Towards language composition


Edd Barrett


Carl Friedrich Bolz


Lukas Diekmann


Laurence Tratt


Naveneetha Krishnan Vasudevan

KING'S College LONDON

Software Development Team

2014-01-24

Our problem

Titian: Sisyphus

Our problem

Titian: Sisyphus

Designing a perfect programming language

Our problem

We want **better** programming languages

Our problem

We want **better** programming languages

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

A solution?

Titian: Pesaro Madonna

A solution?

Titian: Pesaro Madonna

Language composition as salvation?

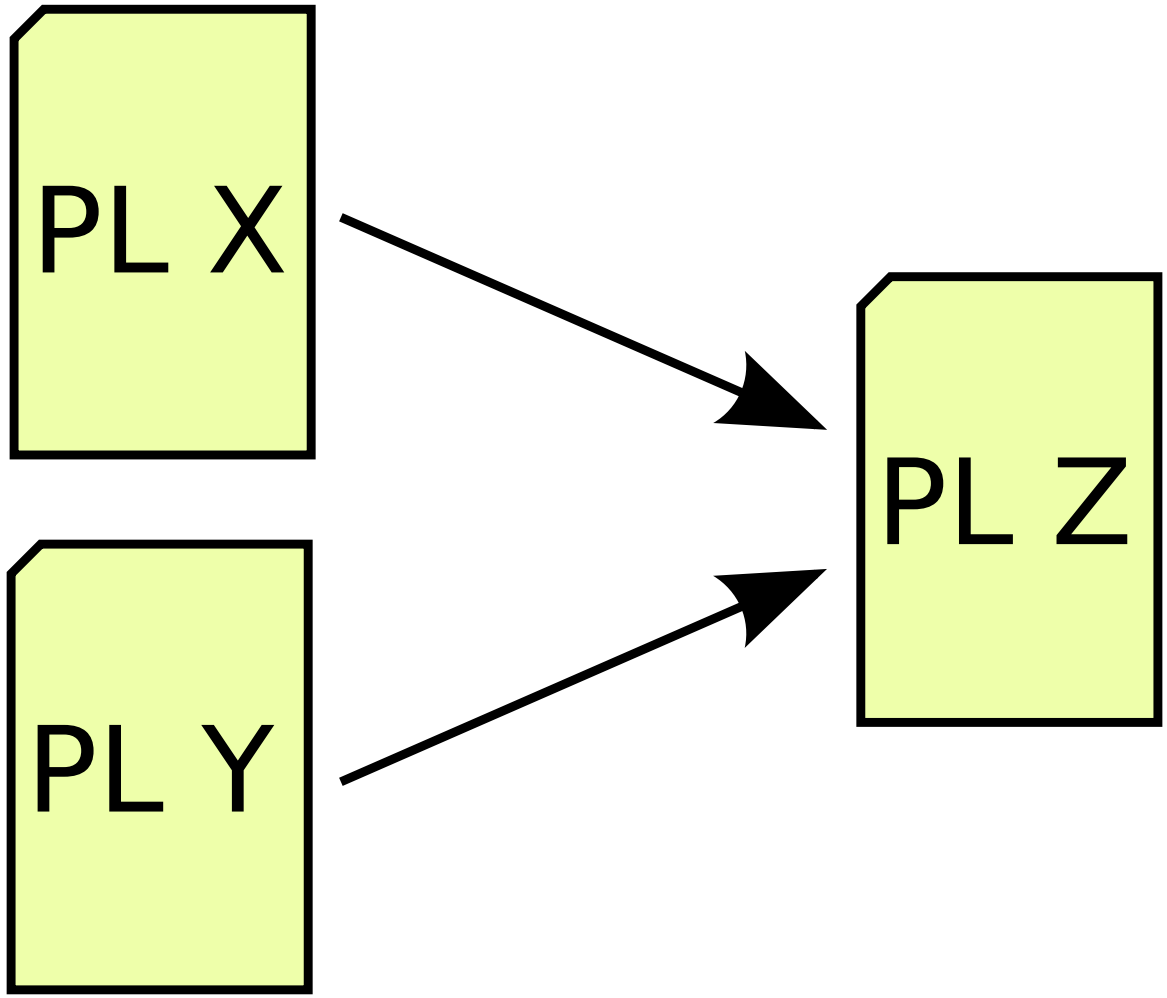
Language composition

Language composition

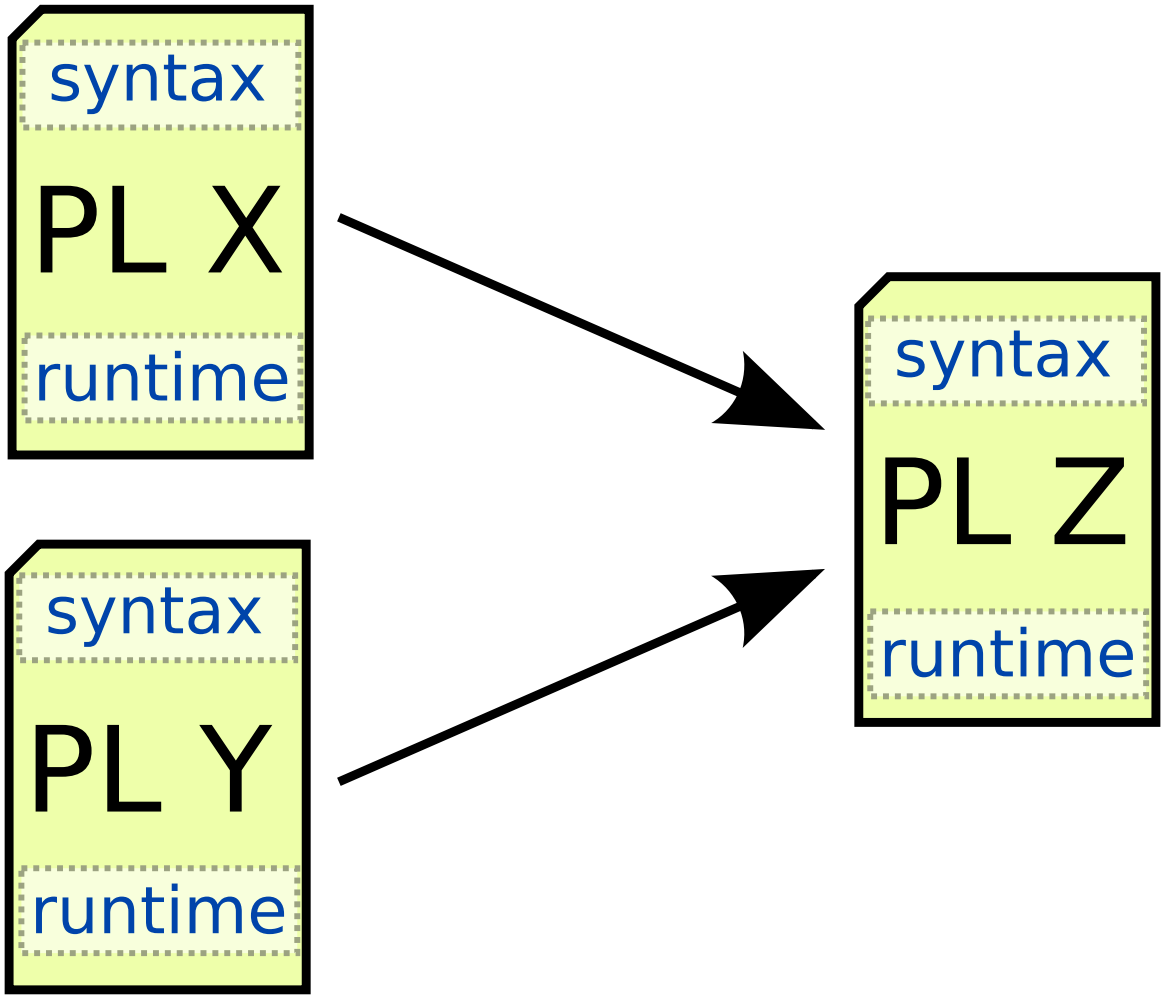
$\triangleq$

mixing languages together

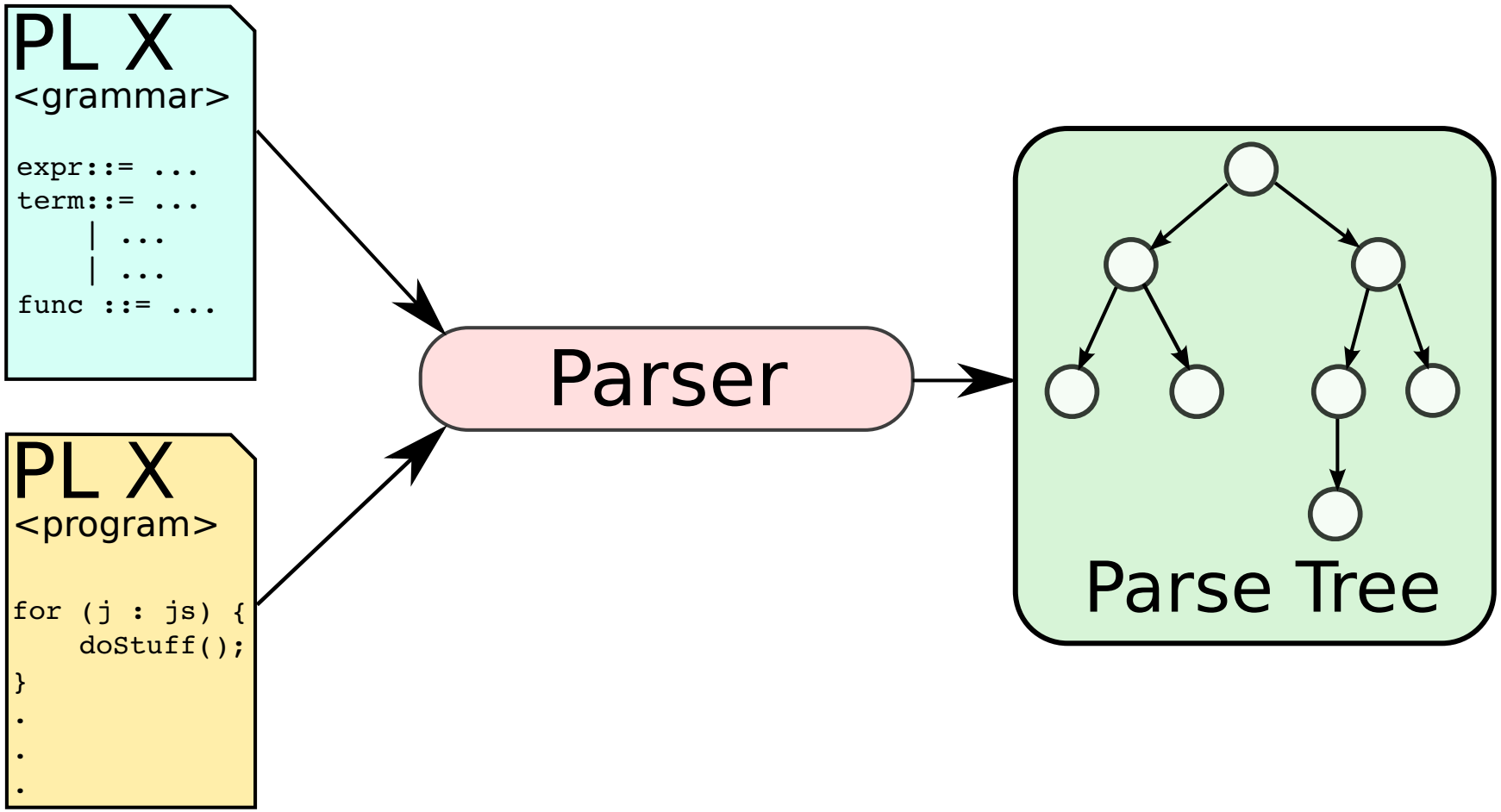
Language composition



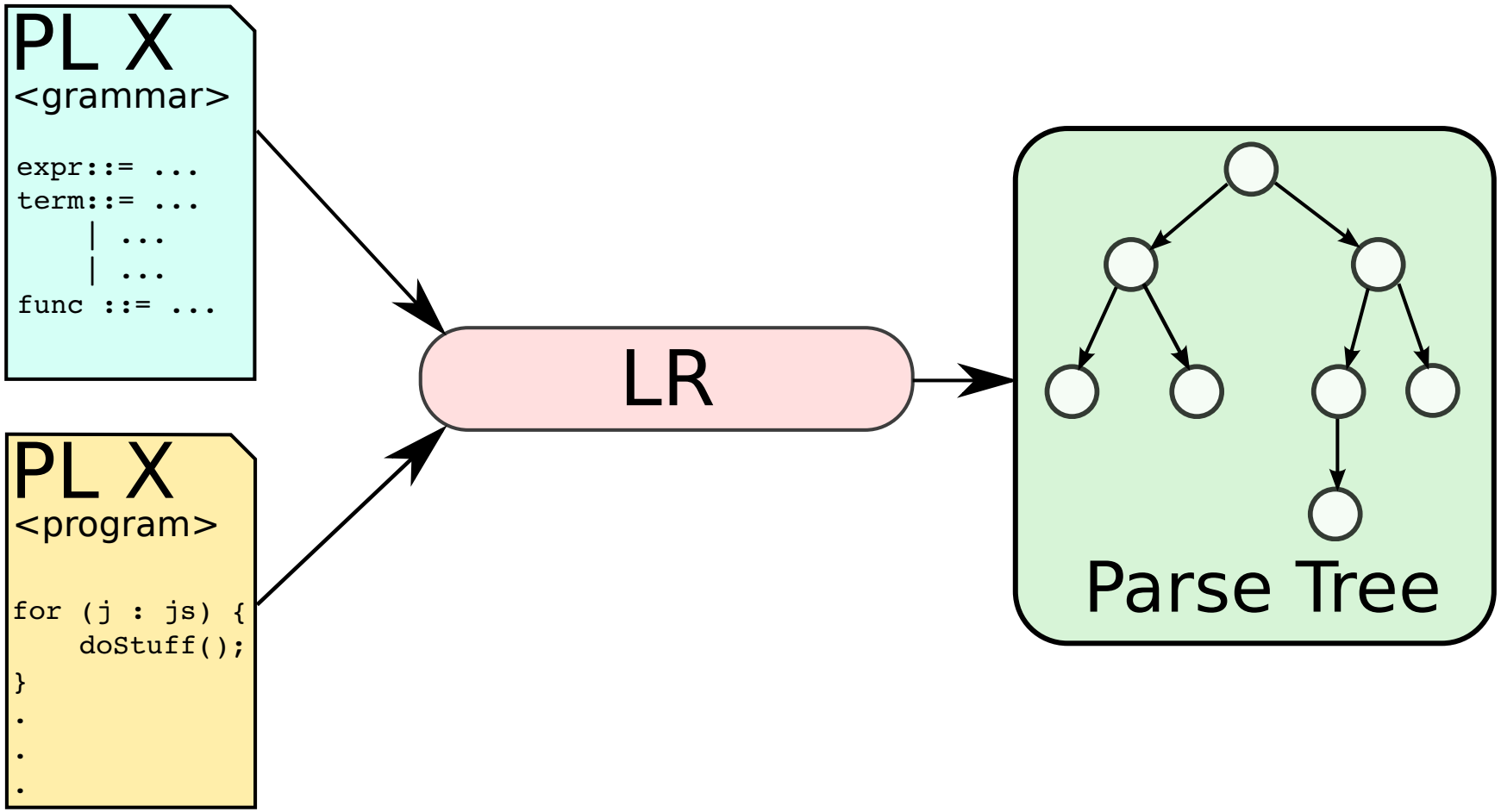
Language composition



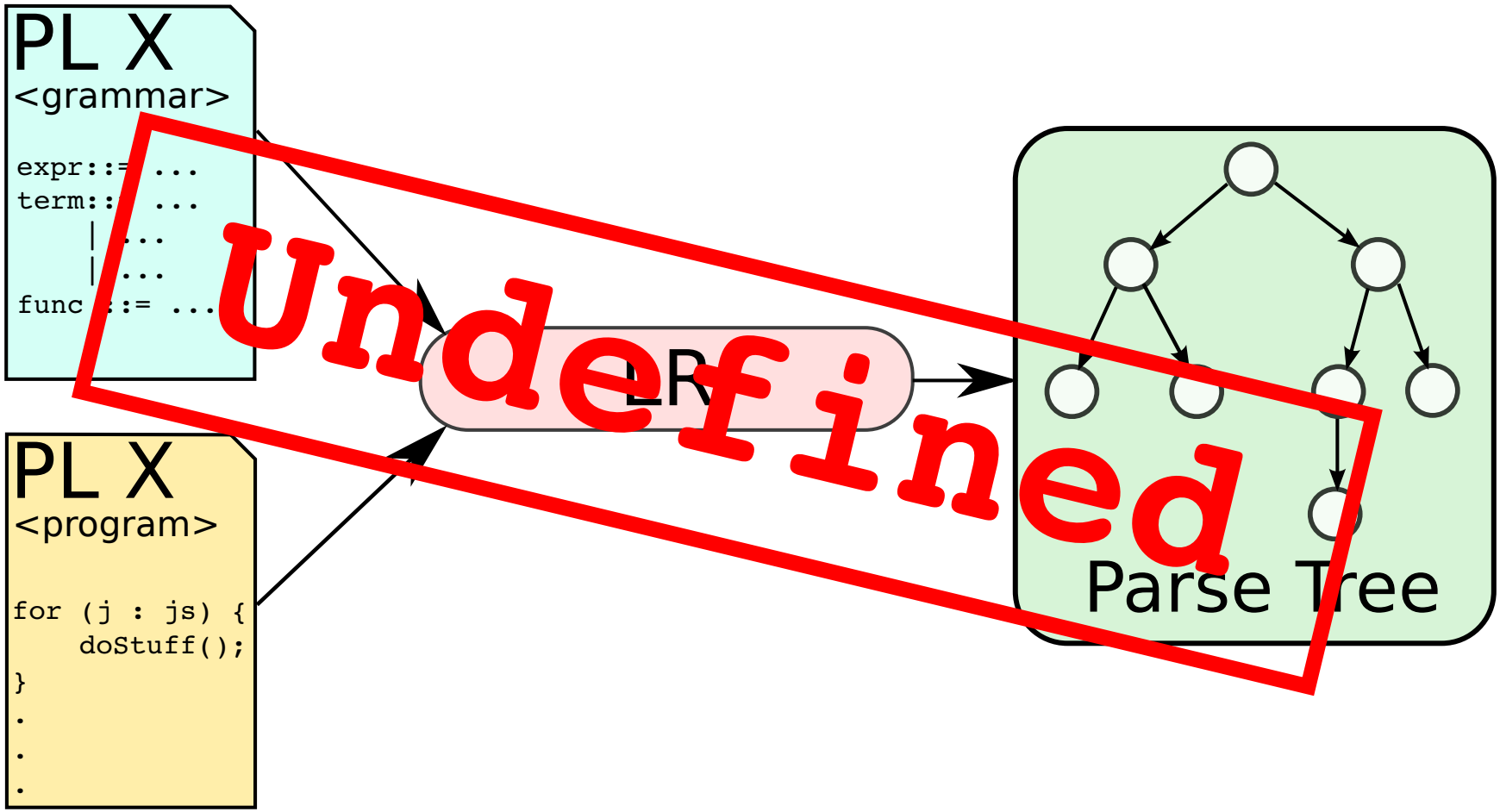
Parsing composition



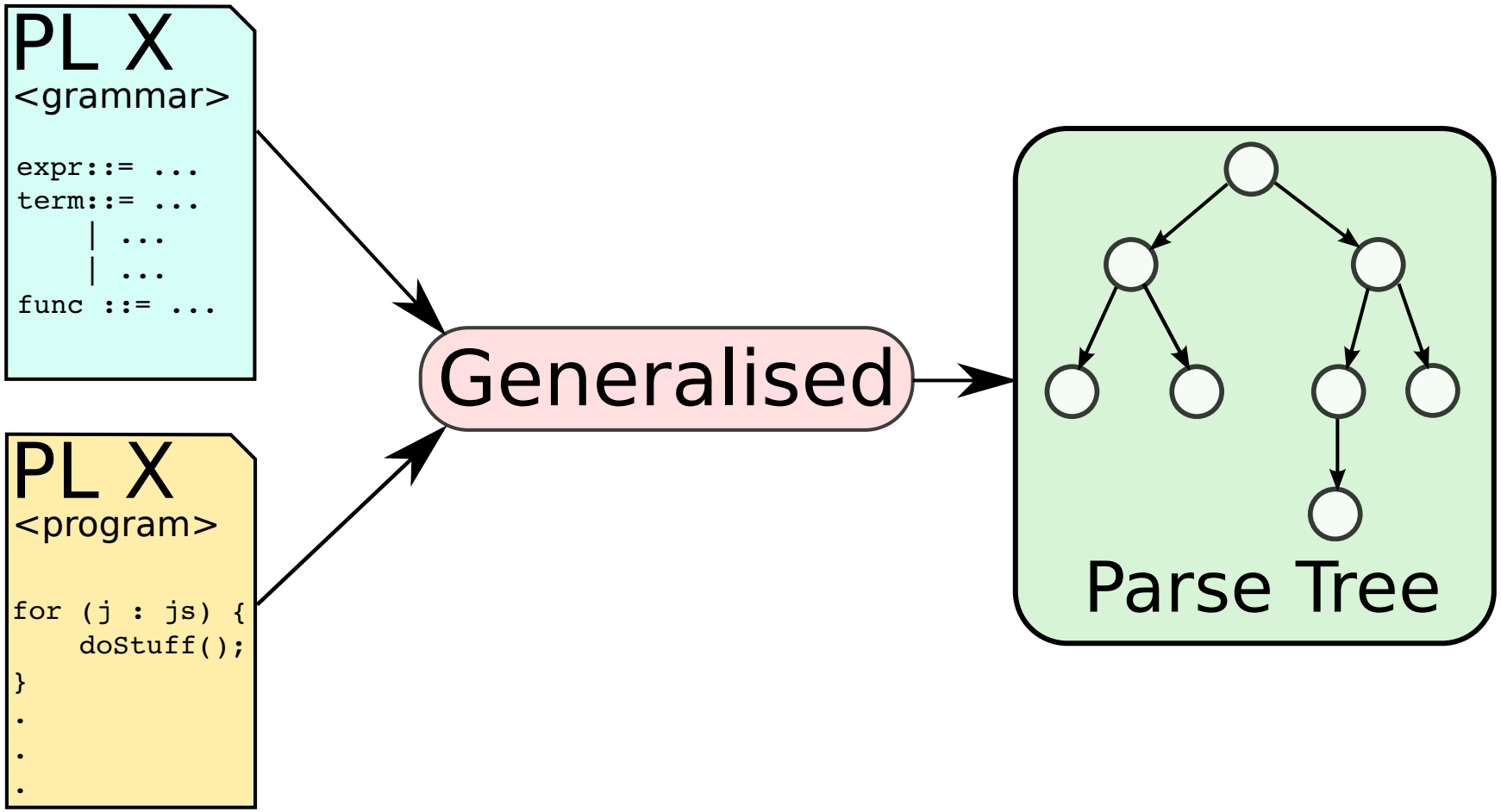
Parsing composition



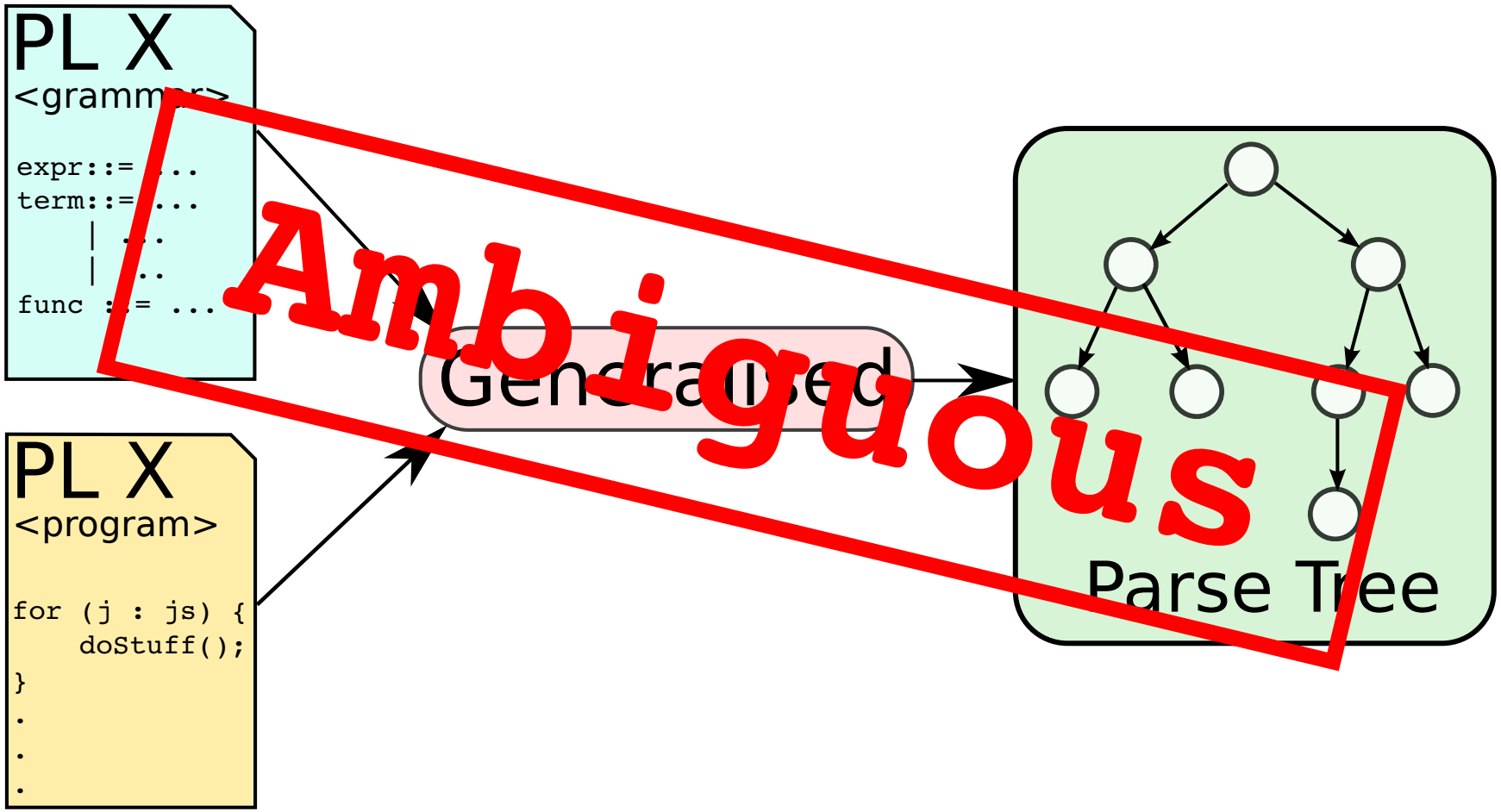
Parsing composition



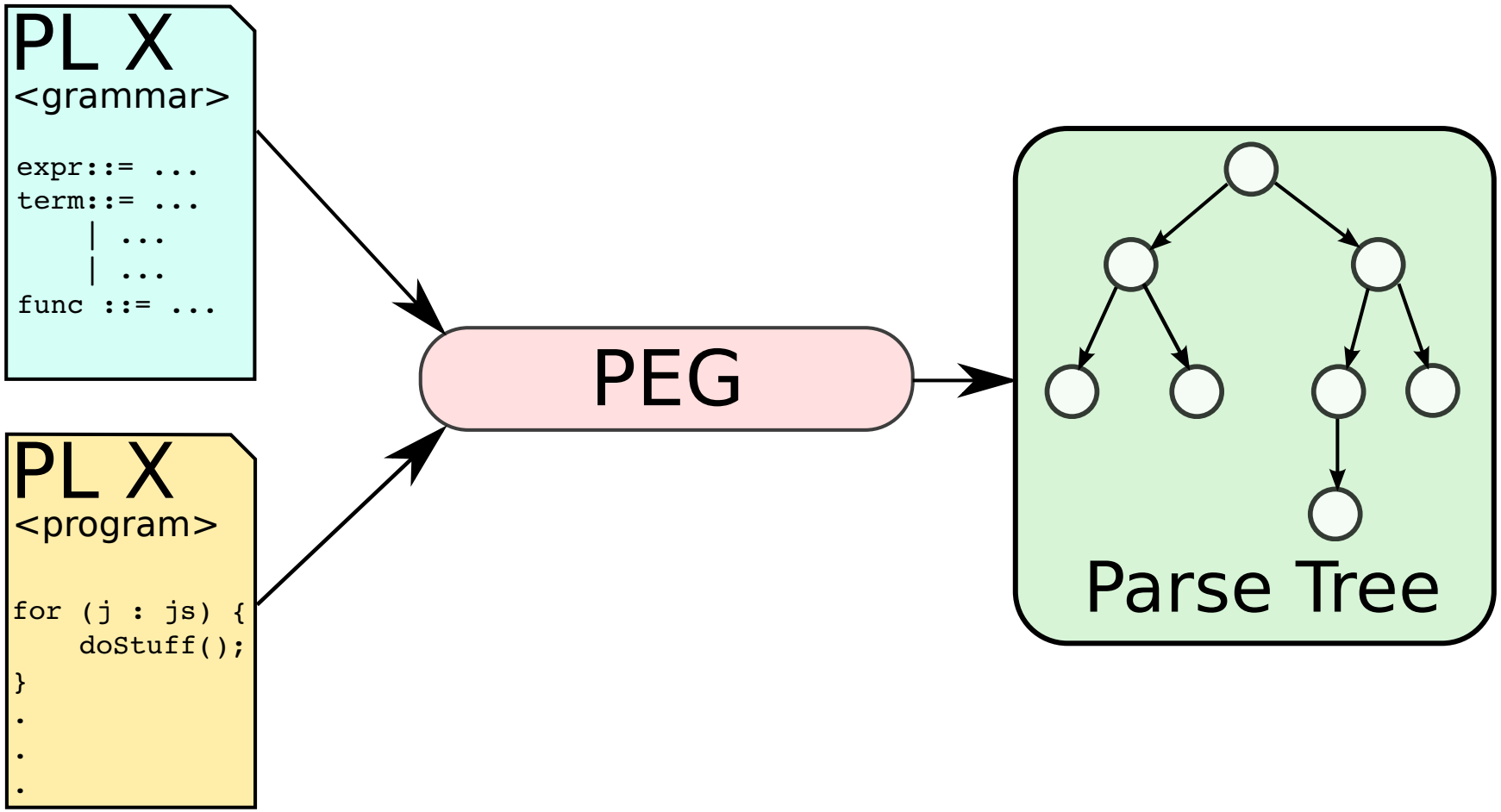
Parsing composition



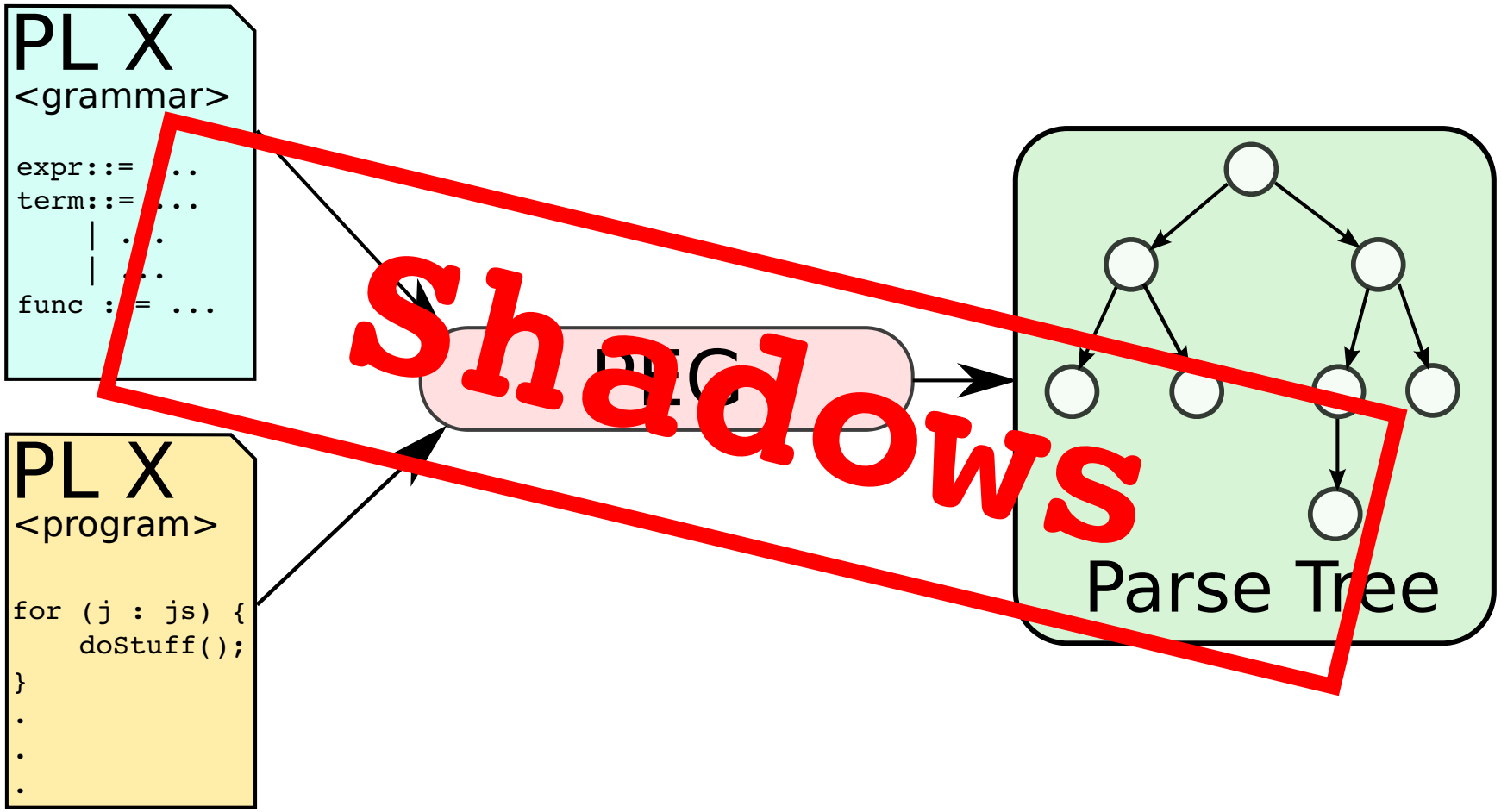
Parsing composition



Parsing composition



Parsing composition



The only choice?

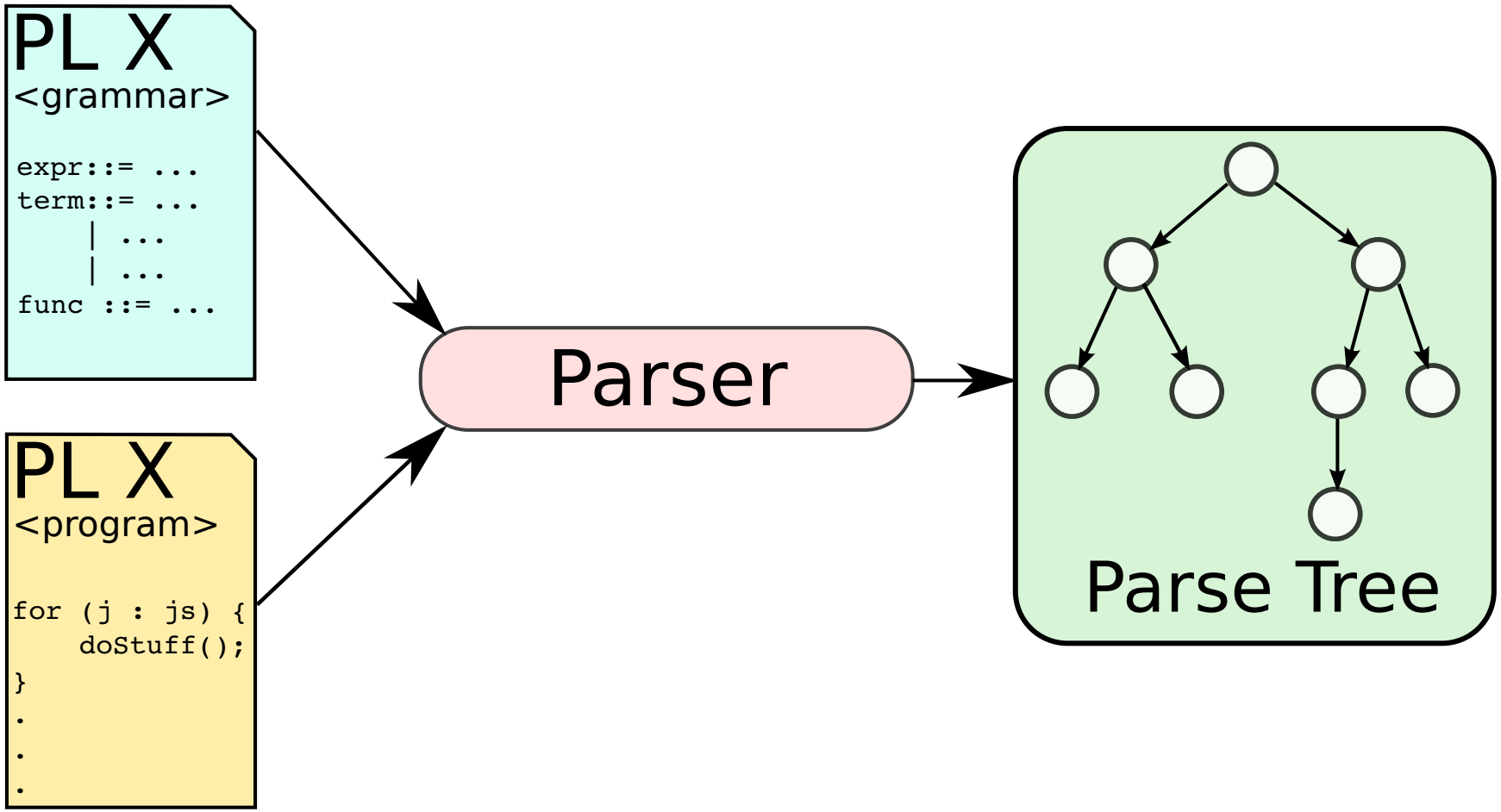
The only choice?

SDE

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

Eco demo

Eco


PL X
<grammar>
expr::= ...
term::= ...
| ...
| ...
func ::= ...
PL X
<program>
for (j : js) {
doStuff();
}
.
.
.
Parser
Parse Tree

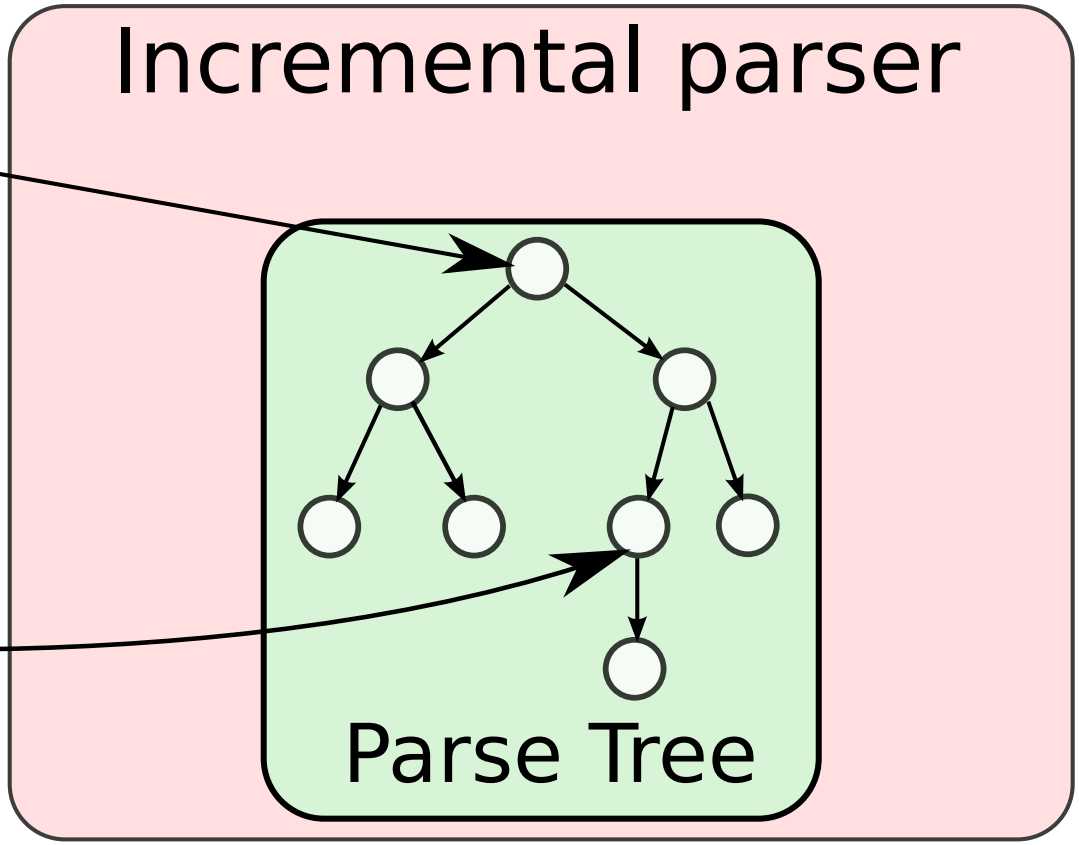
Eco

PL X
<grammar>

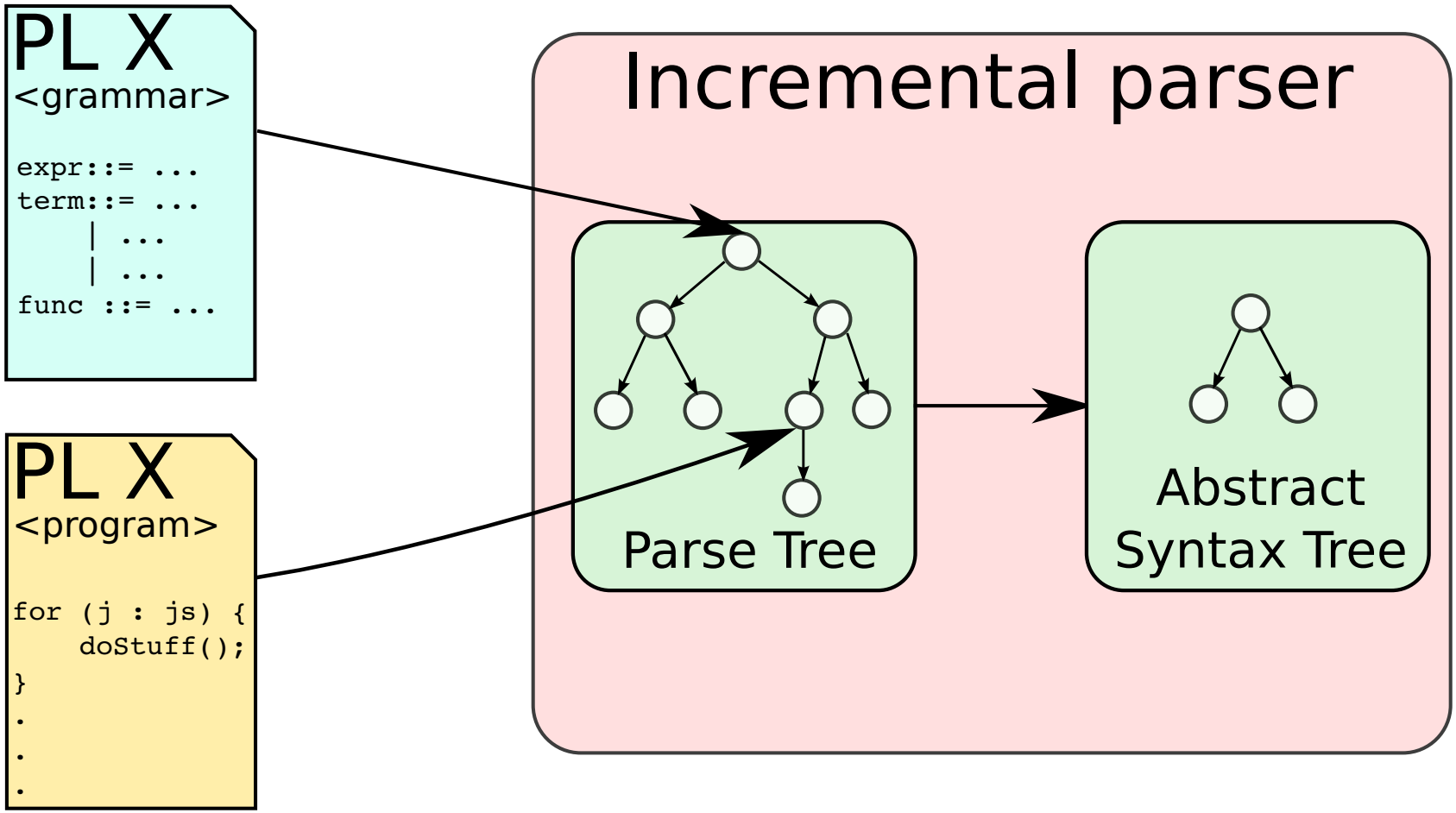
```
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...
```

PL X
<program>

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

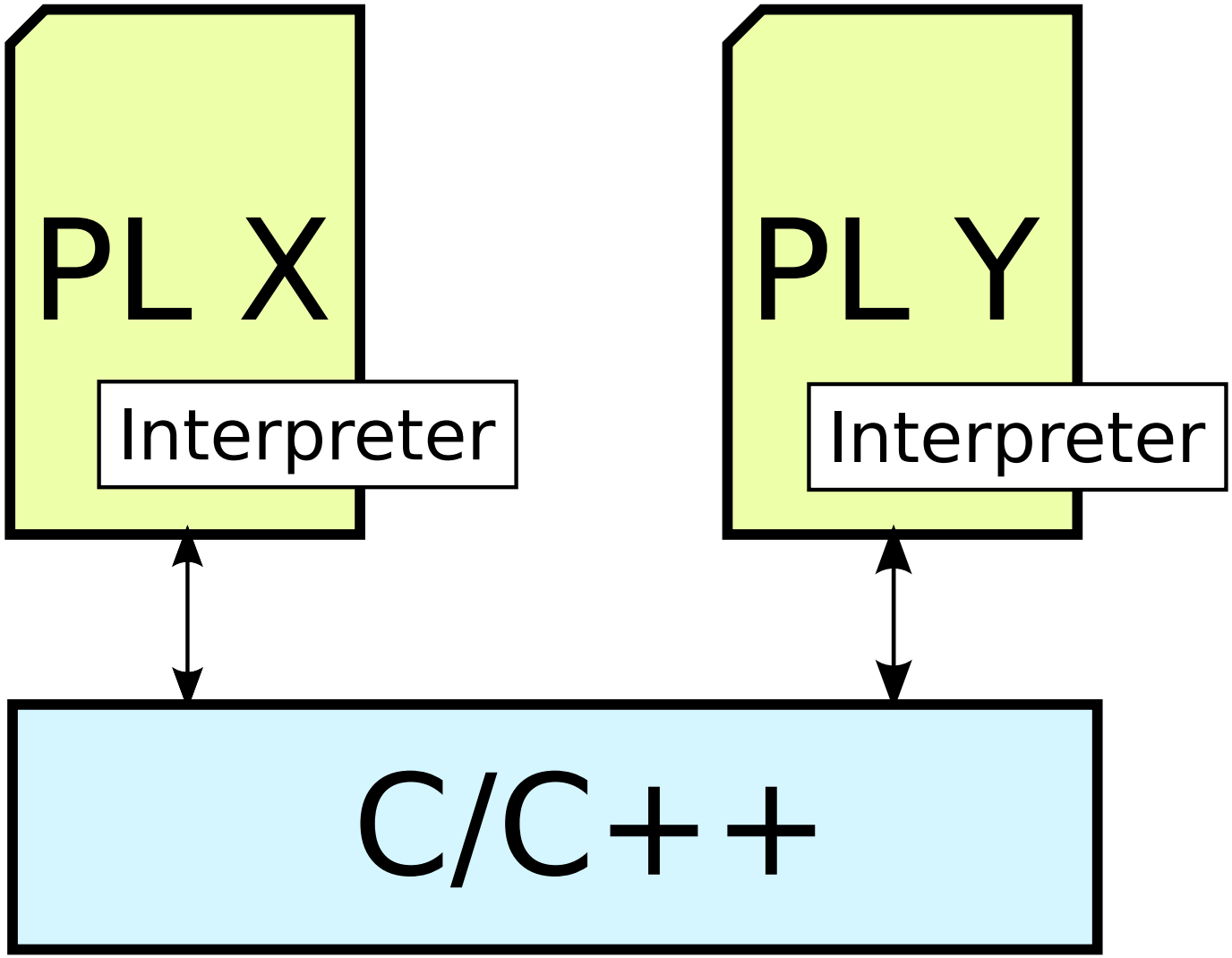


Eco

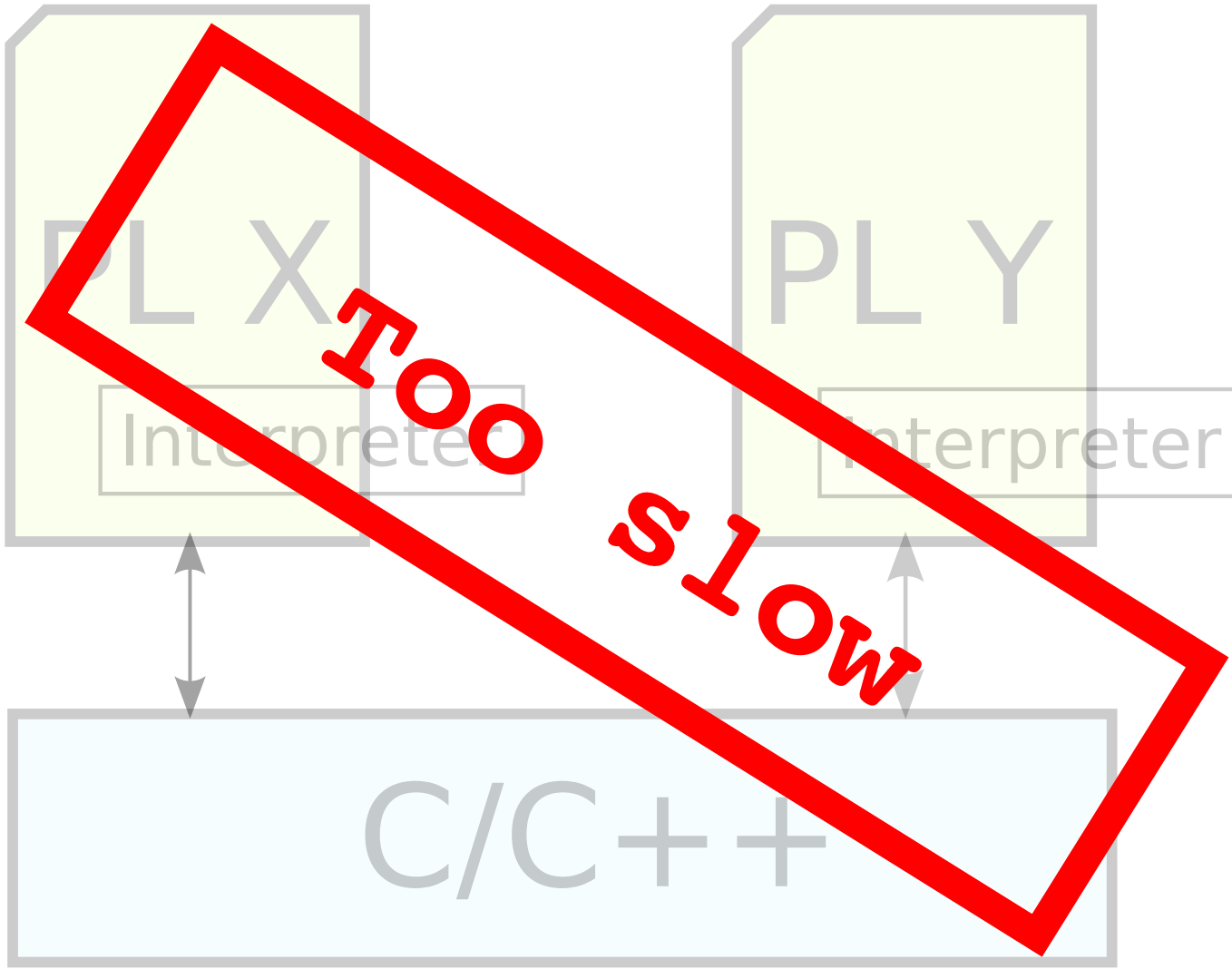

PL X
<grammar>
expr::= ...
term::= ...
| ...
| ...
func ::= ...
PL X
<program>
for (j : js) {
doStuff();
}
.
.
.
Incremental parser
Parse Tree
Abstract
Syntax Tree

Runtime composition

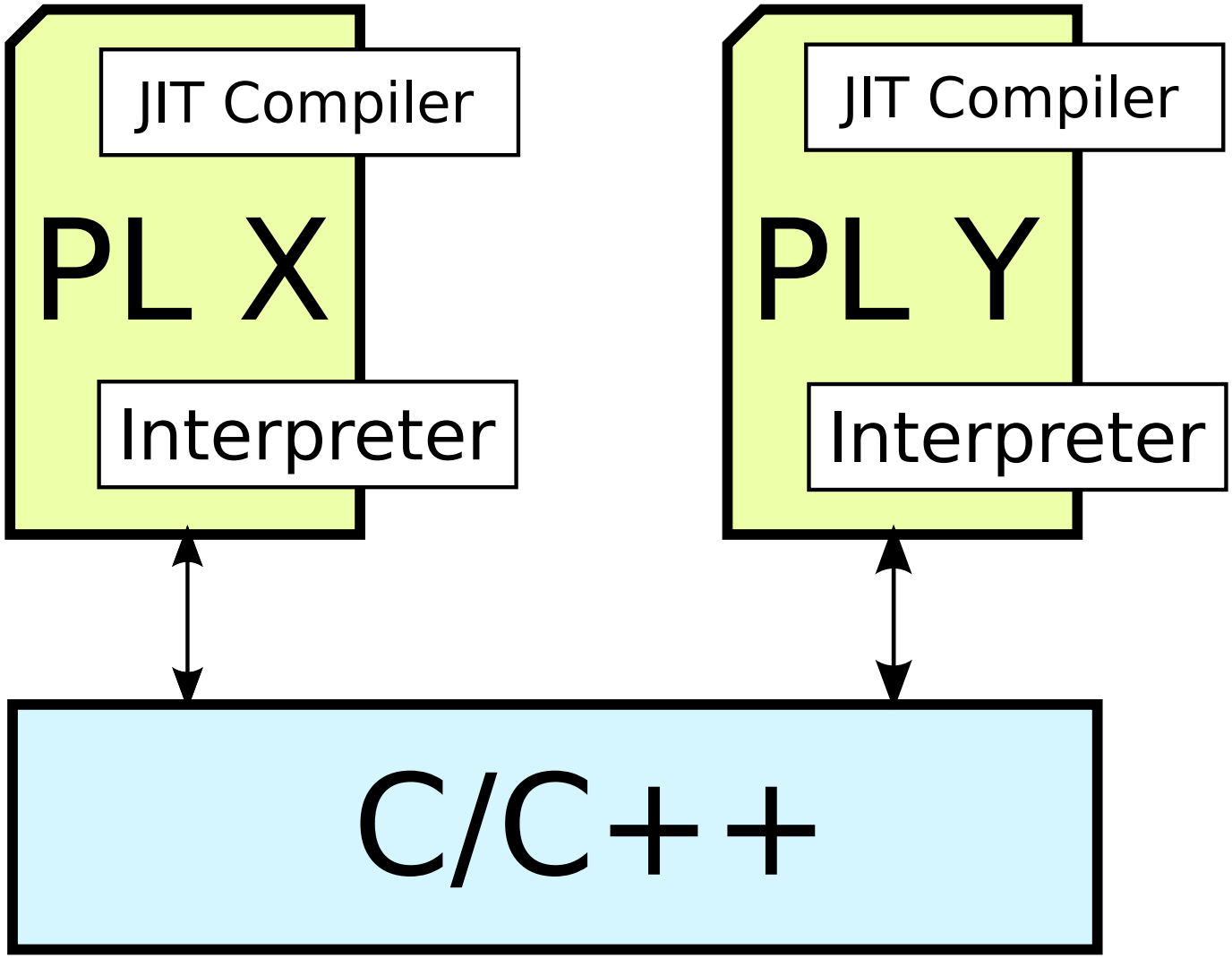
Runtime composition



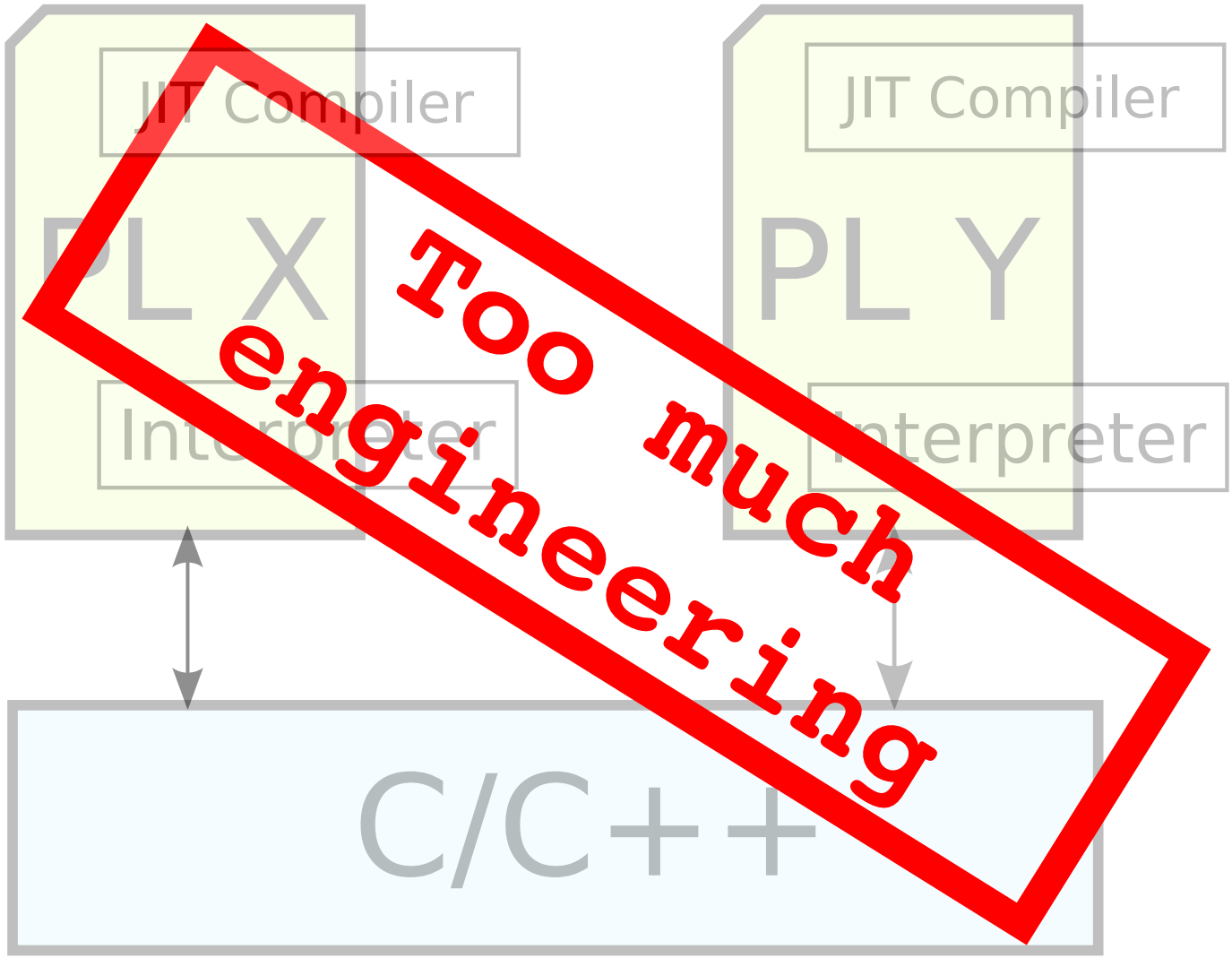
Runtime composition



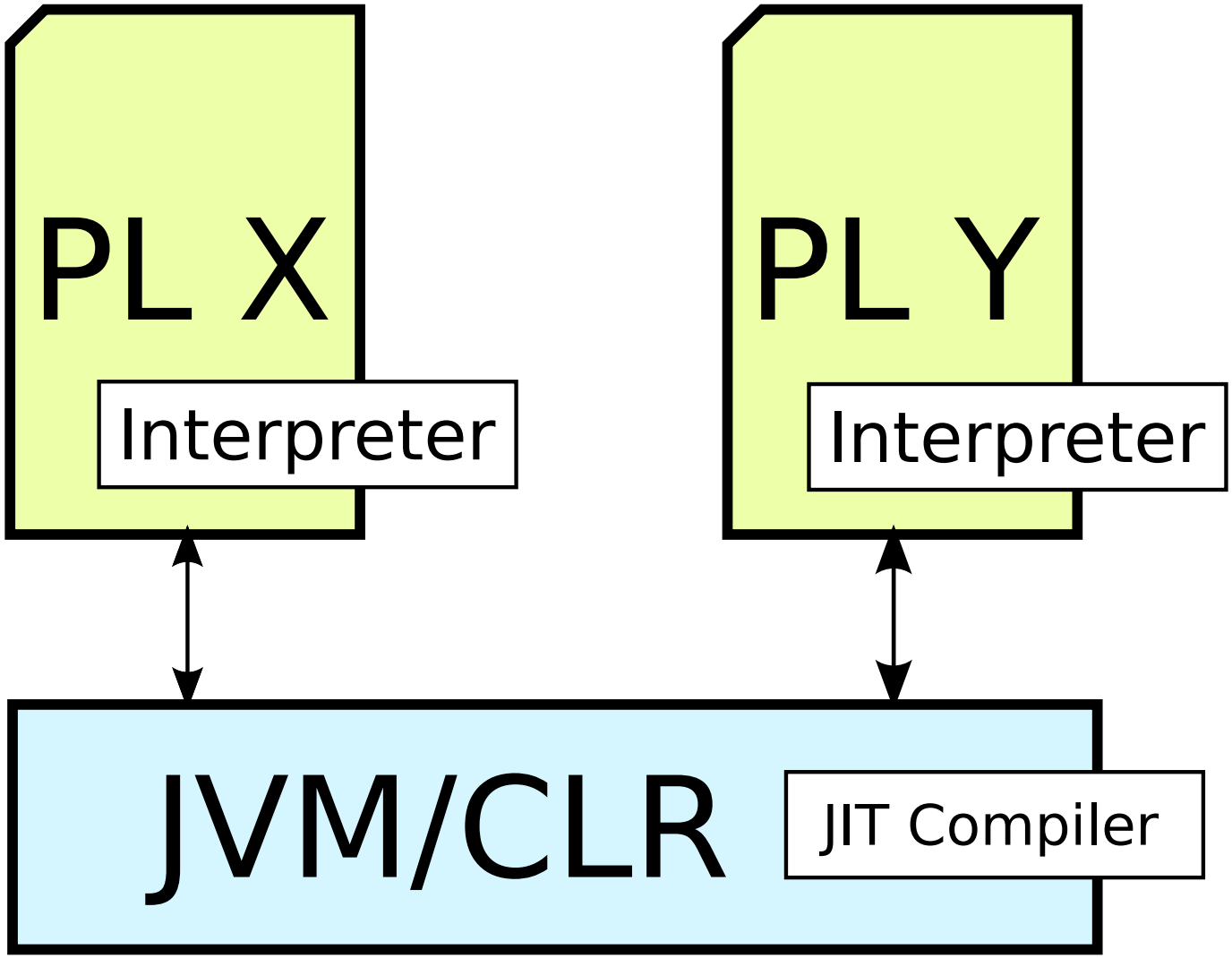
Runtime composition



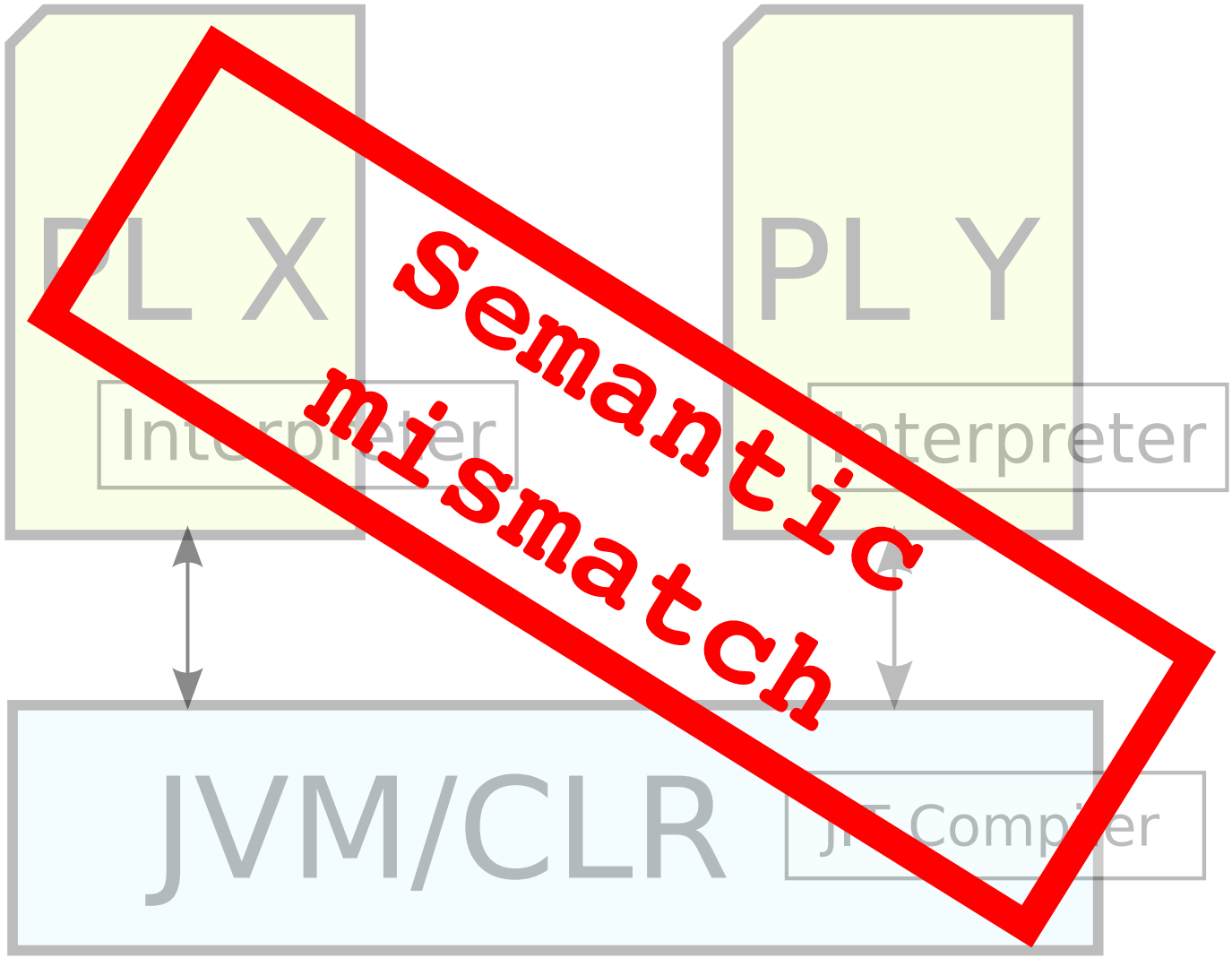
Runtime composition



Runtime composition

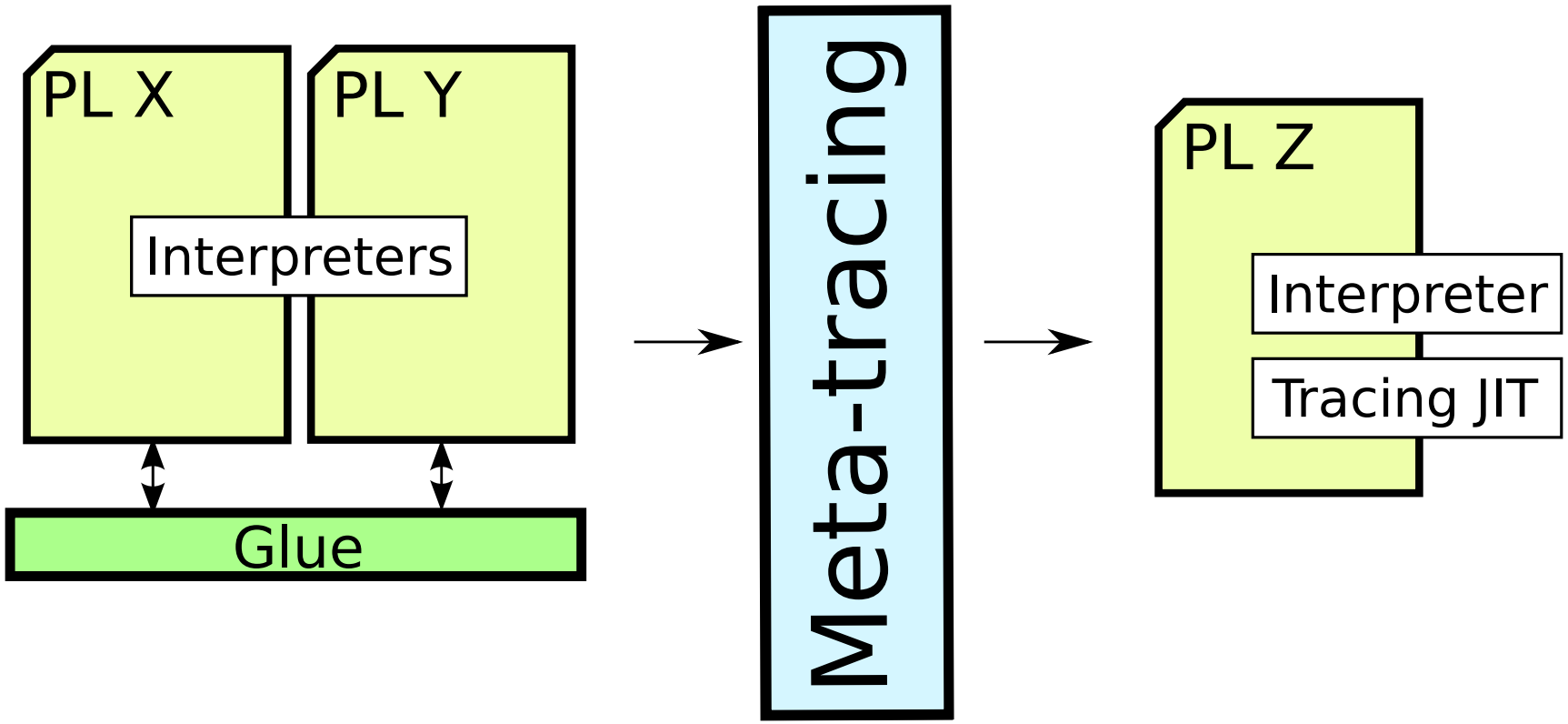


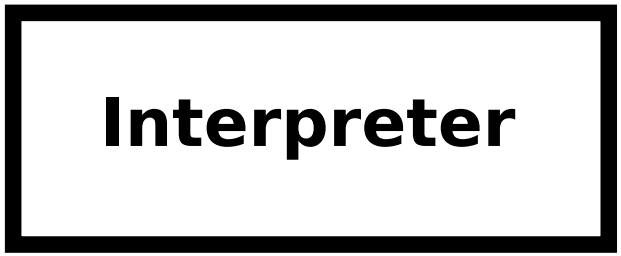
Runtime composition



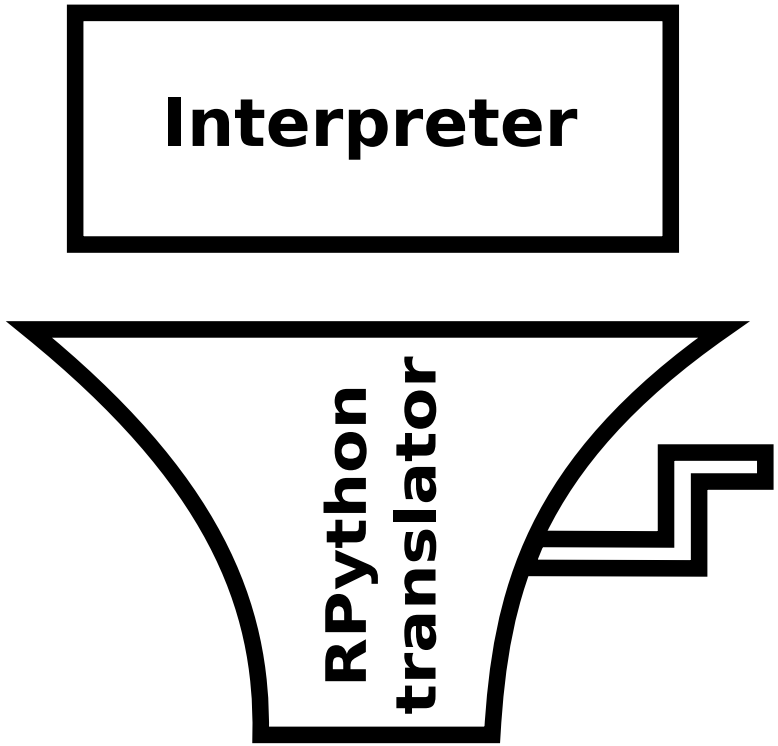
Runtime composition

Runtime composition

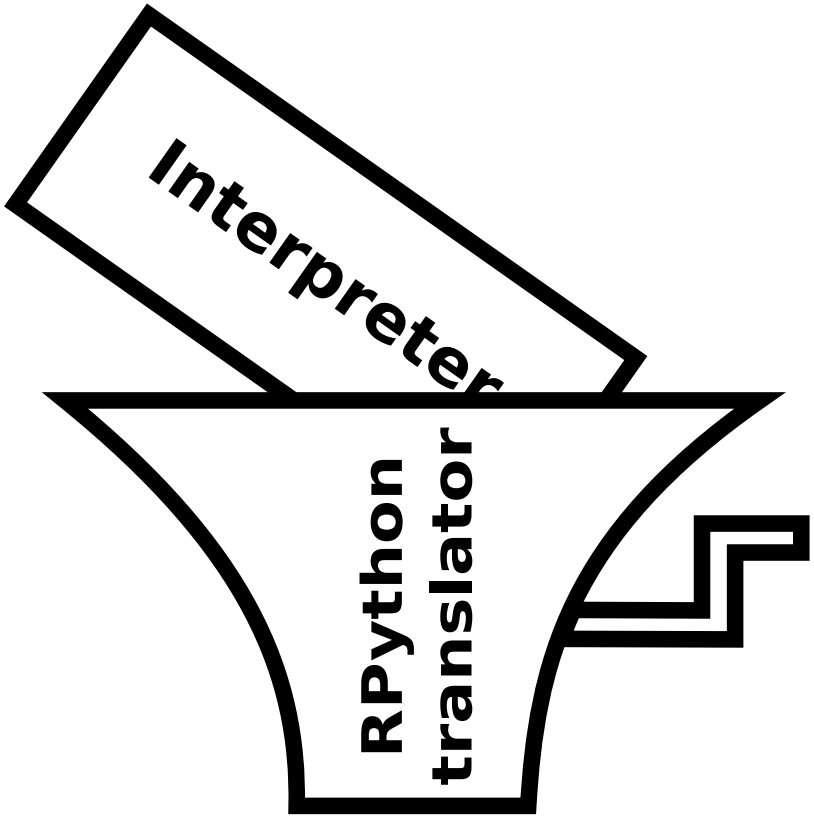



Interpreter

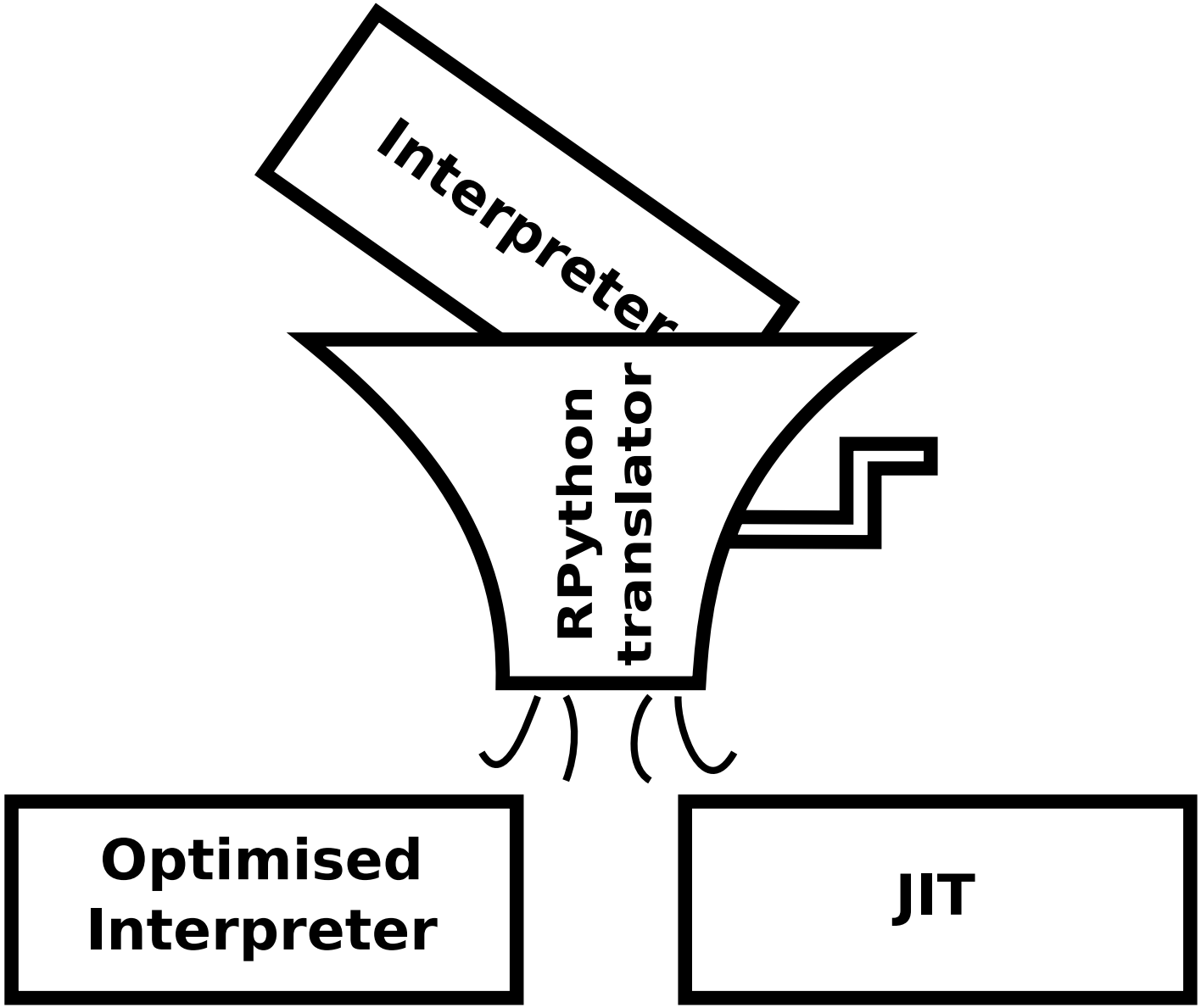
Meta-tracing translation with RPython



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 ...:
        ...
```

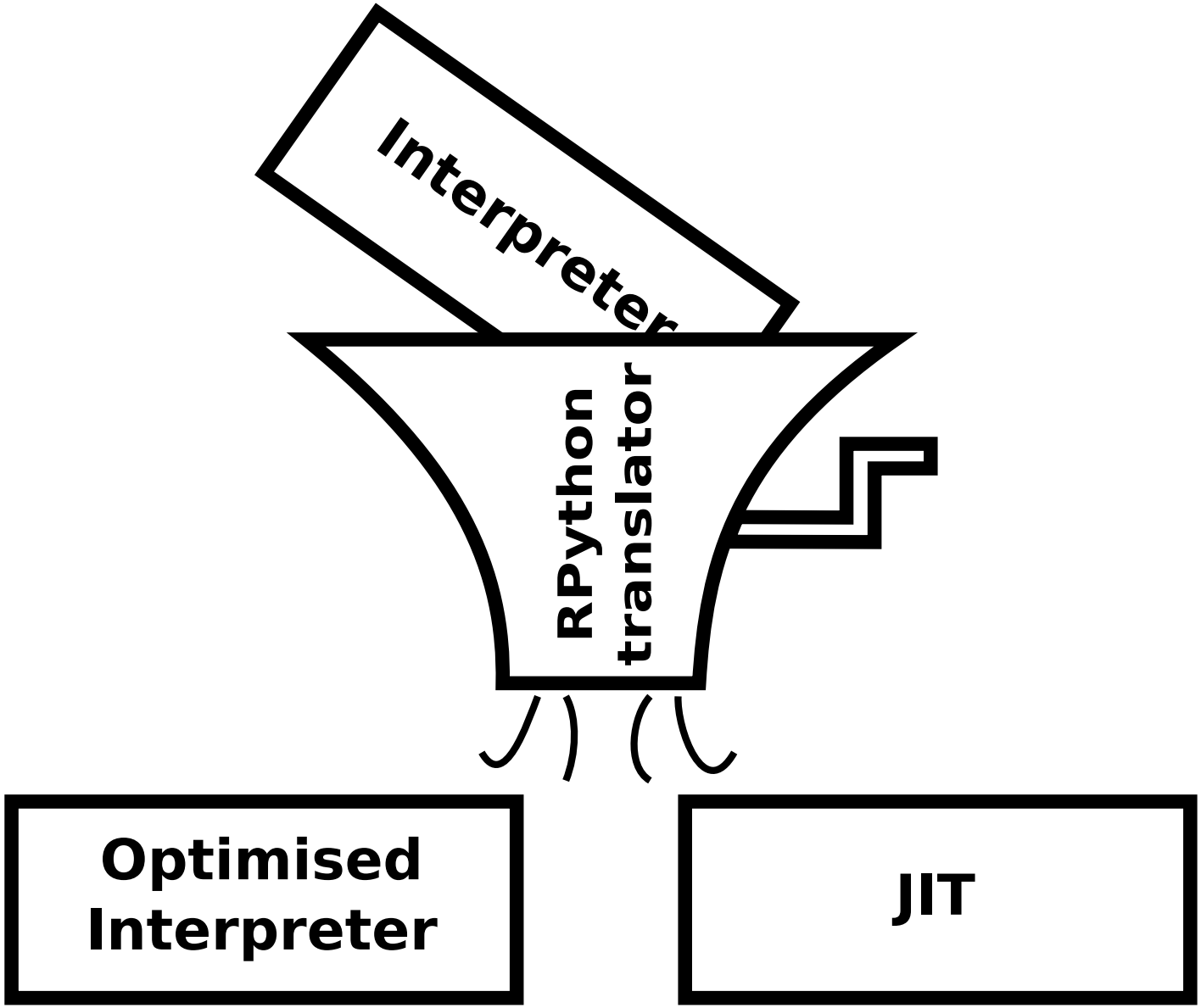
Observation: interpreters are big loops.

Adding a JIT to an RPython interpreter

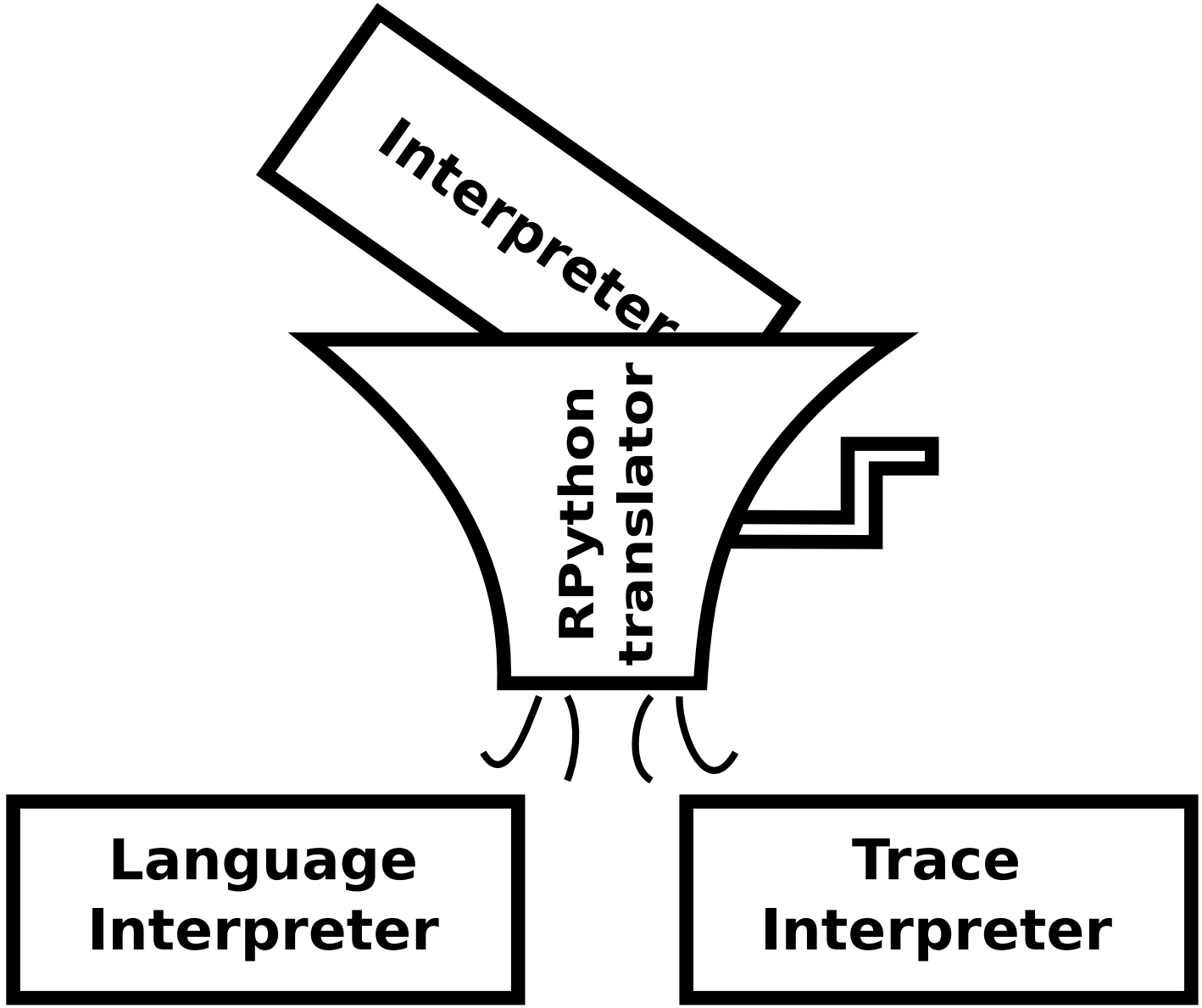
```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

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

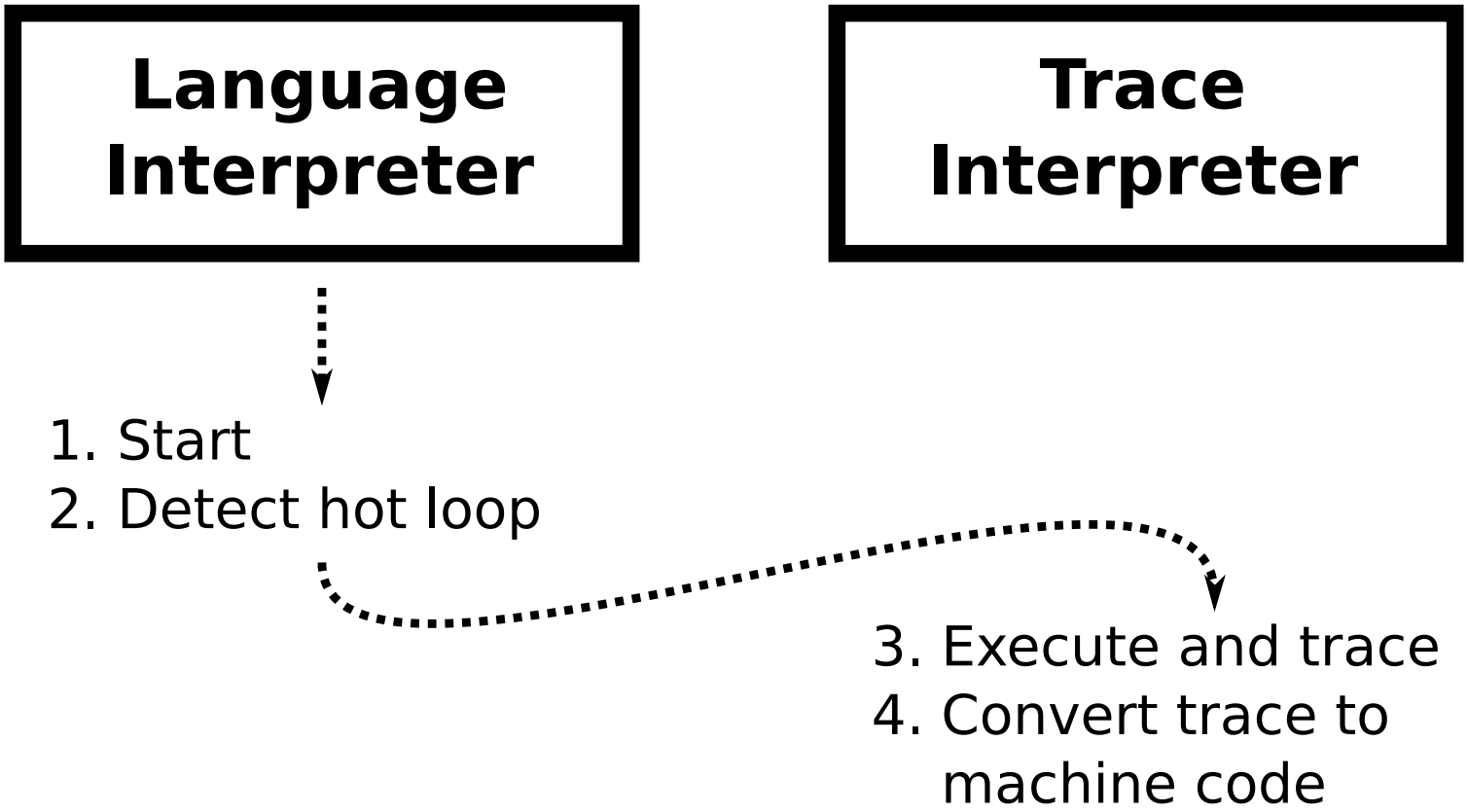
`if x < 0:`
`  x = x + 1`
`else:`
`  x = x + 2`
`x = x + 3`
	`guard_type(x, int)`
	`guard_not_less_than(x, 0)`
	`guard_type(x, int)`
	`x = int_add(x, 2)`
	`guard_type(x, int)`
	`x = int_add(x, 3)`

Tracing JITs

User program (lang *FL*)	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)
```

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)
```

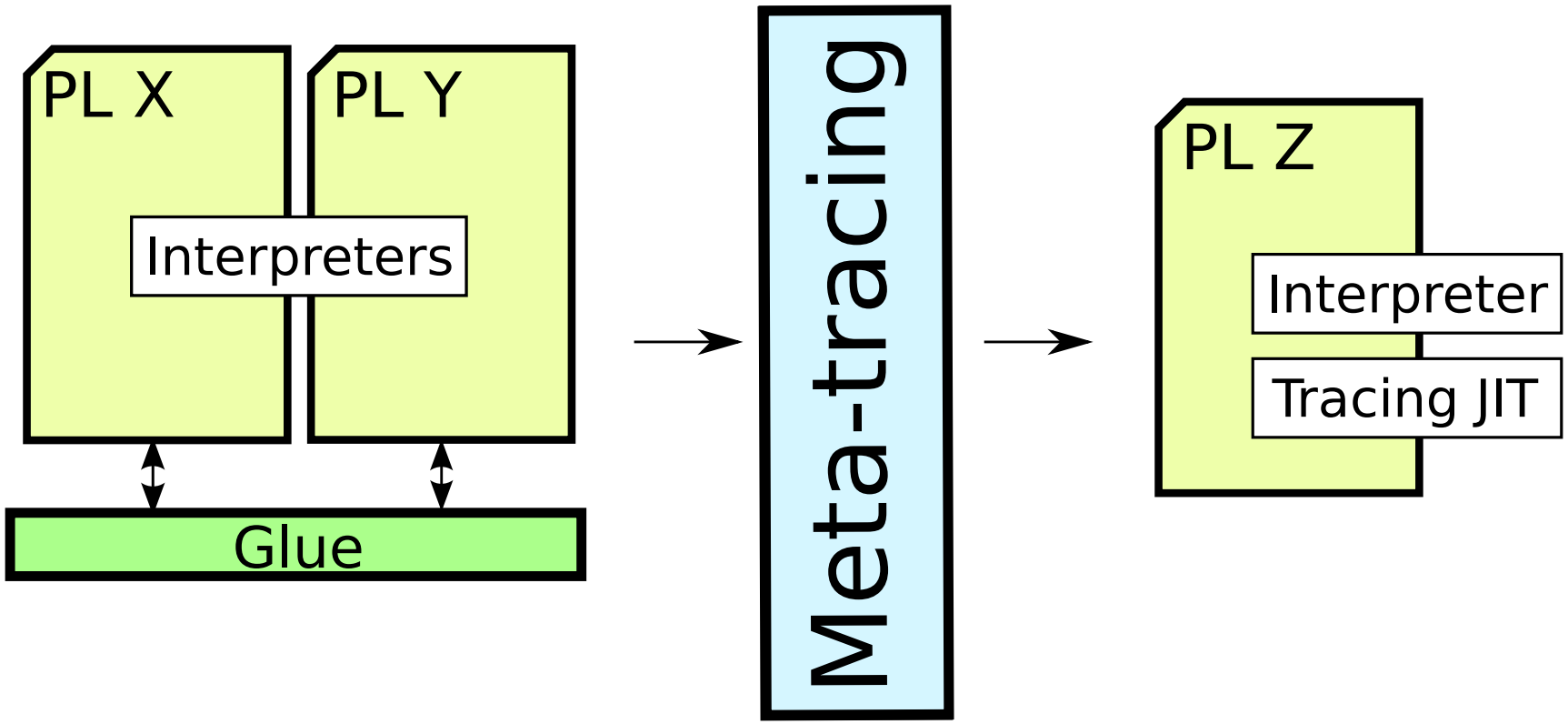
Arithmetic folded trace

```
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



Unipycation demo

Benchmarking VM composition

Warning: draft numbers ahead

Warning: draft numbers ahead

Absolute timing comparison

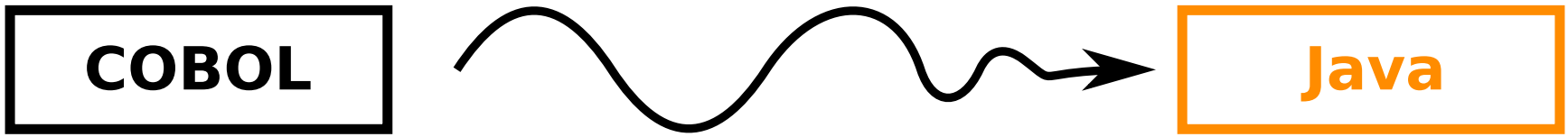
VM	*Benchmark*	*Python*	*Python → Prolog*	*Prolog*
Unipycation	Lists	0.8561	3.8672	1.4409
	Loop1Arg0Result	0.0927	0.1429	0.0967
	Loop1Arg1Result	0.1204	0.2192	0.1282
	NondetLoop1Arg1Result	0.5403	16.8096	0.6566
	SmallFunc	0.0808	73.9304	0.6157
	TermConstruction	6.2775	39.0568	2.4647
Jythog	Lists	6.9740	5665.6390	8110.4420
	Loop1Arg0Result	1.1410	195.7100	196.1970
	Loop1Arg1Result	2.0160	285.3680	277.1590
	NondetLoop1Arg1Result	7.6780	2995.9500	2376.0980
	SmallFunc	9.1240	6980.6260	284.9500
	TermConstruction	670.9370	10419.3540	9209.6390
Swithon	Lists	10.0579	2364.8683	25.0133
	Loop1Arg0Result	3.3848	13.2199	7.3749
	Loop1Arg1Result	5.0255	26.6831	18.8551
	NondetLoop1Arg1Result	8.3630	964.1763	18.5926
	SmallFunc	15.5103	N/A	25.1020
	TermConstruction	282.3256	4952.8222	48.8246
Swithon/pypy	Lists	0.8562	179.7972	24.9756
	Loop1Arg0Result	0.0884	9.3383	7.3305
	Loop1Arg1Result	0.1168	20.5607	18.9826
	NondetLoop1Arg1Result	0.4848	92.9537	19.0223
	SmallFunc	0.0809	2330.3041	25.6858
	TermConstruction	6.8538	438.9820	48.6694

Relative timing comparison

VM	*Benchmark*	*Python*	*Python → Prolog*	*Prolog*
Unipycation	Lists	1.0	4.5174	1.6831
	Loop1Arg0Result	1.0	1.5414	1.0430
	Loop1Arg1Result	1.0	1.8205	1.0643
	NondetLoop1Arg1Result	1.0	31.1097	1.2153
	SmallFunc	1.0	915.4442	7.6240
	TermConstruction	1.0	6.2218	0.3926
Jythog	Lists	1.0	812.3945	1162.9541
	Loop1Arg0Result	1.0	171.5250	171.9518
	Loop1Arg1Result	1.0	141.5516	137.4797
	NondetLoop1Arg1Result	1.0	390.1993	309.4684
	SmallFunc	1.0	765.0839	31.2308
	TermConstruction	1.0	15.5296	13.7265
Swithon	Lists	1.0	235.1261	2.4869
	Loop1Arg0Result	1.0	3.9056	2.1788
	Loop1Arg1Result	1.0	5.3096	3.7519
	NondetLoop1Arg1Result	1.0	115.2901	2.2232
	SmallFunc	1.0	N/A	1.6184
	TermConstruction	1.0	17.5429	0.1729
Swithon/pypy	Lists	1.0	209.9968	29.1706
	Loop1Arg0Result	1.0	105.5859	82.8838
	Loop1Arg1Result	1.0	175.9643	162.4585
	NondetLoop1Arg1Result	1.0	191.7370	39.2376
	SmallFunc	1.0	28789.8303	317.3359
	TermConstruction	1.0	64.0490	7.1010

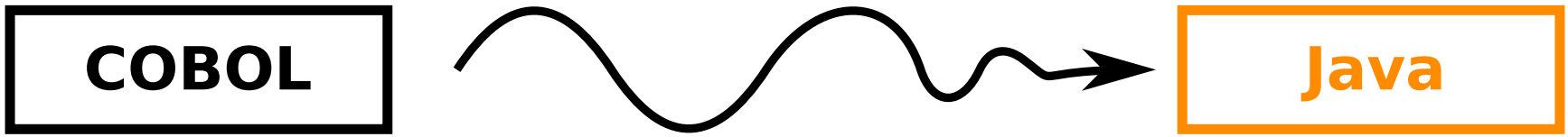
What can we use this for?

What can we use this for?

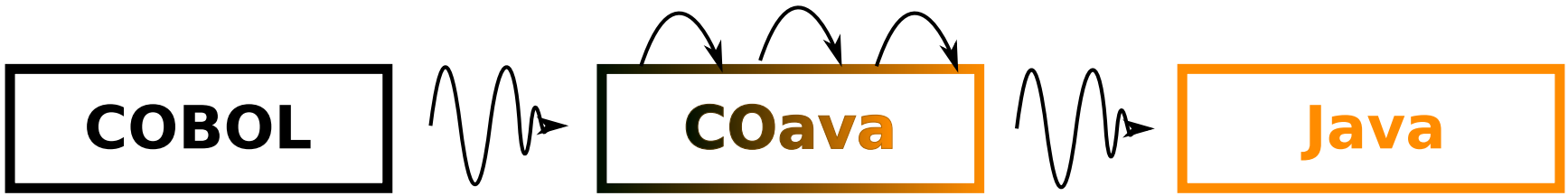


What can we use this for?

"Big Bang" translation

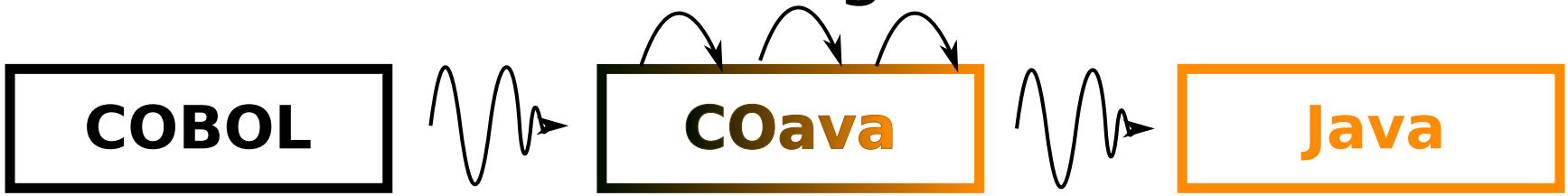


What can we use this for?



What can we use this for?

Gradual migration



What's next?

Editor: incremental semantic analysis on ASTs and code generation

What's next?

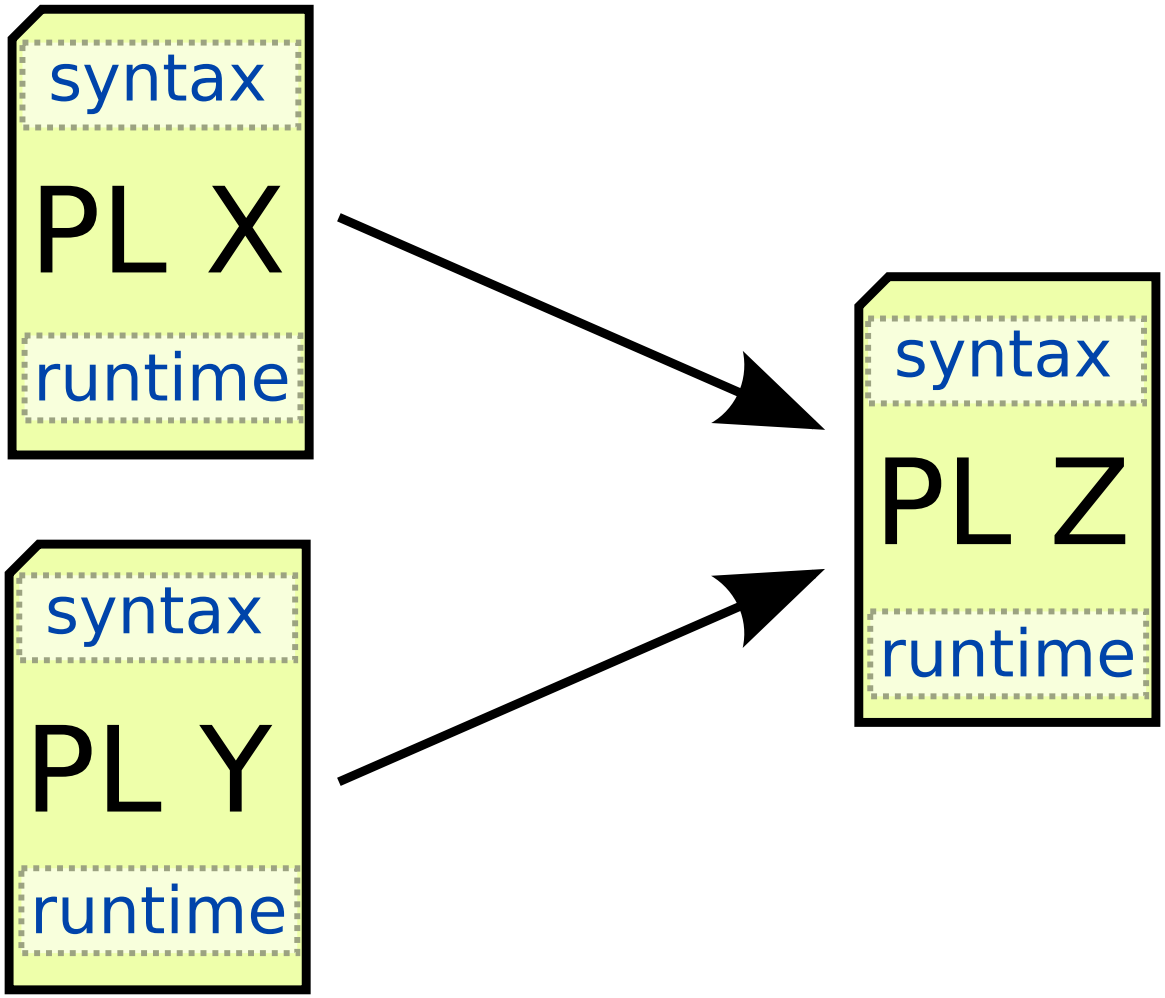
Editor: incremental semantic analysis on ASTs and code generation

+

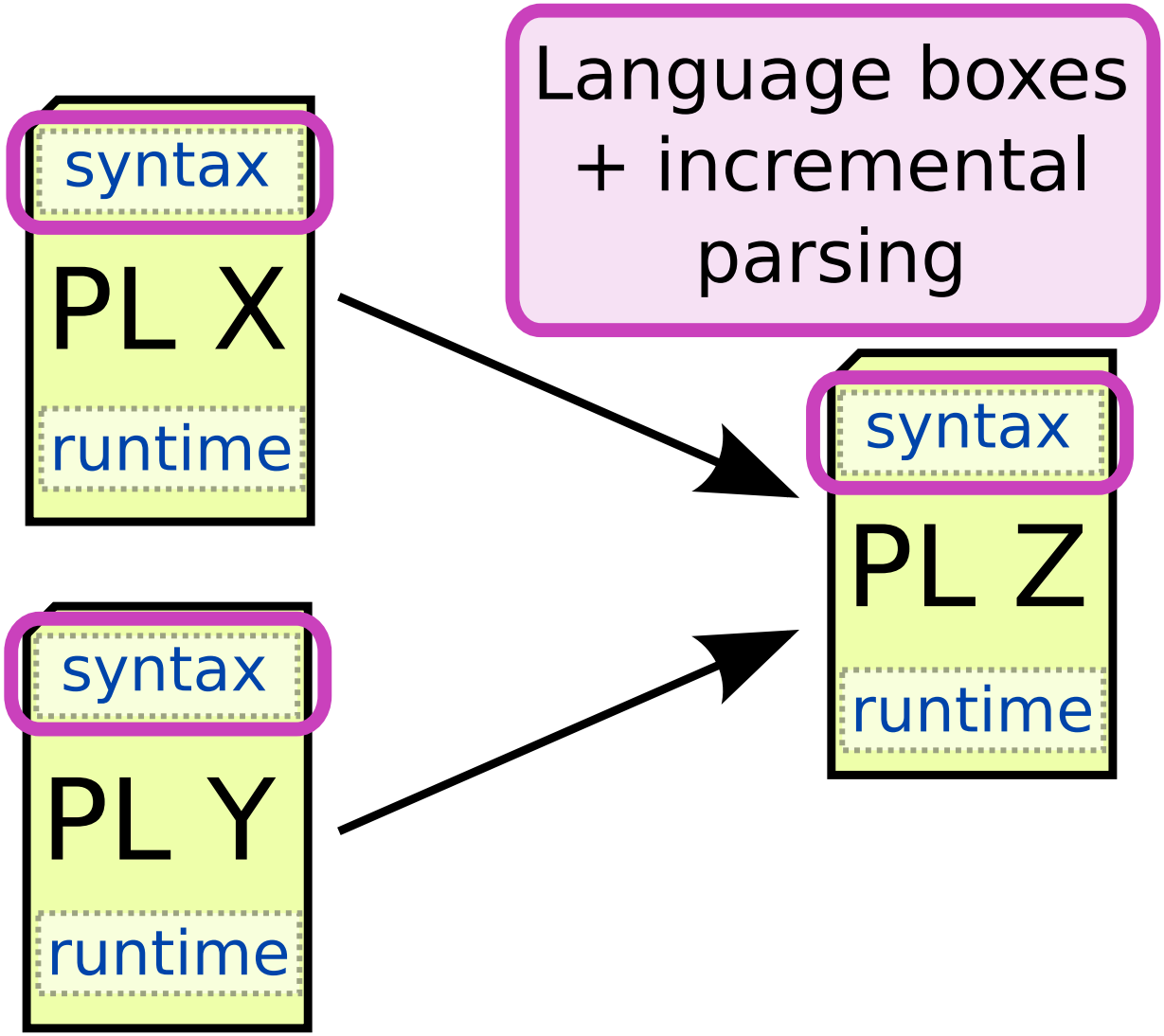
VMs: uncovering common idioms

Summary

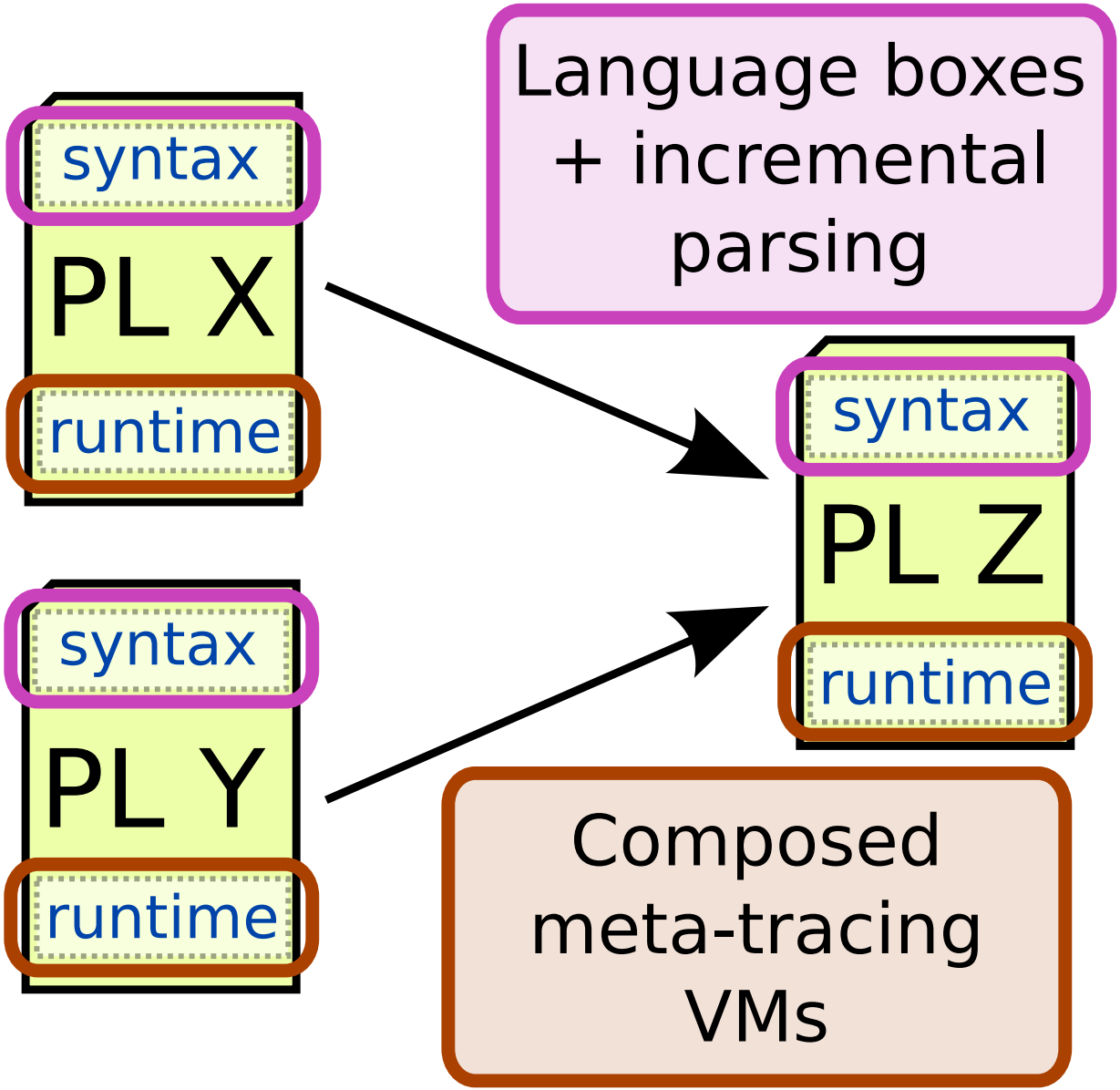
Summary



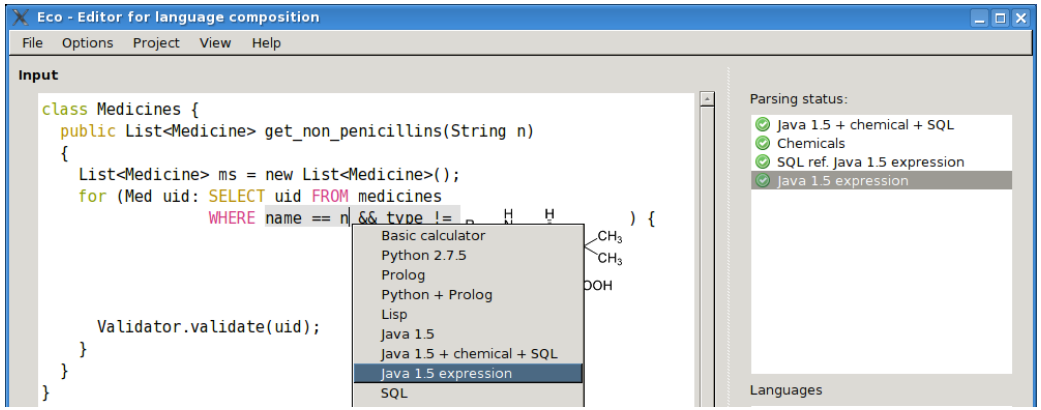
Summary

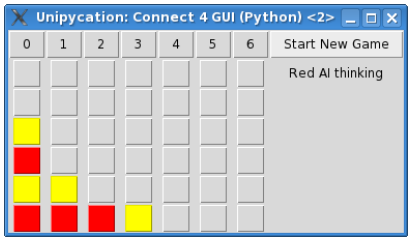

Language boxes
+ incremental
parsing
syntax
PL X
runtime
syntax
PL Y
runtime
syntax
PL Z
runtime

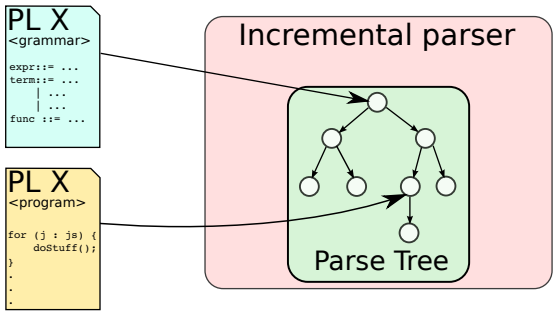
Summary

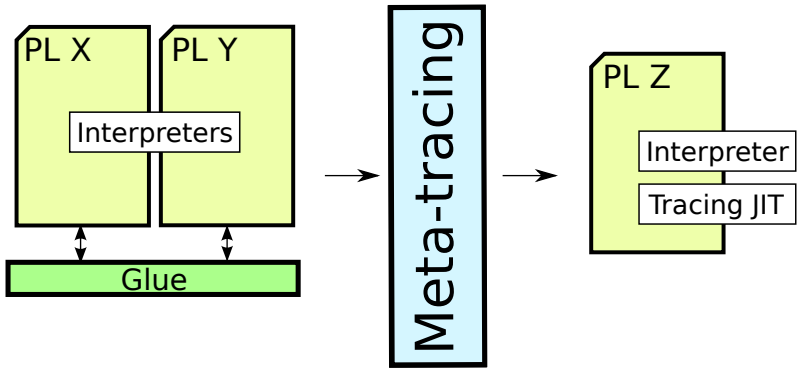

Language boxes
+ incremental
parsing
syntax
PL X
runtime
syntax
PL Y
runtime
syntax
PL Z
runtime
Composed
meta-tracing
VMs

Thanks for listening









http://soft-dev.org/