Language Composition with RPython

PyConUK 2015


Edd Barrett


Lukas Diekmann


Laurence Tratt


Carl Friedrich Bolz

Software Development Team

September 19, 2015

Roadmap

- Language composition background
- Challenges
- Our approach
- Concrete example: PHP + Python

Language Composition

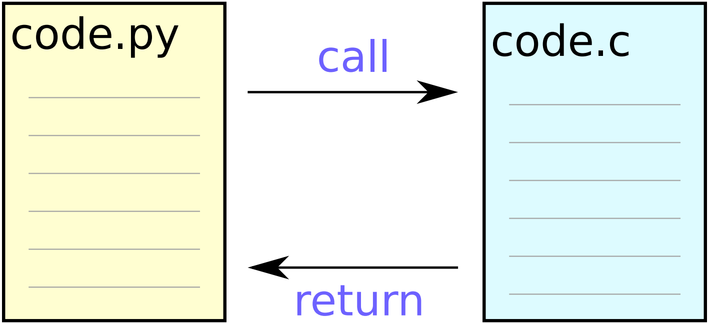
"The ability to write a computer program in a mixture of programming languages."

Why Compose Languages?

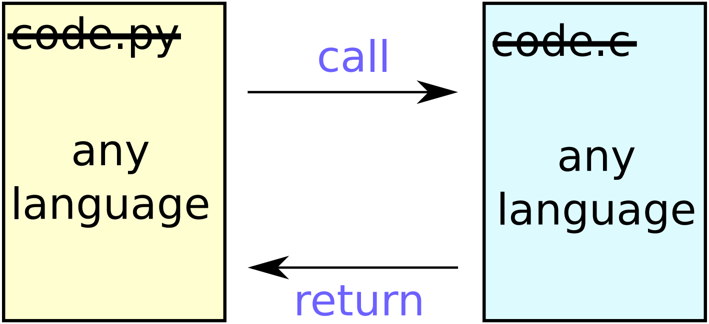
- Different parts of a program expressed best with different langs
- Language migration

What about FFIs?

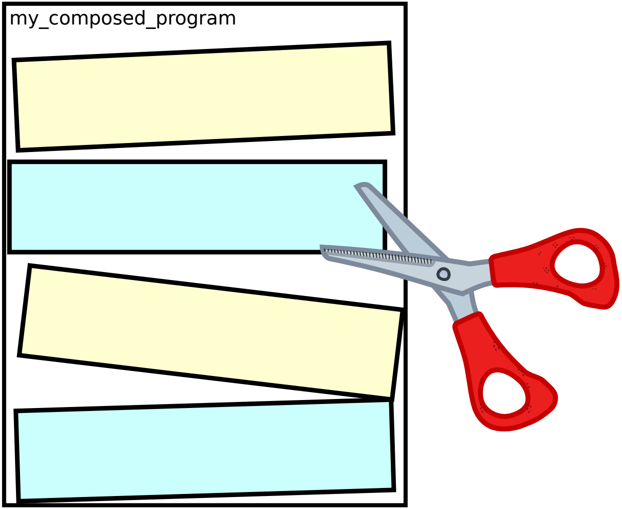
```
import cffi
```

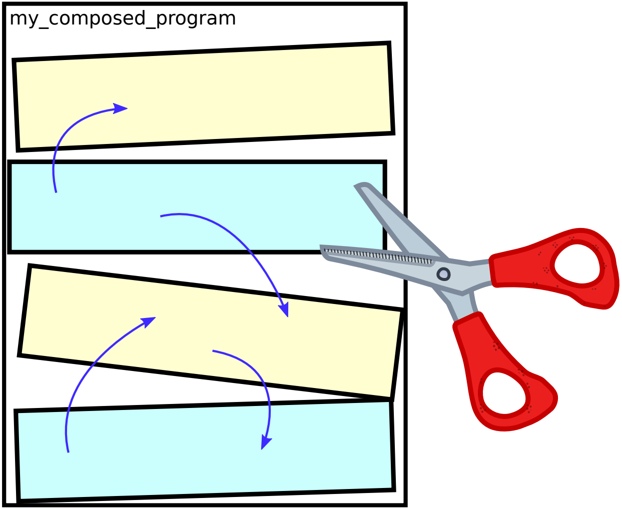

code.py
call
return
code.c

Rethinking the FFI



Rethinking the FFI

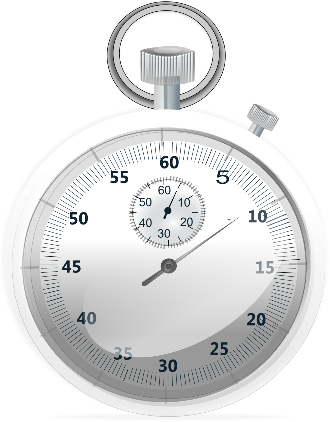



my_composed_program

From different languages, mix:

- modules
- functions
- methods
- expressions
- scopes

Performance Matters



Users won't adopt composition if it slows their programs down

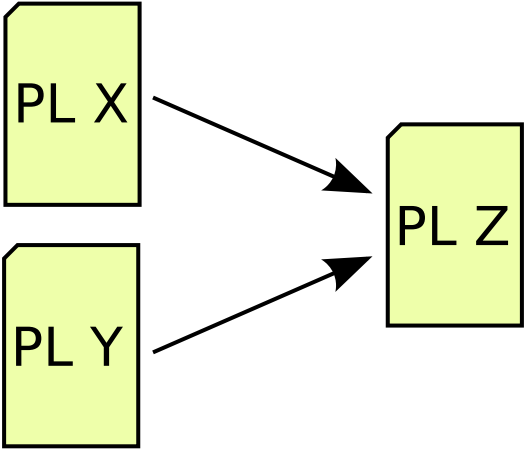
Goal

We want language compositions which are:

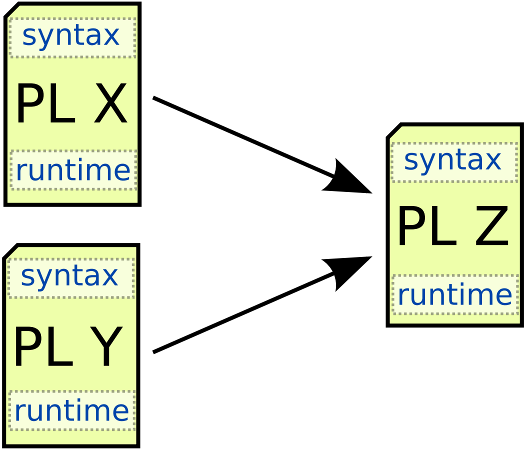
FINE-GRAINED and *FAST*

How?

Breaking it Down



Breaking it Down



Composing Syntax

PL X
<grammar>

```
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...
```

∪

PL Y
<grammar>

```
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...
```

=

PL Z
<grammar>

```
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...
```

Easy?

Composing Syntax

- LR $\rightarrow$ Possibly undefined
- PEG $\rightarrow$ Shadows
- GLR $\rightarrow$ Ambiguous

Syntax Directed Editing

```
public class Say extends <none> implements <none> {

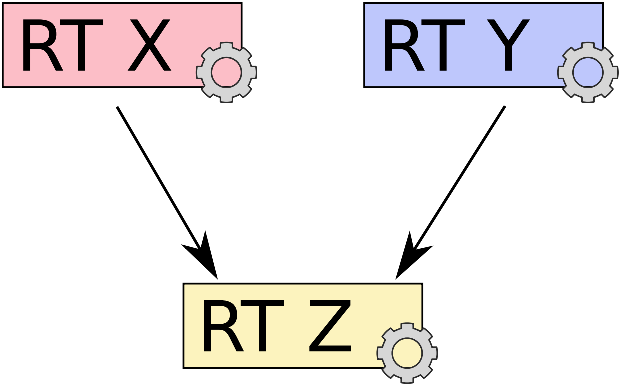
  private String textchanged;
  <<properties>>
  <<initializer>>
  public Say(String text) {
    <no statements>
  }

  <<methods>>

  <<nested classifiers>>
}
```

Poor editing experience

Syntax Composition



Easy?

Approaches to Interpreter Composition

Edd Barrett, Carl Friedrich Bolz, Laurence Tratt

Software Development Team, Informatics, King's College London, UK

Abstract

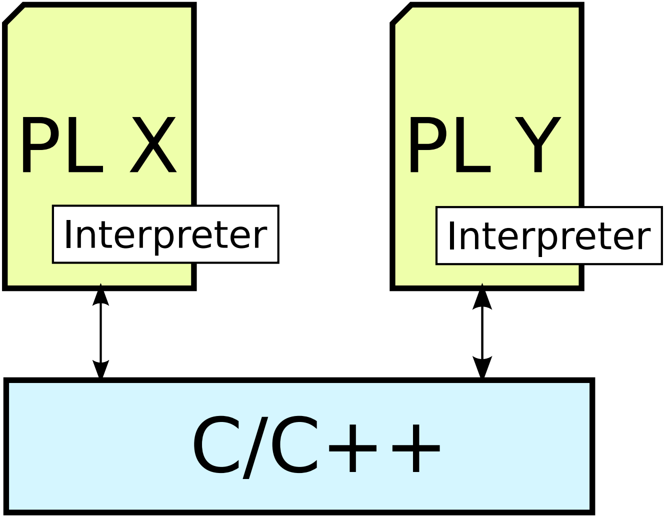
In this paper, we compose six different Python and Prolog VMs into 4 pairwise compositions: one using C interpreters; one running on the JVM; one using meta-tracing interpreters; and one using a C interpreter and a meta-tracing interpreter. We show that programs that cross the language barrier frequently execute faster in a meta-tracing composition, and that meta-tracing imposes a significantly lower overhead on composed programs relative to mono-language programs.

Keywords: Language composition, Virtual machines

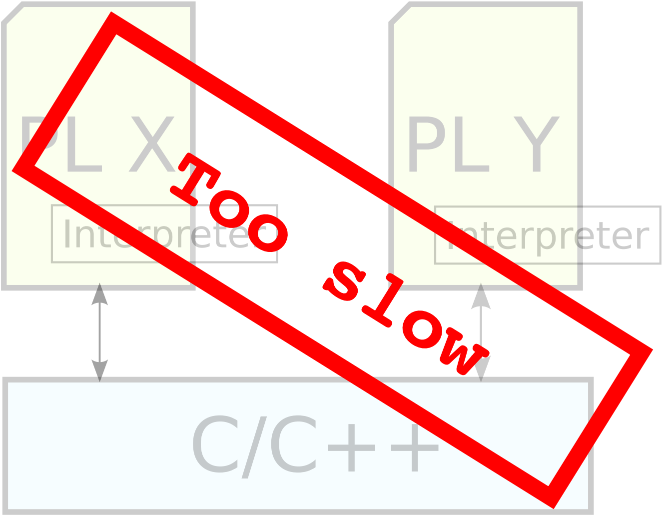
1. Overview

Programming language composition aims to allow the mixing of programming languages in a fine-grained manner. This vision brings many challenging problems, from the interaction of language semantics to performance. In this paper, we investigate the runtime performance of composed programs in high-level languages. We start from the assumption that execution of such programs is most likely to be through composing language implementations that use interpreters

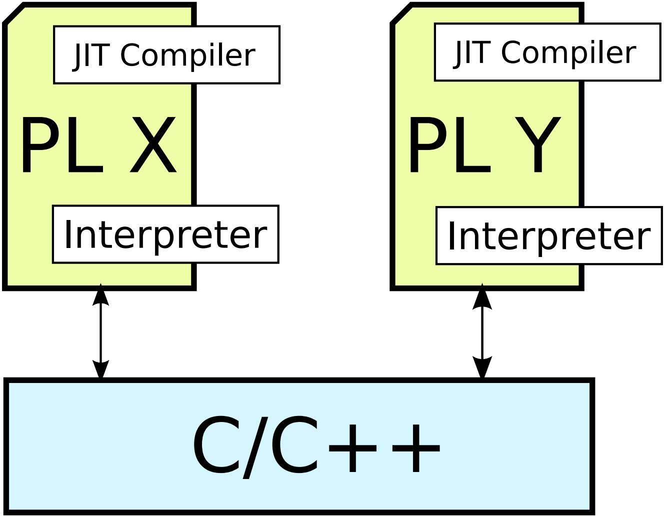
Runtime composition



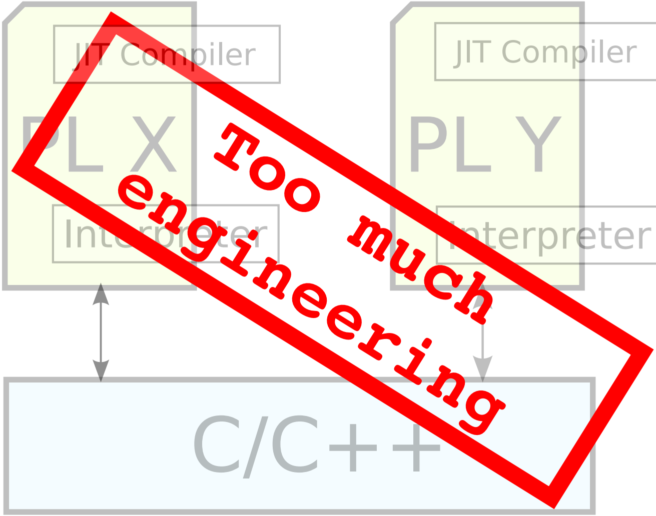
Runtime composition



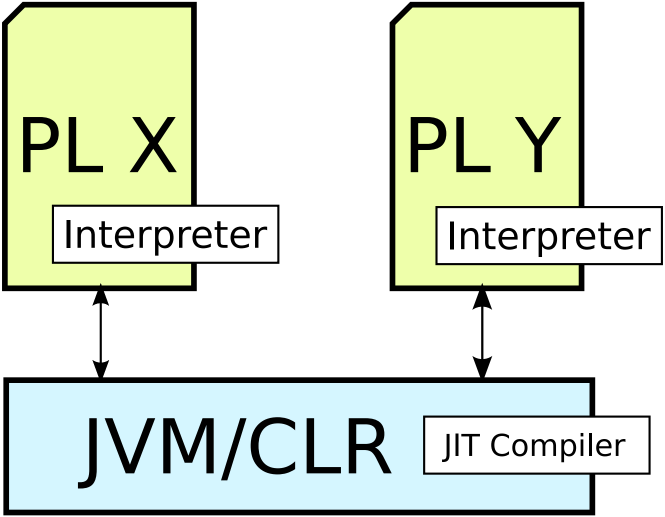
Runtime composition



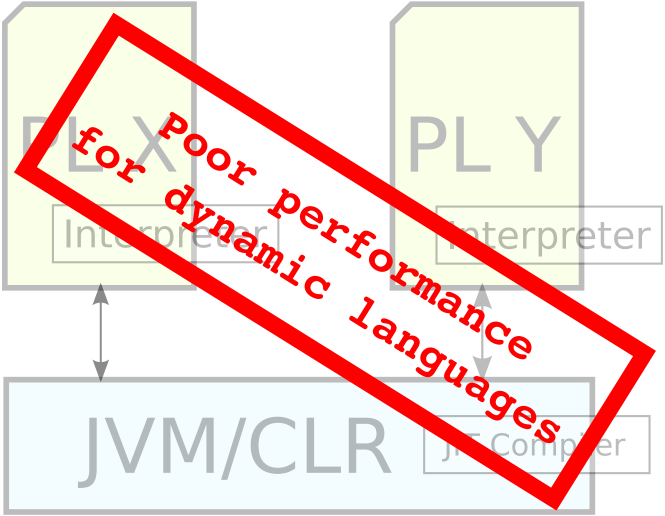
Runtime composition



Runtime composition

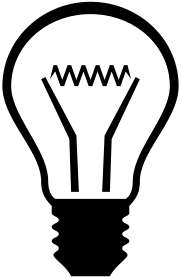


Runtime composition



Runtime Composition

Our Approach



Language Boxes + Meta-tracing

Language Boxes

- Borrows ideas from SDE
- Palatable editing experience
- Simple and practical way to compose grammars

Begin writing Java code

```
for (string s :
```

Language Boxes: E.g. Java + SQL

```
for (string s :
```

Open SQL language box

Language Boxes: E.g. Java + SQL

Write SQL code

```
for (string s : SELECT * FROM tbl WHERE
```

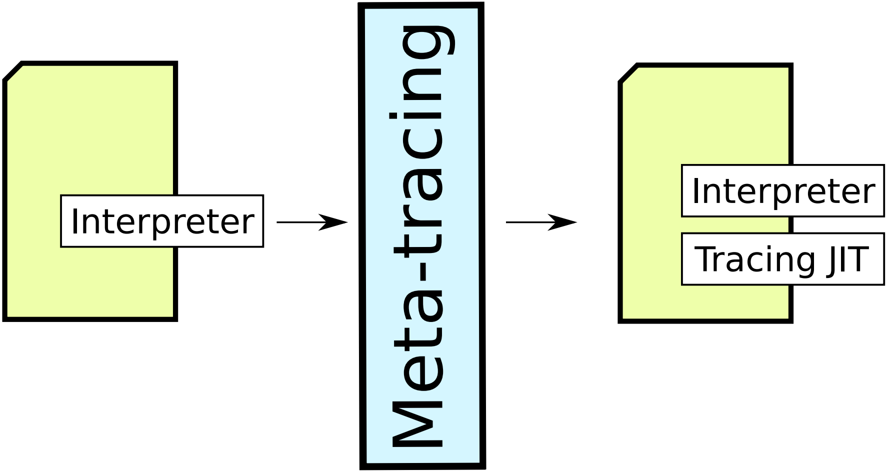
Language Boxes: E.g. Java + SQL

```
for (string s : SELECT * FROM tbl WHERE
name = this.name;) {
```

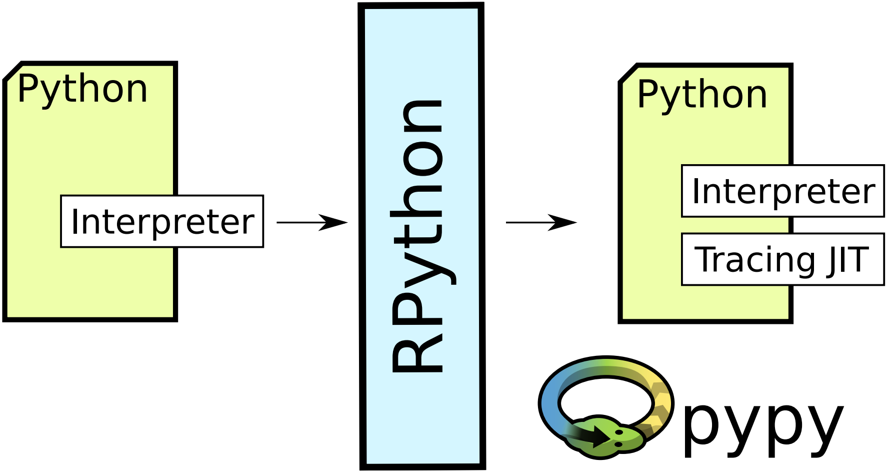
Java code


Syntactic Composition

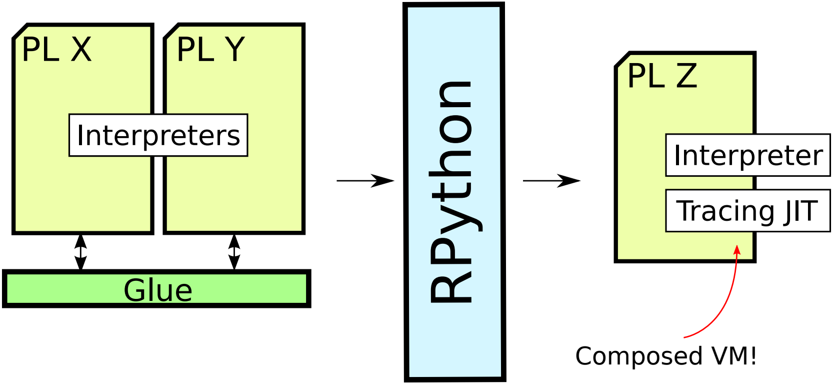
Meta-tracing



Meta-tracing



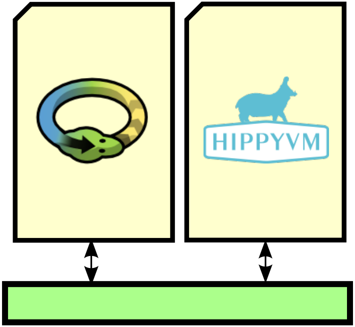
How Does this Apply to VM Composition?



Little Engineering + High Performance


Runtime Composition

A Concrete Example: PyHyp



+

Language box editor

FFI-like Features

- Import Python modules into PHP
- Calling Python functions and methods from PHP
- Calling PHP functions and methods from Python
- Automatic type “conversion”

“Advanced Features”

- Foreign functions in the same file
  - And arbitrarily nested
- Python expressions in PHP
- “Embedding” Python methods inside PHP classes
  - And integration with PHP access modifiers
- Adds support for references to Python
- Cross-language scoping
- Mixed-frames in backtraces new!

Demo

How Much Effort?

Implementing desired behaviour: relatively easy

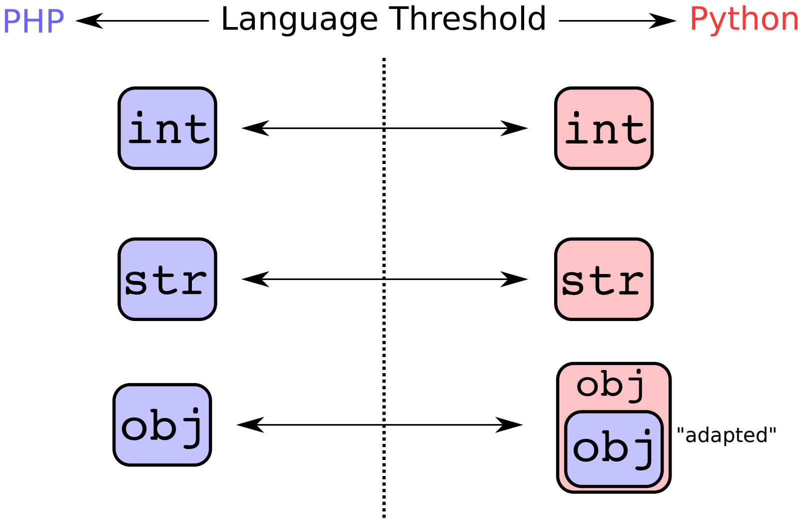
Deciding the correct behaviours: hard

Compromises sometimes must be made

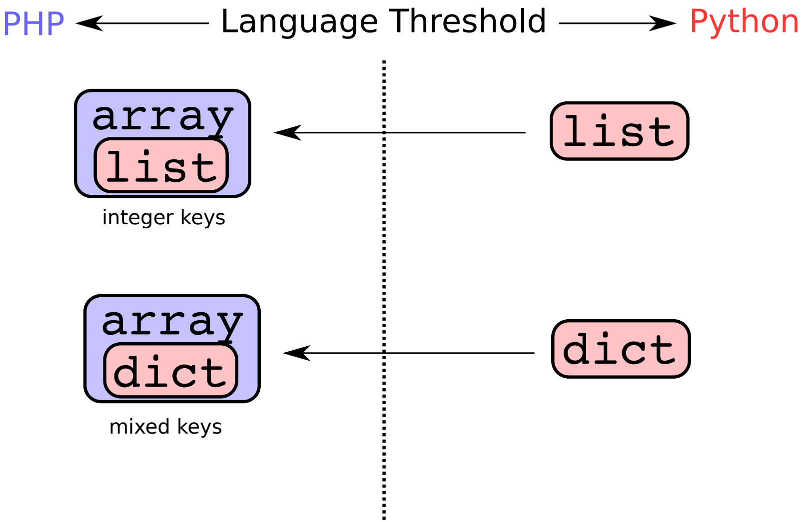
Making Compromises

Example: Collection types across languages

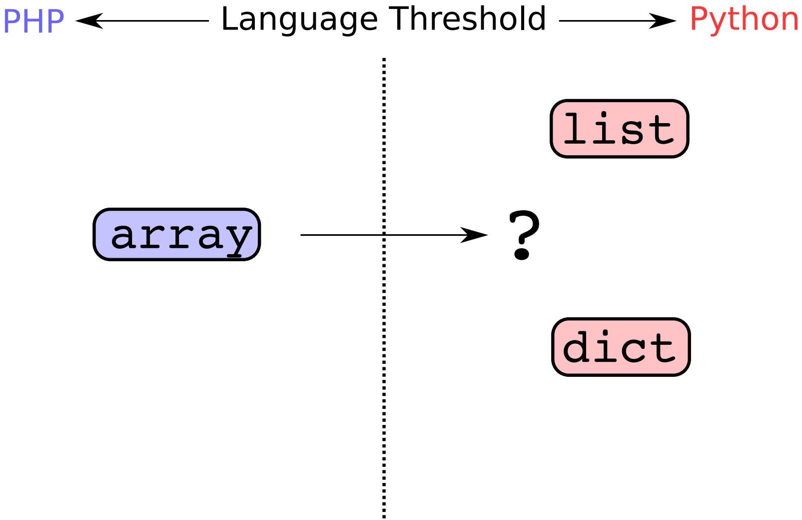
Array/Dict/List Conversions



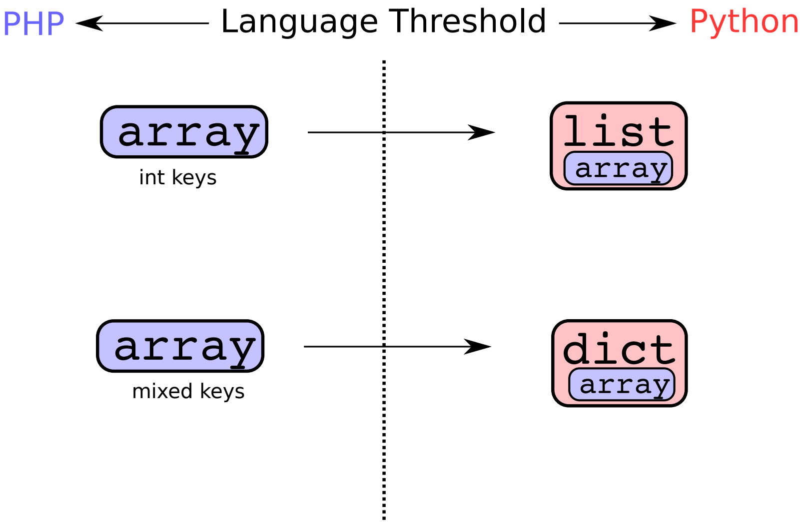
Array/Dict/List Conversions



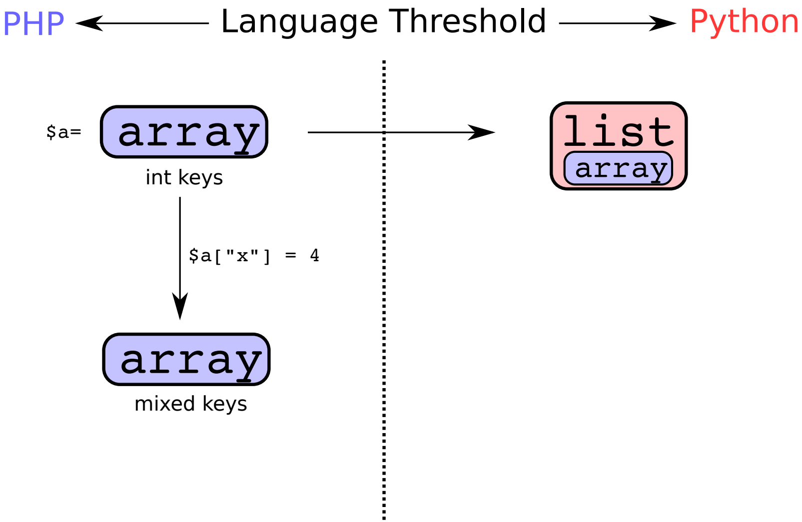
Array/Dict/List Conversions



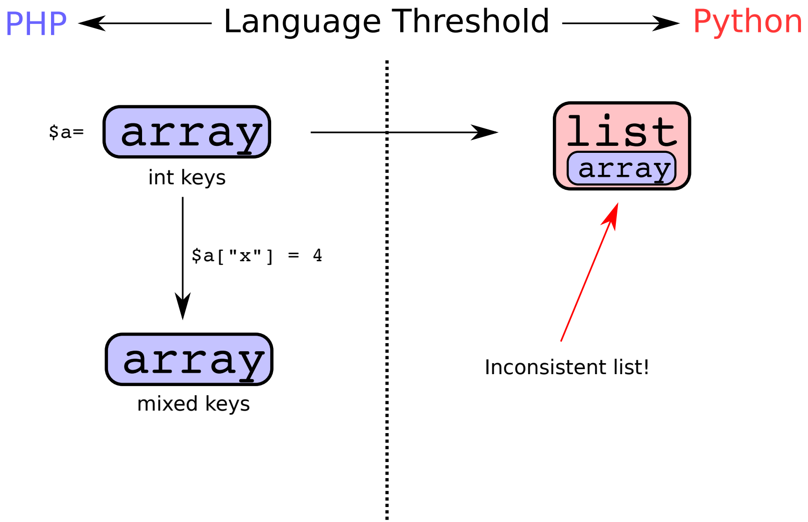
Array/Dict/List Conversions



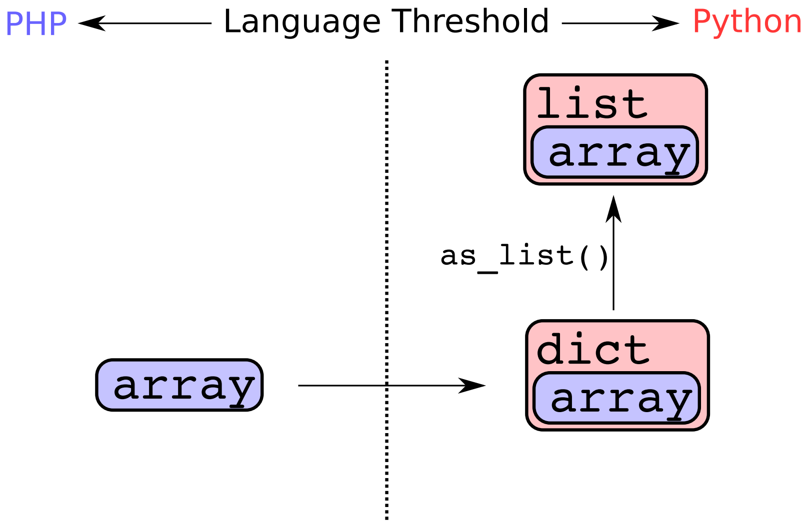
Array/Dict/List Conversions



Array/Dict/List Conversions



Array/Dict/List Conversions



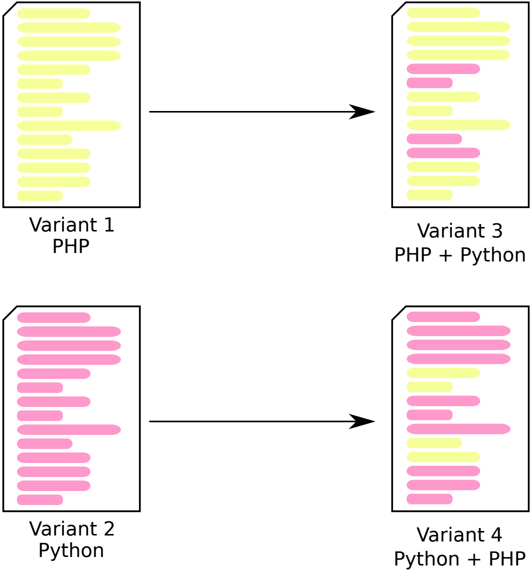
Performance Evaluation

“Good Performance”

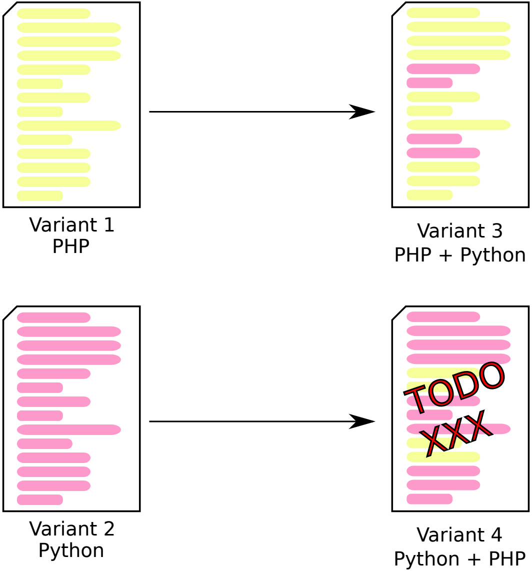
Composed programs should perform “close” to equivalent mono-language programs

Aim for between 1-2x slower. 3x is too slow

Benchmark “Variants”



Benchmark “Variants”



Benchmarks

- deltablue
- fannkuch
- mandel
- richards
- instchain
- l1a0r
- l1a1r
- lists
- ref_swap
- return_simple
- scopes
- smallfunc
- sum
- sum_meth
- sum_meth_attr
- total_list
- walk_list

Microbenchmarks: Relative to PyHyp Variant3

Benchmark	CPython	HHVM	HippyVM	PyHyp$_m$	PyPy	Zend
instchain	22.172 ±0.0859	6.209 ±0.0234	0.969 ±0.0036	0.967 ±0.0039	0.477 ±0.0019	24.248 ±0.1191
l1a0r	71.633 ±1.4869	3.770 ±0.0793	1.230 ±0.0255	1.230 ±0.0256	1.231 ±0.0298	37.752 ±1.1719
l1a1r	76.171 ±0.1207	3.000 ±0.0038	1.285 ±0.0003	1.285 ±0.0002	1.144 ±0.0077	41.052 ±0.3498
lists	7.485 ±0.0227	0.879 ±0.0072	0.966 ±0.0037	0.977 ±0.0041	0.520 ±0.0018	16.106 ±0.0736
ref_swap		6.911 ±0.0054	1.000 ±0.0003	1.000 ±0.0003		55.360 ±0.7395
return_simple	108.576 ±0.2690	6.915 ±0.0009	1.000 ±0.0001	1.000 ±0.0002	0.889 ±0.0002	83.708 ±0.7264
scopes	123.284 ±1.5081	14.969 ±0.0391	4.528 ±0.0099	4.512 ±0.0588	1.000 ±0.0003	156.443 ±0.5742
smallfunc	184.778 ±0.3071	12.818 ±0.0099	1.000 ±0.0003	1.000 ±0.0003	1.000 ±0.0003	243.318 ±0.8453
sum	299.582 ±0.3659	19.083 ±0.0172	1.000 ±0.0005	1.000 ±0.0005	0.874 ±0.0003	427.513 ±2.6441
sum_meth	328.894 ±1.2870	23.714 ±0.0955	0.998 ±0.0030	0.999 ±0.0030	0.873 ±0.0026	456.739 ±2.9270
sum_meth_attr	127.800 ±0.1907	17.819 ±0.2655	1.001 ±0.0019	1.116 ±0.0015	0.925 ±0.0016	148.167 ±0.6554
total_list	14.266 ±0.0248	2.080 ±0.0031	0.695 ±0.0005	0.696 ±0.0019	0.510 ±0.0014	30.356 ±0.1679
walk_list	4.869 ±0.0340	0.373 ±0.0025	0.773 ±0.0060	0.774 ±0.0107	1.099 ±0.0082	10.700 ±0.0846

Larger Benchmarks: Relative to PyHyp Variant3

Benchmark	CPython	HHVM	HippyVM	PyHyp_m	PyPy	Zend
deltablue	19.199 ± 0.6900	860.108 ± 31.1392	4.739 ± 0.1684	4.888 ± 0.1766	0.405 ± 0.0151	181.209 ± 6.7871
fannkuch	18.616 ± 0.0362	3.212 ± 0.0133	1.869 ± 0.0034	1.879 ± 0.0024	1.009 ± 0.0046	14.998 ± 0.1032
mandel		0.883 ± 0.0006	1.013 ± 0.0089	1.003 ± 0.0011		8.290 ± 0.0623
richards	28.291 ± 0.1091	12.726 ± 0.1296	0.745 ± 0.0036	0.766 ± 0.0044	0.531 ± 0.0026	27.081 ± 0.1445

Overall: Relative to PyHyp Variant3

Benchmark	CPython	HHVM	HippyVM	PyHyp$_m$	PyPy	Zend
Geometric Mean	48.575 $\pm$0.1493	6.698 $\pm$0.0188	1.206 $\pm$0.0032	1.218 $\pm$0.0035	0.785 $\pm$0.0024	56.521 $\pm$0.1833

Overall: Relative to PyHyp Variant3

Benchmark	CPython	HHVM	HippyVM	PyHyp$_m$	PyPy	Zend
Geometric Mean	48.575 $\pm$0.1493	6.698 $\pm$0.0188	1.206 $\pm$0.0032	1.218 $\pm$0.0035	0.785 $\pm$0.0024	56.521 $\pm$0.1833

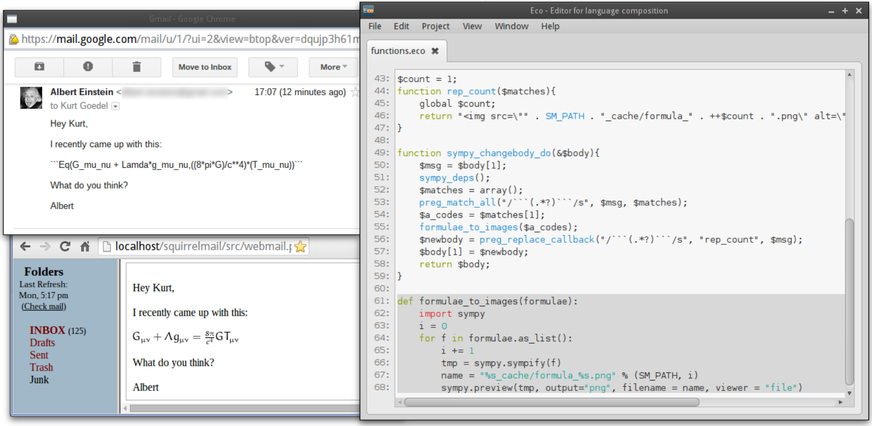
Conclusions

- Language boxes:
  - Practical syntax composition
  - Decent editor experience
- Meta-tracing:
  - Compositions with relatively little effort
  - Overall good performance
- Implementing x-lang behaviours is easy
- Designing x-lang behaviours is hard

Future Work

- Implement and measure benchmark variant 4
- Cross-language debugger
- Compositions with $>$2 languages involved
- Statically typed languages

Thanks


Gmail - Google Chrome
https://mail.google.com/mail/u/1/?ui=2&view=btop&ver=dqujp3h61m
Move to Inbox
More
Albert Einstein
17:07 (12 minutes ago)
to Kurt Goedel
Hey Kurt,
I recently came up with this:
```Eq(G_mu_nu + Lamda*g_mu_nu,((8*pi*G)/c**4)*(T_mu_nu))```
What do you think?
Albert
localhost/squirrelmail/src/webmail.p
Folders
Last Refresh:
Mon, 5:17 pm
(Check mail)
INBOX (125)
Drafts
Sent
Trash
Junk
Hey Kurt,
I recently came up with this:
$G_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi}{c^4} G T_{\mu\nu}$
What do you think?
Albert
Eco - Editor for language composition
File Edit Project View Window Help
functions.eco
$count = 1;
function rep_count($matches){
global $count;
return "<img src=\"" . SM_PATH . "_cache/formula_" . ++$count . ".png\" alt=\"
}
function sympy_changebody_do(&$body){
$msg = $body[1];
sympy_deps();
$matches = array();
preg_match_all("/```(.*?)```/s", $msg, $matches);
$a_codes = $matches[1];
formulae_to_images($a_codes);
$newbody = preg_replace_callback("/```(.*?)```/s", "rep_count", $msg);
$body[1] = $newbody;
return $body;
}
def formulae_to_images(formulae):
import sympy
i = 0
for f in formulae.as_list():
i += 1
tmp = sympy.sympify(f)
name = "%s_cache/formula_%s.png" % (SM_PATH, i)
sympy.preview(tmp, output="png", filename = name, viewer = "file")
```