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
2018-11-06
JVMs bring "gcc -O2" to the masses

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

- Compilation
- Profiling Interpreter

iteration time

in-process iteration
What do VM claims pertain to?

![Diagram showing iteration time versus in-process iteration with stages labeled Compilation, Profiling Interpreter, and Peak Performance.](http://soft-dev.org/)
What do VM claims pertain to?

![Graph showing iteration time and in-process iteration with a warmup phase.]
Users *always* perceive warmup
Users *always* perceive warmup

Maybe we should know how long it is?
Measure warmup of modern language implementations
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)
Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism
The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism

We added checksums to all benchmarks
Method 2: How long to run?

2000 \textit{in-process iterations}
Method 2: How long to run?

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\textsubscript{4790}, Debian 8, 24GiB RAM
- Linux\textsubscript{E3-1240v5}, Debian 8, 32GiB RAM
- OpenBSD\textsubscript{4790}, OpenBSD 6.0, 32GiB RAM
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
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, ...)

[Image: HTTP://SOFT-DEV.ORG/]
The experiment has gone through many versions.
The experiment has gone through many versions.

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

Fannkuch Redux, LuaJIT, OpenBSD\textsuperscript{4790}, Proc. exec. #14 (warmup)
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD4790, Proc. exec. #14 (warmup)
Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #14 (warmup)

In-process iteration

Time (secs)
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD\textsubscript{4790}, Proc. exec. #14 (warmup)

Changepoint

In-process iteration

Time (secs)
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD\textsubscript{4790}, Proc. exec. #14 (warmup)

Changepoint

Changepoint segment

Changepoint
Warmup & flat (1)

N-Body, PyPy, Linux, Proc. exec. #24 (flat)
Classification algorithm (steps in order):

All segs are equivalent: *flat*
Method 7: Classification

Classification algorithm (steps in order):

All segs are equivalent: \textit{flat}

Final seg is in fastest set: \textit{warmup}
Warmup & flat (2)

Fasta, V8, Linux\textsuperscript{4790}, Proc. exec. #15 (warmup)

Time (secs)

In-process iteration

1.15432
1.14995
1.14558
1.14121
1.13685
1.13248
1.12811

1 201 401 601 801 1001 1201 1401 1601 1801 2000

14/51 HTTP://SOFT-DEV.ORG/
Method 7: Classification

Classification algorithm (steps in order):

All segs are equivalent: *flat*

Final seg is in fastest set: *warmup*
Method 7: Classification

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*
In-process iteration

Time (secs)

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*
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
Warmup or no steady state?

Fannkuch Redux, HotSpot, Linux, Proc. exec. #1 (warmup)

In-process iteration vs. Time (secs)
Inconsistent Process-executions

Binary Trees, V8, Linux4790, Proc. exec. #15 (warmup)

Binary Trees, V8, Linux4790, 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 iter (s)</th>
<th>Steady perf (s)</th>
<th>Class</th>
<th>Steady iter (#)</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>Graal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TruffleRuby</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Results

- **C**
  - Steady iter (#): 1003.0
  - Steady iter (s): 2621.96
  - Steady perf (s): 2.61136

- **Graal**
  - Steady iter (#): 228.0
  - Steady iter (s): 106.34
  - Steady perf (s): 0.46327

- **HHVM**
  - Steady iter (#): 1.0
  - Steady iter (s): 0.00
  - Steady perf (s): 0.47422

- **HotSpot**
  - Steady iter (#): (25, 5, 1)
  - Steady iter (s): (1.0, 5.38) ± 0.0136
  - Steady perf (s): 0.00

- **LuaJIT**
  - Steady iter (#): (21, 1, 1)
  - Steady iter (s): 2.0
  - Steady perf (s): 0.88490

- **PyPy**
  - Steady iter (#): (26, 3, 11)
  - Steady iter (s): 0.32
  - Steady perf (s): 0.47684

- **TruffleRuby**
  - Steady iter (#): (25, 3, 11)
  - Steady iter (s): 2.0
  - Steady perf (s): 0.47684

- **V8**
  - Steady iter (#): (25, 2, 7)
  - Steady iter (s): 1.0
  - Steady perf (s): 0.47684

---

**Software Development Team**

http://soft-dev.org/
## Individual benchmark stats

<table>
<thead>
<tr>
<th>Class.</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>(\times) ((27\downarrow, 2\leftarrow, 1\uparrow))</td>
<td>775.0 ((1.5,780.0))</td>
<td>425.16 ((0.246,426.809))</td>
</tr>
<tr>
<td>Graal</td>
<td>(\Uparrow)</td>
<td>14.0 ((2.0,94.6))</td>
<td>13.60 ((0.830,98.737))</td>
</tr>
<tr>
<td>HHVM</td>
<td>(\times) ((29\downarrow, 1\omega))</td>
<td>spectral</td>
<td>n/a</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(\Uparrow)</td>
<td>7.0 ((7.0,7.5))</td>
<td>1.91 ((1.902,3.645))</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>—</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>PyPy</td>
<td>(\times) ((27\downarrow, 3\uparrow))</td>
<td>1.0 ((1.0,45.2))</td>
<td>0.00 ((0.000,20.597))</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(\times) ((25\downarrow, 5\omega))</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>V8</td>
<td>(\Uparrow)</td>
<td>3.0 ((3.0,3.0))</td>
<td>0.52 ((0.523,0.526))</td>
</tr>
</tbody>
</table>

---

**Software Development Team**

22 / 51

[http://soft-dev.org/](http://soft-dev.org/)
## 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%          |
## Overall benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux_{4790}</th>
<th>Linux_{1240v5}</th>
<th>OpenBSD_{4790}†</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>⟨VM, benchmark⟩ pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>(\supset)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\cong)</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
<tr>
<td></td>
<td>Process executions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>22.0%</td>
<td>23.3%</td>
<td>37.7%</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>48.3%</td>
<td>43.9%</td>
<td>35.2%</td>
</tr>
<tr>
<td>(\supset)</td>
<td>20.1%</td>
<td>22.1%</td>
<td>12.1%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>9.6%</td>
<td>10.8%</td>
<td>15.0%</td>
</tr>
</tbody>
</table>
Classical warmup occurs for only:
Summary

Classical warmup occurs for only:

67.2%–70.3% of process executions
Summary

Classical warmup occurs for only:

67.2%–70.3% of process executions

37.8%–40.0% of (VM, benchmark) pairs
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\textsubscript{E3 – 1240v5}, Proc. exec. #2 (no steady state)

In-process iteration

Time (secs)

GC

JIT

0.00000
0.00723
0.01446

0.759 \times 10^9
1.491 \times 10^9
2.222 \times 10^9

2.65441
2.57464
2.49487
2.41511
2.33534
2.25557
2.17580

1 201 401 601 801 1001 1201 1401 1601 1801 2000

Software Development Team

http://soft-dev.org/
Are odd effects correlated with compilation and GC?

Richards, HotSpot, Linux4790, Proc. exec. #4 (slowdown)

In-process iteration

Time (secs)

JIT (secs)

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

Fannkuch Redux, HotSpot, Linux4790, Proc. exec. #3 (no steady state)

In-process iteration

Time (secs)

GC (secs)

JIT (secs)
Benchmark suites
Benchmarks guide our optimisations
Benchmark suites

Benchmarks guide our optimisations

Are they complete guides?
Symptom: poor performance of a Pyston benchmark on PyPy
Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion
A war story

Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion

Fix: Check for recursion before tracing
A war story: the basis of a fix

diff --git a/rpython/jit/metainterpreter/pyjitpl.py b/rpython/jit/metainterpreter/pyjitpl.py
--- a/rpython/jit/metainterpreter/pyjitpl.py
+++ b/rpython/jit/metainterpreter/pyjitpl.py
@@ -951,9 +951,31 @@
     if warmrunnerstate.inlining:
         if warmrunnerstate.can_inline_callable(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)
A war story: mixed fortunes

Success: slow benchmark now 13.5x faster
Success: slow benchmark now 13.5x faster

Failure: some PyPy benchmarks slow down
A war story: mixed fortunes

Success: slow benchmark now 13.5x faster

Failure: some PyPy benchmarks slow down

Solution: allow *some* tracing into recursion
#unrollings |  1  |  2  |  3  |  5  |  7  | 10  |
-----------------+------+------+------+------+------+------+
hexiom2          | 1.3 | 1.4 | 1.1 | 1.0 | 1.0 | 1.0 |
raytrace-simple  | 3.3 | 3.1 | 2.8 | 1.4 | 1.0 | 1.0 |
spectral-norm    | 3.3 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
sympy_str        | 1.5 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
telco            | 4   | 2.5 | 2.0 | 1.0 | 1.0 | 1.0 |
-----------------+------+------+------+------+------+------+
polymorphism     | 0.07| 0.07| 0.07| 0.07| 0.08| 0.09|

http://marc.info/?l=pypy-dev&m=141587744128967&w=2
The benchmark suite said 7 levels, so that’s what I suggested
The benchmark suite said 7 levels, so that’s what I suggested

Even though I doubted it was the right global value
Benchmark suites (2)
Benchmarks guide our optimisations
Benchmarks guide our optimisations

Are they correct guides?
17 JavaScript benchmarks from V8
17 JavaScript benchmarks from V8

Let’s make each benchmark run for 2000 iterations
Octane: pdf.js explodes

$ d8 run.js
Richards
DeltaBlue
Encrypt
Decrypt
RayTrace
Earley
Boyer
RegExp
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) -> 1093.3 (1434.4) MB, 274.4 / 0.0 ms [allocation failure] [GC in old space
14908421 ms: Mark-sweep 1093.3 (1434.4) -> 1100.5 (1418.4) MB, 280.9 / 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 Type1Parser_extractFontProgram) [pdfjs.js:17004] [pc=0x3a13b275421b] (this=0x3de358283581 <a type1Parser with map 0x1f822131a411>,stream=0x4603fbdc4e1 <an Uint8Array with map 0x393de2707fe1>)
  3: new Type1Font [pdfjs.js:17216] [pc=0x3a13b2752078] (this=0x4603fbd9c09 <String[12]: JTYMKN+CMR17>,file=0x4603fb...)

# Fatal error in CALL_AND_RETRY_LAST
# Allocation failed - process out of memory
#
zsh: illegal hardware instruction  d8 run.js
In-process iteration

Time (secs)

Process execution #1
In-process iteration
-0.2442
2.0974
4.4389
6.7805
9.1221
11.4636
13.8052
Time (secs)
Process execution #1
In-process iteration

0 276 551 826 1102 1378 1653 1928 2204 2480 2755

Software Development Team
http://soft-dev.org/
Octane: debugging

```javascript
// Benchmark set-up. (c) by Google. //

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

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

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

    // Cycle through all pages.
    function renderPages(i, j) {
      if (i > j) return;
      context.clearRect(0, 0, 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(i, pdf.numPages);
    }

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

itratt wants to merge 2 commits into chromium:master from itratt:master

Conversation 5  Commits 2  Files changed 1

Changes from all commits 1 file +1 -0

1 pdfjs.js

```javascript
@@ -43,6 +43,7 @@ function setupPdfJS() {
                     }
     function runPdfJS() {
         canvas_logs.length = 0;
         PDFJS.getDocument(pdf_file).then(function(pdf) {
             var canvas = PdfJS_window.document.getElementById('canvas');
             var context = canvas.getContext('2d');
```
pdfjs isn’t the only problem
pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak
pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak

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

Timings are made with a non-monotonic microsecond timer
Summary

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

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

My thesis: benchmarking and benchmarks are performance destiny.
Summary

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

My thesis: benchmarking *and* benchmarks are performance destiny.

Ours have misled us.
How to benchmark a bit better

1. Don't benchmark for long to uncover issues.
2. Accept that neither peak performance nor steady state may occur.
3. Always report warm-up times.
4. Avoid over-training on small benchmark suites.
5. Collect more benchmarks.
6. Focus on predictable performance.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
1. Run benchmarks for longer to uncover issues.
2. Accept that neither peak performance or a steady state may occur.
How to benchmark a bit better

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.
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.
How to benchmark a bit better

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.
How to benchmark a bit better

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.
The big question

Can we exist 6-S? At least a bit... but a lot? I'm not sure. If we can't, I have an idea...

http://soft-dev.org/
Can we fix existing VMs?
Can we fix existing VMs?

At least a bit... but a lot? Unclear.
Can we fix existing VMs?

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

In case we can’t, I have an idea...
**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
    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
```

**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: ...
    program_counter += 1
```
## Meta-tracing JITs

### FL Interpreter

```python
def 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)

```python
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**

```plaintext
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 → Safepoint → Hot → Compile → Guard failure

http://soft-dev.org/
Meta-tracer states

Interpreter → Tracer → Machine code

Hot

Compile

Safepoint

Guard failure

Blackhole interpreter
Meta-tracer states

Interpreter → Tracer → Machine code

Hot

Compile

Guard failure

Safepoint

Blackhole interpreter
Meta-tracer states

- Interpreter
- Tracer
- Machine code

- Hot
- Compile

- Safepoint
- Guard failure

- Blackhole interpreter
Meta-tracer states

Interpreter → Hot → Tracer → Compile → Machine code

Safepoint → Blackhole interpreter → Guard failure
Meta-tracer states

Interpreter
Tracer
Machine code

Hot → Tracer

Compile → Machine code

Guard failure → Blackhole interpreter

Safepoint → Interpreter
Meta-tracer states

Interpreter → Tracer → Machine code

Interpreter
Tracer
Machine code
Blackhole interpreter

Safepoint
Guard failure

Hot
Compile
Meta-tracer states

Interpreter → Tracer → Machine code → Blackhole interpreter → Interpreter

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

Interpreter -> Tracer -> Machine code

1x

Safepoint

Guard failure
Meta-tracer performance (now)

Interpreter

Tracer

Machine code

Blackhole interpreter

1x

0.1x

Hot

Compile

Safepoint

Guard failure

HTTP://SOFT-DEV.ORG/
Meta-tracer performance (now)

Interpreter → Tracer → Machine code

1x → 200x → 0.1x

Safepoint → Guard failure

Hot → Compile
Meta-tracer performance (our aim)

Interpreter → Tracer → Machine code

1x → Hot

Tracer → Blackhole interpreter

2x

Blackhole interpreter → Guard failure

0.1x
References

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
How long to run things for (0.8)
How long to run things for (0.8)
<table>
<thead>
<tr>
<th>Class</th>
<th>steady iter (#)</th>
<th>steady iter (s)</th>
<th>steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarytrees</td>
<td>~ (27l, 2f, 1w)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fannkuch_redux</td>
<td>~ (26l, 4w)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fasta</td>
<td></td>
<td>4.0 (3.034.4)</td>
<td>0.75 (0.535, 5.873)</td>
</tr>
<tr>
<td>nbody</td>
<td>~ (5.0, 7.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>richards</td>
<td></td>
<td>2.0 (2.0, 35.3)</td>
<td>0.95 (0.879, 9.745)</td>
</tr>
<tr>
<td>spectralnorm</td>
<td>~ (2.0, 94.6)</td>
<td>14.0 (2.0, 94.6)</td>
<td>13.60 (0.890, 98.737)</td>
</tr>
</tbody>
</table>

Diffing results (0.8 → 1.5)