

Fast enough VMs in fast enough time

Laurence Tratt

<http://tratt.net/laurie/>

King's College London

2012-05-09

Language designers dilemma

Implementation Effort

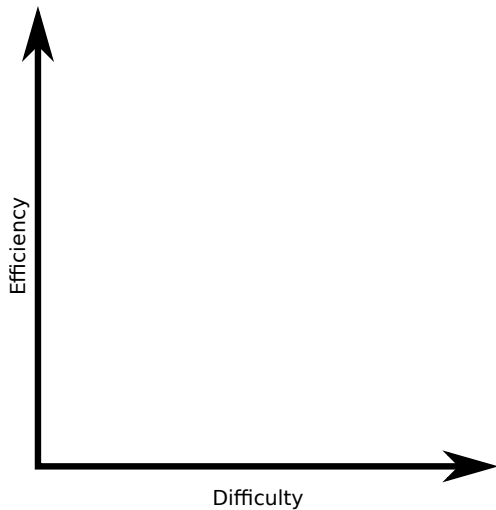
Language designers dilemma



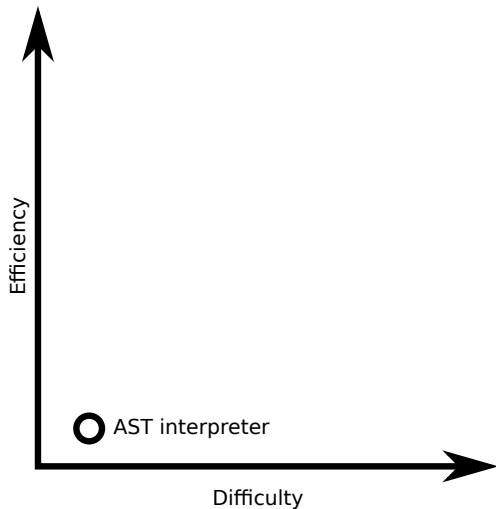
Language designers dilemma



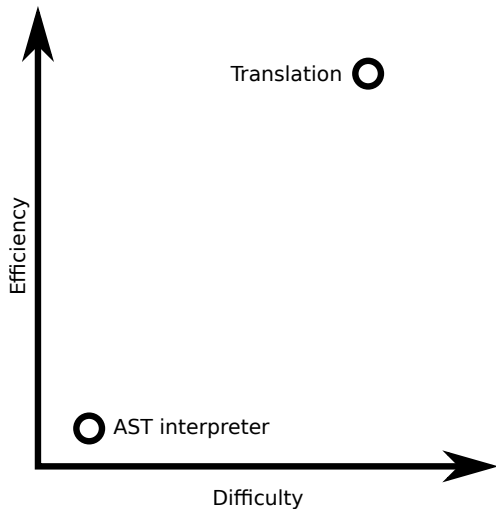
The traditional routes



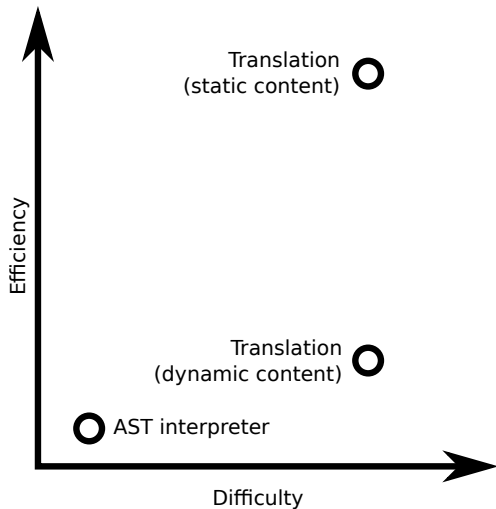
The traditional routes



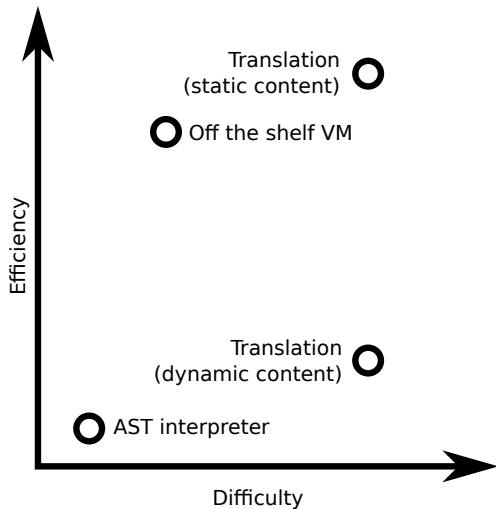
The traditional routes



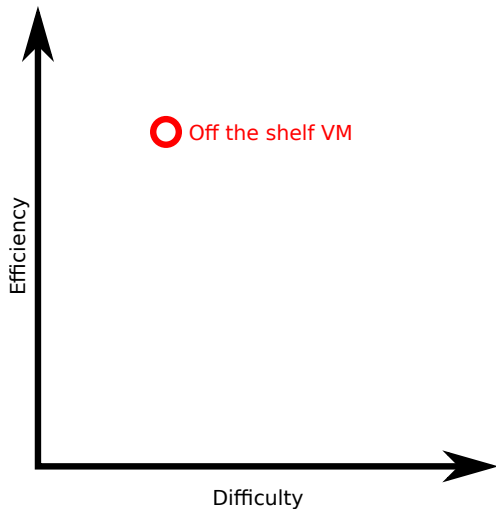
The traditional routes



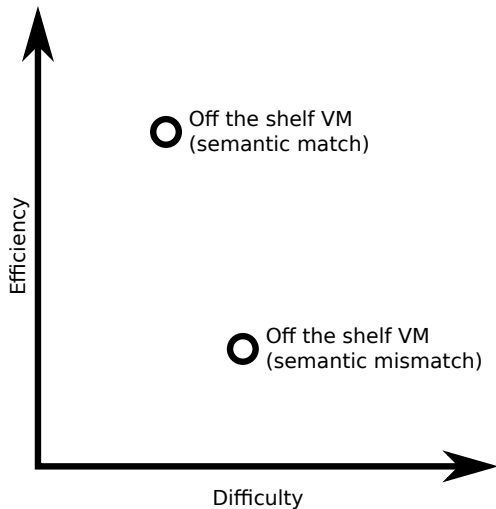
The traditional routes



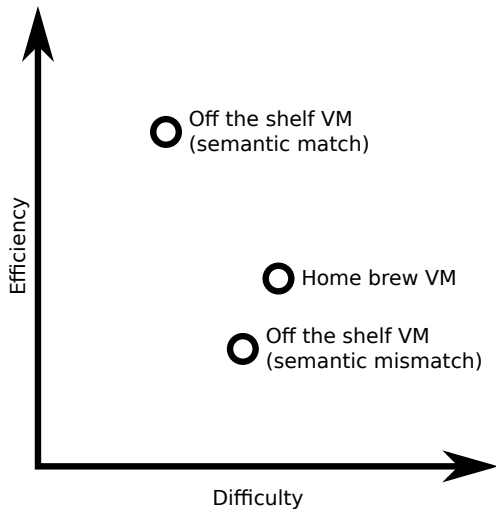
The traditional routes



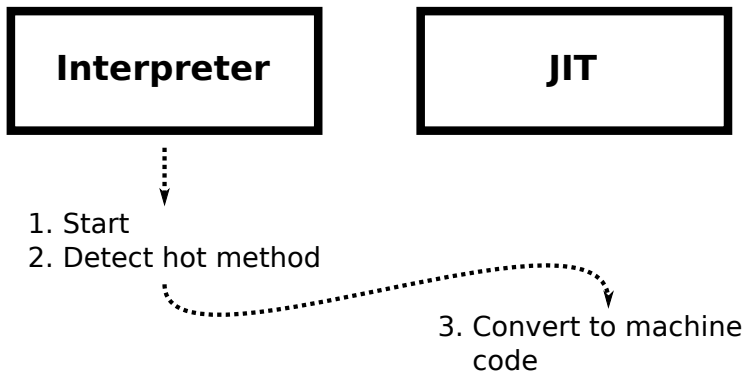
The traditional routes



The traditional routes



JIT VM components



Interpreter

JIT

Interpreter

- + Easy to write
- Slow

JIT

Interpreter

- + Easy to write
- Slow

JIT

- + Fast
- Difficult to write

Interpreter

- + Easy to write
- Slow

JIT

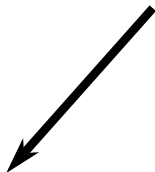
- + Fast
- Difficult to write



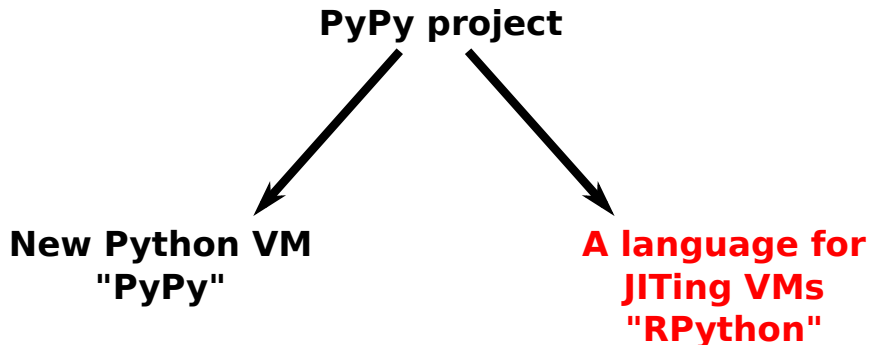
- Keeping interpreter and JIT in sync

PyPy project

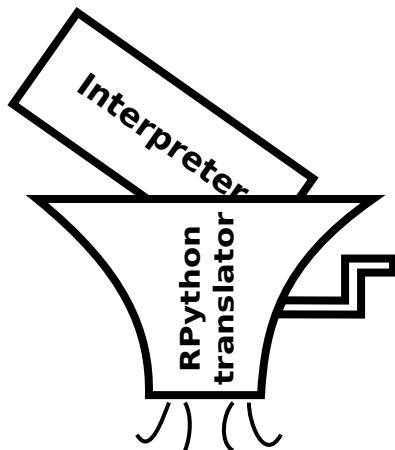
PyPy project



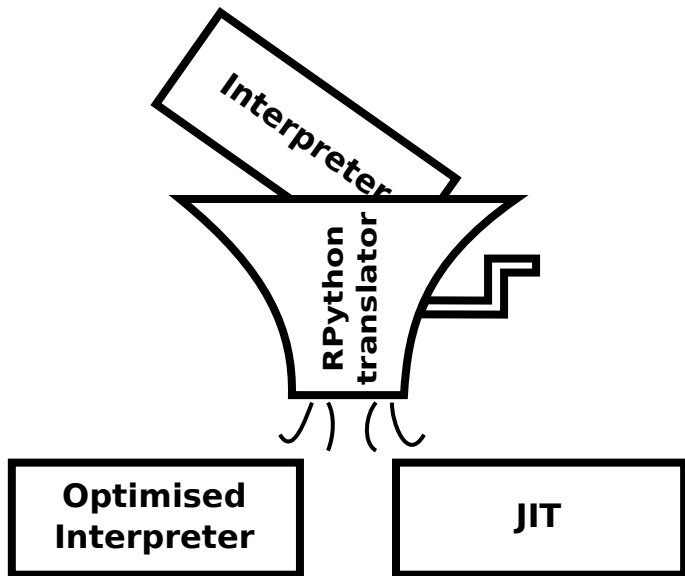
**New Python VM
"PyPy"**



RPython translation



RPython translation



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 ...:
        ...
```

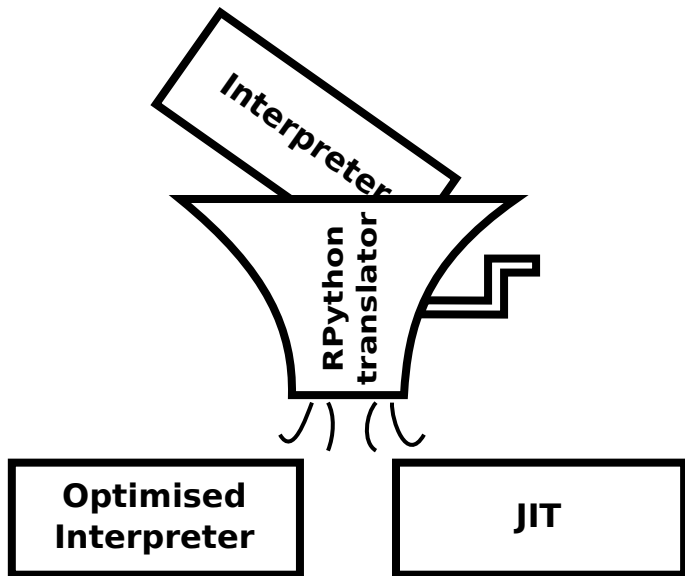
Observation: interpreters are big loops.

Adding a JIT to an RPython interpreter

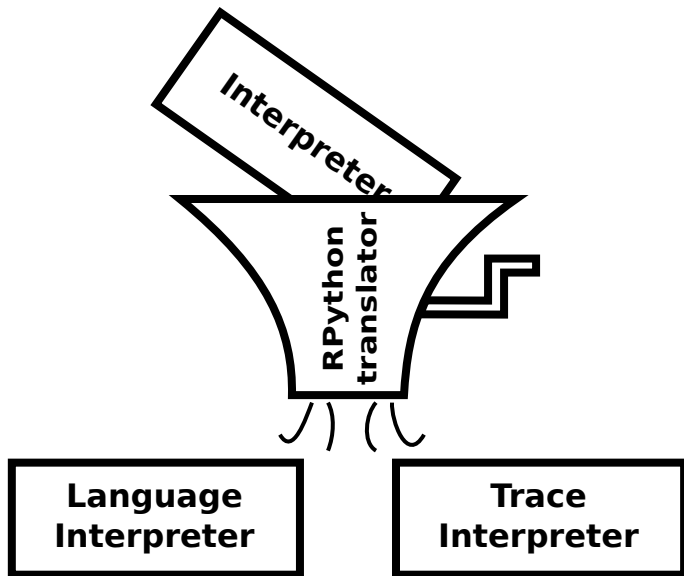
```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

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
```

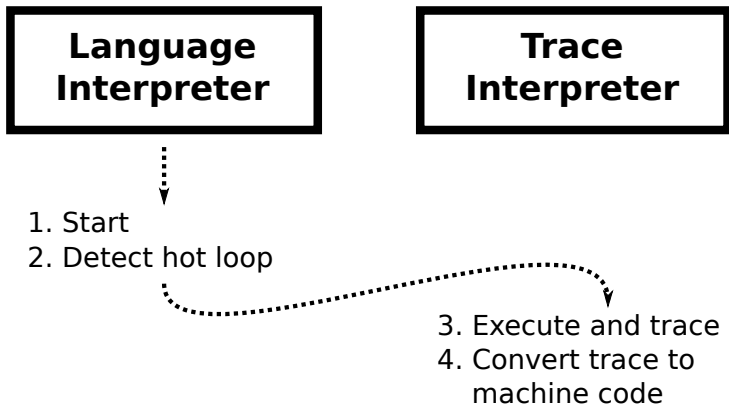
User program (lang <i>FL</i>)	Trace when x is set to 6
--------------------------------	--------------------------

<pre>if x < 0: x = x + 1 else: x = x + 2 x = x + 3</pre>	<pre>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)</pre>
---	---

User program (lang <i>FL</i>)	Optimised trace
--------------------------------	-----------------

<pre>if x < 0: x = x + 1 else: x = x + 2 x = x + 3</pre>	<pre>guard_type(x, int) guard_not_less_than(x, 0) x = int_add(x, 5)</pre>
---	---

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)
v14 = add(v11, 2)
...
```

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)
```

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)
```

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

A Converge VM in RPython.

A Converge VM in RPython.

How hard can it be?

Converge 1 VM

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.

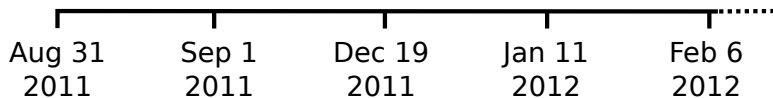
Converge 1 VM

- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.
- Highlights: C-level continuations; conservative GC; ported to Linux / OpenBSD / OS X / Windows.

Converge 1 VM

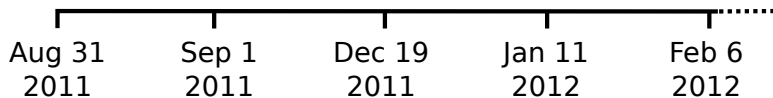
- Size: ~13KLoC. [Typical for medium-size VMs.]
- Approximate effort: 18 man months.
- Highlights: C-level continuations; conservative GC; ported to Linux / OpenBSD / OS X / Windows.
- Lowlights: *slow* and increasingly hard to maintain / optimise.

The experiment



The experiment

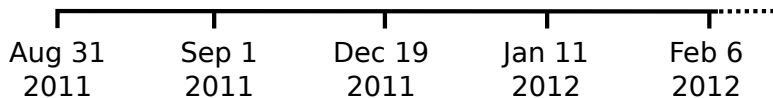
Clueless



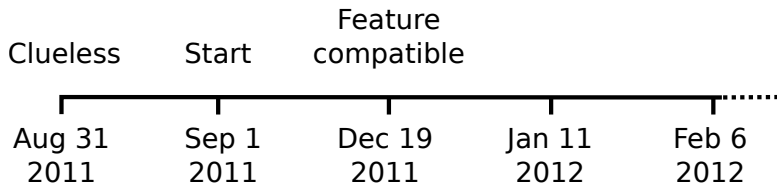
The experiment

Clueless

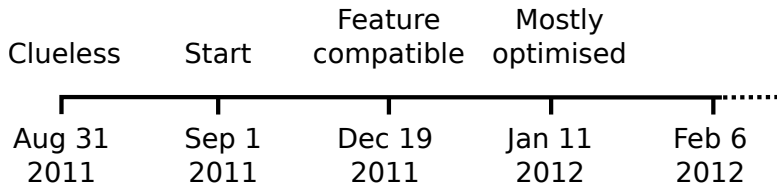
Start



The experiment



The experiment



The experiment



Converge 2 VM

- Size: 5.5KLoC (vs. ~13KLoC.)
- Approximate effort: 3 man months (vs. 18 man months).

Converge 2 VM

- Size: 5.5KLoC (vs. ~13KLoC.)
- Approximate effort: 3 man months (vs. 18 man months).
- Performance: worst case 2-3x speedup. Best case: 100+x.

Converge 2 VM

- Size: 5.5KLoC (vs. ~13KLoC.)
- Approximate effort: 3 man months (vs. 18 man months).
- Performance: worst case 2-3x speedup. Best case: 100+x.
- Now some numbers.
- [Full experimental details – and / or our repeatable benchmark – on request.]

Dhystone benchmark

VM	stone(50000)	stone(5000000)
HotSpot (1.7.0_147)	0.150	0.476
Lua (5.1.5)	0.361	36.558
LuaJIT2 (git #5dbb6671a3)	0.020	2.085
Converge1 (git #9084f0cdaf)	3.709	375.672
Converge2 (git #e44800ec7c)	0.258	5.859
CPython (2.7.2)	0.677	65.621
Jython (2.5.2)	4.166	12.777
PyPy (1.8)	0.150	3.260

Richards benchmark

VM	richards(10)	richards(100)
HotSpot (1.7.0_147)	0.157	0.271
Lua (5.1.5)	0.596	12.033
LuaJIT2 (git #5dbb6671a3)	0.080	1.429
Converge1 (git #9084f0cdaf)	18.256	185.150
Converge2 (git #e44800ec7c)	1.606	8.794
CPython (2.7.2)	2.806	27.976
Jython (2.5.2)	5.673	28.672
PyPy (1.8)	0.598	1.221

Fannkuch-redux benchmark

VM	fannkuch(10)	fannkuch(11)
HotSpot (1.7.0_147)	0.578	6.215
Lua (5.1.5)	13.619	177.926
LuaJIT2 (git #5dbb6671a3)	0.530	6.945
Converge1 (git #9084f0cdaf)	†	†
Converge2 (git #e44800ec7c)	6.415	79.714
CPython (2.7.2)	15.712	195.150
Jython (2.5.2)	16.486	167.376
PyPy (1.8)	3.242	40.023

- 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.

Inlining

User program

```
def f(x):  
    return 2 + g(x) + 3
```

```
def g(x):  
    if x < 0:  
        return 0  
    else:  
        return 1
```

Trace without inlining

```
return 2 + g(x) + 3
```

Trace with inlining

```
guard_not_less_than(x, 0)  
return 2 + 1 + 3
```

Backtracking code (à la Icon)

```
func range(x):
  i := 0
  while i < x:
    yield i
    i += 1
  fail

func main():
  for Sys::println(x := range(100) \
    & x % 2 == 0 & x)
```

Backtracking code (à la Icon)

```
func range(x):  
  i := 0  
  while i < x:  
    yield i  
    i += 1  
  fail  
  
func main():  
  for Sys::println(x := range(100) \  
    & x % 2 == 0 & x)
```

Bytecode for `i + 1`

```
ADD_FAILURE_FRAME(11)  
VAR_GET(x)  
INT(1)  
ADD  
REMOVE_FAILURE_FRAME()
```

Backtracking code (à la Icon)

```
func range(x):  
  i := 0  
  while i < x:  
    yield i  
    i += 1  
  fail  
  
func main():  
  for Sys::println(x := range(100) \  
    & x % 2 == 0 & x)
```

Bytecode for `i + 1`

```
ADD_FAILURE_FRAME(11)  
VAR_GET(x)  
INT(1)  
ADD  
REMOVE_FAILURE_FRAME()
```

Failure frame instructions ~25-30% of executed instructions

RPython: the bad points

- Tracing JITs unpredictable performance.

RPython: the bad points

- Tracing JITs unpredictable performance.
- Sparse documentation.

RPython: the bad points

- Tracing JITs unpredictable performance.
- Sparse documentation.
- The *Python* in *RPython*.

RPython: the bad points

- Tracing JITs unpredictable performance.
- Sparse documentation.
- The *Python* in *RPython*.
- Type inference errors.

RPython: the bad points

- Tracing JITs unpredictable performance.
- Sparse documentation.
- The *Python* in *RPython*.
- Type inference errors.
- Whole program translation.

RPython: the bad points

- Tracing JITs unpredictable performance.
- Sparse documentation.
- The *Python* in *RPython*.
- Type inference errors.
- Whole program translation.
- Implementation inconsistencies.

Summary

What language designers dilemma?

Summary

- *Fast enough VMs in fast enough time*, 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.