Demystifying Python's Internals

Diving into CPython by implementing a pipe operator

Sebastiaan Zeeff

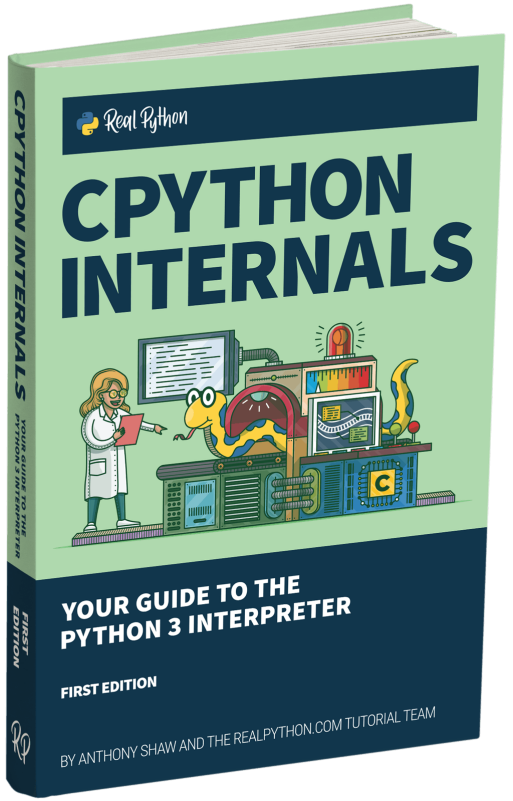
Sebastiaan Zeeff

Source code: https://github.com/SebastiaanZ/pypethon

Talk Outline

- We're going to dive into **CPython's Internals**, including:
  - Tokens, Grammar, & the PEG parser
  - Abstract Syntax Trees (AST)
  - The compiler, bytecode, and "opcodes" (instructions)
  - The Evaluation Loop
- We'll do that by implementing a **pipe operator**
- Due to **time constraints,** this talk will feature**:**
  - **Blatant omissions**
  - **Gross oversimplifications**

Source code: https://github.com/SebastiaanZ/pypethon



Python Developer's Guide

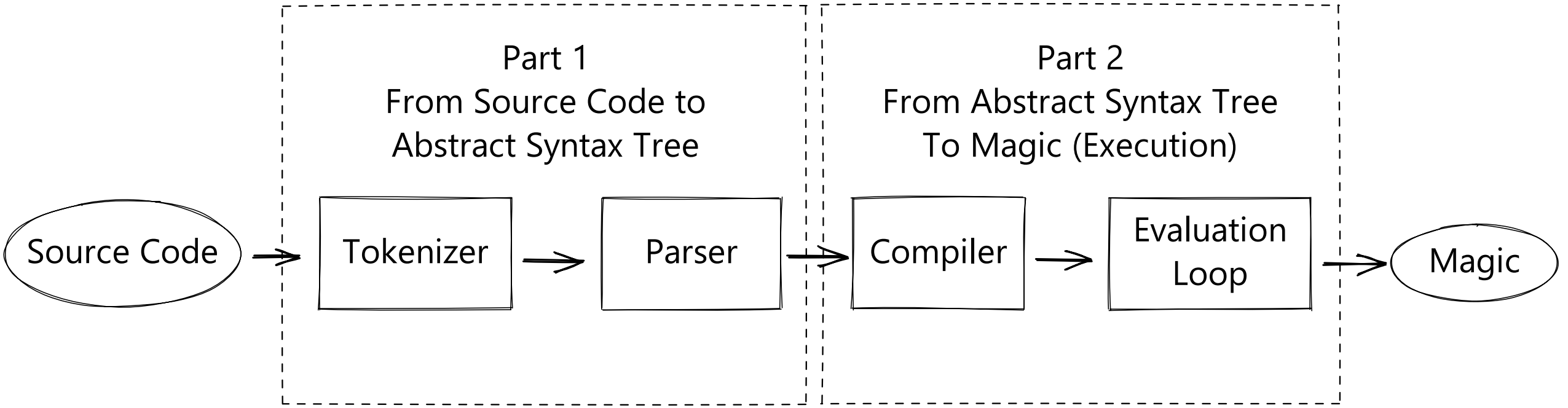
https://devguide.python.org

CPython Internals

Anthony Shaw & Real Python

Source code: https://github.com/SebastiaanZ/pypethon

From source code to execution



Source code & slides are available on GitHub:

https://github.com/SebastiaanZ/pypethon

Adding a new binary operator: A pipe operator (|>)

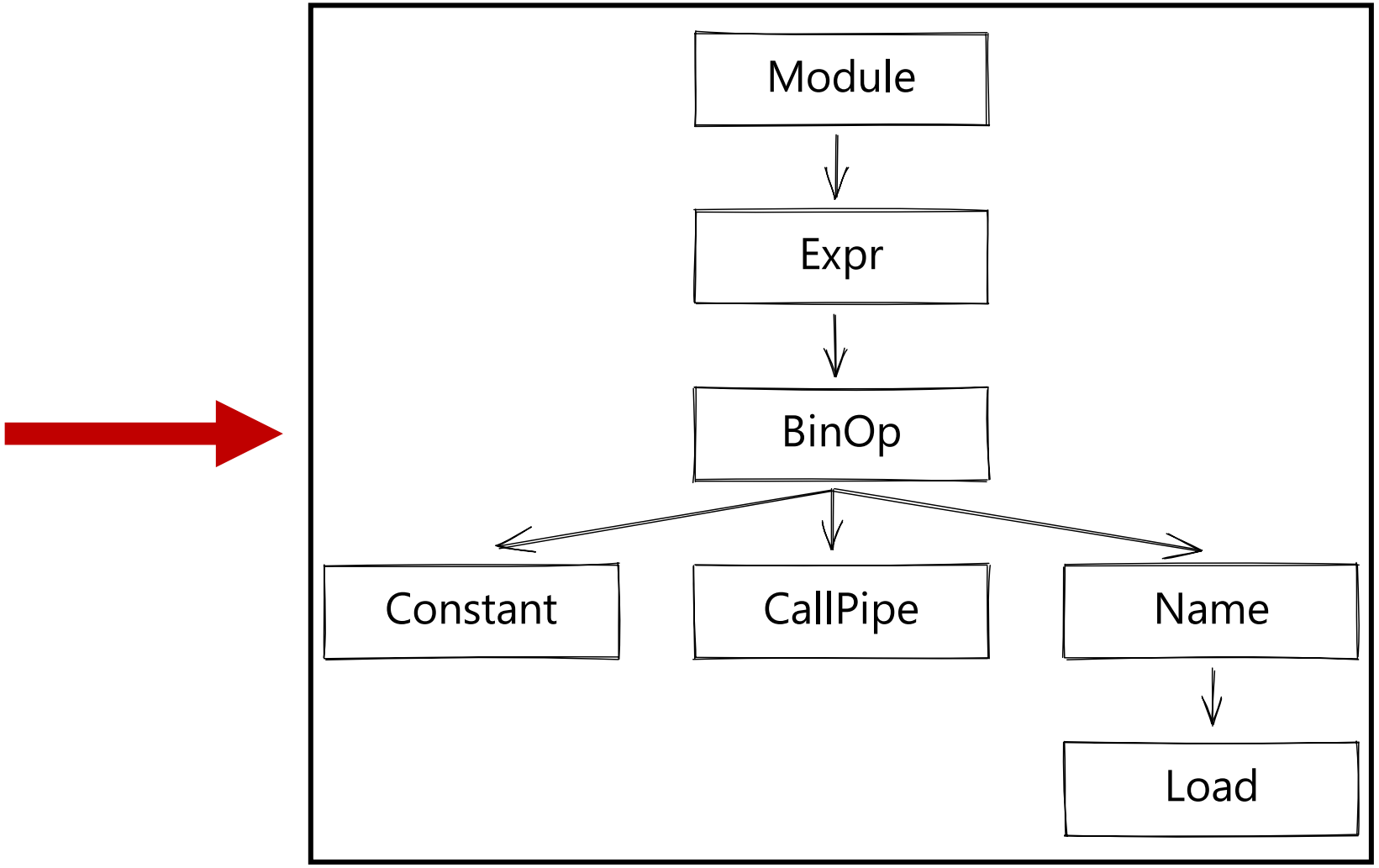
```
>>> def double(number):
...     return number * 2

>>> 1 |> double  # double(1)
2

>>> 1 |> double |> double  # double(double(1))
4
```

- Just to be clear: This operator is **not** a part of Python.
  - *I don't think there are any plans to add such an operator to Python.*
- The implementation is purely **educational**: Feel free to try and extend it!

Source code: https://github.com/SebastiaanZ/pypethon

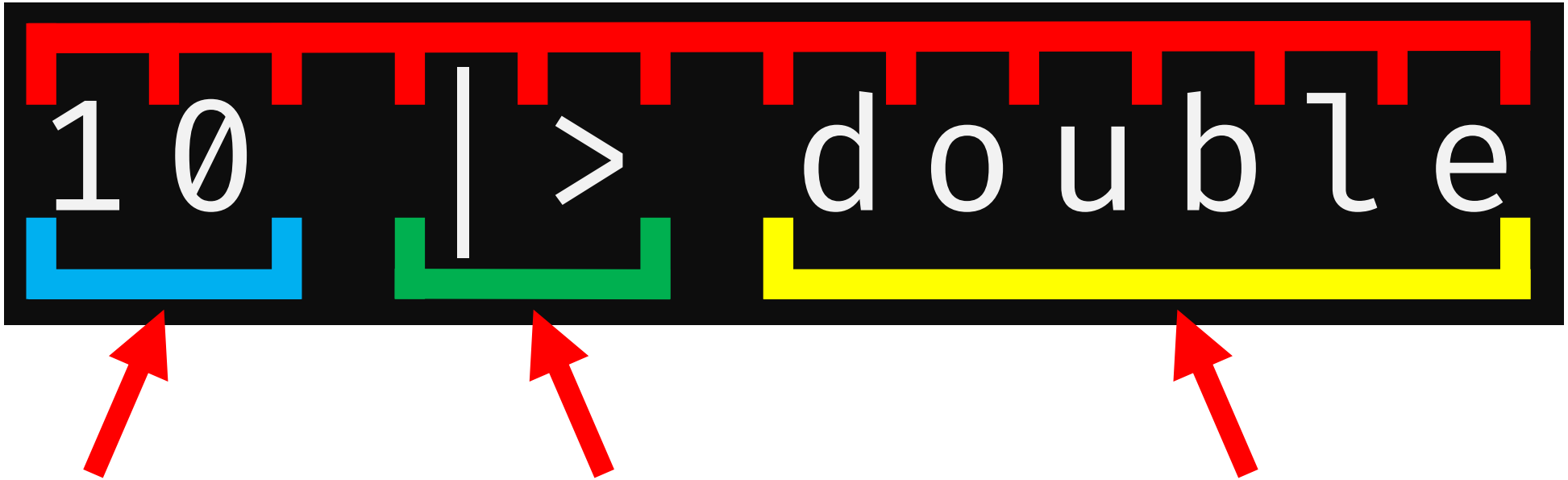
Part I: From Source Code to Abstract Syntax Tree (AST)

```
10 |> double
```



Source code: https://github.com/SebastiaanZ/pypethon

Tokenization: from raw text to a stream of tokens



NUMBER token

??? token

NAME token

Source code: https://github.com/SebastiaanZ/pypethon

Tokenization: Add a token for the operator

File: Grammar/Tokens

```
# (...)
ATEQUAL                 '@='
RARROW                  '->'
ELLIPSIS                '...'
COLONEQUAL              ':='
VBARGREATER             '|>'
```

Linux/Mac

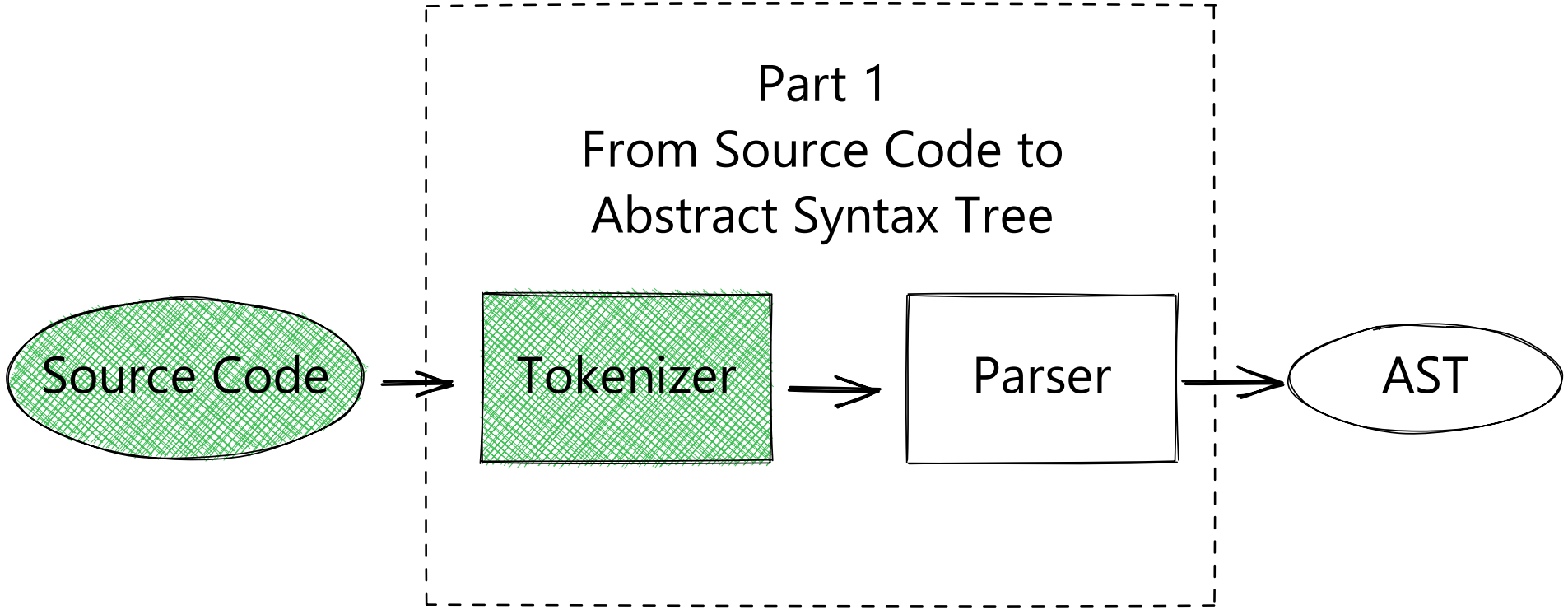
```
$ make regen-token
```

Windows

```
> PCBuild\build.bat --regen
```

Source code: https://github.com/SebastiaanZ/pypethon

Part 1: Progress



To do in part 1:

- Add support for the operator in the grammar
- Add support for the operator in the AST

Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

```
sum:
    | atom '+' atom
    | atom

atom:
    | NUMBER
```

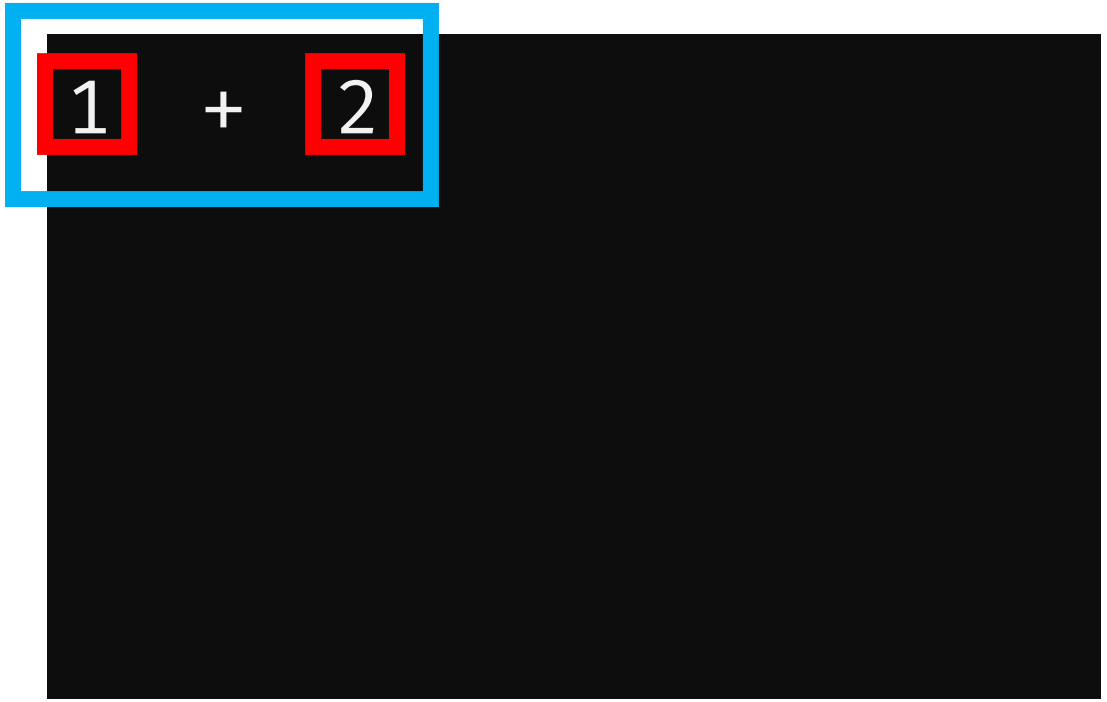
```
1
```

Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

```
sum:
     | atom '+' atom
     | atom

atom:
     | NUMBER
```

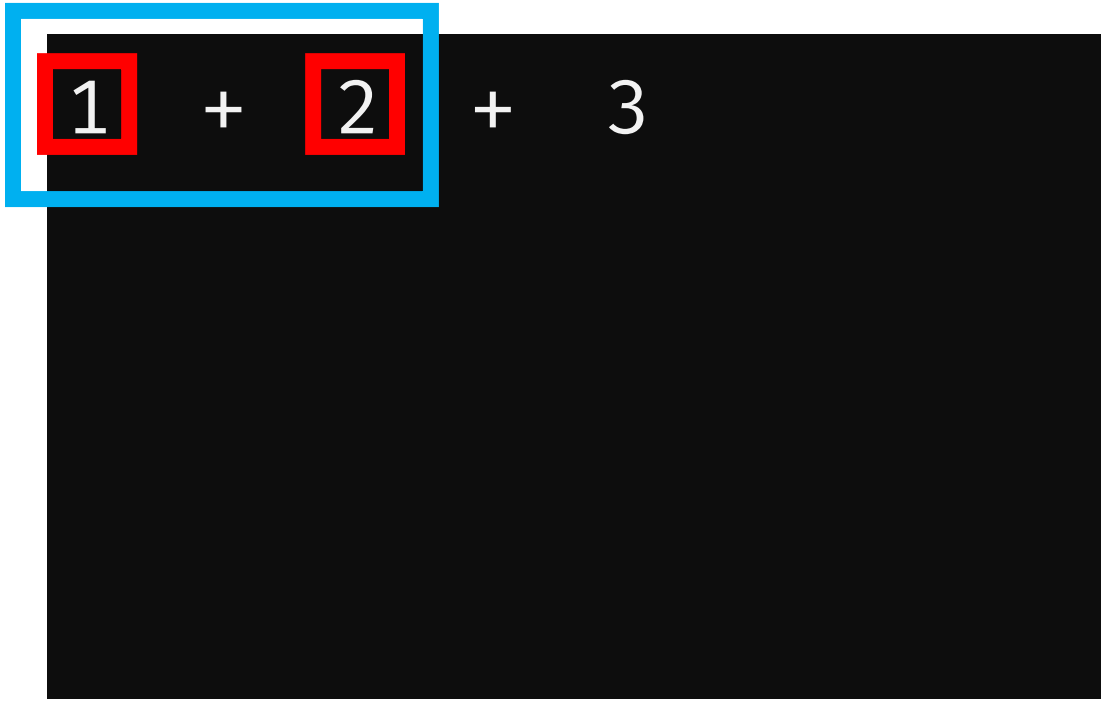


Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

```
sum:
     | atom '+' atom
     | atom

atom:
     | NUMBER
```

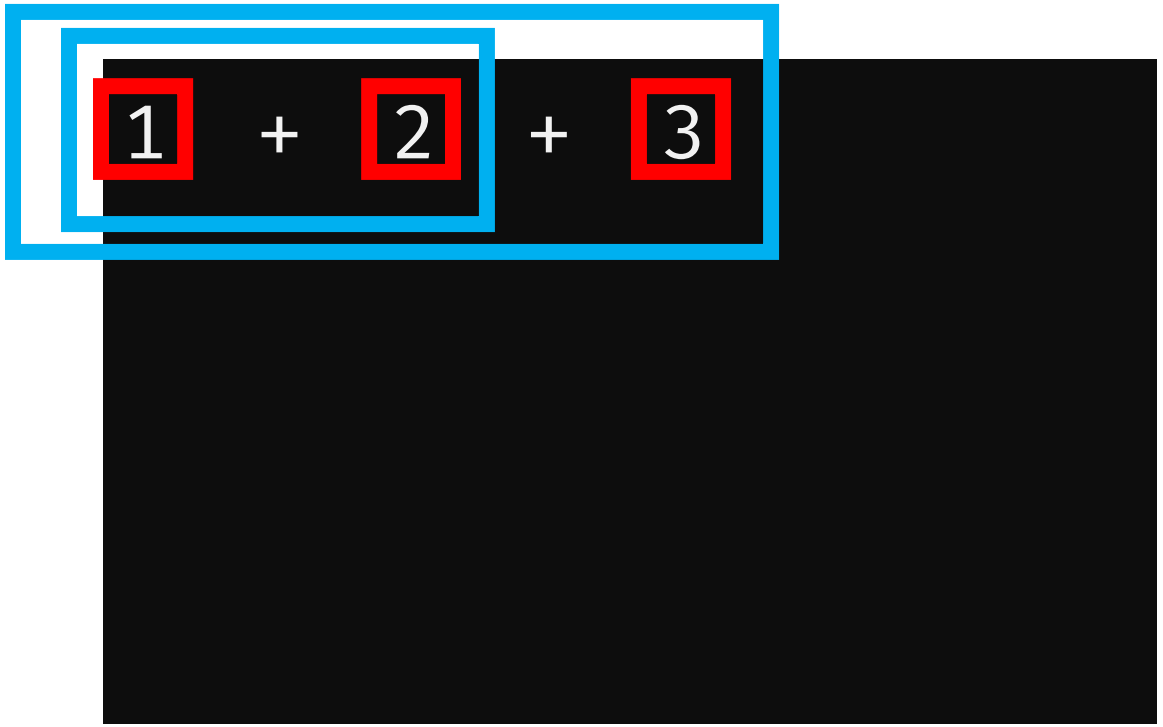


Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

```
sum:
    | sum '+' atom
    | atom

atom:
    | NUMBER
```



Source code: https://github.com/SebastiaanZ/pypethon

Python's Grammar & the PEG Parser

- We need to add a **grammar rule** for the pipe operator
- For that, we'll need to use **Parsing Expression Grammar (PEG)** syntax

```
10 |> double |> triple |> double |> print
```

Source code: https://github.com/SebastiaanZ/pypethon

Implementing our new grammar rule

File: Grammar/python.gram

```
shift_expr[expr_ty]:
    | a=shift_expr '<<' b=sum { _Py_BinOp(a, LShift, b, EXTRA) }
    | a=shift_expr '>>' b=sum { _Py_BinOp(a, RShift, b, EXTRA) }
    | sum

pipe[expr_ty]:
    | pipe '|>' sum
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
    | a=sum '-' b=term { _Py_BinOp(a, Sub, b, EXTRA) }
    | term
```

Implementing our new grammar rule

File: Grammar/python.gram

```
shift_expr[expr_ty]:
    | a=shift_expr '<<' b=sum { _Py_BinOp(a, LShift, b, EXTRA) }
    | a=shift_expr '>>' b=sum { _Py_BinOp(a, RShift, b, EXTRA) }
    | sum

pipe[expr_ty]:
    | pipe '|>' sum
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
    | a=sum '-' b=term { _Py_BinOp(a, Sub, b, EXTRA) }
    | term
```

Implementing our new grammar rule

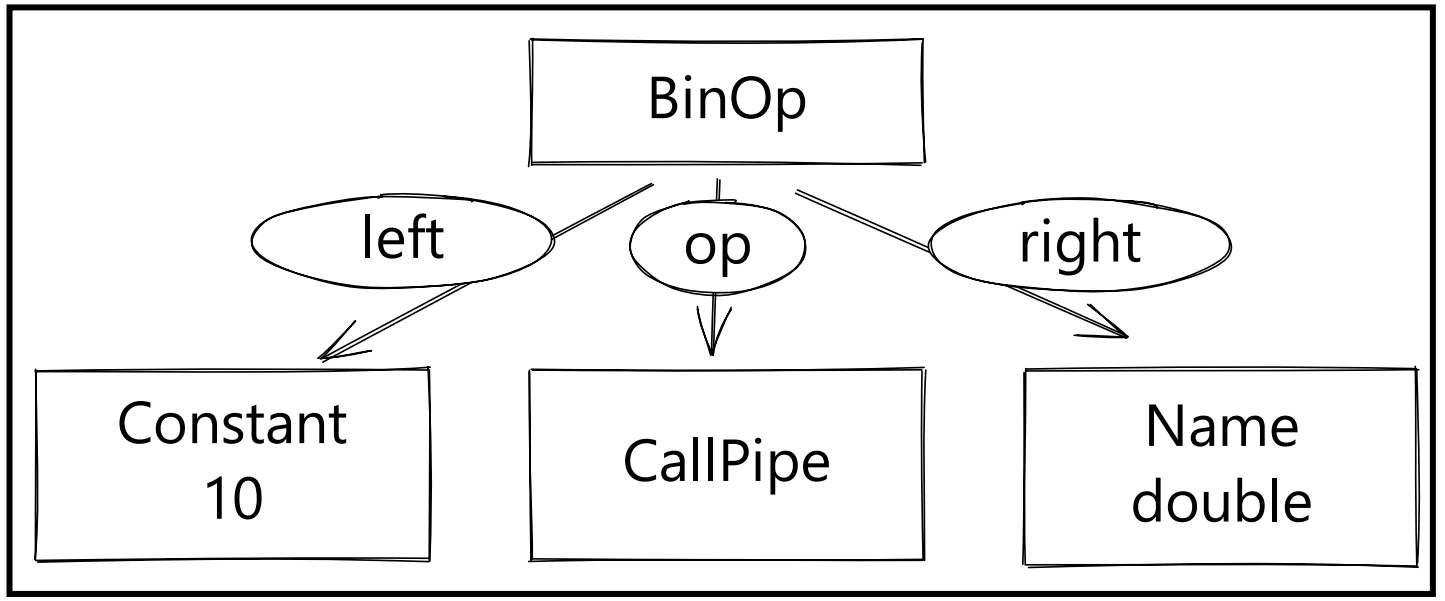
File: Grammar/python.gram

```
shift_expr[expr_ty]:
    | a=shift_expr '<<' b=pipe { _Py_BinOp(a, LShift, b, EXTRA) }
    | a=shift_expr '>>' b=pipe { _Py_BinOp(a, RShift, b, EXTRA) }
    | pipe

pipe[expr_ty]:
    | pipe '|>' sum
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
    | a=sum '-' b=term { _Py_BinOp(a, Sub, b, EXTRA) }
    | term
```

Grammar Actions: from a match to an AST node

```
10 |> double
```

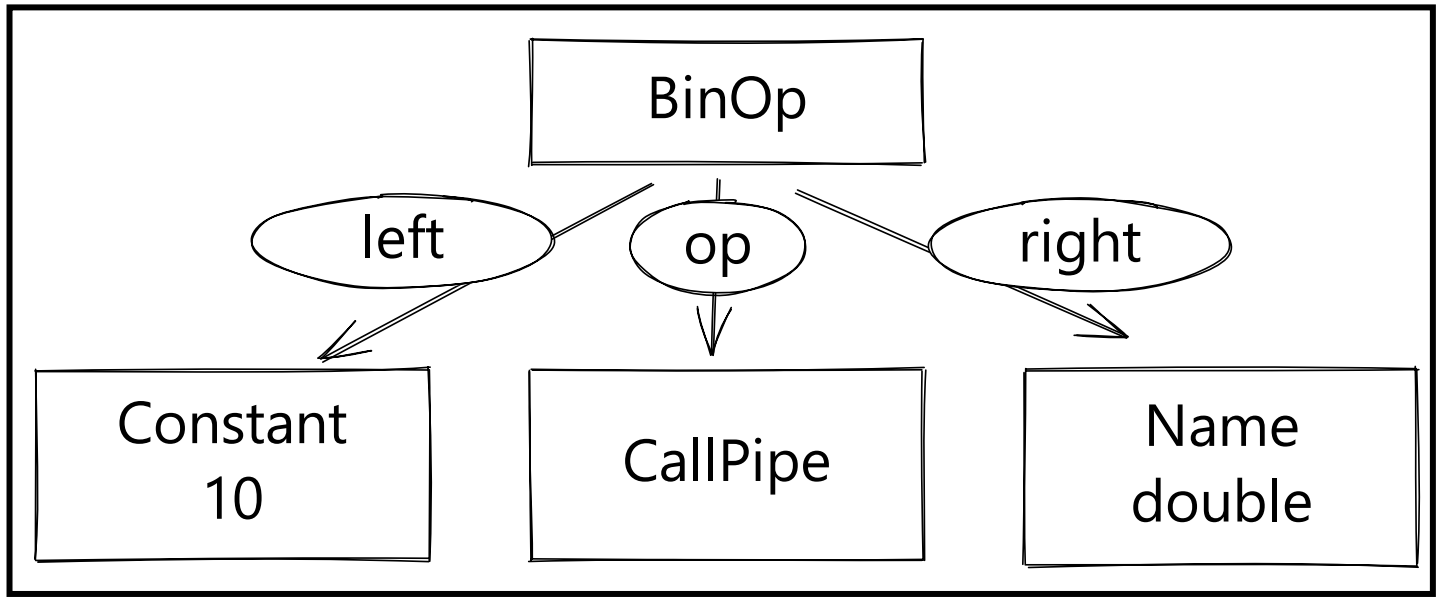
PARSED!



Source code: https://github.com/SebastiaanZ/pypethon

Grammar Actions: from a match to an AST node

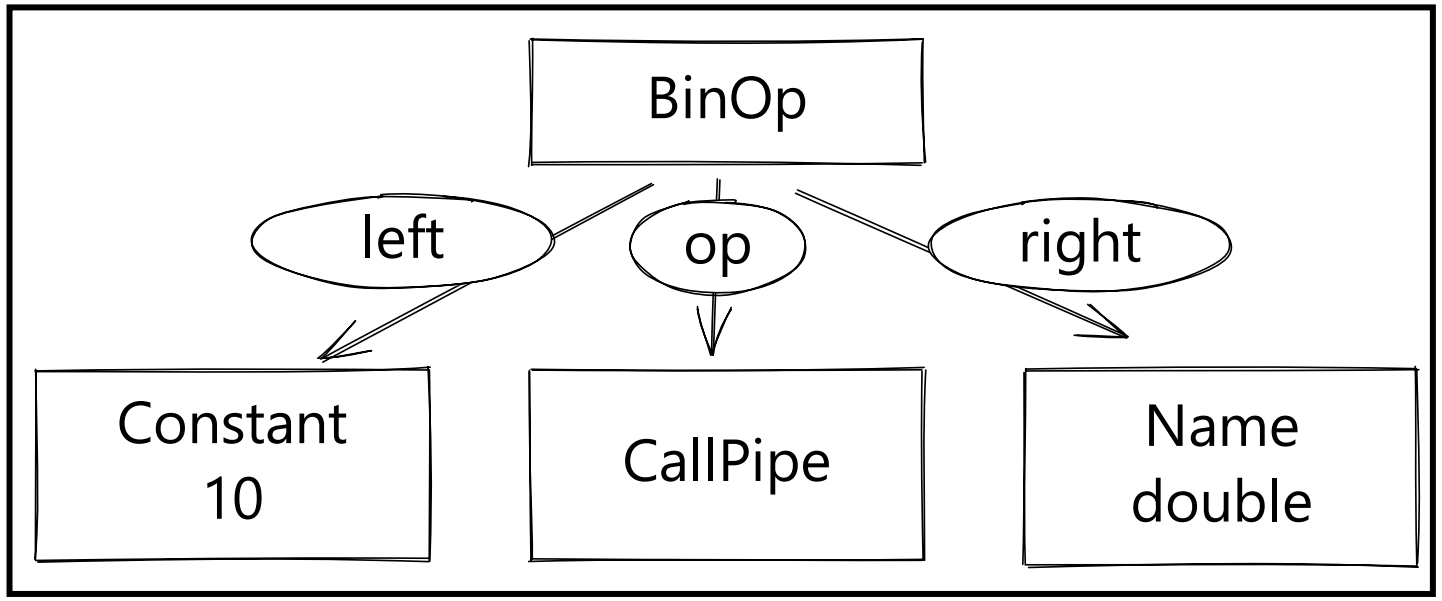
```
10 |> double
```




```
pipe[expr_ty]:
    | pipe '|>' sum
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
```

Grammar Actions: from a match to an AST node

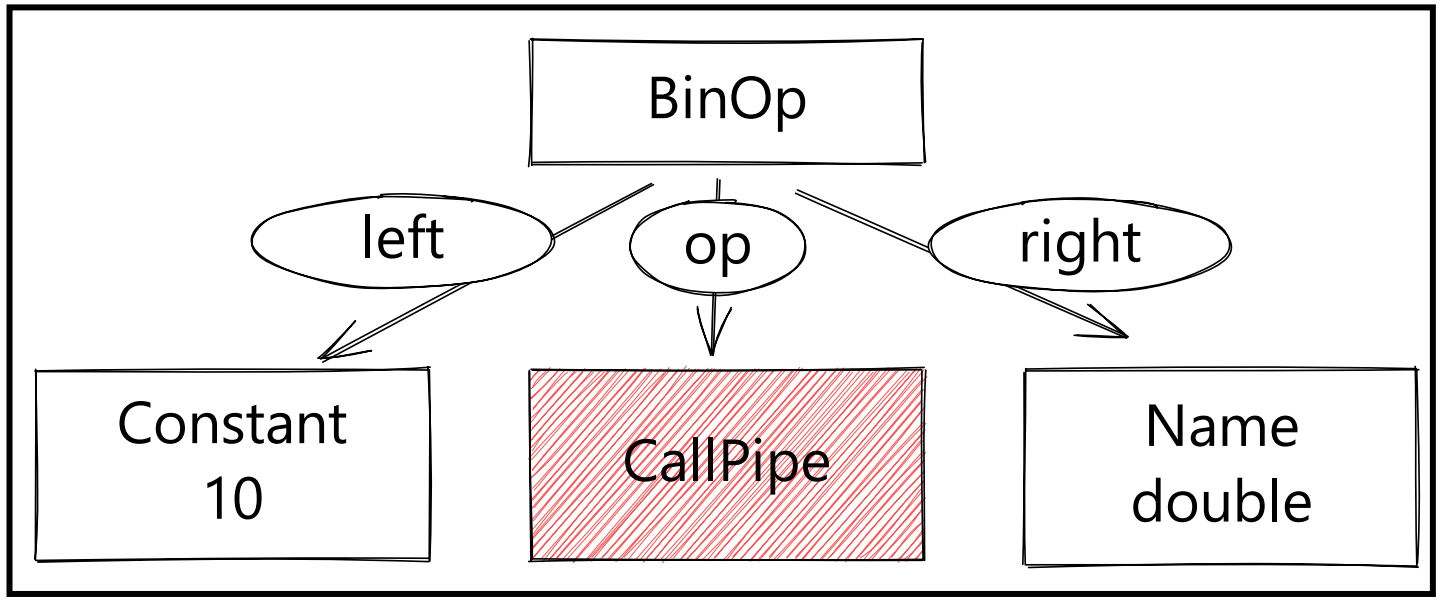
```
10 |> double
```




```
pipe[expr_ty]:
    | a=pipe '|>' b=sum { _Py_BinOp(a, CallPipe, b, EXTRA) }
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
```

Grammar Actions: from a match to an AST node

```
10 |> double
```




```
pipe[expr_ty]:
    | a=pipe '|>' b=sum { _Py_BinOp(a, CallPipe, b, EXTRA) }
    | sum

sum[expr_ty]:
    | a=sum '+' b=term { _Py_BinOp(a, Add, b, EXTRA) }
```

Add a CallPipe AST Node

File: Parser/Python.asdl

```
boolop = And | Or

operator = Add | Sub | Mult | MatMult | Div | Mod | Pow | LShift
             | RShift | BitOr | BitXor | BitAnd | FloorDiv

unaryop = Invert | Not | UAdd | USub
```

Source code: https://github.com/SebastiaanZ/pypethon

Add a CallPipe AST Node

File: Parser/Python.asdl

```
boolop = And | Or

operator = Add | Sub | Mult | MatMult | Div | Mod | Pow | LShift
             | RShift | BitOr | BitXor | BitAnd | FloorDiv | CallPipe

unaryop = Invert | Not | UAdd | USub
```

Linux/Mac

```
$ make regen-ast
```

Windows

```
> PCBuild\build.bat --regen
```

Source code: https://github.com/SebastiaanZ/pypethon

Regenerate the parser based on the new grammar

Linux/Mac

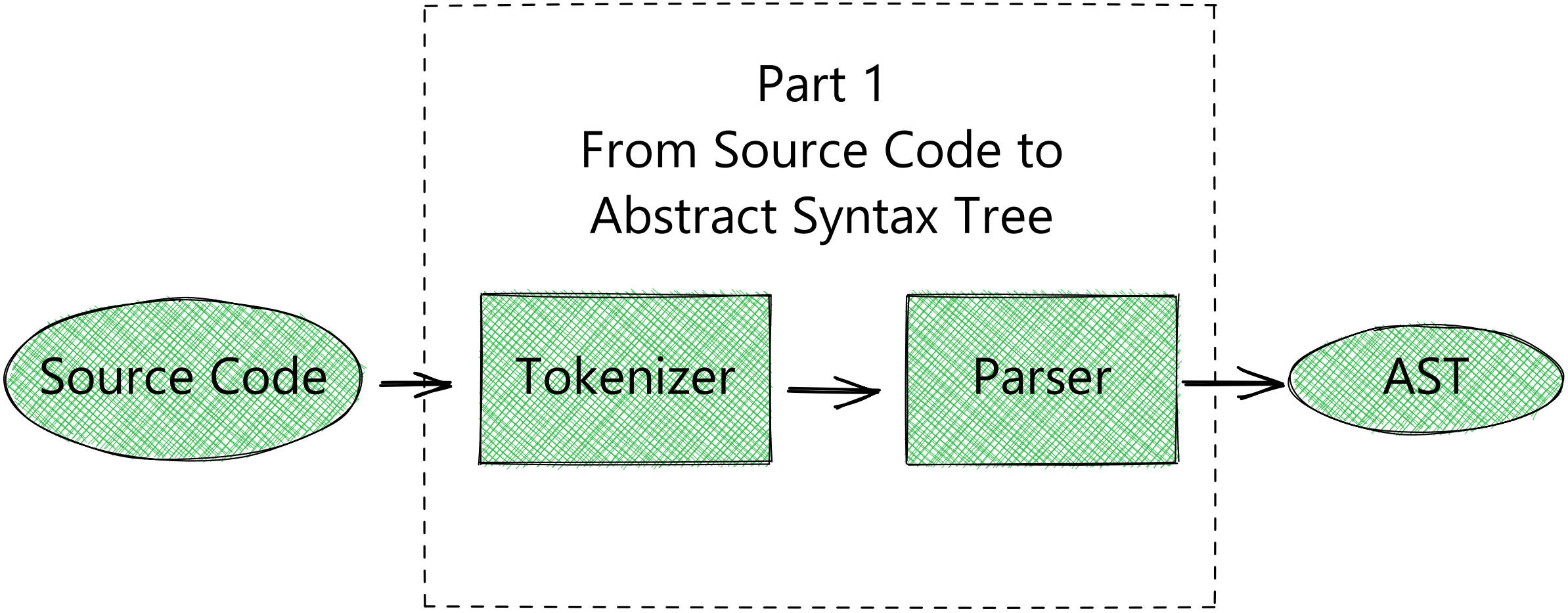
```
$ make regen-pegen
```

Windows

```
> PCBuild\build.bat --regen
```

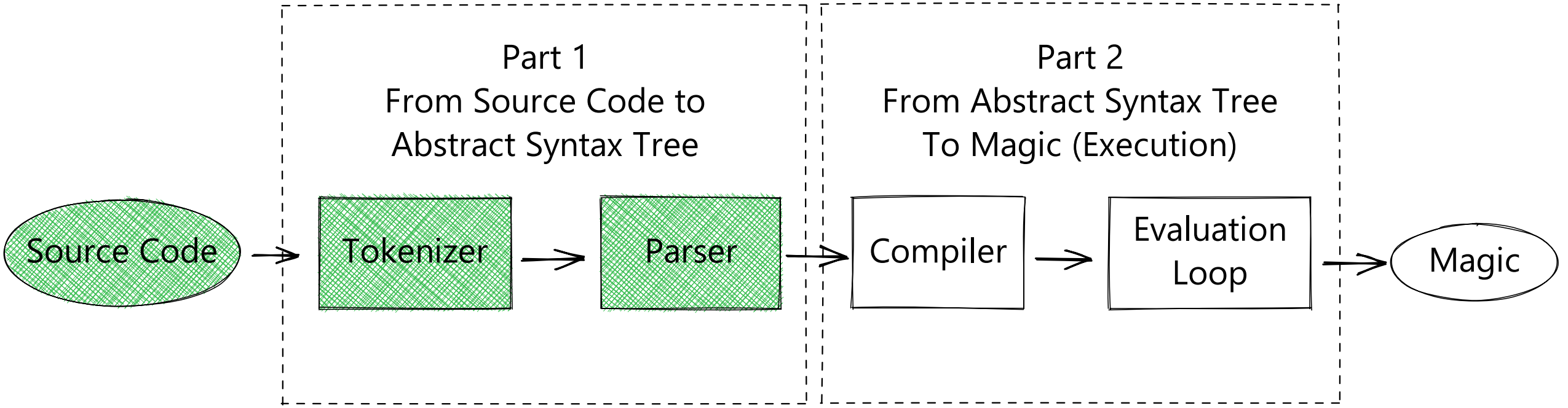
```
Python 3.9.9 (tags/v3.9.9-dirty:ccb0e6a345, Apr 10 2022, 13:28:32)
[GCC 9.4.0] on linux
>>> import ast
>>> tree = ast.parse("10 |> double")
>>> tree.body[0].value
<ast.BinOp object at 0x7f26f872beb0>
>>> tree.body[0].value.op
<ast.CallPipe object at 0x7f26f86542d0>
```

Part 1: Done



Source code: https://github.com/SebastiaanZ/pypethon

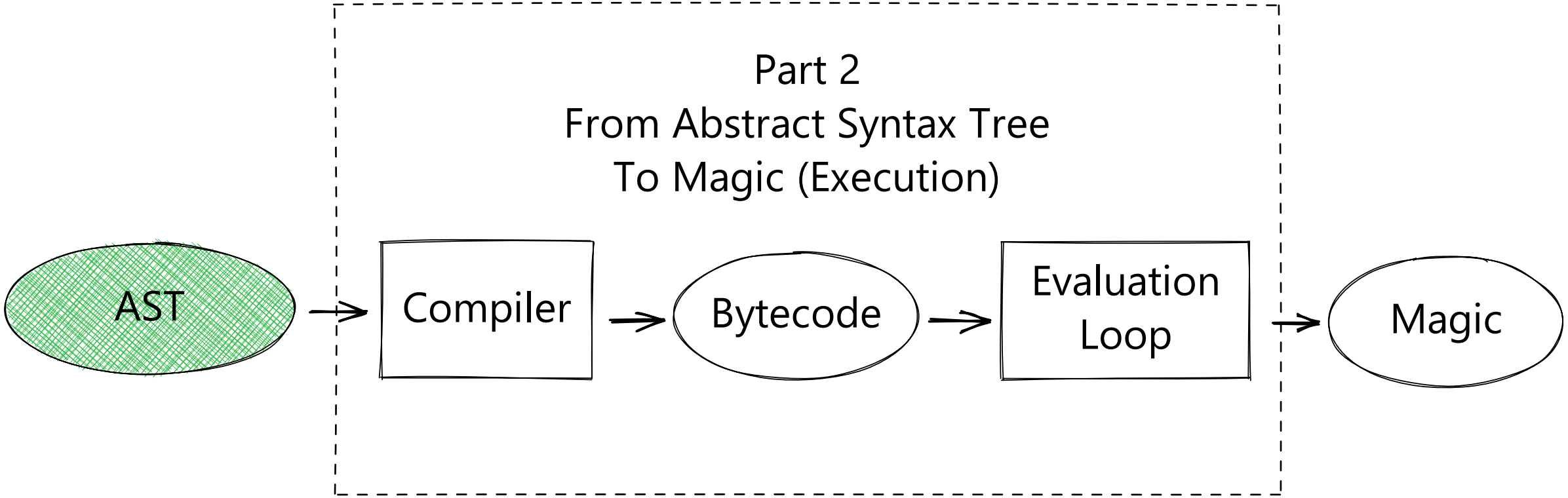
Progress



- Now it's time for Part 2, which is where the magic happens.

Source code: https://github.com/SebastiaanZ/pypethon

Part 2: From Abstract Syntax Tree to Magic (Execution)



Source code: https://github.com/SebastiaanZ/pypethon

Compiling the AST into Bytecode

- The compiler takes the AST and turns it into bytecode
- Bytecode consists of a list of **instructions** for the **evaluation loop**
- Each unique instruction has its own "opcode" (operation code)
- Let's add an opcode for our new operator[1]

1) We're ignoring the existing opcode for calling functions for didactic reason.

Source code: https://github.com/SebastiaanZ/pypethon

File: Lib/opcode.py

```
def_op('POP_EXCEPT', 89)



# Opcodes from here have an argument:
HAVE_ARGUMENT = 90
name_op('STORE_NAME', 90)
name_op('DELETE_NAME', 91)
```

Source code: https://github.com/SebastiaanZ/pypethon

File: Lib/opcode.py

```
def_op('POP_EXCEPT', 89)
def_op('BINARY_PIPE_CALL', 90)

# Opcodes from here have an argument:
HAVE_ARGUMENT = 91
name_op('STORE_NAME', 91)
name_op('DELETE_NAME', 92)
```

Linux/Mac

```
$ make regen-opcode
```

Windows

```
> PCBuild\build.bat
```

Making the compiler use the new "opcode"

File: Python/compile.c

```
static int
compiler_visit_expr1(struct compiler *c, expr_ty e)
{
    switch (e->kind) {
    /* Other cases removed */
    case BinOp_kind:
        VISIT(c, expr, e->v.BinOp.left);
        VISIT(c, expr, e->v.BinOp.right);
        ADDOP(c, binop(e->v.BinOp.op));
        break;
    /* Other cases removed */
```

Making the compiler use the new "opcode"

File: Python/compile.c

```
static int
compiler_visit_expr1(struct compiler *c, expr_ty e)
{
    switch (e->kind) {
    /* Other cases removed */
    case BinOp_kind:
        VISIT(c, expr, e->v.BinOp.left);
        VISIT(c, expr, e->v.BinOp.right);
        ADDOP(c, binop(e->v.BinOp.op));
        break;
    /* Other cases removed */
```

Making the compiler use the new "opcode"

File: Python/compile.c

```
static int
binop(operator_ty op)
{
    switch (op) {
    case Add:
        return BINARY_ADD;
    case Sub:
        return BINARY_SUBTRACT;
    /* And so on */
```

Making the compiler use the new "opcode"

File: Python/compile.c

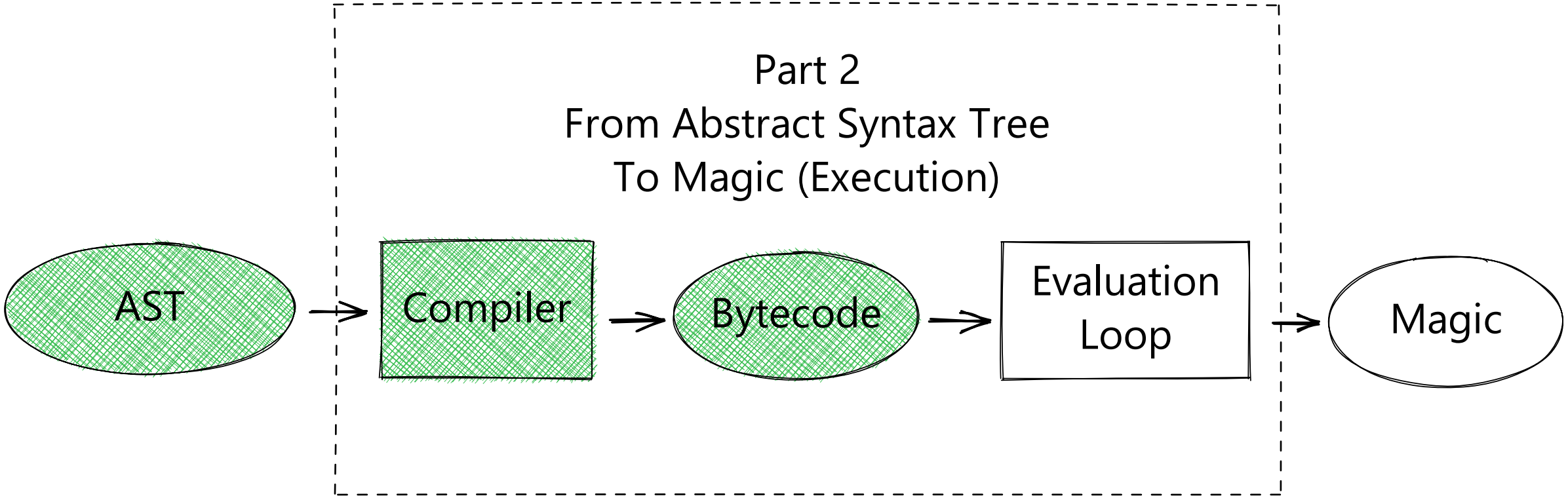
```
static int
binop(operator_ty op)
{
    switch (op) {
    case Add:
        return BINARY_ADD;
    case Sub:
        return BINARY_SUBTRACT;
    case CallPipe:
        return BINARY_PIPE_CALL;
    /* And so on */
```

Making the compiler use the new "opcode"

File: Python/compile.c

```
static int
stack_effect(int opcode, int oparg, int jump)
{
    switch (opcode) {
    /* Binary operators (most removed) */
    case BINARY_ADD:
    case BINARY_SUBTRACT:
    case BINARY_TRUE_DIVIDE:
    case BINARY_PIPE_CALL:
        return -1;
```

Part 2: We've got Bytecode!



- Now, we get to where the magic happens: the **Evaluation Loop**.

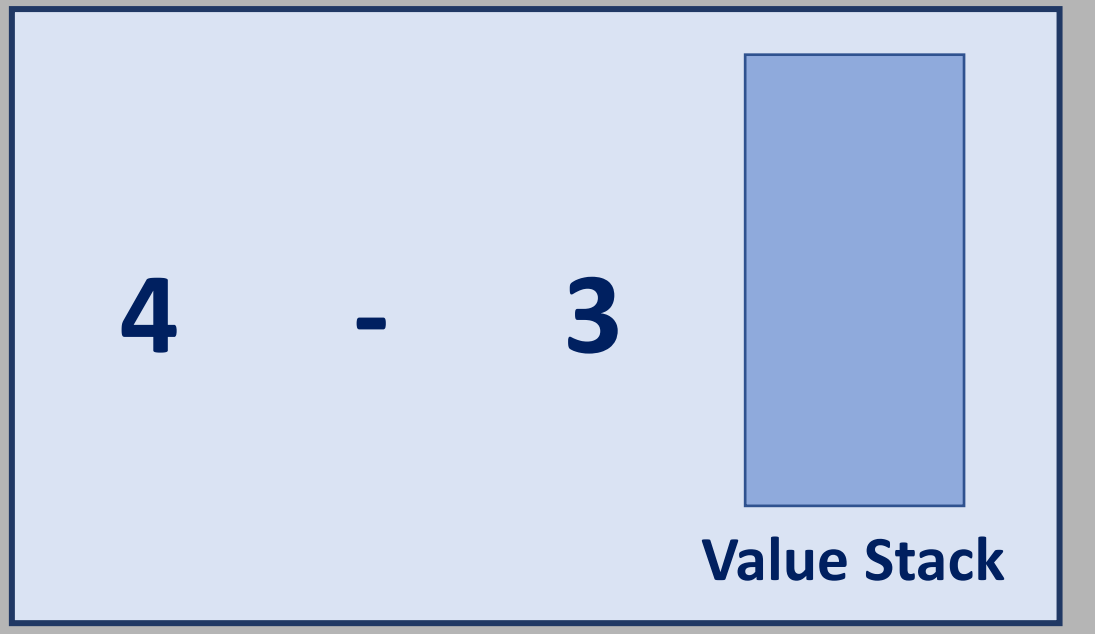
Source code: https://github.com/SebastiaanZ/pypethon

File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

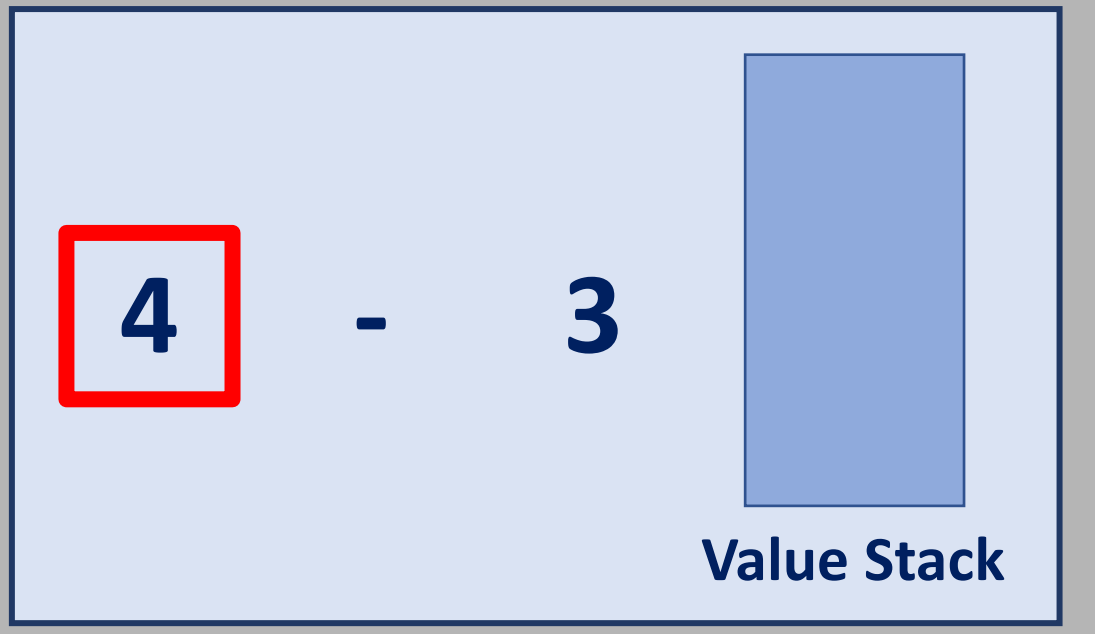
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



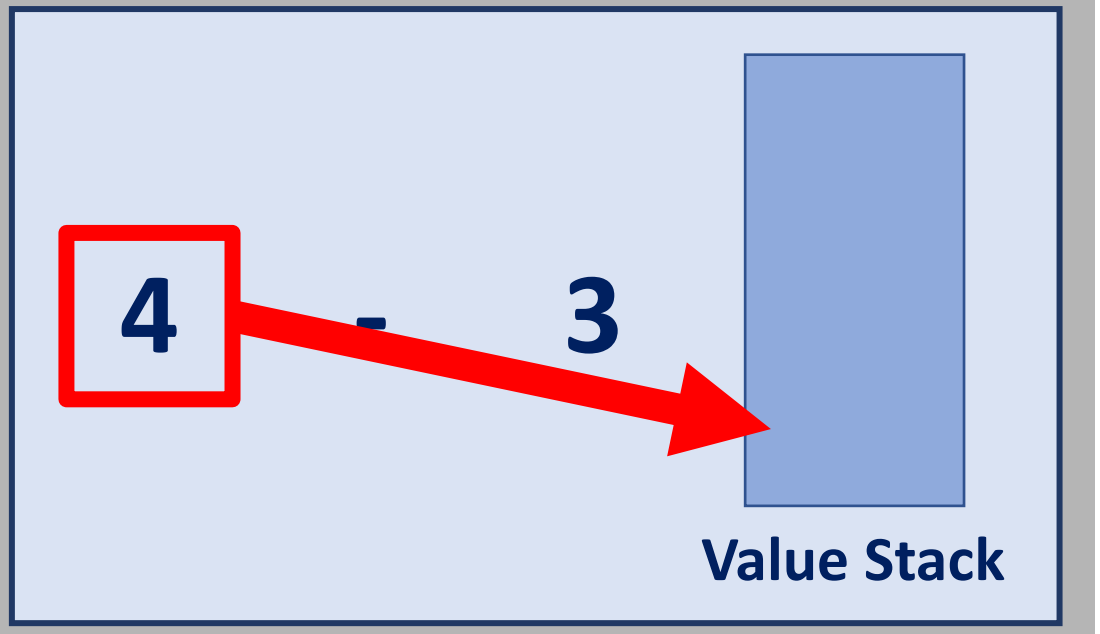
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



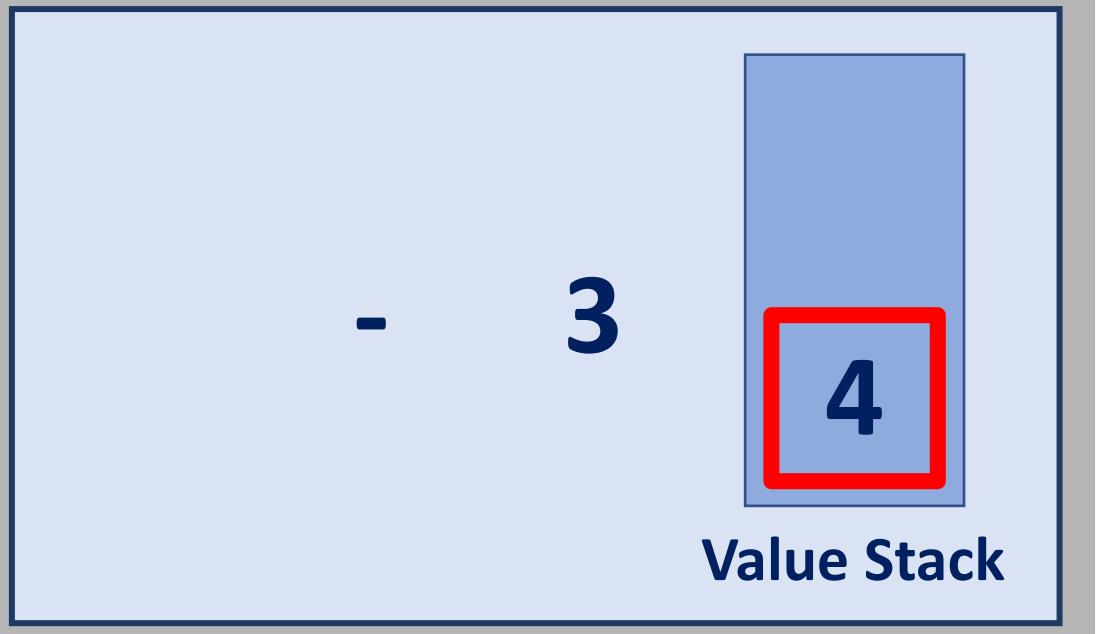
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



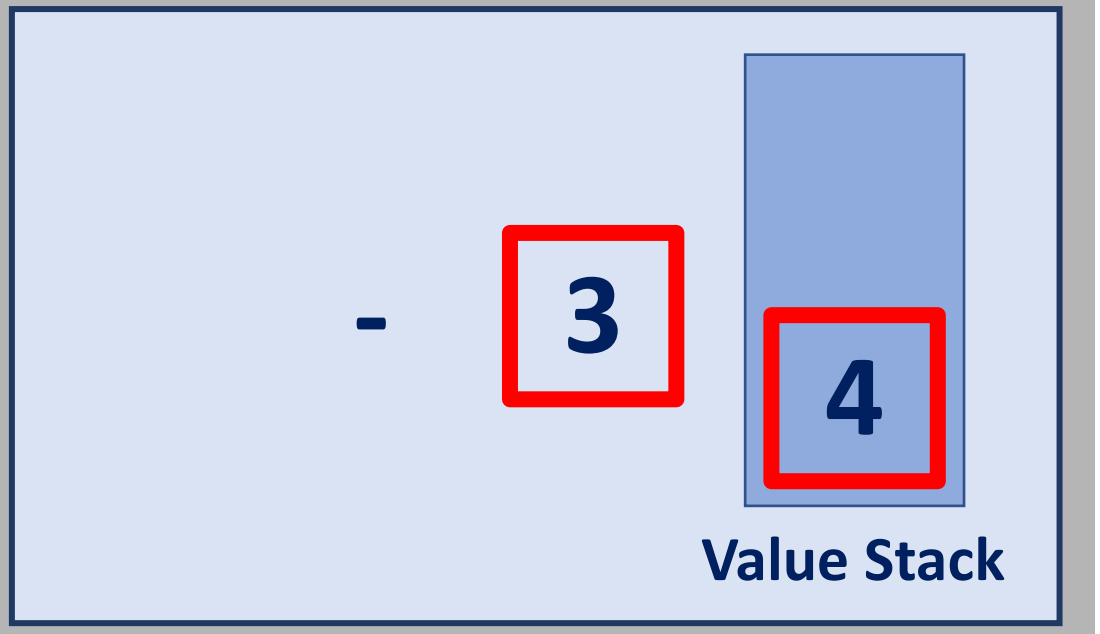
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



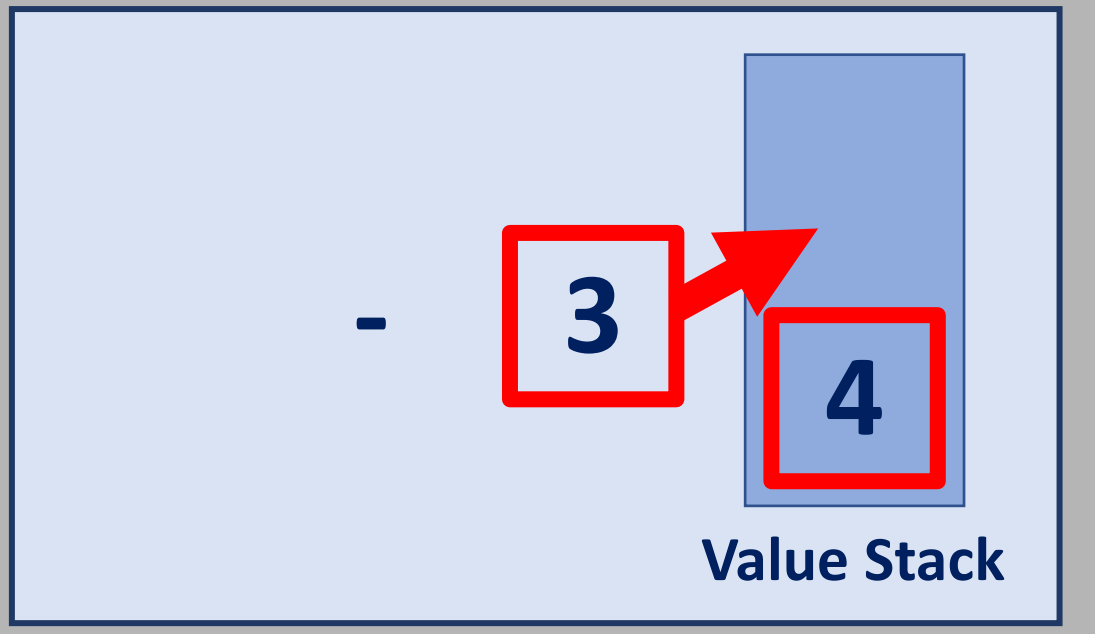
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



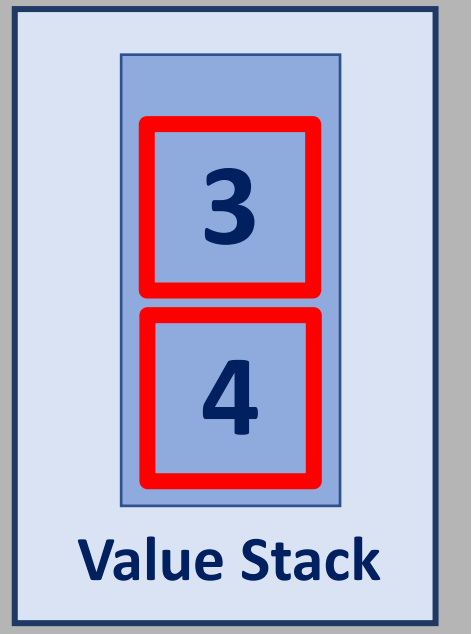
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



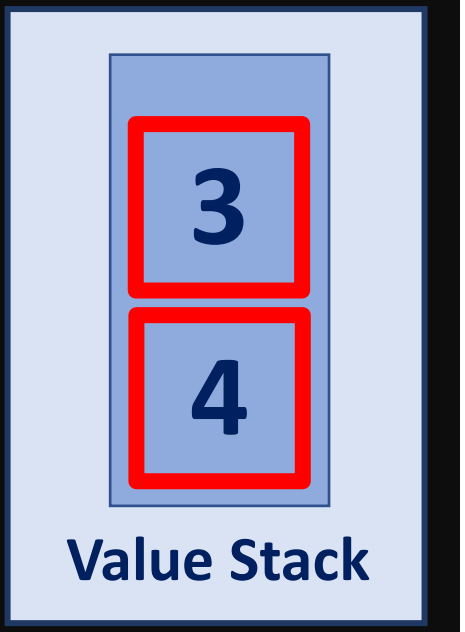
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

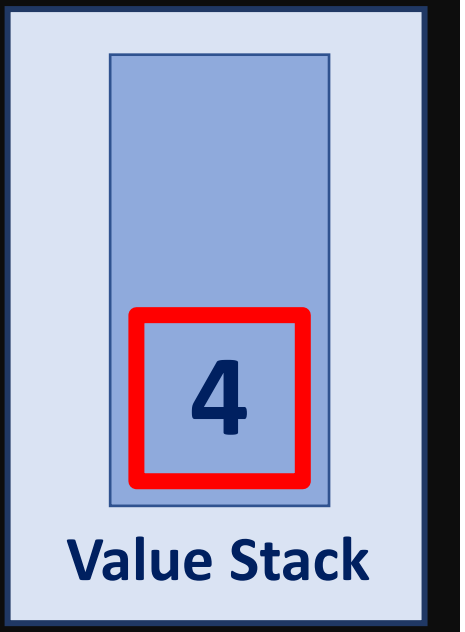


File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

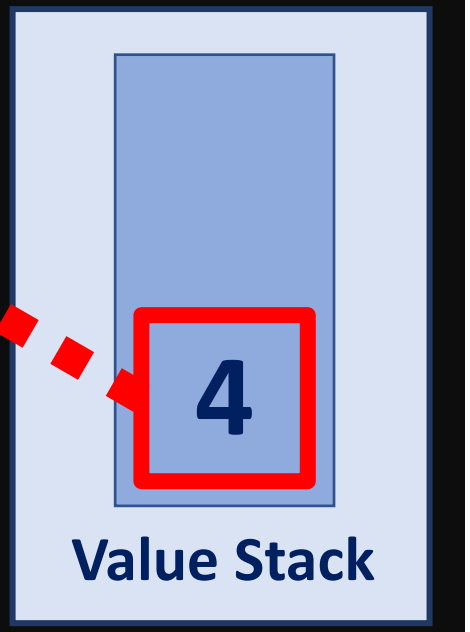
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



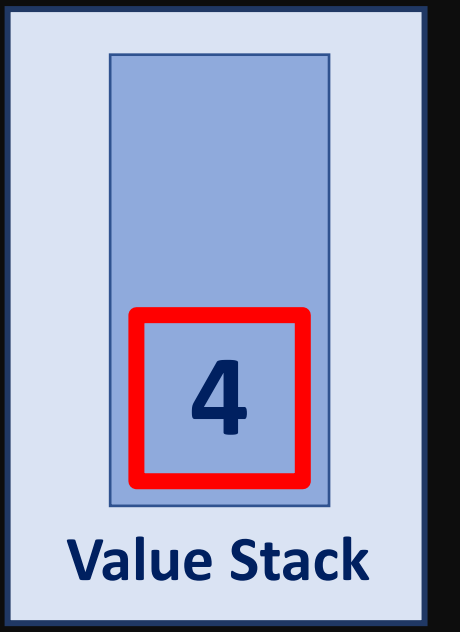
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



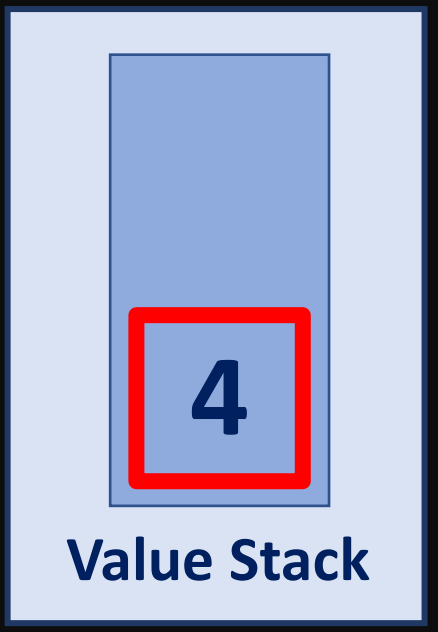
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = 3 P();
    PyObject *left = 4 P();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



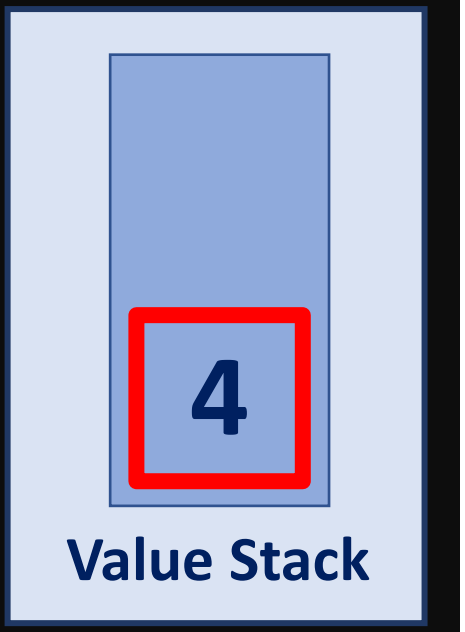
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



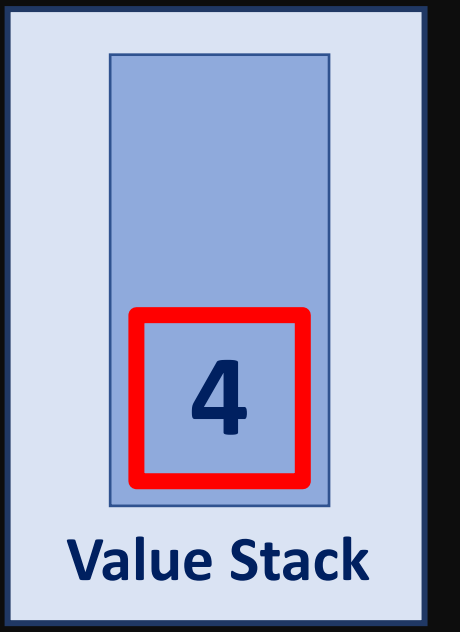
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



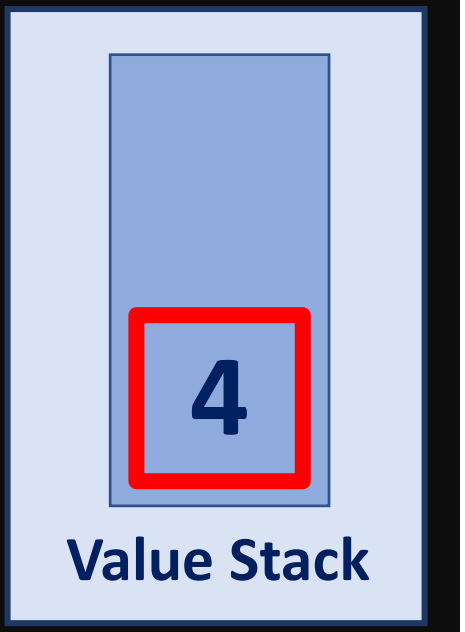
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

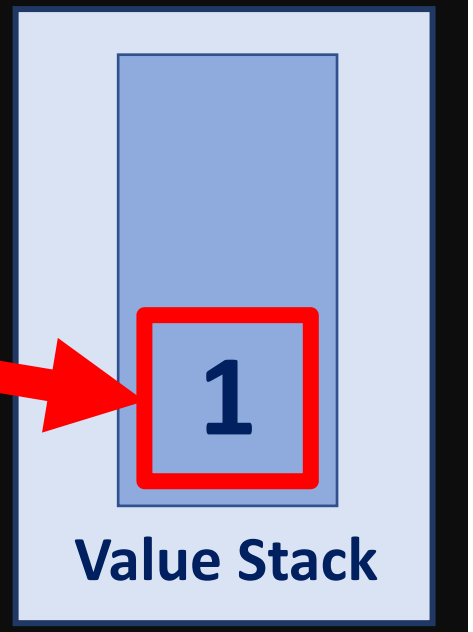


File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

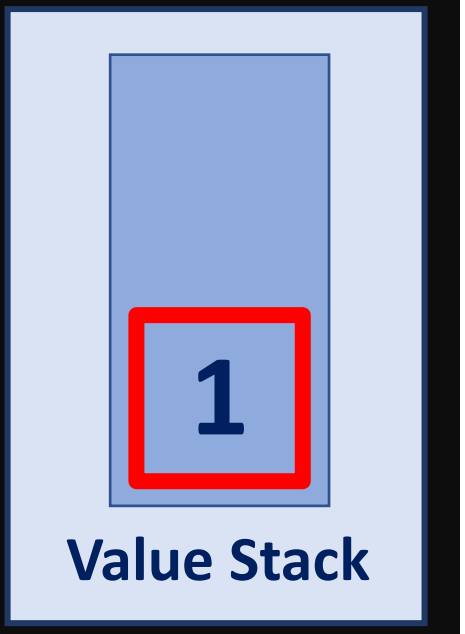
File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



File: Python/ceval.c

```
case TARGET(BINARY_SUBTRACT): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```



File: Python/ceval.c

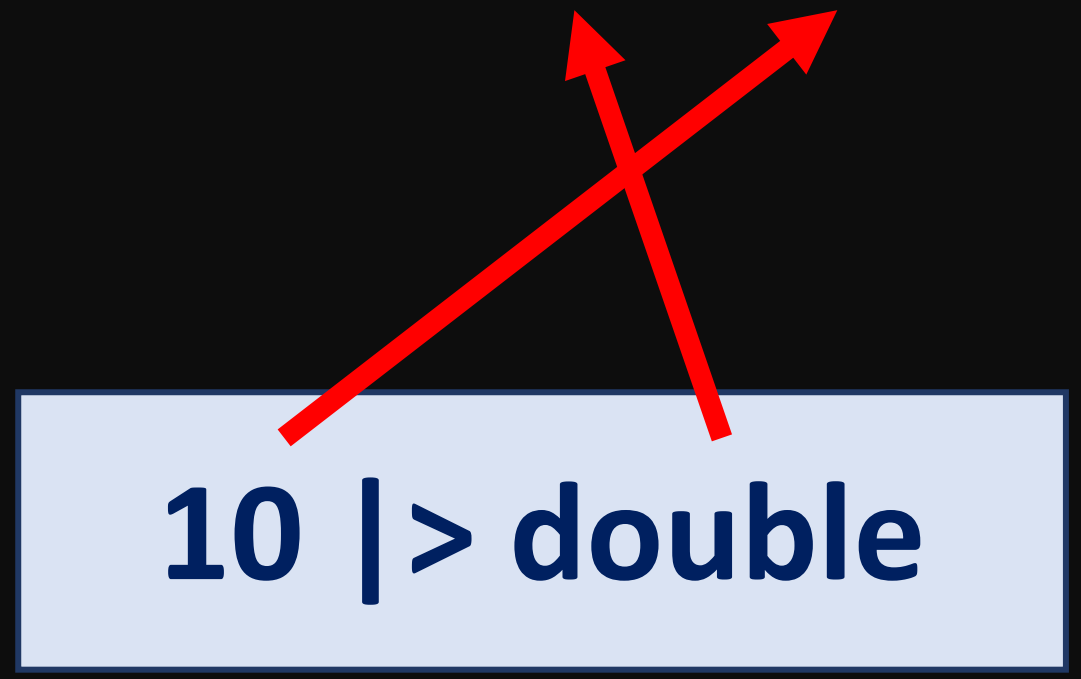
```
case TARGET(BINARY_PIPE_CALL): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Subtract(left, right);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

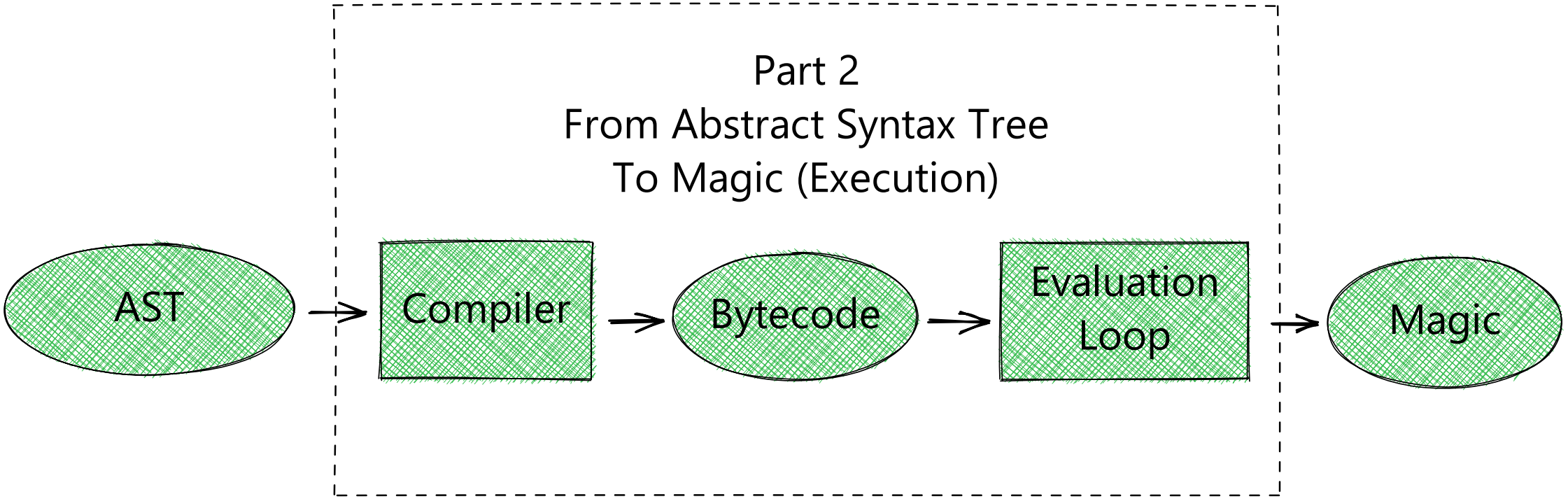
File: Python/ceval.c

```
case TARGET(BINARY_PIPE_CALL): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyObject_CallOneArg(right, left);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

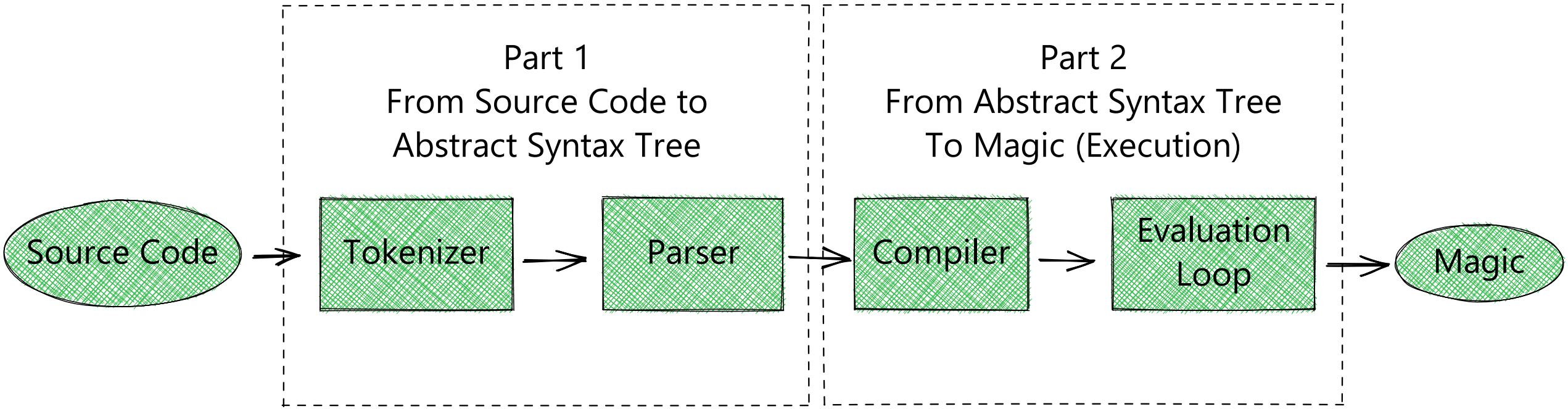
File: Python/ceval.c

```
case TARGET(BINARY_PIPE_CALL): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyObject_CallOneArg(right, left);
    Py_DECREF(right);
    Py_DECREF(left);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```





Source code: https://github.com/SebastiaanZ/pypethon



Source code: https://github.com/SebastiaanZ/pypethon

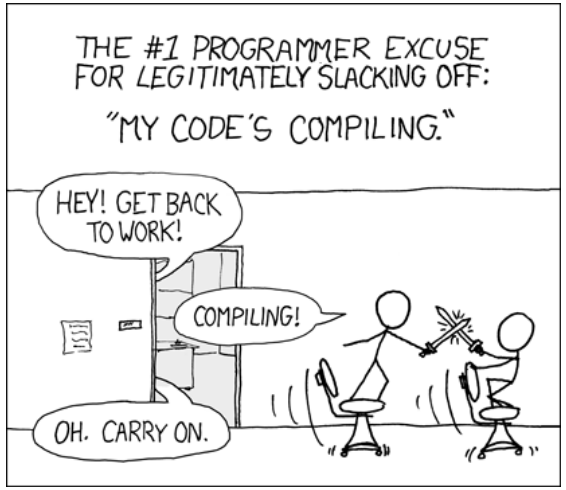
Let's compile and run it!

Linux/Mac

```
$ make -j2
```

Windows

```
> PCbuild\build.bat -d -p x64
```



https://xkcd.com/303/


```
Pypethon 3.9.9
>>> def double(number):
...     return number * 2
>>> 1 |> double |> double |> double |> double
16
```

Recap & Remarks

- We've seen a lot of CPython Internals in a short time
- Source code & slides are available online:
  - **https://github.com/SebastiaanZ/pypethon**
- If you get weird errors, try "clean" before giving up:
  - Run **make clean** or **PCBuild\build.bat -t CleanAll**
- See also the addendum about the MAGIC_NUMBER

Before we go...

https://europython.eu

Demystifying Python's Internals

Diving into CPython by implementing a pipe operator

Sebastiaan Zeeff @ PyCon US 2022

Get in touch with me:

- LinkedIn: https://www.linkedin.com/in/sebastiaanzeeff/
- Python Discord: https://discord.com/invite/python (Sebastiaan#0008)
- Twitter: https://twitter.com/SebastiaanZeeff

ORDINA
Ahead of change

Codesmiths
Ordina Software Development

Addendum: What about existing .pyc files?

- Old bytecode will no longer work, as opcodes have changed.
- To force recompilation, you can increase the "MAGIC_NUMBER"
  - Relevant file: **Lib/importlib/_bootstrap_external.py**
- You may also need to update "magic_values" in **PC/Launcher.c** to include the new value in the specified range.
- Run **make regen-importlib** or **PCBuild\build.bat --regen**

Source code: https://github.com/SebastiaanZ/pypethon