Fast enough VMs in fast enough time

Laurence Tratt

Software Development Team

KING'S College LONDON

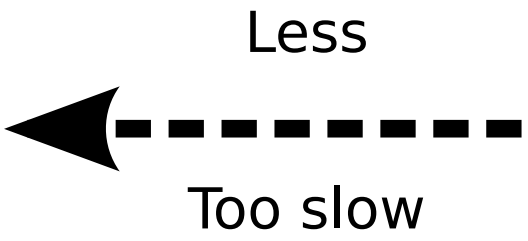
2012-08-20

Language designers dilemma

Language designers dilemma

Implementation
Effort

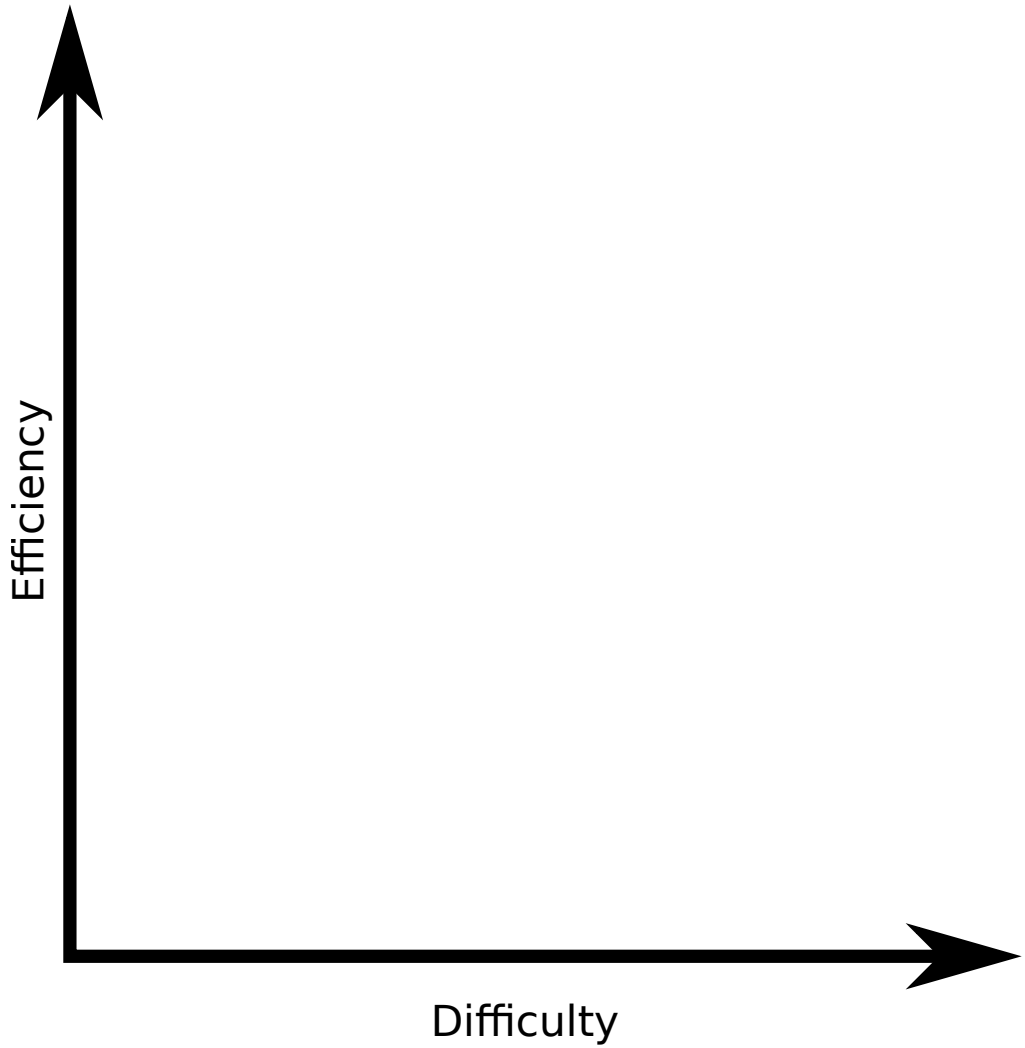
Language designers dilemma



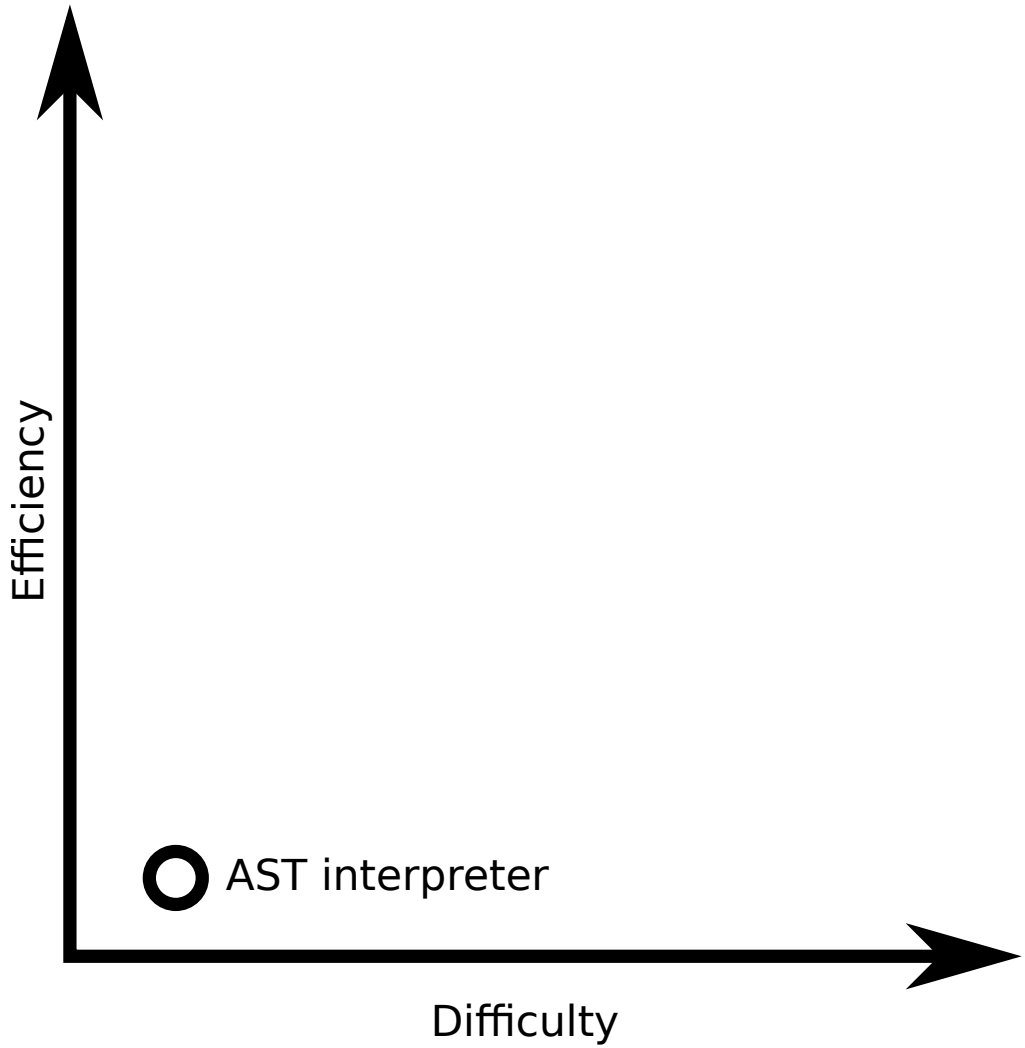
Implementation Effort

Language designers dilemma

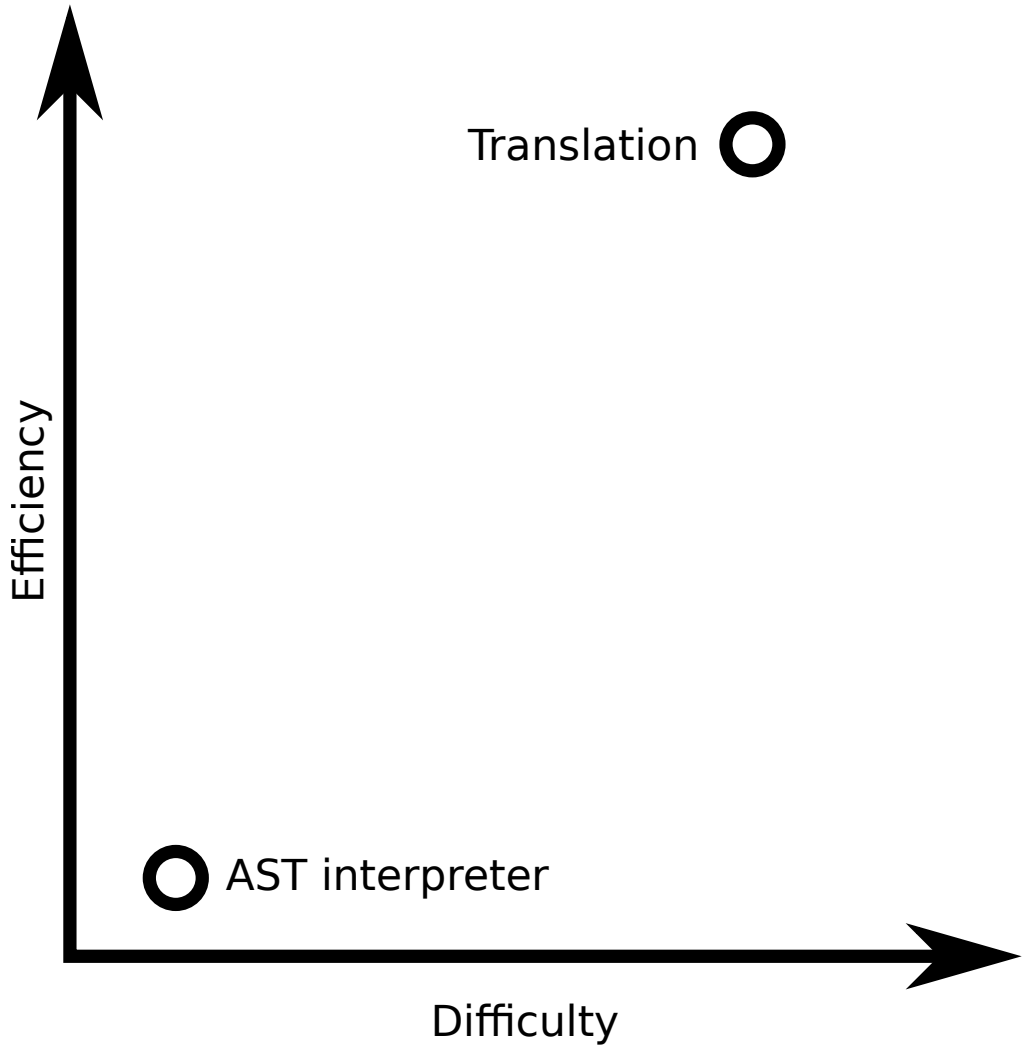
The traditional routes



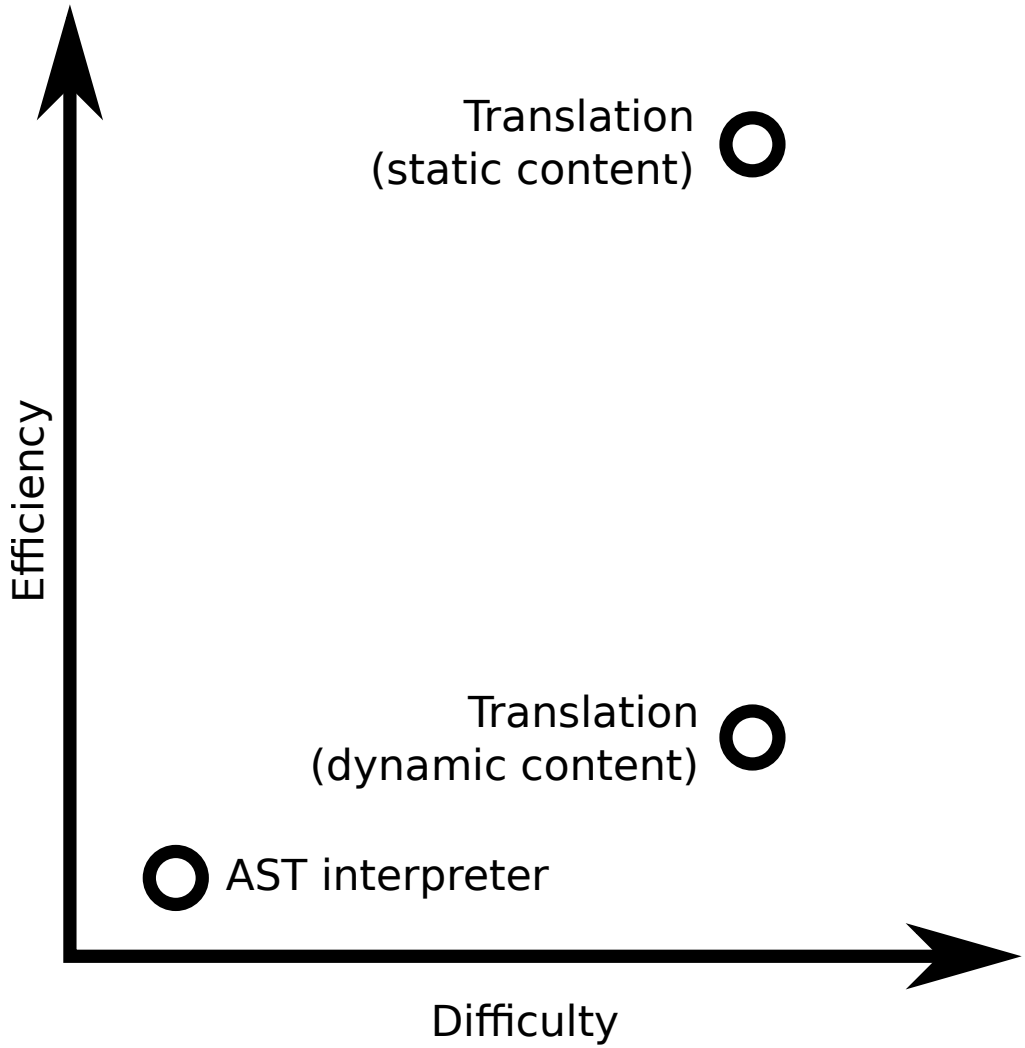
The traditional routes



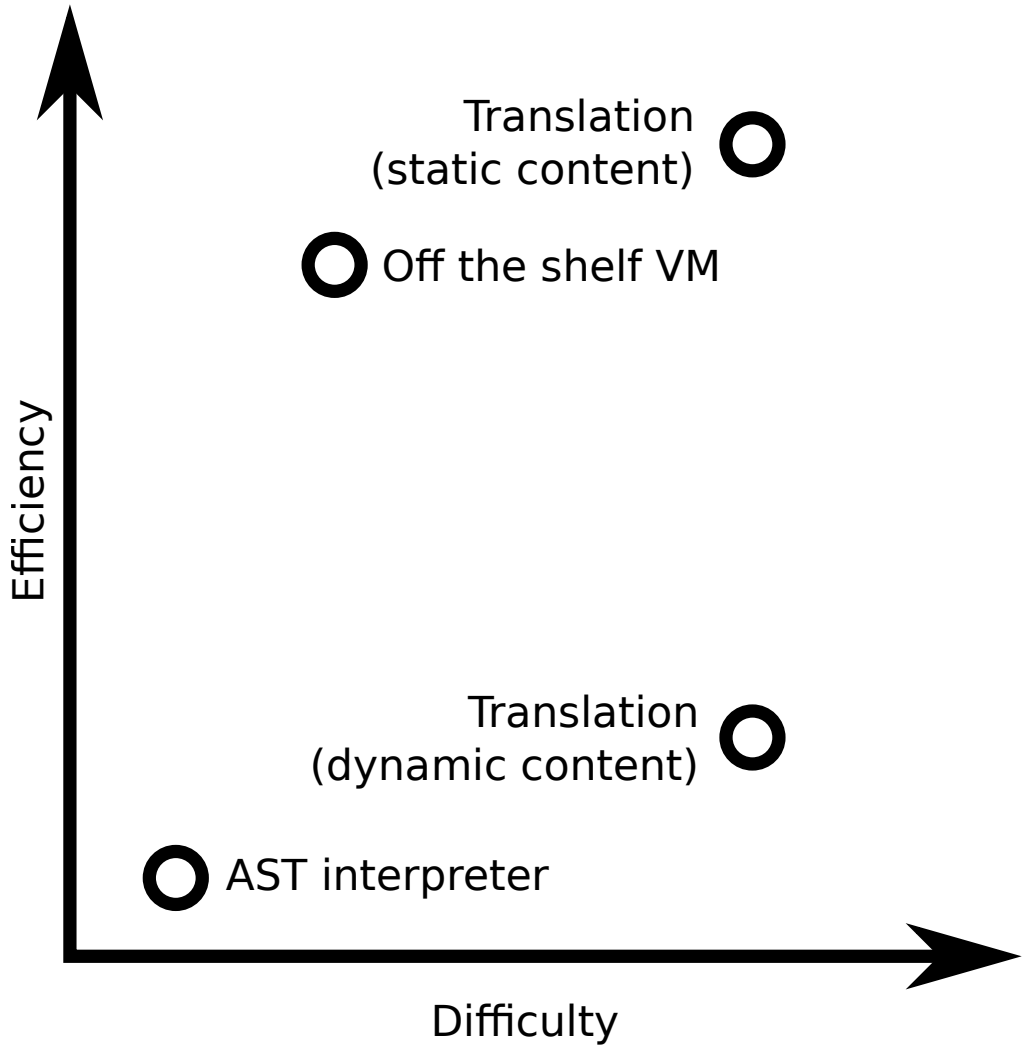
The traditional routes



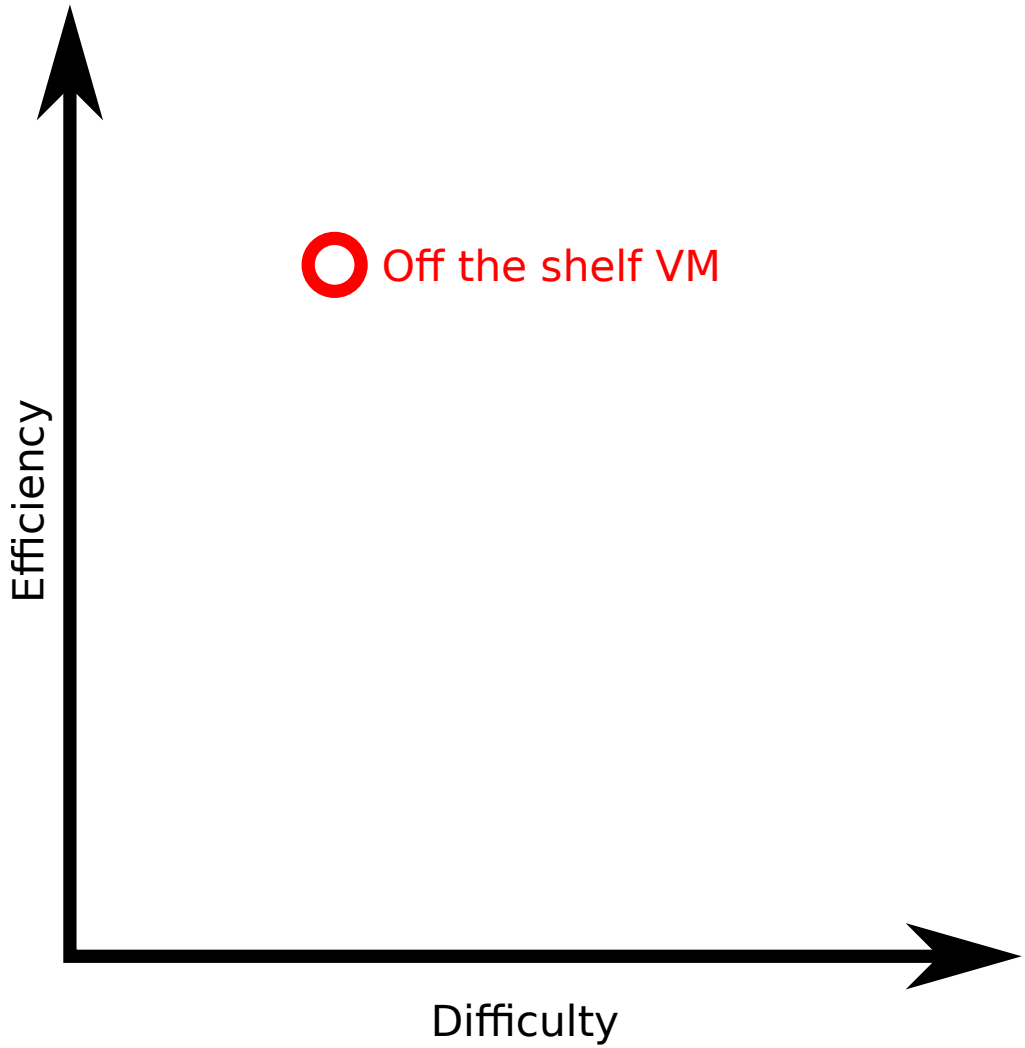
The traditional routes



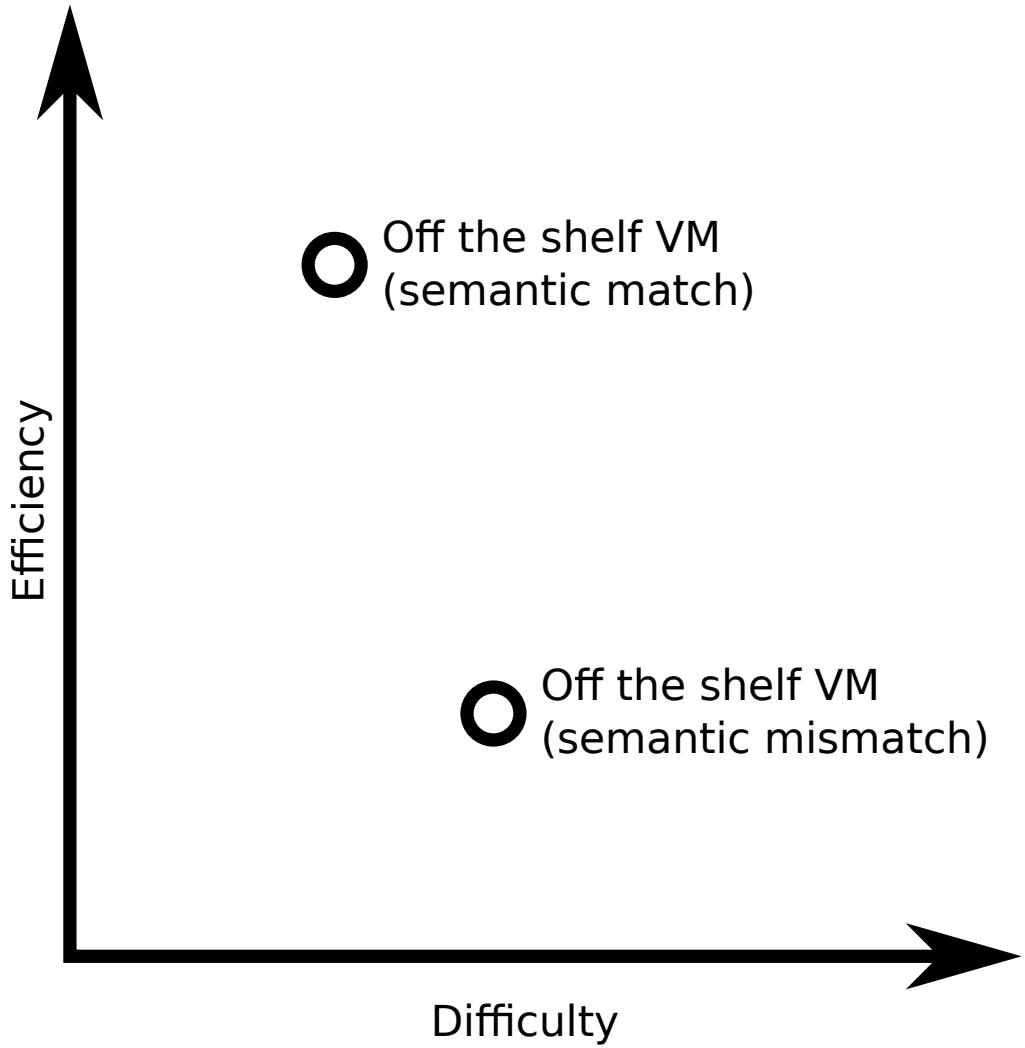
The traditional routes



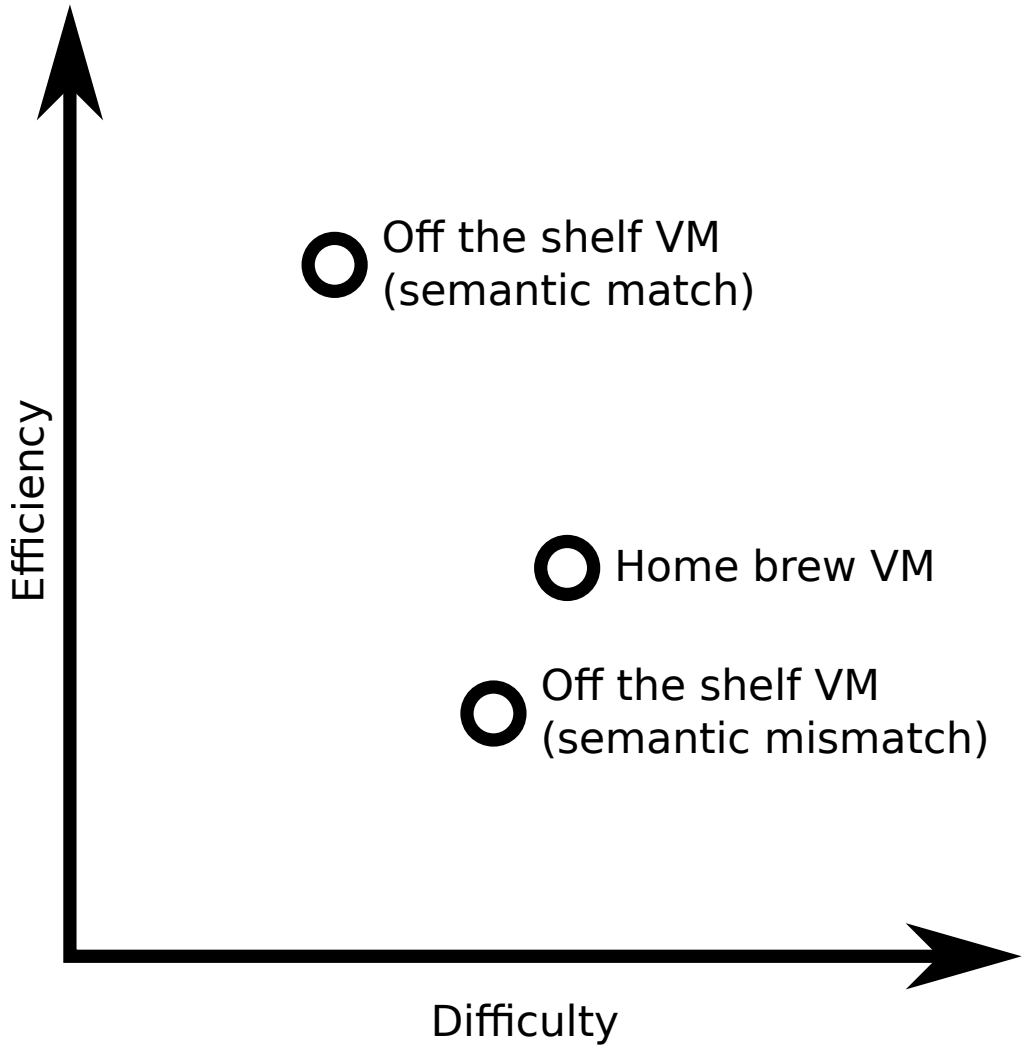
The traditional routes



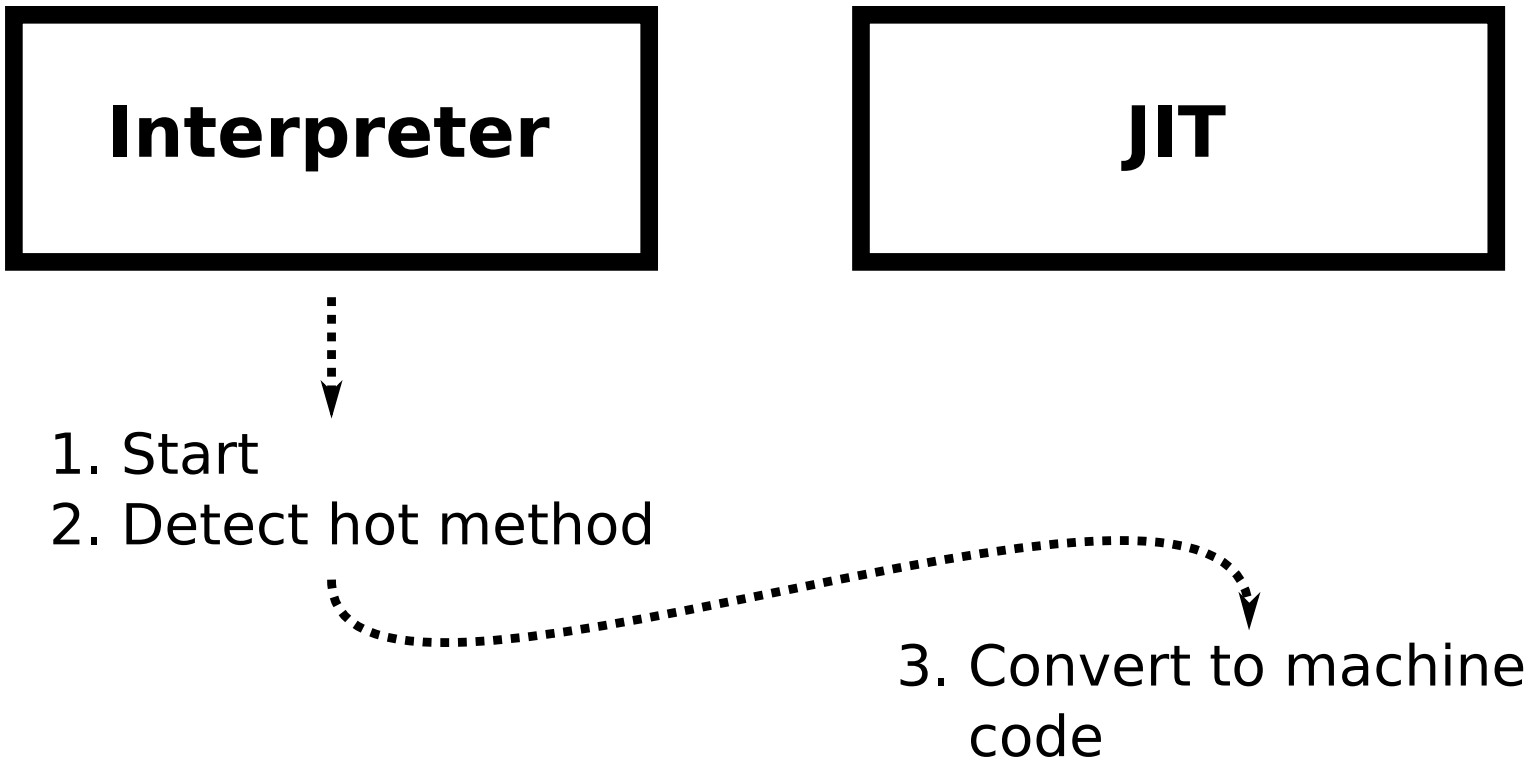
The traditional routes



The traditional routes



JIT VM components



JIT VM issues

Interpreter

JIT

JIT VM issues

Interpreter

+ Easy to write
- Slow

JIT

JIT VM issues

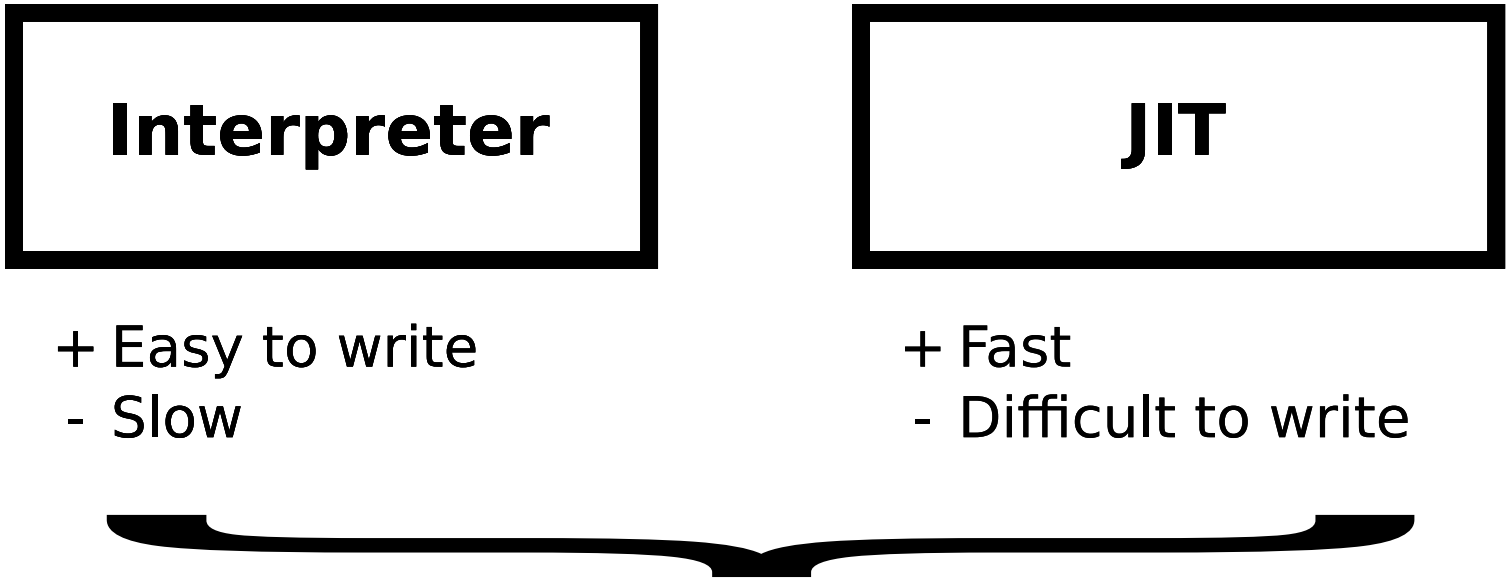
Interpreter

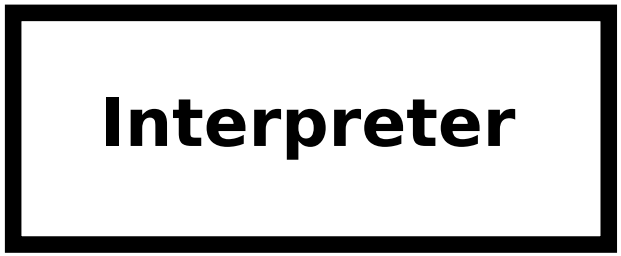
+ Easy to write
- Slow

JIT

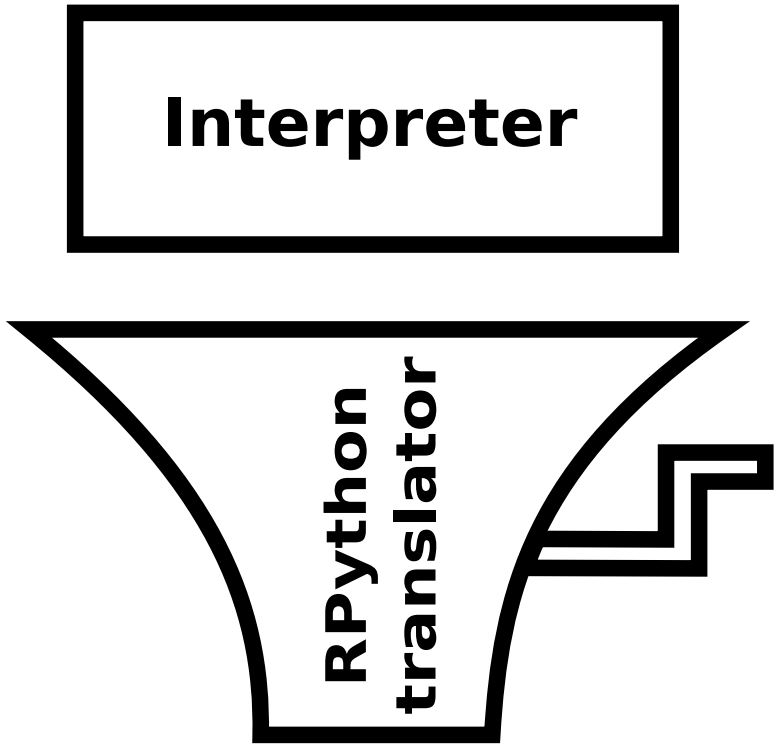
+ Fast
- Difficult to write

JIT VM issues

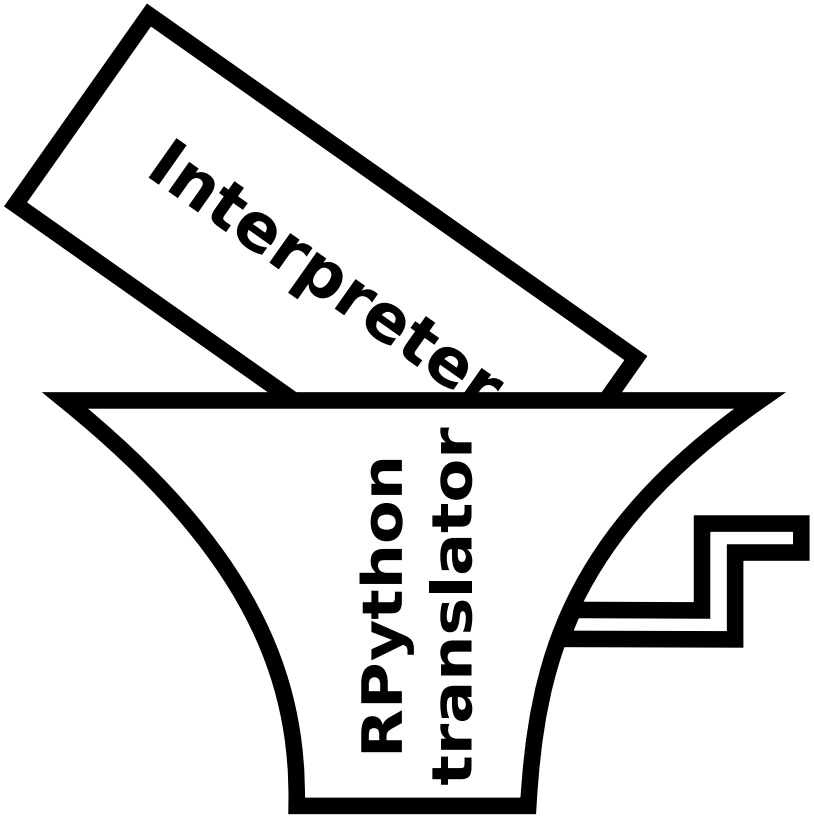



Interpreter

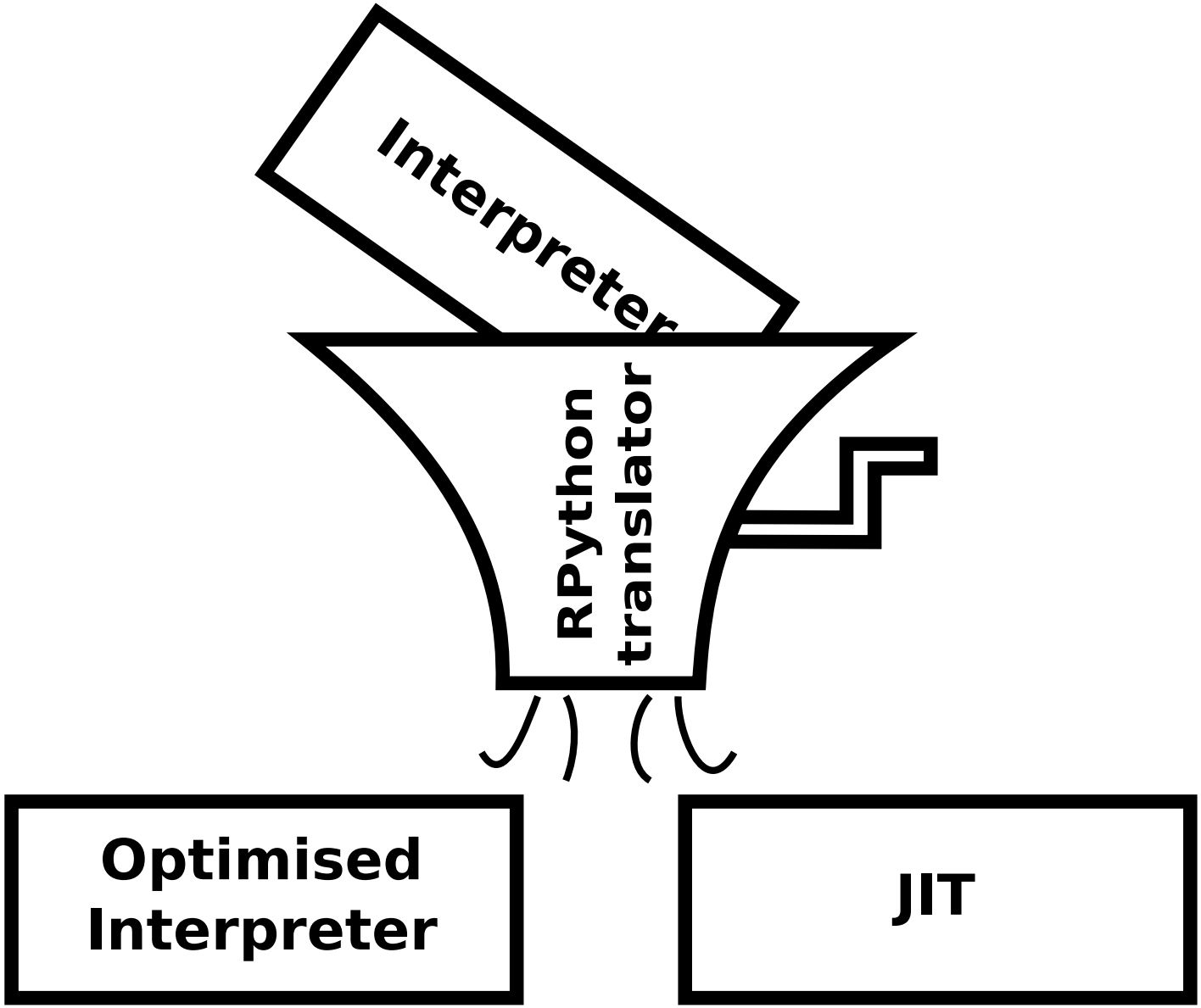
Meta-tracing translation with RPython



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 ...:
        ...
```

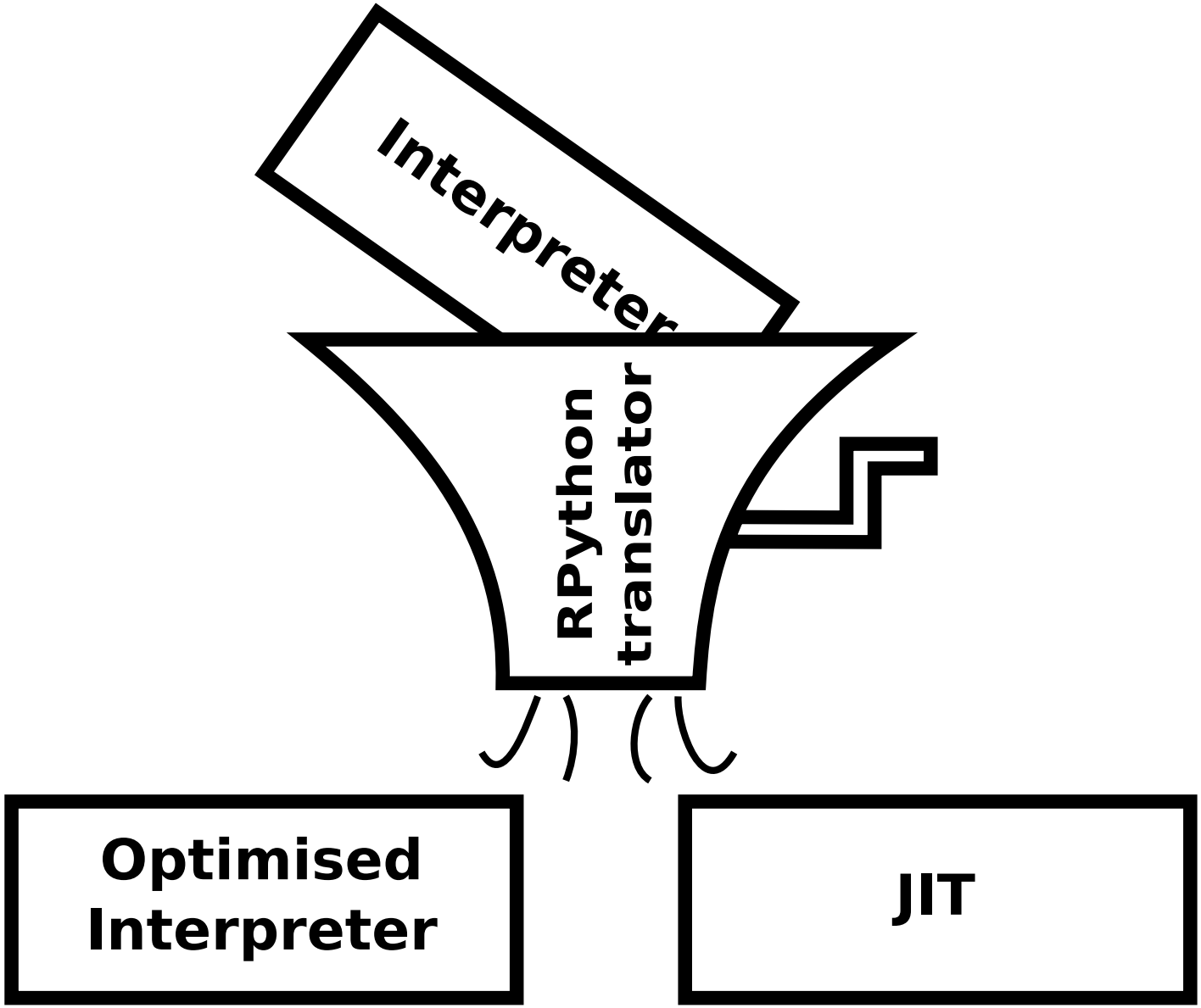
Observation: interpreters are big loops.

Adding a JIT to an RPython interpreter

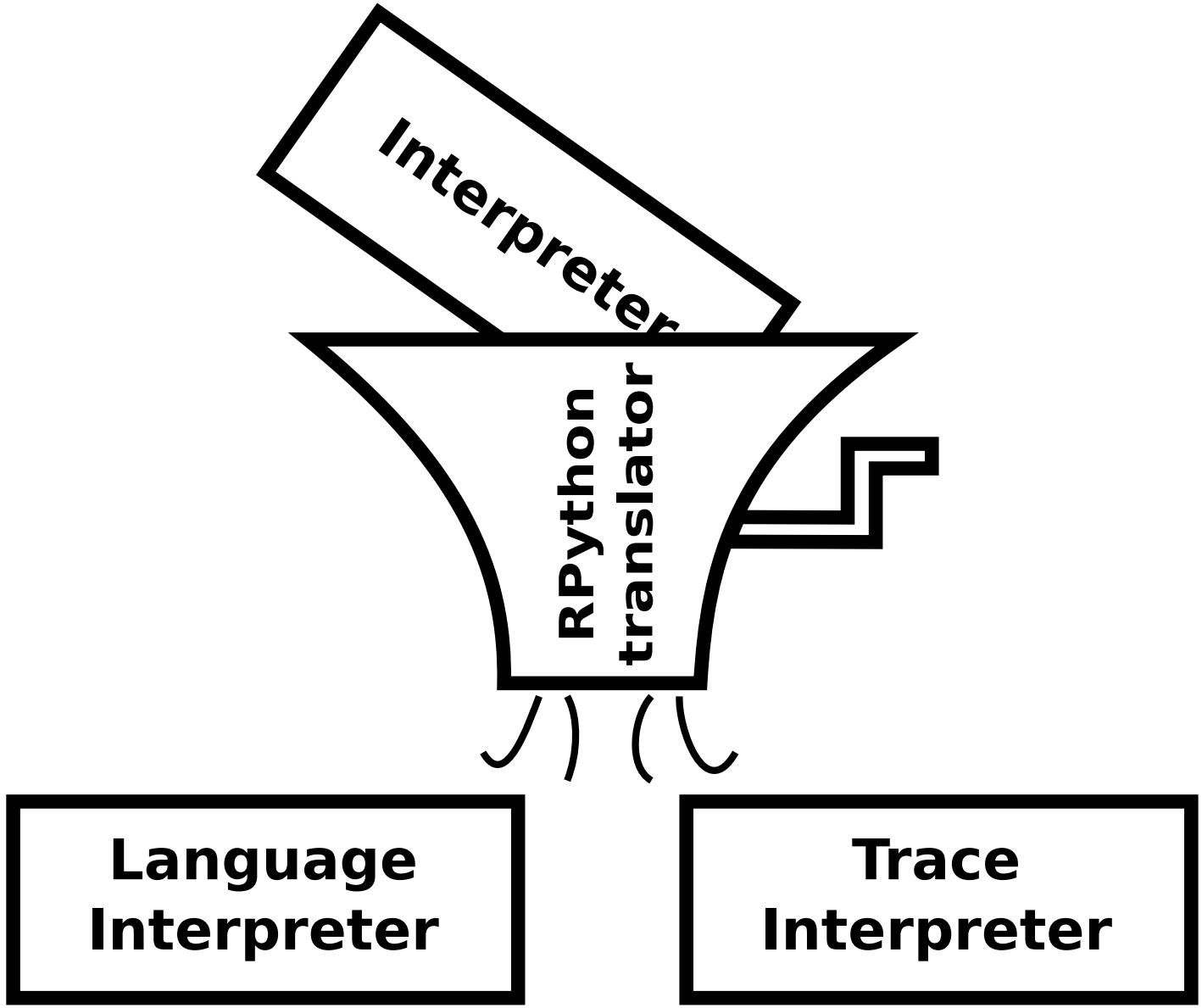
```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

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

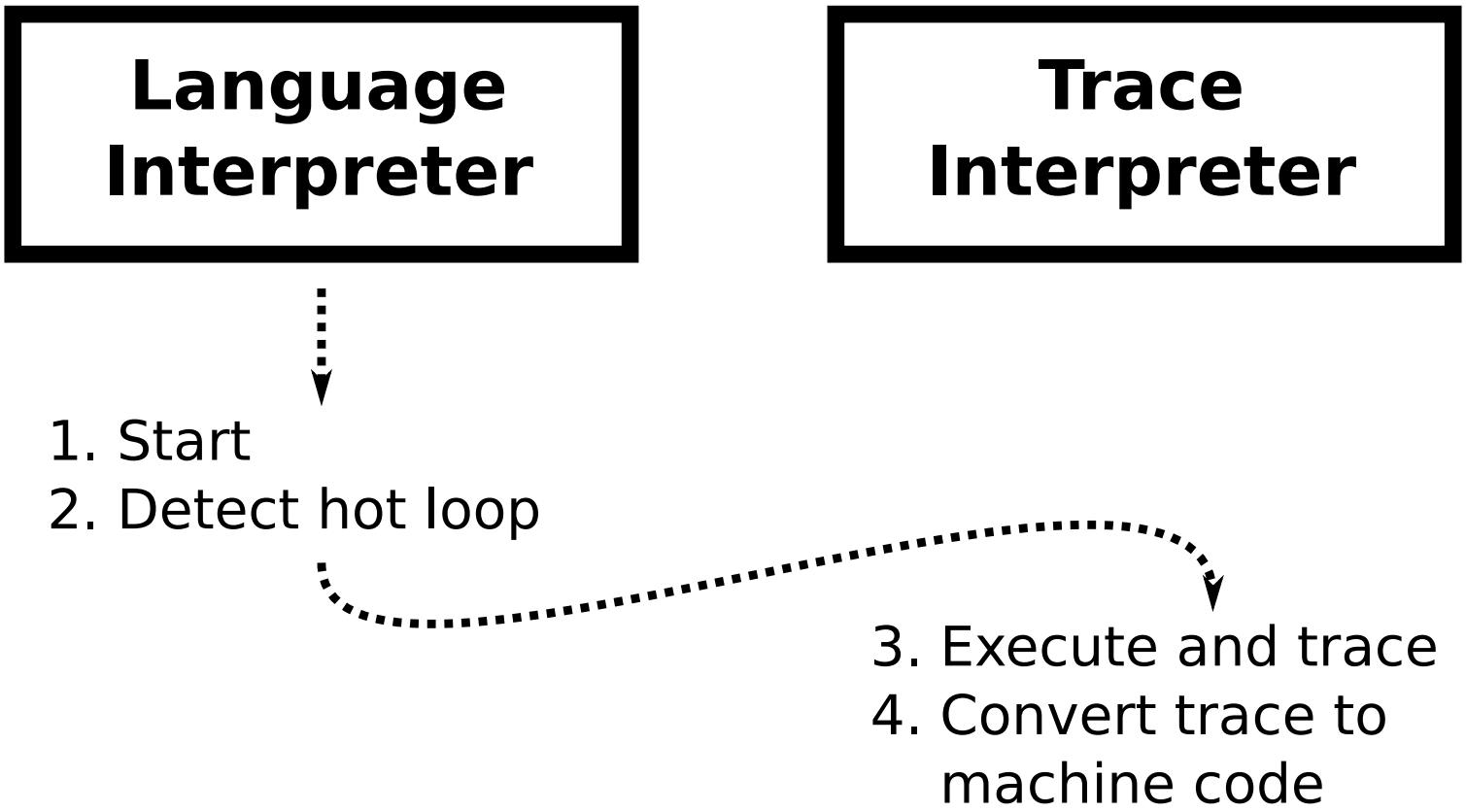
`if x < 0:`
`  x = x + 1`
`else:`
`  x = x + 2`
`x = x + 3`
	`guard_type(x, int)`
	`guard_not_less_than(x, 0)`
	`guard_type(x, int)`
	`x = int_add(x, 2)`
	`guard_type(x, int)`
	`x = int_add(x, 3)`

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.

An RPython experiment

An RPython experiment

A Converge VM in RPython.

An RPython experiment

A Converge VM in RPython.

How hard can it be?

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)		
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance	x	

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance	x	2-150x

Dhrystone benchmark

	50000	5000000
GCC (4.7.2)	0.003 ± 0.002	0.135 ± 0.008
HotSpot (1.7.0_09)	0.106 ± 0.008	0.250 ± 0.010
Converge1 (git #68c795d2)	1.892 ± 0.041	189.050 ± 4.359
Converge2 (git #52bc61a3)	0.121 ± 0.002	2.641 ± 0.067
V8 (3.20.15)	0.030 ± 0.006	1.202 ± 0.078
Lua (5.2.2)	0.192 ± 0.008	18.712 ± 0.261
LuaJIT (2.0.2)	0.014 ± 0.006	0.783 ± 0.022
CPython (2.7.5)	0.362 ± 0.008	35.090 ± 0.421
Jython (2.5.3)	1.881 ± 0.029	29.881 ± 0.504
PyPy–nonopt (2.1)	0.152 ± 0.008	8.741 ± 0.257
PyPy (2.1)	0.073 ± 0.006	1.558 ± 0.065
Ruby (2.0.0-p247)	0.270 ± 0.010	25.050 ± 0.529
JRuby (1.7.4)	2.251 ± 0.029	11.844 ± 0.343
Topaz (nightly)	0.301 ± 0.006	4.739 ± 0.084

Dhrystone benchmark

	50000	5000000
GCC (4.7.2)	0.003 ± 0.002	0.135 ± 0.008
HotSpot (1.7.0_09)	0.106 ± 0.008	0.250 ± 0.010
Converge1 (git #68c795d2)	**1.892** ± **0.041**	**189.050** ± **4.359**
Converge2 (git #52bc61a3)	**0.121** ± **0.002**	**2.641** ± **0.067**
V8 (3.20.15)	0.030 ± 0.006	1.202 ± 0.078
Lua (5.2.2)	0.192 ± 0.008	18.712 ± 0.261
LuaJIT (2.0.2)	0.014 ± 0.006	0.783 ± 0.022
CPython (2.7.5)	0.362 ± 0.008	35.090 ± 0.421
Jython (2.5.3)	1.881 ± 0.029	29.881 ± 0.504
PyPy–nonopt (2.1)	0.152 ± 0.008	8.741 ± 0.257
PyPy (2.1)	0.073 ± 0.006	1.558 ± 0.065
Ruby (2.0.0-p247)	0.270 ± 0.010	25.050 ± 0.529
JRuby (1.7.4)	2.251 ± 0.029	11.844 ± 0.343
Topaz (nightly)	0.301 ± 0.006	4.739 ± 0.084

Richards benchmark

	10	100
GCC (4.7.2)	0.011 ± 0.008	0.066 ± 0.008
HotSpot (1.7.0_09)	0.120 ± 0.006	0.182 ± 0.008
Converge1 (git #68c795d2)	9.552 ± 0.133	95.956 ± 1.542
Converge2 (git #52bc61a3)	0.662 ± 0.006	3.414 ± 0.047
V8 (3.20.15)	0.021 ± 0.006	0.032 ± 0.008
Lua (5.2.2)	0.637 ± 0.010	6.308 ± 0.098
LuaJIT (2.0.2)	0.080 ± 0.006	0.726 ± 0.029
CPython (2.7.5)	1.584 ± 0.018	15.697 ± 0.218
Jython (2.5.3)	3.035 ± 0.057	15.319 ± 0.280
PyPy–nonopt (2.1)	0.753 ± 0.010	4.326 ± 0.080
PyPy (2.1)	0.275 ± 0.008	0.633 ± 0.022
Ruby (2.0.0-p247)	0.696 ± 0.020	6.229 ± 0.206
JRuby (1.7.4)	2.412 ± 0.024	4.269 ± 0.122
Topaz (nightly)	0.305 ± 0.008	0.698 ± 0.033

Richards benchmark

	10	100
GCC (4.7.2)	0.011 ± 0.008	0.066 ± 0.008
HotSpot (1.7.0_09)	0.120 ± 0.006	0.182 ± 0.008
Converge1 (git #68c795d2)	9.552 ± 0.133	95.956 ± 1.542
Converge2 (git #52bc61a3)	0.662 ± 0.006	3.414 ± 0.047
V8 (3.20.15)	**0.021 ± 0.006**	**0.032 ± 0.008**
Lua (5.2.2)	0.637 ± 0.010	6.308 ± 0.098
LuaJIT (2.0.2)	0.080 ± 0.006	0.726 ± 0.029
CPython (2.7.5)	1.584 ± 0.018	15.697 ± 0.218
Jython (2.5.3)	3.035 ± 0.057	15.319 ± 0.280
PyPy–nonopt (2.1)	0.753 ± 0.010	4.326 ± 0.080
PyPy (2.1)	**0.275 ± 0.008**	**0.633 ± 0.022**
Ruby (2.0.0-p247)	0.696 ± 0.020	6.229 ± 0.206
JRuby (1.7.4)	**2.412 ± 0.024**	**4.269 ± 0.122**
Topaz (nightly)	0.305 ± 0.008	0.698 ± 0.033

Fannkuch-redux benchmark

	10		11	
GCC (4.7.2)	0.166	± 0.006	1.982	± 0.008
HotSpot (1.7.0_09)	0.319	± 0.010	3.149	± 0.022
Converge1 (git #68c795d2)	†		†	
Converge2 (git #52bc61a3)	3.490	± 0.061	44.085	± 0.559
V8 (3.20.15)	0.216	± 0.008	2.563	± 0.049
Lua (5.2.2)	7.483	± 0.071	98.079	± 1.037
LuaJIT (2.0.2)	0.362	± 0.006	4.717	± 0.010
CPython (2.7.5)	9.466	± 0.621	116.777	± 6.934
Jython (2.5.3)	7.487	± 0.051	70.205	± 2.985
PyPy–nonopt (2.1)	1.403	± 0.020	16.953	± 0.478
PyPy (2.1)	1.335	± 0.020	16.329	± 0.598
Ruby (2.0.0-p247)	11.320	± 0.508	147.890	± 6.464
JRuby (1.7.4)	5.211	± 0.159	45.698	± 2.015

Fannkuch-redux benchmark

	10		11	
GCC (4.7.2)	0.166	± 0.006	1.982	± 0.008
HotSpot (1.7.0_09)	0.319	± 0.010	3.149	± 0.022
Converge1 (git #68c795d2)	†		†	
Converge2 (git #52bc61a3)	3.490	± 0.061	44.085	± 0.559
V8 (3.20.15)	0.216	± 0.008	2.563	± 0.049
Lua (5.2.2)	7.483	± 0.071	98.079	± 1.037
LuaJIT (2.0.2)	0.362	± 0.006	4.717	± 0.010
CPython (2.7.5)	9.466	± 0.621	116.777	± 6.934
Jython (2.5.3)	7.487	± 0.051	70.205	± 2.985
PyPy–nonopt (2.1)	1.403	± 0.020	16.953	± 0.478
PyPy (2.1)	1.335	± 0.020	16.329	± 0.598
Ruby (2.0.0-p247)	11.320	± 0.508	147.890	± 6.464
JRuby (1.7.4)	5.211	± 0.159	45.698	± 2.015

Inlining

User program	Trace without inlining	Trace with inlining

`def f(x):`
`  return 2 + g(x) + 3`
 	
`def g(x):`
`  if x < 0:`
`  return 0`
`  else:`
`  return 1`
	`return 2 + g(x) + 3`	`guard_not_less_than(x, 0)`
		`return 2 + 1 + 3`

Optimising an RPython JIT

- Replace resizable lists with arrays.

Optimising an RPython JIT

- Replace resizable lists with arrays.
- *Promote* values.

Optimising an RPython JIT

- Replace resizable lists with arrays.
- *Promote* values.
- *Elide* functions.

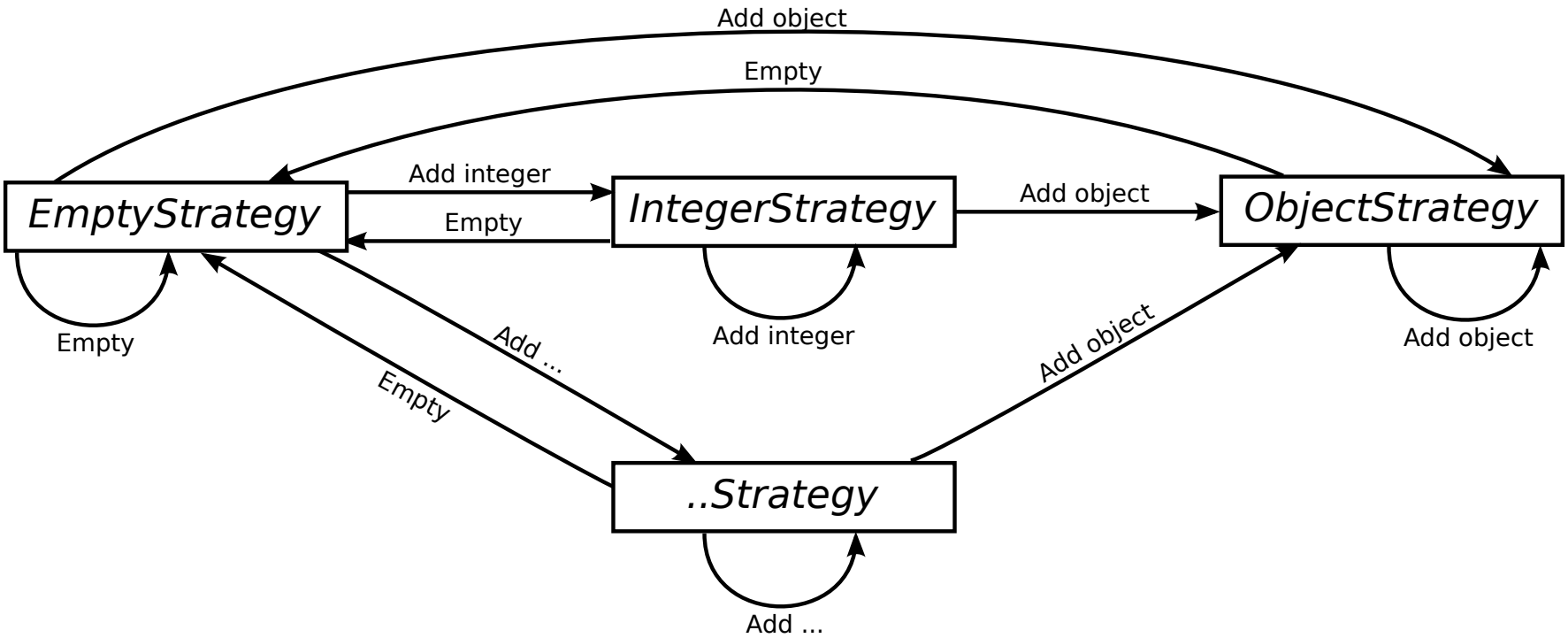
JIT experimentation test bed

How to optimise collections in dynamically typed languages?

How to optimise collections in dynamically typed languages?

Storage strategies

JIT experimentation test bed



JIT experimentation test bed

```
class W_ListObject(W_Object):
  def __init__(self):
    self.strategy = EmptyListStrategy()
    self.lstorage = None

  def append(self, w_item):
    self.strategy.append(self, w_item)

@singleton
class ListStrategy(object):
  def append(self, w_list, w_item):
    raise NotImplementedError("abstract")

@singleton
class EmptyListStrategy(ListStrategy):
  def append(self, w_list, w_item):
    if is_boxed_int(w_item):
      w_list.strategy = IntegerListStrategy()
      w_list.lstorage = new_empty_int_list()
    elif ...:
      ...
    else:
      w_list.strategy = ObjectListStrategy()
      w_list.lstorage = new_empty_object_list()
    w_list.append(w_item)
```

```
@singleton
class IntegerListStrategy(ListStrategy):
  def append(self,  w_list, w_item):
    if is_boxed_int(w_item):
      w_list.lstorage.append_int(\
        unbox_int(w_item))
      return
    self.switch_to_object_strategy(w_list)
    w_list.append(w_item)

  def switch_to_object_strategy(self, \
                                  w_list):
    lstorage = new_empty_object_list()
    for i in w_list.lstorage:
      lstorage.append_obj(box_int(i))
    w_list.strategy = ObjectListStrategy()
    w_list.lstorage = lstorage

@singleton
class ObjectListStrategy(ListStrategy):
  def append(self,  w_list, w_item):
    w_list.lstorage.append_obj(w_item)
```

JIT experimentation test bed

Factor	Count
LoC	1500

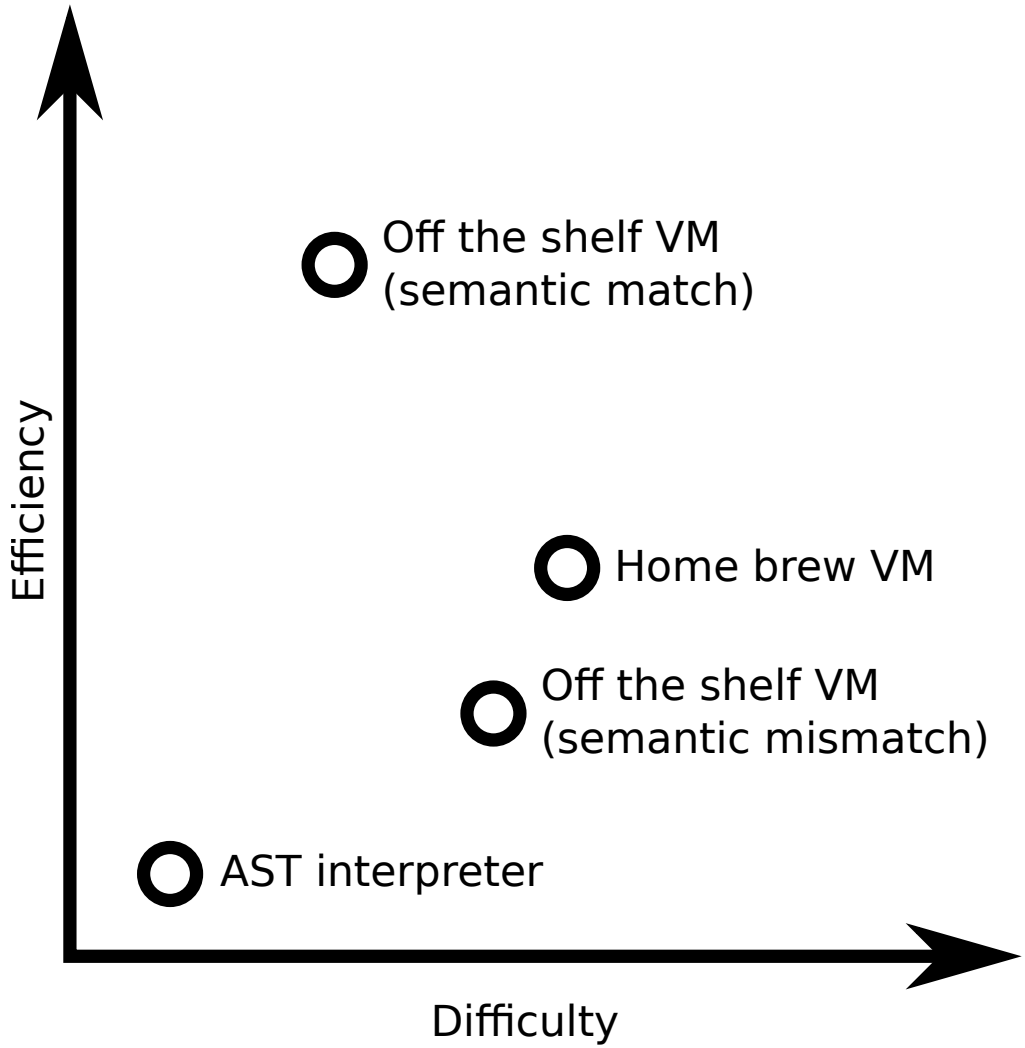
JIT experimentation test bed

Factor	Count
LoC	1500
Peak memory improvement	15%

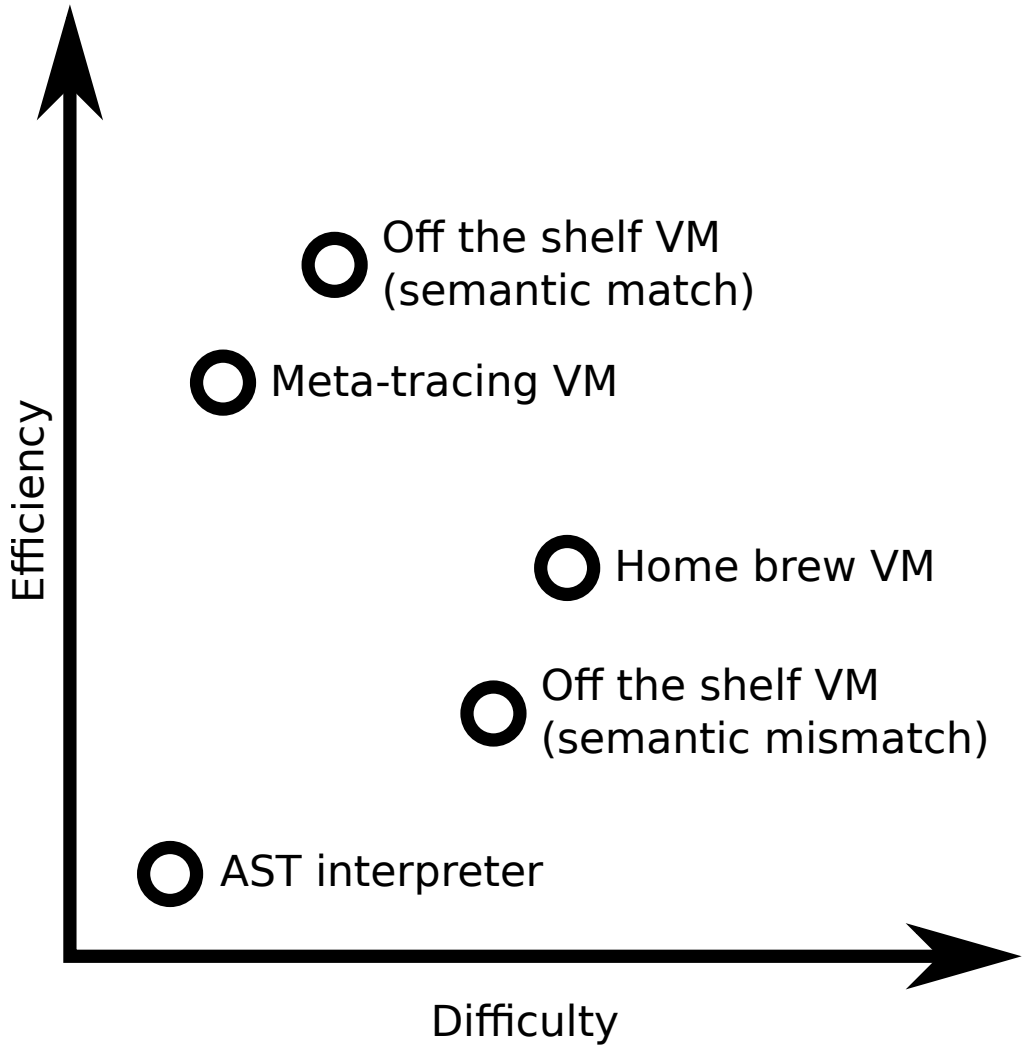
JIT experimentation test bed

Factor	Count
LoC	1500
Peak memory improvement	15%
Speed improvement	18%

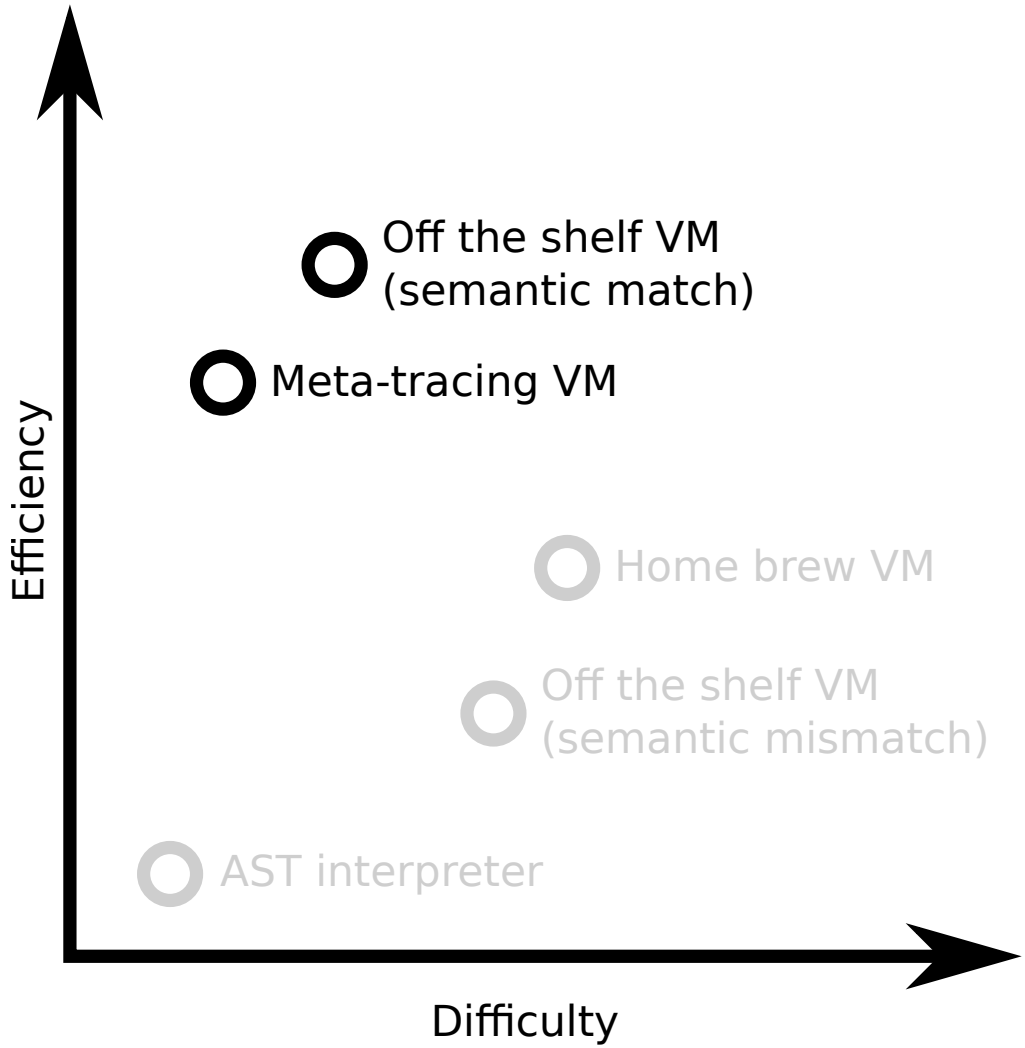
Traditional and new routes



Traditional and new routes



Traditional and new routes



Some personal thoughts on the future

Some personal thoughts on the future

- More RPython VMs.

Some personal thoughts on the future

- More RPython VMs.
  - Many more ‘medium-sized’ VMs: Converge, Pyrolog, etc.

Some personal thoughts on the future

- More RPython VMs.
  - Many more ‘medium-sized’ VMs: Converge, Pyrolog, etc.
  - Some more ‘industrial strength’ VMs: Topaz.

Some personal thoughts on the future

- More RPython VMs.
  - Many more ‘medium-sized’ VMs: Converge, Pyrolog, etc.
  - Some more ‘industrial strength’ VMs: Topaz.
- Meta-tracing experimentation.

Some personal thoughts on the future

- More RPython VMs.
  - Many more ‘medium-sized’ VMs: Converge, Pyrolog, etc.
  - Some more ‘industrial strength’ VMs: Topaz.
- Meta-tracing experimentation.
  - Reducing warmup times (caching profiling data?).

Some personal thoughts on the future

- More RPython VMs.
  - Many more ‘medium-sized’ VMs: Converge, Pyrolog, etc.
  - Some more ‘industrial strength’ VMs: Topaz.
- Meta-tracing experimentation.
  - Reducing warmup times (caching profiling data?).
  - Speeding up the meta-tracing interpreter.

Some personal thoughts on the future

Some personal thoughts on the future

- General experimentation.

Some personal thoughts on the future

- General experimentation.
  - Storage strategies etc.

Some personal thoughts on the future

- General experimentation.
  - Storage strategies etc.
- Meta-tracing $\neq$ RPython.

Some personal thoughts on the future

- General experimentation.
  - Storage strategies etc.
- Meta-tracing $\neq$ RPython.
  - New meta-tracing languages?

Some personal thoughts on the future

- General experimentation.
  - Storage strategies etc.
- Meta-tracing $\neq$ RPython.
  - New meta-tracing languages?
  - Related approaches (e.g. Truffle)?

Some personal thoughts on the future

- Never-before-plausible experimentation.

Some personal thoughts on the future

- Never-before-plausible experimentation.
  - VM composition.

Summary

What language designers dilemma?

Summary

- *Fast enough VMs in fast enough time*, Tratt
- Storage strategies for collections in dynamically typed languages, Bolz, Diekmann, Tratt
- The Impact of Meta-Tracing on VM Design and Implementation, Bolz, Tratt
- *Allocation removal by partial evaluation in a tracing JIT*, Bolz, Cuni, Fijalkowski, Leuschel, Pedroni, Rigo.
- Converge: http://convergepl.org/

Thank you for listening