Benchmarking over JVM

Challenges and issues

Context

- Study of diversity in DSL
  - Patterns (Interpreter, Visitor, Revisitor, Switch and Truffle)
  - JVMs (Hotspot, OpenJ9, GraalVM)
  - Programs
- Impact of this diversity on the performance of the interpreters
  - Especially Truffle's optimizations

Issues of performance evaluation

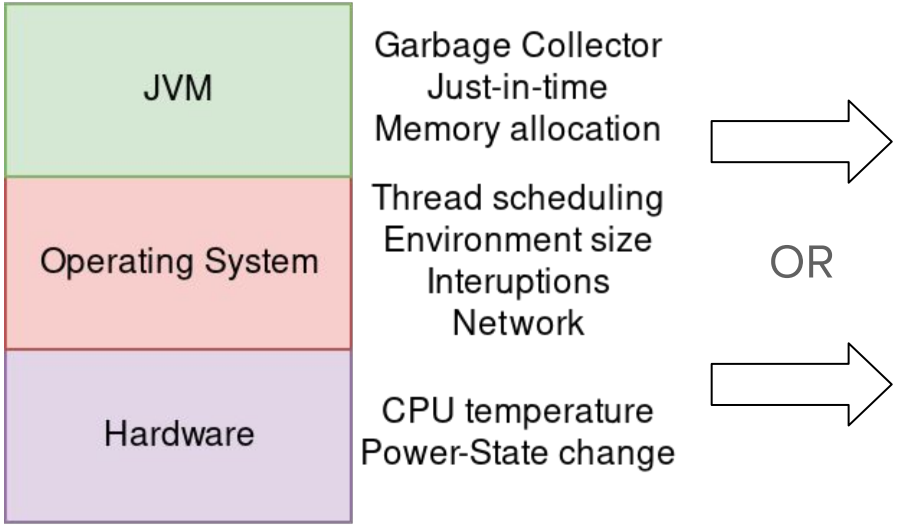
The main issue is the reproducibility of experimental results

➔ Two executions of the same program will last different time

> “*One of the distinguishing features of anything that aspires to the name of science is* ***the reproducibility of experimental results****.*”
>
> - Matthew Stewart

Causes of the lack of reproducibility

- ➔ Due to computer’s non-deterministic behaviors

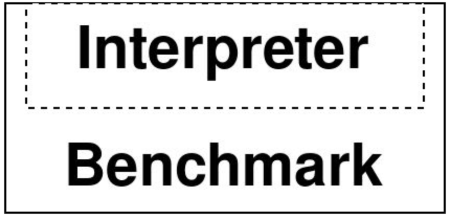


Limit the impact of those behaviours

OR

Debiasing results using statistical methods

From benchmark’s design

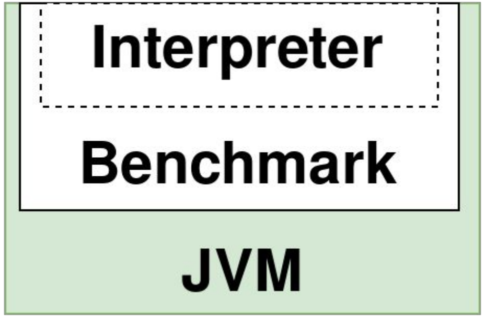


Interpreter

Benchmark

Problems	Solutions
[Shipilev, 2013] Optimization of your benchmark : - Common subexpression elimination - Dead Code Elimination - Constant Folding, ...	Carefully design your benchmarks Use tools (JMH, AutoJMH) [Rodriguez-Cancio et al, 2016]
[Shipilev, 2014] Precision of JVM time counters : - Latency of the function - Granularity of the counters	Nothing, it’s a physical limit Hopefully, it’s insignificant if you don’t do *nano*-benchmarks

From JVM



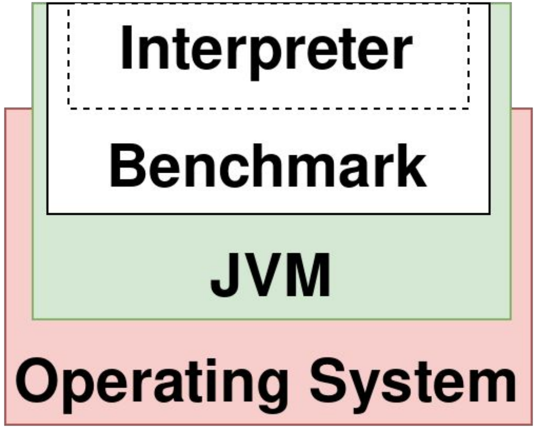
Interpreter

Benchmark

JVM

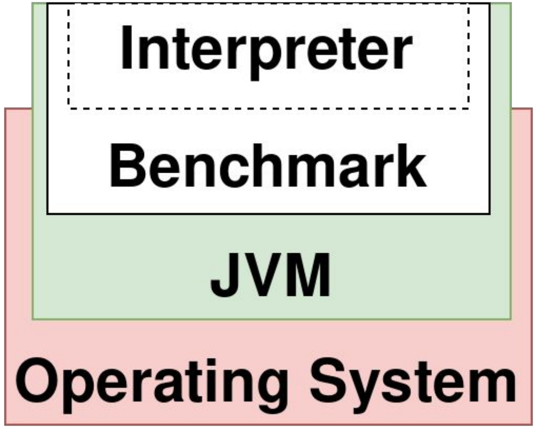
Problems	Solutions
[Arnold et al, 2005] Just-In-Time Compilation	Replay compilation [Georges et al, 2008]
[Blackburn et al, 2004] Garbage Collector	Environment variability management [Barrett et al, 2017]

From Operating System



Problems	Solutions
[Shipilev, 2013] Thread scheduling	Run benchmarks longer
[Mytkowicz et al, 2009] UNIX environment size	Fix the environment size [Barrett et al, 2017] Shake the input [Tsafrir et al, 2007]
[Horký et al, 2015] False memory sharing	Add padding to your data [Shipilev, 2013]
[Barrett et al, 2017] Memory Swapping	Restart between each VM invocation [Barrett et al, 2017]
....	....

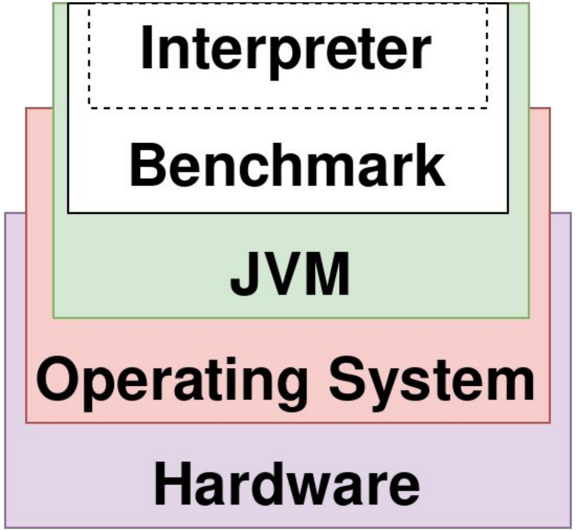
From Operating System



[Kalibera et al, 2005]

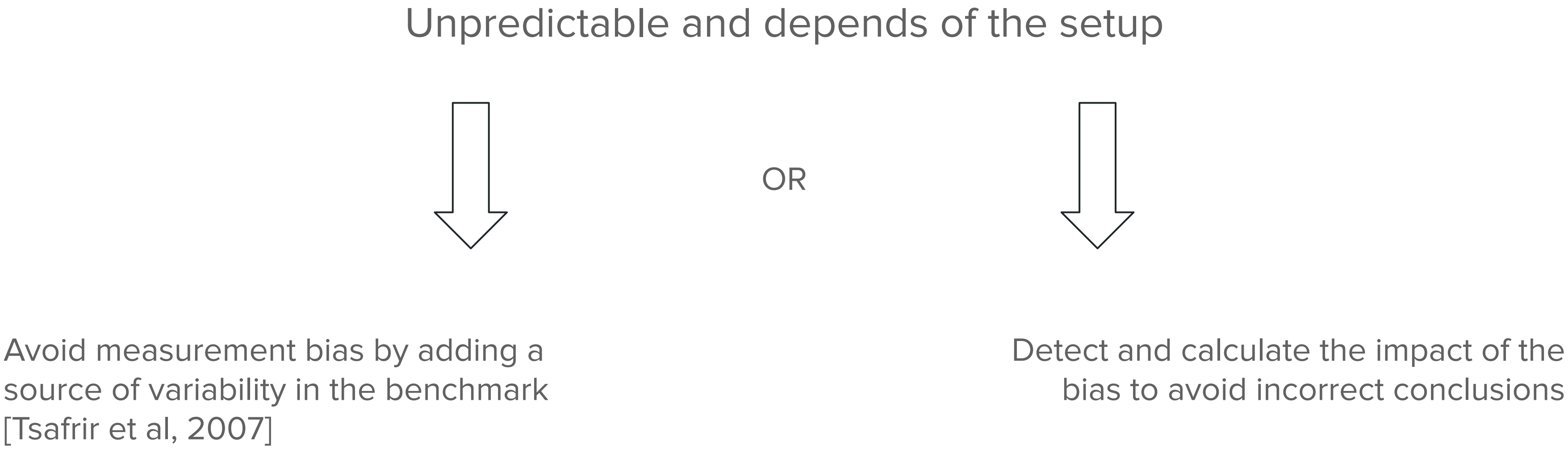
Problems	Solutions
[Shipilev, 2013] Thread scheduling	Run benchmarks longer
[Mytkowicz et al, 2009] UNIX environment size	Fix the environment size [Barrett et al, 2017] Shake the input [Tsafrir et al, 2007]
[Horký et al, 2015] False memory sharing	Add padding to your data [Shipilev, 2013]
[Barrett et al, 2017] Memory Swapping	Restart between each VM invocation [Barrett et al, 2017]
….	….

From Hardware



Problems	Solutions
[Barrett et al, 2017] CPU Frequency/Power State	Set a constant frequency and monitor it Monitor the temperature too [Barrett et al, 2017]
[Barrett et al, 2017] CPU temperature	
[Barrett et al, 2017] Hardware error	Monitor dmesg buffer [Barrett et al, 2017]

Measurement bias [Mytkowicz et al, 2009]



Unpredictable and depends of the setup

OR

Avoid measurement bias by adding a source of variability in the benchmark [Tsafrir et al, 2007]

Detect and calculate the impact of the bias to avoid incorrect conclusions

Quantifying Performance (through statistics)

*"For example, we found that **none of the papers** in APLOS 2008, PACT 2007, PLDI 2007, and CGO 2007 address measurement bias satisfactorily."*

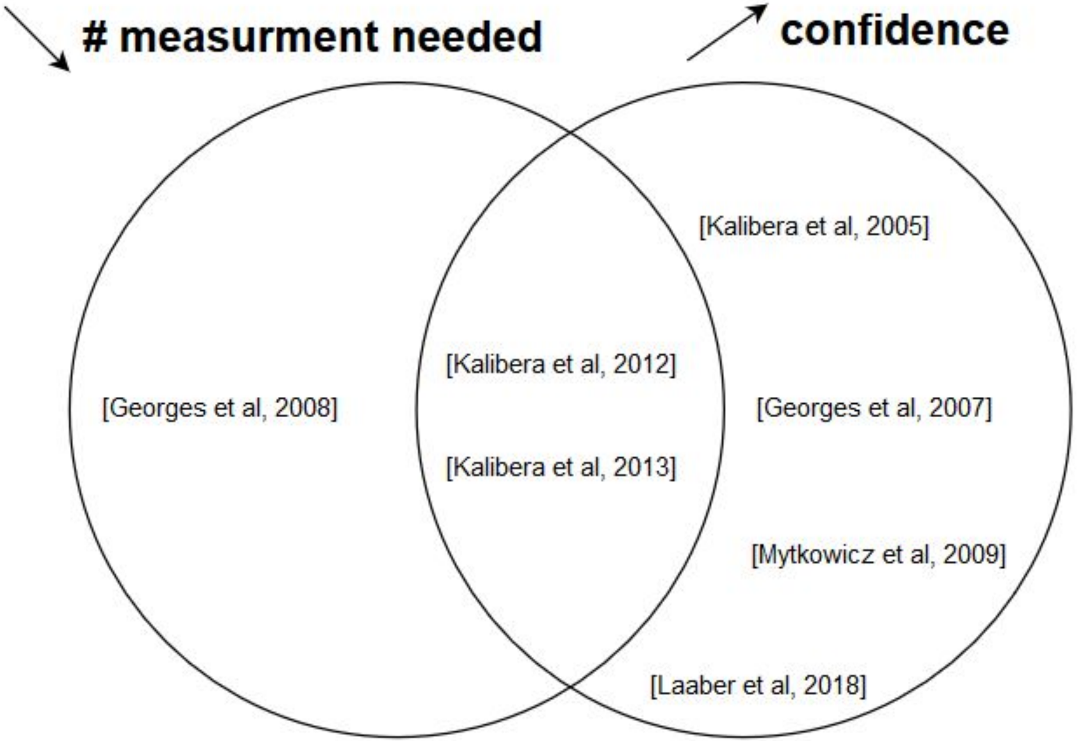
[Mytkowicz et al, 2009]

"71 of these 90 papers completely ignored the question of uncertainty in the measured times."* - [Kalibera et al, 2012]

*about ASPLOS, ISMM, PLDI, TACO, and TOPLAS 2011 papers

In 2019, 30 PLDI papers use benchmarks, but only 9 give a confidence interval. In addition, only 4 explain how they calculated it

Quantifying Performance (through statistics)



↘ # measurment needed

↗ confidence

[Georges et al, 2008]

[Kalibera et al, 2012]

[Kalibera et al, 2013]

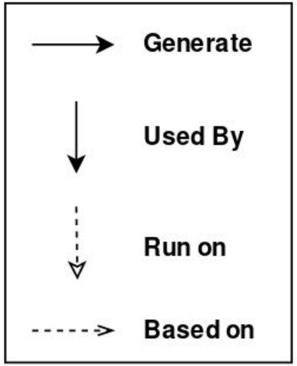
[Kalibera et al, 2005]

[Georges et al, 2007]

[Mytkowicz et al, 2009]

[Laaber et al, 2018]

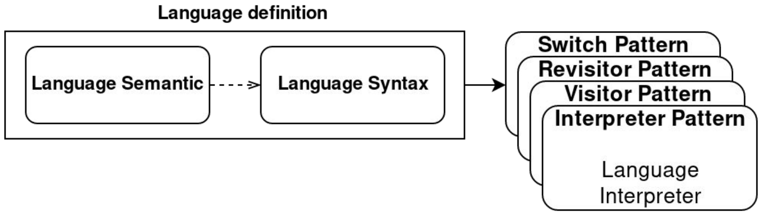
Our Case


Generate

Used By

Run on

Based on


Language definition

Language Semantic

Language Syntax

Switch Pattern

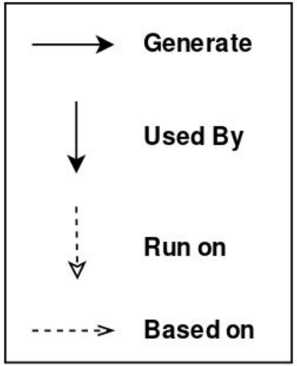
Revisitor Pattern

Visitor Pattern

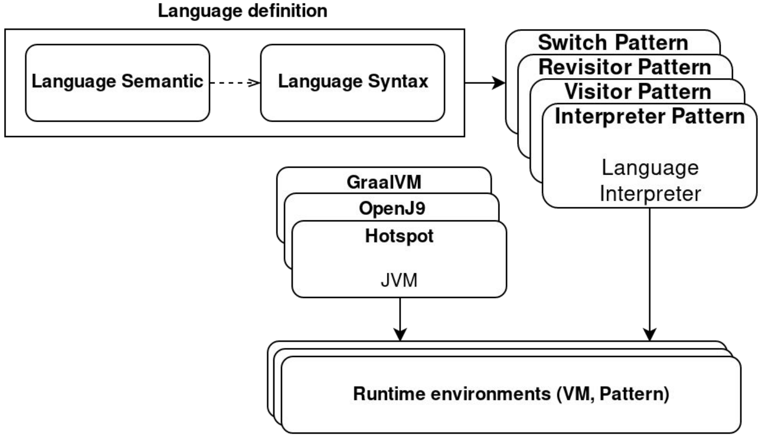
Interpreter Pattern

Language Interpreter

Our Case



Generate
Used By
Run on
Based on



Language definition
Language Semantic
Language Syntax
Switch Pattern
Revisitor Pattern
Visitor Pattern
Interpreter Pattern
Language Interpreter
GraalVM
OpenJ9
Hotspot
JVM
Runtime environments (VM, Pattern)

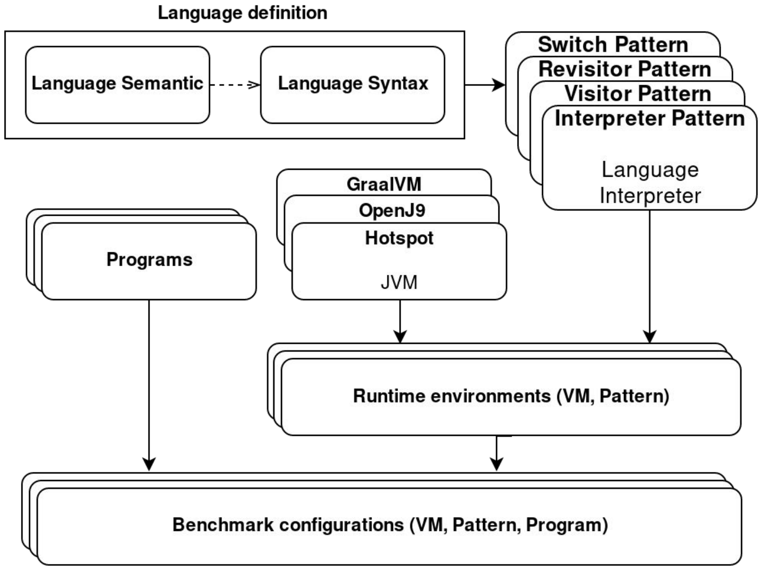
Our Case

→ Generate

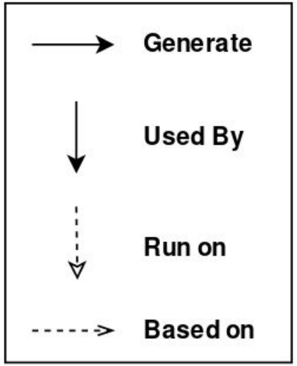
↓ Used By

⇣ Run on

⇢ Based on



Our Case

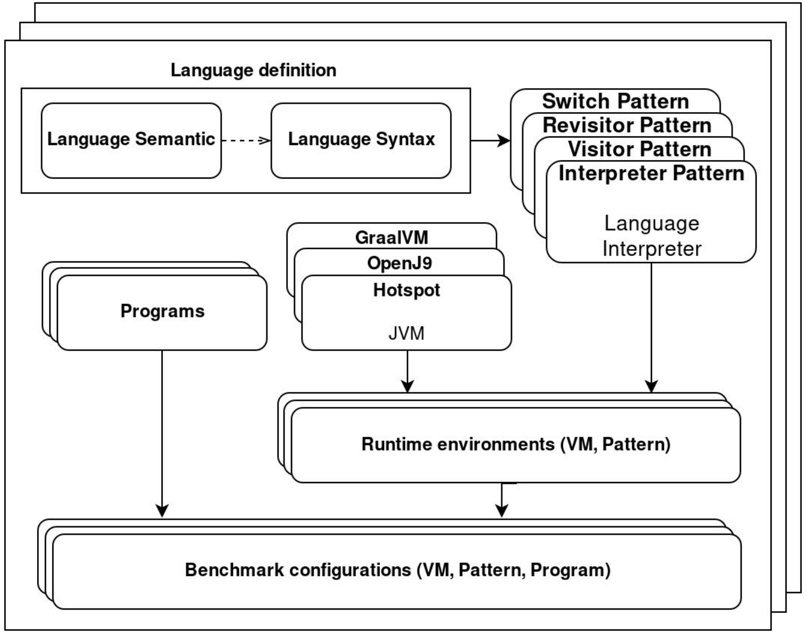


Generate

Used By

Run on

Based on



Language definition

Language Semantic

Language Syntax

Switch Pattern

Revisitor Pattern

Visitor Pattern

Interpreter Pattern

Language Interpreter

GraalVM

OpenJ9

Hotspot

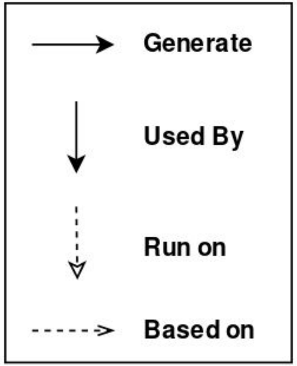
JVM

Programs

Runtime environments (VM, Pattern)

Benchmark configurations (VM, Pattern, Program)

Our Case

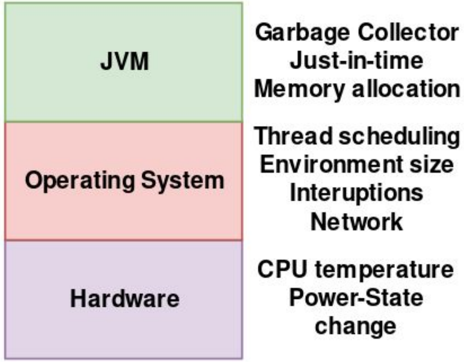


→ Generate

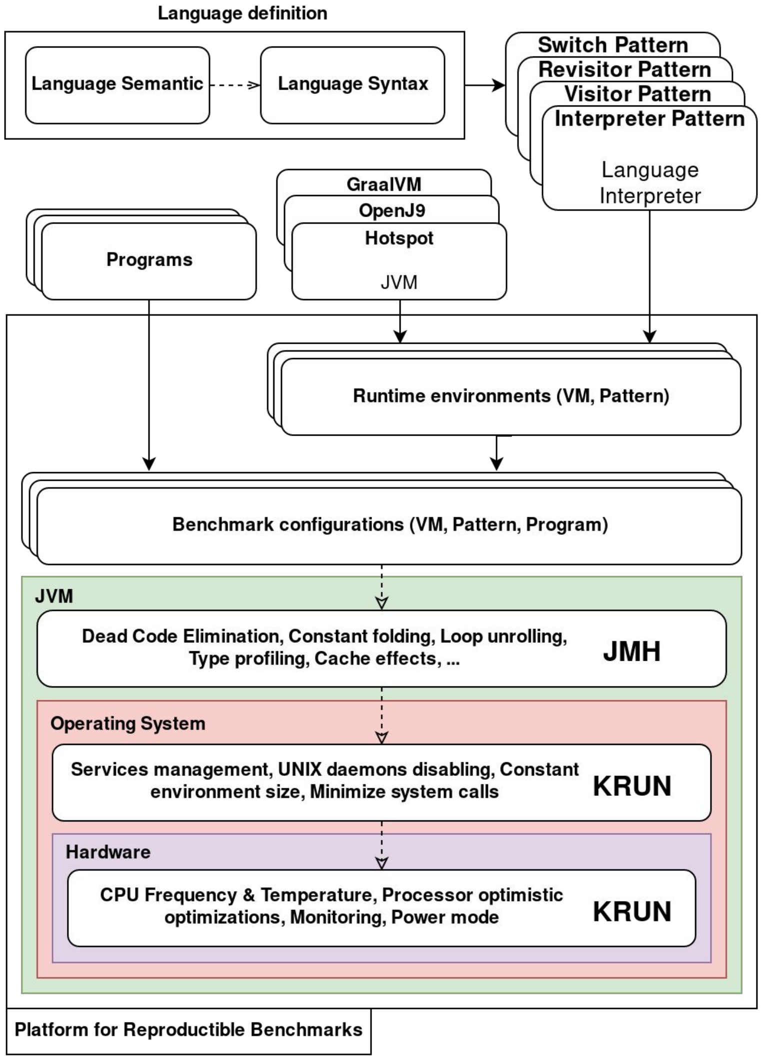
↓ Used By

⇣ Run on

⇢ Based on

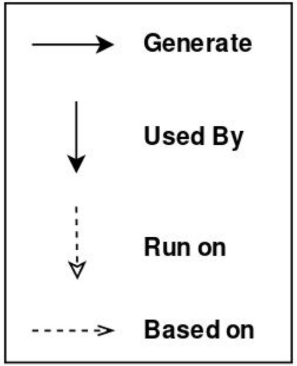


JVM	Garbage Collector Just-in-time Memory allocation
Operating System	Thread scheduling Environment size Interuptions Network
Hardware	CPU temperature Power-State change



Language definition
Language Semantic
Language Syntax
Switch Pattern
Revisitor Pattern
Visitor Pattern
Interpreter Pattern
Language Interpreter
GraalVM
OpenJ9
Hotspot
JVM
Programs
Runtime environments (VM, Pattern)
Benchmark configurations (VM, Pattern, Program)
JVM
Dead Code Elimination, Constant folding, Loop unrolling, Type profiling, Cache effects, ...
JMH
Operating System
Services management, UNIX daemons disabling, Constant environment size, Minimize system calls
KRUN
Hardware
CPU Frequency & Temperature, Processor optimistic optimizations, Monitoring, Power mode
KRUN
Platform for Reproductible Benchmarks

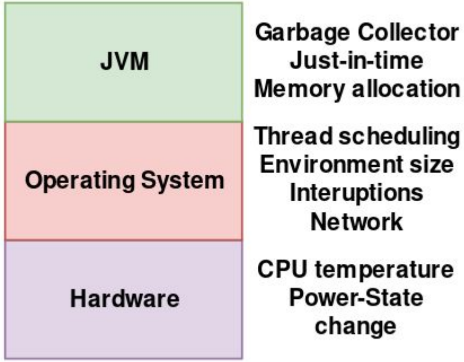
Our Case


Generate

Used By

Run on

Based on


JVM

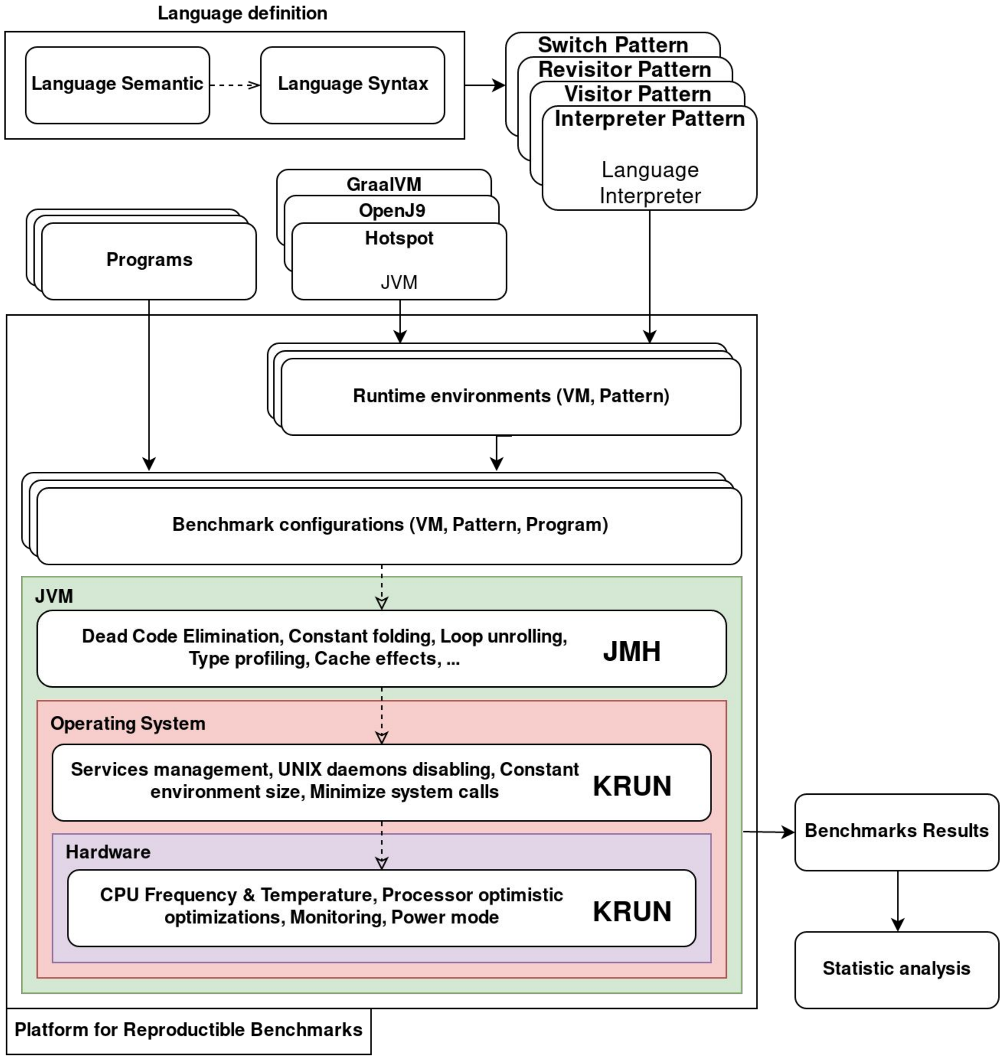
Garbage Collector
Just-in-time
Memory allocation

Operating System

Thread scheduling
Environment size
Interuptions
Network

Hardware

CPU temperature
Power-State
change


Language definition

Language Semantic

Language Syntax

Switch Pattern

Revisitor Pattern

Visitor Pattern

Interpreter Pattern

Language Interpreter

Programs

GraalVM

OpenJ9

Hotspot

JVM

Runtime environments (VM, Pattern)

Benchmark configurations (VM, Pattern, Program)

JVM

Dead Code Elimination, Constant folding, Loop unrolling, Type profiling, Cache effects, ...

JMH

Operating System

Services management, UNIX daemons disabling, Constant environment size, Minimize system calls

KRUN

Hardware

CPU Frequency & Temperature, Processor optimistic optimizations, Monitoring, Power mode

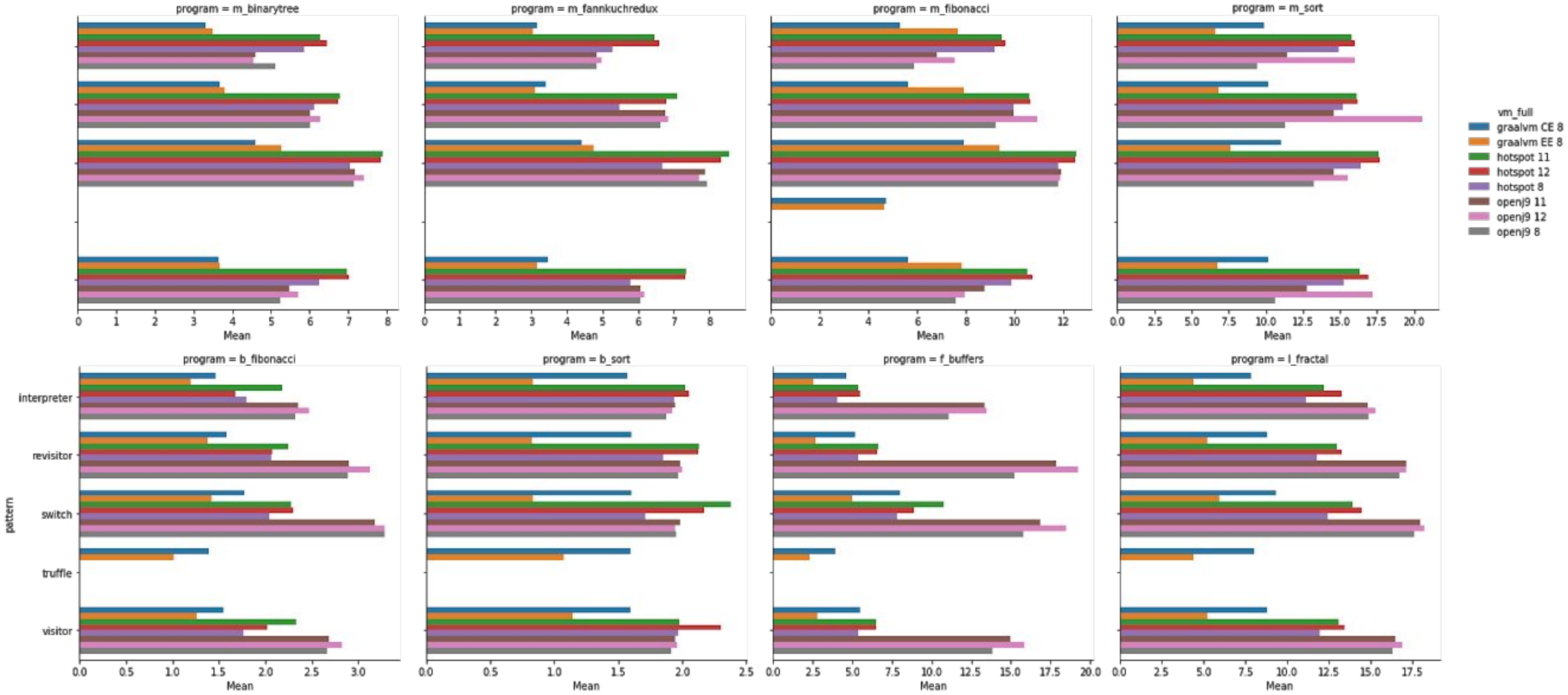
KRUN

Benchmarks Results

Statistic analysis

Platform for Reproductible Benchmarks

Results


program = m_binarytree
program = m_fannkuchredux
program = m_fibonacci
program = m_sort
program = b_fibonacci
program = b_sort
program = f_buffers
program = l_fractal
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 2 4 6 8 10 12
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0.0 0.5 1.0 1.5 2.0 2.5
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Mean
pattern
interpreter
revisitor
switch
truffle
visitor
vm_full
graalvm CE 8
graalvm EE 8
hotspot 11
hotspot 12
hotspot 8
openj9 11
openj9 12
openj9 8

What to take home ?

➔ Now, you should be doubtful about your benchmark’s results

- (Especially) When they are pleasant
- (Even) When they are unpleasant

- Reliable results implies rigorous methodologies
- All precautions are necessary but not sufficient …
- So use statistics to quantify the confidence of your results

References

A.Shipilev,“Java microbenchmarks harness (the lesser of two evils),”2013.

A. Shipilev, “Nanotrusting the nanotime,” 2014.

M. Rodriguez-Cancio, B. Combemale, and B. Baudry, “Automatic microbenchmark generation to prevent dead code elimination and constant folding,” in 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2016, pp. 132–143.

M. Arnold, S. J. Fink, D. Grove, M. Hind, and P. F. Sweeney, “A survey of adaptive optimization in virtual machines,” Proceedings of the IEEE , vol. 93, no. 2, pp. 449–466, 2005.

S. M. Blackburn, P. Cheng, and K. S. McKinley, “Myths and realities: The performance impact of garbage collection,” in ACM SIGMETRICS Performance Evaluation Review , vol. 32, no. 1. ACM, 2004, pp. 25–36.

A. Georges, L. Eeckhout, and D. Buytaert, “Java performance evaluation through rigorous replay compilation,” in ACM Sigplan Notices, vol. 43, no. 10. ACM, 2008, pp. 367–384

E. Barrett, C. F. Bolz-Tereick, R. Killick, S. Mount, and L. Tratt, “Virtual machine warmup blows hot and cold,” Proceedings of the ACM on Programming Languages , vol. 1, no. OOPSLA, p. 52, 2017.

References

C. Laaber, J. Scheuner, and P. Leitner, “Performance testing in the cloud. how bad is it really?” PeerJ PrePrints, vol. 6, p. e3507v1, 2018.

T. Kalibera, L. Bulej, and P. Tuma, “Benchmark precision and random initial state,” in Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005) , 2005, pp. 484–490

T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, “Producing wrong data without doing anything obviously wrong!” ACM SIGARCH Computer Architecture News, vol. 37, no. 1, pp. 265–276, 2009

D. Tsafrir, K. Ouaknine, and D. G. Feitelson, “Reducing performance evaluation sensitivity and variability by input shaking,” in 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE, 2007, pp. 231–237

T. Kalibera and R. Jones, “Quantifying performance changes with effect size confidence intervals,” Citeseer, Tech. Rep., 2012

T. Kalibera and R. Jones, “Rigorous benchmarking in reasonable time,” ACM SIGPLAN Notices, vol. 48, no. 11, pp.63–74, 2013

A. Georges, D. Buytaert, and L. Eeckhout, “Statistically rigorous java performance evaluation,” in ACM SIGPLAN Notices, vol. 42, no. 10. ACM, 2007, pp. 57–76.

Starter pack

For JVM

- A.Shipilev,“Java microbenchmarks harness (the lesser of two evils),”2013.

For Operating System and Hardware

- E. Barrett, C. F. Bolz-Tereick, R. Killick, S. Mount, and L. Tratt, “Virtual machine warmup blows hot and cold,” Proceedings of the ACM on Programming Languages , vol. 1, no. OOPSLA, p. 52, 2017.

For measurement bias

- T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, “Producing wrong data without doing anything obviously wrong!” ACM SIGARCH Computer Architecture News, vol. 37, no. 1, pp. 265–276, 2009

For statistics

- A. Georges, D. Buytaert, and L. Eeckhout, “Statistically rigorous java performance evaluation,” in ACM SIGPLAN Notices, vol. 42, no. 10. ACM, 2007, pp. 57–76.
- C. Laaber, J. Scheuner, and P. Leitner, “Performance testing in the cloud. how bad is it really?” PeerJ PrePrints, vol. 6, p. e3507v1, 2018.