BooTRIST: Detecting and Isolating Mercurial Cores at the Booting Stage

Yihao Luo[1,*], Yunjie Deng[1,*], Jingquan Ge[3],
Zhenyu Ning[2], Fengwei Zhang[1,✉]

[1] Southern University of Science and Technology,

[2] Hunan University, [3] Nanyang Technological University

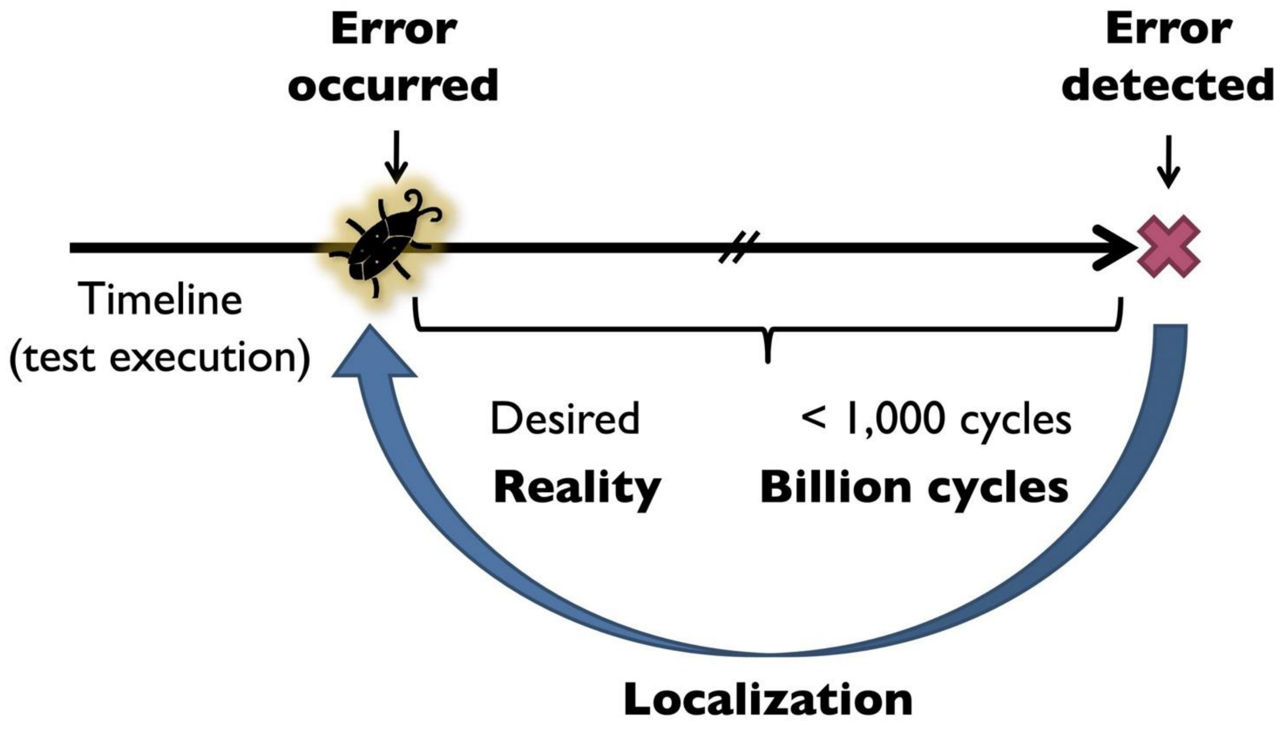
SUSTech Southern University of Science and Technology

COMPASS Lab COMPuter And System Security Lab

Mercurial Core

Mercurial core:

- The CPU core with failures
- Explicit faults causing program crashes or exceptions
- Data errors introduced without explicit error reports



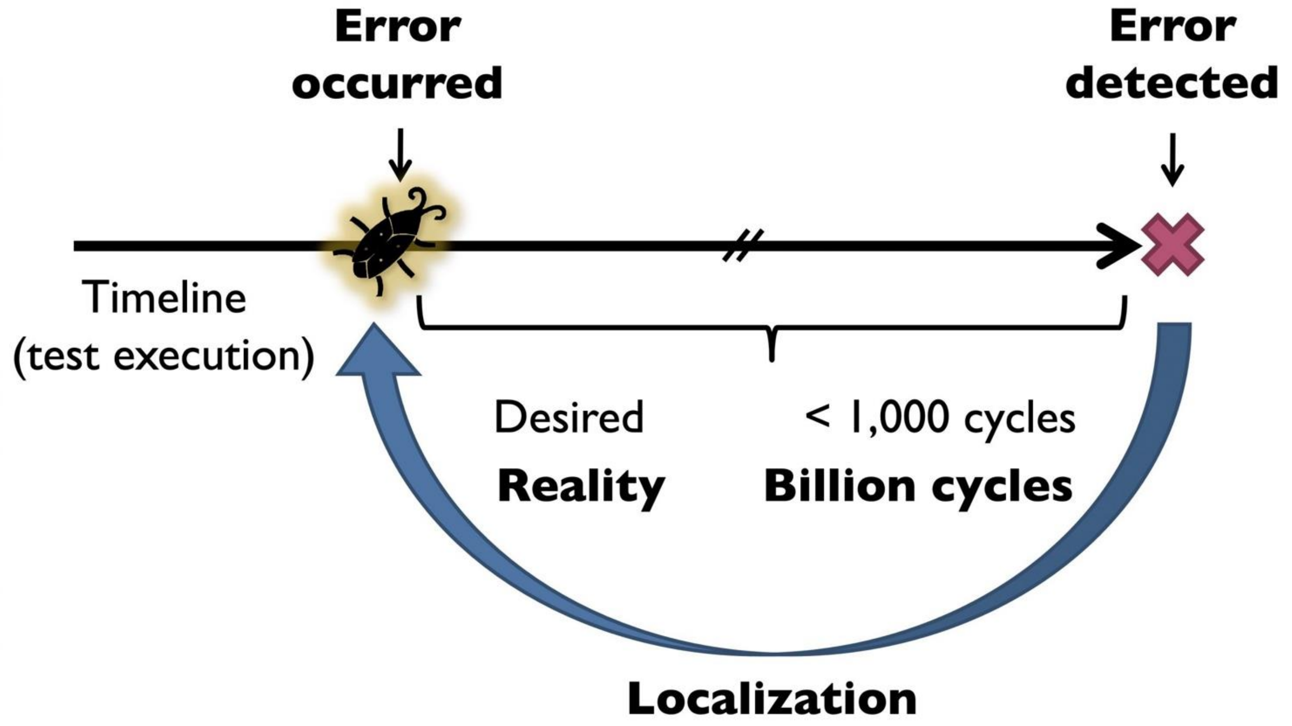
Silent data corruptions[1]

1. https://www.sigarch.org/wp-content/uploads/2022/08/error-decection-latencies-scaled.jpg

Mercurial Core

Google and Meta have discovered that some server CPUs can repeatedly produce random wrong results without any error logs

- Low occurrence frequency
- Difficult to debug
- High prevalence
- Common in complex computation instructions

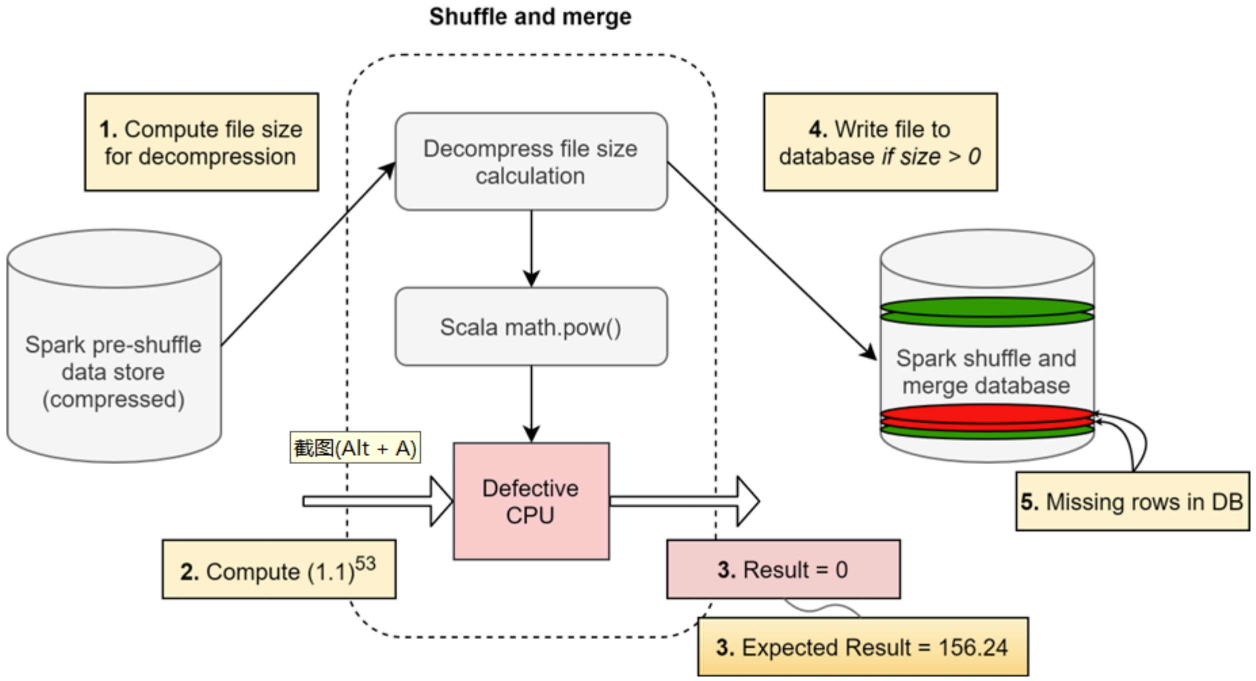


Silent data corruptions[1]

1. https://www.sigarch.org/wp-content/uploads/2022/08/error-decection-latencies-scaled.jpg

What harm might it cause?

- Google[1]:
  - Garbage collection module failure
  - Kernel state corruption
- Meta[2]:
  - File metadata calculation error lead to file loss
- Ali Cloud[3]:
  - Wrong verification calculation results caused the requested data to be frequently mistaken for damage.

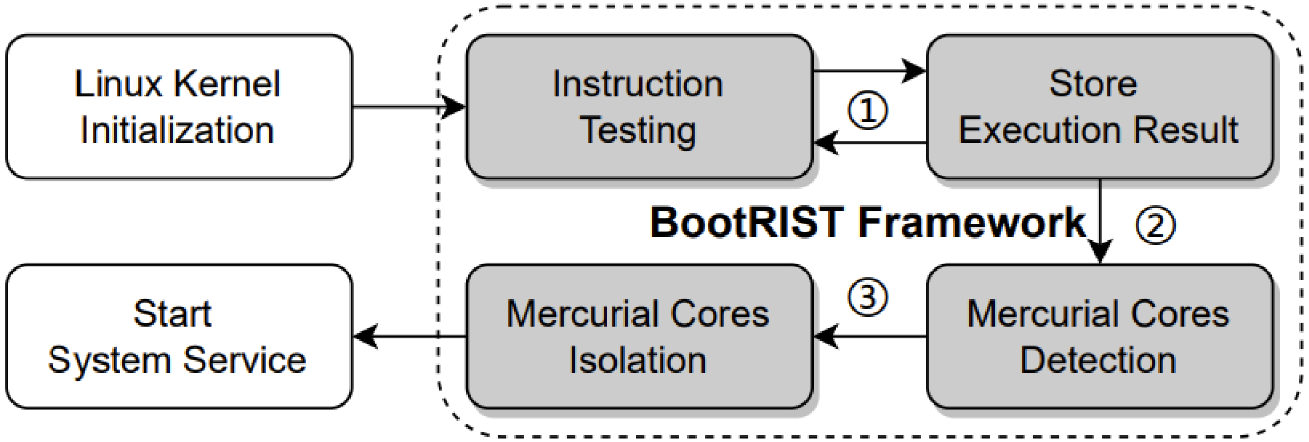


Meta Mercurial Core Example[2]

BootRIST

Objective: Efficient, low-overhead and generic tool for mercurial core detection and isolation.

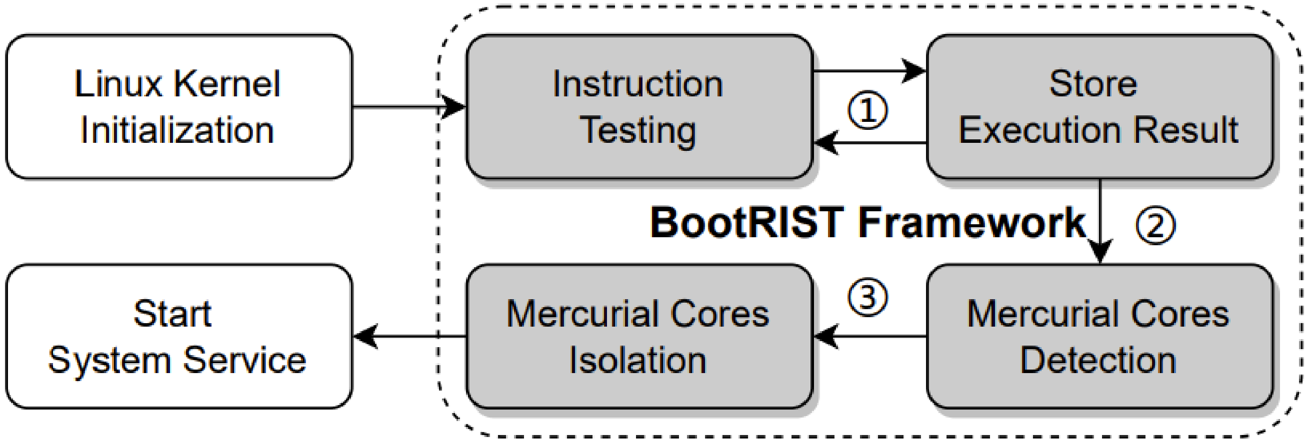
- Software-based solution
- High efficiency
- High portability



Design of BootRIST

Objective: Efficient, low-overhead and generic tool for mercurial core detection and isolation

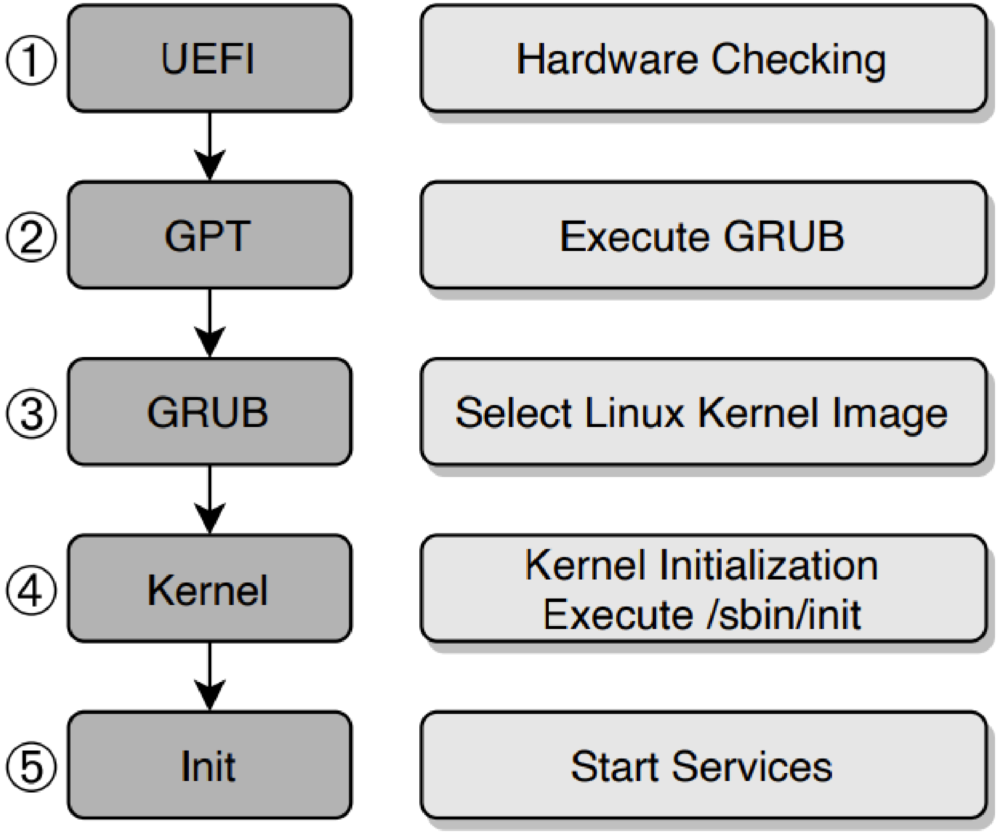
- Instruction consistency testing
- Mercurial cores detection based on voting algorithm
- Mercurial cores isolation based on cpu hotplug



Design of BootRIST

BootRIST executes during the Linux kernel boot stage

- Between step 4 and step 5
- Avoid affecting high-level applications



Linux Kernel Booting Process

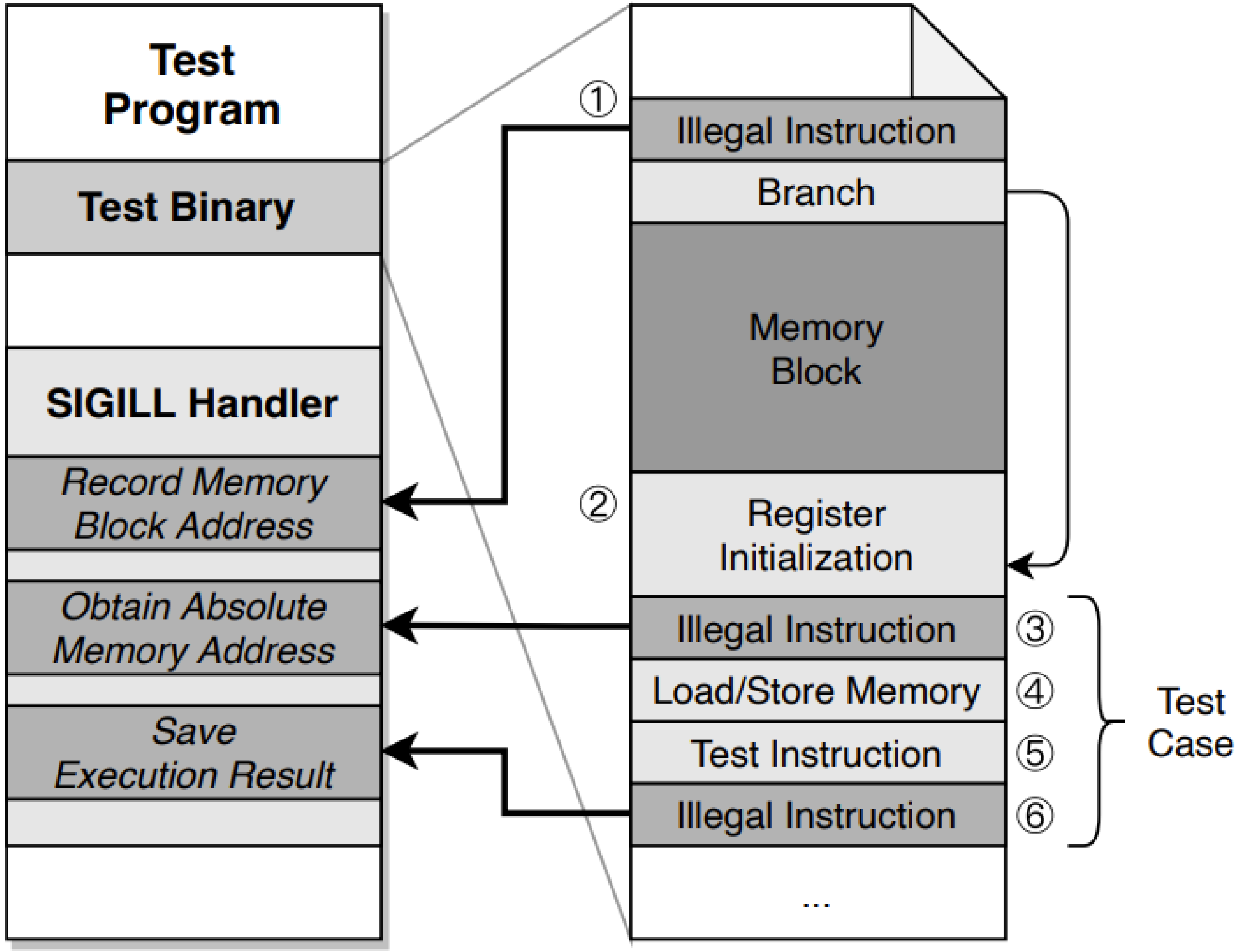
Instruction Testing

Test program:

- Load the risu[1] test binary
- Handle dedicated SIGILL events in test binary

risu test binary:

- Memory Blocks
  - Maintain the memory data for memory load/store operations
- Registers initialization
- Test Instructions
- Illegal Instruction
  - Assist with memory access operations and execution result saving opreations



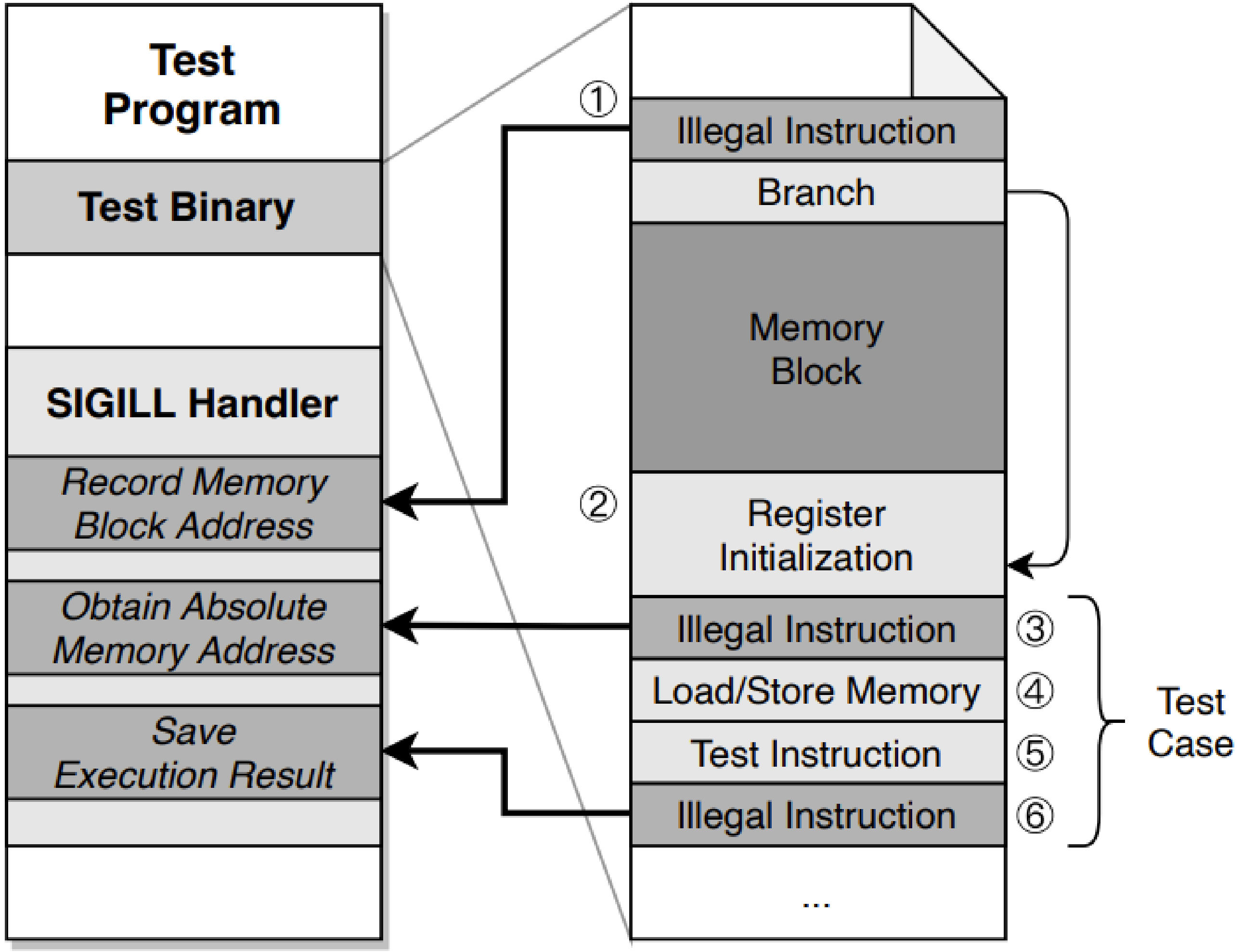
Instruction testing process

1. https://github.com/Xaltotun/risu

Instruction Testing

Workflow:

- Initialize registers
- Execute test cases
  - Load/Store data into memory block
  - Run test instruction
  - Save execution result
- Stop after completing all test cases



Instruction testing process

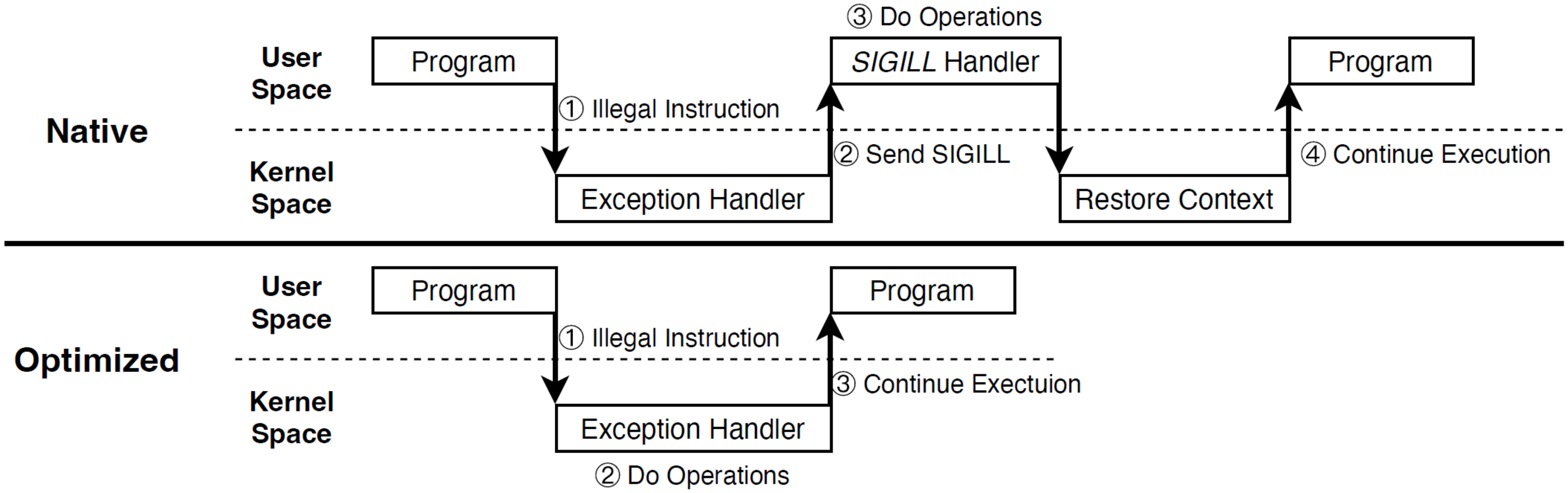
Optimize SIGILL Handler

Frequent illegal instruction causes large overhead

- Multiple context switches when entering and exiting the SIGILL handler

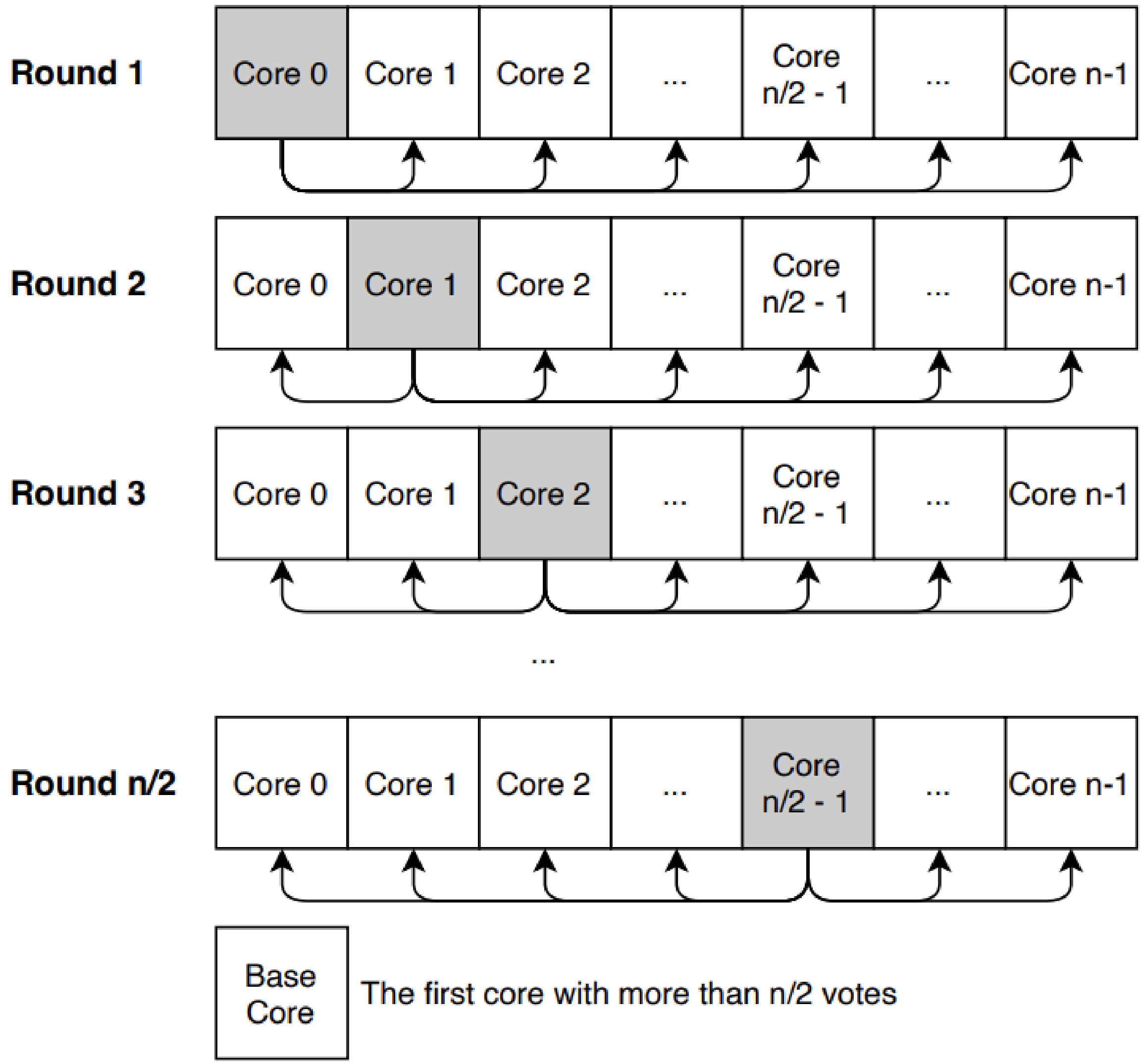
Optimization

- Reduce the number of context switch to improve performance



Mercurial Core Detection

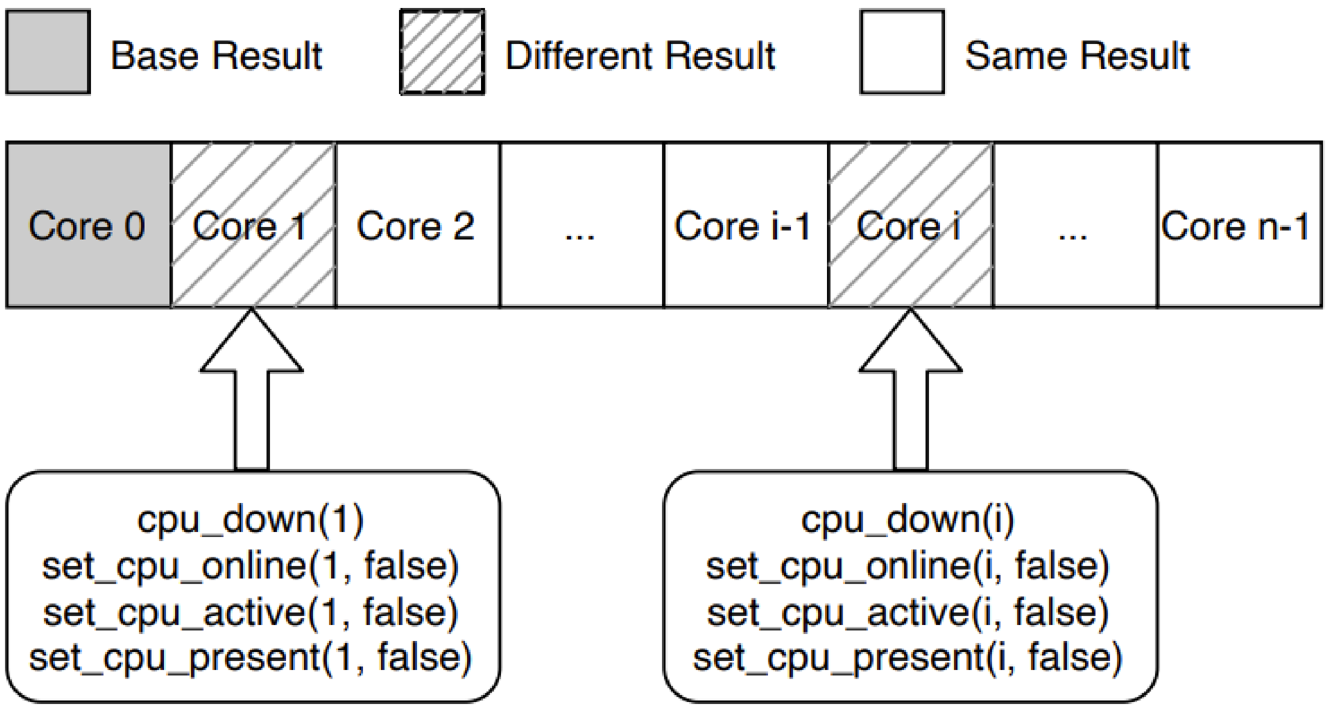
- Based on voting algorithm
- Up to n/2 round, algorithm complexity is controllable



Mercurial Core Isolation

Based on Linux CPU hotplug mechanism, clear the **online**, **active**, and **present** state for mercurial cores.

- Present: The core is plugged
- Online: The core is available for scheduling
- Active: The core is available for task migration



Experiments Setup

Platform: Huawei Taishan 2280 server (Kunpeng 920 processor, ARMv8 architecture, 96 cores)

Operating System: EulerOS 20.03, Linux kernel 4.19.90

Software Configuration: Each core is allocated 8192 bytes for memory results and 1024 bytes for register results.

BootRIST Performance

Performance improvements after optimizing SIGILL handler:

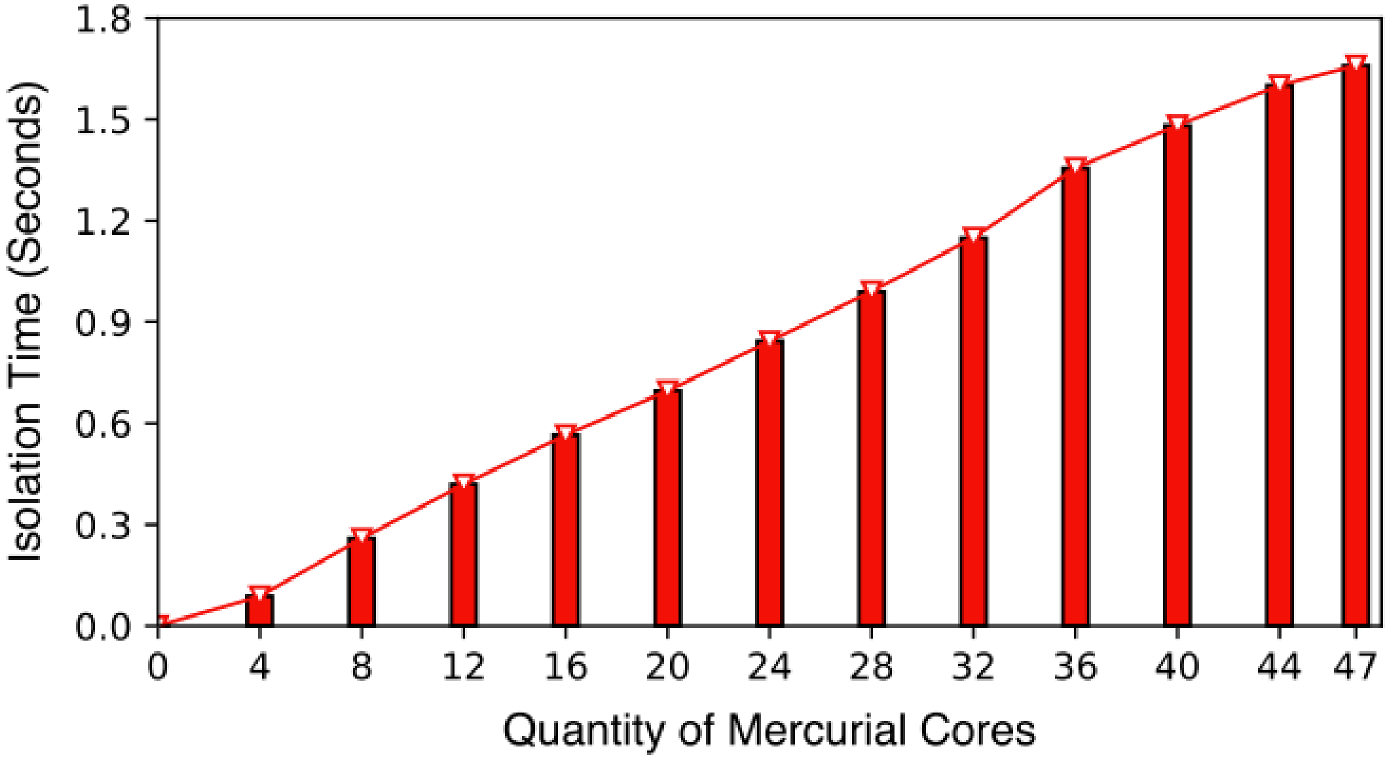
- 20% improvement compared to risu

The average server startup time is approximately 162.87 seconds

Instructions	BootRIST		risu [18]	
	Test Time	Percentage	Test Time	Percentage
10	0.01	0.01%	0.01	0.01%
100	0.02	0.01%	0.03	0.02%
1000	0.18	0.11%	0.23	0.14%
10000	1.86	1.13%	2.31	1.40%
100000	18.78	10.34%	23.10	12.42%
500000	93.75	36.53%	115.36	41.46%

Performance of Instruction Testing

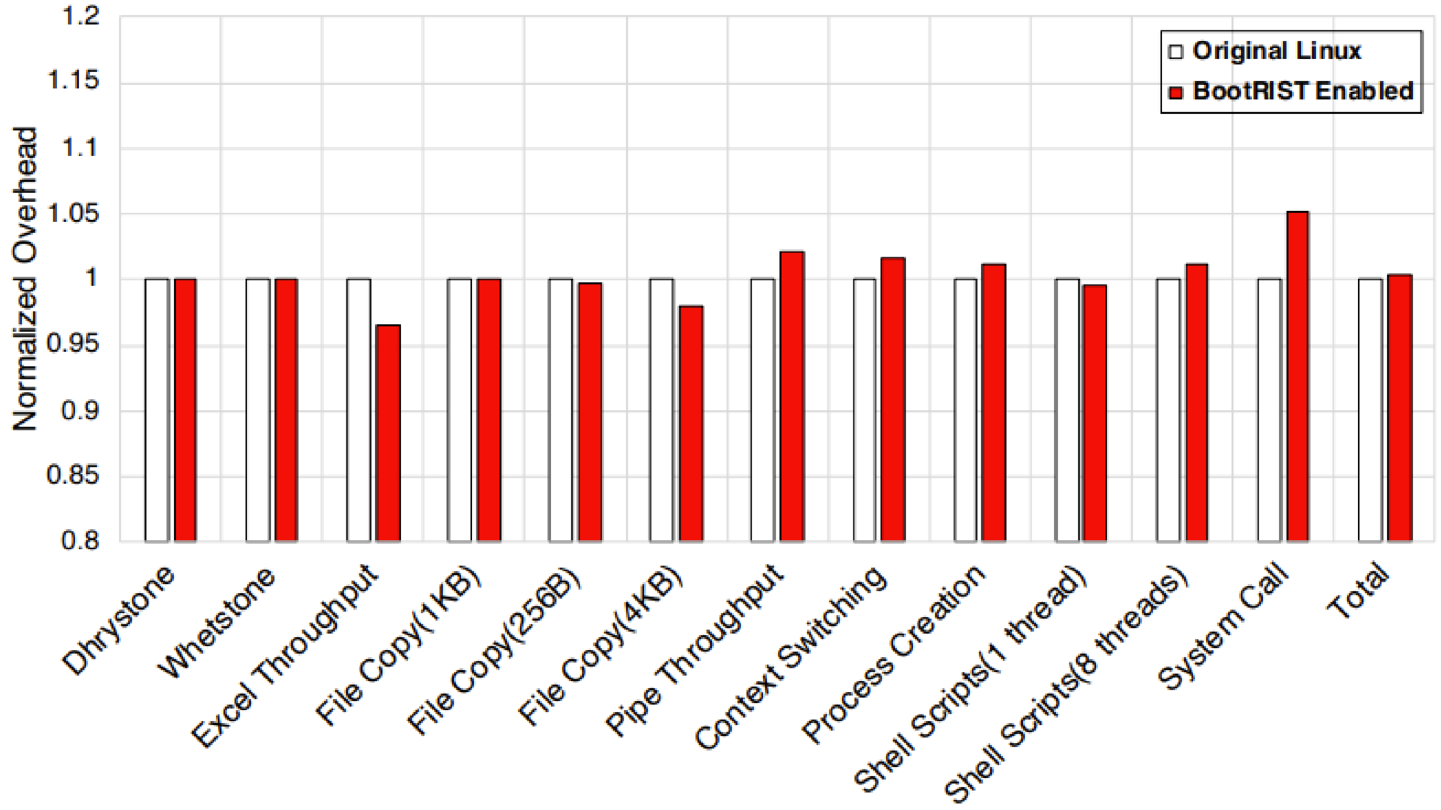
Due to voting algorithm, the number of simulated mercurial cores cannot exceed half of the total number



System Overall Overhead

Unixbench: A benchmark for evaluating kernel mode performance on Unix-like systems

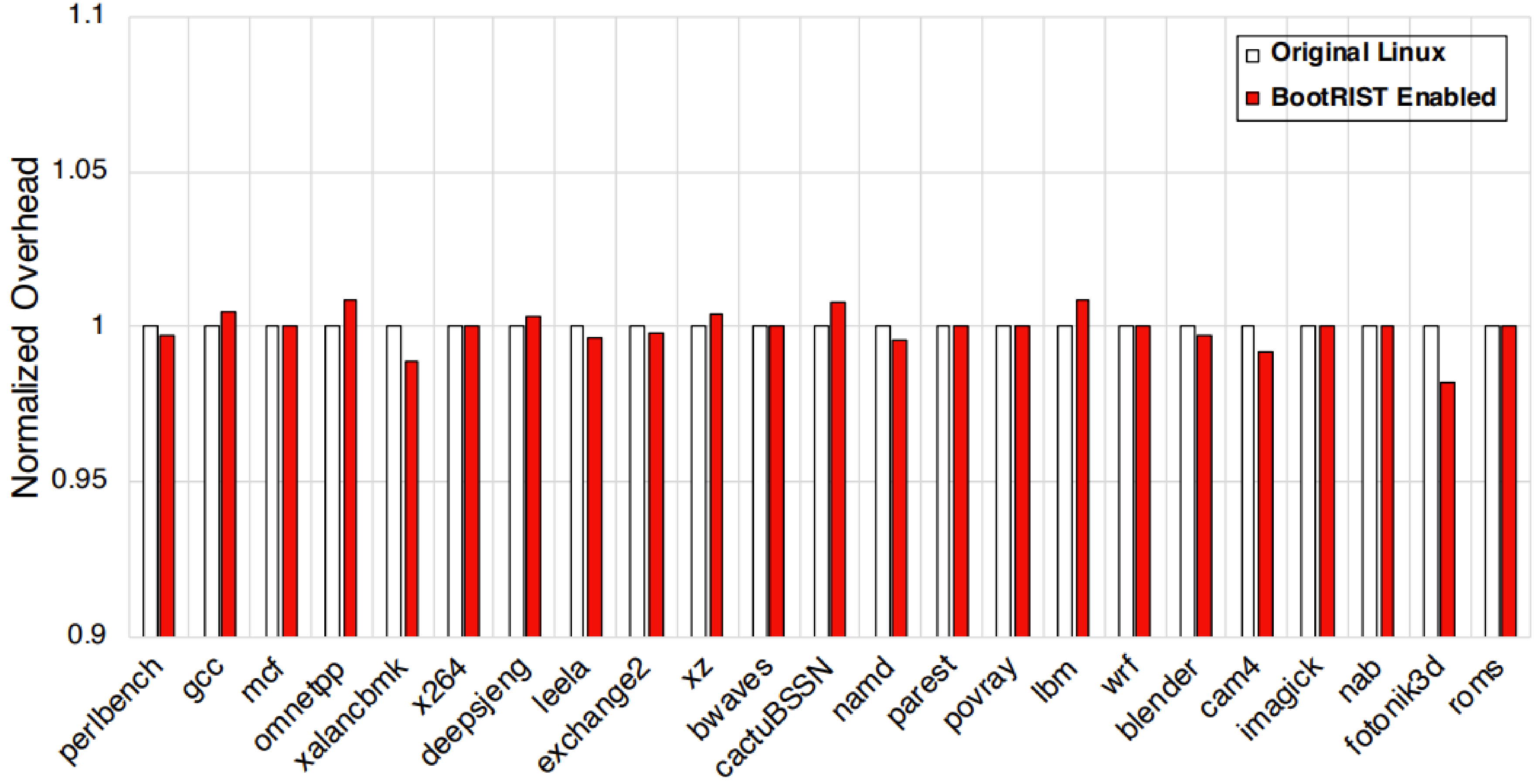
- Maximum overhead: ~5.17%
- Average overhead: ~0.419%



System Overall Overhead

SPEC_CPU 2017: compute-intensive test cases for evaluating user-mode performance

- Maximum overhead: ~0.89%



Conclusion

BootRIST: a software based solution to detect and isolate mercurial cores at the booting stage

- Efficient instruction testing framework
- Low system overhead
- High portability

Thanks for listening!

Q & A

COMPASS Lab: zhangfw@sustech.edu.cn

References

1. **Hochschild, Peter H., et al. "Cores that don't count."** ***Proceedings of the Workshop on Hot Topics in Operating Systems*****. 2021.**
2. **Dixit, Harish Dattatraya, et al. "Detecting silent data corruptions in the wild."** ***arXiv preprint arXiv:2203.08989*** **(2022).**
3. **Wang, Shaobu, et al. "Understanding Silent Data Corruptions in a Large Production CPU Population."** ***Proceedings of the 29th Symposium on Operating Systems Principles*****. 2023.**