Contributing to CPython

Sadhana Srinivasan

What we'll cover

- What is CPython?
- How does CPython work?
- System specific compilation
- The contribution cycle, where to get help

What is CPython?

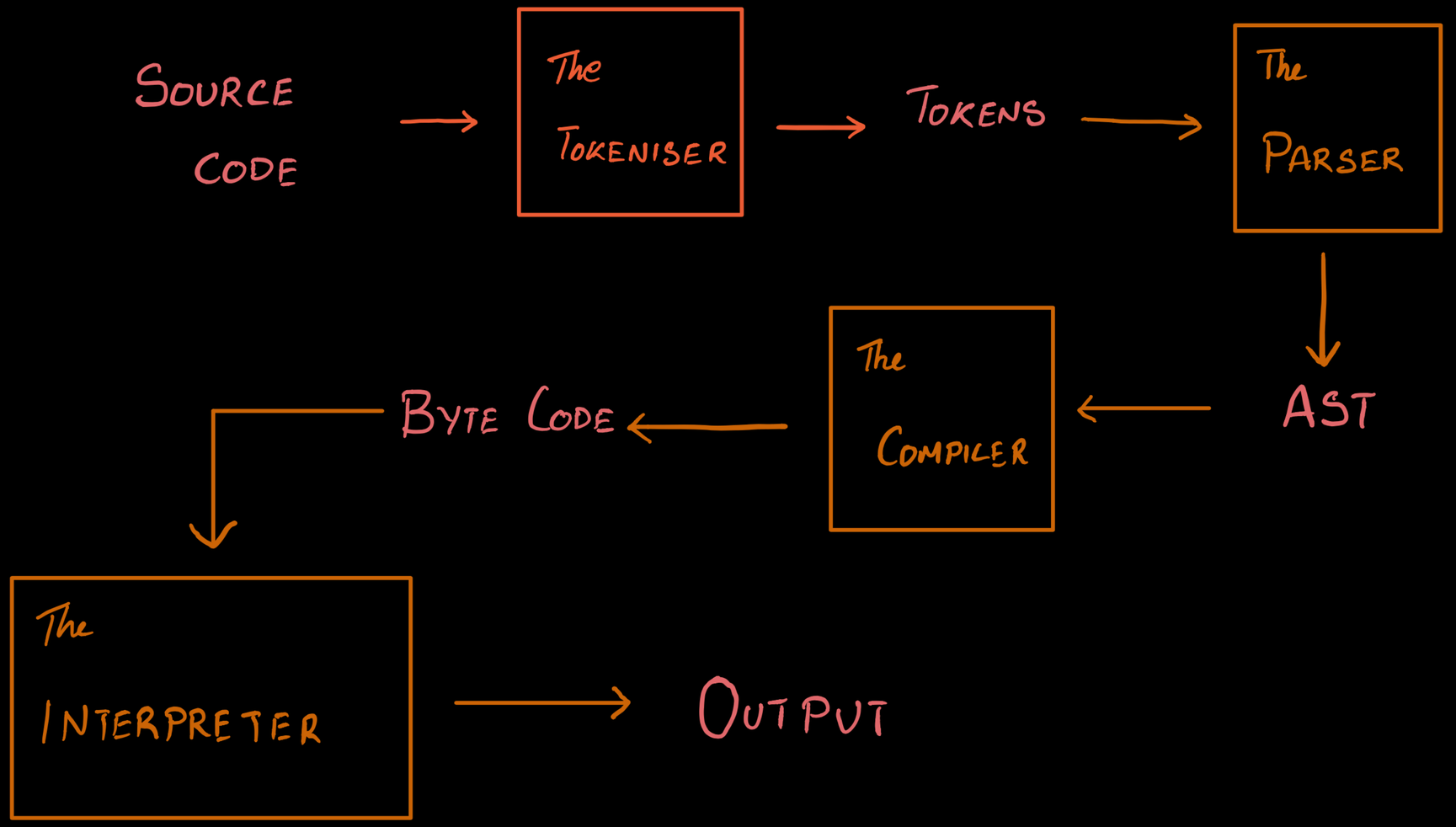
Python has a compiler

- .py files need to be translated to run on a system
- It has all the traditional components
- Python’s interpreter does most of the work

Alternatives to CPython

- Jython
  - Python on Java
- Iron Python
  - Python for the .NET Framework
- PyPy
  - Written in RPython
  - Just In Time Compilation
- Stackless
  - Microthreaded Python

How does CPython Work?


Source Code
The Tokeniser
Tokens
The Parser
AST
The Compiler
Byte Code
The Interpreter
Output

The Tokeniser

- Reads an input stream and finds the tokens
- Implemented as a list of rules
  - Is it a newline?
  - Is it a quote?
  - Could it be the start of an identifier?

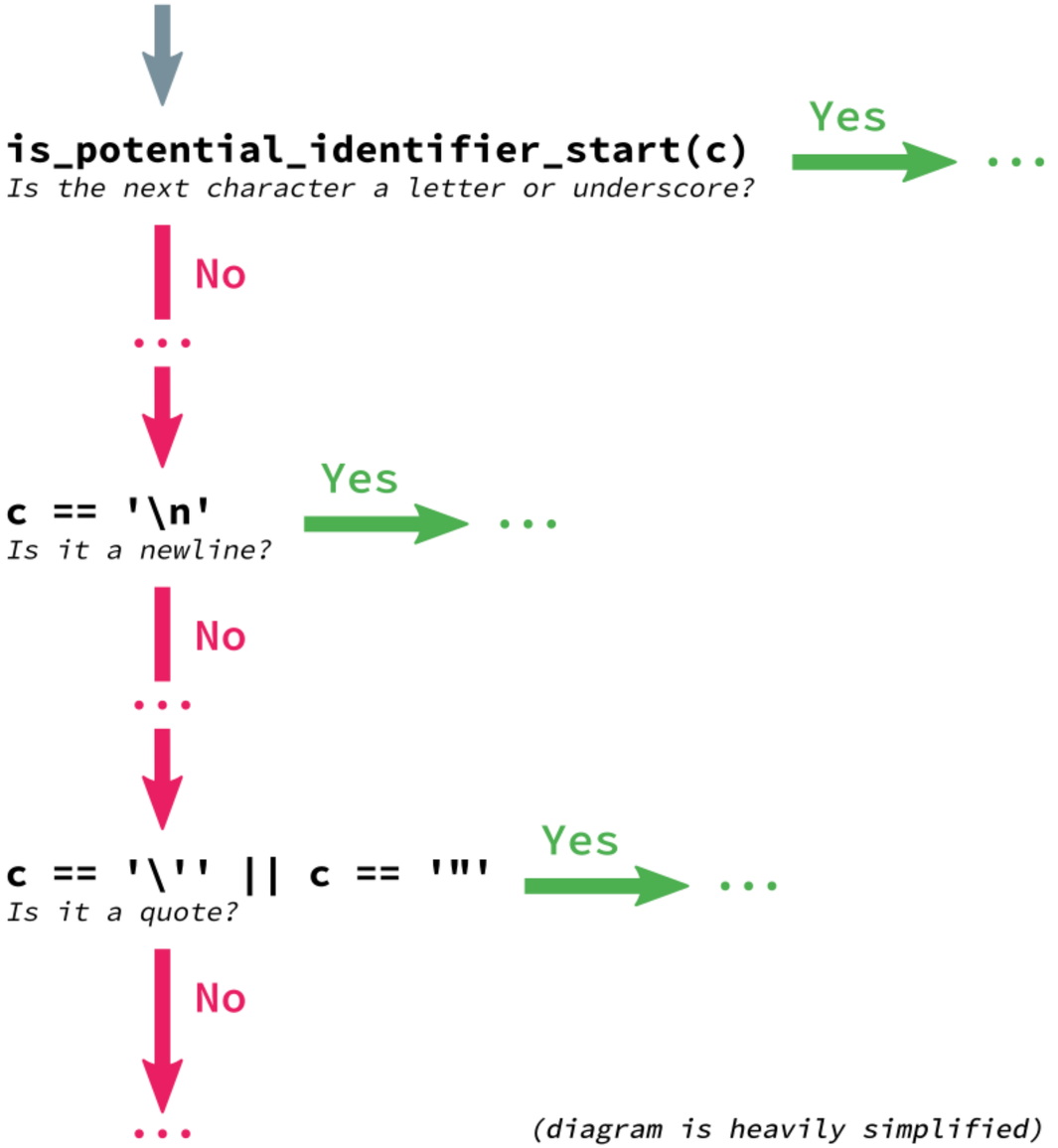


Image Source: https://benjam.info/blog/posts/2019-09-18-python-deep-dive-tokenizer/

b”abc”

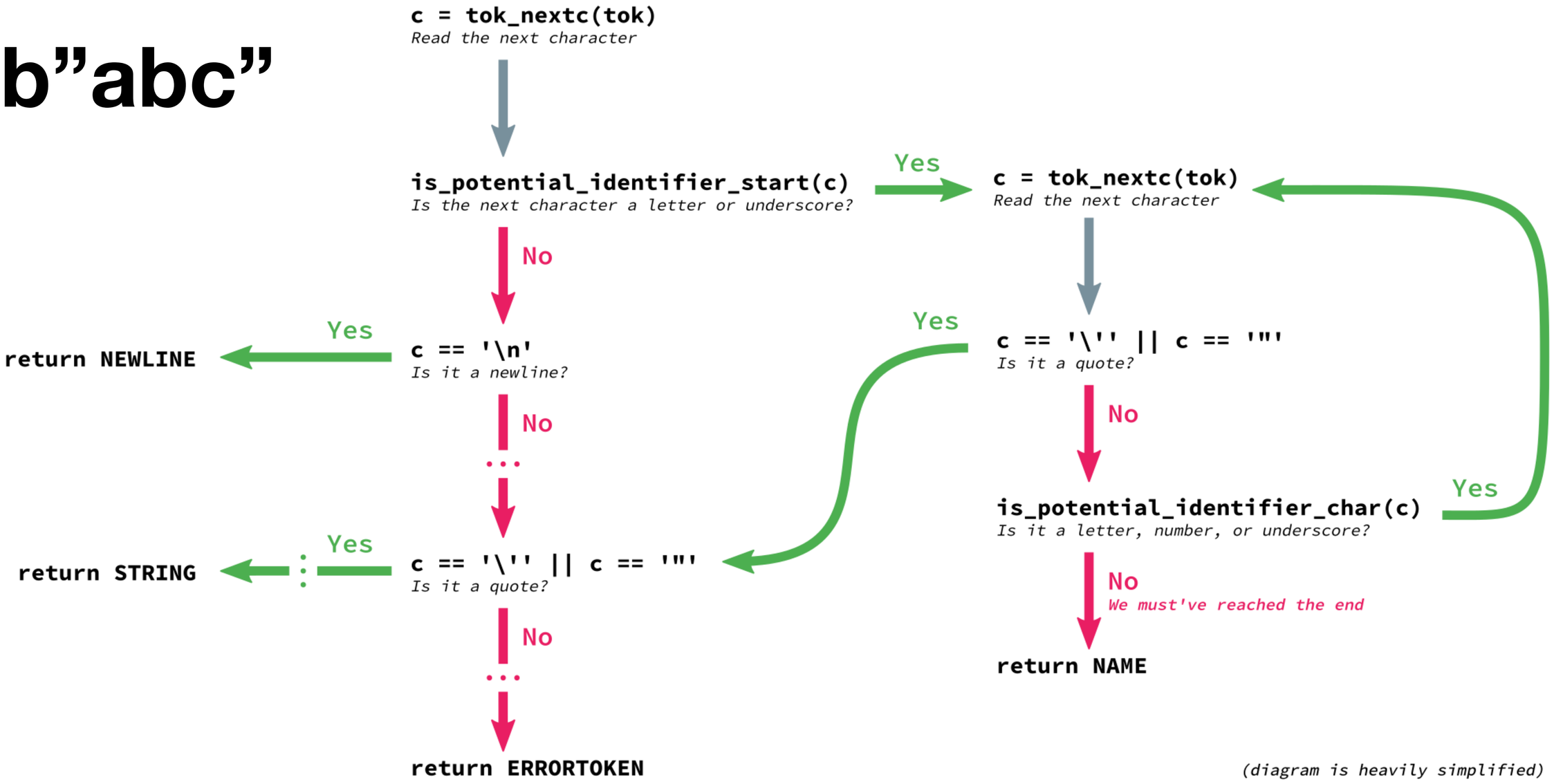


Image Source: https://benjam.info/blog/posts/2019-09-18-python-deep-dive-tokenizer/

The Parser

- The parser is generated from the grammar rules
- EBNF which includes ‘?’ ‘+’ ‘*’

The Old Parser

- LL(1) parser
- Had some ambiguity
- Generated a CST which was then converted to an AST by another program
- Without extra rules `2 + 2 = 5` was a valid statement

The PEG Parser

- No ambiguity
  - Rule: A | B | C
- Genrates an AST
- In Python as of 3.10

The Compiler

- Produces a Control Flow Graph
  - A Graph of byte code blocks where the edges jump to other blocks
- It does three things
  - Figure out the scope of each variable
  - Compile to byte code
  - Flatten CFG into a list and calculate the jumps

python -m dis <filename>

A Very Interesting Function

```
1 def func(x):
2     return x ** 2
3
4 x = 5
5 func(x)
```

```
  1           0 LOAD_CONST               0 (<code object func at 0x107104b30, file "talk.py", line 1>)
              2 LOAD_CONST               1 ('func')
              4 MAKE_FUNCTION            0
              6 STORE_NAME               0 (func)

  4           8 LOAD_CONST               2 (5)
             10 STORE_NAME               1 (x)

  5          12 LOAD_NAME                0 (func)
             14 LOAD_NAME                1 (x)
             16 CALL_FUNCTION            1
             18 POP_TOP
             20 LOAD_CONST               3 (None)
             22 RETURN_VALUE

Disassembly of <code object func at 0x107104b30, file "talk.py", line 1>:
  2           0 LOAD_FAST                0 (x)
              2 LOAD_CONST               1 (2)
              4 BINARY_POWER
              6 RETURN_VALUE
```

The Interpreter

- It’s one big infinite loop.*
- Works with OPCODEs
  - opcodes are defined in opcode.h

*** This statement will be modified soon**

```
1385
1386 main_loop:
1387     for (;;) {
1388         assert(stack_pointer >= f->f_valuestack); /* else underflow */
1389         assert(STACK_LEVEL() <= co->co_stacksize);  /* else overflow */
1390         assert(!_PyErr_Occurred(tstate));
```

ceval.c

```
1487        switch (opcode) {
1488
1489        /* BEWARE!
1490           It is essential that any operation that fails must goto error
1491           and that all operation that succeed call [FAST_]DISPATCH() ! */
1492
1493        case TARGET(NOP): {
1494           FAST_DISPATCH();
1495        }
1496
1497        case TARGET(LOAD_FAST): {
```

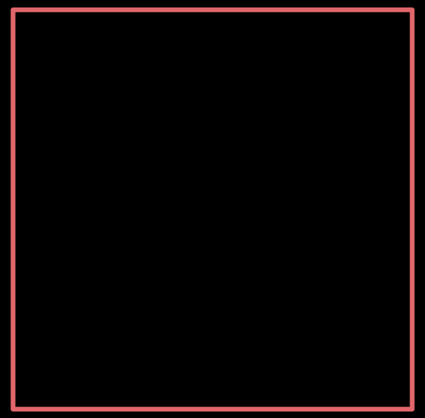
```
1497        case TARGET(LOAD_FAST): {
1498            PyObject *value = GETLOCAL(oparg);
1499            if (value == NULL) {
1500                format_exc_check_arg(tstate, PyExc_UnboundLocalError,
1501                                     UNBOUNDLOCAL_ERROR_MSG,
1502                                     PyTuple_GetItem(co->co_varnames, oparg));
1503                goto error;
1504            }
1505            Py_INCREF(value);
1506            PUSH(value);
1507            FAST_DISPATCH();
1508        }
```

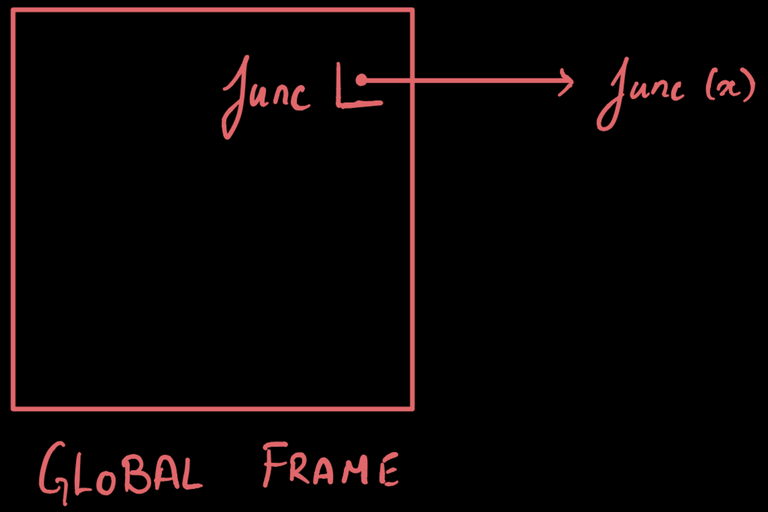
How does a Program run?

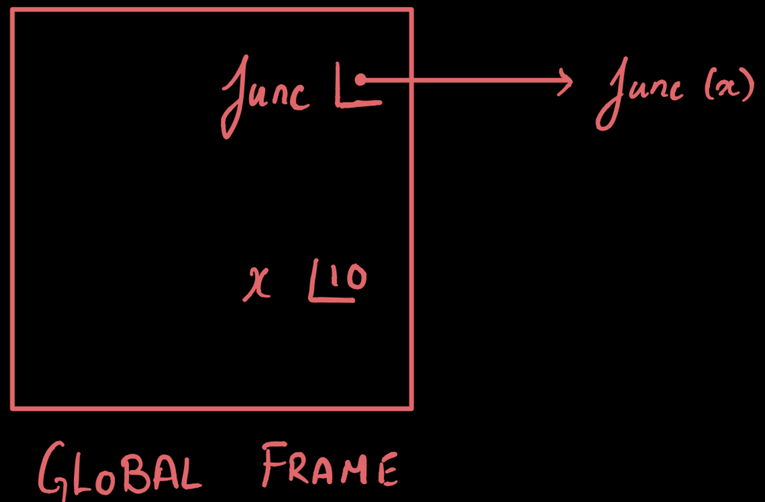
A Very Interesting Function

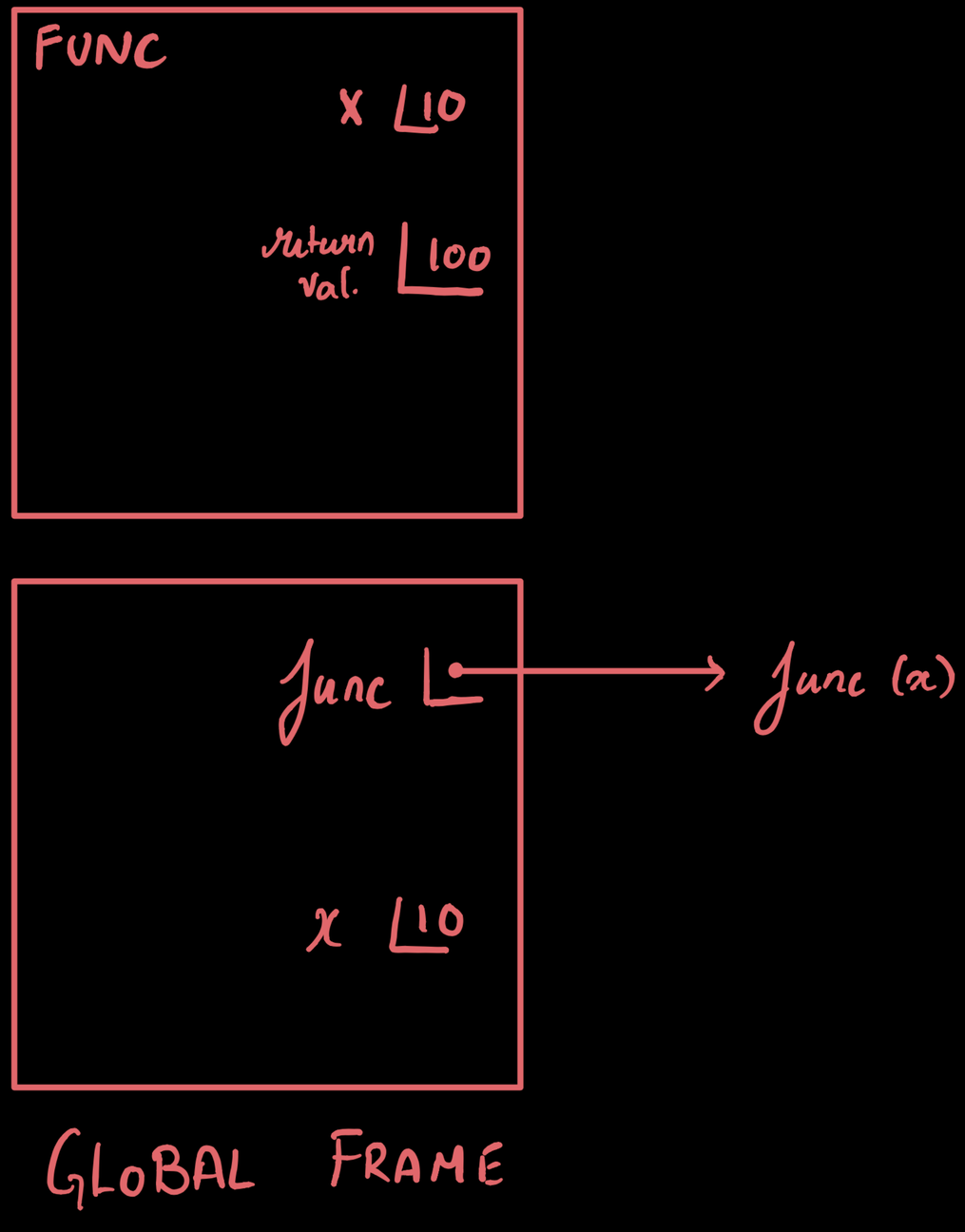
```
1 def func(x):
2     return x ** 2
3
4 x = 5
5 func(x)
```

```
876 PyObject *
877 PyEval_EvalFrameEx(PyFrameObject *f, int throwflag)
878 {
879     PyThreadState *tstate = _PyThreadState_GET();
880     return _PyEval_EvalFrame(tstate, f, throwflag);
881 }
```

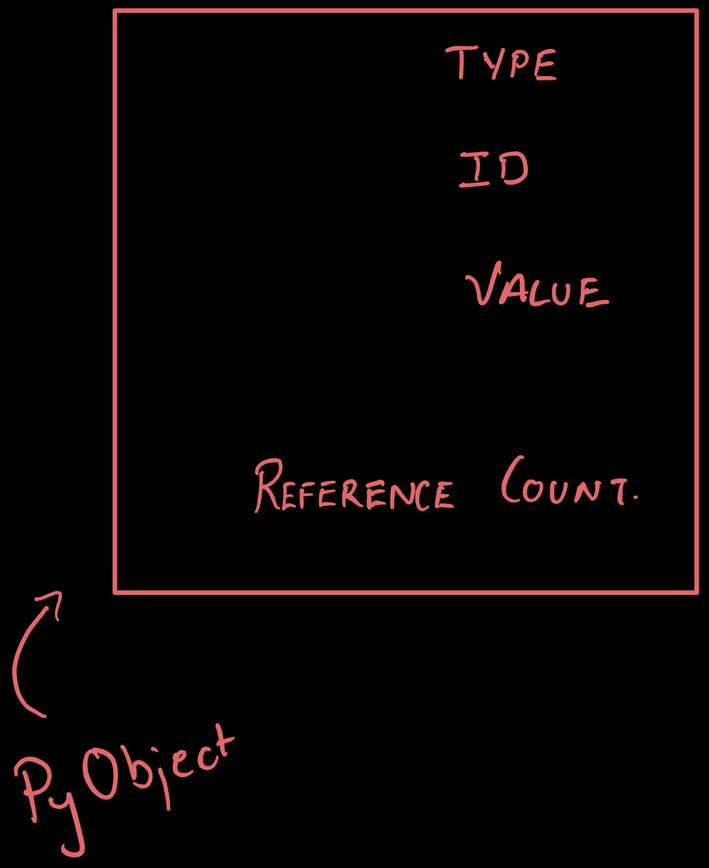

GLOBAL FRAME

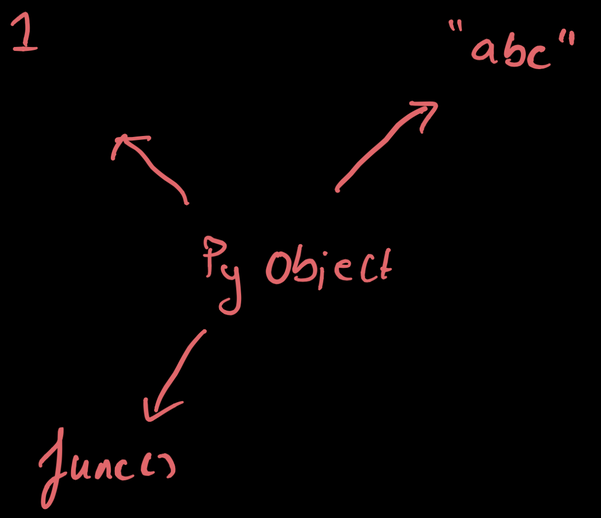

func
func (x)
GLOBAL FRAME


func L
func (x)
x 110
GLOBAL FRAME


FUNC
x 10
return val. 100
func 1
func (x)
x 10
GLOBAL FRAME

Python Data Model





Computed GOTOs

Having a big while loop with a switch case is not the most efficient

```
1052 #define FAST_DISPATCH() \
1053     { \
1054         if (!_Py_TracingPossible(ceval2) && !PyDTrace_LINE_ENABLED()) { \
1055             f->f_lasti = INSTR_OFFSET(); \
1056             NEXTOPARG(); \
1057             goto *opcode_targets[opcode]; \
1058         } \
1059         goto fast_next_opcode; \
```

```
1497        case TARGET(LOAD_FAST): {
1498            PyObject *value = GETLOCAL(oparg);
1499            if (value == NULL) {
1500                format_exc_check_arg(tstate, PyExc_UnboundLocalError,
1501                                     UNBOUNDLOCAL_ERROR_MSG,
1502                                     PyTuple_GetItem(co->co_varnames, oparg));
1503                goto error;
1504            }
1505            Py_INCREF(value);
1506            PUSH(value);
1507            FAST_DISPATCH();
1508        }
```

Compilation

```
1041 #ifdef LLTRACE
1042 #define FAST_DISPATCH() \
1043     { \
1044         if (!lltrace && !_Py_TracingPossible(ceval2) && !PyDTrace_LINE_ENABLED()) { \
1045             f->f_lasti = INSTR_OFFSET(); \
1046             NEXTOPARG(); \
1047             goto *opcode_targets[opcode]; \
1048         } \
1049         goto fast_next_opcode; \
1050     }
1051 #else
1052 #define FAST_DISPATCH() \
1053     { \
1054         if (!_Py_TracingPossible(ceval2) && !PyDTrace_LINE_ENABLED()) { \
1055             f->f_lasti = INSTR_OFFSET(); \
1056             NEXTOPARG(); \
1057             goto *opcode_targets[opcode]; \
1058         } \
1059         goto fast_next_opcode; \
1060     }
1061 #endif
```

```
1037 #define TARGET(op) \
1038     op: \
1039     TARGET_##op
```

```
1493         case TARGET(NOP): {
1494             FAST_DISPATCH();
1495         }
```

Contribution Process

The life of a PR

- Look at the python issues tracker (Not GitHub)
- Pick up an issue and tell people you'll work on it
- Solve it and send a PR with the correct title
- Update the issue tracker to notify people of your fix
- If it's accepted or not: Always look on the bright side of life!

Mailing Lists

- Python Ideas
- Python Dev
- Core Mentorship

PEP - 0

- A PEP to keep track or all PEPs

- Some upcoming changes (PEP 617, 570, 578)
  - PEG parser for Python
  - Positional Only Parameters
  - Python Runtime Audit Hooks

Thank You!

The Recursion Limit

```
743 #ifndef Py_DEFAULT_RECURSION_LIMIT
744 #define Py_DEFAULT_RECURSION_LIMIT 1000
745 #endif
```