Fine-grained language composition without a common VM







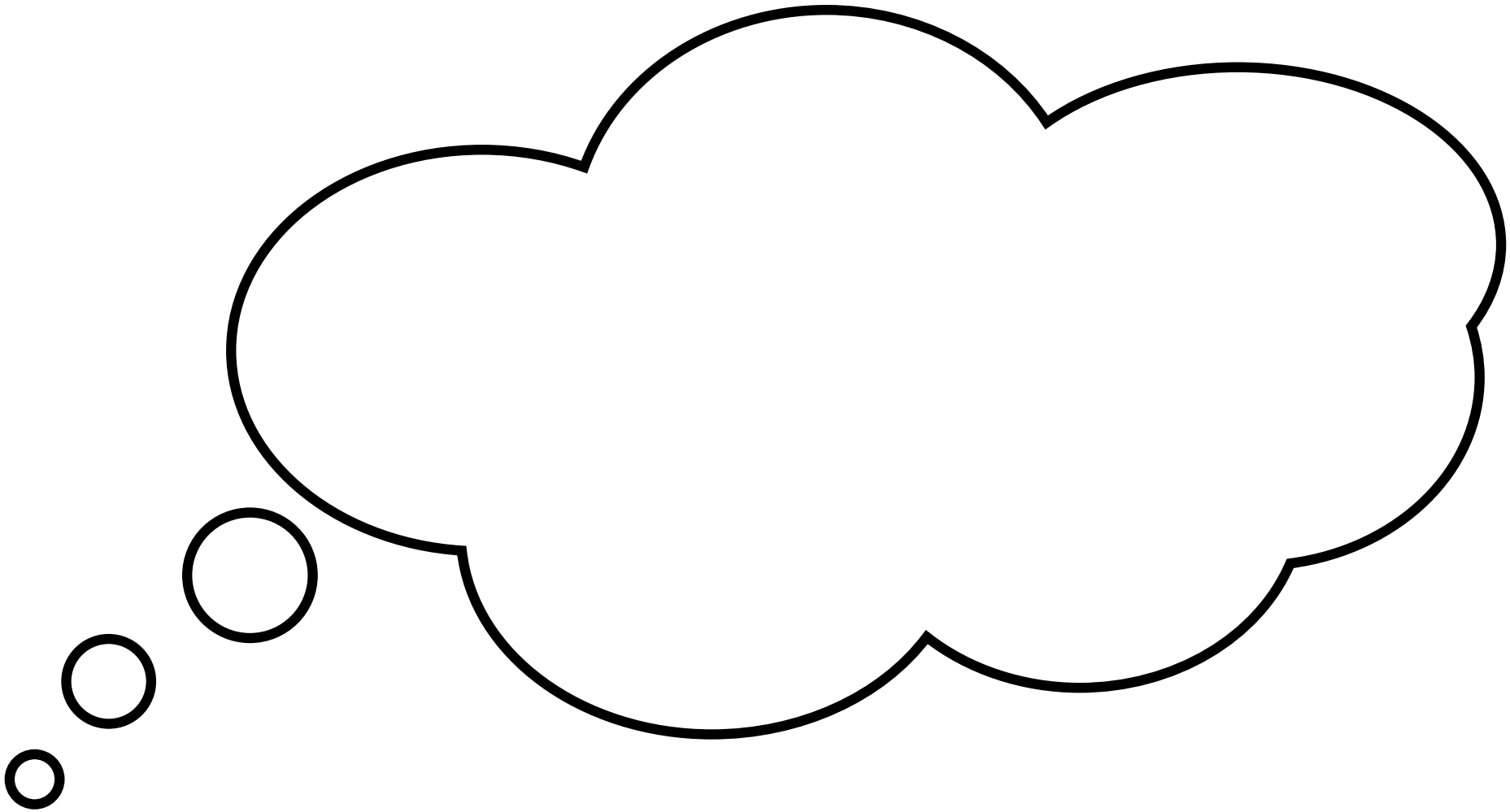
Edd Barrett, Carl Friedrich Bolz, Lukas Diekmann, Sarah Mount, Jasper Schulz, Laurence Tratt, Naveneetha Krishnan Vasudevan

KING'S College LONDON

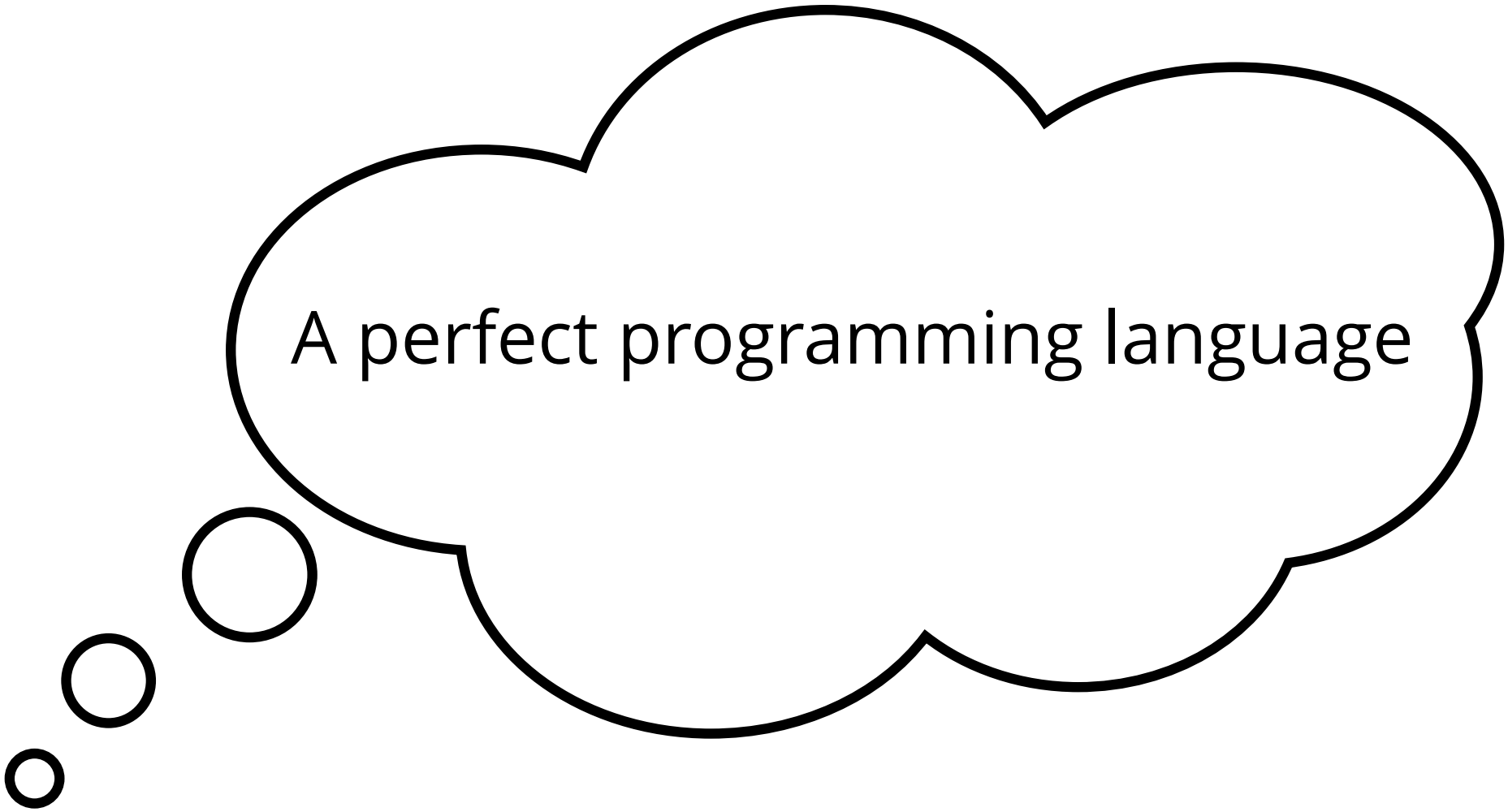
Software Development Team

2016-10-30

Background



Background



Solution

Solution

A new programming language

Reality

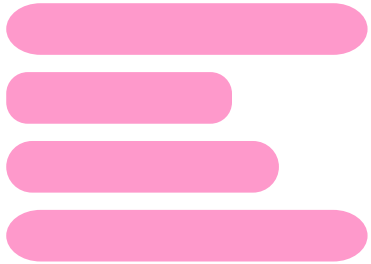
Reality

Another imperfect programming language

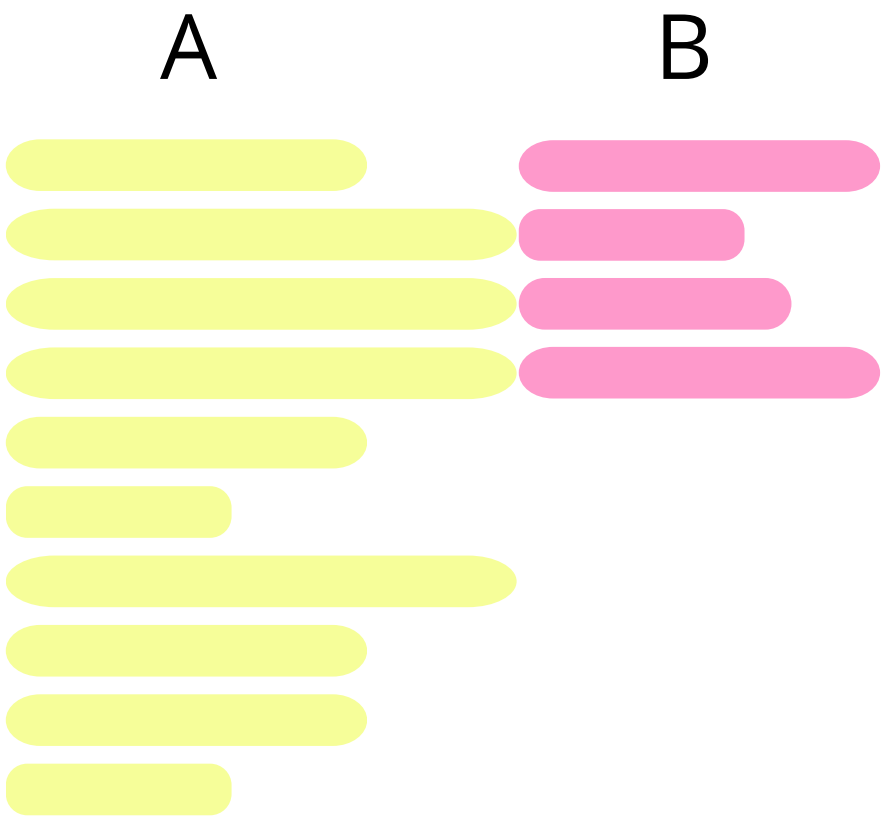
What to expect from this talk

A

B

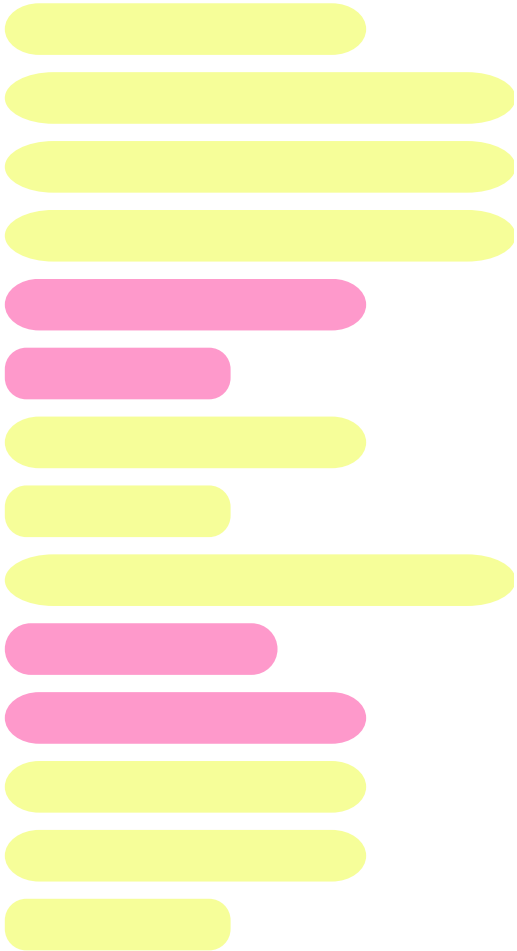


What to expect from this talk



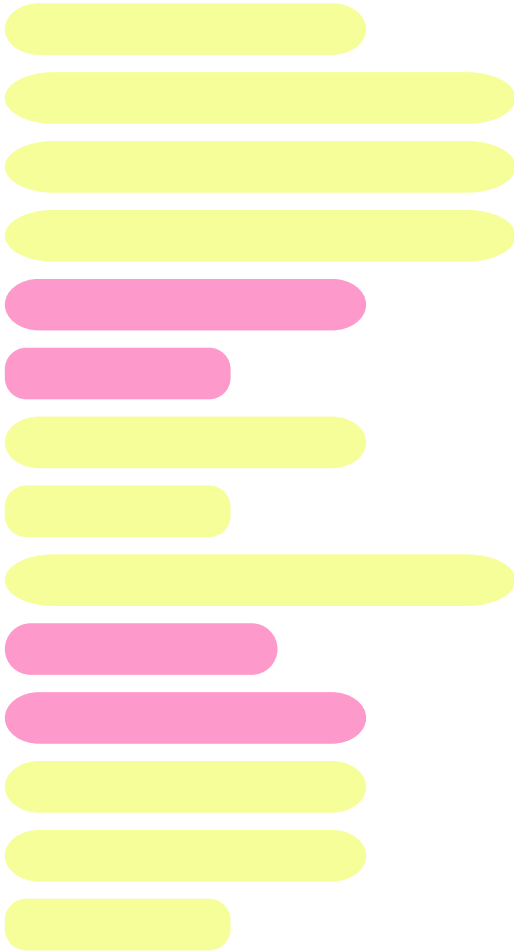
What to expect from this talk

$A \cup B$



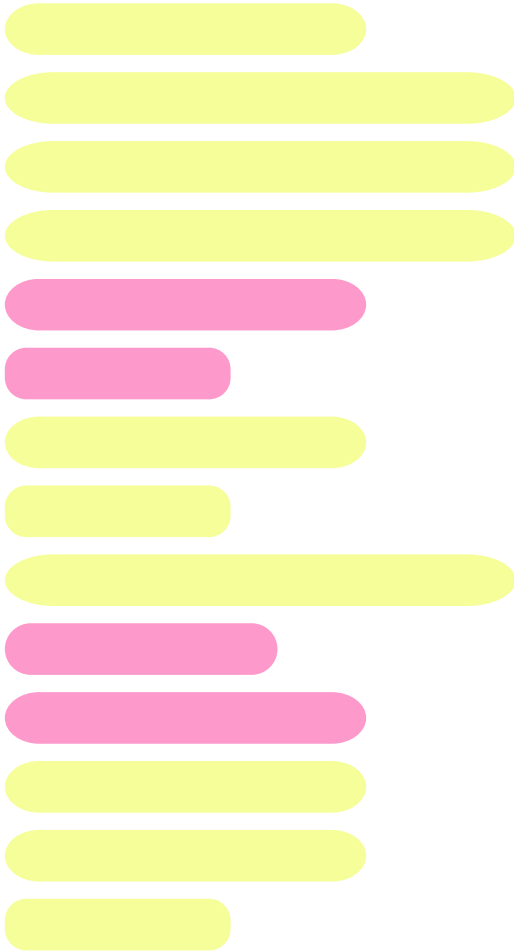
What to expect from this talk

Python ∪ Prolog



What to expect from this talk

Python ∪ PHP



Two levels of challenge

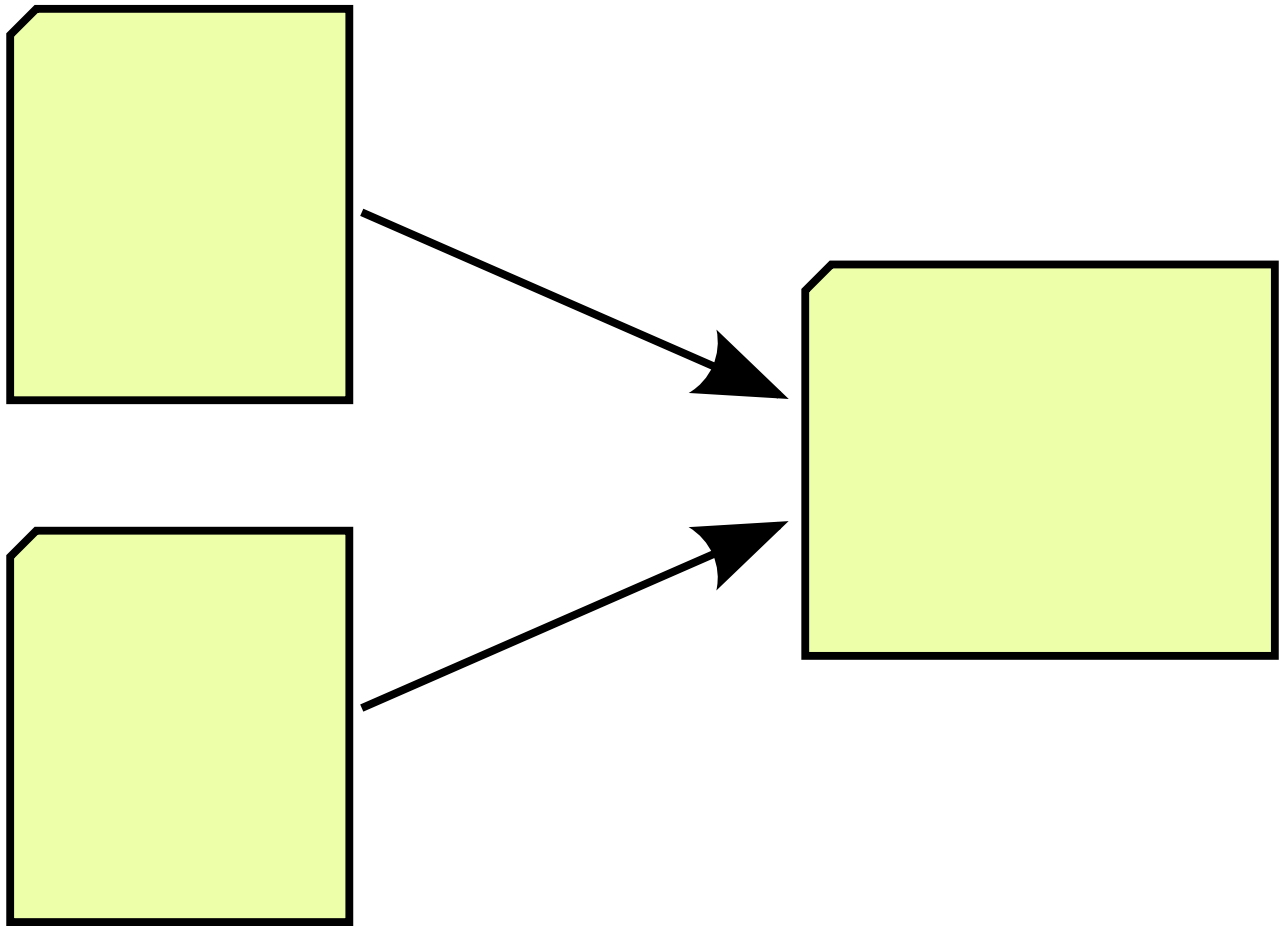
Tooling

Two levels of challenge

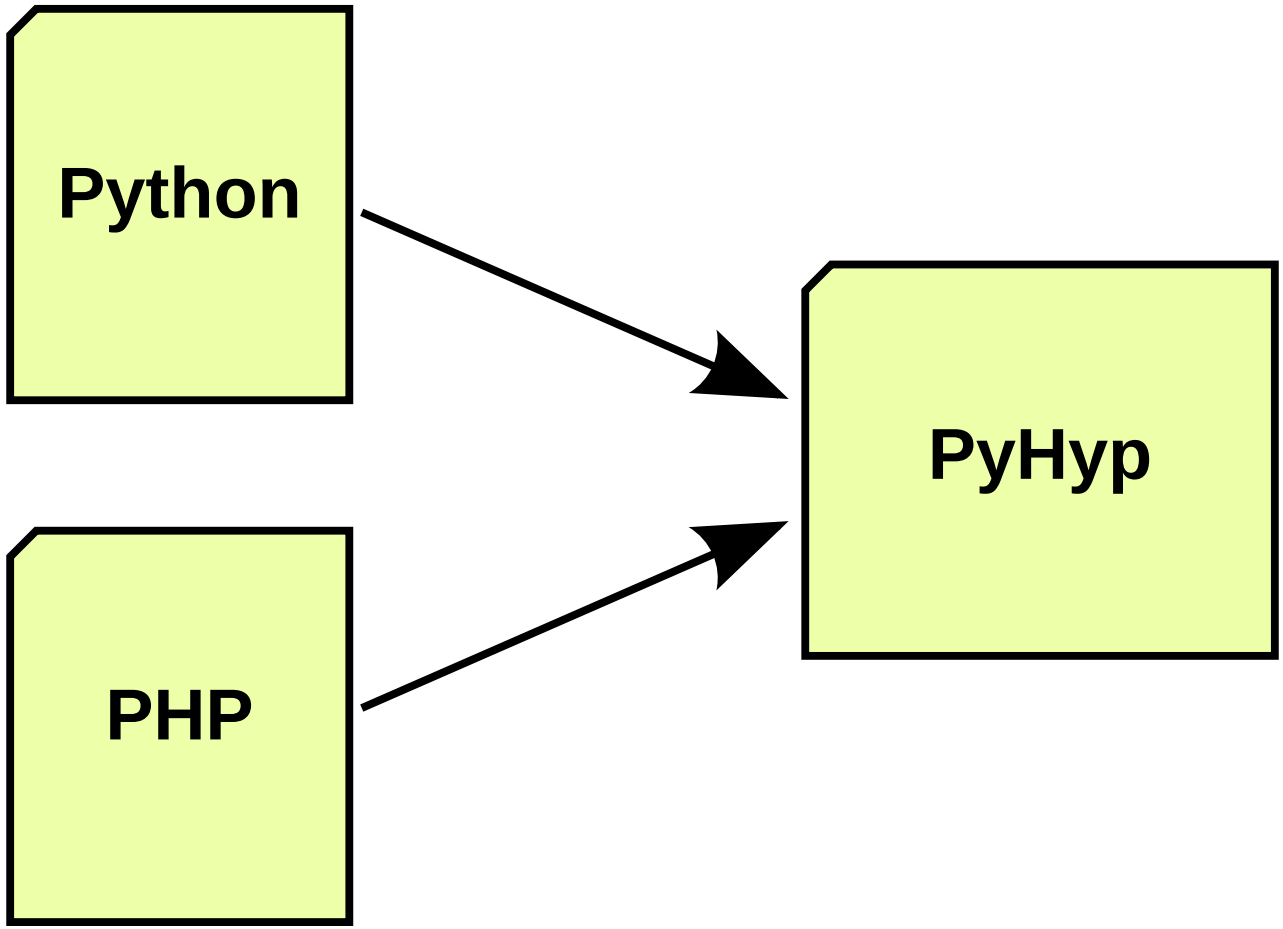
Tooling

Language friction

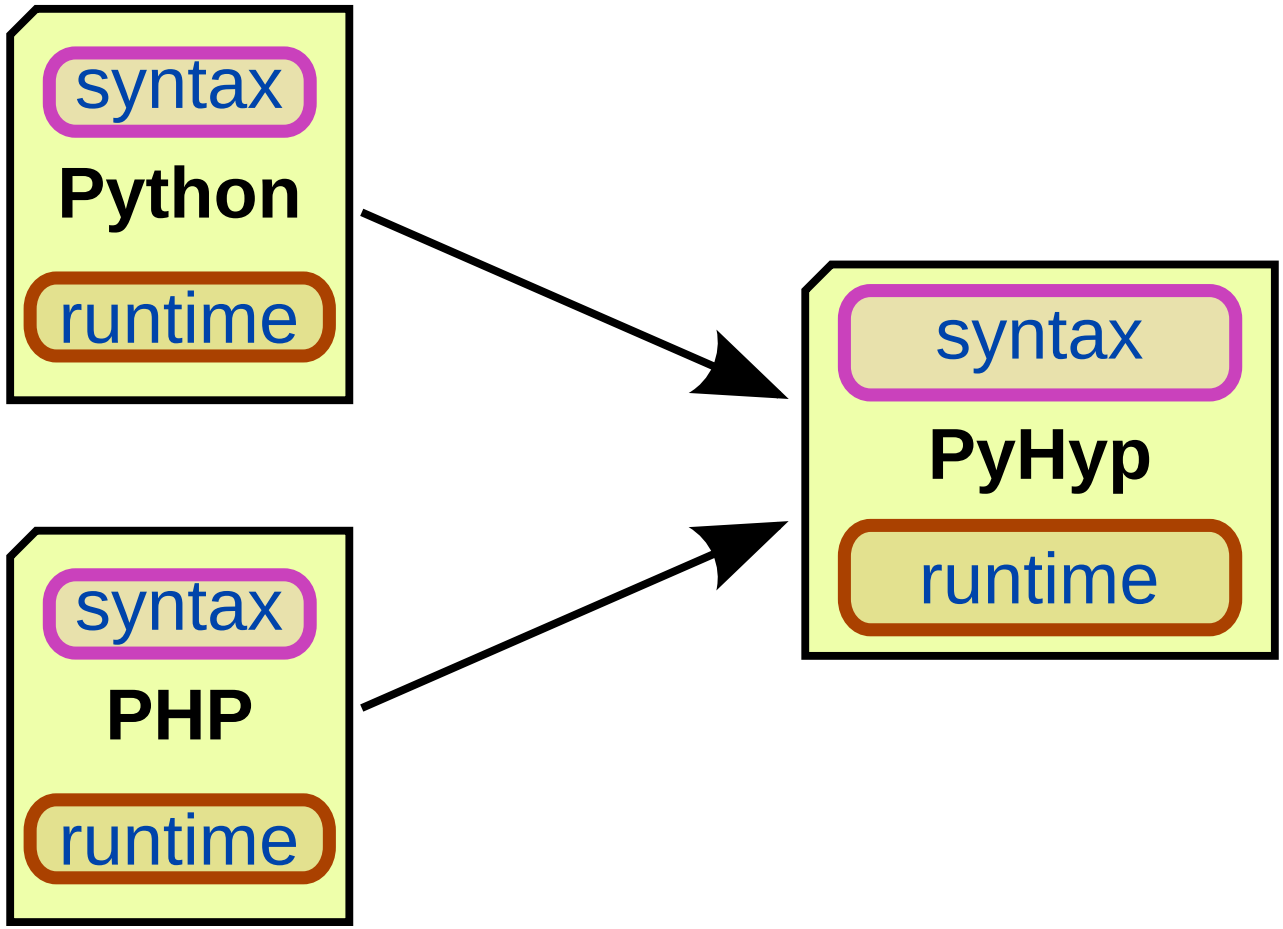
Tooling challenges



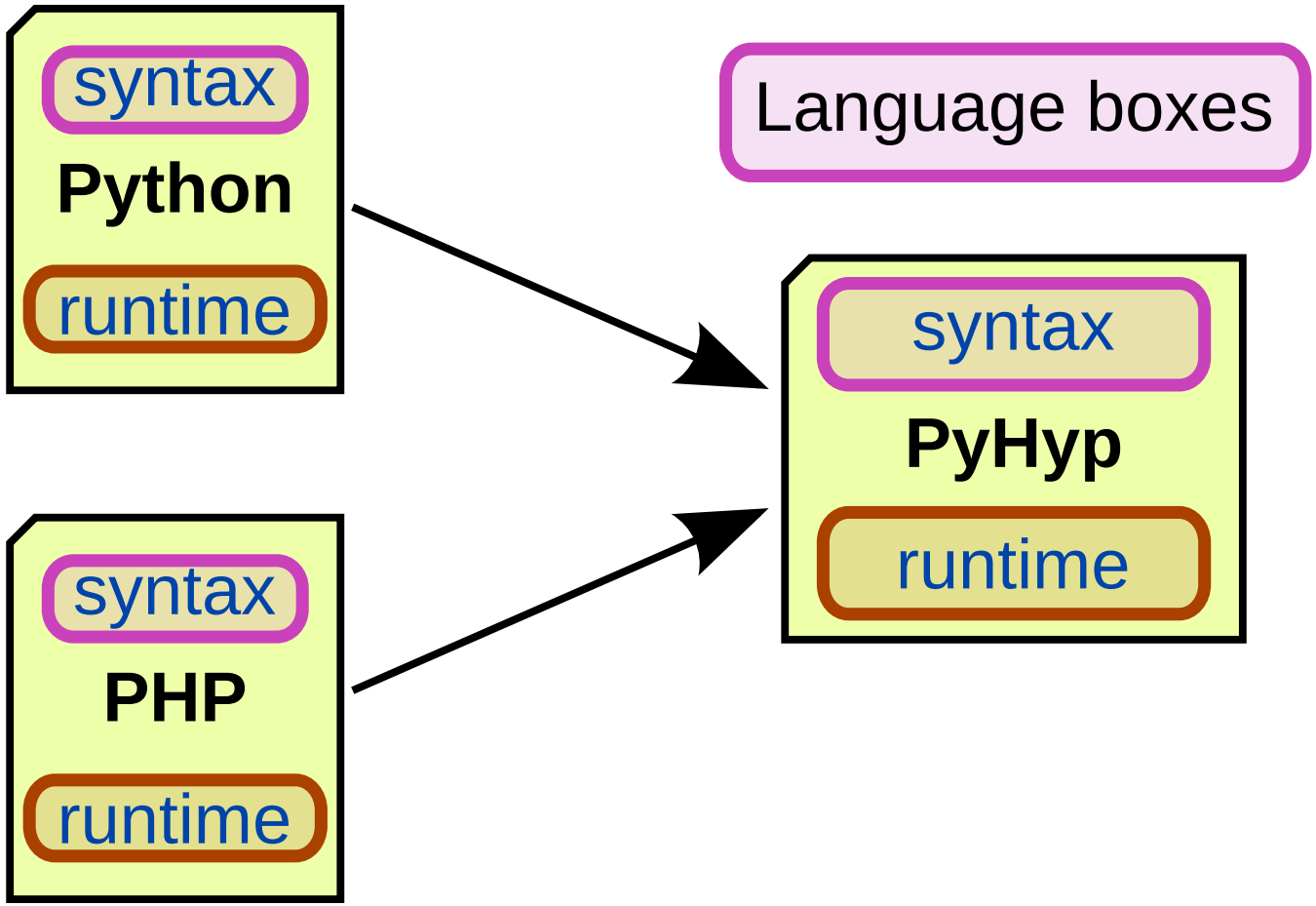
Tooling challenges



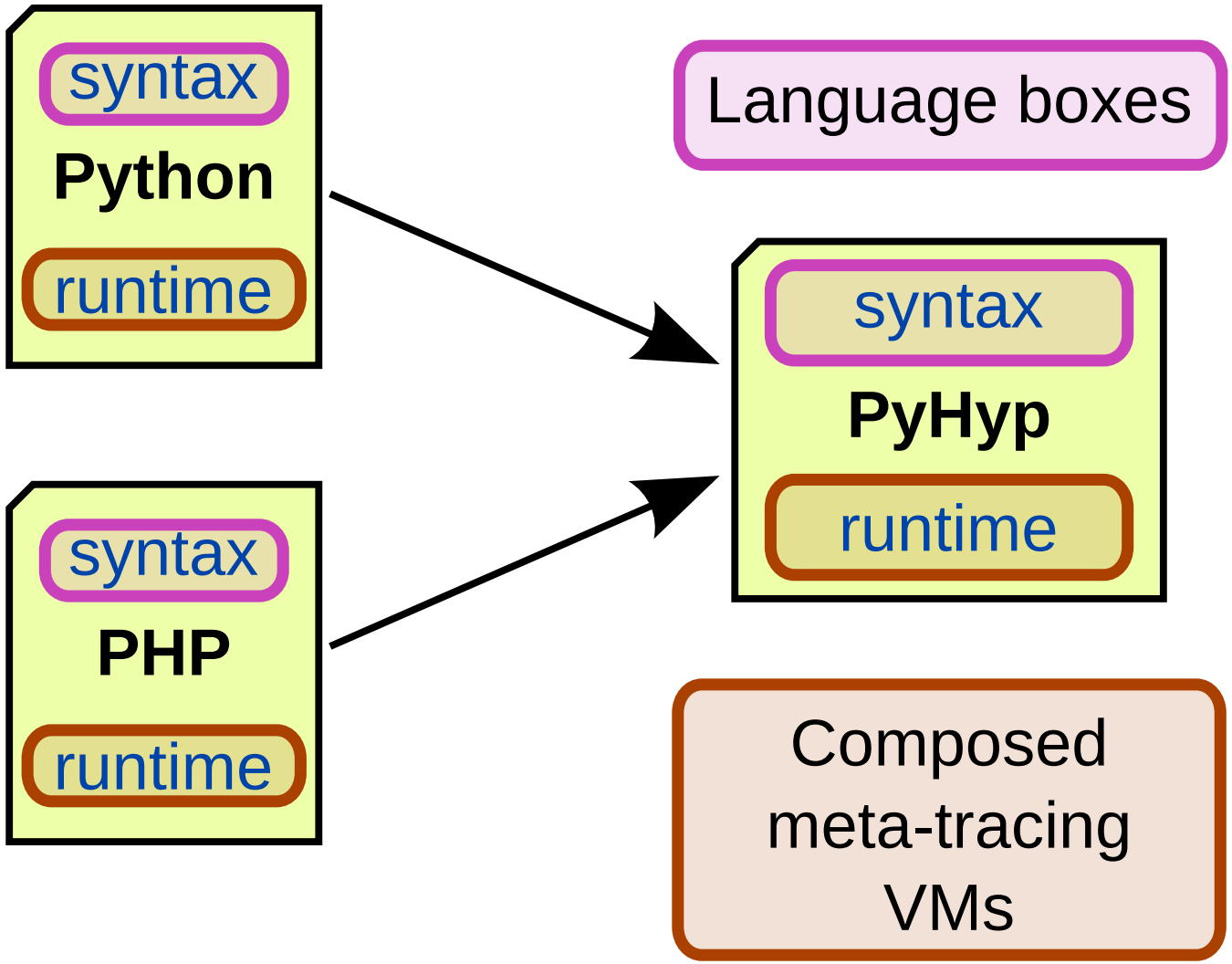
Tooling challenges



Tooling challenges



Tooling challenges



Syntax composition

PL X

<grammar>

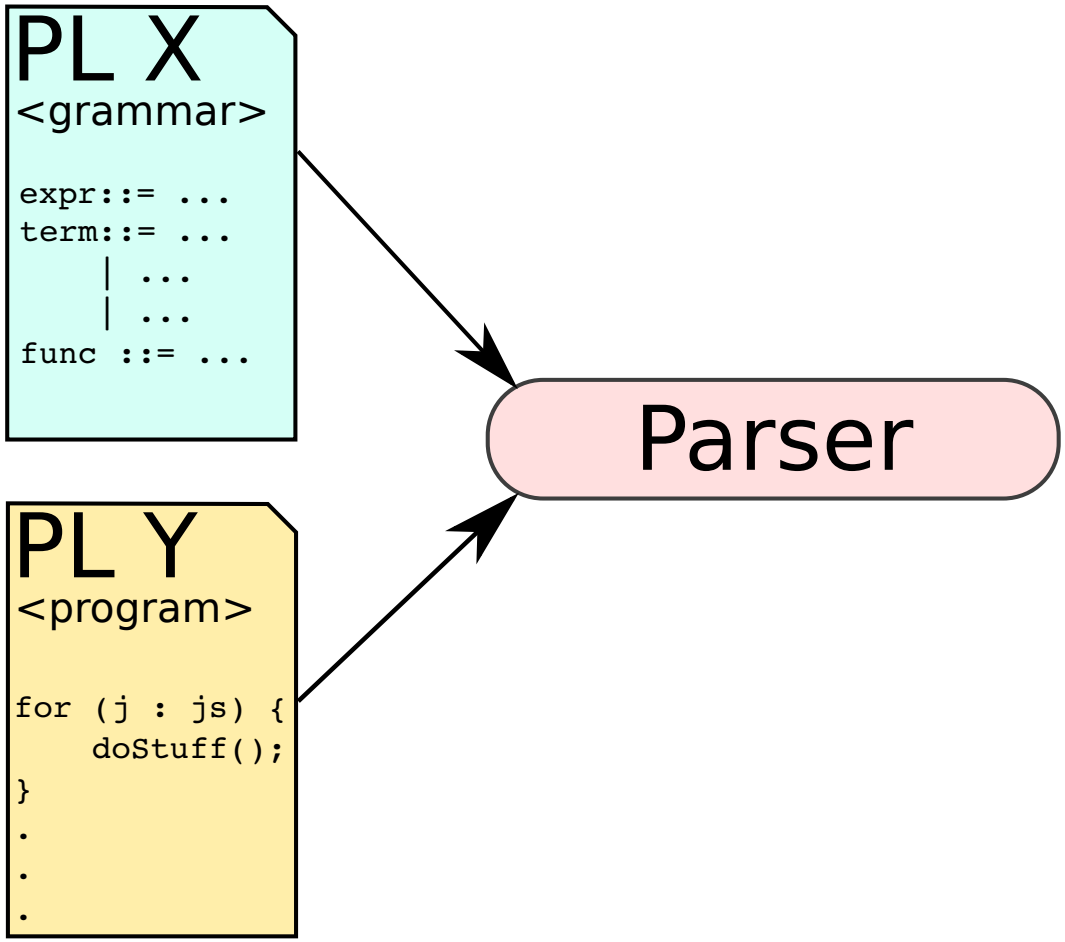
```
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...
```

PL Y

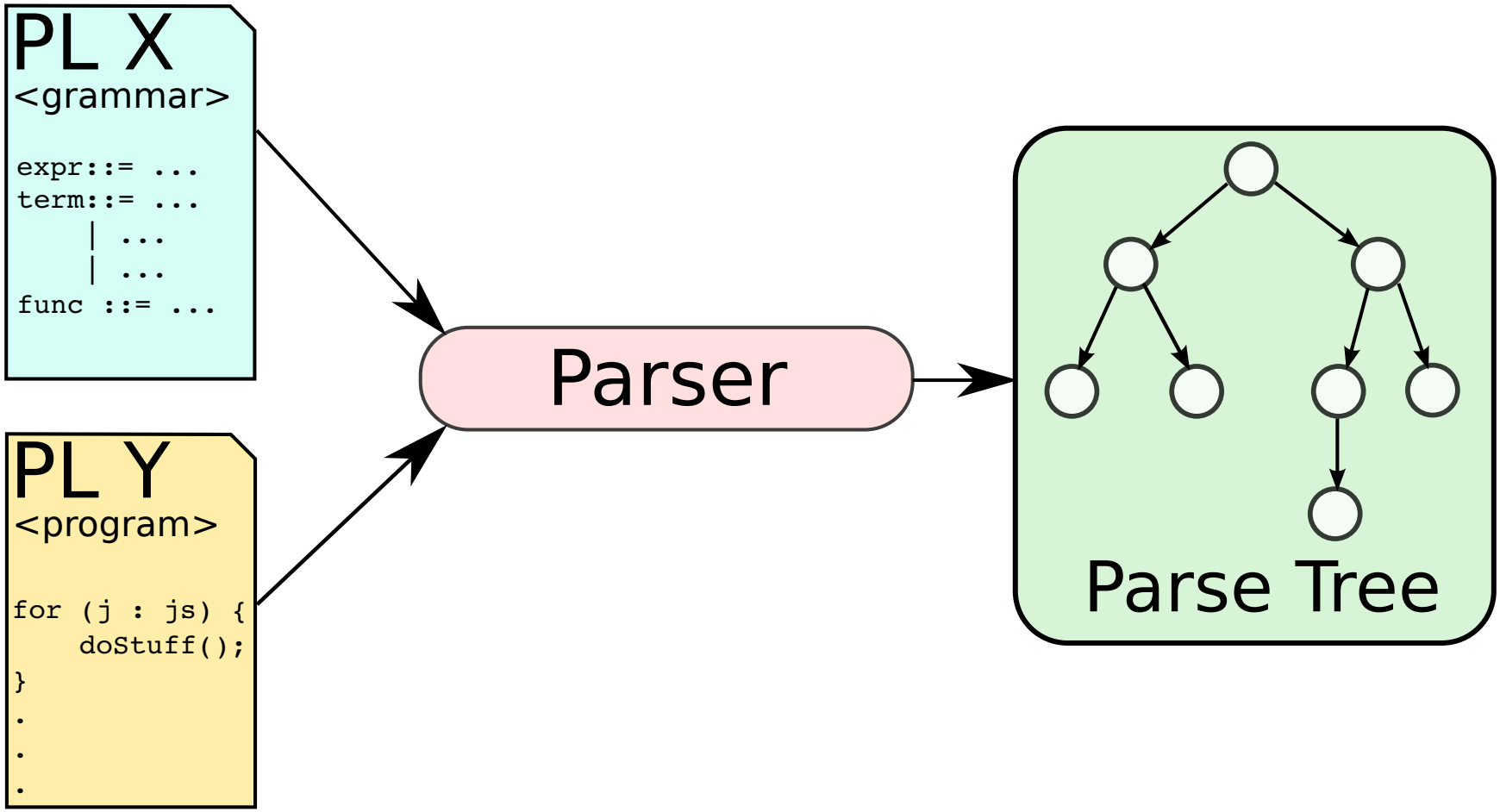
<program>

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

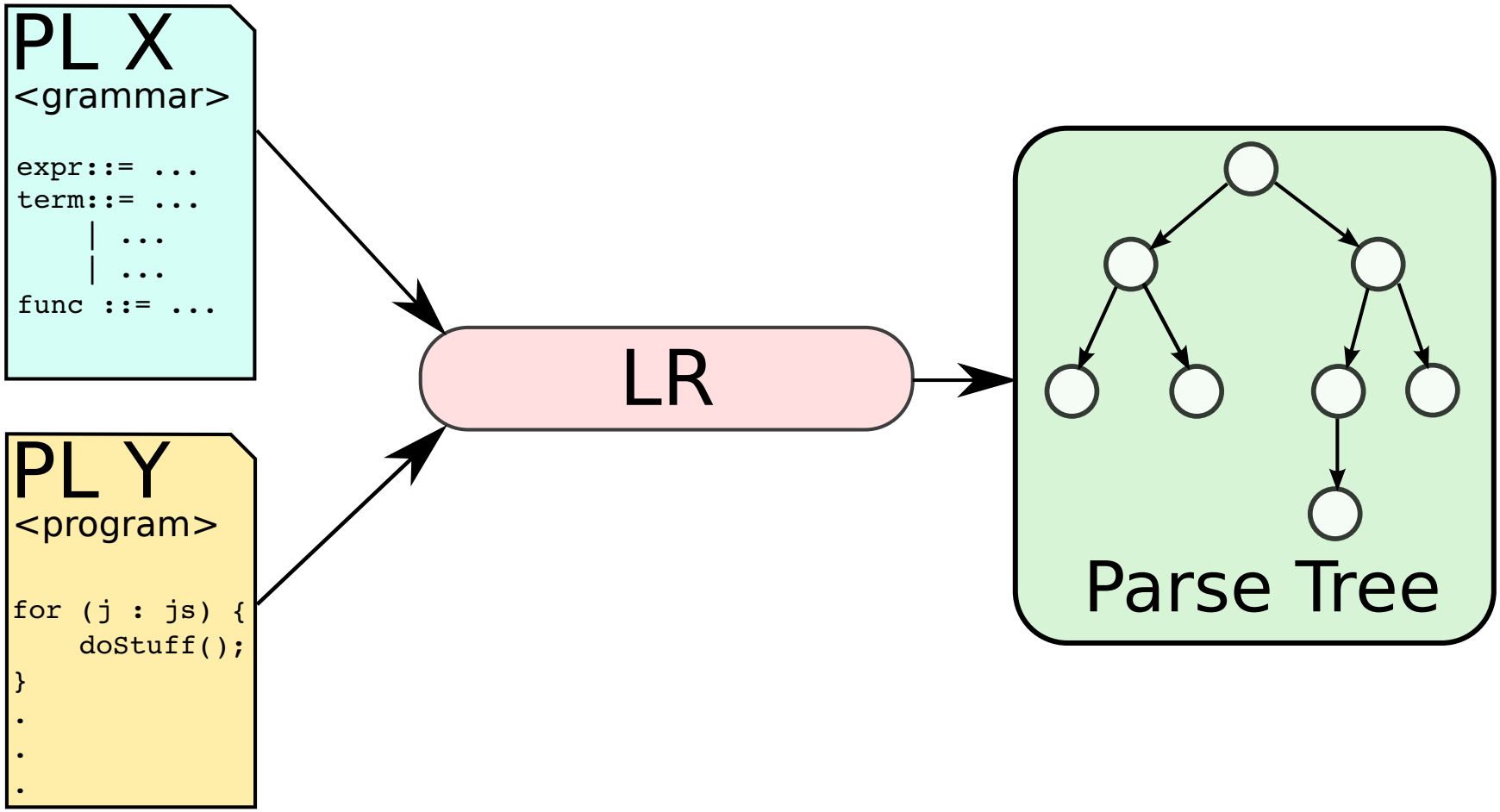
Syntax composition



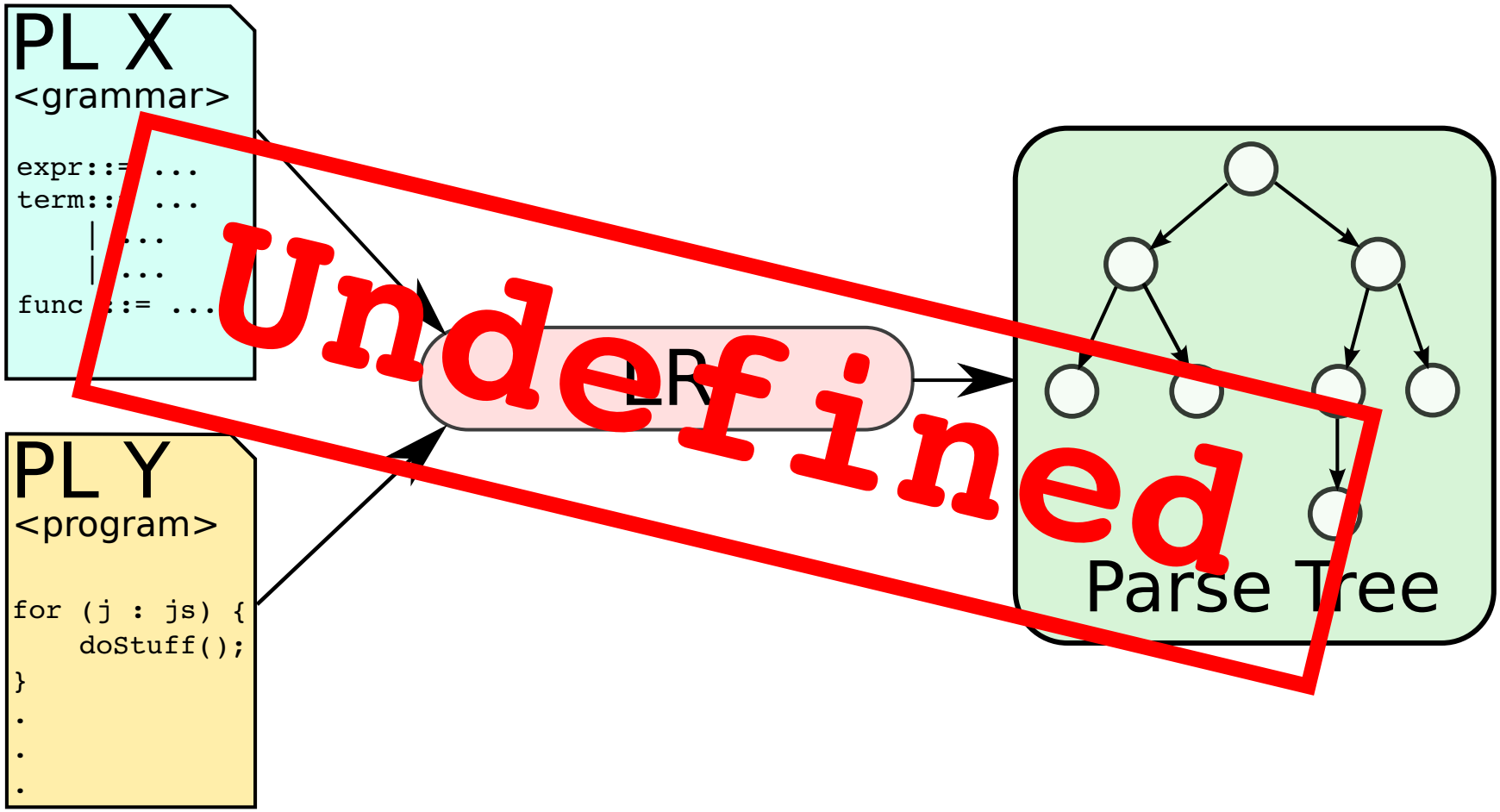
Syntax composition



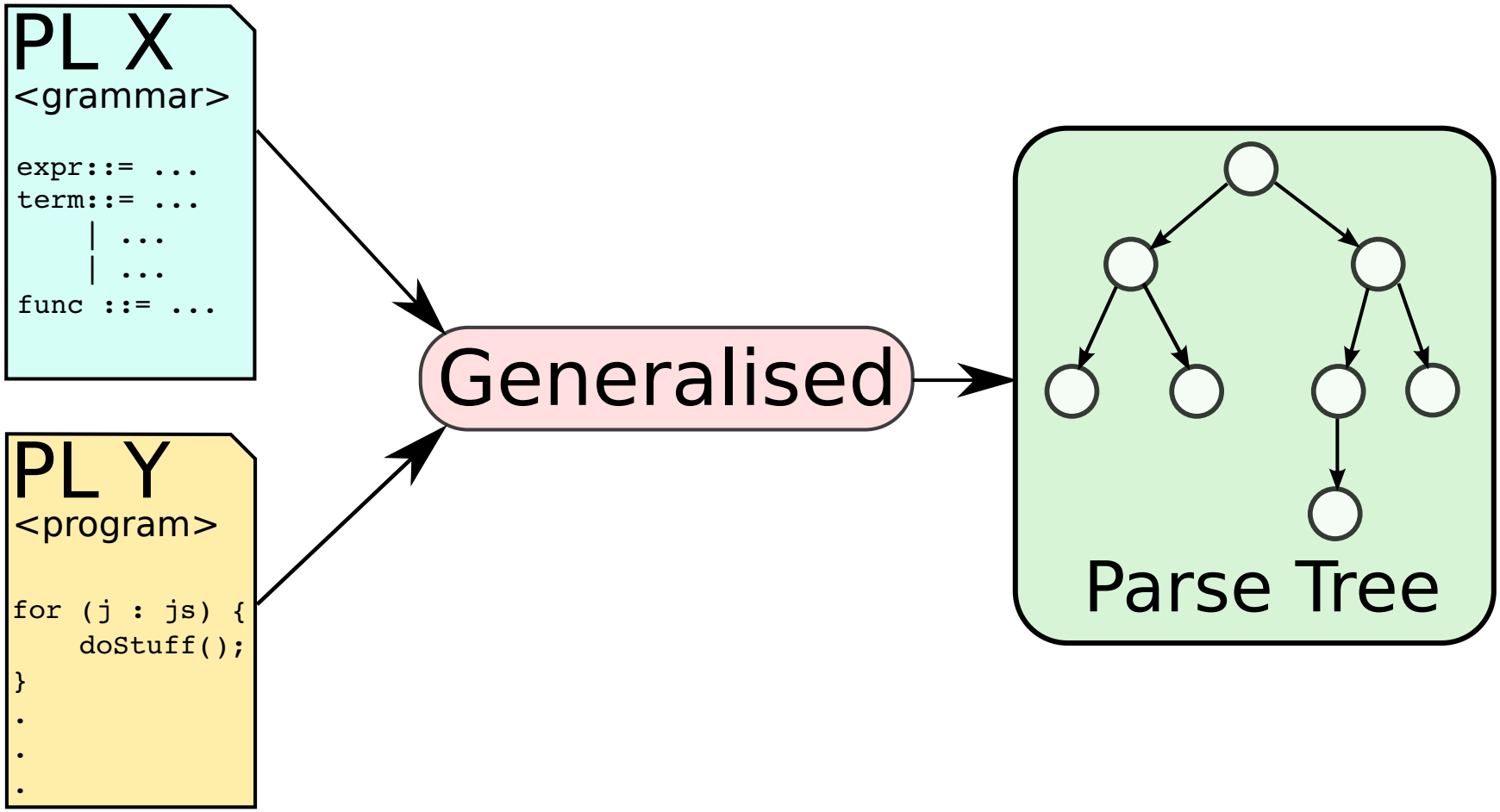
Syntax composition



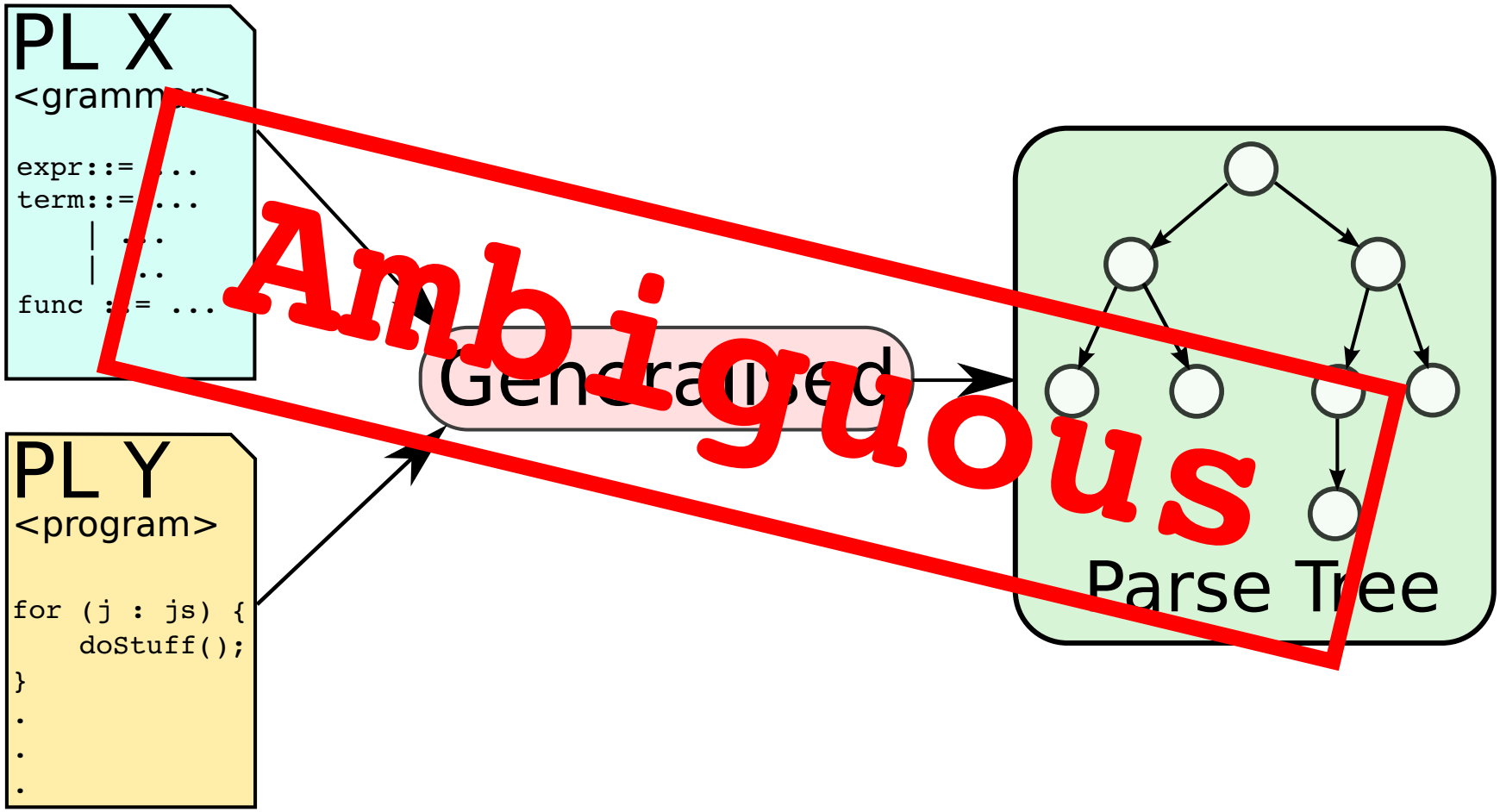
Syntax composition



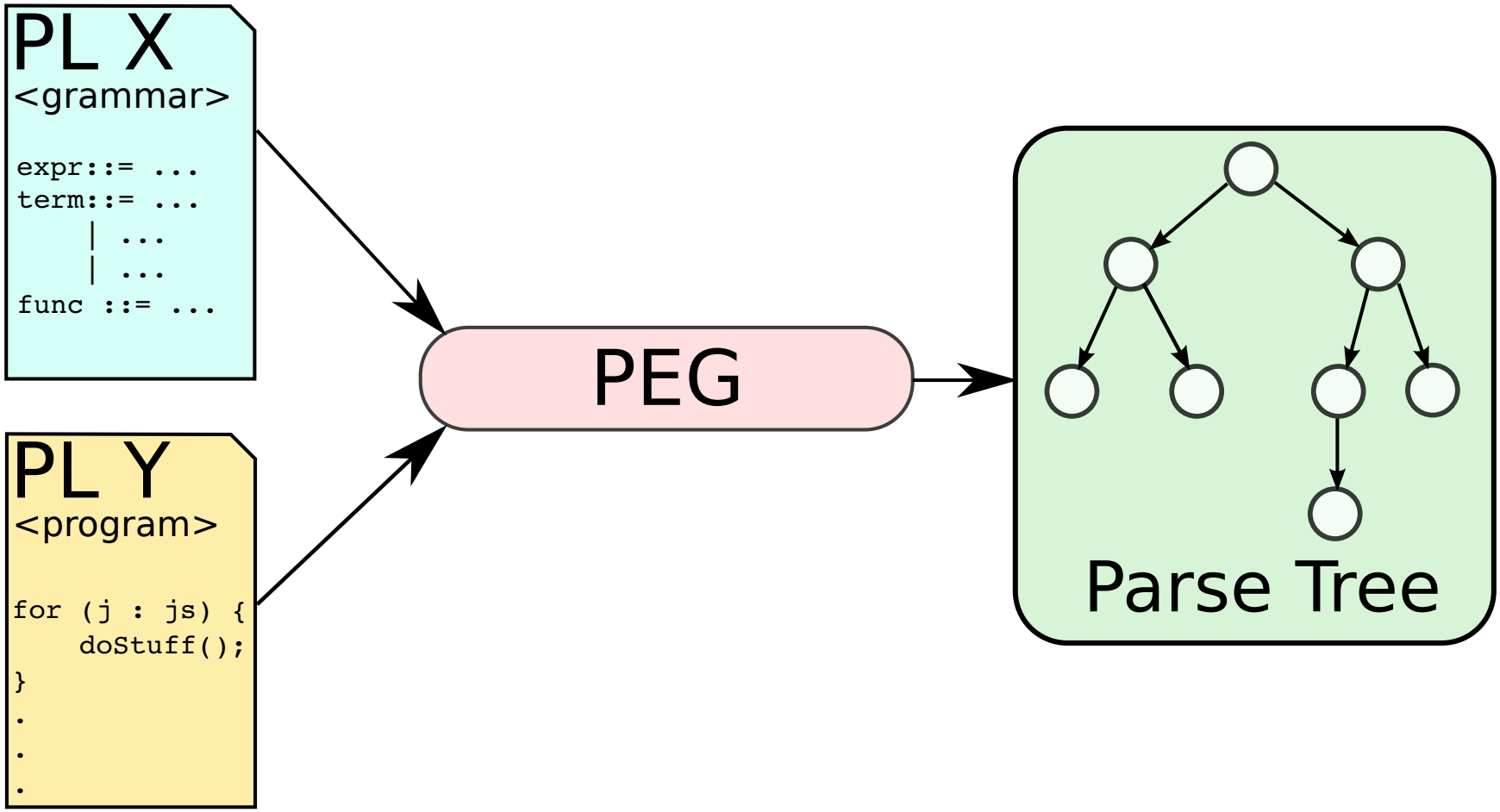
Syntax composition



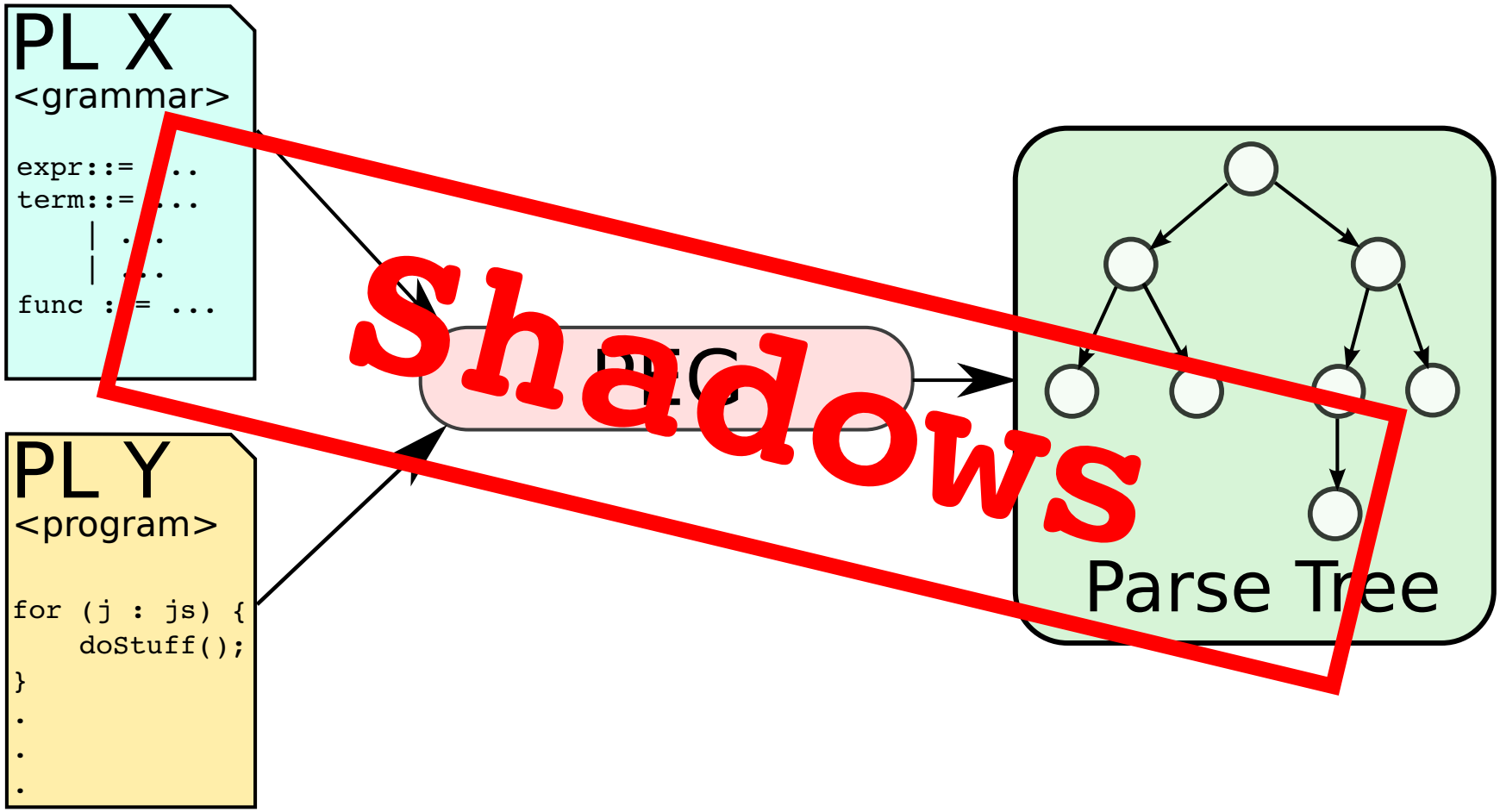
Syntax composition



Syntax composition



Syntax composition



The only choice?

The only choice?

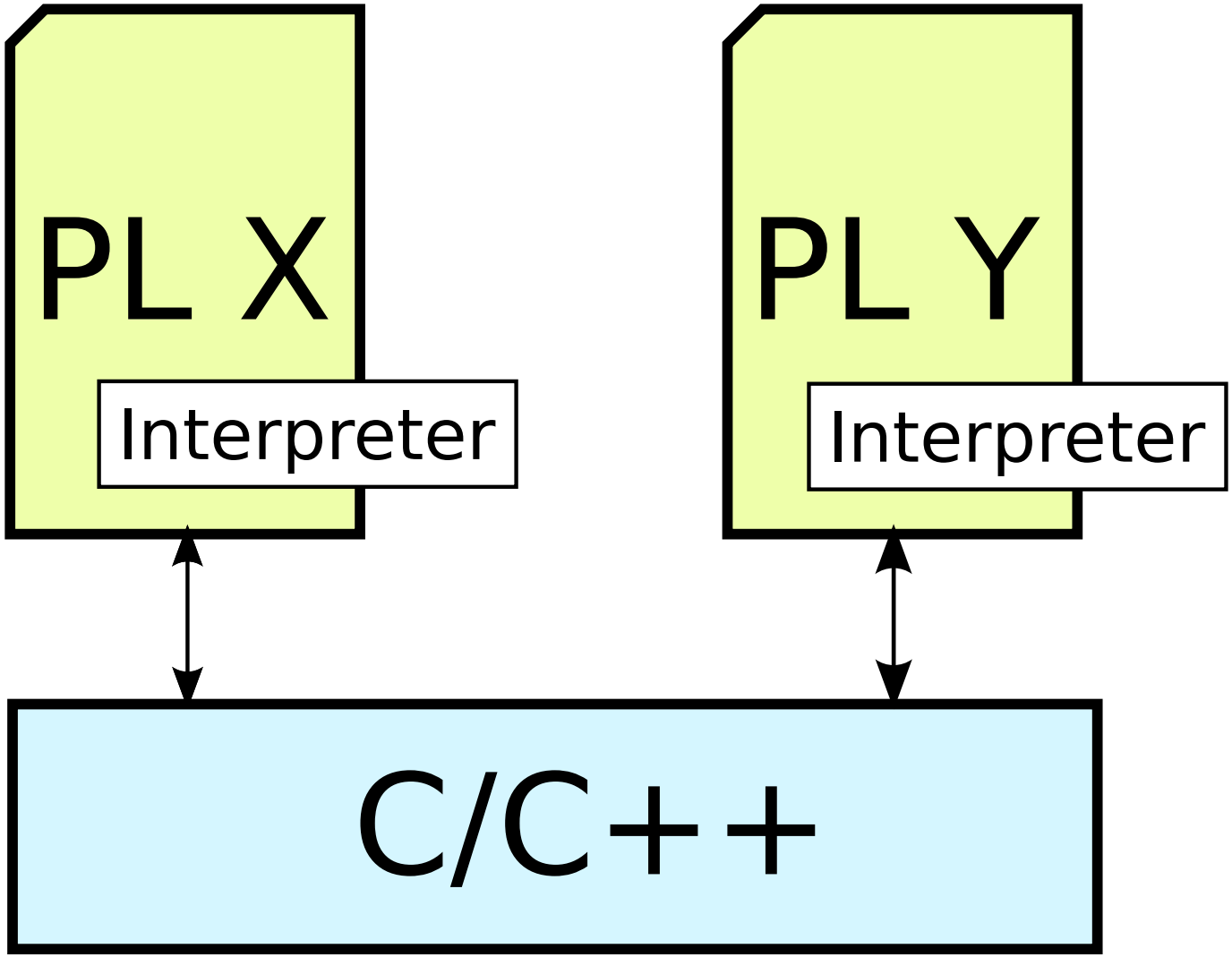
SDE

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

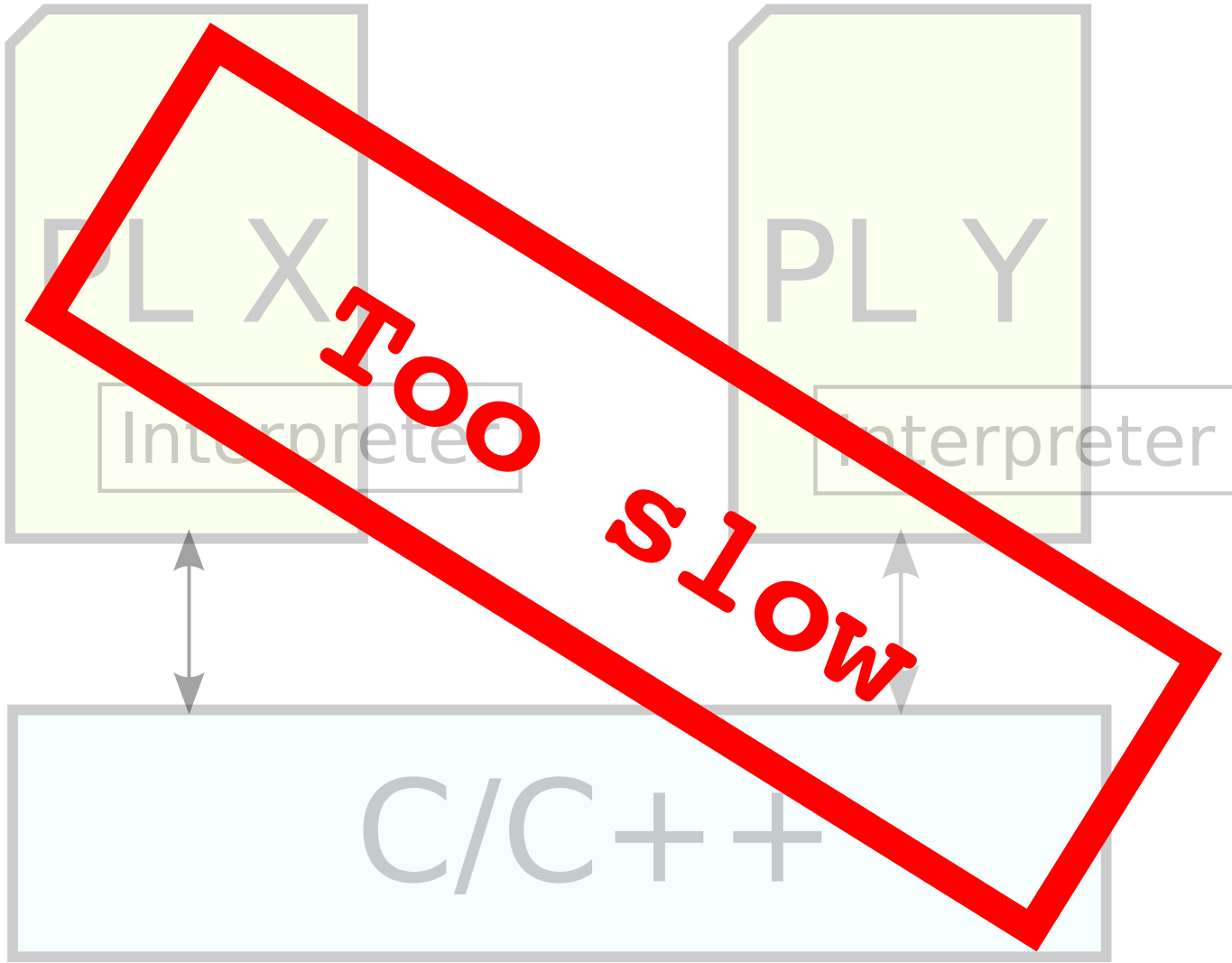
Eco demo

Runtime composition

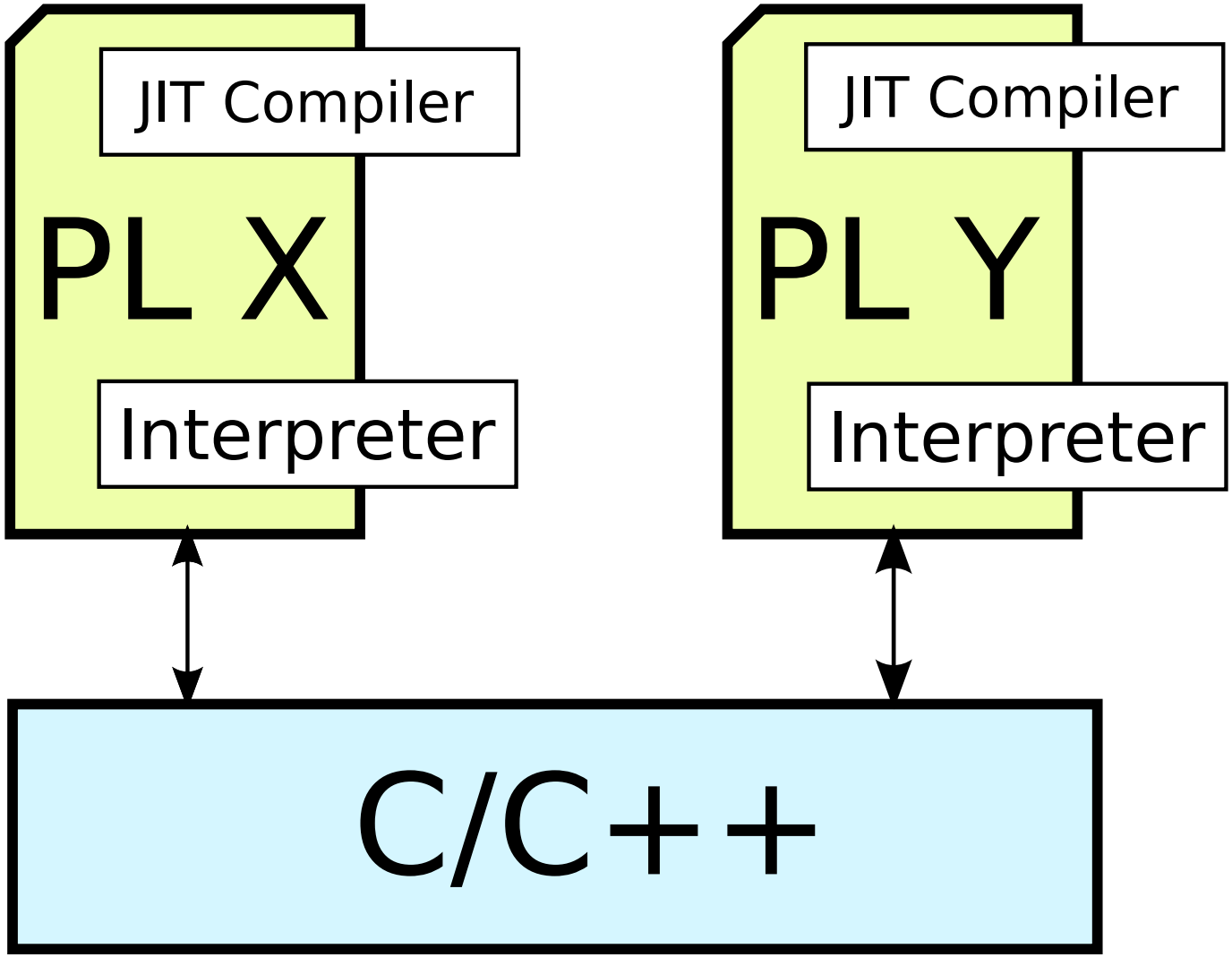
Runtime composition



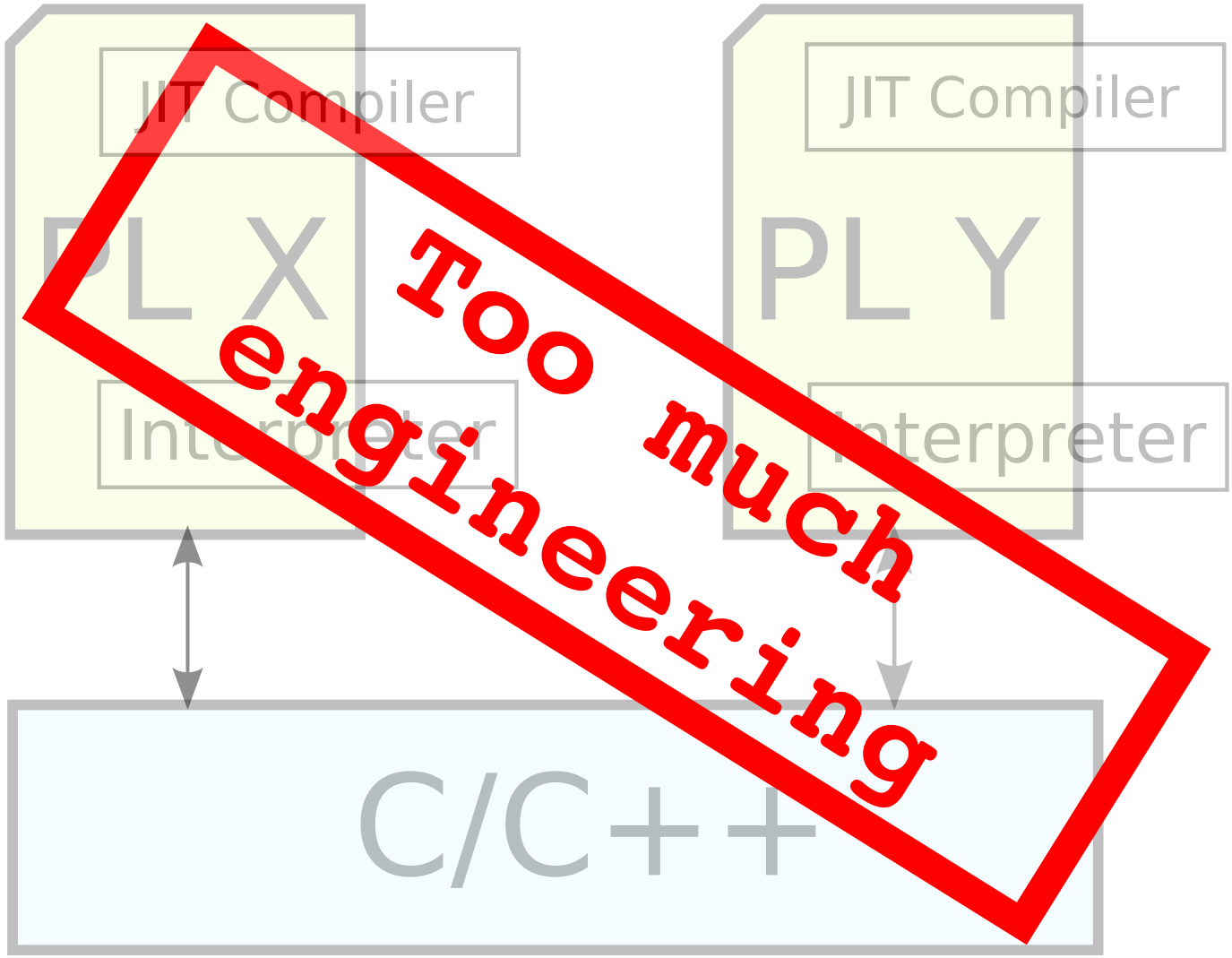
Runtime composition



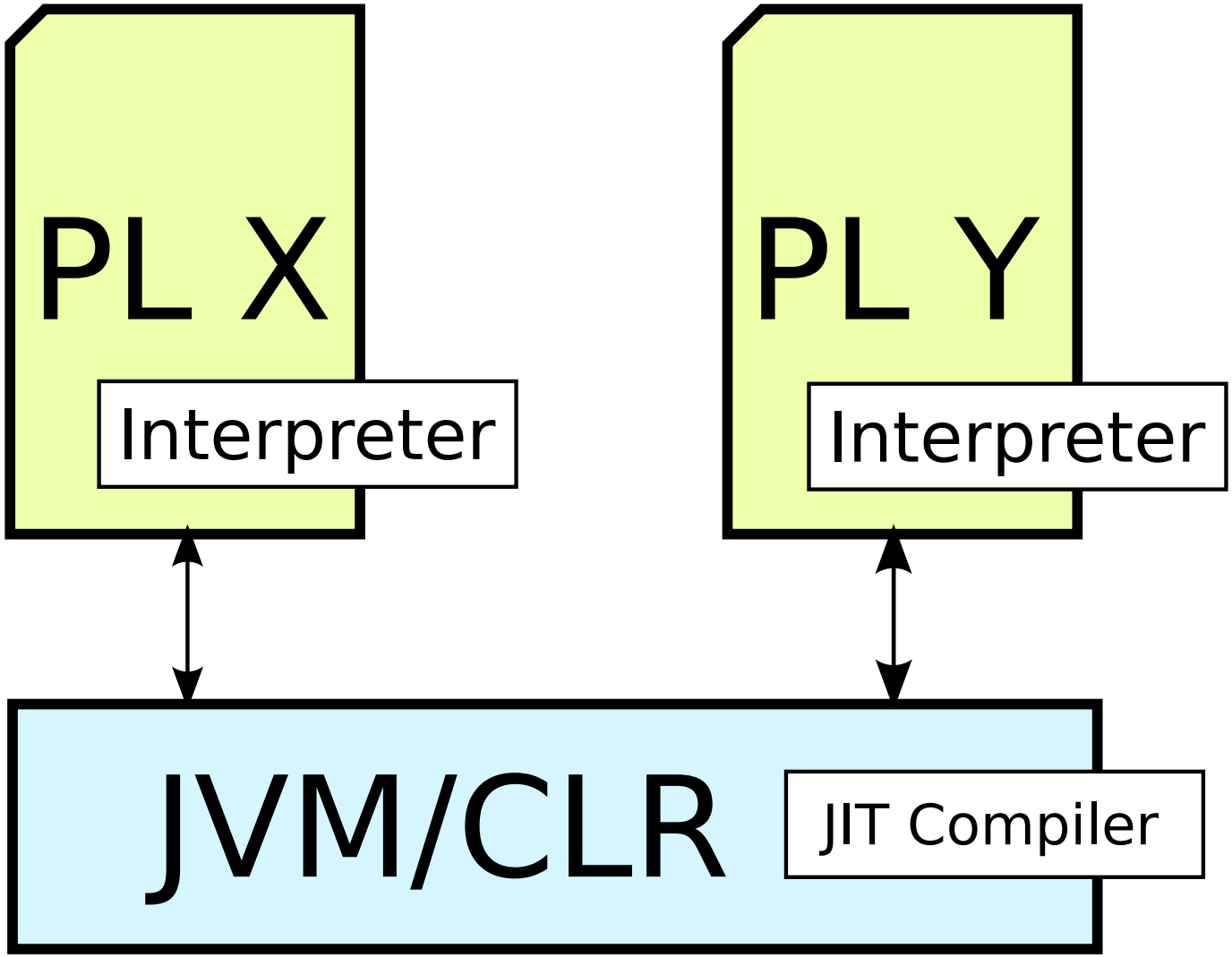
Runtime composition



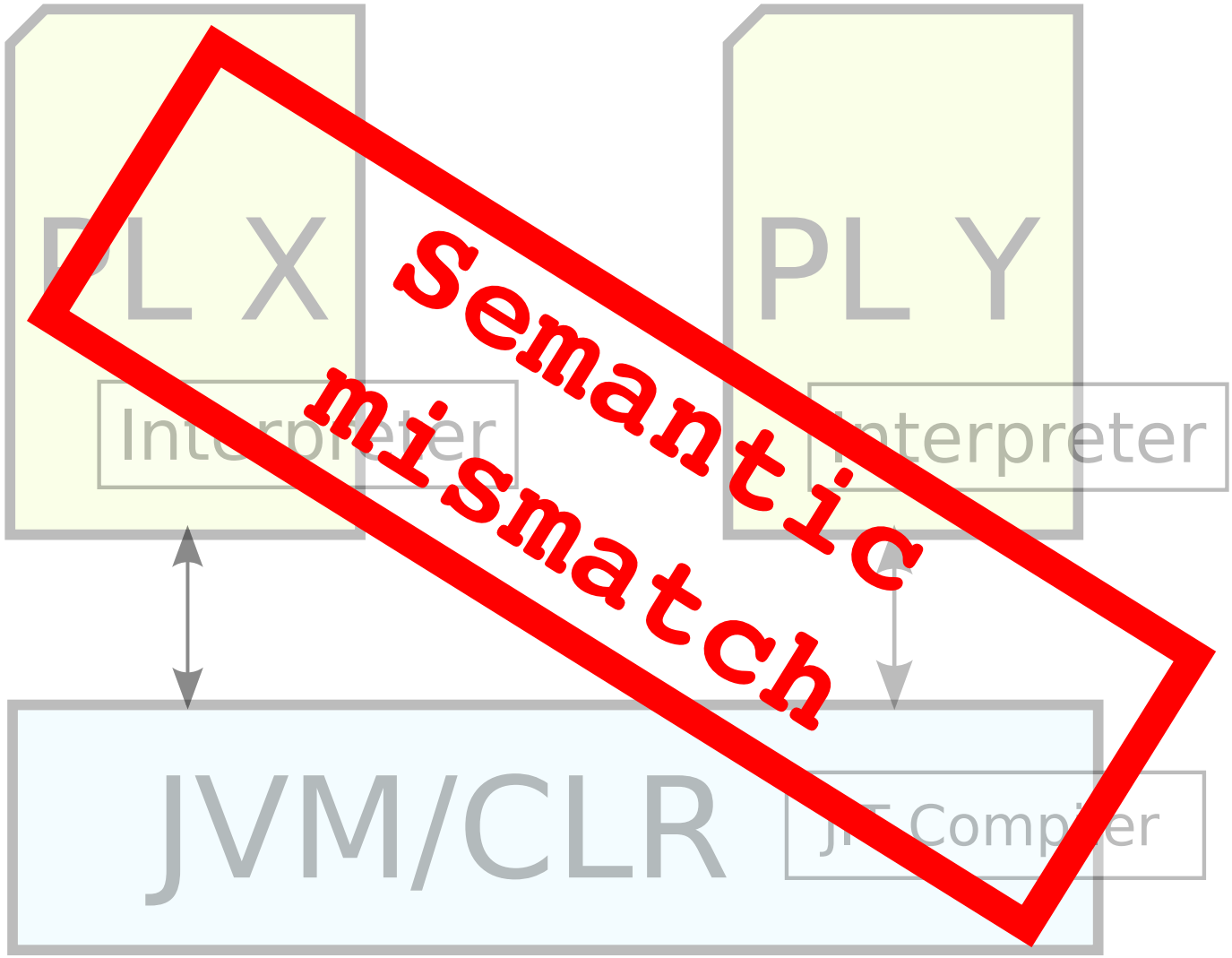
Runtime composition



Runtime composition

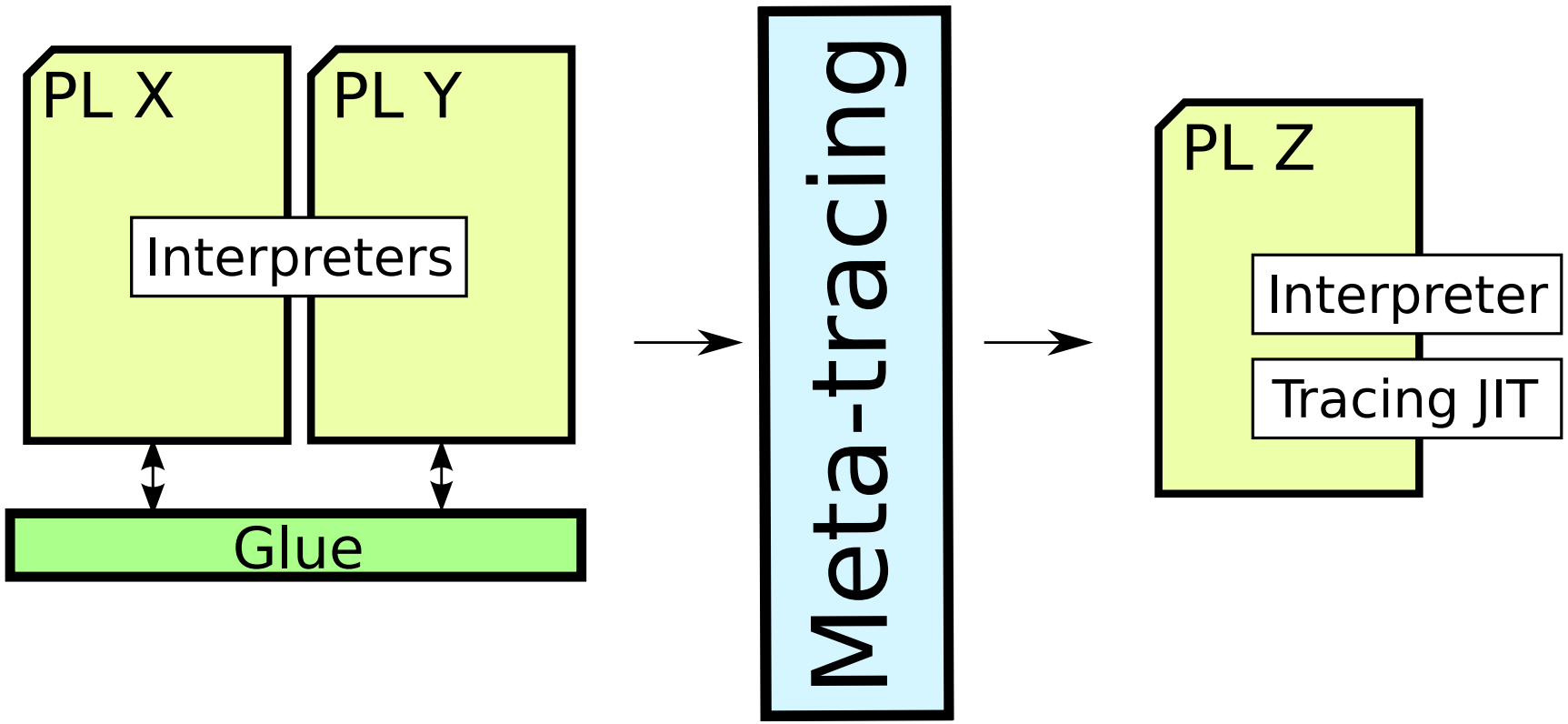


Runtime composition

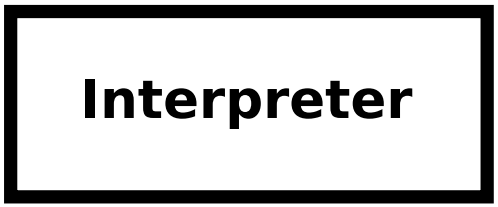


Runtime composition

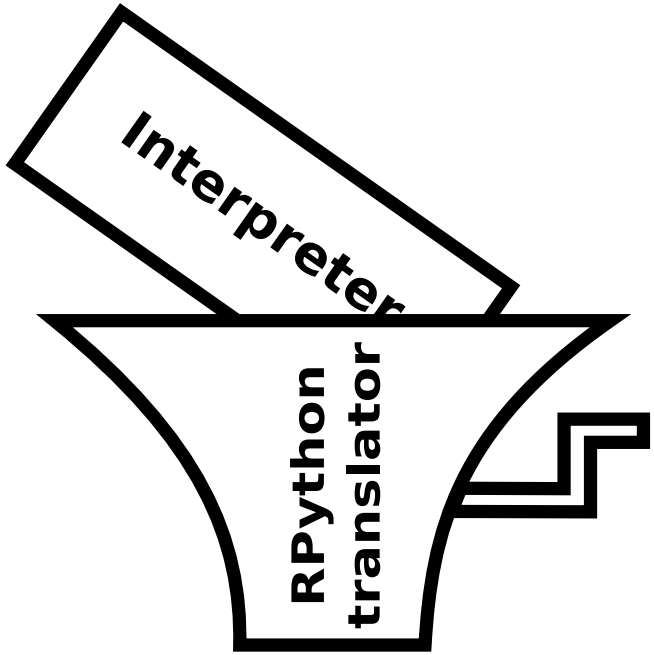
Runtime composition



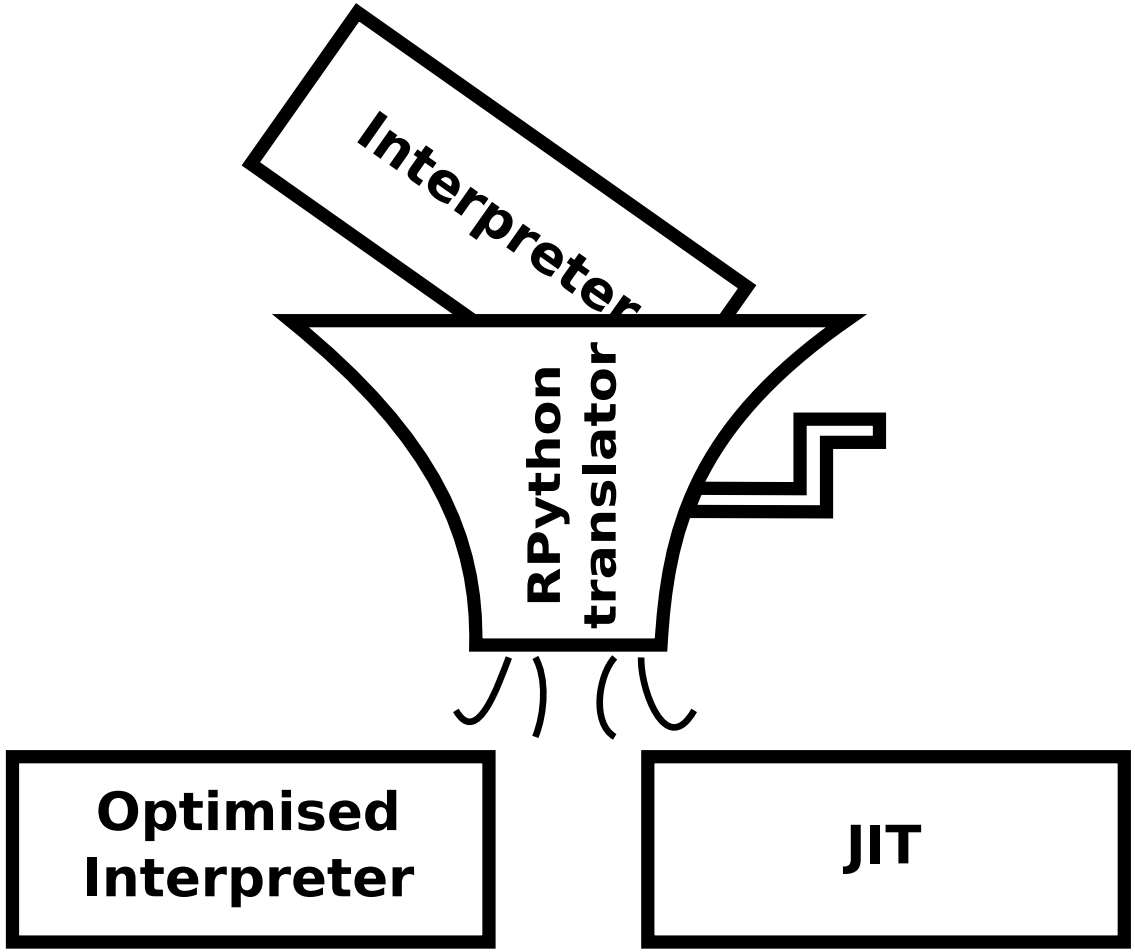
Meta-tracing translation with RPython



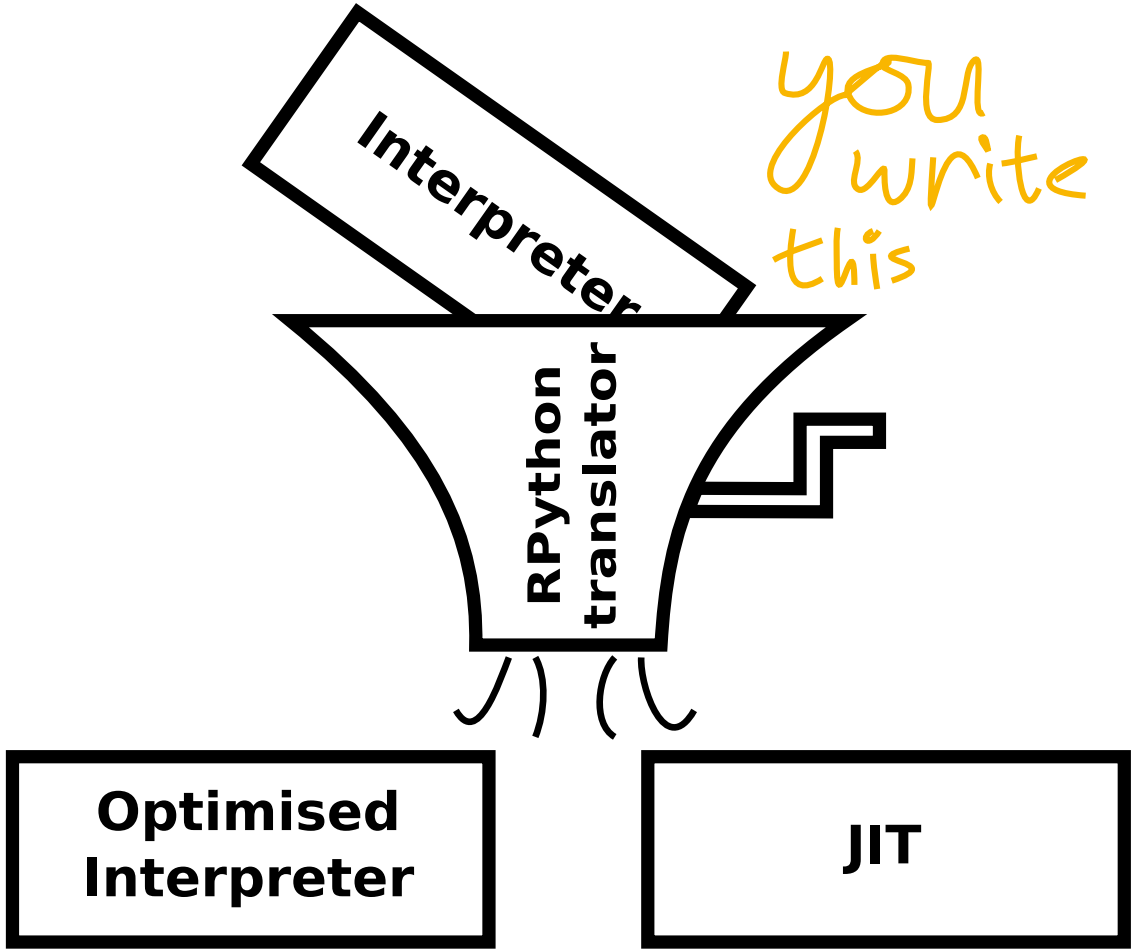
Meta-tracing translation with RPython



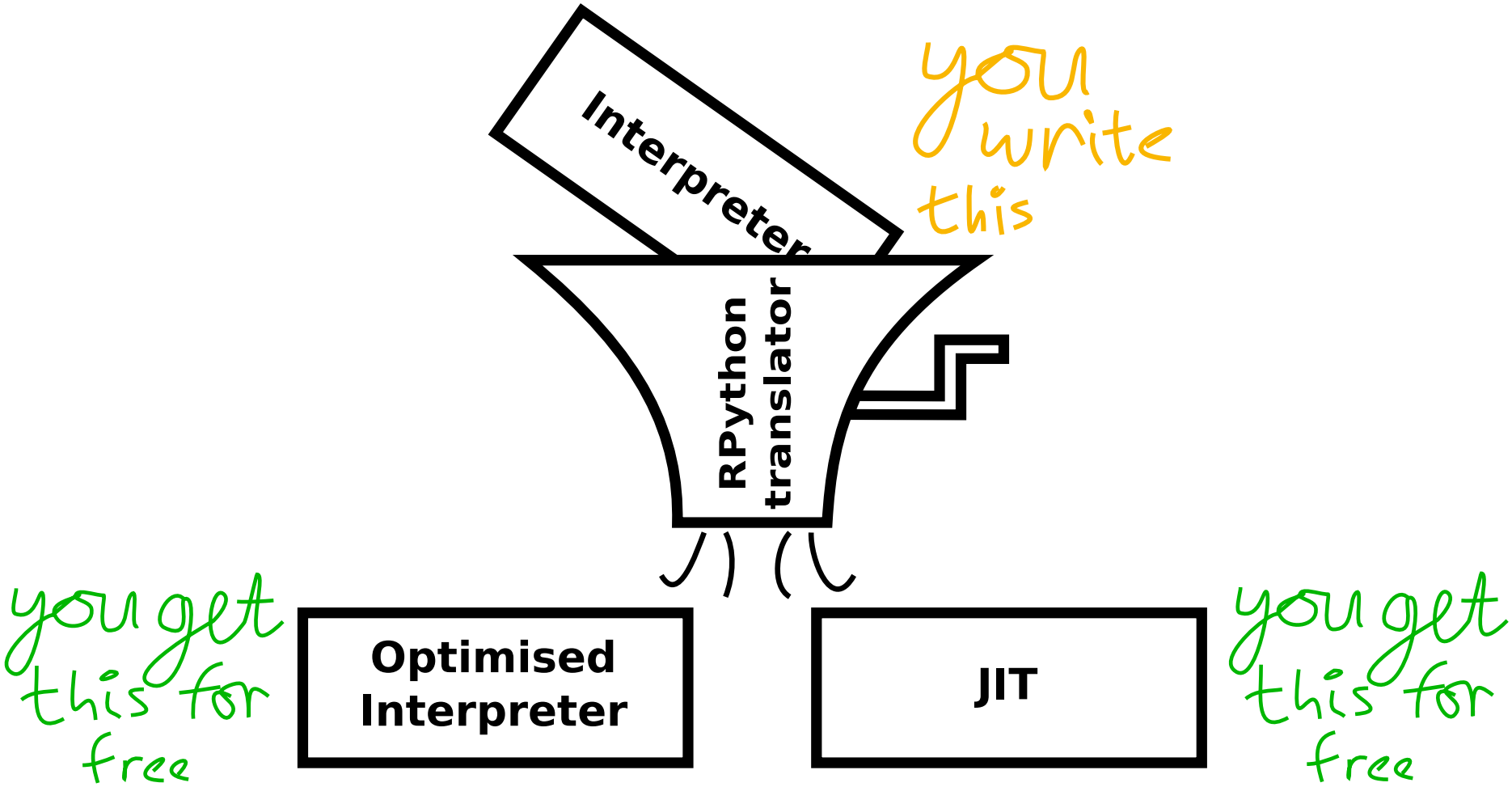
Meta-tracing translation with RPython



Meta-tracing translation with RPython



Meta-tracing translation with RPython



Adding a JIT compiler 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 ...:
        ...
```

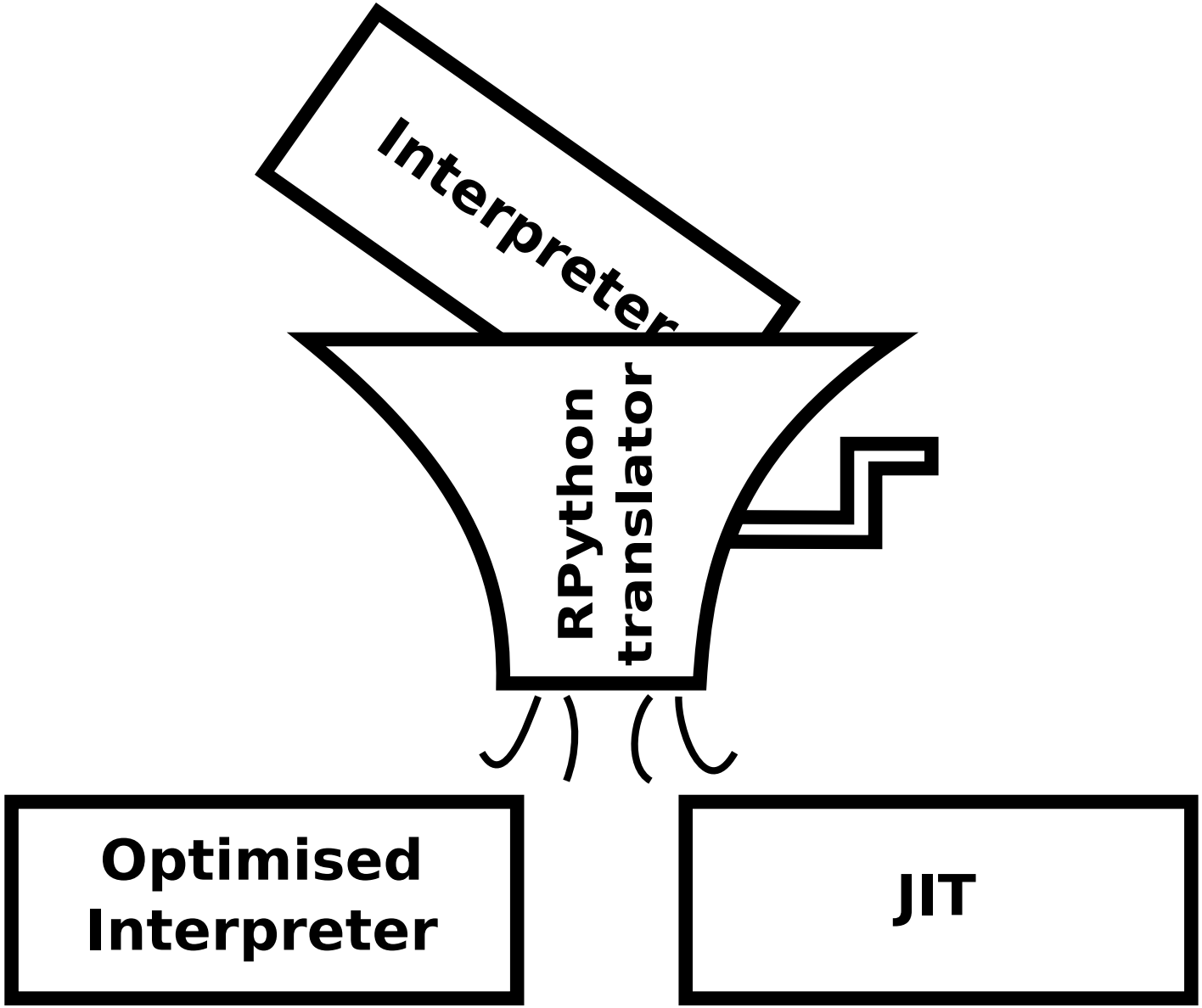
Observation: interpreters are big loops.

Adding a JIT compiler to an RPython interpreter

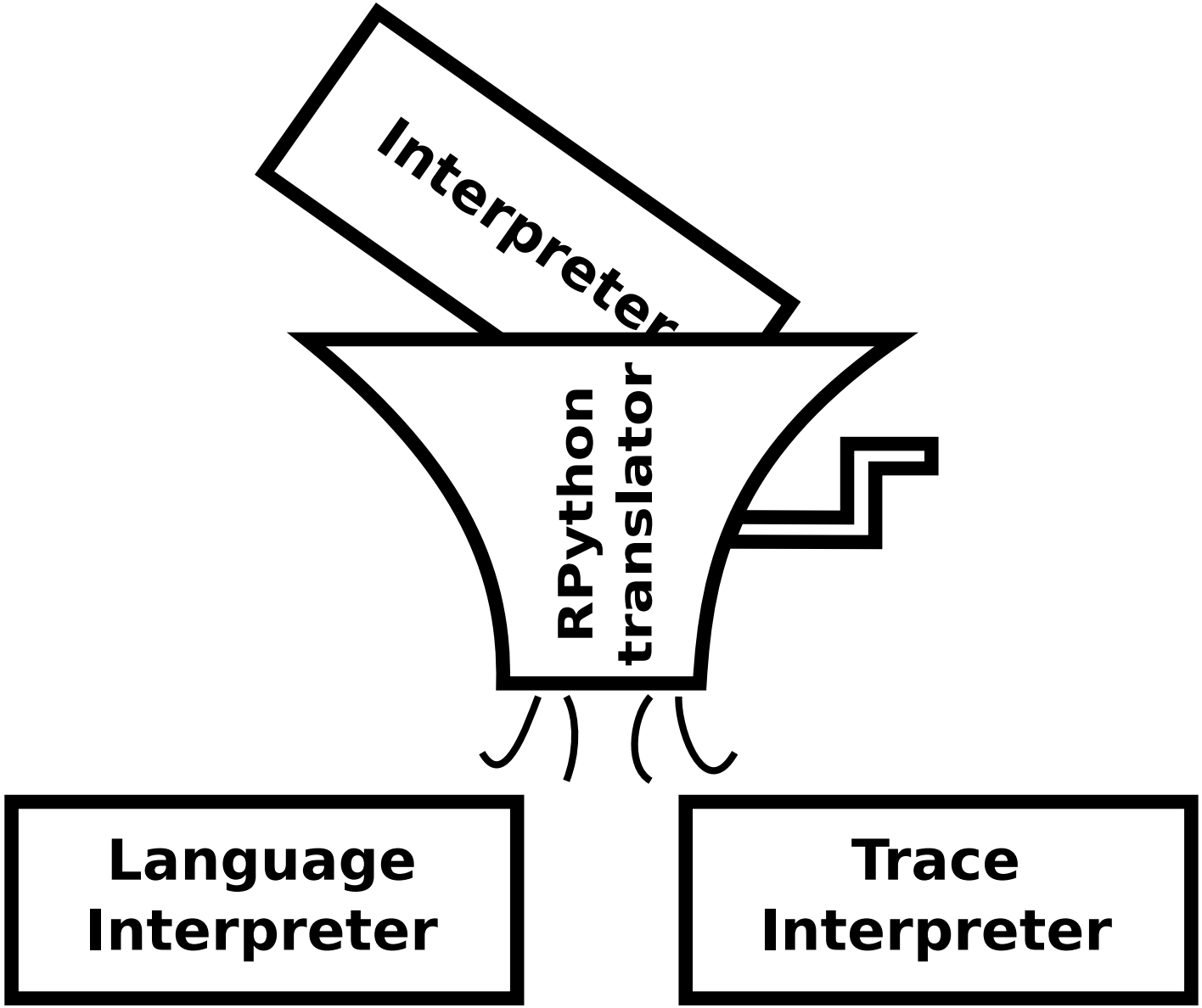
```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

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

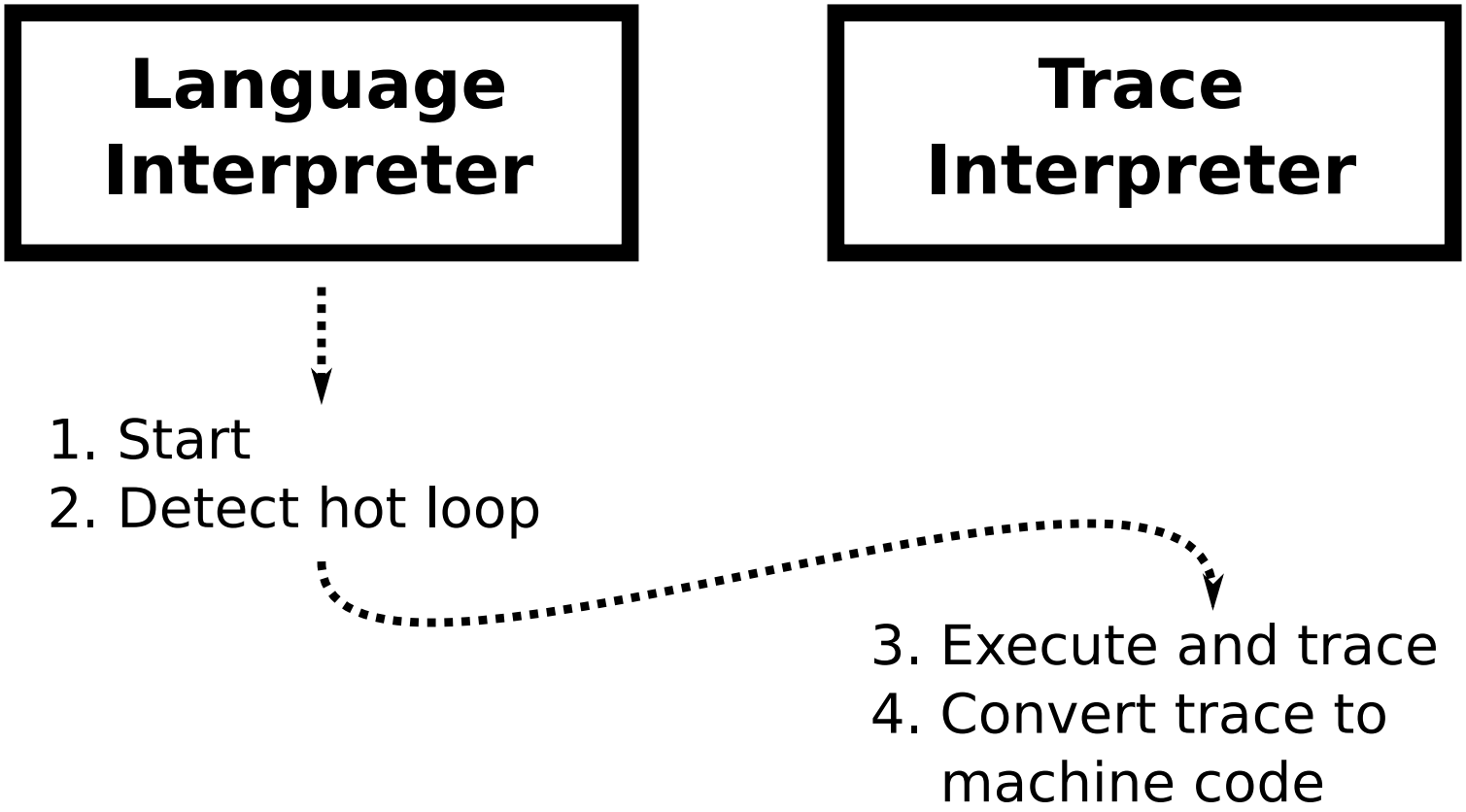
`if x < 0:`
`  x = x + 1`
`else:`
`  x = x + 2`
`x = x + 3`
	`guard_type(x, int)`
	`guard_not_less_than(x, 0)`
	`guard_type(x, int)`
	`x = int_add(x, 2)`
	`guard_type(x, int)`
	`x = int_add(x, 3)`

Tracing JITs

User program (lang *FL*)	Optimised trace

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

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
```

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)
```

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)
```

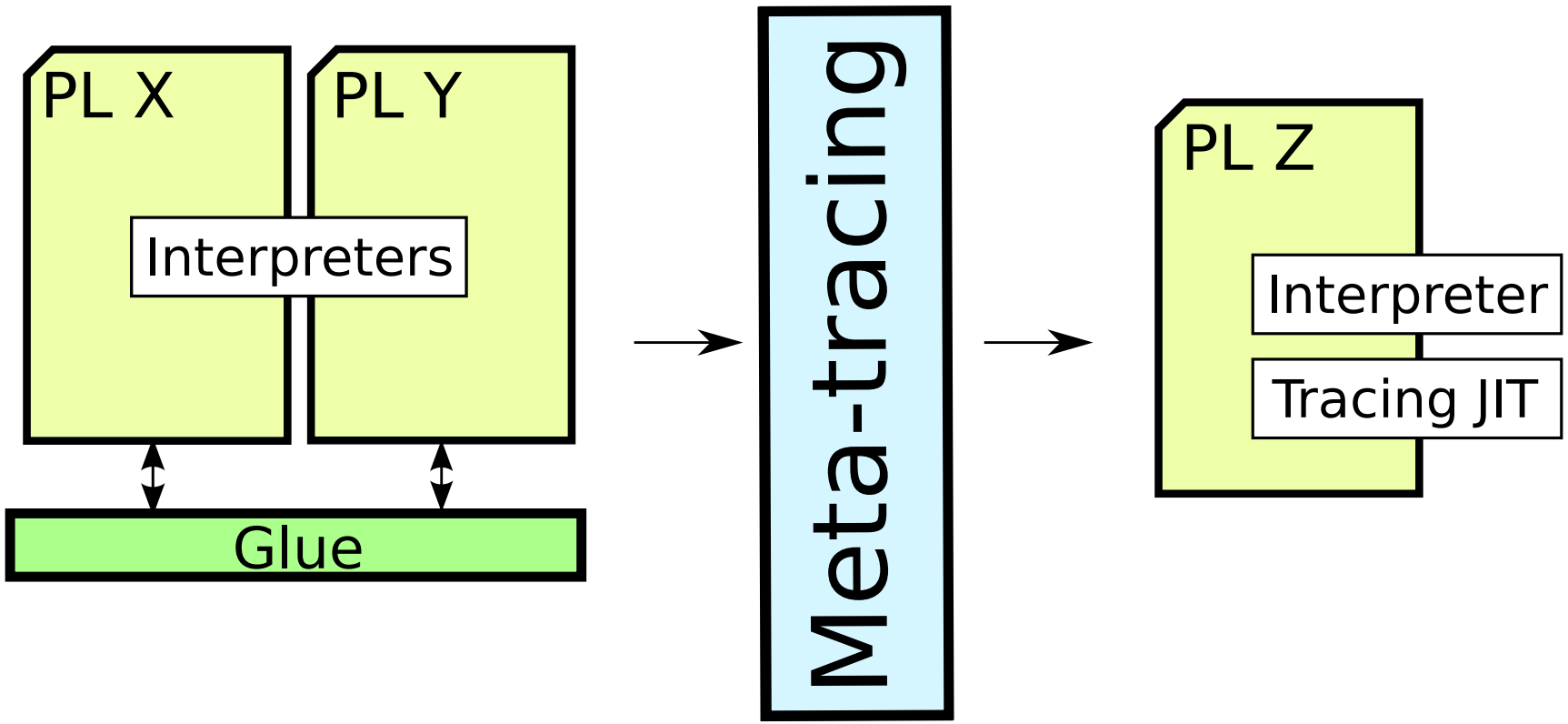
Arithmetic folded trace

```
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)
```

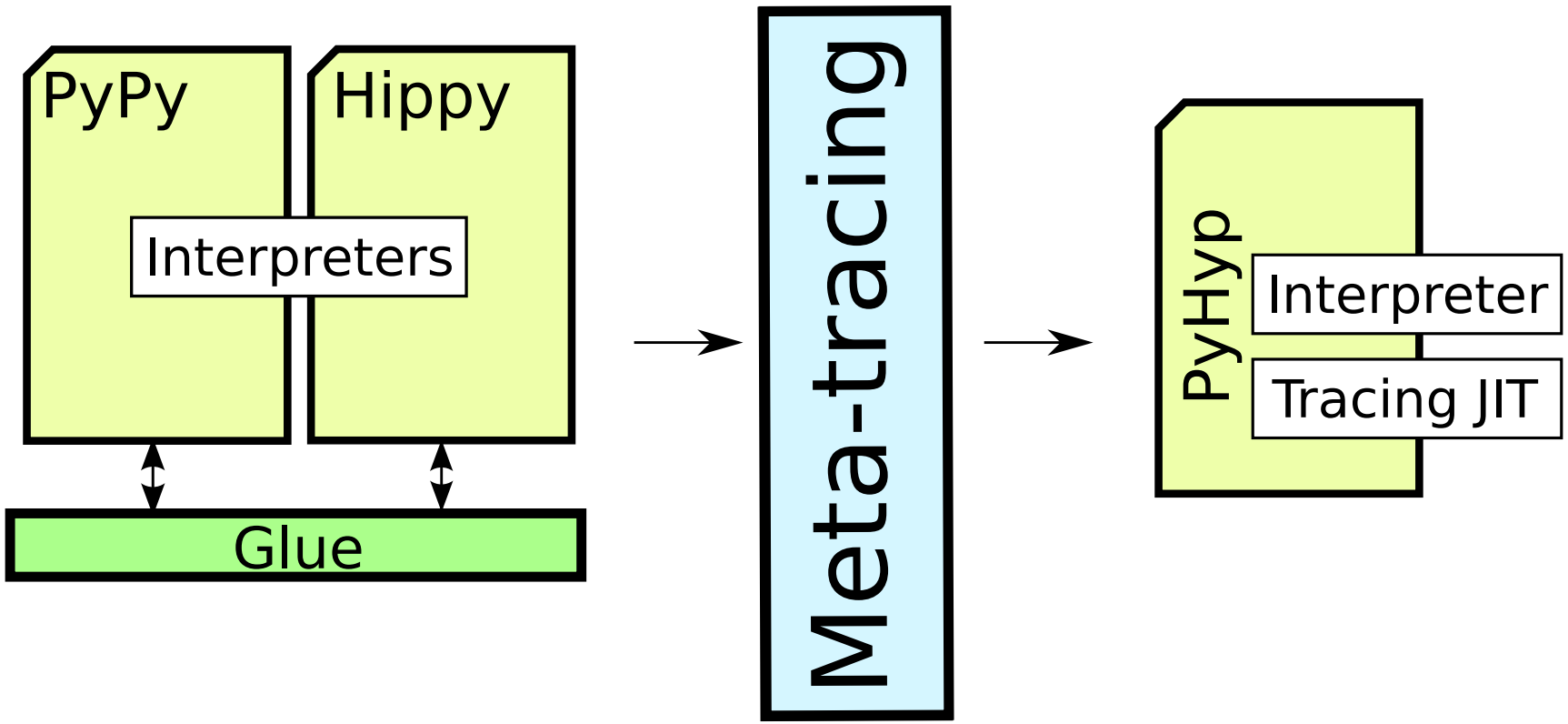
Trace optimisation: from 72 trace elements to 7.

Runtime composition recap

Runtime composition recap



Runtime composition recap



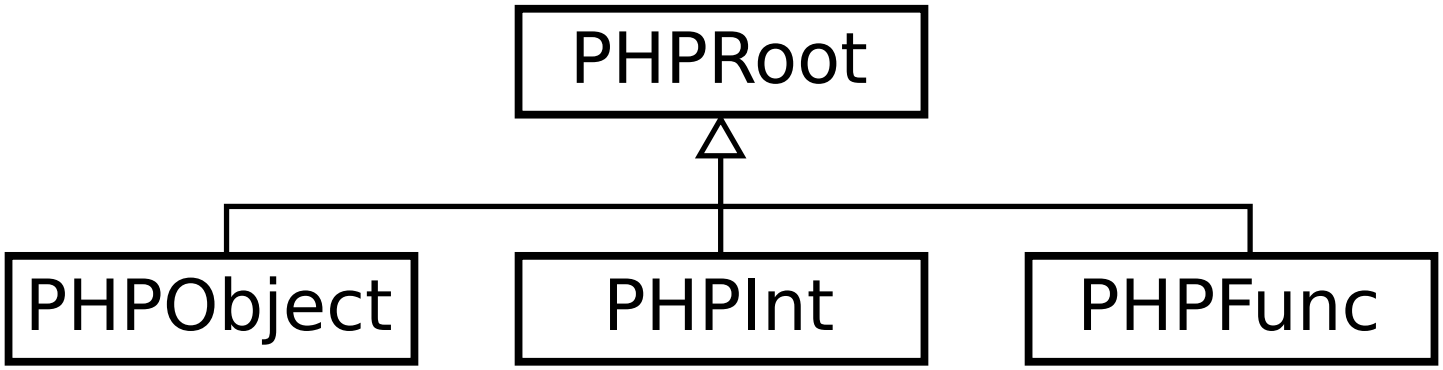
Composed Richards vs. other VMs

Type	VM	
Mono	CPython 2.7.7	9.475 ± 0.0127
	HHVM 3.4.0	4.264 ± 0.0386
	HippyVM	0.250 ± 0.0008
	PyPy 2.4.0	0.178 ± 0.0006
	Zend 5.5.13	9.070 ± 0.0361

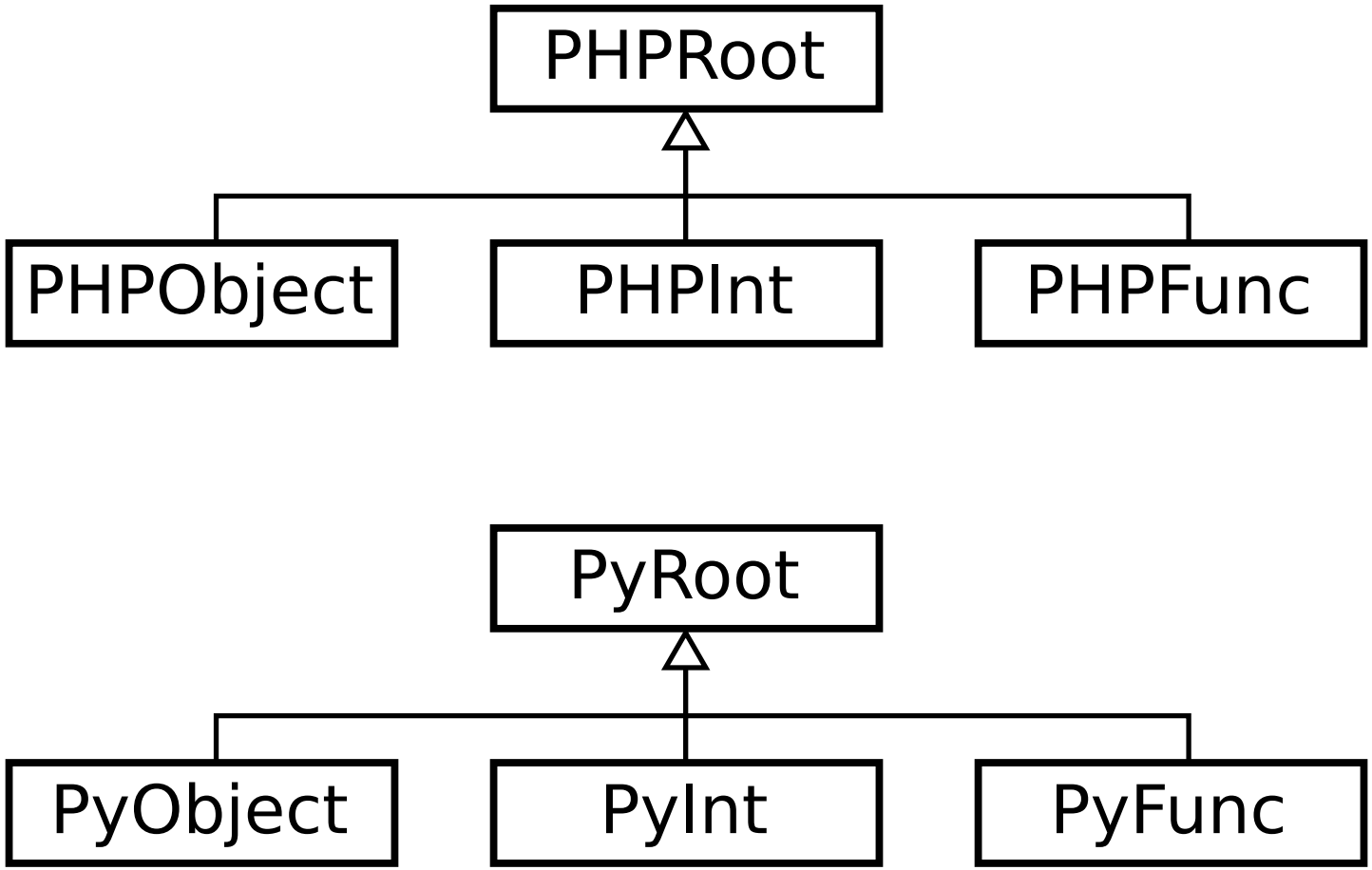
Composed Richards vs. other VMs

Type	VM	
Mono	CPython 2.7.7	9.475 ± 0.0127
	HHVM 3.4.0	4.264 ± 0.0386
	HippyVM	0.250 ± 0.0008
	PyPy 2.4.0	0.178 ± 0.0006
	Zend 5.5.13	9.070 ± 0.0361
Composed	PyHyp	0.335 ± 0.0012

Datatype conversion



Datatype conversion

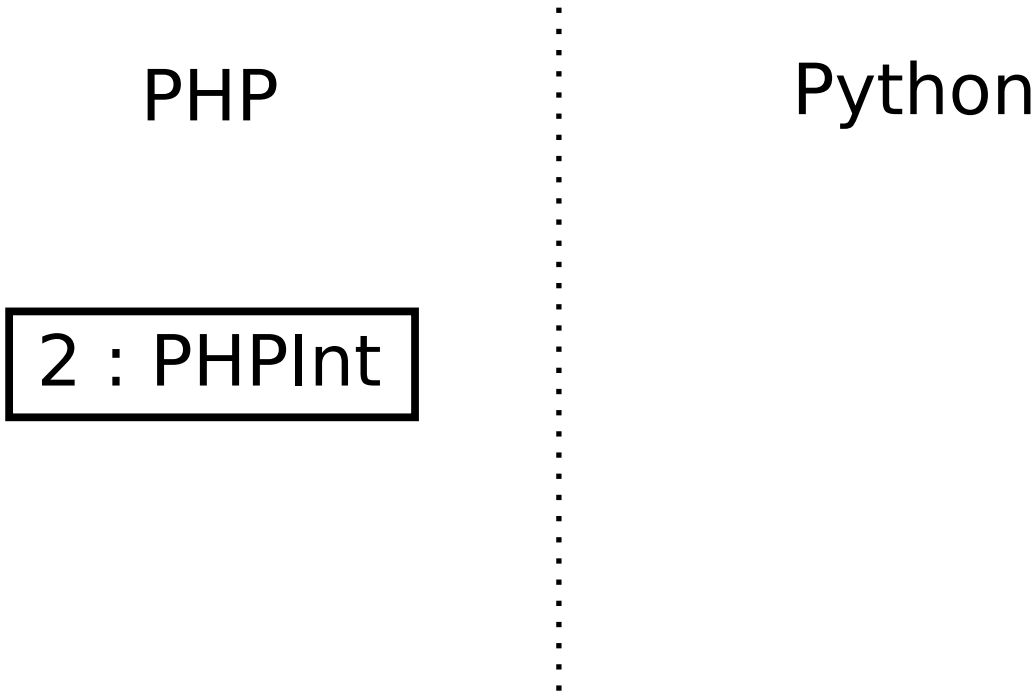


Datatype conversion: primitive types

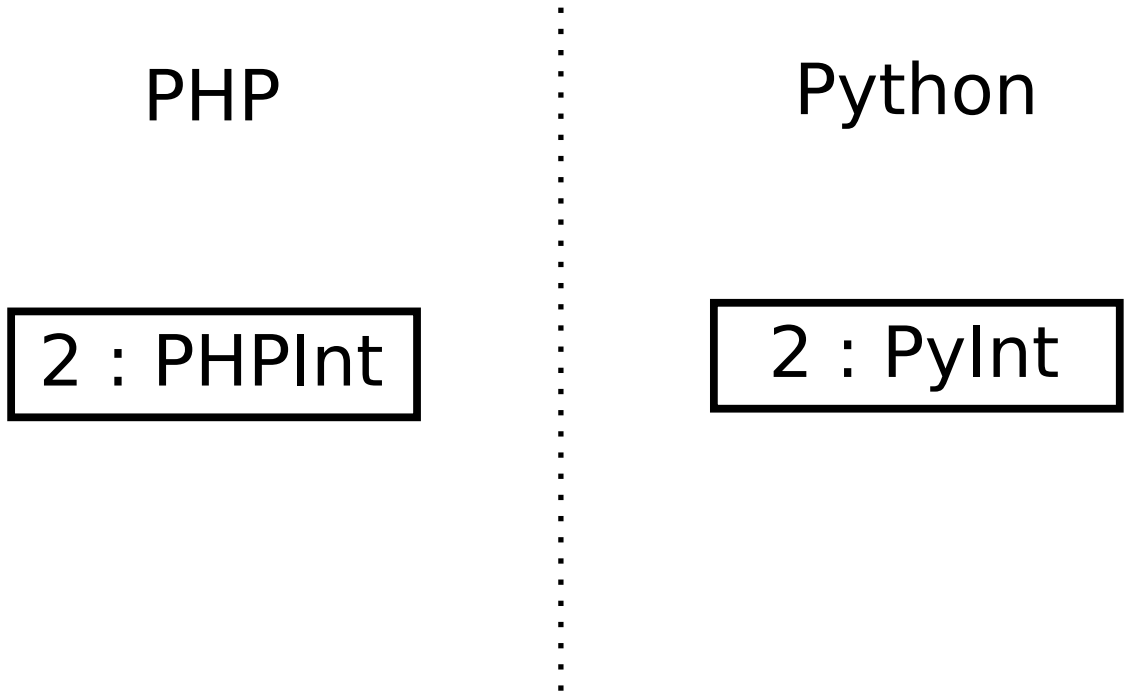
PHP

Python

Datatype conversion: primitive types



Datatype conversion: primitive types

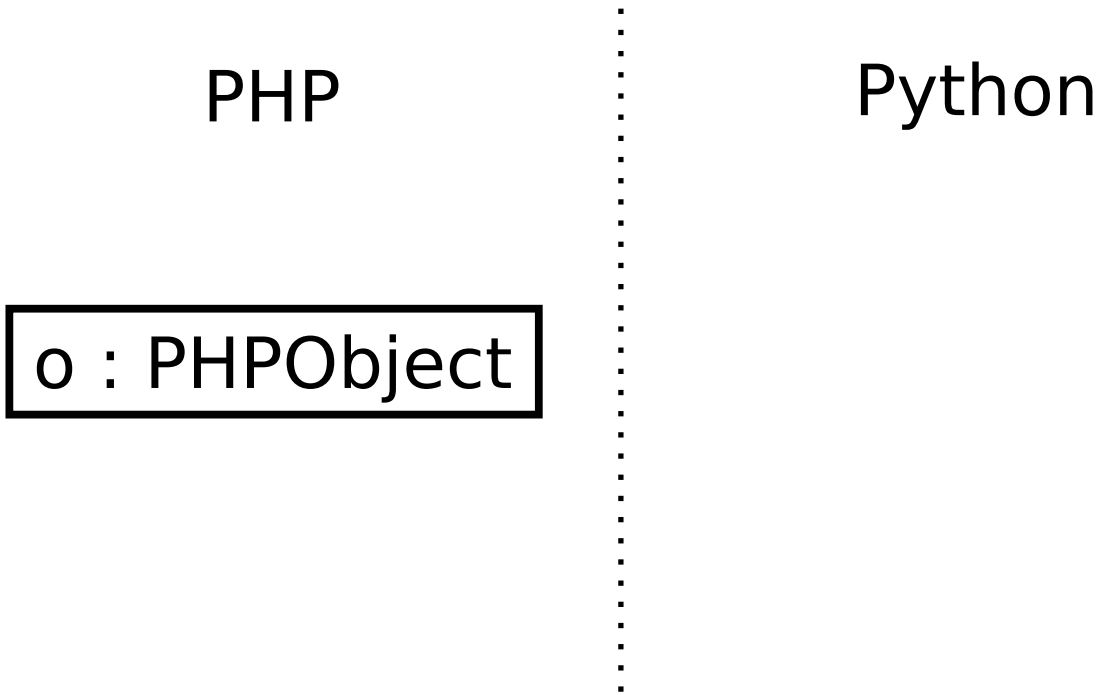


Datatype conversion: user types

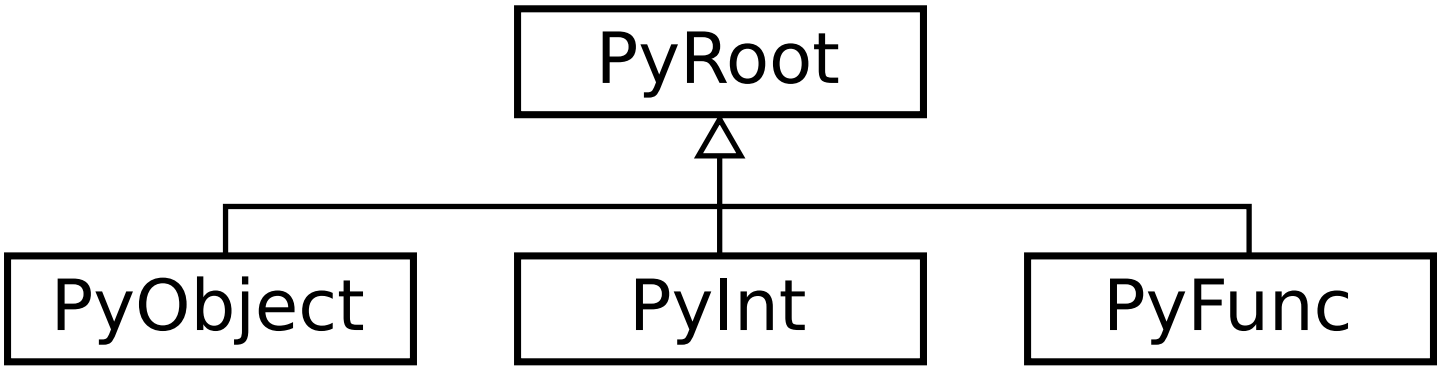
PHP

Python

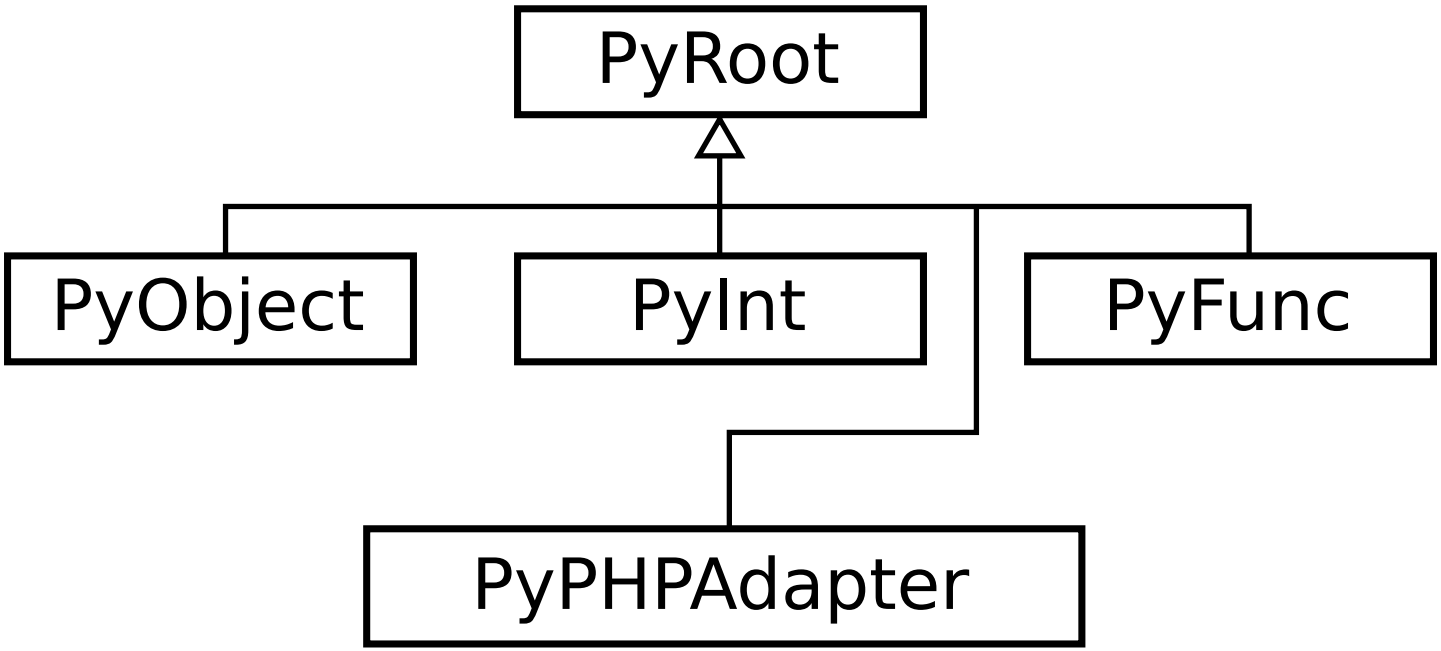
Datatype conversion: user types



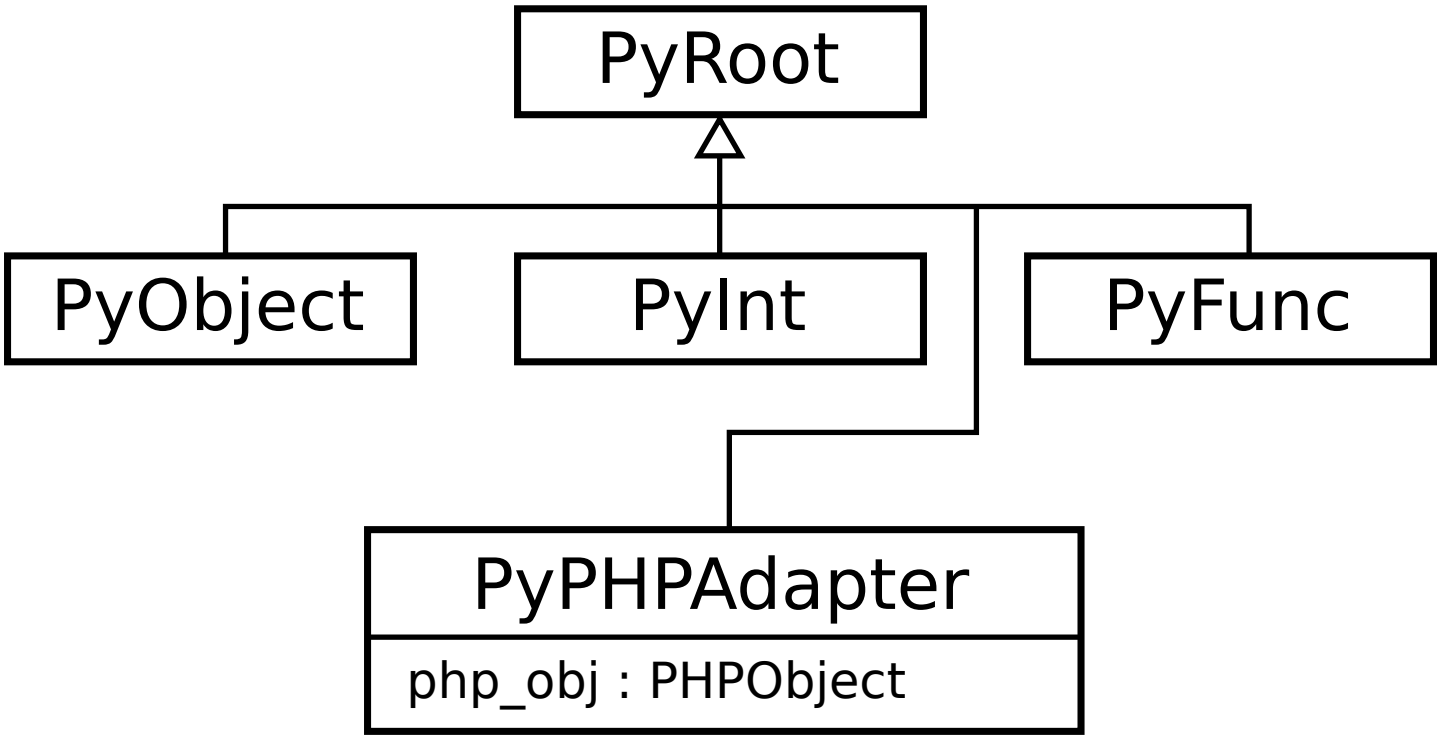
Datatype conversion: user types



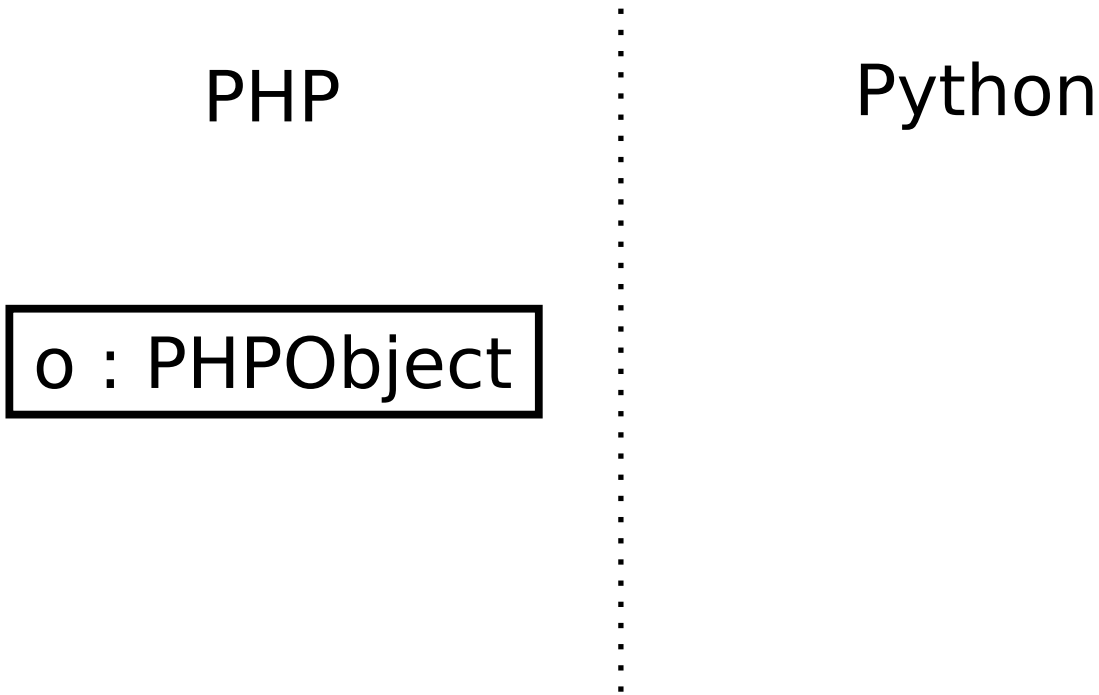
Datatype conversion: user types



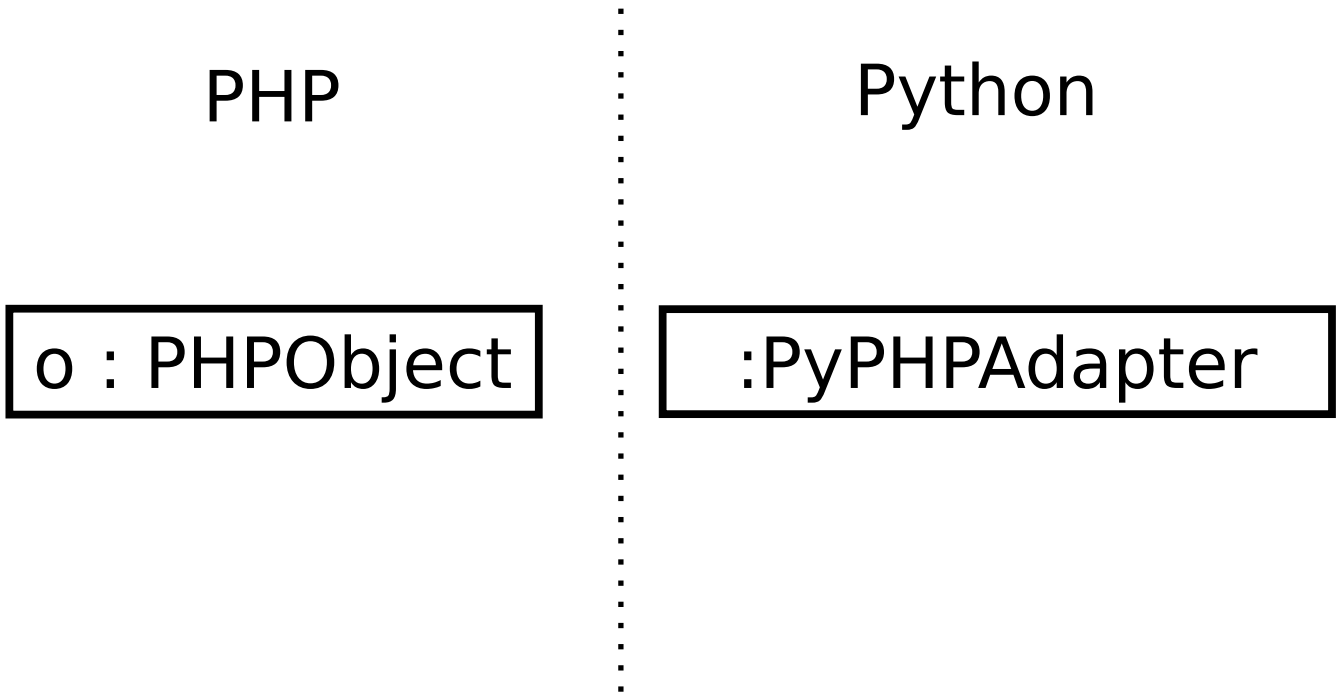
Datatype conversion: user types



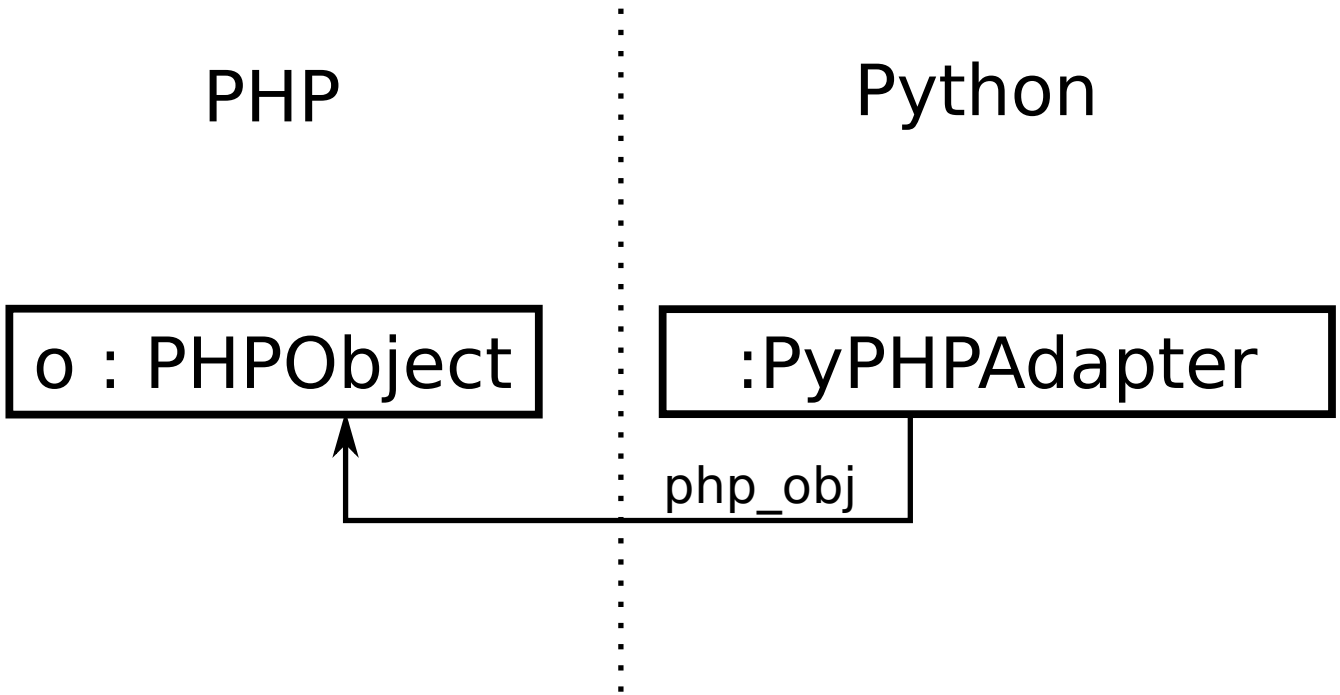
Datatype conversion: user types



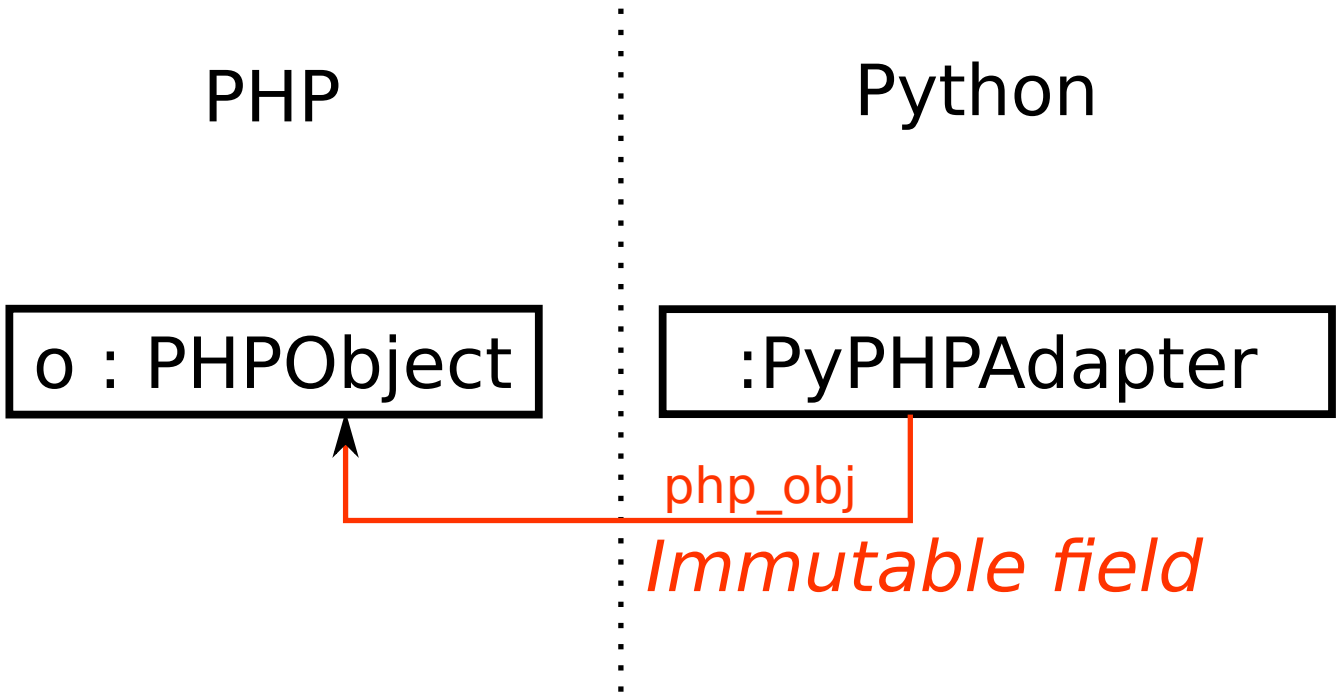
Datatype conversion: user types



Datatype conversion: user types



Datatype conversion: user types



Friction

A good composition needs to reduce *friction*.

Friction

A good composition needs to reduce *friction*. Some examples:

- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)

Friction

A good composition needs to reduce *friction*. Some examples:

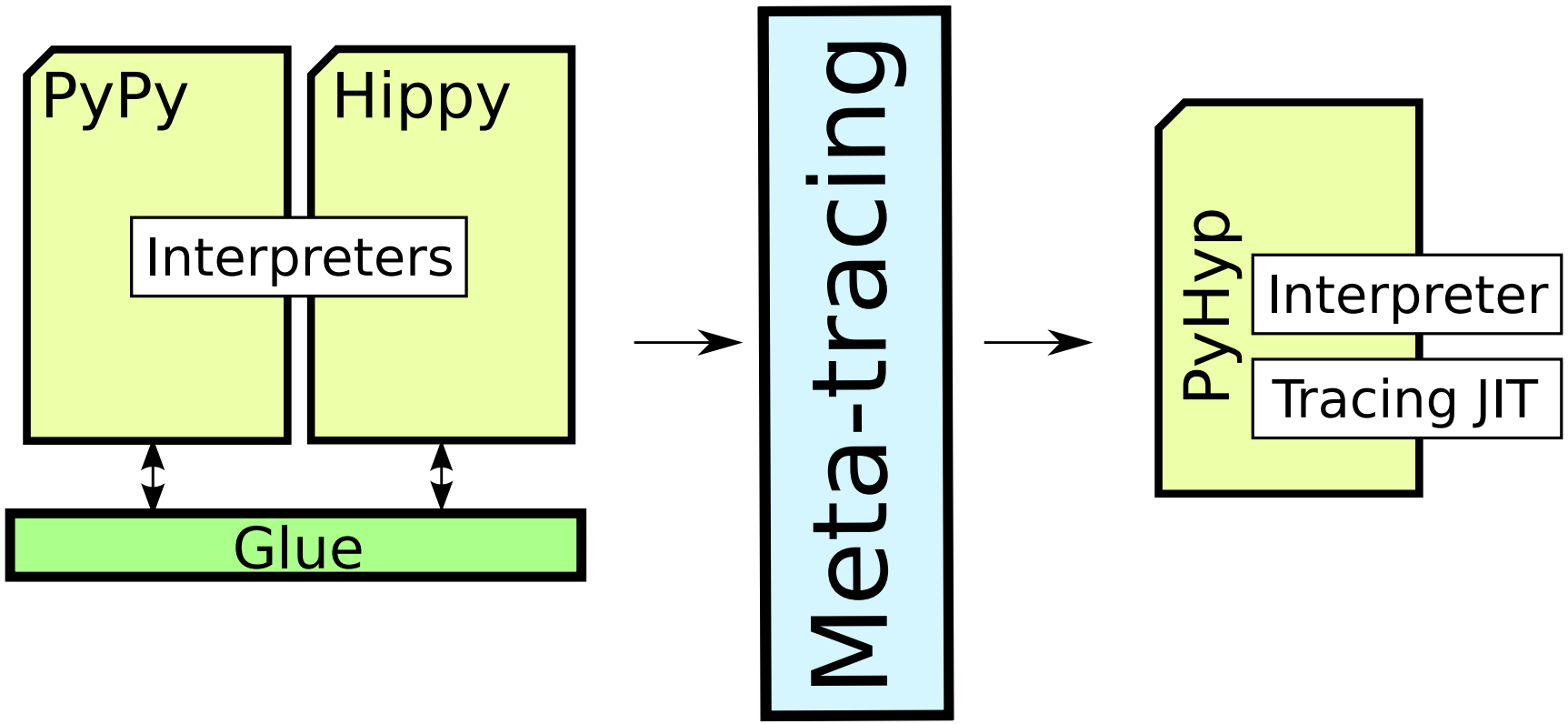
- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)
- PHP datatypes are immutable except for references and objects; Python's are largely mutable (semantic and performance friction)

Friction

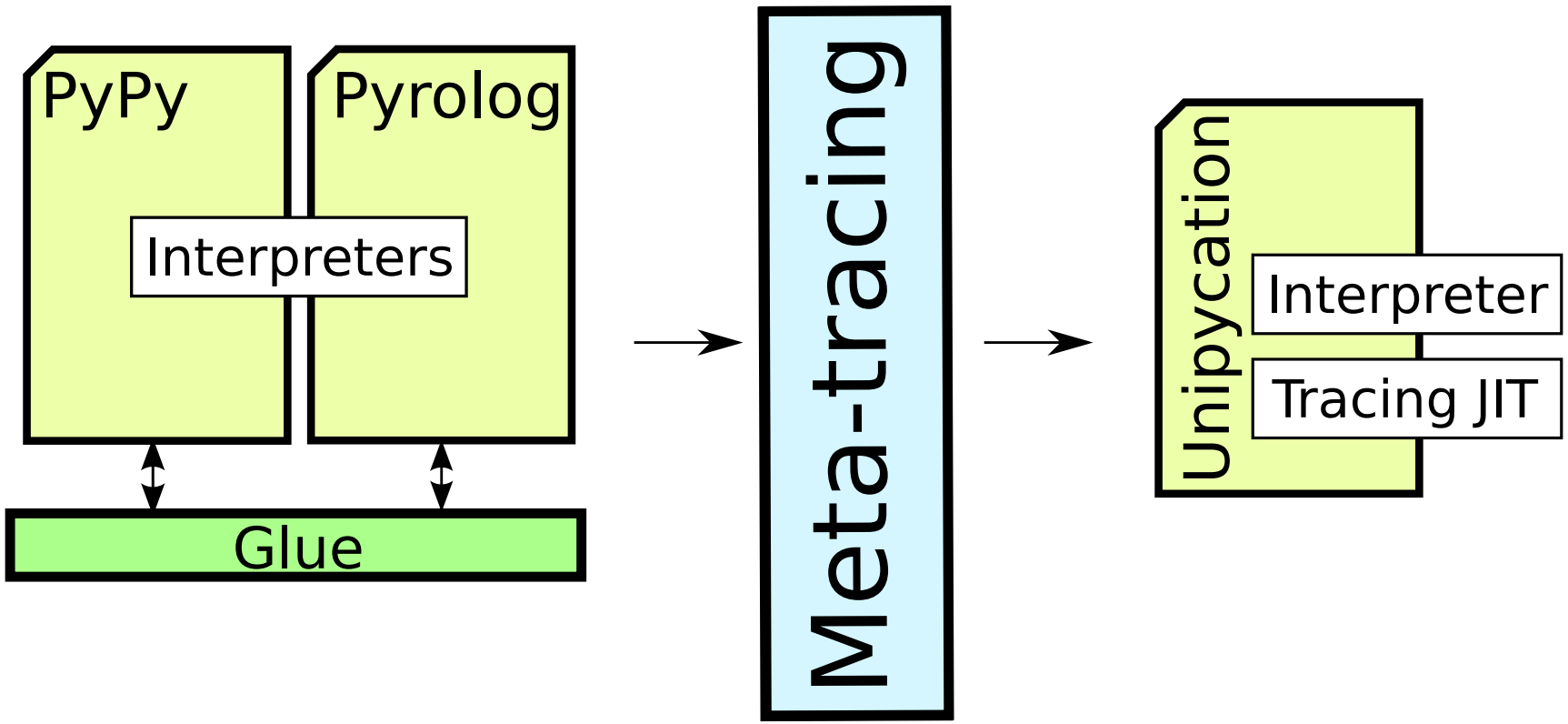
A good composition needs to reduce *friction*. Some examples:

- Lexical scoping (or lack thereof) in PHP and Python (semantic friction)
- PHP datatypes are immutable except for references and objects; Python's are largely mutable (semantic and performance friction)
- PHP has only dictionaries; Python has lists and dictionaries (semantic friction)

Unipycation



Unipycation



Unipycation demo

Absolute timing comparison

VM	Benchmark	*Python*		*Prolog*		*Python* → *Prolog*	
CPython-SWI	SmallFunc	0.125s	±0.007	0.257s	±0.002	28.893s	±0.227
	L1A0R	2.924s	±0.284	7.352s	±0.048	9.310s	±0.084
	L1A1R	4.184s	±0.038	18.890s	±0.111	20.865s	±0.067
	NdL1A1R	7.531s	±0.080	18.643s	±0.197	667.682s	±6.895
	TCons	264.415s	±2.250	48.819s	±0.252	2185.150s	±18.225
	Lists	9.374s	±0.059	25.148s	±0.221	2207.304s	±16.073
Unipycation	SmallFunc	0.001s	±0.000	0.006s	±0.001	0.001s	±0.000
	L1A0R	0.085s	±0.000	0.086s	±0.000	0.087s	±0.000
	L1A1R	0.112s	±0.000	0.114s	±0.000	0.115s	±0.000
	NdL1A1R	0.500s	±0.003	0.548s	±0.085	2.674s	±0.012
	TCons	6.053s	±0.288	2.444s	±0.003	36.069s	±0.225
	Lists	0.845s	±0.002	1.416s	±0.003	5.056s	±0.035
Jython-tuProlog	SmallFunc	0.088s	±0.003	3.050s	±0.053	52.294s	±0.475
	L1A0R	1.078s	±0.009	206.590s	±3.846	199.963s	±2.476
	L1A1R	2.145s	±0.232	293.311s	±5.691	294.781s	±6.193
	NdL1A1R	7.939s	±0.457	1857.687s	±5.169	1990.985s	±15.071
	TCons	543.347s	±3.289	8014.477s	±17.710	8202.362s	±24.904
	Lists	5.661s	±0.046	6981.873s	±18.795	5577.322s	±15.754

Relative timing comparison

VM	Benchmark	$\frac{Python \to Prolog}{Python}$		$\frac{Python \to Prolog}{Prolog}$		$\frac{Python \to Prolog}{\text{Unipycation}}$	
CPython-SWI	SmallFunc	231.770×	±13.136	112.567×	±1.242	27821.079×	±2331.665
	L1A0R	3.184×	±0.300	1.266×	±0.014	107.591×	±0.995
	L1A1R	4.987×	±0.049	1.105×	±0.007	181.899×	±0.590
	NdL1A1R	88.654×	±1.368	35.814×	±0.554	249.737×	±2.922
	TCons	8.264×	±0.101	44.760×	±0.453	60.583×	±0.637
	Lists	235.459×	±2.314	87.772×	±1.017	436.609×	±4.415
Unipycation	SmallFunc	1.295×	±0.105	0.182×	±0.054	1.000×	
	L1A0R	1.020×	±0.002	1.012×	±0.002	1.000×	
	L1A1R	1.025×	±0.002	1.002×	±0.003	1.000×	
	NdL1A1R	5.349×	±0.045	4.879×	±0.924	1.000×	
	TCons	5.959×	±0.282	14.756×	±0.092	1.000×	
	Lists	5.982×	±0.045	3.569×	±0.026	1.000×	
Jython-tuProlog	SmallFunc	592.904×	±19.517	17.143×	±0.338	50354.204×	±4341.413
	L1A0R	185.460×	±2.818	0.968×	±0.021	2310.844×	±28.093
	L1A1R	137.427×	±14.537	1.005×	±0.028	2569.873×	±52.847
	NdL1A1R	250.776×	±14.666	1.072×	±0.009	744.699×	±6.726
	TCons	15.096×	±0.106	1.023×	±0.004	227.409×	±1.592
	Lists	985.149×	±8.674	0.799×	±0.003	1103.206×	±8.338

What can we use this for?

What can we use this for?

First-class languages

What can we use this for?

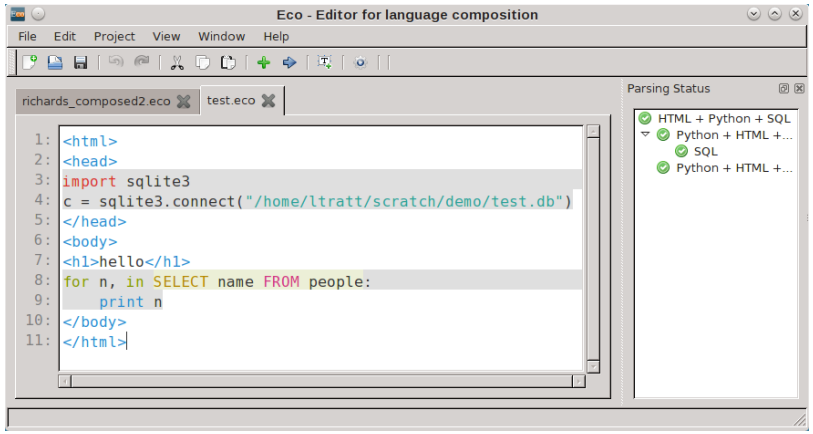
First-class languages

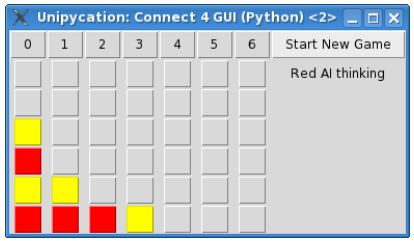
Language migration

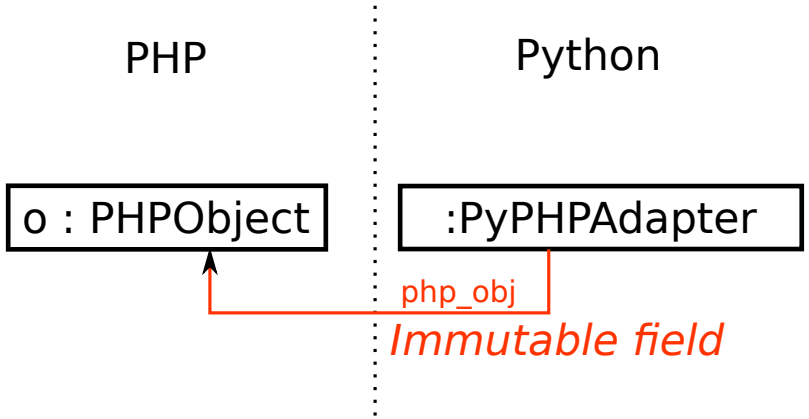
Thanks to our funders

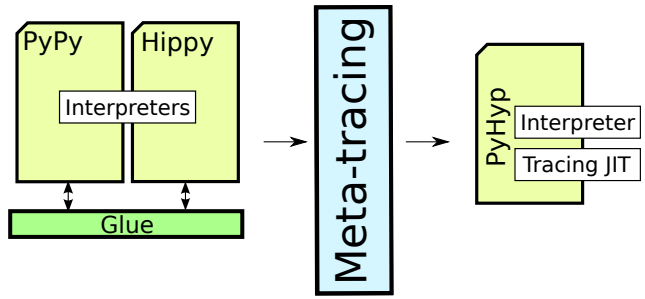
- EPSRC: *COOLER* and *Lecture*.
- Oracle: various.

Thanks for listening









http://soft-dev.org/