

## Issues with dynamic languages

Dynamic scripting languages are everywhere

- Python, Matlab, R

Problem

- Highly flexible, Poor performance
- Usually interpreted, not compiled

What do we need?

- Good native code compilers for dynamic languages

Writing compilers for each new dynamic language is hard

- Language, platform-specific issues
- Memory management
- Lack of type information

## Translate to a functional language

Steal the results of functional programmers' labor

Why functional languages?

- Excellent memory management
- Highly expressive type systems
- Precise expression of dynamic language semantics
- Infrastructure for re-usable compiler optimizations

Demonstration

- Translate Python to native code via OCaml
- Use OCaml's existing compiler infrastructure
- Build both the Python-to OCaml translator and the Python runtime environment in OCaml

## Python in 2 minutes

```

z = 10
class MyInt(int):
    # constructor
    def __init__(self, x=0):
        self.v = x
    # '+' operator
    def __add__(self, x):
        return 2*(self.v+x)
i1 = MyInt(5)
x = i1 + z # x = 30
y = i1 * 2 # y = 10

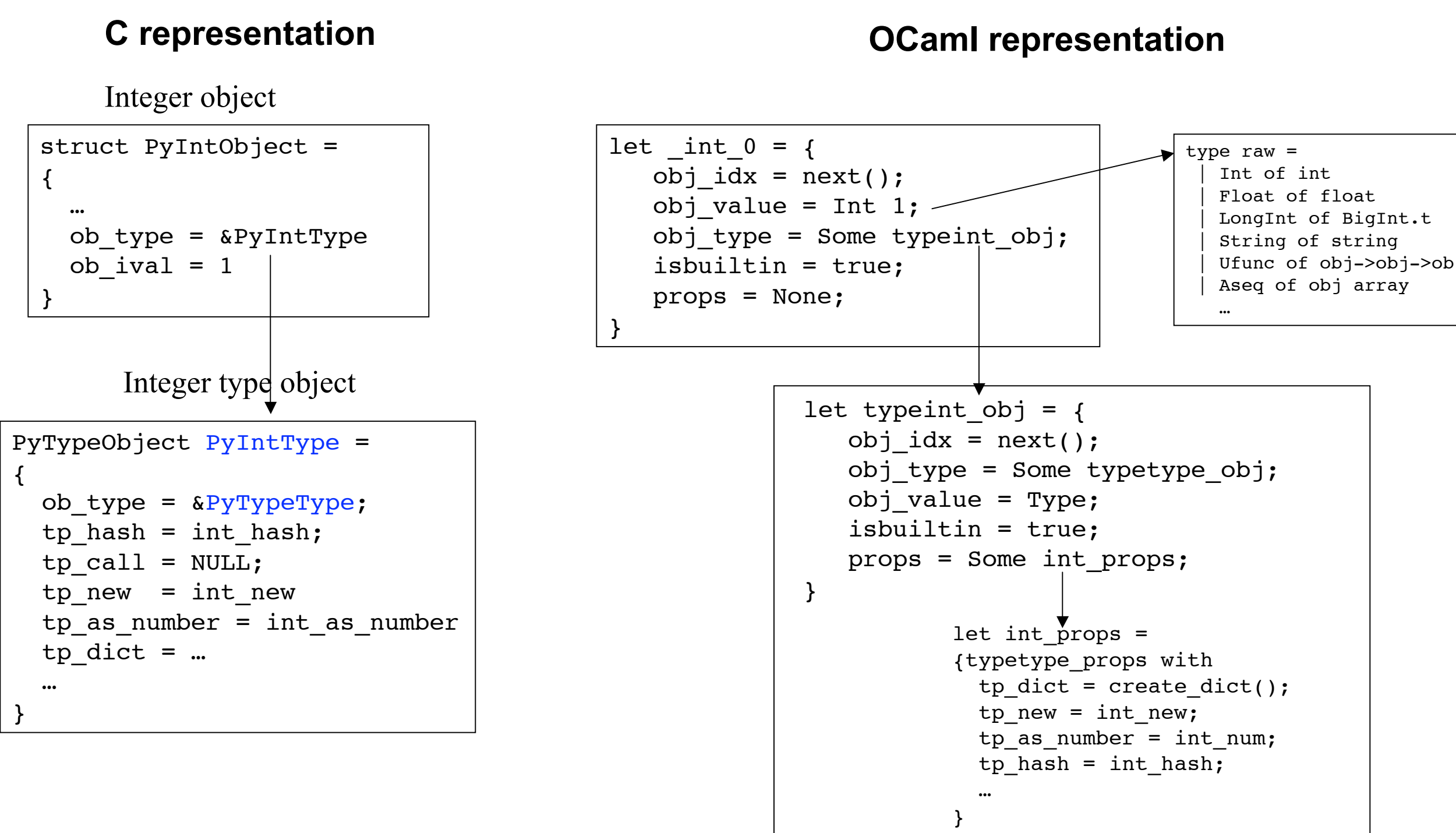
```

Annotations:

- Built-in object with type 'int'
- User-defined type object
- Method objects, stored in object dictionary
- Instance of user-defined type

## Representing Python objects

Every object has a type, describing its structure & behavior

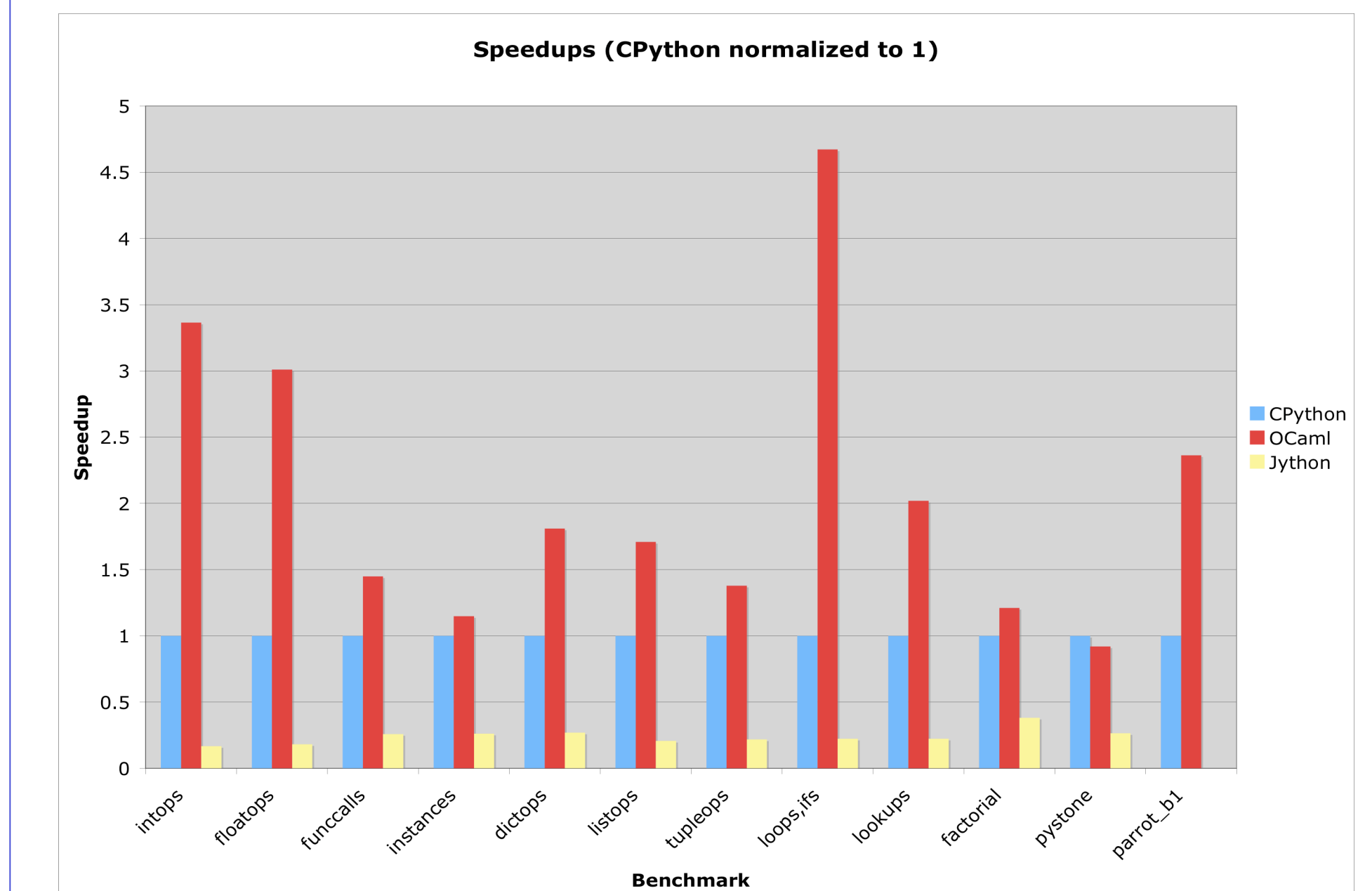


## Translating key Python features

Comments	Python code	Translation to OCaml
<b>Operators</b> • Operators are internally method calls • $x + y$ is internally $x.__add__(y)$ • Python and OCaml evaluate function args in different order, has to be taken into account	$x + y$	$[x+y]$ scope = <pre> let _t1_ = [x] scope in let _t2_ = [y] scope in   pynumber_add _t1_ _t2_ </pre>
<b>Function definitions</b> • Creates a new scope and a new function object • Can have both positional and keyword (named) arguments • Special case: function called with only positional args equal to total number of parameters	<pre> def add(x,y=0):     return 2*(x+y)  x1 = add(10,5) #30 x2 = add(5) #10 x3 = add() #Error </pre>	<pre> [def add params defaults body] scope = let newscope = getnewscope("add") in let fcode = (fun pl kw -&gt;   if  pl  =  params  &amp;&amp;  kw =0 then (     match pl with [x;y] -&gt;       _x:=x;_y:=y   ) else (     match (process params kw) with     [self;x] -&gt;       _x := x; _y:=y   );   [body] newscope ) in func_new fcode </pre>
<b>New types(classes)</b> • A class itself is an object, an instance of a metaclass • 3 steps in class construction: • find metaclass • find primary base • initialize dictionary	<pre> class MyInt(int):     def __init__(self, x):         self.v = x     ... </pre>	<pre> Metaclass bases type._new_(type, "MyInt", [int], {__init__:&lt;-&gt;}) Type construction function name Initial dict </pre>
<b>Class Instantiation</b> • Create a new instance using the <code>__new__</code> function of the object's type • Initialize the newly created object using the <code>__init__</code> method, if provided	<code>MyInt(5)</code>	<pre> Instance construction function Type Arguments i1 = int._new_(MyInt, [5]) MyInt.__init__[i1;5] Initialization function </pre>

## Preliminary Results

Better performance on several benchmarks



Smaller code size for compiler and runtime environment

	#lines in OCaml	#lines in C (approx)
Parser + compiler	3,000	16,000
Runtime environment	6,500	50,000

Primary bottlenecks in performance (from profiling):

- Memory allocation during creation of new objects
- Method lookup: finding the correct method to call at runtime and extracting it from object dictionary

## Current status & Future work

Some Python features need to be implemented:

- Stack traces for exceptions
- Deletion(finalization) methods
- Foreign language interface

Subsequent steps:

- Finish compiler for 'pure Python' (no foreign interfaces) and release in public domain
- Compiler optimizations on the OCaml intermediate targeting performance e.g. **tag elimination**

## Summary

We have obtained

- A representation of source dynamic language objects in terms of target functional language data types
- A formal semantics for the source language in the form of a translator
- A compiler which generates faster code than the standard implementation, even without significant compiler-level optimization

This approach to building compilers improves productivity of the language/compiler writer, and the end user of these dynamic languages