

Language Composition

MMNet 2015



Edd Barrett



Lukas
Diekmann



Laurence
Tratt



Carl
Friedrich
Bolz

KING'S
College
LONDON

Software Development Team
2013-07-15

Roadmap

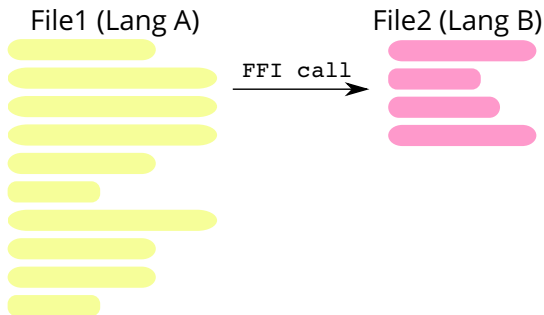
- Language composition background.
- Challenges.
- Our approach.
- Concrete Example: PHP + Python.

“The ability to write a computer program in a mix of programming languages.”

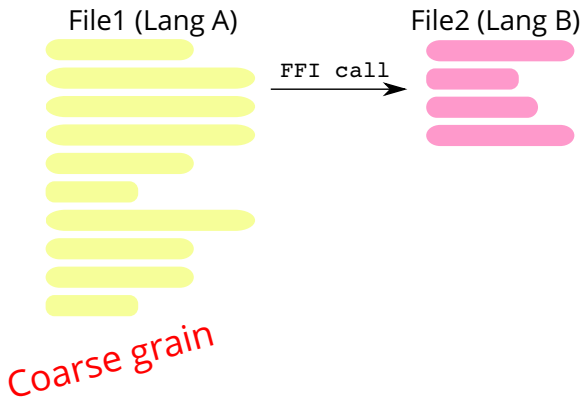
Why Compose Languages?

- Parts of a program are expressed best with different languages.
 - User Interface
 - String manipulation
 - Statistical analysis
 - Constraint solving
 - ...
- Performance.
 - Start writing in expressive language X.
 - Port bottlenecks to fast language Y.
- Language migration.
 - Gradual reimplementations.

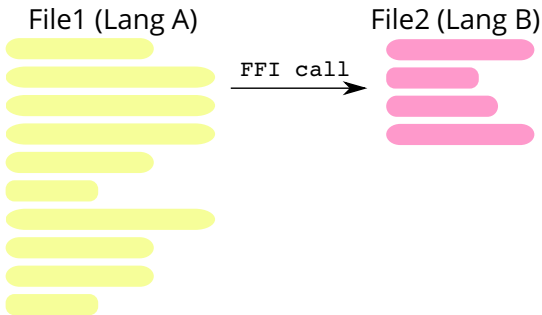
Most languages have a Foreign Function Interface



Most languages have a Foreign Function Interface



Most languages have a Foreign Function Interface

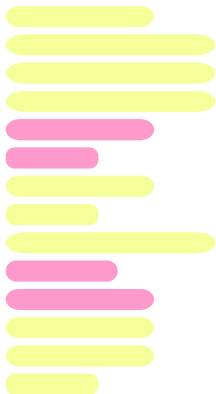


Coarse grain

Limited choice of langs

(second lang is nearly always C)

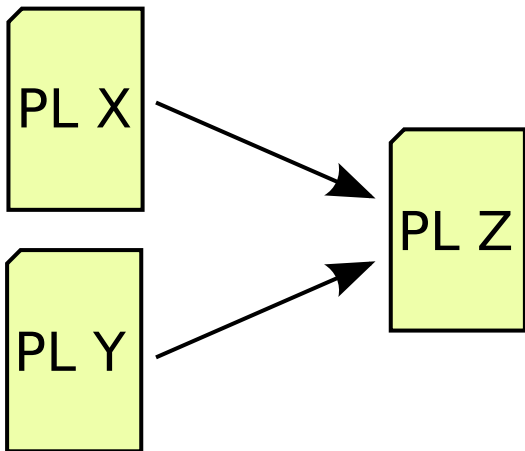
Can we do better?



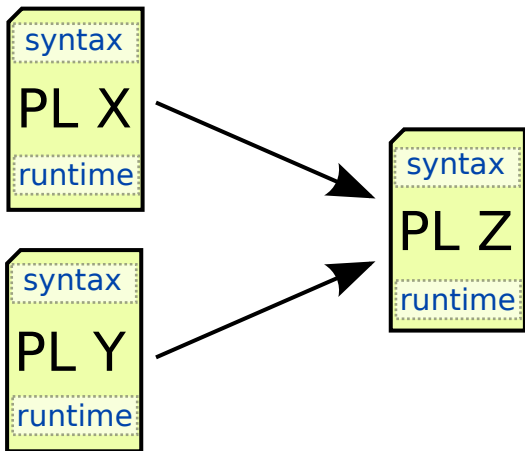
Our Aims:

- Fine-grain composition
 - Composition in the same file
 - Mix {methods, functions, expressions}
 - Integrate scoping
- Arbitrary languages
 - For now, dynamic languages.
- Make the composition fast.

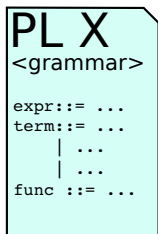
Breaking it Down



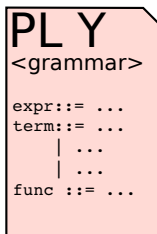
Breaking it Down



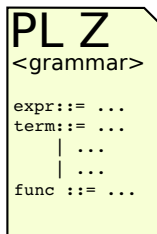
Composing Syntax



U



=

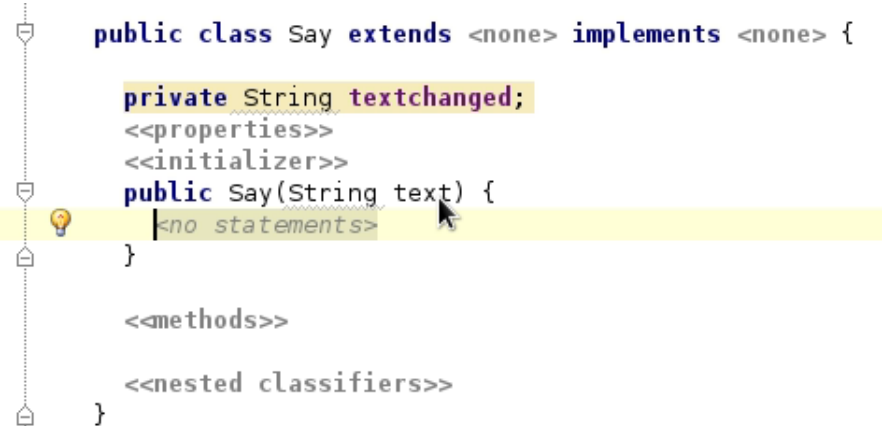


Easy?

Composing Syntax

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

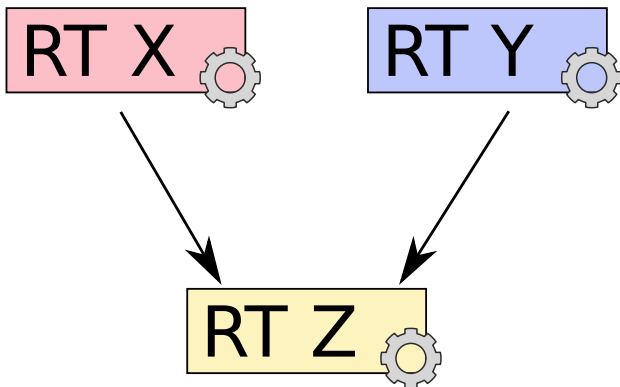
Syntax Directed Editing



```
public class Say extends <none> implements <none> {  
    private String textchanged;  
    <<properties>>  
    <<initializer>>  
    public Say(String text) {  
        <no statements>  
    }  
  
    <<methods>>  
  
    <<nested classifiers>>  
}
```

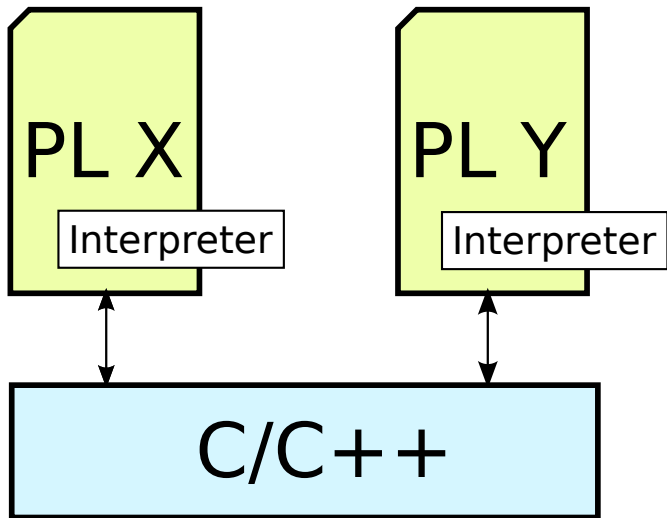
Poor editing experience.

Composing Runtimes

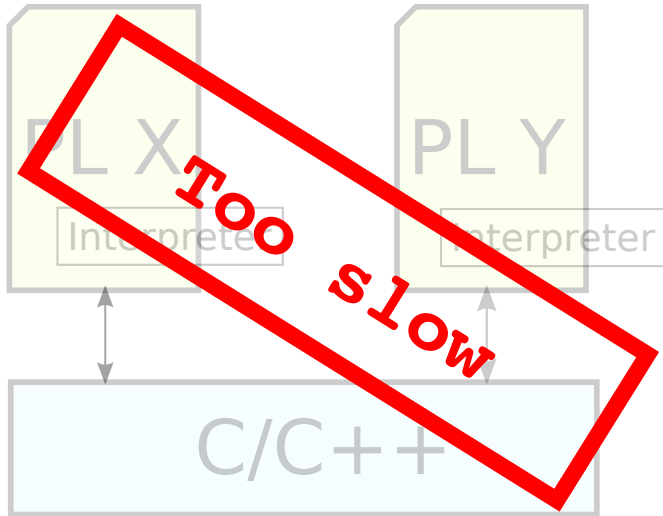


Easy?

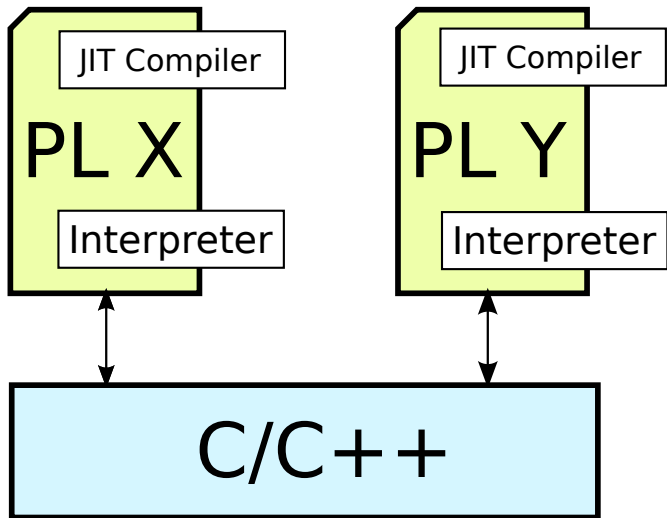
Runtime composition



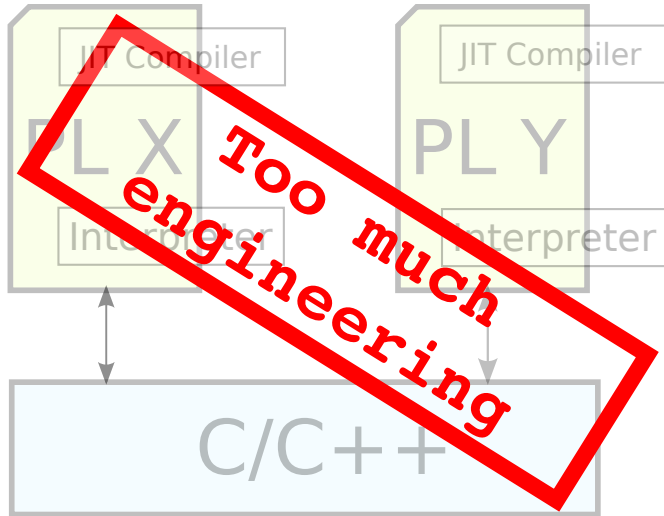
Runtime composition



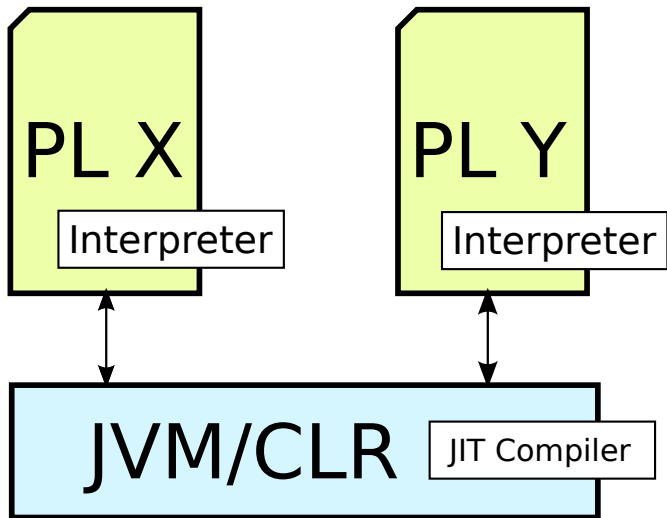
Runtime composition



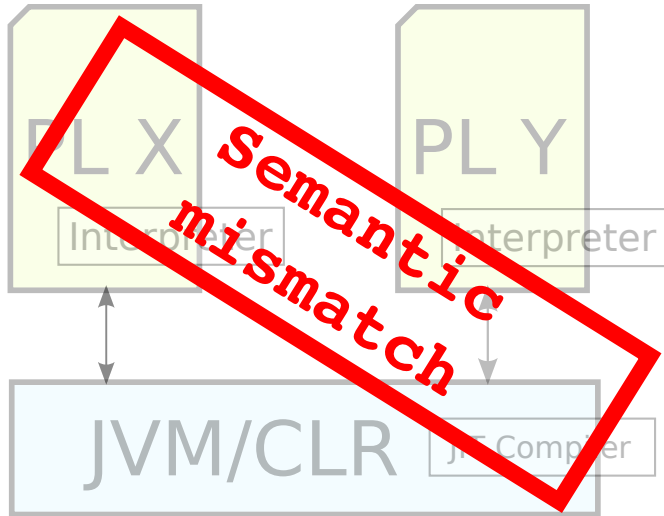
Runtime composition



Runtime composition



Runtime composition



Our Approach

Summary:

We need a practical way of composing syntax and runtimes.

Our Approach

Summary:

We need a practical way of composing syntax and runtimes.



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

An arrow with a black outline and a black arrowhead, pointing from the text 'Open SQL language box' up and to the left towards the light green box.

Language Boxes: E.g. Java + SQL

Write SQL code




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

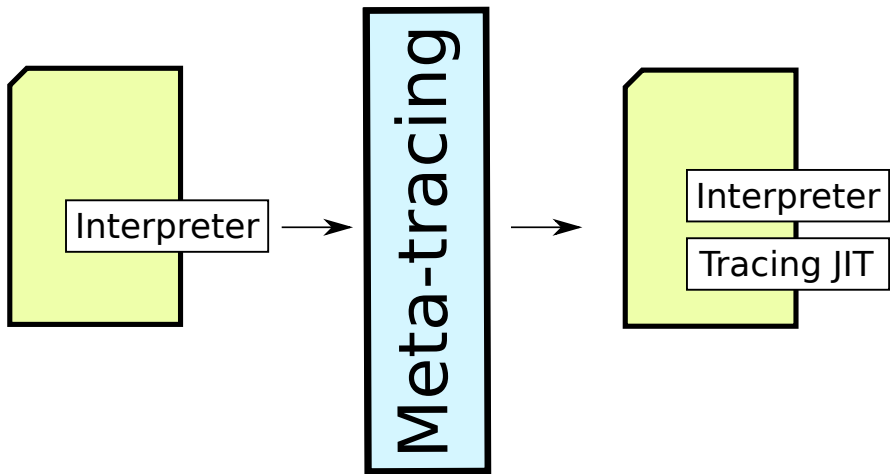
Language Boxes: E.g. Java + SQL

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

Java code

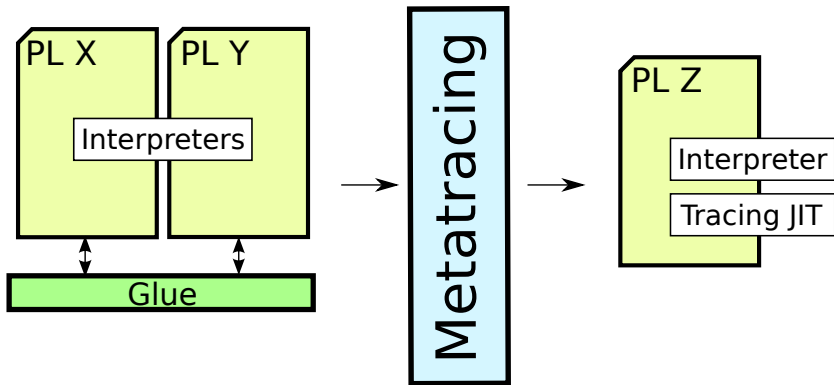


Meta-tracing



How Does this Apply to VM Composition?

How Does this Apply to VM Composition?



Summarising our Approach

- Editing with Language boxes.
 - Traditional “code editor” look and feel.
 - Practical syntactic composition.

- Interpreter Composition with Meta-tracing
 - Relatively little engineering effort.
 - Compose any two languages written in RPython.
 - Language agnostic JIT optimisations.

Our Compositions

Eco + RPython

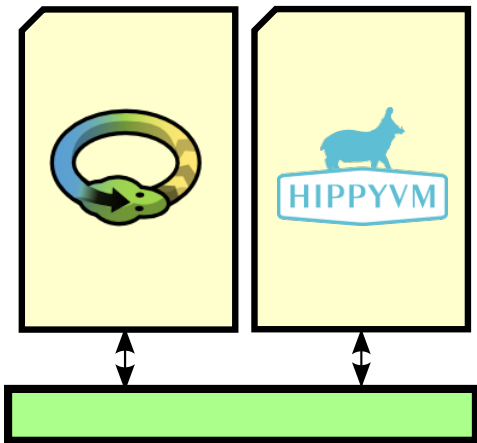
Our Language Compositions

- Python + Prolog
- Python + PHP
- Python + SQLite

Our Language Compositions

- Python + Prolog
- Python + PHP = PyHyp
- Python + SQLite

PyHyp

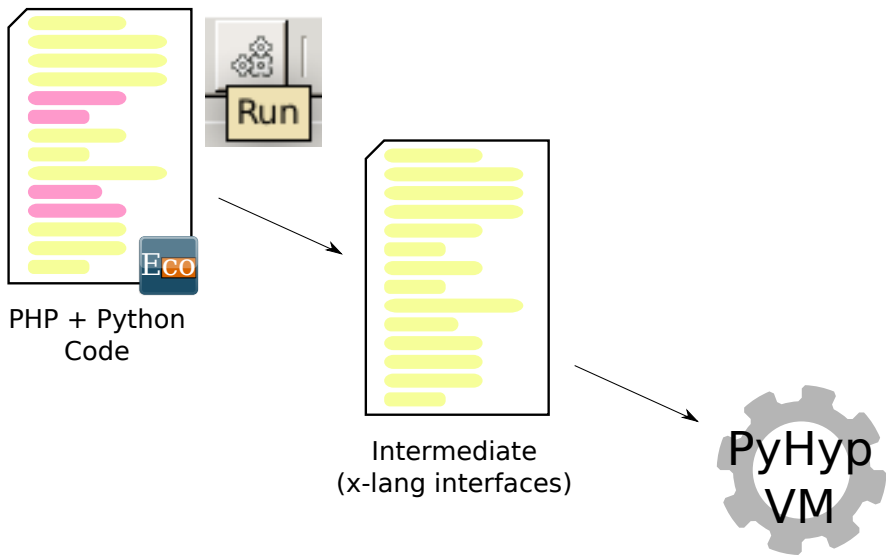


Features of PyHyp

- FFI-like features
 - Calling Python functions and methods from PHP
 - Calling PHP functions and methods from Python
 - Automatic type “conversion”

- Advanced features
 - Adds support for references to Python
 - Arbitrary nesting of foreign functions
 - Cross-language scoping
 - Python expressions in PHP
 - “Embedding” Python methods inside PHP classes
 - Access modifiers

Edit/Execute



“Semantic Friction”

Implementing desired behaviour: relatively easy

Deciding the correct behaviours: hard

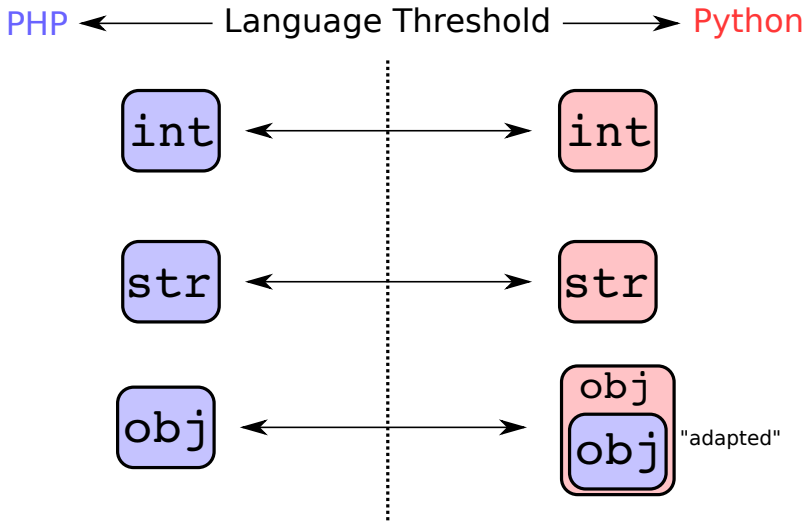
“Semantic friction”

Compromises sometimes must be made.

“Semantic Friction”

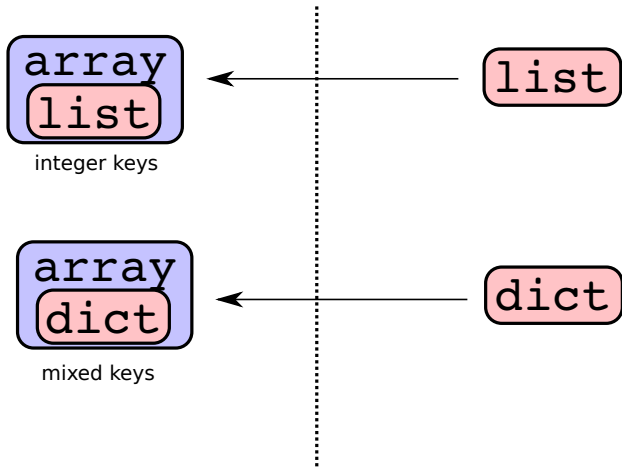
Example: Collection types across languages.

Semantic Friction: Array/Dict/List Conversions



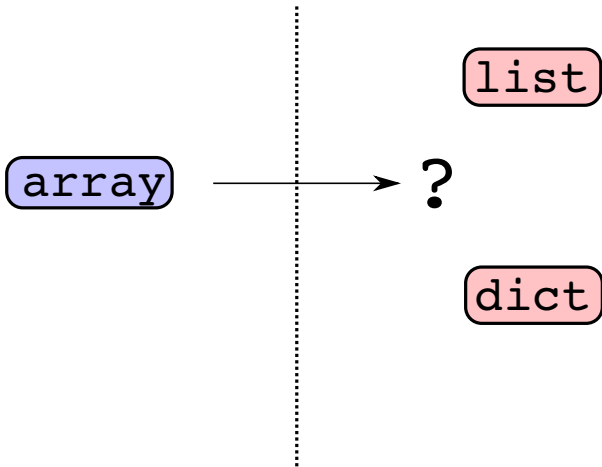
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



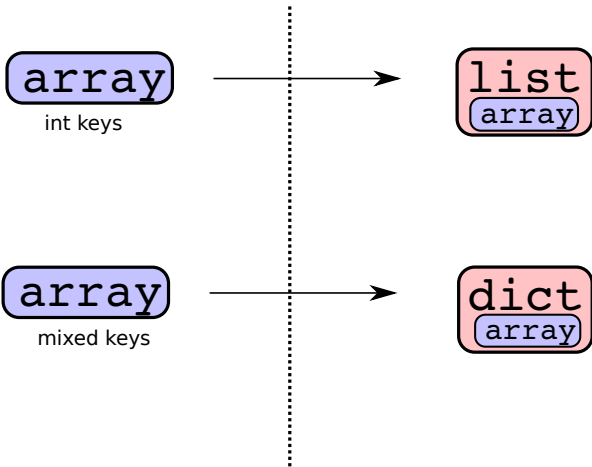
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



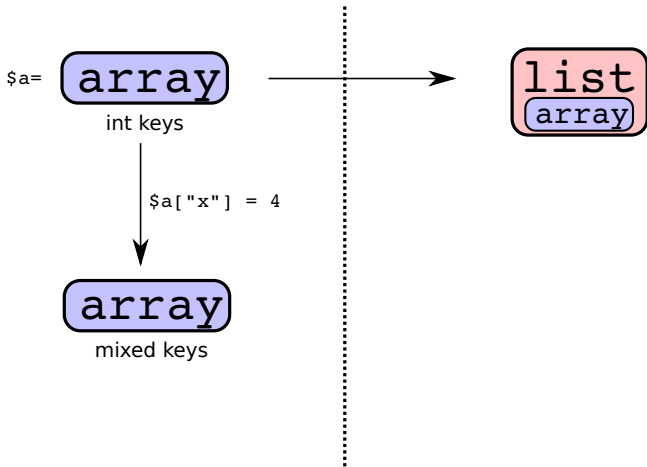
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



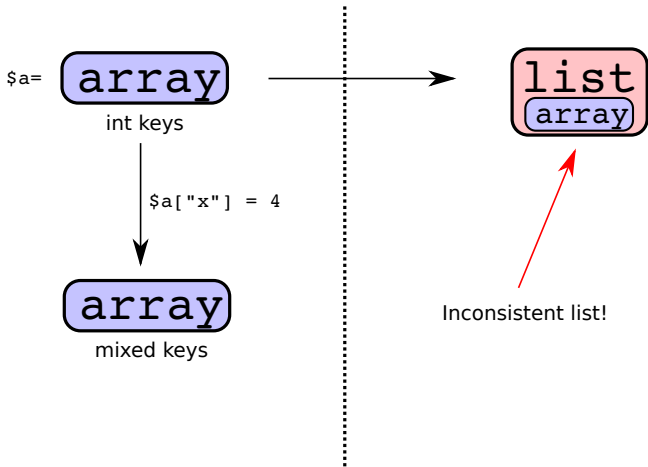
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



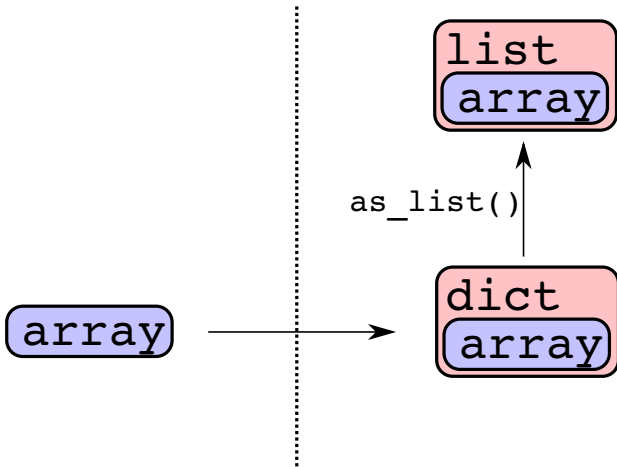
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python



Experimental Evaluation

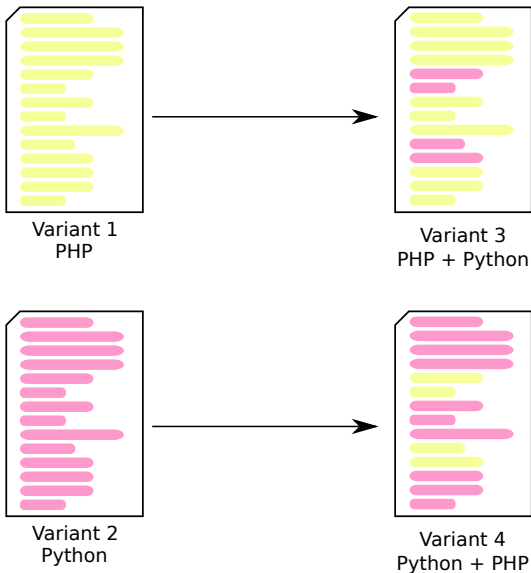
Benchmarks

Microbenchmarks

“Larger” benchmarks

Benchmark Variants

Benchmarks



“Good Performance”

Composed variant on PyHyp should perform “close” to mono variants on constituent interpreters.

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

For completeness, benchmark against other PHP and Python implementations too.

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 ±0.1493	6.698 ±0.0188	1.206 ±0.0032	1.218 ±0.0035	0.785 ±0.0024	56.521 ±0.1833



Conclusions

- Language boxes:
 - Practical composition of PL syntax.
 - 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.
 - Thanks to semantic friction.

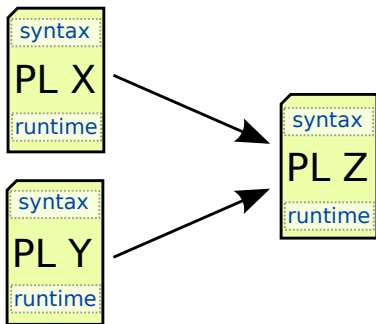
Future Work

- Debugging
 - Proper backtrace information.
 - Cross-language debugger.

- Compositions with >2 languages involved.

- Statically typed languages.

Thanks



Language Boxes + Meta-tracing