What Exactly do we Mean by *JIT Warmup*?






Edd Barrett, Carl Friedrich Bolz, Rebecca Killick (Lancaster),
Vincent Knight (Cardiff), Sarah Mount, Laurence Tratt

KING'S College LONDON

Software Development Team
April 20, 2016

Agenda

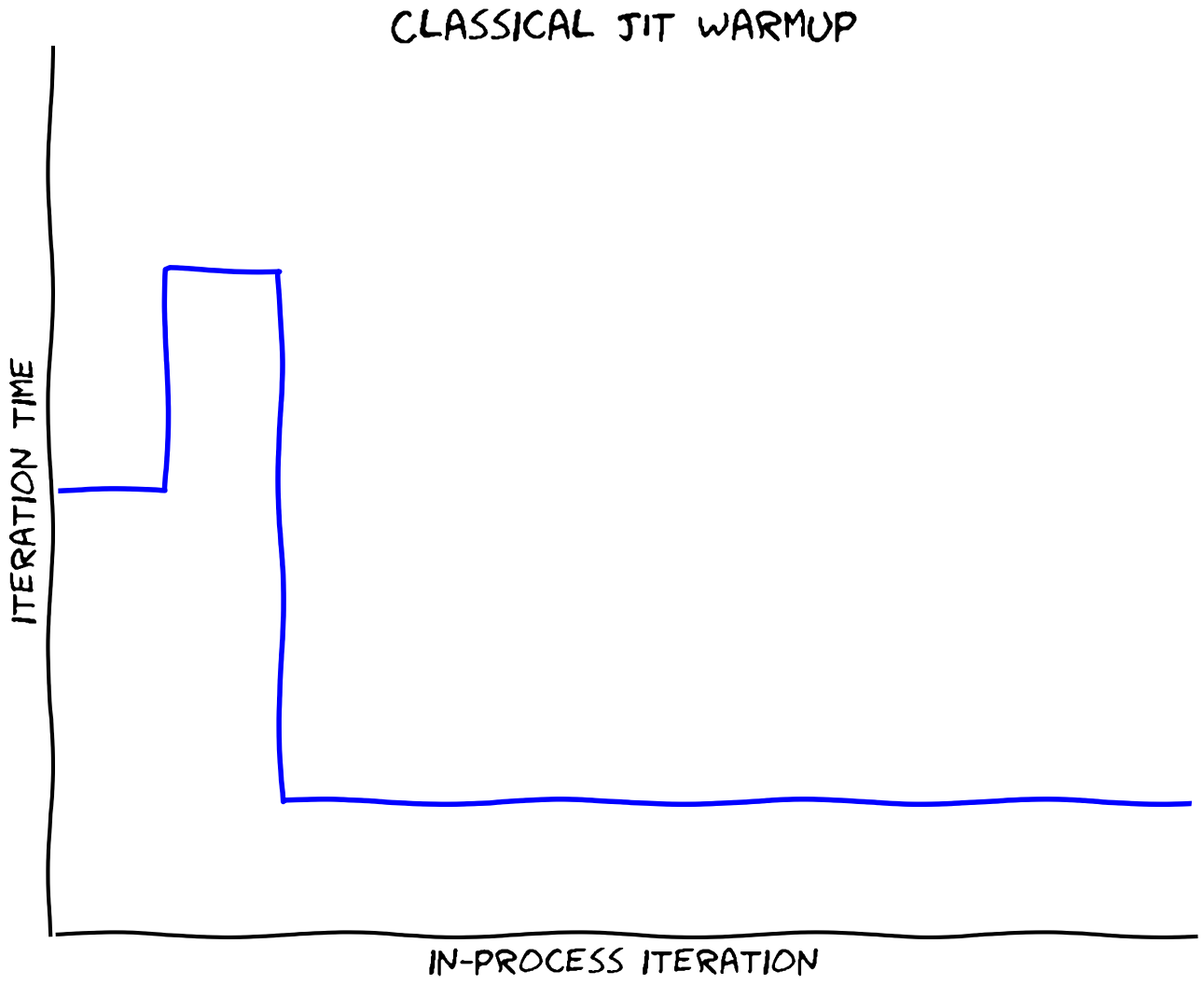
JIT Warmup Background

JIT Warmup Background

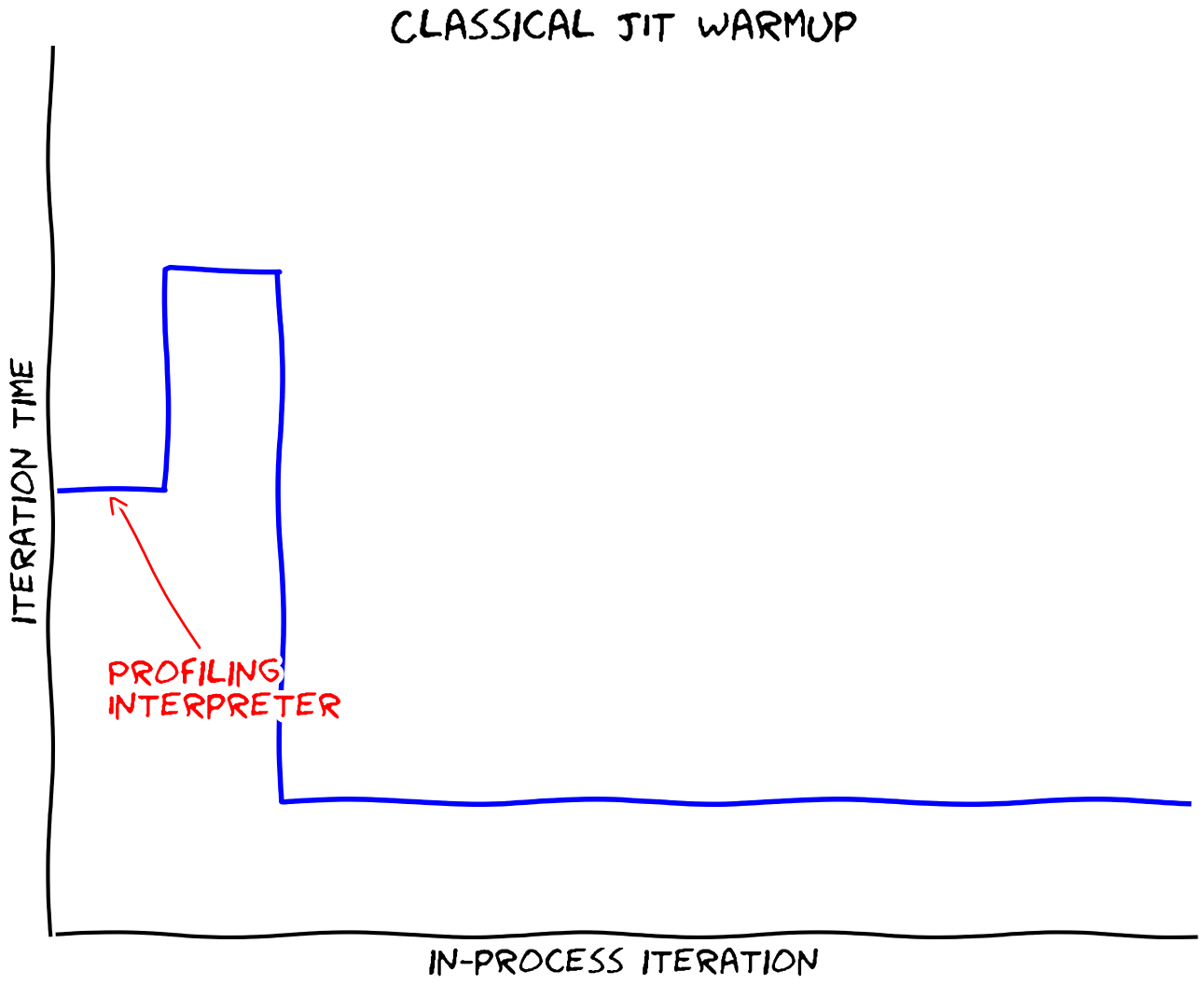
Informally:

Time taken for a JITted VM to reach peak performance

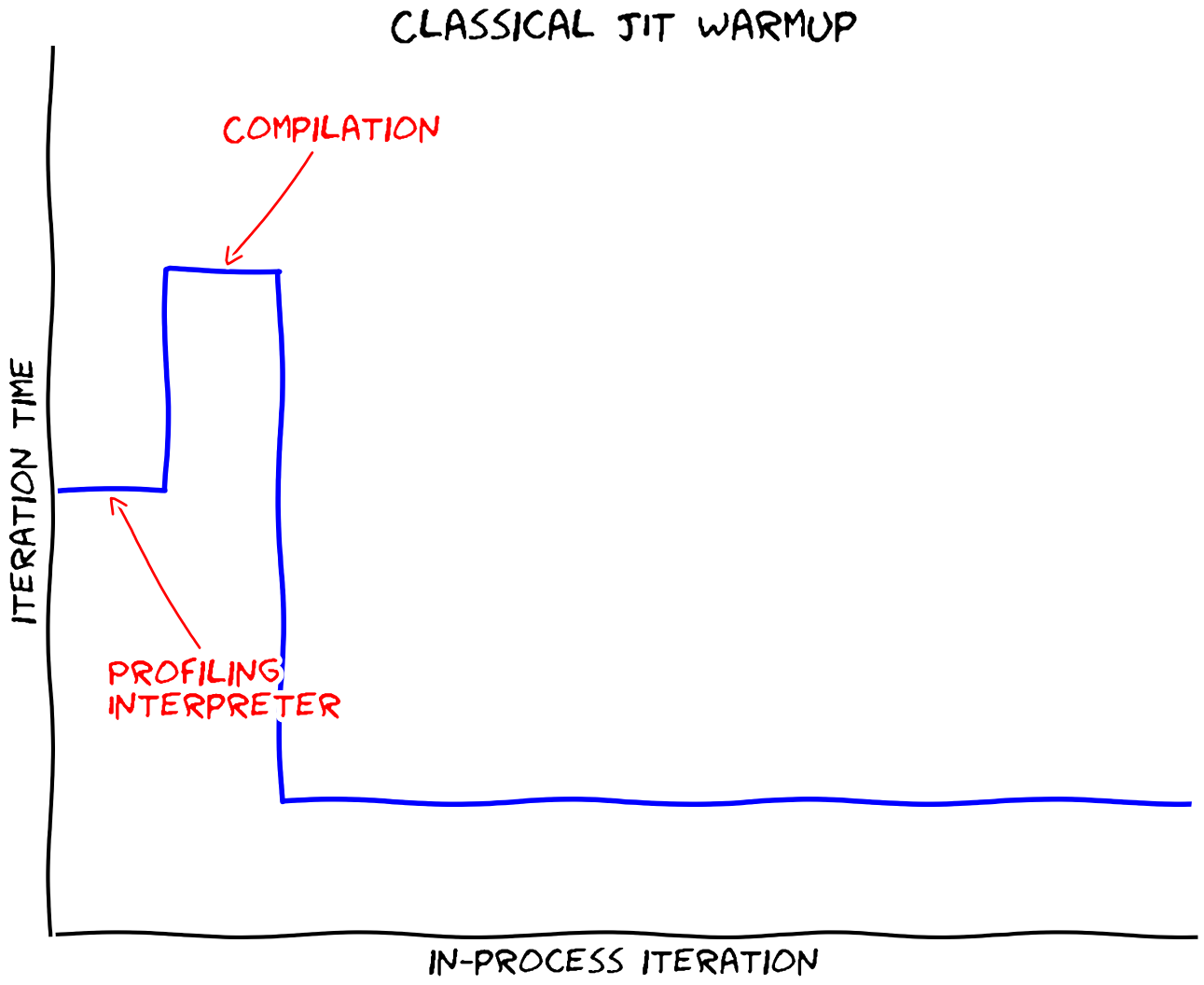
JIT Warmup Background



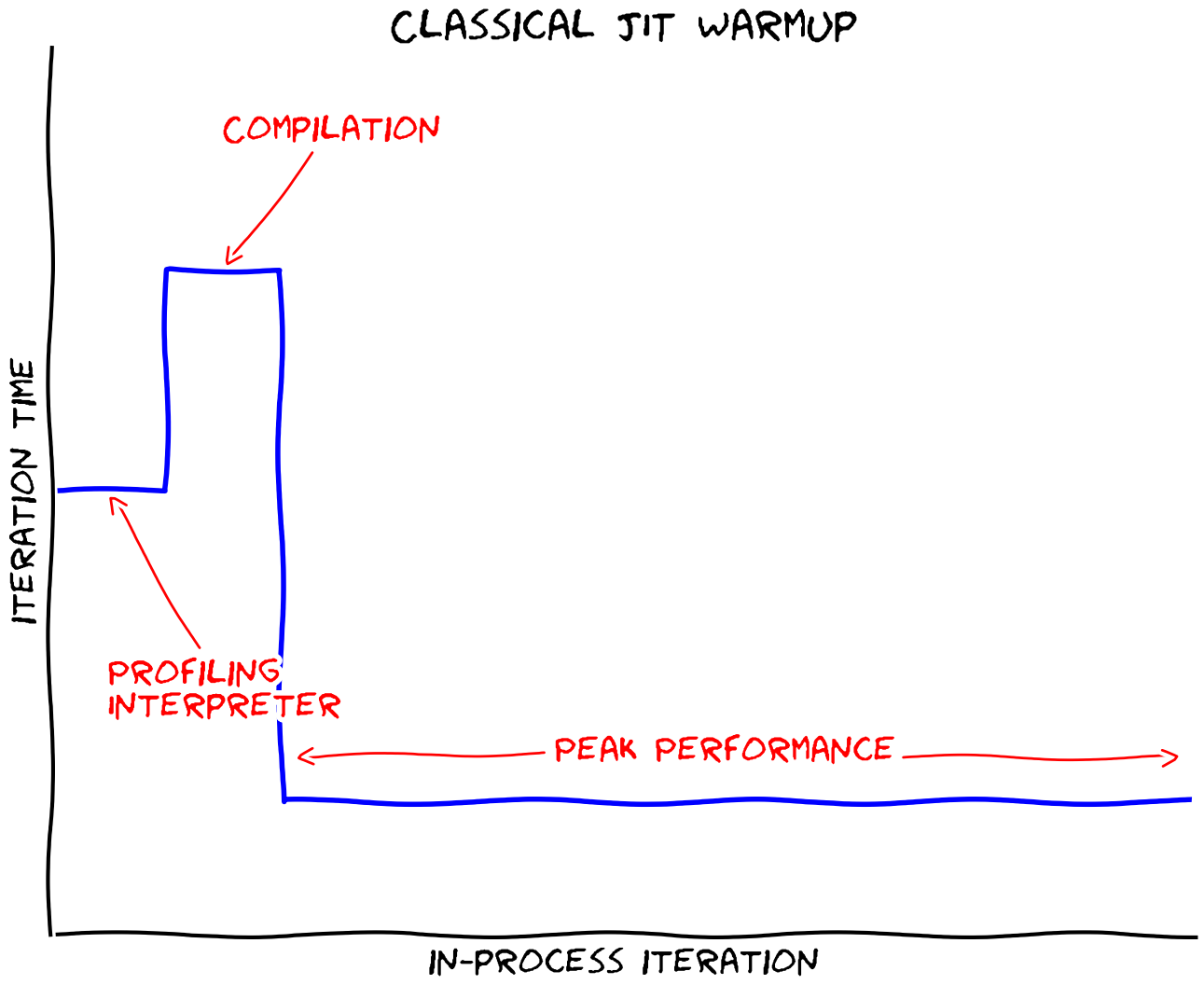
JIT Warmup Background



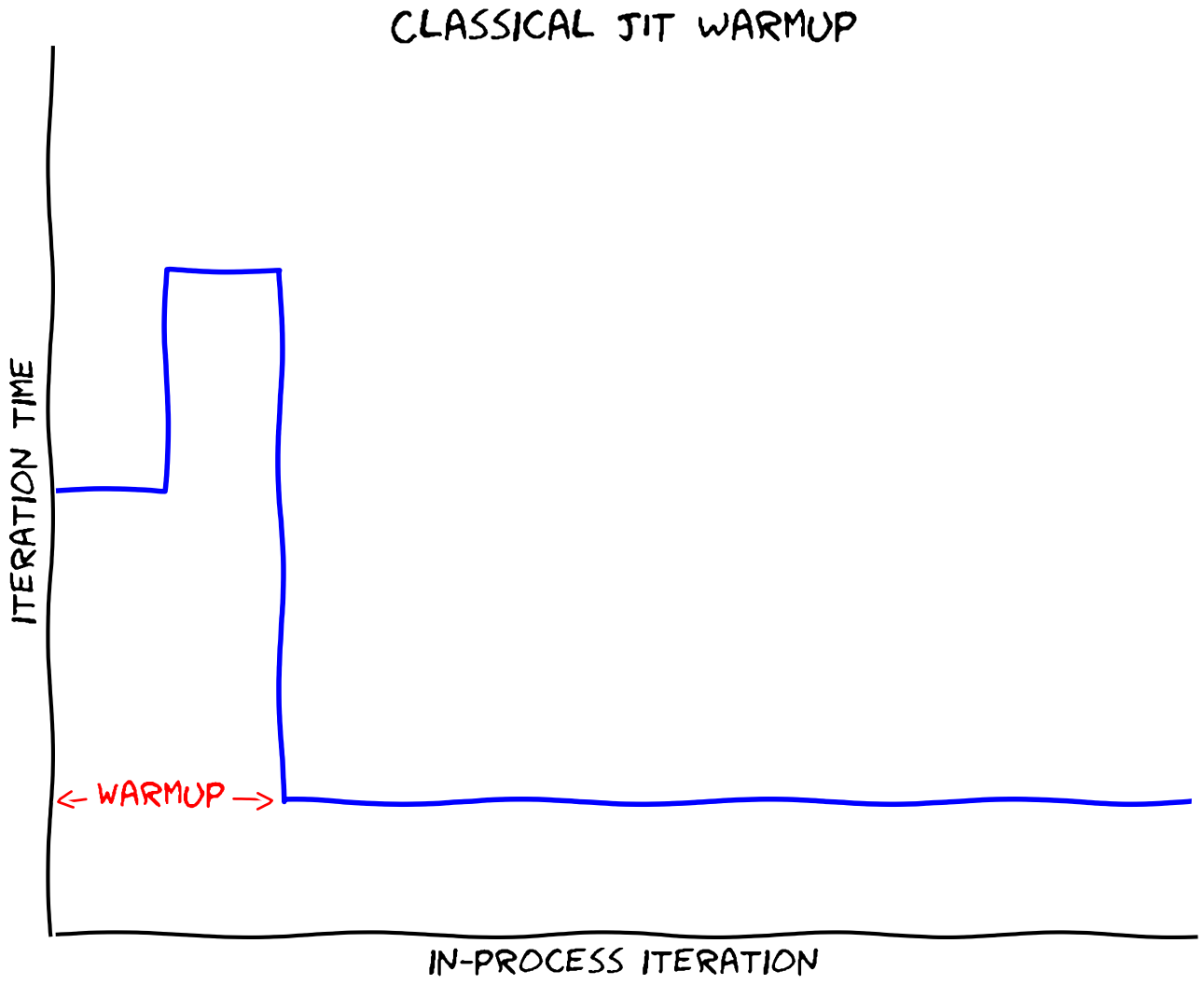
JIT Warmup Background



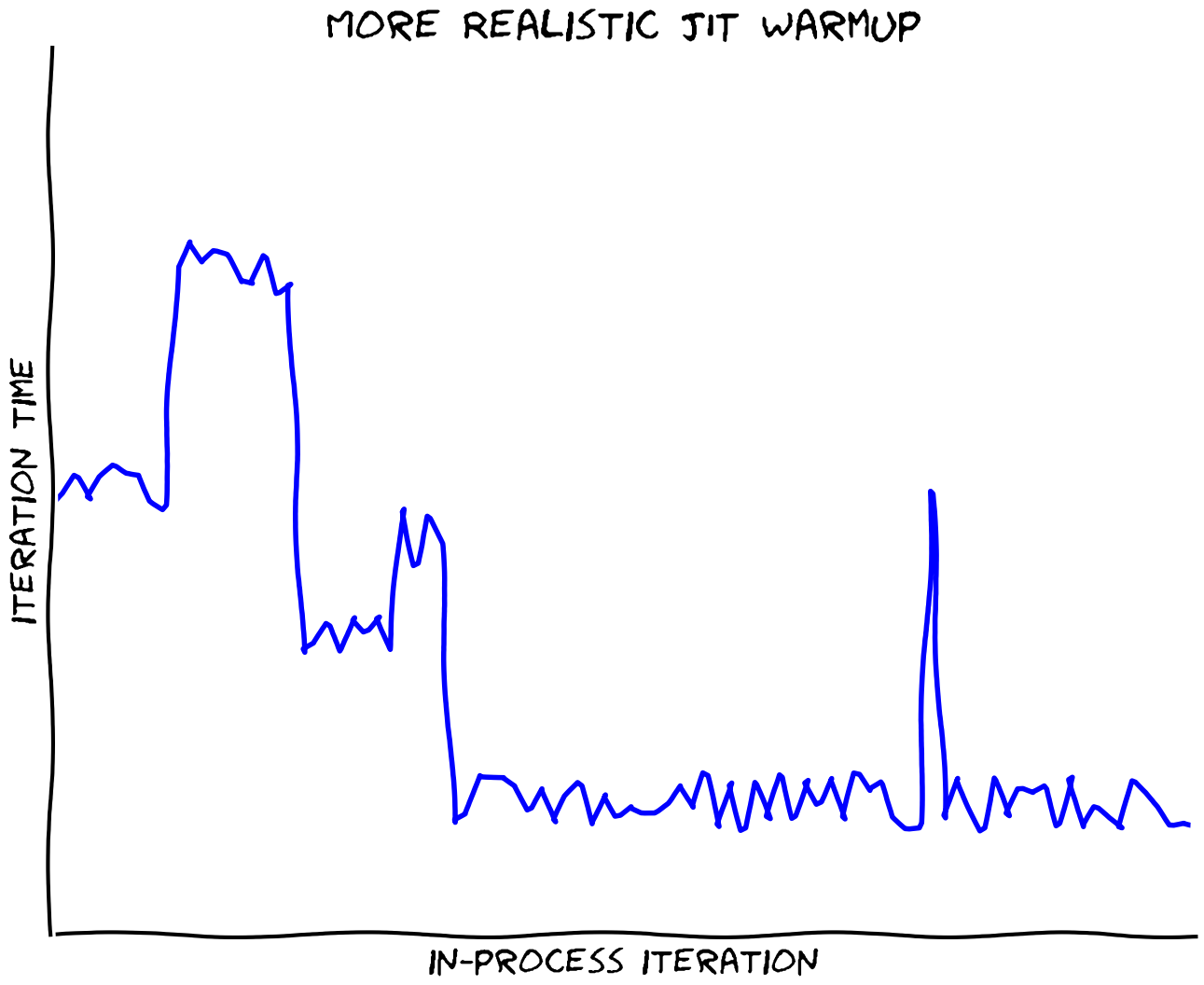
JIT Warmup Background



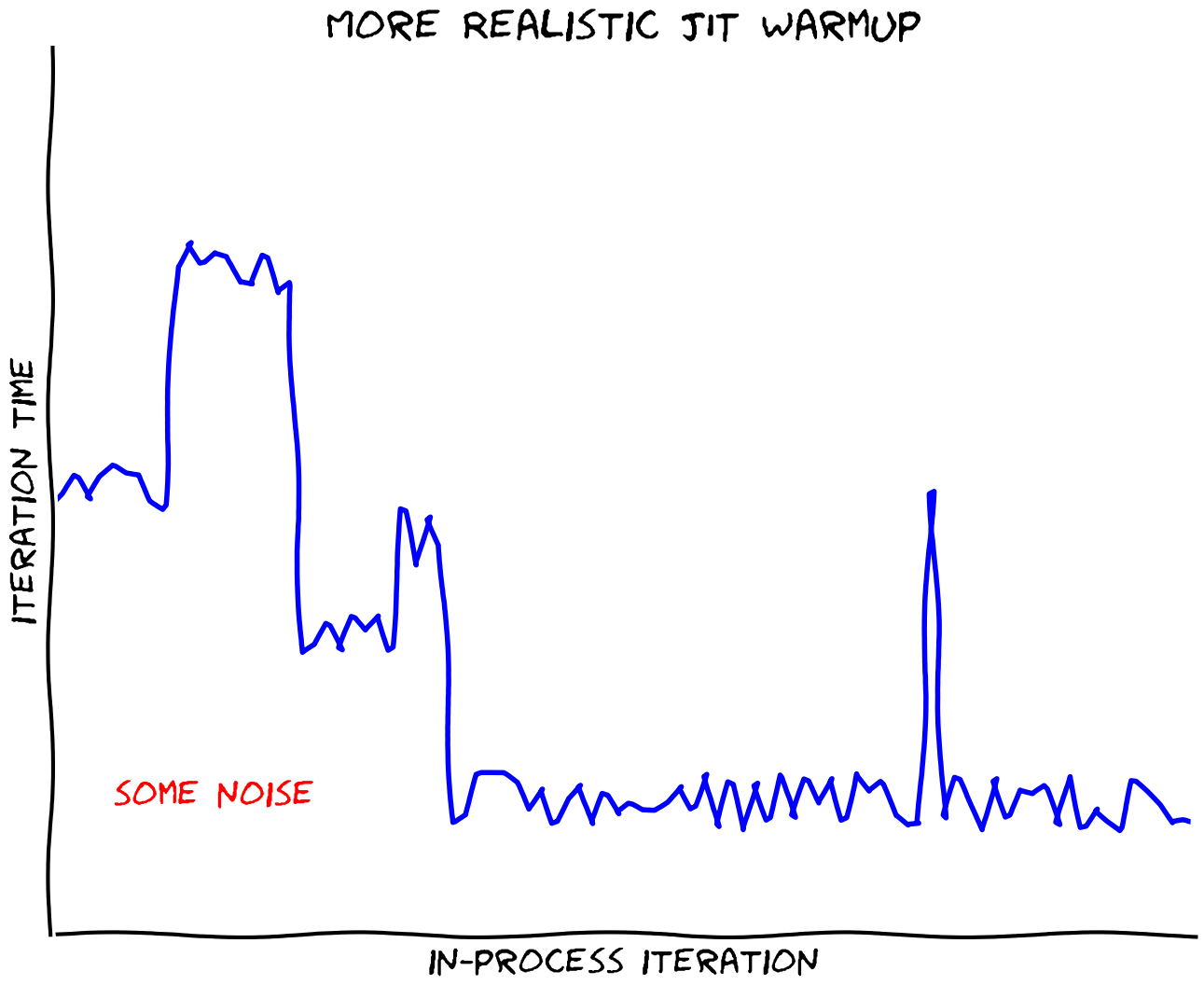
JIT Warmup Background



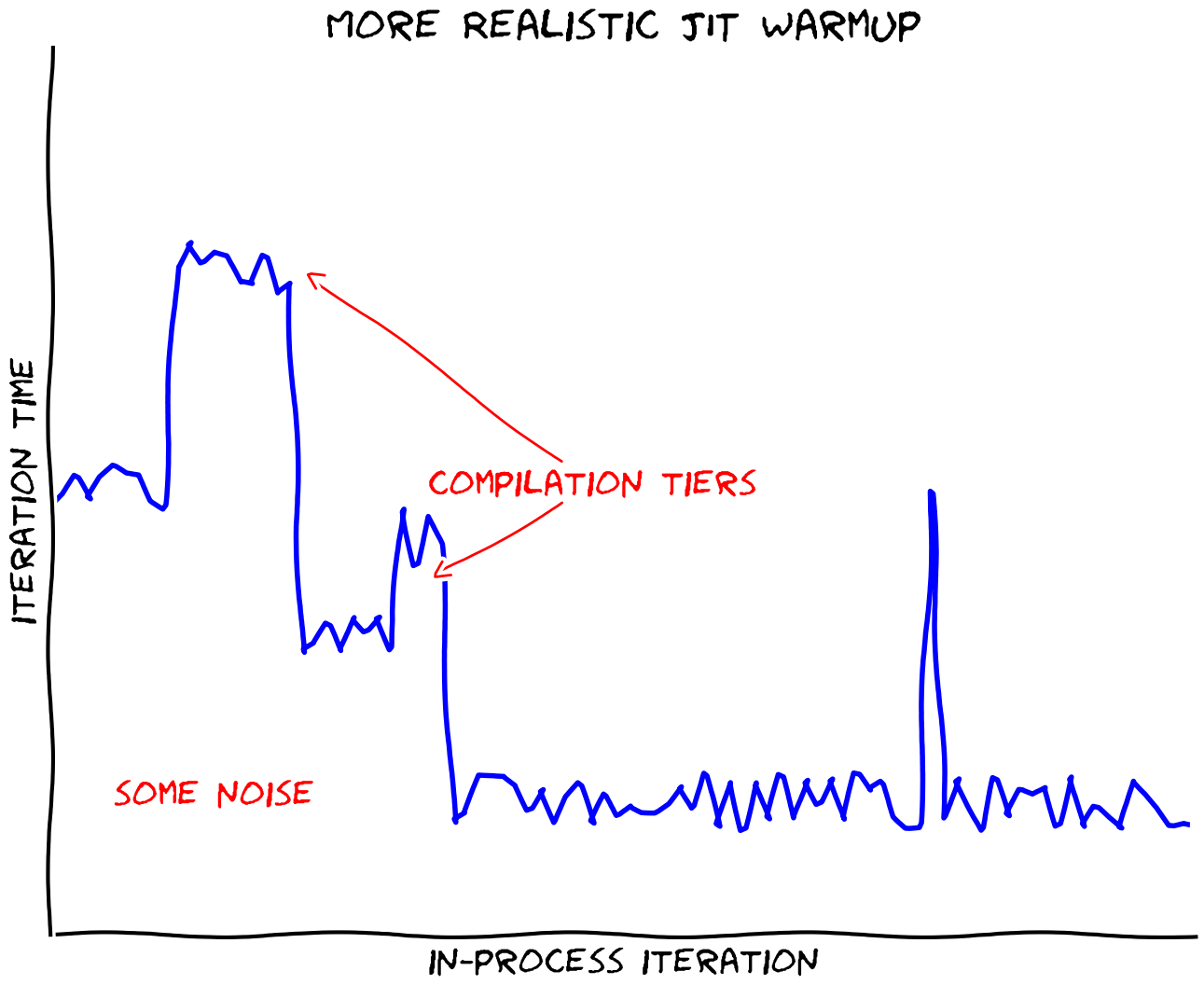
JIT Warmup Background



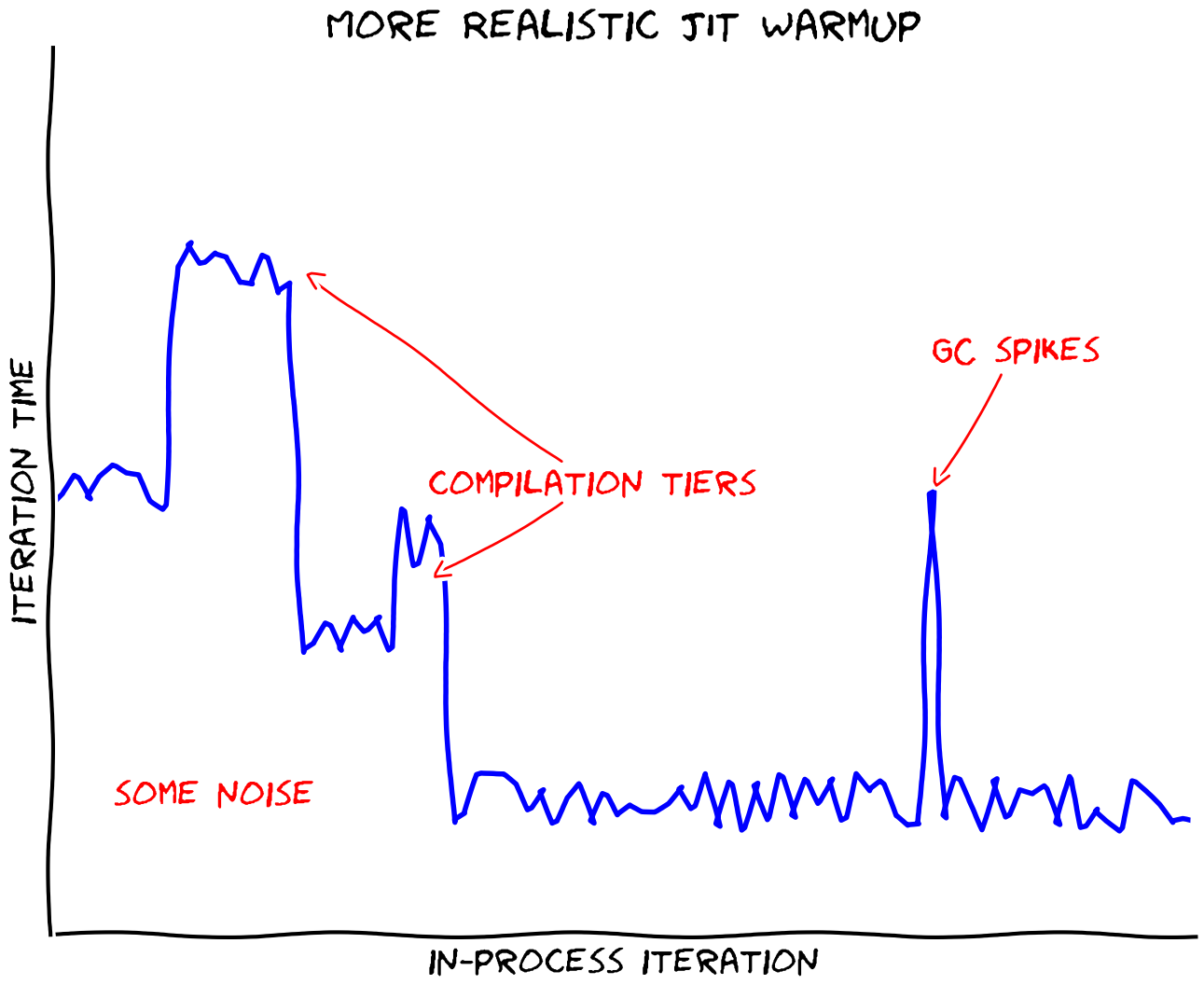
JIT Warmup Background



JIT Warmup Background



JIT Warmup Background



Why is Warmup Important?

- Warmup contributes to overall performance.
- Long warmup is bad for user-facing and short-lived programs.
- VM authors report peak performance.

The Back-Story

The Back-Story

- We have a hunch that warmup is longer than people expect.
- We have some preliminary ideas to improve warmup.

The Back-Story

Goal:

Measure how long modern JITs take to warm up.

The Warmup Experiment v1.0

- Microbenchmarks
- Reasonable number of repetitions.
  - 10 process executions.
  - 50 in-process iterations.
- Run on various VMs.
- Plot and report warmup time.

Weird Results

Many benchmarks don’t warmup under the classic model.

The Back-Story

New goal:

Try to understand why we see “weird” results.

The Warmup Experiment v2.0

Microbenchmarks Revisited

- CFG determinism.
  - Each run takes same path through CFG.
- Checksums.
  - Ensures different languages do the same work.
  - Harder for VMs to optimise away whole benchmark.

Code for microbenchmarks:

`https://github.com/softdevteam/warmup_experiment`

Krun

Benchmark runner that aims to control sources of variation.

WRT: memory limits, I/O, system state, . . .

`https://github.com/softdevteam/krun`

VMs

- Graal-0.13
- HHVM-3.12.0
- JRuby/Truffle (recent git version)
- Hotspot-8u72b15
- LuaJit-2.0.4
- PyPy-4.0.1
- V8-4.9.385.21
- GCC-4.9.3 (not really a VM)

Same GCC across the board, minor VM patching.

Machines

- Linux-Debian8/i4790K, 24GiB RAM
- Linux-Debian8/i4790, 32GiB RAM
- OpenBSD-5.8/i4790, 32GiB RAM

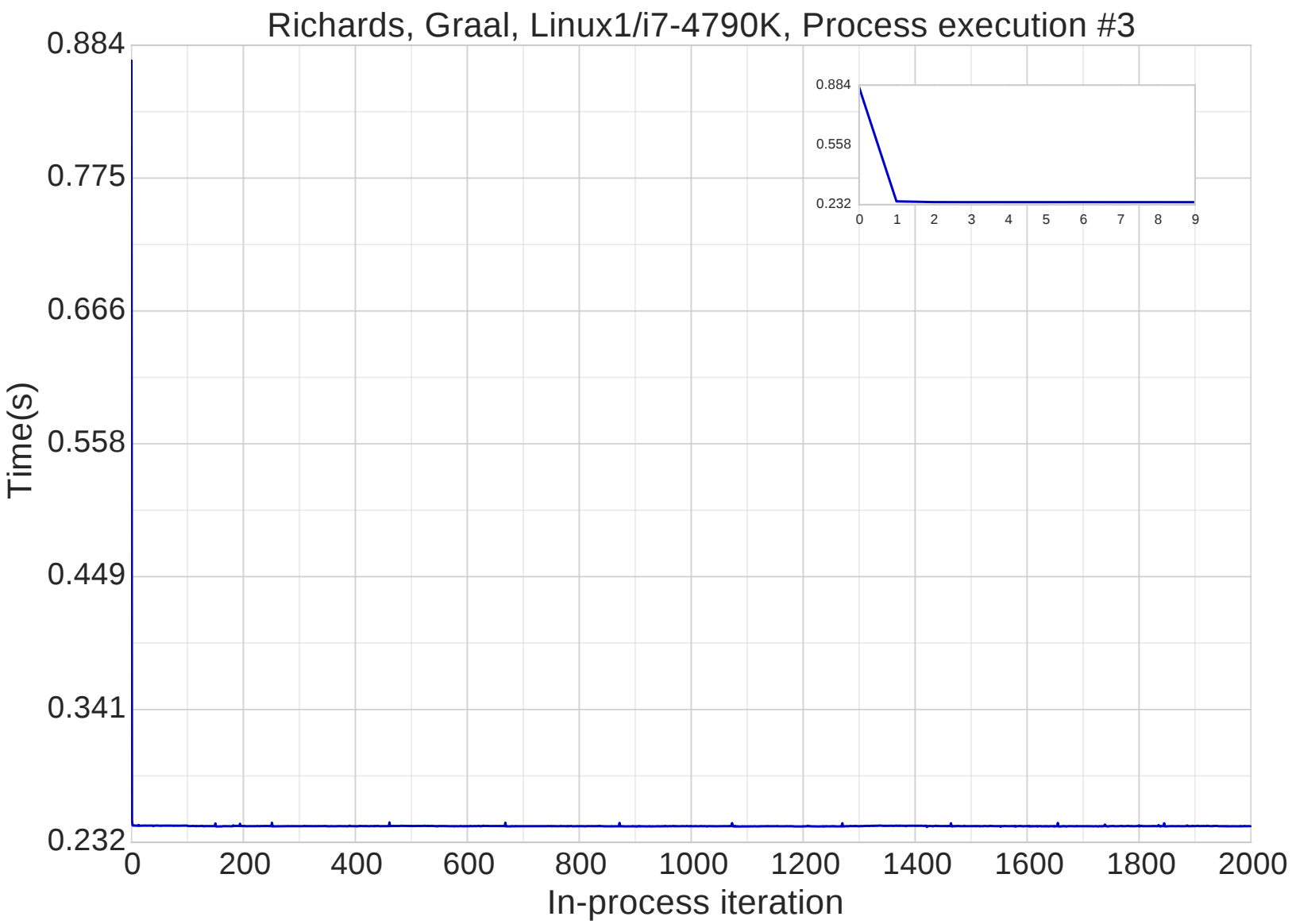
- “Turbo boost” disabled.
- SSH blocked from non-local machines.
- Daemons disabled (e.g. cron, smtpd).

Run for Longer

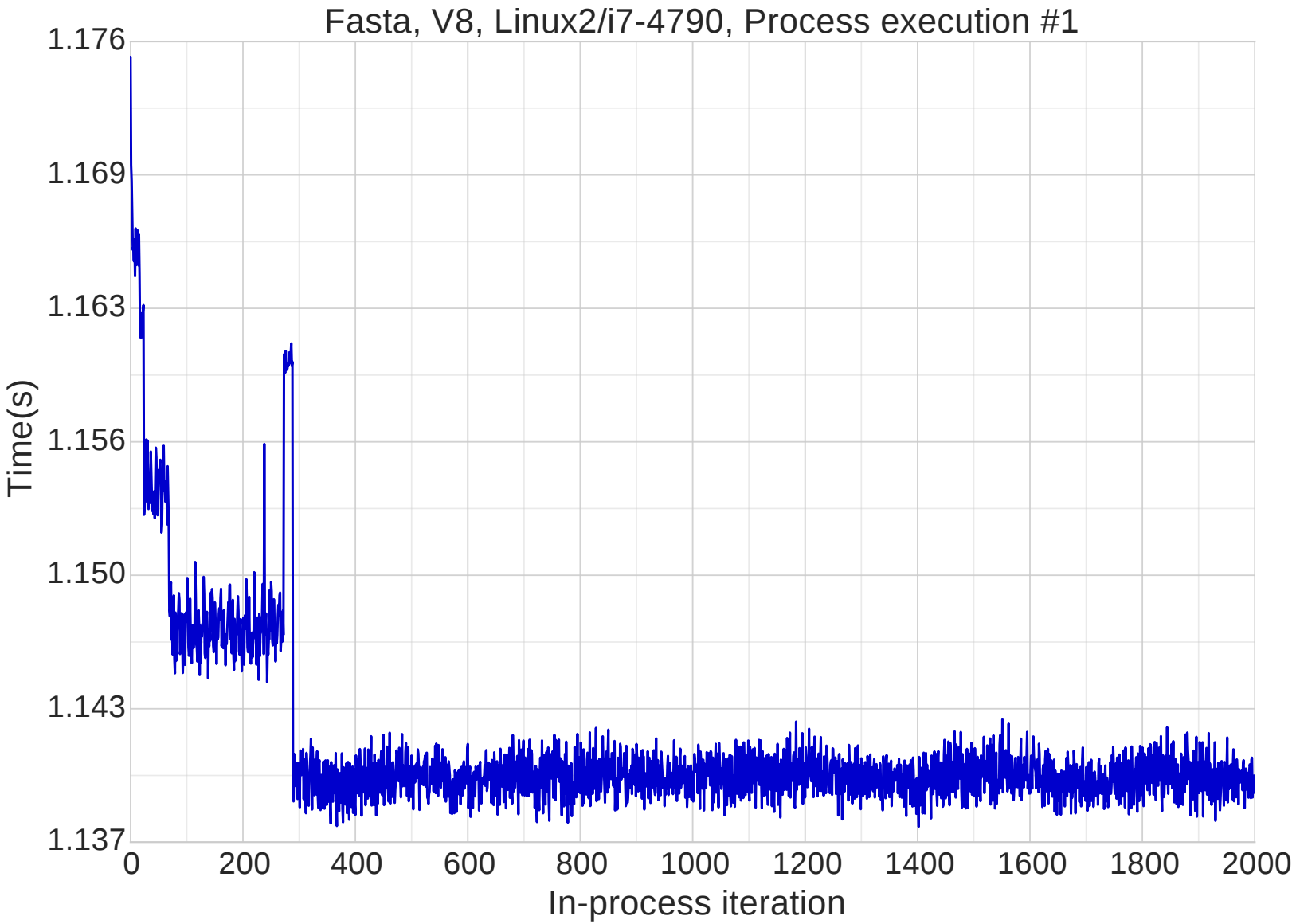
- Run many more in-process iterations (2000).
- Plot results and see if we see classic warmup now.

Results

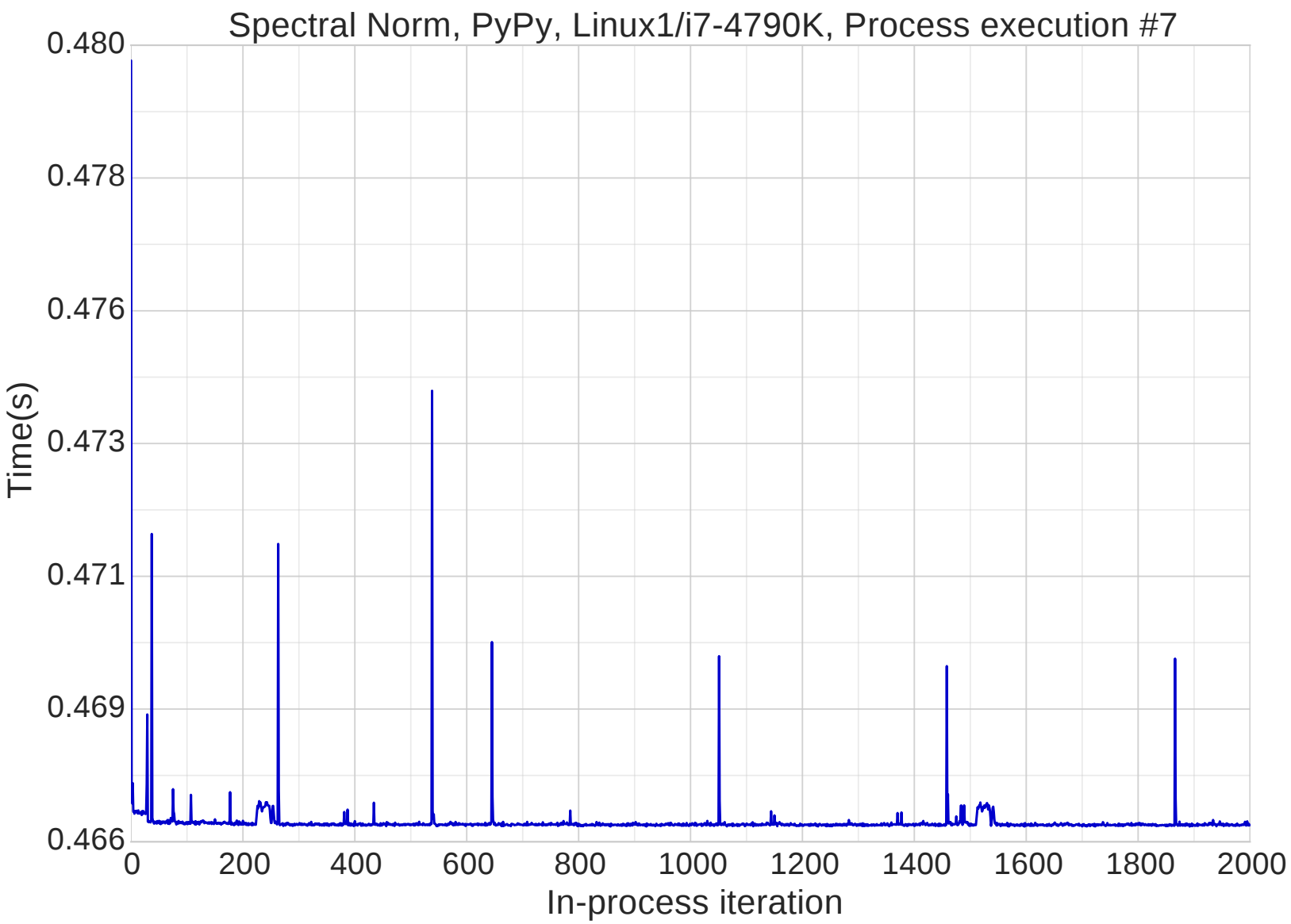
Classical Warmup



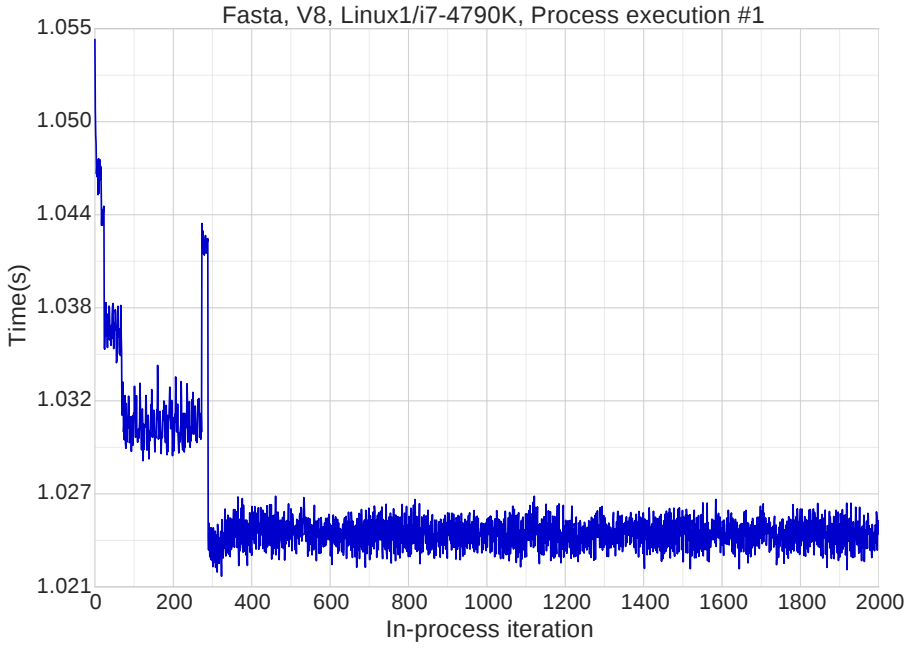
Classical Warmup

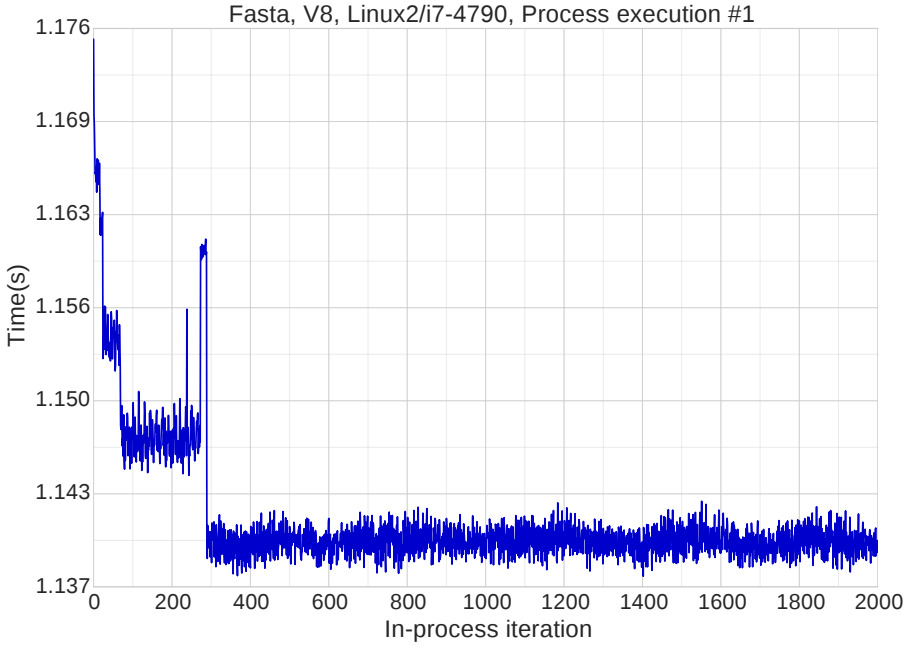


Classical Warmup



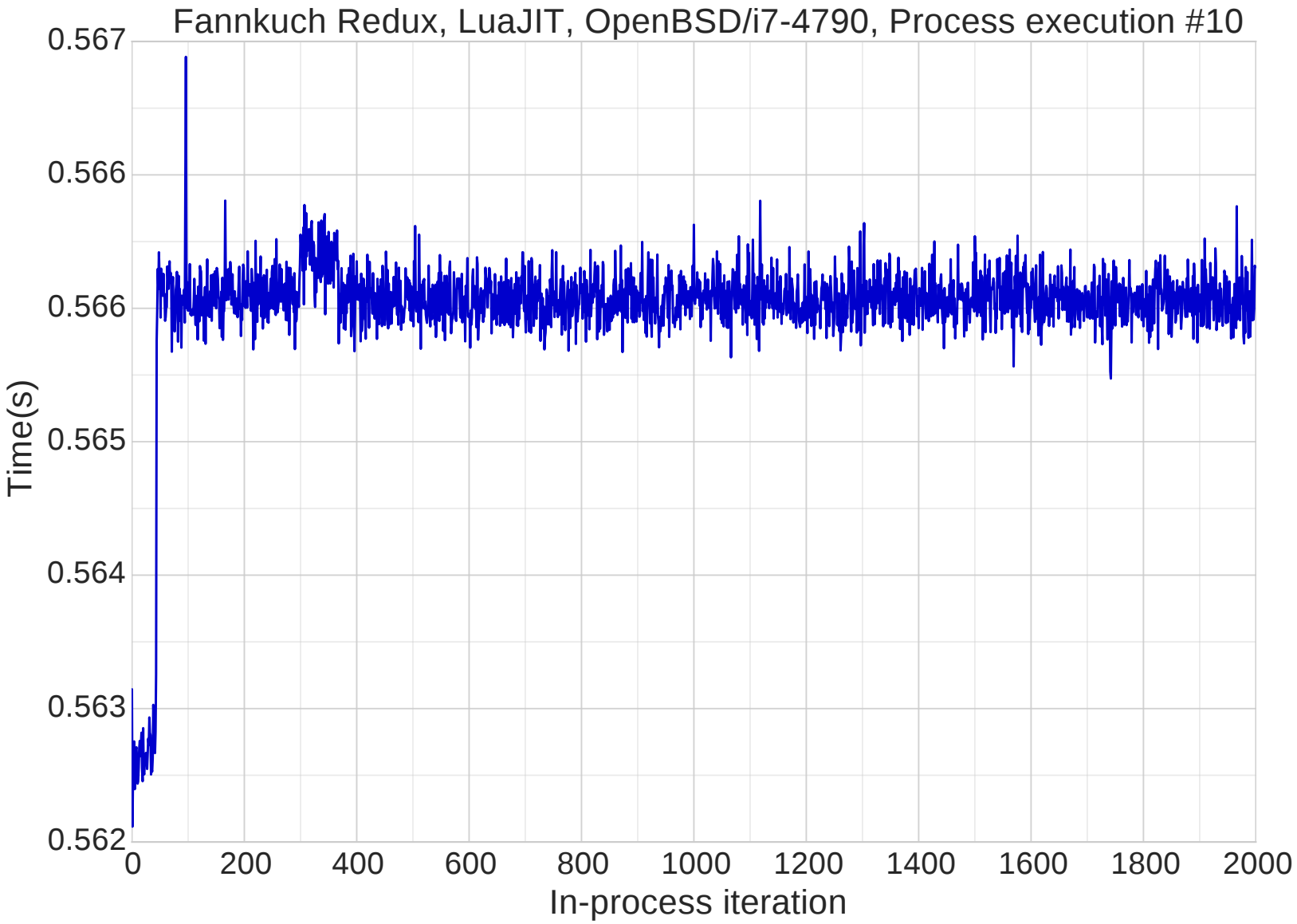
Classical Warmup





(Different machines)

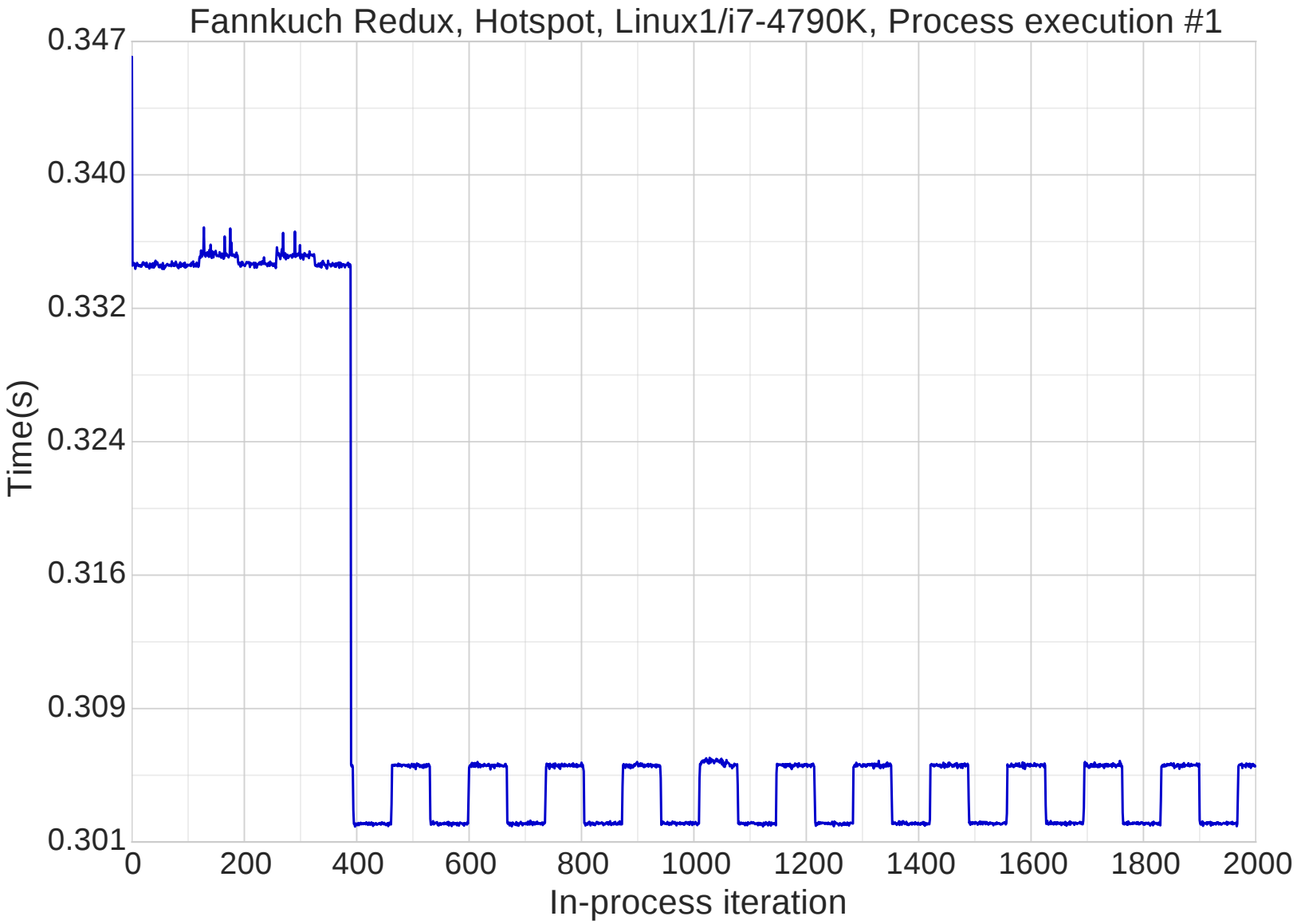
Slowdown


Fannkuch Redux, LuaJIT, OpenBSD/i7-4790, Process execution #10
Time(s)
0.567
0.566
0.566
0.565
0.564
0.563
0.562
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

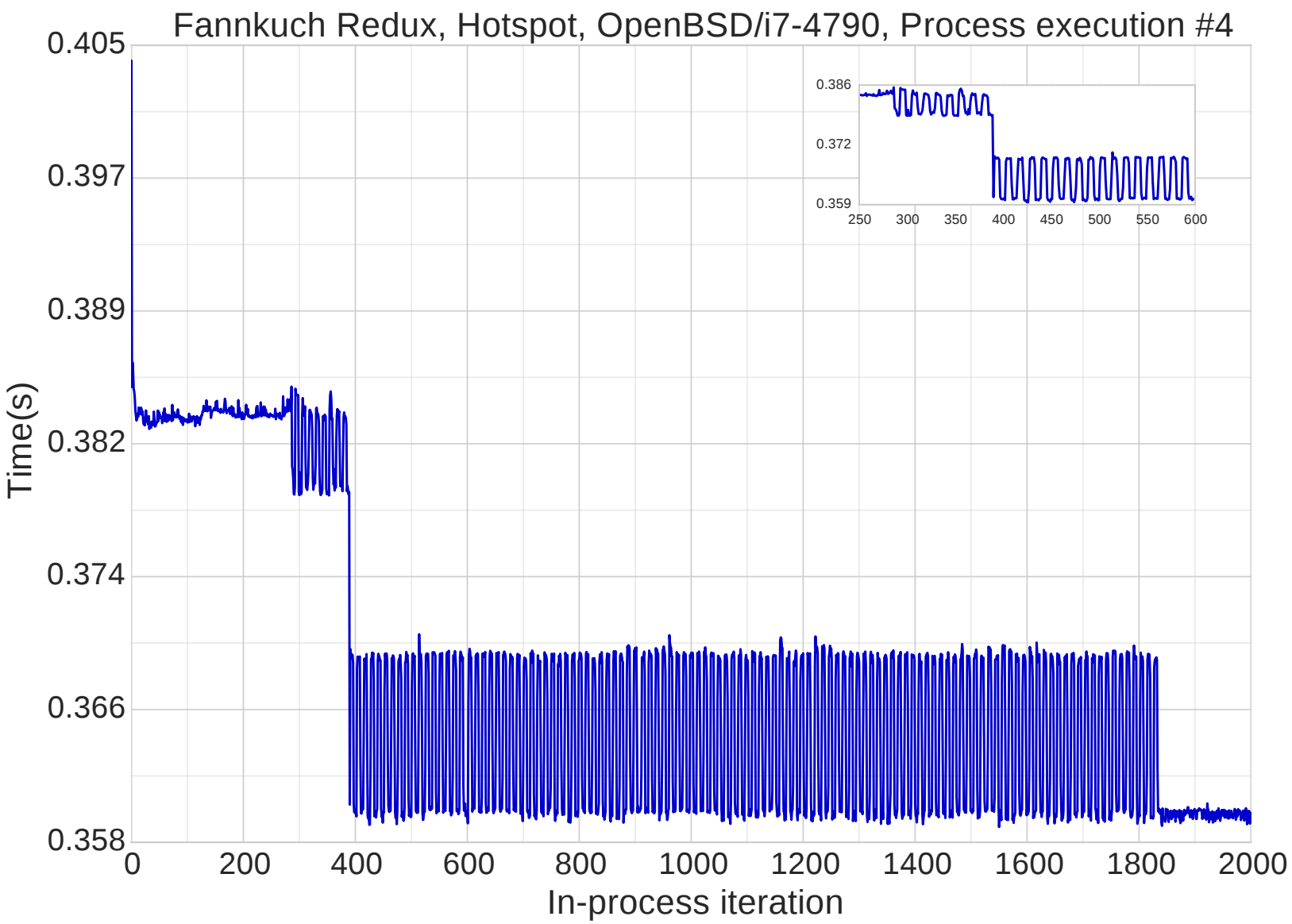
Slowdown


Richards, Hotspot, Linux2/i7-4790, Process execution #2
Time(s)
0.298
0.293
0.287
0.282
0.276
0.271
0.266
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

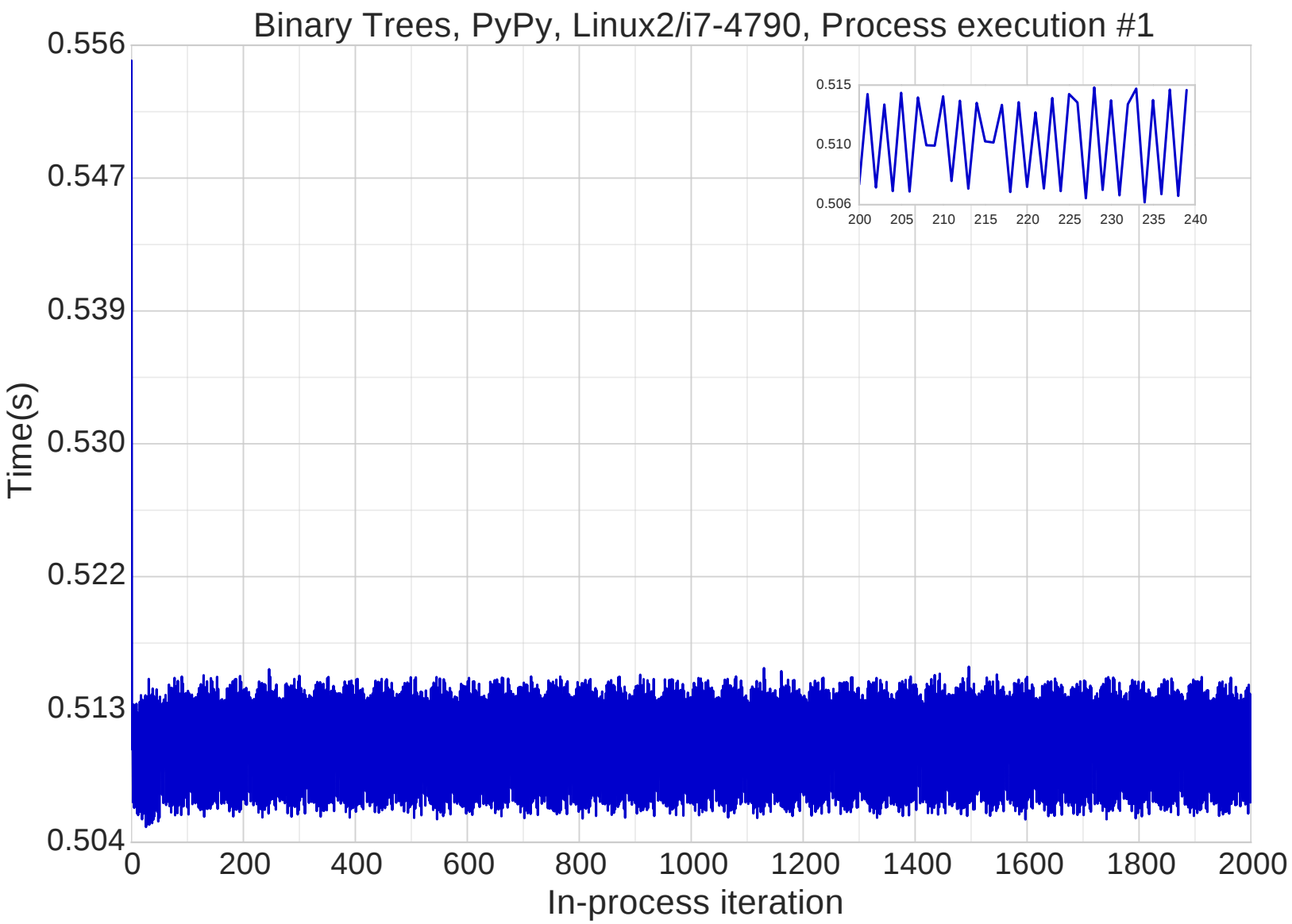
Cycles


Fannkuch Redux, Hotspot, Linux1/i7-4790K, Process execution #1
Time(s)
0.347
0.340
0.332
0.324
0.316
0.309
0.301
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

Cycles

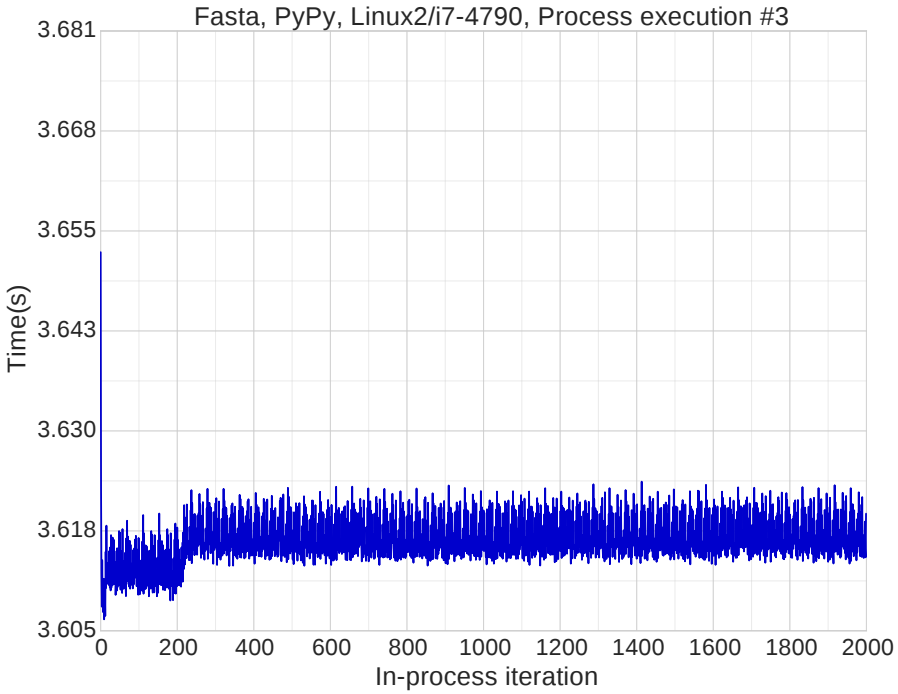

Fannkuch Redux, Hotspot, OpenBSD/i7-4790, Process execution #4
Time(s)
0.405
0.397
0.389
0.382
0.374
0.366
0.358
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration
0.386
0.372
0.359
250
300
350
400
450
500
550
600

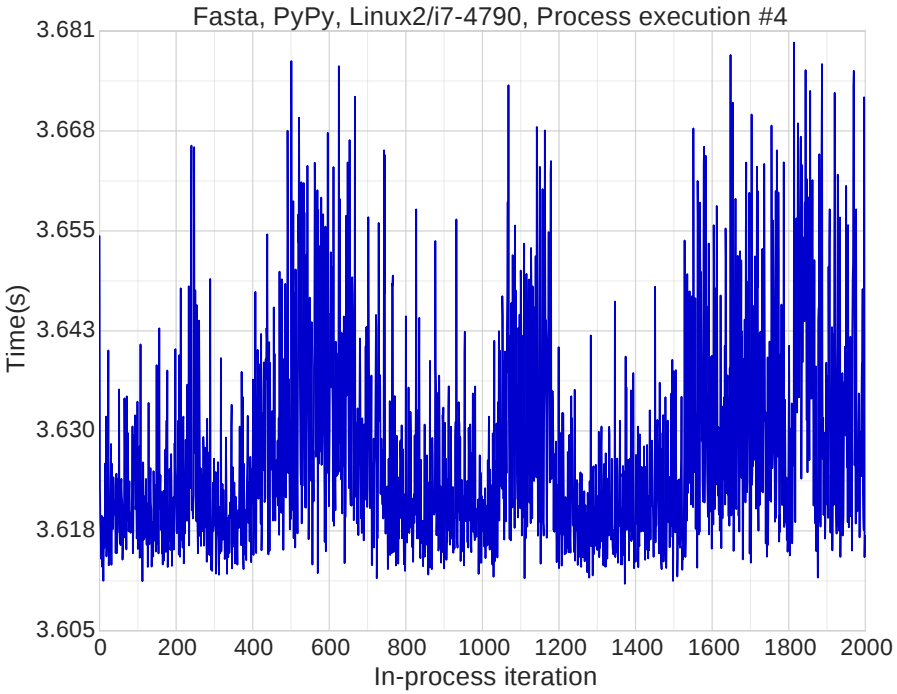
Cycles


Binary Trees, PyPy, Linux2/i7-4790, Process execution #1
Time(s)
0.556
0.547
0.539
0.530
0.522
0.513
0.504
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration
0.515
0.510
0.506
200
205
210
215
220
225
230
235
240

Changing Phases

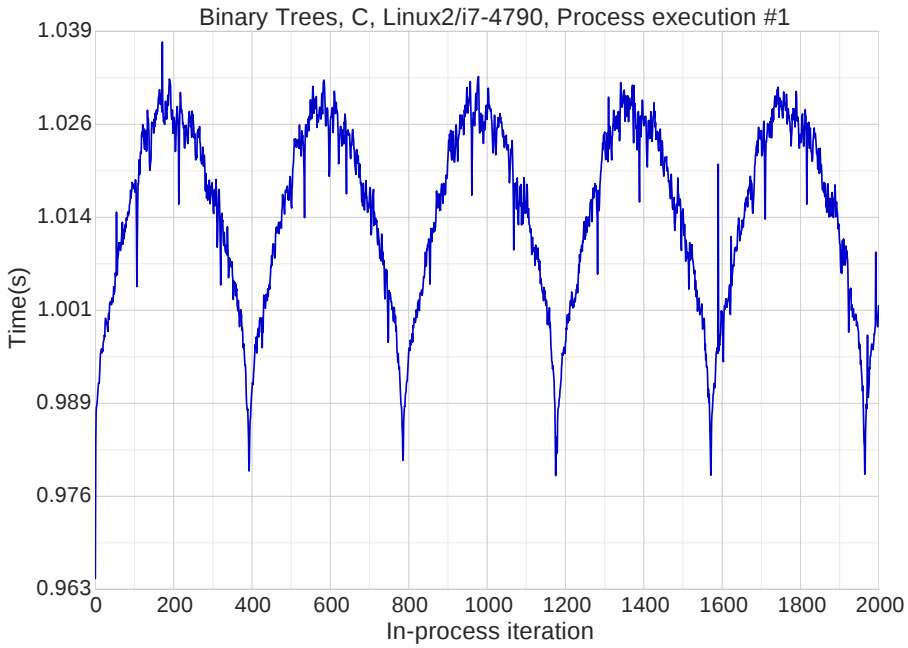
Vastly Inconsistent Process-executions





(same machine)

Vastly Inconsistent Process-executions



(Different machines. Bouncing ball pattern Linux-specific)

Full Results

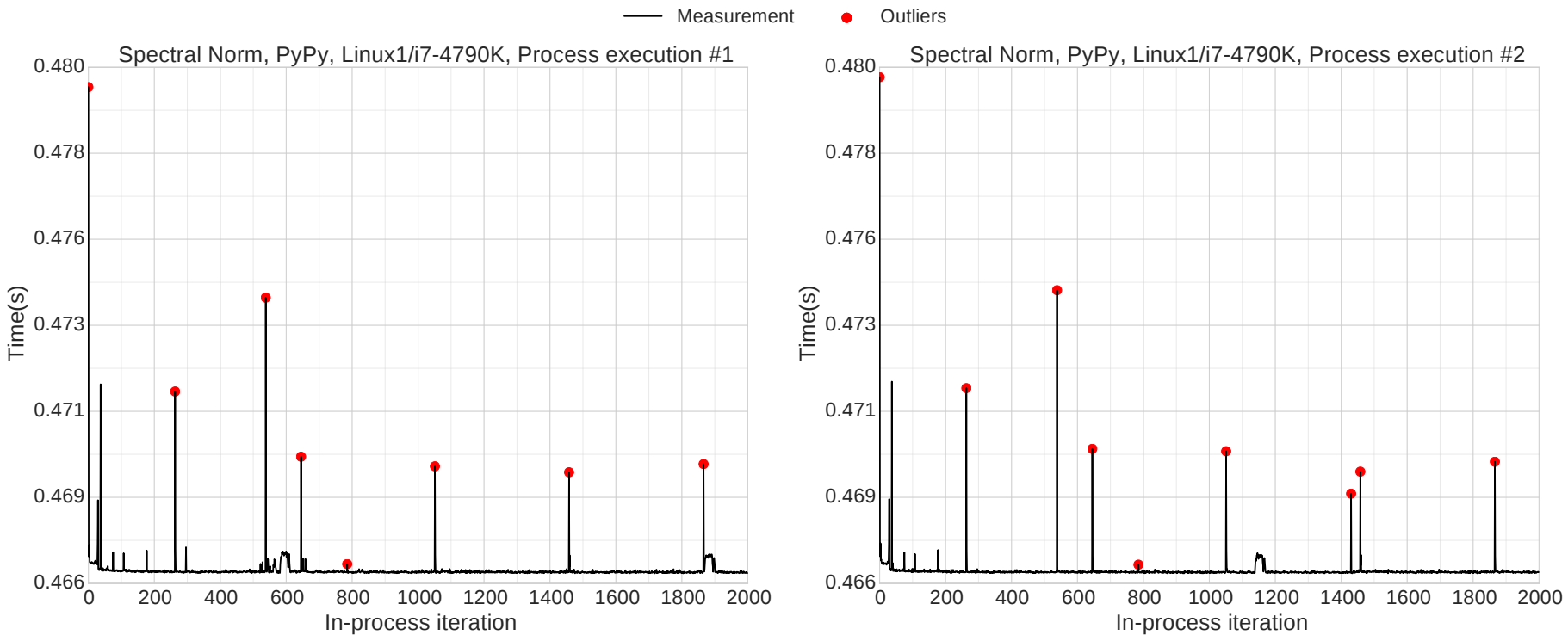
https://archive.org/download/softdev_warmup_experiment_artefacts/v0.2/

all_graphs.pdf All plots in one huge PDF.

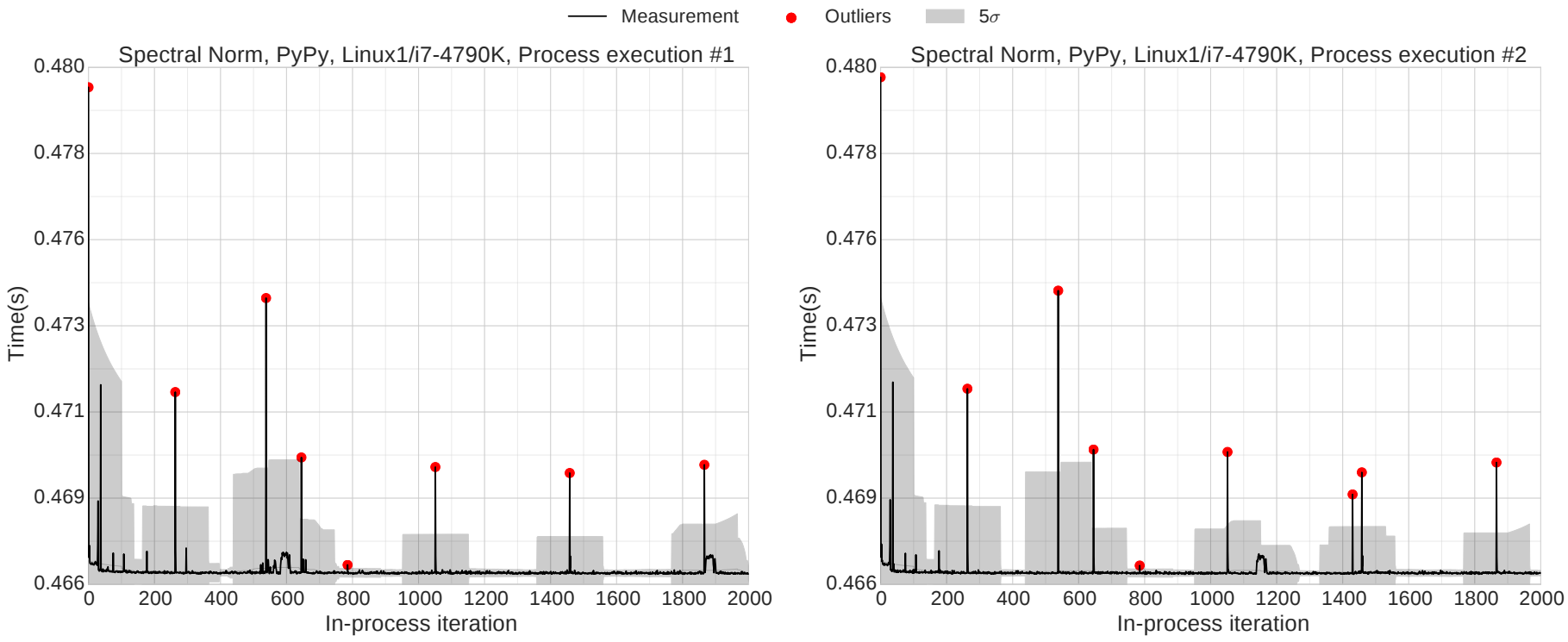
warmup_results*.json.bz2 Raw results.

Automated Analyses

Automated Analyses: Outlier Detection

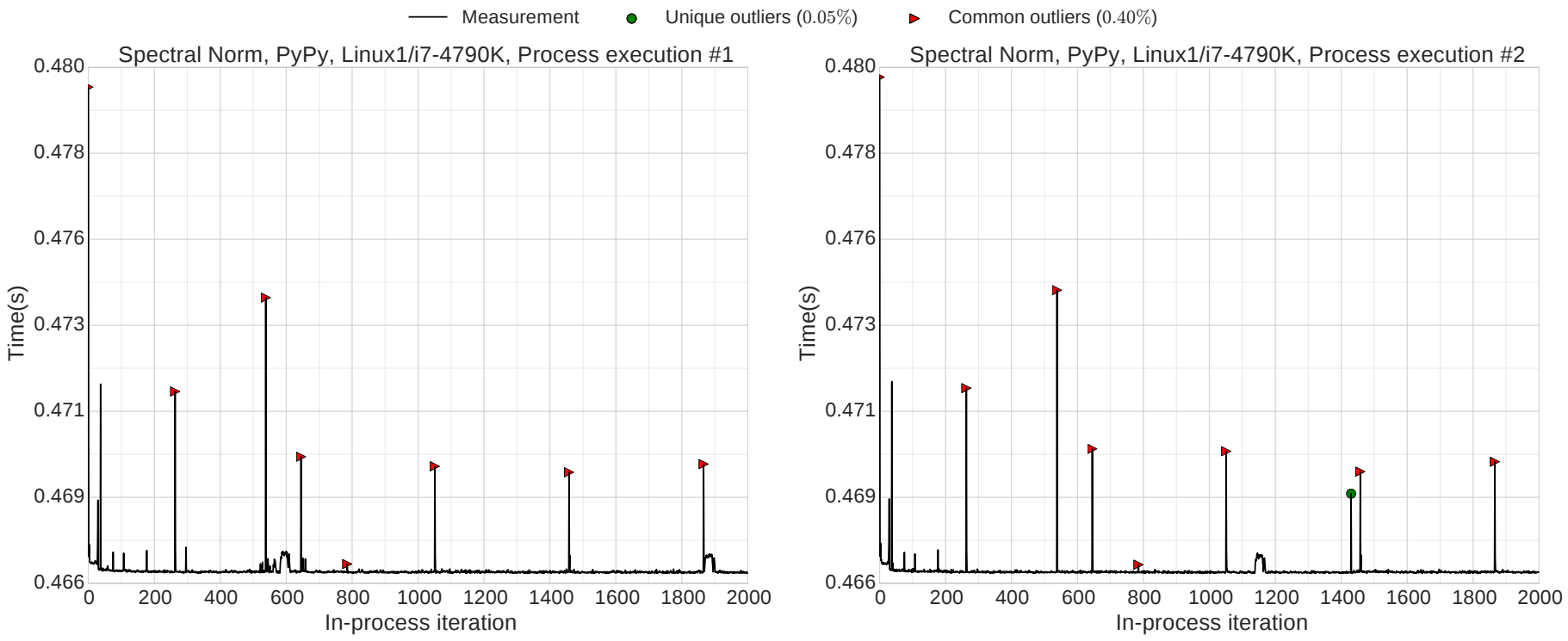


Automated Analyses: Outlier Detection



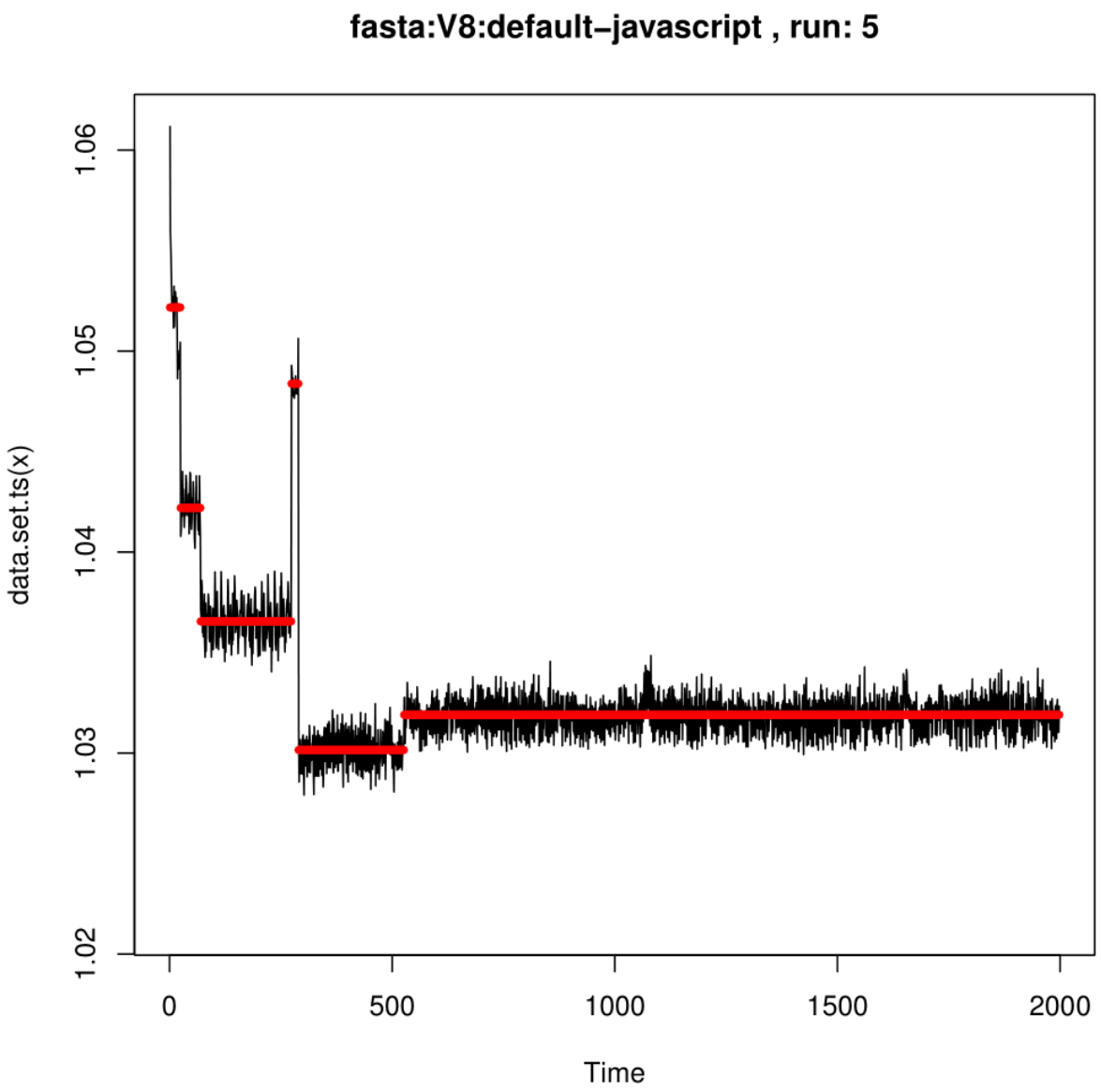
outliers outside 5σ of rolling average

Automated Analyses: Outlier Detection

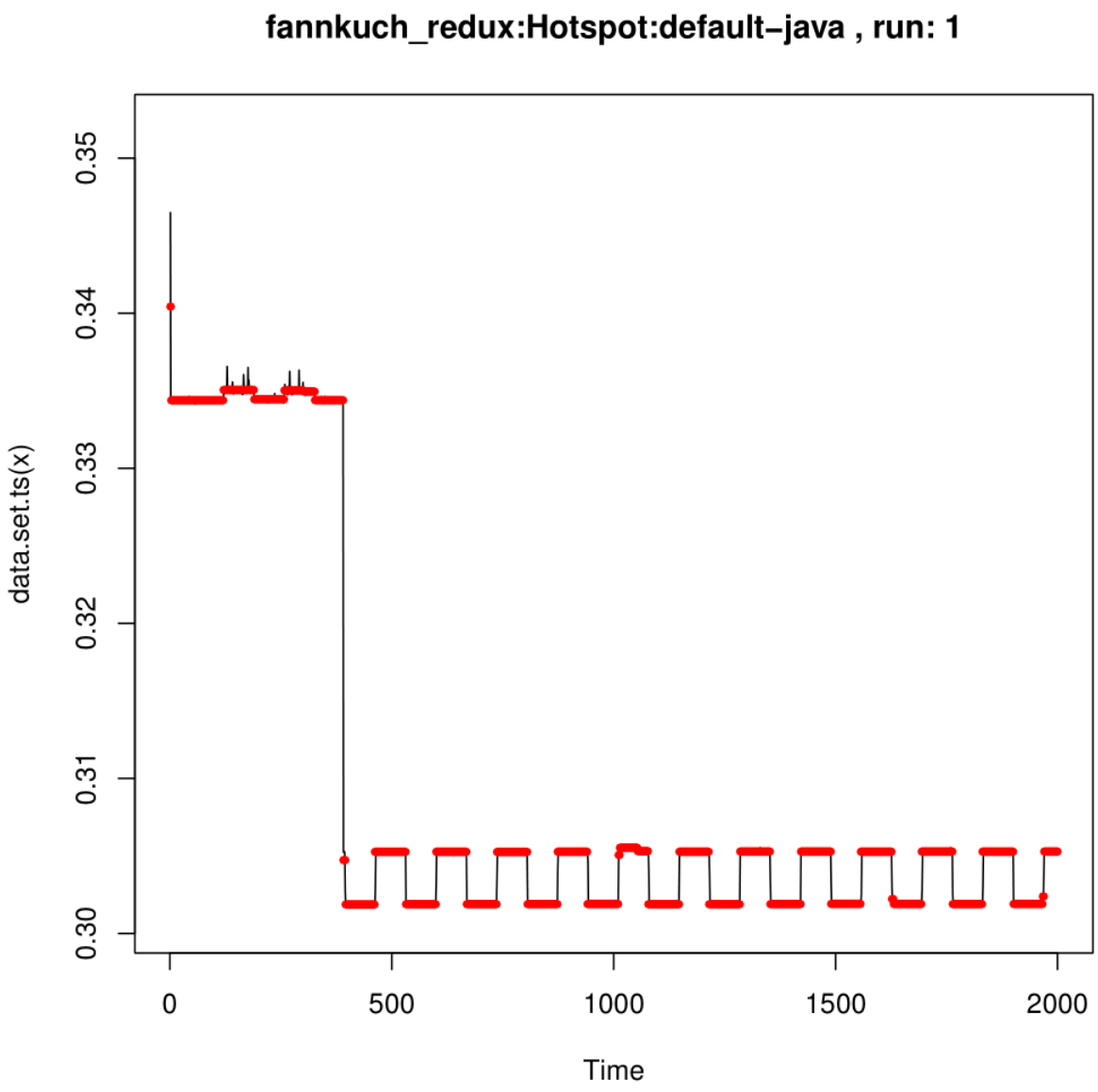


Recurring outliers

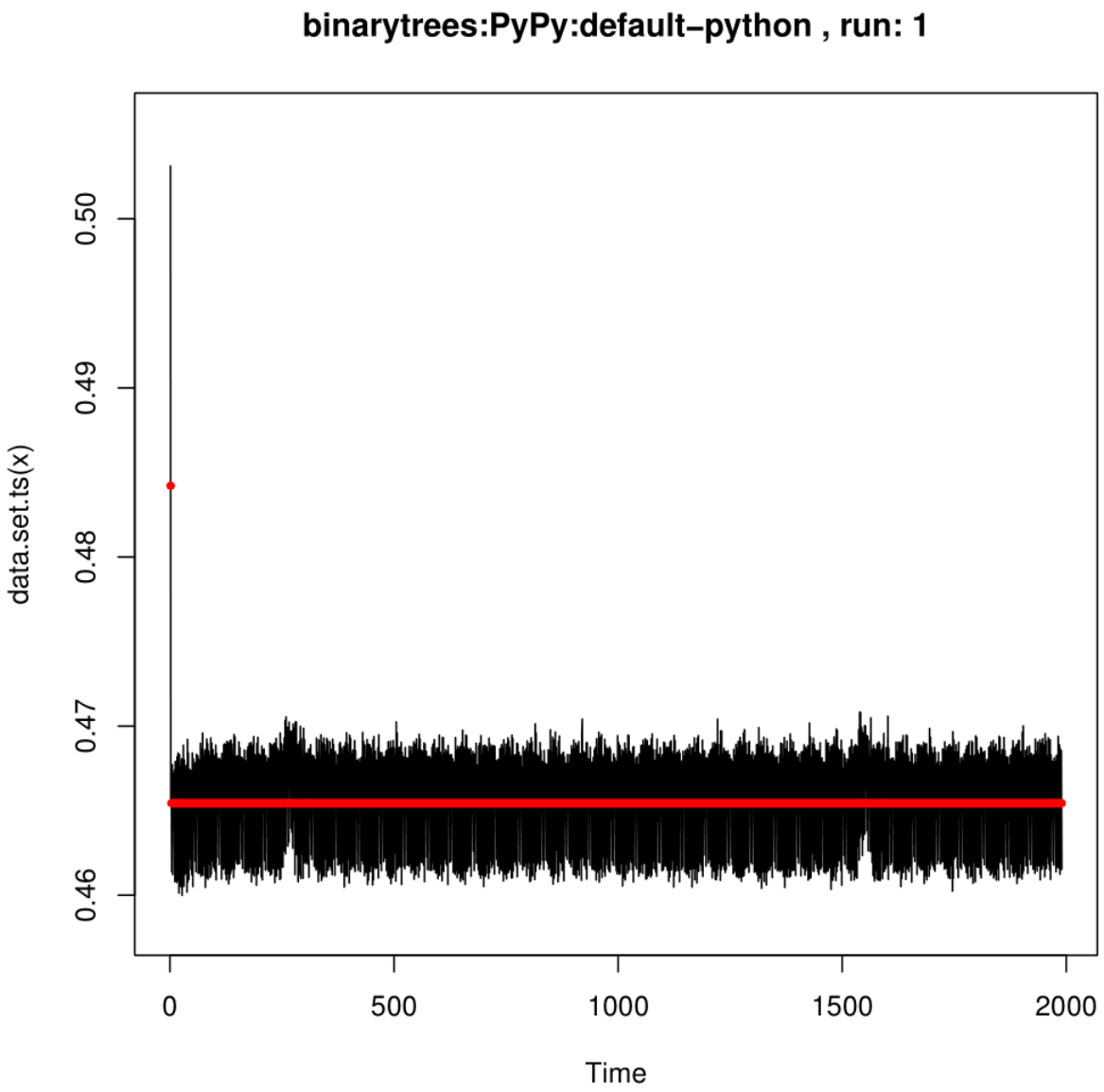
Automated Analyses: Change-point Analysis



Automated Analyses: Change-point Analysis



Automated Analyses: Change-point Analysis



Conclusion and Future Work

Conclusion

We can’t rely on the classical warmup model.

Future Work

- Extend automated analyses.
- More {benchmarks, VMs, arches, OSs}.
- Try to assign meaning to artefacts in plots.
  - E.g. is that spike at $x = 78$ actually {GC, compilation, ...}.
- Memory consumption over time.
  - Correlation with iteration times?
- Look at hardware performance counters?
- What else?

References

JIT 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

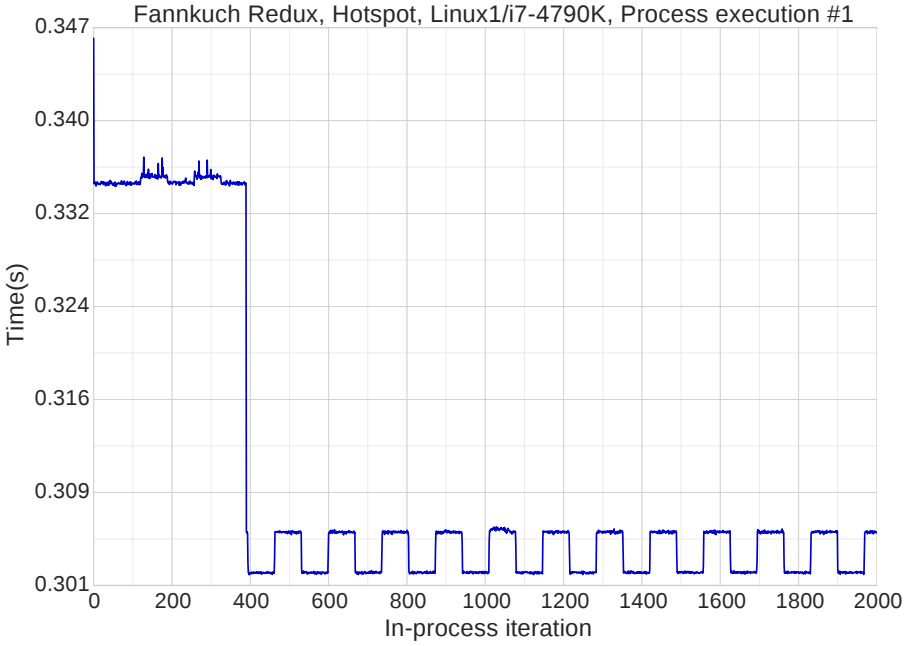
References

NO_HZ: Reducing Scheduling-Clock Ticks
Linux Kernel Documentation

Intel P-state driver
Linux Kernel Documentation

malloc.conf(5)
OpenBSD Manual Pages

That’s a wrap. Thanks!



Krun Controls

Platform independent controls:

- Minimises I/O.
- Consistently limits heap and stack ulimits.
- Drops privileges to a fresh *krun* UNIX account.
- Automatically reboots the system prior to each proc. exec.
- Checks dmesg for changes after each proc. exec.
- Checks system is at same temperature for each proc. exec.

Krun Controls

Linux controls. Krun checks:

- Intel P-state support is disabled in the kernel.
- The `performance` governor is used.
- The kernel is "tickless" (`NO_HZ_FULL_ALL`).
- The `perf` sample rate is lowest possible (1Hz).
- ASLR is disabled.

(Note: Linux ignores ulimits)

Krun Controls

OpenBSD controls. Krun checks:

- Malloc flags are the least noise inducing.
- `apmd` is running in performance mode.

On OpenBSD:

- We can't disable ASLR
- We can't disable ticks.
- We can't disable P-states in software.
- There is no kernel profiler (good for us).