Understanding PyPy

Francisco Fernandez Castano

Rushmore.fm

francisco.fernandez.castano@gmail.com *@fcofdezc*

November 8, 2014

Overview

First PyPy impressions

First PyPy impressions

```
Python 2.7.3 (87aa9de10f9ca71da9ab4a3d53e0ba176b67d0
[PyPy 2.2.1 with GCC 4.8.3 20140624 (Red Hat 4.8.3-
Type "help", "copyright", "credits" or "license" fo
>>>> 1 + 1
2
>>>> def f(x): return x + 1
>>>> f(1)
2
>>>>
```

First PyPy impressions

Ok... just Python...

Second PyPy impressions

europython 2014
berlin 21–27 july

What is PyPy?

PyPy is a fast, compliant alternative implementation of the Python language (2.7.8 and 3.2.5).

Why is Python slow?

- Interpretation overhead
- Boxed arithmetic and automatic overflow handling
- Dynamic dispatch of operations
- Dynamic lookup of methods and attributes
- Everything can change on runtime
- Extreme introspective and reflective capabilities

Why is Python slow?

Boxed arithmetic and automatic overflow handling

```
i = 0
while i < 10000000:
    i = i +1
```

Why is Python slow?

Dynamic dispatch of operations

```
# while i < 1000000
9  LOAD_FAST                0 (i)
12 LOAD_CONST               2 (10000000)
15 COMPARE_OP               0 (<)
18 POP_JUMP_IF_FALSE        34

# i = i + 1
21 LOAD_FAST                0 (i)
24 LOAD_CONST               3 (1)
27 BINARY_ADD
28 STORE_FAST               0 (i)
31 JUMP_ABSOLUTE            9
```

Why is Python slow?

Dynamic lookup of methods and attributes

```
class MyExample(object):
    pass

def foo(target, flag):
    if flag:
        target.x = 42

obj = MyExample()
foo(obj, True)
print obj.x #=> 42
print getattr(obj, "x") #=> 42
```

Why is Python slow?

Everything can change on runtime

```
def fn():
    return 42

def hello():
    return 'Hi! PyConEs!'

def change_the_world():
    global fn
    fn = hello

print fn() #=> 42
change_the_world()
print fn() => 'Hi! PyConEs!'
```

Why is Python slow?

Everything can change on runtime

```
class Dog(object):
    def __init__(self):
        self.name = 'Jandemor'

    def talk(self):
        print "%s: guau!" % self.name

class Cat(object):
    def __init__(self):
        self.name = 'CatInstance'

    def talk(self):
        print "%s: miau!" % self.name
```

Why is Python slow?

Everything can change on runtime

```
my_pet = Dog()
my_pet.talk() #=> 'Jandemor: guau!'
my_pet.__class__ = Cat
my_pet.talk() #=> 'Jandemor: miau!'
```

Why is Python slow?

Extreme introspective and reflective capabilities

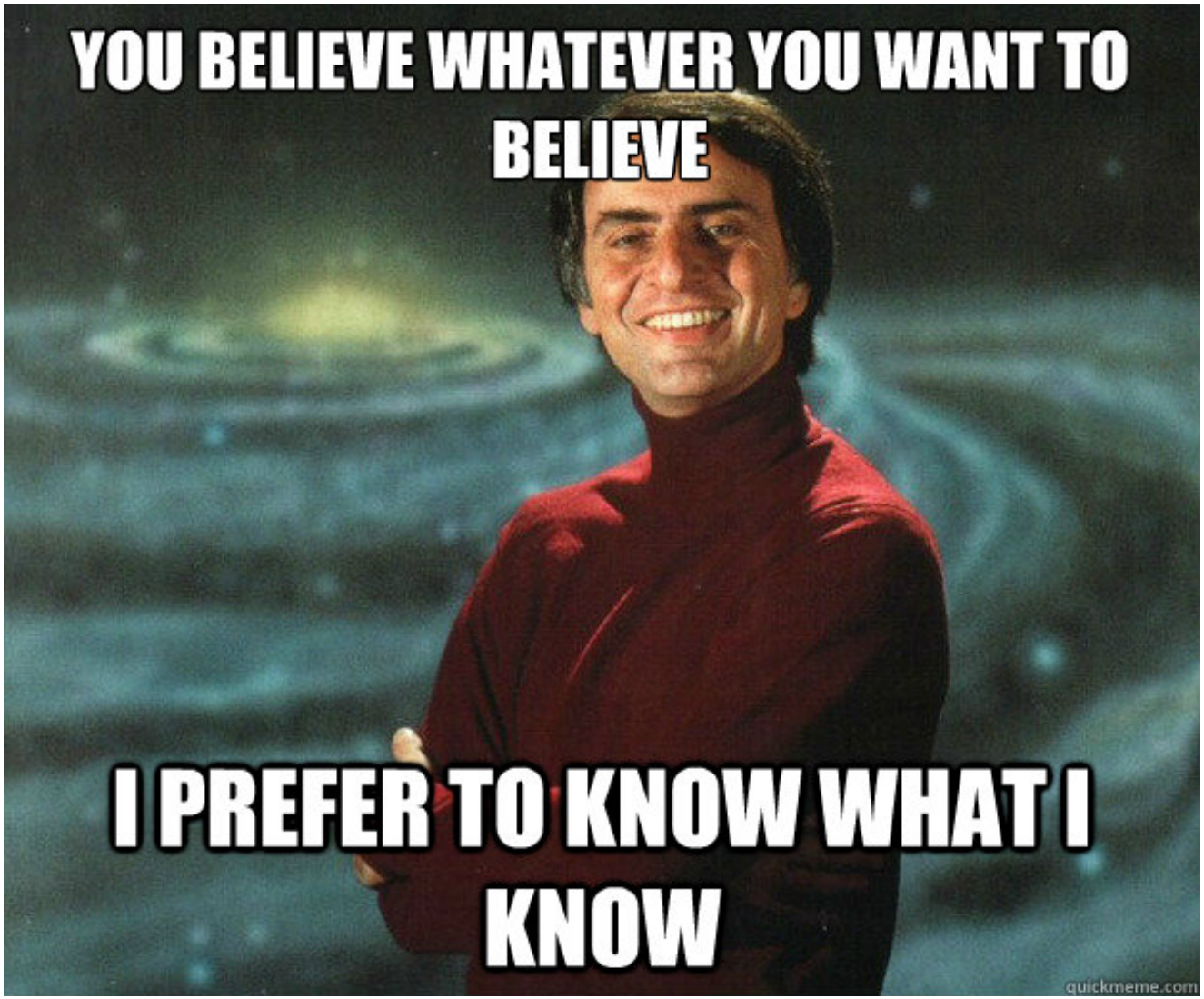
```
def fill_list(name):
    frame = sys._getframe().f_back

    lst = frame.f_locals[name]
    lst.append(42)

def foo():
    things = []
    fill_list('things')
    print things #=> 42
```

Why is Python slow?

CPython is a clean and maintainable software.

Why is Python slow?

Show me the numbers



Why is Python slow?

Show me the numbers

- speed.pypy.org
- Video processing example

What is PyPy?

PyPy is a python interpreter written in *Python*.

What is PyPy?


INCEPTION
EXPERIENCE IT JULY 16 IN THEATERS AND IMAX
© 2010 WARNER BROS. ENT.

What is PyPy?

PyPy is a python interpreter written in *RPython*.

Motivation

- The **Pareto princpile**: the 20% of the program will account for the 80% of the runtime.
- The **Fast path principle**.

Definition

RPython is a restricted subset of Python that is amenable to static analysis.

Goals

- Abstract as much as possible creating Interpreters for DL.
- l * o * p
- l - language to analyze.
- o - optimize and tweak depending on different factors. (GC ie)
- p - being able to produce interpreter for different platforms.

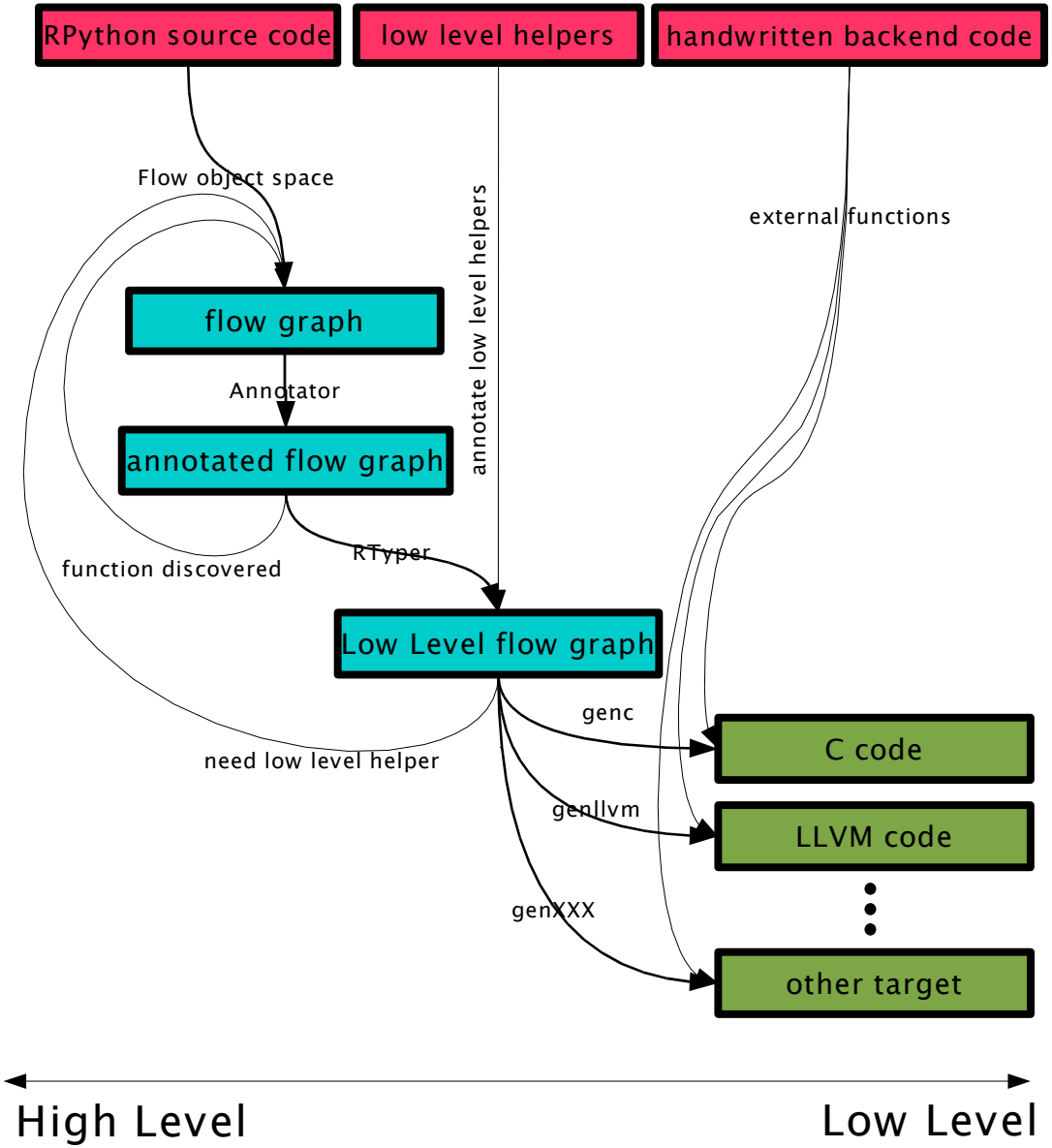
Hello World

Mandatory Hello World.

Translation process

- Complete program is imported generating *control flow graph*.
- The **Annotator** does type inference.
- The **Rtyper** uses high level types to transform into low level ones.
- Some optimizations are applied.
- Next step is prepare *graphs* to be translated.
- The C backend generates source files.

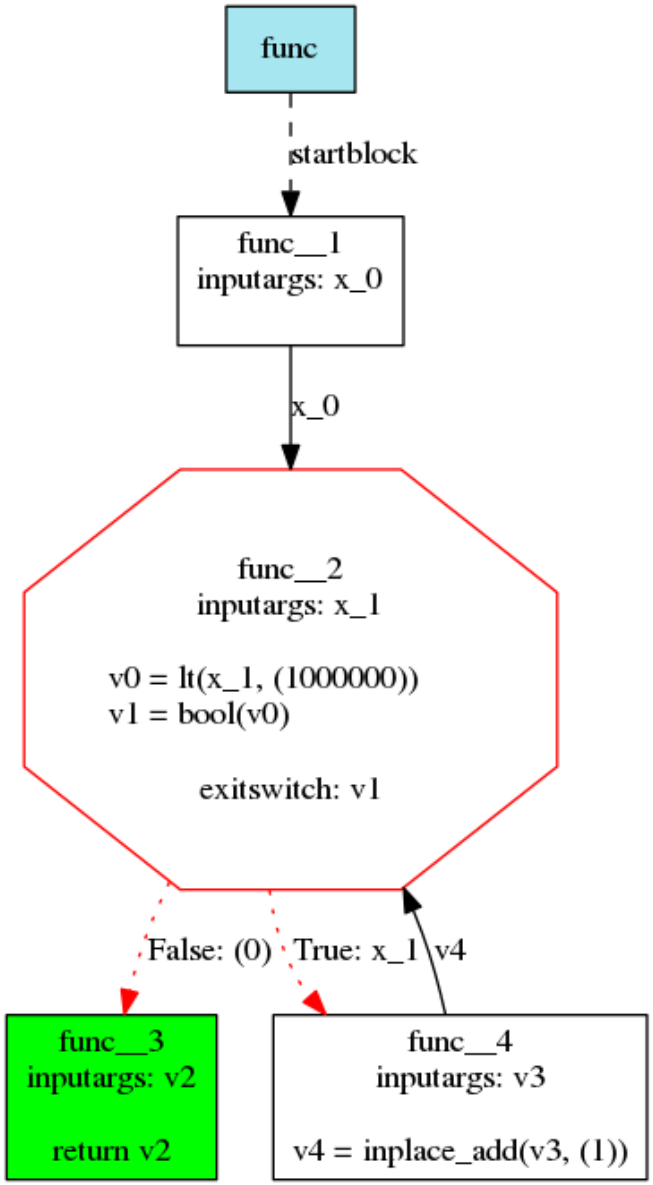
Translation process



Building Flow Graphs

- Code is **NOT** parsed.
- Uses *code objects* that define the behaviour.
- *Flow Graph builder* using an *abstract interpretation* producing *control flow graph*.

Control flow graph



Building Flow Graphs

```
def f(n):
   return 3*n+2
```

Building Flow Graphs

```
Block(v1):          # input argument
  v2 = mul(Constant(3), v1)
  v3 = add(v2, Constant(2))
```

Annotation phase

$$\frac{z = \mathrm{add}(x, y),\ b(x) = Int,\ Bool \leq b(y) \leq Int}{b' = b \text{ with } (z \to Int)}$$

$$\frac{z = \mathrm{add}(x, y),\ Bool \leq b(x) \leq Int,\ b(y) = Int}{b' = b \text{ with } (z \to Int)}$$

$$\frac{z = \mathrm{add}(x, y),\ Bool \leq b(x) \leq NonNegInt,\ Bool \leq b(y) \leq NonNegInt}{b' = b \text{ with } (z \to NonNegInt)}$$

$$\frac{z = \mathrm{add}(x, y),\ Char \leq b(x) \leq NullableStr,\ Char \leq b(y) \leq NullableStr}{b' = b \text{ with } (z \to Str)^1}$$

Annotation phase

- Each variable that appears in the *flow grap* is *annotated*.
- With all possible values.

Annotation phase

- SomeObject
- SomeInteger
- SomeString
- SomeChar
- SomeTuple([s1, s2..., sn])
- SomeList
- SomeDict
- SomeInstance

RTyper

```
v3 = add(v1, v2)
```

```
v1 -> SomeInteger()
v2 -> SomeInteger()
v3 -> SomeInteger()
```

Result

```
v3 = int_add(v1, v2)
```

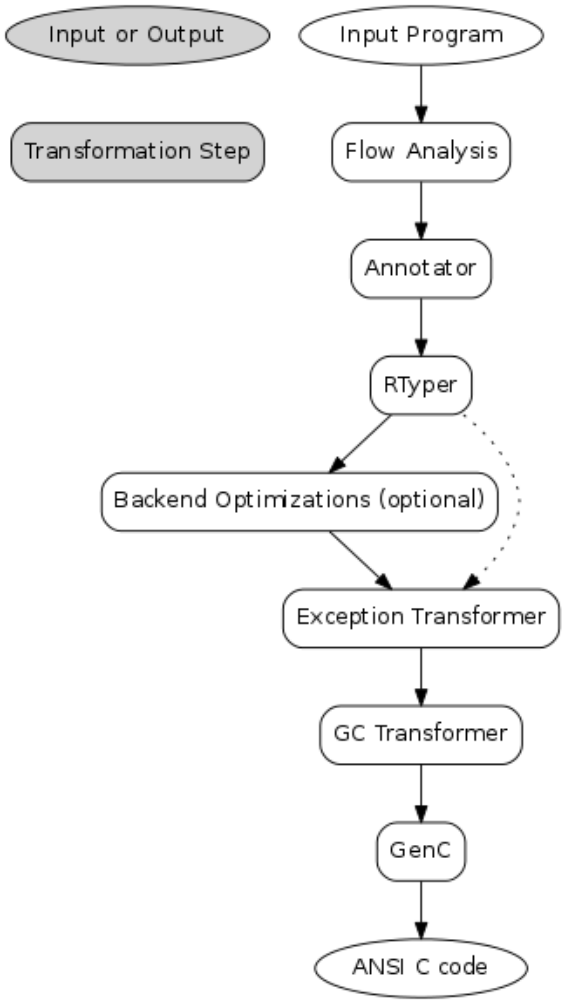
Backend Optimizations

- Function inlining.
- Malloc removal.

Preparation for source generation

- Exception handling, since C doesn't have that concept.
- Memory Management, GC *Pluggable*.

Overview

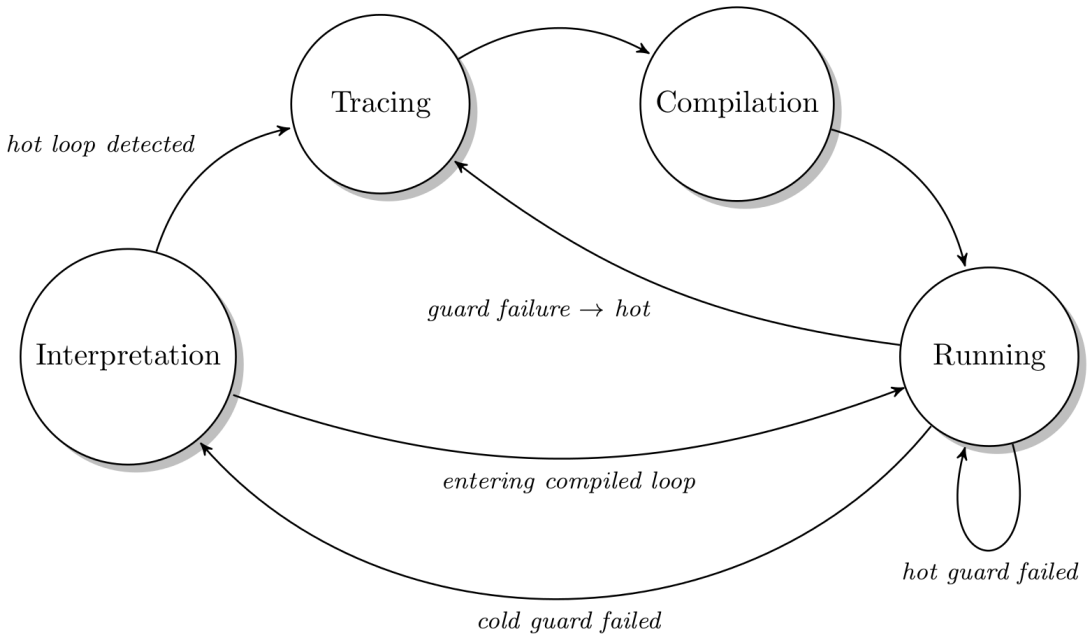

Input or Output
Input Program
Transformation Step
Flow Analysis
Annotator
RTyper
Backend Optimizations (optional)
Exception Transformer
GC Transformer
GenC
ANSI C code

Jit

Definition

In computing, just-in-time compilation (JIT), also known as dynamic translation, is compilation done during execution of a program at run time rather than prior to execution. Wikipedia.

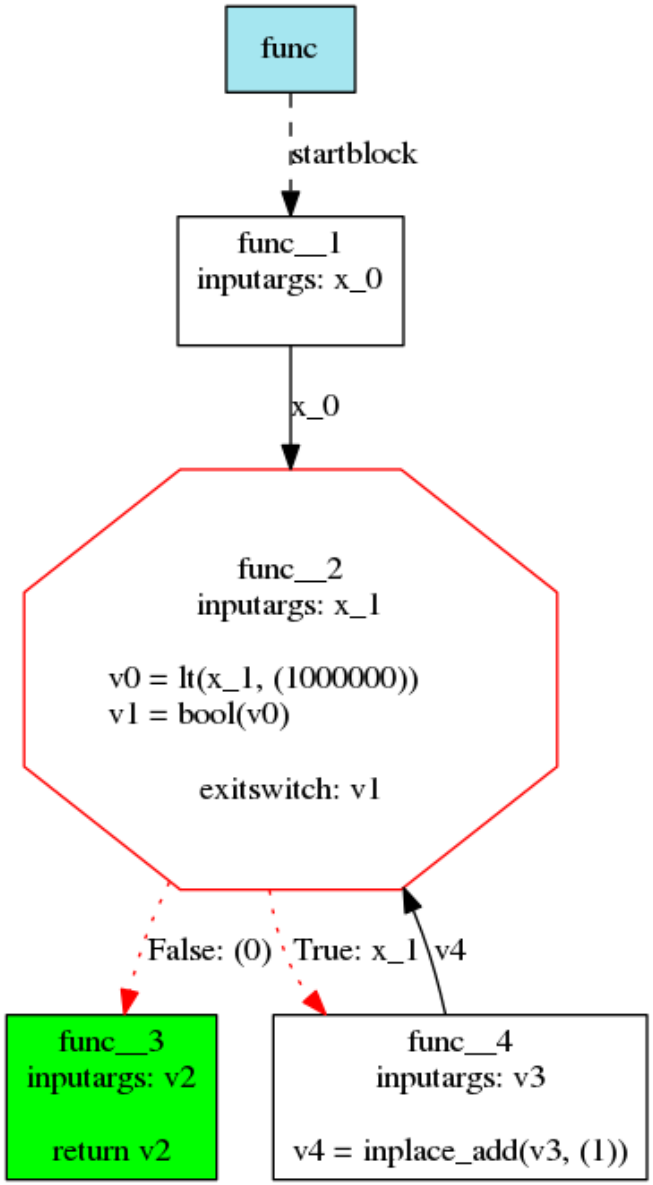
Overview


Tracing
Compilation
hot loop detected
guard failure → hot
Interpretation
Running
entering compiled loop
hot guard failed
cold guard failed

PyPy strategy

- RPython generates a tracing JIT.
- Instead of user code is interpreter code who launch compiled code.
- It comes mostly for free to language implementators.

Control flow graph

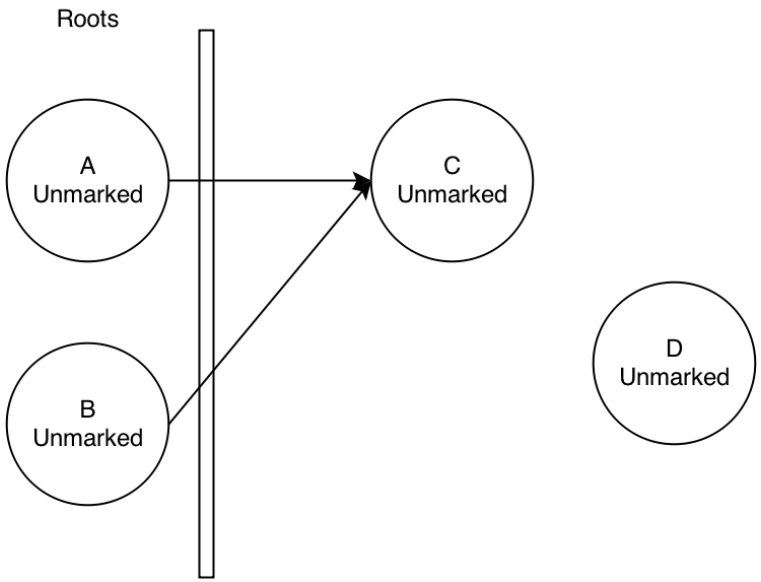


```
tlrjitdriver = JitDriver(greens = ['pc', 'bytecode'],
                         reds = ['a', 'regs'])

def interpret(bytecode, a):
    regs = [0] * 256
    pc = 0
    while True:
        tlrjitdriver.jit_merge_point()
        opcode = ord(bytecode[pc])
        pc += 1
        if opcode == JUMP_IF_A:
            target = ord(bytecode[pc])
            pc += 1
            if a:
                if target < pc:
                    tlrjitdriver.can_enter_jit()
                pc = target
        elif opcode == MOV_A_R:
```

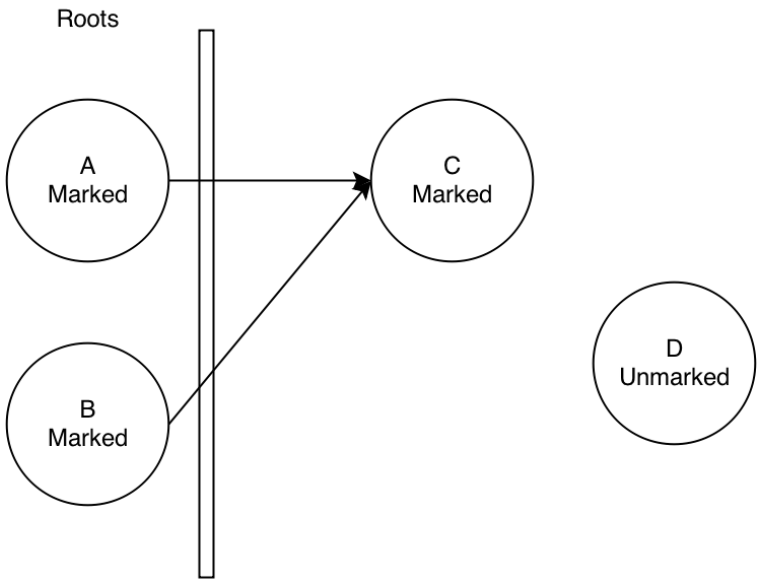
PyPy strategy

```
[elem * 2 for elem in elements]
balance = (a / b / c) * 4
'asdadsasd-xxx'.replace('x', 'y').replace('a',
foo.bar()
```

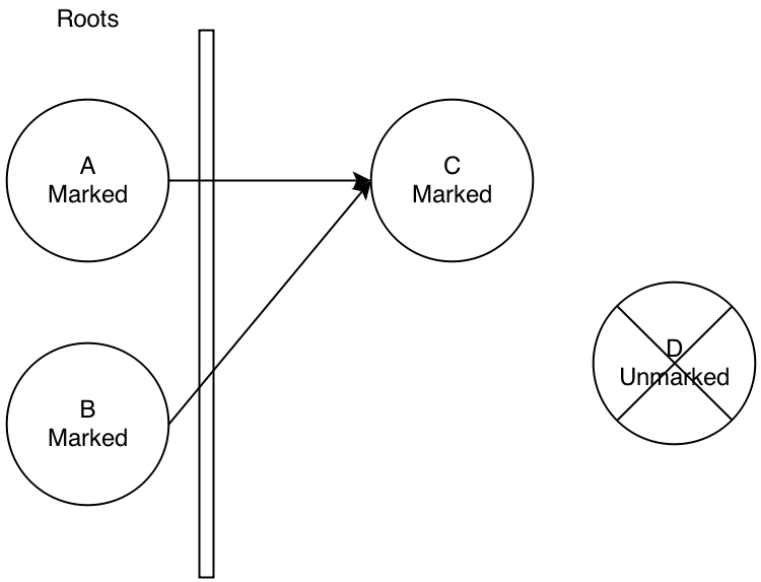
Mark and Sweep Algorithm



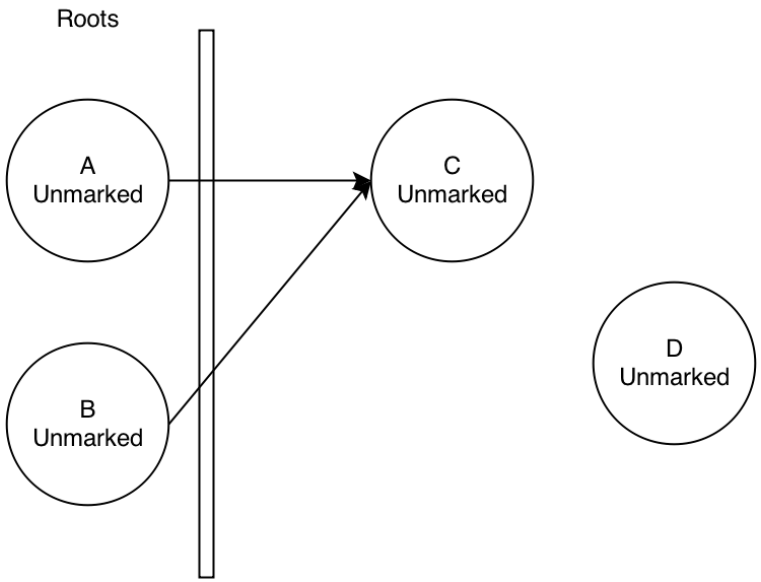
Mark and Sweep Algorithm



Mark and Sweep Algorithm



Mark and Sweep Algorithm



Mark and sweep

- Pros: Can collect cycles.
- Cons: Basic implementation stops the world

Questions?

The End