Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

2016 © Microsoft Corporation. All rights reserved.
Why Aren’t More Users More Happy With Our VMs?

Laurence Tratt

Warmup work in collaboration with: Edd Barrett, Carl Friedrich Bolz, Rebecca Killick, and Sarah Mount

KING’S
College
LONDON

Software Development Team
2018-05-17
JVMs bring "gcc -O2" to the masses

--Cliff Click: A JVM does that?
What do VM claims pertain to?

![Diagram showing iteration time and in-process iteration with phases of Compilation, Profiling Interpreter, and Peak Performance.]
What do VM claims pertain to?

![Graph showing iteration time and in-process iteration](image-url)
Users *always* perceive warmup

Maybe we should know how long it is?
The Warmup Experiment

Measure warmup of modern language implementations

Hypothesis: Small, deterministic programs reach a steady state of peak performance.
Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism

We added checksums to all benchmarks
2000 *in-process iterations*

30 *process executions*
Method 3: VMs

- Graal-0.22
- HHVM-3.19.1
- TruffleRuby 20170502
- Hotspot-8u121b13
- Luajit-2.0.4
- PyPy-5.7.1
- V8-5.8.283.32
- GCC-4.9.4

Note: same GCC (4.9.4) used for all compilation
Method 4: Machines

- Linux$_{4790}$, Debian 8, 24GiB RAM
- Linux$_{E3-1240v5}$, Debian 8, 32GiB RAM
- OpenBSD$_{4790}$, OpenBSD 6.0, 32GiB RAM

- Turbo boost and hyper-threading disabled
- Network card turned off.
- Daemons disabled (cron, smtpd)
Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible e.g.:

- Minimises I/O
- Sets fixed heap and stack ulimits
- Drops privileges to a 'clean' user account
- Automatically reboots the system prior to each proc. exec
- Reruns any proc. exec where the CPU was throttled
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)

Software Development Team
http://soft-dev.org/
The experiment has gone through many versions.

The following data is from the 1.5 run.
Warmup & flat (1)

N-Body, PyPy, Linux4790, Proc. exec. #24 (flat)

In-process iteration

Time (secs)

1.84603
1.83731
1.82860
1.81988
1.81116
1.80244
1.79372

1 201 401 601 801 1001 1201 1401 1601 1801 2000
Method 7: Classification

Classification algorithm (steps in order):

- All segs are equivalent: flat
- Final seg is in fastest set: warmup
Warmup & flat (2)

Fasta, V8, Linux_4790, Proc. exec. #15 (warmup)

In-process iteration

In-process iteration

1 201 401 601 801 1001 1201 1401 1601 1801 2000

Time (secs)

1.15432
1.14995
1.14558
1.14121
1.13685
1.13248
1.12811
1.15053
1.14273
1.13493

Software Development Team
http://soft-dev.org/
The graph shows the performance metrics of Richards, HotSpot, Linux\textsuperscript{E3-1240v5}, Proc. exec. #8 (slowdown). The x-axis represents the in-process iteration, ranging from 1 to 2000, while the y-axis represents the time in seconds, ranging from 0.23718 to 0.27458. The graph highlights a slowdown event at iteration 1801.
Classification algorithm (steps in order):

- All segs are equivalent: *flat*
- Final seg is in fastest set: *warmup*
- Final seg is not in fastest set: *slowdown*
Slowdown (2)

Fasta, V8, Linux_{4790}, Proc. exec. #26 (slowdown)
No steady state (1)

Binary Trees, V8, Linux\textsubscript{4790}, Proc. exec. #6 (no steady state)
Classification algorithm (steps in order):

All segs are equivalent: flat
Final seg is in fastest set: warmup
Final seg is not in fastest set: slowdown
Else: no steady state
Classification algorithm, in order:

All segs are equivalent: **flat**

Final seg is in fastest set: **warmup**

Final seg is not in fastest set: **slowdown**

Else: **no steady state**

**Good**
Classification algorithm, in order:

All segs are equivalent: *flat*

Final seg is in fastest set: *warmup*

Final seg is not in fastest set: *slowdown*

Else: *no steady state*

**Bad**
Inconsistent Process-executions

Binary Trees, V8, Linux_x86_64, Proc. exec. #15 (warmup)

Binary Trees, V8, Linux_x86_64, Proc. exec. #6 (no steady state)

(Same machine)
Inconsistent Process-executions

(Different machines. Bouncing ball Linux-specific)
## Individual benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Steady iter (#)</th>
<th>Steady perf (s)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grail</td>
<td>(271, 27, 1&lt;sup&gt;st&lt;/sup&gt;)</td>
<td>0.13277</td>
<td>6.0</td>
<td>0.06</td>
</tr>
<tr>
<td>HHVM</td>
<td>(105, 72, 4&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.86</td>
<td>2.0</td>
<td>0.2584</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(161, 137, 7&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.3509</td>
<td>2.0</td>
<td>0.13089</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(285, 3, 2&lt;sup&gt;nd&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>PyPy</td>
<td>(278, 36)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(202, 111)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>V8</td>
<td>(35, 92, 8&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grail</td>
<td>(271, 3, 4&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.3359</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>HHVM</td>
<td>(289, 2&lt;sup&gt;nd&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(281, 181, 1&lt;sup&gt;st&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(285, 181, 7&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>PyPy</td>
<td>(287, 27)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(202, 33)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>V8</td>
<td>(293, 2&lt;sup&gt;nd&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grail</td>
<td>(271, 2, 1&lt;sup&gt;st&lt;/sup&gt;)</td>
<td>0.54951</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>HHVM</td>
<td>(132, 101, 7&lt;sup&gt;th&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(271, 31)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(202, 6, 1&lt;sup&gt;st&lt;/sup&gt;)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>PyPy</td>
<td>(277, 31)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(271, 31)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>V8</td>
<td>(281, 181)</td>
<td>0.00</td>
<td>2.0</td>
<td>0.00</td>
</tr>
<tr>
<td>Class</td>
<td>Steady iter (#)</td>
<td>Steady iter (s)</td>
<td>Steady perf (s)</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>(27L, 2Î„, 1Î”, 1s)</td>
<td>775.0 (1.5,780.0)</td>
<td>425.16 (0.246,426.800)</td>
<td>0.54581 ± 0.033116</td>
</tr>
<tr>
<td>Graal</td>
<td>(2)</td>
<td>14.0 (2.0,94.6)</td>
<td>13.60 (0.830,98.737)</td>
<td>1.05685 ± 0.000126</td>
</tr>
<tr>
<td>HHVM</td>
<td>(29L, 1Î”)</td>
<td>7.0 (7.0,7.5)</td>
<td>1.91 (1.902,3.645)</td>
<td>0.31472 ± 0.169143</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(1l)</td>
<td>1.0 (1.0,45.2)</td>
<td>0.00 (0.000,20.597)</td>
<td>0.46480 ± 0.000085</td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td>0.00 (0.000,20.597)</td>
<td>± 0.000085</td>
</tr>
<tr>
<td>PyPy</td>
<td>(27Î„, 3Î”)</td>
<td>3.0 (3.0,3.0)</td>
<td>0.52 (0.523,0.529)</td>
<td>± 0.000034</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(25L, 5Î”)</td>
<td></td>
<td></td>
<td>± 0.000034</td>
</tr>
</tbody>
</table>

*spectralnorm*
### Overall benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux\textsubscript{4790}</th>
<th>Linux\textsubscript{1240v5}</th>
<th>OpenBSD\textsubscript{4790}†</th>
</tr>
</thead>
<tbody>
<tr>
<td>〈VM, benchmark〉 pairs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>⊲</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>⊳</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>≈</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>=</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>≆</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

| Process executions | | | |
|−     | 22.0\%                      | 23.3\%                      | 37.7\%                      |
|⊳     | 48.3\%                      | 43.9\%                      | 35.2\%                      |
| ⊳     | 20.1\%                      | 22.1\%                      | 12.1\%                      |
| ≈     | 9.6\%                       | 10.8\%                      | 15.0\%                      |
Summary

Classical warmup occurs for only:

67.2%–70.3% of process executions

37.8%–40.0% of (VM, benchmark) pairs

12.5% of benchmarks for (VM, benchmark, machine) triples
Are odd effects correlated with compilation and GC?

Fasta, PyPy, Linux_E3-1240v5, Proc. exec. #4 (no steady state)

In-process iteration

Time (secs)
Are odd effects correlated with compilation and GC?

Richards, HotSpot, Linux\textsubscript{E3-1240v5}, Proc. exec. #3 (slowdown)

- Time (secs)
- JIT (secs)
- GC (secs)

![Graph showing time, JIT, and GC over in-process iterations.](http://soft-dev.org/)
Are odd effects correlated with compilation and GC?

Fannkuch Redux, HotSpot, Linux E3-1240v5, Proc. exec. #4 (slowdown)

In-process iteration

Time (secs)

JIT (secs)

GC (secs)
Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion
A war story: the basis of a fix

diff --git a/rrpython/jit/metainterpreter.py b/rrpython/jit/metainterpreter.py
--- a/rrpython/jit/metainterpreter.py
+++ b/rrpython/jit/metainterpreter.py
@@ -951,9 +951,31 @@
     if warnrunnerstate.inlining:
         if warnrunnerstate.can_inlineCallable(greenboxes):
             # We've found a potentially inlinable function; now we need to
-+             # see if it's already on the stack. In other words: are we about
-+             # to enter recursion? If so, we don't want to inline the
-+             # recursion, which would be equivalent to unrolling a while
-+             # loop.
-+             portal_code = targetjitdriver_sd.mainjitcode
-+             return self.metainterpreter.perform_call(portal_code, allboxes,
-+                 greenkey=greenboxes)
+             inline = True
+             if self.metainterpreter.is_main_jitcode(portal_code):
+                 for gk, _ in self.metainterpreter.portal_trace_positions:
+                     if gk is None:
+                         continue
+                     assert len(gk) == len(greenboxes)
+                     i = 0
+                     for i in range(len(gk)):
+                         if not gk[i].same_constant(greenboxes[i]):
+                             break
+                     else:
+                         # The greenkey of a trace position on the stack
+                         # matches what we have, which means we're definitely
+                         # about to recurse.
+                         inline = False
+                         break
+                 if inline:
+                     return self.metainterpreter.perform_call(portal_code, allboxes,
+                         greenkey=greenboxes)
Success: slow benchmark now 13.5x faster
### A war story: data

<table>
<thead>
<tr>
<th>#unrollings</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>hexiom2</td>
<td>1.3</td>
<td>1.4</td>
<td>1.1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>raytrace-simple</td>
<td>3.3</td>
<td>3.1</td>
<td>2.8</td>
<td>1.4</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>spectral-norm</td>
<td>3.3</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>sympy_str</td>
<td>1.5</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>telco</td>
<td>4</td>
<td>2.5</td>
<td>2.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>polymorphism</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.08</td>
<td>0.09</td>
</tr>
</tbody>
</table>

http://marc.info/?l=pypy-dev&m=141587744128967&w=2
The benchmark suite said 7 levels, so that’s what I suggested.
Benchmarks guide our optimisations
17 JavaScript benchmarks from V8
Octane: pdf.js explodes

```bash
5 d8 run.js
Richards
DeltaBlue
Encrypt
Decrypt
RayTrace
Earley
Boyer
ReqExp
Splay
NavierStokes
PdfJS

<--- Last few GCs --->
14907865 ms: Mark-sweep 1093.9 (1434.4) -> 1093.4 (1434.4) MB, 274.8 / 0.0 ms [allocation failure] [GC in old space
14908140 ms: Mark-sweep 1093.4 (1434.4) -> 1093.4 (1434.4) MB, 274.4 / 0.0 ms [allocation failure] [GC in old space
14908421 ms: Mark-sweep 1093.4 (1434.4) -> 1100.5 (1418.4) MB, 260.9 / 0.0 ms [last resort gc]
14908703 ms: Mark-sweep 1100.5 (1418.4) -> 1107.8 (1418.4) MB, 282.1 / 0.0 ms [last resort gc]

<--- JS stacktrace --->
==== JS stack trace ===============================

Security context: 0x20d333ad3ba9 <JS Object>
  2: extractFontProgram (aka TypeParser_extractFontProgram) [pdf.js:17004] [pc=0x3a13b275421b] {this=0x3de358283}
  3: new TypeFont [pdf.js:17216] [pc=0x3a13b2752078] {this=0x4603fbd9ea9 <a TypeFont with map 0x1f82213447e1>}

# Fatal error in CALL_AND_RETRY_LAST
# Allocation failed - process out of memory
#
# zsh: illegal hardware instruction d8 run.js
```
Octane: analysing pdf.js

Process execution #1

Time (sec.)

In-progress iteration

Software Development Team

35 / 51

http://soft-dev.org/
Octane: analysing pdf.js
Octane: debugging

```javascript
var pdf_file = "test.pdf";
var canvas_logs = [];

var PdfJS = new BenchmarkSuite("PdfJS", [10124921], [
  new Benchmark("PdfJS", false, false, 24,
    setupPdfJS, tearDownPdfJS, null, 4)
]);

NORMAL /master /pdfjs.js jav...-utf-8[unix] 0% 28/33053 : 1

function runPdfJS() {
  PDFJS.getDocument(pdf_file).then(function(pdf) {
    var canvas = PdfJS_window.document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var renderContext = (canvas.getContext: context);
    canvas_logs.push(context.__log__);

    // Cycle through all pages.
    function renderPages(i, j) {
      if (i >= j) return;
      context.clearRect(0, 6, canvas.width, canvas.height);
      pdf.getPage(i).then(function(page) {
        renderContext.viewport = page.getViewport(1);
        canvas.height = renderContext.viewport_height;
        canvas.width = renderContext.viewport_width;
        page.render(renderContext).then(renderPages.bind(null, i + 1, j));
      });
    }
    renderPages(1, pdf.numPages);
  });

  // Wait for everything to complete.
  PdfJS_window.flushTimeouts();
}
```
Fix memory leak in pdfjs.js. #42

@litratt wants to merge 2 commits into chromium:master from litratt:master

Changes from all commits: 1 file

pdfjs.js

```javascript
function setupPDFJS() {
  canvas_logs.length = 0;
}

function runPDFJS() {
  PDFJS.getDocument(pdf_file).then(function(pdf) {
    var canvas = PDFJS_window.document.getElementById('canvas');
    var context = canvas.getContext('2d');
  });
}
```
Octane: other issues

pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak

zlib complains that Cannot enlarge memory arrays in asm.js (a memory leak? I don’t know)
Summary

Why aren’t more users more happy with our VMs?

My thesis: benchmarking and benchmarks are performance destiny.
1. Run benchmarks for longer to uncover issues.
2. Accept that neither peak performance or a steady state may occur.
3. Always report warmup time.
4. Stop over-training on small benchmark suites.
5. Collect more benchmarks.
6. Focus on predictable performance.
Can we fix existing VMs?

At least a bit... but a lot? Unclear.

In case we can’t, I have an idea...
Meta-tracing JITs

**FL Interpreter**

```python
program_counter = 0; stack = []
vrs = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vs[read_var_name_from_instruction()]
        )
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vs[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:
            ...  # ... else...
        program_counter += 1
    elif instr == INSTR_ADD:
        lhs = stack.pop()
        rhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            stack.push(lhs + rhs)
        else:
            ...  # ... else...
        program_counter += 1
    elif instr == INSTR_IF:
        result = stack.pop()
        if result -- True:
            program_counter += 1
        else:
            program_counter += 1
    read_jump_if_instruction()  # Read and discard next
...  # ... else...
```

http://soft-dev.org/
**FL Interpreter**

```python
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)**

```plaintext
if x < 0:
    x = x + 1
else:
    x = x + 2
    x = x + 3
```
**FL Interpreter**

```python
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**

```python
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)
...  
```
<table>
<thead>
<tr>
<th>FL Interpreter</th>
<th>User program (lang FL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_counter = 0; stack = []</td>
<td>if x &lt; 0:</td>
</tr>
<tr>
<td>vars = [...]</td>
<td>x = x + 1</td>
</tr>
<tr>
<td>while True:</td>
<td>else:</td>
</tr>
<tr>
<td>jit_merge_point(program_counter)</td>
<td>x = x + 2</td>
</tr>
<tr>
<td>instr = load_instruction(program_counter)</td>
<td>x = x + 3</td>
</tr>
<tr>
<td>if instr == INSTR_VAR_GET:</td>
<td></td>
</tr>
<tr>
<td>stack.push(</td>
<td></td>
</tr>
<tr>
<td>vars[read_var_name_from_instruction()]</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_VAR_SET:</td>
<td></td>
</tr>
<tr>
<td>vars[read_var_name_from_instruction()]</td>
<td></td>
</tr>
<tr>
<td>= stack.pop()</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_INT:</td>
<td></td>
</tr>
<tr>
<td>stack.push(read_int_from_instruction())</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_LESS_THAN:</td>
<td></td>
</tr>
<tr>
<td>rhs = stack.pop()</td>
<td></td>
</tr>
<tr>
<td>lhs = stack.pop()</td>
<td></td>
</tr>
<tr>
<td>if isinstance(lhs, int) and isinstance(rhs, int):</td>
<td></td>
</tr>
<tr>
<td>if lhs &lt; rhs:</td>
<td></td>
</tr>
<tr>
<td>stack.push(True)</td>
<td></td>
</tr>
<tr>
<td>else:</td>
<td></td>
</tr>
<tr>
<td>stack.push(False)</td>
<td></td>
</tr>
<tr>
<td>else: ...</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
</tbody>
</table>
FL Interpreter

```python
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:
        stack.push(read_int_from_instruction())
        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:
            stack.push(False)
        program_counter += 1
```

Initial trace

```python
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)
...```
Meta-tracer states

- Interpreter
- Tracer
- Machine code
- Blackhole interpreter

States:
- Hot
- Compile
- Safepoint
- Guard failure
Meta-tracer performance (now)

- Interpreter: 1x
- Tracer: 200x
- Machine code: 0.1x

Relationships:
- Hot
- Compile
- Safepoint
- Guard failure

Software Development Team
http://soft-dev.org/
Meta-tracer performance (our aim)

Interpreter (1x) → Tracer (2x) → Machine code (0.1x) → Safepoint (Guard failure)

Hot

Compile
Meta-tracing JITs

**FL Interpreter**

```python
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:
            ...  # Pseudo-code
        program_counter += 1
```

**Initial trace**

```python
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)
...```

VM Warmup Blows Hot and Cold
E. Barrett, C. F. Bolz, R. Killick, V. Knight, S. Mount and L. Tratt.

Rigorous Benchmarking in Reasonable Time
T. Kalibera and R. Jones

Specialising Dynamic Techniques for Implementing the Ruby Programming Language
C. Seaton (Chapter 4)

Quantifying performance changes with effect size confidence intervals
T. Kalibera and R. Jones
**warmup_stats**  Use our statistical method on your VMs
http://soft-dev.org/src/warmup_stats/

**Krun**  Run experiments in a controlled manner
http://soft-dev.org/src/krun/
• EPSRC: COOLER and Lecture.
• Oracle.
• Cloudflare.
Thanks for listening