

# Language integration and migration



Edd Barrett



Carl  
Friedrich  
Bolz



Lukas  
Diekmann



Laurence  
Tratt



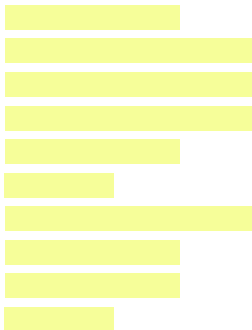
Naveneetha  
Krishnan  
Vasudevan

**KING'S**  
*College*  
**LONDON**

Software Development Team  
2014-10-22

# What to expect from this talk

A



B



# What to expect from this talk

A U B



# What to expect from this talk

## Python $\cup$ Prolog



# What to expect from this talk

## Python $\cup$ PHP



# Our problem

# Our problem

We want **better** programming languages

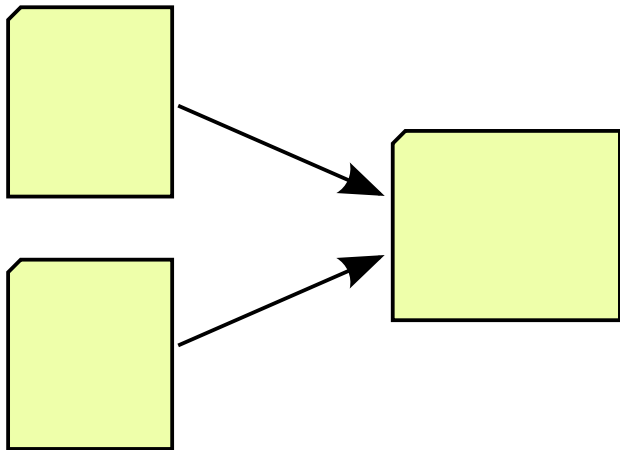
## Our problem

We want **better** programming languages

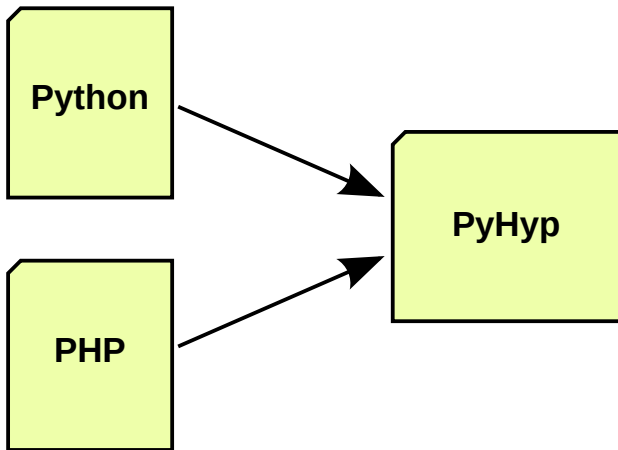
But better always seems to end up **bigger**



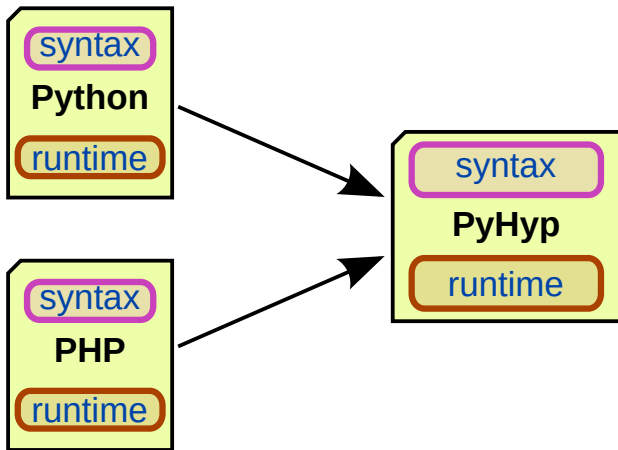
# Underlying language composition challenges



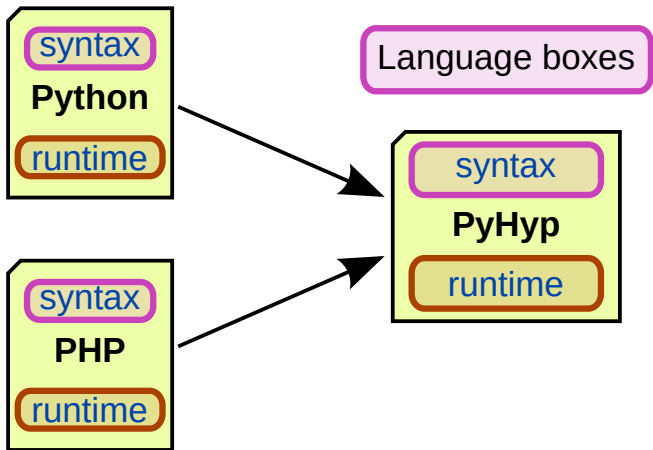
# Underlying language composition challenges



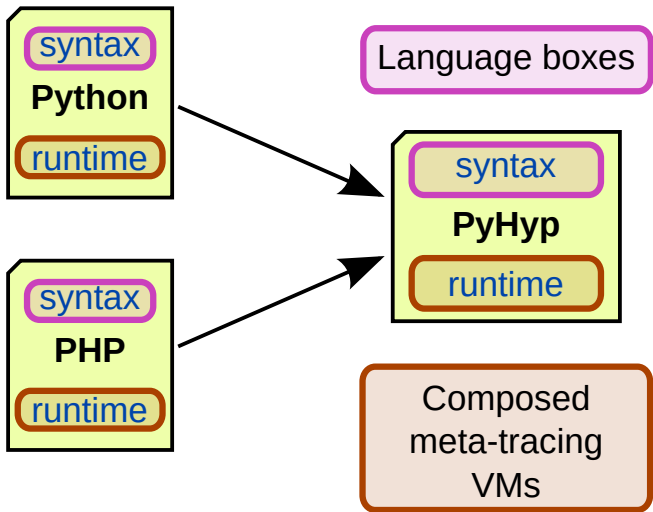
# Underlying language composition challenges



# Underlying language composition challenges



# Underlying language composition challenges



# Syntax composition

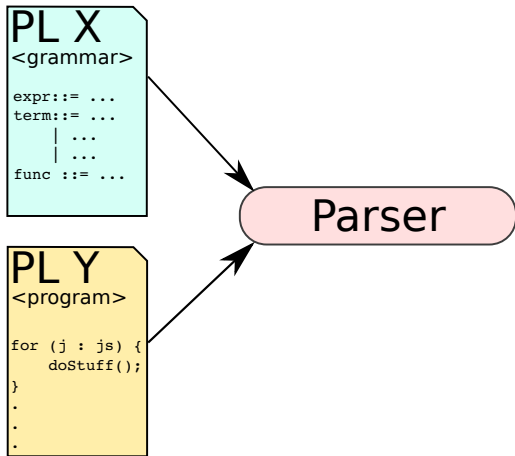
**PL X**  
<grammar>

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

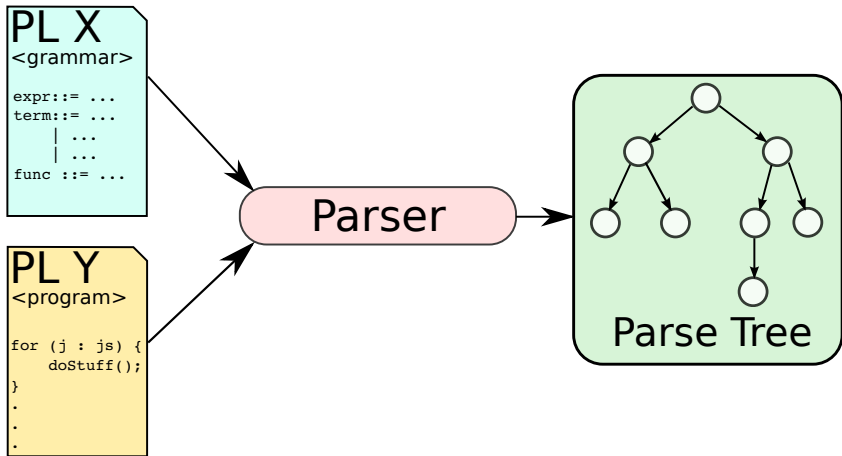
**PL Y**  
<program>

```
for (j : js) {  
    doStuff();  
}  
.  
.  
.
```

# Syntax composition

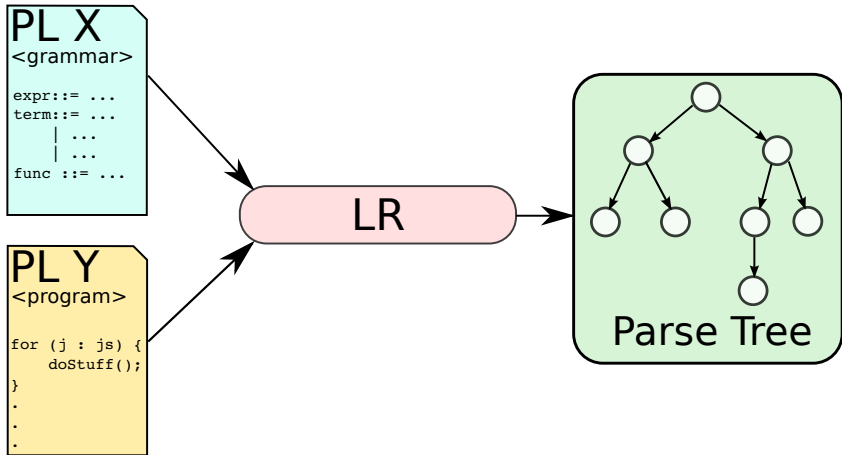


# Syntax composition

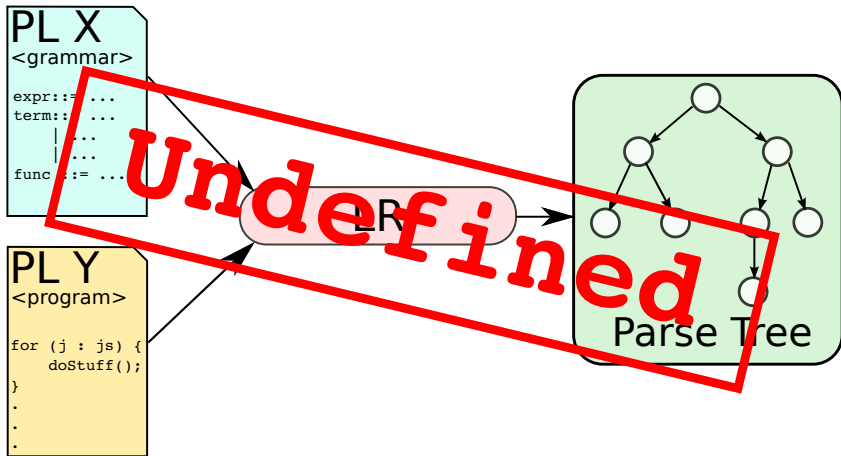




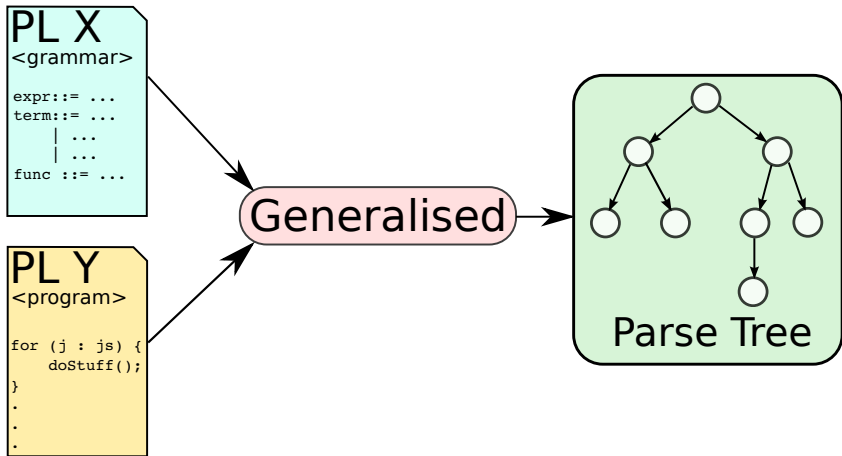
# Syntax composition



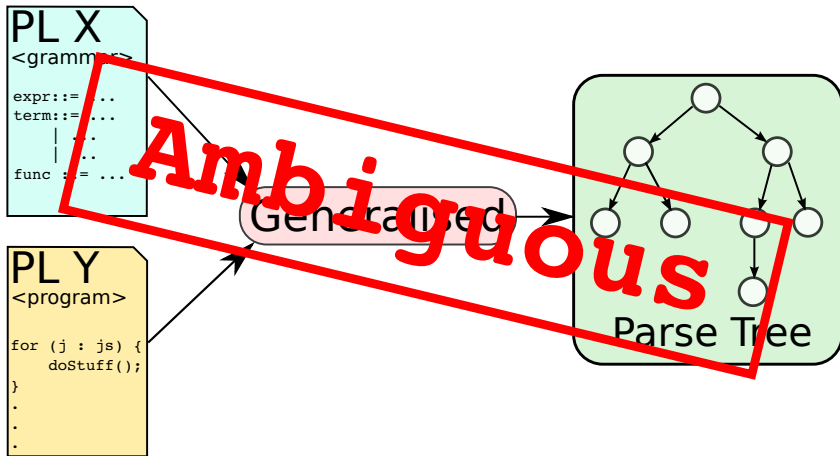
# Syntax composition



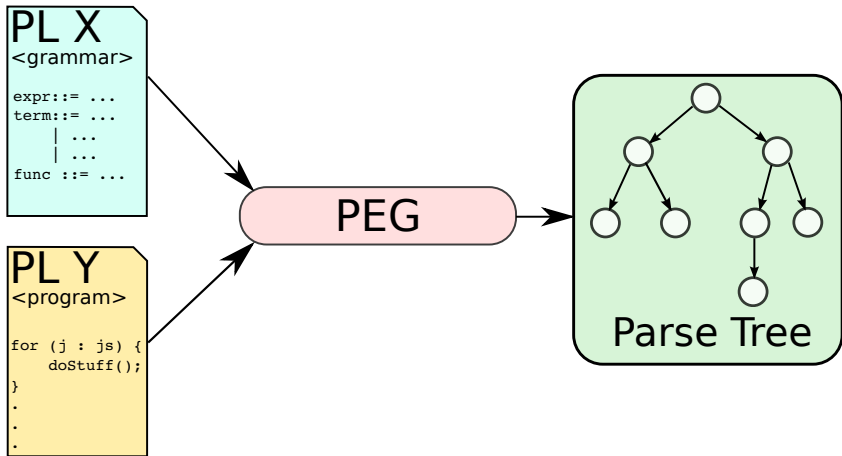
# Syntax composition



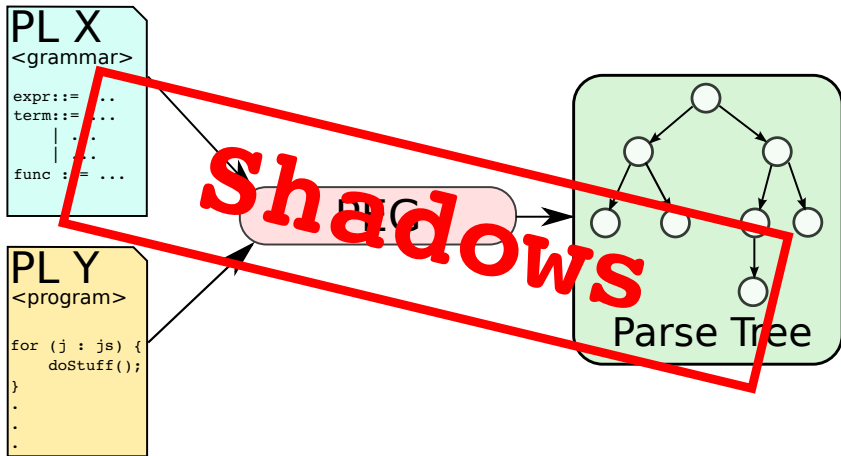
# Syntax composition



# Syntax composition



# Syntax composition



# The only choice?

The only choice?

SDE

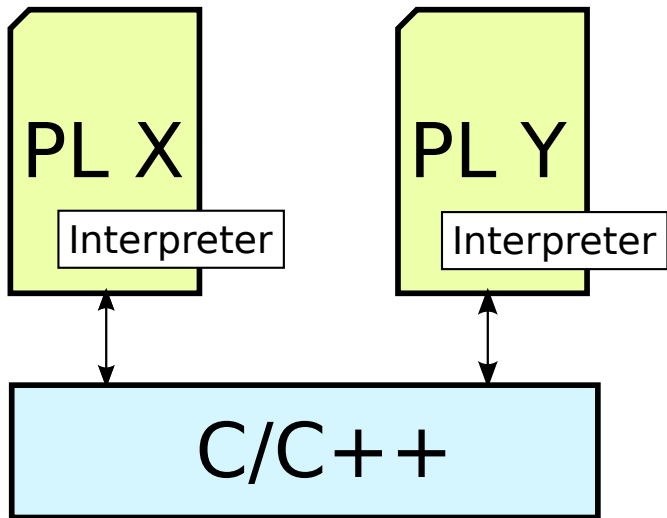


Challenge:  
SDE's power +  
a text editor feel?

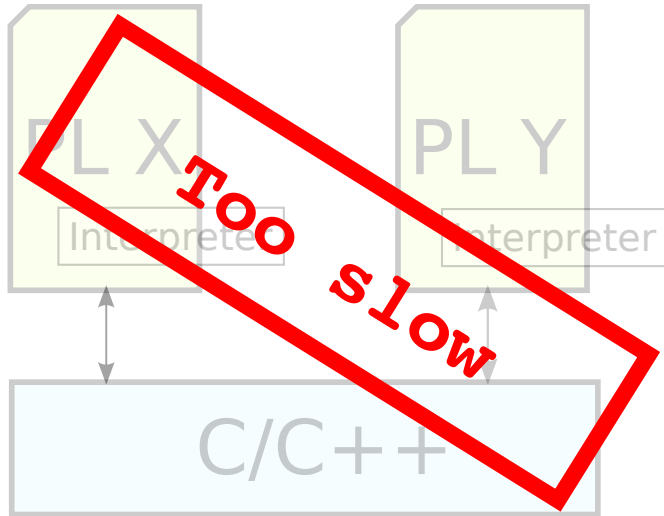
# Eco demo

# Runtime composition

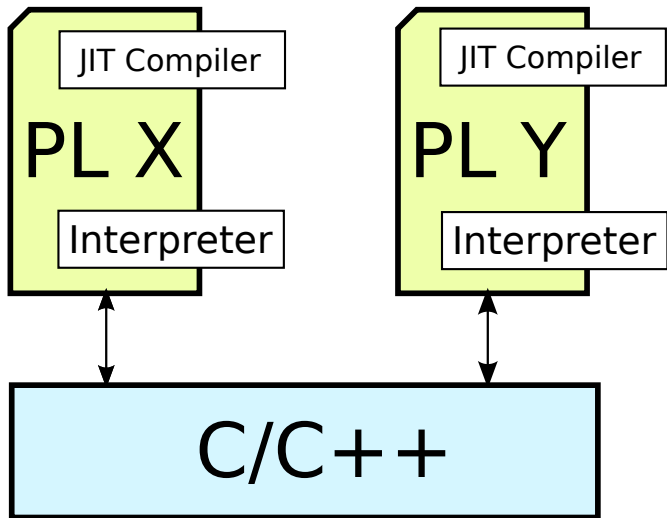
# Runtime composition



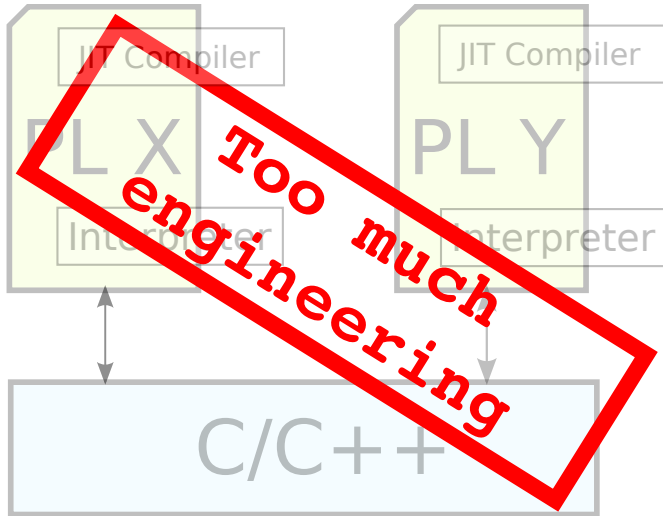
# Runtime composition



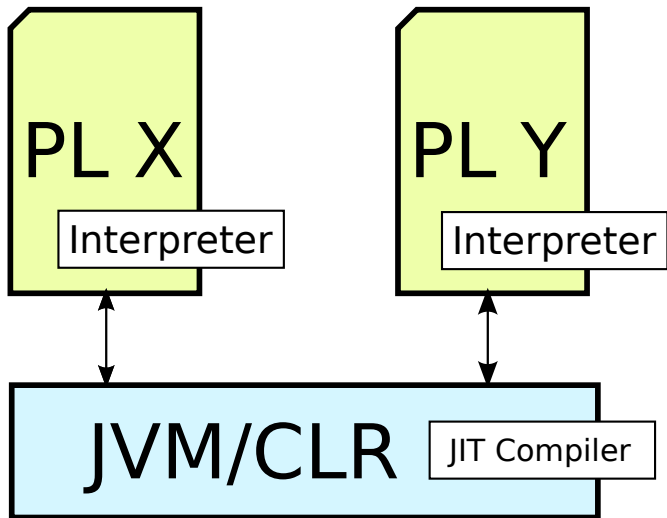
# Runtime composition



# Runtime composition

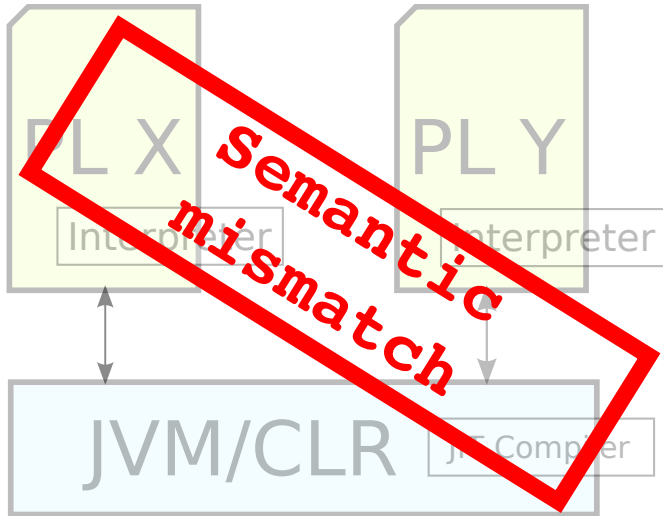


# Runtime composition



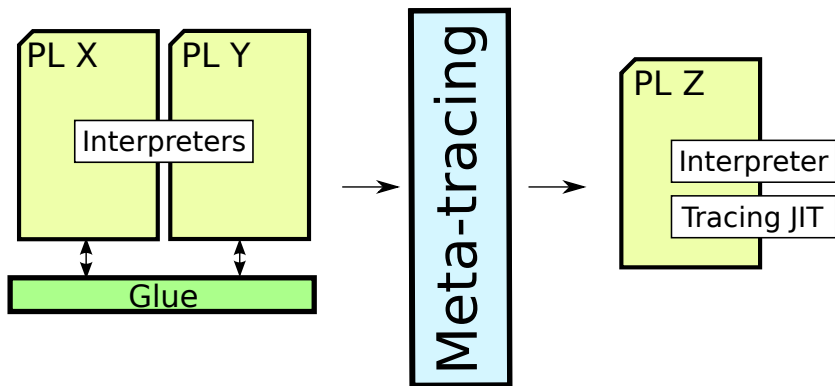


# Runtime composition



# Runtime composition

# Runtime composition

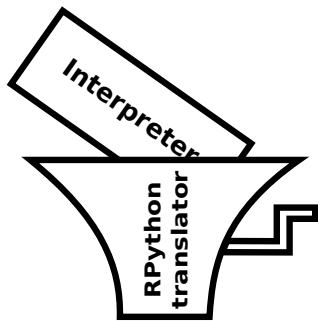


# Meta-tracing translation with RPython

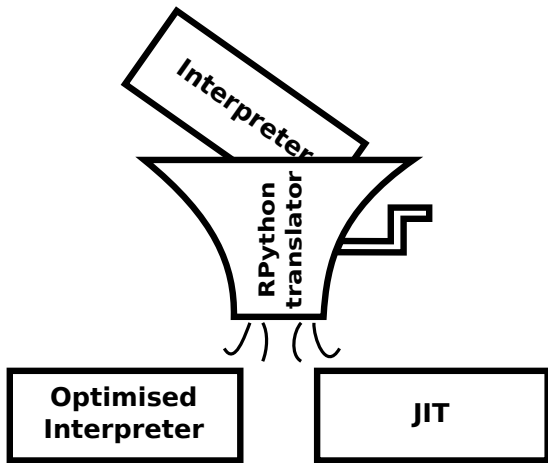


**Interpreter**

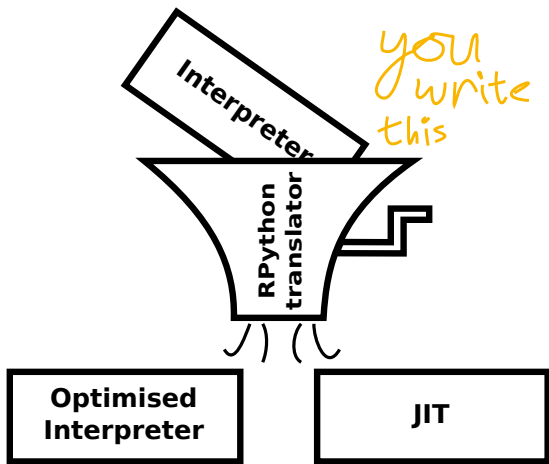
# Meta-tracing translation with RPython



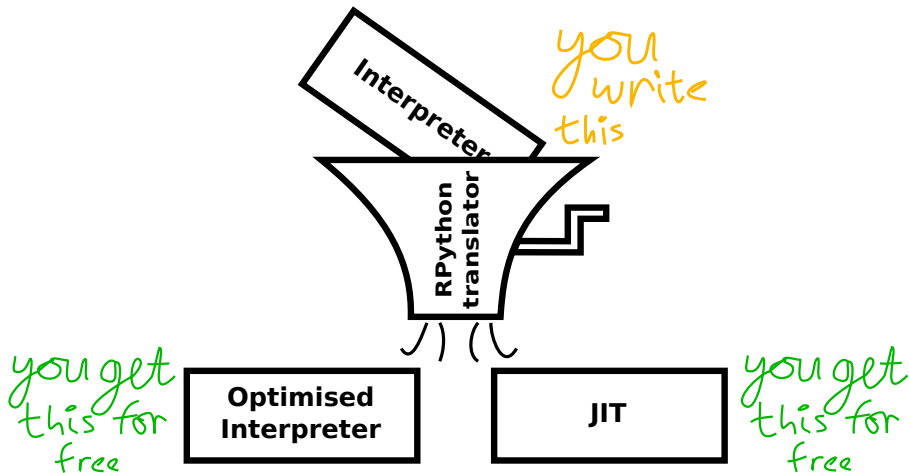
# Meta-tracing translation with RPython



# Meta-tracing translation with RPython



# Meta-tracing translation with RPython





# Adding a JIT to an RPython interpreter

```
...
pc := 0
while 1:

    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)

        pc += off
    elif ...:
        ...
```

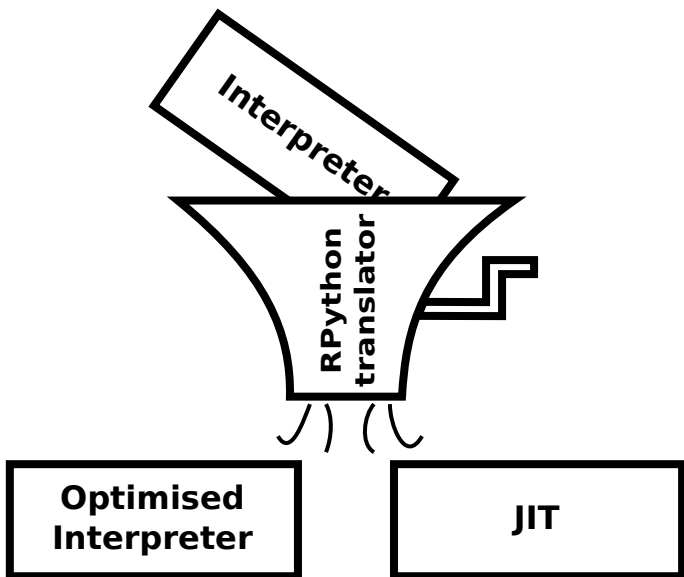
Observation: interpreters are big loops.

# Adding a JIT to an RPython interpreter

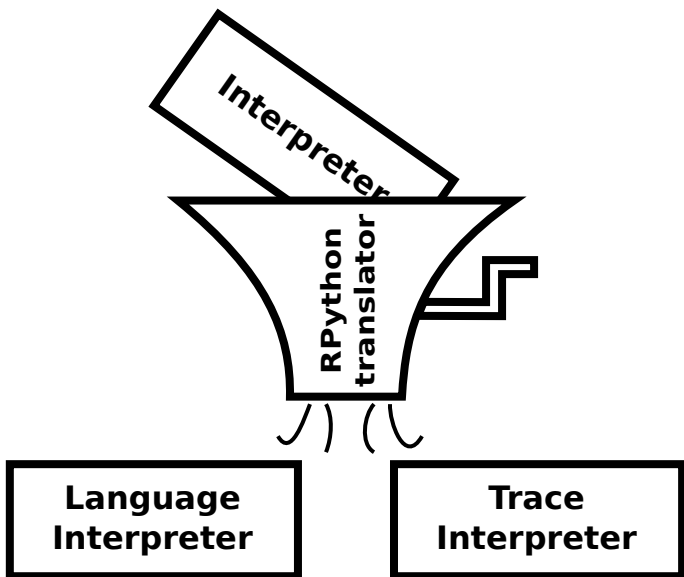
```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

Observation: interpreters are big loops.

# RPython translation



# RPython translation



---

## User program (lang *FL*)

---

```
if x < 0:  
    x = x + 1  
else:  
    x = x + 2  
x = x + 3
```

---

# Tracing JITs

---

User program (lang *FL*)

---

Trace when x is set to 6

---

```
if x < 0:  
    x = x + 1  
else:  
    x = x + 2  
x = x + 3
```

```
guard_type(x, int)  
guard_not_less_than(x, 0)  
guard_type(x, int)  
x = int_add(x, 2)  
guard_type(x, int)  
x = int_add(x, 3)
```

---

# Tracing JITs

---

User program (lang *FL*)

---

```
if x < 0:  
    x = x + 1  
else:  
    x = x + 2  
x = x + 3
```

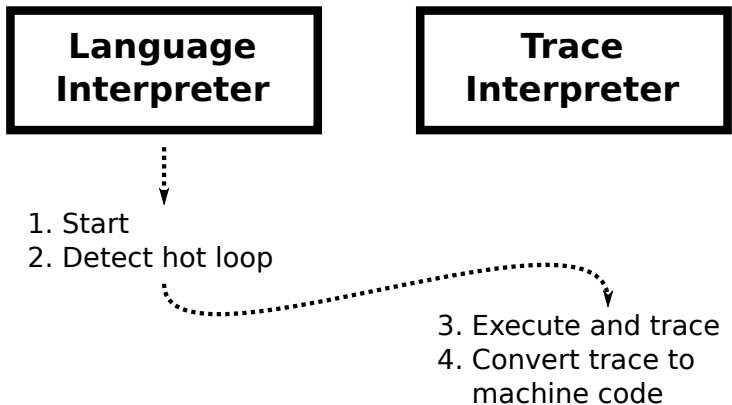
Optimised trace

---

```
guard_type(x, int)  
guard_not_less_than(x, 0)  
x = int_add(x, 5)
```

---

# Meta-tracing VM components





---

## FL Interpreter

---

```
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
        = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
    program_counter += 1

elif instr == INSTR_IF:
    result = stack.pop()
    if result == True:
        program_counter += 1
    else:
        program_counter +=
            read_jump_if_instruction()
elif instr == INSTR_ADD:
    lhs = stack.pop()
    rhs = stack.pop()
    if isinstance(lhs, int)
    and isinstance(rhs, int):
        stack.push(lhs + rhs)
    else: ...
    program_counter += 1
```

---

## FL Interpreter

---

```
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
            = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
    program_counter += 1
```

---

# Meta-tracing JITs

---

## FL Interpreter

```
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
            = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
    program_counter += 1
```

---

## User program (lang FL)

```
if x < 0:
    x = x + 1
else:
    x = x + 2
x = x + 3
```

# Meta-tracing JITs

---

## FL Interpreter

---

```
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
            = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
    program_counter += 1
```

---

## Initial trace

---

```
v0 = <program_counter>
v1 = <stack>
v2 = <vars>
v3 = load_instruction(v0)
guard_eq(v3, INSTR_VAR_GET)
v4 = dict_get(v2, "x")
list_append(v1, v4)
v5 = add(v0, 1)
v6 = load_instruction(v5)
guard_eq(v6, INSTR_INT)
list_append(v1, 0)
v7 = add(v5, 1)
v8 = load_instruction(v7)
guard_eq(v8, INSTR_LESS_THAN)
v9 = list_pop(v1)
v10 = list_pop(v1)
guard_type(v9, int)
guard_type(v10, int)
guard_not_less_than(v9, v10)
list_append(v1, False)
v11 = add(v7, 1)
v12 = load_instruction(v11)
guard_eq(v12, INSTR_IF)
v13 = list_pop(v1)
guard_false(v13)
...
```

---

---

## Initial trace in full

---

```
v0 = <program_counter>
v1 = <stack>
v2 = <vars>
v3 = load_instruction(v0)
guard_eq(v3, INSTR_VAR_GET)
v4 = dict_get(v2, "x")
list_append(v1, v4)
v5 = add(v0, 1)
v6 = load_instruction(v5)
guard_eq(v6, INSTR_INT)
list_append(v1, 0)
v7 = add(v5, 1)
v8 = load_instruction(v7)
guard_eq(v8, INSTR_LESS_THAN)
v9 = list_pop(v1)
v10 = list_pop(v1)
guard_type(v9, int)
guard_type(v10, int)
guard_not_less_than(v9, v10)
list_append(v1, False)
v11 = add(v7, 1)
v12 = load_instruction(v11)
guard_eq(v12, INSTR_IF)
v13 = list_pop(v1)
guard_false(v13)
v14 = add(v11, 2)

v15 = load_instruction(v14)
guard_eq(v15, INSTR_VAR_GET)
v16 = dict_get(v2, "x")
list_append(v1, v16)
v17 = add(v14, 1)
v18 = load_instruction(v17)
guard_eq(v18, INSTR_INT)
list_append(v1, 2)
v19 = add(v17, 1)
v20 = load_instruction(v19)
guard_eq(v20, INSTR_ADD)
v21 = list_pop(v1)
v22 = list_pop(v1)
guard_type(v21, int)
guard_type(v22, int)
v23 = add(v22, v21)
list_append(v1, v23)
v24 = add(v19, 1)
v25 = load_instruction(v24)
guard_eq(v25, INSTR_VAR_SET)
v26 = list_pop(v1)
dict_set(v2, "x", v26)
v27 = add(v24, 1)
v28 = load_instruction(v27)
guard_eq(v28, INSTR_VAR_GET)
v29 = dict_get(v2, "x")

list_append(v1, v29)
v30 = add(v27, 1)
v31 = load_instruction(v30)
guard_eq(v31, INSTR_INT)
list_append(v1, 3)
v32 = add(v30, 1)
v33 = load_instruction(v32)
guard_eq(v33, INSTR_ADD)
v34 = list_pop(v1)
v35 = list_pop(v1)
guard_type(v34, int)
guard_type(v35, int)
v36 = add(v35, v34)
list_append(v1, v36)
v37 = add(v32, 1)
v38 = load_instruction(v37)
guard_eq(v38, INSTR_VAR_SET)
v39 = list_pop(v1)
dict_set(v2, "x", v39)
v40 = add(v37, 1)
```

---

# Trace optimisation (1)

---

## Removing constants (from jit\_merge\_point)

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
list_append(v1, v4)
list_append(v1, 0)
v9 = list_pop(v1)
v10 = list_pop(v1)
guard_type(v9, int)
guard_type(v10, int)
guard_not_less_than(v9, v10)
list_append(v1, False)
v13 = list_pop(v1)
guard_false(v13)
v16 = dict_get(v2, "x")
list_append(v1, v16)
list_append(v1, 2)
v21 = list_pop(v1)
v22 = list_pop(v1)
guard_type(v21, int)
guard_type(v22, int)
v23 = add(v22, v21)
list_append(v1, v23)
v26 = list_pop(v1)
dict_set(v2, "x", v26)
v29 = dict_get(v2, "x")
list_append(v1, v29)

list_append(v1, 3)
v34 = list_pop(v1)
v35 = list_pop(v1)
guard_type(v34, int)
guard_type(v35, int)
v36 = add(v35, v34)
list_append(v1, v36)
v39 = list_pop(v1)
dict_set(v2, "x", v39)
```

---

# Optimisation #2 & #3

---

## List folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v16 = dict_get(v2, "x")
guard_type(v16, int)
v23 = add(v16, 2)
dict_set(v2, "x", v23)
v29 = dict_get(v2, "x")
guard_type(v29, int)
v36 = add(v29, 3)
dict_set(v2, "x", v36)
```

---

# Optimisation #2 & #3

---

## List folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v16 = dict_get(v2, "x")
guard_type(v16, int)
v23 = add(v16, 2)
dict_set(v2, "x", v23)
v29 = dict_get(v2, "x")
guard_type(v29, int)
v36 = add(v29, 3)
dict_set(v2, "x", v36)
```

---

## Dict folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 2)
guard_type(v23, int)
v36 = add(v23, 3)
dict_set(v2, "x", v36)
```

---



# Optimisation #4 & #5

---

## Type folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 2)
v36 = add(v23, 3)
dict_set(v2, "x", v36)
```

---

# Optimisation #4 & #5

---

## Type folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 2)
v36 = add(v23, 3)
dict_set(v2, "x", v36)
```

---

## Arithmetic folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 5)
dict_set(v2, "x", v23)
```

---

# Optimisation #4 & #5

---

## Type folded trace

---

```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 2)
v36 = add(v23, 3)
dict_set(v2, "x", v36)
```

---

## Arithmetic folded trace

---

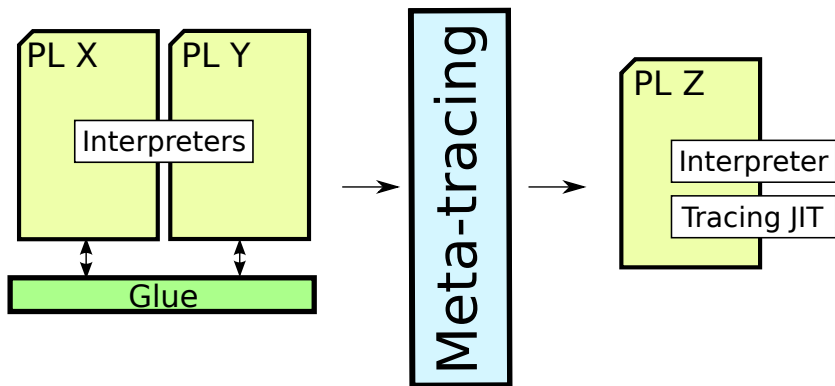
```
v1 = <stack>
v2 = <vars>
v4 = dict_get(v2, "x")
guard_type(v4, int)
guard_not_less_than(v4, 0)
v23 = add(v4, 5)
dict_set(v2, "x", v23)
```

---

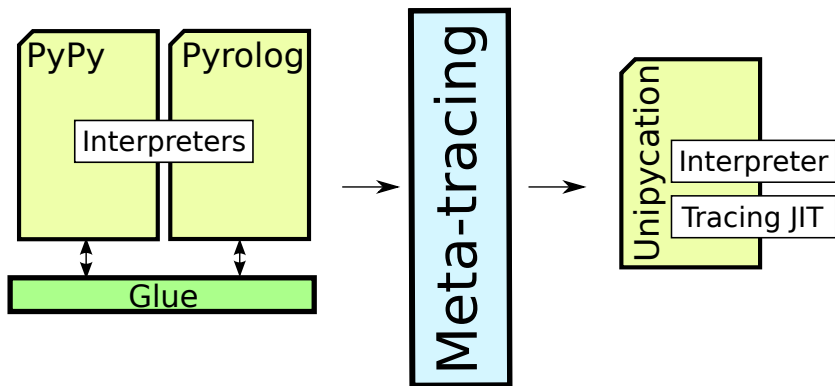
Trace optimisation: from 72 trace elements to 7.

# Runtime composition recap

# Runtime composition recap



# Runtime composition recap



# Unipycation demo

Warning: draft numbers ahead



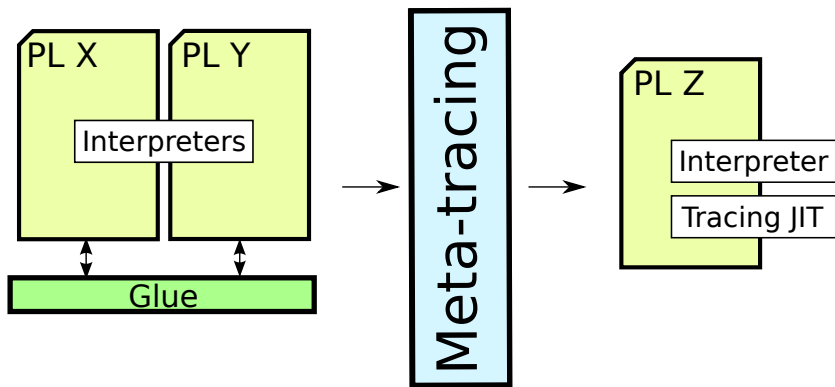
# Absolute timing comparison

VM	Benchmark	<i>Python</i>		<i>Prolog</i>		<i>Python</i> → <i>Prolog</i>	
CPython-SWI	SmallFunc	0.125s	±0.006	0.257s	±0.001	28.893s	±0.175
	Loop1Arg0Result	2.924s	±0.215	7.352s	±0.037	9.310s	±0.065
	Loop1Arg1Result	4.184s	±0.028	18.890s	±0.082	20.865s	±0.050
	NondetLoop1Arg1Result	7.531s	±0.065	18.643s	±0.159	667.682s	±5.594
	TermConstruction	264.415s	±1.815	48.819s	±0.208	2185.150s	±14.251
	Lists	9.374s	±0.046	25.148s	±0.182	2207.304s	±12.344
Unipycation	SmallFunc	0.001s	±0.000	0.006s	±0.001	0.001s	±0.000
	Loop1Arg0Result	0.085s	±0.000	0.086s	±0.000	0.087s	±0.000
	Loop1Arg1Result	0.112s	±0.000	0.114s	±0.000	0.115s	±0.000
	NondetLoop1Arg1Result	0.500s	±0.002	0.548s	±0.064	2.674s	±0.010
	TermConstruction	6.053s	±0.218	2.444s	±0.002	36.069s	±0.171
	Lists	0.845s	±0.002	1.416s	±0.003	5.056s	±0.026
Jython-tuProlog	SmallFunc	0.088s	±0.002	3.050s	±0.036	52.294s	±0.371
	Loop1Arg0Result	1.078s	±0.007	206.590s	±2.884	199.963s	±1.784
	Loop1Arg1Result	2.145s	±0.175	293.311s	±4.270	294.781s	±4.746
	NondetLoop1Arg1Result	7.939s	±0.341	timeout		timeout	
	TermConstruction	timeout		timeout		timeout	
	Lists	timeout		timeout		timeout	

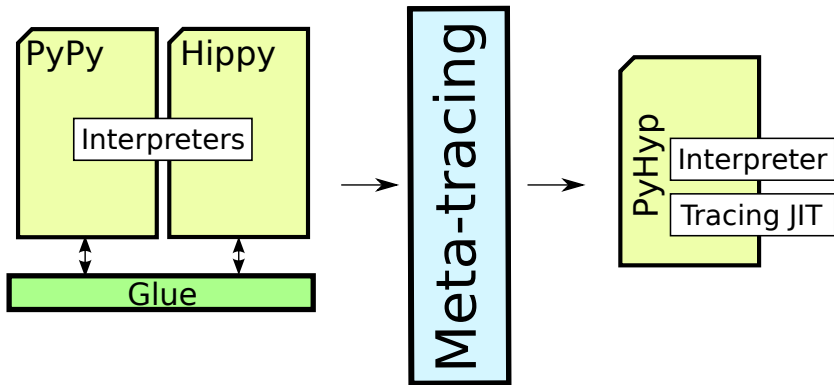
# Relative timing comparison

VM	Benchmark	$\frac{\text{Python} \rightarrow \text{Prolog}}{\text{Python}}$		$\frac{\text{Python} \rightarrow \text{Prolog}}{\text{Prolog}}$		$\frac{\text{Python} \rightarrow \text{Prolog}}{\text{Unipycation}}$	
CPython-SWI	SmallFunc	231.770 ×	±10.154	112.567 ×	±0.934	27821.079 ×	±1896.725
	Loop1Arg0Result	3.184 ×	±0.232	1.266 ×	±0.011	107.591 ×	±0.779
	Loop1Arg1Result	4.987 ×	±0.039	1.105 ×	±0.006	181.899 ×	±0.444
	NondetLoop1Arg1Result	88.654 ×	±1.026	35.814 ×	±0.389	249.737 ×	±2.244
	TermConstruction	8.264 ×	±0.081	44.760 ×	±0.348	60.583 ×	±0.487
	Lists	235.459 ×	±1.742	87.772 ×	±0.789	436.609 ×	±3.494
Unipycation	SmallFunc	1.295 ×	±0.086	0.182 ×	±0.036	1.000 ×	
	Loop1Arg0Result	1.020 ×	±0.001	1.012 ×	±0.002	1.000 ×	
	Loop1Arg1Result	1.025 ×	±0.002	1.002 ×	±0.002	1.000 ×	
	NondetLoop1Arg1Result	5.349 ×	±0.035	4.879 ×	±0.631	1.000 ×	
	TermConstruction	5.959 ×	±0.224	14.756 ×	±0.069	1.000 ×	
	Lists	5.982 ×	±0.034	3.569 ×	±0.019	1.000 ×	
Jython-tuProlog	SmallFunc	592.904 ×	±14.602	17.143 ×	±0.259	50354.204 ×	±3330.993
	Loop1Arg0Result	185.460 ×	±2.182	0.968 ×	±0.017	2310.844 ×	±21.996
	Loop1Arg1Result	137.427 ×	±11.805	1.005 ×	±0.022	2569.873 ×	±41.331
	NondetLoop1Arg1Result	timeout		timeout		timeout	
	TermConstruction	timeout		timeout		timeout	
	Lists	timeout		timeout		timeout	





# PyHyp



# PyHyp demo

Warning: even draftier numbers ahead!

# Composed Richards vs. other VMs

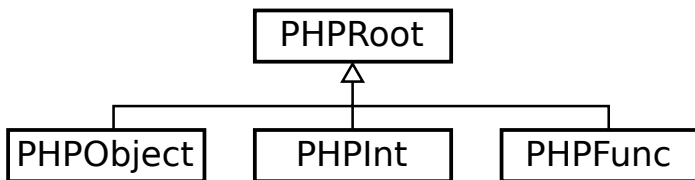
Type	VM	
Mono	PyPy 2.4.0	0.370 ± 0.000
	Hippy	0.553 ± 0.008
	PyHyp	0.556 ± 0.006
	HHVM 3.2.0	5.353 ± 0.262
	ZEND 5.4.4	10.406 ± 0.106



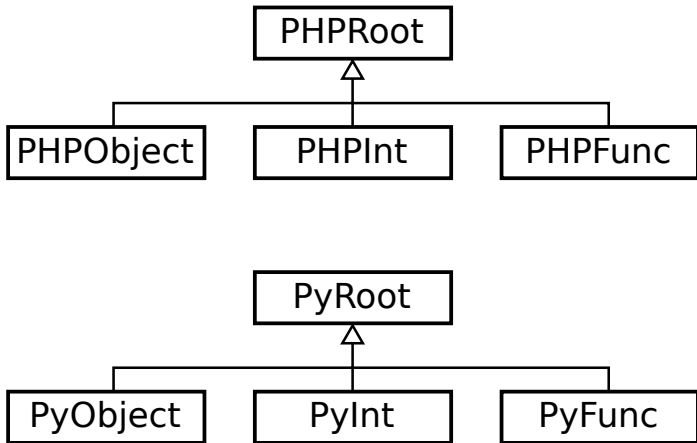
# Composed Richards vs. other VMs

Type	VM	
Mono	PyPy 2.4.0	$0.370 \pm 0.000$
	Hippy	$0.553 \pm 0.008$
	PyHyp	$0.556 \pm 0.006$
	HHVM 3.2.0	$5.353 \pm 0.262$
	ZEND 5.4.4	$10.406 \pm 0.105$
Composed	PyHyp	$0.936 \pm 0.038$

# Datatype conversion



# Datatype conversion



# Datatype conversion: primitive types

PHP

Python

# Datatype conversion: primitive types

PHP

2 : PHPInt

Python

# Datatype conversion: primitive types

PHP

2 : PHPInt

Python

2 : PyInt

# Datatype conversion: user types

PHP

Python



# Datatype conversion: user types

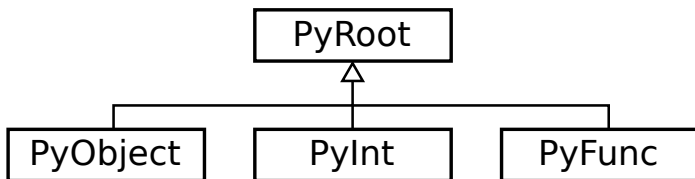
PHP

`o : PHPObject`

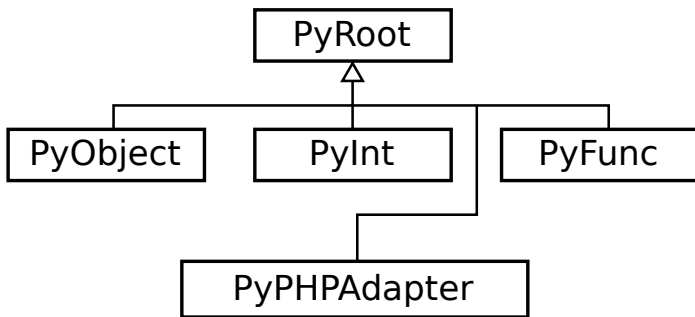
Python



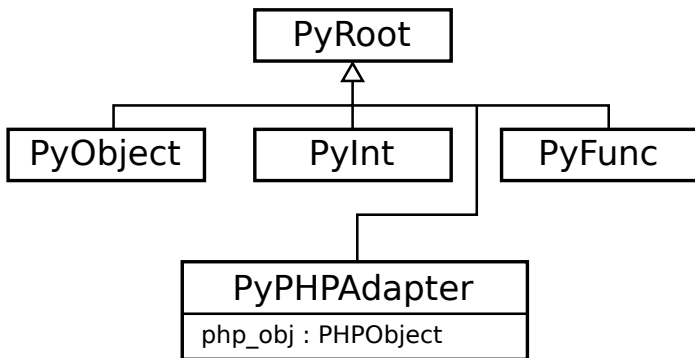
## Datatype conversion: user types



## Datatype conversion: user types



## Datatype conversion: user types



# Datatype conversion: user types

PHP

`o : PHPObject`

Python

# Datatype conversion: user types

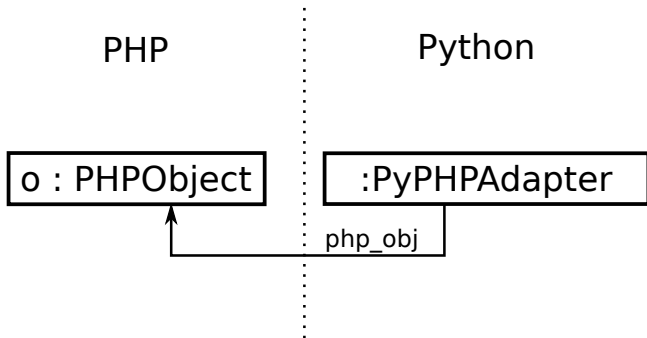
PHP

`o : PHPObject`

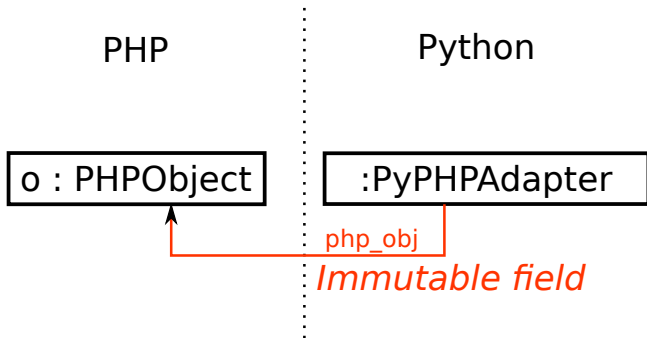
Python

`:PyPHPAdapter`

# Datatype conversion: user types



# Datatype conversion: user types



# Some thoughts

- Critical: single meta-language (e.g. RPython / Truffle).



# Some thoughts

- Critical: single meta-language (e.g. RPython / Truffle).
- Simplicity: good performance, yet understandable.

# Some thoughts

- Critical: single meta-language (e.g. RPython / Truffle).
- Simplicity: good performance, yet understandable.
- Immutable adapters give near-native performance.

# Some thoughts

- Critical: single meta-language (e.g. RPython / Truffle).
- Simplicity: good performance, yet understandable.
- Immutable adapters give near-native performance.
- **Whole new world of challenges for language designers & formalisers.**

# What can we use this for?

What can we use this for?

First-class languages

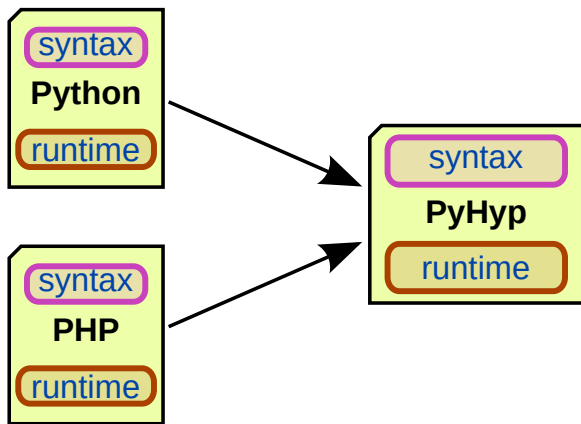
# What can we use this for?

First-class languages

Language migration

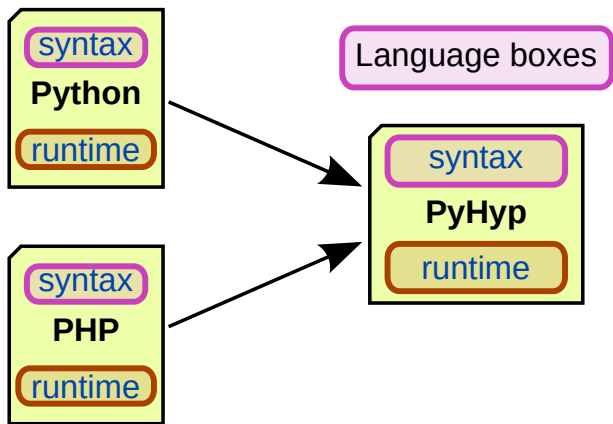
# Summary

# Summary

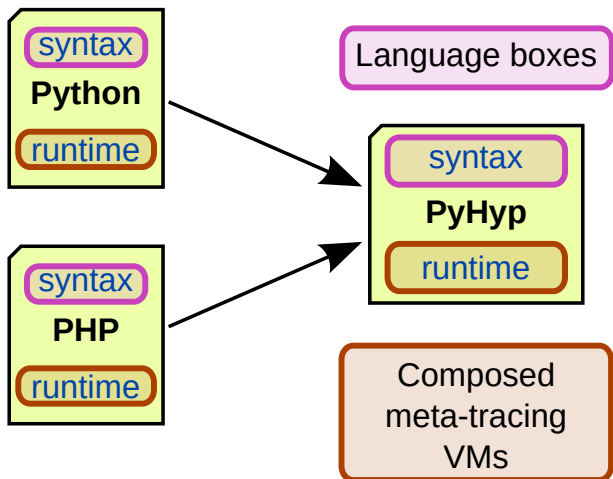




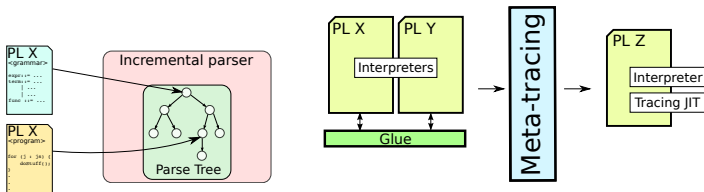
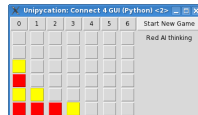
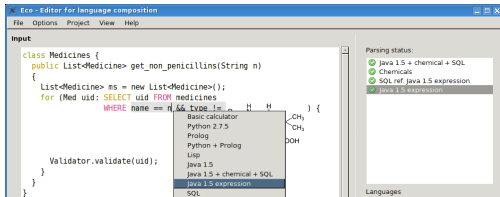
# Summary



# Summary



# Thanks for listening



<http://soft-dev.org/>