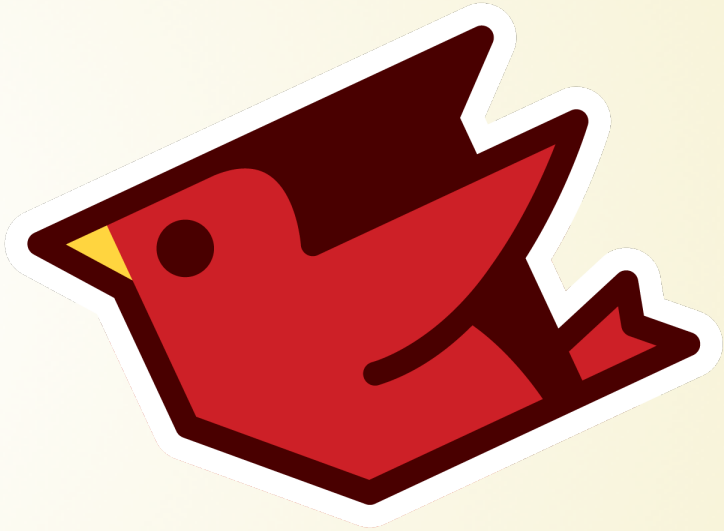
WHY YOU SHOULD BE USING JRUBY IN PRODUCTION



WHO AM I

IVO ANJO

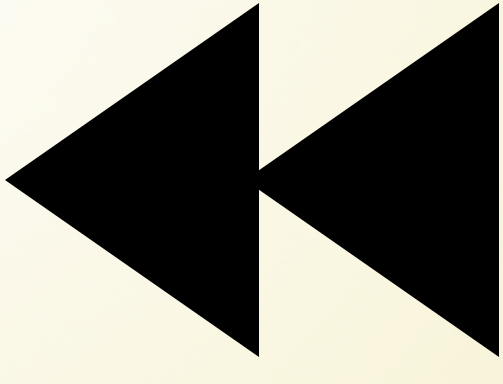
RUBY ❤

CONCURRENCY ❤ } JRUBY ❤

```
{talkdesk_ruby_ninja: self}
```

 **@knux**

JRUBY?



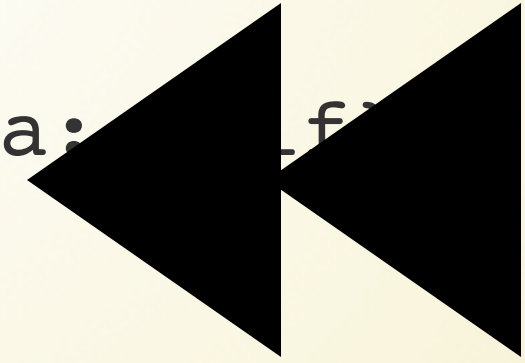
WHO AM I

IVO ANJO

RUBY ❤
CONCURRENCY ❤
} JRUBY ❤

{talkdesk_ruby_ninja:

@knux



WHY YOU SHOULD BE USING RUBY IN PRODUCTION

MINSWAN
MATZ IS NICE SO WE ARE NICE

EXPRESSIVE

DUCK TYPING

EVERYTHING IS AN OBJECT

EASY TO WRITE DSL

LESS BOILERPLATE

AWESOME PRY REPL

DSLs: Sonic PI uses ruby dsl to livecode music

AWESOME WEB LIBRARIES

RAILS

NOT RAILS: GRAPE, RODA, HANAMI

AWESOME TESTING LIBRARIES

RSPEC

CUCUMBER

AWESOME * LIBRARIES

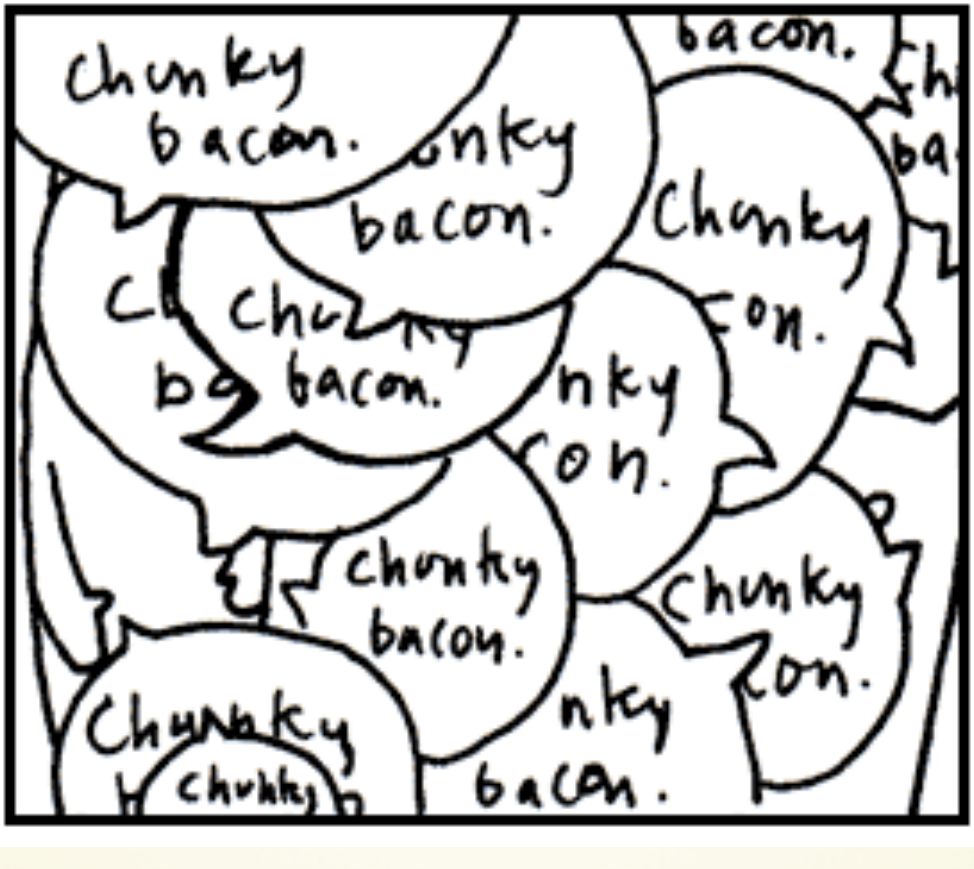
DATA STORES

APIS

* SERVER

* PROTOCOL

“THERE'S A GEM FOR THAT”


chunky bacon.
onky bacon.
bacon.
Chunky
on.
nky on.
chunky bacon.
chunky bacon.
nky bacon.
chunky

SO... JRUBY?

RUBY ON TOP OF THE JAVA VM

SO... JRUBY?

WHY #1: PERFORMANCE

JIT COMPILER

STATE-OF-THE-ART GC

GC:
* G1GC
* Azul Pauseless
* Parallel+concurrent

SO... JRUBY?

WHY #2: MASSIVELY PARALLEL

UNLEASH THE THREADS
...AND THE ASYNC WORK

SO... JRUBY?

WHY #3: TROUBLESHOOTING

USE AWESOME JAVA TOOLING

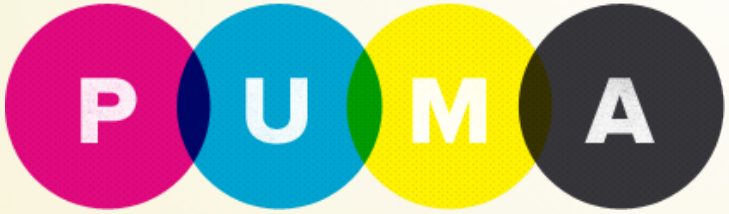
- PROFILERS
- MONITORING
- DEBUGGERS

} IN PRODUCTION!

SO... JRUBY?

WHY #4: EASY TO DEPLOY

RUBY TOOLING OR JAVA TOOLING



SO... JRUBY?

WHY #4½: EASY TO DEPLOY

DEPLOY AS RUBY SOURCES
OR DEPLOY AS SINGLE .JAR/.WAR

HOW HARD IS IT TO START USING?

```
$ RVM INSTALL JRUBY-9.1.7.0
$ RVM USE JRUBY-9.1.7.0
$ GEM INSTALL BUNDLER
$ BUNDLE INSTALL
```

SPENT A FEW MINUTES PORTING TALKDESK'S RAILS MONOLITH

```
knuckles@maruchan:~/talkdesk/main$ git diff --stat
 Gemfile          | 20 ++++++++++----------
 Gemfile.lock     | 24 ++++++++++++++++++++++---
 lib/talkdesk.rb | 18 ++++++++++++----
 3 files changed, 45 insertions(+), 17 deletions(-)
```

```
1635 examples, 19 failures, 41 pending
```

SPENT A FEW MINUTES PORTING TALKDESK'S RAILS MONOLITH

```
knuckles@maruchan:~/talkdesk/main$ git diff --stat
 Gemfile          | 20 ++++++++++----------
 Gemfile.lock     | 24 ++++++++++++++++++++++---
 lib/talkdesk.rb | 18 ++++++++++++++----
 3 files changed, 45 insertions(+), 17 deletions(-)
```

AUTHENTICATE, PLACE & LIST CALLS, ACCESS SETTINGS, ETC.

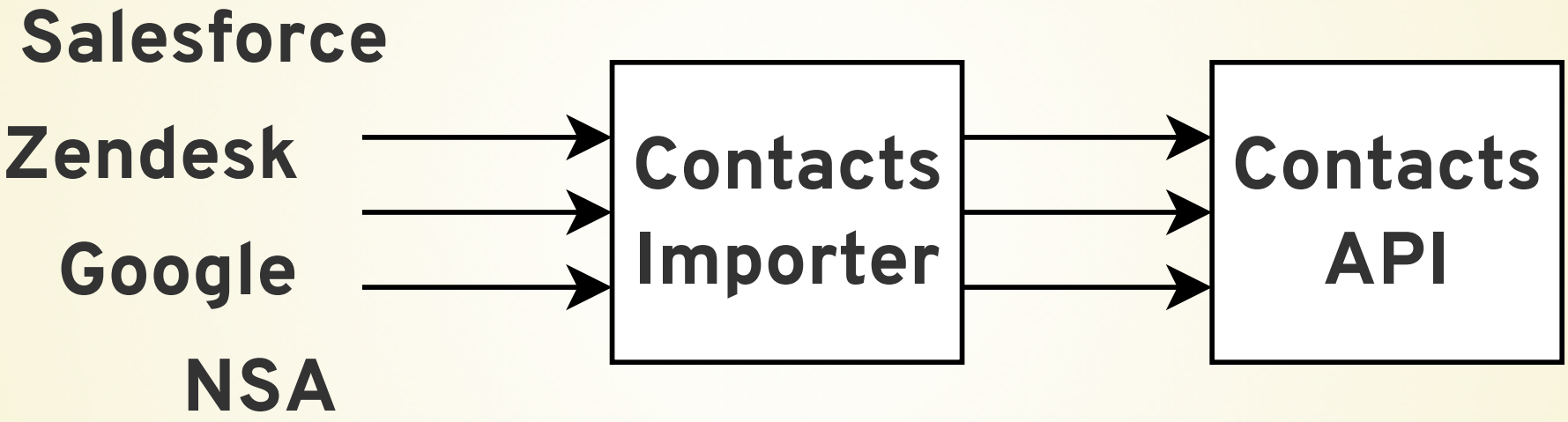
GOING COLD-TURKEY IS A BIG STEP

I STILL RECOMMEND YOU CAREFULLY VALIDATE YOUR APP :)

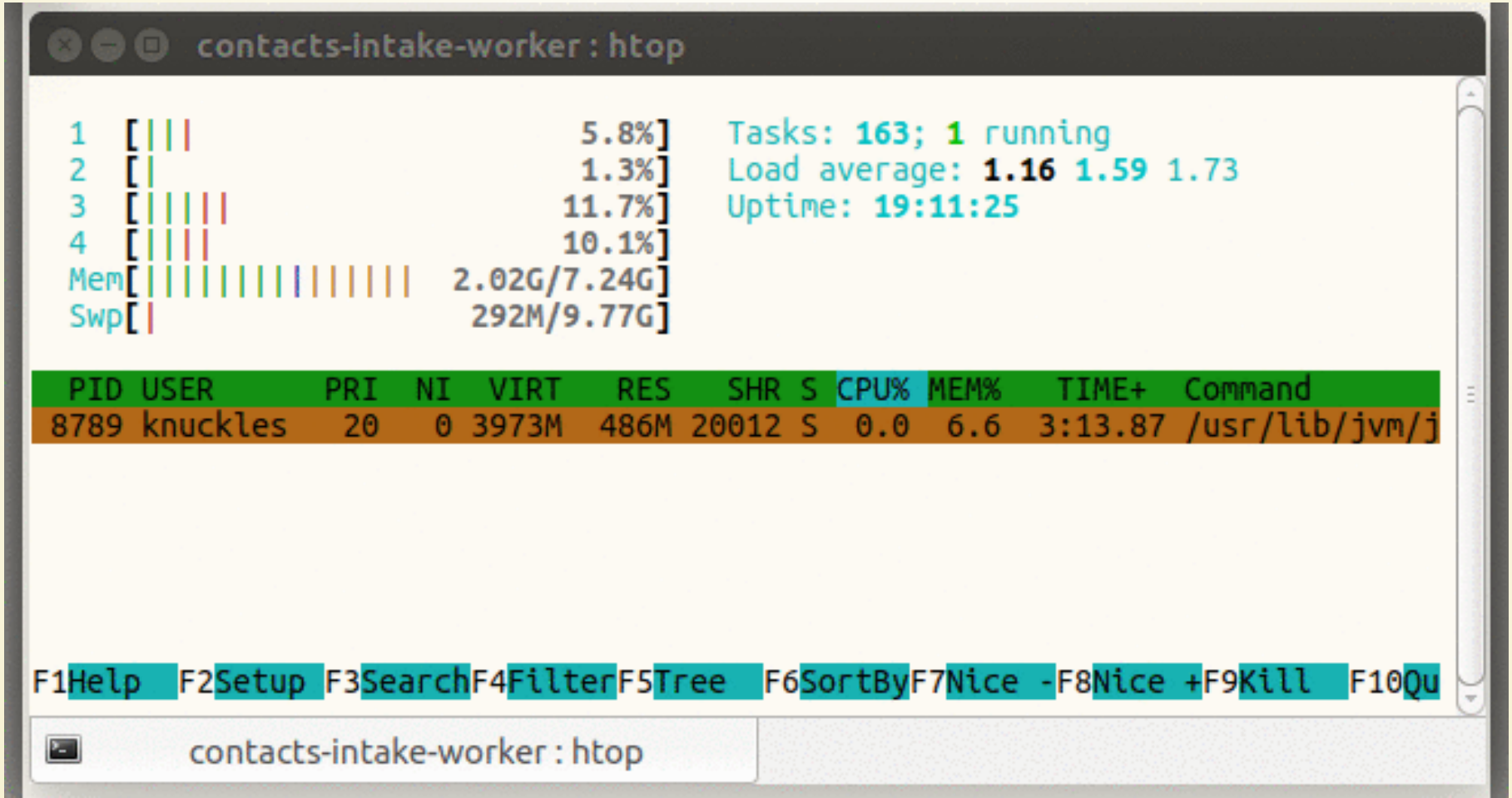
“THREAD CAREFULLY”?

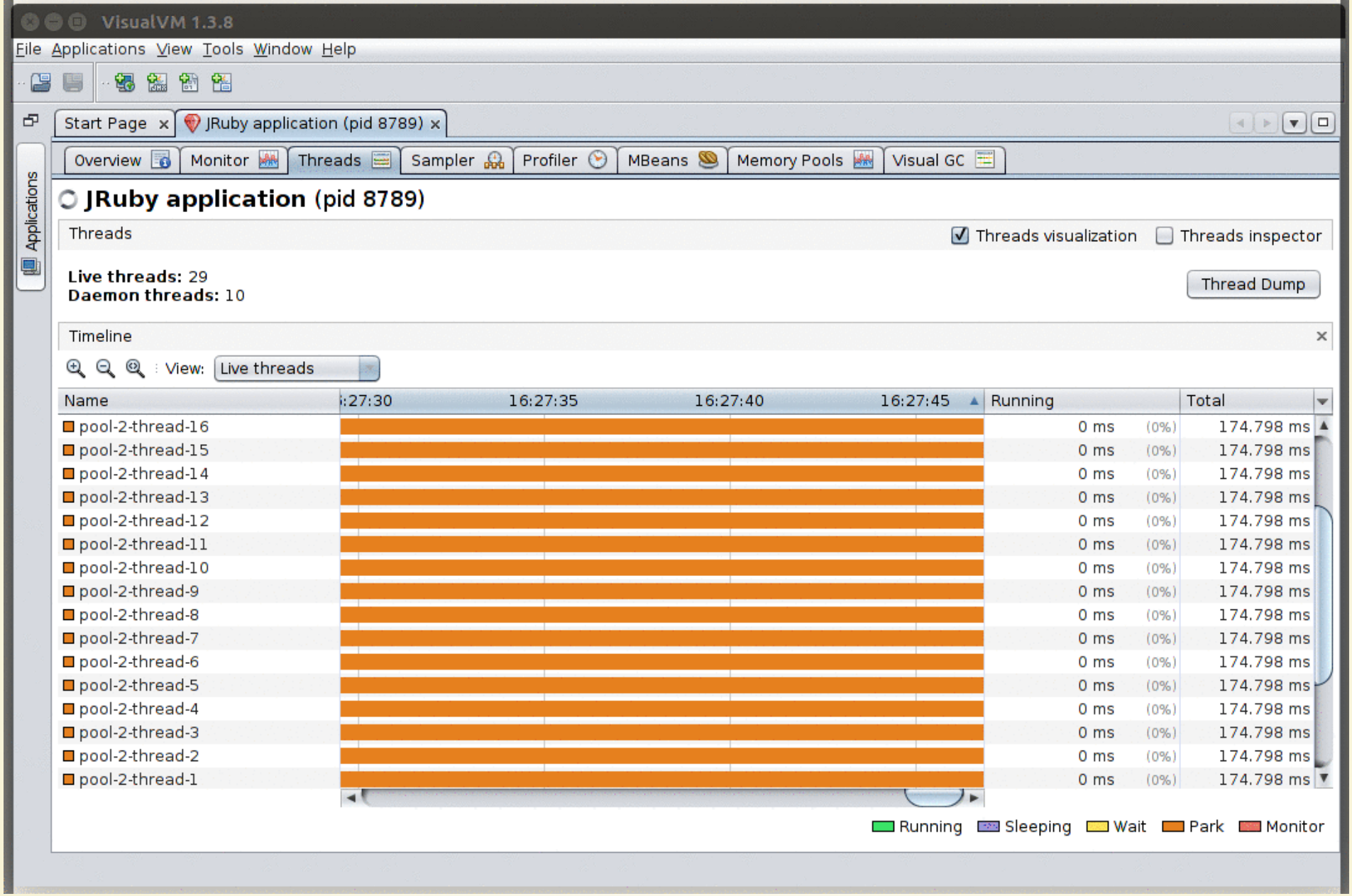
TALES FROM THE TRENCHES

CONTACTS IMPORTER



V0: UNLEASH THE THREADS





LOAD UP VISUALVM

GOOGLING AROUND

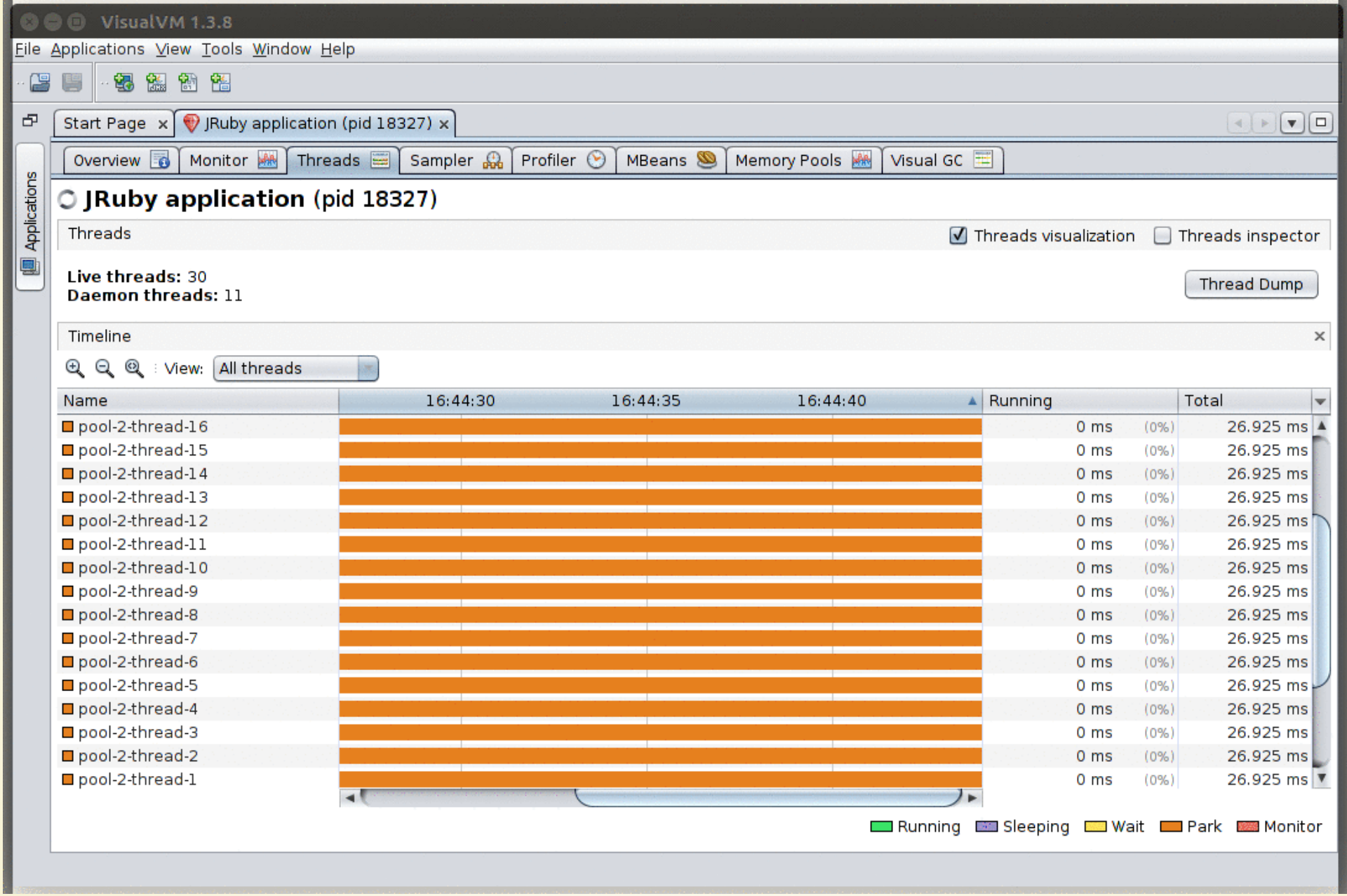
Unable to get multi-threading working #76

Closed **lardcanoe** opened this issue on 25 Mar 2015 · 3 comments

michaelklishin commented on 25 Mar 2015 Member +😀

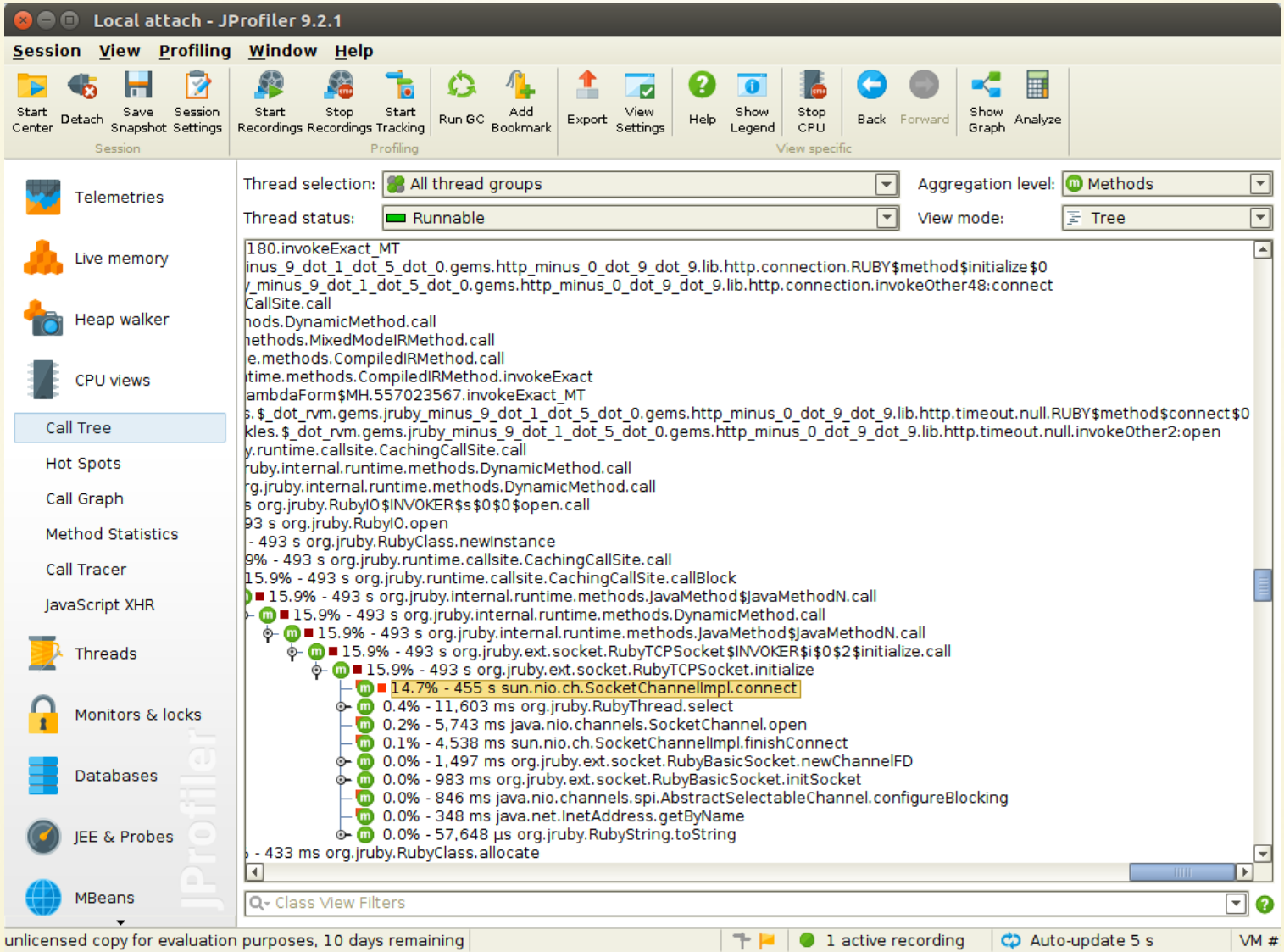
March Hare passes the executor to the Java client. It then uses batched execution of server-sent methods (e.g. `basic.deliver`) in the executor. That currently means 1 thread per channel because we need to guarantee ordering.

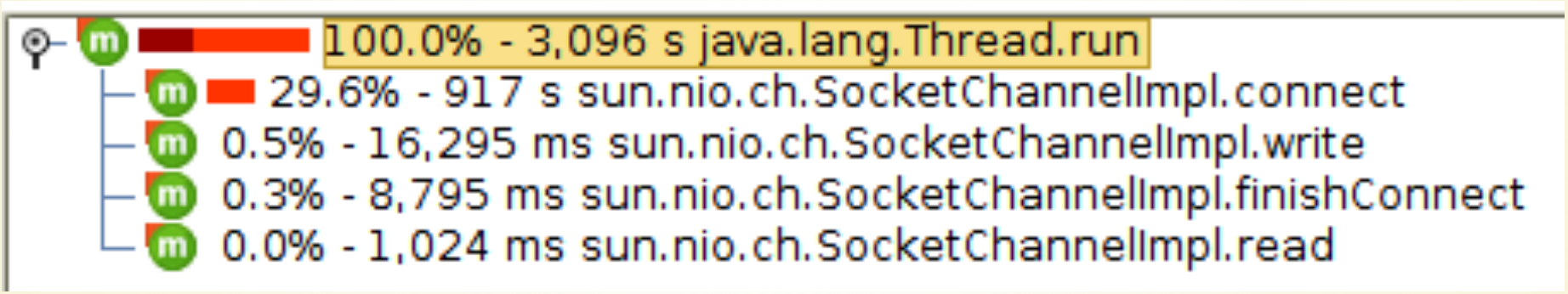
OOPS, WE WERE USING THE GEM WRONG


VisualVM 1.3.8
File Applications View Tools Window Help
Start Page
JRuby application (pid 18327)
Overview Monitor Threads Sampler Profiler MBeans Memory Pools Visual GC
Applications
JRuby application (pid 18327)
Threads
Threads visualization
Threads inspector
Live threads: 30
Daemon threads: 11
Thread Dump
Timeline
View: All threads
Name
16:44:30
16:44:35
16:44:40
Running
Total
pool-2-thread-16 0 ms (0%) 26.925 ms
pool-2-thread-15 0 ms (0%) 26.925 ms
pool-2-thread-14 0 ms (0%) 26.925 ms
pool-2-thread-13 0 ms (0%) 26.925 ms
pool-2-thread-12 0 ms (0%) 26.925 ms
pool-2-thread-11 0 ms (0%) 26.925 ms
pool-2-thread-10 0 ms (0%) 26.925 ms
pool-2-thread-9 0 ms (0%) 26.925 ms
pool-2-thread-8 0 ms (0%) 26.925 ms
pool-2-thread-7 0 ms (0%) 26.925 ms
pool-2-thread-6 0 ms (0%) 26.925 ms
pool-2-thread-5 0 ms (0%) 26.925 ms
pool-2-thread-4 0 ms (0%) 26.925 ms
pool-2-thread-3 0 ms (0%) 26.925 ms
pool-2-thread-2 0 ms (0%) 26.925 ms
pool-2-thread-1 0 ms (0%) 26.925 ms
Running Sleeping Wait Park Monitor

👍 FIXED

V1: CPU-LIMITED? WAT?





THIS DOESN'T SEEM RIGHT...

pezra / **hal-client**

Watch 4

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs

Reuse connections for requests with same headers #58

Merged **pezra** merged 3 commits into `pezra:master` from `ivoanjo:fix-connection-reuse` on 8 Nov 2016

Conversation 3 | Commits 3 | Files changed 4

ivoanjo commented on 3 Nov 2016 — Contributor

The #client_for_get and #client_for_post methods attempted to reuse HTTP::Client instances whenever no override headers were received (when they were nil).

Unfortunately, both always received as override headers the result of #auth_headers, which is always at least at empty hash.

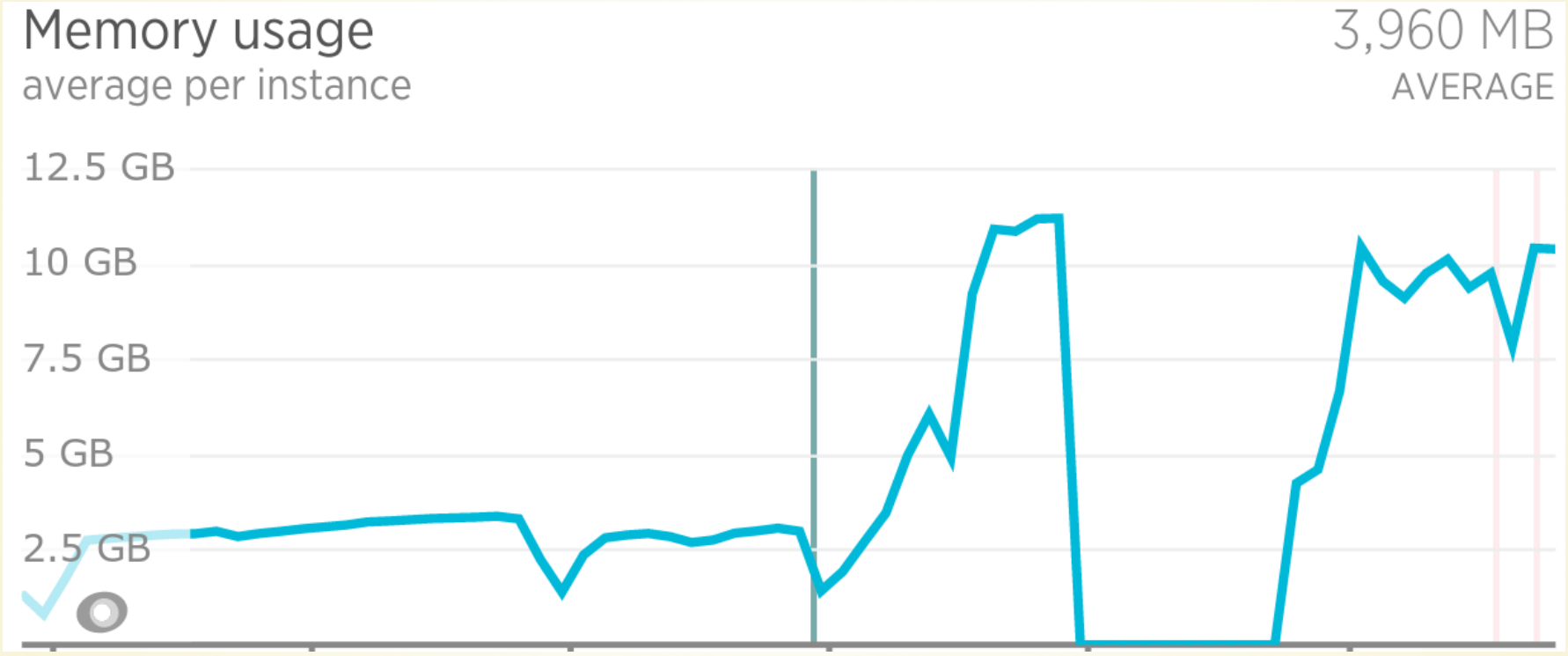
This meant that HTTP::Client instances were never actually reused, and instead a new one was created on every request, even if the gem users never actually specified custom headers for requests.

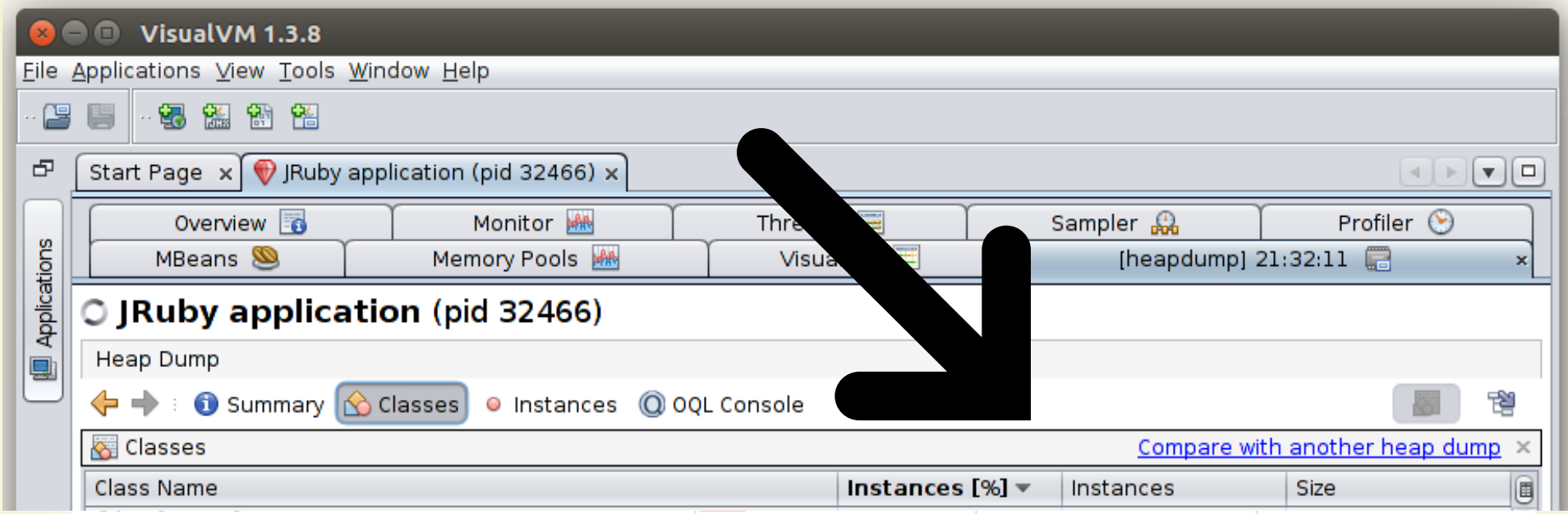
Furthermore, even if the above scheme did not have a bug, we would still always create a new HTTP::Client instance for every request with custom headers, which is rather inefficient.

Rev
No
Ass
No
Lab
Nor
Pro
Nor
Mile

👍 FIX MERGED UPSTREAM!

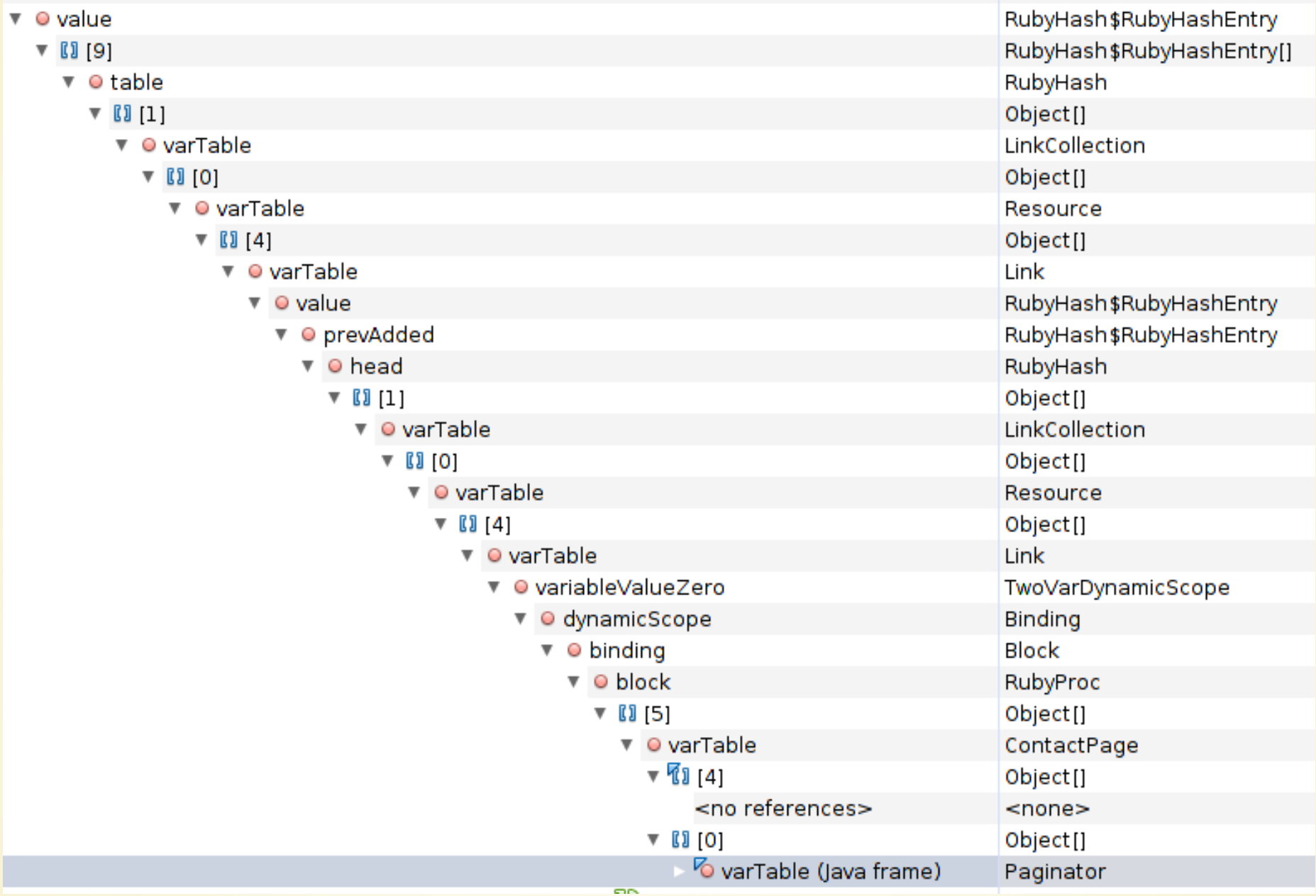
V2: MEMORY LEAKS :(



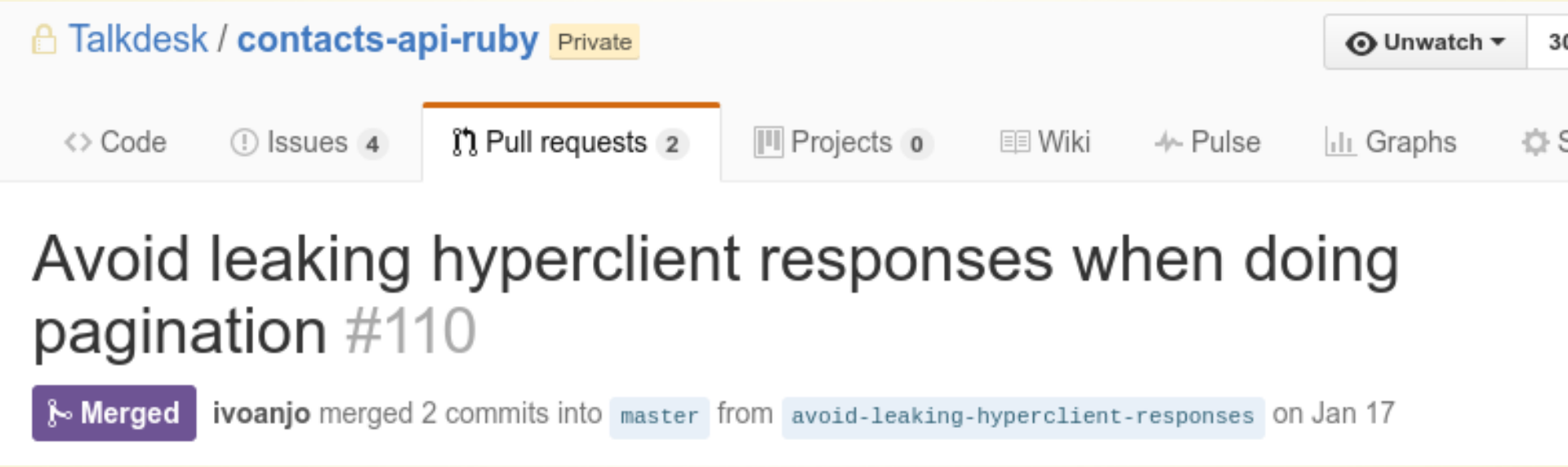


Class Name	Instances [%]	Instances	Size
org.jruby.**RubyHash$RubyHashEntry[]**		846.338 (4,3%)	94.758.424 (8,4%)
org.jruby.**RubyHash**		846.336 (4,3%)	67.706.880 (6%)
org.joni.**StackEntry**		533.517 (2,7%)	19.206.612 (1,7%)
org.jruby.runtime.builtin.**IRubyObject[]**		354.629 (1,8%)	54.150.632 (4,8%)
org.jruby.**RubyArray**		354.023 (1,8%)	20.179.311 (1,8%)
java.lang.**Object[]**		291.114 (1,5%)	17.880.888 (1,6%)
char[]		128.598 (0,7%)	15.779.694 (1,4%)
java.lang.**String**		127.537 (0,6%)	3.571.036 (0,3%)
java.util.**HashMap$Node**		117.267 (0,6%)	5.159.748 (0,5%)
java.util.concurrent.**ConcurrentHashMap$Node**		75.839 (0,4%)	3.336.916 (0,3%)
java.util.**HashMap**		43.058 (0,2%)	2.755.712 (0,2%)
rubyobj.Hyperclient.**Link**		42.375 (0,2%)	1.695.000 (0,2%)
rubyobj.Hyperclient.**Attributes**		40.700 (0,2%)	1.628.000 (0,1%)
rubyobj.Hyperclient.**ResourceCollection**		40.700 (0,2%)	1.628.000 (0,1%)
rubyobj.Hyperclient.**LinkCollection**		40.700 (0,2%)	1.628.000 (0,1%)
rubyobj.Hyperclient.**Resource**		40.700 (0,2%)	1.628.000 (0,1%)

LOAD UP VISUALVM



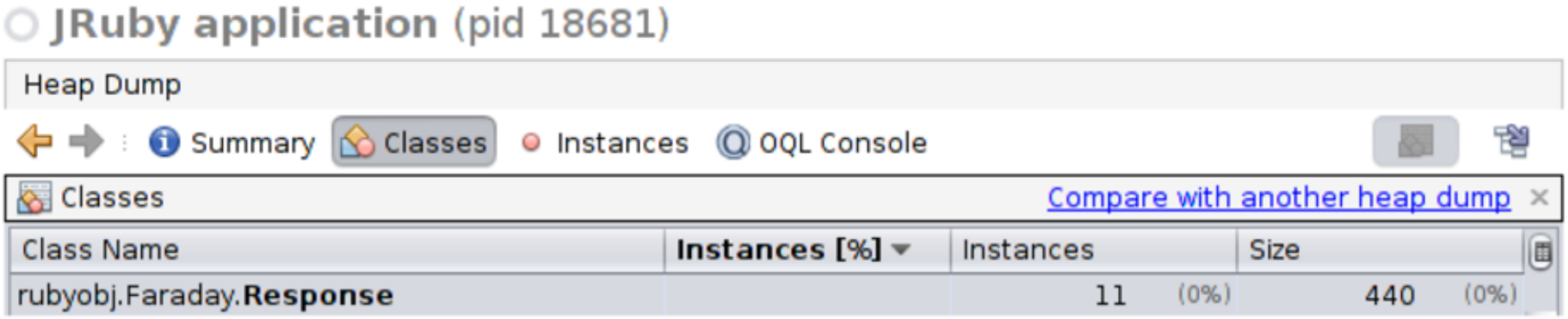
FOLLOW THE BREADCRUMBS


Talkdesk / contacts-api-ruby
Private
Unwatch
Code
Issues 4
Pull requests 2
Projects 0
Wiki
Pulse
Graphs
Avoid leaking hyperclient responses when doing pagination #110
Merged
ivoanjo merged 2 commits into master from avoid-leaking-hyperclient-responses on Jan 17

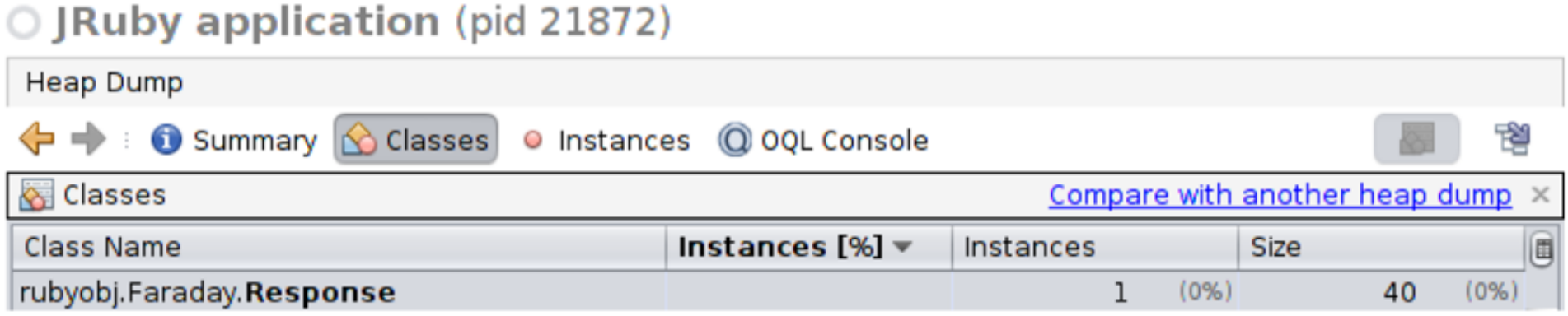
👍 FIXED

Obligatory VisualVM screenshots

- Before (after fetching some pages):

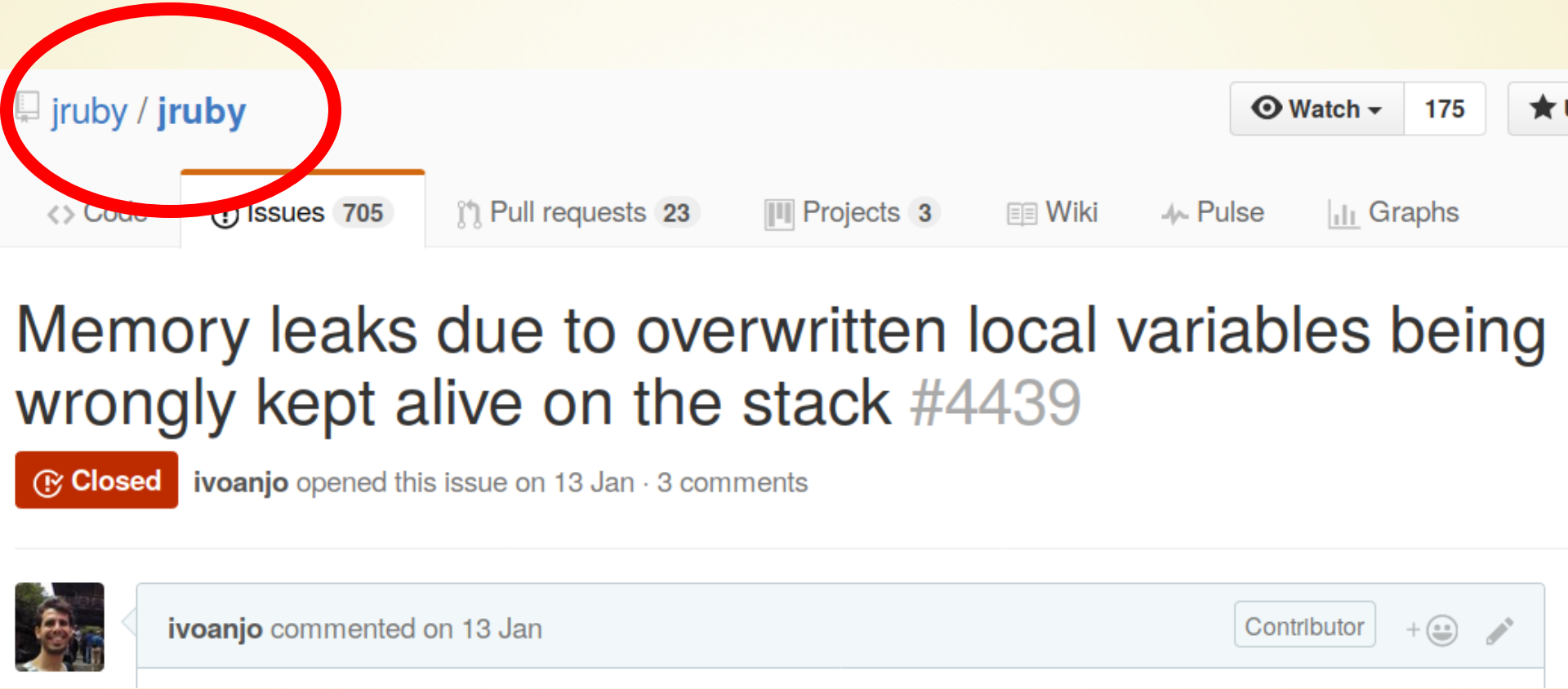


- After:



👍 FIXED

V2½: MEMORY LEAKS :(



Expected Behavior

Example code:

```
class Dummy
  def initialize
    @payload = Array.new(100000)
  end

  def next
    @next = Dummy.new
  end
end

class Test
  def self.test
    dummy = Dummy.new
    iterations = 0

    while true
      dummy = dummy.next
      iterations += 1

      puts "ran iteration #{iterations}"

      GC.start if iterations % 10 == 0
      sleep if iterations == 200 && ENV['PAUSE_AFTER_ITERATIONS'] == '1'
    end
  end
end

Test.test
```

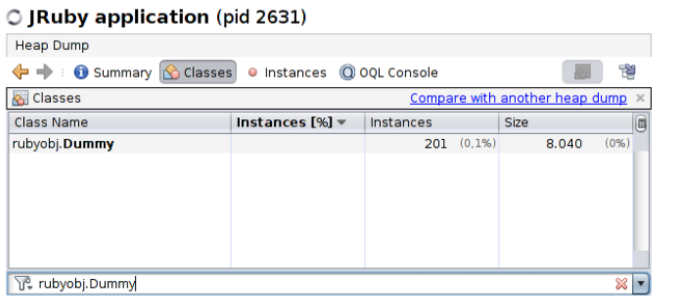
On MRI this code seems to behave correctly (e.g. it does not leak memory) and I can get up to hundreds of thousands of iterations with < 16MB memory being used.

Actual Behavior

Running with jruby:

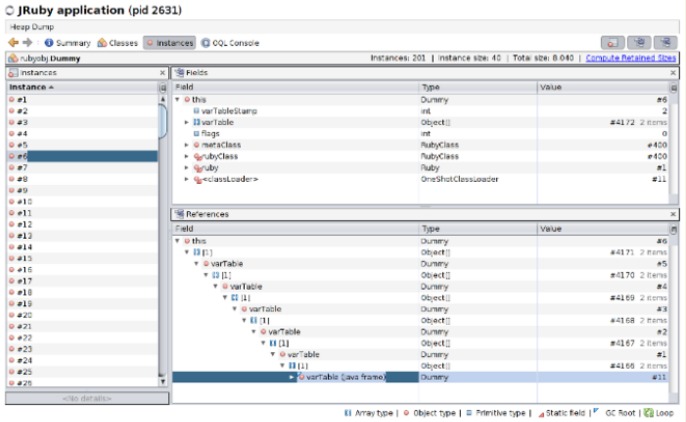
```
$ jruby -J-Xmx200m leak_testcase.rb
ran iteration 1
ran iteration 2
...
ran iteration 466
ran iteration 467
Error: Your application used more memory than the safety cap of 200M.
Specify -J-Xmx####M to increase it (#### = cap size in MB).
Specify -w for full java.lang.OutOfMemoryError: Java heap space stack trace
```

By pausing after a few iterations and looking at a memory dump (with `PAUSE_AFTER_ITERATIONS=1 jruby -J-Xmx200m leak_testcase.rb`), we can see the following:

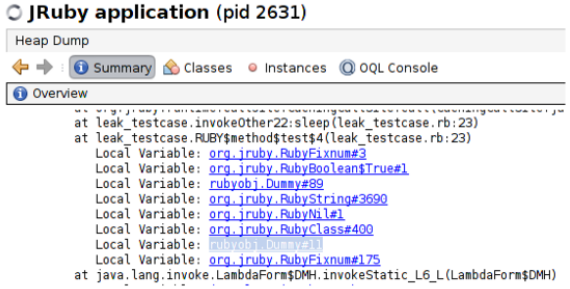


E.g. all instances of `Dummy` are still alive, whereas looking at the code we are creating a linked list but we never keep references to previous nodes so we should only have a single instance visible in memory.

Looking at one of the instances, we see that they are being kept alive due to a reference stored on the stack:

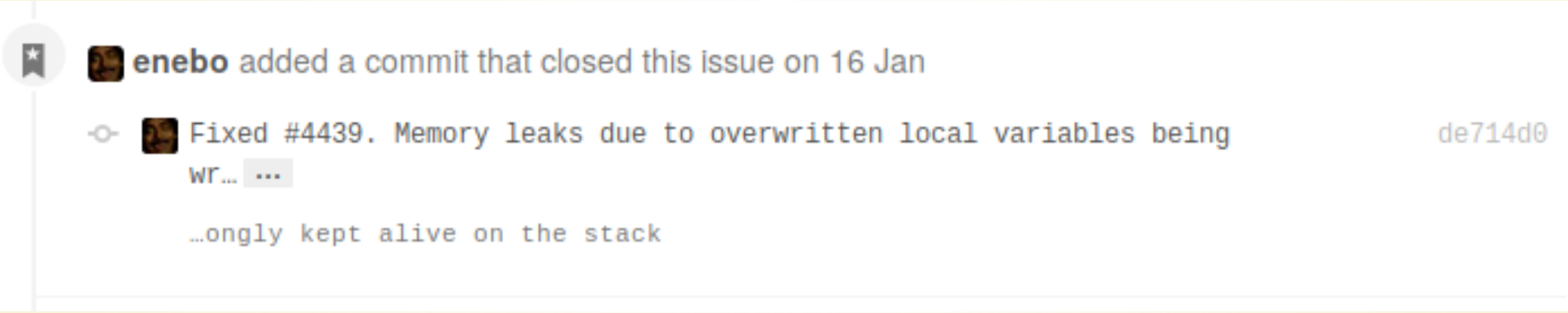


and by asking for a list of stack references we see that there are **two** `Dummy` instances being referred to from the stack, not one as expected.



It seems like the code generating is leaving the first dummy instance referenced on the stack and thus causing the memory leak as the whole list will be kept in memory, rather than just the last element.

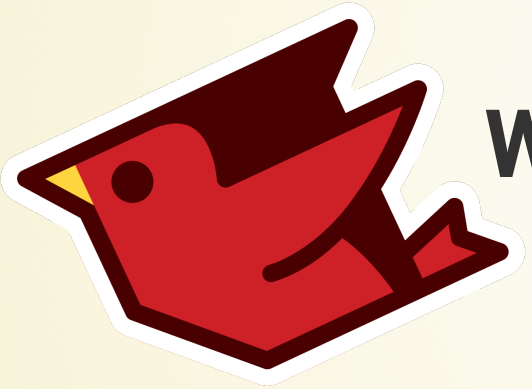
We (**@Talkdesk**) hit this in production as we were iterating a large collection using `hyperclient` and thus were hitting memory limits even though we were doing it a slice at a time.


enebo added a commit that closed this issue on 16 Jan
Fixed #4439. Memory leaks due to overwritten local variables being wr…
de714d0
…ongly kept alive on the stack

3 DAYS LATER! :)

CONTACTS IMPORTER

HUGE PRODUCTIVITY BOOST

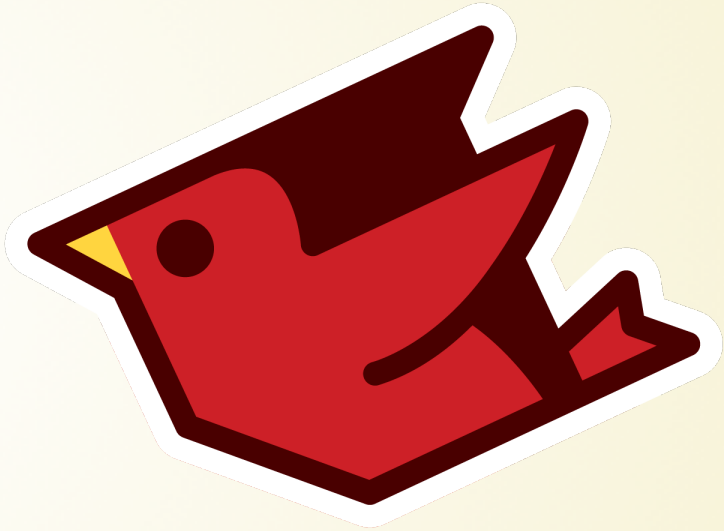


WIZARD-LEVEL DEBUGGING MADE EASY

HIT DOUBLE OUR PERFORMANCE TARGET ON A SINGLE INSTANCE

JRUBY

101



IN DEVELOPMENT

```
export JRUBY_OPTS=--dev

bundle install --binstubs
```

IN DEVELOPMENT

```
gem 'pry-byebug',
  platforms: :ruby
gem 'pry-debugger-jruby',
  platforms: :jruby
```

```
export JRUBY_OPTS="--dev --
debug"
```

Note that --dev speeds up start-up at the cost of steady-state execution time, and --debug slows down execution time even more.

IN DEVELOPMENT

```
gem 'multi_json'
gem 'oj', platforms: :ruby
gem 'jrjackson', platforms: :jruby

if RUBY_PLATFORM == 'java'
  require 'jrjackson'
  MultiJson.use :jr_jackson
else
  require 'oj'
  MultiJson.use :oj
end
```

IN DEVELOPMENT

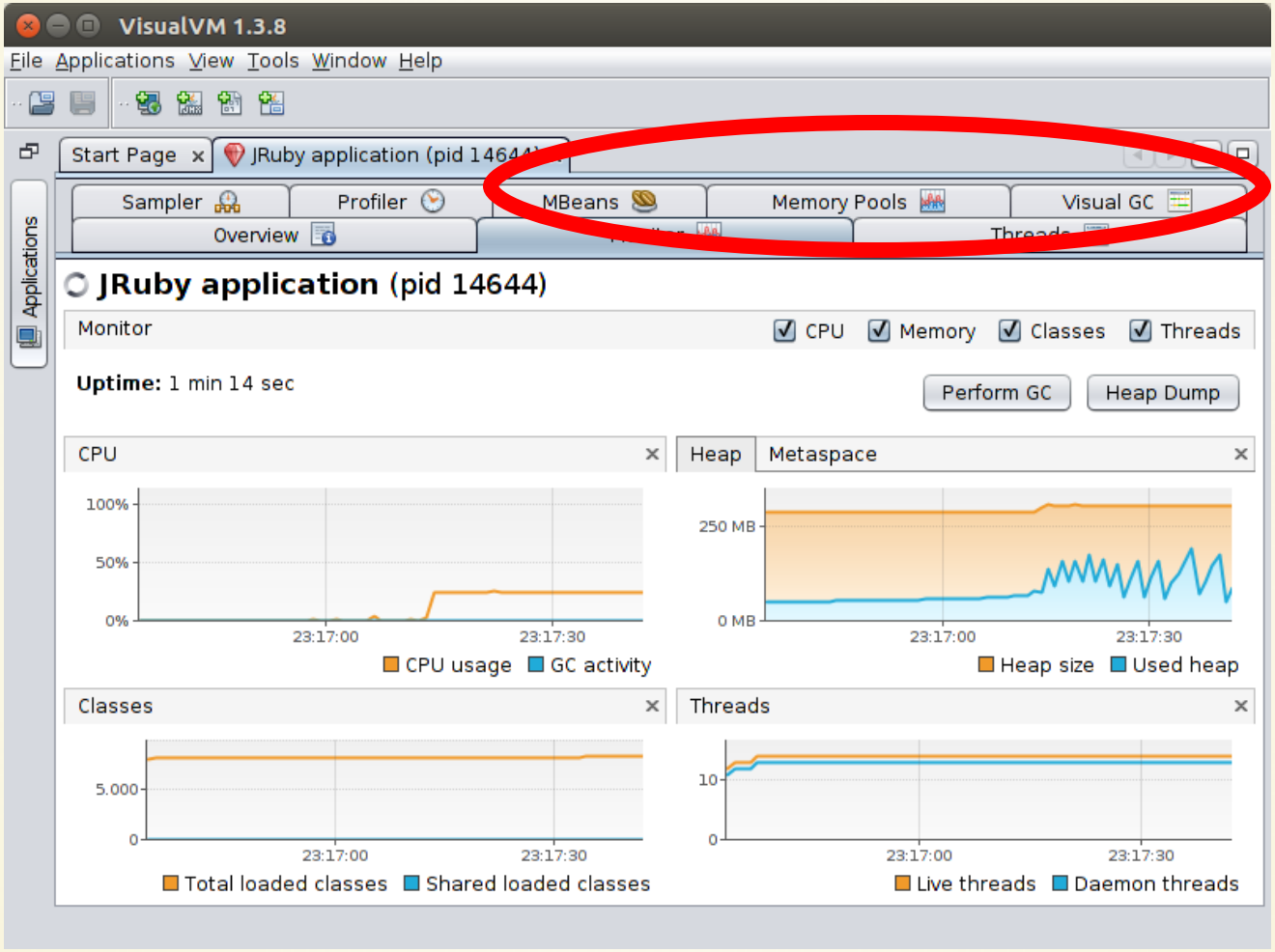
```
./Gemfile:24: [gems, warning] Found gem 'oj' which is
reported to have some issues:
Try gson, json or json_pure instead.
Try any one of the following JRuby-based servers:
Trinidad, Mizuno, Kirk or Puma (though make sure to use
the JRuby-native version of the gem).
./Gemfile:43: [gems, warning] Found gem 'bson_ext' which
is reported to have some issues:
bson_ext isn't used with JRuby. Instead, some native Java
extensions are bundled with the bson gem.
./app/controllers/api/user_controller.rb:25: [nonatomic,
warning] Non-local operator assignment is not guaranteed
to be atomic
```

JRUBY-LINT

IN DEVELOPMENT

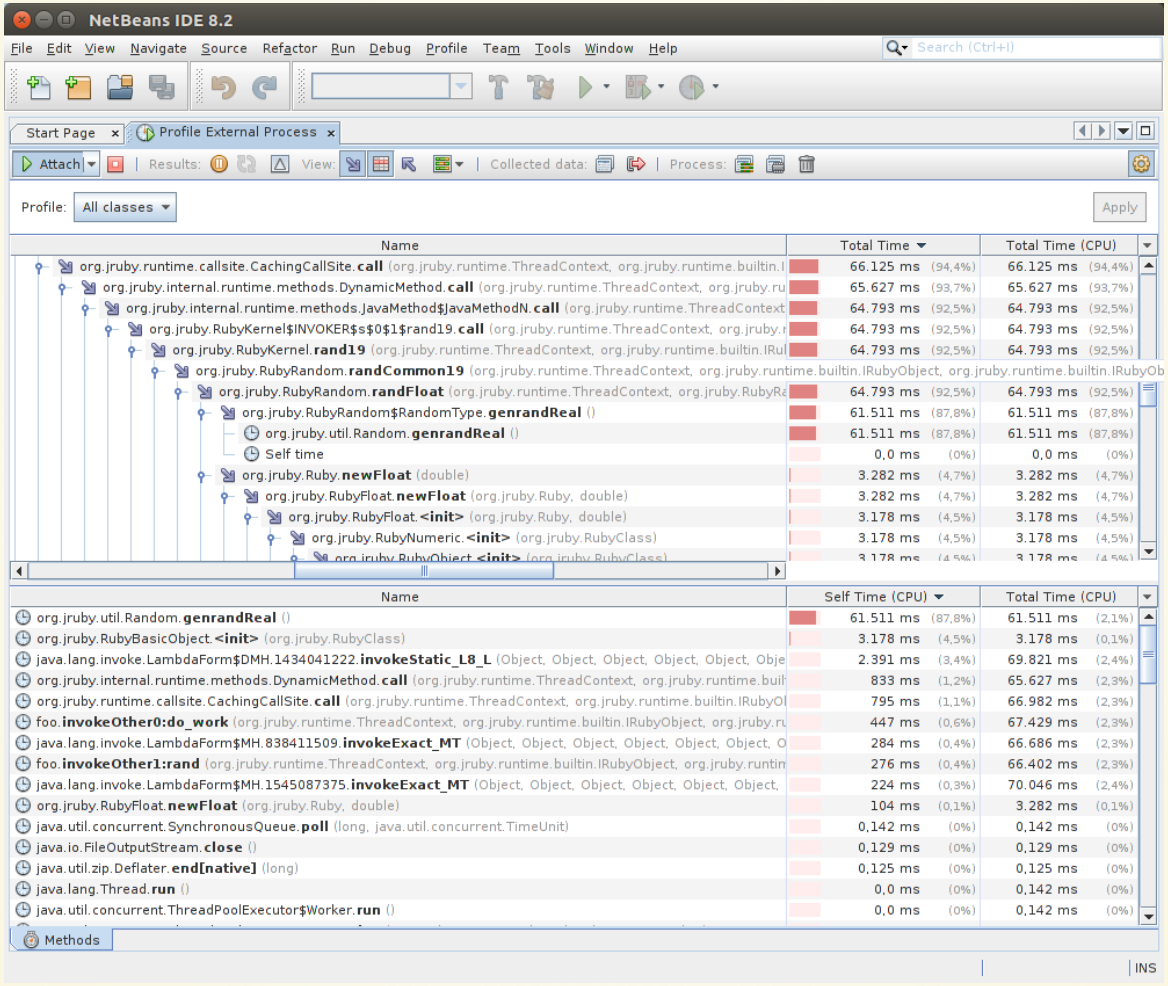
CONCURRENT-RUBY
IS YOUR FRIEND

IN DEVELOPMENT



VISUALVM

IN DEVELOPMENT



NETBEANS PROFILING

IN DEVELOPMENT

```
export JRUBY_OPTS="-Xreify.classes=true"
```

```
$ jmap -histo 17118 | grep rubyobj
  17:          9000         288000  rubyobj.Example.A
 337:            98           3136  rubyobj.Gem.Requirement
 435:            52           1664  rubyobj.Gem.Dependency
 470:            42           1344  rubyobj.OtherStuff.B
 477:            41           1312  rubyobj.Gem.Version
 762:            13            416  rubyobj.Gem.Specification
1103:             6            192  rubyobj.Gem.Platform
4060:             1             32  rubyobj.Gem.PathSupport
4061:             1             32  rubyobj.Gem.StubSpecification
4062:             1             32  rubyobj.Gem.StubSpecification.StubLine
4063:             1             32  rubyobj.Monitor
```

JMAP/JSTACK

Can even be used on heroku: https://devcenter.heroku.com/articles/java-memory-issues (don't forget to add the buildpack)

IN DEVELOPMENT

YOURKIT & JPROFILER

IN PRODUCTION

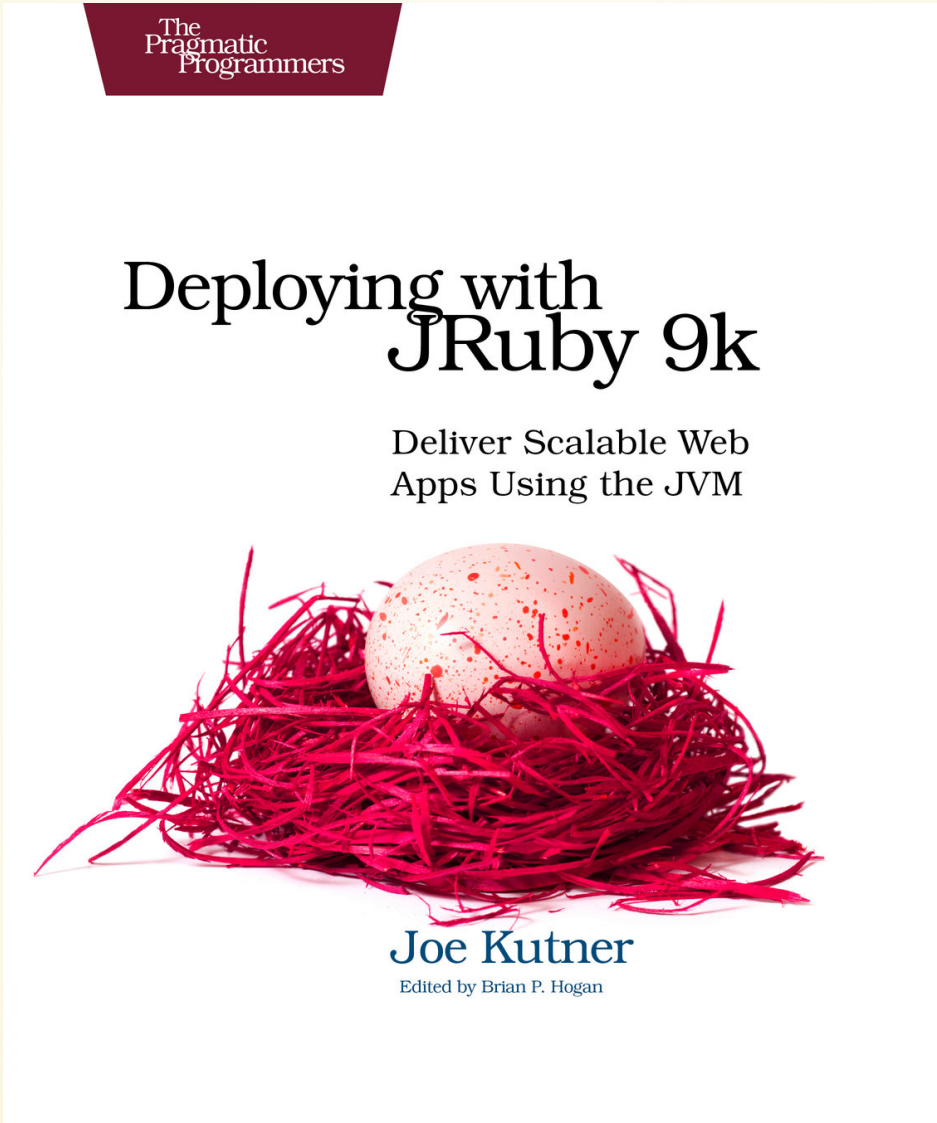
```
export JRUBY_OPTS=\
"-Xcompile.invokedynamic=true -
  Xmanagement.enabled=true"

export JAVA_OPTS=\
"-Xmx3g -XX:+PrintGCDetails"
```

IN PRODUCTION

```
export JAVA_OPTS=\
"-Xss512k -Xmx500m -
XX:ReservedCodeCacheSize=90m -
XX:InitialCodeCacheSize=90m -
XX:MaxMetaspaceSize=120m -
XX:+PrintGCDetails
-XX:+PrintCodeCache"
```

NEEDS SOME TUNING FOR SMALLER MACHINES :/

IN PRODUCTION



THANKS!

TWITTERS: @KNUX

BLOG: IVOANJO.ME