A Sampling Profiler for JITing in Ř

Masters Presentation II

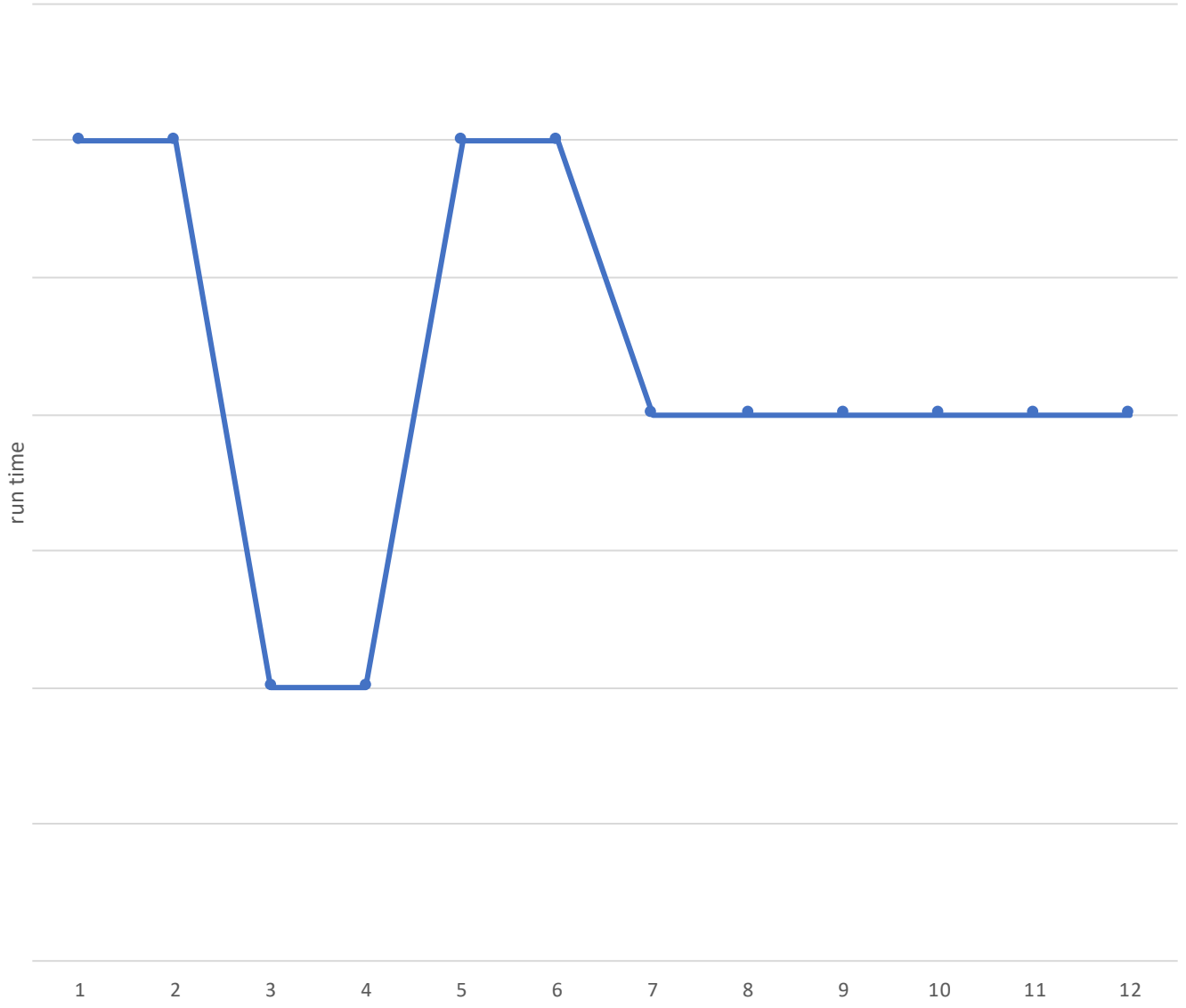
Andreas Wälchli

Supervisor: Olivier Flückiger

26. 05. 2020

Problem: Polluted Type Feedback in JIT-Compiler

```
a <- 1
f <- function() { a + 1 }
f()
...
a <- 1L
f()
...
a <- 1
f()
...
```



Compilation with Dynamic Typing

- Problem: no good type information from static analysis
  - need to run code to get type information
  - types may change between executions

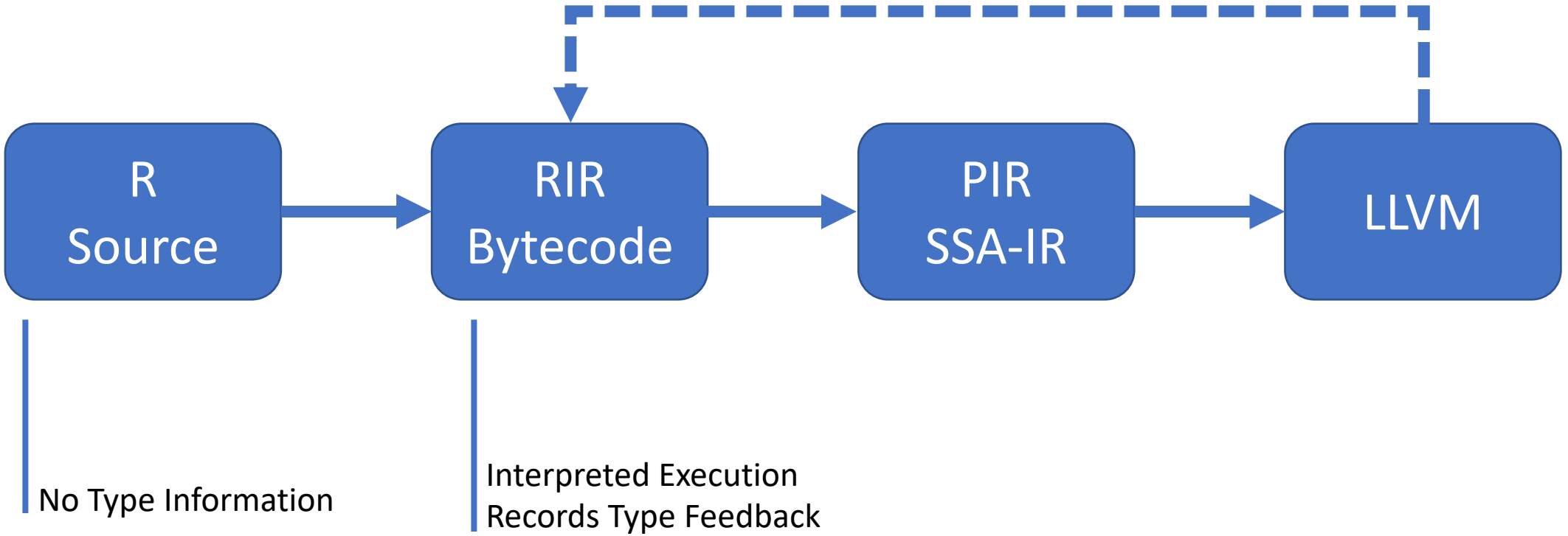
- Solution: Warm-Up Phase
  - run code in interpreter
  - record type information during execution
  - compile with collected type information after several runs

Research Question

- (How) Can a sampling profiler be used to improve JIT-compiler output?

- Approach:
  - detect LLVM compiled code and sample dynamic types at runtime
  - trigger recompilation when sufficient data is collected

Ř Compilation Pipeline



Sampling Values for Optimisation

Sampling Profiler

- Stop program execution at regular intervals
- Sample the program state
- Usually uses signals and signal handlers

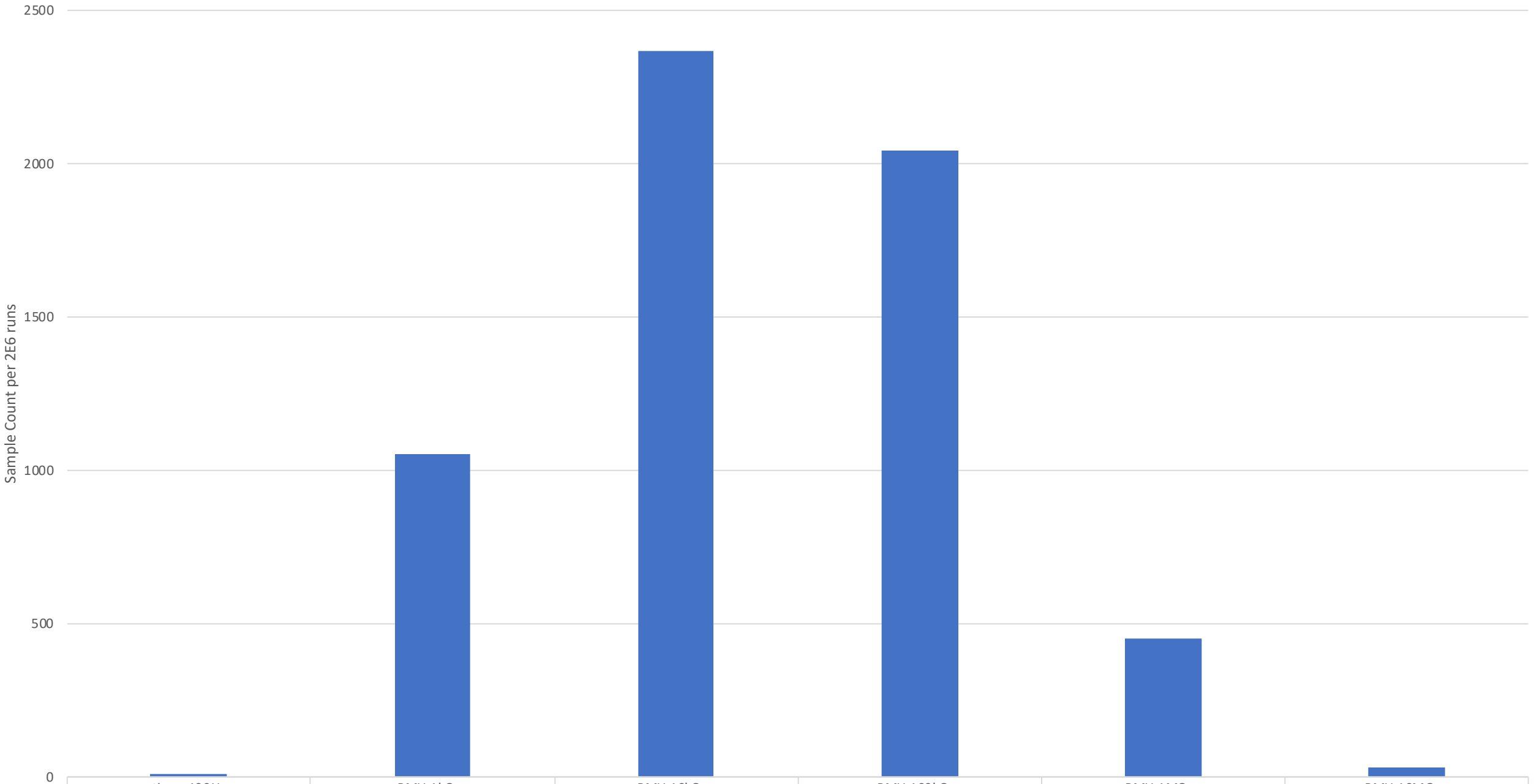
Sample Triggering

POSIX Timer

- Precise Timing
- generate a SIGNAL on timeout
- can deadlock with SYSCALL at high frequency (>100Hz)

Performance Measuring Unit (PMU)

- Imprecise Triggering
- generate a SIGNAL after n Instructions in User Mode
- Sampling Rate not consistent between different CPUs
- Interference with SYSCALL unlikely
- Higher Possible Sample Rate

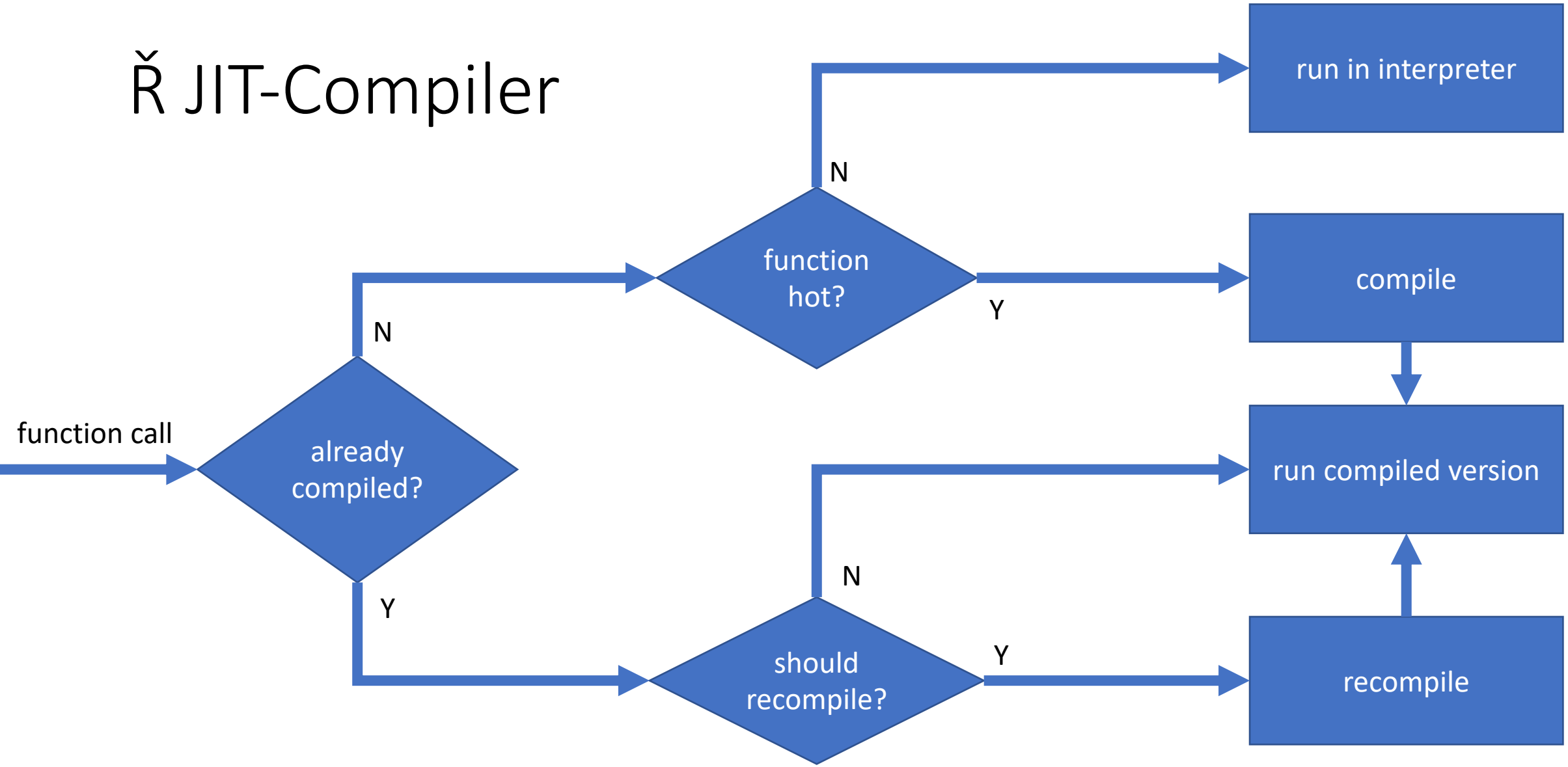


	timer 100Hz	PMU 1kOps	PMU 10kOps	PMU 100kOps	PMU 1MOps	PMU 10MOps
samples	7	1055	2366	2045	453	33

Sample Collection

- Boxed Values live on a Stack
- Inspect the top-most Stack Frame
- We have a Mapping from Stack Slots to TypeFeedback Entries
- When sampling, take any live Slots and fill their TypeFeedback Entry

Ř JIT-Compiler



Triggering Compilation

- When is a function ready for recompilation?
  - At least 50% of slots have at least 10 samples
  - At least one “ready” slot has more precise feedback than during compilation
  - Discard entry after 100 samples (prevent pollution)

- When compiling, the compiler takes ”our” feedback for every slot that is “ready” (even when inlining the function somewhere else)

Results

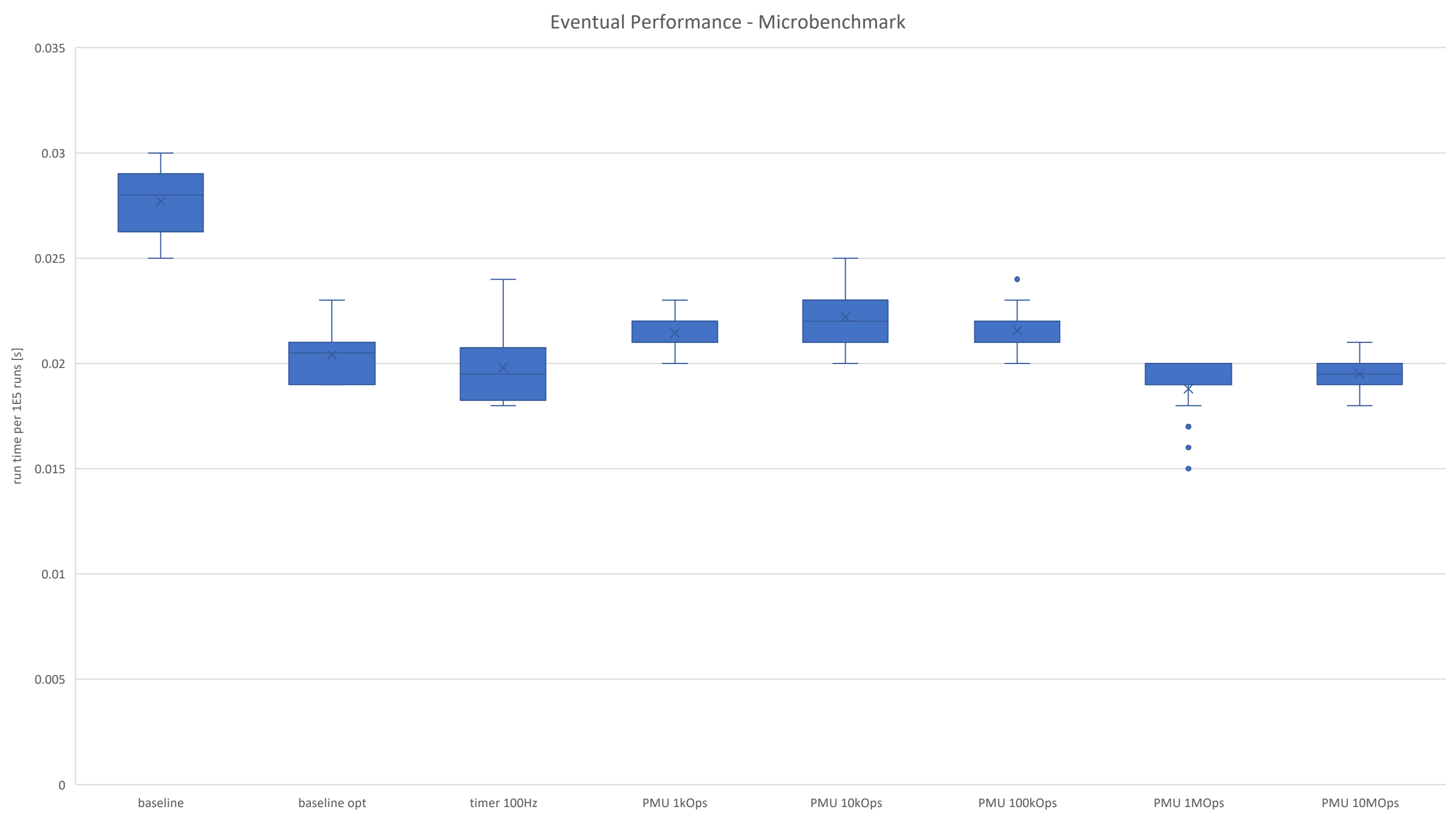
2 Examples

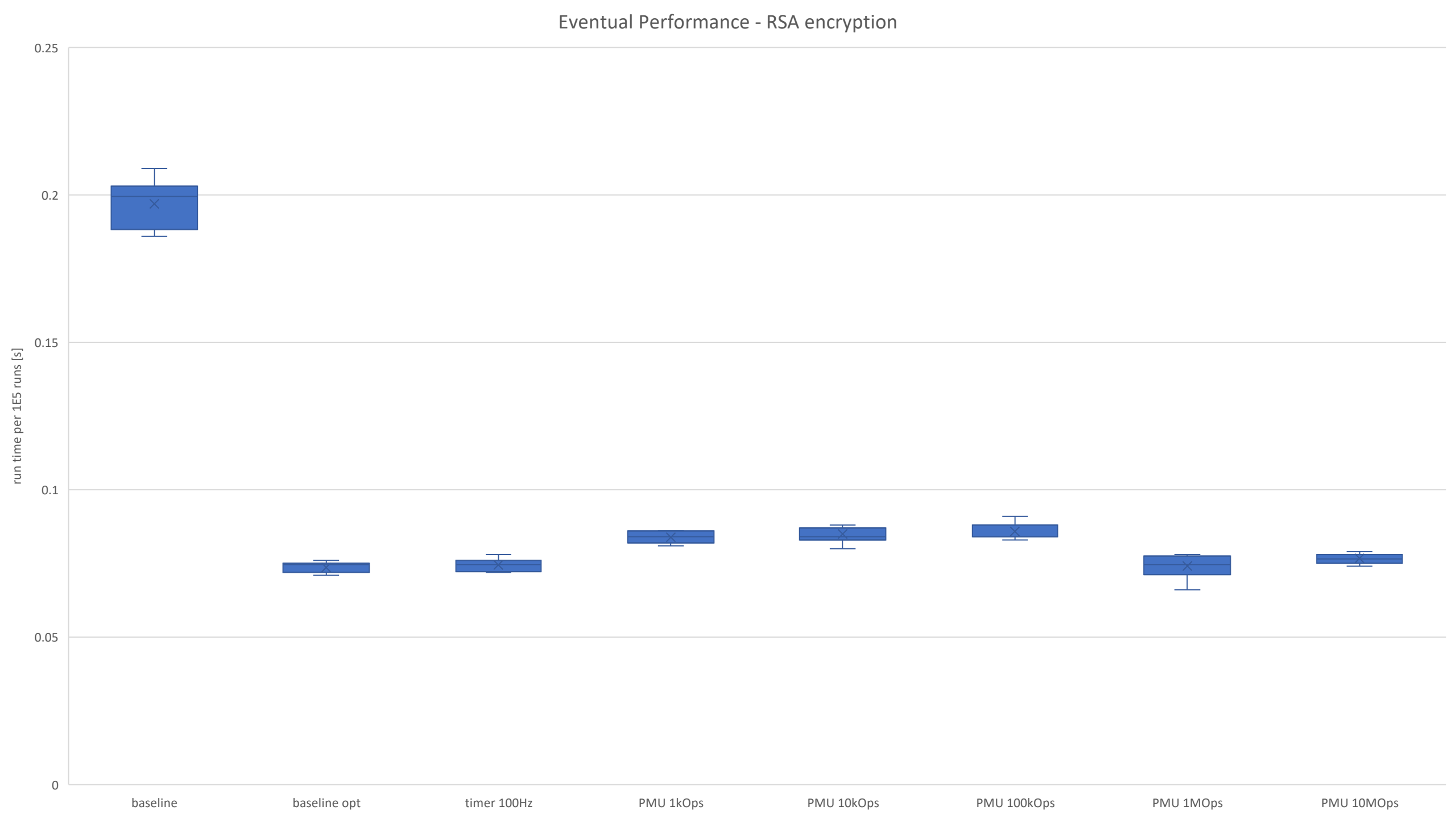
Microbenchmark

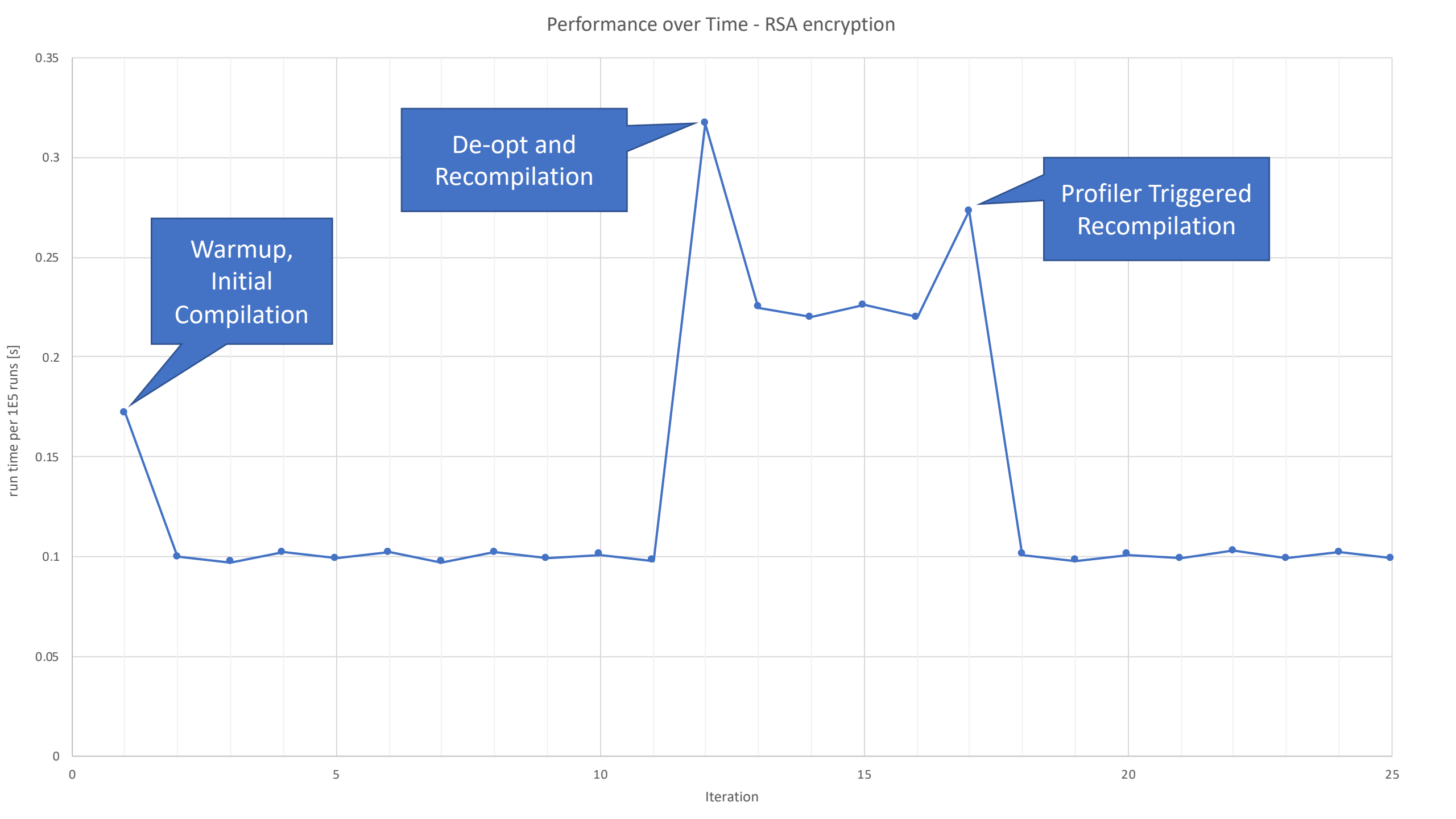
```
a <- 1

f <- function() a+a+a+a+1L

f()
f()

a <- 1L

f()
f()

system.time(for (i in 1:100000) f())
```

RSA Encryption

```
p1 <- 971
p2 <- 383
n <- p1 * p2
e <- 17

encrypt <- function(msg) {
  p <- 1
  a1 <- msg
  for(i in 1:e) {
    p <- p*a1
    p <- p%%n
    i <- i+1
  }
  p
}

system.time(for(i in 1:100000) encrypt(i))

n <- 371893L

system.time(for(i in 1:100000) encrypt(i))
```


Eventual Performance - Microbenchmark
run time per 1E5 runs [s]
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
baseline
baseline opt
timer 100Hz
PMU 1kOps
PMU 10kOps
PMU 100kOps
PMU 1MOps
PMU 10MOps


Eventual Performance - RSA encryption
run time per 1E5 runs [s]
0.25
0.2
0.15
0.1
0.05
0
baseline
baseline opt
timer 100Hz
PMU 1kOps
PMU 10kOps
PMU 100kOps
PMU 1MOps
PMU 10MOps


Performance over Time - RSA encryption
Warmup, Initial Compilation
De-opt and Recompilation
Profiler Triggered Recompilation
run time per 1E5 runs [s]
Iteration
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25

Demo – RSA Encryption

Conclusion

- A Sampling profiler can improve performance in JIT-compiled code
- Large gains with type changes in global variables

Ř Compilation Pipeline