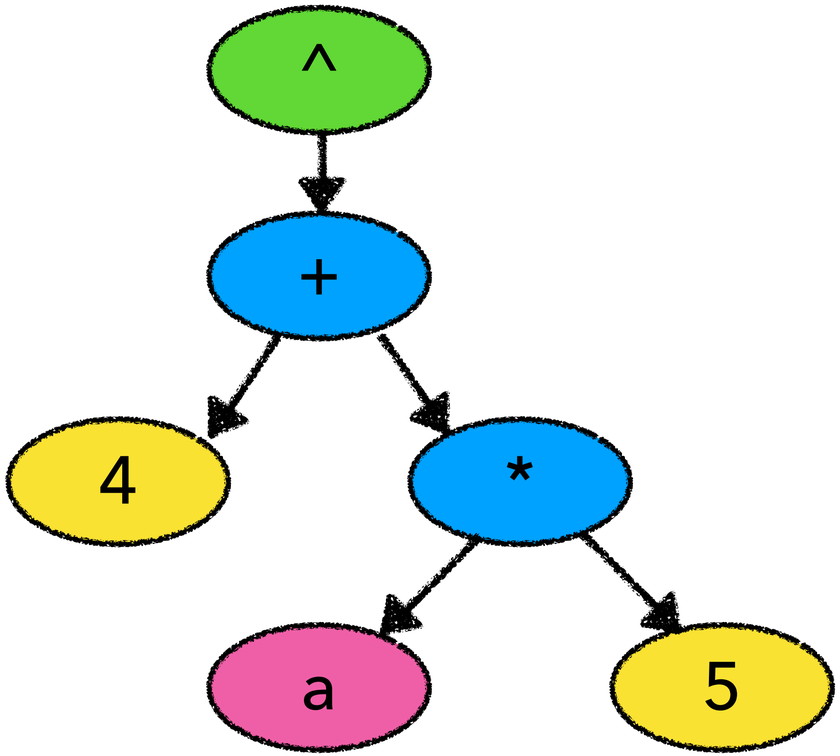


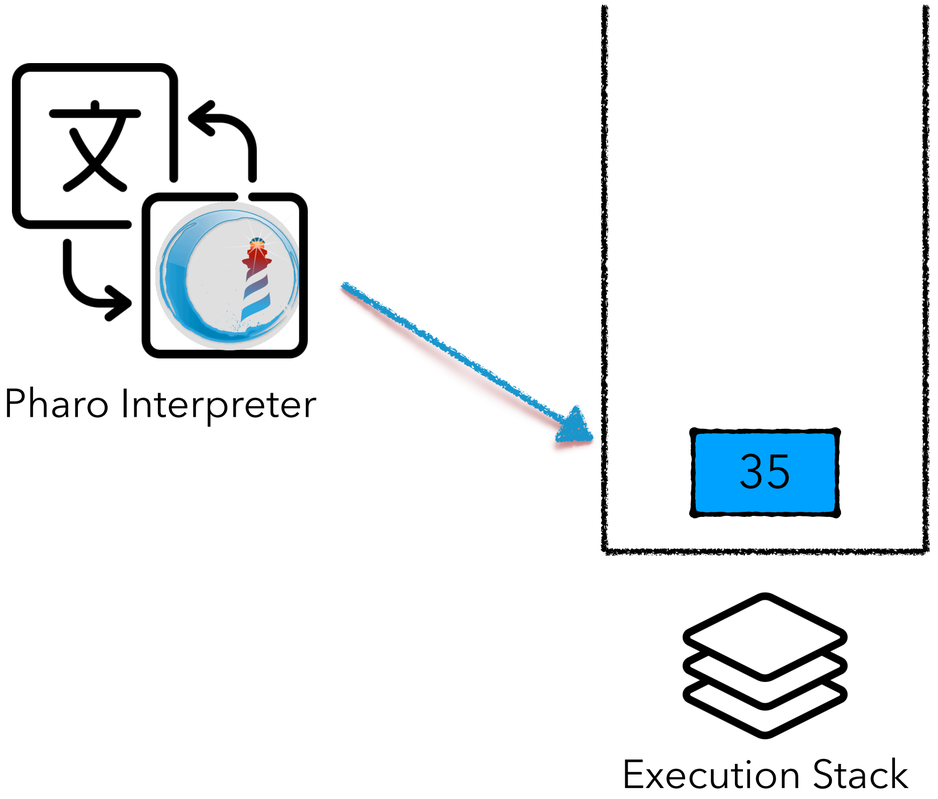




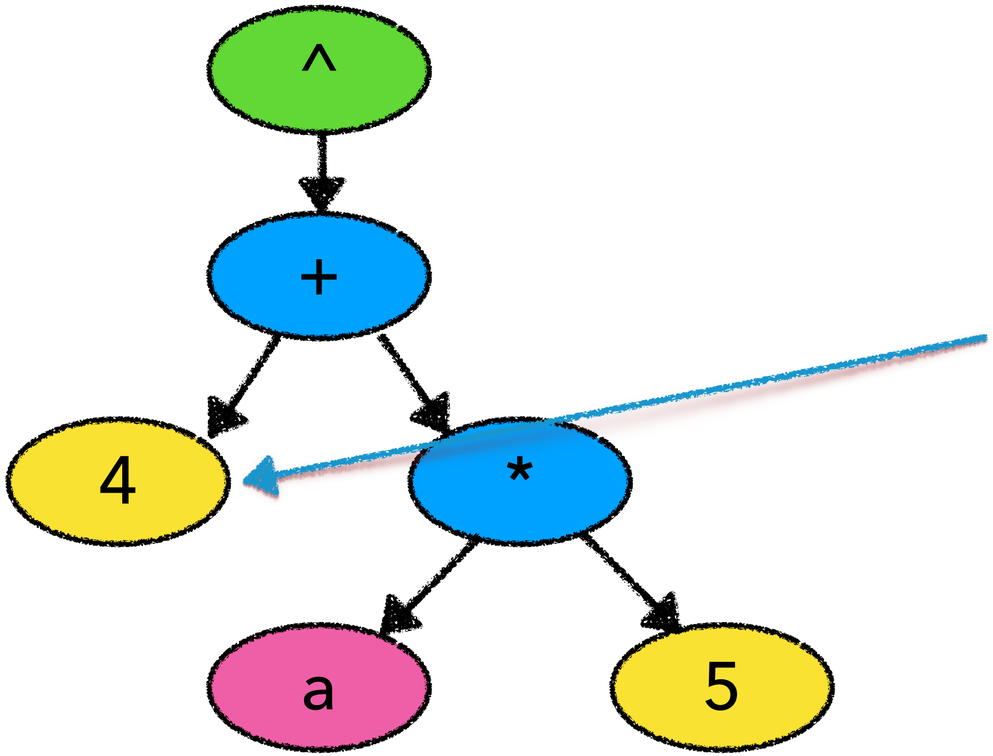
Execution Stack

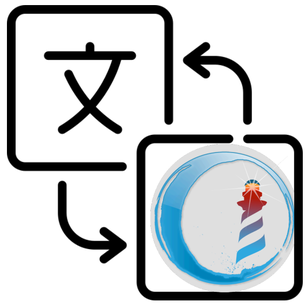
Interpreting the AST



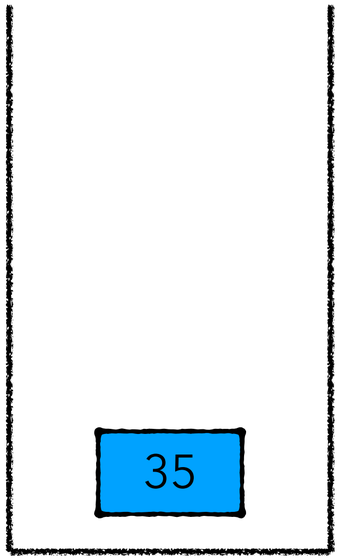


Interpreting the AST



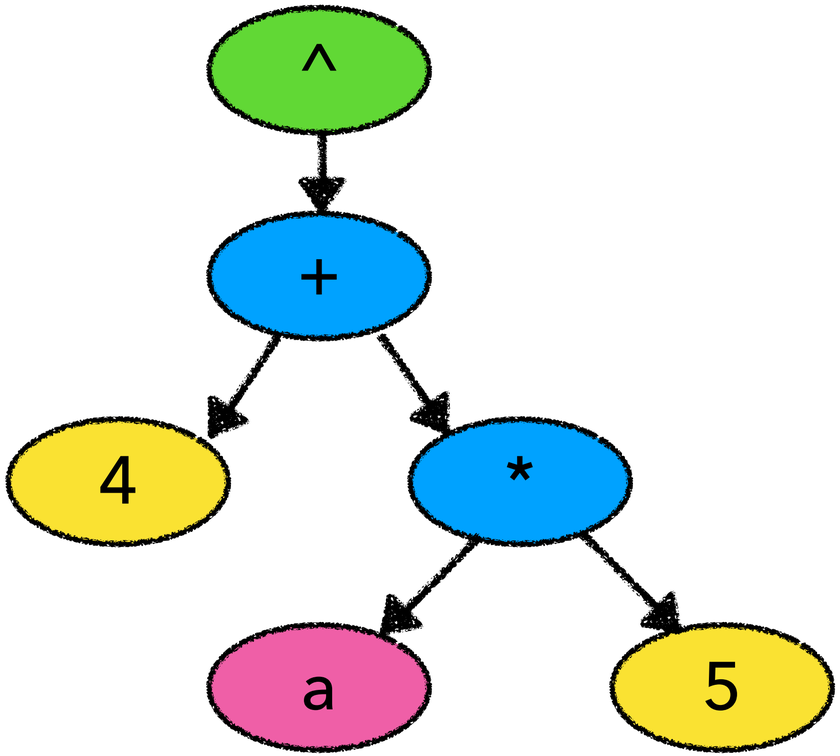


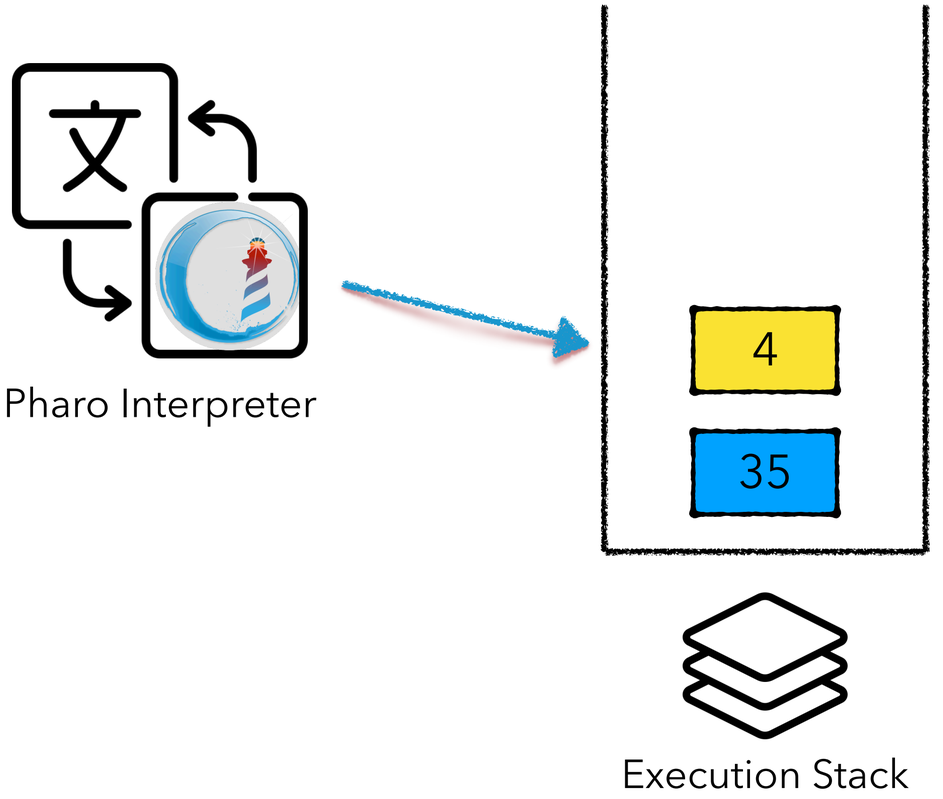
Pharo Interpreter



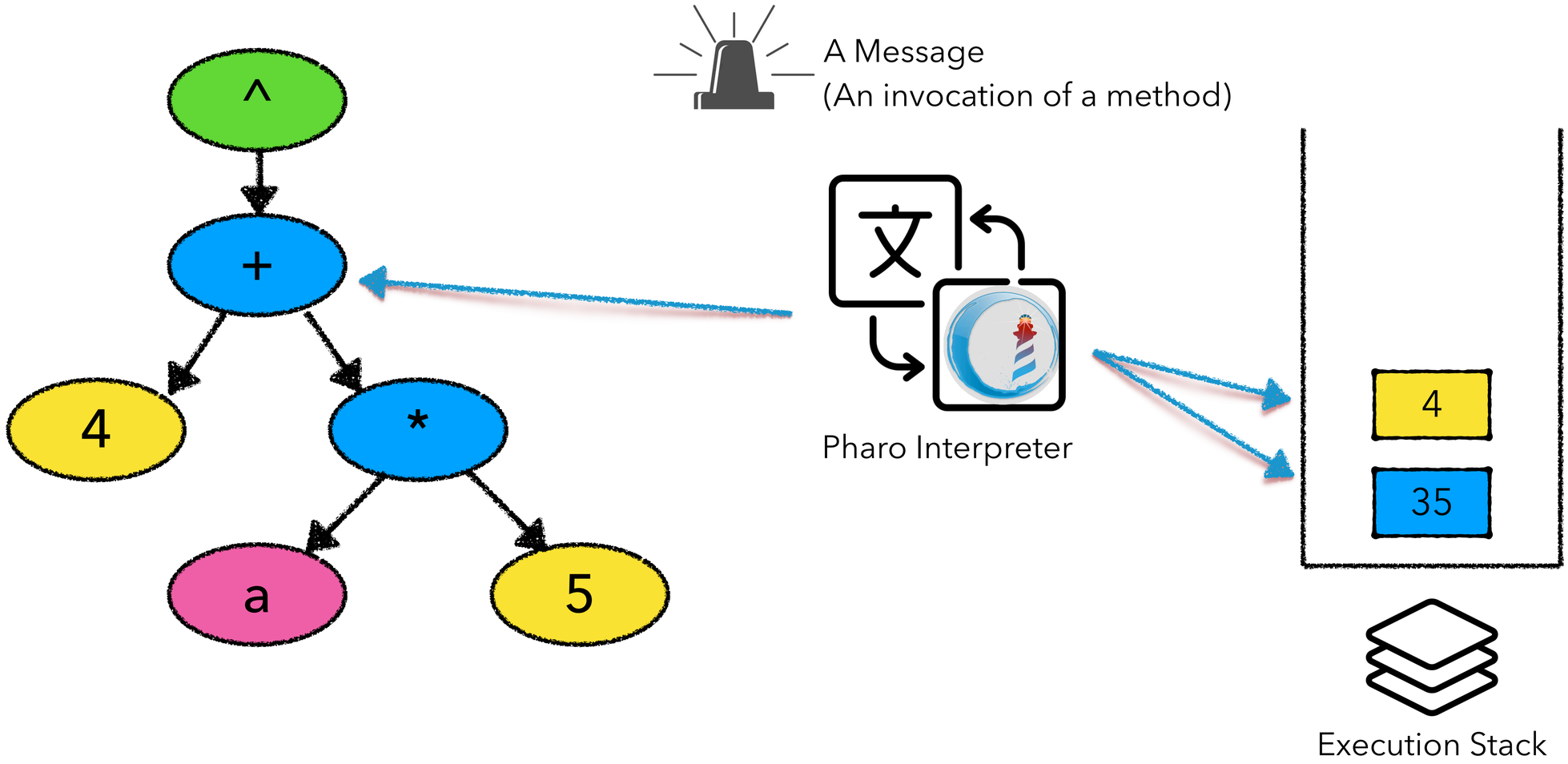
Execution Stack

Interpreting the AST

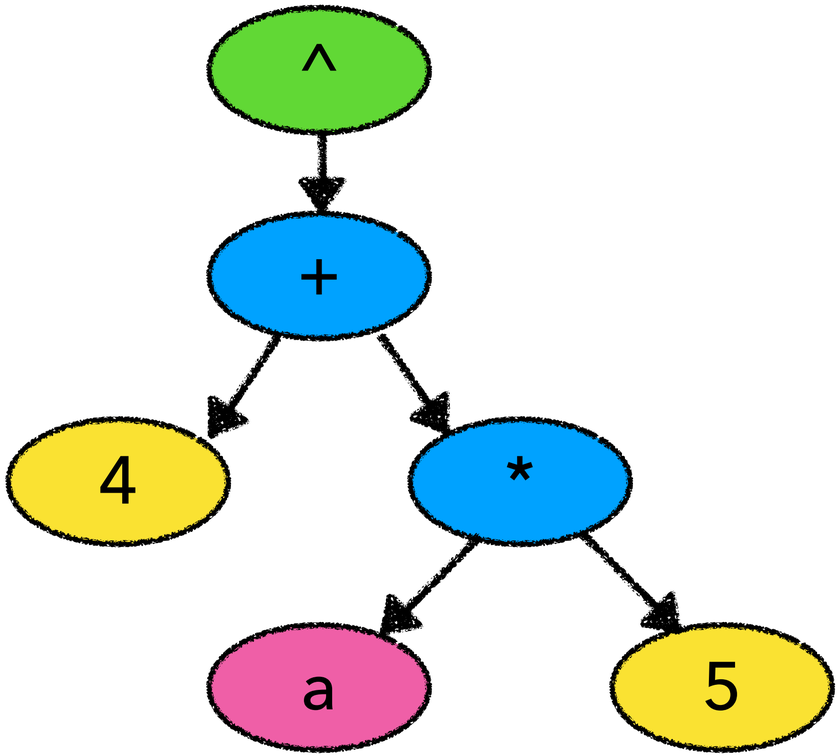


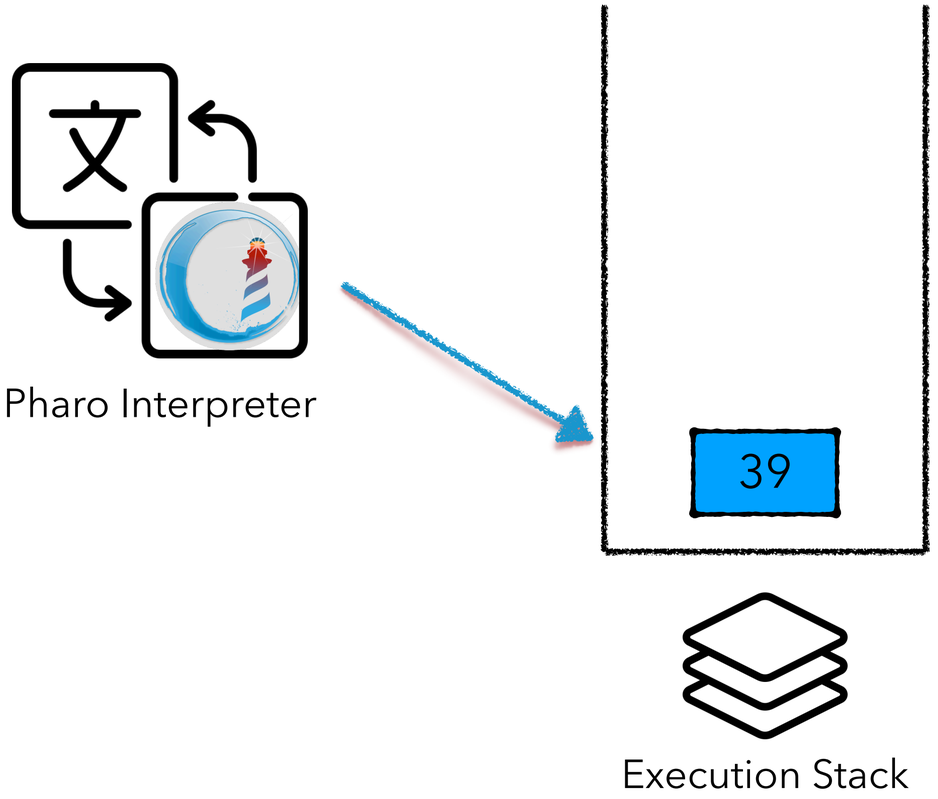


Interpreting the AST

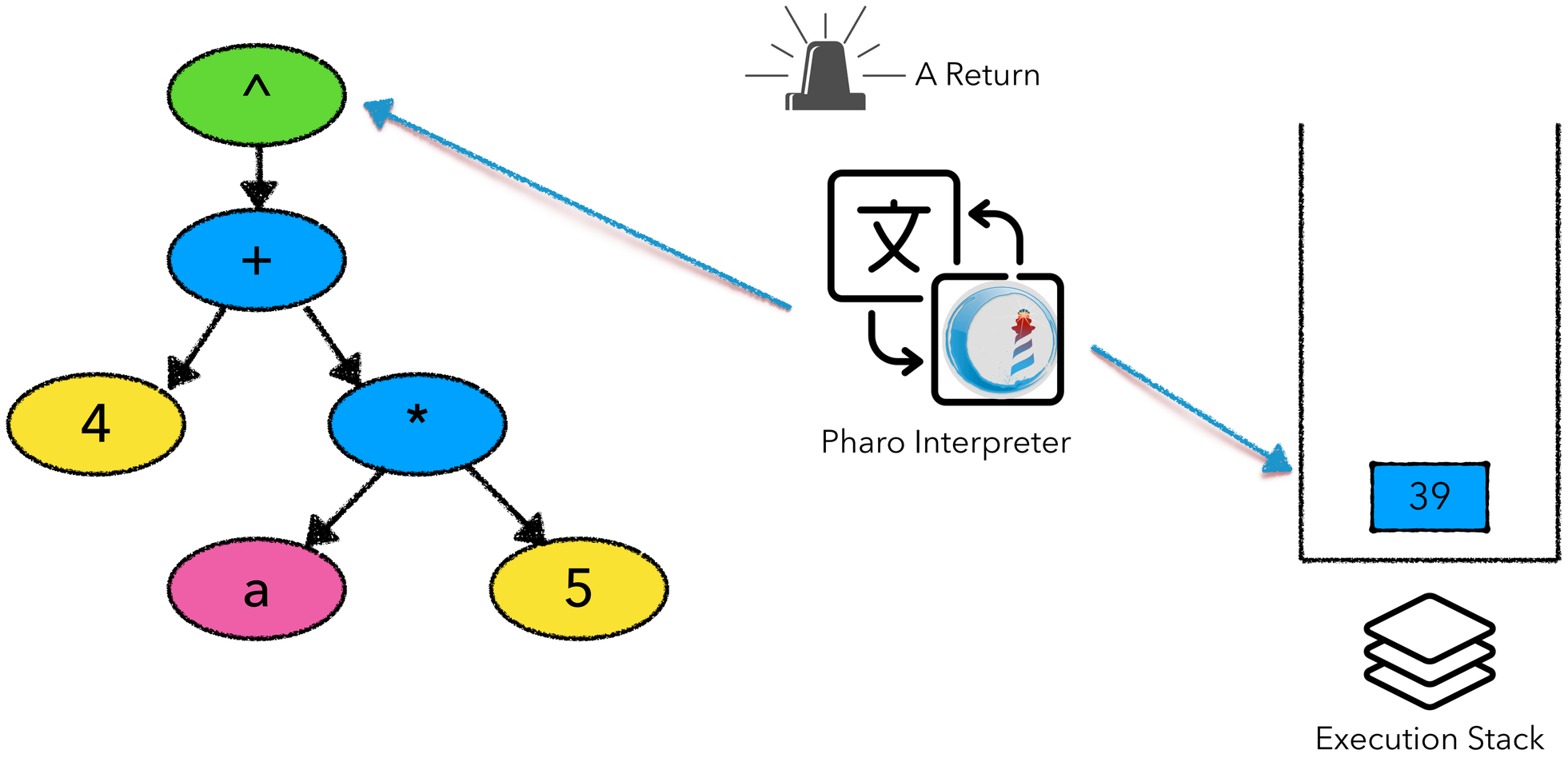


Interpreting the AST



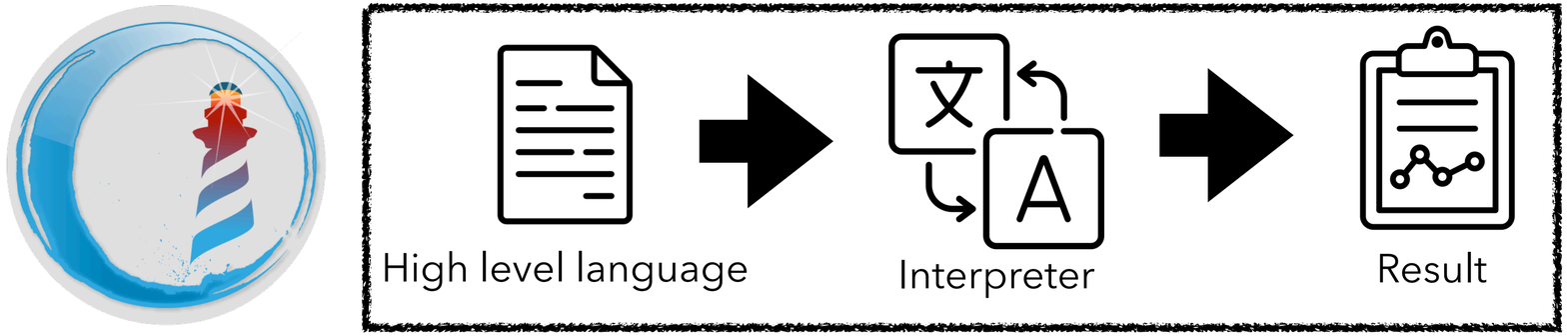


Interpreting the AST



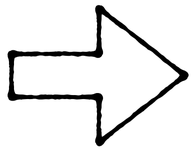
The Interpreter is a Program

Written in a Programming Language




```
^ 4 + (a * 5)
```

Pharo Code

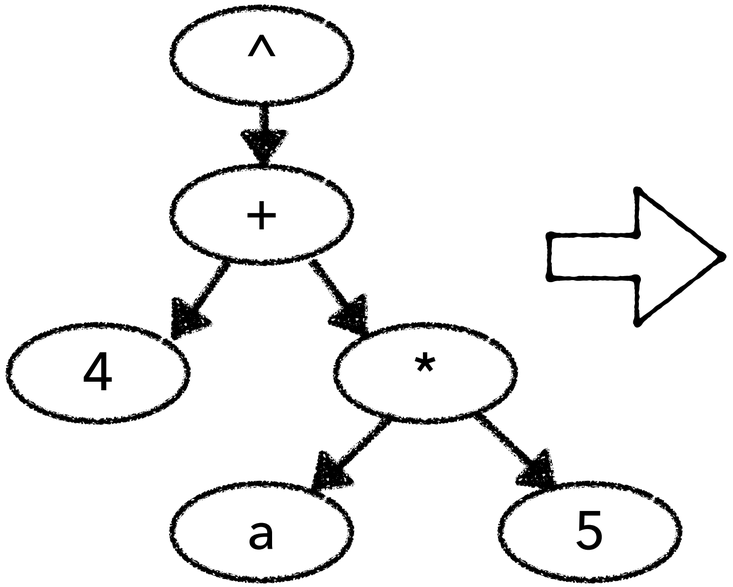


39

Result

Compiling the AST

Code Generation



An AST

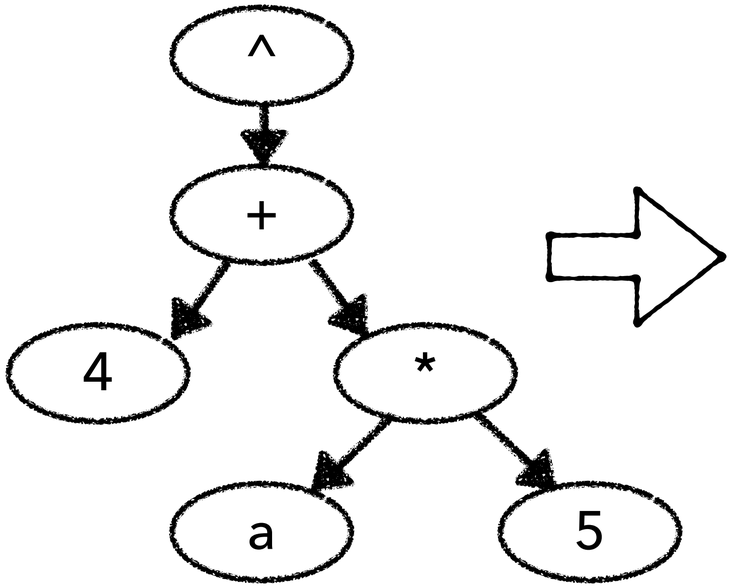
The Pharo Compiler

Code generation

```
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
...
```

Compiling the AST

Code Generation



An AST

The Pharo Compiler

Code generation

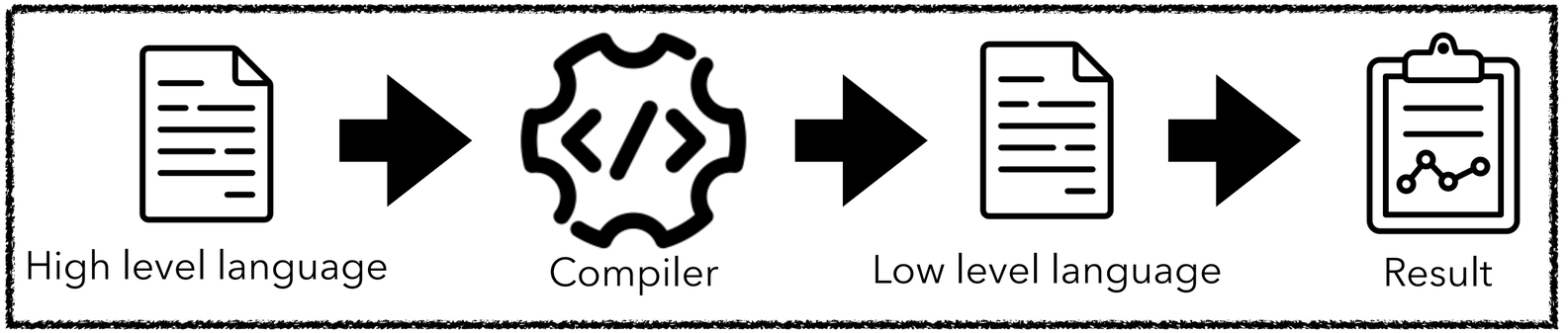
```
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
...
```

```
FADD d0, d0, d1
FMOV d1, #3
FDIV d0, d1, d0
```

```
FADD.D ft1, fa0, fa1
FDIV.D fa0, ft0, ft1
CALL ceil@plt
```

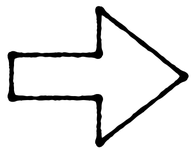
The Compiler is (also) a Program

(That is also) Written in a Programming Language




```
^ 4 + (a * 5)
```

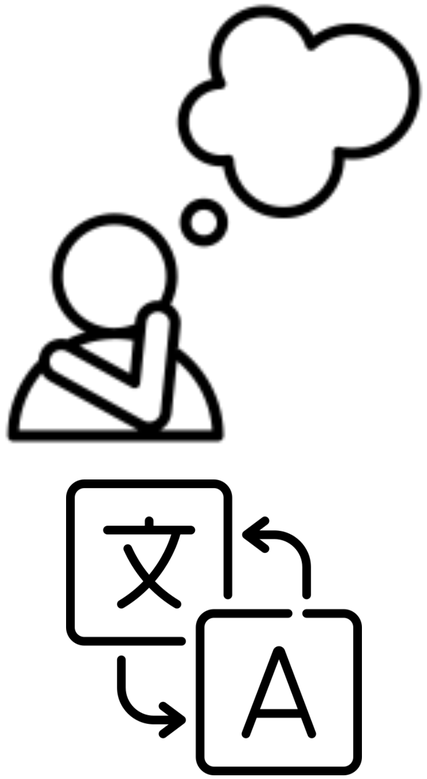
Pharo Code



Low Level Language

So, which is better?

Interpreting is Slower

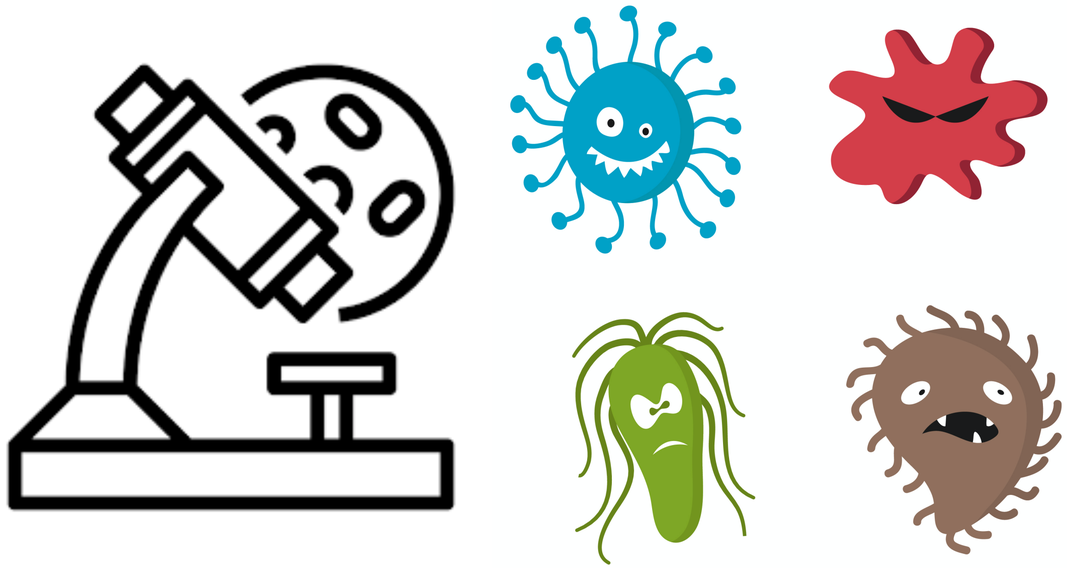


The Interpreter

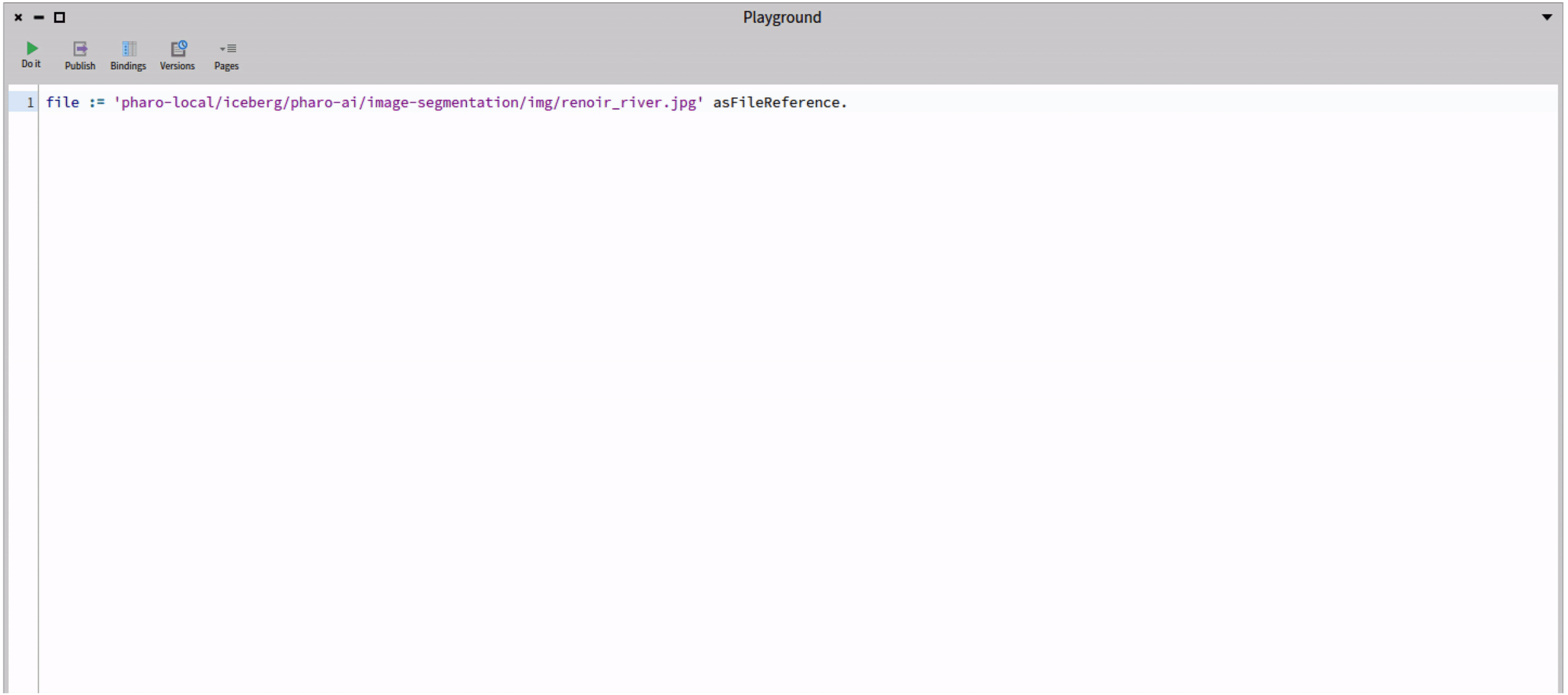
Compiling is Faster

The Compiler

Interpreting gives you Reflective Capabilities



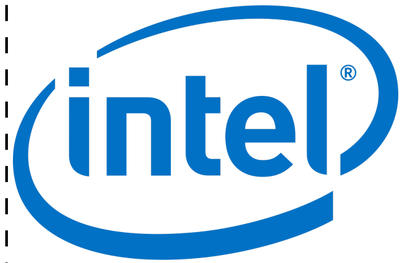
Interpreting gives you Reflective Capabilities



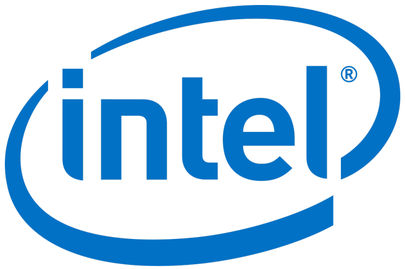
Compiling makes you loose information

About the Behaviour of the Code

```
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
...
```



Compiling is Platform Dependent

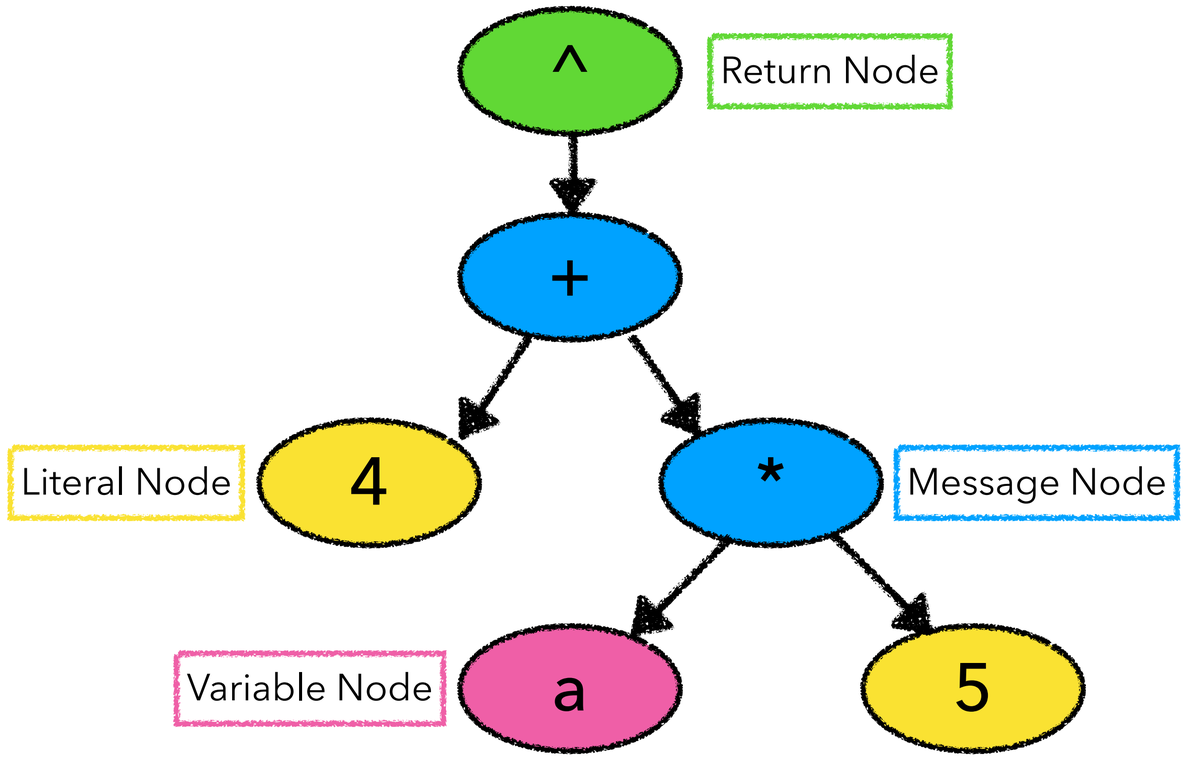


Combining Both Strategies

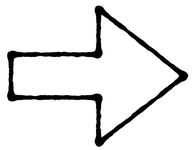
Introducing Bytecode

Bytecode

A Set of Instructions not Dependent of the Platform



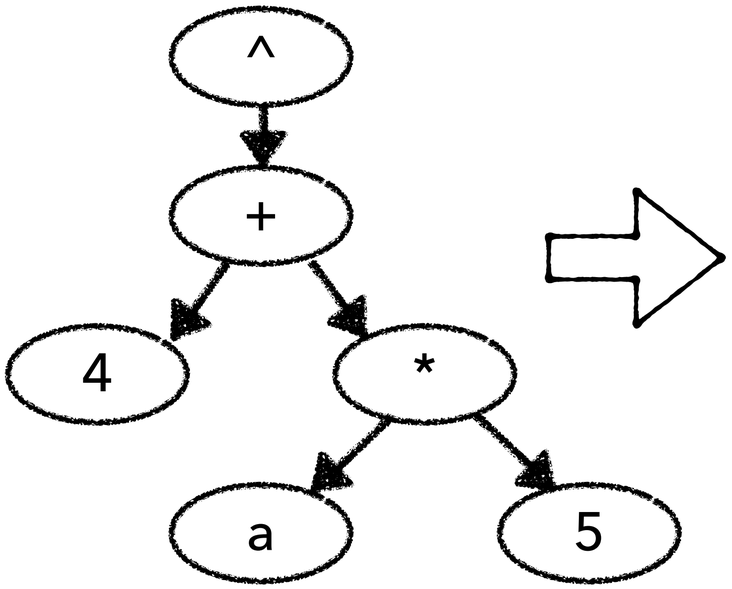
An AST



```
pushConstant 4
pushRcvr a (7)
pushConstant 5
send *
send +
returnTop
```

Compiling the AST

To Bytecode



An AST

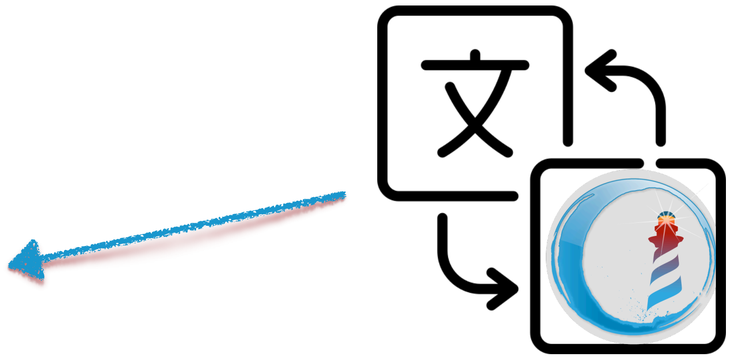
The Pharo Bytecode Compiler

Code generation

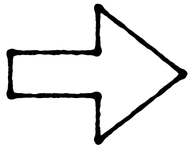
```
pushConstant 4
pushRcvr a (7)
pushConstant 5
send *
send +
returnTop
```

Interpreting the Bytecode

```
pushConstant 4
pushRcvr a (7)
pushConstant 5
send *
send +
returnTop
```



Pharo Byte Code Interpreter

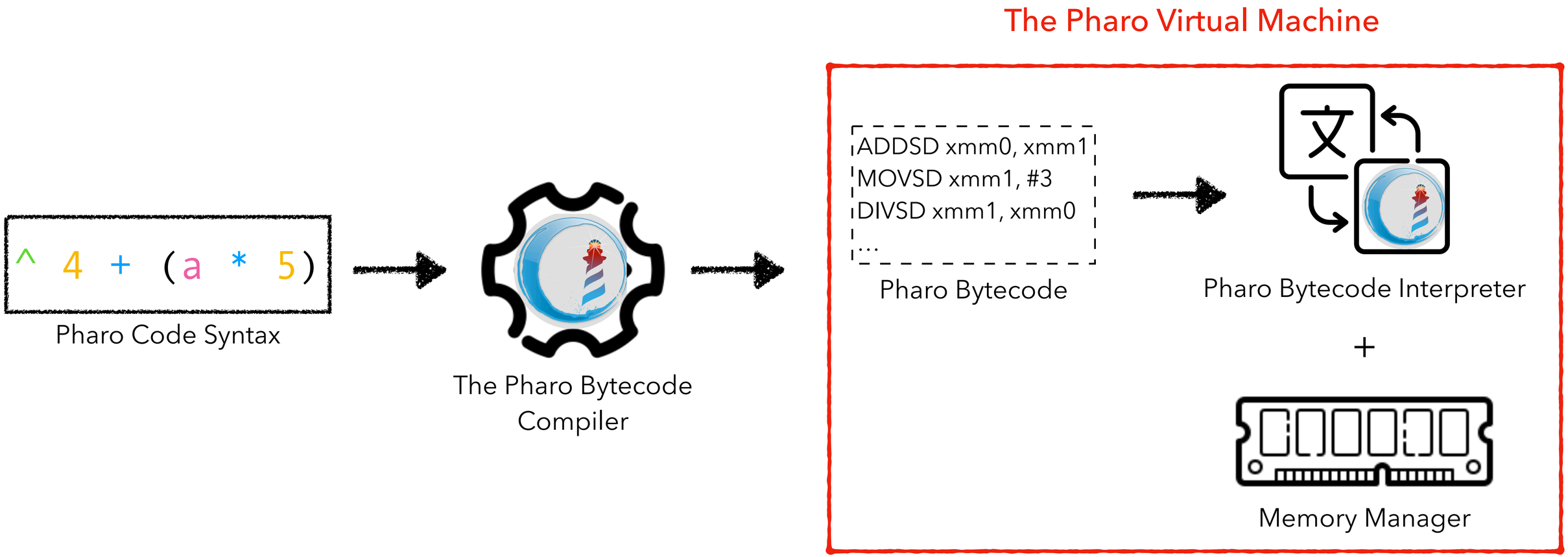


39

Result

So, What is a Virtual Machine?

A Virtual Machine



A Programming Language

The Pharo Programming Language

^ 4 + (a * 5)

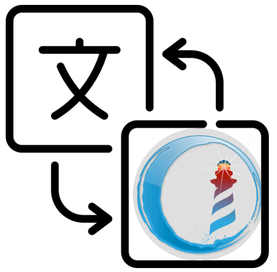
Pharo Code Syntax

The Pharo Bytecode Compiler

The Pharo Virtual Machine

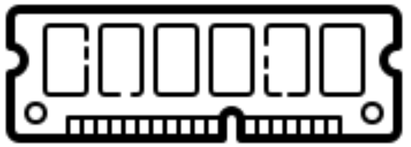
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
…

Pharo Bytecode



Pharo Bytecode Interpreter

+



Memory Manager

The Pharo Programming Language

Language Libraries

Math

Web

Machine Learning

+

Integrated Development Environment (IDE)

+

Refactoring Tools

+

The Pharo Virtual Machine

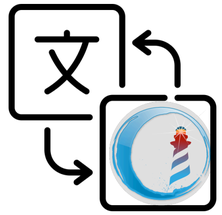
```
^ 4 + (a * 5)
```

Pharo Code

The Pharo Bytecode Compiler

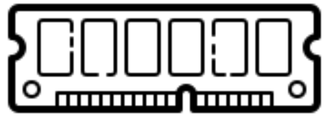
```
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
...
```

Pharo Bytecode



Pharo Bytecode Interpreter

+



Memory Manager

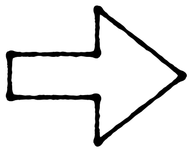
Bonus 1: JIT Compiler

The JIT Compiler

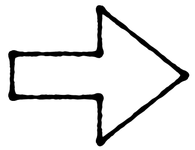
From Bytecode to Machine Code

```
pushConstant 4
pushRcvr a (7)
pushConstant 5
send *
send +
returnTop
```

Pharo Bytecode



The Pharo JIT Compiler



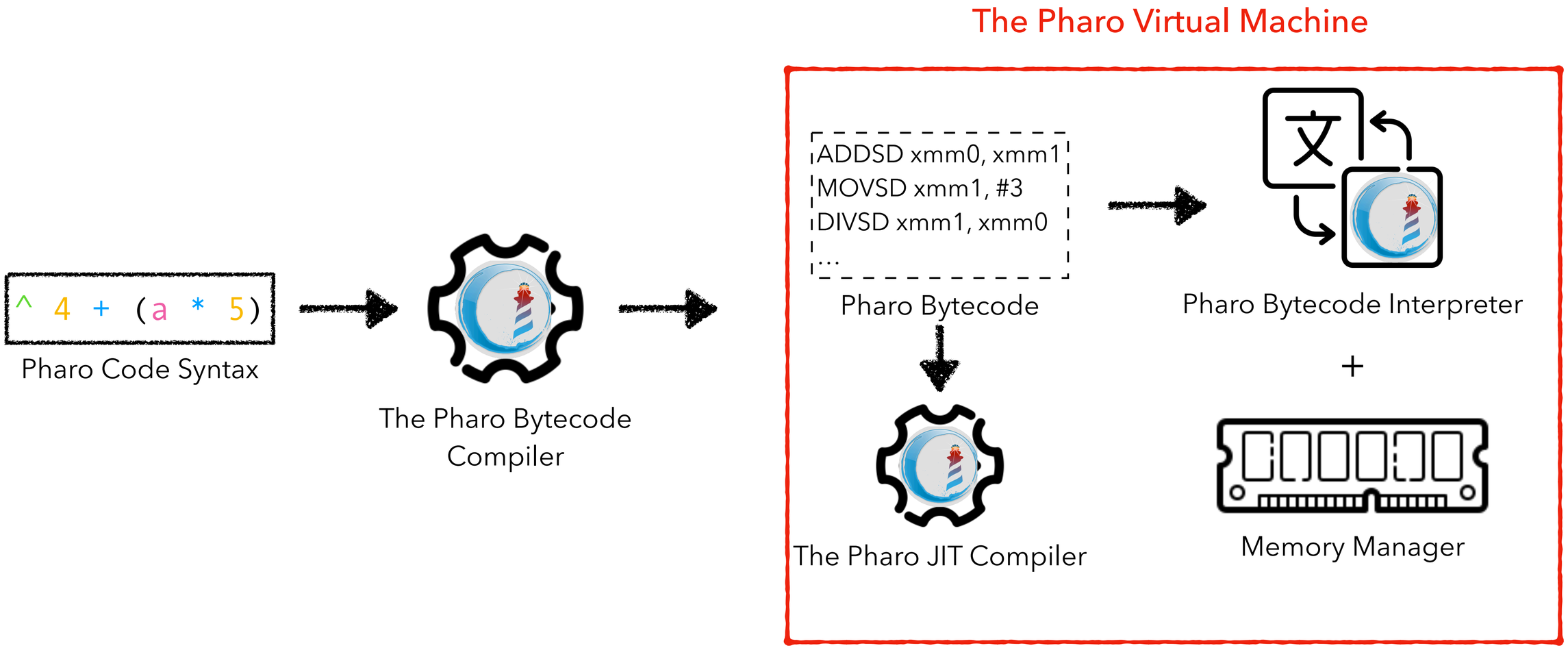
Code generation

Intel Machine Code

```
ADDSD xmm0, xmm1
MOVSD xmm1, #3
DIVSD xmm1, xmm0
...
```

The Actual Pharo Virtual Machine

With The JIT Compiler



Virtual Machines and Programming Languages

pharo.org

Sebastian JORDAN MONTAÑO

sebastian.jordan@inria.fr

RMod

cnrs

Inria

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

Université de Lille

centralelille
ÉCOLE CENTRALE DE LILLE

December 2022