On Performing Accurate Time Measurements of SGX Enclave Instructions

Miro Haller

Advisors: Prof. Dr. S. Capkun, I. Puddu, M. Schneider

Motivation

1Password

Tesseract

IBM Cloud

Azure

Attacks on SGX Enclaves

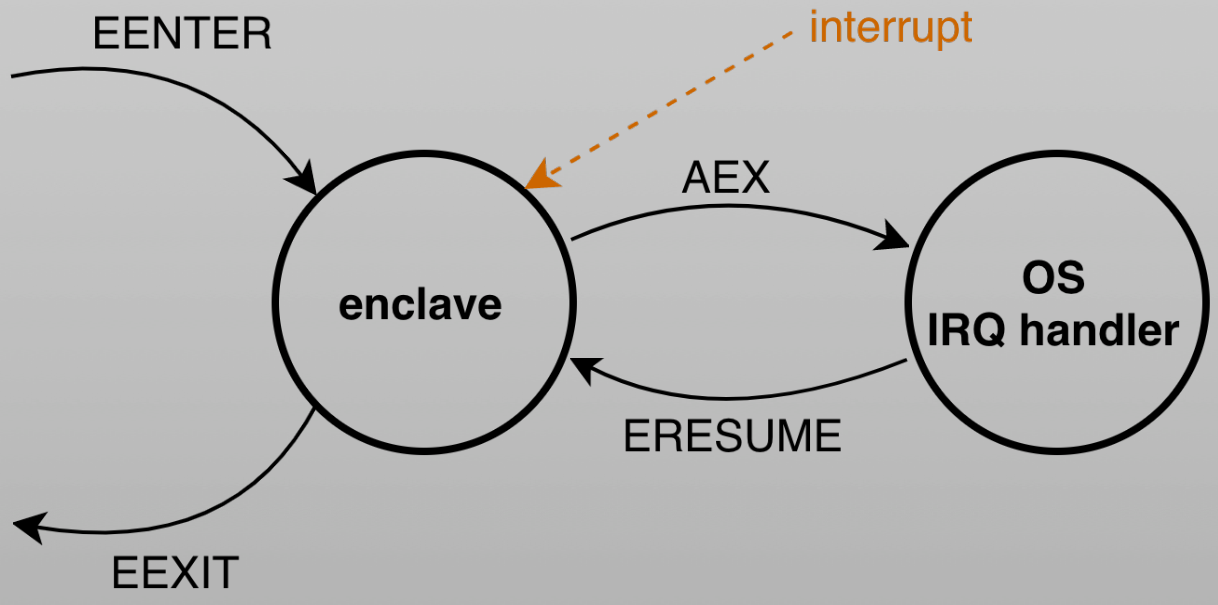
- Cache side channel (Software Grand Exposure)
- Interrupt side channel (Nemesis)
- Foreshadow

Attacks on SGX Enclaves

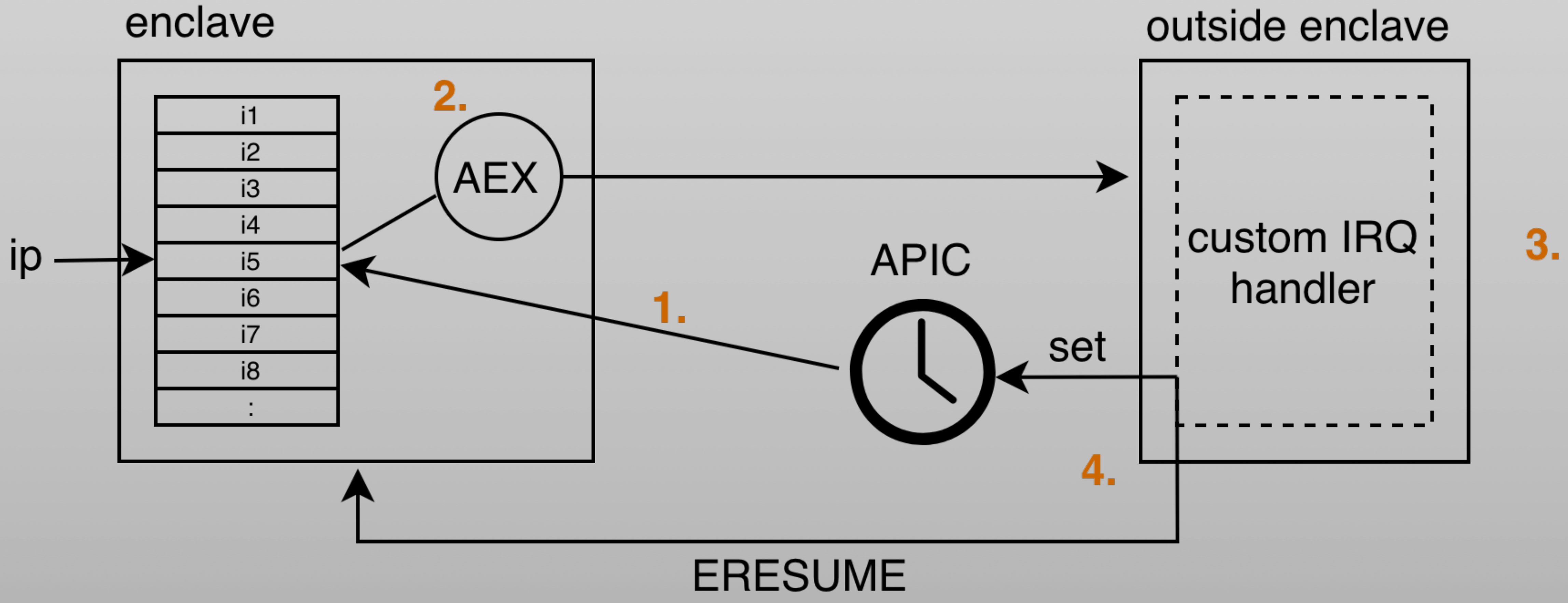
- Cache side channel (Software Grand Exposure)
- **Interrupt side channel** (Nemesis)
- Foreshadow

Background – SGX Enclaves

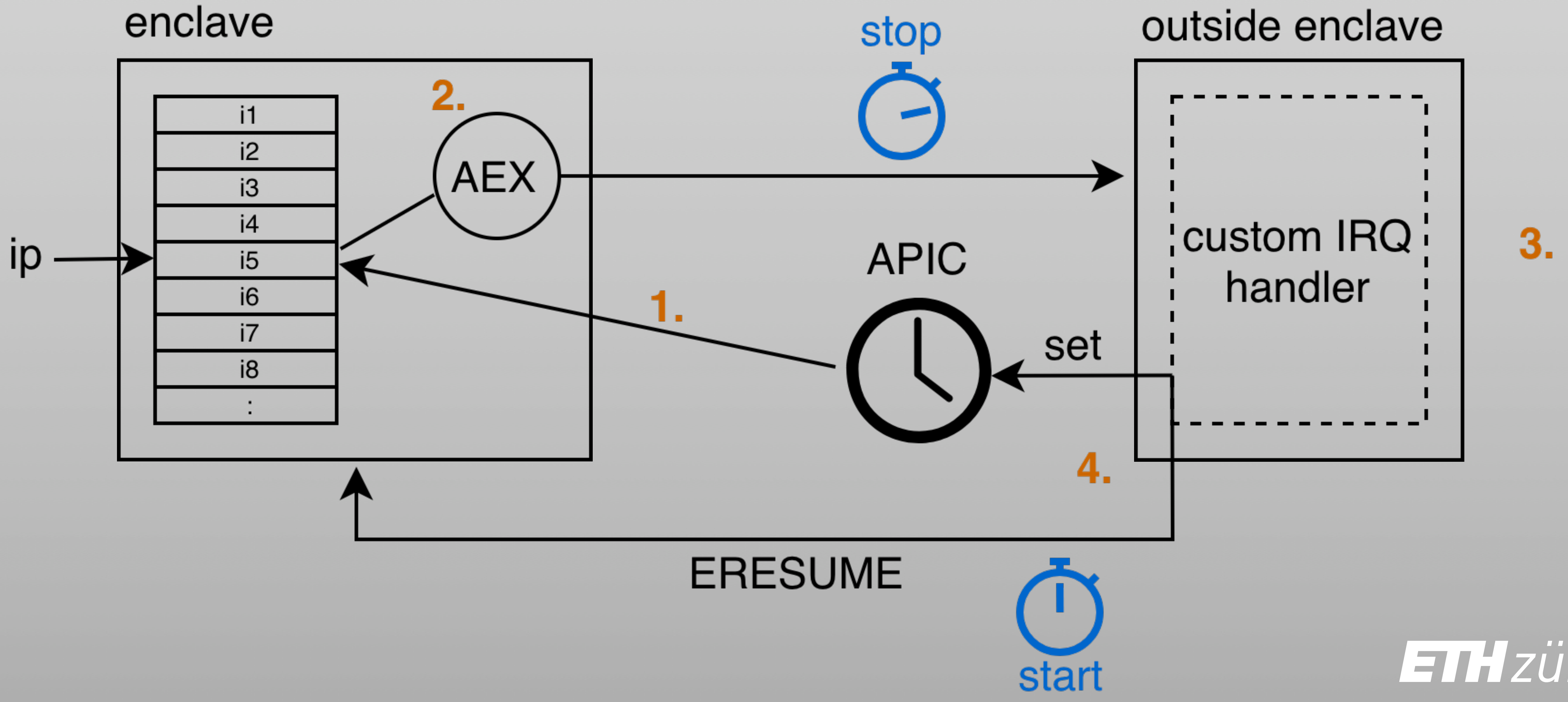
- Threat model
- SGX enclave life cycle



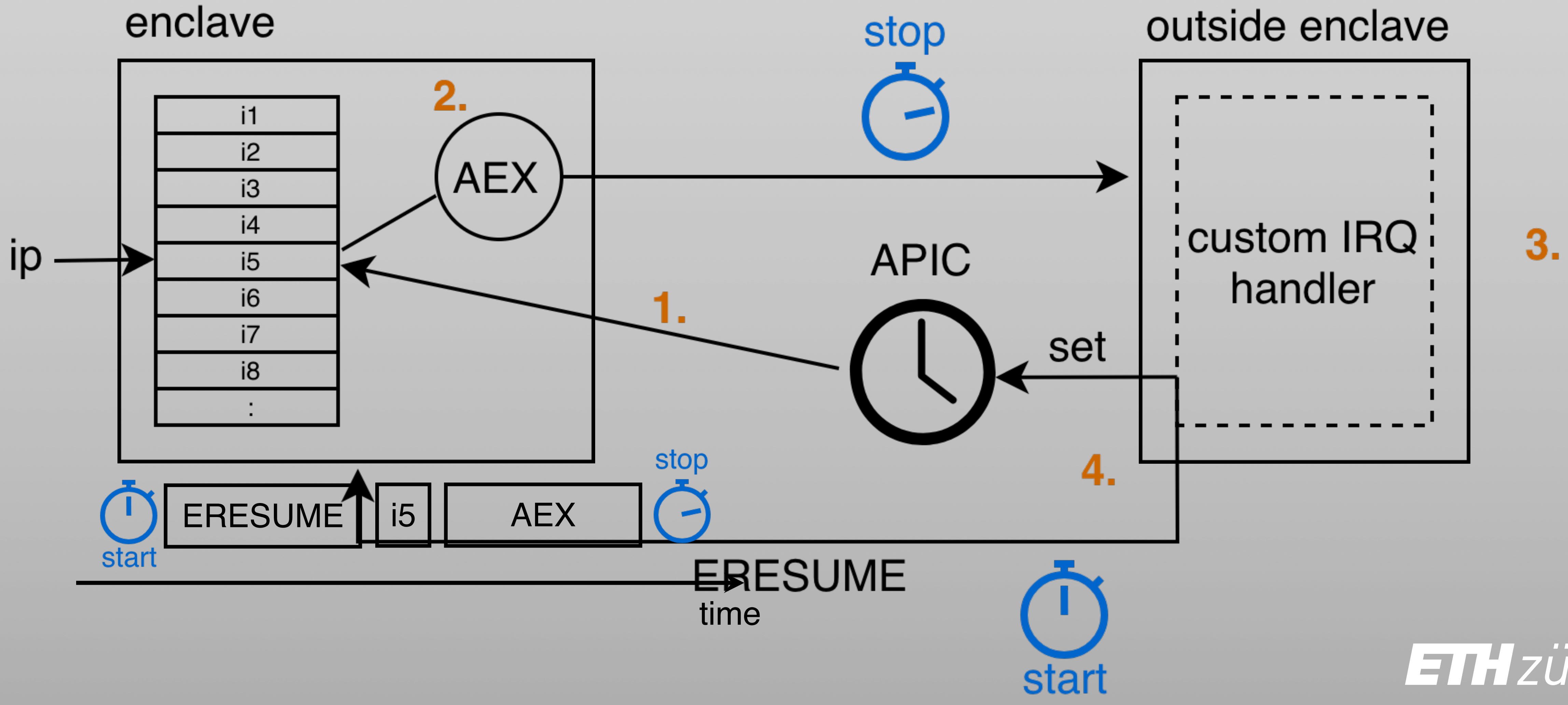
Background – SGX-Step



Background – SGX-Step

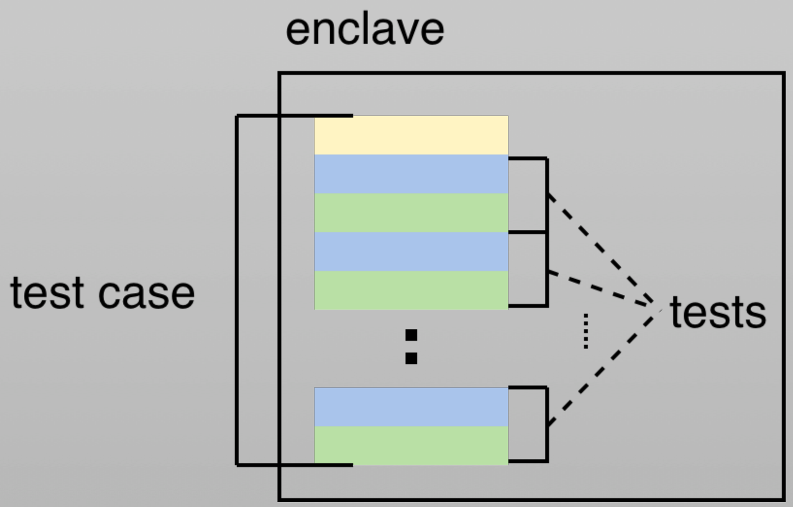


Background – SGX-Step



Measurements

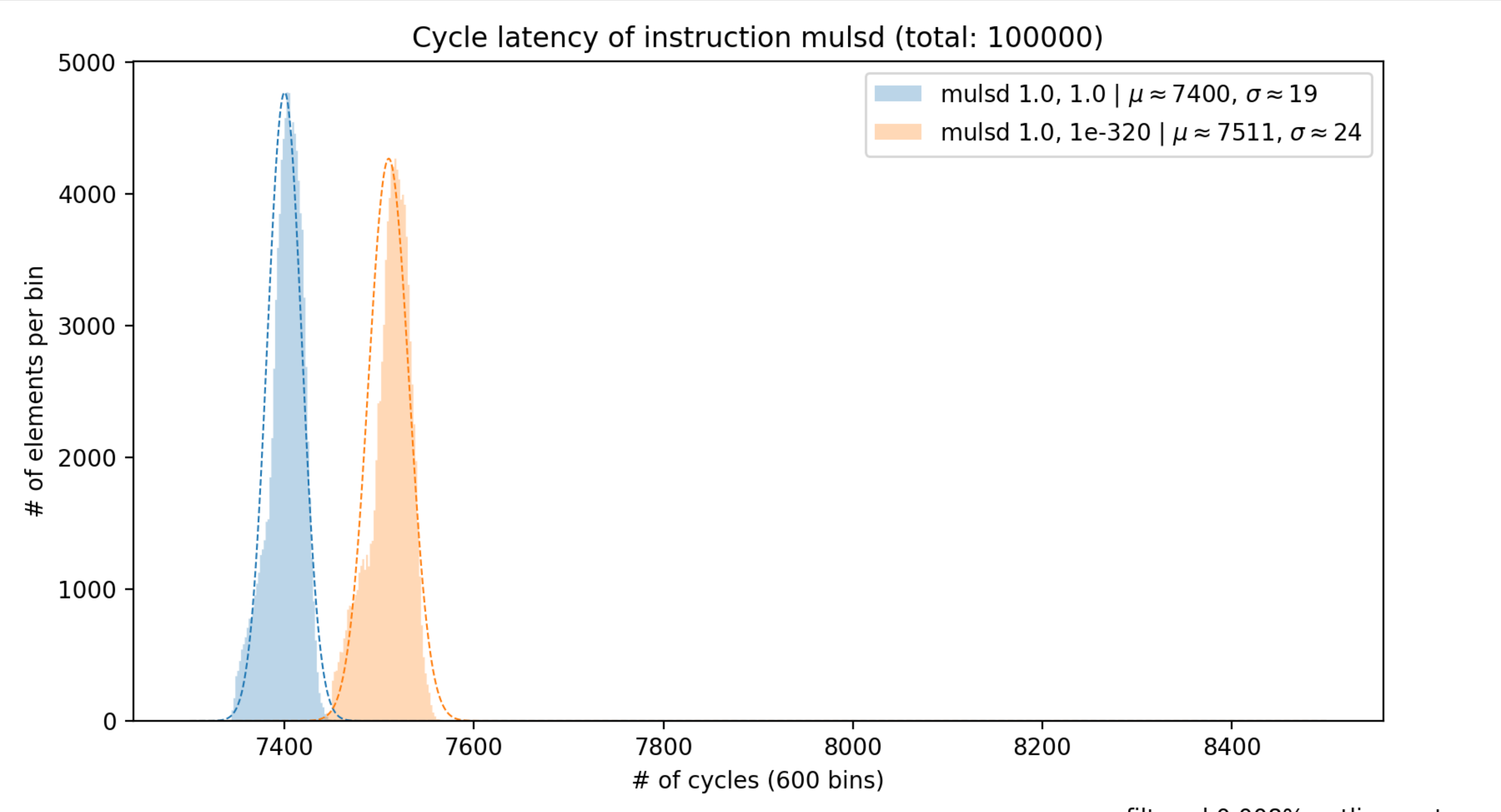
- Reduce interference
  - Hyper-Threading, dyn. frequency scaling, isolated core
- Terminology



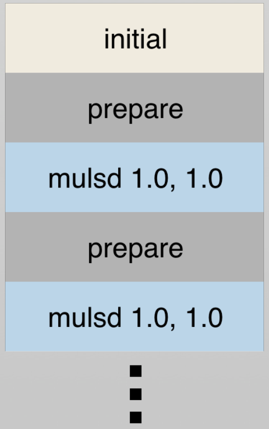
Legend:

- 🟨 initial instructions
- 🟦 prepare instructions
- 🟩 test instruction

Measurements – Example



filtered 0.008% outliers out



initial

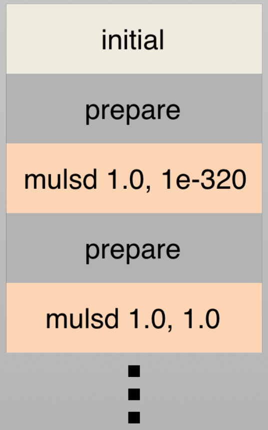
prepare

mulsd 1.0, 1.0

prepare

mulsd 1.0, 1.0

⋮



initial

prepare

mulsd 1.0, 1e-320

prepare

mulsd 1.0, 1.0

⋮

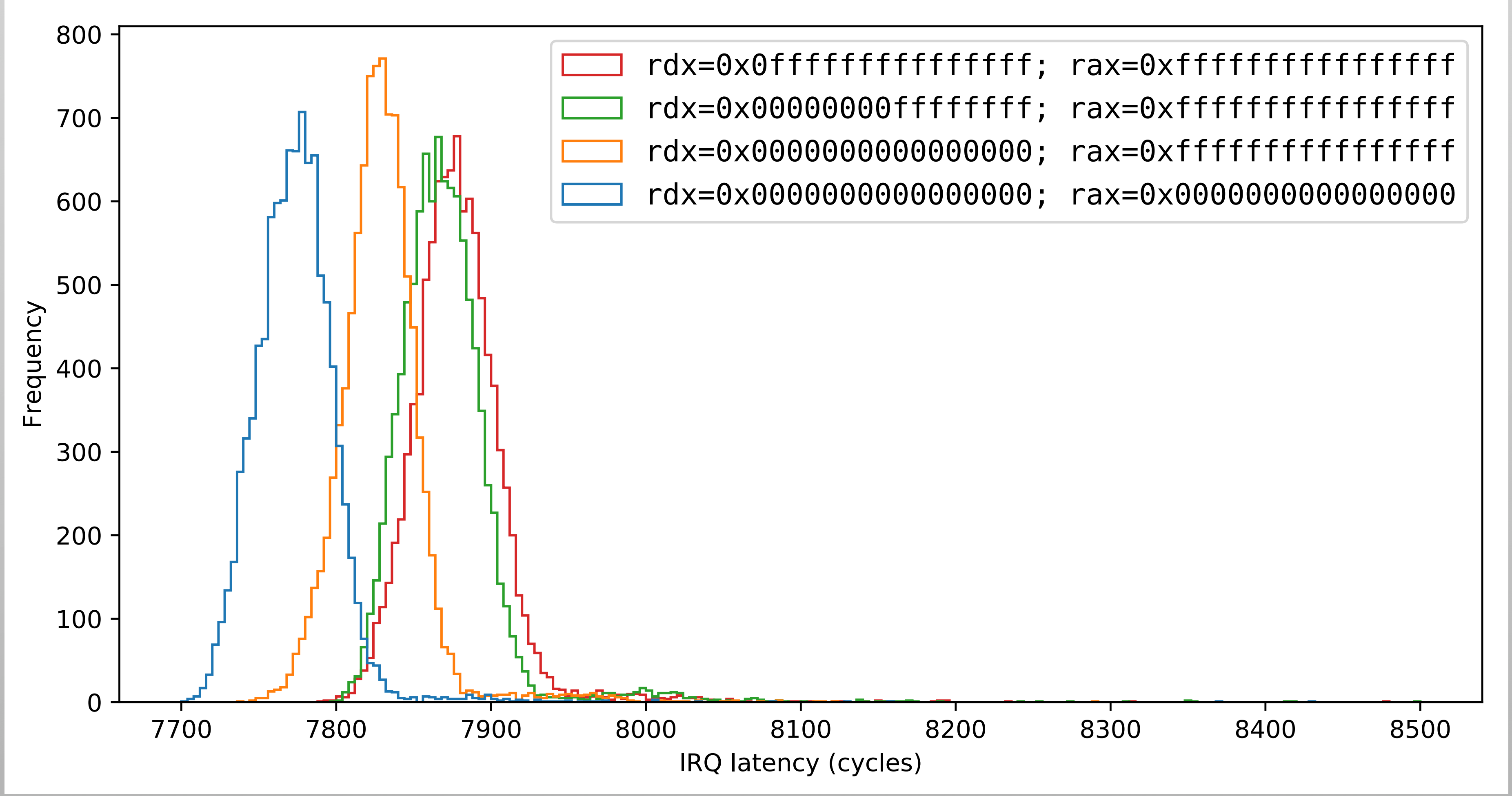
Challenges

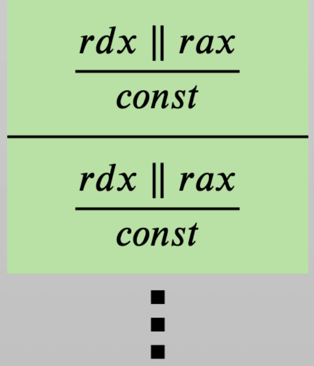
1. Instruction tracking
2. Page borders
3. Cache conflicts
4. Incomparable enclaves
5. Constant time code
6. Verifying tests
7. Build advanced test cases
8. Two sources of noise
9. Synthetic state on AEX

Challenges

1. Instruction tracking
2. Page borders
3. Cache conflicts
4. **Incomparable enclaves**
5. Constant time code
6. Verifying tests
7. Build advanced test cases
8. Two sources of noise
9. Synthetic state on AEX

Challenges – Incomparable Enclaves

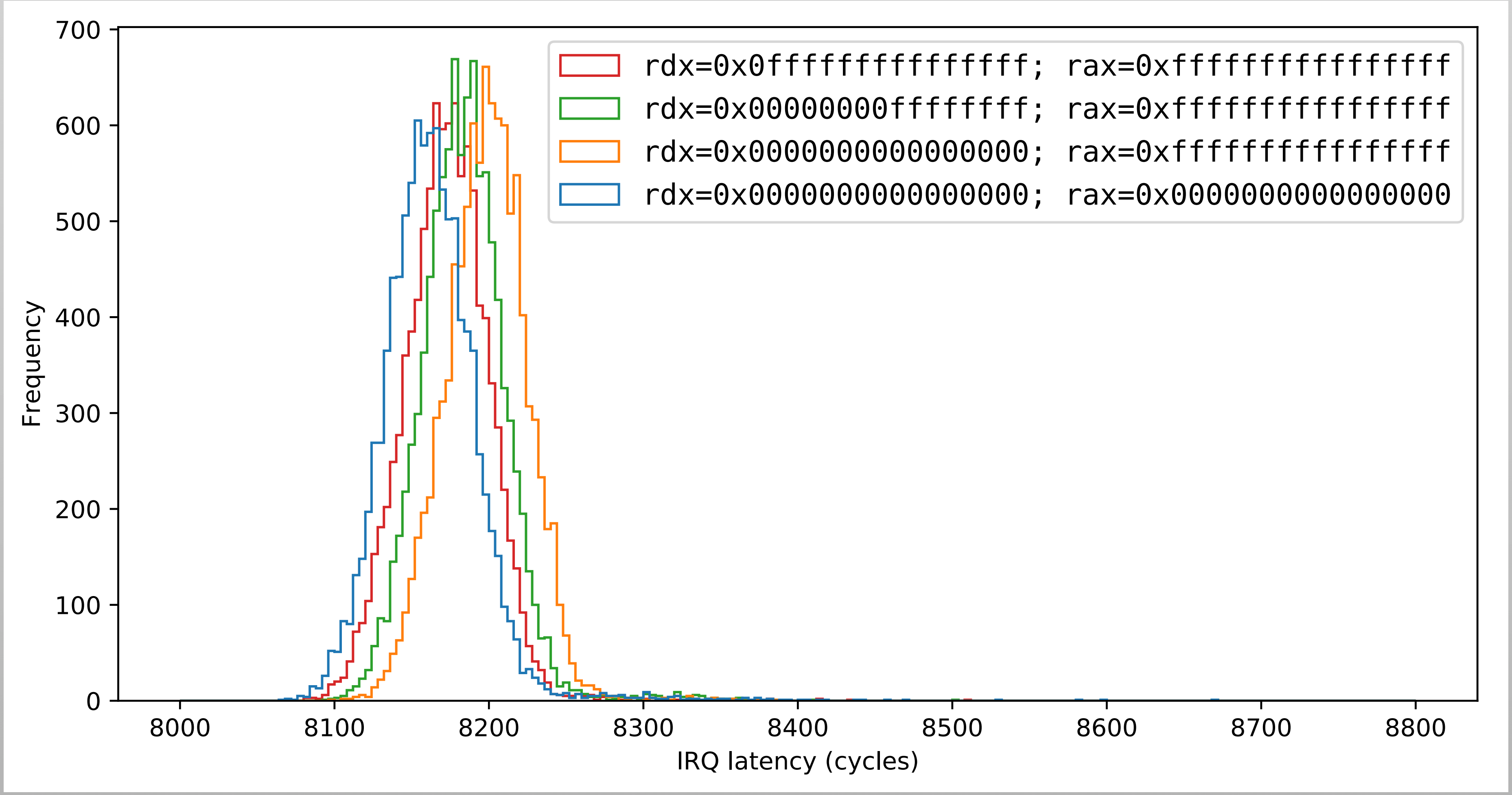




Source: Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In *ACM Conference on Computer and Communications Security*, 2018.

ETHzürich

Challenges – Incomparable Enclaves

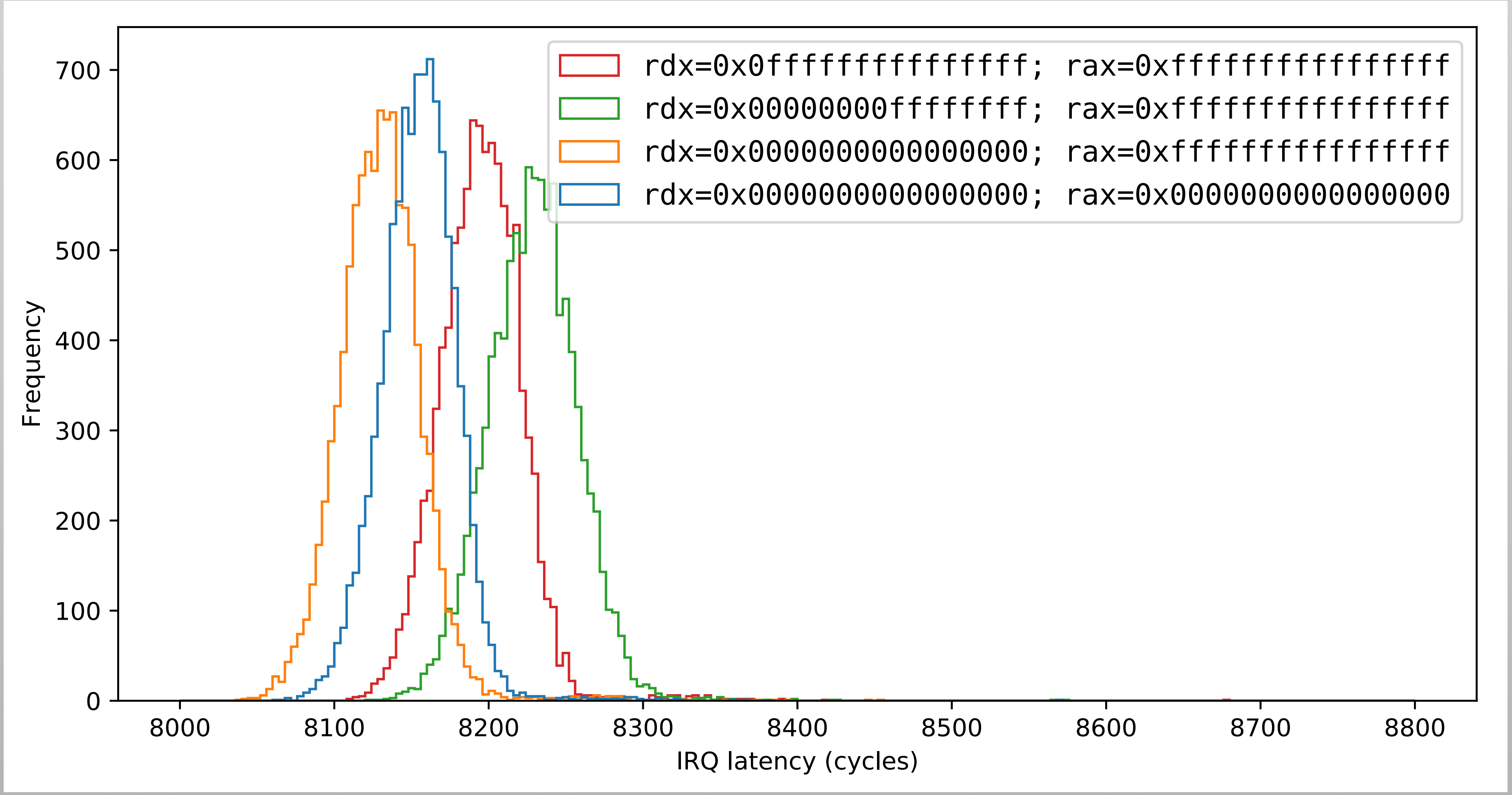


$$\frac{rdx \parallel rax}{const}$$

$$\frac{rdx \parallel rax}{const}$$

⋮

Challenges – Incomparable Enclaves

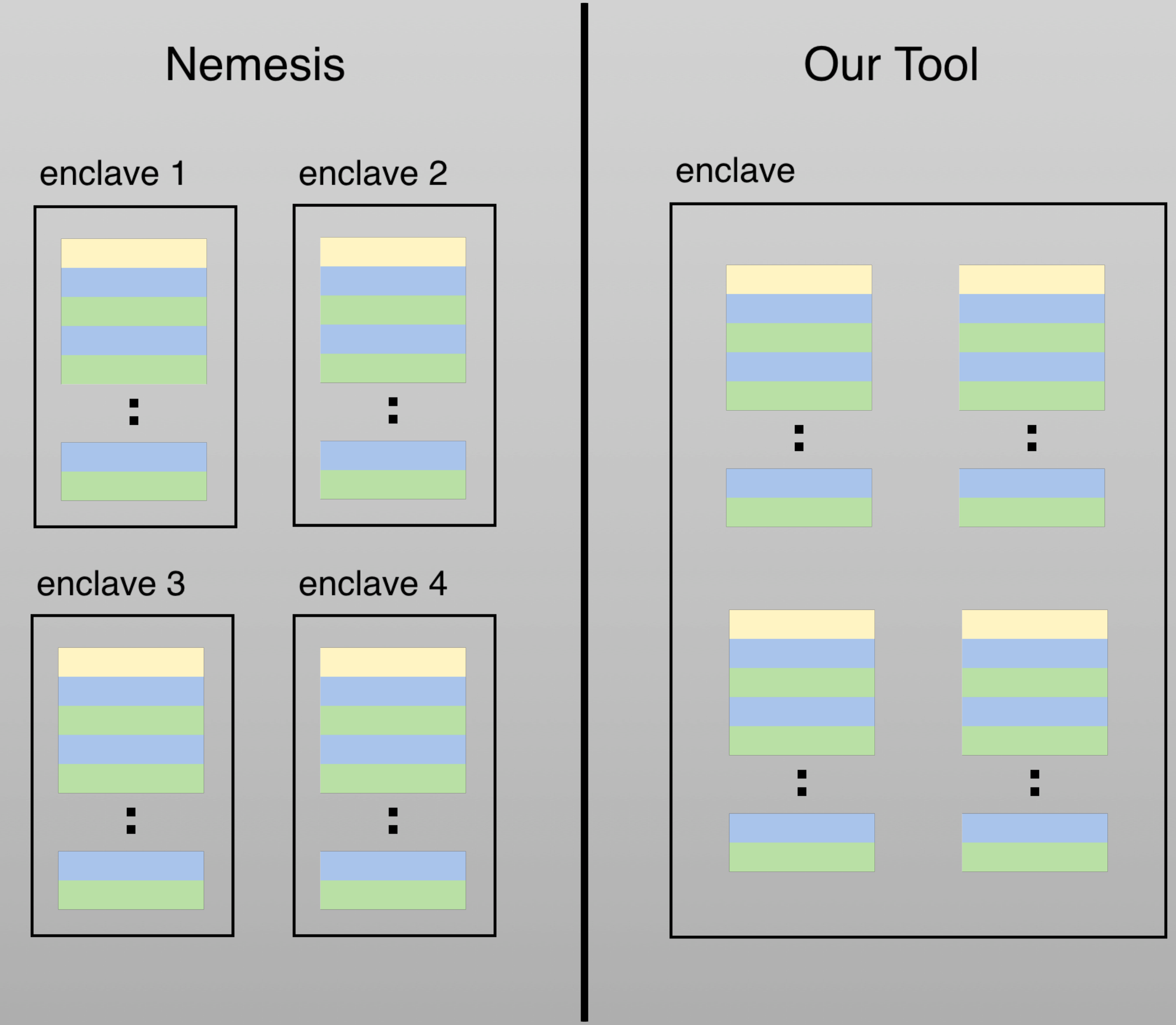


$$\frac{rdx \,\|\, rax}{const}$$

$$\frac{rdx \,\|\, rax}{const}$$

$$\vdots$$

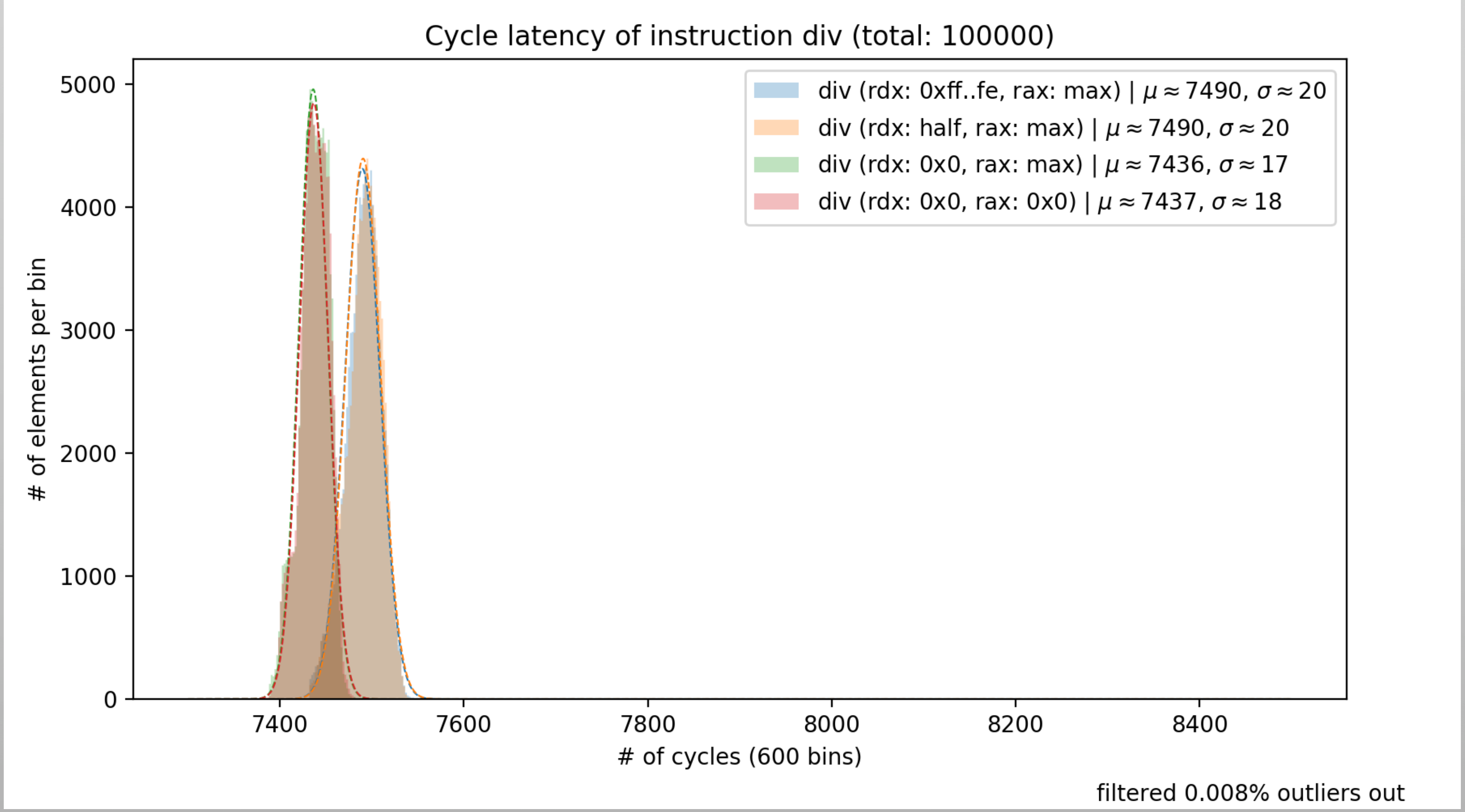
Challenges – Incomparable Enclaves

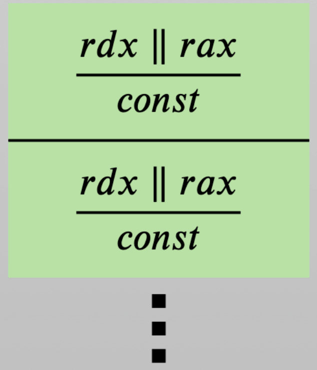


Legend:

🟨 initial instructions
🟦 prepare instructions
🟩 test instruction

Challenges – Incomparable Enclaves



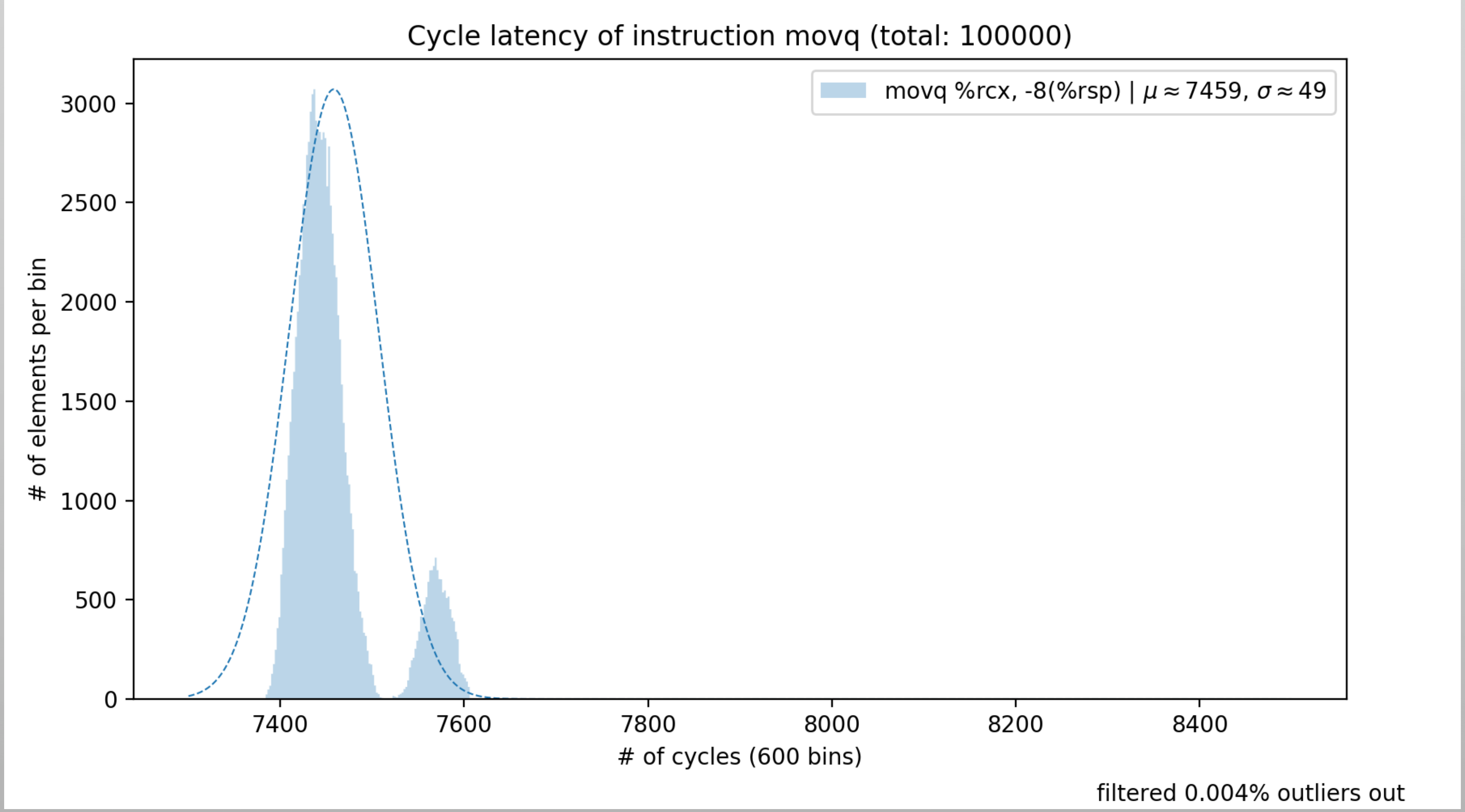


$$\frac{rdx \,\|\, rax}{const}$$

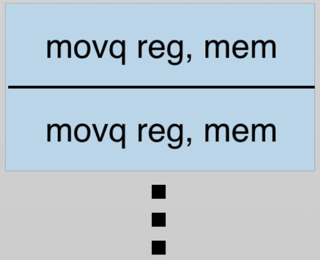
$$\frac{rdx \,\|\, rax}{const}$$

$\vdots$

Applications – Double Peaks



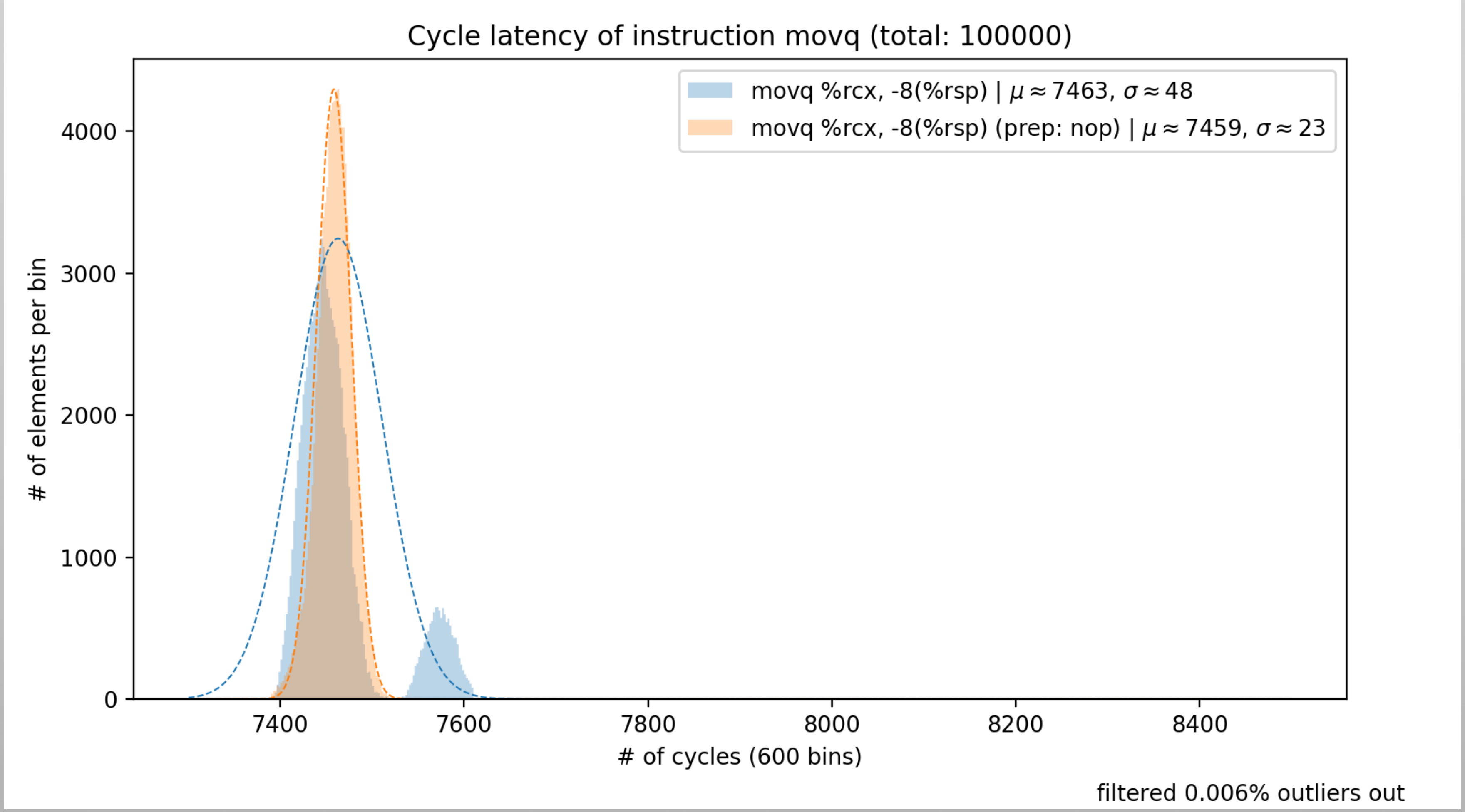
Cycle latency of instruction movq (total: 100000)

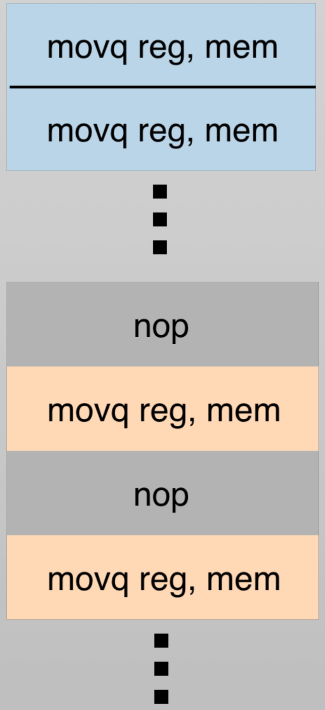


movq reg, mem

movq reg, mem

Applications – Double Peaks





movq reg, mem

movq reg, mem

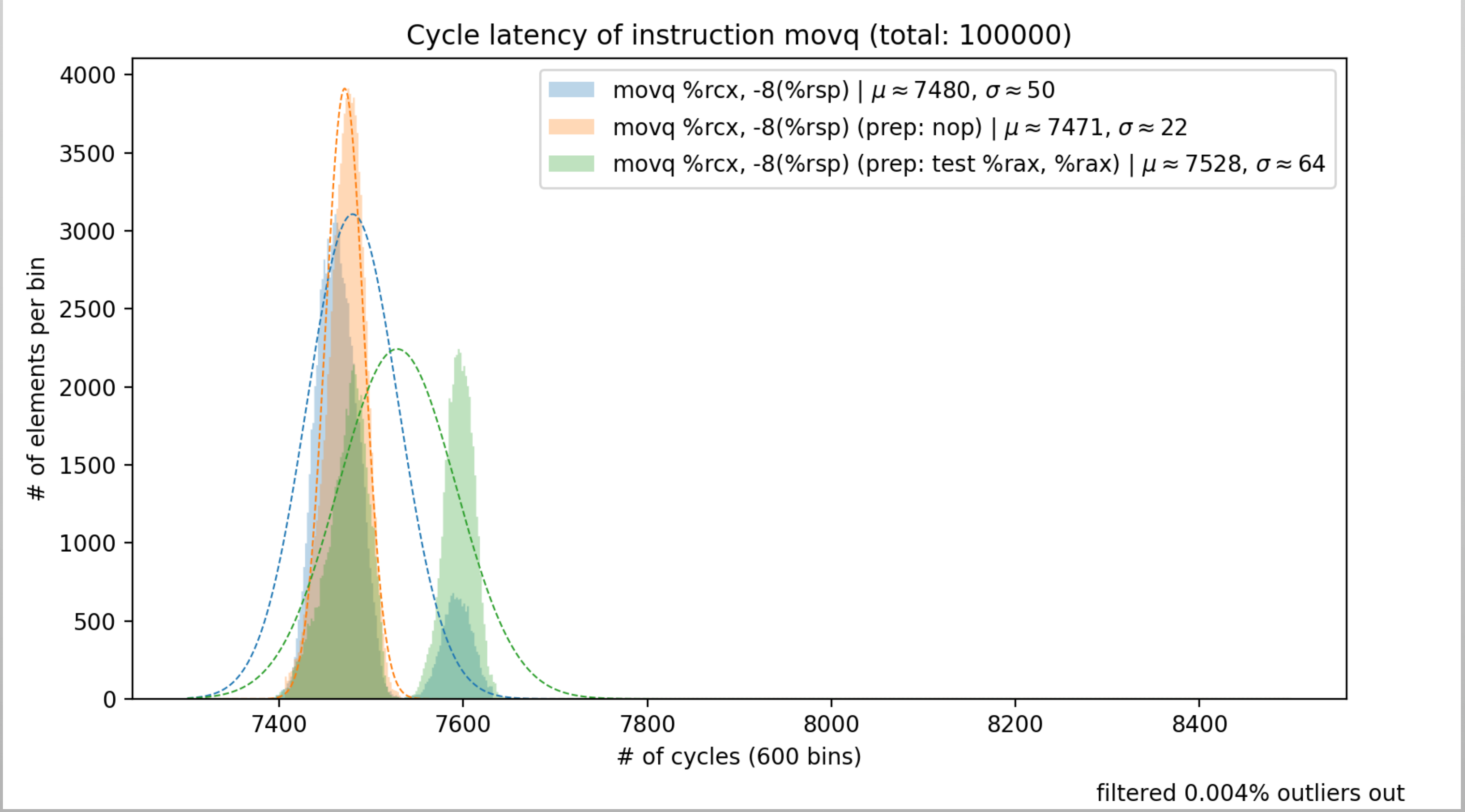
nop

movq reg, mem

nop

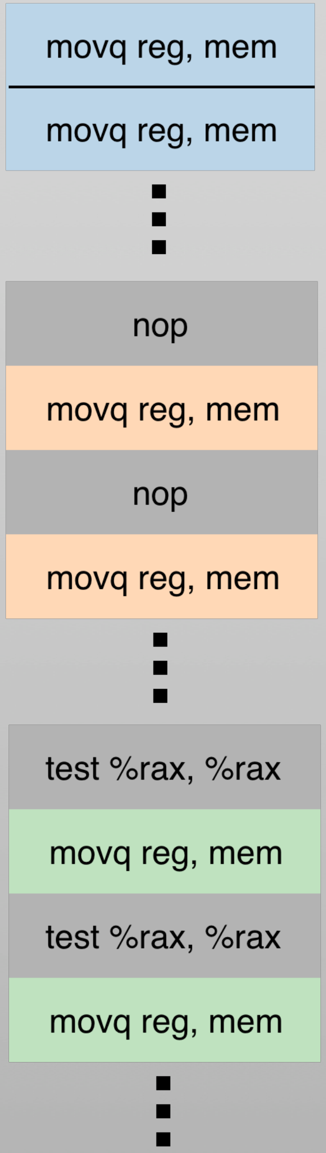
movq reg, mem

Applications – Double Peaks



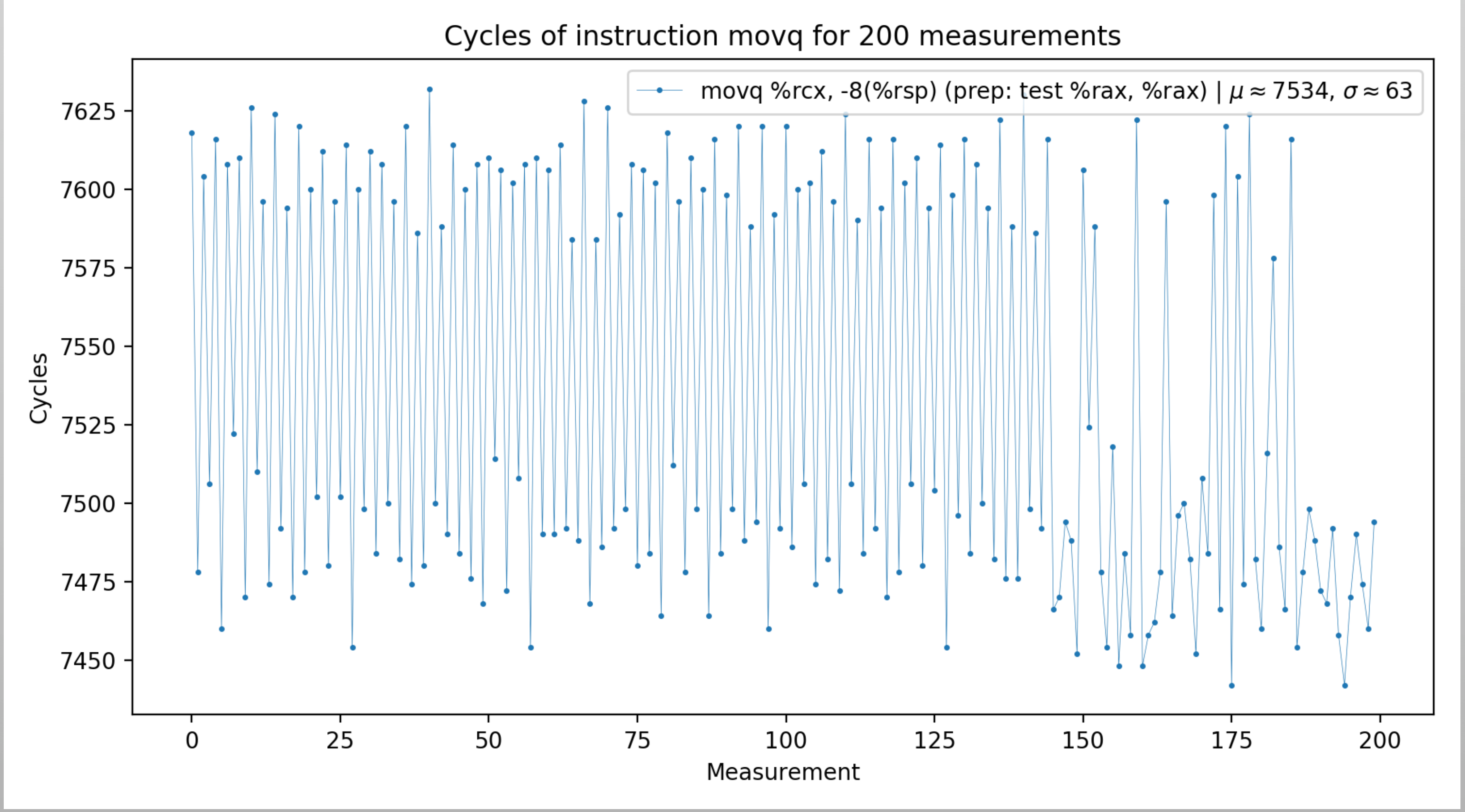
Cycle latency of instruction movq (total: 100000)

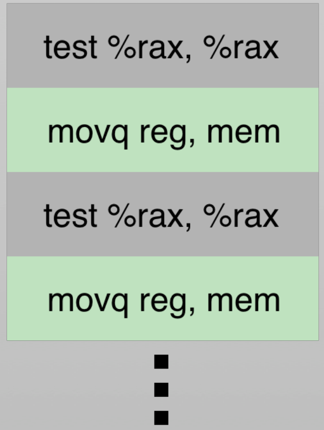
filtered 0.004% outliers out



ETH zürich

Applications – Double Peaks





test %rax, %rax

movq reg, mem

test %rax, %rax

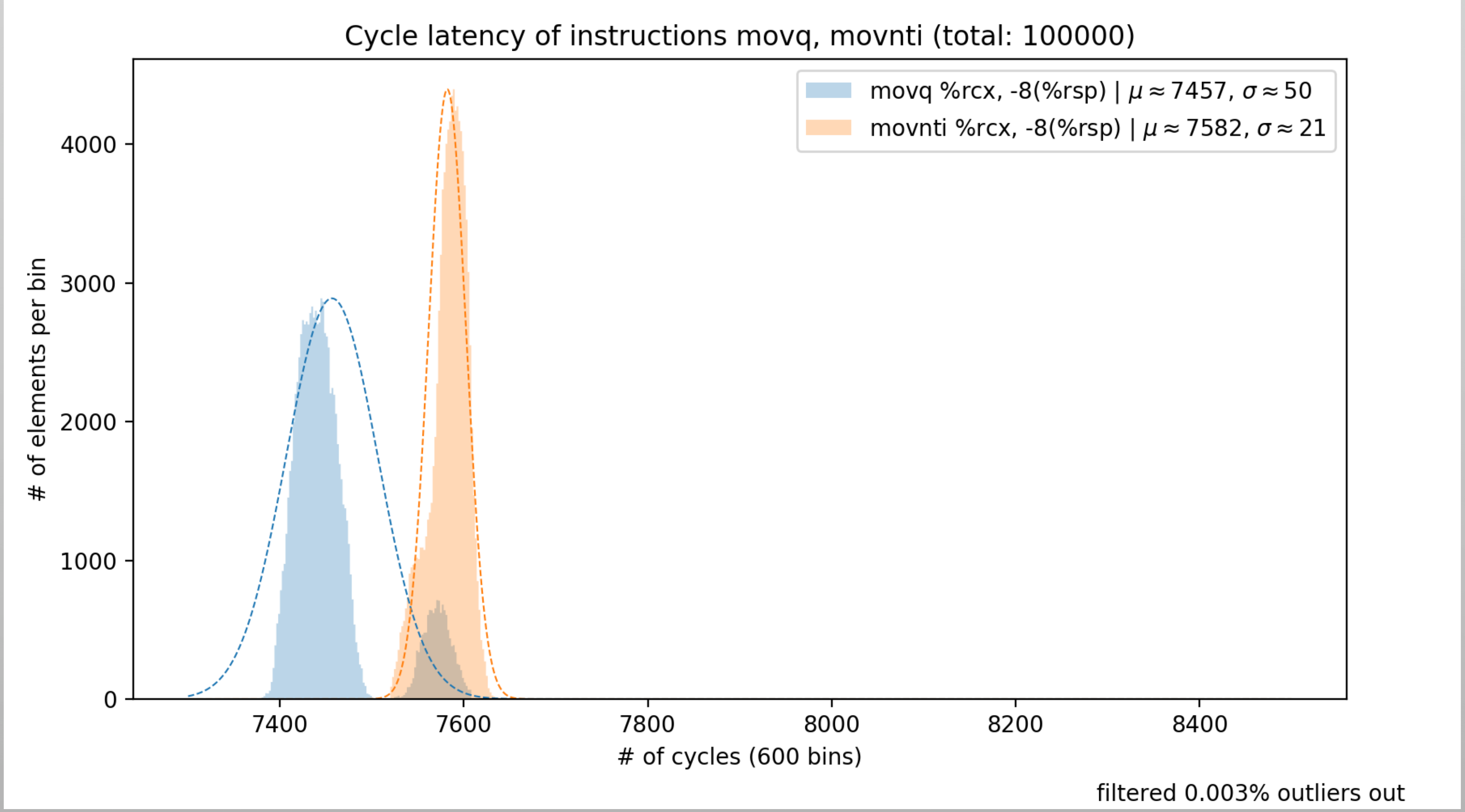
movq reg, mem

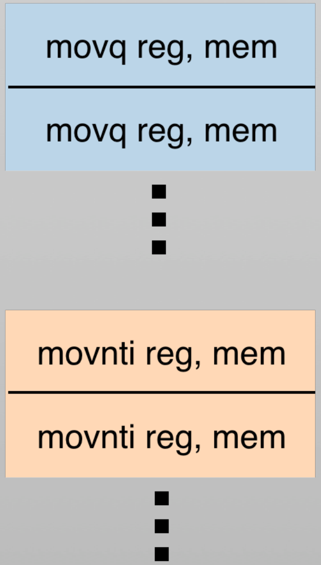
⋮

Applications – Double Peaks

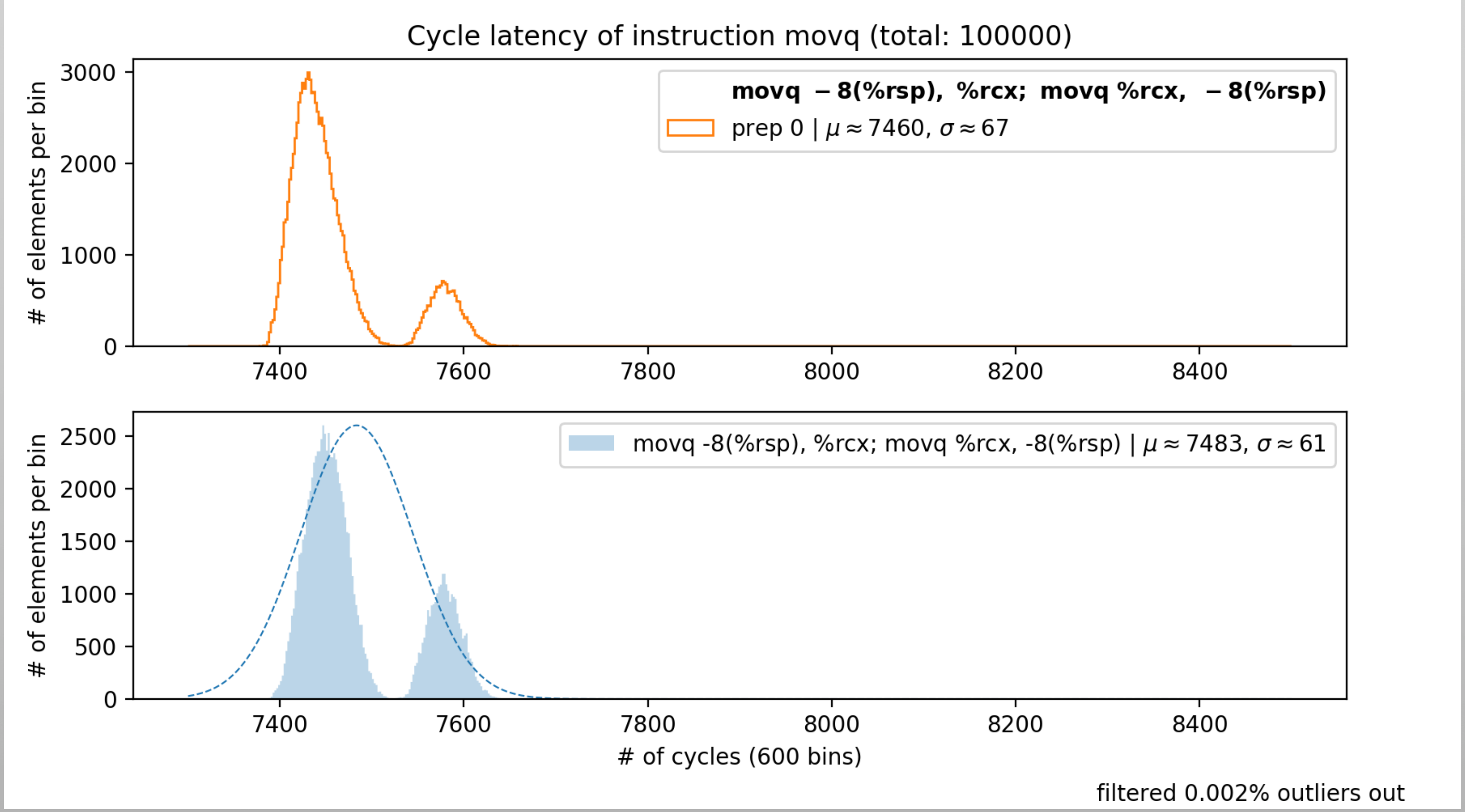
- Examples
- Possible explanation
  - Bypass the cache
  - Instruction termination
  - Microarchitectural state
- Supporting plots

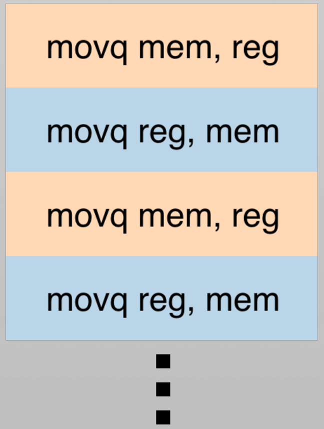
Applications – Double Peaks





Applications – Double Peaks





movq mem, reg

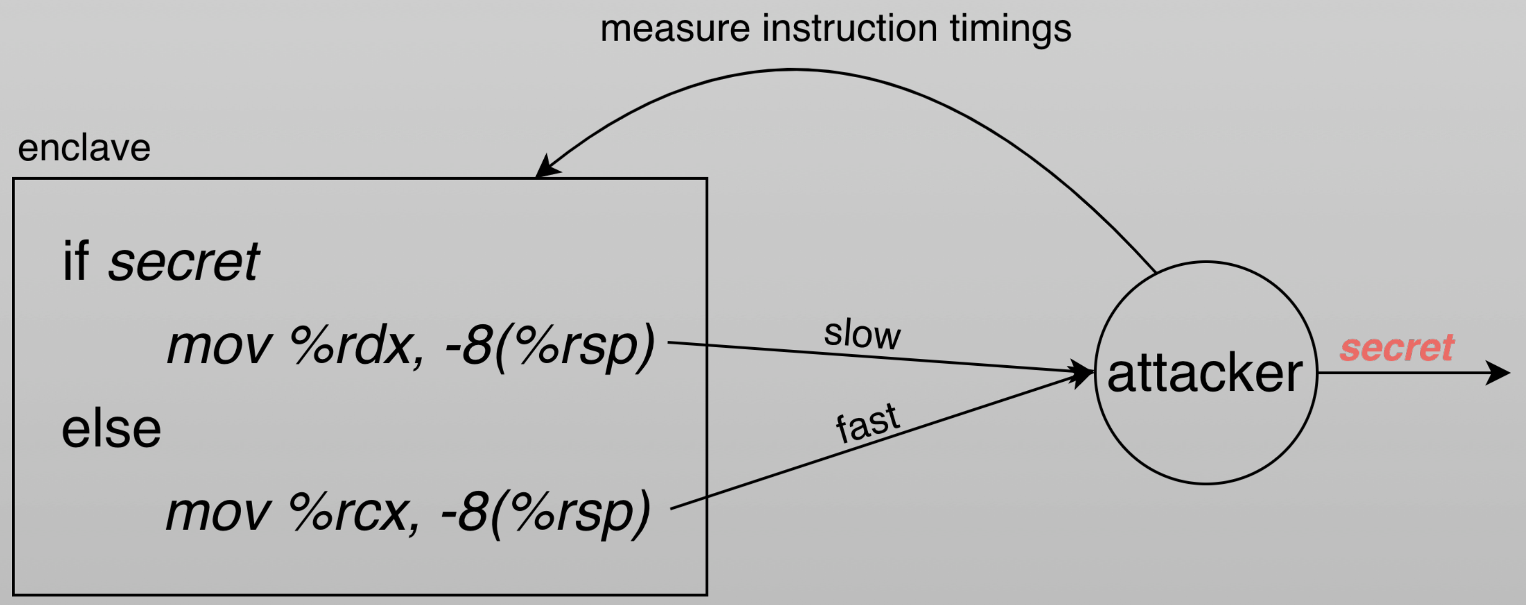
movq reg, mem

movq mem, reg

movq reg, mem

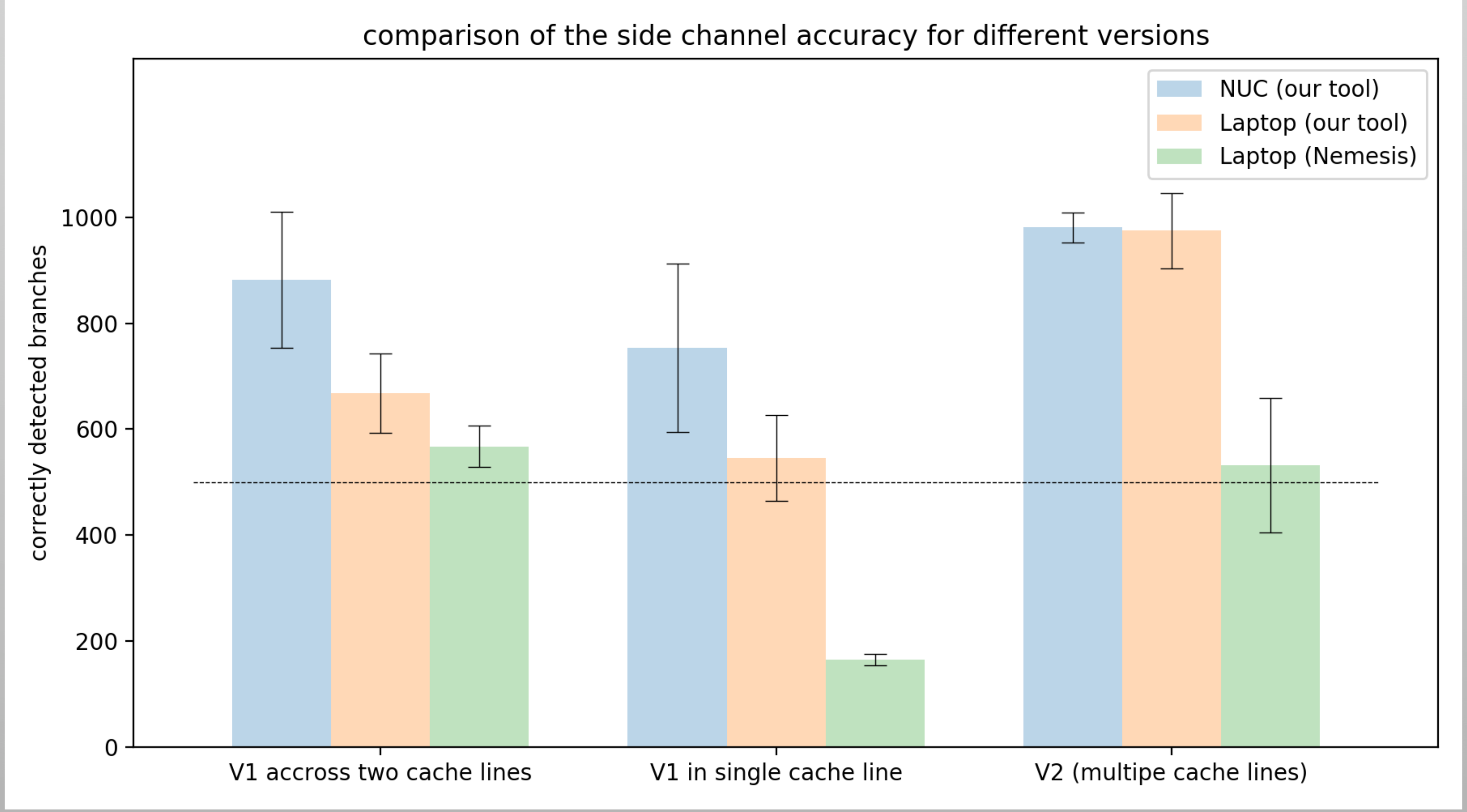
⋮

Applications – Side Channel Attack



```
if secret
        mov %rdx, -8(%rsp)
else
        mov %rcx, -8(%rsp)
```

Applications – Side Channel Attack



Conclusion

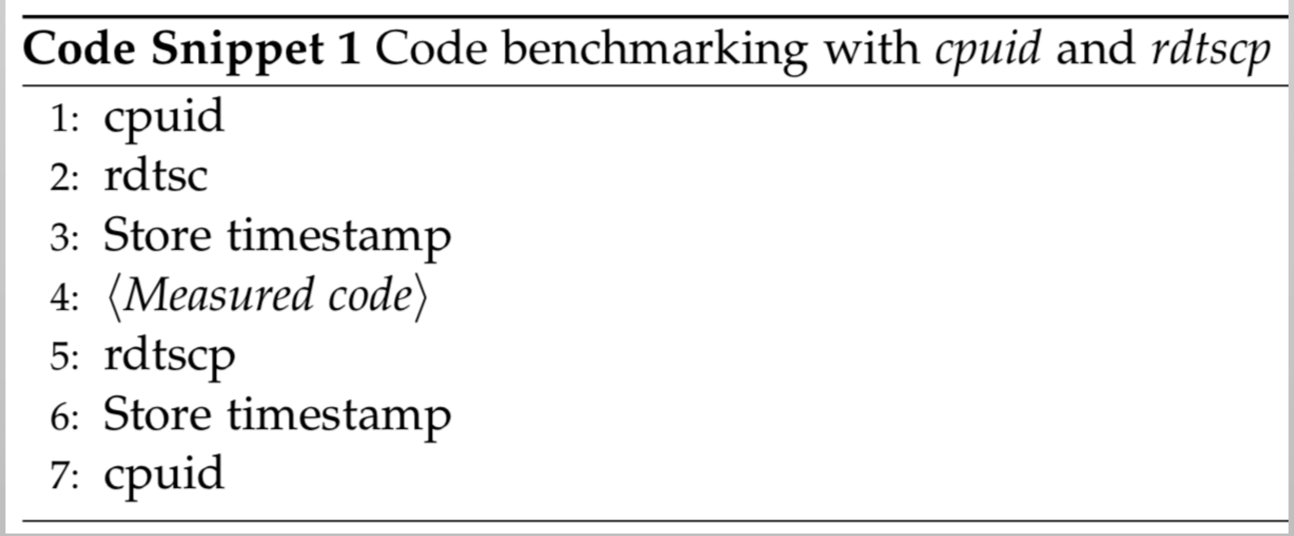
- Measuring is not trivial
- Increased precision
- Tool for further research
  - Double peaks
  - Difference between enclaves
  - Multi-steps
- Questions?

Sources

- Microsoft Azure logo: https://commons.wikimedia.org/wiki/File:Microsoft_Azure_Logo.svg
- 1Password logo: https://1password.com/de/press/
- IBM Cloud logo: https://www.cncf.io/logoshowcase/ibm/attachment/ibm-cloud/

Backup slides

CPUID and RDTSCP

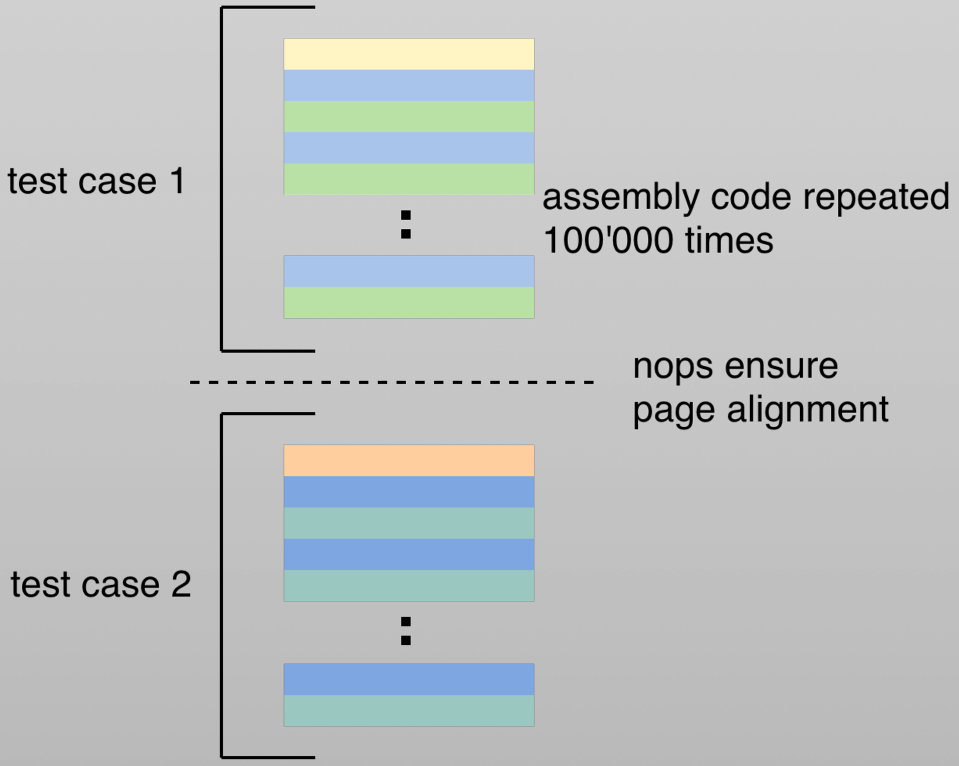


Code Snippet 1 Code benchmarking with *cpuid* and *rdtscp*

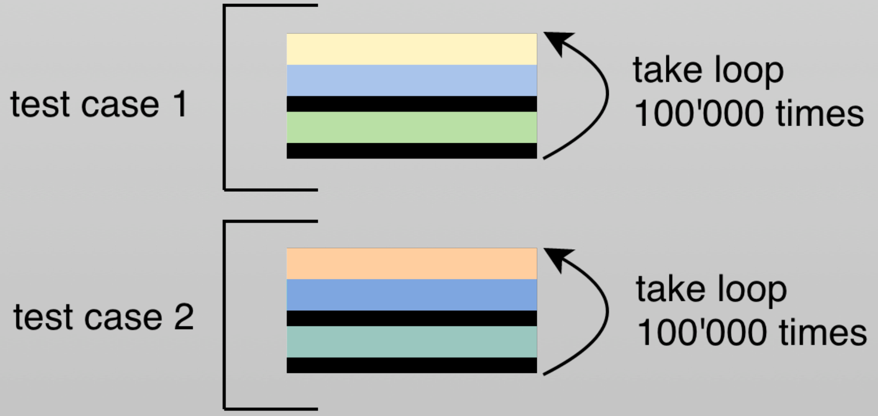
```
cpuid
rdtsc
Store timestamp
⟨Measured code⟩
rdtscp
Store timestamp
cpuid
```

Method Comparison I

SGX-Step Method



Counter Method & Outside Enclave



Legend:

■	start/stop timer	
test case 1:		test case 2:
🟨	initial instructions	🟧
🟦	prepare instructions	🟦
🟩	test instruction	🟩

Method Comparison II

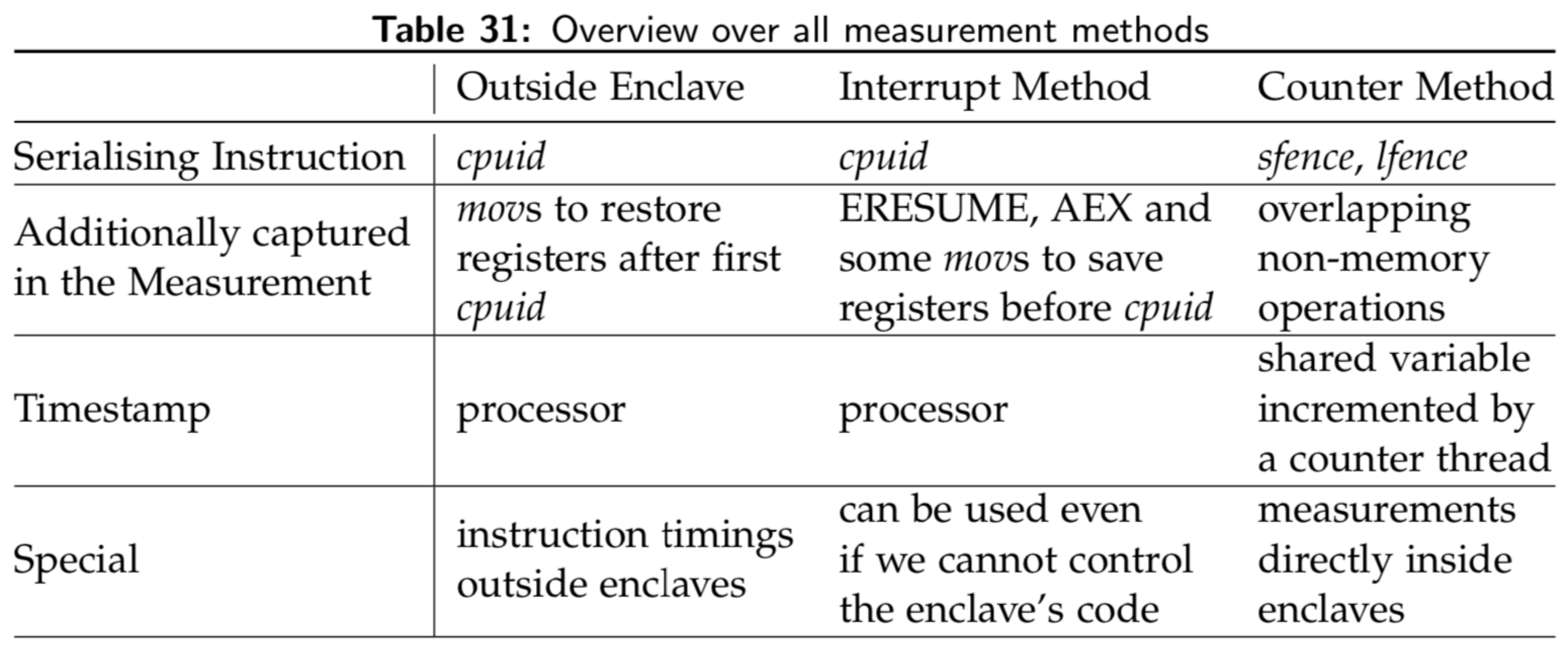
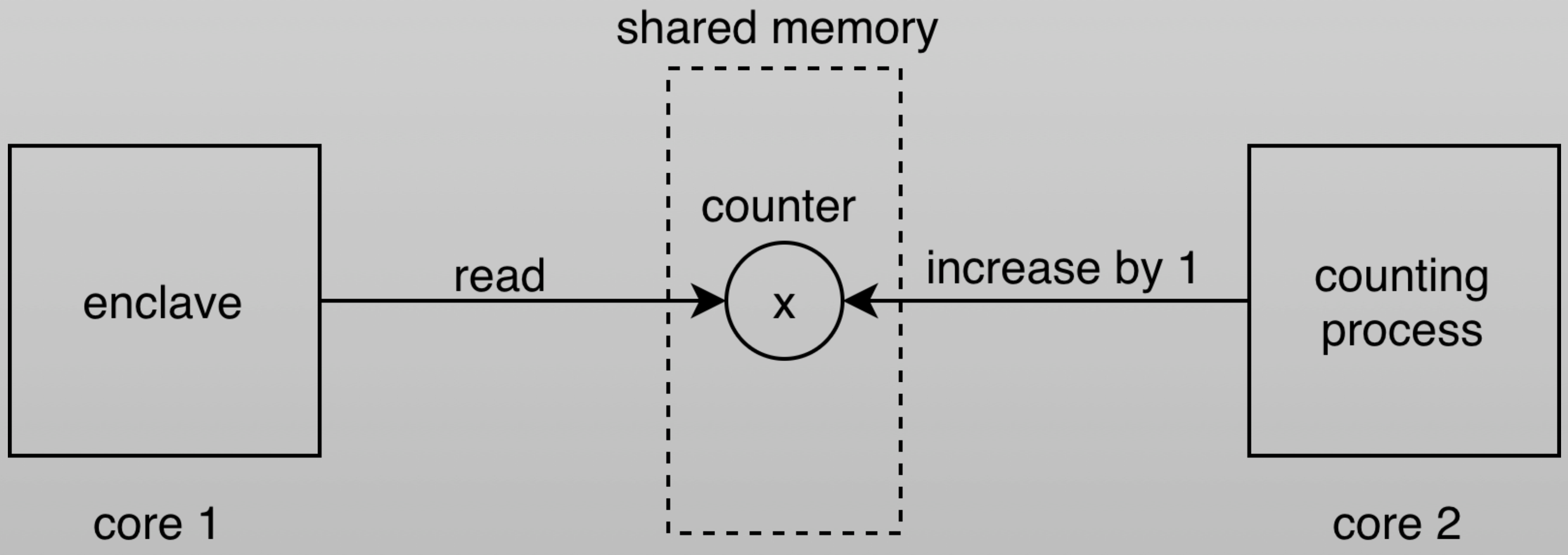


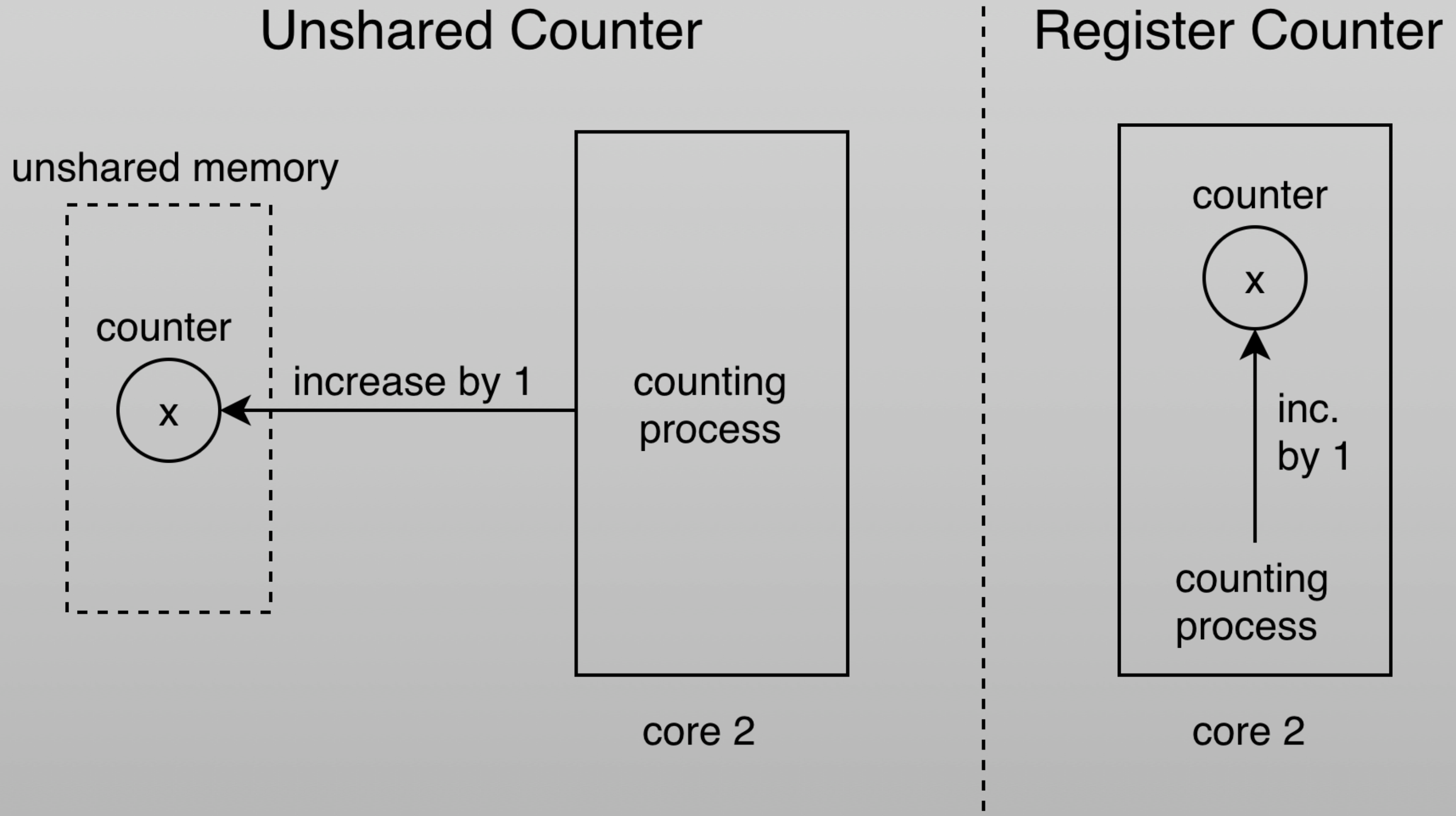
Table 31: Overview over all measurement methods

	Outside Enclave	Interrupt Method	Counter Method
Serialising Instruction	*cpuid*	*cpuid*	*sfence*, *lfence*
Additionally captured in the Measurement	*mov*s to restore registers after first *cpuid*	ERESUME, AEX and some *mov*s to save registers before *cpuid*	overlapping non-memory operations
Timestamp	processor	processor	shared variable incremented by a counter thread
Special	instruction timings outside enclaves	can be used even if we cannot control the enclave's code	measurements directly inside enclaves

Counter Method



Counter Method Limitation



Challenges – Brief Summary

- Incomparable enclaves
- Measuring across page borders
  - Support multiple pages of code
  - Deal with outliers at page borders
- Cache conflicts
  - Minimize cache pollution between AEX and ERESUME

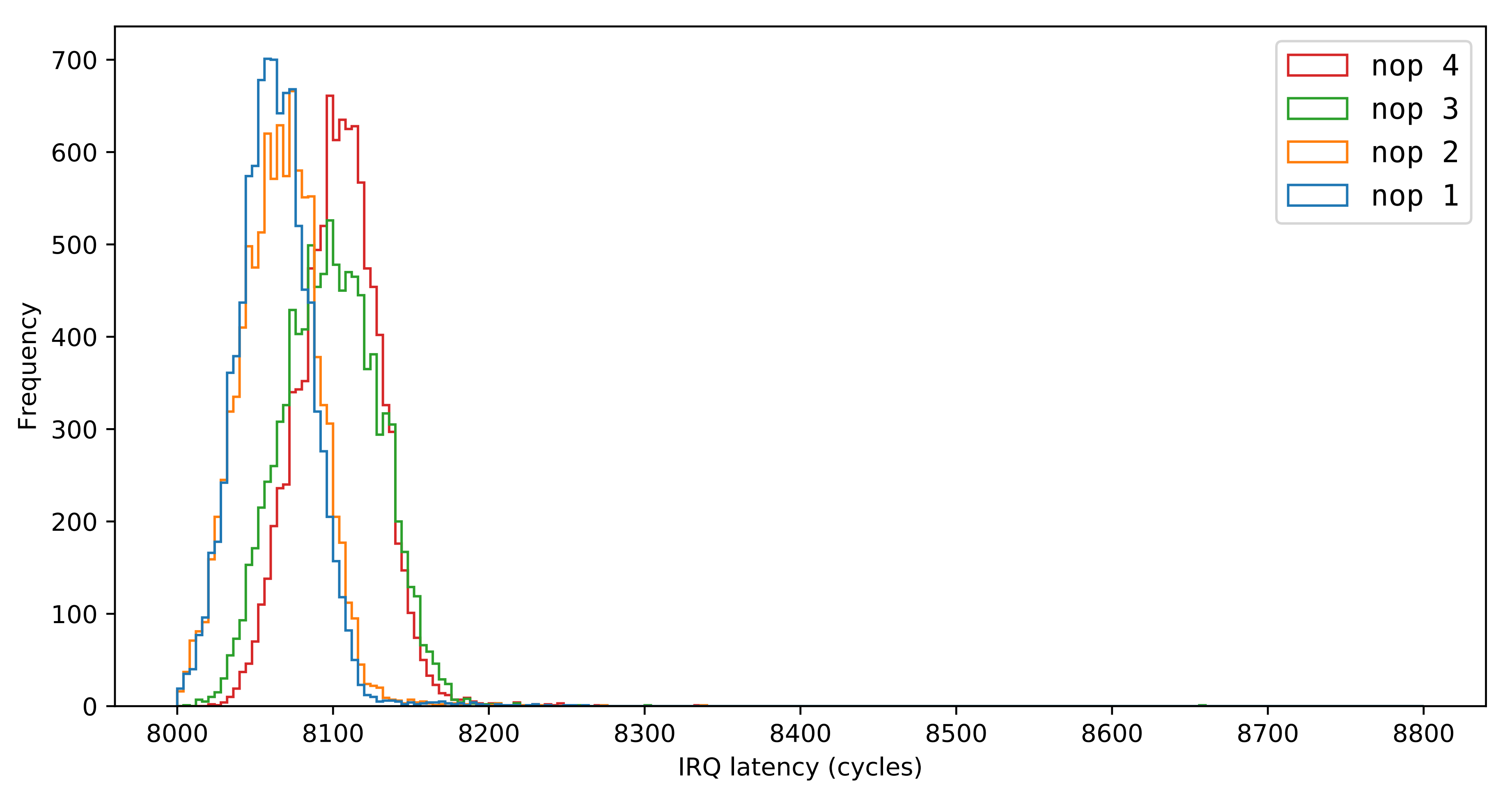
Challenges – Brief Summary

- Constant time code
  - Code between two measurements should be the same independent of the measured instruction
- Precise tracking of instructions
  - Time shifts between enclaves can desynchronise
  - Some assembly instructions perform two operations and can be interrupted in between
- Verifying tests
  - Prepare instructions influence test instructions
  - Make sure that they do not trigger exceptional behaviour

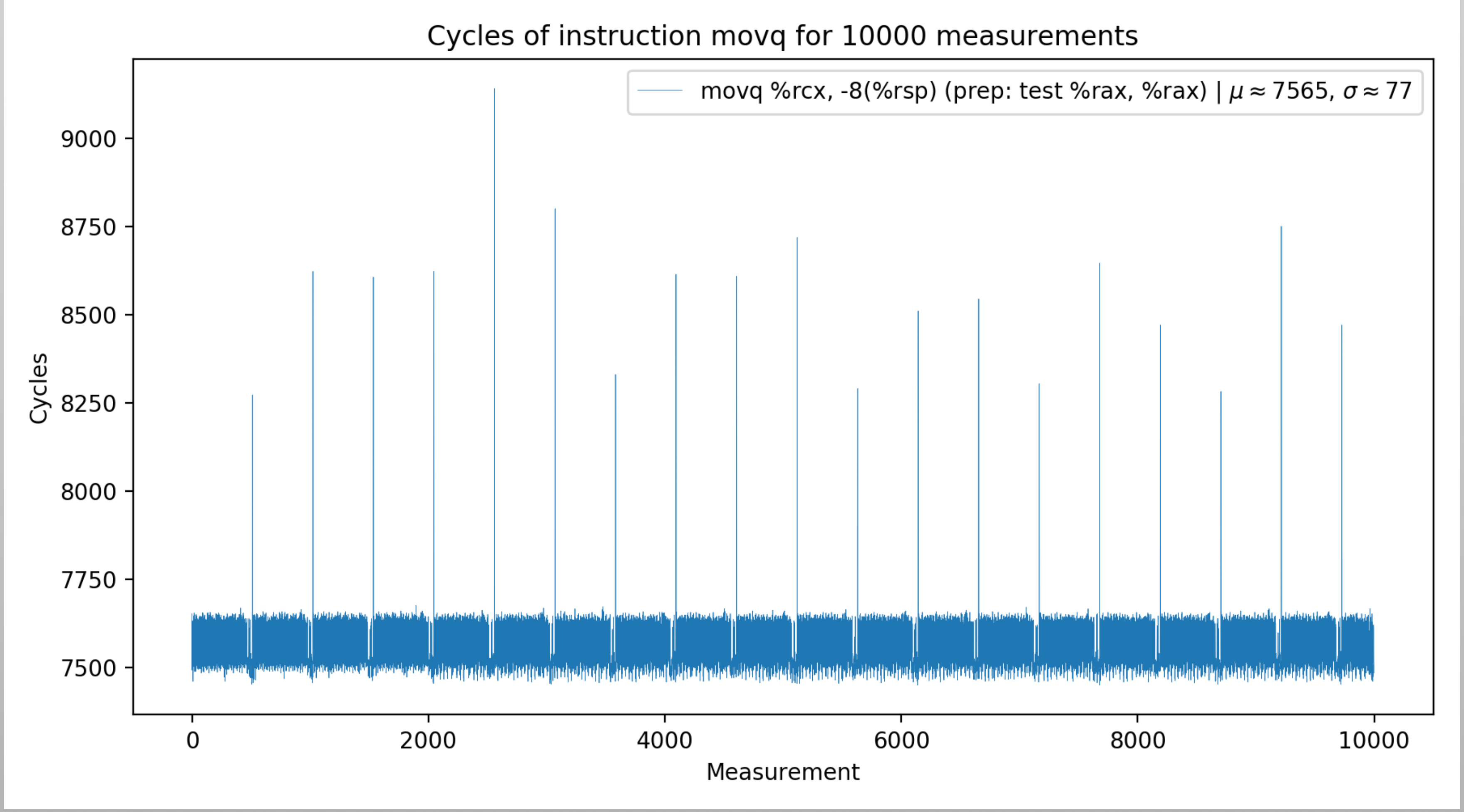
Challenges – Brief Summary

- Build advanced test cases
  - Setting flags
  - Write to memory
- Two sources of noise
  - ERESUME/AEX
  - Instruction measurement
- Synthetic state on AEX
  - State that AEX creates on exit must be manually preserved

C1: Incomparable Enclaves

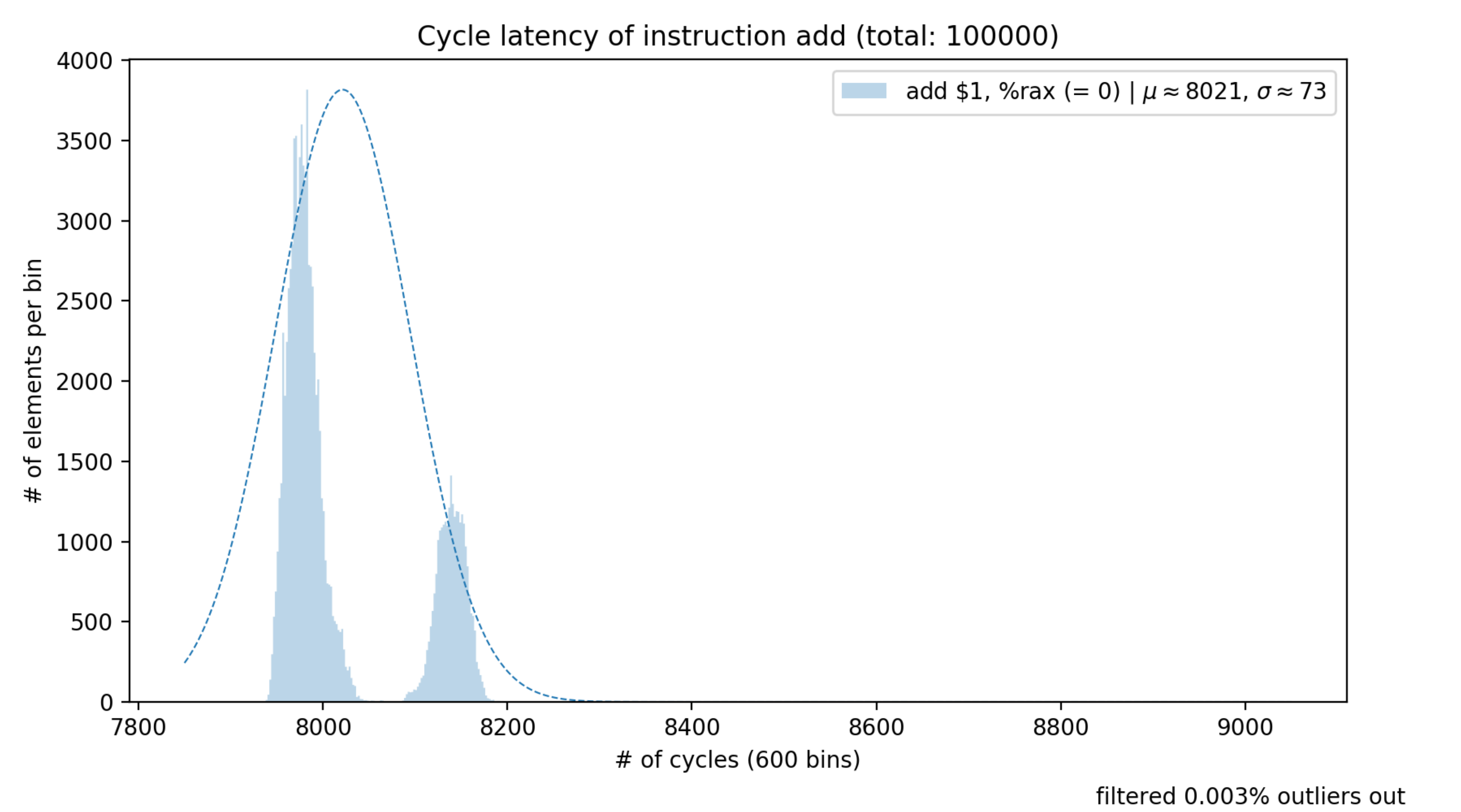


C2: Measuring Across Page Borders

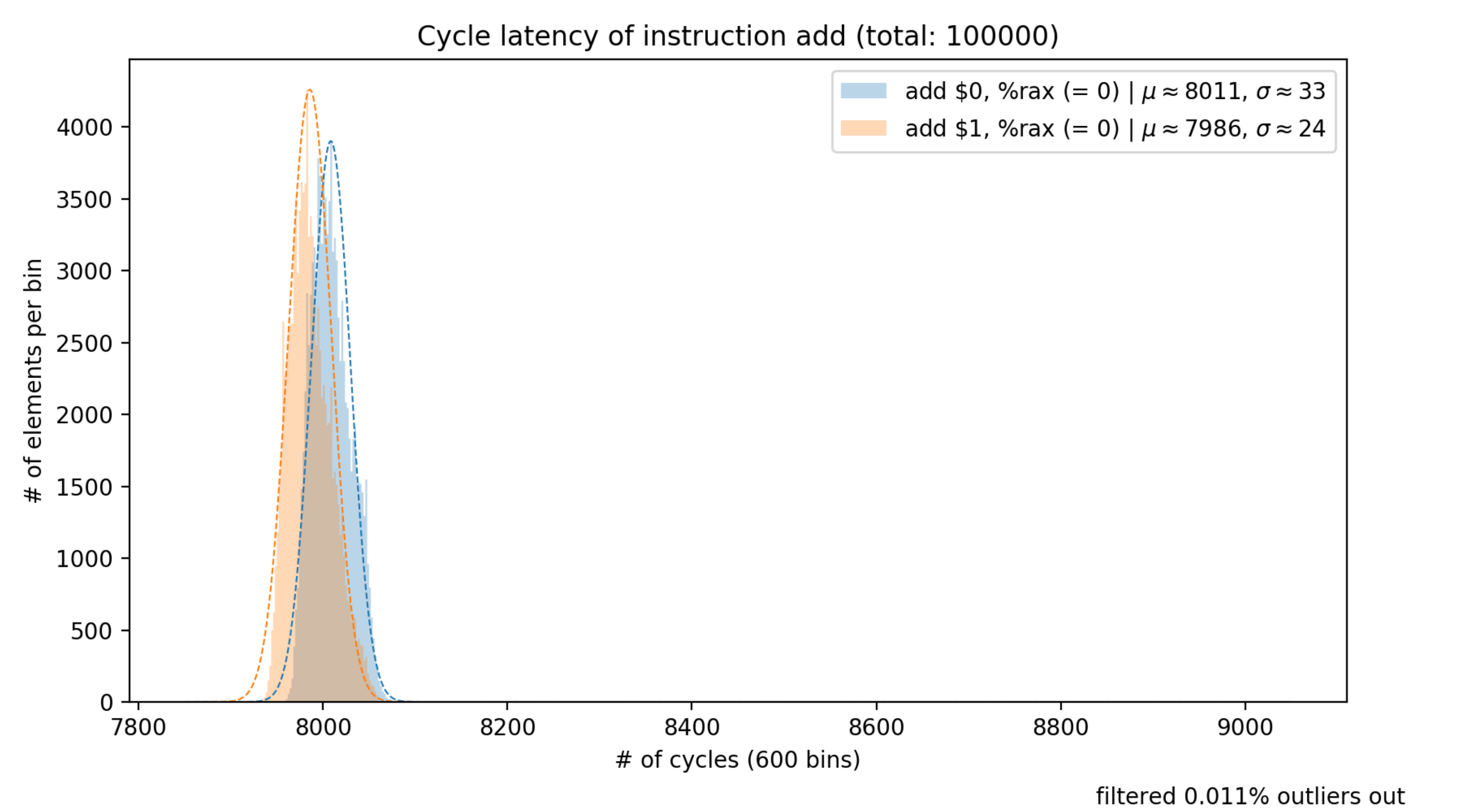


ETHzürich

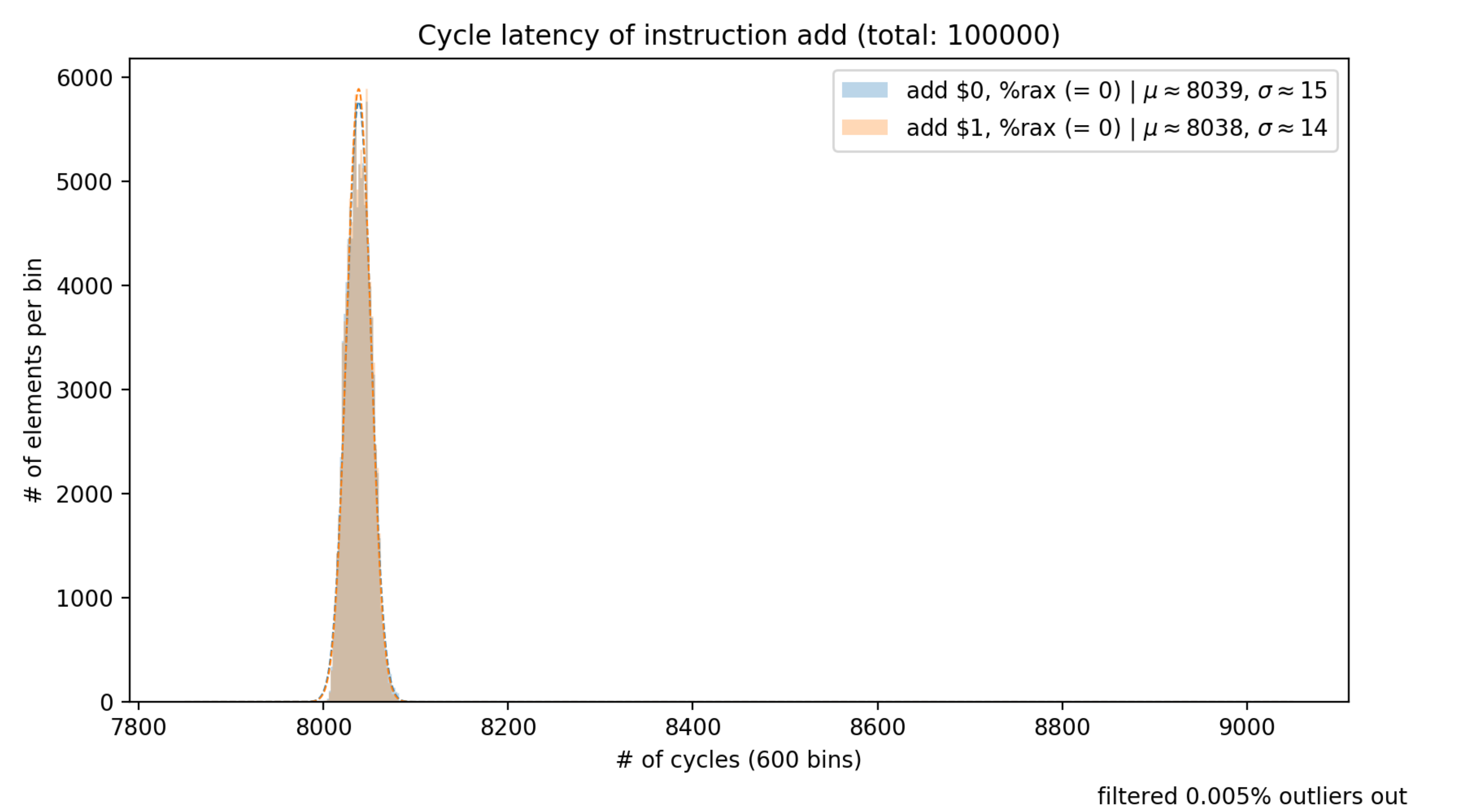
C3: Cache Conflicts



C4: Constant Time Code

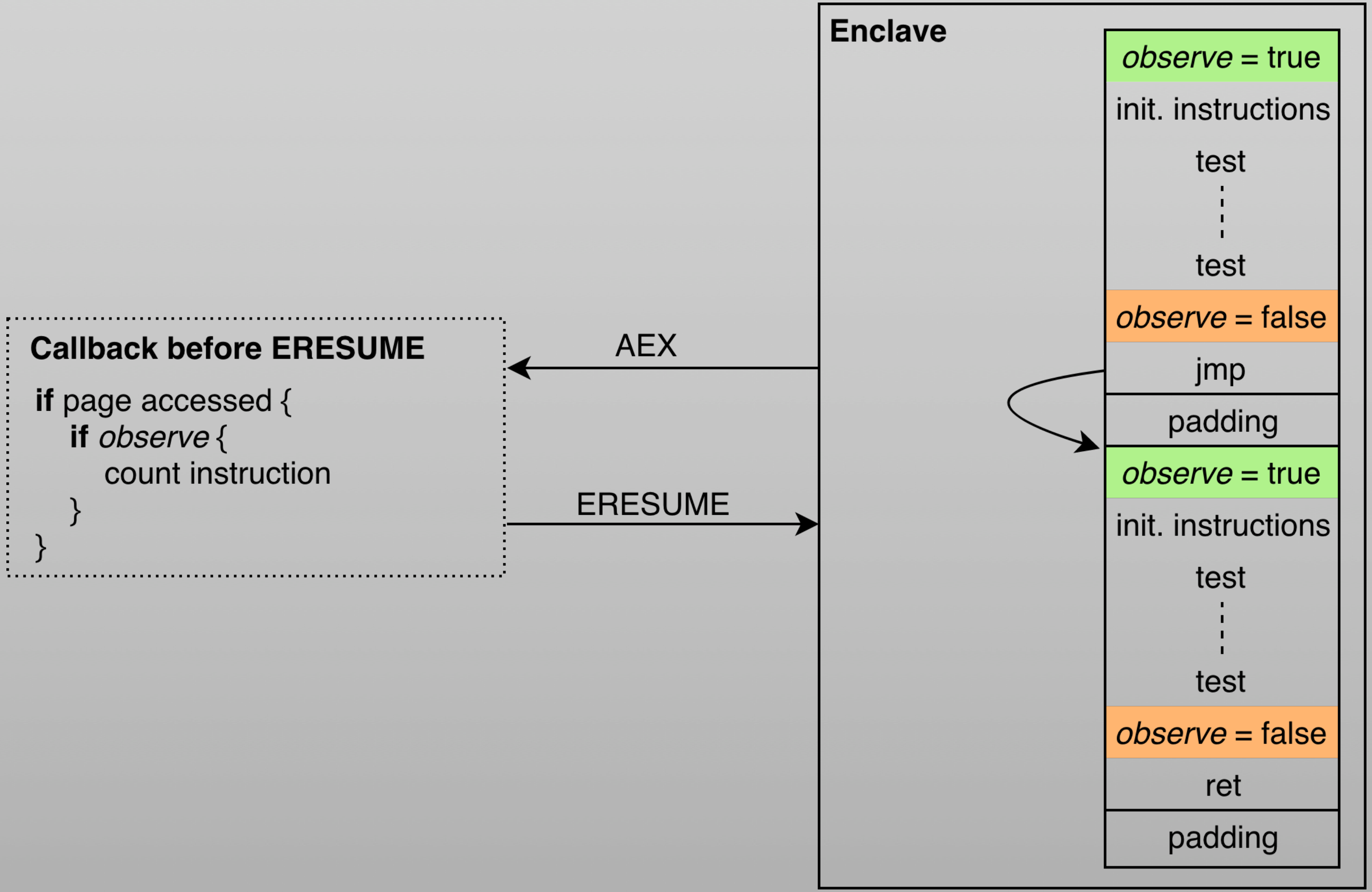


C4: Constant Time Code



ETHzürich

C6 & C7: Precise Instruction Tracking



Callback before ERESUME

```
if page accessed {
    if observe {
        count instruction
    }
}
```

AEX

ERESUME

Enclave

observe = true

init. instructions

test

test

observe = false

jmp

padding

observe = true

init. instructions

test

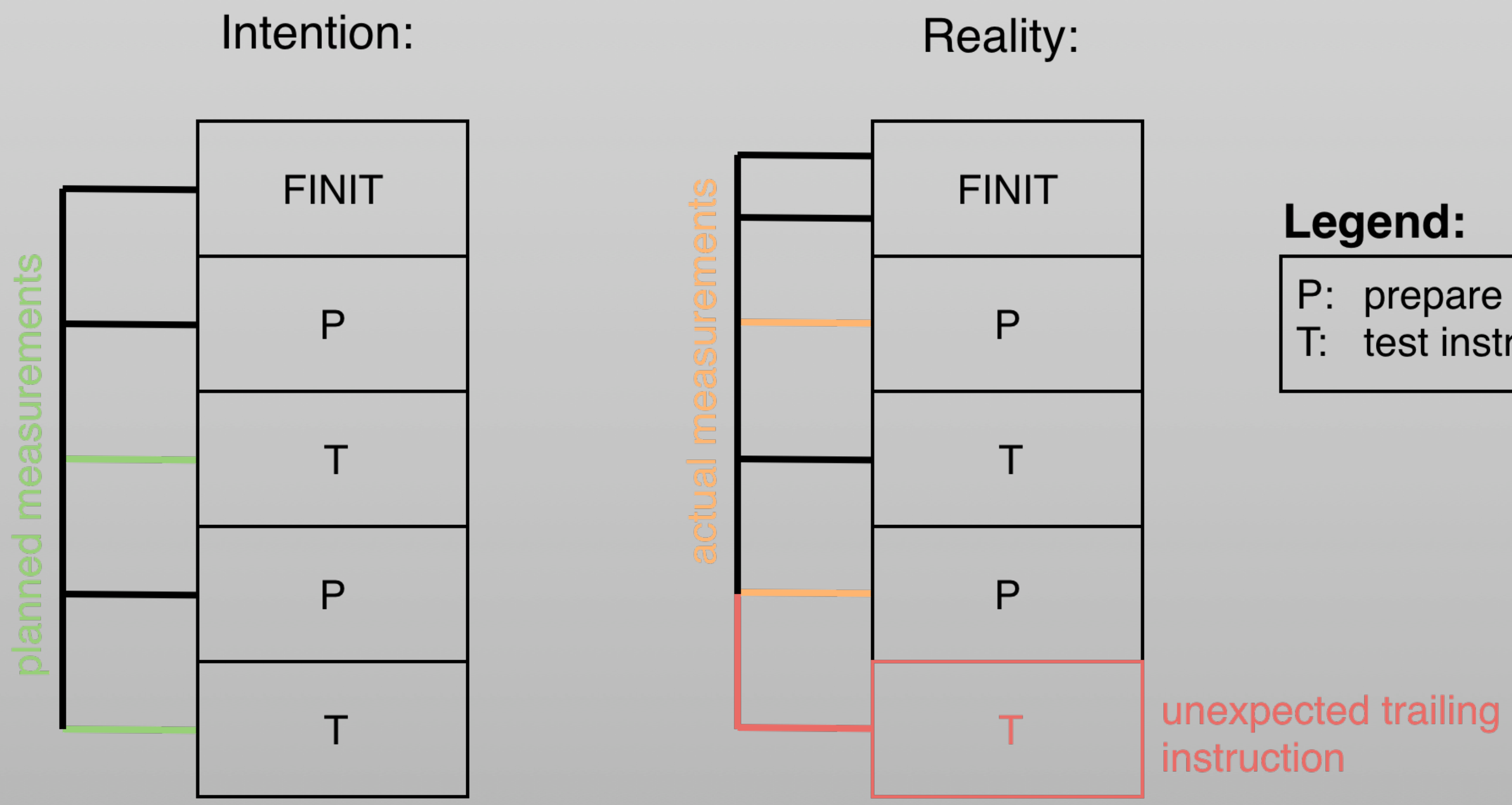
test

observe = false

ret

padding

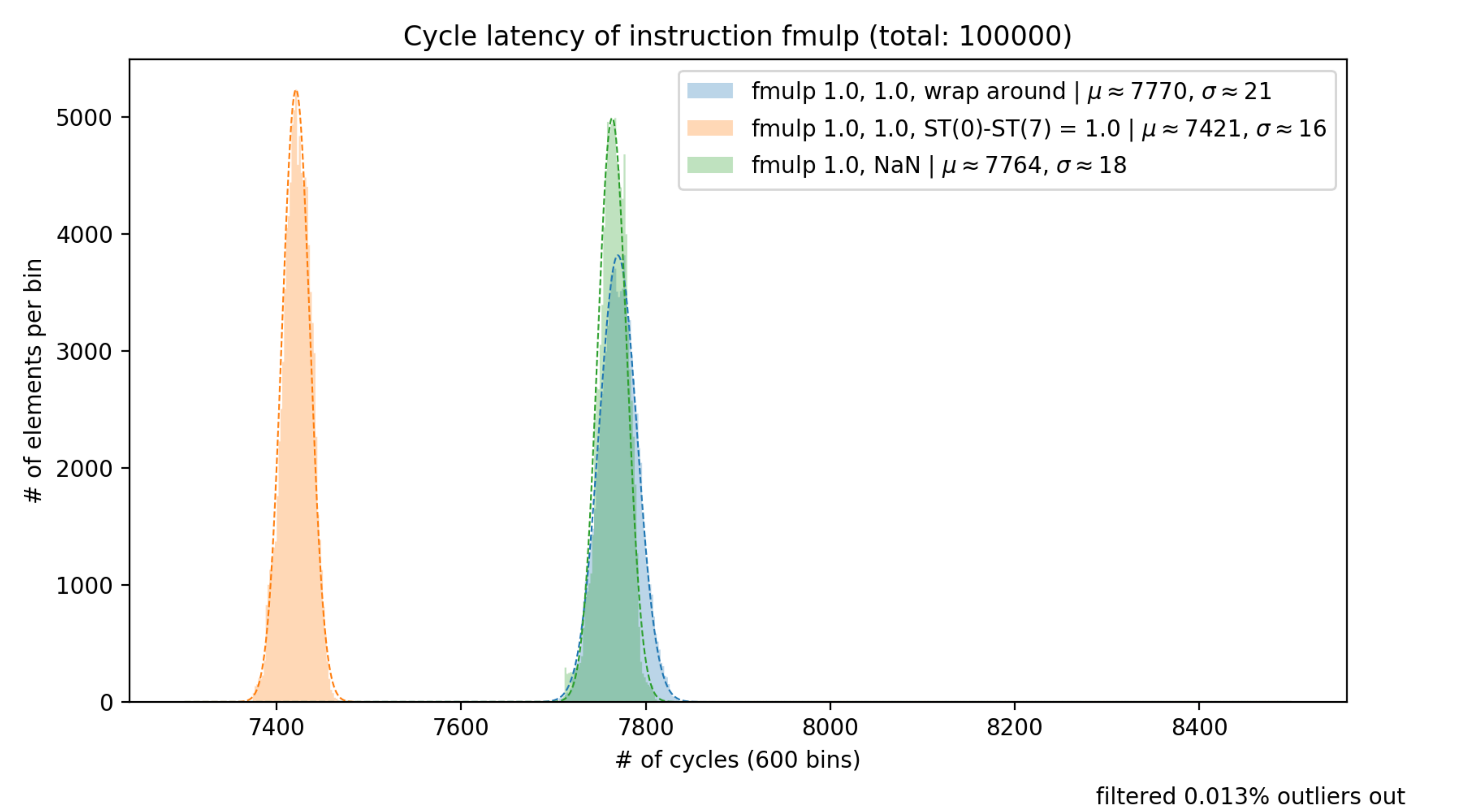
C6 & C7: Precise Instruction Tracking



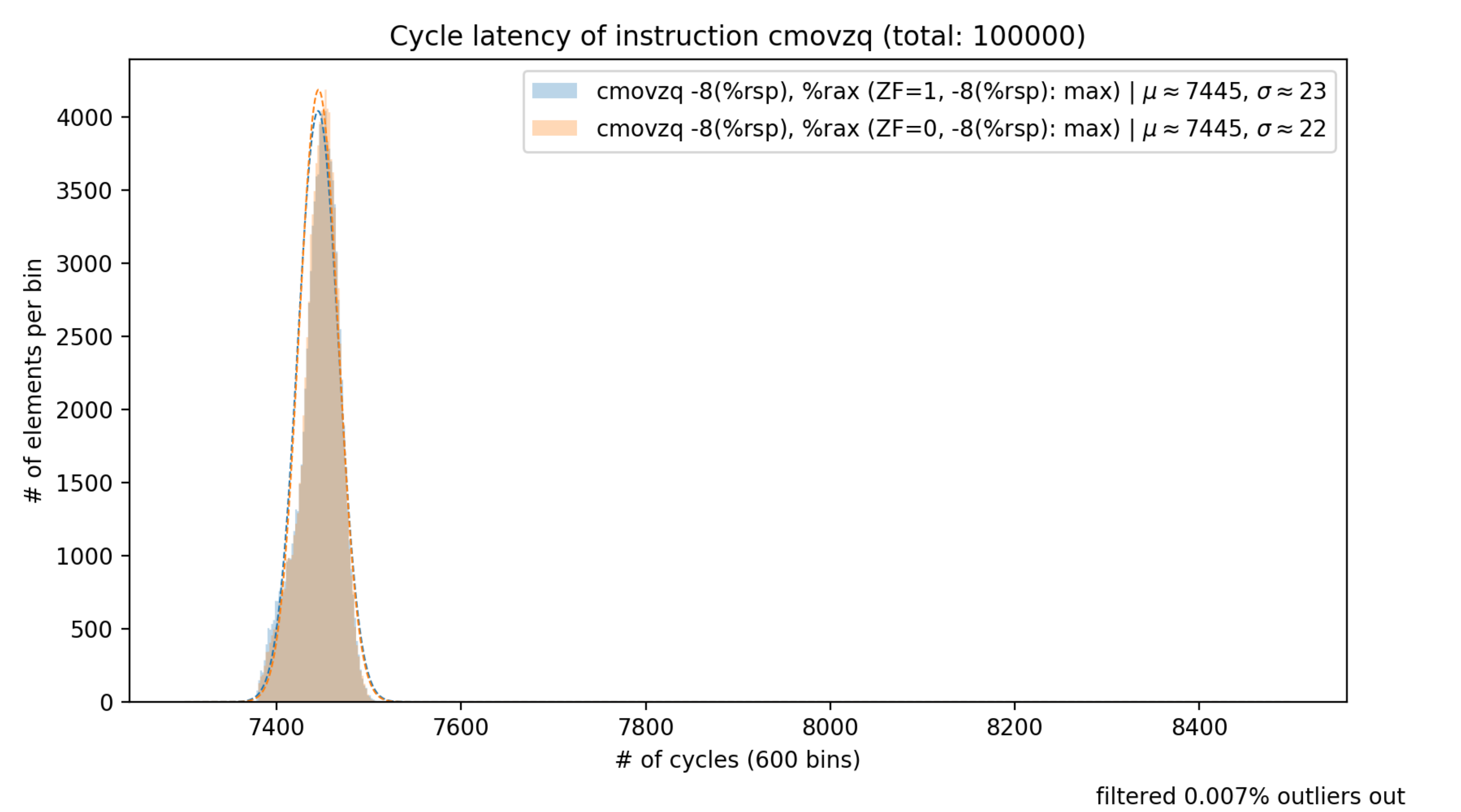
Legend:

P:	prepare instruction
T:	test instruction

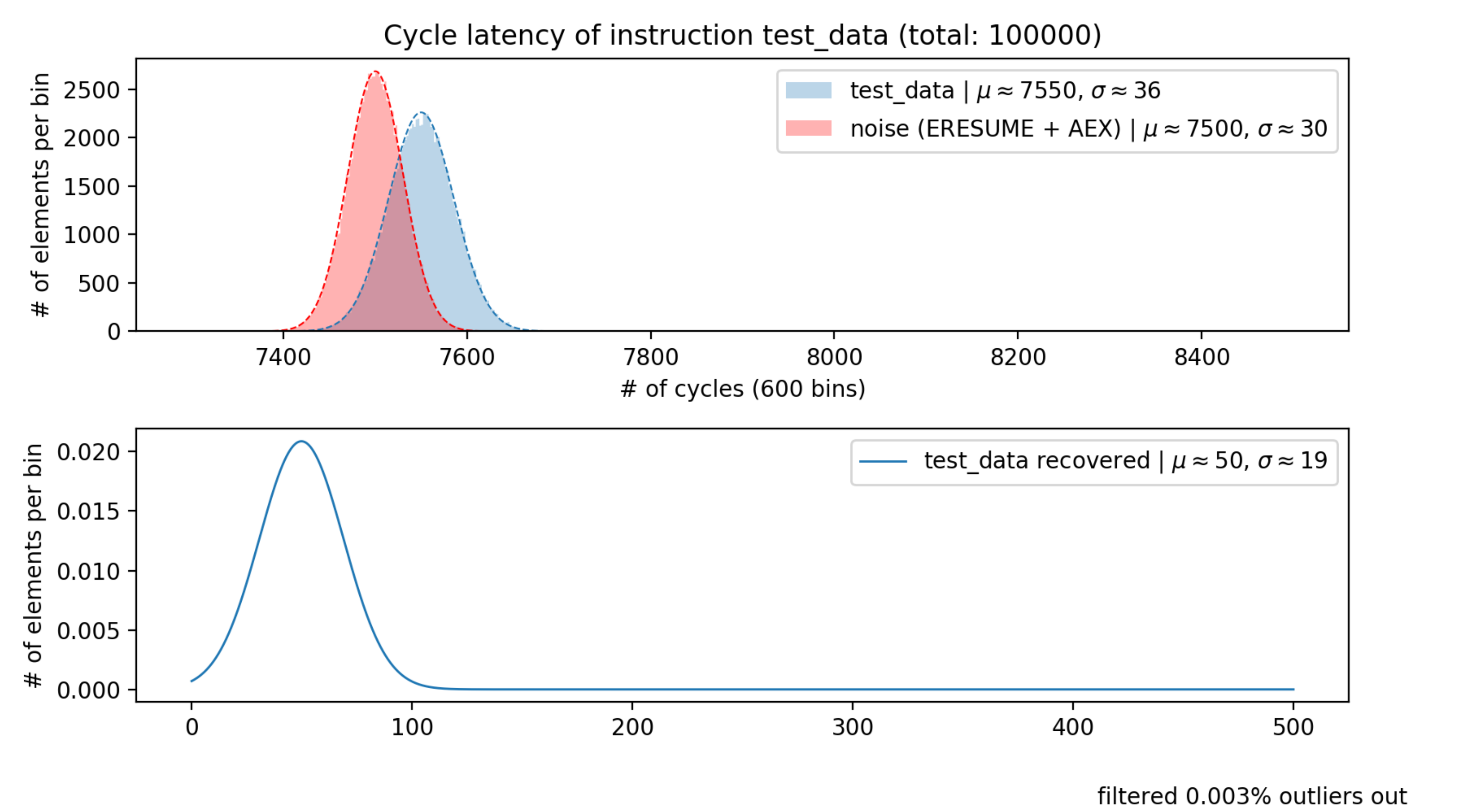
C8: Verifying Tests



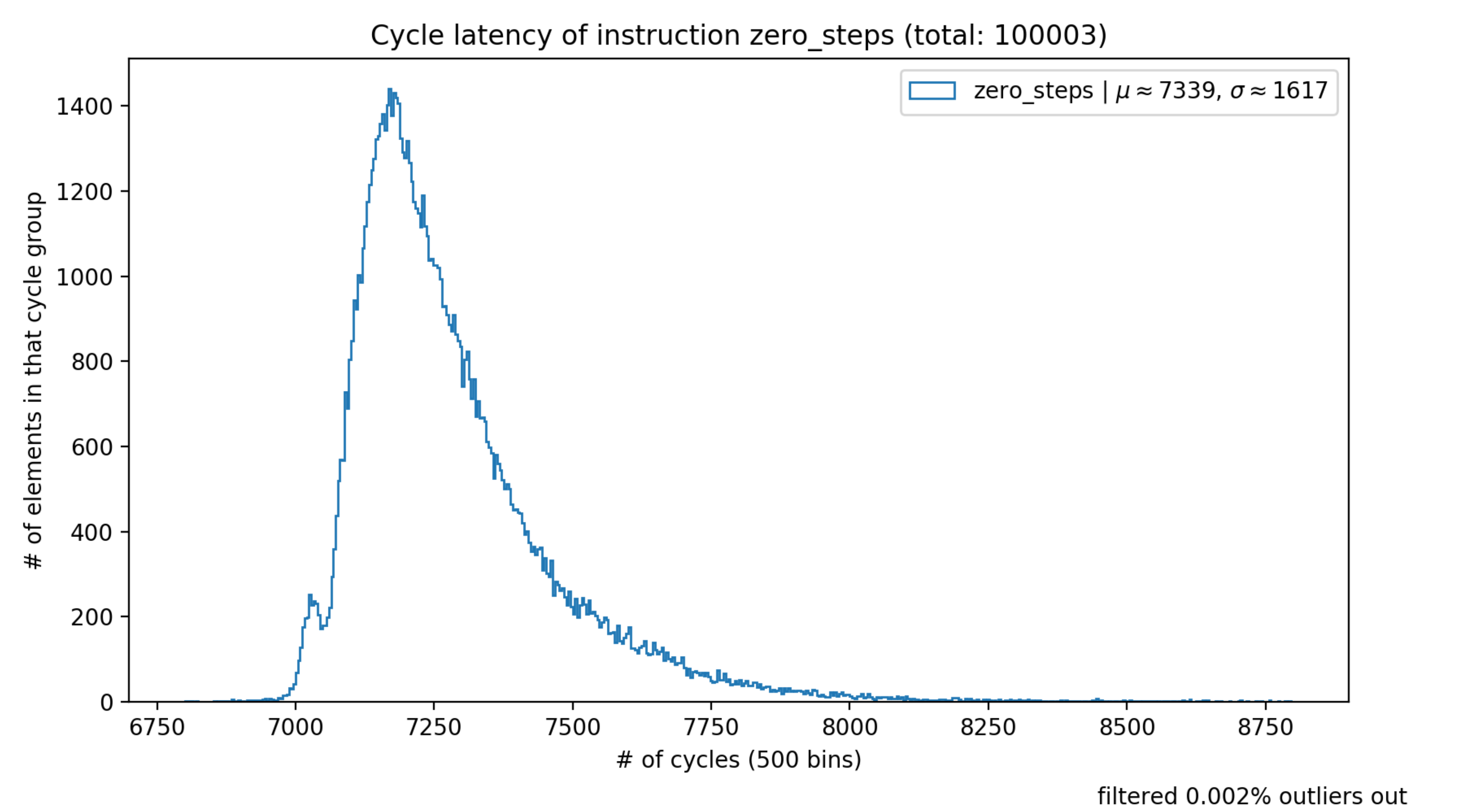
C9: Setting Flags



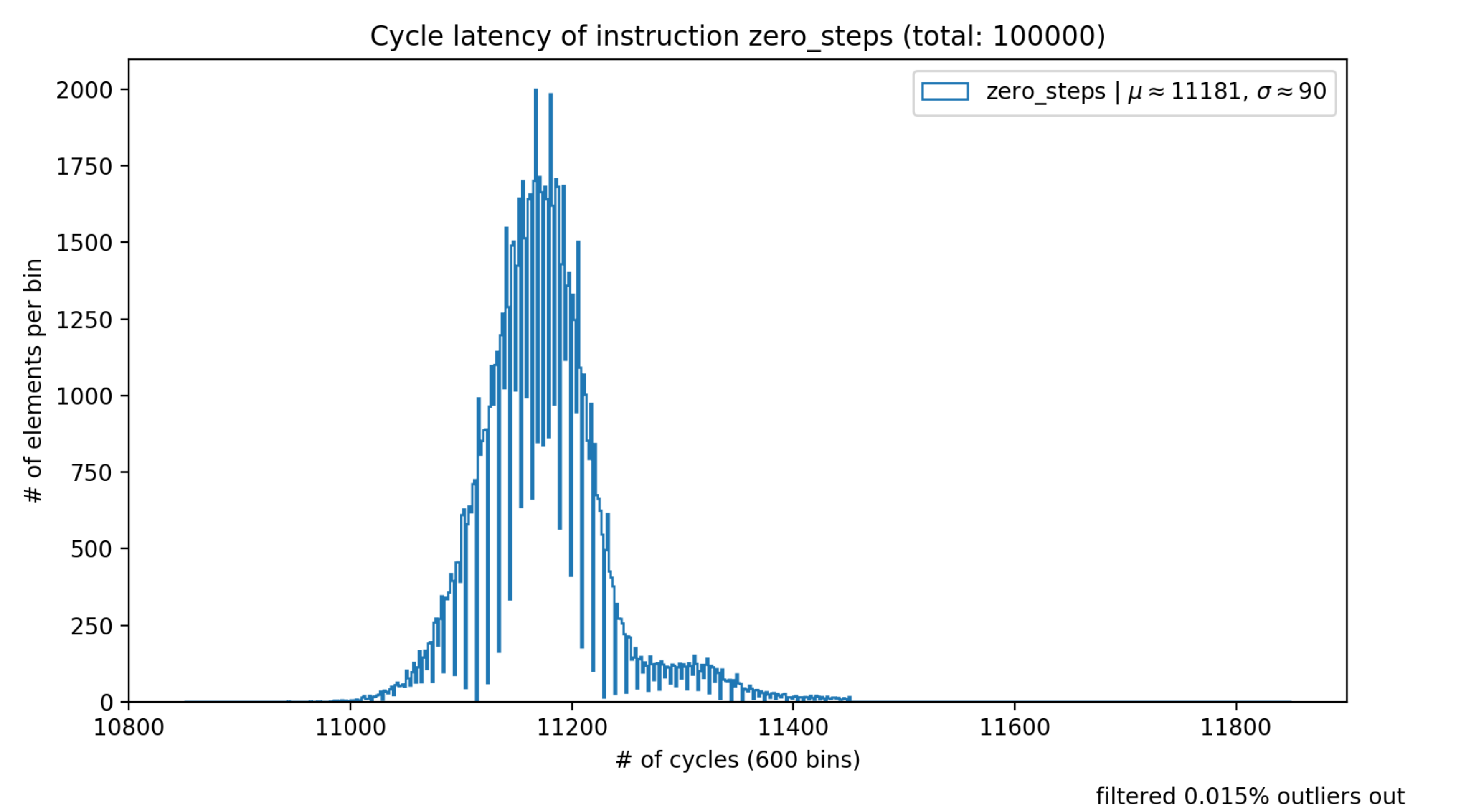
C10: Two Noise Sources



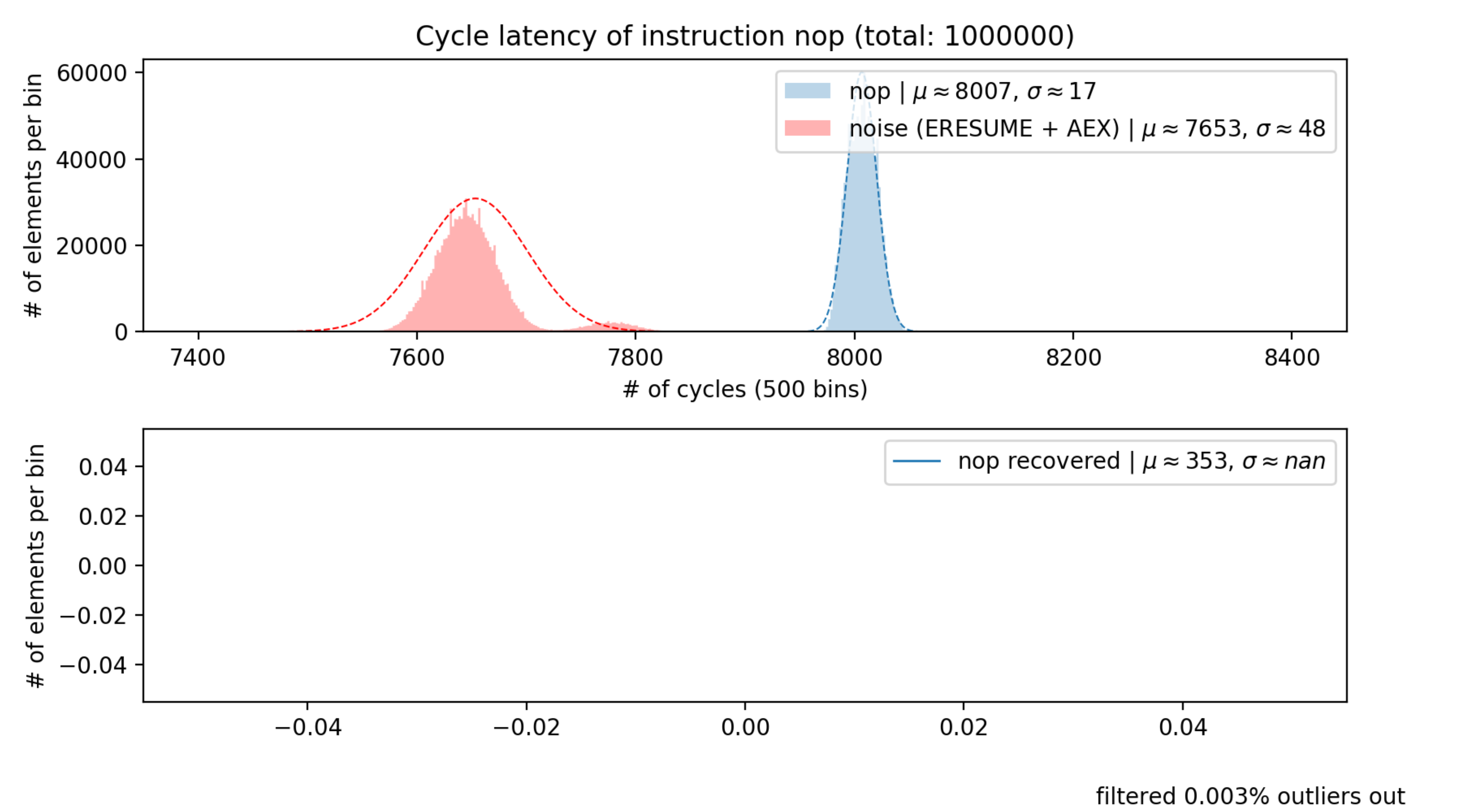
C10: Two Noise Sources



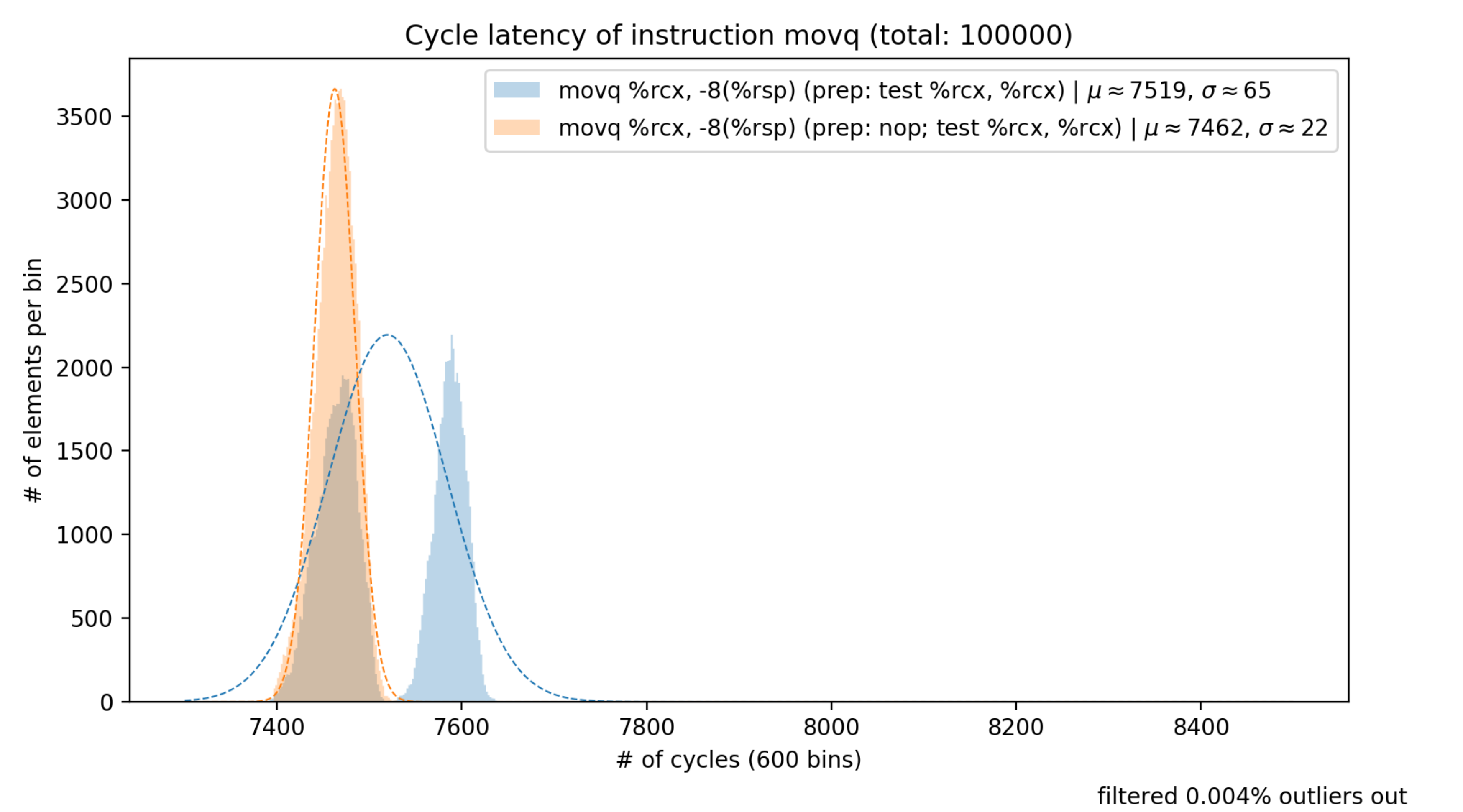
C10: Two Noise Sources



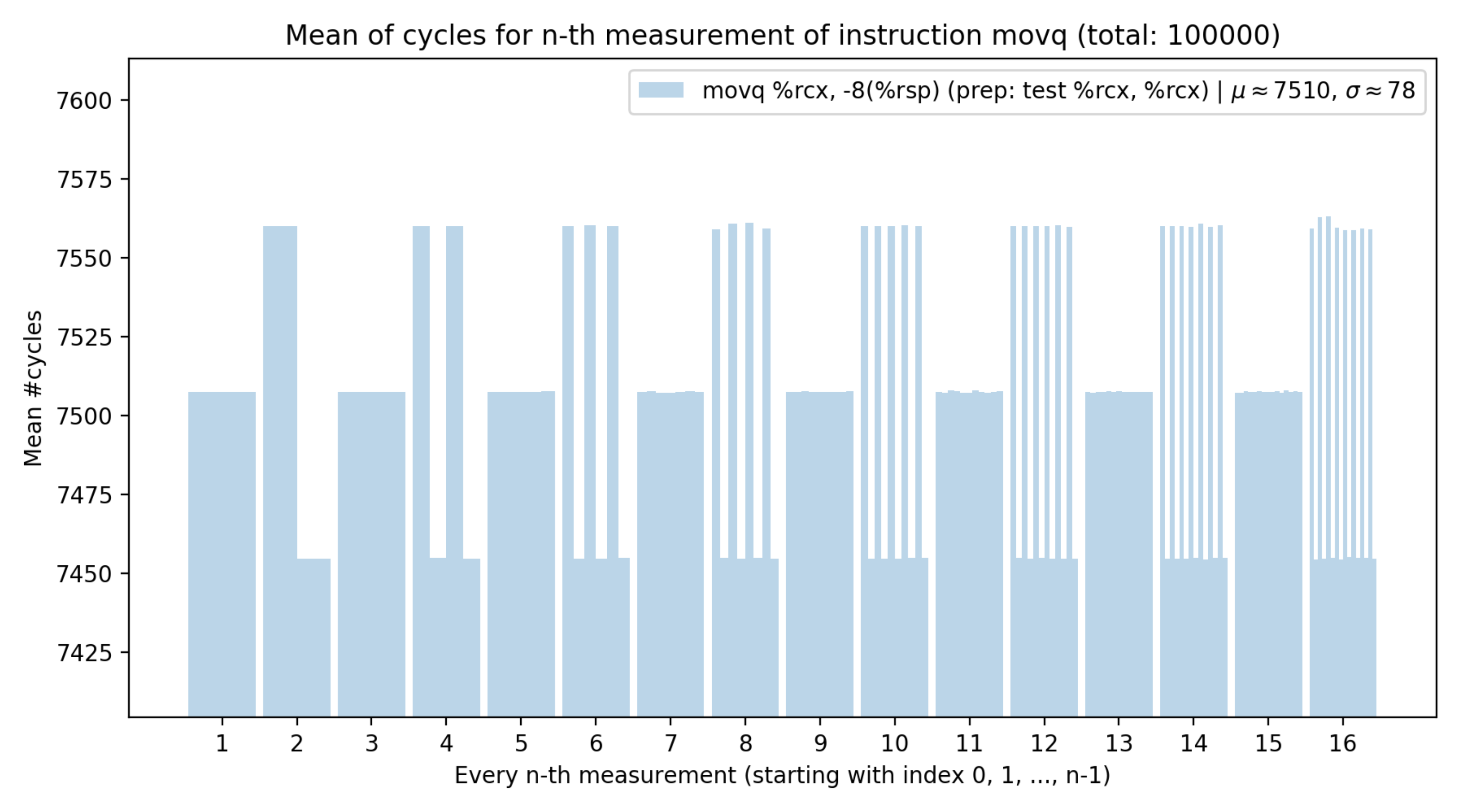
C10: Two Noise Sources



Applications – Double Peaks

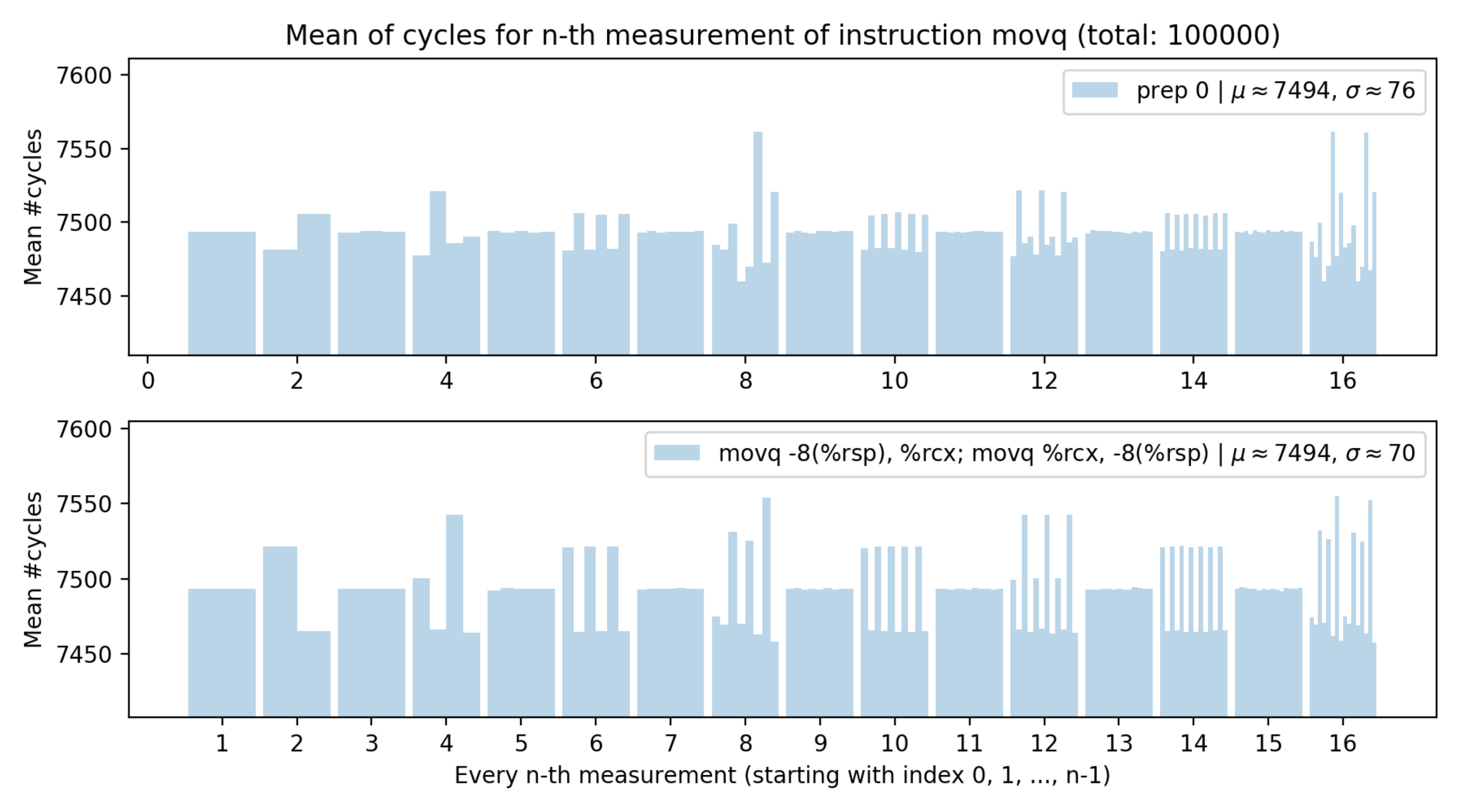


Applications – Double Peaks

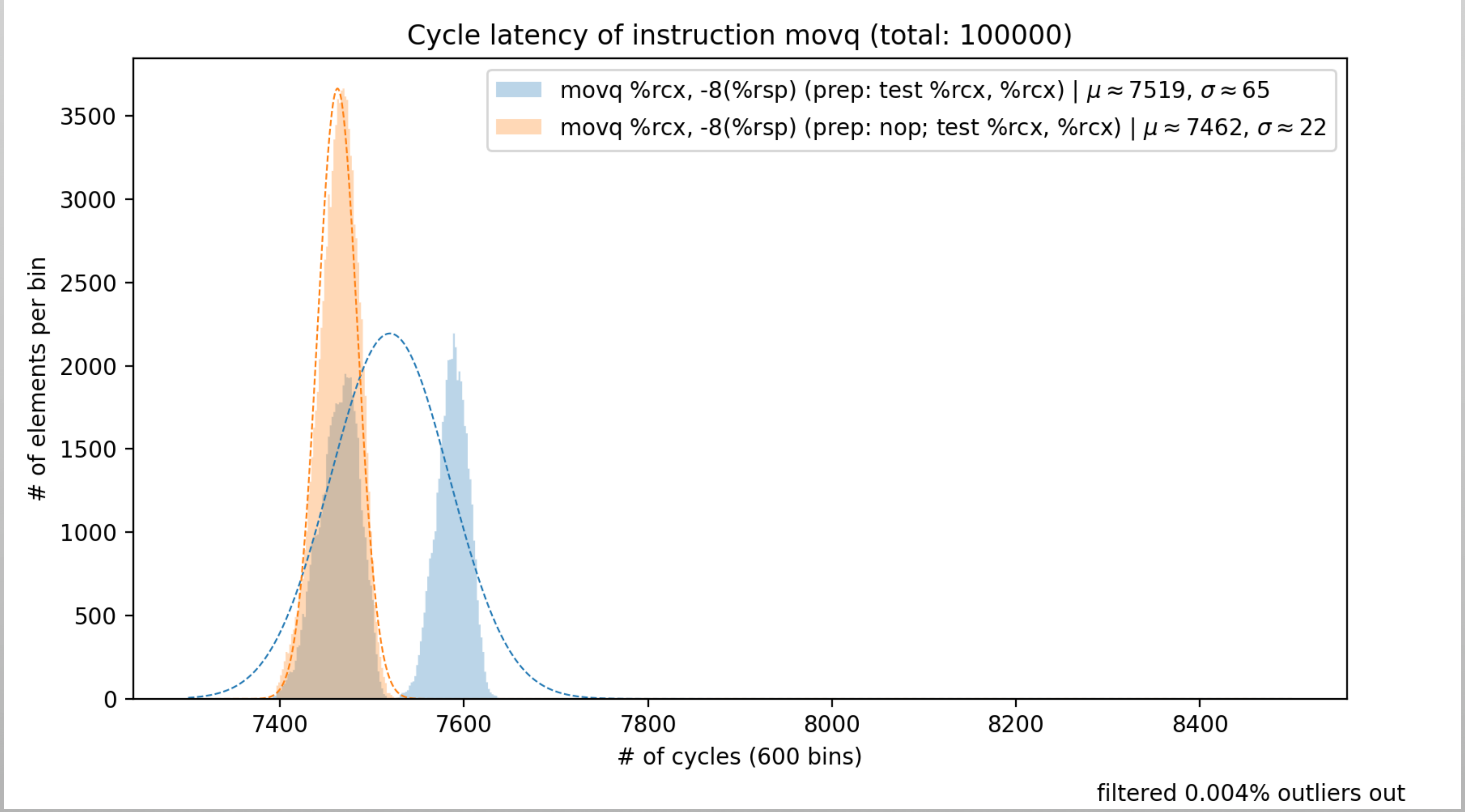


ETH zürich

Applications – Double Peaks

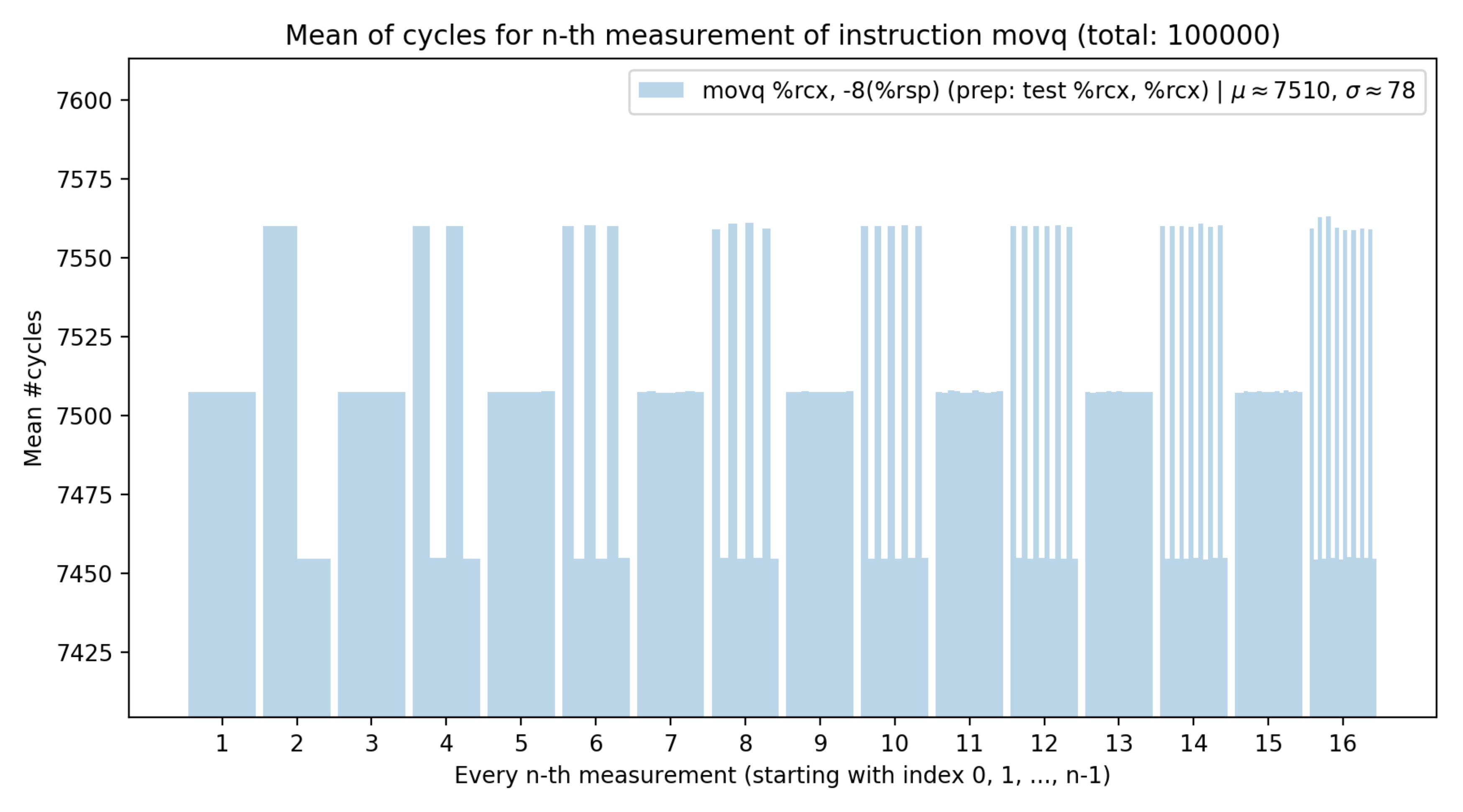


Applications – Double Peaks

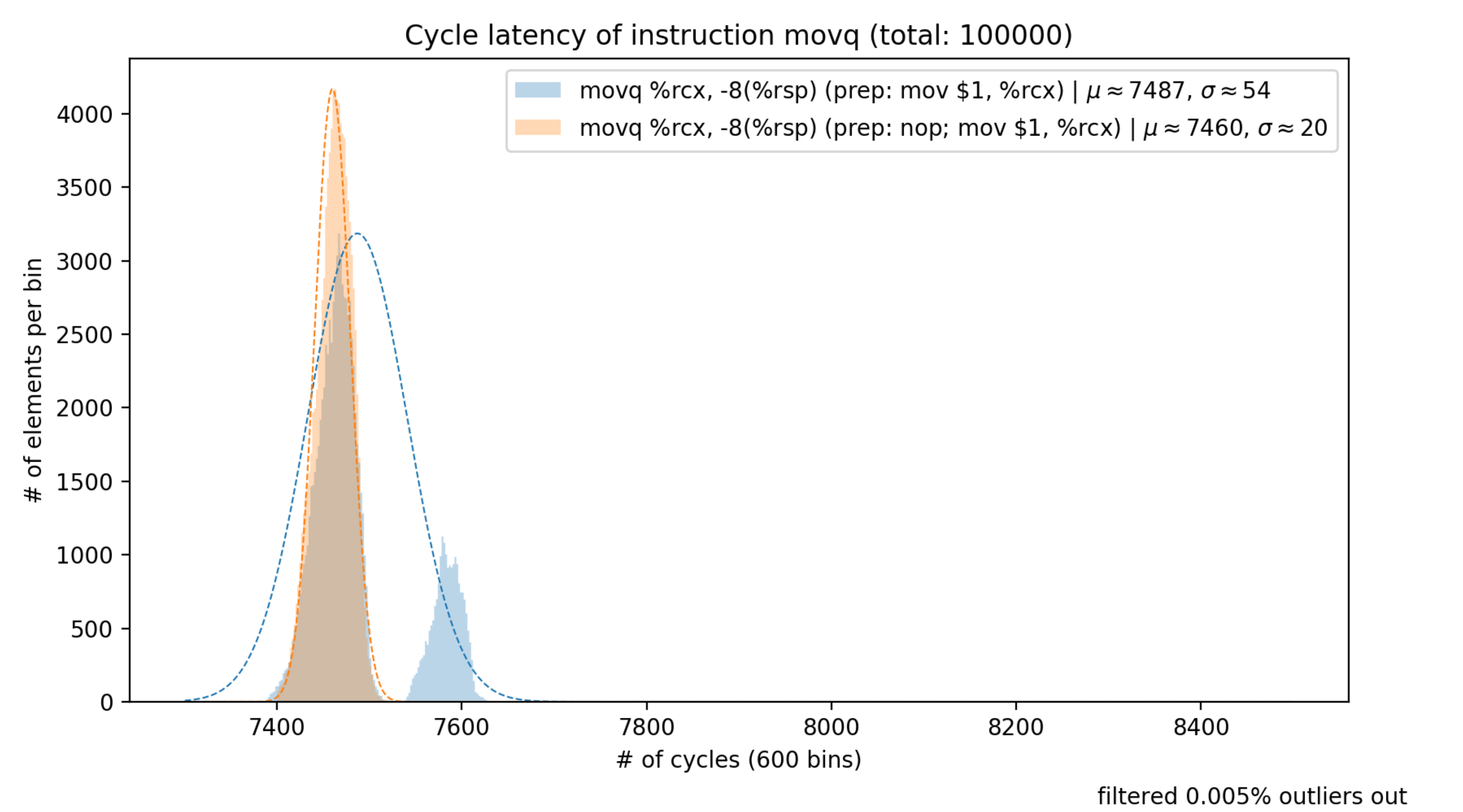


ETHzürich

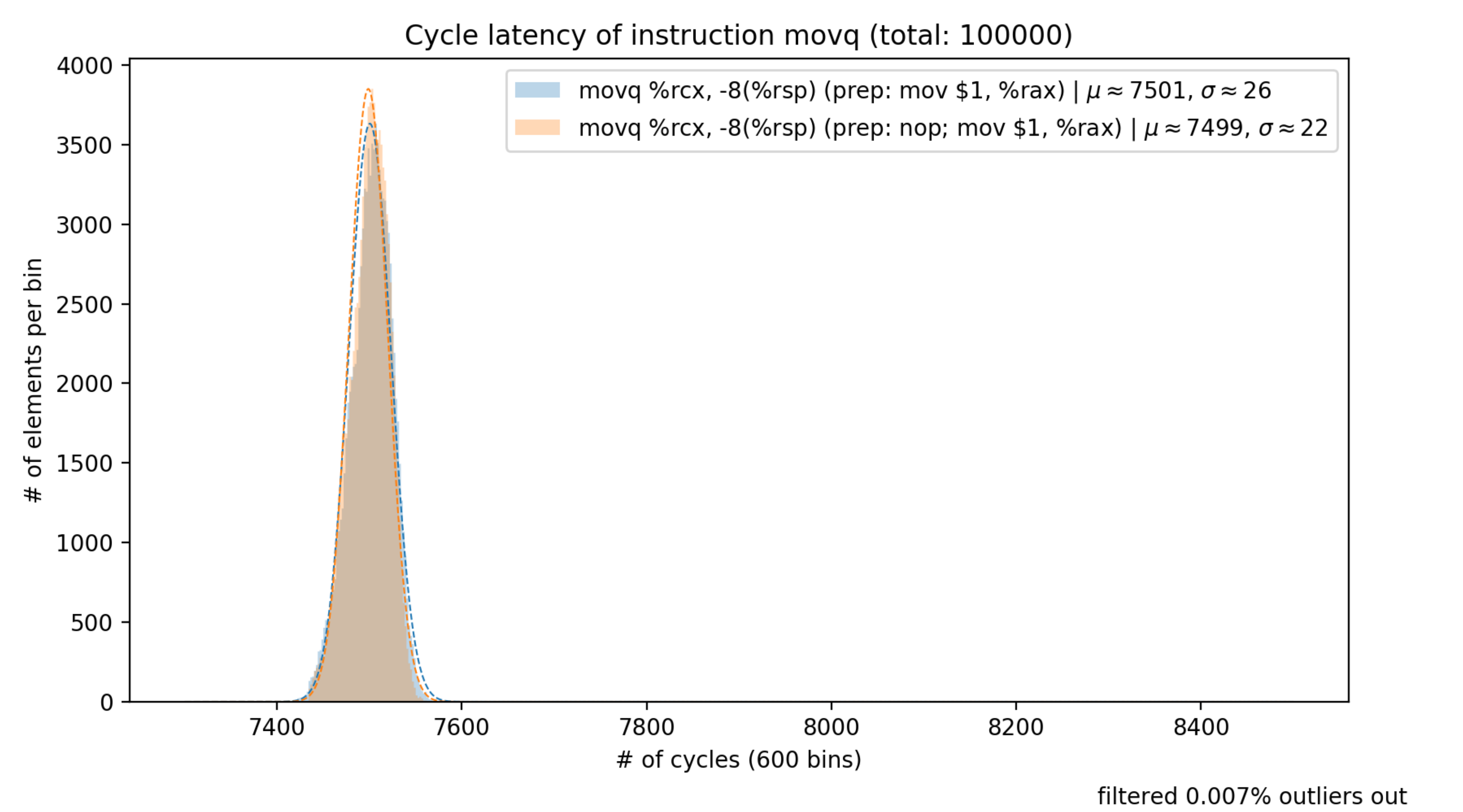
Applications – Double Peaks



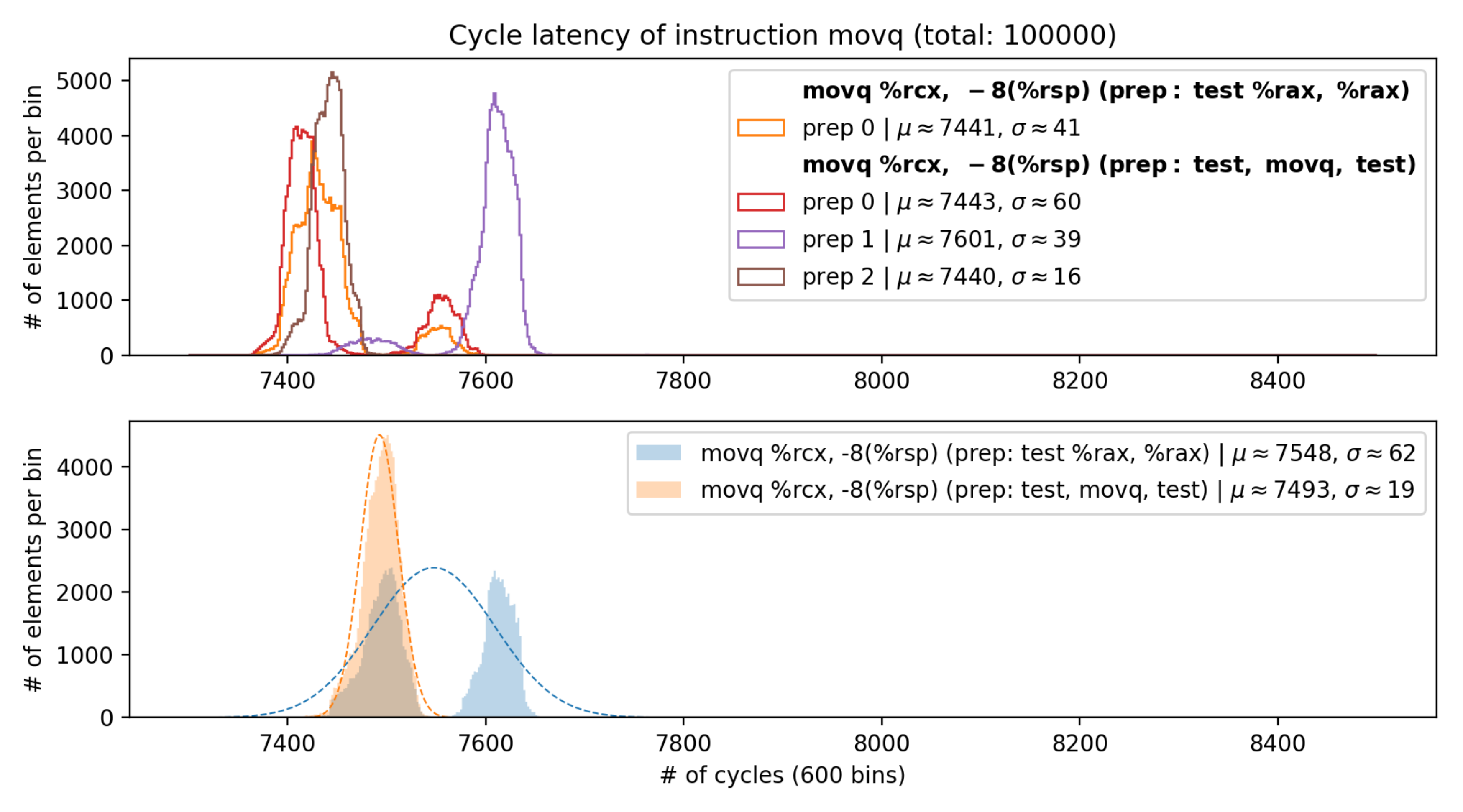
Double Peaks – Operand Dependent



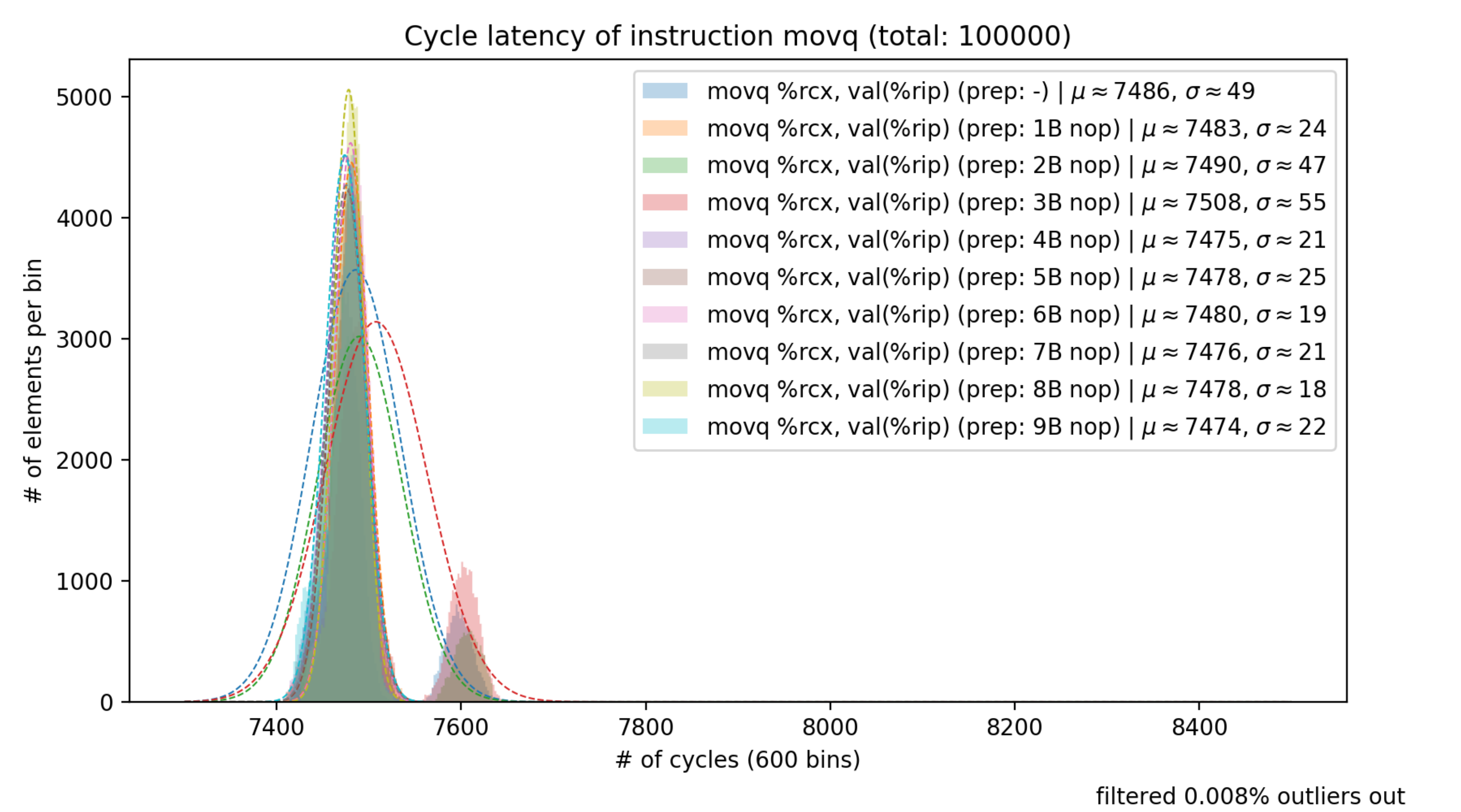
Applications – Operand Dependent



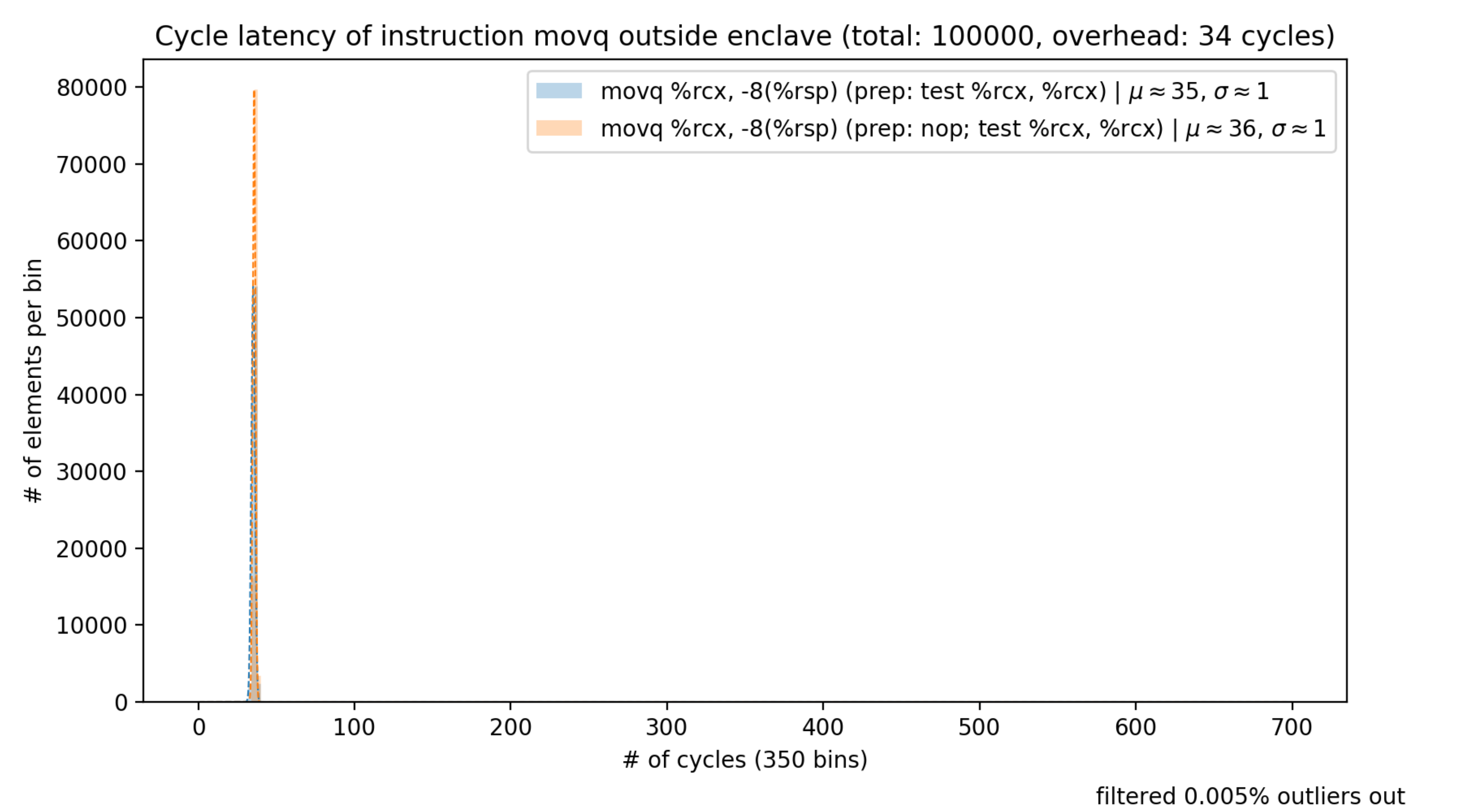
Hidden Double Peaks



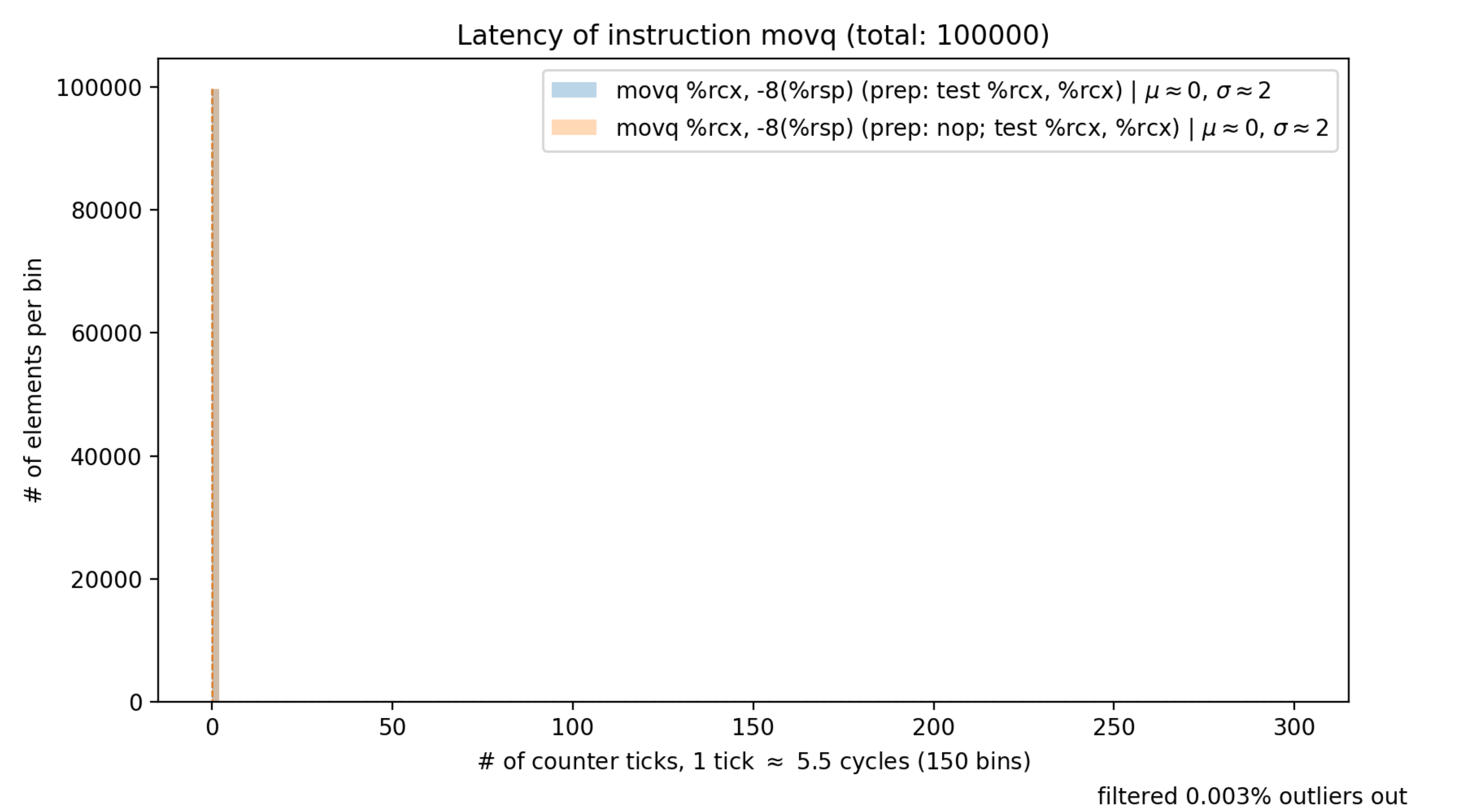
Double Peaks Prepare NOPs



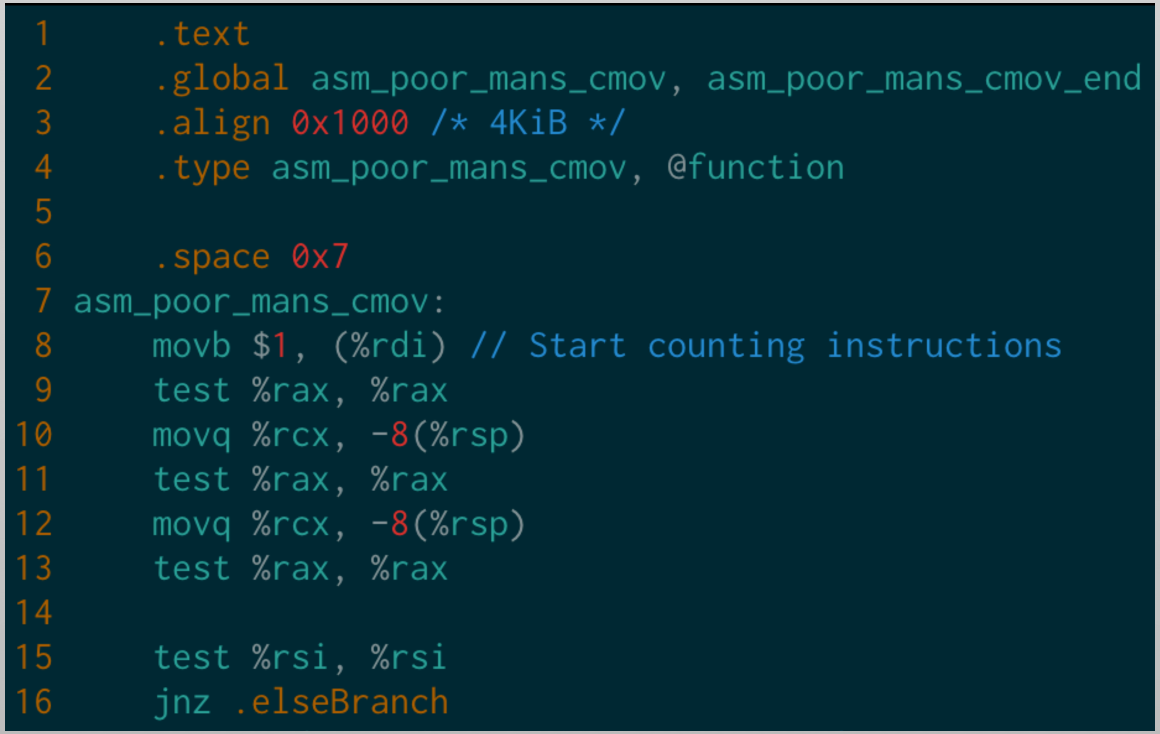
No Double Peaks – Outside



No Double Peaks – Counter Method



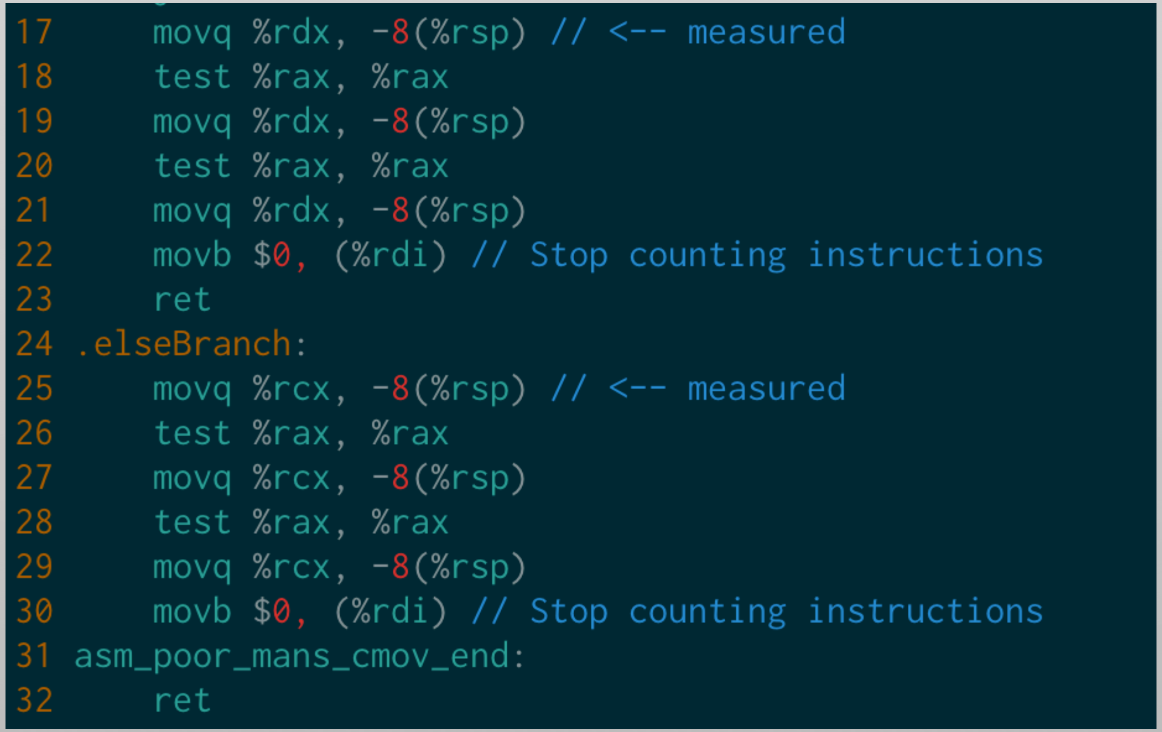
Poor Man’s CMOV – Short Code



```
    .text
    .global asm_poor_mans_cmov, asm_poor_mans_cmov_end
    .align 0x1000 /* 4KiB */
    .type asm_poor_mans_cmov, @function

    .space 0x7
asm_poor_mans_cmov:
    movb $1, (%rdi) // Start counting instructions
    test %rax, %rax
    movq %rcx, -8(%rsp)
    test %rax, %rax
    movq %rcx, -8(%rsp)
    test %rax, %rax

    test %rsi, %rsi
    jnz .elseBranch
```



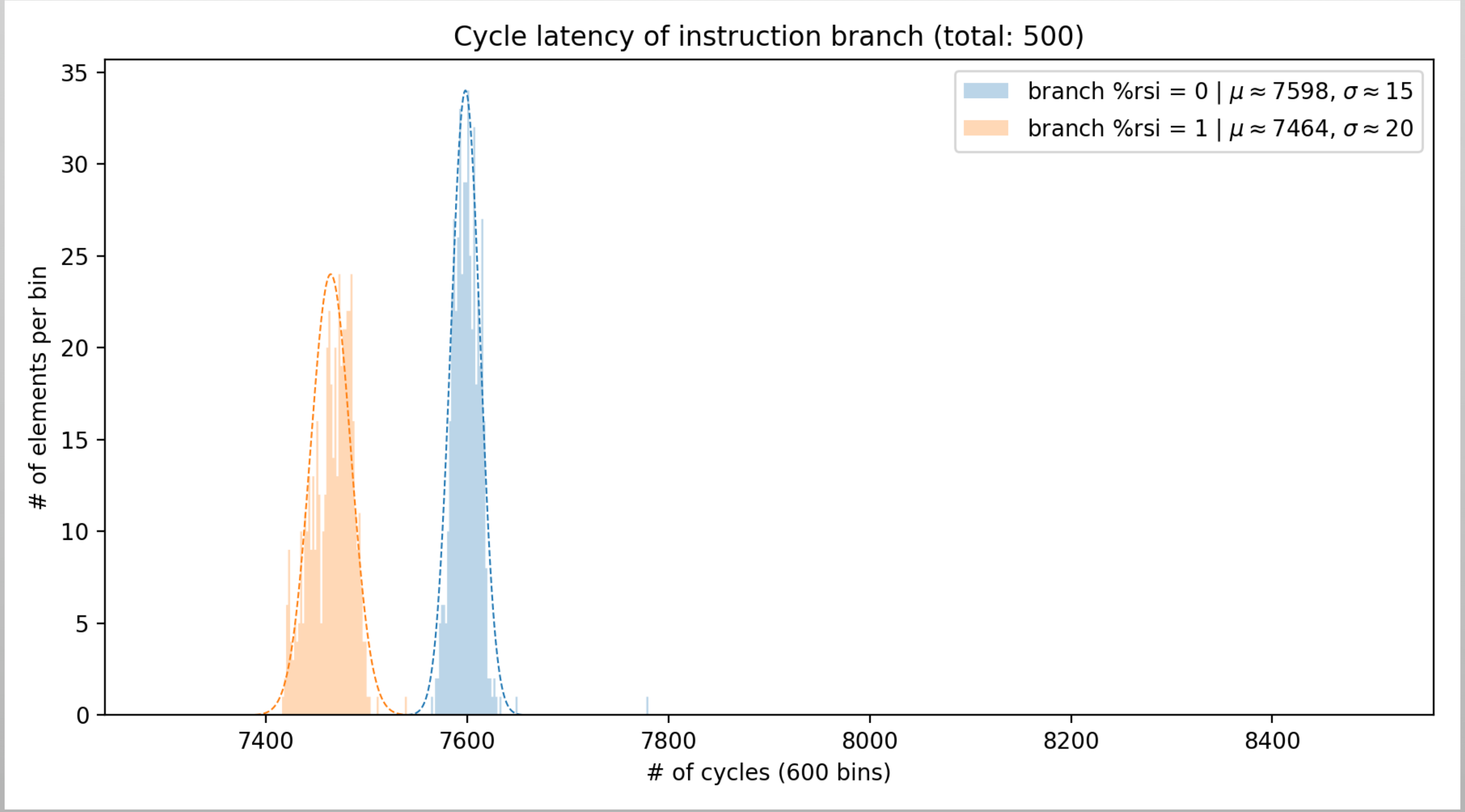
```
    movq %rdx, -8(%rsp) // <-- measured
    test %rax, %rax
    movq %rdx, -8(%rsp)
    test %rax, %rax
    movq %rdx, -8(%rsp)
    movb $0, (%rdi) // Stop counting instructions
    ret
.elseBranch:
    movq %rcx, -8(%rsp) // <-- measured
    test %rax, %rax
    movq %rcx, -8(%rsp)
    test %rax, %rax
    movq %rcx, -8(%rsp)
    movb $0, (%rdi) // Stop counting instructions
asm_poor_mans_cmov_end:
    ret
```

Poor Man's CMOV – Precise results

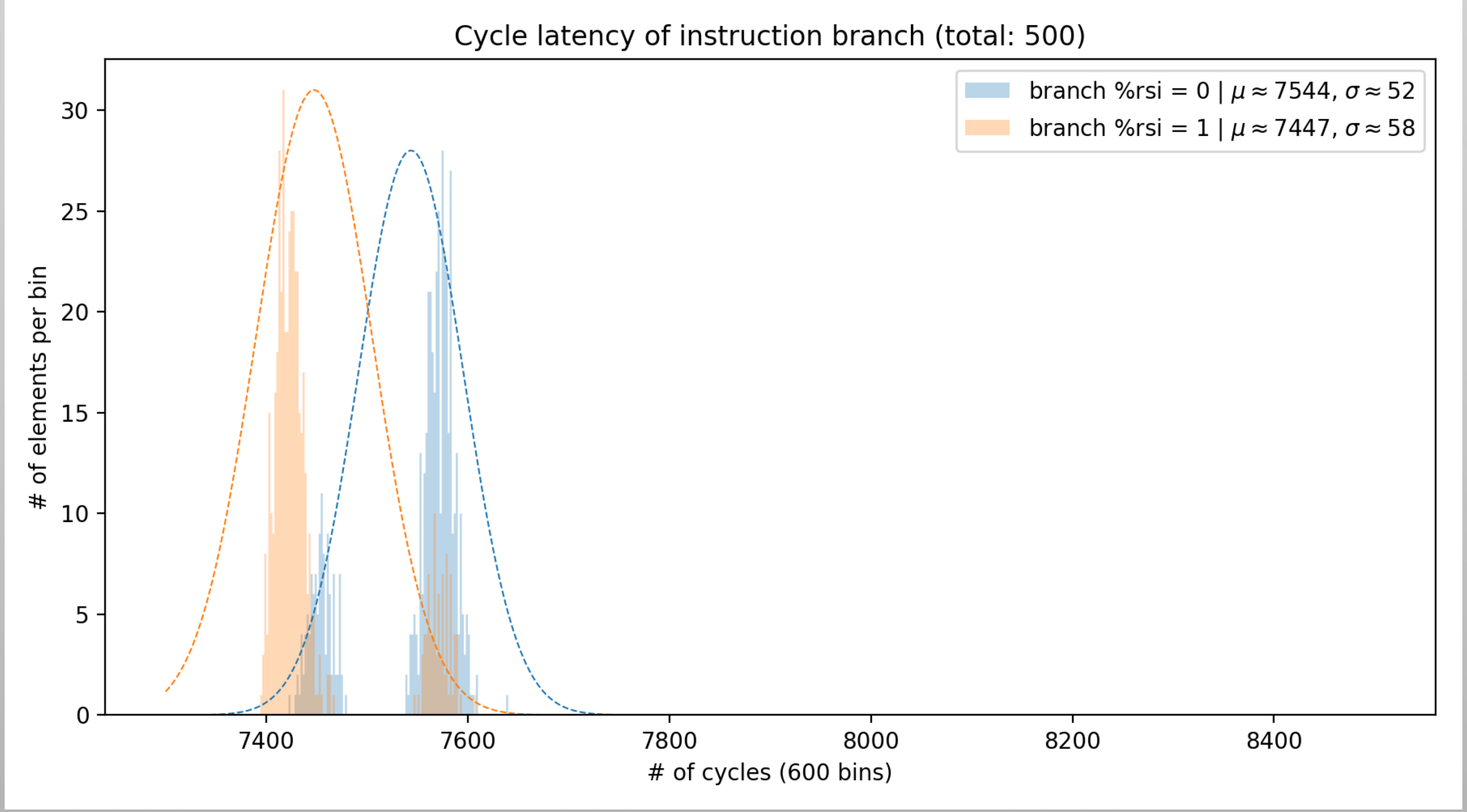
- Results

	Type (Alignment)	Correctly Captured	Mean	Standard Deviation
Our tool	Long (0x07)	100%	97.5	7.1
	Short (0x27)	100%	54.6	8.1
Nemesis	Long (0x07)	95.6%	55.6	13.3
	Short (0x27)	27.7%	59.5	3.7

Poor Man's CMOV – Long Version



Poor Man’s CMOV – 1 Cache Line


Cycle latency of instruction branch (total: 500)
branch %rsi = 0 | $\mu \approx 7544$, $\sigma \approx 52$
branch %rsi = 1 | $\mu \approx 7447$, $\sigma \approx 58$
# of elements per bin
0
5
10
15
20
25
30
7400
7600
7800
8000
8200
8400
# of cycles (600 bins)